Towards Efficient and Generalizable Natural Language Processing

Hao Peng

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2022

Reading Committee:
Noah A. Smith, Chair
Luke Zettlemoyer
Kevin Jamieson

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Towards Efficient and Generalizable
Natural Language Processing

Hao Peng

Chair of the Supervisory Committee:
Professor Noah A. Smith
Paul G. Allen School of Computer Science and Engineering

Natural language processing (NLP) is having a paradigm shift. Scaling up in terms of the sizes of models and data plays an increasingly important role. Despite the remarkable empirical gain in various well-bounded NLP tasks, the ability to generalize to complex unseen scenarios is still elusive. Further, the growing computational requirements have heightened the barriers to entry to NLP research. The inquiry of this thesis can be broadly divided into two research questions: What algorithms can integrate symbolic structures into NLP models and improve their generalization? Can we build neural architectures that are both efficient and accurate?

We first focus on the synergy between modern deep learning models and the classical linguistic structures. We explore a surrogate gradient method that allows for incorporating discrete structured prediction as intermediate layers in neural nets, facilitating training structured NLP pipelines end-to-end. Further, we augment neural language models with attention modules that can be trained to syntactically inform the representations or used to induce syntactic structures in an unsupervised manner. We experiment with various linguistic structures—syntactic and semantic—and apply them in real-world NLP tasks, including structured prediction, language modeling, and text classification.

In the second part, we aim to improve the efficiency of state-of-the-art deep learning architectures. We discuss two efficient attention models that reduce the overhead of transformers

from quadratic to linear in input lengths. In language modeling, text classification, and machine translation, they substantially improve the efficiency of state-of-the-art models without losing accuracy. Finally, we present a formal analysis of recurrent neural networks and connect them to traditional automaton methods, which offers a flexible way to devise more efficient and interpretable neural architectures imbued with desired inductive biases.

Acknowledgements

First and foremost, I would like to thank my advisor Noah Smith. Noah is a vast repository of knowledge. It is no overstatement to say that without his consistent guidance, tutelage, and support, this thesis would never have existed. Noah has been my role model, and taught me how to teach, advise, and inspire.

My sincere thanks must also go to my committee members, Chris Dyer, Zaid Harchaoui, Kevin Jamieson, and Luke Zettlemoyer. They generously offered me their time and gave valuable feedback on improving this thesis. I want to extend my thanks to: Yejin Choi, who has always been thoughtful and supportive; Tim Althoff, for teaching me how to design a course by example; Elise Dorough, Eric Eto, Jin Xu, and other staff at the University of Washington, for their patience and help when I procrastinated.

I am most grateful to: Yansong Feng, Lili Mou, and Weiwei Sun at Peking University, for whetting my appetite for NLP when I was an undergraduate; Miltiadis Allamanis and Charles Sutton at the University of Edinburgh, and Chew-Yew Lin and Jing Liu at Microsoft Research Asia for introducing me to "cousins" of NLP; Dipanjan Das, Manaal Faruqui, Ankur Parikh at Google Research, for piquing my interest in text generation; Phil Blunsom, Lingpeng Kong, and Dani Yogatama at DeepMind, for the the insightful conversations during an amazing internship.

I was privileged to have the chance to learn from my wonderful collaborators: Iz Beltagy, Chris Brokett, Liqun Chen, Weizhu Chen, Yunchuan Chen, James Cross, Bhuwan Dingra, Jesse Dodge, Bill Dolan, Matt Gardner, Ivy Guo, Gabriel Ilharco, Zhi Jin, Jungo Kasai, Dianqi Li, Ge Li, Yi Mao, Will Merrill, Ivan Montero, Nikolaos Pappas, Matt Peters, Tobias Rohde, Alexis Ross, Roy Schwartz, Ming-Ting Sun, Swabha Swayamdipta, Chenhao Tan, Sam Thomson, Sherry Wu, Zhaofeng Wu, Yan Xu, Lu Zhang, Michael Zhang, and Yizhe Zhang. To ARK, UWNLP, and my

friends, thank you for making this arduous journey more fun. To my parents, words cannot convey my gratitude for your unconditional love and support.

The work reported in this thesis received support from the Defense Advanced Research Projects Agency by grant FA8750-12-2-0342, the National Science Foundation by grants IIS-1562364 and IIS-2113530, a Jeff Dean - Heidi Hopper Endowed Regental Fellowship, and a Google Ph.D. Fellowship.

Most of all, Yue Guo, who lights up my life, is thanked.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to understand natural language is a foundation of artificial intelligence (AI). Recent advances in representation learning have led to a new paradigm for natural language processing (NLP). With computationally-intensive deep learning models, general-purpose representations are pretrained on massive textual data and then finetuned on task-specific datasets. Despite the remarkable progress in almost every aspect of NLP, challenges come along. First, the growing computational requirements have heightened the barriers to entry to state-of-the-art research. For example, training GPT-3 (Brown et al., 2020), a state-of-the-art language model, is estimated to cost 4.6 million USD; and PaLM (Chowdhery et al., 2022), another, 10 million. Exploring these large models is hardly affordable to less-resourced research groups. Second, just like their predecessors, these models often fail to generalize to complex unseen settings. This dissertation presents a series of algorithms towards addressing these challenges, and aims to build more efficient and generalizable representation learners for NLP.

In Part I, we visit an estranged old friend of NLP—structures. Language is structured: meanings are articulated and perceived through hierarchical compositions of words. In the past, they played a central role. For example, through an NLP pipeline, a sentence is first tagged with parts of speech, then parsed into a syntactic tree, then semantically analyzed, before being fed into a question-answering system. Linguistic structures provide valuable inductive biases that help models generalize: as we show in Wu et al. (2021), explicitly encoding semantic graphs enhances the sample efficiency and robustness of pretrained language models. Why has the community's interest in structures been fading? A primary reason is that traditional NLP pipelines are prone

to cascading errors and are less flexible than end-to-end learning. Part I discusses algorithms that integrate structures into modern deep learning architectures.

Part II aims to improve the efficiency of state-of-the-art deep learning architectures. We first focus on the transformers (Vaswani et al., 2017), a crucial ingredient of today's NLP and the backbone of many foundation models (Bommasani et al., 2021) such as BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020). We present two efficient attention models that reduce the overhead of transformers from quadratic to linear in the input lengths in Chapters 4 and 5. Chapter 6 turns to *recurrent* neural networks (RNNS), another important family of neural architectures. We present a unifying framework for many existing RNNs through the lens of weighted finite-state automata (WFSAs). It provides new insights into existing models and inspires future data-efficient and interpretable architectures.

1.1 Thesis Outline

Chapter 2 introduces SPIGOT, an algorithm that allows for incorporating discrete structured prediction as intermediate layers in neural nets, facilitating training NLP pipelines end-to-end. Predicting discrete structures involves the undifferentiable structured argmax, incompatible with neural learning SPIGOT derives a stable and efficient surrogate for the gradients for structured argmax respecting the structured constraints. Our empirical results verify that using SPIGOT to incorporate "learnable" structural inductive biases yields more accurate and interpretable decisions in downstream tasks. It can also be used in semi-supervised learning and unsupervised structure induction.

In Chapter 3, we introduce PaLM, a hybrid parser and language model. PaLM integrates syntactic structured inductive biases into neural language models with an attention layer over *text spans*. An unsupervised constituency parser can be derived from its attention weights using a greedy decoding algorithm. If syntactic annotations *are* available, the attention component can be trained in a supervised manner, providing syntactically-informed representations of the context, and further improving language modeling performance.

We present random feature attention (RFA) in Chapter 4. Drawing inspiration from the classical kernel methods, we devise linear-complexity alternatives to softmax attention. Canonical

attention calculates pairwise similarities between input tokens using a softmax normalization. RFA devises a linear-complexity approximation to it using random feature techniques and a kernel trick. RFA significantly improves time and memory efficiency when applied in transformers, with negligible or no accuracy loss. Besides, it reveals new insights into the connections between attention and recurrent neural networks (RNNs), another important model family in NLP.

We then introduce a holistic view of several recent efficient transformer models in Chapter 5. Although seemingly disparate, they can be subsumed into one unified abstraction, "compressing" the context with various strategies, primarily using hand-crafted heuristics. This perspective connects several efficient attention variants that would otherwise appear apart. Besides, it inspires a new algorithm that learns to compress the context in a context-dependent manner.

Finally, Chapter 6 formally studies the connections between recurrent models and weighted finite-state automata (WFSAs). WFSAs naturally relate to pattern-matching methods, offering an intuitive way to explain their decisions; together with their siblings hidden Markov models, they are fundamental tools for NLP. Besides, WFSAs are well-studied, interpretable, efficient, and flexible to design. Our theoretical findings prove that a family of modern recurrent and convolutional neural networks (CNNs), at least in their single-layer cases, are WFSAs parameterized with neural networks, which we dub *rational recurrences*. The significance is both theoretical and empirical: rational recurrences provide a unifying framework for existing approaches and a new way to characterize their capacities; they lead to an algorithm that "translates" WFSAs into neural networks, offering a flexible way to devise better-performing, more efficient, and more interpretable models neural architectures imbued with desired inductive biases.

Part I

Improving the Generalization of NLP Models with Symbolic Structures

Chapter 2

Backpropagating through Structured Argmax using a SPIGOT

Language is structured: meanings are articulated and perceived through hierarchical compositions of words. Linguistic structures provide valuable inductive biases that help models generalize to unseen scenarios (Mou et al., 2015; Xu et al., 2015; Peng et al., 2017, 2018c; Wu et al., 2021). In this part, we will first introduce SPIGOT, a new algorithm that integrates structured inductive biases into NLP models and allows for having discrete symbolic structured prediction as intermediate layers in the neural nets (Chapter 2). In Chapter 3, we will discuss PaLM, a hybrid parser and neural language model that improves language modeling through unsupervised syntactic parsing.

2.1 Introduction

Learning methods for natural language processing are increasingly dominated by end-to-end differentiable functions that can be trained using gradient-based optimization. Yet traditional NLP often assumed modular stages of processing that formed a pipeline; e.g., text was tokenized, then tagged with parts of speech, then parsed into a phrase-structure or dependency tree, then semantically analyzed. Pipelines, which make "hard" (i.e., discrete) decisions at each stage, appear to be incompatible with neural learning, leading many researchers to abandon earlier-

The material in this chapter is adapted from Peng et al. (2018b).

stage processing.

Inspired by findings that continue to see benefit from various kinds of linguistic or domain-specific preprocessing (He et al., 2017; Oepen et al., 2017; Ji and Smith, 2017), we argue that pipelines can be treated as layers in neural architectures for NLP tasks. Several solutions are readily available:

- Reinforcement learning (most notably the REINFORCE algorithm; Williams, 1992), and **structured attention** (SA; Kim et al., 2017). These methods replace argmax with a sampling or marginalization operation. We note two potential downsides of these approaches: (i) not all argmax-able operations have corresponding sampling or marginalization methods that are efficient, and (ii) inspection of intermediate outputs, which could benefit error analysis and system improvement, is more straightforward for hard decisions than for posteriors.

- The **straight-through estimator** (STE; Hinton, 2012) treats discrete decisions as if they were differentiable and simply passes through gradients. While fast and surprisingly effective, it ignores *constraints* on the argmax problem, such as the requirement that every word has exactly one syntactic parent. We will find, experimentally, that the quality of intermediate representations degrades substantially under STE.

This chapter introduces a new method, the **structured projection of intermediate gradients** optimization technique (SPIGOT; §2.2), which defines a proxy for the gradient of a loss function with respect to the input to argmax. Unlike STE's gradient proxy, SPIGOT aims to respect the constraints in the argmax problem. SPIGOT can be applied with any intermediate layer that is expressible as a constrained maximization problem, and whose feasible set can be projected onto. We show empirically that SPIGOT works even when the maximization and the projection are done approximately.

We offer two concrete architectures that employ structured argmax as an intermediate layer: semantic parsing with syntactic parsing in the middle, and sentiment analysis with semantic parsing in the middle (§2.3). These architectures are trained using a joint objective, with one part using data for the intermediate task, and the other using data for the end task. The datasets are not assumed to overlap at all, but the parameters for the intermediate task are affected by both parts of the training data.

Our experiments (§2.4) show that our architecture improves over a state-of-the-art semantic dependency parser, and that SPIGOT offers stronger performance than a pipeline, SA, and STE. On sentiment classification, we show that semantic parsing offers improvement over a BiLSTM, more so with SPIGOT than with alternatives. Our analysis considers how the behavior of the intermediate parser is affected by the end task (§2.5).

2.2 Method

Our aim is to allow a (structured) argmax layer in a neural network to be treated almost like any other differentiable function. This would allow us to place, for example, a syntactic parser in the middle of a neural network, so that the forward calculation simply calls the parser and passes the parse tree to the next layer, which might derive syntactic features for the next stage of processing.

The challenge is in the *backward* computation, which is key to learning with standard gradient-based methods. When its output is discrete as we assume here, argmax is a piecewise constant function. At every point, its gradient is either zero or undefined. So instead of using the true gradient, we will introduce a *proxy* for the gradient of the loss function with respect to the inputs to argmax, allowing backpropagation to proceed through the argmax layer. Our proxy is designed as an improvement to earlier methods (discussed below) that completely ignore constraints on the argmax operation. It accomplishes this through a projection of the gradients.

We first lay out notation, and then briefly review max-decoding and its relaxation (§2.2.1). We define SPIGOT in §2.2.2, and show how to use it to backpropagate through NLP pipelines in §2.2.3.

Notation. Our discussion centers around two tasks: a structured *intermediate* task followed by an *end* task, where the latter considers the outputs of the former (e.g., syntactic-then-semantic parsing). Inputs are denoted as $\mathbf{x}$, and end task outputs as $\mathbf{y}$. We use $\mathbf{z}$ to denote intermediate structures derived from $\mathbf{x}$. We will often refer to the intermediate task as "decoding", in the structured prediction sense. It seeks an output $\hat{\mathbf{z}} = \text{argmax}_{\mathbf{z} \in \mathcal{Z}} S$ from the feasible set $\mathcal{Z}$, maximizing a (learned, parameterized) scoring function S for the structured intermediate task. L

denotes the loss of the end task, which may or may not also involve structured predictions. We use $\Delta^{k-1} = \{\mathbf{p} \in \mathbb{R}^k \mid \mathbf{1}^\top \mathbf{p} = 1, \mathbf{p} \geq \mathbf{0}\}$ to denote the $(k-1)$-dimensional simplex. We denote the domain of binary variables as $\mathbb{B} = \{0, 1\}$, and the unit interval as $\mathbb{U} = [0, 1]$. By projection of a vector $\mathbf{v}$ onto a set $\mathcal{A}$, we mean the closest point in $\mathcal{A}$ to $\mathbf{v}$, measured by Euclidean distance: $\text{proj}_{\mathcal{A}}(\mathbf{v}) = \text{argmin}_{\mathbf{v}' \in \mathcal{A}} \|\mathbf{v}' - \mathbf{v}\|_2$.

2.2.1 Relaxed Decoding

Decoding problems are typically decomposed into a collection of "parts", such as arcs in a dependency tree or graph. In such a setup, each element of $\mathbf{z}$, z_i, corresponds to one possible part, and z_i takes a boolean value to indicate whether the part is included in the output structure. The scoring function S is assumed to decompose into a vector $\mathbf{s}(\mathbf{x})$ of part-local, input-specific scores:

$$\hat{\mathbf{z}} = \arg\max \mathbf{z} \in \mathcal{Z} S(\mathbf{x}, \mathbf{z}) = \arg\max \mathbf{z} \in \mathcal{Z} \mathbf{z}^\top \mathbf{s}(\mathbf{x}) \tag{2.1}$$

In the following, we drop $\mathbf{s}$'s dependence on $\mathbf{x}$ for clarity.

In many NLP problems, the output space $\mathcal{Z}$ can be specified by linear constraints (Roth and Yih, 2004):

$$\mathbf{A} \begin{bmatrix} \mathbf{z} \\ \psi \end{bmatrix} \leq \mathbf{b}, \tag{2.2}$$

where ψ are auxiliary variables (also scoped by argmax), together with integer constraints (typically, each $z_i \in \mathbb{B}$).

The problem in Equation 2.1 can be NP-complete in general, so the $\{0, 1\}$ constraints are often relaxed to $[0, 1]$ to make decoding tractable (Martins et al., 2009). Then the discrete combinatorial problem over $\mathcal{Z}$ is transformed into the optimization of a linear objective over a convex polytope $\mathcal{P} = \{\mathbf{p} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{p} \leq \mathbf{b}\}$, which is solvable in polynomial time (Bertsimas and Tsitsiklis, 1997). This is not necessary in some cases, where the argmax can be solved exactly with dynamic programming.

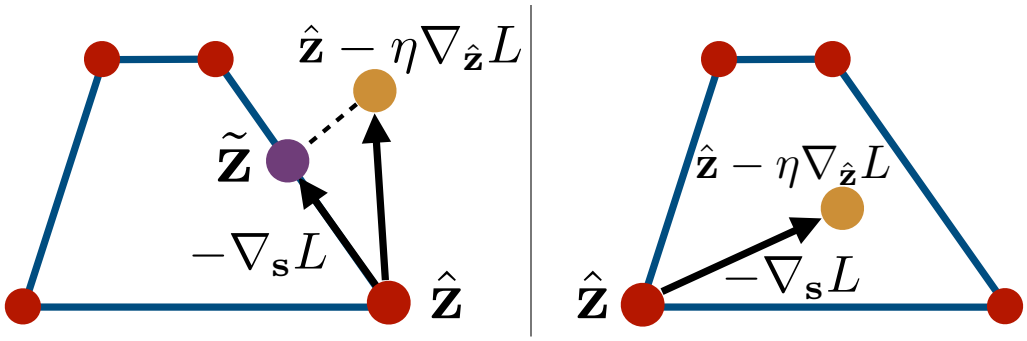


Figure 2.1: The original feasible set $\mathcal{Z}$ (red vertices), is relaxed into a convex polytope $\mathcal{P}$ (the area encompassed by blue edges). Left: making a gradient update to $\hat{\mathbf{z}}$ makes it step outside the polytope, and it is projected back to $\mathcal{P}$, resulting in the projected point $\tilde{\mathbf{z}}$. $\nabla_{\mathbf{s}} L$ is then along the edge. Right: updating $\hat{\mathbf{z}}$ keeps it within $\mathcal{P}$, and thus $\nabla_{\mathbf{s}} L = \eta \nabla_{\hat{\mathbf{z}}} L$.

2.2.2 From STE to SPIGOT

We now view structured argmax as an activation function that takes a vector of input-specific part-scores $\mathbf{s}$ and outputs a solution $\hat{\mathbf{z}}$. For backpropagation, to calculate gradients for parameters of $\mathbf{s}$, the chain rule defines:

$$\nabla_{\mathbf{s}} L = \mathbf{J}\ \nabla_{\hat{\mathbf{z}}} L, \tag{2.3}$$

where the Jacobian matrix $\mathbf{J} = \frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{s}}$ contains the derivative of each element of $\hat{\mathbf{z}}$ with respect to each element of $\mathbf{s}$. Unfortunately, argmax is a piecewise constant function, so its Jacobian is either zero (almost everywhere) or undefined (in the case of ties).

One solution, taken in *structured attention*, is to replace the argmax with marginal inference and a softmax function, so that $\hat{\mathbf{z}}$ encodes probability distributions over parts (Kim et al., 2017; Liu and Lapata, 2018). As discussed in §2.1, there are two reasons to avoid this modification. Softmax can only be used when marginal inference is feasible, by sum-product algorithms for example (Eisner, 2016; Friesen and Domingos, 2016); in general marginal inference can be #P-complete. Further, a soft intermediate layer will be less amenable to inspection by anyone wishing to understand and improve the model.

In another line of work, argmax is augmented with a strongly-convex penalty on the solutions (Martins and Astudillo, 2016; Amos and Kolter, 2017; Niculae and Blondel, 2017; Niculae et al., 2018; Mensch and Blondel, 2018). However, their approaches require solving a relaxation even when exact decoding is tractable. Also, the penalty will bias the solutions found by the decoder, which may be an undesirable conflation of computational and modeling concerns.

A simpler solution is the STE method (Hinton, 2012), which replaces the Jacobian matrix in Equation 2.3 by the identity matrix. This method has been demonstrated to work well when used to "backpropagate" through hard threshold functions (Bengio et al., 2013; Friesen and Domingos, 2018) and categorical random variables (Jang et al., 2016; Choi et al., 2017). We will test this method in a structured NLP setting for the first time in §2.4.

Consider for a moment what we would do if $\hat{\mathbf{z}}$ were a vector of parameters, rather than intermediate predictions. In this case, we are seeking points in $\mathcal{Z}$ that minimize L; denote that set of minimizers by $\mathcal{Z}^*$. Given $\nabla_{\hat{\mathbf{z}}} L$ and step size η, we would update $\hat{\mathbf{z}}$ to be $\hat{\mathbf{z}} - \eta \nabla_{\hat{\mathbf{z}}} L$. This update, however, might not return a value in the feasible set $\mathcal{Z}$, or even (if we are using a linear relaxation) the relaxed set $\mathcal{P}$.

SPIGOT therefore introduces a *projection* step that aims to keep the "updated" $\hat{\mathbf{z}}$ in the feasible set. Of course, we do not directly update $\hat{\mathbf{z}}$; we continue backpropagation through $\mathbf{s}$ and onward to the parameters. But the projection step nonetheless alters the parameter updates in the way that our proxy for "$\nabla_{\mathbf{s}} L$" is defined.

The procedure is defined as follows:

$$\hat{\mathbf{p}} = \hat{\mathbf{z}} - \eta \nabla_{\hat{\mathbf{z}}} L, \tag{2.4a}$$

$$\tilde{\mathbf{z}} = \text{proj}_{\mathcal{P}}(\hat{\mathbf{p}}), \tag{2.4b}$$

$$\nabla_{\mathbf{s}} L \triangleq \hat{\mathbf{z}} - \tilde{\mathbf{z}}. \tag{2.4c}$$

First, the method makes an "update" to $\hat{\mathbf{z}}$ as if it contained parameters (Equation 2.4a), letting $\hat{\mathbf{p}}$ denote the new value. Next, $\hat{\mathbf{p}}$ is projected back onto the (relaxed) feasible set (Equation 2.4b), yielding a feasible new value $\tilde{\mathbf{z}}$. Finally, the gradients with respect to $\mathbf{s}$ are computed by Equation 2.4c.

Due to the convexity of $\mathcal{P}$, the projected point $\tilde{\mathbf{z}}$ will always be unique, and is guaranteed to be no farther than $\hat{\mathbf{p}}$ from any point in $\mathcal{Z}^*$ (Luenberger and Ye, 2015).[1] Compared to STE, SPIGOT involves a projection and limits $\nabla_{\mathbf{s}} L$ to a smaller space to satisfy constraints. See Figure 2.1 for an illustration.

When efficient exact solutions (such as dynamic programming) are available, they can be

[1] Note that this property follows from $\mathcal{P}$'s convexity, and we do not assume the convexity of L.

used. Yet, we note that SPIGOT does not assume the argmax operation is solved exactly.

2.2.3 Backpropagation through Pipelines

Using SPIGOT, we now devise an algorithm to "backpropagate" through NLP pipelines. In these pipelines, an intermediate task's output is fed into an end task for use as features. The parameters of the complete model are divided into two parts: denote the parameters of the intermediate task model by $\boldsymbol{\phi}$ (used to calculate $\mathbf{s}$), and those in the end task model as $\boldsymbol{\theta}$.[2] As introduced earlier, the end-task loss function to be minimized is L, which depends on both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$.

Algorithm 1 describes the forward and backward computations. It takes an end task training pair $\langle \mathbf{x}, \mathbf{y} \rangle$, along with the intermediate task's feasible set $\mathcal{Z}$, which is determined by $\mathbf{x}$. It first runs the intermediate model and decodes to get intermediate structure $\hat{\mathbf{z}}$, just as in a standard pipeline. Then forward propagation is continued into the end-task model to compute loss L, using $\hat{\mathbf{z}}$ to define input features. Backpropagation in the end-task model computes $\nabla_{\boldsymbol{\theta}} L$ and $\nabla_{\hat{\mathbf{z}}} L$, and $\nabla_{\mathbf{s}} L$ is then constructed using Equations 2.4. Backpropagation then continues into the intermediate model, computing $\nabla_{\boldsymbol{\phi}} L$.

Due to its flexibility, SPIGOT is applicable to many training scenarios. When there is no $\langle \mathbf{x}, \mathbf{z} \rangle$ training data for the intermediate task, SPIGOT can be used to induce latent structures for the end-task (Yogatama et al., 2017; Kim et al., 2017; Choi et al., 2017, *inter alia*). When intermediate-task training data *is* available, one can use SPIGOT to adopt joint learning by minimizing an interpolation of L (on end-task data $\langle \mathbf{x}, \mathbf{y} \rangle$) and an intermediate-task loss function $\widetilde{L}$ (on intermediate task data $\langle \mathbf{x}, \mathbf{z} \rangle$). This is the setting in our experiments; note that we do not assume any overlap in the training examples for the two tasks.

2.3 Solving the Projections

In this section we discuss how to compute approximate projections for the two intermediate tasks considered in this chapter, arc-factored unlabeled dependency parsing and first-order semantic dependency parsing.

[2] Nothing prohibits tying across pre-argmax parameters and post-argmax parameters; this separation is notationally convenient but not at all necessary.

Algorithm 1 Forward and backward computation with SPIGOT.

procedure SPIGOT($\mathbf{x}, \mathbf{y}, \mathcal{Z}$)
  Construct $\mathbf{A}$, $\mathbf{b}$ such that $\mathcal{Z} = \{\mathbf{p} \in \mathbb{Z}^d \mid \mathbf{A}\mathbf{p} \leq \mathbf{b}\}$
  $\mathcal{P} \leftarrow \{\mathbf{p} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{p} \leq \mathbf{b}\}$ ▷ Relaxation
  Forwardprop and compute $\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})$
  $\hat{\mathbf{z}} \leftarrow \operatorname{argmax}_{\mathbf{z} \in \mathcal{Z}} \mathbf{z}^\top \mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x})$ ▷ Intermediate decoding
  Forwardprop and compute L given $\mathbf{x}$, $\mathbf{y}$, and $\hat{\mathbf{z}}$
  Backprop and compute $\nabla_{\boldsymbol{\theta}} L$ and $\nabla_{\hat{\mathbf{z}}} L$
  $\tilde{\mathbf{z}} \leftarrow \operatorname{proj}_{\mathcal{P}}(\hat{\mathbf{z}} - \eta \nabla_{\hat{\mathbf{z}}} L)$ ▷ Projection
  $\nabla_{\mathbf{s}} L \leftarrow \hat{\mathbf{z}} - \tilde{\mathbf{z}}$
  Backprop and compute $\nabla_{\boldsymbol{\phi}} L$
end procedure

In early experiments we observe that for both tasks, projecting with respect to *all* constraints of their original formulations using a generic quadratic program solver was prohibitively slow. Therefore, we construct relaxed polytopes by considering only a subset of the constraints.[3] The projection then decomposes into a series of singly constrained quadratic programs (QP), each of which can be efficiently solved in linear time.

The two approximate projections discussed here are used in backpropagation only. In the forward pass, we solve the decoding problem using the models' original decoding algorithms.

Arc-factored unlabeled dependency parsing. For unlabeled dependency trees, we impose $[0, 1]$ constraints and single-headedness constraints.[4]

Formally, given a length-n input sentence, excluding self-loops, an arc-factored parser considers $d = n(n-1)$ candidate arcs. Let $i \rightarrow j$ denote an arc from the ith token to the jth, and $\sigma(i \rightarrow j)$ denote its index. We construct the relaxed feasible set by:

$$\mathcal{P}_{\text{DEP}} = \left\{ \mathbf{p} \in \mathbb{U}^d \,\middle|\, \sum_{i \neq j} p_{\sigma(i \rightarrow j)} = 1, \forall j \right\}, \tag{2.5}$$

i.e., we consider each token j individually, and force single-headedness by constraining the number of arcs incoming to j to sum to 1. Algorithm 2 summarizes the procedure to project onto $\mathcal{P}_{\text{DEP}}$. Line 3 forms a singly constrained QP, and can be solved in $O(n)$ time (Brucker, 1984).

[3] A parallel work introduces an active-set algorithm to solve the same class of quadratic programs (Niculae et al., 2018). It might be an efficient approach to solve the projections in Equation 2.4b, which we leave to future work.

[4] It requires $O(n^2)$ auxiliary variables and $O(n^3)$ additional constraints to ensure well-formed tree structures (Martins et al., 2013).

Algorithm 2 Projection onto the relaxed polytope $\mathcal{P}_{\text{DEP}}$ for dependency tree structures. Let bold $\boldsymbol{\sigma}(\cdot \to j)$ denote the index set of arcs incoming to j. For a vector $\mathbf{v}$, we use $\mathbf{v}_{\boldsymbol{\sigma}(\cdot \to j)}$ to denote vector $[v_k]_{k \in \boldsymbol{\sigma}(\cdot \to j)}$.

procedure DEPPROJ($\hat{\mathbf{p}}$)
  for $j = 1, 2, \ldots, n$ **do**
    $\tilde{\mathbf{z}}_{\boldsymbol{\sigma}(\cdot \to j)} \leftarrow \text{proj}_{\Delta^{n-2}} \left(\hat{\mathbf{p}}_{\boldsymbol{\sigma}(\cdot \to j)} \right)$
  end for
  return $\tilde{\mathbf{z}}$
end procedure

First-order semantic dependency parsing. Semantic dependency parsing uses labeled bilexical dependencies to represent sentence-level semantics (Oepen et al., 2014, 2015, 2016). Each dependency is represented by a labeled directed arc from a head token to a modifier token, where the arc label encodes broadly applicable semantic relations. Figure 2.2 diagrams a semantic graph from the DELPH-IN MRS-derived dependencies (DM), together with a syntactic tree.

We use a state-of-the-art semantic dependency parser (Peng et al., 2017) that considers three types of parts: heads, unlabeled arcs, and labeled arcs. Let $\sigma(i \xrightarrow{\ell} j)$ denote the index of the arc from i to j with semantic role ℓ. In addition to $[0, 1]$ constraints, we constrain that the predictions for labeled arcs sum to the prediction of their associated unlabeled arc:

$$\mathcal{P}_{\text{SDP}} \left\{ \mathbf{p} \in \mathbb{U}^d \,\middle|\, \sum_{\ell} p_{\sigma(i \xrightarrow{\ell} j)} = p_{\sigma(i \to j)}, \forall i \neq j \right\}. \tag{2.6}$$

This ensures that exactly one label is predicted if and only if its arc is present. The projection onto $\mathcal{P}_{\text{SDP}}$ can be solved similarly to Algorithm 2. We drop the determinism constraint imposed by Peng et al. (2017) in the backward computation.

2.4 Experiments

We empirically evaluate our method with two sets of experiments: using syntactic tree structures in semantic dependency parsing, and using semantic dependency graphs in sentiment classification.

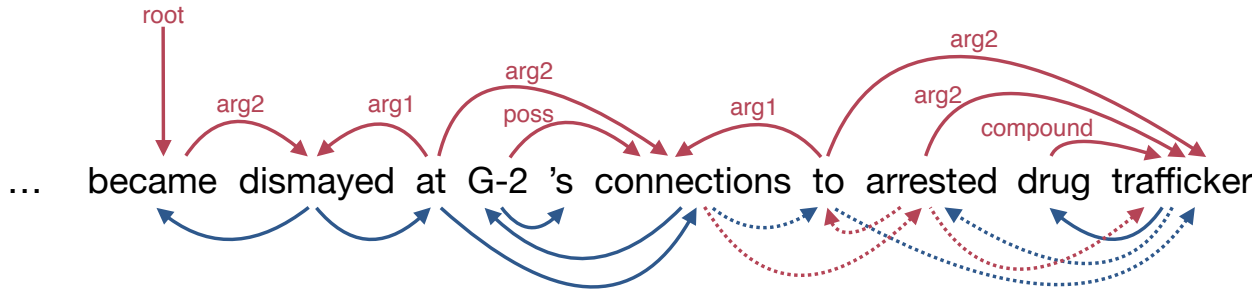


Figure 2.2: A development instance annotated with both gold DM semantic dependency graph (red arcs on the top), and gold syntactic dependency tree (blue arcs at the bottom). A pretrained syntactic parser predicts the same tree as the gold; the semantic parser backpropagates into the intermediate syntactic parser, and changes the dashed blue arcs into dashed red arcs (§2.5).

2.4.1 Syntactic-then-Semantic Parsing

In this experiment we consider an intermediate syntactic parsing task, followed by semantic dependency parsing as the end task. We first briefly review the neural network architectures for the two models, and then introduce the datasets and baselines.

Syntactic dependency parser. For intermediate syntactic dependencies, we use the unlabeled arc-factored parser of Kiperwasser and Goldberg (2016). It uses bidirectional LSTMs (BiLSTM) to encode the input, followed by a multilayer-perceptron (MLP) to score each potential dependency. One notable modification is that we replace their use of Chu-Liu/Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967) with the Eisner algorithm (Eisner, 1996, 2000), since our dataset is in English and mostly projective.

