Artificial Intelligence II

Nefeli Tavoulari

Fall Semester 2021

1 Vaccine Sentiment Classification

In this notebook I trained two bidirectional stacked Recurrent Neural Networks with LSTM/GRU cells, which classify tweets as Neutral, Pro-vax or Anti-vax.

1.1 Preprocessing

First of all, I performed some preprocessing on the data, similar to the one I had done in the previous assignments. I removed all empty or duplicate tweets, as they offer no useful information to the model, I lowercased tweets, removed 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, I experimented with all the dimensions and I finally used the 100 dimensional embeddings. In this assignment, I took advantage of the Torchtext Machine Learning Framework, that is provided by PyTorch, I built the vocabulary based on training and validation data and I ended up with a matrix of $numberOfUniqueWords * EmbeddingsDimension$. Then, I split the data into batches of 32 instances, since that seems to be giving the best results.

1.2 Models

- The best bidirectional stacked LSTM model I created consists of an embedding layer, 3 recurrent layers, with hidden size of 64 and of course, a linear layer at the end. Also, I used Cross Entropy Loss to evaluate the model, Adam with learning rate of 1e-4 as optimizer and gradient clipping with max norm of 2, in order to avoid the problem of exploding gradients. Moreover, I used batches of 32 and 40 epochs to train the model.

- The best bidirectional stacked GRU model I created consists of an embedding layer, 3 recurrent layers, with hidden size of 128 and of course, a linear layer at the end. Also, I used Cross Entropy Loss to evaluate the model, Adam with learning rate of 1e-4 as optimizer and gradient clipping with max norm of 2, in order to avoid the problem of exploding gradients. Moreover, I used batches of 32 and 40 epochs to train the model.

After preprocessing the data, using 2 LSTM/GRU layers, 64 hidden size, bidirectional RNNs,no gradient clipping,learning rate of 1e-4, batch size of 64 and vocabulary built on both training and validation data, the accuracy/precision/recall/F1 results I got, experimenting with different configurations, were the following on the validation data:

	LSTM	*GRU*
200*d*	0.6038	0.6362
100*d*	0.6529	**0.6538**
50*d*	**0.6555**	0.6437
25*d*	0.5626	0.6354

So, using the 100d embeddings:

	LSTM	*GRU*
without preprocessing	0.5836	0.5990
vocabulary without validation data	0.5985	0.6327
with preprocessing and validation data in vocab	**0.6529**	**0.6538**

Then, with preprocessing of tweets, to avoid noise and using training and validation data to build the vocabulary:

	LSTM	*GRU*
batch size 64	0.6529	0.6538
batch size 32	**0.6752**	**0.6779**

Using batch size 32:

	LSTM	*GRU*
2 *recurrent layers*	0.6752	0.6779
3 *recurrent layers*	**0.6858**	**0.6871**

Using 3 layers:

	LSTM	*GRU*
128 *hidden size*	0.6822	**0.6954**
64 *hidden size*	**0.6858**	0.6871

Using 64 hidden size for recurrent layers in LSTM and 128 hidden size for recurrent layers in GRU:

1e − 4 learning rate	**0.6858**	**0.6954**
1e − 3 learning rate	0.6779	0.7068(*overfitting*)

So, we observe the importance of finding the appropriate learning rate, since the results got much ameliorated with the smaller one. Using learning rate of 1e-4:

	LSTM	*GRU*
no clip	0.6858	0.6954
clip with max norm 1	0.6853	0.6792(*overfitting*)
clip with max norm 2	**0.7050**	**0.7063**
clip with max norm 5	0.6801	0.6678(*overfitting*)

Using gradient clipping with a max norm of 2, to mitigate the exploding gradients problem :

	LSTM	*GRU*
0.2 *dropout after all LSTM layers*	0.6783	0.6831(*overfitting*)
0.5 *dropout after all LSTM layers*	0.6770	0.6577
except last one	0.6787	0.6945(*overfitting*)
0.2 *dropout only after last LSTM layer*	0.6818	0.6901(*overfitting*)
0.2 *dropout after linear layer*	0.6546	0.6945
relu and dropout	0.6858	0.6879(*overfitting*)
without anything	**0.7050**	**0.7063**

So, the model performs better without any activation function or dropout.

Lastly, another comparison using skip connections with hidden size of 100, in order to avoid the problem of layers not being able to learn even identity mappings:

	LSTM	*GRU*
with skip connections	0.4014	0.4022
without skip connections	**0.7050**	**0.7063**

Paradoxically, the model is overfitting using the skip-connections technique.

