Learning Representations of Text through Language and Discourse Modeling: From Characters to Sentences

by Yacine Jernite

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2018

Professor David Sontag

Dedication

To my loving family and my wonderful partner.

Acknowledgements

I would first like to thank my advisor David Sontag, who made this work possible and taught me many valuable skills over the last five years. David has been an ever-reliable source of varied and pertinent insights on pretty much every research problem I've ever thought of, and his extensive knowledge and inexhaustible ingenuity have helped me out of several tight spots. His advice has also been inestimable in helping me learn to navigate the world of research, and will no doubt continue to do so in my future scientific endeavors.

My committee is also comprised of Slav Petrov, Sasha Rush, Kyunghyun Cho, and Sam Bowman, who have each at different times helped shape my understanding of the field of natural language processing. From Slav's statistical NLP class when I arrived at NYU in the Fall of 2012, to collaborating with Sasha on my first language papers (and first major coding project!), to several enlightening conversations with Kyunghyun after his arrival at NYU made language a focus of the lab, to my more recent research with Sam which helped me round out this thesis in the best possible way. This work could not have existed in its current form without all of their inputs.

It takes a village to raise a child... or to write a thesis. Too many people to do justice to have contributed to enriching my experience of these five and a half years. I am grateful to all of my co-authors and collaborators during this graduate program. I also want to thank all the friends from my previous lives and the ones I have made since moving to New York whose company I have enjoyed and who have allowed me to keep growing every day beyond my research field.

Finally, I would like to thank my family, whose unwavering support I can always count on. My parents, who have been and continue to be the best role models a person

could hope for. Salim and Kenza, whose drive and intelligence make me always want to be the best version of myself I can imagine. And last but not least, my extraordinary partner Amanda, who has been a light in my life and a beacon of wisdom for these last three years.

Abstract

In this thesis, we consider the problem of obtaining a representation of the meaning expressed in a text. How to do so correctly remains a largely open problem, combining a number of inter-related questions (e.g. what is the role of context in interpreting text? how should language understanding models handle compositionality? etc...) In this work, after reflecting on some of these questions and describing the most common sequence modeling paradigms in use in recent work, we focus on two specifically: what level of granularity text should be read at, and what training objectives can lead models to learn useful representations of a text's meaning.

In a first part, we argue for the use of sub-word information for that purpose, and present new neural network architectures which can either process words in a way that takes advantage of morphological information, or do away with word separations altogether while still being able to identify relevant units of meaning.

The second part starts by arguing for the use of language modeling as a learning objective, and provides algorithms which can help with its scalability issues and propose a globally rather than locally normalized probability distribution. It then explores the question of what makes a good language learning objective, and introduces discriminative objectives inspired by the notion of discourse coherence which help learn a representation of the meaning of sentences.

Statement of Contributions

This thesis is the product of a number of collaborations over the years with several fantastic co-workers. Chapter 2 is built on work which was first published in [Kim et al., 2016] and [Jernite et al., 2016], while chapters 3, 4, and 5 use work which was originally presented in [Jernite et al., 2017b], [Jernite et al., 2015], and [Jernite et al., 2017a] respectively. The work presented in [Kim et al., 2016] was the product of Yoon Kim's work during his Master at NYU under my and David Sontag's supervision: we worked together on the model and methodology, and he wrote the code used in the experiments and most of the paper with help from Sasha Rush, my self and David. Most of the code and writing used in the rest of the thesis is mine, with help from Sasha, Edouard Grave and Haoyue Shi for the experiments of [Jernite et al., 2015], [Jernite et al., 2016], and [Jernite et al., 2017a] respectively. Anna Choromanska was the author of the original proof for the theorem and some of the lemmas in [Jernite et al., 2017a], although all have been modified to some extent for this thesis. Sasha, Anna and Sam Bowman were also involved in the writing of [Jernite et al., 2015], [Jernite et al., 2017b], and [Jernite et al., 2017a], and my advisor David was heavily involved in the writing and editing of all papers except [Jernite et al., 2016] (which was work done during my internship at the Facebook FAIR lab).

Table of Contents

Dedication ii

Acknowledgements iii

Abstract v

Contributions vi

List of Figures x

List of Tables xii

1 Introduction **1**

1.1 Motivation 1

1.2 Overview of the Problem 3

1.3 Representations of Meaning in NLP 5

1.4 Background: Neural Language Modeling 13

1.5 Background: Recurrent Neural Networks 15

1.6 Organization of this Thesis 20

2 Using Sub Word Information **23**

2.1 Introduction 24

2.2 Character-Aware Word Embeddings: Model 29

2.3 Character-Aware Word Embeddings: Experiments 33

2.4 Adaptive Character Level Encoding: Model 45
2.5 Adaptive Character Level Encoding: Experiments 50
2.6 Discussion 58

3 Speeding up Word Level Language Modeling 62

3.1 Introduction 63
3.2 Background 67
3.3 Adaptive Tree Model and Learning Algorithm 76
3.4 Theoretical Properties of the Objective 90
3.5 Experiments 106
3.6 Discussion 112

4 Globally Normalized Language Modeling 114

4.1 Introduction 115
4.2 A Markov Random Field Language Model 116
4.3 Efficient Learning of a Chain MRF 119
4.4 Experiments 130
4.5 Discussion 136

5 Towards Better Learning Objectives 138

5.1 Introduction 139
5.2 Background 142
5.3 Discourse Inspired Objectives 145
5.4 Experiments 148
5.5 Discussion: Further Work on Discourse 152
5.6 Towards Formalizing Meaning for NLP Tasks 153

Conclusion **164**

Bibliography **168**

List of Figures

1.1 RNN language model 19

2.1 Character-aware word embedding architecture 30
2.2 n-gram representations 42
2.3 Two time steps of a VCU 47
2.4 Per-bit computation 55
2.5 Per-character computation 56
2.6 Bits per character on Europarl 57
2.7 Per-character computation: morphoogy 57

3.1 Hierarchical predictor 64
3.2 Hierarchical vs flat language model 65
3.3 Hierarchical language model 66
3.4 J objective 77
3.5 J objective: probas 79
3.6 Tree learning algorithm 83
3.7 J objective: alpha and beta 91
3.8 Tree learned from the Gutenberg corpus 108
3.9 Test perplexity per epoch. 110

4.1 A segment of the linear-chain model over a small corpus. 117
4.2 The cyclic model with $N = 16$ 121
4.3 Covering forests 125
4.4 Lifted inference sub-problem tree 127
4.5 Exact log-partition vs upper bound 131
4.6 Learning with lifted bound vs exact objective 132
4.7 Lifted TRW MRF bound vs feed-forward log-likelihood 133
4.8 POS tagging model for $K = 2, M = 3$ 135

5.1 Joint training of sentence encoder 144
5.2 The *eval function* 159

List of Tables

2.1 Multilingual corpus statistics 34

2.2 Small vs large CharLSTM 36

2.3 CharLSTM: PTB performance 37

2.4 Test set perplexities for DATA-S 38

2.5 Test set perplexities for DATA-L 40

2.6 CharLSTM: nearest neighbor words 41

2.7 Perplexity on Penn Treebank. 43

2.8 Influence of vocabulary and training data size 44

2.9 Music modeling 52

2.10 Average amount of computation 52

2.11 VCRNN perplexity 54

3.1 Classification performance on the YFCC100M dataset 107

3.2 Examples of learned deep nodes 111

3.3 Flat vs hierarchical language modeling 111

4.1 Nearest neighbours in different embeddings 134

4.2 POS tagging accuracy 135

5.1 ORDER objective 145

5.2 NEXT objective 146
5.3 CONJUNCTION objective 147
5.4 List of conjunctions. 147
5.5 Intrinsic evaluation results. 149
5.6 Text classification results 151

Chapter 1

Introduction

1.1 Motivation

One of the greatest driving forces of the advance of civilizations is the ability to share information: individuals who want to advance science and knowledge need to have the means to build on top of their predecessors' and contemporaries' efforts. In Antiquity, great collections and concentrations of expensive books in centers of knowledge such as the famed Alexandria library were instrumental in the progress of technique and thought. In the early Middle Ages, the loss of such centers in Western Europe spelled a period of sharp decline, until those texts were finally re-discovered and made available in the late 13th to early 14th century. More than a century later, the development of a more efficient printing press around 1440 CE by Johannes Gutenberg largely democratized access to written knowledge by reducing the cost of the production and dissemination of texts. This had a profound impact on society, and the influence of this innovation can be traced to the Reformation, the Enlightenment, and much of what marked the beginning of the Modern Era. Further technological advances in the following cen-

turies continued this trend of making information more persistent and readily available, including the development of lithographs, the 19th century Industrial Revolution steam-powered presses, and culminating in the rise of digitization and the Internet in the 1980s and 1990s. Today, the physical cost of accessing or publishing a book, news article or scientific document is next to nothing.

However, as the cost of producing and spreading texts became virtually nil, new problems arose in being able to find and make use of relevant knowledge. In a scientific research setting, for example, every new endeavor begins with questions about the state of the art. Who has considered the same problem? Who has applied similar methods? What were their results? As more and more articles are published every passing day, this information becomes a needle in an increasingly large haystack. But the difficulties do not stop at finding the needle. In some domains, such as medical research, one routinely encounters articles reaching different conclusions in answer to the same question. In such a case, any worthy knowledge needs to be aggregated from an ever increasing number of publications, which requires significant effort.

This difficulty becomes even more apparent when looking beyond scientific research at civil society. The same way that technological advance is supported by an easy access to the work of the whole scientific community, social progress and the good functioning of democracy depend on every citizen's understanding of the institutions that support their societies and of contemporary events. However, journalism has found itself dealing with a significant crisis in recent years, a crisis which has something to do with the shifting of production costs away from the printing of physical newspapers and magazines. We do not know exactly yet the extent of what has been called the "fake news" problem, but what is sure is that information now comes from a variety of sources of varying quality and reliability up to and including public posts on social media, and that

combining all of this information, discerning the true from the false, and linking it to relevant contexts can be a task of awe-inspiring proportions.

Thus, we find that the development of mass writing and production of text, while undoubtedly beneficial to society, raises new questions of its own. If we are to keep making knowledge more available, more accessible, and more useful, we need to develop mass reading: new systems and inventions which can consider and combine all of this available information for us. There have already been some tremendous advances in that direction. Smarter search engines have become a reality of everyday life, and manage to somewhat alleviate the "needle in a haystack" problem we mentioned earlier. But, as noted above, finding the needle is only the start of the problem. In order to combine and compare information for the benefit of the user, a system needs to be able to understand the meaning of a text, and encode it in a usable way. In this thesis, we consider two of the main problems which need to be addressed in order to do so.

1.2 Overview of the Problem

Our goal is then to go from a text, which can be defined as a sequence of characters or words, to a representation of its meaning. A number of questions arise on the way from the one to the other; too many for one thesis, or even one person, to answer. Still, we can start by identifying those questions, and contribute to answering at least some of them.

One of the most important features of language is compositionality. Characters combine into morphemes and words. Words combine into phrases, into sentences, into paragraphs. These combinations are discrete in nature, following processes which are not always explicitly observed. They are sometimes understood: grammar explains the phe-

nomenology of morphemes, syntactic analysis gives us some information on how words combine. On the other hand, some questions are still very much open: how general knowledge about the world or the intended reader of a text affects these combinations, for example.

While the answers to these questions are interesting for their own sake, they also correspond to concrete modeling choices when attempting to design a language understanding system. Do we treat each word we encounter as an independent entity, or do we obtain word-level information by combining characters? Do we want a model which explicitly combines words according to a syntactic tree, or one which has the ability to learn combining schemes by itself? Does the combination happen in a discrete or continuous space? And should we explicitly model general knowledge about the world, or do we expect that the parameters of the model will implicitly encode that knowledge?

A more fundamental question also seems to stand in the way of our objective; namely, what exactly is it that we understand by "meaning"? And tied to that question: what shape or form should our representation take? Should we define meaning in relation to a physical world, or to the writer or reader's state of mind? Should we have a symbol for each concept or entity, or share information across different parameters? Does the representation need to be discrete, continuous, a combination of both? And how does the final form of our encoding of meaning reflect the property of compositionality outlined above? One way to start answering this question is to consider some existing formalisms of meaning, how they relate to each other, and on which axes they differ.

Finally, assuming that we manage to decide on a usable definition of the concept of meaning and are able to formulate a family of models which have the capacity to learn to extract it from text, we still have to work out exactly how to learn such a model. Is the

usual machine learning paradigm of learning to solve a specific task from training data appropriate? Is there a task which fully captures meaning, or do we need to combine a set of tasks? What makes a good objective? Do we need to go beyond this training/test setting?

We do not claim to make significant progress on each and every one of these questions in this thesis. However, we believe that advances on any of them should always be guided by a good understanding of this global context. On the one hand, some common principles (e.g. compositionality) can be found at every level from character to document, so that answers to one of the questions may prove helpful in solving others. On the other hand, the solutions to all of the above mentioned problems depend on each other; it would make no sense to come up with a "good" representation of meaning by linguistics standards which makes model training next to impossible.

This thesis focuses on two of the problems presented here. On the one hand, we investigate the question of the choice of a reading level and consider different ways of combining characters into larger units of meaning (morphemes, words,...). On the other hand, we consider how to make language modeling, a commonly used language learning objective, more efficient, and work to find other good language learning tasks. Because we strongly believe in the necessity of synergy to make global progress on the wider question of language understanding, we also endeavor to put these advances in context and explore how they relate to problems which fall beyond our scope.

1.3 Representations of Meaning in NLP

As outlined in Section 1.2, there are a great many applications to text understanding which rely on being able to compute a representation of the meaning expressed in a text.

What then does learning such a representation look like for each of these applications? Is it possible to use a common definition and representation of meaning for all of them? Indeed, learning a different system for each would appear to be both impractical and wasteful, as they rely on many of the same mechanisms (*e.g.* compositionality…).

We can look to linguistics for general formalisms of meaning which might be useful in a natural language processing setting. The family of Model Theoretic Semantics, for example, proposes one way of describing and reasoning over the meaning of a sentence or text. Approaches which fall under the umbrella of Model Theoretic Semantics, such as those presented in [Tarski and Vaught, 1956] or [Montague, 1973], start with a model of the world, which usually consists of a base set of entities and relations or n-ary properties. Some form of predicate logic (e.g. first or higher-order logic, lambda calculus) then makes it possible to formulate information about how the entities and properties relate to each other. In this framework, understanding the meaning of a text can then be seen as mapping an utterance or sentence to a logical formula encoding this information, and indeed the problem has been the focus of a number of works in the field of NLP within the area of semantic parsing [Zettlemoyer and Collins, 2005, Wong and Mooney, 2006, Liang et al., 2011, Liang, 2016]. For example, consider a model of the world which has the following set of properties:

$$\{\textsc{IsCat}(\cdot), \textsc{IsRed}(\cdot), \textsc{IsMat}(\cdot), \textsc{IsAbove}(\cdot,\cdot)\}$$

and suppose we choose first order logic as our predicate logic. In this setting, understanding the sentence *"A red cat sat on a mat."*, would mean producing the following formula:

$$S(x,y) = \exists x, \exists y \quad \textsc{IsCat}(x) \wedge \textsc{IsRed}(x) \wedge \textsc{IsMat}(y) \wedge \textsc{IsAbove}(x,y)$$

It is easy to see how this formalism can be used to answer questions about a document by computing which answers are implied by its statements. For example, the question *"Is there something red on a mat?"* would be mapped to:

$$Q(x, y) = \exists x, \exists y \quad \textsc{IsRed}(x) \wedge \textsc{IsMat}(y) \wedge \textsc{IsAbove}(x, y)$$

Since we can compute that $S(x, y) \Rightarrow Q(x, y)$, we know that the answer to question Q in context S is *yes*. This is an extremely simplified example of a use case of Model Theoretic Semantics. A lot of attention has been given in the field of Natural Language Processing to learning such mappings to more complex models (such as Freebase [Bollacker et al., 2007]) and choices of predicate logics, including lambda calculus [Wong and Mooney, 2007] and frame semantic parsing [Das et al., 2014], and to making practical use of those for tasks such as Question Answering [Zelle and Mooney, 1996, Berant et al., 2013] or Knowledge Base Completion [Guu et al., 2015] by mapping questions and statements respectively to logical formulas. On the theoretical side, this family of formalisms has also proven to be largely useful in a number of areas of semantics and pragmatics [Lewis, 1970, Montague, 1973, Hobbs, 1979], including work on defining and analyzing the recently popularized tasks of Natural Language Inference and Recognizing Textual Entailment [Sánchez Valencia, 1991, MacCartney, 2009].

One weakness of this approach is quickly made obvious however: the scope of the formalism is heavily dependent on both the model and choice of predicate logic. In our simple example, *"A cat sat on a mat."* is indistinguishable from e.g. *"The cat sat on the mat."*, *"A cat sits on a mat."* or even *"Some cat flies over a mat."*. The definite article can be handled by adding grounding constants to the model, the tense can be handled by having different properties for each time, or by allowing properties to act

over other properties, and the difference between sitting on and flying over by having more granularity in our set of properties. Still, this simple model shows that devising a model/logic pair which covers a reasonable amount of sentences in an open domain while remaining tractable can be quite challenging. Given these limitations, computing a model-theoretic representation of meaning as a first step can then be impractical for a number of application domains of Natural Language Processing, and there have been a number of successful approaches to these problems which do not rely on an explicit model of the world.

In order to showcase how methods to handle (or dispense with) representations of meaning have shaped the development of Natural Language Processing, we consider the case of machine translation, as it has at times relied heavily both on symbolic knowledge-intensive and on more unsupervised statistical methods. Other tasks have leaned more strongly towards the one or the other, but have mostly followed similar paths.

Machine translation consists in taking a sentence or document in a source language as input, and outputting a sentence or document in a target language with the same meaning. The description of the task would seem to imply that obtaining a good representation of the source language document's meaning should be necessary, and indeed some of the early successful applications of automated machine translation relied on some form of semantic parsing. Strategies have included using a rule based system to go from the source language to an abstract "script", then using another set of rules to go from the script to the target language [Schank and Abelson, 1977, Cullingford, 1978, Carbonell et al., 1981]. One significant advantage of this setting is that the size of the models grows linearly in the number of languages rather than the number of language pairs, since it only requires an encoder from natural language to script and

a decoder from script to natural language for each. However, the approach "requires general semantic information and domain specific knowledge roughly proportional to the semantic knowledge base that a human translator would bring to bear." [Carbonell and Tomita, 1985], which can be costly to obtain, and limits its application to restricted domains, such as brief topic specific news text or technical documentation.

The popularization of statistical machine translation following the development of the IBM models in the early 1990s [Brown et al., 1990, Brown et al., 1993] was required to allow the field to move beyond the narrow application domains of those early knowledge-intensive settings. These methods rely on parallel corpora of documents available in several languages, such as the European Parliament proceedings [Koehn, 2005], and use a statistical model to learn a maximum likelihood word-to-word or word-to-phrase alignment between two versions of the same documents. When using the system, this alignment model is then combined with a language model in the target language to obtain a translation. Compared to the previous knowledge-based approaches, the statistical methods present the significant advantage of being mostly unsupervised: all that is required to learn a translation system between two languages is a sufficiently large aligned bilingual corpus. However, without any common representation of meaning, the generalization ability of the models across languages suffers significantly. At a high level, this approach reduces the notion of meaning and of translation to something which is defined for the granularity of a word or phrase: it identifies cross-lingual pairs of "equivalent" words, along with a re-ordering function from a sentence in the source to one the target language. This implies that while some of the parameters (such as those of the language model) can be learned once for each language, most need to be trained for each language pair. This dependence is reduced somewhat in systems which better handle phrase level translation [Och and Ney, 2004] or take advantage of syntax [Chiang,

2005, Galley et al., 2006, Venugopal et al., 2007, Zhang and Gildea, 2008, Katz-Brown et al., 2011], but remains a limiting factor for low resource language pairs.

The next significant advance in the field of automatic machine translation came in the form of neural methods [Kalchbrenner and Blunsom, 2013, Sutskever et al., 2014, Cho et al., 2014a]. These approaches use a neural network to compute a distributed representation of the sentence in the source language, and another neural network to decode this representation into a target language sentence. In practice, having a single vector as representation of the input sentence happens to be too much of a bottleneck, and the success of these techniques often depend on some form of attention mechanism [Bahdanau et al., 2014], but whether the intermediary state consists in the final state of a recurrent network or a set of recurrent or convolutional states, the paradigm can broadly be defined as mapping a sentence to an implicit representation of its meaning, and decoding this representation into a sentence in another language. While the representation isn't as interpretable as the output of a semantic parser, this allows the systems to learn more efficiently by taking advantage of the common mechanisms across language pairs sharing a language [Johnson et al., 2017] and even by learning from non-aligned monolingual corpora [Lample et al., 2017].

The idea of using distributed representations for concepts and meaning did not start with machine translation, and has had an impact in most other fields of Natural Language Processing. Early examples can be found in the connectionist literature of the 1980s [Hinton, 1986], with some successful applications to word level language modeling starting in the works of [Schwenk and Gauvain, 2002] or [Bengio et al., 2003]. In those settings, a neural network is used to obtain a distributed representation of the immediate context, or of the whole text read so far in the case of recurrent neural network language models [Mikolov et al., 2010]. Similar approaches to representing the input text have

also recently lead to significant progress in fields such as text summarization, allowing to go beyond the extractive [Kupiec et al., 1995] and to start tackling the abstractive setting [Rush et al., 2015, Nallapati et al., 2016], as well as in the area of question answering [Iyyer et al., 2014, Sukhbaatar et al., 2015, Andreas et al., 2016].

All of the above mentioned recent works correspond to a common approach: the neural network model learns an implicit representation of a text's meaning (or at least of all of its relevant properties for the purpose of solving the task at hand, which is our working definition of "meaning") in the form of an embedding or a set of embeddings in a continuous space. This representation is not as interpretable as a semantic parse, and it is not quite obvious how to use it to reason about the properties of language and draw insights about linguistic phenomena. However, its flexibility and self-organization properties makes it extremely practical for semi- or un-supervised training of various language processing systems. Still, one question remains: do we need to learn separate representation functions for each of those tasks, or can we take advantage of the synergies between them in a more efficient way?

One possible way of doing so would be to come up with a task, or set of tasks, which require a system to learn all of the properties of text and language we shall ever need. That is, a set of tasks which are strictly more difficult taken together than any application we want to set for language understanding. We argue that language modeling provides a first shot at this, which has been helpful for developing systems which learn generally useful representations of words [Mikolov et al., 2013] and sentences [Kiros et al., 2015]. Indeed, understanding the difference between sequences of words which make sense (sentences which are likely to occur in data) and sequences which do not (and should therefore not occur, or very little), requires an understanding of grammar, and the realities which the language can refer to. For example, consider the following

text sample:

> I wanted to keep the files secure. There were a simple wooden box and safe on the table, but I did not know how to open the... safe.

When doing word level language modeling, we need to compute the probability of each word given the ones to its left. Let's focus on the final *safe*. One can see for example how an understanding of grammar of syntax tells us that *to the...* will likely be followed by a noun. A system also needs to have some form of medium term memory to see that the most likely candidates are the recently mentioned nouns, here *files*, *table*, *box*, and *safe*. Finally, general knowledge about the world tells us that of these, a *safe* can have a combination and is use to keep items safe. While this does not guarantee that good language modeling performance will always require a thorough understanding of the meaning of a text, it does seem to indicate that it might be a good first step to reach this goal, and so Chapters 2, 3, and 4 focus on this task (we revisit the choice of learning objective in Chapter 5).

In the rest of this Chapter, we provide some general algorithmic and conceptual background for the rest of this work. We start by introducing the problem of language modeling, then present the class of Recurrent Neural Networks, which is of broad use in language processing applications in general and in the work presented in this thesis in particular. The next Chapters then show how to devise models which can produce distributed representations of meaning as described in the previous paragraphs, and how to use freely available text data to learn these models' parameters through language and discourse modeling.

Note on Conventions and Notations. We use the following notations in the rest of this thesis. Constants are denoted as upper case letters (e.g. dimension D). Vectors

are denoted as bold lower case letters (e.g. $\mathbf{v} \in \mathbb{R}^D$) and matrices as bold upper case letters (e.g. $\mathbf{M} \in \mathbb{R}^{D \times D}$). Sets are denoted as calligraphic upper case letters (e.g. $\mathcal{W}$). A sequence of words is denoted either as a sequence of word indices in a vocabulary $\mathcal{V}$ (e.g. $(w)_{1,N} = (w_1, w_2, \ldots, w_N)$ where $\forall i, w_i \in \{1, \ldots, |\mathcal{V}|\}$) or of word embeddings (e.g. $(\mathbf{w})_{1,N} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N)$ where $\forall i, \mathbf{w}_i \in \mathbb{R}^D$).

1.4 Background: Neural Language Modeling

The task of language modeling consists in learning a probability distribution over text. At a high level, this corresponds to being able to discriminate between sequences of words which correspond to natural language and sequences which do not: whether they be a-grammatical, incoherent, or fail to correspond to an interpretable and plausible meaning in some other way. Being able to formulate such a distribution has practical applications in any Natural Language Processing (NLP) task whose output format is a sequence of words, since it can be used to limit the search space, or re-rank proposals. Those include speech recognition, text generation, machine translation, or dialogue systems, among others.

Conditioning on the Past Let us now formally define the class of contextual language models. As noted previously, the goal is to formulate a probability distribution over word sequences (document, sentence, paragraph,...) $(w)_{1,N} = (w_1, \ldots, w_N)$ (the distribution can be defined over sequences of any length through the use of a special <END> token). Contextual language models simply decompose this distribution by

using the chain rule from left to right:

$$p((w)_{1,N}) = \prod_{i=1}^{N} p(w_i|(w)_{1,i-1})$$

The above formulation holds without loss of generality. However, early approaches to language modeling have found it useful to make an additional n-th order Markov assumption. That is, the probability of a word given its left context only depends on its n immediate left neighbors:

$$p(w_i|(w)_{1,i-1}) \approx p(w_i|w_{i-n}, \ldots, w_{i-1})$$

The model is then entirely defined by the probability of a word given its immediate context $p(w_i|w_{i-n}, \ldots, w_{i-1})$. Works such as [Chen and Goodman, 1998] estimate those via counting and subsequent smoothing . The probability of the next word given the past is then represented as a multinomial distribution conditioned on the local left context.

These count-based models are simple to train, but probabilities of rare n-grams can be poorly estimated due to data sparsity (despite smoothing techniques). Neural n-gram models have been developed to address this data sparsity issue by learning low dimensional embeddings of the words and local context and using them as inputs to a neural network [Bengio et al., 2003]. More formally, to each word $w \in \mathcal{V}$ corresponds a vector $\mathbf{w} \in \mathbb{R}^D$, and a transition matrix $\mathbf{R} \in \mathbb{R}^{D \times D}$ is used to obtain a representation of the context. This model estimates:

$$p(w_i|w_{i-n}, \ldots, w_{i-1}) = \frac{\exp((\mathbf{w}_{i-n}^{n} + \ldots + \mathbf{w}_{i-1}^{1})^T \mathbf{R} \mathbf{w}_i)}{Z}$$

Where Z is the local partition function. Word embeddings obtained through these mod-

els exhibit interesting linguistic properties, where the difference between the vectors representing two words can give some information on how they relate to each other semantically and morphologically [Mikolov et al., 2013] (similar to ones obtained with document-level non-neural techniques such as Latent Semantic Analysis [Deerwester et al., 1990]).

More recent Neural Network Language Models (NNLMs) have expanded upon this general idea by using a rich family of neural networks to represent the local context, such as log-bilinear [Mnih and Hinton, 2007], sum-product [Cheng et al., 2014], and convolutional [Wang et al., 2015, Dauphin et al., 2017] networks, and even allowing the model access to context beyond the local neighborhood through the use of recurrent architectures [Mikolov et al., 2010]. NNLMs have been shown to produce competitive results with n-gram models using many fewer parameters. Additionally the parameters themselves have proven to be useful for other language tasks [Collobert et al., 2011]. Unfortunately, while the parameterization of NNLMs is significantly more compact than that of n-gram models, training can be much slower, often requiring expensive gradient computations for each token, and developing techniques to speed up training in practice has been an active area of research [Mnih and Hinton, 2008, Mnih and Teh, 2012].

1.5 Background: Recurrent Neural Networks

As mentioned in Section 1.4, neural n-gram models approximate the chain rule by making an additional Markov assumption: this is necessary as keeping the full left context in memory for documents of any reasonable size is intractable. An alternative, however, would be to build a summary of the left context as we read the document. One way to do this is to use the class of Recurrent Neural Networks (RNNs), which we

describe in this Section.

Recurrent Neural Network Definition Let us start by formally defining the class of RNNs. For tasks such as language modeling, we are interested in defining a probability distribution over sequences $(w)_{1,T} = (w_1, \ldots, w_T)$. Using the chain rule, the negative log likelihood of a sequence can be written:

$$\mathcal{L}((w)_{1,T}) = -\sum_{t=1}^{T} \log \left(p\big(w_t | \mathcal{F}\left(w_1, \ldots, w_{t-1}\right)\big)\right). \tag{1.1}$$

where $\mathcal{F}$ is a filtration, *i.e.* a function which summarizes all the relevant information from the past. RNNs are a class of models that can read a sequence of arbitrary length to provide such a summary in the form of a hidden state $h_t \approx \mathcal{F}(w_1, \ldots, w_{t-1})$, by applying the same operation (recurrent unit) at each time step. More specifically, the recurrent unit is defined by a recurrence function g which takes as input the previous hidden state $\mathbf{h}_{t-1}$ at each time step t, as well as a representation of the input $\mathbf{x}_t$ (where $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ are D-dimensional vectors), and (with the convention $\mathbf{h}_0 = 0$,) outputs the new hidden state:

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{1.2}$$

It is easy to extend a RNN to two (or more) layers by having another network whose input at t is $\mathbf{h}_t$ (from the first network). Indeed, having multiple layers is often crucial for obtaining competitive performance on various tasks [Pascanu et al., 2013a].

Elman Unit The unit described in [Elman, 1990] is often considered to be the standard unit. It is parameterized by $\mathbf{U}$ and $\mathbf{V}$, which are square, D-dimensional transition

matrices, and uses a sigmoid non-linearity to obtain the new hidden state:

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t) \tag{1.3}$$

In theory the RNN can summarize all historical information up to time t with the hidden state $\mathbf{h}_t$. In practice however, capturing long-range dependencies with a vanilla RNN is difficult due to vanishing/exploding gradients [Bengio et al., 1994], which occurs as a result of the Jacobian's multiplicativity with respect to time.

LSTM Unit Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] addresses the problem of learning long range dependencies by augmenting the RNN with a memory cell vector $\mathbf{c}_t \in \mathbb{R}^n$ at each time step. Concretely, one step of an LSTM takes as input $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}$ and produces $\mathbf{h}_t$, $\mathbf{c}_t$ via the following intermediate calculations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1} + \mathbf{b}^i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1} + \mathbf{b}^f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1} + \mathbf{b}^o) \\
\mathbf{g}_t &= \tanh(\mathbf{W}^g\mathbf{x}_t + \mathbf{U}^g\mathbf{h}_{t-1} + \mathbf{b}^g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{1.4}$$

Where $\sigma(\cdot)$ and $\tanh(\cdot)$ are the element-wise sigmoid and hyperbolic tangent functions, and $\odot$ is the element-wise multiplication operator. $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are referred to as *input*, *forget*, and *output* gates respectively. At $t = 1$, $\mathbf{h}_0$ and $\mathbf{c}_0$ are initialized to zero vectors. Parameters of the LSTM are $\mathbf{W}^j, \mathbf{U}^j, \mathbf{b}^j$ for $j \in \{i, f, o, g\}$. Memory cells in the LSTM

are additive with respect to time, alleviating the gradient vanishing problem. Gradient explosion is still an issue, though in practice simple optimization strategies (such as gradient clipping [Pascanu et al., 2013b]) work well. LSTMs have been shown to outperform vanilla RNNs on many tasks, including on language modeling [Sundermeyer et al., 2012].

Gated Recurrent Unit The Gated Recurrent Unit (GRU) was introduced in [Cho et al., 2014b]. The main difference between the GRU and Elman unit consists in the model's ability to interpolate between a proposed new hidden state and the current one, which makes it easier to model longer range dependencies. More specifically, at each time step t, the model computes a reset gate $\mathbf{r}_t$, an update gate $\mathbf{z}_t$, a proposed new hidden state $\tilde{\mathbf{h}}_t$ and a final new hidden state $\mathbf{h}_t$ as follows:

$$\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{V}^r \mathbf{x}_t + \mathbf{U}^r \mathbf{h}_{t-1} + \mathbf{b}^r) \\
\mathbf{z}_t &= \sigma(\mathbf{V}^z \mathbf{x}_t + \mathbf{U}^z \mathbf{h}_{t-1} + \mathbf{b}^z) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{V}^h \mathbf{x}_t + \mathbf{U}^h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}^h) \\
\mathbf{h}_t &= \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1}
\end{aligned} \tag{1.5}$$

RNN language models Let $\mathcal{V}$ be the fixed size vocabulary of words. A conditional language model as defined in Section 1.4 specifies a distribution over w_{t+1} (whose support is $\mathcal{V}$) given the historical sequence $w_{1,t} = (w_1, \ldots, w_t)$. A recurrent neural network language model (RNN-LM) does this by applying an affine transformation to the hidden layer of a recurrent neural network on the word embeddings followed by a softmax, as

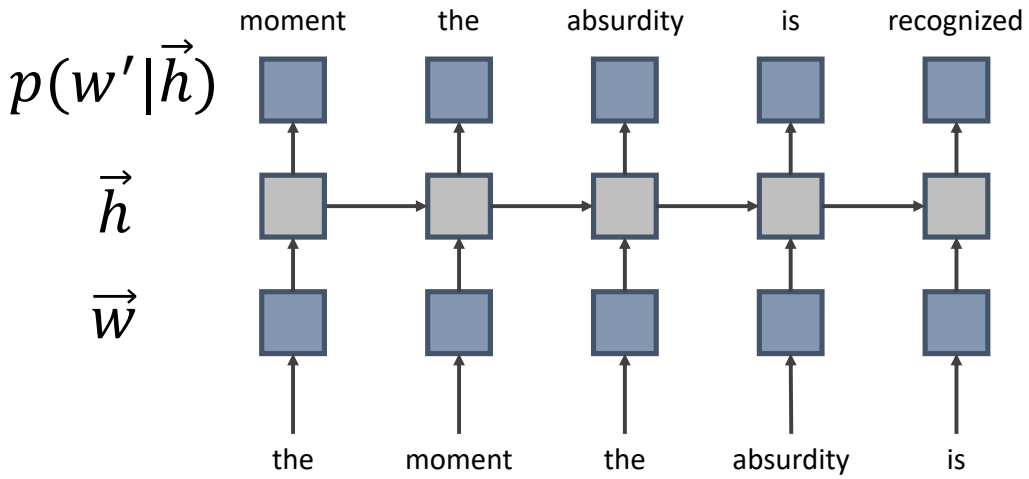

Figure 1.1: Recurrent neural network language model. At each time step, the hidden state $\mathbf{h}_t$ is updated based on its the previous time step version $\mathbf{h}_{t-1}$ and the embedding of the current word w_t. The model then outputs a distribution over the next word $p(w_{t+1}|\mathbf{h}_t)$

illustrated in Section 1.1

$$p(w_{t+1} = j|w_{1,t}) = \frac{\exp(\mathbf{h}_t \cdot \mathbf{p}^j + q^j)}{\sum_{j' \in \mathcal{V}} \exp(\mathbf{h}_t \cdot \mathbf{p}^{j'} + q^{j'})} \tag{1.6}$$

where $\mathbf{p}^j$ is the j-th column of $\mathbf{P} \in \mathbb{R}^{D \times |\mathcal{V}|}$ (also referred to as the *output embedding*), and q^j is a bias term. Similarly, for a conventional RNN-LM which usually takes words as inputs, if $w_t = k$, then the input to the RNN-LM at t is the *input embedding* $\mathbf{w}^k$, the k-th column of the embedding matrix $\mathbf{W} \in \mathbb{R}^{D_e \times |\mathcal{V}|}$. If we denote $w_{1:T} = (w_1, \cdots, w_T)$ to be the sequence of words in the training corpus, training involves minimizing the negative log-likelihood of the sequence (Equation 1.1) which is typically done by truncated backpropagation through time [Werbos, 1990, Graves, 2013].

1.6 Organization of this Thesis

In this Section, we provide a brief overview of the rest of this thesis. Chapter 2 focuses on the problem of choosing a reading level for a text understanding system. A majority of models use words as a smallest semantic unit. We consider the motivation for this choice and present arguments for also making use of sub-word information. We describe two approaches to making use of character-level input. The first one computes a representation of a word's meaning in the form of an embedding vector by composing all of its characters and sub-sequences of characters, making use of a word level language modeling objective to learn the parameters of the system. We show that in addition to learning some notion of morphology and allowing our system to handle new words it has not seen before, these embeddings lead to an improvement in language modeling performance. Our second approach also looks at combining characters, but in a bottom up rather than top down fashion, by giving a recurrent neural network the ability to vary the amount of computation it performs after reading each character. We show that this leads to a more computation efficient model than comparable character-level approaches, and that the system also learns a notion of morphology and word meanings.

The second part of the thesis is concerned with the learning objective question. As outlined above, there are a great many applications to text understanding. Learning a different system for each of these would be both impractical and wasteful, as they have many mechanisms in common. Machine learning research often takes a task-driven approach to learning: recognizing objects in a picture, predicting the outcome of a situation... If we want to apply this paradigm to language learning, we need to come up with a task, or set of tasks, which requires a system to learn all of the properties of text and language we shall ever need. That is, a set of tasks which are strictly more difficult

taken together than any application we want to set for language understanding.

Language modeling provides a first shot at this, which has been helpful for developing systems which learn representations of words [Mikolov et al., 2013] and sentences [Kiros et al., 2015]. The idea is that understanding the difference between sequences of words which make sense (sentences which are likely to occur in data) and sequences which do not (and should therefore not occur, or very little), requires an understanding of grammar, and the realities which the language can refer to. Chapters 3 and 4 are both concerned with improving existing language modeling objectives in different ways. Most existing language models compute the probability of each word given their context. This can be quite expensive, as it requires to compute a score at each time step for every word in a vocabulary whose size can be in the hundreds of thousands or millions. Chapter 3 aims to speed up this step by proposing an adaptive hierarchical objective. Hierarchical prediction significantly reduces the cost of computing the probability of a word, and we provide an algorithm which allows to learn a good tree structure for this purpose along with the other parameters of the model. Chapter 4, on the other hand, introduces a new class of language models which is globally rather than locally normalized; that is to say, we provide a score for the text as a whole, and do approximate normalization over all possible sequences of words. This can have several advantages, from avoiding the label bias problem to speeding up the objective computation for large text corpora.

Finally, other objectives can also help a system learn the relevant properties of language, with possibly better local minima (language modeling can rely on grammar and local word-level semantics significantly). Understanding the structure of discourse, for example, requires a good grasp on both semantics and pragmatics, which constitute a higher level of text understanding. Inspired by this, Chapter 5 proposes a set of new

discriminative objectives based on discourse structure. We follow this by sketching a possible formalism for descriptions of meaning based on continuous representations and function approximations.

We conclude by giving a summary of the contributions of this thesis, and reviewing some of the remaining open questions presented in Section 1.2, as well as those raised in the following Chapters.

Chapter 2

Using Sub Word Information

So far, we have noted how a number of fundamental language understanding mechanisms are connected to the Language Modeling task, and proposed to use the family of neural networks, and more specifically recurrent neural networks, as a category of architectures to learn to perform language modeling. At this point, there are still several design choices to be made to meet this objective, from the structure of the RNN to the dimensions of the intermediary representations. But one of the most important choices, which corresponds to one of our original questions, is that of the reading level. We explore this question in the following Section. More specifically, we argue for the use of sub-word information. First, we show that a word level recurrent network language model can benefit from using word embeddings which depend on a word's characters. Conversely, we also show that a character level model can learn to organize its computational load to make use of morphological and word-level structure.

2.1 Introduction

Text can be considered at different levels of granularity: as a sequence of characters, morphemes, words... We could even try to go so far as to consider phrases or sentences as units of meaning. So far, we have focused on using words: evaluating their likelihood and learning embeddings encoding their relevant properties. There are some good reasons to consider words. Contrary to characters, one can assign a notion of meaning to a single word: verbs often refer to actions, nouns to objects, etc... Segmentation of text into words is relatively straightforward in English. Words are also relatively tractable, as most datasets have vocabularies of at most a few hundred thousands or million words. On the other hand, there are far too many phrases, let alone sentences to handle in any language if one were to treat them as independent units.

However, words are compositional (for example, a priori, *eventful*, *eventfully*, *uneventful*, and *uneventfully* should have structurally related embeddings in the vector space), have a history, and the distribution of their occurrences follows Zipf's law; embeddings of rare words can thus be poorly estimated, leading to high perplexities for rare words and words surrounding them. While we could probably learn everything we need to know while considering different words as independent entities, looking at their characters allows to take compositionality and formation history into account, and to alleviate the difficulty of learning relevant information about rarely occurring data. In this Chapter, we consider two approaches to taking advantage of morphological information. On the one hand, one can keep using words as units, but encourage their representations to depend on their composition by building character-aware word embeddings. On the other hand, one can build sentence representations directly from characters, and let the model figure out when higher level reasoning (word-level, phrase-level) is necessary

through adaptive computation.

Character-Aware Word Embeddings Word embeddings capture a continuous representation of a word's meaning. More specifically, when these embeddings are learned in the context of language modeling, it is expected that two words which appear in similar contexts should map to vectors whose Euclidean distance is small. These kinds of abstractions of a word's meaning have been shown to be useful, as Neural Language Models outperform count-based n-gram language models [Mikolov et al., 2011]. However, they are also blind to the kind of subword information which humans have access to (and make use of) when parsing text, such as prefixes, suffixes, declinations or conjugations of words, etc... This can make it more difficult for them to learn a representation for rare words, especially in morphologically rich languages with long-tailed frequency distributions or domains with dynamic vocabularies (e.g. social media).

Previous work has looked into taking advantage of morphological information to mitigate this rare world problem. For example, [Botha and Blunsom, 2014, Luong et al., 2013] use morphological tagging and segmentation as a pre-processing step, which allows them to learn and combine morpheme vectors to obtain a word embedding. [Alexandrescu and Kirchhoff, 2006] (building on analogous work on count-based n-gram language models by Bilmes and Kirchhoff [Bilmes and Kirchhoff, 2003]) represent a word as a set of shared factor embeddings. Their Factored Neural Language Model (FNLM) can incorporate morphemes, word shape information (e.g. capitalization) or any other annotation (e.g. part-of-speech tags) to represent words. A specific class of FNLMs leverages morphemic information by viewing a word as a function of its (learned) morpheme embeddings [Luong et al., 2013, Botha and Blunsom, 2014, Qiu et al., 2014]. For example [Luong et al., 2013] apply a recursive neural network over

morpheme embeddings to obtain the embedding for a single word. While such models have proved useful, they require morphological tagging as a preprocessing step. Another direction of work has involved purely character-level NNLMs, wherein both input and output are characters [Sutskever et al., 2011, Graves, 2013]. Outside of language modeling, improvements have been reported on part-of-speech tagging [dos Santos and Zadrozny, 2014] and named entity recognition [dos Santos and Guimarães, 2015] by representing a word as a concatenation of its word embedding and an output from a character-level CNN, and using the combined representation as features in a Conditional Random Field (CRF). Finally, [Ling et al., 2015] apply a bi-directional LSTM over characters to use as inputs for language modeling and part-of-speech tagging. They show improvements on various languages (English, Portuguese, Catalan, German, Turkish).

