GEORGETOWN UNIVERSITY
The Graduate School of Arts & Sciences

Word2Vec: A Skip-Gram Model Implementation Using TensorFlow

ANLY 561 PROJECT REPORT

Weile Chen, Shaoyu Feng, Chelsea Wang,
Yunjia Zeng, Mengtong Zhang

Contents

Introduction

Vector representation of text/words is a useful technique to overcome the sparsity issues of text data in Natural Language Processing (NLP). There are two types of vector space models, count-based methods and predictive methods. Introduced by Mikolov et al. [1], Word2vec model is a computationally-efficient predictive model for training word embeddings from text data, and it has gained numerous attentions in recent years. Word2vec takes a large corpus of text as input and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

There are two kinds of word2vec models, namely the Continuous Bags-of-Words model (CBOW) and the Skip-Gram model. The Skip-gram model architecture usually tries to achieve the reverse of what the CBOW model does. It tries to predict the source context words (surrounding words) given a target word (the center word). [9]

This paper will focus on the implementation of the skip-gram model using TensorFlow and the use of both Stochastic Gradient Descent (SGD) [8] and Adaptive Moment Estimation (Adam) optimizer to optimize the training process.

The report will cover three parts of the analysis. Part 2 focuses on dataset being used, exploratory data analysis (EDA), data preprocessing, and the optimization problem. Part 3 illustrates the construction of training data, the neural network model formation, negative sampling, SGD optimizer, and Adam optimizer. Part 4 discusses the performance comparison of training SGD and Adam.

Data Set, Data Pre-processing and the optimization problem

2.1 About the Data

The data used in this project ("text8") is 100 MB of text from "fil9", which is the cleaned text from Wikipedia text "enwik9". This clean version was prepared with the goal of retaining only text. It could be viewed on the Wikipedia web page and read by a human. The file only keeps regular article text and image captions, and tables or links to foreign language versions, footnotes, and markup were removed. Hypertext links were converted to ordinary text, and numbers were spelled out. Also, all upper-case letters were converted to lower cases, and at last all sequences of characters not in the range of a-z were converted to a single space.[5]

2.2 EDA Analysis

The data is a pure text dataset. Figure 2-2-1 shows pie chart statistics of characters. Moreover, the Word Cloud of the text8 represents the most frequency word. As a result, the most frequent word category is number. Therefore, we should expect the word embedding results to have a dense cluster consists of numbers in training results.

