Artificial Intelligence II

Nefeli Tavoulari

Fall Semester 2021

1

The Cross Entropy loss function for one instance is defined as:

$\mathcal{J}(y, \hat{y})$ = - $\sum_{k=1}^{K}(y_k \log(\hat{y}_k))$

where K is the number of classes and $\hat{y}$ is the softmax function.

Calculating the derivative of Cross Entropy with respect to the logit j:

$\frac{\partial \mathcal{J}}{\partial \theta_j} = -\frac{\partial}{\partial \theta_j} \sum_{k=1}^{K}(y_k \log(\hat{y}_k))$

$\frac{\partial \mathcal{J}}{\partial \theta_j} = -\sum_{k=1}^{K} y_k \frac{\partial}{\partial \theta_j} \log \hat{y}_k$, where:

$\log \hat{y}_k = \log(\frac{e_i^\theta}{\sum_{k=1}^{K} e_k^\theta}) = \theta_i - \log(\sum_{k=1}^{K} e_k^\theta)$

Now:

$\frac{\partial(\theta_i - \log(\sum_{k=1}^{K} e_k^\theta))}{\partial \theta_j} = \frac{\partial \theta_i}{\partial \theta_j} - \frac{1}{\sum_{k=1}^{K} e_k^\theta} * \frac{\partial \sum_{k=1}^{K} e_k^\theta}{\partial \theta_j} =$

$-\frac{e_i^\theta}{\sum_{k=1}^K e_k^\theta} + \frac{\partial\theta_i}{\partial\theta j} = -\hat{y} + \frac{\partial\theta_i}{\partial\theta j}$, where,

if j=i then $\frac{\partial\theta_i}{\partial\theta_j} = 1$, otherwise $\frac{\partial\theta_i}{\partial\theta} = 0$

So, $\frac{\partial\mathcal{J}}{\partial\theta_j} = -\sum_{k=1}^K y_k(\{i == k\} - \hat{y})$

$= -\sum_{k=1}^K y_k\{i == k\} + \sum_{k=1}^K y_k\hat{y}$

$= -y_j + \sum_{k=1}^K y_k\hat{y}$

$= -y_j + \hat{y}\sum_{k=1}^K y_k$

$= -y_j + \hat{y}\sum_{k=1}^K y_k$

$= -y_j + \hat{y}$

since the one-hot encoded vector y has a sum equal to 1.

So, $\frac{\partial\mathcal{J}}{\partial\theta} = \hat{y} - y$ and the same applies if we have m instances. In that case $\hat{y} - y$ is a vector.

2

Following the stages of the computational graph we get:

$a = x * m + b$
$y = RELU(a) = max(0, a) = max(0, x * m + b)$
$f = LOSS = (y - y*)^2 = (max(0, a) - y*)^2 = (max(0, x * m + b) - y*)^2$

So, now I have to compute the gradients with respect to x,m,b,y*:

$\frac{\partial x}{\partial x} = \frac{\partial m}{\partial m} = \frac{\partial b}{\partial b} = \frac{\partial y*}{\partial y*} = 1$

$\frac{\partial a}{\partial x} = m$ $\frac{\partial a}{\partial m} = x$ $\frac{\partial a}{\partial b} = 1$ $\frac{\partial a}{\partial y*} = 0$

Using the chain rule:

$\frac{\partial y}{\partial x} = \frac{\partial max(0,a)}{\partial x} = 0$ or m $(= \frac{\partial a}{\partial x})$

$\frac{\partial y}{\partial m} = 0$ or x $(= \frac{\partial a}{\partial m})$

$\frac{\partial y}{\partial b} = 0$ or 1 $(= \frac{\partial a}{\partial b})$

$\frac{\partial y}{\partial y*} = 0$

$\frac{\partial f}{\partial x} = \frac{\partial (max(0,a)-y*)^2}{\partial x} = \frac{2*\partial (max(0,a)-y*)}{\partial x} * \frac{\partial y}{\partial x} =$

0 or $2*(x*m+b-y*)*\frac{\partial a}{\partial x} = 2*(x*m+b-y*)*m$

$\frac{\partial f}{\partial m} = 0$ or $2*(x*m+b-y*)*\frac{\partial a}{\partial m} = 2*(x*m+b-y*)*x$

$\frac{\partial f}{\partial b} = 0$ or $2*(x*m+b-y*)*\frac{\partial a}{\partial b} = 2*(x*m+b-y*)$

$\frac{\partial f}{\partial y*} = 0$ or $2*(x*m+b-y*)*\frac{-\partial y*}{\partial y*} = -2(x*m+b-y*)$

(0 if y = 0)

3 Vaccine Sentiment Classification

In this notebook I trained a neural network, which classifies tweets as Neutral, Pro-vax or Anti-vax.

