
Royal Institute of Technology

DD2424

# Deep Learning

## Project for DD2424

### NLP Sentiment Classification for Movie Reviews With Numerical Review Prediction

***Students:***
Maryam Cherradi
Nathan Rougier
Angela Sun

**Abstract**

With the overabundance of movie reviews now available online, judging a movie subjectively has become a time-consuming task. From our work on sentiment classification and numerical review prediction, we tried to give a scientific context to a subjective task. Sentiment classification and scoring are two typical human jobs and are very hard to approximate by a machine. We proceeded in steps to handle the gradually increasing complexity of the two tasks : binary sentiment classification and scoring prediction on a 10-point numerical scale. For the multi-class classification, after testing several networks with different sets of parameters, we selected the best performing setting and used it to predict the scores of original reviews. The achieved results clearly indicate that the learning was efficient but the subjectivity of the task makes it very complex and therefore the obtained test accuracies are limited. Nevertheless, this work expands on much of the sentiment classification work popular in the field of natural language processing and provides foundation for further multi-class classification of typical tasks attributed to humans.

# Contents

# 1 Related Work

The core of our work was inspired from pre-designed networks found on the web. Many tutorials explore text classification and the first task of our work, binary sentiment classification, was entirely available on the web through blogs [2]. Many blogs, however, had different approaches to the construction of the layers in the networks, which were explored and compared by us [3 - 8].

The second task, on the other hand, is an original one and was only inspired from the existing literature on multi-class text classification. Several studies have been led on the topic with varying conditions, such as on comparisons between multiclass and multiple binary classifiers [9], on highly dimensional and limited labeled data [10], or on highly multiclass data [11]. However, albeit interesting, most of this work does not directly apply to our problem. Thus, we consulted more heavily from blogs and the Keras API itself to alter the binary classfiers to our needs, since our data was not particularly highly dimensional or complicated.

# 2 Data

## 2.1 Understanding data processing

For a first approach on how to use Keras, we used the IMDB dataset provided by Keras. This dataset of size 50000 (25000 for training data and 25000 for testing) is loaded as lists. The reviews are each represented as a list of integers, each integer denoting one word. The labels are stored in lists and are either 1 (positive review) or 0 (negative review).

To understand how to process data on Keras, we then downloaded the same dataset from [1] and tried to match the results with what we had obtained previously. The data is organised in folders: one for training and one for testing, within each folder, there are two different folders containing exclusively positive or negative reviews each in a separate text file. We processed the data to get the same format as the Keras IMDB dataset.

## 2.2 Webscraping our own data

For our last task, we used IMDB reviews with their corresponding score that we gathered from webscraping [5]. First, by downloading a list of all movies present on IMDB and sorting them by most-to-least number of reviews, we gathered a list of around 11,500 most viewed titles. This guaranteed that each page we visited from this list of titles gave us a sufficient amount of review data, so we would not waste time visiting pages with little data. Then, by writing a webscraping script to visit each of the review pages from those titles and storing the numerical rating (out of 10) as well as the subjective written review, we gathered almost 200,000 diverse reviews from many movies.

First, we just gathered the most relevant reviews for each movie, which was used to train our best network. We then realized that our dataset was unbalanced since most relevant reviews are positive, so we then queried for reviews from each rating category (i.e. from each rating 1 through 10) for each movie in order to have a relatively balanced data set. However, the results did not follow as expected (See section 4.6). Reviews were processed to remove new line characters, for data organizational convenience.

Data was split into datasets of around 180,000 reviews for training, and 20,000 for testing.

The data is organized in a tab delimited csv file. Each row contains one review with its score. The script can be found on the GitHub Repo (See Bottom of References).

# 3 Methods

We chose to use Keras because from our research on deep learning frameworks, it seemed the most relevant to our natural language processing work. As previously discussed, there exists a lot of research already utilizing Keras to build sentiment classification neural networks and some multi-class classifiers. In addition, we found Keras to be the most beginner-friendly and thus the most efficient for our use.

