UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *<Kleopatra Karapanagiotou>*
sdi: <It12200010>

Course: *Artificial Intelligence II (M931)*
Semester: *Winter Semester 2023*

Contents

1. Abstract 2

2. Data processing and analysis 2
2.1. Pre-processing 2
2.2. Analysis 2
2.3. Data partitioning for train, test and validation 3

3. Algorithms and Experiments 4
3.1. Network Architecture 4
3.2. Experiments 5
3.2.1. Table of trials. 8
3.3. Hyper-parameter tuning 9
3.4. Optimization techniques 9
3.5. Evaluation 10
3.5.1. ROC curve and AUC. 10
3.5.2. Learning Curves. 12
3.5.3. Classification Reports. 14

4. Results and Overall Analysis 16
4.1. Results Analysis 16
4.2. Comparison with the first project 18

5. BibliographyReferences 19

1. Abstract

This is a multiclass classification task on Greek tweets, where given a tweet the classifier should classify it into one of our 3 classes: POSITIVE, NEGATIVE, NEUTRAL.

In this experiment series we will use Feedforward Network architectures along with Word2Vec embeddings to develop our tweet sentiment classifier. Several activation functions are explored for overfitting avoidance along with optimization techniques for hyperparameter tuning. Data preprocessing, word2vec model creation, network architectures, hyperparameter tuning approaches, and evaluation metrics are presented in the following sections of the report.

2. Data processing and analysis

2.1. Pre-processing

Preprocessing steps applied on the Text column of both the train and validation sets:

- Removal of NaN values
- Pattern removal: RT, @usernames, html, http, https, url links
- Lowercasing
- Text normalization and replacement of multiple characters with one
- Removal of punctuation, Latin characters, numbers, and words with character length of 1.: this was done in order to remove single characters like 'κ' for 'και', which is a common abbreviation in Greek tweets.
- Removal of emojis
- Uppercasing
- Removal of stopwords excluding specific ones: ΟΧΙ, ΜΗΝ, ΔΕΝ, ΠΟΛΥ, ΚΑΘΟΛΟΥ, ΟΤΙ, since these stopwords when removed they can shift the meaning of the sentence: (e.g ΚΑΚΟΣ vs ΚΑΘΟΛΟΥ ΚΑΚΟΣ), or intensify it like: «ΧΕΙΡΟΤΕΡΟ vs. ΟΤΙ ΧΕΙΡΟΤΕΡΟ.»
- Text tokenization and Stemming with the Greek Stemmer[1], a stemmer generated based on the Paper of Georgios Ntais[2].
- Removal of NaN rows and updating of the whole dataframe , in case any row left empty after the preprocessing applied.

  For practical reasons removal of NaN values before and after applying preprocessing was skipped on the test set.

2.2. Analysis

Since the dataset Analysis was done exhaustively on the first assignment in order to end up on the above preprocessing steps, I skip the Analysis part. Since the only Preprocessing step, I added on this experiment series is the Stemming of tokenized words, in order to create word2vec embeddings on stemmed words of the train and the validation set.

[1] https://gist.github.com/Patelis-GM/e1f8cf553f27ff40ed49db8c310611b3

[2] https://people.dsv.su.se/~hercules/papers/Ntais_greek_stemmer_thesis_final.pdf

2.3. Data partitioning for train, test, and validation

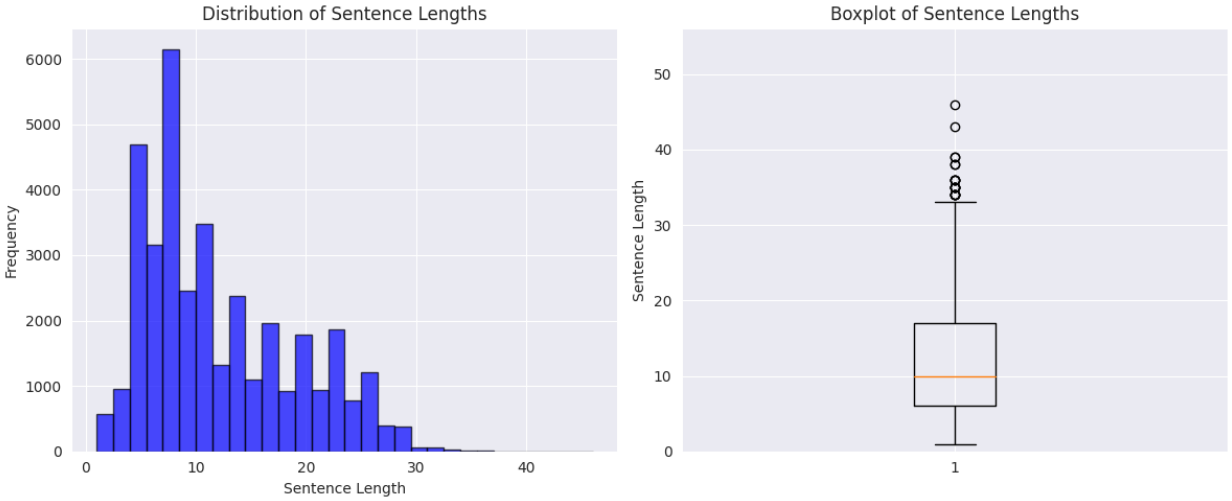
In our experiment setting our data are already partitioned on 3 .csv files, train, validation, and test. No further partitioning was performed. Training data were used for the training of the Network, validation data were used as the evaluation part of the network and the plotting of evaluation metric results and learning curves. And finally test data were used to predict the polarity of each tweet by creating a new column named: Predicted and adding the predictions of our classifier.

2.4. Vectorization

2.4.1 Word2Vec

For vectorization of the data a word2vec model was built and trained on the vocabulary of the train and validation sets, after applying the same preprocessing steps as mentioned above. The vocabulary consists of Greek stemmed words, so that the saved model can also be applied on the test set. The word2vec model was trained with the following parameters:

- Skipgram model (given a target word the model tries to predict the surrounding context words in a specific window)
- Min_count=1 (words must appear at least once to be taken into account for the word2vec model's vocab. This small min_count was chosen due to the small vocabulary of the words and since during preprocessing highly frequent words (stop words) have been removed.)
- Window=4 (considering that the mean tweet length after preprocessing is 11.8 ~12 tokens, a window size of 4 was chosen.)



