Machine Learning course project report

Movie synopsis classification

Text classification with CNN vs RNN

José Miguel Cano Santín

1. Introduction

This project has two goals. The main one is to compare a Recurrent Neural Network (RNN) and a Convolutional Neural Network (CNN) for multi-class short text classification task and see how well they perform.

CNN are specially good to detect and capture patterns and features of multiple sizes —in NLP: adjacent words or n-grams— through convolutional filters. Otherwise, RNN are specially good to handle sequences of any length and deal with short and long term dependencies, specially Long Short-Term Memory (LSTM) models, using an structure that loops to keep history of previous time steps and build the new ones taking into account that history.

There are already some experiments —the links are available in the references section— that aboard the comparison between different types of NN for text classification. Those experiments are the basis for this one and also have served as an inspiration to build the CNN models, as I already knew the basis of the designing of RNN models. Since my experience designing CNN was almost zero before starting my project, I have taken those existing models while I put effort in understanding them and adapting to this project. Even if my models are not optimal, I can say that I have learn a lot about how to design CNN for NLP tasks.

To observe the differences in performance, the main criteria is to compare the loss and accuracy of the training and validation data —through plotting the history of the model— and evaluating those same metrics on a test dataset. The models are trained in different data sizes —corresponding to 70%, 50% and 30% of the whole dataset— to see how that affects to the performance of both RNN and CNN models. I also keep track of training time. It is expected that CNN outperforms the LSTM model in this regard, but it will be good for learning purposes to actually observe how big is the difference.

The second objective of the project is to observe if it is possible to predict automatically movie genres based on the synopsis with a language model. Since a movie synopsis is a summary of the major premises of the plot, it is expected that the reader can infer to some degree the genre/s of the movie. For instance, looking at the following synopsis:

> An assassin is shot at the altar by her ruthless employer, Bill and other members of their assassination circle – but 'The Bride' lives to plot her vengeance. Setting out for some payback, she makes a death list and hunts down those who wronged her, saving Bill for last.

Assassins, shots, vengeance, death lists... Looking at those features, any reader would expect an action movie about criminals for sure. “Crime” and “action” are the genres tagged in our dataset for the movie *Kill Bill: Vol. 1*, which corresponds to this synopsis.

Then, it would be good to see if a trained language model is also capable of infer those genres based on synopsis information. Of course, not all the synopsis are as evident as the one above and our dataset may be too small to train a decently enough model for this task. This will be evaluated in the last sections of this report through observation of predictions on real synopsis.

2. The dataset

The dataset employed to train and evaluate the models has been taken from the Kaggle website[1]. The file employed —AllMoviesDetailsCleaned.csv— is a collection of 329,045 movies and their respective metadata taken from The Movie Database website.

As stated in the introduction, the synopsis and the genres are the only data that are used for this project. The format of both fields is as following:

- Synopsis.

> 'We come in peace' is not what those green men from Mars mean when they invade our planet, armed with irresistible weapons and a cruel sense of humor. This star studded cast must play victim to the alien’s fun and games in this comedy homage to science fiction films of the '50s and '60s.

- Genres.

[1] https://www.kaggle.com/stephanerappeneau/350-000-movies-from-themoviedborg#AllMoviesDetailsCleaned.csv

Comedy|Fantasy|Science Fiction

As can be observed, the format of the data is easy to treat: the synopsis are just plain text and the genres are very easy to split. Another thing very good for this project is that the genres classification is very simple with only 20 unique genres in all. This makes the classification task very efficient since the categories are not too disperse. The genres are:

> Action, Adventure, Animation, Comedy, Crime, Documentary, Drama, Family, Fantasy, Foreign, History, Horror, Music, Mystery, Romance, Science Fiction, TV Movie, Thriller, War, Western.

3. Preprocessing

What I have detailed in section 2 can be called the "bright side" of the dataset. But, when arriving to the preprocessing, soon it is clear that not all of it is ideal.

Firstly, there are a lot of movies without genres and/or synopsis. They have to be removed since they are not useful for training or evaluating our supervised model, but they can be used later to test the genre prediction.

