Soam Word Embeddings

Amit Baran Roy
Sourav Chakraborty
University of Colorado
Boulder, CO

June 2020

1 Introduction

It all started with the need to correctly encode words into mathematical objects which could help in more important tasks like training a model to predict the next word or analysing the sentiment of a review. The first and trivial case was to use a one-hot encoded multidimensional vectors. In this scheme, every word gets one dimension. The number of dimensions of the vector space would be the size of the vocabulary of the text. For instance, a text with a vocabulary: ('a', 'good', 'text') would have their corresponding vectors as [1 0 0], [0 1 0] and [0 0 1]. This is quite convenient, however, it is *highly inefficient*. If you notice, in this model, things get really problematic once the size of the vocabulary increases. This is generally a case of a *sparse* embedding.

The successor to this is the *dense* embedding. The efficient way is to fix the number of dimensions, which is low enough to make sure that it does not create computation problems, but large enough to capture the rich relationships between various words. So, instead of having sparse one-hot vectors for each word in the vocabulary, we will have fixed-dimensional *real* valued vector. A common configuration is to take 300 dimensions. Increasing the vocabulary size will not increase the dimensions, but will only increase the density of the vector space.

Now that we have the motivation for the dense representations, the natural questions are as follows:

1. How to make sure that the vectors are unique.
2. More importantly, how to make sure that the various relationships are captured in their vector values.

The keyword here is *context*. The basic assumption here is that words used in similar contexts are bound to be similar. For instance, if we have two sentences which says

1. Alice likes to eat apples.
2. Bob likes to eat oranges.

Then, we can say that apples and oranges are probably of the same class or kind. It is used in the same way. In this case: it is to eat. More nuanced sentences will reveal that they belong to a subclass of the things that are eaten: fruits. They will be a little different than vegetables like lettuce or cabbage, but will be different from something like a truck. These relationships are visible in their vector values.

2 Our Contribution

Word embeddings, in general, were produced as a by-product of feed forward neural networks. Such a network would take in words from vocabulary, embedding them into vectors in lower dimensional space which forms the first embedding layer. The embeddings (or weights) of this layer gets updated with backpropagation as the model is trained. It was in 2013, when Google's word2vec model was released whose sole purpose was to generate word embeddings using a computationally less expensive architecture having only 2 layers. Word2Vec is one of the most popular word embedding model and it has been trained on two different architectures- CBOW and SkipGram. In this project, we decided to implement the skip gram model.

A skipgram model consists of a sliding window (window length of 10 is used in word2vec) that scans the entire document line by line. The middle word of the window is the target word and the other words to the left and right of the target word in the given window are the context words. Training samples are formed such that a pair of context and target word is assigned a label score of 1, whereas a pair of non context word and target word is assigned a label score of 0. In this project, we decided to change this logic. Instead of binary score assignment, we chose our algorithm such that the label score is maximum if the context word lies to the immediate left or right of the target word and decreases gradually as the distance between context and target words increases. We have tried different scoring algorithms which are discussed in next section followed by the results.

3 Implementation

We implemented a skipgram model based on python with keras tensorflow framework on Google Colab. For the dataset, we used NLTK library to import five different Gutenberg texts which together made a vocabulary size of around 19178 unique words. For text preprocessing, we did the following:

- Changed all characters to lower case
- Normalized all characters to Ascii format
- Removed numeric digits, special characters and punctuations
- Removed stopwords
- Excluded some repetitive headings and title words

The dataset is then tokenized and vocabulary is generated. Every word is now represented with their corresponding word Id's instead of the word itself. This data representation is now used to generate the skipgram pairs which then forms the training samples. Our algorithm is applied in this skip gram generation phase. We tried different scoring variations and achieved interesting results which are discussed in results section. First, we tried the strict scoring algorithm of $\frac{1}{2^n}$ such that the immediate left/right target-context word pair is assigned a score of 1 and the score decreases with the rate of $\frac{1}{2^n}$ gradually with every next target-context word pair. Then we relaxed the scoring using the rate as $\frac{1}{n}$ and then we finally arrived at the scoring rate of $\frac{1}{1+((pos-1)^2*\alpha)}$, where pos is the distance of the context word from the target word and α is a constant. This setting gave better results as we decreased the weightage of context words in a non-linear fashion giving higher weights to nearby words and lesser weights to far away words.

