Title
Scalable Text Analysis with Efficient Distributed Word Representation

Author
Xue, Zijun

Publication Date
2020

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Scalable Text Analysis with Efficient Distributed Word Representation

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Zijun Xue

2020

ABSTRACT OF THE DISSERTATION

Scalable Text Analysis with Efficient Distributed Word Representation

by

Zijun Xue
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2020
Professor Junghoo Cho, Chair

The demand for Natural Language Processing has been thriving rapidly due to the various emerging Internet services such as social networks, e-commerce, intelligent assistant, etc. The large volume of natural language data and the new market requirements have brought significant challenges and opportunities for academia and industry. The challenges include (1) efficiently processing a massive volume of data and devising a highly scalable architecture, (2) extending the horizon of the current natural language processing to tackle problems that cannot be handled well. This thesis demonstrates several advanced techniques to tackle the challenges by boosting scalability and efficiency or extending the horizon of existing core NLP techniques on different granularities. This thesis introduces (i) a neural-based lexical simplification algorithm that utilizes contextualized word embedding to capture the meaning more accurately, (ii) a sample-efficient algorithm that solves extreme large-output classification for contrastive representation learning. (iii) a scalable and efficient algorithm for the probabilistic topic model LDA.

First, we propose amplified negative sampling and rigorous mathematical analysis of the famous negative sampling method for the word level distributed representations. Specifically, we distinguish the impact of various loss functions for the contrastive representation learning process. Our theoretical analysis provides a better understanding of a few empirical observations that are well known among practitioners when negative sampling is employed. We also propose the *amplified negative sampling* based on our analysis. It's a simple yet efficient method that can effectively improve the negative sampling efficiency and boost the downstream tasks' performance.

Second, we propose the neural simplicity ranking (NSR) model that utilizes the contextualized word representations in an unconventional way for lexical simplification for the sentence level tasks. This model creatively uses the contextualized word embeddings beyond its original masked language model (MLM) task to identify the relatedness among candidate words and the original words. By redesigning the feature set for the simplicity measure and the ranking scheme, the NSR model achieves new state-of-the-art performance.

Finally, we propose sd-LDA for the paragraph-level tasks, a scalable disk-based LDA model with sparse-prior sampling. The sd-LDA brings a highly optimized disk-based design that reduces the memory consumption by two orders of magnitude, making it possible to do large topic analysis on mobile devices. A sparse prior sampling technique is introduced in this work to accelerate the processing time. The sd-LDA algorithm can effectively reduce the memory consumption and processing time at the same time without loss of performance.

The dissertation of Zijun Xue is approved.

Yingnian Wu

Wei Wang

Carlo Zaniolo

Junghoo Cho, Committee Chair

University of California, Los Angeles

2020

To my parents and Mr. Xu Feng

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would love to thank my advisor, Professor Junghoo Cho. I feel very grateful to spend my Ph.D. journey with him. Over the past years, his encouragement helps me go through numerous challenges. I cannot imagine how I would have finished finish this challenging journey without supports from him. He is always full of new ideas, full of energy, stays positive, and bringing such spirit to me.

I would also love to thank Professor Carlo Zaniolo, Professor Wei Wang, and Professor Yingnian Wu as my committee members and their valuable feedbacks that immensely helped me with my research.

I would love to thank my friends: Mingda Li, Manoj Reddy, Muhao Chen, Ling Ding, Jin Wang, Youfu Li, Jiaqi Gu, Bin Bi, Mohan Yang, Ruirui Li, Zuyu Li, Junheng Hao, Chelsea Ju, Ariyam Das, and Xuelu Chen. We shared happiness and sorrow. We explore, learn, and face challenges together. I learned a lot from you, and the friendship helps me pass numerous difficult times.

I would also love to thank Mr. Xu Feng, who enlightened me and inspired me to face life difficulties.

Chapter 4 of this thesis is based on a collaborated work with Dr. Mingda Li. Zijun contributes to the main theorem and corollary derivations. The ANS was proposed by both of them under the guidance of Professor Junghoo Cho. Mingda contributes to the exploration of the property of amplified negative samplings.

VITA

2008-2012	B.S. Computer Science Department Peking University Beijing, China
2012–2020	Ph.D Student Computer Science Department University of California, Los Angeles Los Angeles, California
2015,2016	Research Intern Google Irvine, US
2017	Research Intern Google Mountain View, US
2018,2019	Data Scientist Intern Intuit Central Data Organization Mountain View, US

PUBLICATIONS

Zijun Xue, Manoj Reddy, Junghoo Cho. "NSR: A Neural Simplicity Ranking Model for Lexical Substitution Ranking" In review by the *EACL 2021*

Mingda Li*, Zijun Xue*, Junghoo Cho. "Amplified Negative Sampling: Improving Contrastive Representation Learning via Sample-Efficient Large-Classifier Training" In review by the *AAAI*

2021

Zijun Xue, Ting-Yu Ko, Neo Yuchen, Ming-Kuang Wu, Chu-cheng Hsieh. “Isa: Intuit Smart Agent, A Neural-Based Agent-Assist Chatbot” Accepted by *ICDM 2018* Demo Session

Zijun Xue, Ruirui Li, Mingda Li. “Recent Progress in Converesational AI.” Accepted by *KDD Conversational AI workshop 2018*

Muhao Chen Yingtao Tian Xuelu Chen Zijun Xue Carlo Zaniolo. “Embedding-based Relation Prediction for Ontology Population” Accepted by *SDM 2018*

Zijun Xue, Junghoo Cho. “sdLDA: Scalable Disk-based LDA with Sparse Prior Sampling” Unpublished draft

Zijun Xue “Scalable Text Analysis” Accepted by *WSDM 2017* Doctoral Consortium

Zijun Xue, Junghoo Cho. “Disk-based LDA” *UCLA CSD Technical Report 2014*

Junjie Yao, Bin Cui, Zijun Xue, Qinyun Liu . “Provenance-based Indexing Support in Micro-blog Platforms” Accepted by *ICDE 2012*

Zijun Xue, Junjie Yao, Bin Cui “Temporal Provenance Discovery in Micro-blog Systems” Accepted by *SIGMOD 2012*, Poster

CHAPTER 1

Introduction

1.1 Motivations

With the rapid development of the Internet and the adoption of intelligent devices, natural language processing has been a flourishing area. Massive text data brings new challenges and demands for novel and efficient algorithms. Neural networks and Bayesian networks have played crucial roles in accommodating these challenges. Distributed word representation started a new era for neural-based natural language processing. Prominent works, such as word2vec (MCC13a), seq2seq, transformer, BERT (DCL18a) and GPT, have revolutionized the natural language processing area by achieving new state-of-the-art results in tasks such as translation(WSC16), POS tagging (ABV18), constituent parsing (KK18), or even surpassing human beings in tasks such as question answering (RJL18). Meanwhile, Bayesian networks are still playing an essential role in topic analysis in domains such as marketing, sociology, political science, and the digital humanities; (BNJ03a)

Although these exciting breakthroughs bring plenty of opportunities, severe challenges coming together with these new technologies. The challenges include:

- **New text processing tasks** With the development of natural language processing tasks, NLP tasks' horizon has been significantly expanded. Traditional NLP tasks include POS tagging, parsing, translation, etc. More complex demands have attracted attention from industry and academia in recent decades, such as lexical simplification, text style transfer, and grammatical correction.

  One of the prominent examples is the lexical simplification problem. It has many applications but cannot be solved using a conventional model or simply solved by any existing

neural-based framework. The broad spectrum of applications includes second language education, dyslexia treatment, or pedagogical education (YPD10). Moreover, the lexical simplification cannot be simply solved using any sequence-to-sequence model or applying a language model directly. The task of lexical simplification brings a new dimension of text transformation: simplicity of the text.

Currently, existing models utilize static word embeddings and predefined lexical features to solve this problem. (PS17b; MX18). However, the validity of existing features and the usage of word embeddings should be revisited. Moreover, a proper ranking algorithm will also substantially improve the performance of the lexical simplification problem.

- **Theoretical foundations of negative sampling in representation learning algorithm** The negative sampling technique in representation learning has been widely adopted in various areas such as entity representation (GL16a), word representation (MCC13a) and knowledge representation. The distributed representation has become an indispensable foundation for the neural network in those domains. The foundation of negative sampling can be traced back to noise-contrastive estimation(NCE) (GH10), and later this technique was ported into the NLP domain by (MT12). The above work provides the intuition of negative sampling. However, how do different components affect the learning result of negative sampling? There are a few works that discussed the properties of negative sampling. For example, (MC18) studied the statistical property of the conditional NCE, and (AKK19) attempts to analyze the performance error bound between the representation learning and the downstream tasks for a specific setting.

  Moreover, the training cost of negative sampling is quite expensive as well. Empirical evidence shows larger sample size can lead to better downstream task performances (MCC13a). A rigorous analysis can better explain such a phenomenon and propose a better optimization based on the theory.

- **Scalable text analysis for ultra-large corpus** Statistical topic models were initially developed for analyzing textual datasets, but they are now being used for a wide range of applications, from mining text articles, analyzing image contents to extracting meaningful

patterns from genome sequences. "Topics" in topic models originally meant the topical categories that a particular text or word belongs to. In general, topics are *latent causes* or *hidden variables* behind any observed data, which can be utilized to identify implicit correlations among observed data (BNJ03b). The rise of ultra-large datasets in the Internet era, including online customer product-purchase history, social-network user-interaction trace, and large-scale historical news-article archive, fueled the increasing demand and development of various statistical topic models. Unfortunately, the ever-growing data size makes scalability a major bottleneck to even wider adoption of topic models.

We take a new approach to improving LDA's scalability by focusing on the algorithm's *memory footprint*. By utilizing the sparsity of data distribution and locality of data access (YMM09), we can effectively design a highly efficient algorithm with compact storage consumption without sacrifice too much performance. Thus, we can attain a more scalable and efficient topic model.

1.2 Contributions

The contributions of this thesis are summarized as follows:

- We explore a novel lexical simplification task on the sentence level task that unconventionally utilizes the neural-based NLP techniques. We (1) carefully analyze, streamline, and improve existing features for lexical simplification and (2) propose a new neural pair-wise candidate-word ranking model by adopting the models known to enhance performance in other application domains to the lexical simplification task. We also utilize the contextualized word embeddings as a feature to evaluate the relatedness among words. We demonstrate the effectiveness of our method on the SemEval 2012 task and our method achieves the new state-of-the-art result.

- On the word-level, we conduct a rigorous analysis of the core technique in representation learning: negative sampling. Our analysis can easily explain the reason for a lot of empirical phenomena such as the degenerated model without any negative samples. Our analysis also

distinguishes the effect of the different loss functions. Our study also proposes an efficient sampling method: *amplified negative sampling*, which can effectively reduce the sampling overhead and improve the convergence speed.

- On the paragraph level, we propose a highly scalable and efficient topic modeling algorithm: sdLDA. We present the first disk-based LDA algorithm that utilizes the plentiful disk space during the LDA inferencing process to significantly reduce the algorithm's memory footprint. We propose various optimizations for the LDA inferencing algorithm so that our disk-based algorithm exhibits the same or even better running-time performance compared to in-memory algorithms. We conduct extensive experiments demonstrating the effectiveness of our proposed techniques. Again, our proposed approach is compatible with existing LDA optimization techniques and can be utilized *in addition to these existing techniques*. Moreover, it can be adopted as a single-node LDA implementation under a distributed setting to improve its scaling property further.

1.3 Thesis Outline

In this dissertation, we present three works on the natural language process with different granularity. We present three representative works in this thesis. Firstly, we presente how to utilize a deep neural network technique to solve an unconventional natural language processing problem, lexical simplification. In Chapter 2, we demonstrate neural simplicity ranking(NSR) model that adopts a pairwise neural ranking function. It utilizes different features to differentiate the simplicity and relatedness among candidate words and target words. By adopting a highway network and contextualized word embeddings, we demonstrate the state-of-the-art performance in SemEval 2012 dataset.

Secondly, we make a rigorous analysis of the core technique: negative sampling in representation learning in Chapter 3. We prove the convergence point of negative sampling in the training phase for different loss functions and get an analytical formula to represent the relationship between negative sampling and convergence points under different loss functions. Based on the theoretical result, we also propose the amplified negative sampling(ANS), which can effectively

reduce the training time without losing accuracy.

Lastly, we present an efficient, scalable topic modeling algorithm in Chapter 4. It is a disk-based latent Dirichlet allocation model with a sparse prior sampling technique. We first conduct a thorough analysis of the memory access pattern in LDA model. Then we present the disk-based LDA and the sparse sampling technique. We demonstrate the sdLDA reduces the memory consumption by 100x less than the conventional LDA without losing accuracy and efficiency. We conclude the dissertation and discuss the future directions in Chapter 5.

CHAPTER 2

NSR: A Neural Simplicity Ranking Model for Lexical Substitution Ranking

Lexical Simplification (LS) aims to replace complex words or phrases in a sentence with simpler alternatives and has abundant applications. The simplicity of a word is a core concept in lexical simplification ranking. In this work, we claim that a word's frequency is *the* key signal for its simplicity; when a word's frequency is measured after appropriate morphological transformations, its frequency alone works as a very strong indicator of its simplicity, unnecessiating other features commonly used in the literature, such as word length and syllable count. We evaluate the existing system's feature set based on this perspective and propose new frequency-based features for lexical simplification. We also propose a new neural pairwise-ranking method that enhances the existing state-of-the-art ranking model. We evaluate the effectiveness of our methods on the SemEval 2012 lexical simplification task and achieve a new state-of-the-art result of 67.54 (P@1 measure) and 0.72 (Pearson correlation).

2.1 Introduction

Text Simplification is a long-standing topic in natural language processing that aims to simplify the text by reducing the reading and grammatical complexity while keeping sentences' meaning unmodified (DT98). Lexical simplification restricts the scope to the lexical replacement, which aims to replace difficult words with simpler or more frequently used ones. It has a broad spectrum of applications such as second language education, dyslexia treatment, or pedagogical education (YPD10). Normally, lexical simplification has four steps: complex word identification (CWI), substitution generation (SG), substitution selection (SS), substitution ranking (SR). In this paper,

we mainly focus on the substitution-ranking phase.

In recent years, lexical substitution ranking utilizes the feature-based learning model to rank candidate substitution words. Through trial and error, researchers have identified several important features that are effective in the substitution ranking, including word frequency, length, and language model probability to predict the ranking score of given words. (HMK14; GS15; PS16; PS17a; MX18; QLZ19) In this paper, we (1) carefully analyze, streamline, and improve existing features for lexical simplification and (2) propose a new neural pair-wise candidate-word ranking model by adopting the models known to improve performance in other application domains to the lexical simplification task.

Simplicity and relatedness are two core concepts in substitution ranking. Biran (BBE11) introduced a simplicity metric and pioneered a path of utilizing multiple features to determine the simplicity. Among all features, word frequency is known to be the most effective feature for predicting the simplicity of a word (SJM12a; PS17a; MX18), but many other features, including word length and syllable counts, are also believed to be effective features and are widely used by existing systems. In this paper, we argue that the word frequency is *the* feature for simplicity prediction, as long as we apply appropriate transformations to the candidate word before frequency counting. In particular, we introduce two new frequency-based features, (1) morphologically-adjusted word frequency and (2) phrase bottleneck frequency, and show that once these two features are added, other existing features like word length do not provide any additional signal for the word simplicity prediction — in fact, adding them *deteriorates* the prediction accuracy!

Relatedness is another key concept for lexical simplification, which captures the relevance of a candidate word to the particular context in which the original word appeared. In existing systems, relatedness is commonly measured using standard language models (HMK14) and word embedding cosine similarity scores (GS15). In this paper, given the enormous success of the context-sensitive word embedding in many NLP tasks, we evaluate the use of BERT as a potential feature for measuring relatedness. In particular, we propose *weighted BERT* as a minimal adjustment to the standard BERT embedding that customizes it to the task of lexical simplification with minimal computational and training overhead.

We also propose a new neural candidate-word ranking model by (1) incorporating the highway network to the standard feed-forward network used by the current state-of-the-art system (MX18; PS17b) and (2) using a ranking metric and a training loss function more appropriate for the task at hand. Combined together, our proposed methods achieve a new state-of-the-art result of 67.54 (Precision@1) and 0.72 (Pearson correlation) when evaluated on the SemEval 2012 lexical simplification task.

In summary, we make the following contributions in this paper:

- We carefully analyze the commonly used features for lexical simplification and propose new frequency features that provide a much more reliable signal of simplicity for lexical simplification ranking. We also identify and eliminate several features that are of little help through extensive experiments.
- We introduce BERT-based features for relatedness measures and design several models to test out the most effective way to utilize the BERT embedding similarities.
- We improve the pairwise neural ranking model by (1) replacing the rank difference measure with the binary (+1/-1) measure together with a Hinge loss function and (2) replacing the feed-forward neural network with the highway network.
- We perform extensive experiments with SemEval 2012 lexical simplification task and show that our methods achieve the new state-of-the-art results.

The rest of the paper is organized as below. In section 2, the fundamental concepts in lexical simplification tasks are briefly reviewed. Then, the existing problem of features in the lexical simplification system is analyzed in section 3. In section 4, a new neural-based simplicity ranking model will be introduced with a simplicity-related feature set. Section 5 shows the experimental result of the new lexical simplification system's performance and the ablation study result of the discussed features. Related works are reviewed in section 6. Section 7 concludes this paper and talks about potential further directions.

2.2 Fundamentals

As we briefly stated in the introduction, lexical simplification has four steps: complex word identification, substitution generation, substitution selection, and substitution ranking. Among the four, this paper focuses on substitution ranking. In this section, we briefly introduce the key concepts for substitution ranking.

2.2.1 Simplicity and Relatedness

In ranking candidate words for substitution, two key aspects of the candidate word should be considered: (1) Is the word *simpler* than the original word? (2) Is the word closely *related* to the given context?

Simplicity Simplicity indicates how easily a word can be recognized by a language speaker. Since the primary goal of lexical simplification is to rewrite the original text with simpler words, simplicity is the key concept that should be considered in ranking candidate words.

Relatedness The relatedness of a candidate word in a sentence is a direct measure of the level of the preservation of the exact meaning of the original word. This is a vital metric since we will not admit a simplification is valid if the original meaning is distorted. The relatedness might be restricted both semantically or syntactically.

Note that in replacing an original word, we want a simpler *and* related word. Thus, these two concepts are frequently utilized by the research community for the substitution ranking task (PS17a; QLZ19).

2.2.2 Commonly-Used Word Features

Substitution ranking evaluates the simplicity and the relatedness of a candidate word based on a set of word features. Through extensive experiments, researchers have suggested that the following features are useful in measuring the simplicity and relatedness of a word.

Frequency Frequency has been suggested as the most effective feature for measuring the simplicity of a word, based on the intuition that the more frequently a word is used, the more familiar

it is to readers (PS17a). In the next section, we explain how we can further improve the effectiveness of frequency as the simplicity feature by considering all morphological variations of a word in counting its frequency.

Word Length and Syllable Count A Word's length (in terms of the number of alphabets) and its syllable count have been suggested as a feature for measuring simplicity and are used in many existing systems (PS17a). The usefulness of this feature is questionable, however, because the length of a word does not necessarily reflect the cognitive difficulty. This has been partly confirmed by (MX18) who showed that 21% of the word length's information is misleading, meaning that a shorter word can be more complex. In our experiment section, we investigate whether word length is indeed a useful feature for simplicity.

Simple Wikipedia Frequency This feature was introduced in the (PN15), which computes the relative frequency of a particular word in simple Wikipedia compared with normal Wikipedia based on the intuition that if a word is more frequently used in the simple Wikipedia than in the general Wikipedia, the word is likely to be a simpler word.

Word Complexity Lexicon The complexity lexicon was introduced in (MX18). This is a manually annotated dictionary which includes 15,000 words with the complexity score by the ESL readers. Every word in this lexicon has a complexity score and this score has been proven effective for lexical simplicity ranking in various experiments (MX18).