Secondly, there are a good amount of synopses with non informative or non relevant content such as: "no overview", "not available" or "third movie in series". These texts do not talk about the movie plot, so they have to be removed. Since almost all of them are five words long or less, I have removed all movies with synopsis which are that short.

Dropping all those cases reduces a lot the number of movies to train the model. Now there are 171,892 movies, which are almost half of the original dataset.

The first step to process the synopses left is deciding how long will be the inputs of the model. The mean length of the synopses is about 65 words, but there are a lot of very short ones and a lot of them that exceed the 200 words. So, the standard deviation is too big to just input the synopsis padded to the one with the maximum length.

Therefore, the first decision has been splitting the synopses into sentences using the NLTK sent_tokenize function. After this, the length mean has gone down to 22 words, but there are still a good amount of sentences with 100 or more words. It is better, but since the difference is too big, I decided that is better to split those sentences into chunks of 50 words. This length is enough to be able to input a whole short synopsis when evaluating the prediction of our models and the difference between the shorter chunks are the longer ones is not that big.

After splitting the texts, I have decided to lowercase and tokenize the words using the NLTK word_tokenize. Despite being quite slow, it keeps the relevant punctuation attached to the words —for instance: “co-worker” ,“ 're”, “ 've”, etc.—. As for the rest of the punctuation, it keeps them as individual words, so they are easy to delete.

After all the tokenization, there are a lot of sentences left that are empty or only have one element. I have deleted all those sentences since almost all of the one-word ones only consisted of URLs, numbers, dates, times, isolated punctuation or non relevant words like “imdb” or “wikipedia”.

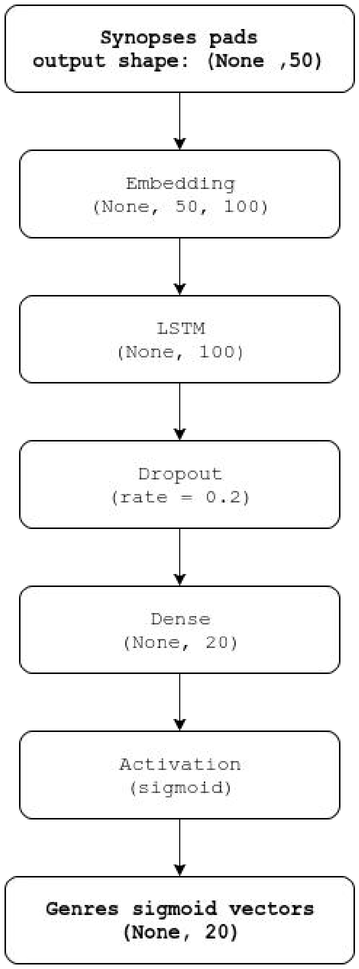
Then, the list of chunks and genres is randomized and split into three different training sizes: 70% —330,942 chunks—, 50% —236,387 chunks— and 30% —141,832 chunks— of the entire dataset. With the rest of the data I build a validation split —10%— and a test split —20%—. Those splits are saved in pickle files inside the “splits/” folder to be able to use the same data for all the trained models and test always on the same testing split.

After this process, the only things left are building a word index —using the Keras Tokenizer class with a number of words of 50,000 out of almost 200,000—, transform the words into sequences and pad all of them to 50 words, which is the length of the chunks. This will be the input of all of the models of this project.

The output preprocessing is way more simple. First, I build a sorted list with the 20 genres. Then, I use the indexes of that list to transform the genres of each caption into a vector of 0 and 1 values. For instance, if a synopsis has the genres “action”, “mystery” and “thriller”, the output vector will look as following:

[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0]

4. Recurrent Neural Network design



For the RNN I have decided to build a Long Short-Term Memory (LSTM) based model because it can deal very well with learning long term dependencies, which is the limitation of other RNN layers.

As can be observed in the diagram on the right, the design only has a simple LSTM layer. I decided to keep it simple because the training data does not seem big enough to add more depth to the model with consecutive LSTM layers. As I have observed when evaluating the assignment 2, a RNN model for text classification does not seem to perform better with more LSTM layers for a small amount of data, as it is the case of this model.

