IMDB Sentiment Analysis

The aim of this project was to train different neural network architectures to perform sentiment analysis on movie reviews. A quick summary of the results can be seen in Table 1.

Table 1

Name of model	Training accuracy	Validation accuracy
Shallow Feed-Forward Neural Network	90.47 %	82.84 %
Shallow Convolutional Neural Network	90.56 %	89.05 %
Residual Neural Network	97.66 %	88.84 %
Multi-Branch Convolutional Neural Network	97.57 %	89.41 %

Data set

The data set was obtained from [1]. It contains 25,000 highly polar movie reviews, half of them good and the other half bad. The data set consists of just two columns: the review text and the appertaining sentiment.

Cleaning the Texts

On inspection, it appeared that some common stop words had already been removed from the texts in the data set. The further cleaning of the texts proceeded as described in the following. First, all letters were set to be lower case. Then instances of, e.g., “isn’t” was replaced by “is not”. See the full list of such replacements in Table 2.

Table 2

Before:	**After:**
isn’t	is not
aren’t	are not
wasn’t	was not
weren’t	were not
didn’t	did not
haven’t	have not
hasn’t	has not
won’t	will not

wouldn't	would not
doesn’t	does not
can’t	can not
shouldn't	should not
mightn't	might not
mustn’t	must not

Lastly, all punctuation was removed.

After this cleaning, all texts that were below 100 characters of length as well as all texts that had a length of more than 2000 characters were omitted. Texts of such lengths were clear outliers, as can be seen in Figure 1, and were removed to avoid the need of excessive padding in later steps.

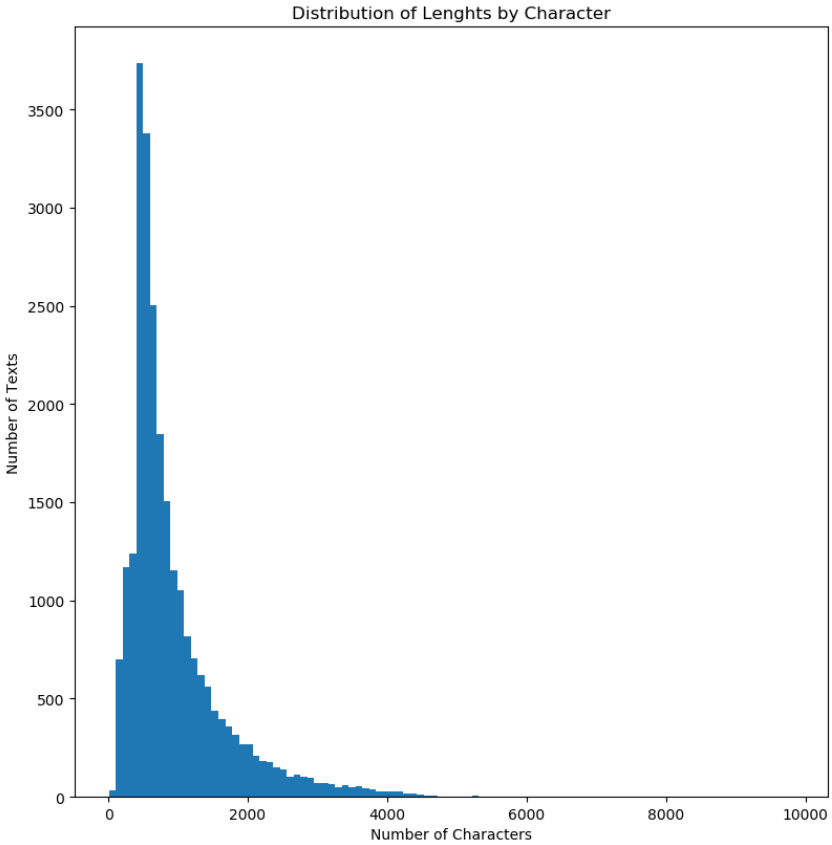

Figure 1

GloVe

The texts were made into inputs by the use of the pre-trained GloVe word vectors acquired from [2]. The word vectors used here were trained on the Common Crawl data set. This set of GloVe word vectors consists of approximately 2.2 million row vectors, each of which has a length of 300. The Euclidean distance between two words vectors can provide a measure for semantic similarity of the words that the vectors represent. In other words, these word vectors provide a way of capturing meanings of words in a form that can be used as a in input for neural networks.

