1 Task Overview

In this task we received a corpus of tweets. The goal was to classify the tweets in one of the following three categories; `'Neutral', 'Pro Vax' and 'Anti Vax'`, using **Recurrent Neural Networks** and **GloVe Embeddings**.

2 Dataset Preprocessing

2.1 General

The training dataset contains *15976* rows and *3* columns. The first column holds the index of each row and therefore was discarded. The second and third columns contain the tweets and their labels respectively. The distribution of the labels is shown on Fig.1. In the same notion, the validation dataset contains *2282* rows. Its columns were treated similarly to the ones of the train dataset. The distribution of the labels is shown on Fig.2.

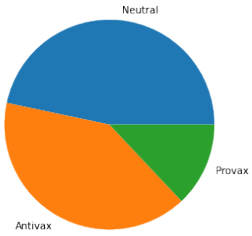

Figure 1: Train Dataset Cardinality

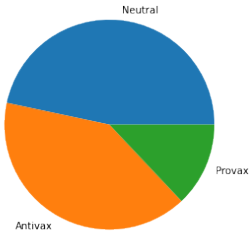

Figure 2: Validation Dataset Cardinality

We observe that the distribution of labels is analogous in both sets. However, there is a smaller number of samples labeled as `'Provax'` in contrast to the other categories. We expect that the imbalance of the datasets will affect our results later on.

2.2 Data Cleaning

Regarding the preprocessing step, all the URLs were removed from the tweets. Additionally, all the new line characters, double spaces and punctuation signs were deleted. The stop words were, also, removed and the remaining ones were stemmed. The resulting sentences were stored in csv files under the names *cleaned_train.csv* and *cleaned_test.csv* each one corresponding to new train and test datasets respectively.

In order to exploit the preprocessed data, two objects of class type `Field` and `Label` were defined. The object `Field` was used to specify how to preprocess each data column in the datasets. In this case, the `spacy tokenizer` was utilized (in order to tokenize the sentences) and the `batch_first` flag was specified to `True`, so that the batch dimension would be first. The object `Label` was used to define the labels in the classification task.

These fields are later passed as field parameters to `TabularDataset`. `TabularDataset` is a `TorchText` data structure that specifically deals with tabular datasets (including *csv* files that fit our case). Here, it is used to import the train and test data, wraping all the columns, both tweets and their labels into a single object.

2.3 GloVe Embeddings

For this assignment the **'glove.6B.200d.txt'** embedding vectors were used. In order to build a vocabulary utilizing the aforementioned pretrained vectors alongside our train dataset, the `build_vocab` method was used.

2.4 Iterators

To iterate over the datasets in batches, `TorchText` provides a `BucketIterator` that groups sequences together. It returns a `Batch` object that contains the data of one batch while the text and labels can be accessed via the column names `tweet` and `label` respectively. Here, the `batch_size` of both datasets was set to *1024*, resulting in *16* batches for the train dataset and *3* batches for the test dataset.

3 RNN Models

3.1 Model Selection

For this task the following two RNN models are examined:

1. Recurrent Neural Network with GRU cells
2. Recurrent Neural Network with LSTM cells

To implements the models, the `nn.GRU` and `nn.LSTM Pytorch` classes were utilized. In both cases, the output of the models passes through a `Dropout` and a `Linear` FC layer. Finally, the logits are retrieved by applying `Softmax` on the output of the FC layer.

Dropout Layer

The Dropout layer randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Study [1] describes that the use of a Dropout layer has proven to be an effective technique for regularization, preventing the co-adaptation of neurons. In addition, study [2] states that a common value of `p` is *0.5* when retaining the output of each node in a hidden layer and a value closer to *1.0*, when retaining inputs from the visible layer. In this assignment, a thorough experimentation regarding the most suitable value in the range [0.5, 0.8] has been performed.

Linear Layer

The Linear layer simply applies the linear transformation $y = xA^T + b$ to the data as a final step.

Softmax function

The Softmax function ensures that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.