After the LSTM layer there is a dropout with the default 0.2 rate to help prevent over-fitting and a dense layer with a size of 20 to represent the number of classes of the output, which is determined by a Sigmoid function.

5. Convolutional Neural Network design

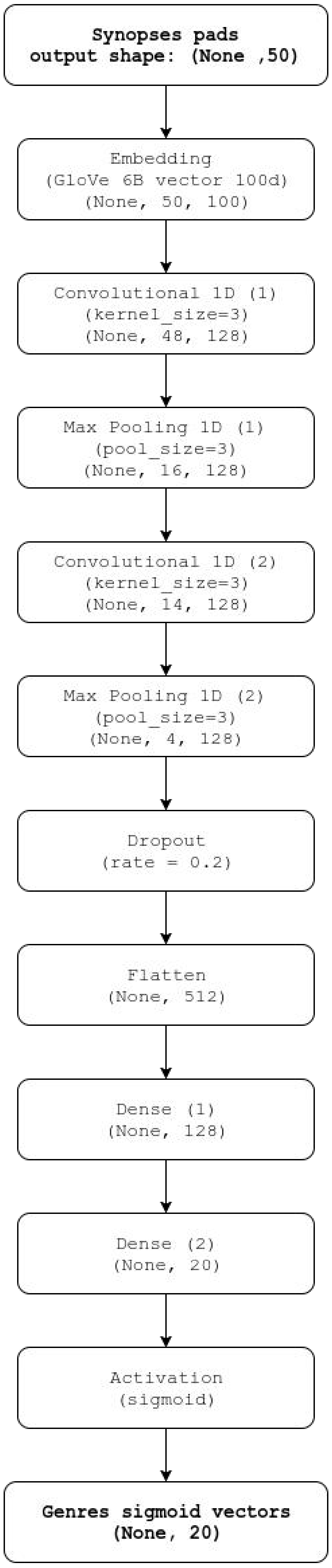
Since my previous experience building CNN was almost zero and they take very short to train, I have tried two different trees with two different filters value each in order to understand and learn how the designing of CNN for NLP works at least at a basic level. Therefore, in all I have done four different CNN trains —each with three different models for the different data sizes— inside the path *models/cnn/*. Those two architectures have been taken from two of the works linked in the References section which I will specify later.

In order to keep this section more organized, I will only detail the configuration that has given the best results. The rest are at the end of this report as an appendix. The best performing design corresponds to the models inside *models/cnn/1-128/*.

The architecture and the embeddings function are based on a tutorial available on the Keras blog[2]. The model inputs 50-word long pads and processes them through an embedding layer that map the words to a set of pre-trained GloVe embeddings —Glove 6B vector 100d—.

After the embeddings, the model consists on two sets with a convolutional layer and a max pooling layer each. The number of convolutional filters is 128 with a kernel size of 3. After the second pooling, there is a dropout layer with the default rate to prevent over-fitting. Then I flatten and I use two dense layers to reduce the size to 20, which represents the number of classes of the output. The activation output is again determined by a Sigmoid function.

I have also tried to change the convolutional filters to 256 to see the difference, but the performance was a bit worse. The evaluation results can be seen in the appendix.



[2] CHOLLET, F. (2016). Title and URL are available in the References section.

6. Evaluation results of the models

Train data size	Architecture	Time per epoch	Test loss	Test accuracy
30%	RNN	135 sec	0.563482	0.889502
50%	RNN	219 sec	0.434482	0.898601
70%	RNN	292 sec	0.377454	0.904932
30%	CNN	11 sec	0.244383	0.916477
50%	CNN	18 sec	0.232999	0.918035
70%	CNN	24 sec	0.230185	0.918488

As can be observed in the results above, both models seem to achieve fairly good scores in all the tests for both loss and accuracy scores. However, the CNN models have outperformed the RNN in all the metrics, even comparing the RNN with the bigger training set and the CNN with the smallest one. They are also faster to train by far, which was expected.

This may be due to the fact that LSTM models need a lot more data to have a good performance, so maybe the size of the data that I have used is not big enough to compensate using an LSTM based architecture over a CNN one.