### 3.1 Understanding Keras

First, we used the IMDB dataset provided by Keras to run preliminary experiments and understand the functioning of Keras. This data can easily be found by importing 'imdb' from 'keras.datasets'. After processing this dataset and building a working binary sentiment classifier with this data, we downloaded raw data from IMDB itself to build a data processor that can convert raw csv data into a form better suited for our network (see Data section).

### 3.2 Data processing

Then, we tried to match the previous results with experiments ran on the data we downloaded from the IMDB website [1]. This helped us understand how to process the data before training our own network on them. To reproduce the shape of the Keras data, we used the Keras module "Tokenizer", which creates a new dictionary mapping the words into integers and converts the reviews into lists of integers. Specifically, we specified a vocabulary of the 20,000 most relevant words in the English language from our movie review data. After that, we follow the same procedure as with the Keras dataset to process the data : each review was either padded or truncated to the last 200 words. (See reasoning and details in Experiments)

After successfully processing the data, we tried simple models to understand how to use Keras and compared the different methods that work well for analyzing the movie reviews. For example, as explained further in Experiments, word embedding was critical for obtaining good results.

### 3.3 Multi-class Classification

Finally, using our own data collected from webscraping various top movies (around 11000 movies) from the IMDB website, we applied the knowledge gained from running tests described above on the binary-classification models to our own multi-class network classifier.

We ran several experiments (detailed in the Experiments section) in order to approach an optimal network for our webscraped data. Some experiments were run on how to most optimally measure losses and accuracy given the subjectiveness of our task.

Once we had identified a promising network, our method was to run several trials to determine which set of parameters gave the best results on that network.

# 4 Experiments

### 4.1 Preliminary testings to understand Keras' existing layers

For the simple models, we just tried to understand what could be the relevant parameters to fix in our network to do the sentiment classification. For this experiment, we chose four models. The first one is a model with just one dense layer. The second

is using the embedding size with the size 128. The third is using MaxPooling1D and a Dense layer of size 50 and 1, and the final model is using LSTM.

Because the first input to the model needs to have training data with the same dimensions, which is not the case with movie commentaries, we are using the function **pad.sequence** to fix every commentary to the same length (we choose the size 100). But it is possible to take the beginning of the commentary or the end by choosing between 'post' (beginning) and 'pre' (ending) as parameters of **pad.sequence** . As can be seen below on the graphic, we get better results by using the end of commentaries. This may be explained from a common habit where people give their honest opinion at the end.

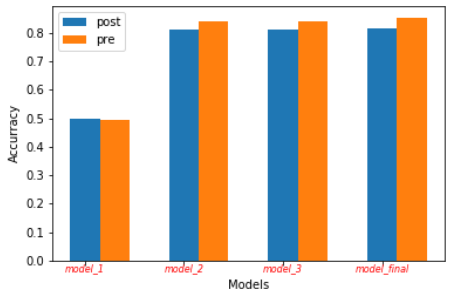


Figure 1: Comparison of the accuracy for our different model with the parameter 'pre' and 'post'.

Another relevant parameter is the word embedding. It greatly improves our results, as can be seen from the difference between the accuracy of the first model and the second. The embedding layer represents the words of our vocabulary (that are encoded as integers) as a vector, which are closer when the meanings of the words are closer. It strongly increases the performances of our networks.

Finally, we can see that we don't observe such a strong difference between the model using LSTM and the other models even if they are simpler. The reason is that our amount of data with the IMDB dataset is too small to show the advantages of LSTM.

We ran other experiments to test the relevance of other parameters (as an example, the influence of the size of MaxPooling or the use of GlobalMaxPooling), but we will not detail them here.

### 4.2 Testings for the first task

For the first task, we compared the results of the networks we found on the web. The best performing network is the one derived from this work [2]. It achieves 88 % test accuracy.

According to experiments with multiple varied implementations with LSTM, we finally concluded that often simpler networks are more accurate [3].

### 4.3 Testings for the second task

For the second task, classifying reviews according to their score, we compared several networks that we found on the internet ,[6],[7] and tried to adapt the parameters to best fit our set of data.

