AI2-Project 3

Gkika Violetta -111520160022

January 29, 2022

Exercise 1

For these models I used Adam as an optimizer and the cross-entropy loss function to calculate the loss as advised. The preprocess procedure is the same as in the two previous projects. (removed tags,hashtags,links,numbers,stop words empty tweets and lowercased all the tweets.) Also I have used GloVe pre-trained word embeddings as the embeddings of the inputs of my models. In the file .ipynb is shown my best model which uses the parameters as described below. Also in the file are shown precision, recall and F1 for each class , the learning curve plot for training and validation data as well as the ROC curve.

Best model

There you can find a general idea about the chosen hyperparamters. Next section explains briefly how I concluded on these hyperparameters.

0.1 Number of stacked RNNs

For the best model the number of layers that seemed to be better was 2. This is a relatively small value but as the numbers of layers increased the performance decreased.

0.2 Hidden Size

I experimented with values in range of 32 up to 512 (all values were powers of 2 (2^k)). And seems that the best value was 128.

0.3 Type of cells (LSTM/GRU)

In general models with same hyperparameters values but different types of cells (LSTM/GRU) did not have significant differences but I ended up choosing LSTM for my best model.

0.4 Bidirectional

All the layers of the model were bidirectional.

0.5 GloVe dimesnions

The more the dimensions of GloVe pretrained vector increased the better the model become. I ended up choosing 300 dimensions for each vector but the model will perform also very well for other dimensions too.

0.6 Epochs

I concluded that 10 epochs are enough to train the model but I chose to train my best model with 20 epochs to improve the F1 score.After 20 epochs the "validation-train curve gap" become larger and the model tend to overfit.

0.7 Learning Rate

A small learning rate in combination with a small batch size worked for the model. Exact value equals 0.000015. Note that if we increase the batch size is better to decrease the leraning rate.

0.8 Gradient Clipping

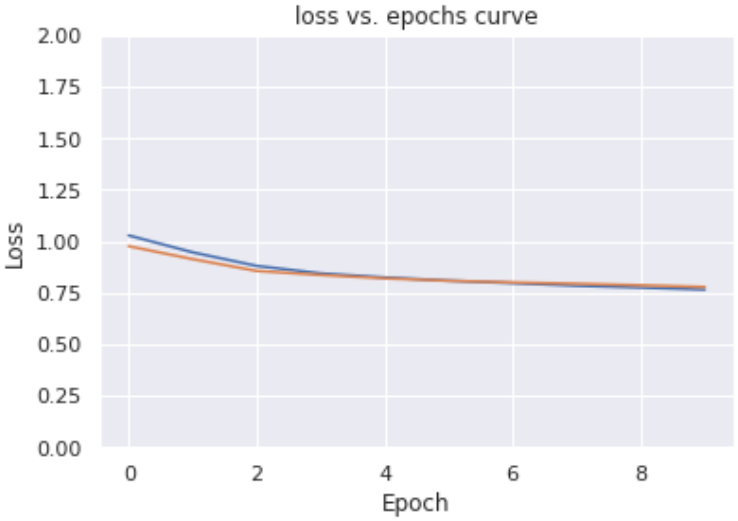
Gradient clipping definitely had an impact on the training process even though its impact depends a lot on the other hypeparameters too. I chose to use gradient clipping by norm with a max_norm = 5 and type_norm = 2.

0.9 Dropout Probability

A dropout probability equal to 0.2 worked best. Increasing the probability the model tend learn slowly or not at all and if no dropout layer is applied the model ,as expected, overfits.

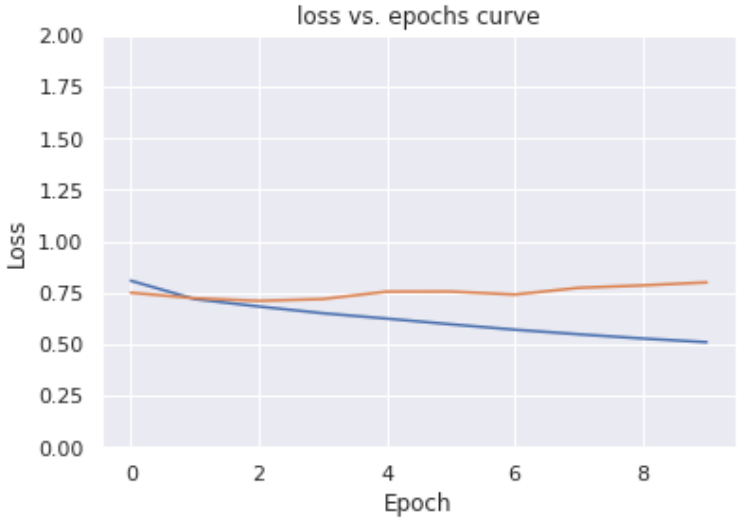
Notes about model's improvement procedure.

