Master in Data Sciences & Business Analytics

NLP Assignment 1

Konstantinos Mira
Nevina Dalal
Maria Isabel Vera Cabrera
Zhenning Li

February 18th, 2022

1 Introduction

Natural Language Processing (NLP) is the part of Artificial Intelligence charged with having machines learn from all forms of language-based data, such as raw text, raw audio, documents, etc. For this report, the focus is placed on an approach to learn raw text.

Word embedding, also known as word representation, is the task of transforming text data into numerical data, which is at the core of NLP. In order for machines to learn from raw text, which tends to be unorganized and messy, raw text needs to be transformed into a numerical vector space. By turning words into numbers, we can make analysis and comparison between words much easier. The word vectors created from the raw text then can be compared based on their proximity to each other. That is, word vectors that are close to each other are also related. But, not all word vectors are created equally.

1.1 Motivation

Since any language has an immense vocabulary, the transformation of an entire language's vocabulary into numerical format is not feasible by hand.

A naive approach could be to one-hot encode all words. For instance, a vocabulary with 10,000 words would have vectors of 1x10,000 for each word. This approach has several issues. Firstly, the size of the vectors would be massive, as it depends on the size of our vocabulary. This results in the curse of dimensionality, as both space and time complexity increase exponentially. Secondly, these models would suffer with unseen text data, and may require re-training and readjusting of vocabulary.

Thus, an unsupervised learning technique that can learn the context of any word must be used, or vice versa. By learning the relationships between contexts and words, dimensionality is kept at bay, models are more efficient with transfer learning, and more meaningful analysis could be performed. Two well-known neural network architectures are used for accomplishing this unsupervised learning task, Continuous Bag of Words Model (CBOW) and Skip-Gram. The CBOW model aims at predicting a target word given a context word(s), a group of words. On the other hand, Skip-Gram aims to identify the context word(s) given a target word. This report focuses on Skip-Gram, more specifically with Negative Sub Sampling.

1.2 Problem Definition

How does one train a neural network to predict word embedding without labeled data? Rather than being interested in the inputs and outputs, the neural network task will focus on learning the weights of the hidden layer, which will actually be the word vectors. Moreover, the algorithm adjusts the weights in the back pass propagation step for each of the context words by calculating the errors vectors. The errors vectors (of 1xV dimensions, V being the size of the vocabulary) correspond to each context word at the output layer. An element-wise sum is performed to get a 1xV dimension vector, which is used to update the weights of the hidden layer. Negative Sampling aims to only update the weights of relevant vectors, so as to reduce the complexity of the algorithm. Moreover, the established model will undergo hyper-parameter tuning in order to optimize its performance.

2 Data Pre-Processing

Since the data is unstructured we first began by cleaning the data. To do this we performed 5 major pre-processing tasks.

1. Replaced the special characters and escape characters with a empty space except for the special character (') since sentences like "I'd like to do this homework" could exist;

2. Deleted numbers since we are dealing with text, so the number might not add much information to text processing. So, numbers can be removed from text;

3. Removed white spaces since most of the time text data contain extra spaces or while

performing the above pre-processing techniques more than one space is left between the text so we need to control this problem;

4. Converted the text to lower case since upper and lower case vocabulary have the same meaning and the models are case sensitive;

5. Removed sentences which has less than 1 word and converted the sentences to word lists separated by commas.

The result that will be fed into model is a list of list with the pre-processed words, one for each sentence.

3 Methodology

An input word will be represented as a one-hot vector which contains, for example 10,000 components.
The network's output is a single vector (10,000 components) giving the probability that a randomly picked neighboring word is the vocabulary term for each word in our vocabulary.
The input is a one-hot vector representing the input word, and the training output is also a one-hot vector representing the output word when training this network on word pairs. When evaluating the trained network on an input word, the output vector will be a probability distribution after transformation.
We want to maximize the probability of predicting, for example, the j^{th} word in the c^{th} context and hence minimize the loss function defined above. To accomplish so, we use two matrices to represent the vertices that correspond to the indexes: W for word embedding and C for context. Finding the optimum matrices of representations W and C is our aim.
The vector for a specific "target word" is represented by a row of matrix W, while a word as a context is represented by a row of the matrix C.

3.1 Vocabulary

We first created a dictionary of unique words present in our data set. This dictionary is known as vocabulary and is known words to the system and links every word to its key. In case of new data the word is tagged as unknown in the vocabulary. We keep a count and list of unique words and the 'Word to ID' allows us to assign a specific line to each word in W and C. A reverse dictionary is also developed for convenience, going from the key to the word.

