UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning

(6CS012)

# Sarcasm Detection in Headlines using RNN, LSTM, and Word2Vec Embeddings

| | |
|---|---|
| Student Id | : 2329256 |
| Student Name | : Sanjeev Kumar Sah |
| Group | : L6CG3 |
| Tutor | : Mr. Aatiz Ghimire |
| Lecturer | : Ms. Sunita Parajuli |
| Date | : 12th May, 2025 |

## Abstract

The report focuses on developing a deep learning-based text classification model for sarcasm detection using the ‘Sarcasm Headlines Dataset’. The dataset contains headlines from sources like The Onion and HuffPost, labeled as sarcastic or non-sarcastic. The main objective of this project is to build models capable of accurately identifying sarcasm in textual data. The report outlines a complete workflow including data preprocessing steps such as text normalization, tokenization, stop word removal, and lemmatization. Then it compares various deep learning models including Simple RNN, LSTM, Bidirectional LSTM, and LSTM with pretrained Word2Vec embeddings. Each model integrates an embedding layer, recurrent layers, and dense output layers with sigmoid activation for binary classification. Training involves appropriate loss functions, optimizers, and early stopping techniques. Evaluation is carried out using accuracy, confusion matrix, and classification metrics. While several models perform well, the Bidirectional LSTM shows better accuracy. Then finally the report concludes by identifying limitations and suggesting directions for future enhancements in sarcasm detection and related natural language processing tasks.

## Table of Content

## Table of Figure

## 1. Introduction

The dataset of choice for this project is the “Sarcasm Headlines Dataset” sourced from online media platforms such as The Onion and HuffPost. It contains news headlines with the labels “sarcastic” (1) or “non-sarcastic” (0). The primary challenge in this text classification task is to identify whether a given headline conveys sarcasm, which often relies on subtle linguistic cues and contextual contradictions. Sarcasm detection is an important area of sentiment analysis and natural language understanding, especially in domains like social media monitoring, opinion mining, and conversational AI systems.

The aim of the task is to develop a deep learning-based text classification model that can accurately detect sarcasm in headlines.

The objectives of the task are:

- Preprocessing of headline text data to ensure consistency and suitability for training.
- Implementation of various deep sequential neural architectures including Simple RNN, LSTM, Bidirectional LSTM, and LSTM with pretrained Word2Vec embeddings.
- Training the models over a set number of epochs while monitoring training and validation performance.
- Evaluating model accuracy, loss, and classification metrics to determine the best-performing architecture.

The deep learning architecture used in this project consists of several essential components:

- Embedding Layer for converting tokens into dense vectors.
- Recurrent Layers such as Simple RNN, LSTM, and Bidirectional LSTM.
- Dropout Layers to prevent overfitting during training.
- Dense Layers with ReLU activation for non-linearity.
- Output Layer with Sigmoid activation for binary classification.

For training and optimization, the Adam optimizer, binary cross-entropy as the loss function, and accuracy as the main metric were used to assemble the models. Confusion matrices, classification reports, and loss/accuracy graphs were used to evaluate the model's performance.

This report contains an introduction providing backgrounds and goals for the task. The methodology covers data preparation and model architecture. The experiments present results and evaluations. And finally, conclusions summarize findings and future improvement.

## 2. Dataset

The dataset used in this project is “Sarcasm Headlines” which is published on Kaggle and compiled from “The Onion and HuffPost”. The dataset contains approximately 28620 news headlines which is labeled as “Sarcastic (1) or “non-sarcastic” (0).

The two key features of dataset are:

I. Headline (Text)
II. Is_sarcastic (Binary)

The list of preprocessing applied are:

- Lowercasing text
- Removing punctuation, numbers, and extra whitespaces
- Expanding contractions (e.g., "can't" → "cannot")
- Removing stopwords
- Lemmatization
- Tokenization using Keras
- Padding sequences for uniform length

## 3. Methodology

Before training the model, it is crucial to ensure that the data we are going to feed into deep learning network is clean and suitable to build a model. In this section we will explain step-by-step data cleaning and text preprocessing, model architecture of four different models used for sarcasm detection.

## 3.1. Text Preprocessing

The dataset contains three fields: “headline”, “is_sarcastic” and “article_link”. Since only the textual content and sarcasm label were relevant, the “article_link” column was dropped.

After keeping the relevant columns, we move onto data cleaning. Here we checked for null values and dropped any missing entries and removed duplicate headlines to avoid training bias.

We then created a function clean_text(text) that performs the following operations:

- Convert to lowercase: Standardizes all text to lowercase.
- Remove URLs: Removes web links using regular expressions.
- Remove user mentions and hashtags: Deletes “@” user and “# “symbols.
- Remove punctuation: Removes all punctuation marks using regex.
- Remove numbers: Eliminates all numeric digits.
- Remove extra whitespace: Collapses multiple spaces into one and trims the text.

Additional preprocessing functions included:

- Contraction Expansion: A directory “contraction_map” and a function expand_contractions(text) are used to replace common contractions with expanded forms. Example: *don't" → "do not"*, *"it's" → "it is".*
- *Lemmetization: A function lemmatize_text(text) was used to:*
    - *Tokenize text using word_tokenize()*
    - *Remove stopwords using NLTK English stopword list*
    - *Lemmatize tokens using WordNetLemmatizer.*

    *For example: "running", "ran", "runs" → "run".*

- Tokenization: Keras’ Tokenizer was used to convert cleaned text into sequences of integers. An Out-of-Vocabulary (OOV) token was included to handle unseen words during inference.

- Padding Sequences: We applied pad_sequences() to ensure all sequences have the same length for model input.
    - Padding type: post
    - Truncation type: post

Then we create train_test_split(): The dataset was split into 80% training and 20% testing to evaluate generalization on unseen headlines

