2016

EVALUATING DISTRIBUTED WORD REPRESENTATIONS FOR PREDICTING MISSING WORDS IN SENTENCES

Saniya Saifee
CUNY City College

EVALUATING DISTRIBUTED WORD REPRESENTATIONS FOR PREDICTING MISSING WORDS IN SENTENCES

A Thesis

Submitted to

The City College of New York

Department of Computer Science

City University of New York

New York, New York

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Saniya Saifee

May 2016

ABSTRACT

In recent years, the distributed representation of words in vector space or word embeddings have become very popular as they have shown significant improvements in many statistical natural language processing (NLP) tasks as compared to traditional language models like N-gram. In this thesis, we explored various state-of-the-art methods like Latent Semantic Analysis, word2vec, and GloVe to learn the distributed representation of words. Their performance was compared based on the accuracy achieved when tasked with selecting the right missing word in the sentence, given five possible options. For this NLP task we trained each of these methods using a training corpus that contained texts of around five hundred 19th century novels from Project Gutenberg. The test set contained 1040 sentences where one word was missing from each sentence. The training and test set were part of the Microsoft Research Sentence Completion Challenge data set. In this work, word vectors obtained by training skip-gram model of word2vec showed the highest accuracy in finding the missing word in the sentences among all the methods tested. We also found that tuning hyperparameters of the models helped in capturing greater syntactic and semantic regularities among words.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am extremely grateful to Dr. **Jie Wei** for providing me an exciting opportunity to work with him. His advice and support were critical for completion of this work. I had a very fruitful learning experience throughout the course of this research, which would not have been possible without Dr. Wei's guidance.

I would like to acknowledge the support from The City College of New York for providing good learning opportunities and fostering meaningful collaboration with industry for advancing relevant technical skills.

Finally, I would also like to thank my family for their love and support during the course of my study. Last but not the least I would like to thank my husband Anees for proofreading my thesis and for his encouragement.

CHAPTER 1: INTRODUCTION

1.1 Overview

Our brains provide us with an array of reciprocally connected information processing sites that exploit records of recent events in order to rapidly and accurately anticipate future events, such as the next word in this text, in a friend's conversation or navigating a busy sidewalk while reading one's cell phone. Our quality of life and, indeed, our survival depend on our ability to quickly anticipate objects and events in our immediate environment. Most of those objects, such as words, the primary object of our research, occur in a rich time and space context. Exploiting this contextual information prior to recognizing the next word helps the perceptual apparatus anticipate each next word so that only a fraction of its features may be needed for fast accurate recognition. The payoff for this savings is increased information processing per unit time and increased the availability of cortical processing resources for higher order analyses such as word and sentence comprehension.

In 1950, Alan Turing published his paper “Computing Machinery and Intelligence” [1] in which the opening line was “I propose to consider the question, 'Can machines think?”. The famous Turing test evolved from this paper which asserted that a computer is considered as intelligent as human if it can communicate with human beings without the human realizing that they are talking to machines. Natural language processing (NLP) has been considered one of the hot topics for artificial intelligence ever since Turing proposed this game.

Natural Language Processing (NLP) is related to human and machine interaction. The aim of the research is to understand how humans acquire knowledge, build their vocabulary and create tools and methods that can help computer in understanding and manipulating

natural languages [2]. NLP is a multi-disciplinary field that includes computer science, mathematics, computational linguistics and artificial intelligence. There are many applications of NLP – machine translation, automatic speech recognition, information retrieval, text summarization etc. As suggested in [3] a major challenge in NLP is to ascribe the correct meaning to a word while accounting its contextual information. For example, consider the following two sentences

a) I am going to the *bank* to deposit money.

b) She is standing near the *bank* of the river.

In the first sentence, the word bank refers to a financial institution while in the second sentence it refers to the land alongside a river. Humans can figure out this difference in meaning based on the context words surrounding the central word.

Also, the meaning of a word is determined by the situation in which it is used. Consider the following two sentences

a) She chopped the vegetables with *knife*.

b) The man was stabbed with a *knife*.

In the first sentence *knife* is used as a tool but in the second it is a weapon.

With the advent and growth of Internet, social media, and blogs a large amount of unstructured text data exist. It is difficult to analyze a large corpus of text to discover the structure within the data using computational methods. In the traditional NLP approach, words are considered as single atomic units and are represented as indices in the vocabulary. This kind of representation of word fails to capture the similarity among words. But this representation of the word is very popular because of its simplicity and robustness. When

trained on large dataset n-gram model shows better performance than the complex models trained on smaller datasets [4].

Many tasks in NLP such as Part-Of-Speech (POS) tagging, chunking, document classification, information retrieval use supervised learning methods. These supervised methods require annotated corpora and manually selected features as input. Hand annotation of the input data is a time consuming and expensive task. Many available annotated data like Penn Treebank are very small - around 50K sentences, compared to two trillion tokens used in [5]. The annotated data are hard to adopt for new tasks and languages. The robustness of the models trained using supervised methods raises concerns as they are trained using a small amount of data and may not perform efficiently on new data [6]. The limited amount of annotated training data for Machine Translation and Automatic Speech Recognition systems has restricted the scope of improvement in their performance. NLP classification tasks trained using a different text data performs poorly on text chosen from a completely different genre.

Due to the limitations of supervised learning algorithms the research in NLP has started focusing on implementation of efficient unsupervised or semi-supervised algorithms. The main advantage of unsupervised algorithms is that the input is unannotated raw text so a lot of time and effort is saved as the costly hand-annotation is not required. These unsupervised algorithms performed much better that the supervised ones [7].

One unsupervised method that is gaining a lot of traction these days is the distributed representation of words in vector space. These word representations are vectors in multi-dimensional space where each dimension value is a word feature and represent a syntactic or semantic property of the word. These feature values are learned by the vectors themselves

[7]. These word representations should encode similarities - semantic and grammatical, among words. Similar words should be closer to each other in the vector space.

The next step after inducing continuous representation of words is to combine them into a meaningful and grammatically correct sequence. This is called *language modeling*. The language model is an algorithm that defines the probability distribution over all possible sequence of words or utterances [8]. The main purpose is to capture regularities in the language to improve the performance of various NLP tasks.

Language models are a way for evaluating human-machine interaction. Given a sentence like *'I love driving cars'* and a word '*pizza*', a good language model should be able to tell that the word '*pizza*' is not relevant to the sentence. One way to evaluate a language model is its performance in the task of predicting a missing word in the sentence. For example, given a sentence

$$I\ want\ to\ eat\ \_\_\_\_\_.$$

An efficient language model should predict that the missing word should be some edible items.

Traditionally the most popular way of implementing statistical language model is the *n-gram model*. The statistical language models have been used successfully in various NLP applications like machine translation, spelling correction, and automatic speech recognition [8]. The n-gram model uses the frequency of occurrence of sequences of words of length one, two, three or more in the training data.

Consider that a word sequence ending in $\cdots w_{t-2}w_{t-1}w_t w_{t+1}$ occurs frequently in the training corpus. The n-gram language model will estimate the probability of the word w_{t+1}

given a sequence of words $w_1 \cdots w_{t-2} w_{t-1} w_t w_{t+1}$ by ignoring the context beyond $n-1$ words. If we are using trigram model, then

$$P(w_{t+1}|w_1 \cdots w_{t-2} w_{t-1} w_t w_{t+1}) = P(w_{t+1}|w_{t-2} w_{t-1} w_t) \quad (1)$$

$$P(w_{t+1}|w_{t-2} w_{t-1} w_t) = \frac{count(w_{t-2} w_{t-1} w_t w_{t+1})}{count(w_{t-2} w_{t-1} w_t)} \quad (2)$$

One issue with n-gram model is that if a sequence of words is not observed in training set then that sequence will be assigned zero probability. This is known as the *curse of dimensionality*. The n-gram model requires a large amount of training data to account for the large size of vocabulary in the language and for efficient estimation of parameters used by the model. Various smoothing techniques are used to handle sequences of words that are missing in the training set. More details about these smoothing techniques will be discussed in chapter 3. The statistical language model considers words as a single unit and though they exploit the previous n context word around the target words they do not take into account the deep structure and meaning of the words.

To fight the curse of dimensionality and to generalize language model to unseen data a new approach for the distributed representation of word [9] in continuous vector space has become very popular. This word representation is also known as word embedding. Suppose we have a vocabulary, denoted by V, of the unique words in the corpus. The i^{th} word in V is represented as an embedding by $w_i \in \mathbb{R}^d$ where d is smaller than the total size of V. Each dimension in this continuous d-dimensional vector is a word feature that captures the syntactic and semantic regularities of the word in the training corpus. The words that are similar to each other should share some of these latent features and should be closer to each other in the vector space. The main advantage of the distributed word representation is that

they allow language model to generalize well on unseen data. The word embedding represents a large set of possible meaning in a compact form. There are many ways of creating these word embeddings. One of the earlier famous methods of learning the distributed representation of word is Latent Semantic Indexing [9, 10]. The Latent Semantic Analysis/ Latent semantic Indexing perform singular value decomposition on a term-document matrix created from the training corpus.

One of the word embedding technique that has garnered a lot of news coverage is the word2vec [4, 12] neural network based language model. Word2vec uses two kinds of architecture - *Continuous Bag-of-Words* and *Skip-Gram* for computing distributed representation of words.

The other famous unsupervised algorithm for finding similarities among the words using low d-dimensional distributed representation is *GloVe: Global Vectors for Word Representation* [13].

These representations can be subsequently used in many natural language processing applications and for further research. Distributed representations of words have shown to benefit a diverse set of NLP tasks including syntactic parsing [14, 15], named entity recognition [16] and sentiment analysis [17]. Additionally, because they can be created using raw words as input, they are likewise available in domains and languages where traditional linguistic resources do not exist.

1.2 Thesis Goal

In this thesis, we study various methods of creating the distributed representation of words in the vector space. We also assess the performance of these word embeddings on the task of

filling in the missing word in a sentence, given five-word choices. We used Microsoft Research Challenge Data training and test set to evaluate our methods [18].

1.3 Motivation

The task of predicting the missing word in a sentence by assessing how the different words fit into the sentence and the ability of the language model to answer such questions has a wide impact on the variety of NLP tasks like automatic speech recognition, optical character recognition, word prediction software used in word processor and augmentative communication systems, machine translation and text generation to calculate likelihood of a sequence of words or sentences. Most of the modern spell checking software can detect if the word is misspelled but these programs will fail to identify the issue with the following sentence as it is a grammatically correct sequence of words "*Please answer her fall"*. An algorithm that could catch this error would thus need to look beyond what letters form words and instead attempt to determine what word is most probable in a given sentence.

The data used in speech recognition and optical character recognition can be noisy due to unclear speech or handwriting. But if the model can exploit the context around the missing words and predict the missing word efficiently then the performance of these tasks can improve dramatically.

CHAPTER 2: RELATED WORK

The most recent work in the SAT style sentence completion task was published in a 2012 paper by Zweig et al [19]. Two approaches for solving this task are proposed – classical n-gram model that used the local context information and latent semantic analysis that uses distributed representation of words. Zweig et. al. evaluated their methods on practice SAT sentence completion questions and on the Microsoft Research Sentence Completion Challenge data that consists of 1040 test sentences. They achieved around 53% accuracy by using the linear combination of the outputs of n-gram and LSA language model. Lee et al [20] proposed an approach for solving SAT style sentence completion task by using web-scale data. They used n-gram probability distribution for identifying the candidate word and handled data sparseness issues by using *Backoff smoothing* techniques. They achieved 52-80% accuracy.

Another related work in sentence completion task was proposed by [21] where the probability of a sentence was estimated as the probability of the lexicalization of a given syntactic dependency tree. They tested their approach on Microsoft Research Sentence Completion Challenge data set and showed 8.7% performance improvement of n-gram model. In [22] the substitution word for the target word was estimated through the use of Word Sense Disambiguation and based on it an appropriate synonym was targeted. This Word Sense Disambiguation language model was trained using a large unannotated corpus. The system also participated in the SemEval-2007 Task 10: English Lexical Substitution Task and was ranked first.

Giuliano et al [23] also participated in the SemEval-2007 Task 10: English Lexical Substitution Task and created two systems. They used a synonym list created using WordNet and Oxford dictionary. One system used Latent Semantic Analysis (LSA) to rank the synonym list closest to the target word while the other system was trained using the Web IT 5-gram corpus. They achieved state-of-art performance with the 5-gram statistical language model.

A lot of work has also been done on evaluating the performance of language model on TOEFL synonyms, GRE antonym questions, and SAT analogy questions. Mohammad et al [24] presented a new empirical approach to find whether the words in a pair are antonyms of each other. They randomly selected a large set of antonym set from WordNet and observed their statistics in the British National Corpus using the context size of 5. They evaluated their approach on the GRE antonyms questions and reported an accuracy of 80%. Turney et al. [25] presented an algorithm to solve SAT-style analogy questions. Their model based on Vector Space Model of information retrieval was trained using unlabeled data. The model correctly answered 47% of analogy questions in the test set.

The other type of work that can be deemed similar to sentence completion task is the prediction of the word given the neighboring words. The main aim of word prediction software is to reduce the number of keystrokes required to complete the text being entered by the user by suggesting the list of most probable words that the user anticipates to type before actually typing it.

The word completion programs developed in early 80's were based on simple language models which suggested the list of most frequent words based on the current partially typed word [26]. These systems did not consider the context of a word which affect

their performance. WordQ [27] is the software that uses n-gram language models to predict the next word. While making predictions the software that exploits the context also consider the frequent words recently used by the users in order to adapt to the user's typing behavior [28]. Researchers in [29] used part-of–speech (POS) tagging for words to improve the algorithms for predicting next words. Systems that employ POS tagging first estimate the probability of the POS tag for the next word. They used this probability as well as bigram/trigram model for word suggestions. FASTY [30], a multilingual word prediction software, also uses POS tags of the words to exploit the syntactic information. FASTY uses collocation-based statistics of a corpus to include distant relationship among words. Collocations are expressions of multiple words which commonly co-occur like *red wine*. The use of POS tags further adds the complexity of annotating the words in the corpus with their corresponding POS tags. Some of the researchers in the word prediction that use part of speech tags along with bigram/trigram model have observed that probability estimation of POS tag of next word does not contribute much to the performance improvement in word prediction as compared to context based prediction. As per [28] when a bigram/trigram language model is used, it also captures the corresponding tag probabilities.