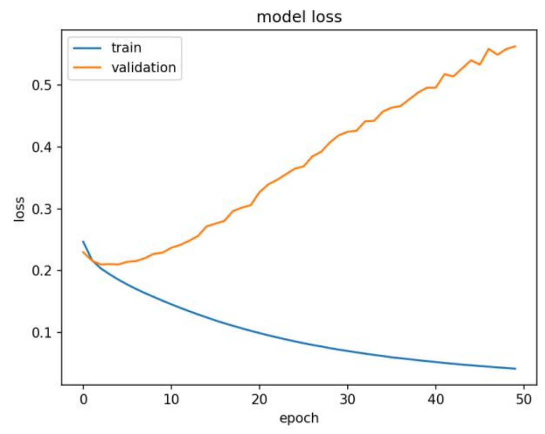
Maybe with more training data the LSTM architecture would end up surpassing the CNN. I have this clue since it can be observed that the difference in the loss and accuracy between the smallest dataset and the largest one is bigger in the RNN models than in the CNN models, in which the differences are more difficult to appreciate.

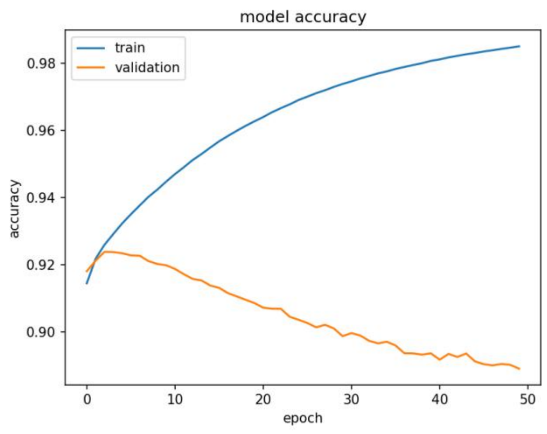
In sum, it may be possible that CNN models do not need a really big dataset to achieve good performance, but RNN have more potential to achieve better results when built with a lot more data since the performance grows more than with CNN.

The training and accuracy plots can be observed in the next two pages. They don't look as nice as the tests results for sure, or at least the validation results. The train loss and accuracy behaves as expected —the first decreases and the second increases—, but the validation metrics do the contrary to what they should do, which is not right for sure. It is expected to see some local maximums and minimums, but there is a consistent increasing in loss and decreasing in accuracy, specially in the RNN plots.

