NORTHWESTERN UNIVERSITY

Methods for Improving Natural Language Processing Techniques with Linguistic Regularities Extracted from Large Unlabeled Text Corpora

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Michael Ryan Lucas

EVANSTON, ILLINOIS

September 2019

Abstract

Natural Language Processing methods have become increasingly important for a variety of high- and low-level tasks including speech recognition, question answering, and automatic language translation. The state of the art performance of these methods is continuously advancing, but reliance on labeled training data sets often creates an artificial upper bound on performance due to the limited availability of labeled data, especially in settings where annotations by human experts are expensive to acquire. In comparison, unlabeled text data is constantly generated by Internet users around the world and at scale this data can provide critical insights into human language.

This work contributes two novel methods of extracting insights from large unlabeled text corpora in order to improve the performance of machine learning models. The first contribution is an improvement to the decades-old Multinomial Naive Bayes classifier (MNB). Our method utilizes word frequencies over a large unlabeled text corpus to improve MNB's underlying conditional probability estimates and subsequently achieve state-of-the-art performance in the semi-supervised setting. The second contribution is a novel neural network method capable of simultaneous generation of multi-sense word embeddings and word sense disambiguation that does not rely on a sense-disambiguated training corpus or previous knowledge of word meanings. In both cases, our models illustrate the benefit of how modern machine learning approaches can benefit from the disciplined integration of large text corpora, which are constantly growing and only becoming cheaper to collect as technology advances.

Acknowledgments

I would like to extend my deepest gratitude to my PhD advisor, Doug Downey. I found every minute we spent together invaluable and I am humbled by the depth of knowledge and experience you shared throughout our work together. Thank you for your research advice, real-world advice, teaching and leadership opportunities, and helping me through good and bad times over the years. You were my navigator through the murky world of graduate research studies and helped me find my own path to becoming a better researcher, educator, mentor, and dreamer.

I would also like to extend sincere thanks to my dissertation committee members, Brenna Argall and Haoqi Zhang, who provided indispensable feedback and direction at various stages of my research at Northwestern. Your insights into the strengths and weaknesses of my work and helped me discover and share the bigger picture of my work.

Over the years, I had countless engaging interactions with a number of other incredible professors at Northwestern University. I especially want to thank Lance Fortnow, Jason Hartline, Nicole Immorlica, Brent Hecht, Ming Kao and Jorge Nocedal for your mentorship and advice.

To Yi Yang, Chandra Sekhar Bhagavatula, Thanapon Nor, Mohammed Alam, Zack Witten, and David Demeter – my colleagues in the WebSAIL program – thank you for the countless hours we spent at whiteboards, pouring over papers, having a beer, writing code, rewriting code, and being great friends along the way. I always look forward to hearing about your recent accomplishments.

I would like to thank all of the incredible who made a significant impact on my life during my graduate education: Justine Albert, Matt McLure, Yorgos Askilidis, Mark Cartwright, Christine Hosey, Soraya Lambotte, Philipp Tillmann, Dan Zou, Dan Nguyen, Jessica Evans, Parker Guidry, Royen Kent, Micah Kronlokken, Zafar Rafii, Arefin Huq, Chen Liang, Jody Moser, Marc Palmeri, Rob & Haley Yaple, Robert Sandoval, Justin Connell, Eric Sczygelski, Martin McBriarty, Albert Lipson, Reda Al-Bahrani, Eric Sheets, Matt Montalbano, Paul Olczak, David Meyer, Douglas Boyd, Anthony Giannini, Ray Laurens, Tri Banh, Nima Haghpanah, Greg Stoddard, Bach Ha, Darrell Hoy, Jason Sendelbach, Monique Filardi, Bo Guthrie, Dustin Fox, Mat Barber, Jason Taylor, and Zane Blanton. You kept me sane. You opened my eyes to so many unique perspectives. I hold every one of you close to my heart. I hope that as we spread across the world, we find new ways to share the best of ourselves with the lives we touch.

Above all, I want to thank my family. Aunt Patsy, thank you for the years of education you gave me when I was young and PhD advice you shared with me years later. My brother Thomas will always feel like a twin to me. To sufficiently express my full appreciation of him would require another PhD of research, but I hope that all of the positive energy and work that I put into this world find their ways back to him. My parents Theresa, Bill, and Hilary can’t be thanked enough. They are my guiding lights, my shining stars in the night.

This work would not have been possible without the funding provided by Northwestern University and DARPA contract D11AP00268.

Dedication

For Frances Ryan, Willy & Mary Helen Almaraz, and Mary Lucas & Terry Vanduzee.

This work is a manifestation of the unconditional love and support you gave me at every step.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

After decades of artificial intelligence research, computational methods for interpreting natural language have yet to achieve human-level performance, in large part due to the complexity of human languages. Modern natural language processing (NLP) approaches can create approximate representations of linguistic concepts such as vocabulary and grammar, but these methods lack a true means of interacting with non-digital aspects of the world, so they lack the ability to understand human experience and the many reasons that language exists in the first place.

Fortunately, many other research fields within artificial intelligence (AI) have achieved super-human abilities in recent years without requiring an understanding of the physical world. The AI milestones of IBM's Deep Blue beating Kasparov in 1999 and Google's AlphaGo beating Lee Sedol in 2016 gathered the world's attention as search-based methods have mastered increasingly complex adversarial games. Similarly, image processing methods are being used to navigate self-driving cars (Bojarski et al., 2016) and scan high-resolution images of tissue biopsies to detect the cancer composition at a higher rate than professional cancer histologists (Sirinukunwattana et al., 2017).

The most public exhibition of state-of-the-art NLP was a performance by IBM's Watson engine over the course of three episodes of NBC's Jeopardy!, in which Watson competed against the two winningest human contestants in the show's history, Ken Jennings and Brad Rutter. Watson soundly beat both human opponents despite making occasional errors. This victory relied on pairing game-theoretic strategies with a number of advanced NLP techniques including: the parsing of short phrases, identification of explicit and implied concepts, disambiguation of words or phrases that have multiple meanings, utilization of expansive knowledge bases and ontologies, and logical chaining of concepts and “thoughts” to arrive at a single final answer for each prompt. Many of these tasks require hand-tuned symbolic reasoning approaches over extensive knowledge bases, which contrast heavily against neural network approaches which are trained over billions of words of human writing.

Historically, training the large neural networks of today’s AI research was prohibitively expensive due to the limitations of computer hardware and a lack of effective technologies for distributed computing. Theoretically interesting models such as Long Short Term Memory networks (Hochreiter and Schmidhuber, 1997) achieved state-of-the-art results over a decade after their initial conception due to drastic decreases in the cost of computation (Sundermeyer et al., 2012). Many other neural network advancements were driven by new software frameworks such as Torch (Collobert et al., 2011), Tensorflow (Abadi et al., 2016), and Keras (Chollet et al., 2015), which provide user-friendly abstractions of most fundamental neural network components, a variety of training and analysis methods, and automatic support for distributed computing.

In NLP research, many modern advancements are due to unsupervised neural networks that learn from enormous collections of high-quality human writing. These models have demonstrated a surprising ability to extract and compress the semantic meaning of vocabularies by learning one high-dimensional vector per word. A number of distinct neural network architectures have been shown to learn a final distribution of words within high-dimensional space that is also intuitive to human interpretation (e.g. synonyms are often found very close to each other). Even more incredible, these vector representations can be applied to a variety of NLP tasks that the initial neural network wasn’t explicitly learning to accomplish such as word sense disambiguation (Huang et al., 2012), language-to-language translation (Zou et al., 2013), and named entity recognition (Pennington et al., 2014). A notable advantage to training these neural networks is that they only require large amounts of human writing as a training input. Enormous data sets of news articles, books, product reviews, research publications, and conversations have been gathered from the Internet and shared to the NLP research community. Training these models over larger and more diverse data sets typically provides better results, so faster neural network frameworks and felicitous neural architectures are vital to the advancement of neural network NLP research.

The work we present in this dissertation focuses on the second direction: extending previous machine learning methods to identify and learn specific features of language from large unlabeled text corpora. Models that can efficiently learn from unlabeled text increasingly outperform state-of-the-art supervised models, especially in settings where training labels must be provided by expensive human annotators.

Our major contributions are two novel machine learning methods that can integrate a large unlabeled text corpus during training in order to drastically improve the model's representations of the English language. The first of these contributions is a semi-supervised modification to the Multinomial Naive Bayes (MNB) method for text classification. Our work demonstrates how a highly scalable measurement of word frequencies over large text corpora can be calculated once in advance and subsequently used to generate improved MNB classifiers over any binary classification task within this corpus. Our experiments show that these improved classifiers outperform existing semi-supervised models in the Sentiment Analysis and Topic Classification tasks while simultaneously incurring only a marginal cost to training time over the traditional MNB model. Historical methods for integrating large unlabeled corpora for text classification tasks are significantly slower and only outperform our proposed model in a small fraction of our experiments.

The second major contribution is a fully unsupervised neural network language model that is capable of learning and distinguishing between distinct meanings of polysemous words in the

English language. We demonstrate that an n-gram neural network language model designed to predict into high-dimensional word embedding space is capable of learning distinct vector representations for each distinct meaning of a polysemous word. Our experiments show that conflating multiple meanings of a word into a single vector has a negative effect both on the model's performance and on the interpretability of the model's learned word embedding. Although our experiments are limited to variations of our own model, we expect that the results of these experiments indicate that traditional neural network language models may be constrained by the assumption that a single vector representation per word is sufficient for modeling the English language. The most interesting result of this model is in its ability to disambiguate the meaning of polysemous words using the surrounding context words. Word Sense Disambiguation is a traditionally supervised and requires humans to manually label the meaning of each occurrence of a polysemous word, however we find that our model is capable of inferring the number of meanings of polysemous words as well as training representative word vectors for each meaning.

Chapter 2
Multinomial Naive Bayes with Feature Marginals

Multinomial Naive Bayes (MNB) is a machine learning method can be trained to classify documents as belonging to one of two or more distinct predefined classes. MNB classifiers are generic and can be used for a variety of text classification tasks (e.g. to predict whether a play was authored by Shakespeare, to estimate the star-rating of an Amazon review, or to categorize a news article within a fixed number of of news categories). Importantly, the traditional implementation of MNB is a *supervised* machine learning method, which means that it can only learn from documents that were previously tagged with the correct classification (Manning, 1999). Subsequently, every new classification task to be learned over the same text corpus requires a new labeling effort – if each document in a corpus of blog posts is labeled as being about cosmetic supplies or not, it would need to be labeled again to train a MNB classifier if blog posts are about computers or not.

Labeling a text corpus for tasks such as text classification requires human effort, which means limits the size of the labeled corpus by the cost of obtaining human labels. This is especially challenging in settings where highly-paid experts are required for labeling. In contrast, the Internet has provided a wonderful medium for global communication and humans are increasingly connecting the Internet in order to write news articles, product reviews, messages for their friends, and an almost uncountable number of other reasons. Our work in Lucas and Downey (2013) provides a method for utilizing information extracted in advance from a large unlabeled text corpora to improve the performance of the traditional MNB classifier. Our method, called Multinomial Naive Bayes with Feature Marginals (MNB-FM), requires a single pass over a large unlabeled text corpus in which we calculate word frequencies. Notably, calculating these word frequencies is efficient and easily parallelizable. MNB-FM is designed to use these frequencies to improve the MNB's classification performance for classification tasks over documents within this corpus. When training over a labeled data set, MNB-FM specifically updates MNB's per-word parameters to correct those which are over- or under-represented in any small labeled corpus given more global estimates of word frequency within the large unlabeled corpus. Existing state-of-the-art semi-supervised text classifiers require multiple iterations over the unlabeled corpus and sometimes prohibitively complex training calculations for every new classification task. MNB-FM's distinct advantages over these methods are word frequency calculations over the unlabeled corpus are both efficient to compute and are limited to a single run before any training begins. Afterwards, the frequencies be efficiently be applied to any number of classification tasks within this large text corpus. The time required during training and evaluation of MNB-FM is independent of the size of the unlabeled corpus. Furthermore, MNB-FM generally outperforms supervised classifiers that use larger labeled data sets (details in Section 2.4), allowing us to explore practical trade-offs such as labeling smaller data sets over many more classification tasks while still achieving the same classification performance of previous state-of-the-art methods.

In this chapter, we will explain the common text classification tasks of Topic Classification and Sentiment Analysis (Section 2.1) as well as the standard design of the MNB classifier and its limitations when trained over small labeled data sets (Section 2.2). Next, we will describe the Multinomial Naive Bayes with Feature Marginals (MNB-FM) classifier published in Lucas and Downey (2013) (Section 2.3). We demonstrate how MNB-FM's improved parameter estimates increase its performance in the Topic Classification and Sentiment Analysis tasks (Sections 2.4 and 2.5). Finally, we explain a method for expanding the MNB-FM classifier to the multi-class setting (Section 2.6) and follow up on unexpectedly poor results of the MNB-FM model in Zhao et al. (2016) (Section 2.7).

2.1 Topic Classification and Sentiment Analysis

Machine learning research in natural language spans a large number of challenges, from translating a document between two languages to identifying which Wikipedia article is most relevant to a mention of Chicago (the Broadway musical or the city in Illinois) to identifying the verb in a sentence such as "The old man the boats" (the answer is "man," though it's not obvious even to humans). In our work on extending the Multinomial Naive Bayes classifier to the semi-supervised setting, our experiments were limited to the tasks of Topic Classification and Sentiment Analysis. This section will explain both tasks as well as the numerous metrics that are used to measure performance of text classifiers.

Topic Classification. Given a fixed set of known document labels, *topic classification* is the challenge of identifying which labels are relevant to a given document. For example, one might expect that a newspaper article titled "President Obama Celebrates Chicago Cubs' World Series Win at the White House" should be labeled as a sports article instead of a politics article even though Barack Obama and the White House are mentioned in many political articles. The article text will likely focus more on sports than on politics, but this case illustrates how topic classification can be challenging for machine learning methods – it's simply not enough to count topic-related words or concepts. The overall context must be taken into account, which is much more difficult.

The task of topic classification specifically assumes that for any given set of target topics, **T**, there is a function **f** that can map individual documents to the subset of topics each contains:

$$f_T : d \rightarrow T_d \subseteq T$$

Machine learning methods for topic classification assume that the hypothetical $\mathbf{f_T}$ can be approximated given a large enough set of documents with corresponding accurate topic labels. Most text classification methods convert the multi-topic classification problem of into a series of one-versus-all problems that rely on classifying **|T|** *binary classifiers*. Each binary classifier is responsible for generating a prediction of whether its topic applies to a given document by approximating a theoretical topic-specific labeling function:

$$f_i : d \rightarrow \{True, False\}$$

Depending upon the machine learning classifier, binary classification may require heavily redundant computations when |**T**| dependent classifiers are independently trained. The results we explain in Section 2.5 compare implementations of MNB, MNB-FM, and state-of-the-art topic classifiers in the binary classification setting.

Sentiment Analysis. In Sentiment Analysis, document labels are focused on subjective aspects of the writing, such as whether the writer is happy or enjoyed a recent restaurant visit. One can imagine that phone manufacturers are interested in whether online discussions about a newly released phone are generally positive or negative, but that it would cost too much money to pay employees to read and label every conversation. Instead, training a Sentiment Analysis classifier on a smaller set of labeled phone reviews can allow the manufacturer to quickly assess whether the general reaction to the phone is positive or negative at a much lower cost.

We limit our experiments to binary classification, but provide multi-class extensions to our work in Section 2.6 (page 39). Multi-class sentiment analysis models can be trained to distinguish between multiple emotions such as happiness, sadness, frustration, and indifference, but a separate approach is to use one-vs-many binary classifiers that learn one target emotion per classifier. Our work focuses on a single sentiment for which there are large available data sets: whether an online product review is positive or negative. We use a well-known collection of Amazon product reviews and assume that any review with a rating below 3 stars is negative, above 3 stars is positive, and discard 3-star ratings altogether. The Amazon sentiment classification data set Blitzer et al. (2007) is described in more detail in Section 2.3 (page 24).

Our experiments on the Blitzer et al. (2007) data set accurately reflect the above example of determining whether a cell-phone review is positive or negative. Sentiment classification of this data set is traditionally trained and evaluated separately for each distinct category of product, which allows researchers to identify if there are category-specific differences in language that can assist or hinder various classifiers. For example, the word 'buckle' in a product review may indicate that a ladder collapsed (indicating a likely negative review) or refer to a piece of an article of clothing (which is unlikely to indicate review polarity). In Sentiment Analysis, the binary classification task for each product category is reduced to independently approximating functions of the form:

$$f_i : d \rightarrow \{Positive, Negative\}$$

Evaluation Metrics. To evaluate the performance of various machine learning methods on Topic Classification and Sentiment Analysis, we use standard binary classification metrics. Individual binary classifiers are trained to predict whether documents in a training corpus **D** belong to a specific topic $\mathbf{T_i}$. In the supervised setting, human labels are provided for every document in **D** that assert whether the document belongs to topic $\mathbf{T_i}$. If **D** was selected uniformly at random from a larger pool of documents, classification methods that use prior estimates of the frequency of topic $\mathbf{T_i}$ will often perform much better when applied to the documents that were not selected for labeling. Once the training stage is complete, we can objectively evaluate the classification performance of a classifier by having it predict the labels of documents in a held-out evaluation corpus, **D***.

The metrics we use to evaluate model performance directly compare the models' predictions for each document in the evaluation corpus against its corresponding human label. For documents in **D*** that a human labeled as belonging to topic $\mathbf{T_i}$, correct predictions by any classifier are known as t*rue positives* (often abbreviated as **TP**), while incorrect predictions are known as *false positives* (**FP**). Similarly, the documents labeled by the classifier as not belonging to topic **T**$_i$ are either correct, *true negatives* (**TN**), or incorrect, *false negatives* (**FN**).

The above intermediate collations provide the foundation for higher-level metrics, which are designed to more accurately convey classification performance because simple metrics can be misleading in a variety of settings. One of the most basic classification metrics is *accuracy* – the fraction of predictions over the test corpus that were correct:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

One can imagine a number of settings in which accuracy may not be the ideal measure. For example, it may be of critical importance that a binary classifier designed to diagnosis medical patients always identifies patients who are positive for the disease, even if the classifier has a tendency to overly predict the presence of disease because a doctor can perform a followup analysis without risking that the disease goes undiagnosed. In such a setting, the *recall* metric explains a model's ability to detect all at-risk patients:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Conversely, the *precision* metric is useful in settings where it is much more important to be correct when a positive prediction is made than to incorrectly classify a negative as a positive. One example is an essay plagiarism classifier, which would be responsible for detecting whether a student has submitted an essay that likely plagiarizes a known document. It would be dangerous to rely on such a model unless it was known that the model rarely flags a paper unless it is plagiarized. Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

F1-Score. One of the most important metrics used for evaluating binary classification models natural language applications is the *F1-Score*, a simplified version of the *F-Measure*, which strikes a balance between Precision and Recall and only yields a perfect score when both scores are perfect, but yields a 0% if either Precision or Recall is 0. The F-Measure is frequently used in place of accuracy when the positive and negative classes are heavily imbalanced, which is often the case in natural language tasks where identification of a rare text topic is common. F-Measure uses a predefined parameter **β** and is defined as follows:

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} = \frac{(1+\beta^2) \cdot TP}{(1+\beta^2) \cdot TP + \beta^2 \cdot FN + FP} \tag{4}$$

The best value for **β** depends upon the relative importance of Precision and Recall for the given application. When **β**<1, F-Measure is more heavily weighted toward the recall metric and similarly when **β**>1, it is weighted more toward precision. For our experiments, we rely on the F-Measure metric to evaluate classification performance because F-Measure penalizes classifiers that over-classify toward the majority class in highly-skewed data sets, while accuracy encourages this behavior. Our experiments do not have a motivation for favoring either precision or recall, so we set **β**=1 for our use of the F-Measure. This is also known known as the *harmonic mean* of precision and recall or the F1-Score:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

2.2 Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is a classic machine learning technique for binary document classification. MNB classifiers learn from a set of training documents that are labeled as either positive (belonging to a specified target class) or negative. For example, if a company wanted to determine whether new blog posts about their newly released product was positive or negative, the company could train a MNB classifier over the text of known positive and negative reviews for similar products, feed the text of a new blog post into the MNB classifier, and receive a prediction of whether model believes the author had a positive sentiment for the product..

Traditionally, the MNB classifier is a *supervised* machine learning method – it learns from a training set of documents with known labels. For the above task, a training set could be cheaply collected by purchasing an existing collection of Amazon reviews or automating a web browser to navigate Amazon product listing pages, extracting customer review text and ratings, and labeling each review as positive or negative based on its associated star rating. Regardless of the method used to collect the text of the Amazon reviews, we assume that the corresponding star-rating for each review is also included, which makes labeling the training reviews essentially free compared to the cost of collecting the reviews themselves. Amazon reviews with 4-5 stars (out of 5) are typically interpreted as positive labels, while reviews with 1-2 stars are labeled as negative and reviews with 3 stars are discarded.

Training. At its core, the MNB classifier estimates conditional word probabilities over a labeled text corpus so that it can then classify future unseen documents. In the training data set, each document is labeled as positive or negative, depending on whether or not it belongs to the target class. The MNB model ignores order of text within a document and instead a sparse *bag-of-words* representation for each document as its input, $\mathbf{d = \{w^1, \ldots, w^{|V|}\}}$, where $\mathbf{w^i}$ is the number of times word $\mathbf{i}$ occurs in the given document. This clearly has the effect of removing any contextual or nuanced information that must be discerned from word order. For example, the bag-of-words representation of an Amazon review such as "The shipping time was awful, but the phone itself is amazing" is equivalent to that of "The shipping time was amazing, but the phone itself is awful," but we expect the second review to have a much lower star-rating than the first.

Training a MNB classifier boils down to the calculation of *conditional word frequencies*. For each word, we calculate the word's frequency within the positive and negative classes separately.

For the positive class, these conditional word frequencies are represented as **P(w$_i$|C) –** the probability that a word token drawn uniformly at random from all known documents of class **C** will be word **w$_i$**. For clarity, we use **P(w$_i$|¬C)** to represent word's conditional probability over the negatively labeled training documents (i.e. documents that belong to "not the class"). **P(w$_i$|C)** is calculated over the labeled training documents according to Equation 6.

$$P(w_i|C)=\frac{\sum_{d\in D_C} w^i}{\sum_{d\in D_C}\sum_{w^j\in d} w^j} \tag{6}$$

Calculation of these feature marginals is straightforward: Take one pass over the training set and accumulate word counts for every unique word in the training set's vocabulary, distinguishing between occurrences in positive and negative documents. Then for each word, calculate the ratio of occurrences of that word vs all words separately for both classes.

Classification. Given an unknown document **d = {w^1, …, w$^{|V|}$}**, the MNB classifier determines the likelihood that **d** was drawn from the class **C** using the marginal probabilities according to the Bayes' theorem of conditional probability:

$$P(C|d)=\frac{P(C)P(d|C)}{P(d)}=\frac{P(C)P(w^1,\ldots,w^{|V|}|C)}{p(w^1,\ldots,w^{|V|})} \tag{7}$$

The MNB classifier assumes that the class that produces the highest probability according to Equation 7 is the most likely class for the given document. Our experiments are limited to binary classification, so Equation 7 is evaluated twice – once to estimate the probability of the positive class using conditional word frequencies estimated over the positive training documents and a second time for the negative class. Note that the denominator in Equation 7 is equal for both classes, so calculations can be simplified by calculating and comparing only the numerator for each class. One can iteratively simplify this calculation as follows:

$$P(C|d) \propto P(C)P(d|C)$$
$$P(C|d) \propto P(C)P(w_1|C)^{w^i} P(w_2, \ldots, w_{|V|}|C) \tag{8}$$
$$P(C|d) \propto P(C)P(w_1|C)^{w^i} P(w_2|C, w_1)^{w^2} \ldots P(w_{|V|}|C, w_{1,} \ldots, w_{|V|-1})^{w^{|V|}}$$

MNB classifications that use Equations 7 or 8 are generally impossible to calculate for documents of any interesting length because they would require many documents with exactly the same bag-of-words representation to be labeled, but human writing has incredible variety and we don't expect to see this in practice. Instead, the numerator of Equation 7 can be simplified if one makes the naive Bayes assumption – the MNB classifier assumes pair-wise independence in the conditional probabilities between words, allowing the following simplification:

$$P(C|d) \propto P(C)P(w_1|C)P(w_2|C) \ldots P(w_{|V|}|C)$$
$$P(C|d) \propto P(C) \prod_{w \in d} (p(w_i|C)^{w^i}) \tag{9}$$

Smoothing Techniques. MNB implementations require the use of a *smoothing technique* to give non-zero probability to words that haven't been seen. One argument in favor of naively smoothing 0-probability words is that any word should have a non-zero probability of occurring in any context, especially given that the conditional probabilities are estimated over small labeled data sets in the supervised learning setting. The standard baseline smoothing technique is A*dd-1 Smoothing*:

$$\theta_w^C = \frac{N_w^C + 1}{N^C + |V|} \tag{10}$$

Unknown words. When classifying documents that don't belong to the training set, it is not uncommon to encounter a word that didn't occur in the positive or negative training documents. For a word that did not occur in a specific class's labeled training documents, a literal implementation of Equation 9 would assign 0 probability to that class when $P(w_i|C)=0$ for any word **w_i**. Many MNB implementations add an unknown word to the vocabulary, $\mathbf{w_{unk}}$. The conditional frequency assigned to $\mathbf{w_{unk}}$ in each class depends on the implementation, but the most common values are 1 (because of Add-1 Smoothing) or to combine the rarest words in the labeled corpus order to estimate the likelihood of new unknown rare words. This combination process is often called *pruning,* which involves replacing all words that occur less than a fixed number of times over the training set (often 3) with $\mathbf{w}_{\mathbf{unk}}$. This typically reduces the model's understanding of any individual rare word, but allows it to accurately model the general likelihood of rare words in aggregate.

