Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Periodicity, Surprisal, Attention:

Skip Conditions for
Recurrent Neural Networks

Dissertation

submitted in partial fulfilment of
the requirements for the degree of
Doctor rerum naturalium
(Dr. rer. nat.)

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

Tayfun Alpay

Hamburg, 2021

Day of submission:
January 04, 2021

Day of oral defense:
March 24, 2021

Abstract — *English*

Recurrent neural networks (RNNs) are a type of artificial neural network that can be used to learn sequential data. Leading to significant breakthroughs in deep learning, they have been established as powerful tools for sequence learning, particularly in the domain of natural language processing. These developments have particularly been made possible by the scalability of modern machine learning algorithms, allowing researchers and practitioners to utilize both larger amounts of data and significantly larger models. However, as the computational requirements for most recent state-of-the-art systems keep outgrowing hardware advancements, designing more efficient models becomes an increasingly important research goal. At the same time, some of the core limitations of neural networks remain present even when scaled up, suggesting the need for more fundamental advancements in model design, before achieving further significant breakthroughs.

In this thesis, we focus on the typical limitation of recurrent neural networks to process sequences step by step, linearly through time. As not every timestep holds valuable information, this can be considered both computationally inefficient and a barrier towards developing more natural and intelligent data processing. Our efforts focus on contributing to the alternative paradigm of conditional computation, a relatively new research area. Conditional computation provides an alternative to traditional sequence processing models which process inputs and update at every timestep. Instead, the objective is to define or learn useful constraints which allow the network to *skip* certain parts of the input sequence which are not taken into account during training. This type of attentive processing can lead to more computationally efficient and more interpretable models.

As our first skip condition, we investigate periodic activation with the Clockwork RNN, comparing it to more traditional recurrent models. We develop this original model further by integrating memory gates. Running an ablation study on the architectural components, we take an in-depth look at the strengths and weaknesses of a strong inductive bias for skipping. Based on these initial studies, we develop a novel skipping mechanism based on surprisal, called Surprisal-based Activation (SBA) which we investigate in a variety of gated architectures for language modeling. After adapting SBA in hierarchical models for question answering, we investigate its compatibility to hierarchical attention mechanisms, particularly pointer attention. We use this as a basis to develop our third approach, Attention-based Skipping (ABS).

Our results demonstrate that modular designs and carefully constructed inductive biases can form effective constraints that facilitate skipping in recurrent models. Furthermore, we show that models of salience provide an alternative to more traditional compression methods, which allows us to minimize, and sometimes even avoid, trade-offs between model performance and compression in the form of skipping.

Zusammenfassung — *Deutsch*

Rekurrente neuronale Netze (RNN) bilden eine Form von künstlichen neuronalen Netzen, die sequenzielle Daten lernen können. Durch ihren signifikanten Beitrag zu Durchbrüchen im Bereich des Deep Learning haben sie sich als mächtiges Werkzeug für sequenzielles Lernen, insbesondere im Bereich der Verarbeitung natürlicher Sprache, etabiliert. Diese Entwicklungen wurden insbesondere durch die Skalierbarkeit moderner Machine Learning Algorithmen vorangetrieben, was folglich Forschern und Anwendern ermöglicht hat sowohl größere Datenmengen, als auch signifikant größere Modelle einzusetzen. Dennoch haben gestiegene Anforderungen an maschinelle Rechenleistungen über die Entwicklungen von Hardwarekapazitäten hinaus dazu geführt, dass die Entwicklung von effizienteren Modellen ein immer wichtigeres Forschungsziel wird. Gleichzeitig bleiben jedoch einige der grundlegenden Bebeschränkungen von neuronalen Netzen trotz Skalierbarkeit bestehen, was eine Notwendigkeit für weitere fundamentale Fortschritte im Design von Modellen erforderlich macht, bevor weitere deutliche Fortschritte ermöglicht werden.

In dieser Arbeit fokussieren wir uns auf die typische Limitierung von rekurrenten neuronalen Netzen, welche darin besteht Sequenzen schrittweise, einem linearen Zeitverlauf folgend, abzuarbeiten. Da nicht jeder Zeitschritt zwangsläufig wertvolle Informationen enthält, kann dieser Umstand zum einen als rechnerisch ineffizient angesehen werden und zum anderen auch als eine Barriere gegen die Entwicklung einer natürlicheren und intelligenteren Datenverarbeitung verstanden werden. Unsere Anstrengungen fokussieren sich daher auf einen Beitrag zum alternativen Paradigma der "bedingten Berechnung" (engl.: conditional computation), bei dem es sich um ein relativ neues Forschungsgebiet handelt. Bedingte Berechnungsmodelle liefern eine Alternative zu traditionellen Sequenzverarbeitungsmodellen, bei denen die Eingaben in jedem Zeitschritt verarbeitet und aktualisiert werden. Dieser alternative Ansatz verfolgt das Ziel, nützliche Einschränkungen zu definieren oder zu lernen, welche dem Netz erlauben gewisse Teile der Eingabesequenz zu *überspringen* (engl.: skipping), die beim Training nicht berücksichtigt werden. Diese Art von Aufmerksamkeit gesteuerter Verarbeitung kann zu rechnerisch effizienteren und besser interpretierbaren Modellen führen.

Als unsere erste Bedingung zum Überspringen von Verarbeitungsschritten untersuchen wir periodische Aktivierung mit Hilfe des sog. "Clockwork RNNs" im Vergleich zu traditionelleren rekurrenten Modellen. Wir entwickeln das Originalmodell weiter, indem wir Memory Gates integrieren. Dank einer methodischen Dekonstrution ihrer jeweiligen Komponenten reflektieren wir über die Stärken und Schwächen eines induktiven Bias für Skipping. Davon ausgehend entwickeln wir einen neuen Skipping-Mechanismus, der auf sog. "Surprisal" basiert, die Surprisal-based Activation (SBA), welche wir auf unterschiedlichen Architekturen mit Memory Gates für Sprachmodellierung untersuchen. Nachdem wir SBA in hierarchische Modelle für das automatisierte Beantworten von Fragen adaptieren, untersuchen wir seine Kompatibilität mit hierarchischen Aufmerksamkeitsmechanismen, insbesondere der sog. “Pointer-Aufmerksamkeit”. Wir nutzen dies als Entwicklungsgrundlage für unseren dritten Ansatz, dem Attention-based Skipping (ABS).

Unsere Ergebnisse zeigen, dass modulare Entwürfe und umsichtig konstruierte, induktive Biase effektive Beschränkungen formen können, die das Skipping in rekurrenten Modellen erleichtern. Darüber hinaus zeigen wir, dass Modelle für Salienz eine Alternative zu herkömmlichen Kompressionsmethoden bieten, was uns erlaubt, einen Kompromiss zwischen der Leistungsfähigkeit eines Modells und seiner Komprimierung durch Skipping zu minimieren - und in manchen Fällen sogar zu verhindern.

Contents

Appendices

List of Figures

List of Tables

Nomenclature

Abbreviations

ABS	Attention-based Skipping
ACT	Adaptive Computation Time
CBT	Childrens Book Test
CBT-CN	Childrens Book Test (Common Nouns)
CBT-NE	Childrens Book Test (Named Entities)
CC	Conditional Computation
CNN	Convolutional Neural Network
CWLSTM	Clockwork LSTM
CWRNN	Clockwork RNN
ERG	Embedded Reber Grammar
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HAN	Hierarchical Attention Network
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
PCA	Principle Component Analysis
PTB	Penn Treebank
ptr	Pointer (Attention)
QA	Question Answering
RAN	Recurrent Attention Network
RNN	Recurrent Neural Network
RPN	Recurrent Plausibility Network
SBA	Surprisal-based Activation
SCRN	Structurally Constrained Recurrent Network
SENS	Surprisal-based Ensembling
SGD	Stochastic Gradient Descent
SGRU	Surprisal-based GRU
SRN	Simple Recurrent Network

STE	Straight-Through Estimator
ZO	Zoneout

Notations

$\mathbf{v}$	Vector
$\mathbf{M}$	Matrix
v	Variable
$(x_1, ..., x_n)$	Sequence
$\mathbf{h}$	Hidden layer
$\lvert\mathbf{h}\rvert$	Hidden layer size
$\mathbf{h}_t$	Hidden state vector at timestep t
h_t	Hidden unit at timestep t
$\mathbf{W}_{xy}$	Weight matrix with connections from layer x to y

Symbols and Operations

$[\mathbf{a}; \mathbf{b}]$	Concatenation of vectors **a** and **b**
$\alpha >_\epsilon n$	α is larger than n by a small difference $(\alpha - n) = \epsilon$, $\epsilon > 0$
$\approx$	Approximation
$\eta_{.90}$	90-th Percentile
$[\![\mathcal{L}]\!]$	Iverson Bracket: $[\![\mathcal{L}]\!] = \begin{cases} 1 & \text{if logical proposition } \mathcal{L} \text{ is true} \\ 0 & \text{otherwise} \end{cases}$
$\odot$	Dot product
$\otimes$	Hadamard product (element-wise multiplication)
$X \sim \mathcal{A}$	X has the probability distribution of $\mathcal{A}$

Measurements and Metrics

ACC	Accuracy
MAD	Mean Absolute Deviation
PPL	Perplexity
SD	Standard Deviation
UR	Update Rate

Part I

Background and Motivation

Chapter 1

Introduction

"I learned to write fiction the way I learned to read fiction - by skipping the parts that bored me."

—Jonathan Lethem, 2013

1.1 Motivation

Imagine sitting in your car and driving from Hamburg to Berlin. At any given point in time during this journey, you do not know precisely how much time you have left until you reach your destination. However, every dozen miles or so, the road presents you with a sign that tells you the remaining distance until Berlin. "How much further?", your kids ask from the back-seat. Naturally, you answer with an estimate based on the last sign that you saw about half an hour earlier. Now, imagine being asked this same question *once per second for the entirety of your journey*. To make matters worse, you are forced to answer *every time*. In addition, before giving your answer, you have to memorize and recall not just the last road sign that you've seen but *all* signs that you have encountered since your departure, even including parking signs, stop signs, speed limits etc. While most would agree that any person capable of this feat has remarkable memory (and endurance), they would also question why anybody would choose such a complicated process over only providing updates after encountering a relevant road sign.

Unfortunately, this analogy captures the way Recurrent Neural Networks (RNNs) process information, making continual predictions based on recalling past context. The traditional sequence processing paradigm builds a chain of past events, weighting important subsequences more than those that provide less information. Nevertheless, there is no concept of "forgetting" or "jumping" back to an earlier memory without unrolling and traversing this linked chain of events. This drawback has given rise to recurrent models with event-based update mechanisms. Particularly important is the ability of *skipping* redundant or uninformative parts of a sequence. Recently, a novel framework, named *conditional computation* has been proposed to address this research problem. In this framework, states are not forced to update at every timestep but do this conditionally, based on other constraints. In this thesis, we will investigate different types of constraints that allow us to determine when a model should be updating and when it should be skipping.

1.2 Research Objectives

The main idea of conditional computation is to only allow a neural network to update its internal state if a certain boolean condition is met. This condition builds the basis for a network's decision on when to perform a state update and when to skip it. Consequently, designing neural networks with meaningful skipping capabilities requires certain assumptions in the form of constraints that formalize what separates important from unimportant input.

The main objective of this thesis is to investigate promising candidates for skip conditions and constraints to facilitate skipping. As such, we pursue the overarching research goal of *modeling skipping conditions in recurrent neural networks to learn more efficient representations*. We define "efficient representations" as compressed representations that exclude redundancies as much as possible without leading to a decline in model accuracy and approach this problem with the main objective to develop RNN models that are capable of filtering out redundancy in their representations by *updating only when necessary*.

Overall, we approach this research goal in this thesis by addressing the following research questions:

1. Which constraints facilitate effective skipping?
2. How can we minimize, or even avoid, any trade-offs between model performance and skipping?

In the first question, we consider different design principles for the modeling of our architectures. Overall, we can categorize these into two types of inductive biases: those reflected in constraints that we impose on skip conditions, and those that are structural in the context of the network design. For structural constraints, we primarily focus on incorporating modularity and hierarchy as design concepts to drive units towards diverse self-organization during training and specialized representations. For constraints on the skip conditions, we start with more restrictive approaches (such as periodic activation) and gradually loosen these assumptions with each approach, ending with salience-based skipping and keeping only structural biases.

In the second question, we address the problem that large amounts of skipping can lead to critical loss of information in the encoding, causing drops in model accuracy. While compression rates can be adjusted to be lossless, most predictive conditional computation approaches lead to a decreased accuracy with increased skipping, forming a natural trade-off between accuracy and activation sparsity. As part of this thesis, we will explore methodologies to minimize any such trade-offs by way of not only focusing on minimizing state updates but also improving representation qualities to maintain or even increase baseline accuracies.

Our approach to answering these questions comes with the additional side-objectives to model networks which provide representations that are calculated efficiently, allow near-lossless skipping, and a level of interpretability comparable to related approaches. The three main constraints that we investigate for skip conditions are:

1. Periodicity
2. Surprisal
3. Attention

For periodicity, we build on an existing architecture, the Clockwork RNN, and extend it, using gating capabilities from the Long Short-Term Memory (LSTM) model, to the Clockwork LSTM. For surprisal, we introduce a novel architecture with fewer constraints than periodic activation. Finally, we utilize recurrent attention for a salience-based model for skipping. Overall, we use modularity as a design principle for networks with periodic and surprisal-based activations and focus on hierarchical abstraction in the final part of the thesis, where we explore skipping based on surprisal and attention.

To evaluate our models, we mainly focus on applications in the domain of Natural Language Processing (NLP), particularly language modeling and question answering. Nevertheless, we include several additional tasks to validate that our findings generalize to other domains.

1.3 Contributions

This thesis contributes detailed studies and evaluations of different design methodologies for conditional computation in RNNs. Overall, we propose two completely novel methodologies for skipping in recurrent networks, namely surprisal-based activation and attention-based skipping. As part of this effort, we introduce two new classes of recurrent models and three progressions on existing work, each consisting of multiple variants, leading to a total of 24 novel models that we introduce and study.

Summarizing our main contributions in this thesis, we:

- Evaluate periodic activation language modeling.

- Evaluate the role of unidirectional connections in the Clockwork RNN.
- Introduce the Clockwork LSTM (4 variants).
- Introduce Surprisal-based Activation (SBA) on the SRN, LSTM (7 variants), and Gated Recurrent Unit (GRU; 2 variants).
- Provide the first extensive evaluation of the role of the LSTM gates for zoneout (4 variants).
- Extend the Hierarchical Attention Network (HAN) for question answering tasks (2 variants).
- Explore the dynamics between attention and skipping (2 variants).
- Propose a methodology capable of Attention-based Skipping (ABS) that can be used with traditional models without modifying the underlying update process (2 variants).

In contrast to other conditional computation approaches, our models do not lose accuracy despite high skipping rates, in many cases even improving on the baselines. With ABS, we also present an approach requiring no fine-tuning of regularization penalties, reward functions, or hyperparameters, providing compatibility with existing recurrent layers without additional modifications. Throughout this thesis, we perform extensive ablation studies for each introduced model. This methodology helps us to gain a more profound knowledge of the causes and effects of incorporating specific designs in the presented models, contributing towards more fundamental knowledge of how to effectively model recurrent neural networks.

1.4 Thesis Outline

This thesis is structured into five different parts. While parts I and V build the motivation and closing thoughts, respectively, the main research is presented in the chapters of parts II-IV. Each part builds on the results from the previous but introduces a different methodology for designing recurrent networks capable of skipping.

Part I: Background and Motivation After this introduction in Chapter 1, we provide some fundamental background to recurrent neural networks. In Chapter 2, we give a brief historical overview before identifying essential design principles such as modularity and hierarchical structure that can be found in many state-of-the-art systems. As part of this review, we examine past and current research goals and challenges for recurrent networks. In Chapter 3, we argue that some of these challenges could be addressed by abandoning some of the traditional assumptions found in the temporal processing pipelines of past models. Drawing analogies to the human reading process and other models of time, we consequently introduce the recently proposed framework of conditional computation and review related models, addressing currently open questions that we aim to contribute to with this thesis.

Part II: Periodic Activation: We start our investigation in Chapter 4 with a comparison between more traditional models with leaky memory and the Clockwork RNN (CWRNN) as a proxy for conditional computation models in the context of learning multiple timescales with modular networks. The CWRNN does not update states at every timestep but based on periodic schedules referenced by an internal clocking mechanism. We compare the memory access capabilities of the CWRNN to alternative memory models, particularly leaky memory. We accomplish this with two different experiments, comparing five networks on continuous signals and sequences of discrete tokens. Using various visualization techniques, we take a closer look at how periodic activations self-organize. Chapter 5 extends the original CWRNN architecture with memory gates, evaluating this novel model, the CWLSTM, with four different gating variants on a language modeling task. Our results and subsequent analysis demonstrate both strengths and downsides of a periodic inductive bias.

Part III: Surprisal-Based Activation: Based on the conclusions from the previous part, we introduce a novel skipping mechanism based on surprisal in Chapter 6. Surprisal-based Activation (SBA) is designed to be modular and allows the underlying RNN model to observe its own rate of activation change while processing an input sequence and can thus inhibit updates if the information gain for the calculated latent state is considered too low. We primarily evaluate our model and the resulting seven variants on language modeling but also demonstrate that our main findings can be transferred to other prediction tasks. As part of an extensive evaluation in Chapter 7, we demonstrate that our SBA models have certain regularizing properties that can lead to slightly better performance than a baseline LSTM while simultaneously performing only 14% as many state updates. As part of this, we evaluate three different regularization penalties and four different variants of zoneout, improving the original algorithm.

Part IV: Hierarchical Attention and Surprisal: As successfully detecting important inputs and filtering out uninformative redundancies or noise requires models to have an understanding of salience, we continue our investigation with recurrent attention models. We start in Chapter 8 by exploring different approaches towards implementing attention in a hierarchical setup performing question answering tasks. Our modeling efforts focus on two different bottom-up approaches of hierarchically building up attention representations from words to sentences. Based on these findings, we introduce and evaluate two different skipping models for hierarchical processing in Chapter 9. Evaluating the interplay between skipping and attention, we find that our models maintain accuracies close to their baselines despite low update rates. Finally, in Chapter 10, we use attention itself as the basis for skipping. This allows us to push the models towards their limit, skipping most of the words in documents without a significant loss in accuracy. We achieve this with two different methodologies, using i) a wrapper model for recurrent architectures without introducing any additional hyperparameters, and b) ranked attention, which gives us control over the skip rates of the networks.

Part V: Closing: In Chapter 11, we summarize our findings and conclude with thoughts on remaining open questions and future research opportunities.

Chapter 2

Current Approaches and Challenges in Modeling Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are powerful models for sequence processing that have seen widespread use in a variety of applications in the past decade. Many advances result from a perpetual virtuous cycle - a self-reinforcing feedback loop - between the research community gradually enhancing models' capabilities and adopting more complex tasks. Therefore, practical requirements for model capabilities can vary as much as their practical applications. Nevertheless, some of the most impactful advances in Deep Learning have come from more foundational contributions to model design that improve the previous state of the art irrespective of the considered task scenario. The large impacts of Convolutional Neural Networks (CNN; LeCun et al. (1989)) on vision and speech systems, or of the Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber (1997)) on most sequence processing tasks come to mind as popular examples.

The goal of this chapter is to review the state of the art in modeling Recurrent Neural Networks and to provide an overview of some of the most pressing research questions that are still unanswered. By doing so, we will be discussing which future model capabilities might be essential to the development cycle of a new generation of universal problem solvers that are successful in a wide range of different tasks. As we will see, some of the approaches to model these capabilities have shown a remarkable promise in a number of different studies, whereas others have not yet been successfully scaled up to more complex scenarios due to a number of different reasons, indicating a potential impasse.

After providing some necessary foundations and a brief history of recurrent neural networks (Section 2.1), we will particularly focus on the topic of improving the representations of sequence learning models as these mirror the expressiveness and capabilities of neural models. In this context, we will introduce and review the challenges of learning representations that are scale-invariant in the temporal dimension, disentangled, modular, interpretable, and hierarchical. The chosen approaches share a similar background as they are vital to learning spatial and temporal compositionality in neural networks.

Most of the core principles explained in this chapter build the foundation for our modeling choices throughout this thesis.

2.1 A Brief History of Recurrent Neural Networks

The idea of using recurrence to model neural networks can be traced to the early 1980s in which some unsupervised recurrent models such as the Hopfield network (Hopfield, 1982) started to gain some traction. Ultimately, however, it was only after Rumelhart et al. (1986) significantly contributed towards popularizing the idea of training neural networks with backpropagation, that recurrent backpropagation networks followed soon after. This was perhaps not coincidentally as the authors of the seminal paper already propose a formulation of Backpropagation Through Time (BPTT) in order to learn “sequential structures”. The idea of implementing such sequential structures through recurrence was first introduced in the same year by Jordan (1986) and later widely popularized by Elman (1990), who introduced the concept of recurrent hidden units that can be trained with BPTT.

Elman’s network is nowadays often called Simple Recurrent Network (SRN) as it provides the basic building block for more complex recurrent models of the current Deep Learning era[1]. The SRN architecture has been shown to be Turing-complete, i.e. computationally universal (Siegelmann and Sontag, 1995).

2.1.1 The Simple Recurrent Network

An SRN extends a Multi-Layer Perceptron (MLP) by adding a recurrence term for the hidden layer before the transformation by an activation function f:

$$\mathbf{h}_t = f(\mathbf{x}_t\ \mathbf{W}_{xh} + \mathbf{h}_{t-1}\ \mathbf{W}_{hh} + \mathbf{b}_h) \tag{2.1}$$

where $\mathbf{x}_t$ is the input, $\mathbf{W}_{xh}$ and $\mathbf{W}_{hh}$ define the respective weight matrices of the input and recurrent layers, and $\mathbf{b}_h$ a bias vector. Making the hidden state $\mathbf{h}_t$ a function of its previous input $\mathbf{h}_{t-1}$, allows the network to consider context sequentially (compare Figure 2.1). The elegant simplicity of encoding states recurrently in such a way is that, at any given timestep, the network’s representation implicitly encodes the past context. However, it is easy to see the difficulty in having a

[1]Which is often seen to have started in 2012 with Convolutional Neural Networks winning the first ImageNet competition by a large margin (Krizhevsky et al., 2012).

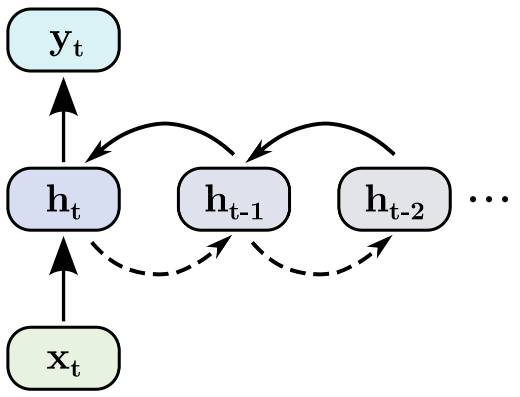


Figure 2.1: Illustration of a Simple Recurrent Network, unrolled in time. Solid arrows denote learned weights, dashed arrows the memory access.

single state encoding represent *all* past events. Thus, this efficient compression can quickly lead to memory loss, more broadly also known as catastrophic forgetting (Parisi et al., 2019), as the window for the past context grows over time. To access the entire temporal context, recurrent models are trained with BPTT, which can - in theory - unfold the network over its entire history. In practice, many early studies focused on solving problems where there was no need for long-term memory capabilities. Part of the reason for this was that researchers were quick to discover that the capability of retrieving context in SRNs with BPTT can, even for toy problems, be limited to less than 10 timesteps (Hochreiter, 1998). This restricted training SRNs with backpropagation to problems where only short-term memory was needed. The cause behind this limitation is called the Vanishing Gradient Problem (VGP).

2.1.2 Initial Challenges and Resurgence in the Deep Learning Era

While the difficulty of training RNNs and deep networks was experimentally verified by the late 1980s (Schmidhuber, 2015), Hochreiter (1991) was the first to formally identify the cause which was subsequently called the vanishing or exploding gradient problem. The problem arises when using standard activation functions such as sigmoid(x) or $\tanh(x)$ activation functions in hidden layers. As backpropagation is based on the chain rule, each layer's activation function derivatives have to be multiplicatively chained together. If each of these error terms is smaller than 1, each successive multiplication will decrease the cumulative error term further, decaying the memory exponentially with each introduced layer and causing the gradient to "vanish". Due to the range of the derivatives for sigmoid(x) and $\tanh(x)$, this is bound to have a measurable impact after about 3 layers or timesteps. Now, if all the error terms are actually above 1, the reverse effect occurs, where each subsequent layer increases the error term exponentially, causing "exploding gradients".

Research on modeling long-term capabilities was a focus throughout the 1990s and the vanishing gradient problem was studied from different angles such as, for example, the perspective of dynamical systems theory (Bengio et al., 1994). The first successful architectural solution to the vanishing gradient problem is the Long Short-Term Memory model (Hochreiter and Schmidhuber, 1997). It approaches the problem by keeping each error term constant (at *exactly* 1) through the use of linear activations. To offset the significant limitations of linear activation, Hochreiter introduces non-linear memory gates that can modulate the linear memory cell[2].

Despite this achievement, subsequent improvements to the architecture (Gers et al., 2000), and later attempts to simplify the theory behind LSTMs (Hochreiter et al., 2001), it took another decade for LSTMs to be demonstrated on more complex tasks such as speech recognition (Graves and Schmidhuber, 2005; Graves et al., 2006) or handwriting recognition (Graves et al., 2009). At the same time that LSTMs achieved state-of-the-art results on the TIMIT phoneme recognition benchmark (Graves, 2013), significant advances in Deep Learning (Krizhevsky et al., 2012) started to cause a significant shift towards neural networks in machine learning research. Nevertheless, one of the most important catalysts for research was the following rise of open source frameworks such as Theano (Bergstra et al., 2010), Keras (Chollet et al., 2015), TensorFlow (Abadi et al., 2016), and PyTorch (Paszke et al., 2019) along with the simultaneous transition towards GPU computing which made training (initially) more cost-effective than with CPU-based computing clusters. This development is still ongoing and has, since its start, allowed more researchers to train pre-implemented networks such as LSTMs and CNNs independently, verifying their effectiveness on a variety of different tasks in ever-faster research development cycles. In recent years, the LSTM model has additionally been simplified further (Greff et al., 2017). A model which only uses two gates, the Gated Recurrent Unit (GRU) has been introduced by Bahdanau et al. (2015), showing similar performance to LSTMs while requiring less memory (Chung et al., 2014).

Consequently, LSTM networks, as well as more advanced models that incorporate the LSTM, have successfully been deploying in a wide array of different sequence learning tasks, including production environments (Schmidhuber, 2015). The practical use of Deep Learning in real-world applications has increasingly moved research towards training ever-larger models on data collections that are often in the hands of private companies. The training cost behind most of the current state-of-the-art models makes them therefore impossible to reproduce in most academic institutions. As an example, a single Transformer network can incur up to \$981 of cloud compute costs (excluding any development costs from hyperparameter searches), whereas a neural architecture search can cost up to \$3M, simultaneously emitting as much CO_2 as five cars do on average for an entire lifetime (Strubell

[2]The LSTM will be introduced in more detail in Chapter 4 where we will use it in the context of a comparative study between different recurrent memory models.

et al., 2019). Similarly, according to one estimate (Li, 2020), training the recently released language model GPT-3 by (Brown et al., 2020), would cost over \$4.6M using Tesla V100 cloud instances.

In this research environment, the recently proposed Transformer model (Vaswani et al., 2017) has been shown to outperform RNNs on a large number of benchmarks, particularly in large-scale natural language processing tasks. While Transformers are sequential models, they do not model memory recurrently. Instead, they use self-attention, which was originally proposed in the context of recurrent models, and is repurposed in the Transformer to model sequential memory access. Nevertheless, most of the research questions that originated from RNN research, such as modeling long-term dependencies (Dai et al., 2019), can be transferred to Transformers, as both architectures are similar enough to share some of the same disadvantages.

2.2 Learning Multiscale Representations

As previously discussed, recurrence compresses state representations heavily, causing memory decay over long time spans. This is a byproduct of unrolling a network for training, which chains each timestep together. Above all, this biases the network towards the previous timestep $t - 1$ and without an explicit model of time, making it difficult to learn concepts such as “time spans” or events that have a clear beginning and end (see also Section 3.2). Consequently, ordinary RNNs are highly non-resilient to time rescaling, and an ordinary task can be rendered impossible, e.g., by inserting a small number of zeros between elements of the input sequence (Tallec and Ollivier, 2018). Therefore, designing explicit memory decay takes central importance in efforts to improve recurrent memory access. The previously discussed and widely popular LSTM addresses this using memory gating, which allows to dynamically scale the influence of previous states more independently from the unrolled state representation. An alternative approach to dealing with long-term dependencies is to model a network to operate at multiple different timescales simultaneously.

2.2.1 Challenges with Timescales

The basic idea of multiscale learning is to have some parts of the model operate on short timescales, accessing memory from the immediate context as usual, whereas other parts focus on long-term dependencies, encoding sequences that happen on longer timescales. This necessitates a mechanism that allows to either ignore the immediate context or biases the model towards long-term relationships. Suffice to say, the time spans encoded in “long” and “short” timescales are highly application-specific and timescales do not have to be as binary, i.e. there can be varying

Process	Timescale	Description
1 Planck time	10^{-44} s	Shortest theoretically measurable time interval
Shortest measured event	$2.47 \cdot 10^{-19}$ s	Photon traveling across a hydrogen molecule (Grundmann et al., 2020)
Action potential	10^{-3} s	Single neuron spike
Visual response	$1 \cdot 10^{2}$ s	Reflex response
Exchange	$2 - 10^{2}$ s	Dialogue
Solar Day	24 h	1 Earth rotation
Astronomical month	27.32 d	1 orbital period of the moon
1 Semester	4 months	
Solar Cycle	$\approx$ 11 a	Cyclic variation in solar activity
Human Lifespan	79 a	Life expectancy (developed world)
Formation of Mnt. Everest	$\approx$ 50 Ma	Ongoing process
Half-life of ^{235}U isotope	703.8 Ma	

Table 2.1: Examples for timescales.

degrees of timescales between those two extremes in which events take place in. Similarly, in real-world scenarios, identifying the number of timescales in a data stream is as difficult as identifying the number of “events” and their individual temporal boundaries. This is because neural networks, even those without recurrent connections, usually do not encode features discretely but work with continuous distributed representations. Without an additional mechanism to model discrete event boundaries, they can therefore not explicitly define a time span with a clear beginning and end.

However, timescales can even be challenging to define on a conceptual level. Table 2.1 shows an example of different events and their timescales. As can be seen, some time intervals can be physically measured (e.g. an astronomical month) whereas others are merely cultural inventions (e.g. a month in the Gregorian calendar). Likewise, while many scales are based on periodic events, others, such as the formation of a mountain, do not have an exact beginning and end but are part of a process with blurry event boundaries. Additionally, large changes in scales require the use of different units of time such as the metric prefix Ma (Million years) or relative units such as 10^{-x} s, as the human short-term memory has difficulty processing measurements with many digits and is generally limited to about 7 ± 2 “chunks” of information (Miller, 1956).

Due to this difficulty to clearly define timescales in the context of most machine learning applications, most multiscale models only define 2-3 timescales explicitly (Heinrich et al., 2018; Mikolov et al., 2015; Yamashita and Tani, 2008). Alternatively, this could also be explained by a data or task bias in which more timescales simply do not improve performance. An additional motivation for multiscale networks comes from computational neuroscience and the interest to model neural oscillations and spikes that are observed to operate at different frequencies in the brain. In this context, timescales are typically shorter and modeled with smaller frequencies (< 60 Hz (Wang, 2003)) and micro-level activities on shorter timescales are projected to macro-level activities on longer timescales (Déli et al., 2017). Recent models of neural oscillations also give some evidence that some tasks, such as predictive speech processing, can be broken down to only two timescales (Donhauser and Baillet, 2020). It is important to note, however, that neural time frames are significantly more limited than more abstract concepts and events present in complex machine learning problems. As there is a certain trend to move away from biologically plausible models in machine learning, more high-level representations could thus theoretically benefit from more timescales.

Goodfellow et al. (2016) write in their foundational book on Deep Learning that there are three main approaches to model multiscale training: i) adding temporal skip connections, ii) removing short-term connections for some units, and iii) using leaky units. The first two methods are closely related and reflect the main approach taken in this thesis. As such, these will be introduced more in-depth in Chapter 3 and applied to our own models. The leaky memory approach will be introduced next.

2.2.2 Leaky Memory

The main idea behind *leaky memory* (also called *leaky integration* or *leaky activation*) is to control the memory decay with a fixed parameter, the *leak rate* $\alpha \in [0, 1]$. The leak rate controls the amount of the previous context that is taken into account. As per Bengio et al. (2013a), a leaky hidden layer is defined as follows:

$$\mathbf{h}_t = \alpha \ \mathbf{h}_{t-1} + (1-\alpha) \ f(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2.2}$$

The introduced leak rate modifies the state update such that the influence of the past state is reduced by the amount α. Therefore, letting different parts of the network operate on different leak rates biases the model towards different long-term dependencies and thus timescales. Nevertheless, as Pascanu et al. (2013) note, the vanishing gradient effect is still present due to $\alpha < 1$, even though the leak should ideally expand its timescale. This is helped by the fact that leaky memory acts similar to the momentum term in gradient descent in that it can prevent sudden changes. This is because leaky memory is simply the *exponentially weighted moving average* (EWMA), the simplest form of exponential smoothing (Brown, 1963), applied to RNNs. Despite this simplicity, Tallec and Ollivier (2018) show that leaky models are invariant towards time rescaling in continuous-time models, still able to represent an input sequence $x(t)$ if it is rescaled by α such to $x(\alpha t)$. They additionally show that leaky models are invariant to time warping, the process of repeating each token multiple times, therefore stretching out the sequence in time. This holds true for discrete-time models as long as the data is not sampled faster than the model's own sampling rate.

More generally, leaky memory is related to the theory of self-organized criticality (Bak et al., 1988) and the separation of timescales axiom which dictates that external inputs enter a dynamical system at a much slower rate than the timescales of the internal dynamics (Das and Levina, 2019). In the context of machine learning, this has been adapted to Slow Feature Analysis (SFA), an unsupervised learning algorithm (Wiskott and Sejnowski, 2002), which is based on the more general slowness principle[3] and has been proposed by Becker and Hinton (1992). The slowness principle acknowledges that events on longer timescales differ from those in shorter timescales by the amount of temporal variation in the input signal. Consequently, applied to RNN representations (see also Section 2.5), the slowness constraint translates to $\mathbf{h_t} \approx \mathbf{h_{t-1}}$.

Mozer (1992) were one of the first to suggest leaky activation in so-called Reduced Description Networks. While this technique was perhaps most popularized by Echo State Networks (ESN; Jaeger et al. (2007)), the general principle has been applied in a variety of other models[4]. ESNs use a leak rate that is global for the entire network. This is typically different in approaches for backpropagation-trained networks where leak rates act as multiscale constraints. For example, Bengio et al. (2013a) investigate leaky memory in SRNs and sample leak rates from a predetermined range. Other models, such as the Multiple Timescale Recurrent Neural Network (MTRNN), limit themselves to only two timescales, i.e. "slow" and "fast" context units (Yamashita and Tani, 2008). Similarly, the Structurally Constrained Recurrent Network (SCRN) by Mikolov et al. (2015) simply uses a normal SRN layer to model shorter timescales and adds a parallel layer with leaky activation for longer timescales. A similar grouping of units by timescales is done in the Temporal Overdrive RNN by Futrell and Levy (2017). Modeling different leak rates for each layer has also been proposed by Wermter et al. (1999) in Recurrent Plausibility Networks (RPN), which uses temporal delays similar to the NARX model (Lin et al., 1996). The NARX model has later also been extended to Temporal Kernel Recurrent Neural Networks (TKRNN) which use leaky integrators for every unit (Sutskever and Hinton, 2010).

As discussed previously, choosing the correct leak rate remains a difficult task for all models as the timescales themselves are often not clearly interpretable outside of artificial toy tasks. As shown in Quax et al. (2020) and one of our recent studies (Heinrich et al., 2018), it can, therefore, be more beneficial to adaptively

[3]This principle has also been called the prior for "temporal and spatial coherence" (Bengio, 2013; Jayaraman and Grauman, 2016) and can even be interpreted from the perspective of Newton's first law of motion, which dictates that external forces gradually slow down physical objects in motion due to their velocity and momentum (Jonschkowski and Brock, 2015).

[4]In Chapter 4, we will show an experimental comparison between some of these models and compare them to memory models with skip connections.

learn the leak rates. Regardless, it is important to note that LSTMs and GRUs inherently have this capability as the forget and reset gates act as a dynamic leak rates which modulate the amount of context that is taken into account at each timestep. Despite this theoretical capability, gated models are not considered multiscale models as they are outperformed in respective tasks by more specialized architectures (see Chapter 4). This could indicate that even adaptive mechanisms require more explicit constraints such as the slowness principle to capture specific timescales effectively. Likewise, traditional models have no constraints that enforce diversity between learned representations such that they can end up focusing on different scales. Nevertheless, the GRU and LSTM have both been studied previously with leaky activations, such as in the Multiple Timescaled GRU (Kim et al., 2016; Moirangthem et al., 2017) or the Continuous Timescale LSTM (Yu et al., 2017c).

As discussed above, learning multiple timescales is closely linked with the idea of having different constraints and biases in different parts of the network. Grouping units and modularizing or partitioning layers by constraints is, therefore, an important modeling technique, which we will be reviewing next. However, most other approaches can be grouped into modeling hierarchical structures (see Section 2.4) or using shortcut connections and skip mechanisms (see Chapter 3), which we will review in the respective sections.

2.3 Specialization of Units

Training units to specialize on different features is of importance for multiple different research goals. First, as deep neural networks are high-dimensional systems, they are often difficult to interpret. Causes, among others, can be emerging codependencies (Srivastava et al., 2014) or redundancies (Rudy Setiono and Huan Liu, 1997) between neurons, a lack of structured data (Fan et al., 2020), an exploitation of beneficial artifacts in the data (Montavon et al., 2018), a critical information loss from breaking down complex representations into simpler symbols (Gilpin et al., 2018), or the previously described difficulty that representations might not recognize the appropriate timescales. In contrast, units that serve a specific function are thought to be interpretable concerning this function, which is an important requirement for explainability in machine learning. Second, generative models such the Generative Adversarial Model (GAN), can exploit distributions of specialized units to allow the generation of specific unsupervised features, which is of high interest for real-world applications in which users require full control and direct influence over small features of the generated output. And finally, as discussed in the context of learning multiple timescales, modeling multiple constraints into a network requires a local separation between units because ordinary networks have issues focusing on multiple features at different scales using the same representation.

Recent research efforts on modeling specialized units have concentrated on three key areas: i) learning disentangled representations, ii) using modularization to enforce different constraints, and iii) competitive learning, often in the context of avoiding catastrophic forgetting. In this section, we will be introducing these concepts and give a brief review of the existing research and main challenges.

2.3.1 Disentanglement

Recently, there has been much interest in unsupervised learning of disentangled representations for generative models. An intuitive definition of disentanglement is that "changes in a single ground truth factor should lead to changes in the representation" (Locatello et al., 2019). For example, given a dataset of face images, we want a specific unit to react to features of sunglasses, while another unit focuses on hats, maximizing their respective activations in the presence of these features. Generating images with different sunglasses from the one unit's distribution, a possible goal is, e.g., to avoid the generation of any changes in the hats - even though both concepts might be conceptually or statistically correlated through the data. What makes this problem difficult is that features such as "sunglasses" and "hats" are not necessarily class labels. Instead, they can be mere attributes that the model is supposed to find and isolate in an unsupervised manner. As such, disentangling representations can be seen as a special part in the overall research effort to learn specialized representations.

A common ground for most approaches is to favor disentangled representations that are both *discriminable* (displaying numerical variation) and *explainable* (by humans). Although the exact relationship between discriminability and explainability is not agreed upon (Hohman et al., 2020), it is both clear that there is no necessary causal relationship and that there can be no unique solution for both objectives, respectively. In particular, disentanglement requires discriminability, whereas discriminability does not have to automatically result in disentanglement. This already reflects the dilemma that the research community has neither found a clear well-formed definition of what it means to learn disentangled representations (Higgins et al., 2018). Instead, the approach is often a consequence of the specific task that is solved. Working towards a clear definition of disentangled representations, Eastwood and Williams (2018) posit that disentanglement, completeness, and informativeness should be required criteria. A slightly different definition by Ridgeway and Mozer (2018) requires representations to show modularity, compactness, and explicitness. In any case, most definitions have in common that the goal is to transform low-level and high-dimensional representations into high-level and low-dimensional symbolic variables (Higgins et al., 2018).

Due to differing perspectives on disentanglement, the topic has been approached by a variety of different methods such as penalizing the predictability of representations (Schmidhuber, 1992), maximizing mutual information between input and output (Hjelm et al., 2019), the F-statistic (Ridgeway and Mozer, 2018), meta-learning a transfer distribution (Bengio et al., 2020), factorizing (Kim and Mnih, 2018), and penalizing the total correlation between variables (Chen et al., 2018). Most of these assume generating factors to be statistically independent. As such, independent features have also been learned with adversarial objectives (Brakel and Bengio, 2017).

Nevertheless, a recent large scale evaluation of popular disentanglement metrics and objectives has found that many do not work reliably and that successfully reproduced disentanglement is more caused by favorable hyperparameters and random seeds (Locatello et al., 2019). The authors also postulate that the task is fundamentally impossible without inductive biases on both models and data sets. Similarly, Leavitt and Morcos (2020a) present evidence that regularizing for "class selectivity" (specializing each neuron on a specific class label) can impair generalization of CNNs, despite better model interpretability. Overall, these findings suggest a current overreliance on intuition-based approaches in interpretability research (Leavitt and Morcos, 2020b), a lack of robust evaluation metrics to allow fair comparisons between models (Do and Tran, 2020), as well as a lack of reproducibility on a set of diverse datasets (Locatello et al., 2019).

Moreover, the majority of the literature focuses on disentanglement in the visual domain. While some studies have investigated the disentanglement of representations in sequential tasks (Yang et al., 2015; Goyal et al., 2019), few explicitly model temporal characteristics such as temporal variations (Hsu et al., 2017; Higgins et al., 2018), suggesting that research on disentangling temporal representations is still in its infancy. A lack of research could result from the difficulty to clearly define temporal disentanglement and, therefore, further research objectives. While it is simple to define spatial features such as "a hat", a temporal feature can either be a spatial object's movement through time or something that is entirely disconnected from any concrete spatial objects. We can consider movies as an example as they are often structured in three acts with a beginning, middle, and ending. While a human observer can distinguish the beginning of a movie from the ending if the story is provided as context, there is usually no single scene, line of dialogue, or visual or auditory cue that defines the feature "introductory scenes". Rather, this feature is correlated with time itself (in this case, the movie's running time). As most recurrent networks do not explicitly model time, it is therefore challenging to learn actual temporal features. Consequently, it is an open research question whether and how the idea of disentanglement could be transferred to the temporal domain.

2.3.2 Modularity

Most deep learning systems are built in the form of a single large network with stacked and layered structures, which can lead to issues with generalization (Lamb et al., 2020; Goyal et al., 2019). This can be approached by partitioning a network into subnetworks or modules that operate based on different priors or constraints.

The hope is that, similar to ensemble learning, this can lead to a combination of experts that specialize in different features or functions (Jacobs et al., 1991). Modularized networks are different from disentangled representations so far as in their module constraints are typically much looser as modularized networks are typically not designed for interpretability and modeling distributions in generative models. As a consequence, modularization has a stronger focus on modeling distinct *functions* and *compositionality* rather than representations.

The success of modularity as a design principle is also reflected in the evolution of biological systems (Clune et al., 2013). The brain itself is an example of a biological network that is thought to be organized into different modules supporting distinct aspects of behavioral function (Yue et al., 2017). It is generally thought that ordinary artificial neural networks do not lead to the emergence of functional modularity from learning. Csordás et al. (2020) have conducted an analysis on FFNNs, CNNs, LSTMs, and Transformers, finding that these popular networks fail to reuse submodules, even when performing weight sharing between different tasks.

Applications for modularity include causal modeling (Parascandolo et al., 2018; Ke et al., 2019), meta-learning (Alet et al., 2018; Chen et al., 2020), conditional computation (Chapter 3), decomposing language (Andreas et al., 2016), as well as learning multiple timescales. Besides the leaky memory approach discussed in Subsection 2.2.2, specialization on timescales can also be achieved by different means using modular structures: for example, Carta et al. (2020) modularize hidden layers to separate low- and high-frequency features, adding modules to the network iteratively to progressively learn longer dependencies. A similar approach can be found in the Clockwork RNN which has modules operating on different update frequencies (Koutník et al. (2014); Chapter 4). Rahaman et al. (2020) design RNN subsystems that interact sparsely via a bottleneck of attention. Their Spatially Structured Recurrent Modules (S2RM) unifies the concepts of modularity and spatio-temporal structures in a single framework. Furthermore, a number of conditional computation approaches utilize modularization to learn multiple timescales (Chapter 3).

One common challenge with training modular networks is a negative effect called *module collapse* (Kirsch et al., 2018; Rosenbaum et al., 2019). It can occur if some modules are prematurely and greedily optimized over others due to them having simpler constraints and therefore affecting the loss function more immediately. Ultimately, this results in a low diversity between modules and can, in the extreme, lead to a similar outcome as with networks without modules. This is typically addressed by regularizing for variety in the modules (Kirsch et al., 2018). Feature diversity can also be achieved by routing. Routing networks consist of two components: a function block (e.g. a neural network layer) and a router which chooses a function block for recursive processing (Rosenbaum et al., 2018). In this setup, a single expert is active at any given time. Besides module collapse, challenges for routing networks include overfitting on spurious activations and achieving a stable state after initialization (Rosenbaum et al., 2019). Additionally, as with all previously discussed methods such as multiscale learning or disentangling representations, *testing* evaluating compositionality is difficult and remains an unsolved problem (Hupkes et al., 2020).

2.3.3 Competitive Networks

While modular networks typically achieve compositionality through the collaboration of subnetworks, a slightly different approach is to let them directly compete for the best solution. Historically, this is done in an unsupervised manner, even though this idea has been adapted to supervised models more recently. The idea of *competitive learning* in neural networks was first introduced by Rumelhart and Zipser (1985) who proposed a set of hierarchically layered units where units within clusters dynamically compete for propagating their activations. Self-Organizing Maps (SOM; Kohonen (1982)) are one example of such networks that have since been extended to more sophisticated networks such as Grow-When-Required (GWR) networks to enable continual learning (Parisi et al., 2019).

In the context of supervised learning, competitive activation units have been proposed in CNNs for learning multiscale convolutional filters (Du et al., 2018; Liao and Carneiro, 2015). To a certain degree, max-pooling in CNNs can also be seen as competitive though it differs from winner-take-all approaches in that reduces the number of features (Srivastava et al., 2013). This gap is closed by maxout networks (Goodfellow et al., 2013), which augment max-pooling with dropout in a winner-take-all fashion. Local winner take all (LWTA) networks operate similarly by selecting linearly activated units, binarizing the activations via thresholding (Srivastava et al., 2013). Srivastava et al. (2015b) analyze the emerging subsystems of LWTA, maxout, and ReLu networks, concluding that low interference between subnetworks facilitates training.

In recurrent networks, lateral inhibition has been used for grouping units competitively (Xie et al., 2002; Mao and Massaquoi, 2007). Currently, there is an increased research focus on continual learning with RNNs in which competitive learning is often utilized. Continual learning describes the scenario in which a model has to continually learn a sequence of tasks in an incremental manner. The main difficulty of transferring knowledge between tasks is *catastrophic forgetting*, which occurs when memory from a previous task is being overwritten. In this context, Parisi et al. (2018) propose a self-organizing dual-memory model with memory replay, whereas other approaches, inspired by predictive coding, introduce local sparsity constraints (Ororbia et al., 2020). Nevertheless, Ehret et al. (2020) argue in their analysis of RNNs for continual learning that regularization approaches that enforce constraints on units offer more versatility than competing models as they require no replay or additional parameters.

2.4 Hierarchical Models

Most complex systems, whether physical, biological, or social, take hierarchical forms (Simon, 1962; Ravasz and Barabási, 2003). An important attribute of hierarchical systems is that concepts at higher levels can be decomposed into concepts at lower levels in a top-down manner. As such, an intuitive approach to model compositionality is to design hierarchical networks.

An early example for a hierarchical network is the Neocognitron (Fukushima, 1988) which, inspired by cells from the brain's visual system (Hubel and Wiesel, 1962), alternates between layers of "S-cell" feature extractors and "C-cells" learning positional invariance and reducing the feature dimensions between layers. Convolutional Neural Networks (CNN) adapt the same principle using convolution and max-pooling (LeCun et al., 1989). As modern deep learning systems have historically emerged from CNNs (Krizhevsky et al., 2012), designing large networks with multiple stacked layers and hierarchical representations is one of the most important design principles of current state-of-the-art deep learning systems (Vaswani et al., 2017; Szegedy et al., 2017; He et al., 2016).

For recurrent networks, the concept of "depth" is a bit more complicated to define and model. On the one hand, temporal depth is an inherent architectural feature as RNNs are unfolded in time and backpropagated through many timesteps. RNNs can therefore be considered as "the deepest of all NNs" (Schmidhuber, 2015). On the other hand, horizontal pathways of ordinary RNNs offer linear transitions that are still considered shallow and without any hierarchical structure (Pascanu et al., 2014). Therefore, it is not sufficient to stack multiple RNN layers in order to learn temporally deep representations.

For this reason, a number of proposed "deep RNNs" modify the underlying temporal transitions with more representational power. An early example of such a model is the Deep Transition RNN (DT-RNN) by Pascanu et al. (2014), which adds non-linear layers to the recurrent transitions. One problem with increasing recurrence depth in such a way is that any vanishing or exploding gradient effects are amplified. As such, the authors propose the use of shortcut connections, which effectively shorten gradient paths (Zhang et al., 2016). Shortcut connections are, therefore, an important design principle of deep RNNs (see also Section 3.5). Zilly et al. (2017) take the idea of deep transitions one step further with Recurrent Highway Networks (RHN) by adding Highway Networks (Srivastava et al., 2015a) to the recurrent transitions of LSTM networks. More recently, attention has also been used to model sparse transition dynamics: recurrent Independent Mechanisms (RIM) can dynamically control information flow (Goyal et al., 2019), even bidirectionally (Mittal et al., 2020). The main idea with these models is to model nearly independent transition dynamics in modules. Independent units are also modeled with the Independently Recurrent Neural Network (IndRNN; Li et al. (2018)) which replaces the dot product with the Hadamard product and encodes relationships between units through a hierarchy of multiple layers. Not suffering from the vanishing and exploding gradient problem, the authors were able to train networks successfully with up to 21 layers.

A different approach to hierarchical RNNs is to explicitly model inductive biases from the underlying task in a model's structure. One such example are (Recurrent) Graph Neural Networks, which model relational inductive biases and can be used with graph-structured data (Wu et al., 2020). Applications in natural language processing are especially suitable for hierarchical inductive biases as natural language is hierarchical and often modeled in tree-like structures such as parse trees. Hierarchical organization that is modeled in tree-like topologies has particularly been shown to be successful for tasks such as sentiment analysis as demonstrated by the Tree-LSTM (Tai et al., 2015) or Recursive Neural Networks which have been used to train the *Stanford Sentiment Treebank* (Socher et al., 2013) and to parse natural scene images and sentences (Socher et al., 2011). Shen et al. (2019) propose to order neurons in their ON-LSTM to implement a hierarchical tree structure of LSTM cells with additional multiscale learning mechanisms: lower-ranking neurons update more frequently (retaining short-term information), while higher-ranking neurons update less frequently (keeping long-term information). This is realized by ensuring that each neuron can only update/forget if all its child nodes (i.e. lower-ranking neurons) had an update/forget operation. Nevertheless, while such models excel at tasks they were designed for, such a strong inductive bias makes them difficult to use in tasks where the underlying assumptions do not hold. As we will show in Chapter 8, simpler hierarchical structures that are based on characters, words, and sentences (which can be found in most NLP tasks), can already serve to improve baseline performance. Similarly, separating semantic and syntactic processing has been shown to lead to compositional generalization (Russin et al., 2020).

2.5 The Stability/Discriminability Dilemma of Temporal Coherence

As discussed in Subsection 2.2.2, the slowness principle of enforcing temporal coherence can be used to modulate temporal relationships on different timescales. In this section, we will present an informal argument as to why it is difficult to design recurrent multiscale models that are both stable and discriminable, unless they have a modularized or hierarchical structure. Our analysis motivates multiscale learning approaches for models where high- and low-level features are distinguished by their slow-/fastness such that a sequence of learned low-level representations changes faster than those of high-level representations. This includes methods based on Slow Feature Analysis (SFA), traditional multiscale oscillation models, and other models with structures of temporal variance.

To recap, the slowness principle enforces temporal coherence, minimizing variation between timesteps. Translated to RNN state updates, this can be formulated as $\mathbf{h}_t \approx \mathbf{h}_{t-1}$ such that units only change slowly over time. In order to understand the importance of temporal coherence for recurrent networks, we have to first establish that a class of popular recurrent memory models follows this principle. This class is defined by leaky memory models[5]. To see the relationship between slowness and leaky memory, let us consider how the range of the leak rate α affects the representations formed by:

$$\begin{aligned} \mathbf{h}_t &= \alpha\ \mathbf{h}_{t-1} + (1-\alpha)\ f(\mathbf{h}_{t-1}\mathbf{x}_t) \\ &= T_1 \qquad + T_2 \end{aligned} \tag{2.3}$$

where the term $T_1 = \alpha\ \mathbf{h}_{t-1}$ scales the past activation and $T_2 = (1-\alpha)\ f(\mathbf{h}_{t-1}, \mathbf{x}_t)$ the current one. Given an $\alpha \in [0,1]$ that is close to 1, i.e. $(1-\alpha) = \epsilon$ for a small $\epsilon > 0$ (which we denote as $\alpha <_\epsilon 1$), we can then infer that the first term T_1 is close to $\mathbf{h}_{t-1}$ and the second term T_2 close to 0:

$$T_1 = \alpha\ \mathbf{h}_{t-1} <_\epsilon \mathbf{h}_{t-1} = \mathbf{h}_{t-1} - \epsilon_1, \tag{2.4}$$

$$T_2 = (1-\alpha)\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) >_\epsilon 0 = \epsilon_2, \text{ since } (1-\alpha) >_\epsilon 0, \tag{2.5}$$

which leads to $\mathbf{h}_t = \mathbf{h}_{t-1} - \epsilon_1 + \epsilon_2$, or in other words, $\mathbf{h}_t \approx \mathbf{h}_{t-1}$, which is the slowness constraint. Similarly, $\alpha >_\epsilon 0$ leads to $\mathbf{h}_t = \epsilon_1 + f(\mathbf{h}_{t-1}, \mathbf{x}_t) - \epsilon_2$ and therefore $\mathbf{h}_t \approx f(\mathbf{h}_{t-1}, \mathbf{x}_t)$, which describes the "fast" regular update. It follows that leaky memory models the slowness constraint, where α modulates the degree of slowness.

The unparameterized slowness constraint $\mathbf{h}_t \approx \mathbf{h}_{t-1}$ can also directly be modeled as a regularization penalty. Krueger and Memisevic (2016) demonstrate this by penalizing the *norm* of successive activations:

$$R = \beta \frac{1}{T} \sum_{t=1}^{T} \Big(\|\mathbf{h}_t\|_2 - \|\mathbf{h}_{t-1}\|_2 \Big)^2, \tag{2.6}$$

where $\beta \in [0, \infty[$ is a hyperparameter to weigh the cost. This norm-stabilization mainly addresses stability issues with unbounded activation growth (such as with ReLu activation). Naturally, it is possible to directly stabilize the hidden state activations $\mathbf{h}_t$, rather than using their norm (Jonschkowski and Brock, 2015). This leads to explicit temporal coherence between states, minimizing variations between timesteps such that $\mathbf{h}_t \approx \mathbf{h}_{t-1}$. However, this approach has a practical issue: $R = \beta \frac{1}{T} \sum_{T}^{t=1} (\mathbf{h}_t - \mathbf{h}_{t-1})^2 = 0$ can be achieved by the trivial (constant) solution $\mathbf{h}_t = \mathbf{h}_{t-1}$. Such a regularization penalty negates the intended purpose and penalizes *any* temporal relationships globally. Solutions involve applying this only

[5]Even though gated memory models such as the LSTM or GRU can be seen as dynamic variants of leaky memory due to their forget or reset gates (see Subsection 2.2.2), this adaptivity can theoretically cause constant resets, resulting in low temporal coherence. Gated models are therefore not included in this observation.

locally or conditionally[6], as well as applying the slowness constraint on a mapping of the states such as the norm or higher-order derivatives (Jayaraman and Grauman, 2016). Minimizing changes on higher-order derivatives of the feature space leads to temporal coherence based more on the feature *transitions* than differences.

As we have shown in Subsection 2.2.2, slowness constraints can be applied at an arbitrary level of abstraction to encode features with varying timescales. As transition dynamics can vary dynamically, we can interpret slowness as inherently tied to the problem of learning multiple timescales. Indeed, SFA is based on the assumption that high-level features emerge from more slowly varying representations than low-level abstractions which are separately encoded based on more frequent or faster variations (Wiskott and Sejnowski, 2002). Above, we have reduced SFA to the general idea of time-averaging with leaky memory to minimize representation differences between timesteps. We can therefore investigate similarities between SFA and time-averaging time scaling approaches.

Goroshin et al. (2015) group most SFA approaches into three categories: works that investigate i) the feature parameterization, ii) methods to avoid trivial solutions, and iii) imposing additional priors or constraints such as independence or sparsity on learned features. They further point out that the existence of the trivial (constant) solution results in an unavoidable trade-off between discriminability and stability of the learned features which is evident from the fact that constant features (high stability) reduce state entropy (low discriminability). We argue that such a *temporal discriminability-stability trade-off* can be viewed as a special case of the general bias-variance trade-off in machine learning since the achieved balance is entirely dependent on the learned features' *temporal* variance and bias, which in turn is dictated by the learned recurrent weights.

Under this assumption, all time-averaging approaches for redundancy elimination, regardless of their level of abstraction, suffer from this trade-off that forces a balance or choice between stability and discriminability of temporal features (see also Figure 2.2). It is however important to highlight that this only concerns the temporal dimension since we restrict our discussion to temporal redundancy. Under these assumptions, we can arrive at a solution for models that can achieve both: in cases where features change over larger timescales, a high temporal discriminability is not only unnecessary, it is by definition counterproductive. Consequently, on the temporal dimension, discriminability is *ideally* inversely correlated to the associated feature's timescale. The ideal stability on the other hand is directly correlated to the timescale. That is, features encoded over longer time spans (larger timescales) benefit from lower discriminability (and higher stability), while features from short-term events (smaller timescales) benefit from a larger discriminability and lower stability. Almost all multiscale models that we have reviewed in this

[6]As a matter of fact, the trivial solution $\mathbf{h}_t = \mathbf{h}_{t-1}$ describes the process of skipping state updates. If this is executed conditionally, we arrive at the definition for conditional computation, which is described in Section 3.5 and applied in all our models presented in this thesis.

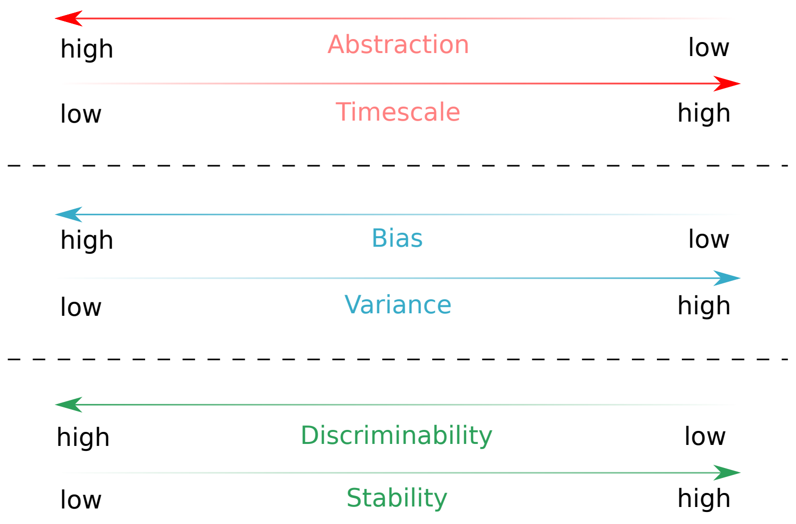


Figure 2.2: Antagonistic relationships between abstraction and timescale, bias and variance, discriminability and stability, leading to necessary trade-offs.

chapter comply with this principle by modeling these two functions separately under different constraints, e.g., by separating dynamics with modular or hierarchical structures.

Such architectures lead to the realization of *multiple*, *parallel* trade-off solutions, i.e. multiple timescales. In such models, individual representations (in the form of units, modules, or layers) and their dynamics diverge between discriminability and stability. More generally, an ensemble of N independent models can model N different trade-offs. For very large N, we can then find such an ensemble that covers the entire range, effectively eliminating the trade-off problem for the larger model (even though it still holds for each individual representation). Note how this analogous to the more general bias-variance trade-off problem which motivates ensemble learning: individual independent models of low bias and high variance can be averaged into a single model, lowering the overall variance. The same can be achieved with regularization, which includes the above-discussed slowness constraint.

Let us briefly also address the controversial question of whether building increasingly larger models can present a viable alternative to constructing inductive biases. On the one hand, there is growing evidence that, for modern deep learning models, variance can actually decrease with growing model size (Neal et al., 2018) and that over-parameterized complex models are better at high-dimensional interpolation due to their inductive bias (Belkin et al., 2019). It has also been shown that the intrinsic dimension[7] for common benchmarks is lower than previously expected (Li et al., 2018), which hints at many standard models being inherently complex. Consequently, it is possible that, given the right inductive bias, an RNN with increasing parameter size could lead to an improved approximation of larger

[7]The minimum number of parameters necessary to encode and solve a task.

timescales instead of overfitting on short-term dependencies. On the other hand, extremely large modern language models such as GPT-3 (Brown et al., 2020) can, for example, produce very realistic short snippets of text but still have difficulties maintaining narratives over longer periods of time, consistency of gender or personality (Elkins and Chun, 2020), and have a critical lack of reasoning capabilities (Floridi and Chiriatti, 2020). This indicates that current state-of-the-art sequence learning models still lack the appropriate inductive bias to model concepts based on long-term relationships.

2.6 Chapter Summary

In this chapter, we have reviewed current research challenges of modeling recurrent neural networks. The past decade's shift towards deep learning approaches has brought an increased focus on improving compositionality in large networks. Likewise, the growing use of deep learning systems in real-world applications has led to a stronger demand for more powerful representations that allow, e.g., symbolic manipulation, multi-task learning, causal inference, or model interpretability.

From this perspective, we have identified several important design principles for recurrent networks such as multiple timescales, modularity, disentanglement, locally applied constraints, and hierarchical organization. While these principles have led to significant progress, each of the models presented in this chapter follow a linear discrete model of time in which state updates and computations are performed in a synchronized manner for each timestep. In the next chapter, we will introduce conditional computation, an alternative framework based on asynchronous processing, and discuss why it naturally fits the discussed objectives, addressing many of the research challenges presented in this chapter.

Chapter 3

Modeling Natural Language Processing with Conditional Computation

This chapter motivates the need for sequence learning models that do not process inputs in a chained manner, but can skip between inputs. As such, we will be introducing the conditional computation framework and reviewing related approaches. The presented processing methodology holds opportunities for both more computationally efficient models and learning more explicit temporal representations. As this thesis primarily concentrates on Natural Language Processing (NLP) applications, we start by giving a brief overview of deep learning methods for NLP and introduce our main applications of language modeling and question answering (Section 3.1). We continue by discussing the main difficulties with modeling time (Section 3.2) before arguing how the human reading process is a familiar example for efficiently processing sequential inputs, despite processing information very differently than recurrent networks (Section 3.3). After briefly examining the importance of efficient processing and representations for the future of deep learning (Section 3.4), we introduce the main concepts behind the methodologies used in this thesis which can be found under the framework of conditional computation (Section 3.5). We end by exploring current challenges and open questions in the field of conditional computation (Section 3.6).

3.1 Natural Language Processing

Natural Language Processing (NLP) is a wide multi-disciplinary research field that is concerned with the automatic processing, analysis, and representation of natural language. The purpose of NLP is to achieve human-like language processing capabilities in a number of different applications (Liddy, 2001). Examples for core NLP applications include machine translation, document classification, sentiment analysis, question answering, dialogue systems, natural language generation, summarization, named entity recognition, or parsing. The ultimate goal of NLP research, achieving human-like performance on such tasks, is often described by the term Natural Language Understanding (NLU) and considered AI-hard. While current systems are able to emulate individual aspects of language understanding and reading comprehension, comprehensive language understanding requires understanding on a semantic level rather than a syntax-level. Semantic understanding and subsequent manipulating of high-level concepts is necessary for more advanced goals that go beyond information retrieval, such as drawing inferences or performing causal reasoning.

Historically, NLP research can be differentiated by symbolic, statistical, and connectionist approaches (Liddy, 2001). While symbolic approaches represent knowledge-based methodologies such as rule-based systems, statistical approaches such as the Naive Bayes classifier model statistical relationships from example documents with the aim of generalizing to unseen documents. While these approaches have dominated early research, connectionist models (neural networks) have extended early statistical models, and occupy a central position in modern NLP research as the majority of current state-of-the-art systems for NLP are based on neural networks. While this is driven by increased computational resources and data availability, some key advances in neural network training and modeling, particularly in representation learning, have also played a vital role. In the following, we will briefly introduce these advances by introducing the main applications that we use in this thesis for evaluating our models.

3.1.1 Language Modeling

Language Modeling describes the task of learning statistical models for text prediction. Such language models can then, e.g., be used for autocomplete or text generation systems. The general formulation of the task is to learn the probability distribution of a word sequence $\mathbf{w}$ consisting of $|\mathbf{w}|$ words w_i ($1 < i < |\mathbf{w}|$), where each word is conditionally dependent on the previous words of the sequence (Russell and Norvig, 2009):

$$\begin{aligned} P(\mathbf{w}) &= P(w_1, \ldots, w_{|\mathbf{w}|}) \\ &= P(w_1) \cdots P(w_{|\mathbf{w}|} \mid w_1, \ldots, w_{|\mathbf{w}|-1}) \\ &= \prod_{i=1}^{|\mathbf{w}|} P(w_i \mid c_i) \\ &\approx \prod_{i=1}^{|\mathbf{w}|} P(w_i \mid w_{i-N+1}, \ldots, w_{i-1}) \end{aligned}$$

We refer to $c_i = w_1, \cdots, w_{i-1}$ as the *context* from which w_i is predicted, which we limit to $N-1$ words to model finite memory. This gives us above approximation which is also known as N-grams. N-grams provide the simplest language modeling approach by predicting the next item in the sequence based on the previous $N-1$ words. While N-grams can provide useful features, neural language models have become more popular as distributed representations scale better with larger vocabulary size (Bengio et al., 2003) and require lower memory complexity (Mikolov et al., 2012).

When using RNNs for language modeling, the goal is to now model the joint probability distribution $P(\mathbf{w})$ by approximating $P(w_i|w_{i-N+1}, \cdots, w_{i-1})$. This means that the RNN processes the text sequentially, word by word, continually predicting the next word at each timestep without supervision from labels. Overall, we can also generalize the input from a sequence of words to an arbitrary symbolic sequence. A single input of this sequence, also called *input token*, can then be based on different resolutions such as a characters, sentences, or paragraphs.

Generally, language modeling is performed at the word level. While word-level language models have the downside of an increased dimensionality with large vocabulary sizes, they are able to work with smaller sequence lengths than character-level language models. Even though character sequences are significantly longer, requiring significantly more computational resources for training, character-level models have no predetermined vocabulary except for the characters. Instead, models have to learn the vocabulary on their own by continually predicting on character-level. This leads to the additional advantage that they are able to learn novel words and morphological aspects, though this has been shown to be difficult in practice without additional information (Vania et al., 2018; Santos and Zadrozny, 2014).

As we can continually predict tokens from a trained language model, we can use it to generate text for evaluation. While human evaluation is considered optimal for evaluating different aspects of the produced language, it is oftentimes too labor-intensive or expensive in practice. Evaluation metrics provide a simpler alternative, particularly for large-scale experiments in the development stage. In general, the quality of a trained language model is evaluated by their *perplexity* (PPL), i.e. the exponentiated Cross-Entropy loss $H(\mathbf{p}, \mathbf{q})$ of a given model prediction $\mathbf{q}$ based on the true label distribution $\mathbf{p}$. The Cross-Entropy itself is the average per-word log-probability:

$$\begin{aligned} \mathrm{PPL}(q) &= 2^{H(\mathbf{p},\mathbf{q})} \\ &= 2^{-\sum_{i=1}^{N} p(w_i)\log_2 q(w_i)} \\ &= 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 q(w_i)} \end{aligned} \tag{3.1}$$

For character-level models, performance is generally reported directly in entropy, i.e. bits per character (BPC). The perplexity expresses how many equally likely predictions the model has on average. Therefore, a low perplexity indicates a more confident language model that captures the underlying probabilities better than a model with high perplexity. To avoid overfitting on low perplexities, this metric is only used to test for generalization on validation and test sets, whereas Cross-Entropy is typically used as the training loss.

Besides text prediction and generation, neural language models have an additional advantage that allows their use in a number of different tasks. As the model learns a distributed representation for the words, it clusters words with similar context closer in the parameter space, which can lead to the encoding of some basic semantic context. Such representations, called *word embeddings*, can then be used as features in other tasks.

3.1.2 Word Embeddings

Word embeddings provide distributed word representations that encode semantic similarities between words based on their statistical co-occurrences. While embeddings have been studied earlier in the context of dimensionality reduction (Bengio et al., 2003), they have been widely popularized after the introduction of the word2vec algorithm by Mikolov et al. (2013). The general idea is to use a shallow neural network to predict a word based on its surrounding context (or alternatively predicting the surrounding context of a word). By mapping discrete word representations (e.g. bag of words or one-hot) to continuous representations, the learned weight matrix, in this context called the *embedding matrix*, then contains a continuous word vector representation for each word in the vocabulary. Word vectors with a smaller euclidean distance can then be interpreted as more similar.

While word embeddings can be trained end-to-end to capture the statistics of the training corpus, it is also possible to initialize the first layer of a network with an embedding matrix that was *pre-trained* on a different, typically much larger, corpus. If these embeddings are transferred to a new model and training is continued with different data, we speak of *fine-tuning* these embeddings. The practice of transfer and fine-tuning allows networks trained on smaller datasets to include semantic relationships from more data sources. This development and the subsequent availability of word embeddings trained on large amounts of data has significantly contributed to the success of neural networks in a wide array of NLP applications.

After the introduction of word2vec, subsequent approaches have attempted to incorporate more knowledge into embeddings. For example, GloVe (Pennington et al., 2014) includes global corpus statistics in the form of co-occurrence probabilities. Speer and Lowry-Duda (2017) even extend word embeddings with further knowledge from ConceptNet, a multilingual semantic network about word meanings and relationships.

Most recently, there has been an important shift away from pre-training shallow representations, towards contextualized and hierarchical multi-layer representations. This development has culminated in Bidirectional Encoder Representations from Transformers (BERT; Devlin et al. (2019)) which is currently considered state of the art for a large number of NLP applications. BERT builds on and integrates multiple prior key approaches. The first approach, Embeddings from

Language Models (ELMo; Peters et al. (2018)) addresses the problem that traditional word embeddings only provide a single vector for a word, even if there can be multiple meanings depending on the context. Instead of using the surrounding context, ELMo uses all layers of a bidirectional language model, encoding the entire history of the sequence. This provides contextualized word embeddings allowing word-sense disambiguation, e.g., `drawing` gets a different word vector in the context of "`drawing from experience`" than in "`drawing a picture`". The second development was Universal Language Model Fine-tuning (ULMFiT; Howard and Ruder (2018)), which improves transfer learning of language models by addressing the common issue that fine-tuning of embeddings can lead to catastrophic forgetting. The third building block for BERT is the use of Transformers networks which have shown better scalability on large corpora (see also Section 2.1). BERT integrates these concepts by providing embeddings from the decoder layers of a Transformer language model that has been trained on large amounts of text data. These embeddings can then be transferred to other models for further fine-tuning on other, so-called *downstream tasks* for prediction or classification.

3.1.3 Question Answering

Question Answering (QA) is a typical machine learning problem which is concerned with the automated answering of questions. Generally, NLP research distinguishes text-based QA from Knowledge-based QA (KBQA). While text-based QA is based on information retrieval from text documents, KBQA produced answers from knowledge bases such as Freebase (Bollacker et al., 2008) or DBpedia (Auer et al., 2007). Other forms of question answering include community-based QA (Liu et al., 2008) or Visual Question Answering (VQA) based on images (Wu et al., 2017).

Most text-based QA systems are focused on *factoid questions*. Questions like "`who is the head of state of Germany?`" can be answered with one or few words by querying simple facts (Jurafsky and Martin, 2008). In contrast, non-factoid questions like "`how do you become successful?`" are less aimed at extracting facts but at providing longer and potentially more abstract answers. As such, they often require some form of interpretation or reasoning.

Figure 3.1 illustrates an abstract template for most existing neural network approaches for text-based QA. Most approaches lean on word embeddings in that they assume that the answer is semantically similar to text passages that semantically relate to the question. Consequently, both document and question are encoded by neural networks. Their respective word embeddings are then integrated into a single representation, often by measuring similarities between embeddings through attention mechanisms[1] or a dot product. This integrated embedding, representing parts of the text most similar to the question, is then used by a decoder to produce

[1] Chapter 8 provides a deeper introduction into this methodology.

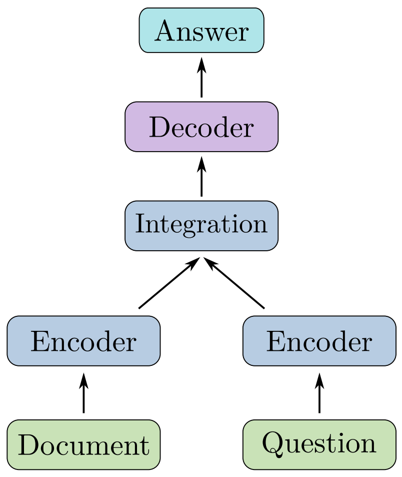


Figure 3.1: The basic components of a neural network performing text-based question answering. Documents are encoded separately from questions and integrated into a joint representation. A decoder maps this representation to an answer. In practice, the encoder, decoder, and integration are modeled with arbitrarily complex multi-layer network structures and can include additional information (e.g. from knowledge bases) or further processing (e.g. for document or passage retrieval).

an output answer. For non-factoid questions, the output can be a sequence which is constructed by decoder generating text. Alternatively, this can also be modeled without sequence generation, namely as a classification problem by providing the start and end positions in the document for the answer. If the answer is a single word, the network can also be modeled as a classifier using a softmax output layer.

Depending on the form and structure of the answer, different evaluation metrics can be used to measure the performance of systems. Typical classification metrics such as accuracy can be measured based on whether answers exactly match the data labels. The F1-Score can be used for span-based answer labels (Rajpurkar et al., 2016), while free responses are generally evaluated with metrics known from machine translation or summarization, e.g., BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), or METEOR (Banerjee and Lavie, 2005). However, such N-gram-based metrics have been shown to perform worse on more abstractive questions (Chen et al., 2019a).