The NLP task of predicting the missing word or the next word in a sentence is a more difficult problem than filling the blank in the SAT style sentence completion question where five-word choices are given.

CHAPTER 3: RESEARCH LITEARTURE

In this chapter, we will review the literature that will provide us with the necessary background to explore various models to be used for the sentence completion task. We will start with the classical n-gram statistical language model, followed by the discussion of different types of the vector representation of the words and their uses. Then we will explore three popular methods to create these word embedding.

3.1 N-gram Models

As we have seen in the previous chapter the n-gram model is the most popular statistical language model for many NLP tasks – both syntactic as well as semantic tasks. Initially, the word prediction task was also considered a statistical natural language processing task. A language model would predict the missing word based on probability distribution of a sequence of words in the training corpus. This probability distribution is calculated by training the model using a large corpus of text like Brown Corpus.

Let $w_1, w_2, \dots, w_{t-1}$ be the sequence of words in a sentence and w_t is the next word in this sequence. The probability estimation function is denoted by

$$\mathrm{P(W)} = \mathrm{P}(w_t \mid w_1, w_2, \dots, w_{t-1}) \tag{3}$$

where $W = w_1, w_2, \dots, w_{t-1}, w_t$. Using Bayes theorem,

$$P(W) = \prod_{t=1}^{N} P((w_t \mid w_1, w_2, \dots, w_{t-1}) \tag{4}$$

The parameter space of the model will become very large if we consider the entire history of the word w_t. The n-gram model uses Markov assumption to simplify the probability estimation of the word sequence. Markov assumption is used to describe a model where Markov property is assumed to hold.

Per Markov property the stochastic processes are memory-less. The conditional probability distribution of the future word depends only on the preceding word. Formally the n-gram language model is represented as

$$\mathrm{P}(w_t \mid w_1, w_2, \ldots, w_{t-1}) = \mathrm{P}(w_t \mid w_{t-n+1}, \ldots, w_{t-1}) \tag{5}$$

If n=1, it is a *unigram model* and

$$\mathrm{P}(w_t \mid w_1, w_2, \ldots, w_{t-1}) = \mathrm{P}(w_t) \tag{6}$$

If n = 2, it is a *bigram model* and

$$\mathrm{P}(w_t \mid w_1, w_2, \ldots, w_{t-1}) = \mathrm{P}(w_t \mid w_{t-1}) \tag{7}$$

If n=3, it is a *trigram model* then

$$\mathrm{P}(w_t \mid w_1, w_2, \ldots, w_{t-1}) = \mathrm{P}(w_t \mid w_{t-2}\, w_{t-1}) \tag{8}$$

n-gram probabilities are estimated using maximum likelihood estimation (MLE). The MLE for n-gram model is evaluated by taking the frequency of words in the corpus and then normalizing the frequency to get the probability distribution [31].

The bigram probability is calculated as

$$P(w_t \mid w_{t-1}) = \frac{count(w_{t-1}\, w_t)}{count\ (w_{t-1})} \tag{9}$$

The trigram probability is given by

$$P(w_t \mid w_{t-2} w_{t-1}) = \frac{count(w_{t-2} w_{t-1}\, w_t)}{count\ (w_{t-2} w_{t-1})} \tag{10}$$

For the general case n-gram probability is estimated as

$$P(w_t|w_{t-n+1}^{t-1}) = \frac{count(w_{t-n+1}^{t-1}\, w_t)}{count\ (w_{t-n+1}^{t-1})} \tag{11}$$

3.1.1 N-gram Language model evaluation

A good language model is one that assigns a high probability to grammatically correct and frequently seen sentences. For a specific language model two different sets are required:

a) The training set for training the model and calculating the required probabilities.

b) The test set for evaluating model efficiency.

Training and testing data should be different. Testing on the training data might lead to overfitting and the model will not perform well on unseen data.

There are two methods to evaluate a language model – extrinsic evaluation and intrinsic evaluation [31]. Suppose we have to compare two language model M_1 and M_2. We embed these models in our application like a word predictor and run the task end to end. The model that improves the application word prediction accuracy is the better language model. The major drawback of extrinsic evaluation is that it is very slow and can take days and week to finish the evaluation. Instead, we use intrinsic evaluation that focuses on the efficiency and accuracy of language model rather than the application.

One of the popular metrics for intrinsic evaluation is perplexity. Perplexity (PP) is calculated on a test set as the inverse of the probability of the test set and is normalized by the number of words in the sequence. The major intuition of perplexity can be dated back to the information theory proposed by Shannon in 1948 [32]. Consider we have a very long sentence X in a test set with *m* words. Thus $X = w_{1\,,}\, w_2, \dots, w_m$. The perplexity of X is

$$PP(X) = P(w_{1\,,}\, w_2, \dots, w_m)^{\frac{-1}{m}} \tag{12}$$

Using chain rule,

$$PP(X) = \sqrt[m]{\prod_{t=1}^{m} \frac{1}{\mathrm{P}(w_t \mid w_1, w_2, \ldots, w_{t-1})}} \tag{13}$$

For a bigram model perplexity is given by

$$PP(X) = \sqrt[m]{\prod_{t=1}^{m} \frac{1}{\mathrm{P}(w_t \mid w_{t-1})}} \tag{14}$$

Since perplexity is the inverse of the conditional probability of test set, a better language model should minimize the perplexity of the language model. To compare unigram, trigram and bigram model using the perplexity matrix, each of these language model was trained using the Wall Street Journal corpus. This corpus contained 38 million tokens and the size of the vocabulary was 19, 979. The test set contained 1.5 million words [31].

	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

Table 1. Comparison of perplexity in Language model [31]

As we increase the context size, the perplexity decreases which means the language model is increasingly accurate. If we are using perplexity as our evaluation metric we should make sure that the test set is unseen. None of the sentences in the test set should occur in training set. The training should not have any prior knowledge about the test set or its vocabulary. Any prior knowledge of the test set would lead to overfitting and would lead to bias in perplexity calculation. Secondly, we should use the same test set to compare two language models that are trained using the same corpus.

3.1.2 Generalization of N-gram language model

The probabilities estimated by training the n-gram model encode facts about the training corpus. Thus the model created is dependent on the training set. As we increase the value of N, the context size, better performance is obtained. On borrowing the techniques of [31] and [33] to generate random sentences from these language models, probabilities are assigned to unigram, bigram, trigram and 4-gram sequences using maximum likelihood estimation. In case of unigram language model, we select the word with highest probability followed by a random word according to its probability and so on until we encounter the sentence ending token </s>. For bigram model using a priori distribution we select the bigram token that starts with <s> followed by randomly chosen bigram based on its probability and so on. We observe that the sentences generated by unigram language model have no coherent relation among words. This should be the case because unigram language model does not take into account the previous context of a word. But the sentences generated by bigram model will have some coherence mostly word to word because it takes into account the previous word. The sentence grammatical structure and meaning would keep on improving as we increase the context size to three, four and five.

Since the language model is dependent on the training set, the model will not be very efficient if we test on the data that is completely different from the training set. For example, if the language model is trained using Shakespeare's work and the test set is based on Wall Street Journal corpus then the model will perform poorly on the test set. One way to deal with this issue is to select training corpus from the genre that is similar to the task we are trying to solve.

But this can not solve the issue of data sparsity. The training corpus will always be limited in size. Many English words, word sequences, sentences will be missing from the training set. Thus, the n-gram matrix will have many zero entries. If any of these sequences will occur in the test data, the model will estimate zero probability and the perplexity which is the inverse of the probability will be undefined. To illustrate this issue, consider the following example from [35]. Suppose our training set consist of three sentences as shown.

a) JOHN READ MOBY DICK.

b) MARY READ A DIFFERENT BOOK. SHE READ A BOOK BY CHER.

We train the bigram model using the eq. (1). The probability of a sentence is

$$P(S) = \prod_{t=1}^{n+1} \mathrm{P}(w_t \mid w_{t-1}) \tag{15}$$

Suppose our testing set contains the sentence S_t = JOHN READ A BOOK.

$$P(\mathrm{S_t}) = P(JOHN|.)\ P(READ|JOHN)\ \ P(A|READ)\ P(BOOK|A)\ \ P(.|BOOK)$$

$$= \frac{count(.JOHN)}{count(.)}\ \ \frac{count(JOHN\ READ)}{count(JOHN)}\ \frac{count(READ\ A)}{count(READ)}\ \frac{count(A\ BOOK)}{count(A)}\ \frac{count(BOOK\ .)}{count(BOOK)}$$

$$= \frac{1}{3}\qquad \frac{1}{1}\qquad \frac{2}{3}\qquad \frac{1}{2}\qquad \frac{1}{2}$$

$$\approx 0.06$$

For $S_{t=}$ CHER READ A BOOK.

$$P(S_t) = P(CHER|.)\ P(READ|CHER)\ \ P(A|READ)\ P(BOOK|A)\ \ P(.|BOOK)$$

$$= \frac{0}{3}\qquad \frac{0}{1}\qquad \frac{2}{3}\qquad \frac{1}{2}\qquad \frac{1}{2}$$

$$= 0$$

3.1.3 Smoothing

To enable the n-gram model to generalize well to unseen data, we have to prevent the language model from assigning zero probability to unseen events. This can be achieved by applying smoothing to probability estimation. Smoothing discounts, the positive counts, counts of frequently occurring events in the training set, and reallocates the probability to unseen events whose count is zero. Smoothing is also called discounting. There are various ways of doing smoothing as discussed below.

3.1.3.1 Laplace Smoothing

This is the simplest smoothing technique. In Laplace smoothing [36] all the counts are incremented by one before normalizing to get probability estimate. Thus the unseen events count change from zero to one and other positive counts are incremented by one too. If we apply Laplace smoothing to unigram model, the probability estimate that was initially estimated as

$$P(w_i) = \frac{count(w_i)}{N} \tag{16}$$

will change to

$$P_{lap}(w_i) = \frac{count(w_i) + 1}{N + |V|} \tag{17}$$

where N is the total number of words in the training set and |V| is the vocabulary size. The denominator changes from N to N + |V| because we incremented the count of all the words by one which has to be taken into account. The smoothed bigram model probability will be estimated as

$$P_{lap}(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i) + 1}{count(w_{i-1}) + |V|} \tag{18}$$

On applying Laplace smoothing to the previous example,

P(JOHN READ A BOOK.) = $\frac{1+1}{3+11}\ \frac{1+1}{1+11}\ \frac{2+1}{3+11}\ \frac{1+1}{2+11}\ \frac{1+1}{2+11} \approx 0.001$

$P(CHER\ READ\ A\ BOOK.)$ = $\frac{0+1}{3+11}\ \frac{0+1}{1+11}\ \frac{2+1}{3+11}\ \frac{1+1}{2+11}\ \frac{1+1}{2+11} \approx 0.003$

Laplace smoothing is easy to apply but it overestimates the probability of an unseen event and can discount the estimate of frequent event that can reduce the efficiency of a language model. To fix this problem we can use *add-k smoothing* where instead of adding one to each count we add a fraction $k < 1$. The bigram probability estimate in add-k smoothing is given by

$$P_{add-k}(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i) + k}{count(w_{i-1}) + k|V|} \tag{19}$$

We need an additional validation set to estimate the value of parameter k such that it optimizes the performance of language model.

3.1.3.2 Good-Turing Smoothing

The main intuition behind this smoothing is to re-estimate the amount of probability mass to be assigned to n-gram with zero or lower counts by observing the n-gram of higher count [37]. Let N_c be the number of n-grams that occur c times. The new smoothed count is

$$\text{count}^* = (\text{count} + 1)\frac{N_{c+1}}{N_c} \tag{20}$$

3.1.3.3 Backoff Smoothing

By using smoothing, we can estimate the probability of novel events in the set by using only the raw frequency of n-gram. Backoff smoothing is another way of estimating these unseen data probabilities. Consider for example the computation of the trigram probability of the sequence w_1 w_2 w_3 where the count of this sequence in the data is 0. We compute the $P(w_3|w_1w_2)$ by estimating the bigram probability of bigram w_2w_3. But if the sequence w_2w_3 also does not exist then we compute unigram probability $P(w_3)$. Thus, we try to generalize the n-gram language model on the unseen events. For the correct probability distribution, the Backoff model should use smoothed counts. The Backoff model used with smoothing/discounting is called *Katz-Backoff* [38]. Trigram version of Katz-Backoff Smoothing model is

$$P_{kb}(w_i|w_{i-1}w_{i-2}) = \begin{cases} P(w_i|w_{i-1}w_{i-2}) & if\ count(w_{i-1}w_{i-2}w_i) > 0 \\ \alpha_1 P(w_i|w_{i-1}) & if\ count(w_{i-1}w_{i-2}w_i) = 0 \\ & but\ count(w_{i-1}w_{i-2}) > 0 \\ \alpha_2 P(w_i), & otherwise \end{cases} \quad (21)$$

Mostly Good-Turing smoothing is used with Katz-Backoff model.

3.1.3.4 Linear Interpolation

The linear interpolation estimates the trigram probabilities by taking a weighted sum of all trigram, bigram and unigram estimate.

$$P_{li}(w_i|w_{i-1}w_{i-2}) = \lambda_1 P(w_i|w_{i-1}w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i) \quad (22)$$

such that $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0\ and\ \lambda_1 + \lambda_2 + \lambda_3 = 1.$

The common way of estimating the parameter λ is to create a development test data which is different from training and test data. Let $c(w_{j-2}w_{j-1}w_j)$ be the frequency of trigram $w_{j-2}w_{j-1}w_j$ in the development test set. The log likelihood of development test data is a function of $\lambda_1, \lambda_2, \lambda_3$.

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_{j-2}w_{j-1}w_j} c(w_{j-2}w_{j-1}w_j) \log P_{li}(w_j | w_{j-1}w_{j-2}) \tag{23}$$

The value of λ should be so chosen to $arg \max_{\lambda_1, \lambda_2, \lambda_3} L(\lambda_1, \lambda_2, \lambda_3)$

3.1.3.5 Kneser-Ney Smoothing

Kneser-Ney smoothing [39] is the most popular smoothing algorithm. The main idea behind this algorithm is that the lower order model is important only when higher order model is sparse.

