Artificial Intelligence II
Assignment 3 Report

Andreas - Theologos Spanopoulos (sdi1700146@di.uoa.gr)

December 27, 2020

Regular Model

The Notebook (AI2_HW3.ipynb) is properly commented, so I will not go into deep details regarding most matters.

Preprocessing

Every sentence from the dataset in preprocessed using the "main_pipeline" function defined in the preprocessing.py file. It removes urls, twitter tags, most useless punctuation, numbers, multiple whitespace and it converts every letter to lowercase. Some sentences turned out to have a length of 0 after being preprocessed, so they got removed fromt the dataset.

Architecture

The architecture of the final model is pretty straightforward:

$$\begin{aligned}
\text{Input X} &\xrightarrow{\text{Embedding}} \text{X embedded} \\
&\xrightarrow{\text{RNN Cell}} \text{RNN output, } \underline{\text{last hidden state output}} \\
&\xrightarrow{\text{Dropout}} \text{Dropout output} \\
&\xrightarrow{\text{FC Layer}} \text{FC output} \\
&\xrightarrow{\text{Sigmoid}} \text{logit (model output)}
\end{aligned}$$

Note that from the RNN only the last hidden state is used.

Hyperparameters

The hyperparameters found to be optimal (using a Grid Seach and many trial/errors) for the model are:

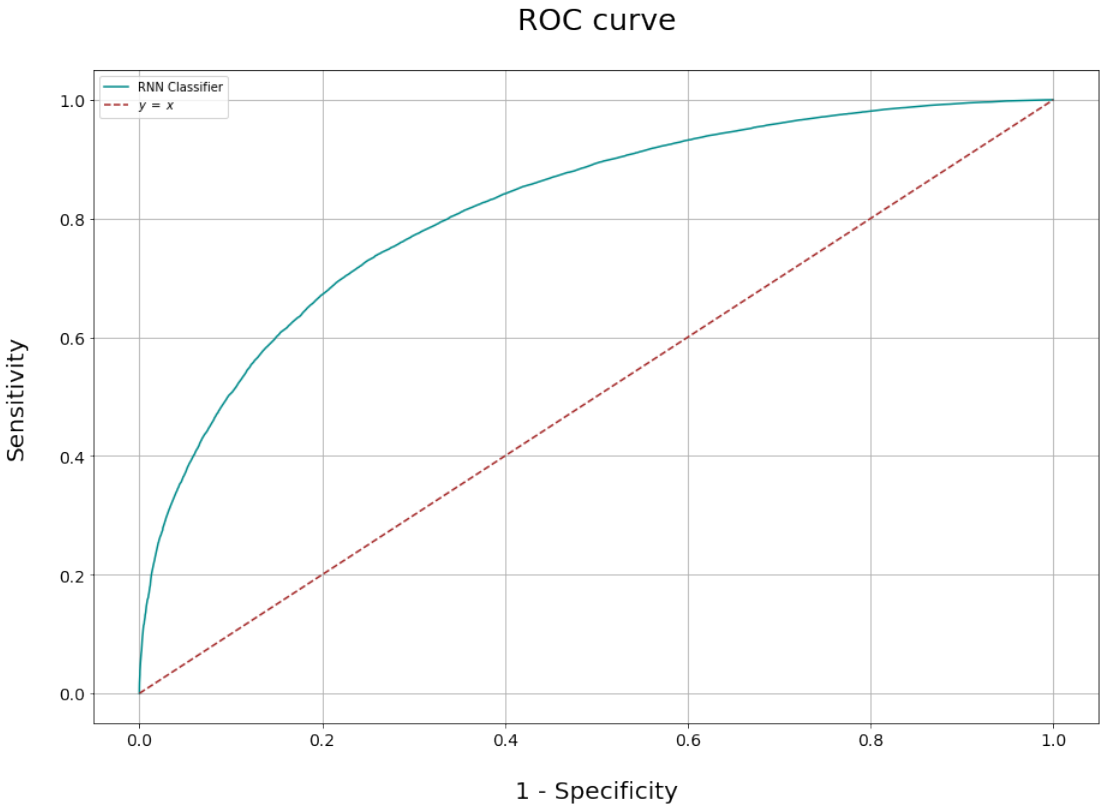
- Batch Size = 32
- Max Vocabulary Size = 25000
- Epochs = 5
- Learnign Rate = 0.0075
- Clipping amount = 10
- GloVe = glove.twitter.27B.100d
- RNN Type = LSTM
- Number of Stacked RNNs = 1
- Skip Connections = False
- Embedding Dimension = 100 (from GloVe)
- Vocabulary Size = length(vocabulary)
- Hidden Dimension = 512
- RNN Layers = 2
- RNN Dropout = 0.1
- Dropout Rate = 0,1
- Bidirectional = True
- Optimizer = Adam
- Loss = Binary Cross Entropy