Recent neural approaches increasingly focus on questions answering in the context of reading comprehension. A particular variant of this is cloze-style question answering (Hill et al., 2016). In a cloze-style query, a word is removed from a text passage and the system needs to infer how to fill this slot correctly. For QA tasks, the removed word is often part of the question, requiring the model to reconstruct the question itself by understanding its relationship to the document. However, even though current state-of-the-art systems are capable of simplified forms of reading comprehension, they are generally still too simplified to be considered NLU systems as too many approaches rely on information retrieval based on rich word representations extracted from large corpora. Advancing the state of question answering systems is a challenging problem as it is time-consuming and cost-intensive. This not only concerns human evaluation but especially the creation of datasets. As hand-crafting large datasets requires significantly more effort than in other NLP tasks (Rajpurkar et al., 2016), many datasets are crowd-sourced or automatically generated by extracting questions from text based on specific rules. The difficulty is to avoid data biases that come from rule-based labeling, which can result in machine learning models to focus more on reconstructing these rules rather than gaining a semantic understanding about the questions.

3.2 Modeling Time

3.2.1 The Problem of Time Perception

The difficulty of designing models capable of temporal abstraction can be traced back to the difficulty of modeling time such that it reflects our own experience and perception of time. Moreover, the perception of time is a highly subjective experience that can differ from objective time (Van Wassenhove et al., 2008).

There is even a cultural bias to the perception of time as was first suggested by Hall (1976). The study of chronemics has identified *monochronic* cultures (time is segmented into small precise units, things are done one at a time) vs *polychronic* cultures (multiple things can be done at the same time, time is perceived continuously with no particular structure). While the OECD developed countries are primarily monochronic, there are significantly more polychronic cultures in developing countries (Hall, 2000). Würtz (2017) note that these differences even correlate to whether cultures have high-context or low-context communication such that, for example, polychronic cultures in South America use more nonverbal and paraverbal communication, also drawing from the temporal context in which the conversation takes place. Cultural differences such as these can manifest as data biases in machine learning models.

Cognitive science has so far not identified a singular brain region for the perception of time. Ivry and Schlerf (2008) suggest that current models can be classified into two different categories: they either a) assume the presence of specialized, modular, and distributed representations or b) surmise that temporal information is not represented explicitly in the brain. Models of the latter category include those that encode time as entirely context-dependent (Karmarkar and Buonomano, 2007). While Recurrent Neural Networks are not cognitive models, they also encode temporal relationships only indirectly through context.

This brings certain challenges (see also Section 2.2) since most time units are discrete (e.g. seconds, minutes, hours, years). If we wish to encode temporally variable events such as a “moment” or “dialog act” in a machine, they have to be both modeled *and* represented based on the internal temporal model of the machine. This is difficult for recurrent models as they model temporal relationships and time intervals entirely through context and without explicit representation. Therefore, the manner in which context memory is *accessed* during processing defines what kind of relationships can be encoded. Typical RNNs model time in evenly discretized time intervals and inputs are read sequentially in one direction. At each time step, only the previous timestep’s context is taken into account recurrently during forward passes, leading to a short-term memory bias on ordinary RNNs. Constantly updating the internal model in every timestep, even if the input sequence provides no new information, thus provides both a computational overhead as well as a temporal model that is strongly biased towards relationships that are chained sequentially. Consequently, alternative temporal models have been suggested for recurrent models.

3.2.2 Computational Models of Time

The original recurrent model utilizes difference equations to model evenly discretized time intervals. This can be generalized to continuous-time models with differential equations (Sherstinsky, 2020). Such a system of equations has been developed by Hopfield (1984); Hopfield and Tank (1986) in the context of associative memory, and unified under the Continuous-Time Recurrent Neural Network (CTRNN). While a continuous-time signal is more flexible than a discrete-time signal, domains such as natural language processing use symbolic encodings for low-level representations such as text, which are then translated into continuous representations through the process of reading. Early work has therefore suggested hybrid symbolic processing (Wermter, 1995) but the overwhelming success of discrete-time models in deep learning (as evidenced by current state-of-the-art sequence models) has shifted attention away from these approaches. Only recently, interest in modeling symbolic representations has gained some momentum in the context of providing more interpretability of neural networks (see Subsection 2.3.1).

From a neuroscientific perspective, temporal models can be classified into two distinct classes: those with encodings based on neuron firing *rates* vs those based on the *timing* of sparse neuron spikes (where rates are decoded from spike trains). While most current network architectures are rate-based, the latter model is reflected in Spiking Neural Networks (SNN), which allow more sparse activations and processing (Ghosh-Dastidar and Adeli, 2009). Following a hybrid approach, both models can also be incorporated into one, learning to switch between both forms of representations (Mochizuki and Shinomoto, 2014).

However, these models still do not incorporate explicit temporal representations, e.g., in the form of clocks or timestep counters. In this regard, previous work has suggested parameterized sinusoidal models to provide explicit temporal reference points for representations (Kazemi et al., 2019; Vaswani et al., 2017; Koutník et al., 2014). One benefit of sinusoid encodings over simple timestep counters is that they are numerically bounded and that the periodicity provides reoccurring reference points similar to discrete timesteps[2].

Conditional computation, and skipping in general, can be seen as a compromise between discrete-time and continuous-time models since they operate on discrete time steps but can warp and dilate them such that events stretch over different time intervals. Since skipping can be implemented in current state-of-the-art networks, it is also possible to see this as an extension of rate-based models with sparse spiking capabilities, allowing the encoding of event-based or periodic signals.

3.3 Efficient Reading

The traditional processing model of recurrent networks is an insufficient reflection of our own reading process. To give an example, capable readers do not always explicitly process text word by word and letter by letter. On the contrary, there is evidence to suggest that we process lexical units very differently from sublexical units (Joubert et al., 2004), which is thought to be even more different outside of alphabetic systems, such as with the Chinese language (Zhou et al., 2009). Therefore, reading is a highly complex and non-linear process that is thought to be guided by prediction and inference (Linderholm, 2002).

In general, the process of reading is still not fully understood on a cognitive and psychological level. Goodman (1967) has described reading as a "psycholinguistic guessing game, [...] involv[ing] an interaction between thought and language". His proposal that readers *guess* meaning based on syntactic, semantic, and graphic cues, has found widespread adoption while at the same time being subject to great criticism in a scientific debate spanning 40 years that is sometimes called "the reading wars" (Kim, 2008) or "the never-ending debate" (Smith, 1992). Goodman's promotion of a whole-language (literature-based) approach stands opposed to the idea of concentrating early language education more on phonics (Smith, 1992), i.e. the sounds of letters and words, which has been promoted by cognitive scientists such as Perfetti (1985) as an alternative model. As Castles et al. (2018) observe, favoring arguments between these two different approaches have shifted for many years, particularly in the context of defining school curricula in English-speaking countries.

Cognitive neuroscience research on dyslexia (reading disorders) was able to identify deficits in phonological processing of speech sounds early on. As Ramus (2003) points out, a phonological explanation for dyslexia stands opposed to an explanation assuming an underlying general sensorimotor deficit that can extend to other impairments such as auditory, visual, or motoric. Recent neuroimaging studies sup-

[2]We will evaluate periodic activation in Chapter 4 and Chapter 5.

port a multisensory integration in the brain that involves phonological processing, visual detection, as well as inference over multiple critical brain areas (D'Mello and Gabrieli, 2018). Bottom-up approaches to studying reading have been to investigate the role of eye movements (Rayner, 1978), whereas top-down perspectives investigate the link between reading comprehension and abilities to make inferences from text (Castles et al., 2018).

On the other end of the spectrum of readers with impairments, stand highly efficient readers who are able to read text quickly with a high level of comprehension. This process is known as *speed reading*. While it is generally accepted that speed reading involves a tradeoff between speed and accuracy (Rayner et al., 2016), the speed threshold at which reading comprehension starts to suffer is not well studied (Bell, 2001). Similarly, the process of rereading words (supported by rapid eye movements) has been shown to be critical for developing text comprehension during speed reading (Schotter et al., 2014). *Text skimming* is a form of speed reading that explicitly sets the goal of efficiency over full text comprehension. Typical applications for skimming include proofreading, scanning text for answers, or simply getting a general idea about the contents.

As part of this thesis' approach, we argue that speed reading, and more specifically skimming, is a valid and promising approach for a variety of natural language processing tasks in which extensive text comprehension is not the goal. Typical examples for this include sentiment analysis, document classification, question answering, or summarization. In these tasks, often only a small part of the input is actually relevant to produce the correct output. Most existing recurrent models, however, read every single input token sequentially, without any ability to skip over unimportant words or rereading important words in a new context. As discussed previously, this is mostly due to time being processed linearly in discretized manner in recurrent neural networks. As such, conditional computation (Section 3.5) can serve as a framework for text skimming, opening up possibilities for both better temporal representations as well as more efficient processing.

3.4 Efficient Learning

Computational requirements for deep learning models have grown significantly in recent years, often with limited performance improvements (Thompson et al., 2020). Particularly in the domain of natural language processing, training models with billions of parameters requires significant optimizations w.r.t. parallelization (Shoeybi et al., 2019) to be able to fit state-of-the-art models such as GPT-3 (Brown et al., 2020) on conventional hardware. Difficulties in deploying and reproducing such large models has increased the demand for more efficient solutions. Furthermore, studies have estimated that the development of modern systems causes a significant environmental footprint (Schwartz et al., 2020; Strubell et al., 2019), which has, e.g., led to calls to report efficiency metrics in studies (Lacoste et al., 2019). While trade-offs between accuracy and speed constitute novel research areas, the compression of pre-trained large networks for the purposes of deployment on smaller architectures, such as mobile phones, is a well-studied research topic.

A number of approaches exist to reduce the memory footprints of neural networks. *Pruning* describes the process of identifying and removing weights on a trained networks without significant loss of accuracy. The pruning ratio can be predefined or automatically discovered, e.g., with neural architecture search, based on identifying efficient network structures (Liu et al., 2019b). Low-Rank Approximation (LRA), is based on the constraint to approximate a data matrix with a lower-ranked matrix, lowering the complexity of a model. Other dimensionality reduction techniques such as Principle Component Analysis (PCA) can be seen as a special case of LRA (Markovsky and Usevich, 2012). While pruning and LRA can be applied on pre-trained models, other compression techniques need to be implemented into the training procedure itself. A popular such example is *knowledge distillation* (Hinton et al., 2015) in which a second, smaller network is trained to imitate the original, larger network. A more technical approach is to use quantization, which reduces the numerical precision of 32-bit floating point parameters to 8 or 4 bits.

These compression techniques have seen widespread adoption for large pre-trained models such as BERT (Devlin et al., 2019) for use in downstream tasks. To reduce computational requirements, BERT has, for example, been pruned (McCarley, 2019), distilled (Sanh et al., 2019), and quantized (Zafrir et al., 2019). While it is possible to compress these large models, this does not solve the initial issue that initial model training requires large amounts of computational resources. For this reason, some research has focused on identifying opportunities for more cost-effective training by using more efficient representations during the entire training process and ignoring redundancies. Such efforts to reducing computations have often concentrated on improving the sequence processing methodology by identifying subsequences (Yu et al., 2019). Using conditional computation, deep learning models have recently been significantly scaled up in capacity, i.e. parameter size, without increasing the computational budget (Lepikhin et al., 2020; Bolukbasi et al., 2017; Shazeer et al., 2017). Tao et al. (2019) even present an approach for skipping state updates in an already pre-trained network without the need for fine-tuning. This opens up additional possibilities for cases where opportunities to retrain large models are limited due to financial or computational restrictions. Overall, these initial results make conditional computation a promising research direction for more efficient models.

3.5 Conditional Computation

The core idea of conditional computation is to utilize skip mechanisms to sparsify state updates, vertical depth, sequential inputs, or latent representations. The skipping itself is controlled conditionally, sometimes based on specific constraints, sometimes controlled with free parameters.

Generally, we speak of a *skip connection* when it can be active at certain times to copy information forward in time. The concept also exists for feed-forward neural networks and is arguably most prominently used in the ResNet in the form of residual skip connections, which allow to dynamically "adjust" the depth (in terms of involved layers) of the learned representations (He et al., 2016)[3].

In the context of conditional computation, skip connections are often defined as connections that allow the skipping of state updates though this does not always have to be the case, especially on implementation-level. Some authors also define this process as "dropping" connections or entire computation paths, although this mostly just provides a different perspective on the same process of introducing the concept of inactive units or paths. We consequently use both terms interchangeably based on the context.

Conditional skipping is an inherently binary process in that states are either updated or skipped. Even a reweighing of the influence or magnitude of the update is only possible if the update is actually calculated and propagated. This leads to a binary decision problem that is unique to skip connections. One of the first clear proposals for conditional computations were put forward by Bengio (2013). He proposed it as a future research path towards scaling up computations in deep learning by designing novel architectures that are capable of sparse updates. In this specific context, the goal is to use as few computations or active states as possible while maintaining close-to baseline accuracy. However, as we will see later, conditional computation is not exclusively tied to reducing computational cost and can be used for entirely different purposes such as, e.g., eliminating redundancy, learning multiple timescales, long-term relationships, and more complex temporal dynamics.

3.5.1 Definition

Any backpropagation neural network model that has activation functions can be extended to perform conditional computation. The computation decision can most generally be expressed as a binary projection of each state h_t to a binary decision, i.e. $D : \mathbb{R} \rightarrow \{0, 1\}$ such that $D(h_t) = d_t$. The main idea of conditional computation

[3]However, it might be more accurate to use the term "shortcut connections" for residual learning since no *calculations* are actually skipped: the identity mapping and residual projection are performed simultaneously. The point of the residual identity mapping is to rather precondition the model with a reference of the input than to actually skip or fully mask any activations. Wang et al. (2018) have modified the ResNet model to actually skip convolutional blocks.

is to decide this mapping with a conditional if-then-else statement:

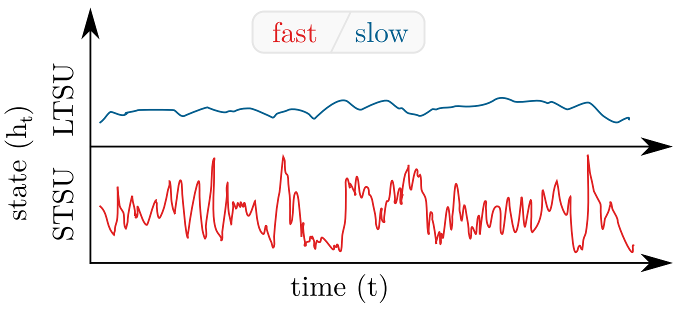
$$D(h_t) = \begin{cases} 1 & (\textit{update}) \quad \text{if } \texttt{[boolean condition]} \\ 0 & (\textit{skip}) \quad \text{else} \end{cases} \tag{3.2}$$

where *update* and *skip* are model-specific operations that exist to model both the process of a normal update and the hibernating process of skipping an update. For most neural networks, a non-update is achieved by the identity mapping $id(x) = x$ which inhibits the activation function. For recurrent connections and activations, this means that $h_t = h_{t-1}$ in most RNN models. In this context, the skip operation is therefore also sometimes called the *copy* operation. For an RNN unit h_t, a general conditional update process can then be defined as:

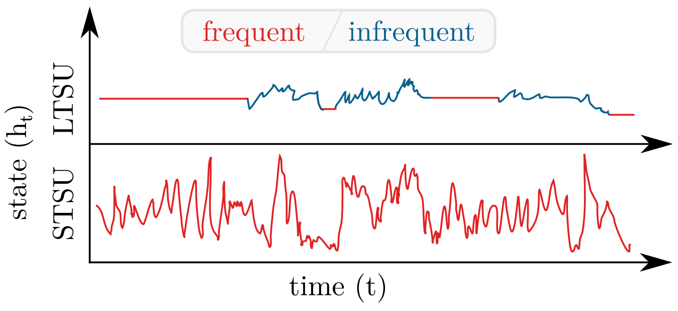
$$h_t = \begin{cases} F(x_t, h_{t-1}) & (\textit{update}) \quad \text{if } \texttt{[boolean condition]} \\ h_{t-1} & (\textit{skip}) \quad \text{else} \end{cases} \tag{3.3}$$

where $F(x_t, h_{t-1})$ describes the regular state update of the underlying RNN model. Note that redefining this update rule for a vector $\mathbf{h}_t$, leads to a slightly more restricted model in which all units of the state perform the same action synchronously, since we have moved the level at which decisions are made from units to layers. We can also see how this formulation incorporates the "trivial solution" for the slowness principle discussed in Section 2.5. As such, long-term relationships are expressed differently from slowness approaches. As skipping is based on *timing*, we can therefore modulate activation *frequency* as opposed to only the degree to which change occurs. This difference is illustrated in Figure 3.2 in the context of long- vs short-term context and timescales.

One major benefit of temporal skipping by copying previous activations is that the unaltered sequence of states leads to constant gradients along the temporal (Zhang et al., 2016). Therefore, gradients can neither vanish nor explode during skipping, instead being copied until the next timestep in which an activation occurs. The core research question in conditional computation (and in extension, this thesis) is to find a suitable `[boolean condition]`. This can be according to criteria such as minimizing calculations, improving loss, or maximizing activation sparsity. Indeed, many ideas have been put forward, often varying drastically in their approach, sometimes even being proposed from a different research focus outside of conditional computation. The following sections provide an overview of the most important methods and studies in the field. We group skipping methodologies into three main categories: i) randomized conditions (Subsection 3.5.2), ii) constrained conditions (Subsection 3.5.3), and iii) adaptive conditions (Subsection 3.5.4). While adaptive conditions are parameterized to learn the timing of skips, constrained conditions skip based on an inductive bias such as, e.g., structural or temporal constraints. Different from these network modeling approaches, random skipping views conditional computation through the lens of regularization.



(a)



(b)

Figure 3.2: (a): The slowness principle $h_t \approx h_{t-1}$ leads to slower changes in long timescale units (LTSU) than in short timescale units (STSU).
(b): Skipping with $h_t = h_{t-1}$ allows activations to be completely inactive. This allows a modulation of activation frequency for purposes of encoding long-term relationships independently from short-term context.

3.5.2 The Random Condition

The arguably simplest procedure to sample update decisions is to sample them from a random distribution. For this purpose, we can parameterize Equation 3.3 by defining a binary decision neuron $b_t \in \{0, 1\}$ which decides whether to use the update function $F(x_t, h_{t-1})$ or to swap it with the identity mapping $id(x_t, h_{t-1})$, skipping the update through the copy mechanism:

$$\begin{aligned} h_t &= b_t \cdot F(x_t, h_{t-1}) + (1 - b_t) \cdot id(x_t, h_{t-1}) \\ &= b_t \cdot F(x_t, h_{t-1}) + (1 - b_t) \cdot h_{t-1} \\ &= \begin{cases} F(x_t, h_{t-1}) & \text{if } b_t = 1 \\ h_{t-1} & \text{if } b_t = 0 \end{cases} \end{aligned} \tag{3.4}$$

While the unit b_t can be learned (see Subsection 3.5.4), it can also be sampled from a random distribution. More formally, defining b_t as a stochastic unit, we can sample it from a Bernoulli distribution, i.e. $b_t \sim Bernoulli(p)$, where $P(Bernoulli(p) = 1) = p$ and $P(Bernoulli(p) = 0) = 1 - p$.

This idea has been developed further by Krueger et al. (2017) as *zoneout*, a regularization method for RNNs. Zoneout randomly preserves the previous activations of hidden units and can therefore be seen as a recurrent variant of dropout (Srivastava et al., 2014) in which temporal connections are masked with ones (skipping temporal connections) instead of zeros (dropping feed-forward connections). Different from most skipping models, sampling from a random distribution allows to define the skip probability p explicitly. Parallel to the study by Krueger et al. (2017), zoneout has also been proposed in the context of residual networks (Singh et al., 2016) and is somewhat related to the concept of stochastic depth (Huang et al., 2016), which allows training on shorter gradient paths and inference during test time on deeper networks. However, the application focus of zoneout is to provide regularization. As such, it is similar to dropout, in that it can slightly improve baseline accuracy and prevent overfitting to a certain extent.

3.5.3 Constrained Conditions

To set up controlled skipping behavior, models can be designed with suitable inductive biases. Historically, models with skip connections can be seen as precursors to conditional computation. Skip connections offer a popular methodology to model structural biases for temporal dynamics. In fact, early work on skip connections is often formulated in the form of time delay connections for learning long-term relationships (Lin et al., 1996; Kim, 1998; Wermter et al., 1999) (see also Subsection 2.2.2). More recently, skip connections have become more relevant again in the context of deep networks (Zhang et al., 2016). While there is still ongoing research on learning long-term dependencies with skipping, addressing computational efficiency with skipping is a relatively new concept.

The Clockwork RNN (CWRNN; Koutník et al. (2014)) is one of the first models to be explicitly modeled based on Equation 3.3. It has a hidden layer that is partitioned into modules, which are activated periodically at specific timesteps, leading to different activation frequencies between units. Inactive modules simply preserve their previous hidden activations until they are triggered to activate again. These activation triggers are under periodic cycles, static, and chosen as hyperparameters. Each module has a different periodicity, and they are ordered from high to low update frequency while only low-frequency modules have a direct connection to high-frequency modules[4]. A key disadvantage of the CWRNN is that the periodic activation conditions are i) predefined hyperparameters and not learned, and ii) global for the entire sequence. This can lead to challenges when dealing with varying temporal distances between dependencies or phase shifts, which are common in real-world applications.

The CWRNN model has been extended in some related studies. Carta et al. (2020) use periodic activations but iteratively add infrequently updated modules when re-

[4]See Chapter 4 for a deeper analysis.

quired. Chung et al. (2015) generalize the CWRNN to the Gated Feedback RNN, which uses gates to learn the temporal connectivity but is not constrained to activate periodically. Similarities can also be found in the subsequently proposed WaveNet model and causal and dilated convolutions. Similar to the CWRNN's ordered modules, causal convolutions follow a specific order connecting past to present filters. Dilated convolutions, on the other hand, extend canonical convolutions by skipping pixels, spanning a larger receptive field without more adding more parameters. The concepts have been transferred back to recurrent models with dilated recurrent skip connections and exponentially increasing time delays (Chang et al., 2017). Similarly, residual connections have been adapted for recurrent networks (Yue et al., 2018). Neil et al. (2016) propose the Phased LSTM which uses an internal oscillation model to decide how long to keep activations constant. Interestingly, Yu and Liu (2018) show that similar effects to skipping can be achieved by redesigning the input sequences such that the network operates on sliced sequences.

3.5.4 Adaptive Conditions

While conditions can be stochastic or constraint-based, they can also be learned adaptively. To achieve this, representations need to be mapped to binary update decisions. The simplest form of binarization is thresholding, which is known in the form of the original perceptron and the threshold activation function $\mathbf{1}(x)_{x>\theta}$:

$$f_\theta(\mathbf{h}_t) = \begin{cases} 1 & \text{if } \mathbf{W} \cdot \mathbf{x}_t > \theta \\ 0 & \text{else} \end{cases} \tag{3.5}$$

Naturally, this function is non-differentiable as it has a gradient of 0 almost everywhere and is infinitely steep at the threshold θ. This function was therefore historically limited to single-layer perceptrons before backpropagation with continuous and smooth activation functions provided a more practical alternative (Rumelhart et al., 1986). Nevertheless, there are a number of different approaches for dealing with non-differentiable functions in neural networks that have seen some adoption in conditional computation frameworks. They all share the main idea of using the original function in the forward pass and using an estimate of its gradient during the backward pass. The three most popular gradient estimators are currently the *Straight-Through Estimator* (Bengio et al., 2013b), *REINFORCE* (Williams, 1992), and *Gumbel-Softmax* (Jang et al., 2017). Table 3.1 gives an overview over these three methods which we will be introducing in more detail next.

The Straight-Through Estimator

The idea of the Straight-Through Estimator (STE) goes back to Rosenblatt's original perceptron training algorithm (Rosenblatt, 1961) which differentiates the

Name	Gradient Estimator	Advantage	Disadvantage
Straight-Through Estimator	$\frac{\partial f_\theta(x)}{\partial x} = 1$	simple	biased
Gumbel-Softmax	$y_i = \frac{\exp\left((\log(\pi_i)+g_i)/\tau\right)}{\sum_{j=1}^{k}\exp\left((log(\pi_j)+g_j)/\tau\right)}$	relaxation	high variance (small τ), biased (large τ)
REINFORCE	$\sum_{\mathbf{b}} \nabla \pi_b(\mathbf{b})\mathcal{R}_{\mathbf{b}} \approx \frac{1}{S}\sum_{s=1}^{S} \mathbb{E}_{\mathbf{b}^s \sim \pi_b^s}[\nabla \log \pi_b^s(\mathbf{b}^s)\mathcal{R}_{\mathbf{b}}^s]$	unbiased	high variance

Table 3.1: Comparison between the three main gradient estimators used in conditional computation.

threshold activation function by using the derivative of the identity function as a proxy (Yin et al., 2019). This idea has recently been proposed in a lecture by Hinton (2013) for the purpose of backpropagating through binary activation networks, before being adopted by Bengio et al. (2013b) as a solution to backpropagating through stochastic neurons in the context of conditional computation. Outside of conditional computation, the estimator has been used as a tool for a number of different binarization or discretization problems such as quantization (Chen et al., 2019b; Razavi et al., 2019; Yin et al., 2019), end-to-end training of a chained ASR and TTS system (Tjandra et al., 2019), attentive routing in capsule networks (Ahmed and Torresani, 2019), and constructing adversarial examples (Athalye et al., 2018).

The Straight-Through Estimator differentiates a non-differentiable function $f_\theta(x)$ by treating it like the identity function:

$$\frac{\partial f_\theta(x)}{\partial x} = 1, \tag{3.6}$$

Within the framework of conditional computation, $f_\theta(x)$ can be any non-differentiable binarization function for the purpose of mapping inputs to binary update or skip decisions such as, e.g., rounding $f_{round(x)}$, threshold activation $\mathbf{1}(x)_{x>\theta}$, or the step function. In practical terms, this estimator simply passes the previous gradient through the non-differentiable layer. Yin et al. (2019) show that this estimated gradient correlates positively with the population gradient if the STE function is chosen correctly. While they indicate that there is little theoretical justification for using the identity function, its use has prevailed in practice.

The biggest advantage of STE is its simple implementation and the fact that it requires no additional computational resources. Its main disadvantage is that it is biased, i.e. potentially inaccurate, since the non-differentiable function is completely replaced by the identity during the backward pass. For this reason, Chung et al. (2017) propose the *slope annealing trick* which reduces the bias by gradually increasing the slope of the hard sigmoid function until it gets close to the step function, lowering the difference between forward and backward pass.

REINFORCE

Another possibility to train non-differentiable functions is to use reinforcement learning, and in particular, the policy-gradient method REINFORCE (Williams, 1992), which was originally proposed for training networks with stochastic units and can be integrated with backpropagation. Using REINFORCE for conditional computation requires a reinterpretation of the task as a reinforcement learning problem, designing a reward function (which can be non-differentiable), and applying the algorithm to approximate the policy with the network. The gradient can then be estimated as its updates are approximately proportional to the policy updates (Sutton et al., 2000).

REINFORCE has successfully been used for a variety of different models operating with discrete symbols, such as, e.g., hard attention (Mnih et al., 2014), natural language processing with GANs (Yu et al., 2017b), or end-to-end visually grounded dialogue systems (Strub et al., 2017). For conditional computation, REINFORCE was first proposed in Bengio et al. (2013b) and later refined in Bengio et al. (2015).

The problem formulation is as follows: assuming a set of actions consisting of binary decisions $b_i \in \{0, 1\}$, it is necessary to find the optimal model policy π_b over the sequence vector of decisions $\mathbf{b} = (b_1, \cdots, b_n)$ under some reward $\mathcal{R}_\mathbf{b}$ (Ke et al., 2018b):

$$\sum_{\mathbf{b}} \nabla \pi_b(\mathbf{b}) \mathcal{R}_\mathbf{b} = \mathbb{E}_{\mathbf{b} \sim \pi_b} [\nabla \log \pi_b(\mathbf{b}) \mathcal{R}_\mathbf{b}] \tag{3.7}$$

Since it is intractable to infer the exact policy gradient due to the high dimensionality, it is also possible to approximate the above term by running S examples (Yu et al., 2017a):

$$\sum_{\mathbf{b}} \nabla \pi_b(\mathbf{b}) \mathcal{R}_\mathbf{b} \approx \frac{1}{S} \sum_{s=1}^{S} \mathbb{E}_{\mathbf{b}^s \sim \pi_b^s} [\nabla \log \pi_b^s(\mathbf{b}^s) \mathcal{R}_\mathbf{b}^s] \tag{3.8}$$

The reward itself can then be designed by incorporating sparsity regularization, i.e. minimizing the number of computations by setting e.g. $\mathcal{R}_\mathbf{b} = -\sum_i \mathbf{b}_i$ or $\mathcal{R}_\mathbf{b} = \frac{\partial h_t}{\partial b_t}$ (Ke et al., 2018b), while others define a sparsity rate (Bengio et al., 2015). Depending on the overall setup, it can also be necessary or helpful to include another term for the prediction (Fu and Ma, 2018; Hansen et al., 2019). Even though REINFORCE has the advantage of being an unbiased estimator, it has the downside of having high variance. While some authors caution that this causes slow convergence (Jang et al., 2017), it has also been shown to lead to a faster training process in combination with variance regularization (Bengio et al., 2015).

A number of other tweaks have been suggested to lower the estimator's variance (Williams, 1992; Ke et al., 2018b; Fu and Ma, 2018).

Gumbel-Softmax

The Gumbel-Softmax distribution was independently discovered by two research teams (Maddison et al., 2017; Jang et al., 2017) (being based on earlier work about the Gumbel-Max trick (Maddison et al., 2014)) as a solution for applying the reparameterization trick (Kingma and Welling, 2014) to discrete-valued, categorical random variables. While Maddison et al. (2017) have coined the phrase "concrete distribution", it is usually called the Gumbel-Softmax distribution as proposed by Jang et al. (2017)[5].

The continuous Gumbel-Softmax distribution approximates samples from a discrete distribution. The process of drawing samples from the Gumbel distribution is defined as follows:

$$g_i \sim \text{Gumbel}(0,1) = -\log(-\log(\text{Uniform}(0,1))) \tag{3.9}$$

Assuming a categorical variable z with class probabilities $\pi_1 \cdots \pi_k$ and a k-dimensional one-hot encoding, we then draw k independent and identically distributed (i.i.d.) samples g_i from this categorical distribution, known as the Gumbel-Max trick (Jang et al., 2017):

$$z = \text{one\_hot}(\arg\max_i[\log \pi_i + g_i]) \tag{3.10}$$

The intent behind $\arg\max$ is to sample the class with the highest probability, while g_i serves to add i.i.d. noise to this process. It is important to note that the noise is injected pre-normalization, i.e. on a network's output layer y, the noise would be added to the logits rather than to the post-softmax probabilities. In other words: a continuous parameterization of a discrete distribution by means of the softmax distribution can alternatively be achieved by adding Gumbel noise to the continuous logits that need to be transformed before taking the $\arg\max$. This is called the "Gumbel-Max" trick.

Naturally, the $\arg\max$ function is not differentiable. Since the softmax function is by definition a smooth approximation of the $\arg\max$ function, it can be used in its stead to generate continuous k-dimensional sample vectors $\mathbf{y_i}$:

$$\mathbf{y_i} = \frac{\exp\Big((\log(\pi_i) + g_i)/\tau\Big)}{\sum_{j=1}^{k} \exp\Big((\log(\pi_j) + g_j)/\tau\Big)} \tag{3.11}$$

This provides the Gumbel-Softmax distribution proposed by (Maddison et al., 2017; Jang et al., 2017), which has a well-defined gradient for $\frac{\partial \mathbf{y}}{\partial \pi}$. The temperature

[5]We adapt their nomenclature for this section.

parameter τ regulates how much the Gumbel-Softmax distribution approximates the categorical distribution. For $\tau \to 0$ the representations move closer to the one-hot representation, while for larger τ the samples converge to a uniform distribution over the categories. Naturally, the closer we are to the categorical distribution ($\tau \to 0$), the higher the estimator's variance. For this reason, Jang et al. (2017) propose an annealing strategy for τ, decreasing it gradually towards zero during training (similar to the slope annealing trick in REINFORCE). According to Jang et al. (2017), it is also possible to set $\tau = 0$, using the discretized distribution in the forward pass, but the continuous approximation in the backward pass. Due to its resemblance to the estimator proposed by Bengio et al. (2013b), this is called the *Straight-Through (ST) Gumbel-Softmax estimator*.

Alternative Estimators

While these three discussed approaches are the most popular for conditional computation (compare Table 3.2), there are alternative methods and extensions for estimating gradients of non-differentiable functions. For example, REBAR provides lower-variance, unbiased gradient estimates for discrete latent variable models Tucker et al. (2017). RELAX generalizes this method by jointly optimizing the original parameters with a control variate from a surrogate network, leading to a similar result as the reparameterization trick (Grathwohl et al., 2018). REBAR and RELAX have not yet been studied in a conditional computation setting. Wang et al. (2018) combine REINFORCE with supervised pre-training to have the entire training process differentiable with continuous gates that are only treated as binary at inference time. They demonstrate this approach for skipping convolutional layers.

3.5.5 Models for Conditional Computation

While the Clockwork RNN (Koutník et al., 2014) is an early example for a model utilizing conditional activations, it is fully differentiable as skipped updates are modeled in the form of skip connections that can be implemented by masking the recurrent matrix. Similar to most models with skip connections, whether the respective computations are actually saved, is highly dependent on implementation details.

One of the first models utilizing adaptive skipping is the Skip RNN (Campos et al., 2018), which uses a binary state update gate for the decision unit b_t from Equation 3.4. It uses the candidate activation to emit an update probability that is then binarized to b_t and backpropagated with the help of the Straight-Through Estimator. By passing this activation step between timesteps, it is theoretically possible to avoid any redundant computations in cases where $b_t = 0$. To minimize computation, they propose an additional regularization term that is added to the

Estimator	Model	Applications
Straight-Trough Estimator	Skip RNN Campos et al. (2018)	IC, AL, ST
	Selective Activation RNN Hartvigsen et al. (2020)	AD/TSC
	constr. Hierarch.-Multisc. Hard-GRU Tavarone and Badino (2018)	PR
REINFORCE	Hierarchical Multiscale LSTM (Chung et al., 2017)	LM, HWSG
	LSTM-Jump (Yu et al., 2017a)	SA, DC, Q/A
	Length Adaptive Recurrent Model (Huang et al., 2017)	DC
	LSTM-Shuttle (Fu and Ma, 2018)	SA, DC, Q/A
	Focused Hierarchical RNN (Ke et al., 2018b)	ST, Q/A
	Hierarchically Structured LSTM (Zhang et al., 2018)	SA, DC
	Dynamic LSTM (Gui et al., 2019)	NER, LM, SA, ST
	Structural-Jump-LSTM (Hansen et al., 2019)	SA, DC, Q/A
Gumbel-Softmax	Variable Computation RNN (Jernite et al., 2017)	MM, LM
	Skim-RNN (Seo et al., 2018)	SA, DC, Q/A
	Leap-LSTM (Huang et al., 2019)	SA, DC
	Clipped Maxout (Lin et al., 2019)	IC
	Adaptively Scaled RNNs (Hu et al., 2019)	ST, IC, MGR, LM
	Adaptive Attention Span (Sukhbaatar et al., 2019)	LM

Table 3.2: Overview of adaptive conditional computation models, their respective gradient estimator, and the tasks they were evaluated on (LM: language modeling, SA: sentiment analysis, DC: document classification, Q/A: question answering, ST: synthetic tasks, IC: image classification, AL: action localization, AD: anomaly detection, PR: phone recognition, TSC: (continuous) time series classification, HWSG: handwriting sequence generation, MM: music modeling, MGR: music genre recognition, NER: named entity recognition).

loss and minimizes the number of state updates. Similar regularization terms can be found in most approaches reviewed in this section which generally leads to the introduction of a hyperparameter for weighting the influence of this term. A similar model using REINFORCE has been presented by Gui et al. (2019).

While the Skip RNN takes a decision at every timestep, the LSTM-Jump (Yu et al., 2017a) also has the ability to stop reading a sequence at any given time, but is additionally forced at each timestep to learn how many input tokens to skip next. With this setup, skipping occurs on a per-input basis. In order to modulate the skipping, there are a total of three hyperparameters for the number of allowed jumps, maximum jump size, and the amount of tokens read between jumps. The Structural-Jump-LSTM by Hansen et al. (2019) combines the two concepts of jumping and skipping by using two separate networks to skip over words and jump over larger structures indicated by punctuation marks. Contrary to these methods, Huang et al. (2019) take a time-averaging approach towards modeling skip decisions: their Leap-LSTM decides to skip based on a sliding window of current, previous, and future timesteps. These three encodings are integrated and fed to a softmax layer from which a binary update decision is sampled. Although they report no variance, document classification experiments show a comparable performance to the Skip RNN and Skim-RNN. In a different approach, Hu et al. (2019) learn multiscale structures by explicitly defining timescales but parameterizing them so that they can be learned with Gumbel-Softmax. Similar to the Clockwork RNN, their Adaptively Scaled RNN (ASRNN) stands out from these approaches as it reads inputs at every timestep but encodes this information based on skip connections.

It is also possible to model the skipping process in a separate network. A second “coordinator” network can then predict which states should be updated, either based on the input (Jernite et al., 2017) or on already partially pre-computed state updates (Hartvigsen et al., 2020). Similar adaptive reading architectures have been presented outside of the conditional computation framework, e.g., for varying the input windows with 3 separate networks (Huang et al., 2017) or achieving multi-turn reasoning (Shen et al., 2017). As skipping can lead to a loss of information, the idea of rereading inputs has, e.g., been modeled by Fu and Ma (2018) in their LSTM-Shuttle, which can also skip backwards in time. A different approach for this problem is to perform “soft skips”, as demonstrated by the Skim-RNN (Seo et al., 2018). Different from previous approaches such as the LSTM-Jump, this model never actually skips inputs completely. They are rather “skimmed over” by passing them to a second, smaller RNN cell while the larger main RNN cell updates based on the remaining inputs. Since this allows the Skim-RNN to store more information than strict skipping approaches, which compress the networks more heavily, it is reported to hold or even slightly improve on baseline performance for some NLP tasks. Nevertheless, while multi-network approaches have shown promising results, further studies are necessary to understand whether these gains come from introducing network modularity or from an increase in the parameter size as well as an ability to ignore ineffective constraints in one of the networks. Tao et al.

(2019) approach this question by comparing logistic regression and random forest classifiers to feed-forward neural networks in the role of predicting skip timing and find that neural networks are better at identifying important information.

As discussed in Chapter 2, the design of hierarchical, modular, multiscale structures is an important research topic for recurrent networks. As conditional computation provides a toolset to conditionally separate information processing, it fits these concepts particularly well. An early demonstration of learning multiple timescales in a hierarchical network is the Hierarchical Multiscale LSTM (HM-LSTM; Chung et al. (2017)). It addresses the fundamental problem that most hierarchical models have structural biases with regard to hierarchical boundaries (e.g., by separating character- from word-processing). In reality, hierarchical boundaries are often less clearly defined (e.g., by including syllables) and discovering these structures is a difficult problem (see also Subsection 2.2.1). The HM-LSTM uses a multiscale structure with binary boundary detectors at each layer which decide when information is passed to higher layers and how long they keep processing.

Later studies have studied the HM-LSTM for different applications and varied specifics of the architecture: Cherry et al. (2018) have found that using the model for temporal compression in character-level neural machine translation results in better performance but less compression than fixed-stride temporal pooling. Kádár et al. (2018) show in their reproduction and ablation study that the HM-LSTM can be simplified even further to achieve additional performance gains. Consequently, the model has been further simplified in the Focused Hierarchical RNN which uses less layers and adds an attention mechanism (Ke et al., 2018b). Zhang et al. (2018) propose another related variant which learns structured representations with REINFORCE instead of the Straight-Through Estimator. Tavarone and Badino (2018) investigate bidirectional layers with the HM-LSTM and propose an iterative improvement to its gating mechanism. To enforce a stricter multiscale structure, where deeper layers activate less frequently than lower ones, they constrain the activation of boundary gates to only fire if lower levels boundary gates fire as well.

Finally, it is worth noting that conditional computation is not by any means limited to the models and applications that we focus on in this thesis. For example, Lin et al. (2019) adapt maxout for conditional computation in order to conditionally rehearse training examples to reduce the effect of catastrophic forgetting. The idea has also been applied to selectively learning filters in CNNs to allow better training on new tasks (Abati et al., 2020) after conditional convolutions (Yang et al., 2019) and dynamic channel selection (Hua et al., 2019) have been proposed to increase model capacity without additional computational requirements. Other studies have presented alternatives to skipping depth with residual connections (He et al., 2016) by investigating skipping convolutional layers to adaptively scale depth in deep networks (Wang et al., 2018). With regard to Transformer networks, Sukhbaatar et al. (2019) have proposed to learn adaptive attention spans using a similar mask as the one used by Jernite et al. (2017). Correia et al. (2019) build on this work by introducing adaptively sparse attention in Transformers. These studies suggest a broad applicability of conditional computation to a number of different research problems, tasks, and model designs.

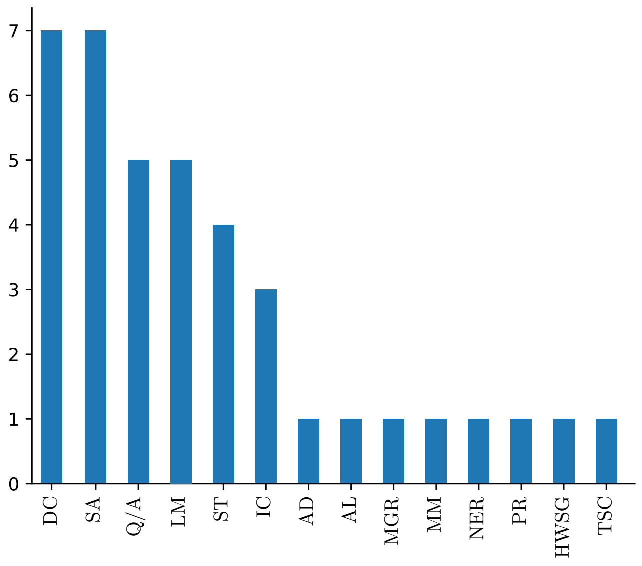
3.5.6 Adaptive Computation Time

While conditional computation focuses on modeling the *timing of computations* for state updates, Adaptive Computation Time (ACT; Graves (2016)) is concerned with the *computation time* spent updating states. As such, ACT can be seen as an alternative approach to model salience based on computation focus that is different from the local skipping of CC and global weighting of attention.

This is realized by allowing the model to perform an arbitrary number of "sub-timesteps" per input $\mathbf{x}_t$ at timestep t. During these, the state is repeatedly activated in a loop. The update process is eventually stopped by a halting probability which grows over time by the successive accumulation of a sigmoidal halting unit's output.

Initial experiments on character-level language modeling have shown that the model increases computation at word boundaries, i.e. whenever the prediction uncertainty is high. Further experiments integrating ACT with alternating attention (Sordoni et al., 2016) have yielded mixed results showing i) that additional computation doesn't always yield better performance and ii) that the model can benefit from setting the number of computation steps as a fixed hyperparameter (Neumann et al., 2016). This has been partly reproduced in an ablation study on ACT by Fojo et al. (2018) using the parity and addition tasks. Their results indicate that the repetition mechanism has a much larger effect than the adaptively determined differences in computation time. Since these differences are the only tool in ACT to have more expressive power in the learned representations, the potential of ACT to develop salience-based temporal representations seems limited.

While subsequent studies with ACT have shown some promise in the visual domain (Figurnov et al., 2017, 2018), studies with RNNs have mostly been confined to evaluations on synthetic tasks and it is therefore currently an open question whether ACT can provide an alternative approach to conditional computation in RNNs. Similarly, current work is limited to a layer-wise application of ACT. Therefore, it remains an open question as to how ACT would benefit in modular or hierarchical architectures. Since ponder time seems to correlate with prediction uncertainty (Graves, 2016), the methodology holds some promise for identifying feature complexity and badly trained samples. As such, it would be interesting to study whether ACT can be used for continual learning and multi-task learning. Recently, there is regained interested in ACT as its addition has been shown to make Transformers Turing-complete (Universal Transformer; Dehghani et al. (2019)), closing an important theoretical gap between Transformers and RNNs.



DC: document classification
SA: sentiment analysis
QA: question answering
LM: language modeling
ST: synthetic tasks
IC: image classification
AD: anomaly detection
AL: action localization
MGR: music genre recognition
MM: music modeling
NER: named entity recognition
PR: phone recognition
HWSG: handwriting sequence generation
TSC: time series classification

Figure 3.3: Application focus of selected studies on modeling with conditional computation (N=17).

3.6 Challenges and Open Questions

Applications where a classification or prediction is based on only a small part of the input stand to benefit the most from conditional computation. In these scenarios, the majority of the input is either uncorrelated to class labels or simply redundant, acting as noise. A popular method of choice for such tasks are attention models (Galassi et al., 2020). Since attention is modeled with softmax distributions, they share the same requirement, as heavily increasing the softmax dimensions causes the mean probability to be lower without ever leading to actual sparsity in the vector.

A number of tasks that fulfill these requirements can be found in the NLP domain, especially in question answering, sentiment analysis, document classification, and language modeling. Figure 3.3 shows a strong tendency in existing research for adaptive conditions to focus on these tasks for benchmarking. The capability to learn when to update could open up promising directions for a number of current research problems, such as dealing with event-driven extremely long sequences (e.g. raw audio data) without hand-crafted features, having adaptive computation times (Subsection 3.5.6), learning multiple timescales and their boundaries (Subsection 2.2.2), or integrating sensory data under asynchronous sampling conditions (Neil et al., 2016). The latter is particularly important for neuromorphic architectures who process information event-based (Iyer et al., 2018). In conclusion, future applications of conditional computation include, among others, event-based prediction, change-point detection, sparse coding, pruning, compression, modeling long-term memory, improving computational efficiency, providing better interpretability.

Which constraints can be imposed on update conditions successfully, will influence future applications of conditional computation. This constitutes an important research question that is investigated in current research as well as in this thesis. The majority of past suggestions either provide fully adaptive models or design structural biases in the temporal dimension (e.g. timescales) or the spatial dimension (e.g. hierarchies and modularity).

A strong tendency for all skipping models is to lose baseline accuracy with increased skipping activity. While this is a natural consequence of heavy compression, the specific compression ratios at which models can remain lossless have not been studied well yet and require further research. Similarly, it was only recently that research has shown that many benchmarks require significantly smaller model capacities than previously expected and that the effects of increasing model sizes beyond these capacities is not well understood (Li et al., 2018; Belkin et al., 2019). In contrast to approaches that try to maximize computational efficiency are a few that try to improve representation quality, e.g., by regularization such as in zoneout, which can even lead to slight improvements in accuracy. Nevertheless, balancing the trade-off between accuracy and skip rates remains one of the most important challenges of conditional computation. Therefore, this thesis addresses this challenge and contributes to answering the question of how we can minimize or avoid these trade-offs.

3.7 Chapter Summary

In this chapter, we have discussed difficulties of perceiving and structuring time which have translated to computational models of time. In this context, we have argued for the importance of efficient sequence processing methodologies, giving the human reading process as a practical example. Conditional computation provides a framework for modeling sequential processing with conditional skips, preventing networks from constantly updating each of their states in every timestep, regardless of the importance of the processed input tokens and context.

As part of our review, we have presented stochastic, constrained, and adaptive approaches. The current challenges in designing effective skip mechanisms for recurrent networks directly tie into our main research questions about i) identifying suitable constraints and inductive biases for skipping, and ii) increasing skip rates in networks without negatively affecting model performance. In the following chapter, we will provide further experimental motivation by comparing the effects of skipping in modular architectures to more traditional RNNs using leaky activation and shortcut connections.

Part II

Periodic Activation

Chapter 4

From Leaky to Periodic Activation

While traditional recurrent models regulate the scales of temporal memory dynamics through leaky memory or dynamic memory decay, the conditional computation framework suggests sparse activation by means of skipping state updates. Both the traditional processing framework and conditional computation promote certain design concepts such as modularity and specialization of units (see Subsection 2.3.2). In this chapter, we will provide an experimental evaluation of different recurrent architectures implementing modular designs. The goal of this is to identify critical design concepts for skipping networks and to get a first experimental insight into how encodings from conditional updates differ from traditional update models. At the same time, we investigate the advantages of stronger inductive biases for skip constraints. For this purpose, we choose to compare the SRN as a baseline to leaky memory models (see Subsection 2.2.2) and the LSTM which implements dynamic memory decay. As a representative for conditional computation, we use the Clockwork RNN, a model capable of skipping based on periodic update constraints. The models are compared according to their ability to learn multiple timescales in two different tasks, namely sequence generation and learning Embedded Reber Grammar.

4.1 Motivation

Until recently RNNs were mainly of theoretical interest as their initially perceived shortcomings proved too severe to be used in complex applications. One deficiency that has been reported early on is the vanishing gradient problem (Bengio et al.,

Sections 4.1-4.3.3 of this chapter have previously been published in a preliminary study and are based on Alpay et al. (2016).

1994). When RNNs are trained with backpropagation, error signals over time vanish exponentially in RNNs. This has led to multiple highly specialized architectures such as the Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber (1997)). Their success has sparked a renewed research interest in RNNs, which has led to a number of recently proposed RNN architectures, including those that try to improve control over the self-organization of temporal dynamics by learning on multiple timescales. However, as these novel approaches have not yet been rigorously compared, the fundamental principles that allow the capturing of dynamics on different timescales are still unknown.

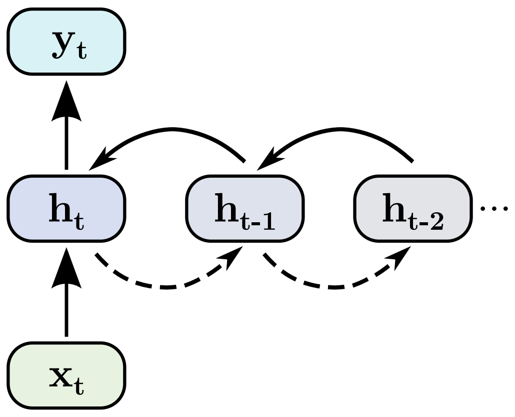
In this chapter, we therefore aim at contributing to the following research question: what are key concepts that allow RNNs to build long-term memory and learn on multiple timescales? We approach this question by investigating the Clockwork RNN (CWRNN; Koutník et al. (2014)), which has been shown to allow emergence of multiple timescales by restricting update frequencies to temporal constraints. A different method with the same effect is the use of leakage and hysteresis parameters that constrain the amount of change within a system between time steps. The concept of leakage is most popularly used in the Echo State Network (ESN; Jaeger et al. (2007)) but has also been shown to improve the Simple Recurrent Network (SRN; Bengio et al. (2013a)). A related concept can be found in the Recurrent Plausibility Network (RPN; Wermter et al. (1999)) which introduces a related hysteresis parameter φ to perform time-averaging. It also has shortcut connections, which provide shorter error propagation paths for the temporal context layers. Shortcuts have been shown to allow better training in very deep networks (Pascanu et al., 2014). Both shortcuts and leaky units are used in the Structurally Constrained Recurrent Network (SCRN; Mikolov et al. (2015)) that additionally partitions its layer into modules, similarly to the CWRNN.

As the RPN, SCRN, and CWRNN share similar architectural concepts such as leakage, shortcuts, and partitioning the hidden layer into modules, their investigation is of particular interest for studying the effect of these concepts on the self-organization of the temporal dynamics. We evaluate these architectures on sequence generation and prediction tasks, using the SRN and the LSTM as a baseline. Even though the LSTM has no specific time scaling mechanism, it is included in the experiments due to its reported ability to capture long-term dependencies.

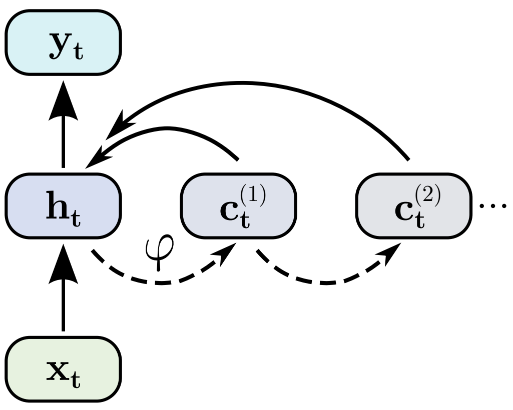
4.2 Investigated Recurrent Neural Network Architectures

4.2.1 Simple Recurrent Network

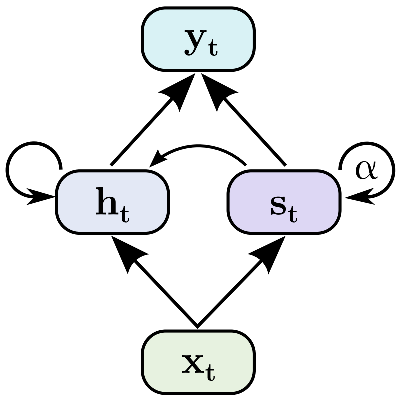
The Simple Recurrent Network (SRN) is one of the earliest RNN architectures and has originally been proposed by Elman (1990).



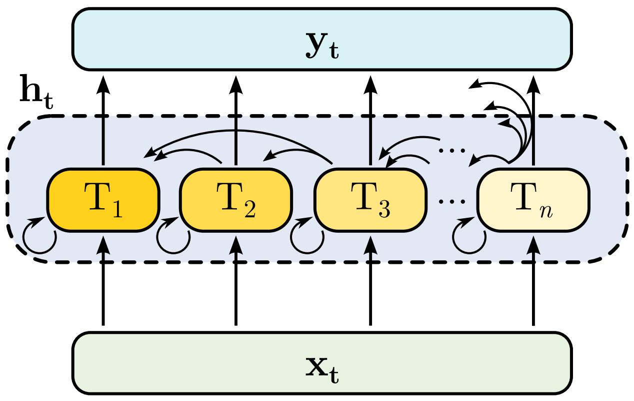
(a) Simple Recurrent Network (SRN)



(b) Recurrent Plausibility Network (RPN)



(c) Structurally Constrained Recurrent Network (SCRN)



(d) Clockwork RNN (CWRNN)

Figure 4.1: Comparison of investigated RNN architectures. Figure **(a)** shows an SRN unfolded in time. The RPN **(b)** extends the SRN with its temporal shortcuts and the hysteresis φ. In case of a deep RPN, each vertical layer $\mathbf{h}_t^{(n)}$ can have its own hysteresis value φ_n. The SCRN **(c)** has an additional layer $\mathbf{s}_t$ that learns slower than in $\mathbf{h}_t$ due to its high leakage $\alpha = 0.95$. The modules T_k of the CWRNN **(d)** are sorted by increasing numbers from left to right and are only updated for t mod $T_i = 0$.

To recap our introduction from Subsection 2.1.1, a recurrent hidden layer is activated by an activation function f as follows (compare Figure 4.1a):

$$\mathbf{h}_t = f(\mathbf{x}_t\ \mathbf{W}_{xh} + \mathbf{h}_{t-1}\ \mathbf{W}_{hh} + \mathbf{b}_h) \tag{4.1}$$

where $\mathbf{x}_t$ is the input, $\mathbf{W}_{xh}$ and $\mathbf{W}_{hh}$ define the respective weight matrices of the input and recurrent layers, and $\mathbf{b}_h$ a bias[1].

[1]We will generally omit the bias for the purposes of brevity and clarity in this thesis. We assume that every model used in this thesis has a corresponding bias vector added to each of its weight matrices unless explicitly stated otherwise.

4.2.2 Recurrent Plausibility Network

The Recurrent Plausibility Network (RPN) was originally developed to learn and represent semantic relationships while disambiguating contextual relationships (Wermter, 1995). It is based on the state of an unfolded SRN during truncated BPTT, i.e. each hidden layer $\mathbf{h}$ has its own set of m context layers $\mathbf{c}^{(k)}$ ($k \in \{1, ..., m\}$) which store past activations. The main difference to an unfolded SRN is the use of temporal shortcut connections for shorter context propagation paths, making vanishing or exploding gradients less likely (compare Figure 4.1b). For time step t, the units of the hidden layer $\mathbf{h}$ are activated as follows:

$$\mathbf{h}_t = f\left(\mathbf{x}_t\ \mathbf{W}_{xh} + \sum_{k=1}^{m} \mathbf{c}_{t-1}^{(m)}\ \mathbf{W}_{mh}\right), \tag{4.2}$$

where the vector $\mathbf{c}$ denotes the context layers, that are activated by shifting their contents with $\mathbf{c}_t^{(m-1)} = \mathbf{c}_{t-1}^{(m)}$. The respective context activation for units in $\mathbf{c}^{(m)}$ is further constrained under the hysteresis parameter φ (Arevian and Panchev, 2007):

$$\mathbf{c}_{\mathbf{t}}^{(k)} = \begin{cases} (1-\varphi)\cdot \mathbf{h}_{\mathbf{t-1}} + \varphi \cdot \mathbf{c}_{\mathbf{t-1}}^{(k)} & \text{if } k = 1, \\ \mathbf{c}_{\mathbf{t-1}}^{(k-1)} & \text{otherwise} \end{cases} \tag{4.3}$$

The hysteresis mechanism allows for a finer adjustment of context memory than in the SRN. Rather than accumulating past activations in a single feedback loop, the network is able to specifically learn the contribution between specific time frames due to the temporal shortcuts.

4.2.3 Structurally Constrained Recurrent Network

The Structurally Constrained Recurrent Network (SCRN) was recently proposed by Mikolov et al. (2015). The motivation behind the architecture is to achieve specialization of hidden layers by partitioning them into parallel “modules” that operate independently and under distinct temporal constraints. This theoretically allows to train on multiple timescales. While the left path in the SCRN equals a SRN with a regular hidden layer $\mathbf{h}_t$, the additional module $\mathbf{s}_t$ has units with different temporal characteristics (compare Figure 4.1c). It is initialized with the recurrent identity matrix and its updates constrained by a leakage parameter $\alpha \in [0, 1]$. The authors set this leakage to 0.95, causing the states to change on a much slower scale than in $\mathbf{h}_t$. Similarly to the RPN, this architecture makes use of shortcut connections ($\mathbf{W}_{sh}$) that allow $\mathbf{h}_t$ to access long-term context which is learned in $\mathbf{s}_t$. The update rules of the SCRN are as follows:

$$\mathbf{s}_t = (1-\alpha)\mathbf{W}_{xs}\ \mathbf{x}_t + \alpha\ \mathbf{s}_{t-1}, \tag{4.4}$$

$$\mathbf{h}_t = f_h(\mathbf{W}_{sh}\ \mathbf{s}_t + \mathbf{W}_{xh}\ \mathbf{x}_t + \mathbf{W}_{hh}\ \mathbf{h}_{t-1}), \tag{4.5}$$

$$\mathbf{y}_t = f_y(\mathbf{W}_{hy}\ \mathbf{h}_t + \mathbf{W}_{sy}\ \mathbf{s}_t), \tag{4.6}$$

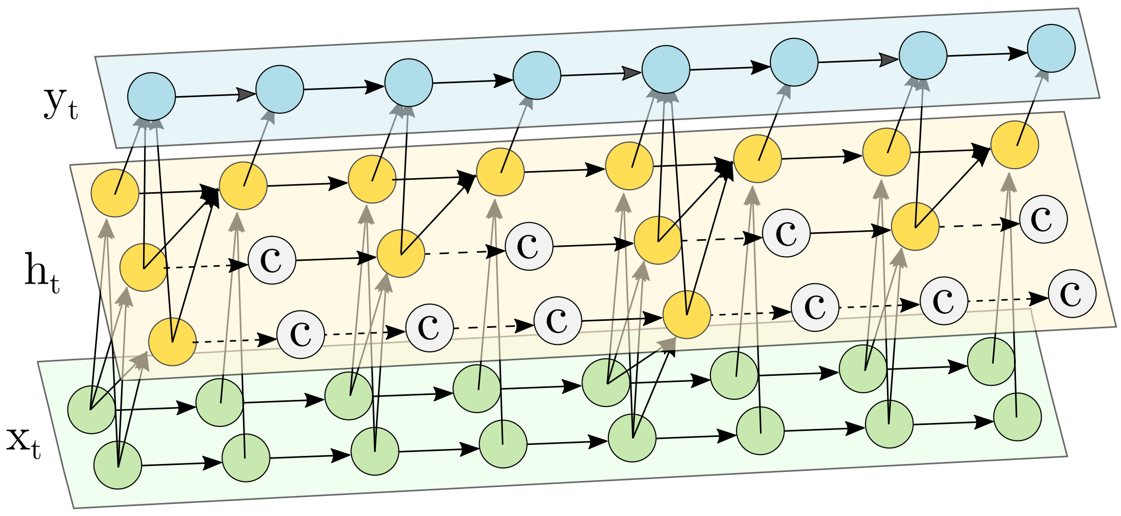


Figure 4.2: Unrolled Clockwork RNN. Units have varying update frequencies, i.e. they are updated periodically and otherwise copied between timesteps (indicated by the dotted arrows and the copied units marked with ⓒ). The modules are sorted by update frequency (here, from top to bottom: 1, 2, 4) and the unidirectional connection scheme only allows weights from modules with long periods to those with short periods.

where f_h and f_y are the respective activation functions for the hidden and output layers.

4.2.4 Clockwork Recurrent Neural Network

The discussed idea of partitioning the hidden layer into parallel modules with distinct temporal properties can also be found in the Clockwork Recurrent Neural Network (CWRNN). While the CWRNN is also a modular multiscaling architecture, it is different from the other models in that it uses no leaky memory model but conditional computation. Therefore, the main difference is that multiple timescales are not achieved by varying leakage but rather an external clock that determines *when* a module gets updated. This means that a module m is only updated if its clock period T_m satisfies the criterion $t \bmod T_m = 0$. Otherwise, the module is inactive in which case the previous activation $\mathbf{h}_{t-1}^{(m)}$ gets copied over:

$$\mathbf{h}_t^{(m)} = \begin{cases} f\left(\mathbf{x}_t\ \mathbf{W}_{xm} + \sum_{l=m}^{n} \mathbf{h}_{t-1}^{(l)}\ \mathbf{W}_{lm}\right) & \text{if } t \bmod T_m = 0, \\ \mathbf{h}_{t-1}^{(m)} & \text{otherwise} \end{cases} \tag{4.7}$$

An additional constraint is that $T_l > T_m$ for $l < m$, i.e. the modules are ordered by increasing numbers from left to right (compare Figure 4.1d). Therefore, modules on the left are updated more frequently than those on the right. Consequently, modules with greater periods (on the right) will self-organize slower and to long-term dependencies while those with small periods (on the left) change more often, focusing on short-term dependencies. The interplay between units with different update frequencies is additionally illustrated in Figure 4.2. To achieve a stricter separation and modularity between different temporal dynamics, modules within each layer are only connected from right to left, i.e. infrequently updating modules connect to more frequently updating modules[2].

It is important to highlight that, compared to the other models, the CWRNN has a strong inductive bias on periodic processing. As part of our experiments, we will evaluate the model separately on a task that reflects this bias (learning sinusoid sequences) and one which does not (learning grammar rules).

4.2.5 Long Short-Term Memory

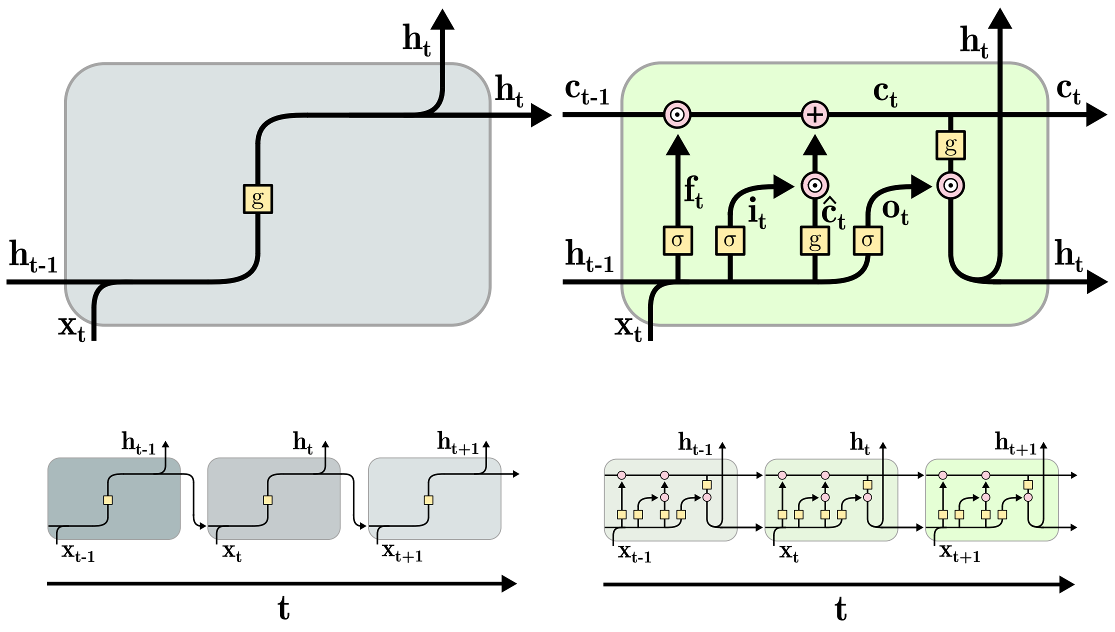
The Long Short-Term Memory (LSTM) model has seen widespread use in recent years and has lead to an increased success in using RNNs in natural language processing applications and other real-world tasks (Greff et al., 2017). The main advantage of the LSTM over the SRN is its capability to flexibly handle long-term dependencies without suffering from vanishing gradients. In simple terms, this is achieved by having a linear activation (which gives a constant gradient that cannot vanish) on the main state vector, called the *cell state* $\mathbf{c}_t$. Since we still need nonlinearities, we outsource this task to special "memory gates" with sigmoid activation. These gates regulate the influence of neighboring and past states. While leaky activation models memory as exponentially decaying over time, memory gates are fully trainable and do therefore not have a static decay function. Instead, the memory decay is fully based on the training data.

The LSTM has three different gates, namely the forget gate $\mathbf{f}_t$, the output gate $\mathbf{o}_t$, and the input gate $\mathbf{i}_t$. Combined with the cell state $\mathbf{c}_t$, they form the basis for calculating the hidden state $\mathbf{h}_t$ as follows (Greff et al., 2017):

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \tag{4.8}$$
$$\hat{\mathbf{c}}_t = g(\mathbf{W}_{xc} \cdot \mathbf{x}_t + \mathbf{W}_{hc} \cdot \mathbf{h}_{t-1}) \tag{4.9}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{c}_t) \tag{4.10}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} \cdot \mathbf{x}_t + \mathbf{W}_{hi} \cdot \mathbf{h}_{t-1}) \tag{4.11}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} \cdot \mathbf{x}_t + \mathbf{W}_{hf} \cdot \mathbf{h}_{t-1}) \tag{4.12}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{W}_{ho} \cdot \mathbf{h}_{t-1}) \tag{4.13}$$

where σ denotes the sigmoid activation function, $g(x)$ a freely selectable activation function (but typically $\tanh(x)$), $\hat{\mathbf{c}}_t$ is the candidate activation for the cell state, $\mathbf{x}_t$ the input vector, and $\odot$ the dot product.

[2]In the context of the CWRNN, we denote this connectivity scheme as "unidirectional" and will investigate its role closer in Chapter 5 by comparing this to "bidirectional" connectivity schemes.



(a) SRN unit **(b)** LSTM memory cell

Figure 4.3: Comparison between SRN and LSTM cells. Top: Memory cell. Bottom: Memory cell unrolled backwards in time. Yellow blocks indicate transformations, pink circles operations (addition or dot product). Inputs and outputs on horizontal axis ($\mathbf{c}_t, \mathbf{h}_t$) symbolize the memory cell recurrently passing values to itself between timesteps, whereas those on the vertical axis denote the regular lateral cell input ($\mathbf{x}_t$) and output ($\mathbf{h}_t$). LSTM figure adapted from Olah (2015).

The entire block of computation is called the *memory cell* and can be visually compared against a regular SRN unit in Figure 4.3. Since $\mathbf{c}_t$ is an internal state that is not exposed to other cells, only $\mathbf{h}_t$ is passed to other cells (top right in the figure) and serves as the external representation. The cell state $\mathbf{c}_t$ and $\mathbf{h}_t$ are however saved and referenced recurrently between two consecutive timesteps.

4.3 Comparative Experiments

All five architectures, the SRN, RPN, SCRN, CWRNN, and LSTM have been evaluated on two tasks; sequence generation of a sinusoid wave and sequence prediction of words created by Embedded Reber Grammar. They have been trained with RMSProp, which divides the current gradient by a sliding average of recent gradients (Graves, 2013). Based on preliminary experiments, momentum was empirically set to 0.9 and the networks trained for a maximum number of 5000 epochs using early stopping. Weights were initialized using normalized initialization, sampling from $\mathcal{N}(0, 1/\sqrt{n+m})$ where n is the number of incoming and m the number of outgoing weights in the respective layer (Glorot and Bengio, 2010). Linear and non-linear activation (tanh) were explored. The forget gate bias was initialized with a higher value of 2 to avoid initial forgetting (Jozefowicz et al., 2015). All other hyperparameters were set empirically for each network and task. Each setup was run 100 times with different random initializations.

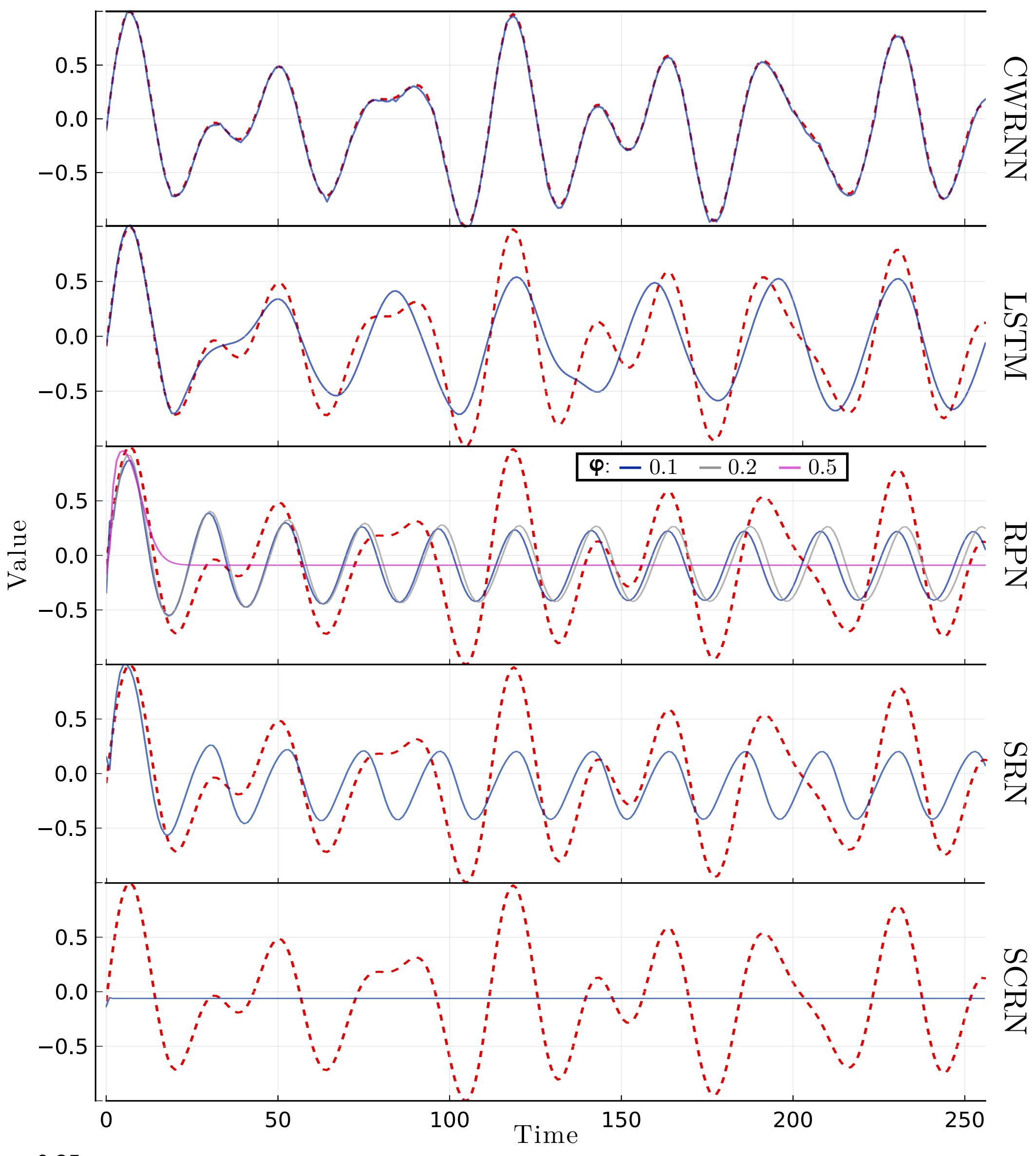
4.3.1 Sequence Generation

In the first task, the networks have to learn how to generate a target sequence. They receive no input while a single sequence is sequentially presented as the target. This sequence of length 256 is a composition of three different sine waves, normalized to $[-1, 1]$. A single output unit y_t encodes the respective sequence value at time step t. All networks were trained to minimize the mean squared error (MSE) with a learning rate of $\gamma = 10^{-4}$ and 64 hidden units. For the RPN, a context width $m = 5$ and $m = 15$ was explored with hysteresis values of $\varphi \in \{0.1, 0.2, 0.5\}$. Two variants of the SCRN were trained: i) a constant leakage of $\alpha = 0.95$ and ii) an adaptive leakage α_t that is trained as described in Mikolov et al. (2015). For the CWRNN, 8 equally sized modules with clock periods growing by the powers of 2 ($P_1 = (1, 2, 4, 8, 16, 32, 64, 128)$) are compared with a more coarse setup of 4 modules with the periods $P_2 = (1, 4, 16, 64)$.

The results for the best networks are depicted in Figure 4.4. The CWRNN generates the most accurate sequences, which indicates an ability to capture the underlying subfrequencies, learning multiple timescales. It was also found that the investigated clock-timings P_1 (8 modules) and P_2 (4 modules) perform equally well. The SRN on the other hand merely captures the most dominant subfrequency of the sequence while the LSTM gives a sliding average. The SCRN always converges to the mean, being the only network which seems to be completely unable to learn this task. Similar to the SRN, the RPN is able to capture only one subfrequency. For the tested φ values, only 0.1 and 0.2 lead to convergence that is not located around the mean. There is also a slight difference that can be observed between these values: increasing φ from 0.1 to 0.2 causes an increasing phase shift, i.e. the prediction gets increasingly delayed over time. This effect can be explained by the fact that the temporal context, which is time-averaged by the hysteresis, will span a larger time window with growing hysteresis values.

4.3.2 Embedded Reber Grammar

In the second task, the networks are trained to sequentially predict the next symbol produced by Embedded Reber Grammar (ERG). The ERG is a well-known test for RNNs. It is more difficult to learn than Reber grammar and an SRN cannot be trained with BPTT to learn the grammar due to the presence of long-term dependencies.



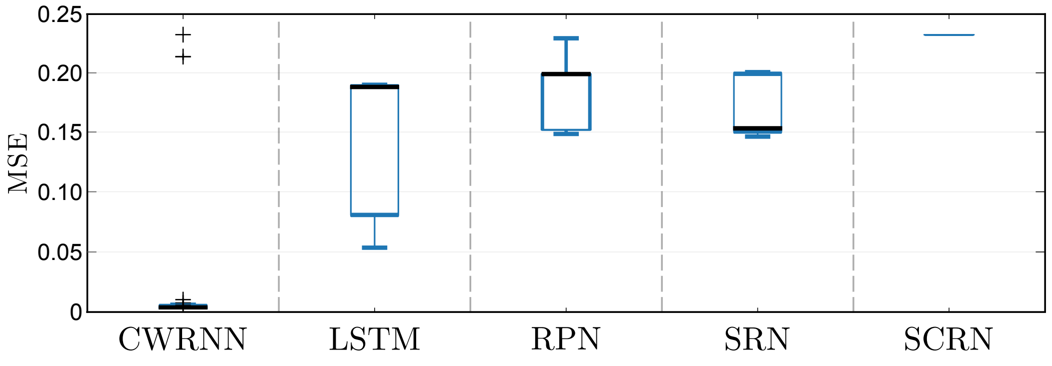


Figure 4.4: Top: Sequences with the lowest MSE for the best trials. Generated sequences (solid lines) are plotted against the target sequence (dotted, red line). Bottom: MSE for each network. Boxes show 25% and 75% quartiles as well as the median (black line). The shown best trials were achieved with $\varphi = 0.2, m = 15$ for the RPN and $P_2 = (1, 4, 16, 64)$ for the CWRNN (P_1 produced nearly identical results).