## 3.2. Model Architecture

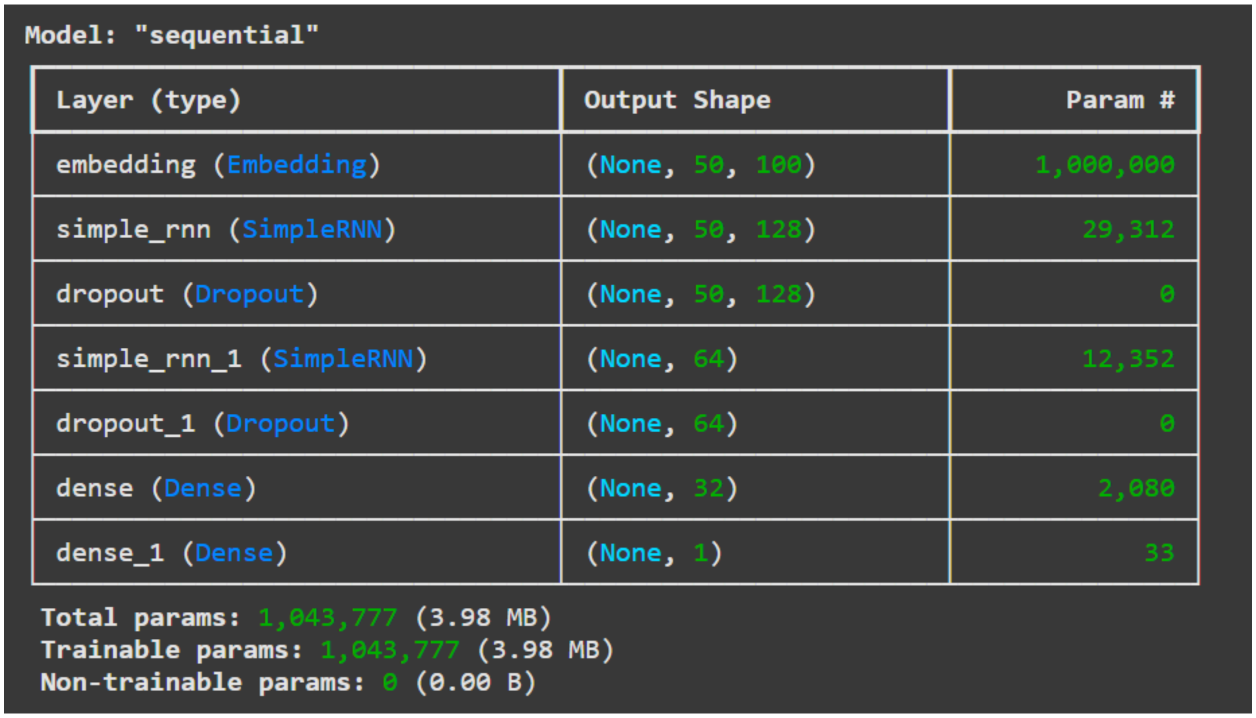
In this task we developed four models based on Recurrent Neural Network (RNN) architecture. All models are designed using the Sequential API from keras and include layers such as embedding, dropout, LSTM including bidirectional and dense layers.

1. Simple RNN

The first model in our experiment is a stacked Recurrent Neural Network (RNN) designed to process sequences of words in sarcastic headlines.

Model Architecture:

- Embedding Layer: Transforms each input word into a dense vector representation.
- SimpleRNN (128 units): First RNN layer processes the input sequence and returns full sequences for stacking.
- Dropout (0.3): Regularization to prevent overfitting.
- SimpleRNN (64 units): Second RNN layer condenses sequence output to a fixed-length vector.
- Dropout (0.3): Additional regularization.
- Dense (32 units, ReLU): Adds non-linearity before classification.
- Output Layer (Sigmoid): Predicts the probability of sarcasm.



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ (None, 50, 100)        │     1,000,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn (SimpleRNN)          │ (None, 50, 128)        │        29,312 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 50, 128)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn_1 (SimpleRNN)        │ (None, 64)             │        12,352 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            33 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,043,777 (3.98 MB)
 Trainable params: 1,043,777 (3.98 MB)
 Non-trainable params: 0 (0.00 B)
```

*Figure 1 Model Architecture Simple RNN*

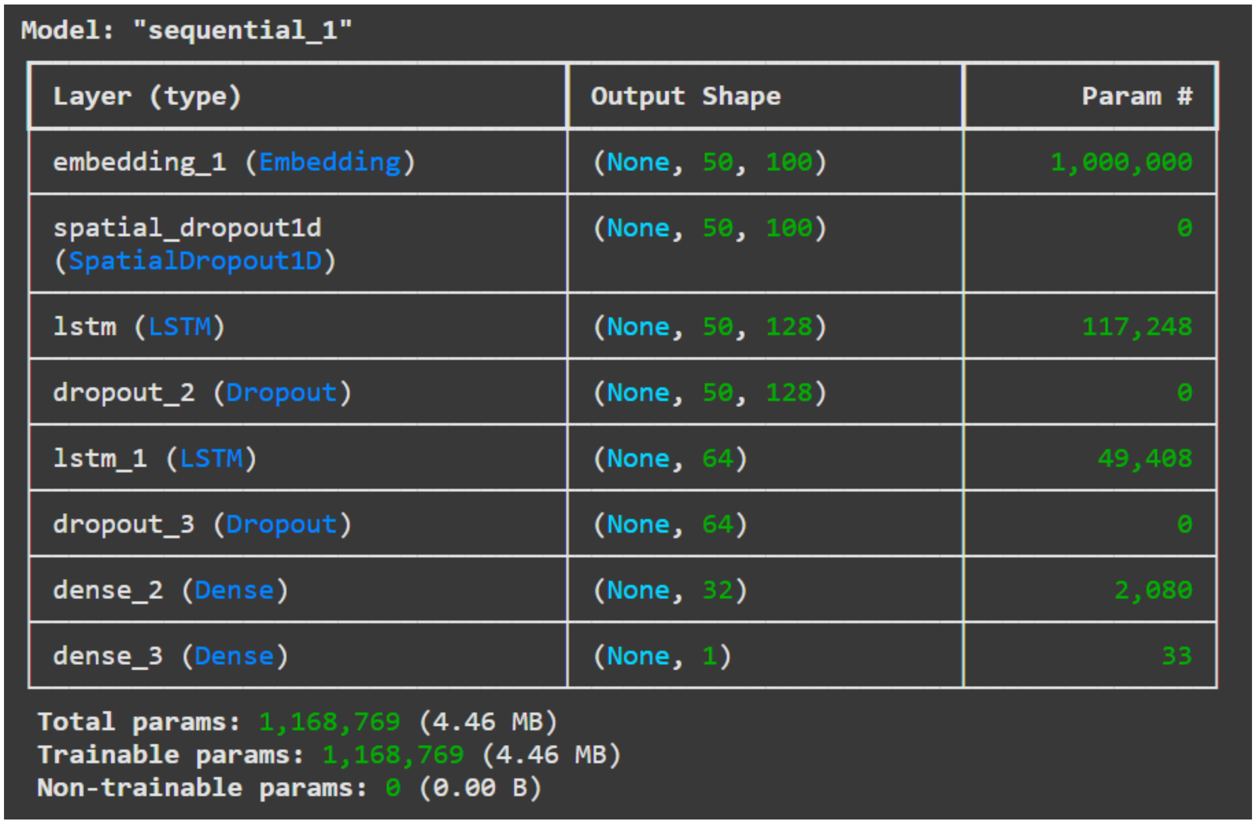
2. LSTM

The second model is based on Long Short-Term Memory (LSTM), a type of RNN specifically designed to overcome the vanishing gradient problem. LSTM networks retain information over long sequences using gating mechanisms, making them well-suited for sarcasm detection where contextual understanding is crucial.

Model Architecture:

- Embedding Layer: Transforms words into dense vectors.
- SpatialDropout1D (0.2): Regularization technique applied to embeddings to prevent overfitting.
- LSTM (128 units): First LSTM layer processes the sequence and returns full output to the next LSTM layer.
- Dropout (0.3): Randomly drops neurons to improve generalization.
- LSTM (64 units): Second LSTM layer reduces the sequence to a fixed-size vector.
- Dropout (0.3): Additional regularization layer.

- Dense (32 units, ReLU): Adds non-linearity and learns higher-level features.
- Output Layer (Sigmoid): Predicts the probability of sarcasm.