2.3 Augmenting Multinomial Naive Bayes with Feature Marginals

This section and the remaining sections of Chapter 2 discuss our published results in Lucas and Downey (2013). The traditional Multinomial Naive Bayes (MNB) classifier is a *supervised* machine learning method, meaning it requires each document in the training corpus to be hand-labeled, but we are motivated by classification tasks for which document labels are not affordable on a large scale. Creating a data set of news articles with labels of political and apolitical would require collecting a large number of news articles, which is very cheap compared to having a human annotator read each article and label it according to the target class for a training corpus. There are a number of existing news article data sets that include topic labels assigned by human annotators for each document (Lewis et al., 2004; Yang and Liu, 1999), but more challenging labeling tasks are inherently more expensive. Another common caveat is that human annotators may disagree about subjective labels or provide an incorrect label altogether, so accurately labeled data sets generally require multiple human annotators per document and agreement metrics such as inter-rater reliability are used to verify the accuracy of the training labels (Artstein and Poesio, 2008). It is no surprise then that Amazon review data sets, which assume that written reviews about purchased products are in agreement with the reviewer's assigned numeric rating (Amazon's rating system is a range of 1 to 5 stars), are significantly cheaper to collect than news topic data sets.

There are a number of important natural language tasks for which we still need hand-labeled data sets to train machine learning algorithms. To collect large training sets affordably, one obvious approach is to minimize the per-label cost. Alternatively, the same improvement achieved by labeling larger data sets can often be obtained by maximizing the amount of information gleaned from each human annotation.

A variety of approaches have been designed to augment small human-labeled data sets by including external sources of data. *Semi-Supervised Learning* (SSL) is a category of machine learning methods that learn from both labeled and unlabeled training instances of the same type (documents in our case), taking distinct advantage of the fact that unlabeled data sources are constantly growing and becoming more cheaply available while hand-labeling remains expensive. SSL methods have shown that intelligent use of unlabeled data can often lead to a drastic increase in the performance of machine learning methods (Chapelle, 2006).

In Lucas and Downey (2013), we consider the scalability concerns of existing SSL methods for binary document classification. Existing approaches such as Expectation-Maximization (Nigam et al., 2000) and Label Propagation (Zhu and Ghahramani, 2002) require multiple passes over the entire unlabeled corpus, but this often limits the size of the unlabeled text corpus that can be utilized by these methods. Of further concern is that some SSL methods can't transfer knowledge extracted from the unlabeled corpus to another model, so training must begin anew for each new target concept. Our approach, which we call Multinomial Naive Bayes with Feature Marginals (MNB-FM), utilizes word counts over a very large unlabeled corpus to improve existing classifiers that can be trained over different classes at a later time. These word counts will not change, so they only need to be computed once in advance and can then be utilized to improve any number of MNB classifiers.

The traditional MNB classifier requires a training set of documents that are labeled as to whether or not each document belongs to a target class. At training time, MNB calculates every word's frequencies in both the positive and negative labeled document classes. These word frequencies are the parameters the MNB classifier uses to generate predictions on new unseen documents. Our MNB-FM model relies on the fact that most labeled text training sets are relatively small compared to some of the recently available text corpora collected for NLP research that contain sometimes billions of words (Chelba et al., 2013). Calculating word counts over extremely large text corpora can be made more efficient with distributed computation methods and does not rely on human-curated labels.

MNB-FM utilizes these word frequencies from the large unlabeled text corpus to improve the accuracy of the MNB classifier's marginal probabilities, especially for words that have large difference in estimated frequencies between the labeled and unlabeled data sets. One clear advantage to this method is that only word counts are recorded over the large unlabeled corpus, which is a relatively compact amount of information and counting word tokens scales easily for massive corpora. A second advantage is that MNB-FM improves the estimates of MNB's conditional probabilities while simultaneously accounting for the fact that natural language data sets are often heavily skewed – for example, Amazon reviews in the Music category of the Amazon Aptemod data set are 91.7% positive (4 or 5 stars in a scale from 1 to 5) (Blitzer et al., 2007). We will show how MNB-FM's MLE constraint (Equation 17, page 29) accounts for imbalances between the token counts within the positive and negative labeled document classes and accordingly adjusts the conditional probability estimates.

We compared MNB-FM against both Supervised and Semi-Supervised Learning methods: Multinomial Naive Bayes, Naive Bayes with Expectation Maximization (Nigam et al., 2000), Logistic Regression (Cox, 1958), Label Propagation (Zhu and Ghahramani, 2002), and a recent state-of-the-art method known as Semi-supervised Frequency Estimation (Su et al., 2011). Our experiments require drastic limitations of the Label Propagation method due to its computational requirements and the number of documents in our unlabeled corpora. Our experiments demonstrated that MNB-FM was faster and more accurate than the competing SSL methods.

MNB-FM. In Lucas and Downey (2013), we utilized accurately measured *feature marginals* (single-word frequencies) from a large unlabeled corpus, $\mathbf{D_U}$, to improve the noisy conditional probability estimates that the MNB classifier typically only learns using a small labeled corpus. MNB-FM showed that these simple measurements were sufficient to smooth the MNB's conditional probabilities more accurately than simple smoothing methods such as Add-1 Smoothing (Equation 10) and state-of-the-art semi-supervised text classifiers. After calculating the feature marginals over the unlabeled text corpus, our MNB-FM method can be used to improve any MNB classifier. Consider the following equation:

$$\begin{aligned} P(w) &= \theta_w^C P_t(C) + \theta_w^{\neg C} P_t(\neg C) \\ &= \theta_w^C P_t(C) + \theta_w^{\neg C} (1 - P_t(C)) \end{aligned} \tag{11}$$

Equation 11 decomposes the marginal probability of word $\mathbf{w}$ into its conditional probabilities given the positive and negative classes, and $\mathbf{P_t(C)}$, the probability that a randomly drawn token from the labeled data set, $\mathbf{D_L}$, occurs in a document of class $\mathbf{C}$ can be calculated using Equation 12**:**

$$P_t(C) = \frac{\sum_{d \in D_C} |d|}{\sum_{d \in D_L} |d|} \tag{12}$$

The equality in Equation 11 holds for estimates of $\mathbf{P(w)}$ over the labeled corpus. However, if we assume that that the language in the unlabeled and labeled corpora are very similar, we can rely on the more reliable estimate of $\mathbf{P(w)}$ by replacing it with $\mathbf{P_U(w)}$, the marginal probability for the same word over the much larger unlabeled corpus. In our work, we assume that we can substitute $\mathbf{P_U(w)}$ directly for $\mathbf{P(w)}$ in Equation 11, but this will break the equality in most cases. The key idea in MNB-FM is to assume that **P$_U$(w)** is the correct value of word **w**'s frequency, so it is similarly logical to assume that at least one of the three remaining parameters in Equation 11 must also be adjusted for the equality to hold. Each word token in the training corpus contributes to the estimate of **P$_t$(C)**, so we assume that **P$_t$(C)** is accurately estimated by the training corpus and use MNB-FM to identify more accurate values for the conditional probabilities θ_w^C and $\theta_w^{\neg C}$.

Solution Derivation. MNB-FM uses a Maximum Likelihood Estimation (MLE) approach to adjust the marginal probabilities. For a given word **w**, the maximum likelihood estimates of θ_w^C and $\theta_w^{\neg C}$ given the training data are:

$$\begin{aligned} &\underset{\theta_w^C, \theta_w^{\neg c}; \forall w \in V}{argmax} \; P(D_L | \theta_w^C, \theta_w^{\neg C}) \\ &= \underset{\theta_w^C, \theta_w^{\neg c}; \forall w \in V}{argmax} \; (\theta_w^C)^{(N_w^C)} (1-\theta_w^C)^{(N_{\neg w}^C)} (\theta_w^{\neg C})^{(N_w^{\neg C})} (1-\theta_w^{\neg C})^{(N_{\neg w}^{\neg C})} \\ &= \underset{\theta_w^C, \theta_w^{\neg c}; \forall w \in V}{argmax} \; N_w^C \ln(\theta_w^C) + N_{\neg w}^C \ln(1-\theta_w^C) + N_w^{\neg C} \ln(\theta_w^{\neg C}) + N_{\neg w}^{\neg C} \ln(1-\theta_w^{\neg C}) \end{aligned} \qquad (13)$$

Note that the third line in Equation 13 is the natural log of the second line, though steps were skipped in simplifying the distribution of the natural log function. Taking the natural log is a *strictly monotonic increasing* transformation of line 2, so maximizing line 3 is equivalent to maximizing line 2. Before we take the derivative, we rewrite the constraint from Equation 11 as:

$$\theta_w^{\neg C} = K - \theta_w^C L \qquad (14)$$

Where we made the following substitutions for compactness and interpretability of the subsequent equations:

$$K = \frac{P(w)}{P_t(\neg C)} ; L = \frac{P_t(C)}{P_t(\neg C)} \qquad (15)$$

By substituting Equation 14 into Equation 13, the optimization is reduced to one free variable:

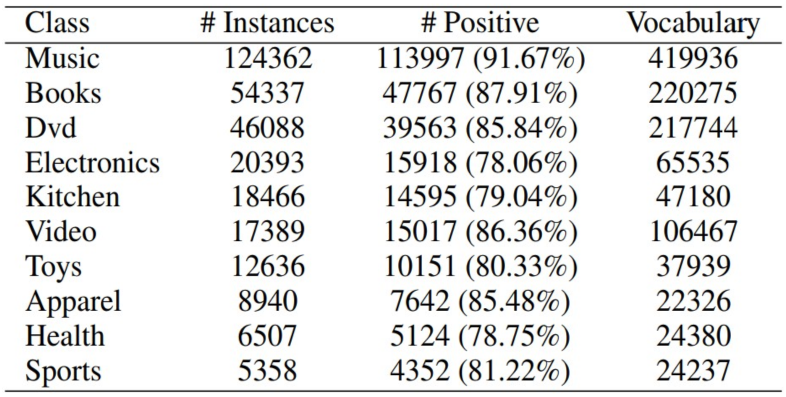
$$\begin{aligned} &\underset{\theta_w^C,\,\theta_w^{\neg c}}{argmax}\, P\left(D_L|\theta_w^C,\theta_w^{\neg C}\right) \\ &= \underset{\theta_w^C}{argmax}\, N_w^C \ln\left(\theta_w^C\right) + N_{\neg w}^C \ln\left(1-\theta_w^C\right) + N_w^{\neg C} \ln\left(K - L\theta_w^C\right) + N_{\neg w}^{\neg C} \ln\left(1-K+\theta_w^C\right) \end{aligned} \tag{16}$$

Now, to identify the values which maximize Equation 16, we take its derivative. The optimal values for θ_w^C can be found at the solutions of Equation 17:

$$0 = \frac{N_w^C}{\theta_w^C} + \frac{N_{\neg w}^C}{\theta_w^C - 1} + L\frac{N_w^{\neg C}}{L\theta_w^C - K} + L\frac{N_{\neg w}^{\neg C}}{\theta_w^C - K + 1} \tag{17}$$

There are four solutions to this equation, but θ_w^C and $\theta_w^{\neg C}$ are probabilities and are thus constrained to values in [0,1]. This only occurs for both probabilities when $0<\theta_w^C<\frac{K}{L}$. If N_w^C and $N_w^{\neg C}$ have non-zero counts, then vertical asymptotes at 0 and $\frac{K}{L}$ guarantee a solution in this range. We use the Newton-Rhapson method to solve this equation (Newton, 1969). If an answer does not exist in the valid range, we default to Add-1 Smoothing of the conditional probabilities, a standard practice of the MNB classifier. Finally, after optimizing θ_w^C and $\theta_w^{\neg C}$ for every word, we normalize the sum of the marginals for each class to 1 in order to obtain valid probabilities.

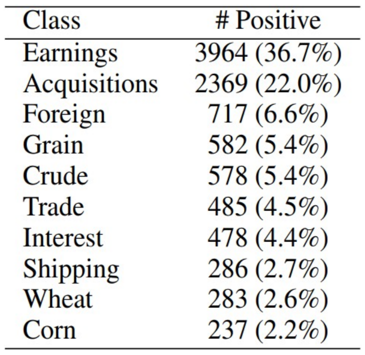
Data Sets. A number of classification experiments were conducted over multiple data sets to determine whether MNB-FM was capable of improving MNB's classification performance. We compared our results against those of existing state-of-the-art Supervised and Semi-Supervised methods and used multiple data sets to conduct our evaluations, specifically Amazon sentiment analysis data set (Blitzer et al., 2007), the ApteMod Reuters news corpus (Yang and Liu, 1999), and the RCV1 news corpus (Lewis et al., 2004).



Class	# Instances	# Positive	Vocabulary
Music	124362	113997 (91.67%)	419936
Books	54337	47767 (87.91%)	220275
Dvd	46088	39563 (85.84%)	217744
Electronics	20393	15918 (78.06%)	65535
Kitchen	18466	14595 (79.04%)	47180
Video	17389	15017 (86.36%)	106467
Toys	12636	10151 (80.33%)	37939
Apparel	8940	7642 (85.48%)	22326
Health	6507	5124 (78.75%)	24380
Sports	5358	4352 (81.22%)	24237

Table 1. Summary of the Amazon reviews data set from Yang and Liu (1999)*, reprinted from Lucas and Downey (2013).*

The Amazon data set is composed of online product reviews and was designed for Sentiment Analysis evaluations. Specifically, each document includes the text of a review from the internet retailer Amazon.com as well as the reviewer's corresponding rating of the product. A 4-star or 5-star rating is interpreted as *positive*, while a 1-star or 2-star rating is *negative* and 3-star ratings are discarded as ambiguous. The classifiers are trained to determine whether a product review is positive based solely upon the review text. As Table 1 demonstrates, the Amazon categories are skewed strongly toward 4- and 5-star reviews, presumably because higher-rated items are purchased more frequently, which biases the overall likelihood of a positive review.



Class	# Positive
Earnings	3964 (36.7%)
Acquisitions	2369 (22.0%)
Foreign	717 (6.6%)
Grain	582 (5.4%)
Crude	578 (5.4%)
Trade	485 (4.5%)
Interest	478 (4.4%)
Shipping	286 (2.7%)
Wheat	283 (2.6%)
Corn	237 (2.2%)

Table 2. Summary of the Reuters ApteMod data set from Blitzer et al. (2007), reprinted from Lucas and Downey (2013).

Class	# Positive
CCAT	381327 (47.40%)
GCAT	239267 (29.74%)
MCAT	204820 (25.46%)
ECAT	119920 (14.91%)
GPOL	56878 (7.07%)

Table 3. Summary of the RCV1 data set from Blitzer et al. (2007), reprinted from Lucas and Downey (2013).

The Reuters Aptemod and RCV1 data sets are two traditional news article data sets collected for topic classification. Each data set specifies a series of news topics that machine learning classifiers are expected to learn to identify (e.g. “economics” or “agriculture”). ApteMod consists of only 10,788 categorized news articles, each of which can have multiple classes. Our experiments were limited to the 10 largest classes, which are listed in Table 2. The class skew varies considerably, but only Earnings comes close to a 50/50 split (which makes it difficult for models to simply default to one class or another).

ApteMod was a popular data set for many years, but is now considered too small to model the “large unlabeled data sets” that motivate semi-supervised techniques. RCV1 is a Text Classification data set of news articles with multiple labeled categories, as with the ApteMod data set, however the document categories are nested hierarchically, so we took the 5 largest base classes, shown in Table 3.

In order to evaluate the various methods, we trained each one on small labeled data sets of 10, 100, or 1000 documents. These training sets were selected as uniform random subsets of 1000 documents from the training corpus. The smaller training sets were simply the first 10 or 100 documents generated for each training set of 1000 documents. The remaining (N-1000) documents were used as the evaluation set, but the SSL methods were given these (N-1000) documents as the unlabeled training corpus, which is a practice known as Transductive Learning.

Competing Methods. In order to evaluate the performance of our model against other likely choices, we used a variety of Supervised and Semi-Supervised classifiers. The first was an implementation of Naive Bayes with Expectation Maximization based on (Nigam et al., 2000). We selected our parameters on a separate data set (Mann and McCallum, 2010) and found that they slightly outperformed the published results, validating our design choices. Our implementation ran Expectation Maximization for 15 iterations and weighted unlabeled examples as 1/5 the weight of labeled examples.

Our Logistic Regression implementation used L2-Normalization, which we found to outperform the L1-Normalized and Non-normalized implementations for our data sets. The strength of the normalization was a parameter that we selected using cross-validation separately for each training set. The selection of this parameter was based upon F1-Score using 10-fold cross-validation.

To provide a more broad comparison of state-of-the-art semi-supervised methods, we also implemented a heavily constrained version of label propagation (Newman, 2004). A true implementation of label propagation creates an edge between all pairs of documents in both the labeled and unlabeled corpora. Given that the number of edges in a fully connected graph is proportional to the square of the total document count, this becomes prohibitively large to model for larger data sets, including those that we evaluate. Instead, our implementation of label propagation calculates a tf-idf weighted bag-of-word vector for each document in the labeled and unlabeled corpus and uses cosine similarity (Equation 29, page 71) to connect each document to its 10 most similar documents.

In label propagation, the labeled documents propagate their labels using their connections to unlabeled documents. If the entire corpus is connected, we assume that the labels will eventually propagate through the network over a number of propagation runs. During classification, the model classifies each unlabeled document according to the class with the higher weight. We limited our implementations to 100 iterations of the label propagation algorithm. Even with these aggressive constraints, label propagation was only able to be evaluated over the smallest 10 of our 20 data sets.

Finally, we implemented a version of a recent model called Semi-Supervised Frequency Estimate (SFE) (Su et al., 2011). Like MNB, SFE requires a smoothing method, but Su et al. (2011) did not specify one. Our implementation uses Add-1 Smoothing, which yields similar results in our experiments to those the published by Su et al. (2011), but may not be optimal. SFE is designed to improve MNB with marginal probability estimates **P(w)** computed over an unlabeled corpus, but differently to the approach in MNB-FM, by leveraging the following equality:

$$P(\mathrm{C}|w) = \frac{P_C(w)}{P(w)} \tag{18}$$

SFE then uses the **P(w)** estimate from the corpus to adjust both **P(C|w)** and **$\mathbf{P_C}$(w)**. However, it does so in a way that adjusts **P(C|w)** and **$\mathbf{P_C}$(w)** the same amount, where we use word counts to estimate the accuracy of our feature marginals and adjust the probabilities accordingly.

2.4 Experimental Results

The primary results of our classification experiments are shown in Table 4. Unfortunately, Label Propagation scales too inefficiently to run within the memory limits of our evaluation servers for all of our data sets, but results on the ApteMod data set and the 5 smallest Amazon classes are included for comparison.

Data Set	MNB-FM	SFE	MNB	NBEM	LProp	Logist.
Apte (10)	0.306	0.271	**0.336**	0.306	0.245	0.208
Apte (100)	**0.554**	0.389	0.222	0.203	0.263	0.330
Apte (1k)	**0.729**	0.614	0.452	0.321	0.267	0.702
Amzn (10)	**0.542**	0.524	0.508	0.475	0.470*	0.499
Amzn (100)	**0.587**	0.559	0.456	0.456	0.498*	0.542
Amzn (1k)	0.687	0.611	0.465	0.455	0.539*	**0.713**
RCV1 (10)	**0.494**	0.477	0.387	0.485	-	0.272
RCV1 (100)	**0.677**	0.613	0.337	0.470	-	0.518
RCV1 (1k)	0.772	0.735	0.408	0.491	-	**0.774**

* Limited to 5 of 10 Amazon categories

Table 4. F1-Score for various text classification experiments, number of documents in the labeled corpus in parentheses (Lucas and Downey, 2013).

The F1-Score evaluations from Lucas and Downey (2013) are summarized in Table 4. Not only did MNB-FM drastically improve upon the performance of the standard MNB classifier in most settings, but it outperformed competing state-of-the-art semi-supervised classification methods on 6 of the 9 binary classification tasks. Specific breakdowns of topic classifications within the RCV1 data set on smaller labeled training set sizes provide deeper insight into the results:

Class	MNB-FM	SFE	MNB	NBEM	Logist.
CCAT	0.641	**0.643**	0.580	0.639	0.532
GCAT	0.639	0.686	0.531	**0.732**	0.466
MCAT	**0.572**	0.505	0.393	0.504	0.225
ECAT	**0.306**	0.267	0.198	0.224	0.096
GPOL	0.313	0.283	0.233	**0.326**	0.043
Average	**0.494**	0.477	0.387	0.485	0.272

Table 5: F1-Score over the RCV1 corpus with with a labeled corpus of 10 documents (Lucas and Downey, 2013).

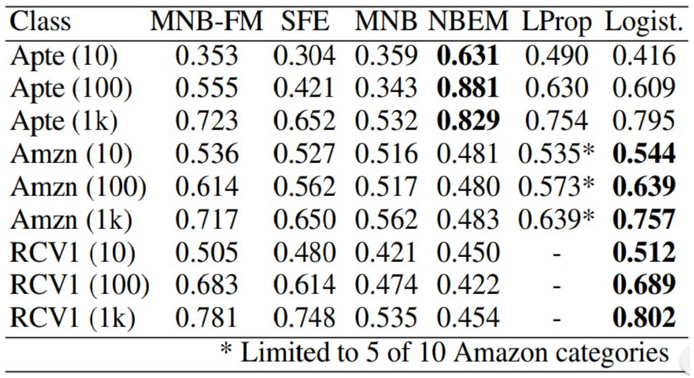
Class	MNB-FM	SFE	MNB	NBEM	Logist.
CCAT	**0.797**	0.793	0.624	0.713	0.754
GCAT	**0.849**	0.848	0.731	0.837	0.831
MCAT	**0.776**	0.737	0.313	0.516	0.689
ECAT	**0.463**	0.317	0.017	0.193	0.203
GPOL	**0.499**	0.370	0.002	0.089	0.114
Average	**0.677**	0.613	0.337	0.470	0.518

Table 6: F1-Score over the RCV1 corpus with with a labeled corpus of 100 documents (Lucas and Downey, 2013).

With 100 labeled documents, MNB-FM outperforms every method on the 5 RCV1 classes (Table 6). We can also see that SFE, a state-of-the-art semi-supervised method, performs very similarly in the 3 largest topics, but MNB-FM notably outperforms all competing methods over the 2 smaller topics where labeled document distribution is much more heavily skewed away from the target topic (ECAT and GPOL respectively account for 15% and 7% of the RCV1 corpus).

Limiting our training set to only 10 labeled documents, we can see that MNB-FM outperforms all competing methods in only 2 out of 5 classes (Table 5). Notably, MNB-FM's performance remains competitive in every topic, especially in those with highly-skewed class distributions. Both Logistic Regression and Label Propagation perform poorly in these cases because they tend to default to the majority class when the labeled data set is highly skewed (this results in high accuracy, but F1-Score penalizes this form of over-fitting).

Analysis – While calculating the F1-Score in our experiments, we also evaluated each method's ability to rank documents using the *R-Precision* metric. R-Precision doesn't require a perfect ordering of the test documents. Instead it measures the Precision (Equation 3, page 19) of the **R** test documents the most positively, where **R** is the total number of positive instances in the test set.



Class	MNB-FM	SFE	MNB	NBEM	LProp	Logist.
Apte (10)	0.353	0.304	0.359	**0.631**	0.490	0.416
Apte (100)	0.555	0.421	0.343	**0.881**	0.630	0.609
Apte (1k)	0.723	0.652	0.532	**0.829**	0.754	0.795
Amzn (10)	0.536	0.527	0.516	0.481	0.535*	**0.544**
Amzn (100)	0.614	0.562	0.517	0.480	0.573*	**0.639**
Amzn (1k)	0.717	0.650	0.562	0.483	0.639*	**0.757**
RCV1 (10)	0.505	0.480	0.421	0.450	-	**0.512**
RCV1 (100)	0.683	0.614	0.474	0.422	-	**0.689**
RCV1 (1k)	0.781	0.748	0.535	0.454	-	**0.802**

* Limited to 5 of 10 Amazon categories

Table 7: R-Precision for various text classification experiments, training size in parentheses (Lucas and Downey, 2013).

Table 7 shows the result of our R-Precision experiments in Lucas and Downey (2013). These experiments show that MNB-FM is a poor ranking measure over the ApteMod data set, where it performs in the bottom half of the models for each training set size. On the Amazon and RCV1 data sets, we see that MNB-FM is quite competitive with the state-of-the-art methods, but Logistic Regression is very well suited to ranking in a way that maximizes R-Precision.

Method	1000	5000	$10k$	$50k$	$100k$
MNB-FM	1.44	1.61	1.69	2.47	5.50
NB+EM	2.95	3.43	4.93	10.07	16.90
MNB	1.15	1.260	1.40	2.20	3.61
Labelprop	0.26	4.17	10.62	67.58	-

Table 8: Scalability of competing SSL methods as the number of documents in the unlabeled text corpus grows (Lucas and Downey, 2013). The table displays average run-times measured in seconds.

We also measured run times for varying sizes of the unlabeled data set, which are recorded in Table 8. NB+EM and Label Propagation methods scale poorly due to the number of passes they require over the entire unlabeled data set during training. MNB scales well due to the Naive Bayes assumption and MNB-FM's calculations based on previously measured feature marginals results in only a small run-time increase, but yields a significant increase in F1-Score and R-Precision performance.

Finally, we analyzed the impact that our MNB-FM method has on the feature marginals themselves. In order to determine whether MNB-FM actually improved the MNB estimates, we compared the final conditional probabilities of both classifiers with the true conditional probabilities over the large unlabeled set.