Kneser-Ney uses absolute discounting [39] on higher order N-gram along with linear interpolation on lower order N-grams. The equation for interpolated absolute discounting applied to bigram is:

$$P_{abs\_dis}(w_i | w_{i-1}) = \frac{count(w_{i-1}w_i) - d}{count(w_{i-1})} + \lambda_{w_{i-1}} P(w) \tag{24}$$

The first term is the discounted bigram estimate and the second term is the interpolated unigram probability.

Consider that our bigram language model is trained on data where the word *Francisco* occurred frequently [35]. The maximum likelihood estimate of this unigram will be high. If we are trying to predict the next word in the sentence "*I cannot see without my reading* _____.", the absolute discounting interpolation will assign *Francisco* higher

probability than the *glasses*. But *Francisco* occurs only when the previous word is *San*. The word *glasses* has a much wider distribution and we need to capture this intuition. The *Kneser-Ney smoothing* provides an elegant solution to this problem by modifying the unigram model where the probability estimate of unigrams is based on the count of different context the word w has occurred rather than the raw probabilities based on the frequency of the word in the corpus.

$$P_{cont}(w_i) = \frac{|\{w_{i-1}: count(w_{i-1}w_i) > 0\}|}{|\{\{w_{j-1}w_j: count(w_{j-1}w_j) > 0\}|} \tag{25}$$

Now $P_{cont}(francisco) < P_{cont}(glasses)$

The Kneser-Ney algorithm is

$$P_{kneser\_ney}(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i) - d}{count(w_{i-1})} + \lambda_{w_{i-1}} P_{cont}(w_i) \tag{26}$$

We need to estimate the parameter d and $\lambda_{w_{i-1}}$ and the details are in [39].

3.1.3.6 Stupid Backoff smoothing

With the growth of the internet, a large amount of text data exists on the web which can be used to create large language model. Efficiency is an important consideration in the large language model. Stupid Backoff [40] is a simple algorithm to create efficient large language model from web text data. It does not discount higher order n-gram probability. If the frequency of higher order n-gram is 0 in the set it backs off to lower order n-gram weighted by a fixed context independent weight factor. This algorithm does not estimate probabilities so following [40]

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{freq(w_{i-k+1}^{i})}{freq(w_{i-k+1}^{i-1})} & if\ freq(w_{i-k+1}^{i}) > 0 \\ \alpha S(w_i|w_{i-k+2}^{i-1}) & otherwise \end{cases} \quad (27)$$

This Backoff step terminates in unigram where

$$S(w) = \frac{freq(w)}{N} \quad (28)$$

where $N = total\ number\ of\ words\ in\ the\ vocabulary$. Using heuristics, [40] set the value of $\alpha = 0.4$ in all experiments.

3.2 Learning Distributed Representation of Words

Even though we have a large amount of text available online that can help to resolve the data sparsity issue in language models, the recent research in IBM has shown that the improvement in statistical language model has reached saturation. Word embeddings or distributed representation of words capture the syntactic and semantic regularities of the word in the training data and represent them in low d-dimensional space. The use of these word embeddings has outperformed n-gram model in many NLP tasks. The models that use the distributed word representation achieve greater generalization as these word representations take into account similarity among words.

3.2.1 Different form of word representation in vector space

Word is the basic unit in most of the NLP applications. In computers, words are represented as strings - a sequence of characters that does not convey the meaning of the word. Syntactically we can create similar words by modifying them using morphological derivations (*write- writer*) and inflections (*rat – rats, decide - decided*). These words are

similar in meanings as well as in terms of syntax. Similarity among words with similar characters can be captured using Levenshtein Distance [50] or Edit Distance. But there are many words like *hotel* and *motel*, feline and *cat* which have similar meaning but Levenshtein Distance or Edit Distance will fail to find similarity using these raw words. We need a better representation of words that cluster similar word, both in meaning and syntax, together.

To encode the similarities among words we can represent words as vectors. The simplest form of word vector is the one-hot vector. The first step is creating the vocabulary V by processing the text in the large training corpus. We start by segmenting the raw text into sentence followed by tokenization and then if required normalizing the word tokens. We also remove any non-textual content like HTML tags, Java Script codes from the text. If our corpus contains text from British English as well as American English then there are certain words like *color* and *colour*, *behavior* and *behaviour* that are spelled differently but they have the same meaning and we might want to normalize them into one. We can also conduct stemming and lemmatization based on our needs.

When creating a vocabulary one should know the difference between *type* and *token*. Consider the following sentence '*democracy is for the people, by the people and of the people*'. There are 12 tokens and 8 types. A type is a sequence of characters that represents a word while a token is an instance of the type.

In our vocabulary we keep the inflections and derivations of the same basic form like car and cars i.e. *car* and *cars* exist in vocabulary. Let |V| denotes the size of the vocabulary. Every word in our text is represented as $\mathbb{R}^{|V|\times|V|}$vector with one at the index position of the word in the vocabulary and zero elsewhere. For example, our vocabulary consists of following types:

$$V = \{zebra, animal, which, \dots, queen\}$$

The one-hot vector representation is

$$w_{zebra} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad w_{animal} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad w_{queen} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

These one-hot vectors are sparse with only one dimension on and they also do not capture the similarity among words [2]. Thus $(w_{zebra})^T . w_{animal} = 0$. We need a dense word vector for NLP related task. One of such vector is distributional vector.

3.2.1.1 Distributional vectors

Distributional vectors are based on distributional hypothesis – "*you shall know a word by the company it keeps*" [41]. By analyzing the statistical pattern of the words written or uttered by humans, we can help the computer in understanding the human language. The main idea behind distributional hypothesis is that the words that occur in similar context tend to have similar meanings. Let each word be represented by a $|V|$ dimensional vector where $|V|$ is the length of the vocabulary. Instead of using a one-hot vector representation of the word, where only the index corresponding to ID of the word in the vocabulary is on, we also take into account the neighboring words. Let $c = 5$ be the context size around the center word (e.g. running). Given a large corpus, we search for the occurrence of the center word. Along with updating the frequency of the center word we also update the frequency of the context words around the center word in its vector. After looping over the entire corpus, the frequency of the central word and neighboring words are normalized to get the probability distribution.

Distributional word representations are based on co-occurrence matrix X of size |V| by |C| where |V| is the size of vocabulary. The various choices for co-occurrence matrix X are given below.

a) Word-document matrix

It is typically used to capture similarity among documents. Each word in the vocabulary V of the corpus is a row $x_{i:}$ in the matrix and the documents represent the column $x_{:j}$ of the matrix. The documents are represented as a bag of words. In mathematics, a bag is like a set where duplicates are allowed. Consider the following document containing a single sentence "to be or not to be". This document contains 6 tokens and 4 types.

$V = \{to, be, or, not\}$ and the bag of word representation is $\{2, 2, 1, 1\}$

Let our corpus contain m words and n documents. The entry $x_{ij} = 0$ if the word i does not occur in document j else $x_{ij} = frequency\ of\ word\ i\ in\ document\ j$.

The row vector $x_{i:}$ tells the meaning of the word i and the column vector $x_{:j}$ helps in determining the topic of the document based on the types of words in the document. Though the bag-of-words approach does not take into account the order of words in a document but these vectors still efficiently capture similarity of documents. This is based on the fact that if two documents are related then the choice of words in these documents will be similar. [42] used this approach to measure document similarity. [40] focused on row vectors to capture similarity among words.

b) Word-context matrix

In a word-context matrix the context can be words, sentences, phrases, paragraph or documents. This matrix is based on the distributional hypothesis – words that occur in similar context have similar meanings. Two row vectors with similar pattern of numbers indicate

similarity in their meanings. If the context in a word-context is a word, then it is a word-word co-occurrence matrix. Consider the following sentences:

a) *I love reading*.

b) I like pizza.

c) I like NLP.

The word-word co-occurrence matrix X is given as

	I	***like***	***love***	***NLP***	***pizza***	***reading***
I	*0*	*1*	*1*	*0*	*0*	*0*
like	*1*	*0*	*0*	*1*	*1*	*0*
love	*1*	*0*	*0*	*0*	*0*	*1*
NLP	*0*	*1*	*0*	*0*	*0*	*0*
pizza	*0*	*1*	*0*	*0*	*0*	*0*
reading	*0*	*0*	*1*	*0*	*0*	*0*

Table 2. . Example of a word-word co-occurrence matrix with context size=1

The distributional vectors capture the similarity among words based on the similarity among neighboring words. These vectors can be compared using measures like cosine similarity. These vectors are of size |V| - typically the value of v can be in hundreds of thousands. The storage and computation of these vectors can be expensive. We need to reduce the size of vectors from $\mathbb{R}|V|$ to d where $|d| \ll |V|$ in such a way that the low d-dimensional vector encodes the similarity among word vectors.

3.2.1.2 Distributed Representation/Word embedding

Word embedding is the vector representation of high dimensional distributional word vectors of size |V|, where V is the vocabulary, into low d – dimensional (d is typically between 50 to 1000) vector where$|d| \ll |V|$. In 1980, the neural networks were revived in the research community and the distributed representation was the core topic. Initially the distributed representation was associated with the way human brain would be learning new concepts, as distributed representations contain many features that help in generalizing to new concepts which are similar to the known ones. The distributed representation is different from one-hot vector representation in which only one or few indices are on. The distributed representation of words has helped in fighting the *curse of dimensionality* issue that has made the statistical language modelling task intrinsically difficult [9]. According to [9], every word in the vocabulary is represented as an embedding in vector space. These distributed word vectors are real valued vectors. The dimensions of these word vectors range from 50-1000 which is much smaller than the size of the vocabulary. The joint probability function of the sequence of words is expressed in terms of these newly learned distributed word vectors of the words in the given sequence. The models simultaneously learn the distributed word vectors and the parameters of the joint probability function. To maximize the log-likelihood of the training data, the models iteratively tuned the parameters using stochastic gradient descent. There are many popular methods for learning distributed representations of words like matrix factorization, singular value decomposition, neural network and deep learning [4, 9, 10, 11,12, 46]. The distributed representation of words generalizes easily to an unseen sequence of words as compared to n-gram models. The model will assign high probability to this unseen sequence of words if the words are similar to the words in the seen sentences. To

illustrate, consider that the model has already seen the sentence "*The cat is meowing in the room*". Suppose the model is given a new unseen sentence "*The dog is barking in the bathroom*". The model should assign a high probability to this unseen sentence as in the distributed representation, *cat* is similar to *dog*, *meowing* is semantically closer to *barking* and the vectors for *the* and *a,* and *room* and *bathroom* are also similar in the vector semantic space. In the following sections we will discuss various methods for learning distributed representation of words in vector space.

3.3 Methods to Obtain Distributed Representation of Words

3.3.1 Latent Semantic Analysis

Latent Semantic Analysis (LSA) [43] is a popular method for creating vector representation of the words in low-dimensional space. It is a statistical technique where the input is raw text segmented into sentences or tokenized into words. LSA does not need annotated corpus or handcrafted features [44].

The raw text is converted to a word-context matrix X where each row in X represents a unique word in the vocabulary extracted from the text. Each column in X represents the context. Generally, in LSA the context is a document but it can be a paragraph, sentence or even a word. The entry x_{ij} in the matrix is the frequency of the word represented by $X_{i:}$ in the context $X_{:j}$.

The words in a document help in understanding the topic of documents but there exist certain words called stop words like *the, a, an, she etc.* that occur in all the document frequently and do not help in encoding the meaning of the word or the context in which they occur. We need some weighing techniques that assign more weight to words that reflect the

meaning of the context and less weight to the stop words. One such technique is TF-IDF- which is a product of two components:

a) Term Frequency

b) Inverse Document Frequency

Term frequency is an indication of how frequently a word occurs in a document. In case, our context is a document and a word might occur more frequently in a long document as compared to a short document. Thus, we normalize the raw count by the total number of words in the document to account for the different length of documents in the corpus. Let w_i represent the i^{th} word in the matrix X and d_j be the document under consideration. So

$$TF(w_i) = \frac{count(w_i): w_i \in d_j}{|d_j|} \tag{29}$$

where $|d_j|$ is the total number of words in document d_j.

Inverse Document Frequency measures if the word is common or rare across all context.

$$IDF(w_i) = log\left(\frac{N}{|d \in D: w_i \in d|}\right) \tag{30}$$

Here, N is the total number of documents which is total number of columns in the matrix $|d \in D: w_i \in d|$ which represents total number of documents where the word w_i appears.

The next step in LSA is applying Singular Value Decomposition (SVD) to the matrix decomposing X into USV^T

$$X = USV^T \tag{31}$$

If the original matrix X is of size $m \times n$ then

U is of size $m \times m$ and the columns of U are eigenvectors of XX^T

V is of size $n \times n$ and the columns of V are eigenvectors of X^TX

S is a diagonal matrix of size $m \times n$ and the value of diagonal element, arranged in descending order, is the eigenvalue. The rows of U can be used as word embeddings for all the words in the vocabulary. To obtain low d-dimensional distributed word representation we can cut the singular values in matrix S at index d based on the desired percentage variance captured:

$$\frac{\sum_{i=1}^{d} \sigma_i}{\sum_{i=1}^{|V|} \sigma_i} \tag{32}$$

We take $U_{1:|V|,1:d}$ to be the word embedding matrix where each word is represented as a d-dimensional vector.

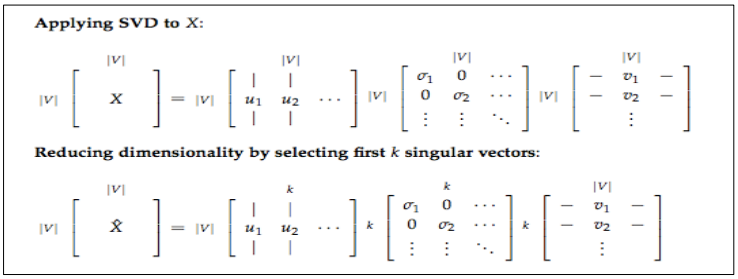


Figure 1. Effect of applying SVD o a square matrix of size |V|X|V|

There are certain problems associated with the SVD based methods:

a) The matrix changes often as the new words and size of the corpus frequently get updated.

b) As the training data is fixed and many possible word will not co-occur the matrix will be sparse.

c) SVD based methods are very expensive to train.