The first network that we tested had 3 Dense layers and achieved poor results. The dense layer is a regular fully connected layer.

```
model = Sequential()
model.add(Dense(20, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 2: First basic tested network for the scoring task

```
Epoch 1/2
169990/169990 [==============================] - 15s 88us/step - loss: 28.0670 - accuracy: 0.2628 - val_loss: 2.0693 - val_accuracy: 0.2698
Epoch 2/2
169990/169990 [==============================] - 11s 65us/step - loss: 2.0398 - accuracy: 0.2805 - val_loss: 2.0561 - val_accuracy: 0.2698
20000/20000 [==============================] - 1s 29us/step
Test score: 2.05614245170354 83
Test accuracy: 0.26980000734329224
```

Figure 3: Corresponding results

```
Epoch 1/2
169990/169990 [==============================] - 250s 1ms/step - loss: 1.5542 - accuracy: 0.4185 - val_loss: 1.5479 - val_accuracy: 0.4129
Epoch 2/2
169990/169990 [==============================] - 251s 1ms/step - loss: 1.1962 - accuracy: 0.5491 - val_loss: 1.7210 - val_accuracy: 0.3938
20000/20000 [==============================] - 1s 65us/step
Test score: 1.7209984447062014
Test accuracy: 0.39375001192092896
```

Figure 4: Corresponding results when adding an embedding layer

Once again, after adding the embedding layer the accuracy of the network rose and the losses dropped.

We then tried what has been selected as our best performing network detailed in the next subsection. We ran experiments on other networks that we found in the literature but none achieved as good and efficient results as that second network :

```
model = Sequential()
model.add(Embedding(max_features,embedding_size, input_length=maxlen))
model.add(Dropout(0.25))
model.add(Conv1D(filters, kernel_size, activation='relu'))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(Conv1D(filters, kernel_size, activation='relu'))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(Conv1D(filters, kernel_size, activation='relu'))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(Conv1D(filters, kernel_size, activation='relu'))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
model.add(LSTM(gru_node, return_sequences=True, recurrent_dropout=0.2))
model.add(LSTM(gru_node, recurrent_dropout=0.2))
model.add(Dense(32,activation='relu'))
model.add(Dense(encoder_length))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 5: Third network tested for the scoring task

```
Epoch 1/3
152991/152991 [==============================] - 458s 3ms/step - loss: 1.7488 - accuracy: 0.3410 - val_loss: 1.7132 - val_accuracy: 0.3461
Epoch 2/3
152991/152991 [==============================] - 460s 3ms/step - loss: 1.5554 - accuracy: 0.3997 - val_loss: 1.6205 - val_accuracy: 0.3753
Epoch 3/3
152991/152991 [==============================] - 465s 3ms/step - loss: 1.4559 - accuracy: 0.4319 - val_loss: 1.5963 - val_accuracy: 0.3880
20000/20000 [==============================] - 17s 870us/step
Test score: 1.591667025756836
Test accuracy: 0.383899986743927
```

Figure 6: Corresponding results

We tested other networks, but we will only expand on our best performing setting :

### 4.4 Best performing network for our experiments

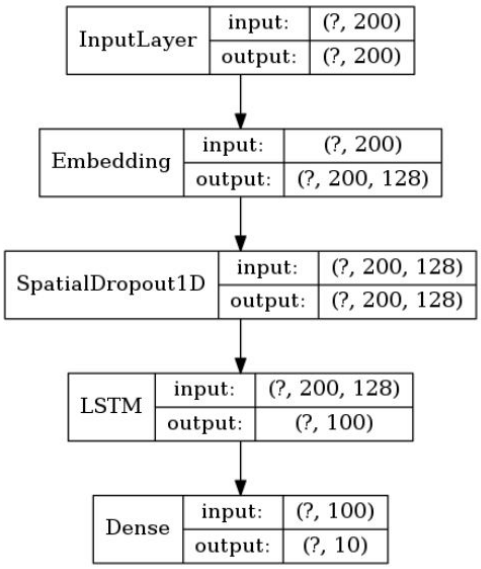


Figure 7: Best performing network for the scoring task

```
Epoch 1/3
152991/152991 [==============================] - 787s 5ms/step - loss: 1.6434 - accuracy: 0.4014 - val_loss: 1.5390 - val_accuracy: 0.4206
Epoch 2/3
152991/152991 [==============================] - 793s 5ms/step - loss: 1.4300 - accuracy: 0.4665 - val_loss: 1.5025 - val_accuracy: 0.4285
Epoch 3/3
152991/152991 [==============================] - 819s 5ms/step - loss: 1.3345 - accuracy: 0.4984 - val_loss: 1.5093 - val_accuracy: 0.4364
20000/20000 [==============================] - 26s 1ms/step
Test score: 1.5056110345840454
Test accuracy: 0.43230000138282776
```