	Fraction Improved vs MNB			Avg Improvement vs MNB			Probability Mass		
Word Freq.	Known	Half Known	Unknown	Known	Half Known	Unknown	Known	Half Known	Unknown
0-10^{-6}	-	0.165	**0.847**	-	-0.805	**0.349**	-	0.02%	7.69%
10^{-6}-10^{-5}	0.200	0.303	**0.674**	**0.229**	-0.539	**0.131**	0.00%	0.54%	14.77%
10^{-5}-10^{-4}	0.322	0.348	**0.592**	-0.597	-0.424	**0.025**	0.74%	10.57%	32.42%
10^{-4}-10^{-3}	**0.533**	**0.564**	0.433	**0.014**	**0.083**	-0.155	7.94%	17.93%	7.39%
$> 10^{-3}$	-	-	-	-	-	-	-	-	-

Table 9: Conditional Probability Improvement of MNB-FM over MNB ($|D_L|$ = 10) (Lucas and Downey, 2013).

	Fraction Improved vs MNB			Avg Improvement vs MNB			Probability Mass		
Word Freq.	Known	Half Known	Unknown	Known	Half Known	Unknown	Known	Half Known	Unknown
0-10^{-6}	**0.567**	0.243	**0.853**	**0.085**	-0.347	**0.143**	0.00%	0.22%	7.49%
10^{-6}-10^{-5}	0.375	0.310	**0.719**	-0.213	-0.260	**0.087**	0.38%	4.43%	10.50%
10^{-5}-10^{-4}	0.493	0.426	**0.672**	-0.071	-0.139	**0.067**	18.68%	20.37%	4.67%
10^{-4}-10^{-3}	**0.728**	**0.669**	-	**0.233**	**0.018**	-	31.70%	1.56%	-
$> 10^{-3}$	-	-	-	-	-	-	-	-	-

Table 10: Conditional Probability Improvement of MNB-FM over MNB ($|D_L|$ = 100) (Lucas and Downey, 2013).

Tables 9 and 10 demonstrate that MNB-FM improves the marginal estimates for a majority of unknown words, as well as the most common known and half-known words, even with only 10 labeled documents. We bucketed words by their frequencies and whether they occurred in both the positive and negative classes ("Known" words), only in one of the two classes ("Half Known"), or only in the unlabeled data set that the Feature Marginals were extracted from ("Unknown"). Both tables display the fraction of improved feature marginals as well as the average absolute improvement applied to each feature marginal, where *improvement* is how much closer a feature marginal is to the ground truth value after the MNB-FM adjustment. For context of the impact of these results, we also include the total probability mass of each bucket. These experiments were conducted over the MCAT category, which comprises 25% of the 801k documents in the RCV1 corpus.

In Table 9, the buckets that showed an improvement over MNB accounted for 80.0% of the total probability mass. This demonstrates that with only 10 labeled documents, we can still calculate accurate marginal probabilities if we leverage a large unlabeled data set. In Table 10, we find that the improved probability mass is reduced to 55.9%.

2.5 Conclusion

In Lucas and Downey (2013), we proposed MNB-FM, a new method for smoothing MNB's conditional probability estimates using simple estimates from a large unlabeled corpus. The evaluations show that the improved conditional probability estimates calculated by MNB-FM not only improved the performance of the traditional MNB classifier, but outperformed state-of-the-art supervised and semi-supervised learning methods on text classification and sentiment analysis. In particular, MNB-FM consistently outperformed the recent SFE model which also used marginal probabilities from unlabeled corpus to directly improve MNB classification performance.

These results were obtained without increasing the computational complexity of the MNB classifier. Instead, MNB-FM relies on counting marginal probabilities over a large unlabeled corpus only once, which can be used later to optimize the conditional probabilities for any MNB classifier. Given the ability to reuse the same marginal probabilities and the inherent simplicity of the MNB classifier’s training calculations, MNB-FM scales extremely well compared to existing SSL methods.

2.6 Multi-Class Extensions of MNB-FM

Our work in Lucas and Downey (2013) was limited to the binary classification setting of the MNB classifier. However, MNB's conditional probabilities are calculated independently for each class and it is trivial to extend MNB to any number of classes (we simply need a set of labeled documents for each considered class). In the multi-class setting, the final prediction of a MNB classifier is the class which has the largest probability according to Equation 9 (page 23).

A multi-class extension of MNB-FM is less straightforward, but not difficult to implement. However, we have not derived a method for calculating the exact conditional probability values that optimize the MLE equation (Equation 13, page 28). In the multi-class setting, optimizing the MLE equation is equivalent to identifying the conditional probabilities that maximize the following generalization of Equation 13:

$$\begin{aligned} &\underset{\theta_w;\forall w\in V}{argmax}\, P(D_L|\theta_w^{c_2}, \theta_w^{c_1}, \ldots, \theta_w^{c_k}) \\ =&\underset{\theta_w;\forall w\in V}{argmax} \prod_{c\in C} (\theta_w^c)^{(N_w^c)}(1-\theta_w^c)^{(N_{\neg w}^c)} \\ =&\underset{\theta_w;\forall w\in V}{argmax} \sum_{c\in C} N_w^c \ln(\theta_w^c) + N_{\neg w}^c \ln(1-\theta_w^c) \end{aligned} \tag{19}$$

In the binary classification setting, we use Equation 11 (page 27) to reduce Equation 13 to one free parameter for which we can solve. In the multi-class setting, Equation 11 generalizes to:

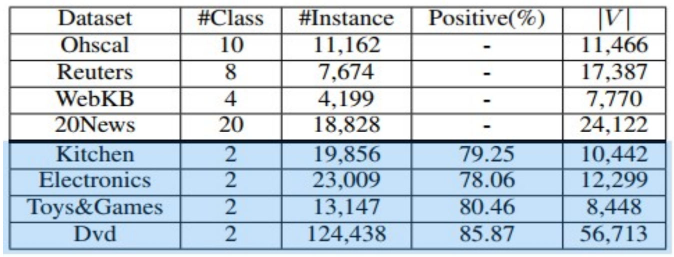
$$P(w) = \sum_{c\in C} \theta_w^C P_t(C) \tag{20}$$

Although under-specified optimization problems do not guarantee a single solution, we expect that the easiest method of solving this system of equations for any number of classes is by implementing an existing optimization method that doesn't require fully-specified systems.

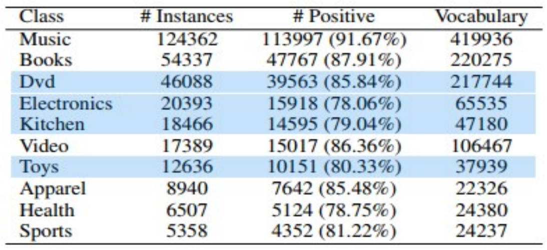
2.7 Analysis of Zhao et al. (2016)

Following our publication of Lucas and Downey (2013), our model was independently implemented for comparison against other semi-supervised text classifiers in Zhao et al. (2016). This publication demonstrated an alternative method for semi-supervised scaling of MNB and directly compared against an independent implementation of the MNB-FM method. The performance of the Zhao et al. (2016) implementation performed much worse than expected given the results in Lucas and Downey (2013). We have thoroughly compared these implementations and found that differences in the preprocessing of text data sets thoroughly explain the results in both publications. The following section will resolve the discrepancies between these papers and demonstrate that MNB-FM's poor performance in Zhao et al. (2016) can be traced to the removal of rare words.

In 2016, another semi-supervised improvement to MNB that leverages feature marginals was published (Zhao et al., 2016). In a direct comparison between the two approaches, Zhao et al. (2016) found that their Multinomial Naive Bayes model with Word-level Statistical Constraints (**MNB-WSC**) frequently outperformed both MNB-FM and SFE (introduced at the end of Section 2.3). Many of Zhao's experiments were performed over the same data sets we used in Lucas and Downey (2013), but the reported results over identical classes showed that MNB-FM performed much worse in Zhao et al. (2016) than in Lucas and Downey (2013). We found it very important to resolve these differences due to the surprising difference in performance between these two implementations of our method.



Dataset	#Class	#Instance	Positive(%)	$\|V\|$
Ohscal	10	11,162	-	11,466
Reuters	8	7,674	-	17,387
WebKB	4	4,199	-	7,770
20News	20	18,828	-	24,122
Kitchen	2	19,856	79.25	10,442
Electronics	2	23,009	78.06	12,299
Toys&Games	2	13,147	80.46	8,448
Dvd	2	124,438	85.87	56,713



Class	# Instances	# Positive	Vocabulary
Music	124362	113997 (91.67%)	419936
Books	54337	47767 (87.91%)	220275
Dvd	46088	39563 (85.84%)	217744
Electronics	20393	15918 (78.06%)	65535
Kitchen	18466	14595 (79.04%)	47180
Video	17389	15017 (86.36%)	106467
Toys	12636	10151 (80.33%)	37939
Apparel	8940	7642 (85.48%)	22326
Health	6507	5124 (78.75%)	24380
Sports	5358	4352 (81.22%)	24237

Table 11: Comparison of the Amazon data set summary between Zhao et al. (2016) (left) and Lucas and Downey (2013) (right).

In Zhao et al. (2016), many of the reported experiments showed that MNB-FM only slightly improved over the traditional MNB classifier. This was unexpected given the calculated significance of the improvements we published in Lucas and Downey (2013). Fortunately, many of the data sets between the two publications overlapped, which made it very easy to re-evaluate our model and determine the reasons for the poor performance. Zhao et al. (2016) published experimental results over 5 different data sets, two of which were RCV1 and Amazon ApteMod. This included 4 of the 10 Amazon classes in Lucas and Downey (2013). Crucially, table 11 displays fundamental disagreements of the data set and vocabulary sizes for a number of document classes used in both publications.

Although both publications relied on the same source data sets for our experimentation, we were able to contact the authors of this publication and obtain the data sets they used to train the various models. Notably, these data sets were were preprocessed and converted to the Weka ARFF format, a format compatible with the Weka machine learning platform (Holmes et al., 1994). An initial inspection showed that the number of documents and vocabulary size matched the values published in Zhao et al. (2016), but not the values in Lucas and Downey (2013).

Next, we found that we could duplicate the results published in Zhao et al. (2016) using the provided ARFF-format data sets with our own implementation of MNB-FM, the same implementation we used in Lucas and Downey (2013). Our implementation verified the MNB-FM performance in Zhao et al. (2016). The two MNB-FM implementations have slightly different results over the ARFF-format data sets, but these slight differences are likely explained by rounding errors or different approaches to solving the MLE equation from MNB-FM (Equation 17, page 29). As a reminder, we used a Scipy (Jones et al., 2001) implementation of Newton's method to solve Equation 17 and are unsure of the specific details of the implementation in Zhao et al. (2016).

At this stage, our original implementation of MNB-FM agreed with the performance of Zhao et al. (2016) over the same input data, but discrepancies in the published results and differences in the data set statistics remained unresolved. We expected that identifying the cause of the data set discrepancies would explain the difference in MNB-FM's performance. Notably, Zhao et al. (2016) explains that the authors "preprocess Multi-Domain Sentiment data set in a similar way as Lucas and Downey (2013), since punctuation could indicate strong sentiment," a reference to our statement that we removed punctuation and stopwords. Both the Amazon Sentiment and Reuters Aptemod data provide lists of stopwords for removal, which makes duplicating this process straightforward. Our inspection of the ARFF files provided in email correspondence from the authors of Zhao et al. (2016) showed that word counts for the rarest words were not included, those words that occurred fewer than 3 times in a given class' documents. We then verified that removing stopwords, punctuation, and words with fewer than 3 occurrences from these data sets yielded counts that matched those reported in Zhao et al. (2016).

In Lucas and Downey (2013), our analysis of MNB-FM included an explanation that many of the performance gains of MNB-FM were due to better predictions of the conditional probabilities for rare words. MNB fails to take into account the fact that very rare words have a non-zero probability of occurring in a small labeled data set and the estimated conditional probabilities for rare words will often be overestimated. Contrarily, MNB-FM considers the global frequency of rare words and will decrease them if the word's frequency within the much larger unlabeled documents is lower than estimated over the labeled documents. Removing words that occur fewer than 3 times from the vocabulary many of the words that MNB-FM utilized to improve the MNB classifier. Our analysis in Lucas and Downey (2013) demonstrated that rare words were one of the largest segments of the vocabulary where our adjusted conditional probability estimates directly contributed to the improved performance of MNB-FM (Table 10, page 37). Words this rare are very hard to model statistically due to the inherent random and noisy nature of sampling events with very low probability. However, accurately modeling conditional probabilities for rare words may not have much *individual* value, but Lucas and Downey (2013) shows that MNB-FM uses these words *in aggregate*. Due to the long-tailed distribution of word frequency, rare words make up a large fraction of a typical text corpus – 23% of RCV1's MCAT vocabulary occurs less than once per 100,000 tokens (Table 10). MNB-FM adjusts the conditional probabilities of these rare words in the same manner as all other words, but the frequency of rare words as as whole allows them to provide a significant impact to the performance of MNB-FM's predictions.

Our follow-up analysis of the Lucas and Downey (2013) and Zhao et al. (2016) and the conflicting results therein provide clear evidence that the different preprocessing techniques lead to poor performance of the MNB-FM model in Zhao et al. (2016). This analysis allowed us to gain further insight into how the mathematical underpinnings that drive MNB-FM lead to methodological and sensible improvements in the conditional probability estimates of the MNB classifier.

Chapter 3
Multi-prototype Neural Network Language Models

The neural network is one of the oldest machine learning models, dating back to perceptron networks (Rosenblatt, 1957) that were inspired by new discoveries of how brain cells send and receive electrical signals. Many subsequent improvements showed promising theoretical implications, but were computationally infeasible to evaluate or even train given limitations of computer hardware (Rowe, 1969). The advent of the backpropagation algorithm for training complex neural networks (Rumelhart, 1985) and recent advances in computing hardware and programming languages have facilitated the efficient training of complex neural networks on large data sets. Significant improvements in neural network performance have been achieved thanks to the modern computational power of GPU's paired with matrix-based neural network software frameworks such as Torch (Collobert et al., 2011), Tensorflow (Abadi et al., 2016), and Keras (Chollet et al., 2015). Simultaneously, these technologies have driven a revival of previous neural network architectures. With modern software and hardware, these historical architectures often achieve state-of-the-art performance in a number of language-related tasks, for example Recurrent Neural Networks (**RNN's**) (Williams and Zipser, 1989) and Long Short Term Memory Neural Networks (**LSTM's**) (Hochreiter and Schmidhuber, 1997) have respectively achieved state-of-the art performance in Language Modeling experiments thanks respectively to modern implementations by Mikolov et al. (2010) and Sundermeyer et al. (2012).

Natural Language Processing (NLP) research in the early 21st century has been heavily driven by the recent realization that neural networks are capable of generating powerful encodings of human vocabularies that can be trained over huge amounts of human-generated text and subsequently applied to many supervised and unsupervised NLP tasks. These neural network approaches are similar to previous methods that simultaneously learned vocabulary and task-specific information from unlabeled corpora. For example, Landauer and Dumais (1997) demonstrated that Latent Semantic Analysis could estimate word similarity while simultaneously discovering and clustering documents by latent topics. Bengio et al. (2003) showed that by representing each word as an initially random high-dimensional vector and using these word vectors as inputs to a neural network, the backpropagation algorithm for training neural networks can update the encoding of these word vectors and the result is a full encoding of a language's vocabulary. Experimental results also showed that many variations of this neural architecture were capable of outperforming n-gram language models in the Language Modeling task (Bengio et al., 2003).

After this breakthrough result, neural network architectures have achieved increasing state-of-the art performance in a variety of linguistic tasks – Question Answering (Liu et al., 2017), Sentiment Analysis (McCann et al., 2017), and Named Entity Recognition (Peters et al., 2017). This capability is due in large part to learning high-dimensional vector encodings of vocabularies using neural networks as demonstrated by Bengio et al. (2013). These encodings are now commonly referred to as *word embeddings.* Word embeddings were initially thought to be useful only as inputs the specific model that created these embeddings while learning to accomplish another NLP task. It was later shown that word embeddings can encode general linguistic information, such as in the surprising analysis of the word2vec model proposed by Mikolov et al. (2013a). Mikolov et al. (2013b) later showed that neural networks designed for the *language modeling* task (essentially predicting which word that was removed from a given sub-sentence) were capable of encoding complex relational information in their word embeddings. Fully training a word2vec model produces a word embedding that can be used independent of the word2vec model to complete SAT-style analogies such as “man:woman::king:____”.

Most modern state-of-the-art word embeddings are generated under the assumption that each word has a single meaning, though we know this is not the case in human languages such as English. For example, consider the multiple definitions of the *polysemous* word “pound,” which is frequently used as a British currency, a synonym for “hit” (pounding a table), a unit of weight, and a building for containing stray animals. Traditionally, neural networks are not taught to identify or disambiguate a word’s disparate meanings and instead represent each word as a single vector in the word embedding. Our analysis shows that this conflation warps the word embeddings related to polysemous words and has a downstream effect of reducing overall language modeling performance. Using the above example, two meanings of the word pound – the British currency, a noun, and the synonym for “hit”, a verb – are distinct not only in their meanings, but in their contexts. Due to the grammatical differences in the way nouns and verbs are used, these two meanings will have different positions within English sentence structure, but we also expect the meanings to be used in very different contexts – one when speaking about currency or transactions, the other when applying force to an object (perhaps with another object). As such, any neural network that trains only one vector per word must fold information about these distinct meanings of “pound” into a single representation.

Our work provides thorough evidence that two problems arise when polysemous words are represented with a single vector. First, there may not be a sensible location for a polysemous word’s vector that will suit the needs of a neural network for input for every meaning of the word. Neural networks that achieve state-of-the-art performance on NLP tasks rely on the fact that linguistically similar words are co-located within the high dimensional vector space of a word embedding. This is intuitively useful – if synonyms such as “bucket” and “pail” can be used interchangeably, learning an embedding where their vector representations are as similar as possible allows the neural network to interpret these words interchangeably as well. We also find that antonyms are often co-located when word embeddings are trained over the Language Modeling task (discussed in detail in Section 3.1) due to the fact that words are used in similar contexts as their antonyms (e.g. “The weather is a lot hotter/colder than it was yesterday.”). It then stands to reason that “pound” could be co-located with currencies such as “dollar” and “yen” or with verbs such as “hammer” and “hit” in a word embedding. However, we don’t expect that both sets of words should be co-located because words like “hammer” and “yen” don’t share contexts or meanings, they are simply linked through the word “pound.” It is then unclear where “pound” should be located, but our analysis shows that polysemous words that have one dominant meaning are typically located with words related to that meaning. For example, we find that neural networks learn representations for the word “Chicago” close to American cities, but not next to Broadway musicals or classic rock bands.

The second notable issue can be illustrated with our previous example of the word “pound” above. We find that when two meanings of a word are equally common, word embeddings will frequently represent the polysemous word’s vector in the space between the two regions with words related to these two meanings. When the distance between these two regions is large (as we find it to be between currencies and synonyms for “hit”), the word is then interpreted as unlike both sets of related words. We similarly find that the sets of words related to these two meanings are also negatively affected – the unrelated words actually grow closer together due to the difficulty of placing just one word.

The fact that neural networks should only learn one representation per word is due to the most common method for training word embeddings. Most word embeddings are created by training a neural network on the task of *Language Modeling,* which is the task of scanning over huge amounts of text and learning to predict which word would most likely fill the blank in a given context (e.g. “I ate _____ for breakfast today”). This is a generalization of the Language Modeling task, but variations of Language Modeling and their specific objectives are discussed in Section 3.1. Language Modeling is a very complicated challenge for machine learning algorithms – high performance requires a deep understanding of a language’s vocabulary and a comprehensive range of contexts for most words. Further, the task itself is inherently impossible to perform perfectly in English – if only one word can fill any given blank, language would be incredibly rigid and less expressive.

Recent research has shown that some neural network architectures are very capable of high performance on language modeling tasks. Word embeddings are learned over incredibly large text corpora billions of words in length, making it prohibitively expensive to label every token of a polysemous word with its corresponding meaning throughout a training corpus. Nevertheless, modern neural network language models perform well under the assumption that all constituent meanings of a given word can be encoded in a traditional word embedding. Crossley (2010) reasons that polysemy is a natural trait of human languages due to the *law of least effort:* “speakers will economize their vocabulary by extending word senses in order to conserve lexical storage space... Because frequent words have the most senses, learners encounter highly polysemous words most often.” The failure of modern word embeddings to learn multiple meanings per word implies that the disparate natural locations of a polysemous word’s many synonyms must be compressed into a single representation and we theorize that this undermines the semantic structure of trained word embeddings.

In this chapter, we will demonstrate the design and performance of our proposed Multi-Prototype Neural Network Language Model (MPNNLM). This model was designed to alleviate a number of limitations of previous single-prototype and multi-prototype language models. The MPNNLM was designed with the following motivations:

- Model multiple word embedding vectors for polysemous words.
- Use high-error predictions to identify when a word has an unmodeled meaning that requires a new vector representation.
- Disambiguate words in any unlabeled unlabeled text without relying on an external clustering or topic modeling approach.

The following sections will provide a comprehensive background of the language modeling task and neural network models designed for generating single and multi-prototype word embeddings. In Sections 3.5 and 3.6, we will explain the design of our multi-prototype neural network language model and demonstrate its performance in a number of linguistic tasks.

3.1 Language Modeling

At a general level, a language model is any machine learning model that algorithmically learns from human writing to accurately estimate the probability that a human would write or say a given sequence of words. Calculating the likelihood of a single phrase is not itself very useful, but *comparing* the likelihood of similar phrases is the underpinning of a number of natural language processing systems. For example, modern computers are capable of transcribing human speech to text because accurate language models understand that “It was great to meet you” is more likely to be written by a human than “It was grate to meat you.”

N-gram Language Models. A variety of language model designs have been proposed over the years and, for the most part, these models have a fixed ‘window size’ that defines the number of input words. Limiting the size of the input alleviates a number of empirical concerns, including how a model could store every known sentence and its likelihood of occurring. When longer sequences of words are broken into chunks of n sequential words, it is called *an n-gram* and the any language model that limits its input to n-grams is called an *n-gram language model*. In the first example above, a 4-gram model would compute individual probabilities for the “It was great to”, “was great to meet”, and “great to meet you,” combine the estimates into a single probability for the entire phrase and finally compare this probability to a variety of rhyming sentences’ probabilities to determine which transcription is most likely.

Early logic-based and rule-based models could not scale to large text data sets, but early n-gram models could efficiently extract statistics over and achieve state-of-the-art results in language modeling (Jelinek and Mercer, 1980) and variations of the n-gram language model represented the state of the art in language modeling from the early 80s to the late 90s (Goodman, 2001). The most straightforward n-gram language model is a lookup table that stores the number of occurrences of each n-gram that appears in the training text. This makes individual n-gram frequencies easy to compute.

Over time, text corpora grew in size and the combination of increasing computational efficiency and decreased data storage costs lead to improved n-gram language models. 3-gram (trigram) models were shown to outperform 2-gram (bigram) models, but it was often the case that certain word triplets were too rare to get accurate frequency estimates. Language models rely on the assumption that language is randomly generated based upon preceding contextual words. In such a setting, very rare words are difficult to accurately model because they have high variance compared to their expected probabilities. As such, a number of *back-off n-gram models* were designed, which smoothed language modeling probabilities with bigrams where trigram probabilities were sparse (Jelinek and Mercer, 1980; Katz 1987). Since then, data sets large enough to yield accurate 5-gram probabilities have been released (Brants and Franz, 2006) and back-off models continue to perform well (Chelba et al., 2013).

The storage required for n-gram lookup tables is theoretically exponential in the value of n, but in practice many n-gram data sets remove any sequence of words that occurs less than a specific number of times over the entire corpus. The Brants and Franz (2006) data set compresses a 1 trillion word corpus into a 5-gram lookup table with 13.6 million unigrams, 315 million bigrams, 977 million trigrams, 1.31 billion 4-grams, and 1.18 billion 5-grams. In this data set, grams of any size that occurred less than 3 times were removed from the final lookup table and the initial release was still 24 gigabytes when compressed. Generating and storing accurate n-gram language models with large values of n remain prohibitively expensive, so NLP research is often focused on more clever approaches that focus on more efficient parameter encodings.

N-gram models are often used to chain conditional probabilities for a given sequence of words to predict the next word in a sequence. For example, a bigram model can be used to estimate the probability that “England” follows the sequence “The queen of …” by computing:

$$\begin{aligned} &P(\text{The}, \text{queen}, \text{of}, \text{England}) \\ &\qquad \propto P(\text{The} | \text{<s>}) P(\text{queen} | \text{The}) P(\text{of} | \text{queen}) P(\text{England} | \text{of}) \end{aligned} \tag{21}$$

In this case, **P(The | <s>)** is the likelihood that “The” begins a sentence. We use the meta word **<s>** to represent a “start-sentence” token, which allows us to estimate the probability that sentences begin with the word “The”. As discussed previously, n-gram approaches requires a large text corpus to accurately estimate conditional probabilities for common and especially rare sequences of words. Then, we can estimate the true probability that “England” would occur next, but this requires calculating Equation 21 for every word in the vocabulary in place of “England.” With accurate bigram probabilities, we expect words like “France,” “Sheba,” and “hearts” to receive higher probabilities than words unrelated to queens.

Purpose. Language modeling is very important for a variety of natural language processing tasks where the context provided by previous words in a sequence can improve accuracy on subsequent predictions. The most competitive speech recognition methods use two models: an underlying language model, which attempts to reduce the search space over a given set of words, and an acoustic model, which parses an audio recording into a sequence of *phonemes* – sounds that combine to form words (e.g. "tough" is composed of the phonemes **T-UH-F**). For example, when a speech recognizer reaches the word "park" in an audio recording of the sentence "He is reading in the park," an acoustic model may output **P-AH-R-K**. If the acoustic model has trouble determining whether the final word is "park" or "pork," it can rely on a language model, which can combine the previously predicted words with its learned model of the human language over many previous mentions of "reading in the" to inform a more accurate prediction.

