

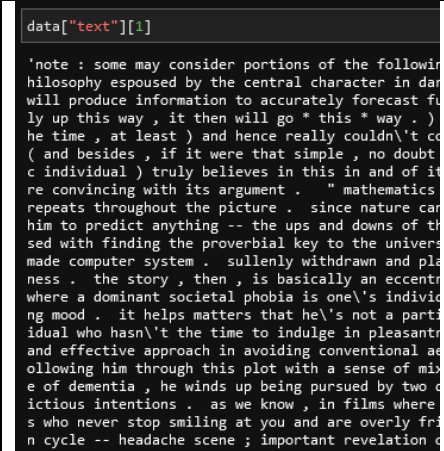
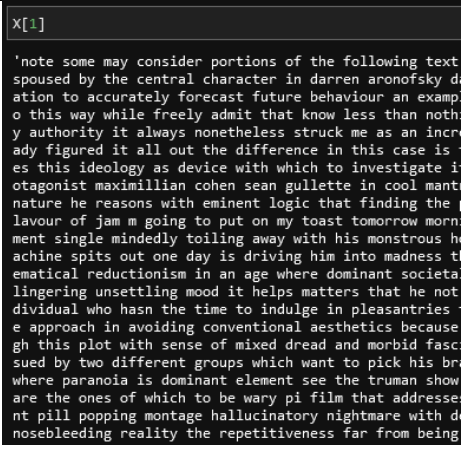
Data Wrangling Coursework 2 – Natural Language Processing and Neural Networks

40340711 – Krzysztof Dworczyk

Introduction (Data Cleaning and Preparation)

First step in any good machine learning process is good data preparation. Our data is movie reviews made up of 2 columns which I will refer to as “text” for the first column, and “label” as the sentiment of the review which correspond to either 1 being positive or 0 being a negative review. I load the train.csv and test.csv files into variables to be processed separately.

First, we remove any punctuation and numbers from the text. Since each review consists of a string containing the full review, we can simply use regex to remove any punctuations and join the words back together using whitespace. We then remove any single characters that may have been left due to punctuation being removed from plural words.

	
<i>Figure 1 - Before pre-processing the text</i>	<i>Figure 2- after pre-processing the text</i>

Once we have our data cleaned and ready, we apply the keras Tokenizer which will split the character by whitespace and lowercase them all for us. We can then fit the text onto the tokenizer and prepare it for the embeddings.

Text Representation (Word Embeddings)

Before we can feed our learning algorithm any type of data, we need to choose a representation for it. We are dealing with texts so have a couple options such as bag of words or simply using bigrams and trigrams. If we used Bag of Words, then relationships between words and tokens are ignored or forced in bigram or other ones like trigrams. For these reasons I have chosen to go with the word embeddings which put the words into 3-dimensional spaces where every word has a real-valued vector which correspond to how closely related the words are.

To create such models takes a lot of computer power so for that reason I have gone with a pre-trained model approach like GloVe with 300 dimensions for representing the words. For the CNN to work, all the inputs need to be the same sequence length, so if our max sequence or sentence length is 400, then every other input will need to have some padding added to it so that the final length is correct for every trainable sentence.

Machine Learning Model (Convolutional Neural Network)

A big argument for CNNs is that they are extremely fast with large datasets. Our data is around 1.4 thousand rows which should be very easy for our CNN to work through. The convolutions learn patterns we do not even have to explicitly program. The convolutions help in extracting more features and condensing the data through the layers into a prediction.

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 400, 300)	10148400
conv1d (Conv1D)	(None, 395, 200)	360200
global_max_pooling1d (Global	(None, 200)	0
flatten (Flatten)	(None, 200)	0
dense (Dense)	(None, 10)	2010
dense_1 (Dense)	(None, 1)	11

```

Total params: 10,510,621
Trainable params: 10,510,621
Non-trainable params: 0

```

Figure 7 – Summary of our Model

In Figure 7 we can see a summary generated for our model. The first layer is our embedding layer which contains 10,148,400 parameters, this is because we have a vocabulary size 33,828 which we represent as a 300 dimensional array, therefore we will have 3,382,800 trainable parameters. We then have our convolutional layer which results in 395 sentences length and we apply 200 filters to it. Lastly, we have our GlobalMaxPooling layer which reduces the dimensions complexity and keeps the significant information of the convolution. Followed by 2 dense layers where we finally arrive at our output.

Hyperparameter tuning

To help find the best parameters to use in the model I do a randomized search with a dictionary of specified parameters. The parameters can be seen in the table below.