- Vector_size=200 /300. (All experiments apart from Exp 2 were conducted with 200d tweet vectors)
- Alpha=0.01 (learning rate)
- Negative sampling and number of negative samples=10 (Randomly select 10 negative words. Considering the rule of thumb that 5-20 negative words are better for smaller datasets, I chose the 10 negative samples)
- Subsampling=0.001 (default value)
- Epochs=20
- Shrink_windows=True (to weigh the context words differently based on their position in the context window)

To get an idea of the resulting embedding representations for each word, here are the 10 most similar words for each political party:

```
[ ] model_w2v.wv.most_similar(positive="ΝΔ")
```

```
[('ΞΕΦΤΙΛ', 0.6204831004142761),
 ('ΑΠΑΤΕΩΝ', 0.601689875125885),
 ('ΑΔΩΝΙΣ', 0.5563056468963623),
 ('ΚΛΕΦΤ', 0.5394425392150879),
 ('ΝΔΟΥΛ', 0.5316289067268372),
 ('ΥΠΟΔΙΚ', 0.5283469557762146),
 ('ΒΟΡΙΔ', 0.5209547877311707),
 ('ΝΟΥΔΟΥΛ', 0.5142523646354675),
 ('ΤΑΣ', 0.5125272870063782),
 ('ΝΕΑΔΗΜΟΚΡΑΤ', 0.5091320872306824)]
```

```
[ ] model_w2v.wv.most_similar(positive="ΣΥΡΙΖ")
```

```
[('ΤΕΛ', 0.5790039300918579),
 ('ΤΣΙΡΚ', 0.5011270046234131),
 ('ΔΡΑΜ', 0.4929939806461334),
 ('ΕΦΟΔ', 0.4913572669029236),
 ('ΑΝΟΜ', 0.4853563904762268),
 ('ΞΕΚΟΥΜΠΙΣΤ', 0.48422759771347046),
 ('ΣΥΜΜΑΧ', 0.4817162156105041 5),
 ('ΣΙΓΟΥΡΑΚ', 0.48059192299842834),
 ('ΜΠΟΥΛ', 0.4781439900398254 4),
 ('ΣΥΡΙΖΑΝΕΛ', 0.477820485830307)]
```

```
model_w2v.wv.most_similar(positive="ΚΚΕ")
```

```
[('ΚΟΥΤΣΟΥΜΠ', 0.6262622475624084),
 ('ΜΛ', 0.6091932654380798),
 ('ΔΙΓΕΝ', 0.5854487419128418),
 ('ΣΕΜΙΝ', 0.5564846396446228),
 ('ΠΑΠΑΡΗΓ', 0.5406489372253418),
 ('ΛΑΕ', 0.5374833345413208),
 ('ΚΙΜΟΥΛ', 0.5365214943885803),
 ('ΔΥΝΑΜΩΣ', 0.5289780497550964),
 ('ΑΝΤΙΛΑΪΚ', 0.5264309048652649),
 ('ΑΝΤΑΡΣΥ', 0.5256466269493103)]
```

```
model_w2v.wv.most_similar(positive="ΠΑΣΟΚ")
```

```
[('ΠΑΛΙ', 0.5933243036270142),
 ('ΟΡΘΟΔΟΞ', 0.49397915601730347),
 ('ΠΑΣΟΚΑΡ', 0.49046579003334045),
 ('ΑΦΜ', 0.48736920952796936),
 ('ΞΕΧΑΣ', 0.4862816333770752),
 ('ΠΟΤΑΜ', 0.4756200313568115),
 ('ΓΑΠ', 0.46699559688568115),
 ('ΠΑΛ', 0.45340874791145325),
 ('ΕΓΚΛΗΜΑΤΙΚ', 0.4531550109386444),
 ('ΑΥΤΟΝΟΜ', 0.4515827000141144)]
```

2.4.2. Average Word2Vec

In order to generate a single vector representation for each tweet Average word2vec was implemented. For words not present in the word2vec's model vocabulary a vector of 0s and vector size 200, is generated. One very important drawback of Average word2vec, which is worth noting, is that it does not consider the word order, meaning that two tweets like the following:

- our president is a good leader he will not fail
- our president is not a good leader he will fail

will have the same average word embedding, even though it is obvious for us that those two tweets have the opposite meaning. This is why newer sentence representations like sentence-BERT gained more attention and interest.[3]

But for our task since we have to work with Word2Vec embeddings it is important to keep that issue in mind when explaining the experiment results.

3. Algorithms and Experiments

3.1. Network Architecture

A feedforward neural network is one of the simplest types of artificial neural networks devised. In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network.

[3] https://stats.stackexchange.com/questions/318882/what-does-average-of-word2vec-vector-mean

3.2. Experiments

Experiments 1,3-8: I trained a (200d) word2vec model on the train and development set and then loaded it in order to use it for use in the network development, the inference and predictions sections.

Experiment 2: A (300d) word2vec model was generated for each dataset separately, when running the scripts. With this approach I tried to avoid the possibility of no word of the word2vec vocabulary appearing on the test set.

- Experiment_1:

Network Architecture	FFNN (ReLU) n_layers: 1 n_units_l0: 95
Weight initialisation	-
Regularization techniques	Dropout: 0.2 Batch normalization
Optimizer	Adam
Learning rate	0.0001
lr_scheduler	-
n_epochs	40
Batch_size	10
output activation function	Sigmoid

- Experiment_2:

Network Architecture	FFNN (ReLU) n_layers: 1 n_units_l0: 13
Weight initialisation	-
Regularization techniques	dropout_l0: 0.27 Batch normalization
Optimizer	RMSprop
Learning rate	lr: 0.005
lr_scheduler	-
n_epochs	40
Batch_size	10
output activation function	Sigmoid

- Experiment_3:

To ensure stability of gradients during training, so that the inputs and outputs preserve the same variance across the entire network, I attempted to apply self-normalizing neural networks with SELU. In a self-normalizing neural net the output of each layer will tend to preserve a mean of 0 and standard deviation of 1 during training, which solves the vanishing/exploding gradients problem.
Followed prerequisites for implementation:

- Sequential model of fully connected layers
- Weights initialization with the LeCun normal initialization technique[4]
- Scaled ELU activation function.
- Inputs are standardized (e.g. with StandardScaler)

An implementation of the paper was attempted based on code structure found in github[5]. The followed implementation makes a comparison between an MLP with ReLU and batch normalization and a self-Normalizing Network with SELU. I kept only the classes for SELU, the Self-normalizing Network and the LeCun weight initialization function.

