Class: Deep Learning
Instructor: Georgios Petasis
Student: Panagiotis Mavrogiannis

Purpose of this report is to show the procedure that was followed in order to correctly identify the sentiment from a list of IMDB reviews, using a recurrent neural model. Tensorflow-Keras and PyTorch frameworks were used for the trials in order to implement the model but, the final solution was implemented using Tensorflow-Keras. NLTK framework was also used for the pre-process. Most of the implementation has come from the hyperlinks in the assignment file and then, adjustments were made in order to tune the model performance.

Dataset

The dataset that was used was downloaded from http://ai.stanford.edu/~amaas/data/sentiment/. The files contain 25000 reviews for training and 25000 for testing purposes. Only the training part of the total dataset was used. This was done due to the fact that, it is possible to download the same dataset from Keras and its size also amounts to 25000 reviews. So, I used both datasets to verify any differences in the results from the training. No differences were identified. Because, Keras dataset is pre-processed, the Stanford dataset was preferred in order to implement the pre-processing on my own.

Pre-Processing

The Stanford training dataset is equally balanced with positive and negative reviews amounting to 12500 each. Each review is stored in a file and so, each file was parsed and its output was stored in a list object then, the list was shuffled for the positive and negative reviews to blend. The implementation for this part was found in https://github.com/iworeushankaonce/ . After that, all reviews were merged in a single text in order to remove punctuation and convert words to lowercase. Stopwords were removed using NLTK, reducing the total words from 5.8 millions to 3.1 millions thus, improving accuracy about 1-2%. Text vocabulary was extracted and then, IDs were assigned for each word. Using these IDs, text was then transformed again into distinct reviews which finally, were transformed into list of integers using the vocabulary. Next step was to extract the maximum review length which was done by plotting a histogram with the review lengths. 75th percentile value was found to be 152 and so was chosen as the maximum review length. The final pre-processing step was to pad the reviews which had less length than 152 in order to be inputted to the model.

Model

The final model comprises an Embedding, a SpatialDropout1D, a LSTM and a Dense-output layer. Input dimension for the Embedding layer is equal to the vocabulary length + 1 for the padding character. Initially, I tried to implement the model in PyTorch in order to take advantage of my experience from the 1st assignment and to use the GPU from my machine but, due to compiling errors, I resorted to Keras without GPU support which affected the number of trials. In general, increasing the depth and the layer size of the network (using dense layers) resulted in unbearable training times but also, seems that a simple

model has a better result. Only LSTM was used because it seems to have better results according to sources in the internet. Dropout layers were added after the initial trials for regularization purposes and apart from the SpatialDropout1D layer, simple Dropout layers were applied in different levels. Dropout regularization was also applied from within the LSTM layer. The output layer has size of one unit due to the fact that, the model has an architecture of many-to-one and because the output is binary, the activation function is a sigmoid. For the same reasons, binary cross entropy was used as the loss function. Regarding the optimizer, Adamax, Adadelta, SGD and Adam were tested with varying learning rates. The latter from the optimizers was used because of yielding better results. Finally, gradient clipping using norm, was applied in order to handle exploding gradients. Regarding all of the above, a good approach would also be to implement the model using a custom constructor instead of the default provided from the Keras. Doing so, allows access to each of the layers outputs and would be much more easy to delve in the network logic. A much more better approach would be the implementation in pytorch in order to check if indeed there is an issue with exploding gradients. Applying attention layers was also considered but, was not applied due to insufficient studing.

Training

The 25000 were divided into training, validation and test set with a ration 8:1:1 respectively. The number of epochs was selected to be 4 because of the learning curves in the trials and initially had the value of 10. During the trials it was set to 3 and only changed to 4 after reducing the number of layers by removing the hidden dense layers. Plots were consulted for the regularization of the model.

Conclusion

The initial implementation from the hyperlinks, yielded an approximate .83 test accuracy and after the tuning, it reaches an approximate .885 while also reaching .89 at times. A further improve would be the application of Attention which I think it would probably improve the model.