The Embedded Reber Grammar is defined as follows:

```
S → btRte | bpRpe      A → sA | x      C → xBD | s
R → btACe | bpBDe      B → tB | v      D → pC | v
```

Similar to the sinusoid functions, ERG sequences are partially periodic as specific tokens in the state automaton can be visited repeatedly and cause recurring subsequences. However, due to the recurrent nature of the grammar, these subsequences have different lengths and are therefore discontinued at each recurrence level. These irregularities should make the task slightly more difficult than the previous sequence generation task.

We randomly generate two different sets with respective sequence lengths of 20 and 30. Both data sets consist of 250 sequences and are further split into 60% training, 20% test, and 20% validation sets for cross validation. Each symbol is encoded with a feature vector of size 7 (1 unit per symbol), while softmax activation in the output layer yields the symbol probabilities. The minimized loss function is the Kullback-Leibler divergence (Kullback, 1997). For all networks, the number of hidden units was set to 15. For the SCRN, a learning rate of $\gamma = 0.01$ was found to be optimal, whereas $\gamma = 10^{-4}$ worked best for the other architectures. The CWRNN's hidden layer was partitioned into 5 modules with the periods $P = (1, 2, 4, 8, 12)$. All other hyperparameters are set as in the first task.

The results for the best trials are depicted in Figure 4.5. When trained with sequences of length 20, the SCRN with $\alpha = 0.95$ emerges as the best performing architecture, whereas the CWRNN seems to have the most difficulties. The LSTM shows an average accuracy, while the RPN seems to be less prone to bad initialization than the SRN. Especially for longer sequences, a large number of SRNs yield considerably more prediction errors than all other networks, which in turn share a similar overall performance.

4.3.3 Experiment Summary

Our results show that parallel hidden layers, which learn under different temporal constraints can lead to an emergence of multiple timescales in RNNs. Furthermore, shared weights in the form of shortcut connections (such as in the SCRN and CWRNN) allow units, which self-organize to short-term context, to take long-term dependencies into account from specialized units that operate on a larger timescale. While the SCRN achieves this by means of leakage, the CWRNN utilizes periodic module activations. For the sequence generation task, the CWRNN was the only architecture to learn the decomposition of the trained sinusoid wave into all its subfrequencies. All other networks converged to the mean or a single subfrequency. This suggests that the CWRNN is able to store the entire sequence in the memory of the clocked modules, although it has half as many parameters as the SRN (Koutník et al., 2014). For the second task, the complete opposite can be observed;

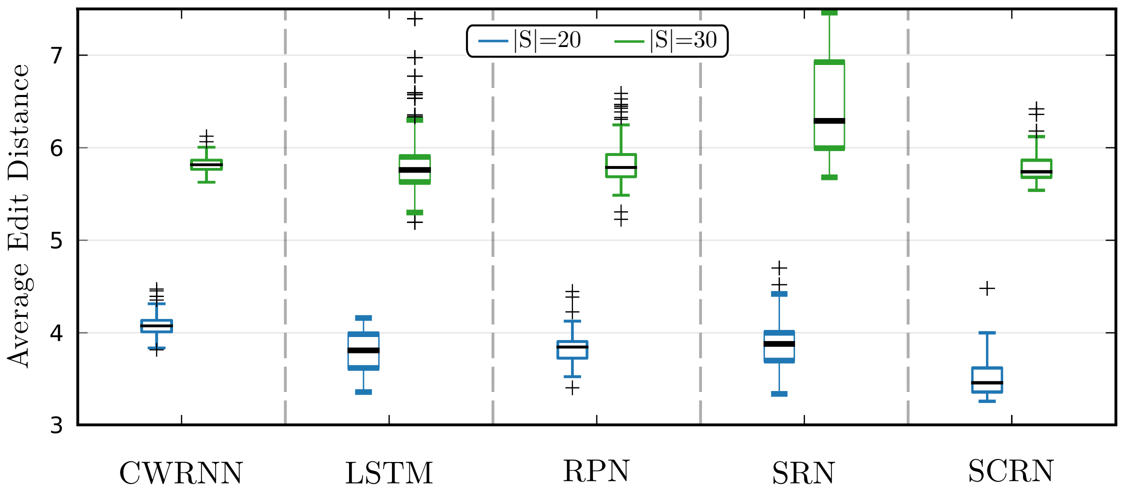


Figure 4.5: Average edit distances (number of wrongly predicted symbols) for sequences of length $|S| = 20$ (left, blue) and $|S| = 30$ (right, green). Boxes show 25% and 75% quartiles as well as the median (black line). The best RPN trials were achieved with $\varphi = 0.2$.

the SCRN is able to outperform all other networks for sequence lengths of 20 while the CWRNN has difficulties.

Our findings suggest that the SCRN and RPN seem to work better for discrete, symbolic long-term decisions while the CWRNN is better at decomposing real-valued signals. Partitioning hidden layers with distinct temporal constraints has shown to be a viable method to capture different timescales.

In the following section, we will explore these hypotheses more deeply by first analyzing the CWRNN representations for the sinusoid sequence learning task Section 4.4, before investigating in Chapter 5 whether our initial observations on solving discrete tasks with the CWRNN hold true if we scale up the task complexity to the language modeling domain.

4.4 Analyzing the Clockwork RNN Representations

As we have previously hypothesized that periodic activations cause the CWRNN to outperform the other networks on the sinusoid sequence learning task, we now aim to inspect them more closely. Therefore, we visualize the internal states of the CWRNN and compare the differences in the encodings between the different modules after (Subsection 4.4.1) and during (Subsection 4.4.3) convergence. We continue in a similar experimental setup though we use a slightly more noisy sequence to enforce slightly more diversity between the modules.

4.4.1 Activation Maps

In order to gain a deeper understanding about the representations learned by the CWRNN, we visualize the encodings in activation maps. For this purpose, we plot the activations of the hidden layer over time. We do this after the network loss has converged close to 0 and show the activations that result from feeding the learned target sequence as input.

The CWRNN networks that we showcase predict the next timestep in sequences of length 256, using $|\mathbf{h}| = 64$ hidden units, 8 modules with $P = (1, 2, 4, 8, 16, 32, 64, 128))$, and $\tanh(\cdot)$ activation. We additionally compare the activations maps of the CWRNN to those of the SRN which serves as the most related network architecture, and the LSTM, which is generally considered to be the state-of-the-art RNN model and, therefore, a sensible baseline.

Investigating the resulting visualizations, we have found marginal differences when slightly adjusting the learning rates, which would occasionally result in the overall activation level (minimum and maximum activation) to moderately go up or down for the entire sequence. Regardless of this, we were able to observe the same patterns but use this observation to only showcase networks where we have set the common hyperparameters between the different network architectures to the same values. In particular, we train with Adam optimization (Kingma and Ba, 2015) and an initial learning rate of 0.01.

We give a representative example visualization in Figure 4.6, which highlights the different network representations. As can be seen, all three networks have very distinct representations though they share some common elements. The SRN representations are highly redundant and most oscillations are on a short timescale. While some long-term oscillations can be identified, they happen less frequently than in the LSTM. This is unsurprising as the LSTM is specifically designed to deal with long-term dependencies. Its prevailing activation pattern is caused by tracing the learned dominant subfrequency of the target sequence (Figure 4.6: circle A). Consequently, these units have its maximum activation maxima in the periodically appearing descent of the valley between the target sequence's maxima and minima - a segment which is sufficient to reconstruct the entire subfrequency.

The same pattern can also be found in the CWRNN. Primarily, the module T_8, which activates every 8 timesteps, has picked up this frequency, indicating that this module is mostly responsible for reconstructing the dominant subfrequency. We have found that this frequency is learned early on during training as it provides a rough fit to the target sequence (for more on this, see Appendix Section A.2). Modules with smaller periods lead to very similar short-term activation patterns as in the other networks (Figure 4.6: circle B) and are most likely used to reconstruct the other target subfrequencies with smaller periodicity. Different from both the SRN and the LSTM, we can observe how the global activation level slightly rises throughout the sequences in the CWRNN. This could potentially be caused by the

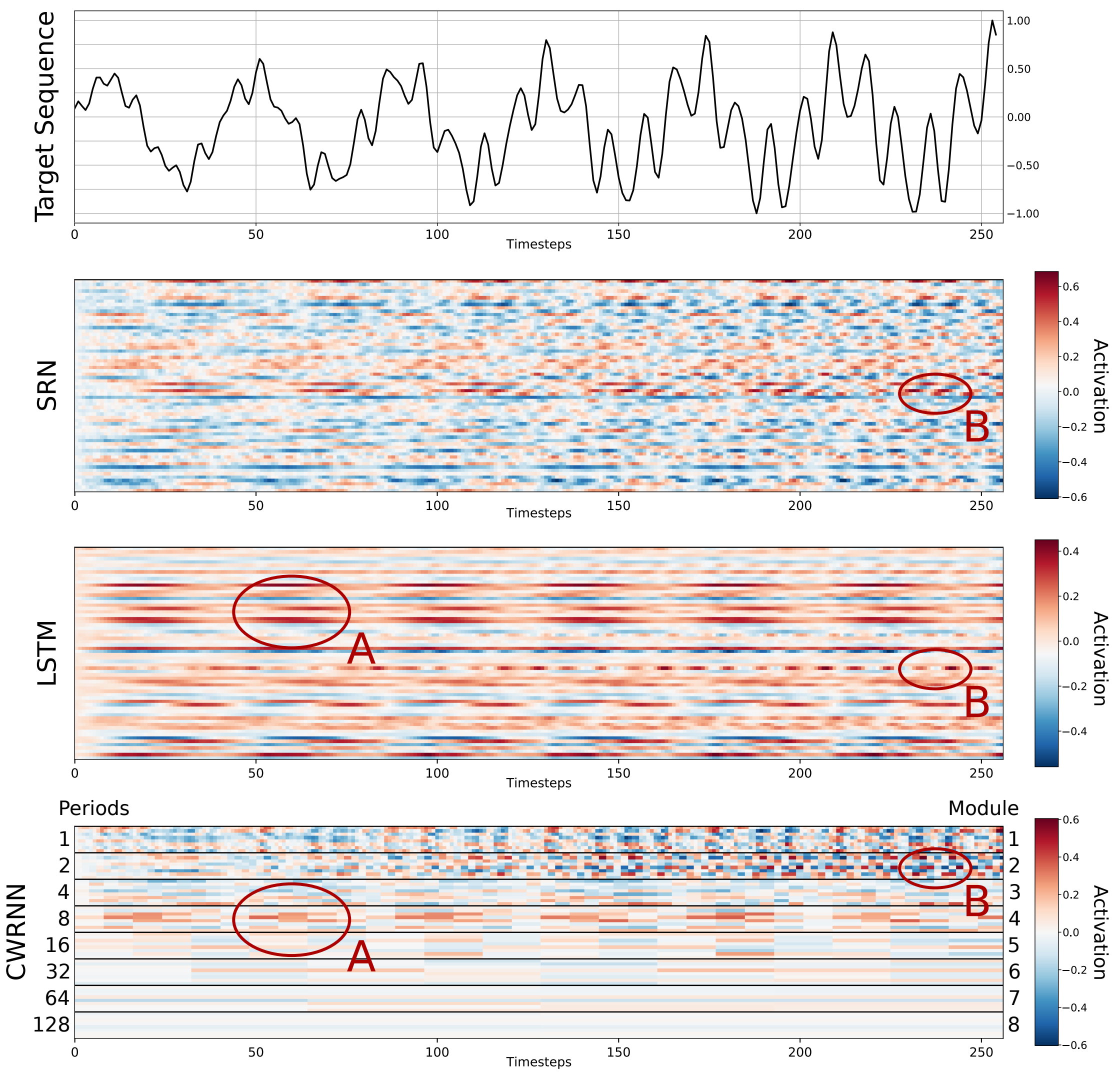


Figure 4.6: Comparison between the hidden activation maps of the SRN, LSTM, and CWRNN after successful convergence. The network encodings of the target sequence differ significantly from each other with regard to the redundancies in the encoding: the SRN is highly redundant and oscillates within short time ranges, the LSTM has smoother activations, focusing on long-term dependencies, the CWRNN does both, depending on the module, but offers more sparse activations overall. A: Traces from the dominant subfrequency of the target sequence. B: Periodic short-term features.

unidirectional connections from modules with larger activation periods who have little to contribute at the beginning of the sequence where they haven't updated yet. We have found some support for this explanation in that we were able to observe for converged networks that the majority of fitting errors occurred in the initial few timesteps of the sequence. This hypothesis is further supported by a different study which was able to show fewer mistakes in earlier parts of the sequence if the influence of slower modules is reduced (Carta et al., 2020).

It can also be seen that CWRNN modules with larger periods yield very neutral activation patterns and therefore seem to contribute little to the overall sequence. This effect has previously been observed in other conditional computation approaches with submodules and has generally been described as *module collapse* (Kirsch et al., 2018). It can be the outcome if some modules are prematurely optimized over others due to them having simpler constraints and having a more immediate effect on the loss function. This kind of greedy self-reinforcement can lead to a lack of diversity in the modules. However, the under-utilization of some modules does not necessarily have to always be a cause of module collapse. If the underlying bias of the constraints does not fit the data bias, it is natural to assume that avoiding these modules will all in all lead to a better error. In our case, the modules with very low activation frequencies are simply unable to contribute much as there are no events that occur every 64-128 timesteps that would be useful for contributing to a smaller error.

While this reduces overall redundancy in the network representations, visualizing the CWRNN module activations allows us to quickly infer by how many units the network is roughly over-parameterized by, whereas this information is more difficult to draw from the other networks due as their feature adaptation is not localized by any predetermined module structure. On the whole, this structure makes trained CWRNNs more easy to interpret than the SRN or LSTM.

4.4.2 Recurrent Activation Trajectories

We now remove the barely utilized modules with large periods and focus our attention on the same network configuration but with the shorter periods $P = (1, 2, 4, 8)$. In these networks, we were able to observe how each module is successfully able to specialize on different subfrequencies of the target sequence T. To highlight this behavior more distinctly, we take the recurrent activations $\mathbf{h}_t$ of the entire sequence and apply Principle Component Analysis (PCA) to project each module's unit into a 2-dimensional plane. The resulting visualization shows the activation trajectories with regard to the principle components of the temporal activation space.

Figure 4.7 illustrates an example visualization obtained after convergence of the previously discussed network. As can be seen, the trajectories for each module are distinct and reflect the underlying three sinusoid subfrequencies of the learned sequence. This is additionally indicated by the directional change in the activation space of each module's trajectory. The overall trajectory movement also mirrors the overall trained sinusoid sequence. Together with the previously discussed activation maps, this gives us additional evidence towards our hypothesis that the different activation periods of the CWRNN allow the network to learn distinct features, in particular the different subfrequencies of the trained sequence.

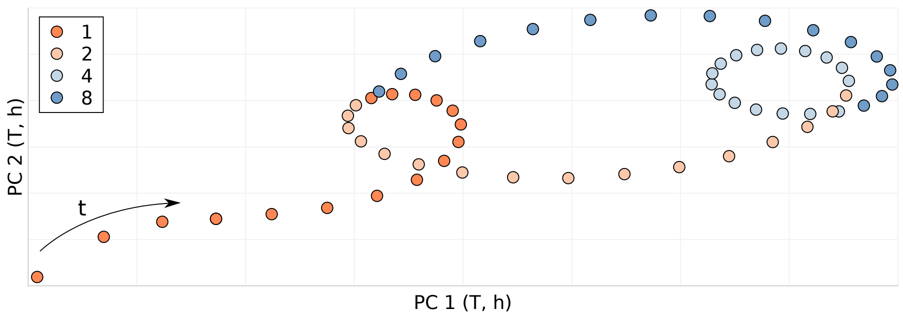


Figure 4.7: Module-wise trajectories of the CWRNN after learning a sinusoid sequence (Principle components of recurrent layer activations $\mathbf{h}_t$ along sequence T).

4.4.3 Recurrent Weight Trajectory Plots

So far, we have seen how the CWRNN modules can lead to the development of distinct representations and functions. To better understand how these representations arise, we visualize their evolution during the training procedure. For this purpose, we develop visualizations which allow us to display how the recurrent weights develop from epoch to epoch. We record this information by storing all weight values for each epoch in the high-dimensional tensor $(\mathbf{W}_{hh}, T)$ with $T = [1, \ldots, \text{nr\_epochs}]$. To project it into a two-dimensional space, we use Principal Component Analysis (Gabella et al., 2020) and use a sequential color gradient to display each weights trajectory over time. We name the resulting plots *recurrent weight trajectory plots*.

Figure 4.8 shows different weight trajectory plots for the SRN, LSTM, and CWRNN. To successfully interpret the plots, it is important to consider the color gradient. The dark-blue data points signify the initial epochs (1-150), whereas the green-yellow data points show the later epochs (150-300). As we initialize with small random weights, the weights start close to the origin. The more data points can be found around this initial region, the longer the network has stayed near this configuration. If this region is more sparse, the weights have more quickly evolved away from its initial values. As can be seen, the green-yellow tails are generally a small section of each trajectory, signaling only small changes, and therefore how weights have started to converge to local minima between epoch 150 and 300.

The figure shows that all three networks are being optimized quite differently from each other. For the SRN, we can observe a spiral motion which covers the weight space almost evenly (see Figure 4.8a). Most trajectories linger around the origin for the initial epochs but then quickly jump further away in the weight space with a high momentum as the direction is only changed slightly before convergence. On the one hand, this suggests that the SRN weight trajectories move to sufficient local optima early on. On the other hand, we can see that all weight trajectories have the same motion and differ only in their direction. Assuming that specialization

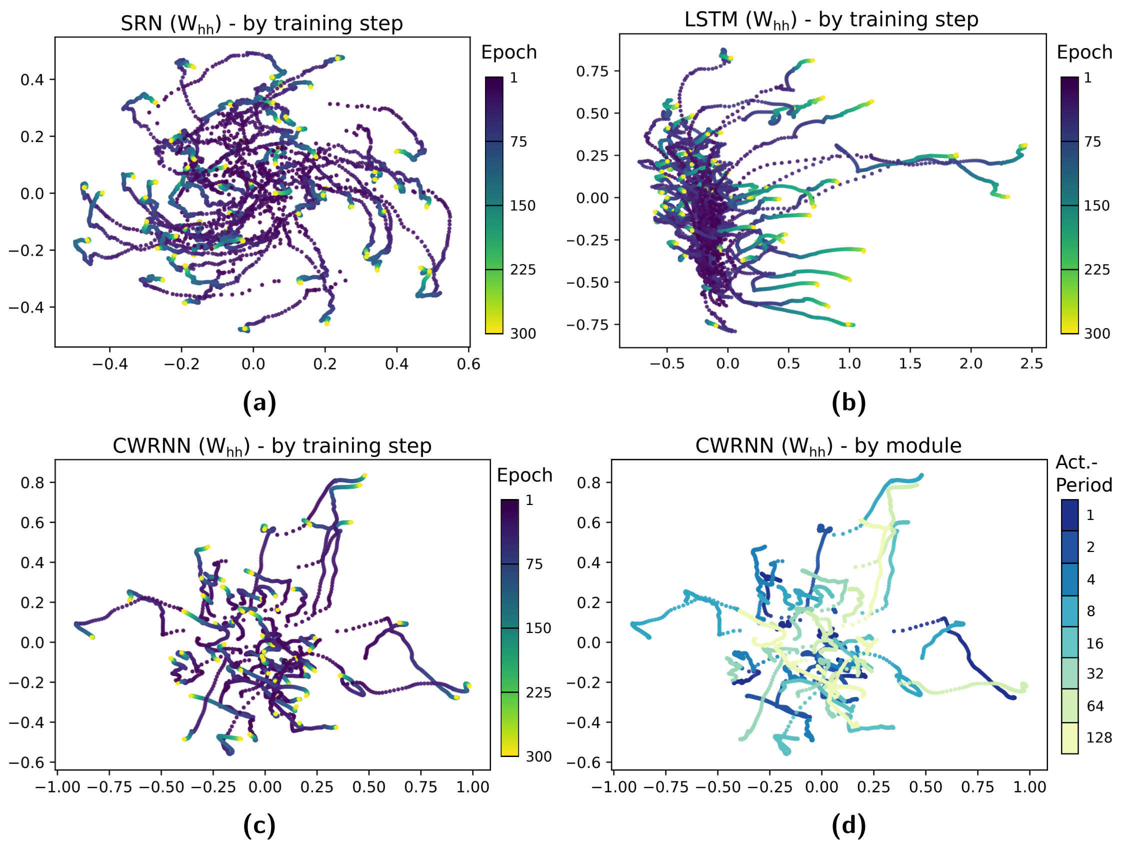


Figure 4.8: Recurrent weight trajectory plots ($|\mathbf{h}| = 64$, 300 epochs) for the recurrent weights of the **(a)** SRN, **(b)** LSTM, and **(c)** CWRNN. Each point represents the principle components of a unit's weight at the end of a training epoch. Training epochs are colored in ascending order from violet to yellow. The resulting trajectories illustrate each unit's trajectory trough the parameter space during training. The shorter the yellow tails, the longer the time that the parameter has converged to the respective region. **(d)**: CWRNN colored by module (dark: frequent; light: infrequent activation).

between units causes differences on the optimization surface, this would indicate a lower diversity between the weights of the SRN.

Most LSTM weights (Figure 4.8b) do not evolve much from their initial value, instead covering a large portion of this region. Similarly to the SRN, this hints towards most of the LSTM weights serving the same function, causing low diversity and large redundancy in the representations (which would confirm our observations from the activation maps in Subsection 4.4.1). Nevertheless, a small number of weights can be seen to shoot further away to other regions where they slowly converge to local minima, indicating a different type of purpose for these weights.

Similar to the previously shown activation maps, the CWRNN weights (Figure 4.8c) indicate a combination between the optimization behavior of the

SRN and LSTM. While the SRN's spiral shape is more condensed into a star shape, a large number of weights can be seen to branch out. Different from the other networks, the CWRNN weights move away almost immediately from the initialized values. While this slight "overshooting" often results in subsequent directional changes in the parameter space, they settle to their local minima in a very stable manner. Unsurprisingly, comparing the weights colored by module (Figure 4.8d), we can see from the length of the trajectories that weights from modules with high activation frequencies are often optimized more quickly than those with large periods. The respective module trajectories are mostly distinct from each other even though they share some local minima. This indicates that modules can have overlapping purposes, likely caused by the *modulo* operation allowing simultaneous updates between modules and therefore some degree of codependencies. The degree to which codependencies occur, however is significantly lower than in the LSTM or SRN, again demonstrating a larger diversity between the CWRNN representations.

Appendix Section A.1 provides additional visualizations which condense our weight trajectory plots by further clustering the trajectories depending on their closeness in the parameter space and projecting them on topological maps using the Mapper algorithm as outlined in (Gabella et al., 2020). The resulting maps show which areas of the parameter space are mostly preferred during training and how many outliers exist outside of these areas. The resulting visualizations (shown in Figure A.2) give further evidence of a larger diversity in the visited parameter space of the CWRNN when compared to the SRN or LSTM.

4.5 Chapter Summary

In this chapter, we have explored various design concepts such as modularity that allow the emergence of multiple timescales and long-term memory in RNNs. Leaky and periodic activations have been investigated together with partitioning hidden layers into modules and using shortcut connections by comparing a number of architectures on the tasks of sequence generation and learning Embedded Reber Grammar.

Our results give additional evidence to similar studies that partitioning hidden layers with distinct temporal constraints is a viable method to capture different timescales. As part of our analysis of activation patterns, we found evidence that modular processing leads to more diversity in the representations.

The CWRNN has a strong inductive bias due to the periodic update constraint. Evaluating it on a sinusoidal task, which reflects this bias, has shown that such a bias outperforms other, more general models. On the other hand, in the absence of such a bias in the data, other processing models seem to perform better. Furthermore, our initial results indicate that the CWRNN is better at decomposing real-valued signals than discrete, symbolic sequences. In the following Chapter 5, we will explore this more closely by scaling up our task complexity to the language modeling domain. In our analysis of activation maps, we were also able to see indications for the emergence of patterns that can be observed in both the LSTM and SRN. As we have found evidence that suggests how the LSTM leads to more weight specialization than the SRN, we will also proceed with implementing memory gates into the CWRNN to observe whether this improves the original architecture with regard to how skipped inputs are handled.

Chapter 5

Gated Skipping with the Clockwork LSTM

In the previous chapter, we have outlined how the modules of the CWRNN can lead to more diverse representations that can serve distinct functions, such as e.g., operating on different timescales by focusing on different subfrequencies of the learned target sequence. We were also able to observe difficulties for learning Embedded Reber Grammar, which suggests a downside for discrete non-periodic tasks. However, this capability is important for using the model in more complex real-world tasks, particularly those related to natural language processing (NLP). In order to investigate these capabilities more thoroughly, we investigate whether periodic activations can be successfully used in language modeling, a typical NLP benchmark for RNNs.

Current state of the art language models in the area of natural language processing are typically based on gated recurrent models such as the LSTM or the GRU. Since the CWRNN implements the skipping mechanism on the SRN, a more basic RNN architecture, we hypothesize that it has an inherent performance disadvantage. Furthermore, we will show that the introduction of memory gates allows us to perform more fine-grained skipping where, similar to text skimming, some form of information can be absorbed during skipping, even if the state itself does not fully update. To this end, we propose the CWLSTM - an extension of the CWRNN with memory gates. As we will show, this novel architecture improves heavily on the CWRNN in terms of performance and scalability. As part of our approach, we propose multiple possibilities for integrating the memory gates into the existing skipping mechanism and present different variants for a CWLSTM.

5.1 The Clockwork LSTM

CWLSTM Skip Targets The LSTM has two state variables, i.e. the cell state $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t$. It also has three gates, namely the forget gate $\mathbf{f}_t$, output gate $\mathbf{o}_t$, and input gate $\mathbf{i}_t$ (see Subsection 4.2.5). Consequently, the objective to "skip the state" is more ambiguous than in the SRN and we are left with the possibility to either skip the hidden state $\mathbf{h}_t$, the cell state $\mathbf{c}_t$, or both simultaneously. We define the set of states or gates that the network skips on as *skip targets* (ST). Different skip targets lead to slightly different CWLSTM models. Generalizing all different skip targets into a single update equation for each module k in the CWLSTM, we get the following:

$$\tau_t^{(m)} = \begin{cases} \tau_t^{(m)} = f\left(\mathbf{x}_t\ \mathbf{W}_{xm} + \sum_{l=m}^{n} \tau_{(t-1)}^{(l)}\ \mathbf{W}_{lm}\right) & \text{if } t \bmod T_m = 0 \\ \tau_{(t-1)}^{(m)} & \text{otherwise,} \end{cases} \tag{5.1}$$

where $\tau \in \{\mathbf{c}, \mathbf{h}, \mathbf{i}\}$ denotes the chosen skip target state or gate. Depending on the chosen skip target, there are additional resulting changes to the update of the LSTM memory cells (see Equations 4.8-4.13) within the CWLSTM:

Skip target c By skipping the cell state, the cell state candidate $\hat{\mathbf{c}}_t$ is never used. This causes the CWLSTM to disregard the current forget and input gates and to only exploit the output gate $\mathbf{o}_t$ to update the hidden state $\mathbf{h}_t$:

$$\mathbf{c}_t = \mathbf{c}_{t-1} \tag{5.2}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{c}_{t-1}) \tag{5.3}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{W}_{ho} \cdot \mathbf{h}_{t-1}) \tag{5.4}$$

Skip target h By skipping $\mathbf{h}$, we completely ignore the values of all states and gates for the current timestep:

$$\mathbf{h}_t = \mathbf{h}_{t-1} \tag{5.5}$$

However, the cell state $\mathbf{c}_t$ is actively computed in the background and passed on to timestep $t+1$, saving unused information for the future and opening up the possibility to later refer to these events even though we skipped over them in this timestep. Consequently, the following updates are performed for possible use in timestep $t+1$ and later:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \tag{5.6}$$
$$\hat{\mathbf{c}}_t = g(\mathbf{W}_{xc} \cdot \mathbf{x}_t + \mathbf{W}_{hc} \cdot \mathbf{h}_{t-1}) \tag{5.7}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} \cdot \mathbf{x}_t + \mathbf{W}_{hi} \cdot \mathbf{h}_{t-1}) \tag{5.8}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} \cdot \mathbf{x}_t + \mathbf{W}_{hf} \cdot \mathbf{h}_{t-1}) \tag{5.9}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{W}_{ho} \cdot \mathbf{h}_{t-1}) \tag{5.10}$$

Skip targets c and h To avoid any background computation on $\mathbf{c}_t$ and force a complete skip that wipes the memory of all traces of the current timestep, we can simultaneously skip $\mathbf{c}_t$ and $\mathbf{h}_t$, leading to no other operations:

$$\mathbf{c}_t = \mathbf{c}_{t-1} \tag{5.11}$$
$$\mathbf{h}_t = \mathbf{h}_{t-1} \tag{5.12}$$

Skip target i In addition to state skipping, we also explore the idea of skipping the input gate $\tau = \mathbf{i}$ which avoids updates on the input gate but at the same time allows adaptive memory decay through the forget gate. This allows the state to change during skipped time intervals, albeit independent of the received input (see also Subsection 6.3.2):

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \\ &= \mathbf{f}_t \odot \mathbf{c}_{t-1}, \end{aligned} \tag{5.13}$$

which leads to the following state update equation for the hidden state $\mathbf{h}_t$:

$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{f}_t \odot \mathbf{c}_{t-1}) \tag{5.14}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} \cdot \mathbf{x}_t + \mathbf{W}_{hf} \cdot \mathbf{h}_{t-1}) \tag{5.15}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{W}_{ho} \cdot \mathbf{h}_{t-1}) \tag{5.16}$$

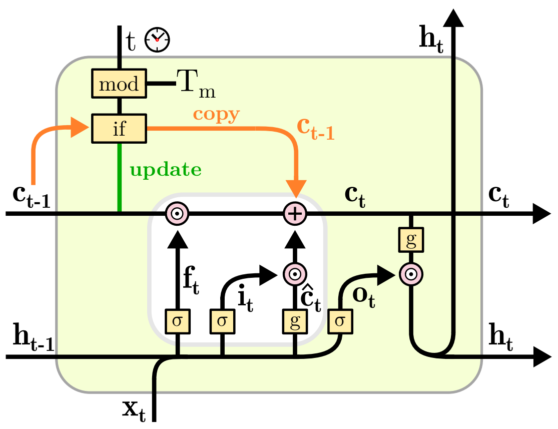
The differences between the CWLSTM variants are visualized in Figure 5.1. Skipping both $\mathbf{c}_t$ and $\mathbf{h}_t$ leads to the most amount of skipped computations, whereas only skipping $\mathbf{h}_t$ leads to the least amount of saved computations since the cell state $\mathbf{c}_t$ requires all gates.

5.2 Language Modeling Ablation Study

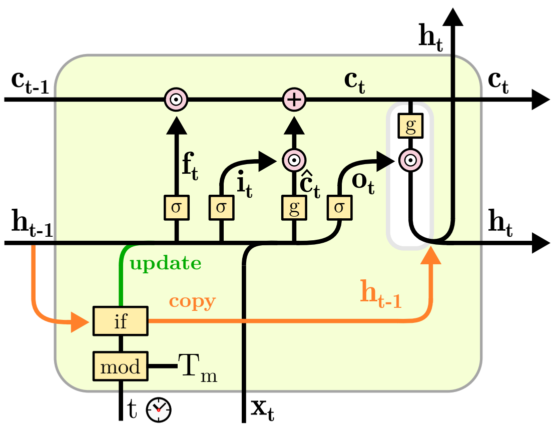
So far, we have seen that the CWRNN performs exceptionally well on continuous sequences with periodic subsequences. On the other hand, it lags behind other networks such as the LSTM when it comes to learning Extended Reber Grammar, where it has to deal with discrete symbols and a less obvious periodicity in the sequences.

However, this capability is exceptionally important to apply the CWRNN successfully on real-world natural language processing tasks. Consequently, we scale up the task complexity by investigating both the CWRNN and the newly introduced CWLSTM on language modeling (see Subsection 3.1.1) with news articles. In addition, we run an ablation study on the networks to clarify how much the design choices of the Clockwork networks contribute to the overall performance.

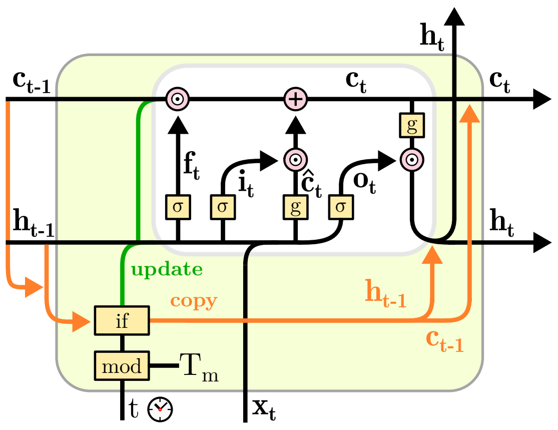
The CWRNN handling of the periodic context during language modeling is illustrated with an example in Figure 5.2. As can already be seen, the distance of



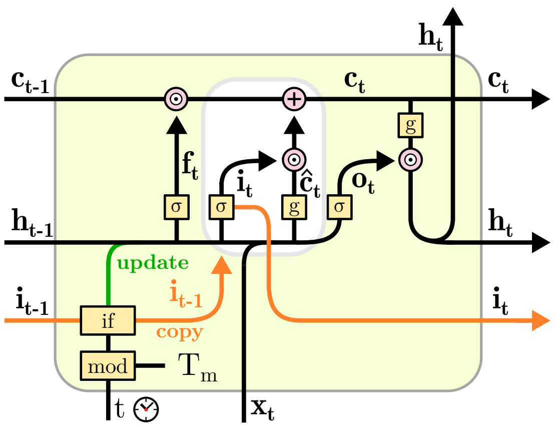
(a) CWLSTM: Skip Target $\mathbf{c}_t$



(b) CWLSTM: Skip Target $\mathbf{h}_t$



(c) CWLSTM: Skip Targets $\mathbf{c}_t$ and $\mathbf{h}_t$



(d) CWLSTM: Skip Target $\mathbf{i}_t$

Figure 5.1: Comparison of all 4 CWLSTM variants. Orange arrows denote copy operations that are executed whenever a skip is triggered. In this case, the computational flow is rerouted and computations marked inside the white boxes are skipped or disregarded. If no skip is executed, the computational flow is routed along the green paths to resume the original update pathways.

words that share semantic meaning or any relationship is typically not periodic. This makes it very questionable whether this type of architecture is suitable for the given task. However, it can be argued that the CWRNN has the same capabilities of a regular RNN as we always use a 1-module which processes the input normally without any skipping. The question is therefore whether the additional modules that periodically capture (deterministic but seemingly random) long-term dependencies can at least lead to a regularization effect or whether they add too much noise to the 1-module to give any comparable performance to a baseline RNN.

We argue that the regularization potential comes from some evident similarities to the widely used dropout and zoneout regularization methods: each Clockwork

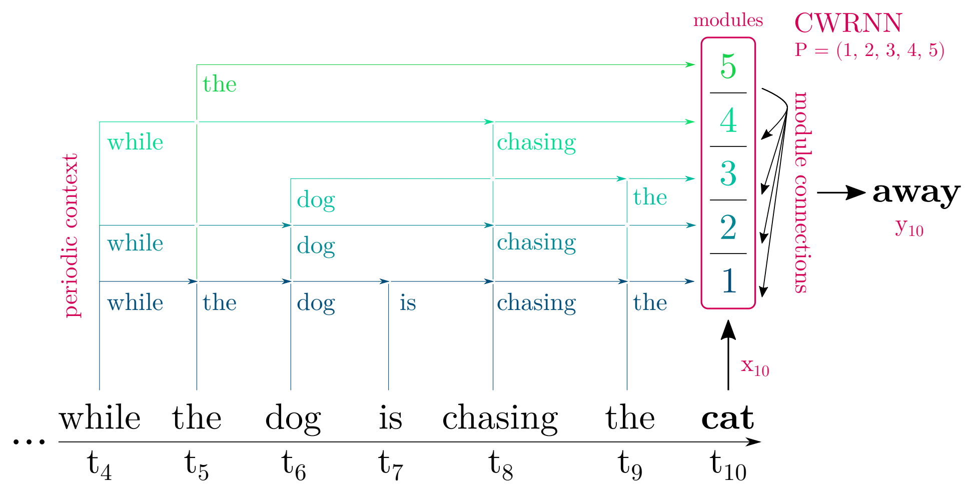


Figure 5.2: Illustrative example of a CWRNN with the five modules $P = (1, 2, 3, 4, 5)$, receiving the word **cat** at timestep t_{10} and predicting the next output word (**away**) based on each module's periodic context window.

module receives only a predetermined part of the input sequence. While this is not truly random, the periodic patterns are assumed to have no correlation to relationships in the underlying data. The modules could therefore be considered as an ensemble in which each submodel receives an incomplete sequence with "dropped out" inputs. At the same time, the architecture can also be interpreted to "drop in" seemingly random (though deterministic) skip connections for each module. Since CWRNNs and zoneout regularization have in common their basic skipping principles, CWRNNs can also be seen as a form of deterministic zoneout with a stronger bias on long-distance skipping (based on the manually set periods).

5.2.1 Experimental Setup

For our ablation study, we train on the Penn Treebank corpus, a relatively small but well-known benchmark. The corpus includes a collection of 2499 news articles sampled from the Wall Street Journal (WSJ). We utilize the widely-used preprocessed version by Mikolov (2012) where 930k words are used as training data, 74k words as validation data, and 82k words as test data. The most frequent 10k training words are used as the vocabulary while the remaining words are replaced by an `<unk>` token.

On this dataset, we run a grid search for the CWRNN and CWLSTM, as well as the regular LSTM and SRN as their respective baselines. We set the same gradient descent parameters for all networks to enable a fair comparison under the same training conditions, as the practice of individually tuning each architecture separately with grid search has previously been criticized and is assumed to have caused severe comparability issues between different neural language modeling studies working on the same datasets (Melis et al., 2018). We therefore opt to train each architecture ourselves and avoid comparisons to other architectures from studies with different or incompletely reported experimental setups.

After initial tests, we find that a learning rate of 0.1 leads to comparable convergence times on all networks using Stochastic Gradient Descent (SGD) optimization and early stopping with holdout cross validation. Each language model has a context window of 35 steps, a batch size of 20, and a very light L^2 regularization scaled by 10^{-7}. Moreover, we explore if the networks require traditional non-linear transformations in the form of tanh activation (which can cause vanishing gradients) or whether they can also handle linear activation.

Typically, LSTMs are trained with more than $|\mathbf{h}| = 1000$ hidden units on this dataset to achieve a satisfying baseline performance (Zilly et al., 2017; Zaremba et al., 2015). Since we're interested in how well each clocking mechanism compresses information, we also explore significantly smaller networks that, to our knowledge, have so far not been reported ($|\mathbf{h}| \in \{50, 100, 250, 500, 1000, 1500\}$). As part of the ablation study, we train all Clockwork networks with their unidirectional connection scheme which connects modules only from right to left, but also compare this to a "bidirectional"[1] connection scheme which connects units in both directions as is usual in regular RNNs. This allows us to study the impact of the unique Clockwork connectivity scheme where where modules with lower update frequency are connected to those with higher update frequency, but not the other way around.

Unfortunately, the language modeling task provides no clear periodic relationships in the data that can be extracted. This makes the choice of the clocking periods somewhat challenging. Consequently, we base our choice according to module constraints that are tied to the layer size and sequence length. Conformable to our chosen number of hidden units, we only investigate networks with five modules as the number of modules has to be a common divisor between the chosen layer sizes to achieve same-sized modules. Since we unroll 35 steps, we consider network periods where the module with the lowest frequency is at least activated two times. As we previously focused on exponential clocking periods, we repeat this setup with $P_1 = (1, 2, 4, 8, 16)$ but also more linearly growing periods by exploring $P_2 = (1, 2, 3, 4, 5)$ and $P_3 = (1, 2, 4, 8, 10)$.

It is evident that slow modules that activate only a handful times per sequence will likely be very difficult to train, particularly in the original unidirectional connection scheme, as these modules have very little information to work with. To properly understand if this is actually the case and how much this affects the faster modules with higher update frequencies, we take the period with the largest periodic update

[1] We use the term "bidirectional" to contrast this scheme against the one-sided (unidirectional) connections of the CWRNN. This connectivity pattern is equivalent to the default RNN connection scheme in which recurrent units are fully connected in both directions. Our use of the term is not to be confused with "bidirectional RNN layers" which offer an additional layer in which the sequence is processed backwards.

Model	ST	PPL	Period P
SRN	–	163.4	–
CWRNN	–	173.3	$(1, 2, 3, 4, 5)$
LSTM	–	132.7	–
CWLSTM	i	127.5	$(1, 2, 4, 8, 16)$
CWLSTM	ch	130.2	$(1, 2, 4, 8, 16)$
CWLSTM	h	128.1	$(1, 2, 3, 4, 5)$
CWLSTM	c	**124.3**	$(1, 2, 3, 4, 5)$

Table 5.1: Best networks by test perplexity (the lower, the better) using unidirectional connections and $|\mathbf{h}| = 1000$. Results include the best-performing period configuration (ST = Skip Target).

frequency ($P_1 = (1, 2, 4, 8, 16)$) and train five different variants of it. To this end, we "shift" the periods to the right by adding an additional 1-module and removing the slowest module. In other words, we observe the effect of gradually removing the longest shortcuts by comparing $P = (1, 2, 4, 8, 16)$ to $P = (1, 1, 2, 4, 8)$, $P = (1, 1, 1, 2, 4)$, $P = (1, 1, 1, 1, 2)$, and $P = (1, 1, 1, 1, 1)$.

5.3 Results and Analysis

The results of the experiment are presented in this section. After a brief overview in Subsection 5.3.1, we will separately evaluate the impact of the different activation functions (Subsection 5.3.2), connectivity schemes (Subsection 5.3.3), periods (Subsection 5.3.4), and skip targets (Subsection 5.3.5).

5.3.1 Overview

Before discussing the main findings of the ablation study, we present a short overview of each network's best recorded generalization performance on the task. The results can be compared in Table 5.1. The shown CWRNN and CWLSTM networks all have their original (unidirectional) connection scheme, $|\mathbf{h}| = 1000$ hidden units, and are chosen between those trained with $P_1 = (1, 2, 4, 8, 16)$, $P_2 = (1, 2, 3, 4, 5)$, and $P_3 = (1, 2, 4, 8, 10)$ (the additional hyperparameters are part of the ablation study presented in the following sections).

From this data, we can see that the CWRNN falls behind the SRN baseline noticeably. As expected, the LSTM baseline performs better than both of these two networks. This verifies the generally established importance of gated memory in RNNs for natural language processing tasks. Interestingly, all our gated CWLSTM networks, are able to additionally improve on the baseline LSTM.

Comparing the gated networks against the non-gated networks reveals that periodic activations can indeed cause network performance to degrade (as in the case of the SRN and CWRNN), likely due to the network's difficulty to ignore additional long-term information whenever it is not helpful. On the other hand, this problem seems to be successfully addressed by a gated memory mechanism which is specifically designed to allow the network to ignore any distracting long-term dependencies with the forget gate. Consequently, adding periodic activations to the LSTM is not leading to a deteriorating but an actually improved performance.

Between all introduced CWLSTM variations, using the cell state $\mathbf{c}$ as a skip target in the CWLSTM is leading to the best results, whereas using both cell and hidden state leads to only slight improvements compared to the baseline CWLSTM, while the networks with the skip targets $\mathbf{i}$ and $\mathbf{h}$ lie in-between. As we will show in the following sections, all networks can be significantly improved by removing the original Clockwork connection scheme and using traditional bidirectional connections, but at the same time minimizing the periodicity in the modules. These CWLSTMs, trained with $P = (1, 1, 1, 1, 2)$, improve by a significant margin to 119.2 PPL for $\mathbf{ch}$ (-8.4%), 119.8 PPL for $\mathbf{c}$ (-3.6%), 120.0 PPL for $\mathbf{h}$ (-6.3%), and 118.2 PPL for $\mathbf{i}$ (-7.3%).

We will now turn to evaluating the results of our ablation study in the following sections and discuss our findings regarding the impact of the architectural components of the Clockwork networks.

5.3.2 Linear vs. Non-linear Activation

In the previous experiments with sine curve prediction and learning Embedded Reber Grammar, we have found that the CWRNN works just as well with a linear activation function, even though the other investigated models (including the baseline SRN) have better performance with non-linear activations. We investigate what makes linear activations more suitable for CWRNNs by turning our focus on the two main differences between the CWRNN and the SRN: the partitioned periodic activations and the unidirectional connection scheme which only connects modules from right to left.

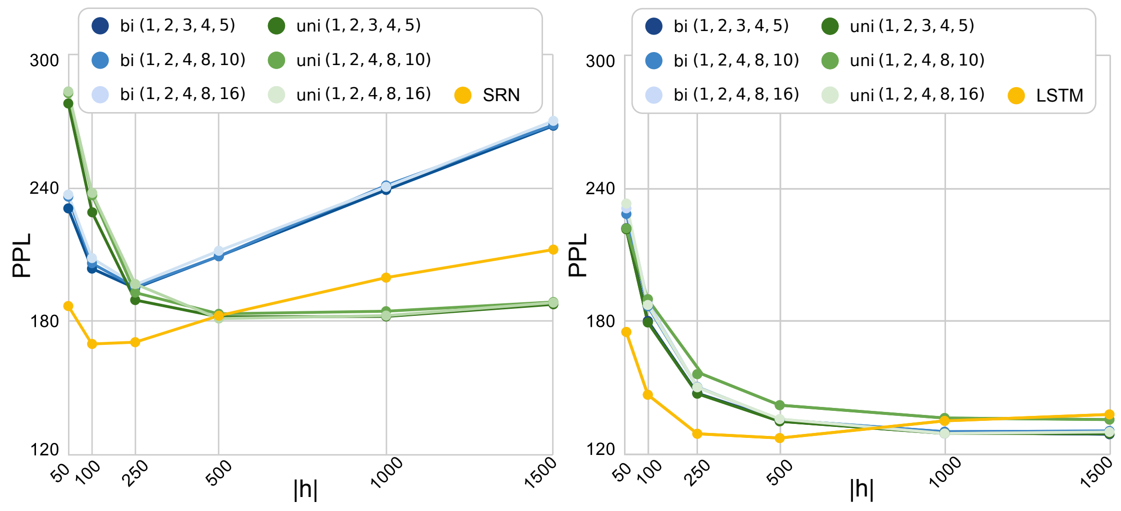
To find out which of these has the biggest impact on the variations caused by switching the activation function, we look at the difference in the mean absolute deviation (MAD) between all CWRNN networks trained with a unidirectional (the default setting) and a bidirectional connection scheme (as in an SRN), while keeping the periodic activations intact for both setups. We observe a difference between linear and tanh activation of 9.9 PPL for the unidirectional CWRNNs and a difference of 6.8 for the bidirectional CWRNNs. By the magnitude of both MAD differences, we can infer that the activation function seems to play a larger role in language modeling than in the previously investigated tasks. The effect can be attributed to this task's input representations (high-dimensional word embeddings) being quite different. This is also supported by the fact that all tanh CWRNNs perform better than the linear networks by an average of 22.1 PPL (independent of connection scheme), while this relationship was reversed in the previous tasks. From the larger performance deviation caused by unidirectional connections, we can additionally infer that these seem to have an influence on how the different activation functions are processed.

When looking at the MAD differences between different activations for the CWLSTM, we can similarly observe a larger difference with unidirectional connection schemes (0.1 PPL) than with bidirectional connection schemes (0.002 PPL). However, since the overall differences are close to 0, they can mostly be explained by small variances caused by the different random initializations. Taken together, these results suggest that the gated mechanism of the CWLSTM seems to make the choice of activation function significantly less important than for the CWRNN, though using the unidirectional connections increases the deviations.

5.3.3 Uni-/Bidirectional Connectivity

Our results validate the importance of the unidirectional connectivity scheme in the original CWRNN architecture. As shown in Figure 5.3a, the one-sided connections are essential for stable performance. In fact, the performance of bidirectional connections seems to correlate inversely to the number of hidden units. This indicates that bidirectional connections can quickly cause overparameterization which can lead to overfitting. While training a baseline SRN leads to a better language model, it is still important to note that the SRN suffers from this same issue, i.e., only the original CWRNN architecture is able to successfully scale with additional parameters. The figure also shows that all three investigated periods lead to no significant differences (we will analyze the periods more in the following Subsection 5.3.4).

As we established in Subsection 5.3.1, using memory gates leads to significantly better language models (see Figure 5.3b). Both LSTM and CWLSTM lead to a better average perplexity, scaling positively with the network size. However, different from the CWRNN, the connectivity scheme of the CWLSTM seems to play a much smaller role for the three investigated periods. With the exception of $P = (1, 2, 4, 8, 10)$, all other CWLSTMs lead to similar results with $|\mathbf{h}| > 500$, regardless of the chosen periods or connectivity. While it is still useful to prefer unidirectional connections (as this eliminates half of the recurrent weights), this heavily indicates that gated memory is a necessary addition to Clockwork networks for more complex tasks. On the other hand, we can see in Figure 5.3a how a regular LSTM is able to outperform the CWLSTM if we train smaller networks ($|\mathbf{h}| < 500$). We need to therefore differentiate that our CWLSTM improves the CWRNN while it does not beat the baseline LSTM for all hyperparameter configurations.



(a) CWRNN (uni vs bi) **(b)** CWLSTM (uni vs bi)

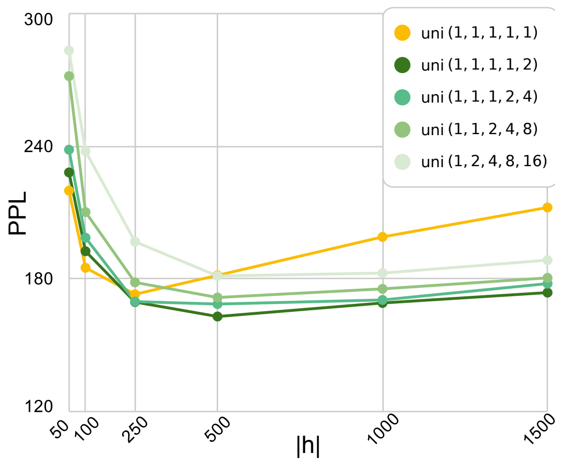
Figure 5.3: Average validation perplexity vs network size with regard to different periods, with the original unidirectional Clockwork connectivity scheme (green) and without it, i.e. regular recurrent connections in both directions (blue). Comparison between **(a)** CWRNN and **(b)** our CWLSTM.

Overall, our findings favor our CWLSTM architecture over both the CWRNN and the two baseline architectures. This is not only simply due to a better performance but also supported by a stronger robustness of the CWLSTM towards its hyperparameters and a better scalability in network size. As we will show in Subsection 5.3.5, the more surprising aspect is that eliminating most (but not all) of the long-term periodic relationships in the bidirectional CWLSTM boosts the results even further.

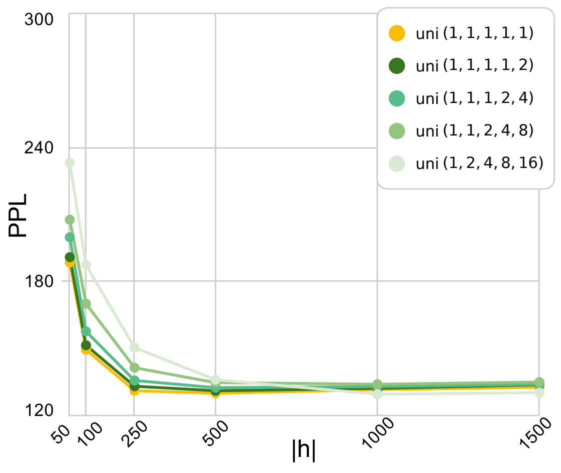
5.3.4 Impact of Large Periods

In this section, we will have a more in-depth look at how low update frequencies impact the Clockwork modules. For this purpose, we compare networks trained with $P = (1, 2, 4, 8, 16)$, $P = (1, 1, 2, 4, 8)$, $P = (1, 1, 1, 2, 4)$, $P = (1, 1, 1, 1, 2)$, and $P = (1, 1, 1, 1, 1)$ (i.e. no periodic activations).

Our results, illustrated in Figure 5.4a, confirm our previously stated hypothesis that low update frequencies with long-distance shortcuts have a negative effect on the CWRNN. We can observe a consistent gradual increase in the average validation perplexity each time we remove a module with large periods from the network and add a new module that is active at each timestep. The exception to this is are the networks with $P = (1, 1, 1, 1, 1)$ which indicates that the issue is not with the periodic activations themselves but with the infrequent updates in modules with large periods. Additionally, this is in line with previous evidence



(a) CWRNN (uni, shifting periods)



(b) CWLSTM (uni, shifting periods)

Figure 5.4: Average validation perplexity vs network size with regard to successively increasing long-term dependencies in the periods (indicated by color gradient). All networks have the original Clockwork connectivity scheme (unidirectional). Comparison between **(a)** CWRNN and **(b)** our CWLSTM.

that these long-term shortcuts add noise that the CWRNN is not able to deal with.

As the LSTM's gates are better able to suppress noise and not useful long-term relationships by utilizing the forget gate, it is not surprising to see that the unidirectional CWLSTM (Figure 5.4b) does not directly suffer from this issue: the networks have mostly consistent performance for $|\mathbf{h}| > 500$, which does neither deteriorate, nor improve with larger periods. However, these differences increase with smaller networks and if the network becomes small enough, we can observe the same relationships as in the CWRNN.

5.3.5 CWLSTM Skip Targets

We now turn to evaluating the different skip targets of the CWLSTM and their relationship to the chosen periods and connectivity schemes. Table 5.2 shows the respective results for $|\mathbf{h}| = 1000$.

For the original unidirectional CWLSTM and "regular" period setups, such as the linear series $P = (1, 2, 3, 4, 5)$ and the exponential series $P = (1, 2, 4, 8, 16)$, using the cell state $\mathbf{c}_t$ as the skip target gives the best results. The slightly more irregular $P = (1, 2, 4, 8, 16)$ seems to cause issues for all variants. The overall worst performance can be observed by using both cell $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$ as skip targets (ch). In line with previous observations, the differences between unidirectional and bidirectional are smallest for all skip targets whenever the periods seem to work properly.

	unidirectional				bidirectional			
periods	c	ch	h	i	c	ch	h	i
$(1,1,1,1,1)$	130.8	129.9	130.4	130.5	134.2	134.2	134.2	134.2
$(1,1,1,1,2)$	131.3	131.2	131.5	131.0	121.9	120.7	121.7	120.0
$(1,1,1,2,4)$	132.6	132.5	132.0	129.3	121.7	122.6	122.8	121.2
$(1,1,2,4,8)$	133.6	133.2	134.4	130.5	122.9	124.5	123.7	121.9
$(1,2,4,8,10)$	136.2	137.4	137.2	132.3	126.3	132.4	129.6	128.2
$(1,2,4,8,16)$	125.8	130.2	129.5	127.4	125.1	129.8	129.4	127.4
$(1,2,3,4,5)$	124.3	131.7	128.1	128.1	124.9	131.2	129.5	128.6

Table 5.2: Best recorded validation perplexities for all CWLSTM variants with $|\mathbf{h}| = 1000$. Normalized color gradient between minimum (green) and maximum (red) perplexities.

However, as mentioned in Subsection 5.3.3, eliminating most (but not all) of the long-term periodic relationships in the bidirectional CWLSTM gives larger performance gains than in the original unidirectional setup. While we were previously unable to see clear differences between the connectivity schemes for the CWLSTM, this paints a more differentiated picture, particularly for the “shifting periods”. In fact, there is a clear tendency that the best results are achieved by having as few periodic activations as possible, limiting the long-term dependencies and avoiding one-sided connections. On the one hand, this clearly demonstrates that the mechanisms of the Clockwork networks tend to have an overall detrimental effect. On the other hand, eliminating every single periodic activation ($P = (1,1,1,1,1)$) leads to significantly much more issues in training and the CWLSTM is still able to beat the LSTM baseline on commonly used network sizes.

5.4 Further Limitations and Approaches

As we have demonstrated, the original CWRNN architecture can be significantly improved by integrating memory gates with the periodic skipping mechanism. Our resulting CWLSTM architecture leads to better models in the language modeling domain, consistently beating the CWRNN and SRN baselines. One of the additional advantages of the model is its capability to scale in network size without significant performance changes. However, our ablation study hints towards the conclusion that the use of memory gates simply mitigate existing issues in the core mechanisms of the Clockwork architecture.

Our experiments identify difficulties to find good hyperparameter settings whenever larger periods are used. While the CWLSTM is much more robust towards this issue, the general tendency is present in both architectures (compare Table 5.2). The results of the ablation study additionally show that the unidirectional connection scheme has a negative effect on the CWRNN. For the CWLSTM, we can observe that gradually scaling back the impact of the core mechanisms improves the language models, even though we generally stay above baseline performance.

In summary, our experiments on non-periodic data show three important findings:

1. Integrating memory gates into the skipping mechanism vastly improves the performance on the language modeling task.
2. Periodic activations have a negative effect on the performance if modules with *low update frequencies* (i.e. large periods) dominate.
3. If we avoid *low update frequencies* in the CWLSTM, the negative effects of unidirectional connections increase.

Unfortunately, this leads to detrimental circular dependencies between the main components of the Clockwork architecture. As a consequence, it is impossible to fully utilize *all* core mechanics simultaneously without partially scaling down the impact of at least *one* of the components. By comparison, the experiments on periodic data (generating sinusoid sequences) heavily favor the characteristics of the CWRNN over other established RNN models. As a whole, we can therefore conclude that the inductive periodicity bias of the CWRNN is very advantageous for continuously periodic data, whereas it is potentially disadvantageous for non-periodic tasks. This main weakness of the periodic skipping mechanism seems to make the CWRNN and CWLSTM inappropriate for a wide range of complex tasks that other baseline architectures such as the LSTM are able to deal with sufficiently.

We will now discuss the main limitations of the CWRNN and consider potential solutions as well as alternative approaches towards skipping with a weaker inductive bias.

Periodicity As demonstrated, the biggest strength of the CWRNN, the inductive periodicity bias, is simultaneously its largest weakness as it requires periodic relationships in the data. In their absence, periodic connections are not tied to semantic meaning or any long-term relationships underlying the input (see example in Figure 5.2). In contrast, the general idea behind shortcut connections is to provide direct memory access to relevant previous states, secondarily avoiding noise from irrelevant states. The way the skipping mechanism is set up, this can naturally only be achieved by accident or if there are actual periodic relationships in the data. While a periodic signal can be used as a "reference point" to encode arbitrary non-periodic relationships[2], this requires the ability to phase shift either the periodic signal itself or the encoding. This is impossible if the signal is not parameterized and if the encoding process itself only takes place within this periodic update signal.

Dependence on System Clock While there is a case to be made that the periods can be set up in a way to increase the number of "lucky accidents" to have a statistically higher chance to capture more useful relationships, there is an additional issue that makes this harder: the fact that the periodicity is immutably tied to a system clock. This design flaw is independent of the task and domain and significantly affects generalization capability. Specifically, the skip condition $t \bmod T_k = 0$ ties each module's period directly to multiples of T_k. As it is impossible to dynamically learn an offset to T_k, the network can have large difficulties generalizing to a previously trained sequence that is now slightly shifted by a few timesteps in any direction. The missing shift invariance also makes it impossible for the network to adapt to data that is only partially periodic, e.g. relevant in event-based prediction tasks, as we have to assume periodicity throughout the entire sequence. Although it is simple to add a static offset t_o such that $t \bmod (T_k + t_o) = 0$, it has to be trainable as we would otherwise just end up with the original issue at timestep $t + t_o$ instead of t. However, integrating arbitrary shifts by a trainable t_o would effectively remove the assumption of periodicity and lead to a fully adaptive skipping mechanism (which we will explore in Part III and Part IV). While such a model would arguably be more flexible, it would necessarily have to discard the concept of a periodic inductive bias.

Adaptive Periods A simpler approach to making the periods adaptive is to learn them based on global statistics, keeping them static throughout the sequence and strictly enforcing periodicity. However, this does not address the problem that periodicity can sometimes be paused only to be resumed later, potentially with a slight offset. It might also bias the network to greedily focus on short-term relationships as frequent updates provide quantitatively more error signals. Since backpropagation cannot look ahead multiple timesteps into the future and plan towards a later update, it will always choose to directly and immediately minimize the error of modules with large periods. It is open how this could be addressed, though it would be possible to use bidirectional layers in order to allow the network to look ahead in the sequence. Regardless, whenever the adaptive mechanism decides to accidentally set very large periods, the respective module would effectively stop training for the rest of the sequence, potentially requiring additional solutions similar to dead neurons caused by ReLu activations.

[2]Positional encodings in the Transformer (Vaswani et al., 2017) provide such an example.

Modules and Connectivity Disregarding effects from periodicity, it can be argued that the modules, their ordering by update frequency, and the resulting one-sided connectivity do not constitute actual design flaws. Quite contrary, our visualizations show increased diversity as modules are actually *trained* on different time scales. This effect has also been demonstrated with a number of other modular models (see Subsection 2.3.2).

Clockwork RNN Extensions Recent work has built on addressing some of the discussed shortcomings of the CWRNN. One such example is the Adaptive Clockwork Convnet (ACC) which is used for semantic segmentation in videos and operates under the assumption that the semantic content of the frames changes on a slower timescale than the pixel differences between the video frames (Shelhamer et al., 2016). For this purpose, skips are triggered whenever the difference between two successive frames is larger than a predetermined threshold. As discussed previously, such an approach removes any periodicity from the model and can thus, in the context of similar existing research, be seen as a straight-forward skipping condition in a more general conditional computation approach[3]. While the authors of the study posit the idea of learning periodic update functions, they do not explore concrete methods to do so.

A separate study achieves shift invariance in their Dense Clockwork RNN (DCWRNN) architecture at the cost of additional parameters and computational resources (Neverova et al., 2016). They introduce M^{p_m} parallel threads to each of the M modules (with their respective period p_m) that are all shifted with respect to each other. While this causes all modules to constantly update, the authors claim that this leads to a speedup in the training process.

Carta et al. (2020) address the issues around training difficulties of slow modules by initially starting out with fast modules and adding slower modules incrementally in a pre-training process before training each model fully with SGD. They evaluate their model on the sequence generation task and are able to improve on the mistakes which the CWRNN typically does in the first few timesteps of each sequence (see discussion in Subsection 4.4.1). One set of later approaches that are potentially inspired by the CWRNN are Dilated CNNs (Eppe et al., 2018) and Dilated RNNs (Chang et al., 2017). They utilize multi-resolution skip connections that are based on dilated causal convolutions which were first introduced with the WaveNet architecture (van den Oord et al., 2016). The common ground for these approaches and the CWRNN is the idea to exponentially expand the resolution of the representations in time or space to capture multiscale features.

However, a significantly larger amount of research has since moved away from the idea of periodic activations and towards the general idea of conditional computation where skip conditions are either learned or have less restrictive constraints (see Sec-

[3]In fact, we will later introduce our Surprisal-based Activation models which operate on a comparable thresholding mechanism.

tion 3.5 for an overview). Similarly, the rest of this thesis will explore non-periodic mechanisms to successfully skip updates in recurrent neural networks.

5.5 Chapter Summary

Our previous analysis of the CWRNN has shown that its inductive bias is particularly useful for learning periodic relationships in data. We have confirmed initially observed difficulties with non-periodic data by evaluating the CWRNN on a widely known language modeling task. While we were able to significantly improve the CWRNN by integrating memory gates in our novel CWLSTM architecture, our ablation study demonstrates that periodic activations are inherently problematic in a number of settings. As part of our discussion, we have identified that addressing this issue with more flexible adaptive skipping mechanisms can quickly lead to abandoning the concept of periodic activations altogether.

Since most real-world applications have shifting dynamics and timescales, a single clock rate that is fixed in time (e.g. determined by the average periodicities in the data) seems to be not robust enough to process scale dynamics that deviate significantly from a periodic bias. This leads to the first main issue of the CWRNN, namely the global nature of the clock rates (periods). The other main drawback of the CWRNN is the shift-variance caused by the hard-coded clock timing. The exact same input sequence will result in different outcomes when placed on different starting times t. As this is completely data independent, minor phase shifts to previously learned input render predefined periods useless and, consequently, the model itself impractical. This shows that both adaptive periods and data-driven clocking necessitate each other: a model with adaptive periods will cover most periodicities of the input but will still be susceptible to phase shifts. An adaptive clocking mechanism, on the other hand, is by definition unpredictable before execution and is therefore incompatible with manually set static periods. Consequently, a dynamical and robust clock needs to be both local and adaptive - a function of time and data.

In the next part of this thesis, we will thus explore how we can continue and further develop locally adaptive skipping mechanisms with a weaker (non-periodic) inductive bias that also require less expert knowledge and tweaking. As part of this effort, we will design a new skipping model, integrating the design concepts of the previous chapters that have shown initial promise, such as partitioning the hidden layer into modules to increase unit diversity, and jointly activating groups of neurons in modules.

Part III

Surprisal-Based Activation

Chapter 6

Preserving Activations with Surprisal-Based Activation

One of the current main challenges of Recurrent Neural Networks (RNNs) is to dynamically adapt to multiple temporal resolutions and scales in order to learn hierarchical representations in time. Since they operate in discrete timesteps and update at every timestep, it is generally difficult to learn temporal features that have a significantly different resolution than their input frequency. In particular, many applications, such as speech recognition or video analysis, require a high data resolution to capture very short but important events (Zelnik-Manor and Irani, 2001). However, increasing the input resolution has the negative side effect that task-defining events are then very sparsely distributed over the observed time series while most data points are redundant and mostly irrelevant for the task. This causes computational redundancy in state updates of RNNs, ultimately leading to unnecessarily large computational graphs that are hard and expensive to train with Back-Propagation Through Time (BPTT).

This issue with RNNs is most commonly tackled by avoiding high-resolution raw data as input and instead relying on task-specific time-averaging features or down-sampling where possible. There is a number of alternative approaches that can be categorized under conditional computing (see Chapter 3) in which state updates are optional and based on conditions. Theoretically, this allows the model to actually learn which data points to encode, discarding the rest, which can ultimately even be utilized to save computation time. We hypothesize that these approaches can be utilized to train (temporal) feature-learning RNNs more efficiently in an end-to-end manner. The capability to learn *when* to update could open up promising directions for a number of current research problems (see Section 3.6).

Sections 6.2-6.5.1 of this chapter have been published as part of this thesis and are based on Alpay et al. (2018) and Alpay et al. (2019).

In this chapter, we introduce surprisal-based activation (SBA), an extension to existing RNNs, which allows the inhibition of state updates based on the surprisal over changes in the latent space over time. Preserving activations allows to store explicit memory for an arbitrarily long time before it is accessed and aims to avoid unnecessary changes in the encoding. We demonstrate how SBA can be integrated with the SRN as well as gated models such as the LSTM and provide an extensive analysis of our methodology. Compared to our previous approach of periodic activation with the CWLSTM (Chapter 5), we use a significantly weaker inductive bias as a skipping constraint but revisit the idea of using submodules to increase diversity within a layer.

We start this chapter by presenting our motivation and design goals for the introduced methodology in Section 6.1 before providing an overview of related work that has influenced our approach in Section 6.2. Our method is introduced and explained in Section 6.3. We evaluate our approach extensively on language modeling in Section 6.4 and investigate whether these findings generalize to other sequence learning tasks in Section 6.5. We conclude with our closing thoughts in Section 6.6.

6.1 Background and Design Goals

The primary goal for our model will be to use skip mechanisms to reduce as much redundancy as possible, while maintaining an accuracy close to the baseline. Any improvements beyond the baseline would signal that the network is benefiting from sparse representations. To identify redundancy, we utilize surprisal in a modular network design (see also Chapter 2). Before jumping to our model design, we will briefly cover these concepts along with the required background.

Entropy

Entropy is a measure for uncertainty in the outcome of a probability distribution $P = (p_1, \ldots, p_n)$, introduced by Shannon (1948) in the context of communication and information theory. In this context, the non-negative entries of a probability vector P sum to 1. According to Rényi et al. (1961), entropy can be quantified as follows:

$$
\begin{aligned}
H(P) &= H(p_1, \ldots, p_n) && (6.1)\\
&= \sum_{k=1}^{n} p_k \ \log_2\left(\frac{1}{p_k}\right) && (6.2)\\
&= -\sum_{k=1}^{n} p_k \ \log_2(p_k) && (6.3)
\end{aligned}
$$

As such, entropy can also be seen as the average *surprisal*. Surprisal is an information-theoretic metric that measures the amount of information conveyed by a particular event, particularly its “unexpectedness” or “surprise”. It is inversely related to probability so that improbable events carry more information than probable ones. Surprisal (also called information content $\mathcal{I}$) for a random variable X is formally defined as:

$$\mathcal{I}(X) = \log\left(\frac{1}{P(X)}\right) = -\log(P(X)) \tag{6.4}$$

$\mathcal{I}(X)$ is therefore bounded between 0 and the number of bits to store X. Unlikely events X lead to a large surprisal whereas highly probable, in our context *redundant*, events lead to a low surprisal.

Lossless Compression

Source coding concerns encoding a sequence of symbols (in the context of communication: a message) that is subsequently decoded at the receiving end. If the encoding has less symbols than the source, it has been *compressed*. Furthermore, if the decoded symbols of the smaller message reproduce the original sequence exactly, i.e. without loss of information, we speak of *lossless compression*. Shannon's source coding theorem (Shannon, 1948) states that the entropy $H(X)$ provides a measurement as to how many bits X can be reduced in an encoding before we risk losing information which is unrecoverable by the decoder. Shannon (1948) expresses this quantity with *relative entropy* $H_r(X)$, the ratio between the actual and maximum entropy. The goal of compression is to reduce the amount of redundancy in the sequence of symbols. The information *redundancy* is therefore defined as $1 - H_r(X)$ which gives the amount of bits by which the encoding can be reduced while still allowing a lossless compression.

The *principle of maximum entropy* by Jaynes (1957) can be applied when inference has to be made based on only partial information. It states that the distribution with the maximum entropy should form the basis for inference as it makes the least amount of assumptions. As such, it can be seen as a form of Occam's razor which is also reflected in formulations around the minimum description length (MDL) principle which favors models fitting the data with the least amount of parameters.

In the context of decoding, high redundancy makes it easier to decode (high predictability) even though there is little information in the code. High entropy, on the other hand, makes it more difficult to decode (high unpredictability) even though there is a lot of information content. In the context of reading, predictability is correlated with word skipping (Rayner et al., 2011) as readers pay more attention to unpredictable words.

Other measures for redundancy have been proposed such as measuring the change in surprisal between variables (Ince, 2017). Surprisal is also used as a human sen-

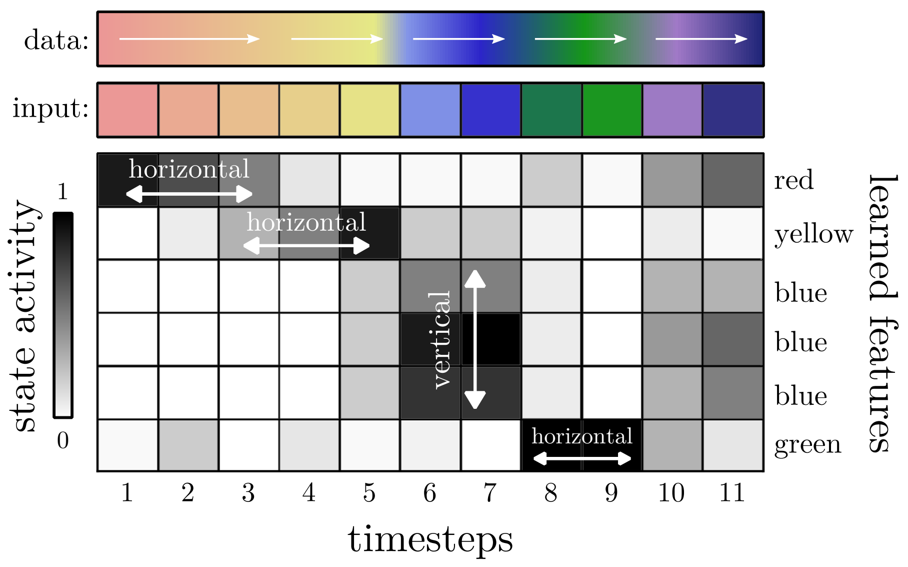


Figure 6.1: Horizontal vs vertical redundancy: Vertical redundancy is often caused by co-dependencies and over-parameterization (such as the 3 "blue" units behaving exactly the same), whereas horizontal redundancy is primarily a consequence of the necessity to update at every timestep.

tence processing cost model in linguistics (Futrell and Levy, 2017). The underlying assumption is that the processing cost of each word is proportional to the resulting belief change from processing this word. This is particularly difficult for memory-based models which have to syntactically integrate words over long time-gaps and therefore suffer from locality effects (Futrell and Levy, 2017). In this context, it is also important to distinguish temporal redundancy from spatial redundancy.

Redundancy in Recurrent Networks

We distinguish two types of encoding redundancy in the context of recurrent networks, namely vertical and horizontal redundancy (see Figure 6.1). Vertical redundancy denotes correlations between units as they encode the same attributes and features. Units can have a strong correlation, showing the same activation patterns for all inputs, or have a weak correlation, by only sharing similar sensitivity to certain inputs. Horizontal redundancy, on the other hand, occurs if units specialize on certain features. For timesteps, in which these features do *not* occur, we would therefore typically expect low activations that show weak correlations to other units or simply produce noisy patterns as a result of overfitting.

The general idea behind preserving activations is illustrated with a color classification example in Figure 6.2: an analog continuous-time stream of data has to be down-sampled into discrete timesteps to be processed, reducing entropy in the process. Consequently, some timesteps might contain overlapping information due to a low sampling rate. Modules (groups of units) specializing on distinct and discrete features (such as color groups) are therefore often active beyond the boundaries of certain events. This is amplified by additional noise and the general behavior of backpropagation, causing units to be constantly active, picking up co-dependencies[1], increasing redundancy, and making specialization difficult (see top matrix for regular RNN). An idealized group of units learning distinct features should be able to eliminate redundancy in such a way that the state activity clearly reflects the respective event boundaries of features (see bottom matrix for preserving RNN). As can be seen, preserving activations until it is "worth" updating, introduces another layer of entropy reduction (apart from the down-sampling of the input), resulting in simpler hypotheses generation and interpretation.

However, while vertical redundancy is a clear computational and representational issue (addressed in the field of learning disentangled representations), the benefits and issues of horizontal redundancy are arguably harder to assess. It is easy to propose a number of scenarios in which the continuous space between two discrete representations (such as e.g. "red" and "yellow") holds potentially useful information ("orange"). Although this can only follow if this feature is in fact statistically relevant as otherwise the network would converge to one of the two defined representations ("red" or "yellow"). In our constructed example of Figure 6.2, the representation for "orange" is a bad predictor for the entire dataset and therefore a wholly *redundant* feature. If we expect this feature to never be relevant, eliminating the parameter space encoding "orange" would lower model complexity without loss of information, additionally allowing us to more clearly identify the temporal relevance of the remaining discrete features.

