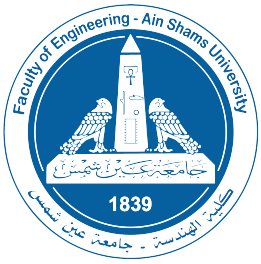

Ain Shams University - Faculty of Engineering
Computer and Systems Engineering

CSE485: Deep Learning

Supervised by: Dr Mahmoud Khalil, Eng Mahmoud Sohail

Student Name	**ID**
Farida Waleed Fakhry	21P0167
Monica Hany Makram Derias	21P0173
Yasmina Nasser Hamam	21P0211
Yomna Mohamed Hachem	21P0189

Contents

1 Problem Definition

The widespread use of social media has increased the presence of harmful content such as hate speech and abusive language, which can negatively impact individuals and society. Due to the large volume of user-generated text, manual content moderation is not feasible, making automated detection methods essential. Since social media data is primarily unstructured text, Natural Language Processing (NLP) techniques are well-suited for this task.
This project focuses on the task of **sentence classification**, a fundamental NLP application where each text input is assigned to a predefined category based on its semantic content. The goal is to automatically classify tweets into different categories according to the presence and type of harmful language.

2 Dataset Exploration

2.1 Dataset Import

The dataset is loaded into a Pandas DataFrame for further exploration and preprocessing.

```
file_id = '1tn2X9JtOjqG7PKTZLUEy8sJS0r_WV7Qf'
url = f'https://drive.google.com/uc?id={file_id}'
gdown.download(url, 'hate_speech.csv', quiet=False)

# Load dataset
df = pd.read_csv('hate_speech.csv')
print(f"Shape: {df.shape}")
print(df['class'].value_counts().sort_index())
df.head()
```

Figure 1: Dataset import and initial inspection

2.2 Basic Dataset information

```
Shape: (24783, 2)
class
0     1430
1    19190
2     4163
Name: count, dtype: int64
```

	class	tweet
0	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

Figure 2: Dataset exploration

The dataset is structured into two main columns: one column containing the textual content of each tweet, and another column representing the corresponding class label. This structure makes the dataset suitable for supervised text classification tasks.

Each tweet is annotated into one of the following three classes:

- **Class 0 – Hate Speech:** Tweets that explicitly target individuals or groups with harmful, discriminatory, or hateful intent.
- **Class 1 – Offensive Language:** Tweets that contain offensive or abusive language but do not necessarily express hate speech.
- **Class 2 – Neither:** Tweets that are neutral and do not contain offensive or hateful content.

This labeled dataset enables supervised learning approaches, where the model learns patterns and linguistic features associated with each category.

