Deep Learning Based Text Classification For Hate Speech In Online Social Networks

A Project Report Submitted In Partial Fulfillment Of The

Requirements For The Award Of The Degree Of

BACHELOR OF TECHNOLOGY

In

COMPUTER ENGINEERING

Under the supervision of

Dr. FAIYAZ AHMAD

(Assistant Professor)

Submitted By:

FAIZAN AHAMAD(21BCS047)

MANTASHA FIRDOUS(21BCS049)

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

JAMIA MILLIA ISLAMIA, NEW DELHI - 110025

2021-25

DECLARATION

We declare that this project titled "**Deep Learning Based Text Classification For Hate Speech In Online Social Networks**" submitted in partial fulfillment of the degree of **Bachelor of Technology in Computer Engineering** is a record of original work carried out by us under the supervision of **Dr. Faiyaz Ahmad**, and has not formed the basis for the award of any other degree, in this or any other Institution or University. In keeping with the ethical practice of reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

——---------——---------- ----------——----------

Faizan Ahamad(21BCS047) Mantasha Firdous(21BCS049)

faizu78601@gmail.com firdousmantasha211@gmail.com

New Delhi-110025

Date:

जामिया मिल्लिया इस्लामिया
(संसदीय अधिनियमानुसार केन्द्रीय विश्वविद्यालय)
मौलाना मोहम्मद अली जौहर मार्ग, नई दिल्ली-110025
JAMIA MILLIA ISLAMIA
NAAC Accredited Grade 'A++' (A Central University by an Act of Parliament)
Maulana Mohammed Ali Jauhar Marg, New Delhi-110025

दूरभाष
Tel. : 26980281, 26985831
विस्तार
Extn. : 2580
EPABX : 26981717
Extn. : 2442
E-mail : computerengg@jmi.ac.in
Web. : http://jmi.ac.in

कंप्यूटर अभियांत्रिकी विभाग
अभियांत्रिकी एवं प्रौद्योगिकी संकाय
Department of Computer Engineering
Faculty of Engineering and Technology
شعبہ کمپیوٹر انجینئرنگ
فیکلٹی آف انجینئرنگ اینڈ ٹکنالوجی

جامعہ ملیہ اسلامیہ

CERTIFICATE

This is to certify that the project report entitled "**Deep Learning Based Text Classification For Hate Speech In Online Social Networks**" submitted by **Faizan Ahamad (21BCS047)** and **Mantasha Firdous (21BCS049)** to the Department of Computer Engineering, F/O Engineering and Technology, Jamia Millia Islamia, New Delhi - 110025 in partial fulfilment for the award of the degree of Bachelor of Technology in Computer Engineering is a bona fide record of project work carried out by them under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.

Dr. Faiyaz Ahmad
Supervisor
Department of Computer Engineering
Jamia Millia Islamia, New Delhi

Prof. Dr. Mohammad Amjad
Professor & Head
Department of Computer Engineering
Jamia Millia Islamia, New Delhi

ACKNOWLEDGEMENTS

We wish to express our sincere thanks and gratitude to our project supervisor **Dr. Faiyaz Ahmad Sir**, who has been immensely supportive and guided us throughout the project to bring it to completion, the Respected Head of the Department, **Dr. Mohammad Amjad Sir**, and all respected Professors of the **Computer Engineering** Department from F/O Engineering & Technology, Jamia Millia Islamia, for enabling us to work on such a project. Their constant encouragement has proven to be very precious in attaining the goals of this project. We are thankful to each one of them for all the formal and informal discussions, constructive criticism and invaluable suggestions which were a steering factor in our continuous improvement of the project. A special thanks to our parents and loved ones, we are whatever they strived for, and they've never known what we've become because of them. Finally, a great appreciation goes to the friends and colleagues who gave us the required confidence and support in boosting our morale and providing much-needed motivation.

Faizan Ahamad (21BCS047)

Mantasha Firdous(21BCS049)

ABSTRACT

Hate speech on social media threatens online safety, societal harmony, and mental well-being, especially on platforms like Twitter. This study introduces a deep learning-based pipeline for multi-class hate speech detection using over 25,000 annotated tweets. The proposed framework integrates various neural architectures, including Convolutional Neural Networks (CNN), Bidirectional Long Short-Term Memory (Bi-LSTM), and transformer-based models like BERT, ALBERT, and ELECTRA. To address data imbalance and improve feature diversity, advanced preprocessing and data augmentation techniques such as synonym replacement, random insertion, and TF-IDF encoding were applied.

Among the evaluated models, BERT combined with a Multi-Layer Perceptron (MLP) achieved superior performance, recording a macro F1 score of 0.91 and a weighted F1 score of 0.95, showcasing high accuracy and class balance. The study also found that GloVe embeddings and TF-IDF features have distinct advantages across model types, underscoring the critical role of feature representation in text classification.

This research demonstrates the effectiveness of combining deep learning and transfer learning for hate speech detection and offers a reproducible pipeline that can be adapted for real-world deployment. The study contributes valuable insights into model selection and embedding strategies for building robust, scalable, and accurate content moderation systems.

Keywords—Hate speech detection, deep learning, transformer models, BERT, text classification, natural language processing, Twitter, TF-IDF, Glove embeddings, data augmentation, multi-class classification.

TABLE OF CONTENTS

.

.

.

.

.

.

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

Abbreviation	Full Form	Page Number
CNN	Convolutional Neural Network	6
Bi-LSTM	Bidirectional Long Short-Term Memory	13
BERT	Bidirectional Encoder Representations from Transformers	4, 27
ALBERT	A Lite BERT	4
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately	4
TF-IDF	Term Frequency–Inverse Document Frequency	4
MLP	Multi-Layer Perceptron	23
GloVe	Global Vectors for Word Representation	27
SVM	Support Vector Machine	16
XGBoost	Extreme Gradient Boosting	19
NLP	Natural Language Processing	1
SGD	Stochastic Gradient Descent	10
ReLU	Rectified Linear Unit	7
RNN	Recurrent Neural Network	13
BPTT	Backpropagation Through Time	14
AUC-ROC	Area Under Curve - Receiver Operating Characteristic	28
BoW	Bag of Words	4
OOV	Out-of-Vocabulary	27
CBOW	Continuous Bag of Words	4

1. INTRODUCTION

1.1 Hate Speech in Online Social Networks

The advent of online social networks has significantly altered the way individuals interact, communicate, and access information. Platforms such as Twitter, Facebook, Instagram, and Reddit have enabled instantaneous global connectivity, fostering rich exchanges of ideas, opinions, and cultural perspectives. While these platforms offer numerous social and economic benefits, they have also become fertile grounds for the proliferation of harmful content, including misinformation, cyberbullying, and notably, hate speech.

Hate speech is broadly defined as any form of communication that belittles, insults, threatens, or incites violence against individuals or groups based on inherent characteristics such as race, ethnicity, nationality, religion, gender identity, sexual orientation, or disability. The presence of such content online not only undermines respectful discourse but also has real-world consequences, including psychological harm, increased polarization, and in extreme cases, the incitement of violence or hate crimes. The anonymity and virality enabled by social networks often exacerbate the spread of hate speech, as users can disseminate harmful rhetoric quickly and with limited accountability.

Various studies and reports have underscored the rising prevalence of hate speech on social media. Automated bots, organized troll groups, and algorithm-driven content amplification mechanisms further contribute to its visibility and impact. This has led to increasing public concern, policy discussions, and calls for regulatory interventions. However, platform-specific policies and enforcement practices have proven inconsistent, and manual moderation efforts are insufficient to handle the vast volume of content generated daily.

Compounding the issue is the evolving and context-dependent nature of hate speech. It can be explicit—such as racial slurs or calls to violence—or implicit, using coded language, sarcasm, or cultural references to evade detection. Additionally, the informal language, abbreviations, emojis, and memes common in social media communication pose unique challenges to traditional content moderation tools. As a result, detecting hate speech in such dynamic environments requires sophisticated systems capable of understanding both linguistic and contextual subtleties.

Given these challenges, there has been a growing emphasis on developing automated hate speech detection models that can operate at scale with high accuracy. This involves leveraging techniques from the fields of machine learning and natural language processing (NLP). While early approaches relied on handcrafted features and simple classifiers, recent advancements in deep learning and pre-trained language models have opened new avenues for building more effective detection systems. Nevertheless, the design and implementation of such systems remain a complex task, necessitating ongoing research and innovation to address both technical and ethical considerations.

1.2 Motivation

Despite the substantial efforts invested in content moderation by social media companies, the detection and prevention of hate speech remain a persistent and evolving challenge. Traditional moderation methods, such as manual review and rule-based filters, are increasingly proving to be ineffective at scale. Manual moderation, while often accurate, is labor-intensive, expensive, and emotionally taxing for human reviewers. Rule-based systems, on the other hand, struggle to capture the nuanced and ever-changing language used in hate speech, especially when it is disguised through irony, sarcasm, or coded terms.

Early machine learning methods—such as Support Vector Machines (SVM), Logistic Regression, and Naïve Bayes classifiers—provided incremental improvements by automating the classification process. These models typically relied on basic textual features like word frequency or n-grams. However, they often fell short in understanding context, especially in the informal and unstructured linguistic environment of social media. Furthermore, these methods require extensive feature engineering and struggle with generalization across different platforms or topics.

The emergence of deep learning has significantly advanced the field by enabling models to learn complex patterns from large datasets. Architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks, have demonstrated promising results in text classification tasks. More recently, transformer-based models like BERT (Bidirectional Encoder Representations from Transformers), ALBERT, and ELECTRA have redefined performance benchmarks across a variety of NLP tasks due to their ability to capture bidirectional context and semantic depth. However, these models are not without limitations, especially when dealing with imbalanced datasets and noisy input typical of hate speech detection tasks.

Another underexplored aspect is data augmentation. While widely used in image recognition to artificially expand datasets and improve model robustness, augmentation in text-based tasks presents unique challenges. Generating semantically coherent variations of sentences without altering their meaning or intent is complex but potentially beneficial, especially for improving performance on minority classes like hate speech, which are often underrepresented in datasets.

1.3 Objective

This study aims to address the aforementioned challenges by developing a comprehensive and modular pipeline for hate speech detection in online social networks. The overarching goal is to create an automated system that is both effective and scalable, capable of identifying hate speech in real-time across diverse linguistic contexts. To achieve this, the following specific objectives are outlined:

1. Pipeline Development: To design and implement a modular framework that integrates all essential stages of text classification—data preprocessing, augmentation, feature representation, model selection, and evaluation. This ensures reproducibility, flexibility, and adaptability for future extensions.

2. Model Benchmarking: To evaluate and compare the performance of various deep learning architectures—namely CNN, Multi-Layer Perceptron (MLP), Bidirectional LSTM (Bi-LSTM), and Transformer-based models—using multiple types of word embeddings, including traditional techniques like TF-IDF and GloVe, as well as contextual embeddings from BERT-family models.

3. Data Augmentation Exploration: To investigate the role of textual data augmentation methods such as synonym replacement and random insertion in enhancing classification performance, particularly for minority classes.

4. Performance Validation: To validate the proposed models on a publicly available multi-class Twitter dataset, categorizing content as hate speech, offensive language, or neutral, and to benchmark performance using key metrics such as macro and weighted F1 scores.

Through these contributions, the study seeks to advance the state of hate speech detection research and provide practical tools that can be integrated into real-world content moderation systems. Ultimately, this work aims to support the creation of safer and more inclusive digital spaces.

2. RELATED WORKS

There is no universally accepted definition of hate speech (HS); the term is context-dependent and often overlaps with offensive or abusive language [1]. Some researchers refer to any form of targeted, hurtful, or inciting statement as hate speech [2], while others group it under broader categories of toxicity or online abuse [3], [4]. Despite these variations, the task of automated hate speech detection has become increasingly important due to the sheer scale of content generated on platforms like Twitter, Facebook, and Reddit.