3.2 Negative Sub Sampling

If we consider a pair (w, c), where w is the target/center word and c is its context, we want to determine if this pair was derived from the context window or from the noise of the distribution. Let $p(D = 1|w, c)$ be the probability that (w, c) is observed in the true corpus, while $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ is the probability that (w, c) is not observed in the true corpus. There are parameters θ, the weights matrix, controlling the probability distribution: $p(D = 1|w, c; \theta)$.

The $p(D = 1|w, c; \theta)$ can be also defined as:

$p(D = 1|w, c; \theta) = \frac{1}{1+\exp(-\bar{c}_{output_{(j)}} \cdot w)}$

The goal of Negative Sub Sampling is to optimize the parameters θ by maximizing the probability of observing positive pairs (w, c_{pos}), pairs that are more related, while minimizing the probability of observing negative pairs (w, c_{neg}). For each positive pair, the model will randomly draw K negative words $W_{neg} = \{c_{neg,j}|j = 1, ..., K\}$. Therefore, optimizing the parameters θ we aim to perform the following objective function:

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

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

3.3 Loss Function Derivation

From this definition of the problem, we can derive the Loss Function (L) appropriate for this algorithm, first by applying a natural log to simplify the previous calculations, which transforms into the following thanks to the property of logs:

$argmax_\theta \log p(D = 1|w, c_{pos}; \theta) + \sum_{c_{neg} \in W_{neg}} \log(1-p)(D = 1|w, c_{pos}; \theta))$

Now, we update the equation with our previous definition of the binomial probability to:

$argmax_\theta \log \frac{1}{1+\exp(-\bar{c}_{pos} \cdot \bar{w})} + \sum_{c_{neg} \in W_{neg}} \log \frac{1}{1+\exp(\bar{c}_{neg} \cdot \bar{w})}$

Finally, we simplify the Loss Function L by applying the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$:

$argmax_\theta \log \sigma(\bar{c}_{pos} \cdot \bar{w}) + \sum_{c_{pos} \in W_{neg}} \log \sigma(\bar{c}_{neg} \cdot \bar{w})$ (1)

$L = \log \sigma(\bar{c}_{pos} \cdot \bar{w}) + \sum_{c_{pos} \in W_{neg}} \log \sigma(\bar{c}_{neg} \cdot \bar{w})$ (2)

This is our final L function and it was implemented in the code as the `loss_function`.

4 Model Training and Similarity Measure

4.1 Training and Optimization

The goal of training the model is to learn the word embeddings we want. The way to do that is to convert our problem into a classification task, where the network predicts what words appears in the context. Negative sampling converts multi-classification task into binary-classification task. The new objective is to predict, for any given word-context pair (w,c), whether the word (c) is in the context window of the the center word (w) or not.

We initialized the W and C matrices randomly. The size of these 2 matrices are the same with number of unique words as rows and the dimension of embeddings as columns.We also define the window size randomly between 1 and the initial window size. Start and end for each window is also defined using this random window size.

We tried some different initialization ranges but there was no clear difference seen in the random versus the tried ones and hence we did not explore this further.

We optimize the W and C matrices progressively. Let (x, y) we a pair of word. We'll refer to W(x) as the word representation of x. We'll refer to C(y) as the context representation of y. We will compute the derivatives of L with respect to W(x) and C(y) and then update W(x) and C(y) accordingly.

$\frac{dL}{dW(x)}$ is a k-vector so is W(x) and hence the update is as follows:

$$\mathbf{W(x)} -= \mathbf{lr}.\frac{\mathbf{dL}}{\mathbf{dW(x)}}$$

The derivation for it is as follows:

Let θ be our weight matrix which is similar to the W(x) defined above.We take the derivative of the loss function defined above with respect to θ Assuming stochastic gradient descent, we have the following general update equations for the weight matrix (θ)

$\frac{\partial L}{\partial \theta} = (\sigma(\bar{c}_{pos}.h)) - 1) \frac{\bar{c}_{pos}.h}{\partial \theta} + \sum_{c_{neg} \in W_{neg}} \sigma(\bar{c}_{neg}.h)) \frac{\bar{c}_{neg}.h}{\partial \theta}$ (3)

Since θ is a combination of the W and C matrices the cost function needs to be differentiated with respect to each. We will call them as W_{input} and W_{output}

Gradients with respect to $\frac{\partial L}{\partial W_{output}}$: In negative sampling we only update a fraction of the W_{output} matrix. We take k+1 word vectors in the output word matrix. We take partial derivatives to the cost function defined in eq(1) and eq(2) with respect to positive words and negative words.

The only change made in eq(3) is changing the derivative with respect to $\bar{c}_{pos}$ and $\bar{c}_{neg}$ respectively.

