UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Lampropoulos Konstantinos*
sdi: sdi1800092

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

In this task,we were asked to train a FeedForward Neural Network in order to classify the sentiment of some tweets in 3 categories: Negative,Neutral,Positive. We will start to tackle this by pre-processing our data,just like in the previous assignment,create a Word2Vec model,create a vector representation for each tweet and finally feed our Network and tune it.

2. Data processing and analysis

2.1. Pre-processing

In this section we will cover the pre-processing techniques involved.To begin with,we drop the unnecessary columns(NewID , Party) since we will not be using them in the classification process.After that, we convert all the text column in lowercase letters,remove the accents ,as well as, the stopwords and replace them with a single whitespace character.Following that,we replace multiple whitespace characters with a single white space,remove any links and non-greek words as well as retweets (and replace them with single whitespace characters).Finally,we lemmatize the text column to replace words with their dictionary values.The purpose of this pre-processing action is to breakdown the text and remove any unnecessary words and also provide the classifier with the dictionary value of the words so they are more easily recognized.

2.2. Analysis

2.2.1. Word Cloud.

In a word cloud,the most common words have greater size compared to the rest.Below I showcase an image with the results of the wordcloud implementation:

As we can see ,words like: Μητσοτάκη,Τσίπρα,νδ,συριζα, are really common in the database.We also have some words like: ο,σε,εχω ,which shouldn't be kept after removing stopwords.These should be removed for better results,but were not in this project.

2.2.2. Token Frequency.

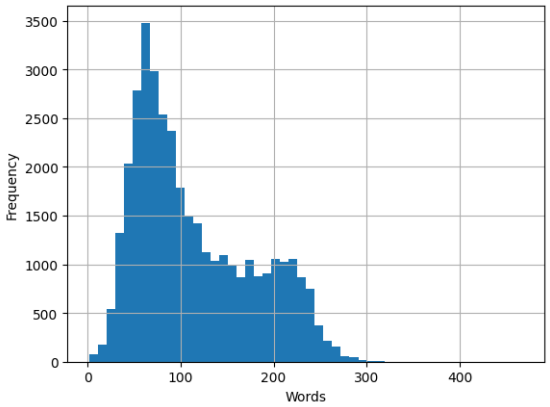
Token frequency is used to find the most common words in the dataframe.In this dataframe our 10 most common words are:

word	count
ο	18181
τσιπρα	9889
μου	8189
νδ	8168
μητσοτακη	6883
σε	4528
εκλογος	4298
εχω	4207
συριζα	2943
συριζας	2254

We can see that some stopwords appear as most common words.These should be removed manually as they may impact the training of the model,but were not for this project.As we can see some of the most common words are τσιπρα,νδ,μητσοτακη which means that most of the common words are correctly identified and will likely help us better identify the sentiment

2.2.3. Text Length Distribution.

Text length distribution is a method which finds the most common text lengths in the dataframe.



As we can see,the most common text lengths are between 40 to 80 words,which is the size of a common text or tweet.

2.3. Data partitioning for train, test and validation

The dataset remained the same as the one given in the assigment.

2.4. Vectorization

For the vectorization,the Word2Vec model was used with these parameters: vector size 300,window 5,minimum count 2 and negative 10. Vector size means that each word will be represented by a vector of 300 values,window 5 means that the model will check up to 5 words before and up to 5 words after,to learn the context of the current word.Minimum count 2 means that words appearing less than 2 times will be excluded,and negative is used to speed up training as it adjusts only a number of weights for each sample.Below is an example of the most similar words of a certain word:

word	Similarity Rate
Μητσοτακης	0.5311290621757507
συρριχνωσει	0.3812375068664551
φορων	0.3534460961818695
μεγαλοεπιχειρηματιος	0.3458263874053955
προχωρησει	0.3422377109527588
μητσοτακη	0.339499831199646
λουτσεσκος	0.33809322118759155
πειστικος	0.321236252784729
περικοπω	0.3175898790359497
διαλογο	0.31199774146080017

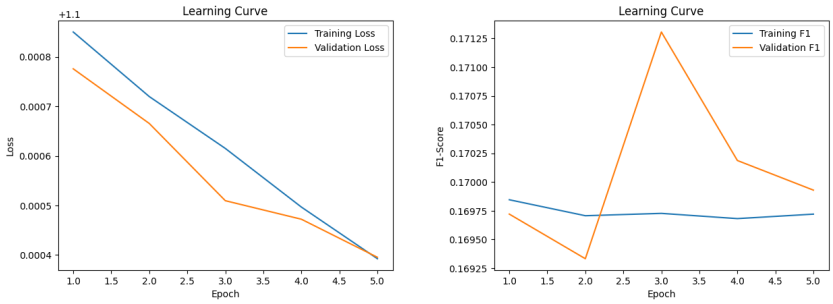
As we can see the most similar words to μητσοτακη ,are words that have some context to the nature of the current word(μεγαλοεπιχειρηματιος,πειστικος), or some similarity(Μητσοτακης,μητσοτακη).

It is somewhat interesting that the word Μητσοτακης,has a capital letter,since all words were converted to lowercase.This may be due to different encoding to the first letter of the word(since accents were removed correctly).