Figure 8: Obtained losses and accuracies for this network

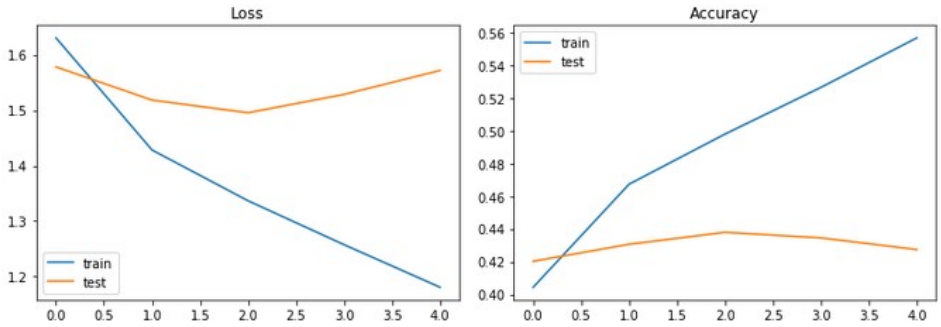


Figure 9: Obtained losses and accuracies for this network - curves

This network (derived from [6]) is composed of:

- an Embedding layer (described previously).
- a SpatialDropout1D layer that is meant to avoid over-fitting. The purpose is to switch some neurons off at each epoch with a probability that is a specified parameter.
- an LSTM layer (Long Short Term Memory) which is used in a recurrent neural network that adresses the vanishing gradient problem (two related words may be separated by a collection of useless words)
- a Dense layer (described previously) with a softmax activation to get normalized values and be able to select the prediction with the higher weight

Overall, this network is efficient for multi-class text classification because it is simple and uses the two important layers Embedding and LSTM that are particularly well suited for this task.

The parameters that we used are the following: embedding size = 128, dropout factor = 0.2, LSTM layer's number of nodes = 100, batch size = 64, number of epochs = 3, maximum length of a review = 100, size of the vocabulary = 20 000, optimizer = ADAM, loss calculation = categorical_crossentropy.

When increasing the embedding size (after 100) the results were better but the code took longer to run. A value of 128 is a good compromise between good results and time-consuming calculation. Above 200, the code takes too long to run and the accuracy values decrease. After 3 epochs of learning, the network seems to be over-fitting so we stopped training there.

### 4.5 Predicting the score of a review

Finally, we ran experiments to predict the score of a movie on our own faux "reviews" with the best performing network. Although the obtained results are subjective, they clearly indicate that the learning was efficient.

| Review | Predicted Rating |
|---|---|
| 'Oh my God ! This movie was awesome, just perfect, I could not think of anything better. All the scenes were just so nice to watch' | 10/10 |
| 'What a stupid movie. I really did not like it, it was very bad and the director did the worse job of his whole career' | 1/10 |
| 'I am quite mitigated, I don't know what to think of that movie. It was good and bad at the same time. Surprising and predictible.' | 8/10 |

It was also interesting to understand the importance of the words used, as well as their order and weight in the review:

| Review | Predicted Rating |
|---|---|
| 'bad bad good good good' | 1/10 |
| 'good good good bad bad' | 7/10 |
| 'bad good good good good good' | 8/10 |

From these experiments, we notice that the predicted scores are very coarse for negative reviews. This may be related to the fact that our dataset contained a majority of positive reviews that obtained high scores.

### 4.6 Further testing

We ran several experiments in order to improve the performances of our selected network:

- **Changing the loss and accuracy calculation:** We first looked at this scoring task as a categorization task and used Keras' categorical_crossentropy (with one-hot encoding of the labels) or sparse_categorical_crossentropy (without one-hot encoding of the labels) for the loss calculation. But, to best interpret our model's results, we tried to change the accuracy calculation and implemented our own metrics [8]. Given the subjectivity of a review score, we considered that the prediction was correct if it gave the true score plus or minus one. This experiment did not lead to convincing results.