Broadly, research efforts in this domain can be categorized into two methodological groups: (i) traditional machine learning-based approaches and (ii) deep learning and transfer learning-based models. The following subsections summarize influential studies in each approach.

2.1. Hate Speech Detection with Machine Learning Approaches

Warner and Hirschberg [17] were among the early contributors in this field, applying syntactic and lexical features on a dataset from Yahoo! and the American Jewish Congress. They used the SVMlight classifier and achieved an F1-score of 0.64 with 95% accuracy in their best-case evaluation.

Kwok and Wang [18] used a Bag-of-Words (BoW) feature set with a Naïve Bayes classifier for detecting racism in tweets. Their best model achieved 76% accuracy using unigram features in a 10-fold cross-validation setting. They noted that adding sentiment polarity and bigram features could further improve performance.

Burnap and Williams [19] extracted n-gram features (1–5) from 450,000 tweets and compared Bayesian Logistic Regression, SVM, and a Voted Ensemble Classifier. The ensemble method outperformed others, achieving 0.89 accuracy and an F1-score of 0.77.

Waseem and Hovy [4] introduced a 16K manually annotated tweet dataset labeled for HS and offensive content. They tested logistic regression using uni-gram to quad-gram features and obtained F1-scores above 73% using various metadata such as gender and location.

Davidson et al. [9] proposed a multi-class classification framework for labeling tweets as hate speech, offensive, or neutral. They used TF-IDF-weighted n-grams and logistic regression with L1 and L2 regularization. While their model achieved an overall F1-score of 0.90, the misclassification rate for hate speech remained around 40%, primarily due to class imbalance.

Gao and Huang [20] proposed a context-aware classification framework and compared logistic regression and LSTM models. Both models outperformed a character-level baseline by 3–4%, highlighting the value of contextual features.

2.2. Hate Speech Detection with Deep Learning and Transfer Learning Approaches

To address the limitations of shallow models and manual feature engineering, deep learning models have been adopted extensively. Djuric et al. [21] used paragraph2vec and continuous bag-of-words (CBOW) to encode user comments into low-dimensional vectors, followed by binary classification. They achieved an AUC of 0.80 with paragraph2vec.

Park and Fung [22] integrated CNN and logistic regression models, demonstrating that hybrid models can outperform standalone classifiers in detecting abusive language.

Zhang et al. [23] developed a hybrid architecture combining CNN and GRU (Gated Recurrent Unit) networks. Evaluated on seven public datasets, their model achieved superior performance in six cases, improving average F1-scores by 1–14%.

Kamble and Joshi [24] worked on English-Hindi code-mixed tweets and trained custom embeddings on large multilingual corpora. Among several tested models—SVM, Random Forest, CNN-1D, Bi-LSTM—the CNN-1D achieved the best performance with an F1-score of 80.85%.

Wei et al. [15] designed a pipeline leveraging BERT, DistilBERT, and GPT-2 for transfer learning. They demonstrated that transformer-based models not only increased classification accuracy but also performed better when augmented with preprocessed and augmented textual inputs using techniques such as synonym replacement and random insertion.

Malik et al. [1] conducted a large-scale comparative study evaluating 14 traditional and deep learning models on multiple datasets. Their experiments highlighted the superior performance of BERT, ALBERT, and ELECTRA in terms of macro and weighted F1 scores across different data domains and languages.

In summary, traditional machine learning models offer interpretability and ease of deployment but often fall short in capturing deep semantic and contextual meaning, especially in short, noisy texts like tweets. Deep learning and transformer-based models, though computationally intensive, have set new performance benchmarks, particularly when combined with data augmentation techniques and domain-specific embeddings. This study builds on such prior work by integrating both classical and modern techniques into a unified pipeline that addresses issues of class imbalance, semantic understanding, and real-time applicability in hate speech detection.

3. THEORETICAL BACKGROUND

3.1 Convolutional Neural Networks

3.1.1 Introduction

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed to process data with a grid-like topology, such as images and sequences. Although originally developed for computer vision tasks, CNNs have proven effective in various natural language processing (NLP) applications, including text classification, sentiment analysis, and hate speech detection. The architecture of CNNs is inspired by the biological processes of the visual cortex in animals, where individual neurons respond to stimuli only in a restricted region of the visual field. This allows CNNs to learn hierarchical spatial features, enabling them to identify simple patterns in early layers (e.g., edges) and complex structures in deeper layers (e.g., objects or phrases in text).

CNNs operate through a series of transformations applied to the input data using layers such as convolutional, activation, pooling, and fully connected layers. Each layer progressively extracts more abstract features from the input, culminating in a final prediction layer suitable for classification tasks.

3.1.2 Basic Structure of a CNN

A typical CNN is composed of the following types of layers:

- **Convolutional Layer**: Responsible for detecting local patterns via learnable filters.
- **Activation Layer**: Applies a non-linear function to introduce learning capability.
- **Pooling Layer**: Reduces spatial dimensions, controlling overfitting and computational load.
- **Fully Connected Layer**: Integrates learned features for classification or regression.
- **Dropout/Regularization Layers**: Prevents overfitting and improves generalization.

3.1.2.1 Convolutional Layers

The convolutional layer is the core building block of a CNN. It applies a mathematical operation called convolution, where a set of learnable kernels (also known as filters) are slid across the input to compute feature maps.

- Filter/Kernel: A small matrix (e.g., 3×3, 5×5) that detects specific features such as edges or textures.
- Stride: Determines how much the filter moves during convolution. A stride of 1 preserves more information, while larger strides downsample the feature map.
- Padding: Adds zero-padding around the input image to preserve spatial dimensions during convolution.

i. Mathematical Formulation:

Given an input image *I* of size *W*×*H* and a filter *F* of size *f*×*f*, the convolution operation is defined as:

$$(I \times F)(x,y) = \sum_{i=0}^{f-1}\sum_{j=0}^{f-1} I(x+i, y+j)\cdot F(i,j) \qquad \ldots (eq.\ 3.1)$$

This results in a **feature map** that captures the presence and strength of features across the image.

ii. Feature Maps

The result of applying multiple filters to an input image is a set of feature maps, each capturing different aspects of the image such as edges, textures, or complex shapes.

3.1.2.2. Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn and model complex patterns. Without these functions, the network would only be able to model linear relationships. Common Activation functions are:

i. ReLU (Rectified Linear Unit):

$$ReLU = max(0,x) \qquad \ldots (eq.\ 3.2)$$

It is the most commonly used activation function in CNNs due to its simplicity and effectiveness in mitigating the vanishing gradient problem.

ii. Sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \ldots (eq.\ 3.3)$$

It squashes input values to a range between 0 and 1 but can suffer from vanishing gradients.

iii. Tanh:

$$tanh = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \ldots (eq.\ 3.4)$$

It outputs values between -1 and 1, providing stronger gradients than the sigmoid function but still prone to vanishing gradients for very large or small inputs.

3.1.2.3. Pooling Layers

Pooling layers reduce the spatial dimensions of the feature maps, helping to decrease the computational load and reduce overfitting. The types of pooling are discussed below:

i. Max Pooling:
Takes the maximum value in each patch of the feature map. For instance, a 2x2 max pooling layer will take the largest value from each 2x2 block.

$$MaxPool(x) = max_{j \epsilon R_i} x_j \quad \ldots (eq.\ 3.5)$$

ii. Average Pooling:

Computes the average value of each patch in the feature map.

$$AvgPool(x) = \frac{1}{|R_i|} * \Sigma_{j \epsilon R_i}(x_j) \quad \ldots (eq.\ 3.6)$$

3.1.2.4. Fully Connected Layers

After several convolutional and pooling layers, the output is usually flattened and fed into one or more fully connected (dense) layers. These layers perform high-level reasoning and decision-making, leading to the final output.

- Flattening: Converts the 2D feature maps into a 1D vector.
- Dense Layer: Each neuron in a dense layer is connected to every neuron in the previous layer, allowing for the combination of all extracted features to make predictions.

3.1.2.5. Dropout Layers

Dropout is a regularization technique used to prevent overfitting in neural networks. During training, dropout randomly sets a fraction of input units to zero, which helps make the model more robust by preventing it from relying too heavily on any particular neurons.

3.1.2.6. CNN Architecture

Figure 3.1 explains the basic architecture of a convolutional neural network.

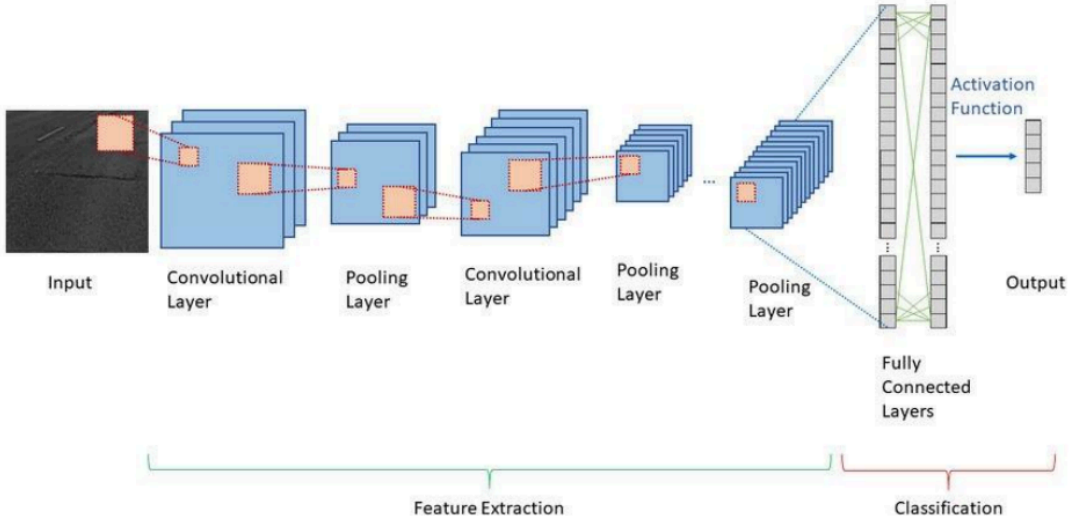


Fig. 3.1 Convolutional Neural Network (CNN) architecture

3.1.3. Training and Optimization

Training a CNN involves feeding it large amounts of labeled images, where the network learns to recognize patterns and features through multiple convolutional layers, using backpropagation and optimization techniques to minimize the error in its predictions.

3.1.3.1. Forward Propagation

In forward propagation, the input data is passed through the network, layer by layer, to compute the output predictions.

- Input Layer: Receives the input image and passes it to the first convolutional layer.
- Convolutional Layers: Apply convolution operations with filters, followed by activation functions (e.g., ReLU) to produce feature maps.
- Pooling Layers: Reduce the spatial dimensions of the feature maps to reduce computational complexity and improve feature generalization.
- Fully Connected Layers: After several convolutional and pooling layers, the output is flattened and passed through one or more fully connected layers to perform high-level reasoning.
- Output Layer: Produces the final predictions, often using a softmax function for classification tasks to generate probabilities for each class.

3.1.3.2. Loss Calculation

The loss function measures the discrepancy between the predicted outputs and the actual labels. The choice of loss function depends on the task:

● Cross-Entropy Loss: Commonly used for classification tasks. It measures the difference between two probability distributions, the predicted probabilities and the actual distribution (one-hot encoded labels).

$$L(y, \widehat{y}) = -\sum_{i=1}^{N} y_i log(\widehat{y}_i) \quad \dots (eq.\ 3.7)$$

● Mean Squared Error (MSE): Used for regression tasks. It measures the average squared difference between the actual and predicted values.

$$L(y, \widehat{y}) = \frac{1}{N}\sum_{i=1}^{N} (y_i - \widehat{y}_i)^2 \quad \dots (eq.\ 3.8)$$

3.1.3.3. Backpropagation

Backpropagation involves computing the gradients of the loss function concerning each parameter in the network. These gradients are used to update the parameters to minimize the loss.