3.3.2 Neural Network Language Model (NNLM)

NNLM is another popular way of simultaneously learning distributed representation of words and the parameters of the joint probability functions of the sequence of words. Bengio et. al.

[9] have shown that distributed representation of words combined with neural network probability predictions has outperformed traditional n-gram language models in many statistical NLP tasks. Various experiments on distributed representation of words have shown meaningful linear substructures. A neural probabilistic language model was introduced in [9].

3.3.2.1 Neural Network Language Model Architecture

The NNLM consists of an input projection layer, non-linear hidden layer and an output layer.

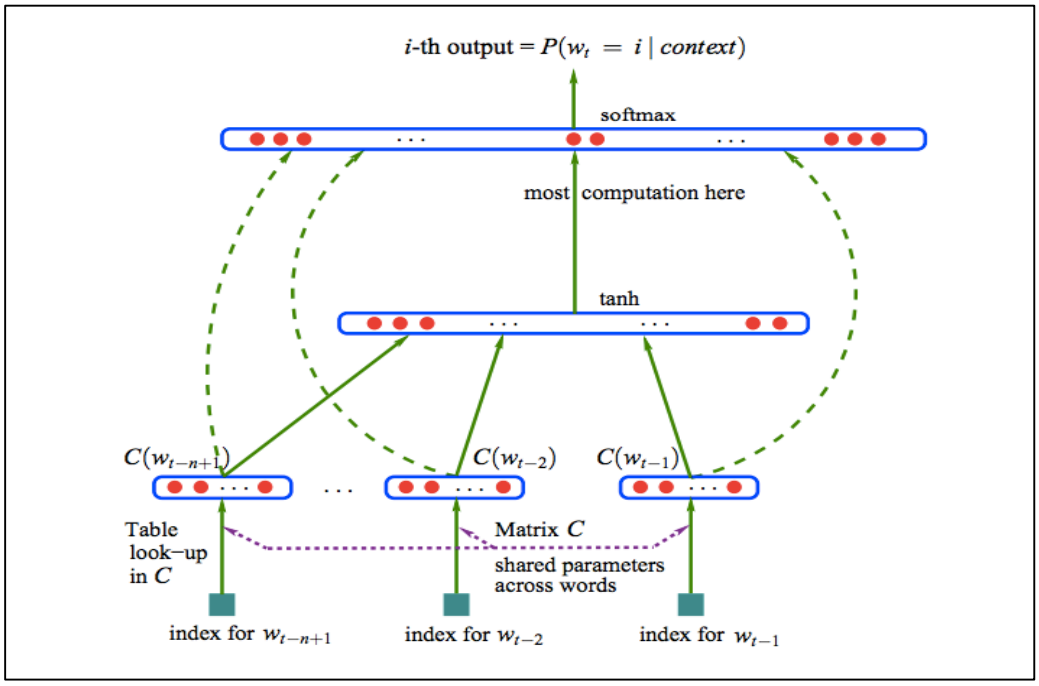


Figure 2. Architecture of neural net language model [9]

Using chain rule, the probability of a sequence of word is given as

$$P(w_1, w_2, \dots w_{t-1}, w_t) = P(w_1)P(w_2|w_1) \dots P(w_t|w_1, w_2 \dots w_{t-1}) \quad (33)$$

Using Markov assumption, we limit the history to $n-1$ words.

$$P(w_1, w_2, \dots w_{t-1}, w_t) = P(w_t|w_{t-n+1}, \dots w_{t-1}) \quad (34)$$

The main aim of NNLM in [9] is to learn a good model that maximize the log likelihood of output probability $\hat{P}(w_t|w_1^{t-1})$. The input to the NNLM are the indices of the context words in the vocabulary V of the model and the output of the model is a vector of size |V| where each feature i is the posterior probability of the word w_i at the index i in the vocabulary , $P(w_t = w_i|w_{t-n+1}, \dots w_{t-1})$.

In [9], initially the input context words are given as one-hot vector and as the model is trained each word w_{t-i} is mapped to d-dimensional feature vector $C_{w_{t-i}} \in \mathbb{R}^d$. Let the input vector $\vec{x}$ be represented by concatenation of $n-1$ feature vectors. This is the distributed representation of the word in vector space.

$$\vec{x} = (C_{w_{t-n+i},1}, \dots C_{w_{t-n+i},d}, \dots C_{w_{t-1},1}, \dots C_{w_{t-1},d}) \tag{35}$$

Let V be the input hidden layer weight matrix and W be the hidden output layer matrix. Let h be the number of hidden units and d be the dimensions of the distributed word vectors. Let b_i be the bias term and a_j be the hidden layer activation functions given by

$$a_i = \tanh(b_i + \sum_{j=1}^{(n-1)d} V_{ij}\, x_j) \tag{36}$$

The output layer activation function is given by

$$y_k = l_k + \sum_{i=1}^{h} W_{ki} a_k \tag{37}$$

The probabilistic function of the next word is estimated using the *softmax* function

$$P(w_t = w_i|w_{t-n+1}, \dots w_{t-1}) = \frac{e^{a_i}}{\sum_{k=1}^{V} e^{a_k}} \tag{38}$$

Along with matrix C the other parameters that the model needs to learn are bias vectors $\vec{b}$ and $\vec{l}$ and also the weight matrices V and W. Let θ denote all the parameters that the NNLM

needs to learn. The objective function of the NNLM is to maximize the log-likelihood of the training set

$$L(\theta) = \sum_{w \in T} log\, P(w|w_{t-n+1}, \dots w_{t-1}) \tag{39}$$

The model is trained using stochastic gradient descent based back propagation and the parameters are updated iteratively.

3.3.3 Word2Vec

Word2vec is a popular method for generating dense word embedding and it is very similar to feed-forward neural network language model [4]. In word2vec the hidden layer is not non-linear. As we have seen the neural network model learns the distributed representation of context word in the process of estimating the probability of next word given the $n-1$ context words. The main driver behind these dense word embedding is that words with similar contexts occur next to each other in the text. The neural models learn distributed representation of words by starting with random vectors and iteratively making them similar to the distributed word vectors that occur near them and less similar to the distributed word vectors that do not occur nearby. There are two algorithms for learning distributed representation of word in word2vec.

a) Continuous-Bag-Of-Word model (CBOW) [4]
b) Skip-gram model (SKIP) [4]

The basic difference between the CBOW and Skip-gram model is that the CBOW predicts the word based on the context while Skip-gram predicts the surrounding words given the current word.

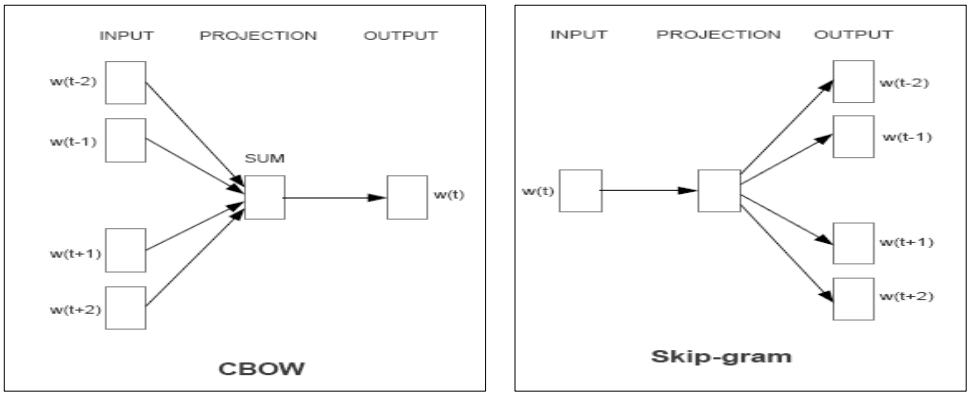


Figure 3. Difference between CBOW and skip -gram model [4]

The main advantage of word2vec is that they are fast and efficient to train.

3.3.3.1 Continuous-Bag-Of-Word model (CBOW)

Consider a very simple version of CBOW model [4] where only one context word c is given as input and the model has to predict the next word w. Figure 1 shows the simplified CBOW model where V is the size of vocabulary and N is the size of the hidden layer and the dimension of the word embedding produced by this model. The size of the output layer is same as the input layer. The size of output vector $\vec{y}$ is same as the cardinality of the vocabulary. The value at each index of the output vector represent the probability that next word is the word at index i of the vocabulary and so on. Thus

$$y_i = \mathrm{P}(\mathrm{w}|\mathrm{c}) \tag{40}$$

$$\sum_{V} y_i = 1 \tag{41}$$

The input to this network $\vec{x}_v$ is one-hot vector representation of the context word. Thus $x_k = 1$ and $x_{k'} = 0$ for $\forall\, k' \neq k$ [45].

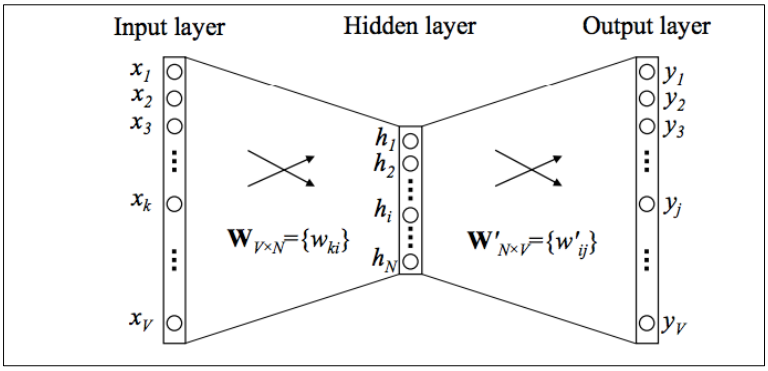


Figure 4. A simple CBOW model with only one word in the context [45]

There are two sets of weight matrices – one between input layer and hidden layer and is denoted by $\boldsymbol{W}$ and the other weight matrix between the hidden layer and output layer denoted by $\boldsymbol{W'}$. The size of the matrix $\boldsymbol{W}$ is $V\times N$ and of matrix $\boldsymbol{W'}$ is $N\times V$.

$$\vec{h} = \boldsymbol{W}^{\boldsymbol{T}}\vec{x} \tag{42}$$

Since $\vec{x}$ is a sparse vector with only one dimension on at the k^{th} position so the vector $\vec{h}$ is k^{th} row of $\boldsymbol{W}$. Let $W_{k:}$ be the k^{th} row of $\boldsymbol{W}$. $\vec{h}$ is represented as

$$\vec{h} = \boldsymbol{W}^{T}\vec{x} \;=\; W_{k:} \;=\; \vec{v}_c \tag{43}$$

Thus activation function of the inner hidden layer is a linear function. Suppose

$$y_i = \varphi(u_i) \tag{44}$$

and,

$$\vec{u} = \boldsymbol{W'}^{T}\vec{h} \tag{45}$$

$$u_i = W'_{:i}\,\vec{h} \tag{46}$$

Let us denote $W'_{:i}$ as $\vec{v}_w$. Therefore

$$u_i = \vec{v}_w\,\vec{h} \tag{47}$$

From eq. (43)

$$u_i = \vec{v}_w\vec{v}_c \tag{48}$$

y_i is the non linear function φ applied to u_i. Since we want to interpret y_i as a probability so we will use softmax – a log-linear classification model to obtain the probability distribution of the words.

$$y_i = \frac{e^{u_i}}{\sum_{i'} e^{u_{i'}}} \tag{49}$$

From eq. (48)

$$y_i = \frac{e^{\vec{v}_w \vec{v}_c}}{\sum_{w'} e^{\vec{v}_{w'} \vec{v}_c}} \tag{50}$$

where $w' \in$ {all the words in vocabulary V}. $\vec{v}_w$ and $\vec{v}_c$ are the parameter of this model that we need to find in a way that the likelihood of the model is maximized. Let θ denote the parameters of the CBOW model such that $\theta = \{\vec{v}_w, \vec{v}_c\}$. Let $L(\theta)$ denote the likelihood of the parameter.

$$L(\theta) = \prod_{w \in T} P(w|c) \tag{51}$$

On taking the log of the likelihood $L(\theta)$ denoted by l

$$l(\theta) = \sum_{w \in T} \log P(w|c) \tag{52}$$

From eq. (50)

$$l(\theta) = \sum_{w \in T} log \frac{e^{\vec{v}_w \vec{v}_c}}{\sum_{w'} e^{\vec{v}_{w'} \vec{v}_c}} \tag{53}$$

$$l(\theta) = \sum_{w \in T} \vec{v}_w \vec{v}_c - log \sum_{w'} e^{\vec{v}_{w'} \vec{v}_c} \tag{54}$$

On taking the derivative of log likelihood $l(\theta)$ w.r.t. $\vec{v}_w$

$$\frac{\delta l(\theta)}{\delta \vec{v}_w} = \vec{v}_c - \frac{1}{\sum_{w'} e^{\vec{v}_{w'} \vec{v}_c}} e^{\vec{v}_{w'} \vec{v}_c} \vec{v}_c \tag{55}$$

$$\frac{\delta l(\theta)}{\delta \vec{v}_w} = \vec{v}_c - P(w|c)\vec{v}_c \tag{56}$$

$$\frac{\delta l(\theta)}{\delta \vec{v}_w} = \vec{v}_c(1 - P(w|c)) \tag{57}$$

Using stochastic gradient descent, we can obtain the weight updating equation for hidden→output layer,

$$\vec{v}_w = \vec{v}_w - \eta \, \vec{v}_c(1 - P(w|c)) \tag{58}$$

Similarly, on taking derivative of log likelihood $l(\theta)$ w.r.t. $\vec{v}_c$, the weight updating equation for input→hidden layer is estimated by

$$\vec{v}_c = \vec{v}_c - \eta \, \vec{v}_w(1 - P(w|c)) \tag{59}$$

Multi-word context

In this case the input is not the single word context but multiple words $x_1, x_2, \cdots, x_c$.

