Introduction

The aim of this assignment was to develop a model that can predict the sentiment of a given text. The model was developed using movie reviews, and is, thus, able to classify reviews that are either negative or positive.

In particular, three MLPs were trained; one using pre-trained word embeddings, one that learns its own word embeddings and one that takes TF-IDF vectors as an input. Out all of them, the third performed the best, yielding 90.5% accuracy on the test set.

Dataset

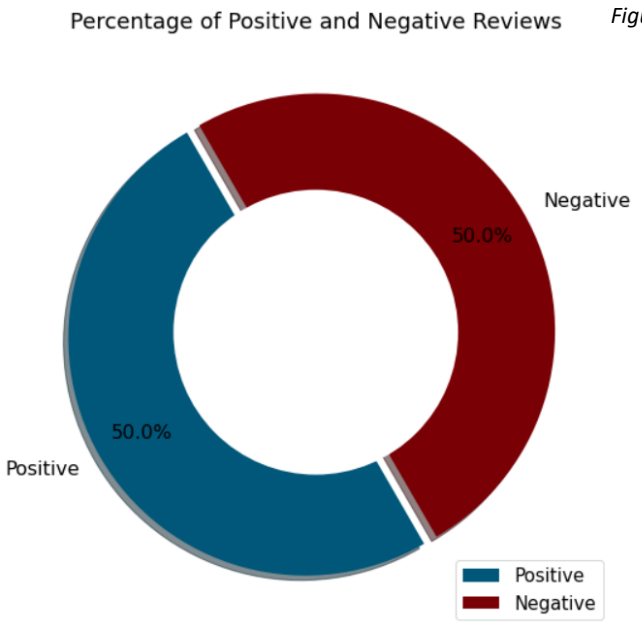


Figure 1

The dataset that was used contains 50,000 highly polar movie reviews[1]. After the stop words were removed 99,257unique words were left. Furthermore, there were not any imbalances in the dataset. That is, 50% of the reviews were positive and 50% of the reviews were negative. Lastly, the average number of words in a review were 120, with a standard deviation of 90 words.

Preprocessing

First of all, each review was thoroughly cleaned. Any number or punction was removed, leaving only words behind, with a length of at least 2. Stop-words were also removed.

For the first two models that used word embeddings, for each unique word of the *training set* an index was assigned, using *keras' Tokenizer.* Then, each review was transformed to a sequence of those unique indices. Lastly, each of those sequences was padded or truncated, so that all of them have the same predefined length. The

[1] https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

length that was chosen was the average number of words in a review plus their standard deviation (210).

For the third model, TF-IDF vectors were used as the input. The 7,000 unigrams/bigrams with the highest term frequency in the training set were used as features and each set of reviews was transformed accordingly.

Modeling

The dataset was split into three different sets; the training, the development and the test set. The training set consists of 70% of the whole dataset. The remaining 30% of the dataset was split into a development set (70%) and a test set (30%).

All the models were trained using the training set, tuned using the development set and tested using the test dataset. As a baseline (in order to evaluate each model's performance) the best performing model of the previous Assignment will be used (the ensemble classifier which scored 90% accuracy).

Pre-trained Word Embeddings

For the first model, Word2Vec's pre-trained word embeddings[2] were used as the input. Those 300-dimensional vectors were trained on about 100 billion words, which were part of Google News.

An embedding matrix was created that contained the word embeddings of each unique word of the training set (plus a row containing only zeros, due to the sentence-padding mentioned before). The matrix was then fed to an *Embedding Layer*. That Layer was set 'frozen', that is, the word embeddings were not updated during training.

[2] https://code.google.com/archive/p/word2vec/

Then, the *Embedding Layer* was flattened, and with a *Dropout* rate of 0.5, its output was passed to a *Dense Layer* of 64 neurons. Using a ReLu as the activation function, *Batch Normalization* and a *Dropout* rate of 0.7, the output of the first layer was passed to another *Dense Layer* of 16 neurons. A *ReLu* was used again as the activation function and with a *Dropout rate* of 0.6, the output of the second layer was passed to the final output layer, which was a *Dense Layer* with only one neuron and a *Sigmoid* as the activation function.