Semantic dependency parser. We use the basic model of Peng et al. (2017) (denoted as NeurboParser) as the end model. It is a first-order parser, and uses local factors for heads, unlabeled arcs, and labeled arcs. NeurboParser does not use syntax. It first encodes an input sentence with a two-layer BiLSTM, and then computes part scores with two-layer tanh-MLPs. Inference is conducted with AD^3 (Martins et al., 2015). To add syntactic features to NeurboParser, we concatenate a token's contextualized representation to that of its syntactic head, predicted by the intermediate parser. Formally, given length-n input sentence, we first run a BiLSTM. We use the concatenation of the two hidden representations $\mathbf{h}_j = [\overrightarrow{\mathbf{h}}_j; \overleftarrow{\mathbf{h}}_j]$ at each position j as the contextualized token representations. We then concatenate $\mathbf{h}_j$ with the representation of its head $\mathbf{h}_{\text{HEAD}(j)}$

by

$$\widetilde{\mathbf{h}}_j = [\mathbf{h}_j; \mathbf{h}_{\text{HEAD}(j)}] = \left[\mathbf{h}_j; \sum_{i \neq j} \hat{z}_{\sigma(i \to j)}\, \mathbf{h}_i\right], \tag{2.7}$$

where $\hat{\mathbf{z}} \in \mathbb{B}^{n(n-1)}$ is a binary encoding of the tree structure predicted by by the intermediate parser. We then use $\widetilde{\mathbf{h}}_j$ anywhere $\mathbf{h}_j$ would have been used in NeurboParser. In backpropagation, we compute $\nabla_{\hat{\mathbf{z}}} L$ with an automatic differentiation toolkit (DyNet; Neubig et al., 2017).

We note that this approach can be generalized to convolutional neural networks over graphs (Mou et al., 2015; Duvenaud et al., 2015; Kipf and Welling, 2017, *inter alia*), recurrent neural networks along paths (Xu et al., 2015; Roth and Lapata, 2016, *inter alia*) or dependency trees (Tai et al., 2015). We choose to use concatenations to control the model's complexity, and thus to better understand which parts of the model work.

We refer the readers to Kiperwasser and Goldberg (2016) and Peng et al. (2017) for further details of the parsing models.

Training procedure. Following previous work, we minimize structured hinge loss (Tsochantaridis et al., 2004) for both models. We jointly train both models from scratch, by randomly sampling an instance from the union of their training data at each step. In order to isolate the effect of backpropagation, we do not share any parameters between the two models.[5] Implementation details are summarized in Appendix C.1.

Data.

- For semantic dependencies, we use the English dataset from SemEval 2015 Task 18 (Oepen et al., 2015).[6] Among the three formalisms provided by the shared task, we consider DELPH-IN MRS-derived dependencies (DM) and Prague Semantic Dependencies (PSD).[7] It includes §00–19 of the WSJ corpus as training data, §20 and §21 for development and

[5] Parameter sharing has proved successful in many related tasks (Collobert and Weston, 2008; Søgaard and Goldberg, 2016; Ammar et al., 2016; Swayamdipta et al., 2016, 2017, *inter alia*), and could be easily combined with our approach.

[6] http://sdp.delph-in.net/

[7] We drop the third (PAS) because its structure is highly predictable from parts-of-speech, making it less interesting.

Model	DM		PSD	
	UF	LF	UF	LF
NeurboParser	–	89.4	–	77.6
Freda3	–	90.4	–	78.5
pipeline	91.8	90.8	88.4	78.1
sa	91.6	90.6	87.9	78.1
ste	92.0	91.1	**88.9**	**78.9**
spigot	**92.4**	**91.6**	88.6	**78.9**

(a) F_1 on in-domain test set.

Model	DM		PSD	
	UF	LF	UF	LF
NeurboParser	–	84.5	–	75.3
Freda3	–	85.3	–	76.4
pipeline	87.4	85.8	85.5	75.6
sa	87.3	85.6	84.9	75.9
ste	87.7	86.4	**85.8**	76.6
spigot	**87.9**	**86.7**	85.5	**77.1**

(b) F_1 on out-of-domain test set.

Table 2.1: Semantic dependency parsing performance in both unlabeled (UF) and labeled (LF) F_1 scores. Bold font indicates the best performance.

in-domain test data, resulting in a 33,961/1,692/1,410 train/dev./test split, and 1,849 out-of-domain test instances from the Brown corpus.[8]

- For syntactic dependencies, we use the Stanford Dependency (de Marneffe and Manning, 2008) conversion of the the Penn Treebank WSJ portion (Marcus et al., 1993). To avoid data leak, we depart from standard split and use §20 and §21 as development and test data, and the remaining sections as training data. The number of training/dev./test instances is 40,265/2,012/1,671.

Baselines. We compare to the following baselines:

- A pipelined system (pipeline). The pretrained parser achieves 92.9 test unlabeled attach-

[8]The organizers remove, e.g., instances with cyclic graphs, and thus only a subset of the WSJ corpus is included. See Oepen et al. (2015) for details.

ment score (UAS).[9]

- Structured attention networks (SA; Kim et al., 2017). We use the inside-outside algorithm (Baker, 1979) to populate $\mathbf{z}$ with arcs' marginal probabilities, use log-loss as the objective in training the intermediate parser.

- The straight-through estimator (STE; Hinton, 2012), introduced in §2.2.2.

Results. Table 2.1 compares the semantic dependency parsing performance of SPIGOT to all five baselines. FREDA3 (Peng et al., 2017) is a state-of-the-art variant of NEURBOPARSER that is trained using multitask learning to jointly predict three different semantic dependency graph formalisms. Like the basic NEURBOPARSER model that we build from, FREDA3 does not use any syntax. Strong DM performance is achieved by using joint learning and an ensemble (Peng et al., 2018c), which is beyond fair comparisons to the models discussed here.

We found that using syntactic information improves semantic parsing performance: using pipelined syntactic head features brings 0.5–1.4% absolute labeled F_1 improvement to NEURBOPARSER. Such improvements are smaller compared to previous works, where dependency path and syntactic relation features are included (Almeida and Martins, 2015; Ribeyre et al., 2015; Zhang et al., 2016), indicating the potential to get better performance by using more syntactic information, which we leave to future work.

Both STE and SPIGOT use hard syntactic features. By allowing backpropation into the intermediate syntactic parser, they both consistently outperform PIPELINE. On the other hand, when marginal syntactic tree structures are used, SA outperforms PIPELINE only on the out-of-domain PSD test set, and improvements under other cases are not observed.

Compared to STE, SPIGOT outperforms STE on DM by more than 0.3% absolute labeled F_1, both in-domain and out-of-domain. For PSD, SPIGOT achieves similar performance to STE on in-domain test set, but has a 0.5% absolute labeled F_1 improvement on out-of-domain data, where syntactic parsing is less accurate.

[9]Note that this number is not comparable to the parsing literature due to the different split. As a sanity check, we found in preliminary experiments that the same parser architecture achieves 93.5 UAS when trained and evaluated with the standard split, close to the results reported by Kiperwasser and Goldberg (2016).

2.4.2 Semantic Dependencies for Sentiment Classification

Our second experiment uses semantic dependency graphs to improve sentiment classification performance. We are not aware of any efficient algorithm that solves marginal inference for semantic dependency graphs under determinism constraints, so we do not include a comparison to SA.

Architectures. Here we use NEURBOPARSER as the intermediate model, as described in §2.4.1, but with no syntactic enhancements.

Sentiment classifier. We first introduce a baseline that does not use any structural information. It learns a one-layer BiLSTM to encode the input sentence, and then feeds the sum of all hidden states into a two-layer ReLU-MLP.

To use semantic dependency features, we concatenate a word's BiLSTM-encoded representation to the averaged representation of its heads, together with the corresponding semantic roles, similarly to that in Equation 2.7.[10] Then the concatenation is fed into an affine transformation followed by a ReLU activation. The rest of the model is kept the same as the BiLSTM baseline.

Training procedure. We use structured hinge loss to train the semantic dependency parser, and log-loss for the sentiment classifier. Due to the discrepancy in the training data size of the two tasks (33K vs. 7K), we pre-train a semantic dependency parser, and then adopt joint training together with the classifier. In the joint training stage, we randomly sample 20% of the semantic dependency training instances each epoch. The model is trained for up to 30 epochs in the joint training stage. We apply early-stopping based on sentiment classification development accuracy. For semantic dependency parser, we follow the hyperparameters described in §2.4.1.

Data. For semantic dependencies, we use the DM dataset introduced in §2.4.1.

We consider a binary classification task using the Stanford Sentiment Treebank (Socher et al., 2013). It consists of roughly 10K movie review sentences from Rotten Tomatoes. The full dataset

[10] In a well-formed semantic dependency graph, a token may have multiple heads. Therefore we use average instead of the sum in Equation 2.7.

Model	Accuracy (%)
BiLSTM	84.8
PIPELINE	85.7
STE	85.4
SPIGOT	**86.3**

Table 2.2: Test accuracy of sentiment classification on Stanford Sentiment Treebank. Bold font indicates the best performance.

includes a rating on a scale from 1 to 5 for each constituent (including the full sentences), resulting in more than 200K instances. Following previous work (Iyyer et al., 2015), we only use full-sentence instances, with neutral instances excluded (3s) and the remaining four rating levels converted to binary "positive" or "negative" labels. This results in a 6,920/872/1,821 train/dev./test split.

Results. Table 2.2 compares our SPIGOT method to three baselines. Pipelined semantic dependency predictions brings 0.9% absolute improvement in classification accuracy, and SPIGOT outperforms all baselines. In this task STE achieves slightly worse performance than a fixed pre-trained PIPELINE.

2.5 Analysis

We examine here how the intermediate model is affected by the end-task training signal. Is the end-task signal able to "overrule" intermediate predictions?

We use the syntactic-then-semantic parsing model (§2.4.1) as a case study. Table 2.3 compares a pipelined system to one jointly trained using SPIGOT. We consider the development set instances where both syntactic and semantic annotations are available, and partition them based on whether the two systems' syntactic predictions agree (SAME), or not (DIFF). The second group includes sentences with much lower syntactic parsing accuracy (91.3 vs. 97.4 UAS), and SPIGOT further reduces this to 89.6. Even though these changes hurt syntactic parsing accuracy, they lead to a 1.1% absolute gain in labeled F_1 for semantic parsing. Furthermore, SPIGOT has an overall less detrimental effect on the intermediate parser than STE: using SPIGOT, intermediate dev. parsing UAS drops to 92.5 from the 92.9 pipelined performance, while STE reduces it to 91.8.

Split	# Sent.	Model	UAS	DM
SAME	1011	PIPELINE	97.4	94.0
		SPIGOT	97.4	94.3
DIFF	681	PIPELINE	91.3	88.1
		SPIGOT	89.6	89.2

Table 2.3: Syntactic parsing performance (in unlabeled attachment score, UAS) and DM semantic parsing performance (in labeled F_1) on different groups of the development data. Both systems predict the same syntactic parses for instances from SAME, and they disagree on instances from DIFF (§2.5).

We then take a detailed look and categorize the changes in intermediate trees by their correlations with the semantic graphs. Specifically, when a modifier m's head is changed from h to h' in the tree, we consider three cases: (a) h' is a head of m in the semantic graph; (b) h' is a modifier of m in the semantic graph; (c) h is the modifier of m in the semantic graph. The first two reflect modifications to the syntactic parse that rearrange semantically linked words to be neighbors. Under (c), the semantic parser removes a syntactic dependency that reverses the direction of a semantic dependency. These cases account for 17.6%, 10.9%, and 12.8%, respectively (41.2% combined) of the total changes. Making these changes, of course, is complicated, since they often require *other* modifications to maintain well-formedness of the tree. Figure 2.2 gives an example.

2.6 Related Work

Joint learning in NLP pipelines. To avoid cascading errors, much effort has been devoted to joint decoding in NLP pipelines (Habash and Rambow, 2005; Cohen and Smith, 2007; Goldberg and Tsarfaty, 2008; Lewis et al., 2015; Zhang et al., 2015, *inter alia*). However, joint inference can sometimes be prohibitively expensive. Recent advances in representation learning facilitate exploration in the joint learning of multiple tasks by sharing parameters (Collobert and Weston, 2008; Blitzer et al., 2006; Finkel and Manning, 2010; Zhang and Weiss, 2016; Hashimoto et al., 2017, *inter alia*).

Differentiable optimization. Gould et al. (2016) review the generic approaches to differentiation in bi-level optimization (Bard, 2010; Kunisch and Pock, 2013). Amos and Kolter (2017) extend their efforts to a class of subdifferentiable quadratic programs. However, they both require that

the intermediate objective has an invertible Hessian, limiting their application in NLP. In another line of work, the steps of a gradient-based optimization procedure are unrolled into a single computation graph (Stoyanov et al., 2011; Domke, 2012; Goodfellow et al., 2013; Brakel et al., 2013). This comes at a high computational cost due to the second-order derivative computation during backpropagation. Moreover, constrained optimization problems (like many NLP problems) often require projection steps within the procedure, which can be difficult to differentiate through (Belanger and McCallum, 2016; Belanger et al., 2017).

2.7 Summary

In this chapter, we presented SPIGOT, a novel approach to backpropagating through neural network architectures that include discrete structured decisions in intermediate layers. SPIGOT devises a proxy for the gradients with respect to argmax's inputs, employing a projection that aims to respect the constraints in the intermediate task. We empirically evaluate our method with two architectures: a semantic parser with an intermediate syntactic parser, and a sentiment classifier with an intermediate semantic parser. Experiments show that SPIGOT achieves stronger performance than baselines under both settings, and outperforms state-of-the-art systems on semantic dependency parsing.

Chapter 3

PaLM: A Hybrid Parser and Language Model

This chapter presents PaLM, a hybrid **pa**rser and neural **l**anguage **m**odel. Building on an RNN language model, PaLM adds an attention layer over *text spans* in the left context. An unsupervised constituency parser can be derived from its attention weights, using a greedy decoding algorithm. We evaluate PaLM on language modeling, and empirically show that it outperforms strong baselines. If syntactic annotations *are* available, the attention component can be trained in a supervised manner, providing syntactically-informed representations of the context, and further improving language modeling performance.

3.1 Introduction

Recent language models have shown very strong data-fitting performance (Jozefowicz et al., 2016; Merity et al., 2018). They offer useful products including, most notably, contextual embeddings (Peters et al., 2018; Radford et al., 2019), which benefit many NLP tasks such as text classification (Howard and Ruder, 2018) and dataset creation (Zellers et al., 2018).

Language models are typically trained on large amounts of raw text, and therefore do not explicitly encode any notion of structural information. Structures in the form of syntactic trees have been shown to benefit both classical NLP models (Gildea and Palmer, 2002; Punyakanok

The material in this chapter is adapted from Peng et al. (2019).

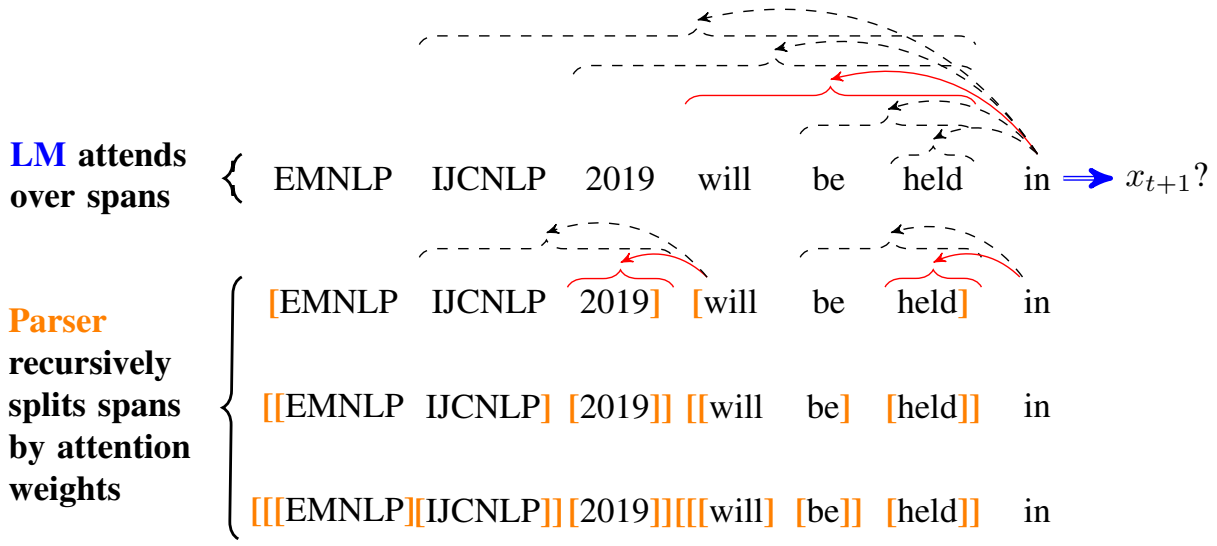


Figure 3.1: An illustration of PaLM. The LM (first line) predicts the next word (x_{t+1}, double blue arrow) by attending over previous spans ending in time $t-1$ (dashed lines). The parser (lines 2–4) splits the prefix into two spans (line 2) by the taking the top scoring attended span (red solid line) and the prefix leading to it. It then recursively splits the two sub-spans using the same procedure (line 3). Finally, spans of length two are trivially split into terminal nodes (line 4).

et al., 2008; Das et al., 2012, *inter alia*) and recent state-of-the-art neural models (Dyer et al., 2016; Swayamdipta et al., 2018; Strubell et al., 2018, *inter alia*). In this paper we show that LMs can benefit from syntactically-inspired encoding of the context.

We introduce PaLM (**pa**rser and **l**anguage **m**odel; Figure 3.1), a novel hybrid model combining an RNN language model with a constituency parser. The LM in PaLM attends over *spans of tokens*, implicitly learning which syntactic constituents are likely. A span-based parser is then derived from the attention information (Stern et al., 2017).

PaLM has several benefits. First, it is an intuitive and lightweight way of incorporating structural information (§3.2.1), requiring no marginal inference, which can be computationally expensive (Jelinek and Lafferty, 1991; Chelba and Jelinek, 1998; Roark, 2001; Dyer et al., 2016; Buys and Blunsom, 2018; Kim et al., 2019, *inter alia*). Second, the attention can be syntactically informed, in the sense that the attention component can optionally be supervised using syntactic annotations, either through pretraining or by joint training with the LM (§3.2.2). Last, PaLM can derive an unsupervised constituency parser (§3.2.2), whose parameters are estimated purely using the language modeling objective.

To demonstrate the empirical benefits of PaLM, we experiment with language modeling

(§3.3). PaLM outperforms the AWD-LSTM model (Merity et al., 2018) on both the Penn Treebank (PTB; Marcus et al., 1993) and WikiText-2 (Merity et al., 2017) datasets by small but consistent margins in the unsupervised setup. When the parser is trained jointly with the language model, we see additional perplexity reductions in both cases.

3.2 PaLM—Parser and Language Model

We describe PaLM in detail. At its core is an attention component, gathering the representations of preceding *spans* at each time step. Similar to self-attention, PaLM can be implemented on top of RNN encoders (Parikh et al., 2016), or as it is (Vaswani et al., 2017). Here we encode the tokens using a left-to-right RNN, denoted with vectors $\mathbf{h}_t$.[1]

Below we describe the span-attention component and the parsing algorithm. We use $[i, j], i \leq j$ to denote text span $x_i \dots x_j$, i.e., inclusive on both sides. When $i = j$, it consists of a single token.

Notations and background. A language model learns a joint distribution of a corpus, which typically autoregressively factorizes as

$$\mathbb{P}_\theta(\mathbf{x}) = \prod_{t=1}^{n} \mathbb{P}_\theta(x_t \mid \mathbf{x}_{<t}), \tag{3.1}$$

where $\mathbf{x}_{<t} = x_1 \dots x_{t-1}$ denotes the preceding tokens up to time step t (exclusive), and θ denotes the model parameters.

Recurrent neural networks provide natural solutions to encoding variable-length inputs, and prove successful in modeling languages (Zaremba et al., 2014; Gal and Ghahramani, 2016; Merity et al., 2018, *inter alia*). At a very high level, a recurrent cell can be seen as a parameterized function, which is recessively applied along the sequence:

$$\mathbf{h}_t = \mathbf{f}\left(\mathbf{h}_{t-1}, \mathbf{z}_t\right), \tag{3.2}$$

where $\mathbf{z}_t$ denotes the embedding vector of token x_t, and $\mathbf{h}_t$ vectors the encoded hidden states, which are used for onward computation, e.g., feeding into the next RNN layer or into a classifier.

[1] We experiment with a strong LSTM implementation for language modeling (Merity et al., 2018), see §3.3.

Such architectures are sometimes augmented with self attention mechanisms, allowing for better modeling of long-range dependencies (Parikh et al., 2016). An attention function gathers the context, usually with a (normalized) weighted sum, where the weights are computed from the current hidden state and the context. A language model is constrained from conditioning on future tokens, and hence its attention function can only access $\mathbf{h}_{<t}$ at timestep t.

3.2.1 Span Attention

We want the language model attention to gather context information aware of syntactic structures. A constituency parse can be seen as a collection of syntactic constituents, i.e., token spans. Therefore we attend over preceding spans, which we describe in this section.[2]

At step t, PaLM attends over the spans ending at $t-1$, up to a maximum length m, i.e., $\{[i, t-1]\}_{i=t-m}^{t-1}$.[3] Essentially, this can be seen as splitting the prefix span $[t-m, t-1]$ into two, and attending over the one on the right. Such a span attention mechanism is inspired by the top-down greedy span parser of Stern et al. (2017), which recursively divides phrases. In §3.2.2, we will use a similar algorithm to derive a constituency parser from the span attention weights.

Bidirectional span representation with rational RNNs. Meaningful span representations are crucial in span-based tasks (Lee et al., 2017a; Peng et al., 2018c; Swayamdipta et al., 2018, *inter alia*). Typical design choices are based on start and end token vectors contextualized by bidirectional RNNs. However, a language model does not have access to future words, and hence running a backward RNN from right to left is less straightforward: one will have to start an RNN running at each token, which is computationally daunting (Kong et al., 2016). To compute span representations efficiently, we use *rational* RNNs (RRNNs; Chapter 6).

RRNNs are a family of RNN models, where the recurrent function can be computed with weighted finite-state automata (WFSAs). We use the unigram WFSA–inspired RRNN (Chapter 6),

[2]Standard token-based self-attention naturally relates to dependency structures through head selection (Strubell et al., 2018). In a left-to-right factored language model, dependencies are less natural if we want to allow a child to precede its parent.

[3]m is set to 20. This reduces the number of considered spans from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$. Besides practical concerns, it makes less sense if a phrase goes beyond one single sentence (the average sentence length of WSJ training sentences is 21).

where the cell state update is

$$\mathbf{f}_t = \boldsymbol{\sigma}\left(\mathbf{W}_f \mathbf{h}_t\right), \tag{3.3a}$$

$$\mathbf{u}_t = (\mathbf{1} - \mathbf{f}_t) \odot \tanh\left(\mathbf{W}_u \mathbf{h}_t\right), \tag{3.3b}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t. \tag{3.3c}$$

$\mathbf{f}_t$ is a forget gate implemented with the elementwise sigmoid function $\boldsymbol{\sigma}$, and $\odot$ denotes elementwise multiplication. $\mathbf{W}_u$ and $\mathbf{W}_f$ are learned matrices. Bias terms are suppressed for clarity.[4]

Slightly overloading the notation, let $\overrightarrow{\mathbf{c}}_{i,j}$ denote the encoding of span $[i, j]$ by running a forward RRNN in Eq. 3.3, from left to right. It can be efficiently computed by subtracting $\overrightarrow{\mathbf{c}}_{i-1}$ from $\overrightarrow{\mathbf{c}}_j$, weighted by a product of forget gates:

$$\overrightarrow{\mathbf{c}}_{i,j} = \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \bigodot_{k=i}^{j} \overrightarrow{\mathbf{f}}_k. \tag{3.4}$$

$\overrightarrow{\mathbf{f}}_k$ vectors are the forget gates. See §D.2 for a detailed derivation.

Using this observation, we now derive an efficient algorithm to calculate the span representations based on bidirectional RRNNs. In the interest of space, Algorithm 3 describes the forward span representations. It takes advantage of the distributivity property of rational RNNs (Chapter 6), and the number of RNN function calls is linear in the input length.[5] Although overall asymptotic time complexity is still quadratic, Algorithm 3 only involves elementwise operations, which can be easily parallelized on modern GPUs. The backward one (right to left) is analogous.

Computing attention. As in standard attention, we use a normalized weighted sum of the span representations. Let $\mathbf{g}([i, j]) = [\overrightarrow{\mathbf{c}}_{i,j}; \overleftarrow{\mathbf{c}}_{i,j}]$ denote the representation of span $[i, j]$, which concatenates the forward and backward representations calculated using Algorithm 3. The context vector

[4] Unlike other RNNs such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014), RRNNs do not apply an affine transformation or a nonlinear dependency of $\mathbf{c}_t$ on $\mathbf{c}_{t-1}$.

[5] In contrast, the dynamic program of Kong et al. (2016) for segmental (span-scoring) RNNs requires a quadratic number of recurrent function calls, since they use LSTMs, where distributivity does not hold.

Algorithm 3 RRNN-based span representation. / denotes elementwise division. Both elementwise product and division are implemented in log-space.

procedure SPANREPR($\{\overrightarrow{\mathbf{c}}_t, \overrightarrow{\mathbf{f}}_t\}$)
  ▷ Accumulate forward forget gates
  for $i = 1, \ldots, n$ **do**
    $\overrightarrow{\mathbf{f}}_{1,i} = \overrightarrow{\mathbf{f}}_{1,i-1} \odot \overrightarrow{\mathbf{f}}_i$
  end for
  for $j = 1, \ldots, n$ **do**
    for $i = 1, \ldots, j$ **do**
      $\overrightarrow{\mathbf{c}}_{i,j} = \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \odot \overrightarrow{\mathbf{f}}_{1,j} / \overrightarrow{\mathbf{f}}_{1,i-1}$
    end for
  end for
  return $\overrightarrow{\mathbf{c}}_{i,j}$ vectors
end procedure

$\mathbf{a}_t$ is

$$\mathbf{a}_{t+1} = \sum_{i=0}^{m-1} \omega_{t,i}\, \mathbf{g}([t-i, t]), \tag{3.5a}$$

$$\omega_{t,i} = \frac{\exp s_{t,i}}{\sum_{j=0}^{m-1} \exp s_{t,j}}. \tag{3.5b}$$

Here $s_{t,i}$ is implemented as an MLP, taking as input the concatenation of $\mathbf{h}_{t+1}$ and $\mathbf{g}([t-i, t])$ and outputs the attention score. The context vector is then concatenated with the hidden state $\bar{\mathbf{h}}_{t+1} = [\mathbf{h}_{t+1}; \mathbf{a}_{t+1}]$, and fed into onward computation.

In summary, given an input sequence, PaLM:

- First uses a standard left-to-right RNN to calculate the hidden states $\mathbf{h}_t$.
- Feed $\mathbf{h}_t$ vectors into a one-layer bidirectional rational RNN (Eq. 3.3), using Algorithm 3 to compute the span representations.
- Attends over spans (Eq. 3.5b) to predict the next word.

3.2.2 Attention-Based Constituency Parsing

We next describe the other facet or PaLM: the constituency parser. Our parsing algorithm is similar to the greedy top-down algorithm proposed by Stern et al. (2017). It recursively divides a span into two smaller ones, until a single-token span, i.e., a leaf, is reached. The order of the

partition specifies the tree structure.[6] Formally, for a maximum span length m, at each time step $j+1$, we split the span $[j-m+1, j]$ into two smaller parts $[j-m+1, k_0]$ and $[k_0+1, j]$. The partitioning point is greedily selected, maximizing the attention scores of spans ending at j:[7]

$$k_0 = \arg\max k \in \{0, \ldots, m-1\} s_{j,k}. \tag{3.6}$$

The span is directly returned as a leaf if it contains a single token. A full parse is derived by running the algorithm recursively, starting with the input as a single span (with a special end-of-sentence mark at the end). The runtime is $\mathcal{O}(n^2)$, with $n-1$ partitioning points. See Figure 3.1 for an illustration.

Supervising the attention. Now that we are able to derive phrase structures from attention weights, we can further inform the attention if syntactic annotations are available, using oracle span selections. For each token, the gold selection is a m-dimensional binary vector, and then normalized to sum to one, denoted $\mathbf{y}_t$.[8] We add a cross-entropy loss (averaged across the training data) to the language modeling objective, with λ trading off between the two:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \frac{\lambda}{N} \sum_{t=1}^{N} \mathcal{H}(\mathbf{y}_t, \boldsymbol{\omega}_t), \tag{3.7}$$

with $\boldsymbol{\omega}$ being the attention distribution at step t, and N the length of the training corpus. As we will see in §3.3, providing syntactically guided span attention improves language modeling performance.

Discussion. PaLM provides an intuitive way to inject structural inductive bias into the language model—by supervising the attention distribution. This setting can be seen as a very lightweight multitask learning, where no actual syntactic tree is predicted during language modeling training or evaluation. The attention weight predictor (i.e., the s scores in Eq. 3.5b) can be replaced with

[6] It is only able to produce binarized unlabeled trees.

[7] Another natural choice is to maximize the sum of the scores of $[i, k_0]$ and $[k_0+1, j]$. The attention score of $[i, k_0]$ is computed at time step k_0, and hence does not know anything about the other span on the right. Therefore we consider only the score of the right span. We find in preliminary experiments that including the left span score in the maximization slightly hurts the parsing performance.

[8] Not necessarily one-hot: multiple spans can end at the same token.

an off-the-shelf parser, or deterministically set (e.g., to simulate left/right-branching).

3.3 Experiments

We evaluate PaLM on language modeling. We experiment with the Penn Treebank corpus (PTB) and WikiText-2 (WT2). We follow the preprocessing of Mikolov et al. (2010) for PTB and Merity et al. (2018) for WT2. More implementation details are described in Appendix D.1. We compare two configurations of PaLM:

- PaLM-U builds on top of AWD-LSTM (Merity et al., 2018), a state-of-the-art of LSTM implementation for language modeling. The span attention is included *before* the last layer.[9]

- PaLM-S is the same model as PaLM-U, but uses phrase syntax annotation to provide additional supervision to the attention component (§3.2.2).[10]

We compare against the AWD-LSTM baseline. On PTB, we also compare to two models using structural information in language modeling: parsing-reading-predict networks (PRPN; Shen et al., 2018a) predicts syntactic distance as structural features for language modeling; ordered-neuron LSTM (ON-LSTM; Shen et al., 2018b) posits a novel ordering on LSTM gates, simulating the covering of phrases at different levels in a constituency parse. On PTB we also compare to PaLM-RB, a baseline deterministically setting the attention scores (Eq. 3.5b) in decreasing order, such that the derived trees will be right-branching.[11]

Tables 3.1 and 3.2 summarize the language modeling results. On both datasets, the unsupervised configuration (PaLM-U) outperforms AWD-LSTM. On PTB, PaLM-U achieves similar performance to ON-LSTM and much better performance than PRPN. PaLM-S further reduces the perplexity by 1.6–3.4% (relative), showing that incorporating structural information with supervised span attention helps language modeling. Naively promoting right-branching attention (PaLM-RB) yields no improvement over the baseline.

[9] Preliminary experiments show that including the span attention after the last layer yields similar empirical results, but is more sensitive to hyperparameters.

[10] We use the WSJ portion of PTB for parsing annotations.

[11] We set scores to $m, m-1, \ldots, 1$, before the softmax.

Model	# Params.	Dev.	Test
AWD-LSTM	24M	60.0	57.3
PRPN	-	-	62.0
ON-LSTM	25M	58.3	56.2
PaLM-U	24M	58.6	56.4
PaLM-RB	24M	60.1	57.5
PaLM-S	24M	**57.9**	**55.5**

Table 3.1: PTB language modeling perplexity (lower is better). Bold fonts indicate best performance. Several recent works report better language modeling perplexity (Yang et al., 2019; Takase et al., 2018; Dai et al., 2019, *inter alia*). Their contribution is orthogonal to ours and not head-to-head comparable to the models in the table.

Model	# Params.	Dev.	Test
AWD-LSTM	33M	68.6	65.8
PaLM-U	36M	68.4	65.4
PaLM-S	36M	**65.5**	**63.2**

Table 3.2: WikiText-2 language modeling perplexity (lower is better). Bold fonts indicate best performance.

Unsupervised constituency parsing. We evaluate the parser component of PaLM-U on WSJ-40. It uses the same data as in language modeling, but filters out sentences longer than 40 tokens after punctuation removal. The model is selected based on language modeling validation perplexity.

In addition to PRPN, we compare to DIORA (Drozdov et al., 2019), which uses an inside-outside dynamic program in an autoencoder. Table 3.3 shows the F_1 results. PaLM outperforms the right branching baseline, but is not as accurate as the other models.[12] This indicates that the type of syntactic trees learned by it, albeit useful to the LM component, do not correspond well to PTB-like syntactic trees.

Discussion. Despite its strong performance, the parsing algorithm used by Shen et al. (2018a) and Shen et al. (2018b) suffers from an incomplete support issue (Dyer et al., 2019). More precisely, it fails to produce "close-open-open," i.e.,) ((structures. As a result, the parser is intrinsically biased toward right-branching structures. PaLM, on the other hand, scores all the spans, and therefore can produce any binary tree spanning a given sentence: the algorithm recovers any given binary tree by letting $s_{j,j-i} = 1$ if the tree contains nonterminal $[i, j]$, and 0 otherwise.[13]

[12] Evaluation on WSJ-10, which contains sentences with 10 or less tokens, shows a similar trend.
[13] The maximum span length m is only forced in language modeling training and evaluation.

Model	Unlabeled F_1
Right Branching	40.7
†DIORA	60.6
‡PRPN	52.4
‡PaLM-U	42.0

Table 3.3: Unlabeled unsupervised parsing F_1 on WSJ-40. ‡ trains on the training split of WSJ, while † trains on AllNLI (Htut et al., 2018). The PRPN result is taken from Drozdov et al. (2019).

	% Left Splits	% Right Splits
Random	$39.3_{\pm 10.5}$	$41.2_{\pm 8.8}$
PaLM-U	1.1	85.6
Gold	6.5	52.7

Table 3.4: Percentage of left and right splits. The first row shows the numbers averaging over 25 differently randomly initialized PaLM models, without training. $\pm$ indicates standard deviation.