6.2 Preserving Activations

Some recent approaches focus on the idea of suppressing hidden unit activations under specific conditions. The Clockwork RNN (CWRNN; Koutník et al. (2014)) has a hidden layer that is partitioned into *modules* which are only activated at specific timesteps. Inactive modules simply preserve their previous hidden activations until they are triggered to activate again. These activation triggers are under periodic cycles, static, and determined empirically. It can be shown that such training with multiple update resolutions can facilitate learning multiple timescales (see Chapter 4). A key disadvantage of the CWRNN, however, is that the periodic activation conditions are i) predefined and not learned, and ii) global for the entire sequence. This can lead to challenges when dealing with varying temporal distances between dependencies or phase shifts which are common in real-world applications (see Chapter 5). This idea has been developed further as a regularization method for RNNs with zoneout (Krueger et al., 2017). Zoneout randomly preserves the previous activations of hidden units and can therefore be seen as a

[1]To keep model complexity low, we address horizontal redundancy directly but only indirectly work against vertical redundancy by varying the number of available modules.

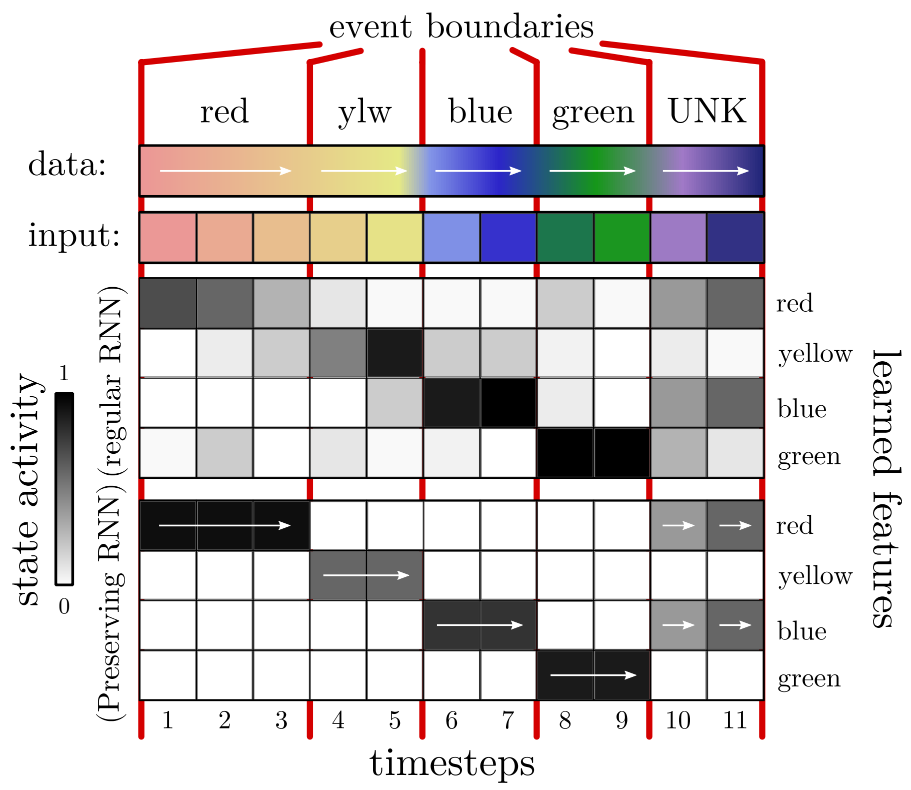


Figure 6.2: Design goal for SBA, illustrated by the example task of color classification. A typical RNN is constantly active and therefore picks up co-dependencies, amplifying redundancy and making unit specialization difficult. As a result, even a well-trained network exhibiting signs of unit specialization is outputting redundant, irrelevant information outside of event boundaries (marked by red vertical lines). In an ideal target design (bottom matrix), the network would be able to have clear event boundaries and only update when the observed feature changes significantly (above: timesteps $1, 4, 6, 8, 10, 11$). Within these event boundaries, activities should be preserved to coincide with the labels (indicated by arrows). Activation preservation further increases sparsity and reduces entropy, improving general interpretability and computational efficiency.

variant of dropout in which connections are masked with a random mask of ones (copy) instead of zeros (drop). In the context of recurrent architectures, zoneout is also a special case of the CWRNN with a single module and clocking timings that are sampled randomly at each timestep.

Skipping irrelevant information has also been achieved with RNNs trained with reinforcement learning (Yu et al., 2017a; Johansen and Socher, 2017) although there are successful strategies to estimate gradients for conditional computation (Bengio et al., 2013b). Adaptively learning computational boundaries has been achieved with gradient descent using computation penalties (Graves, 2016), binary boundary gates (Chung et al., 2017), and time gates based on rhythmic oscillations (Neil et al., 2016). The Skip RNN even uses binary update gates (Campos et al.,

2018) which is similar to our approach, even though our update model enforces an information-theoretic constraint based on surprisal.

Surprisal of the error signal has previously been proposed for a form of adaptive zoneout (Rocki, 2016). However, while this is fully adaptive, it depends on additional supervised information in both training and test phases. An important feature of surprisal is that it works well for segmentation (Griffiths et al., 2015) which we utilize to detect information boundaries and preserve activations.

6.3 Surprisal-Based Activation

We utilize the previously discussed properties of surprisal to implicitly segment the state update computations along boundaries of high surprisal and activity. For this purpose, we redefine any event X as the currently calculated state $\mathbf{h}_t$ at timestep t which we seek to segment according to its activations.

In the following, we assume a single hidden layer $\mathbf{h}_t$ of a standard RNN (recurrent and fully-connected) which naturally generalizes to a multi-layer network. The standard RNN candidate activation $\hat{\mathbf{h}}_t$ is calculated as follows:

$$\hat{\mathbf{h}}_t = f(\mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1}), \tag{6.5}$$

where $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$ are the input and recurrent weight matrices, and $\mathbf{x}_t$, $\mathbf{h}_{t-1}$ the input and previous state, respectively. Our aim is to now observe $\hat{\mathbf{h}}_t$ and to retroactively determine which of the contained activations to "rewind" to the *previous* timestep $t-1$, thus preserving them for an additional timestep. Consequently, this determines which activations to keep from the *current* timestep, passing them to the final layer output $\mathbf{h}_t$.

Figure 6.3 illustrates all computation steps of our approach. We start by partitioning our hidden layer into M modules $\mathbf{m}_t^{(i)}$ of equal size[2] such that:

$$\hat{\mathbf{h}}_t = [\hat{\mathbf{m}}_t^{(1)}, \ldots, \hat{\mathbf{m}}_t^{(M)}] \tag{6.6}$$

The purpose of these modules is to introduce stability through majority weighting since the decision to apply the candidate activation $\hat{\mathbf{h}}_t$ or to preserve $\mathbf{h}_{t-1}$ is made independently for each module. We therefore pool the candidate activations in the next step to compress the activations on a per-module-basis, resulting in the pooling vector $\mathbf{p}_t$ with $|\mathbf{p}_t| = M$:

$$\mathbf{p}_t = g(\hat{\mathbf{h}}_t) = [g(\hat{\mathbf{m}}_t^{(1)}), \ldots, g(\hat{\mathbf{m}}_t^{(M)})], \tag{6.7}$$

[2]For a simplified implementation and to avoid further hyperparameters, we define the same module size depending on M. Choices of M are thus restricted by the constraint $0 \equiv |\mathbf{h}| \mod M$. Therefore, having more modules leads to less units per module and vice versa.

where $g(\cdot)$ is a pooling operator such as *max* or *avg*, applied on each module such that each module is projected to a single element with $g(\hat{\mathbf{m}}_t^{(i)}) = \mathbf{p}_t^{(i)}$ while the complete pooling vector $\mathbf{p}_t$, which consists of all of these elements, represents the entirety of all module projections, i.e. the locally pooled hidden layer. In the next step, we normalize with softmax σ and calculate the surprisal s_t from the resulting probability distribution:

$$s_t = \mathcal{I}(\mathbf{p}_t) = \log\left(\frac{1}{\sigma(\mathbf{p}_t)}\right) = \log\left(\frac{\sum_{i=1}^{M}\exp(\mathbf{p}_t^{(i)})}{\exp(\mathbf{p}_t)}\right) \tag{6.8}$$

As a final step, each module $\mathbf{m}_t^{(i)}$ is activated depending on their respective change in surprisal. We choose candidate activations where the surprisal grows larger than an experimentally determined hyperparameter θ, and otherwise preserve all states of the i-th module:

$$\mathbf{m}_t^{(i)} = \mathcal{S}(\hat{\mathbf{m}}_t^{(i)}) = \begin{cases} \hat{\mathbf{m}}_t^{(i)} & \text{if } s_t > s_{t-1} + \theta, \\ \mathbf{m}_{t-1}^{(i)} & \text{otherwise,} \end{cases} \tag{6.9}$$

where $\mathcal{S}(\cdot)$ denotes the module-wise transformation of the activations, which is ultimately applied on the candidate activations $\hat{\mathbf{h}}_t$ so that we end up with the final hidden layer output $\mathbf{h}_t$:

$$\mathbf{h}_t = \mathcal{S}(\hat{\mathbf{h}}_t) = [\mathcal{S}(\hat{\mathbf{m}}_t^{(1)}), \ldots, \mathcal{S}(\hat{\mathbf{m}}_t^{(M)})] \tag{6.10}$$

As a consequence, some modules end up being preserved by $\mathcal{S}(\cdot)$ for this final state $\mathbf{h}_t$ (depending on the condition met in Equation 6.9), while others use the original candidate activations, updating their internal model for the current timestep. We henceforth call the resulting model RNN+S (Figure 6.3). All other parts of the RNN are trained normally and backpropagation through time is executed without any modifications.

For a simple interpretation of our model, it is possible to look at it without the information-theoretic motivation which is expressed in the definition of surprisal $\mathcal{I}$ (Equation 6.4). The application of surprisal in Equation 6.8 causes the pooled activations to be projected to negative log space. This both rescales the activations and introduces a lower bound at 0. While sequential activation comparisons are possible in the original activation space, the negative log space is much more feasible for numerical comparisons as the *difference* can now be expressed regardless of the activation scale. For example, $\mathcal{I}(0.1) - \mathcal{I}(0.2) = \mathcal{I}(0.0001) - \mathcal{I}(0.0002)$, whereas omitting the projection leads to a very large difference between both sides of the equation. One benefit of this is that it allows us to look at this difference and set a threshold θ that is mostly disconnected from the scale of our activations. Since we additionally normalize via softmax before projecting the pooled activations (compare Equation 6.8), this leads to an overall more predictable numerical range in which we try to compare activations. From this perspective, our approach can be seen as segmenting activations according to their change in values.

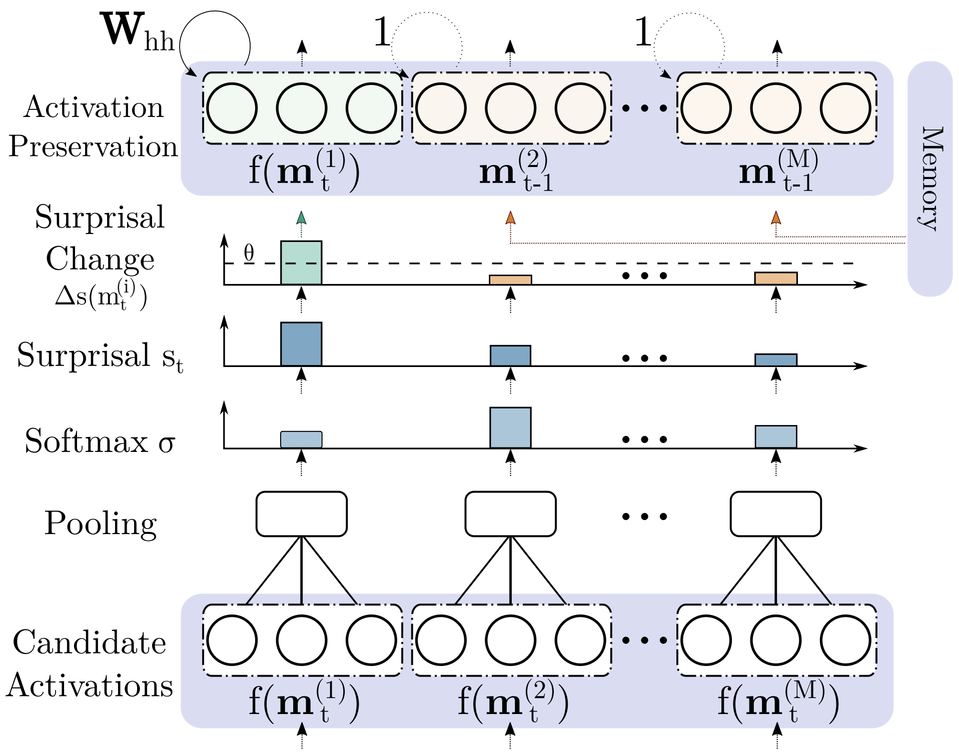


Figure 6.3: Overview of surprisal-based activation within a hidden layer. Module candidate activations in t that are significantly different from the previous timestep $t-1$ are kept (green), all other modules preserve their previous states $\mathbf{m}_{t-1}^{(i)}$ (red).

It is also important to note that, while skipping redundant words eliminates these from the processing chain, the copy mechanism itself causes an increase in redundancy for the representation itself, therefore increasing surprisal for non-skipped entries in the same timestep. Thus, high surprisal on the representation represents portions of larger activations. During sequential processing, comparing the surprisal is therefore meant to give an understanding of how the activations of specific units or modules change over time. As we compare condition activations on the differences between neighboring timesteps, we can also see this as related to the principle of temporal coherence (Subsection 2.2.2) even though we do not enforce slowness explicitly.

We hypothesize that the combination of local pooling and preservation of previous states based on surprisal maintains the regularizing effect of zoneout, since surprisal-based activation is specifically designed to ignore small perturbations to hidden states, which we would expect to regularize transition dynamics. Another hypothesis is that our design can also lead to a self-organization process in which modules separately learn independent features over long time distances during which the input, and subsequently the encoding, does not undergo significant changes.

6.3.1 Activation Decay

One potential pitfall of preserving activations by internal surprisal is a "deadlock" in which an encoding sequence with mostly constant surprisal is never updated. To counteract this, we investigate the effect of activation decay. We apply decay right before Equation 6.9 on the preserved state $\mathbf{h}_{t-1}$ and define this decayed state to be $\acute{\mathbf{h}}_t$. This changes the negative condition of Equation 6.9 to preserving the decayed version of the state:

$$\mathbf{m}_t^{(i)} = \mathcal{S}(\hat{\mathbf{m}}_t^{(i)}) = \begin{cases} \hat{\mathbf{m}}_t^{(i)} & \text{if } s_t > s_{t-1} + \theta, \\ \acute{\mathbf{m}}_{t-1}^{(i)} & \text{otherwise,} \end{cases} \tag{6.11}$$

Our first approach is to apply a constant decay $d = 1 - \alpha$ $(0 < \alpha < 1)$ to the preservation state $\hat{\mathbf{h}}_{t-1}$ such that $\acute{\mathbf{h}}_t = \mathbf{h}_{t-1} \cdot (1 - d)$ constantly decays by the amount α. As a second variant, we introduce a more local and random method by element-wise multiplying the hidden units with a decay mask:

$$\acute{\mathbf{h}}_t = \mathbf{h}_{t-1} \odot \mathbf{d}_t \text{ with } \mathbf{d}_t = \big(P(\mathbf{d}_t^{(1)}), \cdots, P(\mathbf{d}_t^{(|\mathbf{h}|)})\big) \tag{6.12}$$

where $\mathbf{d}_t$ is a decay vector, and $P(d_j = 1 - \alpha) = p_d$ and $P(d_j = 1) = (1 - p_d)$ are probabilities for decaying the activation of unit j by α with probability p_d. We hypothesize that this introduces variability on a sub-module level and allows single units to counteract potential information loss from pooling. We set these values to $\alpha = 0.01$ and $p_d = 0.2$ as the result of initial tests which indicated that these parameters are not very sensitive as long as they remain small. Since our initial assumption is that deadlocks are global (otherwise other interconnected units would break locally "dead" units) and that few units have to be perturbed to reactivate the entire system, a small probability p_d of causing these perturbations fits this intention. It is however important to note that α is dependent on the range of the used activation function $f(\cdot)$. The same holds true for the threshold θ even though it operates in the log space of the activations. Unbounded activation functions (such as linear or ReLu activation) would result in different numerical properties and therefore a potentially different set of values. In this chapter, our conclusions are specifically based on working with the bounded tanh and sigmoid activation functions.

6.3.2 Surprisal-Based Activation in the LSTM

As the LSTM has more parameters than a standard RNN, there are more possibilities to apply surprisal-based activation. The LSTM has two state variables, i.e. the cell state $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t$. It also has three gates, namely the forget gate $\mathbf{f}_t$, output gate $\mathbf{o}_t$, and input gate $\mathbf{i}_t$. Their interplay, as defined by Greff et al.

(2017), is as follows (omitting bias for clarity):

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t && (6.13)\\
\hat{\mathbf{c}}_t &= g(\mathbf{W}_{xc} \cdot \mathbf{x}_t + \mathbf{W}_{hc} \cdot \mathbf{h}_{t-1}) && (6.14)\\
\mathbf{h}_t &= \mathbf{o}_t \odot f(\mathbf{c}_t) && (6.15)\\
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi} \cdot \mathbf{x}_t + \mathbf{W}_{hi} \cdot \mathbf{h}_{t-1}) && (6.16)\\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf} \cdot \mathbf{x}_t + \mathbf{W}_{hf} \cdot \mathbf{h}_{t-1}) && (6.17)\\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{W}_{ho} \cdot \mathbf{h}_{t-1}) && (6.18)
\end{aligned}$$

The preservation function $\mathcal{S}(\cdot)$ can therefore naturally be applied to either or both of $\mathbf{c}_t$ and $\mathbf{h}_t$. For these cases, we can apply Equation 6.9 directly on the LSTM. We call the resulting variations S_c, S_h, and S_{ch}.

Since the LSTM forget gate presents an alternative model of memory retrieval to our approach ($\mathbf{f}_t^{(i)} = 0$ causes forgetting, $\mathbf{f}_t^{(i)} = 1$ causes remembering), we also investigate tightly integrating both approaches, i.e. forcing an indirect activation preservation by forcing the network to remember through the forget gates. We therefore extend the preservation function $\mathcal{S}(\cdot)$ for the forget gate:

$$\mathbf{f}_t^{(i)} = \mathcal{S}(\mathbf{f}_t^{(i)}, \mathbf{k}_t) := \begin{cases} \hat{\mathbf{f}}_t^{(i)} & \text{if } \mathcal{I}(\mathbf{k}_t) > \mathcal{I}(\mathbf{k}_{t-1}) + \theta, \\ \mathbf{1} & \text{otherwise,} \end{cases} \tag{6.19}$$

where $\mathcal{I}(\mathbf{k}_t) = s_t$ gives the surprisal s_t based on the observed vector $\mathbf{k}_t$. We investigate $\mathbf{k}_t \in \{\mathbf{h}_t, \mathbf{c}_t, \mathbf{f}_t\}$ to clarify which of these qualify as the best trigger for the forget gate. These three S_{fk} methods working on the forget gate do not actively modify the cell and state but instead lead to $\mathbf{f}_t^{(i)} = 1$ as the preservation condition. The cell update $\mathbf{c}_t$ therefore changes along with $\mathbf{h}_t$ as follows:

$$\begin{aligned}
\mathbf{h}_t &= \mathbf{o}_t \odot g(\mathbf{c}_t) \\
&= \mathbf{o}_t \odot g(\mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t) \\
&= \mathbf{o}_t \odot g(\mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t)
\end{aligned} \tag{6.20}$$

From $\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t$, it follows that the S_{fk} variants differ from S_c (where $\mathbf{c}_t = \mathbf{c}_{t-1}$) by whether the input gate $\mathbf{i}_t$ is frozen along with the cell $\mathbf{c}_t$ or not.

Setting the input gate $\mathbf{i}_t = \mathbf{0}$, gives a particularly interesting variant in which the cell does not receive any new inputs but simply decays with the forget gate (as briefly discussed but not explored in Krueger et al. (2017)):

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \\
&= \mathbf{f}_t \odot \mathbf{c}_{t-1}
\end{aligned} \tag{6.21}$$

We therefore also explore this variant, calling it S_{ic}, in which we selectively deactivate the input gate to block *all* outside influence on the memory cell, based on the cell:

$$\mathbf{i}_t^{(i)} = \mathcal{S}(\mathbf{i}_t^{(i)}, \mathbf{c}_t) := \begin{cases} \hat{\mathbf{i}}_t^{(i)} & \text{if } \mathcal{I}(\mathbf{c}_t) > \mathcal{I}(\mathbf{c}_{t-1}) + \theta, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{6.22}$$

All of these discussed variants operate locally and preservations are decided for each module. Overall, we evaluate the following seven variants of surprisal-based activation in an LSTM:

- $LSTM + S_h$: Preserving **hidden states** with $\mathcal{S}(\hat{\mathbf{h}}_t)$
- $LSTM + S_c$: Preserving **cell states** with $\mathcal{S}(\hat{\mathbf{c}}_t)$
- $LSTM + S_{ch}$: Preserving **both** cell and hidden states with $\mathcal{S}(\hat{\mathbf{c}}_t)$, $\mathcal{S}(\hat{\mathbf{h}}_t)$
- $LSTM + S_{ff}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **forget gate** with $\mathcal{S}(\mathbf{f}_t, \mathbf{f}_t)$
- $LSTM + S_{fh}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **hidden state** with $\mathcal{S}(\mathbf{f}_t, \mathbf{h}_t)$
- $LSTM + S_{fc}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **cell state** with $\mathcal{S}(\mathbf{f}_t, \mathbf{c}_t)$
- $LSTM + S_{ic}$: Setting $\mathbf{i}_t = \mathbf{0}$ based on observing the **cell state** with $\mathcal{S}(\mathbf{i}_t, \mathbf{c}_t)$

To summarize, these proposed LSTM variants can be grouped into three separate main approaches: i) those that preserve states directly (S_h, S_c, S_{ch}), ii) those that preserve states by forcing the forget gate to remember (S_{ff}, S_{fh}, S_{fc}), and finally iii) blocking all input to the memory cell (S_{ic}). We compare and evaluate these variants experimentally in the following sections. The objective is to find the most successful setup and also to investigate how the introduced core mechanism interacts with different parameters and how this influences the internal dynamics.

6.4 Evaluation on Language Modeling

6.4.1 Experimental Setup

To investigate the effect of surprisal-based activation, we evaluate our model with language modeling.

While many recent studies make heavy use of regularization techniques to combat the corpus' high susceptibility for overfitting and to reach the lowest possible perplexity, we are mainly concerned with evaluating performance gains from our approach in controlled conditions. We therefore run our own RNN and LSTM models as a baseline and apply our method under the same experimental conditions. Avoiding explicit regularization and task-related tuning methods also allows us to investigate how prone our networks are towards overfitting by themselves. Our hyperparameters for gradient descent follow (where possible) the previous state of the art for this dataset, achieved by Recurrent Highway Networks (Zilly et al., 2017), even though we restrict ourselves to a single medium-sized layer with 1000 hidden units in order to explore a larger hyperparameter space. Ultimately, we run a total of 1440 different network configurations. We test variations for pooling (max vs. average), activation decay (none, probabilistic, constant), number of modules

Model	Best Val.	Test
RNN	130.4	140.8
RNN+S	131.5	**126.4**
LSTM	123.6	121.5
LSTM+S_{ch}	128.4	125.7
LSTM+S_c	123.1	120.8
LSTM+S_h	123.9	120.4
LSTM+S_{ff}	125.5	124.0
LSTM+S_{fh}	125.2	123.6
LSTM+S_{fc}	122.3	120.2
LSTM+S_{ic}	116.3	**114.4**

Table 6.1: Single model validation and test perplexity (the lower, the better) of our models (+S) compared against the LSTM and RNN baselines on the Penn Treebank corpus.

$M \in \{2, 4, 8, 10, 25, 50, 100, 250, 500, 1000\}$, and threshold $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$. The LSTM models use *tanh* for activation, the RNN models the sigmoid activation function. Standard gradient descent is used for training with a mini-batch size of 20, a segmented sequence length of 35, and the gradient clipped at 10. Additionally, we use tied word embeddings as described in Press and Wolf (2017). To avoid overfitting, training stops at 30 epochs for the RNN and 20 for the LSTM variants, or earlier when the validation perplexity stops improving. We record the epoch with the best validation perplexity. After selecting the best models by this validation perplexity, we then run the evaluation on the test set.

6.4.2 Generalization and Comparison to Baseline

Table 6.1 shows the overall results comparing our lowest achieved perplexities against the baselines. As can be seen, surprisal-based activation in an RNN (RNN+S) significantly improves the test perplexity when compared to the baseline (RNN) which can be interpreted as an overall better generalization. It is important to note that, using an extensive hyperparameter search, an unregularized RNN language model can be trained to test perplexities as low as 124.7 (Mikolov, 2012), which in turn would also improve our RNN-specific hyperparameters, and consequently also our RNN+S model. From the LSTM variants, S_h, S_{fc}, and S_c show a slightly better albeit comparable performance to a regular LSTM even though with a slightly better test result. S_{ch}, S_{ff}, and S_{fh} fail to improve the baseline while S_{ic} significantly outperforms the baseline by more than 7 PPL on both validation and test set. In the following sections, all

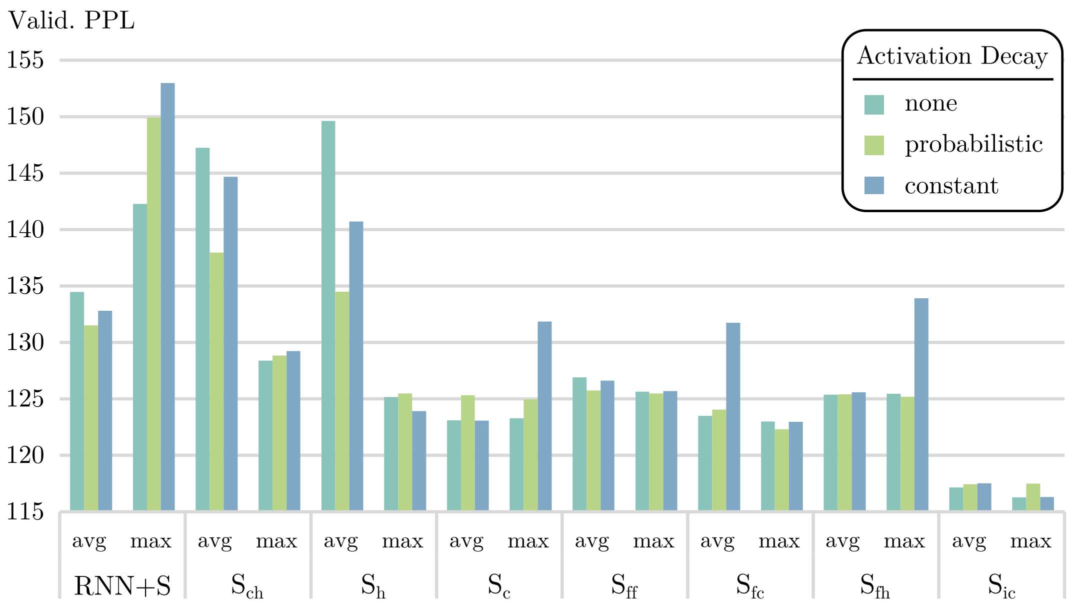


Figure 6.4: Comparing decay and pooling methods by validation perplexity for all proposed models.

hyperparameter analysis is presented based on the validation set in order to not overfit the test by model selection.

6.4.3 Pooling and Decay

The impact of pooling and decay can be seen in Figure 6.4. Average pooling yields better perplexities for the RNN+S. The negative effect of max pooling is, however, reduced for the LSTM+S, most likely due to a low numerical difference between the average activations in each module and their maximum. Especially in networks with more modules, max pooling has a more positive effect as there is a smaller information loss in pooling smaller modules. Average pooling turns out to be especially detrimental in the LSTM whenever the hidden state is preserved (S_h, S_{ch}), indicating that the activation variance is larger in the hidden state $\mathbf{h}_t$ than the cell $\mathbf{c}_t$. This seems different for the sigmoid-activated gates as all gate-based methods give a similar result for both pooling methods with max pooling yielding a better peak performance.

The RNN+S benefits the most from applying a probabilistic unit-level decay mask when using average pooling. It boosts the performance most significantly for S_h and S_{ch} which is in line with our expectation that an initially large activation variance can be toned down by locally applying a decay, lowering the maximum values and improving the average pooling effect. This could be potential evidence that random variations on a lower granularity than the pooling process might help with any surprisal-based "deadlocks" (see Section 6.3), especially when large modules are used which seemingly makes pooling difficult.

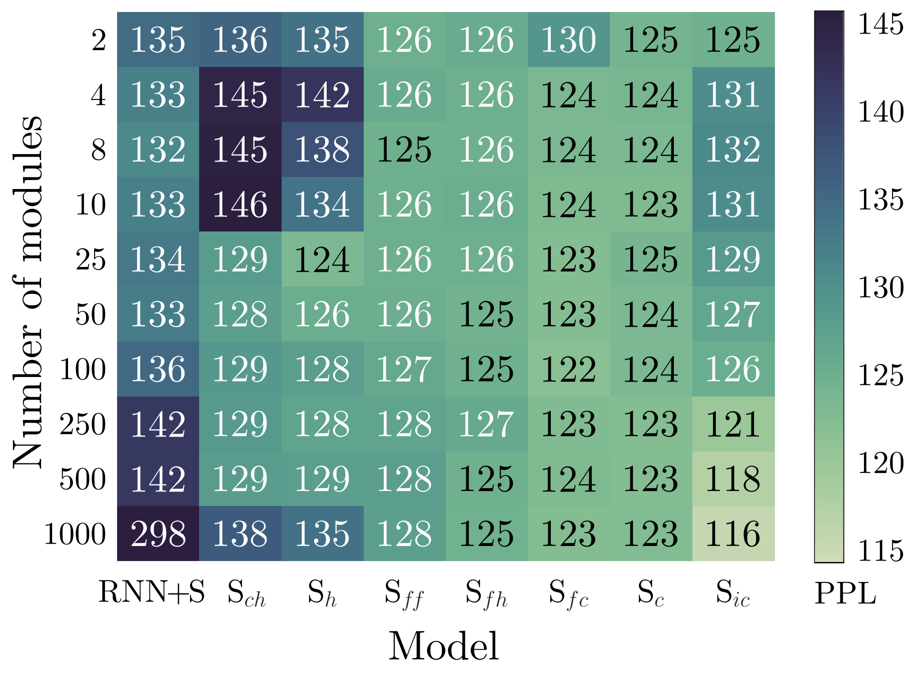


Figure 6.5: Heatmap showing validation perplexities (PPL) for the RNN+S and all LSTM variants with regard to the module choice M. While S_c gives the most consistent performance for all M, S_{ic} gives the overall best performance for $M \in \{500, 1000\}$ but struggles for smaller choices of M (as do S_{ch}, S_h, and RNN+S).

Applying the decay globally and at a constant rate has a worse effect for almost all models and shows itself to be an unreliable approach. For a few instances, it is on-par with having no decay at all. One potential explanation for these observations could be that a constant decay at best has no adverse (or beneficial) effect when the activations are converging towards lower values, and at worst, counteracts a change towards larger values. We would particularly expect the approaches based on preserving through the trainable forget gate to not benefit from a constant decay of the gate, as indeed seems to be the case.

Overall, the concept of decay turns out not to be vital for surprisal-based activation in LSTMs as long as the preservation happens through the forget or input gate. The probabilistic decay can potentially serve as a minor performance boost in these cases.

6.4.4 Number of Modules

Figure 6.5 illustrates the effect of the number of modules M on the performance. As can be seen, the LSTM variants S_{ch} and S_h are unable to converge to good solutions for $M < 10$. Since less modules equate to more units per module, the performance drop for $M < 10$ can additionally be explained by the larger area of effect of pooling which is consistent with our previous observations in Subsection 6.4.3 (see

S_{ch}	S_h	S_c	S_{ff}	S_{fc}	S_{fh}	S_{ic}	RNN+S
0.31	0.56	0.11	0.42	0.18	0.32	0.43	4.8

Table 6.2: Mean absolute deviation (MAD) of validation perplexities with different $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$, given the best configuration per model.

Section 6.5 for a deeper analysis). However, the opposite end of the spectrum, such as for $M = 1000$ (where each module consists of a single unit) shows that the absence of pooling worsens these two language models (S_{ch} and S_h) considerably. While this effect is even stronger for the RNN+S, the other models do not suffer from this issue. In fact, we can observe an inverse behavior for the best model, the LSTM+S_{ic}, which indicates that it might have quite different dynamics. This model shows itself to be only superior to the other methods for $M \in \{500, 1000\}$. Similar to the state-based preservation methods, its results are quite different for $M < 10$. The LSTM+S_{ic} variant outperforms the others and is best for $M = |\mathbf{h}|$. It also has the added benefit that, for this specific configuration, pooling is actually not performed (see Section 6.3) and we ultimately end up with less tunable hyperparameters and an overall simpler model with good performance. S_{ff} and S_c are the only variants which are consistent for all M where the latter gives an overall better result.

6.4.5 Threshold

The threshold hyperparameter θ ultimately proved to be quite robust to changes for the investigated task. After choosing the best hyperparameter configuration for each model, we have noted the mean average deviations (MAD) of their validation perplexities by looking at different choices of $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$. The results are reported in Table 6.2.

As can be seen, the deviation is largest for the RNN+S, mostly because $\theta = 10^{-7}$ gives better results for $M < 10$ than larger θ configurations. All LSTM variants show a larger robustness towards the choice of θ while the impact is lowest on S_c and highest on S_h. This seems to correlate with their ability to solve the given task. Therefore, the only LSTM configuration where a significant difference can be observed is for the S_h and S_{ch} networks with $M < 10$. As can be seen from Figure 6.5, this configuration for M leads to generally bad performance for these two variants, which partly explains this observation.

One potential explanation for the generally low impact of θ is that it is applied within the log-space of the state activations $\mathbf{c}_t$ and $\mathbf{h}_t$ (except for S_{ff} which directly measures the forget gate activation $\mathbf{f}_t$). This means that networks with larger absolute value changes between timesteps are naturally affected less by smaller θ while networks with smaller numerical variance can be forced by larger θ to preserve states more often. Following this assumption, our evaluation shows that our experimental setup operates below this cut-off point, i.e., the chosen θ are small enough to enable learning but also large enough to frequently trigger the activation preservation.

6.5 Further Analysis

6.5.1 State Analysis

As discussed in Section 6.3, our motivation for surprisal-based activation originates from the idea to utilize surprisal to temporally segment activations according to their change. Specifically, we look at increasing changes to apply state updates, and block updates for decreasing or low changes. In the context of language modeling, we hypothesize that this improves generalization by learning to preserve important context. This can, for example, happen when the language model is being presented with previously unseen or very rare words. Since wrong predictions can cause a cascade of errors, we want to be able to preserve context and skip timesteps with noisy and wrong predictions.

To investigate our hypothesis, we visualize the internal states. Figure 6.6 shows an example sequence which is given to the LSTM+S_{fc}. It can be seen, how measuring the surprisal values s_t on the cell state, helps in segmenting along some boundaries. For example, both `<eos>` tokens lead to a change in surprisal before the end of the sentence. The natural change of context that occurs at the sentence boundaries is also slightly reflected in the forget gate where a slightly larger activity can be observed in timestep 8.

A particularly interesting behavior can be observed between timesteps 13-16 where the input is: `<unk> conglomerate ssangyoung group`. The surprisal map shows very small values after processing `conglomerate ssangyoung` which leads to multiple triggered preservations in the forget gate, leading large parts of the network to skip over these rare words. Since only 2 out of 6972 occurrences for the word `<unk>` are followed by `conglomerate` in the training set, and `ssangyoung` only occurs once during training, this could in fact be explained by a likely misprediction of this previously unseen context. Since we use a softmax output layer, we believe mispredictions from rare inputs to cause mostly low and widely distributed hidden activations. Very certain predictions, on the other hand, would lead to specific units having high activations due to larger weights. When looking at the hidden activations $\mathbf{h}_t$ (top part of Figure 6.6), it is in fact difficult to recognize very strong localized activations, which supports the hypothesis that the network causes mispredictions during these timesteps. Assuming these mispredictions, we can see this behavior as quite beneficial, as the predictions starting at timestep 16 will have access to the preserved context from timesteps before 13, ignoring its previous mistakes and accessing context that it can utilize.

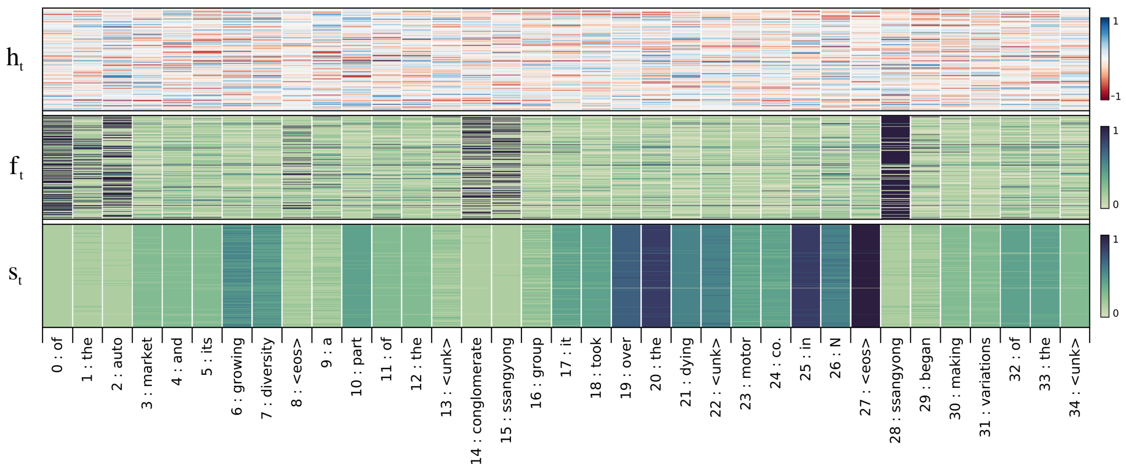


Figure 6.6: LSTM+S_{fc} with $M = |\mathbf{h}|$ processing an example sequence from left to right. The three illustrated maps are from bottom to top (same as the order of calculation): i) the measured surprisal s_t on the cell $\mathbf{c}_t$, ii) the forget gate activations $\mathbf{f}_t$ after applying $\mathcal{S}(\mathbf{f}_t, \mathbf{c}_t)$, and iii) the resulting hidden state activations $\mathbf{h}_t$.

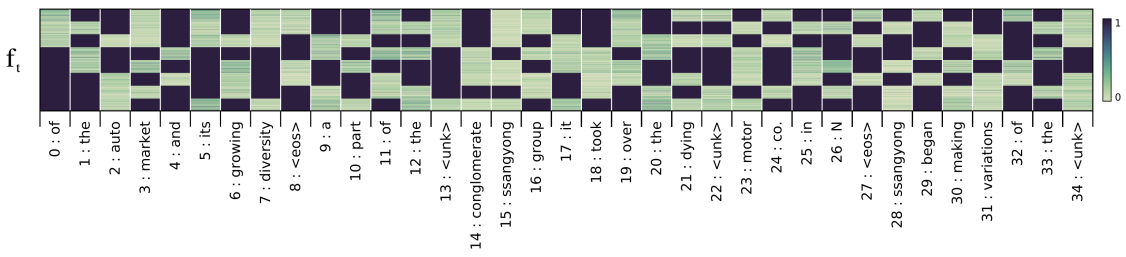


Figure 6.7: LSTM+S_{fh} with $M = 8$ processing an example sequence from left to right: For small M, the networks converge to a setup in which preservations are triggered around 50% of the time, resembling standard zoneout.

In order to deepen our understanding, we have also investigated examples that have performed worse than the baseline. As discussed in Subsection 6.4.4, these networks often have a low number of modules M. For these networks, we have found that surprisal-based activation is sometimes applied seemingly at random. Figure 6.7 shows the forget gate activations of the best trained LSTM+S_{fh} network that has $M = 8$ modules (i.e. each module has $1000/8 = 125$ units) and a validation perplexity of 126. It can be seen that preservation is seemingly applied randomly, resembling standard zoneout (Krueger et al., 2017). In fact, we have measured that preservation is applied in 50.11% of the cases for this network. The same network, but with $M = 50$, leads to a significantly lower percentage of preservation with only 6.38%. These findings could potentially explain previous observations where $M < 10$ can lead to worse performance than larger M. It also supports our hypothesis that surprisal-based activation works best when pooling over smaller groups of neurons. For all gate-based variants, setting $M = |\mathbf{h}|$ or

$M = |\mathbf{h}|/2$ to apply preservation on unit-level seems to be the most consistently safe approach. The LSTM+S_{ic} with $M = 250$ has preservations occurring 23.54% of the time. This percentage does, in fact, go up to 38.16% when raising the number of modules to $M = |\mathbf{h}| = 1000$, additionally lowering the test perplexity by about 5 points and resulting in our best recorded LSTM+S_{ic} model. From this we can see that frequently triggered preservations do indeed help in generalization when using larger M[3].

In conclusion, our overall results suggest that, in the context of our experiments, the LSTM cell state is more fit than the hidden state to implement surprisal-based activation. Both the S_{ch} and S_{fh} variants give worse results while operating on $\mathbf{h}_t$ than S_{fc} and S_c, which both observe surprisal changes in the cell $\mathbf{c}_t$. However, measuring and updating only $\mathbf{h}_t$ with the S_h variant, gives a stable performance, indicating that any manipulation of the gates should only be due to changes in the cell states (which is also supported by the worse results for S_{ff}. From our conducted analysis, we can conclude that surprisal-based activation works best by preserving the input gate using the cell state. This model, the $LSTM+S_{ic}$, has also no requirement to fine-tune the pooling method and number of modules M, making it more convenient to use and implement than the other LSTM variants.

6.5.2 Additional Experiments

As our evaluation has so far been focused on language modeling with the Penn Treebank corpus, we now scale up the task difficulty by training with the larger WikiText-2 dataset and aim to validate whether our previous results can also be transferred to other benchmark tasks such as numbers addition and sequence copying.

Experiment 1: WikiText-2

The WikiText-2 corpus is specifically designed to be as similar as possible to the Penn Treebank corpus while improving some of its downsides (Merity et al., 2017). Most importantly, WikiText-2 is over 2 times larger, features a richer vocabulary that is more than 3 times larger, and purposefully avoids preprocessing in order to preserve the original cases, punctuations, and numbers. In addition, it features slightly less out of vocabulary (OOV) words to be able to utilize more of the source text. Table 6.3 shows a full comparison between the statistical properties of both datasets.

We reuse the same optimization parameters and train each model with its best hyperparameter configuration established on the Penn Treebank dataset. The re-

[3]For a deeper investigation of the emerging update rates and their relationship to the SBA hyperparameters see Section 7.1.

	Penn Treebank			**WikiText-2**		
	Train	Valid	Test	Train	Valid	Test
Articles	-	-	-	600	60	60
Tokens	887,521	70,390	78,669	2,088,628	217,646	245,569
Vocab	10,000			33,278		
OOV	4.8%			2.6%		

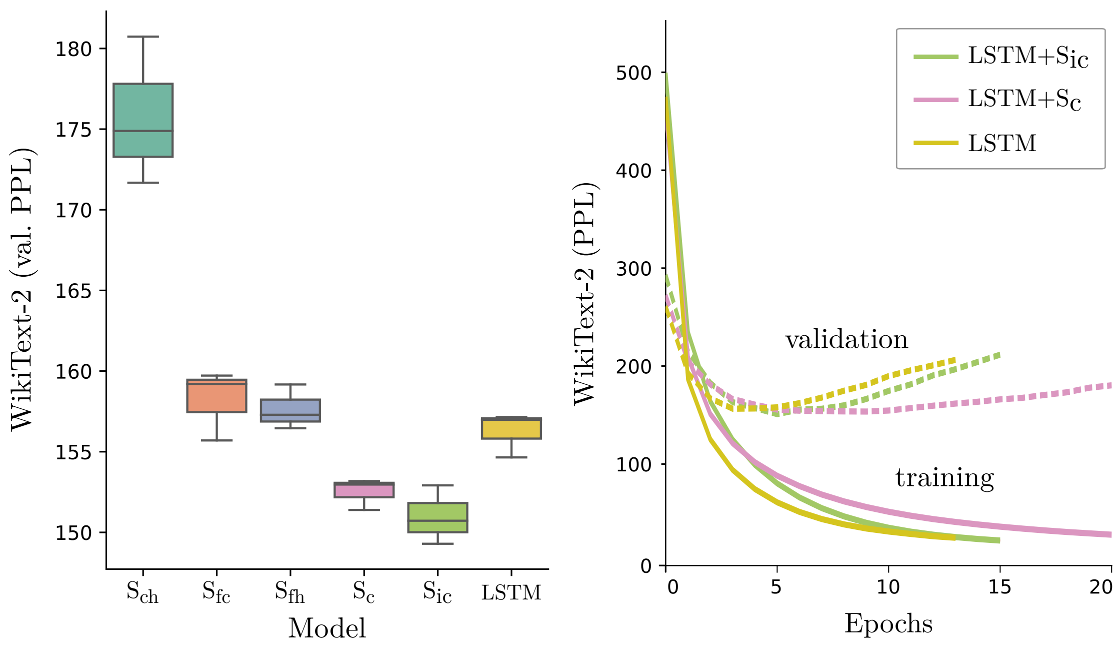
Table 6.3: Comparison between the Penn Treebank Corpus and WikiText-2 (Merity et al., 2017). Out of vocabulary (OOV) denotes the percentage of rare words replaced by an `<unk>` token.

Model	Best Val.	UR	Test
LSTM	154.65	-	144.74
LSTM+S_{ch}	171.68	0.50	156.93
LSTM+S_{fh}	156.45	0.44	145.89
LSTM+S_{fc}	155.67	0.98	145.22
LSTM+S_{c}	151.39	**0.10**	141.11
LSTM+S_{ic}	**149.29**	0.83	**138.79**

Table 6.4: WikiText-2 results: networks with minimum validation perplexity and their corresponding update rate (UR) and test perplexity.

sulting validation perplexities for all models are illustrated in Figure 6.8a. As can be seen, the networks rank like in the previous language modeling task, despite the slight increase in task difficulty, which is reflected by the overall worse perplexity that all models, including the baseline LSTM, achieve. The training curves shown in Figure 6.8b indicate that the S_{ic} and S_c seem to generally perform better than the baseline LSTM due to overfitting slower. This effect is strongest for the S_c which also explains why it has the lowest perplexity variance of all models.

The best recorded validation perplexities and the corresponding update rates (UR) and test perplexities of each model are summarized in Table 6.4. Test perplexity is generally about 10 PPL points better than validation perplexity, except for the S_{ch} which ranks as the worst model in our comparison. Most importantly, the S_c model achieves a very low update rate by performing only 10% of all possible updates and skipping most of its states. Compared to the LSTM (which has no skipping mechanism and therefore has a baseline update rate of 100%), this gives some initial evidence that SBA with the S_c model can lead to significantly more efficient models while simultaneously improving performance. To confirm this finding, we perform a deeper analysis of the update rates and the resulting trade-offs with model performance in Section 7.1.



(a) WikiText-2 Validation Perplexity **(b)** WikiText-2 Validation vs. Training Curves

Figure 6.8: WikiText-2 results. **(a)**: Boxplot comparing each model by validation perplexity. **(b)**: Learning curves for the LSTM, S_c, and S_{ic} models. S_c has a slightly slower learning process and is therefore in less danger of overfitting quickly.

Experiment 2: Addition Task

To evaluate the encoding and decoding capabilities of our approach, we investigate the sequence transduction task of addition. Each number can have up to 3 digits and each digit is randomly picked from 0 to 9, resulting in an input string such as "653+ 84". The actual network input is a concatenation of one-hot vectors of size 12 where each vector represents a digit of the addition sequence, and 2 tokens are additionally used for '+' and ' ' to mark an empty position such as in ' 84' due to a fixed sequence length. A sequence to sequence (seq2seq) model takes this input sequence and maps it to the correct output sequence of digits representing the correct answer. We generate 50000 unique sequences of which we set aside 10% for validation. Both encoder and decoder have 128 units and the same type of layer activated with tanh activation. For optimization, we use Adam (Kingma and Ba, 2015) with a learning rate of 0.001 and a mini-batch size of 512. We explore all possible module sizes of $M \in \{2, 4, 8, 16, 32, 64, 128\}$, average and max pooling, all 3 decay types (none, probabilistic, constant), and empirically determine the threshold to be $\theta = 10^{-4}$.

Different than the previous language modeling task, we focus our attention on evaluating convergence behavior and generalization speed for the variants, as all networks are able to solve this comparably simple task, given enough data and

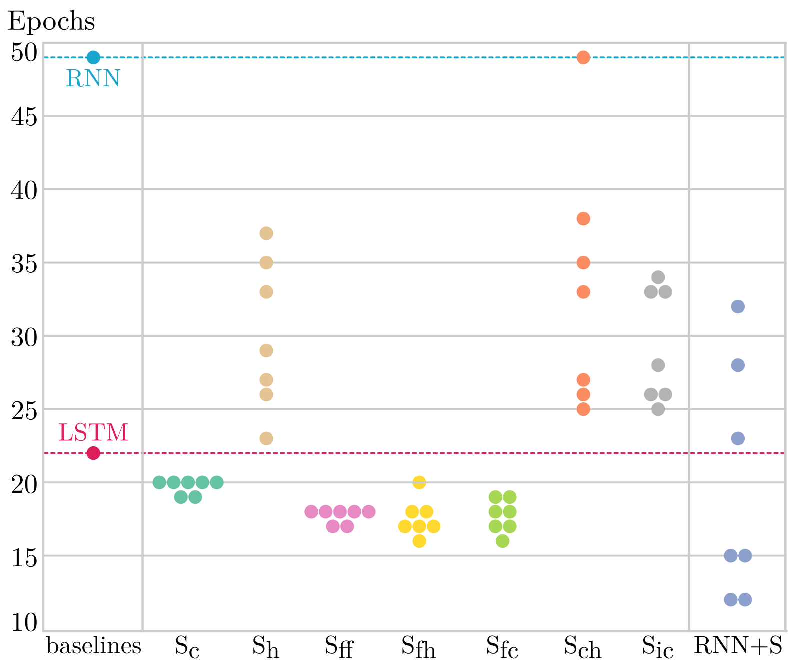


Figure 6.9: Addition Task: Swarm plot showing the number of epochs until the validation accuracy reaches 0.99 or greater. Shown are medians of the baseline LSTM and all trained LSTM+S variants where $M \in \{2, 4, 8, 16, 32, 64, 128\}$.

Model	S_c	S_h	S_{ff}	S_{fh}	S_{fc}	S_{ch}	S_{ic}	RNN+S
Avg	5.02	10.35	5.03	5.42	5.26	11.13	7.09	10.18

Table 6.5: Average number of epochs to reach optimal validation perplexity per model in the language modeling task.

iterations (but overall need longer than on the language modeling task). Figure 6.9 illustrates how the proposed LSTM variants based on the forget gate reach the validation accuracy threshold of 0.99 significantly faster than the baseline LSTM, independent of the chosen number of modules M. The S_c model behaves similarly, whereas the other variants converge slower and show larger variance due to a higher sensitivity to M. S_{ch} and S_h in particular, act very similar compared to the previous task. Furthermore, the impact of activation decay turns out to be negligible as it does not seem to change the behavior in a significant way.

When compared with the language modeling task, it can be seen that specific configurations on the RNN+S are able to actually converge the fastest to the accuracy threshold. The f-gate-based methods (S_{ff}, S_{fc}, S_{fh}) are very consistent, while the S_{ic} variant learns the task later than the LSTM. This is somewhat expectable as initially detrimental blocking of the input gate can lead to a need for more iterations to self-correct, leading to a potentially higher peak accuracy but a possibility of longer training times. Overall, the results show that surprisal-based activation is suitable for use in a seq2seq model and can improve the generalization time.

Since the RNN+S gave the worst performance in the language modeling task but reaches peak performance quickest in the addition task (while the S_{ic} performed best but requires more epochs than the forget-gate variants), it can be deduced that the time to convergence in the addition task is practically inversely correlated to the performance in the language modeling task. Comparing the results from Figure 6.9 to the convergence times from the earlier task (Table 6.5), confirms that the addition task reproduces the same general behavior that we have observed in the language modeling task.

Experiment 3: Copy Task

The purpose of the copy task is to test a model's capability to memorize a sequence (Arjovsky et al., 2016). The goal is to encode a random sequence of tokens that is to be repeated after a time lag. The copied subsequence is randomly sampled from a set of 9 characters (represented by the numbers 1-9), whereas the following time lag consists of a special "blank" symbol `X`. After the time lag, the network receives `0` symbols and needs to generate the previously read subsequence. As an example, given a random sequence `9132` that needs to be remembered over a time lag of 8 steps, the network receives `9132XXXXXXXX0000` as an input sequence and `9132XXXXXXXX9132` as the target.

While most recurrent networks can solve this task easily given enough iterations and shorter time lags, its difficulty can be scaled with increasing time lags until the networks can no longer solve the task. Additionally, we can define a limited number of iterations to achieve a specific accuracy threshold, penalizing networks that need too much time until convergence. Consequently, both constraints allow us to compare the networks by their memory limitations. Therefore, we utilize the copy task to evaluate how our integration of SBA into the LSTM impacts its long-term memory capabilities and, more specifically, whether SBA improves the LSTM's ability to model long-term relationships.

We approach this task by designing a seq2seq model, i.e. an encoder-decoder network, where the encoder converts the initial sequence into a single hidden vector which is then used by the decoder to sequentially predict the output sequence. We choose a length of 8 characters for the copied sequence and vary the task difficulty by using time lags of 8, 16, 24, and 32 timesteps as we find that networks start to encounter significant memorization issues around 32 timesteps. All characters are encoded as one-hot encodings using a softmax output and Cross Entropy loss. Training is done using Adam optimization with an initial learning rate of 0.001, batch size of 32, and hidden layer sizes of $|\mathbf{h}| = 128$ for encoder and decoder, respectively. We generate 10000 sequences of which we use 90% for training and the remaining 10% for holdout cross-validation, and compare all LSTM variants of SBA to a baseline LSTM. After an initial grid search with $N = 5$ trials per network configuration, we pick the best parameters for the model variants and note the point at which most networks start to fail the task, namely around time lags

	Time Lag			
Model	8	16	24	32
S_c	14.6	17.8	20.6	27.2
S_{ch}	28.2	27.4	25.4	33.6
S_{ff}	18.6	30.8	46.4	58.4
S_{fc}	15.0	19.0	24.2	30.2
S_{fh}	16.6	24.4	52.75	58.5
S_h	24.0	22.0	21.4	29.2
S_{ic}	19.2	26.2	47.8	—
LSTM	8.4	22.4	34.8	—

(a) Average number of epochs to achieve an accuracy of 90%. All networks have a 100% success rate (SR) on this threshold, with the exceptions of the LSTM and S_{ic} (SR=0 for TL=32) and S_{fh} (SR=80% for TL=24 and SR=40% for TL=32)

	Time Lag			
Model	8	16	24	32
S_c	100	100	100	100
S_{ch}	80	80	80	20
S_{ff}	100	100	40	0
S_{fc}	100	100	100	80
S_{fh}	100	100	20	0
S_h	40	40	100	60
S_{ic}	100	100	40	0
LSTM	100	100	80	0

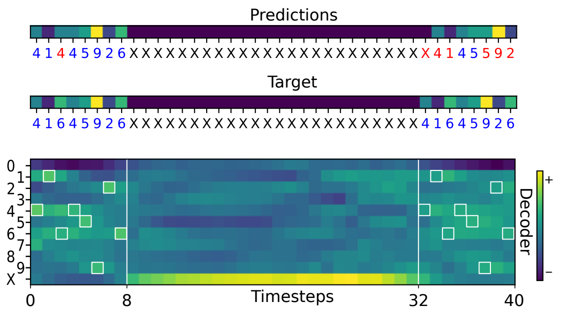
(b) Success rate for achieving a 95% accuracy threshold within the given time frame.

Table 6.6: Copy task results. The longer the Time Lag (TL) between reading (encoding) and copying (decoding), the harder the task. **(a)**: Almost all networks (137/140) converge above 90% accuracy, even for longer time lags. **(b)**: The success rate of achieving 95% accuracy is significantly lower for longer time lags. For the maximum lag, only the S_c model is able to always converge at high accuracy.

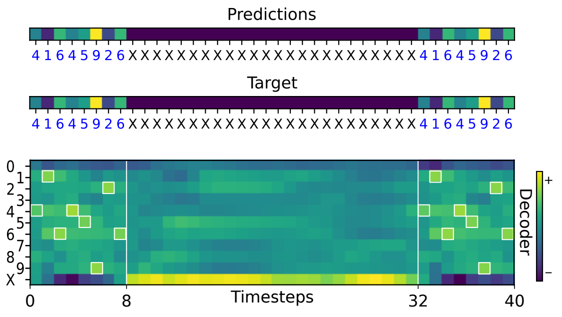
between 24 and 32 steps and an accuracy threshold of 95% to consider a successful run for a single network within 75 epochs.

Table 6.6 summarizes the results of the experiment. Most networks are able to solve the task rather quickly with time lags of 8 and 16 steps, but start to fail with increasing time lags. Most networks achieve a 100% success rate (i.e. each of the N trials is successful) of achieving an accuracy of 90% (Table 6.6a). However, within the same time frame, most are unable to reach 95% accuracy with time lags of 32 timesteps (Table 6.6b). Additionally, we can observe the trend that increasing the time lag also increases the required training time. Overall, these findings indicate that long time lags impact the convergence process in that it is more difficult to reach the top accuracies in the same amount of training time.

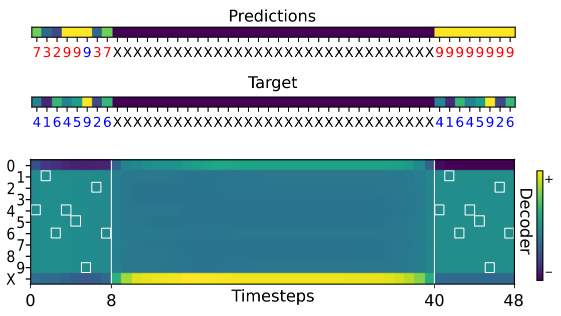
Comparing our SBA variants to the baseline LSTM, we can see that the LSTM is significantly faster to solve the task with the shortest time lag. However, for the longest time lag, none of the trained LSTMs reach the required accuracy threshold, thereby failing the task. On the other hand, most SBA models achieve accuracies



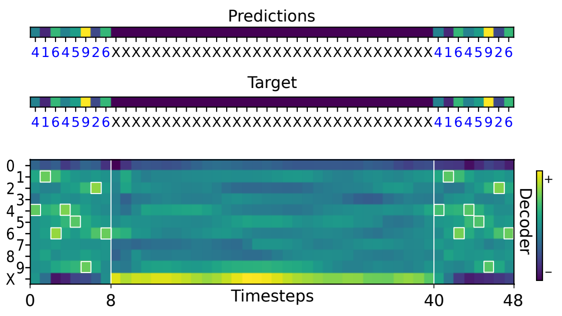
(a) LSTM (lag: 24)



(b) LSTM+S_c (lag: 24)



(c) LSTM (lag: 32)



(d) LSTM+S_c (lag: 32)

Figure 6.10: Copy Task: Comparison between an LSTM decoder (left) vs. the S_c model (right) for time lags of 24 (top) and 32 (bottom) timesteps. The locations of the correct targets are marked as white boxes in the activation maps. Targets and correct predictions for the copied subsequence are marked in blue, wrong predictions in red. The two white vertical lines segment the initially read subsequence (first 8 steps) and the blank middle section from the final subsequence (final 8 steps). The goal is to match the final segment with the beginning.

between 90%-95% for the longest time lag. This indicates that the LSTM is more biased towards short-term relationships, whereas SBA seems to be more biased towards long-term relationships.

Comparing the different variants, we can find that S_c outperforms all other networks in that it never fails one of the tasks, even with the highest time lag and accuracy threshold. It is followed by the S_{fc} model, which converges similarly fast. S_{ic} is the only SBA variant to never reach accuracies above 90% on the largest time lag, whereas S_h gives the most consistent performance, often converging around the same number of epochs regardless of task difficulty, but having the lowest overall success rate for the highest accuracy threshold.

To see the difference between a baseline LSTM and SBA, we visualize the activation maps of the respective decoders. Figure 6.10 compares the activations of the S_c model against the baseline LSTM based on the shown representative examples. For time lags of 24, most LSTMs are able to encode the initial sequence and produce some sequence near the end, even though the copied output often contains mistakes (as shown in Figure 6.10a). The S_c model, on the other hand, is always able to encode both the initial subsequence as well as its copy (Figure 6.10b). Compared to the LSTM, the respective characters have much larger activations, indicating a greater confidence in the overall prediction. The activations of the LSTM have a smaller numerical variance, especially in the final 8 steps of the sequence, indicating a stronger tendency to converge towards an average of the tokens. For the longest time lag (32 steps), the LSTM fails to copy the sequence. Instead, it learns to encode the middle section, presumably as the blanks are most frequent in the input sequence. This seems to hinder its capability to properly encode the subsequence at the beginning and end (Figure 6.10c). The S_c model accomplishes this task without mistakes (Figure 6.10d), giving a similar solution to experiments with smaller time lags.

Overall, the results for the copy task provide additional evidence that integrating SBA with the memory gates of the LSTM improves its capability to learn long-term dependencies and memorize inputs over longer time spans. The differences between the S_c model and the other SBA variants are slightly larger than in previous tasks, as this model consistently converges much quicker to higher accuracies. This indicates that the S_c model is better at memorizing and decoding. The SBA variants modulating the forget gates, however, show more difficulties for this particular setup, suggesting that the forget gate itself plays an important role for memorization in SBA, particularly as the baseline LSTM has more difficulties with this task.

6.6 Chapter Summary

In this chapter, we have introduced a novel method to preserve activations and skip updates in RNNs based on surprisal. By investigating and evaluating multiple LSTM integration variants, we have also demonstrated that a baseline LSTM can be improved by skipping based on surprisal constraints and by modeling memory decay of skipped representations through the forget and input gates. With the LSTM+S_{ic}, we have presented an accurate model when applying surprisal-based activation on unit-level ($M = |\mathbf{h}|$), eliminating the necessity to fine-tune most of the introduced hyperparameters.

Our approach provides additional evidence to the overall utility of preserving activations and skipping state updates with neural prediction models. In particular, the presented analysis shows the potential of using surprisal to detect and segment context boundaries. However, further research is needed to conclusively identify whether the detected boundaries do indeed lead to emerging hierarchical structures. As our evaluation was limited to a small set of prediction tasks, it is also necessary to explore SBA with other types of tasks, particularly classification. As demonstrated, surprisal-based activation can be integrated and combined with different types of recurrent layers. In particular, it can easily be extended to GRUs

(Chung et al., 2014), which would lead to an additional reduction in complexity. Consequently, we introduce the Surprisal-based GRU (SGRU) and evaluate it in conjunction with more advanced architectures on question answering in Chapter 9.

The evaluation in this chapter provides an initial view into the potential of SBA even though some open questions remain. It is particularly desirable to have more control over the trade-off between accuracy and update rate as the ideal balance between both metrics is different for each situation and application. For this, a deeper understanding is necessary on how the update rates develop during training and how they are affected by the hyperparameters. The development of methods to directly control update rates based on scenario requirements would facilitate this further. The focus of the next chapter will be to investigate these questions, developing the introduced model further.

Chapter 7

Bridging the Gap Between Surprisal-Based Activation and Zoneout

In this chapter, we will take a closer look at controlling trade-offs between skipping and model performance, investigating how we can reduce activations without negatively impacting the performance. For this purpose, we examine the relationship between skipping with Surprisal-based Activation (SBA) and regularization-based skipping. Our goal is to provide a better understanding towards the question whether skipping should be driven by regularization or as part of a model's update function. Our evaluation will focus on two main aspects related to regularization: i) the regularization capabilities of SBA, and ii) the use of regularization penalties to reduce the number of state updates. As zoneout (Krueger et al., 2017) is the most popular regularization method based on skip mechanisms, we provide a comparison between SBA and zoneout. As part of this analysis, we run a large ablation study, successively removing the differences between SBA and zoneout.

An overview of our methodology is provided in Figure 7.1. We start by thoroughly exploring the naturally emerging update rates of the introduced SBA variants and how they can be reduced further for additional sparsity gains (Section 7.1). In this context, we introduce different regularization penalties to the objective function (Section 7.2) and evaluate how much control they give us over the network behavior. The role of penalties is to provide a middle ground between precise control over predetermined update rates (as in zoneout) and the naturally emerging update rates of SBA which are performance-driven and not directly controllable. Following this, we turn SBA into pure regularization method and observe how this affects generalization (Section 7.3). Finally, we turn zoneout regularization into a model architecture which allows us to explicitly define update rates, and compare this to the other approaches (Section 7.4). To our knowledge, this study constitutes the largest variety of different zoneout masks investigated to date.

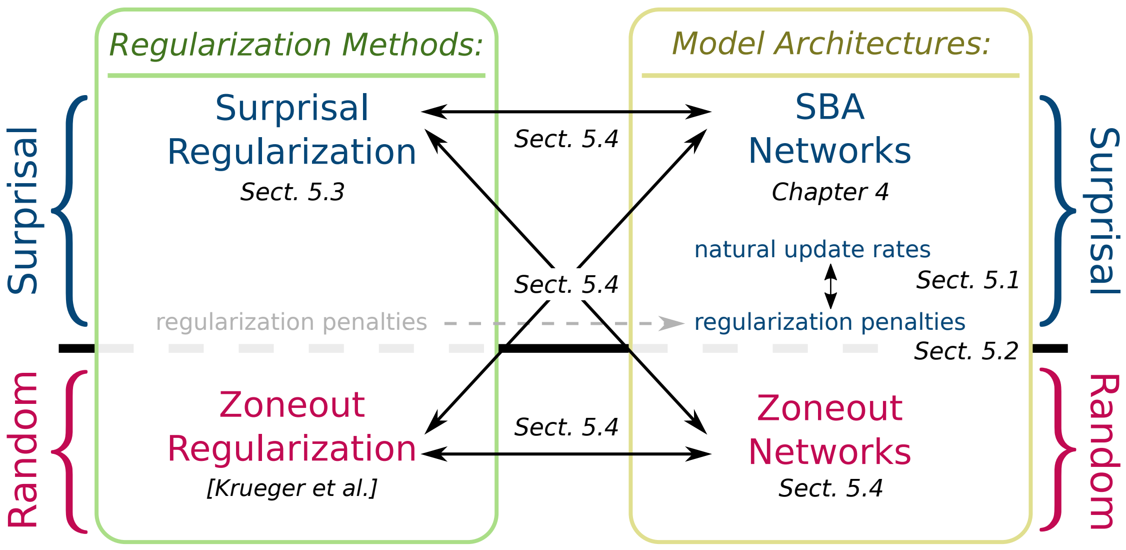


Figure 7.1: Overview over this chapter's methodology to bring together zoneout and SBA in the context of both skipping models and regularization methods.

7.1 Convergence of Update Rates

Before trying to influence the update rates of our models to minimize network updates, we will first systematically investigate which update rates emerge naturally during training. We informally define the *update rate* u as the ratio of the number of updated states to the number of skipped states. For example, an update rate of 0.6 means that 60% of all state updates proceeded normally while 40% were skipped (averaged over all units, batches, and timesteps). Introducing this second metric allows us to quantify the skipping behavior and evaluate the relationship to each model's performance (using perplexity as the performance metric).

To analyze the model update rates, we repeat the main hyperparameter search from Chapter 6 and additionally compute the update rate of each model. The resulting update rates of all models are illustrated in Figure 7.2 and put in relation to task performance (PPL) and number of modules $M \in \{2, 4, 8, 10, 25, 50, 100, 250, 500, 1000\}$. What can be seen is that the S_{ic} model, previously found to give best task performance, is actually one of the models with both a higher update rate and a larger range (from 0.5 to 0.7). Only S_{ic} and S_{fc} have a similarly large update range. For the other models, the update rates converge to a specific point that seems to be difficult to influence significantly with a different choice of M. Using this new perspective of evaluating the models under two different metrics, allows us to also reevaluate S_c: while this model is only the second best performance-wise, it can achieve this with half as many state updates on average ($u_c = 0.33$) than the best S_{ic} ($u_{ic} = 0.58$). For $M = 2$ it maintains the same performance (from 123.7 PPL to 124.1 PPL) but pushes its update rate further down to $u_c = 0.26$. Consequently, compared to a regular LSTM, this model allows

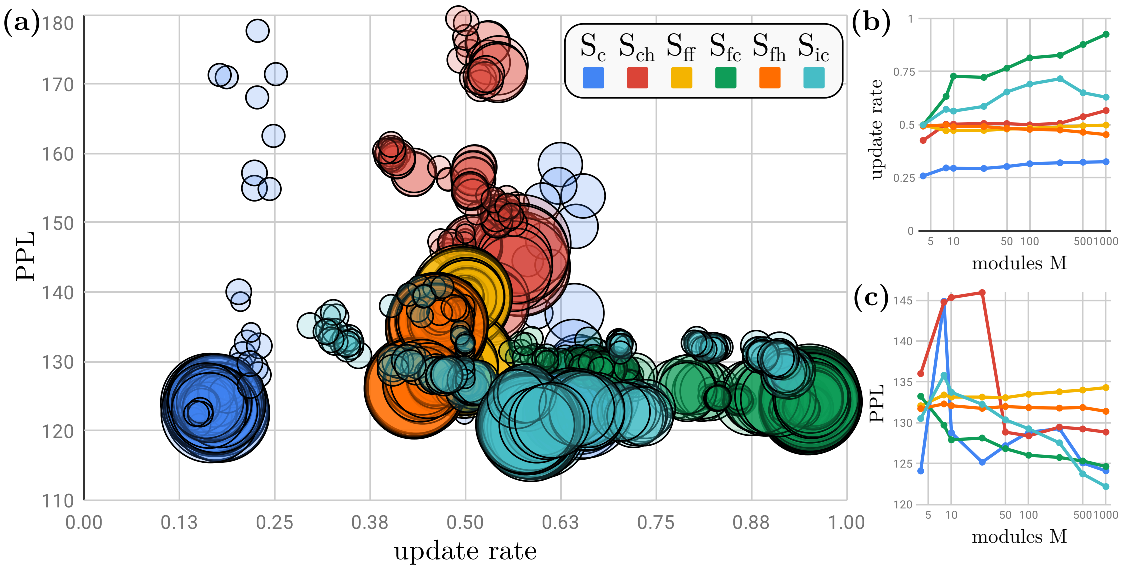
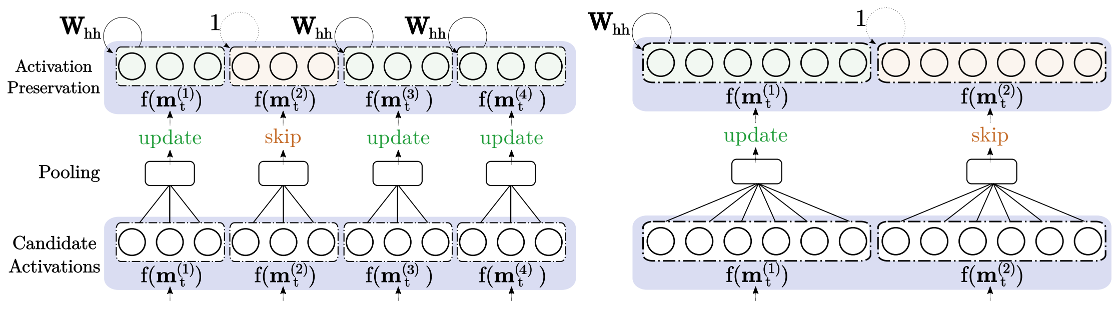


Figure 7.2: (a) Valid. PPL vs. update rate for each model. Bubble size scales with number of modules (small size indicates small M). The **(b)** best average update rate vs M and **(c)** the PPL vs M can also separately be seen on the right side (x-axes for M scaled logarithmically).

us to maintain baseline performance while additionally preventing 74% of all state updates on average.

What can also be seen in Figure 7.2 is that, with growing M, there is a strong logarithmic trend in the growth of the update rates for S_c ($R^2 = 0.81$), S_{fc} ($R^2 = 0.87$), a logarithmic decrease for S_{fh} ($R^2 = 0.91$), and a slightly weaker trend for S_{ch} ($R^2 = 0.65$) and S_{ic} ($R^2 = 0.58$). The S_{ff} model is the only one where this trend isn't as clear, although this is mostly likely caused by the hyperparameter having less influence on this model (see Subsection 6.4.4).

This shows a generally strong tendency towards a logarithmic correlation between the naturally emerging update rates and the number of modules used in each model: the smaller our choice for the number of modules M (i.e. more units are grouped together in less modules), the smaller the network's update rate. We hypothesize that this is primarily caused by the pooling mechanism. To illustrate this with an example (see Figure 7.3), we can compare the differences between partitioning a layer with 12 units into $M = 4$ modules and into $M = 2$ modules. If the layer size remains unchanged, $M = 4$ is equivalent to 3 units per module and $M = 2$ to 6 units per module. Since the decision to update or skip is done on a per-module-basis (after pooling), it follows that the network with $M = 2$ will keep more units inactive for each skip decision compared to the network with $M = 2$. Consequently, the modularization via M practically determines the possible range of update rates. The network with 2 modules can (for each timestep) only achieve update rates of $0, 0.5$, and 1, whereas the network with 4 modules has more fine-grained control and can achieve update rates of $0, 0.25, 0.5, 0.75$, and 1.



(a) SBA with $M = 4$ modules.
Possible update rates: $0, 0.25, 0.5, 0.75, 1$.

(b) SBA with $M = 2$ modules.
Possible update rates: $0, 0.5, 1$.

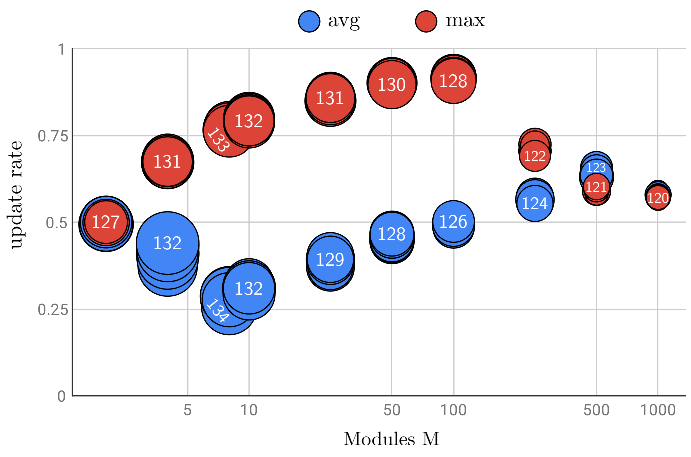
Figure 7.3: Effect of number of modules M on SBA illustrated on two examples with $|\mathbf{h}| = 12$. **(a)** $M = 4$ causes a smaller number of units to skip for each decision. **(b)** Each skip in $M = 2$ affects twice as many units as the pooled vectors are larger. The network on the left has a more fine-grained control over its update rates and empirically leads to larger average per-unit update rate than the one on the right.

The fact that this more powerful network has a higher update rate can partially be explained by the fact that update rate and accuracy seem to be positively correlated: the network tries to optimize for a cross entropy loss and will therefore achieve a better outcome with larger update rates close to 1[1]. It is worth noting that this also affects generalization as we observe similar update rates on training, validation, and test sets.

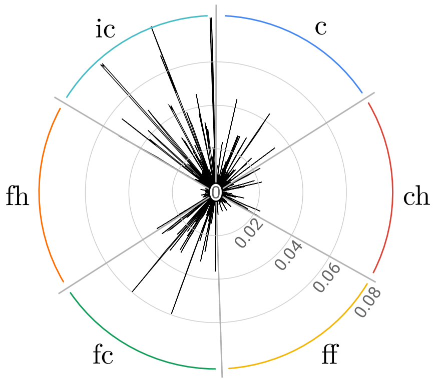
Nevertheless, as we have seen in Figure 7.2, most SBA models with $M = |\mathbf{h}| = 1000$ converge to update rates significantly lower than 1 (S_{fc} being the exception). Our observation can therefore not fully be explained by the loss function favoring large update rates. As a second reason, we can identify that the larger area of pooling, caused by small M, additionally leads to more skip decisions for average pooling. This is evident from the fact that outlier activations will be averaged out stronger in a larger vector, leading to a lower change in surprisal. Max pooling, on the other hand, has an increased chance of catching an outlier in a larger vector, leading to more changes in the surprisal. Empirical support for this is illustrated in Figure 7.4a for the S_{ic} model and our overall analysis sees this pattern present for the other models as well. Consequently, we conclude that max pooling causes more skips for *smaller* modules and average pooling causes more skips for *larger* modules. This difference gets smaller with growing M.