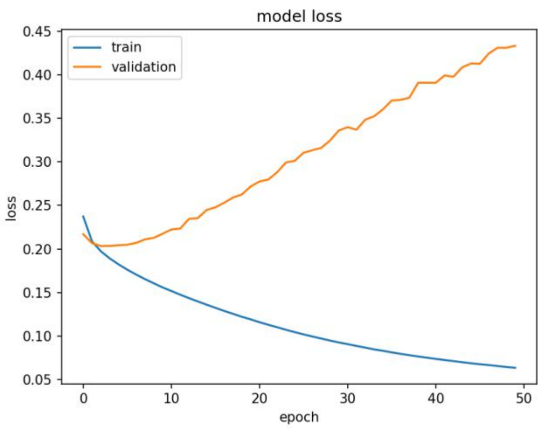
RNN plots

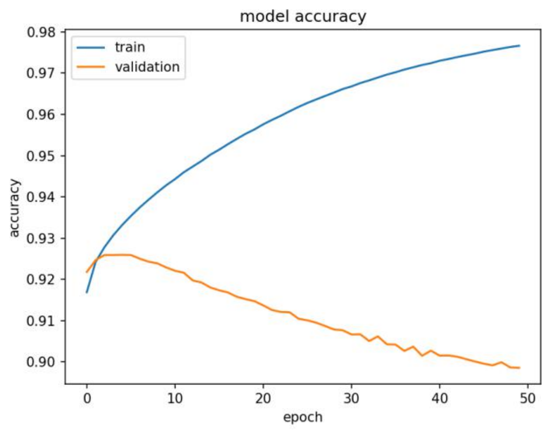
30%



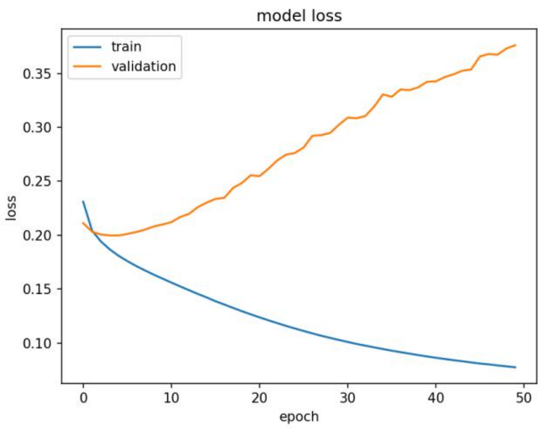


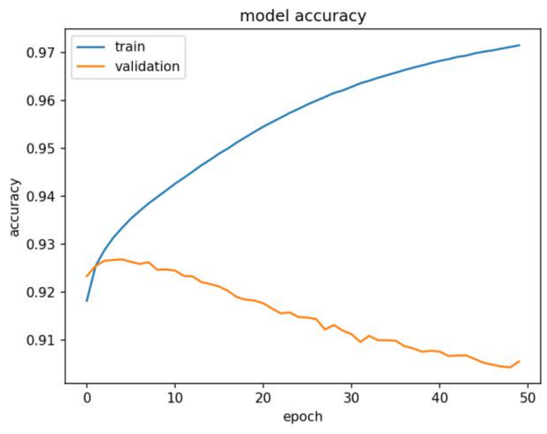
50%





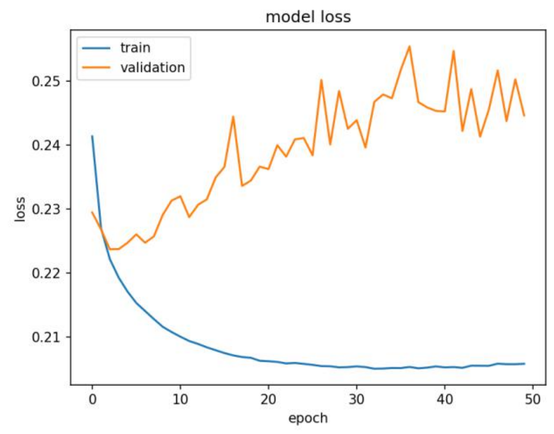
70%

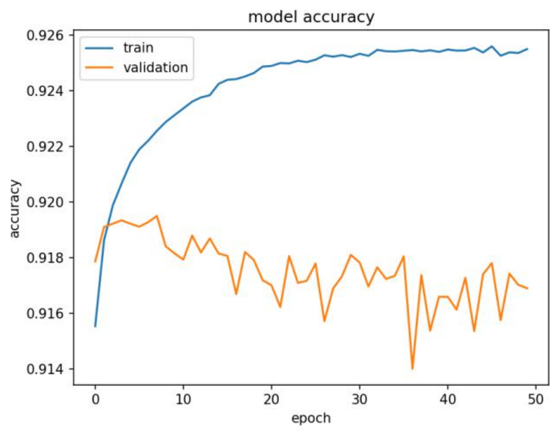




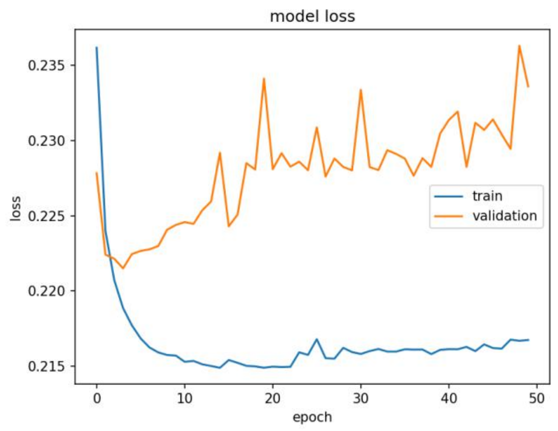
CNN plots

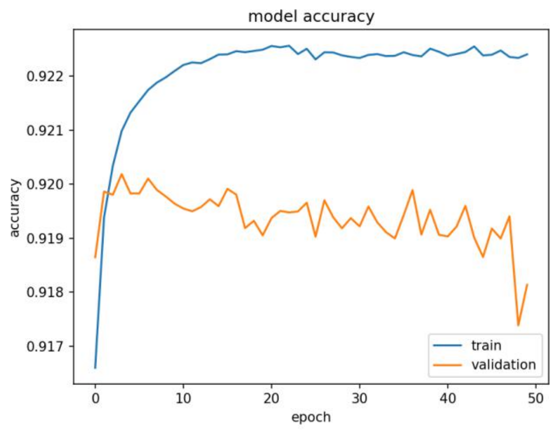
30%



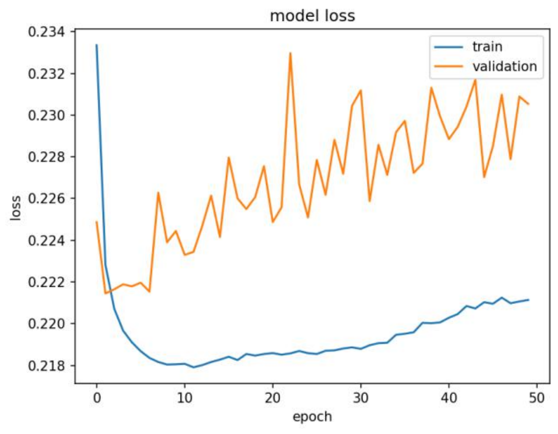


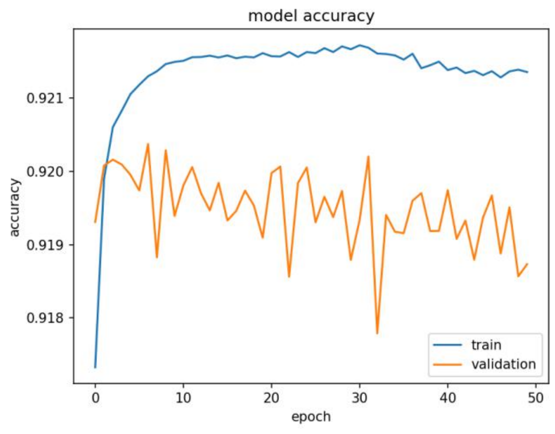
50%





70%





I am not completely sure about the reason for this behaviour, but I have an small hypothesis. My designed models are not optimized enough for sure. I have only trained once the RNN and even the CNN do not have that many tries. It would be better to retrain with different hyper-parameters —i.e. batch size, number of epochs, dropout, output dimensions of different layers, convolutional filters, kernel size, learning rate, etc.— and see how it affects the performance.

Due to time constrictions, there is no time to retrain the models for this project and, unless I can obtain a lot more data to train the model, maybe it is not even worth the try. In fact, having a too small dataset may be the reason for the strange behaviour with the validation data.

7. Testing predictions

As stated in sections 4 and 5, the output of the models is determined by a Sigmoid function. The Sigmoid function returns for each synopsis a vector of 20 scores from 0 to 1. Each of the scores correspond to one of the genres. For each of the scores, the closer it is to 1, the more confident is the model that the category of that score corresponds to the input sequence.

For doing these predictions, I have taken a few synopsis from the dataset that were about 50 words long and do not have genres classification, so they are not part of the training data for sure. To know the genres of those movies and compare them with the results, I have just searched them on IMDb and Filmaffinity.

Looking at the results in the following two pages, they do not seem bad, taking into account the problems seen when doing the evaluation.

For the first film, the RNN model manages to predict correctly the comedy genre always with a very high score. Although, it has more problems with the romance genre, but it manages to guess it with a good score in the end. It predicts other genres with high score —I do not know what the "Foreign" genre refers to—, but maybe the movie can be seen also as a drama, for instance.

The CNN seems to perform worse for this film and it is more inconsistent. It clearly improves from 30% of the training dataset to 50%, but with the biggest dataset the correct genres have worse predictions.