Is PaLM empirically biased toward any branching direction? In greedily selected trees, we measure the percentage of left-branching splits (dividing $[i, j]$ into $[i, j-1]$ and j) and right-branching splits (dividing $[i, j]$ into i and $[i+1, j]$).[14] Table 3.4 summarizes the results on WSJ-40 test set. The first row shows the results for randomly initialized models without training. We observe no significant trend of favoring one branching direction over the other. However, after training with the language modeling objective, PaLM-U shows a clear right-skewness more than it should: it produces much more right-branching structures than the gold annotation. This means that the span attention mechanism has learned to emphasize longer prefixes, rather than make strong Markov assumptions. More exploration of this effect is left to future work.

3.4 Summary

This chapter presented PaLM, a hybrid parser and language model. PaLM attends over the preceding text spans. From its attention weights phrase structures can be derived. The attention component can be separately trained to provide syntactically-informed context gathering. PaLM outperforms strong baselines on language modeling. Incorporating syntactic supervision during training leads to further language modeling improvements. Training our unsupervised model

[14] We exclude trivial splits dividing a length-2 span into two tokens.

on large-scale corpora could result in both stronger language models and, potentially, stronger parsers.

Part II

Efficient Neural Architectures for NLP

Chapter 4

Random Feature Attention

Artificial intelligence applications have seen unprecedented progress, which can be primarily attributed to the development of computationally-intensive deep learning models. However, their growing computational requirements have heightened the barriers to entry to state-of-the-art research and negatively impacted the environment. For example, searching for optimal architectures for certain NLP tasks (So et al., 2019) is estimated to cost more than 1 million USD, hardly affordable to less-resourced research groups. It emits a comparable amount of carbon dioxide as an average car does in 5 years (Strubell et al., 2019). The first part of this thesis aims to improve the efficientcy of state-of-the-art NLP models, thereby promoting the accessibility of cutting-edge NLP research, especially for less-funded institutions, and mitigate its environmental concerns. We explore two directions to achieve this. In Chapter 4, we introduce a linear complexity alternative to the well established attention mechanism, and use it to improve the efficiency of transformer models without losing accuracy. Chapter 5 derives a holistic view of several recent efficient transformer models, and discuss the insights and practical benefits of this new perspective.

4.1 Introduction

Transformer architectures (Vaswani et al., 2017) have achieved tremendous success on a variety of sequence modeling tasks (Ott et al., 2018; Radford et al., 2019; Parmar et al., 2018; Devlin et al.,

The material in this chapter is adapted from Peng et al. (2021).

2019; Parisotto et al., 2020, *inter alia*). Under the hood, the key component is attention (Bahdanau et al., 2015), which models pairwise interactions of the inputs, regardless of their distances from each other. This comes with quadratic time and memory costs, making the transformers computationally expensive, especially for long sequences. A large body of research has been devoted to improving their time and memory efficiency (Tay et al., 2020c). Although better *asymptotic* complexity and prominent gains for long sequences have been achieved (Lee et al., 2019; Child et al., 2019; Beltagy et al., 2020, *inter alia*), in practice, many existing approaches are less well-suited for moderate-length ones: the additional computation steps required by some approaches can overshadow the time and memory they save (Kitaev et al., 2020; Wang et al., 2020b; Roy et al., 2020, *inter alia*).

This chapter proposes **r**andom **f**eature **a**ttention (RFA), an efficient attention variant that scales linearly in sequence length in terms of time and space, and achieves practical gains for both long and moderate length sequences. RFA builds on a kernel perspective of softmax (Rawat et al., 2019). Using the well-established random feature maps (Rahimi and Recht, 2007; Avron et al., 2016; §4.2), RFA approximates the dot-then-exponentiate function with a kernel trick (Hofmann et al., 2008): $\exp(\mathbf{x} \cdot \mathbf{y}) \approx \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$. Inspired by its connections to gated recurrent neural networks (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and fast weights (Schmidhuber, 1992), we further augment RFA with an optional gating mechanism, offering a straightforward way of learning with recency bias when locality is desired.

RFA and its gated variant (§4.3) can be used as a drop-in substitute for the canonical softmax attention, and increase the number of parameters by less than 0.1%. We explore its applications in transformers on language modeling, machine translation, and long text classification (§4.4). Our experiments show that RFA achieves comparable performance to vanilla transformer baselines in all tasks, while outperforming a recent related approach (Katharopoulos et al., 2020). The gating mechanism proves particularly useful in language modeling: the gated variant of RFA outperforms the transformer baseline on WikiText-103. RFA shines in decoding, even for shorter sequences. In our head-to-head comparison on machine translation benchmarks, RFA decodes around 2× faster than a transformer baseline, *without* accuracy loss. Comparisons to several recent efficient transformer variants on three long text classification datasets show that RFA is competitive in terms of both accuracy and efficiency.

Our analysis (§4.5) shows that more significant time and memory efficiency improvements can be achieved for longer sequences: 12× decoding speedup with less than 10% of the memory for 2,048-length outputs.

4.2 Background

4.2.1 Attention in Sequence Modeling

The attention mechanism (Bahdanau et al., 2015) has been widely used in many sequence modeling tasks. Its dot-product variant is the key building block for the state-of-the-art transformer architectures (Vaswani et al., 2017). Let $\{\mathbf{q}_t\}_{t=1}^{N}$ denote a sequence of N **query** vectors, that attend to sequences of M **key** and **value** vectors. At each timestep, the attention linearly combines the values weighted by the outputs of a softmax:

$$\text{attn}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right) = \sum_i \frac{\exp\left(\mathbf{q}_t \cdot \mathbf{k}_i / \tau\right)}{\sum_j \exp\left(\mathbf{q}_t \cdot \mathbf{k}_j / \tau\right)} \mathbf{v}_i^\top. \tag{4.1}$$

τ is the temperature hyperparameter determining how "flat" the softmax is (Hinton et al., 2015).[1]

Calculating attention for a single query takes $\mathcal{O}(M)$ time and space. For the full sequence of N queries the space amounts to $\mathcal{O}(MN)$. When the computation *cannot* be parallelized across the queries, e.g., in autoregressive decoding, the time complexity is quadratic in the sequence length.

4.2.2 Random Feature Methods

The theoretical backbone of this chapter is the unbiased estimation of the Gaussian kernel by Rahimi and Recht (2007). Based on Bochner's theorem (Bochner, 1955), Rahimi and Recht (2007) proposed random Fourier features to approximate a desired shift-invariant kernel. The method nonlinearly transforms a pair of vectors $\mathbf{x}$ and $\mathbf{y}$ using a **random feature map** ϕ; the inner product between $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ approximates the kernel evaluation on $\mathbf{x}$ and $\mathbf{y}$. More precisely:

[1] $M = N$ in self-attention; they may differ, e.g., in the cross attention of a sequence-to-sequence model.

Theorem 1 (Rahimi and Recht, 2007). *Let $\phi : \mathbb{R}^d \to \mathbb{R}^{2D}$ be a nonlinear transformation:*

$$\phi(\mathbf{x}) = \sqrt{1/D}\Big[\sin(\mathbf{w}_1 \cdot \mathbf{x}), \ldots, \sin(\mathbf{w}_D \cdot \mathbf{x}), \cos(\mathbf{w}_1 \cdot \mathbf{x}), \ldots, \cos(\mathbf{w}_D \cdot \mathbf{x})\Big]^\top. \tag{4.2}$$

When d-dimensional random vectors $\mathbf{w}_i$ are independently sampled from $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$,

$$\mathbb{E}_{\mathbf{w}_i}\left[\phi(\mathbf{x}) \cdot \phi(\mathbf{y})\right] = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2\right). \tag{4.3}$$

Variance of the estimation is inversely proportional to D (Yu et al., 2016).

Random feature methods proved successful in speeding up kernel methods (Oliva et al., 2015; Avron et al., 2017; Sun, 2019, *inter alia*), and more recently are used to efficiently approximate softmax (Rawat et al., 2019). In §4.3.1, we use it to derive an unbiased estimate to $\exp(\langle \cdot, \cdot \rangle)$ and then an efficient approximation to softmax attention.

4.3 Model

This section presents RFA (§4.3.1) and its gated variant (§4.3.2). In §4.3.3 we lay out several design choices and relate RFA to prior works. We close by practically analyzing RFA's complexity (§4.3.4).

4.3.1 Random Feature Attention

RFA builds on an unbiased estimate to $\exp(\langle \cdot, \cdot \rangle)$ from Theorem 1, which we begin with:

$$\begin{aligned} \exp\left(\mathbf{x} \cdot \mathbf{y} / \sigma^2\right) &= \exp\left(\|\mathbf{x}\|^2 / 2\sigma^2 + \|\mathbf{y}\|^2 / 2\sigma^2\right) \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2\right) \\ &\approx \exp\left(\|\mathbf{x}\|^2 / 2\sigma^2 + \|\mathbf{y}\|^2 / 2\sigma^2\right)\ \phi(\mathbf{x}) \cdot \phi(\mathbf{y}). \end{aligned} \tag{4.4}$$

The last line does *not* have any nonlinear interaction between $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$, allowing for a linear time/space approximation to attention. For clarity we assume the query and keys are unit

vectors.[2]

$$\begin{aligned}\operatorname{attn}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right) &= \sum_i \frac{\exp\left(\mathbf{q}_t \cdot \mathbf{k}_i / \sigma^2\right)}{\sum_j \exp\left(\mathbf{q}_t \cdot \mathbf{k}_j / \sigma^2\right)} \mathbf{v}_i^\top \\ &\approx \sum_i \frac{\phi\left(\mathbf{q}_t\right)^\top \phi\left(\mathbf{k}_i\right) \mathbf{v}_i^\top}{\sum_j \phi\left(\mathbf{q}_t\right) \cdot \phi\left(\mathbf{k}_j\right)} \\ &= \frac{\phi\left(\mathbf{q}_t\right)^\top \sum_i \phi\left(\mathbf{k}_i\right) \otimes \mathbf{v}_i}{\phi\left(\mathbf{q}_t\right) \cdot \sum_j \phi\left(\mathbf{k}_j\right)} = \textsc{Rfa}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right).\end{aligned} \tag{4.5}$$

$\otimes$ denotes the outer product between vectors, and σ^2 corresponds to the temperature term τ in Eq. 4.1.

RFA can be used as a drop-in-replacement for softmax-attention.

(a) The input is revealed in full to **cross attention** and **encoder self-attention**. Here RFA calculates attention using Eq. 4.5.

(b) In **causal attention** RFA attends only to the prefix.[3] This allows for a recurrent computation. Tuple $(\mathbf{S}_t \in \mathbb{R}^{2D \times d}, \mathbf{z}_t \in \mathbb{R}^{2D})$ is used as the "hidden state" at time step t to keep track of the history, similar to those in RNNs. $2D$ denotes the size of $\phi(\cdot)$. Then

$$\textsc{Rfa}(\mathbf{q}_t, \{\mathbf{k}_i\}_{i \le t}, \{\mathbf{v}_i\}_{i \le t}) = \frac{\phi\left(\mathbf{q}_t\right)^\top \sum_{i \le t} \phi\left(\mathbf{k}_i\right) \otimes \mathbf{v}_i}{\phi\left(\mathbf{q}_t\right) \cdot \sum_{j \le t} \phi\left(\mathbf{k}_j\right)} \tag{4.6}$$

Let $\mathbf{S}_t \triangleq \sum_{i \le t} \phi(\mathbf{k}_i) \otimes \mathbf{v}_i$, and $\mathbf{z}_t \triangleq \sum_{i \le t} \phi(\mathbf{k}_i)$; both can be calculated recurrently. Assuming $\mathbf{S}_0 = \mathbf{0}$ and $\mathbf{z}_0 = \mathbf{0}$:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \phi\left(\mathbf{k}_t\right) \otimes \mathbf{v}_t, \quad \mathbf{z}_t = \mathbf{z}_{t-1} + \phi\left(\mathbf{k}_t\right), \quad t \ge 1. \tag{4.7}$$

Algorithms 4 and 5 describe causal and cross random feature attention's computation procedures, and Figure 4.1 compares RFA against the softmax attention.

Analogously to the softmax attention, RFA has its multiheaded variant (Vaswani et al., 2017). In our experiments we use causal RFA in a transformer language model (§4.4.1), and both cross and causal RFA in the decoder of a sequence-to-sequence machine translation model.

[2] This can be achieved by ℓ_2-normalizing the query and keys. See §4.3.3 for a related discussion.
[3] It is also sometimes called "decoder self-attention" or "autoregressive attention."

Algorithm 4 Causal random feature attention.

procedure RFA-CAUSAL($\{\mathbf{q}_i\}_{i=1}^N$, $\{\mathbf{k}_i\}_{i=1}^N$, $\{\mathbf{v}_i\}_{i=1}^N$)
  ▷ $\mathbf{S}$ is a $D \times d$ matrix
  ▷ $\mathbf{z}$ is a D-dimensional vector
  $\mathbf{S}, \mathbf{z} \leftarrow \mathbf{0}, \mathbf{0}$
  for $i = 1$ **to** N **do**
    $\widetilde{\mathbf{q}}_i, \widetilde{\mathbf{k}}_i \leftarrow \phi(\mathbf{q}_i), \phi(\mathbf{k}_i)$ ▷ Random feature maps
    $\mathbf{S} \leftarrow \mathbf{S} + \widetilde{\mathbf{k}}_i \otimes \mathbf{v}_i$
    $\mathbf{z} \leftarrow \mathbf{z} + \widetilde{\mathbf{k}}_i$
    $\mathbf{h}_i^\top \leftarrow \widetilde{\mathbf{q}}_i^\top \mathbf{S} / (\widetilde{\mathbf{q}}_i \cdot \mathbf{z})$
  end for
  return $\{\mathbf{h}_i\}_{i=1}^N$
end procedure

4.3.2 RFA-Gate: Learning with Recency Bias

The canonical softmax attention does *not* have any explicit modeling of distance or locality. In learning problems where such inductive bias is crucial (Ba et al., 2016; Parmar et al., 2018; Miconi et al., 2018; Li et al., 2019, *inter alia*), transformers heavily rely on positional encodings. Answering to this, many approaches have been proposed, e.g., learning the attention spans (Sukhbaatar et al., 2019; Wu et al., 2020), and enhancing the attention computation with recurrent (Hao et al., 2019; Chen et al., 2019) or convolutional (Wu et al., 2019; Mohamed et al., 2019) components.

RFA faces the same issue, but its causal attention variant (Eq. 4.6) offers a straightforward way of learning with recency bias. We draw inspiration from its connections to RNNs, and augment RFA with a learned gating mechanism (Hochreiter and Schmidhuber, 1997; Cho et al., 2014, *inter alia*):

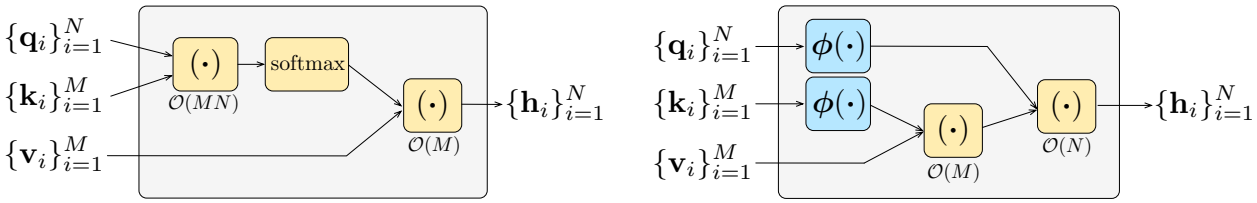
$$
\begin{aligned}
g_t &= \text{sigmoid}(\mathbf{w}_g \cdot \mathbf{x}_t + b_g), \\
\mathbf{S}_t &= g_t \, \mathbf{S}_{t-1} + (1 - g_t) \, \phi(\mathbf{k}_t) \otimes \mathbf{v}_t, \\
\mathbf{z}_t &= g_t \, \mathbf{z}_{t-1} + (1 - g_t) \, \phi(\mathbf{k}_t).
\end{aligned}
\tag{4.8}
$$

$\mathbf{w}_g$ and b_g are learned parameters, and $\mathbf{x}_t$ is the input representation at timestep t.[4] By multiplying the learned scalar gates $0 < g_t < 1$ against the hidden state $(\mathbf{S}_t, \mathbf{z}_t)$, history is exponentially decayed, favoring more recent context.

[4] In multihead attention (Vaswani et al., 2017), $\mathbf{k}_t$ and $\mathbf{v}_t$ are calculated from $\mathbf{x}_t$ using learned affine transformations.

Algorithm 5 Cross random feature attention.

procedure RFA-CROSS($\{\mathbf{q}_i\}_{i=1}^N$, $\{\mathbf{k}_i\}_{i=1}^M$, $\{\mathbf{v}_i\}_{i=1}^M$)
  ▷ $\mathbf{S}$ is a $D \times d$ matrix
  ▷ $\mathbf{z}$ is a D-dimensional vector
  $\mathbf{S}, \mathbf{z} \leftarrow \mathbf{0}, \mathbf{0}$
  for $i = 1$ **to** M **do**
    $\widetilde{\mathbf{k}}_i \leftarrow \phi(\mathbf{k}_i)$ ▷ Random feature map
    $\mathbf{S} \leftarrow \mathbf{S} + \widetilde{\mathbf{k}}_i \otimes \mathbf{v}_i^\top$
    $\mathbf{z} \leftarrow \mathbf{z} + \widetilde{\mathbf{k}}_i$
  end for
  for $i = 1$ **to** N **do**
    $\widetilde{\mathbf{q}}_i \leftarrow \phi(\mathbf{q}_i)$ ▷ Random feature map
    $\mathbf{h}_i^\top \leftarrow \widetilde{\mathbf{q}}_i^\top \mathbf{S} / (\widetilde{\mathbf{q}}_i \cdot \mathbf{z})$
  end for
  return $\{\mathbf{h}_i\}_{i=1}^N$
end procedure



(a) Softmax attention.

(b) Random feature attention.

Figure 4.1: Computation graphs for softmax attention (left) and random feature attention (right). Here, we assume cross attention with source length M and target length N.

The gating mechanism shows another benefit of RFA: it would be otherwise more difficult to build similar techniques into the softmax attention, where there is no clear sense of "recurrence." It proves useful in our language modeling experiments (§4.4.1).

4.3.3 Discussion

On query and key norms, and learned random feature variance. Eq. 4.5 assumes both the query and keys are of norm-1. It therefore approximates a softmax attention that normalizes the queries and keys before multiplying them, and then scales the logits by dividing them by σ^2. Empirically, this normalization step scales down the logits (Vaswani et al., 2017) and enforces that $-1 \leq \mathbf{q}^\top \mathbf{k} \leq 1$. In consequence, the softmax outputs would be "flattened" if not for σ, which can be set *a priori* as a hyperparameter (Yu et al., 2016; Avron et al., 2017; Sun, 2019, *inter*

alia). Here we instead learn it from data with the reparameterization trick (Kingma and Welling, 2014):

$$\widetilde{\mathbf{w}}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d), \quad \mathbf{w}_i = \boldsymbol{\sigma} \circ \widetilde{\mathbf{w}}_i. \tag{4.9}$$

$\mathbf{I}_d$ is the $d \times d$ identity matrix, and $\circ$ denotes elementwise product between vectors. d-dimensional vector $\boldsymbol{\sigma}$ is learned, but random vectors $\widetilde{\mathbf{w}}_i$ are *not*.[5]

This norm-1 constraint is never mandatory. Rather, we employ it for notation clarity and easier implementation. Here we present a RFA *without* imposing this constraint. Let $C(\mathbf{x}) = \exp(\|\mathbf{x}\|^2 / 2\sigma^2)$. From Eq. 4.4 we have $\text{attn}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right) =$

$$\begin{aligned}\sum_i \frac{\exp\left(\mathbf{q}_t \cdot \mathbf{k}_i / \sigma^2\right)}{\sum_j \exp\left(\mathbf{q}_t \cdot \mathbf{k}_j / \sigma^2\right)} \mathbf{v}_i^\top &\approx \sum_i \frac{C(\mathbf{q}_t)\, C(\mathbf{k}_i)\, \boldsymbol{\phi}\left(\mathbf{q}_t\right)^\top \boldsymbol{\phi}\left(\mathbf{k}_i\right) \mathbf{v}_i^\top}{\sum_j C(\mathbf{q}_t)\, C(\mathbf{k}_j)\, \boldsymbol{\phi}\left(\mathbf{q}_t\right) \cdot \boldsymbol{\phi}\left(\mathbf{k}_j\right)} \\ &= \frac{\boldsymbol{\phi}\left(\mathbf{q}_t\right)^\top \sum_i C(\mathbf{k}_i)\, \boldsymbol{\phi}\left(\mathbf{k}_i\right) \otimes \mathbf{v}_i}{\boldsymbol{\phi}\left(\mathbf{q}_t\right) \cdot \sum_j C(\mathbf{k}_j)\, \boldsymbol{\phi}\left(\mathbf{k}_j\right)}.\end{aligned} \tag{4.10}$$

The specific attention computation is similar to those in §4.3.1. In sum, lifting the norm-1 constraint brings an additional scalar term $C(\cdot)$. In preliminary experiments we find that the norm-1 constraint has little impact on the performance when σ is set properly or learned from data.

Going beyond the Gaussian kernel. More broadly, random feature methods can be applied to a family of shift-invariant kernels, with the Gaussian kernel being one of them. In the same family, the order-1 arc-cosine kernel (Cho and Saul, 2009) can be approximated with feature map: $\boldsymbol{\phi}_{\text{arccos}}(\mathbf{x}) = \sqrt{1/D}[\text{ReLU}(\mathbf{w}_1 \cdot \mathbf{x}), \ldots, \text{ReLU}(\mathbf{w}_D \cdot \mathbf{x})]^\top$ (Alber et al., 2017).[6] In our experiments, the Gaussian and arc-cosine variants achieve similar performance. This supplements the exploration of alternatives to softmax in attention (Tsai et al., 2019; Gao et al., 2019).

Random feature attention as an associative memory. RFA can be seen as a particular case of the heteroassociative memory (Binder et al., 2009).[7] An memory entry $\mathbf{v}_t$ binds to the transformed key $\boldsymbol{\phi}(\mathbf{k}_t)$ with an outer product $\boldsymbol{\phi}(\mathbf{k}_t)\mathbf{v}_t^\top$, and is then stored to the memory with a Hebbian

[5] This departs from Eq. 4.2 by lifting the isotropic assumption imposed on the Gaussian distribution: note the difference between the vector $\boldsymbol{\sigma}$ in Eq. 4.9 and the scalar σ in Eq. 4.3. We find this improves the performance in practice (§4.4), even though the same result in Theorem 1 may not directly apply.

[6] Apart from replacing the sinusoid functions with ReLU, it constructs $\mathbf{w}_i$ in the same way as Eq. 4.9.

[7] In an associative memory (Dayan and Abbott, 2001), another variant, the keys and the stored memories are identical.

learning rule $\mathbf{S}_t = \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)\mathbf{v}_t^\top$ (Hebb, 1949). A query $\mathbf{q}$ retrieves from the memory with $\phi(\mathbf{q})^\top \mathbf{S}_t$; the output is then normalized by $\phi(\mathbf{q}_q)^\top \mathbf{z}_t$.

This reveals that RFA is limited in terms of the number of entries it can store *without interference*. Without loss of generality, let's assume $\phi(\mathbf{q})$ and $\{\phi(\mathbf{k}_i)\}$ are unit vectors. $\mathbf{v}_t$ is said to be stored without interference, if these exists a $\phi(\mathbf{q})$ such that $\phi(\mathbf{q})^\top \mathbf{S}_t = \mathbf{v}_t$. When $\{\mathbf{v}_i\}$ are bound to linearly-independent keys, they each can be stored without interference (Touretzky, 2013). For RFA, this suggests that when the number of entries M is larger than $2D$, every entry can only be retrieved with a mixture of others and thus none can be stored without interference; and this could happen even when $M < 2D$.

The associative memory view of RFA justifies RFA-GATE—the gating mechanism decays older entries, "making place" for newer ones. In §5.3.3, we describe a method to explicitly pop older entries from the memory.

Relations to prior work. Katharopoulos et al. (2020) inspire the causal attention variant of RFA. They use a feature map based on the exponential linear unit activation (Clevert et al., 2016): $\text{elu}(\cdot) + 1$. It significantly *underperforms* both the baseline and RFA in our controlled experiments, showing the importance of a properly-chosen feature map. Random feature approximation of attention is also explored by a concurrent work (Choromanski et al., 2021), with applications in masked language modeling for proteins. They propose positive random features to approximate softmax, aiming for a lower variance in critical regions. RFA instead normalizes the queries and keys before random projection to reduce variance. Going beyond both, RFA establishes the benefits of random feature methods as a more universal substitute for softmax across all attention variants, facilitating its applications in, e.g., sequence-to-sequence learning.

There are interesting connections between gated RFA and fast weights (Schmidhuber, 1992, 1993; Ba et al., 2016; Miconi et al., 2018, *inter alia*). Emphasizing recent patterns, they learn a temporal memory to store history similarly to Eqs. 4.8. The main difference is that RFA additionally normalizes the output using $\phi(\mathbf{q}_t) \cdot \mathbf{z}$ as in Eq. 4.6, a by-product of approximating softmax's partition function. It is intriguing to study the role of this normalization term, which we leave to future work.

4.3.4 Complexity Analysis

Time. Scaling linearly in the sequence lengths, RFA needs less computation (in terms of number of operations) for long sequences. This implies speedup wherever the quadratic-time softmax attention *cannot* be fully-parallelized across time steps. More specifically:

- Significant speedup can be expected in autoregressive *decoding*, both conditional (e.g., machine translation) and unconditional (e.g., sampling from a language model). For example, 1.9× speedup is achieved in our machine translation experiments (§4.4.2);[8] and more for longer sequences (e.g., 12× for 2,048-length ones; §4.5).

- Some applications (e.g., language modeling, text classification) reveal inputs to the model in full.[9] When there are enough threads to parallelize softmax attention across time steps, hardly any speedup from RFA can be achieved; when there are not, typically for very long sequences (>1,000), substantial speed gain is possible. For example, RFA does *not* achieve any speedup when working with 512-length context (§4.4.1), but achieves a 5.3× speedup with 4,000-length context (§4.4.3).

Memory. Asymptotically, RFA has a better memory efficiency than its softmax counterpart (linear vs. quadratic). To reach a more practical conclusion, we include in our analysis the cost of the feature maps. ϕ's memory overhead largely depends on its size D. For example, let's consider the cross attention of a decoder. RFA uses $\mathcal{O}(4D + 2Dd)$ space to store $\phi(\mathbf{q}_t)$, $\sum_i \phi(\mathbf{k}_i) \otimes \mathbf{v}_i$, and $\sum_i \phi(\mathbf{k}_i)$ (Eq. 4.5; line 12 of Algo. 5).[10] In contrast, softmax cross attention stores the encoder outputs with $\mathcal{O}(Md)$ memory, with M being the source length. In this case RFA has a lower memory overhead when $2D \ll M$. Typically D should be no less than d in order for reasonable approximation (Yu et al., 2016); In a transformer model, d is the size of an attention head, which is usually around 64 or 128 (Vaswani et al., 2017; Ott et al., 2018). This suggests that RFA can achieve significant memory saving with longer sequences, which is supported by our empirical analysis in §4.5. Further, using moderate sized feature maps is also desirable, so that its overhead does not overshadow the time and memory RFA saves. We experiment with D at d and $2d$; the benefit of using $D > 2d$ is marginal.

[8] For example, WMT14 EN-DE training data has an average output length of 29.6.

[9] A causal masking is usually used to prevent the model from accessing future tokens in language models.

[10] RFA *never* constructs the $M \times 2D \times d$ tensor $[\phi(\mathbf{k}_i) \otimes \mathbf{v}_i]_i$, but sequentially processes the sequence.

Table 4.1 considers a sequence-to-sequence model, and breaks down the complexity comparisons to training (with teacher forcing; Williams and Zipser, 1989) and autoregressive decoding. Here we assume enough threads to fully parallelize softmax attention across timesteps when the inputs are revealed to the model in full. RFA has a lower space complexity, since it never explicitly populates the attention matrices. As for time, RFA trains in linear time, and so does the softmax attention: in teacher-forcing training a standard transformer decoder parallelizes the attention computation across time steps. The trend of the time comparison differs during decoding: when only one output token is produced at a time, RFA decodes linearly in the output length, while softmax attention decodes quadratically.

		Time Complexity			**Space Complexity**		
Setting	**Model**	**Encoder**	**Cross**	**Causal**	**Encoder**	**Cross**	**Causal**
Training w/ teacher forcing	softmax	$\mathcal{O}(M)$	$\mathcal{O}(M)$	$\mathcal{O}(N)$	$\mathcal{O}(M^2)$	$\mathcal{O}(MN)$	$\mathcal{O}(N^2)$
	RFA	$\mathcal{O}(M)$	$\mathcal{O}(M)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$	$\mathcal{O}(M+N)$	$\mathcal{O}(N)$
Decoding	softmax	$\mathcal{O}(M)$	$\mathcal{O}(MN)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M^2)$	$\mathcal{O}(MN)$	$\mathcal{O}(N^2)$
	RFA	$\mathcal{O}(M)$	$\mathcal{O}(M+N)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$	$\mathcal{O}(M+N)$	$\mathcal{O}(N)$

Table 4.1: Time and space complexity comparisons between RFA and its softmax counterpart in a sequence-to-sequence attentive model, assuming an infinite amount of available threads. M and N denote the lengths of the source and target sequences respectively. Teacher forcing training (Williams and Zipser, 1989) and autoregressive decoding are assumed. Blue color indicates the cases where RFA asymptotically outperforms softmax attention.

4.4 Experiments

We evaluate RFA on language modeling, machine translation, and long text classification.

Data	**Train**	**Dev.**	**Test**	**Vocab.**
WikiText-103	103M	218K	246K	268K
WMT14 EN-DE	4.5M	3K	3K	32K
WMT14 EN-FR	4.5M	3K	3K	32K
IWSLT14 DE-EN	160K	7K	7K	9K/7K

Table 4.2: Some statistics for the datasets. WikiText-103 split sizes are in number of tokens, while others are in number of instances.

4.4.1 Language Modeling

Setting. We experiment with WikiText-103 (Merity et al., 2017). It is based on English Wikipedia. Table 4.2 summarizes some of its statistics. We compare the following models:

- BASE is our implementation of the strong transformer-based language model by Baevski and Auli (2019).
- RFA builds on BASE, but replaces the softmax attention with random feature attention. We experiment with both Gaussian and arc-cosine kernel variants.
- RFA-GATE additionally learns a sigmoid gate on top of RFA (§4.3.2). It also has a Gaussian kernel variant and a arc-cosine kernel one.[11]
- ϕ_{elu} is a baseline to RFA. Instead of the random feature methods it uses the $\text{elu}(\cdot) + 1$ feature map, as in Katharopoulos et al. (2020).

To ensure fair comparisons, we use comparable implementations, tuning, and training procedure. All models use a 512 block size during both training and evaluation, i.e., they read as input a segment of 512 consecutive tokens, *without* access to the context from previous mini-batches. RFA variants use 64-dimensional random feature maps. We experiment with two model size settings, **small** (around 38M parameters) and **big** (around 242M parameters); they are described in Appendix A.2.1 along with other implementation details.

Results. Table 4.3 compares the models' performance in perplexity on WikiText-103 development and test data. Both kernel variants of RFA, *without* gating, outperform ϕ_{elu} by more than 2.4 and 2.1 test perplexity for the small and big model respectively, confirming the benefits from using random feature approximation.[12] Yet both *underperform* BASE, with RFA-Gaussian having a smaller gap. Comparing RFA against its gated variants, a more than 1.8 perplexity improvement can be attributed to the gating mechanism; and the gap is larger for small models. Notably, RFA-GATE-Gaussian outperforms BASE under both size settings by at least 1.2 perplexity. In gen-

[11] This gating technique is specific to RFA variants, in the sense that it is less intuitive to apply it in BASE.

[12] All models are trained for 150K steps; this could be part of the reason behind the suboptimal performance of ϕ_{elu}: it may need 3 times more gradient updates to reach similar performance to the softmax attention baseline (Katharopoulos et al., 2020).

Model	Small		Big	
	Dev.	Test	Dev.	Test
BASE	33.0	34.5	24.5	26.2
ϕ_{elu} (Katharopoulos et al., 2020)	38.4	40.1	28.7	30.2
RFA-Gaussian	33.6	35.7	25.8	27.5
RFA-arccos	36.0	37.7	26.4	28.1
RFA-GATE-Gaussian	**31.3**	**32.7**	**23.2**	**25.0**
RFA-GATE-arccos	**32.8**	**34.0**	24.8	26.3
RFA-GATE-Gaussian-Stateful	**29.4**	**30.5**	**22.0**	**23.5**

Table 4.3: Language model perplexity (lower is better) on the WikiText-103 development and test sets. Bolded numbers outperform BASE.

eral, RFA models with Gaussian feature maps outperform their arc-cosine counterparts.[13] From the analysis in §4.3.4 we would *not* expect speedup by RFA models, nor do we see any in the experiments.[14]

Closing this section, we explore a "stateful" variant of RFA-GATE-Gaussian. It passes the last hidden state $(\mathbf{S}_t, \mathbf{z}_t)$ to the next mini-batch during both training and evaluation, a technique commonly used in RNN language models (Merity et al., 2018). This is a consequence of RFA's RNN-style computation, and is less straightforward to be applicable in the vanilla transformer models.[15] From the last row of Table 4.3 we see that this brings a more than 1.5 test perplexity improvement.

