Final report - Word embedding

Abstract:

Main goal for this article is to implement word embedding technique on a verse that has been taken from the "Torah" and try a good classifying method on them that will predict the book it was taken from, in this article will be implemented and used Keras Sequential model to try and predict our verse, in hopping to beat the last task prediction record that used Neural network method for prediction the same data.

Introduction:

Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

Main problem in this article is to classifying verses from the "Torah" to the book it was taken from, "Torah" includes 5 different books: Genesis, Exodus, Leviticus, Numbers, Deuteronomy, which mean the program will have to try and predict 5 different classes for each verse.

In this essay we will focus about natural language processing (NLP) with solution using **word embedding,** which is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e., angle close to 0.

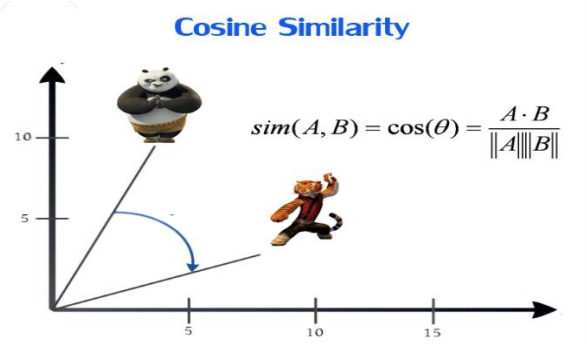

Figure 1

Related work and required background:

To solve the problem, it is needed a prior knowledge about what NLP is, what is word embedding, also it is needed to know how does classifying works and a machine learning process, with his optimizer and loss function.

Project description:

First, need to organize and get the data properly, put each book and its verses in an array of strings, each string in the array is a verse, then use keras.preprocessing.text tokenizer on the text, tokenizer will split each verse into words, then use its fit_on_texts function on the text update internal vocabulary based on a list of texts, this method creates the vocabulary index based on word frequency.
Second, split the data into train data and test data, in this project the split is 80%/20% for the train data, now use on the train, test the texts_to_sequences function which transform each text in the texts to a sequence of integers, so it basically takes each word in the text and replace it with its corresponding integer value from the word index dictionary, after that use pad_sequences to fill each vector of words with 0 at the end of its word so every vector will have the same size.
Third, now build the Keras Sequential model that will fit on the train data and try to classify each verse from the test data to the book it was taken from. The current model has 3 layers, one input layer, one middle layers with 85 neurons, and last layer has 5 neurons. Each move between those layers also included a Dropout of 0.2% which mean 80% of the neurons from that layer will move to the next layer and will not be ignored, that method will help to prevent overfitting the data and have a better result using that.
Model will compile a loss function as 'categorical cross entropy' and optimizer will be 'adam' (Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments).

Final step is to train the model, the model fit function shown the best result when were with 30 epochs and 1024 batch size, batch size is a number of samples processed before the model is updated. number of epochs is the number of complete passes through the training dataset. This means that the dataset will be divided into 5 batches, since all the train data is 4675 sentences to train with each with 1024 samples. The model weights will be updated after each batch of 1024 samples. This also means that one epoch will involve 5 batches or 5 updates to the model.
After the model finished training session it will go for validation data and try to predict the test data according to the labels we gave him, more right prediction will mean more accuracy for the model.

Experiments/simulation results:

Results of the experiment were when the Keras Sequential model has 3 layers, first one is the input layer, the second layer were 82 neurons with an 'Relu' activation, and in the third layer were 5 neurons with a 'SoftMax' activation. Embedding dimension is set to 300 which will represent each word by a vector of this much dimensions. maximum length of a sentence vector is set to 75 words, on the model epochs are set to 30 batch_size is set to 1024.
also adding a Dropout function with 0.2 (that means 80% of neurons are active and in use.) to the model between each layer, the result of 83.68% of a correct prediction for a verse to his book.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 75, 300)           1468800

 global_max_pooling1d (Globa  (None, 300)              0
 lMaxPooling1D)

