1

Having a vector of the output values of the last hidden layer of a neural network as $\theta = [\theta_1, \theta_2, \theta_3 .. \theta_n]$, called logits, then the output of the softmax can be given by the equation:

$$\hat{y} = softmax(\theta)$$

and the Cross Entropy function [L] can be given from the following equation:

$$L = -\sum_{i=1}^{C} y_i \cdot log(\hat{y_i}), \tag{1}$$

where C represents the number of classes.

To calculate the derivative of L with respect to the logits, we first need to calculate the partial derivative for each θ_k:

$$\begin{aligned} \frac{\partial L}{\partial \theta_k} &= -\frac{\partial}{\partial \theta_k} \sum_{i=1}^{C} y_i \cdot log(\hat{y_i}) \\ &= -\sum_{i=1}^{C} y_i \cdot \frac{\partial}{\partial \theta_k} log(\hat{y_i}) \end{aligned} \tag{2}$$

For simplicity issues, we will also omit the index of the denominator:

$$\hat{y_i} = softmax(\theta_i) = \frac{e^{\theta_i}}{\sum_{j=1}^{C} e^{\theta_j}}$$

Then, we need to split the computation of the derivative to the following two cases:

• Calculating the derivative of a softmax output with respect to its matching input:

$$\begin{aligned} \frac{\partial}{\partial \theta_k} \hat{y_k} = \frac{\partial}{\partial \theta_\theta} softmax(\theta_k) &= \frac{\partial}{\partial \theta_k} \frac{e^{\theta_k}}{\sum_{j=1}^{C} e^{\theta_j}} \\ &= \frac{(\frac{\partial}{\partial \theta_k} e^{\theta_k}) \sum_{j=1}^{C} e^{\theta_j} - (\frac{\partial}{\partial \theta_k} \sum_{j=1}^{C} e^{\theta_j}) e^{\theta_k}}{(\sum_{j=1}^{C} e^{\theta_j})^2} \\ &= \frac{e^{\theta_k} \sum_{j=1}^{C} e^{\theta_j} - e^{\theta_k} e^{\theta_k}}{(\sum_{j=1}^{C} e^{\theta_j})^2} \\ &= \frac{e_k^{\theta}}{\sum_{j=1}^{C} e^{\theta_j}} \frac{\sum_{j=1}^{C} e^{\theta_j} - e^{\theta_k}}{\sum_{j=1}^{C} e^{\theta_j}} \\ &= \frac{e_k^{\theta}}{\sum_{j=1}^{C} e^{\theta_j}} (1 - \frac{e^{\theta_k}}{\sum_{j=1}^{C} e^{\theta_j}}) \\ &= \hat{y_k}(1 - \hat{y_k}) \end{aligned} \tag{3}$$

• Calculating the derivative of an output with respect to any non - matching input:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\hat{y_i} = \frac{\partial}{\partial\theta_\theta}softmax(\theta_i) &= \frac{\partial}{\partial\theta_k}\frac{e^{\theta_i}}{\sum_{j=1}^{C}e^{\theta_j}} \\ &= \frac{\frac{\partial}{\partial\theta_k}e^{\theta_i}\sum_{j=1}^{C}e^{\theta_j} - \frac{\partial}{\partial\theta_k}\sum_{j=1}^{C}e^{\theta_j}e^{\theta_i}}{(\sum_{j=1}^{C}e^{\theta_j})^2} \\ &= \frac{0 - e^{\theta_k}e^{\theta_i}}{\sum_{j=1}^{C}e^{2\theta_j}} \\ &= -\hat{y_k}\hat{y_i}\end{aligned} \tag{4}$$

Combining (2), (3), (4) we get:

$$\begin{aligned}\frac{\partial L}{\partial\theta_k} = -\frac{\partial}{\partial\theta_k}\sum_{j=1}^{C}y_i\cdot log(\hat{y_i}) &= -\sum_{i=1,i\neq k}^{C}y_i\cdot\frac{\partial}{\partial\theta_k}log(\hat{y_i}) + y_k\cdot\frac{\partial}{\partial\theta_k}log(\hat{y_k}) \\ &= -\sum_{i=1,i\neq k}^{C}y_i\cdot\frac{1}{\hat{y_i}}\cdot\frac{\partial}{\partial\theta_k}\hat{y_i} + y_k\cdot\frac{1}{\hat{y_k}}\cdot\frac{\partial}{\partial\theta_k}\hat{y_k} \\ &= -\sum_{i=1,i\neq k}^{C}\frac{y_i}{\hat{y_i}}\cdot(1-\hat{y_k}) + \frac{y_k}{\hat{y_k}}\cdot\hat{y_k}(-\hat{y_k}\hat{y_i}) \\ &= -\sum_{i=1,i\neq k}^{C}\frac{y_i}{\hat{y_i}}\cdot(-\hat{y_k}\hat{y_i}) + \frac{y_k}{\hat{y_k}}\cdot\hat{y_k}(1-\hat{y_k}) \\ &= -\sum_{i=1,i\neq k}^{C}-y_i\cdot\hat{y_k} + y_k\cdot(1-\hat{y_k}) \\ &= \sum_{i=1,i\neq k}^{C}y_i\cdot\hat{y_k} - y_k + y_k\cdot\hat{y_k} \\ &= \Big(\sum_{i=1,i\neq k}^{C}y_i\cdot\hat{y_k} + y_k\cdot\hat{y_k}\Big) - y_k \\ &= \sum_{i=1,i\neq k}^{C}y_i\hat{y_k} - y_k \\ &= \hat{y_k} - y_k\end{aligned} \tag{5}$$

