UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Artemis Lazanaki*
sdi: sdi2100081

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

Contents

1. Abstract

This project addresses the problem of sentiment classification on English-language tweets. The goal is to assign each tweet a binary label: positive or negative sentiment. We use pre-trained Word2Vec embeddings to convert tweets into vector representations and train a simple yet effective Multi-Layer Perceptron (MLP) classifier on top of these vectors.

2. Data processing and analysis

2.1. Pre-processing

To handle the informal, noisy, and user-generated nature of Twitter data, we designed a robust preprocessing pipeline tailored to sentiment analysis. The preprocessing aimed to normalize variations, reduce vocabulary sparsity, and preserve sentiment-relevant information.

The full pipeline includes the following transformations:

- **Mojibake Fixing**: Corrects corrupted Unicode text using the `ftfy` library.
- **Lowercasing**: Converts all characters to lowercase for case-insensitive modeling.
- **URL and Mention Replacement**: Substitutes hyperlinks and @mentions with neutral placeholders (`url`, `username`) to reduce noise.
- **Repetition Normalization**: Limits repeated characters to a maximum of two to normalize elongated expressions (e.g., `soooo` $\rightarrow$ `soo`).
- **Emoticon and Symbol Substitution**: Replaces emoticons and symbols (e.g., `<3`,) with descriptive sentiment tokens (e.g., `love`, `heart`).
- **Punctuation Cleanup**: Removes specific symbols and replaces remaining punctuation with whitespace to standardize token boundaries.
- **Slang Replacement**: Expands common informal expressions (e.g., `gr8`, `bcuz`, `fkin`) using a manually crafted dictionary.
- **Short Word Removal**: Discards words with one or two characters, which tend to be noisy and semantically weak.
- **Contraction Expansion**: Rewrites common contractions (e.g., `can't`, `he'll`) to their expanded forms to support better syntactic and polarity parsing.
- **Negation Handling**: Marks the word following a negator (e.g., `not`, `never`) with a `NOT_` prefix (e.g., `not happy` $\rightarrow$ `NOT_happy`) to preserve polarity shifts.
- **Whitespace Normalization**: Collapses multiple spaces and trims the final result.

This pipeline was applied prior to tokenization and Word2Vec training. Each transformation was selected based on its linguistic motivation and observed utility in handling the noisy structure of tweets. A more detailed performance analysis of each step is presented in the experiments section.

2.2. Vectorization

To convert tweets into fixed-length numerical representations suitable for neural modeling, we employed a Word2Vec embedding approach. The model was trained from scratch using the `gensim` implementation with the skip-gram algorithm (`sg=1`), which is well-suited for learning high-quality embeddings from small texts such as tweets.

Each tweet was first preprocessed using one of the cleaning pipelines described earlier, and then tokenized using simple whitespace splitting. These token sequences were used to train the Word2Vec model on the training set alone, ensuring that validation data remained untouched during embedding learning.

The key Word2Vec parameters were treated as tunable hyperparameters in our study. We used **Optuna** to explore different values for:

- `vector_size` (e.g., 200 or 300)
- `window` size for context
- `min_count` for vocabulary inclusion

All models were trained deterministically by setting a fixed seed and restricting training to a single thread (`workers=1`). Detailed analysis of Optuna's impact on performance is presented in the Experiments section.

After training, each tweet was represented by aggregating the vectors of its constituent tokens. We evaluated three common strategies:

- **Mean Pooling**: Computes the average of all token vectors. This method was used in the preprocessing experiments.
- **Sum Pooling**: Sums token vectors element-wise. This was used in the Optuna search for best model configurations.
- **Max Pooling**: Takes the element-wise maximum, highlighting dominant word features.

The resulting tweet vectors—of size 200 or 300 depending on the embedding dimensionality—were optionally standardized using `StandardScaler`. We tested different scaling modes (`with_mean`, `with_std`) to assess their influence on downstream classification. These decisions were also part of the Optuna tuning framework and are discussed in detail in the experimental analysis.

3. Algorithms and Experiments

3.1. Preprocessing

Our baseline sentiment classifier consists of a feedforward neural network trained on tweet representations constructed from raw, unprocessed text. Each tweet was tokenized by simple whitespace splitting, with no normalization, filtering, or cleaning applied.

We trained a skip-gram Word2Vec model with 300-dimensional embeddings on this raw token set. For each tweet, token embeddings were aggregated via mean pooling to produce a single vector input. These were fed into a 4-layer multilayer perceptron (`BaselineNet`) with ReLU activations. The architecture included hidden layers of size 128, 64, and 32, followed by a final linear layer projecting to a single output neuron. Since we framed sentiment analysis as a binary classification problem, we applied the `BCEWithLogitsLoss` function during training. The optimizer used was Adam with a fixed batch size of 64 and no weight decay.

This baseline setup achieved a weighted F1 score of 0.7460 on the validation set. It serves as the reference configuration against which all other preprocessing variants were evaluated, using the same Word2Vec and MLP architecture but with different cleaned input texts.