After generating the skipgrams and training data, we designed the model using Keras functional API. The 2 layer model has an embedding layer and an output layer with sigmoid activation. The embedding layer is actually a dot product layer of two different embedding layers, essentially one for target word and another for context word. The dotted embedding layer is then fully connected to a single unit Dense output layer with sigmoid activation. We have chosen this output layer setting because our task is regression with single class prediction in the range 0 to 1. As it is a regression problem, we chose the loss function as mean squared error. We chose RMSProp optimizer as the task required gradient descent for the loss function to reach global minimum. After the model is ready, we trained the model for 6 epochs as the dataset was quite huge and we achieved quite reasonable results.

4 Results

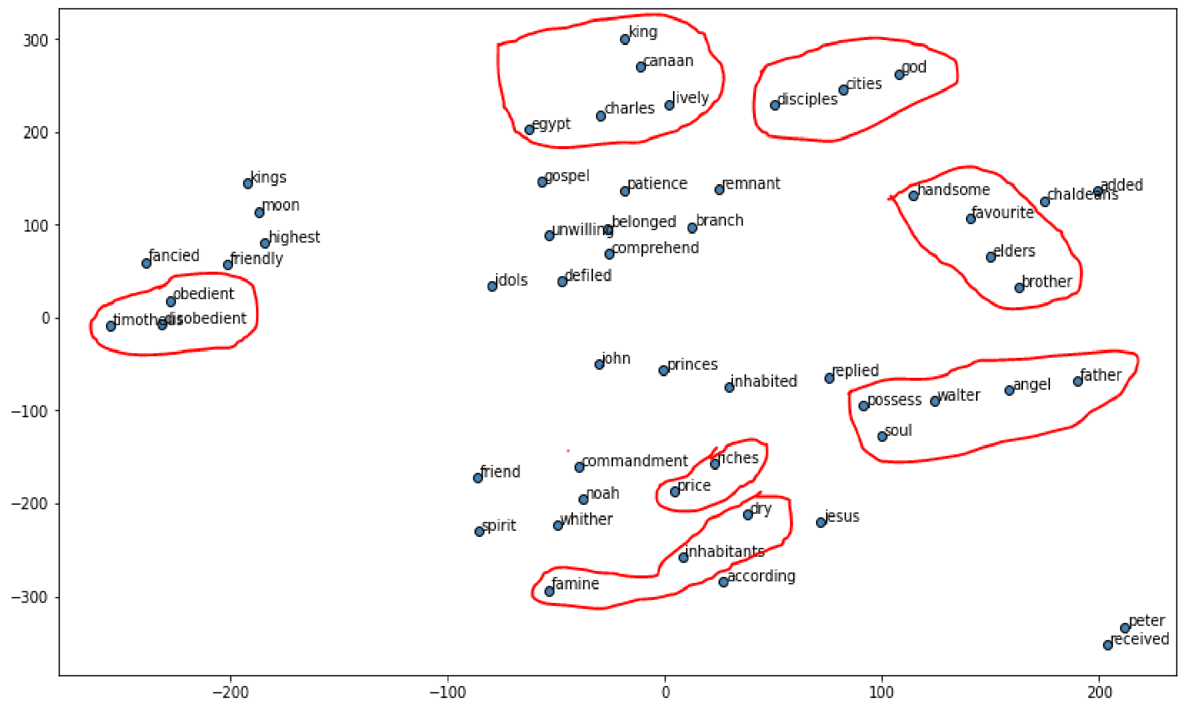
To interpret the results, we extracted the weights of the embedding layer. We picked few words to test and calculated the euclidean distances of the weights of those words with other words and chose the lowest distance values to group the most similar words together. We also used t-SNE to visualize the most similar words grouped together as shown below:-

- For scoring algorithm, $score = \frac{1}{1+((pos-1)^2 * \alpha)}$, we get:

```
similar_words
```

```
(19177, 19177)
{'egypt': ['canaan', 'dry', 'whither', 'inhabitants', 'possess'],
 'famine': ['defiled', 'inhabited', 'chaldeans', 'price', 'branch'],
 'god': ['father', 'according', 'brother', 'received', 'soul'],
 'gospel': ['riches', 'highest', 'patience', 'idols', 'moon'],
 'handsome': ['fancied', 'lively', 'comprehend', 'favourite', 'unwilling'],
 'jesus': ['angel', 'peter', 'commandment', 'disciples', 'spirit'],
 'john': ['walter', 'charles', 'added', 'friend', 'replied'],
 'king': ['kings', 'cities', 'remnant', 'elders', 'princes'],
 'noah': ['friendly', 'obedient', 'disobedient', 'belonged', 'timotheus']}
```

```
Total words: 54         Word Embedding shapes: (54, 100)
```

