UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Aikaterini-Methodia Zacharioudaki*

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Spring Semester 2025*

Contents

1. Abstract

The task of this assignment is to develop a sentiment classifier for a tweet dataset. We will convert our data to GloVe Embeddings and develop a feed forward neural network, using PyTorch, for classification.

2. Data processing and analysis

2.1. Pre-processing

We follow the steps below for pre-processing:

- Demojize the tweets and replace emoticons with their descriptive words
- Make lowercase the entire text
- Remove HTML tags and emails
- Replace tags and links with the words 'user' and 'link'
- Replace timestamps with the word time
- Remove " ' " so that contractions are one word, "didn't" becomes "didnt"
- Remove multiple consecutive letters, leave only two
- Replace some common misspelt words with the correct spelling
- Replace common slang and abbreviations with the correct words
- Remove numbers
- Remove all non-word and non-whitespace characters
- Replace all excess whitespace characters with a single space
- Lemmatize all words
- Tokenize the sentence

We also tested removing stopwords, but then some sentences were empty so we decided against that.

2.2. Vectorization

We will convert our data to Word2Vec Embeddings. These vector representations of words reflect the similarity and relationship among the corresponding words, thus enabling us to better understand the sentiment of a sentence. To achieve that we will use the GloVe Twitter pre-trained word vectors, as our dataset is also tweets, meaning that the intricacies of Twitter will be better identified.

3. Algorithms and Experiments

3.1. Baseline Model

We first implement a simple model that will serve as our baseline.

```
class BaseNN(nn.Module):
def __init__(self):
    super().__init__()
    self.linear_stack = nn.Sequential(
        nn.Linear(200, 1)
    )

def forward(self, x):
    out = self.linear_stack(x)
    return out
```

Listing 1: Baseline Model

With this model we achieve an accuracy score of 0.7047 and a mean loss of 0.6223. We will improve the model to raise these scores.

3.2. Optimization techniques

We will use the Optuna framework to optimize our model. We will experiment with the architecture of the model as well as the optimizer.

3.3. Hyper-parameter tuning

Specifically, we will experiment with the following parameters:

- n_layers : How many hidden layers the model will have, apart for the last Linear layer that is added manually. We choose from: 1, 2
- activation : The activation function to add between the Linear layers. We choose from the list: nn.ReLU, nn.LeakyReLU, nn.Sigmoid, nn.Tanh, nn.SiLU
- dropout : The probability for the nn.Dropout layer. We choose from: 0.0 (no dropout), 0.1, 0.2, 0.3
- batchnorm : Whether to include a layer of nn.BatchNorm1d in our model. We choose from: True, False
- h1 : The output dimension of the first hidden layer/the input dimension of the second hidden layer. We choose from: 32, 64, 128, 256
- div : This is used to calculate the output dimension of the second hidden layer/the input dimension of the third layer, if we have 3 hidden layers in total. We choose from: 2, 4. For example, if we have 3 layers and h1 $= 128$ and div $= 2$, then the output of the second layer is $128/2 = 64$
- optimizer : The optimizer to use. We choose from: torch.optim.Adagrad, torch.optim.Adam, torch.optim.AdamW, torch.optim.Adamax, torch.optim.NAdam, torch.optim.RAdam

3.4. Experiments

We create an Optuna study with the objective to maximize the accuracy. We set the number of trials to be 300, using the optuna.pruners.MedianPruner and the optuna.samplers.TPESampler to help the study run faster and more efficiently.

Below are the results of a sample of 30 trials

3.4.1. Table of trials.

Trial	n_layers	activation	dropout	batchnorm	h1	div	optimizer	Score
1	2	SiLU	0.0	False	128	2	Adam	0.7578
2	2	SiLU	0.3	True	64	2	NAdam	0.7660
3	2	SiLU	0.1	False	64	2	Adagrad	0.7075
4	1	Tanh	0.0	True	32	4	Adam	0.7605
5	2	Sigmoid	0.1	False	32	2	Adagrad	0.4999
6	1	Tanh	0.1	True	64	2	AdamW	0.7500
7	1	ReLU	0.3	False	128	4	NAdam	0.7693
8	1	LeakyReLU	0.0	False	32	2	AdamW	0.7655
9	2	LeakyReLU	0.2	True	256	4	RAdam	0.7774
10	2	ReLU	0.2	True	256	4	RAdam	0.7740
11	2	Sigmoid	0.2	True	256	4	RAdam	0.7488
12	2	Tanh	0.3	True	256	4	RAdam	0.7703
13	2	Sigmoid	0.1	True	256	4	Adam	0.7550
14	2	LeakyReLU	0.2	True	128	4	NAdam	0.7731
15	2	LeakyReLU	0.1	False	128	2	Adam	0.7809
16	2	SiLU	0.1	False	128	2	Adam	0.7617
17	2	LeakyReLU	0.0	False	128	2	Adam	0.7821
18	1	ReLU	0.0	False	128	2	Adam	0.7755
19	2	LeakyReLU	0.0	False	128	2	AdamW	0.7823
20	2	ReLU	0.0	False	128	2	AdamW	0.7830
21	2	ReLU	0.3	True	128	4	AdamW	0.7838
22	2	ReLU	0.2	True	64	2	AdamW	0.7842
23	1	ReLU	0.3	False	256	4	AdamW	0.7847
24	1	ReLU	0.3	True	128	4	Adamax	0.7860
25	2	ReLU	0.1	False	256	4	Adamax	0.7865
26	2	ReLU	0.2	False	256	4	Adamax	0.7872
27	2	ReLU	0.3	False	256	4	NAdam	0.7873
28	2	ReLU	0.3	False	256	4	Adamax	0.7880
29	2	ReLU	0.3	False	128	2	NAdam	0.7887
30	2	ReLU	0.3	False	128	2	AdamW	0.7894