To assess the impact of preprocessing, we designed a series of increasingly complex pipelines applied to the tweet text before tokenization. Starting from only minimal transformations (e.g., mojibake fixing and lowercasing), we incrementally added URL and mention masking, repeated letter reduction, emoticon normalization, punctuation cleanup, slang replacement, short word removal, contraction expansion, and negation handling.

For each of the nine preprocessing configurations:

- A new Word2Vec model was trained on the cleaned token sequences.
- Aggregation was performed via **mean pooling**, as in the baseline.
- The same `BaselineNet` model was trained for five epochs using a fixed random seed and batch size of 64.

Table 1: Validation performance after 5 training epochs across preprocessing levels. Tweet vectors were computed using **mean aggregation** of Word2Vec embeddings.

#	**Preprocessing Setup**	**Accuracy**	**Precision**	**Recall**	**F1**
1	No preprocessing	0.7464	0.7480	0.7464	0.7460
2	Fix Mojibake + Lowercase	0.7554	0.7615	0.7554	0.7539
3	+ URL + Mention removal	0.7593	0.7600	0.7593	0.7591
4	+ Repetition reduction	0.7622	0.7641	0.7622	0.7618
5	+ Emoticon and punctuation normalization	0.7708	0.7735	0.7708	0.7702
6	+ Slang replacement	0.7725	0.7730	0.7725	0.7724
7	+ Short word removal	0.7732	0.7734	0.7732	0.7731
8	+ Contraction expansion	0.7719	0.7724	0.7719	0.7718
9	+ Negation handling	**0.7758**	**0.7778**	**0.7758**	**0.7754**

We observe consistent improvements across preprocessing levels, with F1 rising from 0.7460 (no preprocessing) to 0.7754 (full preprocessing). Emoticon normalization and punctuation cleanup (Step 5) provided a major boost, while final gains came from slang normalization, short word removal, and negation handling. Interestingly, contraction expansion (Step 8) did not improve performance over Step 7, but it was a necessary precondition for negation marking. Overall, full preprocessing improved performance by **+2.9 F1 points** over the baseline.

3.2. Vector Aggregation Strategy

To evaluate the impact of different vector aggregation strategies on tweet-level sentiment classification, we compared three common methods for combining Word2Vec embeddings: mean pooling, sum pooling, and max pooling. The experiment was conducted using tweets that had been fully preprocessed with our best cleaning pipeline (including mojibake fixing, lowercasing, URL and mention replacement, repetition reduction, emoticon and punctuation normalization, slang replacement, short word removal, contraction expansion, and negation handling). A new Word2Vec model was trained on the cleaned tokenized data for each aggregation strategy.

Token embeddings for each tweet were then aggregated using either mean, sum, or max pooling to form a single vector representation. These were fed into the same 4-layer MLP classifier (`BaselineNet`), which was trained for 5 epochs using a fixed seed and identical hyperparameter settings across all runs.

Table 2: Comparison of Word2Vec aggregation strategies for tweet representation. All models trained for 5 epochs on fully preprocessed data using identical architecture and settings.

Aggregation	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Mean	0.7758	0.7778	0.7758	0.7754
Sum	**0.7767**	**0.7781**	**0.7767**	**0.7764**
Max	0.7546	0.7561	0.7546	0.7543

The results show that both mean and sum aggregation yield strong and comparable performance, with sum achieving slightly better results across all metrics. Max pooling, on the other hand, significantly underperformed—likely due to its loss of contextual nuance by selecting only the largest component per dimension. Based on these findings, we recommend sum pooling as the default aggregation method for tweet representations using Word2Vec in this task.

3.3. Feature Standardization

To investigate whether input normalization benefits our classifier, we experimented with applying `StandardScaler` to the aggregated Word2Vec tweet vectors. We tested all four combinations of the scaler's parameters: `with_mean` and `with_std`, each set to either `True` or `False`.

All experiments used sum-aggregated Word2Vec vectors from fully preprocessed tweets and were trained for 5 epochs using the same architecture and optimization settings.

Table 3: Impact of StandardScaler configuration on validation performance.

with_mean	with_std	Accuracy	Precision	Recall	F1 Score
True	True	**0.7807**	**0.7808**	**0.7807**	**0.7807**
True	False	0.7766	0.7766	0.7766	0.7766
False	True	0.7786	0.7787	0.7786	0.7785
False	False	0.7767	0.7781	0.7767	0.7764

Results show that full standardization (`with_mean=True`, `with_std=True`) produces the best validation performance, with a weighted F1 score of **0.7807**. Both partial transformations (only centering or only scaling) also yield improvements compared to using raw vectors, indicating that normalization helps stabilize and accelerate MLP training. Based on this, we adopt full standardization for all downstream experiments.

3.4. Parameters Turing

3.4.1. Optuna Study: Dynamic Word2Vec + 2-Layer MLP (Broad Study). In our first hyperparameter optimization experiment, we explored the combination of dynamic Word2Vec embeddings with a 2-layer multilayer perceptron. Our objective was to tune both the Word2Vec parameters and the MLP architecture to maximize validation accuracy.