In general, language models perform well when they are trained on large amounts of human-generated text. Early machine learning approaches for language were trained on comparatively small corpora, frequently in the form of digitized newspaper articles. This raises a common concern in training machine learning models on text – a domain-specific corpus will likely bias the model for better or good. Speech recognition algorithms generally rely on an underlying language model and a newspaper trained language model may perform very well for a news reporter transcribing notes, but very poorly for a doctor.

Fortunately, modern technology provides access to massive amounts of human-generated text in a variety of formats, qualities, and topics. Provided access to enough computation and storage, n-gram lookup tables generated from large text corpora are capable of performing well, even on domain-specific tasks (Chelba et al., 2013). For larger values of **n**, the increased context window allows the n-gram model to capture consistencies in language in a way that surreptitiously fragments domain-specific language.

Perplexity. The *perplexity* metric is frequently used to determine how different language models compare in overall modeling of a given corpus. Language modeling requires a model to "read" a sequence of words in text "**w(t-i)** … **w(t-1) w(t)**" as context and output a probability distribution over the entire vocabulary for the next word – a vector containing a probability estimate for every word in the vocabulary:

$$\vec{P}(t)=\langle P(w_1|\text{context}), P(w_2|\text{context}), \ldots, P(w_{|V|}|\text{context})\rangle \tag{22}$$

A perfect model would assign 100% of the probability mass to word **w(t+1)** at every time **t**, which is essentially impossible given the stylistic variety of human-generated text. Even given the first 99% of Shakespeare's Othello, it seems unlikely that any computational or human method will ever reliably generate the remaining 1% of the script. As such, accuracy is a highly restrictive metric that ignores the unpredictable nature of human language.

Perplexity is effectively a variation of accuracy – it rewards models for assigning high probabilities to the correct next word in a sequence, but ignores the predicted probabilities of all other words. For any given text sequence, **s**, and a language model's prediction over the entire vocabulary for every word in **s**, $P\in\mathbb{R}^{|d|}\times\mathbb{R}^{|V|}$, perplexity is defined as:

$$\mathit{Perplexity}=2^{-L} \quad \mathit{where} \quad L=\frac{1}{|s|}\sum_{t=0}^{|s|} P_t(s_t)\log_2(P_t(s_t)) \tag{23}$$

L is the *log-likelihood*, but perplexity is more generally interpretable – a perplexity of 10 means that on average, the language model assigned the correct word a probability of 1/10 = 0.1. For clarity, **P$_t$(s$_t$)** is the probability the language model assigned to the correct value of **w(t)** at time **t**. High values of log-likelihood, and in turn low Perplexity, indicate that the model has high predictive ability over the sequence in **s**.

Word Error Rate. A popular alternative to perplexity is *word error rate* (WER), which is suited to fields such as speech recognition. If a language model predicted "He reads in the pork" in the previous speech recognition example, the subsequent time step would assume the prediction was correct and generate a subsequent prediction over an incorrect input sequence, which can cause errors to propagate quickly. Language models that perform well on WER return to the correct text quickly after an incorrect prediction.

WER measures is specifically designed to measure the relatedness between the true text and the predicted text, which can differ in content and length. Using hand-labeled test examples (in Speech Recognition, this would be audio transcriptions), WER is evaluated as:

$$WER = \frac{S + D + I}{N} \tag{24}$$

The numerator totals the number of errors made by the language model: **S** is the number of substitutions, **D** is the number of deletions, and **I** is the number of insertions. The denominator, **N**, is the total number of words in the test corpus text.

Perplexity and WER provide a general idea of a model’s performance for most language model use cases. Perplexity measures whether a language model is representative of given text sequence and WER evaluates performance when language model doesn't know if its input sequence is grammatical or sensible, but is especially useful when a language model is used in generative settings. Language Modeling papers tend to show both metrics, giving both perspectives to the reader.

Recently, a simple n-gram-inspired neural network was shown to outperform the best 5-Gram models, decreasing perplexity by 24% (Bengio et al., 2003). Our proposed language model continues in the direction of neural network language modeling, which can be seen in Section 3.4. Section 3.2 will provide the background necessary to interpret the design and results of the present state-of-the-art neural network language models which are described in detail in Section 3.3.

3.2 Overview of Neural Networks

Here we provide a technical background of the various neural network concepts and models required to understand many of the concepts in subsequent sections. We will explain the important aspects of neural network design, training/backpropagation, and evaluation, but we limit the scope of this section to the topics necessary for explaining the multiple-prototype neural network language model we propose in Section 3.5. For a more comprehensive introduction to neural networks, Bishop (1995) is an excellent resource.

Perceptrons. Artificial neural networks are machine learning models that have a long history, dating back to early attempts to computationally model the brain's cellular structure. The first successful models were based upon the concept of a *perceptron*, a simplified computer implementation of the cellular neuron. Modern neural networks are capable of multi-class classification and regression, perceptrons are limited to binary output.

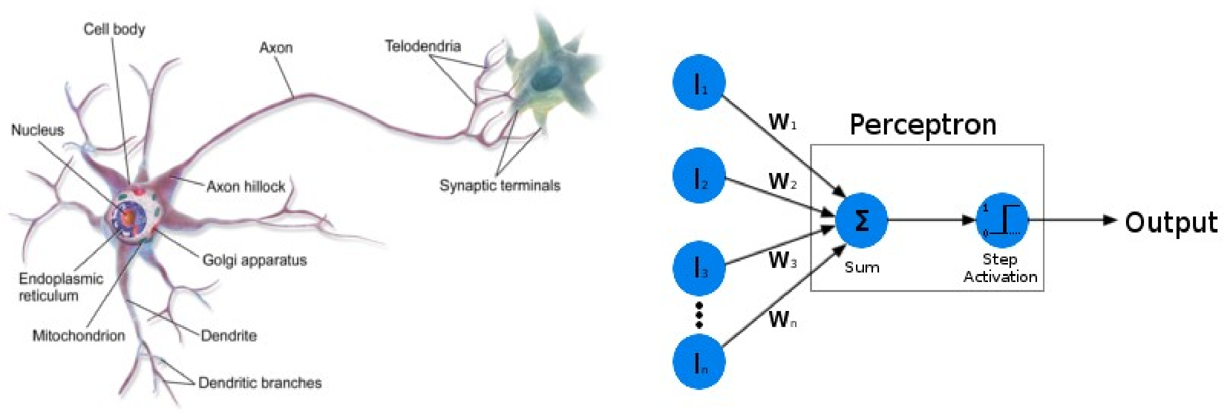


Figure 1. Left: A biological neuron (BruceBlaus, 2017).
Right: An illustration of a single neuron-inspired perceptron with n inputs.

Inside of the brain, specialized cells called neurons provide the basis for anatomical computation. Due to the brain's heavily interconnected structure, its computations are massively parallel.

Computationally simulating a neural network of the scale of the human brain remains infeasible, but a number of simplifications boosted perceptron network research to the forefront of artificial intelligence research.

Each neuron receives chemical signals from its neighbors through its dendrites. The neuron has a complicated chemical logic that determines which chemical signals it should emit to neighboring neurons through its axon terminals. Similar to the brain's neuronal structure, each perceptron may be connected to a number of other perceptrons and can aggregate signals from other perceptrons through its *input connections.* A perceptron may also emit signals to other perceptrons through its *output connections*. The perceptron design retains a simplified version of this concept of neuronal firing: a single perceptron will weight each of its input connections and when the total weighted signal is above a given threshold, the perceptron will itself fire and signal to its output connections. This firing was calculated with a very simple model:

$$output = \begin{Bmatrix} 1\ if\ w \cdot x + b > 0 \\ 0\ otherwise \end{Bmatrix} \tag{25}$$

Feed-forward Networks. The human brain provided initial motivation for the perceptron network, but the brain's highly connected nature was computationally infeasible to model. Many of the successful perceptron networks of the 1960's were limited to *Feed-forward* designs, where the *input layer* of perceptrons don't receive signals from other perceptrons, but instead directly receive data as input from an external data set and propagate their outputs to the next layer in the perceptron network. In feed-forward networks, this process is repeated in a layer-by-layer fashion: each layer receives input from the perceptrons in the previous layer, computes the activation function for each of its perceptrons, and its activated outputs are then passed to the connected perceptrons in the next layer. The final output layer yields the network's overall prediction of the objective function.

Feed-forward perceptron networks are binary classifiers with one or more outputs. For any given binary function **f** over a real feature space of **d** dimensions, a perceptron classifier attempts to approximate $f:\mathbb{R}^d \rightarrow \{0,1\}^{|o|}$. A training set can be used to verify the accuracy of the perceptron network, but learning the optimal structure and weights of the connections was a complicated process.

Unfortunately, programmatic methods for training complex perceptron networks had not been designed. The best approaches for training complex perceptron networks, hand-tuning and heuristic-informed brute force, were inefficient and unreliable and ultimately limited the complexity of perceptron networks. Rather than create multi-layer networks, many approaches found success in one- or two-layer perceptron networks, but the difficulty of identifying optimal weights for deeper perceptron networks limited their classification abilities (Minsky an Papert, 1969). If perceptrons were expected to generate powerful models, they were expected to solve atomic logic problems in a simple manner in order to scale. Further obstacles to advancing perceptron network research arose when Minsky an Papert (1969) demonstrated that single-layer perceptron networks were fundamentally limited in their classification ability, including generating a simple XOR classifier from a single perceptron.

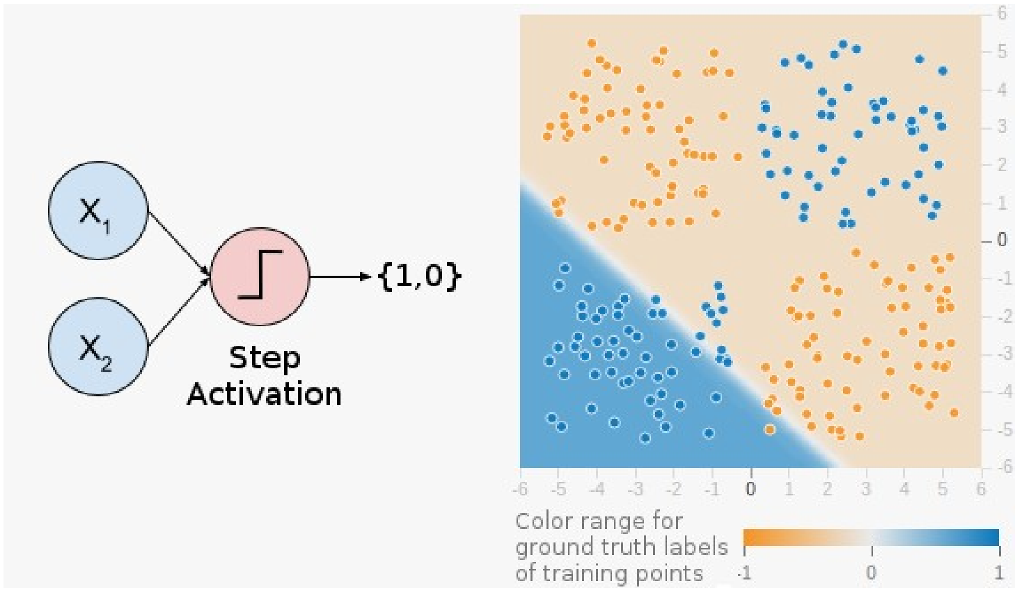


Figure 2. Left: A single-layer perceptron can only solve linearly-separable classification problems. Right: An example of the perceptron's limited ability to create an XOR classifier (http://playground.tensorflow.org).

A three-layer solution to the XOR-problem exists, but Minsky an Papert (1969) provided a thorough exposition of the limitations of perceptron networks. If a complex classification problem required a number of chained XOR layers, it would be nearly intractable to identify the correct weights for the perceptron network without obviating the need for neural networks altogether. If an oracle is necessary to identify the structure and weights for a perceptron network to best approximate a target function with heavy XOR logic, using the same oracle to build an XOR network would be much more sensible.

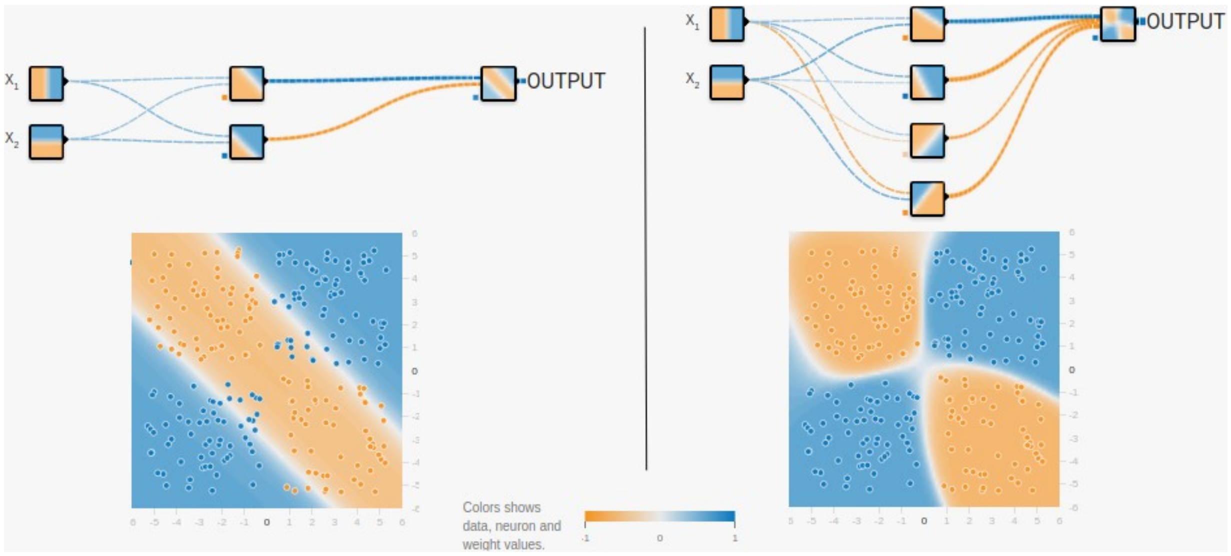


Figure 3. According to the Universal Approximation Theorem, any function can be approximated with a large enough hidden layer. Illustrations generated by (http://playground.tensorflow.org) demonstrate how two neural networks with different hidden layer dimensions may generate different approximations of the XOR function.

Two decades after Minsky an Papert (1969), the advent of the *backpropagation algorithm* provided an efficient method for training neural networks (Rumelhart, 1985), again fueling interest in the area of neural network research. Cybenko (1989) proved the *Universal Approximation Theorem*, demonstrating that neural networks with a single hidden layer that has a large enough number of nodes could approximate any given function of any number of inputs and outputs variables and was not limited to simple such as the XOR problem. In Figure 3, one can see that a 2-node hidden layer is capable of solving the XOR problem with inputs $x \in \{0,1\}^2$ and that doubling the hidden layer allows for $x \in [0,1]^2$ though we can see that the second neural network in Figure 3 is myopically overfit to match XOR very close to the origin, but its predictions do not generalize as the values move further from the origin. In many cases, a neural network can achieve a similar or better approximation by increasing the number of hidden layers and decreasing the number of nodes per hidden layer. Although neural networks with multiple hidden layers or many nodes per hidden layer may be theoretically capable of approximating any function, identifying optimal weights of a complex neural network by hand or through brute force is becomes incredibly difficult as the complexity of the neural network increases. The backpropagation algorithm is an instrumental method for efficiently determining correct weights for feed-forward neural network structures and facilitated the training of the deep neural networks we see in research today.

Backpropagation. Many of the perceptron network limitations described by Minsky an Papert (1969) are caused by the difficulty of communicating to the network how to identify the optimal weights for multi-layer or heavily connected perceptron networks. These limitations were later dispatched with the introduction of *backpropagation*, an algorithmic method for improving the weights in *neural network* models without requiring human hand-tuning or brute force methods (Rumelhart, 1985). The backpropagation algorithm proposed by Rumelhart (1985) required the softening of the activation function such that it is continuously differentiable (sigmoidal functions such as the logistic function or tanh are common choices). Backpropagation also allows each neuron to improve its input weights based only on local information gathered through its input and output connections. The modern concept of a neural network is simply a perceptron network that uses a continuously differentiable activation function in place of the traditional step function and a continuously differentiable *loss function* to judge the error of each prediction by the network. The continuously differentiable requirement was the critical change that enabled the training of neural networks to learn using gradient descent. Training a supervised neural network with backpropagation follows an iterative process over a set of input examples with corresponding expected outputs. In this loop, we assume that the inputs and outputs are numeric, but we later discuss how text can be converted to a representative numeric form:

1. One training example is given as the input to the neural network and propagated forward through the network to generate output values in the same feed-forward manner as in a perceptron network;

2. If the model's prediction is different from the corresponding training label, a human-specified *loss function* is used to estimate the *loss* (or error) of the network, which we then use in the following gradient descent step;
3. A gradient descent method is applied to slowly decrease the network loss over many training examples by updating the network weights in reverse order (called backpropagation) – from the output layer to the input layer.

The forward propagation step of neural networks differs from perceptron networks only in the requirement that the activation function in each node is continuously differentiable, an important requirement for gradient descent (step 3). One effect of using sigmoidal activation functions is that the output of each node is generally limited to the ranges [-1,1] or [0,1], though a threshold can be used to convert the output to a binary prediction (Cybenko, 1989).

One large benefit of backpropagation is that it is an automated method for using a labeled data set to train a neural network. When training the neural network, a single input vector $x_i \in \mathbb{R}^d$ is used in each iteration in order to improve the model's ability to predict its corresponding label $y_i \in \{0,1\}^{|o|}$. The forward propagation step of the model concludes when the neural network outputs **p_i**, the model's current prediction of **x_i**'s label.

Once the neural network generates a prediction, the prediction loss is calculated using a loss function. The loss function, $L(y_i, p_i)$, is specified before training the neural network begins and converts the error between the prediction $\mathbf{p_i}$ and the expected output $\mathbf{y_i}$ to a real value. The backpropagation algorithm is designed to decrease the loss of the neural network during training. The loss function must align with the purpose of the neural network – minimizing loss should maximize the performance of the neural network. Similar to the activation function, backpropagation requires that the loss function is partially differentiable with respect to the neural network output, $\mathbf{p_i}$.

Given any neural network constructed with continuously partially differentiable loss and activation functions, backpropagation can be used to train the network over a labeled data set. At a given time step **t**, the training instance $\mathbf{x_t}$ is fed as input to the neural network, which generates a prediction, $\mathbf{p_t}$, of the corresponding training label $\mathbf{y_t}$ in a typical feed-forward fashion. Once the loss is calculated, the weights connecting the output layer to the final hidden layer are updated:

$$\delta_{ij} = \frac{\partial L(y_t, p_t)}{\partial w^{ij}} = \frac{\partial L}{\partial o_t^j} \frac{\partial o_t^j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \tag{26}$$

After every weight feeding into the output layer is updated, the backpropagation is sequentially used to update input weights for each hidden layer in reverse order, from the output layer toward the input layer. A full explanation and example of the backpropagation algorithm can be found in Rumelhart (1985).

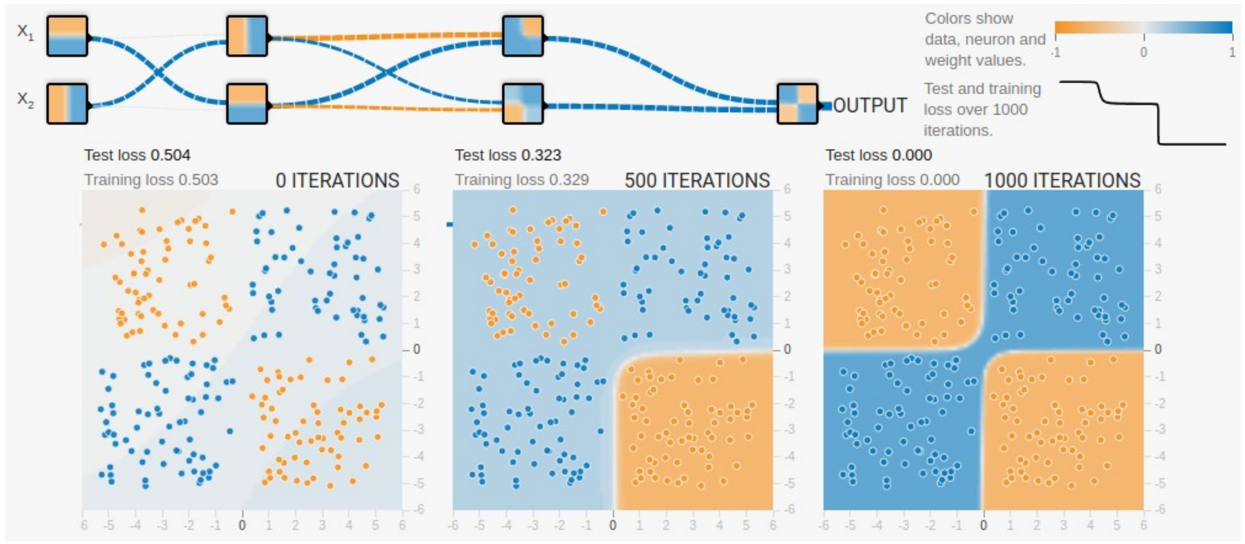


Figure 4. Illustrations from (http://playground.tensorflow.org) demonstrate a neural network with 2 hidden layers solving the XOR problem over 1000 training iterations of backpropagation.

Backpropagation is a gradient descent algorithm, so it requires a large number of incremental updates in order to converge to a stable setting of the weights within a neural network. In practice, this frequently involves iterating over the training set multiple times. Figure 4 illustrates the output of a neural network solving the XOR-problem. The total loss calculated over the test and training set is plotted in the top right and shows two major loss drop-offs. The first major improvement occurs around 250 iterations, when the model is capable of fully distinguishing the negative (orange) points, visible in the middle snapshot taken at 500 iterations. The second major drop in total loss occurs at approximately 600 iterations, when the neural network begins distinguishing the upper left negative region from the remaining positive (blue) points. This plateauing behavior is typical of the backpropagation algorithm – if the gradient is small, it may take many iterations before the neural network may make progress.

3.3 Neural Network Language Modeling Methods

The first state-of-the-art performance achieved in language modeling achieved by a neural network model was demonstrated in Bengio et al. (2003). The approach outperformed smoothed n-gram language models, which were the previous state-of-the-art, but a more exciting result of the neural network was unknown at the time: the neural network was designed to learn high-dimensional real-valued *feature vectors* for every word in the vocabulary and these vector representations would prove extremely useful as inputs to machine learning for other natural language processing tasks. For the remainder of this chapter, we will refer to these feature vectors as *word vectors*, which is the more common terminology today. These terms are synonymous, but there may be confusion when referencing earlier citations. Similarly, when word vectors are learned by neural network language models, it is standard to encode the entire vocabulary into the same high-dimensional space, which is commonly referred to as a *word embedding*.

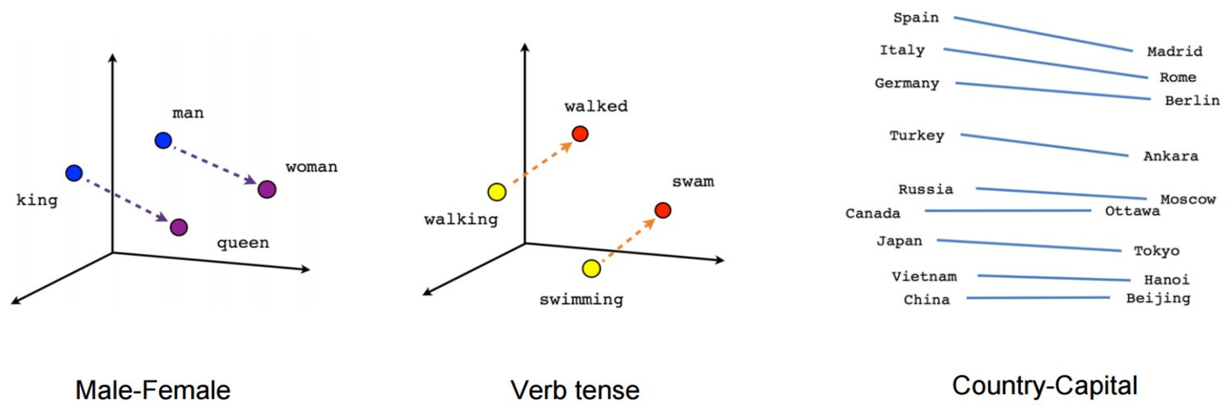


Figure 5. Word embeddings are capable of encoding deeper linguistic information such as concept relationships. Seen here are 2- and 3-dimensional projections of high-dimensional word vectors that demonstrates analogies encoded in the word embedding. (http://www.tensorflow.org/tutorials/word2vec)

Fully-trained word embeddings have proven surprisingly useful for downstream tasks. Mikolov et al. (2013b) demonstrated that some word embeddings can encode relational information that may allow one to complete analogies such as "**man:woman::king:__?__**". Machine translation can be improved by embedding English and Chinese words in the same high-dimensional vector space (Zou et al., 2013) or by training a dual-language neural network language model (English and French) to simultaneously model both languages and generate related word embeddings (Bahdanau et al., 2014).

This section will cover various models that represent the state-of-the-art of the nascent field of neural network language modeling and that provide important context and background for our neural network language model. Notably, these models assume that each word should have a single meaning, though the following section provides a broad overview of existing models that can learn multiple encodings per word.

Feed-forward Language Models. Bengio et al. (2003) proposed the first Neural Network Language Model (NNLM) that was capable of outperforming state-of-the-art n-gram language models. Bengio's proposed neural network is itself an n-gram model, as inspired by state-of-the-art smoothed n-gram language models of the era – the neural network takes **n** words as input ($\mathbf{w}_{t-n}$, …, $\mathbf{w}_{t-1}$) and makes a prediction of the next word in the sequence, $\mathbf{w}_t$.

