©Copyright 2018
Aaron Jaech

Low-Rank RNN Adaptation for Context-Aware Language Modeling

Aaron Jaech

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Mari Ostendorf, Chair

Hannaneh Hajishirzi

Noah Smith

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Low-Rank RNN Adaptation for Context-Aware Language Modeling

Aaron Jaech

Chair of the Supervisory Committee:
Professor Mari Ostendorf
Electrical Engineering

A long-standing weakness of statistical language models is that their performance drastically degrades if they are used on data that varies even slightly from the data on which they were trained. In practice, applications require the use of adaptation methods to adjust the predictions of the model to match the local context. For instance, in a speech recognition application, a single static language model would not be able to handle all the different ways that people speak to their voice assistants such as selecting music and sending a message to a friend. An adapted model would make its predictions conditioned on the knowledge of who is speaking and what task they are trying to do.

The current standard approach to recurrent neural network language model adaptation is to apply a simple linear shift to the recurrent and/or output layer bias vector. Although this is helpful, it does not go far enough. This thesis introduces a new approach to adaptation, which we call the FactorCell, that generates a custom recurrent network for each context by applying a low-rank transformation. The FactorCell allows for a more substantial change to the recurrent layer weights. Different from previous approaches, the introduction of a rank hyperparameter gives control over how different or similar the adapted models should be.

In our experiments on several different datasets and multiple types of context, the increased adaptation of the recurrent layer is always helpful, as measured by perplexity, the standard for evaluating language models. We also demonstrate impact on two applica-

tions: personalized query completion and context-specific text generation, finding that the enhanced adaptation benefits both. We also show that the FactorCell provides a more effective text classification model, but more importantly the classification results reveal that there are important differences between the models that are not captured by perplexity. The classification metric is particularly important for the text generation application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor Prof. Mari Ostendorf for her excellent mentorship and for her constant encouragement, patience, and support. I have had the pleasure of working with her on many projects during the last five years and in each case her deep experience has been a valuable asset. I would also like to thank the other members of my thesis committee, Hannaneh Hajishirzi and Noah Smith, with whom I have had the pleasure of collaborating.

My peers and colleagues at the University of Washington have played a key role in my graduate experience. My interactions with them has been the most enjoyable part of my graduate education. I would like to thank Ji He, Hao Fang, Vicky Zayats, Yi Luan, Hao Cheng, Trang Tran, Kevin Lybarger, Farah Nadeem, Rik Koncel-Kedziorski, and Shobhit Hathi. I am very fortunate to have been able to work alongside these students. I thank Arjun Sondhi for the many fruitful discussions.

I am grateful to the mentors I had during my internships, including Henry Schneiderman, Larry Heck, Eric Ringger, and Hetunandan Kamisetty. Each of them taught me important research skills and changed my perspective on the field. This thesis would not have been possible without the experience I gained from working with them.

I thank my parents Jeff and Rebecca Jaech, who have given me their unwavering support, unconditional love, and constant encouragement during all my many years of schooling. Lastly, I acknowledge my lovely girlfriend Gayoung Park, who understands the struggles of a Ph.D student and who has spent many long days and late nights

working by my side. I thank her for her help and for reminding me to always have fun.

Chapter 1
INTRODUCTION

1.1 Overview

Language is a highly adaptable means of communication and as humans we routinely vary our usage of it to match our environment. Changes in language from one setting to another can include large differences in topic, register, politeness, etc. depending on a host of variables such as the social context, the medium of communication, the time and location, and the task at hand. If statistical language models can be made to mimic this contextual adaptability, then they will be useful in a wider range of applications, including speech recognition, machine translation, abstractive summarization, text generation, and more.

Without context awareness, static models are incredibly brittle, meaning they are "extremely sensitive to changes in the style, topic, or genre" (Rosenfeld, 2000). Performance drastically degrades when a model is used in a context other than the one for which it was trained. An early study looked at incorporating text from the Associated Press for modeling contemporaneous articles from the Wall Street Journal (Rosenfeld, 1996). Even though the difference in style between two American news organizations is relatively minor compared to the variations in language that are likely to be observed in other applications, the Associated Press text was practically useless for this task.

Unfortunately, it is often the case that there is a limited amount of text available for learning a language model to characterize a particular context. Therefore, researchers have long sought ways to more effectively use different sources of data. This area of research is often referred to as domain adaptation, characterized by making use of data from multiple contexts to target a particular context. Domain adaptation has been explored for a wide variety of contexts, as shown in Table 1.1. As an illustration of the importance of context,

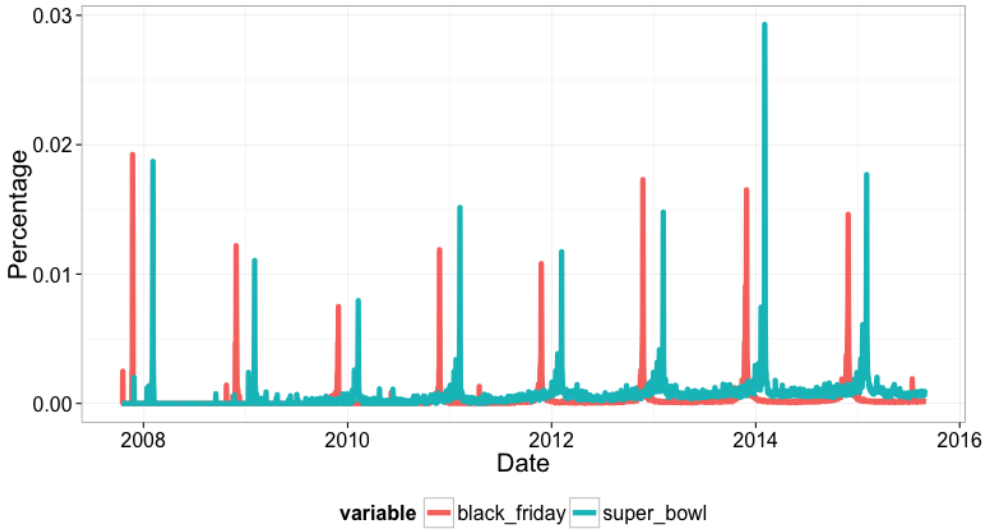

Figure 1.1: Usage of the terms "Black Friday" and "Super Bowl" on Reddit during an eight year time period.

Figure 1.1 shows the bursty variation of the frequency of two terms in eight years of text from Reddit. Mentions of these two events increase in likelihood by several orders of magnitude during predictable but narrow windows of time. This illustrates both why models that ignore context are brittle and why there is so much to be gained from adaptation.

Context is often represented by partitioning data into in-domain and out-of-domain sets. The in-domain data is considered to be sampled from the same or a similar distribution as the evaluation data and the out-of-domain data is everything else. A simple adaptation method is to build separate models for the in-domain corpus and out-of-domain corpora and then choose the interpolation weights to give the best performance on the in-domain data. There can be multiple out-of-domain corpora that are treated separately but typically no attempt is made to model the similarities or differences between domains/contexts. The problem with this adaptation approach is that the notions of discrete domains are much too crude when the goal is to mimic the fine-grained contextual adaptations that humans typically employ.

Category	Purpose
Topic	Adjust to variations in topic across documents or in speech. This is the most popular use case for adaptation techniques.
Temporal	Changing language usage patterns over time (Rosenfeld, 1995; Osborne et al., 2014; Yogatama et al., 2014)
Geographic	Variations in speaking style in different geographic regions (Eisenstein et al., 2010; Chelba et al., 2015; Halpern et al., 2016)
Modality	Adapt a model trained on written text for use in conversational speech (Bulyko et al., 2003; Jaech and Ostendorf, 2015; Mendels et al., 2015)
Language	Share information between similar languages (Ragni et al., 2016; Östling and Tiedemann, 2017)
TV Programs & Youtube Channels	Adapt to styles and topics of different television shows (Chen et al., 2015; Deena et al., 2016)
Lectures, Talks, & Meetings	Use text from slides, lecture titles, and other written materials to bias language model in speech recognition (Schwarm et al., 2004; Glass et al., 2007; Hsu and Glass, 2008; Hoang et al., 2016)
Dialog State	Generate an appropriate response given the dialog state (Riccardi and Gorin, 2000; Liu and Lane, 2017)
Personalization	Match the model predictions to the style of each individual from a large group (Tseng et al., 2015; Li et al., 2016)

Table 1.1: Example use cases for language model adaptation

In the newspaper example, instead of relying solely on a single binary variable indicating domain membership (AP vs. WSJ), additional contextual variables could have been used. For example, we could have a variable to indicate which section of the paper the article appeared in, or who the author was, or the date of publication. In general, the context representation can leverage multiple discrete and continuous variables—the more expressive the contextual representation, the greater the ability of the model to adapt to it. Ideally, a context-aware language model could model text using the style of one newspaper and the topic of a different publication. While researchers have tried to do this with class language models, equating word class sequences with style, many real-world applications are best characterized by interacting combinations of several contextual factors. Therefore, this thesis investigates models that can use rich context representations.

Language is often produced in association with some information about its context. For example, in speech recognition, if a user is speaking to a personal assistant then the system might know the time of day or the identity of the task that the user is trying to accomplish. If the user takes a picture of a sign to translate it with their smart phone, the system would have contextual information related to the geographic location and the user's preferred language. The probability of certain terms and phrases appearing can change dramatically with respect to geographic location (Chelba and Shazeer, 2015).

When adapting to context information, the language model conditions on both the context and the previous words in the sequence. It is computing $P(w_{1:n}|\text{context})$. The mechanism for computing $P(w_{1:n})$ impacts the approach for accounting for context. Recurrent neural networks (RNNs) have been shown to be very effective language models compared to previous approaches such as n-gram models or maximum entropy models (Mikolov et al., 2010), in part because they are able to make use of arbitrarily long word histories. RNN based language models are the focus of this thesis due to their recent successes and current widespread use. Improvements in adapting RNNs are likely to have a positive impact on multiple tasks. In addition, the continuous-space approach is well-suited to characterizing multiple contextual variables.

The standard method of adapting RNN language models is due to Mikolov and Zweig (2012) and involves learning an embedding to represent the context (originally the output of a topic model but any type of learned embedding will work) and including it via concatenation as an additional input to the model. We refer to this method of adapting the recurrent layer as the ConcatCell because of its reliance on concatenation of the context embedding. As we will show later on, when using this adaptation method most of the parameters of the model are static with respect to context. We propose a more powerful mechanism for using a context vector, which we call the FactorCell. Rather than simply using context as an additional input, it is used to control a factored (low-rank) transformation of the recurrent layer weight matrix. The motivation is that allowing a greater fraction of the model parameters to be adjusted in response to the input context will produce a model that is more adaptable and responsive to that context. In addition, we introduce a mechanism for handling new contexts that emerge after the model has been trained and deployed, showing how adaptation can be effective in these scenarios as well.

1.2 Key Contributions

The primary goal of this work is to increase the adaptability of recurrent neural network language models. Our main contribution is to propose a novel mechanism, which we call the FactorCell, for using a context embedding vector to transform the weights of the recurrent layer. This is a fundamentally different way of adapting the recurrent layer. Instead of viewing context as an additional input to the RNN, we create a function that uses context information to output the weights of a custom RNN that matches the given context. Moreover, by using precomputation and caching techniques, the FactorCell delivers superior adaptation at little to no extra computational cost.

We demonstrate the superiority of the FactorCell model over commonly used methods for RNN adaptation, including several recent approaches that do not adapt at the recurrent layer, preferring to focus on the output bias vector. Experiments on nine datasets with varying domains, contexts, and model and vocabulary sizes confirms that adapting the recurrent layer

always helps. We also show that our model beats the current standard method of adapting the recurrent layer. The extensive experimentation leads to some useful observations for predicting when context conditioning will be more or less successful for a given dataset.

The many prior use cases for adaptation mentioned in Section 1.1 all have in common the requirement that all contexts be known during training. Another contribution is to introduce an online learning method for adapting to contexts that emerge after the model has been deployed. Online learning improves the quality of the adaptation and also widens the set of possible applications. We use a personalized query auto-completion task to demonstrate how this method can be successfully used. Again, our FactorCell model beats the standard approach to adaptation and the performance gap widens as more data becomes available over time.

Adaptation is vital for language models to work in real world applications. We show how the FactorCell model impacts multiple applications and discuss potential implications for many more. The first application is the just mentioned personalized query completion. Another important application is context-specific or controllable text generation. We show how the FactorCell model is significantly more controllable than the standard adaptation methods. The gap between our model and the standard approach is not easily closed even when we give the baseline an advantage by doubling the dimensionality of its recurrent layer. Automatic evaluation of context-specific text generation can be difficult. We propose a metric based on text classification that is predictive of human ratings for text generation performance and avoids many of the headaches of prior evaluation techniques. For both the query completion and text generation applications we show through analysis that the FactorCell model has qualitative benefits that are not fully captured by perplexity as an evaluation criterion.

1.3 Thesis Overview

The remainder of the thesis proceeds as follows.

In Chapter 2, we provide background information on language modeling and adaptation.

We review relevant prior work in these areas including the ConcatCell and SoftmaxBias models from Mikolov and Zweig (2012) that serve as the principal baselines for the rest of the thesis.

In Chapter 3, we show that the ConcatCell model is effectively a constant additive bias in the recurrent layer. We then explore trade-offs of different architectures for adapting at the recurrent layer and the output layer. We make some observations about what factors make a dataset more or less amenable to adaptation.

In Chapter 4, we introduce the FactorCell, a more powerful model for RNN adaptation. By controlling the rank of a low-rank context-dependent weight transformation, the FactorCell can be adjusted to allow for more or less sharing of information between contexts depending on the situation. This model remedies the prominent weakness of the ConcatCell, namely, that it often does not allow its predictions to be changed enough in response to context. We show that the FactorCell beats the ConcatCell in terms of perplexity and that it also does better at capturing the relationship between context and language in text classification experiments.

In Chapter 5, we introduce an approach for adapting to new contexts that emerge after the model has been trained and deployed, and apply the FactorCell model to the task of personalized query auto-completion. The key result is that stronger adaptability at the recurrent layer enables the model to better take advantage of information from new users' query histories to personalize their predictions.

Chapter 6 deals with the use of language model adaptation for context-specific text generation. We measure context-specificity by checking if the text sampled from or generated by the model matches the properties of the specified context. We show that when the adaptation is weaker (as in the ConcatCell) then context-specificity suffers; the FactorCell model gives clear wins for context-specificity. We propose a metric based on text classification that is predictive of human ratings for text generation performance and avoids many of the headaches of prior evaluation techniques.

Finally, Chapter 7 concludes the thesis by summarizing the contributions and suggesting

future directions for additional research.

Chapter 2

BACKGROUND

This chapter gives the necessary background information on language modeling and prior literature on adaptation. The contributions of the thesis build on these ideas.

2.1 General Language Modeling Background

Language models compute a probability distribution over word sequences $w_{1:n}$ where each w_i is drawn from a vocabulary V. Typically, the probability is factored using the chain rule: $P(w_{1:n}) = P(w_1) \prod_{i=2}^{n} P(w_i|w_{1:i-1})$. The $w_{1:i-1}$ term is often referred to as the history.

Dealing with data sparseness is fundamental to language modeling because there will always be many valid word sequences that are not observed in the training data. One way to categorize language models is to look at how they deal with the sparseness or generalization problem: n-gram models use a back-off strategy, maximum entropy models rely regularization techniques, and the different classes of neural networks generalize by finding low dimensional continuous space representations of language.

In the basic n-gram language model only the most recent $n-1$ words from the history are considered (Bahl et al., 1978). In the common case that $n = 3$, known as a trigram model, $P(w_i|w_{1:i-1})$ is approximated as $P(w_i|w_{i-1}, w_{i-2})$. Even with this simplifying assumption, sparseness is a problem. Estimating the parameters of an n-gram language model using maximum likelihood fails to assign proper estimates to the n-grams that are unobserved in the training data, and there will be many such n-grams. The remedy is to borrow from the probability mass given to some of the observed n-grams and distribute it to unobserved ones (Katz, 1987). The redistribution is done recursively, reducing the size of the n-gram history in each step. The methods used for the back-off smoothing has been improved over time

(Kneser and Ney, 1995; **?**), but the basic idea has remained the same.

One improvement was to add skip-grams, which are like n-grams except they can skip over some words to look farther back in the history (Siu and Ostendorf, 2000; Shazeer et al., 2015). A variant of skip-grams is to identify long range triggers such as by using information from a syntactic parse (Bellegarda, 2000).

The class language model is an extension of the standard n-gram model that operates on a set of word equivalence classes (Brown et al., 1992). For a trigram, the probabilities are factored as $P(w_i|w_{i-1}, w_{i-2}) = P(w_i|c_i)P(c_i|c_{i-1}, c_{i-2})$. The use of classes reduces the effective size of the vocabulary when estimating $P(c_i|c_{i-1}, c_{i-2})$ and thereby improves the reliability of those estimates and possibly allowing for use of a higher n-gram order. This is the advantage of the class language model formulation. However the gain in reliability is associated with a loss of detail that can lead to mixed results.

The difficulty, for several decades, in finding a practical improvement over n-gram language models, despite their obvious weaknesses, was described as being a "source of considerable irritation" to researchers in the field (Jelinek, 1991). A decade later, Rosenfeld (2000) calls it ironic that "the most popular language models (n-grams) take no advantage of the fact that what's being modeled is language." And yet, even now with many alternatives to choose from, n-gram models continue to be heavily used. Their strengths are that they make few assumptions other than the Markov property, training the models is as fast as counting n-grams, and they use a vast number of parameters to memorize idiomatic and exceptional expressions.

2.1.1 Vocabulary & Input Representation

An important design choice in language modeling is to define the vocabulary. Typically, the vocabulary is set by taking the set of all words that appear at least k times in the training corpus for some small k. Over time, as more powerful computers and bigger text corpora became available, the typical vocabulary size used in applications has likewise increased. Figure 2.1 shows the change in vocabulary size for large vocabulary continuous speech recog-

nition systems over time with a doubling in vocabulary size approximately every four years. Similar increases are observed for other applications.

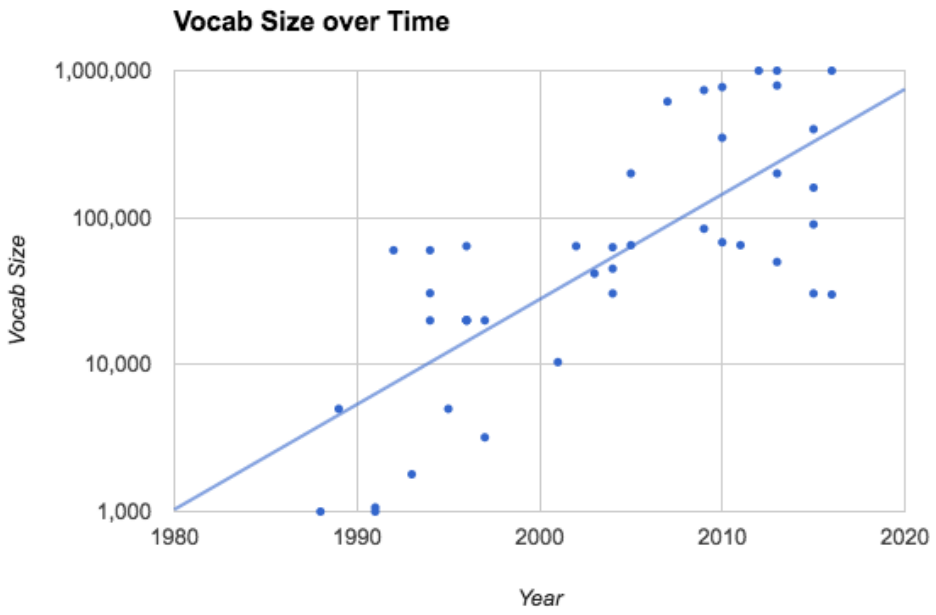

Figure 2.1: Vocabulary size in large vocabulary continuous speech recognition systems over time.

Exploding vocabulary sizes lead to exponential growth in model size and exponential increases in the time required for speech decoding. This trend is obviously not sustainable. Dealing with the exponential growth is a challenge but also an opportunity for considerable amounts on innovation.

Research is proceeding into neural systems that can decode audio directly into text one character at a time (Graves and Jaitly, 2014; Chan et al., 2015; Bahdanau et al., 2016; Maas et al., 2015). The language model can be a character n-gram model trained separately or an LSTM that is part of the larger neural network and trained end-to-end. When training end-to-end, the objective is to minimize the error rate instead of aiming to minimize perplexity. Machine translation is also moving away from using word-based language models (Lee et al.,

2017; Chung et al., 2016). Google's machine translation system uses a small vocabulary of around 30,000 tokens that are a mix of words and subwords (Wu et al., 2016).

Character-based models are far from a new concept. Subword models have found use before especially when working with highly inflected or low-resource languages (Tucker et al., 1994; Creutz et al., 2007; Saraçlar et al., 2010); however, there has been a recent surge of interest in character- and subword-based models. This thesis makes a point of testing on both word- and character-level models so as to have maximum impact on future applications.