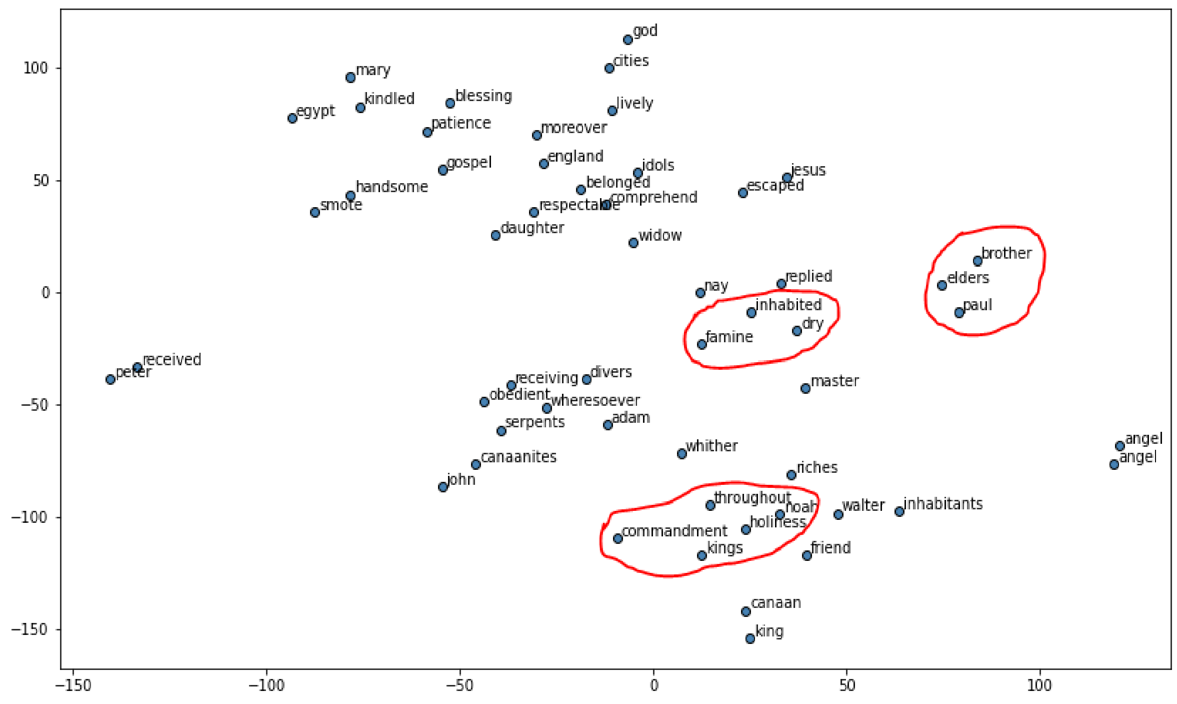


- For scoring algorithm, $score = \frac{1}{n}$, we get:

```
similar_words
```

```
(19177, 19177)
{'egypt': ['canaan', 'dry', 'inhabitants', 'whither', 'throughout'],
 'famine': ['inhabited', 'canaanites', 'divers', 'escaped', 'widow'],
 'god': ['brother', 'kindled', 'angel', 'received', 'master'],
 'gospel': ['patience', 'holiness', 'riches', 'idols', 'blessing'],
 'handsome': ['lively', 'receiving', 'comprehend', 'england', 'respectable'],
 'jesus': ['angel', 'peter', 'commandment', 'moreover', 'paul'],
 'john': ['walter', 'friend', 'replied', 'nay', 'mary'],
 'king': ['cities', 'kings', 'elders', 'daughter', 'smote'],
 'noah': ['adam', 'belonged', 'serpents', 'obedient', 'wheresoever']}
```

```
Total words: 54        Word Embedding shapes: (54, 100)
```