Figure 2-2-1: Pie Chart of character count statistics

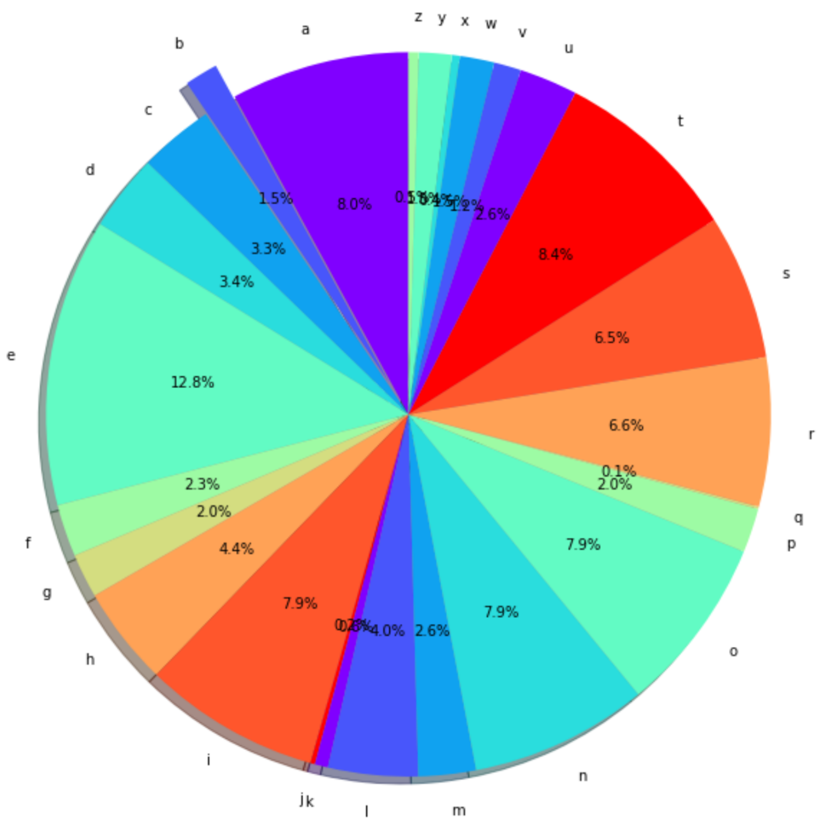
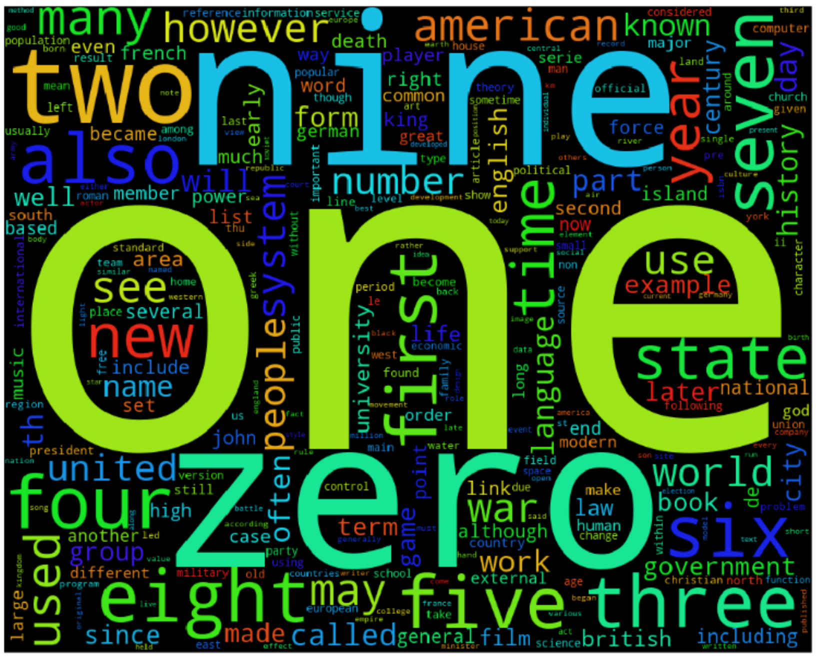


Figure 2-2-2: Word Cloud



2.3 Data Preprocessing

Text8 data is the cleaned Wikipedia dump from scraped pages without meaningless contents such as HTML tags. However, it still contains punctuations that need to be removed. We perform pre-processing on the text8 data to clean all punctuations and low-frequency words for better results. The pre-processing includes four parts:

- Replace the punctuations with tags
- Split the text
- Construct corpus
- Construct vocabulary embedding lookup table

The total number of vocabulary is 16680599, with 63641 unique words.

Word2Vec Model Construction and Optimization Problem

3.1 Model Construction and the Optimizer

The basic idea of Word2Vec is to train a simple neural network with a single hidden layer. Given a specific word in the middle of a sentence (the input word), we look at the words nearby and pick one at random. The network tells us the probability for every word in our vocabulary of being the "nearby word" that we chose, which is represented by weight matrix in the hidden layer.

The following parts introduce the details of the input layer, hidden layer and output layer for the skip-gram model. Also, the sampling method and two kinds of optimizer algorithm used in this model will be covered.

3.1.1 Input layer to hidden layer

It is impossible to feed a word just as a text string into a neural network. Therefore, for the input layer, we need to represent words as one-hot vectors. For example, for the word "the" in the text data, the one-hot vector of "the" will have M components (one for every word in vocabulary). Also, "1" will be put into the position corresponding to the word "the", and 0s are put in all of the other positions.

For the hidden layer, we define a weight matrix W with N rows (one for every word in the vocabulary) and M columns (one for every hidden neuron). The i^{th} row of weight matrix is the word vector of the i^{th} word in the whole vocabulary list. The goal of this model is to learn this hidden layer weight matrix and we'll toss the output layer when the training process is finished.