Adaptive Character Level Sequence Modeling Another option does away with word level embeddings entirely, and modeling text as a sequence of characters. However, doing that well relies on being able to at least implicitly be able to handle processes happening at different time scales, and having more computational power at word boundaries can certainly help. For example, after reading the left context *The prime...*, the model should be able to put a higher likelihood on the sequence of characters that make up the word *minister*. However, we can take this idea one step further: after reading *The prime min...*, the next few characters are almost deterministic, and the model should require little computation to predict the sequence *i-s-t-e-r*.

Handling concurrent processes with various schedules is a problem of wide interest in sequence modeling: consider sequential data such as video feeds, audio signal, or language. In video data, there are time periods where the frames differ very slightly, and where the underlying model should probably do much less computation than when

the scene completely changes. When modeling speech from an audio signal, it is also reasonable to expect that the model should be able do little to no computation during silences. Thus, we want recurrent neural networks to be able to efficiently model processes happening at different and possibly varying time scales, without prior knowledge of the sequence's time structure. While many sequential data can have highly variable information flow, most recurrent models still consume input features at a constant rate and perform a constant number of computations per time step, which can be detrimental to both speed and model capacity.

How to properly handle sequences which reflect processes happening at different time scales has been a widely explored question. Among the proposed approaches, a variety of notable systems based on Hidden Markov Models (HMMs) have been put forward in the last two decades. The Factorial HMM model of [Ghahramani and Jordan, 1997] (and its infinite extension in [Gael et al., 2008]) use parallel interacting hidden states to model concurrent processes. While there is no explicit handling of different time scales, the model achieves good held-out likelihood on Bach chorales, which exhibit multi-scale behaviors. The hierarchical HMM model of [Fine et al., 1998] and [Murphy and Paskin, 2001] takes a more direct approach to representing multiple scales of processes. In these works, the higher level HMM can recursively call sub-HMMs to generate short sequences without changing its state, and the authors show a successful application to modeling cursive writing. Finally, the Switching State-Space Model of [Ghahramani and Hinton, 2000] combines HMMs and Linear Dynamical Systems: in this model, the HMM is used to switch between LDS parameters, and the experiments show that the HMM learns higher-level, slower dynamics than the LDS.

On the side of recurrent neural networks, the idea that the models should have mechanisms that allow them to handle processes happening at different time scales is not a

new one either. On the one hand, such early works as [Schmidhuber, 1992] already presented a two level architecture, with an "automatizer" acting on every time step and a "chunker" which should only be called when the automatizer fails to predict the next item, and which the author hypothesizes learns to model slower scale processes. On the other hand, the model proposed in [Mozer, 1991] has slow-moving units as well as regular ones, where the slowness is defined by a parameter $\tau \in [0, 1]$ deciding how fast the representation changes by taking a convex combination of the previous and predicted hidden state. Both these notions, along with different approaches to multi-scale sequence modeling, have been developed in more recent work. [Mikolov et al., 2014] expand upon the idea of having slow moving units in an RNN by proposing an extension of the Elman unit which forces parts of the transition matrix to be close to the identity. The idea of having recurrent layers called at different time steps has also recently regained popularity. The Clockwork RNN of [Koutník et al., 2014], for example, has RNN layers called every 1, 2, 4, 8, etc. . . time steps. The conditional RNN of [Bojanowski et al., 2015] takes another approach by using known temporal structure in the data: in the character level level language modeling application, the first layer is called for every character, while the second is only called once per word. It should also be noted that state-of-the art results for language models have been obtained using multi-layer RNNs [Józefowicz et al., 2016], where the higher layers can in theory model slower processes. However, introspection in these models is more challenging, and it is difficult to determine whether they are actually exhibiting significant temporal behaviors.

Finally, even more recent efforts have considered using dynamic time schedules. [Chung et al., 2016] presents a multi-layer LSTM, where each layer decides whether or not to activate the next one at every time step. They show that the model is able to

learn sensible time behaviors and achieve good perplexity on their chosen tasks. Another implementation of the general concept of adaptive time-dependent computation is presented in [Graves, 2016]. In that work, the amount of computation performed at each time step is varied not by calling units in several layers, but rather by having a unique RNN perform more than one update of the hidden state on a single time step. There too, the model can be shown to learn an intuitive time schedule.

2.2 Character-Aware Word Embeddings: Model

In this Section, we propose a language model that leverages subword information through a character-level convolutional neural network (CNN), whose output is used as an input to a recurrent neural network language model (RNN-LM). Unlike previous works that utilize subword information via morphemes [Botha and Blunsom, 2014, Luong et al., 2013], our model does not require morphological tagging as a pre-processing step. And, unlike the recent line of work which combines input word embeddings with features from a character-level model [dos Santos and Zadrozny, 2014, dos Santos and Guimarães, 2015], our model does not utilize word embeddings at all in the input layer. Given that most of the parameters in Neural Network Language Models (NNLMs) are from the word embeddings, the proposed model has significantly fewer parameters than previous NNLMs, making it attractive for applications where model size may be an issue (e.g. cell phones). The architecture of our model, shown in Figure 2.1, is straightforward. Whereas a conventional NNLM takes word embeddings as inputs, our model instead takes the output from a single-layer character-level convolutional neural network with max-over-time pooling.

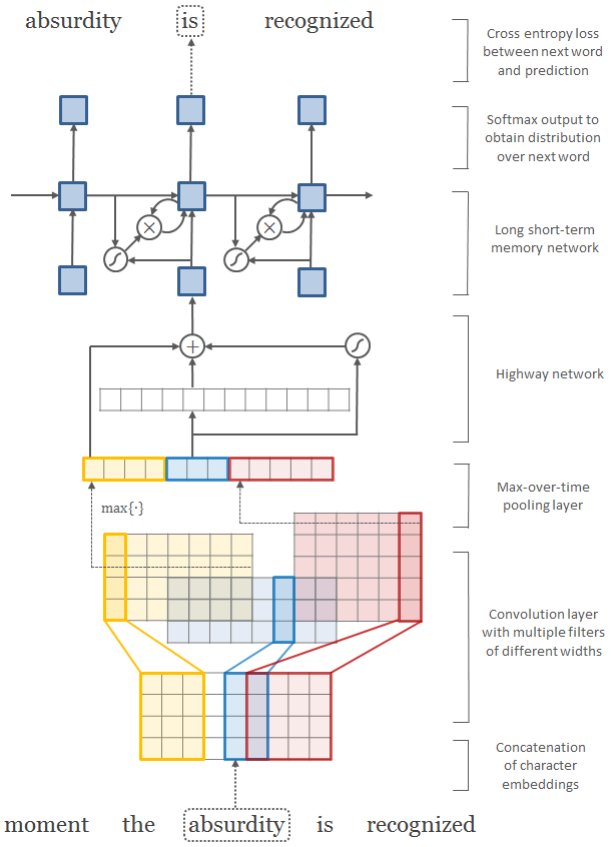

Figure 2.1: Character-aware word embedding architecture. Here the model takes *absurdity* as the current input and combines it with the history (as represented by the hidden state) to predict the next word, *is*. First layer performs a lookup of character embeddings (of dimension four) and stacks them to form the matrix $\mathbf{C}^k$. Then convolution operations are applied between $\mathbf{C}^k$ and multiple filter matrices. Note that in the above example we have twelve filters—three filters of width two (blue), four filters of width three (yellow), and five filters of width four (red). A max-over-time pooling operation is applied to obtain a fixed-dimensional representation of the word, which is given to the highway network. The highway network's output is used as the input to a multi-layer LSTM. Finally, an affine transformation followed by a softmax is applied over the hidden representation of the LSTM to obtain the distribution over the next word. Cross entropy loss between the (predicted) distribution over next word and the actual next word is minimized. Element-wise addition, multiplication, and sigmoid operators are depicted in circles, and affine transformations (plus nonlinearities where appropriate) are represented by solid arrows.

Character-level CNN In our model, the input at time t is an output from a character-level convolutional neural network (CharCNN), which we describe in this section. CNNs [LeCun et al., 1989] have achieved state-of-the-art results on computer vision [Krizhevsky et al., 2012] and have also been shown to be effective for various NLP tasks [Collobert et al., 2011]. Architectures employed for NLP applications differ in that they typically involve temporal rather than spatial convolutions.

Let $\mathcal{C}$ be the vocabulary of characters, d be the dimensionality of character embeddings,[1] and $\mathbf{Q} \in \mathbb{R}^{d \times |\mathcal{C}|}$ be the matrix character embeddings. Suppose that word $k \in \mathcal{V}$ is made up of a sequence of characters $[c_1, \ldots, c_l]$, where l is the length of word k. Then the character-level representation of k is given by the matrix $\mathbf{C}^k \in \mathbb{R}^{d \times l}$, where the j-th column corresponds to the character embedding for c_j (i.e. the c_j-th column of $\mathbf{Q}$).[2]

We apply a narrow convolution between $\mathbf{C}^k$ and a *filter* (or *kernel*) $\mathbf{H} \in \mathbb{R}^{d \times w}$ of width w, after which we add a bias and apply a nonlinearity to obtain a *feature map* $\mathbf{f}^k \in \mathbb{R}^{l-w+1}$. Specifically, the i-th element of $\mathbf{f}^k$ is given by:

$$\mathbf{f}^k[i] = \tanh(\langle \mathbf{C}^k[*, i : i + w - 1], \mathbf{H} \rangle + b) \tag{2.1}$$

where $\mathbf{C}^k[*, i : i+w-1]$ is the i-to-$(i+w-1)$-th column of $\mathbf{C}^k$ and $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{Tr}(\mathbf{A}\mathbf{B}^T)$ is the Frobenius inner product. Finally, we take the *max-over-time*

$$y^k = \max_i \mathbf{f}^k[i] \tag{2.2}$$

[1] Given that $|\mathcal{C}|$ is usually small, some authors work with one-hot representations of characters. However we found that using lower dimensional representations of characters (i.e. $d < |\mathcal{C}|$) performed slightly better.

[2] Two technical details warrant mention here: (1) we append start-of-word and end-of-word characters to each word to better represent prefixes and suffixes and hence $\mathbf{C}^k$ actually has $l + 2$ columns; (2) for batch processing, we zero-pad $\mathbf{C}^k$ so that the number of columns is constant (equal to the max word length) for all words in $\mathcal{V}$.

as the feature corresponding to the filter $\mathbf{H}$ (when applied to word k). The idea is to capture the most important feature—the one with the highest value—for a given filter. A filter is essentially picking out a character n-gram, where the size of the n-gram corresponds to the filter width.

We have described the process by which *one* feature is obtained from *one* filter matrix. Our CharCNN uses multiple filters of varying widths to obtain the feature vector for k. So if we have a total of h filters $\mathbf{H}_1, \ldots, \mathbf{H}_h$, then $\mathbf{y}^k = [y_1^k, \ldots, y_h^k]$ is the input representation of k. For many NLP applications h is typically chosen to be in $[100, 1000]$. Note also that while having all filters be of maximum widths should in theory be sufficient to capture all relevant properties of a word, in practice having different sizes helps the model learn to pick up shorter morphemes in a more data- and computation-efficient way.

Highway Network We could simply replace the word embedding with $\mathbf{y}^k$ at each t in the RNN-LM, and as we show later, this simple model performs well on its own (Table 2.7). One could also have a multilayer perceptron (MLP) over $\mathbf{y}^k$ to model interactions between the character n-grams picked up by the filters, but we found that this resulted in worse performance. Instead we obtained improvements by running $\mathbf{y}^k$ through a *highway network*, recently proposed by Srivastava et al. [Srivastava et al., 2015]. Whereas one layer of an MLP applies an affine transformation followed by a nonlinearity to obtain a new set of features,

$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b}) \tag{2.3}$$

one layer of a highway network does the following:

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H \mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y} \tag{2.4}$$

where g is a nonlinearity, $\mathbf{t} = \sigma(\mathbf{W}_T \mathbf{y} + \mathbf{b}_T)$ is called the *transform* gate, and $(\mathbf{1} - \mathbf{t})$ is called the *carry* gate. Similar to the memory cells in LSTM networks, highway layers allow for training of deep networks by adaptively *carrying* some dimensions of the input directly to the output.[3] By construction the dimensions of $\mathbf{y}$ and $\mathbf{z}$ have to match, and hence $\mathbf{W}_T$ and $\mathbf{W}_H$ are square matrices.

2.3 Character-Aware Word Embeddings: Experiments

Experimental Setup We conduct experiments with an LSTM-based Recurrent Neural Network language model, as described in Section 1.5, whose inputs are the product of the character-aware embedding model described in Section 2.2. As is standard in language modeling, we use perplexity (PPL) to evaluate the performance of our models. Perplexity of a model over a sequence $(w_1, \ldots, w_T)$ is given by

$$PPL = \exp\Big(\frac{\mathcal{L}}{T}\Big) \tag{2.5}$$

where $\mathcal{NL}$ is calculated over the test set. We test the model on corpora of varying languages and sizes (statistics available in Table 2.1).

We conduct hyperparameter search, model introspection, and ablation studies on the English Penn Treebank (PTB) [Marcus et al., 1993], utilizing the standard training (0-20), validation (21-22), and test (23-24) splits along with pre-processing by [Mikolov

[3] Srivastava et al. [Srivastava et al., 2015] recommend initializing $\mathbf{b}_T$ to a negative value, in order to militate the initial behavior towards *carry*. We initialized $\mathbf{b}_T$ to a small interval around -2.

	DATA-S			DATA-L		
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{C}\rvert$	T	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{C}\rvert$	T
English (EN)	10 k	51	1 m	60 k	197	20 m
Czech (CS)	46 k	101	1 m	206 k	195	17 m
German (DE)	37 k	74	1 m	339 k	260	51 m
Spanish (ES)	27 k	72	1 m	152 k	222	56 m
French (FR)	25 k	76	1 m	137 k	225	57 m
Russian (RU)	62 k	62	1 m	497 k	111	25 m
Arabic (AR)	86 k	132	4 m	–	–	–

Table 2.1: Corpus statistics. $|\mathcal{V}|$ = word vocabulary size; $|\mathcal{C}|$ = character vocabulary size; T = number of tokens in training set. The small English data is from the Penn Treebank and the Arabic data is from the News-Commentary corpus. The rest are from the 2013 ACL Workshop on Machine Translation. $|\mathcal{C}|$ is large because of (rarely occurring) special characters.

et al., 2010]. With approximately 1m tokens and $|\mathcal{V}| = 10$k, this version has been extensively used by the language modeling community and is publicly available.[4]

With the optimal hyperparameters tuned on PTB, we apply the model to various morphologically rich languages: Czech, German, French, Spanish, Russian, and Arabic. Non-Arabic data comes from the 2013 ACL Workshop on Machine Translation,[5] and we use the same train/validation/test splits as in [Botha and Blunsom, 2014]. While the raw data are publicly available, we obtained the preprocessed versions from the authors,[6] whose morphological NNLM serves as a baseline for our work. We train on both the small datasets (DATA-S) with 1m tokens per language, and the large datasets (DATA-L) including the large English data which has a much bigger $|\mathcal{V}|$ than the PTB. Arabic data comes from the News-Commentary corpus,[7] and we perform our own preprocessing and train/validation/test splits.

In these datasets only singleton words were replaced with <unk> and hence we ef-

[4] http://www.fit.vutbr.cz/~imikolov/rnnlm/
[5] http://www.statmt.org/wmt13/translation-task.html
[6] http://bothameister.github.io/
[7] http://opus.lingfil.uu.se/News-Commentary.php

fectively use the full vocabulary. It is worth noting that the character model can utilize surface forms of OOV tokens (which were replaced with <unk>), but we do not do this and stick to the preprocessed versions (despite disadvantaging the character models) for exact comparison against prior work.

Optimization The models are trained by truncated backpropagation through time [Werbos, 1990, Graves, 2013]. We backpropagate for 35 time steps using stochastic gradient descent where the learning rate is initially set to 1.0 and halved if the perplexity does not decrease by more than 1.0 on the validation set after an epoch. On DATA-S we use a batch size of 20 and on DATA-L we use a batch size of 100 (for greater efficiency). Gradients are averaged over each batch. We train for 25 epochs on non-Arabic and 30 epochs on Arabic data (which was sufficient for convergence), picking the best performing model on the validation set. Parameters of the model are randomly initialized over a uniform distribution with support $[-0.05, 0.05]$.

For regularization we use dropout [Hinton et al., 2012] with probability 0.5 on the LSTM input-to-hidden layers (except on the initial Highway to LSTM layer) and the hidden-to-output softmax layer. We further constrain the norm of the gradients to be below 5, so that if the L_2 norm of the gradient exceeds 5 then we renormalize it to have $||\cdot|| = 5$ before updating. The gradient norm constraint was crucial in training the model. These choices were largely guided by previous work of Zaremba et al. [Zaremba et al., 2014] on word-level language modeling with LSTMs.

Finally, in order to speed up training on DATA-L we employ a hierarchical softmax [Morin and Bengio, 2005]—a common strategy for training language models with very large $|\mathcal{V}|$—instead of the usual softmax. We pick the number of clusters $c = \lceil\sqrt{|\mathcal{V}|}\rceil$ and randomly split $\mathcal{V}$ into mutually exclusive and collectively exhaustive subsets $\mathcal{V}_1, \ldots, \mathcal{V}_c$

		Small	Large
CNN	d	15	15
	w	$[1, 2, 3, 4, 5, 6]$	$[1, 2, 3, 4, 5, 6, 7]$
	h	$[25 \cdot w]$	$[\min\{200, 50 \cdot w\}]$
	f	tanh	tanh
Highway	l	1	2
	g	ReLU	ReLU
LSTM	l	2	2
	m	300	650

Table 2.2: Architecture of the small and large models. d = dimensionality of character embeddings; w = filter widths; h = number of filter matrices, as a function of filter width (so the large model has filters of width $[1, 2, 3, 4, 5, 6, 7]$ of size $[50, 100, 150, 200, 200, 200, 200]$ for a total of 1100 filters); f, g = nonlinearity functions; l = number of layers; m = number of hidden units.

of (approximately) equal size. In this application, we found that random simpy used random clusters[8]; Chapter 3 shows how to learn the hierarchy for improved performance. Then $\Pr(w_{t+1} = j | w_{1:t})$ becomes,

$$\begin{aligned} \Pr(w_{t+1} = j | w_{1:t}) = & \frac{\exp(\mathbf{h}_t \cdot \mathbf{s}^r + t^r)}{\sum_{r'=1}^{c} \exp(\mathbf{h}_t \cdot \mathbf{s}^{r'} + t^{r'})} \\ & \times \frac{\exp(\mathbf{h}_t \cdot \mathbf{p}_r^j + q_r^j)}{\sum_{j' \in \mathcal{V}_r} \exp(\mathbf{h}_t \cdot \mathbf{p}_r^{j'} + q_r^{j'})} \end{aligned} \tag{2.6}$$

where r is the cluster index such that $j \in \mathcal{V}_r$. The first term is simply the probability of picking cluster r, and the second term is the probability of picking word j given that cluster r is picked. We found that hierarchical softmax was not necessary for models trained on DATA-S.

English Penn Treebank We train two versions of our model to assess the trade-off between performance and size. Architecture of the small (LSTM-Char-Small) and large

[8]While Brown clustering/frequency-based clustering is commonly used in the literature (e.g. [Botha and Blunsom, 2014] use Brown clustering), we used random clusters as our implementation enjoys the best speed-up when the number of words in each cluster is approximately equal. We found random clustering to work surprisingly well.

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 [Mikolov and Zweig, 2012]	141.2	2 m
RNN† [Mikolov and Zweig, 2012]	124.7	6 m
RNN-LDA† [Mikolov and Zweig, 2012]	113.7	7 m
genCNN† [Wang et al., 2015]	116.4	8 m
FOFE-FNNLM† [Zhang et al., 2015a]	108.0	6 m
Deep RNN [Pascanu et al., 2013a]	107.5	6 m
Sum-Prod Net† [Cheng et al., 2014]	100.0	5 m
LSTM-1† [Zaremba et al., 2014]	82.7	20 m
LSTM-2† [Zaremba et al., 2014]	78.4	52 m

Table 2.3: Performance of our model versus other neural language models on the English Penn Treebank test set. PPL refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline.†For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

(LSTM-Char-Large) models is summarized in Table 2.2. As another baseline, we also train two comparable LSTM models that use word embeddings only (LSTM-Word-Small, LSTM-Word-Large). LSTM-Word-Small uses 200 hidden units and LSTM-Word-Large uses 650 hidden units. Word embedding sizes are also 200 and 650 respectively. These were chosen to keep the number of parameters similar to the corresponding character-level model.

As can be seen from Table 2.3, our large model is on par with the existing state-of-the-art (Zaremba et al. 2014), despite having approximately 60% fewer parameters. Our small model significantly outperforms other NNLMs of similar size, even though it is penalized by the fact that the dataset already has OOV words replaced with <unk> (other models are purely word-level models). While lower perplexities have been reported with model ensembles [Mikolov and Zweig, 2012], we do not include them here as they are

		DATA-S					
		CS	DE	ES	FR	RU	AR
Botha	KN-4	545	366	241	274	396	323
	MLBL	465	296	200	225	304	–
Small	Word	503	305	212	229	352	216
	Morph	414	278	197	216	290	230
	Char	401	260	182	189	278	196
Large	Word	493	286	200	222	357	172
	Morph	398	263	177	196	271	**148**
	Char	**371**	**239**	**165**	**184**	**261**	**148**

Table 2.4: Test set perplexities for DATA-S. First two rows are from [Botha, 2014] (except on Arabic where we trained our own KN-4 model) while the last six are from this paper. KN-4 is a Kneser-Ney 4-gram language model, and MLBL is the best performing morphological logbilinear model from Botha [Botha, 2014]. Small/Large refer to model size (see Table 2.2), and Word/Morph/Char are models with words/morphemes/characters as inputs respectively.

not comparable to the current work.

Other Languages The model's performance on the English PTB is informative to the extent that it facilitates comparison against the large body of existing work. However, English is relatively simple from a morphological standpoint, and thus our next set of results (and arguably the main contribution of this paper) is focused on languages with richer morphology (Table 2.4, Table 2.5).

We compare our results against the morphological log-bilinear (MLBL) model from [Botha, 2014], whose model also takes into account subword information through morpheme embeddings that are summed at the input and output layers. As comparison against the MLBL models is confounded by our use of LSTMs—widely known to outperform their feed-forward/log-bilinear cousins—we also train an LSTM version of the morphological NNLM, where the input representation of a word given to the LSTM is a summation of the word's morpheme embeddings. Concretely, suppose that $\mathcal{M}$ is the set of morphemes in a language, $\mathbf{M} \in \mathbb{R}^{n \times |\mathcal{M}|}$ is the matrix of morpheme embeddings,

and $\mathbf{m}^j$ is the j-th column of $\mathbf{M}$ (i.e. a morpheme embedding). Given the input word k, we feed the following representation to the LSTM:

$$\mathbf{x}^k + \sum_{j \in \mathcal{M}_k} \mathbf{m}^j \tag{2.7}$$

where $\mathbf{x}^k$ is the word embedding (as in a word-level model) and $\mathcal{M}_k \subset \mathcal{M}$ is the set of morphemes for word k. The morphemes are obtained by running an unsupervised morphological tagger as a preprocessing step.[9] We emphasize that the word embedding itself (i.e. $\mathbf{x}^k$) is added on top of the morpheme embeddings, as was done in Botha and Blunsom [Botha, 2014]. The morpheme embeddings are of size $200/650$ for the small/large models respectively. We further train word-level LSTM models as another baseline.

On DATA-S it is clear from Table 2.4 that the character-level models outperform their word-level counterparts despite, again, being smaller. The difference in parameters is greater for non-PTB corpora as the size of the word model scales faster with $|\mathcal{V}|$. For example, on Arabic the small/large word models have 35m/121m parameters while the corresponding character models have 29m/69m parameters respectively. The character models also outperform their morphological counterparts (both MLBL and LSTM architectures), although improvements over the morphological LSTMs are more measured. Note that the morpheme models have strictly more parameters than the word models because word embeddings are used as part of the input.

Due to memory constraints (all models were trained on GPUs with 2GB memory) we only train the small models on DATA-L (Table 2.5). Interestingly we do not observe significant differences going from word to morpheme LSTMs on Spanish, French, and

[9] We use *Morfessor Cat-MAP* [Creutz and Lagus, 2007], as in [Botha, 2014].

		DATA-L					
		CS	DE	ES	FR	RU	EN
Botha	KN-4	862	463	219	243	390	291
	MLBL	643	404	203	227	**300**	273
Small	Word	701	347	186	202	353	236
	Morph	615	331	189	209	331	233
	Char	**578**	**305**	**169**	**190**	313	**216**

Table 2.5: Test set perplexities on DATA-L. First two rows are from [Botha, 2014], while the last three rows are from the small LSTM models described in the paper. KN-4 is a Kneser-Ney 4-gram language model, and MLBL is the best performing morphological logbilinear model from [Botha, 2014]. Word/Morph/Char are models with words/morphemes/characters as inputs respectively.

English. The character models again outperform the word/morpheme models. We also observe significant perplexity reductions even on English when $\mathcal{V}$ is large. We conclude this section by noting that we used the same architecture for all languages and did not perform any language-specific tuning of hyperparameters.

Learned Word Representations We explore the word representations learned by the models on the PTB. Table 2.6 has the nearest neighbors of word representations learned from both the word-level and character-level models. For the character models we compare the representations obtained before and after highway layers. Before the highway layers the representations seem to solely rely on surface forms—for example the nearest neighbors of *you* are *your, young, four, youth*, which are close to *you* in terms of edit distance. The highway layers however, seem to enable encoding of semantic features that are not discernable from orthography alone. After highway layers the nearest neighbor of *you* is *we*, which is orthographically distinct from *you*. Another example is *while* and *though*—these words are far apart edit distance-wise yet the composition model is able to place them near each other. The model also makes some clear mistakes (e.g. *his* and *hhs*), highlighting the limits of our approach, although this could be due to the small

	In Vocabulary				
	while	*his*	*you*	*richard*	*trading*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*
	letting	*her*	*we*	*robert*	*advertising*
	though	*my*	*guys*	*neil*	*turnover*
	minute	*their*	*i*	*nancy*	*turnover*
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*
	whole	*hhs*	*young*	*rich*	*training*
	meanwhile	*is*	*four*	*richer*	*reading*
	white	*has*	*youth*	*richter*	*leading*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*
	whole	*this*	*your*	*gerard*	*training*
	though	*their*	*doug*	*edward*	*traded*
	nevertheless	*your*	*i*	*carl*	*trader*

	Out-of-Vocabulary		
	computer-aided	*misinformed*	*looooook*
LSTM-Word	–	–	–
	–	–	–
	–	–	–
	–	–	–
LSTM-Char (before highway)	*computer-guided*	*informed*	*look*
	computerized	*performed*	*cook*
	disk-drive	*transformed*	*looks*
	computer	*inform*	*shook*
LSTM-Char (after highway)	*computer-guided*	*informed*	*look*
	computer-driven	*performed*	*looks*
	computerized	*outperformed*	*looked*
	computer	*transformed*	*looking*

Table 2.6: Nearest neighbor words (based on cosine similarity) of word representations from the large word-level and character-level (before and after highway layers) models trained on the PTB. Last three words are OOV words, and therefore they do not have representations in the word-level model.

dataset. The learned representations of OOV words (*computer-aided*, *misinformed*) are positioned near words with the same part-of-speech. The model is also able to correct for incorrect/non-standard spelling (*looooook*), indicating potential applications for text normalization in noisy domains.

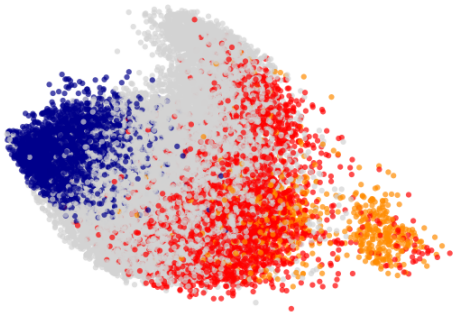

Figure 2.2: PCA of character n-gram representations for English. Colors correspond to: prefixes (red), suffixes (blue), hyphenated (orange), and all others (grey). Prefixes refer to character n-grams which start with the start-of-word character. Suffixes likewise refer to character n-grams which end with the end-of-word character.

Learned Character N-gram Representations As discussed previously, each filter of the CharCNN is essentially learning to detect particular character n-grams. Our initial expectation was that each filter would learn to activate on different morphemes and then build up semantic representations of words from the identified morphemes. However, upon reviewing the character n-grams picked up by the filters(i.e. those that maximized the value of the filter), we found that they did not (in general) correspond to valid morphemes. To get a better intuition for what the character composition model is learning, we plot the learned representations of all character n-grams (that occurred as part of at least two words in $\mathcal{V}$) via principal components analysis (Figure 2.2). We feed each character n-gram into the CharCNN and use the CharCNN's output as the fixed dimensional representation for the corresponding character n-gram. As is apparent from Figure 2.2, the model learns to differentiate between prefixes (red), suffixes (blue), and others (grey). We also find that the representations are particularly sensitive to character n-grams containing hyphens (orange), presumably because this is a strong signal of a word's part-of-speech.

	LSTM-Char	
	Small	Large
No Highway Layers	100.3	84.6
One Highway Layer	92.3	79.7
Two Highway Layers	90.1	78.9
One MLP Layer	111.2	92.6

Table 2.7: Perplexity on Penn Treebank.

Effect of the Highway Layers We quantitatively investigate the effect of highway network layers via ablation studies (Table 2.7). We train a model without any highway layers, and find that performance decreases significantly. As the difference in performance could be due to the decrease in model size, we also train a model that feeds $\mathbf{y}^k$ (i.e. word representation from the CharCNN) through a one-layer multilayer perceptron (MLP) to use as input into the LSTM. We find that the MLP does poorly, although this could be due to optimization issues. We hypothesize that highway networks are especially well-suited to work with CNNs, adaptively combining local features detected by the individual filters. CNNs have already proven to be been successful for many NLP tasks [Collobert et al., 2011, Shen et al., 2014, Kalchbrenner et al., 2014, Kim, 2014, Zhang et al., 2015b, Lei et al., 2015], and we posit that further gains could be achieved by employing highway layers on top of existing CNN architectures. We also anecdotally note that (1) having one to two highway layers was important, but more highway layers generally resulted in similar performance (though this may depend on the size of the datasets), (2) having more convolutional layers before max-pooling did not help, and (3) highway layers did not improve models that only used word embeddings as inputs.

Effect of Corpus/Vocab Sizes We next study the effect of training corpus/vocabulary sizes on the relative performance between the different models. We take the German

		$\lvert\mathcal{V}\rvert$			
		10 k	25 k	50 k	100 k
T	1 m	17%	16%	21%	–
	5 m	8%	14%	16%	21%
	10 m	9%	9%	12%	15%
	25 m	9%	8%	9%	10%

Table 2.8: Influence of vocabulary and training data size on perplexity improvement. Perplexity reductions by going from small word-level to character-level models based on different corpus/vocabulary sizes on German (DE). $|\mathcal{V}|$ is the vocabulary size and T is the number of tokens in the training set. The full vocabulary of the 1m dataset was less than 100k and hence that scenario is unavailable.

(DE) dataset from DATA-L and vary the training corpus/vocabulary sizes, calculating the perplexity reductions as a result of going from a small word-level model to a small character-level model. To vary the vocabulary size we take the most frequent k words and replace the rest with <unk>. As with previous experiments the character model does not utilize surface forms of <unk> and simply treats it as another token. Although Table 2.8 suggests that the perplexity reductions become less pronounced as the corpus size increases, we nonetheless find that the character-level model outperforms the word-level model in all scenarios.

Further Observations Combining word embeddings with the CharCNN's output to form a combined representation of a word (to be used as input to the LSTM) resulted in slightly worse performance (81 on PTB with a large model). This was surprising, as improvements have been reported on part-of-speech tagging [dos Santos and Zadrozny, 2014] and named entity recognition [dos Santos and Guimarães, 2015] by concatenating word embeddings with the output from a character-level CNN. While this could be due to insufficient experimentation on our part,[10] it suggests that for some tasks, word

[10]We experimented with (1) concatenation, (2) tensor products, (3) averaging, and (4) adaptive weighting schemes whereby the model learns a convex combination of word embeddings and the CharCNN

embeddings are superfluous—character inputs are good enough.

While our model requires additional convolution operations over characters and is thus slower than a comparable word-level model which can perform a simple lookup at the input layer, we found that the difference was manageable with optimized GPU implementations—for example on PTB the large character-level model trained at 1500 tokens/sec compared to the word-level model which trained at 3000 tokens/sec. For scoring, our model can have the same running time as a pure word-level model, as the CharCNN's outputs can be pre-computed for all words in $\mathcal{V}$. This would, however, be at the expense of increased model size, and thus a trade-off can be made between run-time speed and memory (e.g. one could restrict the pre-computation to the most frequent words).

Thus, we have introduced a neural language model that utilizes only character-level inputs for word level language modeling. Despite having fewer parameters, our model outperforms baseline models that utilize word/morpheme embeddings in the input layer, showing the advantage of using a flexible enough character composition model. Analysis of word representations obtained from the character composition part of the model further indicates that the model is able to encode, from characters only, rich semantic and orthographic features.

2.4 Adaptive Character Level Encoding: Model

Section 2.3 shows how useful reading text at the character level can be. It appears natural, then, to take this idea further and see whether we can do away with using word separations entirely, for example by applying a recurrent neural network directly to the

outputs.

sequence of characters.

Most existing recurrent models take one of two approaches regarding the amount of computation they require. Either the computational load is constant over time, or it follows a fixed (or deterministic) schedule [Koutník et al., 2014, Mikolov et al., 2014]. The latter approach has proven especially useful when dealing with sequences which reflect processes taking place at different levels (and time scales) [Bojanowski et al., 2015]. However, we believe that taking a more flexible approach could prove useful. In this work, we show how to modify two commonly used recurrent unit architectures, namely the Elman and Gated Recurrent Unit, to obtain their variable computation counterparts. This gives rise to two new architecture, the Variable Computation RNN and Variable Computation GRU (VCRNN and VCGRU), which take advantage of these phenomena by deciding at each time step how much computation is required based on the current hidden state and input. We show that the models learn time patterns of interest, can perform fewer operations, and may even take advantage of these time structures to produce better predictions than the constant computation versions.

The bulk of the computation in most character level recurrent neural network models comes from the linear layers; a natural option to reduce the number of operations would then be to only apply the linear transformations to a sub-set of the hidden dimensions. These could in theory correspond to any sub-set indices in $\{1, \ldots, D\}$; however, we want a setting where the computational cost of the choice is much less than the cost of computing the new hidden state. Thus, we only consider the sets of first d dimensions of $\mathbb{R}^D$, so that there is a single parameter d to compute.

Our Variable Computation Units (VCUs) implement this idea using two modules: a *scheduler* decides how many dimensions need to be updated at the current time step, and the VCU performs a *partial update* of its hidden state accordingly, as illustrated in

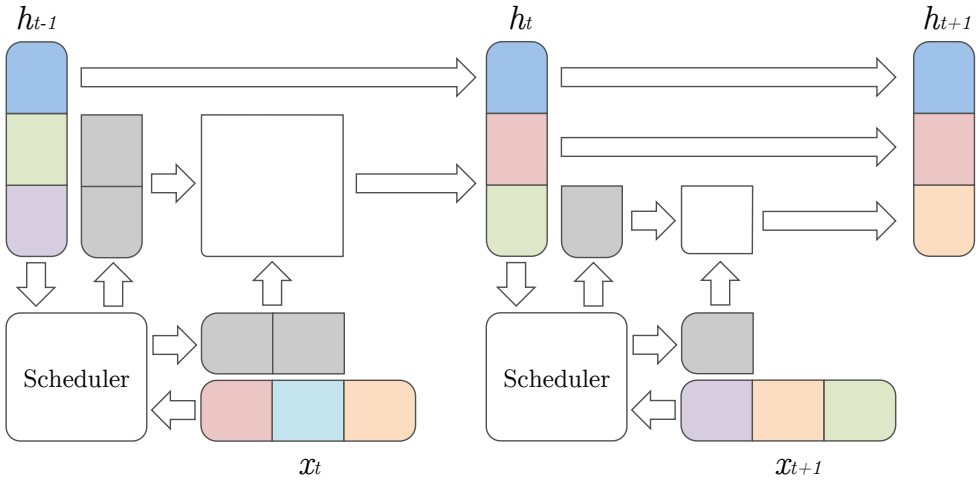

Figure 2.3: Two time steps of a VCU. At each step t, the scheduler takes in the current hidden vector $\mathbf{h}_{t-1}$ and input vector $\mathbf{x}_t$ and decides on a number of dimensions to use d. The unit then uses the first d dimensions of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ to compute the first d elements of the new hidden state $\mathbf{h}_t$, and carries the remaining $D - d$ dimensions over from $\mathbf{h}_{t-1}$.

Figure 2.3. In the rest of this Section, we first describe the scheduler and partial update operations, then outline the procedure to jointly learn both modules.

Scheduler. The model first needs to decide how much computation is required at the current time step. To make that decision, the recurrent unit has access to the current hidden state and input; this way, the model can learn to ignore an uninformative input, or to decide on more computation when it is unexpected given the current hidden state. The *scheduler* is then defined as a function $m : \mathbb{R}^{2D} \rightarrow [0, 1]$ which decides what portion of the hidden state to change based on the current hidden and input vectors. In this work, we decide to implement it as a simple log-linear function with parameter vectors $\mathbf{u}$ and $\mathbf{v}$, and bias b, and at each time step t, we have:

$$m_t = \sigma(\mathbf{u} \cdot \mathbf{h}_{t-1} + \mathbf{v} \cdot \mathbf{x}_t + b). \tag{2.8}$$

Partial update. Once the scheduler has decided on a computation budget m_t, the VCU needs to perform a *partial update* of the first $\lceil m_t D \rceil$ dimensions of its hidden state. Recall the hidden state $\mathbf{h}_{t-1}$ is a D-dimensional vector. Given a smaller dimension $d \in \{1, \ldots, D\}$, a partial update of the hidden state would take the following form. Let g_d be the d-dimensional version of the model's recurrence function g as defined in Equation 1.3, which uses the upper left d by d square sub-matrices of the linear transformations ($\mathbf{U}_d, \mathbf{V}_d, \ldots$), and $\mathbf{h}_{t-1}^d$ and $\mathbf{x}_t^d$ denote the first d elements of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$. We apply g_d to $\mathbf{h}_{t-1}^d$ and $\mathbf{x}_t^d$, and carry dimensions $d+1$ to D from the previous hidden state, so the new hidden state $\mathbf{h}_t$ is defined by:

$$\mathbf{h}_t^d = g_d(\mathbf{h}_{t-1}^d, \mathbf{x}_t^d) \quad \text{and} \quad \forall i > d,\ h_{t,i} = h_{t-1,i}.$$

Soft mask. In practice, the transition function we just defined would require making a hard choice at each time step of the number of dimensions to be updated, which makes the model non-differentiable and can significantly complicate optimization. Instead, we approximate the hard choice by using a gate function to apply a *soft mask*. Given $m_t \in [0, 1]$ and a sharpness parameter λ, we use the gating vector $\mathbf{e}_t \in \mathbb{R}^D$ defined by:

$$\forall i \in 1, \ldots, D,\ (e_t)_i = \text{Thres}_\epsilon\left(\sigma\big(\lambda(m_t D - i)\big)\right), \tag{2.9}$$

where Thres_ϵ maps all values greater than $1-\epsilon$ and smaller than ϵ to 1 and 0 respectively (in practice, this simply means that the model does not need to compute an update when the mask value would be smaller than machine precision, since the sigmoid function never actually reaches 0). That way, the model performs an update using the first $(m_t \times D + \eta)$ dimensions of the hidden state, where η goes to 0 as λ increases, and leaves its last $((1 - m_t) \times D - \eta)$ dimensions unchanged. Thus, if g is the recurrence

function defined in Equation 1.2, we have:

$$\bar{\mathbf{h}}_{t-1} = \mathbf{e}_t \odot \mathbf{h}_{t-1}, \quad \bar{\mathbf{i}}_t = \mathbf{e}_t \odot \mathbf{x}_t, \quad \text{and} \quad \mathbf{h}_t = \mathbf{e}_t \odot g(\bar{\mathbf{h}}_{t-1}, \bar{\mathbf{x}}_t) + (1 - \mathbf{e}_t) \odot \mathbf{h}_t. \tag{2.10}$$

The computational cost of this model at each step t, defined as the number of multiplications involving possibly non-zero elements is then $O(m_t^2 D^2)$.

Variable Computation Elman and Gated Recurrent Unit. First, we derive a variable computation version of the Elman RNN to get the Variable Computation Recurrent Neural Network (VCRNN) by transforming Equation 1.3 as follows:

$$\mathbf{h}_t = \mathbf{e}_t \odot \sigma(\mathbf{U}\bar{\mathbf{h}}_{t-1} + \mathbf{V}\bar{\mathbf{x}}_t) + (1 - \mathbf{e}_t) \odot \mathbf{h}_t. \tag{2.11}$$

Secondly, we obtain the Variable Computation Gated Recurrent Unit (VCGRU) by deriving the variable computation of the GRU architecture. This is achieved by modifying the Equations 1.5 as follows:

$$\mathbf{r}_t = \sigma(\mathbf{U}^r\bar{\mathbf{h}}_{t-1} + \mathbf{V}^r\bar{\mathbf{x}}_t), \qquad \mathbf{z}_t = \mathbf{e}_t \odot \sigma(\mathbf{U}^z\bar{\mathbf{h}}_{t-1} + \mathbf{V}^z\bar{x}_t) \tag{2.12}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{U}^h(\mathbf{r}_t \odot \bar{\mathbf{h}}_{t-1}) + \mathbf{V}^h\bar{\mathbf{x}}_t) \tag{2.13}$$

And:

$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \tag{2.14}$$

Learning Since the *soft mask* $\mathbf{e}_t$ is a continuous function of the model parameters, the scheduler can be learned through back-propagation. However, we have found that the naive approach of using a fixed sharpness parameter and simply minimizing the

negative log-likelihood defined in Equation 1.1 led to the model being stuck in a local optimum which updates all dimensions at every step. We found that the following two modifications allowed the model to learn better parameterizations.

First, we can encourage m_t to be either close or no greater than a target $\bar{m}$ at all time by adding a penalty term Ω to the objective. For example, we can apply a ℓ_1 or ℓ_2 penalty to values of m_t that are greater than the target, or that simply diverge from it (in which case we also discourage the model from using too few dimensions). The cost function defined in Equation 1.1 then becomes:

$$\mathcal{O}(\mathbf{w}, \mathbf{U}, \mathbf{V}, \mathbf{u}, \mathbf{v}, b) = \mathcal{NL}(\mathbf{w}, \mathbf{U}, \mathbf{V}, \mathbf{u}, \mathbf{v}, b) + \Omega(m, \bar{m}). \tag{2.15}$$

Secondly, for the model to be able to explore the effect of using fewer or more dimensions, we need to start training with a smooth mask (small λ parameter), since for small values of λ, the model actually uses the whole hidden state. We can then gradually increase the sharpness parameter until the model truly does a partial update.

2.5 Adaptive Character Level Encoding: Experiments

We ran experiments with the Variable Computation variants of the Elman and Gated Recurrent Units (VCRNN and VCGRU respectively) on several sequence modeling tasks. All experiments were run using a symmetrical ℓ_1 penalty on the scheduler m, that is, penalizing m_t when it is greater or smaller than target $\bar{m}$, with $\bar{m}$ taking various values in the range $[0.2, 0.5]$. In all experiments, we start with a sharpness parameter $\lambda = 0.1$, and increase it by 0.1 per epoch to a maximum value of 1.