2.1.2 Evaluation & Metrics

Language model evaluation is often done on a held-out test set using perplexity as a metric:

$$\text{perplexity} = \exp(-\frac{1}{N}\sum_{i=1}^{N}\log p(w_i)). \tag{2.1}$$

See Equation 2.1. Perplexity was always recognized as a "crude" metric (Jelinek, 1990, 1991) but is still widely used because it offers a quick and easy task-independent way of evaluating a language model.

Perplexity is not the only factor that matters when comparing models, however. Models can be interesting for other reasons such as speed (Brants et al., 2007). In some cases, the models are evaluated using downstream metrics like word error rate in speech recognition or BLEU score in machine translation (Kirchhoff and Yang, 2005) instead of using perplexity. For text generation, human evaluations can be high quality but are costly to perform. Perplexity will be the main evaluation metric used in this thesis due to the desire to be task-independent and the need to control costs, but some experiments with other metrics are included.

Recently, some language modeling papers have used "dynamic evaluation", whereby the model is allowed to continue to train on the test data after making predictions for each segment (Krause et al., 2017). This is a form of online updating and it helps to adapt to shifts between the training and the test data. Dynamic evaluation makes less sense for certain applications such as speech recognition because it can reinforce errors in the transcription.

Language models can be fairly compared without the use of dynamic evaluation. Thus, we do not make use of it in this thesis except that we do use a form of online updating for one of our experiments that will be introduced in Chapter 5, where the application makes sense.

2.2 Neural Language Modeling

Continuous-space language models such as the neural probabilistic language model (Bengio et al., 2003a) obtain an advantage over n-gram models because they share information between n-grams by projecting to a low-dimensional continuous space. Recurrent neural network (RNN) language models (Mikolov et al., 2010) extend that advantage by permitting the incorporation of information from arbitrarily long word histories. The RNN language model in its basic form has three layers: an input layer, a recurrent layer, and an output layer.

The input layer learns a word embedding matrix $\mathbf{E} \in \mathbb{R}^{p \times |V|}$ that consists of a p-dimensional vector for each word in the vocabulary V. If the input $w_1, w_2, \ldots w_n$ is represented as one-hot encoded vectors then multiplying by $\mathbf{E}$ gives a sequence of word embeddings, $e_{w_1}, e_{w_2}, \ldots e_{w_n} = \mathbf{E}w_1, \mathbf{E}w_2, \ldots \mathbf{E}w_n$. The recurrent layer uses a weight matrix $\mathbf{W} \in \mathbb{R}^{q \times (p+q)}$ and a bias vector $b_1 \in \mathbb{R}^q$ to transform the word embedding for the current step e_{w_t} and its own output from the previous step h_{t-1} into a hidden state vector h_t that summarizes the sequence up to that point. The formula is given by Equation 2.2 where σ is the activation function, typically the hyperbolic tangent or a restricted linear unit (ReLU):

$$h_t = \sigma(\mathbf{W}[e_{w_t}, h_{t-1}] + b_1). \tag{2.2}$$

The output layer uses the hidden state vector h_t to estimate a probability distribution y_t over the vocabulary for w_{t+1}:

$$y_t = \text{softmax}(\mathbf{E}^T h_t + b_2). \tag{2.3}$$

The bias vector $b_2 \in \mathbb{R}^{|V|}$ acts as a prior on the unigram distribution and is a crucial part of the output layer. If the word embedding size p is not the same as the recurrent layer

dimensionality q then a linear projection can be inserted and the word embedding matrix $\mathbf{E}$ from the input layer can be reused in the output layer to save on parameters and increase generalizability (Press and Wolf, 2017; Inan et al., 2016). In this case, the equation for the output layer would be $y_t = \text{Softmax}(\mathbf{E}^T\mathbf{P}h_t + b_2)$. The parameters of the model, $\mathbf{E}$, $\mathbf{W}$, b_1, and b_2, can all be learned jointly via backpropagation through time towards the objective of maximizing the likelihood of the data.

One limitation of neural language models, as originally proposed, is that the training time scales poorly with the size of the vocabulary. A variety of techniques were developed as workarounds. Neural models were trained to predict only the words from a subset of the vocabulary known as a shortlist (Schwenk, 2004), and the predictions from the neural model were interpolated with an n-gram model that could handle a full-sized vocabulary. Shortly thereafter, hierarchical neural models were developed. These train faster with only a small decrease in perplexity (Morin and Bengio, 2005). Techniques such as importance sampling (Bengio et al., 2003b) and noise contrastive estimation (Mnih and Kavukcuoglu, 2013) make it possible to quickly train full size vocabulary models without a hierarchy. The method used for training large vocabulary models in this thesis is the sampled softmax from Jean et al. (2014). The sampled softmax constrains the vocabulary to a small random subset when computing the loss for each sequence avoiding the need to backpropagate to the full vocabulary for every weight update. For a thorough review of methods for training large vocabulary neural language models, see (Chen et al., 2016).

The basic RNN has a flaw known as the vanishing/exploding gradient problem that prevents it from learning to use information from far back in the history (Pascanu et al., 2013). In practice, this is mitigated by using alternate architectures such as long short-term memory (LSTM) or the gated recurrent unit (GRU) (Sundermeyer et al., 2012). These architectures use gating mechanisms to control the flow of information and importantly they permit the preservation of information from the hidden state over time. Our experiments make extensive use of these RNN variants. Other techniques have been developed to further increase the stability of the RNN. In some of our experiments, we make use of layer normalization

which involves normalizing the first and second moments of the $\mathbf{W}[e_{w_t}, h_{t-1}] + b_1$ term in Equation 2.2 (Ba et al., 2016).

In the early 1990s it was noted that n-gram language models benefit from boosting the probability of words observed in previous utterances or in earlier parts of a document because word usage is "bursty" (Jelinek et al., 1991). Once a word is observed in a given document, the probability of seeing it again is greatly increased. Models that boost the probability of recently seen words are called cache language models (Kuhn and De Mori, 1990). These techniques have been extended to RNN language models by allowing the model to look back at the previous hidden states from the same sentence or document (Merity et al., 2017b; Grave et al., 2017a). Usually, state-of-the-art language modeling results are reported both with and without the inclusion of a cache, e.g. Merity et al. (2017a), because the gain from a cache is considered to be orthogonal to other modeling improvements. The experiments in this thesis are reported without the inclusion of a cache in order to focus on our contributions to adaptation.

2.3 Language Model Adaptation

There is a long history of adapting n-gram language models starting from early work on mixture modeling (DeMori and Federico, 1999; Bellegarda, 2004). Since this thesis builds on neural models, the review of prior work will only cover these methods. The long-established mixture techniques also apply to neural network models (Irie et al., 2018), but our focus is on techniques that are specific to neural network adaptation.

We are most interested in cases with explicit representations of context, however, one adaptation method, model fine-tuning, does not require the use of a context embedding. The language model is first trained on general background data and then learning is briefly continued on smaller in-domain data to "fine-tune" the weights (Gangireddy et al., 2016; Zhang et al., 2017). Fine-tuning suffers from the possibility of catastrophic forgetting, where the model loses access to the information it learned from training on the background data (Goodfellow et al., 2013). One way of dealing with catastrophic forgetting is to freeze portions

of the model during fine-tuning. Some approaches combine fine-tuning with the addition of a new linear transformation in between the hidden and output layers (Deena et al., 2016), or occasionally a non-linear transformation is used instead (Ma et al., 2017). This is motivated in part by similar approaches for adapting acoustic models in speech recognition (Gemello et al., 2007).

Adaptation of neural language models has two parts: representing context information and using the context representation to alter the model predictions. We discuss these next.

2.3.1 Context Representations

For neural language model adaptation, context information is typically represented as an embedding vector. Neural networks have the advantage in that they are quite versatile in the types of inputs that they can accept. Another advantage is that the network can learn the context representations as part of the end-to-end language modeling task. Some of the types of context that could be or have been used are

1. Topic model vectors that summarize the topic of long documents (Mikolov and Zweig, 2012).

2. Context is the title of a TED talk and it is represented as an embedding using either bag-of-words features or using an RNN or CNN (Hoang et al., 2016).

3. Context is a one-hot encoded vector indicating an Amazon product identifier and another one-hot vector indicating the sentiment of a review of that product (Tang et al., 2016).

The context information can be categorical, numeric, textual information, or a combination of these. It could even be composed of attributes that are predicted using a machine learning system based on other features. The majority of experiments in this thesis use categorical context variables but the methods all apply to other cases.

We assume the availability of contextual information (metadata or other side information) that is represented as a set of context variables $f_{1:n} = f_1, f_2, \ldots f_n$, from which we produce a k-dimensional representation in the form of an embedding, $c \in \mathbb{R}^k$. Each of the context variables, f_i, represents some type of information or metadata about the sequence and can be either categorical or numerical.

We adopt the strategy from Tang et al. (2016) of combining information from multiple context variables using a simple neural network. This strategy is well-suited for the types of context variables that we will see in our experiments, particularly high dimensional contexts such as hotel identity in a review or user identity in a query completion task. For each context variable f_i, we learn an associated embedding matrix $\mathbf{F}_i$, $i = 1, \ldots, n$. If $n = 1$ then the embedding can directly be used as the context representation. Otherwise, a single layer neural network is used to combine the embeddings from the individual variables.

$$c = \tanh(\sum_i \mathbf{M}_i \mathbf{F}_i f_i + b_0) \tag{2.4}$$

In some cases the tanh activation function is replaced with the ReLU function instead. $\mathbf{M}_i$ and b_0 are parameters learned by the model. The context embedding, c, is used for adapting both the hidden and the output layer of the RNN.

2.3.2 Adaptation Mechanisms

Mikolov and Zweig (2012) were the first to propose a method for adapting RNN language models by augmenting both Equations 2.2 and 2.3 with an extra term. The adaptation depends on having a summary of the context information contained in an embedding vector $c \in \mathbb{R}^k$. To adapt the recurrent layer they concatenate the context embedding c with the word embedding at every step of the input, which we show in the next chapter is equivalent to an additive context-dependent bias. We refer to this form of adaptation as the ConcatCell.

$$h_t = \sigma(\mathbf{W}[e_{w_t}, h_{t-1}, c] + b_1). \tag{2.5}$$

To adapt the output layer, an adaptation term $\mathbf{G}c$ is added,

$$y_t = \text{softmax}(\mathbf{E}^T h_t + \mathbf{G}c + b_2), \tag{2.6}$$

which has the effect of altering the softmax bias in a context dependent way. We refer to models that use this form of adaptation as the SoftmaxBias model. SoftmaxBias adaptation appears to be a reasonable approach in cases where the context concerns topic, since we know that the unigram distribution is often sufficient to capture topical information. As we will show later on, SoftmaxBias adaptation alone is not sufficient to get the best results. This is particularly obvious when dealing with character-level models, where the unigram distribution carries little information about topic or style.

In the special case where the context variable is discrete and of low cardinality then the bias vector can be adapted by directly learning independent bias vectors for each context, i.e. replacing the context embedding c with a one-hot encoded vector. When the cardinality of the context variables is high then learning independent bias vectors carries a high memory cost. Although it is not often framed in this way, the $\mathbf{G}c$ term acts as a low-rank approximation to the strategy of learning independent bias vectors. We will use both strategies in this thesis, as the situation warrants.

These two adaptation strategies have been adopted for a variety of tasks, including personalization, adapting to television show genres (Chen et al., 2015), adapting to long range dependencies in a document (Ji et al., 2015), etc. See Table 2.1 for a listing of more prior work, showing which of these methods were employed. As shown in the table, a variety of contexts have been used. Topic based adaptation is popular but categorical variables are also used like product identity or sentiment level. When doing personalization, the context can be an identifier for the person (Li et al., 2016), or alternatively the information can be given as a bag-of-words representation of the persons prior utterances or writings (Wen et al., 2013).

Few studies have tested the relative merits of adapting at the recurrent layer versus the output layer. Ji et al. (2015) compares the two approaches, which they refer to as ccDCLM

Model	CC	SB	Context
TopicRNN (Dieng et al., 2016)		X	Topic model
Context Aware Generation (Tang et al., 2016)		X	Product identity and sentiment
Generative Text Classification (Yogatama et al., 2017)		X	Miscellaneous
Controlling Style in Text Generation (Ficler and Goldberg, 2017)		X	Movie review stylistic features
Context Dependent LM (Mikolov and Zweig, 2012)	X	X	Topic model
Language Model Personalization (Wen et al., 2013)	X	X	Social media text
Multi-genre Speech Recognition (Chen et al., 2015)	X	X	Topic model
Contextual LSTM (Ghosh et al., 2016)	X		Topic model
Persona Based Conversation (Li et al., 2016)	X		Personalized response generation
Feature-based RNNLM adaptation (Deena et al., 2016)	X		Multi-genre broadcast speech

Table 2.1: Approaches to RNN language model adaptation in prior work. The X indicates the use of the ConcatCell (**CC**) or the SoftmaxBias (**SB**) adaptation strategy

.

and coDCLM. They find that both give similar perplexities. SoftmaxBias wins by 3% on one dataset and by less than 1% on the other. However, adapting the recurrent layer does better at an auxiliary sentence ordering task. Differences between models that are more prominent in other metrics besides perplexity is a theme that we will return to later on in the thesis. They do not test any models that adapt both the recurrent and output layers. Hoang et al. (2016) also consider adapting at the hidden layer vs. at the softmax layer. They report a small advantage towards adapting the output layer but the comparison is only made on a single dataset.

It should be noted that not all models fit cleanly into the above framework, although it is the dominant paradigm. The (Hoang et al., 2016) model differs from the SoftmaxBias approach because it use an extra perceptron layer in the output. Luan et al. (2016) use the recurrent layer bias approach plus an extra context-dependent linear projection in between the recurrent and the output layer. Wen et al. (2015) uses a custom gating architecture to adapt to dialogue states.

Chapter 3

EXPLORING CONTEXT-AWARE RNNS

In this chapter[1], we show that the popular RNN hidden layer adaptation strategy described in the previous chapter corresponds to a static additive bias term. Then, we study the impact of adapting RNNs at the recurrent layer versus the output layer using different architectures and techniques. Starting with the unadapted RNN that makes no use of context information, we consider two mechanisms each of adapting the recurrent layer and the output layer respectively. One of the methods for adapting the recurrent layer is a novel multiplicative rescaling of the hidden state dimensions. Using experiments on three datasets, we make some observations about what factors make a dataset more or less amenable to adaptation. These studies provided the groundwork that led to the proposal of the FactorCell model.

3.1 Additive Bias Adaptation

As described in Section 2.3.2, the standard approach to recurrent layer adaptation is to include (via concatenation) the context embedding as an additional input to the recurrent layer. When the context embedding is constant across the whole sequence, it is easy to show that this concatenation is equivalent to using a context-dependent bias at the recurrent layer:

$$\begin{aligned} h_t &= \sigma(\hat{\mathbf{W}}[e_{w_t}, h_{t-1}, c] + b_1) \\ &= \sigma(\mathbf{W}[e_{w_t}, h_{t-1}] + \mathbf{Q}c + b_1) \\ &= \sigma(\mathbf{W}[e_{w_t}, h_{t-1}] + b_1'), \end{aligned} \tag{3.1}$$

[1]The content of this chapter draws from our previously published work (Jaech and Ostendorf, 2017)

where $\hat{\mathbf{W}} = [\mathbf{W}\ \mathbf{Q}]$ and $b' = \mathbf{Q}c + b$ is the context-dependent bias, formed by adding a linear projection of the context embedding. Thus, for this scenario where the context only changes sporadically, concatenating the context embedding with the input to the recurrent layer is the same as using a context-dependent bias vector. Some people perform the concatenation explicitly despite its inefficiencies compared to directly learning the context-dependent bias because modern deep learning libraries do not always make it easy to alter the internal workings of the RNN or LSTM.

3.2 Models

We consider two methods each for adapting the recurrent and output layers respectively. In total there are $2^4 = 16$ total possible models that can be constructed by enabling or disabling each of the four adaptations. All of the models with adaptation incorporate a context embedding c using the method described in Section 2.3.1.

3.2.1 Adapting the recurrent layer

We consider two methods for adapting the recurrent layer. The first is the ConcatCell from Equation 3.1. It can be implemented by simply concatenating the context vector c with the word embedding e_{w_t} at each timestep at the input to the recurrent layer, but the effect of this adaptation is to apply a linear additive shift to the recurrent layer bias.

To increase the adaptability of the hidden layer, we introduce a context-dependent multiplicative rescaling of the hidden layer weights. The method is inspired from Ha et al. (2017), where a similar multiplicative scaling is used to dynamically adjusting the parameters of a language model in response to the previous words in the sentence. Using this row rescaling technique on top of the additive adaptation from Equation 2.4, the equation becomes

$$h_t = \sigma(\mathbf{C}c \odot \mathbf{W}[e_{w_t}, h_{t-1}] + \mathbf{Q}c + b_1) \tag{3.2}$$

where $\mathbf{C} \in \mathbb{R}^{q \times k}$ is a new model parameter and $\odot$ is the elementwise multiplication operator. The element-wise multiplication is a low-cost operation and can even be pre-calculated so

that model evaluation can happen with no extra computation compared to a vanilla RNN.

3.2.2 Adapting the output layer

One way of adapting the output layer is to let each context have its own bias vector. This requires the use of a matrix of size $|V| \times |C|$, where $|V|$ is the size of the vocabulary and $|C|$ is the total number of possible contexts. This may be intractable when both $|V|$ and $|C|$ are large. Mikolov and Zweig (2012) use a low-rank factorization of the adaptation matrix, replacing the $|V| \times |C|$ matrix with the product of a matrix $\mathbf{G}$ of size $|V| \times k$ and a context embedding c of size k.

$$y_t = softmax(\mathbf{E}^T h_t + \mathbf{G}c + b_2) \tag{3.3}$$

The total number of parameters is now a much more manageable $O(|V| + \sum_i |C_i|)$ instead of $O(\sum_i |V||C_i|)$, where C_i is the cardinality of the i-th context variable. The advantage of a low-rank adaptation is that it forces the model to share information between similar contexts. The disadvantage, is that important differences between similar contexts can be lost.

We employ feature hashing to reduce the memory requirements but retain some of the benefits of having an individual bias term for each context-word pair. The context-word pairs are hashed into buckets and individual bias terms are learned for each bucket. The hashing technique relies on having direct access to the context variables $f_{1:n}$. Representing context as a latent topic distribution precludes the use of this hashing adaptation.

The choice of hashing function is motivated by what is easy and fast to perform inside the Tensorflow computation graph framework[2]. If w is a word identifier (ID) and $f_{1:n}$ are context variable ID's, then the hash table index is computed as

$$h_i(w, f_i) = wr_0 + f_i r_i \bmod l \tag{3.4}$$

where l is the size of the hash table and r_0 and the r_i's are all fixed random integers. The

[2]Newer versions of Tensorflow make it easier to do feature hashing than what we describe here.

value of l is usually set to a large prime number. The function $H : \mathbb{Z} \rightarrow \mathbb{R}$ maps hash indices to hash values and is implemented as a simple array.

Since l is much smaller than the total number of inputs, there will be many hash collisions. Hash collisions are known to negatively effect the perplexity (Mikolov et al., 2011). To deal with this issue, we restrict the hash table to context-word pairs that are observed in the training data. A Bloom filter data structure records which context-word pairs are eligible to have entries in the hash table. The design of this data structure trades off a compact representation of set membership against a small probability of false positives (Bloom, 1970; Talbot and Brants, 2008; Xu et al., 2011). A small amount of false positives is relatively harmless in this application, because they do not impair the ability of the Bloom filter to eliminate almost all of the hash collisions.

The function $\beta : \mathbb{Z} \rightarrow [0, 1]$ is used by the Bloom filter to map hash indices to binary values.

$$B(w, f_i) = \prod_{j=1}^{16} \beta(h_{i,j}(w, f_i))$$

The hash functions $h_{i,j}$ are defined in the same way as the h_i's above except that they use distinct random integers and the size of the table, l, can be different. Because β is a binary function, the product $B(w, f_i)$ will always be zero or one. Thus, any word-context pairs not found in the Bloom filter will have their hash values set to zero.

The final expression for the hashed adaptation term is given by

$$\text{Hash}(w, f_{1:n}) = \sum_{i=1}^{n} H(h_i(w, f_i))B(w, f_i) \tag{3.5}$$

$$y_t = \text{softmax}(\mathbf{E}^T h_t + \mathbf{G}c + b_2 + Hash(w_t, f_{1:n})) \tag{3.6}$$

3.3 Data

The experiments make use of three corpora chosen to give a diverse perspective on adaptation in language modeling. Summary information on the training set for each source (Reddit, Twitter, and SCOTUS) is provided in Table 3.1 and each source is discussed individually

Source	Size	Vocabulary	Context (Dimensions)
Reddit	8,000K	68,000 words	Subreddit (5800)
Twitter	77K	194 chars	Language (9)
SCOTUS	864K	18,000 words	Case (1765), Speaker (2276), Role (3)

Table 3.1: Number of sentences, vocabulary size and context variables for the three corpora.

below. The Reddit and SCOTUS data are tokenized and lower-cased using the standard NLTK tokenizer (Bird et al., 2009).