From related work (see Section 3.6), we can find that decreasing update rates can lead to decreased performance. Aside from the aforementioned explanations, it is also evident that a strong decrease in activations equates to a loss in information.

[1]This is why an optimization for mere performance can be problematic if one desires to minimize metrics unrelated to task performance (such as the number of updates). The next section will therefore investigate how the loss function can be modified with an additional optimization goal unrelated to the task performance.



(a) LSTM+S_{ic} update rates (M vs pooling).



(b) MAD of update rates.

Figure 7.4: (a): The effect of average (blue) vs. max (red) pooling on the update rate and number of modules M for the model LSTM+S_{ic}. Bubble size and text labels indicate model perplexity. Shown are best achieved update rates per model. Max pooling causes more skips for smaller modules, average pooling causes more skips for larger modules ($M = 1000$ equals no pooling operations). **(b)**: Radial chart of the mean absolute deviations (MAD) of the update rates, recorded over all hyperparameter configurations (each line equals the MAD of 3 runs on one configuration). Even though the update rates are affected by hyperparameters, the overall deviation is very low, both generally and for each SBA variant, respectively.

The more the encoding is compressed, the likelier it is to become lossy. Ultimately, this means that, even though we can influence the network into small update rates by choosing small M, this will in many cases also lead to suboptimal performance. Nevertheless, the inverse relationship between update rate and performance means that there is a potential optimal trade-off point somewhere in the middle that can be found by either hyperparameter optimization or careful design of the objective function.

In order to find networks with a satisfying trade-off, we propose an initial model selection strategy with a stronger bias on accuracy. First, we only consider networks without activation decay and with the smallest θ. This helps us to simplify the process as we have previously shown that these parameters have a limited influence on the best models. Then, we select the remaining 3 configurations (with different M and pooling) by the lowest perplexity achieved throughout 3 runs. From these we choose the model with the lowest update rate u.

The results of this process are presented in Table 7.1. Earlier we had noted that S_c is able to skip 74% of all updates on average. As we can see, we can now identify networks (using max pooling and $M = 2$) with an even lower update rate at $u = 0.14$ (mean $\mu_u = 0.14$, standard deviation $\sigma_u = 0.0009$), i.e. 86% of all possible updates are not performed. The important thing to note is that this

Model	Val. PPL	u	M	pooling
LSTM	123.6	1.0	–	–
LSTM+S_{ch}	143.18	0.50	2	max
LSTM+S_{ff}	124.95	0.46	4	max
LSTM+S_{fh}	125.47	0.49	25	max
LSTM+S_{fc}	121.73	0.92	50	max
LSTM+S_{ic}	**119.73**	0.58	1000	–
LSTM+S_c	120.67	**0.14**	2	max

Table 7.1: Models with the lowest update rate u, selected from the three model configurations with the lowest perplexity (PPL). No activation decay applied. While the S_{ic} is consistently accurate, the S_c is significantly more effective at the cost of a little accuracy.

network still performs better than a regular LSTM (by about 3 PPL) and has 4 times as much activation sparsity as the S_{ic} ($u = 0.58$, $\mu_u = 0.14$, $\sigma_u = 0.0049$) even though it only sacrifices about 1 PPL in performance. This makes the S_c model an ideal candidate with both good performance and very sparse activity. Generally, these results indicate that the two SBA variants S_c and S_{ic} are able to avoid the aforementioned trade-off between performance and skipping. However, this seems to only be the case for specific hyperparameter configurations and the other SBA variants even fall slightly behind the baseline performance.

Aside from hyperparameter influence, all models have in common that there is a significantly low variance in their update rates (see Figure 7.4b). Regardless of hyperparameter configuration, no model has an MAD over 0.08 in their update rate. Taking our previous analysis into account, of how both pooling and module size have measurable effects, but only shift the update rate significantly from the mean in very specific setups, this shows us that update rates are largely fixed to certain ranges partly determined by the hyperparameters. While the large variety of models and hyperparameters in our study allow us to cover a large spectrum of behavior, gradient descent seems to cause the models to always converge to a specific update rate that satisfies the training objective. As the objective is based on task performance and not update sparsity, adapting it for this purpose is a potential solution that we will investigate next.

Intermediate Conclusion

To summarize, there are three main conclusions from this experiment:

First of all, task performance and update rate are generally positively correlated. While a small amount of skips can improve the performance, a large reduction of activity towards 0 degrades accuracy noticeably for most of the models.

Second, adjusting the number of modules and the pooling operator allows us some level of control when trying to find a good balance between accuracy and the number of state updates although it requires significant analytical work in the currently proposed approach. The only model with potential for very good accuracy *and* computational efficiency is the S_c, which performs better than a regular LSTM even though it saves 86% of all state updates.

Nevertheless, the hyperparameter influence is limited, which leads us to the third finding: given a specific hyperparameter configuration, update rates converge to a specific point with very little variance. When optimizing for task performance, this gives us models that perform similar to the baseline (sometimes even slightly better) but provide additional activation sparsity through skipping.

However, it is possible to come up with hypothetical scenarios in which we might accept even stronger losses in model accuracy as long as the resulting model gives a different benefit such as, e.g., increased computational efficiency. Similarly, we can imagine scenarios where it is more important to maintain baseline accuracy and any additional update sparsity is optional. Since the underlying choices of such a trade-off are highly situational and entirely dependent on the use case, it is consequently necessary to develop additional methods which allow us to more directly and conveniently define what kind of trade-off the model should optimize for. In the following sections we will therefore explore additional methods to more explicitly describe to the network our notion of an optimal balance between task-performance and skipped updates. In the next section, we will therefore start with introducing a second objective into the objective function in the form of regularization penalties which we explicitly link to the update rate that we want to control.

7.2 Minimizing Computations with Update Penalties

Our primary research goal is to reduce update rates as much as possible while maintaining accuracies close to the baseline. However, until now, SBA training has only been influenced by the cross entropy loss and the metrics that we use for model selection. This causes a bias towards maximizing accuracy over minimizing update rates. Therefore, we will next introduce and apply different regularization penalties to indirectly affect the update rates in our models.

In the previous section, we have determined that optimizing task-performance and minimizing the number of state updates can be two adversarial objectives. Consequently, it is necessary to have mechanisms with which to describe the desired trade-off in order to find an appropriate network by training with gradient descent rather than manual selection after a hyperparameter search.

In this section, we will explore if these two goals can be balanced by gradient descent as long as they are both represented explicitly in the objective function. This means that we define a second objective $\mathcal{R}$ which we add to our cross entropy loss function $\mathcal{L}_{CE}$:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{R} \tag{7.1}$$

Even though it is not directly tied to the goal of improving generalization, we call this second objective $\mathcal{R}$ "regularization penalty" since we aim to penalize state updates and achieve a form of sparsity regularization[2] in the process. As a result, we work with a more general definition of regularization in the context of this thesis: a penalty term that (during training) shapes the solution into a desired form according to some constraint that is **not** tied to the training error or the goal of improving generalization[3]. For our specific use case, we will use the term "*update penalty*".

As our central approach, we propose to apply recurrent stability regularization for the novel context of minimizing computations with SBA. As we will show, this regularizer is complementary to the main mechanism of SBA. While this approach has not been used for minimizing computations before, there are other regularizers that have been studied with similar architectures. Therefore, we test our method against two additional methods, which have have established themselves well enough to serve as baselines to determine which method finds the best trade-off between performance (perplexity) and computational resources (update rate). The first of these approaches is to use a computing budget (Campos et al., 2018) or ponder cost (Graves, 2016) which simply accumulates the number of updates or computational steps and adds them to the loss. The second approach is to define a target update rate and penalize the mean deviations from it (similar to the objective function of the Leap-LSTM (Huang et al., 2019)). Each of these three different regularization penalties is aimed at minimizing the computational cost.

7.2.1 Regularizer 1: Computational Budget ($\mathcal{R}_{CB}$)

We refer back to the previously introduced conditional update for SBA (see Equation 6.19) which leads to $\mathcal{S}(\mathbf{g}_t, \mathbf{k}_t)$ updating the gate $\mathbf{g}_t$ based on the surprisal $\mathcal{I}(\mathbf{k}_t)$ of gate $\mathbf{k}_t$, and more specifically, whether it exceeds a certain threshold such that $\mathcal{I}(\mathbf{k}_t) > \mathcal{I}(\mathbf{k}_{t-1}) + \theta$. Using the Iverson bracket notation, we can now quantify each of these binary decisions $\mathcal{S}(\mathbf{g}_t, \mathbf{k}_t)$ such that each positive if-condition (i.e.

[2]Sparsity regularization requires the same trade-off between sparsity and accuracy.

[3]This more general perspective on regularization penalties has also recently been "rediscovered" in literature on learning disentangled representations where regularization penalties are for example introduced for information maximization instead of preventing overfitting (e.g., see (van Steenkiste et al., 2019))

update) leads to a 1, and each negative if-condition (i.e. skip) leads to a 0:

$$[\![\mathcal{I}_t]\!] = \begin{cases} 1 & \text{if } \mathcal{I}(\mathbf{k}_t) > \mathcal{I}(\mathbf{k}_{t-1}) + \theta \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

Quantifying the logical proposition used in the condition of Equation 6.19, allows us to directly count the number of decisions in a sequence of inputs with $0 < t \leq T$. We define this sum as the computational cost of a sequence:

$$\mathcal{R}_{CB} = \lambda \cdot \sum_{t=1}^{T} [\![\mathcal{I}_t]\!], \tag{7.3}$$

where the hyperparameter λ determines the regularization strength. This computational budget (CB) penalty $\mathcal{R}_{CB}$ is then added to the cross entropy (CE) loss function such that $\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{R}_{CB}$ satisfies Equation 7.1. This penalty somewhat resembles L^1 regularization which penalizes large weights. The main difference is that L^1 serves to prevent overfitting by increasing *representational sparsity* (setting weights to 0) while our penalty increases *computational sparsity* (setting recurrent activations to 0). In addition, our constraint is not primarily motivated by the goal to reduce overfitting, even though we would expect the sparsification of activations to result in a better generalization due to a reduced influence of redundant parameters.

7.2.2 Regularizer 2: Target Update Rate ($\mathcal{R}_{MSURE}$)

The idea of the second regularizer is to minimize the update rate towards a manually specified target. For this purpose, we define the target update rate $u_t \in [0, 1]$ as a new hyperparameter and utilize it as follows:

$$\mathcal{R}_{MSURE} = \lambda \cdot (u_t - u)^2, \tag{7.4}$$

where, as previously, λ determines the regularization strength and u is the actual update rate $u = \frac{\sum_T [\![\mathcal{I}_t]\!]}{T}$ over all sequences. Since the update rates are averaged over the entire dataset, $\mathcal{R}_{MSURE}$ is effectively the mean squared error (MSE) for the update rate (UR), which we therefore name the "mean squared update rate error" (MSURE).

One potential issue with this regularizer is that it requires, in addition to λ, a second hyperparameter u_t. From a practical standpoint, it might be the most intuitive approach to set u_t close to 0 ($u_t >_\epsilon 0$). After all, this would ensure that the update rate u is minimized as much as possible. However, we hypothesize that this is actually not the case. To illustrate the reason, we can consider the local minimum to which the cross entropy loss $\mathcal{L}$ converges at the end of training. As shown previously, each SBA variant has its own update equilibrium to which it converges with little variance, assuming that we only optimize for task-performance and to

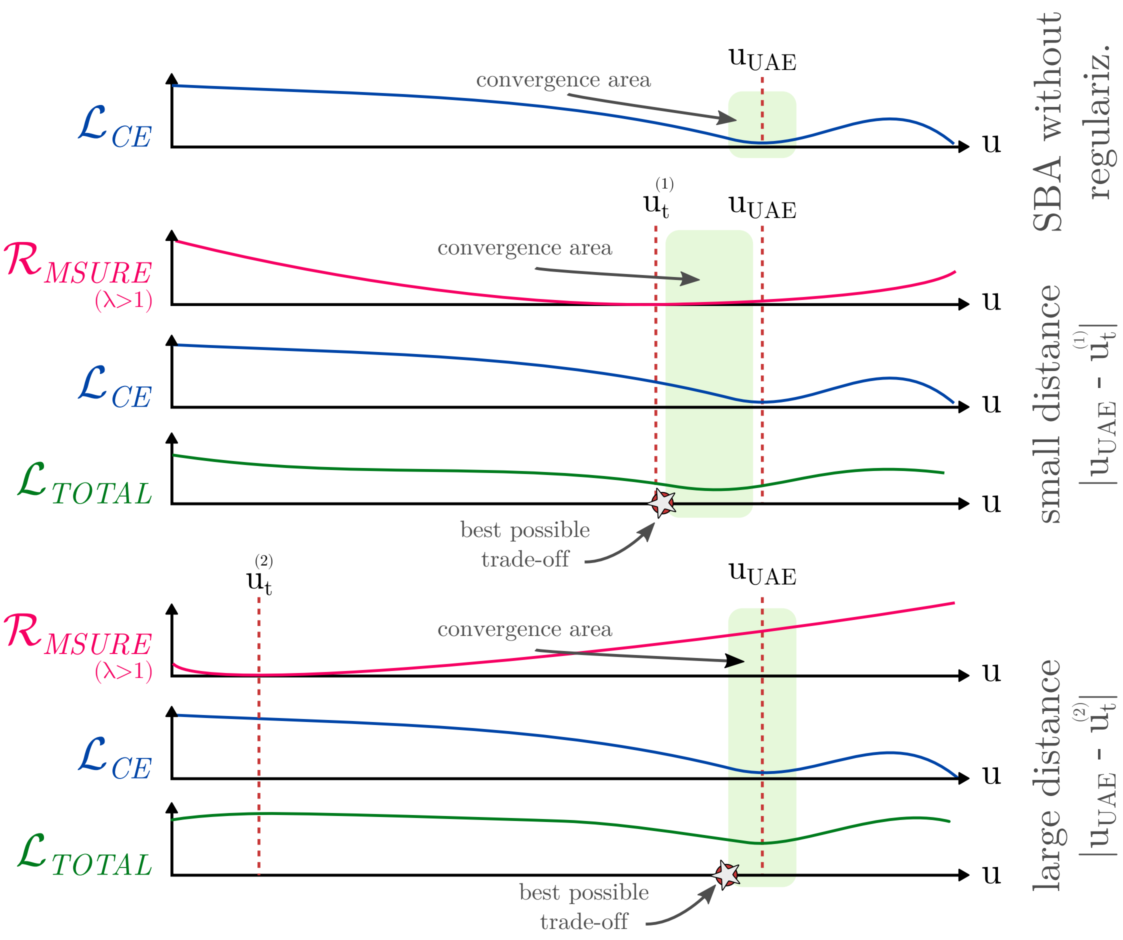


Figure 7.5: Example scenarios with idealized update-loss curves illustrate how a lower target update rate u_t might counterintuitively lead to larger update rates than targets with a larger value that are however closer to u_{UAE}. Example assumes that λ is chosen such that both objectives have roughly equal contribution in $\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{R}_{MSURE}$.

no other secondary objective. Let us define the update rate that occurs naturally, when optimizing for the primary metric, as the *update-accuracy equilibrium* (u_{UAE}). Then, the goal of a secondary objective is to push the update rate u_t away from u_{UAE} towards 0 (in particular, until the loss from the decreasing accuracy gets too large).

Figure 7.5 illustrates how the relationship between $\mathcal{R}_{MSURE}$ and $\mathcal{L}_{CE}$ influences $\mathcal{L}_{total}$. Without any regularization penalty, the area of the convergence is by definition $u_{UAE} \pm \epsilon$ for some small ϵ that defines the valley of the local CE loss minimum. It follows from the definition of $\mathcal{R}_{MSURE}$ that it is a parabola with a vertex (minimum penalty) at the target rate u_t and an openness scaled by λ. Let us now assume that λ is chosen in a reasonable manner such that L_{CE} and R_{MSURE} have roughly the same contribution towards $\mathcal{L}_{total}$. Then, if the distance between these two minima ($|u_{UAE} - u_t|$) is small, the smallest value for u would be at the lower border of the convergence area (because the area itself is shifted towards 0 due to the update penalty). On the other hand, if this distance is large, $\mathcal{R}_{MSURE}$ will have a larger contribution near u_{UAE} and since the combined loss is additive, the convergence area will shift towards higher update rates u in general.

In conclusion, assuming that λ is chosen in such a way that one objective is not weighted significantly more than the other, the best possible trade-off between accuracy and update-rate is accomplished whenever the target rate u_t is close to the equilibrium defined by u_{UAE}.

7.2.3 Regularizer 3: Norm Stabilization ($\mathcal{R}_{NS}$)

We will now turn to norm stabilization, an activation regularization method proposed by Krueger and Memisevic (2016) and previously discussed in the context of the stability-discrimination dilemma (see Section 2.5). Norm stabilization is defined as the average squared difference between the L^2 norm $\|\cdot\|_2$ of two consecutive hidden activations $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$, measured over a time window $t = (1, ..., T)$:

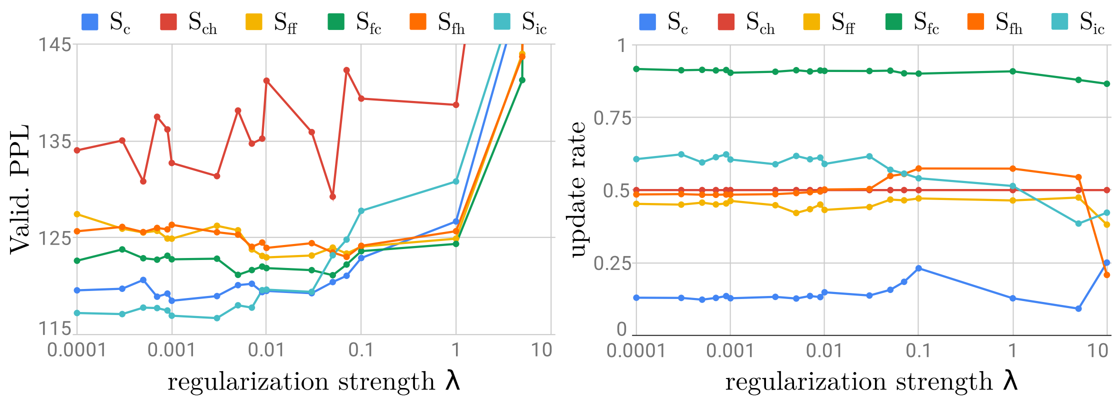
$$\mathcal{R}_{NS} = \lambda \cdot \frac{1}{T} \sum_{t=1}^{T} \Big(\|\mathbf{h}_t\|_2 - \|\mathbf{h}_{t-1}\|_2 \Big)^2, \tag{7.5}$$

where λ is the regularization strength parameter familiar from the other regularizers. While the previous two regularization penalties $\mathcal{R}_{CB}$ and $\mathcal{R}_{MSURE}$ achieve a minimization of the update rate by directly tying the update rate to a penalty, norm stabilization is originally unrelated to update rates. Instead, it minimizes the state difference between timesteps. Since SBA makes update decisions based on the state changes between timesteps (expressed in surprisal), norm stabilization ends up influencing the update rate when used in conjunction with SBA. Therefore, $\mathcal{R}_{NS}$ can only be considered an update penalty when used with SBA.

This regularizer facilitates slow feature analysis as it penalizes large state changes between consecutive timesteps. This fits perfectly with the SBA mechanism which only permits updates with large state changes and suppresses small changes into inactive skip states. Consequently, we hypothesize that combining both should result in state changes getting smaller and therefore triggering more state inactivity, further reducing computation.

7.2.4 Experimental Setup

All three regularization penalties are measured over the sequences of the entire dataset and added to the training loss. Naturally, the regularization strength $\lambda \in (0, \infty)$ differs for each method as they all operate on different numerical scales. In order to find each regularizer's suitable range for λ values, we run initial experiments, recording how much the update penalties influence the total loss. We



(a) λ vs. valid. PPL **(b)** λ vs. update rate

Figure 7.6: Norm Stabilization Regularizer ($\mathcal{R}_{NS}$)

call this the regularization influence RI and express it as the following ratio:

$$RI = \sum_{i=0}^{N} \frac{\mathcal{R}_i}{\mathcal{L}_{CE,i} + \mathcal{R}_i}, \tag{7.6}$$

where N is the total number of iterations over which we sum both the loss and penalty values.

Looking at the RI values for different λ configurations, we have identified for each regularizer the numerical range for λ in which RI is roughly in the range $[0.1, 0.9]$. For $\mathcal{R}_{MSURE}$, we explicitly test the target rates in the range $u_t = [0.1, 0.9]$ with increasing step sizes of 0.1, giving 9 different combinations. Due to a significantly larger hyperparameter space compared to other setups in this chapter, we have chosen the best hyperparameter configuration for each SBA variant (i.e. predefined M, θ, d, pooling function) such that any observed variance is only from the regularization parameters.

7.2.5 Results

The effect of λ on the Norm Stabilizer is illustrated in Figure 7.6. As can be seen, for $\lambda > 0.05$, all networks degrade in performance as RI gets close to 1. In particular, $\lambda > 1$ causes the error to go up significantly for all models. This in turn causes the update rate to go down for all models except S_c. This model seems to actually be *negatively* correlated with the task performance (decreased accuracy leads to increased update rates). As we have observed in Section 7.1 that the u_{UAE} of S_c is very low compared to the other models, the update rate of S_c is evidently difficult to improve further using this update penalty.

For lower values of λ, the Norm Stabilizer is unable to effectively reduce the update rate (Figure 7.6b). A comparison with the corresponding perplexities (Figure 7.6a)

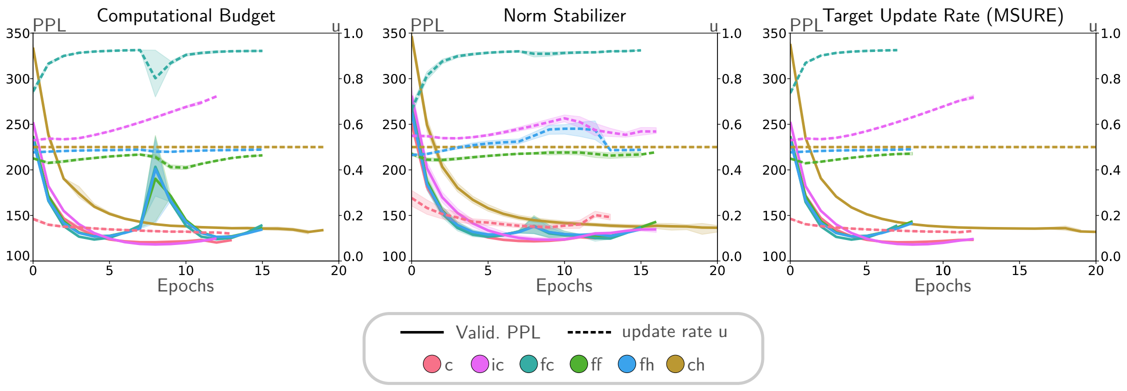


Figure 7.7: Validation perplexity (solid lines) and update rate (dashed lines), recorded after each epoch for the three proposed penalties. Band shows 95% confidence interval.

indicates that the networks are successfully converging, mostly according to the cross-entropy objective. Interestingly, we have observed this same behavior for all three of the regularizers.

Figure 7.7 shows a comparison between all three regularizers. In most cases, the Norm Stabilizer seems to lead to a slightly better loss. In some cases (e.g., S_{ic}) it also leads to better update rates at the time of early stopping. R_{MSURE} starts similarly but stops the training process earlier for all models (except S_{ic}). The variants skipping based on the forget gates climb in perplexity after dropping initially. At this point in training, between epoch 7 and 10, R_{CB} actually continues training, whereas R_{MSURE} stops due to the resulting spike being short enough to not be detected by the monitoring window in early stopping.

Our results also suggest that the choice of the target u_t for $\mathcal{R}_{MSURE}$ has had little influence on the actual outcome of the update rate, independent of parameterization. One potential explanation for this could be that the update rate convergence area (compare discussion regarding Figure 7.5) is simply too narrow. This means that the overall variance is simply so low that the point of optimal trade-off is very close to u_{UAE}.

However, since all three methods are unable to significantly push down the update rate during training and validation, it stands to reason that there is a more comprehensive issue at hand that prevents the regularizers to change the update rates significantly. As the methods themselves have been shown to work with other models, it is most likely that the penalties simply can not achieve the desired goal in conjunction with the core mechanisms of SBA. An alternative explanation could be that the penalties only work in a very small range of λ (comparable to budget regularization in the Skip RNN (Campos et al., 2018)) which our experiments were not able to pick up.

As our regularization penalties seem to be insufficient to reduce update rates further while maintaining baseline accuracy, we will therefore explore in the next section how SBA itself can be turned into a regularizer and whether this allows a more successful approach to lowering each model's update rate.

7.3 The SBA Regularizer

In the previous sections, we have approached SBA from the perspective of designing a neural architecture. In order to investigate the regularization capabilities of the main mechanisms of SBA, we now turn to transforming it into an actual regularization method.

With the goal of testing the capabilities of SBA to act as a traditional regularizer, we keep the original training process with the SBA models but use regular LSTM activations for the validation and test process. The main point of this experiment is to probe the generalizability of the skip decisions themselves. In other words, if testing with a regular LSTM improves the model, then the generalization was worse using SBA during testing. Fundamentally, this tells us how much the SBA representations improve generalization to unseen examples in LSTMs.

To distinguish this new setup from the previous, we call our original SBA models SBA+M in contrast to the newly introduced regularizer variants SBA+R. We keep our main experimental setup from Subsection 6.4.1 and run the same grid search as we did previously, but keep all units are active during validation and testing. Interestingly, our results show that the overall influence of the SBA hyperparameters is quite low. Measured by MAD, the choices for both pooling and the threshold parameter seem to have a small effect. Only S_{ch} and S_{ic} show MADs over 1 PPL for pooling (8.40 and 4.79 respectively) which is similar to that of regular SBA models.

For all models, constant decay has the same effect as having no decay, while probabilistic decay decreases performance significantly for S_c, S_{ch}, S_{ff}, and S_h. From this, it can be inferred that activation decay is not helpful if SBA is only applied at training time. To an extent, it also gives us evidence that the previously observed positive effects of activation decay mostly lead to improved generalization with the SBA mechanism. Since it has no positive effect if the testing is performed with regular LSTMs instead of LSTM+S variants, it is likely that regular LSTMs can mitigate representational irregularities introduced by SBA, while LSTM+S slightly benefits from activation decay.

The only parameter that seems to be important for SBA regularization is the number of modules M. The relative improvements from turning SBA into a regularization method are illustrated in Figure 7.8. The differences between SBA+R and SBA+M are significant for most of the models. Moreover, while the total deviation along M is high, there are no trends observable that are independent of each model, except for $M = 4$ being the best choice on average. Generally, lower M seem to give better performance, which is contrary to the original SBA models.

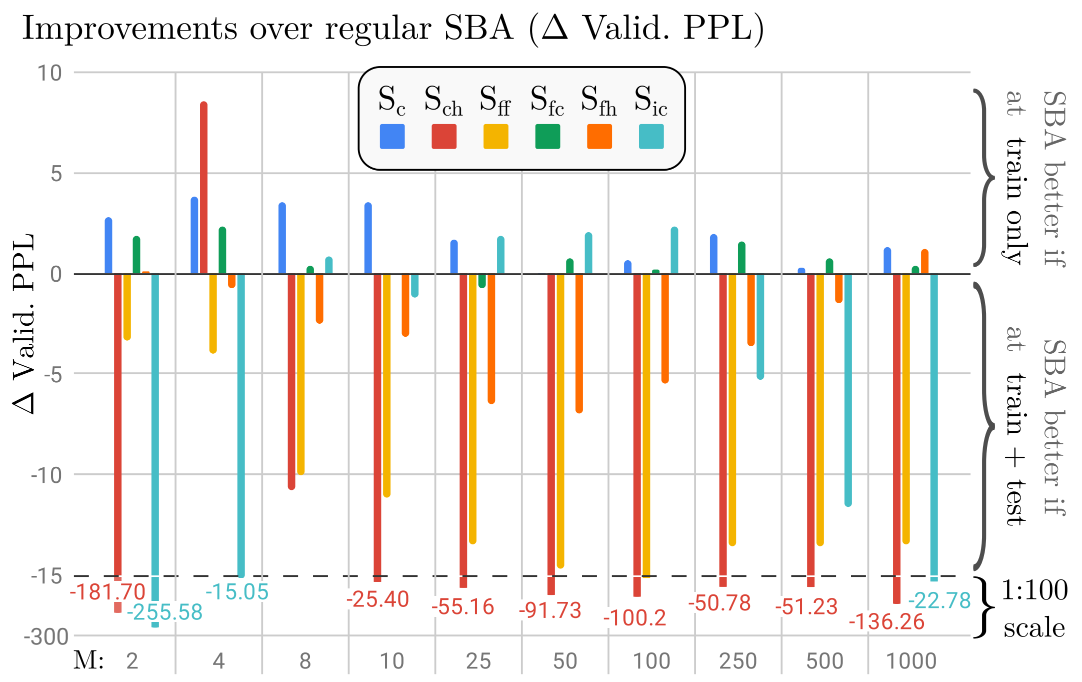


Figure 7.8: Relative validation PPL improvements (Δ Valid. PPL) when SBA is only applied during training instead of both training and testing. $\Delta > 0$: Model performance improves when using SBA only for training (SBA+R), $\Delta < 0$: Model performance improves when using SBA in both training and testing (SBA+M baseline). Comparison is shown by number of modules M. Bottom part of the chart ($\Delta \in [-300, -15]$) is scaled down 1:100 to improve readability.

S_{ic} seems to only converge properly when M is neither low nor high, while S_{fh} only improves for *either* low or high M.

Looking at the overall top test results of the entire hyperparameter search (see Table 7.2a), restricting SBA to training time (SBA+R) leads to a measurable effect on most models, whether it results in an increase or decrease in the top performance. The most noteworthy change can be seen from the LSTM+S_c which improves by 6.8 PPL and supersedes the LSTM+S_{ic}. The LSTM+S_{ic} model is the only model which significantly changes its update rate by more than 30%. We hypothesize that this is mostly because of an increase in training time (the update rate grows monotonically during training of LSTM+S_{ic}).

The large improvement in perplexity for SBA+R+S_c leads to the conclusion that the original model SBA+M+S_c is worse at generalizing with the SBA mechanism than a regular LSTM. Since the LSTM is able to make better use of these representations *without* any training, it stands to reason that the LSTM simply has more access to memory than the more restricted LSTM+S_c variant. This is also supported by the fact that the model seems very sparse with an update rate of only 14%.

This gives us an indication that optimizing for two antagonistic objectives (accuracy vs computation) might generally benefit from changes to the early stopping procedure which traditionally only monitors the loss. If the training procedure only stops based on task-performance, a secondary objective, such as e.g. minimizing the update rate, remains unmonitored. In addition, the results show that the update rates themselves generalize well to regular LSTMs and that S_c representations transfer even better to a regular LSTM than SBA itself during the testing phase. While this tells us that some of our models do indeed have innate regularization capabilities (particularly S_c), we still lack a clear way to directly control the update rates and have also not fully answered the question on how far we can push the update rates away from their natural equilibrium.

In the next section, we will introduce the possibility for having full control over the update rates by stochastically sampling update decisions similar to zoneout. Furthermore, we will perform the opposite procedure from the previous experiment where we turned our models into regularizers: since zoneout is a regularization method, we transfer the stochastic update process to models running at training and test time and evaluate the resulting differences.

7.4 Fusing SBA and Zoneout

As the results of earlier sections have shown, explicitly minimizing the number of state updates is difficult to achieve with regularization. While this could either result from SBA itself or the suggested approaches, we have also stated that there might inherently be model-specific lower bounds that prevent the update rates from decreasing beyond a certain level unless we accept a significant tradeoff w.r.t. model performance. To investigate this experimentally, we require the ability to set the update rates manually. This can help in conclusively determining whether the update rates, that are adaptively learned by SBA, can indeed be improved further beyond the equilibrium u_{UAE} or if they are indicative of a model-specific lower bound.

7.4.1 Sampling Random Update Decisions

To set the update rates of the networks in a precise and controlled manner, we draw some inspiration from the regularization method zoneout[4]. The main mechanism that we adopt is to sample random decisions, which we aim to compare against our adaptive surprisal-based update decisions from SBA.

[4]See Subsection 3.5.2 for an introduction to zoneout.

Consequently, we change our main update decision function (Equation 6.9) to the following:

$$Z(\mathbf{h}_t^{(i)}) = \begin{cases} F(\mathbf{x}_t, \mathbf{h}_{t-1}^{(i)}) & \text{if } \mathbf{b}_t^{(i)} = 1 \\ \mathbf{h}_{t-1}^{(i)} & \text{if } \mathbf{b}_t^{(i)} = 0, \end{cases} \tag{7.7}$$

where $\mathbf{b}_t \in \{0,1\}^{|\mathbf{h}|}$ is a binary decision gate vector and $\mathbf{b}_t^{(i)} \in \{0,1\}$ represents the i-th element of this vector (with $|\mathbf{b}_t| = |\mathbf{h}_t|$). We sample the decision gate from the Bernoulli distribution with *update probability* p:

$$\mathbf{b}_t \sim Bernoulli(p), \tag{7.8}$$

whereas the distribution itself is defined such that $P(Bernoulli(p) = 1) = p$ and $P(Bernoulli(p) = 0) = 1 - p$. Naturally, each element of $\mathbf{b}_t$ is binary and models the decision process of updating vs skipping for each unit i. Consequently, we temporarily abandon the concepts from SBA about grouping units in modules, pooling, and thresholding (activation decay on the other hand is still possible).

Similar to Equation 6.19, we can generalize zoneout such that we explicitly define which gate or state is being preserved during inactive periods:

$$\mathbf{k}_t^{(i)} = \mathcal{Z}(\mathbf{k}_t^{(i)}) := \begin{cases} Z(\mathbf{k}_t^{(i)}) & \text{if } \mathbf{b}_t^{(i)} = 1 \\ \mathbf{k}_{t-1}^{(i)} & \text{if } \mathbf{b}_t^{(i)} = 0, \end{cases} \tag{7.9}$$

where $\mathbf{k}_t$ can either be the LSTM cell state, hidden state, or one of the memory gates. We denote the resulting models as Z_k, respectively.

Applying the activation preservation mechanism of $\mathcal{Z}(\cdot)$ instead of $\mathcal{S}(\cdot)$, results in our different SBA variants "collapsing" to multiple new variants of zoneout. For reference, the original zoneout algorithm applies the skip mechanism to both the cell state $\mathbf{c}$ and the hidden state $\mathbf{h}$. We get to this variant by making S_{ch} stochastic, which we denote as $Z_{ch} := S_{ch}$. Furthermore, with the introduction of stochastic decision sampling, the forget-gate based variants automatically become the same model $Z_f := S_{ff} = S_{fh} = S_{fc}$, while $Z_i := S_{ic}$ and $Z_c := S_c$. This gives us 3 novel zoneout approaches as well as the original method (Z_{ch}). As noted, zoneout is conceptually equivalent to S_{ch} with random decision sampling at training time. A small difference in our implementation for Z_{ch} is that we use shared masks between $\mathbf{c}$ and $\mathbf{h}$ which is not the case for regular zoneout (Krueger et al., 2017). We choose this alternative approach for consistency and to minimize the differences to SBA.

Since zoneout is a regularization technique that is only applied at training time, we mirror our experimental setup from the previous section where we have turned our SBA models into regularizers. Specifically, we run separate experiments for i) zoneout regularization (ZO+R) where testing is being done by a regular LSTM without sparse activation and ii) zoneout models (ZO+M) which additionally skip during testing.

It is important to note that the ZO+M models are *not* performing random guessing at test time since we only subject specific components of the LSTM to random decision sampling. For example, while Z_f will cause random forget gate behavior, we are still able to remember the trained input gate and use it to process each input $\mathbf{x}_t$, which leads to an LSTM output that is not entirely affected by the gate's randomness. Since the network has the means to explicitly learn how to overcome this additional noise during training, we hypothesize that the resulting model is capable of generalization.

7.4.2 Experiments and Results

For our experiment, we evaluate Z_{ch}, Z_f, Z_i, and Z_c with update probabilities of $p = [0.1, 0.2, ..., 0.9, 1.0]$ and our 3 activation decay schemes. Due to the large number of sampled decisions, we are able to observe during training that update rates quickly approximate their expected value, i.e. the update probability p such that $u \approx p$ for ZO+R and ZO+M. Overall, we have found that masking any parameter other than the hidden state leads to successful improvements over the baseline LSTM. Since exploding gradients have led to significant difficulties training Z_h, we have not investigated this variant further as model-specific changes to the optimization procedure would have compromised a fair comparison.

The resulting test perplexities are presented in Table 7.2b, side-by-side with the previous results for the SBA+R regularizer and the original SBA model (SBA+M). One of the most important findings is that the adaptive mechanism of SBA generally leads to slightly smaller update rates in models compared to those with zoneout (ZO+M). This indicates that SBA is not only able to capture the optimal average update rate (which we measure through a grid search on zoneout), but is capable of additional optimizations. Since zoneout yields skip decisions from a uniform distribution and SBA decides for each unit based on its temporal and spatial context, we see this tuning capability of SBA as the main explanation for the smaller update rates.

However, we also observe the opposite relationship for the perplexity metric, i.e. zoneout models (ZO+M) perform better than SBA and even zoneout regularization itself. While this may seem somewhat surprising, it can partially be explained by the fact that the slightly better update rates of SBA require a trade-off with accuracy. Comparing the zoneout training regularizer (ZO+R) to the zoneout model (ZO+M), we can see that our zoneout model leads to both better perplexities and update rates compared to all variants of the standard zoneout setup.

In an overall comparison between zoneout and SBA models, SBA leads to smaller update rates while zoneout leads to smaller perplexities on the best model configurations. Comparing the zoneout training regularizer (ZO+R) to the zoneout model (ZO+M), we can see that our zoneout models outperform their respective regularizer variants with regard to to both perplexities and update rates. Furthermore,

Model	SBA+M	u	SBA+R	u
LSTM	123.6	1.00	–	–
$+S_{ch}$	135.9	0.50	131.3	0.59
$+S_{ff}$	124.7	0.46	125.1	0.50
$+S_{fh}$	125.0	0.49	121.8	0.44
$+S_{fc}$	121.7	0.92	119.3	0.96
$+S_{ic}$	**119.2**	0.58	120.2	0.90
$+S_{c}$	119.9	**0.14**	**113.1**	**0.21**

(a) Learned update rates

Model	ZO+M	p	ZO+R	p
$+Z_{ch}$	124.9	0.6	137.5$^{1)}$	0.9
$+Z_{f}$	118.8	0.7	**117.1**	0.8
$+Z_{i}$	116.0	0.7	123.0	0.9
$+Z_{c}$	**113.3**	**0.2**	117.6	**0.3**

(b) Controlled update rates

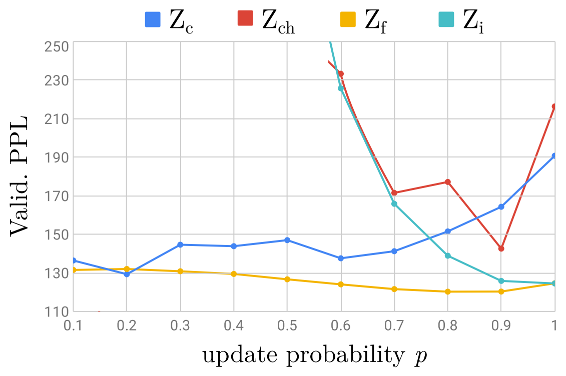
Table 7.2: p : zoneout probability, u: average update rate of the best model configuration. Shown are test perplexities of the best results, independent of decay rate. SBA+M and ZO+M are models applying their skip mechanism on training and test time, SBA-R and ZO+R are training regularizers applied at training time only. Note that u is a measured metric, whereas p is a hyperparameter with $p \approx u$. $^{1)}$: ZO+R+Z_{ch} is most similar to the original zoneout approach.

if we consider Z_{ch} as our baseline (since it is the closest to the original zoneout approach), we can also recognize that all of our introduced concepts drastically improve on the original zoneout regularization approach.

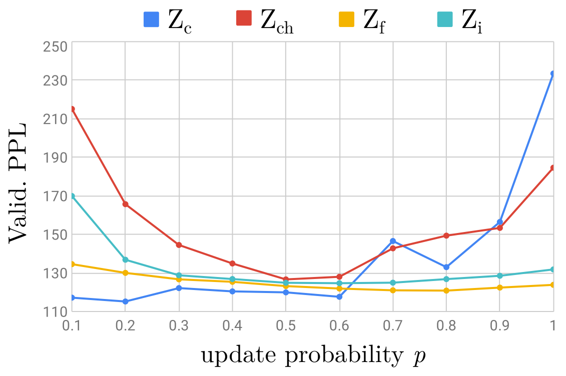
The authors of the original zoneout study claim that shared random masks, i.e. reusing the same mask for both $\mathbf{c}$ and $\mathbf{h}$ seems to work worse than initializing two separate random masks (Krueger et al., 2017). Considering that this is the main difference between zoneout and our Z_{ch} regularizer, our results seem to validate this claim. However, we can extend this original finding as we have shown that shared masks only seem to be an issue when skipping *both* cell and hidden state. Unfortunately, this setup is the only one considered in the original study and, therefore, still known as the default method of applying zoneout regularization. Therefore, we see considerable evidence that our proposed variants might improve the original zoneout method, even though further research on different tasks is necessary to verify our results.

7.4.3 Effects of Update Probabilities on Convergence

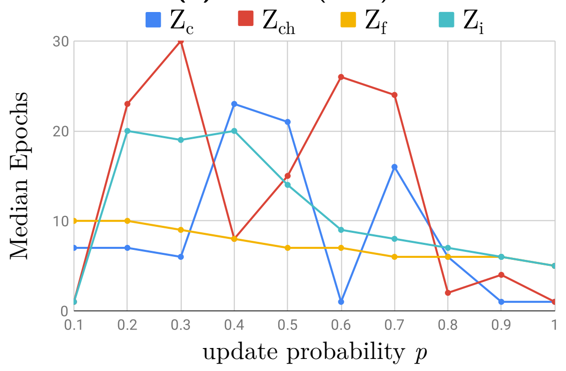
We will now take a more in-depth look at how the update probabilities influence convergence and, more specifically, at which update rates the models start to deteriorate in performance. Setting the update rates through p allows us to push each model variant beyond its natural update rate equilibrium and to observe whether it can still successfully solve the given task under the enforced update sparsity. This also allows us to directly measure the individual trade-offs between both metrics



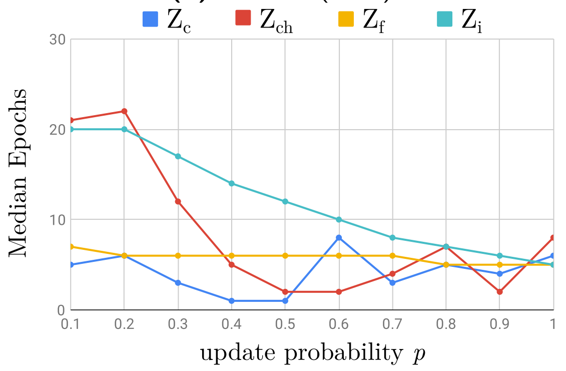
(a) ZO+R (PPL)



(b) ZO+M (PPL)



(c) ZO+R (Epochs)



(d) ZO+M (Epochs)

Figure 7.9: Comparison between zoneout regularization (ZO-R) and the zoneout models (ZO-M), which are a simplification of SBA in which stochastic sampling replaces the surprisal mask. **Top row:** Validation Perplexity. **Bottom row:** Convergence time in median epochs.

and to get an understanding as to which gates and states provide the best skipping foundation for an ideal trade-off.

Figure 7.9 shows the effect of the update probability parameter p (which is in practice also the expected update rate u) on ZO+R and ZO+M, respectively. As expected, the smaller the update probability p, the longer the networks take to converge. The trend for zoneout regularization is the same, though it fluctuates more strongly. ZO+M with Z_i and Z_{ch} is significantly more stable during training than ZO+R regularizers as they show unstable convergence. For the other two models Z_c and Z_f, ZO+R and ZO+M both show similar patterns for varying p, although the absolute perplexity is lower for ZO+M.

As in previous experiments, Z_c works best around $p = 0.2$ and degrades in performance the more p (or u) is increased. For the other models, the ideal update probability seems to be somewhere between 0.5 and 0.7. In addition, our previously hypothesized lower boundary for the update rates is clearly visible for two of the ZO+M models, namely Z_{ch} and Z_i. Lowering p beyond this threshold causes a significant trade-off in performance. For the S_c model this relationship is also reversed: this lower bound becomes an upper bound as the network functions best with small update rates.

To summarize, by sampling the update decisions we can further simplify SBA and successfully control the update rates of our networks directly. This is practically useful for the purposes of analysis and having a more fine-grained control in cases where we specifically know how much state updates we want to perform or save. Our zoneout variants work even better when the same network is used for testing instead of a regular LSTM. If we pick the best overall model, the S_c, both the SBA+R and the ZO+M setup give the same performance. This further adds evidence to our hypothesis that each network has its own update-accuracy equilibrium u_{UAE} to which the models tend to converge, while any update rate deviations away from this local minimum always lead to a decrease in accuracy. While we were able to observe some exceptions to this rule, the ideal update rate seems to generally have a large degree of independence from the skipping method that is used.

7.5 Chapter Summary

In this chapter, we have provided a deeper analysis of SBA. We have particularly focused on understanding its relationship to regularization, as well as measuring and improving the trade-off between model performance and update rate.

The different SBA model variants behave differently as they skip based on different memory components of the LSTM. In repeatedly investigating these differences, we were able to gain a better understanding over which memory components have to be controlled with SBA to lead to either better-performing models, a reduction in state updates, or a combination of both. In some instances, we have found that SBA can eliminate significant amounts of redundancy. For example, our LSTM+S_c model can achieve close-to baseline accuracies while eliminating/skipping 86% of its state updates. Depending on which memory component is manipulated, the model always converges to an update rate with low variance. Overcoming this strong attractor for the update rates has shown to be difficult.

Extending the original formulation of SBA, we have investigated multiple possibilities on how the update rates can be minimized alongside the task-related objective. While regularization penalties have shown limited effects, we were able to show that SBA itself has some regularizing properties. In particular, the SBA+R+S_c representations improved significantly at test time when transferred to a regular LSTM, highlighting that the constrained nature of SBA can sometimes lead to issues in memory access and therefore to generalization difficulties in very sparsely activated networks.

Finally, we have introduced and investigated multiple novel variants of zoneout that we have redefined as models with distinct state update rules, moving zoneout closer to SBA. This has allowed further improvements to both our baseline and the original zoneout algorithm. The experiments have also confirmed that SBA is able to reach a slightly better update range than the result of a hyperparameter grid search with zoneout. Furthermore, we have shown that zoneout can be used as an analytical tool to practically construct networks with specific update rates, allowing us to analyze the correlation between update rate and model performance for a given skip mechanism.

The biggest limitation of SBA is that its implementation does not reduce the number of floating point operations. SBA can be considered a form of online pruning. During training, SBA observes candidate activations to decide if they should be taken into account for an update or skipped. Since the candidate activations have to be calculated before they can be evaluated, SBA is not able to prevent these computations and only operates in hindsight. However, repeated processing and hindsight updating has been successfully demonstrated in similar models, e.g. for multi-turn reading (see Subsection 3.5.5). Furthermore, this form of post-processing to focus on a subset of the features is also a typical property of recurrent neural attention mechanisms. To an extent, our approach can therefore be seen as a form of hard attention that is applied at the hidden layer instead of the output layer (Bahdanau et al., 2015; Vinyals et al., 2015). In contrast to attention, which introduces a considerable amount of additional parameters and calculations, SBA nonetheless only adds to the computational demand in the form of a softmax, log-transform, and pooling operation, all of which are bounded below linear complexity. In the following chapters, we will introduce a hierarchical attention framework (Chapter 8) before investigating the interplay between SBA and attention (Chapter 9) and providing a direct comparison between surprisal and attention as a skipping constraint (Chapter 10).

Part IV

Hierarchical Attention and Surprisal

Chapter 8

Hierarchical Attention Networks

To improve skipping in recurrent networks by eliminating redundancies, we require the ability to identify and focus important inputs and representations. Attention models have been proposed in the role of such mechanisms, simulating a more explicit model of a network's focus during processing. The use of attention has been shown to improve both model performance and interpretability. Similarly, hierarchical representations have been proposed to improve learning of structured sequences such as with natural language.

For this reason, this chapter is concerned with introducing hierarchical attention networks. Previously, our application focus has been on language modeling which mostly requires local context and more long-term relationships in some cases. In the following, we turn to a task in which only few parts of the input are highly relevant to the output and most other inputs can be discarded. More specifically, we look at cloze-style question answering which is considered a popular reading comprehension task.

8.1 Motivation

Attention-based recurrent architectures have recently been successfully applied to tasks such as language modeling (Tran et al., 2016a), speech recognition (Bahdanau et al., 2016), or machine translation (Luong et al., 2015). Recently, hierarchical attention has been proposed in the form of the Hierarchical Attention Network (HAN) and successfully been used for document classification (Yang et al., 2016). The main idea of the HAN is to hierarchically apply attention mechanisms at the word- and sentence-level. This way, the network representations are able to mirror

Sections 8.1-8.3 of this chapter have been published as part of this thesis and are based on Alpay et al. (2019), illustrations © 2019 IEEE, reprinted with permission.

the hierarchical structure of documents. A different task that might highly benefit from structured representations is question answering. Assuming direct questions, e.g. about entities, relevant information is typically very sparsely distributed over documents used for question answering. That is, few sentences in the provided document relate to the asked question, and of these only a few contained words need to be queried to infer the correct answer.

Therefore, we investigate how the HAN can be used for this task and we focus on cloze-style question answering (QA). In such recent datasets for QA, queries are generated by removing a single word (e.g. a named entity) from a sentence which then has to be inferred from the remaining text. This allows one to assume that the answer to the question is a single word contained in the text document, and that the dataset provides candidate words for the answer.

Although attention-based models are quite popular for QA tasks (Kadlec et al., 2016; Hermann et al., 2015), few hierarchical approaches with attention mechanisms have been proposed to date. Previous successful approaches for QA mostly use shallow recurrent encoders (Chen et al., 2016) that only operate on word-level or work with task-specific memory representations, such as the end-to-end memory network (Sukhbaatar et al., 2015). A hierarchical variant of memory networks has also been introduced (Chandar et al., 2016) although its hierarchical memory is fixed and not learned. A hierarchical model for long text documents has been proposed by Choi et al. (2016). Different from our bottom-up approach of building hierarchical representations from word- to document-level, they present a more top-down approach where the model first selects relevant sentences before generating answers with reinforcement learning. Hierarchical models have also successfully been used for visual question answering (Lu et al., 2016) although the structure of images is very different from that of text documents.

Other work has investigated novel attention mechanisms that facilitate training specifically for QA tasks. For example, the Attention Sum Reader Network (ASRN; Kadlec et al. (2016)) assumes that the answer is contained in the document, allowing it to directly point to the answer rather than inferring it from a blended representation of words as is usual in similar models.

In this chapter, we investigate both approaches and propose a novel hierarchical model. We adapt the original HAN to QA tasks and extend it twofold to infer an answer from hierarchical representations: first, based on a hierarchical document vector representation (HAN-doc-vec) and second, based on pointer sum attention (HAN-ptr). We evaluate our models on the Children's Book Test (CBT; Hill et al. (2016)) and compare them to a non-hierarchical variant to ultimately address the following research questions:

1. Does hierarchical attention facilitate the task of cloze-style question answering?
2. Does pointer sum attention work better than a blended document vector representation for integrating hierarchical representations?

8.2 Hierarchical Attention Network for Question Answering

In this section, we describe the suggested models. The hierarchical structure is realized by primarily adapting the Hierarchical Attention Network (HAN; Yang et al. (2016)) to the QA task.

The main idea is as follows: at word level, our system encodes a sentence representation from word representations. At the sentence level, it learns to represent the document based on these sentence representations. This final representation is then used to find the most likely word in the document which answers the query. To realize this final step, we propose two variants. The first builds up a document vector (HAN-doc-vec) from hierarchical attention that is used to infer the correct answer from a blended representation. The second uses pointer sum attention at the final layer (HAN-ptr) in order to compute answer probabilities directly from the word and sentence attention values. Additionally, we evaluate our architectures by realizing a non-hierarchical attention model (RAN) as an appropriate baseline.

8.2.1 The Word Level

Our architecture assumes three different inputs, namely the text document D, the question q (sometimes also called query), and a list of candidate words c, which contains the correct answer. We start at the word level by embedding the document D^{inp} and the question q^{inp} into word vectors with the help of the pre-trained embedding matrices $\mathbf{E}_A$ and $\mathbf{E}_q$ (we use GloVe embeddings (Pennington et al., 2014)). This gives us the embedded representations $\mathbf{D}^{emb} = \mathbf{E}_D \cdot D^{inp}$ and $\mathbf{q}^{emb} = \mathbf{E}_q \cdot q^{inp}$.

Word Encoding

Similar to the HAN, we use Gated Recurrent Units (GRUs; Chung et al. (2014)) as recurrent sequence encoders throughout our architecture. Our main motivation of choosing the GRU over the Long Short-Term Memory (LSTM; Greff et al. (2017)) is that it offers similar performance for less parameters. The GRU state $\mathbf{h}_t$ is generally computed with the help of an update gate $\mathbf{z}_t$ and a reset gate $\mathbf{r}_t$ as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz} \cdot \mathbf{x}_t + \mathbf{W}_{hz} \cdot \mathbf{h}_{t-1}), \tag{8.1}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr} \cdot \mathbf{x}_t + \mathbf{W}_{hr} \cdot \mathbf{h}_{t-1}), \tag{8.2}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{W}_{rh} \cdot (\mathbf{r}_t \otimes \mathbf{h}_{t-1})), \tag{8.3}$$

$$\mathbf{h}_t = (1 - z_t) \otimes \mathbf{h}_{t-1} + \mathbf{z}_t \otimes \tilde{\mathbf{h}}, \tag{8.4}$$

where $\tilde{\mathbf{h}}_t$ are the candidate activations, and the matrices $\mathbf{W}$ are trainable connections from the input or previous layer state to the current layer state. The σ denotes the sigmoid activation function, and $\otimes$ signifies element-wise multiplication.

We employ a bidirectional GRU (biGRU; Bahdanau et al. (2016)), where we use one GRU layer to read the sentences forward, from left to right, and another to read them backward in reversed order. This gives us the forward hidden state $\overrightarrow{\mathbf{h}}_t$ and the backward hidden state $\overleftarrow{\mathbf{h}}_t$. Both layer encodings are then combined by concatenation, i.e. $\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t]$, to receive the output of the bidirectional layer. Therefore, by looking at both directions simultaneously, we allow the network to capture more context around each word.

At the word level, we now encode our word embeddings for the document and the question with two separate biGRU layers, calling the resulting encodings $\mathbf{D}^{word}$ and $\mathbf{q}$:

$$\mathbf{D}^{word} = \text{biGRU}(\mathbf{D}_t^{emb}) = [\overrightarrow{\mathbf{h}}_t(\mathbf{D}_t^{emb}), \overleftarrow{\mathbf{h}}_t(\mathbf{D}_t^{emb})] \tag{8.5}$$

$$\mathbf{q} = \text{biGRU}(\mathbf{q}_t^{emb}) = [\overrightarrow{\mathbf{h}}_t(\mathbf{q}_t^{emb}), \overleftarrow{\mathbf{h}}_t(\mathbf{q}_t^{emb})] \tag{8.6}$$

The document is processed sentence by sentence at this word level, i.e. the currently processed sentence serves as the available context. Consequently, $\mathbf{D}_t^{word}$ encodes contextual information for each word within its sentence.

Word Attention

In the next step, we use an attention layer so that each word can have a different contribution to the sentence representation:

$$\beta_{i,j}^{word} = \mathbf{q}^T \cdot \mathbf{D}_{i,j}^{word} \tag{8.7}$$

$$\alpha_{i,j}^{word} = \text{softmax}(\beta_{i,j}^{word}) \tag{8.8}$$

Since the dot product acts as a similarity measure between each word and the question, $\beta_{i,j}^{word}$ acts as an indicator on the relevance of the question to the j-th word of the i-th sentence in the document encoding $\mathbf{D}_{i,j}^{word}$. Applying the softmax gives us a probability distribution which we can use as the word-level attention values $\alpha_{i,j}^{word}$.

In the final step, we use these word attention values as weights for the associated word encodings to blend them into an aggregated sentence representation:

$$\mathbf{D}_i^{blend} = \sum_{k=0}^{n-1} \alpha_{i,k}^{word} \cdot \mathbf{D}_{i,k}^{word} \tag{8.9}$$

This leads to words with larger attention values being represented more prominently in the sentence representations.

8.2.2 The Sentence Level

After the end of the word level, we were able to estimate the importance of each *word* w.r.t. the question. The purpose of the sentence level is to allow the same for the *sentences*. This also allows us to discard previously high word attention values in sentences with low sentence attention which are meant to contain little information to answer the question.

Therefore, the previous steps are now mostly repeated on sentence level: we feed the previously attained sentence representations into a biGRU encoder and gain a document-level representation of weighted sentences by computing attention values on the sentence representations.

Sentence Encoding

At this point, we can enrich the vector of each sentence with context information from the whole document by feeding the sentence representations $\mathbf{D}^{blend}$ to a sentence-level biGRU:

$$\mathbf{D}^{sent} = \text{biGRU}(\mathbf{D}_t^{blend}) = [\overrightarrow{\mathbf{h}}_t(\mathbf{D}_t^{blend}), \overleftarrow{\mathbf{h}}_t(\mathbf{D}_t^{blend})] \tag{8.10}$$

Since this layer processes a sequence of sentence representations, this may lead to each sentence vector $\mathbf{D}_i^{sent}$ containing information on both $\mathbf{D}_i^{blend}$ as well as the neighboring sentences $\mathbf{D}_{i-1}^{blend}$ and $\mathbf{D}_{i+1}^{blend}$.

Sentence Attention

The sentence attention is computed analogous to the word attention values:

$$\beta_i^{sent} = \mathbf{q}^T \cdot \mathbf{D}_i^{sent} \tag{8.11}$$

$$\alpha_i^{sent} = \text{softmax}(\beta_i^{sent}) \tag{8.12}$$

Note that we use the same question vector q from the word level. Using the sentence encodings, β_i^{sent} now indicates the similarity between this question vector and the contextual sentence vector $\mathbf{D}_i^{sent}$.

8.2.3 Computing the Final Output

After calculating the hierarchical attention-based representations in the previous sections, we now introduce two different methods to infer an output answer. The first method uses a document vector representation (HAN-doc-vec), the second pointer sum attention (HAN-ptr) to find the correct answer word in the document.

Document Vector

Calculating a document vector representation after the sentence attention layer is the most natural approach as it mirrors the production of the sentence vectors $\mathbf{D}^{blend}$ on word-level. We therefore blend all weighted sentence vectors together to get $\mathbf{D}^{doc}$:

$$\mathbf{D}^{doc} = \sum_{k=0}^{m-1} \alpha_k^{sent} \cdot \mathbf{D}_k^{sent} \tag{8.13}$$

This vector sums up the entire document based on the sentence attention values. By doing this, we have propagated the attention information of the most-likely words from word to document level. In order to compare $\mathbf{D}^{doc}$ with the candidate answers $\mathbf{c}$, we apply a linear transformation to reduce the vector dimension to that of the embedded candidate vector $\mathbf{c}^{emb}$ and get the final document vector $\mathbf{D}^{final}$:

$$\mathbf{D}^{final} = \mathbf{W}_f \cdot \mathbf{D}^{doc} + \mathbf{b}_f, \tag{8.14}$$

where $\mathbf{W}_f$ is a weight matrix and $\mathbf{b}_f$ are the biases.

To compute the final output using $\mathbf{D}^{final}$ and $\mathbf{c}$, we assume that if a candidate is the answer to the question, it should be represented more prominently in $\mathbf{D}^{doc}$ than the other candidates. While it is possible to simply measure the similarity between $\mathbf{D}^{final}$ and $\mathbf{c}^{emb}$, this would lead to a negative influence of candidate vectors that do not have pre-trained embeddings. Therefore, we still initialize the candidate embeddings with pre-trained word embeddings $\mathbf{E}_c$ but allow the embedding layer to update during training (different from $\mathbf{q}^{emb}$ and $\mathbf{D}^{emb}$ who have static embeddings). Training the candidate embeddings end-to-end now allows the lower recurrent layers to enrich them with semantic context.

With the candidate embeddings, we can finally calculate the final similarity and apply a softmax to see the likelihood for each candidate c_i that it is the correct answer:

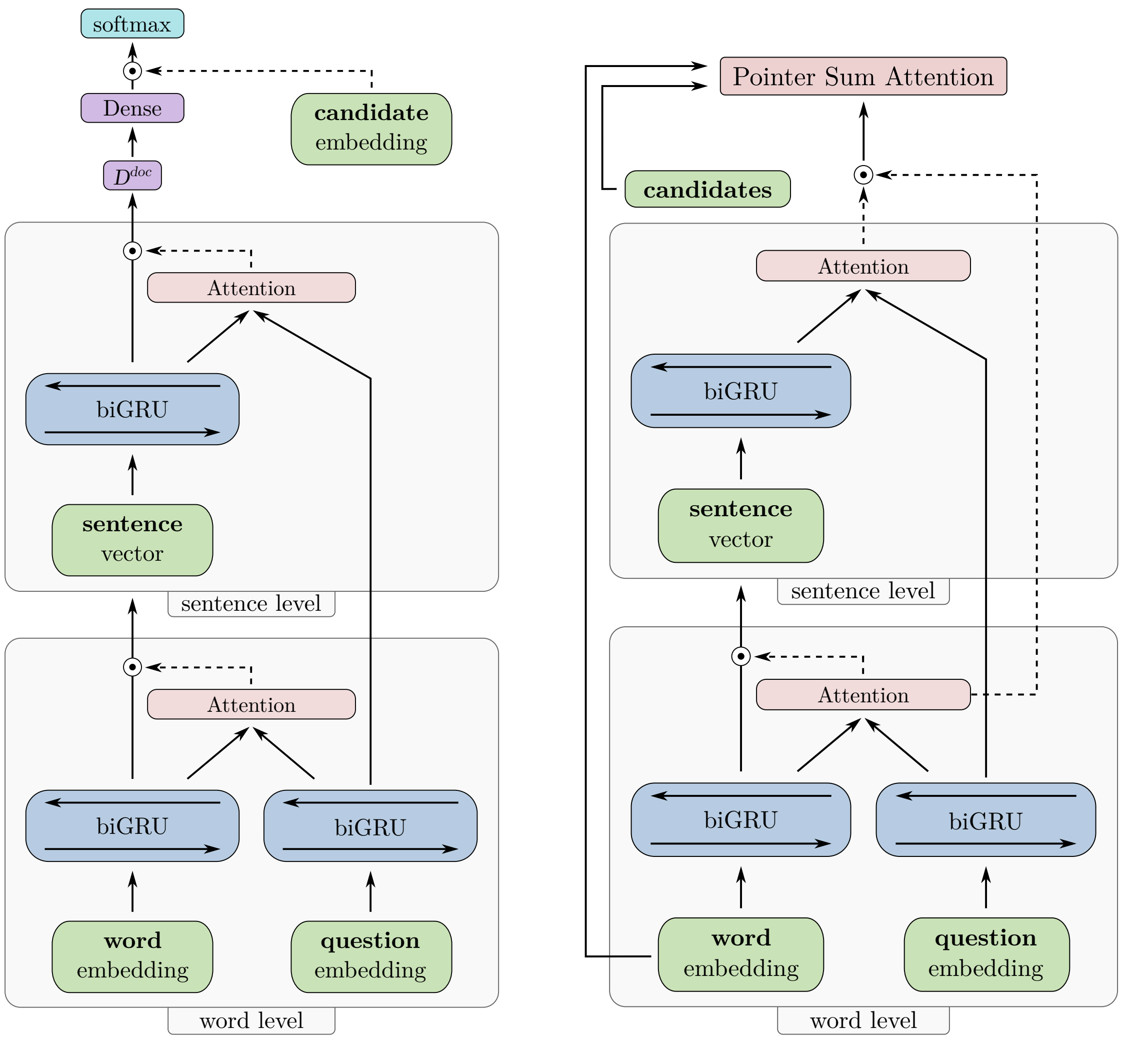
$$\gamma = \mathbf{c}^T \cdot \mathbf{D}^{final}, \tag{8.15}$$

$$P^{doc}(c_i | D^{inp}, q^{inp}, c^{inp}) = \text{softmax}(\gamma) \tag{8.16}$$

The complete HAN-doc-vec model architecture is illustrated in Figure 8.1a.

Pointer Sum Attention

Different than the HAN-doc-vec, we also investigate how we can use the previously computed attention values in order to directly infer the answer without computing a document vector after the sentence level. The main idea is inspired by the work of Kadlec et al. (2016) on their Attention Sum Reader and our resulting model HAN-ptr integrates it in a hierarchical manner.



(a) HAN-doc-vec **(b)** HAN-ptr

Figure 8.1: Comparison between the HAN-doc-vec and the HAN-ptr.

As a first step, we combine the word and sentence attention values to get a probability distribution over all words of the document:

$$\alpha_{i,j}^{doc} = \alpha_i^{sent} \cdot \alpha_{i,j}^{word} \tag{8.17}$$

As a result, the overall attention $\alpha_{i,j}^{doc}$ for each word is linked with the attention it received in its sentence i as well as the attention the sentence i itself received over the document. This allows us to apply *pointer sum attention* as defined by Kadlec et al. (2016) to our hierarchical model.

For the candidates c^{inp}, we now sum the attention values in $\alpha_{i,j}^{doc}$ for each occurrence of c_i^{inp} in the document D^{inp}. The candidate with the most cumulative attention is chosen as the right answer. In other words, let $Y(i)$ be the set of indices defined by:

$$Y(i) = \{(k,l)|D_{k,l}^{inp} = c_i^{inp}, 0 \leq k < m, 0 \leq l < n\}, \tag{8.18}$$

Then, $Y(i)$ includes all positions in the document where the candidate c_i^{inp} occurs, and the overall attention of c^{inp} is then:

$$P^{ptr}(c_i|D^{inp}, q^{inp}, c^{inp}) = \sum_{(k,l)\in Y(i)} \alpha_{k,l}^{doc} \tag{8.19}$$

The idea behind using pointer sum attention in such a fashion is to try to exploit the hierarchical attention structure to produce more accurate attention values than only calculating the attention on word level (as the ASRN does). While the doc-vec approach continuously transforms its representations hierarchically, this method directly uses the captured information from each hierarchical layer.

One limitation of this approach is that the pointer sum attention only looks at candidate words in the text document, i.e. any context information about other words is only coded indirectly through their attention values. The complete model using the pointer sum attention is illustrated in Figure 8.1b.

8.2.4 Baseline Model

In order to investigate the impact of the hierarchical structure in our models, we construct a baseline by removing the sentence level from the HAN-doc-vec (illustrated in Figure 8.2). The resulting model is very similar to that of Chen et al. (2016) except that we still build a document vector representation from the word level, apply the linear transformation from Equation 8.14, and avoid using their bilinear attention form as we found this method to not increase the overall performance significantly enough to warrant a larger parameter size.

For the sake of clarity, we call this baseline the Recurrent Attention Network (RAN). One difference to the other two models is that the network takes the entire document sequence as input since the baseline only operates on word level. So different than the other two models, the recurrent layers of the RAN can keep context information between sentences.

8.2.5 Summary

To summarize, we investigate the following models with increasing complexity:

1. RAN: Recurrent Attention Network (non-hierarchical baseline)
2. HAN-doc-vec: Hierarchical Attention Network with document vector representation
3. HAN-ptr: Hierarchical Attention Network with pointer sum attention

Both HAN variants build up hierarchical representations on word and sentence level. The ptr model infers the answer directly from the attention values while the

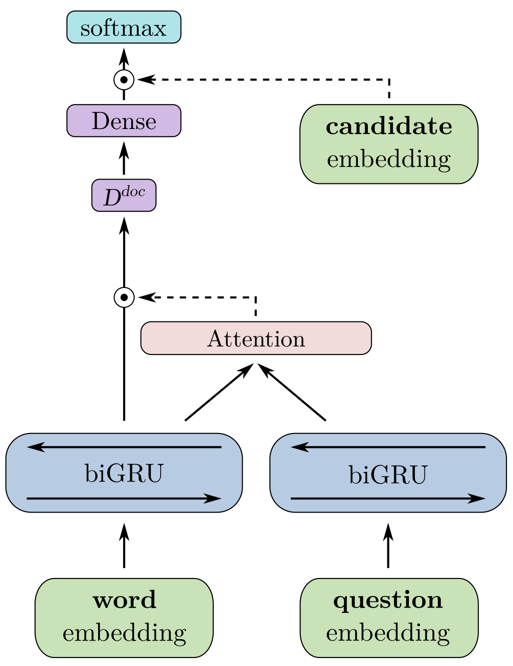


Figure 8.2: The baseline model RAN.

doc-vec model builds an additional representation on document level to summarize the sentences and infer the answer with trainable candidate embeddings.

8.3 Experiments

8.3.1 Dataset

We use the Children's Book Test (CBT; Hill et al. (2016)) to evaluate our models. The dataset is constructed from 108 children's books which typically have a very clear narrative structure. It is also one of the currently more popular cloze-style datasets for automated question answering, wherein the questions and answers are generated by removing a single word from a sentence which then has to be inferred from the remaining text. In the CBT dataset, each document consists of 21 consecutive sentences. One word removed from the 21st sentence serves as the answer while the remaining sentence forms the query. The reader then has to read the previous 20 sentences (the context) and pick one from ten candidate answers which best fit the placeholder in the query.

For this study, we investigate two different types of words that may be treated as placeholders: Named Entities (NE) and Common Nouns (CN). These two categories have been constructed for the CBT dataset using the Stanford Core NLP toolkit (Manning et al., 2014) and are therefore effectively treated as two different datasets. We choose these categories as it has been shown that LSTM language models have difficulty achieving human performance with these two-word types in the CBT dataset (Hill et al., 2016). It is important to note that we only use raw text as input, i.e. there are no entity annotations that are shown to the models. CBT-CN consists of 120,769 and CBT-NE of 108,719 training documents. Both datasets have 2,000 validation and 2,500 test documents. The total number of unique words is roughly 53,000 for both datasets.

8.3.2 Training Details

We use pre-trained GloVe word embeddings (Pennington et al., 2014) and investigate the three available embedding dimensions of $|\mathbf{E}| \in \{100, 200, 300\}$. For unknown words, we draw random numbers from a uniform distribution in the same value range as the embeddings. For the number of hidden units we choose $|\overrightarrow{\mathbf{h}}| = |\overleftarrow{\mathbf{h}}| \in \{256, 384, 512\}$ for both backward and forward layers, i.e. each bidirectional hidden layer has twice as many units in total. We observed smaller sizes leading to significantly worse accuracies, whereas larger layers led to challenging computational requirements. The models are run on a single GPU (NVIDIA GTX 1080 Ti) using a batch size of 64, which leads to an average training time of 20 minutes (HAN-ptr) and 25 minutes (HAN-doc-vec) per epoch on the larger dataset. The RAN model takes 2 hours per epoch since it receives the entire document as a sequence. All networks train for a maximum of 7 epochs and we record the epoch with the best validation accuracy. The recorded median number of epochs for convergence is 4 for the RAN baseline and HAN-doc-vec, and 3 for the HAN-ptr. All our models are trained using the Adam optimizer (Kingma and Ba, 2015) and the categorical cross entropy loss. A learning rate of 0.001 is picked for both datasets as the result of preliminary experiments. Each hyperparameter configuration is trained 10 times with different seeds to account for randomization.

8.3.3 Evaluation

Hyperparameters

Figure 8.3 summarizes our hyperparameter evaluation on both datasets. Overall, smaller GloVe word embedding dimensions seem to work better than larger ones. However, the variance is smaller for the HAN-ptr model, indicating that the other two models could face difficulties training candidate embeddings with high dimensionality (the HAN-ptr uses a simple lookup instead of candidate embeddings).

The number of hidden units per layer seems to positively correlate with the validation accuracy, although the overall variance caused by changing hidden units is much smaller than that observed from different embeddings. This shows an overall robustness of all models w.r.t. their parameter size.

	Common Noun		Named Entity	
Model	valid	test	valid	test
Humans (query) [1]	-	64.4	-	52.0
Humans (query + context) [1]	-	81.6	-	81.6
Maximum Frequency (context) [1]	27.3	28.1	29.9	33.5
LSTMs (query) [1]	61.3	54.1	50.0	40.8
LSTMs (query + context) [1]	62.6	56.0	51.2	41.8
Memory Networks [1]	64.2	63.0	70.4	66.6
Attention Sum Reader Network [2]	68.8	63.4	73.8	68.6
RAN (non-hierarchical baseline)	60.8	57.4	64.4	58.7
HAN-doc-vec	60.4	56.4	62.9	57.7
HAN-ptr	69.1	67.7	75.5	69.9

Table 8.1: Overview and comparison of our results on the CBT dataset for the Common Noun (CN) and Named Entity (NE) categories. Results marked with [1] are from Hill et al. (2016), with [2] are from Kadlec et al. (2016).

As in previous studies, our networks also have an easier time querying for named entities than common nouns (see Table 8.1). The HAN-doc-vec model is even slightly behind our baseline for named entities, while it gives more consistent results for common nouns where it slightly improves on the baseline.

Test Results

After hyperparameter optimization, we select the best models based on the validation accuracy and evaluate them on the test set. Table 8.1 shows our best results for the Common Noun (CN) and Named Entity (NE) categories of the CBT dataset. As can be seen, the HAN-doc-vec falls short to significantly improve on our baseline RAN (especially in the NE category). This shows that the hierarchical processing itself does not guarantee better performance. On the other hand, the HAN-ptr gives significantly better results than both the baseline and the doc-vec version. By comparing this result to the single model Attention Sum Reader Network (ASRN), we can also conclude that the attention sum pointer is not the only cause for the performance gain of the HAN-ptr. For common nouns, in particular, we get a better test accuracy with the HAN-ptr. Since the ASRN also uses the same attention mechanism, we are left to assume that our hierarchical layout is the reason that we can partially improve on the original ASRN. The differences between the two word type categories additionally suggest that a hierarchical pointer sum approach is more beneficial when learning common nouns.