Lastly, since y_i is a one hot encoded vector, then the $\sum_{i=1}^{C}y_i = 1$. So the derivative of L can be finally written as follows:

$$\frac{\partial L}{\partial\theta} = \begin{bmatrix}\hat{y_1} - y_1 \\ \hat{y_2} - y_2 \\ ... \\ \hat{y_n} - y_n\end{bmatrix} = \hat{y} - y \tag{6}$$

2

Given the computational graph displayed at the second part of this homework, we can deduct the following information:

- x, m, b represent the scalar inputs of the neural network
- $a = x\cdot m + b$

- the activation function y(a) = ReLU(a) = max(0,a)
- the loss function $MSE = (y - y*)^2$
- y represents the predicted output of the neural network
- y* represents the true labels

First, we need to calculate the derivative of the ReLU function. ReLU(x) is represented by the following equation:

$$ReLU(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

So, the derivative of the ReLU function can be written as:

$$\frac{\partial ReLU(x)}{\partial x} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases} \tag{7}$$

To compute all the gradients, we start by calculating the derivative of MSE with respect to y:

$$\frac{\partial MSE(y, y*)}{\partial y} = \frac{\partial (y - y*)^2}{\partial y} = 2 \cdot (y - y*) \tag{8}$$

Following the same notion, we calculate the derivative of MSE with respect to y*:

$$\frac{\partial MSE(y, y*)}{\partial y*} = \frac{\partial (y - y*)^2}{\partial y*} = 2 \cdot (y* - y) \tag{9}$$

We continue backpropagating and calculate the derivative of MSE;

* with respect to b:

$$\frac{\partial MSE(y, y*)}{\partial b} = \frac{\partial (y - y*)^2}{\partial y} \cdot \frac{\partial y}{\partial b} = \begin{cases} 0, & a \leq 0 \\ 2 \cdot (y - y*), & a > 0 \end{cases} \tag{10}$$

* with respect to m:

$$\frac{\partial MSE(y, y*)}{\partial m} = \frac{\partial (y - y*)^2}{\partial y} \cdot \frac{\partial y}{\partial m} = \begin{cases} 0, & a \leq 0 \\ 2 \cdot (y - y*) \cdot x, & a > 0 \end{cases} \tag{11}$$

* with respect to x:

$$\frac{\partial MSE(y, y*)}{\partial x} = \frac{\partial (y - y*)^2}{\partial y} \cdot \frac{\partial y}{\partial x} = \begin{cases} 0, & a \leq 0 \\ 2 \cdot (y - y*) \cdot m, & a > 0 \end{cases} \tag{12}$$

3

3.1 General

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

3.2 Dataset

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 contained 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.

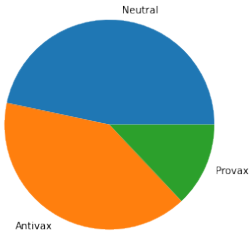

Figure 1: Train Dataset Cardinality

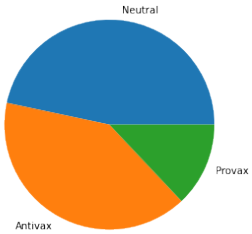

Figure 2: Validation Dataset Cardinality

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.

3.3 Data Cleaning

For the preprocessing step, all the URLs from the tweets were removed. Then, all the new line characters, double spaces and punctuation signs were deleted. Additionally, stop words were removed and the remaining ones were stemmed. In the end, each sentence was tokenized and stored to the corresponding dataframe, under the column 'tokenized_tweets'.

4 GloVe Embeddings

For this assignment the **'glove.6B.200d.txt'** embedding vectors were used. In order to load the pretrained vectors, a dictionary was constructed to hold the mappings between the words and their embedding vectors. Then, each token of the sentences in our corpus was assigned its corresponding embedding. If a token was not present in the constructed dictionary, then it was assigned a zero vector. At the end of the procedure, each tweet was represented by the *mean vector* of its tokens' embedding.