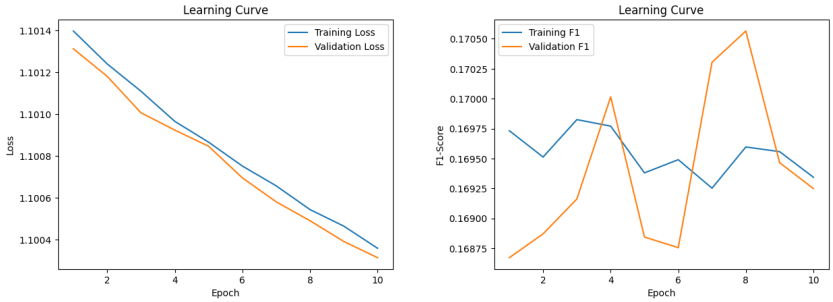
3. Algorithms and Experiments

3.1. Experiments

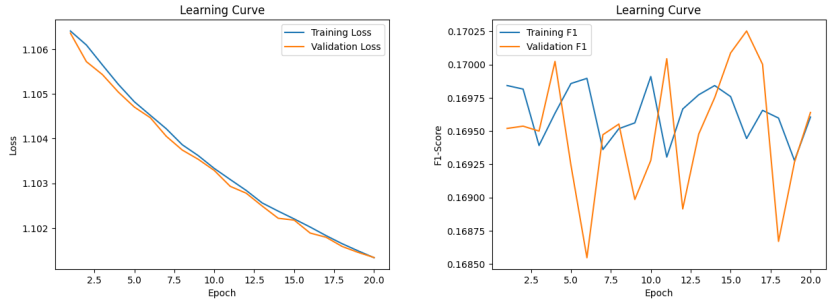
A brute force run was not done,because we already knew from the last assignment that pre-processing had to be done(and was also already implemented to some degree). At first,an experiment was performed with some default values(which were inspired by the lab showcasing Neural Networks) and from there what ever improvement could be done by assessing the learning curves,was done.We will start by analyzing the impact the number of epochs have in the training of a model:



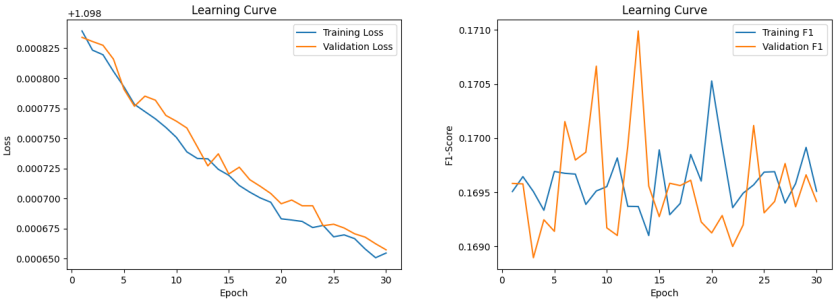
As we can see,with a starting batch size of 64 and 5 epochs,our model cannot correctly capture the relation between the input data and the output,so we increase the number of epochs.



At 10 epochs,still the model hasn't have had enough time to identify any relation.



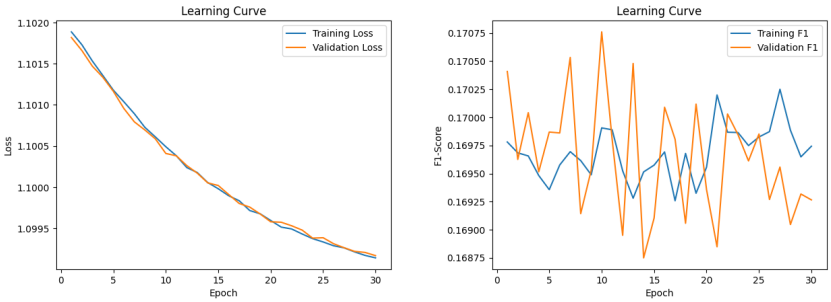
By further increasing the number of epochs,our model is starting to show signs of capturing the relation to a degree,but needs more epochs for that to happen correctly.



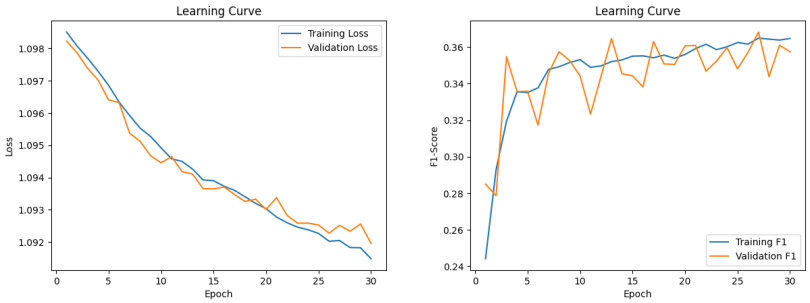
30 epochs seem reasonable for any model with a learning rate of 1e-4.More epochs may be useful ,but will increase training time.Nevertheless we will increase the number of epochs to 60 later.

The next thing we want to configure,is what optimizer we will use.Optimizers use a standard learning rate to try and improve the model's weights with each batch.

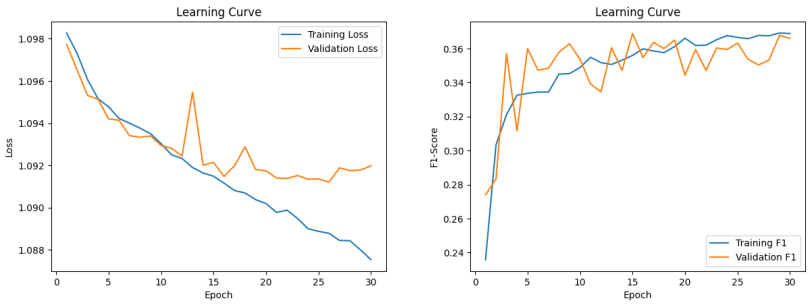
The first optimizer we will test is Stohastic Gradient Descent (SGD):



