# Skip Gram With Negative Sampling

Abhishek Singh Ahmad Chaar Rebecca Erbanni

## I. Introduction: The Derivative

We have implemented the skipgram model with negative sampling using the objective function mentioned in Mikolov's paper:

J = $\log \sigma \left( {v'_{w_o}}^{\mathrm{T}} v_{w_i} \right) + \sum_{j=1}^{k} E_{w_j \sim P_n(w)} \left[ \log \sigma \left( -{v'_{w_j}}^{\top} v_{w_i} \right) \right]$

Originally, we implemented the derivative of the objective function, which takes the form shown below. Under this formulation $v'_{w_o}$ represents the embedding of the positive context, $v_{w_i}$ the embedding for the word and $v'_{w_j}$ the emedding for a negative example:

$$\frac{\partial J}{\partial v'_{w_o}} = \frac{+v_{w_i}}{1+e^{-v_{w_i} \cdot v'_{w_o}}}$$

$$\frac{\partial J}{\partial v'_{w_j}} = \frac{-v_{w_i}}{1+e^{+v_{w_i} v'_{w_j}}}$$

$$\frac{\partial J}{\partial v'_{wj}} = \frac{-v_{w_j}}{1+e^{+v_{w_i} v'_{w_j}}} + \frac{+v_{w_o}}{1+e^{-v_{w_i} \cdot v'_{w_o}}}$$

However, after a few iterations, we noticed that quickly the embeddings became nans. The reason being, that The derivative was computationally unstable due to the heavy use of exponentionals. Such issues are usually handled by Machine Learning software that takes care of the derivative calculation. Therefore, we used an alternative formulation of the cost function that takes advantage of the similarity between context and negative sample derivatives and incorporates a label. Hence, if we define the variable z that is represented by:

$z_o = {v'_{w_o}}^{\mathrm{T}} v_{w_i}$
$z_j = {v'_{w_j}}^{\mathrm{T}}_{w_i}$

The derivative of the resulting of the loss function will take on the form of:

$\frac{\partial J}{\partial z_o} = 1 - \sigma(z_o)$
$\frac{\partial J}{\partial z_j} = -\sigma(z_j)$

This form was much more numerically stable and we were able to parallalize the computation by combining both the negative and context samples into one list and calculating the negative sigmoid and adding a label of one to context samples and a label of 0 to negative samples. In the end the derivative of the word and context embedding takes the form of:

$$\frac{\partial J}{\partial v_{w_i}} = \sum_{j=1}^{k+1} \frac{\partial J}{\partial z_j} v'_{w_j}$$

$$\frac{\partial J}{\partial v_{w_j}} = \frac{\partial J}{\partial z_j} v_{w_i}$$

## II. Negative Sampling

### A. Unigram Table

For negative sampling we randomly picked n words for every context word of the target. Following the paper of Mikolov and listening to couple of video sources by Andrew Ng on the topic we chose words using a unigram distribution where more frequent words are more likely to get selected.

$P\left(w_i\right) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4}\right)}$

We are using the above formula to determine the probability of a word getting selected. Other variants of this formula were tried out experimentally such as picking a random word assuming all words have the same probability of being picked. The results showed that using the 3/4 provided the best MSE results, as sighted in many scholarly articles. The important part is how are we going to use this in the code to pick random negative words. For this we are creating a large zero vector with $10^6$ entries all filled with zero. For each word we calculate the probability using the above formula and then fill the zero matrix with the concerned word $p(w) * 10^6$ times. And then we will randomly pick n words from this large vector to carry out negative sampling. This way the words with higher probability is filled more times and has higher probability to get picked.

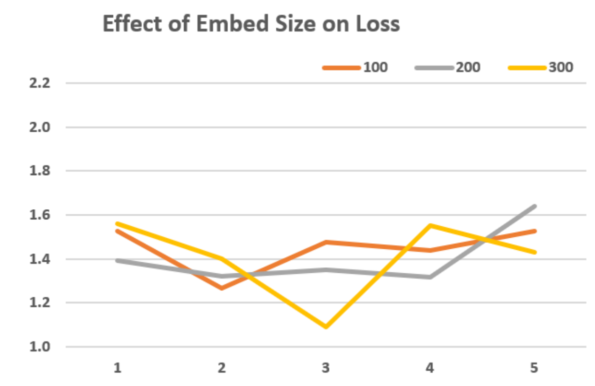
## III. Hyperparameter Tuning

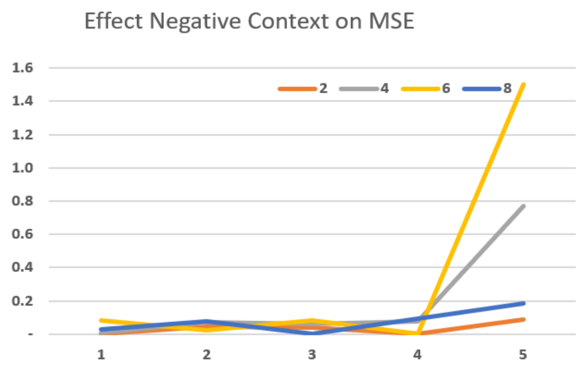
In order to tune our hyper parameters, we used three main measurements to evaluate the effectiveness of our model in getting the embeddings:

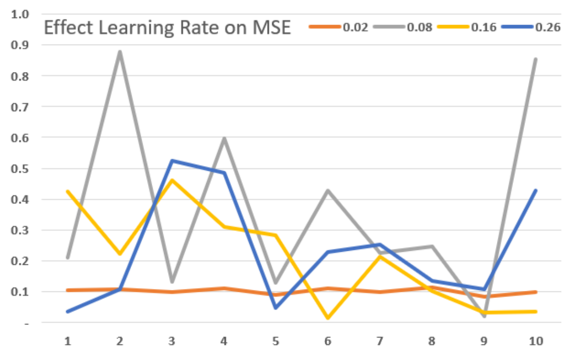
1) Calculation of the loss function.
2) Calculation of the Mean Square Error. This was done by computing the cosine similarity of paris of words obtained under the EN-SIMLEX-999.txt file and comparing to the results available in that file. Note that the scores under the SIMLEX file were divided by 10 in order to be comparable to the cosine scores. We also checked the word similarities
3) Checking the closest K-words for a given target word, using cosine similarity.