Firstly, to get a general idea about how RNN works on our data, I tried a very simple model with just one layer, and not "very strong" hyperparamters (a small number of hidden size , no dropout, ralativly small learning rate, no gradient clipping) and the model performed quite well.



PRECISSION SCORE= 0.650476403510426
RECALL SCORE= 0.6581722319859402
F1 SCORE 0.636277191026107

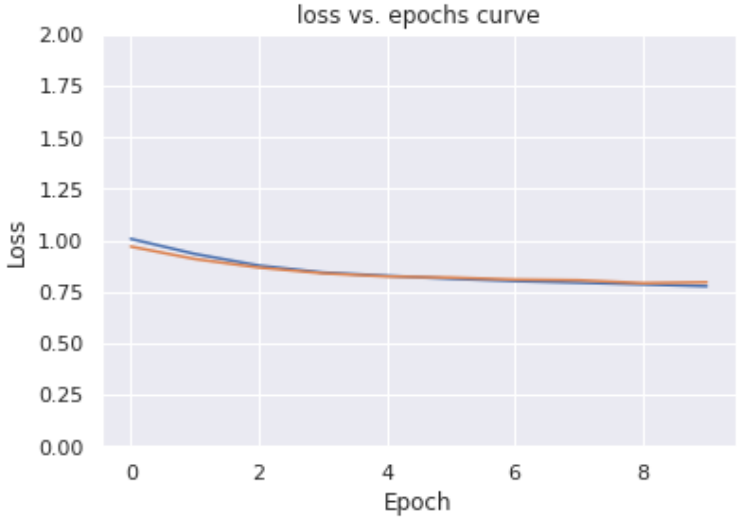
Trying to increase the learning rate the model tend to overfit thats how I decided to use a small learning rate and add a dropout layer on my model.



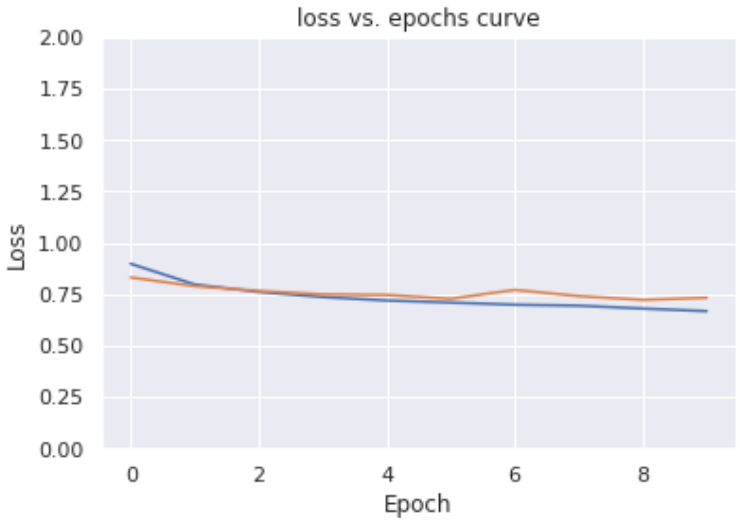
PRECISSION SCORE= 0.6937863558173307
RECALL SCORE= 0.695518453427065
F1 SCORE 0.6926832413622759