Suppose the word 'the' is the i^{th} word in our vocabulary, and the weight matrix is W ($W = \{w_{i,j}\}, i = 1,2, \dots, N, j = 1,2, \dots, M$), then the input one-hot vector of "the" would be $\boldsymbol{e}^{(i)}$($\boldsymbol{e}^{(i)} \epsilon\ \mathbb{R}^N$), and the word vector of "the" would be $\boldsymbol{w_i} = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \dots \\ w_{iM} \end{bmatrix}$

The one-hot vector is very sparse because almost all of the entries are zeros. In real world implementation, M and N are typically large numbers. If we multiply a $1 * N$ one-hot vector by a $M * N$ matrix, it will be more effective if selecting the matrix row corresponding to the "1". This step can be done with 'embedding_lookup' function in Tensorflow.

3.1.2 Output layer

The output layer is a SoftMax regression classifier. Each output neuron (one per word in our vocabulary) will produce an output between 0 and 1, and the sum of all these output values will add up to 1. Suppose that we are trying to predict the next word of the i^{th} word in our vocabulary, then the regression model can be written as:

$$\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}$$

Where $\boldsymbol{x} = \boldsymbol{e}^{(i)}\left(\boldsymbol{e}^{(i)} \epsilon\ \mathbb{R}^N\right), W \epsilon M_{N,M}, \boldsymbol{b} \epsilon\ \mathbb{R}^N$

The prediction of $\boldsymbol{y}$ would be

$$\widehat{\boldsymbol{y}} = softmax\left(\widehat{W}\boldsymbol{x} + \boldsymbol{b}\right) = \begin{bmatrix} \frac{exp\left(\boldsymbol{w}^{(1)}\boldsymbol{x} + b_1\right)}{\sum_{i=1}^{N} exp\left(\boldsymbol{w}^{(i)}\boldsymbol{x} + b_i\right)} \\ \frac{exp\left(\boldsymbol{w}^{(2)}\boldsymbol{x} + b_2\right)}{\sum_{i=1}^{N} exp\left(\boldsymbol{w}^{(i)}\boldsymbol{x} + b_i\right)} \\ \vdots \\ \frac{exp\left(\boldsymbol{w}^{(N)}\boldsymbol{x} + b_N\right)}{\sum_{i=1}^{N} exp\left(\boldsymbol{w}^{(i)}\boldsymbol{x} + b_i\right)} \end{bmatrix}$$

where $\boldsymbol{w}^{(i)}$ is the i^{th} row of $\widehat{W}, i = 1,2, \dots, N$

All entries of $\widehat{\boldsymbol{y}}$ can be interpreted as probabilities,

$$\widehat{y_j} = P(the\ next\ word\ of\ the\ i^{th}\ word\ is\ the\ j^{th}\ word)$$

Cross-entropy is used to generate the loss function:

$$H(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{N} -p_i \log q_i$$

for $\boldsymbol{p}, \boldsymbol{q}\ \epsilon\ \mathbb{R}^N$.

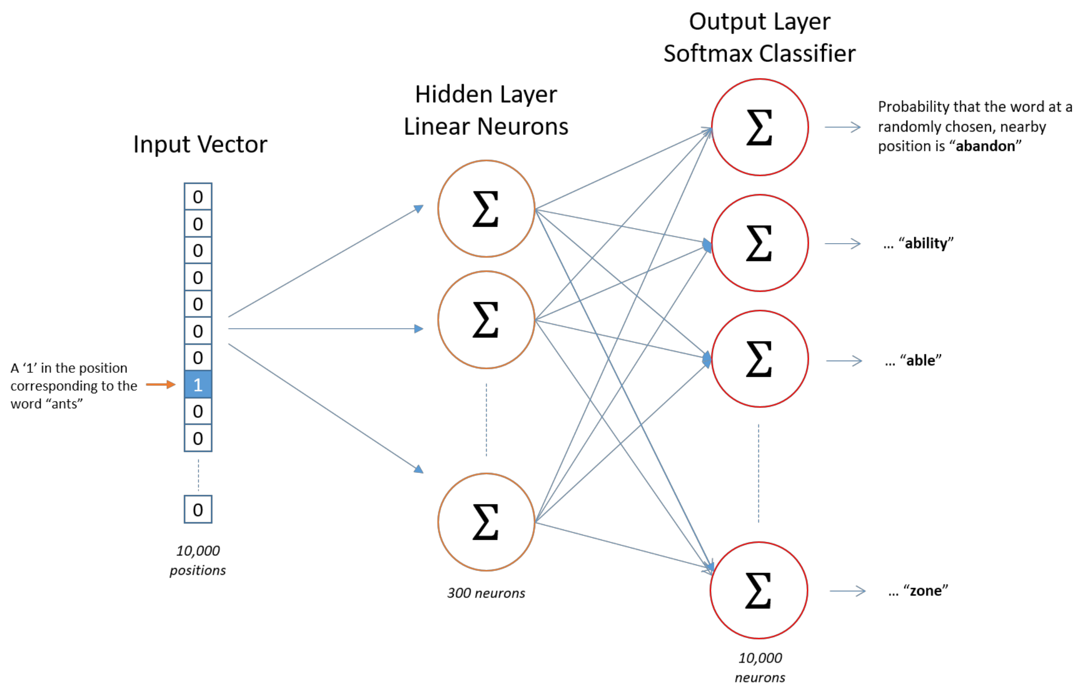
Therefore, the objective function is:

$$f = \frac{1}{K}\sum_{I=1}^{N} H\left(\boldsymbol{y}^{(j)}, softmax(W\boldsymbol{x}^{(j)} + \boldsymbol{b}\right)$$

where $j = 1,2, \dots, K, (\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})$ is the j^{th} pair of training data, and K is the number of word pairs in the training data. [7]

The basic idea of this model can be shown in figure 3-1-1.

Figure 3-1-2: Basic frame work of the skip-gram model



3.1.3 Negative Sampling

For each step of training the neural network, we take a training example and adjust all of the neuron weights slightly so that it predicts training data more accurately. In other words, each training sample will adjust all of the weights in the neural network. As discussed above, the size of word vocabulary is typically very large. Thus it would be slow if updating all entries of the weight matrix for every sample. Negative sampling solves the issue by only modifying a small proportion of the weight matrix for each training sample.

Suppose to train the word pair ('dog', 'barked'), and 'dog' is the i^{th} word in our vocabulary, 'barked' is the j^{th} word in our vocabulary. That is, the output neuron will be $\boldsymbol{e}^{(j)}\left(\ \boldsymbol{e}^{(j)}\epsilon\ \mathbb{R}^N\right)$. We call the j^{th} word 'positive word' and the words other than the j^{th} word 'negative words'. And in implementation, we update the weights of a small number of randomly selected 'negative' words. That is, we only update for $\boldsymbol{w_j}$ and some other rows of W[6]. Negative sampling can be implemented with 'tf.nn.sampled_softmax_loss' in TensorFlow [10].

3.1.4 Stochastic Gradient Descent

In this paper, stochastic gradient descent optimizer is one option used to train the model. Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization.

Instead of computing the gradient of exactly, each iteration estimates this gradient on the basis of a single randomly picked examples:

$$\theta_{t+1} = \theta_t - \eta \Delta\theta^{(t)}$$

$$\Delta\theta^{(t)} = \frac{1}{M}\sum_{J=1}^{M} \nabla_\theta f(\theta; x_{i^{(j)}}, y_{i^{(j)}})$$

where f is the objective function, η is the learning rate, and $Q = \{i^{(1)}, i^{(2)}, i^{(3)}, \ldots, i^{(M)}\} \subset \{1,2,3, \ldots, N\}$ is a random subset of the indices from 1 to N.

'tf.keras.optimizers.SGD' can be used to implement stochastic gradient descent optimizer, and the default value of learning rate η is set to be 0.01[10].

3.1.5 Adam Optimizer

Another optimizer can be used to minimize the SoftMax loss is Adaptive Moment Estimation (Adam)[2], which is also a Gradient descent optimization algorithm.

In addition to store an exponentially decaying average of past squared gradients v_t, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. We compute the decaying averages of past and past squared gradients m_t and v_t, respectively as follows:

$$g_t = \nabla_\theta f(\theta_{t-1})$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

where f is the objective function, m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. As m_t and v_t are initialized as vectors of 0's, they are biased towards zero during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

We counteract these biases by computing bias-corrected first and second moment estimates:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

The Adam updated rule is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\widehat{v_t}^{1/2} + \varepsilon} \widehat{m_t}$$

The default values of hyper-parameters are: 0.9 for β_1, 0.999 for β_2, 0.001 for the learning rate η and 10^{-8} for ε.