In each of our experiments, we are interested in investigating two specific aspects of our model. On the one hand, do the time patterns that emerge agree with our intu-

ition of the time dynamics expressed in the data? On the other hand, does the Variable Computation Unit (VCU) yield a good predictive model? More specifically, does it lead to lower perplexity than a constant computation counterpart which performs as many or more operations? In order to be able to properly assess the efficiency of the model, and since we do not know *a priori* how much computation the VCU uses, we always report the "equivalent RNN" dimension (noted as RNN-d in Table 2.11) along with the performance on test data, *i.e.* the dimension of an Elman RNN that would have performed the same amount of computation. Note that the computational complexity gains we refer to are exclusively in terms of lowering the number of operations, which does not necessarily correlate with a speed up of training when using general purpose GPU kernels; it is however a prerequisite to achieving such a speed up with the proper implementation, motivating our effort.

We answer both of these questions on the tasks of music modeling, bit and character level language modeling on the Penn Treebank text, and character level language modeling on the Text8 data set as well as two languages from the Europarl corpus.

Music Modeling We downloaded a corpus of Irish traditional tunes from *thesession.org*[11] and split them into a training validation and test of 16,000 (2.4M tokens), 1,511 (227,000 tokens) and 2,000 (288,000 tokens) melodies respectively. Each sub-set includes variations of melodies, but no melody has variations across subsets. We consider each (pitch, length) pair to be a different symbol; with rests and bar symbols, this comes to a total vocabulary of 730 symbols.

Table 2.9 compares the perplexity on the test set to Elman RNNs with equivalent computational costs: an VCRNN with hidden dimension 500 achieves better perplexity

[11] https://thesession.org

unit type	equivalent RNN	perplexity
RNN-200	–	9.13
RNN-250	–	8.70
VCRNN-500	233	8.51

Table 2.9: Music modeling, test set perplexity on a corpus of traditional Irish tunes. Our model manages to achieve better perplexity with less computation than the Elman RNN.

with fewer operations than an RNN with dimension 250.

Looking at the output of the scheduler on the validation set also reveals some interesting patterns. First, bar symbols are mostly ignored: the average value of m_t on bar symbols is 0.14, as opposed to 0.46 on all others. This is not surprising: our preprocessing does not handle polyphony or time signatures, so bars en up having different lengths. The best thing for the model to do is then just to ignore them and focus on the melody. Similarly, the model spends less computation on rests (0.34 average m_t), and pays less attention to repeated notes (0.51 average for m_t on the first note of a repetition, 0.45 on the second).

We also notice that the model needs to do more computation on fast passages, which often have richer ornamentation, as illustrated in Table 2.10. While it is difficult to think *a priori* of all the sorts of behaviors that could be of interest, these initial results certainly show a sensible behavior of the scheduler on the music modeling task.

note length	0.25	1/3	0.5	0.75	1	1.5	2
average m	0.61	0.77	0.39	0.59	0.44	0.46	0.57

Table 2.10: Average amount of computation (m_t) for various note lengths. More effort is required for the faster passages with 16th notes and triplets.

Penn TreeBank and Text8 We also chose to apply our model to the tasks of bit level and character level language modeling. Those appeared as good applications since we know *a priori* what kind of temporal structure to look for: ASCII encoding means that we expect a significant change (change of character) every 8 bits in bit level modeling, and we believe the structure of word units to be useful when modeling text at the character level.

We first ran experiments on two English language modeling tasks, using the Penn TreeBank and Text8 data sets. We chose the former as it is a well studied corpus, and one of the few corpora for which people have reported bit-level language modeling results. It is however quite small for our purposes, with under 6M characters, which motivated us to apply our models to the larger Text8 data set (100M characters). Table 2.11 shows bit per bit and bit per character results for bit and character level language modeling. We compare our results with those obtained with standard Elman RNN, GRU, and LSTM networks, as well as with the Conditional RNN of [Bojanowski et al., 2015].

Quantitative Results. We first compare the VCRNN to the regular Elman RNN, as well as to the Conditional RNN of [Bojanowski et al., 2015], which combines two layers running at bit and character level for bit level modeling, or character and word level for character level modeling. For bit level language modeling, the VCRNN not only performs fewer operations than the standard unit, it also achieves better performance. For character level modeling, the Elman model using a hidden dimension of 1024 achieved 1.47 bits per character, while our best performing VCRNN does slightly better while only requiring as much computation as a dimension 760 Elman unit. While we do slightly more computation than the Conditional RNN, it should be noted that our model is not explicitly given word-level information: it learns how to summarize it from

Bit level PTB

unit type	RNN-d	bpb
RNN-100	100	0.287
RNN-500	500	0.227
RNN-1000	1000	0.223
CRNN-100	140	0.222
VCRNN-1000	340	0.231
VCRNN-1000	460	**0.215**

Character level PTB

unit type	RNN-d	bpc
GRU-1024	1450	**1.42**
LSTM-1024	2048	**1.42**
RNN-1024	1024	1.47
CRNN-500	700	1.46
VCRNN-1024	760	1.46
RNN-760	760	1.47
LSTM-380	760	1.44
GRU-538	760	1.43
VCGRU-1024	648	**1.42**
LSTM-324	648	1.46
GRU-458	648	1.47

Character level Text8

unit type	$\bar{m}$	RNN-d	bpc
RNN-512*	-	512	1.80
RNN-1024*	-	1024	1.69
LSTM-512*	-	1024	1.65
LSTM-1024*	-	2048	**1.52**
RNN-512	-	512	1.80
GRU-512	-	725	1.69
GRU-1024	-	1450	1.58
VCGRU-1024	0.3	464	1.69
VCGRU-1024	0.4	648	1.64
VCGRU-1024	0.5	820	1.63

Table 2.11: **Top left:** Bits per character for character level language modeling on Penn TreeBank. CRNN refers to the Conditional RNN from [Bojanowski et al., 2015]. **Top right:** Bits per bit for bit level language modeling on Penn TreeBank. **Bottom:** Bits per character for character level language modeling on Text8. *From [Zhang et al., 2016b]

character-level input.

The comparison between the constant computation and Variable Computation GRU (VCGRU) follows the same pattern, both on the PTB and Text8 corpora. On PTB, the VCGRU with the best validation perplexity performs as well as a GRU (and LSTM) of the same dimension with less than half the number of operations. On Text8, the VCGRU models with various values of the target $\bar{m}$ always achieve better perplexity than other models performing similar or greater numbers of operations. It should be noted that none of the models we ran on Text8 overfits significantly (the training and validation

perplexities are the same), which would indicate that the gain is not solely a matter of regularization.

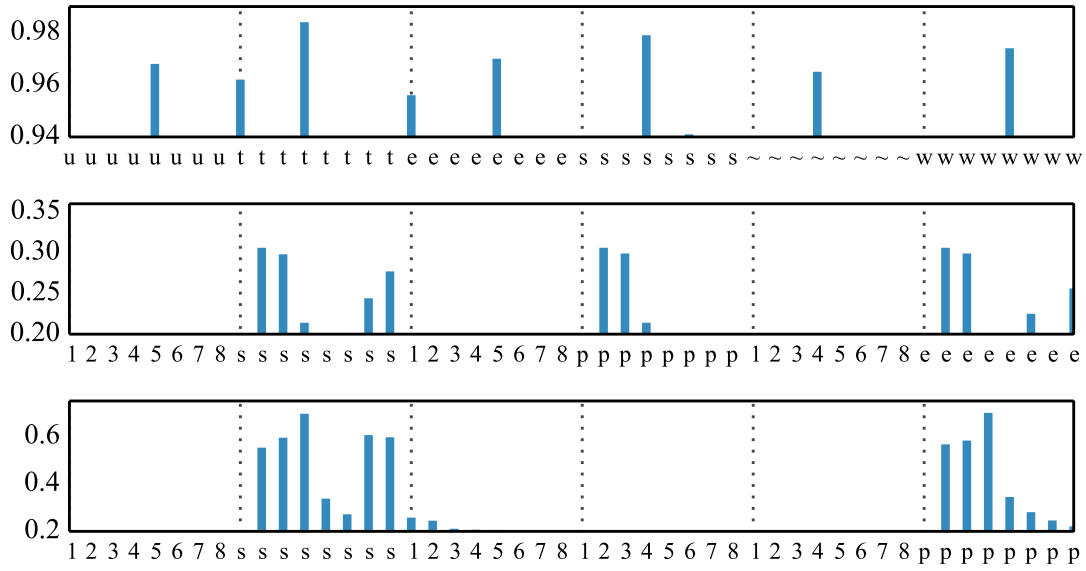

Figure 2.4: **Top:** Per-bit computation by VCRNN, higher dimensions (950 to 1000). **Middle**: adding 8 bits of buffer between every character. **Bottom**: adding 24 bits of buffer between each character.

Bit Level Scheduler. The scheduler in the bit level language model manages to learn the structure of ASCII encoding: Figure 2.4 shows that the higher dimensions are modified roughly every 8 bits. We also created some artificial data by taking the PTB text and adding 8 or 24 0 bits between each character. Figure 2.4, shows that the model learns to mostly ignore these "buffers", doing most of its computation on actual characters.

Character Level Scheduler. On character level language modeling, the scheduler learns to make use of word boundaries and some language structures. Figure 2.5 shows that the higher dimensions are used about once per words, and in some cases, we even observe a spike at the end of each morpheme (long-stand-ing, as shown in Figure 2.7).

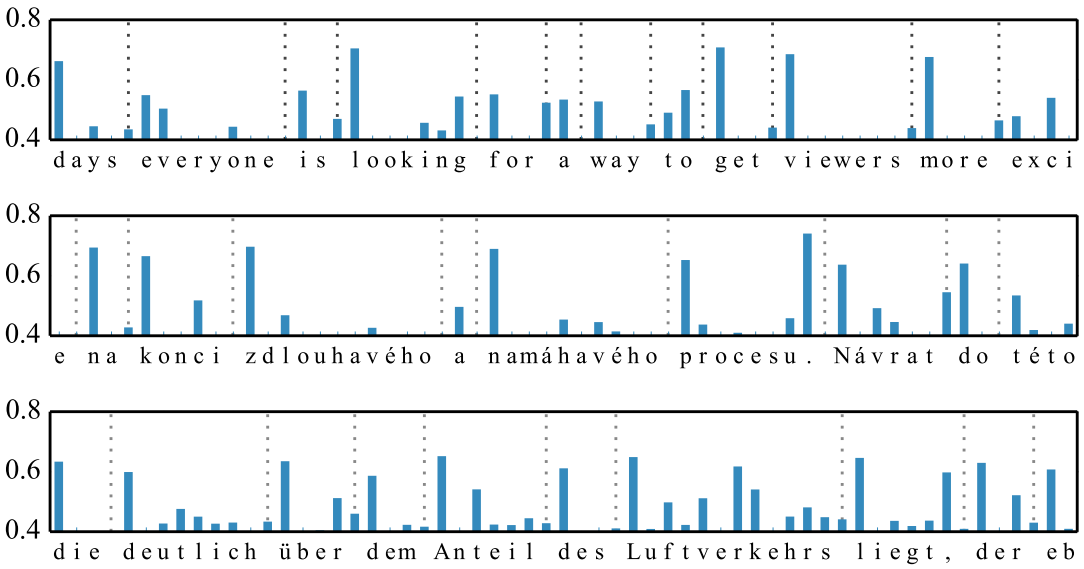

Figure 2.5: Per-character computation by VCRNN. Top: English. Middle: Czech. Bottom: German. All languages learn to make use of word units.

While we provide results for the VCRNN specifically in this Section, the VCGRU scheduler follows the same patterns. Early experiments on a text where the spaces had been artificially removed showed similar patterns, although less marked, indicating that the model's scheduler makes some use of the information provided by word boundaries.

Europarl Czech and German We also ran our model on two languages form the Europarl corpus. We chose Czech, which has a larger alphabet than other languages in the corpus, and German , which is a language that features long composite words without white spaces to indicate a new unit. Both are made up of about 20M characters. We tried two settings. In the "guide" setting, we use the penalty on m_t to encourage the model to use more dimensions on white spaces. The "learn" setting is fully unsupervised, and encourages lower values of m_t across the board.

Figure 2.6 shows that both perform similarly on the Czech dataset, achieving better

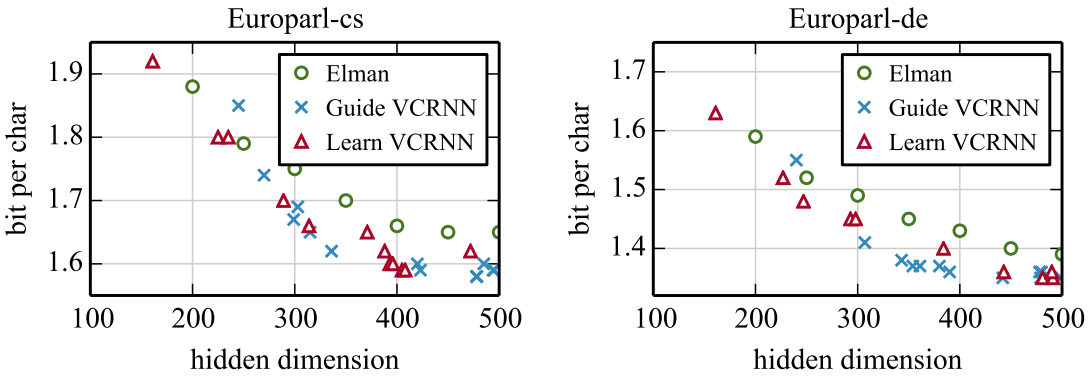

Figure 2.6: Bits per character for different computational loads on the Europarl Czech (left) and German (right) datasets. The VCRNN, whether guided to use boundaries or fully unsupervised, achieves better held-out log-likelihood more efficiently than the standard RNN.

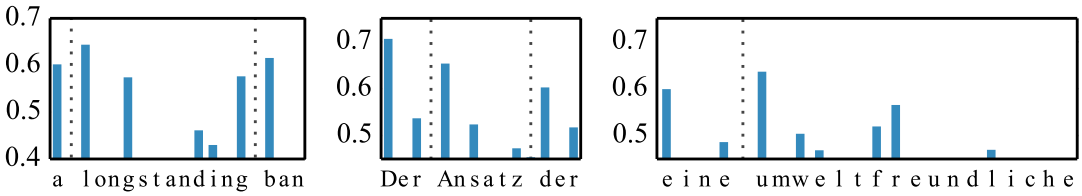

Figure 2.7: Per-character computation by VCRNN. The model appears to make use of morphology, separating sub-word units.

performance more efficiently than the standard RNN. On German, the guided settings remains slightly more efficient than the fully learned one, but both are more efficient than the RNN and achieve the same performance when using more dimensions. Both learn to use more dimensions at word boundaries as shown in Figure 2.5. The German model also appears to be learning interesting morphology (Luft-ver-kehrs, eben-falls in Figure 2.5, An-satz, Um-welt-freund-lich in Figure 2.7), and grammar (focusing on case markers at the end of articles, Figure 2.7).

2.6 Discussion

In this Chapter, we have presented two ways of using character level inputs for language processing systems. The performance gains of the Character Aware LSTM, especially on morphologically rich languages, shows that not only can the characters inform the word embeddings, they can be sufficient to model all of their relevant properties. Indeed, on the English Penn Treebank the model is on par with the existing state-of-the-art despite having 60% fewer parameters. On languages with rich morphology (Arabic, Czech, French, German, Spanish, Russian), the model outperforms word-level or morpheme-level LSTM baselines, again with fewer parameters. This agrees with our initial intuition: by giving the model the ability to explicitly take advantage of what linguistics and morphology in particular tell us are significant phenomena occurring at the character level, we gained both data efficiency and overall performance at the cost of some additional computation.

This would tend to suggest that a fully character-level LSTM or other recurrent model with sufficient capacity might do better than a word-level one in most cases, and indeed there have been recent successes in using such models e.g. in machine translation [Lee et al., 2017]. However, since these models need to update their latent state at each time step, they can be extremely computationally costly, and may have trouble keeping track of information over too many time steps. We took a first step towards alleviating this difficulty through the proposed Variable Computation Recurrent Neural Networks. By giving a recurrent model the ability to efficiently and explicitly decide how much of its latent state it needs to update at each time step, we show that not only can it be more computation efficient while reaching better performances, the computation patterns often agree with our intuitions about morphology and higher level

reasoning for word level modeling. We hope to develop our model further in future work to exploit its capacity better: promising directions include explicitly conditioning the scheduler on morphological features other than the recurrent hidden state, or using morphological knowledge or prediction entropy as an additional signal during training.

Recent Related Work and Future Directions We put forward two hybrid systems which realize different trade-offs between the respective advantages of processing text as a sequence of characters or as a sequence of words. More specifically, one mostly word-level system whose input is augmented with character-level information, and one mostly character-level model whose input is augmented with the ability to handle larger units. In the rest of this Section, we consider how these paradigms have been used in concurrent and more recent works.

One of the cheapest ways of augmenting word level inputs with subword information consists in deterministically splitting words into relevant word pieces or word parts, and treating those as the new units to process. One approach which has become very popular recently (e.g. [Vaswani et al., 2017]) consists in using Byte Pair Encoding (BPE) to construct a data-driven vocabulary of word pieces, following [Sennrich et al., 2016]: among other things, this pre-processing step is currently deployed in very large-scale neural machine translation systems as described in [Wu et al., 2016]. The word piece vocabulary size is usually of the same order of size as $\mathcal{V}$, and the method naturally adapts to the morphological complexity of a language, making this approach a particularly computation-efficient way to add in morphemes or switches to character-level prediction when necessary. This makes the BPE method a good choice when computation is an issue, at the cost of relying on a somewhat crude heuristic to find relevant word parts.

The method proposed in [Bojanowski et al., 2017] is more flexible than BPE while

remaining computationally cheaper than the Character Aware LSTM. There, rather than obtaining representations for character n-grams through convolutions, the authors simply learn them independently, and sum them to obtain a word embedding. The method is kept tractable by using a hash table for all character n-grams, the assumption being that the embedding size is sufficient to handle collisions. This yields an architecture which is more computation-efficient than ours at the cost of memory requirements and loss of expressiveness in the way the word parts are combined.

All of the above, up to and including our own Character-aware LSTM, correspond to different trade offs between computational complexity, performance and memory requirements by considering different ways to integrate character information in the model input. The obvious next question is then whether something similar can be achieved in the output (beyond predicting word pieces given by BPE).

[Józefowicz et al., 2016] attempts to augment word representations on the prediction side a recurrent neural network language model as well as the input side: they try two approaches, the first one parameterizing word embeddings in the softmax using the character CNN we presented in Section 2.2, and the second by using a character-level LSTM to compute the likelihood of each word as a sequence of characters (similarly to [Bojanowski et al., 2015]). However, while both provide similar gains in memory efficiency and model size, they also lag behind independent word embeddings in terms of performance. Arguably, the latter also provides a rough upper bound on the performance of multi-scale fully character level RNN language models such as our Variational Computation RNN, or the systems presented in [Chung et al., 2016] or [Graves, 2016].

In summary, while more and more models routinely leverage sub-word information on the input side to obtain state-of-the-art results on a variety of language task, there is still some work to be done before a fully character level language model can reach

the same performance as a word level system, or before we can efficiently compute the likelihood of a next word in a character aware fashion.

Chapter 3

Speeding up Word Level Language Modeling

In the next part of this thesis, we consider language modeling as a language learning objective. More specifically, in this Chapter, we follow the setting presented in Section 1.4, where we compute a probability distribution for each word given its left context. However, one of the main issues with this setting is the computational cost of computing a probability distribution over a large vocabulary for each token in a corpus. To remedy this difficulty, we propose to resort to hierarchical prediction: replacing one costly operation to compute the likelihood of a word with a series of much cheaper ones. While this is a known technique for speeding up language modeling, we differ from other work by providing an efficient theoretically motivated algorithm to learn a hierarchical structure over the vocabulary in an online fashion, yielding performance gains and potentially useful tree structures.

3.1 Introduction

Central to the language modeling problem is the challenge of scale. It is typical for languages to have lexicons of hundreds of thousands of word types, and language models themselves are often estimated on corpora with billions of tokens [Graff and Cieri, 2003]. The scale of the problem inherently limits the space of distributions that can be effectively applied. Hierarchical prediction can help alleviate this difficulty: for a corpus with C tokens, a hidden representation of dimension D and a vocabulary of size $|\mathcal{V}|$, the complexity of computing the likelihood (and its gradients) can go from $O(C \times D \times |\mathcal{V}|)$ for a flat predictor to $O(C \times D \times \log(|\mathcal{V}|))$ for a hierarchical one. However, the optimal hierarchy for the predictor can depend both on the data and on the text representation function. In this work, we propose to learn both jointly.

The models in this chapter are most closely related to the Log Bi-Linear (LBL) neural n-gram language model of [Mnih and Hinton, 2008]. First, using the chain rule and an order T Markov assumption we model the probability of a sentence $(w)_{1,N} = (w_1, w_2, \ldots, w_N)$ as:

$$p(w_1, w_2, \ldots, w_N) = \prod_{t=1}^{N} p(w_t | w_{t-T,\ldots,t-1})$$

Similarly to their work, we also use a low dimensional representation of the context $(w_{t-T,\ldots,t-1})$. In this setting, each word w in the vocabulary $\mathcal{V}$ has an embedding of dimension D: $\mathbf{U}_w \in \mathbb{R}^D$. A given context $x = (w_{t-T}, \ldots, w_{t-1})$ corresponding to position t is then represented by a context embedding vector $\mathbf{r}_x$ such that

$$\mathbf{r}_x = \sum_{k=1}^{T} \mathbf{R}_k \mathbf{U}_{w_{t-k}},$$

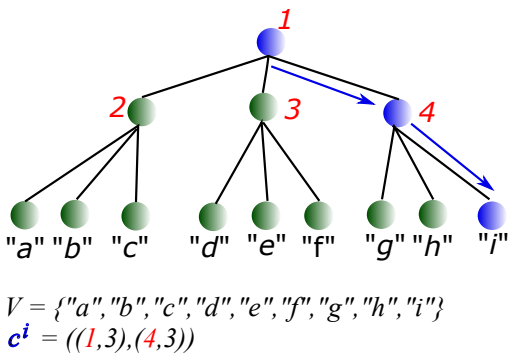

Figure 3.1: Hierarchical predictor: in order to predict label "i", the system needs to choose the third child of node 1, then the third child of node 4.

where $\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times D}$ is the embedding matrix, and $\mathbf{R}_k \in \mathbb{R}^{D \times D}$ is the transition matrix associated with the k^{th} context word.

The most straight-forward way to define a probability function is then to define the distribution over the next word given the context representation as a softmax, as done in [Mnih and Hinton, 2007]. That is:

$$\begin{aligned} p(w_t = i|x) &= \text{softmax}_i(\mathbf{r}_x^\top \mathbf{U} + \mathbf{b}) \\ &= \frac{\exp(\mathbf{r}_x^\top \mathbf{U}_i + b_i)}{\sum_{w \in \mathcal{V}} \exp(\mathbf{r}_x^\top \mathbf{U}_w + b_w)}, \end{aligned}$$

where b_w is the bias for word w. However, the complexity of computing this probability distribution in this setting is $O(|\mathcal{V}| \times D)$, which can be prohibitive for large corpora and vocabularies.

Instead, [Mnih and Hinton, 2008] takes a hierarchical approach to the problem. They construct a binary tree, where each word $w \in \mathcal{V}$ corresponds to a leaf, and can thus be identified with the path from the root to the corresponding leaf by making a sequence of choices of going left versus right (as illustrated in Figures 3.2 and 3.3). In this setting, each word in the vocabulary corresponds to a unique leaf in a binary tree, and is iden-

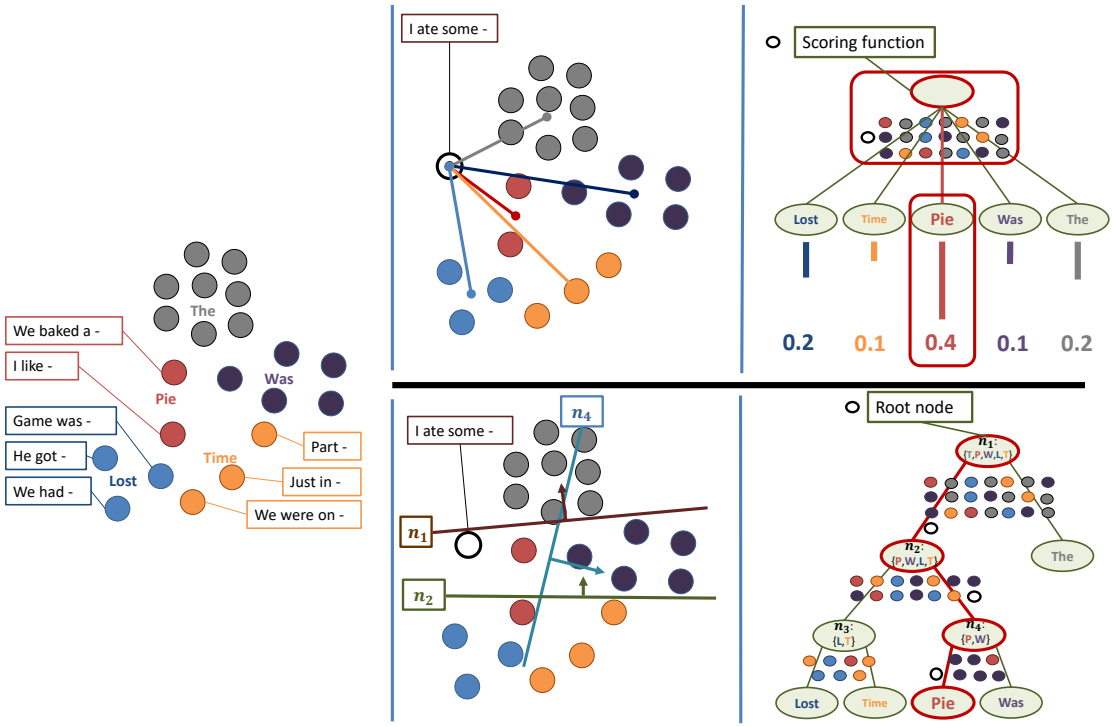

Figure 3.2: Comparing softmax functions for NNLMs. Each context in a corpus (e.g. *"We baked a..."*, *"We were on..."*) has an embedding in $\mathbb{R}^2$ and a target next word in the vocabulary $\mathcal{V} = \{lost, time, pie, wasthe\}$. When the embedding for a new context (*"I ate some..."*) is given, we want to compute the likelihood of the target class (*pie*).
Top: Flat Softmax. The scoring function computes a similarity metric between the new embedding and each of the classes, which yields a score, then normalizes these using a softmax function. This takes $O(|\mathcal{V}|)$ operations. Here, the system makes a five-way prediction between $lost$, $time$, was, and pie.
Bottom: Hierarchical Softmax. The words in the vocabulary correspond to the leaves of a tree. The likelihood of a class is the product of the probabilities of each child on the path from root to leaf. This takes $O(\log(|\mathcal{V}|))$ operations. Here, the path to the leaf corresponding to the class "Pie" is $((n_1, 0), (n_2, 1), (n_4, 0))$, hence the likelihood is obtained with three binary predictions (corresponding to the choice of a child at each of the nodes n_1, n_2, and n_4).

tified with the path from the root to that leaf: $((n_1^i, d_1^i), \ldots, (n_{L_i}^i, d_{L_i}^i))$, where L_i is the depth of the leaf corresponding to word i, $n_l \in \{1, \ldots, N\}$ are the nodes of the tree and $d_l \in \{0, 1\}$ correspond to the decision of going to the left or right child. The likelihood of sending a context x to the right child of a node n is then parameterized by the node

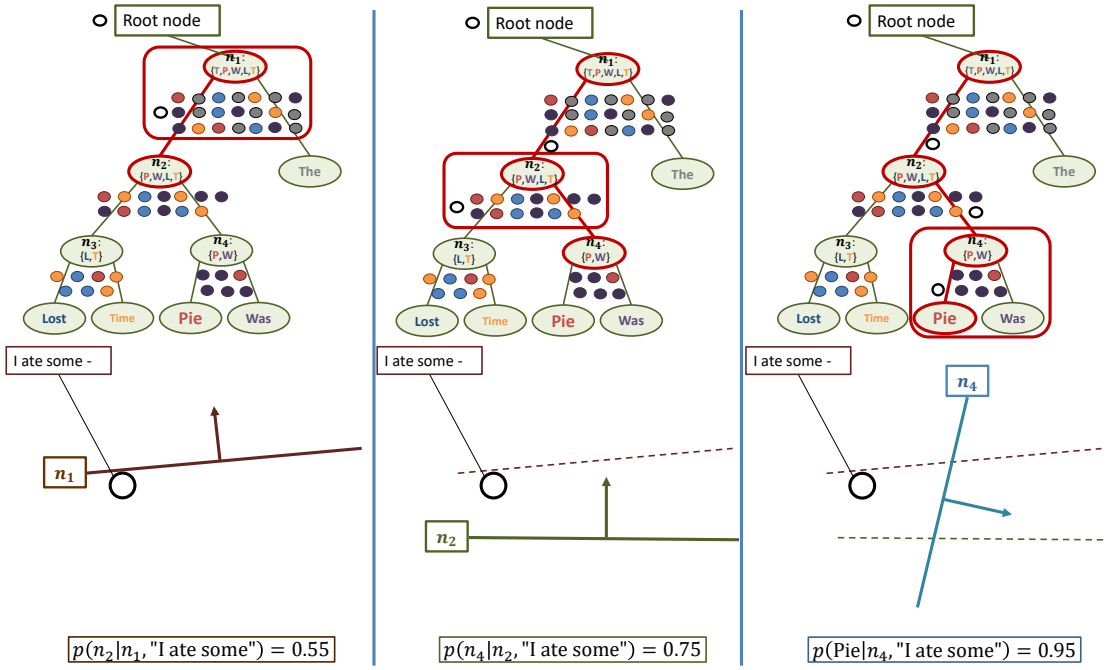

Figure 3.3: Computing the likelihood of the class pie for the embedding corresponding to the context *"I ate some... "*. The context is embedded to $\mathbf{r}_{Iatesome} \in \mathbb{R}^2$, and the path from the root to the label is $((n_1, 0), (n_2, 1), (n_4, 0))$. Following Equations 3.1 and 3.2, we then have: $p(\text{Pie}|\text{"I ate some"}) = 0.55 \times 0.75 \times 0.95 = 0.392$

vector $\mathbf{v}_n$ and bias b_n and corresponds to the output of a sigmoid:

$$p(d = 1|x, n) = \sigma(\mathbf{r}_x \cdot \mathbf{v}_n + b_n) \tag{3.1}$$

Where σ denotes the sigmoid function. Hence:

$$\begin{aligned}
\log p(w_t = i|x) &= \log p(((n_1^i, d_1^i), \ldots, (n_L^i, d_L^i))|x) \\
&= \log \prod_{l=1}^{L_i} p(d = d_l|x, n_l) \\
&= \sum_{l=1}^{L_i} \log p(d = d_l|x, n_l)
\end{aligned} \tag{3.2}$$

Since the depth of a balanced tree is logarithmic in the number of leaves (here $|\mathcal{V}|$), the cost of computing the likelihood of word w can then be reduced to $O(\log(|\mathcal{V}|) \times D)$. In their work, the authors start the training procedure by using a random tree, then alternate parameter learning with using a clustering-based heuristic to rebuild their hierarchy. We expand upon their method by providing an algorithm which uses hierarchies of arbitrary width, and jointly learns the tree structure and the model parameters by introducing a theoretically motivated node objective.

3.2 Background

We consider the general setting of multi-class prediction where the predictor has a hierarchical structure that allows for a very large number of labels both at train and test time. The predictive power of such models can heavily depend on the structure of the tree, and although past work showed how to learn the tree structure (*e.g.* [Madzarov et al., 2009, Bengio et al., 2010, Choromanska and Langford, 2015]), it expected that the feature vectors remained static. We provide a novel algorithm to simultaneously perform representation learning for the input data and learning of the hierarchical predictor. We introduce an objective function which favors balanced and easily-separable multi-way node partitions, and provide a novel algorithm for conditional density estimation which uses it to learn the tree structure along with the other model parameters. We theoretically analyze this objective, showing that it gives rise to a boosting property and a bound on classification error. We empirically validate the algorithm on text classification and language modeling, and show that it compares favorably to common baselines in terms of accuracy and running time.

Related Work The multi-class classification problem in the presence of a large number of classes has been addressed in the literature in a variety of ways. Hierarchical prediction has been a widely explored area. Some, such as [Breiman, 2001], have used random trees, but most more recent works have attempted to learn tree structures in a data dependent fashion. Beyond the rather inefficient enumerate-and-test approach (see *e.g.* [Breiman et al., 1984]) to find a good partition or expensive brute-force optimization [Agrawal et al., 2013], we give an overview of notable examples here. [Madzarov et al., 2009] build a binary tree by using hierarchical clustering in the feature space, then trains SVM classifiers to assign examples to children of a node. [Bengio et al., 2010] learn a tree structure by first learning a set of computationally cheaper one-against-all classifiers, and using their confusion matrix on the label set to learn splits in a top-down fashion via spectral clustering. [Deng et al., 2011] propose to extend on [Bengio et al., 2010] by initializing the tree structure at random and modifying the node partitions at training time once per epoch in a way which allows for overlaps between children, introducing a parameter to trade-off the resulting loss of efficiency with classification accuracy. Other works, such as the Structured OUtput Layer network of [Le et al., 2011] and more recently [Grave et al., 2017a], reduce the model's reliance on the hierarchical structure by relying on a flat prediction between the common classes, and only using hierarchical prediction for the less frequent ones.

Error Correcting Output Codes (ECOC) have been another promising direction, as they are less sensitive than hierarchical prediction to compounding of errors. [Hsu et al., 2009] propose to use sparse ECOC for image classification: since the complexity of the prediction (without the reconstruction step) only depends on the length of the codes, this can lead to gains in computational efficiency for large label sets. [Zhao and Xing, 2013] develop this idea further by learning codes in a data-dependent way, consider-

ing semantic similarity matrix between labels computed on the training data as well as additional constraints. [Beygelzimer et al., 2009b] and [Beygelzimer et al., 2009a] endeavor to bridge the gap between hierarchical prediction and probabilistic ECOC in terms of guarantees of robustness. [Beygelzimer et al., 2009b] show that using an m-elimination tournament rather than a simple tree for hierarchical prediction can have achieve the same kind of guarantees as an ECOC based model nearly as efficiency as tree-structured prediction. [Beygelzimer et al., 2009a] on the other hand propose to use wider trees where ECOC are used to choose between the children, and also provide a rudimentary tree learning method which uses an additional hyper-parameter to find a trade-off between building a balanced and accurate hierarchy.

Another family of works learns to partition the feature space into regions with a limited label space. FastXML [Prabhu and Varma, 2014] (and its slower and less accurate at prediction predecessor [Agrawal et al., 2013]) builds trees to do so by relying on a rank-sensitive loss function and shows an advantage over some other ranking and NLP-based techniques in the context of multi-label classification. Another approach is the SLEEC classifier [Bhatia et al., 2015] for extreme multi-label classification which learns sparse embeddings and employs a nearest neighbors approach to limit the set of candidate labels for a new example. Finally, [Weston et al., 2013] propose a method to make prediction time shorter for a pre-trained label scorer by first partitioning the input space with a weighted variant of k-means which takes advantage of information from the scoring function, and limiting the possible labels of an example in a specific partition to be in a restricted subset of labels.

A common shortcoming of all of the above mentioned methods, however, is that they either learn their partitions or hierarchies based solely on properties of the input space, or rely on an expensive regime of initial training with a standard model. The recently

proposed LOM tree algorithm of [Choromanska and Langford, 2015] differs significantly from the above mentioned works in this regard in that it addresses the problem of learning good-quality binary node partitions on-line using an objective which can be optimized through SGD ([Bottou, 1998]) and naturally balances the needs for efficiency and the compatibility of the hierarchical structure and prediction function. A follow-up to that work which was developed concurrently with the method introduced in this chapter is the Recall Tree of [Daumé III et al., 2017]. The method learns a binary tree to partition the input space into regions which are associated with limited sub-sets of the label space with high recall, and trains a prediction function on the sub-set of labels at each node. Note that both of these works learn a tree-structured partition of the input space, rather than an explicit hierarchy over the output space, which is the focus of our work.

Conditional density estimation can also be challenging in settings where the label space is large. The underlying problem here consists in learning a probability distribution over a set of random variables given some context. For example, in the language modeling setting one can learn the probability of a word given the previous text, either by making a Markov assumption and approximating the left context by the last few words seen (n-grams e.g. [Jelinek and Mercer, 1980, Katz, 1987], feed-forward neural language models [Bengio et al., 2003, Mikolov et al., 2011, Schwenk and Gauvain, 2002]), or by attempting to learn a low-dimensional representation of the full history (RNNs [Mikolov et al., 2010, Mirowski and Vlachos, 2015]). Both the recurrent and feed-forward Neural Network Language Models (NNLM) [Bengio et al., 2003] simultaneously learn a distributed representation for words and the probability function for word sequences, expressed in terms of these representations.

The major drawback of these models is that they can be slow to train, as they grow

linearly with the vocabulary size (anywhere between 10,000 and 1M words), which can make them difficult to apply [Mnih and Teh, 2012]. A number of methods have been proposed to overcome this difficulty. Works such as LBL [Mnih and Hinton, 2007] or Word2Vec [Mikolov et al., 2013] reduce the model to its barest bones, with only one hidden layer and no non-linearities. Another proposed approach has been to only compute the NNLM probabilities for a reduced vocabulary size, and use hybrid neural-n-gram model [Schwenk and Gauvain, 2005] at prediction time. Other avenues to reduce the cost of computing gradients for large vocabularies include using different sampling techniques to approximate it [Bengio and Senecal, 2003, Bengio and Senecal, 2008, Mnih and Teh, 2012], replacing the likelihood objective by a contrastive one [Weston et al., 2011, Gutmann and Hyvärinen, 2012] or spherical loss [de Brébisson and Vincent, 2015], relying on self-normalizing models [Andreas and Klein, 2015], or taking advantage of data sparsity [Vincent et al., 2015]. It is important to note however that while all of these methods drastically reduce training time, using the trained models for classification still requires computing a score for all classes, such that the test-time computational complexity would not be reduced.

Similarly to the classification case, there have also been a significant number of works that use tree structured models to accelerate computation of the likelihood and gradients [Morin and Bengio, 2005, Mnih and Hinton, 2008, Mikolov et al., 2013]. These use various heuristics to build a hierarchy, from using ontologies [Morin and Bengio, 2005] to Huffman coding [Mikolov et al., 2013]. One algorithm which endeavors to learn a binary tree structure along with the representation is presented in [Mnih and Hinton, 2008]. They iteratively learn word representations given a fixed tree structure, and use a criterion that trades off between making a balanced tree and clustering the words based on their current embedding. Their method is not truly on-line however,

as it needs to run a full epoch before updating the hierarchical structure, and relies on a heuristic with an additional hyper-parameter to ensure balanced trees. The language modeling application we present in the second part of this chapter is most closely related to the latter work, and uses a similar embedding of the context. However, where their setting is limited to binary trees, we work with arbitrary width, and provide a tree building objective which is both less computationally costly and comes with theoretical guarantees. We also implemented a simple version of their proposed paradigm, which performs significantly worse than our method.

Background In the next paragraphs, we define the classification and log-likelihood objectives we wish to maximize. Let $\mathcal{X}$ be an input space, and $\mathcal{V}$ a label space. Let $\mathcal{P}$ be a joint distribution over samples in $(\mathcal{X}, \mathcal{V})$, and let $f_\Theta : \mathcal{X} \to \mathbb{R}^{D_r}$ be a function mapping every input $x \in \mathcal{X}$ to a dimension D_r representation $\mathbf{r}_x \in \mathbb{R}^{D_r}$, and parameterized by Θ (e.g. as a neural network).

We consider two objectives. On the one hand, given a function g which takes an input representation $\mathbf{r}_x \in \mathbb{R}^{D_r}$, and predicts for it a label $g(\mathbf{r}) \in \mathcal{V}$, the classification objective is defined as the expected proportion of correctly classified examples:

$$\mathcal{O}^{\text{class}}(\Theta, g) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\mathbb{1}[g \circ f_\Theta(x) = y]\Big] \tag{3.3}$$

On the other hand, if a function $p_\theta(\cdot|\mathbf{r})$ defines a conditional probability distribution (parameterized by θ) over $\mathcal{V}$ for any $\mathbf{r} \in \mathbb{R}^{D_r}$, we can define the log-likelihood objective as the expected log-likelihood of samples from $(\mathcal{X}, \mathcal{V})$:

$$\mathcal{O}^{\text{ll}}(\Theta, \theta) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\log p_\theta(y|f_\Theta(x))\Big] \tag{3.4}$$

The second objective is slightly more general than the first. Indeed, if one can learn a probability distribution p_θ which maximizes the log-likelihood defined in Equation 3.4, then it is possible to obtain a good classification accuracy by replacing g in Equation 3.3 by the deterministic prediction function g^d corresponding to the MAP label:

$$g^d(f_\Theta(x)) = \arg\max_{y \in \mathcal{V}} p_\theta(y|f_\Theta(x)) \tag{3.5}$$

or by the stochastic prediction function g_s corresponding to sampling from the conditional distribution:

$$g^s(f_\Theta(x)) \sim p_\theta(y|f_\Theta(x)) \tag{3.6}$$

The inverse direction is not true however: if a function p_θ generally assigns a high conditional probability to the right label in most cases but predicts it to be close to zero in some cases, the g^d or g^s functions defined above can have a good classification accuracy while leading to a poor log-likelihood. In the rest of this chapter, we practically learn a probability distribution by optimizing a log-likelihood objective, and provide theoretical guarantees on the classification accuracy using the stochastic prediction function g^s.

Let us now show how to express the objectives in Equations 3.3 and 3.4 when using tree-structured prediction functions (with fixed structure) as illustrated in Figure 3.1. Consider a tree $\mathcal{T}$ of depth D and arity M with $K = |\mathcal{V}|$ leaf nodes and N internal nodes. Each leaf l corresponds to a label, and can be identified with the path $\mathbf{c}^l$ from the root to the leaf. In the rest of the paper, we will use the following notations:

$$\mathbf{c}^l = ((c^l_{1,1}, c^l_{1,2}), \ldots, (c^l_{d,1}, c^l_{d,2}), \ldots, (c^l_{D,1}, c^l_{D,2})), \tag{3.7}$$

where $c^l_{d,1} \in [1, N]$ correspond to the node index at depth d, and $c^l_{d,2} \in [1, M]$ indicates

which child of $c^l_{d,1}$ is next in the path. In that case, our classification and density estimation problems are reduced to choosing the right child of a node or defining a probability distribution over children given $x \in \mathcal{X}$ respectively.