3 Dataset Visualization

3.1 Class Distribution

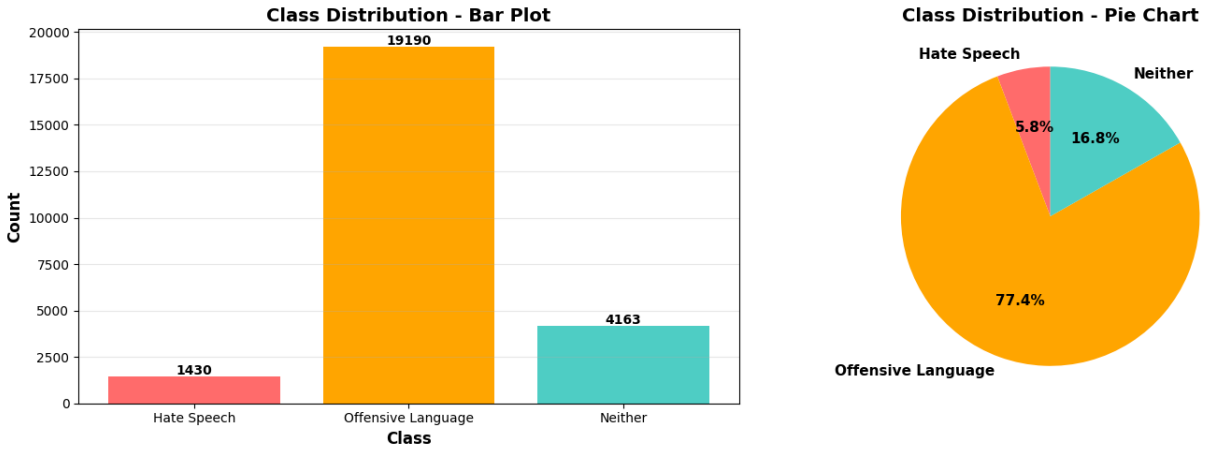

Figure 3: Class Dirstribution

An analysis of the class distribution reveals a severe class imbalance within the dataset. The *Offensive Language* class dominates the data, accounting for approximately 77.4% of the tweets (19,190 samples), followed by the *Neither* class with 16.8% (4,163 samples). In contrast, the *Hate Speech* class is significantly underrepresented, comprising only 5.8% of the dataset (1,430 samples). This results in an imbalance ratio of approximately 13.4:1 between the largest and smallest classes, which poses a challenge for model training, as it may bias the classifier toward the majority class if not properly addressed.

3.2 Word Frequencies by Class

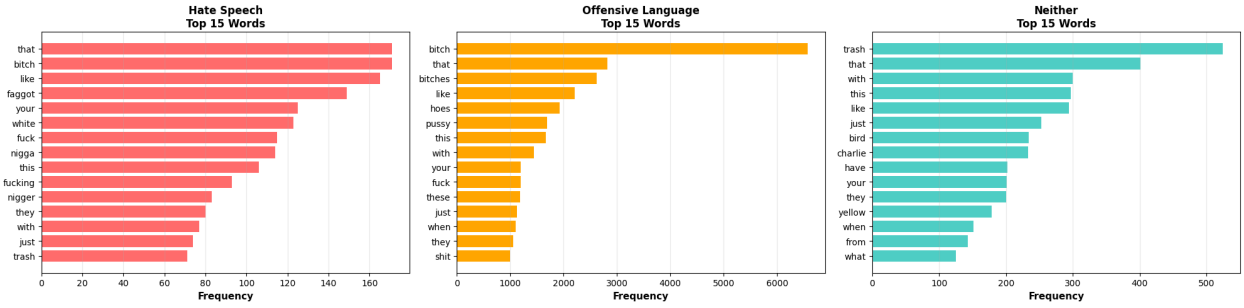

Figure 4: Word Frequencies by Class

To gain insights into the linguistic patterns associated with each class, word frequency analysis was performed separately for hate speech, offensive language, and neutral tweets. The most frequent terms were then extracted and visualized using horizontal bar charts, displaying the top fifteen words for each category. This visualization highlights distinctive vocabulary usage across classes,

revealing how certain words and expressions are more prevalent in offensive and hate-related content compared to neutral tweets.

3.3 Distribution of Tweet Length

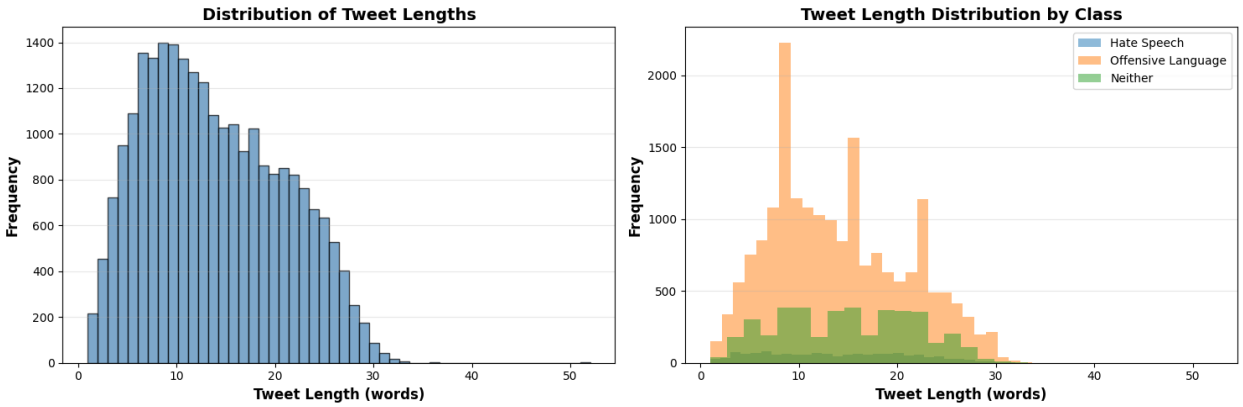

Figure 5: Distribution of Tweet Length

```
Average tweet length: 14.12 words
Median tweet length: 13.00 words
Max tweet length: 52 words
Min tweet length: 1 words
```

Figure 6: Tweet Length Analysis

An analysis of tweet lengths was performed to guide the selection of an appropriate sequence length for model input and padding. Tweet length was measured as the number of words per tweet, revealing an average length of 14.12 words and a median of 13 words, indicating that most tweets are relatively short. While the majority of tweets fall within this range, a small number of outliers reach a maximum length of 52 words, whereas some tweets contain as few as a single word. Based on this distribution, padding and truncation strategies can be effectively applied to ensure that all input sequences have a uniform length.