The Adam algorithm can be implemented with 'tf.train.AdamOptimizer' in TensorFlow[10].

3.2 Training Data Construction

The Skip-Gram model predicts the previous and next words based on one input word, thus the input word will correspond to multiple nearby words.

However, stop words, such as “the” and “a”, will make some noise to our model, and slow down the training speed. Since stop words generally reduce the importance of other meaningful words, we will remove them before constructing the training batches.

3.2.1 Batch Construction
In order to make a prediction of nearby words for each input word, we need to follow two steps:

Step One: Find the previous and next words of each input word.

We define the "get_targets" function to receive the index of an input word. It gets the index of "nearby words", which can be used to construct batches for training. Note that every input word corresponds to several output words. For example, for the sentence "The quick brown fox jumps over lazy dog", the function outputs nearby words [quick, brown, jumps, over] when receives the input word "fox" with window_size=2.

Step Two: Construct batch for training.

By defining a function get_batches, we get a set of samples for training. This function yields several samples that consist of the input words and nearby words from the get_targets function based on the batch size, where batch size is the number of words other than the input word. From the previous example, the input word is "fox" with window_size=2, and nearby words are [quick, brown, jumps, over]. Here we have four training samples when we set batch_size = 1, which are [fox, quick], [fox, brown], [fox. jumps], and [fox, over] respectively.

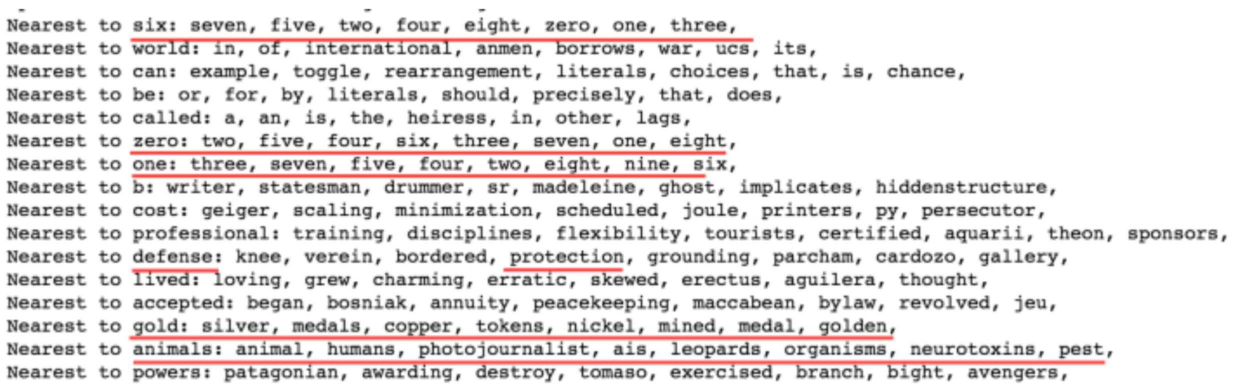
Model Validation and Discussion

Two different optimization methods are applied to minimize the negative sampling loss in this project, the Adam Optimizer and the SGD Optimizer. The training time for Adam Optimizer was 13065 seconds while the SGD Optimizer spent 3163.4 seconds on training, which was way faster than Adam Optimizer.

Unlike typical classification or regression models where the data is properly labeled, input data for word2vec model is not labeled. Therefore, training loss does not effectively reflect the effectiveness of the word embedding in the neural network model. In addition, due to the negative sampling method, the training loss is underestimated. There are generally two categories in evaluating the word embedding models: extrinsic and intrinsic evaluation [3]. Extrinsic evaluation uses word embeddings as input features to a downstream task and measure changes in performance metrics specific to that task. Intrinsic evaluations directly test for syntactic or semantic relationships between words. These tasks typically involve a pre-selected set of query terms and semantically related target words, defined as query inventory. In this paper, we use a simplified intrinsic evaluation method to measure the performance of the word2vec.

In the implementation, several words are chosen randomly as validation examples. During the training stage, each word vector will be calculated and normalized. The similarity of those validation examples relative to other words will be calculated based on cosine similarity. Top K words will be chosen to indicate the similar words associated with the validation examples. The following figure shows the results of the similar words after several iterations of training. As shown in the figure 4-1, the Adam model shows good learning ability in some word categories like numbers, metals.