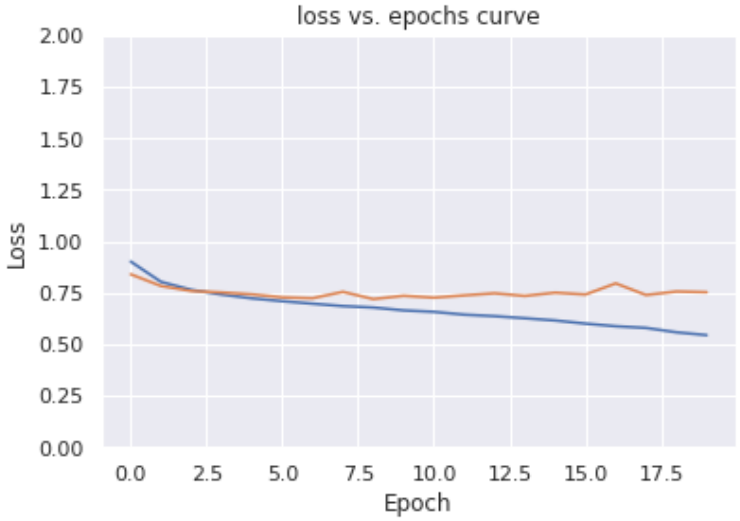
This experiment is a strong indicator of how much the learning rate affects the model.

After dropout layer.

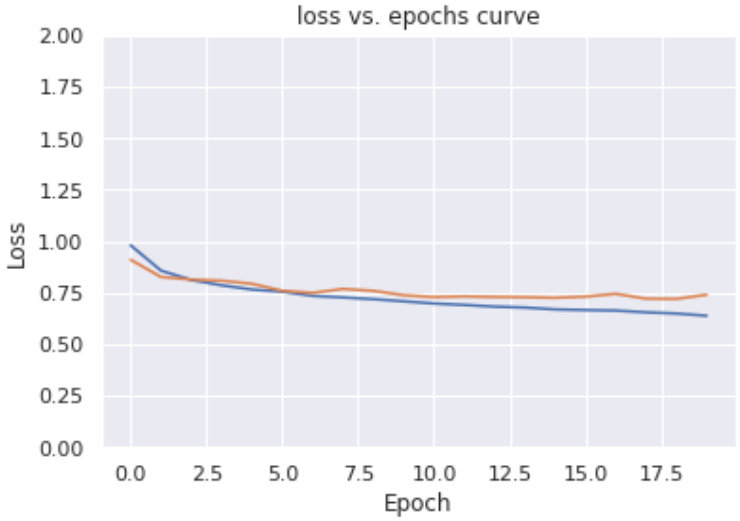


I thought that even though the loss and F1 score values are good enough would be better to lower the loss a bit more. So I tried to make the model more complex by changing the number of stacked layers and the number of hidden size, because from theory we learned that more complex models tend to perform better. So I tried hidden size= 512 and 6 layers. The model tend to overfit and if we increase the number of epochs this becomes clearer.