3.1 Preprocessing

First of all, I performed some preprocessing and feature extraction on the data, similar to the one I had done in the previous assignment. I removed all empty or duplicate tweets, as they offer no useful information to the model, I lowercased tweets, performed lemmatization, removed stopwords, special characters, emojis, words written in a non-latin alphabet and urls, since I noticed that the results are better this way. Then, I used the GloVe pretrained Word Embeddings, to take advantage of the relationships between the words. More precisely, I used the ones that are extracted from tweets, since we are dealing with tweets too and I experimented with all the dimensions. I started by finding every word of each tweet in the embeddings dictionary and then I find the average for every tweet (I sum each tweet's words' embeddings and I divide the sum by the dimension of the embeddings), in order to end up with a matrix of $numberOfInstances * dimension$. Then, I split the dataset into batches.

3.2 Models

The best model I created consists of 3 linear layers, the input size is equal to the dimension of the embeddings and the output size is equal to the number of classes (3). After experimenting with many different combinations of activation functions I found the best results using ReLU and Sigmoid in-between the hidden layers.I did not use an activation function in the output, since I use Cross Entropy as loss function, which combines nn.LogSoftmax() and nn.NLLLoss(). Moreover, I used Batch Normalization, which seemed to ameliorate my results. As far as the optimizer is concerned I used Adam, since it gave much better results than SGD or Adagrad.

The accuracy/precision/recall/F1 results I got were the following, with learning rate=1e-4 and SGD with momentum=0.2 and weight decay=0.01:

	SGD	*Adagrad*	*Adam*
Train	0.60424	0.55884	**0.66878**
Validation	0.61481	0.56310	**0.65644**

Here we observe that there is also something wrong with the SGD and Adagrad results, since the accuracy on the validation set is higher than that of the train set.

Some other results I had to compare, concerning the number of units, hidden layers and activation functions:

	$4-layers-no-activation$	$4-layers-activation$
$Train$	0.60638	**0.69265**
$Validation$	0.60648	**0.64636**
	$3-layers-no-activation$	$3-layers-activation$
$Train$	0.60581	**0.66878**
$Validation$	0.61042	**0.65644**
	$2-layers-no-activation$	$2-layers-activation$
$Train$	0.60884	**0.65228**
$Validation$	0.61481	**0.64241**

It is obvious that the activation functions (ReLU, Sigmoid) and the Batch Normalization help the model become more expressive, in comparison with using just linear layers. Also, we notice that the model with the 3 layers has the best results for the validation set. The number of units of the hidden layers of the 4-layer model is 256, 128, 64, of the 3-layer model is 128, 64 and of the 2-layer one is 64.

Using different number of units (in hidden layers) for the 3-layer model:

	$[128, 64]$	$[256, 128]$	$[64, 32]$
$Train$	**0.66878**	0.60884	0.65568
$Validation$	**0.65644**	0.61481	0.64110

Also, I experimented with some more activation functions, eg softmin, logsoftmax, or dropout, and higher or lower learning rate but still the best results were the ones I mentioned before.

The results with or without Dropout:

	without	*after − relu*	*after − sigmoid*	*both*
Train	**0.66878**	0.65247	0.66280	0.64920
Validation	**0.65644**	0.65512	0.65074	0.64811

Testing the best model with different dimensions of word embeddings:

	25*d*	50*d*	100*d*	200*d*
Train	**0.66878**	0.73729	0.83023	0.94861
Validation	**0.65644**	0.67134	0.67440	0.67090