Figure 4-1: Result of the similar words after several iterations of training



```
Nearest to six: seven, five, two, four, eight, zero, one, three,
Nearest to world: in, of, international, anmen, borrows, war, ucs, its,
Nearest to can: example, toggle, rearrangement, literals, choices, that, is, chance,
Nearest to be: or, for, by, literals, should, precisely, that, does,
Nearest to called: a, an, is, the, heiress, in, other, lags,
Nearest to zero: two, five, four, six, three, seven, one, eight,
Nearest to one: three, seven, five, four, two, eight, nine, six,
Nearest to b: writer, statesman, drummer, sr, madeleine, ghost, implicates, hiddenstructure,
Nearest to cost: geiger, scaling, minimization, scheduled, joule, printers, py, persecutor,
Nearest to professional: training, disciplines, flexibility, tourists, certified, aquarii, theon, sponsors,
Nearest to defense: knee, verein, bordered, protection, grounding, parcham, cardozo, gallery,
Nearest to lived: loving, grew, charming, erratic, skewed, erectus, aguilera, thought,
Nearest to accepted: began, bosniak, annuity, peacekeeping, maccabean, bylaw, revolved, jeu,
Nearest to gold: silver, medals, copper, tokens, nickel, mined, medal, golden,
Nearest to animals: animal, humans, photojournalist, ais, leopards, organisms, neurotoxins, pest,
Nearest to powers: patagonian, awarding, destroy, tomaso, exercised, branch, bight, avengers,
```

Another way to visualize the distance between words vectors is through t-distantributed Stochastic Neighbour Embedding (TSNE). TSNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data [4]. Following figure 4-2 shows word vectors in 2D space based on distance between each other after the training. Based on the zoomed in view of following visualization, the model achieves reasonably good results in learning the word embedding.

Figure 4-2: TSNE plots for Adam Optimizer

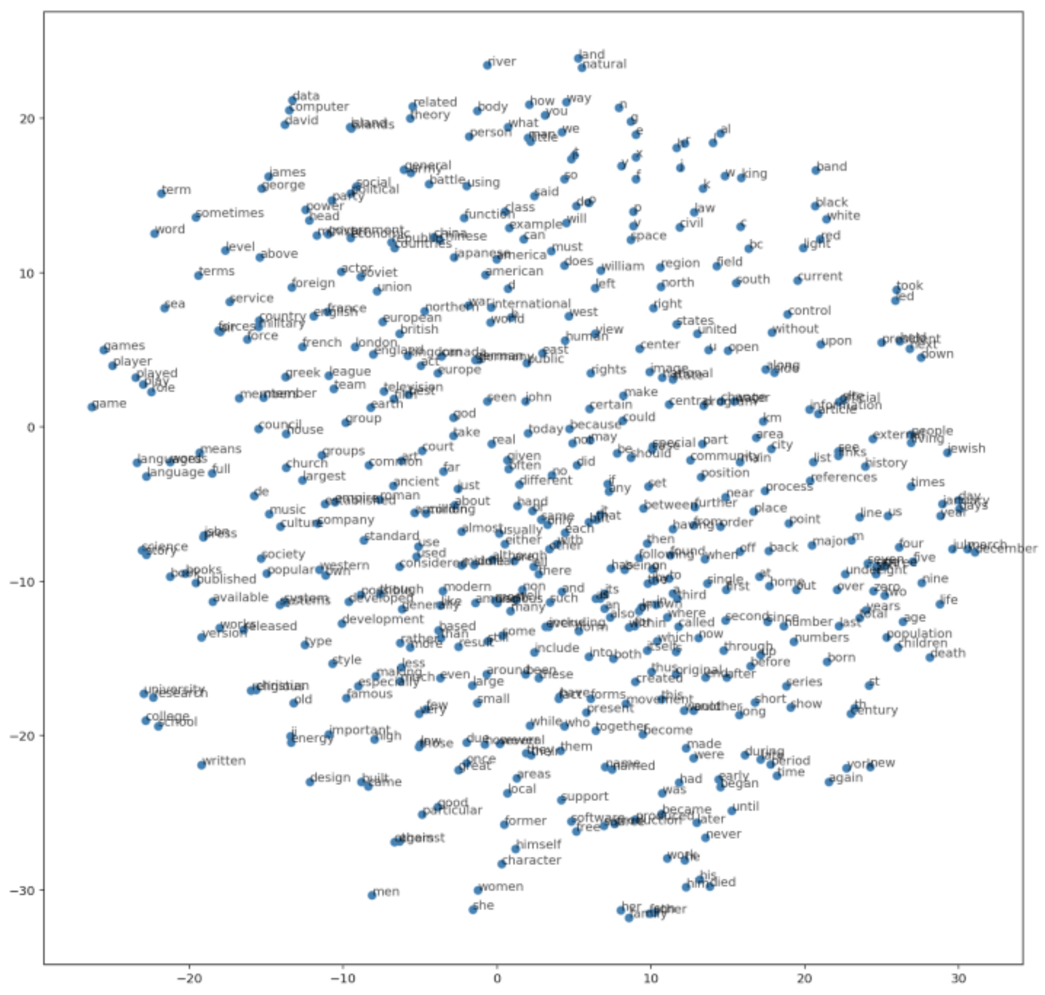
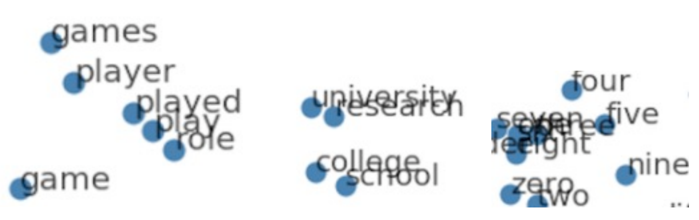