By design, neural networks require real numbers as inputs, so the input words must be encoded as a real-valued vector. For each of the **n** inputs, Bengio et al. (2003) used a *one-hot* encoding of the entire vocabulary – in a vocabulary of size V, each input word was represented as a V-dimensional vector entirely composed of zeros except for a single 1 one at the index corresponding to the given input word.

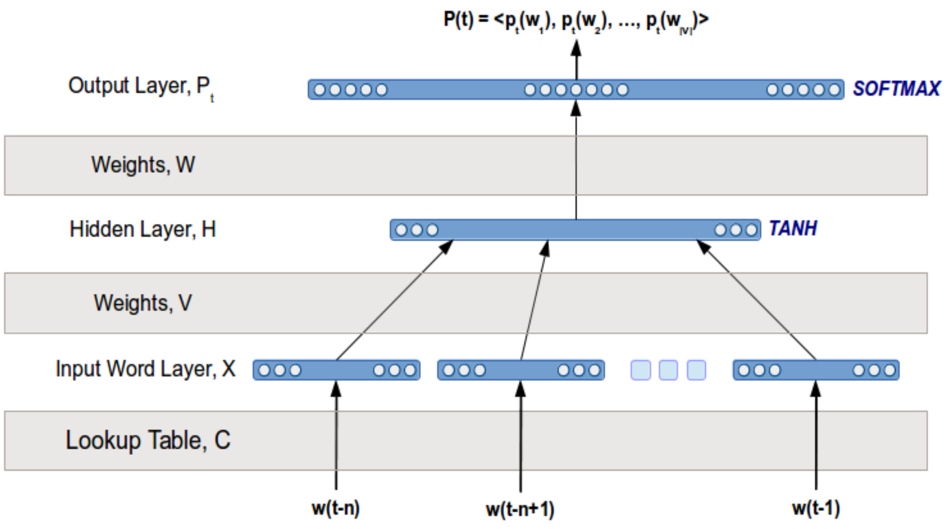


Figure 6. An n-gram-inspired neural network language model (Bengio et al., 2003)

Figure 6 captures the design of the neural network language model proposed by Bengio et al. (2003). Given a the input word sequence, a prediction of the next word begins with the input layer. The model finds the word representation for each of the **n-1** context words in *Lookup Table* **C**, a matrix that stores **d**-dimensional word representations for every word in the vocabulary. In Bengio et al. (2003), the hidden layer outputs are calculated using tanh, a sigmoidal activation function:

$$h_i = \tanh\left(\sum_{x_j \in input\, layer} V_{ij} x_j \right) \tag{27}$$

After the values for the hidden layer nodes are calculated, the output layer calculates a probability distribution over the entire vocabulary predicting which word is likely to occur next in the sequence. The output layer has a node for every word in the vocabulary, each signaling the likelihood of its corresponding word occurring next. The value for each output node, **o(w$_i$)**, is calculated using a weighted sum of the hidden layer values and a *softmax* function normalizes the outputs into a valid probability distribution summing to 1:

$$softmax(o(w_i)) = \frac{e^{o(w_i)}}{\sum_{j=1}^{|V|} e^{o(w_j)}} \quad where \quad o(w_i) = \sum_{h_j \in H} W_{ij} h_j \tag{28}$$

When training the neural network, the backpropagation step improves both the overall model's predictive ability and the vector representations of the input words. Given a prediction of **w(t)**, backpropagation adjusts the weights within the neural network to increase the likelihood of predicting the **w(t)** in future similar contexts. Updates of the network weights propagate to the input layer and input word vectors are similarly updated according to the standard gradient descent algorithm for backpropagation. As the model is trained over a text corpus, the word vectors in the lookup table are constantly updated, eventually generating a well-structured word embedding.

The real strength of the neural network in Bengio et al. (2003) is in the nature of this simultaneous update. In a training process over a large vocabulary of human-generated text, each word vector is modified by backpropagation over hundreds or thousands of training iterations. The ideal destination for each word is uncertain, but the most intuitive result of this training process is that synonyms are frequently very close to each other – if an input word is replaced with a synonym, we expect the output prediction to be very similar.

Initial experiments demonstrated that this NNLM outperformed the best comparable n-gram language model by 24%. The number of parameters in this model scales in **$\Theta(|V|)$** whereas traditional n-gram models scale in **$O(|V|^N)$** (Bengio et al., 2003), though the backpropagation algorithm, which is calculated per word, is computationally expensive and often requires multiple passes over the training corpus.

Recurrent Neural Network Language Models. N-gram neural network language models limit the input context window to a fixed number of words and it has been shown that short contexts provide insufficient information to complete sentences, even for humans. Owens (1997) polled 8 human subjects over 96 sentence-completion tasks. Given a 10-word string with one word randomly removed from the middle, participants were asked to rank the 3 words most likely to fill the gap. Humans were capable of guessing the correct word with their first choice 26% of the time, which was statistically significantly better than the 21% achieved by an n-gram approach. Of note is that humans very strongly outperformed the traditional n-gram methods in predicting the part-of-speech of the missing words. When humans guessed incorrectly, their top choices had the same part-of-speech as the missing word in 70% of their predictions, while the n-gram model was only capable of 52% accuracy (Owens, 1997). By design, n-gram models make predictions based on the limited window of input words and as a result lack other contextual clues such as sentence structure, writing style, or any understanding of mentioned concepts.

Machine learning models can achieve super-human performance at difficult challenges such as playing chess, but some challenges remain *AI-Complete* – the category of machine learning problems that are currently believed to require human-level intelligence to perform. Many language tasks are AI-Complete, such as translation, conversation, and question answering, though it is uncertain whether achieving human-level language modeling performance is as difficult and some modern approaches demonstrate that these may all be equally achievable by neural network language models (Radford et al., 2019). Given that humans have a difficult time accurately completing the language modeling task, one must consider that human performance may provide an artificial ceiling to the performance of machine learning models, but human performance has historically provided insights into how we may improve language modeling algorithms. As with human predictions, the performance of both traditional n-gram models and n-gram NNLM's improve as the input window increases (Bengio et al., 2003). Unfortunately, traditional n-gram models require exponentially large training texts to accommodate larger input windows. Given that humans performed poorly with only 9 context words, it was considered unlikely that an n-gram and neural network language models could perform well given a context window limited to only 5-10 words. A competing NNLM was later proposed by Mikolov et al. (2010) that added an internal memory loop which gives the model statefulness – the model retains information between words as it scans over the training text – and provides a significant increase in results over Bengio et al. (2003). The structure of the RNNLM limited input to a single word, but added a *recurrent connection* to the hidden layer, which effectively adds temporal memory or persistent state that allows it to outperform all previous n-gram models of any context window width.

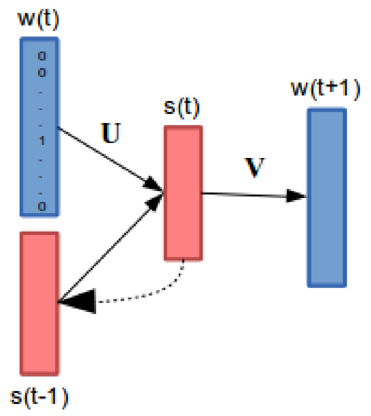


Figure 7. The recurrent neural network language model proposed by Mikolov et al. (2010).

The RNNLM operates very similarly to the standard feed-forward NNLM. At time **t**, the current word, **w(t)**, is given as one of the two inputs to the model. **w(t)** is a vector with length equal to the number of unique words in vocabulary and is composed of zeroes for every word except for the word that occurs at time **t**, which has a value of 1. This is commonly referred to as a *one-hot vector*. Because any input of 0 has no effect on the inputs to **s(t)**, it is expected that we don't actually represent the input as a one-hot vector and calculate O(**|V|**) multiplications that result in 0. Instead, a separate lookup table stores the weights of **U** and at time **t**, we pass the $\mathbf{U}_{\mathbf{w(t)}}$ directly as input to **s(t)**. The hidden layer contains the *state* nodes, **s(t)**, which simultaneously generate the prediction for the next word and encode the current state of the text as it is processed. The width of **s(t)** is manually specified before training and research has shown that 300- and 500-dimension state vectors are sufficient to compete against previous state-of-the-art language models, but increasing the number of nodes as high as 1600 continues to improve performance at the cost of increasing training time and storage requirements (Mikolov et al., 2013a).

The second input vector represents the *previous state* and is typically represented as a *recurrent connection* from the hidden layer at time **t-1**. At time **t**, the computed states of **s(t)** from the previous time period, **t-1**, are given as input to the neural network. This recurrent connection allows the RNNLM to retain contextual information over a theoretically infinite sequence of words, which in turn improves the model's prediction accuracy (Mikolov et al., 2010). Training a RNNLM does not limit backpropagation to the current time step. One of the necessary parameters in training a RNNLM is the number of time periods that should be cached for backpropagation. When the model weights are updated, the backpropagation continues through the series of cached hidden states, which improves the RNNLM's ability to selectively retain information that will improve future predictions (Mikolov et al., 2010).

Notably, the RNNLM is impacted by the *vanishing gradient problem* if backpropagation continues through the recurrent connection for too many time steps (Mikolov et al., 2014). Small gradient steps of the tanh and other sigmoidal functions decrease exponentially through each layer in backpropagation, so the vanishing gradient problem describes the fact that error will eventually reduce to zero and beyond a certain number of cached time steps, backpropagation will no longer improve the performance of a RNNLM (Mikolov et al., 2014). Similarly, the RNNLM's recurrent connection can retain contextual information for a theoretically infinite number of time steps, but contextual information provided by more recent words drowns this information out and the context retention of a RNNLM is limited to approximately 10 or 20 words in practice (Mikolov et al., 2014).

3.4 Previous Multi-Prototype Word Embedding Methods

The neural network language models described in previous sections have a number of strengths and have achieved state-of-the-art performance on a number of natural language processing tasks. These models all make the same naive assumption – that every word should only be represented with a single word vector. This is due to the assumption is that high-dimensional embeddings are capable of encoding a polysemous word's independent meanings and that the word's surrounding context is sufficient to disambiguate the word during training and evaluation, but we find that this assumption disagrees with the semantic structures evident in modern word embeddings.

However, assuming that each word should only have a single word vector has two benefits: it decreases the amount of storage required to store the entire vocabulary and disarms the difficult technical question of how one can learn the multiple meanings per word if polysemous words aren't disambiguated in a training text. The latter benefit is especially important due to the nature of training modern NNLM's on huge amounts of text collected from a variety of sources (typically through the Internet). Learning effective word embeddings requires a large text corpus with a variety of topics and linguistic styles, so disambiguation of every word in such a large corpus is impossible. The following approaches rely on a disambiguation algorithm to cluster word meanings in the training corpus before training of the neural network language model begins, which requires a predetermined number of word meanings that are chosen either by checking the number of definitions in of each word in a dictionary such as WordNet or assuming **K** meanings for every word.

Clustering Approaches. A number of neural network language models have been designed with the belief that encoding polysemous words with a single vector representation will negatively impact the structure of word embedding spaces. The first successful multi-prototype model was proposed by Reisinger and Mooney (2010). This model generates K prototype vectors for every word in the vocabulary using the Word Sense Discovery method described in Schütze (1998). That is, for every word, all of the word's occurrences over the training text corpus are collected and converted into *occurrence vectors* by calculating the bag-of-words representation of its 10 surrounding words. These occurrence vectors are then clustered using the Buckshot algorithm (Cutting et al., 1992), a generalized agglomerative clustering algorithm. To generate a word's **K** prototype vectors using the Buckshot algorithm, one would initially cluster only $\sqrt{K \cdot n}$ of the word's **n** occurrence vectors. The remaining occurrence vectors are then assigned to their respective nearest cluster using cosine similarity. Cosine similarity is an effective metric for semantic analysis of fully trained word embeddings and is defined as the cosine of the angle between any two vectors:

$$\textit{cosine similarity}(v_1, v_2) = \cos(\theta(v_1, v_2)) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} \in [-1, 1] \qquad (29)$$

Finally, an Expectation Maximization stage iteratively re-calculates cluster centroids and then re-assigns documents until a locally optimal solution is found. The K discovered cluster means are used as the embedding vectors for each of the word's K prototypes.

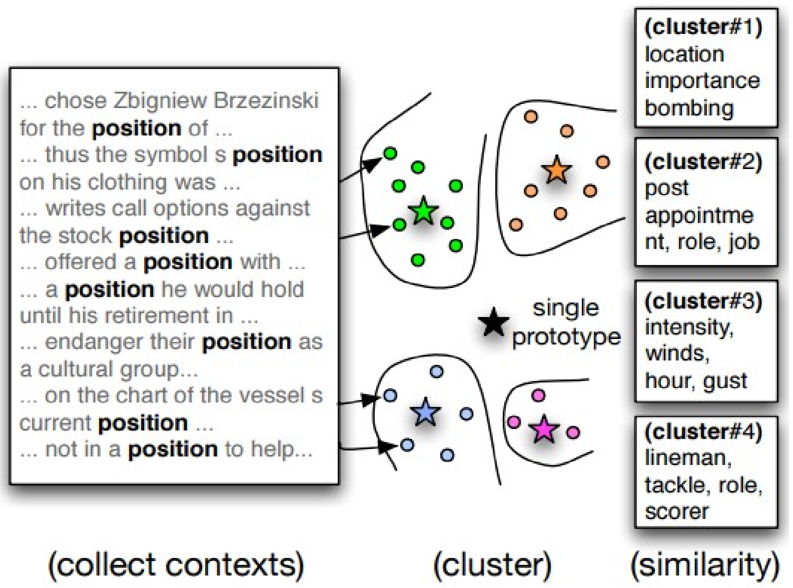

... chose Zbigniew Brzezinski for the **position** of ...
... thus the symbol s **position** on his clothing was ...
... writes call options against the stock **position** ...
... offered a **position** with ...
... a **position** he would hold until his retirement in ...
... endanger their **position** as a cultural group...
... on the chart of the vessel s current **position** ...
... not in a **position** to help...
single prototype
(cluster#1) location importance bombing
(cluster#2) post appointme nt, role, job
(cluster#3) intensity, winds, hour, gust
(cluster#4) lineman, tackle, role, scorer
(collect contexts)
(cluster)
(similarity)

Figure 8. The context-clustering step proposed by Reisinger and Mooney (2010). The words listed for each cluster are determined by highest cosine similarity to each respective cluster mean.

Figure 8 demonstrates the intuition behind the Reisinger and Mooney (2010) clustering approach. Tasked to find four meanings for the word "position," the Buckshot algorithm identifies four clusters of occurrence vectors. The "single prototype" centroid is the mean of all of the word's occurrence vectors and demonstrates how single-prototype models conflate the meanings of polysemous words in such a way that the word's final encoding may not be near any of its meanings' synonyms.

Reisinger and Mooney (2010) also demonstrate how different similarity metrics can be used to identify synonyms and words that are conceptually related to each of the word prototypes. Figure 8 illustrates four clustered regions of context vectors for the various contexts of the word "position" in the training text. Reisinger and Mooney (2010) note the "hurricane" cluster is unusual. It can be expected that techniques that make naive assumptions on the number of prototypes per word will make these mistakes.

The largest notable difference between Reisinger and Mooney (2010) and other multi-prototype word embeddings (including our own) is that word embeddings are not just affected by a given word's multiple contexts, but directly calculated from them. It is also the only multi-prototype encoding technique that does not rely on neural network language models to generate word embeddings. However, the Reisinger and Mooney (2010) technique demonstrates that multi-prototype models designed to encode the multiple meanings of a given word can outperform their single-prototype counterparts in word-similarity tasks.

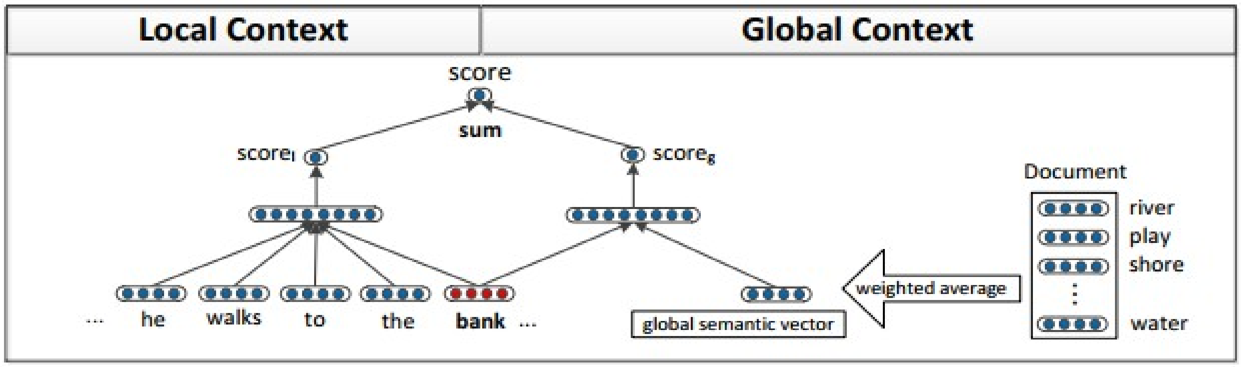


Figure 9. A neural network language model proposed by Huang et al. (2012).

A second clustering-based approach for generating multi-prototype word embeddings was proposed by Huang et al. (2012). This approach used a neural network of a unique design for language modeling. In comparison to the neural network language models proposed by Bengio et al. (2003) and Mikolov et al. (2010), the Huang et al. (2012) neural network's output did not make a prediction of the next word. Instead, a candidate for the next word is one of the model's inputs and the neural network is trained to output a score indicating whether the candidate word likely to occur as the next word in the sequence. A second major difference to this model was that the word vectors for all of the words in the document were given as one of the input vectors to the neural network. To allow documents of any size to be given as input, all of the vectors were averaged together, weighted by their respective idf scores calculated over all of the documents in the entire training corpus, **D**.

$$idf(w, D) = \log \frac{|D|}{|\{d \in D ; w \in d\}|} \tag{30}$$

The Huang et al. (2012) model has a number of benefits. The number of weights in the model is drastically reduced compared to preceding models that used a soft-max output layer to calculate a probability distribution over the entire vocabulary. As such, training and computation time can be reduced. Training can mix positive reinforcement (using backpropagation to increase the output score of the correct word) and negative reinforcement (choosing random incorrect words and using backpropagation to decrease their scores). The model also contributes a novel method for integrating global document context for the prediction. Previous models with contextual memory faced difficulties with the vanishing gradient problem, such as the Mikolov et al. (2010) recurrent neural network language model, which would lose contextual information after 5-10 words (Mikolov et al., 2014). Although the idf-weighted global semantic vector may average out critical contextual clues, the model avoids exponentially decaying memory.

Once the neural network training performance stabilizes, Huang et al. (2012) explain how the resulting single-prototype word embeddings can be used to generate a multi-prototype word embedding. The approach is very similar to the Reisinger and Mooney (2010) method described above. For every word, the occurrences over the training set are collected. The vectors for the 5 words before and after the target word are averaged, weighted by their respective idf scores. The resulting occurrence vectors are then clustered using the Spherical K-Means algorithm with K=10. Each occurrence of the word in the training set is then labeled with its corresponding cluster. Afterwards, a new word embedding is trained on the labeled corpus.

The Huang et al. (2012) model provides an effective method for generating multi-prototype word embeddings in two stages. A single-prototype word embedding is generated in the first stage and facilitates the disambiguation of word occurrences in text. This model relies on the context-clustering algorithm to be highly accurate in order to learn accurate prototype vectors in the second training iteration. We will return to this assumption later, when we discuss our neural network language model and how it is designed to explicitly learn word sense disambiguation over the training corpus and create a multi-prototype embedding that is useful for both word sense disambiguation and language modeling.

Inspired by both Reisinger and Mooney (2010) and Huang et al. (2012), Neelakantan et al. (2014) proposed a model architecture that further expanded upon the clustering-based approaches for generating multi-prototype word embeddings. The Multi-Sense Skip-Gram (MSSG) model proposed by Neelakantan et al. (2014) utilizes a traditional skip-gram model (Mikolov et al., 2013a), which is explained in detail in Section 3.4, and subsequently trains a multi-prototype word embedding using the MSSG model illustrated in Figure 10.

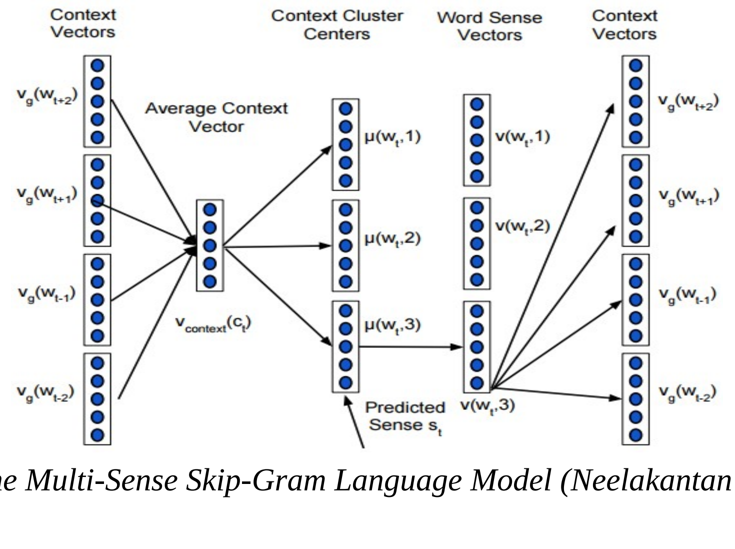


Figure 10. The Multi-Sense Skip-Gram Language Model (Neelakantan et al., 2014).

The MSSG approach first utilizes a skip-gram model to generate a single-prototype word embedding over the entire vocabulary. These single-prototype vectors are called *global vectors* and referred to as $\mathbf{v_g(w)}$ for any given word **w**. Although the MSSG model is designed to generate multi-prototype vectors, it continues to use the learned single-prototype global vectors as input.

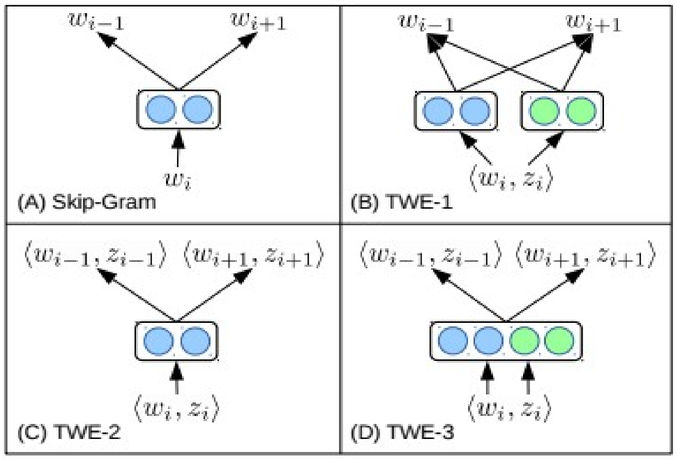
Once initial training of the single-prototype word embedding is complete, *context vectors* are calculated for every word in the corpus and are then clustered to induce sense vectors. For each word in the vocabulary, each of its occurrences in the training corpus is encoded as a context vector by calculating the non-weighted average of the 4 surrounding words in text. Then, a K-means clustering algorithm is run over the context vectors for a given word to identify K=3 sense regions. The cluster centroids, $\boldsymbol{\mu}_\mathbf{g}\mathbf{(w)}$, are saved for disambiguation and training the multi-prototype word embedding.

Training sense vectors for the entire vocabulary relies on using the MSSG model (Figure 10). At time **t**, the global vectors for the context words surrounding word **w(t)** are averaged. The context vector, $\mathbf{v_{context}(c_t)}$, is compared to the word's context cluster centroids, $\boldsymbol{\mu}\mathbf{(w_t)}$, and the correct word sense is assumed to be the sense with the nearest centroid. The final two layers in Figure 10 represent a standard Skip-gram model. Once the word sense is chosen according to the nearest centroid, the Skip-gram model is trained to increase the likelihood that the chosen sense vector can predict the original input context's global vectors.

MSSG's backpropagation step at each time **t** updates the global vectors for the surrounding context words, the chosen sense vector, and also updates the context cluster associated with the chosen sense. After updating the context words, the averaged context vector is added to the chosen cluster and the cluster centroid is recalculated. This leads to a parallel training process in which the word senses themselves and the disambiguation method are simultaneously improved.

Neelakantan et al. (2014) also proposes a non-parametric variant of the MSSG that removes the assumption that every word has only **K** senses or prototype vectors. Instead, the proposed NP-MSSG model utilizes an online clustering method originally designed for optimal facility placement (Meyerson, 2001). The Meyerson (2001) algorithm is capable of clustering high-dimensional vectors as they arrive in an online fashion and determining when a new cluster is necessary. The algorithm has a run-time of O(1)-competitive time per vector when vectors arrive in random order and worst-case run-time of O(log(n)) in the adversarial setting. This allows the NP-MSSG model to generate new word senses when a newly encountered average context vector $\mathbf{v}_{\mathbf{context}}(\mathbf{c}_t)$ is too distant from the existing context centroids $\boldsymbol{\mu}(\mathbf{w}_t)$. When all cluster centroids have a cosine similarity below the hyperparameter $\boldsymbol{\lambda}\mathbf{=-0.5}$, a new sense cluster is generated and $\mathbf{v}_{\mathbf{context}}(\mathbf{c}_t)$ becomes its only member.