We used Optuna to search over the following hyperparameters:

- **Word2Vec**: vector dimension {200, 300}, window size {3–7}, min_count {1–5}, skip-gram (`sg=1`)
- **MLP**: Hidden sizes $H_1 \in [128, 512]$, $H_2 \in [64, H_1]$, activation {ReLU, Leaky ReLU}, dropout rate $\in [0.1, 0.5]$
- **Training**: learning rate $\in [10^{-5}, 10^{-2}]$, optimizer {Adam, AdamW}, weight decay $\in [10^{-6}, 10^{-2}]$, batch size {32, 64, 128}

Each trial involved training Word2Vec from scratch on the full preprocessed tokenized training corpus. For tweet-level vectorization, we used sum pooling and applied standard scaling.

The top-performing configuration in this study achieved an F1 score of **0.7875**, with the following hyperparameters:

- **Word2Vec:** vector_size=300, window=6, min_count=3
- **MLP:** H1=431, H2=276, dropout=0.287, activation=ReLU
- **Training:** optimizer=Adam, lr=1.362×10^{-4}, weight_decay=7.41×10^{-6}, batch size=64

This study revealed that higher-dimensional embeddings and deeper hidden layers tended to perform well, especially when regularized with appropriate dropout and small weight decay. We also observed that ReLU activation generally outperformed Leaky ReLU for this architecture.

3.4.2. Optuna Study: Refined Word2Vec + 2-Layer MLP (Refined Study). Following the promising results of the initial study, we conducted a more focused hyperparameter search. We fixed the embedding dimension to 300 and activation to ReLU, and constrained the search space to ranges identified as effective in the previous study. We also limited the window size and min_count to values where performance had peaked.

The updated search included:

- **Word2Vec:** window {6–7}, min_count {3–4}
- **MLP:** $H_1 \in [350, 480]$, $H_2 \in [200, H_1]$, dropout $\in [0.25, 0.3]$
- **Training:** lr $\in [10^{-4}, 3 \cdot 10^{-4}]$, optimizer {Adam, AdamW}, weight decay $\in [10^{-6}, 10^{-4}]$, batch size {64, 128}

The best trial from this refined study achieved an F1 score of **0.7890**, improving marginally over the previous best. The corresponding configuration was:

- **Word2Vec:** window=7, min_count=4
- **MLP:** H1=375, H2=374, dropout=0.2704
- **Training:** optimizer=AdamW, lr=1.07×10^{-4}, weight_decay=3.57×10^{-6}, batch size=64

These results confirm that performance can be fine-tuned further by focusing on a narrower and more informed hyperparameter range. The refined study not only achieved a slightly better F1 score but also confirmed the importance of compact but expressive MLP layers and moderately regularized training dynamics.

3.4.3. Optuna Study: Dynamic Word2Vec + 3-Layer MLP (Broad Study). To further improve our sentiment classification performance, we extended our architecture to a 3-layer MLP and ran a comprehensive Optuna study optimizing both Word2Vec and MLP hyperparameters. The goal was to maximize validation accuracy.

The search space included:

- **Word2Vec**: vector dimension {200, 300}, window size {4–8}, min_count {1–5}, skip-gram (sg=1)
- **MLP**: Hidden sizes $H_1 \in [128, 512]$, $H_2 \in [64, H_1]$, $H_3 \in [32, H_2]$, activation {ReLU, Leaky ReLU}, dropout $\in [0.1, 0.5]$
- **Training**: learning rate $\in [10^{-5}, 10^{-2}]$, optimizer {Adam, AdamW}, weight decay $\in [10^{-6}, 10^{-2}]$, batch size {32, 64, 128}

Each trial retrained Word2Vec embeddings on the preprocessed dataset and vectorized tweets via sum pooling followed by standardization. Models were trained for 5 epochs and evaluated using validation accuracy.

The best performing configuration achieved an accuracy of **0.7888**, with the following hyperparameters:

- **Word2Vec:** vector_size=300, window=7, min_count=3
- **MLP:** H1=476, H2=362, H3=293, dropout=0.191, activation=ReLU

- **Training:** optimizer=AdamW, lr=2.698×10^{-4}, weight_decay=2.351×10^{-6}, batch size=128

This result showed that deeper architectures can be beneficial, particularly when combined with careful regularization and moderate dropout. While the performance was slightly below our best 2-layer result, the 3-layer model offered more capacity and could be valuable in ensemble settings.