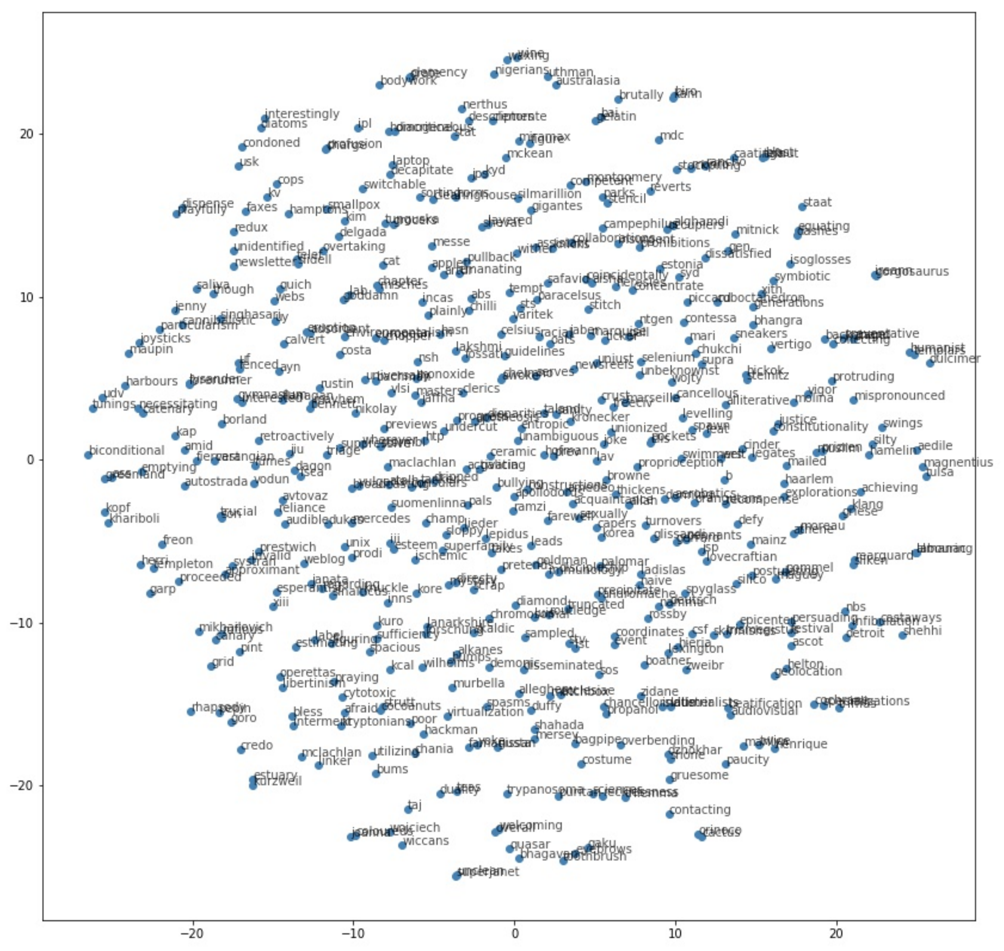
Figure 4-3: zoomed in graph



The two TSNE plots of results also show that Adam Optimizer works better than SGD, since the word clusters are much more meaningful. For Adam Optimizer, there are some clear and meaningful clusters such as numbers, games, universities, and languages that illustrated in model validation section. Surprisingly, it is very hard to find even one meaningful cluster in the training results of SGD. One explanation might be Adam Optimizer only updates parameters that are actually employed. It can make a big difference in a sparse setting like a language model, since Adam Optimizer applies no updates to rare words until they appear, at which time they get a big update. More common words are updated more frequently.

One problem for SGD is that it is hard to find an optimal learning rate for training. Normally, it is better to set up a range of learning rate and compare the results to find the optimal is required, but due to the limited computational power of our machine a fixed learning rate was set. Sometimes SGD may converge to local optimal only, and trapped at saddle points if without proper settings of initialization and step size. Compare to SGD, Adam has more adaptive learning rate to each step of parameter updating.

Figure 4-4: TSNE plots for SGD



Conclusion

To sum up, a skip-gram model is implemented using TensorFlow framework. The model has been trained using both SGD and Adam optimizer. The performance of the two optimizer has been discussed and we observe more superior performance of Adam Optimizer over SGD Optimizer. However, there is no quantitative metric discussed in this paper in determining the effectiveness of word embedding. And the hyper-parameter of the model training is not well studied. The future work of this paper will focus on the hyper-parameter optimization of the Neural Network Model, research on the more quantitative approach to evaluate the effectiveness of word2vec model, and the application of word2vec model in NLP and text analytics.