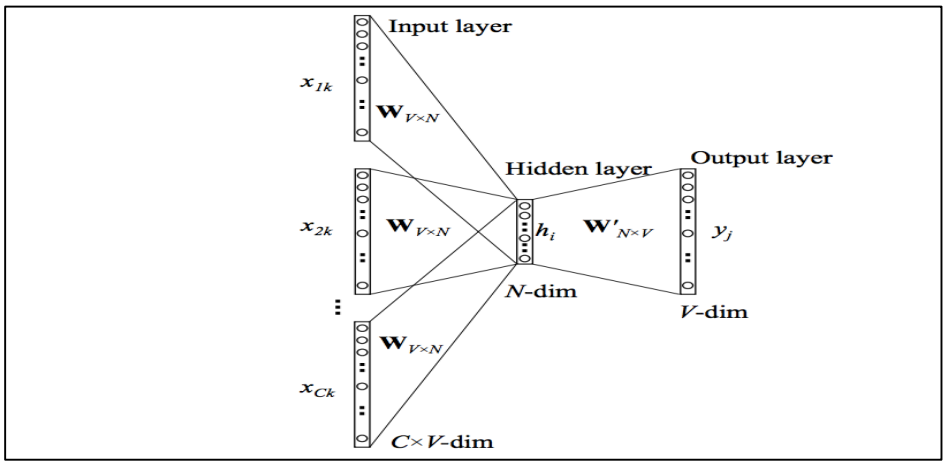


Figure 5. A CBOW model with C words (C>1) in the context [45]

The hidden layer output h is given by

$$\vec{h} = \frac{W^T(x_1 + x_2 + \cdots + x_c)}{C} \tag{60}$$

$$\vec{h} = \frac{\vec{v}_{w_1} + \vec{v}_{w_2} + \cdots + \vec{v}_{w_c}}{C} \tag{61}$$

C is the context window size. Let $L(\theta)$ denotes the likelihood of the parameter θ.

$$L(\theta) = \prod_{w \in T} P(w|c_1, c_2, \ldots c_C) \tag{62}$$

On taking the log of the likelihood denoted by $l(\theta)$

$$L(\theta) = \prod_{w \in T} P(w|c_1, c_2, \ldots c_C) \tag{62}$$

$$l(\theta) = \sum_{w \in T} log \frac{e^{\vec{v}_w \vec{h}}}{\sum_{w'} e^{\vec{v}_{w'} \vec{h}}} \tag{63}$$

$$l(\theta) = \sum_{w \in T} \vec{v}_w \vec{h} - log \sum_{w'} e^{\vec{v}_{w'} \vec{h}} \tag{64}$$

Using stochastic gradient descent, we can obtain the weight updating equation for hidden→output layer,

$$\vec{v}_w = \vec{v}_w - \eta \vec{h} - P(w|c_1, c_2, \ldots c_C)) \tag{65}$$

Similarly, on taking derivative of log likelihood $l(\theta)$ w.r.t. $\underline{v_c}$, the weight updating equation for input→hidden layer is estimated by

$$\vec{v}_c = \vec{v}_c - \frac{1}{C} \eta \, \vec{v}_w (1 - P(w|c_1, c_2, \ldots c_C)) \tag{66}$$

3.3.3.2 Skip-gram model

This model was introduced in [12]. In this model the aim is to predict the neighboring word given the central word as the input. The input is one-hot vector representation of the central word and is represented by $\vec{x}$.

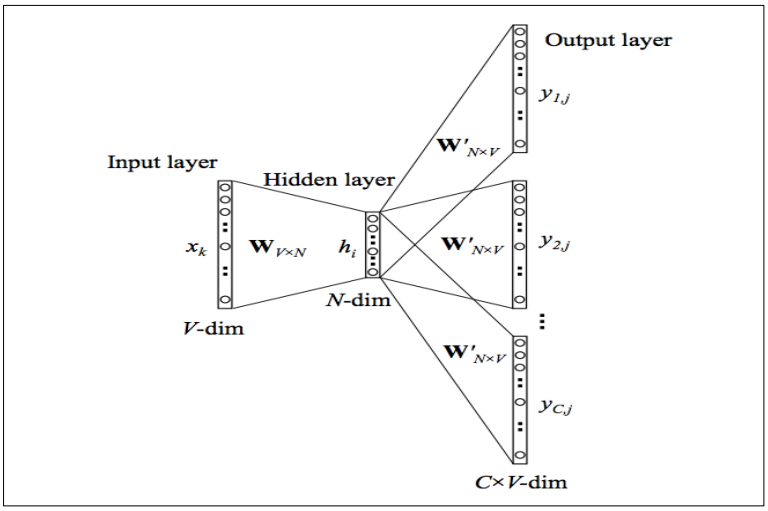


Figure 6. The skip gram model [45]

The input vector for the word $\vec{x}$ is denoted by $\vec{v}_c$. Similar to the single word context model of CBOW, there are two weight matrices denoted by W and W'. The output of hidden layer is given by

$$\vec{h} = W^T \vec{x} \; = \; W_{k:} = \; \vec{v}_c \tag{67}$$

Score vector for the output word is generated using

$$u_{cj} = u_j = \; W'^T \, \vec{h} = \vec{v}_{w_j} \vec{h} = \vec{v}_{w_j} \vec{v}_c \tag{68}$$

for $c = 1, 2 \ldots, C$

$v_{w_j}$is the output vector of j^{th} word in the vocabulary and it is also the j^{th} column of the output matrix $\boldsymbol{W'}$.

Using the softmax function, each of these score is converted to a corresponding probability. The objective function of this model is that probability vector generated to match the actual probabilities which is $y^{(i-c)}, \dots, y^{(i-1)}, y^{(i+1)}, \dots, y^{(i+c)}$, the one hot vector of actual output. This probability is estimated by the formula below.

$$P\left(w_{c,j} = w_{O,c} \middle| w_I\right) = y_{c,j} = \frac{e^{u_{c,j}}}{\sum_{j'=1}^{V} e^{u_{c,j'}}} \tag{69}$$

We need to define an objective function to evaluate the skip-gram model. Using the Naïve-Bayes assumption, given the central word, all the neighboring words are completely independent.

3.3.4 Optimizing Word2Vec Computational Efficiency

In both CBOW and skip-gram, the model learns two distributed representation of words in the vocabulary - input vector $\vec{v}_c$ and output vector $\vec{v}_w$. At the output of these models we have the softmax layer which gives us the probability of observing each word in the vocabulary following the given input context. One problem with this approach is that the dimension of the output softmax layer is in the order of the size of the vocabulary which can be in hundreds of thousands or even millions. When we are performing the calculations we need to compute all the activations in the hidden layer and calculate the softmax probabilities. This is computationally expensive both in terms of memory and time required to train the models as it is required for every training instance. Thus we need an efficient way of defining this probability distribution function. One approach is to limit the number of output vectors to be updated per training instance. There are two ways to achieve this – *hierarchical softmax* and *negative sampling*.

3.3.4.1 Hierarchical Softmax

Hierarchical Softmax algorithm was introduced by Morin and Bengio [46]. In hierarchical softmax instead of $|V|$ dimensional output softmax layer, we create a binary tree representation of output layer where the leaves of the tree are the words in the vocabulary. For each word there is a unique path from the root of the tree to the leaf representing that word and this path is used to estimate the probability of the output word.

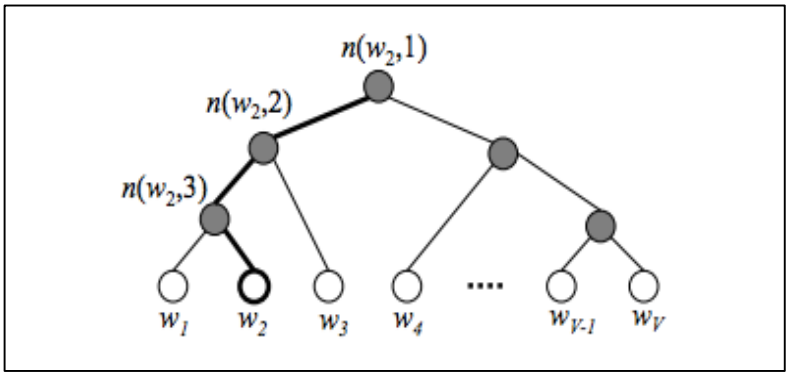


Figure 7 Hierarchical Softmax [45]

Let $n(w,j)$ be the j^{th} node on the path from root to word w and let $L(w)$ be the length of the path. There are $V-1$ inner units and for each inner unit there is an output vector $\vec{v}'_{n(w',j)}$. The probability of the k^{th} word w_k in the vocabulary to be the output word is given in [12] by

$$P(w=w_k)=\prod_{j=1}^{L(w)-1}\sigma(\llbracket n(w,j+1)=ch(n(w,j))\rrbracket.\vec{v}'^{T}_{n(w',j)}\vec{h} \tag{70}$$

where $ch(n)$ is the left child on inner unit n and $\vec{h}$ is the activation value of hidden layer.$\llbracket x\rrbracket$ is defined as

$$\llbracket x\rrbracket=\begin{cases}1 & if\ x\ is\ true\\ -1 & otherwise\end{cases} \tag{71}$$

In word2vec, binary Huffman tree is used as it results in faster training.

3.3.4.2 Negative Sampling

Negative sampling is the other efficient computation scheme for calculating the output probability in word2vec. It is a completely different optimization function. It is based on the assumed to be a classification problem. Consider two sets D and D' where D is a set of all (w, c) pairs that are in the corpus and D' is a set of all (w, c) pairs that are not in the corpus. Given a pair as an input to the classifier, it outputs one if it is in the corpus else 0. We define two probabilities based on the output:

$\Pr(z = 1|(w, c))$ is the probability that (w, c) is in D and $\Pr(z = 1|(w, c))$ is the probability that (w, c) is in D'

Using logistic regression to solve this classification problem,

$$\Pr\big(z = 1\big|(w, c)\big) = \frac{1}{1 + e^{-\vec{v}_c \vec{v}_w}} \tag{72}$$

$$\Pr\big(z = 0\big|(w, c)\big) = \frac{1}{1 + e^{\vec{v}'_c \vec{v}_w}} \tag{73}$$

The joint probability of the distribution is given by

$$\mathrm{P}\big(z\big|(w, c)\big) = \left(\frac{1}{1 + e^{\vec{v}'_c \vec{v}_w}}\right)^z \left(\frac{1}{1 + e^{\vec{v}'_c \vec{v}_w}}\right)^{1-z} \tag{74}$$

$\vec{v}_c$ and $\vec{v}_w$ are the parameters that the model needs to learn. By maximum likelihood estimation,

$$L(\theta) = arg \max_{\theta} \prod_{(w,c) \in D \cup D'} \mathrm{P}\big(z\big|(w, c)\big) \tag{75}$$

The objective function is to maximize the log likelihood of training data

$$l(\theta) = arg\max_{\theta} \sum_{(w,c)\in D\cup D'} z\, log\left(\frac{1}{1+e^{-\overrightarrow{v'}_c\vec{v}_w}}\right) + (1-z)\left(\frac{1}{1+e^{\overrightarrow{v'}_c\vec{v}_w}}\right) \tag{76}$$

$$l(\theta) = arg\max_{\theta} \sum_{(w,c)\in D} log\left(\frac{1}{1+e^{-\overrightarrow{v'}_c\vec{v}_w}}\right) + \sum_{(w,c)\in D'} \left(\frac{1}{1+e^{\overrightarrow{v'}_c\vec{v}_w}}\right) \tag{77}$$

In [12] the negative sampling objective function is given by

$$\log\sigma\left(\overrightarrow{v'}_c\vec{v}_w\right) + \sum_{i=1}^{k} \mathrm{E}_{w_i}\sim P_n(w) \log\sigma\left(\overrightarrow{v'}_c\vec{v}_w\right) \tag{78}$$

where k is the number of negative samples sampled from $P_n(w)$ and σ is the sigmoid function such that

$$\sigma = \frac{1}{1+e^{-z}} \tag{79}$$

The best choice for $P_n(w)$ was found to be unigram distribution $U(w)$ raised to 3/4 power.

3.3.5 Global Vector for Word Representation (GloVe)

GloVe [13] is an unsupervised machine learning method for representing words in the vector space model. GloVe is a global log-bilinear model that uses two methods – matrix factorization and local context window size [13]. In GloVe a word-word co-occurrence matrix, denoted by X is used. The entries X_{ij} denote the count of the event when the word at the j^{th}index in the vocabulary V occurs in the context of the word at the i^{th}index in V. Let X_i denote the count of the event when any word in the vocabulary occurred in the context of the word i and is estimated by

$$X_i = \sum_{k\in V} X_{ik} \tag{80}$$

Let P_{ij} be the probability of the event when word j occurred in the context of the word i and is calculated as

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} \tag{81}$$

An example in [13] shows that the meaning of a word can be encoded by using the ratio of co-occurrence probabilities. Two word $i = ice$ and $j = steam$ are chosen to understand their relationships in this illustration. The probabilities of these two words with selected context word are calculated from the corpus of around six billion tokens. The ratio of co-occurrence probability is given by $\frac{P_{ik}}{P_{jk}}$. For word k related to ice, the ratio $\frac{P_{ik}}{P_{jk}}$ will be large and if k is related to steam this will be small. But if k is related to both target words or not related to both the words then this ratio will be close to 1. Table below shows these probabilities and confirm our arguments.

$\boldsymbol{Probability}$ & $\boldsymbol{ratio}$	$\boldsymbol{k = solid}$	$\boldsymbol{k = gas}$	$\boldsymbol{k = water}$	$\boldsymbol{k = fashion}$
$\boldsymbol{P(k\|ice)}$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$\boldsymbol{P(k\|gas)}$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\boldsymbol{P(k\|ice)/P(k\|gas)}$	8.9	8.5×10^{-2}	1.36	0.96

Table 3. Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus [13]

The general form of the GloVe model is given by,

$$F\left(w_i, w_j, \vec{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{82}$$

where $w \in \mathbb{R}^d$ is d-dimensional word embedding and $\vec{w}$ is d-dimensional context word embedding. GloVe produces word vectors by minimizing the following cost functions using gradient descent

$$J = \sum_{i,j} f\left(X_{ij}\right) \left(w_i^T \vec{w}_j + b_i + \vec{b}_j - \log X_{ij}\right)^2 \tag{83}$$

b_i and $\vec{b}_j$ are the parameters measuring word specific biases. The function $f(x)$ should have the following properties:

a) $f(0) = 0$

b) $f(x)$ should be non-decreasing to prevent over-weighing of rare co-occurrences

c) $f(x)$ should be relatively small for large value of x so that frequently appearing co-occurrences are not overweighed.