Train-test-split and Inputs

The data set was divided into a training set, a validation set and a testing set. 80 % of the data points was allocated to training, and the remaining data points were split evenly into validation and testing sets.

There are two requirements for the data, if it is to be used as input to neural networks: 1) the input must be expressed in terms of numeric values, and 2) all data points must be of the same size.

In order to efficiently convert the texts to inputs to the neural networks described below, each word in the vocabulary of the training set represented with a unique integer. These integers were used to create a so called ‘embedding matrix’. The embedding matrix had 90461 rows, one for each word in the training set’s vocabulary, and 300 columns, to fit the dimensionality of the word vectors. For each integer, the corresponding word was searched for in the in the GloVe vocabulary. If the word was found, the row number corresponding to the integer was set to be equal to the GloVe word vector of the word. If the word was not found, the corresponding row in the embedding matrix was simply set to be all zeros.

To make the texts into suitable inputs, all words in the texts were replaced by their corresponding integer, such that each “text” was now a sequence of integers. These sequences were made to have equal lengths by padding all sequences shorter than 300 integers with zeros in the beginning of the sequence. For a demonstration, see Figure 2.

Now that the data points were represented with numbers and all had the same length, they could be fed as inputs to neural networks. Each neural network will need to have an embedding layer as its first layer. The output of the embedding layer will be a matrix of size (300,300) since each sequence

is 300 integers long and each word vector contains 300 elements. The first row of the output of the embedding layer will be the word vector taken from the row of the embedding matrix corresponding to first integer in the sequence.

Shallow Feed-Forward Neural Network

The first neural network trained on the data set was a shallow feed-forward neural network with one hidden layer of 256 neurons with the ReLU activation function. This layer had a dropout rate of 0.8 as a means to prevent rapid over-fitting. The output layer of the network consisted of two neurons activated by the softmax function. The “adam” optimizer was used to optimize the binary crossentropy loss function.

The model was trained for 50 iterations, and the “callbacks” method [3] was used, so that the model was evaluated on the validation set after each iteration. Only models that performed better than any previous models were saved. This was done to ensure that the models were saved before the over-fitting tendencies of the neural network would ruin the model’s ability to generalize to instances beyond the data set.

After having completed the training, the model that performed best on the validation set achieved a training accuracy of 90.47 % and validation accuracy of 82.84 %.

Shallow Convolutional Neural Network

This neural network consisted of a one-dimensional convolutional layer with 50 filters, a kernel size of 2, and a stride of 1. The activation function used was again the ReLU. The dropout rate for this layer was 0.75. The convolutional layer was followed by a global max-pooling layer. The output layer of this network was the same as in the feed-forward network described above.

Once again, callbacks was used. After the 50 iterations, the model that performed best on the validation set achieved a training accuracy of 90.56 % and validation accuracy of 89.05 %.

Residual Neural Network

A feed-forward residual network was implemented. This network had six hidden layers. The first layer was a convolutional layer with 100 filters, kernel size of 2, and stride of 1. The four layers were dense consisted of 256 neurons. The last hidden layer was also dense and consisted of 100 neurons. All these layers had a dropout rate of 0.8 and were activated by the ReLU function. The output of the convolutional layer was saved and added to the output of the last hidden dense layer. In this way, the output of the convolutional layer “carried over” all the way to the end of the network.

The output layer was once again the same as the output layers described above.

After the 50 iterations, the model that performed best on the validation set achieved a training accuracy of 97.66 % and validation accuracy of 88.84 %.

Multi-Branch Convolutional Neural Network

This network consisted of three parallel convolutional layers, of which the outputs were concatenated before the output layer. All three had 100 filters, and a stride of 1. Further, they were all activated by the ReLU function and had a dropout rate of 0.75. The layers only differed in their kernel sizes; one had kernel size 2, another had kernel size 3, and the last had kernel size 4.
The output layer was the same as the output layers describe in the previous models.
After the 50 iterations, the model that performed best on the validation set achieved a training accuracy of 97.57 % and validation accuracy of 89.41 %.

References:

[1] https://www.kaggle.com/oumaimahourrane/imdb-reviews
[2] https://nlp.stanford.edu/projects/glove/
[3] https://keras.io/callbacks/