Network Architecture	(FFNN) Self-NormalizingNN (SELU) • n_layers: 1 • n_units_l0: 95
Weight initialisation	LeCun
Regularization techniques	Dropout (0.2)
Optimizer	Adam
Learning rate	0.0001
lr_scheduler	-
n_epochs	40
Batch_size	10
output activation function	Sigmoid

- Experiment_4:

Network Architecture	FFNN (Randomized Leaky ReLU) • n_layers: 1 • n_units_l0: 95

[4] https://medium.com/@gidim/part-2-selecting-the-right-weight-initialization-for-your-deep-neural-network-cc27cf2d5e56

[5] https://github.com/MaximeVandegar/Papers-in-100-Lines-of-Code/blob/main/Self_Normalizing_Neural_Networks/selu.py

Weight initialization	Glorot
Regularization techniques	Scaling (StandardScaler) Dropout (0.2) Batch Normalisation
Optimizer	Adam
Learning rate	0.0001
lr_scheduler	-
n_epochs	40
Batch_size	10
output activation function	Sigmoid

- Experiment_5:

Same parameters as Experiment 4 without Dropout ratio, in order to cross-check the model's train and validation loss.

- Experiment_6:

Network Architecture	FFNN (RReLU) n_layers: 1 n_units_l0: 52
Weight initialisation	Glorot
Regularization techniques	Scaling (StandardScaler) Dropout (0.2) Batch Normalisation
Optimizer	RMSprop
Learning rate	lr: 0.005
lr_scheduler	Exponential (gamma=0.5)
n_epochs	50
Batch_size	128
output activation function	Softmax

- Experiment_7:

Same parameters as Experiment 4 without Dropout ratio, in order to cross-check the model's train and validation loss.

- Experiment_8(submission):

Network Architecture	FFNN (RReLU) n_layers: 1 n_units_l0: 43
Weight initialisation	He
Regularization techniques	Scaling (StandardScaler) Dropout (0.4) Batch Normalisation
Optimizer	RMSprop
Learning rate	lr: 0.004
Lr_scheduler	Exponential (gamma=0.5)
n_epochs	50

Batch_size	128
output activation function	Softmax

Experiment series	**W2V**	**Accuracy**	**Weighted avg Precision**	**Weighted avg Recall**	**Weighted avg F1**
Experiment_1	AvgWord2Vec(200)	0.39	0.39	0.39	0.39
Experiment_2	AvgWord2Vec(300)	0.33	0.11	0.33	0.17
Experiment_3	AvgWord2Vec(200)	0.37	0.38	0.37	0.37
Experiment_4	AvgWord2Vec(200)	0.39	0.39	0.39	0.39
Experiment_5	AvgWord2Vec(200)	0.39	0.39	0.39	0.39
Experiment_6	**AvgWord2Vec(200)**	**0.40**	**0.40**	**0.40**	**0.39**
Experiment_7	AvgWord2Vec(200)	0.39	0.40	0.39	0.39
Experiment_8	**AvgWord2Vec(200)**	**0.40**	**0.41**	**0.40**	**0.39**

3.2.1. Table of trials.

3.3. Hyper-parameter tuning

Following hyperparameters were tuned based on suggested trials of the Optuna hyperparameter optimization framework[6] : number of layers and units, learning rate, Optimizer, Dropout ratio.

3.4. Optimization techniques

For the first two experiments I used the default optimization search space given in the **Optuna** tutorial in github (pytorch_simple.py) for multiclass classification[7]. Specifically, the given tutorial optimizes the following hyperparameters giving the following search spaces:

- N_layers (1-3)
- N_units (4-128)
- Dropout ratio (0.2-0.5)
- Otimizer (Adam, RMSprop, SGD)
- Learning rate: (1e-5, 1e-1)
- BATCHSIZE =128
- EPOCHS = 10
- Loss func: CrossEntropyLoss
- N_trials: 100
- Direction: maximize accuracy

Pruning on non-promising trials is also implemented.
Also, the train and validation data get limited during training and validation for faster epochs. Train data are limited to the size of the Batch size*30 and validation are limited to the size of the batch size*10.

Learing rate scheduling: Exponential LR (Experiments 6,7,8)
The Exponential Learning Rate scheduling technique divides the learning rate every epoch (or every evaluation period in the case of iteration trainer) by the same factor called **gamma**. Thus, the learning rate will decrease abruptly during the first several epochs and slow down later, with most epochs running with lower values. The learning rate aspires to zero but never reaches it.

- **Gamma** - a multiplicative factor by which the learning rate is decayed every epoch. In our experiments we start with a learning rate of 0.004 and a gamma set to 0.5. At each epoch he learning rate is multiplied by the gamma factor leading to an exponentially dropped learning rate.

[6] https://optuna.org/

[7] https://github.com/optuna/optuna-examples/blob/main/pytorch/pytorch_simple.py

3.5. Evaluation

For the classifier's evaluation the ROC Curves of each model were plotted, along with the Learning curves of the Evaluation and Training Loss and Accuracy and the classification reports.

3.5.1. ROC curve and AUC.

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

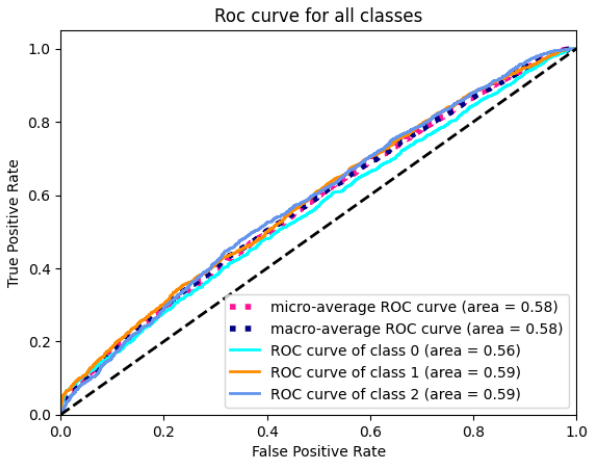
True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