One function with above properties which works well with the model is

$$f(x) = \begin{cases} 0 & x = 0 \\ (x/x_{\max})^{\alpha} & x < x_{\max} \\ 1 & otherwise \end{cases} \tag{84}$$

CHAPTER 4: EXPERIMENTS AND EVALUATION

4.1 The Data

The training and test data provided by Microsoft Research Sentence completion challenge [18] are used to assess the performance of our methods. The test set consists of 1040 fill-in-the-blank sentences similar to the SAT style sentence completion questions. The aim of these questions is to test the student's ability to select the correct word that is meaningful in the given context. The correct determination of missing words cannot be made just on the basis of grammatical correctness. The 1040 sentences are picked from the five Sir Arthur Conan Doyle's Sherlock Holmes novels. Each question is provided with 5 choices and the task is to identify which word among them is the correct. The missing word in these sentences is infrequent in the training data. The other four alternatives for the missing word are generated using n-gram model. The n-gram model is trained using 500 19th century novel from Project Gutenberg. All the five alternatives look plausible and the models are required to encode semantic meaning for making a correct guess. We used the same training set to train our models.

```
The sun had set and _____ was settling over the moor.
    a) dusk
    b) mischief
    c) success
    d) disappointment
    e) laughter
```

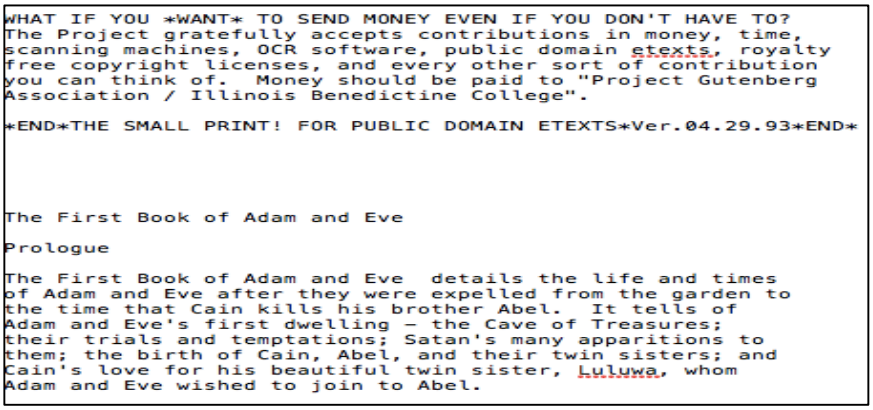
```
The darkness was _____ , but much was still hidden by the shadows.
    a) healed
    b) rising
    c) ponderous
    d) attractive
    e) neglected
```

Figure 8. Two example MSR sentence completion questions [18]. The correct answers are a and b respectively.

4.2 Data pre-processing

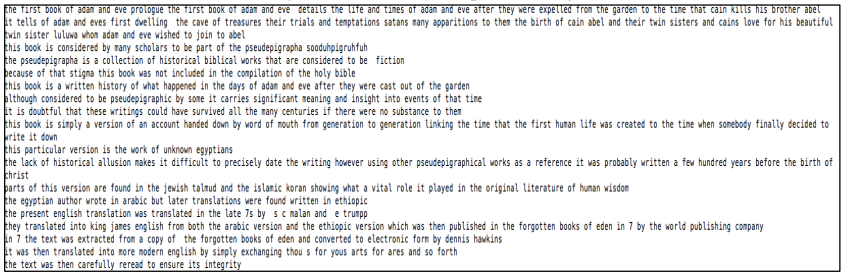
4.2.1 Cleaning the text

Before training and evaluating our methods on sentence completion task, we need to preprocess our training and test data. The MSR Sentence Completion Challenge data [18] provided a training corpus that contains 19th - century novel text from Project Gutenberg. The training set consists of 522 text files. These texts contain meta data and source text information. We stripped all the metadata tags, author name, text source and extracted the novel text. The entire text is converted to lowercase. We also removed the punctuation marks. After preprocessing, each sentence in the text is a single line. We ignored the task of lemmatization – replacing the word with its corresponding vocabulary lemma as this would be time-consuming. We also did not perform stemming – removing the suffixes from the words, as this would introduce word ambiguity.



```
WHAT IF YOU *WANT* TO SEND MONEY EVEN IF YOU DON'T HAVE TO?
The Project gratefully accepts contributions in money, time,
scanning machines, OCR software, public domain etexts, royalty
free copyright licenses, and every other sort of contribution
you can think of.  Money should be paid to "Project Gutenberg
Association / Illinois Benedictine College".

*END*THE SMALL PRINT! FOR PUBLIC DOMAIN ETEXTS*Ver.04.29.93*END*



The First Book of Adam and Eve

Prologue

The First Book of Adam and Eve  details the life and times
of Adam and Eve after they were expelled from the garden to
the time that Cain kills his brother Abel.  It tells of
Adam and Eve's first dwelling - the Cave of Treasures;
their trials and temptations; Satan's many apparitions to
them; the birth of Cain, Abel, and their twin sisters; and
Cain's love for his beautiful twin sister, Luluwa, whom
Adam and Eve wished to join to Abel.
```

Figure 9. Text before processing



```
the first book of adam and eve prologue the first book of adam and eve  details the life and times of adam and eve after they were expelled from the garden to the time that cain kills his brother abel
it tells of adam and eves first dwelling  the cave of treasures their trials and temptations satans many apparitions to them the birth of cain abel and their twin sisters and cains love for his beautiful
twin sister luluwa whom adam and eve wished to join to abel
this book is considered by many scholars to be part of the pseudepigrapha sooduhpigruhfuh
the pseudepigrapha is a collection of historical biblical works that are considered to be  fiction
because of that stigma this book was not included in the compilation of the holy bible
this book is a written history of what happened in the days of adam and eve after they were cast out of the garden
although considered to be pseudepigraphic by some it carries significant meaning and insight into events of that time
it is doubtful that these writings could have survived all the many centuries if there were no substance to them
this book is simply a version of an account handed down by word of mouth from generation to generation linking the time that the first human life was created to the time when somebody finally decided to
write it down
this particular version is the work of unknown egyptians
the lack of historical allusion makes it difficult to precisely date the writing however using other pseudepigraphical works as a reference it was probably written a few hundred years before the birth of
christ
parts of this version are found in the jewish talmud and the islamic koran showing what a vital role it played in the original literature of human wisdom
the egyptian author wrote in arabic but later translations were found written in ethiopic
the present english translation was translated in the late 7s by  s c malan and  e trumpp
they translated into king james english from both the arabic version and the ethiopic version which was then published in the forgotten books of eden in 7 by the world publishing company
in 7 the text was extracted from a copy of  the forgotten books of eden and converted to electronic form by dennis hawkins
it was then translated into more modern english by simply exchanging thou s for yous arts for ares and so forth
the text was then carefully reread to ensure its integrity
```

Figure 10. Text after processing

4.2.3 Building Vocabulary

The next step after cleaning the data is to form a vocabulary that maps the words in the text to a unique ID. After cleaning the training data, every document was segmented into sentences and each sentence was represented as a single line in the text file. We then parsed this document line by line to convert it into a bag-of-word representation.

4.3 Methodology

We evaluated various techniques for word representation in vector space, namely LSA, Word2Vec and GloVe. The trained models select the answers based on the cosine similarity between each answer choice and other words in the sentences. The cosine similarity between two word vectors is defined as

$$cosine\_sim(\vec{w}_1, \vec{w}_2) = \frac{\vec{w}_1 . \vec{w}_2}{|\vec{w}_1||\vec{w}_2|} \tag{85}$$

The sentence with the highest score is selected as the answer.

4.3.1 Latent Semantic Analysis (LSA)

LSA is a technique particularly useful in distributional semantics for analyzing the relations between documents based on the words in the documents and is used to extract topics of the document. The first step in LSA is to create a word – document frequency matrix where each word is represented as a row in matrix and each column is the document. As the main task is predicting the missing word in the sentence we created a word – sentence matrix where each sentence is in the preprocessed training text set. This is done as explained below.

Consider a training corpus C containing n sentences T_i for $i = 1, \ldots . n$ denoted by $C = \{T_1, T_2, \ldots, T_n\}$ where each element T_i is a sequence of words w_j for $j = 1, \ldots . r$, $T_i = \{w_1, w_2, \ldots, w_r\}$. A dictionary D of m words d_i is constructed from the training corpus given by $D = \{d_1, d_2, \ldots, d_m\}$. The words in the dictionary are mapped to the index i. The word-sentence matrix X of size $m \times n$ is created where each row i represents the word d_i in D. The element x_{ij} is the term frequency which is frequency of word d_i in the document T_j. We normalized the values in the frequency matrix using TF-IDF weighing scheme which decreases the frequency of stop words. The frequency matrix is very high-dimensional. To obtain the distributed representation of words we apply the SVD to the matrix X and obtain the best d rank approximation of X

$$X_d = U_d S_d V_d^T \tag{86}$$

where U_d is a matrix of size $m \times d$, S_d is a diagonal matrix of size $d \times d$ and V_d is a matrix of size $n \times d$.We can evaluate the distributed representation of words in the dictionary D by multiplying U_d and S_d. The rows in the new matrix can be compared using cosine similarity.

The LSA model was trained using Gensim package [49]. We followed Zweig et al [19] approach to find the score for each option provided for the sentence completion task. Let LSA_Score be the score of option a for question Q.

$$LSA\_Score(a, Q) = \sum_{w \in Q, w \neq a} cosine\_sim(\vec{w}, \vec{a}) \tag{87}$$

The answer a with the highest score is selected as the answer.

4.3.2 Word2Vec

Word2vec has gained a lot of traction in the news and is a popular method for creating state-of-art distributed word representations. As discussed before word2vec uses a feed-forward neural network language model to predict the missing word given the context around the word [3, 11]. We evaluated the performance of word2vec models, trained using the Microsoft Research training corpus that contains around 50M words, on the Microsoft Research sentence completion test set. The word2vec models were trained using CBOW and Skip-Gram methods. In our experiments we used both, hierarchical softmax and negative sampling, to train the word2vec CBOW and skip gram language models. The performance of word2vec models depends on the hyper-parameter choice. We trained the word2vec models by using various hyper-parameter choices:

1. Word2vec offers two different architectures – CBOW and skip-gram. We trained both the models. To train word2vec models we used Python implementation of word2vec in the Gensim package [49].
2. The word2vec uses softmax function to estimate the probability prediction of the central word in case of CBOW model and the probability prediction of the context word if skip-gram model is being used. As these models are trained using very large corpus where dictionary size can be in hundreds of thousands, efficient methods are required to compute the softmax probabilities. Word2vec suggested two training algorithm - hierarchical softmax and negative sampling for efficient computation of the output probabilities.
3. We varied the dimensionality of word vectors to see if high dimensional word vectors performed better than low dimensional.

4. Finally, we also trained models using different context window size of 5 and 10.

4.3.2.1 Continuous Bag of Word

The CBOW language model predicts the central word given C context words before and after the central word. The CBOW model was trained using two different context window sizes – 5 and 10. We also trained the CBOW model to produce word vectors of different dimensions ranging from 300-1000. Let $CBOW\_Score$ be the score of option a for the question, Q. The answer a with the highest score is selected as the answer.

$$CBOW\_Score(a, Q) = \sum_{w \in Q, w \neq a} cosine\_sim(\vec{w}, \vec{a}) \tag{88}$$

4.3.2.2 Continuous Skip-Gram

The skip-gram language model of word2vec predicts the C context words before and after the given central word. Similar to CBOW, we trained the skip-gram model with different context window size – 5 and 10. We also trained the skip-gram model to produce word vectors of different dimensions ranging from 300-1000. Let $CKIP\_Score$ be the score of option a for the question, Q. The answer a with the highest score is selected as the answer.

$$CSKIP\_Score(a, Q) = \sum_{w \in Q, w \neq a} cosine\_sim(\vec{w}, \vec{a}) \tag{89}$$

4.3.3 GloVe

GloVe is another unsupervised learning algorithm for obtaining vector space representation of words which is trained on word-word co-occurrence matrix. The distributed word vectors produced have shown 75% accuracy on word analogy task. Better performance than related

models on word similarity and named entity recognition task is also reported [13]. We constructed unigram counts from the corpus, and thresholds from the resulting vocabulary based on the minimum frequency count of 5. After forming the vocabulary, we created the word-word co-occurrence matrix using the context window size of 5 and 10. The GloVe model was trained using AdaGrad - an algorithm for gradient-based optimization. The order in which data is presented to stochastic gradient descent algorithm is critical. If input data fed to algorithm is in a meaningful form, this can bias the gradient and will result in poor convergence. To avoid the biasing issue, we randomly shuffle the co-occurrence matrix and then train the GloVe model on this shuffled co-occurrence frequency matrix. We used the code provided at http://nlp.stanford.edu/projects/glove/ to train the GloVe models. Let $GloVe\_Score$ be the score of option a for the question, Q. The answer a with the highest score is selected as the answer.

$$GloVe\_Score\ (a, Q) = \sum_{w \in Q, w \neq a} cosine_{sim}(\vec{w}, \vec{a}) \tag{90}$$

4.4 Results and Discussions

4.4.1 Analysis of word2vec performance on tuning various hyper-parameters

For word2vec, we evaluated the performance of CBOW and skip-gram models on Microsoft Research Sentence Completion Challenge data set. The performance of these models on semantic NLP tasks depends on the tuning of various hyper-parameters, particularly dimensionality of word embedding, context-window size and training algorithm. We trained both skip-gram and CBOW using hierarchical softmax and negative sampling algorithm.

These models were trained to produce word embedding of different dimensions. We also used two context sizes - 5 and 10.