Reddit Reddit is the world's largest online discussion forum and is comprised of thousands of active subcommunities dedicated to a wide variety of themes. Our training data is 8 million sentences (100 million words) from Reddit comments during the month of April 2015. Only the first sentence from each comment was used. The 68,000 word vocabulary is selected by taking all tokens that occur at least 20 times in the training data. The remaining tokens are mapped to a special UNK token leaving us with an out of vocabulary rate of 2.3%. The validation data and test data are each contain one eighth the number of sentences as the training data.

The context variable is the identity of the subreddit, i.e. community, that the comment came from. There are 5,800 subreddits with at least 50 training sentences. The remaining ones are grouped together in an UNK category. The largest subreddit occupies just 4.5% of the data. By using a large number of subreddits, we highlight an advantage of model adaptation which is to be able to use a single unified model instead of training thousands of separate models for each individual community. Similarly, using context dependent bias vectors for this data instead of the hash adaptation would require learning 400 million additional parameters.

Twitter The Twitter training data has 77,000 Tweets (848,000 words) each annotated with one of nine languages: English, German, Italian, Spanish, Portuguese, Basque, Catalan, Galician, and French. The corpus was collected by combining resources from published data for language identification tasks during the past few years. Tweets labeled as unknown, ambiguous, or containing code-switching were not included. The data is unbalanced across languages with more than 32% of the Tweets being Spanish and the smallest four languages (Italian, German, Basque, and Galician) each representing less than 1.5% of the total. There are 194 unique character tokens in the vocabulary. Graphemes that are surrogate-pairs in the UTF-16 encoding, such as emoji, are split into multiple vocabulary tokens. No preprocessing or tokenization is performed on this data except that newlines were replaced with spaces for convenience. The validation and test data have 12,000 and 15,000 Tweets respectively.

SCOTUS Approximately 864,000 utterances (16 million words) of training data spanning arguments from 1990-2011. These are speech transcripts from arguments before the United States Supreme Court. Utterances are labeled with the case being argued (n=1,765), the speaker ID (n=2,276), and the speaker role (justice, advocate, or unidentified). These three context variables are defined in the same way as in Hutchinson et al. (2013), where a small portion of this data was used in language modeling experiments. The vocabulary size is around 18,000 words. Utterances longer than 45 words (90th percentile) were split into smaller utterances.[3] The validation and test data were each one eighth the size of the training data.

3.4 Experiments

We used an LSTM with coupled input and forget gates for a 20% reduction in computation time (Greff et al., 2016). Dropout was used as a regularizer on the input and outputs of the recurrent layer as described in Zaremba et al. (2014). For the large vocabulary experiments,

[3]Occasionally, the advocates go on for hundreds of words without interruption. Including these utterances would slow down the training.

Parameter	Reddit	SCOTUS	Twitter
Batch Size	400	300	200
Word Embed.	200	200	30
LSTM Size	240	240	200
Dropout	0%	15%	10%
Neg. Samples	100	100	NA
Total Params.	14M	4M	300K

Table 3.2: Summary of Key Hyperparamters

we used a sampled softmax loss to speed up training.

A summary of the key hyperparameters for each class of experiments is given in Table 3.2. We conducted some preliminary experiments to tune the different hyperparameters for each dataset. Then, we fixed the values of these hyperparameters and only varied the adaptation method in each experiment. The total parameter column in this table is based on the unadapted model. Adapted models will have more parameters depending on the type of adaptation. When using hash adaptation of the output layer, the size of the Bloom filter is 100 million and the size of the hash table is 80 million. The model is implemented using the Tensorflow library. Optimization is done using Adam with a learning rate of 0.001. Each model trained in under three days using 8 CPU threads.

Although the model is trained as a language model, it can be used as a generative text classifier. The classification rule is given by $argmax_{f_i} \sum_j \log p(w_j|w_{1:j-1}, c_{k\neq i}, f_i)$. When there are multiple context variables, we treat all but one of them as known values and attempt to identify the unknown one. It is not necessary to compute the probabilities over the full vocabulary to do text classification. The sampled softmax criteria can be used to greatly speed up evaluation of the classifier provided that a) the same negative samples are reused for each class and b) the number of negative samples is increased to around 10% of the vocabulary.

Hidd.		Output				
×	+	**LR**	**Hash**	**PPL**	**ΔPPL**	**AUC**
N	N	N	N	75.2	–	–
N	N	N	Y	69.6	7.3%	76.5
N	N	Y	N	68.0	9.5%	75.5
N	N	Y	Y	66.9	11.0%	78.4
N	Y	N	N	68.4	9.0%	76.1
N	Y	N	Y	66.9	11.0%	78.9
N	Y	Y	N	68.0	9.6%	75.3
N	Y	Y	Y	**66.5**	**11.5%**	78.4
Y	N	N	N	69.4	7.7%	75.9
Y	N	Y	N	68.8	8.5%	75.9
Y	N	Y	Y	67.2	10.6%	78.9
Y	Y	N	N	69.0	8.2%	76.7
Y	Y	N	Y	67.5	10.2%	79.0
Y	Y	Y	N	68.3	9.1%	75.7
Y	Y	Y	Y	67.1	10.7%	**79.2**

Table 3.3: Perplexities and Classification Avg. AUCs for Reddit Models

3.4.1 Reddit Experiments

The size of the subreddit embeddings was set to 25. Table 3.3 gives the perplexities and average AUCs for subreddit detection for different adapted models. The evaluation data contains 60,000 sentences. For comparison, an unadapted 4-gram Kneser-Ney model trained on the same data has a perplexity of 119. The models with the best perplexity do not use multiplicative adaptation of the hidden layer, but it is useful in the detection experiments.

We can inspect the context embeddings learned by the model to see if it is exploiting sim-

Pittsburgh	Python	NBA
Atlanta	CSharp	Warriors
Montana	JavaScript	Rockets
MadisonWI	CPP_Questions	Mavericks
Baltimore	CPP	NBASpurs

Table 3.4: Nearest neighbors to selected subreddits in the context embedding space.

ilarities between subreddits in the way that we expect. Table 3.4 lists the nearest neighbors by Euclidean distance to three selected subreddits. The nearest neighbors are intuitively reasonable. For example, the closest subreddits to Pittsburgh are communities created for other big cities and states. The Python subreddit is close to other programming language communities, and the NBA subreddit is close to the communities for individual NBA teams.

The number of subreddits is large enough that apply a generative classifier to the full set is impractical. We used a smaller subset of subreddit and to avoid bias picked the same ones used in another study (Tran and Ostendorf, 2016). This caused the classification task to turn into a detection one and therefore the decision rule is slightly different from what was described above. The subreddit detection involves predicting the subreddit a given comment came from with eight subreddits to choose from (AskMen, AskScience, AskWomen, Atheism, ChangeMyView, Fitness, Politics, and Worldnews) and nine distractors (Books, Chicago, NYC, Seattle, ExplainLikeImFive, Science, Running, NFL, and TodayILearned). To make a classification decision we evaluate the perplexity of each comment under the assumption that it belongs to each of the eight subreddits. We use z-score normalization across the eight perplexities to create a score for each class. The predictions are evaluated by averaging the AUC of the eight individual ROC curves. The best model for the classification task uses all four types of adaptation. The multiplicative adaptation of the hidden layer is clearly useful for classification even though it does not help with perplexity.

The perplexities for selected large subreddits are listed in Table 3.5. It can be seen that

the relative gain from adaptation is largest when the topic of the subreddit is more narrowly focused. The biggest gains were achieved for subreddits dedicated to specific sports, TV shows, or video games. Whereas, the gains were smallest for subreddits like Videos or Funny for which the content tends to be more diverse. The knowledge that a sentence came from a pro-wrestling subreddit effectively provides more information about the text than the analogous piece of knowledge for the Pics or Videos subreddit. This would seem to indicate that further gains could be possible if additional contextual information could be provided. An alternative explanation, that subreddits with fewer sentences in the training data receive more benefit from adaptation, is not supported by the data.

3.4.2 Twitter experiments

The Twitter evaluation was done on a set of 14,960 Tweets. The language context embedding vector dimensionality was set to 8. When both the vocabulary and the number of contexts are small, as in this case, there is no danger of hash collisions. We disable the Bloom filter making the hash adaptation essentially equivalent to having context-dependent bias vectors.

Table 3.6 reports the results of the experiments on the Twitter corpus. We compute both the perplexity and measure the performance of the models on a language identification task. In terms of perplexity, the best models do not make use of the multiplicative hidden layer adaptation, consistent with the results from the Reddit corpus. In general, the improvement in perplexity from adaptation is small (less than 5%) on this corpus compared to our other experiments where we saw relative improvements two to four times as big. This is likely because the LSTM can figure out by itself which language it is modeling early on in the sequence, capture that in the hidden state, and adjust its predictions accordingly.

Our best model, using multiplicative adaptation of the hidden layer, achieves an accuracy of 94.2% on this task. That is a 19% relative reduction in the error rate from the best model without multiplicative adaptation.

Sometimes there can be little to no perplexity improvement between the unadapted and adapted models. This can be explained if the provided context variables are mostly redundant

Subreddit	Base. PPL	Adapt. PPL	ΔPPL	Description
FlashTV	90.5	68.2	24.6%	A popular TV show
shield	99.4	77.3	22.2%	A tv show
GlobalOffensive	97.1	79.3	18.3%	A PC video game
nba	103.3	86.4	16.3%	National Basketball Association
SquaredCircle	85.7	71.7	16.3%	Professional Wrestling
Fitness	50.1	42.3	15.5%	Exercise and fitness
hockey	85.5	72.4	15.2%	Professional hockey
leagueoflegends	71.1	61.0	14.3%	A PC video game
pcmasterrace	71.7	62.0	13.5%	PC gaming
nfl	84.2	74.0	12.2%	National Football League
AskWomen	62.1	55.3	10.9%	Questions for women
news	70.8	65.0	8.2%	General news stories and discussion
worldnews	85.7	79.7	7.1%	Global news discussion
AskMen	69.4	66.7	3.9%	Questions for men
gaming	79.0	76.1	3.7%	General video games interest group
pics	74.0	71.8	3.0%	Funny or interesting pictures
videos	62.9	61.1	2.9%	Funny or interesting videos
funny	72.6	70.8	2.5%	Sharing humorous content

Table 3.5: Comparison of perplexities per subreddit

Hidden		Output				
×	+	**LR**	**Hash**	**PPL**	**Acc.**	**F1**
N	N	N	N	6.44	–	–
N	N	N	Y	6.43	56.1	44.0
N	N	Y	N	6.37	49.7	36.6
N	N	Y	Y	6.34	57.0	44.5
N	Y	N	N	6.23	91.6	84.2
N	Y	N	Y	6.25	92.5	84.4
N	Y	Y	N	6.21	91.4	82.9
N	Y	Y	Y	**6.15**	92.8	85.2
Y	N	N	N	6.90	93.9	85.6
Y	N	N	Y	6.39	93.6	85.9
Y	N	Y	N	6.28	93.2	85.1
Y	N	Y	Y	6.31	93.7	86.1
Y	Y	N	N	6.28	92.5	84.7
Y	Y	N	Y	6.30	93.7	86.3
Y	Y	Y	N	6.54	**94.2**	**86.3**
Y	Y	Y	Y	6.35	93.3	85.9

Table 3.6: Results on Twitter data.

given the previous tokens in the sequence. To investigate this further, we trained a logistic regression classifier to predict the language using the state from the LSTM at the last time step on the unadapted model as a feature vector. Using just 30 labeled examples per class it is possible to get 74.6% accuracy. Furthermore, we find that a single dimension in the hidden state of the unadapted model is often enough to distinguish between different languages even though the model was not given any supervision signal. This finding is consistent with previous work that showed that individual dimensions of LSTM hidden states can be strong indicators of concepts like sentiment (Karpathy et al., 2015; Radford et al., 2017).

Figure 3.1 visualizes the value of the dimension of the hidden layer that is the strongest indicator of Spanish on three different code-switched tweets. Code-switching is not a part of the training data but it provides a compelling visualization of the ability of the unsupervised model to quickly recognize the language. The fact that it is so easy for the unadapted model to pick-up on the identity of the contextual variable fits with our explanation for the small relative gain in perplexity from the adapted models in these two tasks.

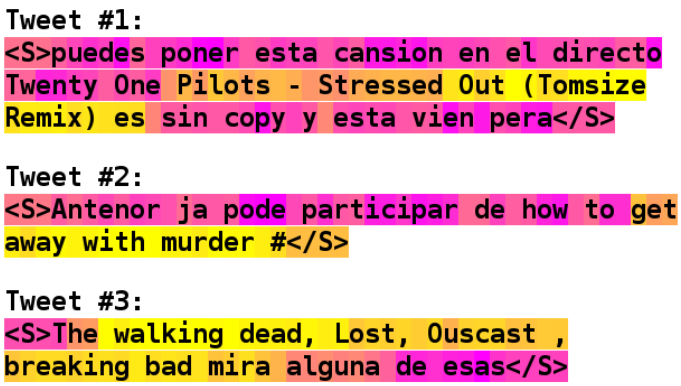

Figure 3.1: The value of the dimension of the LSTM hidden state in an unadapted model that is the strongest indicator for Spanish text for three different code-switched Tweets.

3.4.3 SCOTUS experiments

Table 3.7 lists the results for the experiments on the SCOTUS corpus. The size of the context embeddings are 9, 15, and 8 for the case, speaker, and role variables respectively. For calculating perplexity we use a 60,000 sentence evaluation set. For the classification experiment we selected 4,000 sentences from the test data from eleven different justices and attempted to classify the identity of the justice. The perplexity of the distribution of judges over those sentences is 8.9 (11.0 would be uniform). So, the data is roughly balanced. When classifying justices, the model is given the case context variable, but we do not make any special effort to filter candidates based on who was serving on the court during that time, i.e. all eleven justices are considered for every case.

For both the perplexity and classification metrics, the hash adaptation makes a big difference. The model that uses only hash adaptation and no hidden layer adaptation has a better perplexity than any of the model variants that use both hidden adaptation and low-rank adaptation of the output layer.

To ascertain which of the context variables have the most impact, we trained additional models with using different combinations of context variables. The model architecture is the one that uses all four forms of adaptation. Results are listed in Table 3.8. The most useful variable is the indicator for the case. The role variable is highly redundant—almost every speaker only appears in a single role. Therefore, it is not surprising that the speaker variable is more useful to the model than the role.

In Table 3.9 we list sentences generated from the fully adapted model (same one as the last line in Table 3.7) using beam search. The value of the context variable for the Case is held fixed while we explore different values for the Speaker and Role variables. Anecdotally, we see that the model captures some information about John Roberts role as chief justice. The model learns that Justice Breyer tends to start his questions with the phrase “I mean” while Justice Kagan tends to start with “Well”. Roberts and Kagan appear in our data both as justices and earlier as advocates.

Hidden		**Output**				
$\times$	$+$	**LR**	**Hash**	**PPL**	**ΔPPL**	**ACC**
N	N	N	N	37.3	–	–
N	N	N	Y	31.2	16.5%	29.6
N	N	Y	N	32.9	12.0%	26.2
N	N	Y	Y	30.0	19.6%	28.4
N	Y	N	N	33.2	11.0%	20.0
N	Y	N	Y	29.9	19.8%	26.8
N	Y	Y	N	32.7	12.4%	25.4
N	Y	Y	Y	29.8	20.3%	31.1
Y	N	N	N	33.3	10.7%	17.1
Y	N	N	Y	29.8	20.1%	28.4
Y	N	Y	N	32.3	13.4%	24.5
Y	N	Y	Y	**29.2**	**21.7%**	**32.4**
Y	Y	N	N	33.2	11.0%	18.9
Y	Y	N	Y	29.8	20.1%	29.6
Y	Y	Y	N	32.2	13.7%	26.1
Y	Y	Y	Y	29.4	21.1%	31.9

Table 3.7: Results on the SCOTUS data in terms of perplexity and classification accuracy (ACC) for the justice identification task.

Case	Spkr.	Role	PPL
N	N	N	37.3
N	N	Y	36.5
N	Y	N	33.6
N	Y	Y	33.3
Y	N	N	31.5
Y	N	Y	30.3
Y	Y	N	29.6
Y	Y	Y	29.4

Table 3.8: Perplexities for different combinations of context variables on the SCOTUS corpus.

Spkr.	Role	Sentence
Roberts	J.	`We'll hear argument first this morning in Ayers.`
Breyer	J.	`I mean, I don't think that's right.`
Kagan	J.	`Well, I don't think that's right.`
Kagan	A.	`Mr. Chief Justice, and may it please the court:`
Bork	A.	`--No, I don't think so, your honor.`

Table 3.9: Sentences generated from the adapted model using beam search under different assumptions for speaker and role contexts.

3.5 Comparison to Related Work

The multiplicative rescaling of the recurrent layer weights is used in the Hypernetwork model (Ha et al., 2017). The focus of this model is to allow the LSTM to adjust automatically depending on the context of the previous words. This is different from our work in that we are adapting based on contextual information external to the word sequence. Gangireddy et al. (2016) also use a rescaling of the hidden layer for adaptation but it is done as a fine-tuning step and not during training like our model.

The RNNME model from Mikolov et al. (2011) uses feature hashing to train a maximum entropy model alongside an RNN language model. The setup is similar to our method of using hashing to learn context-dependent biases. However, there are a number of differences. The motivation for the RNNME model was to speedup training of the RNN, not to compensate for the inadequacy of low-rank output layer adaptation, which had yet to be invented. Furthermore, Mikolov et al. (2011) do not use context dependent features in the max-ent component of the RNNME model, nor do they have a method for dealing with hash collisions such as our use of Bloom filters.

The idea of having one part of a language model be low-rank and another part to be an additive correction to the low-rank model has been investigated in other work (Eisenstein et al., 2011b; Hutchinson et al., 2013; Parikh et al., 2014). In both of these cases, the correction term is encouraged to be sparse by including an L1 penalty. Our implementation did not promote sparsity in the hash adaptation features but this idea is worth further consideration.[4] The hybrid LSTM and count based language model is an alternative way of correcting for a low-rank approximation (Neubig and Dyer, 2016).

3.6 Summary

While our results suggest that there is not a one-size-fits-all approach to language model adaptation, it is clear that we improve over the standard adaptation approach. The model

[4]See Appendix A for a deeper look at this idea.

from Mikolov and Zweig (2012), equivalent to using just additive adaptation on the hidden layer and low-rank adaptation of the output layer, is outperformed for all three datasets at both the language modeling and classification tasks. The combined low-rank and hash adaptation of the output layer were consistently required to get the best perplexity. For the classification tasks, the multiplicative hidden layer adaptation is clearly useful, as is the combined low-rank and hash adaptation of the output layer. Importantly, there is not always a strong relationship between perplexity and classification scores.

Our results may have implications for work on text generation where it can be more desirable to have more control over the generation rather than the lowest perplexity model. This issue is explored further in Chapter 6.

Our investigation of the language context in the Twitter experiments gives a useful take-away: context variables that are easily predictable from the text alone are unlikely to be helpful. More studies are needed to get a more complete understanding about what types of context variables will provide the most benefit. To that end, additional contexts are explored in subsequent chapters.

Based on the results from the SCOTUS experiments, we know that an additive transformation of the bias by itself is not always the best way to adapt the recurrent layer. This motivated us to look for a new approach that would give more consistent perplexity gains so that we could be confident recommending its use in most situations. Further investigation led us to develop the FactorCell model, which is the subject of the next chapter.

Chapter 4

FACTOR CELL MODEL

In this chapter we introduce the FactorCell model for adapting the recurrent layer of an RNN language model.[1] This is a major contribution of the thesis. Instead of taking context as an additional input to the model, we conceive of adaptation in a totally new way where the model generates a custom recurrent layer for any context. This is accomplished using a low-rank decomposition in order to control the extent of parameter sharing between contexts, which is important for handling high-dimensional, sparse contexts. The experiments in this chapter will show that the FactorCell improves perplexity and that it also has qualitative differences that set it apart from other models.

The FactorCell model generalizes the ConcatCell and remedies one of its major weaknesses. When there is a large amount of data available per context then there is less need to share information between contexts. Likewise, where there are many contexts and less training data per context it is better to do more parameter sharing. The ConcatCell is not able to trade-off between these scenarios. It always shares almost all of its parameters across contexts. In contrast, the FactorCell rank hyperparameter allows complete control over how much sharing there is between contexts.

Aside from perplexity, computation cost is always a consideration. A reliable and easy way to reduce perplexity is to increase the recurrent layer dimension. In latency-constrained applications (most industry speech recognition systems have strict latency constraints), the recurrent state dimension is limited. By design, the FactorCell permits pre-computation and caching so that its overall computational cost is negligibly more than the much simpler ConcatCell. All of the benefits of more adaptation are delivered with no extra latency.

---

[1] This chapter draws on content from some of our previously published work (Jaech and Ostendorf, 2018a).

The FactorCell is an alternative to the multiplicative transform explored in Chapter 3, which has the advantage of dedicating more parameters to the adaptation and affecting a bigger change on the recurrent layer weights. Unlike the multiplicative transform, we find that the FactorCell model consistently improves perplexity.