Language Model The language model can compute the probability of, say, the 5-token sequence $t_1t_2wt_4t_5$, given the surrounding context $t_1t_2t_4t_5$ and the candidate word w. Intuitively, this probability measures the appropriateness of a word for the given context and can be a useful feature for measuring the relatedness of the word; a low probability suggests some semantic or grammatical violation and low relatedness. (PS16) used the language model for substitution ranking by training it on the SubIMDB dataset.

Word2vec Similarity Word2vec produces a vector embedding of every word so that the embedded vector distance between two words captures their semantic and syntactic distance. Given this, they can be useful in measuring the relatedness of the original word and the candidate word and is used by many existing systems (GS15).

Other Features There are also other features that have been frequently used such as PPDB (PN15). However, not all of these features are significant in the lexical ranking process. We test the effectiveness of the above-mentioned features in section 5.

2.3 Our New Features

In this section, we introduce the new features that we propose for lexical simplification.

2.3.1 Morphologically-Adjusted Word Frequency

A common intuition behind the frequency feature is that the more frequently a word is used, the more familiar it is to readers (PS17a). However, we believe a more accurate description should be: "The more frequently a word is used, the more familiar *itself and its morphological variations* are to readers".

For example, consider "foolishness" and "folly." The word foolishness has a similar frequency as the word folly in the Google 5gram corpus, but most English speakers agree that foolishness is a simpler word than folly (MX18). Part of the reason why "foolishness" is considered cognitively simpler is because its morphological variation "foolish" is a significantly more frequent word than "folly." Because of English speakers' frequent exposure to foolish, its morphological variation foolishness is viewed as an easier word than its frequency count may indicate.

The morphological variation is a prevalent phenomenon in the vocabulary. Generating a derived word from a root word by adding a specific prefix or suffix is common. For example, "beautiful" is derived from "beauty" and "unnecessary" can be derived from "necessary". Unfortunately, current systems do handle this issue carefully, counting every word's frequency independently, ignoring the fact that some words are morphological variations of others.

As a systematic treatment of the morphological variations of a word, we "aggregate" the frequency counts of the words that are morphological variations of each other.

More precisely, assume a stemming function $S(w)$ that will return the stem of the word w. Let F_w be the standard raw frequency of word w (without considering its morphological variations).

Let $F_{S(w)}$ be the frequency of the words that share the same stem $S(w)$ as w. Then we define the morphologically-adjusted frequency count of w as a linear combination of these two frequencies.

$$F_{aw} = (1-\beta) * F_w + \beta \times F_{S(w)} \tag{2.1}$$

Three different strategies for computing the $F_{S(w)}$ have been explored.

Sum For the summation strategy, the aggregated frequency of a morphological word family is defined as the summation of all the words' frequency.

$$F_{S(w)} = \sum_{w':S(w')=S(w)} F_{w'} \tag{2.2}$$

Max For the max strategy, the aggregated frequency is defined as the largest frequency among the words which share the same stem.

$$F_{S(w)} = \max_{w':S(w')=S(w)} F_{w'} \tag{2.3}$$

Average For the average strategy, the aggregated frequency is the average frequency of all the words' that share the same stem.

$$F_{S(w)} = \frac{\sum_{w':S(w')=S(w)} F_{w'}}{|\{w'|S(w') = S(w)\}|} \tag{2.4}$$

Later in our experiment section, we find that the average strategy performs the best out of the above three strategies.

2.3.2 Phrase Bottleneck Frequency

Another common problem of the raw frequency feature is dealing with multi-word substitution candidates. Existing systems use the same n-gram frequency both for single-word and multi-word candidates. However, this frequency value decreases exponentially with the length of a candidate phrase, giving a significant disadvantage to multi-word candidates.

For example, consider the phrase "take away." Since both "take" and "away" are very frequent words, the phrase "take away" is also considered cognitively simple. However, the n-gram frequency of "take away" is significantly smaller than either "take" or "away", and thus, does not reflect the true cognitive simplicity of the phrase.

In addressing this issue, we conjecture that the cognitive simplicity of a multi-word phrase should be measure by its "bottleneck" frequency: the frequency of the least frequent token in a phrase. This metric is by no means perfect but it is a more reasonable metric compared to the n-gram frequency used by existing systems.

More precisely, we define the phrase bottleneck frequency, F_{phr}, for a phrase $phr = w_1, w_2, ..., w_L$ (where w_i's are the words within the phrase) as

$$F_{phr} = \min_{w_i \in phr} F_{w_i} \tag{2.5}$$

2.3.3 Weighted BERT

As we mentioned in the earlier section, word2vec is a widely-used feature to measure the relatedness of a candidate word. But we conjecture that its effectiveness is limited because word2vec embedding is *context independent*. Given a word w, the same embedding is always produced, regardless of the context in which w appears. This can be problematic since a word may take a completely different meaning depending on its context.

Recently, a number of *context-senstive* embedding methods have been proposed, such as ELMO (PNI18) and BERT (DCL18b). For example, BERT produces a context-sensitive word embedding by training on the masked language modeling (MLM) objective. It utilizes a *cloze* task which is aiming at recovering a word given its context on its left and right. That is, given the input sentence $S = \{t_1, t_2, ..., t_K\}$, BERT generates embedding e_i for each token t_i, so that the probability that the token t_i appears in the context $\{t_1, ..., t_{k-1}, t_{k+1}, ..., t_K\}$ is the highest.

Given its success for a wide range of NLP tasks and its context-sensitive nature, we examine whether BERT embedding will be useful for substitution ranking task. As BERT is a stacked structure with 12 layers (DCL18b), the last four layers of the transformer hidden vector are utilized

as the contextualized embeddings in our study.

Cosine Similarity Perhaps, the most direct way to create a relatedness feature based on BERT embedding is to compute the cosine similarity between the BERT embeddings of the candidate word and the original word. That is, let E_c and E_o be the BERT embedding of the candidate and the original words, respectively. Then we use the cosine similarity between E_c and E_o as a relatedness feature of the candidate word:

$$F_{BERT} = cos(E_c, E_o) \tag{2.6}$$

Weighted BERT Embedding

Surprisingly, when we tested this simple cosine similarity in our experiments, we found that BERT embedding does not improve the overall performance of the system compared to the word2vec embedding. Through the careful investigation of this result, we concluded that the unexpected low performance of BERT is because it predicts the word *too accurately*. Since BERT tries to predict the exact word that appeared in the context, it often prefers a more complex word over a simpler word as long as the word is more appropriate in the given context. While this is great for the general task of word prediction, it is a violation of our goal of lexical simplification. To be effective for our task, therefore, we need to preserve the "semantic relatedness dimensions" in the BERT embedding, while discarding it "contextual complexity dimensions."

To implement the idea of "discarding dimensions" irrelevant to lexical simplification, we introduce a weight vector W. That is, let W be a vector whose dimension is the same as the dimension of the BERT embedding. Then, before we compute the cosine similarity, we perform a element-wise multiplication between W and E_c, $W \odot E_c$, and W and E_o, $W \odot E_o$, as follows:

$$F_{BERT} = cos(W \odot E_c, W \odot E_o) \tag{2.7}$$

Note that the weight vector W controls the contribution of a particular dimension of the BERT embedding to the final cosine similarity value. When W's value is zero for a particular dimension, the dimension is "ignored" in similarity computation. The hope is that we can find an appropriate

weight vector W, whose values are high for the dimensions relevant to lexical simplification and low for irrelevant dimensions. In the next section, we describe the neural simplicity ranking model that is used to compute the pair-wise ranking value between two candidate words from the candidate words' feature values. During the training of this network, the weight vector W is trained as well to maximize its prediction accuracy.

2.4 Neural Simplicity Ranking Model

The feature values described in earlier sections need to be combined to produce the final ranking value for each candidate word.

Pair-wise neural ranking model In this paper, we adopt the state-of-the-art neural-based pairwise ranking architecture with the Gaussian Binning (PS17b; MX18) as our backbone model. The model structure is depicted in Figure 2.1.

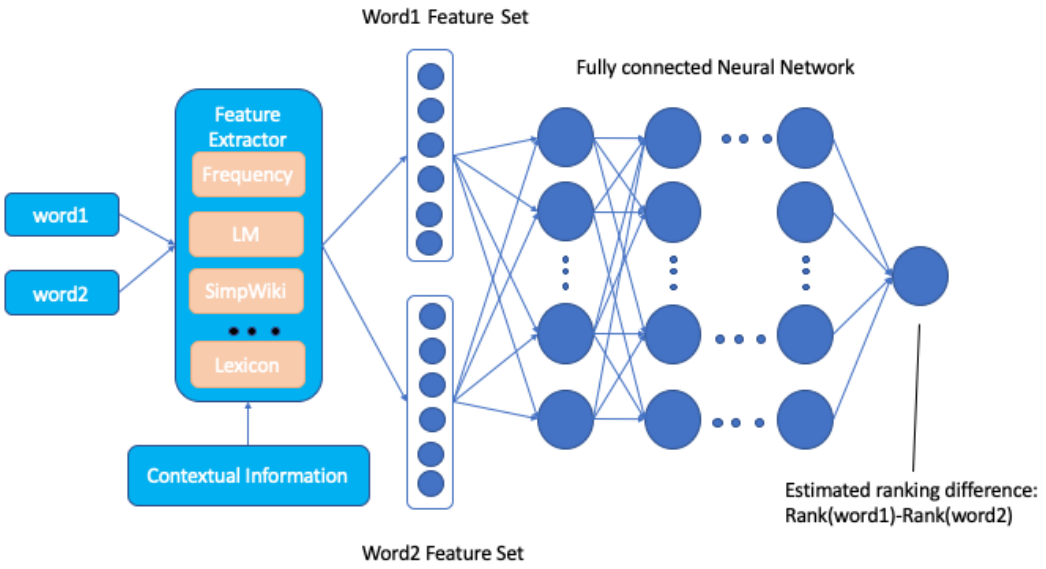


Figure 2.1: The Neural-based Pairwise Word Ranking Model

The neural pair-wise ranking model will generate a pairwise score for two candidate words to predict the relative rank of the pair of substitution candidates. The input of this model are two feature vectors that are generated by a feature extraction function from two substitution candidate

words, and the output will be a single number which predicts the rank difference of two candidate words.

The ranking network consists of 6 layers: one input layer, four intermediate layers, and one output layer. The input layer takes v_1 and v_2 as the input. Each intermediate layer has 8 neurons. The output layer is a single neuron. Each intermediate layer's activation function is the tanh function and there is a dropout layer before each intermediate layer.

In our work, we extend this state-of-the-art backbone model by making the following extensions.

Highway Network vs. FNN Instead of a simple feed-forward neural network (FNN) used in Figure 2.1, we adopt a highway network (SGS15) as part of the ranking model. This is because a highway network, a generalization of a ResNet, is known to produce better results than FNN, partly because its training suffers less from the vanishing-gradient problem than FNN, among many other reasons (SGS15).

Absolute ranking difference vs Order difference In the existing pairwise ranking model, the problem is modeled as a regression problem, which predicts the difference of the ranking positions of the two given candidates.

However, predicting this difference has one critical problem: the pair-wise ranking value that we try to predict using the pair-wise ranking model depends not only on the two candidate words themselves but also on what other words are included in the candidate list. For example, consider a ranked list of three candidate words, c_1, c_2, and c_3. The ranking difference between c_1 and c_3 is 1 for the current list. If c_2 is removed from the list, however, the ranking difference becomes 1. That is, the pair-wise ranking difference value that we need to predict cannot be computed just by looking at the two candidate pairs. We have to consider all candidate words in the list to compute the difference which is clearly impossible given our pair-wise ranking model.

To avoid this problem, we simplify the prediction task as a pair-wise order prediction (represented as +1/-1 labels for the order of two candidates). Since the +1/-1 label is stable regardless of other candidate words, it is much more appropriate for our pair-wise ranking model.

Hinge Loss vs MSE Loss Given the above simplification, we also conjecture that a hinge loss

is a better loss function for the training of the neural network, compared to the MSE loss function used in the previous model (MX18). As long as the sign of our prediction is correct, our network is doing a good job, regardless of how large or small the predicted value is. Hinge loss adjusts a network only if the predicted sign is incorrect, while MSE loss adjusts the network if there is any deviation from the true label, even if the sign is correct. This conjecture is evaluated in our experiment section as well by comparing the overall model performance between MSE and hinge losses.

2.5 Experiment

In the experiment section, we will use experiments to demonstrate (1) the effectiveness of our model in comparison with other existing models to prove the effectiveness of our ideas. We demonstrate our method gains the state-of-the-art result in the lexical simplification ranking result; (2) the ablation study of the isolated features to prove their effectiveness.

Method	**P@1**	**Pearson**	**avg. P@1**	**avg. Pearson**
Biran et al.(**(year?)**)†	51.3	0.505	—	—
Jauhar, Specia(**(year?)**) †	60.2	0.575	—	—
Kajiwara et al(**?**) †	60.4	0.649	—	—
Horn et al.(**(year?)**) †	63.9	0.673	—	—
Glavas, Stajner(**(year?)**) †	63.2	0.644	—	—
Boundry Ranker(**(year?)**) †	65.3	0.677	—	—
Neural Ranking(**(year?)**)	65.02	0.687	63.14	0.665
NRR(**(year?)**)	65.67	0.708	65.08	0.703
Full NSR(ours)	**67.54**	**0.720**	**67.07**	**0.719**

Table 2.1: Substitution ranking performance comparison with existing models. "†" indicates the results are obtained from (MX18). Hyphens denote not available. Top results are boldfaced

2.5.1 Experiment Settings

Task & Dataset The SemEval 12 Substitution Ranking task (SJM12b) is conducted in this section. In this task, a set of candidate words will be given to replace a complex target word. Given the golden candidate ranking, the algorithm targets for generating the best ranking result.

This SemEval 12 dataset consists of 2000 sentences for a total of 200 words. Each word has

10 sentences to demonstrate the various meanings of a word. For each sentence, a target word is labeled to be replaced. A golden ranking of all the candidates is given. The task is to rank the given candidates' list and try to recover the original golden ranking. The dataset is split into a training set that contains 300 sentences and a testing dataset which contains 1700 sentences.

Neural Simplicity Ranking Model Settings Based on our analysis, the golden settings of our final model uses the pair-wise neural ranking model (PS17b) with Gaussian Binning (MX18). It adopts six features: (1) frequency with morphological variations on average strategy, (2) adjusted phrase frequency, (3) simple Wikipedia frequency ratio, (4) n-gram language model probability, (5) BERT embedding, and (6) word complexity lexicon.

Evaluation metrics For each setting, the experiment is executed 100 times. The best result and the average performance among the 100 executions are recorded.

Precision@1 This metric is the top word precision of the ranking result. It indicates how many times we get the top candidate word ranked correctly. Since in the real-world system, we usually replace the target word with the top result. This metric can be treated as a real-world system accuracy.

Pearson correlation This metric is evaluating the golden candidate words' ranking's numerical correlation with the predicted candidate words' ranking. This metric indicates the overall ranking performance. The range of this metric is between -1 and 1. The closer to 1 this number is, the better the ranking result is.

Hyper Parameters In the experiment, the neural-based pair-wise ranking model with the Gaussian binning strategy is adopted as mentioned in (PS17b; MX18). The learning rate is set as 0.0025. The dropout rate is set as 0.2. The training iteration is set as 100 epochs. In this paper, we utilize the pretrained 'bert-base-uncased' model which has 12-layer, 768-hidden, 12-heads, and 110M parameters.

2.5.2 All-in-one Model Comparison

We compare our results with existing systems. We mainly compare our method with two other models that utilize the neural-based substitution ranking model. The neural-based lexical simplifi-

cation model was introduced in (PS17b) and it is used as the baseline system in our experiment. The Neural Readability Ranking(NRR) model was introduced by (MX18), which is the state-of-the-art system in substitution ranking. This NRR model will be another system for comparison in our experiment. Notable previous systems are also listed as references.

In this experiment, our model performs best among all substitution ranking methods and achieves the new state-of-the-art result in the substitution ranking(Semeval 12) task. Our method achieves a 2.8% improvement on the best result in 100 runs and a 3% improvement on average performance in 100 runs.

2.5.3 Ablation Studies

In this section, we demonstrate the effectiveness of our feature improvement and model structure modification. The experiment removes one key modification each time and shows the top-1 item precision and Pearson correlation score. The all-in-one model's performance is also displayed for comparison.

The ablation study can be categorized into two sets: the feature test and the model structure test. For the feature test, the morphological frequency, phrase frequency, and the Bert embedding are tested with normal FNN architecture and MSE loss. For the model structure test, the highway network and hinge loss are removed individually on the NSR model. As shown in Table 2, removing any new features/components reduces the overall performance.

NSR w/o features	**avg. P@1**	**avg. Pearson**
Freq˙avg	66.17	0.714
Phrase-freq	66.34	0.716
Bert_Embedding	66.30	0.719
highway & hinge	66.39	0.719
hinge loss	66.49	0.716
highway net	66.82	0.719
NSR model	**67.07**	**0.719**

Table 2.2: Substitution ranking performance comparison by eliminating features/components from the NSR Model

2.5.4 Miscellaneous Test

We make the feature selection by testing out the existing commonly used features with NSR with FNN and MSE loss.

Feature Removal Test In this test, the features in the NSR model are removed individually. As showed in Table 3, the word lexicon feature shows the biggest decrease in the performance if it is removed. The average P@1 performance is decreased from 66.39 to 65.26 and the average Pearson correlation is decreased from 0.719 to 0.710. Similarly, if we remove the frequency feature, the average precision will reduce from 66.39 to 65.27. Surprisingly, the simple wiki feature doesn't affect the performance significantly when we remove it from the current feature set.

NSR w/o features	**avg. P@1**	**avg. Pearson**
Word lexicon	65.26	0.710
Freq	65.37	0.703
Simple wiki	66.40	0.715
NSR + FNN + MSE	66.39	0.719
NSR model	**67.07**	**0.719**

Table 2.3: Substitution ranking performance comparison by eliminating features from the NSR Model

Feature Insertion/Replacement Test This experiment is conducted by adding/replacing each feature individually to the NSR model. The experiment results in Table 4 show a clear decrease in performance for features: syllable, length. The average precision has decreased to 65.99 and 65.65 respectively.

We test the below features: PPDB (PN15), word2vec, word length, word syllables, original word frequencies, morphological max frequency strategy, and morphological summation frequency strategy. For all these features, the model's performance decreased. The performance decreased especially with the installation of the *syllable number* and *word length* feature, which demonstrated their irrelevance to the simplicity. In terms of word frequency strategies, when we replaced the average strategy with a summation strategy or maximum strategy, the average precision dropped to 65.83 and 66.02 respectively.

Loss function/Label Test In this experiment, we test the performance of hinge loss and +1/-1 label performances. The experiment replaces the original label with the +1/-1 label and changes the

NSR w/ feature	avg. P@1	avg. Pearson
Word2vec	66.38	0.718
PPDB	66.29	0.718
Syllable	65.99	0.717
Word length	65.65	0.716
Freq˙max	66.02	0.716
Freq˙sum	65.83	0.713
NSR + FNN + MSE	66.39	0.719
NSR model	**67.07**	**0.719**

Table 2.4: Substitution ranking performance comparison by adding/replacing features to the NSR model

loss function from MSE loss to hinge loss. The ablation study changes one component each time. The test result in Table 5 indicates the performance of different loss function/label combinations.

Method	avg. P@1	avg. Pearson
Hinge & +1/-1 label	**67.07**	**0.719**
MSE & +1/-1 label	66.84	0.718
Hinge & normal label	66.67	0.718
MSE & normal label	66.49	0.716

Table 2.5: Substitution ranking performance comparison of the loss function/label combinations

2.6 Related work

For the related work, we emphasize three core concepts for lexical simplification: word simplicity computation, relatedness computation, and computation model. For each work listed below, we summarize each work based on these aspects. (BBE11) proposed the simplicity-based substitution ranking method, introduced word length and frequency jointly to decide the simplicity of a word. The Lexical simplification task was initially introduced in the Semeval 2012 conference as a standard task. (SJM12b) summarize the initial attempt by all the participants. The most important discovery is the importance of word frequency as the single most important feature for the lexical simplification. (HMK14) introduces the SVM-rank model which utilizes a supervised learning method in substitution ranking. The word embedding method is introduced in (GS15) to generate the candidate without relying on the rule-based candidate generation, which gives us an alternative

to generating correlated candidates. (PS16) introduces the contextualized embeddings to further utilize the relatedness to improve the system performance. (PS17b) is the first work that introduces the neural pairwise ranking framework into the lexical simplification pipeline and refreshes the state-of-the-art result at that time. It is the first work to introduce the neural network method into this area. (MX18) introduces first complexity lexicon of words and the Gaussian binning technique based on the framework of (PS17b). It achieves the new state-of-the-art result in the substitution ranking result. (QLZ19) utilizes BERT in the candidate generation of the lexical simplification process and successfully achieves the best result of the candidate generation compared with the previous wordNet-based method.