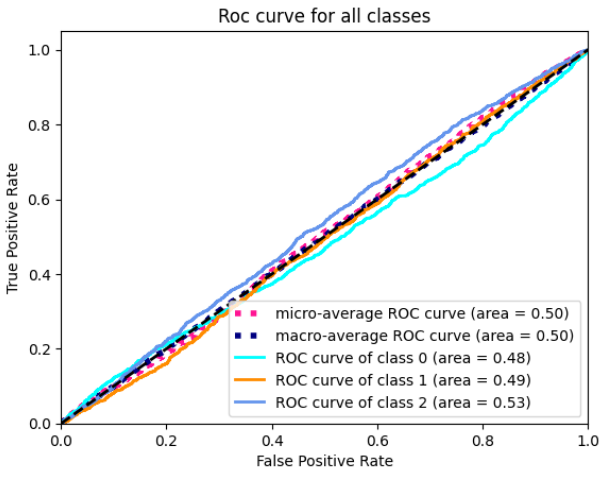
False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

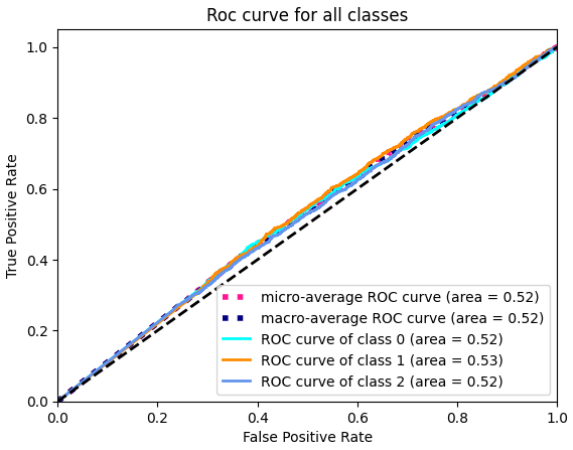
AUC stands for Area Under the Curve, and the AUC curve represents the area under the ROC curve. AUC measures how well a model is able to distinguish between classes. AUC of 0.5 indicates that the model's performance is comparable to that of random chance. It suggests a lack of discriminating ability.



