Natural Language Understanding: Assignment 1 Report

Kapil Pathak
CSA, IISc Bangalore
kapilpathak@iisc.ac.in

Abstract

This report consists of a summary of the assignment 1 in which a python Word2Vec model is implemented from scratch. After the training the model, final embedding were tested on Simlex-999 word similarity task. Later on, these embedding are tested on Google analogy test set to get analogy score for each embedding. Also few efforts have been showed to get any biase present in the embedding, mainly gender bias.

1 Introduction

Word embeddings capture the 'meaning' of a word using a d-dimensional vector in continuous space. The 'meaning' of a word, according to Firth's Hypothesis [1], can be captured by the context in which the word occurs. This implies a very simple map from words to vectors. Construct a matrix M where the rows are indexed by words and the columns are indexed by context (say, an n-tuple of words). The entry (w, c) stores the number of times w appears in context c. The vector corresponding to w is the row of the matrix M corresponding to w. This approach tends to produce high-dimensional vectors because a dictionary of W words can furnish up to $\binom{W}{n}$ n-tuples of context.

2 Word2Vec Model

In (Mikolov et al., 2013), the paper talks about how a new proposed language model overcomes the issue of count-based methods and establishes new prediction based approach where words are predicted given their context word around them. This paper talks about two predictive strategies namely 'skip-gram' and 'CBOW' for the capturing the meaning of the words in the form word embedding. In skip gram model, in every pre-defined window size, the center word predicts the neighbouring word in that window. In case of CBOW model, the context words in the window around the center word help to predict the center word.

In the given assignment, a skip gram version of Word2Vec has been implemented in python without using any machine learning library. In the skip gram model, given a sequence of training words $w_1, w_2, ..., w_T$, we need to maximize log-probability given by

$$\frac{1}{T}\sum_{1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

Here the larger c denotes the larger context window size. At the end, a softmax function is applied to get the probable word as,

$$p(w_O|w_I) = \frac{exp({w'_O}^T w_I)}{\sum_w^W exp({w'_w}^T w_I)}$$

But denominator in the equation is computationally very expensive. Hence, we simplify above operation either by using hierarchical softmax function or negative sampling. In the current implementation, a negative sampling is performed.

2.1 Subsampling

Here Reuters dataset is split into training and testing set. A removal of stop-words is avoided and to counter the imbalance of rare and frequently occurred, subsampling is performed in which each word from the corpus is discarded with a probability given by

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where $f(w_i)$ is unigram count frequency. Here t is kept as 0.00001 as given in the paper.

[1] 'a word is known by the company it keeps'

2.2 Negative Sampling

In the negative sampling, for a current center and context word pair, k corrupted context word are sampled such that these k words never occurred in the context of given center word. These can be done by maintaining a dictionary of center word and a list seen context words. Out of remaining unseen words, k negative samples are sampled according to their unigram counts to the power 0.75 as given in the paper. The process of making a dictionary of (center word, list(context words)) slows down the process slightly but makes sure that the negative context words samples from the corpus actually corrupt the (center word, context word) pair.

2.3 Loss Function

The current implementation consists of a simple gradient ascent algorithm where objective is to maximize following likelihood function.

$$log(\sigma({v'_{w_O}}^T v_{w_I})) + \sum_{k=1}^{K} {}_{w_i \sim P_n(w)} [log\sigma(-{v'_{w_k}}^T v_{w_I})]$$

Here the gradients are calculated and hand-coded for each of center word, true context word, and all negative context word sampled. The number of negative samples to be selected is a hyper-parameter which is tuned for 10,13,15 and 20. At the end of each epoch, an average loss stored to check the convergence of the algorithm.

3 Experimentation

In the current implementation, the experiments have been performed and performance of each embedding generated is tested on SimLex-999 similarity words task. SimLex-999 is a standard where similarity between two words have been captured independent of their relatedness or dependency. The co-occurrence counts often capture relatedness of two words rather than their similarity. From a given source, the first two columns represent two words which are either synonyms or antonyms and their similarity is represented with the scale of 0 to 10. The synonyms are represented with the score closer to 10 while antonyms are represented with score closer to 0.