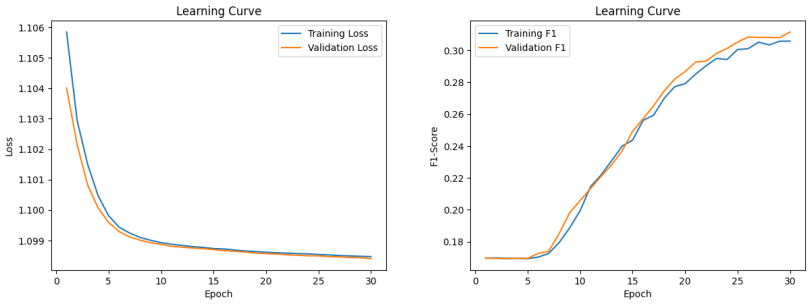
This optimizer didn't perform really well as it returned an f-score of 0.17 at 1.26 minutes.As far as loss goes it seems to generalize well,but f1 score shows inconsistency. The next optimizer is Adamax



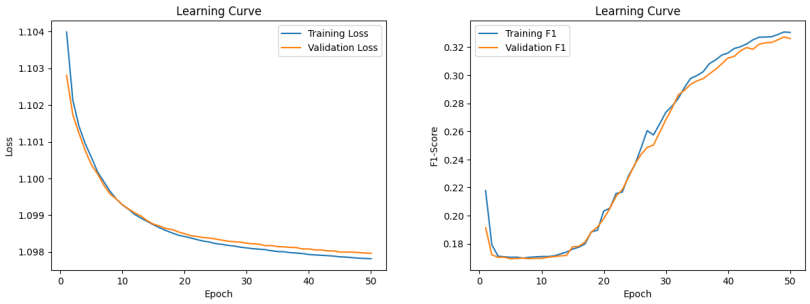
Adamax performed significantly better than SGD reaching an f-score of 0.368 at 1.42 minutes.This makes Adamax a good candidate as our preferred optimizer.
Next is AdamW:



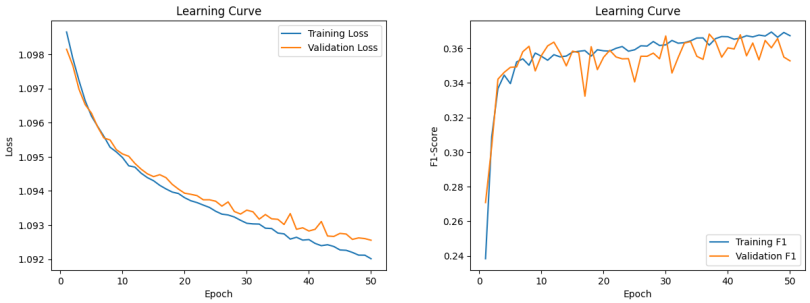
Even though AdamW's performace is really good(f-score 0.368),there are clear signs of overfitting just by looking at the loss learning curve,thus it will not be selected as a preferred optimizer.
The last 2 optimizers where particularly interesting.First is Adagrad:



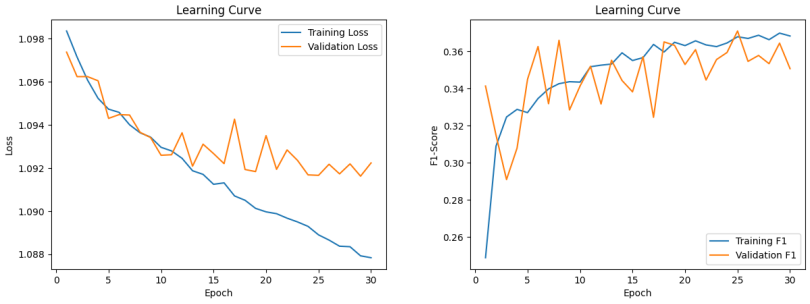
We can see that Adagrad generalizes exceptionally compared to the previous algorithms and even though its f-score was lower(0.311),by inspecting the learning curves we can deduct that more epochs are needed and/or greater learning rate.By adjusting just the number of epochs to 50 we get:



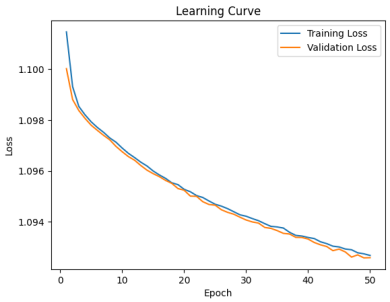
As we can see the algorithm still shows improvement even after 50 epochs,but further increasing the number of epochs will result in much greater training times,so we increase the learning rate to 1e-3:

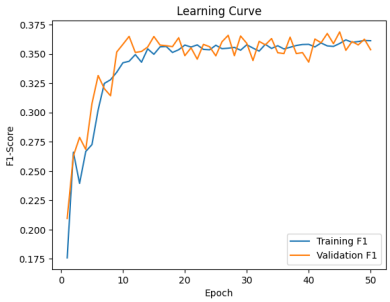


As we can see by increasing the learning rate,the algorithm still generalizes really good,with only slight signs of overfitting(as shown in the loss learning curve) and now achieves a f-score of 0.368.
The final optimizer tested is Adam which showcased similar behaviour to Adagrad.



We can see that with a default learning rate of 1e-4 the optimizer overfits,yet achieves the best f-score yet (0.371).In order to battle overfitting we again increase the number of epochs,but this time decrease the learning rate,and we achieve:



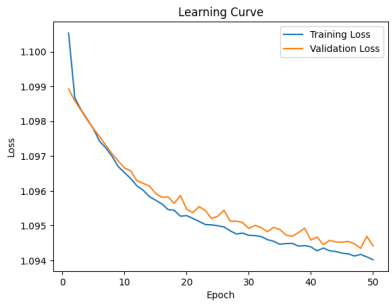


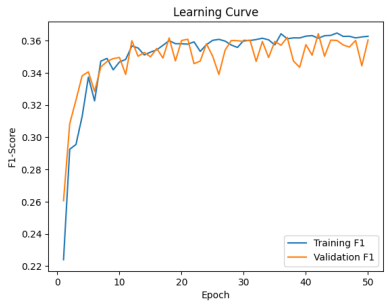
As we can see by both the loss and f-score learning curves,the current model generalizes really well while still achieving relatively good f-scores(0.368). So we will use this optimizer throughout the rest of the tuning process.