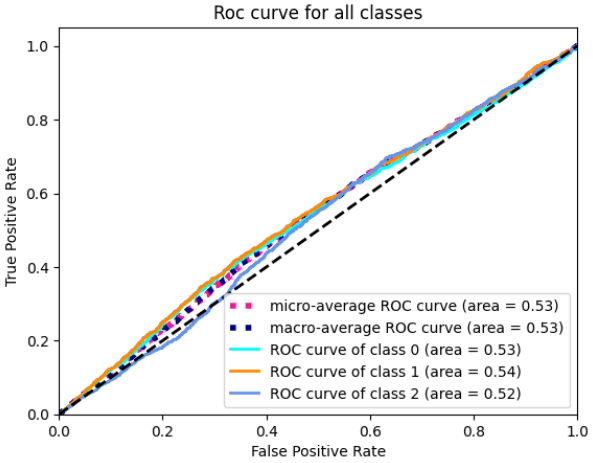
Experiment 1



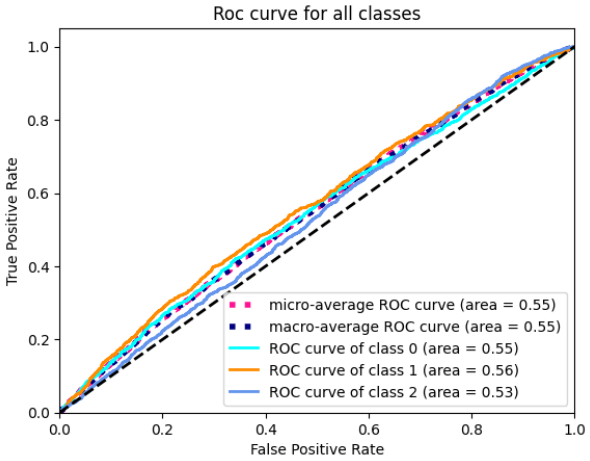
Experiment 2



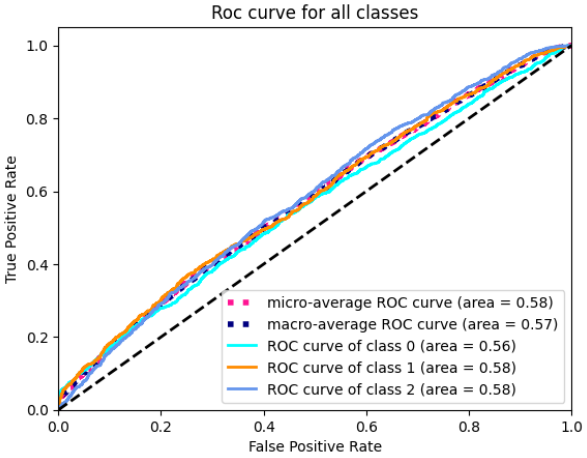
Experiment 3



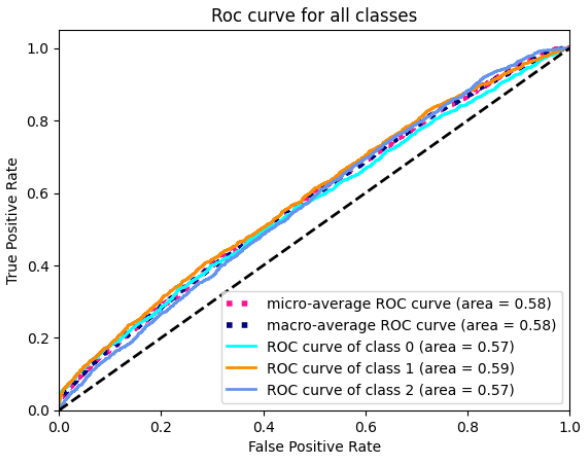
Experiment 4



Experiment 5

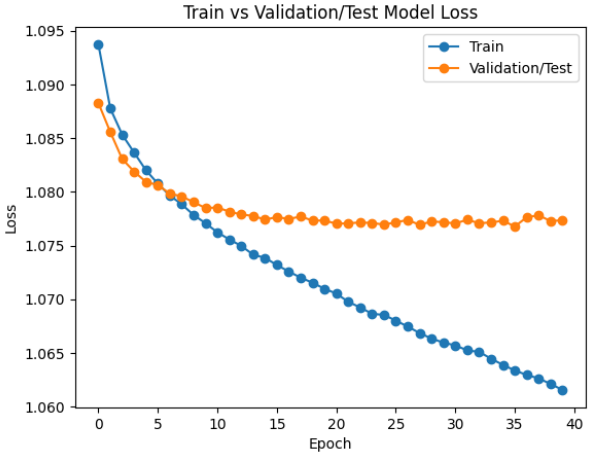


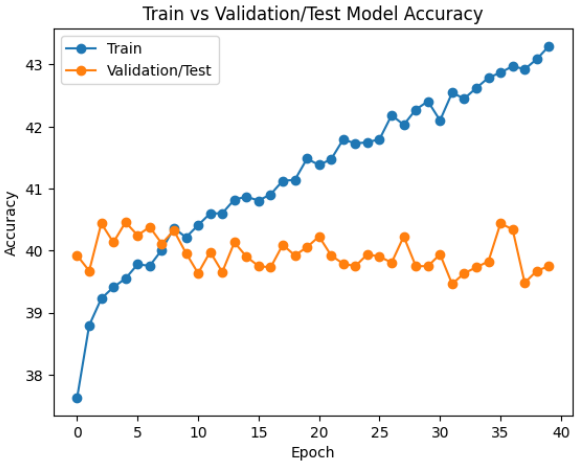
Experiment 6



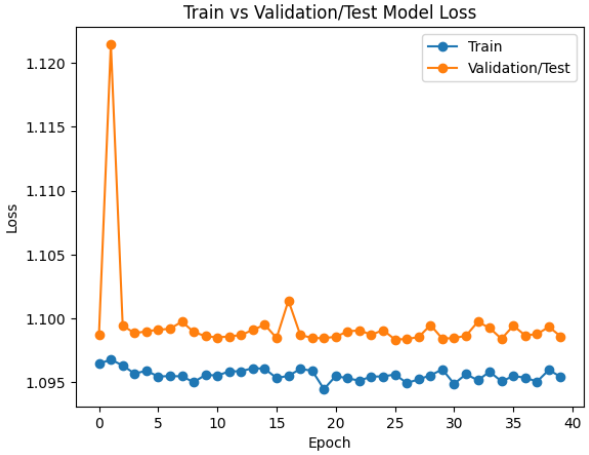
Experiment 7

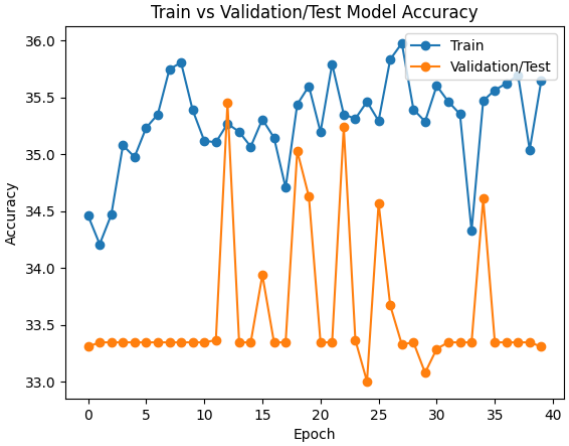
3.5.2. Learning Curves.