```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)         │ (None, 50, 100)        │     1,000,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ spatial_dropout1d               │ (None, 50, 100)        │             0 │
│ (SpatialDropout1D)              │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm (LSTM)                     │ (None, 50, 128)        │       117,248 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_2 (Dropout)             │ (None, 50, 128)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_1 (LSTM)                   │ (None, 64)             │        49,408 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_3 (Dropout)             │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │            33 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,168,769 (4.46 MB)
 Trainable params: 1,168,769 (4.46 MB)
 Non-trainable params: 0 (0.00 B)
```

*Figure 2 Model Architecture LSTM*

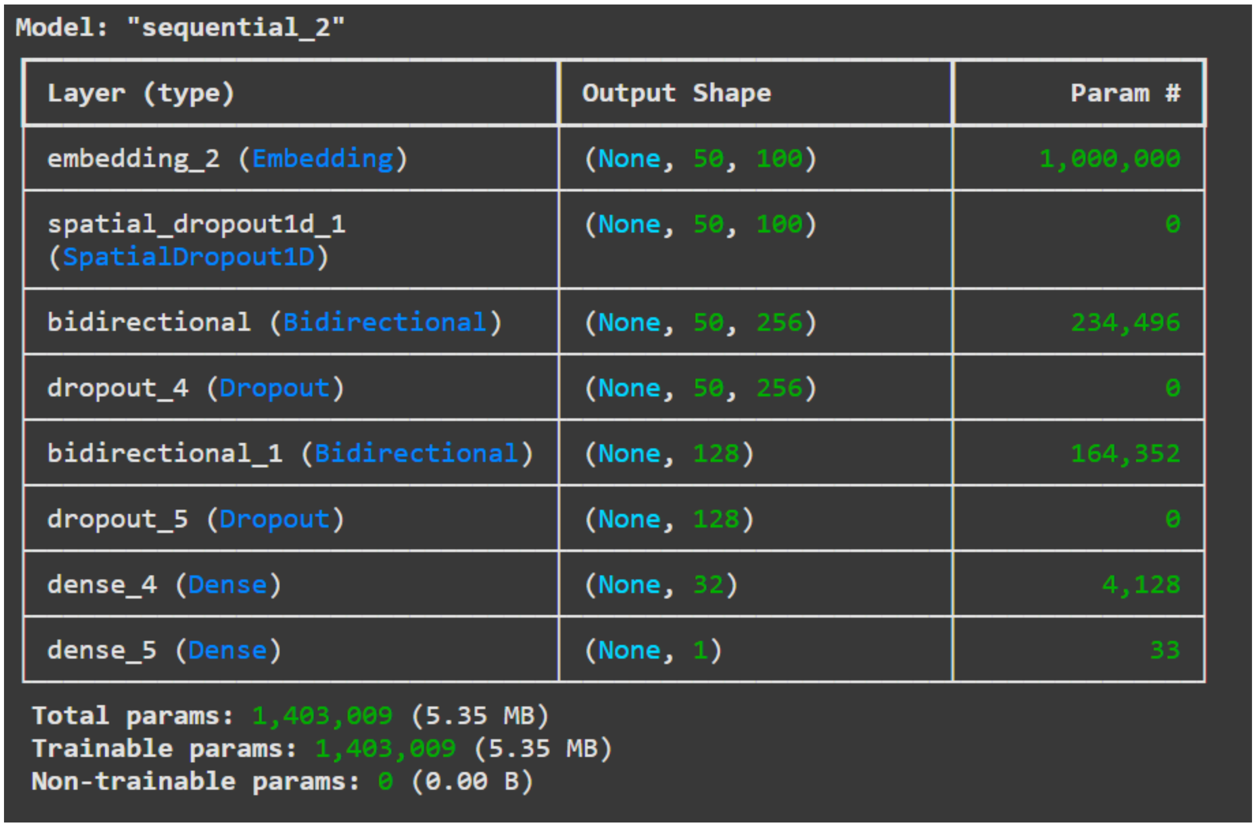
3. Bidirectional LSTM

The third model expands on LSTM by using Bidirectional LSTM layers, which read the input sequence in both forward and backward directions. This helps the model better understand the full context of each word, which is especially beneficial for detecting nuanced language like sarcasm.

Model Architecture:

- Embedding Layer: Converts input tokens into dense vectors.
- SpatialDropout1D (0.2): Applies dropout to the embedding vectors.
- Bidirectional LSTM (128 units): Processes sequence in both directions and returns full output.
- Dropout (0.3): Prevents overfitting after the first BiLSTM.

- Bidirectional LSTM (64 units): Second BiLSTM layer outputs a fixed-length representation.
- Dropout (0.3): Further regularization.
- Dense (32 units, ReLU): Introduces non-linearity.
- Output Layer (Sigmoid): Final binary classification.



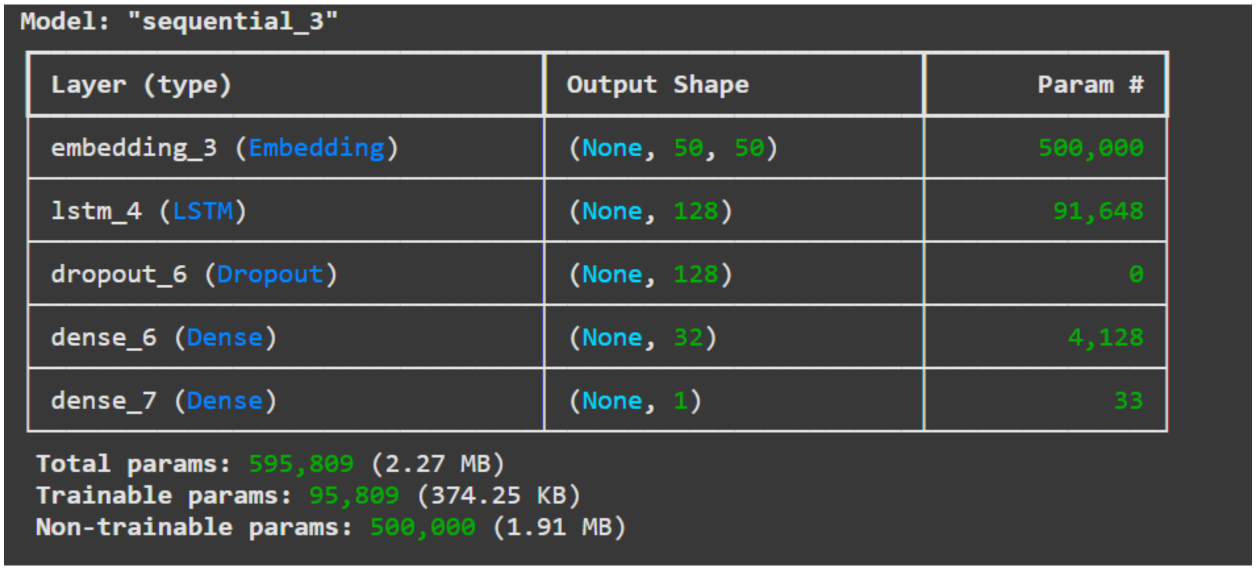
```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ embedding_2 (Embedding)          │ (None, 50, 100)         │      1,000,000 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ spatial_dropout1d_1              │ (None, 50, 100)         │              0 │
│ (SpatialDropout1D)               │                         │                │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ bidirectional (Bidirectional)    │ (None, 50, 256)         │        234,496 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dropout_4 (Dropout)              │ (None, 50, 256)         │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ bidirectional_1 (Bidirectional)  │ (None, 128)             │        164,352 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dropout_5 (Dropout)              │ (None, 128)             │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_4 (Dense)                  │ (None, 32)              │          4,128 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_5 (Dense)                  │ (None, 1)               │             33 │
└──────────────────────────────────┴─────────────────────────┴────────────────┘
 Total params: 1,403,009 (5.35 MB)
 Trainable params: 1,403,009 (5.35 MB)
 Non-trainable params: 0 (0.00 B)
```

*Figure 3 Model Architecture Bidirectional LSTM*

4. LSTM + Word2Vec Embeddings

The final model integrates pretrained Word2Vec (GloVe) embeddings with an LSTM architecture to enhance semantic understanding of text. Instead of learning word representations during training, it leverages GloVe vectors trained on a large external corpus (glove.6B.50d.txt), which helps capture rich relationships between words like “irony” and “sarcasm.”

- Embedding Layer (GloVe 50D, frozen): Loads pretrained word vectors from GloVe (glove.6B.50d.txt). The embedding matrix is set as non-trainable (trainable=False) to retain pretrained semantic structure.
- LSTM (128 units): Processes the sequence of word embeddings, retaining long-term dependencies to identify sarcastic cues.
- Dropout (0.3): Regularization layer that randomly drops neurons to reduce overfitting.
- Dense (32 units, ReLU): A fully connected layer that introduces non-linearity for improved learning of complex patterns.
- Output Layer (Sigmoid): Final binary classification output indicating whether the headline is sarcastic.