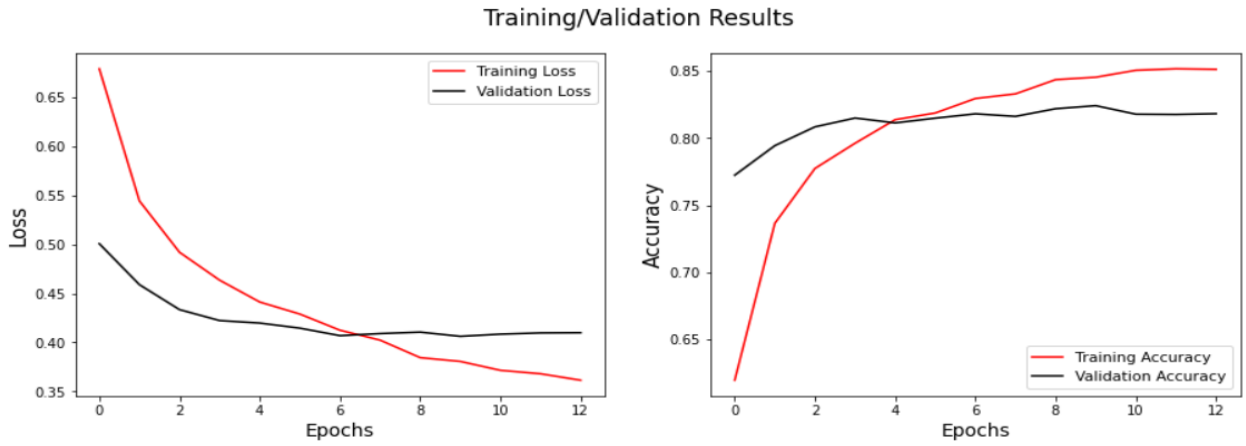


Figure 2

The model performed poorly compared to the baseline and the other two, yielding an Accuracy score equal to 82,4% on the development set and 82.5% on the test set. The scores on the rest of the metrics were also significantly lower and can be observed at table 1.

Table 1: Scores yielded by the first Model

Metrics	Training Set	Development Set	Test Set
Accuracy	0.9438	0.82419	0.825333
Recall0	0.940206	0.821591	0.831416
Recall1	0.947377	0.82682	0.819196
Recall_macro	0.943792	0.824205	0.825306
Precision0	0.946767	0.827547	0.82268
Precision1	0.940887	0.820844	0.828069
Precision_macro	0.943827	0.824196	0.825374
F1_0	0.943475	0.824558	0.827025
F1_1	0.944121	0.823821	0.823609
F1_macro	0.943798	0.82419	0.825317
AUC0	0.986797	0.895264	0.894364
AUC1	0.987342	0.894901	0.89839
AUC_macro	0.987342	0.894901	0.89839

Word2Vec Word Embeddings

For this model, using *Gensim's*[3] implementation of Word2Vec, the word embeddings were created using the *training set*. These embeddings were used in order to initialize an embedding matrix (the same way as before), but this time, they were updated during training.

The architecture of this MLP was similar to the previous one. It contained two hidden *Dense Layers* with 128 and 64 neurons that used a *ReLu* as an activation function. A *Dropout* rate of 0.6 was added on each layer[4], and *Batch Normalization* was used between the first and second hidden layer. The output layer had only one neuron and a s*igmoid* as its activation function.