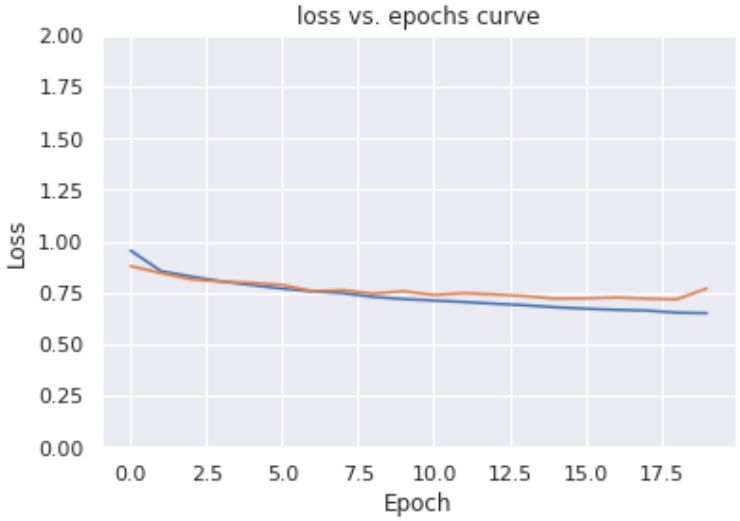


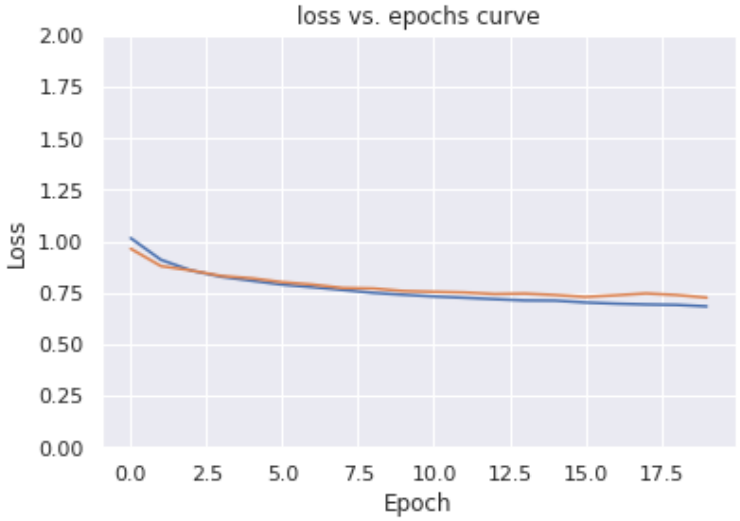


So to solve this, at first I increased the dropout probability and number of batches because as we know small number of batches chunks the data in smaller pieces and the model learns faster.

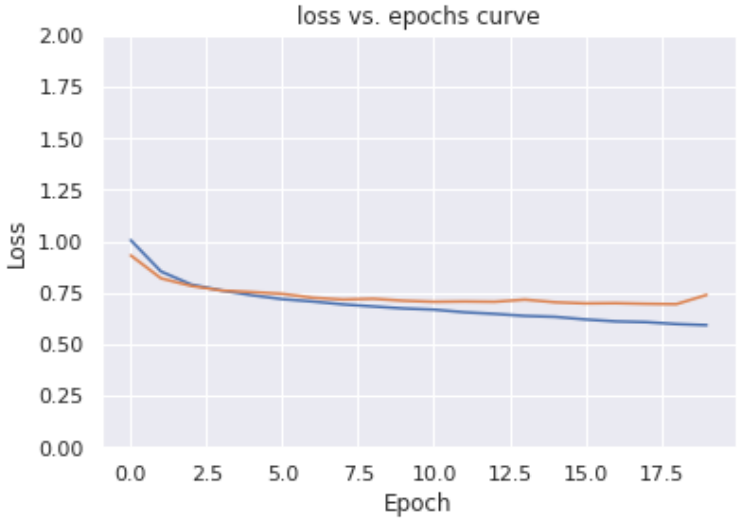


I made some experiments with different dropout probabilities but as the probability increased the model performed worse, meaning it does not learn fast enough,so I concluded to the value = 0.2.
As for the size of batches it "works together" with the learning rate, if I increased the size of batches and decreased the learning rate it was almost the same as decreasing the batch size and increasing the learning rate.
The overfiting problem is not solved though so I tried making the model less complex.

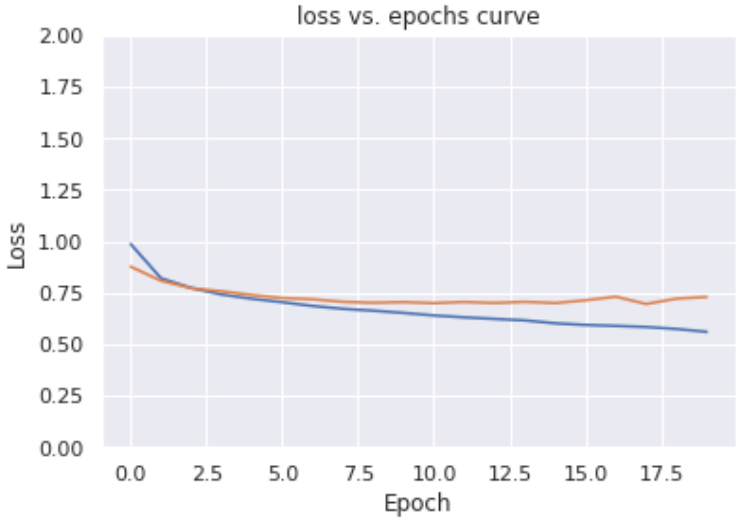




Some experiments with the length of vocabulary of pre trained glove model.