Below lies a table will the experimenting done:

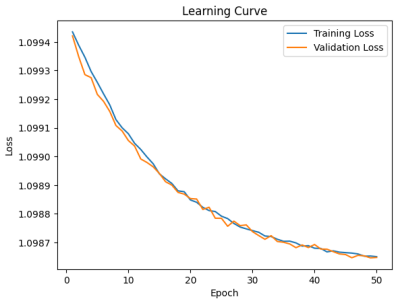
Optimizer	Accuracy	F1-Score	Time
SGD	33.3%	0.170761	1m26.6s
Adagrad	33.8%	0.311493	1m34.1s
Adagrad	34.8%	0.327325	2m35.4s 50 Epochs
Adagrad	36.8%	0.368264	2m36.8s 50 Epochs Learning Rate = 1e-3
Adam	37.1%	0.371074	1m26.7s
Adam	37.3%	0.368761	2m36.8s 50 Epochs Learning Rate = 1e-5
AdamW	37.0%	0.368899	1m25.0s
Adamax	37.0%	0.368293	1m42.4s

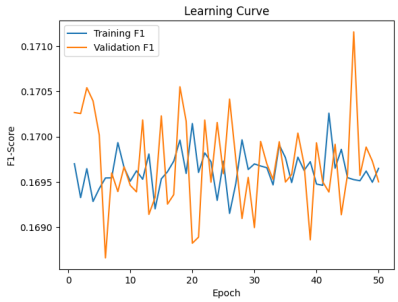
The next parameter,I experimented on ,were the Activators for the Linear Layers of the Neural Network.The default activator with which all the above optimizers were tested was ReLu.As we saw above,it achieved really good results.The next activator was Elu:



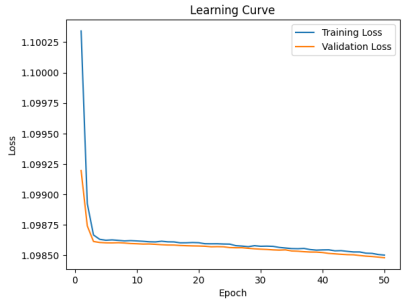


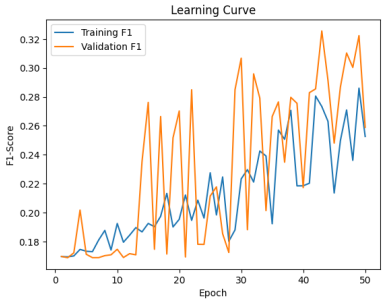
As we can see,Elu doesn't generalize as well as ReLu and returns a lower f-score of 0.364 (compared to 0.368 of ReLu). Next is the HardShrink activator:



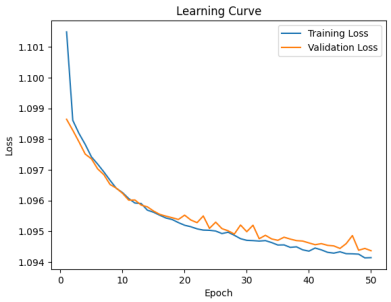


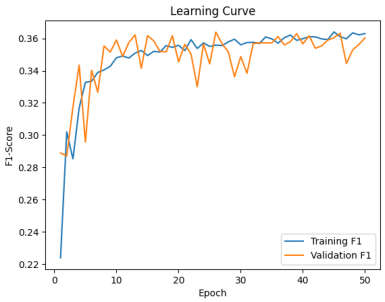
By just looking at the loss learning curve,the activator seems to help generalize,but f-score is pretty low and inconsistent. Next we check HardSigmoid:





HardSigmoid shows the same inconsistency on the f-score as hardshrink.We can also see that validation f-score is almost all the times higher than the training score,which either means we selected the wrong activator,or our validation set is easier for the model to predict.Since we had really good generalization with ReLu, we can deduct that ,simply this is not the appropriate activator for our implementation.
Finally we have HardTanh:

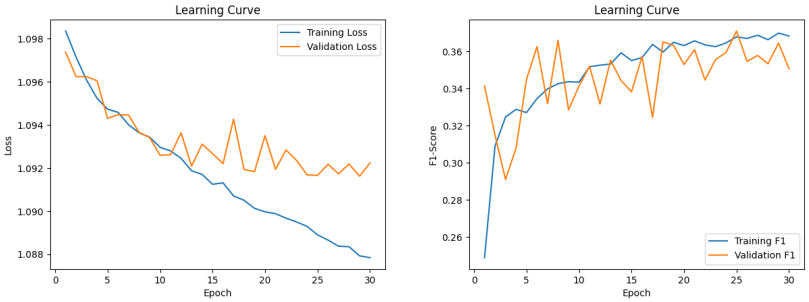




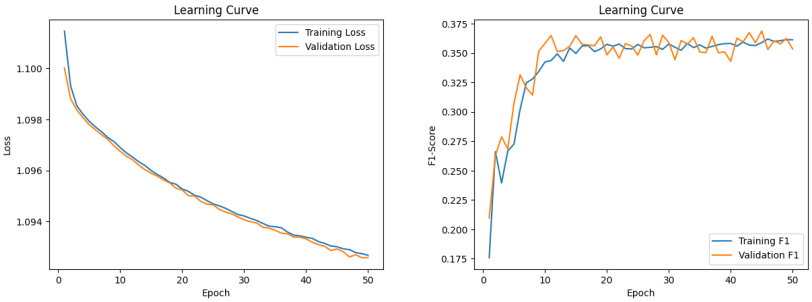
HardTanh does a good job on both the loss and f-score but returns a lower f-score compared to our current best (ReLu),and since we optimize by f-score, we will choose ReLu.
Below is the table of the experiments performed :