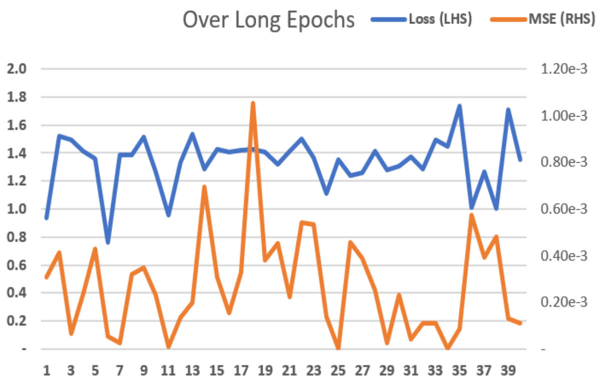
The below hyperparamaters were tuned as per the MSE and Loss. Please note that we only included a sub-sample of the charts checked:

1) Embedding Size: The impact of choosing the dimension of word embedding is critical. We changed the embedding size and checked how this was reflected on the loss and MSE across epochs, below are the results.
   It seemed from the figure above gave that 200 embedding had the most stable results.
2) Step-Size: This is the learning rate by which we will change the target and context word embedding. This has significant effect on how quickly the convergence happens. We looked at step-size with various values and its impact on the mean square error. It looked from the figure that a step size of above 0.8 was very volatile, therefore we chose a step size of 0.02 which gave good results overall.

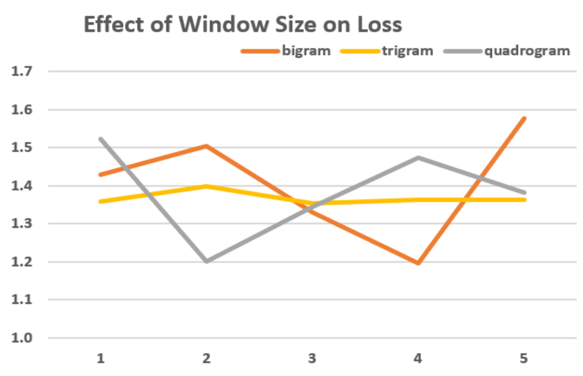








3) Window Size: Here we tried to see how many context words can be picked per target word. We started with window of 1 and tried for 2 and 3. In the end we chose a trigram.



4) Negative Samples per context word: According to paper by Mikolov we learned that size of k for negative samples can be around 5-20 for small datasets and 2-5 for large ones. As we have a huge vocabulary of data and after some testing we chose 4 negative sample words for each context word..
5) Epochs: Epochs were important to decide as the loss function needs to converge based on the epochs. We tried below epochs and observed the trend of loss. It showed that after 5 epochs, the loss function was stable more or less.

## Experiments:

- Building Dataset: Here we took two approaches at trying to build the data. In the first attempt, we read all the sentences, created the vocabulary dictionnary, doing the pre-processing for the vocabulary by creating the context word and negative samples before training starts. The second approach involved creating the context and negative samples on the fly when reading each sentence during training. We chose the first approach as it was computationally much faster in terms as the data capturing part was seperated from the matrix multiplication part of computing the embeddings. Although, the cost was that this approach required much more storage space but it saved a lot of time and allowed us to scale to many more words.
- Negative samples using random word method: Instead of the original unigram table approach we also tried picking random words from the vocabulary. We did not continue this approach as lot of papers cited the approach of unigram table helps improving the skipgram model and the results confirmed this.
- Case of Stopwords: We tried removing stop words from the sentences. However, based on the recommendations received as well as in order to save on computational time, we decided to keep all stop words.

## IV. Results

Having an overview of the results we saw that some were acceptable, although we did find some particularities such as having negative similarity scores for items. This could be due the direction of the change in embeddings.

| Similarity Measure | hard vs. difficult | happy vs. glad | wide vs. narrow | bad vs. terrible | large vs.huge |
|---|---|---|---|---|---|
| SIMLEX | 0.88 | 0.92 | 0.10 | 0.78 | 0.95 |
| Epoch 1 | 0.70 | 0.10 | 0.61 | 0.88 | 0.71 |
| Epoch 2 | 0.54 | 0.86 | -0.37 | 0.48 | 0.71 |
| Epoch 3 | 0.35 | 0.72 | 0.90 | 0.92 | 0.90 |
| Epoch 4 | -0.41 | -0.16 | 0.14 | 0.89 | -0.77 |
| **Epoch 5** | **0.83** | **0.89** | **0.05** | **0.89** | **0.93** |

We also looked at the K nearest word for a particular word, we found a few good results such as passenger had connection as it’s K-Nearest neighbor. We suspect that the size of the vocabulary (1,000 words )was not sufficient to obtain better results.