Performance

The performance of the model using the above hyperparameters yields:

- Training Loss: 0.54, Validation Loss: 0.52
- Training Accuracy: 0.72, Validation Accuracy: 0.74
- Training F1 Score: 0.71, Validation F1 Score: 0.74

The ROC Curve for the model on the Test set, is:



Experiments

The following hyperparameters/experiments/modifications were tried to the model and its architecture:

- Stacking more RNN (LSTM and GRU) layers (2 were tried) did not lead to any significant improvement. In the contrary, it slowed down the training process.
- Skip Connections only make sense for 2+ RNN stacked layers. It was tried, bu since the shape of each batch size is incosistent, padding was required, something that I tried to avoid.
- Number of Hidden Layers: 1, 2, 3 and 4 were tried. 1 Hidden layer performed the worst, but was faster during training. 2 and 3 seems to be performing the best, with models using 3 hidden layers showing a slight improvement.
- LSTM and GRU recurrent networks were tried. GRU was faster during training, but scored always 1-2% less F1 score in each epoch, compared to LSTM.
- Gradient Clipping was tried with values in the set {5, 10, 50, 100}. No difference was noticed. Also, no clipping was applied, which led to increase in the losses.
- Dropout was tried with values from 0.0 up to 0.5. Around 0.2 was found to give the best model, and it didn't make sense to increase the dropout rate as the model was not overfitting.
- Learning rates in the range [0.1, 0.0001] were tried. 0.001 works better in the begginign but it plateaus from the first epoch. 0.0001 starts lower but climbs to a better value after some epochs.
- Fine tuning the Embedding layer was tried, that is, to freeze the embedding layer in the first few epochs and then unfreeze it. No improvements were shown.
- Chaning the Max Vocabulary size in the range [15000, 30000] did not have any effects.

- Adding weight decay did not improve anything, as the model was not overfitting.
- Using bidirectional RNNs improves Accuracy and F1 Score by 5%.
- Different Train sizes were tried, from 80% to 95% of the whole dataset. 90% is believed to preserve an acceptable balance, so it was chosen as the default split.
- Different methods of preprocessing were tried and removed. Very simple preprocessing pipelines (remove urls, keep only alphanumerical values and convert to lowecase) decrease the model performance, but also very thorough ones (everything mentioned in page 1) remove valuable information from the dataset. I believe this is the main reason for why the model was not able to perform better. I will try more configurations of preprocessing methods before submitting.

Comparison with the best model from the previous assignment

The best model from assignment 2 (NN with tfidf) reached a peak score of 0.78 Accuracy and F1. Compared to this model, is perfomed better, but it showed signs of overfitting. This model on the other hand, reaches 0.74 on the same metrics, but it doesn't overfit. Its training time is a bit slower though. Also, the ROC curve of this model is has smaller area than that of the second assignment. So yeap, the previous model performs better in general, but it doesn't have a window of improvement. On the contrary, we will see that once adding Attention to the current model, its performance improves significantly.

Attention Model

The Notebook (AI2_HW3_Bonus.ipynb) is properly commented, so I will not go into deep details regarding most matters. The paper which describes the implementation, is the: A Structured Self-attentive Sentence Embedding.

Preprocessing

Same as in the first model.

Architecture

The architecture of the attention model is pretty straightforward:

$$\text{Input X} \xrightarrow{\text{Embedding}} \text{X embedded}$$

$$\xrightarrow{\text{RNN Cell}} \underline{\text{RNN output}}\text{, last hidden state output}$$

$$\xrightarrow{\text{Attention}} \text{Attention Network output}$$

$$\xrightarrow{\text{Dropout}} \text{Dropout output}$$

$$\xrightarrow{\text{FC Layer}} \text{FC output}$$

$$\xrightarrow{\text{Sigmoid}} \text{logit (model output)}$$

Note that to the Attention Network we feed the whole RNN output (all the hidden states).