Activator	Accuracy	F1-Score
ReLu	37.3%	0.368761
ELU	36.6%	0.364278
HardShrink	33.3%	0.171157
HardSigmoid	34.7%	0.325575
HardTanh	36.6%	0.363918

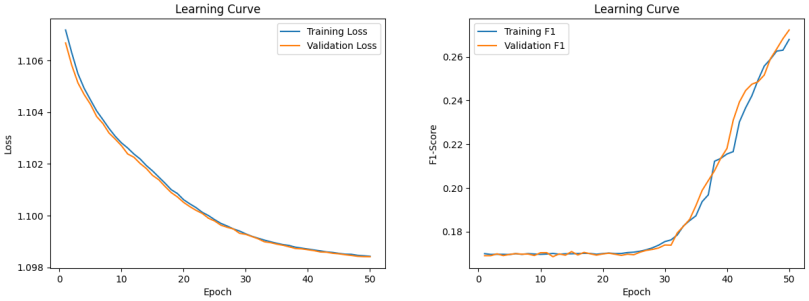
Next we will tune the learning rate for out optimizer.Learning rate is a parameter which determines how quickly or slowly the NNs (Neural Network) weights are updated.We slightly experimented with learning rates when choosing the appropriate optimizer,but now we will determine the best learning rate for our current optimizer.The default learning rate for Adam was 1e-5.
We firstly ,though,experimented with learning rate of 1e-4 which returned these results:



As we can see learning rate of 1e-4 is too much for our optimizer,since it starts overfitting really fast.
Next is our default learning rate of 1e-5:



We can see a significant improvement compared to the previous learning rate,since now our model is able to generalize really well to the validation set.
Last we have learning rate of 1e-6:

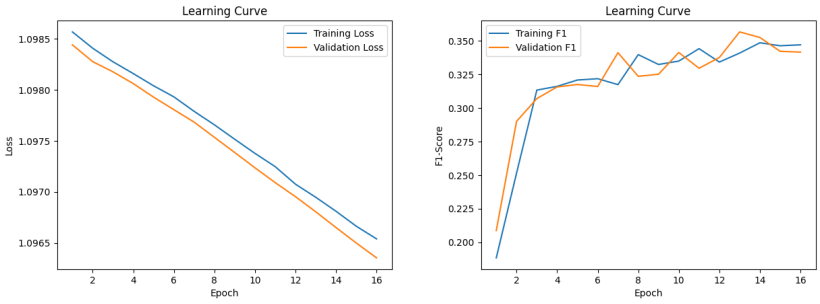


By just looking at the learning curves,this is the best learning rate.But even after 50 epochs,our model scores a f-score of only 0.272,which may after adding more epochs,increase.That would mean ofcourse that we shoduld also increase the number of epochs are model is trained,which would increase training time.
Below is the table of experiments:

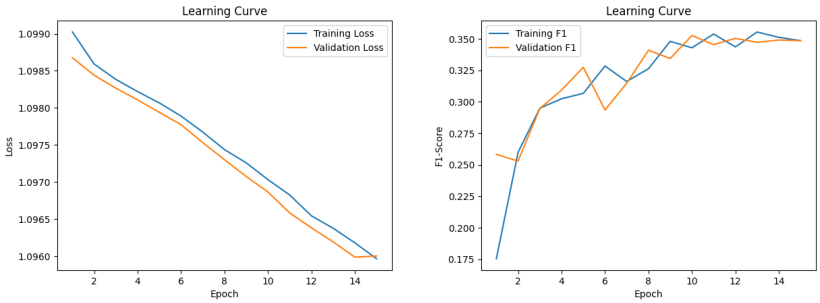
Learning Rate	Accuracy	F1-Score
1e-4	37.1%	0.371074
1e-5	37.3%	0.368761
1e-6	35.2%	0.272236

As we can see the best f-score is scored on learning-rate 1e-4,but since our model shows some degree of overfitting we must prefer the more stable learning rate of 1e-5.

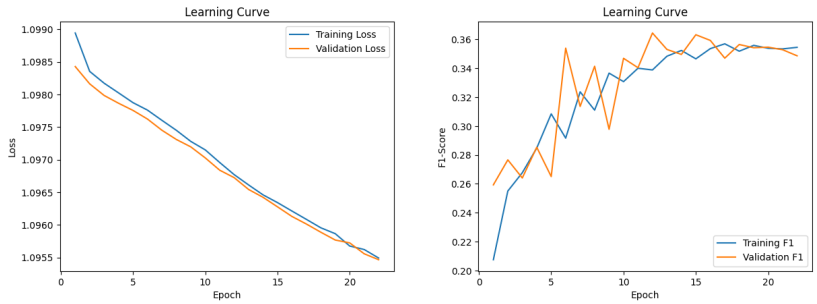
The next parameter I experimented on,is Early Stopping (which from now on will be mentioned as patience for the rest of the report). Patience is used to stop the training process earlier,if after a number of epochs a better model than the current one is not found. We started with relatively low patience (3 epochs) and these are its results:



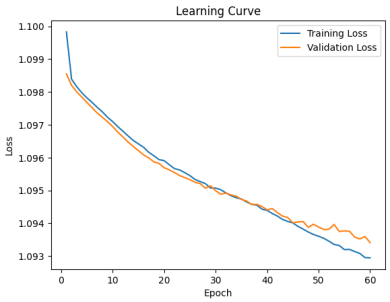
Even though the model seems to generalize and converge really fast (45.8 seconds) it returns an f-score of 0.356.So we increase patience to 5 next:

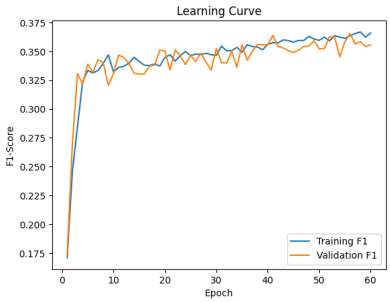


The interesting about this,is that the model converged quicker with 5 epochs patience (44.2 seconds),than with 3.This is occured by the factor that torch dataset have a random seed factor,and will not introduce the dataset the same way each run.Still we can achieve greater f-scores.We increment patience to 10:



Finally we try to have 20 epochs as patience:

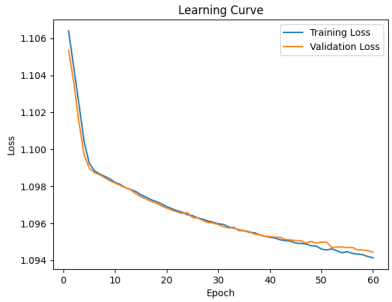




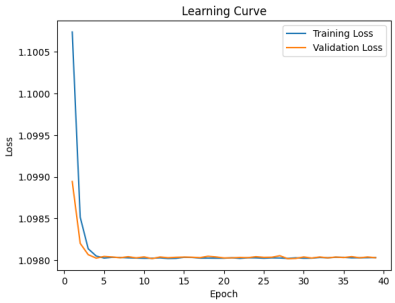
Both patience of 20 and no patience returned the same results,as when patience of 20 epoch was used,the algorithm converged at 60 epochs,returning the best results of f-score 0.365.Even though the trade-off of almost triple the time of training for a mere 0.001 improvement is not worth it ,in most cases, here 3 minutes of training time is acceptable.

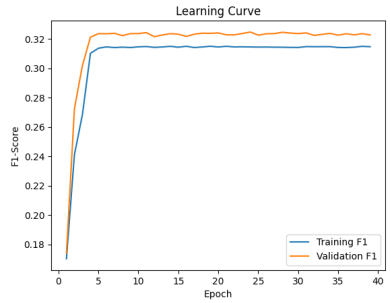
Patience	Accuracy	F1-Score	Time
3	36.0%	0.356683	45.8s
5	35.8%	0.352698	44.2s
10	36.6%	0.364468	1m4.9s
None-20	36.6%	0.365343	3m0.5s

Next we will implement the use of schedulers.Schedulers optimize the learning rate after each epoch.based on the model's current performance. We start with the LinearLR scheduler:



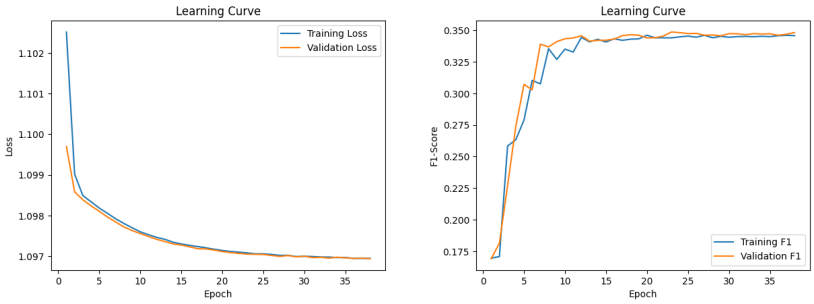
Even by using a scheduler f-score seems to be lower than compared to our previous testing.
Next we used PolynomialLR with a polynomial of 3rd degree as the scheduler:



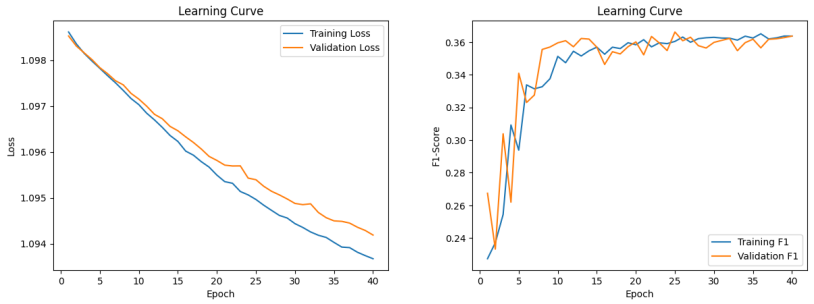


The resulting best model has a mere f-score of 0.324 which is even lower than the linearLR,also validation f-score is higher than training which shows that this is not the

appropriate scheduler.
Next we used ExponentialLR:



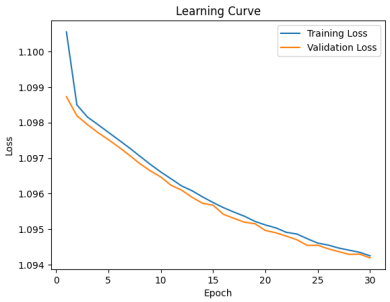
Again our model performs really well on terms of generalization and prevents overfit,but returns an f-score of 0.348.
Finally we use CosineAnnealingLR:

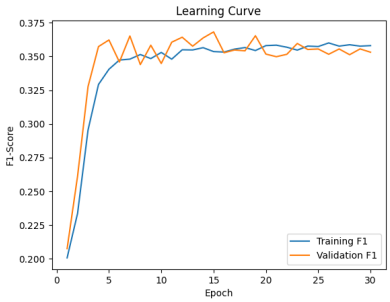


Even though we see some divergence at the end of the training phase,I would argue that a loss difference of 0.001 is acceptable between training and validation, and I will prefer this over the LinearLR since it returns a higher f-score.
Below is the table of experimenting:

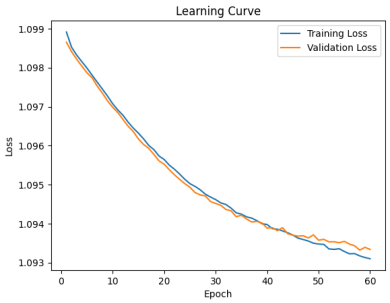
Scheduler	Accuracy	F1-Score
LinearLR	36.6%	0.361416
PolynomialLR	35.2%	0.324691
ExponentialLR	36.4%	0.348560
CosineAnnealingLR	36.6%	0.366208

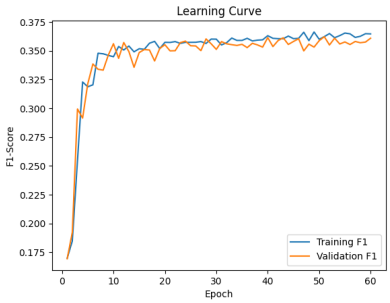
Next we tried to minimize the slight overfit caused by our scheduler using Dropout.Dropout randomly removes some neurons from the network to prevent them from specializing on the specific data,and cause overfitting.We firstly started with dropout rate of 0 which is represented by the learning curve of CosineAnnealingLR above.
Next we use slight Dropout of 0.2:



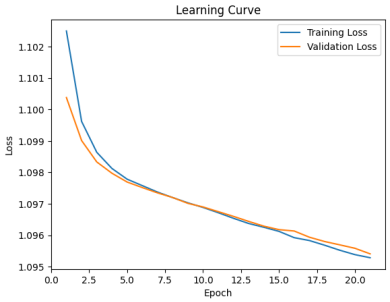


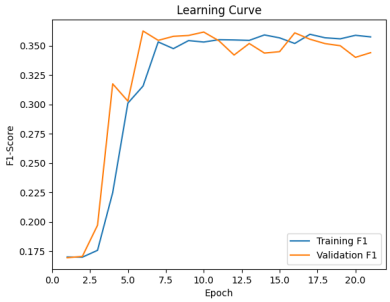
We can see that using Dropout of 0.2 our model generalizes better and returns a better f-score (0.368).
Next is Dropout of 0.3:





We can see the further incereasing the dropout decreases the model's performance. Nevertheless,for experimenting purposes,I experimented also with Dropout of 0.5:



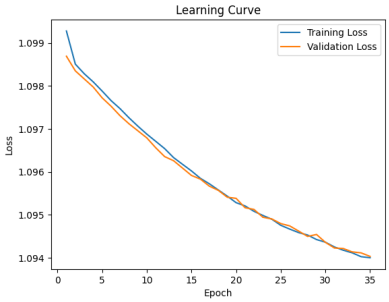


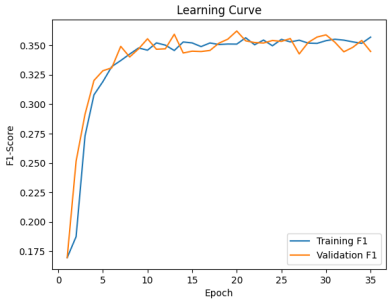
Dropout of 0.5 returns a similar performance to the model of 0.3 ,but still the f-score is substantially lower than 0.2,hence 0.2 dropout will be preferred.
Below is the table of experiments:

Dropout Rate	Accuracy	F1-Score
0	36.6%	0.366208
0.2	36.8%	0.368184
0.3	36.8%	0.362087
0.5	36.4%	0.362567

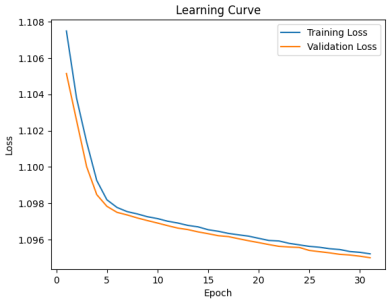
The last parameter tuned was Batch size.Batch size determined at each batch,how many of the training data are passed in the model.Higher batch size decreases training time,but also the number of times the weights are updated.Lower batch size means

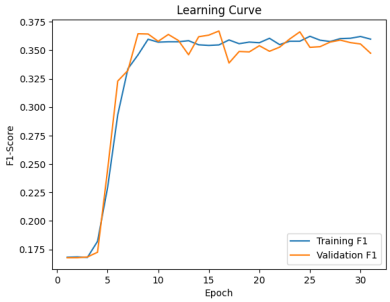
higher training times but more frequent updates to the weights of the model.
I firstly experimented with batch size of 64 (which was used on all the experiments above),but since i wanted also to track the time for each batch size,i re-run the experiment:



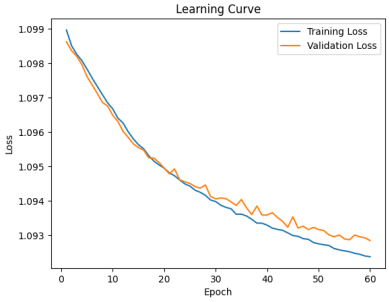


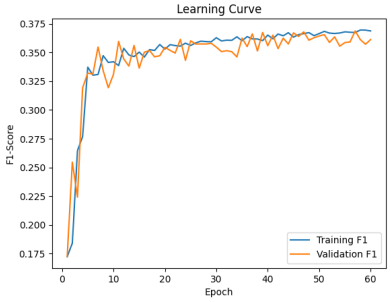
Training with 64 batch size took 1 minute and 35.5 seconds to complete returning as it would seem a low f-score.
Next I tried a higher batch size of 128:



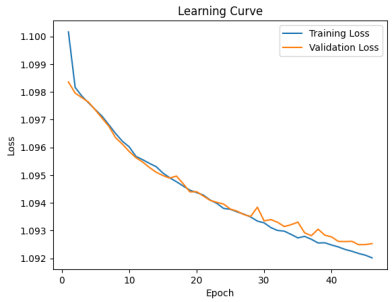


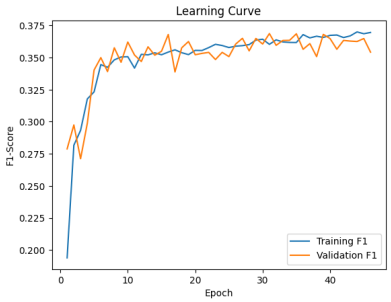
Batch size of 128 returned pretty promising results as in only 47.2 seconds ,it yielded 0.366 f-score without concerning signs of overfitting.
Next I started decreasing the batch size to see what lower batch sizes do.So now batch size is 32:



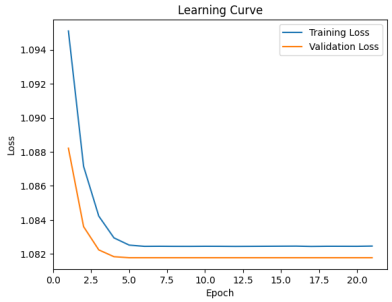


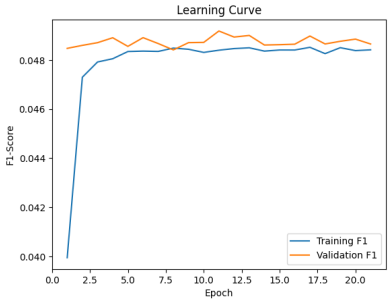
Now we can start to see some slight signs of overfitting (by looking at the loss learning curve),and the training time is increased to 5 minutes and 17.9 seconds.
Next we use 16 batch size:





Batch size of 16 shows less overfitting while taking almost the same time as batch size 32(this is caused by the patience implemented above,and as I stated previously,torch dataset randomly seed the data,so when data is parsed differently,even though the whole dataset is parsed each epoch,different changes will occur at different times and we can end up with a different model).I was not satisfied with the results of 16 batch size so i re-run it and it returned f-score 0.37 at 7 minutes and 33.0 seconds.
Lastly batch size was decreased to 8:



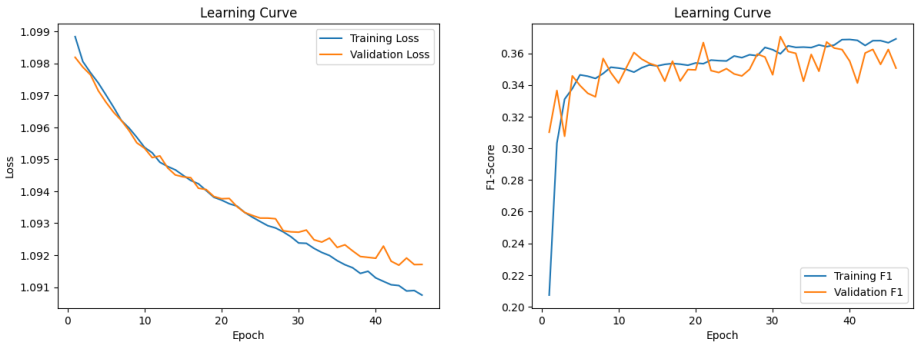


This returned the best f-score of 0.371 but validation f-score was higher than training f-score.I experimented also with higher dropout rate which returned an f-score of 0.369.Nevertheless,batch size of 16 was chosen over all others,since it returned an acceptable f-score with not much overfitting and at a decent time(7 minutes 33 seconds,whereas batch size of 8 took 16 minutes and 5.1 seconds).
Below is the table of experiments:

Batch Size	Accuracy	F1-Score	Time
8	36.8%	0.369066	16m5.1s DROPOUT=0.5
8	36.9%	0.371997	16m5.1s
16	37.2%	0.370520	7m33.0s
16	37.2%	0.368731	5m51.3s
32	37.1%	0.368721	5m17.9s
64	36.6%	0.362117	1m35.5s
128	36.9%	0.366861	47.2s

3.1.1. Learning Curve.

The learning curves for the best model configured are:



We can see that the model shows slight overfitting,which in time (more epochs) would increase but our training is cut short because of patience so no major overfitting occurs.

3.1.2. Confusion matrix.

The Confusion Matrix for this test run is:

3	3	3
0	1	1
1	1	3

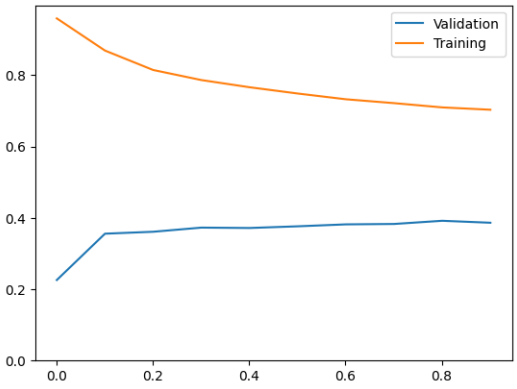
4. Results and Overall Analysis

4.1. Results Analysis

Results are expected,since dataset as mentioned,has some bad labeling(tweets with positive sentiment categorized as negative or neutral etc on the training set), and since we construct a simple FeedForward Neural Network,it is expected that the processing of Natural Language would not be optimal.In order to process the results better we would need a more complex Neural Network.

4.2. Comparison with the first project

Compared to the fist project,even though f-score remains pretty much the same,this time we managed to prevent overfitting to a much greater degree.Below is the learning curve when using the logistic regression classifier:



This learning curve shows the f-score for training and validation.There is a huge difference between both f-scores which would in time decrease and no overfitting would occur,but that would need a much much bigger dataset.So even though perfomarce remained relatively the same,we managed to make our model generalize better to new unseen data.

5. Bibliography

References

[1] Pytorch And Neural Network Labs found on e-class.