To check the performance of the embedding, we first find how many pairs from SimLex-999 are actually present in the given dataset. This is done by stemming each of the words in SimLex-999 and also from a given dataset. Afterwords, the spearman correlation between the cosine similarity vector of matched words and that of SimLex-999 is computed. Higher the correlation score, higher is the performance of the trained word embedding over word similarity task.

3.1 Word Similarity score vs word embedding dimension

In this setting, the number of epochs are 10 and each epoch takes from 16 min to 20 min depending upon the word embedding dimension. The learning rate is kept as 0.001 and number of negative samples taken were 10 with context window size 2.

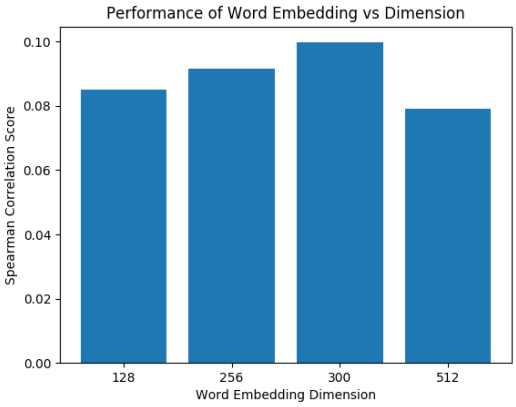

Figure 1: Word Similarity Score for various word embedding dimension

3.2 Word Similarity score vs Number of Negative Samples

In this setting, the number of epochs are 10 and each epoch takes around 16-17 min. The learning rate is kept as 0.001 and word embedding dimension as 300 with context window size 2.

3.3 Word Similarity score vs Half width of Context Window

In this setting, the number of epochs are 10 and each epoch takes around 16-31 min as context window increases, the number of center word and context word pairs increases. The learning rate is kept as 0.001 and word embedding dimension as 300 with negative samples 10.

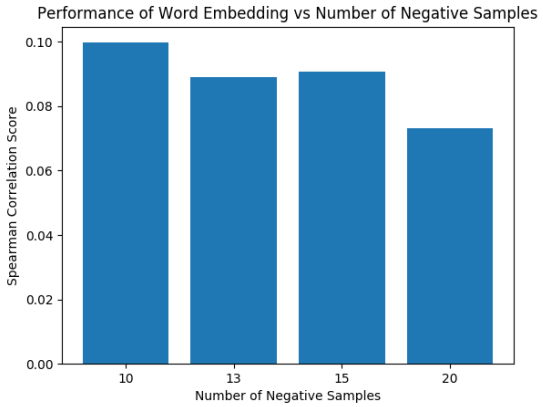

Figure 2: Word Similarity Score for Number of Negative Samples

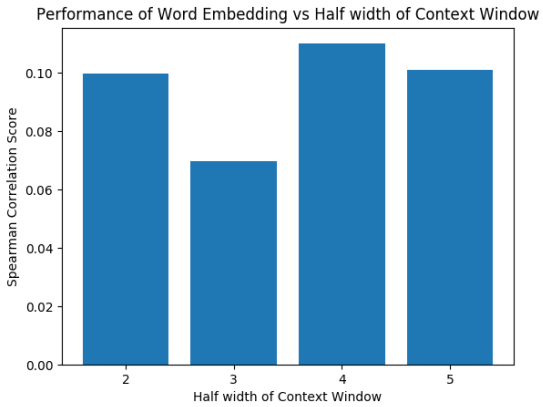

Figure 3: Word Similarity Score for different half context window size

3.4 Loss vs Number of epochs for various settings

In one setting, the number of epochs are 10, context window half width was 2. The learning rate is kept as 0.001 and word embedding dimension as 300 with negative samples 10. Final loss was -3.9629415022500303 at the end of 10 epochs. Here the loss is **increasing** as we are **maximizing** the log likelihood and applying **gradient ascent.**

In another setting, the number of epochs are 10, context window half width was 4. The learning rate is kept as 0.001 and word embedding dimension as 300 with negative samples 10. Final loss was -1.7196471363903516 at the end of 10 epochs. Here the loss is **increasing** as we are **maximizing** the log likelihood and applying **gradient ascent.**