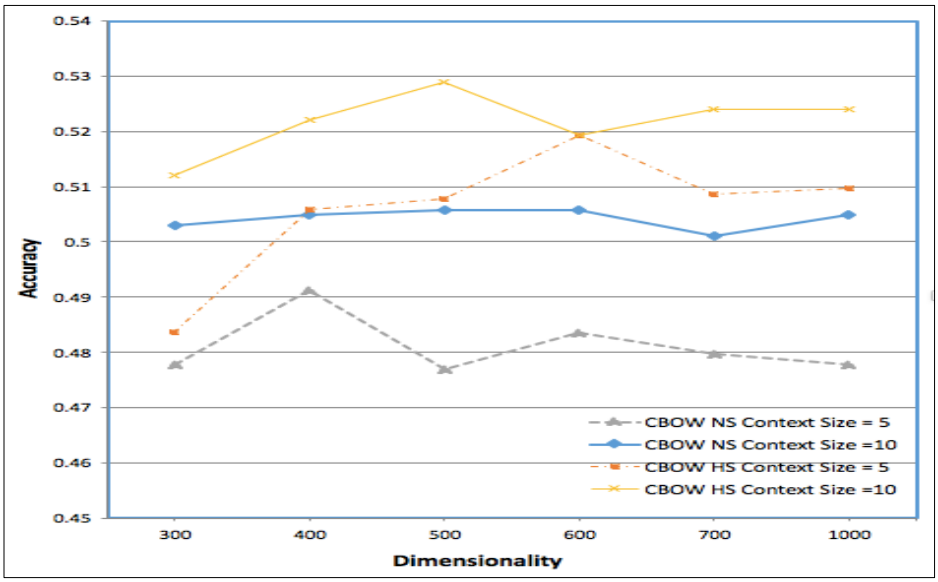


Figure 11. Performance analysis CBOW model trained using hierarchical softmax and negative sampling algorithm using different context size window.

The CBOW model trained using hierarchical softmax performed better than the CBOW model trained using negative sampling for both context window size 5 and 10. Except for vector dimension of 300, CBOW model trained with context size 5 and training algorithm hierarchical softmax performed better than the model trained with context size 10 and negative sampling. Thus for negative sampling algorithm word vectors of dimensions less than 500 perform better when larger context size is used. In case of CBOW model hierarchical softmax worked better even with smaller context size.

We performed the similar analysis on the performance of word embeddings of different dimensions produced by training skip-gram model.

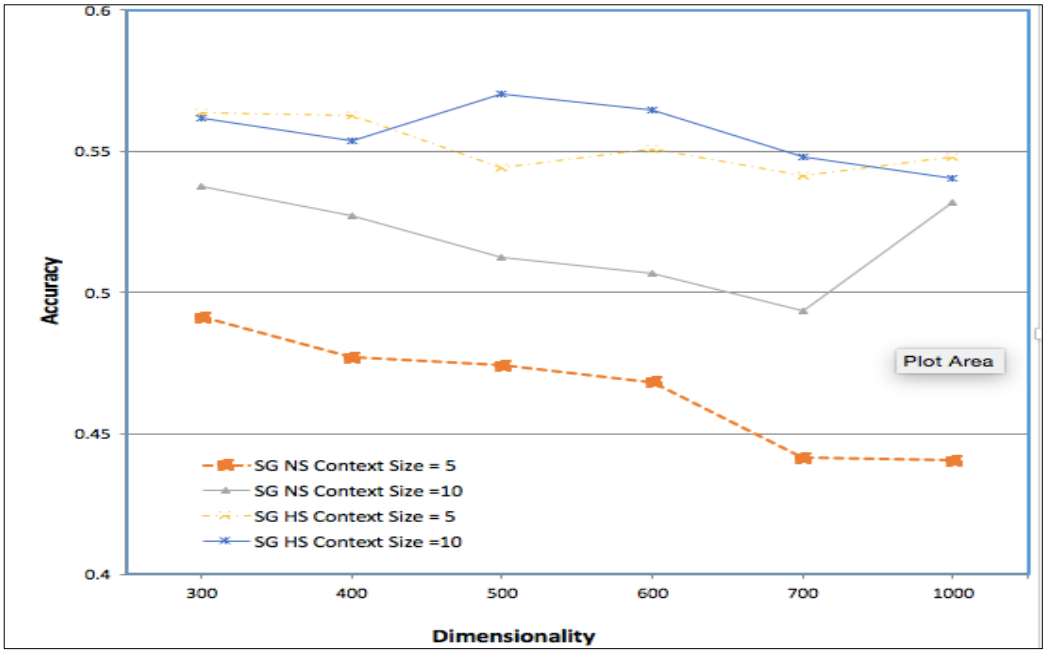


Figure 12. Performance analysis of skip-gram model (SG) trained using hierarchical softmax (HS) and negative sampling (NS) algorithm for different context window size

We found that for both context size of 5 and 10, the skip gram model trained using hierarchical softmax outperformed the skip-gram model trained using negative sampling.

In both CBOW and skip-gram model the hierarchical softmax is better in predicting the missing word in the sentence than the negative sampling. This is because the word missing from the sentence is an infrequent one and as per [3, 11] hierarchical softmax is better for infrequent words.

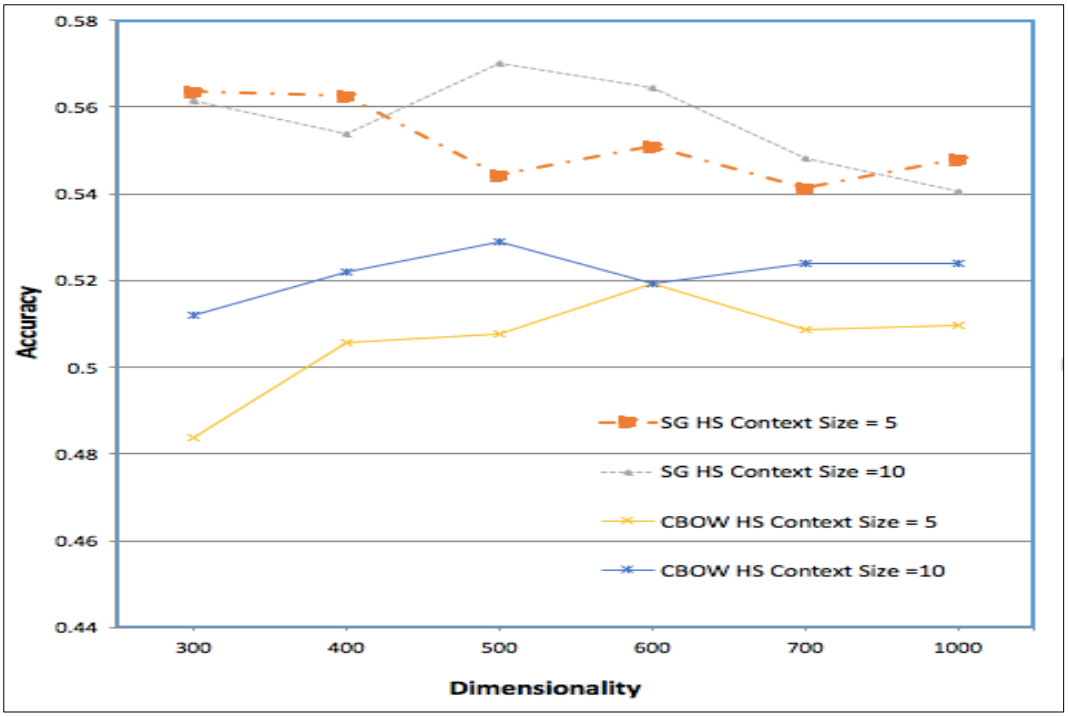


Figure 13. Performance analysis of skip-gram and CBOW model trained using hierarchical softmax algorithm on different context window size.

Thus, in space of hyper-parameters we found that the hierarchical softmax algorithm is a better choice for sentence completion task. The optimal performance of hierarchical softmax algorithm is found for skip-gram model trained to learn distributed representation of words of size 500 for context window of size 10.

4.4.2 Analysis of GloVe model performance on tuning various hyper-parameters

For GloVe we tuned two parameters - dimension of word embedding and context size. The GloVe model prediction accuracy is within 49.1% to 50% for the context size of 10 and the dimensionality of word vectors is between 300-1000. For context size of 5 and similar range for size of word vectors the Glove model performance degraded to 46%.

The context-size hyper-parameter has a greater impact on the quality of the word embeddings produced by GloVe than the dimensionality of the vectors. The GloVe model

performs dimensionality reduction on a word-word co-occurrence matrix to produce word embedding. The large context size captures greater syntactic and semantic similarities among words that occur near each other in the text which improve the quality of word embedding.

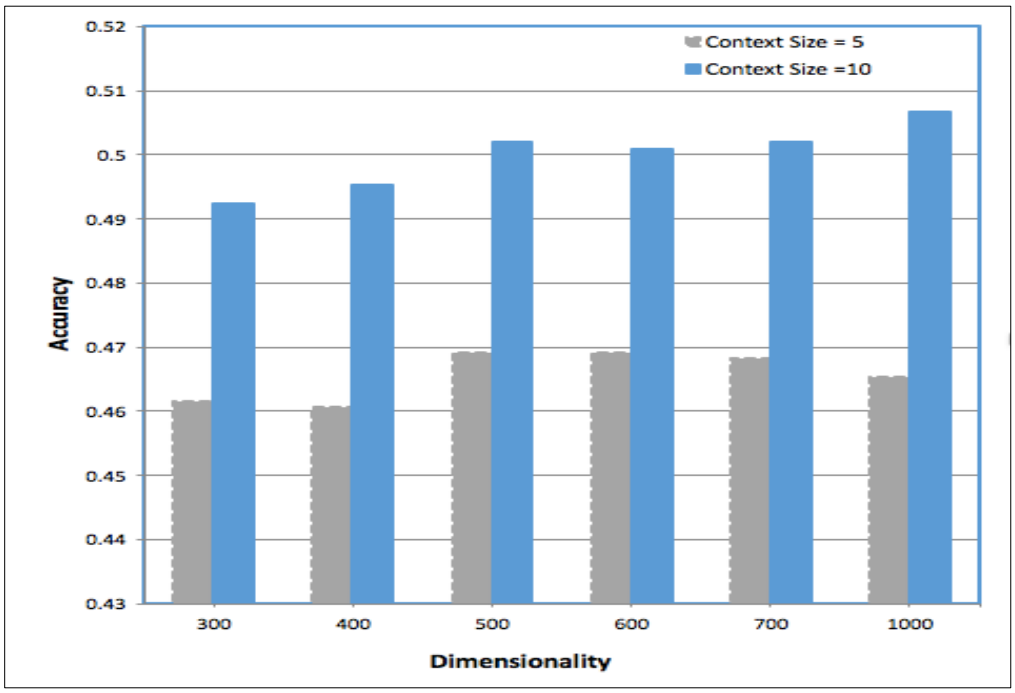


Figure 14. Performance analysis of GloVe trained with different context window size.

4.4.3 Comparison of LSA, CBOW, Skip-Gram and GloVe

We learned the distributed representation of words using three popular word embedding techniques – LSA, CBOW, skip-gram and GloVe and compare their performances on Microsoft Research sentence completion challenge set. For CBOW, skip-gram and GloVe we compared the models that were trained using the context size of 10 and the dimension of word vectors is 500. For CBOW and skip-gram model we used hierarchical softmax as the training algorithm based on analysis in section 4.1. Table 3 summarizes the result of this evaluation.

Method	Accuracies (%)
LSA	37.1
CBOW	52.9
Skip-Gram	57.0
GloVe	50.6

Table 4. Comparison of performance of various models on the Microsoft Sentence Completion

As we can see the skip-gram model has outperformed all other word embedding generation technique. This is because skip-gram works well with small amount of training data and can learn better word representation for rare or infrequent words in the corpus and in the SAT style MSR sentence completion test set the word missing in the sentence is an infrequent word. In our experiments, LSA performed the worst on the test set.

We also compared the performance of different word vectors models on different vector size. In general, the assumption is the quality of word vectors improve as we increase the dimensions.

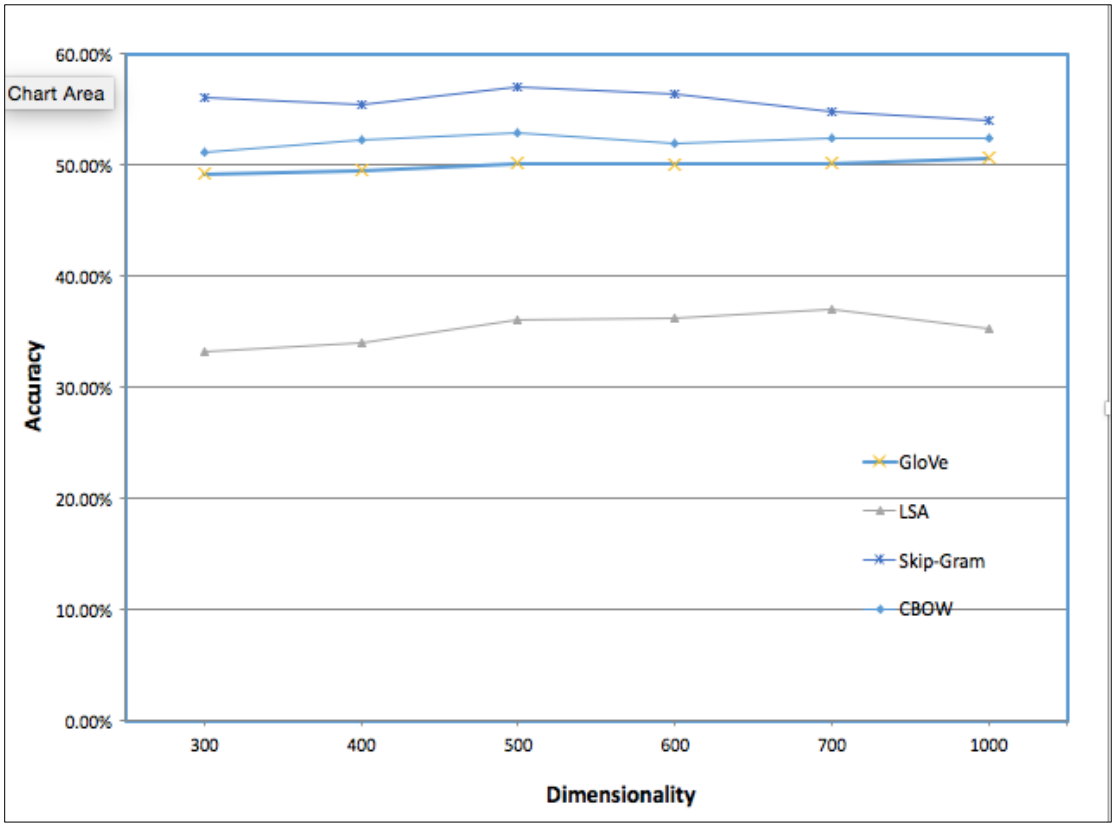


Figure 15 Comparison of GloVe, LSA, Skip-gram and CBOW models

But we found that the performance of the word embedding did not improve on increasing the dimensionality. The skip-gram model showed the highest accuracy of around 57% for the dimensionality of 500 but on increasing the vector size beyond 500 the prediction accuracy decreased. We observed the similar patterns in all the other training methods except GloVe where the accuracy remained and improved slightly for the word vectors of dimensionality 1000. The reason for the decrease in accuracy for word vectors of higher dimensionality can be attributed to over-fitting as we used a smaller training corpus to evaluate the various models. Among the word embedding of different dimensions, the skip-gram model of word2vec has shown the state-of-the-art performance on the prediction of missing words in the sentences and is very well suited for semantic NLP tasks.