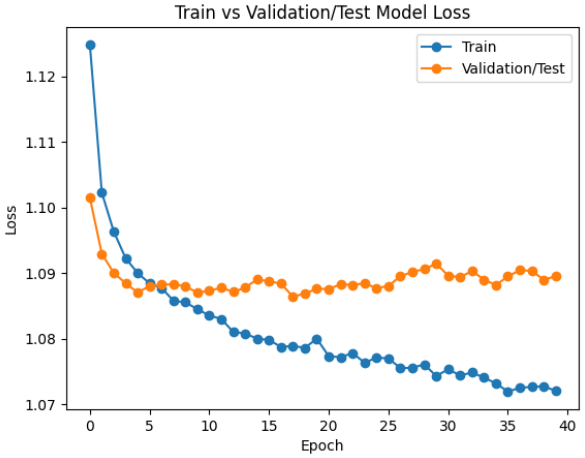


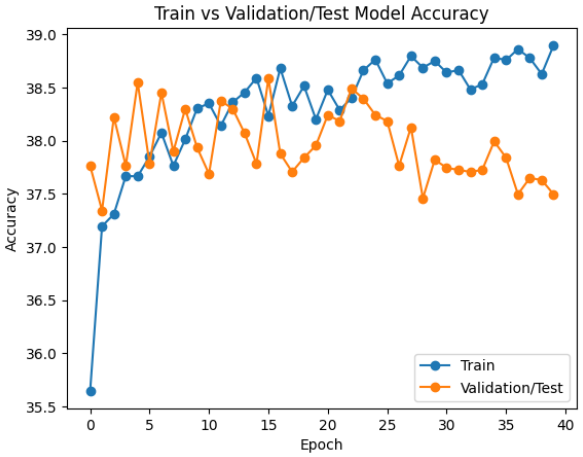
Experiment 1



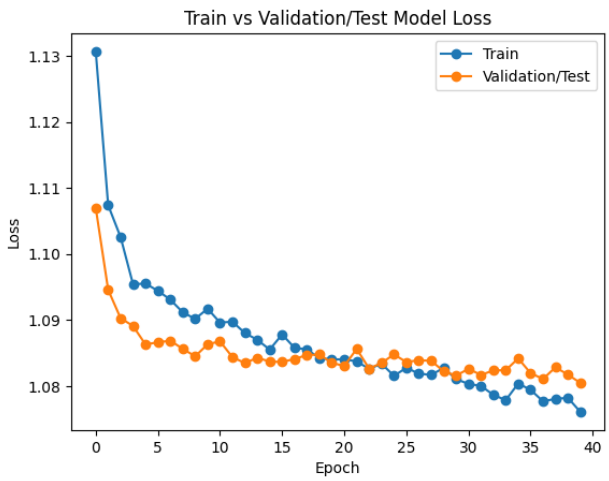


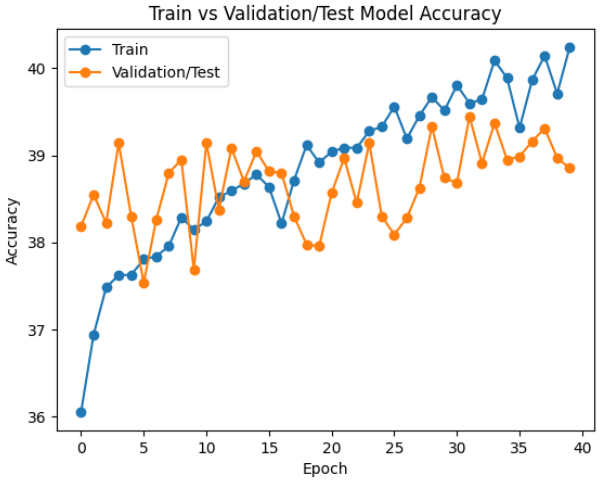
Experiment 2



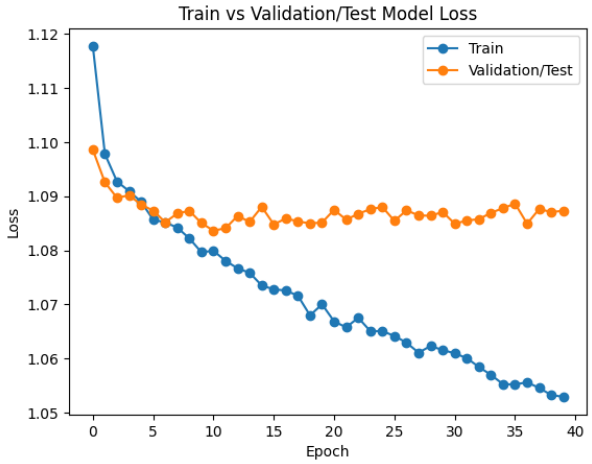


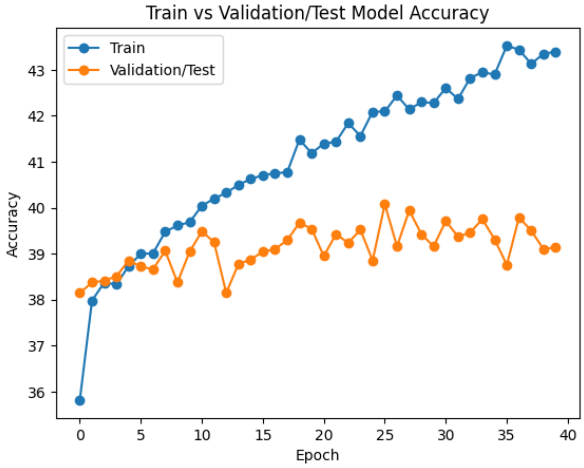
Experiment 3



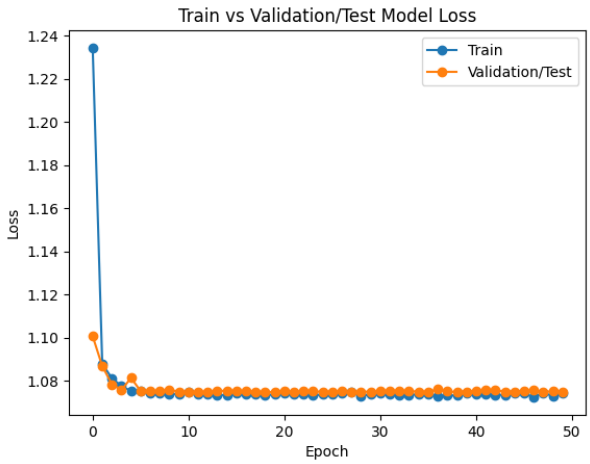


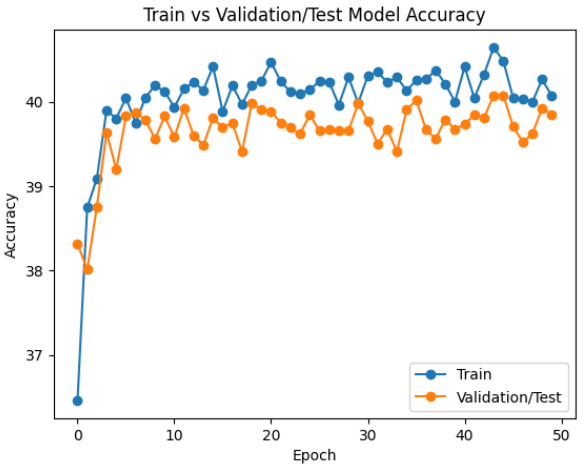
Experiment 4



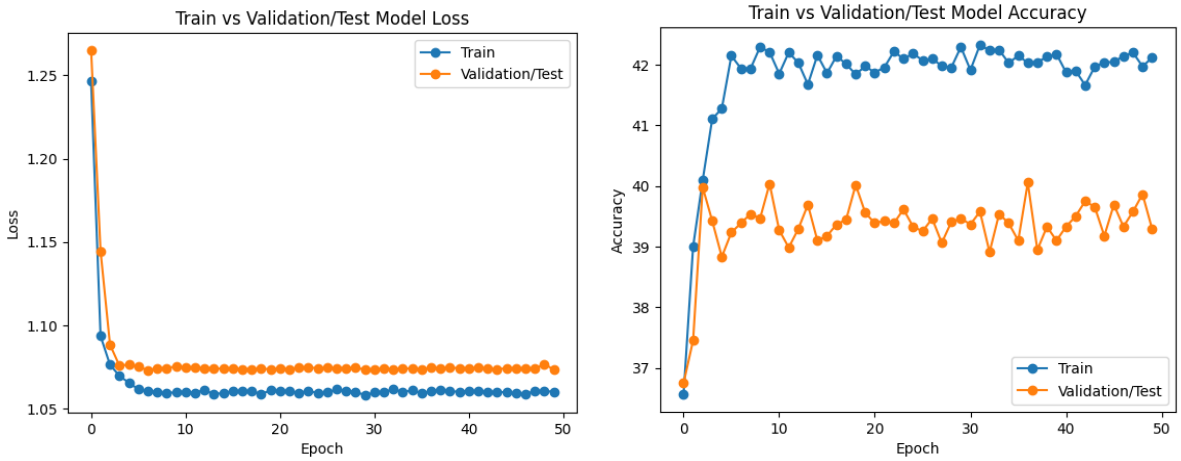


Experiment 5





Experiment 6



Experiment 7

3.5.3. Classification Reports.

	precision	recall	f1-score	support
0	0.40	0.34	0.36	1742
1	0.41	0.36	0.38	1744
2	0.38	0.49	0.43	1744
accuracy			0.39	5230
macro avg	0.40	0.39	0.39	5230
weighted avg	0.40	0.39	0.39	5230

Experiment 1

	precision	recall	f1-score	support
0	0.33	1.00	0.50	1742
1	0.00	0.00	0.00	1744
2	0.00	0.00	0.00	1744
accuracy			0.33	5230
macro avg	0.11	0.33	0.17	5230
weighted avg	0.11	0.33	0.17	5230