4.4.2 Machine Translation

Datasets. We experiment with three standard machine translation datasets.

- WMT14 EN-DE and EN-FR (Bojar et al., 2014). Our data split and preprocessing follow those of Vaswani et al. (2017). We share the source and target vocabularies within each language pair, with 32,768 byte pair encoding types (BPE; Sennrich et al., 2016).

[13] We observe that RFA Gaussian variants are more stable and easier to train than the arc-cosine ones as well as ϕ_{elu}. We conjecture that this is because the outputs of the Gaussian feature maps have an ℓ_2-norm of 1, which can help stabilize training. To see why, $\sin^2(x) + \cos^2(x) = \cos(x - x) = 1$.

[14] In fact, RFA *trains* around 15% slower than BASE due to the additional overhead from the feature maps.

[15] Some transformer models use a text segment from the previous mini-batch as a prefix (Baevski and Auli, 2019; Dai et al., 2019). Unlike RFA, this gives the model access to only a limited amount of context, and significantly increases the memory overhead.

- IWSLT14 DE-EN (Cettolo et al., 2014) is based on TED talks. The preprocessing follows Edunov et al. (2018). Separate vocabularies of 9K/7K BPE types are used for the source and target.

Table 4.2 summarizes some statistics of the datasets.

Setting. We compare the RFA variants described in §4.4.1. They build on a BASE model that is our implementation of the base-sized transformer (Vaswani et al., 2017). All RFA models apply random feature attention in decoder cross and causal attention, but use softmax attention in encoders. This setting yields the greatest decoding time and memory savings (§4.3.4). We use 128/64 for D in cross/causal attention. RFA-GATE learns sigmoid gates in the decoder causal attention. The ϕ_{elu} baseline uses the same setting and applies feature map in both decoder cross and causal attention, but *not* in the encoders. Further details are described in Appendix A.2.2.

	WMT14		IWSLT14	
Model	**EN-DE**	**EN-FR**	**DE-EN**	**Speed**
BASE	28.1	39.0	34.6	1.0×
ϕ_{elu} (Katharopoulos et al., 2020)	21.3	34.0	29.9	2.0×
RFA-Gaussian	28.0	39.2	34.5	1.8×
RFA-arccos	28.1	38.9	34.4	1.9×
RFA-GATE-Gaussian	28.1	39.0	34.6	1.8×
RFA-GATE-arccos	28.2	39.2	34.4	1.9×

Table 4.4: Machine translation test set BLEU. The decoding speed (last column) is relative to BASE. All models are tested on a single TPU v2 accelerator, with batch size 32.

Results. Table 4.4 compares the models' test set BLEU on three machine translation datasets. Overall both Gaussian and arc-cosine variants of RFA achieve similar performance to BASE on all three datasets, significantly outperforming Katharopoulos et al. (2020). Differently from the trends in the language modeling experiments, here the gating mechanism does not lead to substantial gains. Notably, all RFA variants decode more than 1.8× faster than BASE.

Appendix A.3 studies the effect of random feature sizes on translation quality.

4.4.3 Long Text Classification

We further evaluate RFA's accuracy and efficiency when used as text encoders on three NLP tasks from the recently proposed Long Range Arena benchmark (Tay et al., 2021), designed to evaluate

efficient Transformer variants on tasks that require processing long sequences.[16]

Experimental setting and datasets. We compare RFA against baselines on the following datasets:

- ListOps (**LO**; Nangia and Bowman, 2018) aims to diagnose the capability of modelling hierarchically structured data. Given a sequence of operations on single-digit integers expressed in prefix notation, the model predicts the solution, also a single-digit integer. It is formulated as a 10-way classification. We follow Tay et al. (2021) and consider sequences with 500–2,000 symbols.

- Character-level text classification with the **IMDb** movie review dataset (Maas et al., 2011). This is a binary sentiment classification task, where the models classifies input movie reviews into positive or negative sentiments.

- Character-level document retrieval with the ACL Anthology Network (**AAN**; Radev et al., 2009) dataset. The model classifies whether there is a citation between a pair of papers.

To ensure fair comparisons, we implement RFA on top of the transformer baseline by Tay et al. (2021), and closely follow their preprocessing, data split, model size, and training procedure. Speed and memory are evaluated on the IMDb dataset. For our RFA model, we use $D = 64$ for the IMDb dataset, and $D = 128$ for others. We refer the readers to Tay et al. (2021) for further details.

Results. From Table 4.5 we can see that RFA outperforms the transformer baseline on two out of the three datasets, achieving the best performance on IMDb with 66% accuracy. Averaging across three datasets, RFA outperforms the transformer by 0.3% accuracy, second only to Zaheer et al. (2020) with a 0.1% accuracy gap. In terms of time and memory efficiency, RFA is among the strongest. RFA speeds up over the transformer by 1.1–5.3×, varying by sequence length. Importantly, compared to the only two baselines that perform comparably to the baseline transformer model (Tay et al., 2020a; Zaheer et al., 2020), RFA has a clear advantage in both speed and memory efficiency, and is the only model that is competitive in both accuracy and efficiency.

[16] https://github.com/google-research/long-range-arena

Model	Accuracy				Speed				Memory			
	LO	IMDb	AAN	Avg.	1K	2K	3K	4K	1K	2K	3K	4K
Transformer	36.4	64.3	57.5	52.7	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00
Local Attention	15.8	53.0	53.4	40.7	1.1	1.7	3.2	5.3	0.49	0.29	0.19	0.14
Wang et al. (2020b)	35.7	53.9	52.3	47.3	1.2	1.9	3.7	5.5	**0.44**	**0.21**	0.18	**0.10**
Kitaev et al. (2020)	**37.3**	56.1	53.4	48.9	0.5	0.4	0.7	0.8	0.56	0.37	0.28	0.24
Tay et al. (2020b)	17.1	63.6	**59.6**	46.8	1.1	1.6	2.9	3.8	0.55	0.31	0.20	0.16
Tay et al. (2020a)	37.0	61.7	54.7	51.1	1.1	1.2	2.9	1.4	0.76	0.75	0.74	0.74
Zaheer et al. (2020)	36.0	64.0	59.3	**53.1**	0.9	0.8	1.2	1.1	0.90	0.56	0.40	0.30
Katharopoulos et al. (2020)	16.1	65.9	53.1	45.0	1.1	**1.9**	3.7	5.6	0.44	0.22	**0.14**	0.11
Choromanski et al. (2021)	18.0	65.4	53.8	45.7	**1.2**	**1.9**	**3.8**	**5.7**	0.44	0.22	0.15	0.11
RFA-Gaussian (This chapter)	36.8	**66.0**	56.1	53.0	1.1	1.7	3.4	5.3	0.53	0.30	0.21	0.16

Table 4.5: Accuracy (higher is better) of different models on LO, IMDb, and AAN, along with their speed (higher is better) and peak memory consumption (lower is better) varying sequence lengths (1–4K). Speed and memory are evaluated on the IMDb dataset and relative to the transformer's. Bold font indicates the best performance in each column, and underlined numbers outperform the transformer in accuracy. Transformer's and previous works' numbers are due to Tay et al. (2021).

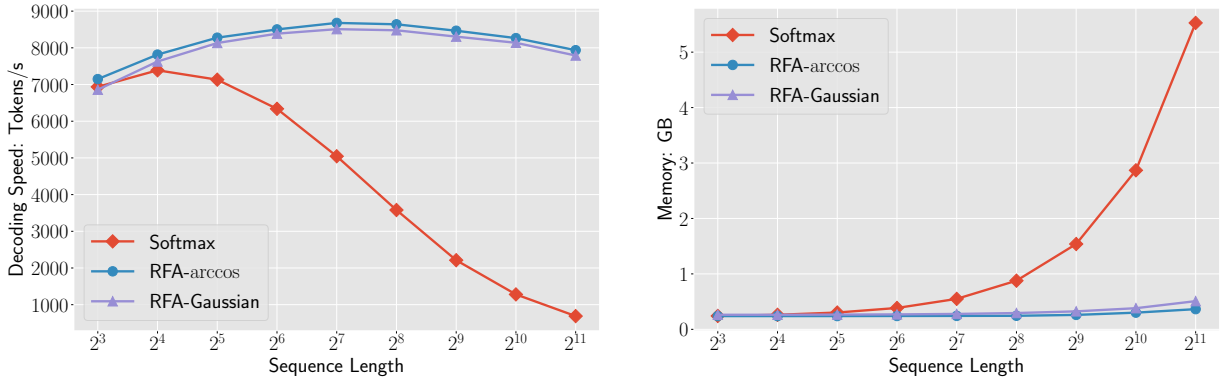
4.5 Analysis

4.5.1 Decoding time and memory varying by sequence length.

§4.3.4 shows that RFA can potentially achieve more significant speedup and memory saving for longer sequences, which we now explore.

We use a simulation conditional generation experiment on to compare RFA's sequence-to-sequence decoding speed and memory overhead against the baseline's. Here we assume the input and output sequences are of the same length. The compared models are of the same size as those described in §4.4.2, with 6-layer encoders and decoders. Other hyperparameters are summarized in §4.4.2 All models are tested using greedy decoding with the same batch size of 16, on a TPU v2 accelerator.

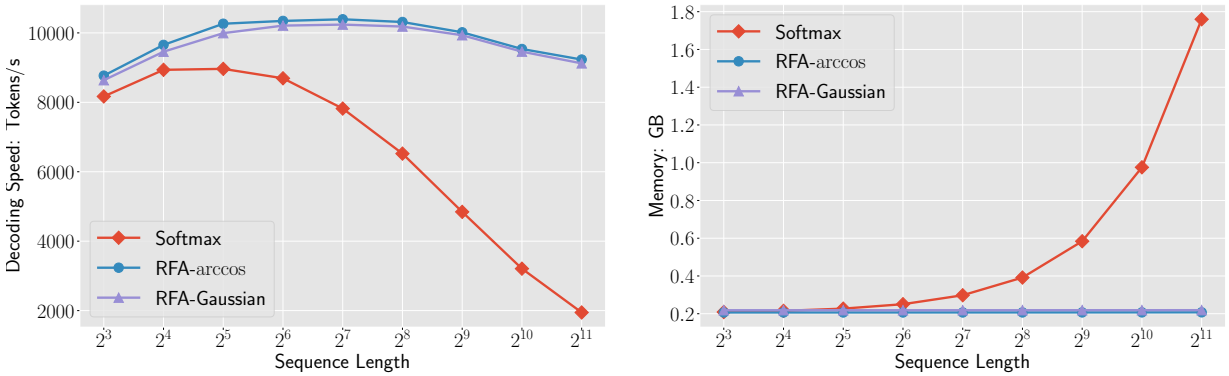
From Figures 4.2 (a) and (b) we observe clear trends. Varying the lengths, both RFA variants achieve consistent decoding speed with nearly-constant memory overhead. In contrast, the baseline decodes slower for longer sequences, taking an increasing amount of memory. Notably, for 2,048-length sequences, RFA decodes around 12× faster than the baseline while using less than 10% of the memory. RFA-arccos slightly outperforms RFA-Gaussian in terms of speed and memory efficiency. This is because when using the same D (as we do here), the ϕ_{arccos} is half the size of ϕ_{Gaussian}. These results suggest that RFA can be particularly useful in sequence-to-sequence tasks



(a) Speed vs. lengths.

(b) Memory vs. lengths.

Figure 4.2: Conditional decoding speed (left) and memory overhead (right) varying the output lengths. All models are tested on a single TPU v2 accelerator, with greedy decoding and batch size 16.



(a) Speed vs. lengths.

(b) Memory vs. lengths.

Figure 4.3: Unconditional decoding speed (left) and memory overhead (right) varying the output lengths. All models are tested on a single TPU v2 accelerator, with greedy decoding and batch size 16.

with longer sequences, e.g., document-level machine translation (Miculicich et al., 2018). Figure 4.3 compares the speed and memory consumption in *unconditional* decoding (e.g., sampling from a language model). The overall trends are similar to those in Figure 4.2.

Notes on decoding speed. With a lower memory overhead, RFA can use a larger batch size than the baseline. As noted by Katharopoulos et al. (2020) and Kasai et al. (2021a), if we had used mini-batches as large as the hardware allows, RFA could have achieved a more significant speed gain. Nonetheless, we control for batch size even though it is not the most favorable setting for RFA, since the conclusion translates better to common applications where one generates a single

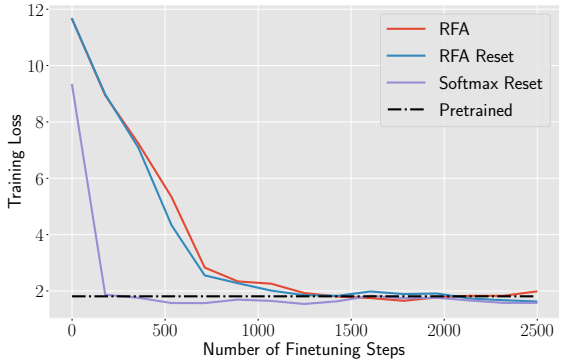


Figure 4.4: Finetuning an RFA-Gaussian model with its parameters initialized from a pretrained softmax-transformer. "Reset" indicates resetting the multihead attention parameters to randomly-initialized ones. The dashed line indicates the training loss of the pretrained model.

sequence at a time (e.g., instantaneous machine translation). For the softmax attention baseline, we follow Ott et al. (2018) and cache previously computed query/key/value representations, which significantly improves its decoding speed (over not caching).

4.5.2 Train and Evaluate with Different Attention Functions

RFA achieves comparable performance to its softmax counterpart. Does this imply that it learns a good approximation to the softmax attention? To answer this question, we consider:

(i) an RFA-Gaussian model initialized from a pretrained softmax-transformer;

(ii) a softmax-transformer initialized from a pretrained an RFA-Gaussian model.

If RFA's good performance can be attributed to learning a good approximation to softmax, both, *without* finetunining, should perform similarly to the pretrained models. However, this is *not* the case on IWSLT14 DE-EN. Both pretrained models achieve more than 35.2 development set BLEU. In contrast, (i) and (ii) respectively get 2.3 and 1.1 BLEU *without* finetuning, hardly beating a randomly-initialized untrained model (Table 4.6). This result aligns with the observation by Choromanski et al. (2021), and suggests that it is *not* the case that RFA performs well because it learns to imitate softmax attention's outputs.

		Train		
		S-attn	**NS-attn**	**Rfa**
Eval.	**S-attn**	35.5	-	0.7
	NS-attn	-	35.3	2.3
	Rfa	0.4	1.1	35.2

Table 4.6: IWSLT14 DE-EN development set BLEU performance. Entry $[i, j]$ indicates the performance of training with attention model j, but evaluating with i. **S-attn** denotes the original softmax attention; **NS-attn** is based on softmax attention, but normalizes the query and keys and adjust the temperature accordingly.

4.5.3 Knowledge Transfer from Softmax Attention to RFA

We first supplement the observation in §4.5.2 by finetuning (i) on the same pretraining data. Figure 4.4 plots the learning curves. It takes RFA roughly 1,500 steps to reach similar training loss to the pretrained model. As a baseline, "RFA Reset" resets the multihead attention parameters (i.e., those for query, key, value, and output projections) to randomly initialized ones. Its learning curve is similar to that of (i), suggesting that the pretrained multihead attention parameters are no more useful to RFA than randomly initialized ones. To further confirm this observation, "softmax Reset" resets the multihead attention parameters *without* changing the attention functions. It converges to the pretraining loss in less than 200 steps.

Takeaway. From the above results on IWSLT14, pretrained knowledge in a softmax transformer *cannot* be directly transferred to an RFA model. However, from Figure 4.4 and a much larger-scale experiment by Choromanski et al. (2021), we do observe that RFA can recover the pretraining loss, and the computation cost of finetuning is much less than training a model from scratch. This suggests some potential applications. For example, one might be able to initialize an RFA language model from a softmax transformer pretrained on large-scale data (e.g., GPT-3; Brown et al., 2020), and finetune it at a low cost. The outcome would be an RFA model retaining most of the pretraining knowledge, but is much faster and more memory-friendly to sample from. We leave such exploration to future work.

Further analysis results. RFA achieves comparable performance to softmax attention. Appendix 4.5.2 empirically shows that this *cannot* be attributed to RFA learning a good approximation to softmax: when we train with one attention but evaluate with the other, the performance is hardly better than randomly-initialized untrained models. Yet, an RFA model initialized from a

pretrained softmax transformer achieves decent training loss after a moderate amount of finetuning steps (Appendix 4.5.3). This suggests some potential applications, e.g., transferring knowledge from a pretrained transformer (e.g., GPT-3; Brown et al., 2020) to an RFA model that is more efficient to sample from.

4.6 Related Work

One common motivation across the following studies, that is shared by this chapter and the research we have already discussed, is to scale transformers to long sequences. Note that there are plenty orthogonal choices for improving efficiency such as weight sharing (Dehghani et al., 2019), quantization (Shen et al., 2020), knowledge distillation (Sanh et al., 2020), and adapters (Houlsby et al., 2019). For a detailed overview we refer the reader to (Tay et al., 2020c).

Sparse attention patterns. The idea behind these methods is to limit the reception field of attention computation. It motivates earlier attempts in improving attention's efficiency, and still receives lots of interest. The sparse patterns can be set *a priori* (Liu et al., 2018; Qiu et al., 2020; Ho et al., 2020; You et al., 2020, *inter alia*) or learned from data (Sukhbaatar et al., 2019; Roy et al., 2020, *inter alia*). For most of these approaches, it is yet to be empirically verified that they are suitable for large-scale sequence-to-sequence learning; few of them have recorded decoding speed benefits.

Compressed context. Wang et al. (2020b) compress the context along the timesteps so that the effective sequence length for attention computation is reduced. Another line of work aims to store past context into a memory module with limited size (Lee et al., 2019; Ainslie et al., 2020; Rae et al., 2020, *inter alia*), so that accessing longer history only moderately increases the overhead. Reminiscent of RNN language models, RFA attends beyond a fixed context window through a stateful computation, *without* increasing time or memory overhead.

4.7 Summary

This chapter presented random feature attention (RFA). It views the softmax attention through the lens of kernel methods, and approximates it with random feature methods. With an optional

gating mechanism, RFA provides a straightforward way of learning with recency bias. RFA's time and space complexity is linear in the sequence length. We use RFA as a drop-in substitute for softmax attention in transformer models. On language modeling, machine translation, and long text classification benchmarks, RFA achieves comparable or better performance than strong baselines. In the machine translation experiment, RFA decodes twice as fast while using half of its memory. Further time and memory efficiency improvements can be achieved for longer sequences.

Chapter 5

ABC: Attention with Bounded Memory Control

5.1 Introduction

Devising accurate and more efficient alternative contextualizations is the key to improving transformers' efficiency, and has become a focal point of the community (Tay et al., 2020c). This chapter goes beyond random feature attention (Chapter 4) and explores a more a holistic view of recent efforts in efficient attention models.

The context of attention can be seen as a random access **memory** whose size linearly grows with the sequence length; each query reads from it using a softmax-normalized linear combination. One way to improve attention's efficiency is to bound its memory size. Imposing a constant-sized constraint over the memory ensures that reading from it has constant time and space overhead, yielding a linear overall complexity in sequence lengths. This is in fact a common strategy adopted by several recent works. In this chapter, we show that some of these works are closely connected in ways that, to date, have gone unremarked. We propose **a**ttention with **b**ounded-memory **c**ontrol (ABC), a unified abstraction over them. In ABC, constant-sized memories are organized with various control strategies, e.g., induced from heuristic patterns (Beltagy et al., 2020; Zaheer et al., 2020; Ainslie et al., 2020; Rae et al., 2020, *inter alia*), locality assumptions (Parmar et al., 2018; Liu et al., 2018), or positions (Wang et al., 2020b).

The material in this chapter is adapted from Peng et al. (2022).

These strategies, by and large, are "context-agnostic." In response to this, we propose ABC$_{\text{MLP}}$, a particular instance of ABC that learns a contextualized control strategy from data. Specifically, ABC$_{\text{MLP}}$ uses a neural network to determine how to store each token into the memory (if at all). Compared to previous bounded-memory models, it strikes a better trade-off between accuracy and efficiency: controlling for the accuracy, ABC$_{\text{MLP}}$ can get away with much smaller memory sizes.

ABC models (including ABC$_{\text{MLP}}$) come with a linear complexity in sequence lengths, and admit recurrent computation graphs in causal attention (self-attention over the prefix). Therefore they are appealing choices in a variety of applications, including text encoding, language modeling and text generation. This leads to a surprising finding. Linformer (Wang et al., 2020b), an established efficient attention method, was previously thought *not* to be applicable in causal attention or autoregressive decoding (Tay et al., 2020c). Through the ABC view, we show that it actually *is*, and achieves competitive performance in our machine translation experiments.

ABC connects existing models that would otherwise seem distinct, reveals new insights into established methods, and inspires new efficient attention architectures. We explore its applications in transformers, as a drop-in substitute for the canonical softmax attention. ABC offers a novel lens that can help future research in the analysis of transformers, where the theoretical insights are still catching up with empirical success. Experiments on language modeling, machine translation, and masked language model finetuning show that our ABC$_{\text{MLP}}$ model outperforms previous ABC approaches in accuracy with a much smaller memory size. Compared to the strong transformer baseline, ABC$_{\text{MLP}}$ achieves a significant speedup and memory savings at inference time, with no or negligible accuracy loss. The efficiency improvements are more prominent for long sequences, suggesting that the asymptotic savings are even more appealing in applications involving long sequences.

5.2 An Outer-Product View of Attention

This section presents our outer-product memory perspective of attention, which allows for a smooth transition to later discussion.

In attention, a sequence of **queries** $\{\mathbf{q}_i\}_{i=1}^N$ attend to a **memory** with N slots, each storing

a **key** and **value** pair: $\mathbf{K} = [\mathbf{k}_1, \ldots, \mathbf{k}_N]^\top, \mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_N]^\top \in \mathbb{R}^{N \times d}$.[1] Query $\mathbf{q}$ reads from the memory using a softmax-normalized linear combination, producing a d-dimensional vector:

$$\text{attn}(\mathbf{q}, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) = \mathbf{V}^\top \text{softmax}(\mathbf{K}\mathbf{q}). \tag{5.1}$$

This takes $\mathcal{O}(N)$ time and space. When the attention with N queries can be parallelized (e.g., in text encoding), it takes linear time and quadratic space; when it *cannot* be (e.g., in decoding), it takes quadratic time and linear space.

The memory can be equivalently represented as sums of vector outer products: $\mathbf{K} = \mathbf{I}\mathbf{K} = \sum_{i=1}^N \mathbf{e}_i \otimes \mathbf{k}_i$, $\mathbf{V} = \sum_{i=1}^N \mathbf{e}_i \otimes \mathbf{v}_i$. $\mathbf{I}$ is the identity matrix, and $\otimes$ denotes the outer product: $[\mathbf{x} \otimes \mathbf{y}]_{i,j} = x_i y_j$. N-dimensional vectors $\{\mathbf{e}_i\}$ form the standard basis: $\mathbf{e}_i$ has the ith element being one and others zeros. We can view $\mathbf{e}_i$ as **control vectors** that determine where to store $\mathbf{k}_i$ and $\mathbf{v}_i$:

$$\begin{aligned} \mathbf{e}_i \otimes \mathbf{k}_i &= \big[\underbrace{0, \ldots 0}_{i-1}, 1, \underbrace{0, \ldots, 0}_{N-i}\big]^\top \otimes \mathbf{k}_i \\ &= \big[\ \underbrace{\mathbf{0}}_{d \times (i-1)}\ ; \mathbf{k}_i;\ \underbrace{\mathbf{0}}_{d \times (N-i)}\ \big]^\top. \end{aligned} \tag{5.2}$$

The N-by-d matrix on the last line has its ith row being $\mathbf{k}_i^\top$ and all others zeros; in this sense, $\mathbf{k}_i$ is stored in the ith slot by $\mathbf{e}_i$, not affecting others.

5.3 Attention with Bounded Memory

A straightforward way to improve attention's efficiency is to bound its memory size. Our outer-product view of attention provides a straightforward way to devise this, by replacing $\{\mathbf{e}_i\}$ with control vectors that select $n \ll N$ vectors to attend to. We dub this approach **a**ttention with **b**ounded-memory **c**ontrol (ABC). Concretely, let $\widetilde{\mathbf{K}}, \widetilde{\mathbf{V}} \in \mathbb{R}^{n \times d}$ denote a constant-size memory with n slots, with n set *a priori*.

$$\widetilde{\mathbf{K}} = \sum_{i=1}^N \boldsymbol{\phi}_i \otimes \mathbf{k}_i, \quad \widetilde{\mathbf{V}} = \sum_{i=1}^N \boldsymbol{\phi}_i \otimes \mathbf{v}_i. \tag{5.3}$$

[1] The number of queries and key-value pairs may differ, e.g., in the cross attention of a sequence-to-sequence model.

$\{\phi_i \in \mathbb{R}^n\}_{i=1}^N$ denotes a sequence of **control** vectors. The output is calculated by attending to $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$: $\textsc{Abc}\left(\mathbf{q}, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}, \{\phi_i\}\right) =$

$$\widetilde{\mathbf{V}}^\top \operatorname{softmax}\left(\widetilde{\mathbf{K}}\mathbf{q}\right). \tag{5.4}$$

We will discuss various ways to construct $\{\phi_i\}$ in the subsequent sections. Reading from the memory takes a constant $\mathcal{O}(n)$ time and space; therefore Abc's overall complexity is $\mathcal{O}(Nn)$, linear in the sequence length.[2]

Eq. 5.3 offers an equivalent recurrent computation, which is particularly useful in causal attention where only the prefix is looked at,

$$\widetilde{\mathbf{K}}_{t+1} = \widetilde{\mathbf{K}}_t + \phi_{t+1} \otimes \mathbf{k}_{t+1}, \tag{5.5}$$

likewise for $\widetilde{\mathbf{V}}_t$. $\widetilde{\mathbf{K}}_t$ and $\widetilde{\mathbf{V}}_t$ can be seen as the **recurrent hidden state** that encodes the prefix.

In what follows, we study several existing efficient attention approaches and show that they are in fact instances of the Abc abstraction.

5.3.1 Linformer

Linformer (Wang et al., 2020b) is an established efficient transformer variant that has proven successful in masked language modeling and text encoding. It assumes fixed-length inputs and learns a low-rank approximation of the attention weights. A learned n-by-N matrix $\mathbf{W}^{\text{LF}}$ down projects the N-by-d dimensional keys and values along the *timestep* dimension, to an n-by-d memory: $\widetilde{\mathbf{K}}^{\text{LF}} = \mathbf{W}^{\text{LF}}\mathbf{K}$, $\widetilde{\mathbf{V}}^{\text{LF}} = \mathbf{W}^{\text{LF}}\mathbf{V}$; they are then used for attention computation with Eq. 5.4. This yields a linear complexity in the input length. Linformer is an Abc instance with $\phi_i^{\text{LF}} = \mathbf{W}^{\text{LF}}_{:,i}$ (ith column), and in this sense, it learns a control vector for each position.

Previous works have noted that Linformer *cannot* be efficiently applied in causal attention (Table 1 of Tay et al., 2020c). Indeed, it is less straightforward to avoid mixing future with the past when projecting along the timestep dimension. Abc reveals that, in fact, Linformer *is* applicable in causal attention. Like all Abc models, it admits a linear-complexity recurrent computation

[2]Using bounded memory distinguishes Abc from softmax attention. If growing-size memory *were* allowed ($n = N$), an Abc with $\phi_i = \mathbf{e}_i$ would fall back to softmax attention.

(Eq. 5.5): $\widetilde{\mathbf{K}}^{\mathrm{LF}}_{t+1} = \widetilde{\mathbf{K}}_t + \phi^{\mathrm{LF}}_{t+1} \otimes \mathbf{k}_{t+1}$. This confirms Abc's benefits: it reveals new insights about existing models and reassesses their applications and impact. Our experiments show that Linformer achieves competitive performance in machine translation.

5.3.2 Clustering-Based Attention

Improving attention's efficiency with clustering has received an increasing amount of interest (Kitaev et al., 2020; Roy et al., 2020; Wang et al., 2020a, *inter alia*). Abc bears interesting connections to clustering-based methods. Here we discuss an approach that closely follows Vyas et al. (2020), except that it clusters keys and values instead of queries, and only attends to the centroids to reduce the effective context size. Formally, keys and values are grouped into $n < N$ clusters $\{\widetilde{\mathbf{k}}^{\mathrm{CL}}_j\}_{j=1}^n$, $\{\widetilde{\mathbf{v}}^{\mathrm{CL}}_j\}_{j=1}^n$.[3] Let an N-by-n binary matrix $\mathbf{M}$ denote the cluster membership shared between keys and values. $M_{i,j} = 1$ iff. $\mathbf{k}_i$ is assigned to cluster $\widetilde{\mathbf{k}}^{\mathrm{CL}}_j$ and $\mathbf{v}_i$ to $\widetilde{\mathbf{v}}^{\mathrm{CL}}_j$. The jth centroid for the keys is

$$\widetilde{\mathbf{k}}^{\mathrm{CL}}_j = \sum_{i=1}^{N} \frac{M_{i,j}}{\sum_{\ell=1}^{N} M_{\ell,j}} \mathbf{k}_i; \tag{5.6}$$

likewise for the values. It then attends over the centroids using Eq. 5.4, with $\widetilde{\mathbf{K}}^{\mathrm{CL}} = [\widetilde{\mathbf{k}}^{\mathrm{CL}}_1, \ldots, \widetilde{\mathbf{k}}^{\mathrm{CL}}_n]^\top =$

$$\begin{aligned}\sum_{j=1}^{n} \mathbf{e}_j \otimes \widetilde{\mathbf{k}}^{\mathrm{CL}}_j &= \sum_{j=1}^{n} \mathbf{e}_j \otimes \sum_{i=1}^{N} \frac{M_{i,j}}{\sum_{\ell=1}^{N} M_{\ell,j}} \mathbf{k}_i \\ &= \sum_{i=1}^{N} \left(\sum_{j=1}^{n} \mathbf{e}_j \frac{M_{i,j}}{\sum_{\ell=1}^{N} M_{\ell,j}} \right) \otimes \mathbf{k}_i.\end{aligned}$$

The last line indicates that this model is an instance of Abc: $\phi_i = \sum_{j=1}^{n} (M_{i,j} / \sum_{\ell=1}^{N} M_{\ell,j}) \mathbf{e}_j$. The stack of centroids can be seen as the constant-size memory. Putting aside the clustering overhead (i.e., constructing $\mathbf{M}$ and computing centroids), it has a linear complexity in the sequence length.

5.3.3 Sliding-Window Attention

In some applications, being able to remove entries from the memory can be beneficial: clearing up older context frees slots for more recent ones, promoting a locality inductive bias. Abc offers

[3] We use $\widetilde{\mathbf{k}}^{\mathrm{CL}}_j$ to denote both the jth cluster and its centroid.

the capability to do so, if augmented with an additional matrix multiplication. We use the sliding-window attention as an example.

Attending to the most recent n input tokens (Beltagy et al., 2020; Zaheer et al., 2020; Sukhbaatar et al., 2021, *inter alia*) can be seen as a first-in-first-out queue that "pops" out the oldest token while "pushing" in the most recent one: $\widetilde{\mathbf{K}}_t^{\text{WD}} = [\mathbf{k}_{t-n+1}, ..., \mathbf{k}_t]^\top$. The pop operation can be achieved by multiplying an n-by-n **upper shift matrix**: $U_{i,j} = \delta_{i+1,j}$, with δ being the Kronecker delta (i.e., $\mathbf{U}$ has ones only on the superdiagonal and zeros elsewhere). Left-multiplying $\mathbf{U}$ against $\widetilde{\mathbf{K}}_t^{\text{WD}}$ shifts its rows one position up, with zeros appearing in the last:

$$\begin{aligned}\mathbf{U}\widetilde{\mathbf{K}}_t^{\text{WD}} &= \mathbf{U}\big[\underbrace{\mathbf{k}_{t-n+1}, \ldots, \mathbf{k}_t}_{n}\big]^\top \\ &= \big[\underbrace{\mathbf{k}_{t-n+2}, \ldots, \mathbf{k}_{t-1}, \mathbf{k}_t}_{n-1}, \mathbf{0}\big]^\top \in \mathbb{R}^{n\times d}.\end{aligned}$$

Then the most recent token can be put into the slot freed up: $\widetilde{\mathbf{K}}_{t+1}^{\text{WD}} = \mathbf{U}\widetilde{\mathbf{K}}_t^{\text{WD}} + \mathbf{e}_n \otimes \mathbf{k}_{t+1}$. $\mathbf{U}$ and $\phi_t = \mathbf{e}_n$ ensure a first-in-first-out queue. Dilated and stride convolution patterns (Beltagy et al., 2020) can be similarly recovered (§5.3.6).

Recurrently multiplying $\mathbf{U}$ simulates the discrete pop operation (Grefenstette et al., 2015; Joulin and Mikolov, 2015; Yogatama et al., 2018) in a differentiable way. This is reminiscent of recurrent neural networks, while in this case $\mathbf{U}$ is *never* updated as parameters.

5.3.4 Sparse Local-to-global Attention

It sparsifies attention pattern to reduce the number of tokens that are attended to (Beltagy et al., 2020; Zaheer et al., 2020, *inter alia*). All queries attend to a subset of $n < N$ "global tokens," while ignoring others. Therefore the effective context size is reduced to n. The global tokens are usually pre-selected by positions according to some heuristics. Local-to-global attention is an instance of ABC: it can be recovered by letting $\phi_t = \mathbf{e}_i$ if x_t is the ith global token ($i = 1, \ldots, n$), and the zero vectors for others.