We then need to replace g and p_θ with node decision functions $(g_n)_{n=1}^N$ and conditional probability distributions $(p_{\theta_n})_{n=1}^N$ respectively. Given such a tree and representation function, since we have a unique path from root to leaf l_y for each label y, we can write:

$$\mathbb{1}[g \circ f_\Theta(x) = y] = \prod_{d=1}^{D} \mathbb{1}[g_{c^{l_y}_{d,1}} \circ f_\Theta(x) = c^{l_y}_{d,2}]$$

and,

$$p_\theta(y|f_\Theta(x)) = \prod_{d=1^D} p_{\theta_{c^{l_y}_{d,1}}}(c^{l_y}_{d,2}|f_\Theta(x))$$

Hence our objective functions become:

$$\mathcal{O}^{\text{class}}(\Theta, g) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\prod_{d=1}^{D} \mathbb{1}[g_{c^{l_y}_{d,1}} \circ f_\Theta(x) = c^{l_y}_{d,2}]\Big] \tag{3.8}$$

$$\mathcal{O}^{\text{ll}}(\Theta, \theta) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\sum_{d=1}^{D} \log p_{\theta_{c^{l_y}_{d,1}}}(c^{l_y}_{d,2}|f_\Theta(x))\Big] \tag{3.9}$$

Similarly to the flat prediction setting, a model trained with the hierarchical log-likelihood objective of Equation 3.9 can lead to a low classification error (Equation 3.8) when using the stochastic or deterministic MAP prediction functions defined above (Equations 3.5 and 3.6 respectively). For a balanced tree, the cost of using the stochastic prediction is logarithmic in the size of the label space: we sample a child from the root, then a child of this child, etc... until we reach a leaf. However, finding the maximum likelihood leaf could in theory require computing a score for each label and be as expensive as a flat prediction. There are two ways to get around this difficulty. On the one hand, we could

perform a greedy beam search for the MAP label. This approach is guaranteed to be logarithmic in complexity, but may fail to find the true MAP if *e.g.* children of the root have similar scores, but more important decisions need to be taken deeper in the tree. On the other hand, we can use a depth-first search with a branch-and-bound type algorithm, since the log-likelihood of reaching a node is an upper bound on the log-likelihood of reaching any of its descendants. We choose the latter in our experiments. This approach is exact, but does not have guaranteed logarithmic complexity. However, we find in practice that the branch-and-bound method seldom has to visit more than a handful of leaves, leading to good running times (see Section 3.5).

The tree objective defined in Equation 3.9, and possibly 3.8 depending on the formulation of the prediction function, can be optimized in the space of parameters of the representation and node functions using standard gradient ascent methods. However, they also implicitly depend on the tree structure $\mathcal{T}$. In the rest of the paper, we provide a surrogate node objective function J_n which determines the structure of the tree in a way which naturally balances tree depth and prediction accuracy. In Section 3.3, we provide an algorithm which learns a hierarchical prediction function's parameters to optimize the data log-likelihood (Equation 3.9) under an adaptive tree structure motivated by J_n. In Section 3.4, we justify our choice of objective by showing how a high value for a specific node leads to a pure and balanced split, and how a lower bound on the J_n's for all nodes leads to an upper bound on the classification error presented in Equation 3.8 (when using the stochastic prediction function of Equation 3.5). Finally, in Section 3.5, we show experimentally that learning a model with our algorithm leads to a low classification error on a tag prediction dataset when using the deterministic MAP prediction function of Equation 3.5, and to a low perplexity (high log-likelihood, Equation 3.9) in a language modeling application.

3.3 Adaptive Tree Model and Learning Algorithm

In this Section, we introduce a per-node objective J_n which leads to good quality trees when maximized, and provide a tree learning algorithm inspired by it.

Objective function In an M-ary tree with K labels, we define the node objective J_n for a non-leaf node n as:

$$J_n = \frac{2}{M} \sum_{i=1}^{K} q_i^{(n)} \sum_{j=1}^{M} |p_j^{(n)} - p_{j|i}^{(n)}|, \tag{3.10}$$

where $q_i^{(n)}$ denotes the proportion of examples reaching node n that are of class i, $p_{j|i}^{(n)}$ is the probability that an example of class i reaching n will be sent to its j^{th} child, and $p_j^{(n)}$ is the probability that an example of any class reaching n will be sent to its j^{th} child. Note that we have:

$$\forall j \in [1, M], \;\; p_j^{(n)} = \sum_{i=1}^{K} q_i^{(n)} p_{j|i}^{(n)}. \tag{3.11}$$

The objective in Equation 3.10 reduces to the LOM tree objective in the case of $M = 2$ (illustrated in Figure 3.4).

At a high level, maximizing the objective encourages the conditional distribution for each class to be as different as possible from the global one; so the node decision function needs to be able to discriminate between examples of the different classes. The objective thus favors balanced and pure node splits. To wit, we call a split at node n *perfectly balanced* when the global distribution $p_\cdot^{(n)}$ is uniform, and *perfectly pure* when each $p_{\cdot|i}^{(n)}$ takes value either 0 or 1, as all data points from the same class reaching node n are sent to the same child. In Section 3.4, we discuss the theoretical properties of this objective in details. We show that maximizing it leads to perfectly balanced and perfectly pure splits. We also provide a boosting theorem which gives an upper bound on

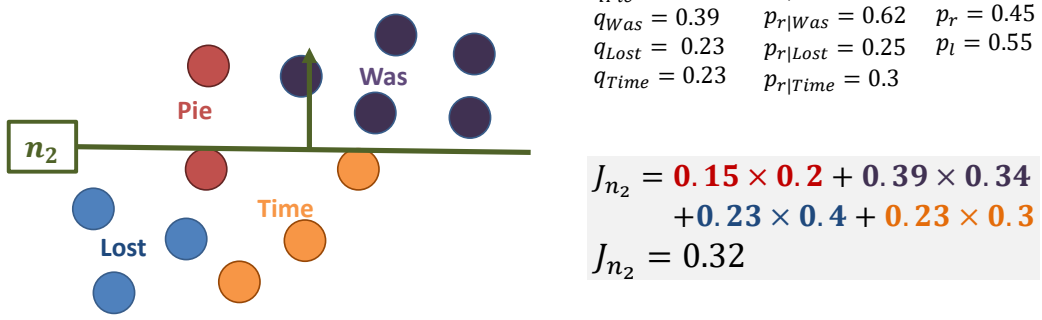

Figure 3.4: Computing the value of the J_n objective for a binary node split ($M = 2$ and $K = |\{Pie, Was, Lost, Time\}| = 4$ in Equation 3.10), here n_2 from Figures 3.2 and 3.3. We have:

$$
\begin{aligned}
J_{n_2} &= q_{Pie} \times (|p_{r|Pie} - p_r| + |p_{l|Pie} - p_l|) + q_{Was} \times (|p_{r|Was} - p_r| + |p_{l|Was} - p_l|) \\
&\quad + q_{Lost} \times (|p_{r|Lost} - p_r| + |p_{l|Lost} - p_l|) + q_{Time} \times (|p_{r|Time} - p_r| + |p_{l|Time} - p_l|) \\
&= 0.15 \times (|0.55 - 0.45| + |0.45 - 0.55|) + 0.39 \times (|0.62 - 0.45| + |0.38 - 0.55|) \\
&\quad + 0.23 \times (|0.25 - 0.45| + |0.75 - 0.55|) + 0.23 \times (|0.3 - 0.45| + |0.7 - 0.55|) \\
&= 0.15 \times 0.2 + 0.39 \times 0.35 + 0.23 \times 0.4 + 0.23 \times 0.3 = 0.32
\end{aligned}
$$

the classification error (Equation 3.8) of a tree where all nodes are "weakly" optimized in J_n (*i.e.*, all J_n are greater than some value).

Comparison to Previous Work While the form of the objective and theoretical guarantees presented in this chapter are somewhat similar to those provided in *i.e.* [Choromanska and Langford, 2015] or [Daumé III et al., 2017], the approach is actually quite different. Indeed, the above mentioned works learn a hierarchy over the *input space*. At each node n, for each example (x, y), a function g_n takes the example's input representation $\mathbf{r}_x$, and decides to deterministically send it to the child predicted by $g_n(\mathbf{r}_x) \in \{1, \ldots, M\}$. Then, when a leaf l is reached, the algorithm predicts among the classes which it has seen in the past: either the majority class in [Choromanska and

Langford, 2015], or using a prediction function g_l with values in the sub-set of labels $\mathcal{V}^l \subset \mathcal{V}$ which have reached the leaf in the past in [Daumé III et al., 2017]. Under this setting, the conditional probability $p^{(n)}_{j|i}$ of sending an example of class i to child j of node n corresponds the expected counts over the data distribution:

$$p^{(n)}_{j|i} = \mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\mathbb{1}[g_n(\mathbf{r}_x) = j]\Big]$$

and the classification objective (the probability that the leaf classifier that an example x is routed to predicts the right label) is defined as an empirical expectation over the data. So if x is sent to leaf l_x, we have:

$$\mathcal{O}^{\text{class}}(g) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\mathbb{1}[g_{l_x}(\mathbf{r}_x) = y]\Big]$$

In this chapter on the other hand, we consider trees over the *label space*, as described in section 3.2, where each label is identified with a unique leaf l and path from the root $\mathbf{c}^l$ (see Equation 3.7). The theoretical results consider the setting where each non-leaf node n of the tree has a conditional probability function $p^{(n)}_\theta$, and a new example x is classified by sampling a child $c \sim p^{(n)}_\theta(\mathbf{r}_x)$ at each node n from the root down until a leaf is reached. The corresponding label is then predicted. In this setting, the classification accuracy is then defined as an expectation over both the data distribution and the node choices:

$$\mathcal{O}^{\text{class}}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{P},(c_1,\ldots,c_D)\sim p_\theta(\mathbf{r}_x)}\Big[\prod_{d=1}^{D}\mathbb{1}[c_d = c^{l_y}_{d,2}]\Big]$$

and $p^{(n)}_{j|i}$ becomes the expected value of $p^{(n)}_\theta$ for examples of class i (as illustrated in Figure 3.5):

$$p^{(n)}_{j|i} \coloneqq \mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}[p^{(n)}_\theta(\mathbf{r}_x)] \tag{3.12}$$

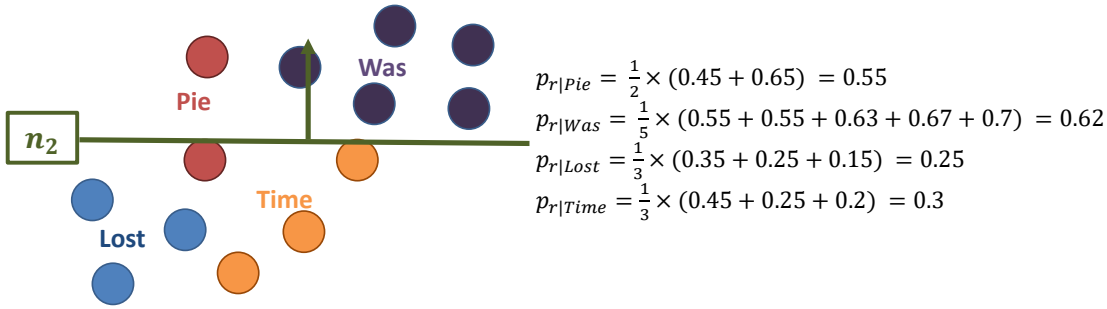

Figure 3.5: Computing the value of the $p_{j|i}^{(n)}$ as an empirical expectation (Equation 3.12. In the binary case, we only show j corresponding to the right child (r), and g_n is a sigmoid function).

There are two main advantages to using the latter formulation over the former. First, it provides a hierarchy over labels, which can be useful in and of itself: most notably, under our model, the conditional likelihood of a label can be estimated in logarithmic time by simply computing the likelihood of the path to the corresponding leaf, rather than all leaves which have ever seen that label. Secondly, and perhaps more importantly, it makes $p_{j|i}^{(n)}$, and thus $p_j^{(n)}$ and in turn J_n, differentiable in the model parameters. In the rest of this Section, we show how we can use the gradients of J_n to inform our tree learning algorithm.

Algorithm We now present an algorithm for simultaneously building a tree over the labels and learning the data representation. We aim at maximizing log-likelihood of the data as defined in 3.9, while ensuring that the tree learned also leads to a good classification accuracy (Equation 3.8) by increasing the value of the objective J_n of Equation 3.10 at each node of the tree (the boosting property presented in Theorem 3.1 shows the connection between the two).

Contrary to what could be expected, and even though we have pointed out that the

Algorithm 3.1 Tree Learning Algorithm

Input Input representation function: f with parameters Θ_f. Node decisions functions $(g_n)_{n=1}^K$ with parameters $(\Theta_n)_{n=1}^K$. Gradient step size ϵ.
Output Learned M-ary tree, parameters Θ_f and $(\Theta_n)_{n=1}^K$.
// SumProbas, Counts and the paths to the leaves $\mathbf{c}^l$ are treated as global variables

procedure InitializeNodeStats ()
  for $n = 1$ to N **do**
    for $i = 1$ to K **do**
      $\text{SumProbas}_{n,i} \leftarrow \frac{1}{M}$
      $\text{Counts}_{n,i} \leftarrow 1$

procedure NodeCompute ($\mathbf{r}$, n, i, target)
  $\mathbf{p} \leftarrow g_n(\mathbf{w})$
  // Keep track of statistics used to compute $\frac{\partial J_n}{\partial p^{(n)}_{\cdot|i}}$
  $\text{SumProbas}_{n,i} \leftarrow \text{SumProbas}_{n,i} + \mathbf{p}$
  $\text{Counts}_{n,i} \leftarrow \text{Counts}_{n,i} + 1$
  // Gradient step in the node parameters
  $\Theta_n \leftarrow \Theta_n + \epsilon \frac{\partial \log p_{\text{target}}}{\partial \Theta_n}$
  // Accumulate gradients in the example embedding
  return $\frac{\partial \log p_{\text{target}}}{\partial \mathbf{r}}$

InitializeNodeStats ()
for Each batch b **do**
  // AssignLabels () re-builds the tree based on the
  // current statistics
  AssignLabels ($\{1, \ldots, K\}$, root)
  for each example $(\mathbf{x}, i)$ in b **do**
    Compute input representation $\mathbf{r} = f(\mathbf{x})$
    $\Delta\mathbf{r} \leftarrow \mathbf{0}$
    // Go down the depth D path to the currently assigned leaf for label i
    for $d = 1$ to D **do**
      Set node id and target: $(n, j) \leftarrow c^i_d$
      $\Delta\mathbf{r} \leftarrow \Delta\mathbf{r}$ + NodeCompute (r, n, i, j)
    // Gradient step in the parameters of f
    $\Theta_f \leftarrow \Theta_f + \epsilon \frac{\partial f}{\partial \Theta_f} \Delta\mathbf{r}$

J_n are (mostly) differentiable in our setting, we do not take a direct gradient descent approach on them; given the value of these gradients, this would correspond to increasing the likelihood of sending members of a single class to more than one child (see Equation 3.13). Instead, we maintain a tree structure over the label set, and take gradient steps in the log-likelihood of our training data (as defined in Equation 3.9) in a batch-SGD

Algorithm 3.2 Label Assignment Algorithm

Input labels currently reaching the node
node ID n
Output Lists of labels now assigned to the node's children

procedure CheckFull (full, assigned, count, j)
  if $|\text{assigned}_j| \equiv 2 \mod (M-1)$ **then**
    count $\leftarrow$ count $-(M-1)$
  if count $= 0$ **then**
    full $\leftarrow$ full $\cup \{j\}$
  if count $= 1$ **then**
    count $\leftarrow 0$
    for j' s.t. $|\text{assigned}_{j'}| \equiv 1 \mod (M-1)$ **do**
      full $\leftarrow$ full $\cup \{j'\}$

procedure AssignLabels (labels, n)
  // first, compute $p_j^{(n)}$ *and* $p_{j|i}^{(n)}$.
  $\mathbf{p}_0^{avg} \leftarrow \mathbf{0}$
  count $\leftarrow 0$
  for i in labels **do**
    $\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg} + \text{SumProbas}_{n,i}$
    count $\leftarrow$ count $+ \text{Counts}_{n,i}$
    $\mathbf{p}_i^{avg} \leftarrow \text{SumProbas}_{n,i}/\text{Counts}_{n,i}$
  $\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg}/\text{count}$
  // then, assign each label to a child of n
  unassigned $\leftarrow$ labels
  full $\leftarrow \emptyset$
  count $\leftarrow (|\text{unassigned}| - (M-1))$
  for $j = 1$ to M **do**
    $\text{assigned}_j \leftarrow \emptyset$
  while unassigned $\neq \emptyset$ **do**
    // *$\frac{\partial J_n}{\partial p_{j|i}^{(n)}}$ is given in Equation 3.13*
    $(i^*, j^*) \leftarrow \underset{i \in \text{unassigned}, j \notin \text{full}}{\text{argmax}} \left(\frac{\partial J_n}{\partial p_{j|i}^{(n)}} \right)$
    if $n =$ root **then**
      $\mathbf{c}^{i^*} \leftarrow (n, j^*)$
    else
      $\mathbf{c}^{i^*} \leftarrow (\mathbf{c}^{i^*}, (n, j^*))$
    $\text{assigned}_{j^*} \leftarrow \text{assigned}_{j^*} \cup \{i^*\}$
    unassigned $\leftarrow$ unassigned $\setminus \{i^*\}$
    CheckFull (full, assigned, count, j^*)
  for $j = 1$ to M **do**
    AssignLabels (assigned_j, $\text{child}_{n,j}$, $d+1$)
  return assigned

like fashion, using the current tree for each example. The key part of the algorithm consists in updating this tree structure once per batch to ensure that the gradient step in

the log-likelihood also improves the value of J_n in each of the nodes visited as much as possible. This way, we obtain an algorithm which generalizes training of fixed structure hierarchical softmaxes while making use of the insights and theoretical gains of the J_n objectives.

The general idea is the following. The node objective is piece-wise linear in the conditional probability distributions $p^{(n)}_{j|i}$, and the gradient on the differentiable parts is (see proof of Lemma 3.1):

$$\frac{\partial J_n}{\partial \log p^{(n)}_{j|i}} = \frac{2}{M} q^{(n)}_i (1 - q^{(n)}_i) p^{(n)}_{j|i} \operatorname{sign}(p^{(n)}_{j|i} - p^{(n)}_j). \tag{3.13}$$

Then, according to Equation 3.13, increasing the log-likelihood of sending label i to any child j of n such that $p^{(n)}_{j|i} > p^{(n)}_j$ increases the objective J_n. Note that we only need to consider the labels i for which $q^{(n)}_i > 0$, that is, labels i which reach node n in the current tree. For practical reasons, we also want to make sure that we have a well-formed M-ary tree at each step, which means that the number of labels assigned to any node is always congruent to 1 modulo $(M-1)$ (this way, we always have the same number of non-leaf nodes, making implementation easier). Algorithm 3.2 provides such an assignment by greedily choosing the label-child pair (i, j) such that j still has room for labels with the highest value of $\frac{\partial J_n}{\partial p^{(n)}_{j|i}}$.

The global procedure, described in Algorithm 3.1, is then the following. At the start of each batch, re-assign targets for each node prediction function, starting from the root and going down the tree. At each node, each label is re-assigned to the child that most increases J_n, which trades off between keeping the node balance and sending the label to the child it has had most affinity with in the past (Algorithm 3.2). This can be seen as a form of hierarchical on-line clustering. Every example now has a unique path depending

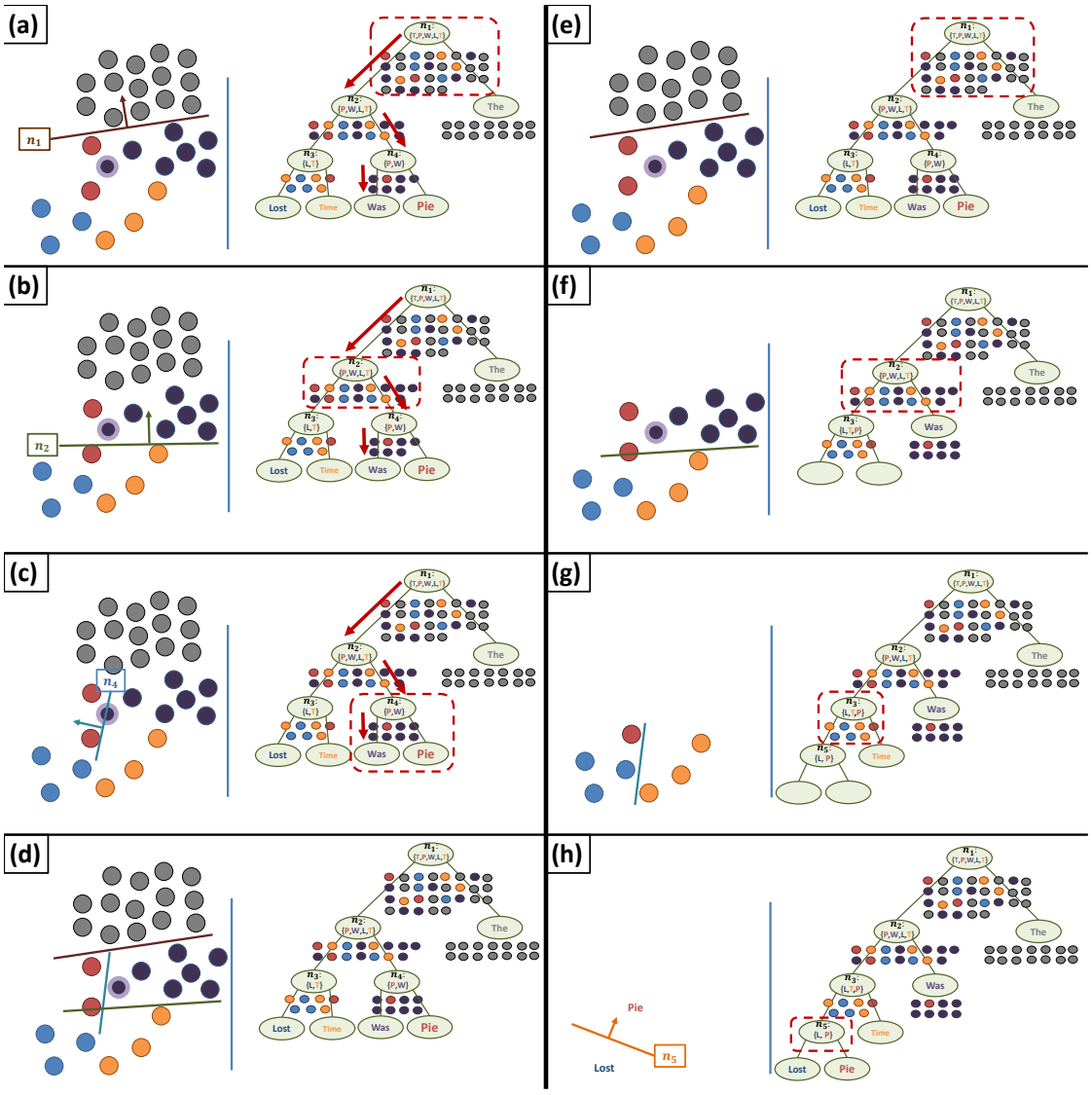

Figure 3.6: Tree learning algorithm: one iteration of Algorithm 3.1 (for a batch size of 1). **(a)-(c):** An example of class Was is sent down the path to the corresponding leaf, computing the gradients in the node likelihood functions on the ways. **(d):** Applying the gradients affects all of the embeddings and node functions on the path. **(e)-(g):** The tree is re-built from the root down. n_2 has become un-balanced and label Pie is re-routed form the right to the left child. **(h):** The new non-leaf node n_5 is initialized at random.

on its label. For each sample, we then take a gradient step in the log-likelihood of the newly assigned child for the label at each node along the assigned path (see Algorithm 3.1, illustrated in Figure 3.6).

Lemma 3.1. *If the gradients of the prediction functions are statistically independent across nodes, taking a stochastic gradient descent step in the log-likelihood under the tree defined in Algorithm 3.2 increases J_n in expectation.*

Proof of Lemma 3.1. Recall the form of the objective defined in 3.10:

$$\begin{aligned} J_n &= \frac{2}{M}\sum_{i=1}^{K} q_i^{(n)}\Big(\sum_{j=1}^{M}|p_j^{(n)} - p_{j|i}^{(n)}|\Big) \\ &= \frac{2}{M}\mathbb{E}_{i\sim q^{(n)}}\Big[f_n^J(i, p_{\cdot|\cdot}^{(n)}, q^{(n)})\Big] \end{aligned}$$

Where:

$$\begin{aligned} f_n^J(i, p_{\cdot|\cdot}^{(n)}, q^{(n)}) &= \sum_{j=1}^{M}\Big|p_j^{(n)} - p_{j|i}^{(n)}\Big| = \sum_{j=1}^{M}\Big|p_{j|i}^{(n)} - \sum_{i'=1}^{K} q_{i'}^{(n)} p_{j|i'}^{(n)}\Big| \\ &= \sum_{j=1}^{M}\Big|\sum_{i'=1}^{K}(\mathbb{1}_{i=i'} - q_{i'}^{(n)})p_{j|i'}^{(n)}\Big| \end{aligned}$$

Hence, on parts of the $p_{j|i}^{(n)}$ space where the absolute value function in the above equation is differentiable, we have:

$$\frac{\partial f_n^J(i, p_{\cdot|\cdot}^{(n)}, q^{(n)})}{\partial p_{j|i}^{(n)}} = (1 - q_i^{(n)})\,\mathrm{sign}(p_{j|i}^{(n)} - p_j^{(n)})$$

Which leads to:

$$\frac{\partial J_n}{\partial p_{j|i}^{(n)}} = \frac{2}{M} q_i^{(n)}(1 - q_i^{(n)})\,\mathrm{sign}(p_{j|i}^{(n)} - p_j^{(n)}) \tag{3.14}$$

And:

$$\frac{\partial p_{j|i}^{(n)}}{\partial \theta} = \mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}[\frac{\partial}{\partial\theta} p_\theta^{(n)}(j|\mathbf{r}_x)]$$

Hence by applying the chain rule and noting that $q_i^{(n)} = \mathbb{E}_{(x,y)\sim\mathcal{P}}[\mathbb{1}[y=i]]$, we have:

$$\begin{aligned}
\frac{\partial J_n}{\partial \theta} &= \frac{\partial J_n}{\partial p^{(n)}_{\cdot|\cdot}} \frac{\partial p^{(n)}_{\cdot|\cdot}}{\partial \theta} \\
&= \sum_{i,j} \frac{2}{M} q_i^{(n)}(1-q_i^{(n)}) \operatorname{sign}(p^{(n)}_{j|i} - p_j^{(n)}) \mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}[\frac{\partial}{\partial\theta} p_\theta^{(n)}(j|\mathbf{r}_x)] \\
&= \frac{2}{M} \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[(1-q_i^{(n)}) \operatorname{sign}(p^{(n)}_{j|i} - p_j^{(n)}) \frac{\partial}{\partial\theta} p_\theta^{(n)}(j|\mathbf{r}_x)\Big] \\
&= \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[a_\theta^{(n)}(i, \mathbf{r}_x)\Big]
\end{aligned}$$

Where $a_\theta^{(n)}(i, \mathbf{r}_x)$ is the gradient step taken on example (x, i) when doing SGD on the J_n objective.

Meanwhile, recall the form of the log-likelihood objective described in Equation 3.9:

$$\mathcal{O}^{\text{ll}}(\Theta, \theta) = \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[\sum_{d=1}^{D} \log p_{\theta_{c^{l_i}_{d,1}}}(c^{l_i}_{d,2}|f_\Theta(x))\Big]$$

Then consider $\mathbb{1}[n \in c^i]$ the indicator function that a node n is on the path from the root to the leaf l_i corresponding to class i, and denote $j_i^{(n)}$ the child of n corresponding to class i if $n \in c^i$, we have:

$$\begin{aligned}
\frac{\partial \mathcal{O}^{\text{ll}}(\Theta, \theta)}{\partial \theta} &= \frac{\partial}{\partial\theta} \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[\sum_{d=1}^{D} \log p_{\theta_{c^{l_i}_{d,1}}}(c^{l_i}_{d,2}|f_\Theta(x))\Big] \\
&= \frac{\partial}{\partial\theta} \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[\sum_{n} \mathbb{1}[n \in c^i] \log p_{\theta_{c^{l_i}_{d,1}}}(c^{l_i}_{d,2}|f_\Theta(x))\Big] \\
&= \sum_{n} \mathbb{1}[n \in c^i] \frac{\partial}{\partial\theta} \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\log p_{\theta_{c^{l_i}_{d,1}}}(c^{l_i}_{d,2}|f_\Theta(x))\Big] \\
&= \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[\sum_{n\in c^i} \frac{\partial}{\partial\theta} \log p_\theta^{(n)}(j_i^{(n)}|\mathbf{r}_x)\Big] \\
&= \mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[b_\theta(i, \mathbf{r}_x)\Big]
\end{aligned}$$

Here $b_\theta(i, \mathbf{r}_x)$ is the gradient step taken on example (x, i) when doing SGD on the O^{ll} objective.

For any node n, we can the compute the dot product between the gradient step for the log-likelihood and J_n objective:

$$\begin{aligned}
\mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[a_\theta^{(n)}(i, \mathbf{r}_x) \cdot b_\theta(i, \mathbf{r}_x)\Big] &= \mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\Big(\frac{2}{M}(1 - q_i^{(n)})\,\text{sign}(p_{j|i}^{(n)} - p_j^{(n)})\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\Big) \\
&\qquad \cdot \Big(\sum_{n'\in c^i}\frac{\partial}{\partial\theta}\log p_\theta^{(n')}(j^{(n')}(i)|\mathbf{r}_x)\Big)\Big] \\
&= \sum_i q_i^{(n)}\mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\Big(\frac{2}{M}(1 - q_i^{(n)})\,\text{sign}(p_{j|i}^{(n)} - p_j^{(n)})\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\Big) \\
&\qquad \cdot \Big(\sum_{n'\in c^i}\frac{\partial}{\partial\theta}\log p_\theta^{(n')}(j^{(n')}(i)|\mathbf{r}_x)\Big)\Big] \\
&= \sum_{i,j}\frac{2}{M}q_i^{(n)}(1 - q_i^{(n)})\,\text{sign}(p_{j|i}^{(n)} - p_j^{(n)})\mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\Big(\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\Big) \\
&\qquad \cdot \Big(\sum_{n'\in c^i}\frac{\partial}{\partial\theta}\log p_\theta^{(n')}(j^{(n')}(i)|\mathbf{r}_x)\Big)\Big]
\end{aligned}$$

Let us assume that the gradients in different node functions are independent, that is:

$$\forall n \neq n', \quad Cov(\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x), \frac{\partial}{\partial\theta}\log p_\theta^{(n')}(j^{(n')}(i)|\mathbf{r}_x)) = 0$$

Where Cov is the covariance. Then, we would have:

$$\begin{aligned}\mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[a_\theta^{(n)}(i,\mathbf{r}_x)\cdot b_\theta(i,\mathbf{r}_x)\Big] &= \sum_{i,j}\frac{2}{M}q_i^{(n)}(1-q_i^{(n)})\,\mathrm{sign}(p_{j|i}^{(n)}-p_j^{(n)})\mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\\ &\qquad\qquad\cdot\frac{\partial}{\partial\theta}\log p_\theta^{(n)}(j_i^{(n)}|\mathbf{r}_x)\Big]\\ &= \sum_{i,j}\frac{\partial J_n}{\partial p_{j|i}^{(n)}}\mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\cdot\frac{\partial}{\partial\theta}\log p_\theta^{(n)}(j_i^{(n)}|\mathbf{r}_x)\Big]\\ &= \sum_{i}\mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\sum_j\frac{\partial J_n}{\partial p_{j|i}^{(n)}}\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\cdot\frac{\partial}{\partial\theta}\log p_\theta^{(n)}(j_i^{(n)}|\mathbf{r}_x)\Big]\end{aligned}$$

Let us note:

$$\sigma^i_{j,j^*}=\mathbb{E}_{(x,y)\sim\mathcal{P}|y=i}\Big[\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)\cdot\frac{\partial}{\partial\theta}\log p_\theta^{(n)}(j^*|\mathbf{r}_x)\Big]$$

Then the expected dot product of the log-likelihood gradient step and of the J_n gradient becomes:

$$\begin{aligned}\mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[a_\theta^{(n)}(i,\mathbf{r}_x)\cdot b_\theta(i,\mathbf{r}_x)\Big] &= \sum_i\Big(\sigma^i_{j_i^{(n)},j_i^{(n)}}\frac{\partial J_n}{\partial p^{(n)}_{j_i^{(n)}|i}}+\sum_{j\neq j_i^{(n)}}\sigma^i_{j,j_i^{(n)}}\frac{\partial J_n}{\partial p^{(n)}_{j|i}}\Big)\\ &= \sum_i\sigma^i_{j_i^{(n)},j_i^{(n)}}\left(\frac{\partial J_n}{\partial p^{(n)}_{j_i^{(n)}|i}}-\sum_{j\neq j_i^{(n)}}\Big(-\frac{\sigma^i_{j,j_i^{(n)}}}{\sigma^i_{j_i^{(n)},j_i^{(n)}}}\Big)\frac{\partial J_n}{\partial p^{(n)}_{j|i}}\right)\end{aligned}\tag{3.15}$$

Since $p_\theta^{(n)}(\cdot|\mathbf{r}_x)$ is a probability distribution,we know that

$$\sum_j\frac{\partial}{\partial\theta}p_\theta^{(n)}(j|\mathbf{r}_x)=\frac{\partial\sum_j p_\theta^{(n)}(j|\mathbf{r}_x)}{\partial\theta}=\frac{\partial 1}{\partial\theta}=0$$

Hence, since the expectation is linear:

$$\forall i, \forall j^*, \sum_j \sigma^i_{j,j^*} = 0$$

And:

$$\forall i, \forall j^*, \quad \sum_{j \neq j^*} \Big(- \frac{\sigma^i_{j,j^*}}{\sigma^i_{j^*,j^*}} \Big) = 1 \tag{3.16}$$

Similarly, since the prediction function is a softmax, we can show that:

$$\forall j^*, \sigma^i_{j^*,j^*} \geq 0 \quad \text{and} \quad \forall j \neq j^*, \sigma^i_{j,j^*} \leq \sigma^i_{j,j} \tag{3.17}$$

Indeed, if we write the scores in the softmax as $o^j_\theta(\mathbf{r}_x)$, we get:

$$\begin{aligned}
\frac{\partial \log p^{(n)}_\theta(j|\mathbf{r}_x)}{\partial \theta} &= \frac{\partial}{\partial \theta} \Big(o^j_\theta(\mathbf{r}_x) - \log \sum_{j'} \exp o^{j'}_\theta(\mathbf{r}_x) \Big) \\
&= \frac{\partial o^j_\theta(\mathbf{r}_x)}{\partial \theta} - \frac{1}{\sum_{j''} \exp o^{j''}_\theta(\mathbf{r}_x)} \sum_{j'} \frac{\partial \exp o^{j'}_\theta(\mathbf{r}_x)}{\partial \theta} \\
&= \frac{\partial o^j_\theta(\mathbf{r}_x)}{\partial \theta} - \sum_{j'} p^{(n)}_\theta(j'|\mathbf{r}_x) \frac{\partial o^{j'}_\theta(\mathbf{r}_x)}{\partial \theta}
\end{aligned}$$

And:

$$\frac{\partial p^{(n)}_\theta(j|\mathbf{r}_x)}{\partial \theta} = p^{(n)}_\theta(j|\mathbf{r}_x) \frac{\partial \log p^{(n)}_\theta(j|\mathbf{r}_x)}{\partial \theta}$$

Let us write $e_\theta(\mathbf{r}_x) = \sum_{j'} p^{(n)}_\theta(j'|\mathbf{r}_x) \frac{\partial o^{j'}_\theta(\mathbf{r}_x)}{\partial \theta}$, then we have:

$$\frac{\partial p^{(n)}_\theta(j|\mathbf{r}_x)}{\partial \theta} \cdot \frac{\partial \log p^{(n)}_\theta(j|\mathbf{r}_x)}{\partial \theta} = p^{(n)}_\theta(j|\mathbf{r}_x) || \frac{\partial o^j_\theta(\mathbf{r}_x)}{\partial \theta} - e_\theta(\mathbf{r}_x) ||^2_2$$

And:

$$\frac{\partial p_\theta^{(n)}(j|\mathbf{r}_x)}{\partial \theta} \cdot \frac{\partial \log p_\theta^{(n)}(j'|\mathbf{r}_x)}{\partial \theta} = p_\theta^{(n)}(j|\mathbf{r}_x)\Big(\frac{\partial o_\theta^{j}(\mathbf{r}_x)}{\partial \theta} - e_\theta(\mathbf{r}_x)\Big) \cdot \Big(\frac{\partial o_\theta^{j'}(\mathbf{r}_x)}{\partial \theta} - e_\theta(\mathbf{r}_x)\Big)$$
$$\leq \frac{\partial p_\theta^{(n)}(j|\mathbf{r}_x)}{\partial \theta} \cdot \frac{\partial \log p_\theta^{(n)}(j|\mathbf{r}_x)}{\partial \theta}$$

Hence, by applying Equations 3.16 and 3.17 to Equation 3.15, and considering that Algorithm 3.2 assigns a class i to the child $j_i^{(n)} = \arg\max_j \frac{\partial J_n}{\partial p_{j|i}^{(n)}}$, we get the lower bound:

$$\mathbb{E}_{(x,i)\sim\mathcal{P}}\Big[a_\theta^{(n)}(i,\mathbf{r}_x)\cdot b_\theta(i,\mathbf{r}_x)\Big] = \sum_i \sigma^i_{j_i^{(n)},j_i^{(n)}}\left(\frac{\partial J_n}{\partial p^{(n)}_{j_i^{(n)}|i}} - \sum_{j\neq j_i^{(n)}}\Big(-\frac{\sigma^i_{j,j_i^{(n)}}}{\sigma^i_{j_i^{(n)},j_i^{(n)}}}\Big)\frac{\partial J_n}{\partial p^{(n)}_{j|i}}\right)$$
$$\geq \sum_i \sigma^i_{j_i^{(n)},j_i^{(n)}}\Big(\frac{\partial J_n}{\partial p^{(n)}_{j_i^{(n)}|i}} - \max_{j\neq j_i^{(n)}}\frac{\partial J_n}{\partial p^{(n)}_{j|i}}\Big)$$
$$\geq 0$$

□

An interesting feature of the algorithm is that since the representation of examples from different classes are learned together, there is intuitively less of a risk of getting stuck in a specific tree configuration. More specifically, if two similar classes are initially assigned to different children of a node, the algorithm is less likely to keep this initial decision since the representations for examples of both classes will be pulled together in other nodes. In the next Section, we provide a theoretical analysis of the objective introduced in Equation 3.10.

3.4 Theoretical Properties of the Objective

We start by showing that maximizing J_n in every node of the tree leads to high-quality nodes, i.e. perfectly balanced and perfectly pure node splits. Let us first introduce some formal definitions.

Definition 3.1 (Balancedness factor)**.** *The split in node n of the tree is $\beta^{(n)}$-balanced if*

$$\beta^{(n)} \leq \min_{j=\{1,2,\ldots,M\}} p_j^{(n)},$$

where $\beta^{(n)} \in (0, \frac{1}{M}]$ is a balancedness factor.

A split is perfectly balanced if and only if $\beta^{(n)} = \frac{1}{M}$.

Definition 3.2 (Purity factor)**.** *The split in node n of the tree is $\alpha^{(n)}$-pure if*

$$\frac{1}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} q_i^{(n)} \min\left(p_{j|i}^{(n)}, 1 - p_{j|i}^{(n)}\right) \leq \alpha^{(n)},$$

where $\alpha^{(n)} \in [0, \frac{1}{M})$ is a purity factor.

A split is perfectly pure if and only if $\alpha^{(n)} = 0$. Figure 3.7 illustrates the computation of both $\alpha^{(n)}$ and $\beta^{(n)}$ for a specific binary node split.

The following lemmas characterize the range of the objective J_n and link it to the notions of balancedness and purity of the split.

Lemma 3.2. *The objective function J_n lies in the interval $[0, J^*]$, with $J^* = \frac{4}{M}\left(1 - \frac{1}{M}\right)$.*

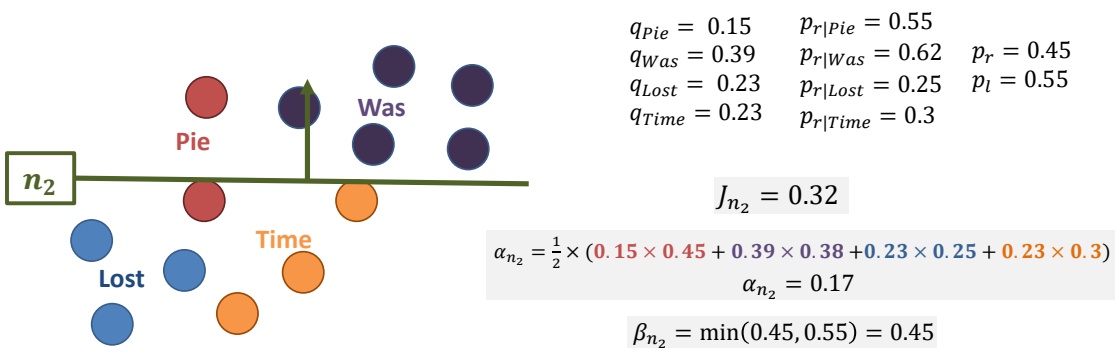

Figure 3.7: Computing the value of the $\alpha^{(n)}$ and $\beta^{(n)}$ for a binary node split. The node is not quite pure, so $\alpha^{(n)}$ is somewhat larger than 0, but mostly balanced, so that $\beta^{(n)}$ is close to $\frac{1}{2}$.

Proof of Lemma 3.2. Recall that we have $p_j^{(n)} = \sum_{l=1}^{K} q_l^{(n)} p_{j|l}^{(n)}$, hence:

$$J_n = \frac{2}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} q_i^{(n)} |p_j^{(n)} - p_{j|i}^{(n)}| = \frac{2}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} q_i^{(n)} \left| \sum_{l=1}^{K} q_l^{(n)} p_{j|l}^{(n)} - p_{j|i}^{(n)} \right|$$

The objective J_n is a convex function of the $p_{j|l}^{(n)}$ which are defined over a simplex, hence its extrema are reached on corners. Thus, define the following two sets:

$$O_j = \{i : i \in \{1, 2, \ldots, K\}, p_{j|i}^{(n)} = 1\} \text{ and } Z_j = \{i : i \in \{1, 2, \ldots, K\}, p_{j|i}^{(n)} = 0\}. \tag{3.18}$$

We omit the n index for ease of reading. Note that $\sum_{j=1}^{M} p_j^{(n)} = 1$ and

$p_j^{(n)} = \sum_{l=1}^{K} q_l^{(n)} p_{j|l}^{(n)} = \sum_{i \in O_j} q_i^{(n)}$, thus $\sum_{j=1}^{M} \sum_{i \in O_j} q_i^{(n)} = 1$, and we have:

$$\begin{aligned} J_n &\leq \frac{2}{M} \sum_{j=1}^{M} \left[\sum_{i \in O_j} q_i^{(n)} \left(1 - \sum_{l \in O_j} q_l^{(n)} \right) + \sum_{i \in Z_j} q_i^{(n)} \sum_{l \in O_j} q_l^{(n)} \right] \\ &= \frac{4}{M} \sum_{j=1}^{M} \left[\sum_{i \in O_j} q_i^{(n)} - \left(\sum_{i \in O_j} q_i^{(n)} \right)^2 \right] \\ &= \frac{4}{M} \left[1 - \sum_{j=1}^{M} \left(\sum_{i \in O_j} q_i^{(n)} \right)^2 \right] \end{aligned}$$

Applying Jensen's inequality to the last line yields:

$$\begin{aligned} J_n &\leq \frac{4}{M} - 4 \left[\sum_{j=1}^{M} \left(\frac{1}{M} \sum_{i \in O_j} q_i^{(n)} \right) \right]^2 \\ &= \frac{4}{M} \left(1 - \frac{1}{M} \right) \end{aligned}$$

□

Lemma 3.3. *The objective function J_n admits the highest value, i.e. $J_n = J^*$, if and only if the split in node n is perfectly balanced, i.e. $\beta^{(n)} = \frac{1}{M}$, and perfectly pure, i.e. $\alpha^{(n)} = 0$.*

Proof of Lemma 3.3. We start by proving that if the split in node n is perfectly balanced and perfectly pure, then $J_n = J^*$. For a maximally balanced split, we have:

$$J_n = \frac{2}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} q_i^{(n)} \left| \frac{1}{M} - p_{j|i}^{(n)} \right|.$$

Since the split is maximally pure, each $p_{j|i}^{(n)}$ can only take value 0 or 1. Thus, with sets

O_j and Z_j defined as in the proof of Lemma 3.2 (Equation 3.18), we have:

$$\begin{aligned} J_n &= \frac{2}{M}\sum_{j=1}^{M}\left[\sum_{i\in O_j} q_i^{(n)}\left(1-\frac{1}{M}\right)+\sum_{i\in Z_j} q_i^{(n)}\frac{1}{M}\right] \\ &= \frac{2}{M}\sum_{j=1}^{M}\left[\sum_{i\in O_j} q_i^{(n)}\left(1-\frac{1}{M}\right)+\frac{1}{M}\left(1-\sum_{i\in O_j} q_i^{(n)}\right)\right] \\ &= \frac{2}{M}\left(1-\frac{2}{M}\right)\sum_{j=1}^{M}\sum_{i\in O_j} q_i^{(n)}+\frac{2}{M} \\ &= \frac{4}{M}\left(1-\frac{1}{M}\right) \end{aligned}$$

This proves the first implication. Next we prove that if $J_n = J^*$, then the split in node n is perfectly balanced and perfectly pure. The latter simply follows from the fact that, as we noted earlier, J_n is a convex function of the $p_{j|l}^{(n)}$ which are defined over a simplex. We give a proof by contradiction that the optimum is also perfectly balanced.