```
Model: "sequential_3"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_3 (Embedding) | (None, 50, 50) | 500,000 |
| lstm_4 (LSTM) | (None, 128) | 91,648 |
| dropout_6 (Dropout) | (None, 128) | 0 |
| dense_6 (Dense) | (None, 32) | 4,128 |
| dense_7 (Dense) | (None, 1) | 33 |

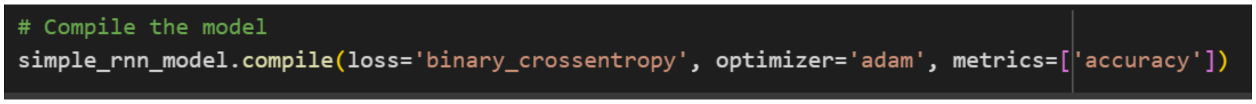
```
Total params: 595,809 (2.27 MB)
Trainable params: 95,809 (374.25 KB)
Non-trainable params: 500,000 (1.91 MB)
```

*Figure 4 Model Architecture LSTM+Word2Vec*

## 3.3. Loss Function

In our task is a Binary Classification (sarcastic vs. non-sarcastic), we used the Binary Cross-Entropy loss function. Because it is well-suited for models with a sigmoid output layer, which generates probabilities between 0 and 1.