5.3.5 Compressive Transformer with Mean Pooling

The compressive transformer (Rae et al., 2020) explores various ways to "squash" long context into smaller and more compact representations. It achieves state-of-the-art performance on several language modeling benchmarks. We show that at least the mean-pooling variant of the compressive transformer can be seen as an Abc instance.

The mean-pooling variant of the compressive transformer compresses the context by

$$\begin{aligned}
\mathbf{K} &= \left[\mathbf{k}_1, \ldots, \mathbf{k}_N\right]^\top \in \mathbb{R}^{N \times d} \\
\rightarrow \widetilde{\mathbf{K}} &= \big[\underbrace{\left(\mathbf{k}_1 + \cdots + \mathbf{k}_c\right)}_{c} / c, \\
&\qquad \underbrace{\left(\mathbf{k}_{c+1} + \cdots + \mathbf{k}_{2c}\right)}_{c} / c \ldots, \\
&\qquad \underbrace{\left(\mathbf{k}_{N-c+1} + \cdots + \mathbf{k}_N\right)}_{c} / c \big]^\top \in \mathbb{R}^{n \times d}.
\end{aligned}$$

where $c = N/n$ is the compression ratio. Here $N \bmod n = 0$ is assumed, since otherwise the sequence can be padded to.

The above model is an Abc instance by letting

$$\phi_i = \mathbf{e}_{\lfloor (i-1)/c \rfloor + 1} / c. \tag{5.7}$$

5.3.6 Dilated Convolution Attention Patterns

The dilated attention pattern is similar to the sliding window attention and only considers the context within a predefined window. It differs in that it attends to every other token:

$$\widetilde{\mathbf{K}}_t = \left[\mathbf{k}_{t-2n+2}, \mathbf{k}_{t-2n+4}, ..., \mathbf{k}_{t-2}, \mathbf{k}_t\right]^\top. \tag{5.8}$$

It can be simulated with two separate queues $\widetilde{\mathbf{K}}^{\text{odd}}$ and $\widetilde{\mathbf{K}}^{\text{even}}$:

$$\widetilde{\mathbf{K}}_t^{\text{odd}} = \begin{cases} \mathbf{U}\widetilde{\mathbf{K}}_{t-1}^{\text{odd}} + \mathbf{e}_n \otimes \mathbf{k}_t, & \text{if } t \text{ is odd} \\ \widetilde{\mathbf{K}}_{t-1}^{\text{odd}}, & \text{otherwise} \end{cases}$$

$$\widetilde{\mathbf{K}}_t^{\text{even}} = \begin{cases} \mathbf{U}\widetilde{\mathbf{K}}_{t-1}^{\text{even}} + \mathbf{e}_n \otimes \mathbf{k}_t, & \text{if } t \text{ is even} \\ \widetilde{\mathbf{K}}_{t-1}^{\text{even}}, & \text{otherwise} \end{cases}$$

Likewise for the values. Depending on t, the query attends to one of the two queues: output $=$

$$\begin{cases} \left(\widetilde{\mathbf{V}}^{\text{odd}}\right)^\top \text{softmax}(\widetilde{\mathbf{K}}^{\text{odd}}\mathbf{q}_t), & \text{if } t \text{ is odd} \\ \left(\widetilde{\mathbf{V}}^{\text{even}}\right)^\top \text{softmax}(\widetilde{\mathbf{K}}^{\text{even}}\mathbf{q}_t), & \text{otherwise.} \end{cases}$$

The above implementation could incur considerable amount of overhead and may be actually more expensive than the the original dilated window formulation. Therefore it has more conceptual value than practical value.

5.3.7 Shared Workspace and Linear Unified Nested Attention

Shared workspace (SW; Goyal et al., 2021) and linear unified nested attention (LUNA; Ma et al., 2021) also propposed methods to learn contextualized memory control strategies. Both can be seen as instances of Abc. At layer ℓ, their ϕ_i^ℓ is a function of previous layer's memory $\widetilde{\mathbf{X}}^{\ell-1} \in \mathbb{R}^{n\times d}$ and current layer's input $\mathbf{X}^\ell \in \mathbb{R}^{N\times d}$:

$$\phi_i = \left[\text{softmax}\left(\widetilde{\mathbf{X}}^{\ell-1}\mathbf{X}^{\ell\top}\right)\right]_{:,i}, \tag{5.9}$$

where $[\cdot]_{:,i}$ denotes the ith column of a matrix. Query, key, and value projections are suppressed for notation clarity.

SW and LUNA reveal the entire sequence to the control vectors, by constructing ϕ as a function of previous layer's memory. Although both admit the recurrent computation as all Abc models do, they are ill-suited for causal attention and autoregressive decoding, since future information is "leaked" to ϕ_i from the previous layer. LUNA resorts to a variant of Katharopoulos

Model	Section	ϕ_t	Mem. Control
Sliding-window	§5.3.3	$\mathbf{e}_n$	$\widetilde{\mathbf{K}}_{t+1} = \mathbf{U}\widetilde{\mathbf{K}}_t + \phi_{t+1} \otimes \mathbf{k}_{t+1}$
Linformer	§5.3.1	$\mathbf{W}^{\text{LF}}_{:,t}$	$\widetilde{\mathbf{K}}_{t+1} = \widetilde{\mathbf{K}}_t + \phi_{t+1} \otimes \mathbf{k}_{t+1}$
L2G Pattern	§5.3.4	$\mathbf{e}_i$ if $\mathbf{x}_t$ is the ith global token	
Abc_{RD}	§5.5	$\mathbf{e}_{i_t}$, where $i_t \sim \text{unif}\{1, n\}$	
Comp. Trans.	§5.3.5	$\mathbf{e}_{\lfloor nt/N \rfloor}$	
Clustering	§5.3.2	$\sum_{j=1}^{n} \left(M_{t,j} / \sum_{\ell=1}^{N} M_{\ell,j} \right) \mathbf{e}_j$	
Abc_{MLP}	§5.4	$\mathbf{exp}\left(\mathbf{W}_\phi \mathbf{x}_t\right) / \sum_{i=1}^{t} \mathbf{exp}\left(\mathbf{W}_\phi \mathbf{x}_t\right)$	

Table 5.1: A comparison of different Abc models. N denotes the sequence length, and n the memory size. ϕ_t denotes the memory control vector for $\mathbf{k}_t$ and $\mathbf{v}_t$, and unif is the discrete uniform distribution.

	Time Complexity			Space Complexity		
Model	**Mem.**	**Per Query**	**Overall**	**Mem.**	**Per Query**	**Overall**
Softmax Attention	-	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$	-	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$
Abc	$\mathcal{O}(N)$	$\mathcal{O}(n)$	$\mathcal{O}(nN)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(nN)$

Table 5.2: Abc's time and space complexity in sequence length against the softmax attention's. "Mem." indicates the time and space needed for calculating and storing memory $\widetilde{\mathbf{K}}, \widetilde{\mathbf{V}}$. N denotes the sequence length, and n the memory size. The time complexity analysis assumes that the softmax attention *cannot* be parallelized across the queries. In practice, this is common in autoregressive decoding or for long sequences where the accelerators (e.g., GPUs) do not have enough threads to fully parallelize softmax attention's computation across different queries.

et al. (2020) in causal attention (Ma et al., 2021). In contrast, Abc_{MLP} never conditions ϕ_i on previous layer's memory, but only on the current layer's input.

Discussion. Abc provides a unified perspective of many existing efficient attention variants and at the same time points out their limitations: their control strategies are context-agnostic. In response to this, in §5.4 we propose to learn a contextualized strategy from data. Table 5.1 analyzes various Abc models.

Table 5.2 compares Abc's time and space complexity against softmax attention's. Its savings come from a per-query complexity reduction from $\mathcal{O}(N)$ to $\mathcal{O}(n)$, for both time and space. Since the memory $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$ is shared across different queries, Abc comes with a linear complexity in sequence lengths. Therefore significant efficiency improvements can be achieved when $n \ll N$.

5.4 Learned Memory Control

The ABC abstraction connects several existing approaches that would otherwise seem distinct. This inspires the design of new architectures. We hypothesize that learning a contextualized strategy can achieve better performance. This section introduces ABC_{MLP}. It parameterizes ϕ with a single-layer multi-layer perceptron (MLP) that takes as input the token's representation $\mathbf{x}_i$, and determines which slots to write it into and how much.

$$\boldsymbol{\alpha}_i = \mathbf{exp}\ (\mathbf{W}_\phi \mathbf{x}_i)\,, \quad \boldsymbol{\phi}_i = \boldsymbol{\alpha}_i \Big/ \sum_{j=1}^{N} \boldsymbol{\alpha}_j. \tag{5.10}$$

Matrix $\mathbf{W}_\phi$ is learned. $\mathbf{exp}$ is an elementwise activation function. The motivation is to allow for storing a "fractional" (but *never* negative) amount of input into the memory.[4] As we will soon show, using $\mathbf{exp}$ yields interesting connections between ABC_{MLP} and hierachical attention.

Using a non-negative activation, however, has a drawback: the scales of $\sum_i \boldsymbol{\phi}_i \otimes \mathbf{k}_i$ and $\sum_i \boldsymbol{\phi}_i \otimes \mathbf{v}_i$ would grow with the sequence lengths, making training less stable. To overcome this, we divide $\boldsymbol{\alpha}_i$ vectors by their sum. This functions as normalization and aims to offset the impact of varying sequence lengths.

In the above, encoder self-attention or cross attention is assumed, and the normalization sums over the entire sequence. Causal attention is slightly different, normalizing by the sum over the prefix instead: $\boldsymbol{\phi}_i = \boldsymbol{\alpha}_i / \sum_{j=1}^{i} \boldsymbol{\alpha}_j$. This does *not* require access to future tokens. An equivalent implementation to Eq. 5.10 is to normalize $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$ instead of $\boldsymbol{\phi}_i$ vectors:

$$\begin{aligned}
\boldsymbol{\alpha}_i &= \mathbf{exp}\ (\mathbf{W}_\phi \mathbf{x}_i)\,, \quad \boldsymbol{\phi}_i = \boldsymbol{\alpha}_i, \\
\bar{\mathbf{K}} &= \widetilde{\mathbf{K}} \Big/ \sum_{j=1}^{N} \boldsymbol{\alpha}_j. \quad \bar{\mathbf{V}} = \widetilde{\mathbf{V}} \Big/ \sum_{j=1}^{N} \boldsymbol{\alpha}_j. \\
\text{output} &= \bar{\mathbf{V}}^\top \text{softmax}(\bar{\mathbf{K}}\mathbf{q}).
\end{aligned}$$

$\mathbf{M}/\mathbf{z}$ divides the ℓth row of matrix $\mathbf{M}$ by vector $\mathbf{z}$'s ℓth dimension. It admits the recurrent computation graph as in Eq. 5.5, and has a linear complexity in the sequence length.

A key design choice of ABC_{MLP} is that its $\boldsymbol{\phi}_i$ depends *only* on current input $\mathbf{x}_i$. This helps (1)

[4] We experiment with other activations in §5.5.2.

keep the recurrent computation efficient in practice (Lei et al., 2018), and (2) make it applicable in not only encoder self-attention and cross attention, but also causal attention. Goyal et al. (2021) and Ma et al. (2021) also proposed methods to learn contextualized control. They compute ϕ_i from *previous* layer's memory, revealing the full sequence to the control vectors. As a result, these two approaches are *unsuitable* for causal attention.[5]

ABC$_{\text{MLP}}$, as other ABC models, can be used as a drop-in replacement for the canonical softmax attention, and we apply its multihead variant in transformers. It is applicable in different scenarios (including encoding self-attention, cross attention, and causal attention), *and* achieves strong performance on all (§5.5). With proper parameter sharing, the number of additional parameters ABC$_{\text{MLP}}$ incurs is small: inspired by Wang et al. (2020b), we tie ϕ-MLP's parameters across different layers, which adds less than 1% parameters to the models. This sharing is inspired by the both best-performing and most parameter-efficient configuration by Wang et al. (2020b).

Abc$_{\text{MLP}}$: context-agnostic then context-dependent attention. We now dissect ABC$_{\text{MLP}}$ and show that it can be seen as a cascade of two attention mechanisms: one with a learned context-agnostic "pseudo query" followed by one with a context-dependent query. Our analysis starts with a one-dimensional example; the conclusion generalizes to higher-dimensional cases.

Example 2. Consider ABC$_{\text{MLP}}$ with a *single* memory slot ($n = 1$). It is parameterized with a learned vector $\mathbf{w}_\phi$, and $\phi_i = \exp(\mathbf{w}_\phi \cdot \mathbf{x}_i) / \sum_{j=1}^{N} \exp(\mathbf{w}_\phi \cdot \mathbf{x}_j)$. Since ϕ_i is a scalar here, $\phi_i \otimes \mathbf{k}_i = \phi_i \mathbf{k}_i^\top$.

$$\begin{aligned}
\widetilde{\mathbf{K}}^\top &= \sum_{i=1}^{N} (\phi_i \otimes \mathbf{k}_i)^\top \\
&= \sum_{i=1}^{N} \frac{\exp(\mathbf{w}_\phi \cdot \mathbf{x}_i)}{\sum_{j=1}^{N} \exp(\mathbf{w}_\phi \cdot \mathbf{x}_j)} \mathbf{k}_i \\
&= \text{attn}\left(\mathbf{w}_\phi, \{\mathbf{x}_i\}_{i=1}^{N}, \{\mathbf{k}_i\}_{i=1}^{N}\right).
\end{aligned}$$

In other words, $\widetilde{\mathbf{K}}$ uses $\mathbf{w}_\phi$ as a "pseudo-query" to attend to $\{\mathbf{x}_i\}$ and $\{\mathbf{k}_i\}$. Likewise, $\widetilde{\mathbf{V}}^\top = \text{attn}(\mathbf{w}_\phi, \{\mathbf{x}_i\}_{i=1}^{N}, \{\mathbf{v}_i\}_{i=1}^{N})$. Despite its similarity to the standard softmax attention, Example 2 has a more efficient linear complexity in sequence lengths. $\mathbf{w}_\phi$'s being context-independent is the

[5] Both are instances of ABC. Ma et al. (2021) resorts to a variant of Katharopoulos et al. (2020) for causal attention.

key to the savings.

Example 2's conclusion generalizes to higher-dimensional cases: Assume that the constant-sized memory has n slots. $\boldsymbol{\phi}_i$ is cauculated as in Eq. 5.10. Then $\widetilde{\mathbf{K}} = \sum_{i=1}^{N} \boldsymbol{\phi}_i \otimes \mathbf{k}_i \in \mathbb{R}^{n \times d}$. Each row of $\widetilde{\mathbf{K}}$ can be seen as a separate attention mechanism with a pseudo query. Let $[\cdot]_\ell$ denote the ℓth row/dimension of a matrix/vector. Then for any $\ell = 1, \ldots, n$,

$$
\begin{aligned}
\left[\widetilde{\mathbf{K}}\right]_\ell &= \sum_{i=1}^{N} [\boldsymbol{\phi}_i]_\ell \otimes \mathbf{k}_i \\
&= \sum_{i=1}^{N} \frac{\exp([\mathbf{W}_\phi]_\ell \cdot \mathbf{x}_i)}{\sum_{j=1}^{N} \exp([\mathbf{W}_\phi]_\ell \cdot \mathbf{x}_j)} \mathbf{k}_i^\top \\
&= \operatorname{attn}\left([\mathbf{W}_\phi]_\ell, \{\mathbf{x}_i\}_{i=1}^{N}, \{\mathbf{k}_i\}_{i=1}^{N}\right)^\top \in \mathbb{R}^{1 \times d}.
\end{aligned}
$$

In other words, there are n attention mechanisms in total, each with a separately-parameterized pseudo-query $[\mathbf{W}_\phi]_\ell$. They summarize the context for n times in parallel, each producing a d-dimensional vectors. These output vectors are then stacked into n-by-d memory $\widetilde{\mathbf{K}}$. $\widetilde{\mathbf{V}}$ is similar.

Intuitively, it is the "real queries" $\{\mathbf{q}_i\}$ that encode "what information is useful for the prediction task." Without access to them, Abc_{MLP} summarizes the input for n times using different pseudo-queries, aiming to preserve enough information in the memory for onward computation. The attention output is calculated with the context-dependent real queries using Eq. 5.4.

Connections to other prior works. Although starting from distinct motivations, Abc_{MLP} closely relates to hierarchical attention (HA; Yang et al., 2016). HA summarizes the context into higher-level representations with a cascade of attention mechanisms, e.g., words to sentences, and then to documents. Abc_{MLP} applies two types of attention. The first learns context-agnostic pseudo-queries and attends to the same sequence for n times in parallel, while the second retrieves from the memory with real queries. HA, in contrast, summarizes non-overlapping segments at each level.

The learned pseudo-queries closely relate to the inducing point method in set attention (ISA; Lee et al., 2019). ISA applies a non-linear feedforward network between a cascade of two attention modules. This precludes the outer-product memory computation and efficient recurrences in Abc.

Another line of work "linearizes" attention through kernel tricks and also applies bounded memory: their feature map dimensions are analogous to memory sizes. They substitute the softmax with approximations (Choromanski et al., 2021), heuristically designed (Katharopoulos et al., 2020; Schlag et al., 2021), or learned (Kasai et al., 2021b) functions. ABC$_{\text{MLP}}$ keeps the softmax, but over a smaller constant-sized context. This can be useful in practice: (1) ABC provides a unified perspective of several efficient attention methods, allowing for borrowing from existing wisdom to design new architectures; (2) it draws a close analogy to the canonical softmax attention, and is better-suited as its drop-in substitute in various application settings, as we will show in the experiments; (3) empirically, we find that ABC$_{\text{MLP}}$ can get away with a much smaller memory size to retain the accuracy. In our machine translation experiment, an ABC$_{\text{MLP}}$ with 32-8 cross-causal attention memory sizes matches transformer's accuracy, while RFA needs 256-128. RFA and Schlag et al. (2021) use gating to promote recency bias. The same technique is equally applicable in ABC models.

The learned contextualized memory control is reminiscent of the content-based addressing in neural Turing machines (NTM; Graves et al., 2014). ABC$_{\text{MLP}}$ computes the control vectors $\{\phi_i\}$ as a function of the input, but *not* of the memory as in NTM. This ensures that the control vectors at different timesteps can be computed in parallel, improving the time efficiency in practice (Lei et al., 2018; Peng et al., 2018a). Analogies between memory and neural architectures are also made by other previous works (Hochreiter and Schmidhuber, 1997; Weston et al., 2015; Le et al., 2020, *inter alia*).

5.5 Experiments

We evaluate ABC models on language modeling (§5.5.1), sentence-level and document-level machine translation (§5.5.2), and masked language model finetuning (§5.5.3).

5.5.1 Language Modeling

Setting. We experiment with WikiText-103, sampled text from English Wikipedia (Merity et al., 2017). Table 5.3 summarizes some statistics of this dataset. The BASE model with standard softmax attention is the strong transformer-based language model by Baevski and Auli (2019).

Data	Train	Dev.	Test	Vocab.	Sent./doc
WikiText-103	103M	218K	246K	268K	-
WMT14 EN-DE	4.5M	3K	3K	32K	-
IWSLT14 ES-EN	1713	8	56	30K	121.5

Table 5.3: Statistics for the datasets. WikiText-103 split sizes are in number of tokens, WMT14 in number of sentences, and IWSLT14 in number of documents.

We compare the following ABC variants, which build on BASE, but replace the softmax attention with linear-complexity bounded-memory attention alternatives while keeping other components the same. For ABC_{MLP} and Linformer, we share the ϕ functions' parameters across different layers, and further share Linformer's between attention heads. This follows the best-performing configuration by Wang et al. (2020b).

- ABC_{MLP}, as described in §5.4, learns a contextualized exp-MLP as the ϕ function.
- Linformer (§5.3.1; Wang et al., 2020b).
- ABC_{RD} stores each token in a randomly-selected memory slot.This is achieved by letting $\phi_t = \mathbf{e}_{i_t}$, where i_t is uniformly drawn from $\{1, \ldots, n\}$ for each t. It is designed as a baseline to ABC_{MLP} and Linformer to quantify the differences between random and learned bounded-memory control. Random sparse attention patterns are explored by Zaheer et al. (2020), where a subset of $n < N$ tokens are randomly selected to be attended to by all tokens. ABC_{RD} is different, and it attends to *all* tokens, but randomly "squash" them into an n-slot memory. This helps us quantify the differences between random and learned bounded-memory controls.

We consider two model size settings:

- 16 layers (Baevski and Auli, 2019). All models have around ∼242M parameters. They train with 512-token segments, and evaluate with 0 or 480 context sizes: a 0- or 480- length prefix precedes each evaluation segment.
- 32 layers (Kasai et al., 2021b). All models have ∼484M parameters. This setting applies layer dropout (Fan et al., 2020), and evaluates with a 256 context size. It aims to com-

Model	n	Dev. 0	Dev. 480	Test 0	Test 480
Base	-	19.8	18.4	20.5	19.0
Linformer	32	25.3	27.0	26.3	28.4
Linformer	64	26.5	27.1	27.2	30.7
Abc$_{\text{RD}}$	32	24.5	23.6	26.5	24.6
Abc$_{\text{RD}}$	64	23.2	22.3	24.0	23.1
Abc$_{\text{MLP}}$	32	21.2	19.7	21.9	20.5
Abc$_{\text{MLP}}$	64	**20.4**	**18.9**	**21.1**	**19.5**

(a) 16-layer setting. 0/480 indicate evaluation context sizes.

Model	n	Dev.	Test
†Base	-	17.9	18.5
†ELU	128	22.0	22.8
†RFA	32	20.4	21.3
†T2R	32	20.1	20.8
Abc$_{\text{MLP}}$	32	**19.2**	**19.9**

(b) 32-layer setting. A 256-length context is used at evaluation time. † numbers are due to Kasai et al. (2021b).

Table 5.4: WikiText-103 language modeling perplexity (**lower is better**). n denotes the memory size. Bold numbers perform the best among linear-complexity models.

pare Abc$_{\text{MLP}}$ to several kernel-based efficient attention variants: ELU (Katharopoulos et al., 2020), RFA (Chapter 4), and T2R (Kasai et al., 2021b).

We use comparable implementations, training, and tuning for all models, to ensure fair comparisons. Further implementation details are described in Appendix B.1.

Results. Table 5.4a compares Abc variants using Baevski and Auli (2019)'s 16-layer setting. Among Abc models, Abc$_{\text{MLP}}$ achieves the best performance for both context sizes. With a memory size $n = 64$, Abc$_{\text{MLP}}$ outperforms both Linformer and Abc$_{\text{RD}}$ by more than 2.9 test perplexity; and the gap is larger with the longer 480-length context: more than 3.6 test perplexity. Abc$_{\text{MLP}}$-32 outperforms its larger-memory Abc counterparts by more than 2.1 test perplexity. These results confirm Abc$_{\text{MLP}}$'s advantages of using a contextualized strategy. Surprisingly, Linformer *underperforms* Abc$_{\text{RD}}$, and its performance drops with the larger 480-length context window. This suggests that, while successful in text encoding, Linformer's position-based strategy is a suboptimal design choice for causal attention, at least for long context. All Abc models *underperform* the Base, with Abc$_{\text{MLP}}$-64 having the smallest gap of 0.5 perplexity. Abc$_{\text{MLP}}$-32 outperforms kernel-based methods by more than 0.9 test perplexity, using Kasai et al. (2021b)'s 32-layer setting (Table 5.4b).

Model	Cross n	Causal n	BLEU
Base	-	-	27.2
Abc$_{\text{RD}}$	32	32	25.7
Abc$_{\text{RD}}$	64	64	26.2
Linformer	32	32	26.6
Linformer	64	64	26.7
Abc$_{\text{MLP}}$	32	8	27.1
Abc$_{\text{MLP}}$	32	32	**27.3**

(a) Bolded number outperforms Base.

Model	Cross n	Causal n	BLEU
Base	-	-	39.9
Linformer	128	64	-
Abc$_{\text{RD}}$	128	64	38.6
Abc$_{\text{MLP}}$	128	64	**39.7**

(b) Linformer fails to converge even with multiple random seeds. Bold number performs the best among Abc models.

Table 5.5: Machine translation test SacreBLEU. Left: sentence-level translation with WMT14 EN-DE; right: document-level translation with IWSLT14 ES-EN.

5.5.2 Machine Translation

Datasets. To assess their performance over various output lengths, we compare Abc models on sentence- and document- level machine translation. See Table 5.3 for some statistics of the datasets.

- Sentence-level translation with WMT14 EN-DE (Bojar et al., 2014). The preprocessing and data splits follow Vaswani et al. (2017). A 32,768 byte pair encoding (BPE; Sennrich et al., 2016) vocabulary is shared between source and target languages.
- Document-level translation with IWSLT14 ES-EN (Cettolo et al., 2014). We follow Miculicich et al. (2018) and use the *dev2010* subset for development and *tst2010-2012* for testing. The tokenization is also the same as Miculicich et al. (2018): we tokenize and truecase Spanish and English with Moses (Koehn et al., 2007) and run byte-pair encoding with 30k splits, shared between the two languages. The final dataset contains 1421, 8, and 42 documents for training, development, and testing. On average, each document contains 126.7 sentences, and each sentence contains 21.7(ES)/22.5(EN) BPE subwords. We use a sliding window with length-4 and stride-one to generate our dataset. During inference, we use predicted context on the target side.

Setting. We compare Abc variants as in §5.5.1. §5.6.2 further compares to the clustering-based (§5.3.2) and sliding-window (§5.3.3) Abc variants.

We compare the Abc variants described in §5.5.1. The Base model they build on is our implementation of transformer-base (Vaswani et al., 2017). Abc variants replace decoder cross attention and causal attention with bounded-memory attention, while keeping softmax attention for the encoder, since its overhead is much less significant (Kasai et al., 2021a); other components are kept the same. §5.6.2 studies a model that replaces *all* softmax attention with Abc_{MLP}. It performs on par with Base, confirming Abc_{MLP}'s broad applicability in various application scenarios. We evaluate with SacreBLEU (Post, 2018). Further details are described in Appendix B.1

Results. Table 5.5a summarizes sentence-level machine translation results on the WMT14 EN-DE test set. Overall Abc_{MLP} performs on par with Base, with either 32-32 cross-causal memory sizes or 32-8. Even with smaller memory sizes, it outperforms other Abc variants by more than 1.1 BLEU. Differently from the trend in the language modeling experiment (§5.5.1), Linformer outperforms Abc_{RD} by more than 0.5 BLEU. We attribute this to the smaller sequence lengths of this dataset. Abc_{MLP} outperforms other Abc models by more than 0.4 BLEU, even with smaller memory sizes.

The trend is similar on document-level translation with IWSLT14 ES-EN (Table 5.5b), except that Abc_{MLP} slightly *underperforms* Base by 0.2 BLEU. This suggests that even with longer sequences, Abc_{MLP} is effective despite its bounded memory size. Linformer fails to converge even with multiple random seeds, suggesting the limitations of its purely position-based strategy in tasks involving decoding varying-length text. Linformer suffers severely in longer sequence generation: it fails to converge even with multiple random seeds.

5.5.3 Masked Language Model Finetuning

Setting. We compare the Abc variants as in §5.5.1. It is interesting to pretrain Abc from scratch, but we lack the resources to do so. Instead, we warm-start from a pretrained RoBERTa-base (Liu et al., 2019) trained with the softmax transformer, swap its attention with Abc variants, and continue pretraining with the masked language modeling (MLM) objective on a concatenation of BookCorpus (Zhu et al., 2015), English Wikipedia, OpenWebText (Gokaslan and Cohen, 2019),

and RealNews (Zellers et al., 2019).[6] Our machine does *not* have a large enough memory to load all the data. We therefore split the training data into 20 shards, after shuffling. Other preprocessing is the same as Liu et al. (2019).[7] The hyperparameters for continued pretraining follow base-sized RoBERTa, part of which are summarized in Appendix B.1 All models are trained on a single TPU v3 accelerator, for 50K steps (with batch size 128, that is about half an epoch), due to limited computational resources. Then the models are finetuned and evaluated on downstream classification datasets from the the GLUE benchbark (Wang et al., 2019). This is an appealing setting, since it avoids reinvesting the huge amounts of resources already put into pretraining.[8] For downstream task finetuning, we use the same hyperparameters as Liu et al. (2019).[9] Table B.4 briefly describes the tasks. The readers are referred to Wang et al. (2019) for further details.

Results. Table 5.6 compares downstream text classification performance. Base indicates a baseline that continues pretraining RoBERTa-base on our data.[10] Following standard practice, we report development accuracy. Linformer achieves competitive performance, aligned with Wang et al. (2020b)'s results. Abc$_{\text{MLP}}$ outperforms Linformer, and performs on par with or better than Base, affirming the benefits of using contextualized memory organization in MLM. Abc$_{\text{RD}}$ fails to converge in continued pretraining even with multiple seeds.

Based on the above results, we think Abc$_{\text{MLP}}$ can achieve competitive performance when pretrained from scratch, just as Linformer does (Wang et al., 2020b). Further empirical exploration is beyond our budget and left for future work.

[6] Our data differs from RoBERTa's, which we do *not* have access to. We replace CC-News (Nagel, 2016) with RealNews, and drop Stories (Trinh and Le, 2018), whose public access is broken at the time of this work: https://console.cloud.google.com/storage/browser/commonsense-reasoning/reproduce/stories_corpus?pli=1.

[7] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.pretraining.md

[8] In preliminary experiments, we explored swapping in Abc, and then directly finetuning on downstream tasks *without* continued MLM pretraining; all models fail.

[9] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.glue.md

[10] Base slightly *underperforms* RoBERTa-base. This could be due to overfitting, or the pretraining data discrepancy.

Model	n	MNLI	QNLI	QQP	SST	Avg.
Base	-	87.2	92.4	91.7	94.3	91.4
Linformer	64	85.3	91.8	90.8	92.4	90.1
Linformer	128	86.1	91.9	91.4	93.7	90.8
Abc$_{\text{MLP}}$	64	85.6	91.8	91.7	93.8	90.7
Abc$_{\text{MLP}}$	128	**87.1**	**92.6**	**91.8**	**94.4**	**91.5**

Table 5.6: Text classification development set accuracy. All models continue pretraining RoBERTa-base on our data with the MLM objective. Bold numbers perform the best among Abc models, and underlined ones perform on par with or better than Base.

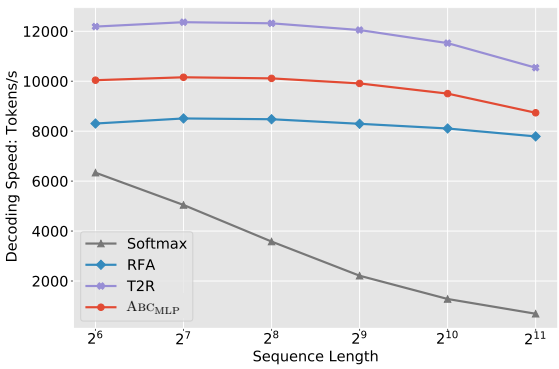
5.6 Analysis

5.6.1 Decoding efficiency over varying sequence lengths.

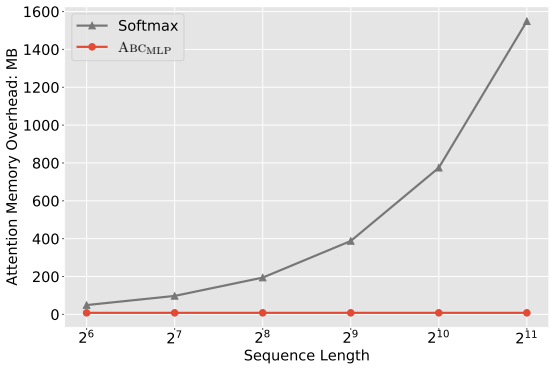
Abc's efficiency gains can be more prominent for long sequences. We study Abc$_{\text{MLP}}$'s decoding overhead with varying sequence lengths. Following Kasai et al. (2021b), we consider a sequence-to-sequence generation experiment. Three linear-complexity models are compared: RFA (with 256/128 cross/causal memory sizes; Chapter 4), T2R (32/4; Kasai et al., 2021b), and Abc$_{\text{MLP}}$ (32/8). The sizes are chosen to maximize efficiency *without* accuracy drop. T2R needs to be finetuned from a pretrained transformer to match its performance, while others don't.

All linear-time models achieve consistent decoding speed for different lengths (Figure 5.1a), substantially outpacing the softmax attention baseline, especially for long sequences. In particular, Abc$_{\text{MLP}}$ decodes $\sim$1.25 times faster than RFA, another competitive model that can match transformer's accuracy *without* a warm start from a pretrained model. This can be attributed to the fact that Abc$_{\text{MLP}}$ achieves similar accuracy with a much smaller memory. T2R's memory sizes are similar to Abc$_{\text{MLP}}$'s, but it decodes about 20% faster. This is because it does *not* compute the softmax when calculating attention output, while Abc$_{\text{MLP}}$ does (Eq. 5.4). These results show that Abc$_{\text{MLP}}$ is an appealing modeling choice for decoding tasks, especially when training from scratch is desired.

Abc$_{\text{MLP}}$ also achieves significant savings in terms of memory overhead. Figure 5.1b compares Abc$_{\text{MLP}}$'s (32-8 memory sizes) attention memory overhead with softmax attention's. Following Kasai et al. (2021b), we consider a synthetic sequence-to-sequence generation task with varying sequence lengths. A batch size of 16 and greedy decoding is used. The models are of the same



(a) Decoding Speed.



(b) Decoding memory overhead.

Figure 5.1: Sequence-to-sequence decoding speed (top) and memory consumption (bottom) varying sequence lengths. Greedy decoding is used, with batch size 16.