4.1 FactorCell Model

Our model uses adaptation in both the recurrent layer and in the bias vector of the output layer. In this section we describe methods for adapting the recurrent layer and the softmax layer, showing that our proposed model is a generalization of most prior methods.

4.1.1 Adapting the recurrent layer

Our proposed model extends the ConcatCell by using a context-dependent weight matrix $\mathbf{W}' = \mathbf{W} + \mathbf{A}$, in place of the generic weight matrix $\mathbf{W}$. (We refer to $\mathbf{W}$ as generic because it is shared across all context settings.) The adaptation matrix, $\mathbf{A}$, is generated by taking the product of the context embedding vector against a set of left and right basis tensors to produce a rank r matrix. The left and right adaptation basis tensors are given as $\mathbf{Z}_L \in \mathbb{R}^{k\times(p+q)\times r}$ and $\mathbf{Z}_R \in \mathbb{R}^{r\times q\times k}$. The two basis tensors together can be thought of as holding k different rank r matrices, $A_j = \mathbf{Z}_{L,j}\mathbf{Z}_{R,j}$, each the size of $\mathbf{W}$. By taking the product between c and the corresponding tensor modes of $\mathbf{Z}_L$ and $\mathbf{Z}_R$ (using $\times_i$ to denote the mode-i tensor product, i.e., the product with the i-th dimension of the tensor), the context determines the weighted combination of the k matrices:

$$\mathbf{A} = (c \times_1 \mathbf{Z}_L)(\mathbf{Z}_R \times_3 c^\intercal). \tag{4.1}$$

(Figure 4.1 is a visualization of the FactorCell architecture.) The number of degrees of freedom of $\mathbf{A}$ is controlled by the dimension k of the context vector and the rank r of the k weight matrices. The rank is treated as a hyperparameter and controls the extent to which the model relies on the generic weight matrix $\mathbf{W}$ versus behaves in a more context-specific manner.

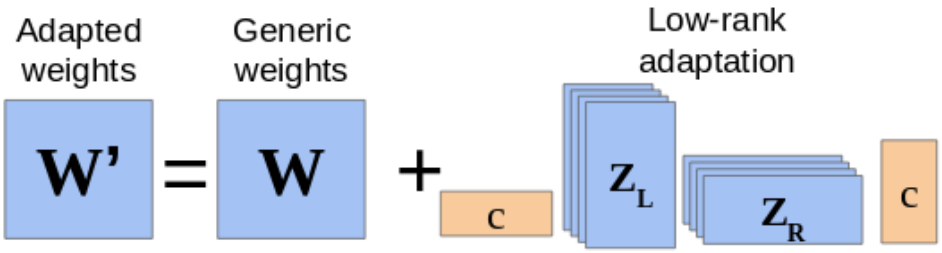

Figure 4.1: Illustration of the FactorCell architecture.

We call this model the FactorCell because the weight matrix has been adapted by adding a factored component. The ConcatCell model is a special case of the FactorCell where $\mathbf{Z}_L$ and $\mathbf{Z}_R$ are set to zero. In summary, the proposed model is given by:

$$\begin{aligned} h_t &= \sigma(\mathbf{W}'[e_{w_t}, h_{t-1}] + b_1') \\ \mathbf{W}' &= \mathbf{W} + (c \times_1 \mathbf{Z}_L)(\mathbf{Z}_R \times_3 c) \\ b_1' &= \mathbf{Q}c + b_1. \end{aligned} \tag{4.2}$$

If the context is known in advance, $\mathbf{W}'$ can be precomputed, in which case applying the RNN at test time requires no more computation than using an unadapted RNN of the same size. This means that for a fixed sized recurrent layer, the FactorCell model can have many more parameters than the ConcatCell model but hardly any increase in computational cost.

When adapting with the FactorCell method, we find it necessary to also include the shift of the bias in the softmax output as described in Equation 2.6.

4.1.2 LSTM FactorCell Equations

Only trivial changes are needed to use the FactorCell method on an LSTM instead of a vanilla RNN. Here, we list the equations for an LSTM with coupled input and forget gates, which is what was used in our experiments.

The weight matrix $\mathbf{W}$ from Equation 2.2 is now size $3q \times (p + q)$ and b is dimension $3q$, where 3 is the number of gates. Likewise, $\mathbf{Z}_R$ from Equation 4.1 is made to be of size

$r \times 3q \times k$. The weight matrix $\mathbf{W}'$ is as defined in Equation 4.2 and after computing its product with the input $[w_t, h_{t-1}]$, the result is split into three vectors of equal size: i_t, f_t, and o_t

$$[i_t, f_t, o_t] = \mathbf{W}'[e_{w_t}, h_{t-1}] + b_1, \tag{4.3}$$

which are used in the input gate, the forget gate, and the output gate, respectively.

Using these three vectors we perform the gating operations to compute h_t using the memory cell m_t as follows:

$$\begin{aligned} f_t &\leftarrow \text{sigmoid}(f_t + 1.0) \\ m_t &= m_{t-1} \odot f_t + (1.0 - f_t) \odot \tanh(i_t) \\ h_t &= \tanh(m_t) \odot \text{sigmoid}(o_t) \end{aligned} \tag{4.4}$$

Note that Equation 3.1, which shows that a context vector concatenated with input is equivalent to an additive bias term, extends to equation 4.3. In other words, in the LSTM version of the ConcatCell model, the context vector effectively introduces an extra bias term for each of the three gates.

4.2 Data

Name	Train	Dev	Test	Vocab	Docs.	Context
AGNews	4.6M	0.2M	0.3M	54,492	115K	4 Newspaper sections
DBPedia	28.7M	0.3M	3.6M	84,341	555K	14 Entity categories
TripAdvisor	127.2M	2.6M	2.6M	88,347	843K	3.5K Hotels/5 Sentiment
Yelp	91.5M	0.7M	7.1M	57,794	645K	5 Sentiment
EuroTwitter*	5.3M	0.8M	1.0M	194	80K	9 Languages
GeoTwitter*	51.7M	2.2M	2.2M	203	604K	Latitude & Longitude

Table 4.1: Dataset statistics: Dataset size in words (* or characters) of Train, Dev and Test sets, vocabulary size, number of training documents, and context variables.

The experiments make use of six datasets: four targeting word-level sequences, and two targeting character sequences. The character studies are motivated by the growing interest in character-level models in both speech recognition and machine translation (Hannun et al., 2014; Chung et al., 2016). By using multiple datasets with different types of context, we hope to learn more about what makes a dataset amenable to adaptation. The datasets range in size from over 100 million words of training data to 5 million characters of training data for the smallest one. When using a word-based vocabulary, we preprocess the data by lowercasing, tokenizing and removing most punctuation. We also truncate sentences to be shorter than a maximum length of 60 words for AGNews and DBPedia and 150 to 200 tokens for the remaining datasets. Summary information is provided in Table 4.1, including the training, development, and test data sizes in terms of number of tokens, vocabulary size, number of training documents (i.e. context samples), and the context variables ($f_{1:n}$). The largest dataset, TripAdvisor, has over 800 thousand hotel review documents, which adds up to over 125 million words of training data.

The first three datasets (AGNews, DBPedia, and Yelp) have previously been used for text classification (Zhang et al., 2015). These consist of newspaper headlines, encyclopedia entries, and restaurant and business reviews, respectively. The context variables associated with these correspond to the newspaper section (world, sports, business, sci & tech) for each headline, the page category on DBPedia (out of 14 options such as actor, athlete, building, etc.), and the star rating on Yelp (from one to five). For AgNews, DBPedia, and Yelp we use the same test data as in previous work. Our fourth dataset, from TripAdvisor, was previously used for language modeling and consists of two relevant context variables: an identifier for the hotel and a sentiment score from one to five stars (Tang et al., 2016). Some of the reviews are written in French or German but most are in English. There are 4,333 different hotels but we group all the ones that do not occur at least 50 times in the training data into a single entity, leaving us with around 3,500. These four datasets use word-based vocabularies.

We also experiment on two Twitter datasets: EuroTwitter and GeoTwitter. EuroTwitter

is the same as the Twitter data used in the previous chapter and consists of 80 thousand Tweets labeled with one of nine languages: (English, Spanish, Galician, Catalan, Basque, Portuguese, French, German, and Italian). The corpus was created by combining portions of multiple published datasets for language identification including Twitter70 (Jaech et al., 2016), TweetLID (Zubiaga et al., 2014), and the monolingual portion of Tweets from a code-switching detection workshop (Molina et al., 2016). The GeoTwitter data contains Tweets with latitude and longitude information from England, Spain, and the United States.[2] The latitude and longitude coordinates are given as numerical inputs. This is different from the other five datasets that all use categorical context variables.

4.3 Experiments with Different Contexts

The goal of the experiments is to show that the FactorCell model can deliver improved performance over current approaches for multiple language model applications and a variety of types of contexts. Specifically, results are reported for context-conditioned perplexity and generative model text classification accuracy, using contexts that capture a range of phenomena and dimensionalities.

Test set perplexity is the most widely accepted method for evaluating language models, both for use in recognition/translation applications and generation. It has the advantage that it is easy to measure and is widely used as a criterion for model fit, but the limitation that it is not directly matched to most tasks that language models are directly used for. Text classification using the model in a generative classifier is a simple application of Bayes rule:

$$\hat{\omega} = \arg\max_{\omega} p(w_{1:T}|\omega)p(\omega) \tag{4.5}$$

where $w_{1:T}$ is the text sequence, $p(\omega)$ is the class prior, which we assume to be uniform. Classification accuracy provides additional information about the power of a model, even if it is not being designed explicitly for text classification. Further, it allows us to be able to directly compare our model performance against previously published text classification

[2]Data was accessed from http://followthehashtag.com.

benchmarks. Although the most effective models for text classification have generally been discriminative, generative models can be competitive when the available training data is small or text samples are short (Yogatama et al., 2017), and we find that the FactorCell makes the generative model more competitive.

Note that the use of classification accuracy for evaluation here involves counting errors associated with applying the generative model to independent test samples. This differs from the accuracy criterion used for evaluating context-sensitive language models for text generation based on a separate discriminative classifier trained on generated text (Ficler and Goldberg, 2017; Hu et al., 2017). We discuss this further in Section 4.5 and Chapter 6.

The experiments compare the FactorCell model (equations 4.2 and 2.6) to two popular alternatives, which we refer to as ConcatCell (equations 2.5 and 2.6) and SoftmaxBias (equation 2.6). As noted earlier, the SoftmaxBias method is a simplification of the ConcatCell model, which is in turn a simplification of the FactorCell model. The SoftmaxBias method impacts only the output layer and thus only unigram statistics. Since bag-of-word models provide strong baselines in many text classification tasks, we hypothesize that the SoftmaxBias model will capture much of the relative improvement over the unadapted model for word-based tasks. However, in small vocabulary character-based models, the unigram distribution is unlikely to carry much information about the context, so adapting the recurrent layer should become more important in character-level models. We expect that performance gains will be greatest for the FactorCell model for sources that have sufficient structure and data to support learning the extra degrees of freedom.

Another possible baseline would use models independently trained on the subset of data for each context. This is the "independent component" case in (Yogatama et al., 2017). This will fail when a context variable takes on many values (or continuous values) or when training data is limited, because it makes poor use of the training data, as shown in that study. While we do have some datasets where this approach is plausible, we feel that its limitations have been clearly established.

4.3.1 Implementation Details

The RNN variant that we use is an LSTM with coupled input and forget gates (Melis et al., 2018). The different model variants are implemented[3] using the Tensorflow library. The model is trained with the standard negative log likelihood loss function, i.e. minimizing cross entropy. Dropout was used as a regularizer in the recurrent connections (Semeniuta et al., 2016). Training is done using the Adam optimizer with a learning rate of 0.001. For the models with word-based vocabularies, a sampled softmax loss is used with a unigram proposal distribution and sampling 150 words at each time-step (Jean et al., 2014). The classification experiments use a sampled softmax loss with a sample size of 8,000 words. This is an order of magnitude faster to compute with a minimal effect on accuracy.

	AgNews	**DBPedia**	**EuroTwtr**	**GeoTwtr**	**Trip**	**Yelp**
Word Embed	150	114-120	35-40	42-50	100	200
LSTM dim	110	167-180	250	250	200	200
Steps	4.1-5.5K	7.5-8.0K	6.0-8.0K	6.0-11.1K	8.4-9.9K	7.2-8.8K
Dropout	0.5	1.00	0.95-1.00	0.99-1.00	0.97-1.00	1.00
Ctx. Embed	2	12	3-5	8-24	20-30	2-3
Rank	12	19	2	20	12	9

Table 4.2: Selected hyperparameters for each dataset. When a range is listed it means that a different values were selected for the FactorCell, ConcatCell, SoftmaxBias or Unadapted models.

Hyperparameter tuning was done based on minimizing perplexity on the development set and using a random search. Hyperparameters included word embedding size e, recurrent state size d, context embedding size k, and weight adaptation matrix rank r, the number of training steps, recurrent dropout probability, and random initialization seed. We conducted more than 700 tuning experiments with iterative refinements. The number of experiments

[3]Code available at http://github.com/ajaech/calm.

per dataset vaires between 74 and 190. The selected hyperparameter values are listed in Table 4.2. For any fixed LSTM size, the FactorCell has a higher count of learned parameters compared to the ConcatCell. However, during evaluation both models use approximately the same number of floating-point operations because $\mathbf{W}'$ only needs to be computed once per sentence. Because of this, we believe limiting the recurrent layer cell size is a fair way to compare between the FactorCell and the ConcatCell.

4.3.2 Word-based Models

	AGNews		**DBPedia**		**TripAdvisor**		**Yelp**	
Model	PPL	ACC	PPL	ACC	PPL	ACC	PPL	ACC
Unadapted	96.2	–	44.1	–	51.6	–	67.1	–
SoftmaxBias	95.1	**90.6**	40.4	95.5	48.8	51.9	66.9	51.6
ConcatCell	93.8	89.7	39.5	97.8	48.3	56.0	66.8	56.9
FactorCell	**92.3**	**90.6**	**37.7**	**98.2**	**48.2**	**58.2**	**66.2**	**58.8**

Table 4.3: Perplexity and classification accuracy on the test set for the four word-based datasets.

Perplexities and classification accuracies for the four word-based datasets are presented in Table 4.3. In each of the four datasets, the FactorCell model gives the best perplexity. For classification accuracy, there is a bigger difference between the models, and the FactorCell model is the most accurate on three out of four datasets and tied with the SoftmaxBias model on AgNews. For DBPedia and TripAdvisor, most of the improvement in perplexity relative to the unadapted case is achieved by the SoftmaxBias model with smaller relative improvements coming from the increased power of the ConcatCell and FactorCell models. For Yelp, the perplexity improvements are small; the FactorCell model is just 1.3% better than the unadapted model.

From (Yogatama et al., 2017), we see that for AGNews, much more so than for other

datasets, the unigram statistics capture the discriminating information, and it is the only dataset in that work where a naive Bayes classifier is competitive with the generative LSTM for the full range of training data. The fact that the SoftmaxBias model gets the same accuracy as the FactorCell model on this task suggests that topic context may benefit less from adapting the recurrent layer.

For the DBPedia and Yelp datasets, the FactorCell model beats previously reported classification accuracies for generative models (Yogatama et al., 2017). However, it is not competitive with state-of-the-art discriminative models on these tasks with the full training set. With less training data, it probably would be, based on the results in (Yogatama et al., 2017).

The numbers in Table 4.3 do not adequately convey the fact that there are hyperparameters with an effect on perplexity that is greater than the sometimes small relative differences between models. Even the seed for the random weight initialization can have a "major impact" on the final performance of an LSTM (Reimers and Gurevych, 2017). We use Figure 4.2 to show how the three classes of models perform across a range of hyperparameters. The figure compares perplexity on the x-axis with accuracy on the y-axis with both metrics computed on the development set. Each point in this figure represents a different instance of the model trained with random hyperparameter settings and the best results are in the upper right corner of each plot. The color/shape differences of the points correspond to the three classes of models: FactorCell, ConcatCell, and SoftmaxBias.

Within the same model class but across different hyperparameter settings, there is much more variation in perplexity than in accuracy. The LSTM cell size is mainly responsible for this; it has a much bigger impact on perplexity than on accuracy. It is also apparent that the models with the lowest perplexity are not always the ones with the highest accuracy. Notably, improvements in perplexity are associated with a decrease in accuracy for the SoftmaxBias models on the DBPedia, TripAdvisor, and Yelp datasets. In Bowman et al. (2016), it was observed that the more powerful the decoder of a variational auto-encoder was the more likely it was to ignore the prior information given by the encoder. A similar effect

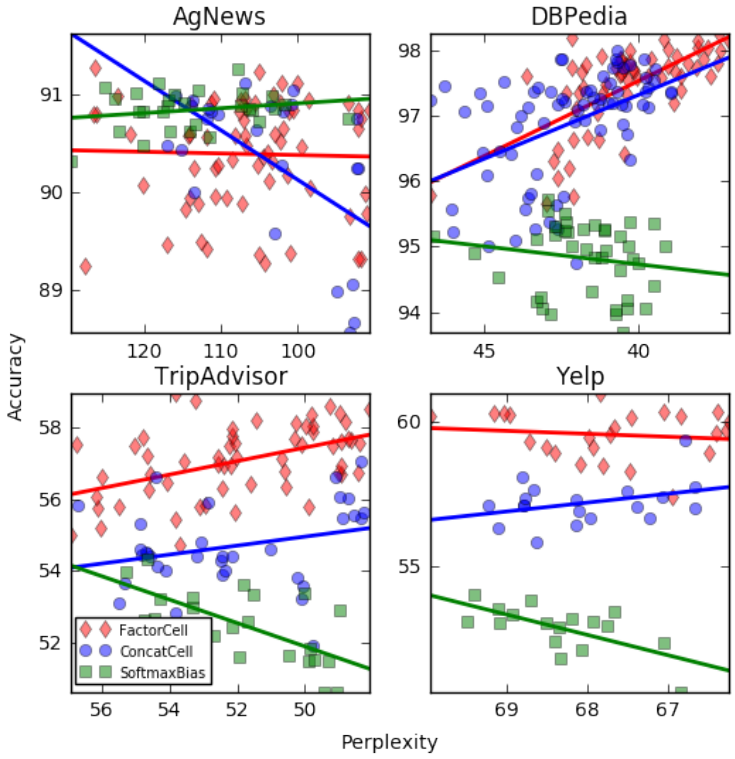

Figure 4.2: Accuracy vs. perplexity for different classes of models on the four word-based datasets.

is happening here where when the recurrent layer size is increased then the model relies less on the unigram prior provided by the adapted softmax bias vector. See Section 4.3.4 for further hyperparameter analysis.

Figure 4.3 is a visualization of the per-word log likelihood ratios between a model assuming a 5 star review and the same model assuming a 1 star review. Likelihoods were computed using an ensemble of three models to reduce variance. The analysis is repeated for each class of model. Words highlighted in blue are given a higher likelihood under the 5 star assumption.

Unigrams with strong sentiment such as "lovely" and "friendly" are well-represented by all three models. The reader may not consider the tokens "craziness" or "5-8pm" to be strong indicators of a positive review but the way they are used in this review is representative of

how they are typically used across the corpus.

As expected, the ConcatCell and FactorCell model capture the sentiment of multi-token phrases. As an example, the unigram "enough" is 3% more likely to occur in a 5 star review than in a 1 star review. However, "do enough" is 30 times more likely to appear in a 5 star review than in a 1 star review. In this example, the FactorCell model does a better job of handling the word "enough."

4.3.3 Character-based Models

Next, we evaluate the EuroTwitter and GeoTwitter models using both perplexity and a classification task. For EuroTwitter, the classification task is to identify the language. With GeoTwitter, it is less obvious what the classification task should be because the context values are continuous and not categorical. We selected six cities and then assigned each sentence the label of the closest city in that list while still retaining the exact coordinates of the Tweet. There are two cities from each country: Manchester, London, Madrid, Barcelona, New York City, and Los Angeles. Tweets from locations further than 300 km from the nearest city in the list were discarded when evaluating the classification accuracy. The classification task is sufficient to investigate the properties of the language model but, unlike some prior work, it is not designed to capture geographic lexical variations in an easily interpretable manner (Eisenstein et al., 2010) nor is it designed to be efficient at geolocation (Han et al., 2014).

Perplexities and classification accuracies are presented in Table 4.4. The FactorCell model has the lowest perplexity and the highest accuracy for both datasets. Again, the FactorCell model clearly improves on the ConcatCell as measured by classification accuracy. Consistent with our hypothesis, adapting the softmax bias is not effective for these small vocabulary character-based tasks. The SoftmaxBias model has small perplexity improvements ($< 1\%$) and low classification accuracies.