2.7 Conclusion

In this paper, we introduced a new model for lexical simplification ranking that successfully boosted the ranking performance and achieves the new state-of-the-art result. The importance of frequency for the word simplicity is discussed so that the new frequency-based features are designed. New neural model architecture and loss function further improved the simplification ranking performance. The effectiveness of our new model is validated on the SemEval 12 lexical simplification task and the ablation study demonstrates the contributions of each component.

CHAPTER 3

Amplified Negative Sampling: Improving Contrastive Representation Learning via Sample-Efficient Large-Classifier Training

In this paper, we propose a new training method, called *amplified negative sampling (ANS)*, to improve the quality and the training efficiency of constrastive representation learning when the learning is performed via a large output-class classifier. ANS is a simple yet effective change to the well-known Negative sampling technique mikolov2013distributed that leads to (1) a lower training cost and (2) a representation that is likely to be more effective for general tasks other than the particular classification task used for training. In developing ANS, we first conduct a careful analysis of the negative sampling technique and explain why and how our ANS method is helpful. We then conduct experiments on real-world datasets to demonstrate that our proposed method works as predicted by our theoretical analysis and leads to sampling cost savings with a performance boost compared to the standard technique. Our theoretical analysis will also provide a better understanding of a few empirical observations that are well known among practitioners when negative sampling is employed.

In this paper, we derive a rigorous analysis for negative sampling (MCC13b) and propose a new sample-efficient negative sampling method for training a multi-class classifier $C : X \to Y$ for representation learning(X: input features, Y: output class labels) when the output-class size is large, say, $|Y| = 50,000$. Typically, when a classifier is modeled as a neural network, the final layer is implemented as a softmax layer with *one output neuron per each output class label* $y \in Y$, making it prohibitively expensive to train even for a reasonably large output-class size.

As multi-class classification has a long history, many solutions has been proposed to tackle the

large output problem, such as importance sampling (BS03; BS08), one-class classification (PZC08; YBL17) and softmax approximation (LLK18; MA16). However, these methods either target on improving the multi-label classification performance or approximating the softmax function with fewer computations which are different for contrastive representation learning. Moreover, these methods either suffer from the low efficiency or unstable performance problem for large output class (LLK18; MT12).

For contrastive representation learning, the goal is to *learn a distributed representation by solving a multi-label classification problem* (MT12; MK13). Negative sampling (MCC13b) can avoid computing the softmax function and its normalizing constant in the classification task because of the Noise-contrastive Estimation (NCE) principle (GH10). Due to its simplicity and efficiency, negative sampling is one of the most popular techniques used in practice (MSC13; WMW17; GL16a; BK16). In particular, it is widely utilized in many embedding frameworks, such as word embedding (MSC13), graph embedding (GL16a; WMW17), and product-user embedding (BK16).

The key idea behind negative sampling is as follows: Given a training data point (x_i, y_i), the standard training algorithm updates the weights of the output neurons for *all* y's $\in Y$, not just for the training label y_i, making the training cost proportional to the output-class size $|Y|$. Negative sampling avoids this high cost by adjusting the weights for (1) the given training label, $y = y_i$ ("the positive sample") and (2) just a few y's that are randomly sampled from $Y - \{y_i\}$ ("negative samples"). Clearly, taking a few negative samples reduces the training cost by several orders of magnitudes when the output class size is large.[1]

In general, it is reported that using a larger negative sample size leads to better downstream performance. For example, when Mikolov used negative sampling to embed words into high-dimensional vectors in (MSC13), he reported between 2-15% increase in downstream task performance when he used the 15 negative samples ($k = 15$) compared to 5 negative samples ($k = 5$). Unfortunately, the training cost of $k = 15$ is three times as large as that of $k = 5$, making its use significantly less appealing in practice. For instance, since training on a larger corpus generally improves the downstream performance as well, it may be the case that using a smaller k on a larger

[1]It worth noting that since the softmax function has been circumvented, if cross-entropy loss is adopted, the loss function will be binary cross-entropy loss instead softmax cross-entropy loss, for more detail please see (GH10).

corpus may be just as good as or even better than using a larger k on a smaller corpus. This is the primary topic of this paper. Is it possible to get the best of both worlds? Can we achieve a higher-quality model trained on a larger k without paying its training cost?

Moreover, How does different loss function affect the negative sampling performance for the classification in contrastive representation learning? What's the exact convergence point of the classification model?

To obtain an answer to this question, we first conduct a rigorous mathematical analysis of the impact of the negative-sampling technique on the accuracy of the learned model. The result of this analysis will show exactly how negative sampling affects the accuracy of the learned model and provide theoretical explanations for a few empirical observations that have been well known among practitioners. It will also shed light on how it can be further improved for higher training efficiency. Based on this insight, we find a surprisingly simple yet effective modification to the negative sampling technique, named *amplified negative sampling*. Compared to the standard negative-sampling technique, our proposed technique can be used to either (1) *improve the prediction accuracy* of the trained model *for the same training cost* or (2) *lower the training cost for the same prediction accuracy*. We demonstrate the effectiveness of our proposed technique through extensive experiments on real-world data sets.

In summary, we make the following contributions in this paper:

- We provide a rigorous mathematical analysis of the impact of negative sampling on the classification model under the different loss functions, such as $L1$, $L2$, and cross-entropy loss. We derive the exact analytical form of the global optimal points when the classifier is trained with negative sampling under the three popular loss functions. The result of our analysis illuminates the implication of negative sampling with different loss functions clearly. For example, we prove that the optimal model trained with the $L2$ loss function is the same as that with the *cross entropy* loss function. We also show that the optimal model trained with the $L1$ loss function is a sparse binary model. As far as we know, this is the first work that conducts a rigorous theoretical analysis for negative sampling in the multi-class classification for contrastive representation learning.

- We propose *amplified negative sampling*, a simple yet effective modification to the widely-used negative-sampling technique that can improve its accuracy and lower its training cost based on our rigorous mathematical analysis.

- We compare the effectiveness of our proposed technique with standard negative sampling by conducting an extensive set of experiments on real-world data sets. Our results show that the effectiveness of our technique is in line with our theoretical prediction and can often be *ten times as sample efficient* as the standard technique.

The rest of this paper is organized as follows. In Section 2, we formally describe the multi-class classification problem and review the standard negative sampling technique. In Section 3, we analyze the mathematical property of negative sampling and discuss its practical implications. Then in Section 4, we propose amplified negative sampling. In Section 5,we present the results of our experiments. We review related work in Section 6 and wrap up the paper in Section 7.

3.1 Framework

In this section, we briefly go over the general problem formulation of multi-class classifier learning and negative sampling to introduce key notation used in this paper.

3.1.1 Preliminaries

We are given a dataset $D = \{(x_1, y_1),(x_2, y_2),...,(x_n, y_n)\}$, where $x_i \in X$ is an *input feature* and $y_i \in Y$ is an *output class label*. We assume a discrete space of feature values x_i and output labels y_i, such that x_i and y_i take an integer value between $1 \leq x_i \leq m$ and $1 \leq y_i \leq m$. The multi-class classifier learning problem is to find a classifier $C : X \rightarrow Y$ that returns the correct label y_i given the input feature x_i: $y_i = C(x_i)$. Due to noise in the dataset and the uncertainty in predicting the correct label, this problem is often formulated as finding a conditional probability distribution $P(y|x)$ from the dataset D, which is interpreted as the probability that the correct output label is y given the input feature x.

Note that this formulation encompasses not just the multi-class classifier learning problem, but

also most of the "data embedding" problems, such as word embedding (MCC13b; MSC13), graph embedding (GL16b), and item embedding (BK16). For example, the well-known skip-gram model for word2vec (MSC13) falls under this formulation by defining a *context word* as an input feature x_i and any *target word* that appears near the context word as an output label y_i.

Learning the conditional probability function $f(x,y) = P(y|x)$ from the dataset D is done by assuming a parameterized hypothesis space $f_\theta(x,y) = P_\theta(y|x)$, where the hypothesis space $f_\theta : (x,y) \to [0,1]$ is a space of differentiable functions parameterized by $\theta \in R^d$. Among all possible parameters $\theta \in R^d$, an *optimal parameter* θ^* is chosen to minimize the loss function $L(f_\theta, D)$, where $L(f_\theta, D)$ captures the "loss of f_θ" or the difference between f_θ and D. Multiple definitions of the loss function $L(f_\theta, D)$ are used in practice, including $L1$, $L2$, and *cross entropy*:

$$L_1 : \sum_{(x_i,y_i)\in D} \sum_{y\in Y} |\mathbb{1}(y = y_i) - f_\theta(x_i,y)| \tag{3.1}$$

$$L_2 : \sum_{(x_i,y_i)\in D} \sum_{y\in Y} (\mathbb{1}(y = y_i) - f_\theta(x_i,y))^2 \tag{3.2}$$

$$\begin{aligned} L_{CE} : - \sum_{(x_i,y_i)\in D} \sum_{y\in Y} & [\mathbb{1}(y = y_i) \log f_\theta(x_i,y) \\ & + (1 - \mathbb{1}(y = y_i)) \log (1 - f_\theta(x_i,y))] \end{aligned} \tag{3.3}$$

Here, $\mathbb{1}(y = y_i)$ is an indicator function that takes the value 1 if $y = y_i$ and 0 otherwise. In order to make our discussion concrete, we primarily assume $L2$ as our loss function in the rest of this paper and simply state the result of our analysis for the other loss functions.

The gradient-descent method is often utilized to identify the parameter θ^* that minimizes the loss function $L(f_\theta, D)$. Given the definition of the $L2$ loss function, its gradient is:

$$\begin{aligned} & \nabla_\theta L_2(f_\theta, D) = \\ & -2 \sum_{(x_i,y_i)\in D} \left[\sum_{y\in Y} (\mathbb{1}(y = y_i) - f_\theta(x_i,y)) \nabla_\theta f_\theta(x_i,y) \right] \end{aligned} \tag{3.4}$$

Note that the inner summation of the above equation makes its computation prohibitively expensive: For each training data $(x_i, y_i) \in D$, we take the inner sum over *every* output label $y \in Y$, not

just the training label y_i.[2] This makes the computational cost *proportional to the output class size* $|Y|$. We refer to the training method that computes the full gradient of Equation 3.4 as *full-gradient training*.

3.1.2 Negative Sampling

Negative sampling is a technique that tries to reduce the high computational cost of full-gradient training. The idea of negative sampling was originally proposed in 2010 as Noisy Contrastive Estimation (NCE) (GH10), which was generalized for natural language processing by Mihn in (MT12). It was used as part of the word2vec computation (MCC13b), which led to a wide adoption for general vector-embedding problems (GL16a; BK16; GL16b).

The basic idea of negative sampling can be summarized as follows: Given a training data (x_i, y_i), we refer to y_i as the "*positive sample*" and all other label $y \in Y - \{y_i\}$ as "*negative samples*." Full-gradient training sums up the gradients from (1) the positive sample y_i and (2) *all* negative samples $y(\neq y_i)$. Negative sampling, instead, sums up the gradients from (1) the positive sample y_i and (2) *just a few randomly-selected negative samples* $y \in Y - \{y_i\}$.

More precisely, we use NEG-$k(i)$ to represent the k randomly-chosen negative samples for the ith training data (x_i, y_i). That is, NEG-$k(i)$ is a size-k random subset of $Y - \{y_i\}$. Negative sampling then *approximates* the $L2$ loss function $L_2(f_\theta, D)$ as follows:

$$L_2(f_\theta, D)$$

$$= \sum_{(x_i,y_i)\in D} \left[\sum_{y\in Y} (\mathbb{1}(y = y_i) - f_\theta(x_i, y))^2 \right] \tag{3.5}$$

$$\tag{3.6}$$

[2] In certain cases, this sum over every $y \in Y$ is implicitly added to the hypothesis space $f_\theta(x, y)$. For example, when implemented as a neural network, the final layer is typically implemented as a softmax layer with one neuron per output label, which has the same effect as summing over ever $y \in Y$.

$$= \sum_{(x_i,y_i)\in D} \left[(\mathbb{1}(y = y_i) - f_\theta(x_i,y))^2 \Big|_{y=y_i} + \sum_{y\in Y-\{y_i\}} (\mathbb{1}(y = y_i) - f_\theta(x_i,y))^2 \right] \tag{3.7}$$

$$\approx \sum_{(x_i,y_i)\in D} \left[(1 - f_\theta(x_i,y_i))^2 + \sum_{y\in \text{NEG-}k(i)} f_\theta(x_i,y)^2 \right] \tag{3.8}$$

From Equation 3.5 to 3.7, the sum over $y \in Y$ is expanded into the sum of $y = y_i$ and $y \in Y - \{y_i\}$. From Equation 3.7 to 3.8, the sum over all negative samples $y \in Y - \{y_i\}$ is approximated by the sum over NEG-$k(i)$. Clearly, this approximation can decrease the cost of computing the loss function significantly when $|Y|$ is large, which is the key reason for its efficiency. But what is the accuracy of this approximation? Will we still be able to get the same accurate model despite this approximation? If not, what is its exact impact? In the next section, we investigate this issue analytically.

3.2 Analysis of Negative Sampling

In this section, we provide the results from our mathematical analysis on the accuracy of the model trained with negative sampling. In stating our results, we use $\#(x)$ to represent the number of times that the input feature value x appears in the training data D and $\#(x,y)$ to represents the number of times that the feature-and-label-value pair (x,y) appears in D. More formally,

$$\#(a) = |\{(x,y) \in D \mid x = a\}|$$
$$\#(a,b) = |\{(x,y) \in D \mid x = a \text{ and } y = b\}|$$

When we select the k negative samples for the ith training data (x_i,y_i), NEG-$k(i)$, we assume that a negative sample $y \in Y - \{y_i\}$ is sampled with replacement with probability p_y. Two popular choices of the sampling probability p_y are (1) the uniform distribution $p_y = c$ for some constant c and (2) according to the frequency of y in D. Our results are stated with the generic symbol p_y without making any explicit assumption on the sampling distribution. With this notation, we now state the

key result of our analysis.

Theorem 1 (Optimal Model of Negative Sampling)**.** *When the hypothesis space f_θ has sufficient capacity,*[3] *the optimal model f_{θ^*} trained with k negative samples is the following with high probability:*

- *For L1 loss:* $f_{\theta^*}(x,y) = \mathbb{1}\left(\frac{\#(x,y)}{\#(x)} > \frac{k \cdot p_y}{k \cdot p_y + 1}\right)$
- *For L2 loss:* $f_{\theta^*}(x,y) = \frac{\#(x,y)}{k \cdot p_y[\#(x) - \#(x,y)] + \#(x,y)}$
- *For cross-entropy loss:*
  $f_{\theta^*}(x,y) = \frac{\#(x,y)}{k \cdot p_y[\#(x) - \#(x,y)] + \#(x,y)}$

That is, for example, let θ_t^* be the parameter that minimizes the L1 loss function after t training epochs. Then for any $\varepsilon > 0$,

$$\lim_{t \to \infty} P\left(\left| f_{\theta_t^*}(x,y) - \mathbb{1}\left(\frac{\#(x,y)}{\#(x)} > \frac{k \cdot p_y}{k \cdot p_y + 1}\right)\right| < \varepsilon\right) = 1,$$

where $\mathbb{1}(a > b)$ is an indicator function whose value is 1 if $a > b$ and 0 otherwise.[4] *Similar statements can be made for L2 and cross entropy loss functions.*

3.2.1 Proof for L2 Loss

We assume a discrete domain of input data D, where each data point $(x,y) \in D$ is an integer value pair of $1 \leq x \leq m$ and $1 \leq y \leq l$. Within the discrete domain, the most general parameterization of the function $f_\theta(x,y)$ is to assign an independent parameter per every pair of values (x,y) within $1 \leq x \leq m$ and $1 \leq y \leq l$. We use the symbol θ_{ab} to represent these parameters, i.e., $f_\theta(a,b) = \theta_{ab}$ for $1 \leq a \leq m$ and $1 \leq b \leq l$, where θ_{ab} can take any value in $[0,1]$.

[3] By having sufficient capacity, we mean that the hypothesis space has an independently adjustable parameter per every discrete (x,y) value pair following the assumption of (GL14).

[4] More precisely, $\mathbb{1}(a > b)$ may take any value between 0 to 1 when $a = b$.

Given the notation, the L_2 loss function of negative sampling is:

$$L_2(f_\theta, D) = \sum_{(x,y)\in D} \left[(1 - f_\theta(x,y))^2 + \sum_{y' \in \text{NEG-}k(x,y)} f_\theta(x,y')^2 \right] \tag{3.9}$$

$$= \sum_{(x,y)\in D} \left[(1 - \theta_{xy})^2 + \sum_{y' \in \text{NEG-}k(x,y)} \theta_{xy'}^2 \right] \tag{3.10}$$

Here, we use the notation NEG-$k(x,y)$ to represent the k negative samples taken for the ith data point $(x,y) \in D$ to avoid adding subscript i to every x and y's. Note that Equation 3.10 is a quadratic function of θ_{xy}'s and takes the minimum value when $\frac{\partial L_2}{\partial \theta_{xy}} = 0$ or at the boundary values $\theta_{xy} = 0$ or 1.

To find the minimum, we take the partial derivative of L_2 for a particular parameter θ_{ab}. The partial derivative of Equation 3.10 can be simplified based on two facts:

- The partial derivative $\frac{\partial}{\partial \theta_{ab}}$ of the first term $(1 - \theta_{xy})^2$ is zero except when $x = a$ and $y = b$. Thus, only the constant value $-2(1 - \theta_{ab})$ remains and is summed as many times as (a,b) appears in D. That is, the sum of the first term is:

$$\frac{\partial}{\partial \theta_{ab}} \sum_{(x,y)\in D} (1 - \theta_{xy})^2 = \#(a,b)(-2)(1 - \theta_{ab}), \tag{3.11}$$

  where $\#(a,b)$ represents the number of times (a,b) appears in D.