3.2 Hyperparameter Tuning

While the **GRU cells** were introduced in 2014, the **LSTM cells** were introduced in 1997, making them more thoroughly explored. In many tasks, both architectures yield comparable performance and therefore, the tuning of hyperparameters may be more important than choosing the appropriate cell [4]. To that extent, our two models were tuned and compared side by side.

To tune the hyperparameters of both networks, a Grid Search algorithm was utilized. The execution of the aforementioned algorithm resulted in the following:

1. Recurrent Neural Network with GRU cells
    - Hidden size: 128
    - Number of recurrent layers: 2
    - Dropout Probability: 0.7

- Skip Connections: False
- Clipping amount: 0.5
- Learning Rate: 0.001

2. Recurrent Neural Network with LSTM cells

- Hidden size: 128
- Number of recurrent layers: 2
- Dropout Probability: 0.7
- Skip Connections: False
- Clipping amount: 1.0
- Learning Rate: 0.001
- Bidirectional: True

In both cases the `Cross Entropy Loss` loss function was utilized, as it is very useful for multi-class classification. Furthermore, the `Adam` optimizer [5] was used. `Adam` optimizer implements an algorithm for gradient based optimization of stochastic objective functions. It combines the advantages of `AdaGrad` [6], which works really well in settings with sparse gradients, but struggles in non-convex optimization of neural networks, and `RMSProp` [7], which tackles to resolve some of the problems of Adagrad and works really well in on-line settings, computing individual adaptive learning rates for different parameters.

4 Experiments

4.1 GRU vs LSTM cell

This experiment compares the results obtained from the two aforementioned models after excecuting for *8* epochs. We can deduct that both models achieve similar scores by observing both the classification results presented in Tables 1 & 2 and the final `ROC curves` illustrated in Figures 3 & 4.

Table 1: Classification Scores - Weighted Average

RNN Model	W. Precision	W. Recall	W. F1-Score
GRU	0.68	0.69	0.68
LSTM	0.69	0.67	0.67

Table 2: Classification Scores - Macro Average

RNN Model	M. Precision	M. Recall	M. F1-Score
GRU	0.62	0.58	0.58
LSTM	0.63	0.59	0.60

Despite presenting both cases, the *weighted average* scores are more suitable for this task, due to the imbalanced nature of our datasets. In general, weighted average returns the average considering the proportion of each label in the dataset, in contrast to macro average that returns the average without considering the proportion for each label in the dataset. Therefore, we can safely conclude that our models achieve an `F1 score` close to 67-68%.

The `loss vs epoch curves` are also presented in Figures 5 & 6. The plots suggest that the validation set slightly overfits as the epochs pass by. Studies like [8] [9] indicate that this behavior lies upon the imbalanced nature of the datasets. Both of the aforementioned studies present new loss models like `Focal loss` [10], in order to reduce the impact that easily classified samples have on the loss.