3.4.4. Optuna Study: Dynamic Word2Vec + 3-Layer MLP (Refined Study). Following the insights gained from our initial 3-layer MLP study, we conducted a refined hyperparameter optimization to improve performance. The search space was narrowed based on the top-performing configurations from the first round, focusing on promising ranges for hidden layer sizes, dropout, learning rate, and regularization.
The tuned search space included:

- **Word2Vec:** window ∈ {6, 7}, min_count ∈ {3–4}, with fixed vector_size=300
- **MLP:** $H_1 \in [440, 480]$, $H_2 \in [360, H_1]$, $H_3 \in [290, H_2]$, activation ∈ {ReLU}, dropout $\in [0.13, 0.21]$
- **Training:** learning rate $\in [10^{-4}, 3 \times 10^{-4}]$, optimizer ∈ { AdamW}, weight decay $\in [10^{-6}, 10^{-4}]$, batch size ∈ {64, 128}

The best-performing configuration achieved an accuracy score of **0.7894** and consisted of:

- **Word2Vec:** window=6, min_count=4
- **MLP:** H1=453, H2=395, H3=302, dropout=0.1602, activation=ReLU
- **Training:** optimizer=AdamW, learning rate=1.1958×10^{-4}, weight decay=6.7527×10^{-6}, batch size=64

This study confirmed the effectiveness of slightly deeper architectures with modest regularization and low learning rates. The combination of Leaky ReLU and sum-pooled Word2Vec embeddings continued to perform robustly under these settings.

3.4.5. Optuna Study: Dynamic Word2Vec + 4-Layer MLP (Broad Study). To explore the potential of deeper architectures, we extended our model to a 4-layer MLP and performed a broad Optuna study to optimize both Word2Vec and MLP hyperparameters, targeting validation accuracy.
The search space included:

- **Word2Vec**: vector_size ∈ {200, 300}, window size ∈ {4–8}, min_count ∈ {1–5}
- **MLP**: $H_1 \in [256, 512]$, $H_2 \in [128, H_1]$, $H_3 \in [64, H_2]$, $H_4 \in [32, H_3]$, dropout $\in [0.1, 0.5]$, activation ∈ {ReLU, Leaky ReLU}
- **Training**: optimizer ∈ {Adam, AdamW}, learning rate $\in [10^{-5}, 10^{-2}]$, weight decay $\in [10^{-6}, 10^{-2}]$, batch size ∈ {32, 64, 128}

Each trial retrained Word2Vec on the training data and used sum-pooled embeddings standardized to zero mean and unit variance. The models were trained for 5 epochs and evaluated based on validation accuracy.

The best result achieved an accuracy of **0.7858**, with the following configuration:

- **Word2Vec:** vector_size=300, window=8, min_count=2
- **MLP:** H1=353, H2=212, H3=183, H4=147, dropout=0.1875, activation=ReLU
- **Training:** optimizer=AdamW, learning rate=1.6751×10^{-4}, weight decay=6.7567×10^{-4}, batch size=128

3.4.6. Optuna Study: Dynamic Word2Vec + 4-Layer MLP (Refined Study). Following the broad study, we conducted a second Optuna optimization with a narrowed search space centered around the top-performing hyperparameters. The objective remained maximizing validation accuracy score.

The refined search space included:

- **Word2Vec:** window $\in \{6, 7, 8\}$, min_count $\in$ {3–5}
- **MLP:** $H_1 \in [340, 500]$, $H_2 \in [200, H_1]$, $H_3 \in [160, H_2]$, $H_4 \in [120, H_3]$, dropout $\in [0.15, 0.3]$, activation $\in$ {ReLU, Leaky ReLU}
- **Training:** optimizer $\in$ {AdamW}, learning rate $\in [8 \times 10^{-5}, 3 \times 10^{-4}]$, weight decay $\in [10^{-6}, 10^{-3}]$, batch size $\in \{64, 128\}$

The best configuration from this refined study achieved an F1 score of **0.78899**, and consisted of:

- **Word2Vec:** window=7, min_count=4
- **MLP:** H1=387, H2=359, H3=320, H4=257, dropout=0.1775, activation=ReLU
- **Training:** optimizer=AdamW, learning rate=1.3767×10^{-4}, weight decay=1.0213×10^{-4}, batch size=64

This study confirmed that compact but expressive 4-layer architectures, when coupled with moderate dropout and a carefully tuned learning rate, can deliver strong classification performance. These results informed the construction of our final ensemble models.

3.5. Best Models Evaluation

In this section, we present the performance of our top models, selected via Optuna tuning, across different MLP depths. For each, we include final validation metrics as well as training curves to highlight convergence behavior and generalization performance.

3.5.1. 2-Layer MLP Evaluation. We evaluated our best-performing 2-layer MLP model using multiple diagnostic tools to assess its generalization capability.

Learning Curves. Figure 1 shows the learning curves over 20 epochs. Both training and validation loss steadily decrease, while training and validation accuracy gradually increase. The gap between train and validation accuracy remains small throughout, suggesting the model does not overfit and learns in a stable and generalizable manner.