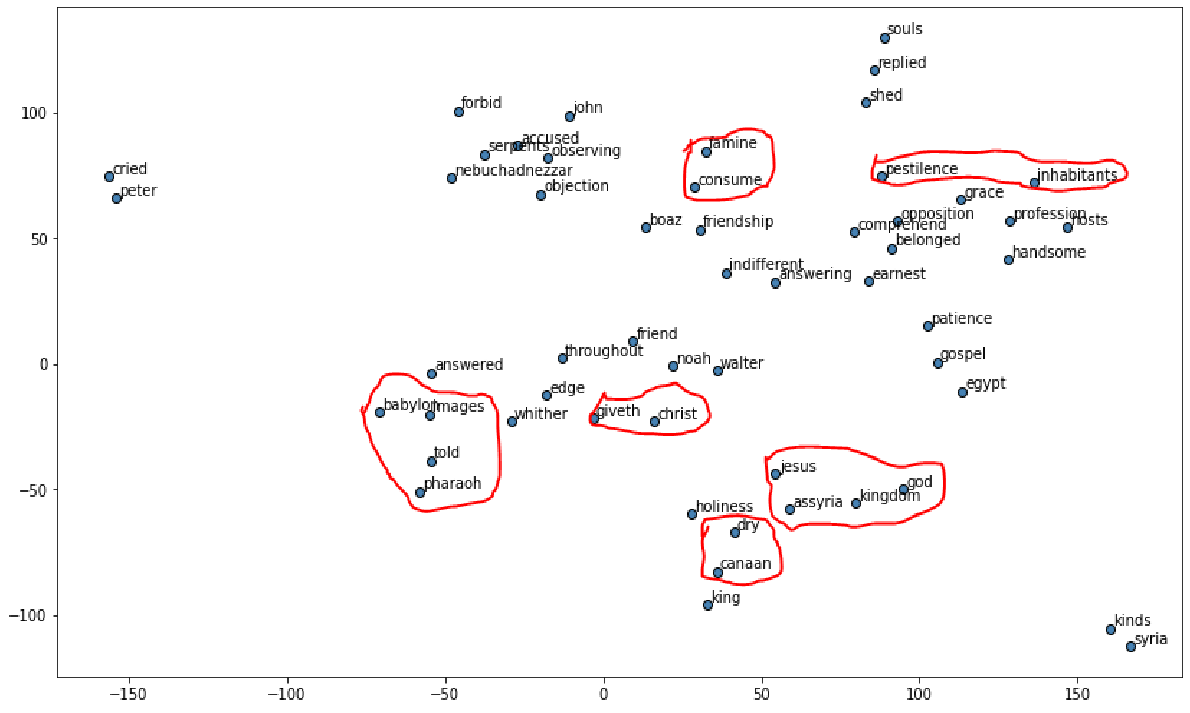


- For scoring algorithm, $score = \frac{1}{2^n}$, we get:

```
similar_words
```

```
(19177, 19177)
{'egypt': ['canaan', 'dry', 'throughout', 'whither', 'inhabitants'],
 'famine': ['images', 'souls', 'consume', 'pestilence', 'edge'],
 'god': ['hosts', 'kingdom', 'forbid', 'grace', 'giveth'],
 'gospel': ['patience', 'earnest', 'profession', 'friendship', 'holiness'],
 'handsome': ['indifferent',
  'opposition',
  'observing',
  'comprehend',
  'objection'],
 'jesus': ['christ', 'answered', 'answering', 'told', 'peter'],
 'john': ['walter', 'replied', 'friend', 'cried', 'shed'],
 'king': ['babylon', 'assyria', 'syria', 'nebuchadnezzar', 'pharaoh'],
 'noah': ['serpents', 'accused', 'kinds', 'boaz', 'belonged']}
```

```
Total words: 54          Word Embedding shapes: (54, 100)
```