This loss function measures the difference between predicted probabilities and actual binary labels. A lower Binary Cross-Entropy value indicates that the model's predictions are closer to the true labels. Binary Cross-Entropy is commonly used in text classification tasks because it penalizes incorrect predictions more heavily when the model is confident, leading to more stable and effective learning.



```
# Compile the model
simple_rnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

*Figure 5 Loss Function used*

## 3.4. Optimizer

We used the Adam optimizer to train all models. Adam (Adaptive Moment Estimation) is one of the most widely adopted optimization algorithms in deep learning due to its robustness and speed. It combines the benefits of two other methods, RMSprop and SGD, with momentum and adjusts learning rates adaptively for each parameter.

Advantages of Adam:

- Fast convergence during training.
- Adaptive learning rates are based on gradient history.
- Performs well with sparse and noisy gradients, especially important for natural language processing tasks involving large vocabularies.

## 3.5. Hyperparameters

The following key hyperparameters were used across all our deep learning models and significantly influenced their training behavior and performance:

- Learning Rate:
  The learning rate controls how quickly the model updates its weights during training. We used a value of 0.001, which allowed the model to converge smoothly without overshooting the optimal solution.
- Batch Size
  Batch size defines the number of samples the model processes before updating its weights. We used a batch size of 32, which provided a good balance between computational efficiency and model performance.
- Number of Epochs
  Each epoch represents a full pass through the training data. We trained the models for 5 to

10 epochs, using early stopping to avoid overfitting. Training was stopped if validation loss did not improve for 2 consecutive epochs.

## 4. Experiments and Results

### 4.1. RNN vs. LSTM Performance

To compare the effectiveness of sequential models for sarcasm detection, we implemented and evaluated three deep learning architectures: Simple RNN, LSTM, and Bidirectional LSTM. The models were assessed based on test accuracy, training/validation loss trends, and training time.

Accuracy Comparison

As shown in figure below the Simple RNN model achieved a test accuracy of 0.7042, the LSTM improved upon this with 0.7732, and the Bidirectional LSTM performed the best, reaching 0.7937. These results reflect the architectural strengths of LSTM-based models in capturing complex sequential patterns compared to a basic RNN.

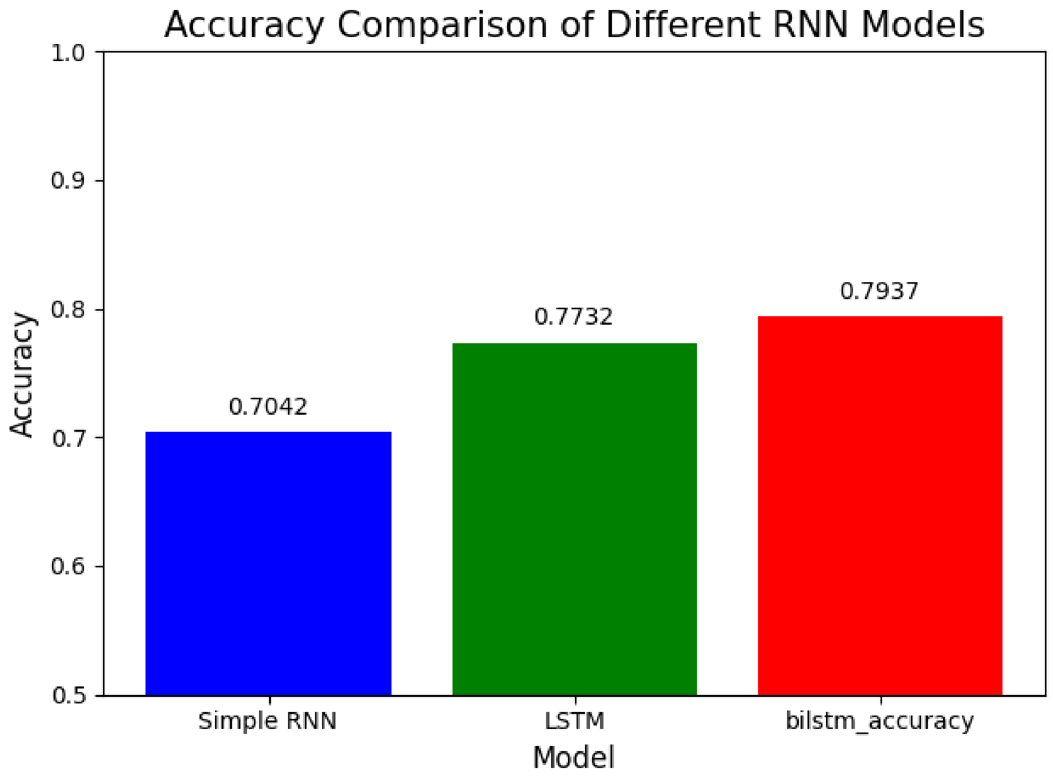


*Figure 6 Model Accuracy comparison*

Loss and Overfitting Trends

The training and validation loss curves provide insight into how well each model generalized during training.