5 Design of the Feed Forward Network

For this task the following forward neural networks were designed:

1. One Layer Forward Neural Network
2. Two Layer Forward Neural Network
3. Three Layer Forward Neural Network

The 2^{nd} and 3^{rd} networks use the *ReLU* activation function between their layers. The *Sigmoid* activation function was also tested but did not hand any sufficient results.

6 Model Construction and Experiments

After extensive testing, *Cross Entropy Loss* function was determined to be the best fit for this task, as it is very useful for multi-class classification, especially in combination with *Softmax*. Due to the imbalance nature of our datasets, a 1D tensor of class weights was given as parameter to the loss function and the reduction method was specified to 'mean'. This specific loss function is selected for all the experiments presented bellow. Regarding the datasets, both train and validation sets are processed in *batches*. To do so, each dataset is fed inside a *dataloader*, and the data are reshuffled in every epoch, to avoid overfitting. The scores presented, are obtained from the validation set.

6.1 A Simple Model

For the first experiment a **Single Layer Feed Forward Network** was tested. This model aims to investigate how a really simple Feed Forward Network would perform in this task. Regarding model's hyperparameters, datasets are processed in *batches of 32* samples. Additionally, the *Stohastic Gradient Descent (SGD)* optimizer was selected with a *learning rate of 0.001*. This model, converges after *14 epochs*. These hyperparameters were selected after performing a Grid Search algorithm to determine the ones that handed in the best results. These results, are presented in Table 1.

Table 1: Classification Scores - 1L FFN

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.61	0.56	0.58	0.53	0.55	0.52

We observe that even a single layer model can achieve sufficient results after an extremely small number of epochs.

6.2 A Two Layer Model

For the second experiment, the performance of a **Double Layer Feed Forward Network** with a *hidden dimension of 64* was tested. Here, datasets were processed in *batches of 64* samples. Again, the *Stohastic Gradient Descent* (SGD) optimizer was selected with a *learning rate of 0.001*. This model, converges after *40 epochs*. The results obtained are presented in Table 2.

Table 2: Classification Scores of 2L FFN

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.60	0.56	0.58	0.52	0.55	0.52

We observe that the model achieves similar results with the previous one, however it needs more epochs to converge. Generally speaking, larger networks usually need more time to converge.

6.3 A Three Layer Model

In this section, several experiments using **Three Layer Feed Forward Networks** were performed. Datasets were processed in *batches of 32* samples and the *hidden dimensions are 128 and 64*. The goal of this section of tests is to experiment with the two most famous optimizers in litrature; *SGD* and *Adam*, as well as experiment with their hyperparameters. In literature [1], it is often stated that *SGD* generalizes better, but *Adam* converges faster. This section aims to investigate how these two optimizers perform in this task.

SGD Optimizer

SGD is a variant of Gradient Descent, that instead of performing computations on a dataset it only computes on a small subset of its data examples. In this experiment, the *learning rate* is set to *0.001*. The model, seems to converge after *50 epochs*.

Table 3: Classification Scores of 3L FFN + SGD

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.62	0.59	0.60	0.55	0.57	0.55

SGD Optimizer + Momemtum

Momentum accelerates the training process but adds an additional hyperparameter. When specified, the weights are modified through a momentum term, which is calculated as the moving average of gradients. Here, the *momentum* parameter is set to *0.9*. Keeping the configuration of the hyperparameters of the previous experiment at the same values, we observe that the model converges in only *7 epochs*, handing us the same results.

Table 4: Classification Scores of 3L FFN + SGD + Momentum

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.61	0.57	0.58	0.53	0.56	0.53

SGD Optimizer + Nesterov

Nesterov Momentum can be thought of as a modification to momentum to overcome the problem of overshooting the minima. According to [2], it is a first-order optimization method with better convergence rate guarantee than gradient descent in certain situations. Again, by keeping the hyperparameters of the previous two experiments at the same value, we get the following results after running for *6 epochs*;

Table 5: Classification Scores of 3L FFN + SGD + Nesterov

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.61	0.62	0.61	0.55	0.55	0.55

Here, is is worth to mention that the current models resulted in the smaller difference between the *precision and recall metrics* for each class, indicating that the model has more confidence in identifying the correct class for each sample. Table 6 shows the analytical classification report of the SGD optimizer experiments.