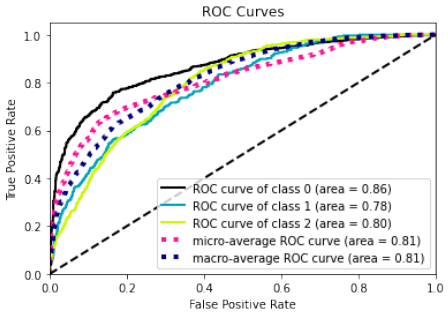

Figure 3: GRU: ROC curve

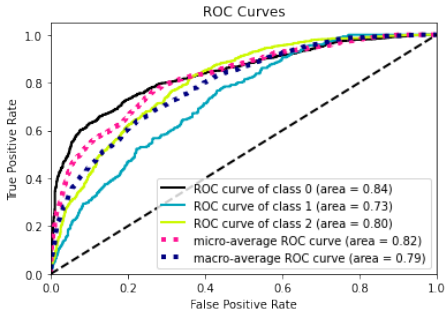

Figure 4: LSTM: ROC curve

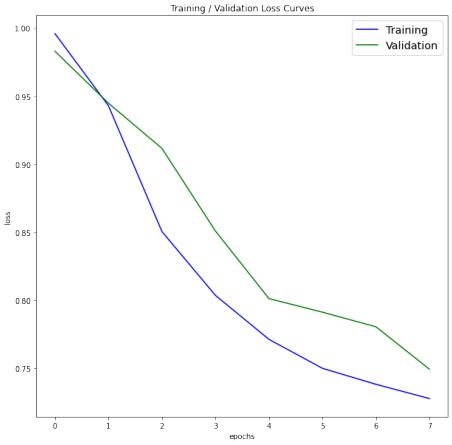

Figure 5: GRU: Loss vs Epochs

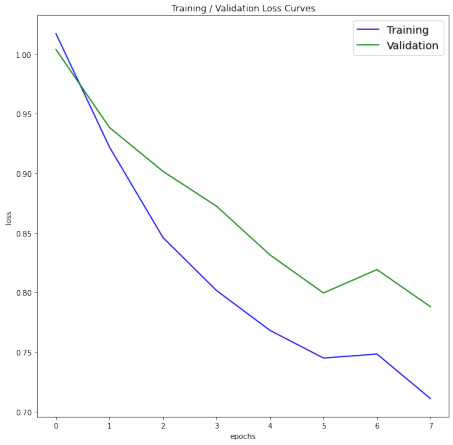

Figure 6: LSTM: Loss vs Epochs

4.2 Adding attention

To improve the performance of both models an `Attention` layer was added. The self-attention mechanism implemented, is presented on [11]. The general idea of the proposed self-attention mechanism is to replace the commonly used max pooling or averaging step methods. Different from previous approaches, this mechanism allows the extraction of different aspects of the sentence into multiple vector representations.

More precisely, the attention mechanism takes the whole LSTM / GRU hidden states `H` as input, and given two t arbitrarily constructed weight matrices Ws_1 and Ws_2, outputs a vector of weights $\texttt{A} = softmax(Ws_2 tanh(Ws_1 H^T))$. More details can be found on [11].

The addition of the `Attention` layer, despite boosting the performance, allows our models to become more aware of the minority class. To support these claims the analytical results of the classification report for the `LSTM` and `LSTM+Attention` models are presented in Table 3. This analytical report clearly shows the improvement of the model in terms of distinguishing the minority class (`class 1`) among the other classes (`class 0 and class 2`). The `GRU` and `GRU+Attention` models resulted in also similar performance.

Table 3: Classification Report of LSTM and LSTM+Attention models

RNN Model	Class	Precision	Recall	F1-Score
LSTM	0	0.81	0.66	0.73
	1	0.33	0.30	0.32
	2	0.62	0.77	0.69
LSTM + Attention	0	0.79	0.76	0.77
	1	0.48	0.34	0.40
	2	0.65	0.74	0.69

Table 4: Classification Scores - Weighted Average

RNN Model	W. Precision	W. Recall	W. F1-Score
GRU+Attention	0.70	0.70	0.69
LSTM+Attention	0.69	0.70	0.69

Table 5: Classification Scores - Macro Average

RNN Model	M. Precision	M. Recall	M. F1-Score
GRU+Attention	0.66	0.60	0.60
LSTM+Attention	0.64	0.61	0.62

In addition, the average scores of the `LSTM+Attention` and `GRU+Attention` models with attention are presented in Tables 4 & 5. Both tables indicate the improvement of the models regarding their performance. This improvement is also highlighted in Figures 7 & 8 as well as in Figures 9 & 10, in which the `ROC` and the `loss vs epoch` curves are plotted respectively. We can also observe that the self-attention mechanism has resulted in the smoothing of the `loss vs epoch` curves, eliminating the issue spotted in our first models.