Let us assume that at least for one value of j, $p_j^{(n)} \neq \frac{1}{M}$, or in other words if we decompose each $p_j^{(n)}$ as $p_j^{(n)} = \frac{1}{M} + x_j$, then at least for one value of j, $x_j \neq 0$. Using the above defined sets O_j and Z_j, and recalling that $p_j^{(n)} = \sum_{l=1}^{K} q_l^{(n)} p_{j|l}^{(n)} = \sum_{i\in O_j} q_i^{(n)}$,

we have:

$$
\begin{aligned}
J_n &= \frac{2}{M}\sum_{j=1}^{M}\left[\sum_{i\in O_j} q_i^{(n)}(1-p_j^{(n)}) + \sum_{i\in Z_j} q_i^{(n)} p_j^{(n)}\right] \\
&= \frac{2}{M}\sum_{j=1}^{M}\left[p_j^{(n)}(1-p_j^{(n)}) + p_j^{(n)}(1-p_j^{(n)})\right] \\
&= \frac{4}{M}\sum_{j=1}^{M}\left[p_j^{(n)} - (p_j^{(n)})^2\right] \\
&= \frac{4}{M}\left[1 - \sum_{j=1}^{M}(p_j^{(n)})^2\right] \\
&= \frac{4}{M}\left[1 - \sum_{j=1}^{M}\left(\frac{1}{M} + x_j\right)^2\right] \\
&= \frac{4}{M}\left(1 - \frac{1}{M} - \frac{2}{M}\sum_{j=1}^{M} x_j - \sum_{j=1}^{M} x_j^2\right) \\
&< \frac{4}{M}\left(1 - \frac{1}{M}\right) = J^*
\end{aligned}
$$

□

We next propose Lemmas 3.4 and 3.5 which analyze balancedness and purity of a node split in isolation, i.e. we analyze resp. balancedness and purity of a node split when resp. purity and balancedness is fixed and perfect. We show that in such isolated setting increasing J_n leads to a more balanced and more pure split.

Lemma 3.4. *If a split in node n is perfectly pure, then*

$$
\beta^{(n)} \in \left[\frac{1}{M} - \frac{\sqrt{M(J^* - J_n)}}{2}, \frac{1}{M}\right].
$$

Proof of Lemma 3.4. Since we assume that the split is perfectly pure, then each $p_{j|i}^{(n)}$ is either 0 or 1. Thus, using sets O_j and Z_j from Equation 3.18, we have:

$$J_n = \frac{2}{M} \sum_{j=1}^{M} \left[\sum_{i \in O_j} q_i^{(n)} (1 - p_j) + \sum_{i \in Z_j} q_i^{(n)} p_j \right]$$

Note that $p_j = \sum_{i \in O_j} q_i^{(n)}$, hence:

$$J_n = \frac{2}{M} \sum_{j=1}^{M} \left[p_j (1 - p_j) + (1 - p_j) p_j \right] = \frac{4}{M} \sum_{j=1}^{M} p_j (1 - p_j) = \frac{4}{M} \left(1 - \sum_{j=1}^{M} p_j^2 \right)$$

Thus:

$$\sum_{j=1}^{M} p_j^2 = 1 - \frac{M J_n}{4}. \tag{3.19}$$

Let us express p_j as $p_j = \frac{1}{M} + \epsilon_j$, where $\epsilon_j \in [-\frac{1}{M}, 1 - \frac{1}{M}]$. Then:

$$\sum_{j=1}^{M} p_j^2 = \sum_{j=1}^{M} \left(\frac{1}{M} + \epsilon_j \right)^2 = \frac{1}{M} + \frac{2}{M} \sum_{j=1}^{M} \epsilon_j + \sum_{j=1}^{M} \epsilon_j^2 = \frac{1}{M} + \sum_{j=1}^{M} \epsilon_j^2, \tag{3.20}$$

since $\frac{2}{M} \sum_{j=1}^{M} \epsilon_j = 0$. Thus combining Equation 3.19 and 3.20

$$\frac{1}{M} + \sum_{j=1}^{M} \epsilon_j^2 = 1 - \frac{M J_n}{4}$$

and thus

$$\sum_{j=1}^{M} \epsilon_j^2 = 1 - \frac{1}{M} - \frac{M J_n}{4}.$$

The last statement implies that

$$\max_{j=1,2,\ldots,M} \epsilon_j \leq \sqrt{1 - \frac{1}{M} - \frac{M J_n}{4}},$$

which is equivalent to

$$\min_{j=1,2,\ldots,M} p_j = \frac{1}{M} - \max_j \epsilon_j \geq \frac{1}{M} - \sqrt{1 - \frac{1}{M} - \frac{MJ_n}{4}} = \frac{1}{M} - \frac{\sqrt{M(J^* - J_n)}}{2}.$$

□

Lemma 3.5. *If a split in node n is perfectly balanced, then $\alpha^{(n)} \leq (J^* - J_n)/2$.*

Proof of Lemma 3.5. Since the split is perfectly balanced, we have:

$$J_n = \frac{2}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} q_i^{(n)} \left| \frac{1}{M} - p_{j|i}^{(n)} \right| = \frac{2}{M} \sum_{i=1}^{K} \sum_{j=1}^{M} q_i^{(n)} \left| \frac{1}{M} - p_{j|i}^{(n)} \right|$$

Define two sets:

$$\mathcal{A}_i = \{j : j \in \{1, 2, \ldots, K\}, p_{j|i}^{(n)} < \frac{1}{M}\} \text{ and } \mathcal{B}_i = \{j : j \in \{1, 2, \ldots, K\}, p_{j|i}^{(n)} \geq \frac{1}{M}\}.$$

Then:

$$\begin{aligned}
J_n &= \frac{2}{M} \sum_{i=1}^{K} \left[\sum_{j \in \mathcal{A}_i} q_i^{(n)} \left(\frac{1}{M} - p_{j|i}^{(n)} \right) + \sum_{j \in \mathcal{B}_i} q_i^{(n)} \left(p_{j|i}^{(n)} - \frac{1}{M} \right) \right] \\
&= \frac{2}{M} \sum_{i=1}^{K} q_i^{(n)} \left[\sum_{j \in \mathcal{A}_i} \left(\frac{1}{M} - p_{j|i}^{(n)} \right) + \sum_{j \in \mathcal{B}_i} \left(p_{j|i}^{(n)} - \frac{1}{M} \right) \right] \\
&= \frac{2}{M} \sum_{i=1}^{K} q_i^{(n)} \left[\sum_{j \in \mathcal{A}_i} \left(\frac{1}{M} - p_{j|i}^{(n)} \right) + \sum_{j \in \mathcal{B}_i} \left((1 - \frac{1}{M}) - (1 - p_{j|i}^{(n)}) \right) \right]
\end{aligned}$$

Recall that the optimal value of J_n is:

$$\begin{aligned} J^* &= \frac{4}{M}\left(1-\frac{1}{M}\right) \\ &= \frac{2}{M}\sum_{i=1}^{N} q_i^{(n)}\left[(M-1)\frac{1}{M}+\left(1-\frac{1}{M}\right)\right] \\ &= \frac{2}{M}\sum_{i=1}^{N} q_i^{(n)}\left[\left(\sum_{j\in\mathcal{A}_i\cup\mathcal{B}_i}\frac{1}{M}\right)-\frac{1}{M}+\left(1-\frac{1}{M}\right)\right] \end{aligned}$$

Note that $\mathcal{A}_i$ can have at most $M-1$ elements. Furthermore, $\forall j \in \mathcal{A}_i, p_{j|i}^{(n)} < 1-p_{j|i}^{(n)}$. Then, we have:

$$J^*-J^n = \frac{2}{M}\sum_{i=1}^{K} q_i^{(n)}\left[\sum_{j\in\mathcal{A}_i} p_{j|i}^{(n)}+\sum_{j\in\mathcal{B}_i}\left((1-p_{j|i}^{(n)})+\frac{1}{M}-(1-\frac{1}{M})\right)-\frac{1}{M}+\left(1-\frac{1}{M}\right)\right]$$

Hence, since $\mathcal{B}_i$ has at least one element:

$$\begin{aligned} J^*-J^n &\geq \frac{2}{M}\sum_{i=1}^{K} q_i^{(n)}\left[\sum_{j\in\mathcal{A}_i} p_{j|i}^{(n)}+\sum_{j\in\mathcal{B}_i}\left(1-p_{j|i}^{(n)}\right)\right] \\ &\geq \frac{2}{M}\sum_{i=1}^{K} q_i^{(n)}\left[\sum_{j=1}^{M}\min(p_{j|i}^{(n)},1-p_{j|i}^{(n)})\right] \\ &\geq 2\alpha \end{aligned}$$

□

Error bound Next we provide a bound on the classification error for a tree based on the value of the J_n. In particular, we show that if the objective is "weakly" optimized in each node of the tree, where this weak advantage is captured in a form of the *Weak Hypothesis Assumption*, then our algorithm will amplify this weak advantage to build a tree achieving any desired level of accuracy. Denote $y(x)$ to be a fixed target function

with domain $\mathcal{X}$, which assigns the data point x to its label, and let $\mathcal{P}$ be a fixed target distribution over $\mathcal{X}$. Together y and $\mathcal{P}$ induce a distribution on labeled pairs $(x, y(x))$. Let $t(x)$ be the label assigned to data point x by the tree. We denote as $\epsilon(\mathcal{T})$ the error of tree $\mathcal{T}$, i.e. $\epsilon(\mathcal{T}) \coloneqq \mathbb{E}_{x\sim\mathcal{P}}\Big[\sum_{i=1}^{K} \mathbb{1}[t(x) = i, y(x) \neq i]\Big]$ (thus $1 - \epsilon(\mathcal{T})$ refers to the accuracy as given by Equation 3.8).

To bound the error, we start by proving a theorem relating the number of weakly optimized non-leaf nodes in a tree to the entropy of the predictions G^e, then relate low entropy to low classification error under the learned tree. Let $\mathcal{L}$ denote the set of leaves of the tree. Let w_l denote the probability that an example of any class reaches leaf l, and $q_i^{(l)}$ be as defined previously, the entropy is defined as:

$$G^e = \sum_{l\in\mathcal{L}} w_l \sum_{i=1}^{K} q_i^{(l)} \ln\left(\frac{1}{q_i^{(l)}}\right)$$

Then, the following theorem holds:

Theorem 3.1. *The Weak Hypothesis Assumption says that for any distribution $\mathcal{P}$ over the data, at each node n of the tree $\mathcal{T}$ there exists a node prediction function $p_\theta^{(n)}$ such that $J_n \geq \gamma$, where $\gamma \in \left[\frac{M}{2} \min_{j=1,2,\ldots,M} p_j, 1 - \frac{M}{2} \min_{j=1,2,\ldots,M} p_j\right]$. Under the Weak Hypothesis Assumption, for any $\kappa \in [0, 1]$, a tree with N non-leaf nodes has entropy $G^e \leq \kappa$ provided N meets the requirement:*

$$N \geq \left(\frac{1}{\kappa}\right)^{\frac{16[M(1-2\gamma)+2\gamma](M-1)}{M^2\gamma^2 \log_2 e} \ln K}$$

Proof of Theorem 3.1. Let the weight of the tree leaf be defined as the probability that a randomly chosen data point x drawn from some fixed target distribution $\mathcal{P}$ reaches this leaf. Suppose at time step t, n is the heaviest leaf and has weight w. Consider

splitting this leaf to M children $n_1, n_2, \ldots, n_M$. Let the weight of the j^{th} child be denoted as w_j. Also for the ease of notation let p_j refer to $p_j^{(n)}$ (recall that $\sum_{j=1}^{m} p_j = 1$) and $p_{j|i}$ refer to $p_{j|i}^{(n)}$, and furthermore let q_i be the shorthand for $q_i^{(n)}$. Recall that $p_j = \sum_{i=1}^{K} q_i p_{j|i}$ and $\sum_{i=1}^{K} q_i = 1$. Notice that for any $j = \{1, 2, \ldots, M\}$, $w_j = w p_j$. Let $\boldsymbol{q}$ be the k-element vector with i^{th} entry equal to q_i.

Recall the expression for the entropy of tree leaves:

$$G^e = \sum_{l \in \mathcal{L}} w_l \sum_{i=1}^{K} q_i^{(l)} \ln \left(\frac{1}{q_i^{(l)}} \right)$$

where $\mathcal{L}$ is a set of all tree leaves. Define the following function: $\tilde{G}^e(\boldsymbol{q}) = \sum_{i=1}^{K} q_i \ln \left(\frac{1}{q_i} \right)$. Before the split the contribution of node n to G^e was equal to $w\tilde{G}^e(\boldsymbol{q})$. Note that for any $j = \{1, 2, \ldots, M\}$, $q_i^{(n_j)} = \frac{q_i p_{j|i}}{p_j}$ is the probability that a randomly chosen x drawn from $\mathcal{P}$ has label i given that x reaches node n_j. For brevity, let $q_i^{n^j}$ be denoted as $q_{j,i}$. Let $\boldsymbol{q}_j$ be the k-element vector with i^{th} entry equal to $q_{j,i}$. Notice that $\boldsymbol{q} = \sum_{j=1}^{M} p_j \boldsymbol{q}_j$. After the split the contribution of the same, now internal, node n changes to $w \sum_{j=1}^{M} p_j \tilde{G}^e(\boldsymbol{q}_j)$. We denote the difference between the contribution of node n to the value of the entropy-based objectives in times t and $t+1$ as

$$\Delta_t^e := G_t^e - G_{t+1}^e = w \left[\tilde{G}^e(\boldsymbol{q}) - \sum_{j=1}^{M} p_j \tilde{G}^e(\boldsymbol{q}_j) \right]. \tag{3.21}$$

The entropy function $\tilde{G}^e$ is strongly concave with respect to l_1-norm with modulus 1, thus we extend the inequality given by Equation 7 in [Choromanska et al., 2016] by

applying Theorem 5.2. from [Azocar et al., 2011] and obtain the following bound

$$\begin{aligned}
\Delta_t^e &= w\left[\tilde{G}^e(\boldsymbol{q}) - \sum_{j=1}^{M} p_j \tilde{G}^e(\boldsymbol{q}_j)\right] \\
&\geq w\frac{1}{2}\sum_{j=1}^{M} p_j \|q_j - \sum_{l=1}^{M} p_l q_l\|_1^2 \\
&= w\frac{1}{2}\sum_{j=1}^{M} p_j \left(\sum_{i=1}^{K}\left|\frac{q_i p_{j|i}}{p_j} - \sum_{l=1}^{M} p_l \frac{q_i p_{l|i}}{p_l}\right|\right)^2 \\
&= w\frac{1}{2}\sum_{j=1}^{M} p_j \left(\sum_{i=1}^{K} q_i \left|\frac{p_{j|i}}{p_j} - \sum_{l=1}^{M} p_{l|i}\right|\right)^2 \\
&= w\frac{1}{2}\sum_{j=1}^{M} p_j \left(\sum_{i=1}^{K} q_i \left|\frac{p_{j|i}}{p_j} - 1\right|\right)^2 \\
&= w\frac{1}{2}\sum_{j=1}^{M} \frac{1}{p_j} \left(\sum_{i=1}^{K} q_i \left|p_{j|i} - p_j\right|\right)^2 .
\end{aligned}$$

Before proceeding, we will bound each p_j. Note that by the *Weak Hypothesis Assumption* we have

$$\gamma \in \left[\frac{M}{2}\min_{j=1,2,\ldots,M} p_j, 1 - \frac{M}{2}\min_{j=1,2,\ldots,M} p_j\right],$$

thus

$$\min_{j=1,2,\ldots,M} p_j \geq \frac{2\gamma}{M},$$

hence all p_js are such that $p_j \geq \frac{2\gamma}{M}$. Then:

$$\max_{j=1,2,\ldots,M} p_j \leq 1 - \frac{2\gamma}{M}(M-1) = \frac{M(1-2\gamma)+2\gamma}{M}.$$

Thus all p_js are such that $p_j \leq \frac{M(1-2\gamma)+2\gamma}{M}$.

$$\begin{aligned}
\Delta_t^e &\geq w\frac{M^2}{2[(M(1-2\gamma)+2\gamma]}\sum_{j=1}^{M}\frac{1}{M}\left(\sum_{i=1}^{K}q_i\left|p_{j|i}-p_j\right|\right)^2 \\
&\geq w\frac{M^2}{2[(M(1-2\gamma)+2\gamma]}\left(\sum_{j=1}^{M}\frac{1}{M}\sum_{i=1}^{K}q_i\left|p_{j|i}-p_j\right|\right)^2 \\
&= w\frac{M^2}{8[(M(1-2\gamma)+2\gamma]}\left(\frac{2}{M}\sum_{j=1}^{M}\sum_{i=1}^{K}q_i\left|p_{j|i}-p_j\right|\right)^2 \\
&= \frac{M^2}{[(M(1-2\gamma)+2\gamma]}\frac{wJ_n^2}{8},
\end{aligned}$$

where the last inequality is a consequence of Jensen's inequality. w can further be lower-bounded by noticing the following:

$$\begin{aligned}
G_t^e &= \sum_{l\in\mathcal{L}} w_l \sum_{i=1}^{K} q_i^{(l)} \ln\left(\frac{1}{q_i^{(l)}}\right) \\
&\leq \sum_{l\in\mathcal{L}} w_l \ln K \\
&\leq w\ln K \sum_{l\in\mathcal{L}} 1 \\
&= [t(M-1)+1]w\ln K \leq (t+1)(M-1)w\ln K,
\end{aligned}$$

where the first inequality results from the fact that uniform distribution maximizes the entropy. This gives the lower-bound on Δ_t^e of the following form:

$$\Delta_t^e \geq \frac{M^2 G_t^e J_n^2}{8(t+1)[M(1-2\gamma)+2\gamma](M-1)\ln K},$$

and by using *Weak Hypothesis Assumption* we get

$$\Delta_t^e \geq \frac{M^2 G_t^e \gamma^2}{8(t+1)[M(1-2\gamma)+2\gamma](M-1)\ln K}$$

Following the recursion of the proof in Section 3.2 in [Choromanska et al., 2016] (note that in our case $G_1^e \leq 2(M-1)\ln K$), we obtain that under the *Weak Hypothesis Assumption*, for any $\kappa \in [0, 2(M-1)\ln K]$, to obtain $G_t^e \leq \kappa$ it suffices to make

$$t \geq \left(\frac{2(M-1)\ln K}{\kappa}\right)^{\frac{16[M(1-2\gamma)+2\gamma](M-1)\ln K}{M^2 \log_2 e \gamma^2}}$$

splits. □

The above theorem shows the number of splits that suffice to reduce the entropy of the predictions of the tree below an arbitrary threshold κ. As shown in the proof of the above theorem, the *Weak Hypothesis Assumption* implies that all p_js satisfy: $p_j \in [\frac{2\gamma}{M}, \frac{M(1-2\gamma)+2\gamma}{M}]$. Below we show a tighter version of this bound when assuming that each node induces balanced split.

Corollary 3.1. *The Weak Hypothesis Assumption says that for any distribution $\mathcal{P}$ over the data, at each node n of the tree $\mathcal{T}$ there exists a partition such that $J_n \geq \gamma$, where $\gamma \in \mathbb{R}^+$. Under the Weak Hypothesis Assumption and when all nodes make perfectly balanced splits, for any $\kappa \in [0, 1]$, to obtain $G^e \leq \kappa$ it suffices to have a tree with*

$$N \geq \left(\frac{1}{\kappa}\right)^{\frac{16(M-1)}{M^2\gamma^2 \log_2 e} \ln K} \qquad \textit{non-leaf nodes}.$$

Proof of Corollary 3.1. Note that the lower-bound on Δ_t^e from the previous prove could be made tighter as follows:

$$\begin{aligned}
\Delta_t^e &\geq w\frac{1}{2}\sum_{j=1}^{M}\frac{1}{p_j}\left(\sum_{i=1}^{K} q_i \left|p_{j|i} - p_j\right|\right)^2 \\
&= w\frac{M^2}{2}\sum_{j=1}^{M}\frac{1}{M}\left(\sum_{i=1}^{K} q_i \left|p_{j|i} - p_j\right|\right)^2 \\
&\geq w\frac{M^2}{2}\left(\sum_{j=1}^{M}\frac{1}{M}\sum_{i=1}^{K} q_i \left|p_{j|i} - p_j\right|\right)^2 \\
&= w\frac{M^2}{8}\left(\frac{2}{M}\sum_{j=1}^{M}\sum_{i=1}^{K} q_i \left|p_{j|i} - p_j\right|\right)^2 \\
&= \frac{M^2 w J_n^2}{8},
\end{aligned}$$

where the first inequality was taken from the proof of Theorem 3.1 and the following equality follows from the fact that each node is balanced. By next following exactly the same steps as shown in the proof of Theorem 3.1 we obtain the corollary. □

Finally, the following Corollary relates both of the above results to the multi-class error rate and the classification objective from Equation 3.8:

Corollary 3.2. *For the tree structure defined in Algorithm 3.1 and under the* Weak Hypothesis Assumption *stating that in each node we have $J_n \geq \gamma$, the classification error ϵ is upper bounded by:*

$$\epsilon \leq \left(\frac{M-1}{K}\right)^{\frac{M^2 \log_2 e}{16[M(1-2\gamma)+2\gamma]\log K}\gamma^2}$$

Further, if all nodes are perfectly balanced, we can write:

$$\epsilon \leq \Big(\frac{M-1}{K}\Big)^{\frac{M^2 \log_2 e}{16(M-1)\log K}\gamma^2}$$

Proof. We next proceed to directly proving the error bound. Denote $w(l)$ to be the probability that a data point x reached leaf l. Recall that $q_i^{(l)}$ is the probability that the data point x corresponds to label i given that x reached l, i.e. $q_i^{(l)} = P(y(x) = i|x \text{ reached } l)$. Algorithm 3.2 assigns labels to leaves by order of descending $q_i(1 - q_i)p_{j|i}$. Notice that if n_j id the j-th child of n, we have $q_i^{(n_j)} = q_i^{(n)} p_{j|i}^{(n)}$, and that $\frac{\partial x(1-x)}{\partial x}$ is greater than 0 for $x < \frac{1}{2}$, this means that a leaf is assigned to label i if and only if the following is true $\forall_{\substack{z=\{1,2,\ldots,k\} \\ z \neq i}} q_i^{(l)} \geq q_z^{(l)}$. Therefore we can write that

$$\begin{aligned}
\epsilon(\mathcal{T}) &= \sum_{i=1}^{K} P(t(x) = i, y(x) \neq i) && (3.22)\\
&= \sum_{l \in \mathcal{L}} w(l) \sum_{i=1}^{K} P(t(x) = i, y(x) \neq i | x \text{ reached } l) \\
&= \sum_{l \in \mathcal{L}} w(l) \sum_{i=1}^{K} P(y(x) \neq i | t(x) = i, x \text{ reached } l) P(t(x) = i | x \text{ reached } l) \\
&= \sum_{l \in \mathcal{L}} w(l)(1 - \max(q_1^{(l)}, q_2^{(l)}, \ldots, q_K^{(l)})) \sum_{i=1}^{K} P(t(x) = i | x \text{ reached } l) \\
&= \sum_{l \in \mathcal{L}} w(l)(1 - \max(q_1^{(l)}, q_2^{(l)}, \ldots, q_K^{(l)})) && (3.23)
\end{aligned}$$

Consider again the Shannon entropy $G(\mathcal{T})$ of the leaves of tree $\mathcal{T}$ that is defined as

$$G^e(\mathcal{T}) = \sum_{l \in \mathcal{L}} w(l) \sum_{i=1}^{K} q_i^{(l)} \log_2 \frac{1}{q_i^{(l)}}. \quad (3.24)$$

Let $i_l = \arg\max_{i=\{1,2,\ldots,K\}} q_i^{(l)}$. Note that

$$\begin{aligned} G^e(\mathcal{T}) &= \sum_{l\in\mathcal{L}} w(l) \sum_{i=1}^{K} q_i^{(l)} \log_2 \frac{1}{q_i^{(l)}} \\ &\geq \sum_{l\in\mathcal{L}} w(l) \sum_{\substack{i=1 \\ i\neq i_l}}^{K} q_i^{(l)} \log_2 \frac{1}{q_i^{(l)}} \\ &\geq \sum_{l\in\mathcal{L}} w(l) \sum_{\substack{i=1 \\ i\neq i_l}}^{K} q_i^{(l)} \\ &= \sum_{l\in\mathcal{L}} w(l)(1 - \max(q_1^{(l)}, q_2^{(l)}, \ldots, q_K^{(l)})) \\ &= \epsilon(\mathcal{T}), \end{aligned} \tag{3.25}$$

where the last inequality comes from the fact that $\forall_{\substack{i=\{1,2,\ldots,K\} \\ i\neq i_l}} q_i^{(l)} \leq 0.5$ and thus $\forall_{\substack{i=\{1,2,\ldots,K\} \\ i\neq i_l}} \frac{1}{q_i^{(l)}} \in [2; +\infty]$ and consequently $\forall_{\substack{i=\{1,2,\ldots,K\} \\ i\neq i_l}} \log_2 \frac{1}{q_i^{(l)}} \in [1; +\infty]$. We next use the proof of Theorem 6 in [Choromanska et al., 2016]. The proof modifies only slightly for our purposes and thus we only list these modifications below.

- Since we define the Shannon entropy through logarithm with base 2 instead of the natural logarithm, the right hand side of inequality (2.6) in [Shalev-Shwartz, 2012] should have an additional multiplicative factor equal to $\frac{1}{\ln 2}$ and thus the right-hand side of the inequality stated in Lemma 14 has to have the same multiplicative factor.

- For the same reason as above, the right-hand side of the inequality in Lemma 9 should take logarithm with base 2 of k instead of the natural logarithm of k.

Using the entropy to bound the classification error and noticing that an M-ary tree needs at least $\frac{K}{M-1}$ internal nodes to have K leaves leads to the statement of Corollary 3.2. □

3.5 Experiments

Data Description We run experiments to evaluate both the classification and density estimation version of our algorithm. For classification, we use the YFCC100M dataset [Thomee et al., 2016], which consists of a set of a hundred million Flickr pictures along with captions and tag sets split into 91M training, 930K validation and 543K test examples. We focus here on the problem of predicting a picture's tags given its caption. For density estimation, we learn a log-bilinear language model on the Gutenberg novels corpus [Stroube, 2003], and compare the perplexity to that obtained with other flat and hierarchical losses.

Classification We follow the setting of [Grave et al., 2017b] for the YFCC100M tag prediction task: we only keep the tags which appear at least a hundred times, which leaves us with a label space of size 312K. We compare our results to those obtained with the FastText software [Grave et al., 2017b], which uses a binary hierarchical softmax objective based on Huffman coding (Huffman trees are designed to minimize the expected depth of their leaves weighed by frequencies and have been shown to work well with word embedding systems [Mikolov et al., 2013]), and to the Tagspace system [Weston et al., 2014], which uses a sampling-based margin loss (this allows for training in tractable time, but does not help at test time, hence the long times reported). We also extend the FastText software to use Huffman trees of arbitrary width. All models use a bag-of-word embedding representation of the caption text; the parameters of the input representation function f_Θ which we learn are the word embeddings $U_w \in \mathbb{R}^d$ and a caption representation is obtained by summing the embeddings of its words. We experimented with embeddings of dimension $d = 50$ and $d = 200$. We predict one tag for each caption, and report the precision as well as the training and test times in Table 3.1.

d	Model	Arity	P@1	Train	Test
50	TagSpace[1]	-	30.1	3h8	6h
	FastText[2]	2	27.2	**8m**	**1m**
	M-ary Huffman Tree	5	28.3	**8m**	**1m**
		20	29.9	10m	3m
	Learned Tree	5	31.6	18m	**1m**
		20	**32.1**	30m	3m
200	TagSpace[1]	-	35.6	5h32	15h
	FastText[2]	2	35.2	**12m**	**1m**
	M-ary Huffman Tree	5	35.8	13m	2m
		20	36.4	18m	3m
	Learned Tree	5	36.1	35m	3m
		20	**36.6**	45m	8m

Table 3.1: Classification performance on the YFCC100M dataset. [1] [Weston et al., 2014]. [2] [Grave et al., 2017b]. M-ary Huffman Tree modifies FastText by adding an M-ary hierarchical softmax objective.

Our implementation is based on the FastText open source version[1], to which we added M-ary Huffman and learned tree objectives. Table 3.1 reports the best accuracy we obtained with a hyper-parameter search using this version on our system so as to provide the most meaningful comparison, even though the accuracy is less than that reported in [Grave et al., 2017b]. We learned our models with SGD with a linearly decreasing rate for five epochs. We run a hyper-parameter search on the learning rate (in $\{0.01, 0.02, 0.05, 0.1, 0.25, 0.5\}$). In the learned tree settings, the learning rate stays constant for the first half of training, during which the AssignLabels() routine is called 50 times. We run the experiments in a Hogwild data-parallel setting using 12 threads on an Intel Xeon E5-2690v4 2.6GHz CPU. At prediction time, we follow the FastText setting in performing a truncated depth first search to find the most likely label (using the same idea as in a branch-and-bound algorithm: if a node score is less than that of the best current label, then all of its descendants are out). This could in theory be somewhat

[1] https://github.com/facebookresearch/fastText

more expensive than simply following the maximum likelihood child path from the root to a label, but in practice the children likelihood are different enough that there is little additional cost.

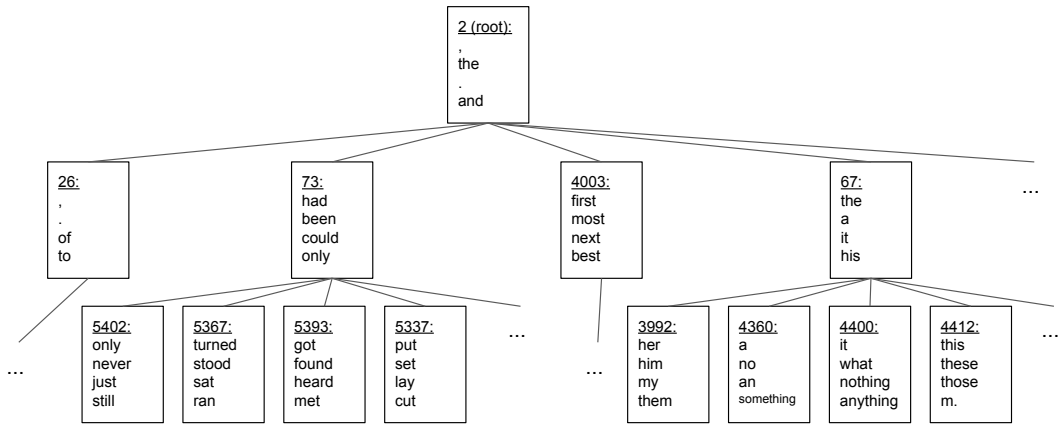

Figure 3.8: Tree learned from the Gutenberg corpus, showing the four most common words assigned to each node.

We gain a few different insights from Table 3.1. First, although wider trees are theoretically slower (remember that the theoretical complexity is $O(M \log_M(N))$ for an M-ary tree with N labels), they run incomparable time in practice and always perform better. Using our algorithm to learn the structure of the tree also always leads to more accurate models, with a gain of up to 3.3 precision points in the smaller 5-ary setting. Further, both the importance of having wider trees and learning the structure seems to be less when the node prediction functions become more expressive. At a high level, one could imagine that in that setting, the model can learn to use different dimensions of the input representation for different nodes, which would minimize the negative impact of having to learn a representation which is suited to more nodes.

Algorithm 3.3 Label Assignment Algorithm under Depth Constraint

Input Node statistics, max depth D
Paths from root to labels: $\mathcal{P} = (\mathbf{c}^i)_{i=1}^K$
node ID n and depth d
List of labels currently reaching n
Output Updated paths
Lists of labels now assigned to each of n's children under depth constraints

procedure AssignLabels (labels, n, d)
// first, compute $p_j^{(n)}$ *and* $p_{j|i}^{(n)}$.
// $\odot$ *is the element-wise*
// multiplication
$\mathbf{p}_0^{avg} \leftarrow \mathbf{0}$
count $\leftarrow 0$
for i in labels **do**
$\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg} + \text{SumProbas}_{n,i}$
count $\leftarrow$ count $+ \text{Counts}_{n,i}$
$\mathbf{p}_i^{avg} \leftarrow \text{SumProbas}_{n,i}/\text{Counts}_{n,i}$
$\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg}/\text{count}$
// then, assign each label to a child
// of n *under depth constraints*
unassigned $\leftarrow$ labels
full $\leftarrow \emptyset$
for $j = 1$ to M **do**
assigned$_j \leftarrow \emptyset$
while unassigned $\neq \emptyset$ **do**
// $\frac{\partial J_n}{\partial p_{j|i}^{(n)}}$ *is given in Equation 3.13*
$(i^*, j^*) \leftarrow \underset{i \in \text{unassigned}, j \notin \text{full}}{\text{argmax}} \left(\frac{\partial J_n}{\partial p_{j|i}^{(n)}} \right)$
$\mathbf{c}_d^{i^*} \leftarrow (n, j^*)$
assigned$_{j^*} \leftarrow$ assigned$_{j^*} \cup \{i^*\}$
unassigned $\leftarrow$ unassigned $\setminus \{i^*\}$
if $|\text{assigned}_{j^*}| = M^{D-d}$ **then**
full $\leftarrow$ full $\cup \{j^*\}$
for $j = 1$ to M **do**
AssignLabels (assigned$_j$, child$_{n,j}$, $d+1$)
return assigned

Another thing to notice is that since prediction time only depends on the expected depth of a label, our models which learned balanced trees are nearly as fast as Huffman coding which is optimal in that respect (except for the dimension 200, 20-ary tree, but the tree structure had not stabilized yet in that setting). Given all of the above remarks, our algorithm especially shines in settings where computational complexity and prediction time are highly constrained at test time, such as mobile devices or embedded systems.

Language Modeling We also ran language modeling experiments on the Gutenberg novel corpus [Stroube, 2003], which has about 50M tokens and a vocabulary of 250,000 words.

One notable difference from the previous task is that the language modeling setting can drastically benefit from the use of GPU computing, which can make using

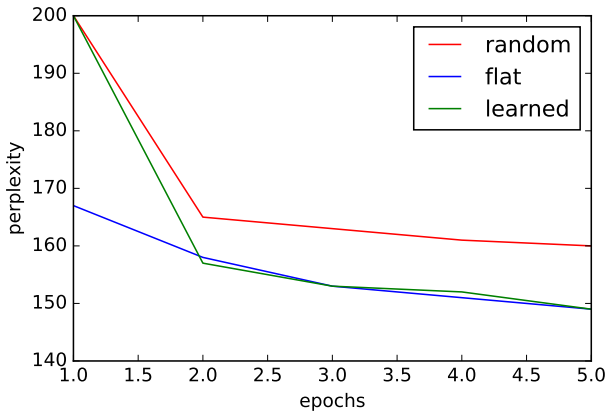

Figure 3.9: Test perplexity per epoch.

a flat softmax tractable (if not fast). While our algorithm requires more flexibility and thus does not benefit as much from the use of GPUs, a small modification of Algorithm 3.2 (described in the Algorithm 3.3) allows it to run under a maximum depth constraint and remain competitive. The results presented in this section are obtained using this modified version, which learns 65-ary trees of depth 3. In our experiments, we use a context window size of 4. We optimize the objectives with Adagrad, run a hyper-parameter search on the batch size (in $\{32, 64, 128\}$) and learning rate (in $\{0.01, 0.02, 0.05, 0.1, 0.25, 0.5\}$). The hidden representation dimension is 200. In the learned tree settings, the AssignLabels() routine is called 50 times per epoch. We used a 12GB NVIDIA GeForce GTX TITAN GPU.

Table 3.3 presents perplexity results for different loss functions, along with the time spent on computing and learning the objective (softmax parameters for the flat version, hierarchical softmax node parameters for the fixed tree, and hierarchical softmax structure and parameters for our algorithm). The learned tree model is nearly three and seven times as fast at train and test time respectively as the flat objective without losing any points of perplexity.

Leaf 229	Leaf 230	Leaf 300	Leaf 231
suggested	vegas	payments	operates
watched	&	buy-outs	includes
created	calif.	swings	intends
violated	park	gains	makes
introduced	n.j.	taxes	means
discovered	conn.	operations	helps
carried	pa.	profits	seeks
described	pa.	penalties	reduces
accepted	ii	relations	continues
listed	d.	liabilities	fails
...	...	...	...

Table 3.2: Example of labels reaching the deepest internal nodes in the final tree learned on Gutenberg. We can identify a leaf for 3rd person verbs, one for past participates, one for plural nouns, and one (loosely) for places.

Model	perp.	train ms/batch	test ms/batch
Clustering Tree	212	**2.0**	**1.0**
Random Tree	160	**1.9**	**0.9**
Flat softmax	149	*12.5*	*6.9*
Learned Tree	**148**	*4.5*	**0.9**

Table 3.3: Comparison of a flat softmax to a 65-ary hierarchical softmax (learned, random and heuristic-based tree).

Huffman coding does not apply to trees where all of the leaves are at the same depth. Instead, we use the following heuristic as a baseline, inspired by [Mnih and Hinton, 2008]: we learn word embeddings using FastText, perform a hierarchical clustering of the vocabulary based on these, then use the resulting tree to learn a new language model. We call this approach "Clustering Tree". However, for all hyper-parameter settings, this tree structure did worse than a random one. We conjecture that its poor performance is because such a tree structure means that the deepest node decisions can be quite difficult.

Figure 3.9 shows the evolution of the test perplexity for a few epochs. It appears that most of the relevant tree structure can be learned in one epoch: from the second epoch on, the learned hierarchical softmax performs similarly to the flat one. Figure 3.8 and Table 3.2 show parts of the tree learned on the Gutenberg dataset, which appears to

make semantic and syntactic sense.

3.6 Discussion

In this Chapter, we have proposed a method to address the challenge of scale in word level language modeling. We introduced a provably accurate algorithm for jointly learning tree structure and data representation for hierarchical prediction, applied it to the task of language modeling and to a multi-class classification problem, and showed our model's ability to achieve favorable accuracy in competitive times in both settings.

A few obvious next steps come to mind in this direction. For example, the use of a single hierarchy can be quite limiting in some applications. Conversely, a forest model could be quite a bit more effective without drastically limiting the speed and computational gains of our methods. Figuring out how to use several trees jointly, and how to combine them in a theoretically grounded manner with similar guarantees, would be a significant development.

Additionally, there is till some theoretical work to be done on the relation between the J_n and the log-likelihood objective, and on relevant optimization guarantees for J_n. Here, we upper bounded the classification error based on the minimum value of all J_n's, while proposing an approximate algorithm to optimize the J_n's locally in parallel, without giving any guarantees about the minimum. In practice however, we observe that while the J_n's do increase from the beginning of training on the root and shallower nodes, the deeper nodes have to wait until the structure of the the tree has stabilized somewhat. In future work, we hope to provide an algorithm tailored more specifically to the weak hypothesis assumption of Theorem 3.1, and to extend the guarantees based on J_n to the log-likelihood objective.

We also note that, since it reduces the computational complexity dependence on the size of the output space, hierarchical prediction could make it possible to extend the prediction space beyond the original vocabulary. One could imagine, for example, adding common multi-word expressions or learning useful phrases through some form of Connectionist Temporal Classification like objective [Graves et al., 2006], possibly leading to better performances.

Some recent works using different topologies, such as hyperbolic spaces [Nickel and Kiela, 2017], have also shown promise in learning to embed inherently hierarchical structures in a continuous space. While these methods have not been used in ways which provide computational gains yet, it is possible that they might give rise to another family of theoretically grounded methods to produce hierarchical prediction systems.

Finally, it is worth keeping in mind that similar methods could be devised for other uses of adaptive hierarchies beyond tree-structured prediction. For example, forms of hierarchical addressing have started being used in neural networks with external memories [Chandar et al., 2016], but they tend to rely on heuristics which only apply to static memories or can be quite computationally expensive without providing any guarantees. Extensions of our proposed algorithm, or other ways of adapting the hierarchy jointly with the other model parameters, would provide a consequent boost to this paradigm.

Chapter 4

Globally Normalized Language Modeling

In this Chapter, we propose to take language modeling beyond word-level prediction, by using a globally normalized model to consider sequences in their entirety. To that end, we propose a Markov Random Field language model, whose structure is inspired by embeddings based neural language models. The main computational difficulty in optimizing such a model consists in computing the partition function to normalize over all possible sequences. We propose to take advantage of symmetries in our model to combine the paradigm of lifting with a Tree Re-Weighted upper bound on the partition function, and so provide an efficient algorithm to compute a lower bound on the global likelihood function in time mostly independent on the size of the corpus.

4.1 Introduction

In Section 1.4, we justified using language modeling as a learning objective by arguing that it was related to fundamental mechanisms of language understanding. We then expressed the language modeling objective as a probability distribution factorized over word choices using the chain rule. While this is one way to make the distribution tractable, it is easy to see how this choice might be less than ideal for our setting. Indeed, at each time step, we are asking a model to make a prediction which has to account for future decisions while only looking at information from the past. A global objective, which considers the whole statement to make a decision, would be much preferable in that respect.

This remark is coherent with the fact that n-gram language models, which draw on little world knowledge and only consider short-term context, can achieve similar perplexity to recurrent neural network models on most text datasets. This can be explained by the fact that *linguistically interesting* cases which rely on higher level reasoning are relatively rare. Consider the example presented in Chapter 3. The model would only achieve a tiny gain in perplexity by refining its candidate set for the last word from *box, safe* to the singleton *safe*, even though this is the step which relies on mid-term context and world knowledge. This appears to be an artifact of left to right word level prediction; indeed, a model which looked at the text as a whole would also increase the contribution to the global score of *secure* or *know how to* given the final *open the safe*. In addition, when a language model relies on latent features such as a parse tree or alternative interpretations of a word (whether from different possible POS tags or different forms of polysemy), *label bias* can also be a problem for locally normalized models [Bottou, 1991, Andor et al., 2016].