- Simple RNN exhibited early signs of overfitting. After a few epochs, validation loss increased while training loss continued to decline, indicating a lack of generalization.

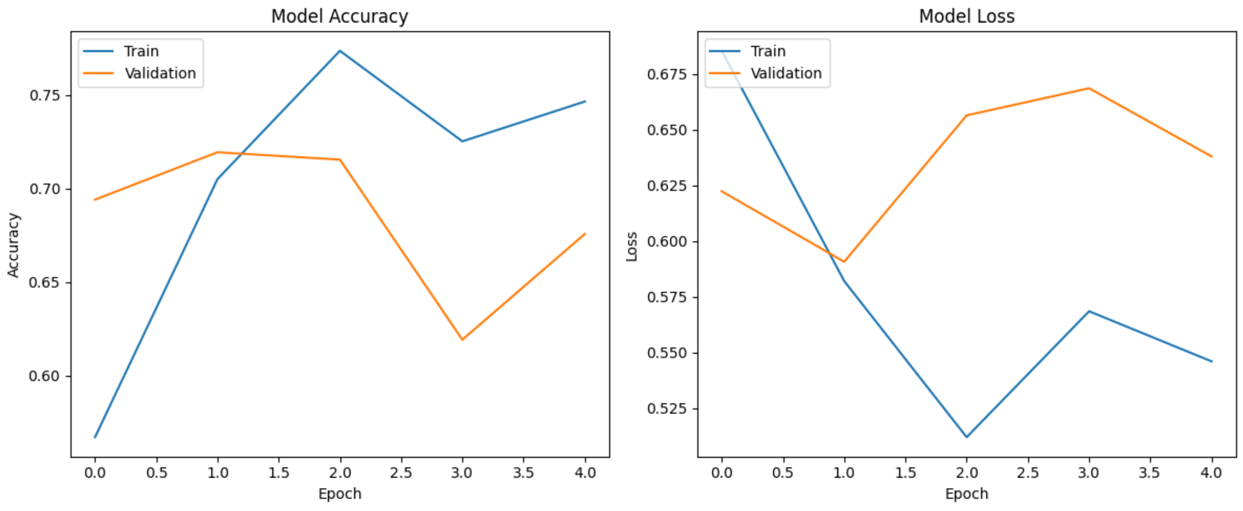


*Figure 7 Accuracy and Loss Simple RNN*

- The LSTM model demonstrated stable loss curves, with training and validation loss decreasing steadily across epochs. This reflects better learning and reduced overfitting due to the use of gating mechanisms.

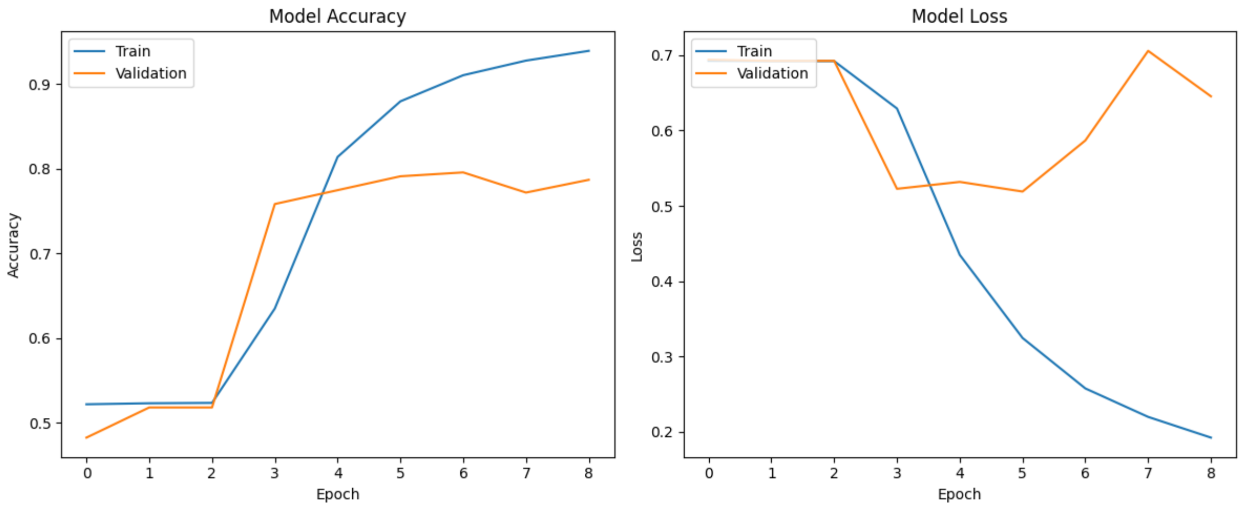


*Figure 8 Model Accuracy and Loss LSTM*

- The BiLSTM model had the most consistent and smooth learning curves, maintaining a small gap between training and validation loss. This suggests strong generalization and the ability to capture context from both directions in the sequence.

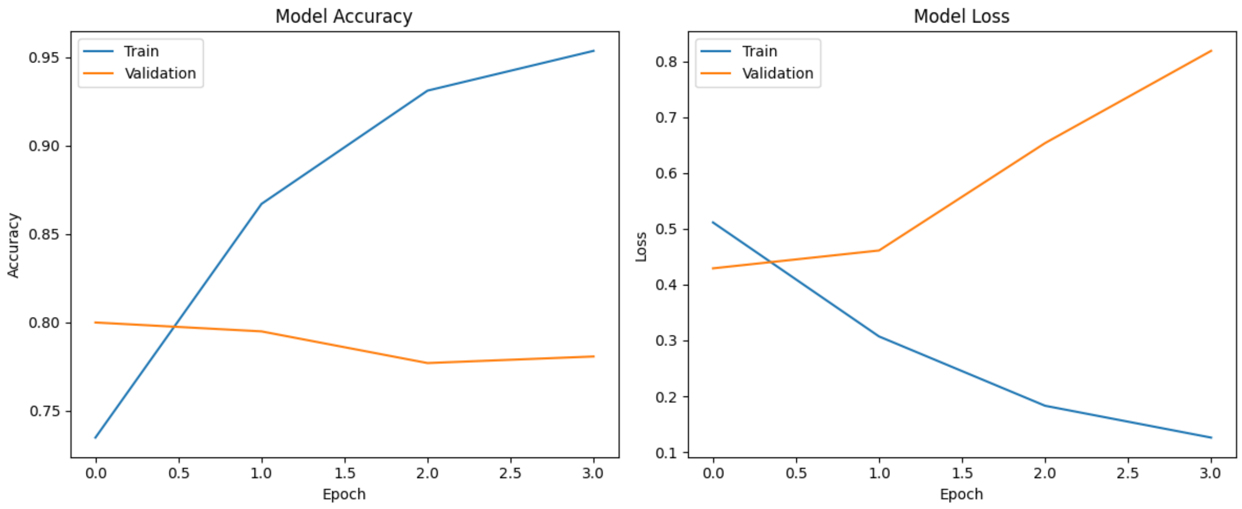


*Figure 9 Model Accuracy and Loss BiLSTM*

In terms of training time, the Simple RNN model was the fastest to train due to its lightweight architecture. The LSTM model required more time because of its gating mechanisms, and the Bidirectional LSTM was the slowest as it doubles the LSTM computation (forward + backward). Despite the additional cost, the performance gain from BiLSTM justified the training time.

## 4.2. Computational Efficiency

To evaluate the computational efficiency of each model, we analyzed training time, memory usage, and hardware resource needs. All experiments were conducted in “Google Collab”, with GPU acceleration enabled T4. This setup allowed for faster training, especially for models with more complex architectures like LSTM and Bidirectional LSTM. Among the four models tested, the “Simple RNN” was the most computationally efficient. It trained quickly due to its shallow structure and had the smallest number of parameters. Its lightweight design made it ideal for rapid experimentation, but it lacked the ability to model long-term dependencies effectively.

The LSTM model introduced additional complexity through its internal memory gates. While it required slightly more training time than the Simple RNN, it significantly improved performance and generalization. The increased parameter count made it more demanding in terms of memory and processing, but still manageable within Colab’s GPU limits. The Bidirectional LSTM was the most computationally expensive model. By processing sequences in both forward and backward directions, it effectively doubled the parameter count compared to the regular LSTM. This led to the longest training time and the highest memory usage, but the model consistently delivered the best performance. The LSTM + Word2Vec model used a pretrained, non-trainable embedding layer. This reduced the computational load during backpropagation and saved memory. However, it did not lead to performance gains due to vocabulary mismatch and the lack of task-specific adaptation, resulting in lower accuracy.