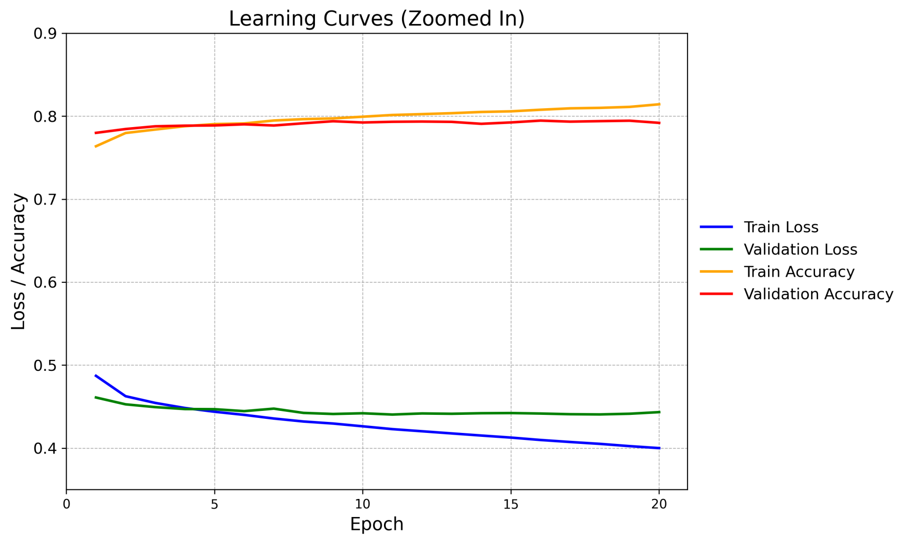

Figure 1: Learning curves (2-layer MLP). Both loss curves decrease smoothly and accuracy curves rise steadily, indicating consistent convergence without overfitting.

Confusion Matrix. The confusion matrix in Figure 2 illustrates the number of true positives, true negatives, false positives, and false negatives. The model performs similarly across both classes, with slightly more false negatives than false positives.

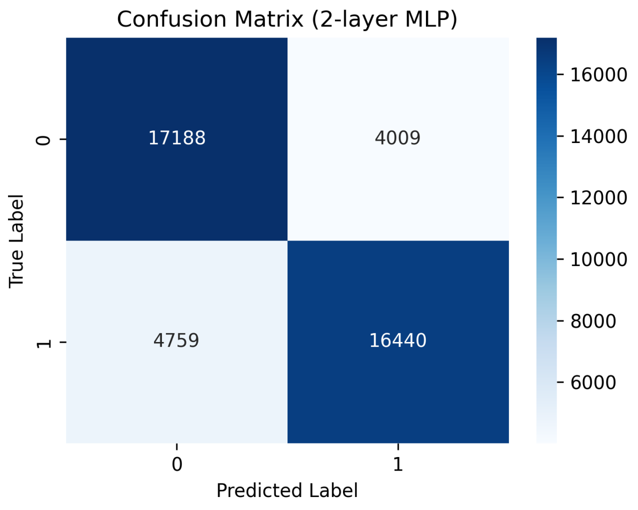

Figure 2: Confusion matrix (2-layer MLP). Class predictions are well-balanced, though class 1 suffers slightly more from false negatives.

ROC Curve. Figure 3 presents the ROC curve, which evaluates the trade-off between true positive rate and false positive rate at various thresholds. The area under the curve

(AUC) is 0.8770, reflecting strong discriminatory power between the two sentiment classes.

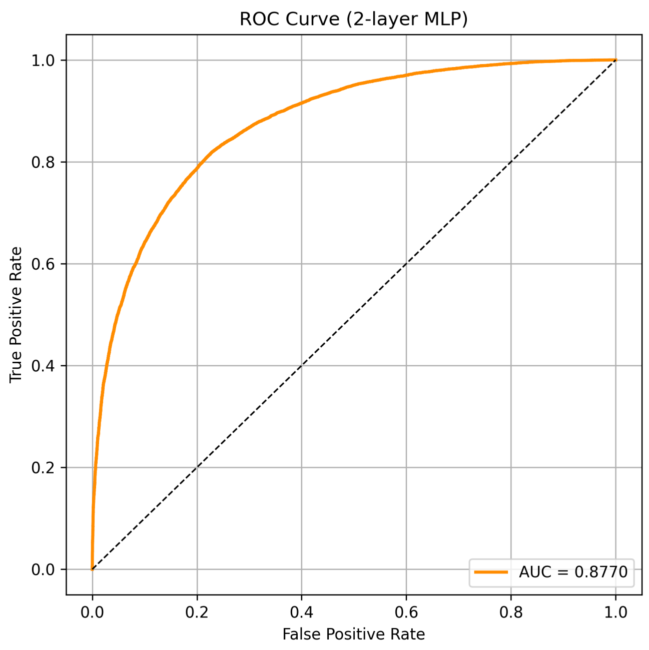

Figure 3: ROC Curve (2-layer MLP). The AUC of 0.8770 demonstrates excellent class separability.

Classification Report. Table 4 shows precision, recall, and F1-score for each class. The model achieves an overall accuracy of 79.32%, with class-wise F1-scores around 0.79–0.80, reflecting balanced and robust performance.

Class	Precision	Recall	F1-score	Support
0	0.7832	0.8109	0.7968	21197
1	0.8040	0.7755	0.7895	21199
Accuracy		0.7932		
Macro Avg	0.7936	0.7932	0.7931	42396
Weighted Avg	0.7936	0.7932	0.7931	42396

Table 4: Classification Report (2-layer MLP). The model maintains consistent and balanced performance across both classes.