- The partial derivative of the second term $\sum_{y' \in \text{NEG-}k(x,y)} \theta_{xy'}^2$ is also zero except when $x = a$ and $y' = b$. This can happen only when $x = a$, $y \neq b$ because a negative sample for (x,y) cannot contain y. Thus,

$$\frac{\partial}{\partial \theta_{ab}} \sum_{(x,y)\in D} \left[\sum_{y' \in \text{NEG-}k(x,y)} \theta_{xy'}^2 \right] = \sum_{\substack{(a,c)\in D \\ c \neq b}} \left[\sum_{b \in \text{NEG-}k(a,c)} 2\theta_{ab} \right] \tag{3.12}$$

  The outer sum is over all data points whose input feature value is $x = a$ except when the output label is $y = b$. Thus, the outer sum is equivalent to multiplying by $[\#(a) - \#(a,b)]$

since the summand is the constant $2\theta_{ab}$. That is,

$$\sum_{\substack{(a,c)\in D\\ c\neq b}} \left[\sum_{b\in \text{NEG-}k(a,c)} 2\theta_{ab} \right] = [\#(a) - \#(a,b)] \sum_{\substack{b\in \text{NEG-}k(a,c)\\ c\neq b}} 2\theta_{ab} \tag{3.13}$$

The remaining sum is over all instances of b within NEG-$k(a,c)$ with $c \neq b$. Since the probability that the value b is chosen as a negative sample is p_b, the expected number of times that b appears in NEG-$k(a,c)$ is kp_b when we take k random negative samples. From the weak law of large numbers, we know that the actual number of occurrence converges in high probability to its expectation as we take more samples. That is, the remaining sum converges in high probability to $(k\,p_b)(2\theta_{ab})$ as we take more new samples for NEG-$k(a,c)$ as the training epoch t passes:

$$\sum_{\substack{b\in \text{NEG-}k(a,c)\\ c\neq b}} 2\theta_{ab} \xrightarrow{P} k\,p_b\,2\theta_{ab} \qquad \text{as } t \to \infty \tag{3.14}$$

Or more precisely, for any ε,

$$\lim_{t\to\infty} P\left(\left| \left(\sum_{\substack{b\in \text{NEG-}k(a,c)\\ c\neq b}} 2\theta_{ab} \right) - k\,p_b\,2\theta_{ab} \right| < \varepsilon \right) = 1. \tag{3.15}$$

Overall, we have shown that the partial derivative of the second term becomes

$$\frac{\partial}{\partial \theta_{ab}} \sum_{(x,y)\in D} \left[\sum_{y'\in \text{NEG-}k(x,y)} \theta_{xy'}^2 \right] \xrightarrow{P} [\#(a) - \#(a,b)] k\,p_b\,2\theta_{ab} \tag{3.16}$$

By combining Equations 3.11 and 3.16, we now have:

$$\frac{\partial L_2}{\partial \theta_{ab}} \xrightarrow{P} \#(a,b)(-2)(1-\theta_{ab}) + [\#(a) - \#(a,b)] k\,p_b\,2\theta_{ab} \tag{3.17}$$

as $t \to \infty$. At the minimum of L_2, $\frac{\partial L_2}{\partial \theta_{ab}} = 0$ for every θ_{ab}, so we have:

$$\frac{\partial L_2}{\partial \theta_{ab}} \xrightarrow{P} \#(a,b)(-2)(1-\theta_{ab}) + [\#(a) - \#(a,b)] k\,p_b\,2\theta_{ab} = 0 \tag{3.18}$$

Solving the equation for θ_{ab}, we get

$$\theta_{ab} \xrightarrow{P} \frac{\#(a,b)}{k\,p_b[\#(a)-\#(a,b)]+\#(a,b)}. \tag{3.19}$$

Note that the right-hand side of Equation 3.19 is always between $[0,1]$, so we do not need to worry about the case when the minimum happens at a boundary value. By definition $f_\theta(a,b) = \theta_{ab}$ and the optimal θ_t^* at training epoch t converges to the value derived above as $t \to \infty$. That is, we have shown that

$$f_{\theta_t^*}(a,b) = \theta^*_{t,ab} \xrightarrow{P} \frac{\#(a,b)}{k\,p_b[\#(a)-\#(a,b)]+\#(a,b)}. \tag{3.20}$$

or equivalently, for any ε,

$$\lim_{t\to\infty} P\left(\left| f_{\theta_t^*}(x,y) - \frac{\#(x,y)}{k\cdot p_y[\#(x)-\#(x,y)]+\#(x,y)}\right| < \varepsilon\right) = 1. \tag{3.21}$$

3.2.2 Proof for L1 Loss

The proof for the $L1$ loss function can be done similarly. Using the argument similar to $L2$, we can show that

$$\frac{\partial L_1}{\partial \theta_{ab}} \xrightarrow{P} -\#(a,b) + [\#(a) - \#(a,b)]k\,p_b \qquad \text{as } t \to \infty \tag{3.22}$$

Note that the right-hand side of Equation 3.22 is a constant independent of θ_{ab}. If this partial derivative is negative, i.e., $\frac{\partial L_1}{\partial \theta_{ab}} = -\#(a,b) + [\#(a) - \#(a,b)]k\,p_b < 0$, the minimum occurs at the right boundary value $\theta_{ab} = 1$. Otherwise, the minimum occurs at the left boundary value $\theta_{ab} = 0$. That is, the optimal parameter $\theta^*_{t,ab}$ becomes

$$\theta^*_{t,ab} \xrightarrow{P} \begin{cases} 1 & \text{if } -\#(a,b) + [\#(a) - \#(a,b)]k\,p_b < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

as $t \to \infty$. The inequality condition in the above equation can be rearranged to

$$\theta^*_{t,ab} \xrightarrow{P} \begin{cases} 1 & \text{if } \frac{\#(a,b)}{\#(a)} > \frac{k \cdot p_b}{k \cdot p_b + 1} \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

which completes our proof.

3.2.3 Proof for Cross Entropy Loss

Again, using a similar argument, we can show that the partial derivative for the cross-entropy loss becomes:

$$\frac{\partial L_{CE}}{\partial \theta_{ab}} \xrightarrow{P} \frac{\#(a,b)}{\theta_{ab}} - \frac{k\, p_b [\#(a) - \#(a,b)]}{1 - \theta_{ab}} \qquad \text{as } t \to \infty \tag{3.25}$$

Solving $\frac{\partial L_{CE}}{\partial \theta_{ab}} \xrightarrow{P} \frac{\#(a,b)}{\theta_{ab}} - \frac{k\, p_b [\#(a) - \#(a,b)]}{1 - \theta_{ab}} = 0$ for θ_{ab} to get the optimal parameter, we get:

$$\theta^*_{t,ab} \xrightarrow{P} \frac{\#(a,b)}{k\, p_b [\#(a) - \#(a,b)] + \#(a,b)} \tag{3.26}$$

Note that the above theorem derives the analytic form of the globally optimal model when trained with k negative samples. However, it does not necessarily guarantee that this globally optimal model is always achievable in practice. For example, a particular training method may be trapped in a local minimum during training and may not reach the global optimum. It is also possible that the hypothesis space used for a particular application is more constrained than the sufficient-capacity assumption and does not contain the globally-optimal model within its parameter space. In those cases, we may consider the above result as providing an *upper bound* to any achievable model. Interestingly, our later experiments show that the globally optimal model is often achieved for popular models used in practice. For example, when we trained the skip-gram model (MCC13b), we were able to achieve the globally optimal model (i.e., the distance of the trained model to the optimal model is close to 0) even though its hypothesis space is significantly more constrained than the sufficient-capacity assumption.

The concrete meaning of Theorem 1 may be difficult to see due to its generality. Thus we use use a few corollaries to explain the implications of the our theorem more concretely. First, we show

the analytic form of the optimal model when trained with the full-gradient method.

Corollary 1.1 (Full-Gradient Model)**.** *When the hypothesis space f_θ has sufficient capacity, the respective optimal models trained with the full-gradient method are the following:*

- *For L1 loss:* $f_{\theta^*}(x,y) = \mathbb{1}\left(\frac{\#(x,y)}{\#(x)} > \frac{1}{2}\right)$
- *For L2 loss:* $f_{\theta^*}(x,y) = \frac{\#(x,y)}{\#(x)}$
- *For cross-entropy loss:* $f_{\theta^*}(x,y) = \frac{\#(x,y)}{\#(x)}$

The above corollary formally proves a few interesting facts: First, despite their difference, the optimal model from both $L2$ and cross-entropy loss functions are *exactly the same maximum-likelihood estimator (MLE)*. Second, the optimal model trained under the $L1$ loss function is a *sparse binary model* whose $f_{\theta^*}(x,y)$ values takes the value of either 0 or 1. Note that these two observations are general properties of a gradient-based method, not just of the full-gradient method. That is, in Theorem 1, we also see that $L1$ loss function leads to a binary model and $L2$ and cross-entropy loss functions result in the same optimal model.

Proof: For the $L2$ loss, when the negative sample sets contain all negative samples except the positive one, the sum $\sum_{b\in\text{NEG-}k(a,c)} 2\theta_{ab}$ becomes $2\theta_{ab}$ because every output label (except the positive one) appears once and only once in NEG-$k(a,c)$. Note that b is not an index that ranges over all output labels, but a particular output label value corresponding to the parameter θ_{ab} for which the partial derivative is taken. The rest of the proof can be completed following the same steps of Theorem 1. Proofs for other loss functions are similar.

Our second corollary formally proves the "model collapse" phenomenon frequently observed in practice.

Corollary 1.2 (Model Collapse with No Negative Sample)**.** *When $k = 0$, i.e. NEG-$k(i) = \varnothing$, the optimal model collapses to* 1 *under all three loss functions. That is,*

$$f_{\theta^*}(x,y) = 1 \text{ for all } (x,y) \in D \tag{3.27}$$

This result explains the "model collapse" phenomenon reported in (GL14). In the study, the authors report that if trained with *zero* negative sample, the model converges to a constant function

1, so it is essential to have a negative sample. The above corollary shows that this phenomenon is an intrinsic property of negative sampling. Overall, Corollaries 1.1 and 1.2 show that as we increase the negative sample size k, the optimal model $f_{\theta^*}(x,y)$ gets closer to the maximum-likelihood estimator $\tilde{P}(y|x) = \frac{\#(x,y)}{\#(x)}$,[5] a trend that makes intuitive sense.

3.3 Amplified Negative Sampling

Our analysis shows that the optimal model trained with the full-gradient method is equivalent to the maximum likelihood estimator $f_{\theta^*}(x,y) = \frac{\#(x,y)}{\#(x)}$ while the model from negative sampling is not. Assuming that the dataset D is a representative sample from the true underlying distribution $P(y|x)$ (that is, $P(y|x) \approx \frac{\#(x,y)}{\#(x)}$) we can expect that full-gradient training are likely to result in a better model than negative sampling for a classification task.

Indeed, this is the general trend reported in the literature – not just for classification tasks but also for embedding tasks where negative sampling is frequently used. For example, when Mikolov used negative sampling to embed words into high-dimensional vectors in (MSC13), he reported noticeable accuracy increase in the word-analogy-task performance when he used $k = 15$ compared to $k = 5$.[6] Unfortunately, the training cost of using k negative samples is proportional to k, making the use of a higher k value unappealing in practice; since training on a larger corpus generally increases the embedding quality as well, one may prefer using a smaller k on a larger corpus than using a larger k on a smaller corpus, assuming that it is bound by the same computational cost. This is where our study of *amplified negative sampling* started. Can we obtain the high quality model of higher k for the low computational cost of lower k? Can we get the best of the both worlds? We now explain how this can be achieved using amplified negative sampling.

Amplifying Factor. The key idea behind our amplified negative sampling comes from Theorem 1. From its analytic form, we observe that the optimal model $f_{\theta^*}(x,y)$ depends *only on the negative-sample size k*, not on how the samples are obtained. Given this, can we use *one negative*

[5] Or its sparsified version $\mathbb{1}(\frac{\#(x,y)}{\#(x)} > \frac{1}{2})$ in case of $L1$.

[6] For embedding tasks, the ultimate goal is to obtain vector representations of words that lead to high accuracy for downstream tasks. For this reason, obtaining the MLE model may not necessarily be "better." However, in a relatively small range of k, it is generally observed that higher k leads to better downstream task performance as well.

sample multiple times during training, pretending that it is the result from multiple random sampling? More formally, what will happen if we change the loss function $L(f_\theta, D)$ of Equation 3.8 (standard negative sampling) to the following?

$$L_2(f_\theta, D) = \sum_{(x_i,y_i)\in D} \left[(1 - f_\theta(x_i, y_i))^2 + \beta \sum_{y \in \text{NEG-}k(i)} f_\theta(x_i, y)^2 \right] \tag{3.28}$$

Note that Equation 3.28 is different from Equation 3.8 just by a constant factor β of the second term. We refer to β as a *amplifying factor*, since its intended role is to artificially "amplify" the effect of the negative samples NEG-k(i) by making its size look larger than they really are. Surprisingly, the following corollary shows that adding a amplifying factor produces this exact outcome.

Corollary 1.3 (Amplified Negative Sampling)**.** *When the hypothesis space f_θ has sufficient capacity, the respective optimal models trained with k negative samples with amplifying factor β under the three loss functions are the following with high probability:*

- *For L1 loss:* $f_{\theta^*}(x,y) = \mathbb{1}\left(\frac{\#(x,y)}{\#(x)} > \frac{\beta \cdot k \cdot p_y}{\beta \cdot k \cdot p_y + 1}\right)$
- *For L2 loss:* $f_{\theta^*}(x,y) = \frac{\#(x,y)}{\beta \cdot k \cdot p_y [\#(x) - \#(x,y)] + \#(x,y)}$
- *For cross-entropy loss:*
  $f_{\theta^*}(x,y) = \frac{\#(x,y)}{\beta \cdot k \cdot p_y [\#(x) - \#(x,y)] + \#(x,y)}$

Note that the factor k in Theorem 1 is replaced with βk in Corollary 1.3. That is, the amplifying factor β makes the optimal model trained with NEG-k effectively identical to the one from NEG-βk! Simply by multiplying a constant β to the loss function, we get the optimal model from a much larger sample size.

Proof: The $L2$ loss function for boosted negative sampling is

$$L_2(f_\theta, D) = \sum_{(x,y)\in D} \left[(1 - f_\theta(x,y))^2 + \beta \sum_{y' \in \text{NEG-}k(x,y)} f_\theta(x,y')^2 \right] \tag{3.29}$$

Since the only difference from Equation 3.9 (standard negative sampling) is the coefficient β of the second summation, we can show that

$$\frac{\partial L_2(f_\theta, D)}{\partial \theta_{ab}} \xrightarrow{P} \#(a,b)(-2)(1 - \theta_{ab}) + [\#(a) - \#(a,b)]\beta\, k\, p_b\, 2\theta_{ab} \tag{3.30}$$

(compare it against Equation 3.17) as $t \to \infty$. By setting $\frac{\partial L_2}{\partial \theta_{ab}} = 0$, we can show that

$$\theta^*_{t,ab} \xrightarrow{P} \frac{\#(a,b)}{\beta\, k\, p_b[\#(a) - \#(a,b)] + \#(a,b)} \qquad \text{as } t \to \infty. \tag{3.31}$$

The proof for other loss functions can be done similarly.

Optimal Amplifying Factor and Model Quality. The similarity of the learning rate and the amplifying factor raises another interesting question. How big a amplifying factor can we safely use? It is well known that a higher learning rate generally leads to a faster convergence rate initially, but it makes the training process less stable at a later stage. Will using a large amplifying factor lead to similar behavior? Or will it always be better to use a larger amplifying factor? Our analysis indicates that the optimal value of β might be $\beta = \frac{1}{kp_y}$ since this value leads to the MLE model.

The results from our experiments do not provide a single answer to this question. In the experiments conducted with our own code, where we measure the model accuracy in terms of the difference of the trained model from MLE, we observe that using a large amplifying factor always produces a model closer to MLE. It also does not introduce much instability to the training process all the way from 1 through $\beta = \frac{1}{kp_y}$. For downstream tasks, the amplify factor almost always brings *improvement* compared with non-amplified cases(see Experiments). An empirical safe upper bound is $\beta = \frac{1}{kp_y}$.

In our experiments, we also explored a few other directions, including (1) decaying the amplifying factor over epochs similarly to decaying the learning rate, (2) dynamically setting the amplifying factor based on the "loss value" of the negative sample similarly to the idea of importance sampling and (3) early stopping, where we stop using the amplifying factor after a few epochs. We find that these changes do not introduce a meaningful difference to the results.

3.4 Experiments

The primary goal of this section is to experimentally investigate the following two issues: (1) Does the result of our analysis hold in practice? We want to examine how well our theoretical results match with experiments. We also want to experimentally explore a few questions raised in

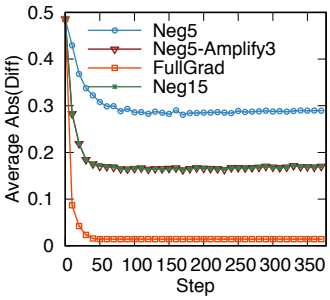


Figure 3.1: Model accuracy

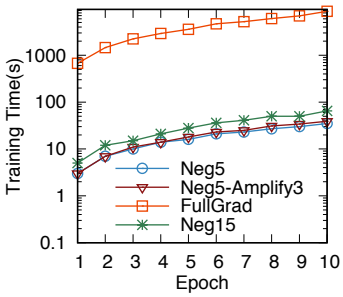


Figure 3.2: Training time.

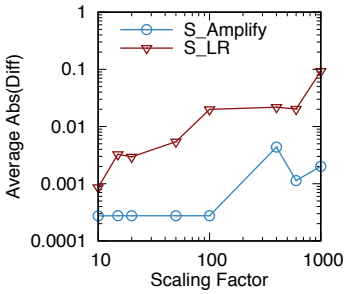


Figure 3.3: Amplifying factor vs learning rate

this paper, including the choice of the optimal amplifying factor and the difference between the learning rate and the amplifying factor. (2) Does amplified negative sampling help other downstream tasks as well? The performance of other downstream tasks may not necessarily depend on how accurately the trained model captures the conditional probability $P(y|x)$, so we want to experimentally check whether amplified negative sampling has positive effects on other downstream tasks or not.

In the subsection (Model Accuracy and Training Efficiency) 3.4.1, we explore the first issue by measuring the difference between the maximum likelihood estimator (MLE) $\tilde{P}(y|x) = \frac{\#(x,y)}{\#(x)}$ and the models trained with (a) full-gradient training (b) negative sampling and (c) amplified negative sampling. The results of our experiments show that the conclusions of our analysis hold in practice to a surprising degree of accuracy. They also show that the learning rate and the amplifying factor have vastly different effects on the trained model. In the subsection (Experiments on Other Downstream Tasks) 3.4.2, we investigate the second issue by running experiments on three different downstream tasks: word-analogy tasks (MCC13b), rare-word-similarity tasks (BGJ17), and graph-node-classification tasks (GL16a). Here, we observe that amplified negative sampling leads to improved performance on these downstream tasks as well.

In summary, our experimental results strongly indicate that there really is not much downside to using amplified negative sampling; as long as we use a reasonably small amplifying factor, say $\beta = 3$, amplified negative sampling leads to lower training time and higher model accuracy.

3.4.1 Model Accuracy and Training Efficiency

Experimental Settings. In this subsection, we experimentally compare four training algorithms, full-gradient training (*FullGrad*), 5 negative samples (*Neg5*), 5 negative samples with the amplifying factor 3 (*Neg5-Amplify3*), and 15 negative samples (*Neg15*), under three different loss functions, $L1$, $L2$, and cross entropy. For the choice of the hypothesis space $f_\theta(x,y)$ and the training set, we use a setting similar to (MCC13b). That is, as our hypothesis space we use the skip-gram model of (MCC13b) with a 100-dimensional hidden layer. As our dataset, we use a subset of Text8 corpus from (MCC13b) by extracting the first 75,000 words and applying the same min_count filter of 5 in (MCC13b).[7] We use the stochastic-gradient descent (SGD) with the batch size of 500 as the training algorithm. All our experiments use the window size 3 and the learning rate 0.025 unless noted otherwise. All other parameter settings are the same as in (MCC13b). All results reported are the average of three independent runs with identical settings. All codes were implemented using PyTorch v1.0.1.

Model Accuracy and Convergence Rate. In Figure 3.1, we compare the model accuracy and convergence rate when the model is trained with the four algorithms (FullGrad, Neg5, Neg5-Amplify3, Neg15) under the $L2$ loss function.[8] In the graph, the horizontal axis corresponds to the stochastic-gradient-descent training batch steps (with roughly 70 steps corresponding to one training epoch) and the vertical axis corresponds to the average absolute difference between the trained model $f_{\theta^*}(x,y)$ and MLE $\tilde{P}(y|x) = \frac{\#(x,y)}{\#(x)}$, i.e., $\sum_{(x,y)\in D} \frac{1}{|D|} \left| f_{\theta^*}(x,y) - \frac{\#(x,y)}{\#(x)} \right|$.

From the graph, a few things are clear: (1) Full-gradient training converges to MLE. Even at epoch 1 (step 70), the mean absolute difference of *FullGrad* is close to zero, indicating that it converged to MLE. (2) The amplifying factor β effectively "increases" the negative sample size by the factor β in terms of model accuracy. The mean absolute difference of Neg5-Amplify3 and Neg15 are the same at every training step — they overlap so closely and it is difficult to tell them

[7] Using a subset of Text8 here is due to the high training cost of *FullGrad* and our desire to keep the training time at a manageable level. In our next experiments on other downstream tasks, we run report our results from experiments on much larger datasets.