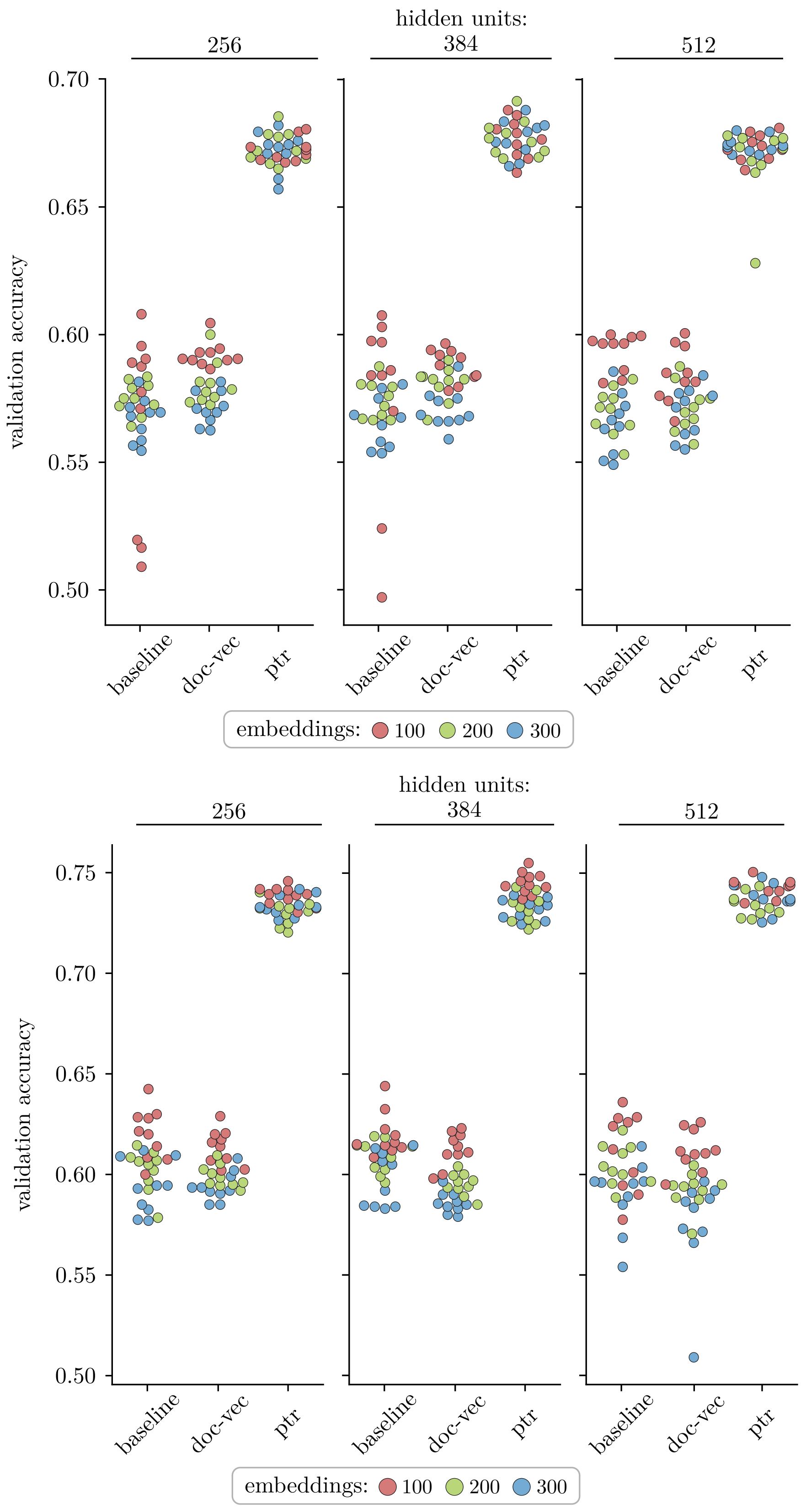


Figure 8.3: Swarm plot with validation accuracies. Evaluating the three models, RAN (baseline), HAN-doc-vec, and HAN-ptr with different embedding and hidden dimensions. Top: CBT-CN. Below: CBT-NE.

However, even though our result improves on related approaches, it does not achieve state-of-the-art results on the CBT dataset. Using a gated attention mechanism, Yang et al. (2017) have achieved higher test accuracies of 72.0% (CBT-CN) and 74.9% (CBT-NE) although their model works on character- and word-level and is therefore assumed to be significantly more computationally expensive.

8.3.4 Pointer Sum Attention on Pre-trained Models

The pointer sum attention mechanism leads to an attention distribution over the entire document, leading to more sparsely distributed attention than in the HAN-doc-vec. This means that the pointer sum attention should identify less crucial words. To gain a deeper understanding of the differences between HAN-doc-vec and HAN-ptr, we apply pointer sum attention at test time, exchanging the output layers of the best pre-trained HAN-doc-vec and RAN model with the ptr layer. This results in the baseline accuracy dropping from 55.28% to 41.08% and the HAN-ptr dropping slightly from 56.24 to 53.64%.

Since both networks have trained with a different objective, it is somewhat predictable that this procedure reduces the overall performance. It is however interesting to note, that the HAN-doc-vec achieves almost the same performance when removing everything after the final sentence attention layer and inferring the answer directly with the two attention layers. Our interpretation of this is that during training, the HAN-doc-vec only transforms the information already contained in the attention values into the document vector. It does not seem to gain much additional information by combining the contextual sentence vectors and attention values to form A^{doc}. It seems that the main challenge for the model is to come to the point of calculating the attention values, which seems to be most critical for the final performance.

We therefore hypothesize that HAN-doc-vec puts enough information into the attention distribution (without losing a significant amount of accuracy) to be able to evaluate it and to transform the attention into an answer while neglecting the associated contextual sentence vectors. HAN-ptr is trained to do exactly this and should, therefore, have an advantage, which may result in the better performance that we have observed.

8.3.5 Attention Distributions

We investigate our models for differences in their attention distribution. For this purpose, we sort words by their total cumulative attention score (in descending order) over the test set. The "most attended" words are shown in Table 8.2. Within each category, the models mostly share similar words. However, the differences between the two word categories are quite large. In particular, when querying

	Common Noun			Named Entity		
#	baseline	doc-vec	ptr	baseline	doc-vec	ptr
1.	business	business	am	the	the	.
2.	obey	pay	obey	and	,	,
3.	am	obey	business	in	and	the
4.	so	in	so	a	to	”
5.	in	meddle	soon	of	he	!

Table 8.2
Words with the highest **cumulative** word attention score in test set.

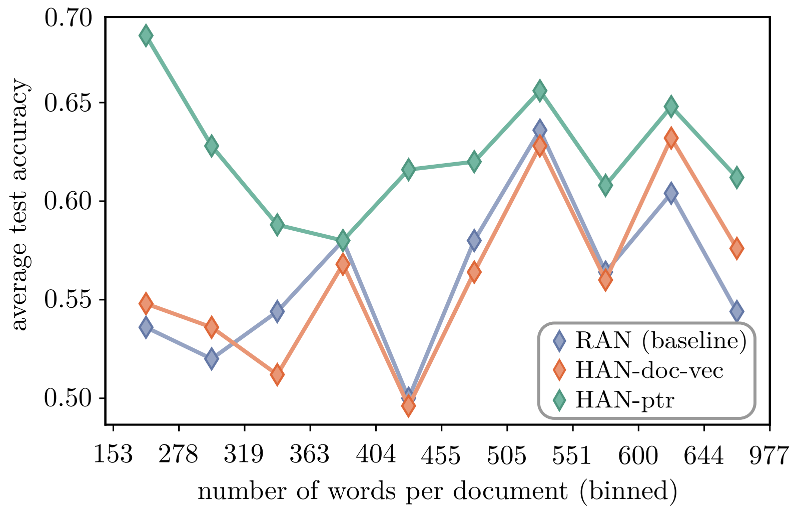
	Common Noun			Named Entity		
#	baseline	doc-vec	ptr	baseline	doc-vec	ptr
1.	crouched	slanderous	blarney	separate	july	somalo
2.	wails	murderous	jewelled	humph	ein	ralston
3.	agreement	seaweed	shirtsleeves	sixteen	ralston	bandmaster
4.	orders	disappear	somalo	gulped	saxby	shelmardine
5.	scraps	resolute	rescue	modern	dancer	emperor

Table 8.3
Words with the highest **relative** word attention score in the test set.

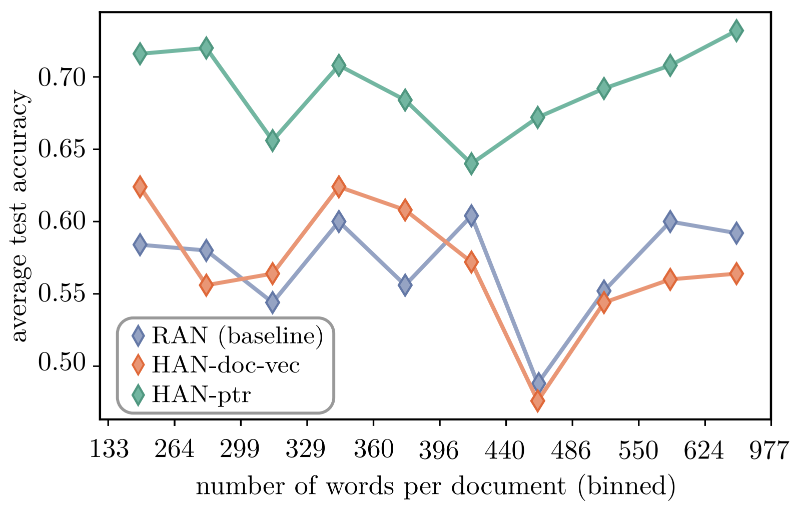
named entities, a lot of attention is put on stop words. The HAN-ptr seems to even generate large attention towards punctuation.

In order to filter out very frequent words, we normalize the total attention score by word frequency which gives us a different list (see Table 8.3). It is visible that these words, with the highest attention score per occurrence, are very context-specific nouns and adjectives, with many of them being uncommon. By comparing this to the absolute attention scores, we can infer that attention is often distributed to meaningless words. However, in relative terms, the attention scores are highest when rare words are encountered that share a semantic context with the answer. Since we can further see that all three models produce quite different word lists when sorted by relative attention scores, this also indicates that all three approaches reach quite different internal models for semantic context.

Since the attention distributions naturally depend on the length of the documents, we have further examined how the testing accuracy depends on the length of the context document as shown in Figure 8.4. We found that the RAN and the HAN-doc-vec show particular strengths for longer documents on common nouns and shorter documents on named entities, while the HAN-ptr provides the best results independent of the document lengths. From detailed data analysis we have indications that these differences are due to the focus of RAN and HAN-doc-



(a) Common Noun (CN)



(b) Named Entity (NE)

Figure 8.4: Average test accuracy for different bins of document lengths.

vec on specific words and sentences in the documents, which were particularly informative for longer documents on common nouns and for shorter documents on named entities, while HAN-ptr is a bit more independent of the document structure.

8.3.6 Visualizing the Attention

In addition to our previous analysis on the attention distributions, we inspected the attention values in detail. Two such examples can be seen in Figure 8.5. In the shown document, both the HAN-doc-vec and the HAN-ptr manage to correctly identify the right answer. The HAN-ptr, however, focuses on punctuation and stop words noticeably often (see also Subsection 8.3.5). Although this observation seems

00 **0.007** no sooner had ferko entered the **0.12** palace than all eyes were turned on the handsome youth , and the king 's **0.34** daughter herself was lost in admiration , for she (...)

01 **0.004** his brothers noticed this , and envy and jealousy were added to their fear **0.18** , so much so that they determined once more to **0.47** destroy **0.16** him .

02 **0.009** they went to **0.12** the king and told him that ferko was a **0.11** wicked magician , who had come to the palace with the intention of carrying off the **0.10** princess . **0.16**

03 **0.006** then the king had ferko brought before him , and said , ` you are accused of being a magician who wishes to rob me of my daughter , and i condemn you to **0.34** death ; but if can fulfil three tasks which i shall set you to do your life shall be spared , on condition (...)

04 **0.003** and turning to **0.50** the **0.12** two wicked brothers he said , ` suggest something for him to do ; no matter how difficult , he must succeed in it or die . '

05 **0.059** they did not think long , but replied , ` let him build your majesty in **0.10** one day a more **0.34** beautiful **0.17** palace **0.11** than this , and if he fails in the attempt let him be (...)

06 **0.001** the **0.23** king **0.13** was pleased with this proposal **0.23** , and commanded ferko to **0.11** set to work on the following day . **0.17**

07 **0.003** the **0.15** two brothers were delighted , for they thought they had now got rid of **0.11** ferko **0.11** for **0.17** ever . **0.33**

08 **0.009** the **0.15** poor youth **0.27** himself was heart-broken , and cursed the hour he had crossed the boundary of the king 's **0.26** domain .

09 **0.045** as he was wandering disconsolately about the meadows round the **0.14** palace **0.13** , wondering how he could escape **0.16** being put to **0.26** death , a little bee flew past , (...)

10 **0.015** can i be of any help to **0.87** you ?

11 **0.022** i am the **0.26** bee whose wing you healed , and would like to **0.36** show my gratitude in some way . '

12 **0.030** ferko recognised the **0.50** queen **0.17** bee , and said , ` alas !

13 **0.003** how could you help **0.88** me ?

14 **0.023** for i have been set to **0.16** do a task which no one in the whole world could do , let him be ever such **0.17** a **0.23** genius !

15 **0.651** to-morrow i must build **0.44** a **0.22** palace more beautiful than the king 's **0.15** , and it must be finished before evening . '

16 **0.004** ` is **0.25** that **0.25** all ? ' **0.33**

17 **0.071** answered the bee , ` then you may comfort yourself ; for before the sun goes down to-morrow night a **0.23** palace shall be built unlike any that **0.32** king has dwelt in before .

18 **0.004** just stay here till i come again and tell you that it **0.10** is **0.14** finished . **0.32** ' **0.16**

19 **0.031** having said this she flew merrily away , and ferko , reassured by her words , lay down on the **0.19** grass **0.30** and **0.16** slept peacefully till the next morning .

Question:
early on the following day the whole town was on its feet , and everyone wondered how and where the stranger would build the wonderful **xxxxx** .

Candidates:
admiration **0.000** domain **0.000** help **0.000** jealousy **0.000** matter **0.000** one **0.000** palace **0.999** proposal **0.000** something **0.000** task **0.001**

Answer: palace

(a) HAN-doc-vec.

00 **0.006** no sooner had ferko entered the palace **0.25** than all eyes were turned on the handsome youth **0.20** , and the king 's daughter herself was lost in admiration , for she (...)

01 **0.001** his brothers noticed this , **0.19** and envy and jealousy were added to their fear , so much so that **0.11** they determined once more to destroy him . **0.40**

02 **0.079** they went to the king and told him that ferko was a wicked magician , who had come to the palace **0.69** with the intention of carrying off the princess . **0.12**

03 **0.013** then the king had ferko brought before him , and said , ` you are accused of being a magician who wishes to rob me of my daughter , and i condemn you to death ; but if you fulfil three tasks which i shall set you to do your life shall be spared , on condition you country **0.29** ; but if you can not perform what i demand you shall be hung on the (...)

04 **0.001** and turning to the two wicked brothers he said , ` suggest something for **0.17** him to do ; no matter how difficult , **0.21** he must succeed in it or die . **0.18** '

05 **0.210** they did not think long , but replied , ` let him build your majesty in one day a more beautiful palace **0.96** than this , and if he fails in the attempt let him be hung . '

06 **0.005** the king was **0.15** pleased with this proposal , and commanded ferko to set to work on the following day . **0.33**

07 **0.002** the two brothers were delighted , **0.12** for they thought they had **0.11** now got rid of ferko for ever . **0.46**

08 **0.005** the poor youth **0.30** himself was heart-broken , and cursed the hour he had crossed the boundary of the king 's domain . **0.22**

09 **0.039** as he was wandering disconsolately about the meadows round the palace **0.59** , (...)

10 **0.001** can i **0.13** be **0.19** of any help to you ? **0.33**

11 **0.006** i am the bee whose **0.12** wing you healed , and would like to show my gratitude in some way **0.14** . **0.36** '

12 **0.001** ferko recognised the queen bee **0.40** , **0.25** and said , ` alas !

13 **0.000** how could **0.19** you help me ? **0.59**

14 **0.012** for i have been set to do a task which **0.19** no one in the whole world **0.22** could do , let him be ever such a genius ! **0.13**

15 **0.347** to-morrow i must build a palace **0.88** more beautiful than the king 's , and it must be finished before evening . '

16 **0.008** is **0.17** that **0.60** all ? **0.14** '

17 **0.230** answered the bee , ` then you may comfort yourself ; for before the sun goes down to-morrow night a palace **0.70** shall be built unlike any that king has dwelt in before .

18 **0.034** just stay here till i come again and tell you that **0.31** it is **0.22** finished **0.11** . **0.20** '

19 **0.002** having said this she flew merrily away , and ferko , reassured by her words , **0.21** lay down on the grass and slept peacefully till the next morning . **0.26**

Question:
early on the following day the whole town was on its feet , and everyone wondered how and where the stranger would build the wonderful **xxxxx** .

Candidates:
admiration **0.000** domain **0.001** help **0.000** jealousy **0.000** matter **0.000** one **0.000** palace **0.998** proposal **0.000** something **0.000** task **0.001**

Answer: palace

(b) HAN-ptr.

Figure 8.5: Illustration of an example document and the calculated word (blue) and sentence (red) attention values. All attention values above 0.1 have been highlighted in the text document. For the HAN-ptr, the overall attention of each word is its word attention multiplied with the sentence attention. The right answer is "palace".

counter-intuitive, this model, interestingly, is able to create large attention on the relevant sentences in which the answer word is contained. Other candidates such as “task” (e.g. sentence 14 in Figure 8.5, right side) receive little to no attention even though they are prominently displayed in the text. This indicates that the generation of mostly “useless” attention values, does not seem to have a major influence on the accuracy of the networks. For the HAN-ptr, this can be explained by its final step in which all attention values are discarded, except for those of the candidates. This could mean that the HAN-ptr explicitly focuses on stop words and punctuation in the absence of embeddings which are semantically related to the candidates, as this would never decrease the loss.

For the HAN-doc-vec and RAN approach, frequent words that do not share any context with the candidates, are also effectively discarded due to the final dot product which measures the similarity between document representation and the question. In this model, “noisy” attention on frequent words can also be observed (e.g. “the” and “to” in Figure 8.5, left side). However, the trainable candidate embeddings seem to play an important role as the network is very certain of the right answer “palace”, even though the word itself receives little attention in the entire document. Note that the attention for the 15th sentence is very high. This means that, even though on word-level “palace” receives little attention, the network was able to localize the correct answer with high certainty from its surrounding context and encode this into its sentence representation.

However, the examples also show that the sentence attention mostly serves to identify the location of the answer on sentence level. Different from the word level with its embedded context vectors, the sentence level encodings do not seem to relate or complement each other. This may be one of the reasons why the hierarchical models improve the performance only to a limited extent.

8.4 Chapter Summary

So far, we have evaluated how Hierarchical Attention Networks can be used for cloze-style question answering tasks. We have presented two approaches for building hierarchical representations based on attention mechanisms. Our results indicate that a hierarchical structure itself does not always necessarily lead to better training but that it depends on how the information from the hierarchical representations is aggregated at the output layer. The pointer sum method has shown itself to harmonize well with the hierarchical architecture in our results. Notably, our results show that recurrent hierarchical models for complex tasks can be designed with a comparably low parameter count and relatively fast training times. Since publication, our results for the HAN-ptr have independently been reproduced and verified by Alsahli and Mirzal (2020).

When examining the sentence attention, it can be seen that the attention values themselves mostly correlate with the likelihood of containing the correct answer. This seems to help training, especially for the ptr model which weights word attention with this likelihood.

One limitation of our approach is that we assume the output answer to be contained in the document. On the other hand, the pointer sum attention is specifically designed for this setup (Kadlec et al., 2016). Additionally, we assume that the network can be provided with candidates, i.e. in the introduced form, our models are able to be trained with cloze-style datasets but would require external knowledge or the generation of candidates for more open questions. Future work should therefore concentrate on evaluating our approach on different types of QA datasets. For example, one could continuously sample from the output layer in order to generate answer sentences from the pointer sum attention model.

Finally, our results show that the top-down pointer attention mechanism is able to recognize important words and sentences in the context of the question answering tasks. In the context of conditional computation, this capability is vital for a model in order to decide when to update. Therefore, we will build on the presented research by utilizing attention mechanisms for skipping in the following chapters.

Chapter 9

Surprisal-Based Activation in Hierarchical Attention Networks

So far, we have successfully demonstrated the usefulness of both hierarchical encoding and cumulative attention mechanisms. In this chapter, we integrate Surprisal-based Activation (SBA) with both concepts to achieve skipping in the recurrent layers of the Hierarchical Attention Network (HAN). For this purpose, we propose integrating SBA to the HAN-ptr model, which performed best in our previous study. Our primary motivation is to utilize attention as a salience model to improve the focus of the skipping model itself. Based on this, we aim to get a better understanding on the interplay and compatibility between attention and skipping mechanisms. We particularly focus on the capabilities to reduce the update rates as much as possible, aiming towards minimizing any trade-offs with model accuracy. Based on this goal, we extend SBA to skip on a per-layer- instead of per-module-basis. We start this chapter by elaborating our motivation to link surprisal and attention (Section 9.1) before introducing (Section 9.2) and evaluating (Section 9.3) our Surprisal-based GRU (SGRU) model, which we subsequently extend to be able to skip on a per-layer-basis in Section 9.4. We evaluate the proposed models on the CBT dataset (Section 9.5) and provide an in-depth analysis in Section 9.6.

9.1 Modeling Salience with Surprisal and Attention

For a stimulus to be perceived as important, it requires features that make it stand out from its local neighborhood. These stimuli are called *salient* and the concept of salience (or saliency) has been used to explain what drives human attention (Parr and Friston, 2017). In vision, attention can be driven by either bottom-up salience (how much low-level features stand out in their local neighborhood) or by top-down salience (which aspects are relevant for the task or goal) (Melloni

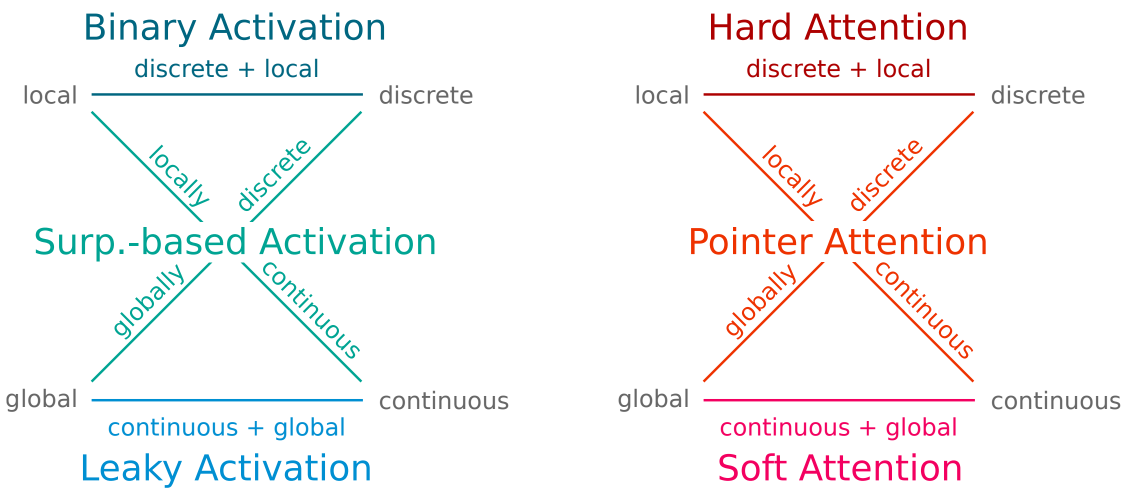


Figure 9.1: Linking activation (left) and attention schemes (right) by their underlying representations (discrete vs. continuous) and the locality of the mechanism's effect (local vs. global).

et al., 2012). A similar distinction is made for linguistic salience as linguistic units (characters, words, and sentences) are similarly hierarchical. Zarcone et al. (2016) argue that, while linguistic salience is often related to surprisal and predictability, it is unclear whether the two can be equated. They argue that surprisal does not account for the interaction between high- and low-levels of processing, and that an emphasis on attention and relevance can improve this aspect.

In the context of our hierarchical attention network, attention is modeled hierarchically in a bottom-up fashion, where the word representations are gradually processed by attentive mechanism into sentence representations. In the final layer, a document representation is built which, in conjunction with the candidate embedding, allows a task-driven top-down interaction with the lower-level attention scores of the word-and sentence layers. In the following sections, we will explore how surprisal-based activation can additional filter out important words in a bottom-up manner, i.e. on the recurrent representations and *before* attention is applied.

Aside from the aforementioned discussions on a potential biological link between surprisal and attention, we argue that there are also clear similarities for how both concepts can be modeled in the context of artificial neural networks. Figure 9.1 illustrates these analogies between representation learning and attention mechanisms in RNNs. While the traditional attention mechanism (also called *soft attention*) is modeled as a global distribution with continuous values, *hard attention* discretely samples features locally, making it non-differentiable (Mnih et al., 2014). We can find a similar situation in recurrent memory models, where leaky and gated activations are continuous and exert a temporally global influence. On the opposing end, we can define purely symbolic networks, e.g. by using binary activations or Gumbel-Softmax representations, which offer discrete-valued local representations.

As we have shown previously, pointer attention can be interpreted as a hybrid between soft and hard attention: while the attention scores are continuous and encoded in a local distribution of the respective word- or sentence layer, they are aggregated globally over all layers and the final "pointing" mechanism is discrete. This is analog to SBA, where only some continuous activations are locally selected for processing, while others are skipped over. The actually processed words are then discretely chosen from a global top-down perspective. As demonstrated in Chapter 8, pointer attention is capable of utilizing pre-attentive representations very sparsely. This is mainly achieved by softmax normalization, which we also use for Surprisal-based Activation to achieve high update sparsity. In traditional (soft) attention, the softmax output, which cannot output a vector item that is exactly 0, is not post-processed and can therefore not produce sparse distributions (Martins and Astudillo, 2016). Therefore, we hypothesize that both concepts could be utilized jointly to increase activation sparsity effectively, maintaining the baseline accuracy.

It is important to emphasize that pointer attention is capable of achieving high accuracy by focusing on very few tokens in the document due to the nature of the underlying question answering task (as illustrated in Figure 8.5). In fact, since the answer to the given question is contained in the presented document, it is theoretically sufficient to only pick the answer and to discard all other words[1]. Unfortunately, the continuous distribution makes this difficult in practice as irrelevant words simply get a low attention score instead of being fully discarded in a discrete manner by setting important words to $\alpha_w = 1$ and others to $\alpha_w = 0$. If tokens with low attention scores appear very frequently, their aggregated score can start getting larger than expected[2]. This can lead to the undesirable side effect of amplifying redundancy and leading to either decreasing performance or worse interpretability of the learned attention maps. Consequently, we hypothesize that the ability of SBA to fully discard certain words can boost the ability of the pointer attention to ignore redundancies more strongly.

9.2 The SGRU Model

Since the previously proposed HAN models for question answering are utilizing Gated Recurrent Units (GRUs) instead of LSTMs, we will next show how the SBA methodology can be adapted to GRUs. After introducing the Surprisal-based GRU (SGRU), we will discuss how this model can be deployed in the existing HAN-ptr architecture. Following this, we will give a theoretical analysis as to how

[1]Note that this also considers the presence of more complex semantic relationships between the question and answer, which can be encoded in the hidden layers even though it is ultimately discarded by the following attention mechanism.

[2]This is best visible in Table 8.2 where some punctuation marks have the highest cumulative attention score.

much we can reduce update rates in the given setup without starting to lose critical information.

9.2.1 Model Definition

The main difference between the GRU and the LSTM is in the number of gates and the resulting organization. Compared to the LSTM, the GRU has an *update gate* $\mathbf{z}_t$ which combines the purposes of the LSTM's forget and input gate into a single gate. As a result, there is no cell state and the entire hidden state content is exposed to the outside of the cell. As these changes lead to a smaller number of trainable parameters, updating a GRU memory cell is computationally more efficient than an LSTM memory cell. To recap, we summarize the GRU state update as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz} \cdot \mathbf{x}_t + \mathbf{W}_{hz} \cdot \mathbf{h}_{t-1}), \tag{9.1}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr} \cdot \mathbf{x}_t + \mathbf{W}_{hr} \cdot \mathbf{h}_{t-1}), \tag{9.2}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{W}_{rh} \cdot (\mathbf{r}_t \otimes \mathbf{h}_{t-1})), \tag{9.3}$$

$$\hat{\mathbf{h}}_t = (1 - z_t) \otimes \mathbf{h}_{t-1} + \mathbf{z}_t \otimes \tilde{\mathbf{h}}_t, \tag{9.4}$$

where $\tilde{\mathbf{h}}_t$ are the GRU candidate activations, and the matrices $\mathbf{W}$ are trainable connections from the input or previous layer state to the current layer state. While $\hat{\mathbf{h}}_t = \mathbf{h}_t$ holds for a regular GRU update, we can now extend $\hat{\mathbf{h}}_t$ to a possible second state which stands for the process of skipping, i.e. $\hat{\mathbf{h}}_t = \mathbf{h}_{t-1}$.

To achieve this goal, we adapt the basic SBA skipping model from Section 6.3 for the GRU, using SBA to decide between the final candidate activation $\hat{\mathbf{h}}_t$ and $\mathbf{h}_{t-1}$ in hindsight by max-pooling module activations in the candidate $\tilde{\mathbf{h}}_t$ and computing the resulting surprisal values s_t based on the pooling result $\mathbf{p}_t$:

$$\mathbf{p}_t = \max(\hat{\mathbf{h}}_t) = [\max(\hat{\mathbf{m}}_t^{(1)}), \ldots, \max(\hat{\mathbf{m}}_t^{(M)})], \tag{9.5}$$

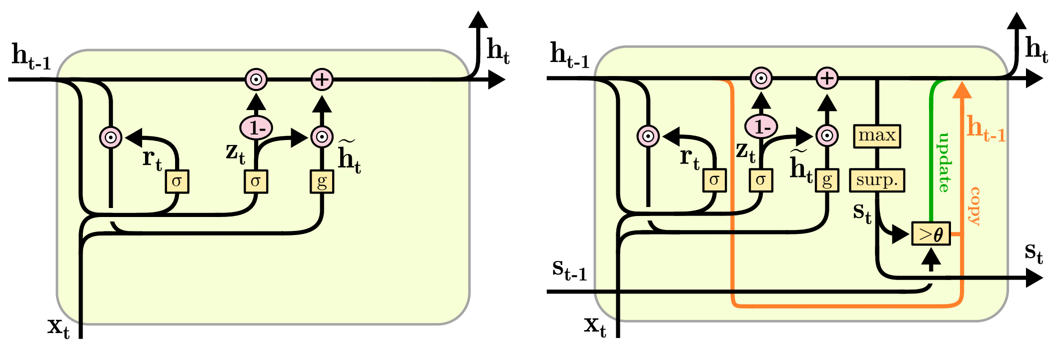
$$s_t = \log\left(\frac{1}{\sigma(\mathbf{p}_t)}\right) = \log\left(\frac{\sum_{i=1}^{M} \exp(\mathbf{p}_t^{(i)})}{\exp(\mathbf{p}_t)}\right) \tag{9.6}$$

Afterwards, the skipping decision is made separately for each of the M modules, leading to the final module activations $\mathbf{m}_t^{(i)}$:

$$\mathbf{m}_t^{(i)} = \mathcal{S}(\hat{\mathbf{m}}_t^{(i)}) = \begin{cases} \hat{\mathbf{m}}_t^{(i)} & \text{if } s_t > s_{t-1} + \theta \\ \mathbf{m}_{t-1}^{(i)} & \text{otherwise,} \end{cases} \tag{9.7}$$

where $\mathcal{S}(\cdot)$ denotes the resulting SBA transformation which keeps module activations with high surprisal and discards others by copying the previous activations and keeping the gradient constant between timesteps. The final layer activation $\mathbf{h}_t$ is then composed of these final modules:

$$\mathbf{h}_t = [\mathbf{m}_t^{(1)}, \ldots, \mathbf{m}_t^{(M)}] \tag{9.8}$$



(a) GRU cell

(b) SGRU cell

Figure 9.2: Comparison between a GRU and an SGRU cell. The conditional operator "$> \theta$" decides between a regular update (green) and a copy operation of the previous state h_{t-1} (orange).

We call the resulting cell model Surprisal-based GRU (SGRU) and show an overview and comparison to the original GRU cell in Figure 9.2.

As a result of our previous analyses from chapters 6 and 7, and to reduce the number of hyperparameters, we deliberately stick to max-pooling and avoid applying any activation decay. Consistent with our previous HAN models, we apply SBA on the forward layer $\overrightarrow{\mathbf{h}}_t$ and the backward layer $\overleftarrow{\mathbf{h}}_t$ separately before concatenating the result:

$$\text{biGRU}(\mathbf{x}_t) = [\mathcal{S}(\overrightarrow{\mathbf{h}}_t(\mathbf{x}_t)), \mathcal{S}(\overleftarrow{\mathbf{h}}_t(\mathbf{x}_t))] \tag{9.9}$$

Since both layers process their input $\mathbf{x}_t$, this can lead to different decisions on which inputs will be ignored in both layers, depending on whether the sequence is processed normally or in reverse.

9.2.2 HAN-SGRU-ptr

We will now discuss the use of SGRU layers in the HAN-ptr. With the exception of the recurrent layers, we will retain the original structure of the HAN-ptr, including its bidirectional layers and hierarchical attention. We call the resulting model *HAN-SGRU-ptr*.

If we take a look at the original HAN-ptr model, we can identify three recurrent layers: two on world-level to process either the words in the document or the query (the question), and one on sentence-level to process the sentence vector which is built bottom-up from the previous two representations. This makes it possible to deploy the SGRU in any of these three encoders.

From the structure of the question answering task, we can immediately see that the query vector contains a higher information density than both the word or sentence vectors of the document. This is because the query is a single sentence which is guaranteed to provide relevant word-level context to the answer, while the document itself contains multiple sentences and words irrelevant to the correct answer. Since the dot product with the query embedding is the central mechanism to correlate similarities and filter important representations in both the word- and sentence-level attention, removing irrelevant words from the query vector should improve the overall task performance.

On the other hand, the query encoder receives significantly less inputs than the word and sentence layers, meaning that a single SGRU query layer can not reduce the total update count of the entire network by much. Since more words are processed in the word layer, more computations can be affected, especially considering that the document holds more words with low information gain. Finally, using the SGRU on sentences will allow the network to potentially skip entire sentences. While this can lead to a significantly large portion of the inputs to be ignored, it can also have negative side effects by accidentally filtering out words that are being used by the attention pointer.

It is important to note that these considerations describe *potential* behaviors in the presented SGRU model since it does not necessarily need to skip on a per-layer basis. Instead, the hidden layer is partitioned into modules which skip independently of each other, meaning that words and sentences can be ignored by some hidden units, while others can choose to process them normally. In the next section, we will present and discuss the results for our HAN-SGRU-ptr model on the CBT dataset, showing that this seems to actually be the case. Built on these observations, we will then suggest modifications to the SGRU in Section 9.4 in order to force the model to operate layer-wise by synchronizing the modules and fully skipping entire words and sentences in the model.

9.2.3 A Lower Bound for Update Rates in the CBT Dataset

The lowest possible update rate that a model can achieve depends on the total number of processed tokens and thus depends on the underlying task and dataset. To be able to judge a model's update rates properly, it is therefore necessary to know the lower bound for update rates on the given dataset. We will now provide these bounds to provide an indication as to how close our model's update rates are to the theoretical limit.

The word and query encoding tensors (for their full definition see Section 8.2) have dimensions $\mathbf{D}^{word} \in \mathbb{R}^{S \times W \times H}$ and $\mathbf{q} \in \mathbb{R}^{W \times H}$, respectively, whereas the sentence encoding tensor has dimension $\mathbf{D}^{sent} \in \mathbb{R}^{S \times H}$. Based on the number of sentences per document $S = 20$, the maximum number of words per sentence $W = 100$, and a hidden layer size of $H = 256$, we get the following minimum update rates per document for the three encoders:

$$u_{min}^{sent} = \frac{1}{S} = \frac{1}{20} = 0.05 = 5\% \tag{9.10}$$

$$u_{min}^{query} = \frac{1}{W} = \frac{1}{100} = 0.01 = 1\% \tag{9.11}$$

$$u_{min}^{word} = \frac{1}{S \cdot W} = \frac{1}{20 \cdot 100} = 0.0005 = 0.05\% \tag{9.12}$$

It is important to understand that these lower bounds constitute no hard technical limit but rather one where we would expect an increased likelihood for the recurrent layer to start encountering critical information loss *on average*. While a degradation in one of the encoders *can* lead to random guessing in the overall model, it doesn't necessarily have to. In fact, we can identify a number of edge cases where one of the encoders can fall slightly below this threshold without significant degrading the model. First of all, considering that the accuracy of human subjects on the CBT dataset is 81.6%, we can conclude that almost every fifth document is potentially not solvable and random guessing on these documents would therefore have no impact. Moreover, since the pointing mechanism accumulates the layer-wise attention scores, any positional biases in the dataset can be exploited by the model (e.g., if one of the 20 sentences contains the answer more often than the others), potentially even more easily than in the baseline, since the pre-attentive input stream consists of repeated tokens that are indistinguishable from each other due to constant skipping. Moreover, as the HAN model has two input streams and a hierarchical layout, the non-skipping layers can still contain the necessary information to solve the task. The Pointer Sum Attention Layer in particular, operates directly on the (unmodified) word embeddings instead of the recurrent word encoding. Finally, the update rates depicted above correspond to a single layer. Since we are using bidirectional layers (and reporting the average of the forward and backward layer), there is the possibility of *one* of the two layers remaining inactive ($u = 0$) while the other focuses on the task ($u \geq u_{min}^{enc}$). This can halve the respective update rates from the bidirectional average to $u_{min}^{sent} = 0.025 = 2.5\%$, $u_{min}^{query} = 0.005 = 0.5\%$, and $u_{min}^{word} = 0.00025 = 0.025\%$.

Since update rates have a practical lower bound that is not 0, the measured update rate $u \in [0, 100]$ does not percentually explain how much lower it can go towards its minimum u_{min}, which is either u_{min}^{word} or u_{sent}^{word}. To account for this, we can convert the measured update rate u into the correct scale using the following linear equation:

$$\begin{aligned}
\tilde{u} - 100 &= \frac{100 - 0}{100 - u_{min}} \cdot (u - 100) \\
\tilde{u} &= \frac{100 \cdot (u - 100)}{100 - u_{min}} + 100 \\
&= \frac{100 \cdot (u_{min} - u)}{u_{min} - 100}
\end{aligned} \tag{9.13}$$

The resulting *effective update rate* $\tilde{u} \in [-\frac{100 \cdot u_{min}}{100 - u_{min}}, 100]$ then expresses for $\tilde{u} > 0$ how much further it can be reduced percentually towards u_{min} as opposed to 0. Likewise, it holds that $\tilde{u} = 0$ for $u = u_{min}$ and a negative value $\tilde{u} < 0$ signals a drop below u_{min}. While $u \approx \tilde{u}$ for larger update rates, the differences become more noticeable with $\tilde{u}$ approaching 0. We use $\tilde{u}$ to additionally report the models' proximity to u_{min} on the respective hierarchical level and the potential for additional gains[3].

9.3 SGRU Results

To ensure comparability with our previous experiments, we reuse all previous hyperparameters for training the HAN-ptr model, though we deliberately limit the search space by using the GloVe word embeddings with dimension $|E| = 100$ and hidden layer sizes $|\mathbf{h}| = 256$, as we have only found limited accuracy gains for larger dimensions in the HAN-ptr (compare Subsection 8.3.3). For the SGRU, we explore an exponentially increasing series for the number of modules $M \in \{2, 4, 8, 16, 32, 64, 128, 256\}$ as well as thresholds $\theta \in \{0.01, 0.1\}$ which we have determined to work well after preliminary experiments. As discussed in the previous section, we additionally compare the different effects of using the SGRU in the query, word, or sentence layers. We additionally test a configuration using SBA in both the word and sentence layer simultaneously, purposefully not including the query layer in order to more clearly see how sentence vectors with skipped words are treated by the SGRU.

The overall results of the experiment for both datasets are illustrated in Figure 9.3 for CBT-CN and Figure 9.4 for CBT-NE. We present the results using the model validation accuracy and the average validation update rate for the respective bidirectional SBA layers[4]. On average, using an SGRU for the query layer yields a slightly better accuracy than for the word layer, whereas skipping sentences decreases accuracy further, especially when the sentence vectors are hierarchically composed of skipped words. This reinforces our hypothesis that the pointer mechanism is more reliant on a small number of words and that the importance of each sentence vector is merely defined by how many of these important words it holds. Since the skipping layers of the hierarchical model only communicate indirectly via the intermediate attention layer, there is a possibility for disagreement between the SGRU and the attention layer as to what is considered important. As the skipping process effectively copies past states, this is particularly likely in cases where irrelevant words are amplified by the copy mechanism of the SGRU, giving the attention layer little to focus on if important words are overwritten by copying.

[3]For the remaining parts of this thesis, we consistently report accuracy and update rate u on the scale $[0, 1]$ in figures to allow for an easier visual comparison between both metrics. To allow for the best numerical comparison, we consistently report both accuracy and u in percent ($[0, 100]$) for all tables, since the update rates *between models* sometimes differ by factors of up to 3000.

[4]Averaging the update rate of the forward and backward layers.

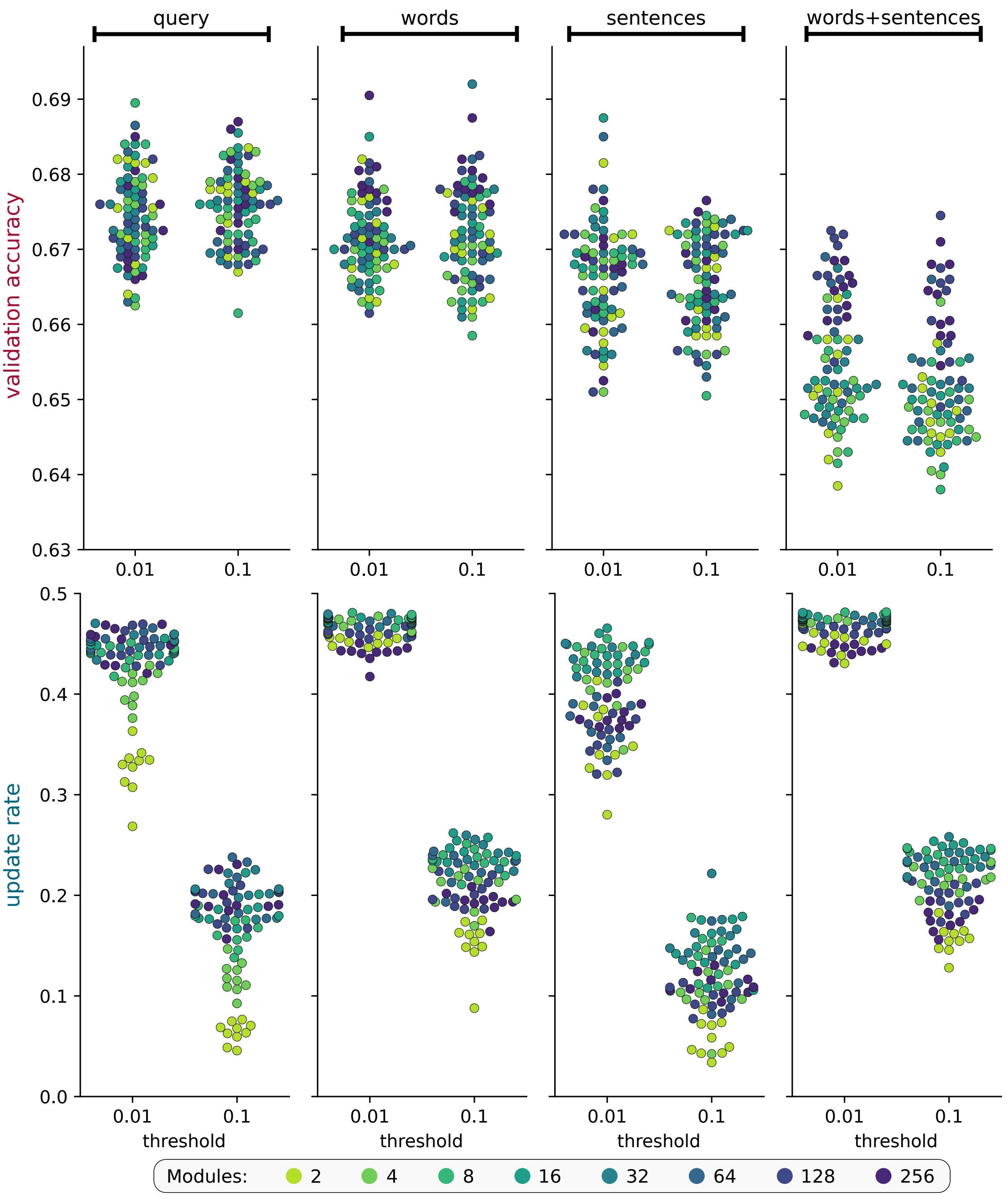


Figure 9.3: HAN-SGRU-ptr **(CBT-CN)**: Swarm plots showing accuracy and update rates on respective y-axes, threshold θ on every x-axis, and SBA layer locations in the columns (colored by number of modules).

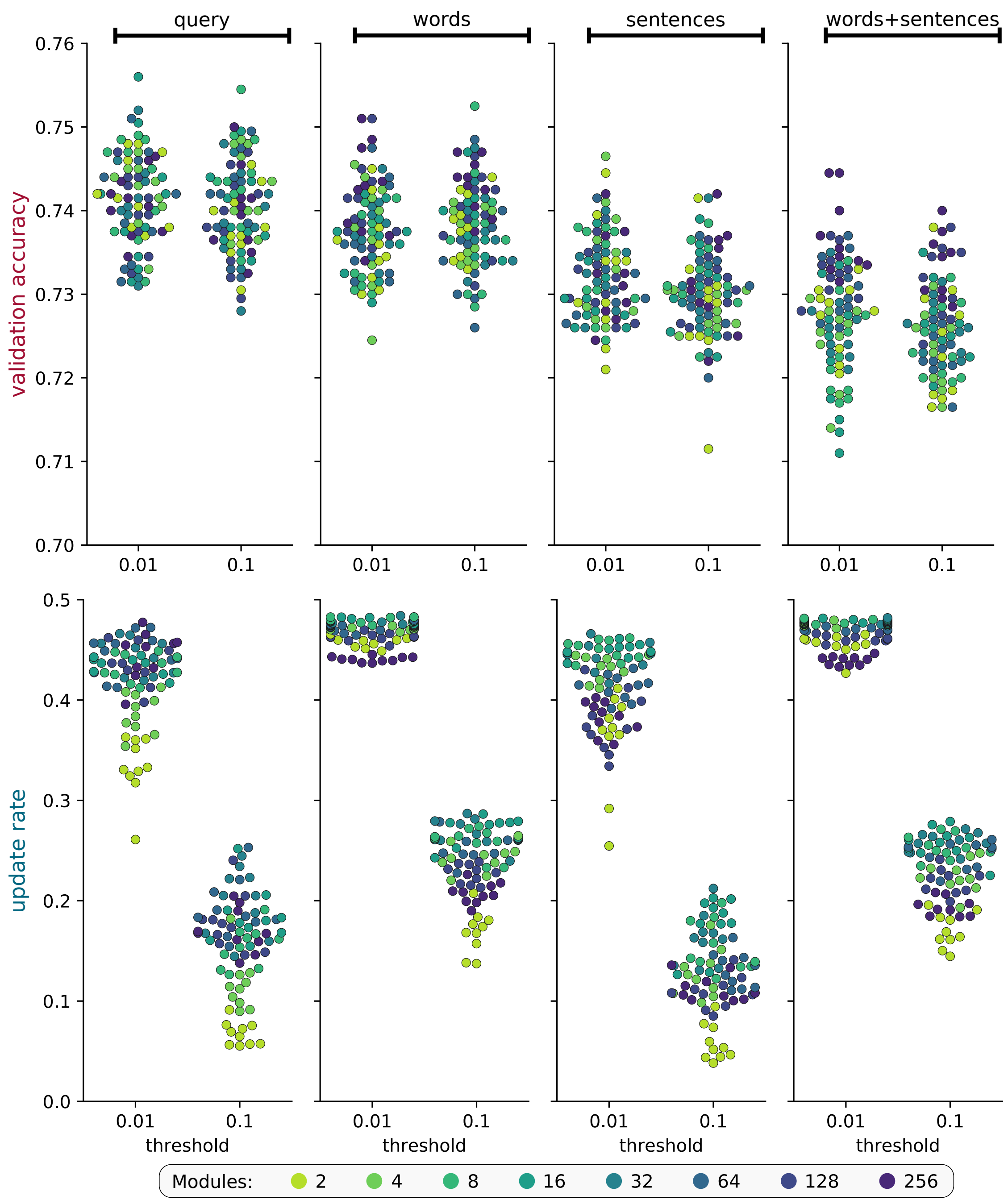


Figure 9.4: HAN-SGRU-ptr **(CBT-NE)**: Swarm plots showing accuracy and update rates on respective y-axes, threshold θ on every x-axis, and SBA layer locations in the columns (colored by number of modules).

This effect seems to be stronger when the answer to the query is a common noun than if it is a named entity.

We can only identify minor differences for the update rates when the SGRU is deployed in different locations, though there is a clear tendency to reach higher update rates when we allow skipping in the word layer. We can also see clear differences for the number of modules that we use in the SGRU layer. For the query layer, the number of modules correlates positively with the update rate, while $M = 2$ and $M = 256$ lead to the lowest update rates in the other configurations. In other words, increasing the diversity through modularization, improves the accuracy while simultaneously raising the update rates slightly.

The overall update rate is generally in the range of $[0.3, 0.5]$ for a threshold $\theta = 0.01$ and in $[0.05, 0.3]$ for $\theta = 0.1$ which demonstrates how the update rate can be adjusted by tuning the threshold. Interestingly, larger differences in the update rate do however only slightly impact the accuracies. While different thresholds lead to measurable differences, the overall correlation between both metrics seems to be low. We hypothesize that this is primarily a reflection of the fact that it is not necessary to read most of the document to achieve a high accuracy. If the network is able to only read the most important tokens which directly lead to the location of the answer, most of the document can be ignored. In fact, we believe this to be the reason why the hierarchical filtering mechanism of the pointer attention mechanism is so successful.

Since we know that the given task has a very low information density (see Subsection 9.2.3), our findings suggest that the SGRU is not achieving its full potential. Instead, increasing the diversity through modularization leads to slightly higher update rates which in turn are likely to cause the better accuracies that we have observed. Similarly, the update rates are capped in certain ranges which seem to be more determined by the parameterization of the threshold θ than the task itself. As previously discussed, this is also somewhat expected as the SGRU only decides to skip for each module, instead of the entire hidden layer, thereby completely discarding words. These observations indicate that a) modularization is actually increasing the update rates instead of reducing them and b) the model itself can be even more strict in its decisions. Therefore, we will next show how we can extend the existing SBA model to skip for the entire layer, and investigate whether this approach can aid us in pushing the update rates down even further.

9.4 The SENS-GRU

Surprisal-based activation synchronizes the units of each module with regard to the decision on whether to update or skip the current input. This means that each module can end up with a different decision, allowing the possibility for disagreement between the modules. This is a relatively loose constraint which favors

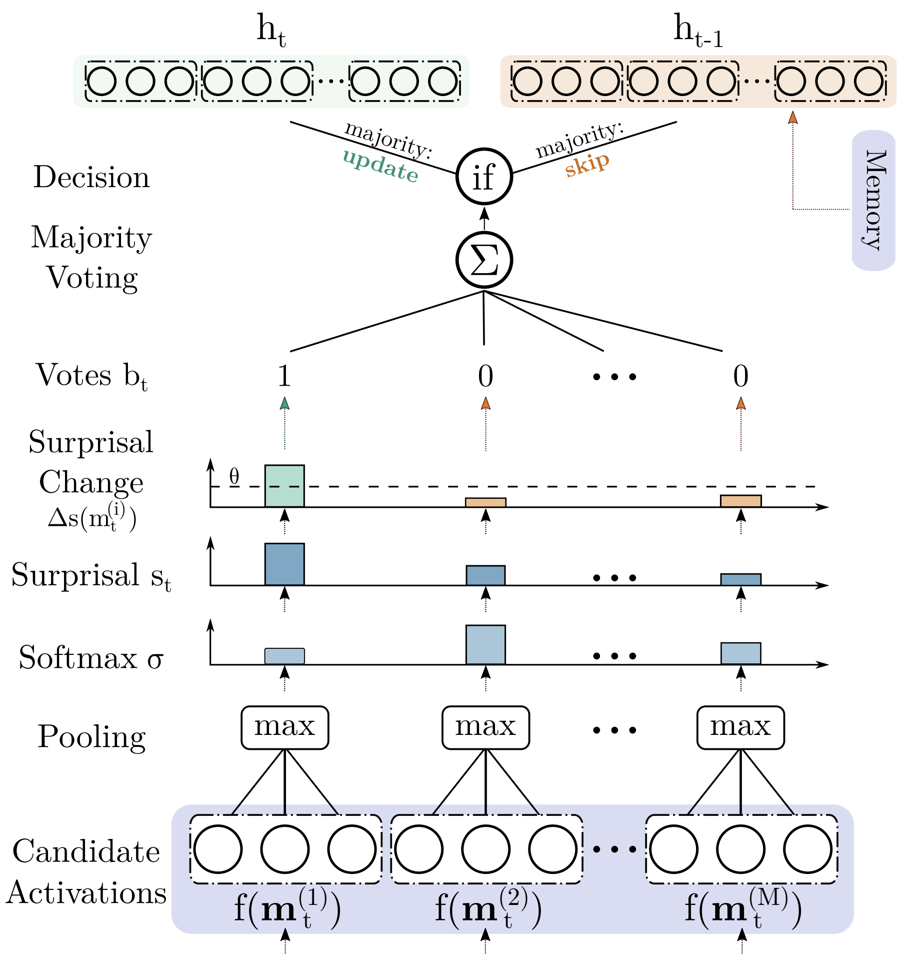


Figure 9.5: Surprisal-based ENSembling (SENS): Instead of activating the modules independently as with SBA, they act as an ensemble and a majority voting process decides whether all modules update or skip together.

increased feature diversity over minimizing the average update rate per layer. As shown in the previous section, the update ranges of SBA seem limited to a certain range. We will now approach this issue by extending SBA such that it outputs a single decision per timestep (and layer) on whether to skip or update.

Our new model differs from SBA in its final steps. It retains the familiar layer partitioning into modules, though they are not treated independently of each other anymore (as was the case in Equation 9.7). Instead, we calculate the resulting update rate from each module's candidate activations and redefine it as the update *probability* of each module. In a final step, we then build an ensemble through a majority voting process of the modules based on these probabilities. The resulting model, which we call *Surprisal-based ENSembling (SENS)*, is depicted in Figure 9.5. Just like with SBA, the basic idea of SENS is mostly independent of the underlying RNN architecture and can also be implemented with an SRN or LSTM. In this work, we adapt SENS to the GRU architecture and name it *SENS-GRU*.

Model Definition

First, we count the number of updates occurring in the M modules of the hidden layer $\mathbf{h}_t$ after applying standard SBA:

$$\mathcal{C}(\mathbf{h}_t) = \frac{|\mathbf{h}_t|}{M} \cdot \sum_{m=1}^{M} [\![s_t^{(m)} > s_{t-1}^{(m)} + \theta]\!], \tag{9.14}$$

where $[\![\ldots]\!]$ indicate Iverson brackets such that $[\![\mathcal{L}]\!] = 1$ if the logical proposition $\mathcal{L}$ is true and $[\![\mathcal{L}]\!] = 0$ if $\mathcal{L}$ is false, and $s_t^{(m)}$ is the pooled surprisal of module m at timestep t. Since the pooling causes every module's unit to have the same surprisal score, we multiply by the size of each module vector $|\mathbf{m}_t| = \frac{|\mathbf{h}_t|}{M}$ to get the total update count $\mathcal{C}(\mathbf{h}_t)$ over all units of the layer. We can then understand this count as the number of "votes" cast from each hidden unit towards updating their own module, based on their module's surprisal scores. The total vote count in the layer can consequently be expressed in relative terms by the ratio of units in $\mathbf{h}_t$ that have voted for an update at timestep t (based on the input word $\mathbf{w}_t$):

$$p_t = \frac{\mathcal{C}(\mathbf{h}_t)}{|\mathbf{h}_t|} \tag{9.15}$$

This voting ratio $p_t \in [0, 1]$ can then be interpreted as the probability for updating the entire layer at timestep t. We can map a probability to a binary decision b_t using a function satisfying $f : [0, 1] \to \{0, 1\}$. For this purpose, we can, e.g., stochastically sample a binary update decision from a Bernoulli distribution $b_t \sim \text{Bernoulli}(p_t)$. However, for a deterministic implementation we simply choose the step function $\text{round}(p_t) = b_t$ such that the layer only updates (towards the candidate $\hat{\mathbf{h}}_t$) if a simple majority of its units vote towards updating:

$$\mathbf{h}_t = \begin{cases} \hat{\mathbf{h}}_t & \text{if } p_t \geq 0.5, \\ \mathbf{h}_{t-1} & \text{if } p_t < 0.5 \end{cases} \tag{9.16}$$

Since $\text{round}(p_t)$ has a binary output, it is not differentiable. To overcome this problem, we use the Straight-Through Estimator (STE), a biased gradient estimator for binary neurons, suggested by Bengio et al. (2013b) (see also Subsection 3.5.4). This estimator approximates the step function by the identity function, thereby passing the incoming gradient "straight through" the node during the backward pass[5]:

$$\frac{\partial \ \text{round}(x)}{\partial x} = 1 \tag{9.17}$$

The resulting SENS-GRU model has several potential advantages over the SGRU. On the one hand, it retains the original module structure which we have empirically seen to affect the self-organization and skipping behavior in the networks.

[5]We use the same estimator for $\mathcal{C}(\mathbf{h_t})$ and any other discrete functions defined in the rest of this chapter.

The pooling operation still takes place and allows for diversity and different focus between modules. On the other hand, the modules are not fully independent of each other anymore, instead forming an ensemble-like structure through majority voting. Most importantly, pooling a single update-skip decision for a layer allows to skip an input token (i.e. word or sentence vectors) in its entirety, further reducing the natural update rate of the network.

9.5 SENS-GRU Results

We train the SENS-GRU on the two CBT datasets using the same hyperparameters as in the previous experiment. Table 9.1 depicts the differences between using the SGRU and the SENS-GRU model in the HAN-ptr architecture in terms of validation and test set accuracy. As indicated in Section 9.3, the SGRU's main benefit is more the reduced update rate compared to the baseline HAN-ptr rather than a noteworthy increase in accuracy. The additional accuracy gains from the SENS-GRU are somewhat limited, though the best configurations improve slightly on both the HAN-ptr and the SGRU. The best results are achieved when using a SENS-GRU layer for encoding the query (except for the test accuracy on the CBT-CN dataset), while encoding the document's words gives similarly good results.

The emerging relationship between validation accuracy and update rate for the SENS-GRU models is depicted in Figure 9.6 for the CBT-CN dataset and in Figure 9.7 for the CBT-NE dataset. Figure 9.6a and Figure 9.7a depict the results in terms of accuracy and update rate for the different encoder types using swarm plots. Here, we include the results for a lower threshold $\theta = 10^{-7}$ to better illustrate the effect of the threshold parameter on both metrics. Figure 9.6b and Figure 9.7b illustrate a linear regression between accuracy and update rate, given different threshold settings and encoder types. Here, we also give Pearson's r and the respective p value for the linear fit.

Compared to the SGRU[6], the swarm plots show stronger differences for the number of modules used in the encoders. This is most apparent when using SENS-GRU word and sentence encoders. While this setup performs slightly worse than the same setup in the SGRU, using two modules ($M = 2$) and a low threshold leads to accuracies comparable to the best models of the other encoders. The largest differences to the SGRU can be seen by the update rates that we can measure with the SENS-GRU: they span over a significantly larger range and we can clearly identify that the update rate negatively correlates with the number of modules that we use. In particular, a higher number of modules, and therefore diversity, seems to clearly reduce the update rates. For the largest threshold setting $\theta = 0.1$, more modules also generally lead to better accuracies. This indicates that a higher

[6]see Figure A.6 and Figure A.7 for a side-by-side comparison between the SGRU and SENS-GRU models.

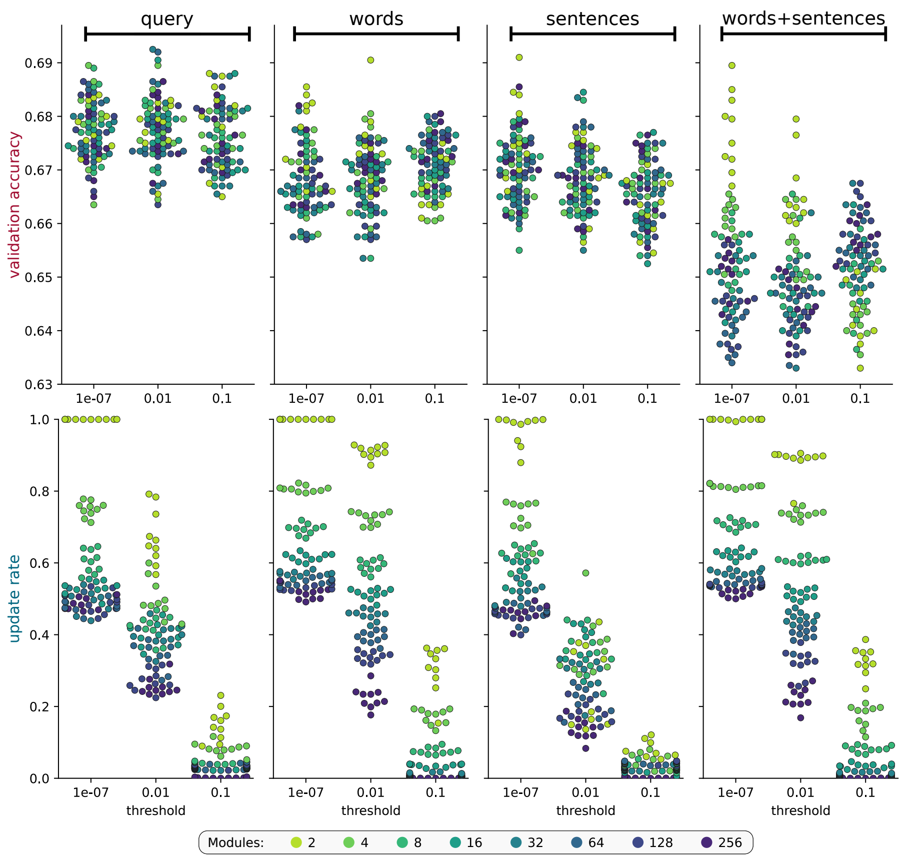
Model	Common Noun valid	Common Noun test	Named Entity valid	Named Entity test
HAN-ptr (baseline)	69.1	**67.7**	75.5	69.9
HAN-SGRU-ptr (q)	69.0	66.0	75.6	70.4
HAN-SGRU-ptr (w)	69.2	65.8	75.3	70.5
HAN-SGRU-ptr (s)	68.8	65.0	74.7	70.0
HAN-SGRU-ptr (ws)	67.5	64.1	74.5	69.8
HAN-SENS-GRU-ptr (q)	**69.3**	66.5	**75.8**	**70.6**
HAN-SENS-GRU-ptr (w)	69.1	65.8	75.6	70.8
HAN-SENS-GRU-ptr (s)	68.5	65.3	74.4	70.2
HAN-SENS-GRU-ptr (ws)	68.0	65.0	74.2	70.0

Table 9.1: Overview and comparison between using the SGRU vs the SENS-GRU with the HAN-ptr architecture on the CBT dataset for the Common Noun and Named Entity categories. Models have been trained using the SGRU/SENS-GRU to encode the query (q), words (w), sentences (s), or both words and sentences (ws). Best accuracies marked in **bold**.

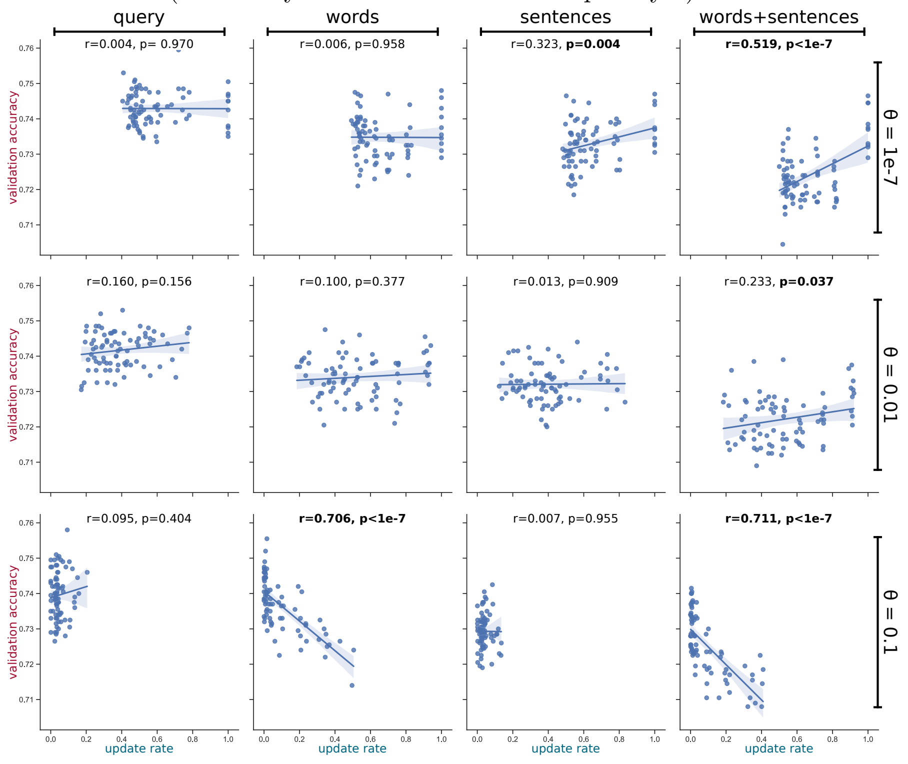
diversity through modularization is highly useful when the network is set to skip frequently according to the chosen threshold value.

While the smallest threshold value $\theta = 10^{-7}$ seems to replicate the results of the original HAN-ptr by updating as much as possible, setting it to $\theta = 0.1$ achieves significantly lower update rates than with the SENS-GRU. In fact, the lowest update rate among the best models is held by the SENS-GRU word encoder model which has an update rate of $u = 0.44\%$ ($\tilde{u} = 0.39\%$) with a validation accuracy of 69.1% for CBT-CN, and $u = 0.03\%$ ($\tilde{u} = -0.02\%$) with an accuracy of 75.5% for CBT-NE. Compared to the SGRU, this yields an update rate reduction by a factor of over 150x and yields an effective update rate $\tilde{u}$ close to 0, which marks the models reaching the lower bound $u_{min}^{word} = 0.05\%$. This is significantly lower than the world-level encoder update rates of the best SGRU model, specifically 18.8% for CBT-CN and 48.4% for CBT-NE.

The general relationship between accuracy and update rate is illustrated in Figure 9.6b and Figure 9.7b. We consider a correlation significant if the respective p value is below a significance level of 0.05. Except for the query encoders, a positive correlation can be seen for most setups in CBT-NE. For the more difficult dataset based on common nouns (CBT-CN), we can observe slightly weaker correlations for the SENS word encoder and, therefore, also when jointly using SENS word and sentence encoders (with $\theta = 10^{-7}$). The same can be observed for the combined word and sentence encoders with $\theta = 0.01$. For the largest threshold $\theta = 0.1$, this setup (word and sentence encoder) leads to a negative correlation, same for the word encoder. Indeed, for both datasets, a *low* threshold results in a *posi-*

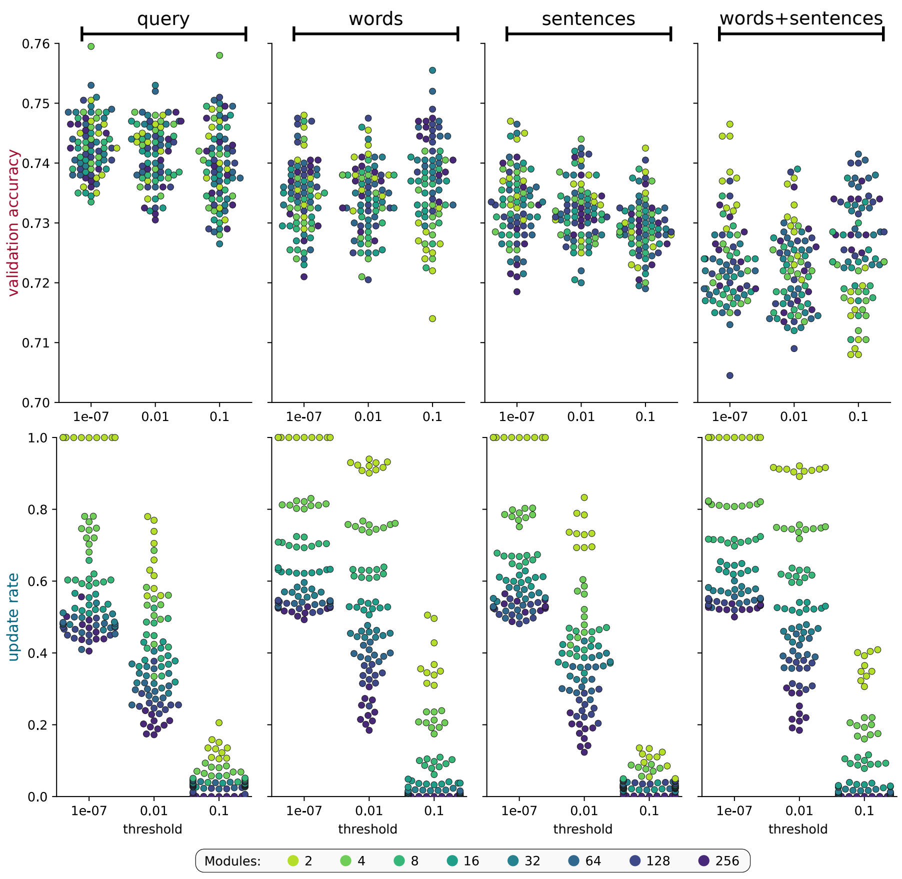


(a) Swarm plots showing accuracy (top) and update rates (bottom) on respective y-axes, threshold θ on every x-axis, and SENS-GRU layer locations in the columns (colored by the number of modules per layer).

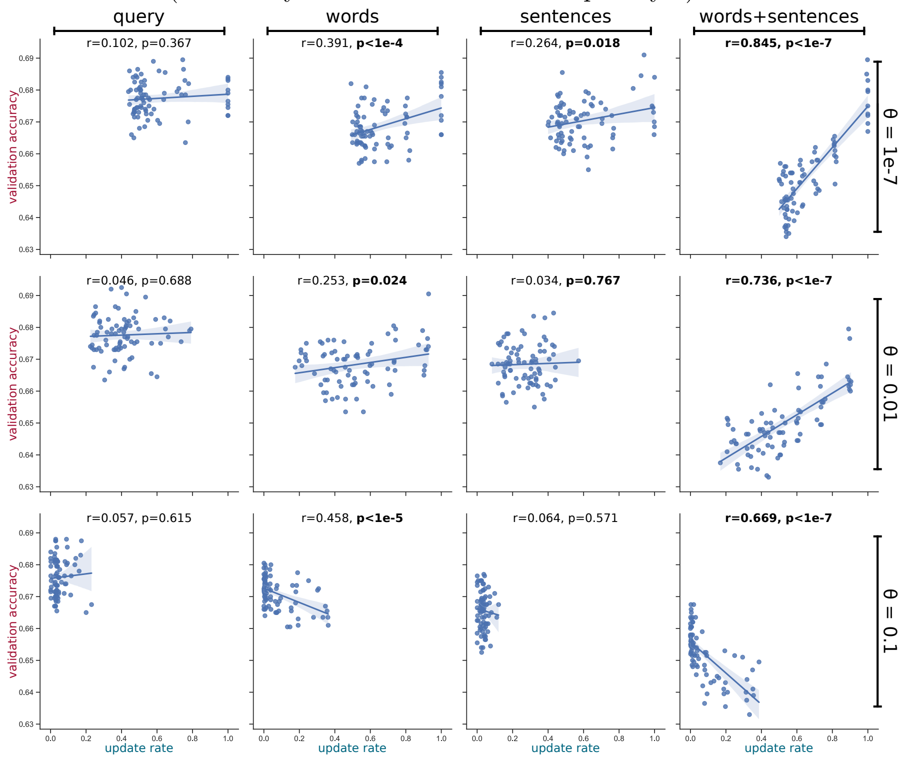


(b) Linear regressions for validation accuracy (y-axis) vs update rate (x-axis). Columns depict on which encoder SENS was applied to, rows the respective threshold value. Each subplot indicates Pearson's r and the respective p-value for the fit. Confidence interval is $c_i = 0.95$.

Figure 9.6: Accuracy vs update rate for SENS-GRU on CBT-CN.



(a) Swarm plots showing accuracy (top) and update rates (bottom) on respective y-axes, threshold θ on every x-axis, and SENS-GRU layer locations in the columns (colored by the number of modules per layer).



(b) Linear regressions for validation accuracy (y-axis) vs update rate (x-axis). Columns depict on which encoder SENS was applied to, rows the respective threshold value. Each subplot indicates Pearson's r and the respective p-value for the fit. Confidence interval is $c_i = 0.95$.

Figure 9.7: Accuracy vs update rate for SENS-GRU on CBT-NE.

tive correlation, whereas a *high* threshold leads to a *negative* correlation. For the medium threshold $\theta = 0.01$, most setups show no strong correlation. These configurations indicate setups where the update rate can successfully be reduced using a SENS-GRU without affecting the baseline accuracy significantly. Furthermore, the negative correlations indicate particularly successful configurations, where the accuracy is highest for the models with the *lowest* update rate.

9.6 Analysis

In order to get a better understanding about the differences between the SENS-GRU and SGRU, we visualize the respective attention layers and skipping decisions with some example documents from the test corpus and directly compare the models against each other. Since the SGRU skips on module-level and the SENS-GRU on word- and sentence-level, we visualize the update rates of the modules on each word for the SGRU and, conversely for the SENS-GRU, based on whether it processed a word or not. The example shown in Figure 9.8 illustrates this difference between the two models: as each module in the SGRU decides individually whether to skip, most words in the document are read by at least some of the modules. This allows the model to capture more of the inputs at the cost of potentially updating some of the modules on irrelevant inputs. The SENS-GRU stands in contrast to this as its modules contribute to a single synchronized decision to either read or skip words and sentences entirely. Consequently, the model reads significantly less parts of the document.

In the shown example, both models also give the correct answer for the given document but the SENS-GRU model seems to only process few key words. On sentence level, the SENS-GRU is able to only focus on sentence 16, which is one of the sentences to contain the correct answer. In the process, other sentences are skipped, even if other correct instances of the answer can be found. This illustrates how skipping in the sentence encoder itself can act as a second filter for previously processed words. For longer paragraphs that do not contain the correct answer (see Figure A.8), both models learn to skim over parts of the sentences. Overall, these examples illustrate how both models present a trade-off between spending less resources reading more (SGRU) and spending more resources reading less (SENS-GRU).

By investigating the bidirectional layers, we also find some differences between the forward layers $\overrightarrow{\mathbf{h}}_t$ and the backward layers $\overleftarrow{\mathbf{h}}_t$ regarding their influence on the update rates. The differences between the two layers are illustrated for both models on word- and sentence-level encoders in Figure A.4. For some of the models, we can see that the more modules M we provide to the models, the higher the update rate difference between backward layer and forward layer. In particular, more modules cause more backwards modules to update, whereas the activity for forward modules goes slightly down. We make this observation for both word- and

SGRU (module-level skipping):

14 **alice** was rather doubtful whether she ought not to lie down on her face like the three gardeners , but she could not remember ever having heard of such a rule at processions ; ` and besides , what would be the use of a procession , ' thought she , ` if people had all to lie down upon their faces , so that they could n't see it ? '
15 so she stood still where she was , and waited .
16 when the procession came opposite to **alice** , they all stopped and looked at her , and the queen said severely ` who is this ? '
17 she said it to the knave of hearts , who only bowed and smiled in reply .
18 ` idiot ! '
19 said the queen , tossing her head impatiently ; and , turning to **alice** , she went on , `what 's your name , child ?'