Topic Modeling Approach. Liu et al. (2015) proposed three models in which a word's meaning can be identified using topic modeling approach. Each word in the training corpus is initially labeled with one of **T** global topics that are shared by the vocabulary. Topic modeling approaches do not directly cluster vectors, but the document regularities are used to identify likely topics for each document and assign word senses based upon the document's assigned topic. The models proposed by Liu et al. (2015) utilize the latent Dirichlet allocation (LDA) algorithm (Blei et al., 2003) and Gibbs sampling (Griffiths and Steyvers, 2004) to generate the initial topic labels over the training corpus.



w_{i-1}
w_{i+1}
w_i
(A) Skip-Gram
w_{i-1}
w_{i+1}
$\langle w_i, z_i \rangle$
(B) TWE-1
$\langle w_{i-1}, z_{i-1} \rangle$
$\langle w_{i+1}, z_{i+1} \rangle$
$\langle w_i, z_i \rangle$
(C) TWE-2
$\langle w_{i-1}, z_{i-1} \rangle$
$\langle w_{i+1}, z_{i+1} \rangle$
$\langle w_i, z_i \rangle$
(D) TWE-3

Figure 11. Illustrations of the skip-gram models proposed by Liu et al. (2015). Each model incorporates document-level topic predictions to identify word senses.

The neural network structure of the three models, labeled TWE-1 through TWE-3, are compared to a standard Skip-gram model in Figure 11. Every word token $\mathbf{w_t}$ in the training corpus is randomly assigned a topic $\mathbf{z} \in \mathbf{T}$ with probabilities weighted according to the conditional probabilities learned from LDA according to the following equation:

$$Pr(z_t | w_t, d) \propto Pr(w_t | z_t) \, Pr(z_t | d) \tag{31}$$

The probabilities $\mathbf{Pr(w_t|z_t)}$ and $\mathbf{Pr(z_t|d)}$ are estimated by the LDA algorithm during the initial topic modeling calculations. In Figure 11, one can see that the TWE-1 and TWE-3 models encode global *topic vectors* that are shared among all words in the vocabulary. This is unlike our motivation of generating unique sense-specific vectors for every word in the corpus. As such, we focus on the design of the TWE-2 model for direct comparisons to our own work.

TWE-2 is a Skip-gram approach that encodes multiple prototypes for every word in the corpus, similar to existing approaches to the MPNNLM model we describe in Section 3.5. All word tokens in the training set are labeled before training the Skip-gram model. All three TWE models evaluated by Liu et al. (2015) outperformed existing state-of-the-art multi-prototype models mentioned above. Notably, all TWE models label word tokens according to the predicted topic of the parent document.

3.5 A Multi-Prototype Neural Network Language Model

For the remainder of this work, we will discuss the design and results of a novel neural network language model (NNLM) that generates multi-prototype word embeddings and probabilistically disambiguates words given their contexts. Our Multi-Prototype Neural Network Language Model (MPNNLM) is influenced by many of the neural network language models in sections 3.3 and 3.4, but is most similar to that of Bengio et al. (2003) and was designed specifically to address previous models' shortcomings in modeling polysemous words with multiple prototypes.

The Multi-Prototype NNLM. The structure of the MPNNLM is most similar to the n-gram inspired NNLM proposed by Bengio et al. (2003) (see Figure 12, page 27). The input to this model is a concatenation of the word vectors for the previous $\mathbf{n}$ words, $\{\mathbf{w}_{t-n}, \ldots, \mathbf{w}_{t-1}\}$ and its output is a soft-max prediction over the entire vocabulary of which word is most likely to occur next $\mathbf{w}_t$.

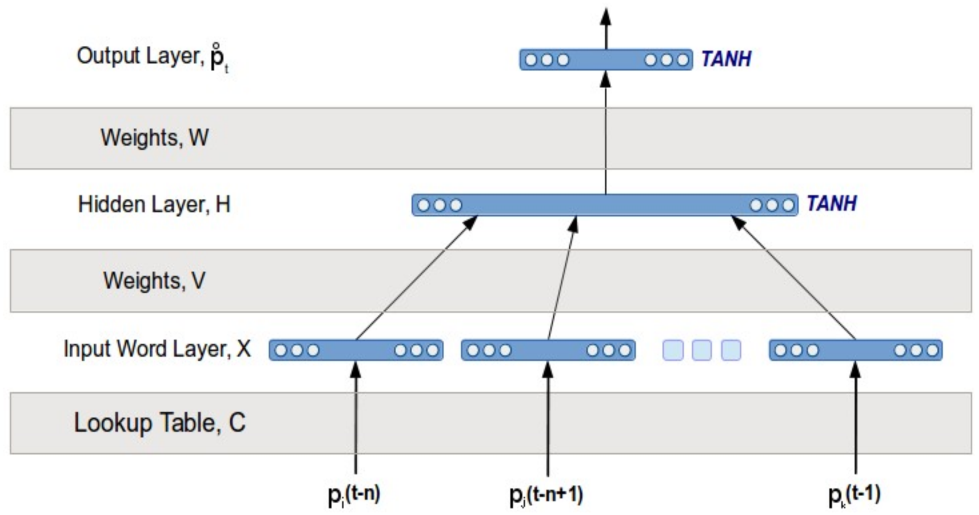


Figure 12. The MPNNLM is very similar to the neural network language model proposed by Bengio et al. (2003), but has a tanh output layer that predicts back into the word embedding vector space.

The MPNNLM is also an n-gram feedforward NNLM with a single hidden layer. To generate a prediction, the input is a concatenation of the prototype vectors corresponding to the preceding **n** word meanings in text, $\{\mathbf{p}_{t-n}, \ldots, \mathbf{p}_{t-1}\}$. We operate under the assumption that all text the MPNNLM will encounter in the training or evaluation text will not include word meanings, so the selection of each corresponding word's prototype can be done at random or through more systematic methods. We will later discuss how the MPNNLM may be used for Word Sense Disambiguation of word $\mathbf{w}_t$ and how this can be used to disambiguate input words to determine the prototypes for input.

The major structural difference between the MPNNLM and Bengio et al. (2003) is in the replacement of the softmax output layer with a vector-space prediction layer. Bengio et al. (2003) directly generates a prediction of the next word using a softmax layer as the output, which has two advantages: the softmax layer estimates the conditional probability over the entire vocabulary and is continuously differentiable, an important requirement for backpropagation. The MPNNLM replaces the softmax layer with **d**-dimensional vector prediction back into the word embedding space. Note that because the MPNNLM uses tanh activation functions in the hidden and output layer, the output is a real-valued prediction vector, $\mathring{p} \in [0,1]^d$, which we later use to generate the actual probability distribution over the next word.

Vector-Space Predictions. Many historical NNLM's make their prediction of $\mathbf{w_t}$ using a vocabulary-sized softmax output layer, which generates a well-formed probability distribution over the entire vocabulary. As mentioned above, the MPNNLM has a similar design to Bengio et al. (2003). The input layers of both models are a **n•d**-length concatenation of the word vectors for the previous **n** words, $\{\mathbf{w_{t-n}}, \ldots, \mathbf{w_{t-1}}\}$. In place of Bengio et al. (2003)'s softmax output layer, the MPNNLM generates a **d**-dimensional *prediction vector* of the same dimension as the word embedding.

Note that the MPNNLM is trained over an unlabeled text corpus without access to labeled meanings for each word token in the corpus, so it is unclear which prototype of $\mathbf{w_t}$ may be the target output vector. In the language modeling sense, the MPNNLM is tasked to predict $\mathbf{w_t}$, so we combine the prototype-level predictions for each word into a single word-level probability. We will explain how the MPNNLM chooses the appropriate prototype vectors for the input layer at time **t** and determines the "correct" output prototype for $\mathbf{w_t}$ in our subsequent discussion of Word Sense Disambiguation.

When the MPNNLM makes its vector-space predictions back into the word embedding, this does not yield a direct probabilistic prediction of the next word as is expected of an NNLM. There are a variety of ways to generate a probability distribution over the target vocabulary given a prediction into the word embedding. For the MPNNLM, we use the following softmax cost function:

$$P(w_t = w_i | w_{t-n}, \ldots w_{t-1}) = P(w_i | \mathring{p}) = \frac{\sum_{p_k \in w_i} e^{\frac{-D(p_k, \mathring{p})^2}{2}}}{\sum_{w_j \in V} \sum_{p_k \in w_j} e^{\frac{-D(p_k, \mathring{p})^2}{2}}} \quad (32)$$

Provided that the distance function **D** is universally continuously differentiable, the overall softmax of calculation in Equation 32 satisfies the continuous differentiability requirement of the backpropagation algorithm (Rumelhart, 1985). In vector-space models, the typical considerations for distance functions are cosine similarity (Equation 29, page 71) and Euclidean distance. The use of cosine similarity as a distance metric may be a beneficial feature to consider in future implementations of the MPNNLM, but for our experiments we used Euclidean distance.

Modeling Word Prototypes. The English language is highly polysemous and state-of-the-art word embeddings typically encode only a single embedding per word. We find that this leads to word embeddings that are accurate for single-meaning (*monosemous*) words, but single-prototype word embeddings often strand a polysemous word's vector representation in an area between various regions that contain the word's synonyms (Li and Jurafsky, 2015). Similarly, due to the focus on high-likelihood predictions, word embeddings often neglect to accurately represent rare meanings of polysemous words (Li and Jurafsky, 2015). The MPNNLM is designed to encode multiple *prototype vectors* that individually model a single meanings or atomic concept within a word. By unbinding these meanings, we expect that the word's prototype vectors will to these regions of synonymy and further improve model's language modeling performance.

In general, it is difficult to determine the appropriate number of meaning vectors for each word and previous multi-prototype models determine the number of prototypes for each word before analyzing the text corpus. A number of previous multi-prototype models have assumed that exactly **K** prototype vectors should be encoded for every word in the vocabulary. Training **K** prototype vectors for every word in the vocabulary relies on an inherent assumption that polysemous and monosemous words require the same number of representations and that the same must be true for extremely rare and common words. This is akin to the assumption that every word should be encoded with a single vector, but modeling **K** prototypes per word been shown to improve language modeling performance (Reisinger and Mooney, 2010; Huang et al., 2012; Neelakantan et al., 2014; Liu et al., 2015). An alternative approach proposed by Chen et al. (2014) allows each word to have a different number of prototypes relies on the number of dictionary definitions in WordNet to determine the number of prototypes per word. This is an intuitive assumption, but requires that WordNet's meanings for each word are fully annotated and may not perform well if some meanings of a word are not used in a given corpus, which is especially likely if we expect the word embedding to be learned over a smaller domain-specific corpus. Further detail about these models can be found in Section 3.3.

Our model is designed to first train a single-prototype word embedding, which provides initial semantic structure to the vector space, and to later generate new prototypes for polysemous words before resuming its training over the text corpus. This is intended to give our model the freedom to identify unknown word meanings that the MPNNLM has difficulty predicting. However, we do limit the number of prototypes per word based upon the word's frequency in the training text. This is inspired by the analysis of Zipf (1945), which demonstrated that a power-law distribution can be fitted to estimate the number of meanings of a word given its word's frequency in written text.

Word Sense Induction. When evaluating the MPNNLM's ability to disambiguate a word token's meaning given its context, we are hindered by the fact that we do not have ground truth meanings for any given token in the training text. The text corpora used to generate word embeddings with NNLM's are typically very large (in the millions or billions of words) and are too expensive to fully disambiguate. Machine learning methods approach this situation in different ways: by bootstrapping the training corpus with a small sense-disambiguated corpus (this would make the MPNNLM a semi-supervised method) or we can infer the meanings of the words in the unlabeled corpus and remain fully unsupervised. In our case, we use the MPNNLM's ability to predict into the vector space to make a prediction of the meaning of a word given its context.

Furthermore, unless prototypes align with clear synonyms or conceptual clusters, it is not always clear exactly what meaning a prototype vector may represent given only its relative position within the word embedding. We may be able to infer various word meanings based on the selections it makes over a word corpus, but the probabilistic selection method may negatively impact that process. Instead, the MPNNLM is evaluated on word sense induction (WSI), an unsupervised version of word sense disambiguation (WSD).

We can interpret predictions of the MPNNLM to be similar to a human approach to language modeling. Due to the difficulty and highly probabilistic nature of language modeling, we do not expect the MPNNLM to generate predictions that are closer to a prototype of $\mathbf{w}_t$ than any other prototype in the word embedding every time. Instead, we expect each prediction to identify a region of perhaps synonymous words that are most likely to occur next given the context. By training multiple prototypes per word, the MPNNLM can model each meaning of a word as a prototype located near related prototypes of synonymous words, which allows us to to infer the actual meaning of $\mathbf{w}_t$ given its context.

In our design of the MPNNLM, we make the assumption that words will cluster with their synonyms, near other words that belong to the same categories categories, and according to other linguistically similar words. The MPNNLM generates predictions based upon the preceding context, so the backpropagation algorithm will improve predictions by effectively clustering words that occur in the same context. This even includes antonyms, although this may defy expectations – "I adjusted the thermostat because it was too ____," has two very likely answers, "hot" and "cold".

When the MPNNLM generates an actual probability distribution during its prediction of **w$_t$**, Equation 32 sums all of the individual prototype probabilities for each candidate word. The MPNNLM also uses these probabilities to infer the meaning of **w$_t$**. The MPNNLM draws uniformly random from the prototype probabilities of **w$_t$** and weights the selection according to the individual softmax distance from Equation 32.

$$P(p_i|w_{t-n},\ldots w_{t-1},w_t,\mathring{p})=\frac{e^{\frac{-D(p_i,\mathring{p})^2}{2}}}{\sum_{p_k\in w_j} e^{\frac{-D(p_k,\mathring{p})^2}{2}}} \tag{33}$$

Once the meaning of **w$_t$** is inferred, we assume that the selected prototype is correct for training of the neural network with backpropagation. Initially, we expect this process to be highly random, but as words begin to cluster, predictions of semantically clustered word vectors will likely identify the prototypes most relevant to a given context. Finally, after the backpropagation step is complete, we can update the input words for the next step. We shift the input context vector by **d** (which effectively removes the oldest input prototype from the input) and append the selected prototype for **w$_t$** to the input vector.

It is clear that the accurate selection of the target word's prototype is critical to the training process for the MPNNLM. If the selection process is inaccurate, the MPNNLM will have a difficult time training a meaningful word embedding and its subsequent predictions will rely on an input vector containing an incorrect meaning of **w$_t$**.

By using the input context to select the input word prototype for the next time step, interesting implications arise. At time **t+1**, the prediction of **w$_t$** is not based solely on words **w$_{t-n}$** through **w$_{t-1}$**, but also on the preceding words that informed prototype selection for the input words as well. As such, the MPNNLM has an inherent memory retention property and predictions are not limited to the simple n-gram input.

Backpropagation. The cost function in Equation 32 allows us to train our word embedding in a more advanced way than typical of NNLM's. We make a number of adjustments to the standard backpropagation algorithm that allow us to not only update the network weights of the neural network and the vector representations of the input prototypes, but we can also use the cost function to intelligently update prototypes for the target word $\mathbf{w_t}$ and all other words in the vocabulary. Once the MPNNLM has generated a prediction vector for $\mathbf{w_t}$, backpropagation proceeds as follows:

1. Generate the probability distribution over the vocabulary using Equation 32 and infer the meaning of $\mathbf{w_t}$ by selecting one of its prototypes according to Equation 33.
2. Update all prototypes for $\mathbf{w_t}$ by first taking the partial derivative of Equation 32 and moving each prototype slightly toward the MPNNLM output prediction vector using a small gradient descent step. This increases the probability that $\mathbf{w_t}$ will be predicted in future iterations with similar contexts.
3. Update all remaining prototypes in the vocabulary using the same partial derivative, but moving *away* from the predicted output vector of the MPNNLM. This decreases the probability of competing words from being predicted in similar contexts in the future.
4. Finally, update the neural network weights using traditional backpropagation by first taking the partial derivative of Equation 32 with respect to each output node. Beyond the fact that the NPNNLM calculates a softmax probability based upon the full vocabulary's word embedding and the MPNNLM output vector, the probability distribution is calculated in a very similar method a the softmax output layer of Bengio et al. (2003), so we take a similar gradient step here to determine the output error of the MPNNLM prediction vector before backpropagating to the hidden layer in the traditional way.

Step 3 is the most computationally expensive step of training the MPNNLM. By generating the probability distribution over the entire vocabulary and updating all incorrect words' prototypes according to gradient descent, each training iteration requires computation that scales linearly in the number of prototypes in the word embedding. However, we can speed up the training process by calculating the output probability distribution only over a random subset of the words at each training iteration. In order to speed up our experiments, we select only a subset of the incorrect words for every prediction and scale their output probability mass to estimate the total probability mass of the entire vocabulary. For a vocabulary in the hundreds of thousands of words, this greatly reduces the time required to train the MPNNLM while

We also used a *momentum*-based gradient descent algorithm for backpropagation. Neural networks are sensitive to the setting of the *learning rate* parameter, which determines how small of an adjustment the neural network should make based upon the gradient of the current prediction, but the learning rate often needs to be adjusted by hand based upon the neural network structure and the data set. Momentum approaches allow the neural network to effectively continue increasing the learning rate as its performance improves and then decrease the learning rate once the performance seems to decrease (perhaps by taking too large of a gradient descent step).

Prototype Creation. A methodical approach for correct placement of new prototypes for the MPNNLM is the largest challenge for generating functional multi-prototype word embeddings. As we will see in Section 3.6, various experiments allowed us to determine methodological approaches for generating prototypes that accurately modeled the English language. We we summarize these insights here as well.

The MPNNLM is not inherently a multi-prototype model. In fact, if the MPNNLM encodes only one prototype for each word in the vocabulary, the MPNNLM framework will sufficiently operate in the single-prototype setting. We have found that initially training the

MPNNLM as a single-prototype model allows the word embedding to learn a high-level semantic structure of the vocabulary based on contextual information and word synonymy. This single-prototype word embedding provides the foundation for the subsequently generated word prototypes. Ideally, these new prototypes will navigate to a region that is synonymous or otherwise semantically relevant to one of the word's meanings and underrepresented by existing prototypes.

A methodical approach to generating new prototypes is a critical component to our model. After the backpropagation step is completed for the current word $\mathbf{w_t}$, the MPNNLM will add a new prototype for $\mathbf{w_t}$ if the following criteria are satisfied:

1. The MPNNLM perplexity over a held-out data set must be less than $\mathbf{\hat{P}}$.
2. All existing prototypes for $\mathbf{w_t}$ must have been selected at training time at least $\mathbf{\hat{T}}$ times.
3. If $\mathbf{w_t}$ occurs $\mathbf{n}$ times in the training corpus, it is limited to $\mathbf{log_{10}(n)}$ prototypes.
4. The new prototype must be generated at least euclidean distance $\mathbf{\hat{D}}$ from the nearest existing prototype.

The first two criteria are straightforward. The MPNNLM periodically evaluates perplexity over a small held out data set. The held-out perplexity must be below $\mathbf{\hat{P}}$, a preselected threshold that we find is corpus-dependent. It is expected that a model's final perplexity is dependent on the uniformity of language within the corpus and size of the modeled vocabulary. During training, most neural network language models have a sharp initial drop in perplexity and begin to level out and we have found that a good value of $\mathbf{\hat{P}}$ is after perplexity begins plateauing. Similarly, to ensure that new prototypes are not generated unless necessary, the second criteria requires that a new prototype is not created until all existing prototypes for a given word have been trained at least $\mathbf{\hat{T}}$ times.

The third criteria is inspired by Zipf (1945), which found that the number of definitions known for a polysemous word correlates logarithmically with its frequency in written text. We use this insight to create limit the number of prototypes the MPNNLM may generate for a given word, regardless of the word's number of definitions. This is in contrast to previous approaches, which decide the number of prototypes a word will have without determining the. Similarly, limiting a word's prototypes based on its frequency over the training corpus seems sensible given that the word will have fewer unique contexts to accurately train its various prototypes.

Finally, once a prototype meets the previous criteria, its location must be chosen. We rely on the previous MPNNLM predictions for a given word to determine likely regions of meaning for a given word. During training, the we store $\mathbf{\hat{P}}$ predictions for every word. These are the MPNNLM predictions into word embedding space that were generated when the model previously attempted to predict word $\mathbf{w_t}$. Assuming that the MPNNLM is sufficiently accurate, we can imagine that even incorrect predictions for a word may be generated into a synonymous space for an unknown meaning of the word and we attempt to identify one of these regions when creating a new word prototype.

We clustered the previous $\mathbf{\hat{P}}$ predictions for $\mathbf{w_t}$ using agglomerative clustering, a hierarchical approach that begins with all previous prediction vectors in their own clusters and iteratively joins the two clusters with the "closest centroids" until a single cluster remains. We use the cosine similarity metric to measure proximity and given a cluster of vectors, the euclidean average of is used to define the cluster's centroid. With $\mathbf{\hat{P}}$ previous predictions, the agglomerative clustering will yield between $\mathbf{\hat{P}}$ and $\mathbf{2\hat{P}\text{-}1}$ candidate cluster centroids for the new prototype. Of these candidate prototypes, we choose the candidate pertaining to the largest cluster with distance at least $\mathbf{\hat{D}}$ from all existing prototypes for the given word.

3.6 Experimental Results

Google Gigaword Data Set. Modern corpora for linguistic tasks are typically chosen based on the task at hand. For example, classification algorithms are frequently evaluated over newspaper articles with topic labels or Amazon reviews with the associated star rating. For the neural language models such as the MPNNLM, very large text corpora are necessary for learning general linguistic regularities over a broad range of topics. We used the Google Gigaword corpus (Chelba et al., 2013), a general text corpus comprised of sentences (not documents or even sequential sentences) with 793,471 vocabulary words and 820 million tokens.

50Cities Corpus. Many of our experiments were run on the 50Cities corpus, a sample of the Google Gigaword corpus that we created for the purpose of evaluating the MPNNLM's performance on targeted NLP tasks. Modern text corpora for modern NLP tasks are either very large and unlabeled (e.g. Google Gigaword (Chelba et al., 2013)) or small and hand-labeled with metadata related to a specific task (e.g. Reuters Aptemod). Our motivation is to train a model that learns linguistic patterns from large unlabeled corpora, but we also want guarantees that our corpus contains linguistic patterns relevant to our objective tasks. In order to evaluate our model on our target tasks of language modeling and set expansion, we sampled the Gigaword corpus for sentences within a targeted domain.

Our custom 50Cities corpus includes 7,219 sentences from the Gigaword corpus that mention at least one of the 50 largest cities in the USA as of January 2015 (US Census Bureau, 2016). Another 15,573 sentences were selected from the data set to provide general coverage of writing style, yielding 20,794 total sentences in our training set and 570,619 tokens. A held-out test set of 400 sentences was sampled from the Gigaword corpus in the same fashion that contained 11,221 word tokens.

Single-Prototype Vector-to-Vector NNLM. In order to validate our vector-to-vector neural network language model, our initial experiment generated a standard single-prototype word embedding. In these experiments, we set the dimension of the word embedding to **d=50**. The neural network was constructed with a context window of 4 input words and 100 hidden layer nodes. All initial weights of the neural network were drawn independently at random from [-0.01, 0.01] for all layers.

We focused our initial evaluations on identifying proper design settings and parameters for the MPNNLM that enable a successful training of a single-prototype word embedding. Due to operating in a vector-to-vector setting, a variety of typical neural network choices needed to be considered. One immediate question was how to generate a prediction for the first word in a sentence without preceding words to generate a prediction. Given zero background information of the context of a new sentence, we use **n** *start sequence prototypes* for use as the n-gram input at the beginning of a sentence, **<seq>**$_1$ through **<seq>**$_4$. These prototypes are given as the input to the model before every sentence begins, so we expect that the MPNNLM will eventually learn to generate the unigram distribution of the first word of every sentence in the corpus. An alternative approach is to use a single **<seq>** start token **n** times at the beginning of each sentence, but initial experiments showed that the model had a marginally improved perplexity when using **n** distinct **<seq>** words and our analysis showed that this was due to improvements of predictions for words **2** through **n-1**. We did not evaluate this in later experiments or evaluate the significance of these performance improvements. We also create **n** *end sequence prototypes* that are expected to occur at the end of the sentence.

Our early experiments on the MPNNLM began with two questions: how or whether the output layer should be normalized. Most neural networks utilize a sigmoidal function as an activation function for each neuron, but it was unclear whether a vector-to-vector model would be improved with an unconstrained word embedding. Without an activation function, calculating each dimension of the prediction would be a simple unscaled dot product, $\mathring{p}_i = \sum W_i \cdot H$, and would allow the neural network to generate unconstrained predictions in $\mathbb{R}^{\mathbf{d}}$. Initial experiments indicated that the tanh activation function provided the best perplexity scores.

The loss function of the MPNNLM (Equation 33, Page 85) has a gradient that we use to improve the model's performance via backpropagation. This gradient can be used in four ways that improve the model's ability to predict $\mathbf{w_t}$: standard backpropagation of the MPNNLM (which updates the model's weights to make a prediction of $\mathbf{w_t}$ more likely in the future given the same input words in the future); moving $\mathbf{w_t}$ a small amount toward the prediction of the MPNNLM (also increasing the probability the MPNNLM would assign to $\mathbf{w_t}$ given the same input words); moving all words other than $\mathbf{w_t}$ a small amount away from the prediction of the MPNNLM (slightly reducing the probability assigned to words other than $\mathbf{w_t}$); and using the estimated backpropagation error at the input layer to modify the input word vectors as well. We specifically evaluated whether using backpropagation to update the input words would conflict with our updates to the word embedding at the output layer. In other words, updating the entire word embedding according to the gradient of the loss function at the output layer optimizes word vectors as *targets* for prediction by the MPNNLM. By updating the input words with backpropagation, we're also adjusting weights to be better *inputs* to the MPNNLM and this experiment was designed to determine whether optimizing for both objectives simultaneously would have a positive effect on model perplexity.