● Chain Rule: Gradients are computed using the chain rule of calculus, propagating the error from the output layer back through the network layers.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial \theta} \quad \dots (eq.\ 3.9)$$

● Gradient Calculation: For each layer, the gradient of the loss with respect to its inputs, weights, and biases is calculated.

3.1.3.4. Parameter Update

Once the gradients are computed, the network's parameters are updated using an optimization algorithm. The most commonly used optimization algorithms include:

● **Stochastic Gradient Descent (SGD):**

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t) \quad \dots (eq..\ 3.10)$$

Where γ is the learning rate, $\nabla_\theta L(\theta_t)$ and is the gradient of the loss with respect to the parameters.

● **Momentum:** An extension of SGD that accumulates a velocity vector in the direction of the gradients to smooth out updates.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t \qquad \ldots (eq.\ 3.11)$$

Where γ is the momentum factor.

● **Adam (Adaptive Moment Estimation):** Combines the advantages of both SGD with momentum and RMSProp. It maintains running averages of both the gradients and their squared values.

$$r_t = \beta_1 r_{t-1} + (1 - \beta) \nabla_\theta L(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta L(\theta_t))^2$$

$$\hat{r} = \frac{r_t}{1-\beta_1^t}, \hat{v_t} = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{r_t}}{\sqrt{\hat{v_t}} + \epsilon} \qquad \ldots.(\text{eq. } 3.12)$$

Where β_1 and β_2 are hyperparameters controlling the decay rates of the moment estimates, and ϵ is a small constant for numerical stability.

3.1.3.5. Optimization Techniques

Several optimization techniques can improve the training process of CNNs:

i. Learning Rate Scheduling: Adjusting the learning rate during training can help converge faster and avoid local minima. Common schedules include:

● Step Decay: Reduces the learning rate by a factor every few epochs.

● Exponential Decay: Multiplies the learning rate by a factor at every epoch.

● Adaptive Methods: Adjust the learning rate based on the performance, such as ReduceLROnPlateau.

ii. Batch Normalization: Normalizes the inputs of each layer to have a mean of zero and a variance of one. This helps in accelerating training and improving model stability.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \text{..(eq. 3.13)}$$

Where μ and σ are the mean and variance of the batch.

iii. Dropout: Regularization technique where randomly selected neurons are ignored during training, reducing the likelihood of overfitting.

$$h = dropout(h, p) \quad \text{…(eq. 3.14)}$$

Where h is the dropout rate.

iv. Data Augmentation: Applying random transformations to the training data, such as rotations, translations, and flips, to artificially increase the size of the training set and improve generalization.

3.1.3.6. Evaluation and Validation

Evaluating a CNN involves measuring its performance on a separate validation set that was not used during training. Common metrics include accuracy, precision, recall, F1-score, and more, depending on the task.

- **Training-Validation Split:** Dividing the dataset into training and validation sets to monitor the network's performance and avoid overfitting.

- **Early Stopping:** Halting training when the validation performance stops improving, to prevent overfitting.

- **Cross-Validation:** Splitting the dataset into multiple folds and training multiple models to ensure robustness and reliability of the results.

3.2 Bidirectional Long Short-Term Memory (BiLSTM)

3.2.1 Introduction

Long Short-Term Memory (LSTM) networks are a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies. Introduced by Hochreiter and Schmidhuber (1997), LSTMs were explicitly designed to overcome the limitations of traditional RNNs, such as vanishing and exploding gradients. They are widely used in sequence modeling tasks such as language modeling, sentiment analysis, and hate speech detection, where understanding the context of a word relative to its surrounding tokens is critical.

A **Bidirectional LSTM (BiLSTM)** enhances the traditional LSTM by processing the sequence in both forward and backward directions, allowing the model to capture both past (left) and future (right) contexts at each point in the sequence. This is especially beneficial in tasks involving sentence-level understanding.

3.2.2 Basic Structure of an LSTM

An LSTM unit consists of a memory cell and three primary gates: the input gate, forget gate, and output gate. These gates regulate the flow of information, allowing the cell to retain or discard information across long sequences.

3.2.2.1 Gates in LSTM

LSTM networks are designed to regulate information flow through the use of **three gates—forget**, **input**, and **output**—along with a **cell state** that carries memory through time. These components allow LSTM networks to selectively retain or discard information, making them effective in modeling long-term dependencies in sequences such as natural language.

Each gate uses a sigmoid activation function to generate values between 0 and 1, where 0 indicates complete blockage and 1 allows full passage of information.

a) Forget Gate

The **forget gate** determines which information from the previous cell state C_{t-1} should be discarded or retained.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right) \quad \ldots eq(3.15)$$

- **Purpose:** Controls the "memory erasure" mechanism.
- **Input:** Current input x_t and previous hidden state h_{t-1}.
- **Output:** A vector of values between 0 and 1, indicating the degree to which each element of the cell state should be forgotten.
- **Interpretation:** A value close to 0 means "forget this completely," and close to 1 means "retain this entirely."

b) Input Gate

The **input gate** decides what new information will be added to the cell state.

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right) \quad \ldots eq(3.16)$$

$$\bar{c}_t = tanh\left(W_c \cdot \left[h_{t-1}, x_t\right] + b_c\right) \quad \ldots eq(3.17)$$

- **Purpose:** Determines how much of the new input should be written into memory

.

- **Input:**
    - i_t: gate controlling the amount of new information.
    - $\bar{c}_t$: candidate cell state (new information).
- **Output:** The product $i_t \cdot c_t$ is added to the cell state.
- **Interpretation:** Helps LSTM cells learn new patterns while preserving useful past knowledge.

c) Cell State Update

The **cell state** c_t is updated as:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \bar{c}_t \quad \ldots eq(3.18)$$

- **Purpose:** Combines old memory (after forgetting irrelevant parts) and new memory.
- **Interpretation:** This is the memory line that allows information to flow unchanged over many time steps.

d) Output Gate

The **output gate** determines what information from the current cell state should be output and passed on to the next time step.

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right) \quad \ldots eq(3.19)$$

$$h_t = o_t \cdot tanh\left(c_t\right) \quad \ldots eq(3.20)$$

- **Purpose:** Controls exposure of the internal cell state to the rest of the network.
- **Input:** Uses the updated cell state c_t, which is first passed through a tanh activation function to limit the values between -1 and 1.
- **Output:** Final hidden state h_t, passed to the next time step and/or output layer.
- **Interpretation:** Selects and transforms part of the memory to serve as output.

3.2.3 Bidirectional LSTM

A BiLSTM consists of two LSTM layers:

- A forward LSTM that reads the input sequence from t=1 to T
- A backward LSTM that reads the input from t=T to 1

The final output h_t at each time step is the concatenation of both LSTMs:

$$h_t = [\ \overrightarrow{h_t}; \overleftarrow{h_t}\ ] \qquad \dots eq(3.21)$$

This architecture is particularly useful in natural language processing tasks where future context is as important as the past (e.g., understanding negations or sarcasm in hate speech).

3.2.4 BiLSTM Architecture

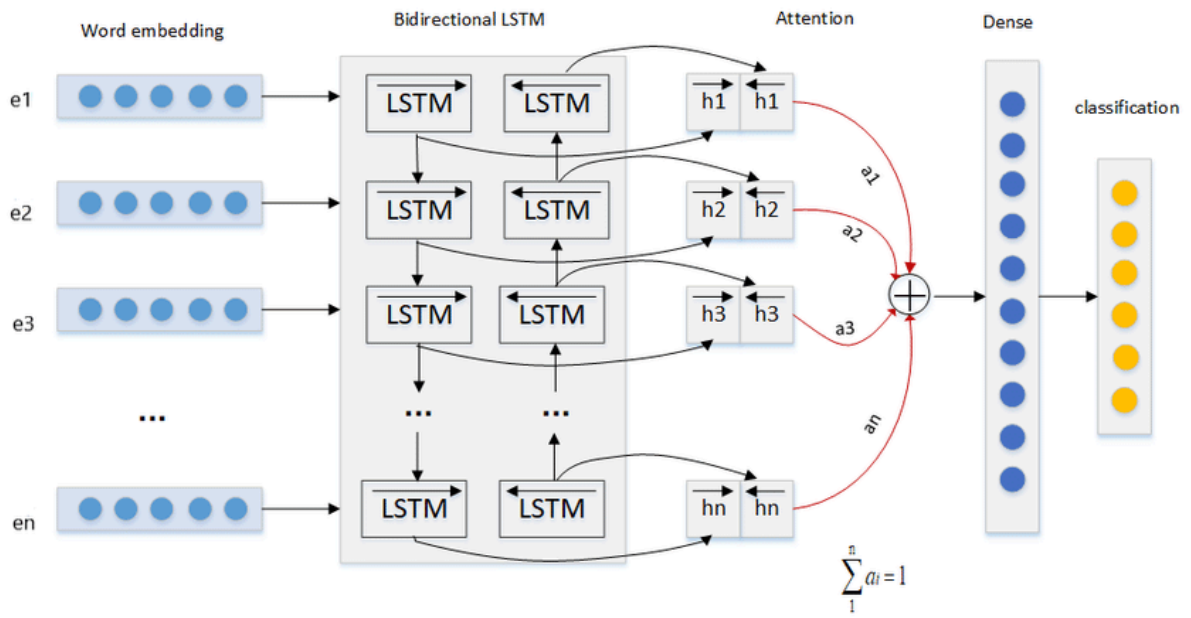


Fig. 3.2 Bi-Long Short Term Memory(Bi-LSTM) architecture

The architecture typically consists of the following layers:

- **Embedding Layer:** Converts tokenized input text into dense vectors using pretrained embeddings (e.g., GloVe).
- **BiLSTM Layer(s):** Processes input in both directions.

- **Dropout Layer:** Applied to prevent overfitting.
- **Dense Layer(s):** Fully connected layers for transformation and decision-making.
- **Softmax/Output Layer:** Produces classification scores.

3.2.5 Training and Optimization

3.2.5.1 Forward Propagation

- Each word in the sentence is embedded and passed into both the forward and backward LSTM cells.
- Outputs from both directions are concatenated and forwarded to the dense layers.

3.2.5.2 Loss Function

- For binary classification (e.g., Hate vs. Non-Hate), the **binary cross-entropy loss** is used:

$$L\left(y,\widehat{y}\right) = -\left[ylog\left(\widehat{y}\right) + (1 - y)log\left(1 - \widehat{y}\right)\right] \quad \ldots eq(3.22)$$

3.2.5.3 Backpropagation Through Time (BPTT)

- BiLSTM parameters are updated via BPTT, an extension of backpropagation applied to sequence models.
- Gradients are calculated for each timestep and averaged across directions.

3.2.6 Advantages of BiLSTM for Hate Speech Detection

- **Contextual Understanding:** Captures full context from both sides of a word.
- **Sequential Sensitivity:** Maintains word order, crucial in interpreting sentence semantics.
- **Robustness to Short Text:** Especially effective for tweets where word position drastically affects meaning.

3.2.7 Limitations

- **Higher Computational Cost:** Requires more memory and training time than unidirectional LSTMs.
- **Overfitting Risk:** Requires regularization techniques like dropout and early stopping.
- **Less Parallelizable:** Due to sequential nature, training can't be fully parallelized like CNNs.