Figure 4.4 compares perplexity and classification accuracy for different hyperparameter settings of the character-based models. Again, we see that it is possible to trade-off some

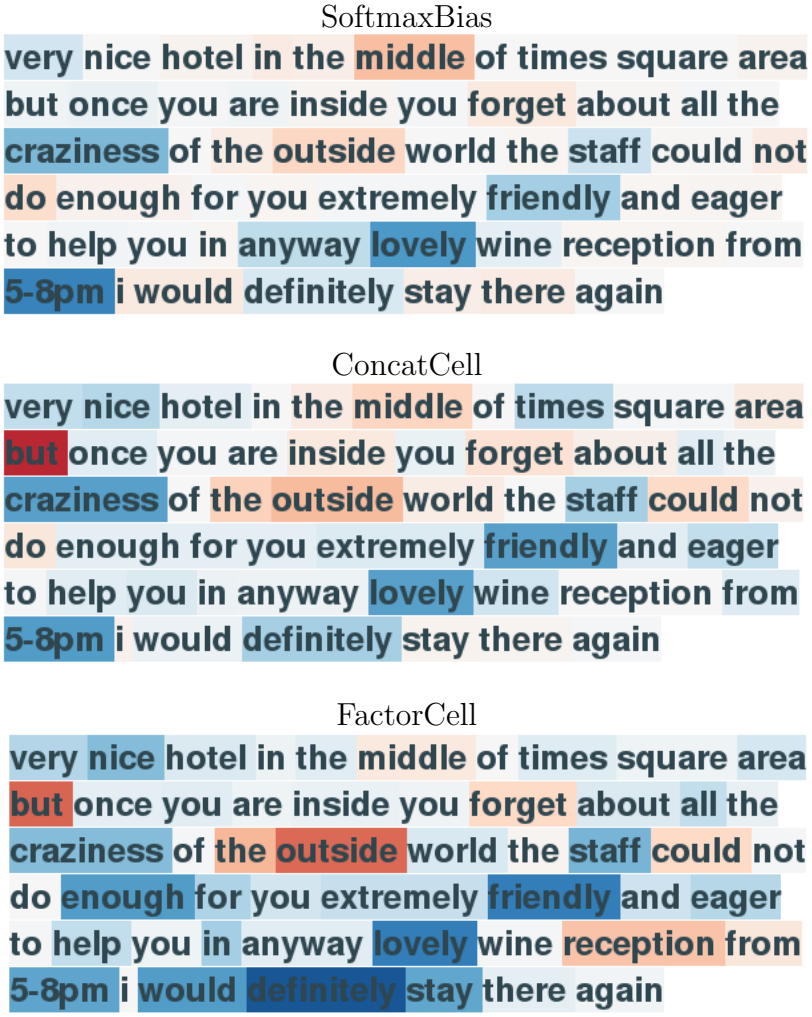

Figure 4.3: Log likelihood ratio between a model that assumes a 5 star review and the same model that assumes a 1 star review. Blue indicates a higher 5 star likelihood and red is a higher likelihood for the 1 star condition.

	EuroTwitter		GeoTwitter	
Model	PPL	ACC	PPL	ACC
Unadapted	6.35	–	4.64	–
SoftmaxBias	6.29	43.0	4.63	29.9
ConcatCell	6.17	91.5	4.54	42.2
FactorCell	**6.07**	**93.3**	**4.52**	**63.5**

Table 4.4: Perplexity and classification accuracies for the EuroTwitter and GeoTwitter datasets.

perplexity for gains in classification accuracy. For EuroTwitter, if tuning is done on accuracy rather than perplexity then the accuracy of the best model is as high as 95%.

4.3.4 Hyperparameter Analysis

The hyperparameter with the strongest effect on perplexity is the size of the LSTM. This was consistent across all six datasets. The effect on classification accuracy of increasing the LSTM size was mixed. Increasing the context embedding size generally helped with accuracy on all datasets, but it had a more neutral effect on TripAdvisor and Yelp and increased perplexity on the two character-based datasets. For the FactorCell model, increasing the rank of the adaptation matrix tended to lead to increased classification accuracy on all datasets and seemed to help with perplexity on AGNews, DBPedia, and TripAdvisor.

Figure 4.5 compares the effect on perplexity of the LSTM parameter count and the FactorCell rank hyperparameters. Each point in those plots represents a separate instance of the model with varied hyperparameters. In the right subplot of Figure 4.5, we see that increasing the rank hyperparameter improves perplexity. This is consistent with our hypothesis that increasing the rank can let the model adapt more. The variance is large because differences in other hyperparameters (such as hidden state size) also have an impact.

In the left subplot we compare the performance of the FactorCell with the ConcatCell as

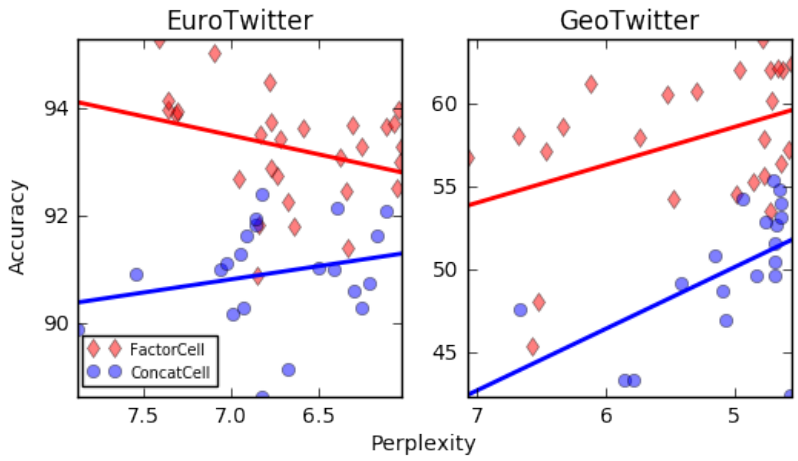

Figure 4.4: Accuracy vs. Perplexity for different classes of models on the two character-based datasets.

the size of the word embeddings and recurrent state change. The x-axis is the size of the $\mathbf{W}$ recurrent weight matrix, specifically $3(e+d)d$ for an LSTM with 3 gates. Since the adapted weights can be precomputed, the computational cost is roughly the same for points with the same x-value. For a fixed-size hidden state, the FactorCell model has a better perplexity than the ConcatCell.

Since performance can be improved both by increasing the recurrent state dimension and/or by increasing rank, we examined the relative benefits of each. The perplexity of a FactorCell model with an LSTM size of 120K will improve by 5% when the rank is increased from 0 to 20. To get the same decrease in perplexity by changing the size of the hidden state would require 160K parameters, resulting in a significant computational advantage for the FactorCell model.

Using a one-hot vector for adapting the softmax bias layer in place of the context embedding when adapting the softmax bias vector tended to have a large positive effect on accuracy leaving perplexity mostly unchanged. Recall from Section 2.3 that if the number of values that a context variable can take on is small then we can allow the model to choose between using the low-dimensional context embedding or a one-hot vector. This option is

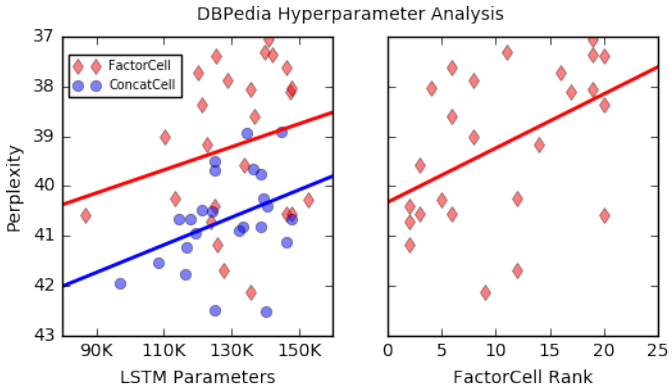

Figure 4.5: Comparison of the effect of LSTM parameter count and FactorCell rank hyperparameters on perplexity for DBPedia.

not available for the TripAdvisor and the GeoTwitter datasets because the dimensionality of their one-hot vectors would be too large.[4] The method of adapting the softmax bias is the main explanation for why some ConcatCell models performed significantly above/below the trendline for DBPedia in Figure 4.2.

We experimented with an additional hyperparameter on the Yelp dataset, namely the inclusion of layer normalization (Ba et al., 2016). (We had ruled-out using layer normalization in preliminary work on the AGNews data before we understood that AGNews is not representative, so only one task was explored here.) Layer normalization significantly helped the perplexity on Yelp ($\approx$ 2% relative improvement) and all of the top-performing models on the held-out development data had it enabled.

4.4 Analysis for Sparse Contexts

The TripAdvisor data is an interesting case because the original context space is high dimensional (3500 hotels $\times$ 5 user ratings) and sparse. Since the model applies end-to-end learning, we can investigate what the context embeddings learn. In particular, we looked at location (hotels are from 25 cities in the United States) and class of hotel, neither of which

[4]Another option for the TripAdvisor data is to use the hashing method from Section 3.2.2.

are input to the model. All of what it learns about these concepts come from extracting information from the text of the reviews.

To visualize the embedding, we used a 2-dimensional principal component analysis (PCA) projection of the embeddings of the 3500 hotels. We found that the model learns to group the hotels based on geographic region; the projected embeddings for the largest cities are shown in Figure 4.6, plotting the 1.5σ ellipsoid of the Gaussian distribution of the points. (Actual points are not shown to avoid clutter.) Not only are hotels from the same city grouped together, cities that are close geographically appear close to each other in the embedding space. Cities in the Southwest appear on the left of the figure, the West coast is on top and the East coast and Midwest is on the right side. This is likely due in part to the impact of the region on activities that guests may mention, but there also appears to be a geographic sampling bias in the hotel class that may impact language use.

Figure 4.7 shows the projected hotel embeddings in the same space as Figure 4.6 except that the points are now colored based on the hotel class. Class is a rating from an independent agency that indicates the level of service and amenities that customers can expect to receive at a hotel. Whereas, the star rating is the average score given to each establishment by the customers who reviewed it. Hotel class does not determine star rating although they are correlated ($r = 0.54$). The dataset does not contain a uniform sample of hotel classes from each city. The hotels included from Boston, Chicago, and Philly are almost exclusively high class and the ones from L.A. and San Diego happen to be low class, so the embedding distributions also reflect hotel class: lower class hotels towards the top left and higher class hotels towards the bottom right. The visualization for the ConcatCell and SoftmaxBias models are similar.

Another way of understanding what the context embeddings represent is to compute the softmax bias projection $\mathbf{G}c$ and examine the words that experience the biggest increase in probability. We show three examples in Table 4.5. In each case, the top words are strongly related to geography and include names of neighborhoods, local attractions, and other hotels in the same city. The top boosted words are relatively unaffected by changing the rating.

Figure 4.6: Distribution of a PCA projection of hotel embeddings from the TripAdvisor FactorCell model showing the grouping of the hotels by city.

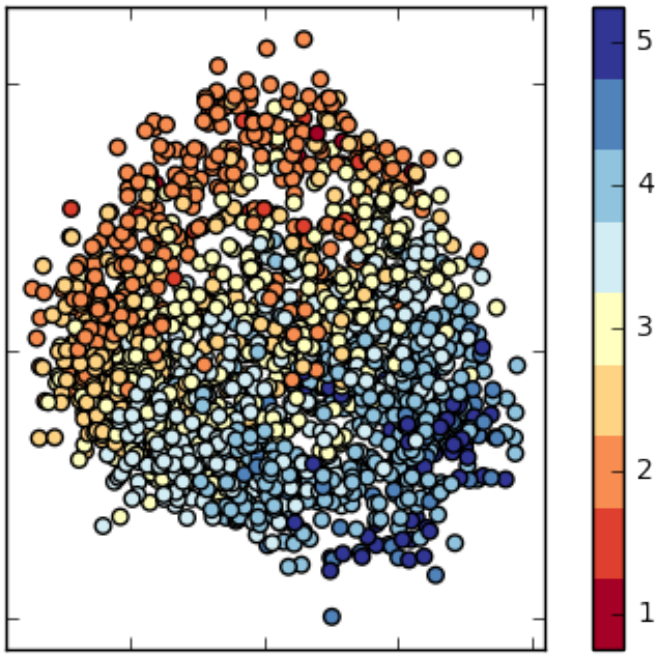

Figure 4.7: Distribution of a PCA projection of the hotel embeddings from the TripAdvisor FactorCell model showing the grouping of hotels by class.

(Recall that the hotel identifier and the user rating are the only two inputs used to create the context embedding.) This table combined with the other visualizations indicates that location effects tend to dominate in the output layer, which may explain why the two models adapting the recurrent network seem to have a bigger impact on classification performance.

Hotel	City	Class	Rating	Top Boosted Words
Amalfi	Chicago	4.0	5	amalfi, chicago, allegro, burnham, sable, michigan, acme, conrad, talbott, wrigley
BLVD Hotel Suites	Los Angeles	2.5	3	hollywood, kodak, highland, universal, reseda, griffith, grauman's, beverly, ventura
Four Points Sheraton	Seattle	3.0	1	seattle, pike, watertown, deca, needle, pikes, pike's monorail, uw, safeco

Table 4.5: The top boosted words in the Softmax bias layer for different context settings in a FactorCell model.

4.5 Comparison to Related Work

The studies that most directly relate to our work are neural models that correspond to special cases of the more general FactorCell model, including those that leverage what we call the SoftmaxBias model (Dieng et al., 2016; Tang et al., 2016; Yogatama et al., 2017; Ficler and Goldberg, 2017) and others that use the ConcatCell approach (Mikolov and Zweig, 2012; Wen et al., 2013; Chen et al., 2015; Ghosh et al., 2016).

The FactorCell model is distinguished by having an additive (factored) context-dependent transformation of the recurrent layer weight matrix. A related additive context-dependent transformation has been proposed for log-bilinear sequence models (Eisenstein et al., 2011a;

Hutchinson et al., 2015), but these are less powerful than the RNN. Factored tensors have been successfully used in other NLP applications such as dependency parsing (Lei et al., 2014). A somewhat different use of low-rank factorization has previously been used to reduce the parameter count in an LSTM LM (Kuchaiev and Ginsburg, 2017), finding that the reduced number of parameters leads to faster training.

Much of the work on context-adaptive neural language models has focused on incorporating document or topic information (Mikolov and Zweig, 2012; Ji et al., 2015; Ghosh et al., 2016; Dieng et al., 2016), where context is defined in terms of word or n-gram statistics. Our work differs from these studies in that the context is defined by a variety of sources, including discrete and/or continuous metadata, which is mapped to a context vector in end-to-end training. Context-sensitive language models for text generation tend to involve other forms of context similar to the objective of our work, including speaker characteristics (Luan et al., 2016; Li et al., 2016), dialog act (Wen et al., 2015), sentiment and other factors (Tang et al., 2016; Hu et al., 2017), and style (Ficler and Goldberg, 2017). Our work is distinctive in assessing performance over a broad variety of context variables.

4.6 Summary

In summary, this chapter has introduced a new model for adapting (or controlling) a language model depending on contextual metadata. The FactorCell model extends prior work with context-dependent RNNs by using the context vector to generate a low-rank, factored, additive transformation of the recurrent cell weight matrix. Experiments with six tasks show that the FactorCell model matches or exceeds performance of alternative methods in both perplexity and text classification accuracy. Findings hold for a variety of types of context, including high-dimensional contexts, and the adaptation of the recurrent layer is particularly important for character-level models. For many contexts, the benefit of the FactorCell model comes with essentially no additional computational cost at test time, since the transformations can be pre-computed. Analyses of a dataset with a high-dimensional sparse context vector show that the model learns context similarities to facilitate parameter sharing.

An adapted language model needs to memorize information about the unique language patterns of each context. One way of thinking about the difference between the ConcatCell and the FactorCell models is to ask where in the model is that information encoded. The context embedding c has too few bits to hold much information by itself. The same is true for the ConcatCell's $\mathbf{Q}$ matrix from Equation 3.1. The information must be held inside the shared weight matrix $\mathbf{W}$. This exposes a weakness of the ConcatCell model. If only one context is being used at a time why should the weights for all the contexts be active all the time? The FactorCell model has many extra parameters stored in the $\mathbf{Z}_L$ and $\mathbf{Z}_R$ tensors. It is able to offload context specific information to these locations and only use it when it is needed.

The models evaluated here were tuned to minimize perplexity, as is typical for language modeling. In analyses of performance with different hyperparameter settings, we find that perplexity is not always positively correlated with accuracy, but the criteria are more often correlated for approaches that adapt the recurrent layer. While not surprising, the results raise concerns about using perplexity as the sole evaluation metric for context-aware language models. More work is needed to understand the relative utility of these objectives for language model design, which we address in Chapter 6.

In real applications, it is possible for the meaning of the contexts to shift over time and the adapted model becomes out-of-date. To fix this, the model needs to be replenished by re-training with new data or it can be updated continuously using online learning. We address the latter of these two strategies in the next chapter.

Chapter 5

PERSONALIZED QUERY AUTO-COMPLETION

5.1 Background

This chapter demonstrates the benefits of the FactorCell model on the real-world task of query auto-completion (QAC).[1] QAC is a feature used by search engines that provides a list of suggested queries for the user as they are typing. For instance, if the user types the prefix "mete" then the system might suggest "meters" or "meteorite" as completions. This feature can save the user time and reduce cognitive load (Cai et al., 2016).

Most approaches to QAC are extensions of the Most Popular Completion (MPC) algorithm (Bar-Yossef and Kraus, 2011). MPC suggests completions based on the most popular queries in the training data that match the specified prefix. One way to improve MPC is to consider additional signals such as temporal information (Shokouhi and Radinsky, 2012; Whiting and Jose, 2014) or information gleaned from a users' past queries (Shokouhi, 2013). This chapter deals with the latter of those two signals, i.e. personalization. Personalization relies on the fact that query likelihoods are drastically different among different people depending on their needs and interests.

Recently, (Park and Chiba, 2017) suggested a significantly different approach to QAC. In their work, completions are generated from a character LSTM language model instead of by ranking completions retrieved from a database, as in the MPC algorithm. This approach is able to complete queries whose prefixes were not seen during training and has significant memory savings over having to store a large query database.

Building on this work, we consider the task of personalized QAC using an LSTM language model, combining the obvious advantages of personalization with the effectiveness of the

[1]The content of this chapter draws from our previously published work (Jaech and Ostendorf, 2018b).

language model in handling rare and previously unseen prefixes. The model must learn how to extract information from a user's past queries and use it to adapt the generative model for that person's future queries. User information is held in the form of a low-dimensional embedding that represents the person's interests and latent demographic factors. The experiments demonstrate that by allowing a greater fraction of the parameters to change in response to the user embeddings, the FactorCell has an advantage over the traditional approach to RNN language model adaptation that increases as more examples from the user are seen.

This task is different from the experiments described in Chapter 4 because new contexts (users) can emerge after the model has been trained and deployed. We introduce a mechanism to do online learning of the user embeddings, something that has not been addressed in prior work on language model adaptation. Table 5.1 provides an anecdotal example from the trained FactorCell model to demonstrate the intended behavior. The table shows the top five completions for the prefix "ba" in a cold start scenario and again after the user has completed five sports related queries. In the warm start scenario, the "baby names" and "babiesrus" completions no longer appear in the top five and have been replaced with "basketball" and "baseball". In the online learning scenario, the FactorCell model makes efficient use of the user query history to quickly improve the quality of the auto-completions.

While the standard implementation of MPC can not handle unseen prefixes, there are variants which do have that ability. Park and Chiba (2017) find that the neural LM outperforms MPC even when MPC has been augmented with the approach from Mitra and Craswell (2015) for handling rare prefixes. There has also been work on personalizing MPC (Shokouhi, 2013; Cai et al., 2014). We did not compare against these specific models because our goal was to show how personalization can improve the already-proven generative neural model approach. RNN's have also previously been used for the related task of next query suggestion (Sordoni et al., 2015).

Wang et al. (2018) show how spelling correction can be integrated into an RNN language model query auto-completion system and how the completions can be generated in real

	Cold Start	Warm Start
1	`bank of america`	`bank of america`
2	`barnes and noble`	`basketball`
3	`babiesrus`	`baseball`
4	`baby names`	`barnes and noble`
5	`bank one`	`baltimore`

Table 5.1: Top five completions for the prefix `ba` for a cold start model with no previous queries from that user and a warm model that has seen the queries `espn`, `sports news`, `nascar`, `yankees`, and `nba`.

time using a GPU. Our method of updating the model during evaluation resembles work on dynamic evaluation for language modeling (Krause et al., 2017), but differs in that only the user embeddings (latent demographic factors) are updated. If the rest of the model is allowed to update then either some user embedding can become stale or there will be a large memory cost to hold different versions of the model for each person. One possibility would be to allow the full model to update and also implement a policy to invalidate stale user embeddings but that is beyond the scope of this work.

5.2 Model

Adaptation depends on learning an embedding for each user, which we discuss in Section 5.2.1, and then using that embedding to adjust the weights of the recurrent layer, discussed in Section 5.2.2.

5.2.1 Learning User Embeddings

During training, we learn an embedding for each of the users. We think of these embeddings as holding latent demographic factors for each user. Users who have less than 15 queries in the training data (around half the users but less than 13% of the queries) are grouped

together as a single entity, $user_1$, leaving k users. The user embeddings matrix $\mathbf{U}_{k \times m}$, where m is the user embedding size, is learned via back-propagation as part of the end-to-end model. The embedding for an individual user is the ith row of $\mathbf{U}$ and is denoted by u_i. The user embedding plays the same role as the context embedding from Section 2.3.1 that elsewhere is denoted by c.

