YS19 - Artificial Intelligence II
Project 2

Efstathios Chatziloizos - 1115201800212

December 2022

A sentiment classifier using a feed forward neural network for imdb movie reviews has been developed. The machine learning framework PyTorch was used. The neural network's input layer is comprised of as many neurons as the number of dimensions of the GloVe embeddings used. For the purposes of this assignment glove.twitter.27B.200d.txt was used, hence 200 input neurons are present.The neural network's output layer consists of one neuron, whose output is a probability p, where $p > 0.5$ indicates a positive sentiment and $p < 0.5$ indicates a negative sentiment.

Architecture

A wide range of architectures were considered, implemented and tested during this assignment. The most prominent ones consisted of 3 hidden layers, although architectures with 2 hidden layers of appropriately sized layers also showed promising results. The main tests where carried out with 3 layers of sizes (128, 64, 32), (200, 200, 200), (100, 100, 100) and (200, 150, 100). In the end the architecture with the best results and the one chosen had 3 layers of sizes (200, 100, 50). It is shaped as a pyramid with larger layers in the beginning and gradually smaller layers towards the end. Even though the usefulness of the pyramid shape is debated, it seems to perform the best for this dataset.

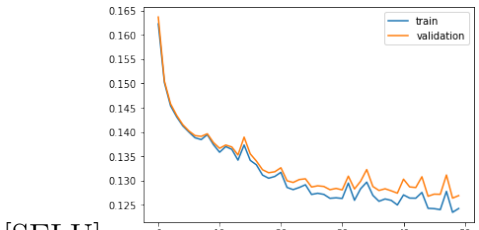
Activation Function

Many activation functions were considered (e.g. Sigmoid, ELU, ReLU, SELU) and the majority of them had similar results. Some were suited more for the majority of the combinations of architecture and hyperparameters, while others had both positive and negative outliers.

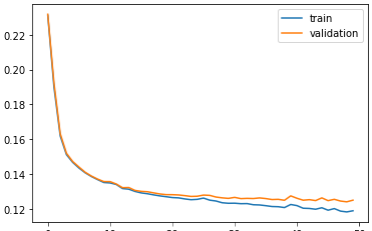
The most consistent of them all was by far ReLU, which performed more than adequately on most combinations of architecture and hyperparameters, with quick convergence and low variance from epoch to epoch.

SELU was a close second, often outperforming ReLU, however its inconsistent nature and high variance between epochs made it a less than optimal activation function for this dataset for most implementations.

The undisputed worst performer both on convergence speed and quality was the Sigmoid activation function. Tuning the hyperparameters, even though it was helpful, almost never closed the gap enough to the rest of the activation functions.



[SELU]

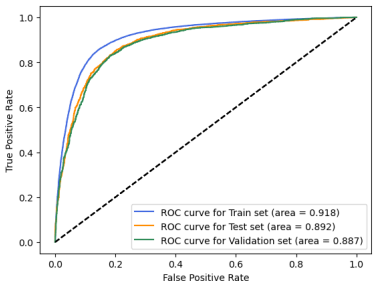


[ReLU]

Figure 1: SELU - Good convergence speed - High variance. Choppy graph.
RELU - Good convergence speed and quality. Smooth graph.

Dropout

Using dropout (ranging from 20% to 60%) did not improve the results of the model. Even when accounting for the "off" neurons and appropriately increasing the number of neurons, the results did not improve. The accuracy of the model decreased significantly and therefore dropout was omitted from the final model.

[ROC] 

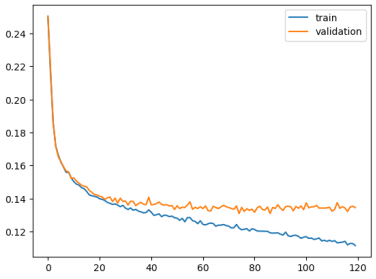
[Learning Curve] 

Figure 2: Example of ROC and learning curves for a model with 50% dropout.

Hyperparameters

Numerous hyperparameters for each architecture were tested. More specifically, the hyperparameters tuned for each model were:

- Number of hidden layers.
- Number of neurons per layer.
- Learning rate.
- Loss function.
- Optimizer.
- Batch Size.
- Activation Function.
- Number of epochs.

Some optimizers tested, allong with their respective best suited rest of hyperparameters are in comments on the source code file, making it easier to test multiple optimizers by simply choosing an optimizer and changing the rest of the hyperparameters according to the instructions mentioned in the comments.

The learning rate, maybe the most important hyperparameter, dictated the pace of the model's learning. Tuning it for each model was crucial. The method of tuning used was to start from a high value and gradually reducing it, so as to not be too slow and not overfit the data.

A plethora of loss functions were tested, some of which are left as comments to easily switch from one to another. The loss function to consistently give close to optimal results for this dataset no matter the architecture was the Mean Squared Error function.

The number of epochs was increased/decreased in such manner to both give enough training iterations for the model to adjust properly, while simultaneously early-stopping to avoid overfitting to the training data.

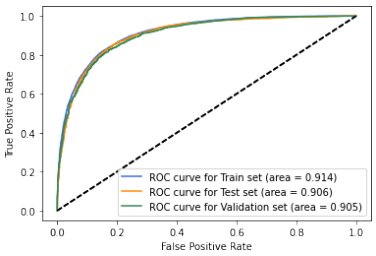
Lastly, the effect of the batch size was not as crucial as initially considered. By increasing the batch size and adjusting the number of epochs, in the way mentioned above, similar results were observed both in convergance speed (in seconds, not in number of epochs) and quality. Additionally, the accuracy of the models did not differ enough in any architecture to declare a clearly best batch size. One effect of increasing the batch size, for example from a value of 16 to 128 is the quicker pass of the data per epoch, although mitigated by the need for an increased amount of epochs.

Chosen Model

The chosen model is comprised of 3 hidden layers (200, 100, 50). The hyperparameters of the model are:

- Activation Function - ReLU.
- Learning rate - 0.00002.
- Loss function - Mean Squared Error.
- Optimizer - NAdam.
- Batch Size - 16.
- Number of epochs - 50.

The dataset is split into train set 80%, validation set 10% and test set 10%.

[ROC] 

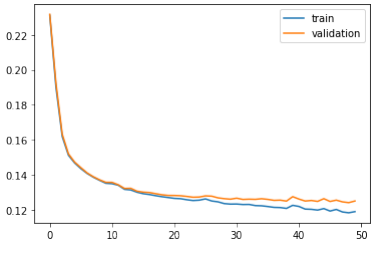
[Learning Curve] 

Figure 3: Chosen Model's ROC and learning curves.

Train: Precision: 0.816, Recall: 0.855, F1: 0.835
Validation: Precision: 0.809, Recall: 0.845, F1: 0.827
Test: Precision: 0.825, Recall: 0.851, F1: 0.838

Extensive comments have been used throughout the whole source code, explaining every step of this implementation.

The source code was written using VSCode, although the code was also tested on https://colab.research.google.com/

2021 Data Mining Assignment

The 2021 Data Mining 1st Assignment also used GloVe embeddings. Snippets of code might have been used from my last year's project. The project was carried out with the help of Loukovitis Georgios - 1115201800100, who passed this class in 2021 and therefore won't be handing in any projects this year.