[8] While we performed experiments under all three loss functions, we report the results only from $L2$ here due to space limit. The conclusions from other loss functions are essentially the same.

apart in the graph — indicating that they both converge to the same model at the same rate. This result is what our theoretical analysis predicts: $(k = 15, \beta = 1)$ leads to the same optimal model as $(k = 5, \beta = 3)$. (3) A model trained on larger k approximates MLE better. The mean absolute difference of Neg15 is significantly smaller than that of Neg5.

Training Time and Computational Cost. In Figure 3.2, we compare the training time of the four algorithms. The horizontal axis corresponds to training epochs and the vertical axis corresponds to training time, which roughly captures the computational cost of each algorithm. The vertical axis is logarithmic; since the training time of *FullGrad* is two orders of magnitude larger than others, its result is not visible in the same graph otherwise. From the graph, we again observe what is predicted by our analysis. The training time of Neg5-Amplify3 is practically the same as that of Neg5. That is, Neg5-Amplify3 works almost like Neg5 in terms of its training time and computational cost, but it works almost like Neg15 in terms of its model accuracy and convergence rate! Amplified negative sampling indeed gives the best of both worlds.

Learning Rate vs Amplifying Factor. In Figure 3.3, we compare the effect of using different learning rates and amplifying factors. The graph is from training the model using Neg15 under $L1$ loss. The curve labeled as *S_LR* is obtained by multiplying the default learning rate of 0.025 by a factor between 10 and 1,000. The curve labeled as *S_Amplify* is obtained by including the amplifying factor β between 10 and 1,000. The vertical axis is again in the logarithmic scale due to the high difference between the two curves and represents the model accuracy (the mean absolute difference from MLE) at the given learning rate/amplifying factor. From the graph, we see that changing the learning rate and changing the amplifying factor lead to vastly different results. As we increase the learning rate, the trained model diverges further away from MLE. When we increase the amplifying factor, however, the trained model stays close to MLE all the way through $\beta = 100$. Only after $\beta > 100$, the model starts to diverge and becomes unstable. This result is consistent with our analysis; according to Corollary 1.3, amplifying converges to MLE at $\beta \approx 140$ under the current setting,[9] so its divergence beyond $\beta > 140$ is expected.

[9] Amplified negative sampling converges to MLE when $\beta k p_y = 1$. Given $k = 15$ and the uniform probability $p_y \approx 1/2000$ for this experiment, $\beta k p_y = 1$ at $\beta \approx 140$.

3.4.2 Experiments on Other Downstream Tasks

We investigate the effect of amplifying factor on downstream tasks: (1) word-analogy tasks (b) rare-word-similarity tasks and (c) graph-node-classification tasks. The word-analogy task includes 14 sub-tasks. Due to the space limit, we only display one sub-task for results on the word-analogy task of amplified Neg5, Neg10, and non-amplified Neg100 at training epoch 15.

Word2vec: Word analogy settings This test is conducted with the Skip-gram model trained on the Text8 corpus (MSC13) using the code downloaded from (wor19) with our amplifying factor code. We use the default parameter settings of the downloaded code.

In Table 3.1 we show the accuracy of the word-analogy task with various amplifying factors on Neg5 and Neg10 comparing with Neg100 for the second word-analogy semantic tasks of (MSC13). From the results, the trend is clear: the performance of Neg5-Amplify reaches an accuracy that closes to Neg100 and Neg10-Amplify can achieve even higher accuracy than Neg100. A clear fact is the amplified cases' performance is *always better* than non-amplified cases.

Table 3.1: Word-analogy task top-1 accuracy with different amplifying factors.

Neg10:β	1	2	4	8	10	20	30	Neg100
Acc	73.24	78.04	78.35	79.83	79.62	**84.32**	77.68	81.92
Neg5:β	1	2	4	10	12	16	20	Neg100
Acc	71.91	74.87	78.35	78.60	78.65	78.75	81.77	**81.92**

3.4.3 Running time experiment

Fasttext: Rare word similarity. For the word-similarity task, we use the code for Fasttext (BGJ17) downloaded from (fas19) using its default settings after the addition of amplifying code. The model is trained on the rare word dataset (RW) (LSM13). The effectiveness of a trained model is measured by the Spearman's rank-correlation coefficient (Spe04) between the human judgment in the dataset and the output from the trained model.

Word2vec: Word analogy settings This test is conducted with the Skip-gram model trained on the Text8 corpus (MSC13) using the code downloaded from (wor19) with our amplifying factor code. We use the default parameter settings of the downloaded code.

Node2vec: Node classification. For the node classification task, we use the node2vec model (GL16a) with the code downloaded from (nod19). The dataset used is BlogCatalog (ZL09). The nodes are embedded into a 50-dimensional space and we use 20-80 train-test split. The default parameter setting of the downloaded code is used after the addition of amplifying code.

Table 3.2: Training time (seconds).

Model	Neg5	Neg5-Amplify3	Neg15
Fasttext	736	731	1404
Word2vec	17.35	17.37	36.93
Node2vec	3.28	3.28	4.00

In Table 3.2, we show the training times of Word2vec, Fasttext, and Node2vec. In all three cases, the training time of Neg5-Amplify is close to Neg5 and is significantly smaller than that of Neg15. In the cases of Fasttext and Word2vec, the difference is by a factor 2. In the case of Node2vec, the difference is much smaller due to the dominance of other training overhead when the absolute training time is small. From our other experiments whose results we could not include here due to space limit, we observe the same general trend: The downstream-task performance of Neg5-Amplify3 is close to Neg15 while its training cost is close to Neg5.

3.4.4 Language Model perplexity test

In this section, the Amplified Negative Sampling method is tested on the Language Model on Penn Tree Bank(PTB) dataset (MSM93) to demonstrate its performance. We use one of the commonly tested RNN Language model(ZSV14) with the medium regularized settings as in (BR17). The PTB dataset contains 929K training, 73k validating and 82k testing words. We keep the original settings of the RNN model and set the hidden dimension as 650.

The perplexity result is reported in Figure 3.4 and 3.5 for Neg5 and Neg15 experiment. For each negative sample setting, the amplifying factor is set to 1, 3, and 10 respectively. As the amplifying factor got enlarged, the perplexity got decreased. Every experiment converges in around 20 epochs. As shown in figure 3.4, the amplifying factor can effectively decrease the perplexity compared with other models when only a few numbers of samples are allowed. Here the result is drawn

from (BR17) to demonstrate the Neg5 result with amplifying factor 10 can compete with (BR17)'s 10 sample result.

Further experiments are made on the effect of the amplifying factor. As shown in Figure 3.6, when the amplifying factor is increased from 1, the perplexity has a clear decrease until 10 and the perplexity starts to increase when the amplifying factor keeps enlarging. The best model performance will be obtained when the amplifying factor is configured around 10. An explanation based on formula (10) is the convergence condition is t(epoch) tends to infinity. However, in the experiment, the training process will be stopped as the performance doesn't change anymore or the model got trapped in the saddle point. In this case, the model won't converge to the optimum value and a large amplifying factor will also change the convergence point to a non-optimum point.

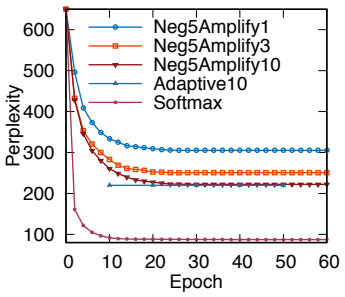


Figure 3.4: PTB: Negative sampling 5.

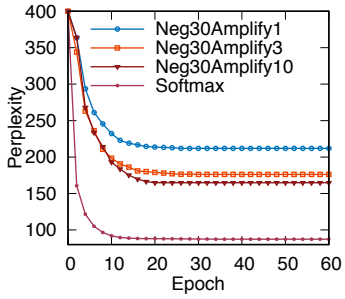


Figure 3.5: PTB: Negative sampling 15.

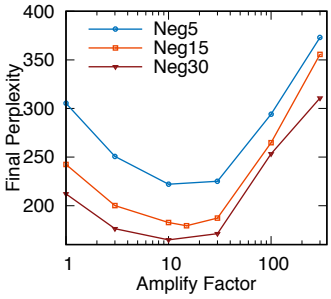


Figure 3.6: Different amplifying factor.

∂

3.5 Related Work

Softmax Function Approximation This line of work includes a variety of techniques to approximate the softmax function. One line of works is the importance sampling technique that estimates the normalizing constant of softmax such as (BS03; BS08; BR17; RCY19). Another important technique is the hierarchical softmax method that utilizes tree structure to reduce the computational complexity to approximate the softmax function (MB05; MH09; STP17). These methods are stick to compute the softmax function efficiently with either a few samples or a

tree computational structure to reduce the computation complexity. The alternatives of softmax function are explored such as spherical softmax (VDB15), Sparsemax (MA16). For this class of method, the computational efficiency for extremely large output classes is still an issue.

Large-class Classification One-class classification(OCC) problem has been investigated in recommender system research (PZC08; YBL17). The OCC problem has a large output class and negative examples are extracted from it. However, the optimization goal for these types of problems is minimizing the classification error instead of learning a distributed representation.

Theoretical Analysis for Contrastive Learning In the NCE (GH10), the author proves the Noise-contrastive estimation is a consistent, asymptotic estimator for given probability density function which utilizes supervised learning to learn the distribution parameter. (MC18) discusses the conditional NCE's statistical property for learning the distribution parameter. These works didn't consider the representation learning and the quality of the learned representations. (AKK19) attempts to discuss the generalization error of distributed representations for a binary classification task. However, this task cannot cover the various kinds of downstream tasks.

3.6 Conclusion

In this paper, we proposed *amplified negative sampling*, a new sample-efficient method for training multi-class classifiers with a large output-class size. Our proposed method was based on our rigorous mathematical analysis. Our extensive set of experiments demonstrated that amplified negative sampling gives us the best of both worlds: It leads to the higher-accuracy model of a larger sample size without paying its high computational cost. Given its simplicity, strong theoretical foundation, and experimental effectiveness, we believe our proposed method will be an important extension to the widely-popular technique that results in meaningful improvements in practice.

CHAPTER 4

sdLDA: Scalable Disk-based LDA with Sparse Prior Sampling

Latent Dirichlet Allocation (LDA) is one of the most popular probabilistic topic models used for diverse classes of applications. While highly effective, one important limitation of LDA is the high memory footprint of its inferencing algorithm, making it difficult to scale to a large dataset. In this paper, we propose *sdLDA*, a highly-scalable disk-based LDA that (1) leverages the abundant space available in the disk to reduce its main-memory footprint and (2) preserves the sparsity of the extracted topics *during sampling* to improve both memory efficiency and sampling complexity of the inferencing algorithm. Our extensive experiments show that *sdLDA* scales to datasets that are two orders of magnitude larger than what existing main-memory-based algorithms can handle and keeps comparable performance compared to existing main-memory-based LDAs in terms of running time. Finally, *sdLDA* is agnostic to data distribution, hence can be straightforwardly adopted as a single-node LDA algorithm for a distributed LDA systems, thus making LDA scale to even further under a distributed architecture.

4.1 Introduction

Statistical topic models were initially developed for analyzing textual datasets, but they are now being used for a wide range of applications, from mining text articles and analyzing image contents, to extract meaningful patterns from genome sequences. ”Topics” in topic models originally meant the topical categories that a particular text or word belongs to, but in general, topics are *latent causes* or *hidden variables* behind any observed data, which can be utilized to identify implicit correlations among observed data. The rise of ultra-large datasets in the Internet era, including online customer product-purchase history, social-network user-interaction trace, and large-scale

historical news-article archive, fueled the increasing demand and development of various statistical topic models. Unfortunately, the ever-growing size of data makes scalability a major bottleneck to an even wider adoption of topic models.

Latent Dirichlet Allocation (LDA) is a highly popular topic model used for a variety of tasks, such as document topic extraction, social network analysis, and collaborative recommendation because of its high-quality output (CBH13; BTS14; YSC13). It also plays a vital role in developing advanced statistical topic models because the algorithmic and performance improvements to LDA can often be migrated to other topic models as well. In this paper, we take LDA to illustrate our approach to improving the scalability of topic models.

A large body of work exists to tackle the scalability issue of LDA. At a high level, existing work can be categorized into two main classes:

1. *Improving the LDA inferencing algorithm*: For this type of work, researchers are attempting to reduce the running time of the LDA inferencing algorithm by examining and optimizing the mathematical properties of the algorithm (YMM09; HBB10; LAR14).

2. *Developing a parallel and distributed LDA*: For this type of work, researchers focus on identifying ways to run the LDA inferencing algorithm on a large cluster of machines in parallel (YGH15; YHY15; NSW07; SWA09; SN10; YZS17).

One important observation has been made by us for all previous work: Existing LDA algorithms rely on main-memory-resident data structures whose size grows linearly to the input data, so there exists a clear upper limit to the size of dataset that LDA can handle. While the parallel and distributed approaches are helpful in this regard because LDA can be run on multiple machines, each node in a parallel cluster is still bound by its main-memory size in its scalability, so its applicability is limited only to those with sizable hardware infrastructure when the input data becomes reasonably large. Because of this scalability problem of LDA, Mikolov et al. describe LDA as "computationally very expensive for large datasets" in their 2013 papers on word2vec (MCC13a; MYZ13), and do not conduct any comparison study to LDA (MYZ13).

In this paper, we take a new approach to improving the scalability of LDA by focusing on

the algorithm's *memory footprint* and develop a solution, called *sdLDA*, that is *compatible with existing approaches*. The basic insight behind our approach can be summarized as two key ideas:

1. *Disk-based algorithm*: Through careful analysis of the LDA inferencing algorithm, we push the data used by the algorithm into the disk as much as possible, leaving only the data critical for performance in main memory. An average machine has disk space two orders of magnitude larger than main memory (1TB vs 16GB), so this can potentially enable LDA to handle a dataset several orders larger than what is possible today. As we show in our experiments, pushing data into the disk has a negligible impact on running time (less than 20%) if implemented carefully.

2. *Compact data representation*: We ensure that the sparsity of LDA data (YMM09) is preserved throughout the LDA inferencing algorithm by a technique called *sparse prior sampling* so that in-memory data can be compactly represented and the complexity of the inferencing algorithm is reduced. Our experiments show that this leads to another order of magnitude decrease in the size of in-memory data and about 2x decrease in running time.

Through a careful combination of optimizations described above, *sdLDA* achieves more than two orders of magnitude improvement in scalability, while its running time is about half that of a popular LDA implementation. Since the ideas behind *sdLDA* are agnostic to data distribution and parallel execution, it is straightforward to adopt *sdLDA* as a single-node LDA algorithm of a distributed LDA system such as (YGH15; YHY15; NSW07; SWA09; SN10), thus making LDA scale to even larger datasets under a distributed setting.

In summary, we make the following contributions in this paper: (1) We propose the first disk-based LDA algorithm that utilizes the plentiful disk space during the LDA inferencing process so that we can significantly reduce the algorithm's memory footprint. (2) We propose various optimizations for the LDA inferencing algorithm so that our disk-based algorithm exhibit the same or even better running-time performance compared to in-memory algorithms. (3) We conduct extensive experiments demonstrating the effectiveness of our proposed techniques. Again, our proposed approach is compatible with existing LDA optimization techniques and can be utilized *in addition*

to these existing techniques and can be adopted as a single-node LDA implementation under a distributed setting to further improve its scaling property.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the background of LDA and Gibbs sampling. In Sections 3 and 4, we describe our *sdLDA* algorithm. In Section 5, we present our experimental results. In Section 6, we discuss the related work and conclude the paper in Section 7.

4.2 related works

The original Latent Dirichlet Analysis (LDA) model was introduced by Blei et. al in (BNJ03b) and it used varational inference method. Subsequently, Gibbs sampling was introduced for estimating the latent variables in (GS04; SC11) and quickly adopted due to its clear form and simple implementation. In order to improve the scalability of Gibbs sampling method for LDA, multiple optimization methods have been explored by research community(YMM09; YHY15; LAR14; TNW07; TKW08; HBB10; MHB12; FBD13; WTK12; HBB10; ZCL13; SN12; LSJ09; YXQ09). Such attempts can be placed into roughly two categories:

4.2.1 Distributed Computation

Distributed and parallel computation is one major direction which has been explored by researchers, which tries to deploy LDA on multiple nodes in a cluster. Several attempts (NSW07; SWA09; SN10; XHD15; WBS09; ZBA12; AAG12; WZS15; YHY15; YZS17) have been done to investigate the parallel version of the LDA model. In (NSW07), the author proposed a synchronous parallel version of LDA, which uses a central master node to coordinate several workers to obtain a higher speed at the same accuracy of the standard LDA. In (SWA09), the author proposed an asynchronous algorithm to learn the potential topics statistically as accurate as those learned by the standard LDA. In (SN10), the author proposed YahooLDA, a parallel and distributed LDA computing framework which utilizes the multi-core computing resources. Afterwards, several distributed systems have been built such as Petuum(XHD15), Peacock(WZS15) and Angel(YZS17).

There are also some efforts to utilize GPU to improve the efficiency of LDA computation such as magnitude.There are also (LCC17). Although sometimes these systems takes several sampling optimization, the distributed and parallel processing play the central role for this type of system.

Our *sdLDA* is agnostic to the deployment of the input dataset and sampling computation, so it can be used almost as a "drop-in replacement" of the standard LDA algorithms used by these approaches to further improve their scalability.

4.2.2 Sampling Optimization

The optimization the sampling method of LDA(PNI08; YMM09; LAR14; CLZ16; YGH15; YZS17) is another main research thread. For this category of works, the major techniques are sparse-aware method and Metropolis-Hasting based algorithms. In (PNI08), the author proposed an optimization on conventional Gibbs sampling method, which has been shown to be more efficient in the experiment. In (YMM09), the first sparse-aware algorithm was proposed. SparseLDA used a novel decomposition method to split the sampling formula into three part. Based on the sparsity of each part, it successfully optimized both the running time and memory consumption. This algorithm not only uses substantially less memory but also gains higher processing speed than the standard LDA. In (LAR14), the author pointed out that as the corpus size grows, the topic sparsity of each word decreases, which is what we also observe in our own experiments. However, based on the observation that the topic sparsity of each document is still preserved, the authors proposed aliasLDA with a different decomposition scheme. It utilizes the topic sparsity in documents and introduced a Metropolis-Hasting-Walker sampler to get samples efficiently from the stable topic distribution of each word. This work take the heritage of the sparse-aware algorithm in (YMM09) and further proposed the Metropolis-Hasting based algorithm. In (YHY15), the author introduced the Fenwick tree, a data structure which has a balanced performance in sampling and updating the underline distribution. By combining the Fenwick tree with the LDA algorithm, the authored achieved performance gain. In (YGH15), the author proposed several optimizations, including a new Metropolis sampling method, a new decomposition scheme, and a new storage structure based on the words distribution and a new parallel implementation. In (CLZ16), the author analyzed pre-

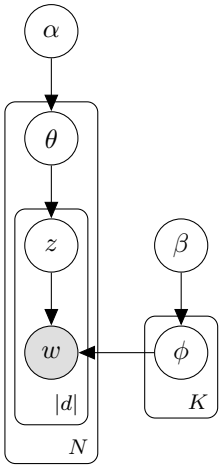


Figure 4.1: Latent Dirichlet Allocation

vious efforts in optimizing the LDA computation and proposed an memory-efficient method to reduce the cache missing so that it improve the efficiency by one order of magnitude.

4.2.3 Our Observation

Existing research efforts mainly focused on improving the computation efficiency of LDA computation while the memory consumption is largely ignored. However, based on our experiment, the huge memory consumption of existing models has been a major bottleneck to further extent the scalability of existing LDA computation methods.

In general, the approach taken by our work is compatible with the existing body of work, because we mainly focus the memory footprint of an LDA algorithm. By carefully analyzing the data access locality of the LDA sampling method and the property of its data structures, we identify optimal ways to push data into the disk, so that we can maximize the amount of data that can be handled by a single machine.