It is important to be able to apply the model to users that are not seen during training. This is done by online updating of the user embeddings during evaluation. When a new person, $user_{k+1}$ is seen, a new row is added to $\mathbf{U}$ and initialized to u_1. Each person's user embedding is updated via back-propagation every time they select a query. When doing online updating of the user embeddings, the rest of the model parameters (everything except $\mathbf{U}$) are frozen.

The learning rate for the online updating should be different than the one used for training the rest of the model. When training the full model, the goal is to converge to a global minimum. During online updating the user embeddings do not converge to a fixed point but continue to track the query history. The optimal learning rate must be found using validation data.

5.2.2 Recurrent Layer Adaptation

We consider three model architectures which differ only in the method for adapting the recurrent layer. First is the unadapted LM, analogous to the model from Park and Chiba (2017), which does no personalization. The second architecture is the ConcatCell, which concatenates a user embedding to the character embedding at every step of the input to the recurrent layer. For the third model, we test the FactorCell's ability to let the user embedding transform the weights of the recurrent layer. Unlike the previous chapter, we do not incorporate the ConcatCell adaptation of the recurrent layer bias when using the FactorCell model. We found that this was unnecessary in preliminary experiments.

When operating at the character-level, the unigram distribution is not particularly informative of the differences between users. This is unlike the word-level models, where the

unigram distribution can carry topic information. For this reason, we do not employ any of the techniques from Chapter 4 for adapting the softmax bias vector.

5.3 Data

The experiments make use of the AOL Query data collected over three months in 2006 (Pass et al., 2006). The first six of the ten files were used for training. This contains approximately 12 million queries from 173,000 users for an average of 70 queries per user (median 15). A set of 240,000 queries from those same users (2% of the data) was reserved for tuning and validation. From the remaining files, one million queries from 20,000 users are used to test the models on a disjoint set of users. The chronological ordering of the queries is ignored during training but respected during testing.

Our results are not directly comparable to Park and Chiba (2017) or Mitra and Craswell (2015) due to differences in the partitioning of the data and the method for selecting random prefixes. Prior work partitions the data by time instead of by user. Splitting by users is necessary in order to properly test personalization over longer time ranges.

5.4 Experiments

5.4.1 Experiment Details

The vocabulary consists of 79 characters including special start and stop tokens. Models were trained for six epochs. The Adam optimizer is used during training with a learning rate of 10^{-3} (Kingma and Ba, 2014). The language model is a single-layer character-level LSTM with coupled input and forget gates and layer normalization (Melis et al., 2018; Ba et al., 2016). We do experiments on two model configurations: small and large. The small models use an LSTM hidden state size of 300 and 20 dimensional user embeddings. The large models use a hidden state size of 600 and 40 dimensional user embeddings. Both sizes use 24 dimensional character embeddings. For the small sized models, we experimented with different values of the FactorCell rank hyperparameter between 30 and 50 dimensions

finding that bigger rank is better. The large sized models used a fixed value of 60 for the rank hyperparemeter. During training only and due to limited computational resources, queries are truncated to a length of 40 characters. The model[2] is implemented using Tensorflow.

When updating the user embeddings during evaluation, we found that it is easier to use an optimizer without momentum. We use Adadelta (Zeiler, 2012) and tune the online learning rate to give the best perplexity on a held-out set of 12,000 queries, having previously verified (See Figure 5.1) that perplexity is a good indicator of performance on the QAC task. The learning rate is the only parameter that needs to be tuned for online learning.

Prefixes are selected uniformly at random with the constraint that they contain at least two characters in the prefix and that there is at least one character in the completion. To generate completions using beam search, we use a beam width of 100 and a branching factor of 4. Results are reported using mean reciprocal rank (MRR), the standard method of evaluating QAC systems. It is the mean of the reciprocal rank of the true completion in the top ten proposed completions. The reciprocal rank is zero if the true completion is not in the top ten.

Neural models are compared against an MPC baseline. Following (Park and Chiba, 2017), we remove queries seen less than three times from the MPC training data to avoid excessive memory usage.

5.4.2 Results

The training objective, minimizing perplexity, differs from the evaluation metric, maximizing the mean reciprocal rank of the query completion. Fortunately, the mismatch between metrics is not large. As shown in Figure 5.1, perplexity correlates with MRR on the development data.

Table 5.2 compares the performance of the different models against the MPC baseline on a test set of one million queries from a user population that is disjoint with the training set.

---

[2]Code is available at http://github.com/ajaech/query_completion.

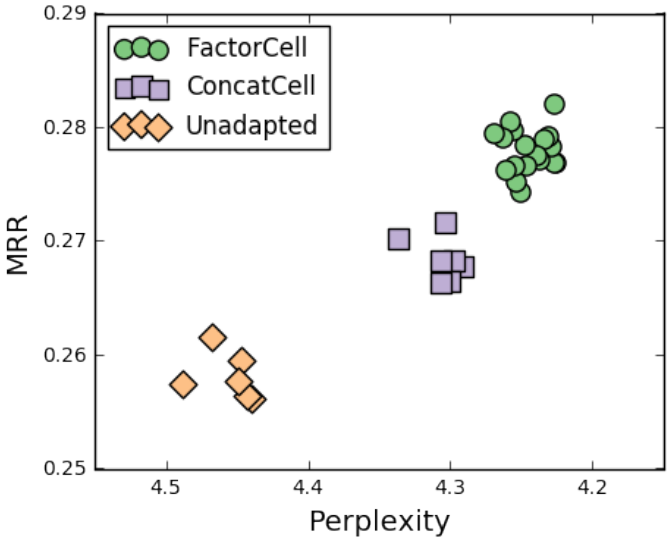

Figure 5.1: Perplexity versus MRR on the development data for different classes of models.

Results are presented separately for prefixes that are seen or unseen in the training data. If the prefix was not seen in the training data then the query is guaranteed to be relatively rare. Consistent with prior work, the neural models do better than the MPC baseline. The personalized models are better than the unadapted one, and the FactorCell model is the best overall in both the big and small sized experiments.

Figure 5.2 shows the relative improvement in MRR over an unpersonalized model versus the number of queries seen per user. Both the FactorCell and the ConcatCell show continued improvement as more queries from each user are seen, and the FactorCell outperforms the ConcatCell by an increasing margin over time. In the long run, we expect that the system will have seen many queries from most users. Therefore, the right side of Figure 5.2, where the FactorCell is up to 2% better than the ConcatCell, is more representative of the relative performance of the two systems. Since the data was collected over a limited time frame and half of all users have fifteen or fewer queries, the results in Table 5.2 do not reflect the full benefit of personalization.

Figure 5.3 shows the MRR for different prefix and query lengths. We find that longer

Size	Model	Seen	Unseen	All
	MPC	.292	.000	.203
	Unadapted	.292	.256	.267
(S)	ConcatCell	.296	.263	.273
	FactorCell	.300	.264	.275
	Unadapted	.324	.286	.297
(B)	ConcatCell	.330	.298	.308
	FactorCell	.335	.298	.309

Table 5.2: MRR reported for seen and unseen prefixes for small (S) and big (B) models.

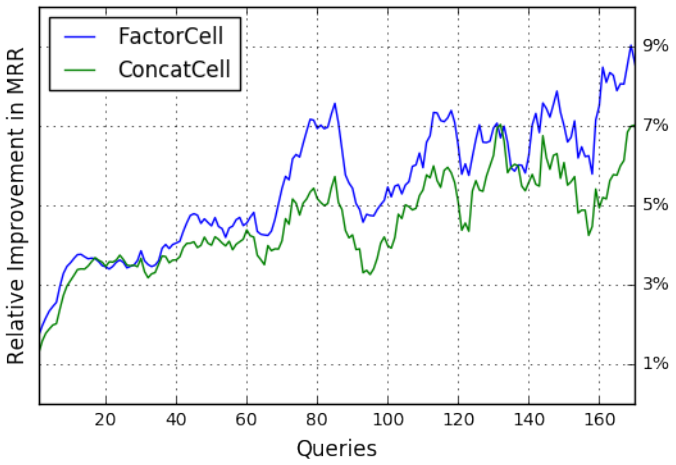

Figure 5.2: Relative improvement in MRR over the unpersonalized model versus queries seen using the large size models. Plot uses a moving average of width 9 to reduce noise.

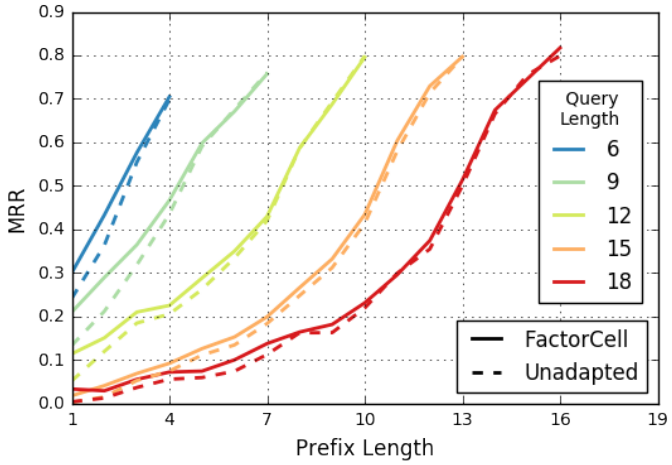

Figure 5.3: MRR by prefix and query lengths for the large FactorCell and unadapted models with the first 50 queries per user excluded.

prefixes help the model make longer completions and (as expected) shorter completions have higher MRR. Comparing the personalized model against the unpersonalized baseline, we see that the biggest gains are for short queries and prefixes of length one or two.

We found that one reason why the FactorCell outperforms the ConcatCell is that it is able to pick up sooner on the repetitive search behaviors that some users have. This commonly happens for navigational queries like when someone searches for the name of their favorite website once or more per day. At the extreme tail there are users who search for nothing but free online poker. Both models do well on these highly predictable users but the FactorCell is generally a bit quicker.

We conducted an analysis to better understand what information is represented in the user embeddings and what makes the FactorCell different from the ConcatCell. From a cold start user embedding we ran two queries and allowed the model to update the user embedding. Then, we ranked the most frequent 1,500 queries based on the ratio of their likelihood from before and after updating the user embeddings.

Tables 5.3, 5.4, and 5.5 show the queries with the highest relative likelihood of the adapted vs. unadapted models after two related search queries: “high school softball” and

	FactorCell	ConcatCell
1	high school musical	horoscope
2	chris brown	high school musical
3	funnyjunk.com	homes for sale
4	funbrain.com	modular homes
5	chat room	hair styles

Table 5.3: The five queries that have the greatest adapted vs. unadapted likelihood ratio after searching for "high school softball" and "math homework help".

"math homework help" for Table 5.3, "Prada handbags" and "Versace eyewear" for Table 5.4, and "discount flights" and "Yellowstone vacation packages" for Table 5.5. In all cases, the FactorCell model examples are more semantically coherent than the ConcatCell examples. In the first case, the FactorCell model identifies queries that a high school student might make, including entertainment sources and a celebrity entertainer popular with that demographic. In the second case, the FactorCell model chooses retailers that carry woman's apparel and those that sell home goods. While these companies' brands are not as luxurious as Prada or Versace, most of the top luxury brand names do not appear in the top 1,500 queries and our model may not be capable of being that specific. There is no obvious semantic connection between the highest likelihood ratio phrases for the ConcatCell; it seems to be focusing more on orthography than semantics (e.g. "home" in the first example). In the last case, the FactorCell suggests a map and four different airlines and the ConcatCell suggests a flight tracker (unlikely to be useful to someone looking to buy a ticket), a bank, and three reformulations of the original "discount flights" query. Not shown are the queries which experienced the greatest decrease in likelihood. For the "high school" case, these included searches for travel agencies and airline tickets—websites not targeted towards the high school age demographic.

	FactorCell	ConcatCell
1	neiman marcus	craigslist nyc
2	pottery barn	myspace layouts
3	jc penney	verizon wireless
4	verizon wireless	jensen ackles
5	bed bath and beyond	webster dictionary

Table 5.4: The five queries that have the greatest adapted vs. unadapted likelihood ratio after searching for "prada handbags" and "versace eyewear".

	FactorCell	ConcatCell
1	yahoo maps	flight tracker
2	delta airlines	suntrust bank
3	alaska airlines	cheap tickets
4	us airways	airline tickets
5	southwest airlines	cheap flights

Table 5.5: The five queries that have the greatest adapted vs. unadapted likelihood ratio after searching for "discount flights" and "yellowstone vacation packages".

5.5 Summary

Our experiments show that the LSTM model can be improved using personalization. The method of adapting the recurrent layer clearly matters and we obtained an advantage by using the FactorCell model. The reason the FactorCell does better is in part attributable to having two to three times as many parameters in the recurrent layer as either the ConcatCell or the unadapted models. By design, the adapted weight matrix $\mathbf{W}'$ only needs to be computed at most once per query and is reused many thousands of times during beam search. As a result, for a given number of floating point operations, the FactorCell model outperforms the ConcatCell model for LSTM adaptation.

The cost for updating the user embeddings is similar to the cost of the forward pass and depends on the size of the user embedding, hidden state size, FactorCell rank, and query length. In addition, we note that updates can be less frequent to reduce costs and in most cases there will be time between queries for updates.

We showed that language model personalization can be effective even on users who are not seen during training. The benefits of personalization are immediate and increase over time as the system continues to leverage the incoming data to build better user representations. The approach can easily be extended to include time as an additional conditioning factor. We leave the question of whether the results can be improved by combining the language model with MPC for future work.

Chapter 6

CONTEXT-SPECIFIC TEXT GENERATION

Context-specific generation is when a generative model is designed such that stylistic, topical, or other properties of the text can be specified when sampling from the model. One of the weaknesses of RNN language models is that it is difficult to control the content or the style of the text that they produce. It is natural to expect that if a language model is conditioned on a particular context then it will produce text that is appropriate for that context. Conditioning, or adapting to, context is one way of controlling generation (Ficler and Goldberg, 2017) and that is the application that we explore in this chapter.

We conduct experiments and analysis using the Yelp restaurant review and the TripAdvisor hotel review datasets from Section 4.2. These experiments reuse the models that were trained for the experiments in Section 4.3, including models adapted using the SoftmaxBias, ConcatCell, and FactorCell strategies. The focus of the experiments in this chapter is to highlight differences in context-specificity between these models by generating reviews from the models and checking if the text is appropriate for the given context. The generated reviews are judged using a classifier that has been trained for that purpose. The automatic judgments are supplemented by human ratings to confirm the reliability of the classifier.

There are ways of producing context-specific text without generating it from a language model such as by retrieving examples with matching context from the training data or combining retrieval with a neural edit model (Li et al., 2018). Our context-specificity metric will highly reward these types of models or any model that produces easy to classify content even if that content is unnatural or an outlier. Retrieval-based methods may work well in certain applications but others require the use of a language model. This chapter focuses specifically on language models. We emphasize that a context-specificity metric by itself

is not enough to judge the quality of a language model. Low perplexity is important for good generation. However, a language model that has low perplexity and is also capable of context-specific generation will have more applications than a non-context-specific model.

6.1 Text Generation

To generate text from an RNN, we use the model to predict the probability of observing each possible word as the next token in the sequence starting by inputting a special start of sentence token for w_1. The next token probabilities are given by the y_t vector from Equation 2.3. One option is to take the next word as the highest probability entry in y_t, known as greedy decoding. Making greedy decisions can lead us down a path that is globally suboptimal and may not result in finding the sequence with the highest overall likelihood. Instead, using an algorithm known as beam search, several possibilities are considered for the next word and are placed in a queue that holds the most likely word sequences found so far. Beam search can find many high probability sentences but oftentimes it suffers from a lack of diversity. The highest probability sequences will all have considerable overlap with each other. This is not desirable in text generation because we generally want multiple texts with non-trivial differences to pick from and also because excessive repetition is tedious for human readers. A simple tweak that leads to more diverse generation is to stochastically expand the beam by sampling l items from the multinomial distribution given by y_t instead of deterministically picking the top l words. When we use this technique we refer to it as a stochastic beam search.

We can trade-off between the stochastic and deterministic versions of beam search by using a temperature parameter T in the softmax function as follows:

$$\text{softmax}(x_i) = \frac{\exp(-\frac{x_i}{T})}{\sum_j \exp(-\frac{x_j}{T})}. \tag{6.1}$$

Lowering the temperature T increases the concentration of probability mass at the head of the distribution. If the temperature is too high then the generated text can be nonsensical. If it is too low then the generation lacks diversity. In the limit when T goes to zero, it

becomes equivalent to greedy decoding. So, finding the right balance is important.

Stochastic beam search deals with repetition between sampled texts. RNN language models also tend to use the same phrase repetitively within a single sentence or document, e.g. a restaurant review that says "The food was good and the service was great and the food was good and...". To deal with this we use a heuristic that does not consider any beam expansion that would create a repeated trigram. Several sophisticated techniques and heuristics exist (such as adding random penalties to subsets of the vocabulary (Juuti et al., 2018) or having special models trained to enforce relevance and avoid contradiction and repetition (Holtzman et al., 2018)) but banning repeated trigrams is sufficient for our purpose of making a comparison between adaptation methods.

6.2 Illustrative Examples

To start, we provide anecdotal examples in Tables 6.1, 6.2, and 6.3 illustrating the behavior of each adaptation technique when generating Yelp restaurant reviews given a star rating. In each case, a sentence template was selected and beam search decoding was used to find the highest probability word sequence that fits the gap in the template. This was repeated once for each of the possible star ratings from one to five. The sentence completions from the FactorCell model are better matched to the specified star ratings. Table 6.2 is probably the best example for the intensity of the selected adjectives to match with the corresponding star ratings. On the other hand, when using the ConcatCell model there is not a clear distinction between ratings. The ConcatCell picks the same completion for multiple star ratings ("`loved it`" in Table 6.1 and "`great!`" in Table 6.2) and in Table 6.3 it gets the wrong polarity for the 5 star rating completion.

Context	FactorCell	ConcatCell	SoftmaxBias
5 stars	it was amazing	loved it !	it was amazing !
4 stars	loved it !	loved it !	it was delicious
3 stars	it was great !	loved it !	had a blast
2 stars	it was horrible !	had a blast	was disappointed
1 star	it was horrible !	it was horrible !	was disappointed

Table 6.1: Top completions for the sentence "My boyfriend and I ate here and _____!" after conditioning on each star rating.

Context	FactorCell	ConcatCell	SoftmaxBias
5 stars	amazing !	great !	delicious
4 stars	great !	great !	delicious
3 stars	good !	great !	delicious
2 stars	just meh	mediocre	mediocre
1 star	awful	mediocre	mediocre

Table 6.2: Top completions for the sentence "This was my first time coming here and the food was _____" after conditioning on each star rating.

Context	FactorCell	ConcatCell	SoftmaxBias
5 stars	be back	never go here	never go here
4 stars	be back	go	never go here
3 stars	go back	go	never go here
2 stars	never go here	never	never go here
1 star	never go here	never	never go

Table 6.3: Top completions for the sentence "I will _____ again" after conditioning on each star rating.

6.3 Experiments and Analysis

6.3.1 Yelp Restaurant Reviews

To quantify the differences between models, we sampled 2,000 Yelp reviews from 36 different models. The models differ in their adaptation method but also in their LSTM size and other hyperparameters including random word embedding sizes, epochs, context embedding size and FactorCell rank. Having variation in the hyperparameter settings allows us to analyze the relationship between perplexity and context-specificity, among other things. The generation used stochastic beam search with a temperature of 1.0, a beam width of 8 and a total beam size of 120. Only the first 70 words of the review were generated and the beam was constrained to not allow repeated trigrams as a heuristic to avoid excessive repetition. A discriminative classifier was trained to predict star rating on the same data that was used to train the language models. We used the fastText model because it is near state-of-the-art and it can be efficiently trained and evaluated (Grave et al., 2017b). This classifier was used to judge the generated reviews to see if they conform to their given ratings. This strategy was used by Hu et al. (2017) to evaluate "controllable" text generation. It is also similar to the text generation evaluation used by Ficler and Goldberg (2017) except that for some attributes they used hand-crafted decision rules instead of a statistical classifier and for other attributes deemed too difficult to classify they used human evaluation.

There are multiple ways to report the judgment of the discriminative classifier. The most straightforward is accuracy, which is the percentage of times where the most probable rating according to the classifier matched the context given to the language model. In addition to this metric, we report results in terms of mean absolute deviation (the average absolute difference between the desired rating and the predicted rating). Achieving perfect accuracy is not realistic for this task because there is some natural ambiguity between adjacent star ratings. The fastText classifier has an accuracy of 63% and a median absolute deviation (MAD) of 0.44 stars on the Yelp test data.