3.5.2. 3-Layer MLP Evaluation. We evaluated the best-performing 3-layer MLP model using the same diagnostic framework to analyze its learning behavior and classification performance.

Learning Curves. Figure 4 shows the learning curves over 20 epochs. Training loss steadily decreases, while validation loss plateaus around epoch 6–7. Similarly, the training accuracy continues to rise while the validation accuracy slightly fluctuates. The increasing gap between training and validation accuracy suggests slight overfitting, though generalization remains strong overall.

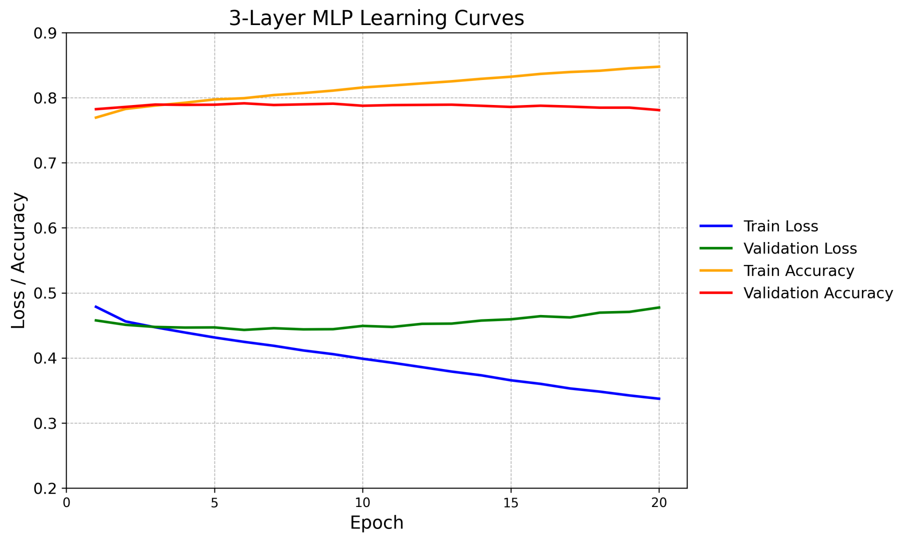

Figure 4: Learning curves (3-layer MLP). Loss and accuracy trends indicate convergence, with mild overfitting beginning after epoch 6.

Confusion Matrix. Figure 5 presents the confusion matrix. The model performs similarly across classes, with a slightly higher number of false negatives (class 1 predicted as class 0), mirroring the behavior seen in the 2-layer model.

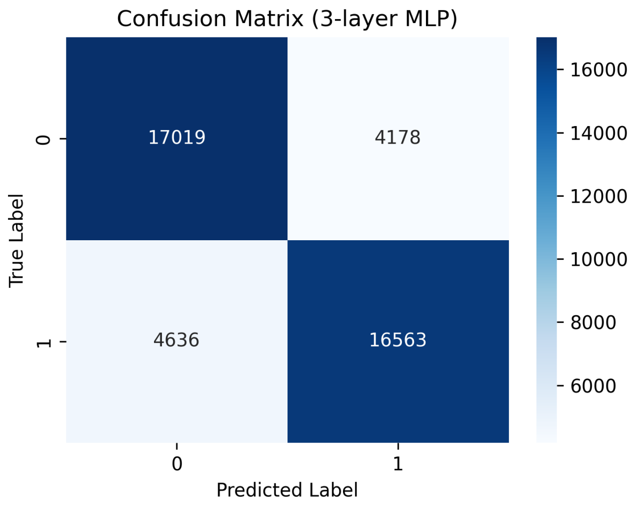

Figure 5: Confusion matrix (3-layer MLP). The distribution of errors is similar to the 2-layer model, with class 1 experiencing slightly more false negatives.

ROC Curve. The ROC curve shown in Figure 6 confirms the model's strong classification ability. The AUC is 0.8751, slightly lower than the 2-layer model but still indicating excellent separability.

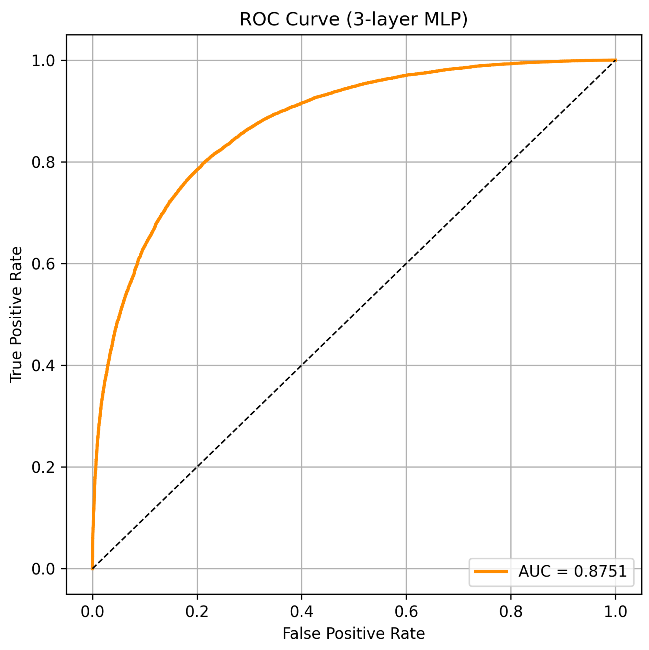

Figure 6: ROC Curve (3-layer MLP). The AUC of 0.8751 reflects high discriminative performance.