Hyperparameters

Using the same hyperparameters as the previous model does not yield good results, as the new model overfits early, plus it takes a lot of time for training. The hyperparameters below seem to be yielding the best results for the attention model:

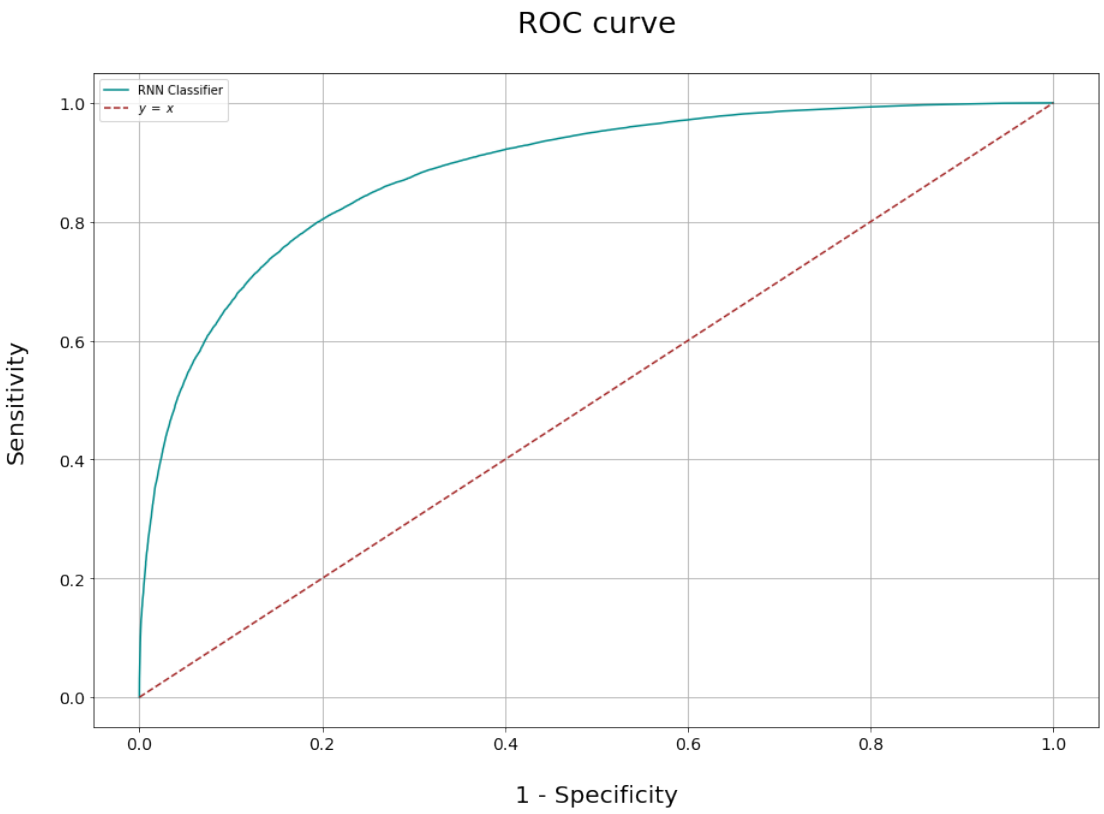
- Batch Size = 128
- Max Vocabulary Size = 25000
- Epochs = 15
- Learnign Rate = 0.003
- Clipping amount = 10
- GloVe = glove.twitter.27B.100d
- RNN Type = LSTM
- Number of Stacked RNNs = 1
- Skip Connections = False
- Embedding Dimension = 100 (from GloVe)
- Vocabulary Size = length(vocabulary)
- Hidden Dimension = 256
- RNN Layers = 2
- RNN Dropout = 0.2
- Dropout Rate = 0,2
- Bidirectional = True
- Optimizer = Adam
- Loss = Binary Cross Entropy
- Max Sentence Length = 40
- Context Dimension (d_a) = 2 * Hidden Dimension = 512

Performance

The performance of the attention model using the above hyperparameters yields:

- Training Loss: 0.44, Validation Loss: 0.43
- Training Accuracy: 0.79, Validation Accuracy: 0.80
- Training F1 Score: 0.79, Validation F1 Score: 0.80

The ROC Curve for the model on the Test set, is:



Takeaways

Using the attention mechanism improves significantly the performance of the model. It slows down training by a small factor, but it's acceptable. The model managed to improve its metrics by quite a big factor. Plus, if you take a look at the Notebook, you can see that the model keeps improving by a bit in every epoch, especially when the scheduler kicks in and reduces the learning rate when the model starts plateauing.

The performance of the model can be further improved by

- Reucing the batch size to 32. Epoch speed will drop at about 4 minutes/epoch, but with 10 epochs the results will be the same as of now.
- Increasing the hidden size of the RNN will improve the performance, but again it will make training slower.
- Adding weight decay and increase the dropout rate in case the model starts overfitting, which it hasn't done yet.
- Increasing the context dimension d_a improves performance.
- Of course, increasing the number of epochs that the model is trained.
- Scheduling the learning rate with less patience, that is, as soon as the Validation Loss plateaus, reduce the learning rate.