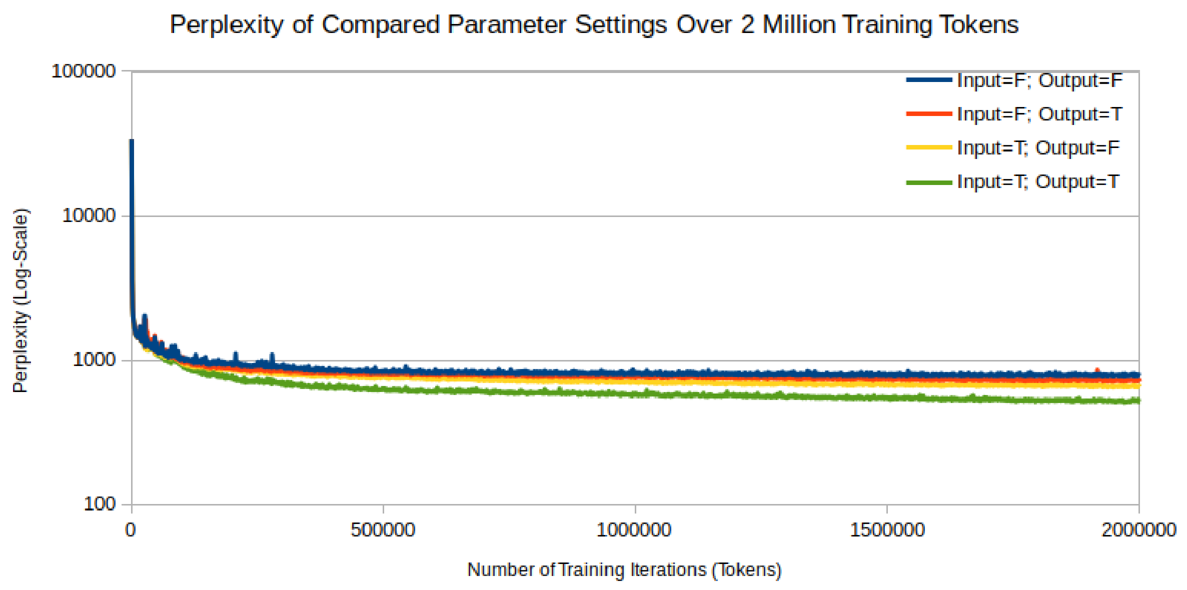


Figure 13. Perplexities of competing initial vector-to-vector settings. In the legend, ***Input*** *refers to whether the model continues backpropagation to the input words.* ***Output*** *is whether the model uses a tanh activation function for the output layer.*

In Figure 13, we see how the these considerations impact the performance of the MPNNLM. We trained the MPNNLM over 2 million tokens of the 50cities corpus. During the evaluation of these four Input/Output combinations of settings, we used the same random seed to generate network weights. Through these experiments, we found that both enabling backpropagation to the input word vectors and using a tanh activation function for the output layer directly improved the MPNNLM's perplexity. In the above chart, perplexity was calculated by generating a prediction over a 1000-word sequence from the held-out corpus.

Training Iterations	Input=F; Output=F	Input=F; Output=T	Input=T; Output=F	Input=T; Output=T
1000	**33962.8**	33963.4	34007.9	34008.1
500000	838.208	799.2	789.512	**637.878**
1000000	805.061	759.97	702.176	**590.135**
1500000	790.015	742.453	683.344	**553.214**
2000000	782.44	713.064	655.45	**516.488**

Table 12. Perplexity of different initial vector-to-vector settings corresponding to the charted results in Figure 13.

By enabling both settings, the MPNNLM finishes with a Perplexity of 516 after 2 million training iterations, compared to the next best performance – a perplexity of 655 achieved by disabling the tanh activation of the output layer. In the single-threaded setting, the training of these four models took approximately 20 hours each. We can see that both settings (backpropagating to input words and enabling tanh activation function of the output layer) individually improve MPNNLM's performance and the overall performance is best when both settings are enabled. The next important evaluation was to determine whether the word embeddings were semantically reasonable. For this experiment, we used the 50cities corpus.

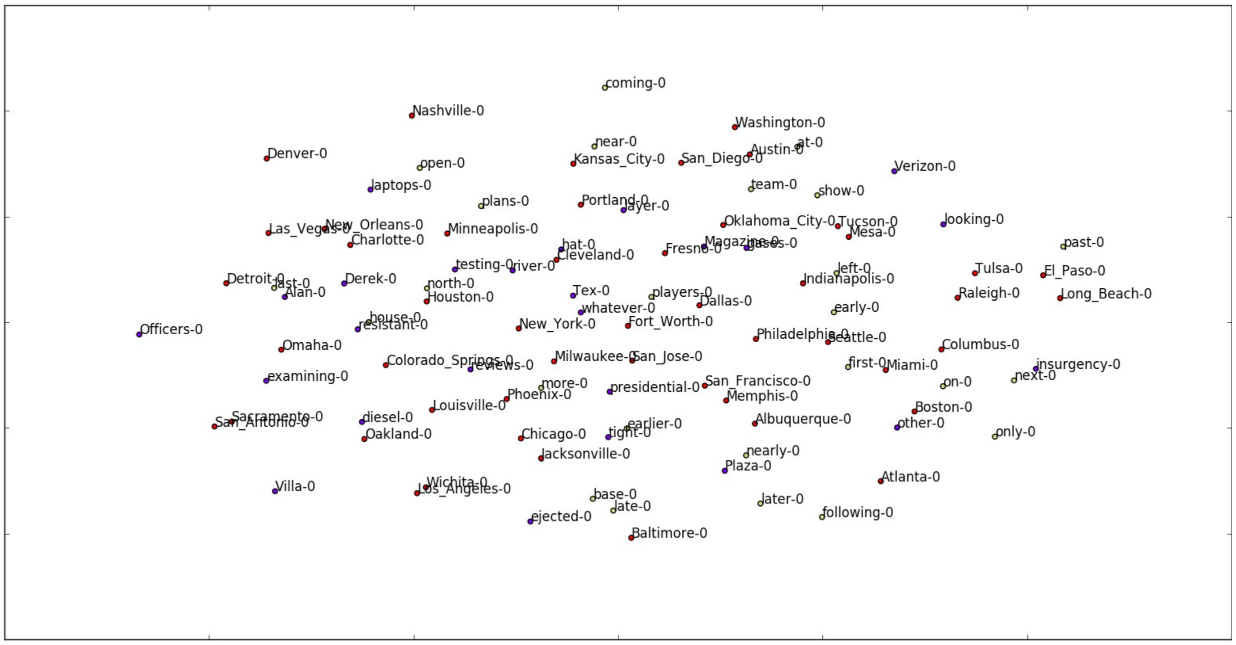


Figure 14. An initial distribution of 50 cities (red), 25 words related to "last" (yellow), and 25 random words (purple) projected to 2 dimensions using the t-SNE algorithm.

The initial word embeddings were distributed uniformly randomly in 50-dimensional space. Each word and each dimension of each word was drawn uniformly at random between -0.0001 and 0.0001. In Figure 14, we provide a 2-dimensional t-SNE projection (van der Maaten and Hinton, 2008) of the initial distribution of 150 words in the 50-dimensional word embedding space – the 50 top cities in the US by population, 25 words that were nearest to the word "last" in the and 25 randomly selected words. Each prototype has a color based upon its membership to these groups, a word label, and also is shown with its prototype to differentiate prototype locations in the multi-prototype setting (in Figure 14, there is only one prototype per word). The choice of the word “last” is due to it having a similar frequency in text as the word “Chicago”, which will be important in our last_Chicago experiment later. Early in our experiments, we collected the list of 25 words nearest to “last” that appear in Figure 14. Although this list consistently changes as we adjustment parameters or the training text corpus, we use this same list of words throughout our t-SNE projections of our experimental results to provide context and grounding for our results.

The t-SNE algorithm projects the word embeddings onto a 2-dimensional plane such that the linear distance between any two points in the projection is nearly proportional to the Euclidean distances of the corresponding prototype vectors in the high-dimensional word embedding space (van der Maaten and Hinton, 2008). t-SNE is an iterative non-deterministic process, so consecutive runs may differ globally (with reflections or rotations) or locally (generally with small perturbations), but the output is very useful for visualizing word embeddings. As expected, one can see in Figure 14 that the initial random placement of words is fairly uniformly distributed.

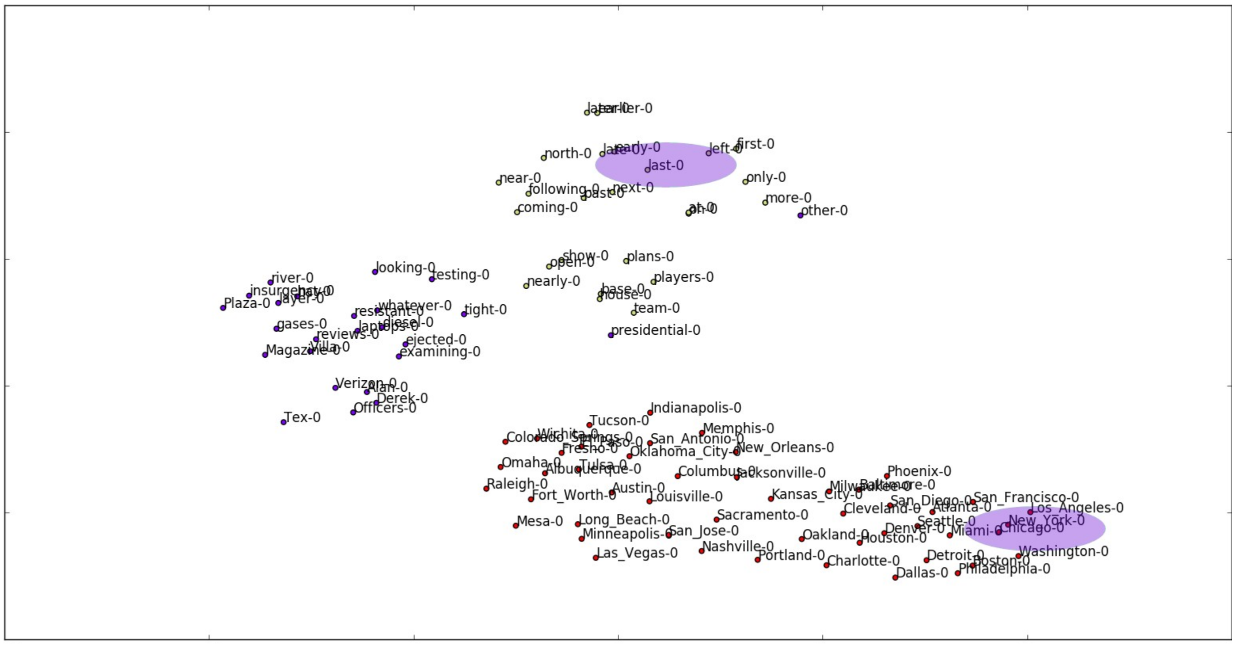


Figure 15. Distribution of word vectors from Figure 14 after 25 iterations of training over the 50Cities corpus.

After training the neural network over the 50Cities corpus for 25 epochs (nearly 15 million word tokens), semantic word clusters are visible in the t-SNE projection in Figure 15. The 50Cities corpus was created to determine the affects of polysemy on the word embedding space. In this traditional single-prototype setting, the visibly clustered words appear reasonable. The word vectors of the 50 cities (red) are tightly clustered in comparison to the randomly selected words (purple). We also see that the words related to “last” are fairly spread apart in comparison to the random words. In this case, it is partly due to many of the random (purple) words being fairly rare, so they are nearer to the origin than many of the other embeddings – untrained vectors are created near the origin and common words are updated the most, so they end up leaving the origin earlier. The words related to “last” are fairly common and although they are the 25 words nearest “last”, they are generally more spread apart than the random and city word clusters.

These results provide a very exciting validation of the initial model design. We can see that words find appropriate semantic clusters and that the MPNNLM is capable of reducing perplexity significantly while generating a word embedding. In the single-prototype setting, the perplexity scores are by no means state-of-the-art. However, our future experiments will continue to allow the MPNNLM model to develop deeper insights as it extends into the multi-prototype setting.

The last_Chicago Experiment. In comparison to the single-prototype vector-to-vector NNLM, the last_Chicago experiment is an incremental step forward. We wanted to determine whether the model could differentiate between two meanings of a known polysemous word. Rather than rely on a small word sense disambiguation corpus with labeled meanings of polysemous words, we used the unlabeled 50Cities corpus. The only change we made to the 50Cities corpus for this experiment was replacing all tokens of the words "last" and "Chicago" with "last_Chicago". These words were chosen because they have very different meanings but very similar word counts over the corpus (760 tokens for "last" and 723 tokens for "Chicago") so they would affect the training of "last_Chicago" nearly equally.

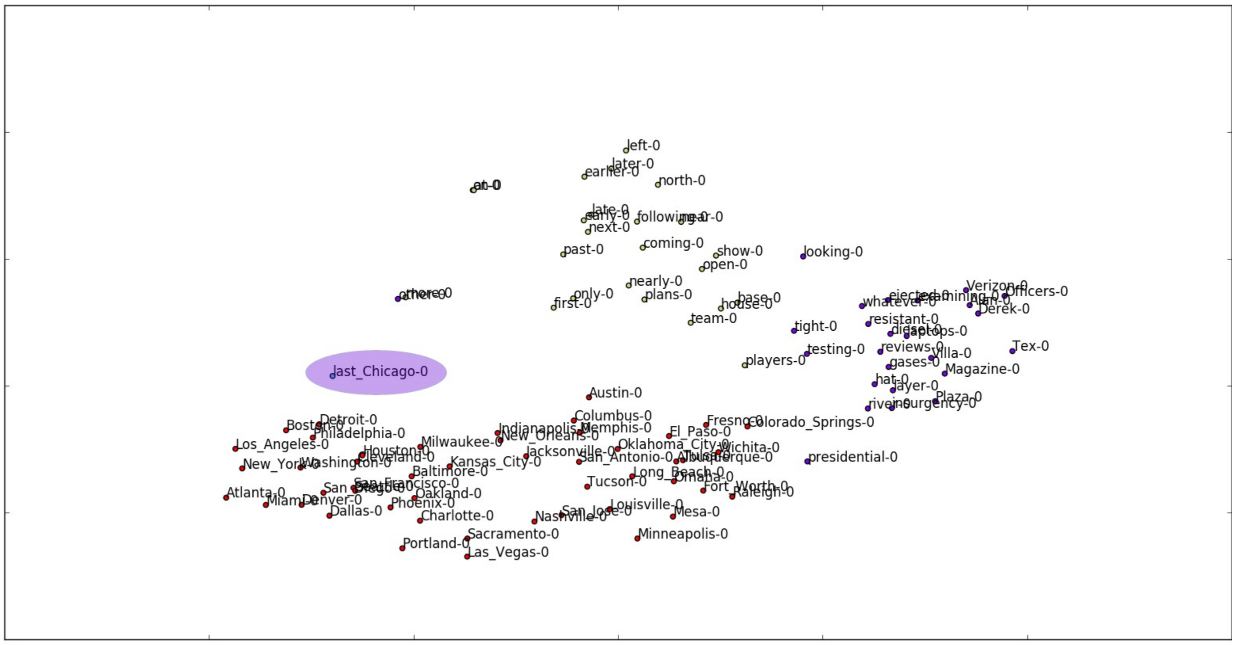


Figure 16. Distribution of single-prototype word vectors after 25 iterations of training over the 50Cities corpus when the word "last" and "Chicago" have been conflated.

As we can see in Figure 16, representing the word last_Chicago as only a single prototype vector has confused the MPNNLM about the meaning of either word. We intentionally chose two words with similar frequencies in order to control for any frequency effects, ensuring that the model cannot maximize perplexity by choosing the more common meaning. In this case, we find that the model leans more toward the Chicago meaning of last_Chicago. Some of the words nearest "last" in the simple vector-to-vector experiment may have been dragged into this open space (e.g. "first" and "only"), while others seem less affected (e.g. "late", "next", "early", and "left"). Similarly, the cities cluster is less homogeneously connected than before. By conflating two unrelated words, it appears we have actually disturbed the semantic structure of the words related to the independent meanings of last_Chicago.

Once we simulated the effect of conflating distinct concepts with a single word and finding that it was detrimental to the word embedding, we wanted to determine whether the MPNNLM would be capable of disambiguating word meanings and modeling multi-prototype words as theorized. For the next stage of the last_Chicago experiment, we attempted to model two prototypes for the word "last_Chicago", but the straightforward approach of initializing the word embedding with two last_Chicago prototypes near the origin among the other words in the vocabulary generally resulted in the two prototypes remaining tightly bound throughout training. In order to train two prototypes for last_Chicago, it required addressing the following open questions:

- Given a prediction, how do we know which prototype for target word $\mathbf{w}_t$ we should select as the truth for training the MPNNLM using backpropagation?
- If generating all prototypes for the vocabulary before training begins doesn't work, what is the best method and schedule for prototype generation?
- If a new prototype is generated after training has begun, can we utilize training information for better placement of new prototypes?

One natural method for disambiguating $\mathbf{w_t}$ given a prediction vector generated by the MPNNLM is to assume that the nearest prototype for $\mathbf{w_t}$ is correct. Alternatively, we can rely on estimates made by the MPNNLM when it generates the probability distribution over the vocabulary for $\mathbf{w_t}$ using Equation 32 (page 82). Although the probability distribution discards information about how much probability mass each prototype contributes, we hypothesized that we could draw the "true" prototype for $\mathbf{w_t}$ by normalizing the individual prototype probabilities for $\mathbf{w_t}$ such that they sum to 1 and drawing one uniformly at random according to these weights. We found that the naive method of choosing the nearest word performed poorly in comparison to this probabilistic method and in settings where a new prototype for a word is generated at the origin or at random, the naive method tends to cause the model heavily favor selection of the original prototype.

As in the initial vector-to-vector experiments, we answered the second question through a number of simple experiments that varied the initial placement of a second prototype for last_Chicago and considered methods for determining when a prototype should be generated. We found that by delaying the generation of the second prototype until the MPNNLM trained over the word "last_Chicago" a few hundred times in the training corpus, the two prototypes were unlikely to converge and obviate the need for multiple prototypes. We found 100 occurrences of the word to be a sufficient amount of time to wait before generating new prototypes, but ran most of our remaining experiments with a delay of 400 training iterations – in later experiments with more than two prototypes for a given word, every existing prototype needs to be selected at least 400 iterations.

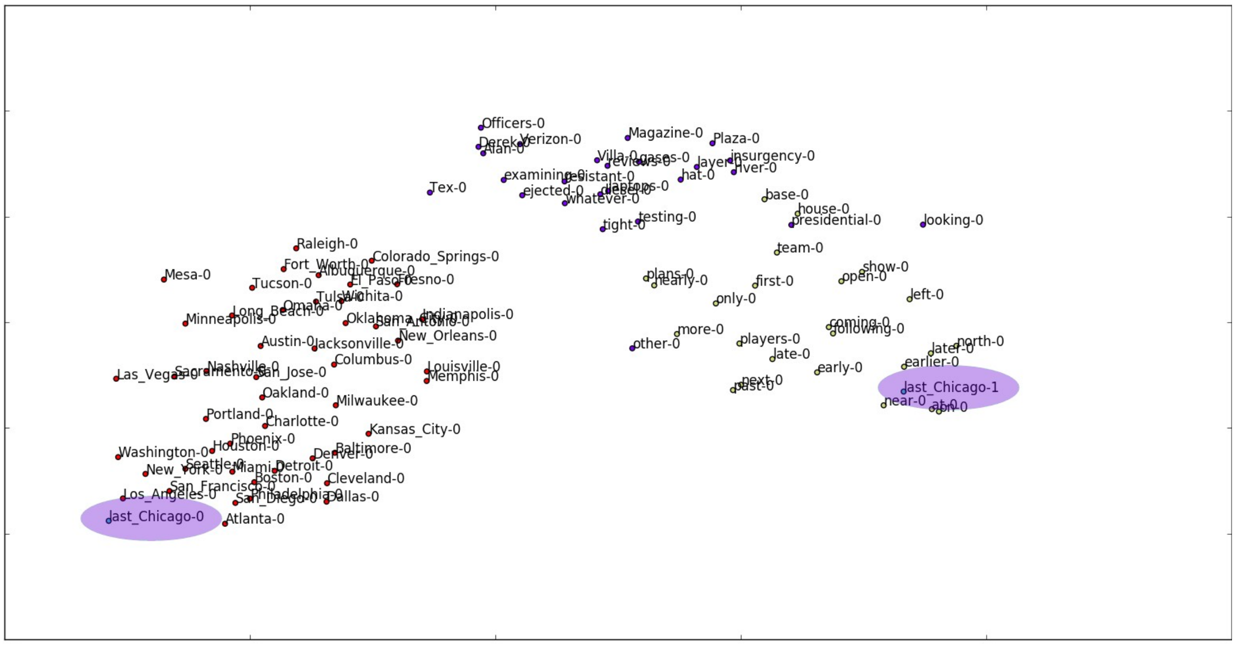


Figure 17. Adding a second prototype vector for the conflated words ("last" and "Chicago") during unsupervised training of the MPNNLM allows the model to correctly identify appropriate meaning clusters for the two word prototypes.

As Figure 17 illustrates, our methodical approach to parameter selection enabled the MPNNLM to identify semantically meaningful locations for the prototypes of last_Chicago that align with our initial unconflated embedding. Although this is a fairly simple experiment, it validates that the MPNNLM is capable of disambiguating word tokens of a single conflated word.

Importantly, the last_Chicago experiment is a useful illustration of how we can tune the MPNNLM parameters for a new text corpus. As we have seen, the MPNNLM has a number of parameters that need to be tuned and it is likely that training the model on a very different text corpus would require a different set of parameters. We do not have a method for algorithmically determining optimal settings of the various parameters discussed, but we believe an automated method could be constructed as follows: randomly choose a set of **k** common words with similar (or even different) frequencies, conflate them in the training corpus, create **k** prototypes for the conflated word, and use a cross-validation experiment to determine optimal parameters for minimizing the affect of the conflation. Presumably, we can identify where the individual words naturally occur in a fully single-prototype setting, so it would make sense to compare the final conflated word embedding with the single-prototype word embedding.

The Multi-Prototype Vector-to-Vector NNLM. The final experiment was to run the MPNNLM according to its initial intention. The initialized word embedding space is contains single-prototype word vectors to start, but all words may generate up to $\mathbf{log_{10}(n)}$ prototypes during training time, as specified in Section 3.5.

During this experiment, we found that the initial method of staggering prototype generation was causing many word prototypes to be generated when they were unnecessary, causing a number of words to have a redundant number of tightly grouped prototypes. We returned to the idea of exploring methods for identifying unrepresented word meanings in text using the MPNNLM's prediction vectors. Our hypothesis is that once the MPNNLM is well-trained, it should be able to predict regions that are related to one of $\mathbf{w_t}$'s meanings. We believe that if many failed predictions tend to cluster in regions without prototype vectors, we can use these as starting points for newly generated prototypes. Similarly, we may find that predictions for a given word are consistently close to existing prototypes and that the generation of a new prototype is unnecessary.

After the backpropagation step is completed for the current word $\mathbf{w_t}$, the MPNNLM will add a new prototype for $\mathbf{w_t}$ if the following criteria are satisfied:

1. The MPNNLM perplexity over a held-out data set must be less than $\mathbf{\hat{P}}$=400.
2. All existing prototypes for $\mathbf{w_t}$ must have been selected at training time at least $\mathbf{\hat{T}}$=400 times.
3. If $\mathbf{w_t}$ occurs $\mathbf{n}$ times in the training corpus, it is limited to $\mathbf{log_{10}(n)}$ prototypes.
4. The new prototype must be generated at least euclidean distance $\mathbf{\hat{D}}$=1 from the nearest existing prototype.

Criteria 1 through 3 are straightforward in light of previous experiments. In order to determine candidate prototype locations based on previous predictions, we found that a simple *agglomerative clustering* technique was very effective. Agglomerative clustering methods generate a hierarchical cluster over the entire set of input vectors based upon a given distance metric. We compared Euclidean distance against the cosine similarity metric due to its previous success at semantic interpretations of word embeddings (Equation 29, page 71). Agglomerative clustering is an iterative process that begins by assigning every input vector its own cluster. As long as multiple clusters remain, distances between the centroids of each cluster are calculated and the two clusters with the smallest distance are merged.

Agglomerative clustering produces a tree with a single root of all of $\mathbf{w_t}$'s previous 100 prediction vectors as members and the leaves are individual prediction vectors. When the process is complete, we navigate the tree from the root and identify the top-most branch that contains a split at least as large as 90% to 10%. This guarantees that we find a large group and we consider the cluster's centroid as the proposed prototype location. Our experiments showed that using cosine distance as the distance metric regularly outperformed Euclidean distance.