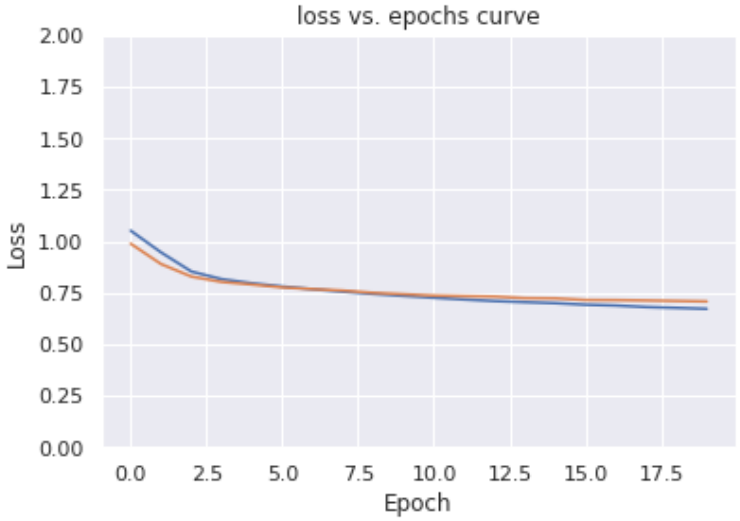


200 dimensions



300 dimensions

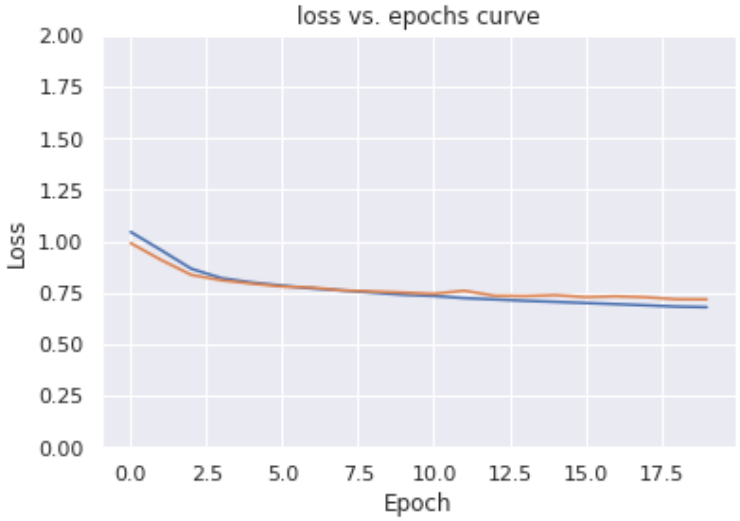
As we can see the larger GloVe dimensions helps the model learn more and faster but it tend to overfit. To avoid this problem I tried decreasing the learning rate but even though that helped with the overfiting problem the model do not learn so much so it gave us a lower F1 score (60). Even with more epochs the loss value do not decrease much because the model do not learn enough. Experimenting with the complexity of the model and the number of batches the model either overfits or do not learn enough and gives a low accuracy. So, a less complicated model with larger GloVe dimensions is definitely better. After some more experiments I concluded that the best model is the one shown below(hyperparameters same as the ones explained in the beginning of the document).