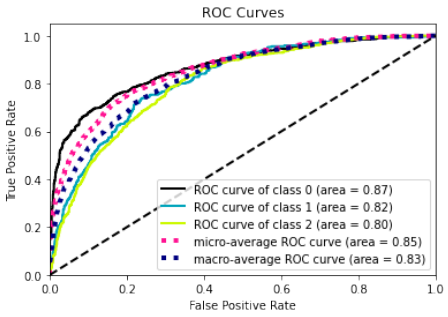

Figure 7: GRU: ROC curve

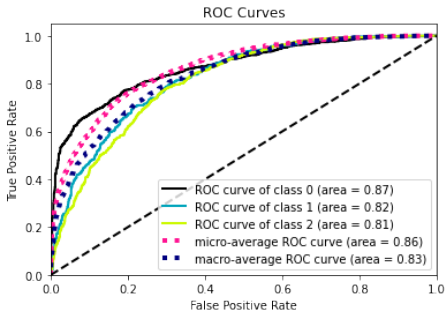

Figure 8: LSTM: ROC curve

5 Final Observations

Both of the models in this Assignment achieved similar scores, resulting in almost *67%* F1 score and *69%* when adding an `Attention` layer. These results are far better in contrast to the models of the previous assignments. In more detail, the model of the 1st Assignment resulted in *63%* F1 score, while the model of the 2nd Assignment attained *56%* F1 score.

To that extent our results are definitely affected by the imbalance nature of our dataset. Utilizing more sophisticated methods such as using the `Focal Loss`, or experimenting with other widely used techniques, such as data oversampling `(ROS)` and undersampling `(RUS)`, may have handed us better scores. Regardless, as previously stated, the models of this Assignment achieved the best scores so far.

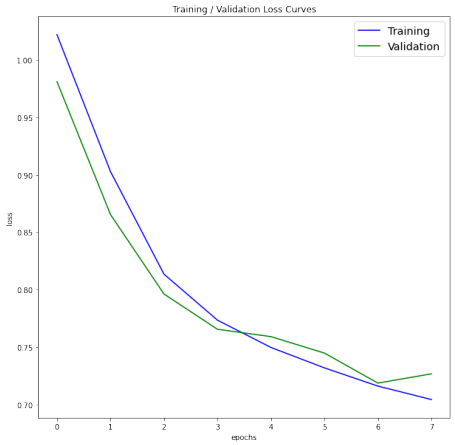

Figure 9: GRU: Loss vs Epochs

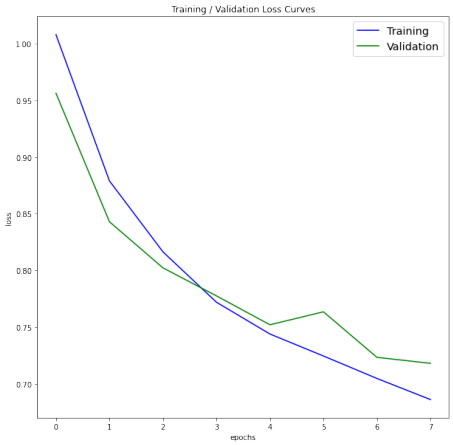

Figure 10: LSTM: Loss vs Epochs

References

[1] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors* , 2012

[2] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014

[3] Nitish Shirish Keskar, Richard Socher. *Improving Generalization Performance by Switching from Adam to SGD*, 2020

[4] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. *LSTM: A search space odyssey*, 2016

[5] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization*, 2014

[6] John Duchi, Elad Hazan, and Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*, 2011

[7] Tijmen Tieleman and Geoffrey Hinton. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.*, 2012

[8] Saptarshi Sinha, Hiroki Ohashi, and Katsuyuki Nakamura. *Class-Wise Difficulty-Balanced Loss for Solving Class-Imbalance*, 2020.

[9] Justin M. Johnson & Taghi M. Khoshgoftaar. *Survey on deep learning with class imbalance*, 2019.

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár. *Focal Loss for Dense Object Detection*, 2017.

[11] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou & Yoshua Bengio. *A structured self-attentive sentence embedding*, 2017.