4 Dataset Preprocessing

4.1 Balancing the Dataset

```python
class_0 = df[df['class'] == 0]  # Hate Speech
class_1 = df[df['class'] == 1].sample(n=4500, random_state=42)  # Offensive Language
class_2 = df[df['class'] == 2]  # Neutral

df_balanced = pd.concat([class_0, class_0, class_0, class_1, class_2], axis=0)
```

Figure 7: Balancing the Dataset

Due to the severe class imbalance observed in the dataset, a data balancing strategy was applied prior to model training. The *Hate Speech* class, which was significantly underrepresented, was oversampled by replicating its instances multiple times to increase its presence in the training data. In contrast, the *Offensive Language* class, which dominated the dataset, was downsampled by randomly selecting a subset of samples to reduce its influence. The *Neither* class was retained without modification. This combination of oversampling and undersampling resulted in a more balanced class distribution, helping to mitigate model bias toward the majority class and improving the model's ability to learn meaningful patterns across all categories.

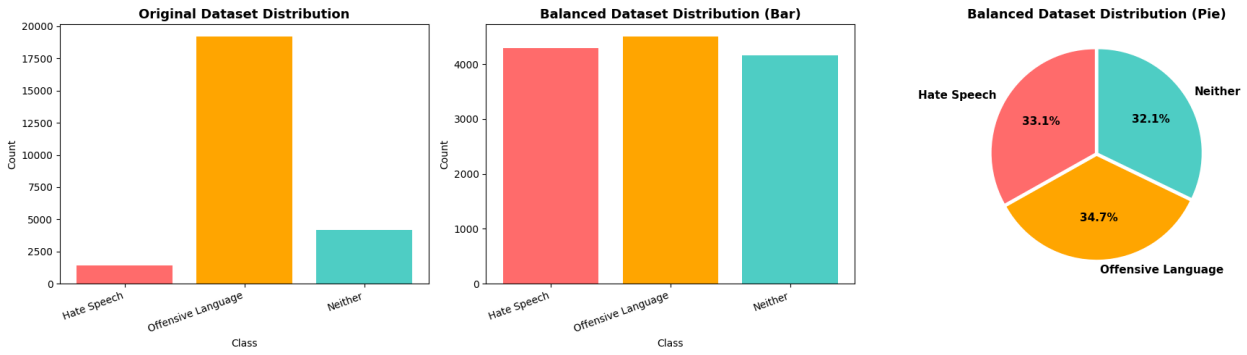

Figure 8: Balanced Dataset

4.2 Text Preprocessing

```python
def preprocess(data):
    clean = []
    # remove the @xxx"
    clean = [remove_user(text) for text in data]
    # remove emojis (specifically unicode emojis)
    clean = [remove_entity(text) for text in clean]
    # remove urls
    clean = [remove_url(text) for text in clean]
    # remove trailing stuff
    clean = [remove_noise_symbols(text) for text in clean]
    # remove stopwords
    clean = [remove_stopwords(text) for text in clean]

    clean = [lemmatization(text) for text in clean]

    return clean

# call the cleaning function
processed_tweet = preprocess(tweet)
```

Figure 9: text preprocessing

Prior to model training, a text preprocessing pipeline was applied to clean the tweet data. Each preprocessing step was designed to reduce noise and improve the quality of textual features provided to the deep learning model. The preprocessing operations are summarized as follows:

- **Removal of User Mentions:** User tags (e.g., `@username`) were removed from the tweets, as they do not contribute to the semantic meaning of the content and may introduce unnecessary noise into the model.
- **Removal of HTML Entities:** HTML-encoded characters and entities commonly present in social media text were eliminated to ensure cleaner and more readable input text.
- **URL Removal:** Hyperlinks were removed from the tweets since they typically do not carry useful information for hate speech classification and can distort word distributions.