Table 6: Classification Report of 3L FFN

Model	Class	Precision	Recall	F1-Score
SGD	0	0.71	0.60	0.65
	1	0.32	0.41	0.36
	2	0.57	0.63	0.60
SGD + momentum	0	0.71	0.68	0.70
	1	0.29	0.53	0.37
	2	0.59	0.47	0.52
SGD + nesterov	0	0.69	0.72	0.70
	1	0.37	0.36	0.36
	2	0.61	0.57	0.59

Adam Optimizer

Adam is an algorithm for gradient based optimization of stochastic objective functions. It combines the advantages of RMSProp and AdaGrad, and computes individual adaptive learning rates for different parameters. However, Adam introduces two new hyperparameters and complicates the hyperparameter tuning problem. Due to the fast converge of the optimizer, *learning rate* was reduced to *0.00001*, as in other cases, the model seemed to overfit really fast. The results after a *12 epoch* run are presented bellow;

Table 7: Classification Scores of 3L FFN + Adam

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.64	0.60	0.61	0.56	0.59	0.56

We observe that compared to the *SGD* optimizer (without the hyperparameters), *Adam* converges faster and reaches slightly better results.

Adam Optimizer + AmsGrad

AMSGrad is an extension to the *Adam* version of gradient descent that attempts to improve the convergence properties of the algorithm, avoiding large abrupt changes in the learning rate for each input variable [4].

Table 8: Classification Scores of 3L FFN + Adam + AmsGrad

Avg. Precision	Avg. Recall	Avg. F1-Score	Macro Precision	Macro Recall	Macro F1-Score
0.64	0.59	0.60	0.56	0.59	0.55

The final model reaches the best scores so far after running for *12 epochs*.

7 Model Selection

Due to the high scores as well as its fast converge, a 3 Layer Feed Forward Network with Adam + AmsGrad optimizer and learning rate of 0.00001, was selected as the most appropriate model for this task. The roc curve and loss plots are presented in Fig. 3, 4 & 5.

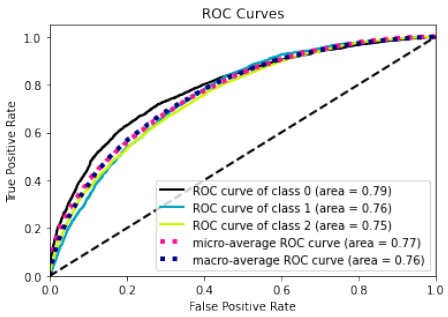

Figure 3: ROC Curve of Train Dataset

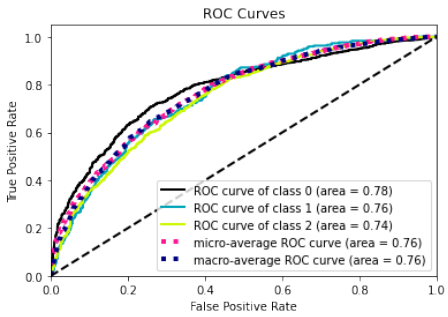

Figure 4: ROC Curve of Validation Dataset

In general, ROC curves indicate how much the model is capable of distinguishing between classes. Here, we notice that our model is confident enough in distinguishing the three classes, handing close 'area scores' between classes. We also observe that the model results in almost similar curves in both datasets, indicating that it is able to generalize. Regarding the loss plot, we can detect that the model converges between the *11th-12th epoch*, while during the next epochs it starts to slightly overfit.

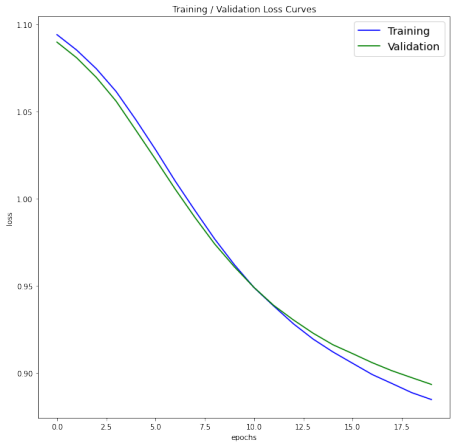

Figure 5: Train / Validation Loss Curves

8 Final Observations

In contrast to the model of Assignment 1, this model seems to achieve slightly worse results, as the macro F1 measure of the previous one was almost 0.63 while this one is closest to 0.56. There are several possible explanations for this outcome:

- The imbalanced nature of the dataset prohibits the model to correctly classify the samples and especially the ones of the minority class. One way to overcome this, could be to implement a custom sampler, which balances the batches of the dataset.
- The use of mean vectors instead of the commonly used padding technique, may have resulted in the loss of information of the embeddings.
- Maybe, a more sophisticated hyperameter tuning procedure needs to be performed.

References

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

[2] Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton *On the importance of initialization and momentum in deep learning*, 2013.

[3] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro and Benjamin Recht *The Marginal Value of Adaptive Gradient Methods in Machine Learning*, 2018

[4] Sashank J. Reddi, Satyen Kale, Sanjiv Kumar On the Convergence of Adam and Beyond, 2018