Model	ϕ	Cross n	Causal n	Encoder n	BLEU
BASE	-	-	-	-	27.2
ABC	Window	32	32	-	26.3
	Cluster	32	32	-	26.8
	MLP-ReLU	32	8	-	-
	MLP-ReLU	32	32	-	26.4
	MLP-sigmoid	32	8	-	26.8
	MLP-sigmoid	32	32	-	27.0
	MLP-exp	32	8	-	27.1
	MLP-exp	32	32	-	**27.3**
	MLP-exp-all	32	32	32	27.0

Table 5.7: ABC variants' performance (SacreBLEU) on the WMT14 EN-DE test set for sentence-level machine translation. MLP-ReLU with 32/8 memory sizes fails to converge. MLP-exp-all applies ABC in both the encoder and the decoder, while others only in the decoders.

size as those in §5.5.2. ABC_{MLP}, RFA, and T2R's curves are similar.

5.6.2 Additional machine translation results.

In addition to the results presented in §5.5.2, Table 5.7 further compares, on the WMT14 EN-DE dataset, the clustering-based (§5.3.2) and sliding-window (§5.3.3) models of ABC, as well as ReLU and sigmoid variants of ABC_{MLP}. Clustering and sliding-window ABC variants *underperform* ABC_{MLP} with the same memory sizes by more than 0.5 BLEU. Both ReLU and sigmoid *underperform* their exp counterpart. We observe that ABC_{MLP}-ReLU suffers a severe "the rich gets

richer" issue: all tokens are stored in a handful of slots, no matter how large the memory size is. This could be the reason for its suboptimal accuracy. Further investigations are deferred to future work.

MLP-exp-all replaces the encoder's softmax attention modules with ABC, in addition to the decoder's. It underperforms ABC_{MLP} by only 0.3 BLEU.

5.6.3 Text encoding efficiency.

We compare the efficiency of ABC_{MLP} against softmax attention and Linformer when used as text encoders. The models' sizes mirror those in the MLM experiment (§5.5.3). Table 5.8 summarizes inference time and memory overhead with 512-length inputs, batch size 16. Both ABC_{MLP} and Linformer achieve inference speed gains and memory savings over BASE. Linformer is faster, since its linear projection is cheaper to compute than ABC_{MLP}'s MLP. Inference speed is measured on the same V100 GPU. The trend in memory overhead is similar.

Although ABC_{MLP} slightly underperforms Linformer in terms of inference speed, it can be a more appealing architectural choice in practice: in all of our 5 experiments, ABC_{MLP} outperforms other ABC models in accuracy. Linformer, in contrast, fails to converge or yields sub-optimal performance on some tasks. This confirms its flexibility and applicability in various settings.

5.6.4 Memory size's impact on accuracy.

Practically, one may want to minimize the memory size to improve efficiency. We use the WMT14 EN-DE experiment to investigate how memory size affects accuracy. Using the §5.5.2's setup, we vary ABC_{MLP}'s cross and causal attention memory sizes and compare their translation quality on the development data. They are selected from $\{8, 16, 32, 64\}$, with cross attention's equal to or larger than causal's: cross attention is more important than causal attention in machine translation (Michel et al., 2019). Our results (Table 5.9) align with this observation: when cross attention memory is large enough, reducing causal attention memory size from 64 to 8 has a minor 0.3 BLEU drop. Surprisingly, ABC_{MLP} with 8-8 sized cross-causal memory is only 1.1 BLEU behind the best-performing configuration. Here we apply greedy decoding *without* checkpoint averaging.

	BASE	Linformer		ABC$_{\text{MLP}}$	
n	-	64	128	64	128
Speed	1.0×	1.7×	1.5×	1.5×	1.3×
Memory	1.0×	0.5×	0.6×	0.5×	0.6×

Table 5.8: Text encoding inference speed (higher is better) and memory (lower is better). Inputs are text segments with 512 tokens and batch size 16.

		Cross n			
		8	16	32	64
Causal n	8	24.7	25.2	25.6	25.5
	16	-	25.4	25.7	25.6
	32	-	-	25.7	25.8
	64	-	-	-	25.8

Table 5.9: ABC$_{\text{MLP}}$'s SacreBLEU on WMT14 EN-DE development data varying memory sizes. All models apply greedy decoding, *without* checkpoint averaging.

5.7 Summary

We presented attention with bounded-memory control (ABC). It provides a unified perspective of several recently-proposed models, and shows that they vary in the organization of the bounded memory. ABC reveals new insights into established methods and inspires new architectures. We proposed ABC$_{\text{MLP}}$, a particular instance of ABC that learns a contextualized memory control. On language modeling, machine translation, and masked language model finetuning, ABC$_{\text{MLP}}$ outperforms previous ABC models. Compared to the strong transformer baseline, ABC$_{\text{MLP}}$ achieves substantial efficiency improvements with no or negligible accuracy loss.

Chapter 6

Rational Recurrences

Despite the tremendous empirical success of neural models in natural language processing, our understanding of deep learning models lags behind. This chapter aims to ground modern neural architectures to formal principles in NLP, and uses the insights to develop better-performing, more efficient, and more interpretable models. In particular, we show that some *recurrent* neural networks also share this connection to weighted finite-state automata (WFSAs). We characterize this connection formally, defining **rational recurrences** to be recurrent hidden state update functions that can be written as the Forward calculation of a finite set of WFSAs. We show that several recent neural models use rational recurrences. Our analysis provides a fresh view of these models and facilitates devising new neural architectures that draw inspiration from WFSAs. We present one such model, which performs better than two recent baselines on language modeling and text classification. Our results demonstrate that transferring intuitions from classical models like WFSAs can be an effective approach to designing and understanding neural models.

6.1 Introduction

Neural models, and in particular gated variants of recurrent neural networks (RNNs, e.g., Hochreiter and Schmidhuber, 1997; Cho et al., 2014), have become a core building block for state-of-the-art approaches in NLP (Goldberg, 2016). While these models empirically outperform classical NLP methods on many tasks (Zaremba et al., 2014; Bahdanau et al., 2015; Dyer et al., 2016; Peng

The material in this chapter is adapted from Peng et al. (2018a).

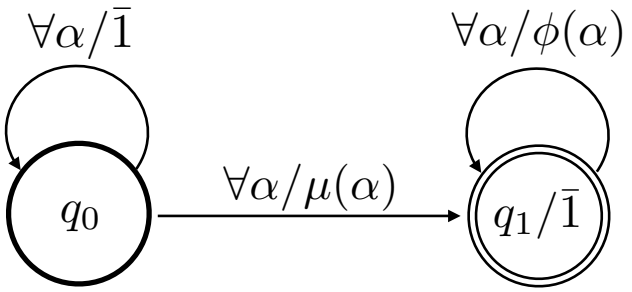


Figure 6.1: A two-state WFSA $\mathcal{B}$ described in 6.2. It is closely related to several models studied in this chapter (6.4.1). Bold circles indicate initial states, and double circles final states, which are associated with final weights. Arrows represent transitions, labeled by the symbols α they consume, and the weights as a function of α. Arcs not drawn are assumed to have weight $\bar{0}$. For brevity, $\forall\alpha$ means $\forall\alpha \in \Sigma$, with Σ being the alphabet.

et al., 2017, *inter alia*), they typically lack the intuition offered by classical models, making it hard to understand the roles played by each of their components. In this chapter we show that many neural models are more interpretable than previously thought, by drawing connections to weighted finite state automata (WFSAs). We study several recently proposed RNN architectures and show that one can use WFSAs to characterize their recurrent updates. We call such models **rational recurrences** (§6.3).[1] Analyzing recurrences in terms of WFSAs provides a new view of existing models and facilitates the development of new ones.

In recent work, Schwartz et al. (2018) introduced SoPa, an RNN constructed from WFSAs, and thus rational by our definition. They also showed that a single-layer max-pooled CNN (LeCun, 1998) can be simulated by a set of simple WFSAs (one per output dimension), and accordingly are also rational. In this chapter we broaden such efforts, and show that rational recurrences are in frequent use (Mikolov et al., 2014; Balduzzi and Ghifary, 2016; Lei et al., 2016, 2017a,b; Bradbury et al., 2017; Foerster et al., 2017). For instance, we will show in §6.4 that the WFSA diagrammed in Figure 6.1 has strong connections to several of the models mentioned above.

Based on these observations, we then discuss potential approaches to deriving novel neural architectures from WFSAs (§6.5). As a case study, we present a new model motivated by the interpolation of a two-state WFSA and a three-state one, capturing (soft) unigram and bigram features, respectively. Our experiments show that in two tasks—language modeling and text classification—the proposed model outperforms recently proposed rational models (§6.6). Fur-

[1] Where the term *regular* is used with unweighted FSAs (e.g., regular languages, regular expressions), *rational* is the weighted analog (e.g., rational series, Sakarovitch, 2009; rational kernels, Cortes et al., 2004).

ther extensions might lead to larger gains, and the rational recurrence view could facilitate easier exploration of such extensions.

6.2 Background: Weighted Finite State Automata (WFSAs)

This section reviews weighted finite-state automata and semirings, which underly our analyses in §6.3. WFSAs extend nondeterministic *unweighted* finite-state automata by assigning weights to transitions, start states, and final states. Instead of simply accepting or rejecting a string, a WFSA returns a score for the string, and this score summarizes the weights along all paths through the WFSA that consume the string. In order for this summary score to be efficiently computable, weights are taken from a **semiring**.

Definition 3 (Kuich and Salomaa, 1986)**.** A **semiring** is a set $\mathbb{K}$ along with two associative binary operations on $\mathbb{K}$, $\oplus$ (addition) and $\otimes$ (multiplication), and two identity elements: $\bar{0}$ for addition, and $\bar{1}$ for multiplication. Semirings also require that addition is commutative, multiplication distributes over addition, and that multiplication by $\bar{0}$ annihilates (i.e., $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$).

One common semiring is the **real** (or plus-times) semiring: $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$. The other one used in this chapter is the **max-plus** semiring $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$. We refer the reader to Kuich and Salomaa (1986) for others.

Definition 4. A **weighted finite-state automaton (WFSA)** over a semiring $\mathbb{K}$ is a 5-tuple, $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \tau, \lambda, \rho \rangle$,[2] with:

- a finite input alphabet Σ;
- a finite state set $\mathcal{Q}$;
- transition weights $\tau : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{K}$;
- initial weights $\lambda : \mathcal{Q} \to \mathbb{K}$;
- and final weights $\rho : \mathcal{Q} \to \mathbb{K}$.

$\varepsilon \notin \Sigma$ marks special **ε-transitions** that may be taken without consuming any input. $\mathcal{A}$ assigns a score $\mathcal{A}[\![\mathbf{x}]\!]$ to a string $\mathbf{x} = x_1 \dots x_n \in \Sigma^*$ by summing over the scores of all possible paths deriving

[2]Some authors define τ, λ, and ρ to be partial functions—applying only a subset of transitions, initial states, and final states respectively. Our definition is equivalent, giving the weight functions value $\bar{0}$ wherever they were undefined.

$\mathbf{x}$. The score of each individual path is the product of the weights of the transitions it consists of. Formally:

Definition 5 (path score)**.** Let $\boldsymbol{\pi} = \pi_1 \dots \pi_n$ be a sequence of adjacent **transitions** in $\mathcal{A}$, with each transition $\pi_i = (q_i, q_{i+1}, z_i) \in \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\varepsilon\})$. The path $\boldsymbol{\pi}$ **derives** string $\mathbf{x} \in \Sigma^*$, which is the substring of $\mathbf{z} = z_1 z_2 \dots z_n$ that excludes ε symbols (for example, if $\mathbf{z} = a\varepsilon bc\varepsilon\varepsilon\varepsilon d$, then $\mathbf{x} = abcd$). $\boldsymbol{\pi}$'s score in $\mathcal{A}$ is given by

$$\mathcal{A}[\boldsymbol{\pi}] = \lambda(q_1) \otimes \left(\bigotimes_{i=1}^{n} \tau(\pi_i) \right) \otimes \rho(q_{n+1}). \tag{6.1}$$

Definition 6 (string score)**.** Let $\Pi(\mathbf{x})$ denote the set of all paths in $\mathcal{A}$ that derive $\mathbf{x}$. Then the score assigned by $\mathcal{A}$ to $\mathbf{x}$ is defined to be

$$\mathcal{A}[\![\mathbf{x}]\!] = \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathbf{x})} \mathcal{A}[\boldsymbol{\pi}]. \tag{6.2}$$

Because $\mathbb{K}$ is a semiring, $\mathcal{A}[\![\mathbf{x}]\!]$ can be computed in time linear in $|\mathbf{x}|$ by the Forward algorithm (Baum and Petrie, 1966). Here, for simplicity, we describe the Forward algorithm without ε-transitions.[3] Its dynamic program is given by:

$$\Omega_0(q) = \lambda(q), \tag{6.3a}$$

$$\Omega_{i+1}(q) = \bigoplus_{q' \in \mathcal{Q}} \Omega_i(q') \otimes \tau(q', q, x_i), \tag{6.3b}$$

$$\mathcal{A}[\![\mathbf{x}]\!] = \bigoplus_{q \in \mathcal{Q}} \Omega_n(q) \otimes \rho(q). \tag{6.3c}$$

$\Omega_i(q)$ gives the total score of all paths that derive $x_1 \dots x_i$ and end in state q.

Example 7. Figure 6.1 diagrams a WFSA $\mathcal{B}$, consisting of two states. A path starts from the initial state q_0 (with $\lambda(q_0) = \bar{1}$); it then takes any number of "self-loop" transitions, each consuming an input without changing the path score (since it's weighted by $\bar{1}$); it then consumes an input symbol a and takes a transition weighted by $\mu(a)$, and reaches the final state q_1 (with $\rho(q_1) =$

[3] ε-transitions can be handled with a slight modification (Schwartz et al., 2018). Note though that if $\mathcal{A}$ contains a cycle of ε-transitions, then either $\mathbb{K}$ must follow the **star semiring** laws (Kuich and Salomaa, 1986), or the number of consecutive ε-transitions allowed must be capped.

$\bar{1}$); it may further consume more input by taking self-loops at q_1, updating the path score by multiplying it by $\phi(\alpha)$ for each symbol α. Then from Definition 6, we can calculate that $\mathcal{B}$ gives the empty string score $\bar{0}$, and gives any nonempty string $\mathbf{x} = x_1 \dots x_n \in \Sigma^+$ score $\mathcal{B}[\![\mathbf{x}]\!] =$

$$\bigoplus_{i=1}^{n-1} \left(\mu(x_i) \otimes \bigotimes_{j=i+1}^{n} \phi(x_j) \right) \oplus \mu(x_n). \tag{6.4}$$

$\mathcal{B}$ can be seen as capturing soft unigram patterns (Davidov et al., 2010), in the sense that it consumes one input symbol to reach the final state from the initial state. It is straightforward to design WFSAs capturing longer patterns by including more states (Schwartz et al., 2018), as we will discuss later in §6.4 and §6.5.

6.3 Rational Recurrences

Before formally defining rational recurrences in §6.3.2, we highlight the connection between WFSAs and RNNs using a motivating example (§6.3.1).

6.3.1 A Motivating Example

We describe a simplified RNN which strips away details of some recent RNNs, in order to highlight the behaviors of the forget gate and the input.

Example 8. For an input sequence $\mathbf{x} = x_1 \dots x_n$, let the word embedding vector for x_t be $\mathbf{v}_t$. As in many gated RNN variants (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), we use a forget gate $\mathbf{f}_t$, which is computed with an affine transformation followed by an elementwise sigmoid function $\boldsymbol{\sigma}$. The current input representation $\mathbf{u}_t$ is similarly computed, but with an optional nonlinearity (e.g., tanh) $\boldsymbol{g}$. The hidden state $\mathbf{c}_t$ can be seen as a weighted sum of the previous state and the new input, controlled by the forget gate.

$$\mathbf{f}_t = \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{6.5a}$$

$$\mathbf{u}_t = (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big), \tag{6.5b}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t. \tag{6.5c}$$

The hidden state $\mathbf{c}_t$ can then be used in downstream computation, e.g., to calculate output state $\mathbf{h}_t = \tanh(\mathbf{c}_t)$, which is then fed to an MLP classifier. We focus only on the recurrent computation.

In Example 8, both $\mathbf{f}_t$ and $\mathbf{u}_t$ depend only on the current input token x_t (through $\mathbf{v}_t$), and not the previous state. Importantly, the interaction with the previous state $\mathbf{c}_{t-1}$ is *not* via affine transformations followed by nonlinearities, as in, e.g., an Elman network (Elman, 1990), where $\mathbf{c}_t = \tanh(\mathbf{W}_c\mathbf{c}_{t-1} + \mathbf{W}_v\mathbf{v}_t + \mathbf{b}_c)$. As we will discuss later, this is important in relating this recurrent update function to WFSAs.

Since the recurrent update in Equation 6.5c is elementwise, for simplicity we focus on just the ith dimension. Unrolling it in time steps, we get

$$\begin{aligned}[\mathbf{c}_t]_i &= [\mathbf{f}_t]_i\,[\mathbf{c}_{t-1}]_i + [\mathbf{u}_t]_i \\ &= \sum_{j=1}^{t-1}\left([\mathbf{u}_j]_i \prod_{k=j+1}^{t}[\mathbf{f}_k]_i\right) + [\mathbf{u}_t]_i,\end{aligned} \tag{6.6}$$

where $[\cdot]_i$ denotes the ith dimension of a vector. As noted by Lee et al. (2017b), the hidden state at time step t can be seen as a sum of previous input representations, weighted by the forget gate; longer histories typically get a smaller weight, since the forget gate values are between 0 and 1 due to the sigmoid function.

Let's recall the WFSA $\mathcal{B}$ (Figure 6.1 and Example 7) using the real semiring $\langle \mathbb{R}, +, \cdot, 0, 1\rangle$. Equation 6.6 is recovered by parameterizing $\mathcal{B}$'s weight functions μ and ϕ with

$$\mu(x_t) = [\mathbf{u}_t]_i, \quad \phi(x_t) = [\mathbf{f}_t]_i. \tag{6.7}$$

Denote the resulting WFSA by $\mathcal{B}_i$, and we have:

Proposition 9. *Running a single layer RNN in Example 8 over any nonempty input string $\mathbf{x} \in \Sigma^+$, the ith dimension of its hidden state at time step t equals the score assigned by $\mathcal{B}_i$ to $\mathbf{x}_{:t}$:*

$$[\mathbf{c}_t]_i = \mathcal{B}_i[\![\mathbf{x}_{:t}]\!]. \tag{6.8}$$

In other words, the ith dimension of the RNN in Example 8 can be seen as a WFSA structurally equivalent to $\mathcal{B}$. Its weight functions are implemented as the ith dimension of Equations 6.5, and

the learned parameters are the ith row of $\mathbf{W}$ and $\mathbf{b}$. Then it is straightforward to recover the full d-dimensional RNN, by collecting d such WFSAs, each of which is parametrized by a row in the $\mathbf{W}$s and $\mathbf{b}$s. Based on this observation, we are now ready to formally define rational recurrences.

6.3.2 Recurrences and Rationality

For a function $\mathbf{c}\colon \Sigma^* \to \mathbb{K}^d$, its *recurrence* is said to be the dependence of $\mathbf{c}(\mathbf{x}_{:t})$ on $\mathbf{c}(\mathbf{x}_{:t-1})$, for input sequence $\forall \mathbf{x} \in \Sigma^+$. We discuss a class of recurrences that can be characterized by WFSAs. The mathematical counterpart of WFSAs are *rational power series* (Berstel and Reutenauer, 1988), justifying naming such recurrences *rational*:

Definition 10 (rational recurrence)**.** The recurrence of $\mathbf{c}\colon \Sigma^* \to \mathbb{K}^d$ is said to be *rational*, if there exists a set of weighted finite state automata $\{\mathcal{A}_i\}_{i=1}^d$ over alphabet Σ and semiring $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ with both $\oplus$ and $\otimes$ taking constant time and space, such that $\forall \mathbf{x} \in \Sigma^*$,

$$\big[\mathbf{c}(\mathbf{x})\big]_i = \mathcal{A}_i[\![\mathbf{x}]\!], \ \forall i \in \{1, 2, \ldots, d\}. \tag{6.9}$$

We restrict that both operations take constant time and space, to exclude the use of arbitrarily complex semirings (§6.4.3). It directly follows from Proposition 9 that

Corollary 11. *Example 8 is rationally recurrent.*

6.4 Relationship to Existing Neural Models

This section studies several recently proposed neural architectures, and relates them to rational recurrences. §6.4.1 begins by relating some of them to the RNN defined in Example 8, and then to the WFSA $\mathcal{B}$ (Example 7). We then describe a WFSA similar to $\mathcal{B}$, but with one additional state, and discuss how it provides a new view of RNN models motivated by n-gram features (§6.4.2). In §6.4.3 we study rational recurrences that are not elementwise, using an existing model.

In the following discussion, we shall assume the real seimiring, unless otherwise noted.

6.4.1 Neural Architectures Related to $\mathcal{B}$

Despite its simplicity, Example 8 corresponds to several existing neural architectures. For instance, quasi-RNN (QRNN; Bradbury et al., 2017) and simple recurrent unit (SRU; Lei et al., 2017b) aim to speed up the recurrent computation. To do so, they drop the matrix multiplication dependence on the previous hidden state, resulting in similar recurrences to that in Example 8.[4] Other works start from different motivations, but land on similar recurrences, e.g., strongly-typed RNNs (T-RNN; Balduzzi and Ghifary, 2016) and its gated variants (T-LSTM and T-GRU), and structurally constrained RNNs (SCRN; Mikolov et al., 2014).

The analysis in §6.3.1 directly applies to SRU, T-RNN, and SCRN. In fact, Example 8 presents a slightly more complicated version of them. In these models, input representations are computed without the bias term or any nonlinearity: $\mathbf{u}_t = \mathbf{W}_u \mathbf{v}_t$. By Proposition 9 and Corollary 11:

Corollary 12. *Single-layer SRU, T-RNN, and SCRN architectures are rationally recurrent.*

It is slightly more complicated to analyze the recurrences of the QRNN, T-LSTM, and T-GRU. Although their hidden states $\mathbf{c}_t$ are updated in the same way as Equation 6.5c, the input representations and gates may depend on *previous inputs*. For example, in T-LSTM and T-GRU, the forget gate is a function of two consecutive inputs:

$$\mathbf{f}_t = \boldsymbol{\sigma}\left(\mathbf{V}_f \mathbf{v}_{t-1} + \mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\right). \tag{6.10}$$

QRNNs are similar, but may depend on up to K tokens, due to the K-window convolutions. Eisner (2002) discuss finite state machines for second (or higher) order probabilistic sequence models. Following the same intuition, we sketch the construction of WFSAs corresponding to QRNNs with 2-window convolutions in Appendix E.1, and summarize the key results here:

Proposition 13. *Single-layer T-GRU, T-LSTM, and QRNN are rationally recurrent. In particular, a single-layer d-dimensional QRNN using K-window convolutions can be recovered by a set of d WFSAs, each with $\mathcal{O}(2\,|\Sigma|^{K-1})$ states.*

[4] The SRU architecture discussed through this work is based on Lei et al. (2017b). In a later updated version, Lei et al. (2018) introduce diagonal matrix multiplication interaction in the hidden state updates, inspired by (Li et al., 2018), which yields a recurrence not obviously rational.

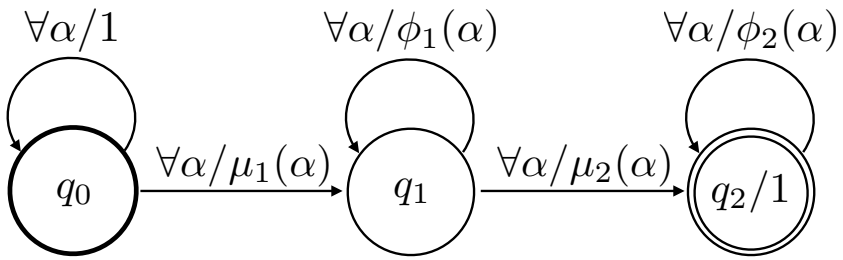


Figure 6.2: A three-state WFSA $\mathcal{C}$ discussed in §6.4.2.

The size of WFSAs needed to recover QRNN grows exponentially in the window size. Therefore, at least for QRNNs, Proposition 13 has more conceptual value than practical.

6.4.2 More than Two States

So far our discussion has centered on $\mathcal{B}$, a two-state WFSA capturing unigram patterns (Example 7). In the same spirit as going from unigram to n-gram features, one can use WFSAs with more states to capture longer patterns (Schwartz et al., 2018). In this section we augment $\mathcal{B}$ by introducing more states, and explore its relationship to some neural architectures motivated by n-gram features. We start with a three-state WFSA as an example, and then discuss more general cases.

Figure 6.2 diagrams a WFSA $\mathcal{C}$, augmenting $\mathcal{B}$ with another state. To reach the final state q_2, at least two transitions must be taken, in contrast to one in $\mathcal{B}$. History information is decayed by the self-loop at the final state q_2, assuming ϕ_2 is between 0 and 1. $\mathcal{C}$ has another self-loop over q_1, weighted by $\phi_1 \in (0, 1)$. The motivation is to allow (but down-weight) nonconsecutive bigrams, as we will soon show.

The scores assigned by $\mathcal{C}$ can be inductively computed by applying the Forward algorithm (§6.2). Given input sequence $\mathbf{x}$ longer than one, let $\mathcal{C}[\![\mathbf{x}_{:0}]\!] = 0$, then $\mathcal{C}[\![\mathbf{x}_{:t+1}]\!] =$

$$\mathcal{C}[\![\mathbf{x}_{:t}]\!]\, \phi_2(x_{t+1}) + \beta_t\, \mu_2(x_{t+1}), \tag{6.11}$$

where

$$\beta_t = \beta_{t-1}\, \phi_1(x_t) + \mu_1(x_t), \tag{6.12}$$

and $\beta_0 = 0$. Unrolling β_t in time, we get $\beta_t =$

$$\sum_{j=1}^{t-1} \left(\mu_1(x_j) \prod_{k=j+1}^{t} \phi_1(x_k) \right) + \mu_1(x_t). \tag{6.13}$$

Due to the self-loop over state q_1, β_t can be seen as a weighted sum of the μ_1 terms up to x_t (Equaltion 6.13). The second product term in Equation 6.11 then provides multiplicative interactions between μ_2, and the weighted sum of μ_1s. In this sense, it captures nonconsecutive bigram features.

At a first glance, Equations 6.11 and 6.12 resemble recurrent convolutional neural networks (RCNN; Lei et al., 2016). RCNN is inspired by nonconsecutive n-gram features and low rank tensor factorization. It is later studied from a string kernel perspective (Lei et al., 2017a). Here we review its nonlinear bigram version:

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \boldsymbol{\lambda}_t + \mathbf{u}_t^{(1)}, \tag{6.14a}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \boldsymbol{\lambda}_t + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}, \tag{6.14b}$$

where the $\mathbf{u}_t^{(j)}$s are computed similarly to Equation 6.5b, and $\mathbf{c}_t^{(2)}$ is used as output for onward computation. Different strategies to computing $\boldsymbol{\lambda}_t$ were explored (Lei et al., 2015, 2016). When $\boldsymbol{\lambda}_t$ is a constant, or depends only on x_t, e.g., $\boldsymbol{\lambda}_t = \boldsymbol{\sigma}(\mathbf{W}_\lambda \mathbf{v}_t + \mathbf{b}_\lambda)$, the ith dimension of Equations 6.14 can be recovered from Equation 6.11, by letting

$$\mu_j(x_t) = [\mathbf{u}_t^{(j)}]_i,\ \phi_j(x_t) = [\boldsymbol{\lambda}_t]_i,\ j = 1, 2. \tag{6.15}$$

It is straightforward to generalize the above discussion to higher order cases: n-gram RCNN corresponds to WFSAs with $n+1$ states, constructed similarly to how we build $\mathcal{C}$ from $\mathcal{B}$ (Appendix E.2).

Proposition 14. *For a single-layer RCNN with $\boldsymbol{\lambda}_t$ being a constant or depending only on x_t, the recurrence is rational.*

As noted later in §6.4.3, its recurrence may not be rational when $\boldsymbol{\lambda}_t = \boldsymbol{\sigma}(\mathbf{W}_c \mathbf{c}_{t-1} + \mathbf{W}_\lambda \mathbf{v}_t + \mathbf{b}_\lambda)$.

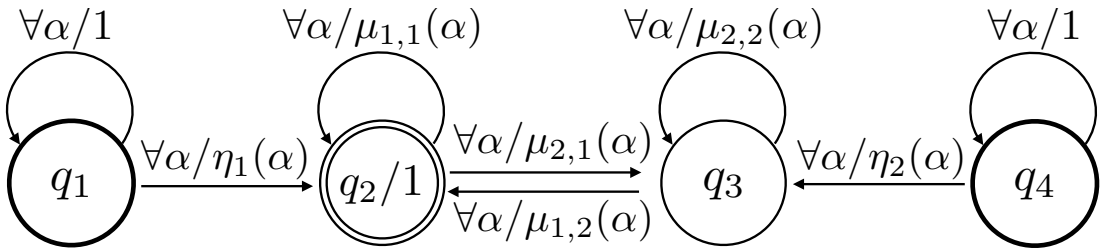


Figure 6.3: WFSA $\mathcal{D}_1$ discussed in §6.4.3. Two initial states q_1 and q_4 are used here.

6.4.3 Beyond Elementwise Operations

So far we have discussed rational recurrences for models using elementwise recurrent updates (e.g., Equation 6.5c). This section uses an existing model as an example, to study a rational recurrence that is not elementwise. We focus on the input switched affine network (ISAN; Foerster et al., 2017). Aiming for efficiency and interpretability, it does not use any explicit nonlinearity; its affine transformation parameters depend only on the input:

$$\mathbf{c}_t = \mathbf{W}_{x_t}\mathbf{c}_{t-1} + \mathbf{b}_{x_t}. \tag{6.16}$$

Due to the matrix multiplication, the recurrence of a single-layer ISAN is not elementwise. Yet, we argue that it is rational. We will sketch the proof for a 2-dimensional case, and it is straightforward to generalize to higher dimensions (E.3).

We define two WFSAs, each recovering one dimension of ISAN's recurrent updates. Figure 6.3 diagrams one of them, $\mathcal{D}_1$. The other one, $\mathcal{D}_2$, is identical (including shared weights), except using q_3 instead of q_2 as the final state. For any nonempty input sequence $\mathbf{x} \in \Sigma^+$, the scores assigned by $\mathcal{D}_1$ and $\mathcal{D}_2$ can be inductively computed by applying the Forward algorithm. Letting $\mathcal{D}_1[\![\mathbf{x}_{:0}]\!] = \mathcal{D}_2[\![\mathbf{x}_{:0}]\!] = 0$, for $t \geq 1$

$$\begin{bmatrix} \mathcal{D}_1[\![\mathbf{x}_{:t}]\!] \\ \mathcal{D}_2[\![\mathbf{x}_{:t}]\!] \end{bmatrix} = \widetilde{\mathbf{W}}_{x_t} \begin{bmatrix} \mathcal{D}_1[\![\mathbf{x}_{:t-1}]\!] \\ \mathcal{D}_2[\![\mathbf{x}_{:t-1}]\!] \end{bmatrix} + \widetilde{\mathbf{b}}_{x_t}, \tag{6.17}$$

where

$$\begin{aligned} \widetilde{\mathbf{W}}_{x_t} &= \begin{bmatrix} \mu_{1,1}(x_t) & \mu_{1,2}(x_t) \\ \mu_{2,1}(x_t) & \mu_{2,2}(x_t) \end{bmatrix}, \\ \widetilde{\mathbf{b}}_{x_t} &= \begin{bmatrix} \eta_1(x_t) \\ \eta_2(x_t) \end{bmatrix}. \end{aligned} \tag{6.18}$$

Then Equation 6.16, in the case of hidden size 2, is recovered by letting $\mathbf{W}_{x_t} = \widetilde{\mathbf{W}}_{x_t}$ and $\mathbf{b}_{x_t} = \widetilde{\mathbf{b}}_{x_t}$.

Proposition 15. *The recurrence of a single-layer ISAN is rational.*

Corollary 16. *For a single-layer Elman network, in the absence of any nonlinearity, the recurrence is rational.*

Discussion. It is known that an Elman network can approximate any recursively computable partial function (Siegelmann and Sontag, 1995). On the other hand, in their single-layer cases, WFSAs (and thus models with rational recurrences) are restricted to rational series (Schützenberger, 1961). Therefore, we hypothesize that models like Elman networks, LSTMs, and GRUs, where the recurrences depend on previous states through affine transformations followed by nonlinearities, are not rational.

This chapter does not intend to propose rational recurrences as a concept general enough to include most existing RNNs. Rather, we wish to study a more constrained class of methods to better understand the connections between WFSAs and RNNs. Therefore in Definition 10, we restrict the semirings to be "simple," in the sense that both operations take constant time and space. Such a restriction aims to exclude the possibility of hiding arbitrarily complex computations inside the semiring, which might allow RNNs to satisfy the definition in a trivial and unilluminating way.

Such theoretical limitations might be less severe than they appear, since it is not yet entirely clear what they correspond to in practice, especially when multiple vertical layers of these models are used (Leshno and Schocken, 1993). We defer to future work the further study of the connections between WFSAs and Elman-style RNNs.

	Models	Recurrence Function	WFSA
§6.4.1	SRU, SCRN T-RNN, QRNN	$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t$	$\mathcal{B}$
§6.4.2	RCNN	$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \boldsymbol{\lambda}_t + \mathbf{u}_t^{(1)}$ $\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \boldsymbol{\lambda}_t + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}$	$\mathcal{C}$
§6.4.3	ISAN	$\mathbf{c}_t = \mathbf{W}_{x_t}\mathbf{c}_{t-1} + \mathbf{b}_{x_t}$	$\mathcal{D}_1, \mathcal{D}_2$

Table 6.1: Recurrent neural network architectures discussed in §6.4 and their corresponding WFSAs.