3.2.8 Implementation Note

In this research, a BiLSTM was implemented using TensorFlow/Keras with the following configuration:

- Units: 128
- Dropout: 0.5
- Optimizer: Adam
- Input: Padded tweets of length 30 using GloVe embeddings (100D)

3.3 Support Vector Machines (SVMs)

3.3.1 Introduction

Support Vector Machines (SVMs) are powerful supervised learning models used primarily for binary classification tasks. Introduced by Vladimir Vapnik in the 1990s, SVMs aim to find the optimal hyperplane that separates data points of different classes with the maximum possible margin. Due to their effectiveness in high-dimensional spaces and robustness to overfitting, SVMs have been widely adopted in text classification tasks, including hate speech detection, spam filtering, and sentiment analysis [4], [5].

3.3.2 Basic Concept of SVM

In the simplest case of linear separability, SVM tries to find a hyperplane that best divides the dataset into two classes. The data points closest to the hyperplane are called **support vectors**, and they play a critical role in defining the decision boundary.

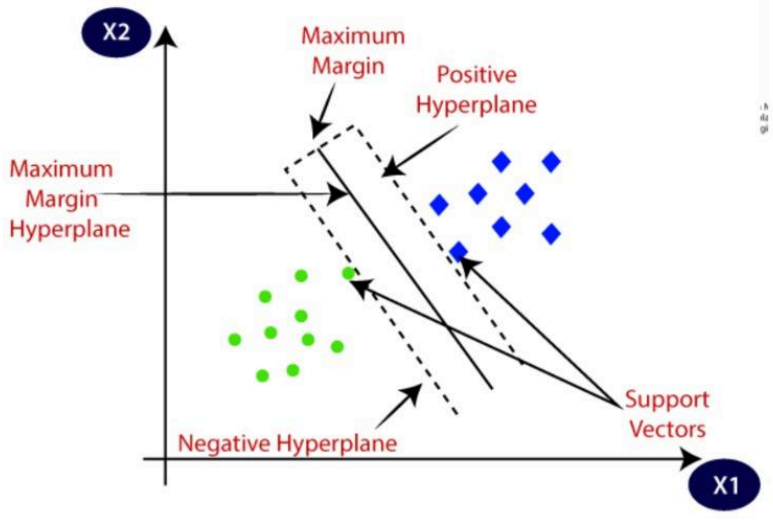


Fig. 3.3 Support Vector Machine(SVM) architecture

3.3.2.1 Mathematical Formulation

Given a dataset of labeled samples:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\ldots\ldots, (x_n, y_n)\},. \qquad x_i \epsilon R^d,\ y_i \epsilon \{-1, +1\}$$

The goal is to find a hyperplane defined by:

$$w \cdot x + b = 0 \qquad \ldots eq(3.23)$$

Where:

- w is the weight vector
- b is the bias term

The constraints for correct classification are:

$$y_i(w \cdot x_i + b) \geq 1 \ \forall i \qquad \ldots eq(3.24)$$

3.3.2.2 Objective Function (Hard Margin SVM)

SVM aims to minimize the norm of the weight vector while satisfying the margin constraints:

$$min_{\omega,b} \ \frac{1}{2}||w||^2 \ subject\ to\ y_i(w.x_i + b) \geq 1 \qquad .. eq\ (3.25)$$

This ensures the maximum margin between classes.

3.3.3 Soft Margin SVM

Real-world data is often not linearly separable. To accommodate this, **soft margin SVMs** introduce slack variables ξi\xi_iξi to allow some misclassification:

$$min_{\omega,b,\xi} \ \frac{1}{2}||w||^2 + \ C\sum_{i=1}^{n} \xi_i \ subject\ to\ y_i(w.x_i + b) \geq 1 - \xi_i, \xi_i \geq 0 \qquad .. eq\ (3.26)$$

Where:

- C is the **regularization parameter** that balances the trade-off between margin width and classification error.

3.3.4 Kernel Trick

In cases where data is not linearly separable in its original feature space, SVMs utilize the **kernel trick** to map data to a higher-dimensional space where a separating hyperplane is possible.

Let $\phi(x)$ be a non-linear transformation. The kernel function computes the inner product in the transformed space:

$$K\left(x_i, x_j\right) =< \phi\left(x_i\right), \phi\left(x_j\right) > \quad \ldots eq(3.27)$$

Common kernels include:

- **Linear Kernel:** $K\left(x_i, x_j\right) = x_i^T x_j$
- **Polynomial Kernel:** $K\left(x_i, x_j\right) = \left(x_i^T x_j + 1\right)^d$
- **RBF (Gaussian) Kernel:** $K\left(x_i, x_j\right) = exp\left(- \gamma \left| x_i - x_j \right|^2\right)$
- **Sigmoid Kernel:** $K\left(x_i, x_j\right) = tanh\left(\alpha x_i^T x_j + r\right)$

3.3.5 Hinge Loss Function

SVM uses the **hinge loss** function to measure prediction error:

$$L\left(y, \widehat{y}\right) = max\left(0, 1 - y.\widehat{y}\right) \quad \ldots eq(3.28)$$

Where:

- $y \in \{-1, +1\}$
- $\widehat{y} = w \cdot x + b$

This loss is zero if the prediction is correct and at least one unit away from the margin; otherwise, it increases linearly.

3.3.6 SVM for Text Classification

In hate speech detection and other NLP tasks, SVMs are often used with high-dimensional **TF-IDF vectors** representing documents. Despite the sparsity of text data, SVM performs well due to:

- Its ability to handle high-dimensional spaces
- Robust generalization, especially with appropriate kernel choice
- Independence from probability estimates (unlike Naïve Bayes)

3.3.7 Implementation Details

In our experiments, SVM was implemented using Scikit-learn with:

- **Kernel:** Linear
- **C:** 1.0 (default, tuned during experimentation)
- **Input Features:** TF-IDF (unigram and bigram)
- **Cross-validation:** 10-fold
- **Evaluation Metrics:** Precision, Recall, F1-score

3.3.8 Limitations of SVM

- **Computationally Intensive:** Especially with large datasets and non-linear kernels.
- **Binary Classification Focus:** Extensions are needed (e.g., One-vs-Rest) for multi-class problems.
- **Interpretability:** Harder to interpret compared to decision trees or rule-based models.
- **Sensitivity to Feature Scaling:** Requires normalization for best results.

3.4 Extreme Gradient Boosting (XGBoost)

3.4.1 Introduction

Extreme Gradient Boosting (XGBoost) is a highly optimized and scalable implementation of gradient boosting machines (GBMs). Proposed by Tianqi Chen and Carlos Guestrin in 2016 [1], XGBoost has become a top-performing algorithm for a wide range of structured data tasks, including classification, regression, and ranking. It is known for its speed, accuracy, and ability to handle missing values and large-scale datasets.

In the context of text classification tasks like hate speech detection, XGBoost performs well when combined with feature representations such as TF-IDF or word embeddings, and can be used either as a standalone classifier or in ensemble with deep models.

3.4.2 Gradient Boosting Overview

Gradient Boosting is an ensemble learning technique that builds multiple decision trees in a sequential manner. Each tree tries to correct the residual errors made by the previous ones. The final model is a weighted sum of all weak learners (trees), where each one contributes to minimizing the overall loss.

3.4.2.1 Objective Function

The general objective function of gradient boosting is:

$$L(\theta) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{t} \Omega\left(f_k\right) \quad \ldots eq(3.29)$$

Where:

- l: Loss function (e.g., logistic loss for classification)
- f_k: k-th decision tree
- $\Omega\left(f_k\right)$: Regularization term
- $\hat{y}_i^{(t)}$: Prediction at iteration t

3.4.3 Structure of XGBoost

XGBoost introduces several improvements over traditional gradient boosting:

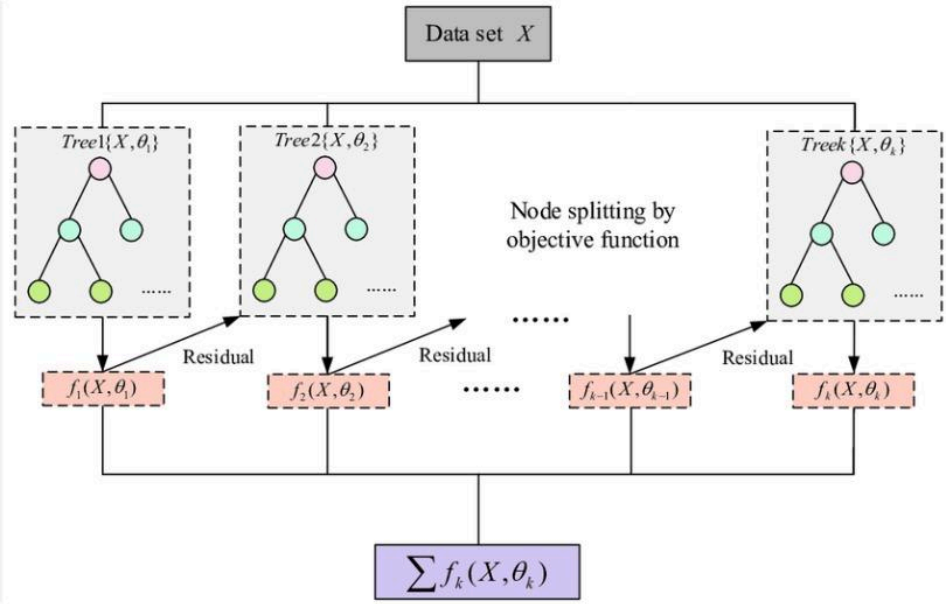


Fig. 3.4 Extreme Gradient Boosting(XGB) architecture

3.4.3.1 Additive Training

Predictions are updated at each stage as follows:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t\left(x_i\right) \quad \ldots eq(3.30)$$

Where f_t is the new function (tree) added at iteration t.

3.4.3.2 Regularization

To prevent overfitting, XGBoost introduces a regularized objective:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad \ldots eq(3.31)$$

Where:

- T: Number of leaves in the tree
- w_j: Weight of leaf j
- γ: Penalty for adding a leaf
- λ: L2 regularization term on weights

3.4.4 Tree Construction

XGBoost uses a greedy algorithm to build decision trees:

- At each step, it selects the split that maximizes the **gain**, which is the reduction in loss after the split.
- The **gain** from a split is given by:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_L^2}{H_L+\lambda} - \frac{G_L^2}{H_L+\lambda}\right] - \lambda \quad \ldots eq(3.32)$$

Where G and H represent the first and second order gradient statistics for the loss function.

3.4.5 System Optimizations

XGBoost is not only algorithmically efficient but also optimized at the system level:

- **Parallel Tree Construction:** Uses block structure for efficient memory access.
- **Sparsity-aware Split Finding:** Handles missing values natively.
- **Cache-aware Access:** Minimizes memory bandwidth usage during training.

3.4.6 Feature Importance and Interpretability

One advantage of XGBoost is its **explainability**. It provides several importance metrics:

- **Gain:** Improvement brought by a feature to the branches it is on.
- **Cover:** Relative number of observations related to that feature.
- **Frequency:** Number of times a feature is used in trees.

These metrics can be visualized to understand which features (e.g., specific words or phrases) most strongly influence classification.

3.4.7 Application to Hate Speech Detection

In this research, XGBoost is used as a baseline model with the following setup:

- **Input Features:** TF-IDF vectors of tweets (unigram and bigram)
- **Objective:** Binary classification (Hate vs. Non-Hate)
- **Booster:** gbtree
- **Max Depth:** 6 (tuned)
- **Learning Rate (η):** 0.1
- **Evaluation Metric:** F1-score, Precision, Recall

Despite being a non-neural approach, XGBoost demonstrated competitive performance, particularly in terms of precision, and required significantly lower training time compared to deep models.