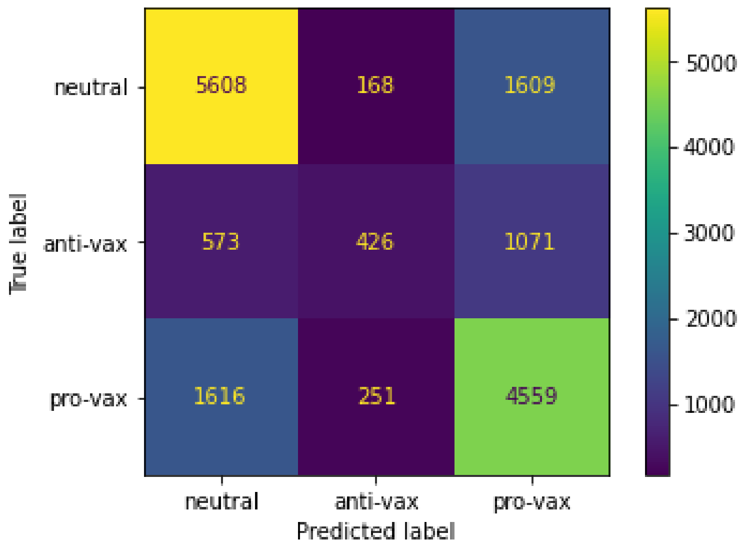
Even though the results seem better for the three last dimensions of embeddings, the corresponding models are overfitting, which is obvious by the loss and accuracy plots, but also by the fact that the training results are much better than the validation ones. So, my best model uses 25d word embeddings.

In the notebook I have also included another model. The main difference this model has from the first one is that it contains one more layer, it uses Dropout and LogSoftmax on the output, because the used loss function is NLLLoss. The results are similar to the ones of my best model:

	Accuracy	*Precision*	*Recall*	*F*1
Train	**0.65499**	0.65499	0.65499	0.65499
Validation	**0.65030**	0.65030	0.65030	0.65030

3.3 Evaluation - Comparison

Using different metrics, precision, recall, f1 score, accuracy, confusion matrix, roc auc, I was able to determine the quality of my model. From the confusion matrix, I understand that my model makes good enough predictions, since the diagonal has quite bigger numbers. I observe also that it does not predict that well the anti-vaxers, but that is expected, because the anti-vaxers in the data are very few, in comparison to the other classes. The confusion matrix ,using the aforementioned model :



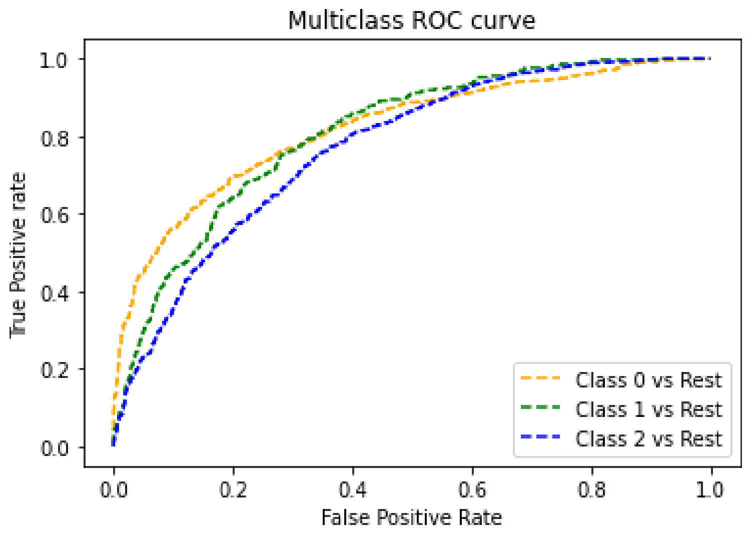
The results as I mentioned them before:

	Accuracy	*Precision*	*Recall*	*F1*
Train	**0.66878**	0.66878	0.66878	0.66878
Validation	**0.65644**	0.65644	0.65644	0.65644