- **Noise and Symbol Cleaning:** Unnecessary punctuation marks and special characters were removed to reduce textual clutter and improve token consistency.
- **Stopword Removal:** Common stopwords, along with retweet indicators such as ‘‘rt’’, were excluded to focus the model on more informative and discriminative words.
- **Lemmatization:** Words were reduced to their base or dictionary forms using lemmatization, allowing the model to better generalize across different grammatical variations of the same word.

All preprocessing steps were combined into a single pipeline function to ensure consistent and efficient cleaning of the dataset before tokenization, padding, and model training.

4.3 Tokenization and Padding

```
features = preprocess(df_balanced['tweet'].tolist())
target = df_balanced['class'].tolist()
target =labels
X_train, X_val, Y_train, Y_val = train_test_split(features, target, test_size=0.2, random_state=42)

# One-hot encode the labels
Y_train = pd.get_dummies(Y_train)
Y_val = pd.get_dummies(Y_val)

# Tokenization
max_words = 5000
max_len = 40
tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
tokenizer.fit_on_texts(X_train)

# Convert text to sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_val_seq = tokenizer.texts_to_sequences(X_val)

# Pad sequences
X_train_padded = pad_sequences(X_train_seq, maxlen=max_len, padding='post', truncating='post')
X_val_padded = pad_sequences(X_val_seq, maxlen=max_len, padding='post', truncating='post')
```

Figure 10: Tokenization and Padding

After preprocessing, the cleaned tweets were transformed into numerical representations suitable for input into a deep learning model. This process involved the following steps

- **Feature and Label Extraction:** The preprocessed tweet texts were extracted as input features, while the corresponding class labels were extracted as target values. These labels represent the three classification categories: hate speech, offensive language, and neutral content.

- **Train–Validation Split:** The dataset was divided into training and validation sets using an 80:20 split. This separation allows the model to be trained on one subset of the data while being evaluated on unseen data, enabling a reliable assessment of generalization performance.
- **One-Hot Encoding of Labels:** Since the task is a multi-class classification problem, class labels were converted into one-hot encoded vectors. This representation is required when using categorical loss functions and allows the model to output probability distributions over the three classes.
- **Tokenization:** A tokenizer was fitted on the training data to build a vocabulary of the most frequent words. Each word was mapped to a unique integer index, limiting the vocabulary size to the top 5,000 most frequent tokens to reduce noise and computational complexity.
- **Text-to-Sequence Conversion:** The tokenized text was transformed into sequences of integers, where each sequence represents a tweet as an ordered list of word indices.
- **Padding and Truncation:** To ensure uniform input length across all samples, sequences were padded or truncated to a fixed maximum length of 40 tokens. Shorter tweets were padded at the end, while longer tweets were truncated, preserving the most relevant information within a consistent input shape for the model.

5 Hyperparameter Tuning and Experimental Evaluation

This section presents a systematic study of hyperparameter tuning conducted on the proposed LSTM-based hate speech classification model. The core architecture was kept fixed throughout all experiments, and only selected hyperparameters were modified in a controlled manner to study their impact on validation performance.

5.1 Experimental Setup

All experiments were conducted using the same preprocessing pipeline, tokenizer configuration, and train–validation split. Early stopping based on validation loss was applied in all trials to prevent overfitting. Performance was evaluated using validation accuracy as well as class-wise precision, recall, and F1-score, with particular attention to the hate speech class due to class imbalance.

5.2 Trial Configurations

Multiple trials were conducted by manually adjusting hyperparameters such as embedding dimension, number of LSTM units, bidirectionality, dense layer size, regularization strength, dropout rate, optimizer choice, and batch size.

Table 1 summarizes the main configurations and results.

Table 1: Summary of Hyperparameter Tuning Trials

Trial	Vocab	Emb	LSTM	BiLSTM	Dense	Reg	Drop	Opt	Batch	Val Acc
T1	10k	32	32	No	512	L1	0.3	SGD	32	0.35
T2	10k	32	32	No	512	L1	0.3	Adam	32	0.88
T3	10k	64	64	No	125	L1	0.2	Adam	16	0.89
T4	5k	64	64	Yes	125	L1	0.2	Adam	8	0.89

5.3 Observations and Analysis

The first trial (T1), which used stochastic gradient descent with a low learning rate, failed to converge and resulted in class collapse, where the model predicted only the majority class. This behavior highlights the sensitivity of recurrent neural networks to optimizer choice, particularly in text classification tasks with imbalanced data.