$\frac{\partial L}{\partial \bar{c}_{pos}} = (\sigma(\bar{c}_{pos}.h)) - 1) * h$

$\frac{\partial L}{\partial \bar{c}_{neg}} = (\sigma(\bar{c}_{neg}.h))) * h$

The update equations are then as follows:

$\bar{c}_{pos}^{(new)} = \bar{c}_{pos}^{(old)} - \eta \cdot (\sigma(\bar{c}_{pos} \cdot h) - 1) \cdot h$

$\bar{c}_{neg}^{(new)} = \bar{c}_{neg}^{(old)} - \eta \cdot \sigma(\bar{c}_{neg} \cdot h)$

Gradients with respect to $\frac{\partial L}{\partial W_{input}}$:

only one word vector that corresponds to the input word w in W_{input} is updated with negative sampling. This is because the input layer is an one-hot-encoded vector.

$\frac{\partial J}{\partial h} = (\sigma(\bar{c}_{pos} \cdot h) - 1) \cdot \bar{c}_{pos} + \sum_{c_{eng} \in W_{neg}} \sigma(\bar{c}_{neg} \cdot h) \cdot \bar{c}_{neg}$

The update equation is then:
$\bar{w}^{(new)} = \bar{w}^{(old)} - \eta \cdot \sum_{c_j \in \{c_{pos}\} \cup W_{neg}} (\sigma(\bar{c}_j \cdot h) - t_j) \cdot \bar{c}_j$

where $t_j = 1$ for positive words and 0 for negative words.

4.2 Similarity

To define the similarity between the words we used the cosine similarity.It calculates the cosine of the angle formed by two vectors projected in three dimensions. Because of the cosine similarity, even if two comparable documents are separated by the Euclidean distance (due to the size of the documents), they are likely to be orientated closer together. The higher the cosine similarity, the smaller the angle. The formula for it as follows:

$$sim(A, B) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

4.3 Saving and Loading

We saved the model in a pickle format which stores the whole Skip-Gram class instance and not only the parameters.
We loaded the model for the doing the test we also loaded it in the pickle format.

5 Hyper-Parameter Tuning

We trained the model with different parameters on a small dataset for saving time, and the following graphs can help us better understand the hyper-parameters.

5.1 Window Size

We tried different values for Window Size which is the number of words to the left and to the right of the target that are considered its context. We tried 4 different values: 2,5,8 and 10 and we trained our model. We observed the average loss in each batch of sentences and the correlation between similarity calculated by the model and the one gotten from human annotations.

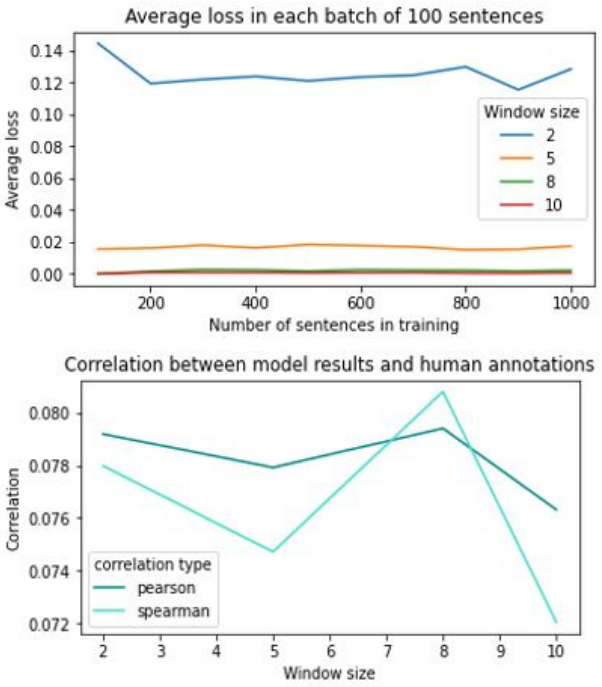

Figure 1: Effect of increasing window size in model performance

We see that as we increase the window size, the average loss decreases. When looking at the Pearson and Spearman correlation, we see they reach a peak at a window size of 8 words. However, when we consider its interaction and combinations with the other hyper-parameters, we found its best value is 5.

5.2 Negative Rate

The negative rate is the number of negative samples (incorrect training pair instances) that are drawn for each good sample. We tried 4 different possible values: 2,5,10 and 20. The average loss decreases as long as we increase the negative rate. However, when we look at the correlations between model and true similarities, we observe that the highest value is obtained with negative rate of 5. As our final evaluation is done in terms of correlation we decided to give more weight to this criteria. In theory, for smaller datasets, 5-20 negative words are a good range and for bigger datasets, 2-5 negative words are preferred. As we do not know the size of the final test data, it makes sense to choose a value that would suit

well both scenarios.