In this chapter, we consider a different class of language models. Instead of estimating the local probability of the next word given its context, we globally model the entire corpus as a Markov Random Field (MRF) language model. Undirected graphical models like MRFs have been widely applied in natural language processing as a way to flexibly model statistical dependencies; however, MRFs are rarely used for language modeling since estimating their parameters requires computing a costly partition function. Our contribution is to provide a simple to implement algorithm for very efficiently learning the parameters of this class of models. We take a variational approach to the optimization problem, and devise a lower bound on the log-likelihood using *lifted inference*. By exploiting the problem's symmetry, we derive an efficient approximation of the partition function. Crucially, each step of the final algorithm has time complexity of $O(K|\mathcal{V}|^2)$ where K is the n-gram context and $|\mathcal{V}|$ is the size of the vocabulary. Note that besides collecting statistics, this algorithm has no time dependence on the number tokens, potentially allowing its estimation speed to scale similarly to n-gram models. Experimentally, we demonstrate the quality of the models learned by our algorithm by applying it to language modeling on the Penn Treebank corpus [Marcus et al., 1993]. Additionally we show that this same estimation algorithm can be effectively applied to other common sequence modeling tasks such as part-of-speech tagging.

4.2 A Markov Random Field Language Model

Markov Random fields To avoid the issue of local normalization, we model our corpus as a sequence of random variables $T_1 \dots T_N$, for which we give a joint, globally normalized distribution. We specify this distribution with a Markov random field. A Markov random field is defined by a graph structure $\mathcal{G}$, and a set of potentials $(\theta^c)_{c \in \mathcal{C}}$,

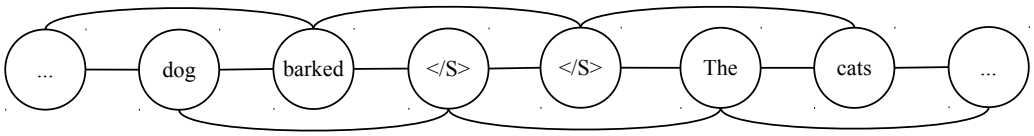

Figure 4.1: A segment of the linear-chain model over a small corpus.

where $\mathcal{C}$ is defined as the set of cliques in graph $\mathcal{G}$. Let $\mathbf{t}$ denote a specific assignment of $T_1 \dots T_N$, let $\mathbf{t}_c = (t_i)_{i \in c}$, the log-probability of a sequence $\mathbf{t}$ is then:

$$\log(p(\mathbf{t}; \theta)) = \sum_{c \in \mathcal{C}} \theta^c_{\mathbf{t}_c} - A(\theta)$$

Where the normalizing factor $A(\theta)$ is called the log-partition function. The complexity of computing this log-partition exactly is in general exponential in the size of the largest clique of $\mathcal{G}$. However, there exist many algorithms to compute an approximation in time $O(N|\mathcal{V}|^2)$. In this work, we go one step further, by devising a method which dispenses with the N factor.

Chain MRF In the rest of the paper, we will focus on a specific class of linear-chain MRFs, where the set of edges $\mathcal{E}$ of $\mathcal{G}$ connects each token to its K immediate neighbors in a sentence. To connect the whole corpus while maintaining independence of tokens across sentences, we also include separator tokens $\langle /S \rangle$ at the end of each sentence. Figure 4.1 presents such a model for $K = 2$. For simplicity of exposition, we will focus on pairwise MRFs, for which only size 2 cliques (edges) have potentials. The lifted inference method, however, can easily be extended to models which consider other cliques.

Under this model, the log-likelihood of a corpus $\mathbf{t}$ is given by:

$$\log(p(\mathbf{t};\theta)) = \sum_{i=1}^{N}\sum_{j=i+1}^{i+K}\theta^{i,j}_{t_i,t_j} - \log(\sum_{\mathbf{s}\in\mathcal{S}^N}\exp(\sum_{i=1}^{N}\sum_{j=i+1}^{i+K}\theta^{i,j}_{s_i,s_j})) \tag{4.1}$$

Where $\mathcal{S}^N$ is the set of possible token sequences of length N, and $\forall s \in \mathcal{S}^N, \forall i \in [1,K],\ t_{N+i} = s_{N+i} = \langle /S \rangle$.

These graphical models define a large family of log-linear distributions, depending on the value of K and the parametrization of the edge log-potentials θ. In the applications that follow θ will either be given (and optimized over) explicitly, or represented as a product of low-rank matrices. We will show how to optimize the likelihood of the corpus for both these settings.

Low Rank Approximation of Chain MRF We now consider different low-rank realizations of the log-potentials θ. Suppose that $\theta^{i,j}$ only depends on $|j-i|$, that is to say, parameters are shared across edges of the same length (in which case we shall simply write $\theta^{i,j} = \theta^{|j-i|}$), we can have for example:

1 $\theta^{|j-i|}_{t_i,t_j} = \mathbf{U}_{t_i}\mathbf{R}^{|j-i|}\mathbf{W}_{t_j}$

2 $\theta^{|j-i|}_{t_i,t_j} = \mathbf{U}_{t_i}\mathbf{W}^{|j-i|}_{t_j}$

3 $\theta^{|j-i|}_{t_i,t_j} = \mathbf{U}^{|j-i|}_{t_i}\mathbf{V}_{t_j} + \mathbf{V}_{t_i}\mathbf{W}^{|j-i|}_{t_j}$

In the experimental results for this paper, we use the second formulation. One interesting property of these models is that since the Markov blanket of a word consists only of its immediate neighbors, its conditional likelihood can be expressed as:

$$p(t_i|\mathbf{t_{-i}}) = p(t_i|t_{i-K,\ldots,i+K})$$
$$= \frac{\exp(\sum_{l=1}^{K} \theta^l_{t_{i-l},t_i} + \theta^l_{t_i,t_{i+l}})}{\sum_{s\in\mathcal{S}} \exp(\sum_{l=1}^{K} \theta^l_{t_{i-l},s} + \theta^l_{s,t_{i+l}})}$$

This class of probability functions corresponds to those defined by a bi-directional log-bilinear neural language model. And in fact model 1 ($\theta = \mathbf{URW}$) can easily be rewritten in terms of the bi-directional version of Mnih's Log Bi-Linear model. Conversely, log-bilinear NNLMs can be seen as optimizing the pseudo-likelihood (defined as $\prod_{i=1}^{n} p(t_i|\mathbf{t^{-i}})$) of a MRF of the class we present in this paper. Since the pseudo-likelihood is a consistent estimator of the likelihood, we expect our factorization to have properties similar to those of the embeddings learned by log-bilinear neural language models.

4.3 Efficient Learning of a Chain MRF

Efficient Learning Using Lifted Variational Inference We now outline our method for optimizing the likelihood of a corpus under our class of models. Learning undirected graphical models is challenging because of the global normalization constant, or partition function. We derive a tractable algorithm by using a variational approximation: we define a lower bound on the data likelihood [Wainwright et al., 2005, Yanover et al., 2008], and alternate between finding the tightest version of that bound and taking a gradient ascent step in the parameters of the model. The novelty of our method comes from the fact that for the bound we define, both the tightening and gradient step only require us to consider K pairwise moments, i.e. the running time of learning will be indepen-

dent of the size of the corpus. We achieve this by showing how to reduce the learning task to *lifted* variational inference, allowing us to build upon recent work by [Bui et al., 2014]. We then derive an algorithm to efficiently perform lifted variational inference using belief propagation and dual decomposition. The overall learning algorithm is simple to implement and runs very fast.

Creating Symmetry using a Cyclic Model To obtain the kind of symmetric lower bound on the likelihood we want, we apply a small change to the original model, which we argue does not change its induced probability distribution. The modification is to make the model completely symmetrical by wrapping the MRF around, and linking the last $\langle/S\rangle$ tokens of the corpus to the K first $\langle/S\rangle$. The new model is shown in Figure 4.2.

Variational lower bound Let θ^0 denote single node unary potentials, and θ^l the edge potentials as defined in Section 4.2, the log-likelihood of text $\mathbf{t}$ under the linear model is then:

$$\begin{aligned}
\log(p(\mathbf{t})) &= \sum_{i=1}^{N}\left(\theta^0_{t_i} + \sum_{l=1}^{K}\theta^l_{t_i,t_{i+l}}\right) - A^0(\theta) \\
A^0(\theta) &= \log(\sum_{\mathbf{s}\in\mathcal{S}}\exp(\sum_{i=1}^{N}\left(\theta^0_{s_i} + \sum_{l=1}^{K}\theta^l_{s_i,s_{i+l}}\right)))
\end{aligned}$$

Our first approximation here is to replace the linear-chain model with the cyclic one to obtain $A(\theta) \geq A^0(\theta)$, which gives us a lower bound on the log-likelihood.

Unfortunately, both of these partition functions are extremely costly to compute for any reasonable vocabulary size, as dynamic programming would have running time $O(N|\mathcal{V}|^K)$. However, it is easy to formulate upper bounds on A, which give rise to a family of lower bounds on the log-likelihood. We start by formulating the computa-

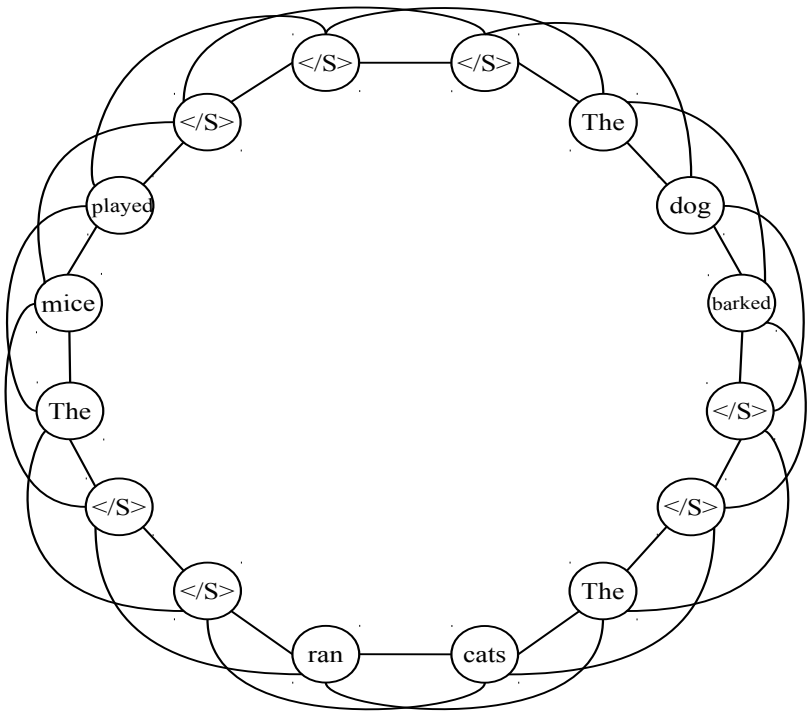

Figure 4.2: The cyclic model with $N = 16$ and a sequence $s \in \mathcal{S}$ corresponding to three sentences of text.

tion of the partition function as an optimization problem:

$$\begin{aligned} A(\theta) &= \log(\sum_{\mathbf{s}\in\mathcal{S}} \exp(\sum_{i=1}^{N} \left(\theta^0_{s_i} + \sum_{l=1}^{K} \theta^l_{s_i,s_{i+l}} \right))) \\ &= \max_{\mu\in\mathcal{M}} \sum_{i=1}^{N} \left(\langle \mu^i, \theta^0 \rangle + \sum_{l=1}^{K} \langle \mu^{i,i+l}, \theta^l \rangle \right) + H(\mu) \end{aligned}$$

where $\mathcal{M}$ denotes the marginal polytope, theset of valid marginal probabilities [Wainwright and Jordan, 2003]. We then make two approximations to make solving this approximation problem easier. First, we replace $\mathcal{M}$ with the local consistency polytope $\mathcal{LC}$. Since $\mathcal{LC} \supset \mathcal{M}$, this gives us an upper bound on the original solution. Secondly, we replace the entropy $H(\mu)$ with the tree reweighted (TRW) upper bound [Wainwright

et al., 2005]:

$$H(\mu) \leq \bar{H}^{\rho}(\mu) = \sum_{i=1}^{N} \left(H(\mu^i) - \sum_{l=1}^{K} \rho_{i,i+l} I(\mu^{i,i+l}) \right)$$

where $\rho_{i,j}$ denotes the probability and $\mu_{i,j}$ the pseudo-marginal of edge (i,j) in a covering set of forests for the MRF. Let:

$$\bar{A}(\theta;\rho) = \max_{\mu \in \mathcal{LC}} \sum_{i=1}^{N} \left(\langle \mu^i, \theta^0 \rangle + \sum_{l=1}^{K} \langle \mu^{i,i+l}, \theta^l \rangle \right) + \bar{H}^{\rho}(\mu)$$

Using this variational approximation, we now have an upper bound on the log-partition function which can be computed by solving a convex optimization problem. Altogether this then gives us the following tractable lower bound on the log-likelihood:

$$\begin{aligned} \log(p(\mathbf{t})) &\geq \sum_{i=1}^{N} \left(\theta^0_{t_i} + \sum_{l=1}^{K} \theta^l_{t_i,t_{i+l}} \right) - \bar{A}(\theta;\rho) \\ &= \bar{\mathcal{L}}(\theta, \mathbf{t}; \rho). \end{aligned} \tag{4.2}$$

Learning using gradient ascent then requires that we compute the derivative of $\bar{A}(\theta;\rho)$, which we will show is the μ that maximizes the variational optimization problem (we return to this process in more detail in the next section). We can therefore reduce the learning task to that of repeatedly performing approximate inference using TRW. Fast combinatorial solvers for TRW exist, including tree-reweighted belief propagation [Wainwright et al., 2005], convergent message-passing based on geometric programming [Globerson and Jaakkola, 2007], and dual decomposition [Jancsary and Matz, 2011]. However, we next show that by taking advantage of the symmetries present in the optimization problem, it is possible to solve it in time which is independent of N, the number of words in the corpus.

Lifting the objective Our key insight is that because of the parameter sharing in our model, each of the random variables in the cyclic model are indistinguishable. More precisely, there is an automorphism group of rotation which can be applied to the sufficient statistic vector and to the model parameters whwithout changing the joint distribution [Bui et al., 2013]. When such symmetry exists, [Bui et al., 2014] show that without loss of generality one can choose the edge appearance probabilities to be symmetric, which in our setting corresponds to choosing a ρ such that $\forall i, j,\ \rho_{i,j} = \rho_{|j-i|}$ (i.e., the tightest TRW upper bound on $A(\theta)$ can be obtained by a symmetric ρ). When the edge appearance probabilities are chosen accordingly, since the objective is strictly concave and the variables are rotationally symmetric, it follows from Theorem 3 of [Bui et al., 2014] that the optimum must satisfy the following property:

$$\forall 1 \leq i, j \leq N, 1 \leq l \leq K, \mu^i = \mu^j \text{ and } \mu^{i,i+l} = \mu^{j,j+l}. \tag{4.3}$$

We will take advantage of this structural property to dramatically simplify the variational optimization problem. In particular, using the notation μ_V^0 to refer to the single-node marginal (there is only one) and $\mu_E^l(x_1, x_2)$ to refer to the edge marginal corresponding to the potential θ^l, we have:

$$\begin{aligned} \bar{A}(\theta) = \max_{\mu} N \times \Big[& \langle \mu_V^0, \theta^0 \rangle + H(\mu_V^0) \\ & + \sum_{l=1}^{K} \Big(\langle \mu_E^l, \theta^l \rangle - \rho_l I(\mu_E^l) \Big) \Big] \end{aligned} \tag{4.4}$$

where the maximization is subject to the non-negativity constraints $\mu_V^0, \mu_E^1, \ldots, \mu_E^K \geq 0$, sum-to-one constraints $\sum_{x_v} \mu_V^0(x_v) = 1$ and $\forall l,\ \sum_{x_1,x_2} \mu_E^l(x_1, x_2) = 1$, and pairwise

consistency constraints:

$$\sum_{x_1} \mu_E^l(x_1, x_2) = \mu_V^0(x_2) \qquad \forall l, x_2, \tag{4.5}$$

$$\sum_{x_2} \mu_E^l(x_1, x_2) = \mu_V^0(x_1) \qquad \forall l, x_1. \tag{4.6}$$

The optimal μ_E^l is guaranteed to be symmetric, and so we could have used a slightly more compact form of the optimization problem c.f. [Bui et al., 2014]. However, we prefer this form both because it is easier to describe and because it is more amenable to solving efficiently.

The lifted problem, Eq. 4.4, has only $|\mathcal{V}| + K|\mathcal{V}|^2$ optimization variables, instead of the $N(|\mathcal{V}| + K|\mathcal{V}|^2)$ of the original objective. However, it remains to figure out how to solve this optimization problem. [Bui et al., 2014] solve the lifted TRW problem using Frank-Wolfe, which has to repeatedly solve a linear program over the same feasible space (i.e., Eqs. 4.5 and 4.6). These linear programs would be huge in our setting, where $|\mathcal{V}|$ can be as large as $10,000$, leading to prohibitive running times.

Dual Decomposition We now derive an efficient algorithm based on dual decomposition to optimize our lifted TRW objective. We will have an upper bound on the log-partition function, and thus a lower bound on the likelihood, for *any* valid edge appearance probabilities. However, our algorithm requires a specific choice (which is the same for all edges):

$$\forall l, \qquad \rho_l = \frac{1}{K+1}. \tag{4.7}$$

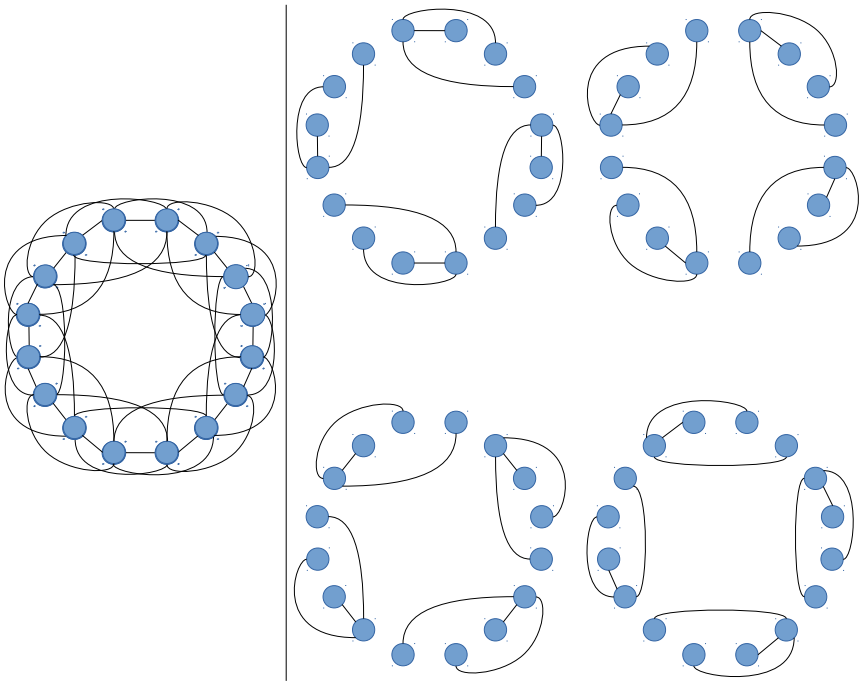

Figure 4.3: The set of $K+1$ covering forests used for $K=3$, $N=16$. Each edge is represented in exactly one forest.

We assume that the corpus length N is a multiple of $K+1$, which can always be achieved by adding "filler" $\langle/S\rangle$ tokens. To prove that Eq. 4.7 defines valid edge appearance probabilities, we demonstrate a set of $K+1$ forests T such that $\rho_{ij} = \sum_{T:ij\in T} \rho_T$, where for all T, $\rho_T = \frac{1}{K+1}$. In particular, we take forests which are made up of disconnected stars, rotated so that each edge is covered exactly once. Figure 4.3 illustrates this choice of forests.

Using this, we can rewrite the objective in Eq. 4.4 as:

$$\bar{A}(\theta) = \frac{N}{K+1} \max_{\mu} (K+1) \Big[\langle \mu_V^0, \theta^0 \rangle + H(\mu_V^0) + \sum_{l=1}^{K} \Big(\langle \mu_E^l, (K+1)\theta^l \rangle - I(\mu_E^l) \Big) \Big]. \tag{4.8}$$

Finally, rather than optimizing over Eq. 4.8 explicitly, we re-write it in a form in which we can use a belief propagation algorithm to perform part of the maximization. To do so, we introduce redundant variables μ_V^l for $l \in [1, K]$, enforce that they are equal to μ_V^0 and use them instead of μ_V^l for each pairwise consistency constraint. The resulting equivalent form of the optimization problem is:

$$\bar{A}(\theta) = \frac{N}{K+1} \max_{\mu} \sum_{l=0}^{K} \langle \mu_V^l, \theta^0 \rangle + \sum_{l=1}^{K} \langle \mu_E^l, (K+1)\theta^l \rangle + \sum_{l=0}^{K} H(\mu_v^l) - \sum_{l=1}^{K} I(\mu_E^l), \tag{4.9}$$

subject to non-negativity and sum-to-1 constraints, and:

$$\begin{aligned} \sum_{x_1} \mu_E^l(x_1, x_2) &= \mu_V^l(x_2) \qquad \forall l \in [1, K], x_2, \\ \sum_{x_2} \mu_E^l(x_1, x_2) &= \mu_V^l(x_1) \qquad \forall l \in [1, K], x_1, \\ \mu_V^l(x_1) &= \mu_V^0(x_1) \qquad \forall l \in [1, K], x_1. \end{aligned} \tag{4.10}$$

If one ignores the equality constraints (4.10) then we see that the constrained optimization problem in (4.9) corresponds exactly to a Bethe variational problem for the tree-structured Markov random field shown in Figure 4.4, and as a result can be maximized in linear time using belief propagation (Theorem 4.2b [Wainwright and Jordan, 2008]).

Our next step is then to introduce these constraints in a way that still allows for efficient optimization.This can be achieved through the use of Langrangian duality, in an approach similar to that of [Hazan and Urtasun, 2010] and [Meshi et al., 2010]: by formulating the right dual problem, we obtain a tight bound on our objective which can

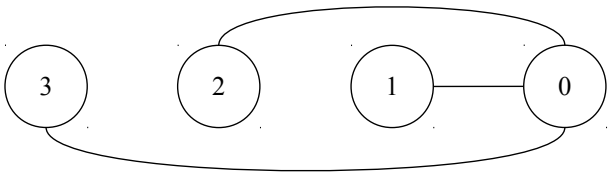

Figure 4.4: The tree corresponding to the maximization sub-problem in the lifted inference, for $K = 3$.

still be maximized through message passing.

We then introduce Lagrange multipliers $\delta_V^l(x_l)$ for each constraint in (4.10) and form the Lagrangian by adding $\sum_{l=1}^{K}\sum_{x_l=1}^{|\mathcal{V}|}\delta_V^l(x_l)(\mu_V^l(x_l) - \mu_V^0(x_l))$ to the objective (4.9). Re-arranging terms and omitting the constant, we obtain that the dual objective is:

$$O^{\delta}(\theta,\mu) = \langle \mu_V^0, \theta^0 - \sum_{l=1}^{K}\delta_V^l\rangle + \sum_{l=1}^{K}\langle \mu_V^l, \theta^0 + \delta_V^l\rangle \\ + \sum_{l=1}^{K}\langle \mu_E^l, (K+1)\theta^l\rangle + \sum_{l=0}^{K} H(\mu_v^l) - \sum_{l=1}^{K} I(\mu_E^l). \tag{4.11}$$

Since the primal problem is concave and strictly feasible (it is feasible with no inequality constraints), Slater's conditions are met and we have strong duality. Thus,

$$A(\theta) \leq \bar{A}(\theta) = \frac{N}{K+1}\min_{\delta}\max_{\mu\in\mathcal{LC}} O^{\delta}(\theta,\mu). \tag{4.12}$$

One useful property of the above is that we have a valid upper bound on $A(\theta)$, the log-partition function of the circular model, for *any* choice of the dual variables δ [Meshi et al., 2010]. For fixed δ, computing the upper bound simply requires one pass of belief propagation in the tree MRF shown in Figure 4.4, for a running time of $O(K|\mathcal{V}|^2)$.

Learning Algorithm Recall that our goal is to estimate parameters to maximize the objective $\bar{\mathcal{L}}(\theta, \mathbf{t}, \rho)$ given in equation (4.2). Letting $\hat{\mu}_{\mathbf{t}}$ denote the observed moments in text $\mathbf{t}$, we have:

$$\begin{aligned}
\bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) &= N \times \langle \hat{\mu}_{\mathbf{t}}, \theta \rangle - \bar{A}(\theta) \\
&= N \times \left(\langle \hat{\mu}_{\mathbf{t}}, \theta \rangle - \frac{\min_\delta \max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu)}{K+1} \right) \\
&= N \times \max_\delta \left(\langle \hat{\mu}_{\mathbf{t}}, \theta \rangle - \frac{\max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu)}{K+1} \right) \\
&= N \times \max_\delta L(\theta, \mathbf{t}; \delta),
\end{aligned}$$

where $L(\theta, \mathbf{t}; \delta) = \langle \hat{\mu}_{\mathbf{t}}, \theta \rangle - \frac{\max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu)}{K+1}$. Hence, for any δ, $N \times L(\theta, \mathbf{t}; \delta)$ defines a lower bound over the log-likelihood of corpus $\mathbf{t}$, which can be made tighter by optimizing over δ. The variational learning algorithm will then consist of alternating between making this bound tight (Algorithm 4.1), and taking gradient steps in θ (Algorithm 4.2).

Tightening the bound For a given value of the parameters θ, the tightest bound is obtained for $\delta^* = \arg\min_\delta \max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu)$. We can find this minimizer through a sub-gradient descent algorithm. Indeed, let μ^* be a maximizer of $O^\delta(\theta, \mu)$, we have the following sub-gradient in δ:

$$\forall l \in [1, K], \quad \nabla \delta_V^l = \mu_V^{l*} - \mu_V^{0*} \tag{4.13}$$

The maximization problem corresponds to computing the partition function of the tree-structured MRF given in Figure 4.4 with the following potentials:

- $\forall l \in [1, K]$ $\bar{\theta}_V^l = \theta^0 + \delta_V^l$
- $\bar{\theta}_V^0 = \theta^0 - \sum_{l=1}^K \delta_V^l$

Algorithm 4.1 Tightening the bound

input: model parameters θ
repeat
  compute $\bar{\theta}(\theta, \delta)$ for the lifted MRF (Figure 4.4)
  compute $\mu(\bar{\theta})$ (belief propagation)
  compute $\nabla\delta$ (Equation 4.13)
  Take sub-gradient step: $\delta^{new} = \delta - \alpha\nabla\delta$
until μ satisfies primal constraints
output: pseudo-marginals μ

- $\forall l \in [1, K], \;\; \bar{\theta}^l_E = (K+1)\theta^l$

This can be solved using the belief propagation algorithm.

Gradient Ascent The marginals computed at δ^* can then be used to compute gradients for our main objective. Recall that our aim is to maximize the objective function

$$\bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) = N \times L(\theta, \mathbf{t}; \delta^*(\theta)),$$

where $\delta^*(\theta)$ is the output of Algorithm 4.1. Moreover, for any value of δ, even before optimality, we have:

$$\nabla_\theta \max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu) = (K+1)\arg\max_{\mu \in \mathcal{LC}} O^\delta(\theta, \mu)$$

Hence:

$$\nabla_\theta \bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) = N \times \left(\hat{\mu} - \arg\max_{\mu \in \mathcal{LC}} O^{\delta^*}(\theta, \mu) \right)$$

The gradients in the model parameters can then be obtained through the application of a simple chain rule. For the second factorization of θ presented in Section 4.2, which we use in experiments, we get:

Algorithm 4.2 Learning algorithm

input: data $\mathbf{t} = (t_i)_{i=1}^n$
collect pairwise moments $\hat{\mu}$ from the data
repeat
  find the tightest bound $L(\theta, \mathbf{t}; \delta)$ (Algorithm 4.1)
  compute gradient in θ and parameters
  take gradient step to update parameters
  re-compute θ
until convergence
output: estimated parameters (e.g. $\mathbf{U}$, $\mathbf{R}$, $\mathbf{W}$)

- $\forall s \in T, \forall d \in [1, D]$,

$$\nabla_{U_{s,d}} \bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) = \sum_{l=1}^{K} \sum_{t \in T} \left(\nabla_{\theta_{s,t}} \bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) \times W_{t,d}^{l} \right)$$

- $\forall t \in T, \forall d \in [1, D], \forall l \in [1, K]$,

$$\nabla_{W_{t,d}^{l}} \bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) = \sum_{s \in T} \left(U_{s,d} \times \nabla_{\theta_{s,t}} \bar{\mathcal{L}}(\theta, \mathbf{t}, \rho) \right)$$

These can be used to perform gradient ascent on the objective function, as outlined in Algorithm 4.2.

4.4 Experiments

We conducted experiments using the lifted algorithm to examine its practical efficiency, effectiveness at estimating gradients, and the properties of the tree re-weighted bound. We implemented models for two standard natural language tasks: language modeling and part-of-speech tagging.

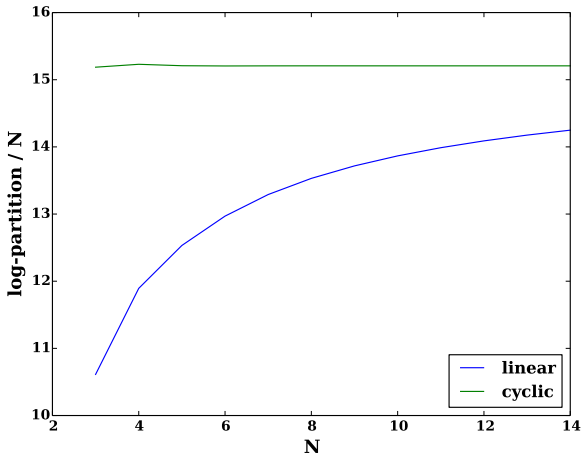

Figure 4.5: Comparison of the linear-chain log-partition value to the cyclic upper-bound as corpus size increases.

Setup For model parameter optimization (the gradient step in algorithm 4.2) we use L-BFGS [Liu and Nocedal, 1989] with backtracking line-search. For dual optimization (algorithm 4.1), we used 200 sub-gradient iterations, each requiring a round of belief propagation. Our sub-gradient rate parameter α was set as $\alpha = 10^3/2^t$ where t is the number of preceding iterations where the dual objective did not decrease. Our implementation of the algorithm and baselines was implemented in multithreaded C++ for efficiency.

For language modeling we ran experiments on the Penn Treebank (PTB) corpus with the standard language modeling setup: sections 0-20 for training ($N = 930$k), sections 21-22 for validation ($N = 74$k) and sections 23-24 ($N = 82$k) for test. For this dataset the lexicon size is $|\mathcal{V}| = 10$k, and rare words are replaced with `UNK`.

For part-of-speech tagging we use the tagged version of the Penn Treebank corpus [Marcus et al., 1993]. We use section 2-21 for training, section 22 for validation and section 23 for test. For this corpus the tag size is $T = 36$ and we use the full vocabulary

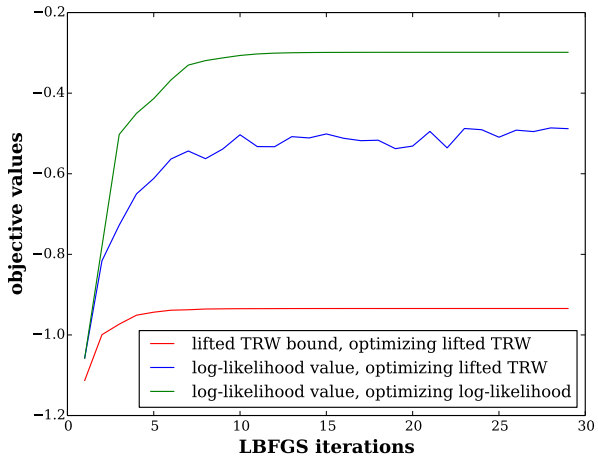

Figure 4.6: Comparison of a model trained by optimizing exact likelihood versus lifted TRW objective. The blue line shows the exact log-likelihood of the model learned by optimizing the lifted TRW bound.

size $|\mathcal{V}|$ is around 30k.

Experiments First to confirm the properties of the algorithm we ran experiments on a small synthetic data set with $N \in [3, 14]$ and $|\mathcal{V}| = 4$. The small size of this data set allows us to exactly compute the log-partition for the original linear-chain formulation. Figure 4.5 shows how the value of the lifted bound gets closer to the exact log-partition as the corpus size increases.

We also explored training with lifted inference using this data set ($N = 12, |\mathcal{V}| = 4$). For these experiments we compare training two models, the first model uses the lifted inference and the second uses exact gradients from the linear-chain formulation. Additionally we also compute the exact log-likelihood using the parameters of the lifted model. Figure 4.6 shows the results of this setup. The lifted log-likelihood gives an underestimate of the log-likelihood, but the learned parameters yield an exact log-likelihood close to the linear-chain model.

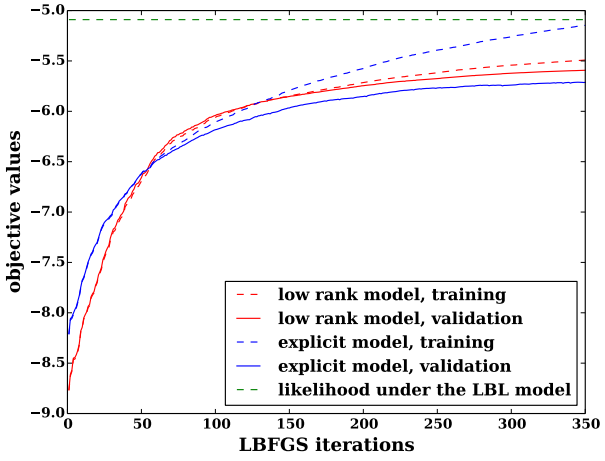

Figure 4.7: The green line shows the fixed value the validation log-likelihood of an LBL model trained on PTB. The red and blue lines give lower bounds on the log-likelihood (lifted objective).

Next we applied the lifted algorithm to a language modelling task on the Penn Treebank corpus. We first compared the explicit full-rank model to the low-rank potentials from Section 4.2, $\theta_{t_i,t_j}^{|j-i|} = \mathbf{U}_{t_i}\mathbf{W}_{t_j}^{|j-i|}$, for $D = 30$ and $K = 2$. The results are presented in figure 4.7. As expected, the explicit model is prone to over-fitting, and gets to a worse validation objective. We also evaluated the loss incurred in bound tightness due to using a non optimal ρ: it turned out to account for less than 0.3 points of log-likelihood.

Another advantage of using low-rank potentials is that they produce embedded representations of the vocabulary. Table 4.1 shows a sample of embeddings learned for the MRF compared to those obtained with the word2vec algorithm (with $D = 100$ and a window size of 4). We also tried training our algorithm by performing stochastic gradient descent on the pseudo-likelihood of the corpus under our model (which, as mentioned in section 4.2, is the same learning objective as that of the log-bilinear model of [Mnih and Hinton, 2007]). The column MRF SGD shows the embeddings obtained

WORD	MRF LIFTED	MRF SGD	WORD2VEC
elected	named appointed assistant	fell billings indicated	named appointed bank-holding
company	firm industry group	he it corp.	holding anacomp uniroyal
red	conservative freedom black	vietnamese delegation judge	cross tape delicious
has	had is was	have had n't	had been have
dollar	currency economy government	intergroup market uses	currency pound stabilized
dollars	francs lows highs	share went accounts	us$ millions billions
jack	richard david carl	like needed first	kemp porter timothy
coffee	food network business	specially humanitarian liberal	flat-rolled sport recycling

Table 4.1: Nearest neighbours in different embeddings. MRF LIFTED are the embeddings learned by our algorithm. MRF SGD are obtained by running stochastic gradient descent for 48 hours on the pseudo-likelihood objective: the algorithm did not converge in that time, hence the mostly bad results. WORD2VEC are the vectors learned by the word2vec software of [Mikolov et al., 2013]

after 48 hours of training. In comparison, our algorithm reaches its optimal objective value on the validation dataset in 3 hours on the Penn Treebank dataset.

Finally we ran experiments on part-of-speech tagging. For this task we use a different MRF graphical structure. Each tag node is connected to its K neighbors as well as the M nearest-words. We use a different set of covering forests for this model which is shown in figure 4.8. As with language modelling the partition function for this model would be very inefficient to compute explicitly. However, given a sentence, the best tagging can be found efficiently by dynamic programming.

For this model, we also employ explicit features for pairwise potentials, i.e. $\theta^m_{t_i,w_i} = \mathbf{u}f(t_i, w_i)$ and $\theta^l_{t_i,t_{i+l}} = \mathbf{v}g(t_i, t_{i+l})$ where $\mathbf{u}, \mathbf{v}$ are parameter vectors and f, g are pre-

Model	Total Acc	Unk Acc
HMM	95.8	65.4
Lifted MRF	96.0	76.0

Table 4.2: Comparison of tagging accuracy between the lifted MRF and an HMM in total and on unseen words.

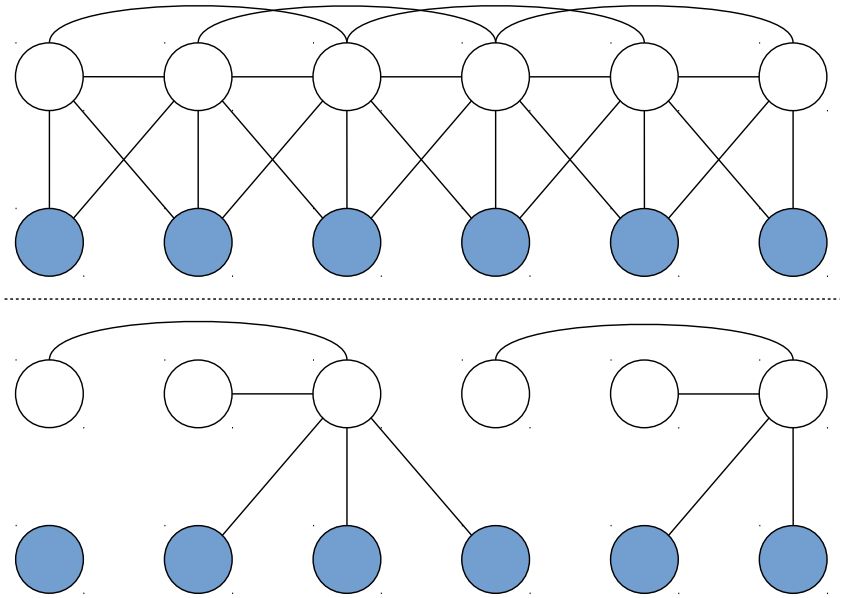

Figure 4.8: The POS tagging model for $K = 2, M = 3$, and a decomposition for the lifted inference algorithm

defined feature functions. For g we use tag-pair indicator features, and for f we use standard features on capitalization, punctuation, and prefixes/suffixes. This model and features are identical to a standard conditional random field tagger; however we optimize for joint likelihood.

It is known that joint models are less effective than discriminative conditional models for this task [Liang and Jordan, 2008], but we can compare performance to a similar joint model. We compare this model with $K = 1$ to a standard first-order HMM tagging

model using the TnT tagger [Brants, 2000] with simple rare word smoothing. Table 4.2 shows the results. The lifted model achieves similar total accuracy, but has much better performance on unseen words, due to its feature structure.

4.5 Discussion

This Chapter introduces a Markov Random Field Language Model which extends upon Neural Language Models, and presents a fast lifted inference algorithm with complexity independent of the length of the corpus. This presents a firs step towards building models which consider statements or even documents in their entirety, which would be an important step towards being able to use language modeling to teach text understanding systems to do more complex reasoning.

Still, our model suffers from some consequential limitations, which will need to be remedied in future works. Complexity wise, while we provide a partition function which can be computed independently of the length of the corpus considered (after gathering relevant statistics once), the computational and memory cost is still quadratic in the vocabulary size, which can restrict the application domain. We could imagine some cheaper approximations with minimal modifications to the algorithm, such as only optimizing the upper and left rectangles of the matrix, but a more principled approach would be appreciated.

The lack of interaction between distant words, as well as the restriction to pair-wise potentials, also somewhat limit the expressiveness of the model. In word-normalized neural language models, these features can be handled by latent recurrent states and the use of more non-linearities. How to adapt those to a globally normalized model remains an open question.

Additionally, a common bottleneck of systems with a natural language output is the $\arg\max$ over the space of possible sequences. In the setting of our Markov Random Field language model, this would correspond to performing *maximum a posteriori* rather than *marginal* inference, which we do in this Chapter. Thus, extending our method to the former setting might be quite significant.

Finally, the lifting method we propose here relies on symmetry, which is broken for conditional models. While this can in theory be remedied by simply using Bayes' rule (as long as we have access to the marginal distribution of the conditioning variables), doing so can be impractical, and we hope that future work can find ways to retain the computational gains of lifting while allowing the use of more flexible models.

Chapter 5

Towards Better Learning Objectives

In this Chapter, we return to the problem of finding a good learning objective to develop and train text understanding systems. To that end, we start by reviewing a set of other common language tasks in terms of how much of the mechanisms they rely o can be learned in a generally useful way, and of whether they can easily be learned from freely available natural language data. We then look towards modeling the coherence of discourse to come up with new tasks which meet our requirements on both fronts. We show that discriminative objectives inspired by this notion can be used to train sentence representation systems in an unsupervised fashion, and that the meaning encoded in these representations can be broadly useful for a variety of language tasks.

5.1 Introduction

In Section 1.3, we argued that language modeling could be a useful objective for learning to represent the meaning of a text. Chapters 3 and 4 then outlined some of the drawbacks of this objective, and worked towards minimizing the effects of these drawbacks. In the present Chapter, we take a slightly different approach, by considering what other tasks can be used to learn similar representations, what their advantages and disadvantages are compared to language modeling, and eventually presenting a new objective based on the notion of discourse.

Most sequences of words (and *a fortiori* characters) do not correspond to interpretable language. Even deviating from natural language sentences in very restricted ways often leads to a loss of coherence. For example, consider changing the mode of the verb in the simple statement "A bird was flying." to "A bird was flies.", or switching the article in the sentence "The sky is blue." to "Three sky is blue.": a language model must learn to assign a low score to each of the two new sentences for reasons which have little to do with the meaning of any previously read text. We already mentioned this issue in Chapter 4 when discussing the limitations of locally normalized language modeling: while the learning objective gives as much importance to each word prediction in the output, higher level reasoning is typically only required for a few of those. This can make it difficult for a model to efficiently learn when making use of longer term context or relying on common sense reasoning is necessary. We proposed a global prediction objective to mitigate this issue. Other works have provided objectives which attempt to focus on more "linguistically significant" words, such as the LAMBADA [Paperno et al., 2016] or Children Books Test corpora [Hill et al., 2015] (the latter also attempts to make the task more relevant by making the system choose between likely confounders).

By focusing on one-word answers, these settings also remain tractable: the complexity is strictly less than that of word-level language modeling. Efforts to automate this process, in the form of an adversarial two-player game where one of the player is in charge of picking the difficult words and the other is supposed to predict them (similarly to *e.g.* [Sukhbaatar et al., 2017]), are also promising.

A related setting is that of extractive question answering, where the model takes a document and question, and is asked to return a text segment from the document as an answer. There is a growing amount of labeled datasets for this task (e.g. SQuAD [Rajpurkar et al., 2016]), and recent work has looked into automatically expanding question answering datasets [Yang et al., 2017]. Still, providing enough data to cover all domains of language will require significant amounts of human effort. Both of the above mentioned problems (focused word completion and extractive question answering) heavily rely on a good encoding or indexing of an input document and the new information expressed within, while attempting to minimize the burden of producing interpretable answer or evaluating the interpretability of a complex statement. Compared to previously proposed language objectives, this adds a useful complementary learning signal, but would not necessarily be enough to learn all of the properties of language which are relevant to text understanding. Additionally, in spite of efforts to automate data set creation, the best results are still currently obtained with curated training sets, requiring significant amounts of human work.