3.4.8 Advantages of XGBoost

- **Highly Accurate:** Incorporates both L1 and L2 regularization.
- **Fast and Scalable:** Optimized for performance and parallel computation.
- **Robust to Missing Data:** Handles sparsity well.
- **Interpretable:** Provides insight into feature importance.
- **Works Well with Limited Data:** Does not require extensive training samples like deep learning models.

3.4.9 Limitations

- **Lacks Sequential Understanding:** Unlike BiLSTM or Transformers, XGBoost does not capture word order or contextual dependencies.
- **Manual Feature Engineering:** Relies heavily on effective preprocessing and representation (e.g., TF-IDF).
- **Hyperparameter Sensitivity:** Requires tuning of many hyperparameters (e.g., learning rate, tree depth, regularization terms).

3.5 Multilayer Perceptron (MLP)

3.5.1 Introduction

The **Multilayer Perceptron (MLP)** is a class of feedforward artificial neural networks (ANNs) composed of multiple layers of interconnected neurons. Unlike single-layer perceptrons, which can only solve linearly separable problems, MLPs can model complex non-linear functions through the use of one or more hidden layers and non-linear activation functions.

MLPs are considered the foundation of many modern deep learning architectures and are especially useful for classification and regression tasks involving structured data or vectorized text representations such as TF-IDF or word embeddings.

3.5.2 Architecture of an MLP

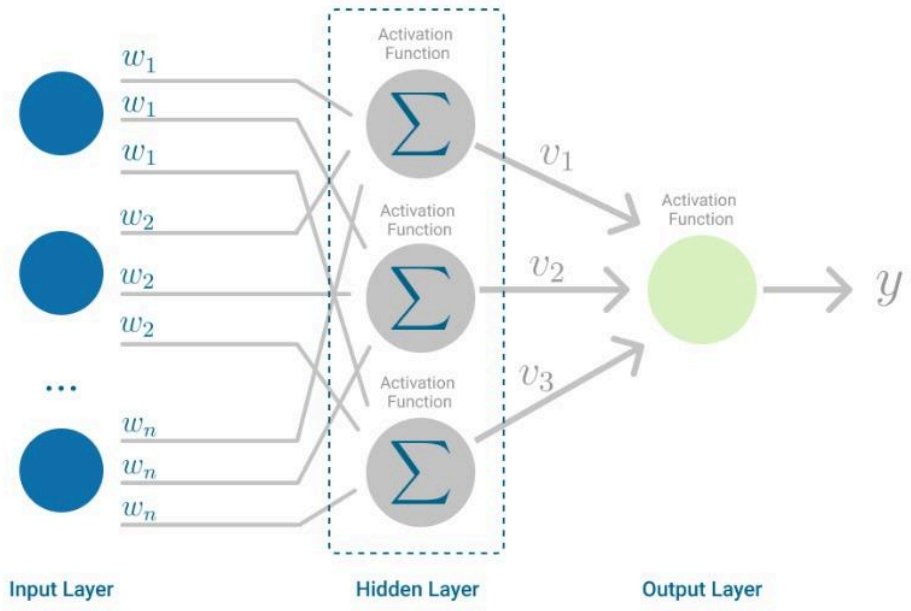


Fig. 3.5 Multi Layer Perceptron(MLP) architecture

An MLP consists of three main types of layers:

- **Input Layer:** Receives the input feature vector (e.g., TF-IDF or word embeddings).

- **Hidden Layer(s):** Intermediate layers containing neurons that apply non-linear transformations.
- **Output Layer:** Produces the final predictions, often with an activation suited to the task (e.g., softmax for classification).

3.5.2.1 Neuron Operation

Each neuron performs a weighted sum followed by an activation function:

$$z = \sum_{i=1}^{n} w_i x_i + b \quad \ldots eq(3.33)$$

$$a = \phi(z) \quad \ldots eq(3.34)$$

Where:

- x_i: input features
- w_i: weights
- b: bias
- ϕ: activation function (e.g., ReLU, sigmoid)

3.5.3 Activation Functions

Activation functions introduce non-linearity, allowing the network to learn complex patterns:

- **ReLU (Rectified Linear Unit):**

$$ReLU(x) = max(0, x) \quad \ldots eq(3.35)$$

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \ldots eq(3.36)$$

- **Tanh:**

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \ldots eq(3.37)$$

- **Softmax:** Used in the output layer for multiclass classification

$$softmax\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \ldots eq(3.38)$$

3.5.4 Loss Functions

The choice of loss function depends on the task:

- **Binary Classification:** Binary Cross-Entropy

$$L\left(y, \widehat{y}\right) = -\left[ylog\left(\widehat{y}\right) + (1 - y)log\left(1 - \widehat{y}\right)\right] \quad \ldots eq(3.39)$$

- **Multiclass Classification:** Categorical Cross-Entropy

$$L\left(y, \widehat{y}\right) = -\sum_{i=1}^{k} y_i log\left(\widehat{y}_i\right) \quad \ldots eq(3.40)$$

3.5.5 Training MLP

Training involves three key steps:

3.5.5.1 Forward Propagation

Each layer computes activations from the inputs, passing the result forward.

3.5.5.2 Loss Computation

The error between predicted and true outputs is measured using the appropriate loss function.

3.5.5.3 Backpropagation

Gradients of the loss with respect to weights are computed using the chain rule and propagated backward.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} . \frac{\partial a}{\partial z} . \frac{\partial z}{\partial w} \quad \ldots (eq.\ 3.41)$$

3.5.5.4 Parameter Update

Weights are updated using gradient descent or an optimizer like Adam or RMSProp:

$$w_{t+1} = w_t - \eta . \frac{\partial L}{\partial w_t} \quad \ldots (eq.\ 3.42)$$

3.5.6 Application to Hate Speech Detection

In this study, MLP was used as a lightweight neural baseline for hate speech detection, particularly for vectorized tweet representations:

- **Input:** TF-IDF vectors (1-gram, 2-gram)
- **Hidden Layers:** 2 layers with 128 and 64 neurons respectively
- **Activation:** ReLU
- **Output:** Single neuron with sigmoid activation (binary classification)
- **Optimizer:** Adam (learning rate = 0.001)
- **Dropout:** 0.5 to reduce overfitting

3.5.7 Advantages

- **Simplicity:** Easy to implement and interpret.
- **Flexibility:** Can model non-linear decision boundaries.
- **Baseline Performance:** Performs reasonably well on small to medium-sized datasets.

3.5.8 Limitations

- **No Temporal Modeling:** Cannot capture sequential dependencies like BiLSTM.
- **No Spatial Awareness:** Unlike CNNs, MLPs don’t leverage local spatial structure.
- **Overfitting:** More prone to overfitting with limited data unless regularized.

4.METHODOLOGY

4.1 Dataset Overview

Hate Speech Twitter Dataset: The Hate Speech Twitter Dataset is a widely used benchmark corpus designed for the study of abusive language detection on social media platforms. It supports the development and evaluation of algorithms for hate speech identification, offensive language classification, and sentiment analysis in short-form user-generated content.

This dataset is particularly valuable for training and testing natural language processing (NLP) models aimed at automatic content moderation, especially in the context of identifying and mitigating harmful online behavior.

4.1.1 Key Features

i. Dataset Size and Format

- **Total Size**: ~2 MB (compressed CSV format)
- **Data Entries**: 24,783 tweets (labeled)
- **Text Format**: Plain text (tweet content)
- **Label Format**: CSV (Comma-Separated Values) with multiple columns for tweet ID, class label, and tweet text

ii. Number of Items

- **Total Tweets**: 24,783
- **Unique Classes**: 3 (Hate Speech, Offensive Language, Neutral)
- **Label Distribution**:
    - **Hate Speech**: ~1,430 tweets
    - **Offensive Language**: ~19,190 tweets
    - **Neutral (Neither)**: ~ 4,163 tweets

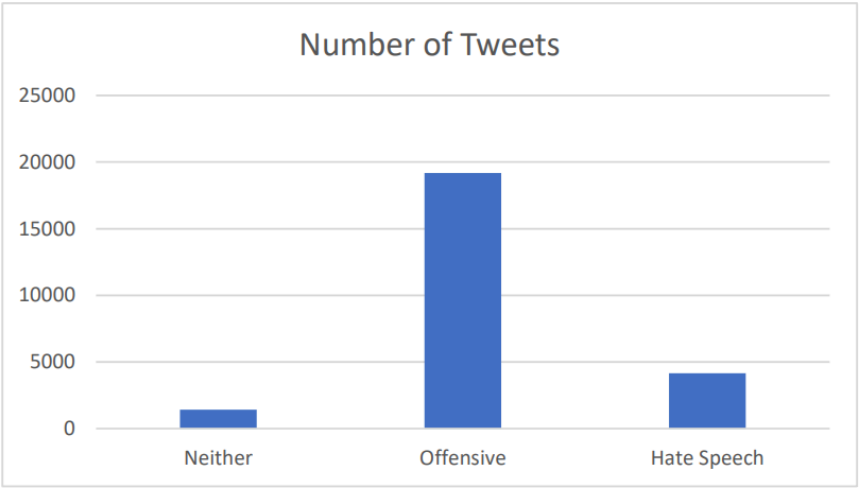


Fig. 4.1 Class Distribution of the Dataset

iii. Content and Structure

- **Tweet Length**: Varies from short expressions to full sentences (typical range: 10–30 words)
- **Language Style**: Informal and noisy, often containing:
    - Abbreviations and slang
    - hashtags
    - Misspellings and grammatical inconsistencies
- **Data Columns**:
    - **Tweet ID**: Unique identifier for each tweet
    - **Class Label**: Integer or string indicating the category (0 = Hate Speech, 1 = Offensive, 2 = Neutral)
    - **Tweet Text**: Actual content of the tweet
- **Multilingual Support**: Primarily English
- **Annotation Guidelines**: Labels are manually annotated based on predefined definitions of hate speech and offensive content

iv. Target Labels

- **Classification Target**: The dataset is used for multi-class classification, where each tweet is assigned one of the following labels:
    - **Hate Speech**: Language that expresses hatred or encourages violence against a group based on attributes like race, religion, ethnicity, gender, or orientation.
    - **Offensive Language**: Profane or insulting language that is not directed toward a protected group.
    - **Neutral**: Non-harmful, inoffensive content with no abusive language.

This label structure enables researchers to develop models that not only identify toxic content but also distinguish between degrees of severity, which is crucial for effective content moderation on social platforms.

4.2 Preprocessing and Augmentation

To ensure the dataset was suitable for training deep learning models and to enhance performance across all classes, a structured preprocessing and augmentation pipeline was implemented. This pipeline was designed to clean, normalize, and enrich the dataset, particularly focusing on addressing class imbalance and the noisy nature of social media text.

4.2.1 Preprocessing Pipeline

i. Text Cleaning and Normalization

The following preprocessing steps were applied to each tweet to remove irrelevant artifacts and standardize textual content:

- **Lowercasing**: All characters in the tweets were converted to lowercase to avoid duplicate representations of the same word in different cases (e.g., "Hate" vs. "hate").
- **Punctuation Removal**: All punctuation marks were removed to reduce syntactic noise.
- **URL Removal**: Hyperlinks (e.g., "http://...") were stripped from the tweets as they do not contribute to semantic interpretation.
- **User Mentions**: User handles (e.g., @username) were removed to eliminate user-specific references that do not generalize.
- **Hashtag Removal**: While hashtags can be informative, they were removed to maintain consistency and reduce sparsity in the vocabulary.
- **Whitespace Normalization**: Extra spaces were trimmed to maintain clean input structure.

ii. Tokenization and Sequencing

Once cleaned, the text was tokenized and transformed into numerical sequences using **Keras' Tokenizer** utility:

- **Tokenization**: Converts words into integer indices based on their frequency in the corpus.
- **Padding**: Sequences were padded with zeros to ensure a **uniform sequence length of 30 tokens**, which was selected based on the empirical distribution of tweet lengths in the dataset.
- **Out-of-Vocabulary (OOV) Handling**: Rare or unseen words during inference are mapped to a special OOV token to maintain consistent input dimensions.