CHAPTER 5: CONCLUSION & FUTURE WORK

In this thesis we compared the performance of distributed word vectors on the task of predicting the missing word in the sentence. In particular, we assessed CBOW and skip-gram models of word2vec, GloVe and LSA by training them using 522 19^{th} century novels provided by Microsoft Research Sentence Completion Challenge data set. The skip-gram architecture of word2vec model trained using hierarchical softmax algorithm achieved 57% accuracy.

Among all methods, the performance of LSA model was the worst on Microsoft Research Sentence Completion test set. LSA is very sensitive to various hyper-parameters tuning [47]. In our experiments for LSA, we only varied the dimensions of word vectors. To improve the performance of LSA on sentence completion task, apart from TF-IDF normalization other techniques like square root normalization [49] can be used. Linear combination of the output of LSA and n-gram model can be performed as done in [19] to attain greater accuracy.

The skip-gram model of word2vec attained the highest accuracy of 57% on the sentence completion task for the word vectors of size 500. The word missing in the sentence completion questions occurred infrequently in the corpus and the skip-gram model tends to perform better on infrequent words.

The CBOW method also performed decently on the given task by answering around 53% questions correctly. As per [4, 12] skip-gram performs better than CBOW on less training data and our experiment results also support this analysis. This is because skip-gram

extracts more information from the available data as compared to CBOW. Also, CBOW model performs better on frequently occurring words. In terms of the training time, CBOW models are faster to train as compared to skip-gram. We can further improve the performance of CBOW models by incorporating more training data.

For word2vec models, we trained both models using two different optimization algorithms – hierarchical softmax and negative sampling. In both architectures, models trained using hierarchical softmax performed better than the one trained using negative sampling. As the missing words in the sentences are infrequent and hierarchical softmax generalize better on infrequent words. We also observed that for negative sampling the quality of lower dimensional vectors was better than that of higher-dimensional vectors.

The Glove model performed 50% on the sentence completion task. GloVe is a count-based model. Following the similar approach as in [44], GloVe model with noise-contrastive estimation can be trained to achieve better quality word embeddings.

To further optimize the performance of these methods detailed analysis of the set of questions that were correctly answered by the methods is required. One can leverage this information to effectively combine these methods to achieve greater accuracy in the prediction tasks. The quality of word embeddings learned using various methods can be further improved by training the models on a large amount of training data of the order of GB.

In recent years, neural network language models have been the focus of various semantic NLP tasks. To obtain high quality word embedding and eventually higher accuracy in sentence completion tasks neural network language model like recurrent neural network language model can be used. The main issue with NNLM is that they are slow and

computationally expensive to train. One can train them using noise-contrastive estimation to reduce the training time. Future analysis could also incorporate syntactic dependencies of the sentence like Part-of-Speech tags and syntax tree of the sentence to determine how well the word fit in the sentence.

It would be interesting to see how the models trained using deep learning neural network like Recurrent Neural Network and Feed-forward Convolution Neural Networks would perform on the sentence completion task.

APPENDIX A: CODE DESCRIPTION

The package contains three projects – GloVe, lsa and word2vec.

A.1 GloVe Package

The directory structure of GloVe is as follow

A.1.1 data_preprocessing.py

The script data_preprocessing.py processes the training corpus raw text and test data. It converts text to lowercase and remove all the meta data information.

A.1.2 Directory nlp_GloVe

The nlp_Glove folder contains the source code provided by [13]. It contains following files

1. **vocab_count**

It reads the tokens in the processed training corpus text file and constructs the unigram counts from a corpus. It creates the vocabulary and also threshold to keep the vocabulary size in a limit.

2. **cooccur**

This script constructs a word-word co-occurrence matrix from the training corpus and the vocabulary created using vocab_count. This also takes as input context size.

3. **shuffle**

This script performs the shuffling of the word-word co-occurrence matrix created by cooccur file. The output is a .bin file.

4. **Glove**

On running this tool, the GloVe model is trained using the co-occurrence matrix produced as an output of running the shuffle.c file. This class takes as an input the vector dimensions along with other parameters.

A.1.3 evaluate_MSR.py

This python file evaluates the various GloVe models by testing their performance on ted Microsoft Research Sentence Completion test set.

A.2 LSA Package

The directory structure consists of following two files

A.2.1 data_preprocessing.py

The data_preprocessing.py script processes the raw text in the training corpus and test data. It converts text to lowercase and removes all the meta data information.

A.2.1 lsa.py

On running this python file, a dictionary is created using the tokens in pre-processed training text. Then we train the LSA model using Gensim [49] package and test the model on the sentence completion task.

A.3 word2vec Package

The directory structure consists of following files:

A.3.1 data_preprocessing.py

The file data_preprocessing.py process the raw text in the training corpus and test data. It converts text to lowercase and removes all the metadata information.

A.3.2 Wor2Vec_CBOW_HS_MSR.py

This file trains the Continuous-Bag-of-Words model of word2vec using hierarchical softmax algorithm. We use Gensim's [49] word2vec implementation in Python to train the models This python class creates models using different context-size and vector-dimensions as discussed before.

A.3.3 Wor2Vec_CBOW_NS_MSR.py

This file trains the Continuous-Bag-of-Words model of word2vec using negative sampling algorithm. We use Gensim's word2vec implementation in Python to train the models This python class creates models using different context-size and vector-dimensions as discussed before.

A.3.4 Wor2Vec_SG_HS_MSR.py

Using this script, we train the Skip-gram model of word2vec using hierarchical softmax algorithm. We use Gensim's word2vec implementation in Python to train the models This python script creates models using different context-size and vector-dimensions as discussed before.

A.3.5 Wor2Vec_SG_NS_MSR.py

This file trains the Skip-gram model of word2vec using negative sampling algorithm. We use Gensim's word2vec implementation in Python to train the models This python class creates models using different context size and vector dimensions as discussed before.

A.3.6 evaluate_word2vec_cbow_hs.py

This file evaluates the performance of Continuous-Bag-of-Words model trained using hierarchical softmax algorithm on the sentence completion task.

A.3.7 evaluate_word2vec_cbow_ns.py

This file evaluates the performance of Continuous-Bag-of-Words model trained using negative sampling algorithm on the sentence completion task.

A.3.8 evaluate_word2vec_sg_hs.py

This file evaluates the performance of Continuous-Bag-of-Words model trained using hierarchical softmax algorithm on the sentence completion task.

A.3.9 evaluate_word2vec_sg_ns.py

This file evaluates the performance of Skip-gram model trained using negative sampling algorithm on the sentence completion task.

BIBLIOGRAPHY

1. Turing, A. M. (1950). Computing machinery and intelligence. Mind, 59(236), 433-460.
2. Chowdhury, G. G. (2003). Natural language processing. Annual review of information science and technology, 37(1), 51-89.
3. Kay, M., Norvig, P., & Gawron, M. (1992). Verbmobil: a translation system for face-to-face dialog. University of Chicago Press.
4. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
5. Peng, T. B. A. C. P., & Dean, X. F. J. O. J. (2007). Large Language Models in Machine Translation. EMNLP-CoNLL 2007, 858.
6. Teixeira, J., Sarmento, L., & Oliveira, E. (2011). A bootstrapping approach for training a NER with conditional random fields. In Progress in Artificial Intelligence (pp. 664-678). Springer Berlin Heidelberg.
7. Yarowsky, D. (1995, June). Unsupervised word sense disambiguation rivaling supervised methods. In Proceedings of the 33rd annual meeting on Association for Computational Linguistics (pp. 189-196). Association for Computational Linguistics.
8. Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here?
9. Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F., & Gauvain, J. L. (2006). Neural probabilistic language models. In Innovations in Machine Learning (pp. 137-186). Springer Berlin Heidelberg.

10. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391.
11. Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., & Harshman, R. (1988, May). Using latent semantic analysis to improve access to textual information. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 281-285). ACM.
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).
13. Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532-1543).
14. Lazaridou, A., Marelli, M., Zamparelli, R., & Baroni, M. (2013). Compositional-ly Derived Representations of Morphologically Complex Words in Distributional Semantics. In ACL (1) (pp. 1517-1526).
15. Bansal, M., Gimpel, K., & Livescu, K. (2014). Tailoring Continuous Word Representations for Dependency Parsing. In ACL (2) (pp. 809-815).
16. Guo, J., Che, W., Wang, H., & Liu, T. (2014). Revisiting Embedding Features for Simple Semi-Supervised Learning. In EMNLP (pp. 110-120).
17. Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the conference on empirical methods in natural language processing (EMNLP) (Vol. 1631, p. 1642).

18. Zweig, G., & Burges, C. J. (2011). The Microsoft Research sentence completion challenge. Technical Report MSR-TR-2011-129, Microsoft.
19. Zweig, G., Platt, J. C., Meek, C., Burges, C. J., Yessenalina, A., & Liu, Q. (2012, July). Computational approaches to sentence completion. InProceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1 (pp. 601-610). Association for Computational Linguistics.
20. Lee, K., & Lee, G. G. (2014, January). Sentence completion task using web-scale data. In Big Data and Smart Computing (BIGCOMP), 2014 International Conference on (pp. 173-176). IEEE.
21. Gubbins, J., & Vlachos, A. (2013). Dependency Language Models for Sentence Completion. In EMNLP (pp. 1405-1410).
22. Yuret, D. (2007, June). KU: Word sense disambiguation by substitution. InProceedings of the 4th International Workshop on Semantic Evaluations (pp. 207-213). Association for Computational Linguistics.
23. Giuliano, C., Gliozzo, A., & Strapparava, C. (2007, June). Fbk-irst: Lexical substitution task exploiting domain and syntagmatic coherence. InProceedings of the 4th International Workshop on Semantic Evaluations (pp. 145-148). Association for Computational Linguistics.
24. Mohammad, S., Dorr, B., & Hirst, G. (2008, October). Computing word-pair antonymy. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (pp. 982-991). Association for Computational Linguistics.
25. Turney, P. D., & Littman, M. L. (2005). Corpus-based learning of analogies and semantic relations. Machine Learning, 60(1-3), 251-278.

26. Newell, A. F., Arnott, J. L., & Swiffin, A. L. (1985). Development of Predictive Word Processing Aids for the Physically Handicapped: Summary Report to the Spastics Society. University of Dundee.
27. Nantais, T., Shein, F., & Johansson, M. (2001). Efficacy of the word prediction algorithm in WordQ. In Proceedings of the 24th Annual Conference on Technology and Disability.
28. Fazly, A., & Hirst, G. (2003, April). Testing the efficacy of part-of-speech information in word completion. In Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods (pp. 9-16). Association for Computational Linguistics.
29. Carlberger, A., Carlberger, J., Magnuson, T., Hunnicutt, S., Palazuelos-Cagigas, S. E., & Navarro, S. A. (1997, July). Profet, a new generation of word prediction: An evaluation study. In Proceedings, ACL Workshop on Natural language processing for communication aids (pp. 23-28).
30. Matiasek, J., Baroni, M., & Trost, H. (2002). FASTY—A multi-lingual approach to text prediction. In Computers helping people with special needs (pp. 243-250). Springer Berlin Heidelberg.
31. Speech and Language Processing. Daniel Jurafsky & James H. Martin.
32. Shannon, C. E. (2001). A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review, 5(1), 3-55.
33. Shannon, C. E. (1951). Prediction and entropy of printed English. Bell System Technical Journal, 30, 50–64
34. Teixeira, J., Sarmento, L., & Oliveira, E. (2011). A bootstrapping approach for training a NER with conditional random fields. In Progress in Artificial Intelligence (pp. 664-678). Springer Berlin Heidelberg.

35. Chen, S. F., & Goodman, J. (1996, June). An empirical study of smoothing techniques for language modeling. In Proceedings of the 34th annual meeting on Association for Computational Linguistics (pp. 310-318). Association for Computational Linguistics.
36. Laplace, P. S. (2012). Pierre-Simon Laplace Philosophical Essay on Probabilities: Translated from the fifth French edition of 1825 With Notes by the Translator (Vol. 13). Springer Science & Business Media.
37. Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. Biometrika, 40(3-4), 237-264.
38. Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. Acoustics, Speech and Signal Processing, IEEE Transactions on, 35(3), 400-401.
39. Ney, H., & Essen, U. (1991, April). On smoothing techniques for bigram-based natural language modelling. In Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on (pp. 825-828). IEEE.
40. Brants, T., Popat, A. C., Xu, P., Och, F. J., & Dean, J. (2007). Large language models in machine translation. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.
41. Firth, J. R. (1957). Papers in linguistics, 1934-1951. Oxford University Press.
42. Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11), 613-620.
43. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391.

44. Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. Discourse processes, 25(2-3), 259-284.
45. Rong, X. (2014). word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.
46. Morin, F., & Bengio, Y. (2005, January). Hierarchical Probabilistic Neural Network Language Model. In Aistats (Vol. 5, pp. 246-252).
47. Lifchitz, A., Jhean-Larose, S., & Denhière, G. (2009). Effect of tuned parameters on an LSA multiple choice questions answering model. Behavior research methods, 41(4), 1201-1209.
48. Spiccia, C., Augello, A., Pilato, G., & Vassallo, G. (2015, February). A word prediction methodology for automatic sentence completion. In Semantic Computing (ICSC), 2015 IEEE International Conference on (pp. 240-243). IEEE.
49. Sojka, P. (2010). Software framework for topic modelling with large corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.
50. Gilleland, M. (2009). Levenshtein distance, in three flavors. Merriam Park Software: http://www. merriampark. com/ld. htm.