  But, this scoring task can also be seen as a regression since the score increases gradually with the quality of the movie (a 9/10 review is closer to a 10/10 review than to a 2/10 review). We therefore changed the loss calculation method to adapt it to this vision. Although we observed that the training was efficient with a mean_square_error loss, it was not as effective as with the categorization loss.

- **Using a pre-trained embedding layer:** We replaced the embedding layer by a pre-trained embedding [12]. The 100 dimensional vectors that represented each word already had weights that reported on their meaning. First we incorporated this layer without training it any further. This made the calculations faster but slightly lessened the test accuracy. Even when training the layer (the pre-trained weights were just an initialization for the embedding), the results did not get better. A possible reason for this method not to be efficient may be that the pre-trained models we found on the web covered a very big vocabulary (400 000 words) compared to ours (20 000 words). The most relevant words to characterize the score of a movie review boil down to a much smaller set that do not require such a big and heavy artillery.

- **Using more data:** Finally, we webscrapped more data because we noticed that our best performing network was over-fitting. We increased the volume of our data from almost 200 000 reviews to more than 400 000 reviews. This led to poorer results even when training longer. A possible explanation may reside in the fact that for the first dataset we only picked the most relevant reviews for a given movie. With the extended dataset, for each movie, we gathered around 10 reviews for each possible score and inevitably picked some irrelevant reviews. This could have disturbed the data and led to poorer results.

## 5 Conclusion

The structure for our best performing multi-class network is simple but efficient. We searched for the best performing set of parameters and eventually got decent results. The final test accuracy achieved is : 43%, which can definitely be improved but has to be put into perspective with regard to the difficulty of this subjective task.

Some reviews are able to be accurately classified, but there is still some work needed for higher accuracy. An exact classification is very difficult given the subjectivity of the task.

Particularly, we have discussed on what could be enhanced in our methods and experiments but also on the difficulty of the task:

- How big of a challenge is the subjectivity of ratings? What we are trying to do may therefore be too hard to achieve and no better accuracy is to be expected. .

- Are we using the proper networks or is there a completely different structure that can be much more efficient? Would testing more complex networks with an adjustable number of layers be efficient?
- Is there a better way to process the data? How far can we go with bigger data?
- How far are we from a possible optimal solution?

Our work can be continued in many ways, but for now, we see that often times simpler networks do perform better, and that classification is possible even with highly subjective data.

# References


[1] Maas, Andrew. "Large Movie Review Dataset." Sentiment Analysis,

`ai.stanford.edu/~amaas/data/sentiment/.`

[2] https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/
[3] https://keras.io/
[4] https://machinelearningmastery.com/
[5] https://www.dataquest.io/blog/web-scraping-beautifulsoup/
[6] https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17
[7] https://medium.com/datadriveninvestor/deep-learning-techniques-for-text-classification-9392ca9492c7
[8] https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/
[9] Sánchez-Maroño, Noelia Alonso-Betanzos, Amparo Garcia-Gonzalez, Pablo Bolón-Canedo, Verónica. (2010). Multiclass classifiers vs multiple binary classifiers using filters for feature selection. The 2010 International Joint Conference on Neural Networks (IJCNN). 1-8. 10.1109/IJCNN.2010.5596567.
[10] Rochac, Juan F. Ramirez, et al. "A Gaussian Data Augmentation Technique on Highly Dimensional, Limited Labeled Data for Multiclass Classification Using Deep Learning." 2019 Tenth International Conference on Intelligent Control and Information Processing (ICICIP), 2019, doi:10.1109/icicip47338.2019.9012197.
[11] Grünwald, Adam. "APPLICATIONS OF DEEP LEARNING IN TEXT CLASSIFICATION FOR HIGHLY MULTICLASS DATA."
[12] https://nlp.stanford.edu/projects/glove/
2019, https://uu.diva-portal.org/smash/get/diva2:1323153/FULLTEXT01.pdf.


Source Code: https://github.com/angelasun2/DeepLearningMovieClassification