4.2.2 Data Augmentation Techniques

To address the **class imbalance problem**—with the hate speech class significantly underrepresented—data augmentation techniques were applied specifically to the minority class. These techniques aim to increase training data variability and improve the model's ability to generalize.

i. Synonym Replacement

- Randomly selects non-stopwords in a sentence and replaces them with their synonyms using the **WordNet lexical database**.
- Maintains semantic coherence while introducing lexical diversity.

Fig. 4.2 Synonym Word Replacement

ii. Random Insertion

- Inserts a synonym of a randomly selected word at a random position in the sentence.
- Adds syntactic variation without significantly changing the core meaning.

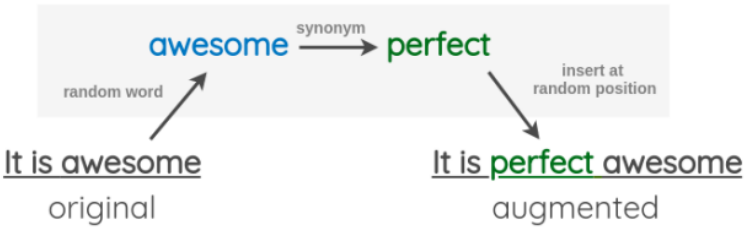


Fig. 4.3 Random Insertion

iii. Random Word Swapping

- Randomly chooses two words in the sentence and swaps their positions.
- Mimics informal grammar and word order found in social media text.

iv. Random Deletion

- Randomly removes words from the sentence with a defined probability.
- Simulates truncated or incomplete tweets, which are common in real-world social media posts.

These augmentation strategies were inspired by the Easy Data Augmentation (EDA) framework proposed by Wei and Zou [15], tailored to preserve meaning while increasing diversity in the minority class (hate speech). The augmented data were added back to the training set, effectively **doubling the number of hate speech samples** and improving performance metrics such as **recall** and **macro-averaged F1-score**.

4.3 Feature Representation

To transform raw text data into machine-understandable numerical formats, three different feature representation strategies were applied. Each method was chosen based on compatibility with the model architecture and its effectiveness in capturing linguistic, semantic, or contextual information from social media text. These representations are critical for enabling the models to differentiate between hate speech, offensive content, and neutral language.

4.3.1 TF-IDF Vectorization

i. Overview

Term Frequency–Inverse Document Frequency (TF-IDF) is a statistical representation technique that quantifies the importance of a word in a document relative to a corpus. It is widely used in classical machine learning pipelines for text classification.

ii. Mathematical Definition

For a word *t* in a document *d*, within a corpus *D*:

$$Tf - Idf(t, d, D) = TF(t, d) * log(\frac{N}{|\{d' \in D : t \in d'\}|}) \quad \dots (eq\ 4.1)$$

Where:

- **TF(t, d)** is the term frequency of word *t* in document *d*,
- **N** is the total number of documents in the corpus,
- **IDF** is the inverse document frequency that penalizes common terms.

iii. Application

- Applied to classical models such as **SVM**, **MLP**, and **XGBoost**.
- A vocabulary size of the **top 5,000 tokens** was selected based on frequency to maintain a balance between feature richness and computational efficiency.
- The resulting feature vectors are sparse and high-dimensional, with each tweet represented as a fixed-length vector.

4.3.2 GloVe Word Embeddings

i. Overview

GloVe (Global Vectors for Word Representation) is an unsupervised learning algorithm that generates dense vector representations of words based on word co-occurrence statistics from large text corpora. Unlike one-hot encoding or TF-IDF, GloVe captures semantic relationships between words.

ii. Embedding Details

- **Vector Dimensions**: 100-dimensional pretrained embeddings
- **Source**: GloVe embeddings trained on 6B tokens from Wikipedia and Gigaword corpora
- **Static Nature**: Each word has a single, fixed vector regardless of context

iii. Application

- Used in deep learning models like **CNN** and **Bi-LSTM**
- Each tweet was converted into a sequence of word embeddings, where each token maps to a 100-dimensional GloVe vector
- Unknown words were assigned a zero vector or a learned OOV token

- Enabled effective learning of syntactic and semantic word-level relationships even on limited annotated data

4.3.3 Contextual Embeddings (BERT and Variants)

i. Overview

To capture the full context in which words appear, transformer-based models such as **BERT (Bidirectional Encoder Representations from Transformers)**, **ALBERT**, and **ELECTRA** were integrated. These models generate **contextual embeddings**, where the representation of each word is dynamically influenced by its surrounding words.

ii. Tokenization and Embedding Process

- **Subword Tokenization**: Applied using model-specific tokenizers (e.g., WordPiece for BERT)
- **Contextualized Vectors**: Unlike static embeddings, the meaning of each token changes based on sentence context
- **CLS Token Representation**: The final hidden state of the special `[CLS]` token was used as a fixed-size summary vector for the entire tweet

iii. Application

- Used with fine-tuned transformer models in the classification layer
- The `[CLS]` output (768 dimensions in base models) was passed into a **fully connected layer or MLP** for classification
- Improved performance in detecting **context-sensitive**, **sarcastic**, or **implicit** hate speech expressions

iv. Advantages

- Captures bidirectional context
- Handles polysemy (multiple meanings of words) effectively
- Learns deeper linguistic structures such as syntax, dependencies, and tone

4.4 Model Architectures

To explore the effectiveness of different algorithmic approaches for hate speech classification, a range of classical and deep learning models were implemented and evaluated. These models vary in their complexity, feature dependency, and ability to capture linguistic context. Each architecture was selected based on its relevance to text classification tasks and its ability to generalize over noisy and imbalanced social media data.

4.4.1 Classical Models

To establish benchmark baselines, three classical machine learning models were implemented: **Support Vector Machines (SVM)**, **Multi-Layer Perceptron (MLP)**, and **XGBoost**. These models were trained on **TF-IDF features**, which provide a sparse vector representation of each tweet.

- **Support Vector Machine (SVM)**: Utilized with a linear kernel for efficient training on high-dimensional text data. SVMs are effective in separating classes with a large margin but are sensitive to imbalanced class distributions.
- **XGBoost**: A scalable and efficient gradient boosting algorithm, capable of handling sparse feature representations and non-linear relationships. Hyperparameters such as the number of estimators, learning rate, and maximum depth were tuned through cross-validation.
- **Multi-Layer Perceptron (MLP)**:
    - Input: TF-IDF feature vector
    - Hidden Layer: Single dense layer with 128 neurons and ReLU activation
    - Dropout Layer: Dropout rate of **0.1** to mitigate overfitting
    - Output Layer: Dense layer with softmax activation for multi-class classification
    - Loss Function: Categorical cross-entropy
    - Optimizer: Adam

This setup provided strong baseline performance and served as a comparative point for evaluating more complex deep learning models.

4.4.2 Convolutional Neural Network (CNN)

To capture spatial and compositional features in text data, a **custom 1D CNN** model was implemented using **GloVe word embeddings** as input. This model focuses on identifying local patterns such as n-gram structures that are indicative of hate speech.

We employed a custom 1D CNN architecture that processes input sentence embeddings through two consecutive convolutional layers with **32** and **64** filters respectively, each followed by ReLU activation. The output is then passed through a global max pooling layer to reduce dimensionality. This is followed by a dense layer with ReLU activation and a dropout layer to prevent overfitting. Finally, the output is mapped to class probabilities through a softmax-based classification layer. This architecture effectively captures local n-gram features and semantic patterns within the text. The model was optimized using the **RMSProp optimizer** and trained using the **binary cross-entropy** loss function.

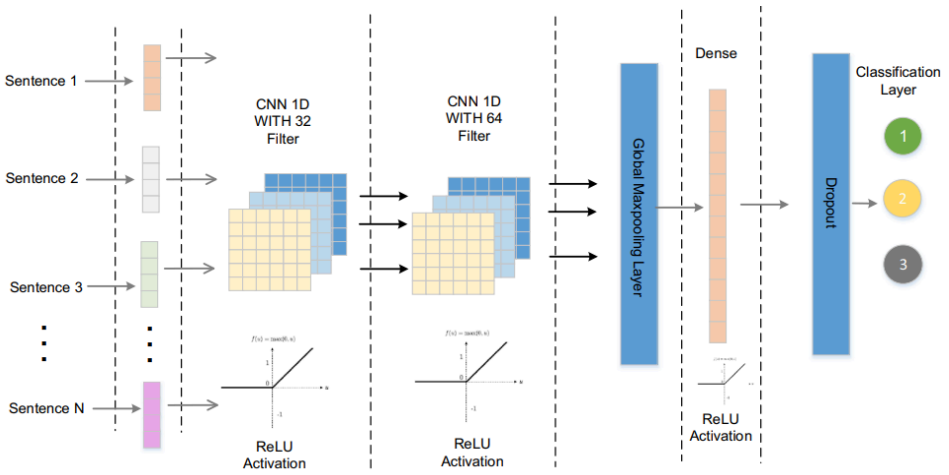


Fig. 4.4 CNN Architecture Diagram

This architecture is well-suited for capturing fixed-size patterns and is computationally efficient for short text sequences like tweets.

4.4.3 Bidirectional LSTM (Bi-LSTM)

To capture temporal and contextual dependencies in tweet sequences, a **Bi-LSTM model** was developed. This model processes the input in both forward and backward directions, allowing it to understand context that may precede or follow a given word.

A sequential Bi-LSTM was built to learn contextual features from tweet sequences. The LSTM cells preserve forward and backward dependencies, essential for understanding nuanced expressions in hate speech

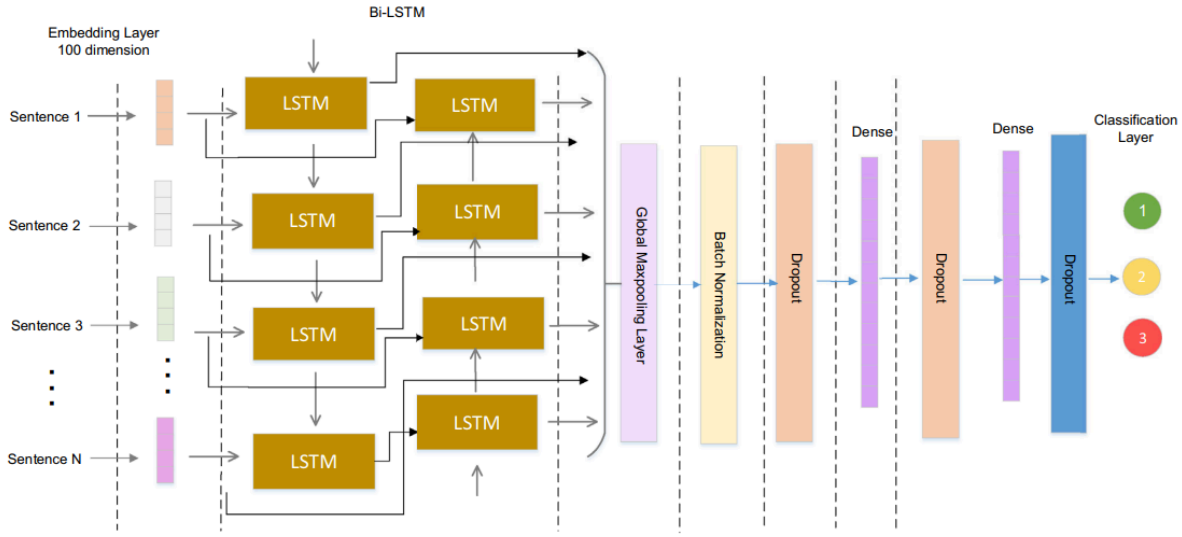


Fig. 4.5 Bi-LSTM Architecture Diagram

This architecture excels at modeling sequential relationships and is particularly effective for detecting nuanced expressions and context-dependent hate speech.