4.3 Background

In this section, we provide a brief overview of the *Latent Dirichlet allocation* (LDA) model and its Gibbs-sampling-based inferencing algorithm as a basis for our future discussion.

4.3.1 Latent Dirichlet Allocation

When we are trying to write a document, we have several potential topics to talk about in our mind. For each topic, we will elaborate it by picking some words to describe the details of the topic. LDA formalizes this intuitive generative process by representing documents as random mixtures over topics and each topic as a multinomial distribution over words (BNJ03b).

In LDA, a corpus $\mathbf{c} = \{d_1, d_2, \cdots, d_N\}$ consists of N documents, which are generated from a vocabulary of V words and K topics. Figure 4.1 describes the LDA's document generative process using the plate notation. That is, for every document d_i $(1 \leq i \leq N)$, a K-dimensional topic probability vector $\theta_{z|d_i} = p(z|d_i)$ (with $\sum_{z=1}^{K} p(z|d_i) = 1$) is drawn from a Dirichlet distribution with parameter α. Given this topic vector, at each word position j of d_i, a single topic z_{ij} is randomly drawn among the K topics through a multinomial process with the probabilities $\theta_{z|d_i}$. Separately, every topic z $(1 \leq z \leq K)$ is associated with a topic-to-word probability distribution $\phi_{w|z} = p(w|z)$ (with $\sum_{w=1}^{V} p(w|z) = 1$), which is drawn from a Dirichlet distribution with parameter β. Given $\phi_{w|z} = p(w|z)$ and the topic z_{ij}, the word at the $j$the location of d_i, w_{ij}, is drawn through a multinomial process with probabilities $\phi_{w_{ij}|z_{ij}} = p(w_{ij} = w|z_{ij} = k)$. The key symbols used throughout this paper are summarized in Table 4.1 for ease of reference.

Notation	Description
$\theta_{z\|d}$	Per-document topic distribution
$\phi_{w\|z}$	Per-topic word distribution
α, β	Parameters of the Dirichlet priors on multinomial distributions
w_{ij}	The jth word in ith document
z_{ij}	The topic of w_{ij}
N	Number of documents in the corpus
V	Number of words in the vocabulary
K	Number of topics

Table 4.1: Notations description

The goal of *an LDA inferencing algorithm* is to identify the optimal parameter values of the LDA model, $\theta_{k|d_i}$, $\phi_{w|z}$ and z_{ij}'s, that *maximize the probability of our corpus* **c** under the LDA model.

4.3.2 Gibbs Sampling in LDA

Gibbs sampling is one of the most prevalent and successful inferencing algorithms to infer the parameters of the LDA model. Gibbs sampling is a Markov Chain Monte Carlo (MCMC) method that iterates over all word positions w_{ij} repeatedly to resample its topic assignment z_{ij} based on the z_{ij} values at other word positions.

More precisely, during the initialization stage of Gibbs sampling described in Algorithm 1, every topic z_{ij} ($z[i, j]$ in the algorithm) of w_{ij} ($w[i, j]$ in the algorithm) is initially set to a random value chosen uniformly between $[1, K]$. The Initialize() function takes the entire corpus w as its

Data: Corpus word sequence w
Result: Initial topic assignment z, Topic assignment statistics WZ and DZ
// initialize topic assignment statistics
for $1 \leq w \leq V$ *and* $1 \leq z \leq K$ **do**
  $WZ[w,z] = 0$;
end
for $1 \leq i \leq N$ *and* $1 \leq z \leq K$ **do**
  $DZ[i,z] = 0$;
end
// assign $z[i,j]$ to a random value **for** i=*1 to* N **do**
  for j=*1 to* $|d_i|$ **do**
    $z[i,j] \sim \text{uniform}(1,K)$;
    $WZ[w[i,j],z[i,j]]$++;
    $DZ[i,z[i,j]]$++;
  end
end

Algorithm 1: Initialize()

input, where $w[i,j]$ represents to the jth word of the ith document, w_{ij}. Given this input, it (1) assigns the topic $z[i,j]$ to a random value between $[1,K]$ and (2) collect statistics on this initial topic assignment. In particular, the array $WZ[w,z]$ ($1 \leq w \leq V$ and $1 \leq z \leq K$) counts the number of times that the word w is assigned to the topic z. The array $DZ[i,z]$ ($1 \leq i \leq N$ and $1 \leq z \leq K$) counts the number of times words in document d are assigned to the topic z.

After all $z[i,j]$'s are initialized to a random value, Gibbs sampling repeatedly iterates over all $z[i,j]$ values to resample them to values between $[1,K]$. This new assignment is performed according to the following probabilities

$$p(z[i,j] = k) \propto (DZ[i,k] + \alpha)\frac{WZ[w[i,j],k] + \beta}{\sum_{w=1}^{V}(WZ[w,k] + \beta)} \tag{4.1}$$

Roughly, above resample probability for z_{ij} considers two factors in assigning a topic $z_{ij}k$ to the word w_{ij}: (1) $DZ[i,k]$, the frequency of the topic k assignment within the document i and (2) $WZ[w[i,j],k]$, the frequency of the topic k assignment for the word w_{ij}.

The Resample() function in Algorithm 2 describes this Gibbs resampling process, where the function gibbs-sample() returns an random integer value between $[1,k]$ according to the probability described in Equation 4.1. Note that the statements before and after the call to gibbs-sample()

ensure that the topic assignment statistics WZ and DZ are updated according to the old and new values of $z[i, j]$.

Data: Corpus word sequence w, current topic assignment z, topic assignment statistics WZ and DZ
Result: New topic assignment z, new topic assignment statistics WZ and DZ
for *i=1 to N* **do**
  for *j=1 to* $|d_i|$ **do**
    $WZ[w[i, j], z[i, j]]$- -;
    $DZ[i, z[i, j]]$- -;
    $z[i, j] \sim (DZ[i, k] + \alpha) \frac{WZ[w[i,j],k]+\beta}{\sum_{w=1}^{V}(WZ[w,k]+\beta)}$;
    $WZ[w[i, j], z[i, j]]$++;
    $DZ[i, z[i, j]]$++;
  end
end

Algorithm 2: Resample()

To obtain a final assignment of $z[i, j]$ values under the Gibb sampling method, existing algorithms call Resample() function either a fixed number of times (typically in the order of hundreds) or until the $z[i, j]$ values converge (i.e., the majority of $z[i, j]$ values do not change between iterations). The property of MCMC method guarantees that the Gibbs sampling eventually converges to a stable distribution which is a local optimal value.

As we can see from the above algorithms, all key data, $w[i, j]$, $z[i, j]$, $WZ[w, z]$ and $DZ[i, z]$, are accessed repeated during the Gibbs sampling process. For this reason, existing implementations of Gibbs LDA keep all these data *in memory* for efficient retrieval of their values. This *in-memory* implementation leads to a high memory footprint and is one of the main bottlenecks in scaling LDA to a large dataset. In the next section, we explore whether and how we can push most of these data *to disk* during the Gibbs sampling process without a meaningful hit on the performance of the sampling process.

4.4 *sdLDA*: Scalable Disk-Based LDA

The key intuition behind the scalability of our *sdLDA* algorithm is two-fold: (1) we push out data into the disk as much as we can as long as pushing the data out does not lead to a noticeable

performance hit and (2) if it is essential to keep certain data in memory, we design an alternative sampling process so that we can preserve and exploit the desirable characteristics of the data to minimize their memory footprint and sampling complexity. In this section, we elaborate more details of our *sdLDA* algorithm.

4.4.1 Size Estimation of Key Data Structures

Structure	Size
$w[i,j]$	$O(N)$
$z[i,j]$	$O(N)$
$DZ[i,z]$	$O(NK)$
$WZ[w,z]$	$O(VK)$

Table 4.2: The property of each data structure in LDA sampling

As we saw from the earlier algorithms, Gibbs LDA requires access to four key data structures, $w[i,j]$, $z[i,j]$, $DZ[i,z]$ and $WZ[w,z]$ during sampling. How do they grow as the input data grows?

In Table 4.2, we show how the sizes of these four "arrays" depend on the key parameters of the input dataset, the number of documents N, the size of vocabulary V and the number of topics K. The correctness of these size formulae is relatively straightforward to check. For example, $w[i,j]$, $z[i,j]$ and $DZ[i,z]$, grows linearly to N, the number of documents in the corpus, since $w[i,j]$, $z[i,j]$ and $DZ[i,z]$ are all indexed by the document number i, and thus has as many "rows" in the array.

Dataset	document N	dictionary V	words/doc	words $w[i,j]$	$DZ[i,z]$	$WZ[w,z]$
NIPS	1,500	12,419	1,267	1,900,000	750,000	6,209,500
KOS blog	3,430	6,906	136	467,714	1,715,000	3,453,000
Enron	39,861	28,192	161	6,400,000	19,930,500	14,050,100
NYtimes	300,000	102,660	333	100,000,000	150,000,000	51,330,000
PubMed	8,200,000	141,043	89	730,000,000	4,100,000,000	70,521,500

Table 4.3: Size of Different Data Structures for real-world datasets. $K = 500$ topics is assumed for $DZ[i,z]$ and $WZ[w,z]$

In Table 4.3, we show the relative sizes of the four arrays on several real-world datasets (BL13). The first three column represent the dataset name, the number of documents N, and the number

of unique words V in the dataset. The fourth column represents the average number of words per document. The fifth column shows the total word count in each corpus, which is equivalent to the sizes of $w[i,j]$ and $z[i,j]$. The sixth column represents the size of the document-topic array $DZ[i,z]$, where we assume $K = 500$ as the number of topics. The sixth column represents the size of the word-topic array $WZ[w,z]$.

From the table, it is clear that as the corpus size grows, the sizes of three tables $w[i,j]$, $z[i,j]$ and $D[i,z]$ grows roughly "linearly." Therefore, it will be critical to push these three arrays out to the disk to handle a large-scale input data. We also see that the fourth array $WZ[w,z]$ is generally much smaller than the other three arrays, especially when the corpus size is large, but its size is still significant. For example, when the vocabulary size V is 1,000,000 and when the number of topics $K = 1,000$ (reasonable assumptions for Web-scale data), $WZ[w,z]$ array has one billion entries.

4.4.2 Disk LDA: Pushing Data To Disk

One distinctive characteristic of a magnetic disk is that there exist at least two orders of magnitude difference in the access time between a sequential and a random access to data. Therefore, in exploring the possibility of pushing (part of) LDA data to the disk, it is critical to investigate how the LDA data are accessed. Is the access sequential or random? Does the access exhibit any locality?

Structure	Access Pattern	Locality	Sparsity
$w[i,j]$	sequential	local	No
$z[i,j]$	sequential	local	No
$DZ[i,z]$	random	local	Yes
$WZ[w,z]$	random	global	Yes

Table 4.4: The property of each data structure in LDA sampling

Fortunately, in Algorithm 2, we see that data in both $w[i,j]$ and $z[i,j]$ are accessed *sequentially*, in the increasing order of i and j. This makes them ideal candidates to push out to the disk; if we

store them in the increasing order of i and j, we can ensure that the entries in the two arrays are accessed sequentially, avoiding a significant performance penalty from a random IO.

Unfortunately, the access to the third array $DZ[i,z]$ is not sequential, since the array is accessed through $[i,z[i,j]]$, not $[i,j]$, breaking the sequentiality for the second index for the array. However, these "random accesses" are still limited within the *same document*, exhibiting intra-document locality; that is, all accesses to $DZ[i,z[i,z]]$ share the same outer-loop index i within the inner-loop of j. This intra-document locality suggests that if the Resample() algorithm "caches" the entire "ith row" of $DZ[i,-]$ values to the main memory in one batch IO before it enters the inner loop, it can avoid the random IO penalty, because the rows of $DZ[i,-]$ are accessed sequentially in the increasing order of i. Once cached, an access to any $DZ[i,-]$ becomes an in-memory operation, meaning that there is a negligible performance penalty for a random access.

The access pattern to the fourth $WZ[w,z]$ array exhibits neither sequentiality nor intra-document locality; the array WZ is accessed through $w[i,j]$ and $z[i,j]$ values, which can be any random word and topic. Therefore, pushing this array to the disk results in a significant performance penalty; every access to this array is likely to incur a random disk IO. In the next subsection, we discuss how we can further reduce the size of this array by exploiting the fact that in most LDA settings, this array is *sparse*.

In Table 4.4, we summarize what we discussed in this section. The first three arrays, $w[i,j]$, $z[i,j]$, and $DZ[i,z]$ exhibits either sequentiality or intra-document locality, so they are suitable for pushing out to the disk. The last column of the table, data sparsity, is the topic of our next discussion.

4.4.3 Preserving Sparsity of $WZ[w,z]$

Since the entries in the fourth $WZ[w,z]$ array is accessed randomly during Gibbs sampling, so we are unable to push them out to the disk unless we incur a significant performance overhead. Fortunately, it is well known that the final values in the $WZ[w,z]$ array are mostly zero and the array is sparse (YMM09).

Intuitively, the sparsity of the $WZ[w,z]$ array (and the $DZ[i,z]$ array as well) is not hard to

imagine; most English words are related to just a few topics, not hundreds. Since the entries in $WZ[w,z]$ represent the number of times that a word w is assigned to the topic z, each row of the matrix will have only a few nonzero values corresponding to the topics that the word is related to, once LDA inference algorithm is over.

In principle, we can exploit this sparsity of $WZ[w,z]$ to significantly reduce the memory footprint of $WZ[w,z]$; we use a data structure optimized for sparse data so that we do not have to explicitly store zero-valued entries. Unfortunately, using a sparse representation for the $WZ[w,z]$ array is not applicable for the existing Gibbs LDA algorithms, because the $WZ[w,z]$ *array is in fact dense after it is initialized.*

More precisely, according to Algorithm 1, every $z[i,j]$ is assigned to a integer value between $[1,K]$ *with equal probabilities*. This uniform sampling of the initial topic assignment implies that, by the time Initialize() is over, each word w is assigned to each topic z roughly the *same number of times*, leading to nonzero values for most entries in $WZ[w,z]$. Of course, as the Gibbs sampling continues and the $z[i,j]$ values are resampled, $WZ[w,z]$ will gradually transform to a sparse array, but the initial high density makes this array unsuitable for sparse representation. As a solution to this high-density initialization problem, we now propose *prior learning with subset sampling*.

4.4.3.1 Prior Learning with Subset Sampling

If we want to reduce the memory footprint of $WZ[w,z]$ by exploiting its eventual sparsity, it is critical to redesign the *initial sampling procedure*, so that the array remains sparse from the beginning, not just at the end. That is, when we assign the initial topic z_{ij} at each word position w_{ij}, the assigned topic should be sampled primarily from a small subset of K, not uniformly at random. How can we achieve this when we have no information on the topic of w_{ij}? Where can we obtain the "prior topic distribution" of each word before we start our Gibbs sampling initialization?

To obtain an approximate "prior topic distribution" of each word, we take a two-step approach: (1) In the first *subset sampling* stage, we run the complete Gibbs sampling procedure on *a small subset of the input dataset* so that we can learn the prior topic distribution of words from in the subset. (2) Once we obtain an approximate prior word-topic distribution from the small sample, we

use this approximate distribution for the initialization stage of the main Gibbs LDA, so that each word comes primarily from a few prior topics, not uniformly at random.

Algorithm 3 describes our modified Initialize() function, referred to as Initialize-with-Prior-Learning(), that incorporates this idea. Before the algorithm tries to initialize $z[i, j]$ values on the full input dataset D, it first takes a small sample D' from D. On this small sample, it runs the Gibbs LDA algorithm to obtain the word-topic assignment count $WZ'[i, z]$ within the subset, which will be used as an approximate initial word-topic distribution in the next stage.

Once we obtain WZ', we start initializing $z[i, j]$ at each word location of the full input dataset D. Note that in this new algorithm, the topic assignment $z[i, j]$ probability changes depending on whether the word $w[i, j]$ is in WZ' (meaning that the word appeared in the subset D') or not. If it is, we use the approximate word-topic count $WZ'[i, z]$ as the "topic prior" of the word $w[i, j]$ and use a sampling probability proportional to this count. If $w[i, j]$ is not in WZ' (meaning that the word did not appear in the subset D'), we have no information on the topic distribution of $w[i, j]$, so we adjust the topic sampling probability only based on "document-topic" distribution of the current document which we have obtained from the topic assignment of known words. This separate topic sampling procedure ensures that the initial topic assignment of each word is biased towards a small number of topics and exploit our "prior knowledge" to the best extent possible.

Later in the experiment section, we will investigate the impact of our prior learning with subset sampling in the memory footprint of the algorithm, the sampling convergence time and the accuracy of the final results.

4.4.4 Minimizing Sampling Complexity

Our scalable disk-based LDA with prior learning (*sdLDA*) enables running the highly successful Gibbs LDA algorithm on a large-scale dataset. In particular, because of its tiny memory footprint, it is now possible to run the LDA algorithm with many topics – with K in the order of thousand – to obtain a detailed topic analysis of the input dataset. When we ran our *sdLDA* with this setting, we encounter another performance bottleneck: linear increase of sampling complexity on topic count K.

Data: Corpus Set $D = w[i, j]$
Result: topic assignment $z[i, j]$, $DZ[i, z]$, $WZ[i, z]$
// obtain word-topic prior from a small sample;
Let D' be a small random sample of D;
Run Gibbs LDA on D';
Let $WZ'[w, z]$ be the word-topic count from D';
// initialize topic assignment stats;
for $1 \leq w \leq V$ *and* $1 \leq z \leq K$ **do**
  $WZ[w, z] = 0$;
end
for $1 \leq i \leq N$ *and* $1 \leq z \leq K$ **do**
  $DZ[i, z] = 0$;
end
// initialize $z[i, j]$ using topic-word prior $WZ'[w, z]$;
for *i=1 to N* **do**
  for *j=1 to* $|d_i|$ **do**
    if $(w[i, j] \in WZ'[w, z])$ **then**
      $z[i, j] \sim \frac{WZ'[w[i,j],k]+\beta}{\sum_{w=1}^{V}(WZ'[w,k]+\beta)}$;
      $WZ[w[i, j], z[i, j]]$++;
      $DZ[i, z[i, j]]$++;
    end
  end
  for *j=1 to* $|d_i|$ **do**
    if $(w[i, j] \notin WZ'[w, z])$ **then**
      $z[i, j] \sim (DZ[i, k] + \alpha)$;
      $WZ[w[i, j], z[i, j]]$++;
      $DZ[i, z[i, j]]$++;
    end
  end
end

Algorithm 3: Initialize-With-Prior-Learning()

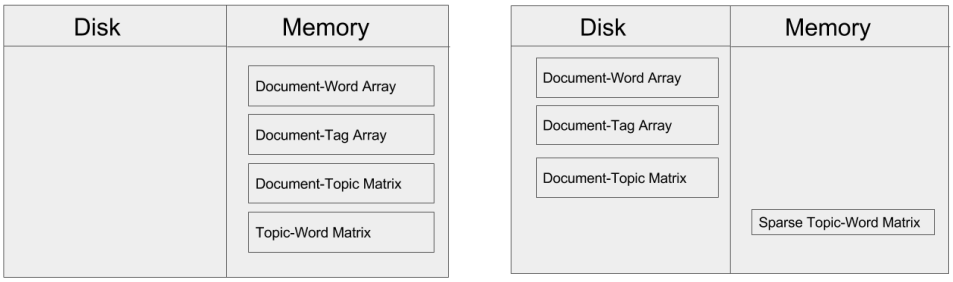
To understand why, we repeat the equation for the topic sampling probability $z[i, j]$ here:

$$p(z[i, j] = k) \propto (DZ[i, k] + \alpha) \frac{WZ[w[i, j], k] + \beta}{\sum_{w=1}^{V}(WZ[w, k] + \beta)} \tag{4.2}$$

Since we need to assign $z[i, j]$ to k according to the above probability, we need to compute the values for the above equation for *every topic*. This implies as we increase the K values, the topic sampling times gets proportionally longer.