Figure 6.1 plots the context classification metric from Chapter 4 (accuracy of classifying

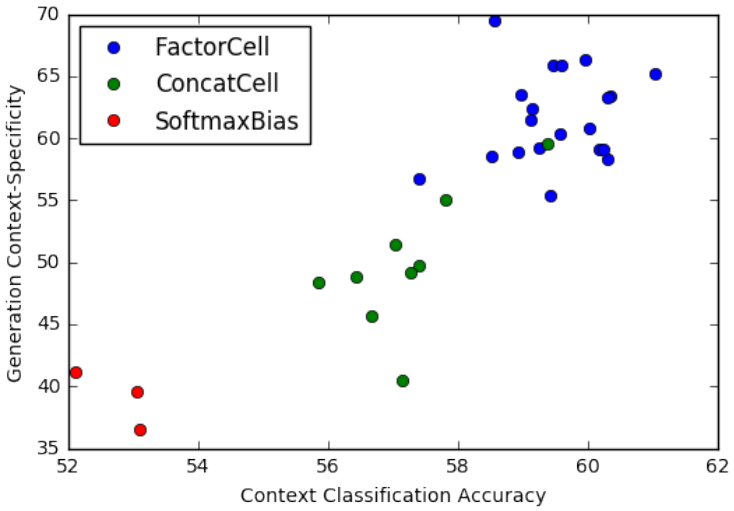

Figure 6.1: Context classification accuracy versus generation context-specificity for each type of adaptation on the Yelp data.

actual reviews using the language model as a classifier) against the generation context-specificity metric (accuracy of classifying automatically generated text with a classifier trained on actual reviews). Each point in this plot is an independent model trained with random hyperparameters. The context classification accuracy is highly predictive of the generation context-specificity. Thus, it is not surprising that the FactorCell models are the most controllable followed by the ConcatCell models and then the SoftmaxBias models.

As shown in Figure 6.2, there is a strong correlation between FactorCell rank and context-specificity ($r = 0.8$) but no relationship ($r = 0.00$) with perplexity. Having low perplexity is obviously important for high-quality generation but low perplexity by itself is not an indicator that the generated text will match the specified conditions. We included some additional models here that were no used in the experiments in Chapter 4 and were trained with bigger (up to double the size) LSTM hidden states. This is to address the question of whether the gap between models is reduced simply by training a bigger model. The bigger LSTM states do help with perplexity, but we find little to no impact on context-specificity.

Table 6.4 lists the perplexity, automatically assessed generation accuracy, and the mean

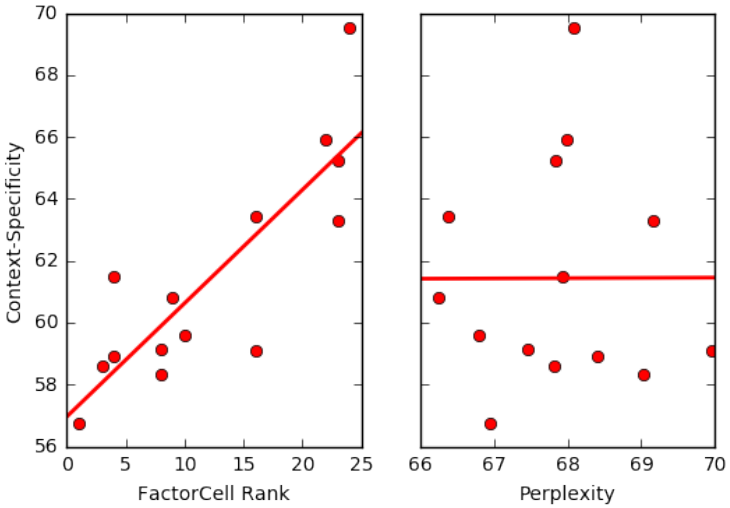

Figure 6.2: Plot of FactorCell rank and perplexity against generation context-specificity accuracy for 14 FactorCell models on the Yelp restaurant data.

Model	**ACC**	**MAD**	**PPL**
Unadapted	21.3	1.63	67.9
SoftmaxBias	41.1	.88	68.1
ConcatCell	59.6	.44	66.8
FactorCell	69.5	.33	68.1

Table 6.4: Automatically judged generation accuracy, mean absolute deviation (MAD), and perplexity for the three methods of adaptation compared to the unadapted baseline using the models learned from the Yelp data.

absolute deviation for each of the four classes of models. Using human raters on a subset of the generated reviews, we confirm that the automatic judgments are correlated with human judgments with $r = 0.823$. Nine raters judged a set of 145 reviews for this analysis. The raters were able to guess the intended star rating 64% of the time for the FactorCell compared to only 59% of the time for the ConcatCell.

6.3.2 TripAdvisor Hotel Reviews

We conduct a similar analysis using the TripAdvisor hotel review data and models from Chapter 4 and check that the hotel class property matches with the generated text. Hotel class is a rating that is reflective of the quality of the service and amenities offered by the hotel and is measured in half integer increments. Only the identity of the hotels and the star rating (sentiment) of each review is provided during training and any information the model learns about class must be derived from the text of the review. Hotel reviews are sampled using a temperature of 1.0, a beam width of 4 and a beam size of 50. Only the first 72 to 100 words of each review were generated and 1,000 samples were collected from 54 independent models. When generating the hotel reviews we pick a random hotel identifier for each one and fix the sentiment at five stars. Like before, these models use a range of hyperparameters in order to study the relationship between the capacity of the model and its performance.

The fastText classifier predicts hotel class with an accuracy of 53% and a mean absolute deviation of 0.36 on the TripAdvisor test data. Using this classifiers to judge the generated reviews, we find that the FactorCell is the the most specific for the hotel class. The complete metrics are in Table 6.5.

Besides their lack of context-specificity, the unadapted models also suffer from lack of diversity. The unadapted models tend to default to generating reviews that match the largest region in the data, New York City. Up to 35% of the generated reviews from the unadapted models mention the Empire State Building (compared to 2% to 3% for the adapted models) even though only 30% of the training data are from hotels in New York City and only 1.6% of that 30% even mention the Empire State Building. Conditioning on context

Model	ACC	MAD	PPL
Unadapted	21.3	.77	51.9
SoftmaxBias	30.1	.61	48.7
ConcatCell	30.4	.57	48.8
FactorCell	32.2	.53	48.9

Table 6.5: Automatically judged generation accuracy, mean absolute deviation (MAD), and perplexity for the three methods of adaptation compared to the unadapted baseline using the models learned from the TripAdvisor data.

information makes it easier to guide the generation away from the mode and towards more diverse samples.

As shown in Figure 6.3, increasing the FactorCell rank helps with context-specificity of the hotel class just as it did with controlling the star rating of the Yelp reviews in Figure 6.2. However, instead of a lack of correlation with perplexity, we see that models with better perplexity also do better at context-specificity. Same as for the Hotel reviews, the accuracy of the models when used as generative text classifiers is predictive of their performance for context-specific generation. We show this relationship in Figure 6.4.

We used the TripAdvisor models to test if the difference in context-specificity between the models was due to the quality of the representation learned in the hotel embeddings. We trained a gradient boosted decision tree to predict the hotel class using the hotel embedding as features. As we saw in Figure 4.7, the hotel embeddings naturally group together by class even though they hotel class labels were not available during training. Using five-fold cross validation on the 2,254 embeddings, we found little to no difference between models with both the ConcatCell and FactorCell correctly predicting the class in 67.1% of cases versus 65.3% for the SoftmaxBias model.

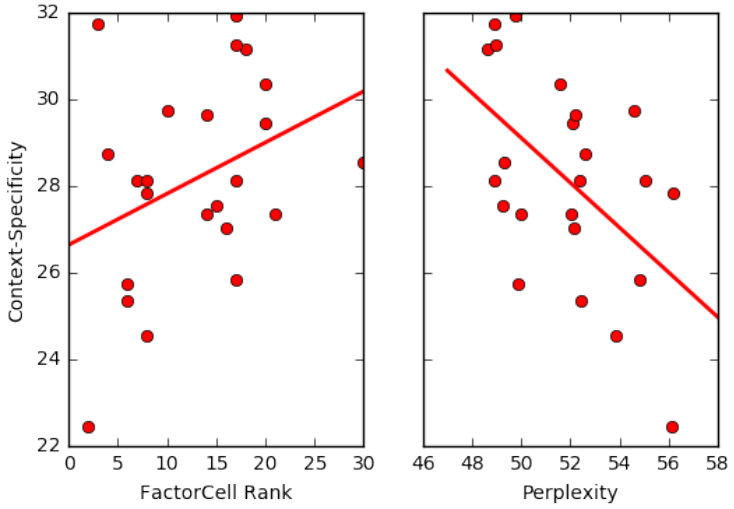

Figure 6.3: Context-specificity of hotel class versus FactorCell rank and perplexity in generated reviews using the models learned on the TripAdvisor data.

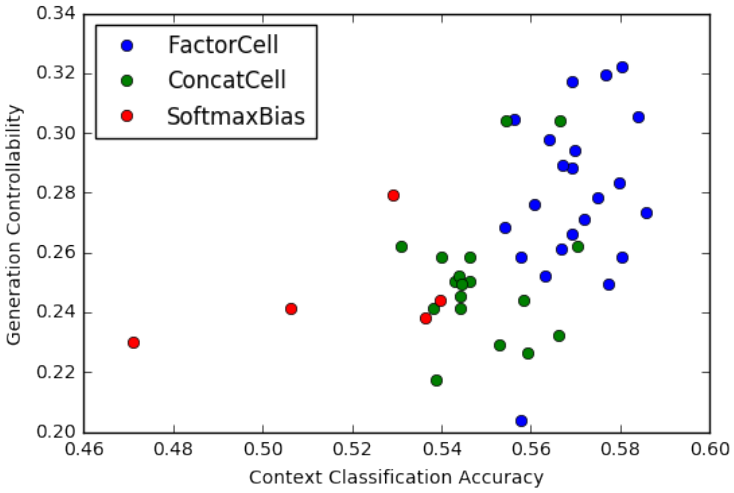

Figure 6.4: Context classification accuracy versus generation context-specificity for each type of adaptation on the TripAdvisor data.

6.4 Summary

Recall that the ConcatCell model only adapts the recurrent layer bias vector. Assuming 200-dimensional word embeddings and a 400 dimensional LSTM, only 1,200 or 0.17% of the recurrent layer parameters change depending on the context. In contrast, a FactorCell model with the same embedding size and recurrent layer dimension and a rank of 25 has 6.25% of its recurrent layer parameters changing depending on context. That's more than 37 times as many as the ConcatCell. When so few of the parameters are adapted, the result is a blurring of the distinctions between different contexts. We saw this qualitatively in the restaurant review sentence completions and quantitatively when measuring the context-specificity of the restaurant review star rating and the hotel review class.

In Chapter 4, we used a context classification metric whereby the language model was used as a generative text classifier on held-out data in order to test its ability to understand the relationship between context and language. The experiments in this chapter show that the context classification metric is a useful predictor of text generation context-specificity. This means that we can assess the relative context-specificity of models independently of the generation hyperparameters (beam size, temperature, etc.) and without the need to construct a classifier or use human ratings to do judgments. This is useful because there are cases where a high quality classifier may not be readily available. For human ratings, not only are they expensive and time-consuming to collect, but there are also certain contexts where humans may be poor judges without special experience or training.

Our method of evaluating context-specificity of the generated text has some limitations. Because the fastText classifier is trained on the same data as the language model, it has the potential to reward the language model for memorizing portions of the training data instead of generating novel sentences. Additionally, as we mentioned in the introduction to this chapter, context-specificity alone is not enough to evaluate a language model because it does not measure the ability of the model to generate well-formed text that is free from excessive repetitions and self-contradictions. We know from Chapter 4 that the FactorCell

model does have a perplexity that is comparable or better than baseline methods. However, a more comprehensive assessment is needed to take into account all of the desired attributes.

Chapter 7
CONCLUSIONS AND FUTURE DIRECTIONS

The fact that language changes so dramatically from one situation to the next is both a challenge and an opportunity. It is a challenge because changes in context that may seem trivial to us as humans are enough to break a language model. It is an opportunity because dramatic changes means that large benefits can be accrued through adaptation. As an illustration, the 2017 Amazon Alexa prize encouraged people to speak to their device using an open-ended conversational style. This was a change from the task-oriented style that people normally used to speak with their home assistants. Initially, a generic language model was used for the conversational interactions. When a language model targeting conversational speech was introduced the recognition word error rate dropped by one third (Ram et al., 2018). Adjusting the language model was crucial to enabling people to have long conversations with Alexa.

Adaptation is vital to the success of language modeling in real applications and is the reason why this work can have high impact. In this chapter we summarize the contributions of the thesis and suggest possible directions for future work.

7.1 Summary of Contributions

Our main contribution is the introduction of the FactorCell model, a new way of thinking about how to adapt the recurrent layer. The FactorCell model moves beyond the paradigm set by Mikolov and Zweig (2012) and used many times since then. The benefits of the FactorCell model include:

- A re-conception of the mechanism for recurrent layer adaptation (using context to transform the model rather than as an additional input);

- Permitting context to affect a greater change in the model parameters (and therefore its predictions) than what is possible using prior methods;

- Including a rank hyperparameter that smoothly trades off between the case where most information is shared between contexts and the opposite situation where the contexts are mostly modeled independently;

- Maintaining the property of the ConcatCell approach that the adapted weights can be precomputed and cached resulting in negligible computational overhead compared to an unadapted baseline; and

- Off-loading of context specific information from the main recurrent weight matrix to special adaptation tensors, which helps prevent the blurring of the boundary between related contexts.

We presented experimental results on nine distinct datasets. Three of them used character-level vocabularies and the other six operated at the word level. One dataset comprised transcripts of spoken language; some involved informal written language; and some were more formal. The biggest dataset had over 100 million words of text for training and the smallest was just 5 million characters. In every situation where it was tested, the FactorCell model gave the best perplexity. However the benefits are greater for some contexts. The diversity of data helped to increase the robustness of our conclusions and led us to make some guidelines about when different aspects of adaptation become more or less important. Specifically, adapting the softmax bias is most effective for topic-based contexts and potentially less effective when the recurrent layer dimensionality is large. Also, the contexts that are most amenable to adaptation are specific and are not easily predicted from a short window of text.

An idea that we encountered more than once is that there are differences between models that are not captured by perplexity. In Chapters 3 and 4, we saw that a more flexible adaptation of the recurrent layer helped the models better distinguish contexts in classification

experiments and Chapter 6 showed how models that are better at distinguishing contexts can be used for context-specific generation.

There were two places where we saw clear qualitative differences between the FactorCell and the ConcatCell. In Chapter 5, the FactorCell discovered semantic associations between search queries whereas the ConcatCell failed to find such associations and sometimes seemed to focus more on orthography. In Chapter 6, the ConcatCell blurred the boundary between star rating levels (associated with sentiment) and the FactorCell was able to make cleaner distinctions. Taken together, these indicate the FactorCell provides a greater benefit than the improvement in perplexity would suggest. This is likely because the gains are on less frequently used words, and perplexity is dominated by frequent words.

We showed how online learning lets the adapted model take advantage of contexts that emerge after training. The method of online adaptation that we introduced in Chapter 5 is generic; it will work for multiple adaptation strategies, including the ConcatCell. However, the FactorCell makes better use of the data stream to continue to increase the quality of its predictions over time.

We made some observations about what factors make a dataset more or less amenable to adaptation. These include, from Chapter 3, that context variables that are easily predictable from the text alone are unlikely to be helpful and, from Chapter 4, that for topic related contexts adapting the softmax bias may be more successful. We also found that having a larger sized recurrent layer can lessen the impact of adapting the softmax bias.

7.2 Future Directions

There are several promising future directions that build on the work in this thesis.

In all six tasks that are explored in Chapter 4, all context factors are available for all training and testing samples. In some scenarios, it may be possible for some context factors to be missing. A simple solution for handling this is to use the expected value for the missing variable(s), since this is equivalent to using a weighted combination of the adaptation matrices for the different possible values of the missing variables.

The experiment scenarios all used metadata to specify context, since this type of context can be more sensitive to data sparsity and has been less studied. In contrast, in many prior studies of language model adaptation, context is specified in terms of text samples, such as prior user queries, prior sentences in a dialog, other documents related in terms of topic or style, etc. The FactorCell framework introduced here is also applicable to this type of context, but the best encoding of the text into an embedding (e.g. using bag of words, sequence models, etc.) is likely to vary with the application and remains an area of study for future work.

The generation experiments that we conducted were generating text from scratch with no objective other than to obey the constraints imposed by conditioning on the context. In applications, such as machine translation, the generation is conditioned on a source sequence in an encoder-decoder model (also called sequence-to-sequence, or seq2seq) (Cho et al., 2014). The FactorCell model is applicable in these situations as well and one future direction is to test how it performs in that setting. Prior work has already shown that a simple language model that incorporates contextual data can provide gains in machine translation (Drexler et al., 2014).

The data we used consisted of one or two context variables such as latitude and longitude or a hotel identifier and star rating. In these cases, it was reasonable to expect the context embedding to summarize the relevant information from both variables. If there were many more context variables then that assumption may no longer hold. More work is needed to find an appropriate dataset that has multiple context variables and to investigate appropriate methods of providing the context information to the FactorCell model.

In Chapter 3, we showed that using hashing to adapt the output layer bias benefits perplexity for high dimensional contexts. This indicates that the low-rank adaptation of the bias does not fully model the context-dependent language. We suggested that the model could benefit from a sparse correction to the low-rank adaptation and that could be accomplished by including an L1 penalty term to the adaptation parameters. This idea is partially explored in Appendix A, but more work is needed to investigate the benefits of sparsity in

language model adaptation.

Our focus was on language modeling, but RNNs are widely used in many other natural language processing tasks and in other domains that have little or nothing to do with language such as acoustic modeling (Graves et al., 2006), time-series analysis, music generation (Goel et al., 2014), and more. For example, some work on other natural language processing tasks, such as spoken language understanding, have already made use of the ConcatCell style adaptation (Mesnil et al., 2015). Personalization would be of high utility in these applications.

This thesis looked at query completion, but there are often text prediction applications that this work is relevant to, including next word prediction for mobile text input and augmentative communication for people with disabilities. For augmentative communications, the method would also apply to icon-based communication, which relies on a language model with icons instead of words or characters (Dudy and Bedrick, 2018).

It is difficult to predict what future state-of-the-art language models will look like. Recurrent neural networks have been reliable winners for the past few years. There is active work on alternate architectures that remedy some of the perceived weaknesses (mostly lack of easy parallelism) of the RNN, such as the Quasi-Recurrent Neural Network (Bradbury et al., 2017) and the Transformer Network (Vaswani et al., 2017). It seems likely that the adaptation techniques from this thesis will apply to those architectures as well, but it remains to be tested.

BIBLIOGRAPHY

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Yoshua Bengio, et al. 2016. End-to-end attention-based large vocabulary speech recognition. In *Proc. ICASSP*, pages 4945–4949.

Lalit Bahl, James Baker, Paul Cohen, Fred Jelinek, Burn Lewis, and R Mercer. 1978. Recognition of continuously read natural corpus. In *Proc. ICASSP*, volume 3, pages 422–424.

Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proc. WWW*, pages 107–116.

Jerome R Bellegarda. 2000. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 88(8):1279–1296.

Jerome R Bellegarda. 2004. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003a. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Yoshua Bengio, Jean-Sébastien Senécal, et al. 2003b. Quick training of probabilistic neural nets by importance sampling. In *Proc. AISTATS*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21. Association for Computational Linguistics.

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

Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proc. EMNLP*.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke. 2003. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proc. HLT-NAACL*, pages 7–9.

Fei Cai, Maarten De Rijke, et al. 2016. A survey of query auto-completion in information retrieval. *Foundations and Trends in Information Retrieval*, 10(4):273–363.

Fei Cai, Shangsong Liang, and Maarten De Rijke. 2014. Time-sensitive personalized query auto-completion. In *Proc. CIKM*, pages 1599–1608.

William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. 2015. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*.

Ciprian Chelba and Noam Shazeer. 2015. Sparse non-negative matrix language modeling

for geo-annotated query session data. In *Proc. Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 8–14.

Ciprian Chelba, Xuedong Zhang, and Keith B Hall. 2015. Geo-location for voice search language modeling. In *Proc. Interspeech*, pages 1438–1442.

Wenlin Chen, David Grangier, and Michael Auli. 2016. Strategies for training large vocabulary neural language models. In *Proc. ACL*.

Xie Chen, Tian Tan, Xunying Liu, Pierre Lanchantin, Moquan Wan, Mark JF Gales, and Philip C Woodland. 2015. Recurrent neural network language model adaptation for multi-genre broadcast speech recognition. In *Proc. InterSpeech*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. In *Proc. ACL*.

Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Võesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. 2007. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing (TSLP)*, 5(1):3.

Salil Deena, Madina Hasan, Mortaza Doulaty, Oscar Saz, and Thomas Hain. 2016. Combining feature and model-based adaptation of RNNLMs for multi-genre broadcast speech recognition. In *Proc. Interspeech*, pages 2343–2347.

Renato DeMori and Marcello Federico. 1999. Language model adaptation. In *Computational models of speech pattern processing*, pages 280–303.

Adji B Dieng, Chong Wang, Jianfeng Gao, and John Paisley. 2016. TopicRNN: a recurrent neural network with long-range semantic dependency. *arXiv preprint arXiv:1611.01702*.

