Esercise 1 Report

Natural Language Processing

Aprajita ARORA, Yago BARDI, Caterina CONZ, Fatmanur SEVER

ACM Reference Format:
Aprajita ARORA, Yago BARDI, Caterina CONZ, Fatmanur SEVER. 2022. Esercise 1 Report: Natural Language Processing. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

This report discusses the implementation of Skip-gram algorithm along with negative sampling as the optimization technique in detail. Other optimization techniques have also been briefly discussed. Hyper-parameter tuning of 5 parameters has been performed and results have been discussed.

2 INTRODUCTION AND MOTIVATION

"You shall know a word by the company it keeps." This quote by an English linguist J.R. Firth perfectly describes the idea behind a vector space model. In Natural language Processing algorithms, it is common to use vector space for representing text. With advancement in machine learning and deep learning over the years, many breakthroughs have been made in NLP, and word embeddings are one of them.

Prior to 2013, Bag-of-words (BOW) was a common method traditionally used for vector representation in NLP. However, there were many issues with the aforementioned technique. In BOW, the size of the vector is equal to the number of elements in the vocabulary. Indeed, each n-gram , or word, is marked with a 0 or a 1 depending on whether or not it appears in a document.

However, one of the challenges and limitations of Bag Of Words representations, is that they do not account for the semantic relationship between words, and in general they do not contain any information related to the meaning of the word itself.

3 LITERATURE REVIEW

3.1 Word2Vec

The main innovation to BOW, introduced by Mikolov et al. in 2013 [1] was proposing a prediction-based modelling technique, called Word2Vec [2], that would use a neural networks to learn intelligent representation of words in a vector space, instead of counting word co-occurrences.

This prediction based modelling technique was based on the idea of word embeddings, which are a type of word representation that allows words with similar meaning to have a similar representation. Embeddings per se, are a way to encode words in a sentence to numbers, that can allow to represent both semantic and syntactic properties of a word. Mathematically, word embeddings are a parametrized function of a word, which can be defined as follows:

$$f_\theta(W_n) = \theta_n$$

Where W is the word in the sentence and θ is the parameter.

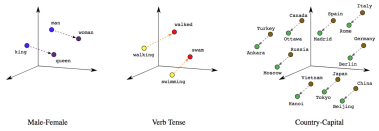

Figure 1: Word Embeddings: Linear Relationships between Words

Word2Vec relies on the distributional hypothesis according to which words which often have the same neighbouring words tend to be semantically similar. This helps to map similar words to geometrically close embedding vectors. Indeed, what the model essentially does is taking a text corpus as input and producing the word vectors as output. First, it builds a vocabulary from the training text data and then, it learns vector representations of words. The main objective is then that of improving predictive ability by minimizing the loss in the prediction of the vector for a target word or context, from the vectors of the surrounding context words, thanks to the Word Cosine distance tool.

More in detail, Word2Vec proposes two different architectures:

- Continuous Bag-Of-Words (CBOW): the model learns to predict a target word leveraging all words in its neighbourhood. The sum of the context vectors are used to predict the target word and the neighbouring words taken into consideration are determined by a predefined window size surrounding the target word.
- Continuous Skip-Gram: the model learns to predict a word, based on the neighbouring context words.

According to Mikolov et al, skip-gram works well with a small amount of the training data and is good at representing even rare words or phrases, while CBOW is several times faster to train than the skip-gram, and has slightly better accuracy for the frequent words. Nevertheless, skip-Gram model is a better choice most of the time due to its ability to predict infrequent words, but this comes at the price of increased computational cost.