1.3 Attention

Using an attention layer, so that the model focuses on one part each time:

LSTM	*GRU*
0.7063(*overfitting*)	0.6700(*overfitting*)

Here, I have to remark that for every "overfitting" note, I mean that the model seemed to be overfitting, looking at the plots of Epochs Vs Accuracy or Epochs Vs Loss, as the validation loss was incrementing and the validation accuracy decrementing, while training.

So, unfortunately, the attention layer did not improve my models' results, but it caused overfitting, as observed by the plots.

To address this issue, I experimenting with Batch Normalization and Layer Normalization, but neither of these seemed to work.

1.4 Evaluation - Comparison

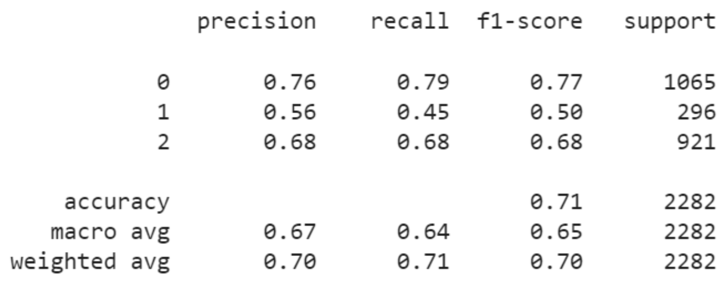
Using different metrics, precision, recall, f1 score, accuracy, confusion matrix, roc auc, I was able to determine the quality of my models. Therefore, the results are the following on the validation data:

	Accuracy	*Precision*	*Recall*	*F*1
LSTM	0.7050	0.7050	0.7050	0.7050
GRU	0.7063	0.7063	0.7063	0.7063

	AUC macro OVO	*AUC weighted OVO*	*AUC macro OVR*	*AUC weighted OVR*
LSTM	0.827906	0.832405	0.836991	0.837574
GRU	0.813940	0.819427	0.821371	0.829215

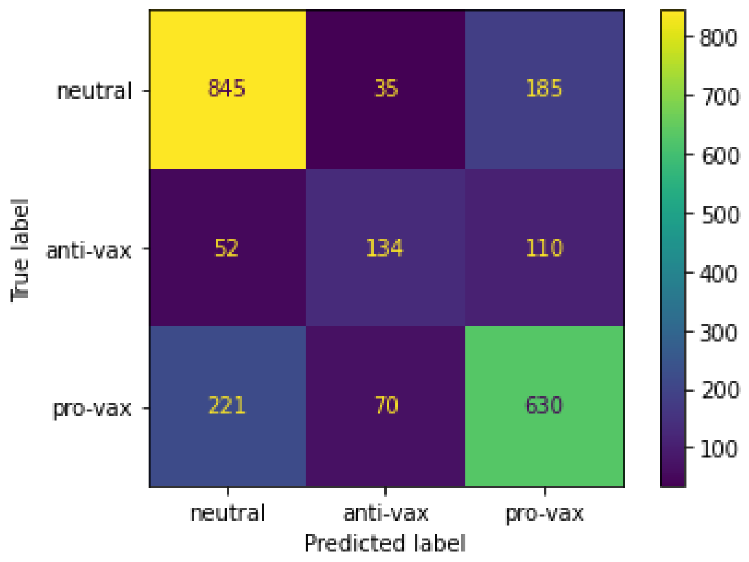
All in all, I consider the LSTM model to be better, because the GRU is slightly overfitting and the ROC scores are not as good.

Here we can also see the results for all classes for the LSTM model:

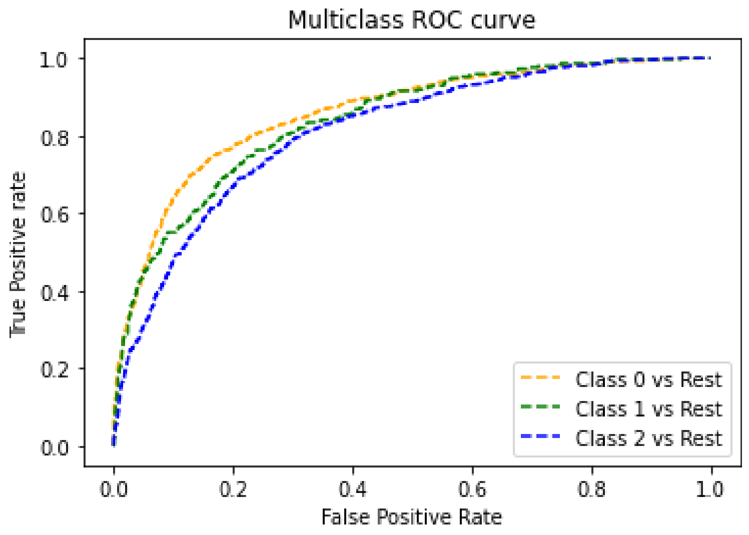


```
              precision    recall  f1-score   support

           0       0.76      0.79      0.77      1065
           1       0.56      0.45      0.50       296
           2       0.68      0.68      0.68       921

    accuracy                           0.71      2282
   macro avg       0.67      0.64      0.65      2282
weighted avg       0.70      0.71      0.70      2282
```