Another approach to helping the model focus on desired properties of language consists on explicitly conditioning a language model for each new sentence on a representation of previous statements. For example, the SkipThought model learns a function which takes as input a sentence and outputs a probability distribution over words in the sentences to the left and right [Kiros et al., 2015], and recent works have attempted to

learn a distribution over the next sentence in a text given representations of the previous ones [Wang and Cho, 2016], or of next utterance in a dialogue given its history [Lowe et al., 2017]. In doing so, the model lets the decoder learn all of the properties of language which have to do with immediate context coherence and grammar, and allows the encoder to focus on properties of the sentence which matter in the longer run. Training data is also readily available, at least in the SkipThought setting (dialogue is somewhat more limited, and restricted to specific domains). However, it is unclear whether the encoding function learned in this setting will remain useful when we need to apply it to full documents, and these works still use a language modeling objective which suffers from the biases and limitations outlined in previous Chapters.

Finally, recent works have endeavored to use supervised language tasks to learn generally useful representations of sentences. One such task has been Natural Language Inference, as it is generally assumed to more often require higher level reasoning and rely on good use of compositionality, common sense reasoning or co-reference. Large scale resources have recently become available [Bowman et al., 2015, Williams et al., 2017], and sentence representations learned for this task have been shown to be widely useful [Conneau et al., 2017]. However, while these results are extremely encouraging in terms of transferability of learned representation, they still rely on vast amounts of human-labeled data. This raises the following question: is there a similar unsupervised or semi-supervised objective which relies on the same mechanisms?

Understanding the structure of discourse, for example, requires a good grasp on both semantics and pragmatics [Hobbs, 1979], which constitute a more elaborate form of text understanding. Inspired by this, the current Chapter proposes a set of new objectives for the unsupervised training of neural network sentence encoders. It exploits signals from paragraph-level discourse coherence to train these models to understand text, perform-

ing, or at the very least approximating, the $read$ function of Section 1.3. Additionally, the proposed objective depends on properties of naturally occurring text, giving it access to the same profusion of training data as language modeling, and is discriminative, making it much cheaper to compute.

5.2 Background

Modern artificial neural network approaches to natural language understanding tasks like translation [Sutskever et al., 2014, Cho et al., 2014b], summarization [Rush et al., 2015], and classification [Yang et al., 2016] depend crucially on subsystems called *sentence encoders* that construct distributed representations for sentences. These encoders are typically implemented as convolutional [Kim, 2014], recursive [Socher et al., 2013], or recurrent neural networks [Mikolov et al., 2010] operating over a sentence's words or characters [Zhang et al., 2015b, Kim et al., 2016]. Most of the early successes with sentence encoder-based models have been on tasks with ample training data, where it has been possible to train the encoders in a fully-supervised end-to-end setting. However, recent work has shown some success in using unsupervised pre-training with unlabeled data to both improve the performance of these methods and extend them to lower-resource settings [Dai and Le, 2015, Kiros et al., 2015, Bajgar et al., 2016, Hill et al., 2016].

This Chapter presents a set of methods for unsupervised pre-training that train sentence encoders to recognize *discourse coherence*. When reading text, human readers have an expectation of coherence from one sentence to the next. In most cases, for example, each sentence in a text should be both interpretable in context and relevant to the topic under discussion. Both of these properties depend on an understanding of the

local context, which includes both general knowledge about the state of the world and the specific meanings of previous sentences in the text. Thus, a model that is successfully trained to recognize discourse coherence must be able to understand the meanings of sentences as well as relate them to key pieces of knowledge about the world.

[Hobbs, 1979] presents a formal treatment of this phenomenon. He argues that for a discourse (here, a text) to be interpreted as coherent, any two adjacent sentences must be related by one of a few set kinds of *coherence relations*. For example, a sentence might be followed by another that elaborates on it, parallels it, or contrasts with it. While this treatment may not be adequate to cover the full complexity of language understanding, it allows Hobbs to show how identifying such relations depends upon sentence understanding, co-reference resolution, and commonsense reasoning.

Recently proposed techniques [Kiros et al., 2015, Ramachandran et al., 2016] succeed in exploiting discourse coherence information of this kind to train sentence encoders, but rely on generative objectives which require models to compute the likelihood of each word in a sentence at training time, thus suffering from the problems of language modeling outlined in earlier Chapters (speed, label bias,...). In this work, we propose alternative objectives which exploit much of the same coherence information in a more direct fashion. In particular, we propose three coherence-based pre-training tasks, show that they can be used together effectively in multitask training (Figure 5.1), and evaluate models trained in this setting on the training tasks themselves and on standard text classification tasks. We find that our approach makes it possible to learn to extract broadly useful sentence representations.

Related Work This approach is inspired most directly by the SkipThought approach of [Kiros et al., 2015], which introduces the use of paragraph-level discourse informa-

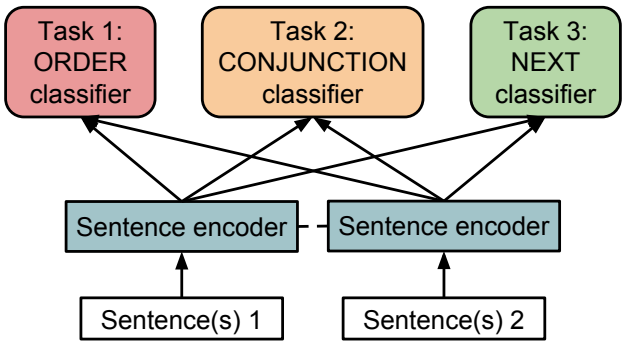

Figure 5.1: We train a sentence encoder (shown as two copies with shared parameters) on three discourse-based objectives over unlabeled text.

tion for the unsupervised pre-training of sentence encoders. Since that work, three other papers have presented improvements to this method (the SDAE of [Hill et al., 2016], also [Gan et al., 2016, Ramachandran et al., 2016]). These improved methods are based on techniques and goals that are similar to ours, but all three rely on conditional language modeling as an objective, which implies specific biases and a consequent computational cost.

In closely related work, [Logeswaran et al., 2016] present a model that learns to order the sentences of a paragraph. While they focus on learning to assess coherence, they show positive results on measuring sentence similarity using their trained encoder. Alternately, the FastSent model of [Hill et al., 2016] is designed to work dramatically more quickly than systems like SkipThought, but in service of this goal the standard sentence encoder RNN is replaced with a low-capacity CBOW model. Their method does well on existing semantic textual similarity benchmarks, but its insensitivity to order places an upper bound on its performance in more intensive extrinsic language understanding tasks.

Looking beyond work on unsupervised pre-training: [Li and Hovy, 2014] and [Li

Sentence Pair	Label	Relation
A strong one at that. *Then I became a woman.*	FLIPPED	elaboration
I saw flowers on the ground. *I heard birds in the trees.*	ORDERED	list
It limped closer at a slow pace. *Soon it stopped in front of us.*	ORDERED	spatial
I kill Ben, you leave by yourself. *I kill your uncle, you join Ben.*	FLIPPED	time

Table 5.1: The binary ORDER objective. Discourse relation labels are provided for the reader, but are not available to the model.

and Jurafsky, 2016] use representation learning systems to directly model the problem of sentence order recovery, but focus primarily on intrinsic evaluation rather than transfer. [Wang and Cho, 2016] train sentence representations for use as context in language modeling. In addition, [Ji et al., 2016] treat discourse relations between sentences as latent variables and show that this yields improvements in language modeling in an extension of the document-context model of [Ji et al., 2015].

Outside the context of representation learning, there has been a good deal of work in NLP on discourse coherence, and on the particular tasks of sentence ordering and coherence scoring. [Barzilay and Lapata, 2005] provide thorough coverage of this work.

5.3 Discourse Inspired Objectives

In this work, we propose three objective functions to be used over paragraphs extracted from unlabeled text. Each captures a different aspect of discourse coherence and together the three can be used to train a single encoder to extract broadly useful sentence representations.

Context
No, not really. *I had some ideas, some plans.* *But I never even caught sight of them.*
Candidate Successors
1. *There's nothing I can do that compares to that.* 2. *Then one day Mister Edwards saw me.* **3. *I drank and that was about all I did.*** 4. *And anyway, God's getting his revenge now.* 5. *He offered me a job and somewhere to sleep.*

Table 5.2: The NEXT objective.

Binary Ordering of Sentences Many coherence relations have an inherent direction. For example, if S_1 is an elaboration of S_0, S_0 is not generally an elaboration of S_1. Thus, being able to identify these coherence relations implies an ability to recover the original order of the sentences. Our first task, which we call ORDER, consists in taking pairs of adjacent sentences from text data, switching their order with probability 0.5, and training a model to decide whether they have been switched. Table 5.1 provides some examples of this task, along with the kind of coherence relation that we assume to be involved. It should be noted that since some of these relations are unordered, it is not always possible to recover the original order based on discourse coherence alone (see *e.g.* the *flowers* / *birds* example).

Next Sentence Many coherence relations are transitive by nature, so that any two sentences from the same paragraph will exhibit some coherence. However, two adjacent sentences will generally be more coherent than two more distant ones. This leads us to formulate the NEXT task: given the first three sentences of a paragraph and a set of five candidate sentences from later in the paragraph, the model must decide which candidate immediately follows the initial three in the source text. Table 5.2 presents an example of such a task: candidates 2 and 3 are coherent with the third sentence of the paragraph,

Sentence Pair	Label
He had a point. *For good measure, I pouted.*	RETURN *(Still)*
It doesn't hurt at all. *It's exhilarating.*	STRENGTHEN *(In fact)*
The waterwheel hammered on. *There was silence.*	CONTRAST *(Otherwise)*

Table 5.3: The CONJUNCTION objective. Discourse relation labels are shown with the text from which they were derived.

addition		**contrast**		**time**
again	furthermore	anyway	contrarily	meanwhile
also	moreover	however	conversely	next
besides	in addition	instead	nonetheless	then
finally		nevertheless	in contrast	now
further		otherwise	rather	thereafter
result	**specific**	**compare**	**strengthen**	**return**
accordingly	namely	likewise	indeed	still
consequently	specifically	similarly	in fact	
hence	notably			**recognize**
thus	that is			undoubtedly
therefore	for example			certainly

Table 5.4: List of conjunctions.

but the elaboration (3) takes precedence over the progression (2).

Conjunction Prediction Finally, information about the coherence relation between two sentences is sometimes apparent in the text [Miltsakaki et al., 2004]: this is the case whenever the second sentence starts with a conjunction phrase. To form the CONJUNCTION objective, we create a list of conjunction phrases (see Table 5.4), then extract from our source text all pairs of sentences where the second starts with one of the listed conjunctions, give the system the pair without the phrase, and train it to recover the conjunction category. Table 5.3 provides examples.

5.4 Experiments

In this Section, we introduce our training data and methods, present qualitative results and comparisons among our three objectives, and close with quantitative comparisons with related work.

Data Creation Despite having recently become a standard dataset for unsupervised learning, the BookCorpus presented in [Zhu et al., 2015] does not exhibit sufficiently rich discourse structure to allow our model to fully succeed—in particular, some of the conjunction categories are severely under-represented. While this precludes a strict apples-to-apples comparison with other published results, our goal in extrinsic evaluation is simply to show that our method makes it possible to learn useful representations quickly, rather than to demonstrate the superiority of our learning technique given fixed data and unlimited time. Instead, we train on system on text from Wikipedia and the Gutenberg project [Stroube, 2003].

We collected the latest Wikipedia dump on 03/29/2017, and downloaded all of the English language books from the Gutenberg project. After sentence and word tokenization with NLTK [Bird, 2006] and lower-casing, we identify all paragraphs longer than 8 sentences and extract a NEXT example from each, as well as pairs of sentences for the ORDER and CONJUNCTION tasks. A CONJUNCTION example is a pair of adjoining sentences where the second starts with one of the conjunction phrases listed in Table 5.4 followed by a comma. Additionally, if the first sentence starts with a conjunction phrase, we remove it. So for example: "Moreover, it was quite a good meal. However, the wine was sub par." becomes ("it was quite a good meal.", "the wine was sub par.", "however"). This gives us 104M examples for ORDER, 1.6M for CONJUNCTION, and 5.4M for NEXT.

	CONJ	ORDER	NEXT
BiGRU single C / O / N	38.9	62.3	33.0
BiGRU joint NEXT + CONJ	38.1	-	**36.1**
BiGRU joint ORDER + CONJ	**39.1**	**63.0**	-
BiGRU joint ALL	**39.1**	60.9	35.6
GRU joint ALL	37.0	58.4	32.7
CBOW joint ALL	33.3	57.4	28.5

Table 5.5: Intrinsic evaluation results.

Intrinsic Evaluation In order to explore the properties of the proposed tasks, we first consider the following three sentence encoding models: a simple 2048D sum-of-words (CBOW) encoding, a 2048D GRU recurrent neural network [Cho et al., 2014b], and a 2×1024D bi-directional GRU. For these last two, following [Conneau et al., 2017], we use max pooling over the recurrent hidden states to obtain the final encoding. Each of the discourse tasks uses a bi-linear classifier which takes as input the dimension D embeddings of the left and the right sentences $\mathbf{s}_l$ and $\mathbf{s}_r$, and computes a score for each class. For the ORDER task, we have a matrix $\mathbf{M}^{order} \in \mathbb{R}^{D \times D}$, and we optimize the cross-entropy of the likelihood that the sentences have been flipped as:

$$p(flipped|\mathbf{s}_l, \mathbf{s}_r) = \sigma(\mathbf{s}_l^T \mathbf{M}^{order} \mathbf{s}_r) \tag{5.1}$$

For NEXT, we concatenate the representations of the three context sentences into a larger context vector $\mathbf{s}_c$, use a projection matrix $\mathbf{M}^{next} \in \mathbb{R}^{D \times D}$, and compute a softmax over the scores for each of the embeddings of the proposals $(\mathbf{s}_{p_1}, \ldots, \mathbf{s}_{p_5})$:

$$p(i|\mathbf{s}_c, \mathbf{s}_p) = \frac{\exp(\mathbf{s}_c^T \mathbf{M}^{next} \mathbf{s}_{p_i})}{\sum_{j=1}^{5} \exp(\mathbf{s}_c^T \mathbf{M}^{next} \mathbf{s}_{p_j})} \tag{5.2}$$

Finally, the CONJ classifier has a matrix $\mathbf{M}_c^{conj} \in \mathbb{R}^{D \times D}$ for each conjunction c, and computes the probability as:

$$p(c|\mathbf{s}_l, \mathbf{s}_r) = \frac{\exp(\mathbf{s}_l^T \mathbf{M}_c^{conj} \mathbf{s}_r)}{\sum_{c'} \exp(\mathbf{s}_l^T \mathbf{M}_{c'}^{conj} \mathbf{s}_r)} \tag{5.3}$$

We train separate models on each of the tasks, and also perform joint training on NEXT + CONJUNCTION, ORDER + CONJUNCTION, and ALL three tasks. All our models are trained for 24 hours, which allows for several epochs on the full CONJUNCTION data (2 to 5 depending on the setting).

Table 5.5 compares the performance of different training regimes on a separate validation set along two axes: encoder architecture and whether we train one model per task or one joint model. As expected, the more complex bidirectional GRU architecture is required to capture the appropriate sentence properties. The results of joint training also inform us about the relation between the difference discourse tasks. Both the ORDER and NEXT tasks benefit from being trained jointly with CONJUNCTION, and the latter reaches its best performance when trained with either ORDER or with both others. Early experiments on the external evaluation also show that the joint BiGRU model substantially outperforms each single model.

Extrinsic Evaluation We evaluate the quality of the encoder learned by our system, which we call DiscSent, by using the sentence representations it produces in a variety of sentence classification tasks. We follow the setting and use the evaluation code of [Conneau et al., 2017] (SentEval). For this application, we use the BiGRU models trained on all our tasks separately (the ORDER, NEXT and CONJ lines in Table 5.6), as well as on all three tasks jointly (the DiscSent lines.) We are specifically interested in measuring whether our model is able to capture information beyond that of a Bag-

Model	Avg.	MR	CR	SUBJ	MPQA	SST-b	TREC	S-R	S-E	MRPC	STS14
GloVe[1]	0.790	78.7	78.5	91.6	87.6	79.8	83.6	0.800	78.6	72.1/80.9	0.54/0.56
ST[1]	0.789	76.5	80.1	93.6	87.1	82.0	**92.2**	0.858	82.3	73.0/82.0	0.29/0.35
InferSent[1]	**0.845**	81.1	**86.3**	92.4	**90.2**	**84.6**	88.2	**0.884**	**86.3**	**76.2/83.1**	**0.70/0.67**
GloVe(ours)	0.787	77.0	78.2	91.3	87.8	80.2	82.4	0.801	78.4	70.0/80.8	0.55/0.56
ST-3M	0.737	72.9	78.5	91.6	80.5	64.6	70.6	0.785	80.6	71.1/80.1	0.45/0.44
ST-3M+G	0.791	80.9	83.6	**94.1**	89.3	71.5	80.0	0.838	82.0	71.4/79.8	0.51/0.50
ST-16M	0.757	73.7	80.4	92.8	82.8	68.7	78.2	0.805	81.0	70.4/80.5	0.45/0.44
ST-16M+G	0.800	79.8	84.0	93.9	88.9	76.4	84.8	0.841	82.9	72.1/80.1	0.50/0.48
ORDER+G[2]	0.800	80.8	84.1	93.4	89.4	75.0	85.8	0.827	80.4	71.1/81.0	0.52/0.53
NEXT+G[2]	0.792	78.6	79.7	91.3	87.4	81.9	83.8	0.805	78.7	70.0/81.0	0.54/0.56
CONJ+G[2]	0.814	**81.9**	85.5	93.5	89.6	80.7	85.8	0.841	81.0	71.5/81.3	0.55/0.56
DiscSent	0.775	73.5	79.0	90.6	81.4	74.1	85.2	0.804	83.3	71.5/80.5	0.51/0.52
DiscSent+G[2]	0.817	81.5	84.7	93.5	89.3	82.3	87.6	0.838	81.1	73.2/81.4	0.56/0.56

Table 5.6: Text classification results, comparing our system to the SkipThought and InferSent sentence encoders. The best result in each column is bolded, the second best is underlined. ST is the SkipThought model of [Kiros et al., 2015], ST-3M and ST-16M are our implementation of the model trained on 3 million examples and 16 million examples drawn from our data respectively. [1] Results were taken from [Conneau et al., 2017]. [2]+G lines concatenate the sum of the GloVe embeddings for the words to the output of the sentence encoder at evaluation time [Pennington et al., 2014].

of-Words approach, and so concatenate the sum of the GloVe embeddings [Pennington et al., 2014] for the words in a sentence to the output of our encoders. In order to better compare to the work of [Kiros et al., 2015], we also trained a SkipThought model with the same encoder on our data. This setting was trained on 3 million sentences in 24 hours. Using the hyper-parameters found in this setting, we also trained a model for five days on 16 million examples.

Table 5.6 presents the transfer learning results obtained with SentEval for all of these settings. We notice that while our model still lags behind the InferSent encoder, it does better than SkipThought trained on our data on average, even when the latter is given significantly more time. We also notice that of all our tasks, CONJUNCTION seems to be the most useful when taken by itself, and combining a model trained on it with a sum-of-word representation using the GloVe embeddings out-performs [Kiros et al., 2015]. However, there is still something to be gained by combining all three discourse

objectives, and DiscSent + GloVe embeddings reaches the best average score among unsupervised systems.

Experimental Details We learn all of our models through stochastic gradient descent with AdaGrad [Duchi et al., 2010], subsampling NEXT by a factor of 4 whenever it is trained jointly with another task (each NEXT example encodes 8 sentences, compared to 2 for the ORDER and CONJUNCTION tasks). We ran a hyper-parameter search using the same grid for all parameters, with a learning rate in $\{2e-3, 5e-3, 2e-2\}$. All parameters are initialized uniformly at random in a $(-x, x)$ range, with $x \in \{0.1, 0.5, 1\}$. We tried combining the recurrent states of the GRUs with a sum and max pooling function. We use an input vocabulary the 100K most common words in our data and learn the word embeddings from scratch. For our version of Skip Thought [Kiros et al., 2015], the decoder is a one-directional GRU recurrent network with a 1024 dimensional hidden state, and the output vocabulary is limited to the 20K most frequent words. For each of the models, we performed the transfer experiments by using the model which has the best validation score on the training task (ORDER, NEXT or CONJUNCTION accuracy or average thereof for our tasks, perplexity for Skip Thought).

5.5 Discussion: Further Work on Discourse

In this Chapter, we proposed to look beyond language modeling for unsupervised training of text understanding models. We described a variety of alternative objectives from the literature, and introduced three new tasks based on the notion of discourse coherence. We then used them to train a sentence representation system which we showed can be of use in a variety of language tasks.

Our approach opens the way to a number of follow-ups. First, there is some room

to improve accuracy on the intrinsic evaluations; leads to achieve such an improvement include using larger sentence representation dimensions, taking advantage of character level inputs as suggested in Chapter 2, or exploring different encoder architectures.

An alternative view of the proposed objectives would see them as a way to explore and evaluate different architectures and sentence encoders trained in other settings: what models can capture the structure necessary for modeling discourse? How do representations learned on the skip-thought objective or on Natural Language Inference fare when transferred to our tasks with minimal fine-tuning? Given the difficulty of automatically and reliably evaluating the general usefulness of sentence encoders, this could provide critical information.

Finally, another notable feature of the present work is the simultaneous use of several objectives with different properties. However, we did this in a trivial way, by simply alternating batches during stochastic gradient descent. While this works reasonably well, it is likely that we are not making the best of our data this way. Better ways if combining objectives might also allow us to train models which also learn from language modeling, and take advantage of the advances described in Chapter 3 and 4. Recent works such as [Kirkpatrick et al., 2016] have started looking into this problem of learning from numerous tasks in a synergistic way, but there are still many open questions in this area.

5.6 Towards Formalizing Meaning for NLP Tasks

We have presented our goal for this thesis as learning to represent the meaning of a text in a usable way. However, we have spent little time so far reflecting on what exactly it is that we understand by the notion of *meaning*: after pointing out the difficulties of scaling systems which rely on a model theoretic representation in Chapter 1, we implied

that the concept of meaning was related to the notion of conditioning on representations of text in language modeling and other common NLP tasks and proceeded to propose ways to better learn those representations in the following Chapters. While we believe this approach to constitute a valid strategy, which can still lead to much progress in the field, we also strongly believe in the usefulness of formalizing the concept in a way that can be more systematically analyzed and reasoned over. Note that our priorities here are slightly different from those of a semanticist or philosopher, two disciplines which have considered this question at length: specifically, we are looking for a formalism which is wide enough to cover all or most of written language, but which can also be described using a system that remains tractable for a machine, and has a natural link to the neural network approaches which have become ubiquitous in recent years.

In this Section, we attempt to sketch such a formalism by drawing from existing theories of semantics while keeping an eye on their tractability and relation to modern practices in the field of natural language processing. More specifically, we consider two salient properties of neural network methods. On the one hand, a common argument for the use of neural networks is that they constitute "universal function approximators" [Hornik et al., 1989, Zhang et al., 2016a]. This allows machine learning systems based on them to learn conditional probability functions or general predictors without having to *e.g.* define domain-specific representation schemes, as long as enough examples of input/output pairs are provided. Similarly, we would like our representation of meaning to be defined through general functions acting on text, without relying on extrinsically defined entities. On the other hand, neural network functions are continuous in nature: they rely on gradient descent to learn parameters and intermediary distributed representations to optimize an objective. While it is unclear whether gradient descent or other forms of local search in a continuous search are a necessary part of learning

to represent meaning, reasoning over continuous rather than discrete objects does have a number of advantages, such as an inherent notion of ranking or natural handling of uncertainty. These prompt us to look for a formalism of meaning using objects defined in continuous spaces whose algebraic properties correspond to similar notions.

Possible Worlds Semantics. In order to arrive at a definition of meaning which meets both of the above stated requirements, we look to the notion of Possible World Semantics [Carnap, 1947, Kripke, 1963, Lewis, 1986]. Richard Montague argues that "The basic aim of semantics is to characterize the notion of a true sentence (under a given interpretation) and of entailment" [Montague, 1970]. This definition lead to the development of a semantics based on first order logic which has proven useful in studying and describing linguistic phenomena (see *e.g.* [Montague, 1973]). However, this formalism does not naturally handle phenomena such as uncertainty, hedging,... The introduction of modal logic [Lewis and Langford, 1959, Hughes and Cresswell, 1968, Hughes and Cresswell, 1996] aimed at solving some of these issues, by allowing to also characterize the notion of a *possibly* or *necessarily* true sentence. Unfortunately, contrary to propositional logic, validity in modal logic cannot be defined through the use of truth tables: Possible Worlds Semantics provide a way to remedy this difficulty.

Possible Worlds Semantics start with a base universe or frame $\Omega^{\emptyset} = \{\mathcal{W}\}$ representing all possible states of the world, and restricts this set every time more information becomes available in the form of a sentence or statement. More formally, let us define a universe as a set of world states: $\Omega = \{\mathcal{W}\}$, and an evaluation function $eval$, and let us define a world state $\mathcal{W}$ by the set of assertions $(\bar{s})$ which are true in $\mathcal{W}$, that is $(eval(\bar{s}, \mathcal{W}) = 1)$, or false in $\mathcal{W}$, that is $(eval(\bar{s}, \mathcal{W}) = -1)$. The evaluation function $eval$ can be trivially extended to work over universes: a statement is true in a universe

Ω if and only if it is true in all world states $\mathcal{W} \in \Omega$. That is:

$$(eval(\bar{s}, \Omega) = 1) \Leftrightarrow (\forall \mathcal{W} \in \Omega, eval(\bar{s}, \mathcal{W}) = 1)$$

Similarly, a statement is false in a universe $(eval(\bar{s}, \Omega) = -1)$ if and only if it is false in all of its world states. Finally, we add a third value for the remaining cases: if $(\bar{s})$ is true in some of the world states and false in others, then it is considered to be *possible*: $(eval(\bar{s}, \Omega) = 0)$.

As mentioned above, we start with a universe of all possible world states: that is, any statement which is neither self-contradictory nor a tautology is possible. The next step is then to define a reading function which updates a universe after reading a sentence: $read(\bar{s}, \Omega) = \Omega'$. This function needs to have certain properties. For example, reading a statement should either give us new information or reiterate information which is already known, and thus cannot increase the number of possible world states: $read(\bar{s}, \Omega) \subseteq \Omega$. Additionally, after reading a statement from a trusted source, it should be considered as true in the new universe (for a more complete discussion of the *De Re/De Dicto* distinction, see *e.g.* [Nortmann, 2002]):

$$\forall \bar{s}, \quad eval(\bar{s}, read(\bar{s}, \Omega)) = 1$$

Additionally, note that the $read$ function needs to be able to change the state of the universe even when applied to a sentence which is already known to be true; this is necessary in order to handle self-referential text, including pronouns. Consider for example a universe where we have $eval(\text{“The sky is blue.”}, \Omega) = 1$. Then, let

$$\Omega_1 = read(\text{“The sun is bright.”}, \Omega)$$

We still have $eval(\text{“The sky is blue.”}, \Omega_1) = 1$. Now consider the sentence “This is caused by the diffraction of light in the atmosphere.” The truth value of this statement in Ω_1 is 0, since there is no direct relation between the brightness of the sun and diffraction. However, reading “The sky is blue.” again, even though it does not change our belief about the state of the universe, changes this $eval$ to 1.

The formalism presented so far meets the first of our requirements: the meaning of a sentence can be defined through the use of general functions which take text as input, namely $eval$ and $read$, and corresponds to the restriction induced by said sentence over an implicitly defined set of possible worlds (universe). Some important questions remain open: for example, where *common knowledge* or *common sense reasoning* fits in this definition: what exactly is the set of all possible world states $\Omega^{\emptyset}$? Does it contain world states where words have different relations (e.g., a truck is a kind of food, blue is the opposite of small)? These questions are inherently tied to the problem of defining the set of possible worlds, which has been the focus of a significnt amount of work in the field of philosophy (see *e.g.* the abstractionist approach of [Plantinga, 1976], combinatorial definition of [Armstrong, 1986], or concretism [Lewis, 1986]), and goes beyond the scope of this thesis. However, from a practical point of vies, one possible answer would be to start by applying the *read* function to e.g. a dictionary or encyclopedia, or to extend *read* to restrict possible world states based on physical evidence (this brings us in turn to the question of whether language needs to be grounded in the physical world or not, which we will not delve into in this work).

Continuous Output and Interpretability. Although it differentiates between *possibly true* and *necessarily true* statements, the current discrete-valued version of the $eval$ function still fails to account for how likely a *possible* statement is. To that end, we

could find it useful for $eval(\bar{s}, \Omega)$ to have values in $[-1, 1]$ rather than $\{-1, 0, 1\}$: the output should still be the same for statements which are *necessarily false* or *necessarily true* (-1 and 1 respectively), and would otherwise depends on the proportions of worlds in the current universe $\mathcal{W} \in \Omega$ for which $eval(\bar{s}, \mathcal{W}) = 1$. A more precise definition would require having access to a measure over the infinite (and uncountable) set of possible worlds, but we can still describe some properties of such a function. For example:

$$\{\mathcal{W} \in \Omega; eval(\bar{s}_a, \mathcal{W}) =\} \subseteq \{\mathcal{W} \in \Omega; eval(\bar{s}_b, \mathcal{W}) =\} \Rightarrow eval(\bar{s}_a, \Omega) \leq eval(\bar{s}_b, \Omega)$$

In other words, if $\bar{s}_a$ implies $\bar{s}_b$, then $\bar{s}_a$ is less likely than $\bar{s}_b$ according to the $eval$ function.

Additionally, we have only considered reasoning over "statements" or "sentences" so far, always assuming that the referred sequences of words were interpretable (*i.e.* correspond to a computable meaning). However, both the $read$ and $eval$ function might be applied to arbitrary sequences of words or characters, especially when considering systems involving automated language generation. Let us start by considering how the $eval$ function should handle such a sequence. We choose to add a special output value μ^η to the world-state-level function, which now has values in $\{-1, 1, \mu^\eta\}$. Note that a sequence might be interpretable in one world state, and not in another: for example, a sequence like *"A tried greenly blue."* is non-interpretable in any universe for syntactic reasons, while *"The purple cat sat on the mat."* would be interpretable in a world state which has a purple cat but non-interpretable in one which does not. Now, let us consider the universe-level $eval$ function. If a sequence $\bar{s}$ is interpretable in all world states, then $eval(\bar{s}, \Omega)$ should behave as described previously. If the sequence is non-interpretable in all world states, then we can simply have $eval(\bar{s}, \Omega) = \mu^\eta$. In all other

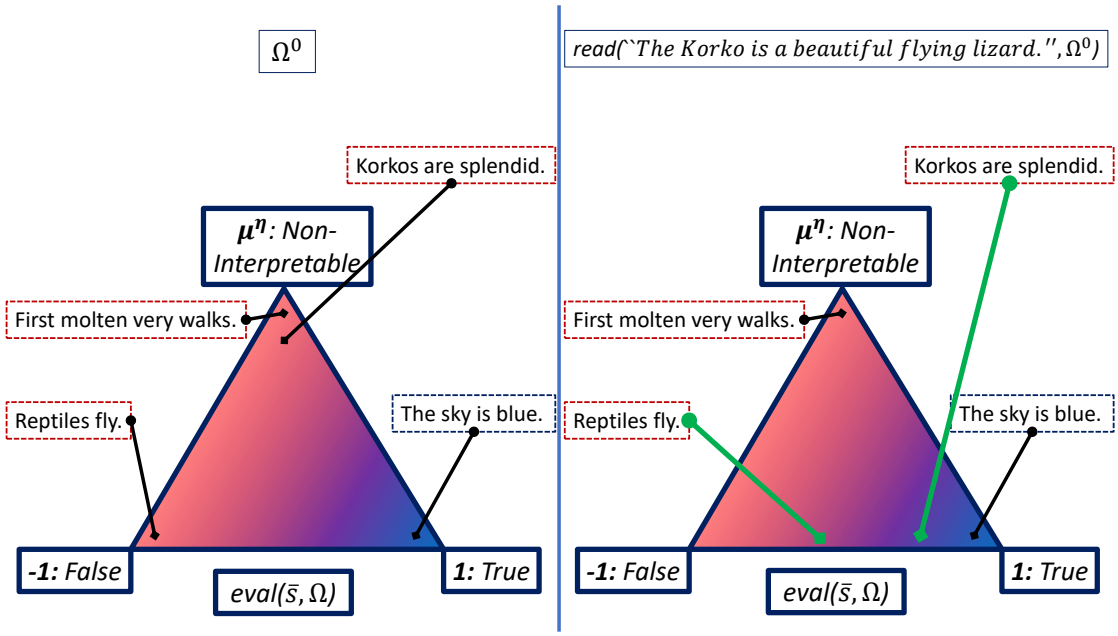

Figure 5.2: Representing the *eval* function in two universes. **Left: base universe.** The base universe Ω^0 corresponds to general knowledge about the world. In this universe, the output of $eval$ for sentences *"The sky is blue."*, *"Lizards can fly."* and *"First molted very walks."* is close to the corners corresponding to *true*, *false* and *non-interpretable* respectively, the first two because of common knowledge about the world, the third for being a-grammatical. As for the sentence *"Korkos are splendid."*, the base universe has no knowledge of what a *Korko* is, and so the sentence is *mostly non-interpretable*. **Right: universe after reading the sentence** *"The Korko is a beautiful flying lizard."*. *"The sky is blue."* is still evaluated as *true*, and *"First molted very walks."* as *non-interpretable*. However, now that we know what a *Korko* is, *"Korkos are splendid."* becomes not only *interpretable*, but also likely *true*. Similarly, now that there is evidence of at least one reptile flying (common knowledge tells us that a lizard is a reptile), *"Lizards can fly."* moves away from *false* and closer to possible. The colors simply illustrates how close to both interpretable and true a sentence is, with blue corresponding to higher scores and red to lower scores.

cases, $eval(\bar{s}, \Omega) = \mu^\eta$ will have a value corresponding to the center of gravity in the $-1, \mu^\eta, 1$ weighted according to the proportion of worlds in the current universe $\mathcal{W} \in \Omega$ for which $eval(\bar{s}, \mathcal{W}) = -1$, $eval(\bar{s}, \mathcal{W}) = \mu^\eta$, and $eval(\bar{s}, \mathcal{W}) = 1$ (computing these weights would rely on the measure mentioned in the previous paragraph). Figure 5.2 illustrates this *eval* function in two different universes. As for the $read$ function, its

role remains the same: even a non immediately interpretable sentence can bring some information, by setting up the state of the universe for *e.g.* when a previously unknown word is defined.

By extending the $eval$ function in this way, we meet our second requirement: by defining the output of our fundamental functions in a continuous space, we have access to ranking judgments (more or less likely to be true, more or less interpretable). Note that we could easily imagine adding more vertices to the output space of $eval$. For example, a vertex υ^{s+} could correspond to sentiment analysis, where $eval(\bar{s}, \mathcal{W}) = \upsilon^{s+}$ for a world $\mathcal{W} \in \Omega$ if $\bar{s}$ expresses a positive sentiment, $eval(\bar{s}, \Omega)$ would then have values in the $\{-1, \mu^{\eta}, \upsilon^{s+}, 1\}$ simplex. We could also have different non-interpretability vertices $(\mu^{\eta_1}, \mu^{\eta_2}, \ldots)$ for a-grammatical sentences, sentences with unknown words, unresolved references, etc... For ease of exposition, we simply use the $\{-1, \mu^{\eta}, 1\}$ simplex in the rest of this Chapter.

Now, let us consider how this formalism can help us define a notion of contextual similarity between the meaning of two sentences. First, we will need to define a distance measure between universes, which are implicitly defined uncountable sets, or at least a way to describe how similar any two universes are to each other. We propose the following: assuming that we have a probability distribution over sentences $p_{\bar{s}}$, we define the distance between two universes Ω^a and Ω^b as:

$$\Delta(\Omega^a|\Omega^b) = \mathbb{E}_{\bar{s}\sim p_{\bar{s}}}[||eval(\bar{s}, \Omega^a) = eval(\bar{s}, \Omega^b)||_2] \tag{5.4}$$

That is, two universes are close to each other if the same statements tend to be true or false in both. Then, we can define the following distance function between the meaning

of two sentences $\bar{s}_a$ and $\bar{s}_b$ in a universe Ω:

$$\delta(\bar{s}_a, \bar{s}_b; \Omega) = \Delta(read(\bar{s}_a, \Omega)|read(\bar{s}_b, \Omega)) \tag{5.5}$$

Note that the notion of distance presented here is *contextual*: for example the sentences "Ernesto saw Francis." and "Francis visited his doctor." will be close to each other in a universe where we know that Francis' doctor is named Ernesto, and far if we know that is not the case. This is quite different from the definition of Semantic Textual Similarity used in works such as [Agirre et al., 2012]. An absolute distance metric could be obtained by either fixing the universe to some Ω^0, or more generally by providing some distribution p^Ω over all possible universes:

$$\delta^0(\bar{s}_a, \bar{s}_b) = \mathbb{E}_{\Omega \sim p^\Omega}[\Delta(read(\bar{s}_a, \Omega)|read(\bar{s}_b, \Omega))] \tag{5.6}$$

Relation to NLP tasks. Now, let us consider how some natural language tasks, such as machine translation [Brown et al., 1990], summarization [Kupiec et al., 1995], or Natural Language Inference [MacCartney, 2009, Bowman et al., 2015] can be described in our framework. In the rest of this Section, the universe Ω^0 corresponds to the context for a task. This would represent at the minimum the set of possible world states which correspond to *common knowledge* (after having *read*, for example, that the sky is blue, or that a container needs to be larger than what it contains...), and can be further refined by previously read text or task-specific additional information.

Then, given a source and a target language, let us assume that we have two $read$ functions, one for the source $read_S$ and one for the target $read_T$, We can define machine translation as the task of finding the sentence in the target language $\bar{s}_T$ which has the closest meaning to the source sentence $\bar{s}_S$ in context, that is, such that evaluating each

of them in the same context leads to universes which are as close as possible:

$$\bar{s}_T = \arg\min_{\bar{s}} \Delta(read_S(\bar{s}_S, \Omega^0) | read_T(\bar{s}, \Omega^0)) \tag{5.7}$$

Similarly, summarization would consist in finding the sentence or text with the closest meaning to document $\bar{d}$ under a length constraint (e.g. fewer than L word):

$$\bar{s}^* = \arg\min_{|\bar{s}|<L} \Delta(read(\bar{d}, \Omega^0) | read(\bar{s}, \Omega^0)) \tag{5.8}$$

Finally, Natural Language Inference considers two statements $\bar{s}_a$ and $\bar{s}_b$, and, considers whether $\bar{s}_a$ implies $\bar{s}_b$, $\bar{s}_b$ contradicts $\bar{s}_a$ or $\bar{s}_a$ and $\bar{s}_b$ lack a logical relations. In the framework we just presented, this might correspond to considering a base universe Ω^0, and deciding whether $(eval(\bar{s}_b, read(\bar{s}_a, \Omega^0)) = 1)$, $(eval(\bar{s}_b, read(\bar{s}_a, \Omega^0)) = -1)$, or $(eval(\bar{s}_b, read(\bar{s}_a, \Omega^0)) \approx 0)$ respectively.

There are other constraints obviously. For example, we want to make sure summarization does not give any information which was not in the initial document (hence $read(\bar{d}, \Omega^0) \subset read(\bar{s}^*, \Omega^0)$), but this shows how the formalism we introduce in this Section can help provide a framework for some language understanding tasks.

Conclusion and Future Work In this Section, we have sketched a formalism to describe the meaning of a text in terms of some general functions $read$ and $eval$, and shown the relation between some common NLP tasks and compositions these functions. This implies that we can use machine learning techniques to train universal function approximators, such as kernel-based functions or neural networks, which have gained in popularity recently, to learn these compositions. Such approximators produce two objects of interest. They are usually *parameterized* in some way, and these parameters

implicitly encode the base universe Ω^0 mentioned previously. They also often work with *intermediary representations* of words or word sequences. Given the requirements of the tasks we have presented in this Section, such intermediary representations need to encode (or at least index in a way which makes computing the $\arg\min$ involved more tractable) all of the information necessary to perform the $eval$ and $read$ operations: from what we have seen so far, this corresponds to encoding the *meaning* of the text. Thus, we have the beginning of an answer to the question of what we understand by the notion of meaning: it can be accessed by learning a set of parameters and finding families of encoding or indexing models which can implicitly represent a universe, as well as implement the $eval$, $read$ and Δ functions. We hope that by developing this formalism further and answering some of the questions which we have left open, we can characterize the objects involved and their relations to their approximators in a way which leads useful linguistic insights and guides the development of new models.

Conclusion

In this thesis, we have considered the problem of combining sequences of characters or words into a representation of the meaning expressed in text. We put forward that the task of language modeling relied on a conditioning step over a representation of a text's meaning, and so proposed to use it as a general objective to discover model architectures and train systems which may be of use for general text understanding.

In Chapter 2, we raised the question of what the appropriate reading level was for text between characters and words. Sections 2.2 and 2.3 showed a clear advantage to at the very least using character information to compute token embeddings in the context of word level language modeling. Then, noting that fully character level sequence modeling still lags behind word level processing in terms of computational efficiency and long term memory, we proposed a Variable Computation Recurrent Network architecture for the purpose of processing sequences of characters which learns to adapt its operations in a way which starts bridging this gap.

Chapters 3 and 4 focused on the language modeling objective itself. In Chapter 3, we considered the challenge of scale caused by using a family of distributions which needs to compute a normalization constant over a large vocabulary for every token of a corpus. We proposed to remedy this difficulty by resorting to hierarchical prediction, and innovated over previous work by providing a theoretically grounded algorithm to

efficiently learn the structure of the hierarchy, in an online fashion and jointly with the other parameters of the model, showing gains in both speed and performance. Then, Chapter 4 argued for the use of global (sequence-level) rather than local (word-level) normalization, proposed a neural network inspired Markov Random Field structured language model, and provided a new and efficient algorithm to approximate its partition function in time mostly independent of the corpus size.

Finally, in Chapter 5, we went back to our initial definition of meaning, and considered other objectives beyond language modeling to train text understanding systems. We found that some discriminative objectives inspired by a notion of discourse coherence presented similar advantages to language modeling, such as having access to vast amounts of naturally occurring training data, while being much cheaper to compute. We used these objectives to train sentence representation systems, and showed that the learned encodings were practically useful. We then sketched a formalism of meaning based on Possible Worlds Semantics which we believe might be helpful in reasoning over the properties of language understanding systems in the future.

What comes next? First, we raised a number of immediate follow up questions in the discussion Sections of Chapters 2 to 5. Indeed, by and large, the algorithms and paradigms introduced in this thesis should be seen as stepping stones or building blocks towards reaching their stated purpose. While it is becoming increasingly obvious that character level information is useful, finding ways to limit the additional computational cost incurred by doing so and finding the right trade-off between character and word level processing for different tasks remains an open problem. The principled tree learning algorithm we introduced to speed up word level language modeling would benefit from being extended to more general structures, such as prediction forests, and from being adapted to other application settings. While we took a first step towards being able to

perform marginal inference in globally normalized language models, there is still much work to be done to make more expressive distributions tractable, and extending the approach to *maximum a posteriori* inference could have a significant impact on many areas of language processing. Finally, we are barely just starting to explore alternative unsupervised training objectives for text understanding beyond language modeling, and to consider their potential to both train and evaluate more complex models.

However, at least as important as all of the above questions are those raised by our initial foray into defining a framework to describe meaning in Section 5.6. For example, what distribution over statements should Equation 5.4 use to define a distance measure between universes? How should it account for interpretability? What can we say about the topology of the $eval$ output space, especially for "well-formed" statements? Do we need a higher dimensional output to account for different sorts of interpretability? How do questions and other types of utterances differ from the kinds of statements we used in our examples?