3.2 Skip-Gram

Mikolov et al's negative-sampling word-embedding method presented in their paper is widely used all around the globe. The departure point of the paper is the skip-gram model, briefly introduced

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

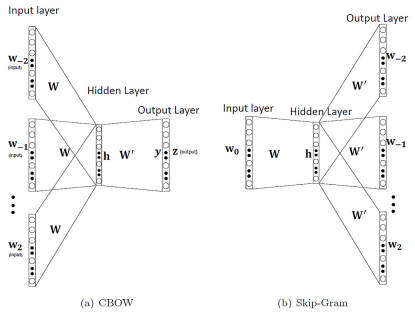

Figure 2: CBOW and Skip-gram models architectures

in the previous sections. Given a center word, this model aims at maximizing the probability of correctly predicting all the context words at the same time. This, translates into optimizing the word weight (embedding) matrix θ that best represents the words in a vector space. θ here, is a concatenation of both the input and the output weight matrices and gets passed into a cost function J in order to be optimized as follows:

$$\underset{\theta}{\operatorname{argmax}}\, p(w_1, w_2, ..., w_C | w_{center}; \theta)$$

Where C is the window size, namely the size of the context location at which the words need to be predicted, and w is the word vector, a context in the case of Skip-Gram.

Nevertheless, in order to classify context words, a SoftMax function is used. The SoftMax function, which, in the Skip-Gram model is defined as follows

$$p(w_{context} | w_{center}; \theta) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

where $W_{output_{context}}$ is a row vector for a context word from the output embedding matrix, and h is a hidden layer word vector for a center word. The SoftMax function is then plugged in the probability function showed before. By doing so, we obtain a new objective function that maximizes the probability of observing all C context words, given a center word, namely:

$$\underset{\theta}{\operatorname{argmax}}\, log \Pi_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

The actual maximization is pursued through the negative log-likelihood minimization, namely the negative of the equation defined here above. Precisely, the cost function to minimize is the following:

$$J(\theta; w^{(t)}) = -log \Pi_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

where c is the index of the context word next to the center word (w_t) and t is the index of the center word within a corpus of size T. Simplifying the equation we will obtain:

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} (W_{output_{(c)}} \cdot h) + C \cdot log \sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)$$

It is worth noting that the window size C is itself a hyper parameter of the model with a typical range of $[1, 10]$ which is set according to the idea that the Skip-Gram model aims at predicting a limited number of neighbouring words. Given this, we can rewrite the SoftMax function adapted for this specific case as follows:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} log \frac{exp(\theta^{(t+j)T} x^{(t)})}{\sum_{i=1}^{K} exp(\theta^{(i)T} x^{(t)})}$$

The aforementioned minimization, is applied through a neural network architecture with one hidden layer h, defined as follows

$$h = W_{input}^{T} \cdot x \in \mathbb{R}^{N}.$$

The matrices $W_{input} and W_{output}$ are optimized through forward and backward propagation and Stochastic Gradient Descent is used for parameter optimization, and it allows to obtain the following parameter update equation:

$$\theta^{new} = \theta^{old} - \eta \cdot \nabla_{J(\theta; w^{(t)})}$$

where η is the learning rate, ∇ is the gradient for the weight matrix, $j(\theta)$ is the cost function and $w^{(t)}$ is the center word.

When using SkipGram, it is also worth considering the computational inefficiency of SoftMax when applied in this context, which is the main reason why what is actually used is not exactly SoftMax but a set of independent binary classifications instead.

The training of the model involves a back propagation phase, in which the weight matrix θ is updated in order to optimize the words' vector representations. However, as we mentioned before, θ is a concatenation of the input and output weight matrices, therefore there will be two update equations, one or each embedding matrix, and the equation will be rewritten as follows:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot \frac{\delta J}{\delta W_{input}}$$

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot \frac{\delta J}{\delta W_{output}}$$

Then, by substituting the above matrices' gradients in their respective equation we obtain:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot x \cdot (W_{output}^{T} \sum_{c=1}^{C} e_c)$$

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot h \cdot \sum_{c=1}^{C} e_c$$

Where e_c is the prediction error for a $c - th$ context word in the window. The hidden layer h is equivalent to $W_{input}^{T} x$.

3.3 Optimization Technique - Negative Sampling

In vanilla skip-gram model, the weight matrices are updated for each training sample. As the size of training corpus increases, updating the wight matrix for every sample becomes unrealistic and excessively expensive in terms of computational cost. To address this issue, we use negative sampling.