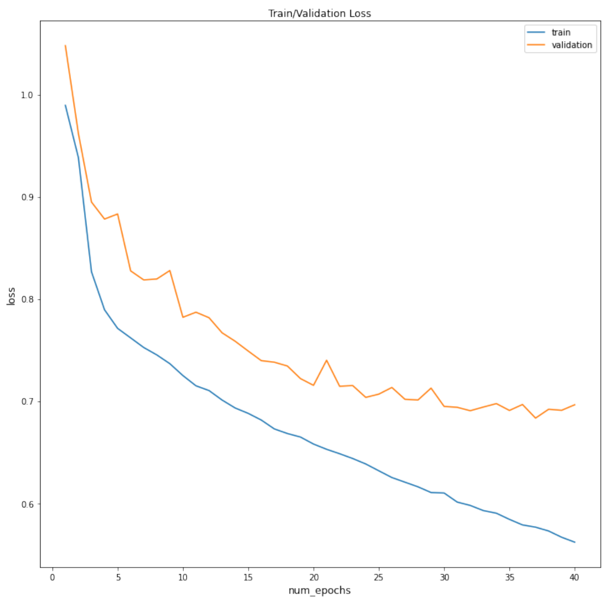
From the confusion matrix, I understand that my LSTM model makes good enough predictions, since the diagonal has quite bigger numbers even for the anti-vaxers who are very few in the data, in comparison to the other classes. The confusion matrix ,using the aforementioned model :

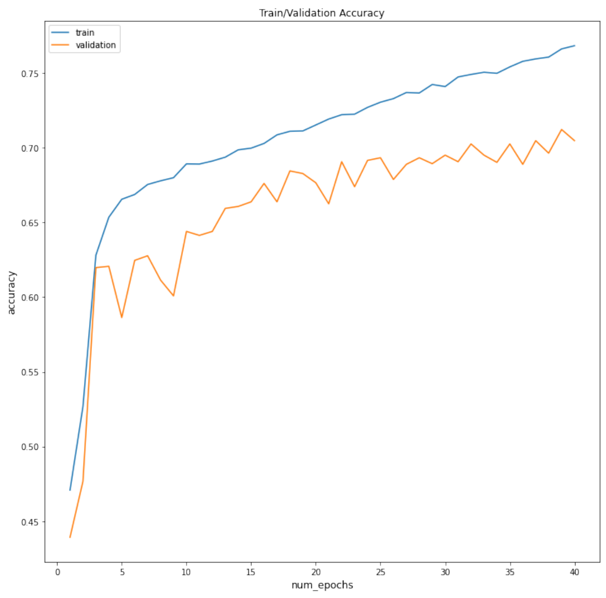


The ROC Curve for all the classes:



We observe that the anti-vax class is not that well predicted as the other two classes, which is what we expected and thus the results are realistic. The curves in general show that the model distinguishes the classes, since they lean towards the top left corner (TPR).





We observe that after some point, 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 first assignment, but better than the ones obtained by the feedforward network of the second assignment. One

reason behind this could be the fact the RNN models are too complex for such a task and of course the dataset is too small to get some good accuracy.
The results out of the 3 models are the following:

AUC	*macro OVO*	*weighted OVO*	*macro OVR*	*weighted OVR*
LSTM	0.825968	0.828989	0.832075	0.831839
feedforward	0.778982	0.785546	0.793916	0.793266
logistic regression	0.846171	0.850188	0.855235	0.854823

	accuracy	*recall*	*f1*	*precision*
LSTM	0.7050	0.7050	0.7050	0.7050
feedforward	0.6520	0.6520	0.6520	0.6520
logistic regression	0.7230	0.7230	0.7230	0.7230

The fact that the RNN model performs better than the feedforward one is very encouraging. Also, the results are satisfactory, considering we are experimenting for the first time with all these complex and interesting concepts.

References

[1] PyTorch Documentation

[2] Torchtext Documentation

[3] Attention

[4] Skip Connections

[5] LSTM example

[6] Skip Connections with Layer Normalization