 dropout (Dropout)           (None, 300)               0

 dense (Dense)               (None, 82)                24682

 dropout_1 (Dropout)         (None, 82)                0

 dense_1 (Dense)             (None, 5)                 415

=================================================================
Total params: 1,493,897
Trainable params: 1,493,897
Non-trainable params: 0
_________________________________________________________________
Training Accuracy: 0.9816
Testing Accuracy:  0.8368
```

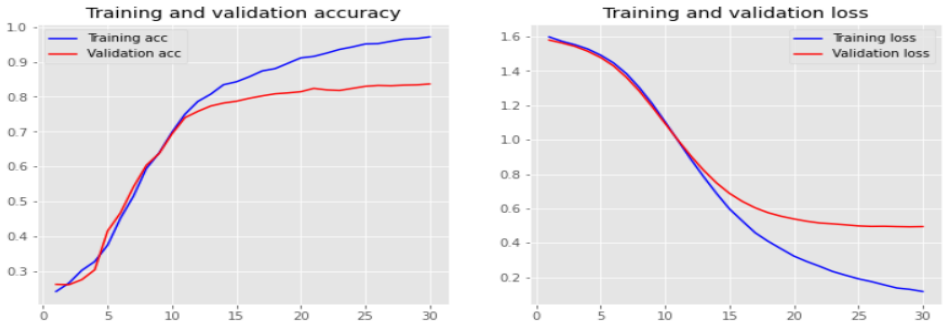

Figure 2

In comparison to a basic MLP:

MLP consists an input layer, a one or more hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training, Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. Previous solution had 3 hidden layers neural network (800->100->50) it's got an accuracy of 68.20513% of a correct prediction for a verse to the book it belongs.

```
0.6820513
```

Figure 3

In comparison to a basic SoftMax:

goal was to solve this classifying problem with a SoftMax version and then use a neural network on the same data, to see what will be a better predicter.
Try to solve the classifying problem was to try a SoftMax version, a try that use python's split() function to split the sentences before we went and implement the nltk tokenize and stem, this thing got an accuracy of 52.478635%.

Figure 4

Conclusions:

main project goal was to implement a word embedded technique in such a way that will try to properly predict a verse from the "Torah" to the book it was taken from. The project description talks about the final and most successful approach towards solving this case, try to implement such a solution is using to convert all data to a word embedded vectors that will hold each word as a number, and by doing that similar meaning sentences will be closer in the vector space. Which in the end will be that after learning the data, the program already knows what a sentence in a book should look like in a vector space and according to the high amount of percentage accuracy it got the program did well when tried to classify those verses.

Previous attempts:

For this project:

First try the model has 3 layers, in the first the input, second layer there were 50 neurons, in the third layer were 5 neurons, embedding dim set to 700, epochs are 12 batch size is 32, it gave a result in the region of 73.93%.

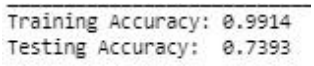

Figure 5

Second try to improve the model now has 4 layers, in the first layer the input, in the second layer there were 85 neurons, in the third layer were 10 neurons and in the fourth layer were 5 neurons, embedding dim set to 700, epochs are 100 batch size is 32, it gave a result of 76.07%.

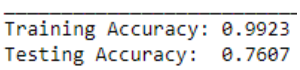

Figure 6

Third try to improve the model now has 4 layers, in the first layer the input, in the second layer there were 85 neurons, in the third layer were 10 neurons and in the fourth layer were 5 neurons, embedding dim set to 700, epochs are 16 batch size is 32, (Overfitting denied from the second try) it gave a result of 80.17%.

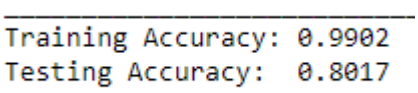

Figure 7

Fourth try to improve the model now has 4 layers, in the first layer the input level, the second layer contains 50 neurons, in the third layer were 15 neurons and in the fourth layer were 5 neurons, embedding dim set to 380, epochs are 19 batch size is 32 also adding a Dropout function with 0.2 (that means 80% of neurons are active) to the model between each layer, 82.48%.

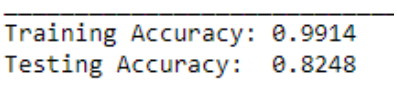

Figure 8

For deliverables 2:

First is in comparison section on regular SoftMax.

Second try used on the regular SoftMax with tokenize and stem that gives accuracy of an 56.41026%.

Figure 9

Third: the next try to improve We use 2 hidden layers that reduces the features from the 3725 there is at the beginning of the program to 500 neurons and then to 50 neurons, got an accuracy of 56.837606%.

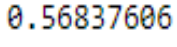

Figure 10