Closing this section, Table 6.1 summarizes the discussed recurrent neural architectures and their corresponding WFSAs.

6.5 Deriving Neural Models from WFSAs

Rational recurrences provide a new view of several recently proposed neural models. Based on such observations, this section aims to explore potential approaches to designing neural architectures in a more interpretable and intuitive way: by deriving them from WFSAs. §6.5.1 studies an interpolation of unigram and bigram features by combining 2-state and 3-state WFSAs (Figures 6.1 and 6.2). We then explore alternative semirings (§6.5.2), an approach orthogonal to what we've discussed so far.

We note that our goal is not to devise new state-of-the-art architectures. Rather, we illustrate a new design process for neural architectures that draws inspiration from WFSAs. That said, in our experiments (§6.6), one of our new architectures performs as well as or better than strong baselines.

6.5.1 Aggregating Different Length Patterns

We start by presenting a straightforward extension to 2-state and 3-state rational models: one combining both. It is inspired by many classical NLP models, where unigram features and higher-order ones are interpolated.

Figure 6.4 diagrams a 4-state WFSA $\mathcal{F}$. Compared to $\mathcal{C}$ (Figure 6.2), $\mathcal{F}$ uses q_1 as a second final state, aiming to capture both unigram and bigram patterns, since a path is allowed to stop at q_1 after consuming one input. The final states are weighted by ρ_1 and ρ_2 respectively.

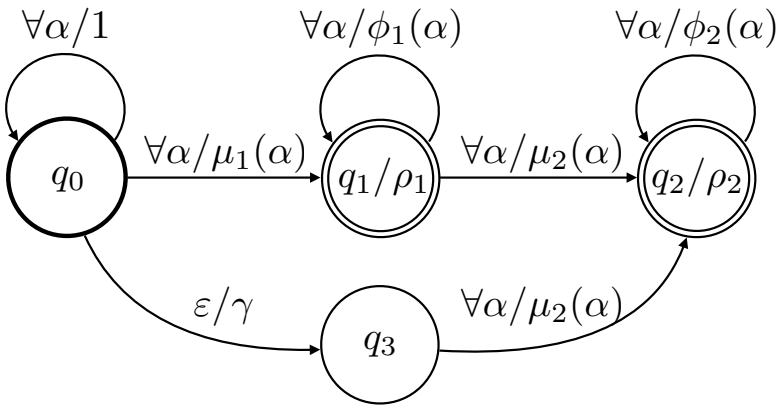


Figure 6.4: A WFSA $\mathcal{F}$ that combines both unigram and bigram features (§6.5.1). Two final states q_1 and q_2 are used, with weights ρ_1 and ρ_2, respectively.

Another notable modification is the additional state q_3, which is used to create a "shortcut" to reach q_2, together with an ε-transition. Specifically, starting from q_0, a path can now take the ε-transition and reach q_3, and then take a transition with weight μ_2 to reach q_2. Recall from §6.2, that ε-transitions do not consume any input, yet they can still be weighted by a (parameterized) function γ *not* depending on the inputs. The ε-transition allows for skipping the first word in a bigram. It can be discouraged by using $\gamma \in (0, 1)$, just as we do in our experiments.

Deriving the neural architecture. As in §6.3, we relate hidden states of an RNN to the scores assigned by WFSAs to input strings. We then derive the neural architecture with a dynamic program. Here we keep the discussion self-contained by explicitly overviewing the procedure. It is a direct application of the Forward algorithm (§6.2), though now in a form that deals with the ε-transition. Such an approach applies, of course, to more general cases, as noted by Schwartz et al. (2018).

Given an input string $\mathbf{x} \in \Sigma^+$, let $z_t^{(j)}$ denote the total score of all paths landing in state q_j *just*

after consuming x_t. Let $z_0^{(j)} = 0$, then for $t \geq 1$,

$$
\begin{aligned}
z_t^{(0)} &= 1 \\
z_t^{(1)} &= z_{t-1}^{(1)}\, \phi_1(x_t) + z_{t-1}^{(0)}\, \mu_1(x_t) \\
z_t^{(3)} &= z_t^{(0)}\, \gamma \\
z_t^{(2)} &= z_{t-1}^{(2)}\, \phi_2(x_t) + (z_{t-1}^{(1)} + z_{t-1}^{(3)})\, \mu_2(x_t) \\
\mathcal{F}[\![\mathbf{x}_{:t}]\!] &= \rho_1\, z_t^{(1)} + \rho_2\, z_t^{(2)}.
\end{aligned}
$$

We now collect d of these WFSAs to construct an RNN, and we parameterize their weight functions with the technique we've been using:

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{6.19a}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + (\mathbf{c}_{t-1}^{(1)} + \mathbf{r}) \odot \mathbf{u}_t^{(2)}, \tag{6.19b}$$

$$\mathbf{c}_t = \mathbf{p}^{(1)} \odot \mathbf{c}_t^{(1)} + \mathbf{p}^{(2)} \odot \mathbf{c}_t^{(2)}, \tag{6.19c}$$

where

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)} \mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \tag{6.20a}$$

$$\mathbf{u}_t^{(j)} = (\mathbf{1} - \mathbf{f}_t^{(j)}) \odot \boldsymbol{g}\big(\mathbf{W}_u^{(j)} \mathbf{v}_t + \mathbf{b}_u^{(j)}\big), \tag{6.20b}$$

$$\mathbf{p}^{(j)} = \boldsymbol{\sigma}(\mathbf{b}_p^{(j)}), \quad \mathbf{r} = \boldsymbol{\sigma}(\mathbf{b}_r). \tag{6.20c}$$

The $\mathbf{p}$ vectors correspond to the final state weights ρ_1 and ρ_2. Despite the similarities, $\mathbf{p}$ are different from output gates (Bradbury et al., 2017), since the former do not depend on the input, and are parameterized (through a sigmoid) by two leanred vectors $\mathbf{b}_p^{(j)}$. The same applies to $\mathbf{r}$ and $\mathbf{b}_r$, which correspond to the weights for ε-transitions γ.

There are, of course, other ways to interpolate patterns of different lengths, e.g., by using WFSAs of different structures in a single neural architecture (Schwartz et al., 2018). We aim for exploring classical WFSA techniques in neural architectures, by using ε-transitions and additional final states. In §6.6 we will show that this architecture outperforms both unigram and bigram baselines.

6.5.2 Alternative Semirings

Our new understanding of rational recurrences allows us to consider a different kind of extension: replacing the semiring. We introduce an example, which modifies Example 8 by replacing its real (plus-times) semiring with the max-plus semiring $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$:

Example 17.

$$\mathbf{f}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{6.21a}$$

$$\mathbf{u}_t = \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big), \tag{6.21b}$$

$$\mathbf{c}_t = \max\{\mathbf{f}_t + \mathbf{c}_{t-1}, \mathbf{u}_t\}. \tag{6.21c}$$

Example 17 does not use the forget gate when computing $\mathbf{u}_t$ (Equation 6.21b), which is different from its plus-times counterpart, where $\mathbf{u}_t = (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big)$. The reason is that, unlike the real semiring, the max-plus semiring lacks a well-defined negation. Possible alternatives include taking the log of a separate input gate, or using $\log(\mathbf{1} - \mathbf{f}_t)$, which we leave for future work.

Example 17 can be seen as replacing sum-pooling with max-pooling. Both max and sum-pooling have been used successfully in vision and NLP models. Intuitively, max-pooling "detects" the occurrence of a pattern while sum-pooling "counts" the occurrence of a pattern. One advantage of max operator is that the model's decisions can be back-traced and interpreted, as argued by Schwartz et al. (2018). Such a technique is applicable to all the models with rational recurrences.

6.6 Experiments

This section evaluates four rational RNNs on language modeling (§6.6.2) and text categorization (§6.6.3). Our goal is to compare the behaviors of models derived from different WFSAs, showing that our understanding of WFSAs allows us to improve existing rational models.

Model	Unigram	Bigram	Semiring
RRNN($\mathcal{B}$)	✓		real
RRNN($\mathcal{B}$)$_{\text{m+}}$	✓		max-plus
RRNN($\mathcal{C}$)		✓	real
RRNN($\mathcal{F}$)	✓	✓	real

Table 6.2: Rational recurrent neural architectures compared in the experiments (§6.6.1).

6.6.1 Compared Models

Our comparisons focus on the *recurrences* of the models, i.e., how the hidden states $\mathbf{c}_t$ are computed (e.g., Equations 6.5c and 6.19c). Therefore we follow Lei et al. (2017b) and use $\mathbf{u}_t^{(j)} = \mathbf{W}_u^{(j)} \mathbf{v}_t^{(j)}$ across all compared models, listed below and as well as in Table 6.2:

- RRNN($\mathcal{B}$), with real semiring (§6.4.1);
- RRNN($\mathcal{B}$)$_{\text{m+}}$, with max-plus semiring (§6.5.2);
- RRNN($\mathcal{C}$), with real semiring (§6.4.2);
- RRNN($\mathcal{F}$), with real semiring (§6.5.1).

We also compare to an LSTM baseline. Aiming to control for comfounding factors, we do not use highway connections in any of the models.[5] In the interest of space, the full architectures and hyperparameters are detailed in Appendices E.4 and E.5.

6.6.2 Language Modeling

Dataset and implementation. We experiment with the Penn Treebank corpus (PTB; Marcus et al., 1993). We use the preprocessing and splits from Mikolov et al. (2010), resulting in a vocabulary size of 10K and 1M tokens.

Following standard practice, we treat the training data as one long sequence, split into mini batches, and train using BPTT truncated to 35 time steps (Williams and Peng, 1990). The input embeddings and output softmax weights are tied (Press and Wolf, 2017).

Results. Following Collins et al. (2017) and Melis et al. (2018), we compare models controlling for parameter budget. Table 6.3 summarizes language modeling perplexities on PTB test set. The

[5] Thus RRNN($\mathcal{B}$) is essentially an SRU without highway connections. We denote it differently, to note its differences from the original implementation (Lei et al., 2017b). Similarly, we do not denote RRNN($\mathcal{C}$) as RCNN (Lei et al., 2016).

Model	ℓ	# Params.	Dev.	Test
LSTM	2	24M	73.3	71.4
LSTM	3	24M	78.8	76.2
RRNN($\mathcal{B}$)	2	10M	73.1	69.2
RRNN($\mathcal{B}$)$_{\text{m+}}$	2	10M	75.1	71.7
RRNN($\mathcal{C}$)	2	10M	72.5	69.5
RRNN($\mathcal{F}$)	2	10M	**69.5**	**66.3**
RRNN($\mathcal{B}$)	3	24M	68.7	65.2
RRNN($\mathcal{B}$)$_{\text{m+}}$	3	24M	70.8	66.9
RRNN($\mathcal{C}$)	3	24M	70.0	67.0
RRNN($\mathcal{F}$)	3	24M	**66.0**	**63.1**

Table 6.3: Language modeling perplexity on PTB test set (lower is better). LSTM numbers are taken from Lei et al. (2017b). ℓ denotes the number of layers. Bold font indicates best performance.

middle block compares all models with two layers and 10M trainable parameters. RRNN($\mathcal{B}$) and RRNN($\mathcal{C}$) achieve roughly the same performance; interpolating both unigram and bigram features, RRNN($\mathcal{F}$) outperforms others by more than 2.9 test perplexity. For the three-layer and 24M setting (the bottom block), we observe similar trends, except that RRNN($\mathcal{C}$) slightly underperforms RRNN($\mathcal{B}$). Here RRNN($\mathcal{F}$) outperforms others by more than 2.1 perplexity.

Using a max-plus semiring, RRNN($\mathcal{B}$)$_{\text{m+}}$ *underperforms* RRNN($\mathcal{B}$) under both settings. Possible reasons could be the suboptimal design choice for computing input representations in the former (§6.5.2). Finally, most compared models outperform the LSTM baselines, whose numbers are taken from Lei et al. (2017b).[6]

6.6.3 Text Classification

Implementation. We use unidirectional 2-layer architectures for all compared models. To build the classifiers, we feed the final RNN hidden states into a 2-layer tanh-MLP. Further implementation details are described in Appendix E.5.

Datasets. We experiment with four binary text classification datasets, described below.

- **Amazon** (electronic product review corpus; McAuley and Leskovec, 2013).[7] We focus on the

[6] Melis et al. (2018) point out that carefully tuning LSTMs can achieve much stronger performance, at the cost of exceptionally large amounts of computational resources for tuning.

[7] http://riejohnson.com/cnn_data.html

Split	Amazon	SST	subj	CR
Train	20K	6.9K	8K	3.0K
Dev.	5K	0.9K	1K	0.4K
Test	25K	1.8K	1K	0.4K

Table 6.4: Number of instances in the text classification datasets (§6.6.3).

positive and negative reviews.

- **SST** (Stanford sentiment treebank; Socher et al., 2013).[8] We focus on the binary classification task. SST provides labels for syntactic phrases; we experiment with a more realistic setup, and consider only complete sentences at either training or evaluating time.
- **subj** (Subjectivity dataset; Pang and Lee, 2004). As **subj** doesn't come with official splits, we randomly split it to train (80%), development (10%), and test (10%) sets.
- **CR** (customer reviews dataset; Hu and Liu, 2004).[9] As with **subj**, we randomly split this dataset using the same ratio.

Table 6.4 summarizes the sizes of the datasets.

Results. Table 6.5 summarizes text classification test accuracy. We report the average performance of 5 trials different only in random seeds. RRNN($\mathcal{F}$) outperforms all other models on 3 out of the 4 datasets. For Amazon, the largest one, we do not observe significant differences between RRNN($\mathcal{F}$) and RRNN($\mathcal{C}$), while both outperform others. This may suggest that the interpolation of unigram and bigram features by RRNN($\mathcal{F}$) is especially useful in small data setups. As in the language modeling experiments, RRNN$(\mathcal{B})_{\text{m+}}$ underperforms all other models in most cases, and in particular RRNN($\mathcal{B}$). These results provide evidence that replacing the real semiring in rational models might be challenging. We leave further exploration to future work.

6.7 Related Work

Weighted finite state automata. WFSAs were once popular among many sequential tasks (Mohri et al., 2002; Kumar and Byrne, 2003; Cortes et al., 2004; Pardo and Birmingham, 2005; Moore et al., 2006, *inter alia*), and are still successful in morphology (Dreyer, 2011; Cotterell et al., 2015; Rastogi

[8] nlp.stanford.edu/sentiment/index.html
[9] http://www.cs.uic.edu/?liub/FBS/sentiment-analysis.html

Model	Amazon	SST	subj	CR
LSTM	$91.2_{\pm 0.3}$	$85.1_{\pm 0.6}$	$93.3_{\pm 0.6}$	$82.4_{\pm 1.5}$
RRNN($\mathcal{B}$)	$92.4_{\pm 0.1}$	$85.8_{\pm 0.3}$	$93.9_{\pm 0.4}$	$84.1_{\pm 1.0}$
RRNN($\mathcal{C}$)	$\mathbf{92.8}_{\pm 0.2}$	$84.8_{\pm 0.4}$	$93.8_{\pm 0.6}$	$84.5_{\pm 0.9}$
RRNN($\mathcal{B}$)$_{\text{m+}}$	$89.2_{\pm 3.1}$	$84.9_{\pm 0.4}$	$92.6_{\pm 0.5}$	$84.3_{\pm 0.5}$
RRNN($\mathcal{F}$)	$92.7_{\pm 0.2}$	$\mathbf{86.5}_{\pm 0.6}$	$\mathbf{94.8}_{\pm 0.5}$	$\mathbf{85.1}_{\pm 0.5}$

Table 6.5: Text classification test accuracy averaged over 5 runs. $\pm$ denotes standard deviation, and bold font indicates best averaged performance.

et al., 2016, *inter alia*). Compared to neural networks, WFSAs are better understood theoretically and arguably more interpretable. They were recently revisited in combination with the former in, e.g., text generation (Ghazvininejad et al., 2016, 2017; Lin et al., 2017) and automatic music accompaniment (Forsyth, 2016).

Recurrent neural networks. RNNs (Elman, 1990; Jordan, 1989) prove to be strong models for sequential data (Siegelmann and Sontag, 1995). Besides the perhaps most notable gated variants (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), extensive efforts have been devoted to developing alternatives (Balduzzi and Ghifary, 2016; Miao et al., 2016; Zoph and Le, 2017; Lee et al., 2017b; Lei et al., 2017a; Vaswani et al., 2017; Gehring et al., 2017, *inter alia*). Departing from the above approaches, this chapter derives RNN architectures drawing inspiration from WFSAs.

Another line of work studied the connections between WFSAs and RNNs in terms of modeling capacity, both empirically (Kolen, 1993; Giles et al., 1992; Weiss et al., 2018, *inter alia*) and theoretically (Cleeremans et al., 1989; Visser et al., 2001; Chen et al., 2018, *inter alia*).

6.8 Summary

In this chapter, we presented *rational recurrences*, a new construction to study the recurrent updates in RNNs, drawing inspiration from weighted finite state automata. We showed that rational recurrences are in frequent use by several recently proposed recurrent neural architectures, providing new understanding of them. Based on such connections, we discussed approaches to deriving novel neural architectures from WFSAs. Our empirical results demonstrate the potential of doing so.

Chapter 7

Conclusion

This thesis takes several steps towards building efficient, robust, generalizable, and interpretable representation learners for NLP. We explored structural inductive biases for better generalization, and studied algorithms to improve the efficiency of state-of-the-art NLP models.

With SPIGOT (Chapter 2), we are able to incorporate discrete structured prediction as intermediate layers into neural architectures and train them end-to-end. It derives a surrogate for the gradients of the nondifferentiable discrete structured argmax respecting the structured constraints. SPIGOT is general, and can be used in unsupervised and semi-supervised learning, as well as the joint training of NLP pipelines. Our empirical results verify that using SPIGOT to incorporate "learnable" structural inductive biases yields more accurate and interpretable decisions in downstream tasks. PaLM augments neural language models with syntactically-informed attention (Chapter 3). The attention component can optionally be supervised using syntactic annotations, providing an intuitive and lightweight way of integrating structural inductive biases into neural models. PaLM can also derive an unsupervised constituency parser, whose parameters are estimated purely using the language modeling objective.

In Part II, we explored algorithms to improve the efficiency of the transformers. RFA provides a linear-complexity approximation to the quadratic-cost canonical softmax attention, drawing inspiration from the well-established random feature methods (Chapter 4). It is broadened by. Chapter 5, which presented ABC, a general framework for several recent efficient attention models. RFA and ABC reveal new insights into existing approaches and show that they bear closer connections to RNNs than previously realized. In a similar spirit, Chapter 6 connects a class of

RNNs with traditional NLP methods through a WFSA view. These findings lead to a unifying perspective of existing approaches, and offers flexible ways to devise better-performing, more efficient, and more interpretable models neural architectures imbued with desired inductive biases.

7.1 Future Directions

Scaling up will continue to play an important role in NLP. From an algorithmic perspective, building efficient neural models helps promote the accessibility and inclusiveness of cutting-edge NLP research and mitigate its negative environmental impact. While the linear complexity attention models we presented achieved promising results, they are provably less expressive. It remains unclear whether the limited capacity will hold them back in large-scale pretraining. Besides, further investigation is needed on how efficient architectures interact with orthogonal ways to improve efficiency. Such exploration could benefit from a better understanding of current approaches. Rational recurrences and Abc laid early ground; a unifying view of attentive, recurrent, and convolutional models could lead to flexible ways to build future generation architectures that would inevitably replace transformers. Future work should pay equal attention to practical concerns and ensure meticulous implementations and familiarity with modern hardware.

Large language models pretrained on massive data have shifted the paradigm of NLP. However, simply scaling up will hardly lead us to the ultimate goal of building AI with general linguistic capability. In fact, it is still being debated whether or not natural language meaning can be learned from surface forms alone (Bender and Koller, 2020). An imminent challenge is to study the knowledge acquired by large language models and better understand the limits of scaling up. Symbolic structures provide a suite of tools, and algorithms such as spigot and PaLM could prove useful. Current pretraining frameworks are less capable of picking up symbolic knowledge from the data. Future work could explore how structured knowledge resources can help design better pretraining paradigms.

Bibliography

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proc. of EMNLP*.

Maximilian Alber, Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Fei Sha. 2017. An empirical study on the properties of random bases for kernel methods. In *Proc. of NeurIPS*.

Mariana S. C. Almeida and André F. T. Martins. 2015. Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data. In *Proc. of SemEval*.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Many languages, one parser. *TACL*, 4:431–444.

Brandon Amos and J. Zico Kolter. 2017. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*.

Haim Avron, L. Kenneth Clarkson, and P. David and Woodruff. 2017. Faster kernel ridge regression using sketching and preconditioning. *SIAM J. Matrix Analysis Applications*.

Haim Avron, Vikas Sindhwani, Jiyan Yang, and Michael W. Mahoney. 2016. Quasi-Monte Carlo feature maps for shift-invariant kernels. *Journal of Machine Learning Research*, 17(120):1–38.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. 2016. Using fast weights to attend to the recent past. In *Proc. of NeurIPS*.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *Proc. of ICLR*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

J. K. Baker. 1979. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*.

David Balduzzi and Muhammad Ghifary. 2016. Strongly-typed recurrent neural networks. In *Proc. of ICML*.

Jonathan F. Bard. 2010. *Practical Bilevel Optimization: Algorithms and Applications*. Springer.

Leonard E. Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

David Belanger and Andrew McCallum. 2016. Structured prediction energy networks. In *Proc. of ICML*.

David Belanger, Bishan Yang, and Andrew McCallum. 2017. End-to-end learning for structured prediction energy networks. In *Proc. of ICML*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv: 2004.05150*.

Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proc. of ACL*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432.

Jean Berstel, Jr. and Christophe Reutenauer. 1988. *Rational Series and Their Languages*. Springer-Verlag, Berlin, Heidelberg.

Dimitris Bertsimas and John Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific.

Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst, editors. 2009. *Heteroassociative Memory*, pages 1829–1829. Springer Berlin Heidelberg, Berlin, Heidelberg.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*.

S. Bochner. 1955. *Harmonic Analysis and the Theory of Probability*. University of California Press.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proc. of WMT*.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the opportunities and risks of foundation models. *arXiv:2108.07258*.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural network. In *Proc. of ICLR*.

Philémon Brakel, Dirk Stroobandt, and Benjamin Schrauwen. 2013. Training energy-based models for time-series imputation. *Journal of Machine Learning Research*, 14:2771–2797.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, P. Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, R. Child, Aditya Ramesh, D. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, E. Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, J. Clark, Christopher Berner, Sam McCandlish, A. Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *arXiv:2005.14165*.

Peter Brucker. 1984. An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters*, 3(3):163 – 166.

Jan Buys and Phil Blunsom. 2018. Neural syntactic generative models with exact marginalization. In *Proc. of NAACL*.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *Proc. of IWSLT*.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proc. of COLING*.

Kehai Chen, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019. Recurrent positional embedding for neural machine translation. In *Proc. of EMNLP*.

Yining Chen, Sorcha Gilroy, Kevin Knight, and Jonathan May. 2018. Recurrent neural networks as weighted language recognizers. In *Proc. of NAACL*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv: 1904.10509*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. of EMNLP*.

Youngmin Cho and Lawrence K. Saul. 2009. Kernel methods for deep learning. In *Proc. of NeurIPS*.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2017. Unsupervised learning of task-specific tree structures with tree-LSTMs. arXiv:1707.02786.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *Proc. of ICLR*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker

Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling language modeling with pathways. arXiv:2204.02311.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. 1989. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (ELUs). In *Proc. of ICLR*.

Shay B. Cohen and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proc. of EMNLP*.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. 2017. Capacity and trainability in recurrent neural networks. In *Proc. of ICLR*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of ICML*.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. 2004. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *TACL*, 3:433–447.

Kornél Csernai. 2017, accessed September 1, 2020. *First Quora Dataset Release: Question Pairs*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proc. of ACL*.

Dipanjan Das, André F. T. Martins, and Noah A. Smith. 2012. An exact dual decomposition algorithm for shallow semantic parsing with constraints. In *Proc. of *SEM*.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Enhanced sentiment learning using twitter hashtags and smileys. In *Proc. of COLING*.

Peter Dayan and Laurence F. Abbott. 2001. *Theoretical Neuroscience*, volume 806. MIT Press, Cambridge, MA.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proc. of ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*.

Justin Domke. 2012. Generic methods for optimization-based modeling. In *Proc. of AISTATS*.

Markus Dreyer. 2011. *A Non-parametric Model for the Discovery of Inflectional Paradigms from Plain Text Using Graphical Models over Strings*. Ph.D. thesis, Johns Hopkins University.

Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. In *Proc. of NAACL*.

David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proc. of NeurIPS*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proc. of NAACL*.

Chris Dyer, Gábor Melis, and Phil Blunsom. 2019. A critical analysis of biased parsers in unsupervised parsing. arXiv:1909.09428.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proc. of NAACL*.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*, pages 29–61. Springer Netherlands.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop. In *Proceedings of the EMNLP Workshop on Structured Prediction for NLP*.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proc. of ICLR*.

Jenny Rose Finkel and Christopher D. Manning. 2010. Hierarchical joint learning: Improving joint parsing and named entity recognition with non-jointly labeled data. In *Proc. of ACL*.

Jakob N. Foerster, Justin Gilmer, Jan Chorowski, Jascha Sohl-Dickstein, and David Sussillo. 2017. Intelligible language modeling with input switched affine networks. In *Proc. of ICML*.

Jonathan P. Forsyth. 2016. *Automatic musical accompaniment using finite state machines*. Ph.D. thesis, New York University.

Abram L. Friesen and Pedro M. Domingos. 2016. The sum-product theorem: A foundation for learning tractable models. In *Proc. of ICML*.

Abram L. Friesen and Pedro M. Domingos. 2018. Deep learning as a mixed convex-combinatorial optimization problem. In *Proc. of ICLR*.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proc. of NeurIPS*.

Yingbo Gao, Christian Herold, Weiyue Wang, and Hermann Ney. 2019. Exploring kernel functions in the softmax layer for contextual word classification. In *International Workshop on Spoken Language Translation*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proc. of EMNLP*.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proc. of ACL System Demonstrations*.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proc. of ACL*.

C. Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *JAIR*, 57:345–420.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proc. of ACL*.

Ian Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Multi-prediction deep Boltzmann machines. In *Proc. of NeurIPS*.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. 2016. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. arXiv:1607.05447.

Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. 2021. Coordination among neural modules through a shared global workspace.

Alex Graves. 2013. Generating sequences with recurrent neural networks. arXiv:1308.0850.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Proc. of NeurIPS*.

Nizar Habash and Owen Rambow. 2005. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proc. of ACL*.

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. 2019. Modeling recurrence for transformer. In *Proc. of NAACL*.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proc. of EMNLP*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *Proc. of CVPR*.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and what's next. In *Proc. of ACL*.

Donald O. Hebb. 1949. *The organization of behavior: A neuropsychological theory*. Wiley, New York.

Geoffrey Hinton. 2012. Neural networks for machine learning. *Coursera*, video lectures.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NeurIPs Deep Learning and Representation Learning Workshop*.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2020. Axial attention in multidimensional transformers. *arXiv: 1912.12180*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. 2008. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proc. of ICML*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proc. of ACL*.

Phu Mon Htut, Kyunghyun Cho, and Samuel Bowman. 2018. Grammar induction with neural language models: An unusual replication. In *Proc. of EMNLP*.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proc. of KDD*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proc. of ACL*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with Gumbel-Softmax. arXiv:1611.01144.

Frederick Jelinek and John D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–353.

Yangfeng Ji and Noah A. Smith. 2017. Neural discourse structure for text categorization. In *Proc. of ACL*.

Michael I. Jordan. 1989. Serial order: A parallel, distributed processing approach. In *Advances in Connectionist Theory: Speech*. Erlbaum.

Armand Joulin and Tomás Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Proc. of NeurIPS*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. arXiv:1602.02410.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2021a. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. In *Proc. of ICLR*.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith. 2021b. Finetuning pretrained transformers into rnns. In *Proc. of EMNLP*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and Francois Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proc. of ICML*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *Proc. of ICLR*.

Yoon Kim, Alexander M. Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. In *Proc. of NAACL*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *Proc. of ICLR*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proc. of ICLR*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bo-

jar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL*.

John F. Kolen. 1993. Fool's gold: Extracting finite state machines from recurrent network dynamics. In *Proc. of NeurIPS*.

Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Segmental recurrent neural networks. In *Proc. of ICLR*.

Werner Kuich and Arto Salomaa, editors. 1986. *Semirings, Automata, Languages*. Springer-Verlag.

Shankar Kumar and William Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proc. of NAACL*.

Karl Kunisch and Thomas Pock. 2013. A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences*, 6(2):938–983.

Hung Le, Truyen Tran, and Svetha Venkatesh. 2020. Self-attentive associative memory. In *Proc. of ICML*.

Yann LeCun. 1998. Gradient-based Learning Applied to Document Recognition. In *Proc. of the IEEE*.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proc. of ICML*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017a. End-to-end neural coreference resolution. In *Proc. of EMNLP*.

Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2017b. Recurrent additive networks. *arXiv:1705.07393*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding CNNs for text: non-linear, non-consecutive convolutions. In *Proc. of EMNLP*.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2017a. Deriving neural architectures from sequence and graph kernels. In *Proc. of ICML*.

Tao Lei, Hrishikesh Joshi, Regina Barzilay, Tommi Jaakkola, Kateryna Tymoshenko, Alessandro Moschitti, and Lluís Màrquez. 2016. Semi-supervised question retrieval with gated convolutions. In *Proc. of NAACL*.

Tao Lei, Yu Zhang, and Yoav Artzi. 2017b. Training RNNs as fast as CNNs. arXiv:1709.02755.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *Proc. of EMNLP*.

Moshe Leshno and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A* CCG parsing and semantic role labelling. In *Proc. of EMNLP*.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proc. of NeurIPS*.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. 2018. Independently recurrent neural network (IndRNN): Building A longer and deeper RNN. In *Proc. of CVPR*.

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. 2017. Adversarial ranking for language generation. In *Proc. of NeurIPS*.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In *Proc. of ICLR*.

Yang Liu and Mirella Lapata. 2018. Learning structured text representations. *TACL*, 6:63–75.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv: 1907.11692*.

David G. Luenberger and Yinyu Ye. 2015. *Linear and Nonlinear Programming*. Springer.

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. Luna: Linear unified nested attention. In *Proc. of NeurIPS*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proc. of ACL*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. Stanford typed dependencies manual. Technical report, Stanford University.

André F. T. Martins and Mariana S. C. Almeida. 2014. Priberam: A Turbo semantic parser with second order features. In *Proc. of SemEval*.

André F. T. Martins, Miguel B. Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. of ACL*.

André F. T. Martins and Ramon Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*.

André F. T. Martins, Mário A. T. Figueiredo, Pedro M. Q. Aguiar, Noah A. Smith, and Eric P. Xing. 2015. AD3: Alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16:495–545.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Polyhedral outer approximations with application to natural language parsing. In *Proc. of ICML*.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of RecSys*.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *Proc. of ICLR*.

Arthur Mensch and Mathieu Blondel. 2018. Differentiable dynamic programming for structured prediction and attention. *arXiv:1802.03676*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *Proc. of ICLR*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Yajie Miao, Jinyu Li, Yongqiang Wang, Shi-Xiong Zhang, and Yifan Gong. 2016. Simplifying long short-term memory acoustic models for fast training and decoding. In *Proc. of ICASSP*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proc. of NeurIPS*.

Thomas Miconi, Kenneth Stanley, and Jeff Clune. 2018. Differentiable plasticity: training plastic neural networks with backpropagation. In *Proc. of ICML*.

Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. 2018. Document-level neural machine translation with hierarchical attention networks. In *Proc. of EMNLP*.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michaël Mathieu, and Marc'Aurelio Ranzato. 2014. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Abdelrahman Mohamed, Dmytro Okhonko, and Luke Zettlemoyer. 2019. Transformers with convolutional context for ASR. *arXiv: 1904.11660*.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Darren Moore, John Dines, Mathew Magimai-Doss, Jithendra Vepa, Octavian Cheng, and Thomas Hain. 2006. Juicer: A weighted finite-state transducer speech decoder. In *Proc. of MLMI*.

Lili Mou, Hao Peng, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2015. Discriminative neural sentence modeling by tree-based convolution. In *Proc. of EMNLP*.

Sebastian Nagel. 2016. News dataset available. https://commoncrawl.org/2016/10/news-dataset-available/.

Nikita Nangia and Samuel Bowman. 2018. ListOps: A diagnostic dataset for latent tree learning. In *Proc. of NAACL tudent Research Workshop*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. arXiv:1701.03980.

Vlad Niculae and Mathieu Blondel. 2017. A regularized framework for sparse and structured neural attention. In *Proc. of NeurIPS*.

Vlad Niculae, André F. T. Martins, Mathieu Blondel, and Claire Cardie. 2018. SparseMAP: Differentiable sparse structured inference. arXiv:1802.04223.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Zdeňka Urešová. 2016. Towards comparability of linguistic graph banks for semantic parsing. In *Proc. of LREC*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*.

Stephan Oepen, Lilja Øvrelid, Jari Björne, Richard Johansson, Emanuele Lapponi, Filip Ginter, and Erik Velldal. 2017. The 2017 shared task on extrinsic parser evaluation. towards a reusable community infrastructure. In *Proc. of the 2017 Shared Task on Extrinsic Parser Evaluation*.

Junier Oliva, William Neiswanger, Barnabas Poczos, Eric Xing, Hy Trac, Shirley Ho, and Jeff Schneider. 2015. Fast function to function regression. In *Proc. of AISTATS*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proc. of WMT*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proc. of ACL*.