Classification Report. Table 5 reports precision, recall, and F1-score for each class. The overall accuracy is 79.21%, with class-wise F1-scores close to 0.79–0.80. The metrics show that the model generalizes well, though marginally behind the 2-layer variant.

Class	Precision	Recall	F1-score	Support
0	0.7859	0.8029	0.7943	21197
1	0.7986	0.7813	0.7898	21199
Accuracy		0.7921		
Macro Avg	0.7922	0.7921	0.7921	42396
Weighted Avg	0.7922	0.7921	0.7921	42396

Table 5: Classification Report (3-layer MLP). Class-wise performance is well-balanced with solid generalization.

3.5.3. 4-Layer MLP Evaluation. We evaluated our best-performing 4-layer MLP model using diagnostic visualizations and classification metrics.

Learning Curves. Figure 7 illustrates the training and validation performance over 20 epochs. The training loss steadily decreases, while the validation loss flattens after epoch 10. Accuracy continues to rise, and the gap between training and validation accuracy remains small, indicating stable convergence without significant overfitting.

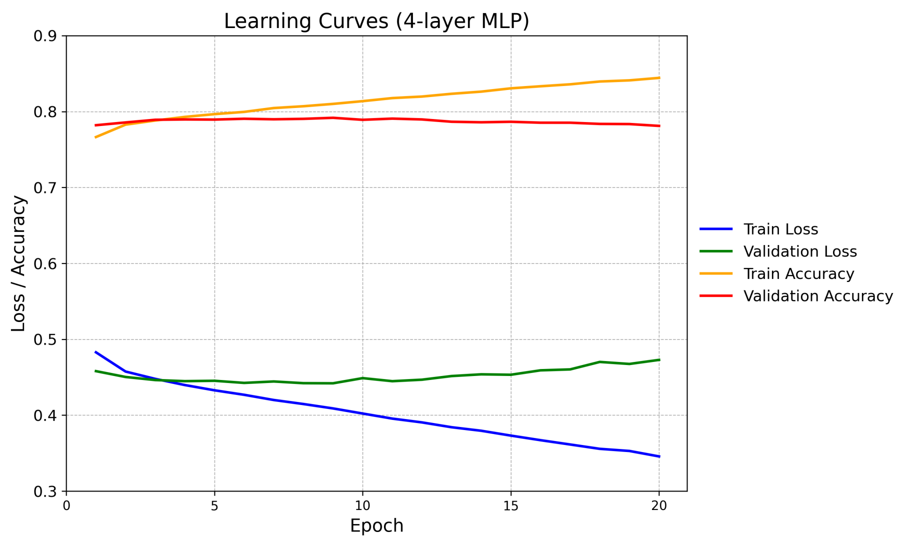

Figure 7: Learning curves (4-layer MLP). The model shows consistent training progress and generalizes well to the validation set.

Confusion Matrix. Figure 8 shows the confusion matrix, revealing balanced class performance. The model misclassifies slightly more class 1 instances than class 0, consistent with trends seen in shallower architectures.

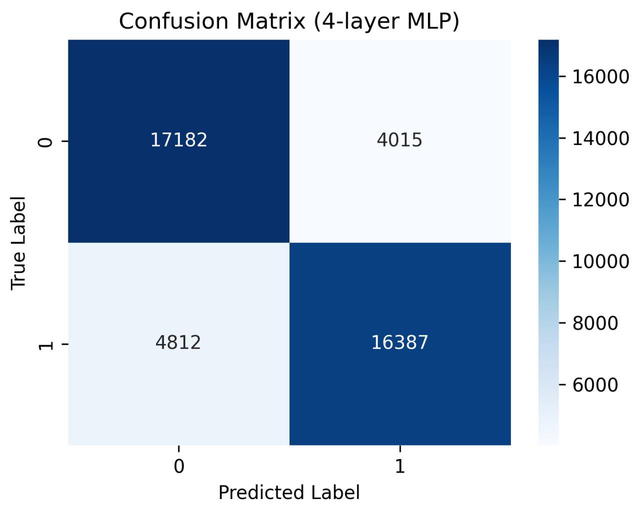

Figure 8: Confusion matrix (4-layer MLP). Misclassifications are slightly more frequent for class 1.

ROC Curve. The ROC curve in Figure 9 demonstrates strong classification capability, with an AUC of 0.8759—comparable to the 2- and 3-layer networks.