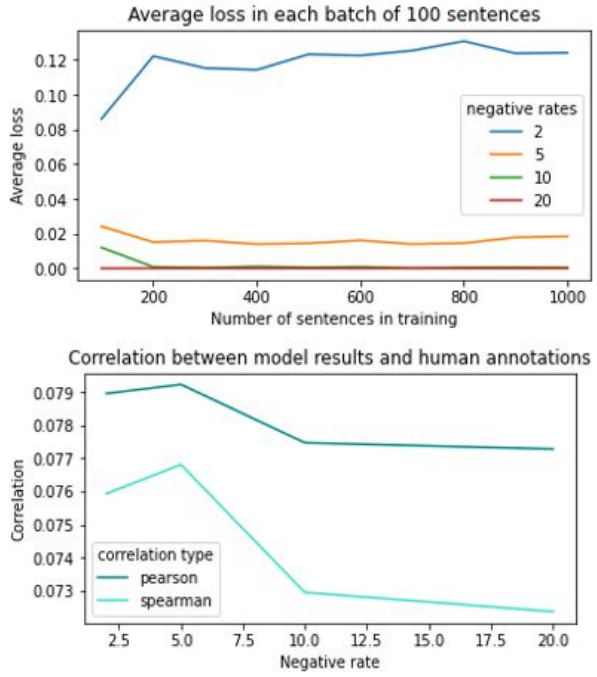

Figure 2: Effect of increasing the negative rate in model performance

5.3 Learning Rate

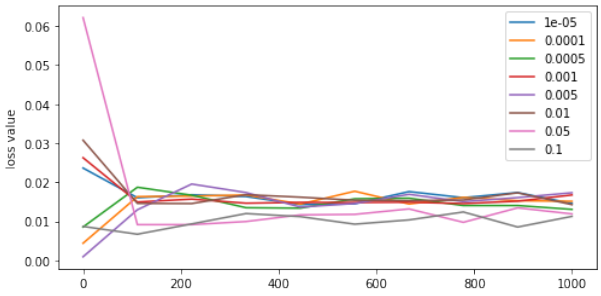

Figure 3: Effect of decreasing the learning rate

The learning rate controls the amount of adjustment made to the weights with respect to the loss gradient. It mainly concerns the training speed and some optimization problems (saddle point etc.). We tried 0.00001, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1 and as you can see in the graph, generally here the learning rate doesn't make huge difference as that of negative rate etc., but we still can see a learning rate of 0.1 and 0.05 performs better in both loss value and correlation.

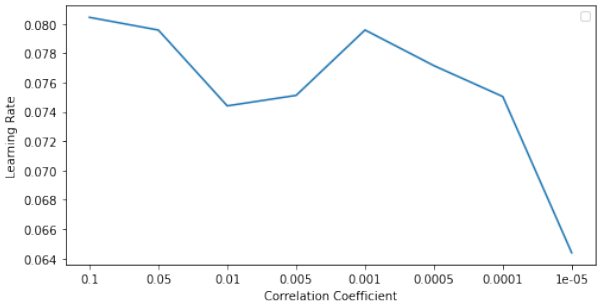

Figure 4: Correlation coefficient of decreasing the learning rate

5.4 Dimensionality of Embedding

This is the dimension of the word embedding, which is the number of features to define word in embedding (the size of the hidden layer). It typically ranges from 100 to 1800 depending on the vocabulary size. Dimension size beyond 300 tends to have diminishing benefit, and many academic paper recommend to use 300 as default value. We tried 50, 80, 100, 150, 200, 250, 300, 1200, 1800 and the result is quite similar to what we got in previous section: loss value decreased suddenly and the performance of 300 is the best both in correlation and loss value.

6 Model Performance

Our model achieved a Spearman correlation of 0.089 and a Pearson correlation of 0.084 with the human-annotated word similarity SimLex-999, a standard resource for the evaluation of word representation learning models.

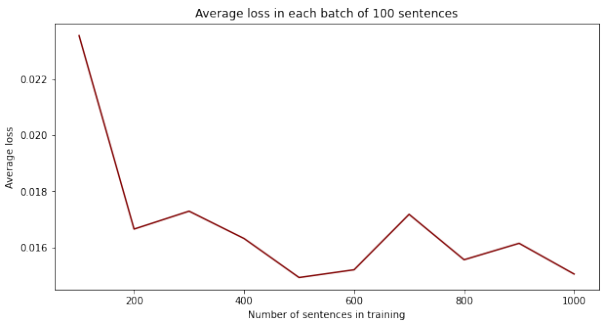

Figure 5: Average loss on each sentences batch

7 References

1. *Kim, Eric. "Optimize Computational Efficiency of Skip-Gram with Negative Sampling." Pythonic Excursions, 26 May 2019*
2. *Word2Vec Tutorial*
3. *Understanding Word Vectors and Implementing Skip-gram with Negative Sampling*
4. *word2vec Parameter Learning Explained*
5. *Word2Vec Tutorial - The Skip-Gram Model*