Jennifer Drexler, Pushpendre Rastogi, Jacqueline Aguilar, Benjamin Van Durme, and Matt Post. 2014. A Wikipedia-based corpus for contextualized machine translation. In *Proc. LREC*, pages 3593–3596.

Shiran Dudy and Steven Bedrick. 2018. Compositional language modeling for icon-based augmentative and alternative communication. *Northwest NLP Regional Workshop*.

J. Eisenstein, A. Ahmed, and E. Xing. 2011a. Sparse additive generative models of text. In *Proc. ICML*.

Jacob Eisenstein, Amr Ahmed, and Eric P. Xing. 2011b. Sparse additive generative models of text. In *Proc. ICML*.

Jacob Eisenstein, Brendan O'Connor, Noah A Smith, and Eric P Xing. 2010. A latent variable model for geographic lexical variation. In *Proc. EMNLP*, pages 1277–1287.

Jessica Ficler and Yoav Goldberg. 2017. Controlling linguistic style aspects in neural language generation. In *Proc. EMNLP Workshop on Stylistic Variation*.

Siva Reddy Gangireddy, Pawel Swietojanski, Peter Bell, and Steve Renals. 2016. Unsupervised adaptation of recurrent neural network language models. In *Proc. Interspeech*, pages 2333–2337.

Roberto Gemello, Franco Mana, Stefano Scanzio, Pietro Laface, and Renato De Mori. 2007. Linear hidden transformations for adaptation of hybrid ANN/HMM models. *Speech Communication*, 49(10-11):827–835.

Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck. 2016. Contextual LSTM models for large scale NLP tasks. *arXiv preprint arXiv:1602.06291*.

James R Glass, Timothy J Hazen, D Scott Cyphers, Igor Malioutov, David Huynh, and Regina Barzilay. 2007. Recent progress in the MIT spoken lecture processing project. In *Proc. Interspeech*, pages 2553–2556.

Kratarth Goel, Raunaq Vohra, and JK Sahoo. 2014. Polyphonic music generation by modeling temporal dependencies using a RNN-DBN. In *Proc. International Conference on Artificial Neural Networks*, pages 217–224.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017a. Improving neural language models with a continuous cache. In *Proc. ICLR*, volume abs/1612.04426.

Edouard Grave, Tomáš Mikolov, Armand Joulin, and Piotr Bojanowski. 2017b. Bag of tricks for efficient text classification. In *EACL*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, pages 369–376. ACM.

Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *Proc. ICML*, volume 14, pages 1764–1772.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28:2222–2232.

David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *Proc. ICLR*.

Yoni Halpern, Keith Hall, Vlad Schogol, Michael Riley, Brian Roark, Gleb Skobeltsyn, and

Martin Baeuml. 2016. Contextual prediction models for speech recognition. In *Proc. Interspeech*.

Bo Han, Paul Cook, and Timothy Baldwin. 2014. Text-based twitter user geolocation prediction. *J. Artif. Intell. Res.*, 49:451–500.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Cong Duy Vu Hoang, Trevor Cohn, and Gholamreza Haffari. 2016. Incorporating side information into recurrent neural network language models. In *Proc. HLT-NAACL*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, and Yejin Choi. 2018. https://openreview.net/forum?id=r1lfpfZAb Learning to write by learning the objective.

Bo-June Paul Hsu and James Glass. 2008. N-gram weighting: reducing training data mismatch in cross-domain language model estimation. In *Proc. EMNLP*, pages 829–838.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Toward controlled generation of text. In *Proc. ICML*, pages 1587–1596.

B. Hutchinson, M. Ostendorf, and M. Fazel. 2015. A sparse plus low-rank exponential language model for limited resource scenarios. *IEEE Trans. Audio, Speech and Language Processing*, 23(3):494–504.

Brian Hutchinson, Mari Ostendorf, and Maryam Fazel. 2013. Exceptions in language as learned by the multi-factor sparse plus low-rank language model. In *Proc. ICASSP*, pages 8580–8584.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*.

Kazuki Irie, Shankar Kumar, Michael Nirschl, and Hank Liao. 2018. RADMM: Recurrent adaptive mixture model with applications to domain robust language modeling.

Aaron Jaech, George Mulcaire, Shobhit Hathi, Mari Ostendorf, and Noah A Smith. 2016. Hierarchical character-word models for language identification. In *EMNLP Workshop on NLP for Social Media,*.

Aaron Jaech and Mari Ostendorf. 2015. Leveraging Twitter for low-resource conversational speech language modeling. *arXiv preprint arXiv:1504.02490*.

Aaron Jaech and Mari Ostendorf. 2017. http://ajaech.me/adeeehllnorru Improving context aware language models. *arXiv preprint arXiv:1704.06380*.

Aaron Jaech and Mari Ostendorf. 2018a. Low-rank RNN adaptation for context-aware language modeling. *TACL*.

Aaron Jaech and Mari Ostendorf. 2018b. Personalized language model for query auto-completion. In *Proc. ACL*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. In *Proc. ACL-IJCNLP*.

Fred Jelinek. 1990. Self-organized language modeling for speech recognition. *Readings in speech recognition*, pages 450–506.

Fred Jelinek. 1991. Up from trigrams. In *Proc. Eurospeech*.

Fred Jelinek, Bernard Mérialdo, Salim Roukos, and M. Strauss. 1991. A dynamic language model for speech recognition. In *Proc. HLT*.

Yangfeng Ji, Trevor Cohn, Lingpeng Kong, Chris Dyer, and Jacob Eisenstein. 2015. Document context language models. *arXiv preprint arXiv:1511.03962*.

Mika Juuti, Bo Sun, Tatsuya Mori, and N Asokan. 2018. Stay on-topic: Generating context-specific fake restaurant reviews. *arXiv preprint arXiv:1805.02400*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. In *Proc. ICLR*.

Slava Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Katrin Kirchhoff and Mei Yang. 2005. Improved language modeling for statistical machine translation. In *Proc. of the ACL Workshop on Building and Using Parallel Texts*, pages 125–128.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proc. ICASSP*, volume 1, pages 181–184.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2017. Dynamic evaluation of neural sequence models. *arXiv preprint arXiv:1709.07432*.

Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*.

Roland Kuhn and Renato De Mori. 1990. A cache-based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 12(6):570–583.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *TACL*, 5:365–378.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi S. Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *ACL*.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A persona-based neural conversation model. *Proc. ACL*.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: A simple approach to sentiment and style transfer. *arXiv preprint arXiv:1804.06437*.

Bing Liu and Ian Lane. 2017. Dialog context language modeling with recurrent neural networks. In *Proc. ICASSP*, pages 5715–5719.

Yi Luan, Yangfeng Ji, and Mari Ostendorf. 2016. LSTM based conversation models. *arXiv preprint arXiv:1603.09457*.

Min Ma, Michael Nirschl, Fadi Biadsy, and Shankar Kumar. 2017. Approaches for neural-network language model adaptation. In *Proc. Interspeech 2017*, pages 259–263.

Andrew L Maas, Ziang Xie, Dan Jurafsky, and Andrew Y Ng. 2015. Lexicon-free conversational speech recognition with neural networks. In *Proc. NAACL*.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *Proc. ICLR*.

Gideon Mendels, Erica Cooper, Victor Soto, Julia Hirschberg, Mark Gales, Kate Knill, Anton Ragni, and Haipeng Wang. 2015. Improving speech recognition and keyword search for low resource languages using web data. In *Proc. INTERSPEECH*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017a. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017b. Pointer sentinel mixture models. In *Proc. ICLR*.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. 2011. Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on*, pages 196–201.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. Interspeech*.

Tomáš Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. In *Proc. SLT*, pages 234–239.

Bhaskar Mitra and Nick Craswell. 2015. Query auto-completion for rare prefixes. In *Proc. CIKM*, pages 1755–1758.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Proc. NIPS*, pages 2265–2273.

Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Thamar Solorio. 2016. Overview for the second shared task on language identification in code-switched data. In *Second Workshop on Computational Approaches to Code Switching*, pages 40–49.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proc. AISTATS*, volume 5, pages 246–252.

Graham Neubig and Chris Dyer. 2016. Generalizing and hybridizing count-based and neural language models. In *Proc. EMNLP*.

Miles Osborne, Ashwin Lall, and Benjamin Van Durme. 2014. Exponential reservoir sampling for streaming language models. *Proc. ACL*.

Robert Östling and Jörg Tiedemann. 2017. Continuous multilinguality with language vectors. *Proc. EACL*.

Ankur P Parikh, Avneesh Saluja, Chris Dyer, and Eric P Xing. 2014. Language modeling with power low rank ensembles. In *Proc. EMNLP*.

Dae Hoon Park and Rikio Chiba. 2017. A neural language model for query auto-completion. In *Proc. SIGIR*, pages 1189–1192.

Razvan Pascanu, Tomáš Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proc. ICML*, pages 1310–1318.

Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. *InfoScale*, 152:1.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. EACL*.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Anton Ragni, Edgar Dakin, Xie Chen, Mark J F Gales, and Kate M Knill. 2016. Multi-language neural network language models. In *Proc. Interspeech*.

Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, et al. 2018. Conversational AI: The science behind the alexa prize. In *Proc. NIPS*.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proc. EMNLP*.

Giuseppe Riccardi and Allen L Gorin. 2000. Stochastic language adaptation over time and state in natural spoken dialog systems. *IEEE Transactions on Speech and Audio Processing*, 8(1):3–10.

Roni Rosenfeld. 1995. Optimizing lexical and ngram coverage via judicious use of linguistic data. In *Proc. Eurospeech*.

Roni Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187–228.

Roni Rosenfeld. 2000. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278.

Murat Saraçlar, Tunga Güngör, et al. 2010. Morphology-based and sub-word language modeling for Turkish speech recognition. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5402–5405.

Sarah E. Schwarm, Ivan Bulyko, and Mari Ostendorf. 2004. Adaptive language modeling with varied sources to cover new vocabulary items. *IEEE Trans. Speech and Audio Processing*, 12:334–342.

Holger Schwenk. 2004. Efficient training of large neural networks for language modeling. In *2004 IEEE International Joint Conference on Neural Networks*, volume 4, pages 3059–3064.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss. In *Proc. COLING*.

Noam Shazeer, Joris Pelemans, and Ciprian Chelba. 2015. Sparse non-negative matrix language modeling for skip-grams. In *Proc. Interspeech*, pages 1428–1432.

Milad Shokouhi. 2013. Learning to personalize query auto-completion. In *Proc. SIGIR*, pages 103–112.

Milad Shokouhi and Kira Radinsky. 2012. Time-sensitive query auto-completion. In *Proc. SIGIR*, pages 601–610.

Manhung Siu and Mari Ostendorf. 2000. Variable n-grams and extensions for conversational speech language modeling. *IEEE Transactions on Speech and Audio Processing*, 8(1):63–75.

Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proc. CIKM*, pages 553–562.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Proc. Interspeech*.

David Talbot and Thorsten Brants. 2008. Randomized language models via perfect hash functions. In *Proc. ACL*.

Jian Tang, Yifan Yang, Sam Carton, Ming Zhang, and Qiaozhu Mei. 2016. Context-aware natural language generation with recurrent neural networks. *arXiv preprint arXiv:1611.09900*.

Trang Tran and Mari Ostendorf. 2016. Characterizing the language of online communities and its relation to community reception. In *Proc. EMNLP*.

Bo-Hsiang Tseng, Hung-yi Lee, and Lin-Shan Lee. 2015. Personalizing universal recurrent neural network language model with user characteristic features by social network crowd-sourcing. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 84–91.

Roger CF Tucker, Michael J Carey, and Eluned S Parris. 1994. Automatic language identification using sub-word models. In *Proc. ICASSP*, volume 1, pages I–301.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. NIPS*, pages 6000–6010.

Po-Wei Wang, J. Zico Kolter, Vijai Mohan, and Inderjit S. Dhillon. 2018. Realtime query completion via deep language models. In *Proc. ICLR*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proc. EMNLP*.

Tsung-Hsien Wen, Aaron Heidel, Hung-yi Lee, Yu Tsao, and Lin-Shan Lee. 2013. Recurrent neural network based language model personalization by social network crowdsourcing. In *Proc. Interspeech*, pages 2703–2707.

Stewart Whiting and Joemon M Jose. 2014. Recent and robust query auto-completion. In *Proc. WWW*, pages 971–982.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krĩkun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Puyang Xu, Sanjeev Khudanpur, and Asela Gunawardana. 2011. Randomized maximum entropy language models. In *Proc. Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on*, pages 226–230.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*.

Dani Yogatama, Chong Wang, Bryan R Routledge, Noah A Smith, and Eric P Xing. 2014. Dynamic language models for streaming text. *TACL*, 2:181–192.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Weinan Zhang, Ting Liu, Yifa Wang, and Qingfu Zhu. 2017. Neural personalized response generation as domain adaptation. *arXiv preprint arXiv:1701.02073*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proc. NIPS*, pages 649–657.

Arkaitz Zubiaga, Inaki San Vicente, Pablo Gamallo, José Ramom Pichel Campos, Iñaki Alegría Loinaz, Nora Aranberri, Aitzol Ezeiza, and Víctor Fresno-Fernández. 2014. Overview of TweetLID: Tweet language identification at sepln 2014. In *Proc. TweetLID@ SEPLN*, pages 1–11.

Appendix A

SPARSE CORRECTIONS FOR OUTPUT LAYER ADAPTATION

In Chapter 3, we saw that adapting the output layer using a low rank term combined with a hash table that stored coefficients for individual words was helpful. The low rank term exploits similarities between contexts to boost the probability of words when appropriate. We hypothesized that there may be certain words that are unique to a particular context and are not well modeled by a low rank adaptation. The purpose of the hashed coefficients was to serve as a correction term to the low rank adaptation. In this appendix, we investigate a sparse correction term, where sparsity is achieved by using an L1 penalty during training, as an alternate method of modeling exceptions to the low rank adaptation.

A.1 Sparse plus low-rank softmax bias adaptation

We test two methods of forming embeddings for low rank adaptation of the output layer. One is to use a neural network as in Equation 2.4 to make a single embedding that summarizes all context variables. The second method is to learn individual embedding matrices for each of the n context variables. In this case Equation 2.6 is replaced with

$$y_t = \text{softmax}(\mathbf{E}^T h_t + \sum_{i=1}^{n} \mathbf{G}_i c_i + b_2), \tag{A.1}$$

which has individual adaptation terms for each context variable.

The sparse correction to the low rank adaptation bias has a similar form except that the low-dimensional context embeddings, c_i are replaced by one-hot encoded vectors, o_i.

$$y_t = \text{softmax}(\mathbf{E}^T h_t + \sum_{i=1}^{n} \mathbf{G}_i c_i + \sum_{i=1}^{n} \mathbf{A}_i o_i + b_2), \tag{A.2}$$

The $\mathbf{A}_i$ matrices will have a rank equal to the cardinality of the i-th context variable and therefore are much larger than the $\mathbf{G}^i$ matrices whose rank is set by the dimensionality of c_i. To encourage sparsity in the $\mathbf{A}_i$ matrices we apply a soft threshold operator:

$$u(s, \lambda) = \begin{cases} s + \lambda & s < -\lambda \\ 0 & -\lambda \leq s \leq \lambda \\ s - \lambda & \lambda < s \end{cases} \tag{A.3}$$

This has the identical effect as introducing an L1 penalty term in the objective except that it sets coefficients to be exactly zero if they are within a certain range. During tuning we search for the optimal penalty term λ_i for each of the four context variables. The soft thresholding operation $u(\mathbf{A}^i o_i, \lambda_i)$ is applied (the function is applied element-wise to each entry in $\mathbf{A}_i o_i$).

We make use of the same TripAdvisor data from Chapter 4 except that instead of using two context variables (hotel identifier and review sentiment), we have four: hotel identifier, month of stay, year of stay, and region. There are 2,918 hotels, 14 years (from 1999 to 2012), 12 months, and 25 regions (major cities in the United States). We hypothesized that the more detailed context would have more specialized language, which might have more potential to benefit from sparse correction.

Because of the success with the hash table, we hypothesized that sparse terms would be most useful at the output layer. Thus, we use no adaptation at the recurrent layer for the following experiments. The vocabulary size is fixed to 10,000 words (11% the size of the one used on the same data in Chapter 4) so that the model can be trained quickly using a full softmax loss. Using a sampled softmax loss may interfere with the application of the added regularization term on the softmax bias vector.

We trained 59 models using a random search strategy for hyperparameter tuning. All models used a fixed word embedding and GRU dimension of 72 and a batch size of 64. Using a smaller GRU dimension, as we did here, places more importance on the softmax layer and gives the adaptation a better chance of having an impact. The hyperparameters that varied between models were the dimensionality of the embeddings of the four context variables,

LR	Sparse	PPL
No	No	44.0
Yes	No	**41.0**
No	Yes	41.4
Yes	Yes	41.2

Table A.1: Perplexity on the validation set of models with no adaptation and varying softmax adaptation strategies. Results are not comparable to those in Chapter 4 because of a difference in vocabulary size.

the λ's for each context variable, and whether to use or disable each of the two forms of adaptation. For a model that uses no low-rank adaptation, the sparse matrix used to adapt the bias for the region variable contains 29 million parameters, 96% of the total. This is intended to be over-parameterized so that the regularization will be useful.

All of our results are on the validation set and not the test set. As seen in Table A.1, the best model used a low rank adaptation and no sparse correction. In theory, including the sparse correction should be no worse than without it as long as the λ's are properly tuned. The fact that performance was worse for the low-rank case indicates that more experiments are needed for proper tuning. Among the models that did use a sparse correction term, we found that the models that worked best had λ's equal to or so close to zero that the regularization had no effect. There was no apparent benefit to having an L1 penalty on this data set with this size of model and vocabulary.

A second finding is that using separate embeddings for each context variable as in Equation A.1 is better for adapting the softmax layer than combining them using a neural network hidden layer as in Equations 2.4 and 2.6. The difference was small (perplexities of 41.0 versus 41.3) but consistent. This indicates that the optimization of the context embedding parameters could be improved.

In predictive text applications, language models are used to suggest the next words

that someone might want to type in order to speed-up the process of typing on a mobile device. The top-3 next word prediction accuracy as a metric is a better fit to this task than perplexity. We computed the top-3 accuracy for the language models used in these experiments and found that using the "sparse"[1] adaptation strategy had a bigger impact on the accuracy than perplexity. The best model had a top-3 accuracy of 46.35% and used both low-rank and sparse adaptation compared to 46.27% for the best model using only low rank adaptation. The difference is small but significant ($p < 0.001$).

A.2 L1 Penalty for Bias Layer Fine-Tuning

We mentioned model fine-tuning as a type of adaptation in Section 2.3.1. In this section we test if including an L1 penalty on the bias term is helpful when fine-tuning a pre-trained language model to match the style of a small set of data from another domain. We created small datasets for fine-tuning by selecting the subset of TripAdvisor reviews that mention the word "Hilton" and those that are for hotels in Boston. A third dataset was created by selecting a random subset of (mostly restaurant) reviews from the Yelp dataset and then a fourth used only the reviews from the Yelp dataset that were for a dentist office.

For the pre-trained model, we used the best unadapted TripAdvisor model from Section A.1. The fine-tuning was done by continuing training of that model on the new smaller dataset until it began to over-fit. In a realistic scenario there would not be a large validation set available to check for over-fitting but in this experiment we are just aiming for a proof-of-concept. We tried selectively freezing layers of the pre-trained model and also varied the size of the training data and the scale of the L1 penalty. In each case the best result was obtained by using zero penalty and early stopping as the only regularizer.

---

[1]It is not actually sparse because the weight on the L1 regularization term was near zero.

A.3 Summary

After conducting these experiments we were unable to find a benefit from using an L1 penalty term to learn a sparse correction to a low-rank model in adapting the softmax bias. There are multiple possible explanations for this negative result.

- It is possible that the data we used is not the right one for this technique. If we had used another dataset where the size or vocabulary was different then it is possible that having a sparse correction would be helpful.

- Our use of random search for tuning the regularization penalties could be improved. We might have seen a different result with better tuning.

- We purposefully created a model that was over-parameterized. Over-parameterization can lead to over-fitting unless regularization is used. Using the L1 penalty is equivalent to setting a Laplacian prior distribution on the parameters. The regularization encourages the parameters to not be too far from zero on average. However, even without regularization, the parameter values do not grow too large during any practical length of time. The training data constrains the parameter values in the bias from becoming too large in the positive direction and. For the negative direction, the bias terms for words that are never used for particular contexts do not reach negative infinity in because their gradients decrease exponentially faster than the parameters themselves. It turns out that stopping gradient descent after a finite amount of time is a more effective means of regularizing the parameters than applying an L1 penalty.

- We hypothesized that sparse corrections would be most useful at the softmax layer, but it is possible that they are helpful at the recurrent layer or for adapting the word embeddings.

- We assumed that not adapting the recurrent layer would give the sparse softmax bias adaptation a better chance of success. It is possible this assumption is incorrect and

that having a sparse correction is only helpful after the low-rank component has been accounted for by adapting the recurrent layer.

In summary, we did not find a use for L1 regularization for adapting the softmax bias vector. More work is needed to confirm these experiments and to understand why.