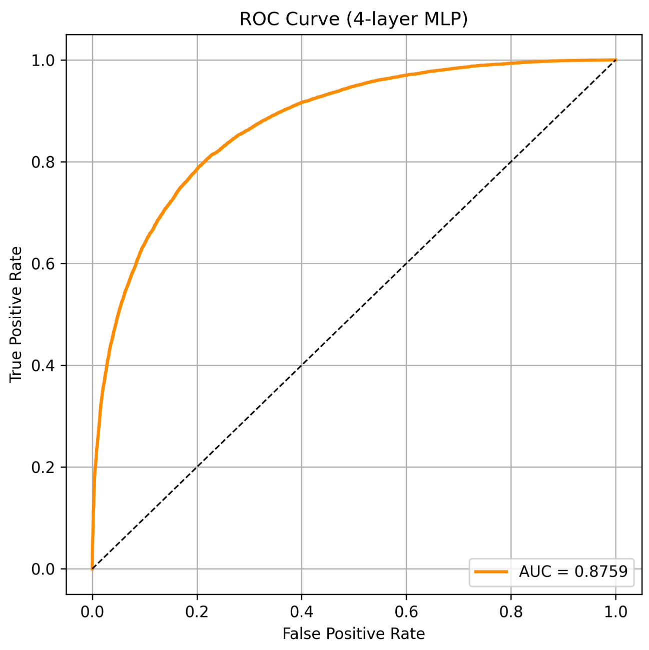

Figure 9: ROC Curve (4-layer MLP). The AUC of 0.8759 indicates high discriminative performance.

Classification Report. Table 6 presents precision, recall, and F1-score for each class. The 4-layer MLP achieves an overall accuracy of 79.18% and a macro-average F1 of 0.7917, with consistent class-wise performance.

Class	Precision	Recall	F1-score	Support
0	0.7812	0.8106	0.7956	21197
1	0.8032	0.7730	0.7878	21199
Accuracy		0.7918		
Macro Avg	0.7922	0.7918	0.7917	42396
Weighted Avg	0.7922	0.7918	0.7917	42396

Table 6: Classification Report (4-layer MLP). Precision and recall are balanced across classes, yielding strong overall performance.

3.6. Model Comparison and Analysis

To understand the trade-offs between model complexity and performance, we compared our best 2-layer, 3-layer, and 4-layer MLP configurations based on validation accuracy, F1 score, AUC, learning behavior, and confusion matrices.

Validation Performance. Table 7 summarizes the key evaluation metrics. The 2-layer MLP achieves the best F1-score, while the 4-layer MLP obtains the highest AUC. The 3-layer model strikes a strong balance with competitive scores across all metrics. All architectures perform similarly overall.

Model	Accuracy	F1 Score	AUC
2-layer MLP	0.7932	**0.7931**	**0.8770**
3-layer MLP	**0.7921**	0.7921	0.8751
4-layer MLP	0.7918	0.7917	0.8759

Table 7: Model comparison across depth.

Learning Behavior. The learning curves (Figures 1–7) indicate that all models converge smoothly. The deeper 4-layer model maintains lower training loss but sees marginal benefit in validation loss, suggesting diminishing returns in generalization beyond 3 layers.

ROC and Confusion Matrix. All models achieve an AUC over 0.875, highlighting their strong classification capability. The confusion matrices consistently show a slight tendency to misclassify class 1, though overall class balance is well-maintained.

4. Final Model Selection

After evaluating all candidate architectures, we selected the 2-layer MLP model as our final submission model. Despite being the simplest among the tested networks, it achieved the highest test accuracy on Kaggle.

Test Performance. The best 2-layer MLP configuration achieved a test accuracy of **79.494%** on the held-out Kaggle dataset, outperforming the deeper 3-layer and 4-layer architectures in this final evaluation.

- **Word2Vec:** vector_size=300, window=7, min_count=4
- **MLP:** H1=375, H2=374, dropout=0.2704, activation=Leaky ReLU
- **Training:** optimizer=AdamW, learning rate=1.0702×10^{-4}, weight decay=3.5708×10^{-6}, batch size=64

Justification. In addition to having the highest test accuracy, the 2-layer model shows strong generalization ability, minimal overfitting in the learning curves, and balanced class-wise performance. It also offers a lower computational cost compared to deeper models, making it suitable for deployment in real-world or resource-constrained settings.

Conclusion. While deeper networks like the 3- and 4-layer MLPs were competitive in validation F1 and AUC metrics, the 2-layer MLP emerged as the best-performing and most efficient model overall. Thus, it was selected for final submission.

5. Conclusion

In this report, we developed and evaluated several neural network architectures for sentiment classification on Twitter data, using Word2Vec embeddings and Multi-Layer

Perceptrons (MLPs) of varying depths. We applied extensive preprocessing, hyperparameter optimization via Optuna, and rigorous evaluation using accuracy, ROC curves, and confusion matrices. Overall, the project demonstrates that with thoughtful design and evaluation, relatively lightweight models can achieve competitive results on real-world NLP tasks.