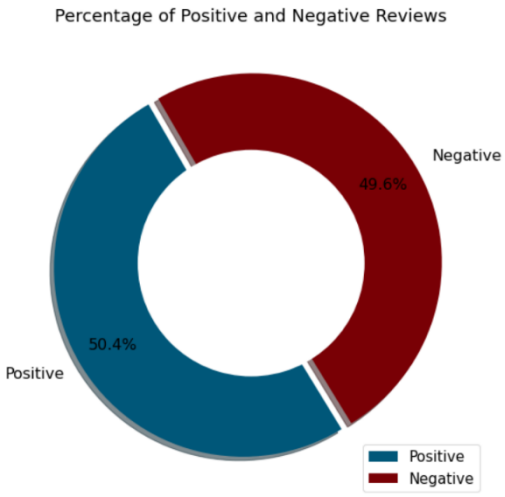
Introduction

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, two RNNs were trained; one using pre-trained word embeddings and one using pretrained word embeddings along with character embeddings as its input. However, they performed poorly compared to the baselines, both in accuracy and training time.

Dataset

Figure 1



The dataset that was used contains 10,000 highly polar movie reviews[1]. After the stop words were removed 51,452 unique words were left. Furthermore, there were not any imbalances in the dataset. That is, 50.4% of the reviews were positive and 49.6% 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 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 length that was chosen was the average number of words in a review, plus their standard deviation (210).

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

An embedding matrix was also created using fasttext's pretrained 300-dimensional word embeddings[2]. If for a word, a word embedding could not be found, the word was iterated backwards (until the length of 3) and the embedding for the corresponding n-gram was assigned to it.

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 (30%) and a test set (70%).

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 and a Logistic Regression were used.

In order to produce the initial RNN model a Bayesian Tuner was used. This way, the tuning process is completed sooner compared to the Random Tuner due to its underlying logic. Furthermore, because of this logic the model produced is more trustworthy due to the lack of randomness. The *build_lstm* method was used by the tuner to initialize models.

The tuning variables included are the number of stacked bidirectional LSTMs (range: 1-3), the variational dropout (range: 0 - 0.5) as well as the inclusion of Layer Normalization and the value of dropout (range: 0-0.5). Moreover, an Attention Layer was added and the number of hidden layers for each MLP (range: 1-3), along with their respective activation function and another dropout layer (range: 0-0.5) were tuned. The tuner then decided if a final Dense Layer would be included, picking from a range of values (0,32,64). If 0 was picked, then the Dense Layer was not included. Otherwise, the dense layer would have as many nodes as the tuner picked, along with a final layer of dropout (range: 0-0.5). Finally, the learning rate used in the Adam optimizer was tuned (range: 1e-4, 1e-2).

Having declared the method, the tuner ran three times in order to produce an appropriate model. By no means is this an acceptable number of trials, but due to the

[2] https://fasttext.cc/docs/en/english-vectors.html

limited time and in order to produce a result it was left as such. Given more time and resources, the end result would have been better. The model produced by the tuner has the following values for its hyperparameters:

```
Trial summary
Hyperparameters:
dropout 1: 0.4689039868634631
number of LSTM layers: 2
LSTM nodes: 200
variotonal dropout lstm: 0.36401422041453746
Normalization ?: False
dropout 2: 0.32162835578120347
Attention Layers ?: 4
Attention activation function ?: relu
dropout 3: 0.41793780561010696
Extra Dense nodes?: 0
dropout 4?: 0.23234270169676097
learning rate: 0.0020312059283255943
```

The model produced with the above hyperparameters was trained for 18 epochs using early stopping and yielded 86% accuracy on the test set.

The plots regarding the train and validation results over the epochs are the following:

The model could not outperform neither the simple logistic regression nor the MLP of the previous assignment, both in terms of accuracy and training time. As previously mentioned though, given enough time and recourses, the hyperparameter tuning could have been more extensive and thus, the model produced by the tuner would have been drastically better.

The score tables of the RNN and baselines are presented below:

Table 1: Logistic regression scores

Metrics	Training set	Development set	Test Set
Accuracy	0.907429	0.857778	0.86762
Recall0	0.882794	0.827189	0.84572
Recall1	0.931561	0.886266	0.89007
Recall_macro	0.907178	0.856728	0.86789
Precision0	0.926667	0.871359	0.88746
Precision1	0.89027	0.846311	0.84913
Precision_macro	0.908468	0.858835	0.8683
F1_0	0.904199	0.8487	0.86609
F1_1	0.910448	0.865828	0.86912
F1_macro	0.907323	0.857264	0.8676
AUC0	0.971324	0.938468	0.9496
AUC1	0.972144	0.942915	0.9448
AUC_macro	0.972144	0.942915	0.9448

Table 2: Feed Forward MLP using TF-IDF vectors

Metrics	Training set	Development set	Test Set
Accuracy	0.976714	0.86	0.88905
Recall0	0.982102	0.880184	0.90216
Recall1	0.971437	0.841202	0.8756
Recall_macro	0.976769	0.860693	0.88888
Precision0	0.971168	0.837719	0.88143
Precision1	0.982271	0.882883	0.89723
Precision_macro	0.976719	0.860301	0.88933
F1_0	0.976604	0.858427	0.89168
F1_1	0.976824	0.861538	0.88629
F1_macro	0.976714	0.859983	0.88898
AUC0	0.997236	0.948438	0.96073
AUC1	0.997405	0.954332	0.95214
AUC_macro	0.997405	0.954332	0.95214

Table 3: Stacked Bidirectional LSTMS using pretrained word embeddings

Metrics	Training set	Development set	Test Set
Accuracy	0.913429	0.853333	0.86667
Recall0	0.887991	0.817972	0.84196
Recall1	0.938348	0.886266	0.892
Recall_macro	0.91317	0.852119	0.86698
Precision0	0.933819	0.870098	0.88878
Precision1	0.895305	0.839431	0.8463
Precision_macro	0.914562	0.854764	0.86754
F1_0	0.910328	0.84323	0.86473
F1_1	0.916321	0.862213	0.86855
F1_macro	0.913325	0.852722	0.86664
AUC0	0.972393	0.938852	0.94971
AUC1	0.973313	0.940421	0.93856
AUC_macro	0.973313	0.940421	0.93856

In an attempt to improve the RNN model, character embeddings were also used. Each review was split into words and each word was split into characters. The character length for each word was set to 12, truncating or padding words when necessary. Each character was assigned a unique index and then each review was transformed to a sequence of those indices.

The character embeddings were created using a bidirectional LSTM in a Time Distributed Layer. The extra dimension of the characters (12) was treated as timesteps and thus, the character embeddings for the whole words were given as a second input along with the word embedding.

The results, however, did not improve. That was expected since the model was not tuned. Given more time and resources, the model could be tuned and would probably yield better scores.

Conclusions

All in all, the lack of resources and time made the attempt to create a well-performing RNN model quite difficult. The model could not outperform even a simple Logistic Regression.

Given more time and resources, the hyperparameter tuning of the model could have been more extensive, which would probably yield better scores!

Code:

https://colab.research.google.com/drive/1wzVVZ3Q7yS_dyhdYSQ1e6PI5_8VekraN?usp=sharing