Switching to the Adam optimizer in Trial T2 significantly stabilized training and led to a substantial improvement in validation accuracy and minority-class recall. This configuration served as a strong baseline.

Trials T3 and T4 explored higher model capacity through increased embedding dimensionality and LSTM units. While both achieved similar validation accuracy, the bidirectional LSTM in Trial T4 demonstrated improved recall for the hate speech class, indicating better contextual representation at the cost of increased computational complexity.

5.4 Classification Performance

Next figures present the classification report of most of configuration.

Figure 11: Classfication Report-1

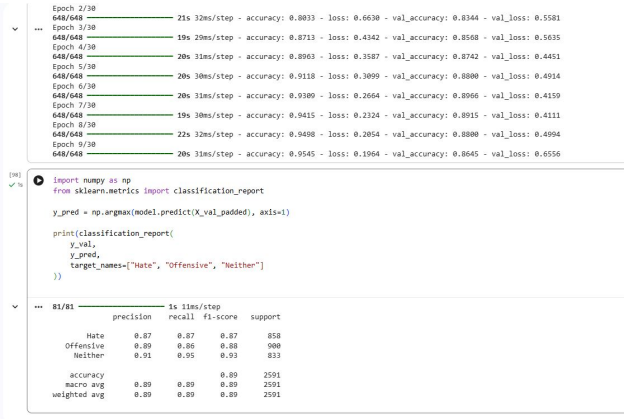

Figure 12: Classification Report-2

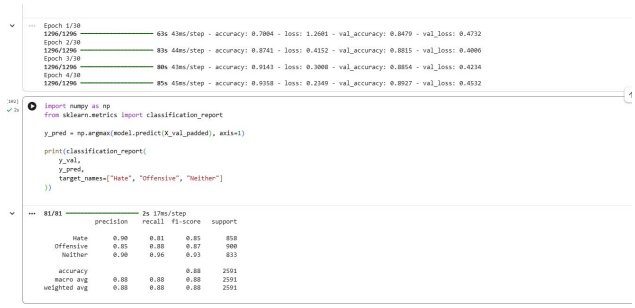

Figure 13: Classification Report-3

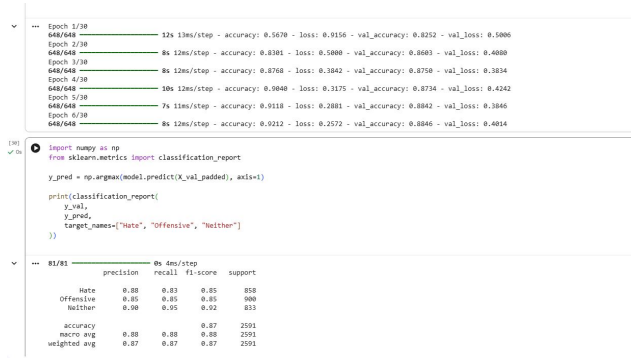

Figure 14: Classification Report-4

5.5 Discussion

Overall, the experiments demonstrate that optimizer selection and model capacity have a significant impact on performance. While larger models achieved marginally higher accuracy, the final configuration was selected based on a balance between validation accuracy, minority-class recall, and training stability rather than peak accuracy alone.

6 Final Model Architectures and Evaluation

After completing extensive hyperparameter tuning and architectural experiments, two final Bidirectional LSTM-based models were selected for detailed evaluation. Both models share a common preprocessing and training pipeline but differ in their capacity and regularization strategies. This section documents their architectures, training behavior, and evaluation results.

6.1 Final Model 1: Baseline Bidirectional LSTM

6.1.1 Architecture Description

The first model represents a high-capacity baseline architecture designed to capture rich contextual information using a Bidirectional LSTM followed by a large fully connected layer. Batch normalization and dropout were applied to stabilize training and reduce overfitting.