And most prominently, while we have given some insights into how the $read(\cdot, \Omega^0)$ and $eval(\cdot, \Omega^0)$ functions relate to several language tasks, we have said little about how to handle more general universes, or, for that matter, how the base universe Ω^0 relates to the original universe $\Omega^\emptyset$. Even if we were to overlook the latter problem, language understanding has to rely at some point on the ability to use and retain new information across settings, or across iterations of a task. A number of promising recent works, such as memory-based Neural Networks [Sukhbaatar et al., 2015] or Neural Turing Machines [Graves et al., 2016] aim to keep longer term memory of read statements, which can help represent a universe in a tractable way. Unfortunately, these models still need to be made drastically faster and more memory efficient before they can be of general use in language understanding settings.

All of the above questions present promising avenues for future research, which will need to be explored on the way to developing generally useful systems for language understanding. We look forward to seeing future research build upon the ideas proposed in the present work to that end, and to add our own further contributions to the advancement of the field.

Bibliography

[Agirre et al., 2012] Agirre, Eneko, Cer, Daniel M., Diab, Mona T., and Gonzalez-Agirre, Aitor (2012). SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity. In *Proceedings of the 6th International Workshop on Semantic Evaluation*, pages 385–393.

[Agrawal et al., 2013] Agrawal, Rahul, Gupta, Archit, Prabhu, Yashoteja, and Varma, Manik (2013). Multi-Label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages. In *WWW 2013, 22nd International World Wide Web Conference*, pages 13–24.

[Alexandrescu and Kirchhoff, 2006] Alexandrescu, Andrei and Kirchhoff, Katrin (2006). Factored Neural Language Models. In *HLT-NAACL 2006, Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*.

[Andor et al., 2016] Andor, Daniel, Alberti, Chris, Weiss, David, Severyn, Aliaksei, Presta, Alessandro, Ganchev, Kuzman, Petrov, Slav, and Collins, Michael (2016). Globally Normalized Transition-Based Neural Networks. In *ACL 2016, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[Andreas and Klein, 2015] Andreas, Jacob and Klein, Dan (2015). When and why are log-linear models self-normalizing? In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 244–249.

[Andreas et al., 2016] Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Klein, Dan (2016). Learning to Compose Neural Networks for Question Answering. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554.

[Armstrong, 1986] Armstrong, David M. (1986). The nature of possibility. *Canadian Journal of Philosophy*, 16(4):575–594.

[Azocar et al., 2011] Azocar, A., Gimenez, J., Nikodem, K., and Sanchez, J. L. (2011). On Strongly Midconvex Functions. *Opuscula Math.*, 31:15–26.

[Bahdanau et al., 2014] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473.

[Bajgar et al., 2016] Bajgar, Ondrej, Kadlec, Rudolf, and Kleindienst, Jan (2016). Embracing Data Abundance: BookTest Dataset for Reading Comprehension. *CoRR*, abs/1610.00956.

[Barzilay and Lapata, 2005] Barzilay, Regina and Lapata, Mirella (2005). Modeling Local Coherence: an Entity-Based Approach. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics*, pages 141–148.

[Bengio et al., 2010] Bengio, Samy, Weston, Jason, and Grangier, David (2010). Label Embedding Trees for Large Multi-Class Tasks. In *NIPS 2010, Advances in Neural Information Processing Systems 23*, pages 163–171.

[Bengio et al., 2003] Bengio, Yoshua, Ducharme, Rejean, and Vincent, Pascal (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bengio and Senecal, 2003] Bengio, Yoshua and Senecal, Jean-Sébastien (2003). Quick Training of Probabilistic Neural Nets by Importance Sampling. In *AISTATS 2003, Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

[Bengio and Senecal, 2008] Bengio, Yoshua and Senecal, Jean-Sébastien (2008). Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model. *IEEE Trans. Neural Networks*, 19(4):713–722.

[Bengio et al., 1994] Bengio, Yoshua, Simard, Patrice Y., and Frasconi, Paolo (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.

[Berant et al., 2013] Berant, Jonathan, Chou, Andrew, Frostig, Roy, and Liang, Percy (2013). Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP 2013, Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

[Beygelzimer et al., 2009a] Beygelzimer, Alina, Langford, John, Lifshits, Yury, Sorkin, Gregory B., and Strehl, Alexander L. (2009a). Conditional Probability Tree Estima-

tion Analysis and Algorithms. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 51–58.

[Beygelzimer et al., 2009b] Beygelzimer, Alina, Langford, John, and Ravikumar, Pradeep (2009b). Error-Correcting Tournaments. In *ALT 2009, Algorithmic Learning Theory, 20th International Conference*, pages 247–262.

[Bhatia et al., 2015] Bhatia, Kush, Jain, Himanshu, Kar, Purushottam, Varma, Manik, and Jain, Prateek (2015). Sparse Local Embeddings for Extreme Multi-Label Classification. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 730–738.

[Bilmes and Kirchhoff, 2003] Bilmes, Jeff A. and Kirchhoff, Katrin (2003). Factored Language Models and Generalized Parallel Backoff. In *HLT-NAACL 2003, Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

[Bird, 2006] Bird, Steven (2006). NLTK: the natural language toolkit. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*.

[Bojanowski et al., 2017] Bojanowski, Piotr, Grave, Edouard, Joulin, Armand, and Mikolov, Tomas (2017). Enriching Word Vectors with Subword Information. *TACL*, 5:135–146.

[Bojanowski et al., 2015] Bojanowski, Piotr, Joulin, Armand, and Mikolov, Tomas (2015). Alternative Structures for Character-Level RNNs. *CoRR*, abs/1511.06303.

[Bollacker et al., 2007] Bollacker, Kurt D., Cook, Robert P., and Tufts, Patrick (2007). Freebase: A Shared Database of Structured General Human Knowledge. In *AAAI*

2007, Proceedings of the Twenty-Second Conference on Artificial Intelligence, pages 1962–1963.

[Botha, 2014] Botha, Jan Abraham (2014). *Probabilistic Modelling of Morphologically Rich Languages*. PhD thesis, University of Oxford, UK.

[Botha and Blunsom, 2014] Botha, Jan A. and Blunsom, Phil (2014). Compositional Morphology for Word Representations and Language Modelling. In *ICML 2014, Proceedings of the 31th International Conference on Machine Learning*, pages 1899–1907.

[Bottou, 1991] Bottou, Léon (1991). *Une Approche Théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis.

[Bottou, 1998] Bottou, Léon (1998). Online algorithms and stochastic approximations. In Saad, David, editor, *Online Learning and Neural Networks*. Cambridge University Press.

[Bowman et al., 2015] Bowman, Samuel R., Angeli, Gabor, Potts, Christopher, and Manning, Christopher D. (2015). A Large Annotated Corpus for Learning Natural Language Inference. In *EMNLP 2015, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642.

[Brants, 2000] Brants, Thorsten (2000). TnT – A Statistical Part-of-Speech Tagger. In *ANLP 2000, 6th Applied Natural Language Processing Conference*, pages 224–231.

[Breiman, 2001] Breiman, Leo (2001). Random Forests. *Machine Learning*, 45(1):5–32.

[Breiman et al., 1984] Breiman, Leo, Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

[Brown et al., 1990] Brown, Peter F., Cocke, John, Pietra, Stephen Della, Pietra, Vincent J. Della, Jelinek, Frederick, Lafferty, John D., Mercer, Robert L., and Roossin, Paul S. (1990). A Statistical Approach to Machine Translation. *Computational Linguistics*, 16(2):79–85.

[Brown et al., 1993] Brown, Peter F., Pietra, Stephen Della, Pietra, Vincent J. Della, and Mercer, Robert L. (1993). The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311.

[Bui et al., 2013] Bui, Hung Hai, Huynh, Tuyen N., and Riedel, Sebastian (2013). Automorphism Groups of Graphical Models and Lifted Variational Inference. In *UAI 2013, Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*.

[Bui et al., 2014] Bui, Hung Hai, Huynh, Tuyen N., and Sontag, David (2014). Lifted Tree-Reweighted Variational Inference. In *UAI 2014, Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 92–101.

[Carbonell et al., 1981] Carbonell, Jaime G., Cullingford, Richard E., and Gershman, Anatole V. (1981). Steps Toward Knowledge-Based Machine Translation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):376–392.

[Carbonell and Tomita, 1985] Carbonell, Jaime G. and Tomita, Masaru (1985). New Approaches to Machine Translation. In *Proceedings of the conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*.

[Carnap, 1947] Carnap, Rudolf (1947). *Meaning and Necessity*. University of Chicago Press.

[Chandar et al., 2016] Chandar, Sarath, Ahn, Sungjin, Larochelle, Hugo, Vincent, Pascal, Tesauro, Gerald, and Bengio, Yoshua (2016). Hierarchical Memory Networks. *CoRR*, abs/1605.07427.

[Chen and Goodman, 1998] Chen, Stanley and Goodman, Joshua (1998). An Empirical Study of Smoothing Techniques for Language Modeling. *Technical Report, Harvard University*.

[Cheng et al., 2014] Cheng, Wei-Chen, Kok, Stanley, Pham, Hoai Vu, Chieu, Hai Leong, and Chai, Kian Ming Adam (2014). Language Modeling with Sum-Product Networks. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association*, pages 2098–2102.

[Chiang, 2005] Chiang, David (2005). A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics*, pages 263–270.

[Cho et al., 2014a] Cho, Kyunghyun, van Merrienboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua (2014a). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

[Cho et al., 2014b] Cho, Kyunghyun, van Merrienboer, Bart, Gülçehre, Çaglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua (2014b). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Ma-

chine Translation. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

[Choromanska et al., 2016] Choromanska, Anna, Choromanski, Krzysztof, and Bojarski, Mariusz (2016). On the Boosting Ability of Top-Down Decision Tree Learning Algorithm for Multiclass Classification. *CoRR*, abs/1605.05223.

[Choromanska and Langford, 2015] Choromanska, Anna and Langford, John (2015). Logarithmic Time Online Multiclass Prediction. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 55–63.

[Chung et al., 2016] Chung, Junyoung, Ahn, Sungjin, and Bengio, Yoshua (2016). Hierarchical Multiscale Recurrent Neural Networks. *CoRR*, abs/1609.01704.

[Collobert et al., 2011] Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel P. (2011). Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

[Conneau et al., 2017] Conneau, Alexis, Kiela, Douwe, Schwenk, Holger, Barrault, Loïc, and Bordes, Antoine (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *EMNLP 2017, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 681–691.

[Creutz and Lagus, 2007] Creutz, Mathias and Lagus, Krista (2007). Unsupervised Models for Morpheme Segmentation and Morphology Learning. *TSLP*, 4(1):3:1–3:34.

[Cullingford, 1978] Cullingford, Richard Edward (1978). Script Application: Computer Understanding of Newspaper Stories. Technical report.

[Dai and Le, 2015] Dai, Andrew M. and Le, Quoc V. (2015). Semi-Supervised Sequence Learning. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 3079–3087.

[Das et al., 2014] Das, Dipanjan, Chen, Desai, Martins, André F. T., Schneider, Nathan, and Smith, Noah A. (2014). Frame-Semantic Parsing. *Computational Linguistics*, 40(1):9–56.

[Daumé III et al., 2017] Daumé III, Hal, Karampatziakis, Nikos, Langford, John, and Mineiro, Paul (2017). Logarithmic Time One-Against-Some. In *ICML 2017, Proceedings of the 34th International Conference on Machine Learning*, pages 923–932.

[Dauphin et al., 2017] Dauphin, Yann N., Fan, Angela, Auli, Michael, and Grangier, David (2017). Language Modeling with Gated Convolutional Networks. In *ICML 2017, Proceedings of the 34th International Conference on Machine Learning*, pages 933–941.

[de Brébisson and Vincent, 2015] de Brébisson, Alexandre and Vincent, Pascal (2015). An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family. *CoRR*, abs/1511.05042.

[Deerwester et al., 1990] Deerwester, Scott C., Dumais, Susan T., Landauer, Thomas K., Furnas, George W., and Harshman, Richard A. (1990). Indexing by Latent Semantic Analysis. *JASIS*, 41(6):391–407.

[Deng et al., 2011] Deng, Jia, Satheesh, Sanjeev, Berg, Alexander C., and Li, Fei-Fei (2011). Fast and Balanced: Efficient Label Tree Learning for Large Scale Object

Recognition. In *NIPS 2011, Advances in Neural Information Processing Systems 24*, pages 567–575.

[dos Santos and Guimarães, 2015] dos Santos, Cícero Nogueira and Guimarães, Victor (2015). Boosting Named Entity Recognition with Neural Character Embeddings. *CoRR*, abs/1505.05008.

[dos Santos and Zadrozny, 2014] dos Santos, Cícero Nogueira and Zadrozny, Bianca (2014). Learning Character-level Representations for Part-of-Speech Tagging. In *ICML 2014, Proceedings of the 31th International Conference on Machine Learning*, pages 1818–1826.

[Duchi et al., 2010] Duchi, John C., Hazan, Elad, and Singer, Yoram (2010). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In *COLT 2010, The 23rd Conference on Learning Theory*, pages 257–269.

[Elman, 1990] Elman, Jeffrey L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

[Fine et al., 1998] Fine, Shai, Singer, Yoram, and Tishby, Naftali (1998). The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41–62.

[Gael et al., 2008] Gael, Jurgen Van, Teh, Yee Whye, and Ghahramani, Zoubin (2008). The Infinite Factorial Hidden Markov Model. In *NIPS 2008, Advances in Neural Information Processing Systems 21*, pages 1697–1704.

[Galley et al., 2006] Galley, Michel, Graehl, Jonathan, Knight, Kevin, Marcu, Daniel, DeNeefe, Steve, Wang, Wei, and Thayer, Ignacio (2006). Scalable Inference and

Training of Context-Rich Syntactic Translation Models. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*.

[Gan et al., 2016] Gan, Zhe, Pu, Yunchen, Henao, Ricardo, Li, Chunyuan, He, Xiaodong, and Carin, Lawrence (2016). Unsupervised Learning of Sentence Representations Using Convolutional Neural Networks. *CoRR*, abs/1611.07897.

[Ghahramani and Hinton, 2000] Ghahramani, Zoubin and Hinton, Geoffrey E. (2000). Variational Learning for Switching State-Space Models. *Neural Computation*, 12(4):831–864.

[Ghahramani and Jordan, 1997] Ghahramani, Zoubin and Jordan, Michael I. (1997). Factorial Hidden Markov Models. *Machine Learning*, 29(2-3):245–273.

[Globerson and Jaakkola, 2007] Globerson, Amir and Jaakkola, Tommi S. (2007). Convergent propagation algorithms via oriented trees. In *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 133–140.

[Graff and Cieri, 2003] Graff, David and Cieri, C (2003). English Gigaword Corpus. *Linguistic Data Consortium*.

[Grave et al., 2017a] Grave, Edouard, Joulin, Armand, Cissé, Moustapha, Grangier, David, and Jégou, Hervé (2017a). Efficient Softmax Approximation for GPUs. In *ICML 2017, Proceedings of the 34th International Conference on Machine Learning*, pages 1302–1310.

[Grave et al., 2017b] Grave, Edouard, Mikolov, Tomas, Joulin, Armand, and Bojanowski, Piotr (2017b). Bag of Tricks for Efficient Text Classification. In *EACL*

2017, Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, Volume 2: Short Papers, pages 427–431.

[Graves, 2013] Graves, Alex (2013). Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850.

[Graves, 2016] Graves, Alex (2016). Adaptive Computation Time for Recurrent Neural Networks. *CoRR*, abs/1603.08983.

[Graves et al., 2006] Graves, Alex, Fernández, Santiago, Gomez, Faustino J., and Schmidhuber, Jürgen (2006). Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *(ICML 2006), Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 369–376.

[Graves et al., 2016] Graves, Alex, Wayne, Greg, Reynolds, Malcolm, Harley, Tim, Danihelka, Ivo, Grabska-Barwinska, Agnieszka, Colmenarejo, Sergio Gomez, Grefenstette, Edward, Ramalho, Tiago, Agapiou, John, Badia, Adrià Puigdomènech, Hermann, Karl Moritz, Zwols, Yori, Ostrovski, Georg, Cain, Adam, King, Helen, Summerfield, Christopher, Blunsom, Phil, Kavukcuoglu, Koray, and Hassabis, Demis (2016). Hybrid Computing Using a Neural Network with Dynamic External Memory. *Nature*, 538(7626):471–476.

[Gutmann and Hyvärinen, 2012] Gutmann, Michael and Hyvärinen, Aapo (2012). Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. *Journal of Machine Learning Research*, 13:307–361.

[Guu et al., 2015] Guu, Kelvin, Miller, John, and Liang, Percy (2015). Traversing Knowledge Graphs in Vector Space. In *EMNLP 2015, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327.

[Hazan and Urtasun, 2010] Hazan, Tamir and Urtasun, Raquel (2010). A Primal-Dual Message-Passing Algorithm for Approximated Large Scale Structured Prediction. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 838–846.

[Hill et al., 2015] Hill, Felix, Bordes, Antoine, Chopra, Sumit, and Weston, Jason (2015). The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations. *CoRR*, abs/1511.02301.

[Hill et al., 2016] Hill, Felix, Cho, Kyunghyun, and Korhonen, Anna (2016). Learning Distributed Representations of Sentences from Unlabelled Data. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377.

[Hinton, 1986] Hinton, Geoffrey E. (1986). Learning Distributed Representations of Concepts. In *Proceedings of the Eight Annual Conference of Cognitive Science Society*, pages 1–12.

[Hinton et al., 2012] Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan (2012). Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *CoRR*, abs/1207.0580.

[Hobbs, 1979] Hobbs, Jerry R. (1979). Coherence and coreference. *Cognitive Science*, 3(1):67–90.

[Hochreiter and Schmidhuber, 1997] Hochreiter, Sepp and Schmidhuber, Jürgen (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[Hornik et al., 1989] Hornik, Kurt, Stinchcombe, Maxwell B., and White, Halbert (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359–366.

[Hsu et al., 2009] Hsu, Daniel J., Kakade, Sham, Langford, John, and Zhang, Tong (2009). Multi-Label Prediction via Compressed Sensing. In *NIPS 2009, Advances in Neural Information Processing Systems 22*, pages 772–780.

[Hughes and Cresswell, 1968] Hughes, GE and Cresswell, MJ (1968). *An Introduction to Modal Logic*. Methuen, London.

[Hughes and Cresswell, 1996] Hughes, GE and Cresswell, MJ (1996). *A New Introduction to Modal Logic*. Routledge, London.

[Iyyer et al., 2014] Iyyer, Mohit, Boyd-Graber, Jordan L., Claudino, Leonardo Max Batista, Socher, Richard, and III, Hal Daumé (2014). A Neural Network for Factoid Question Answering over Paragraphs. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 633–644.

[Jancsary and Matz, 2011] Jancsary, Jeremy and Matz, Gerald (2011). Convergent decomposition solvers for tree-reweighted free energies. In *AISTATS 2011, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 388–398.

[Jelinek and Mercer, 1980] Jelinek, Fred and Mercer, Robert L. (1980). Interpolated Estimation of Markov Source Parameters from Sparse Data. In *Workshop on Pattern Recognition in Practice 2000*, pages 381–397.

[Jernite et al., 2017a] Jernite, Yacine, Bowman, Samuel R., and Sontag, David (2017a). Discourse-Based Objectives for Fast Unsupervised Sentence Representation Learning. *CoRR*, abs/1705.00557.

[Jernite et al., 2017b] Jernite, Yacine, Choromanska, Anna, and Sontag, David (2017b). Simultaneous Learning of Trees and Representations for Extreme Classification and Density Estimation. In *ICML 2017, Proceedings of the 34th International Conference on Machine Learning*, pages 1665–1674.

[Jernite et al., 2016] Jernite, Yacine, Grave, Edouard, Joulin, Armand, and Mikolov, Tomas (2016). Variable Computation in Recurrent Neural Networks. *CoRR*, abs/1611.06188.

[Jernite et al., 2015] Jernite, Yacine, Rush, Alexander M., and Sontag, David (2015). A Fast Variational Approach for Learning Markov Random Field Language Models. In *ICML 2015, Proceedings of the 32nd International Conference on Machine Learning*, pages 2209–2217.

[Ji et al., 2015] Ji, Yangfeng, Cohn, Trevor, Kong, Lingpeng, Dyer, Chris, and Eisenstein, Jacob (2015). Document Context Language Models. *CoRR*, abs/1511.03962.

[Ji et al., 2016] Ji, Yangfeng, Haffari, Gholamreza, and Eisenstein, Jacob (2016). A Latent Variable Recurrent Neural Network for Discourse Relation Language Models. *CoRR*, abs/1603.01913.

[Johnson et al., 2017] Johnson, Melvin, Schuster, Mike, Le, Quoc V., Krikun, Maxim, Wu, Yonghui, Chen, Zhifeng, Thorat, Nikhil, Viégas, Fernanda B., Wattenberg, Martin, Corrado, Greg, Hughes, Macduff, and Dean, Jeffrey (2017). Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *TACL*, 5:339–351.

[Józefowicz et al., 2016] Józefowicz, Rafal, Vinyals, Oriol, Schuster, Mike, Shazeer, Noam, and Wu, Yonghui (2016). Exploring the Limits of Language Modeling. *CoRR*, abs/1602.02410.

[Kalchbrenner and Blunsom, 2013] Kalchbrenner, Nal and Blunsom, Phil (2013). Recurrent continuous translation models. In *EMNLP 2013, Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

[Kalchbrenner et al., 2014] Kalchbrenner, Nal, Grefenstette, Edward, and Blunsom, Phil (2014). A Convolutional Neural Network for Modelling Sentences. In *ACL 2014, Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 655–665.

[Katz, 1987] Katz, Slava M. (1987). Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35(3):400–401.

[Katz-Brown et al., 2011] Katz-Brown, Jason, Petrov, Slav, McDonald, Ryan T., Och, Franz Josef, Talbot, David, Ichikawa, Hiroshi, Seno, Masakazu, and Kazawa, Hideto (2011). Training a Parser for Machine Translation Reordering. In *EMNLP 2011,*

Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 183–192.

[Kim, 2014] Kim, Yoon (2014). Convolutional Neural Networks for Sentence Classification. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751.

[Kim et al., 2016] Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. (2016). Character-Aware Neural Language Models. In *AAAI 2016, Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2741–2749.

[Kirkpatrick et al., 2016] Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil C., Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A., Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, Hassabis, Demis, Clopath, Claudia, Kumaran, Dharshan, and Hadsell, Raia (2016). Overcoming Catastrophic Forgetting in Neural Networks. *CoRR*, abs/1612.00796.

[Kiros et al., 2015] Kiros, Jamie, Zhu, Yukun, Salakhutdinov, Ruslan, Zemel, Richard S., Urtasun, Raquel, Torralba, Antonio, and Fidler, Sanja (2015). Skip-Thought Vectors. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 3294–3302.

[Koehn, 2005] Koehn, Philipp (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

[Koutník et al., 2014] Koutník, Jan, Greff, Klaus, Gomez, Faustino J., and Schmidhuber, Jürgen (2014). A Clockwork RNN. In *ICML 2014, Proceedings of the 31th International Conference on Machine Learning*, pages 1863–1871.

[Kripke, 1963] Kripke, Saul A. (1963). Semantical Analysis of Modal Logic: Normal Modal Propositional Calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96.

[Krizhevsky et al., 2012] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS 2012, Advances in Neural Information Processing Systems 25*, pages 1106–1114.

[Kupiec et al., 1995] Kupiec, Julian, Pedersen, Jan O., and Chen, Francine (1995). A Trainable Document Summarizer. In *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle*, pages 68–73.

[Lample et al., 2017] Lample, Guillaume, Denoyer, Ludovic, and Ranzato, Marc'Aurelio (2017). Unsupervised Machine Translation Using Monolingual Corpora Only. *CoRR*, abs/1711.00043.

[Le et al., 2011] Le, Hai Son, Oparin, Ilya, Allauzen, Alexandre, Gauvain, Jean-Luc, and Yvon, François (2011). Structured output layer neural network language model. In *ICASSP 2011, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 5524–5527.

[LeCun et al., 1989] LeCun, Yann, Boser, Bernhard E., Denker, John S., Henderson, Donnie, Howard, Richard E., Hubbard, Wayne E., and Jackel, Lawrence D. (1989). Handwritten Digit Recognition with a Backpropagation Network. In *NIPS 1989, Advances in Neural Information Processing Systems 2*, pages 396–404.

[Lee et al., 2017] Lee, Jason, Cho, Kyunghyun, and Hofmann, Thomas (2017). Fully Character-Level Neural Machine Translation without Explicit Segmentation. *TACL*, 5:365–378.

[Lei et al., 2015] Lei, Tao, Barzilay, Regina, and Jaakkola, Tommi S. (2015). Molding CNNs for Text: Non-linear, Non-Consecutive Convolutions. In *EMNLP 2015, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1565–1575.

[Lewis and Langford, 1959] Lewis, Clarence Irving and Langford, Cooper Harold (1959). *Symbolic Logic*. Dover Publications New York.

[Lewis, 1970] Lewis, David (1970). General semantics. *Synthese*, 22(1):18–67.

[Lewis, 1986] Lewis, David (1986). *On the plurality of worlds*, volume 322. Oxford University Press.

[Li and Hovy, 2014] Li, Jiwei and Hovy, Eduard H. (2014). A Model of Coherence Based on Distributed Sentence Representation. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 2039–2048.

[Li and Jurafsky, 2016] Li, Jiwei and Jurafsky, Dan (2016). Neural Net Models for Open-Domain Discourse Coherence. *CoRR*, abs/1606.01545.

[Liang, 2016] Liang, Percy (2016). Learning Executable Semantic Parsers for Natural Language Understanding. *Commun. ACM*, 59(9):68–76.

[Liang and Jordan, 2008] Liang, Percy and Jordan, Michael I. (2008). An Asymptotic Analysis of Generative, Discriminative, and Pseudolikelihood Estimators. In *ICML 2008, Machine Learning, Proceedings of the Twenty-Fifth International Conference*, pages 584–591.

[Liang et al., 2011] Liang, Percy, Jordan, Michael I., and Klein, Dan (2011). Learning Dependency-Based Compositional Semantics. In *ACL 2011, The 49th Annual Meeting of the Association for Computational Linguistics*, pages 590–599.

[Ling et al., 2015] Ling, Wang, Dyer, Chris, Black, Alan W., Trancoso, Isabel, Fermandez, Ramon, Amir, Silvio, Marujo, Luís, and Luís, Tiago (2015). Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *EMNLP 2015, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530.

[Liu and Nocedal, 1989] Liu, Dong C. and Nocedal, Jorge (1989). On the Limited Memory BFGS Method for Large Scale Optimization. *Math. Program.*, 45(1-3):503–528.

[Logeswaran et al., 2016] Logeswaran, Lajanugen, Lee, Honglak, and Radev, Dragomir R. (2016). Sentence Ordering Using Recurrent Neural Networks. *CoRR*, abs/1611.02654.

[Lowe et al., 2017] Lowe, Ryan, Noseworthy, Michael, Serban, Iulian Vlad, Angelard-Gontier, Nicolas, Bengio, Yoshua, and Pineau, Joelle (2017). Towards an Automatic Turing Test: Learning to Evaluate Dialogue Responses. In *ACL 2017, Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1116–1126.

[Luong et al., 2013] Luong, Thang, Socher, Richard, and Manning, Christopher D. (2013). Better Word Representations with Recursive Neural Networks for Morphology. In *CoNLL 2013, Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113.

[MacCartney, 2009] MacCartney, Bill (2009). *Natural Language Inference*. Stanford University.

[Madzarov et al., 2009] Madzarov, Gjorgji, Gjorgjevikj, Dejan, and Chorbev, Ivan (2009). A Multi-Class SVM Classifier Utilizing Binary Decision Tree. *Informatica (Slovenia)*, 33(2):225–233.

[Marcus et al., 1993] Marcus, Mitchell P., Santorini, Beatrice, and Marcinkiewicz, Mary Ann (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[Meshi et al., 2010] Meshi, Ofer, Sontag, David, Jaakkola, Tommi S., and Globerson, Amir (2010). Learning Efficiently with Approximate Inference via Dual Losses. In *(ICML-10), Proceedings of the 27th International Conference on Machine Learning*, pages 783–790.

[Mikolov et al., 2013] Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey (2013). Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2011] Mikolov, Tomas, Deoras, Anoop, Kombrink, Stefan, Burget, Lukás, and Cernocký, Jan (2011). Empirical Evaluation and Combination of Advanced Language Modeling Techniques. In *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association*, pages 605–608.

[Mikolov et al., 2014] Mikolov, Tomas, Joulin, Armand, Chopra, Sumit, Mathieu, Michaël, and Ranzato, Marc'Aurelio (2014). Learning Longer Memory in Recurrent Neural Networks. *CoRR*, abs/1412.7753.

[Mikolov et al., 2010] Mikolov, Tomas, Karafiát, Martin, Burget, Lukás, Cernocký, Jan, and Khudanpur, Sanjeev (2010). Recurrent Neural Network Based Language Model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048.

[Mikolov and Zweig, 2012] Mikolov, Tomas and Zweig, Geoffrey (2012). Context Dependent Recurrent Neural Network Language Model. In *IEEE 2012, Spoken Language Technology Workshop (SLT)*, pages 234–239.

[Miltsakaki et al., 2004] Miltsakaki, Eleni, Prasad, Rashmi, Joshi, Aravind K., and Webber, Bonnie L. (2004). The Penn Discourse Treebank. In *LREC 2004, Proceedings of the Fourth International Conference on Language Resources and Evaluation*.

[Mirowski and Vlachos, 2015] Mirowski, Piotr and Vlachos, Andreas (2015). Dependency Recurrent Neural Language Models for Sentence Completion. In *ACL 2015, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, Volume 2: Short Papers*, pages 511–517.

[Mnih and Hinton, 2007] Mnih, Andriy and Hinton, Geoffrey E. (2007). Three New Graphical Models for Statistical Language Modelling. In *ICML 2007, Machine Learning, Proceedings of the 24th International Conference on Machine Learning*, pages 641–648.

[Mnih and Hinton, 2008] Mnih, Andriy and Hinton, Geoffrey E. (2008). A Scalable Hierarchical Distributed Language Model. In *NIPS 2008, Advances in Neural Information Processing Systems 21*, pages 1081–1088.

[Mnih and Teh, 2012] Mnih, Andriy and Teh, Yee Whye (2012). A Fast and Simple Algorithm for Training Neural Probabilistic Language Models. In *ICML 2012, Proceedings of the 29th International Conference on Machine Learning*.

[Montague, 1970] Montague, Richard (1970). Universal Grammar. *Theoria*, 36(3):373–398.

[Montague, 1973] Montague, Richard (1973). The Proper Treatment of Quantification in Ordinary English. In *Philosophy, language, and artificial intelligence*, pages 141–162. Springer.

[Morin and Bengio, 2005] Morin, Frederic and Bengio, Yoshua (2005). Hierarchical Probabilistic Neural Network Language Model. In *AISTATS 2005, Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*.

[Mozer, 1991] Mozer, Michael (1991). Induction of Multiscale Temporal Structure. In *NIPS 1991, Advances in Neural Information Processing Systems 4*, pages 275–282.

[Murphy and Paskin, 2001] Murphy, Kevin P. and Paskin, Mark A. (2001). Linear-Time Inference in Hierarchical HMMs. In *NIPS 2001, Advances in Neural Information Processing Systems 14*, pages 833–840.

[Nallapati et al., 2016] Nallapati, Ramesh, Zhou, Bowen, dos Santos, Cícero Nogueira, Gülçehre, Çaglar, and Xiang, Bing (2016). Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 280–290.

[Nickel and Kiela, 2017] Nickel, Maximilian and Kiela, Douwe (2017). Poincaré Embeddings for Learning Hierarchical Representations. *CoRR*, abs/1705.08039.

[Nortmann, 2002] Nortmann, Ulrich (2002). The Logic of Necessity in Aristotle–an Outline of Approaches to the Modal Syllogistic, Together with a General Account of de dicto-and de re-Necessity. *History and Philosophy of Logic*, 23(4):253–265.

[Och and Ney, 2004] Och, Franz Josef and Ney, Hermann (2004). The Alignment Template Approach to Statistical Machine Translation. *Computational Linguistics*, 30(4):417–449.

[Paperno et al., 2016] Paperno, Denis, Kruszewski, Germán, Lazaridou, Angeliki, Pham, Quan Ngoc, Bernardi, Raffaella, Pezzelle, Sandro, Baroni, Marco, Boleda, Gemma, and Fernández, Raquel (2016). The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context. In *ACL 2016, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[Pascanu et al., 2013a] Pascanu, Razvan, Gülçehre, Çaglar, Cho, Kyunghyun, and Bengio, Yoshua (2013a). How to Construct Deep Recurrent Neural Networks. *CoRR*, abs/1312.6026.

[Pascanu et al., 2013b] Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua (2013b). On the Difficulty of Training Recurrent Neural Networks. In *ICML 2013, Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318.

[Pennington et al., 2014] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. (2014). Glove: Global vectors for word representation. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

[Plantinga, 1976] Plantinga, Alvin (1976). Actualism and Possible Worlds. *Theoria*, 42(1-3):139–160.

[Prabhu and Varma, 2014] Prabhu, Yashoteja and Varma, Manik (2014). FastXML: a Fast, Accurate and Stable Tree-Classifier for Extreme Multi-Label Learning. In *KDD 2014, The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–272.

[Qiu et al., 2014] Qiu, Siyu, Cui, Qing, Bian, Jiang, Gao, Bin, and Liu, Tie-Yan (2014). Co-learning of Word Representations and Morpheme Representations. In *COLING 2014, 25th International Conference on Computational Linguistics*, pages 141–150.

[Rajpurkar et al., 2016] Rajpurkar, Pranav, Zhang, Jian, Lopyrev, Konstantin, and Liang, Percy (2016). SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *EMNLP 2016, Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

[Ramachandran et al., 2016] Ramachandran, Prajit, Liu, Peter J., and Le, Quoc V. (2016). Unsupervised Pretraining for Sequence to Sequence Learning. *CoRR*, abs/1611.02683.

[Rush et al., 2015] Rush, Alexander M., Chopra, Sumit, and Weston, Jason (2015). A Neural Attention Model for Abstractive Sentence Summarization. In *EMNLP 2015, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389.

[Sánchez Valencia, 1991] Sánchez Valencia, Víctor Manuel (1991). *Studies on Natural Logic and Categorial Grammar*. ITLI.

[Schank and Abelson, 1977] Schank, Roger C. and Abelson, Robert P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

[Schmidhuber, 1992] Schmidhuber, Jürgen (1992). Learning Complex, Extended Sequences Using the Principle of History Compression. *Neural Computation*, 4(2):234–242.

[Schwenk and Gauvain, 2002] Schwenk, Holger and Gauvain, Jean-Luc (2002). Connectionist Language Modeling for Large Vocabulary Continuous Speech Recognition. In *ICASSP 2002, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 765–768.

[Schwenk and Gauvain, 2005] Schwenk, Holger and Gauvain, Jean-Luc (2005). Training Neural Network Language Models on Very Large Corpora. In *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 201–208.

[Sennrich et al., 2016] Sennrich, Rico, Haddow, Barry, and Birch, Alexandra (2016). Neural Machine Translation of Rare Words with Subword Units. In *ACL 2016, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[Shalev-Shwartz, 2012] Shalev-Shwartz, Shai (2012). Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194.

[Shen et al., 2014] Shen, Yelong, He, Xiaodong, Gao, Jianfeng, Deng, Li, and Mesnil, Grégoire (2014). A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *CIKM 2014, Proceedings of the 23rd ACM International*

Conference on Conference on Information and Knowledge Management, pages 101–110.

[Socher et al., 2013] Socher, Richard, Perelygin, Alex, Wu, Jean Y., Chuang, Jason, Manning, Christopher D., Ng, Andrew Y., and Potts, Christopher (2013). Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *EMNLP 2013, Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

[Srivastava et al., 2015] Srivastava, Rupesh Kumar, Greff, Klaus, and Schmidhuber, Jürgen (2015). Training Very Deep Networks. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 2377–2385.

[Stroube, 2003] Stroube, Bryan (2003). Literary Freedom: Project Gutenberg. *ACM Crossroads*, 10(1):3.

[Sukhbaatar et al., 2017] Sukhbaatar, Sainbayar, Kostrikov, Ilya, Szlam, Arthur, and Fergus, Rob (2017). Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. *CoRR*, abs/1703.05407.

[Sukhbaatar et al., 2015] Sukhbaatar, Sainbayar, Szlam, Arthur, Weston, Jason, and Fergus, Rob (2015). End-to-End Memory Networks. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 2440–2448.

[Sundermeyer et al., 2012] Sundermeyer, Martin, Schlüter, Ralf, and Ney, Hermann (2012). LSTM Neural Networks for Language Modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association*, pages 194–197.

[Sutskever et al., 2011] Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. (2011). Generating Text with Recurrent Neural Networks. In *ICML 2011, Proceedings of the 28th International Conference on Machine Learning*, pages 1017–1024.

[Sutskever et al., 2014] Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. (2014). Sequence to Sequence Learning with Neural Networks. In *NIPS 2014, Advances in Neural Information Processing Systems 27: Annual Conference*, pages 3104–3112.

[Tarski and Vaught, 1956] Tarski, Alfred and Vaught, Robert L. (1956). Arithmetical Extensions of Relational Systems. *Compositio Mathematicae*, 13:81–102.

[Thomee et al., 2016] Thomee, Bart, Shamma, David A., Friedland, Gerald, Elizalde, Benjamin, Ni, Karl, Poland, Douglas, Borth, Damian, and Li, Li-Jia (2016). YFCC100M: the New Data in Multimedia Research. *Commun. ACM*, 59(2):64–73.

[Vaswani et al., 2017] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia (2017). Attention is all you Need. *CoRR*, abs/1706.03762.

[Venugopal et al., 2007] Venugopal, Ashish, Zollmann, Andreas, and Vogel, Stephan (2007). An Efficient Two-Pass Approach to Synchronous-CFG Driven Statistical MT. In *ACL 2007, Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 500–507.

[Vincent et al., 2015] Vincent, Pascal, de Brébisson, Alexandre, and Bouthillier, Xavier (2015). Efficient Exact Gradient Update for Training Deep Networks with Very Large Sparse Targets. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 1108–1116.

[Wainwright et al., 2005] Wainwright, Martin J., Jaakkola, Tommi S., and Willsky, Alan S. (2005). A New Class of Upper Bounds on the Log Partition Function. *IEEE Trans. Information Theory*, 51(7):2313–2335.

[Wainwright and Jordan, 2003] Wainwright, Martin J and Jordan, Michael I (2003). Variational Inference in Graphical Models: The View from the Marginal Polytope. In *PROCEEDINGS OF THE ANNUAL ALLERTON CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING*, volume 41, pages 961–971.

[Wainwright and Jordan, 2008] Wainwright, Martin J. and Jordan, Michael I. (2008). Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305.

[Wang et al., 2015] Wang, Mingxuan, Lu, Zhengdong, Li, Hang, Jiang, Wenbin, and Liu, Qun (2015). genCNN: A Convolutional Architecture for Word Sequence Prediction. In *ACL 2015, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 1567–1576.

[Wang and Cho, 2016] Wang, Tian and Cho, Kyunghyun (2016). Larger-Context Language Modelling with Recurrent Neural Network. In *ACL 2016, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[Werbos, 1990] Werbos, Paul (1990). Back-Propagation Through Time: what it does and how to do it. In *Proceedings of IEEE*.

[Weston et al., 2011] Weston, Jason, Bengio, Samy, and Usunier, Nicolas (2011). WSABIE: Scaling up to Large Vocabulary Image Annotation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 2764–2770.

[Weston et al., 2014] Weston, Jason, Chopra, Sumit, and Adams, Keith (2014). #TagSpace: Semantic Embeddings from Hashtags. In *EMNLP 2014, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1822–1827.

[Weston et al., 2013] Weston, Jason, Makadia, Ameesh, and Yee, Hector (2013). Label Partitioning for Sublinear Ranking. In *ICML 2013, Proceedings of the 30th International Conference on Machine Learning*, pages 181–189.

[Williams et al., 2017] Williams, Adina, Nangia, Nikita, and Bowman, Samuel R. (2017). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *CoRR*, abs/1704.05426.

[Wong and Mooney, 2006] Wong, Yuk Wah and Mooney, Raymond J. (2006). Learning for Semantic Parsing with Statistical Machine Translation. In *ACL 2006, Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*.

[Wong and Mooney, 2007] Wong, Yuk Wah and Mooney, Raymond J. (2007). Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*.

[Wu et al., 2016] Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Lukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa,

Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff, and Dean, Jeffrey (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144.

[Yang et al., 2017] Yang, Zhilin, Hu, Junjie, Salakhutdinov, Ruslan, and Cohen, William W. (2017). Semi-Supervised QA with Generative Domain-Adaptive Nets. In *ACL 2017, Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1040–1050.

[Yang et al., 2016] Yang, Zichao, Yang, Diyi, Dyer, Chris, He, Xiaodong, Smola, Alexander J., and Hovy, Eduard H. (2016). Hierarchical Attention Networks for Document Classification. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

[Yanover et al., 2008] Yanover, Chen, Schueler-Furman, Ora, and Weiss, Yair (2008). Minimizing and Learning Energy Functions for Side-Chain Prediction. *Journal of Computational Biology*, 15(7):899–911.

[Zaremba et al., 2014] Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol (2014). Recurrent Neural Network Regularization. *CoRR*, abs/1409.2329.

[Zelle and Mooney, 1996] Zelle, John M. and Mooney, Raymond J. (1996). Learning to Parse Database Queries Using Inductive Logic Programming. In *AAAI 96, Proceedings of the Thirteenth National Conference on Artificial Intelligenceand Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1050–1055.

[Zettlemoyer and Collins, 2005] Zettlemoyer, Luke S. and Collins, Michael (2005). Learning to Map Sentences to Logical Form: Structured Classification with Prob-

abilistic Categorial Grammars. In *UAI 2005, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, pages 658–666.

[Zhang et al., 2016a] Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol (2016a). Understanding Deep Learning Requires Rethinking Generalization. *CoRR*, abs/1611.03530.

[Zhang and Gildea, 2008] Zhang, Hao and Gildea, Daniel (2008). Efficient Multi-Pass Decoding for Synchronous Context Free Grammars. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 209–217.

[Zhang et al., 2015a] Zhang, Shiliang, Jiang, Hui, Xu, Mingbin, Hou, Junfeng, and Dai, Li-Rong (2015a). The Fixed-Size Ordinally-Forgetting Encoding Method for Neural Network Language Models. In *ACL 2015, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 495–500.

[Zhang et al., 2016b] Zhang, Saizheng, Wu, Yuhuai, Che, Tong, Lin, Zhouhan, Memisevic, Roland, Salakhutdinov, Ruslan, and Bengio, Yoshua (2016b). Architectural Complexity Measures of Recurrent Neural Networks. In *NIPS 2016, Advances in Neural Information Processing Systems 29*, pages 1822–1830.

[Zhang et al., 2015b] Zhang, Xiang, Zhao, Junbo Jake, and LeCun, Yann (2015b). Character-level Convolutional Networks for Text Classification. In *NIPS 2015, Advances in Neural Information Processing Systems 28*, pages 649–657.

[Zhao and Xing, 2013] Zhao, Bin and Xing, Eric P. (2013). Sparse Output Coding for Large-Scale Visual Recognition. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3350–3357.

[Zhu et al., 2015] Zhu, Yukun, Kiros, Ryan, Zemel, Richard S., Salakhutdinov, Ruslan, Urtasun, Raquel, Torralba, Antonio, and Fidler, Sanja (2015). Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *ICCV 2015, 2015 IEEE International Conference on Computer Vision*, pages 19–27.