Parameter	Value
num_filters	32, 64, 128
kernel_size	3, 5, 7
epochs	10, 20, 30

We will be utilising the *RandomizedSearchCV* from the scikit library, this will allow us to perform a search on the parameters specified.

The val.csv dataset is loaded in to perform the search instead of using the already existing test and train data. This will help in limiting the overfitting of the model as we will be testing the parameters on data never trained on.

```
Fitting 4 folds for each of 5 candidates, totalling 20 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

Figure 4

```
[Parallel(n_jobs=1)]: Done 20 out of 20 | elapsed: 3.7min finished
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7f8b1c1c1c1c>
he excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop,
(1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental
For (3), please refer to https://www.tensorflow.org/guide/function#controlling\_retracing and https://www.tensorflow.org/guide/function#controlling\_retracing
Running source data set
Best Accuracy : 0.5466
{'vocab_size': 33828, 'num_filters': 32, 'maxlen': 400, 'kernel_size': 7, 'embedding_dim': 100}
Test Accuracy : 0.6000
```

Figure 5

Figure 4 and 5 is the results of running the random search on the parameters, we are being given ones which resulted in the best performances of the model.

Results & Evaluation

Results

The best accuracy I managed to achieve can be seen in Figure 5 below which is **84%**.

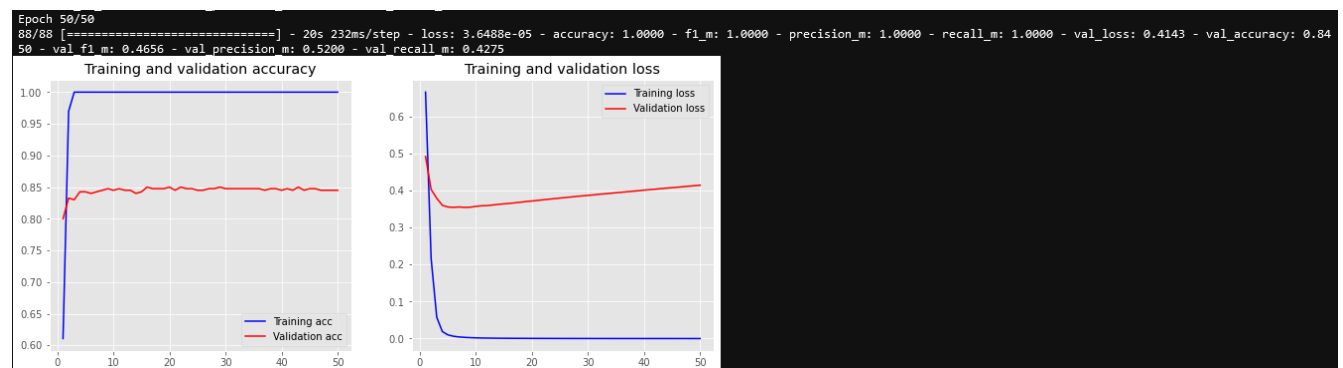


Figure 6 – Graphs showing the training statistics

From Figure 6 we can see that our model does a lot of learning before it gets to epoch 10, then it slows down on learning until we basically get stuck in the 80% range. We can also see that the Training loss decreases with the epochs which means the model becomes better at training which corresponds to less erratic learning.

Evaluation Metrics

For the evaluation metrics I have gone with 4 different ones. Accuracy, Precision and Recall are their own calculations where as F1_Score included both precision and recall together to get a better average. The table below shows the results of each metric being run on our model with the test data.

Three function had been created `recall_m`, `precision_m` and `f1_m`. These will be used as metrics in the model as Keras does not provide any out of the box evaluations.

Metric	Result
<i>Accuracy</i>	84
<i>F1_Score</i>	46.69%
<i>Precision</i>	52%
<i>Recall</i>	43%

The first metric is simply the accuracy of the model and how many true positive we managed to predict.

With precision we are looking to see how many of the predicted trues are actually true.

The recall metric lets us see what proportion of actual positives was identified correctly.

The F1 score gives us a balance between the precision and recall results and avoid any biases either of them might have. Since our precision and recall score weren't above 60% then we expect the recall to be also low.

Reflection

The overall goal has been achieved which was to design and implement an algorithm which would help in predicting on our dataset. The model chosen was a Convolutional Neural Network which took sentences as sequences. An embedding layer was added to help in representing the words and adding more meaning to the context of them. This allowed the final model to produce an accuracy of around 84%.

Further work which could have been done in my opinion to perhaps further help improve the accuracy is to add more convolutional layers to the model. This would allow for more features to be extracted which would increase the overall prediction accuracy. More parameter tuning could have been done with different parameters like epoch or batch size.