INSIGHTS AND OBSERVATIONS

This is an Natural Language Processing (NLP) binary classification task comparing two different models when performing a sentiment analysis of IMDB movie reviews.

The data set:

The dataset is very well balanced, with no missing values and already divided into training and test sets, each of them totalling 25.000 movie reviews. For both sets, reviews are equally divided between positive and negative (12.500 each). Usual data cleaning was applied (removal of punctuation and stop-words, stemming, setting all words to lower case) to improve the quality of the data and, consequentially, of the models.

The choice was made for a word2vec embedding instead of a customized embedding with 'layers.Embedding'. It is not specially adapted to this task, but it is an intrinsically good embedding for different kinds of tasks, and it is faster to be trained. This allowed to try different model architectures without the need to reduce the training dataset during prototyping. Despite being a general algorithm, the associations are still learned according to the training dataset, and good models can be achieved.

Architecture:

Since text is also a form of recurrent data, with a sentence being a sequence of words, and a text being a sequence of sentences, we can think of Recurrent Neural Networks (RNN) to work with it. LSTM networks are RNN extensions designed to learn sequential (temporal) data and their long-term connections more precisely than standard RNNs. A recent work (Sakira, 2018) also classifying movie reviews showed that GRUs performed only slightly better than LSTM in a multiclassification task and were computationally more expensive at the same time.

Convolutional Neural Networks (CNN) are also used in NLP tasks and converge faster that RNNs because of the slightly lower number of parameters. For this reason, I decided to compare both kinds of networks when dealing with this data set.

The next step was how to build the architecture of the models. Some facts needed to be considered. The more parameters a Deep Neural Networks has, the more features from the dataset the model can learn, leading to more robustness. On the other hand, the more it takes to be trained (both computationally and timewise), and the more it tends to overfit. The attempt was, just like in biological systems, to find an optimal solution balancing time constraints, computational power, and good results.

Some methods can be used to prevent overfitting:

- Early Stopping class is called at the end of each epoch. It was used in both models to stop de network when it reaches a minimum validation loss and to prevent it from running extra useless epochs. Neural Networks are stochastic algorithms, and the loss can look very bumpy and noisy sometimes (as it can be seen in the notebooks). If the run is stopped as soon as the validation loss get worse, it might be a premature end. It is necessary to wait for a certain number of iterations without improvement, and this is what the 'patience' parameter does. Both models had 'patience=10' and 'restore_best_weights=True' (restoring the weights corresponding to the best validation loss).
- Regularization is used to lower the complexity of the model during training. A regularization term is added to the loss during the training phase, lowering (or penalizing) high-valued

coefficients. Both models use an L2 (or Ridge regression) regularization, which forces some of the weights towards zero but without becoming necessarily zero. This is the most common regularization to prevent overfitting.

- Dropout layers were also used in both models. During the training, at each iteration, the dropout randomly "shuts down" some neurons, so their weights are not updated. This prevents neurons from over-specializing in one single input and being then unable to generalize.

Two stacked layers were used in the LSTM. This makes the model more complex and more capable of learning important features for the classification task. For the CNN, 3 convolutional 1D layers were use, as well as Max Pooling layers for dimensionality reduction. Simpler architectures did not show very good results.

Following the current standards of good practice, CNN used the relu activation function, while LSTM used tanh. Different optimizers were also used: Adam in the CNN, and RMSprop in the LSTM. To improve the search for the minimum loss, the Adam optimizer was coded using Exponential Decay to decrease the learning rate as the CNN converges. This procedure was not used with the LSTMs due to the RMSprop being an adaptive learning rate method that makes these adjustments by itself.

Performance:

The chosen evaluation metrics was accuracy. First, it is one of the suitable metrics for a classification task. Second, we want to measure how close a predicted value is to a known value. In other words, we want to know how close the models' predictions are from existing data in the test set.

The two models achieved a high accuracy with the testing set (87.204% in 28 epochs for RNN, and 85.868% in 63 epochs for CNN). The latter had a faster run, despite having more than the double of epochs. If we consider the trade-off between accuracy and time, the CNN model wins.

Dropout layers made a difference when compared to simpler models without dropout, but regularization seemed to make a bigger difference in the quality of the architectures. Models without regularization showed an overfitting behaviour even with dropout layers, what shows the importance of this method even if it slows down the computation time.

The main LSTM model has an L2 penalization only in the first recurrent layer. With a slightly lower number of epochs, it achieved very similar results to a draft model with one single layer and more hidden states. When the regularization was applied to both layers, 10 more epochs were necessary, implying a significantly slower run time without any gains in accuracy. Interestingly, when the L2 was applied only in the second recurrent layer, the model also overfitted, maybe because the second layer could already learn a lot of noise from the previous layer. The simplest CNN model with one Conv1D, no regularization and no dropout had a loss function that only increased from the beginning.

Overall, all models had high accuracy (always above 85%, except for the first CNN draft model, with 83%), making better predictions than a baseline model would do by mere random chance (50%). However, the graphics of the training histories made me opt for the two models with better fit. It is possible to understand the importance of looking at the loss function while human metrics could be misleading. Other models can be found in the Appendix of this repository.