In this technique, **K** negative samples are randomly picked from a unigram distribution where more frequent words are more likely to be selected as negative samples. These samples are paired with

the center word and given a probability based on how likely they are to appear together. The probability of observing positive pairs is maximized, while minimizing the probability of negative pairs. This technique also essentially converts the multi-classification task to binary classification task, i.e., weather a chosen word is in the context window of the center word or not, consequently allowing us to use sigmoid function over softmax function.

Let p(D = 1 | w, c) be the probability that a pair (w,c) of center word and its context is observed and p(D = 0 | w, c) be the probability of the pair not observed along with θ as the parameter controlling the probability distribution. For each positive pair, **K** negative words are drawn with $W_{neg} = \{c_{neg,j} \mid j = 1, \dots, K\}$. Thus, our objective function is

$$\underset{\theta}{\text{argmax}}\, p(D=1|w,c_{pos};\theta) \prod_{c_{pos}\in W_{neg}} p(D=0|w,c_{neg};\theta)$$

$$= \underset{\theta}{\text{argmax}}\, p(D=1|w,c_{pos};\theta) \prod_{c_{pos}\in W_{neg}} (1-p(D=1|w,c_{neg};\theta))$$

To simplify taking derivatives, it is common practice to use a natural log for an objective function. As natural log is a monotonically increasing function and hence, incorporating it does not affect the distribution of output. Any value in the original probability function occurs at the same point in the log probability function. However, using properties of log, the calculation becomes easier and efficient. Therefore, we'll take log of the above equation which would give us,

$$= \underset{\theta}{\text{argmax}}\, logp(D=1|w,c_{pos};\theta) \prod_{c_{pos}\in W_{neg}} (1-p(D=1|w,c_{neg};\theta))$$

$$= \underset{\theta}{\text{argmax}}\, logp(D=1|w,c_{pos};\theta) + \sum_{c_{pos}\in W_{neg}} (1-p(D=1|w,c_{neg};\theta))$$

We can replace the binomial probability p(D = 1 | w, c) with $\frac{1}{1+exp(\bar{c}_{output_{(j)}}\cdot w)}$ and can also write it in sigmoid function notation. We would get

$$\underset{\theta}{\text{argmax}}\, log\sigma(\bar{c}_{pos}\cdot\bar{w}) + \sum_{c_{pos}\in W_{neg}} log\sigma(-\bar{c_{neg}}\cdot\bar{w}))$$

Incorporating the optimized objective function into skip-gram model, our cost function becomes

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0} (log\sigma(\bar{c}_{pos}\cdot\bar{w}) + \sum_{c_{pos}\in W_{neg}} log\sigma(-\bar{c_{neg}}\cdot\bar{w}))$$

3.4 Other Optimization techniques

Hierarchical Softmax. It is a technique for approximating the softmax function which reduces the compute cost of training softmax neural network. Using Hierarchical Softmax technique, we are able to retain the probability distribution, but at the cost of less accuracy! In this technique, we organize our vocabulary into binary tree where all the vocabulary words are leaves of the tree. While any organization of words can be used, it is preferable to use Huffman tree for better accuracy.Using Huffman tree would organize the words such that rare words are at deeper levels while frequently used words are shallower levels. This essentially allows us to reach all words by a path from the root through inner node. This path is used to estimate the probability of the words represented by the leaf unit. Thus, each inner unit has an output vector which is defined as

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma(\langle n(w,j+1) = ch(n(w,j))\rangle v'^{T}_{n} v_{w_I}$$

Where angled braces represent boolean checking if the case is true or false; L(w) is the depth of the tree; ch(n) is the child of node n.