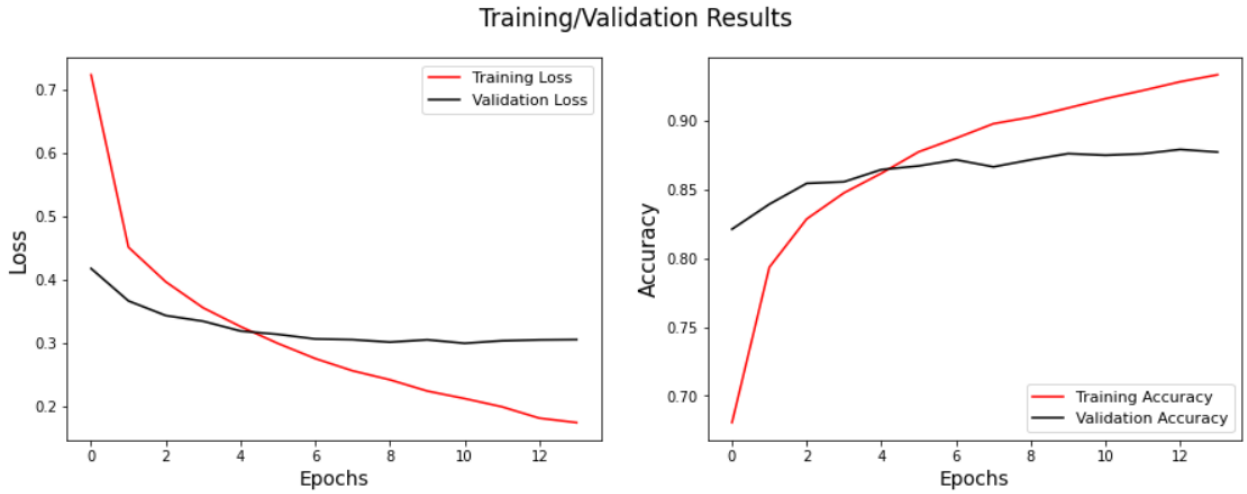


Figure 3

The scores using the development and test set were improved! In particular, the model yielded an Accuracy score of 87.4% on the development set and 88.3% on the test set. The rest of the metrics can be found in table 2. Although the scores were better than the first model, the model did not outperform the baseline!

[3] https://radimrehurek.com/gensim/models/word2vec.html
[4] The dropout rate before the output layer was increased to 0.7.

Metrics	Training Set	Development Set	Test Set
Accuracy	0.979629	0.874952	0.883333
Recall0	0.974399	0.848295	0.868142
Recall1	0.984835	0.901916	0.898661
Recall_macro	0.979617	0.875106	0.883401
Precision0	0.984606	0.897415	0.8963
Precision1	0.974776	0.854602	0.871051
Precision_macro	0.979691	0.876008	0.883676
F1_0	0.979476	0.872164	0.881996
F1_1	0.979779	0.877621	0.884641
F1_macro	0.979627	0.874893	0.883318
AUC0	0.998113	0.950271	0.94931
AUC1	0.998106	0.944855	0.945688
AUC_macro	0.998106	0.944855	0.945688