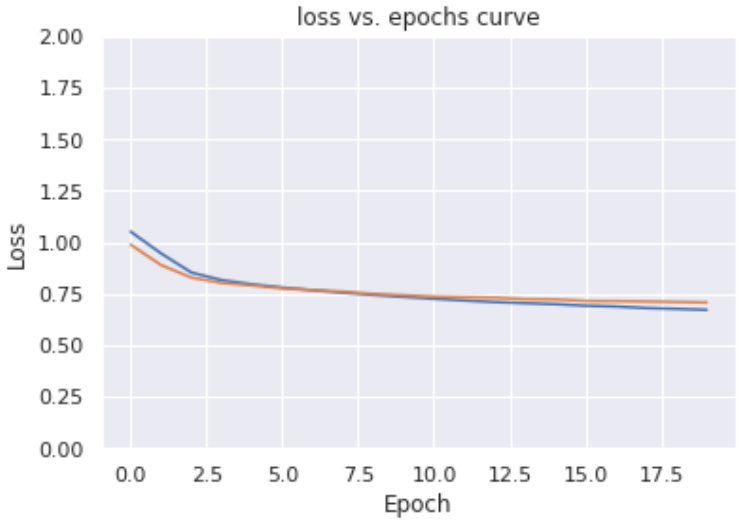
no clipping

PRECISSION SCORE= 0.675713589844091
RECALL SCORE= 0.6862917398945518
F1 SCORE 0.6760705621237717

Something interesting in this process was that I noticed the problem of exploding gradients. In the final best model this problem do not so ... but in some experiments I did when I printed the gradient norm the values ranged from 0.5 up to 5. The average value of gradient norms is advised to be a good value for c (the clipping hyperparameter) but in my experiments a smaller value proved to be more effective and it really helped the model perform better. As for the "best model" the gradient norm range between 0.1 to 1 so I decided to use clipping by norm with max_norm = 5 and norm_type = 2 (L2 Regularization). Even though the differences was not so ... it had an impact on the learning curve.

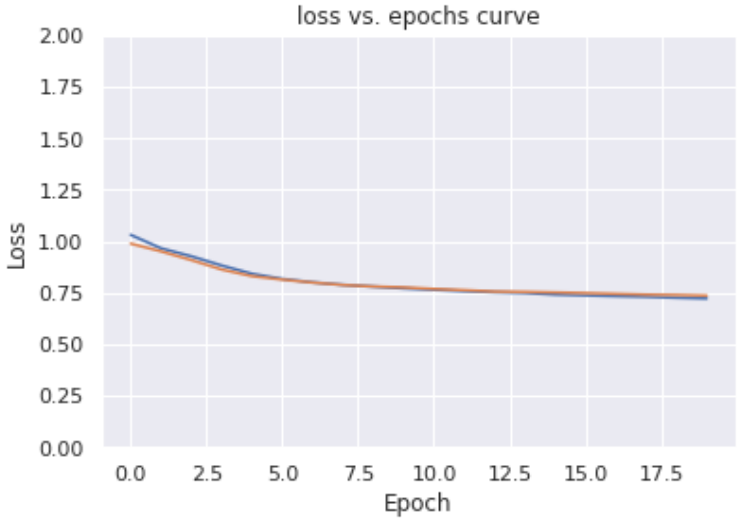


no clipping



clipping

Some experiments with GRU models.
For GRU models I reversed the experimenting process and started by testing how the "best LSTM model" performed if we changed LSTM layers to GRU. I seemed to me that GRU and LSTM performed "the same",



PRECISSION SCORE= 0.6744113688866602
RECALL SCORE= 0.679701230228471
F1 SCORE 0.653175319051565
Not only for "the best" hyperparameters but even when I tried some of the above experiments I did with LSTM layers for GRU layers it gave me almost the same results but GRU seemed to train faster. My idea at first was that GRU model would perform very differently because of the less complex structure, but that proved false and this maybe has to do with our dataset. LSTM gave a little better accuracy so I chose to use LSTM layers on my best model.

Comparisons

In contrast with the previous models here we get a higher validation score. This maybe has to do with the number of epochs because in this Project I ended up training the model for 20 epochs whereas the previous models were trained for 10 epochs. Project two score was around 63% and this one has a score around 65%. What is for sure is that this model converges faster and "better" that the previous two models. For example if we train this model for 10 epochs we get a similar score as the previous project (around 62.5%) but if we train the previous project for 20 epochs the model overfits something that do not happen in this model (in contrary it is helpful).
I was expecting that an RNN model will perform noticeably better than the previous model, because NLP problems, as we were tought in theory, are often

solved with LSTM/GRU models. Also LSTM models are dynamic as they keep words in memory and also the information "travels" forward and backward, in contrast of linear models that the information travels only forward, but I got a similar performance and I think this has to do with the small amount of data we have and also maybe of the vectorization method. In Project 2 I experimented with GloVe pretrained vectors and the performance was significantly worse because the dimensions of the vectors were a lot smaller that the one ones created by TfIdf and maybe these large dimensions helped the linear layers of the previous model to "work" even better that the LSTM layers(as I mentioned before when the dimensions was increased the model performed better).

References

https://www.analyticsvidhya.com/blog/2021/09/sentiment-analysis-with-lstm-and-torchtext-with-code-and-explanation/

https://www.cs.toronto.edu/ lczhang/360/lec/w06/rnn.html?fbclid=IwAR1i-vZJJvZa-GpSfD-M-ShRm6OIDTvYx-hp5X2ueReoirSxyPeOYQaTWsE

https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem: :text=Gradient%20clipping%20ensures%20the%20gradient,irregular%2C%20most%20likely%20a%20cliff.