SENS-GRU (token-level skipping):

14 **alice** was rather doubtful whether she ought not to lie down on her face like the three gardeners , but she could not remember ever having heard of such a rule at processions ; ` and besides , what would be the use of a procession , ' thought she , ` if people had all to lie down upon their faces , so that they could n't see it ? '
15 so she stood still where she was , and waited .
16 when the procession came opposite to **alice** , they all stopped and looked at her , and the queen said severely ` who is this ? '
17 she said it to the knave of hearts , who only bowed and smiled in reply .
18 ` idiot ! '
19 said the queen , tossing her head impatiently ; and , turning to **alice** , she went on , ` what 's your name , child ?'

Question:
` my name is **xxxxx** , so please your majesty , ' said alice very politely ; but she added , to herself , ` why , they 're only a pack of cards , after all .

Candidates:	Answer:
alice first king knave miss queen queens dears processions tulip-roots	**alice**

per-word update rate: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
per-sentence update rate: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

Figure 9.8: Example comparison between SGRU **(top)** and SENS-GRU **(bottom)** on a paragraph from the test set. Different to the SGRU, the SENS-GRU updates and skips on input-token-level, i.e. it can fully ignore words and sentences. The highlighted values depict the average between forward and backward layers for the bidirectional SGRU and whether one of the two layers updated for the bidirectional SENS-GRU.

sentence-level encoders. Since having more modules leads to an increased diversity in the activation distribution, this suggests that the activation diversity is partly tied to how much the model utilizes in the backward layer. A potential explanation for this could be that more fine-granular skipping in the SGRU (i.e., larger M) could cause a more fine-grained utilization of the backward encodings.

Since the model's output is determined by the pointer attention, it is also important to consider the interplay between the skipping and attention distributions. For this, we visualize both skipping and attention on an example document (compare Figure 9.9). In the shown example, most updates are performed by both parts of the bidirectional layer for the SGRU, whereas it is generally one of the two for the SENS-GRU. As the SGRU uses more parameters to encode more parts of the document, its final pointer output has a higher confidence than the SENS-GRU. The SENS-GRU reads the correct answer in sentences 11, 14, and 16. However, the word is only attended in sentence 11, driving the network to predict the answer based on this sentence even though sentence-level attention for this sentence is lower than for sentence 14. Both examples illustrate how the pointer mechanism is

SGRU (module-level skipping):

11 (0.30) the **king** was (0.98) greatly surprised at his valour , and said he knew no one like him , and thanked him heartily for what he had done .
12 (0.03) after (0.14) this the **king** set ring next to himself (0.75) , and all esteemed him highly , and held him to be a great hero ; nor could red any longer say anything against him , though he grew still more determined to destroy him .
13 (0.00) one day a good (0.91) idea came into his head .
14 (0.09) he came to the **king** and said he had something (0.97) to say to him .
15 (0.40) ` what is that ? ' (0.37)
16 (0.17) said the **king** (0.97) .
17 (0.00) red said that he had just remembered the gold cloak , gold chess-board (0.81) , and bright gold piece that the **king** had lost about a year before .
18 (0.00) ` do n't remind (0.88) me of them ! '
19 (0.00) said (0.67) the **king** .(0.67)

SGRU Pointer: **king(0.93)**

SENS-GRU (token-level skipping):

11 (0.04) the **king** (0.71) was (0.16) greatly surprised at his valour , (0.11) and said he knew no one like him , and thanked him heartily for what he had done .
12 (0.02) after this the **king** set ring next to himself (0.95) , and all esteemed him highly , and held him to be a great hero ; nor could red any longer say anything against him , though he grew still more determined to destroy him .
13 (0.01) one day a good (0.96) idea came into his head .
14 (0.55) he came to the **king** and said he had (0.32) something (0.64) to say to him .
15 (0.20) ` what is that ? ' (0.52)
16 (0.08) said the (0.76) **king** .
17 (0.00) red said that he had just remembered the gold cloak , gold chess-board (0.88) , and bright gold piece that the **king** had lost about a year before .
18 (0.00) ` do n't remind (0.93) me of them ! '
19 (0.00) said the **king** .(0.88)

SENS-GRU Pointer: **king(0.40)**

Question:
red , however , went on to say that , since ring was such a mighty man that he could do everything , it had occurred to him to advise the **xxxxx** to ask him to search for these treasures , and come back with them before christmas

Candidates:	Answer:
dog **king** prince red snati attack chess-board day left year	**king**

Legend: update (forward layer) / update (backward layer) | word attention: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 | sentence attention: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

Figure 9.9: Text example comparing the SGRU and SENS-GRU. Word and sentence attention highlighted by red/blue background (attention values above 0.1 in brackets). **Left**: Since the SGRU can update with some modules and skip with others, shown underlines denote instances where more than 40% of modules update. Conversely, not underlined tokens are still processed but with less units. **Right**: For the SENS-GRU, blue underlines denote instances where a token is updated by the forward-layer and red underlines those updated by the backward layer.

able to correct for small mistakes by earlier layers based on the candidate vector.

9.7 Chapter Summary

Attention and surprisal-based activation provide two alternative filters for focusing on important inputs. In this chapter, we have investigated the interplay between these two salience models by integrating surprisal-based activation with the hierarchical attention network. To provide a fair comparison with baseline GRUs, we have proposed the SGRU, an adaptation of SBA to the GRU architecture. Our results for the SGRU show that we can improve model accuracy by increasing its degree of modularity, suggesting a simultaneous increase in diversity.

Our second goal was to minimize any trade-offs between accuracy and update rates. We have found the update rates of the SGRU to be limited to small ranges. Along with an analysis of a lower bound for skipping, we have found some evidence that the models have additional potential to skip even more tokens in the given QA task. To investigate this further, we have transferred the local per-module decision-making process of the SGRU to a global per-layer skipping process with the SENS-GRU by using a majority voting process between modules. For this extended model, we have found some weak or even negative correlations between accuracy and update rate. In these cases, the model is able to maintain a high accuracy despite large reductions in the update rates. In some instances, it is even able to surpass the baseline accuracy of the HAN-ptr. This highlights the benefit of pooling the local modular update decisions to a single synchronized decision that is globally executed within each layer. For the following chapter, we will keep this design decision but provide a direct comparison between these two different models of salience, namely attention and surprisal, by implementing skipping based on attention.

Chapter 10

Attend, Skip, and Point: Question Answering with Hierarchical Attention-Based Skipping

For our final chapter, we will continue to investigate the effect of attention on skipping in hierarchical architectures for cloze-style question answering. While we have previously focused on the emerging representations when skipping and attention are modeled separately, we will now combine both approaches into a single layer, which we call *Attention-based Skipping (ABS)*. By utilizing salience from attention to skip irrelevant tokens, we are able to model an approach without the need for additional hyperparameters. Despite the lack of a strong inductive bias in the skipping constraints, we show that ABS consistently skips most of the words in the documents, compressing representations at the theoretically possible limit for lossless compression. Additionally, we present an alternative formulation with *ranked attention* which allows us to precisely control the desired update rate of the model. This allows us to have a closer look at the correlation between model accuracy and update rates, which we find to be very weak for some of our models, demonstrating that they can maintain baseline accuracy despite skipping over the overwhelming majority of words in the queried documents.

10.1 Attention-Based Skipping

Recent work has investigated the utility of attention to analyze the importance of features in order to compress or prune networks. For example, some recent approaches have used attention masks to identify image features for compression (Chen et al., 2019c; Liu et al., 2019a) or to compress channels (Yamamoto and

Maeno, 2019). In the domain of natural language processing, See et al. (2016) have investigated pruning recurrent neural networks trained for machine translation and have found that lower layers and embeddings hold the most redundancy, whereas the attention and softmax weights are the most important for identifying important features. Similarly, early approaches on sentence compression with LSTMs have proposed mapping words to binary decisions, signifying whether a word is to be deleted or kept (Filippova et al., 2015). Attention has also been used to efficiently find the context for a word in the context of sentence compression (Tran et al., 2016b). Most recently, Kovaleva et al. (2019) have shown that disabling attention heads can be useful for pruning BERT models, which are based on transformers with self-attention.

Despite these advances, there is a very limited amount of work on how attention can be used to skip words in the conditional computation framework. Somewhat related to the idea of conditional computation is Adaptive Computation Time (ACT; see also Subsection 3.5.6), which allows to allocate a dynamic number of training iterations per input token, skipping nothing but allowing a "rereading" of inputs. Neumann et al. (2016) have pooled attention representations to better decide the amount of time that is spent on each entry of the input sequence by ACT. This suggests that a similar mechanism could be used for identifying redundant input for skipping.

Moreover, attention has previously been used in order to skip input words by Hahn and Keller (2016). In their work, a reader network computes a probability distribution over the input at each timestep and a decoder, an attention network with hard attention trying to reconstruct the reader's input, uses this probability to decide which inputs to skip. They model and predict human text skimming behavior, achieving results close to a surprisal-based baseline. Different from our presented approach, however, they do not directly use the attention distribution itself to model skipping behavior. Other approaches have modeled sparse attention through multiple passes on the attention layer: Sparse Attentive Backtracking (SAB) implements a sparse replay mechanism after retrieving memory through attention mechanisms (Ke et al., 2018a) which is in principle similar to our approach even though ours does not require a modification of the underlying LSTM model. Sparse attention is also used by Recurrent Independent Mechanisms (RIMs; Goyal et al. (2019)) which allow a specialization of modules over temporal patterns. Neural Function Modules take this concept one step further performing two passes with bidirectional feedback between lower and higher layers based on attention (Lamb et al., 2020).

In the following, we will present our Attention-based Skipping (ABS) approach. The key idea to our model is to utilize the attention distribution as the update probability for the encoders on word- or sentence level. The simplicity of our approach has the upside of not introducing any additional hyperparameters and offering compatibility to widely-used recurrent and attention models.

10.1.1 Model Definition

To recap, the word-level attention $\alpha_{i,j}^{word}$ is computed in the Hierarchical Attention Network with pointer attention (HAN-ptr; see also Section 8.2) from the dot product of the query $\mathbf{q}$ and the word-level encoding $\mathbf{D}_{i,j}^{word}$:

$$\beta_{i,j}^{word} = \mathbf{q}^T \cdot \mathbf{D}_{i,j}^{word} \tag{10.1}$$

$$\alpha_{i,j}^{word} = \text{softmax}(\beta_{i,j}^{word}), \tag{10.2}$$

where i depicts the sentence index and j the word index. The sentence attention is then calculated in a similar fashion, except that the second input to the attention layer constitutes the composed sentence vector $\alpha_{i,j}^{word}$:

$$\beta_{i}^{sent} = \mathbf{q}^T \cdot \mathbf{D}_{i}^{sent} \tag{10.3}$$

$$\alpha_{i}^{sent} = \text{softmax}(\beta_{i}^{sent}) \tag{10.4}$$

The word- and sentence-level attention are then blended together to have a document-level attention score which is then used by the pointer attention to compare against the candidates:

$$\alpha_{i,j}^{doc} = \alpha_{i}^{sent} \cdot \alpha_{i,j}^{word} \tag{10.5}$$

Both α_{i}^{sent} and $\alpha_{i,j}^{word}$ depict attention scores that model the importance of each input token as a probability distribution of the respective sequence. With the goal of only updating for important words and sentences, and skipping over others, we can therefore directly utilize this distribution to assign update probabilities to each input token. More generally, we model this stochastically by sampling binary decisions $b_t \in \{0, 1\}$ from a Bernoulli distribution with probability α_t such that $b_t = 1$ updates the layer on seeing the i-th input token at timestep t and $b_t = 0$ skips it[1]:

$$b_{i,j} \sim \text{Bernoulli}(\alpha_{i,j}^{word}) \tag{10.6}$$

$$b_{i} \sim \text{Bernoulli}(\alpha_{i}^{sent}) \tag{10.7}$$

By doing so, we are giving the attention mechanism full control over which states are updated. As a consequence, this allows us to use *any* type of recurrent layer in the pre-attention stage and we do not need to modify existing models like with Surprisal-based Activation. In addition, we can use any type of attention mechanism without modification, as long as it outputs a probability distribution which models the relevance of each input token.

After using the attention to get the update decisions, we can then modify the previously computed candidate activations $\hat{\mathbf{h}}_t$ to skip for inputs where $b_t = 0$:

$$\mathbf{h_t} = \begin{cases} \hat{\mathbf{h}}_\mathbf{t} & \text{if } b_t = 1, \\ \mathbf{h_{t-1}} & \text{if } b_t = 0 \end{cases} \tag{10.8}$$

[1]Note that $t = i$ since we assume no partitioned layers. The entire layer either updates or skips for the i-th input token that is presented in timestep t.

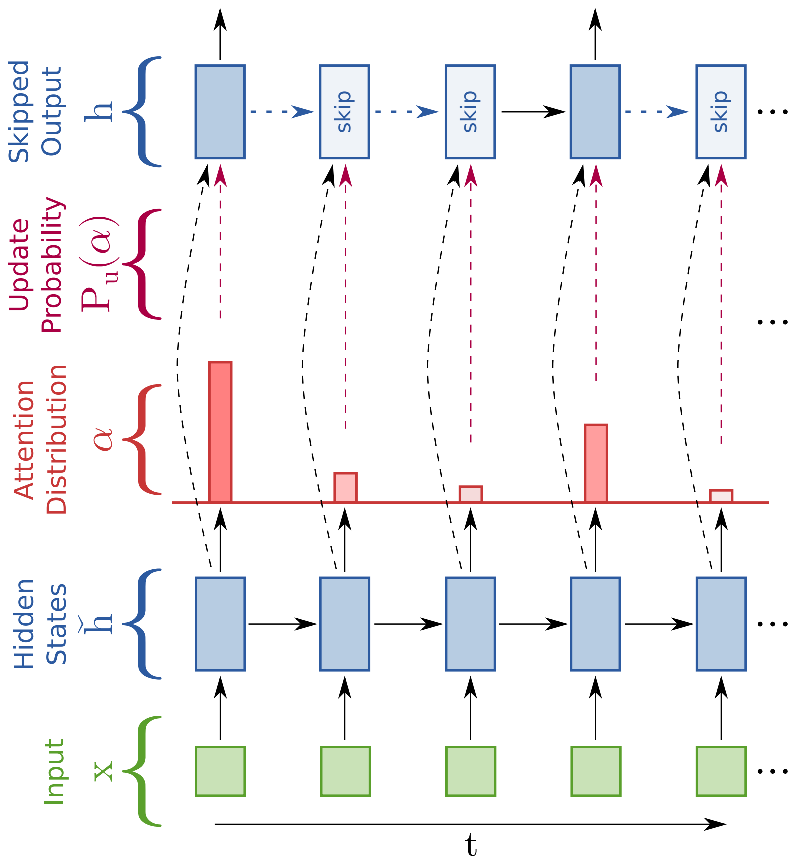


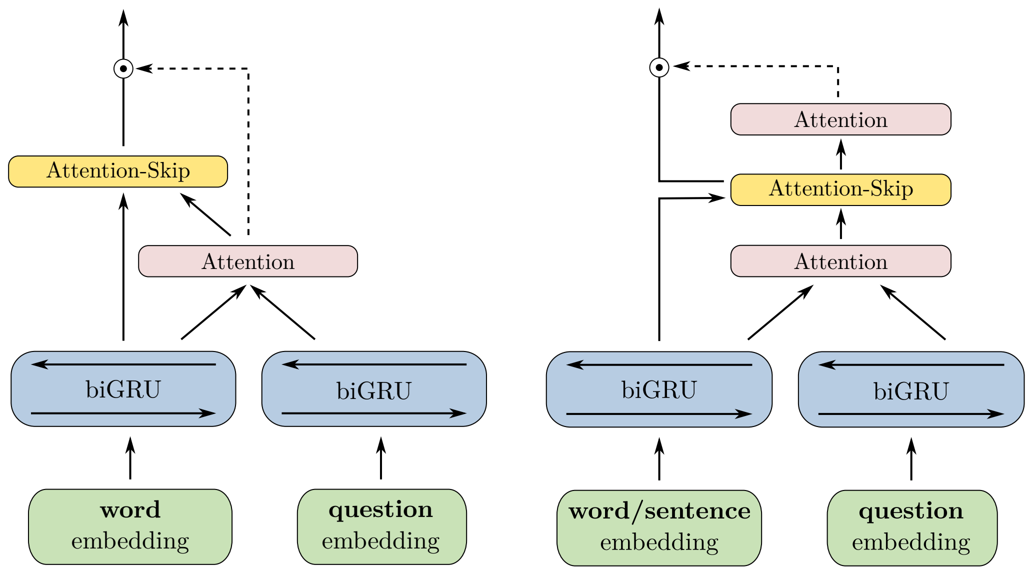
Figure 10.1: Attention-based Skipping. Dotted arrows denote information passed between layers whereas solid arrows denote parameter updates.

Since the attention layer requires the previous recurrent layer's output, it is impossible for the recurrent layer to predict its own attention distribution *before* calculating its own updates. This means, that our approach can only be applied *in hindsight*: after the recurrent layer's output is known, we can calculate the attention scores and use them as update probabilities to mask the already computed updates in the recurrent layer[2]. After masking, this new recurrent layer then contains skipped tokens for all low attention scores.

The entire process for Attention-Based Skipping (ABS) is illustrated in Figure 10.1 and can be summarized in these 4 steps:

1. Calculate the hidden states $\hat{\mathbf{h}}_t$.
2. Calculate the respective attention scores α_t.
3. Replace $\hat{\mathbf{h}}_t$ with $\mathbf{h}_{t-1}$, i.e. skip with probability $p = 1 - \alpha_t$.
4. Forward the resulting vector to the higher layers as $\mathbf{h}_t$.

[2]This allows us to be more computationally efficient by avoiding a complete recalculation of the recurrent updates and storing a second layer representation in the memory. Consequently, the introduced computational overhead is of linear complexity and in $O(|\mathbf{h_t}|)$.



(a) ABS-AS **(b)** ABS-ASA

Figure 10.2: Attention-based Skipping (ABS) with the ABS-AS (attend & skip) model compared to the ABS-ASA (attend, skip, attend) model. For a complete illustration of the resulting hierarchical models, see also Figure B.6 and Figure B.7.

This does, however, open up an interesting question: since the new output $\mathbf{h}_t$ would lead to different attention scores, if we were to attend to it *again*, should we do so? In fact, the cyclical dependency between the recurrent and attention layer allows us to iterate through steps 1-4 an arbitrary number of times, yielding different results each time. Nevertheless, since each iteration reduces the sequence entropy, multiple iterations are arguably limited in their practical usefulness as there is increasingly less unique tokens to attend to.

In the context of the HAN-ptr model, the state vector $\mathbf{h}_t$ is used at the word-level to compose the sentence vector. On the other hand, in the final step, the pointer attention only considers the word-and sentence-level attention. From this, we can see that repeating the attention has a different effect than simply doing it once and forwarding $\mathbf{h}_t$. While the latter would "conserve" the original word-level attention values, but impact the sentence vector composition due to skipped words, the former would build new attention values that reflect what has been skipped on word-level, more directly impacting the pointer attention.

To investigate this open question, we define two different ABS approaches. In the first, we Attend and Skip (ABS-AS), conserving the original attention values based on the candidate activations computed before the skipping takes place. This approach corresponds to the 4 steps discussed above. In the second, we Attend, Skip, and Attend (ABS-ASA), reapplying attention on the encoding after executing the necessary skips. Both approaches can be compared in Figure 10.2. At the sentence level, the recurrent sentence *encoding* is only needed to calculate the sentence attention for the pointer, being discarded afterwards. Therefore, ABS-AS can only be used on word-level. ABS-ASA, on the other hand, can be used on word- and sentence level, since the sentence-level *attention* is modified by the changes in the encoder.

10.1.2 Evaluation

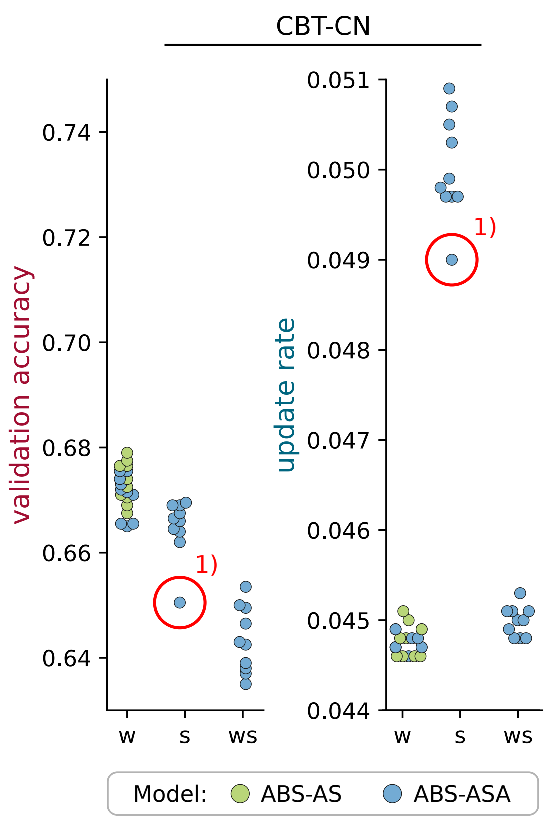
We evaluate the two proposed models in the HAN-ptr architecture. To ensure comparability, we train on the CBT dataset using the same hyperparameter search space as in the previous experiments with 10 trials for each configuration. Since the query encoding is given as a context vector, we do not skip any query embeddings. We evaluate the use of ABS-AS on the word-level attention, and ABS-ASA on word- and sentence level (both independently and jointly). As before, the layers without ABS operate normally utilizing regular GRUs.

The results are shown in Figure 10.3. Both models converge to very low update rates in the range of $[0.044, 0.051]$ for CBT-CN and $[0.048, 0.051]$ for CBT-NE. While we have observed similar rates in some SENS-GRU models (see Section 9.5), ABS achieves this more consistently in every trial, despite the stochasticity from sampling. This indicates a stronger structural bias, either from the presented data or the model itself. From our previous analysis in Subsection 8.3.6, we know that the attention softmax tends to either produce a single input token with high probability or multiple tokens with low probabilities. This bias seems to be reflected in the update rates, as ABS seems to tend towards processing slightly less than 1 word per sentence on average[3]. Indeed, the average update rate of $u = 0.0488$ for the word-level ABS models on CBT-NE corresponds to an effective update rate of $\tilde{u} = 4.83\%$, whereas the average $u = 0.0447$ for CBT-CN corresponds to $\tilde{u} = 4.78\%$, meaning that the word update rates can only be reduced by an additional 4.8% before reaching the lower bound $u_{min}^{word} = 0.0005$. In other words, both models achieve 95.2% of the potential gains.

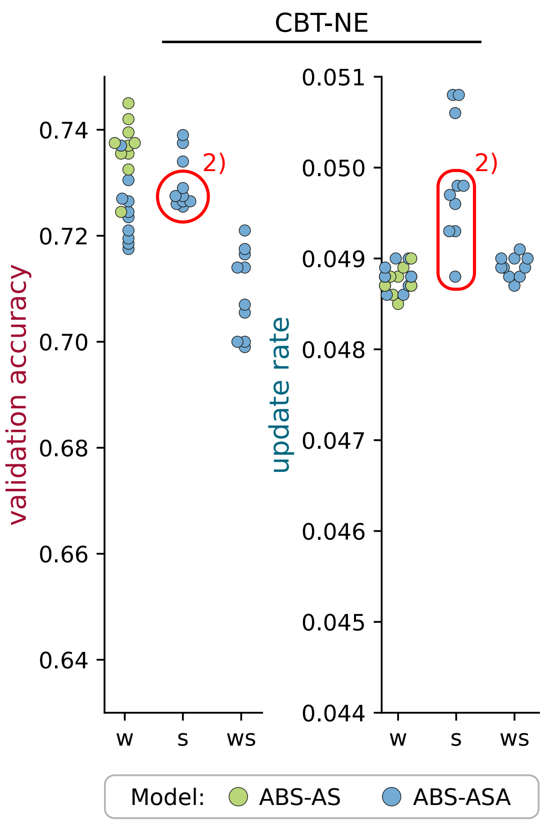
Interestingly, the sentence-level ABS-ASA model actually reaches the lower bound $u_{min}^{sent} = 0.05$. In a majority of cases, it even goes slightly below this threshold. This happens for 6 models with CBT-CN, even though only the model with the lowest update rate drops significantly in accuracy by 1% (red circles marked **(1)** in Figure 10.3). For CBT-NE, we report a similar drop in accuracy for all 7 models that are below this threshold (red circles marked **(2)** in Figure 10.3). This outperformance is reflected by the average-based effective update rates $\tilde{u} \approx u_{min}^{sent} = 0\%$ for CBT-CN and $\tilde{u} = -0.02\%$ for CBT-NE.

As discussed in Subsection 9.2.3, there are a number of explanations as to why a negative effective update rate $\tilde{u}$, i.e. an update rate u below u_{min}^{sent}, does not necessarily have to lead to worse accuracy. Since CBT-CN represents the more difficult task with a lower baseline accuracy, the results suggest that the model, or

[3]Where processing *exactly* 1 word per sentence would lead to $u_{min}^{sent} = 0.05$.



(a) CBT-CN



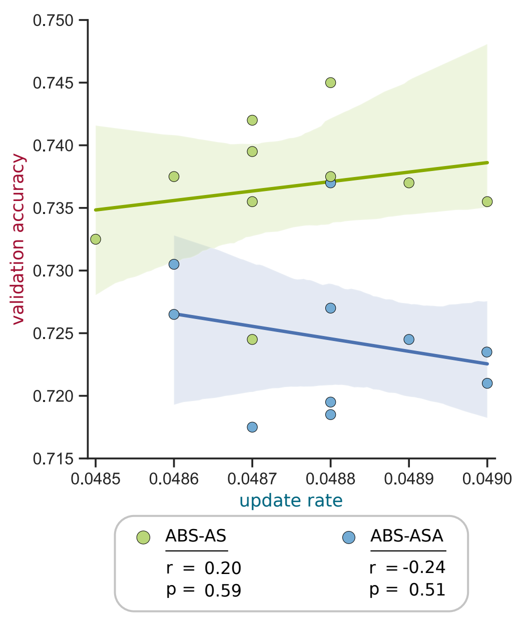
(b) CBT-NE

Figure 10.3: Validation accuracy and update rate from using ABS-ASA (blue) in the word- (w) or sentence-level (s), as well as both (ws). ABS-AS (green) only skips on word-level. Red circles depict models with an update rate below $u_{min}^{sent} = 0.05$ and consequently lower accuracies.

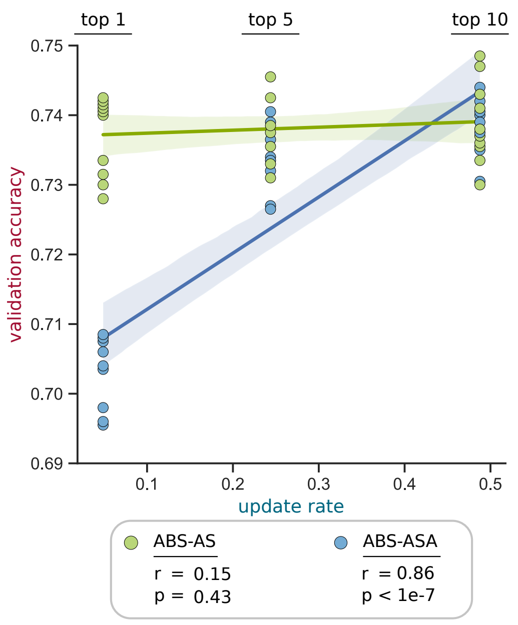
one of the bidirectional layers, sometimes skips over some unsolvable documents entirely, therefore not impacting the baseline accuracy negatively.

Overall, we can report a better average accuracy for the ABS-AS model than for the ABS-ASA model. Even though they lead in terms of achieved update rates, both models are slightly behind the best SGRU and SENS-GRU models in terms of accuracy. The limited range of the emerging update rates suggests a very strong tendency for the models to minimize the amount of tokens that are attended to and processed in the representation. Based on this, we hypothesize, that artificially increasing the update rate can improve the corresponding task accuracy.

To this end, we probe the ABS layers by only activating the words with the highest attention scores. For each experiment, we update exactly k words per document and compare the respective impact of three different setups, choosing $k \in \{1, 5, 10\}$. Figure 10.4 shows a comparison between using the regular stochastic update process in ABS (Figure 10.4a), and deterministically filtering out the most attended k words (Figure 10.4b). For regular ABS, we report no significant correlation be-



(a) Regular ABS: Updating each word with probability $P(b_{i,j} = 1) = \alpha_{i,j}^{word}$.



(b) Only updating the $k \in \{1, 5, 10\}$ words with the highest attention score.

Figure 10.4: (a) Correlation (Pearson's r and p-value) between accuracy and update rate between ABS-AS and ABS-ASA. **(b)** Correlations when only the top k attention scores are picked per sentence (with k being the number of processed words). Depicted models are word-level encoders on CBT-NE. Bands depict confidence intervals with $c_i = 0.95$.

tween accuracy and update rate ($p = 0.59$ for ABS-AS and $p = 0.51$ for ABS-ASA). This is mostly unsurprising due to our above conclusion that most models roughly yield the same update rate. It is similarly expected that updating only 1 word per sentence ($k = 1$) leads to $u_{min}^{sent} = 0.05$. However, increasing the number of updated words per sentence to $k = 5$ and $k = 10$ increases the update rate dramatically for the ABS-ASA model (albeit only slightly for the ABS-AS model), eliminating the performance gap between both models. This increase is reflected in a significant positive linear correlation ($r = 0.86$ and $p < 10^{-7}$), signaling that increasing the number of updated words increases the accuracy.

In the next section, we will introduce a simple modification to ABS which will allow us to freely assign the desired update rate as a hyperparameter for each model. This will enable us to more directly tune the resulting equilibrium between the accuracies and update rates. In the process, we will also evaluate which of the two ABS models gives the best trade-off between both metrics.

10.2 Modulating the Update Rate with Ranked Attention

As demonstrated in the previous section, the stochastic update function leads to a very narrow range of possible update rates, which makes it difficult to fully understand the underlying relationship between accuracy and update rate in ABS. To approach this issue, we propose a ranking method which allows us to directly control the update rate in the network. We propose a simple scheme in this section that we call *ranked attention*.

10.2.1 Ranking the Attention Scores

The main idea of ranked attention is to only activate hidden units with the highest attention score. We have demonstrated a basic implementation of this idea in the previous section, where we selected k most attended words per sentence. To be able to modulate the number of activated units as precisely as possible, we describe the desired update rate as a simple function of the p-th percentile $\eta_{.p}$ of the attention scores. To consider an example, defining an update rate of $u = 10\%$ will lead to an activation of all hidden units that have attention scores above the 90-th percentile $\eta_{.90}$ which is approximately 10% of the states. Following this example, we define the relationship between the desired update rate u and the corresponding percentile p of affected attention scores as:

$$u \approx 1 - p \tag{10.9}$$

To define a ranking operator based on this property, we start by ranking the attention score α_t of each hidden unit $\mathbf{h_t}$ (with $1 < t < |S|$, where $|S|$ defines the sequence length), such that they are sorted according to $\alpha_i < \alpha_j$ iff $i < j$. Then, $u(\%)$ of the highest activations are greater than the value defined by $\eta_{.(1-p)}(\alpha)$. We use nearest-rank interpolation to round the percentile rank $u \cdot |\boldsymbol{\alpha}|$ to the index p of the attention value closest to it. The resulting value $\eta_{.u} = \alpha_\theta$ is then the smallest value of the p-th percentile which marks the lower bound for the highest $u(\%)$ of the attention scores. We can use this threshold to only activate the layer $\mathbf{h}_t$ at timesteps t where the attention values are above this level. We define the resulting rank-operator $\mathcal{R}$ as follows:

$$\mathbf{h}_t = \mathcal{R}(\hat{\mathbf{h}}_t) = \begin{cases} \hat{\mathbf{h}}_t & \text{if } \alpha_t > \alpha_\theta \\ \mathbf{h}_{t-1} & \text{otherwise,} \end{cases} \tag{10.10}$$

where the attention score α_t of the hidden unit candidate activation $\hat{\mathbf{h}}_t$ determines the binary update decision, depending on whether it is above the p-th percentile (identified by α_θ) or not. Considering that the attention values are defined per timestep, all hidden units of a layer either update or skip in synchronized fashion, as is the case for normal ABS.

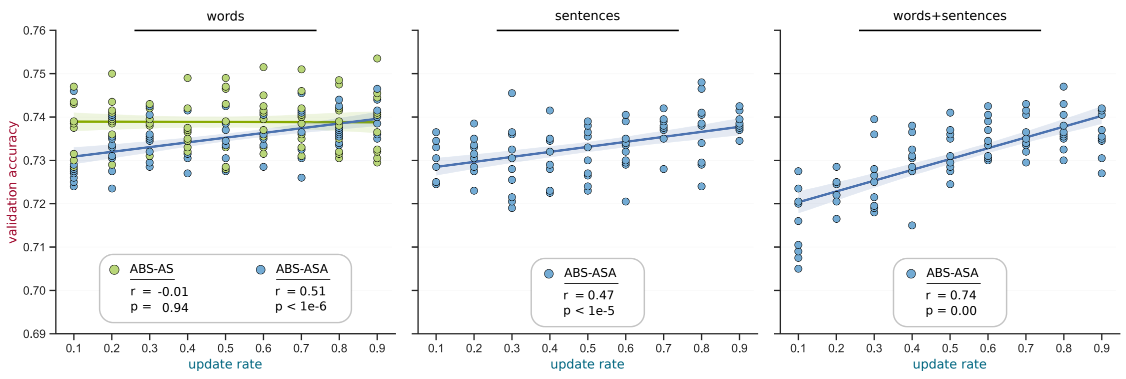


Figure 10.5: Results for ABS with ranked attention: correlation between accuracy and update rate (Pearson's r and p-value).

10.2.2 Evaluating ABS with Ranked Attention

Our main intent with ranked attention is to provide a more in-depth analysis for the ABS models, by being able to modulate their update rates. The key question that we investigate is whether increasing the update rate also increases accuracy and under which circumstances both ABS models yield similar results on word-level.

We train both models, the ABS-AS and the ABS-ASA, with ranked attention on the CBT-NE dataset using update rates of $u \in \{0.1, ..., 0.9\}$, and measure the correlation between validation accuracy and update rate. The results are visualized in Figure 10.5[4]. For the ABS-ASA model, we report statistically significant correlations (below the significance level $p < 0.05$), which are moderately positive on word-level ($r = 0.51$) and sentence-level ($r = 0.47$), whereas using ABS on both levels gives a strong positive relationship ($r = 0.74$). There is no significant correlation with the ABS-AS model, though we can see how the maximum accuracy slightly improves with increased update rates.

The observation that accuracy and update rate are uncorrelated for the ABS-AS model can by explained by two simultaneous effects. First, it can result if the majority of activations that are skipped correlate to low attention values. This would mean that increasing skip rates has no impact on accuracy as it only affects tokens with low attention. Additional support for this hypothesis comes from our previous observation that most of the attention seems to be concentrated on a small number of words in each document. Secondly, this would require models with low update rates to capture most of the essential tokens, as increasing their update activity does not significantly larger accuracy gains. Since the ABS-AS model operates under the inductive bias that all points of interest are provided by the attention mechanism itself, both explanations would be a consequence of the model working as intended, which is confirmed by the result that accuracy and

[4]A different perspective on the same data is additionally provided in the form of a swarm plot in Figure A.5.

update rate are not correlated. Consequently, the results for ABS-ASA suggest that reapplying attention leads to more mistakes with lower update rates.

By comparing both models on word-level, we can conclude that the ABS-AS model performs better with regard to both metrics. It yields the best accuracy for every update rate, though it also has a slightly larger accuracy variance ($SD = 0.60\%$) than the ABS-AS model ($SD = 0.55\%$). Therefore, it yields a better accuracy per read word. Even though both models yield a similar top accuracy for $u = 0.1$, this is mostly based on a statistical outlier from ABS-AS. On average, and based on the intersection of the two linear regression lines, both models perform most similarly with an update rate of $u = 0.8$. In conclusion, these results suggest the ABS-AS model should be the preferred choice to implement attention-based skipping.

10.3 Comparison and Analysis

In this section, we will provide a full overview on how the ABS models compare to our previous SBA approaches. To gain a better understanding of the differences between the different models, we discuss additional example visualizations of the models solving the QA task.

10.3.1 Comparison to Previous Approaches

The fact that the ABS-AS model gives no significant correlation between accuracy and update rate, suggests that conserving the original attention values is more beneficial than updating them based on a skipping layer. This indicates that the attention layer has an overall larger impact than composing the sentence vector from skipped words. Since the sentence vector's ultimate purpose is to generate sentence attention scores for the pointer attention, it therefore seems that the pointer model has a slight bias towards focusing more on words than sentences. Ultimately, a high attention score for sentences is only useful if they contain the answer itself or a necessary semantic relationship to find the answer. Consequently, skipping in sentences which do not meet these criteria, is expected to have a low impact, potentially explaining this bias.

Table 10.1 summarizes our experiments on the CBT dataset. To provide a comparison to other approaches, we also train word encoders with the CWRNN (Chapter 4) and CWLSTM (Chapter 5) on the task. For the CWLSTM, we train all possible skip target variants and module configurations with exponentially increasing update frequencies. We find the best validation results and lowest differences ($< 0.1\%$ on CBT-NE, $< 0.7\%$ on CBT-CN) between all skip targets for periods $P = (1, 2, 4, 8)$, with the skip target $\mathbf{h}$ slightly ahead. As the update rates of the Clockwork models are fixed for a given number of modules and $|\mathbf{h}|$ (Koutník et al., 2014), they all lead to constant update rates of $u = 55.5\%$. Overall, both

	Common Noun			Named Entity		
Model	valid	test	u	valid	test	u
Attention Sum Reader Network**	68.8	63.4	100*	73.8	68.6	100*
RAN	60.8	57.4	100*	64.4	58.7	100*
HAN-doc-vec	60.4	56.4	100*	62.9	57.7	100*
HAN-ptr	69.1	67.7	100*	75.5	69.9	100*
HAN-ptr (CWRNN)	63.1	59.4	55.5*	69.4	65.12	55.5*
HAN-ptr (CWLSTM)	66.6	63.0	55.5*	72.7	67.6	55.5*
HAN-SGRU-ptr	69.2	65.8	18.8	75.3	70.5	48.4
HAN-SENS-GRU-ptr	69.1	65.8	0.44	75.6	70.8	0.03
HAN-ABS-AS-ptr	67.9	64.4	0.04	74.5	69.5	0.05
HAN-ABS-AS-ptr (RA)	69.1	64.0	30*	75.4	69.3	90*
HAN-ABS-ASA-ptr	67.6	64.5	0.04	73.7	68.6	0.05
HAN-ABS-ASA-ptr (RA)	69.0	64.0	80*	74.7	70.4	90*

Table 10.1: Overview and comparison of all the proposed models in this thesis (deployed on word-level with $u_{min}^{word} = 0.05\%$ as a lower bound) for question answering, based on the best accuracy on the CBT dataset for the Common Noun (CN) and Named Entity (NE) categories. All update rates u and accuracies reported in percentage. RA = Ranked Attention, *: static update rates , **: Results from Kadlec et al. (2016).

CWRNN and CWLSTM encoders reach worse accuracy than regular GRUs with the HAN-ptr baseline. This supports our previous analysis from Chapter 5 that a fixed periodic bias is detrimental for most NLP tasks.

Based on the best measured accuracy, we get the best ABS-AS model for $u = 0.9$ for a validation accuracy of 75.4% (-0.1% compared to the HAN-ptr) and a test accuracy of 69.3% (-0.6%). Similarly, we get comparable results to the SGRU and SENS-GRU models. At the same time, ABS does not require any hyperparameters for fine-tuning (when not using ranked attention) and consistently yields lower update rates than both SBA models, getting close to the lower bound in the investigated task. Similarly, SBA requires the underlying recurrent models to be adapted, whereas ABS has the large advantage that it can be used with any of the widely used recurrent models like GRUs or LSTMs requiring model-specific adjustments in their implementation.

Most importantly, our results show attention can be a very efficient filter for finding irrelevant words. Compared to the baseline HAN-ptr, we are consistently able to reduce the amount of words encoded in the recurrent representation from 100% down to 0.05% which corresponds to a significant reduction by a factor of 2000. While we would normally expect strong compression rates to simultaneously cause an irrecoverable loss of information due to the reduced number of bits to store the same information, the best ABS networks decrease in accuracy by only 3.2% (CBT-CN) and 0.4% (CBT-NE) compared to the baseline.

10.3.2 Attention and Skip Distributions

To better understand the differences between the proposed models, we analyze their behavior on exemplary test documents from the CBT dataset. In order to provide a compact visual comparison between the models, we convert the underlying text document into a series of symbols. In doing so, we remove any semantic information, but are able to better visualize where and how the models are processing input tokens. We convert each word into a circle and mark the start for each of the 20 sentences with a rectangle. We mark updated words and sentences with black symbols and skipped tokens with gray symbols. Figure 10.6 shows an example visualization, which illustrates how four different networks process the same test document in different ways.

Using SENS-GRU layers for the word and sentence encoders (Figure 10.6a) results in the situation where more words are read than in any of the ABS models. Similar to the SGRU (see Section 9.6), the SENS-GRU is slightly biased towards updating at the start of a sentence and has an increased likelihood of skipping the longer a sentence gets. The ABS models, on the other hand, activate very sparsely over the document and do not seem to show such a bias. In particular, the ABS-AS model Figure 10.6b correctly reads three of the seven times that the answer is mentioned in the text. For two of these instances, it is even able to skip all words in the sentence that are not the answer.

Previously, we have found that skipping on word *and* sentence layers can be detrimental on model performance if the global update rate is very low. We can see this by comparing an ABS-ASA model that was trained to skip on word- and sentence-level (Figure 10.6c) against the same model that was only allowed to skip on word-level (Figure 10.6d). For the first model, we can see that the correct answer is only captured in one of the sentences. However, the increased update sparsity on sentence level leads to the model choosing the wrong sentence, thereby not being able to provide the correct answer. The second model only operates on word-level, processing all of the sentences, and consequently integrating the sentence where the answer was correctly read. Overall, this exemplifies the difficulties that most of the models encounter when skipping many sentences.

Our initial reason to consider the two different variants ABS-AS and ABS-ASA was that skipping based on attention leads to new representations that can recursively lead to new attention values themselves. This is particularly important in order to understand whether mistakes by the skipping layer are propagated to the second attention layer or whether a second application of attention can actually correct for mistakes. While these mistakes can result from ordinary training errors, they can also be caused by the stochastic activation process which does not guarantee

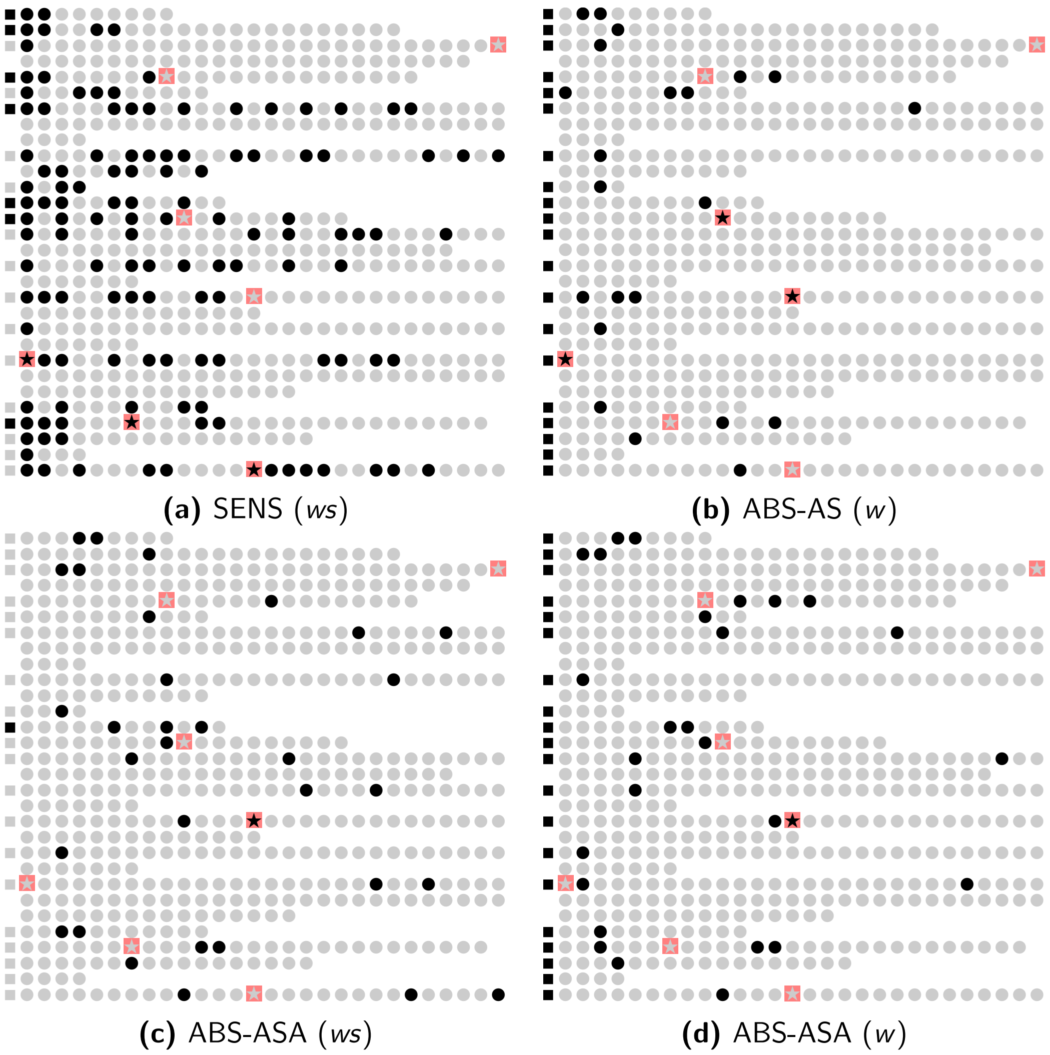


Figure 10.6: The same document is read in different ways by each of the models. Original text (Figure A.9) has been converted into symbols: each circle (●) represents a single word, rectangles (▪) in the first column mark the start of a sentence, and stars (★; highlighted) represent the positions of the correct answer. **Black** symbols denote tokens which triggered a model update in either forward or backward layer, gray symbols indicate skips. *w*: word-level skipping, *ws*: word- and sentence-level skipping.

an update for the layer, even when the first attention layer gives high attention to the currently processed word.

To investigate this, and to see how pre-skip and post-skip attention differ on word level, we consider the example illustrated in Figure 10.7. Here, we visualize timesteps, where the differences between the two attention layers are the largest. In the shown example paragraph, the correct answer (alice) is mentioned in two different sentences. In its first mention (sentence 4), the first attention layer yields almost no attention to the answer, causing the model to skip over it. However,

01 so you see , miss , we 're doing (**0.31→0.27**) our (**0.01→0.27**) best , afore she comes , to -- ' at this moment five , who had been anxiously looking across the garden , called out ` the queen !
02 the queen ! (**0.32→0.29**) ' (**0.03→0.23**)
03 and the three gardeners instantly threw themselves flat upon (**0.25→0.47**) their faces .
04 there was a sound of many footsteps , and (**0.98→0.50**) **alice (0.00→0.50)** looked round , eager to see the queen .
05 first came ten soldiers carrying clubs ; these were all shaped like the three gardeners , (**0.33→0.17**) oblong (**0.01→0.42**) and flat , with their hands and feet at the (**0.38→0.08**) corners : next the ten courtiers ; these were ornamented all over with diamonds , and walked two and two , as the soldiers did .
06 after these came the royal children ; there were ten of them , and the little dears came (**0.00→0.25**) jumping merrily along (**0.79→0.61**) hand in hand , in couples : they were all ornamented with hearts .
07 next came the guests , mostly kings and queens , and among them **alice (0.81→0.85)** recognised the (...)

Figure 10.7: ABS-ASA (attend, skip, attend) example demonstrating the influence of skipping on word attention. Numbers indicate the attention values before and after execution of the skipping layer (*before* → *after*). Skipping can increase (highlighted green) or decrease (highlighted blue) word attention. Updated words are underlined. In the shown example, the initial word-level attention fails to capture the correct answer (**alice**) in sentence 4, but the following attention-skip actually corrects for this mistake, increasing the attention from approximately 0 to 0.5.

this actually leads to a correction by the second attention layer which raises the attention for this word from approximately 0 to 0.5. While this behavior might seem counter-intuitive, it can be explained by the fact that a low update probability from low attention can still eventually lead to an update, causing the model to "peek" at the correct answer. Sentence 4 shows the flip side of this situation as the model does not update when seeing the correct answer despite an update probability of over 80%.

For both cases, it is important to note that all models use the pointer attention mechanism in their final layer to integrate word- and sentence-level attention to produce the network output. Within this setup, each model is able to perform at high accuracies despite possible mistakes from the skipping layer, as long as the attention layers are able to peek at and identify the correct answer. Regardless, our analysis suggests that ABS could benefit from a deterministic decision function, similar to the step function in the SENS-GRU, whereas small levels of stochasticity could still be beneficial to slightly increase variance on the update probabilities.

10.4 Chapter Summary

In this chapter, we have examined how attention can be used to sample update and skip decisions in the context of a question answering task. Compared to surprisal-based activation, we were able to more consistently achieve low update rates, taking only 0.04% of the words into account in some models. We have found a stronger correlation between accuracy and update rate when attention representations are based on skipped words, whereas retaining the original attention yields no correlation. These findings suggest that attention representations capture the most important inputs and are a good basis for model compression.

One advantage of attention-based skipping over other skipping models that we have introduced in this thesis, is that our stochastic variant introduces no additional hyperparameters, i.e. no additional effort is required for tuning the model. While we have introduced ranked attention with the purpose of freely modulating the update rates, we have shown with the ABS-AS model that this additional complexity is not required to consistently achieve very small update rates. Nevertheless, ranked attention offers the possibility to significantly reduce the model's slight losses in tasks, where a specific trade-off between accuracy and sparsity is required. Furthermore, our model requires no modifications to the recurrent layers, as is the case with other approaches presented in this thesis. Instead, attention-based skipping can be used with any of the widely-used models of recurrence and attention.

Our approach has the limitation of not being able to actually save computation time. This is a result of the operations being performed in hindsight, not allowing a prediction of the sparse recurrent representations before the attention scores are known. This is a general downside of neural attention as it is not predictive but based on representations learned in lower network layers. Nevertheless, different to most compression and pruning algorithms that work on pre-trained representations, our model operates online during training and introduces a very small computational overhead of $O(|h|)$. Since our ABS model is capable of skipping over almost all of the inputs without significant losses in accuracy, it can also serve as an empirical baseline for other predictive skipping models as to how many input tokens can be skipped while still solving the task successfully.

More research is needed to determine whether our findings can be adapted to pruning and compression techniques, in order to reduce the network size. It would particularly be interesting to investigate this in the context of networks that dynamically scale their size based on the input such as in Lepikhin et al. (2020). Since recently, transformer-based networks have been outperforming RNNs on most natural language processing tasks, achieving state-of-the-art performance with models like BERT (Devlin et al., 2019). Instead of recurrence, transformers model sequences with self-attention. As our findings show how important the attention representations are in filtering out important input tokens, it gives additional evidence as to why transformer networks might be so successful. Previous work has shown how disabling attention heads can be useful for pruning BERT models (Kovaleva et al., 2019), suggesting more potential to improve the processing pipeline of the original transformer. A natural progression of our work would therefore be to investigate attention-based skipping in transformers.

Overall, our results indicate the usefulness of attention for compressing representations. In the future, it would also be interesting to use attention transfer from a trained network (Zagoruyko and Komodakis, 2017) to another network that can use these attention maps to decide which inputs should be skipped. Such a dualnetwork setup would also allow the second network to actually reduce computation time as long as the savings from the skipping network are larger than the overhead from introducing an attention network. In general, balancing this trade-off is an open question and the main difficulty in designing network models that can actually operate more efficiently in real-world applications.

Part V

Closing

Chapter 11

Conclusion

11.1 Thesis Summary

In this thesis, we have examined three different constraints for skipping state updates in recurrent networks: periodicity, surprisal, and attention. We have started by providing a comparison between more traditional recurrent models with leaky models and the Clockwork RNN as a representative for newer models utilizing conditional computation. Extending this model with LSTM gates, we have evaluated the roles of the Clockwork model's core mechanisms. In aiming to improve the methodology, we have found that its periodic bias is too rigid and that introducing adaptivity poses additional challenges.

Aiming to improve this design in the second part of this thesis, we have set out to find approaches with less rigid assumptions on update timings and what constitutes important and redundant updates. This has led us to propose surprisal-based activation (SBA) which skips updates if the module's information gains are considered too low. Our evaluation with language modeling and other prediction tasks has shown that SBA presents a viable approach that can achieve high skip rates while sometimes even improving baseline accuracy.

In the third part of this thesis, we have integrated SBA in a hierarchical network for question answering, investigating its relationship with attention. We have found that we can lower the model's update rates by utilizing the module dynamics within a majority-voting system, leading to a single update decision per layer on each input token. Finally, we were able to demonstrate that attention itself is an even stronger filter than our previous two methodologies. Exploiting structural biases from the task, attention-based skipping is able to almost exclusively focus on words that are relevant for the classification task, ignoring the rest. Furthermore, this approach allows us to be more flexible than related approaches since we introduce no additional constraints and hyperparameters, and can use it with most state-of-the-art recurrent networks without additional modifications.

11.2 Discussion

The research presented in this thesis aimed to address the research challenge of *modeling skipping conditions in recurrent neural networks to learn more efficient representations*. We have approached this goal by investigating three different methodologies to define update timing: periodicity, surprisal, and attention. Our overall investigation was guided by two main research questions, which we will address and discuss in the following.

Which constraints facilitate effective skipping?

For this question, we have focused on two types of design principles: constraints on the skip condition and structural constraints in the network architecture and connectivity. Generally, these constraints go hand in hand with the introduction of additional assumptions that act as inductive biases in the model.

Concerning the skip constraints, we have found that strong inductive biases can lead to network designs that limit the applicability of the models to certain tasks. This is most clearly visible with periodic activations, which work best when there is a periodic bias in the underlying sequence data that can be exploited. While our SBA methodology works with fewer assumptions, we have initially introduced more hyperparameters to both control and analyze the skipping behavior. In later revisions (Chapter 9), we were able to further simplify the model, showing that surprisal can be effective in determining update timing. Attention-based Skipping (ABS) is the least constrained of these models as its only assumption is that the underlying attention mechanism is capable of filtering out the most important words. As attention provides a very effective mechanism for salience, this approach has led us to achieve the lowest update rates out of any of the investigated models.

Regarding constraints on network structures, we have primarily focused on modular and hierarchical designs. Throughout our findings, a common theme is that modularity through the grouping of units strongly facilitates training and leads to better model performance. One explanation for this consistent observation is that independent decision-making between groups introduces the capability to correct for mistakes. With skipping in particular, bad module decisions can quickly lead to irrecoverable loss of information that has a global effect. Localizing these decisions allows some level of divergence between modules to compensate for such mistakes. Network modularity was most strongly enforced in the Clockwork models (Chapter 4, Chapter 5) and the SENS-GRU (Chapter 9). Our evaluations on these architectures support the alternative hypothesis that modularity increases diversity between modules. While we have seen signs that this can lead to a specialization of units on specific attributes, additional constraints would likely be necessary to achieve this outcome more consistently and in a manner that is easier to interpret. On the other hand, ABS demonstrates that similar, sometimes even better, results can be achieved without modularity (Chapter 10). It is, however, important to note that ABS is not devoid of structural constraints. Instead, it relies heavily on the hierarchical network structure, in particular the pointer attention. Our evaluation of the Hierarchical Attention Network (HAN) in Chapter 8 demonstrates that the mere addition of hierarchical representations does not guarantee improved model performance but that the hierarchically built document representation plays a critical role. As the pointer attention exploits the fact that the correct answer is contained within the document, we can argue that this bias plays an important role in the approach's success. In a way, we can therefore conclude that we have primarily exchanged one constraint with another, allowing the attention mechanism to make skip decisions without any additional constraints outside of the assumption that we can point to the correct answer and build the respective representation in a hierarchical fashion. Ultimately, we find that carefully designed constraints are paramount for successful skipping, while the main challenge lies in finding the right balance in assumptions that are neither too strict nor too loose.

How can we minimize, or even avoid, any trade-offs between model performance and skipping?

Both of our novel SBA and ABS approaches stand in contrast to related conditional computation approaches. They do not lose accuracy despite high skipping rates, in many cases even improving on the baselines. Therefore, our investigation confirms that avoiding the trade-off between model performance and skipping is possible. A key factor for this seems to be to focus less on traditional compression methods, but to figure out effective models of salience (such as surprisal and attention) that help the network to filter out redundancies naturally during backpropagation. In particular, we have found global regularization penalties (Chapter 7) to be too ineffective when used with modular structures. Instead, we have found a more successful approach in incorporating models of salience into the state update functions itself, which seems to harmonize better with the training procedure. These observations lead to the general conclusion that trade-offs can be improved if skip conditions are designed to improve the learned representations themselves.

Overall, our model design utilizes the foundations of gated networks such as the LSTM and the GRU. Our respective ablation studies (Chapter 5, Chapter 6) illustrate that there is a large benefit in considering the role of the gates when modeling skipping with state update functions. Depending on which type of information is skipped with which gate, results can vary strongly. We have found evidence that this can even extend to regularization methods such as zoneout, where our investigation in Chapter 7 has revealed that the original algorithm can be improved by changing the skip targets to different gates. In some of these approaches, information can be maintained passively during skipping, allowing additional opportunities to correct for critical mistakes.

11.3 Future Work

The main motivation for our work has been to learn sparsely updated, efficient representations. As such, our SBA and ABS models operate in hindsight and require the regular computation of state updates in order to analyze which states can be dropped. Consequently, future work should try to use predictive skipping methods to prevent these computations altogether. While some proposals have been made for this problem (Subsection 3.5.5), saving computational cost with conditional computation remains a very difficult problem in practice. In order to prune computational graphs, both the underlying hardware and software framework need to support dynamic differentiation and allocation of resources during execution time. Currently, popular automatic differentiation frameworks offer very limited support for sparse tensors, leading to no actual cost savings with matrix multiplications. On the other hand, we can operate on the full computation graph but reduce energy consumption during inferencing by skipping computations partially but effectively.

While hindsight mechanisms prevent actual pruning of computational graphs, SBA and ABS provide both practical and analytical tools to show the existence of networks in which the update rates can be reduced by large amounts without degrading accuracy. This emphasizes the importance of additional research on instruments that can help identify minimal solutions for various tasks. This stands in contrast to most deep learning approaches in which over-parameterization is considered a feature (Belkin et al., 2019). However, recent research suggests that most neural systems are significantly more complex than the intrinsic dimensionality of commonly benchmarked tasks (Li et al., 2018). In addition, recent trends suggest increasingly lower gains from merely scaling up models along the axes of data and parameter size, leading to additional difficulties of increased energy consumption, training costs, hardware requirements for deployment, complex cycles for development and quality assurance, as well as decreased interpretability (Section 3.4). Consequently, we argue that, instead of training over-parameterized models that are compressed after already spending large amounts of resources on training, they should be trained more efficiently from the beginning.

We believe that efficient skip mechanisms and representations that simultaneously improve model performance benefit heavily from models of salience to identify important features. In this thesis, we have utilized surprisal and attention to model salience. In the future, it would be useful to extend SBA with conditional surprisal to consider more context for skipping. Similarly, further research is necessary to evaluate ABS with different types of attention. Especially Transformers, which model sequences entirely with attention, could benefit from our findings. Consequently, our results extend beyond traditional recurrent models and should be investigated closer with newer sequential models to provide more efficient processing.

Throughout the thesis, we have primarily focused on NLP applications such as language modeling and question answering. As discussed in Section 3.6, there are many more applications that stand to profit from conditional computation. Above all, the integration of different modalities with varying sampling rates is a challenging and unsolved problem. In typical video applications, audio channels can have 16.000 samples for one second of data, whereas image channels only provide 30 frames. A synchronization of channels is most often achieved by down-sampling, averaging, or using feature sets such as MFCCs. Conditional computation provides a potential framework to efficiently operate on the raw data, picking important samples adaptively without relying on these techniques. Similarly, additional research into event-based detection could help improve models in other tasks where long sequences need to be down-sampled or filtered for adaptive processing.

11.4 Conclusion

In conclusion, this thesis contributes to the knowledge about the modeling of effective skip mechanisms for recurrent neural networks. Modularity and hierarchy provide useful structural constraints to separate temporal dynamics and allow specialization of units and error-correction. Models of salience can provide valuable assets to formulate conditional updates, which learn to separate important inputs from redundancy and noise. We have investigated both surprisal and attention as models of salience and found that they give better results than a periodic bias that has shown itself to be too rigid. Our models demonstrate that sparse processing and compressed representations do not always lead to worse model performance. While the practical saving of computational costs remains an open challenge, we have provided approaches that can serve as a basis for future development on models for efficient processing in sequence learning models.

Appendices

Appendix A

Additional Figures

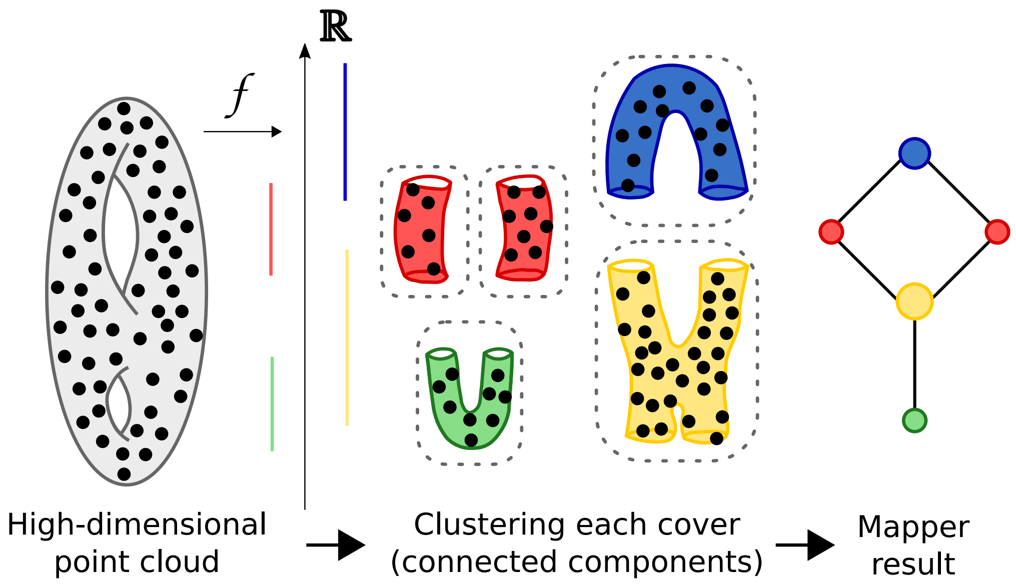
A.1 Topological Maps

In order to project the learning surface of the networks from Subsection 4.4.3 to an even more compact representation, we follow the procedure outlined by Gabella et al. (2020) to build topological maps from the previously computed point clouds using the Mapper Algorithm (Singh et al., 2007) which maps high-dimensional data to a simplicial complex.

We achieve this by clustering 10 principle components (obtained by Principle Component Analysis) with DBSCAN ($\epsilon = 0.2$, min points per cluster: 3) and subsequently map each cluster to a connected graph in two dimensions. The result is a simplicial complex which represents the topological properties of the learned parameter space, particularly which trajectories the weight parameters took during the entire training procedure. As a result of this entire process, we reduce the number of projected nodes down from $|\mathbf{h}| \cdot e = 64 \cdot 50 = 3200$ ($|\mathbf{h}|$ being the number of recurrent units and e the number of trained epochs) to 64 to achieve a compact visualization. For the projection itself we use the KeplerMapper[1]. This procedure is illustrated with an example in Figure A.1. The end result displays connected components as a single node and connects the nodes if they belong to the same point cloud. Two high-dimensional point clouds that are unconnected, result in two *separate* graphs that are also unconnected between each other.

The resulting visualizations are shown in Figure A.2. Both the SRN and the LSTM parameters evolve through similar regions, while the SRN branches out to slightly more regions near the end of the training (yellow nodes). While the LSTM has outlier weights that appear near the end as well, they are further away from the main parameter space of the other weights, making them appear unconnected in the graph. The CWRNN has a more unique graph than the other networks. While there are structural similarities, it is branching out significantly more throughout

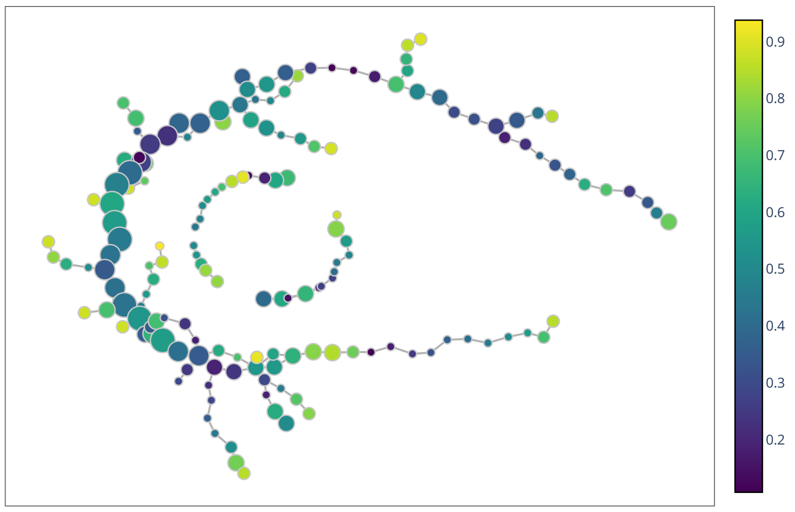
[1] https://kepler-mapper.scikit-tda.org/



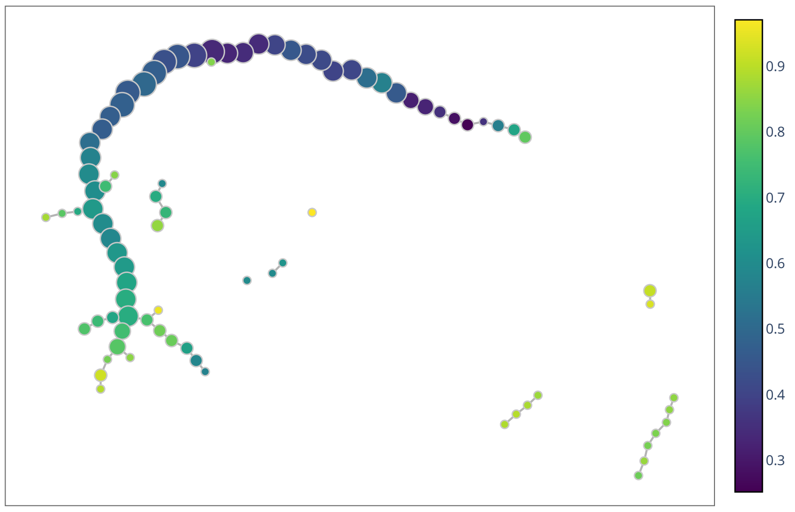
$\mathbb{R}$
f
High-dimensional point cloud
Clustering each cover (connected components)
Mapper result

Figure A.1: Sketch of the Mapper algorithm. High-dimensional point cloud (here: a double torus) is mapped via a filter function f (here: height) to covers that are then clustered in order to map components to nodes in the resulting Mapper graph. Figure adapted from Hoan (2016).

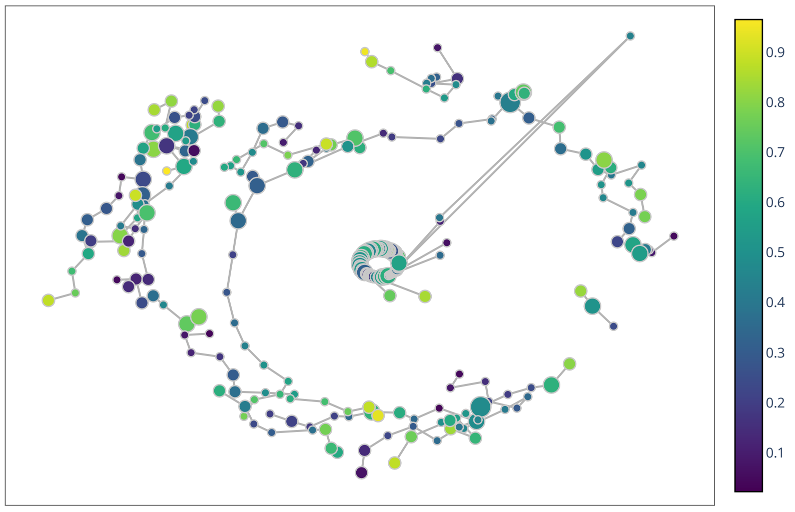
the entire training process, suggesting a larger diversity. In addition, the node colors show how multiple regions of the parameter space are visited both early on as well as late in the training process. This suggests that not all weights converge equally fast, which is unsurprising considering that some CWRNN weight constraints are easier to satisfy than others (in particular those that are updated very infrequently).



(a) SRN



(b) LSTM



(c) CWRNN

Figure A.2: Topologies of weight evolution during training. Color denotes training time (0, purple: start of training. 1, yellow: at the time of convergence). Larger nodes have a larger neighborhood and therefore indicate more training time spent in this region of the parameter space. Unconnected small components indicate weight values that only few parameters visit for a short time frame (typically at the end of the training, indicating attractors around local minima).

A.2 CWRNN Activation Maps During Training

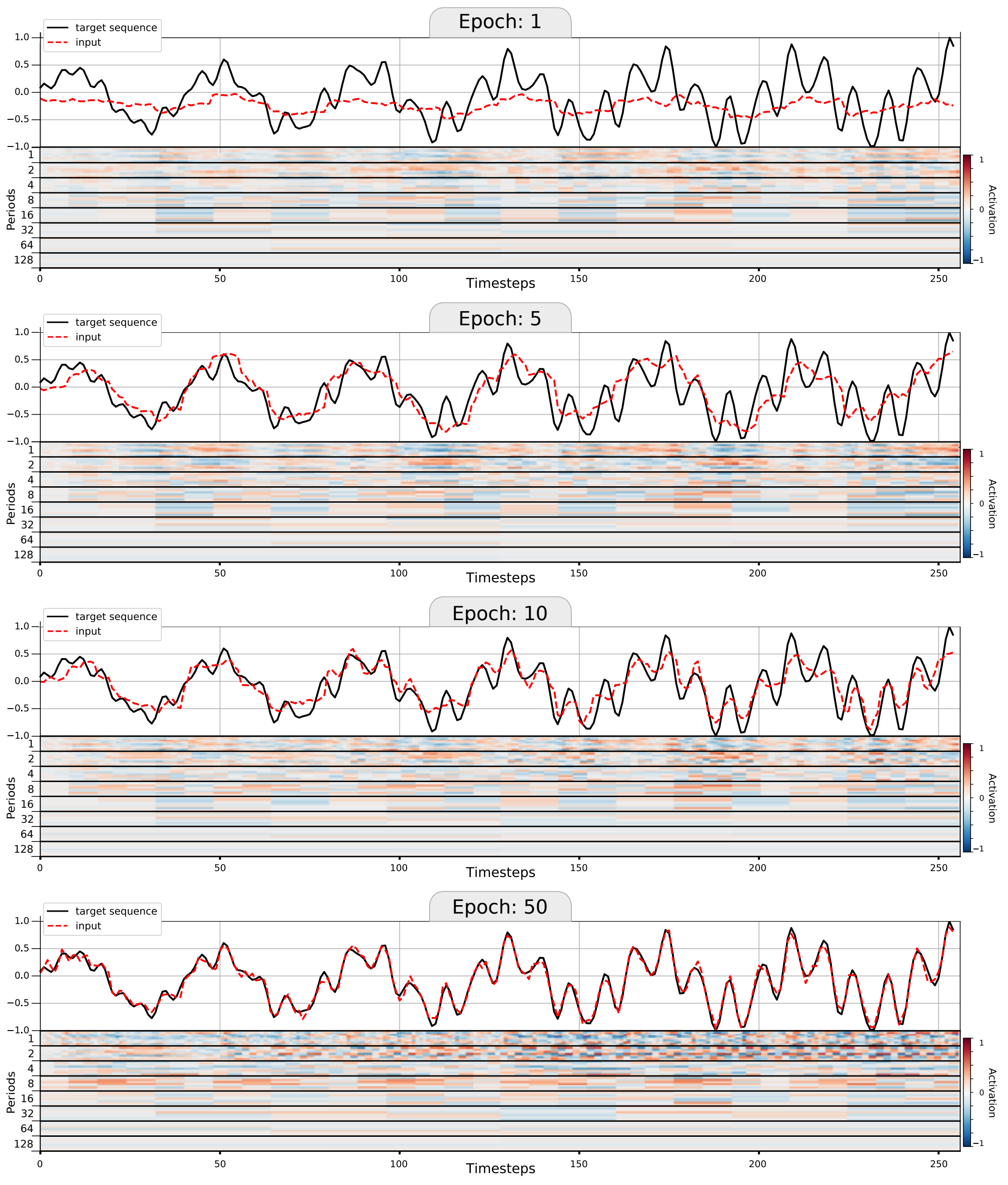


Figure A.3: CWRNN during different training epochs. The network starts learning the dominant frequency first (epochs 1-5). Its representations are formed early on and until epoch 10 in the modules with periods 2,4,8, and partly 16. Over time, only the module with $P = 8$ keeps this representation while the more frequently updating modules start overwriting this with more short-term dependencies.

A.3 Additional Figures for the Question Answering Task

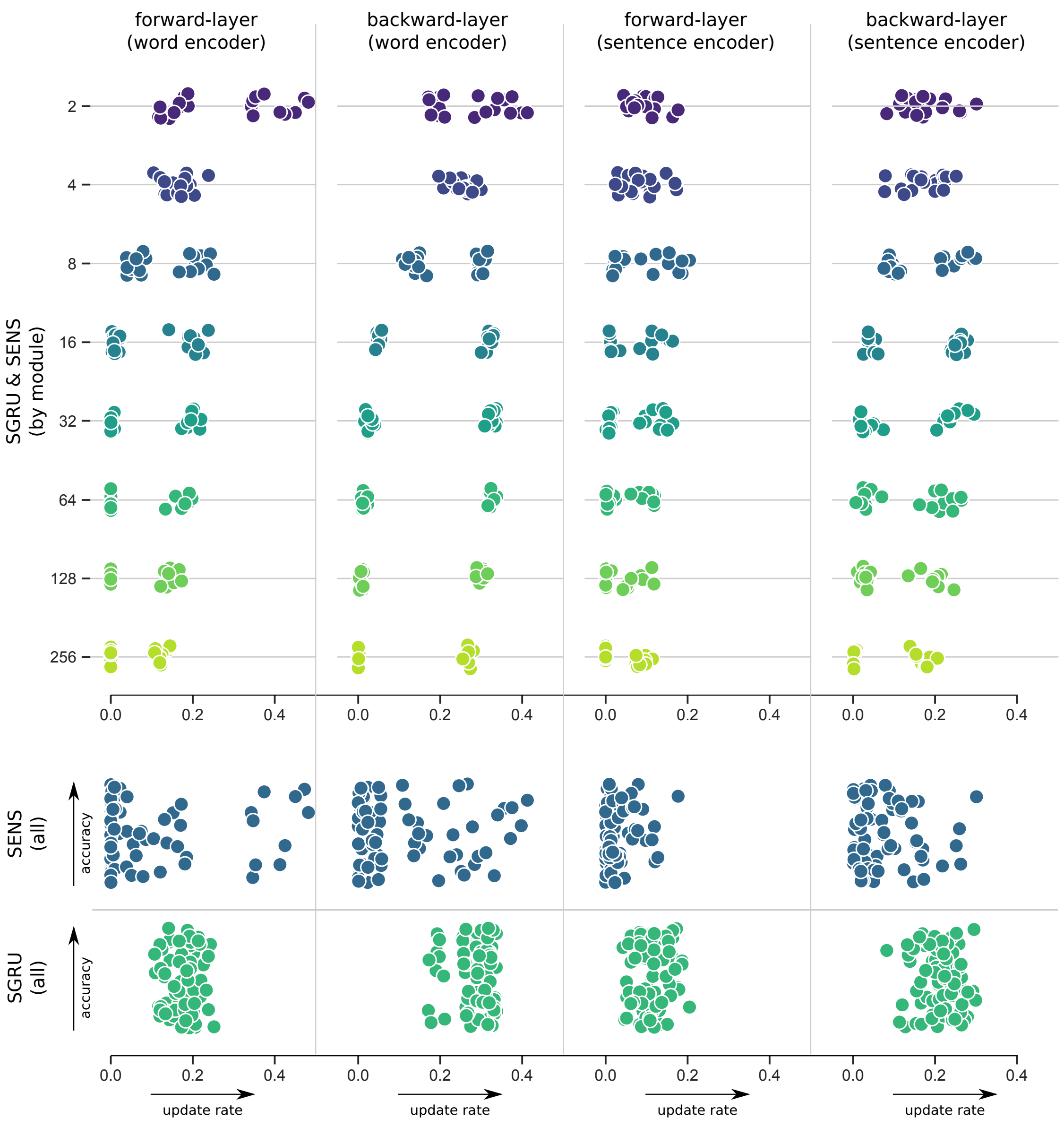


Figure A.4: Comparing the update rates (x-axis) of the forward and backward layers for word-level and sentence-level encoders for the SGRU and SENS-GRU ($\theta = 0.1$) by modules $M \in \{2, \ldots, 256\}$ with the full distribution of the respective models in the two last rows. The figure shows how the update rates of the backward layers increase with M, irrespective of the model and hierarchical layer.