Experiment 2

	precision	recall	f1-score	support
0	0.36	0.48	0.42	1742
1	0.39	0.33	0.36	1744
2	0.38	0.31	0.34	1744
accuracy			0.37	5230
macro avg	0.38	0.37	0.37	5230
weighted avg	0.38	0.37	0.37	5230

Experiment 3

	precision	recall	f1-score	support
0	0.38	0.44	0.41	1742
1	0.40	0.33	0.36	1744
2	0.39	0.40	0.39	1744
accuracy			0.39	5230
macro avg	0.39	0.39	0.39	5230
weighted avg	0.39	0.39	0.39	5230

Experiment 4

	precision	recall	f1-score	support
0	0.38	0.29	0.33	1742
1	0.40	0.44	0.42	1744
2	0.39	0.45	0.42	1744
accuracy			0.39	5230
macro avg	0.39	0.39	0.39	5230
weighted avg	0.39	0.39	0.39	5230

Experiment 5

	precision	recall	f1-score	support
0	0.41	0.27	0.33	1742
1	0.41	0.39	0.40	1744
2	0.39	0.53	0.45	1744
accuracy			0.40	5230
macro avg	0.40	0.40	0.39	5230
weighted avg	0.40	0.40	0.39	5230

Experiment 6

	precision	recall	f1-score	support
0	0.40	0.31	0.35	1742
1	0.41	0.40	0.40	1744
2	0.38	0.47	0.42	1744
accuracy			0.39	5230
macro avg	0.40	0.39	0.39	5230
weighted avg	0.40	0.39	0.39	5230

Experiment 7

4. Results and Overall Analysis

4.1. Results Analysis

Considering the evaluation results of each experiment we can make the following comments:
Experiment_1: A single-hidden layer feedforward neural network with 95 neurons, dropout and batch normalization as regularization techniques, ReLU activation function, Adam optimizer, a learning rate of 0.0001, batch size of 10 and training epochs of 40 gave us a model which is **overfitting** on the training data.

Experiment_2: A single-hidden layer feedforward neural network with significantly lower number of neurons (13), a higher learning rate of 0.005, RMSprop Optimizer, and same dropout , batch normalization, epochs and batch size gave tremendously lower results with a weighted average F1 score of 0.17. The gap in the learning curve of the loss is not as big compared to experiment 1 but the validation loss is higher. Two factors that may have caused this behavior: the smaller number of neurons, the vectorization method used in the second experiment, in which each dataset was vectorized separately during preprocessing. Due to the smaller stemmed vocabulary of the validation set, it is possible that the vector representation of the tweets is less representative compared to training in stemmed words in the whole vocabulary as done in Experiments 1 and 2-8. Also, the higher learning rate may have affected the model's behavior.

Experiment_3: We see that even with Self-normalizing neural networks, the issue of overfitting of the first experiment is not being resolved. This can be due to the fat that addressing the issue of unstable gradients is mainly a problem in deeper neural networks, not in single layer ones.

Experiment_4: Randomized Leaky ReLU proved indeed as a good alternative of ReLU for addressing the issue of overfitting. Also, the use of Glorot weight initialization along with feature scaling, dropout, and batch normalization as regularization techniques and the same rest hyperparameters with Experiment_1 may have contributed to the model's improved fit.

Experiment_5: We see that without the use of Dropout, the model of Experiement_4 is significantly overfitting.

Experiment_6: The same architecture, weight initialization and regularization techniques with Experiment_4, along with a higher learning rate of 0.005, being exponentially scheduled to drop by a multiplicative factor of 0.5 at every epoch. Also the batch size was significantly increased from 10 to 128 and the number of epochs were increased by 10. These changes gave us a very quick decrease and stabilization of the validation and training loss at the same levels, so that the model seems to be fitting well on the data with no overfitting.

Experiment_7: Without the use of dropout the model is still not overfitting the train data, but there is a small gap of 2% between training and validation loss. Also, in terms of training and validation accuracy, the results indicate a bigger gap between training and validation data.

Experiment_8 (submission): The last model was trained with slightly less neurons compared to Experiment 6 and with Kaiming/He weight initialization, to conform with the suggested combination for the ReLU variants. The results showed no important difference compared to Experiment 6. The validation loss and accuracy seem to be slightly outperforming the training loss and accuracy without this indicating a bad performance. For submission the checkpoint of the model with the lowest validation loss at epoch 50 was chosen.

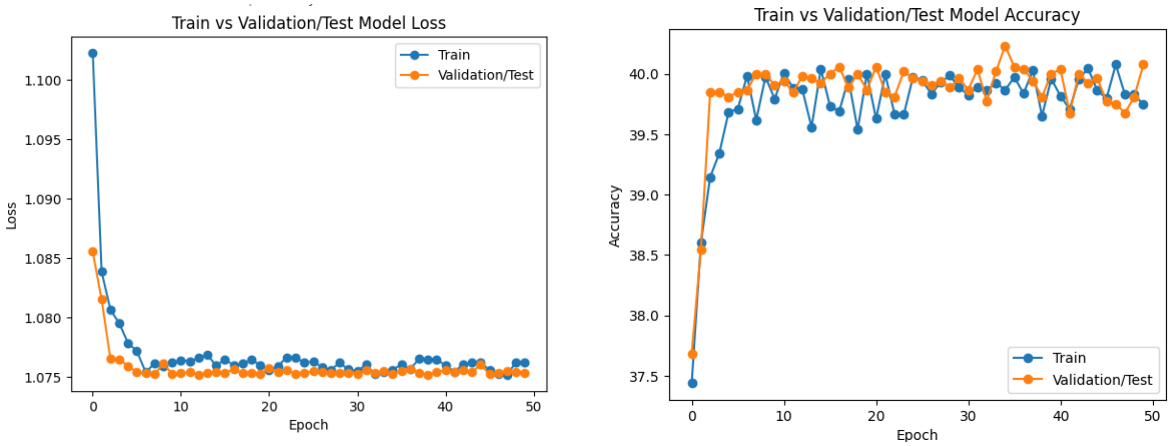
Observations/ Thoughts:

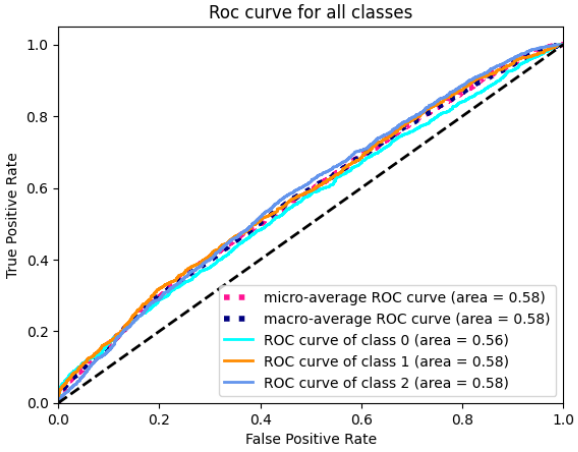
- Randomized Leaky ReLU helped to avoid overfitting and was a good alternative to ReLU
- Exponential LR scheduling helped to reduce validation loss in less epochs.
- Optuna hyperparameter tuning gave almost all times a single layer perceptron.
- Building a word2vec model for each dataset separately may have created less representative word embeddings, which led to worse model performance during training and inference.
- Addressing the issue of unstable gradients is mainly a problem in deeper neural networks, so that's why Self Normalizing neural Nets and weight initialization did not help in our network architecture.
- Batch normalization made our network less sensitive to weight initialization and it proved important for reducing overfitting.
- The use of different output activation function (Sigmoid/Softmax) did not make any difference to the experiment results.

Further experimentation:

- Regularization techniques: Max-Norm Regularization instead of Batch normalization
- Try out other schedulers: one-cycle scheduling.
- Try Early-stopping.

4.1.1. Best trial.





	precision	recall	f1-score	support
0	0.42	0.23	0.30	1742
1	0.41	0.39	0.40	1744
2	0.39	0.58	0.46	1744
accuracy			0.40	5230
macro avg	0.41	0.40	0.39	5230
weighted avg	0.41	0.40	0.39	5230

Experiment 8

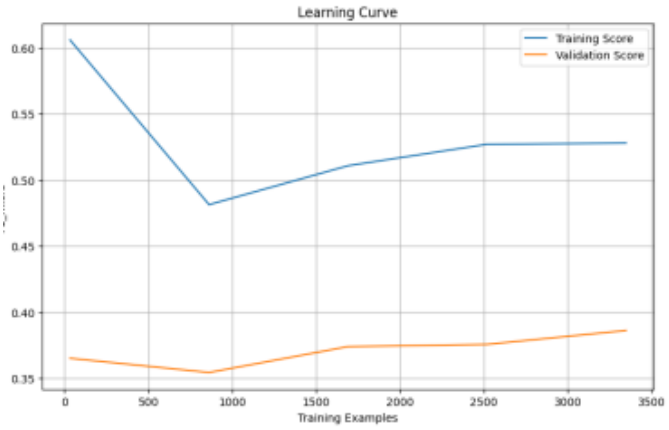
4.2. Comparison with the first project

Comparing the best model results with those of the best model on the previous project (TfIdf unigrams) max_features 1000 +3 numeric, L1 regularisation, saga, C: 1.0, trained with Logistic Regression), we observe two almost identical classification reports. It is worth noting that on the first project a different data partitioning approach was followed. Both train and validation data were splitted into train/test and dev/test with a ratio of 80/20. This is why on the classification report below we see less data (1047).

Performance of the best trial on the validation data

Classification report:

	precision	recall	f1-score	support
-1	0.39	0.34	0.36	360
0	0.38	0.50	0.44	342
1	0.43	0.34	0.38	345
accuracy			0.40	1047
macro avg	0.40	0.40	0.39	1047
weighted avg	0.40	0.40	0.39	1047



Considering learning curves in project_1 we plotted learning curves of F1 score on train and validation data. It was observed that all models were overfitting on the train data, and this was accounted due to the use of a linear model which is probably not suitable for our data since they are not linearly separable. In project_2 we managed to overcome the issue of the model's overfitting by achieving to get training and validation losses on the same level, with a stable behavior. The model's performance did not increase, and the classifier keeps being a random one, but at least now the model is fitting equally to both train and

validation data. One very important drawback considering the architecture of a Feedforward Neural network is that it has connection only in one direction (upstream nodes deliver output to downstream nodes), meaning that a text sequence is being seen only from left to right and not the other direction. In our vector representations this is not important, because the Averaging of Word2vec for each tweet has already created a representation which does not take word order into account. Following the quote "A model is only as good as the data that it is fed", I believe that not only the false labels in the existing data are a cause of mistakes during calculating the loss function, but also the vector representation itself. For our task it will be more beneficial to apply sentence embeddings like Sentence-BERT, to derive semantically more meaningful vector representations.

5. Bibliography

References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. https://arxiv.org/pdf/1502.01852.pdf

[2] Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). *Self-Normalizing Neural Networks*. https://arxiv.org/pdf/1706.02515.pdf

[3] *Papers-in-100-Lines-of-Code/Self_Normalizing_Neural_Networks/selu.py at main · MaximeVandegar/Papers-in-100-Lines-of-Code*. (n.d.). GitHub. Retrieved December 22, 2023, from https://github.com/MaximeVandegar/Papers-in-100-Lines-of-Code/blob/main/Self_Normalizing_Neural_Networks/selu.py

[4]Creator, C. (2023, September 16). *Randomized Leaky ReLU (RReLU) Activation Function*. Medium. https://ai.plainenglish.io/randomized-leaky-relu-rrelu-activation-function-a4475337b81e#5d8b

[5]SNNs/Pytorch/SelfNormalizingNetworks_MLP_MNIST.ipynb at master · bioinf-jku/SNNs. (n.d.). GitHub. Retrieved December 22, 2023, from https://github.com/bioinf-jku/SNNs/blob/master/Pytorch/SelfNormalizingNetworks_MLP_MNIST.ipynb

[6]Quijas, J. (2021, March 4). *Solving the Vanishing Gradient Problem with Self-Normalizing Neural Networks using Keras*. Medium. https://medium.com/towards-artificial-intelligence/solving-the-vanishing-gradient-problem-with-self-normalizing-neural-networks-using-keras-59a1398b779f

[6]Monigatti, L. (2022, December 6). *A Visual Guide to Learning Rate Schedulers in PyTorch*. Medium. https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863

[7]How to test model with unseen data. (2021, May 14). PyTorch Forums. https://discuss.pytorch.org/t/how-to-test-model-with-unseen-data/121218/2

[8]Weight Initialization and Activation Functions - Deep Learning Wizard. (n.d.). Www.deeplearningwizard.com. https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/weight_initialization_activation_functions/

[9]ExponentialLR. (n.d.). CloudFactory Computer Vision Wiki. Retrieved December 22, 2023, from https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/exponentiallr