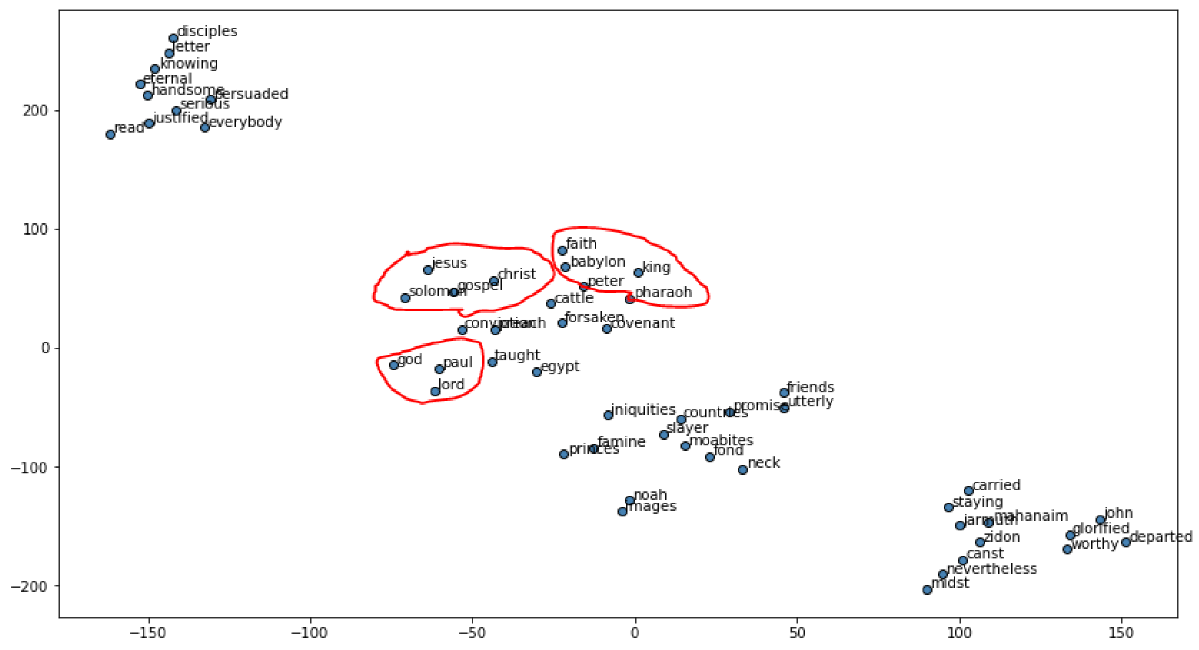


- Finally, for the baseline model where $score = 1$ for all samples, we get:

```
similar_words
```

```
(19177, 19177)
{'egypt': ['pharaoh', 'covenant', 'midst', 'cattle', 'utterly'],
 'famine': ['images', 'countries', 'forsaken', 'iniquities', 'neck'],
 'god': ['lord', 'canst', 'promise', 'justified', 'taught'],
 'gospel': ['preach', 'glorified', 'eternal', 'nevertheless', 'worthy'],
 'handsome': ['staying', 'everybody', 'fond', 'conviction', 'serious'],
 'jesus': ['christ', 'disciples', 'faith', 'peter', 'paul'],
 'john': ['friends', 'persuaded', 'read', 'letter', 'knowing'],
 'king': ['babylon', 'solomon', 'carried', 'departed', 'princes'],
 'noah': ['zidon', 'slayer', 'jarmuth', 'mahanaim', 'moabites']}
```

```
Total words: 54          Word Embedding shapes: (54, 100)
```



5 Conclusion

We have trained the model for only 6 epochs (as the vocabulary size is quite big with around 19000 words) with 'RMSprop' optimizer and learning rate of '0.001'. We can see that our scoring algorithms performed quite good as compared to the baseline result. In the above results, we can visualize the t-SNE plots and see that very similar words (marked in circle) are grouped together. We can also observe that as we make the scoring very strict such as $score = \frac{1}{2^n}$, the weights are distributed such that the immediate nearby target-context word pairs are highly weighted whereas, the distant context words in the window are given exponentially lower weights and they lose significance. We found that our first scoring algorithm performed better as we gradually reduced the weights of the subsequent context words in a non-linear fashion. The related code can be found at this Github link.

References

[1] B. Yoshua, D. Rejean, V. Pascal and J. Christian *A neural probabilistic language model.The Journal of Machine Learning Research* , 3:1137–1155, 2003.

[2] *Implementing Deep Learning Methods and Feature Engineering for Text Data: The Skip-gram Model*

[3] *Learning Word Embedding*

[4] *On word embeddings- Word2vec, Glove*

[5] *Keras: Multiple Inputs and Mixed Data*

[6] *A hands-on intuitive approach to Deep Learning Methods for Text Data — Word2Vec, GloVe and FastText*