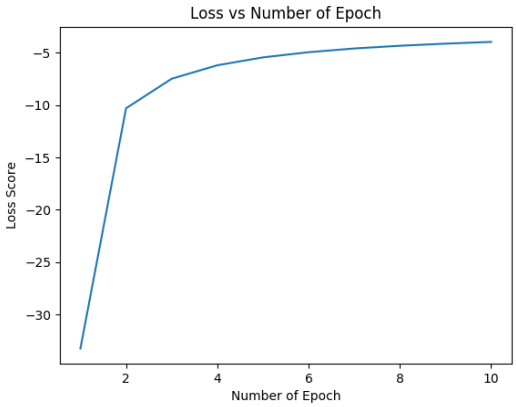

Figure 4: Loss vs number of Epochs for context window having half width 2

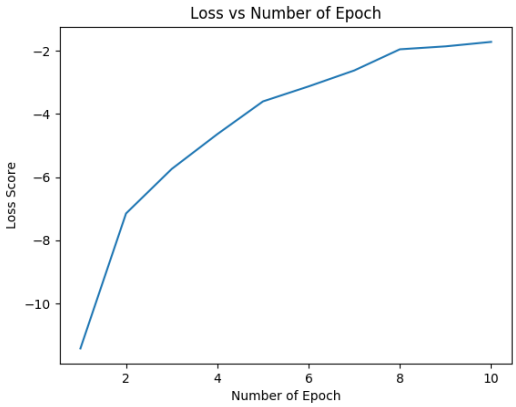

Figure 5: Loss vs number of Epochs for context window having half width 4

After above experimentation, I performed word analogical reasoning task given by Google. `https://aclweb.org/aclwiki/Google_analogy_test_set_(State_of_the_art)`

In the **word analogical reasoning task**, each line in the dataset consists of word pairs such as $Iraq : Baghdad :: India : Delhi$. Here each pair of word has a specific relatedness. We need to capture the closest vector in the trained word embedding to $Iraq - Baghdad + India$ (called it as

Table 1: Word Similarity Scores and Word Analogy Scores for various dimensional embedding

DIMENSION	SIMILARITY SCORE	ANALOGY SCORE
128	0.08519 (8.51%)	122/7807
256	0.09173 (9.17%)	89/7807
300	0.09976 (9.97%)	90/7807

Table 2: Word Similarity Scores and Word Analogy Scores for various Number of Negative Samples

NEGATIVE SAMPLES	SIMILARITY SCORE	ANALOGY SCORE
10	0.09976 (9.97%)	90/7807
13	0.08892 (8.89%)	89/7807
15	0.09063 (9.06%)	79/7807
20	0.07306 (7.30%)	58/7807

analogy vector)and check if the correct word vector for word $Delhi$ appears in **top 10** closest word vectors. Here we will filter out all those word analogy pairs in which one or more words haven't appeared in the training dataset of word embedding. The closeness of vector pairs are checked with cosine similarity.

For each of datapoint in all graphs, we can check its word similarity score and word analogy score from the tables given below. Here the entry in Analogy Score in the first row shows for 122 pairs, correct word was found to be in top 10 closest words to corresponding word analogy vector mentioned above.

3.5 Experimentation with respect to word embedding dimensions

For this section please refer to table 1.

3.6 Experimentation with respect to Number of Negative Samples

For this section, please refer to table 2.

3.7 Experimentation with respect to Half Width Context Window

For this section, please refer to table 3.

Here we can see that word embedding utilizing context window half width equal to 4 and 5 are giving better performance in both similarity as well as analogical task.

Table 3: Word Similarity Scores and Word Analogy Scores for context window of varied half width

HALF WIDTH	SIMILARITY SCORE	ANALOGY SCORE
2	0.09976 (9.97%)	90/7807
3	0.06988 (6.98%)	96/7807
4	0.11000 (11%)	126/7807
5	0.10090 (10.0%)	146/7807

Acknowledgments

All codes, word embeddings and report have been uploaded to `https://github.com/Kapil-Pathak/E1-246-Natural-Language-Understanding-2019--git` Also for the assignment following document is referred `https://cs224d.stanford.edu/lecture_notes/notes1.pdf`

References

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.