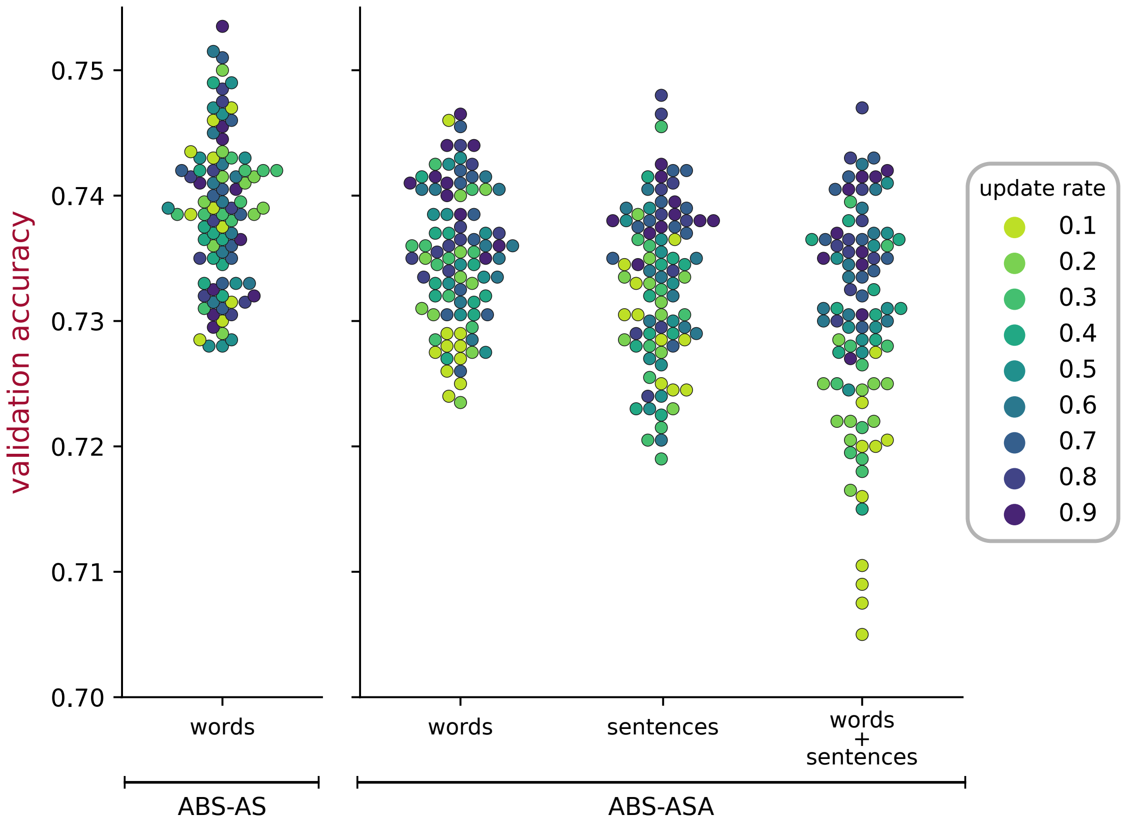


Figure A.5: CBT-CN Results for ABS with rank-based attention mechanism, where the update rate is set as a hyperparameter.

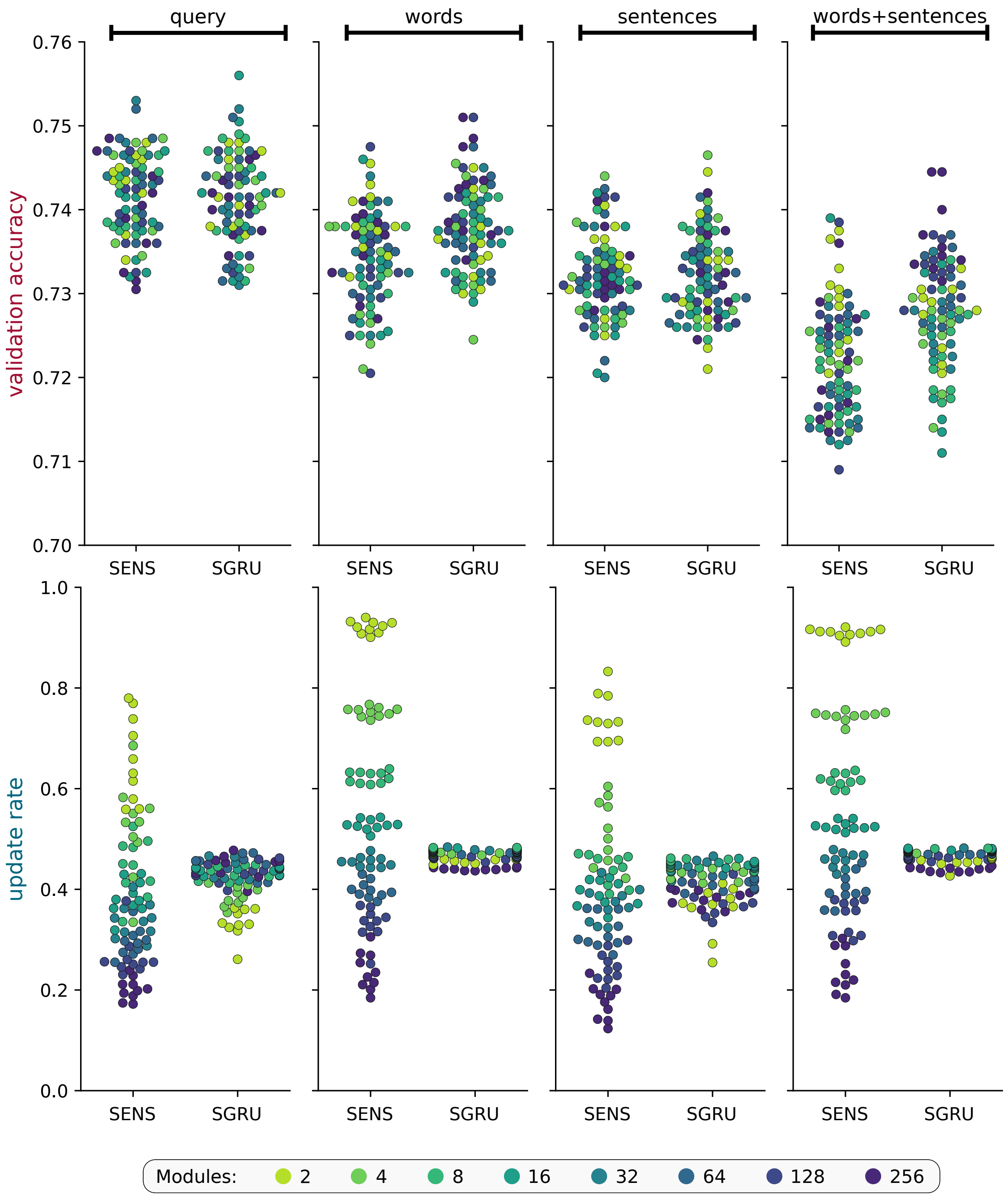


Figure A.6: Comparison between SGRU and SENS-GRU for $\theta = 0.01$ **(CBT-NE)**.

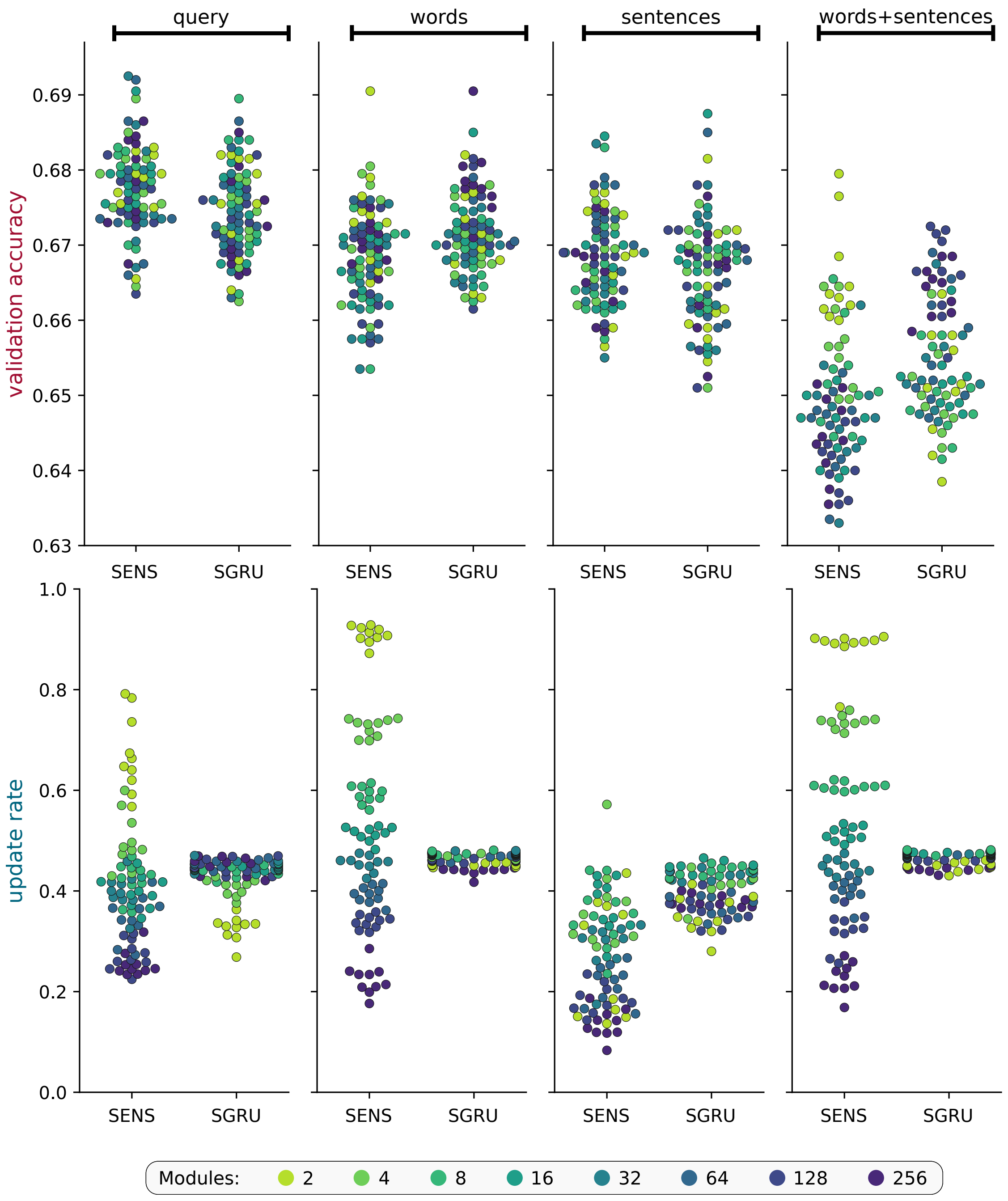


Figure A.7: Comparison between SGRU and SENS-GRU for $\theta = 0.01$ **(CBT-CN)**.

SGRU (module-level skipping):

00 ring grew terribly afraid .
01 ` how do you like them ? '
02 asked snati .
03 ` not well at all , ' said the prince .
04 ` we can do nothing else , ' said snati , ` than attack them , if it is to go well ; you will go against the little one , and i shall take the other . '
05 with this snati leapt at the big one , and was not long in bringing him down .
06 meanwhile the prince went against the other with fear and trembling , and by the time snati came to help him the ox had nearly got him under , but snati was not slow in helping his master to kill it .
07 each of them then began to flay their own ox , but ring was only half through by the time snati had finished his .

SENS-GRU (token-level skipping):

00 ring grew terribly afraid .
01 ` how do you like them ? '
02 asked snati .
03 ` not well at all , ' said the prince .
04 ` we can do nothing else , ' said snati , ` than attack them , if it is to go well ; you will go against the little one , and i shall take the other . '
05 with this snati leapt at the big one , and was not long in bringing him down .
06 meanwhile the prince went against the other with fear and trembling , and by the time snati came to help him the ox had nearly got him under , but snati was not slow in helping his master to kill it .
07 each of them then began to flay their own ox , but ring was only half through by the time snati had finished his .

per-word update rate: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
per-sentence update rate: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

Figure A.8: Example comparing the SGRU (top) and SENS-GRU (bottom) on a paragraph from the test set. Different to the SGRU, the SENS-GRU updates and skips on input-token-level, i.e. it can fully ignore words and sentences. The highlighted values depict the average between forward and backward layers for the bidirectional SGRU and whether one of the two layers updated for the bidirectional SENS-GRU. The shown example illustrates network behavior on paragraphs that do **not** contain the answer.

00 ` yes , it is his business ! '
01 said five , ` and i 'll tell him -- it was for bringing the cook tulip-roots instead of onions . '
02 seven flung down his brush , and had just begun ` well , of all the unjust things -- ' when his eye chanced to fall upon **alice** , as she stood watching them , and he checked himself suddenly : the others looked round also , and all of them bowed low .
03 ` would you tell me , ' said **alice** , a little timidly , ` why you are painting those roses ? '
04 five and seven said nothing , but looked at two .
05 two began in a low voice , ` why the fact is , you see , miss , this here ought to have been a red rose-tree , and we put a white one in by mistake ; and if the queen was to find it out , we should all have our heads cut off , you know .
06 so you see , miss , we 're doing our best , afore she comes , to -- ' at this moment five , who had been anxiously looking across the garden , called out ` the queen !
07 the queen ! '
08 and the three gardeners instantly threw themselves flat upon their faces .
09 there was a sound of many footsteps , and **alice** looked round , eager to see the queen .
10 first came ten soldiers carrying clubs ; these were all shaped like the three gardeners , oblong and flat , with their hands and feet at the corners : next the ten courtiers ; these were ornamented all over with diamonds , and walked two and two , as the soldiers did .
11 after these came the royal children ; there were ten of them , and the little dears came jumping merrily along hand in hand , in couples : they were all ornamented with hearts .
12 next came the guests , mostly kings and queens , and among them **alice** recognised the white rabbit : it was talking in a hurried nervous manner , smiling at everything that was said , and went by without noticing her .
13 then followed the knave of hearts , carrying the king 's crown on a crimson velvet cushion ; and , last of all this grand procession , came the king and queen of hearts .
14 **alice** was rather doubtful whether she ought not to lie down on her face like the three gardeners , but she could not remember ever having heard of such a rule at processions ; ` and besides , what would be the use of a procession , ' thought she , ` if people had all to lie down upon their faces , so that they could n't see it ? '
15 so she stood still where she was , and waited .
16 when the procession came opposite to **alice** , they all stopped and looked at her , and the queen said severely ` who is this ? '
17 she said it to the knave of hearts , who only bowed and smiled in reply .
18 ` idiot ! '
19 said the queen , tossing her head impatiently ; and , turning to **alice** , she went on , ` what 's your name , child ? '

Question:
` my name is **xxxxx** , so please your majesty , ' said alice very politely ; but she added , to herself , ` why , they 're only a pack of cards , after all .

Candidates:	Answer:
alice first king knave miss queen queens dears processions tulip-roots	**alice**

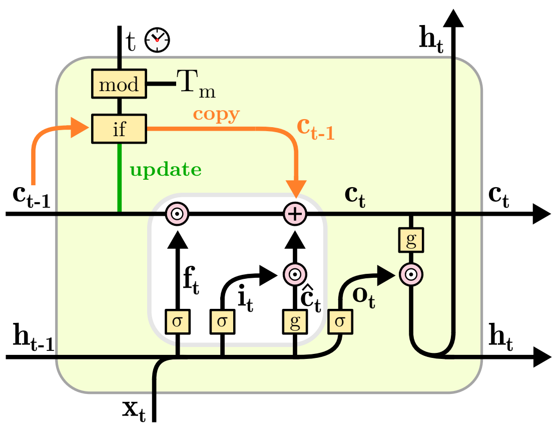
Figure A.9: Original document used for the visualization in Figure 10.6.

Appendix B

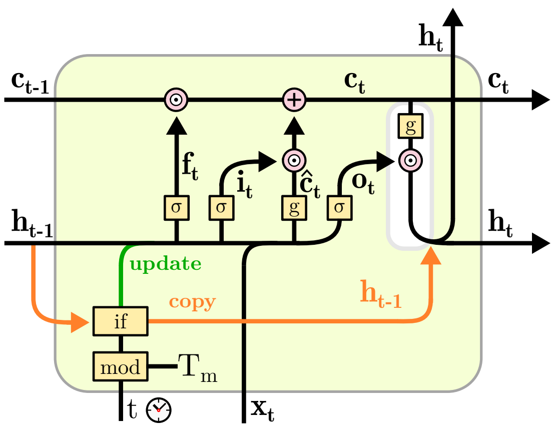
Model Gallery

The following pages provide a brief overview of the seven main models that have been introduced as part of this thesis. We summarize their main ideas and the general motivation behind their design. Additional model variants are described in the respective chapters.

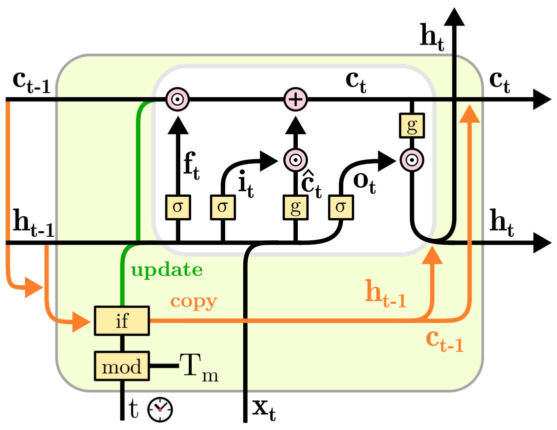
Clockwork LSTM (CWLSTM)



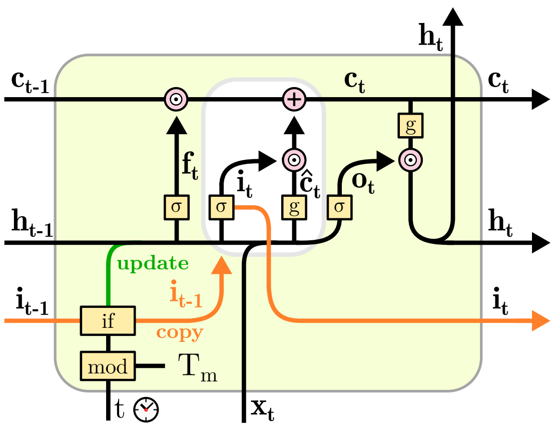
(a) CWLSTM: Skip Target $\mathbf{c}_t$



(b) CWLSTM: Skip Target $\mathbf{h}_t$



(c) CWLSTM: Skip Targets $\mathbf{c}_t$ and $\mathbf{h}_t$



(d) CWLSTM: Skip Target $\mathbf{i}_t$

Figure B.1: Comparison between all 4 CWLSTM variants.

The Clockwork LSTM (CWLSTM) extends the Clockwork RNN (CWRNN) by integrating the original periodic skipping mechanism with the memory gates of the LSTM. Since the LSTM has two states and three memory gates, there are multiple possibilities to achieve this integration. We define and evaluate the four variants displayed above in Chapter 5.

Surprisal-based Activation (SBA)

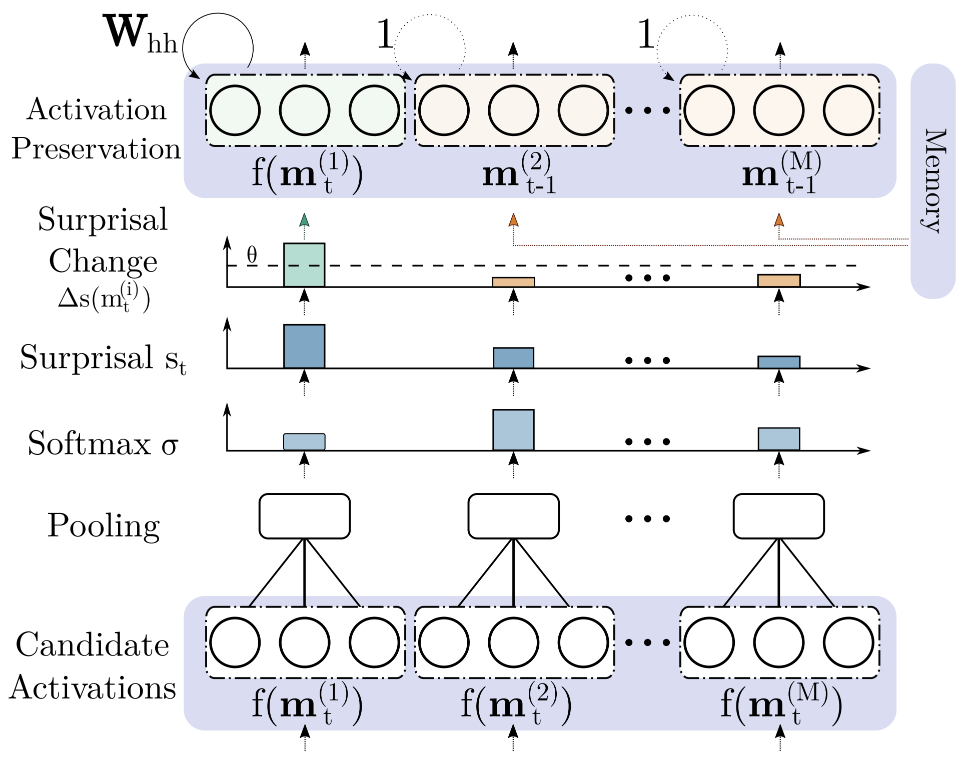


Figure B.2: The SBA model.

Surprisal-based Activation (SBA) is the main architecture proposed in this thesis. The basic idea is to partition a hidden layer into modules that are independently allowed to decide whether they perform an update or skip the current timestep. The decision is formed by observing the change in surprisal of the hidden encoding. The model is introduced in Chapter 6 and further evaluated in Chapter 7 and Chapter 9.

Surprisal-based ENSembling (SENS)

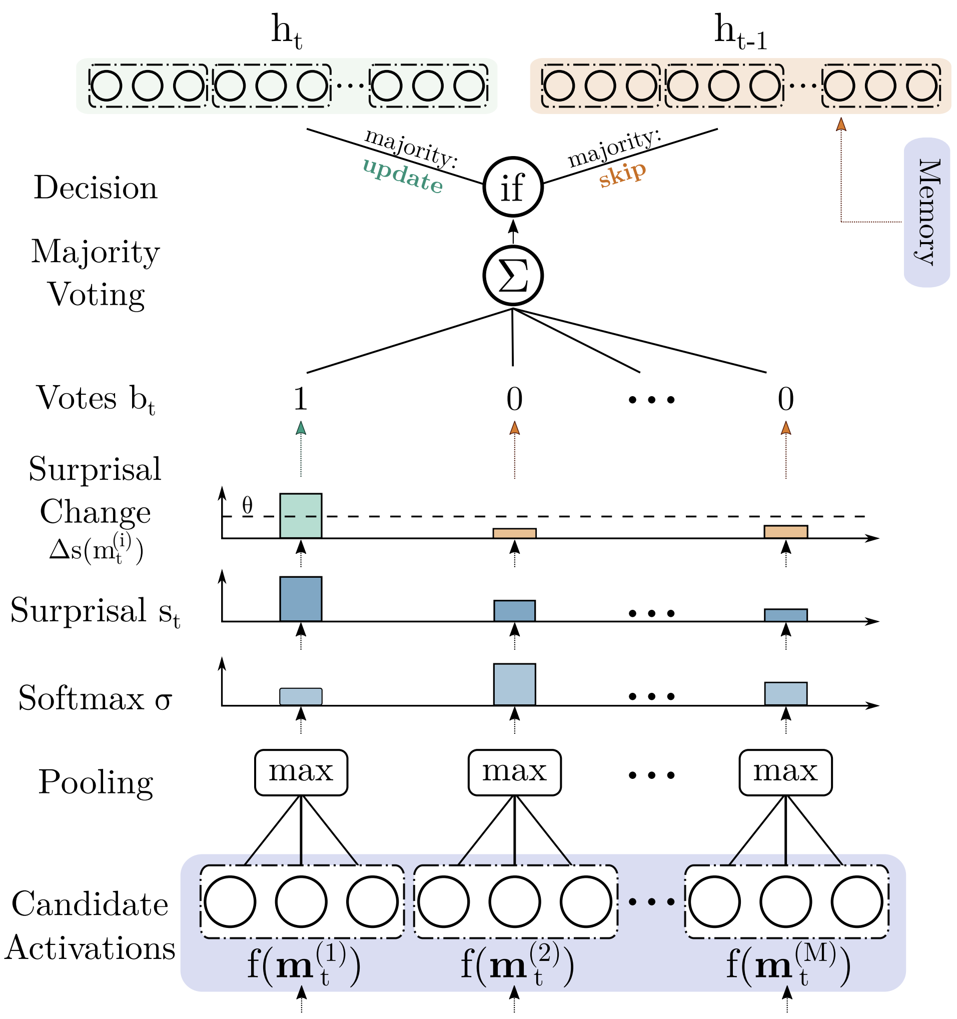


Figure B.3: The SENS model.

Surprisal-based ENSembling (SENS) takes the module-level decision-making process of the SBA and uses it to determine whether the majority of the modules want to update or skip. Based on a majority decision, the model then performs the same action for all modules in the given timestep. Therefore, SENS differs from SBA in that it operates on token-level, as each SENS layer takes a *single* decision for each of its units. The model is discussed in Chapter 9.

HAN-doc-vec

(Document Vector)

As one of the two proposed solutions for hierarchical integration in Chapter 8, the HAN-doc-vec model constructs a document-level representation that is hierarchically based on the sentence- and word-level representations. This final representation is then used to infer similarities to the candidate list in order to find the correct answer to the question/-query.

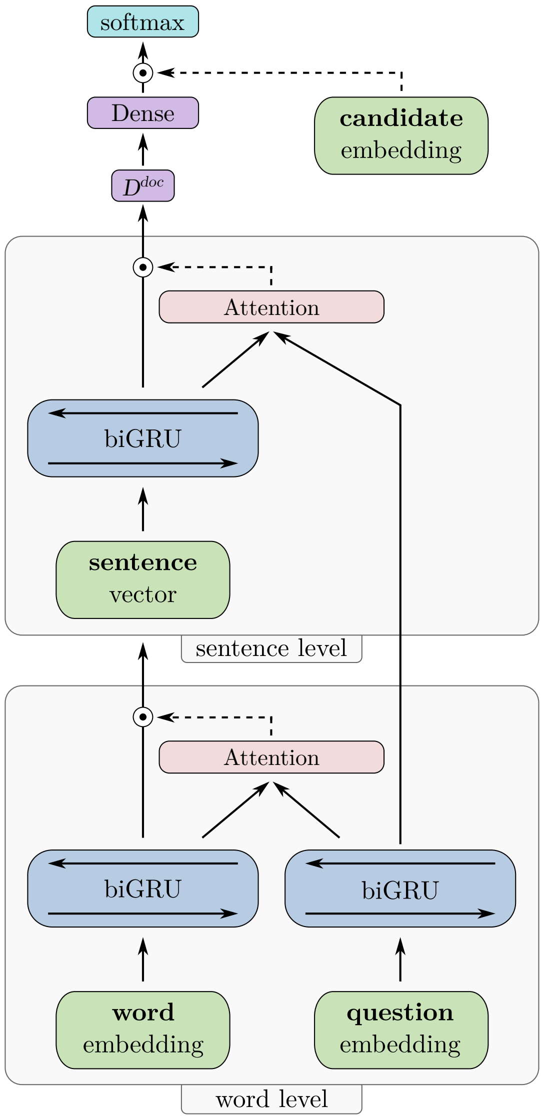


Figure B.4: HAN-doc-vec

HAN-ptr

(Pointer Attention)

The pointer attention mechanism is the other integration mechanism discussed and evaluated in Chapter 8. Instead of integrating the hierarchical representations, it integrates the hierarchical attention by combining word- and sentence-level attention to point to the word that received the most overall attention. Consequently, this mechanism assumes that the correct answer is present in the given document.

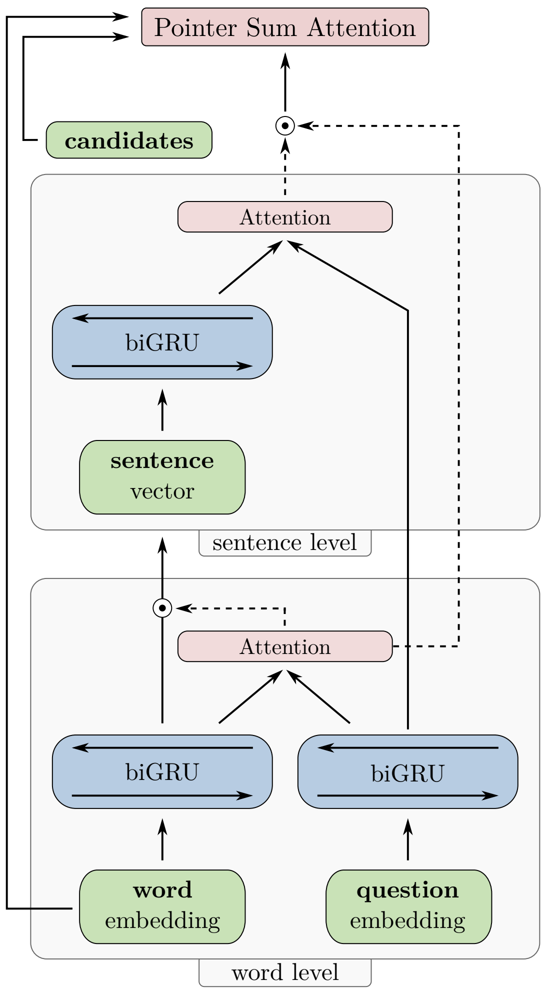


Figure B.5: HAN-ptr

ABS-AS

(Attend and Skip)

The ABS-AS model is one of the two approaches introduced in Chapter 10 to achieve skipping based on surprisal. It is the simpler variant as it adds a single layer in the post-attention stage which masks the previously calculated encoding based on the attention values. Since the sentence level does not build a higher-level representation when using pointer attention, instead forwarding the attention values, this ABS layer can not be used at sentence-level.

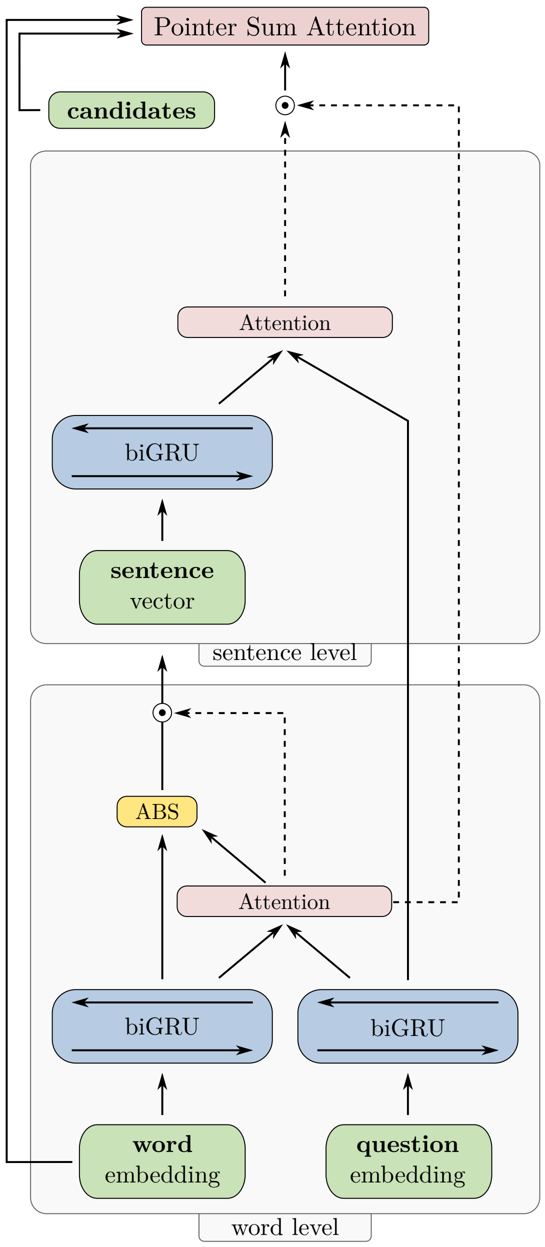


Figure B.6: HAN-ABS-AS-ptr

ABS-ASA

(Attend, Skip, Attend)

The second ABS model discussed in Chapter 10, even more so than the ABS-AS model, focuses more on providing an accurate attention distribution. As such, the attention is recalculated based on the previous skip decisions, allowing the model to incorporate this information in higher layers directly through the attention itself instead of only the word and sentence vectors. Since both word and sentence levels use attention, this allows to perform ABS in both hierarchical levels.

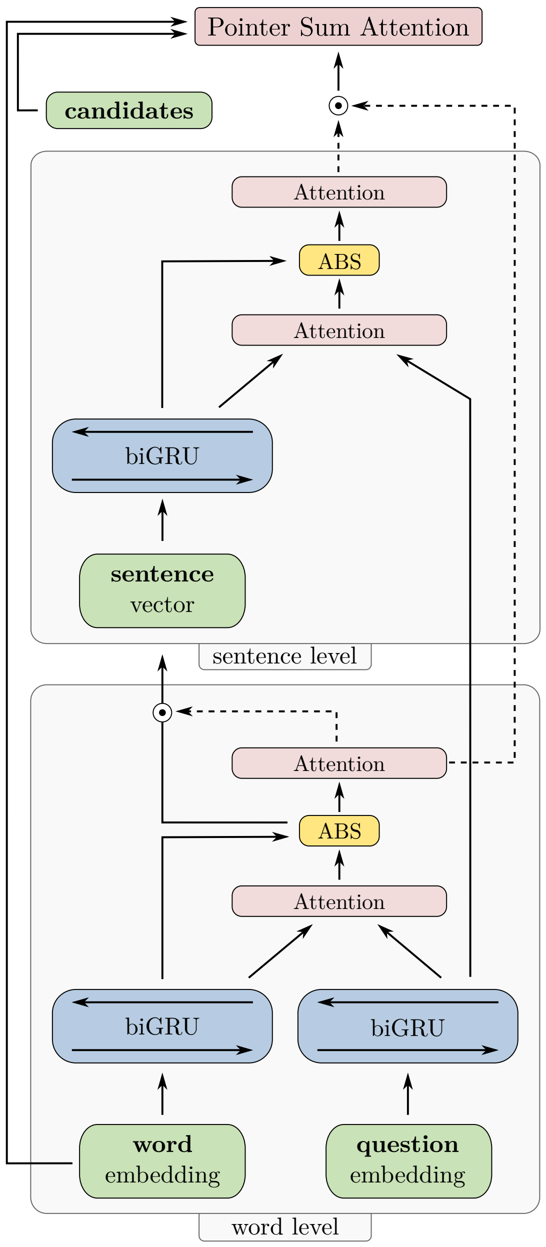


Figure B.7: HAN-ABS-ASA-ptr

Appendix C

Resulting Publications

Journal Articles

- **Alpay, T.**, Abawi, F., Wermter, S. (2019). *Preserving Activations in Recurrent Neural Networks Based on Surprisal.* Neurocomputing, 342, pages 75–82.

Conference Papers

- **Alpay, T.**, Heinrich, S., Wermter, S. *Learning Multiple Timescales in Recurrent Neural Networks.* In Proceedings of the 25th International Conference on Artificial Neural Networks (ICANN 2016), Volume 9886, pages 132–139.
- **Alpay, T.**, Abawi, F., Wermter, S. *Surprisal-Based Activation in Recurrent Neural Networks.* In Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2018), pages 597–602.
- **Alpay, T.**, Heinrich, S., Nelskamp, M., Wermter, S. *Question Answering with Hierarchical Attention Networks.* In International Joint Conference on Neural Networks (IJCNN 2019), Budapest, Hungary, 2019, pp. 1–8.

Publications not included in this thesis

On Neural Networks

- Tietz, M., **Alpay, T.**, Twiefel, J., Wermter, S. *Semi-Supervised Phoneme Recognition with Recurrent Ladder Networks.* In Proceedings of the 26th International Conference on Artificial Neural Networks (ICANN 2017), pages 3–10.
- Heinrich, S., **Alpay, T.**, Wermter, S. *Adaptive and Variational Continuous Time Recurrent Neural Networks.* In IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob 2018), pages 13–18.
- Eppe, M., **Alpay, T.**, Wermter, S. *Towards End-to-End Raw Audio Music Synthesis.* In Proceedings of the 27th International Conference on Artificial Neural Networks (ICANN 2018), pages 137–146.
- Eppe, M., **Alpay, T.**, Abawi, F., Wermter, S. *An Analysis of Subtask-Dependency in Robot Command Interpretation with Dilated CNNs.* In Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2018), pages 25–30.

On Other Topics

- Griffiths, S., **Alpay, T.**, Sutherland, A., Kerzel, M., Eppe, M., Strahl, E., Wermter, S. *Exercise with Social Robots: Companion or Coach?* Workshop on Personal Robots for Exercising and Coaching at the HRI 2018 (HRI 2018).
- Mohammadi, H.B., Xirakia, N., Abawi, F., Barykina, I., Chandran, K., Nair, G., Nguyen, C., Speck, D., **Alpay, T.**, Griffiths, S., Heinrich, S., Strahl, E., Weber, C., Wermter, S. (2019, May). *Designing a Personality-Driven Robot for a Human-Robot Interaction Scenario.* In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA 2019), pages 4317–4324.
- Ali, H., Bhansali, S., Köksal, I., Möller, M., Pekarek-Rosin, T., Sharma, S., Thebille, A.-K., Tobergte, J., Hübner, S., Logacjov, A., Özdemir, O., Parra, R. J., Sanchez, M., Surendrakumar, N. S., **Alpay, T.**, Griffiths, S., Heinrich, S., Strahl, E., Weber, C., Wermter, S. *Virtual or Physical? Social Robots Teaching a Fictional Language Through a Role-Playing Game Inspired by Game of Thrones.* In International Conference on Social Robotics 2019 (ICSR 2019), pages 358–367.

- Arts, E., Zörner, S., Bhatia, K., Mir, G., Schmalzl, F., Srivastava, A., Vasiljevic, B., **Alpay, T.**, Peters, A., Strahl, E. and Wermter, S. *Exploring Human-Robot Trust Through the Investment Game: An Immersive Space Mission Scenario.* In Proceedings of the 8th International Conference on Human-Agent Interaction (HAI 2020), pages. 121–130.

Appendix D

Acknowledgements

I want to thank my advisor Stefan Wermter, reviewer Chris Biemann, and thesis committee chair Frank Steinicke for their feedback, advice, and support.

I'm very grateful to all my former colleagues at WTM who kept this challenging experience fun and entertaining. In particular, I'd like to thank Doreen Jirak, Stefan Heinrich, and Sven Magg for their helpful remarks on my thesis and for always being there when I needed their advice. I also thank Tobias Hinz, Marian Tietz, Alexander Sutherland, German Parisi, Sebastian Starke, and my former research and teaching colleagues for all the fruitful discussions and the many moments of fun.

As with any large organization, the biggest heroes are those operating in the background, often getting the least recognition. I want to thank Reinhard Zierke and Dagmar Schacht for their constant dedication to improving everyone's experience on the campus, Katja Kösters and Erik Strahl for their support at WTM, and Larissa Gebken and Daniel Moldt for caring about a positive academic environment.

I'm also thankful to all my former students for laughing at (most of) my bad jokes and making teaching a very demanding but also extremely rewarding experience. It makes me proud to see so many of you succeed by graduating and building your careers around your passions.

There are countless others who I could always count on. In particular, I want to thank my sister Ella for always believing in me. My biggest gratitude goes to Birte, who made me into the person I am today, constantly pushing me forward when I was stuck and showing incredible patience and understanding. We can finally take a vacation without worrying about deadlines!

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX symposium on operating systems design and implementation (OSDI)*, pages 265–283.

Abati, D., Tomczak, J., Blankevoort, T., Calderara, S., Cucchiara, R., and Bejnordi, B. E. (2020). Conditional channel gated networks for task-aware continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3931–3940.

Ahmed, K. and Torresani, L. (2019). Star-caps: Capsule networks with straight-through attentive routing. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 9101–9110. Curran Associates, Inc.

Alet, F., Lozano-Perez, T., and Kaelbling, L. P. (2018). Modular meta-learning. In Billard, A., Dragan, A., Peters, J., and Morimoto, J., editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 856–868. PMLR.

Alpay, T., Abawi, F., and Wermter, S. (2018). Surprisal-based activation in recurrent neural networks. In *Proceedings of the 26th European Symposium on Artificial Neural Networks (ESANN)*.

Alpay, T., Abawi, F., and Wermter, S. (2019). Preserving activations in recurrent neural networks based on surprisal. *Neurocomputing*, 342:75–82.

Alpay, T., Heinrich, S., Nelskamp, M., and Wermter, S. (2019). Question Answering with Hierarchical Attention Networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Alpay, T., Heinrich, S., and Wermter, S. (2016). Learning Multiple Timescales in Recurrent Neural Networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 132–139. Springer International Publishing.

Alsahli, F. and Mirzal, A. (2020). Development of Hierarchical Attention Network Based Architecture for Cloze-Style Question Answering. In *Emerging Technologies in Computing*, pages 196–213. Springer International Publishing.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48.

Arevian, G. and Panchev, C. (2007). Robust Text Classification Using a Hysteresis-Driven Extended SRN. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 425–434. Springer Berlin Heidelberg.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1120–1128, New York, New York, USA. PMLR.

Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmässan, Stockholm Sweden. PMLR.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. In *Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4945–4949.

Bak, P., Tang, C., and Wiesenfeld, K. (1988). Self-organized criticality. *Physical review A*, 38(1):364.

Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Becker, S. and Hinton, G. E. (1992). Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*.

Bell, T. (2001). Extensive reading: Speed and comprehension. *The reading matrix*, 1(1).

Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2015). Conditional Computation in Neural Networks for faster models. *arXiv preprint arXiv:1511.06297*.

Bengio, Y. (2013). Deep Learning of Representations: Looking Forward. Technical report, Department of Computer Science and Operations Research, Université de Montréal, Canada.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013a). Advances in optimizing recurrent networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8624–8628.

Bengio, Y., Deleu, T., Rahaman, N., Ke, N. R., Lachapelle, S., Bilaniuk, O., Goyal, A., and Pal, C. J. (2020). A meta-transfer objective for learning to disentangle causal mechanisms. In *8th International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, April 26-30, 2020*.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, 3(Feb):1137–1155.

Bengio, Y., Léonard, N., and Courville, A. (2013b). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(2):157–166.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4 (3), pages 1–7. Austin, TX.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. (2017). Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference* *on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 527–536, International Convention Centre, Sydney, Australia. PMLR.

Brakel, P. and Bengio, Y. (2017). Learning independent features with adversarial nets for non-linear ICA. *arXiv preprint arXiv:1710.05050*.

Brown, R. G. (1963). *Smoothing, forecasting and prediction of discrete time series*. Englewood Cliffs, NJ: Prentice-Hall.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems, Pre-proceedings*, volume 33. Curran Associates, Inc.

Campos, V., Jou, B., Giró-i-Nieto, X., Torres, J., and Chang, S. (2018). Skip RNN: Learning to skip state updates in recurrent neural networks. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Carta, A., Sperduti, A., and Bacciu, D. (2020). Incremental Training of a Recurrent Neural Network Exploiting a Multi-Scale Dynamic Memory. *arXiv preprint arXiv:2006.16800*.

Castles, A., Rastle, K., and Nation, K. (2018). Ending the reading wars: Reading acquisition from novice to expert. *Psychological Science in the Public Interest*, 19(1):5–51.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., and Bengio, Y. (2016). Hierarchical Memory Networks. *arXiv preprint arXiv:1605.07427*.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. (2017). Dilated recurrent neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 77–87. Curran Associates, Inc.

Chen, A., Stanovsky, G., Singh, S., and Gardner, M. (2019a). Evaluating question answering evaluation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 119–124.

Chen, D., Bolton, J., and Manning, C. D. (2016). A thorough examination of the cnn/daily mail reading comprehension task. In *Association for Computational Linguistics (ACL)*.

Chen, R. T. Q., Li, X., Grosse, R. B., and Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 2610–2620. Curran Associates, Inc.

Chen, S., Wang, W., and Pan, S. J. (2019b). Metaquant: Learning to quantize by learning to penetrate non-differentiable quantization. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 3916–3926. Curran Associates, Inc.

Chen, T., Liu, H., Ma, Z., Shen, Q., Cao, X., and Wang, Y. (2019c). Neural Image Compression via Non-Local Attention Optimization and Improved Context Modeling. *arXiv preprint arXiv:1910.06244*.

Chen, Y., Friesen, A. L., Behbahani, F., Doucet, A., Budden, D., Hoffman, M., and de Freitas, N. (2020). Modular meta-learning with shrinkage. In *Advances in Neural Information Processing Systems*, volume 33.

Cherry, C., Foster, G., Bapna, A., Firat, O., and Macherey, W. (2018). Revisiting character-based neural machine translation with capacity and compression. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4295–4305, Brussels, Belgium. Association for Computational Linguistics.

Choi, E., Hewlett, D., Lacoste, A., Polosukhin, I., Uszkoreit, J., and Berant, J. (2016). Hierarchical Question Answering for Long Documents. *arXiv preprint arXiv:1611.01839*.

Chollet, F. et al. (2015). Keras: Deep learning library for theano and tensorflow. https://keras.io/.

Chung, J., Ahn, S., and Bengio, Y. (2017). Hierarchical multiscale recurrent neural networks. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *NIPS 2014 Deep Learning and Representation Learning Workshop*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 2067–2075, Lille, France. PMLR.

Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences*, 280(1755):20122863.

Correia, G. M., Niculae, V., and Martins, A. F. T. (2019). Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, Hong Kong, China. Association for Computational Linguistics.

Csordás, R., van Steenkiste, S., and Schmidhuber, J. (2020). Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks. *arXiv preprint arXiv:2010.02066*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2978–2988.

Das, A. and Levina, A. (2019). Critical neuronal models with relaxed timescale separation. *Physical Review X*, 9:021062.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. (2019). Universal transformers. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019*.

Déli, E., Tozzi, A., and Peters, J. F. (2017). Relationships between short and fast brain timescales. *Cognitive neurodynamics*, 11(6):539–552.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

D'Mello, A. M. and Gabrieli, J. D. (2018). Cognitive neuroscience of dyslexia. *Language, Speech, and Hearing Services in Schools*, 49(4):798–809.

Do, K. and Tran, T. (2020). Theory and evaluation metrics for learning disentangled representations. In *8th International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, April 26-30, 2020*.

Donhauser, P. W. and Baillet, S. (2020). Two Distinct Neural Timescales for Predictive Speech Processing. *Neuron*, 105(2):385–393.e9.

Du, X., Qu, X., He, Y., and Guo, D. (2018). Single image super-resolution based on multi-scale competitive convolutional neural network. *Sensors*, 18(3):789.

Eastwood, C. and Williams, C. K. I. (2018). A framework for the quantitative evaluation of disentangled representations. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Ehret, B., Henning, C., Cervera, M. R., Meulemans, A., von Oswald, J., and Grewe, B. F. (2020). Continual learning in recurrent neural networks. *arXiv preprint arXiv:1411.0030*.

Elkins, K. and Chun, J. (2020). Can GPT-3 pass a writer's turing test. *Journal of Cultural Analytics*, 2371:4549.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Eppe, M., Alpay, T., Abawi, F., and others (2018). An Analysis of Subtask-Dependency in Robot Command Interpretation with Dilated CNNs. In *European Symposium on Artificial Neural Networks (ESANN)*.

Fan, F., Xiong, J., and Wang, G. (2020). On interpretability of artificial neural networks: A survey. *arXiv preprint arXiv:2001.02522v2*.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1039–1048.

Figurnov, M., Sobolev, A., and Vetrov, D. (2018). Probabilistic adaptive computation time. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 66(6).

Filippova, K., Alfonseca, E., Colmenares, C. A., Kaiser, L., and Vinyals, O. (2015). Sentence Compression by Deletion with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 360–368, Lisbon, Portugal. Association for Computational Linguistics.

Floridi, L. and Chiriatti, M. (2020). GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, pages 1–14.

Fojo, D., Campos, V., and Giró-i-Nieto, X. (2018). Comparing fixed and adaptive computation time for recurrent neural networks. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*.

Fu, T.-J. and Ma, W.-Y. (2018). Speed reading: Learning to read ForBackward via shuttle. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4439–4448, Brussels, Belgium. Association for Computational Linguistics.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130.

Futrell, R. and Levy, R. (2017). Noisy-context surprisal as a human sentence processing cost model. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL): Volume 1,* *Long Papers*, pages 688–698, Valencia, Spain. Association for Computational Linguistics.

Gabella, M., Afambo, N., Ebli, S., and Spreemann, G. (2020). Topology of Learning in Artificial Neural Networks. *arXiv preprint arXiv:1902.08160v4*.

Galassi, A., Lippi, M., and Torroni, P. (2020). Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–18.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

Ghosh-Dastidar, S. and Adeli, H. (2009). Spiking neural networks. *International Journal of Neural Systems*, 19(04):295–308.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *Proceedings of the 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256. JMLR.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA. PMLR.

Goodman, K. S. (1967). Reading: A psycholinguistic guessing game. *Literacy Research and Instruction*, 6(4):126–135.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and LeCun, Y. (2015). Unsupervised learning of spatiotemporally coherent metrics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4086–4093.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2019). Recurrent Independent Mechanisms. *arXiv preprint arXiv:1909.10893*.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850.*

Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983.*

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 369–376, New York, NY, USA. Association for Computing Machinery.

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5):602 – 610.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

Griffiths, S. S., Mora-Mcginity, M., Forth, J., Purver, M., and Wiggins, G. A. (2015). Information-theoretic segmentation of natural language. In Lieto, A., Battaglino, C., Radicioni, D. P., and Sanguinetti, M., editors, *Proceedings of the 3rd International Workshop on Artificial Intelligence and Cognition, Turin, Italy, September 28-29, 2015*, volume 1510 of *CEUR Workshop Proceedings*, pages 54–67.

Grundmann, S., Trabert, D., Fehre, K., Strenger, N., Pier, A., Kaiser, L., Kircher, M., Weller, M., Eckart, S., Schmidt, L. P. H., Trinter, F., Jahnke, T., Schöffler, M. S., and Dörner, R. (2020). Zeptosecond birth time delay in molecular photoionization. *Science*, 370(6514):339–341.

Gui, T., Zhang, Q., Zhao, L., Lin, Y., Peng, M., Gong, J., and Huang, X. (2019). Long short-term memory with dynamic skip connections. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6481–6488.

Hahn, M. and Keller, F. (2016). Modeling human reading with neural attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 85–95, Austin, Texas. Association for Computational Linguistics.

Hall, E. T. (1976). *Beyond culture.* Anchor.

Hall, E. T. (2000). Monochronic and polychronic time. *Intercultural communication: A reader*, 9:280–286.

Hansen, C., Hansen, C., Alstrup, S., Simonsen, J. G., and Lioma, C. (2019). Neural speed reading with structural-jump-LSTM. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019.*

Hartvigsen, T., Sen, C., Kong, X., and Rundensteiner, E. (2020). Learning to Selectively Update State Neurons in Recurrent Networks. In *29th ACM International Conference on Information and Knowledge Management.*

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Heinrich, S., Alpay, T., and Wermter, S. (2018). Adaptive and variational continuous time recurrent neural networks. In *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 13–18.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1693–1701. Curran Associates, Inc.

Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., and Lerchner, A. (2018). Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230.*

Hill, F., Bordes, A., Chopra, S., and Weston, J. (2016). The goldilocks principle: Reading children's books with explicit memory representations. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

Hinton, G. (2013). Neural networks for machine learning. Coursera, video lectures.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning and Representation Learning Workshop.*

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2019). Learning deep representations by mutual information estimation and maximization. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019.*

Hoan, T. Q. (2016). Tutorial of topological data analysis, part III - mapper algorithm. Tutorial.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diplomarbeit, Technische Universität München.*

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In *A field guide to dynamical recurrent neural networks*. IEEE Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hohman, F., Park, H., Robinson, C., and Polo Chau, D. H. (2020). Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1096–1106.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 79(8):2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences (PNAS)*, 81(10):3088–3092.

Hopfield, J. J. and Tank, D. W. (1986). Computing with neural circuits: A model. *Science*, 233(4764):625–633.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 328–339.

Hsu, W.-N., Zhang, Y., and Glass, J. (2017). Unsupervised learning of disentangled and interpretable representations from sequential data. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 1878–1889. Curran Associates, Inc.

Hu, H., Wang, L., and Qi, G.-J. (2019). Learning to adaptively scale recurrent neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3822–3829.

Hua, W., Zhou, Y., De Sa, C. M., Zhang, Z., and Suh, G. E. (2019). Channel gating neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1886–1896. Curran Associates, Inc.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 646–661. Springer.

Huang, T., Shen, G., and Deng, Z.-H. (2019). Leap-LSTM: enhancing long short-term memory for text categorization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5017–5023. AAAI Press.

Huang, Z., Ye, Z., Li, S., and Pan, R. (2017). Length Adaptive Recurrent Model for Text Classification. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1019–1027, New York, NY, USA. ACM.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106.

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. (2020). Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795.

Ince, R. A. A. (2017). Measuring Multivariate Redundant Information with Pointwise Common Change in Surprisal. *Entropy*, 19(7):318.

Ivry, R. B. and Schlerf, J. E. (2008). Dedicated and intrinsic models of time perception. *Trends in cognitive sciences*, 12(7):273–280.

Iyer, L. R., Chua, Y., and Li, H. (2018). Is Neuromorphic MNIST neuromorphic? Analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv preprint arXiv:1807.01013*.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Jaeger, H., Lukosevicius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks: the official journal of the International Neural Network Society*, 20(3):335–352.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Jayaraman, D. and Grauman, K. (2016). Slow and steady feature analysis: Higher order temporal coherence in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3852–3861.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review*, 106(4):620.

Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. (2017). Variable computation in recurrent neural networks. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings.*

Johansen, A. and Socher, R. (2017). Learning when to skim and when to read. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 257–264, Vancouver, Canada. Association for Computational Linguistics.

Jonschkowski, R. and Brock, O. (2015). Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428.

Jordan, M. (1986). Serial order: a parallel distributed processing approach. ICS report 8604. Technical report, San Diego: Institute for Cognitive Science, University of California.

Joubert, S., Beauregard, M., Walter, N., Bourgouin, P., Beaudoin, G., Leroux, J.-M., Karama, S., and Lecours, A. R. (2004). Neural correlates of lexical and sublexical processes in reading. *Brain and language*, 89(1):9–20.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 2342–2350, Lille, France. PMLR.

Jurafsky, D. and Martin, J. H. (2008). Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall.*

Kádár, Á., Côté, M.-A., Chrupała, G., and Alishahi, A. (2018). Revisiting the hierarchical multiscale LSTM. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 3215–3227.

Kadlec, R., Schmid, M., Bajgar, O., and Kleindienst, J. (2016). Text understanding with the attention sum reader network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 908–918, Berlin, Germany. Association for Computational Linguistics.

Karmarkar, U. R. and Buonomano, D. V. (2007). Timing in the absence of clocks: encoding time in neural network states. *Neuron*, 53(3):427–438.

Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., and Brubaker, M. (2019). Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321.*

Ke, N. R., Bilaniuk, O., Goyal, A., Bauer, S., Larochelle, H., Schölkopf, B., Mozer, M. C., Pal, C., and Bengio, Y. (2019). Learning neural causal models from unknown interventions. *arXiv preprint arXiv:1910.01075.*

Ke, N. R., Goyal, A., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. (2018a). Sparse attentive backtracking: Temporal credit assignment through reminding. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 7640–7651. Curran Associates, Inc.

Ke, N. R., Żołna, K., Sordoni, A., Lin, Z., Trischler, A., Bengio, Y., Pineau, J., Charlin, L., and Pal, C. (2018b). Focused hierarchical RNNs for conditional sequence processing. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2554–2563, Stockholmsmässan, Stockholm Sweden. PMLR.

Kim, H. and Mnih, A. (2018). Disentangling by factorising. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2649–2658, Stockholmsmässan, Stockholm Sweden. PMLR.

Kim, J. S. (2008). Research and the reading wars. *Phi Delta Kappan*, 89(5):372–375.

Kim, M., Moirangthem, D. S., and Lee, M. (2016). Towards abstraction from extraction: Multiple timescale gated recurrent unit for summarization. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 70–77.

Kim, S.-S. (1998). Time-delay recurrent neural network for temporal correlations and prediction. *Neurocomputing*, 20(1-3):253–263.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations (ICLR), Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kirsch, L., Kunze, J., and Barber, D. (2018). Modular networks: Learning to decompose neural computation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 2408–2418. Curran Associates, Inc.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Koutník, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork RNN. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning (ICML)*, volume 32 of *Proceedings of Machine Learning Research*, pages 1863–1871, Bejing, China. PMLR.

Kovaleva, O., Romanov, A., Rogers, A., and Rumshisky, A. (2019). Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374, Hong Kong, China. Association for Computational Linguistics.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A. C., and Pal, C. J. (2017). Zoneout: Regularizing RNNs by randomly preserving hidden activations. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Krueger, D. and Memisevic, R. (2016). Regularizing RNNs by stabilizing activations. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Kullback, S. (1997). *Information Theory and Statistics*. Courier Corporation.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Lamb, A., Goyal, A., Słowik, A., Mozer, M., Beaudoin, P., and Bengio, Y. (2020). Neural function modules with sparse arguments: A dynamic approach to integrating information across layers. *arXiv preprint arXiv:2010.08012*.

Leavitt, M. L. and Morcos, A. (2020a). Selectivity considered harmful: evaluating the causal impact of class selectivity in DNNs. *arXiv preprint arXiv:2003.01262*.

Leavitt, M. L. and Morcos, A. (2020b). Towards falsifiable interpretability research. *arXiv preprint arXiv:2010.12016*.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404. Morgan-Kaufmann.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. (2020). GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *arXiv preprint arXiv:2006.16668*.

Li, C. (2020). Openai's GPT-3 language model: A technical overview. `https://lambdalabs.com/blog/demystifying-gpt-3/`. Online; accessed 2020-12-31.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. (2018). Independently recurrent neural network (IndRNN): Building a longer and deeper RNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5457–5466.

Liao, Z. and Carneiro, G. (2015). Competitive Multi-scale Convolution. *arXiv preprint arXiv:1511.05635*.

Liddy, E. D. (2001). Natural language processing. In *Encyclopedia of Library and Information Science*. Marcel Decker Inc., NY, 2nd edition.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Lin, M., Fu, J., and Bengio, Y. (2019). Conditional Computation for Continual Learning. *arXiv preprint arXiv:1906.06635*.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

Linderholm, T. (2002). Predictive inference generation as a function of working memory capacity and causal text constraints. *Discourse processes*, 34(3):259–280.

Liu, H., Chen, T., Guo, P., Shen, Q., Cao, X., Wang, Y., and Ma, Z. (2019a). Non-local Attention Optimized Deep Image Compression. *arXiv preprint arXiv:1904.09757*.

Liu, Y., Li, S., Cao, Y., Lin, C.-Y., Han, D., and Yu, Y. (2008). Understanding and summarizing answers in community-based question answering services. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 497–504.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019b). Rethinking the value of network pruning. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019*.

Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2019). Challenging common assumptions in the unsupervised learning of disentangled representations. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 4114–4124, Long Beach, California, USA. PMLR.

Lu, J., Yang, J., Batra, D., and Parikh, D. (2016). Hierarchical question-image co-attention for visual question answering. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 289–297. Curran Associates, Inc.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Maddison, C. J., Tarlow, D., and Minka, T. (2014). A* sampling. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27, pages 3086–3094. Curran Associates, Inc.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics (ACL): system demonstrations*, pages 55–60. Association for Computational Linguistics.

Mao, Z.-H. and Massaquoi, S. G. (2007). Dynamics of winner-take-all competition in recurrent neural networks with lateral inhibition. *IEEE Transactions on neural networks*, 18(1):55–69.

Markovsky, I. and Usevich, K. (2012). *Low rank approximation*, volume 139. Springer.

Martins, A. and Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1614–1623, New York, New York, USA. PMLR.

McCarley, J. S. (2019). Pruning a BERT-based question answering model. *arXiv preprint arXiv:1910.06360*.

Melis, G., Dyer, C., and Blunsom, P. (2018). On the state of the art of evaluation in neural language models. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Melloni, L., van Leeuwen, S., Alink, A., and Müller, N. G. (2012). Interaction between bottom-up saliency and top-down control: how saliency maps are created in the human brain. *Cerebral cortex*, 22(12):2943–2952.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Mikolov, T. (2012). *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., and Ranzato, M. (2015). Learning longer memory in recurrent neural networks. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc.

Mikolov, T., Sutskever, I., Deoras, A., Le, H.-S., and Kombrink, S. (2012). Subword language modeling with neural networks. Technical report, Faculty of Information Technology, Brno University of Technology.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81.

Mittal, S., Lamb, A., Goyal, A., Voleti, V., Shanahan, M., Lajoie, G., Mozer, M., and Bengio, Y. (2020). Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 6972–6986, Virtual. PMLR.

Mnih, V., Heess, N., Graves, A., and kavukcuoglu, k. (2014). Recurrent models of visual attention. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27, pages 2204–2212. Curran Associates, Inc.

Mochizuki, Y. and Shinomoto, S. (2014). Analog and digital codes in the brain. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 89(2):022705.

Moirangthem, D. S., Son, J., and Lee, M. (2017). Representing compositionality based on multiple timescales gated recurrent neural networks with adaptive temporal hierarchy for character-level language models. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 131–138.

Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

Mozer, M. C. (1992). Induction of multiscale temporal structure. In Moody, J., Hanson, S., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4, pages 275–282. Morgan-Kaufmann.

Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S., and Mitliagkas, I. (2018). A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, abs/1810.08591.

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 3882–3890. Curran Associates, Inc.

Neumann, M., Stenetorp, P., and Riedel, S. (2016). Learning to Reason With Adaptive Computation. In *NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems*.

Neverova, N., Wolf, C., Lacey, G., Fridman, L., Chandra, D., Barbello, B., and Taylor, G. (2016). Learning human identity from motion patterns. *IEEE Access*, 4:1810–1820.

Olah, C. (2015). Understanding LSTM networks. *URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/*. Online; accessed 2020-12-31.

Ororbia, A., Mali, A., Giles, C. L., and Kifer, D. (2020). Continual learning of recurrent neural networks by locally aligning distributed representations. *IEEE Transactions on Neural Networks and Learning Systems*.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, pages 311–318.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2018). Learning independent causal mechanisms. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4036–4044, Stockholmsmässan, Stockholm Sweden. PMLR.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Parisi, G. I., Tani, J., Weber, C., and Wermter, S. (2018). Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in neurorobotics*, 12:78.

Parr, T. and Friston, K. J. (2017). Working memory, attention, and salience in active inference. *Scientific reports*, 7(1):14678.

Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations (ICLR), Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Perfetti, C. A. (1985). *Reading ability*. Oxford University Press.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Press, O. and Wolf, L. (2017). Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL): Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Quax, S. C., D'Asaro, M., and van Gerven, M. A. J. (2020). Adaptive time scales in recurrent neural networks. *Scientific reports*, 10(1):11360.

Rahaman, N., Goyal, A., Gondal, M. W., Wuthrich, M., Bauer, S., Sharma, Y., Bengio, Y., and Schölkopf, B. (2020). S2RMs: Spatially Structured Recurrent Modules. *arXiv preprint arXiv:2007.06533*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Ramus, F. (2003). Developmental dyslexia: specific phonological deficit or general sensorimotor dysfunction? *Current opinion in neurobiology*, 13(2):212–218.

Ravasz, E. and Barabási, A.-L. (2003). Hierarchical organization in complex networks. *Physical review E*, 67(2):026112.

Rayner, K. (1978). Eye movements in reading and information processing. *Psychological bulletin*, 85(3):618.

Rayner, K., Schotter, E. R., Masson, M. E., Potter, M. C., and Treiman, R. (2016). So much to read, so little time: How do we read, and can speed reading help? *Psychological Science in the Public Interest*, 17(1):4–34.

Rayner, K., Slattery, T. J., Drieghe, D., and Liversedge, S. P. (2011). Eye movements and word skipping during reading: Effects of word length and predictability. *Journal of Experimental Psychology: Human Perception and Performance*, 37(2):514.

Razavi, A., van den Oord, A., and Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 14866–14876. Curran Associates, Inc.

Rényi, A. et al. (1961). On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California.

Ridgeway, K. and Mozer, M. C. (2018). Learning deep disentangled embeddings with the f-statistic loss. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 185–194. Curran Associates, Inc.

Rocki, K. M. (2016). Surprisal-Driven Feedback in Recurrent Networks. *arXiv preprint arXiv:1608.06027*.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. (2019). Routing Networks and the Challenges of Modular and Compositional Computation. *arXiv preprint arXiv:1904.12774*.

Rosenbaum, C., Klinger, T., and Riemer, M. (2018). Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *6th International*

Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc., Buffalo NY.

Rudy Setiono and Huan Liu (1997). Neural-network feature selector. *IEEE Transactions on Neural Networks*, 8(3):654–662.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Rumelhart, D. E. and Zipser, D. (1985). Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

Russin, J., Jo, J., O'Reilly, R., and Bengio, Y. (2020). Compositional generalization by factorizing alignment and translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL): Student Research Workshop*, pages 313–327.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC ^{2})*.

Santos, C. D. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, volume 32 of *Proceedings of Machine Learning Research*, pages 1818–1826, Bejing, China. PMLR.

Schmidhuber, J. (1992). Learning factorial codes by predictability minimization. *Neural computation*, 4(6):863–879.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural networks: the official journal of the International Neural Network Society*, 61:85–117.

Schotter, E. R., Tran, R., and Rayner, K. (2014). Don't believe what you read (only once) comprehension is supported by regressions during reading. *Psychological science*, 25(6):1218–1226.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12):54–63.

See, A., Luong, M.-T., and Manning, C. D. (2016). Compression of neural machine translation models via pruning. In *Proceedings of The 20th SIGNLL Conference* *on Computational Natural Language Learning*, pages 291–301, Berlin, Germany. Association for Computational Linguistics.

Seo, M. J., Min, S., Farhadi, A., and Hajishirzi, H. (2018). Neural speed reading via skim-RNN. In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Shelhamer, E., Rakelly, K., Hoffman, J., and Darrell, T. (2016). Clockwork Convnets for Video Semantic Segmentation. In *European Conference on Computer Vision – ECCV 2016 Workshops*, pages 852–868. Springer International Publishing.

Shen, Y., Huang, P.-S., Gao, J., and Chen, W. (2017). Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1047–1055, New York, NY, USA. Association for Computing Machinery.

Shen, Y., Tan, S., Sordoni, A., and Courville, A. C. (2019). Ordered neurons: Integrating tree structures into recurrent neural networks. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019*.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using GPU model parallelism. *arXiv preprint arXiv:1909.08053*.

Siegelmann, H. T. and Sontag, E. D. (1995). On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482.

Singh, G., Memoli, F., and Carlsson, G. (2007). Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. In Botsch, M., Pajarola, R., Chen, B., and Zwicker, M., editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association.

Singh, S., Hoiem, D., and Forsyth, D. (2016). Swapout: Learning an ensemble of deep architectures. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 28–36. Curran Associates, Inc.

Smith, F. (1992). Learning to read: The never-ending debate. *Phi Delta Kappan*, 73(6):432–441.

Socher, R., Lin, C. C.-Y., Ng, A., and Manning, C. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, ICML '11, pages 129–136, New York, NY, USA. ACM.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Sordoni, A., Bachman, P., Trischler, A., and Bengio, Y. (2016). Iterative Alternating Neural Attention for Machine Reading. *arXiv preprint arXiv:1606.02245*.

Speer, R. and Lowry-Duda, J. (2017). Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 85–89.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015a). Training very deep networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2377–2385. Curran Associates, Inc.

Srivastava, R. K., Masci, J., Gomez, F. J., and Schmidhuber, J. (2015b). Understanding locally competitive networks. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. (2013). Compete to compute. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26, pages 2310–2318. Curran Associates, Inc.

Strub, F., de Vries, H., Mary, J., Piot, B., Courville, A., and Pietquin, O. (2017). End-to-end optimization of goal-driven and visually grounded dialogue systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2765–2771.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. (2019). Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 331–335, Florence, Italy. Association for Computational Linguistics.

Sukhbaatar, S., szlam, a., Weston, J., and Fergus, R. (2015). End-to-end memory networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2440–2448. Curran Associates, Inc.

Sutskever, I. and Hinton, G. (2010). Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. MIT Press.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4278–4284. AAAI Press.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Tallec, C. and Ollivier, Y. (2018). Can recurrent neural networks warp time? In *6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.*

Tao, J., Thakker, U., Dasika, G., and Beu, J. (2019). Skipping RNN State Updates without Retraining the Original Model. In *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*, pages 31–36. ACM.

Tavarone, R. and Badino, L. (2018). Conditional-computation-based recurrent neural networks for computationally efficient acoustic modelling. In Yegnanarayana, B., editor, *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 1274–1278. ISCA.

Thompson, N. C., Greenewald, K., Lee, K., and Manso, G. F. (2020). The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*.

Tjandra, A., Sakti, S., and Nakamura, S. (2019). End-to-end feedback loss in speech chain framework via straight-through estimator. In *Proceedings of the 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6281–6285. IEEE.

Tran, K., Bisazza, A., and Monz, C. (2016a). Recurrent memory networks for language modeling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies*, pages 321–331, San Diego, California. Association for Computational Linguistics.

Tran, N.-T., Luong, V.-T., Nguyen, N. L.-T., and Nghiem, M.-Q. (2016b). Effective attention-based neural architectures for sentence compression with bidirectional long short-term memory. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, SoICT '16, pages 123–130, New York, NY, USA. Association for Computing Machinery.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. (2017). Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 2627–2636. Curran Associates, Inc.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. In *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, page 125. ISCA.

van Steenkiste, S., Locatello, F., Schmidhuber, J., and Bachem, O. (2019). Are disentangled representations helpful for abstract visual reasoning? In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 14245–14258. Curran Associates, Inc.

Van Wassenhove, V., Buonomano, D. V., Shimojo, S., and Shams, L. (2008). Distortions of subjective time perception within and across senses. *PlOS ONE*, 3(1):e1437.

Vania, C., Grivas, A., and Lopez, A. (2018). What do character-level models learn about morphology? The case of dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2573–2583, Brussels, Belgium. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700. Curran Associates, Inc.

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424.

Wang, X.-J. (2003). Neural oscillations. In *Encyclopedia of cognitive science*, pages 272–280. Macmillan.

Wermter, S. (1995). *Hybrid connectionist natural language processing*, volume 7 of Chapman & Hall neural computing series. Chapman & Hall London.

Wermter, S., Panchev, C., and Arevian, G. (1999). Hybrid Neural Plausibility Networks for News Agents. *AAAI/IAAI*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: unsupervised learning of invariances. *Neural computation*, 14(4):715–770.

Wu, Q., Teney, D., Wang, P., Shen, C., Dick, A., and van den Hengel, A. (2017). Visual question answering: A survey of methods and datasets. *Computer Vision and Image Understanding*, 163:21–40.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Würtz, E. (2017). Intercultural Communication on Web sites: a Cross-Cultural Analysis of Web sites from High-Context Cultures and Low-Context Cultures. *Journal of Computer-Mediated Communication*, 11(1):274–299.

Xie, X., Hahnloser, R. H., and Seung, H. S. (2002). Selectively grouping neurons in recurrent networks of lateral inhibition. *Neural computation*, 14(11):2627–2646.

Yamamoto, K. and Maeno, K. (2019). PCAS: pruning channels with attention statistics for deep network compression. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, page 138. BMVA Press.

Yamashita, Y. and Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLOS Computational Biology*, 4(11):e1000220.

Yang, B., Bender, G., Le, Q. V., and Ngiam, J. (2019). Condconv: Conditionally parameterized convolutions for efficient inference. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1307–1318. Curran Associates, Inc.

Yang, J., Reed, S. E., Yang, M.-H., and Lee, H. (2015). Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1099–1107. Curran Associates, Inc.

Yang, Z., Dhingra, B., Yuan, Y., Hu, J., Cohen, W. W., and Salakhutdinov, R. (2017). Words or characters? fine-grained gating for reading comprehension. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings.*

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Yin, P., Lyu, J., Zhang, S., Osher, S. J., Qi, Y., and Xin, J. (2019). Understanding straight-through estimator in training activation quantized neural nets. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019.*

Yu, A. W., Lee, H., and Le, Q. (2017a). Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL): Volume 1, Long Papers*, pages 1880–1890, Vancouver, Canada. Association for Computational Linguistics.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. S. (2019). Slimmable neural networks. In *7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019.*

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017b). Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-first AAAI conference on artificial intelligence*.

Yu, Z. and Liu, G. (2018). Sliced recurrent neural networks. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 2953–2964, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Yu, Z., Moirangthem, D. S., and Lee, M. (2017c). Continuous timescale long-short term memory neural network for human intent understanding. *Frontiers in neurorobotics*, 11:42.

Yue, B., Fu, J., and Liang, J. (2018). Residual recurrent neural networks for learning sequential representations. *Information*, 9(3):56.

Yue, Q., Martin, R. C., Fischer-Baum, S., Ramos-Nuñez, A. I., Ye, F., and Deem, M. W. (2017). Brain modularity mediates the relation between task complexity and performance. *Journal of cognitive neuroscience*, 29(9):1532–1546.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). Q8BERT: Quantized 8bit BERT. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC ^{2})*.

Zagoruyko, S. and Komodakis, N. (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Zarcone, A., van Schijndel, M., Vogels, J., and Demberg, V. (2016). Salience and Attention in Surprisal-Based Accounts of Language Processing. *Frontiers in Psychology*, 7:844.

Zaremba, W., Sutskever, I., and Vinyals, O. (2015). Recurrent neural network regularization. In *International Conference on Learning Representations (ICLR)*.

Zelnik-Manor, L. and Irani, M. (2001). Event-based analysis of video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 123, Los Alamitos, CA, USA. IEEE Computer Society.

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., and Bengio, Y. (2016). Architectural complexity measures of recurrent neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 1822–1830. Curran Associates, Inc.

Zhang, T., Huang, M., and Zhao, L. (2018). Learning structured representation for text classification via reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhou, X., Ye, Z., Cheung, H., and Chen, H.-C. (2009). Processing the chinese language: An introduction. *Language and Cognitive Processes*, 24(7-8):929–946.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 4189–4198, International Convention Centre, Sydney, Australia. PMLR.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

`04.01.2021`	`Tayfun Alpay`
....	....
Ort, Datum	Unterschrift