Movie	**Synopsis**		
Mia Sarah	In the half-light of a an old stately home, Samuel Davila and his grandfather Paul, once a revered writer, create schemes to repel the latest in a long line of psychologists come to cure the grandson of his severe agoraphobia: some delicious biscuits with added spice, a refreshing glass of water with added laxative.		
Original genres	**Architecture**	**Predicted genres**	
Comedy, Romance	RNN 30%	Comedy: 0.996233 Foreign: 0.981395 Drama: 0.969751	Action: 0.965726 [...] Romance: 0.000001
	RNN 50%	Comedy: 0.999972 Action: 0.947655	[...] Romance: 0.000063
	RNN 70%	Drama: 0.997698 Romance: 0.971338	Comedy: 0.899159
	CNN 30%	Documentary: 0.515888 Drama: 0.278499 Horror: 0.252777	Comedy: 0.127993 [...] Romance: 0.040123
	CNN 50%	Comedy: 0.563600 Documentary: 0.342694 Drama: 0.228271	[...] Romance: 0.064502
	CNN 70%	Documentary: 0.482247 Drama: 0.273841 Comedy: 0.203235	[...] Romance: 0.052341

For the second film the results are clearly worse than in the first movie. None of the models manages to give high scores to any of the correct genres, except for Drama in the RNN 50% and in the CNN 30%, but this is not maintained for models with bigger datasets. Maybe this film has a very confusing synopsis for the model —that is, it does not have enough language features to determine the genre correctly—.

Movie	**Synopsis**		
The Violent Ones	When a girl in a town that's populated by Hispanics is attacked, the only thing she says before falling into a coma is that her attacker is an outsider, a Caucasian. So the sheriff arrests the only outsiders there are. All he can do is hope that one of them will admit to being the attacker or that the girl can wake up long enough to identify him. But at the same time her father is preparing a lynch mob.		
Original genres	**Architecture**	**Predicted genres**	
Crime, Drama	RNN 30%	Action: 0.999862 Thriller: 0.999551 Drama: 0.217181	Mystery: 0.066626 Crime: 0.040206
	RNN 50%	Drama: 0.981272 Thriller: 0.976317	Action: 0.215322 Crime: 0.119199
	RNN 70%	Western: 0.999403 Action: 0.041586 Adventure: 0.038937	Drama: 0.028721 [...] Crime: 0.003054
	CNN 30%	Drama: 0.503401 Comedy: 0.371269	[...] Crime: 0.098477
	CNN 50%	Comedy: 0.381512 Drama: 0.292927	Crime: 0.181672
	CNN 70%	Drama: 0.392052 Thriller: 0.295293	Crime: 0.204212

8. Overall conclusions

As observed in sections 6 and 7, it seems that the models need more data and surely more optimization to be able to return better results on the evaluation and predictions. Designing a well optimized architecture would take a lot more time that what is available for this project.

Regarding the comparison of RNN and CNN, it is quite difficult to come with a answer to that question. Looking at the test evaluation metrics in section 6, CNN seem to perform better with this amount of data, while RNN improve more with more data, so it seems that it could potentially achieve better results. However, those results are not definitive and the problem with the plots has to be addressed for sure.

However, considering the limitations the trained models seem to perform decently enough for the given amount of data and the poor optimization. It seems quite promising, maybe it is possible indeed to design a language model architecture that would be able to actually predict movies given the synopsis.

References

[1] BRITZ, D. (2015). *Understanding Convolutional Neural Networks for NLP*. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[2] CHOLLET, F. (2016). *Using pre-trained word embeddings in a Keras model*. https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

[3] HUGHES, M., Li, I., KOTOULAS, S., SUZUMURA, T., (2017). *Medical Text Classification using Convolutional Neural Networks*. https://arxiv.org/abs/1704.06841

[4] KIM, J. (2017). *Understanding how Convolutional Neural Network (CNN) perform text classification with word embeddings*. http://www.joshuakim.io/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-embeddings/

[5] MAHESHWARI, A. (2018). *Report on Text Classification using CNN, RNN & HAN*. https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f

[6] SINHA N. (2018). *Understanding LSTM and its quick implementation in keras for sentiment analysis*. https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47