- Vocabulary size: 5,000
- Maximum sequence length: 30
- Embedding dimension: 32
- Bidirectional LSTM with 16 units
- Dense layer with 512 units and ReLU activation
- L1 regularization on dense layer
- Batch Normalization
- Dropout rate: 0.3
- Output layer: Softmax (3 classes)
- Optimizer: Adam
- Loss function: Categorical Cross-Entropy

```
max_words =5000
max_len = 30

model = keras.models.Sequential([
    layers.Embedding(input_dim=max_words, output_dim=32, input_length=max_len),
    layers.Bidirectional(layers.LSTM(16)),
    layers.Dense(512, activation='relu', kernel_regularizer='l1'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(3, activation='softmax')
])

model.build(input_shape=(None, max_len))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

Figure 15: Implementation of the baseline Bidirectional LSTM model

6.1.2 Training Behavior

The model was trained using early stopping based on validation accuracy and adaptive learning rate reduction based on validation loss. Despite its relatively large dense layer, the model demonstrated stable convergence when trained with the Adam optimizer.

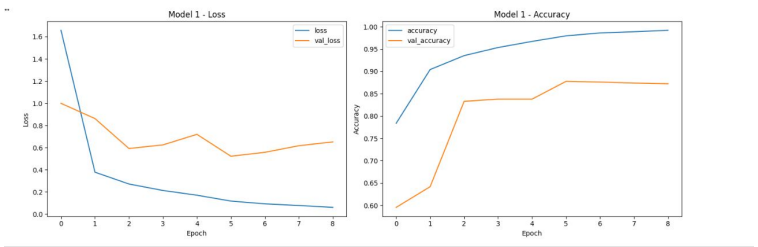

Figure 16: Training and validation loss and accuracy for Model 1

6.1.3 Evaluation Results

Model performance was evaluated on the validation set using accuracy, class-wise precision, recall, F1-score, and a confusion matrix.

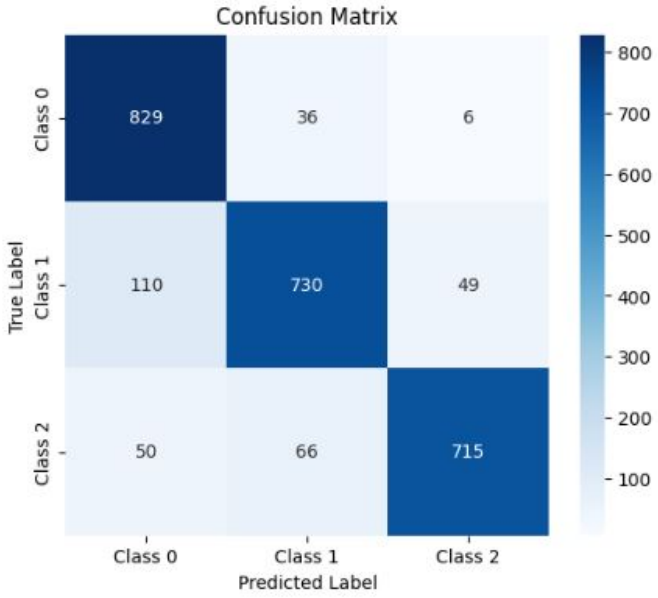

Figure 17: Confusion matrix for Model 1

	precision	recall	f1-score	support
Class 0	0.90	0.90	0.90	858
Class 1	0.81	0.88	0.84	900
Class 2	0.93	0.85	0.89	833
accuracy			0.88	2591
macro avg	0.88	0.88	0.88	2591
weighted avg	0.88	0.88	0.88	2591

Figure 18: Classification report for Model 1

6.2 Final Model 2: Regularized Bidirectional LSTM

6.2.1 Architecture Description

The second model adopts a more compact architecture with stronger regularization mechanisms to improve generalization. Spatial dropout was applied at the embedding level, while L2 regularization was introduced in both the kernel and recurrent connections of the LSTM layer. Additionally, combined L1–L2 regularization was applied to the dense layer.

- Vocabulary size: 5,000
- Maximum sequence length: 30

- Embedding dimension: 32
- Spatial Dropout: 0.3
- Bidirectional LSTM with 8 units
- LSTM kernel and recurrent L2 regularization (1×10^{-4})
- Dense layer with 32 units and ReLU activation
- Dense layer L1–L2 regularization ($L1 = 1 \times 10^{-4}$, $L2 = 1 \times 10^{-3}$)
- Batch Normalization
- Dropout rate: 0.5
- Output layer: Softmax (3 classes)
- Optimizer: Adam
- Loss function: Categorical Cross-Entropy