Table 1: Trials

3.5. Evaluation

To evaluate our model, we will use the following metrics:

- **Accuracy** : The proportion of all predictions that are correct. It is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Confusion matrix** : In our case a 2×2 matrix that displays the number of true positives, true negatives, false positives, and false negatives. This aids in calculating the accuracy and other metrics.

- **Loss Curve** : A plot of loss as a function of the number of training iterations. The loss curve features the training loss and the validation loss. When plotting both, it measures the model's performance on a separate set of data not used during training, helping identify when the model is just memorizing the training data (overfitting) rather than truly understanding the patterns.

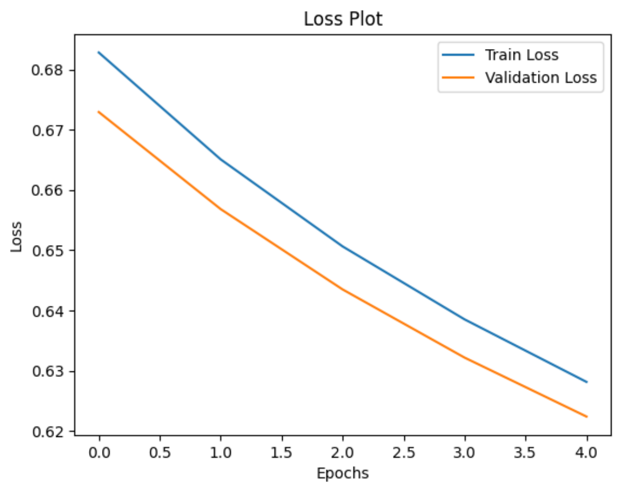

Figure 1: Loss Curve for Baseline Model

- **ROC curve** : Illustrates the performance of a classifier model at varying threshold values. The ROC curve is the plot of the true positive rate (TPR) against the false positive rate (FPR) at each threshold setting.

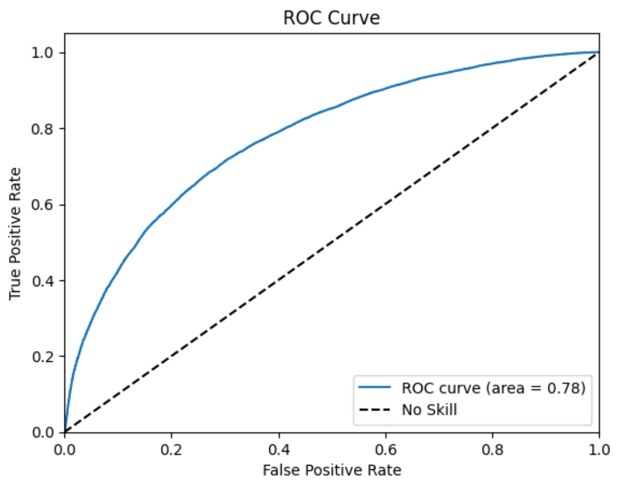

Figure 2: ROC Curve for Baseline Model

4. Results and Overall Analysis

4.1. Results Analysis

After running the optuna study, we determined that the following model was the best.

```
    class FFNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_stack = nn.Sequential(
            nn.Linear(200, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        out = self.linear_stack(x)
        return out
```

Listing 2: Final Model

We trained this model for 50 epochs, but the training function returns the model from the 45th epoch, as it is the one with the best F1 score. As seen in the loss curve (Figure 3) below, after the 45th epoch the validation loss increases, implying that the model starts overfitting. With this model, we achieve an accuracy of 0.7868 in the validation set and an accuracy of 0.7879 in the test set.