Across all models, training was completed within a few minutes per model using a batch size of 32, which balanced memory efficiency with training stability. GPU acceleration in Colab was essential for handling matrix operations in LSTM and BiLSTM layers, and no significant memory or timeout issues were encountered during training.

## 4.3. Training with Different Embeddings

This experiment evaluated how the choice of word embedding affected sarcasm detection performance. We compared two strategies:

- Random Embeddings: These embeddings are initialized randomly and updated during training, allowing them to adapt to the specific patterns in the dataset.

- Word2Vec Embeddings: We used 50 dimensional pretrained GloVe vectors from the glove.6B.50d.txt corpus. These embeddings were loaded into the model as a non-trainable layer, preserving their original semantic structure.

The models that used random, trainable embeddings specifically the LSTM and Bidirectional LSTM—consistently outperformed the model that used pretrained GloVe embeddings. The Bidirectional LSTM with random embeddings achieved the highest accuracy of approximately 0.7937, while the standard LSTM reached around 0.7732. In contrast, the LSTM + Word2Vec model performed poorly, achieving only around 50% accuracy.

This performance gap is primarily due to two factors. First, there was a limited vocabulary overlap between the sarcasm headlines dataset and the GloVe corpus, which reduced the usefulness of the pretrained vectors. Second, the frozen embedding layer (trainable=False) in the GloVe model prevented the embeddings from adapting to sarcasm-specific patterns in the data.

While pretrained embeddings are generally helpful in many NLP tasks, in this case, task-specific learned embeddings proved more effective. The results indicate that embeddings trained directly on the dataset provided better generalization and understanding of sarcastic language than general-purpose pretrained vectors.

## 4.4. Model Evaluation

To thoroughly assess model performance, we evaluated each model using key classification metrics: accuracy, precision, recall, F1-score, and the confusion matrix. These metrics offer deeper insight into how well the models classified sarcastic and non-sarcastic headlines, beyond overall accuracy.

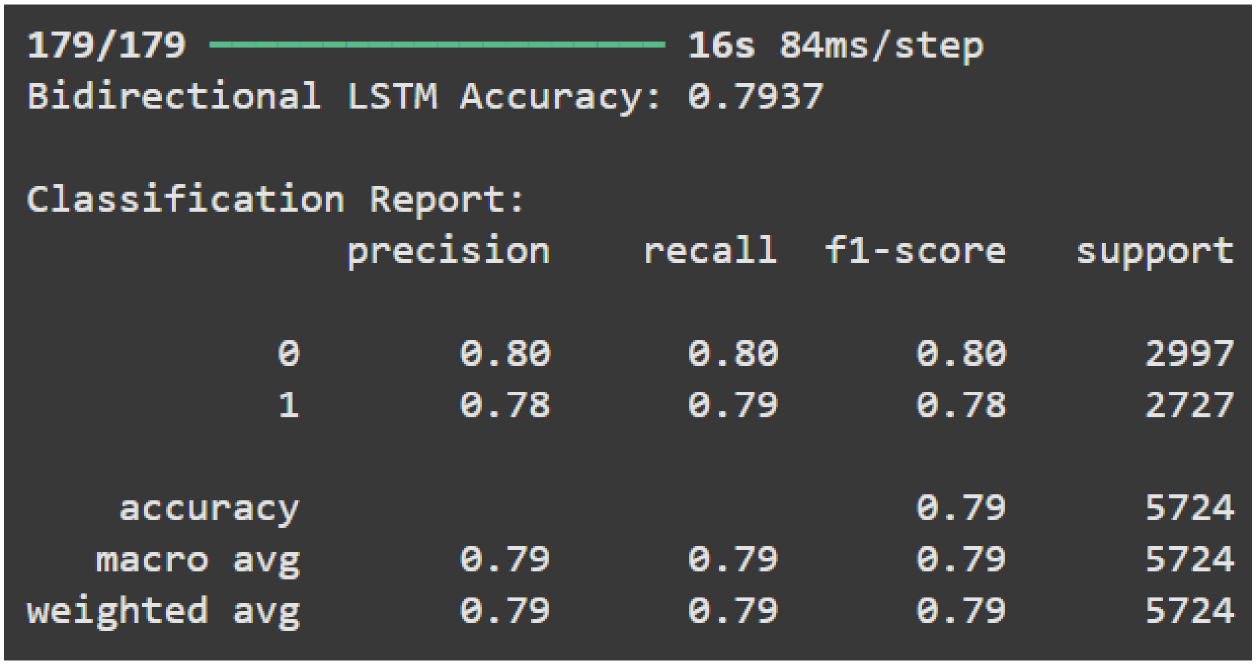
Among all models, the Bidirectional LSTM achieved the best performance, with the highest test accuracy of 0.7937. It also maintained a strong balance between precision and recall, making it the most reliable model for sarcasm detection in our experiments.

Precision, Recall, and F1-Score

The classification report for the Bidirectional LSTM is shown in Figure 6.5. From the label encoding, class 0 represents non-sarcastic headlines, while class 1 represents sarcastic headlines.

- For class 0 (non-sarcastic), the model achieved a precision of 0.80, recall of 0.78, and an F1-score of 0.79.

- For class 1 (sarcastic), the precision was 0.78, recall 0.81, and F1-score 0.79.
- The macro-average F1-score was 0.79, indicating consistent performance across both classes.