Table 2: Scores yielded by the second Model

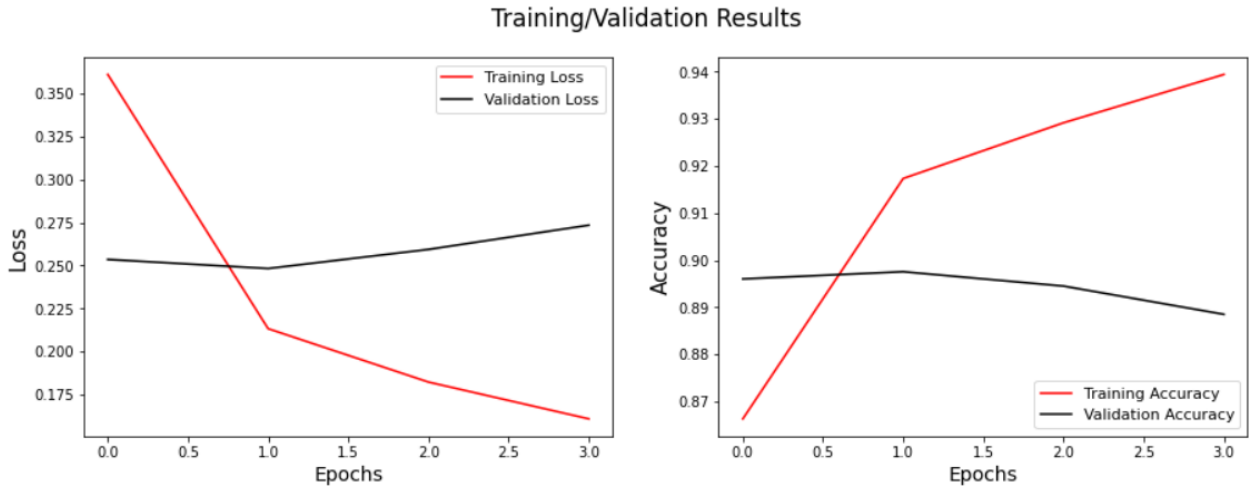
TF-IDF VECTORS

The final model was created by using tf idf vectors in order to transform the reviews. In order to find the best combinations of both the vectorizer and the model itself the keras tuner was used. Starting with the vectorizer, the best values for its parameters were determined to be 7000 feature vectors, using both unigrams and bigrams. The use of log(TF) + 1 values further increased the final score. These parameters were not determined by the tuner itself but rather, manually after examining the results of the tuner having created the model.

In the next step, before using the tuner, a function was created in order to run several models with the help of the tuner, who picks the parameters randomly from a range of values. In other words, this function returns a model with the parameters the tuner picks as if it was an actual neural network model. In order for the function to be used, a tuner was initialized. The parameters to be tuned include the number of layers and the number of neurons in each one along with their respective activation function, the use of dropout and the learning rate. The tuner was set to search over 1000 epochs (with the use of early stopping) by using both the train and the developments sets.

The tuner was run multiple times and yield deferent results in each run. A sample run is presented, though a different one is used as it proved to be more accurate. Both models stem from the tuner.

Using these parameters yielded an accuracy score of 0.905 which is an improvement considering our baseline. It's also important to state that the specific model took less time in order to be trained. We can see the resulting loss and accuracy plots below. The best combination of parameters is computed during the second epoch!



Comparing the scores with the previous models we come to the conclusion that differences are present. Specifically, the training set scores are slightly worse which implies that the model is less prone to overfitting. At the same time the test scores are better, increasing the accuracy by up to 3%. This means that the use of TF-IDF vectors outperforms the use of word embeddings as well as our baseline.

Table 3: Scores yielded by the third Model

Metrics	Training Set	Development Set	Test Set
Accuracy	0.9422	0.897524	0.905556
Recall0	0.939633	0.889773	0.906637
Recall1	0.944755	0.905364	0.904464
Recall_macro	0.942194	0.897568	0.905551
Precision0	0.94423	0.904854	0.905435
Precision1	0.940199	0.890354	0.905677
Precision_macro	0.942214	0.897604	0.905556
F1_0	0.941926	0.89725	0.906036
F1_1	0.942471	0.897796	0.90507
F1_macro	0.942199	0.897523	0.905553
AUC0	0.985374	0.964151	0.966567
AUC1	0.985039	0.960539	0.961272
AUC_macro	0.985039	0.960539	0.961272

Conclusion

In a nutshell, the use of simple neural networks proved to outperform the baseline model both in terms of training time and accuracy. The use of the tuner definitely played an important part in finding the optimal model as it allowed to test different combinations with minimal manual work thus saving time. The use of TF-IDF vectors also proved to be better in terms of accuracy but this doesn't imply that this will be the case in every problem. Given more time to train the embedding models or to fine tune them the results may have been different.

Finally, further inspection needs to take place in order to assert that the reviews containing the word "not" pose a problem for our prediction models. In the previous project we explored the differences the word "not" creates in terms of predictability. More specifically bigrams such as "not good" were regarded as 2 unigrams, emphasizing on the word "good" and thus reducing the importance of the word "not". We cannot be sure that even the use of bigrams as is the case in model 3 solved the problem.

Code:

https://colab.research.google.com/drive/1ZYOTvPIBhm1aRyzsmn_-EHPzC44JchVk?usp=sharing