Bryan Pardo and William Birmingham. 2005. Modeling form for on-line following of musical performances. In *Proc. of AAAI*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proc. of EMNLP*.

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. 2020. Stabilizing transformers for reinforcement learning. In *Proc. of ICML*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *Proc. of ICML*.

Hao Peng, Jungo Kasai, Nikolaos Pappas, Dani Yogatama, Zhaofeng Wu, Lingpeng Kong, Roy Schwartz, and Noah A. Smith. 2022. ABC: Attention with bounded-memory control. In *Proc. of ACL*.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. 2021. Random feature attention. In *Proc. of ICLR*.

Hao Peng, Roy Schwartz, Dianqi Li, and Noah A. Smith. 2020. A mixture of h - 1 heads is better than h heads. In *Proc. of ACL*.

Hao Peng, Roy Schwartz, and Noah A. Smith. 2019. PaLM: A hybrid parser and language model. In *Proc. of EMNLP*.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018a. Rational recurrences. In *Proc. of EMNLP*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proc. of ACL*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2018b. Backpropagating through structured argmax using a spigot. In *Proc. of ACL*.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018c. Learning joint semantic parsers from disjoint data. In *Proc. of NAACL*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proc. of EMNLP*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

B. T. Polyak and A. B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proc. of WMT*.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *American Journal of Computational Linguistics*, 34(2):257–287.

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. 2020. Blockwise self-attention for long document understanding. In *Findings of EMNLP*.

Dragomir R. Radev, Pradeep Muthukrishnan, and Vahed Qazvinian. 2009. The ACL Anthology network. In *Proc. of the Workshop on Text and Citation Analysis for Scholarly Digital Libraries*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI Blog.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *Proc. of ICLR*.

Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *Proc. of NeurIPS*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proc. of EMNLP*.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proc. of NAACL*.

Ankit Singh Rawat, Jiecao Chen, Felix Xinnan X Yu, Ananda Theertha Suresh, and Sanjiv Kumar. 2019. Sampled softmax with random Fourier features. In *Proc. of NeurIPS*.

Corentin Ribeyre, Éric Villemonte De La Clergerie, and Djamé Seddah. 2015. Because syntax does matter: Improving predicate-argument structures parsing using syntactic features. In *Proc. of NAACL*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Dan Roth and Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proc. of NAACL*.

Michael Roth and Mirella Lapata. 2016. Neural semantic role labeling with dependency path embeddings. In *Proc. of ACL*.

Aurko Roy, Mohammad Taghi Saffar, David Grangier, and Ashish Vaswani. 2020. Efficient content-based sparse attention with routing transformers. *arXiv: 2003.05997*.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 105–174. Springer Berlin Heidelberg, Berlin, Heidelberg.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv: 1910.01108*.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *Proc. of ICML*.

J. Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

J. Schmidhuber. 1993. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In *Proc. of ICANN*.

Marcel Paul Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270.

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. SoPa: Bridging CNNs, RNNs, and weighted finite-state machines. In *Proc. of ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of ACL*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In *Proc. of AAAI*.

Yikang Shen, Zhouhan Lin, Chin wei Huang, and Aaron Courville. 2018a. Neural language modeling by jointly learning syntax and lexicon. In *Proc. of ICLR*.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. 2018b. Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proc. of ICLR*.

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

David R. So, Quoc V. Le, and Chen Liang. 2019. The evolved transformer. In *Proc. of ICML*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proc. of ACL*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proc. of ACL*.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. 2011. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proc. of ACL*.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proc. of EMNLP*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *Proc. of ACL*.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. 2021. Not all memories are created equal: Learning to forget by expiring. In *Proc. of ICML*.

Yitong Sun. 2019. *Random Features Methods in Supervised Learning*. Ph.D. thesis, The University of Michigan.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proc. of CoNLL*.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A. Smith. 2017. Frame-semantic parsing with softmax-margin segmental RNNs and a syntactic scaffold. arXiv:1706.09528.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. 2018. Syntactic scaffolds for semantic structures. In *Proc. of EMNLP*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. of ACL*.

Sho Takase, Jun Suzuki, and Masaaki Nagata. 2018. Direct output connection for a high-rank language model. In *Proc. of EMNLP*.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *arXiv: 2005.00743*.

Yi Tay, Dara Bahri, Liu Yang, Don Metzler, and Da-Cheng Juan. 2020b. Sparse sinkhorn attention. In *Proc. of ICML*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena: A benchmark for efficient transformers. In *Proc. of ICLR*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020c. Efficient transformers: A survey. *arXiv: 2009.06732*.

David S. Touretzky. 2013. Computational models of neural systems. https://www.cs.cmu.edu/afs/cs/academic/class/15883-f13/slides/matrix-memory.pdf. Accessed: 2022–08-16.

Trieu H. Trinh and Quoc V. Le. 2018. A simple method for commonsense reasoning.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proc. of EMNLP*.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proc. of ICML*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

Ingmar Visser, Maartje EJ Raijmakers, and Peter CM Molenaar. 2001. Hidden markov model interpretations of neural networks. In *Connectionist Models of Learning, Development and Evolution*, pages 197–206. Springer.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. In *Proc. of NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proc. of ICLR*.

Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. 2020a. Cluster-Former: Clustering-based sparse transformer for long-range dependency encoding. *Findings of ACL*.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *arXiv: 2006.04768*.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proc. of ACL*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *Proc. of ICLR*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proc. of NAACL*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Ronald J. Williams and Jing Peng. 1990. An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *Proc. of ICLR*.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *Proc. of ICLR*.

Zhaofeng Wu, Hao Peng, and Noah A. Smith. 2021. Infusing Finetuning with Semantic Dependencies. *TACL*, 9:226–242.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015. Classifying relations via long short term memory networks along shortest dependency paths. In *Proc. of EMNLP*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2019. Breaking the softmax bottleneck: A high-rank RNN language model. In *Proc. of ICLR*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proc. of NAACL*.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to compose words into sentences with reinforcement learning. In *Proc. of ICLR*.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. 2018. Memory architectures in recurrent neural network language models. In *Proc. of ICLR*.

Weiqiu You, Simeng Sun, and Mohit Iyyer. 2020. Hard-coded Gaussian attention for neural machine translation. In *Proc. of ACL*.

Felix Xinnan X Yu, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. 2016. Orthogonal random features. In *Proc. of NeurIPS*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. *arXiv: 2007.14062*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv:1409.2329*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proc. of EMNLP*.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. In *Proc. of NeurIPS*.

Xun Zhang, Yantao Du, Weiwei Sun, and Xiaojun Wan. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics*, 42(3):353–389.

Yuan Zhang, Chengtao Li, Regina Barzilay, and Kareem Darwish. 2015. Randomized greedy inference for joint segmentation, POS tagging and dependency parsing. In *Proc. of NAACL*.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proc. of ACL*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proc. of ICCV*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *Proc. of ICLR*.

Chapter A

Supplementary Materials for Random Feature Attention

A.1 Random Feature Attention in More Detail

A.1.1 Variance of Random Fourier Features

The following result is due to Yu et al. (2016). Using the same notation as in §4.2.2:

$$\mathrm{Var}(\phi(\mathbf{x}) \cdot \phi(\mathbf{y})) = \frac{1}{2D}\left(1 - e^{-z^2}\right)^2, \tag{A.1}$$

where $z = \|\mathbf{x} - \mathbf{y}\| / \sigma$.

A.1.2 Relating Rfa-Gate to Softmax Attention

Drawing inspiration from gated RNNs, §4.3.2 introduces a gated variant of RFA. Now we study its "softmax counterpart."

$$\begin{aligned} &\widetilde{\mathbf{k}}_i = \mathbf{k}_i\,(1 - g_i)\prod_{j=i+1}^{t} g_j, \quad \widetilde{\mathbf{v}}_i = \mathbf{v}_i\,(1 - g_i)\prod_{j=i+1}^{t} g_j, \quad i = 1, \ldots, t \\ &\mathbf{h}_t = \mathrm{attn}(\mathbf{q}_t, \{\widetilde{\mathbf{k}}_i\}_{i \le t}, \{\widetilde{\mathbf{v}}_i\}_{i \le t}). \end{aligned} \tag{A.2}$$

$\mathbf{h}_t$ is the output at timestep t and is used for onward computation.

At each step, all prefix keys and values are decayed by a gate value before calculating the attention. This implies that the attention computation for $\mathbf{q}_{t+1}$ *cannot* start until that of $\mathbf{q}_t$ is finished. Combined with the linear complexity of softmax normalization, this amounts to quadratic time in sequence length, even for language modeling training.

The above model is less intuitive and more expensive in practice, without the RFA perspective. This shows that RFA brings some benefits in developing new attention models.

# Random Matrices	1	50	100	200
BLEU	24.0	25.7	25.8	25.8

Table A.1: WMT14 EN-DE development set performance varying the number of random matrices to sample from during training. No beam search or checkpoint averaging is used.

A.2 Experimental Details

During training, we sample a different random projection matrix for each attention head. Preliminary experiments suggest this performs better than using the same random projection throughout training (Table A.1). Our conjecture is that this helps keep the attention heads from "over committing" to any particular random projection (Peng et al., 2020). To avoid the overhead of sampling from Gaussian during training, we do this in an offline manner. I.e., before training we construct a pool of random matrices (typically 200), at each training step we draw from the pool. At test time each attention head uses the same random projection, since no accuracy benefit is observed by using different ones for different test instances.

A.2.1 Language Modeling

We compare the models using two model size settings, summarized in Table A.2. We use the fixed sinusoidal position embeddings by Vaswani et al. (2017). All models are trained for up to 150K gradient steps using the Adam optimizer (Kingma and Ba, 2015). No ℓ_2-regularization is used. We apply early stopping based on development set perplexity. All models are trained using 16 TPU v3 accelerators, and tested using a single TPU v2 accelerator.

Hyperprams.	Small	Big
# Layers	6	16
# Heads	8	16
Embedding Size	512	1024
Head Size	64	64
FFN Size	2048	4096
Batch Size	64	64
Learning Rate	$[1 \times 10^{-4}, 2.5 \times 10^{-4}, 5 \times 10^{-4}]$	
Warmup Steps	6000	6000
Gradient Clipping Norm	0.25	0.25
Dropout	[0.05, 0.1]	[0.2, 0.25, 0.3]
Random Feature Map Size	64	64

Table A.2: Hyperparameters used in the language modeling experiments.

A.2.2 Machine Translation

WMT14. We use the fixed sinusoidal position embeddings by Vaswani et al. (2017). For both EN-DE and EN-FR experiments, we train the models using the Adam (with $\beta_1 = 0.1$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$) optimizer for up to 350K gradient steps. We use a batch size of 1,024 instances for EN-DE, while 4,096 for the much larger EN-FR dataset. The learning rate follows that by Vaswani et al. (2017). Early stopping is applied based on development set BLEU. No ℓ_2 regularization or gradient clipping is used. All models are trained using 16 TPU v3 accelerators, and tested using a single TPU v2 accelerator. Following standard practice, we average 10 most recent checkpoints at test time. We evaluate the models using SacreBLEU (Post, 2018).[1] A beam search with beam size 4 and length penalty 0.6 is used. Other hyperparameters are summarized in Table A.3.

Hyperprams.	WMT14	IWSLT14
# Layers	6	6
# Heads	8	8
Embedding Size	512	512
Head Size	64	64
FFN Size	2048	2048
Warmup Steps	6000	4000
Dropout	0.1	0.3
Cross Attention Feature Map	128	128
Causal Attention Feature Map	64	64

Table A.3: Hyperparameters used in the machine translation experiments.

[1] https://github.com/mjpost/sacrebleu

A.3 Effect of Random Feature Size

This section studies how the size of $\phi(\cdot)$ affects the performance. Table A.4 summarize RFA-Gaussian's performance on WMT14 EN-DE development set. The model and training are the same as that used in §4.4.2 except random feature size. Recall from §4.2.2 that the size of $\phi(\cdot)$ is $2D$ for RFA-Gaussian. When the size of $\phi(\cdot)$ is too small (32 or 64 for cross attention, 32 for causal attention), training does not converge. We observe accuracy improvements by using random features sufficiently large (256 for cross attention and 128 for causal attention); going beyond that, the benefit is marginal.

ϕ **Size**	32	64	128	256	512
BLEU	N/A	N/A	24.9	25.8	26.0

(a) Varying cross attention ϕ sizes while fixing that of causal attention to be 128.

ϕ **Size**	32	64	128	256	512
BLEU	N/A	25.3	25.8	25.8	25.6

(b) Varying causal attention ϕ sizes while fixing that of cross attention to be 256.

Table A.4: WMT14 EN-DE development set performance of RFA-Gaussian (the size of ϕ is $2D$; §4.2.2) varying the random feature sizes. N/A indicates training does not converge. No beam search or checkpoint averaging is used.

Chapter B

Supplementary Materials for ABC: Attention with Bounded-memory Control

B.1 Experimental Details

B.1.1 Language Modeling

We closely build on Baevski and Auli (2019) and Kasai et al. (2021b). The hyperparameters are summarized in Table B.1. All models are trained on 4 A100 GPUs.

B.1.2 Machine Translation

We experiment with a sentence-level (WMT14 EN-DE, Bojar et al., 2014) and a document-level benchmark (IWSLT14 ES-EN, Cettolo et al., 2014) to assess model performance over various sequence lengths. The preprocessing and data splits of WMT14 EN-DE follow Vaswani et al. (2017). A 32,768 byte pair encoding (BPE; Sennrich et al., 2016) vocabulary is shared between source and target languages. For IWSLT14, we follow Miculicich et al. (2018) and use the *dev2010* subset for development and *tst2010-2012* for testing. The tokenization is also the same as Miculicich et al. (2018): we tokenize and truecase Spanish and English with Moses (Koehn et al., 2007) and run byte-pair encoding with 30k splits, shared between the two languages. The final dataset contains

1421, 8, and 42 documents for training, development, and testing. On average, each document contains 126.7 sentences, and each sentence contains 21.7(ES)/22.5(EN) BPE subwords. We use a sliding window with length-4 and stride-one to generate our dataset. During inference, we use predicted context on the target side.

We average the checkpoints from the last five epochs to obtain the final model (Vaswani et al., 2017). In inference, we apply beam search with size 5 and length penalty 0.6. Other hyperparameters are summarized in Table B.2. All models are trained on 4 RTX 2080 Ti GPUs.

B.1.3 Masked Language Model Finetuning

Our data for continued pretraining is a concatenation of BookCorpus (Zhu et al., 2015), English Wikipedia, OpenWebText (Gokaslan and Cohen, 2019), and RealNews (Zellers et al., 2019). Our data differs from RoBERTa's pretraining data, which we do *not* have access to. We replace their CC-News (Nagel, 2016) with RealNews, and drop Stories (Trinh and Le, 2018). At the time of this project, the public access to the Stories dataset is broken.[1] Our machine does *not* have a large enough memory to load all the data. We therefore split the training data into 20 shards, after shuffling. Other preprocessing is the same as Liu et al. (2019).[2] The hyperparameters for continued pretraining follow base-sized RoBERTa, part of which are summarized in Table B.3. All models are trained on a single TPU v3 accelerator.

For downstream task finetuning, we use the same hyperparameters as Liu et al. (2019).[3] Table B.4 briefly describes the tasks. The readers are referred to Wang et al. (2019) for futher details.

[1] https://console.cloud.google.com/storage/browser/commonsense-reasoning/reproduce/stories_corpus?pli=1
[2] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.pretraining.md
[3] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.glue.md

Hyperprams.	B&A	Kasai
# Layers	16	32
# Heads	8	8
Embedding Size	1024	1024
Head Size	128	128
FFN Size	4096	4096
Batch Size	64	64
Learning Rate	1.0	1.0
Dropout	0.3	0.3
Layer Dropout	-	0.2
Memory size	[32, 64]	64

Table B.1: Hyperparameters used in the language modeling experiments. B&A: Baevski and Auli (2019); Kasai: Kasai et al. (2021b).

Hyperprams.	WMT14	IWSLT14
# Layers	6	6
# Heads	8	8
Embedding Size	512	512
Head Size	64	64
FFN Size	2048	1024
Warmup Steps	6000	4000
Dropout	0.1	0.3
Cross Attn. n	32	128
Causal Attn. n	8	64

Table B.2: Hyperparameters used in the machine translation experiments.

Hyperprams.	Values
# Layers	12
# Heads	12
Embedding Size	768
Head Size	64
FFN Size	3072
Dropout	0.1
Memory Size	[64, 128]

Table B.3: Hyperparameters for continued pretraining in the masked language model finetuning experiments.

Data	Task	Train	Dev.
MNLI	Entailment	392K	9.8K
QNLI	Entailment	105K	5.5K
QQP	Paraphrase	363K	40K
SST-2	Sentiment	67K	873

Table B.4: GLUE datasets and statistics. MNLI: Williams et al. (2018); QNLI is compiled by GLUE's authors using Rajpurkar et al. (2016); QQP: Csernai (2017, accessed September 1, 2020); SST-2: Socher et al. (2013).

Chapter C

Supplementary Materials for Backpropagating through Structured Argmax using a SPIGOT

C.1 Implementation Details

Our implementation is based on the DyNet toolkit.[1] We use part-of-speech tags and lemmas predicted by NLTK.[2]

C.1.1 Syntactic-then-Semantic Parsing Experiment

Each input token is represented as the concatenation a word embedding vector, a learned lemma vector, and a learned vector for part-of-speech, all updated during training. In joint training, we apply early-stopping based on semantic dependency parsing development performance (in labeled F_1). We do not use mini-batch. We set the step size η for SPIGOT to 1.

Semantic dependency parser. We use the pruning techniques in Martins and Almeida (2014), and replace their feature-rich model with neural networks (Peng et al., 2018c). We observe that the number of parts surviving pruning is linear in the sentence length (5.5× on average), with

[1] https://github.com/clab/dynet
[2] http://www.nltk.org/

$\sim$99% recall.

We do not deviate far from the hyperparameter setting in Peng et al. (2017), with the only exception being that we use 50-dimensional lemma and part-of-speech embeddings, instead of 25.

Syntactic dependency parser. For the max-margin syntactic parsers used in PIPELINE, STE, and SPIGOT, we use the hyperparameters reported in Kiperwasser and Goldberg (2016), but replace their 125-dimensional BiLSTMs with 200-dimensional ones, and use 50-dimensional POS embeddings, instead of 25. We anneal the learning rate at a rate of 0.5 every 5 epochs.

For the marginal syntactic parser in SA, we follow the use of Adam algorithm (Kingma and Ba, 2015), but set a smaller initial learning rate of 5×10^{-4}, annealed at a rate of 0.5 every 4 epochs. The rest of the hyperparameters stay the same as the max-margin parser.

C.1.2 Semantic Parsing and Sentiment Classification Experiment

The model is trained for up to 30 epochs in the joint training stage. We apply early-stopping based on sentiment classification development accuracy. For semantic dependency parser, we follow the hyperparameters described in §C.1.1.

Sentiment classifier. We use 300-dimensional `GloVe` (Pennington et al., 2014) to initialize word embeddings, fixed during training. We use a single-layer BiLSTM, followed by a two-layer ReLU-MLP. Dropout in word embeddings and MLPs is applied, but not in LSTMs. We use Adam algorithm (Kingma and Ba, 2015), and follow the default procedures by DyNet for optimizer settings and parameter initializations. An ℓ_2-penalty of 10^{-6} is applied to all weights. Learning rate is annealed at a rate of 0.5 every 5 epochs. We use mini-batches of 32, and clip the ℓ_2-norm of gradients to 5 (Graves, 2013). We set the step size η for SPIGOT to $\frac{5}{32}$. We explore the same set of hyperparameters based on development performance for all compared models, summarized in Table C.1.

Hyperparameter	Values
MLP dimension	$\{100, 150, 200, 250, 300\}$
BiLSTM dimension	$\{100, 150, 200, 250, 300\}$
Embedding dropout	$\{0.2, 0.3, 0.4, 0.5\}$
MLP dropout	$\{0.0, 0.1, 0.2, 0.3, 0.4\}$

Table C.1: Hyperparameters explored in sentiment classification experiments.

Chapter D

Supplementary Materials for PaLM: A Hybrid Parser and Language Model

D.1 Implementation Details

Neural Network Architecture Our implementation is based on AWD-LSTM (Merity et al., 2018).[1] It uses a three-layer LSTM, with carefully designed regularization techniques. PaLM includes the span attention after the second layer. Preliminary results show that it yields similar results, but is less sensitive to hyperparameters, compared to adding it to the last layer.

The context is concatenated to the hidden state ($\bar{\mathbf{h}}_t = [\mathbf{h}_t; \mathbf{a}_t]$), and then fed to a tanh-MLP controlled by a residual gate $\mathbf{g}_r$ (He et al., 2016), before fed onward into the next LSTM layer:

$$\hat{\mathbf{h}}_t = \mathbf{g}_r \odot \text{MLP}\left(\bar{\mathbf{h}}_t\right) + (\mathbf{1} - \mathbf{g}_r) \odot \mathbf{h}_t. \tag{D.1}$$

The rest of the architecture stays the same as AWD-LSTM. We refer the readers to Merity et al. (2018) for more details.

More details on PaLM-S. PaLM-S uses exactly the same architecture and hyperparameters as its unsupervised counterpart. We derive, from PTB training data, a m-dimensional 0-1 vector for each token. Each element specifies whether the corresponding span appears in the gold parse.

[1] https://github.com/salesforce/awd-lstm-lm

Type	Values
Rational RNN size	200
Context Vector Size	400
LSTM Hidden Size	1020
Weight Dropout	0.45
Vertical Dropout	0.2

Table D.1: The hyperparameters used in the PTB language modeling experiment.

Trivial spans (i.e., the ones over single tokens and full sentences) are ignored. The vector are normalized to sum to one, in order to facilitate the use of cross-entropy loss. λ in Eq. 3.7 is set to 0.01.

Hyperparameters. The regularization and hyperparameters largely follow Merity et al. (2018). We only differ from them by using smaller hidden size (and hence smaller dropout rate) to control for the amount of parameters in the PTB experiments, summarized in Table D.1 For the WikiText-2 experiments, we use 200 rational RNN size and 400 dimensional context vectors. Other hyperparameters follow Merity et al. (2018). The max span length m is set to 20 for PTB experiments, and 10 for WikiText-2.

Merity et al. (2018) start by using SGD to train the model, and switch to averaged SGD (Polyak and Juditsky, 1992) after 5 nonimprovement-epochs. We instead use Adam (Kingma and Ba, 2015) with default PyTorch settings to train the model for 40 epochs, and then switch to ASGD, allowing for faster convergence.

D.2 Span Representations

Below is the derivation for Eq. 3.4.

$$
\begin{aligned}
\overrightarrow{\mathbf{c}}_{i,j} &= \overrightarrow{\mathbf{u}}_j + \sum_{k=i}^{j-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{u}}_j + \sum_{k=1}^{j-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell - \sum_{k=1}^{i-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{c}}_j - \left(\overrightarrow{\mathbf{u}}_{i-1} + \sum_{k=1}^{i-2} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{i-1} \overrightarrow{\mathbf{f}}_\ell \right) \bigodot_{\ell=i}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \bigodot_{k=i}^{j} \overrightarrow{\mathbf{f}}_k
\end{aligned}
$$

Chapter E

Supplementary Materials for Rational Recurrences

E.1 Proof of Proposition 13

Proof. Let's consider a single-layer QRNN with 2-window convolutions:

$$\begin{aligned}
\mathbf{f}_t &= \boldsymbol{\sigma}\left(\mathbf{V}_f \mathbf{v}_{t-1} + \mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\right), \\
\mathbf{u}_t &= (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{V}_u \mathbf{v}_{t-1} + \mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big), \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t.
\end{aligned}$$

A similar analysis applies to T-GRUs and T-LSTMs directly, and it should be straightforward to generalize the discussion to QRNNs with larger convolution windows.

Let Σ denote the alphabet, and let $\mathbf{x} = x_1 x_2 \dots x_n \in \Sigma^+$ be a nonempty input string. consider a WFSA over the real semiring with $2\,|\Sigma| + 1$ states, where q_0 is the initial state with $\lambda(q_0) = 1$; $|\Sigma|$ of them are final states $\mathcal{Q}_2 = \{q_\alpha\}_{\alpha \in \Sigma}$, with $\rho(q_\alpha) = 1$, and the remaining $|\Sigma|$ states are denoted by $\mathcal{Q}_1 = \{p_\alpha\}_{\alpha \in \Sigma}$.

The transition weights τ are constructed by

$$\begin{aligned}
\tau(q_0, p_\alpha, \alpha) &= 1, && \forall p_\alpha, \in \mathcal{Q}_1; \\
\tau(p_\alpha, p_\beta, \beta) &= 1, && \forall p_\alpha, p_\beta \in \mathcal{Q}_1; \\
\tau(p_\alpha, q_\beta, \beta) &= \mu_\alpha(\beta), && \forall p_\alpha \in \mathcal{Q}_1, \forall q_\beta \in \mathcal{Q}_2; \\
\tau(q_\alpha, q_\beta, \beta) &= \phi_\alpha(\beta), && \forall q_\alpha, q_\beta \in \mathcal{Q}_2.
\end{aligned}$$

$\tau = 0$ otherwise. Then one dimension of the reccurent updates of a 2-window QRNN is recovered by parameterizing the weight functions as

$$\mu_{x_{t-1}}(x_t) = [\mathbf{u}_t]_i, \quad \phi_{x_{t-1}}(x_t) = [\mathbf{f}_t]_i. \tag{E.1}$$

The recurrent computation of a 2-window QRNN of hidden size d can then be recovered by collecting d such WFSAs. □

E.2 Proof of Proposition 14

Proof. We present the construction of WFSAs for a single layer n-gram RCNNs of hidden size d.

Let's assume a given input sequence $\mathbf{x} \in \Sigma^+$, with $|\mathbf{x}| > n$, since otherwise one only needs include paddings, just as in a RCNN. Consider a WFSA with $n+1$ states $\mathcal{Q} = \{q_i\}_{i=0}^n$ over the real semiring. Use q_0 as the initial state with $\lambda(q_0) = 1$, and q_n as the final state with $\rho(q_n) = 1$. The transition weight function is defined by

$$\tau(q_i, q_j, \alpha) = \begin{cases} 1, & i = j = 0, \\ \phi(\alpha), & j = i > 0, \\ \mu_j(\alpha), & j = i + 1, \\ 0, & \text{otherwise.} \end{cases}$$

Let $z_t^{(j)}$ denote the total score of all paths landing in state q_j *just* after consuming x_t. Let

$z_t^{(j)} = 0, j = 0, \ldots, n$. By the forward algorithm

$$
\begin{aligned}
z_t^{(0)} &= 1, \\
z_t^{(j)} &= z_{t-1}^{(j)} \phi(x_t) + z_{t-1}^{(j-1)} \mu_j(x_t), \quad j \geq 1.
\end{aligned}
$$

Applying similar parametrization to that in §6.4.2, $z_t^{(n)}$ recovers one dimension of the recurrence. Collecting d such WFSAs we recover the recurrence of a single layer n-gram RCNNs, with $\boldsymbol{\lambda}_t$ being a constant, or depending only on x_t. □

E.3 Proof of Proposition 15

Proof. Closely following the 2-dimensional case in §6.4.3, let's discuss a single layer ISAN of hidden size d.

Consider a WFSA over the real semiring with $2d$ states. Let d of them, denoted by $\mathcal{Q}_2 = \{q_i\}_{i=d+1}^{2d}$ be the initial states, with $\lambda(q_i) = 1, i = 1, \ldots, d$. Denote the other half $\{q_i\}_{i=1}^{d}$ by $\mathcal{Q}_1$. Define transition weight τ by:

$$
\tau(q_i, q_j, \alpha) = \begin{cases} 1, & i = j,\ q_i \in \mathcal{Q}_2, \\ \eta_j(\alpha), & i = j + d, \\ \mu_{j,i}(\alpha), & q_i, q_j \in \mathcal{Q}_1, \\ 0, & \text{otherwise.} \end{cases}
$$

$\forall \alpha \in \Sigma$.

Using $q_i \in \mathcal{Q}_1$ as the final state with $\rho(q_i) = 1$, and denote the resulting WFSA by $\mathcal{G}_i$. By Forward algorithm, $\mathcal{G}_i$ recovers the ith dimension of the single layer ISAN by letting $[\mathbf{W}_{x_t}]_{i,j} = \mu_{i,j}(x_t)$, and $[\mathbf{b}_{x_t}]_i = \eta_i(x_t)$; the d-dimensional recurrent computation is recovered by a set of WFSAs $\{\mathcal{G}_i\}_{i=1}^{d}$ constructed similarly. □

E.4 Compared Models

This section formally describes the models compared in the experiments (§6.6.1).

rrnn($\mathcal{B}$). RRNN($\mathcal{B}$) is derived from $\mathcal{B}$ (§6.4.1).

$$\mathbf{f}_t = \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{E.2a}$$

$$\mathbf{u}_t = (1 - \mathbf{f}_t) \odot \mathbf{W}_u \mathbf{v}_t, \tag{E.2b}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t, \tag{E.2c}$$

$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{E.2d}$$

$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{E.2e}$$

rrnn($\mathcal{B}$)$_{\mathbf{m+}}$. Also derived from $\mathcal{B}$, but uses the max-plus semiring (§6.5.2).

$$\mathbf{f}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{E.3a}$$

$$\mathbf{u}_t = \mathbf{W}_u \mathbf{v}_t, \tag{E.3b}$$

$$\mathbf{c}_t = \max\{\mathbf{f}_t + \mathbf{c}_{t-1}, \mathbf{u}_t\}, \tag{E.3c}$$

$$\mathbf{o}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{E.3d}$$

$$\mathbf{h}_t = \tanh(\mathbf{o}_t + \mathbf{c}_t). \tag{E.3e}$$

rrnn($\mathcal{C}$). RRNN($\mathcal{C}$) is derived from $\mathcal{C}$ (§6.4.2):

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)} \mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \quad j = 1, 2, \tag{E.4a}$$

$$\mathbf{u}_t^{(j)} = (1 - \mathbf{f}_t^{(j)}) \odot \mathbf{W}_u^{(j)} \mathbf{v}_t, \quad j = 1, 2, \tag{E.4b}$$

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{E.4c}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}, \tag{E.4d}$$

$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{E.4e}$$

$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{E.4f}$$

rrnn($\mathcal{F}$). Derived from $\mathcal{F}$ (§6.5.1).

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)}\mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \quad j = 1, 2, \tag{E.5a}$$

$$\mathbf{u}_t^{(j)} = (1 - \mathbf{f}_t^{(j)}) \odot \mathbf{W}_u^{(j)}\mathbf{v}_t, \quad j = 1, 2, \tag{E.5b}$$

$$\mathbf{p}^{(j)} = \boldsymbol{\sigma}(\mathbf{b}_p^{(j)}), \quad j = 1, 2, \tag{E.5c}$$

$$\mathbf{r} = \boldsymbol{\sigma}(\mathbf{b}_r), \tag{E.5d}$$

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{E.5e}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + (\mathbf{c}_{t-1}^{(1)} + \mathbf{r}) \odot \mathbf{u}_t^{(2)}, \tag{E.5f}$$

$$\mathbf{c}_t = \mathbf{p}^{(1)} \odot \mathbf{c}_t^{(1)} + \mathbf{p}^{(2)} \odot \mathbf{c}_t^{(2)} \tag{E.5g}$$

$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o\mathbf{v}_t + \mathbf{b}_o\big), \tag{E.5h}$$

$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{E.5i}$$

The output gates (Equations E.2d, E.3d, E.4e, and E.5h) are optional. They are only used in language modeling experiments, where we empirically find that they improve performance.

E.5 Experimental Setup

E.5.1 Implementation Details

Our implementation is based on Lei et al. (2017b)[1] and Chapter 2,[2] using PyTorch.[3]

E.5.2 Language Modeling

For hyperparameters, we do not deviate much from the language modeling experiments in Lei et al. (2017b). We change the hidden sizes for all compared models based on the trainable parameter budget, and adjust the dropout probabilities accordingly to keep the number of remaining hidden units is roughly the same in expectation. Besides, we observe that RRNN($\mathcal{C}$) and RRNN($\mathcal{F}$) fail to converge when optimized with the SGD algorithm using 1.0 initial learning rate. And thus we use 0.5 for both models. Other hyperparameters are kept the same as Lei et al. (2017b).

[1] https://github.com/taolei87/sru
[2] https://github.com/Noahs-ARK/SPIGOT
[3] https://pytorch.org/

Type	Values
Hidden size	$[100, 300]$
Vertical dropout	$[0.0, 0.5]$
Recurrent dropout	$[0.0, 0.5]$
Embedding dropout	$[0.0, 0.5]$
Learning Rate	$[10^{-2}, 10^{-4}]$
ℓ_2 regularization	$[10^{-5}, 10^{-7}]$
Gradient Clipping	$[1.0, 5.0]$

Table E.1: The hyperparameters explored using random search algorithm in the text classification experiments.

E.5.3 Text classification

We train our models using Adam (Kingma and Ba, 2015) with a batch size of 16 (for Amazon) or 64 (for the smaller datasets). Initial learning rate and ℓ_2 regularization are hyperparameters. We use 300-dimensional GloVe 840B embeddings (Pennington et al., 2014) normalized to unit length and fixed, replacing unknown words with a special UNK token. Two layer RNNs are used in all cases. For regularization, we use three types of dropout: a recurrent variational dropout, vertical dropout and a dropout on the embedding layer.

We tune the hyperparameters of our model on the development set by running 20 epochs of random search. We then take the best development configuration, and train five models with it using different random seeds. We report the average test results. The hyperparameters values explored are summarized in Table E.1. We train all models for 500 epochs, stopping early if development accuracy does not improve for 30 epochs. During training, we halve the learning rate if development accuracy does not improve for 10 epochs.