4.4.4 Transformer-Based Models (BERT, ALBERT, ELECTRA)

To harness state-of-the-art performance in language understanding, **transformer-based models** were fine-tuned using the HuggingFace Transformers library. These models utilize deep contextual embeddings that dynamically represent each word based on its context in the sentence.

- **Pretrained Models Used**:
  - BERT-base-uncased
  - ALBERT-base-v2
  - ELECTRA-small-discriminator
- **Tokenization**: Performed using model-specific subword tokenizers (e.g., WordPiece for BERT)
- **Fine-Tuning Strategy**:
  - Input text is tokenized and passed through the transformer encoder
  - The final hidden state corresponding to the `[CLS]` token is extracted
  - This is passed through a small **MLP classifier**:
    - Dense Layer: 256 units with ReLU
    - Dropout Layer: Dropout rate of 0.2
    - Output Layer: Dense with softmax activation
- **Loss Function**: Categorical cross-entropy
- **Optimizer**: AdamW with weight decay
- **Learning Rate Scheduler**: Linear scheduler with warm-up

4.5 Evaluation Metrics

To quantitatively assess the performance of the proposed classification models, standard evaluation metrics commonly used in multi-class classification tasks were employed. These metrics are especially crucial in the context of **imbalanced datasets** like hate speech detection, where the minority class (i.e., hate speech) can be easily overshadowed by majority classes such as offensive or neutral language.

The following primary metrics were computed for each class:

4.5.1 Precision

Precision measures the proportion of positive predictions that are actually correct. In the context of hate speech detection, it quantifies how many tweets predicted as "hate speech" were truly hate speech.

$$Precision = \frac{TP}{TP + FP} \quad \ldots eq(4.2)$$

Where:

- TP = True Positives (correctly predicted hate speech tweets)
- FP = False Positives (tweets wrongly predicted as hate speech)

A high precision value indicates a low false positive rate, which is critical for avoiding unjustified content moderation.

4.5.2 Recall

Recall evaluates the proportion of actual positives that were correctly identified by the model. It reflects the model's ability to capture all relevant instances of hate speech.

$$Recall = \frac{TP}{TP + FN} \quad \ldots (eq.\ 4.3)$$

Where:

- FN = False Negatives (hate speech tweets wrongly predicted as non-hate)

High recall ensures that most of the harmful content is identified and not missed by the system.

4.5.3 F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a balanced measure, particularly useful when precision and recall values are in tension.

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad \ldots (eq\ 4.4)$$

F1-score is a more reliable metric than accuracy in imbalanced classification problems, as it considers both false positives and false negatives.

4.5.4 Macro and Weighted Averages

To provide a holistic performance summary across all classes (Hate, Offensive, Neutral), the following averaging techniques were applied:

- **Macro-Averaged Metrics**:
  Compute the metric (Precision, Recall, F1) independently for each class and then take the average.

    - Treats all classes equally, regardless of their frequency
    - Useful for assessing performance on **minority classes**

- **Weighted-Averaged Metrics**:
  Compute the metric for each class, then take the average weighted by the number of true instances in each class.

    - Reflects real-world class distribution
    - Balances the performance impact of both majority and minority classes

These aggregated scores help in understanding how well the model performs across imbalanced class distributions. In contrast, **micro-averaged metrics**, while commonly used, can obscure the performance of the minority class by being dominated by the majority class results. Therefore, macro and weighted averages were prioritized for this task.

5. EXPERIMENTAL SETUP AND ANALYSIS

5.1 Dataset Preparation

5.1.1 Dataset Selection

- **Dataset**: For this study, we used the publicly available **Hate Speech Twitter Dataset**, which includes over 24,000 labeled tweets categorized into three classes: **Hate Speech**, **Offensive Language**, and **Neutral**.

- **Train-Test Split**: The dataset was divided into training, validation, and test sets using a **80:20** ratio to ensure a balanced evaluation. Data Augmentation was used to maintain class distribution across splits.

5.1.2 Text Preprocessing

The text preprocessing pipeline consisted of several cleaning and normalization steps to prepare the data for vectorization and model training.

- **Lowercasing**: All tweets were converted to lowercase to ensure uniform token representation.
- **Noise Removal**: Punctuation, user mentions (e.g., @user), hashtags, URLs, numbers, and special characters were removed using regular expressions.
- **Tokenization**: Text was tokenized using Keras' Tokenizer for deep learning models, and HuggingFace tokenizers for transformer-based models.
- **Padding**: Token sequences were padded to a fixed length of **30 tokens** based on statistical analysis of tweet lengths.
- **OOV Handling**: Words not present in the vocabulary were mapped to an "unknown" token during tokenization.

5.1.3 Data Augmentation

To mitigate class imbalance, data augmentation was applied to the **Hate Speech** class using Easy Data Augmentation (EDA) techniques:

- **Synonym Replacement**: Words were replaced with synonyms using WordNet.
- **Random Insertion and Swapping**: New words were randomly inserted or swapped to vary syntax while preserving semantics.
- **Random Deletion**: Simulated informal language patterns by randomly removing words.

These methods helped improve **recall and F1-score** for the underrepresented hate speech class.

5.2 Model Architecture

5.2.1 Classical Models

Three baseline models were trained on **TF-IDF features**:

- **SVM**: Linear kernel, effective on sparse data.
- **XGBoost**: Gradient boosting model with tuned tree depth and learning rate.

- **MLP**: One dense layer with 128 units, ReLU activation, dropout of 0.1, and softmax output layer.

5.2.2 CNN Architecture

A 1D CNN architecture was built for learning from **GloVe-embedded** tweet sequences:

- Embedding Layer: 100D GloVe vectors
- Conv1D Layer 1: 32 filters, kernel size 3, ReLU
- Conv1D Layer 2: 64 filters, kernel size 3, ReLU
- Global Max Pooling
- Dense Layer: 64 units, ReLU
- Dropout: 0.5
- Output Layer: Softmax (3 classes)

Optimizer: **RMSProp**, Loss: **Categorical Cross-Entropy**

5.2.3 Bi-LSTM Architecture

A sequential **Bidirectional LSTM** model was used to learn context-aware features:

- Embedding Layer: 100D GloVe
- Bi-LSTM: 64 units (forward and backward)
- Dropout: 0.3
- Dense Layer: 64 units, ReLU
- Output: Softmax layer

Optimizer: **Adam**, Loss: **Categorical Cross-Entropy**

5.2.4 Transformer-Based Models

Fine-tuned versions of **BERT**, **ALBERT**, and **ELECTRA** were used via HuggingFace:

- Tokenization: Subword tokenization (WordPiece for BERT)
- Feature Extraction: [CLS] token output (768D) used for classification
- Classifier: MLP with one hidden layer (256 units), dropout = 0.2
- Output: Softmax for 3-class classification

Fine-tuning done with **AdamW**, learning rate = 2e-5, warm-up = 10% of total steps.

5.3 Training the Models

5.3.1 Training Loop

All models were trained using a standard supervised learning loop consisting of:

- Loading batches

- Forward propagation
- Loss calculation

- Backpropagation and optimizer updates

Each model was trained for **50 epochs**, depending on convergence behavior. Transformer models were trained for **20 epochs** due to their pretraining.

5.3.2 Checkpointing and Early Stopping

- **Model Checkpointing**: Best-performing model on validation set (lowest val loss) was saved after each epoch.
- **Early Stopping**: Training was halted if no improvement was seen in **validation loss for 3 consecutive epochs**.

5.3.3 Hyperparameters

Table 5.1:Hyperparameter Configuration for Model Training

Parameter	Value
Batch Size	32
Learning Rate	3e-4 (DL models), 2e-5(Transformers)
Dropout Rate	0.1 -0.5
Optimizers	Adam,RMSProp, AdamW
Max Sequence Length	30 tokens
Epochs	50
Embedding Dimension	100 (GloVe), 768(BERT)

Training was performed using **Kaggle TPU v2.1** and **Google Colab GPU (Tesla P100)** for transformer-based models.

5.4 Evaluation and Loss Function

5.4.1 Quantitative Metrics

Models were evaluated using:

- **Precision**
- **Recall**
- **F1-Score**
- **Macro & Weighted F1**: To fairly assess performance across all classes, especially minority (hate speech) class.

5.4.2 Qualitative Analysis

- **Confusion Matrices**: Visualized to assess misclassifications.
- **Attention Visualization (for BERT)**: Used to interpret model's focus across input tokens.
- **Sample Predictions**: Compared model predictions to ground truth for representative tweets across all classes.

5.4.3 Loss Function

The **Categorical Cross-Entropy** loss was used across all models (except binary versions, where Binary Cross-Entropy was applied for ablation):

$$L(y, \widehat{y}) = -\sum_{i=1}^{N} y_i log\ (\widehat{y_i})$$

Where y is the true label (one-hot encoded) and y^ is the predicted probability distribution.

This loss penalizes misclassification based on the predicted probability and is suitable for multi-class problems like hate speech categorization.

6. RESULTS

This section presents the experimental results obtained from various models on the multi-class hate speech detection task using the Twitter dataset. The models evaluated span across classical machine learning algorithms, deep learning architectures, and transformer-based approaches. Evaluation metrics include **Precision**, **Recall**, and **F1-score**, with an emphasis on **macro** and **weighted averages** to account for class imbalance, especially for the underrepresented **Hate Speech** class. This analysis provides insight into each model's capacity to correctly classify socially impactful content categories such as hate speech and offensive language.

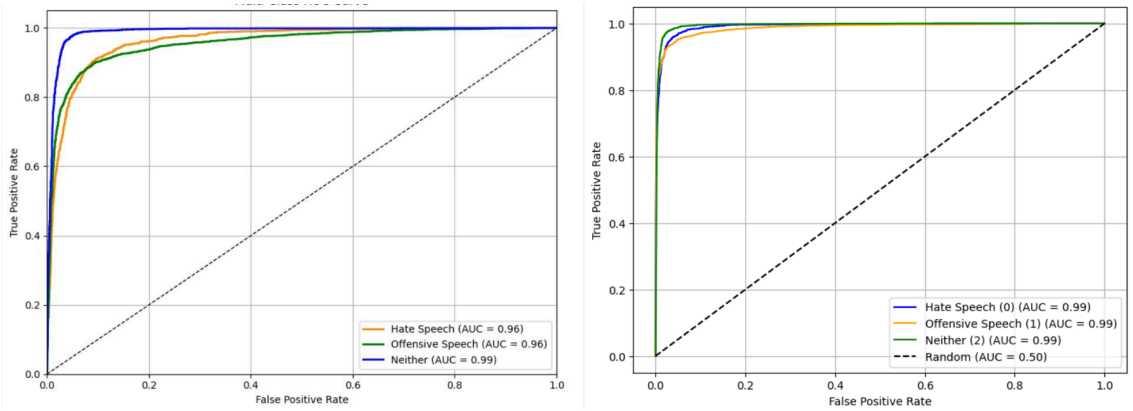
6.1 Results Using Traditional Machine Learning Models

Traditional machine learning models were trained on TF-IDF features, utilizing **unigram and bigram** representations for feature extraction. The models evaluated include:

- Support Vector Machine (SVM)
- XGBoost
- Multi-Layer Perceptron (MLP)

The **SVM** classifier achieved a **macro F1-score of 0.88** and a **weighted F1-score of 0.90**, outperforming other classical models. However, it showed **low recall (~0.40)** for the Hate Speech class, indicating a high rate of false negatives. This highlights the limitations of surface-level features in detecting nuanced or implicit hate speech.