[7] YOUNG LEE, J. and DERNONCOURT F. (2016). *Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks*. https://arxiv.org/abs/1603.03827

[8] ZHOU, C., SUN, C., LIU, Z. and C.M. LAU, F. (2015). *A C-LSTM Neural Network for Text Classification*. https://arxiv.org/abs/1511.08630

Dataset and glove model employed

[1] 350 000+ movies from themoviedb.org. https://www.kaggle.com/stephanerappeneau/350-000-movies-from-themoviedborg#AllMoviesDetailsCleaned.csv

[2] Glove 6B vector 100d. https://nlp.stanford.edu/projects/glove/

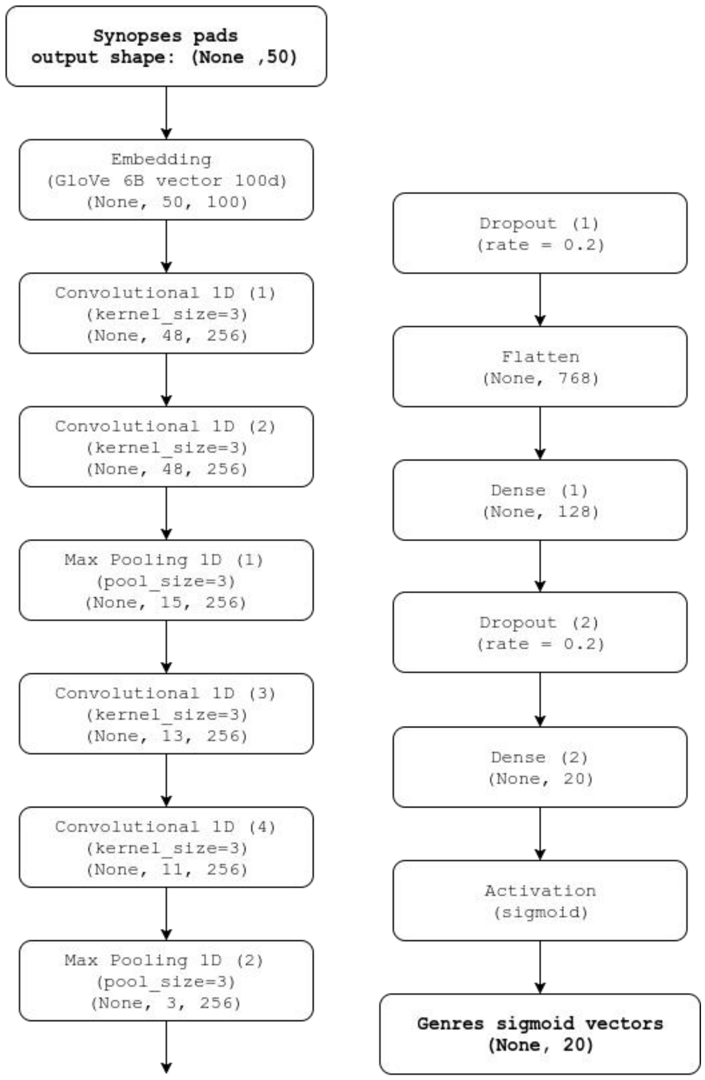
Appendix: other trained CNN models

1. Test results of the CNN described in section 5 but changing the convolutional filters to 256. The models are in the path *models/cnn/1-256/*.

Train data size	Test loss	Test accuracy
30%	0.292246	0.913263
50%	0.246307	0.917704
70%	0.235056	0.917924

2. Another architecture[3] and test results. The tree is in the code as "Architecture 2" Models are available in the paths *models/cnn/2-256/* and *models/cnn/2-128/*.

Convolutional filters = 256		
Train data size	**Test loss**	**Test accuracy**
30%	0.248042	0.915968
50%	0.243991	0.915267
70%	0.242539	0.917448
Convolutional filters = 128		
Train data size	**Test loss**	**Test accuracy**
30%	0.251637	0.915756
50%	0.237550	0.916776
70%	0.240913	0.917023



[3] Based on CHOLLET (2016). Title and URL are available in the References section.