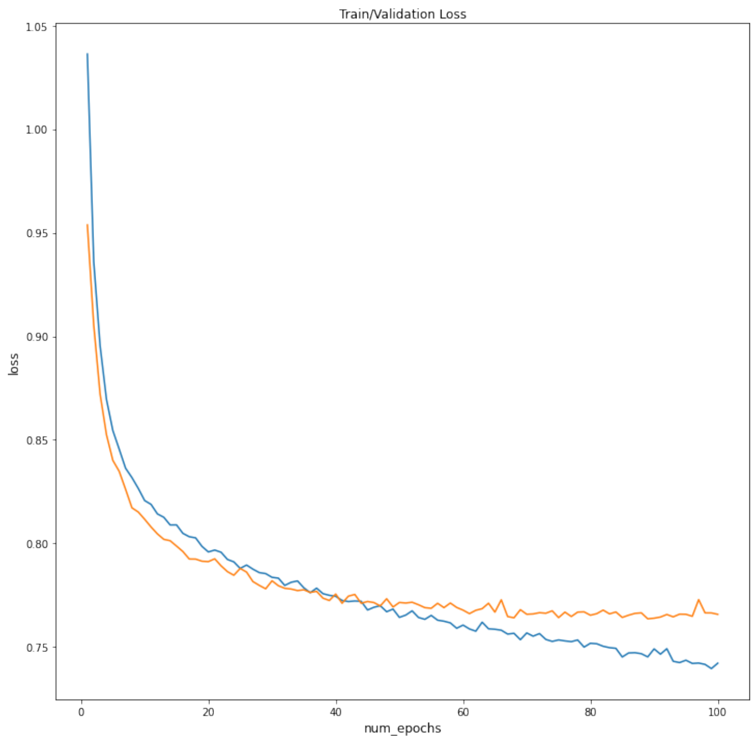
One-vs-One ROC AUC scores:
0.783038 (macro),
0.789281 (weighted by prevalence)

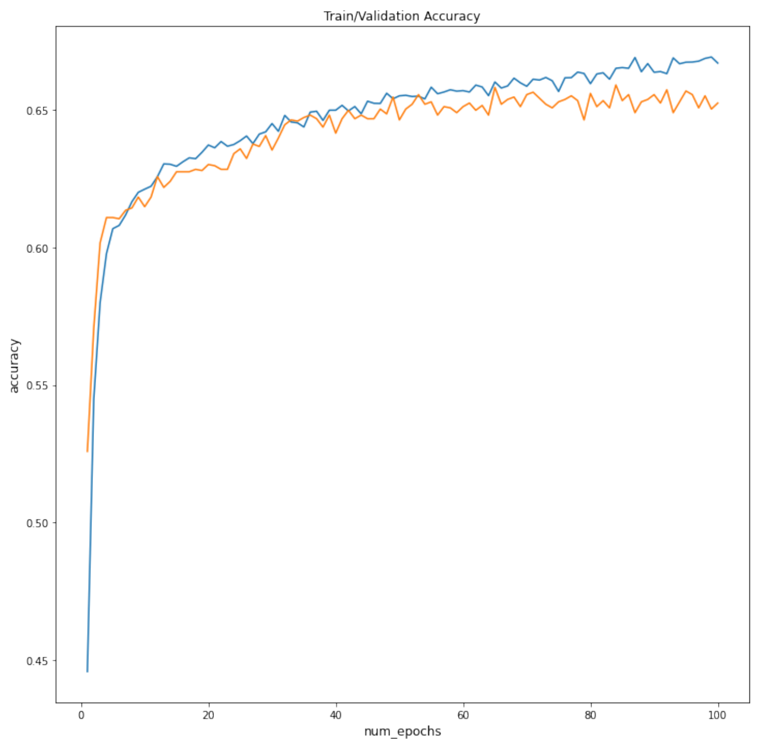
One-vs-Rest ROC AUC scores:
0.796980 (macro),
0.796628 (weighted by prevalence)

And some plots:



We observe that the anti-vax class is not that well predicted as the other two classes. However, the curves in general show that the model distinguishes the classes, since they lean towards the top left corner (TPR).


Train/Validation Loss
loss
num_epochs
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0
20
40
60
80
100



We observe that after approximately the 40th epoch, the model does not learn that much, the loss increases and the accuracy descreases, on the validation set. In general, however, the behavior of the curves is the expected one.That means that the accuracy increases and the loss decreases as the model learns.

Finally, I want to note that the results are sligthly worse than the ones obtained by the Logistic Regression model of the previous assignment. One reason behind this could be the fact that some words related to Covid-19 are probably not present in the Embeddings. Another reason could be that a more clever neural network is needed (RNN,LSTM,GRU, etc).

References

[1] PyTorch Documentation