In (YMM09), Yao et al. proposed a way to break this linear dependence of the topic sampling complexity on K by decomposing the topic sampling probability formula multiple parts and computing each part of the formula separately. In our *sdLDA* implementation, we adopt this general approach proposed in (YMM09) and use a decomposition that is suitable for our *sdLDA* algorithm.

4.5 *sdLDA*: Implementation



(a) Memory-based LDA

(b) sdLDA

Figure 4.2: Comparison of Memory-LDA and sdLDA. The figure above displayed the difference of the storage structure for LDA. All the storage structures are optimized based on their property. Traditionally, all the computation structures are located in main memory. In the sdLDA, the original document and topic assignment array are stored in the Disk using the disk-based array. The state information for topics assigned in each document is stored in the disk by using the disk-based matrix. The topic stat information for each word is using a sparse matrix as the storage structure to minimize the memory footprint.

4.5.0.1 Sparse sampling

The following equation shows how we decompose Equation 4.2 for the *sdLDA* algorithm:

$$p(z_{ij} = k|\mathbf{z}_{-ij}) \propto WZ[w[i,j],k]\frac{DZ[i,k]+\alpha}{\sum_{w=1}^{V}(WZ[w,k]+\beta)} + \beta\frac{DZ[i,k]+\alpha}{\sum_{w=1}^{V}(WZ[w,k]+\beta)} \tag{4.3}$$

Note that the first term has the factor $WZ[w[i,j]k]$. Given the sparsity of $WZ[w,k]$, this term will be zero for most k, implying that we have to calculate this term only for a few topics. The second

term has a nonzero value for most topics, but fortunately, since the term has the factor β — whose value is typically much smaller than $WZ[w,k]$ — the majority of the probability mass lies in the first term. (YMM09).

To compute the sampling probability of topic k, we also we need to calculate the sum of the right-hand side of Equation 4.3 over all topics, which is given below:

$$q = \sum_k WZ[w[i,j],k] \times \frac{DZ[i,k]+\alpha}{\sum_{w=1}^{V}(WZ[w,k]+\beta)} \tag{4.4}$$

$$s = \sum_k \beta \times \frac{DZ[i,k]+\alpha}{\sum_{w=1}^{V}(WZ[w,k]+\beta)} \tag{4.5}$$

Note that once we compute the above two sums s and q, we need to update the s and q values *only twice per each word* $w[i,j]$, not k times; every resampling of $z[i,j]$ changes the frequency counts of just two topics (the old and new topics of $z[i,j]$). Therefore the computational complexity of updating the two sums are $O(1)$, not $O(K)$.

Given these observations, we can naturally split the topic sampling into 2 buckets. Based on Equations 4.4 and 4.5, we draw a random number $U \sim \text{uniform}(0, q+s)$. If the random number U is smaller than threshold q, we only need to check all the non-zero entries in the first term of Equation 4.3. If U is larger than q, we need to check all k entries of the second term of Equation 4.3, which involves $O(k)$ operations.

Given our discussion in the previous section, we push three data structures, $w[i,j]$, $s[i,j]$ and $DZ[i,z]$, to the disk, while keeping $WZ[w,z]$ in main memory as we show in Figure 4.2. We now describe our implementation of these four data structures and our caching strategy for the disk-resident data in more detail.

Variable-length disk-array Since the row lengths of the two arrays, $w[i,j]$ and $z[i,j]$, vary depending on a document, a *variable-length disk-array* class is used to store them in the disk. Their storage structure consists of two parts. The first part is the array D_{length}. This array is an index for each document and also records the length of each document. The second part is the words and topics of each document. Documents are stored consecutively right after another in a single file, as

we show in Figure 4.3.

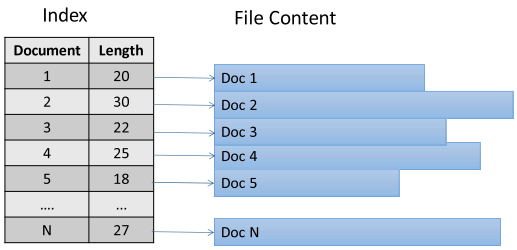


Figure 4.3: data structure of the disk array

Fixed-length disk-array For the document-topic matrix $DZ[i,z]$, a *fixed-length disk-array* class is used to store the data in disk. Essentially, this class is a disk version of a 2D array, where the width of each row is a constant K.

Sparse memory array In order to utilize the sparsity of the word-topic array $WZ[w,z]$, we use the representation described in (YMM09). For each 32-bit integer, the lower 10 bits are used to store the topic of each word and the rest 22 bits are used to store the occurrence of this word. Differently from (YMM09), however, we do not store the entries of each row sorted by their occurrence. Our extensive experiments show that the overhead of keeping the entries sorted actually degrades the overall sampling performance. As the initial number of entries for each row of this array, we use the smaller value between the frequency count of w and the topic count K.

Caching strategy Caching strategy is crucial for the efficiency of the *sdLDA*. Without a proper caching strategy, we will incur a significant slow down in the sampling speed. Our caching strategy is described below.

For the three disk-resident arrays, the caching strategy is to load all entries related to the current document in *one disk IO* before the sampling algorithm enters the inner-loop of Algorithm 2. Similarly, once we finish computing the new values for $z[i,j]$ and $DZ[i,z]$ in the inner loop, we store the new values together to the disk in *one disk IO*. This caching strategy is implemented in two core functions, *load* and *flush*. The *load* operation will load one document into the memory. The *flush* operation will flush the document into the disk. The loaded document will be kept in

memory and not be flushed back until another document is visited or the algorithm ended.

We also ensure that the sequentiality of these data is preserved in the disk throughout the sampling operation so that we can avoid unnecessary random IOs during data access. As we will see in the experiment section later, this caching and storage strategy results in performance comparable to existing memory-based implementations.

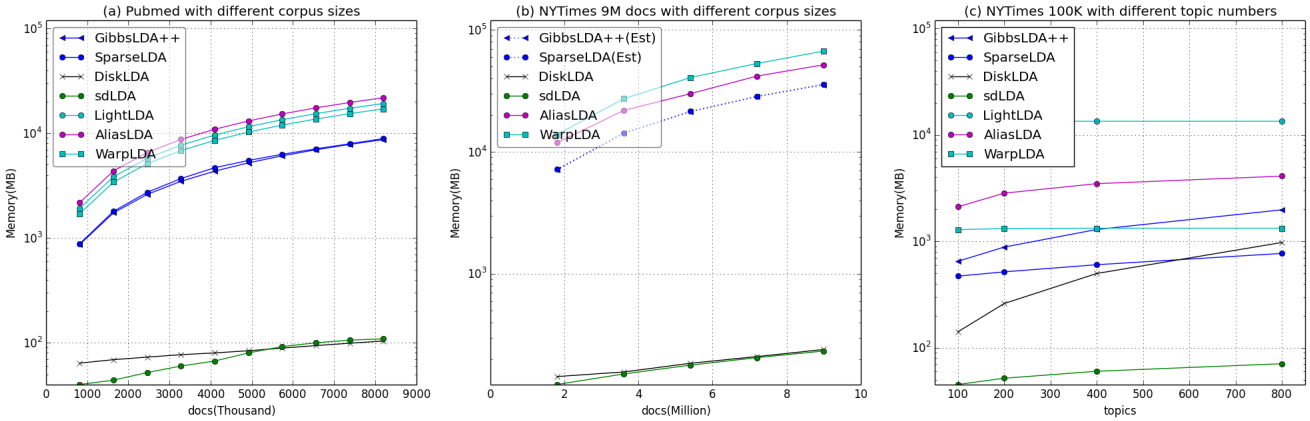


Figure 4.4: Memory consumption experiment for (a) NYTimes dataset with different input file number and (b) NYTimes dataset with a 9M news articles. (c) NYTimes 100K with a different topic number In (a), the sdLDA and DiskLDA show 2 order of magnitude consumption reduction in terms of memory when input file number increase. In (b), the dotted line is estimated memory consumption since GibbsLDA++ and SparseLDA have run out of memory. In (c), the sdLDA shows an almost constant memory size consumption in comparison with DiskLDA and GibbsLDA++.

4.6 Experiment

In this section, we conduct three types of experiments to demonstrate the overall performance of our *sdLDA* algorithm. We compare *sdLDA* against other existing state-of-the-art LDA implementations, SparseLDA (YMM09) and GibbsLDA++ (PN07) on five different public datasets shown in Table 4.3. In order to evaluate the impact of the *prior learning with subset sampling* (Section 4.4.3.1) and *sparse sampling* (Section 4.5.0.1) methods on the memory footprint and running time, we run two versions of *sdLDA*: one with both prior learning and sparse sampling applied (referred to as *sdLDA*) and one *without* either prior learning or sparse sampling applied (referred to as DiskLDA).

All experiments are conducted on a server with 8 cores, 32 Gigabyte memory and a 1 Terabyte magnetic hard disk drive running on Ubuntu 14.04. From the results of these experiments, we will see that *sdLDA* leads to significant improvements both in memory footprint (up to two orders of magnitude reduction in terms of memory consumption) compared to other LDA algorithms.

We first present the memory consumption comparison in Section 4.6.1, then the running time in Section 4.6.2 and finally the convergence rate in Section 4.6.3.

4.6.1 Memory consumption

We first examine how the memory footprints of the various LDA algorithms change as we change (1) the size of the input dataset N and (2) the number of topics K. In both experiments, the sdLDA and the DiskLDA outperform all other algorithms. When the topic number grows sdLDA shows a huge advantage over all other algorithms.

Impact of the input size To evaluate the impact of input size, we use PubMed and an augmented NYTimes dataset with topic value $K = 200$[1]. The PubMed dataset contains 4 million documents and NYTimes dataset contains 9 million documents. The size of these datasets are 4.4Gb and 18Gb respectively.

Figure4.4 (a) depict the input size experiments result on the PubMed dataset. The vertical axis of the figures is logarithmic to include the four graphs that exhibit two orders of magnitude difference. From the first graph, it is clear that the two memory-based LDA algorithms (GibbsLDA++ and SparseLDA) use significantly more memory than *sdLDA*. For example, at $N = 8$ million, LightLDA, AliasLDA, WarpLDA, GibbsLDA++, SparseLDA, DiskLDA, and *sdLDA* use 19.237GB, 21.842GB, 17.099GB, 8.698GB, 8.899GB, 104MB, and 109MB of main memory, respectively, indicating that *sdLDA* leads to two orders of magnitude improvement in memory footprint.

In Figure 4.4(b) show the experiment result with the NYTimes dataset. When the input document size is 1.8 million, the GibbsLDA++ and the SparseLDA consume 7092MB and 7211MB

[1]For larger topic values K, we could not run GibbsLDA++ and LightLDA because they caused out-of-memory error

of memory, the AliasLDA and WarpLDA consumes 11870MB and 13884MB of memory, while DiskLDA and sdLDA consume 144MB and 124MB memory respectively and LightLDA run out of memory in this situation. When 9M documents are all processed, GibbsLDA++ and SparseLDA and other methods are run out of memory, while DiskLDA and sdLDA consume 240MB and 233MB respectively, which further demonstrates the effectiveness of sdLDA in reducing the memory footprint of LDA algorithm.

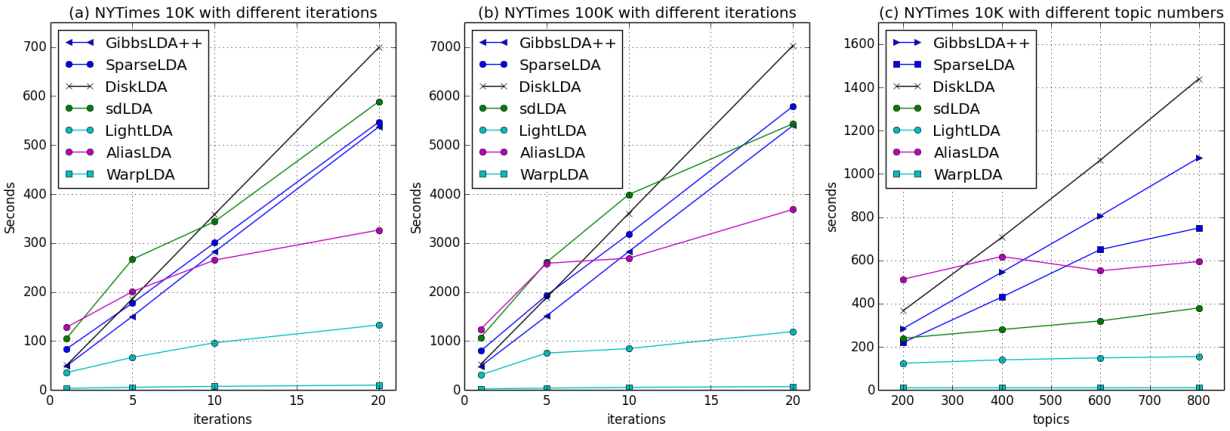


Figure 4.5: Running time with different iterations for NYTimes datasets (a) NYTimes dataset with 10K documents and (b) NYTimes with 100K documents. All four algorithms show very similar running time. In (c), the running time with different topic numbers are depicted. As the topic number grows, the running time of GibbsLDA++ grows linearly. The sdLDA model demonstrates moderate performance over a large topic number K.

Impact of the topic number To measure the impact of topic number, we use the NYTimes 100K documents dataset which is a tailored NYTimes dataset to demonstrate how the memory footprint got affected by the different number of topics.

Figure 4.4(c) depicts the result of how the topic number affect the memory footprint of different algorithms. Clearly, the memory growth rate of *sdLDA* is significantly lower than that of the other six algorithms in the figure, indicating that *sdLDA* is very effective in preserving the sparsity of $WZ[w,z]$ array when the topic number K is large. As we increase the topic number, the memory-size reduction from the prior learning with subset sampling becomes more significant; At $K = 800$, for example, LightLDA, AliasLDA, WarpLDA, GibbsLDA++, SparseLDA, DiskLDA and *sdLDA*

use 13.4GB, 4.1GB, 1.33GB, 2GB, 770MB , 1GB, and 70MB memory respectively.[2]

4.6.2 Running time

In this section, we compare the running time of various LDA algorithms. The experiment is conducted on two subsets of the NYTimes dataset that have 10K and 100K documents, respectively. When the topic number is small, all algorithms will present similar running time. However, when the topic number increases, the sdLDA have demonstrate similar growth rate as LightLDA.

Impact of the number of iterations We use NYTimes 10K and NYTimes 100K dataset to evaluate the relationship between running time and training iterations. The topic number is set to 100. We run the evaluation on the LightLDA, AliasLDA , WarpLDA, GibbsLDA++, SparseLDA, DiskLDA, and sdLDA.

In Figure 4.5(a)(b) we show the wall-clock time of the seven LDA algorithms at the end of each training iteration. In order to make fair comparison, we allow every program run one single thread. From the graph, we see GibbsLDA++, SparseLDA, DiskLDA and sdLDA algorithms finish each iteration roughly at the same time. The Alias LDA use a little bit less time than the above four algorithms. The LightLDA and especially WarpLDA show advantages in running time. However, note the running time of the first iteration of *sdLDA* is consistently higher than others. This is due to the *prior learning with subset sampling* stage (Section 4.4.3.1); we include the time for the prior-learning stage in the first iteration of *sdLDA*, which makes its first iteration time longer than others. This "penalty" of *sdLDA* is quickly amortized over multiple iterations because it is a one-time penalty at the beginning.

Impact of the number of topics The second experiment measures the running time of four algorithms at different topic numbers. In this experiment, we use the NYTimes 10K dataset. We set the topic number K to 200, 400, 600, and 800, and measure the running time at 5th iteration. The result is shown in Figure 4.5(c). In every case, *sdLDA* shows the best running time because of our sparse sampling method. The improvement from sparse sampling becomes more significant as

[2]For this setting, DiskLDA uses more memory than memory-based SparseLDA because when N is small and K is large, $WZ[w,z]$ becomes the largest array among the four. In this case, SparseLDA uses memory more efficiently because of it uses a sparse representation of $WZ[w,z]$.

topic number increases. When $K = 800$, for example, *sdLDA* is 2.5x as fast as SparseLDA, 2x as fast as AliasLDA and 5x as fast as GibbsLDA++.

The reason for the improvement of the efficiency comes from the sparsity of word in each topic and the prior learning process. The sparsity guarantees the fast sampling for each word and the prior learning expedite the converge so that the sparse status can be reached faster.

4.6.3 Convergence rate

Our proposed "prior learning with subset sampling" method (Section 4.4.3.1) preserves the sparsity of the $WZ[w,z]$ array by learning the topic probability distribution of most words using a small sample. This method has another side benefit. Because the initial topic assignments to $z[i,j]$s are likely to be closer to the final assignments, the overall sampling process will converge to the final results more quickly than others. At the same time, since we do not initialize $z[i,j]$'s completely randomly, there is a possibility that the quality of the final topic assignments may not be as good as when we initialize them randomly.

To investigate the effect of our "prior learning" on these two issues, in Figure 4.6, we show the perplexity values of inferred LDA model at each iteration of our two algorithms, one with our prior learning applied (DiskLDA) and one without (*sdLDA*).[3] A lower perplexity value indicates that the inferred model is superior.

From the figure, it is clear that (1) our prior learning improves the overall convergence rate of the sampling process; the initial perplexity value with *sdLDA* is significantly smaller than *DiskLDA* and (2) prior learning has a negligible impact in the final quality of the inferred model. In both graphs, the final perplexity value converges to roughly 1000.

[3]GibbsLDA, SparseLDA and DiskLDA are essentially the same in terms of their sampling process. LightLDA, AliasLDA and WarpLDA didn't design special process to accelerate the convergence. so we only show the result of DiskLDA here.

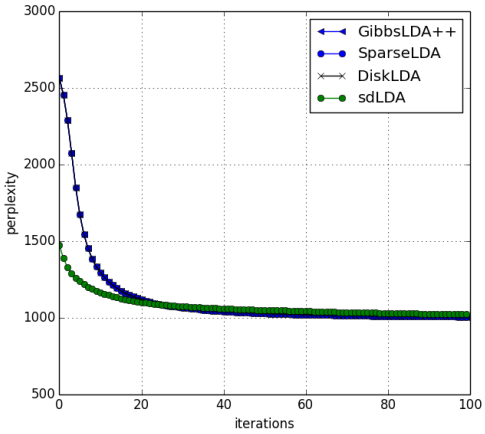


Figure 4.6: the perplexity of GibbLDA++, sparseLDA, DiskLDA, and *sdLDA* during 100 iterations on NIPS dataset. The perplexity of first three initially starts at 2500 and eventually converges to around 1000. For *sdLDA*, perplexity starts at around 1500 and converges to 1000.

4.6.4 The effect of subset sampling

Our subset sampling method can effectively reduce the initialization overhead by select a small portion subset of training example to generate an approximate distribution. Utilizing the approximated distribution can assign the topic label in a non-uniform way so that avoiding the dense topic-word matrix generated by the randomized topic initialization.

In order to investigate the best sampling rate, we conduct the experiment of using different sampling rate and test the perplexity of the result LDA model. We run the experiment on NIPS dataset for 100 iterations. In figure 4.7, we show the result of the perplexity of using different ratio of sampling rate from 10% to 100%. As depicted in Figure 4.7, all curve converge to the same point which empirically demonstrate our subset sampling method can keep the almost the same accuracy.

4.7 Conclusions

In this work, we developed *sdLDA*, a comprehensive solution to reduce the memory footprint of the popular LDA sampling algorithms and to minimize their computational complexity. The first key idea is to introduce a disk-based algorithm that maximally reduces its memory consumption

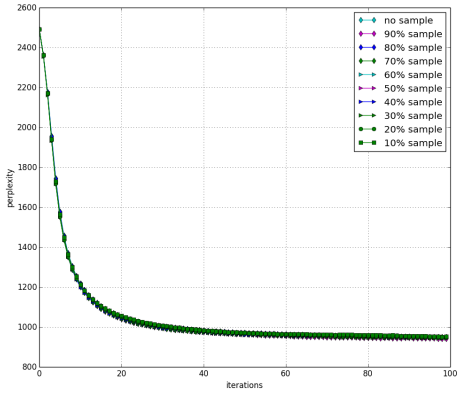


Figure 4.7: the perplexity of *sdLDA* with different sampling rate during 100 iterations on NIPS dataset. The perplexity of models using different sampling rates all converge to similar perplexity value.

with little to no sacrifice in the data access time. The second idea is to exploit the sparsity of data for in-memory data representation and sampling so that we can further reduce LDA's memory footprint and sampling complexity. Through a careful implementation of these ideas, *sdLDA* achieves up to two orders of magnitude reduction in terms of memory consumption and up to a factor two reduction in terms of computational time to the vanilla Gibbs sampling algorithm.