```
179/179 ━━━━━━━━━━━━━━━━━━━━ 16s 84ms/step
Bidirectional LSTM Accuracy: 0.7937

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80      2997
           1       0.78      0.79      0.78      2727

    accuracy                           0.79      5724
   macro avg       0.79      0.79      0.79      5724
weighted avg       0.79      0.79      0.79      5724
```

*Figure 10 Classification BiLSTM*

Confusion Matrix Analysis

To better understand the types of errors the model made, we examined the confusion matrix. The model correctly predicted:

- 2142 sarcastic headlines (True Positives)
- 2401 non-sarcastic headlines (True Negatives)

However, there were:

- 596 False Positives (non-sarcastic predicted as sarcastic)
- 585 False Negatives (sarcastic predicted as non-sarcastic)

These values show that while the model performs well overall, there were moderate misclassifications near decision boundaries, which is common in sarcasm detection due to its nuanced language.

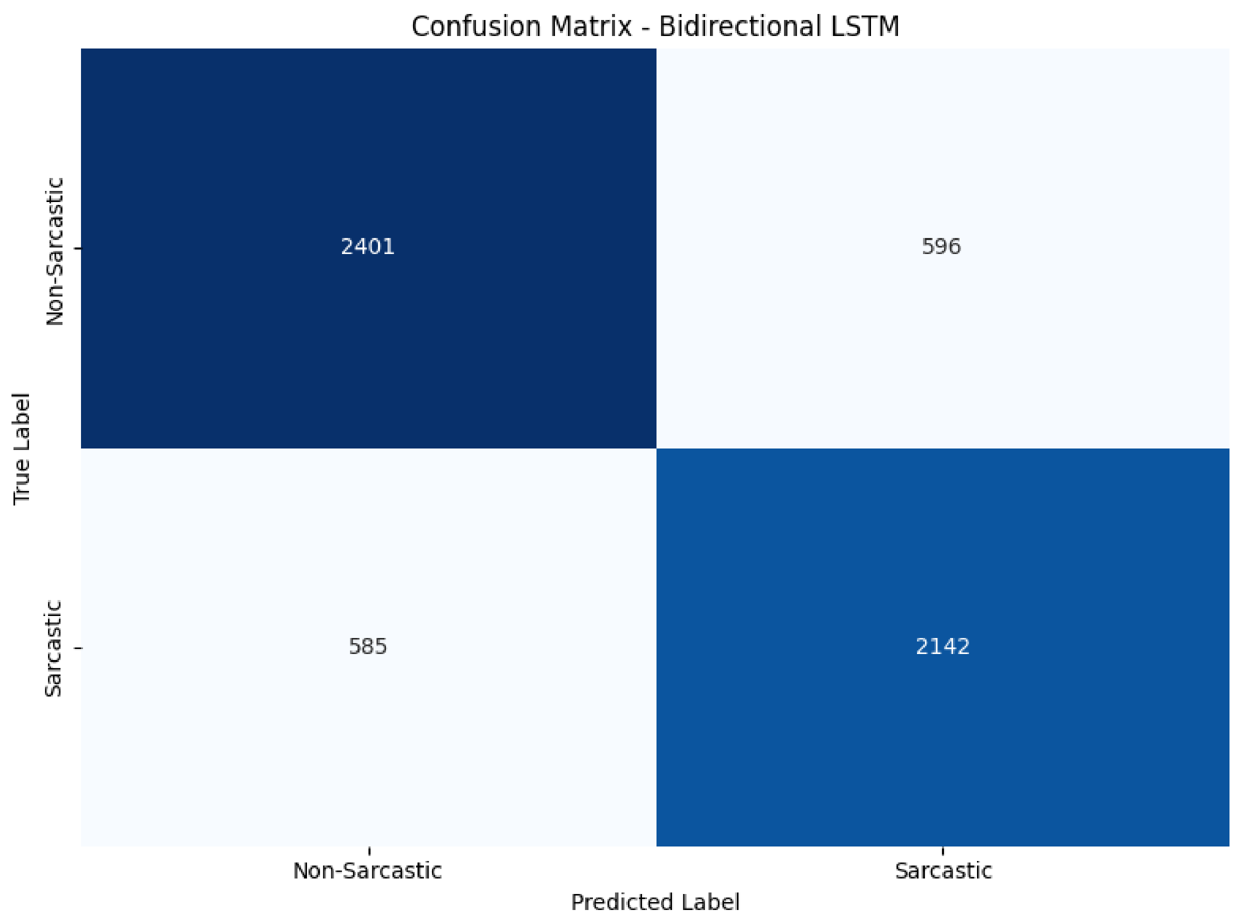


*Figure 11 Confusion Matrix BiLSTM*

## Accuracy and Loss Trends

During training, both training and validation accuracy increased steadily for all models. Bidirectional LSTM showed the most consistent upward trend, indicating stable learning. Additionally, the validation loss remained lower than training loss suggesting that the model generalized well without overfitting.

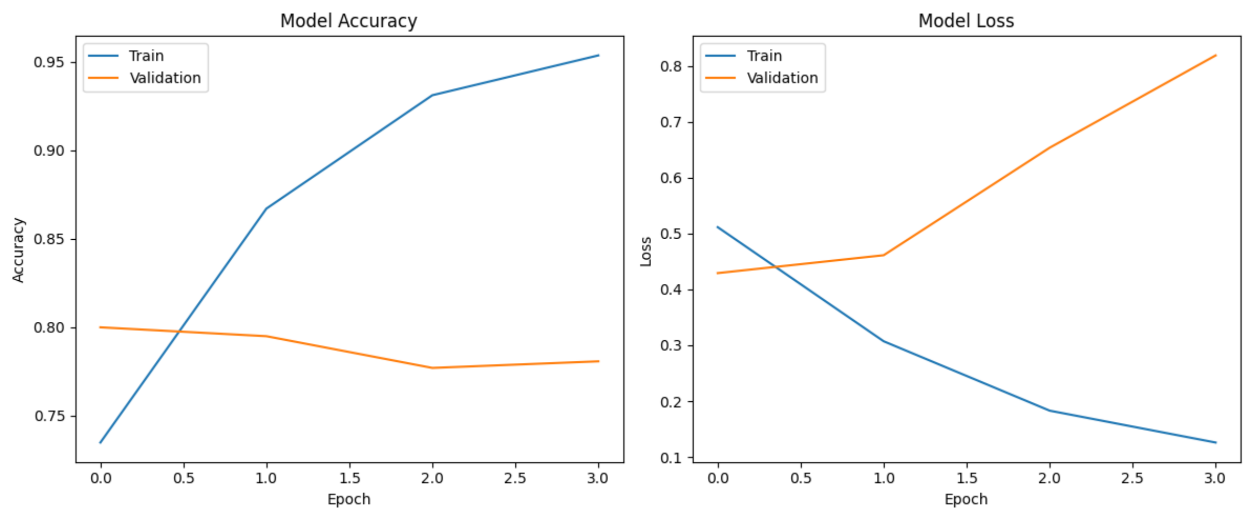


*Figure 12 Accuracy and Loss Trends*

## 5. Conclusion and Future Work

This project explored sarcasm detection in news headlines using deep learning models including Simple RNN, LSTM, Bidirectional LSTM, and LSTM integrated with pretrained Word2Vec embeddings. Models were evaluated using accuracy, precision, recall, F1-score, and confusion matrix metrics, alongside training time and loss trends.

Among all models, Bidirectional LSTM performed best with an accuracy of 0.7937, outperforming both Simple RNN (0.7042) and LSTM (0.7732). Its ability to capture context from both directions proved particularly useful for detecting sarcasm. In contrast, the LSTM + Word2Vec model underperformed (~0.5079 accuracy), likely due to vocabulary mismatch and the frozen embedding layer. Random, trainable embeddings tailored to the task proved more effective.

Evaluation metrics and learning curves confirmed the robustness of BiLSTM, with no major overfitting. However, some limitations were observed, including dataset size, Colab runtime constraints, and the inability to fine-tune pretrained embeddings.

Future Work

To improve the model and extend its applications:

- Explore hyperparameter tuning (e.g., grid search, Bayesian optimization)
- Use larger, more diverse datasets for better generalization

- Experience with transformer-based models like BERT or RoBERTa
- Integrate attention mechanisms for dynamic focus on key tokens
- Deploying the model as a real-time sarcasm detection tool using APIs or Gradio