Hierarchical softmax technique enables the algorithm to be evaluated with logarithm to base 2 of the vocabulary which drastically reduces the computational complexity and number of operations needed for the algorithm.

$$O(V) \rightarrow O(log_2 V)$$

Sub-sampling of Frequent words. There are numerous words in languages which are used very frequently but don't add a lot of value to the meaning of the sentence. They are essentially used as semantics and ensuring the proper grammar is in place. Subsampling is the technique that removes these frequently used but of not much value words. This is comparable to removing stop words from the sentence, but words apart from being stop words are often removed as well based on their frequency of occurring. This technique deals with the imbalance between rare and frequent words. Each word w_i is discarded from corpus while training with probability defined as

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

where t is a chosen threshold usually with the value of 10^{-5} and $f(w_i)$ is the frequency of word w_i

4 PRE-PROCESSING

Text pre-processing is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case. There are multiple techniques that can be implemented to get cleaner data. We have used the following methods.

Punctuation removal.

Punctuation is usually used to express our thoughts better. However, they don't contribute towards assigning or calculating the similarity between words. Hence, they must be removed to obtain cleaner data. We have used python's in-built 'String' library to do so!

Text to lowercase.

As the technique is based on similarity of words, it makes no difference if a word has a lowercase character or uppercase character. Hence, to maintain the uniformity and to enable matches between words we change the entire corpus to lowercase characters.

Stopwords removal.

Since the model is based on the context and the meaning of words in that given context,we added an option of removal of stopwords because their impact on the meaning is ambivalent. We implemented this practice by using the distribution of the words. If the distribution exceeds a determined threshold then we assume that the word is a stopword in our text.

In this stage we did not apply lemmatization. This method has a significant affect on the meaning of the context hence we did not believe that our problem definition required those methods.

5 EXPERIMENTS

5.1 Hyperparameter Tuning

The model uses multiple parameters as inputs. Some of the parameters have default values which were defined in the provided code template for the exercise. We chose five hyperparameters to experiment due to time limitation of running the model. Those parameters are as following; negative size which denotes the number of negatively sampled word for a given word, window size which is the size of the subset considered to be in the same context in the sequence , minimum count which refers to the threshold value to determine a word is considered to be frequent or not, learning rate which is used in training process and the number of epochs which denotes the number of times the model goes over the training dataset.

5.1.1 Window Size. An experience is run to test the range of window size values from 1 to 7. The other parameters are kept as default (minCount=5, negativeSize=5, lr=1e-1, nEmbed=100), and the epoch is set as just one for this experiment. The optimal value in terms of RMSE value is found out to be 3 (Figure 3).

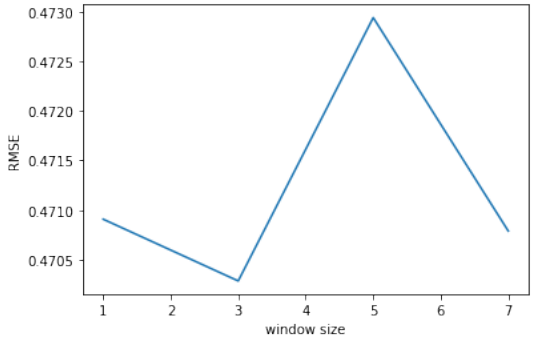

Figure 3: window size vs RMSE

5.1.2 Minimum Count. An experience is run to test the range of minimum count values from 1 to 7. The other parameters are kept as default (winSize=3, negativeSize=5, lr=1e-1, nEmbed=100) and the epoch is set as just 1 for this experiment. The optimal minimum count value in terms of RMSE value is found out to be 3 (Figure 4). The value of 7 is disregarded since there is a steep decrease after 6 and we assumed that the model might be under performing at that point.

5.1.3 Negative Size. An experience is run to test the range of negative count values from 1 to 7. The other parameters are kept as default (winSize=3, minCount=5, lr=1e-1, nEmbed=100) and the epoch is set as just 1 for this experiment. The optimal negative size value in terms of RMSE value is found out to be 6 (Figure 5).