CHAPTER 5

Conclusion

This dissertation presents three representative works on natural language processing for more scalable efficient algorithms. They tackled word-level word representation learning optimization, sentence-level lexical simplification, and paragraph-level scalable topic models. We conclude this dissertation and suggest some promising future directions in this section.

In Chapter 2, we introduced a new neural-based lexical simplification model: neural simplicity ranking(NSR). This work discussed two important factors of lexical simplification: Simplicity and Relatedness. The feature set is carefully tested for these two metrics. We eliminated the word length feature that has been commonly used by previous works while we find it can not provide a clean signal for simplicity. We also introduced the contextualized word embedding based on the BERT model to evaluate the candidate words' relatedness and target words. To capture the feature from different granularity, we adopted a highway network to replace the previous fully-connected neural network. The binary output adoption for the pairwise ranking scheme improves the model accuracy significantly. We also conduct ablation studies to confirm the effectiveness of new feature sets.

In Chapter 3, We present amplified negative sampling(ANS) and A rigorous mathematical analysis for negative sampling. Our derivation, respectively, derives the convergence point of L1, L2, and log loss for negative sampling. L2 loss and log loss share the same convergence point while L1 loss converges to a different convergence point. Moreover, we need at least one negative example to guarantee the negative sampling algorithm to take effect. Our theory shows why the model will degenerate if we don't use any negative examples.

In Chapter 4, In this work, we developed *sdLDA*, a comprehensive solution to reduce the memory footprint of the popular LDA sampling algorithms and to minimize their computational com-

plexity. The first key idea is introducing a disk-based algorithm that maximally reduces memory consumption with little to no sacrifice in the data access time. The second idea is to exploit the sparsity of data for in-memory data representation and sampling to further reduce LDA's memory footprint and sampling complexity. Through careful implementation of these ideas, *sdLDA* achieves up to two orders of magnitude reduction in terms of memory consumption and up to a factor two reduction in terms of computational time to the vanilla Gibbs sampling algorithm.

Altogether, we are excited about all the natural language processing frameworks and algorithms' progress and glad to contribute to this. We think there is still a long way to go and would like to point out future research avenues in our minds.

More powerful knowledge graph construction and integration In this dissertation, we introduced lexical simplification by focusing on simplicity and relatedness. Meanwhile, a large number of knowledge graphs have been built, such as WordNet (Mil98), BabelNet (NP12). Incorporating the knowledge graph information can effectively improve word sense disambiguation capability that is crucial for natural language processing tasks.

One potential direction is constructing a semantic knowledge graph. The knowledge graph such as WordNet, BabelNet model each word or concept as a node, and each word's meaning is attached. However, the distribution of each meaning on each word is still unknown. Moreover, no effective algorithm can quickly calibrate a specific meaning for a word given its context with the knowledge graph. There are existing works (ALL18) which can distinguish different meaning for a polysemy. However, how to effectively calibrate the polysemy's sense with the knowledge graph's entry is another important question.

Theoretical analysis for representation learning The distributed representation has brought plenty of breakthrough in natural language processing. The theoretical foundation and mathematical property of the representation learning are still an active open area. Several works discussed the representation learning's mathematical property, for example (GL14) discussed the matrix decomposition equivalence with the word2vec model. The relationship between representation learning and downstream tasks' performance is another potential direction for the exploration. Early attempts have made, such as (AKK19).

I hope this dissertation could inspire research in natural language processing. Furthermore, I hope the deep learning method, embedding approaches, theoretical analysis and powerful framework could contribute to academic research and industrial innovations.

Bibliography

[AAG12] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J Smola. "Scalable inference in latent variable models." In *Proceedings of the fifth ACM international conference on Web search and data mining*, pp. 123–132. ACM, 2012. 49

[ABV18] Alan Akbik, Duncan Blythe, and Roland Vollgraf. "Contextual String Embeddings for Sequence Labeling." In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. 1

[AKK19] Sanjeev Arora, Hrishikesh Khandeparkar, Mikhail Khodak, Orestis Plevrakis, and Nikunj Saunshi. "A Theoretical Analysis of Contrastive Unsupervised Representation Learning." *arXiv preprint arXiv:1902.09229*, 2019. 2, 45, 73

[ALL18] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. "Linear Algebraic Structure of Word Senses, with Applications to Polysemy.", 2018. 73

[BBE11] Or Biran, Samuel Brody, and Noémie Elhadad. "Putting it Simply: a Context-Aware Approach to Lexical Simplification." In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 496–501, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 7, 17, 21

[BGJ17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching Word Vectors with Subword Information." *Transactions of the Association for Computational Linguistics*, **5**:135–146, 2017. 39, 42

[BK16] Oren Barkan and Noam Koenigstein. "Item2vec: neural item embedding for collaborative filtering." In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2016. 24, 27, 28

[BL13] K. Bache and M. Lichman. "UCI Machine Learning Repository.", 2013. 56

[BNJ03a] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation." *Journal of machine Learning research*, **3**(Jan):993–1022, 2003. 1

[BNJ03b] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *Journal of Machine Learning Research*, **3**:993–1022, March 2003. 3, 49, 52

[BR17] Guy Blanc and Steffen Rendle. "Adaptive sampled softmax with kernel based sampling." *arXiv preprint arXiv:1712.00527*, 2017. 43, 44

[BS03] Yoshua Bengio, Jean-Sébastien Senécal, et al. "Quick training of probabilistic neural nets by importance sampling." In *AISTATS*, pp. 1–9, 2003. 24, 44

[BS08] Yoshua Bengio and Jean-Sébastien Senécal. "Adaptive importance sampling to accelerate training of a neural probabilistic language model." *IEEE Transactions on Neural Networks*, **19**(4):713–722, 2008. 24, 44

[BTS14] Bin Bi, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Junghoo Cho. "Scalable topic-specific influence analysis on microblogs." In *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 513–522. ACM, 2014. 47

[CBH13] Youngchul Cha, Bin Bi, Chu-Cheng Hsieh, and Junghoo Cho. "Incorporating popularity in topic models for social network analysis." In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pp. 223–232. ACM, 2013. 47

[CLZ16] Jianfei Chen, Kaiwei Li, Jun Zhu, and Wenguang Chen. "Warplda: a cache efficient o (1) algorithm for latent dirichlet allocation." *Proceedings of the VLDB Endowment*, **9**(10):744–755, 2016. 50

[DCL18a] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805*, 2018. 1

[DCL18b] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." 2018. 13

[DT98] S Devlin and J Tait. "Linguist Databases, chapter The use of a Psycholinguistic database in the Simplification of Text for Aphasic Readers.", 1998. 6

[fas19] fastText. "Facebook fastText project source code." https://github.com/facebookresearch/fastText.git, 2019. 42

[FBD13] James Foulds, Levi Boyles, Christopher DuBois, Padhraic Smyth, and Max Welling. "Stochastic collapsed variational Bayesian inference for latent Dirichlet allocation." In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 446–454. ACM, 2013. 49

[GH10] Michael Gutmann and Aapo Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010. 2, 24, 28, 45

[GL14] Yoav Goldberg and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv:1402.3722*, 2014. 30, 35, 73

[GL16a] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. 2, 24, 28, 39, 43

[GL16b] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks." In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016. 27, 28

[GS04] Thomas L. Griffiths and Mark Steyvers. "Finding scientific topics." *Proceedings of the National Academy of Sciences of the United States of America*, **101**:5228–5235, 2004. 49

[GS15] Goran Glavas and Sanja Stajner. "Simplifying lexical simplification: Do we need simplified corpora?" In *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, volume 2, pp. 63–68, 2015. 7, 10, 17, 21

[HBB10] Matthew Hoffman, Francis R Bach, and David M Blei. "Online learning for latent dirichlet allocation." In *advances in neural information processing systems*, pp. 856–864, 2010. 47, 49

[HMK14] Colby Horn, Cathryn Manduca, and David Kauchak. "Learning a lexical simplifier using Wikipedia." In *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, volume 2, pp. 458–463. Association for Computational Linguistics, 2014. 7, 17, 21

[KK18] Nikita Kitaev and Dan Klein. "Constituency Parsing with a Self-Attentive Encoder.", 2018. 1

[LAR14] Aaron Q Li, Amr Ahmed, Sujith Ravi, and Alexander J Smola. "Reducing the sampling complexity of topic models." In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 891–900. ACM, 2014. 47, 49, 50

[LCC17] Kaiwei Li, Jianfei Chen, Wenguang Chen, and Jun Zhu. "Saberlda: Sparsity-aware learning of topic models on gpus." *ACM SIGOPS Operating Systems Review*, **51**(2):497–509, 2017. 50

[LLK18] Hae Beom Lee, Juho Lee, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. "DropMax: Adaptive variational softmax." In *Advances in Neural Information Processing Systems*, pp. 919–929, 2018. 24

[LSJ09] Simon Lacoste-Julien, Fei Sha, and Michael I Jordan. "DiscLDA: Discriminative learning for dimensionality reduction and classification." In *Advances in neural information processing systems*, pp. 897–904, 2009. 49

[LSM13] Thang Luong, Richard Socher, and Christopher Manning. "Better word representations with recursive neural networks for morphology." In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pp. 104–113, 2013. 42

[MA16] Andre Martins and Ramon Astudillo. "From softmax to sparsemax: A sparse model of attention and multi-label classification." In *International Conference on Machine Learning*, pp. 1614–1623, 2016. 24, 45

[MB05] Frederic Morin and Yoshua Bengio. "Hierarchical probabilistic neural network language model." In *Aistats*, volume 5, pp. 246–252. Citeseer, 2005. 44

[MC18] Zhuang Ma and Michael Collins. "Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency." *arXiv preprint arXiv:1809.01812*, 2018. 2, 45

[MCC13a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*, 2013. 1, 2, 47

[MCC13b] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*, 2013. 23, 24, 27, 28, 34, 39, 40

[MH09] Andriy Mnih and Geoffrey E Hinton. "A scalable hierarchical distributed language model." In *Advances in neural information processing systems*, pp. 1081–1088, 2009. 44

[MHB12] David Mimno, Matt Hoffman, and David Blei. "Sparse stochastic inference for latent Dirichlet allocation." *arXiv preprint arXiv:1206.6425*, 2012. 49

[Mil98] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998. 73

[MK13] Andriy Mnih and Koray Kavukcuoglu. "Learning word embeddings efficiently with noise-contrastive estimation." In *Advances in neural information processing systems*, pp. 2265–2273, 2013. 24

[MSC13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems*, pp. 3111–3119, 2013. 24, 27, 36, 42

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. "Building a Large Annotated Corpus of English: The Penn Treebank." *Computational Linguistics*, **19**(2):313–330, 1993. 43

[MT12] Andriy Mnih and Yee Whye Teh. "A fast and simple algorithm for training neural probabilistic language models." *arXiv preprint arXiv:1206.6426*, 2012. 2, 24, 28

[MX18] Mounica Maddela and Wei Xu. "A Word-Complexity Lexicon and A Neural Readability Ranking Model for Lexical Simplification." In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3749–3760, 2018. xi, 2, 7, 8, 10, 11, 15, 17, 18, 19, 22

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations." pp. 746–751, 2013. 47

[nod19] Github node2vec. "Node2vec project souce code." https://github.com/snap-stanford/snap.git, 2019. 43

[NP12] Roberto Navigli and Simone Paolo Ponzetto. "BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network." *Artificial Intelligence*, **193**:217–250, 2012. 73

[NSW07] David Newman, Padhraic Smyth, Max Welling, and Arthur U Asuncion. "Distributed inference for latent dirichlet allocation." In *Advances in neural information processing systems*, pp. 1081–1088, 2007. 47, 48, 49

[PN07] Xuan-Hieu Phan and Cam-Tu Nguyen. "GibbsLDA++: A C/C++ implementation of latent Dirichlet allocation (LDA).", 2007. 65

[PN15] Ellie Pavlick and Ani Nenkova. "Inducing lexical style properties for paraphrase and genre differentiation." In *NAACL HLT 2015 - 2015 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, pp. 218–224, 2015. 10, 11, 20

[PNI08] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. "Fast collapsed gibbs sampling for latent dirichlet allocation." In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 569–577. ACM, 2008. 50

[PNI18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." In *Proc. of NAACL*, 2018. 13

[PS16] Gustavo H Paetzold and Lucia Specia. "Unsupervised lexical simplification for non-native speakers." In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 3761–3767, 2016. 7, 10, 17, 22

[PS17a] Gustavo H. Paetzold and Lucia Specia. "A survey on lexical simplification.", nov 2017. 7, 9, 10, 11

[PS17b] Gustavo Henrique Paetzold and Lucia Specia. "Lexical simplification with neural ranking." In *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference*, volume 2, pp. 34–40, 2017. 2, 8, 15, 17, 18, 19, 22

[PZC08] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. "One-class collaborative filtering." In *2008 Eighth IEEE International Conference on Data Mining*, pp. 502–511. IEEE, 2008. 24, 45

[QLZ19] Jipeng Qiang, Yun Li, Yi Zhu, Yunhao Yuan, and Xindong Wu. "A Simple BERT-Based Approach for Lexical Simplification." 2019. 7, 9, 22

[RCY19] Ankit Singh Rawat, Jiecao Chen, Felix Xinnan X Yu, Ananda Theertha Suresh, and Sanjiv Kumar. "Sampled softmax with random fourier features." In *Advances in Neural Information Processing Systems*, pp. 13834–13844, 2019. 44

[RJL18] Pranav Rajpurkar, Robin Jia, and Percy Liang. “Know What You Don’t Know: Unanswerable Questions for SQuAD.”, 2018. 1

[SC11] Lifeng Shang and Kwok-Ping Chan. “A temporal latent topic model for facial expression recognition.” In *Proceedings of the 10th Asian conference on Computer vision - Volume Part IV*, pp. 51–63, 2011. 49

[SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Highway networks.” *arXiv preprint arXiv:1505.00387*, 2015. 16

[SJM12a] Lucia Specia, Sujay Kumar Jauhar, and Rada Mihalcea. “SemEval-2012 Task 1: English Lexical Simplification.” In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 347–355, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics. 7

[SJM12b] Lucia Specia, Sujay Kumar Jauhar, and Rada Mihalcea. “SemEval-2012 task 1: English lexical simplification.” In **SEM 2012 - 1st Joint Conference on Lexical and Computational Semantics*, volume 2, pp. 347–355, 2012. 17, 21

[SN10] Alexander Smola and Shravan Narayanamurthy. “An architecture for parallel topic models.” *Proceedings of the VLDB Endowment*, **3**(1-2):703–710, 2010. 47, 48, 49

[SN12] Issei Sato and Hiroshi Nakagawa. “Rethinking collapsed variational Bayes inference for LDA.” *arXiv preprint arXiv:1206.6435*, 2012. 49

[Spe04] Charles Spearman. “The proof and measurement of association between two things.” *American journal of Psychology*, **15**(1):72–101, 1904. 42

[STP17] Yikang Shen, Shawn Tan, Chrisopher Pal, and Aaron Courville. “Self-organized hierarchical softmax.” *arXiv preprint arXiv:1707.08588*, 2017. 44

[SWA09] Padhraic Smyth, Max Welling, and Arthur U Asuncion. "Asynchronous distributed learning of topic models." In *Advances in Neural Information Processing Systems*, pp. 81–88, 2009. 47, 48, 49

[TKW08] Yee W Teh, Kenichi Kurihara, and Max Welling. "Collapsed variational inference for HDP." In *Advances in neural information processing systems*, pp. 1481–1488, 2008. 49

[TNW07] Yee W Teh, David Newman, and Max Welling. "A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation." In *Advances in neural information processing systems*, pp. 1353–1360, 2007. 49

[VDB15] Pascal Vincent, Alexandre De Brébisson, and Xavier Bouthillier. "Efficient exact gradient update for training deep networks with very large sparse targets." In *Advances in Neural Information Processing Systems*, pp. 1108–1116, 2015. 45

[WBS09] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y Chang. "PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications." *AAIM*, **9**:301–314, 2009. 49

[WMW17] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. "Knowledge graph embedding: A survey of approaches and applications." *IEEE Transactions on Knowledge and Data Engineering*, **29**(12):2724–2743, 2017. 24

[wor19] Github word2vec. "Word2vec project source code." https://github.com/tmikolov/word2vec.git, 2019. 42

[WSC16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol

Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." *CoRR*, **abs/1609.08144**, 2016. 1

[WTK12] Max Welling, Yee Whye Teh, and Hilbert Kappen. "Hybrid variational/Gibbs collapsed inference in topic models." *arXiv preprint arXiv:1206.3297*, 2012. 49

[WZS15] Yi Wang, Xuemin Zhao, Zhenlong Sun, Hao Yan, Lifeng Wang, Zhihui Jin, Liubin Wang, Yang Gao, Ching Law, and Jia Zeng. "Peacock: Learning long-tail topic features for industrial applications." *ACM Transactions on Intelligent Systems and Technology (TIST)*, **6**(4):47, 2015. 49

[XHD15] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. "Petuum: A new platform for distributed machine learning on big data." *IEEE Transactions on Big Data*, **1**(2):49–67, 2015. 49

[YBL17] Hsiang-Fu Yu, Mikhail Bilenko, and Chih-Jen Lin. "Selection of negative samples for one-class matrix factorization." In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 363–371. SIAM, 2017. 24, 45

[YGH15] Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. "Lightlda: Big topic models on modest computer clusters." In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1351–1361. International World Wide Web Conferences Steering Committee, 2015. 47, 48, 50

[YHY15] Hsiang-Fu Yu, Cho-Jui Hsieh, Hyokun Yun, SVN Vishwanathan, and Inderjit S Dhillon. "A scalable asynchronous distributed algorithm for topic modeling." In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1340–1350. International World Wide Web Conferences Steering Committee, 2015. 47, 48, 49, 50

[YMM09] Limin Yao, David Mimno, and Andrew McCallum. "Efficient methods for topic model inference on streaming document collections." In *Proceedings of the 15th ACM*

SIGKDD international conference on Knowledge discovery and data mining, pp. 937–946. ACM, 2009. 3, 47, 48, 49, 50, 58, 62, 63, 64, 65

[YPD10] Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. "For the sake of simplicity: Unsupervised extraction of lexical simplifications from Wikipedia.", 2010. 2, 6

[YSC13] Hongzhi Yin, Yizhou Sun, Bin Cui, Zhiting Hu, and Ling Chen. "Lcars: a location-content-aware recommender system." In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 221–229. ACM, 2013. 47

[YXQ09] Feng Yan, Ningyi Xu, and Yuan Qi. "Parallel inference for latent dirichlet allocation on graphics processing units." In *Advances in Neural Information Processing Systems*, pp. 2134–2142, 2009. 49

[YZS17] Lele Yu, Ce Zhang, Yingxia Shao, and Bin Cui. "LDA*: A Robust and Large-scale Topic Modeling System." *Proceedings of the VLDB Endowment*, **10**(11), 2017. 47, 49, 50

[ZBA12] Ke Zhai, Jordan Boyd-Graber, Nima Asadi, and Mohamad L Alkhouja. "Mr. LDA: A flexible large scale topic modeling package using variational inference in mapreduce." In *Proceedings of the 21st international conference on World Wide Web*, pp. 879–888. ACM, 2012. 49

[ZCL13] Jia Zeng, William K Cheung, and Jiming Liu. "Learning topic models by belief propagation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(5):1121–1134, 2013. 49

[ZL09] R. Zafarani and H. Liu. "Social Computing Data Repository at ASU.", 2009. 43

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent Neural Network Regularization.", 2014. 43