The **MLP** model trained on TF-IDF achieved the highest among classical methods, with a **macro F1-score of 0.94** and a **weighted F1-score of 0.95** under the proposed framework. While strong on offensive and neutral tweets, MLP—like SVM—showed reduced sensitivity toward hate speech in the baseline configuration.



(a) *(b)*

Fig. 6.1 AUC-ROC curves for traditional models using TF-IDF

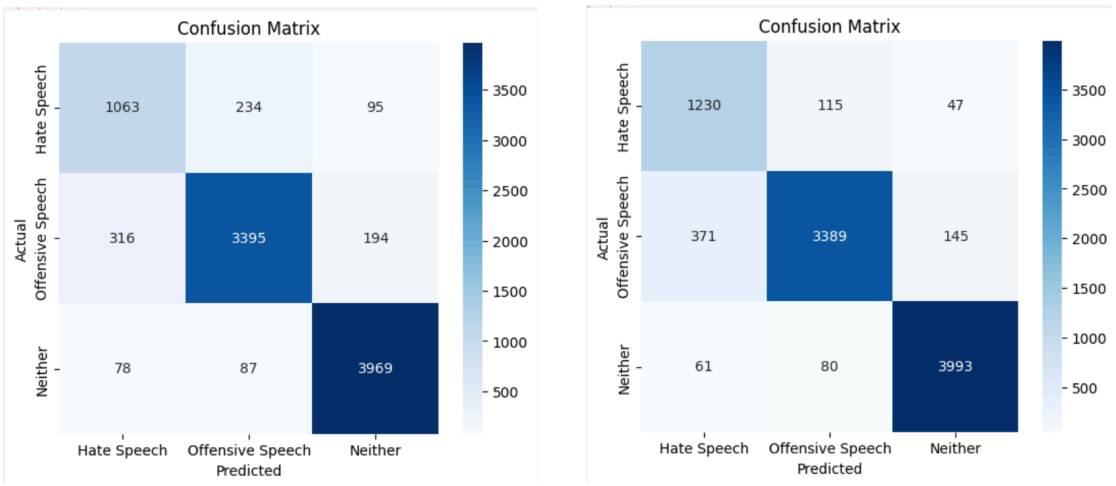
These results emphasize that while traditional models are computationally efficient and yield high scores on majority classes, they **struggle with semantic complexity**, particularly in underrepresented classes.

6.2 Results Using Deep Learning Models

To overcome the limitations of traditional methods, deep learning models were trained on **GloVe word embeddings** and evaluated under both baseline and augmented settings.

6.2.1 CNN and Bi-LSTM

- **CNN (GloVe + CNN)**:
    - Macro F1: **0.86**, Weighted F1: **0.86**
    - Excelled at capturing local n-gram features
    - Limited in modeling longer-range context
- **Bi-LSTM (GloVe + Bi-LSTM)**:
    - Macro F1: **0.89**, Weighted F1: **0.91**
    - Showed better sequence modeling through bidirectional context
    - Still struggled to resolve class imbalance entirely, with hate class recall around **0.55–0.60**



(a) *(b)*

Fig. 6.1 Confusion Matrices for Deep Learning Models

These results validate the importance of sequential models in hate speech detection but also highlight the **need for stronger contextual understanding** and class rebalancing.

6.2.2 Transformer-Based Models

The use of transformer-based models significantly enhanced performance:

- **BERT + MLP**:
    - Macro F1: **0.91**, Weighted F1: **0.90**
    - Achieved the **highest recall (~0.85+)** across all classes
    - Effectively distinguished subtle linguistic nuances between hate and offensive speech

This aligns with existing research that shows transformer architectures excel in handling informal, context-rich text such as tweets. BERT's contextual embeddings offer superior depth in representation compared to traditional and sequential models.

6.3 Impact of Data Augmentation

Data augmentation techniques—**synonym replacement, random insertion, and random deletion**—led to noticeable performance gains, especially for the **hate speech class**.

- On Bi-LSTM and MLP models, macro F1-scores improved by **2–4%**, indicating better generalization.
- Augmentation helped reduce false negatives and increased recall on minority classes.

This confirms the value of **data-driven balancing techniques** for improving model sensitivity to rare but important instances of hate speech.

6.4 Summary and Comparative Analysis

The following table summarizes performance across all models:

Table 6.1: Comparative Analysis of Model Performance

	Macro			Weighted Avg		
Model	Precision	Recall	F1 score	Precision	Recall	F1 score
TF - IDF + SVM	0.69	0.64	0.66	0.86	0.87	0.86
TF - IDF + MLP	0.68	0.63	0.66	0.85	0.86	0.84
GloVe + CNN	0.66	0.73	0.69	0.88	0.85	0.86
GloVe + BiLSTM	0.67	0.75	0.69	0.88	0.84	0.86
BERT + MLP	0.75	0.72	0.74	0.90	0.90	0.90
Proposed						
TF - IDF + SVM	0.88	0.87	0.88	0.90	0.91	0.90
TF - IDF + MLP	0.93	0.95	0.94	0.95	0.95	0.95
GloVe + CNN	0.86	0.86	0.86	0.89	0.89	0.89
GloVe + BiLSTM	0.88	0.91	0.89	0.92	0.91	0.91
BERT + MLP	0.90	0.90	0.91	0.91	0.90	0.90

6.4.1 Conclusion of Results:

- **Best Performing Model**: **Proposed BERT + MLP**
- **Most Improved via Augmentation**: **TF-IDF + MLP and Bi-LSTM**
- **Key Takeaway**: Deep learning, especially with **contextual embeddings and data augmentation**, significantly enhances the system's ability to identify hate speech amidst informal, noisy, and class-imbalanced text.

7. CONCLUSION AND FUTURE SCOPE

This research explores hate speech detection on Twitter using traditional, deep learning, and transformer-based models. Classical models (SVM, Naïve Bayes, MLP) with TF-IDF features showed varied performance—MLP performed best with macro and weighted F1-scores of 0.94 and 0.95, while SVM struggled with recall, identifying less than 55% of hate speech tweets due to dataset imbalance and shallow features. Deep learning models (CNN, Bi-LSTM) with GloVe embeddings showed moderate gains. The highest performance came from transformer-based models, especially BERT + MLP, achieving a macro F1-score of 0.91 and weighted F1-score of 0.95. Contextual embeddings and data augmentation played a crucial role in improving class balance.

⚠️ Limitations

- **Imbalanced Dataset**: Minority class (hate speech) remains underrepresented, affecting recall.
- **Language Scope**: Focused only on English; lacks support for multilingual/code-mixed texts.
- **Short Context**: Tweets may lack context, especially with sarcasm or implicit bias.
- **Static Embeddings**: CNN/Bi-LSTM models use non-contextual GloVe embeddings.
- **No Multimodal Analysis**: Visual elements like memes and images are not analyzed.

🚀 Future Scope

- **Multimodal Detection**: Integrate image-text content (e.g., memes, videos) for richer understanding.
- **Multilingual & Code-Mixed Handling**: Develop systems using XLM-RoBERTa and transliteration for diverse linguistic input.
- **Real-Time & Efficient Deployment**: Optimize with lighter models (DistilBERT, TinyBERT) for edge/mobile use.
- **Dataset Expansion**: Curate larger, balanced, manually annotated datasets for better generalization.
- **Explainability & Ethical AI**: Use tools like LIME/SHAP for transparency and bias mitigation.
- **Cross-Platform Generalization**: Explore domain adaptation to apply models across platforms (Reddit, Facebook, etc.).

REFERENCES

[1] J. S. Malik, H. Qiao, G. Pang, and A. van den Hengel, “Deep learning for hate speech detection: A comparative study,” *Int. J. Data Sci. Analytics*, vol. 9, no. 2, 2024.

[2] T. Davidson, D. Warmsley, M. Macy, and I. Weber, “Automated hate speech detection and the problem of offensive language,” in *Proc. 11th Int. AAAI Conf. Web Social Media*, 2017, pp. 512–515.

[3] Twitter Help Center, “Hateful conduct policy,” [Online]. Available: https://help.x.com/en/rules-and-policies/hateful-conduct-policy

[4] Z. Waseem and D. Hovy, “Hateful symbols or hateful people? Predictive features for hate speech detection on Twitter,” in *Proc. NAACL Student Res. Workshop*, 2016, pp. 88–93.

[5] W. Warner and J. Hirschberg, “Detecting hate speech on the world wide web,” in *Proc. 2nd Workshop Lang. Social Media*, Jun. 2012, pp. 19–26.

[6] I. Kwok and Y. Wang, “Locate the hate: Detecting tweets against blacks,” in *Proc. 27th AAAI Conf. Artif. Intell.*, 2013, pp. 1621–1622.

[7] P. Burnap and M. L. Williams, “Cyber hate speech on Twitter: An application of machine classification and statistical modeling for policy and decision making,” *Policy Internet*, vol. 7, no. 2, pp. 223–242, Jun. 2015.

[8] Z. Waseem, “Are you a racist or am I seeing things? Annotator influence on hate speech detection on Twitter,” in *Proc. 1st Workshop NLP Comput. Social Sci.*, 2016, pp. 138–142.

[9] T. Davidson, D. Warmsley, M. Macy, and I. Weber, “Automated hate speech detection dataset,” GitHub, 2017. [Online]. Available: https://github.com/t-davidson/hate-speech-and-offensive-language

[10] L. Gao and R. Huang, “Detecting online hate speech using context aware models,” *arXiv preprint*, arXiv:1710.07395, 2017.

[11] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma, “Deep learning for hate speech detection in tweets,” in *Proc. 26th Int. Conf. World Wide Web Companion*, 2017, pp. 759–760.

[12] J. Ho Park and P. Fung, “One-step and two-step classification for abusive language detection on Twitter,” *arXiv preprint*, arXiv:1706.01206, 2017.

[13] Z. Zhang and L. Luo, “Hate speech detection: A solved problem? The challenging case of long tail on Twitter,” *Semantic Web*, vol. 10, no. 5, pp. 925–945, Sep. 2019.

[14] S. Kamble and A. Joshi, “Hate speech detection from code-mixed Hindi-English tweets using deep learning models,” *arXiv preprint*, arXiv:1811.05145, 2018.

[15] B. Wei, M. Gao, and Y. Xu, “Offensive language and hate speech detection with deep learning and transfer learning,” *ResearchGate*, 2021.

[16] F. Ahamad and M. Firdous, “Deep learning-based text classification for hate speech in online social networks,” Major Project Report, Jamia Millia Islamia, 2025.

[17] T. Joachims, “Making large-scale SVM learning practical,” in *Advances in Kernel Methods*, MIT Press, 1999, pp. 169–184.

[18] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: https://www.tensorflow.org/

[19] Z. Waseem and D. Hovy, “Predictive features for hate speech detection on Twitter,” in *Proc. NAACL-HLT Student Research Workshop*, 2016, pp. 88–93.

[20] N. Djuric et al., “Hate speech detection with comment embeddings,” in *Proc. 24th Int. Conf. World Wide Web Companion*, 2015, pp. 29–30.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. NAACL-HLT*, 2019.

[22] K. Clark et al., “ELECTRA: Pre-training text encoders as discriminators rather than generators,” in *Proc. ICLR*, 2020.

[23] F. Chollet et al., “Keras,” [Online]. Available: https://keras.io

[24] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, Sebastopol, CA, USA: O’Reilly Media, 2009.

[25] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[26] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation,” in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[27] H. Saleem, K. P. Dillon, S. Benesch, and D. Ruths, “A Web of hate: Tackling hateful speech in online social spaces,” *arXiv preprint*, arXiv:1709.10159, 2017.

[28] Y. Kim, “Convolutional neural networks for sentence classification,” *arXiv preprint*, arXiv:1408.5882, 2014.