5.1.4 Number of Epochs and Learning Rate. An experience is run to test the range of learning rate values of 0.1, 0.01, 0.001, 0.0001 and

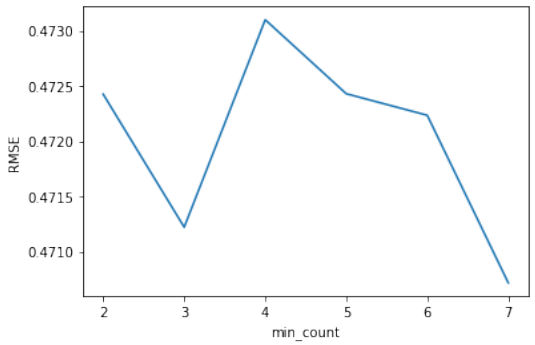

Figure 4: minimum count vs RMSE

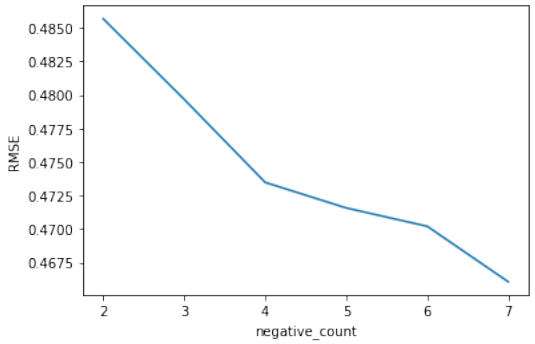

Figure 5: negative count vs RMSE

number of epoch values of 5, 10, 15, 20, 25, 30. The model is trained for the combination of those values with other parameters are kept as default (winSize=3, minCount=5, negativeSize=5, nEmbed=100) The optimal value for the learning rate is found out to be 0.1 and the optimal number of epochs are found to be 10 (Figure 6).

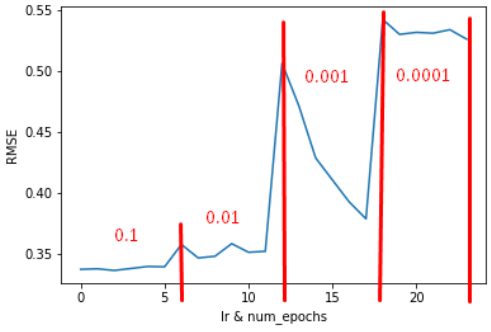

Figure 6: lr and number of epochs vs RMSE

5.2 Additional Experiment

During hyperparameter tuning the number of embeddings (number of features in word representation) was set as 100 as default. An additional experiment is conducted to test the affect of nEmbed

value on the model's performance. The previous optimal parameter values are kept in this test except the number of epochs due to computational cost. nEmbed values in a range between 50 to 1000 are tested for 1 epoch in the same model. The optimal nEmbed value is found to be 50 (Figure 7). According to the result of the experiment as the number of embeddings increases the error of the model also increases. However, if we choose a very small embedding number such as 1 or 10 the performance decreases again and gives and RMSE score above 0.4.

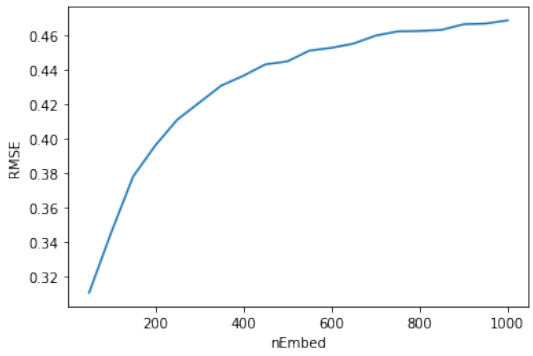

Figure 7: lr and number of epochs vs RMSE

6 CONCLUSION

The fine-tuned model is run with parameters window size = 3, minimum count = 3, negative size = 6, learning rate = 0.1, nEmbed = 100 for 10 epochs and the corresponding root mean square error during testing is 0.3376. However, after implementing the result of the additional experiment and decreasing the nEmbed value to 50, the RMSE score decreased to 0.3068.

We believe that the model has space to be improved but due to the time and computational limitations, not all possible scenarios could be tested.

REFERENCES

[1] Distributed Representations of Words and Phrases and their Compositionality. https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.
[2] Word2Vec. https://code.google.com/archive/p/word2vec/.