4.1.1. Best trial.

Metric	Score
Accuracy	0.7868
Recall	0.7675
Precision	0.7982
F1 score	0.7826

Confusion Matrix:

		Predicted	
		0	1
True	0	17085	4112
	1	4927	16272

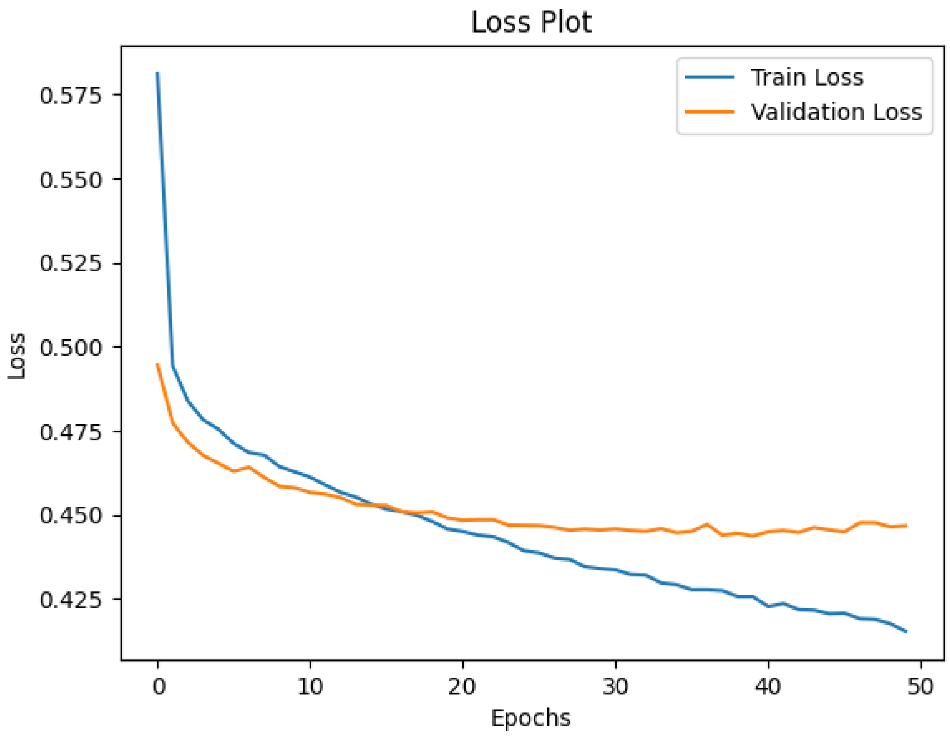

Figure 3: Loss Curve for Best Model

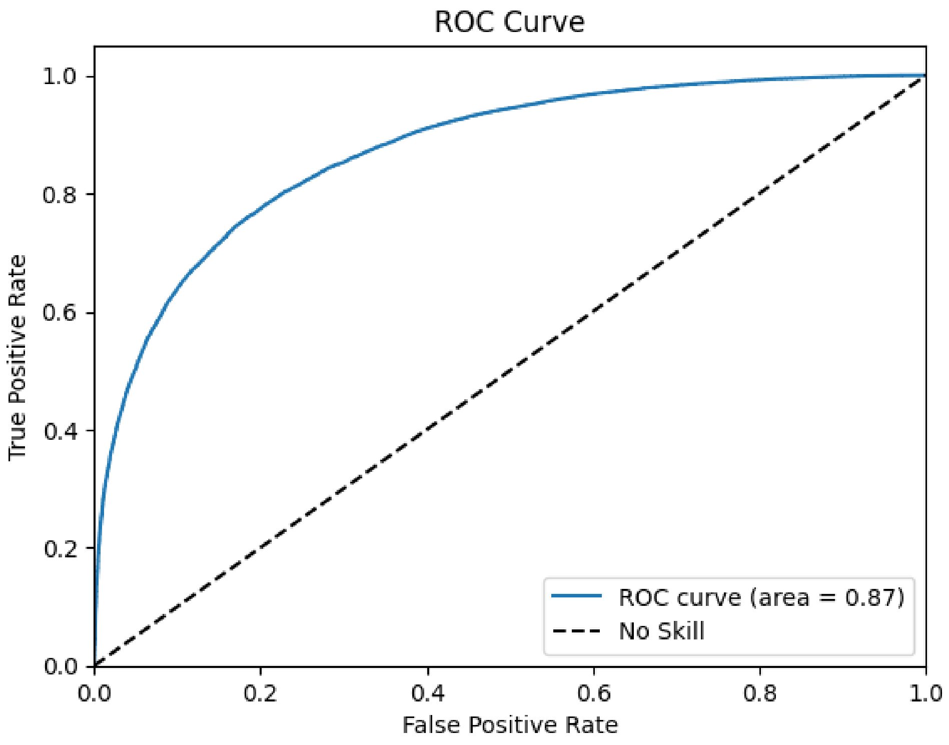

Figure 4: Loss Curve for Best Model