References

[1] Mikolov, Tomas, et al. “Distributed Representations of Words and Phrases and Their Compositionality.” papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[2] Kingma, Diederik P., and Jimmy Lei Ba. “Adam: A METHOD FOR STOCHASTIC OPTIMIZATION.” ICLR 2015, 30 Jan. 2017, pp. 1–15., arxiv.org/pdf/1412.6980.pdf.

[3] Schnabel, Tobias, et al. “Evaluation Methods for Unsupervised Word Embeddings.” The 2015 Conference on Empirical Methods in Natural Language Processing, 17 Sept. 2015, pp. 298–307., www.aclweb.org/anthology/D15-1036.

[4] Keim, Daniel A. “Information Visualization and Visual Data Mining.” IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, Jan. 2002, pp. 100–107., nm.merz-akademie.de/~jasmin.sipahi/drittes/images/Keim2002.pdf.

[5] Perer, Adam, and Ben Shneiderman. “Integrating Statistics and Visualization: Case Studies of Gaining Clarity during Exploratory Data Analysis.” CHI 2008 Proceedings · Visual Synthesis, 5 Apr. 2008, pp. 265–274., cgis.cs.umd.edu/~ben/papers/Perer2008Integrating.pdf.

[6] Fonarev, Alexander, et al. “RIEMANNIAN OPTIMIZATION FOR SKIP-GRAM NEGATIVE SAMPLING.” ICLR 2017, pdfs.semanticscholar.org/9997/a2326c1a1d439febcb3d7f0edd139a7ae24a.pdf.

[7] Rong, Xin. “word2vec Parameter Learning Explained.” 5 June 2016.

[8] Bottou, Leon. “Large-Scale Machine Learning with Stochastic Gradient Descent.” NEC Labs America, leon.bottou.org/publications/pdf/compstat-2010.pdf.

[9] Ling, Wang, et al. “Two/Too Simple Adaptations of Word2Vec for Syntax Problems.” The 2015 Annual Conference of the North American Chapter of the ACL, 31 May 2015, pp. 1299–1304., www.aclweb.org/anthology/N15-1142.

[10] Abadi, Martin, et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.” Preliminary White Paper, 9 Nov. 2015, download.tensorflow.org/paper/whitepaper2015.pdf.