```python
from tensorflow.keras import regularizers

model_2 = keras.Sequential([
    layers.Embedding(max_words, 32, input_length=max_len),
    layers.SpatialDropout1D(0.3),

    layers.Bidirectional(
        layers.LSTM(
            8,
            kernel_regularizer=regularizers.l2(1e-4),
            recurrent_regularizer=regularizers.l2(1e-4),
            dropout=0.3,
            recurrent_dropout=0.0
        )
    ),

    layers.Dense(32, activation='relu',
                 kernel_regularizer=regularizers.l1_l2(l1=1e-4, l2=1e-3)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(3, activation='softmax')
])
model_2.build(input_shape=(None, max_len))

model_2.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model_2.summary()
```

Figure 19: Implementation of the regularized Bidirectional LSTM model

6.2.2 Training Behavior

Due to the aggressive regularization strategy, Model 2 exhibited slower convergence during early epochs but achieved more stable validation performance. Learning rate reduction contributed to smoother convergence and reduced fluctuations in validation loss.

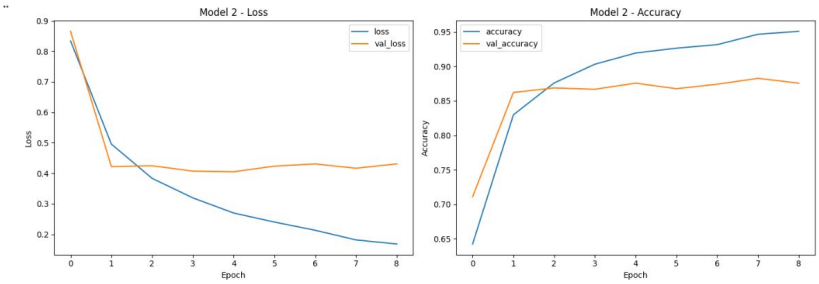

Figure 20: Training and validation curves for Model 2

6.2.3 Evaluation Results

The second model demonstrated improved generalization, particularly for minority classes, as reflected in its confusion matrix and classification metrics.

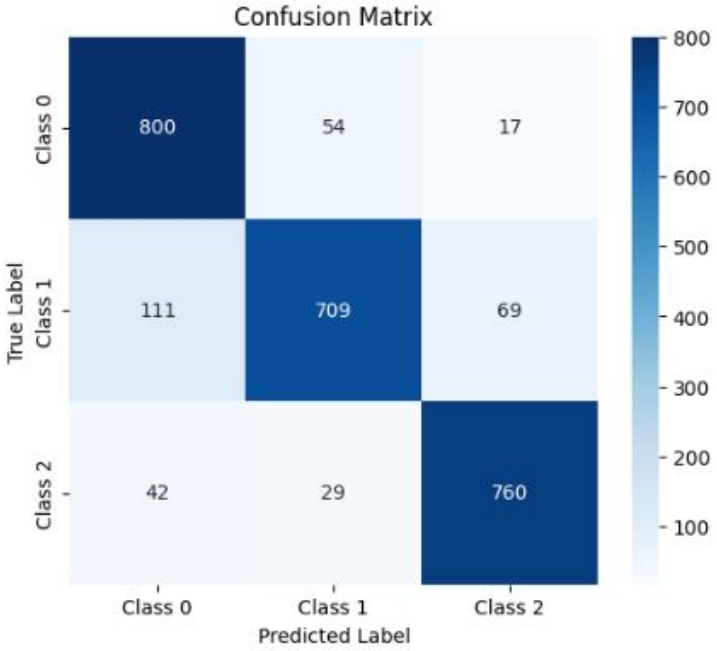

Figure 21: Confusion matrix for Model 2

```
81/81 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
              precision    recall  f1-score   support

        Hate       0.88      0.83      0.85       858
   Offensive       0.85      0.85      0.85       900
     Neither       0.90      0.95      0.92       833

    accuracy                           0.87      2591
   macro avg       0.88      0.88      0.88      2591
weighted avg       0.87      0.87      0.87      2591
```

Figure 22: Classification report for Model 2

6.3 Comparative Discussion

A comparison of the two models reveals that while Model 1 benefited from higher capacity and achieved strong performance, it exhibited a greater tendency toward overfitting. In contrast, Model 2 achieved more balanced and robust performance through extensive regularization, despite having fewer trainable parameters.

As a result, Model 2 is more suitable for deployment scenarios where generalization and robustness are prioritized over raw training accuracy.

6.4 Summary

Two final Bidirectional LSTM models were evaluated in this study. The baseline model demonstrated the effectiveness of high-capacity architectures for text classification, while the regularized model highlighted the importance of controlling overfitting in imbalanced datasets. The systematic comparison enabled an informed selection of the final architecture based on empirical evidence rather than isolated performance metrics.