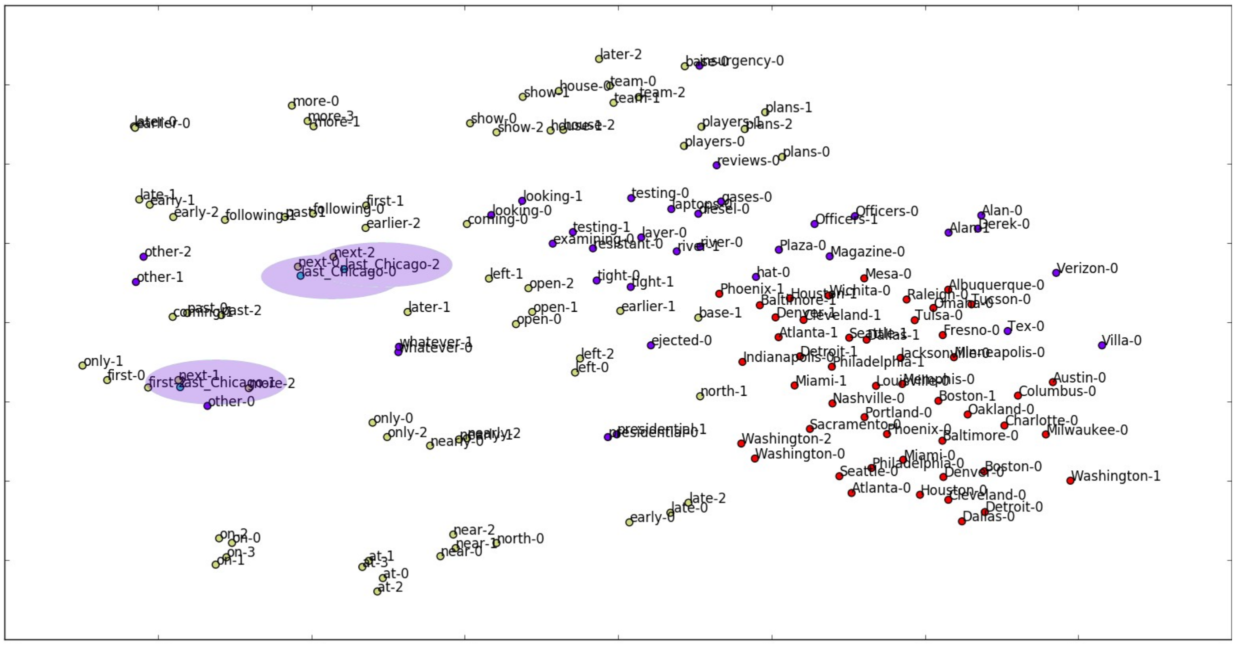


Figure 18. The distribution of multiple prototype vectors per word after training over a 5 million token subset of the Google Gigaword corpus for 4 iterations. Using the same settings as in the last_Chicago experiment results in a less semantically interesting distribution.

After determining functioning rules for generating new word prototypes based upon latent clusters of previous predictions, the final experiment was run over the general Google Gigaword corpus to determine how the model scales. In this case, we did not filter the corpus to a hand-picked subset as we did with the 50Cities corpus. However we did limit the model to 4 iterations of a 5 million token subset of the corpus (closer to 5.31 million including the <seq> and </seq> tokens added for each sentence). Figure 18 illustrates a final embedding of the same words we previously analyzed after training completed. The final parameters tuned to the targeted 50Cities corpus do not appear to translate well, demonstrating the difficulty of designing a system for the training of multi-prototype word embeddings.

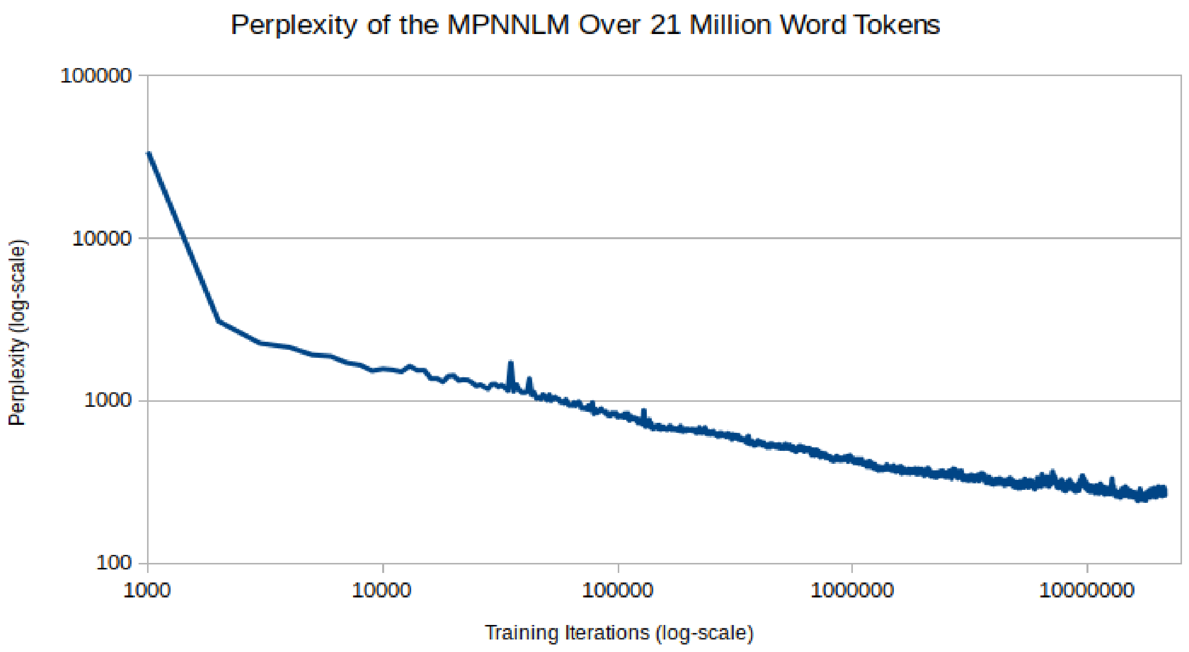


Figure 19. Perplexity of the MPNNLM trained in the fully multi-prototype setting over 21 million word tokens.

Figure 19 shows the MPNNLM's perplexity as it trains over the Google Gigaword corpus. After 21.2 million word tokens, the model achieves a perplexity of approximately 265. Figure 19 uses a log-log scale to allow the reader to more easily identify notable achievements in perplexity and the approximate number of tokens at which these perplexities were achieved (which is more difficult with linear-scaled axes), but this has the effect of appearing that the model has not converged after 21.2 million tokens. We may see improvements of perplexity and small qualitative improvements in our word embedding projections with more training, but Figure 19 implies that it would require orders of magnitude more training.

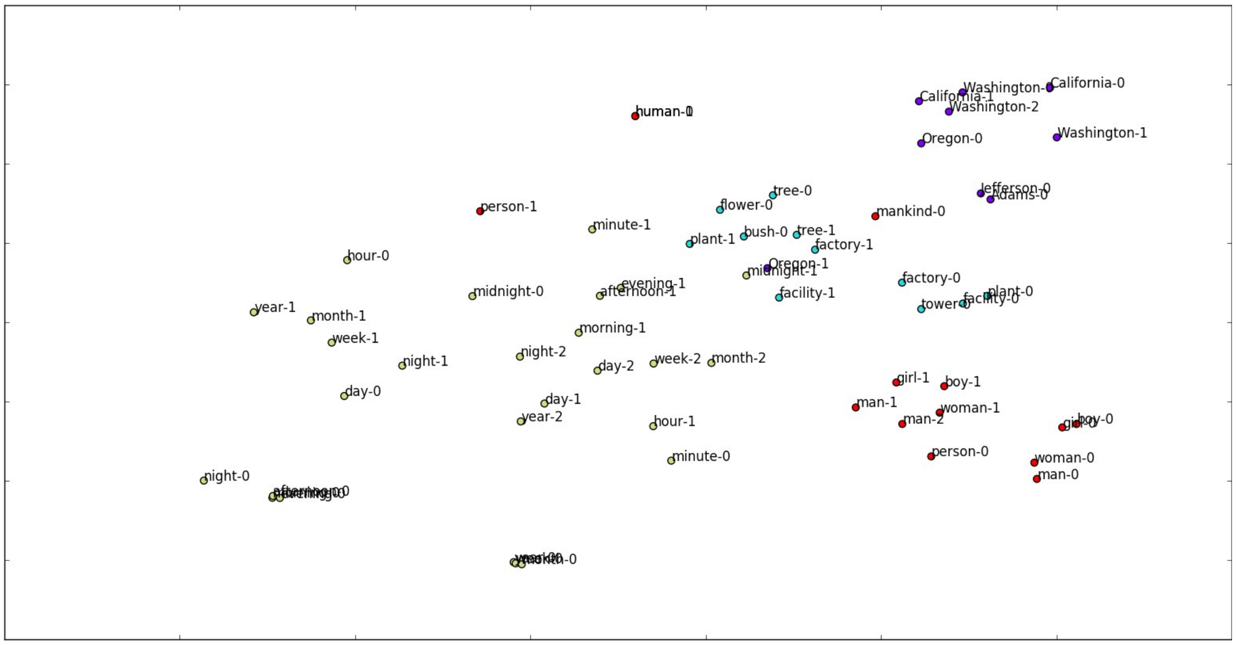


Figure 20. The distribution of various words vectors related to polysemous words ("man", "Washington", "plant", and "day") after training over a 5 million token subset of the Google Gigaword corpus for 4 iterations.

In Figure 20, we use t-SNE to project a number of polysemous words ("man", "Washington", "plant", and "day") along with words related to two of their meanings. As we can see from the t-SNE projection, few polysemous words seem to cluster with related word prototypes. Although the MPNNLM achieves 260 perplexity over this data set, that is not a state-of-the-art performance (state-of-the-art perplexity is typically below 60). We were capable of fully deconflating the meanings of last_Chicago in the previous experiments over the 50Cities corpus, but the same settings of hyperparameters do not naturally extend to the general GigaWord corpus.

3.7 Conclusions

In this chapter, we presented the MPNNLM, a novel machine learning model that can learn an expanded word embedding that contains multiple *prototype vectors* per for polysemous words thanks to a custom loss function designed to incentivize each prototype to represent an independent meaning of the word. Our model is a step toward human-level understanding of language – by modeling explicit representations of distinct meanings of a word, we find that the model can learn to infer which meaning of a word is used in text given the surrounding context and improve the semantic structure of the generated word embedding. Furthermore, the MPNNLM is trained in an unsupervised setting over human-generated text (it does not require a human to disambiguate the meaning of polysemous words in text for it to learn accurate prototypes for each meaning) and is capable of word sense induction, which is a traditionally supervised task. Our work makes the following contributions:

- We share results of a word-conflation experiment with empirical evidence that the single-prototype assumption made by modern neural network language models disrupts a word embedding's ability to distinguish between semantically similar clusters of words.
- We demonstrate a successful vector-to-vector n-gram NNLM with a custom loss function that is capable of learning a semantically meaningful vector for each word in the vocabulary. Notably, this model is a feed-forward network (not recurrent), which reduces training time.
- We demonstrate that this NNLM can be extended to learn a distinct prototype vector for each meaning of a polysemous word.
- We specify a loss function for backpropagation of the MPNNLM that is designed to update the weights of the MPNNLM, the vector representation of the target word's various prototypes, the vector representation of the context words that were given as inputs to the model, and make small improvements to every word in the word embedding.

- The gradient of our loss function directly incentivizes the MPNNLM to improve its performance on both the standard language modeling task as well as unsupervised word sense induction, which is typically a supervised task.
- We demonstrate that a randomized approach to selecting non-target words when calculating the model's loss for each prediction decreases training time without negatively affecting the MPNNLM's ability to generate accurate predictions and intuitive word embeddings.
- We provide a methodological approach for identifying when a new prototype should be generated for a given word, unlike existing methods which assume a fixed number of meanings per word or calculate the number of meanings based upon a word's frequency in the manner of Zipf (1945).
- We specify an agglomerative clustering method over the MPNNLM's previous predictions into the word embedding and find that we can determine when the model is regularly generating predictions for a given target word in a region of the word embedding without a prototype and find that this is a valuable method for identifying when and where to generate new prototypes for existing words while training the MPNNLM.

We began with a goal of observing whether the traditional assumption that a single representation would negatively impact word embeddings. Our controlled experiments demonstrate that neural networks are capable of learning multiple meanings per word and that the single-prototype assumption may have negative impacts on neural network performance and the interpretability of their word embeddings.

In our experiments, we found that our custom neural network framework and MPNNLM implementation share drawbacks with other custom and early neural network frameworks, specifically that they are sensitive to values of the hyperparameters and require a significant amount of manual tuning. The last_Chicago experiment is a useful illustration of how we can tune the MPNNLM parameters for a new text corpus. As previously discussed in this chapter, the MPNNLM has a number of manually-tuned parameters and we find it likely that training the model on a very different text corpus would require different values. We have not identified an optimal method for algorithmically determining the best values for these parameters, but Grid Search, Bayesian Optimization (Močkus, 1975) and kriging (Krige, 1951) are generic approaches that will identify near-optimal values. One could also programmatically score the performance of a model trained with specific parameter settings by using conflation experiments similar to what we have demonstrated. Such an experiment would operate as follows: randomly choose a set of **k** common words, conflate them in the training corpus, create **k** prototypes for the conflated word, and use a cross-validation experiment to determine the optimal parameters for minimizing the effect of the conflation. We can identify where the individual words would have naturally occurred in the single-prototype setting, so a comparison of the final conflated word embedding with the single-prototype word embedding would determine how well the MPNNLM is able to reverse the simulated conflation.

We also expect that an alternative implementation of the MPNNLM in a modern open-source neural network framework such as Torch (Collobert et al., 2011), Tensorflow (Abadi et al., 2016), or Keras (Chollet et al., 2015) would have a number of advantages over our custom neural networks – these frameworks provide improved model and training stability as well as a much larger set of training and analysis tools provided by the open source community. An implementation of the MPNNLM using any of these frameworks would likely provide more stable results and enable more efficient exploration of the MPNNLM's hyperparameter space.

Chapter 4
Conclusions and Future Work

In this work, we presented two novel methods for extracting informative corpus-level statistics to inform a NLP model. We first demonstrated that the critical parameters of the Multinomial Naïve Bayes (MNB) algorithm – the conditional probabilities – are subject to high variance when estimated over a small, supervised training text corpus. Our method, MNB-FM, provides a theoretically motivated approach for accounting for word frequencies measured over a much larger unlabeled corpus in order to tangibly improve the accuracy of MNB's conditional probabilities when they don't accord with the unsupervised corpus. MNB-FM's information theoretic underpinnings consider the expected variance of noisy parameter estimates and adjust the least reliable conditional probabilities the most. The MNB-FM method achieves state-of-the-art performance over two text classification tasks: Sentiment Analysis and Topic Classification. This performance increase effectively improves the amount of information extracted per document, which can be of critical importance in domains where labeling training documents by hand is expensive. For experiments with 100 labeled training documents, MNB-FM is capable of achieving a performance similar to what MNB would get with 1000 labeled documents.

Spurred by the publication of Zhao et al. (2016), we conducted a follow-up examination of our MNB-FM model which provided deeper insights into the model's performance improvements. We found that rare words had a more significant role in the state-of-the-art performance of MNB-FM than previously realized. Due to the long tail of word frequency distributions Zipf (1945), rare words account for a large fraction of probability mass in large text corpora – our analysis in Lucas and Downey (2013) showed that words occurring less than once every 100,000 words accounted for 23% of the MCAT corpus. Many text classification algorithms discard very rare words from the vocabulary because their high variance makes them difficult to model and very rare words by definition have little *individual* impact on classification performance. Our follow-up analysis validates the results of Lucas and Downey (2013), which demonstrated that MNB-FM significantly improved MNB's conditional probability estimates for rare words, and that *in aggregate* these rare words contributed a significant amount of our performance gains. This improvement in estimating conditional probabilities is specifically due to how MNB-FM compares word frequencies within the unlabeled corpus with frequencies in both classes of the labeled corpus while simultaneously accounting for the expected variance of these counts. MNB-FM's variance-minded approach truly enables it to infer more accurate conditional probabilities for the rare words that drive a significant amount of the performance gains that make MNB-FM a state-of-the-art algorithm.

The second contribution we presented was a novel multi-prototype neural network language model (MPNNLM). Many previous attempts at using a neural network language model to generate multiple vectors per word exist and demonstrate that there is an interest in the multi-prototype setting. Proper training of a multi-prototype NNLM should be capable of achieving the state-of-the-art performance of NLP tasks, but in practice multi-prototype models typically have a difficult time outperforming the best single-prototype models (Li and Jurafsky, 2015). The MPNNLM was designed with the primary motivation of unsupervised training of multiple prototype vector representations for polysemous words. Our experimental results motivated the following design decisions within the MPNNLM to address challenges that arise in the multi-prototype setting:

- **New Prototype Generation –** Prototypes are only generated when an unknown meaning is detected and cluster previous predictions of a given word to identify regions that most likely represent an unknown meaning for a word as candidates for new prototype locations.

- **Word Sense Induction –** The MPNNLM must be capable of word sense induction in order to properly train multiple prototypes per word. As such, the MPNNLM can be used for word sense induction on held-out data sets as well.
- **Backpropagation –** We provide methods for updating the neural network weights, input and output word prototype vectors, and all remaining vectors in the corpus during each backpropagation step.
- **Backpropagation Sampling –** We also find that training a random sample of the vocabulary during each iteration of the backpropagation step leads to drastic increases in the without affecting the model's ability to learn a multi-prototype word embedding.

The most exciting result of our MPNNLM research was the successful disambiguation of artificially conflated words in our custom 50Cities data set. However, we found it difficult to fully generalize this result to multiple prototypes for every word in a corpus. This was partly due to the difficulty of identifying correct parameter settings for the general setting – we used a custom neural network framework developed in C++, which gave us a greater flexibility to prototype various components of the MPNNLM that are not typical in standard neural network frameworks (such as the integration of agglomerative clustering methods and integrating a softmax smoothing of Euclidean distances in the cost function). Given the promising results of the current design of the MPNNLM, we expect that implementing the MPNNLM in a modern framework such as Tensorflow (Abadi et al., 2016) would allow for better generalization of the model. Our custom research framework provided great flexibility for prototyping, but we also find it more brittle than modern frameworks when it comes to tuning the parameters of a well-defined neural network.

We also consider other limitations of the MPNNLM which provide directions for further investigation in the future. The cost function of the MPNNLM uses Euclidean distance in the word embedding space to determine a prototype's distance from the output prediction. One may consider using cosine similarity (Equation 29, page 71) in place of Euclidean distance because it has proven to be consistently intuitive in semantic evaluations (Mikolov et al., 2013b, Reisinger and Mooney, 2010). Of course, changing the distance metric has deeper implications in the backpropagation of the MPNNLM – word vectors updated at the output layer of the MPNNLM during backpropagation will not move directly toward or away from the prediction point, but instead rotate around the origin toward (due to the nature of cosine similarity and its gradient). In contrast, backpropagation updates to word prototypes at the input layer will continue to be made according to the gradient of the tanh function and are not limited to hypersphere rotations and may be at odds with the output update methodology.

One final consideration is in investigating the interesting memory-retaining aspects of the model. The MPNNLM relies on its output prediction to determine which prototype of the target word should be used as the input vector for subsequent predictions, allowing the model to aggregate contextual information in the input layer. Models such as the RNNLM (Mikolov et al., 2010) provide insight into how backpropagation can be extended to update the neural network and word vectors in previous time periods, so it may be worth considering methods for extending the backpropagation step of the MPNNLM into previous training tokens. Further, we have not investigated how varying the prototypes of input words may affect the predictions, though we expect this to allow us to both verify the model's performance and to enable a second opportunity for word sense induction. For example, consider the different probability distributions we expect to see given an input context of "The newborn weighed 7 pounds, 3" – we may be able to determine that "pounds" is related to weight and not the British currency by determining which prototype for "pounds" generates a higher probability for the known next word, presumably "ounces".

References

Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

Artstein, Ron, and Massimo Poesio. "Inter-coder agreement for computational linguistics." Computational Linguistics 34.4 (2008): 555-596.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.

Bishop, Christopher M. "Neural networks for pattern recognition." Oxford university press, 1995.

Blitzer, John, Mark Dredze, and Fernando Pereira. "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification." ACL. Vol. 7. 2007.

Brants, Thorsten, and Alex Franz. "Web 1T 5-gram Version 1." (2006).

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." Journal of machine Learning research 3.Jan (2003): 993-1022.

Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised Learning." IEEE Transactions on Neural Networks 20.3 (2009): 542-542.

Chelba, Ciprian, et al. "One billion word benchmark for measuring progress in statistical language modeling." arXiv preprint arXiv:1312.3005 (2013).

Chen, Xinxiong, Zhiyuan Liu, and Maosong Sun. "A unified model for word sense representation and disambiguation." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.

Chollet, François, et al. "Keras." GitHub: https://github.com/fchollet/keras (2015).

Collobert, Ronan, Koray Kavukcuoglu, and Clément Farabet. "Torch7: A matlab-like environment for machine learning." BigLearn, NIPS workshop. Vol. 5. 2011.

Cox, David R. "The regression analysis of binary sequences." Journal of the Royal Statistical Society: Series B (Methodological) 20.2 (1958): 215-232.

Crossley, Scott, Tom Salsbury, and Danielle McNamara. "The development of polysemy and frequency use in English second language speakers." Language Learning 60.3 (2010): 573-605.

Cutting, Douglass R., et al. "Scatter/gather: A cluster-based approach to browsing large document collections." Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1992.

Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of Control, Signals, and Systems (MCSS) 2.4 (1989): 303-314.

Goodman, Joshua T. "A bit of progress in language modeling." Computer Speech & Language 15.4 (2001): 403-434.

Griffiths, Thomas L., and Mark Steyvers. "Finding scientific topics." Proceedings of the National academy of Sciences 101.suppl 1 (2004): 5228-5235.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. "Weka: A machine learning workbench." Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on. IEEE, 1994.

Huang, Eric H., et al. "Improving word representations via global context and multiple word prototypes." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1. Association for Computational Linguistics, 2012.

Jelinek, Fred and Mercer, Robert L.. "Interpolated estimation of Markov source parameters from sparse data." In Proceedings, Workshop on Pattern Recognition in Practice (1980): 381-397.

Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2017-03-20].

Krige, Daniel G. "A statistical approach to some basic mine valuation problems on the Witwatersrand." Journal of the Southern African Institute of Mining and Metallurgy 52.6 (1951): 119-139.

Landauer, Thomas K., and Susan T. Dumais. "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge." Psychological review 104.2 (1997): 211.

Lewis, David D., et al. "Rcv1: A new benchmark collection for text categorization research." The Journal of Machine Learning Research 5 (2004): 361-397.

Li, Jiwei, and Dan Jurafsky. "Do multi-sense embeddings improve natural language understanding?" arXiv preprint arXiv:1506.01070 (2015).

Liu, Yang, et al. "Topical Word Embeddings." AAAI. 2015.

Liu, Xiaodong, et al. "Stochastic answer networks for machine reading comprehension." arXiv preprint arXiv:1712.03556 (2017).

Lucas, Michael, and Doug Downey. "Scaling semi-supervised naive bayes with feature marginals." Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2013.

van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of Machine Learning Research 9.Nov (2008): 2579-2605.

Manning, Christopher D., and Hinrich Schütze. "Foundations of statistical natural language processing." Vol. 999. Cambridge: MIT press, 1999.

McCann, Bryan, et al. "Learned in translation: Contextualized word vectors." Advances in Neural Information Processing Systems. 2017.

Meyerson, Adam. "Online facility location." Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. IEEE, 2001.

Mikolov, Tomas, et al. "Recurrent neural network based language model." Interspeech. Vol. 2. 2010.

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013a).

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2013b.

Mikolov, Tomas, et al. "Learning longer memory in recurrent neural networks." arXiv preprint arXiv:1412.7753 (2014).

Minsky, Marvin, and Seymour A. Papert. Perceptrons: An introduction to computational geometry. MIT press, 2017.

Mnih, Andriy, and Geoffrey E. Hinton. "A scalable hierarchical distributed language model." Advances in neural information processing systems. 2009.

Močkus, Jonas. "On Bayesian methods for seeking the extremum." Optimization Techniques IFIP Technical Conference. Springer, Berlin, Heidelberg, 1975.

Murphy, Gregory. The big book of concepts. MIT press, 2004.

Neelakantan, Arvind, et al. "Efficient non-parametric estimation of multiple embeddings per word in vector space." EMNLP (2014).

Newman, Mark EJ. "Detecting community structure in networks." The European Physical Journal B-Condensed Matter and Complex Systems 38.2 (2004): 321-330.

Newton, I. "De analysi per aequationes numero terminorum infinitas (1669)." London, UK: W. Jones 1711.

Nigam, Kamal, et al. "Text classification from labeled and unlabeled documents using EM." Machine learning 39.2-3 (2000): 103-134.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

Peters, Matthew E., et al. "Semi-supervised sequence tagging with bidirectional language models." arXiv preprint arXiv:1705.00108 (2017).

Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners."

Reisinger, Joseph, and Raymond J. Mooney. "Multi-prototype vector-space models of word meaning." Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2010.

Rosenblatt, Frank. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

Rowe, William D. "Gestalt pattern recognition with arrays of predetermined neural functions." Proceedings of the 1st international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., 1969.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. No. ICS-8506. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Schütze, Hinrich. "Automatic word sense discrimination." Computational linguistics 24.1 (1998): 97-123.

Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47.

Sirinukunwattana, Korsuk, et al. “Gland segmentation in colon histology images: The GLaS challenge contest.” Medical image analysis 35 (2017): 489-502.

Su, Jiang, Jelber S. Shirab, and Stan Matwin. "Large scale text classification using semi-supervised multinomial naive bayes." Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM Neural Networks for Language Modeling." Interspeech. 2012.

U.S. Census Bureau, Population Division. “Annual Estimates of the Resident Population for Incorporated Places of 50,000 or More, Population: April 1, 2010 to July 1, 2015. ” Retrieved from https://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml (May 2016).

“Vector Representations of Words.” TensorFlow, 21 May 2017, http://www.tensorflow.org/tutorials/word2vec

Wikipedia User BruceBlaus. “Multipolar Neuron.” *Wikimedia Commons*, version 2, Wikipedia, 24 May 2017, http://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png.

Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." Neural computation 1.2 (1989): 270-280.

Yang, Yiming, and Xin Liu. "A re-examination of text categorization methods." Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1999.

Zhao, Li, et al. "Semi-supervised multinomial naive bayes for text classification by leveraging word-level statistical constraint." Thirtieth AAAI Conference on Artificial Intelligence. 2016.

Zhu, Xiaojin, and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.

Zipf, George Kingsley. "The meaning-frequency relationship of words." The Journal of general psychology 33.2 (1945): 251-256.

Zou, Will Y., et al. "Bilingual Word Embeddings for Phrase-Based Machine Translation." EMNLP. 2013.