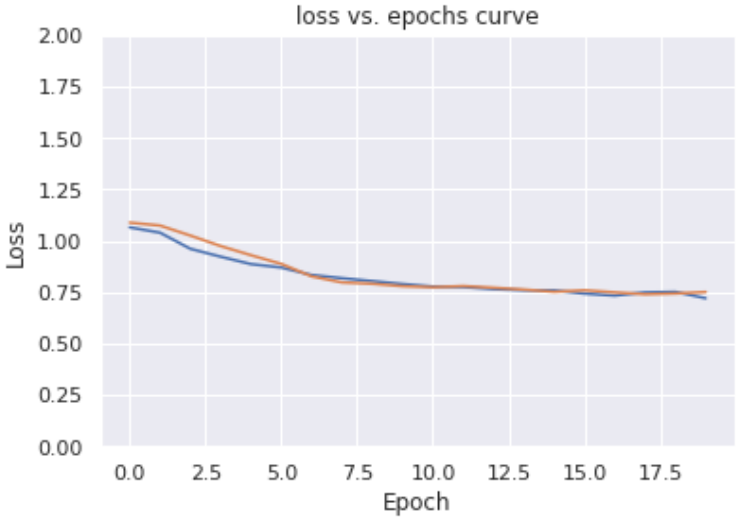
https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48

Exercise 2

If you want to run the model without attention you have to run the first cell after "Actual model definition class" otherwise if you want to run the model with attention you have to run the second cell. The rest of the program is the same.

I tried adding an attention layer expecting the model to perform better because the use of the self-attention model gives a context about the location to which the model should pay more attention in the sentence and so the model activate its neurons more to the areas of the sentence that statistically have shown more contextual importance, but in contrary the results I got were slightly worse.

Here is the best model from Exercise 1 with attention.



GRU

attention model

scores

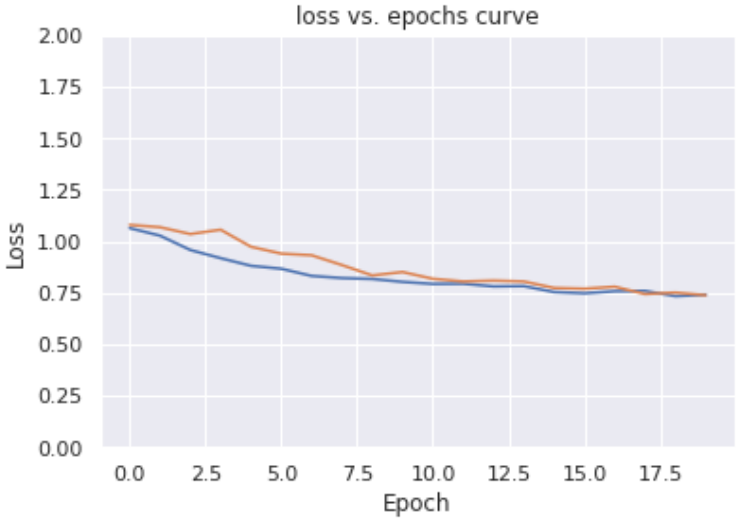
PRECISSION SCORE = 0.67337009463716

RECALL SCORE = 0.6858523725834798

F1 SCORE = 0.6676420399931928

So I thought trying new parameters may help this new model work better. After different trials (I chose not to show here because there was nothing interesting, all performed worse than the non-attention model) I observed that models with GRU layers worked better overall than LSTM models, something that did not occur on the non-attention models.

Here is an instance of an attention model with GRU layers.



GRU

attention model
PRECISSION SCORE = 0.6799182939752779
RECALL SCORE = 0.6898066783831283
F1 SCORE = 0.6766887591564674

From the tested models and parameters, I did not managed to find a model that performed better than the non-attention model. This maybe has to do with the dataset or maybe pre-trained embeddings from a language model may be better at increasing the accuracy of the model. Maybe there are some other combinations of hyperparameters that makes the model wok better but due to time limitations I could not work more on this.

References

https://towardsdatascience.com/create-your-own-custom-attention-layer-understand-all-flavours-2201b5e8be9e
https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/
https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b