Table Of Contents

1. Introduction to Movie Review Sentiment Analysis Using Deep Learning and Machine Learning Models:

- Sentiment Analysis using Deep Learning Models:

  Sentiment analysis is a natural language processing (NLP) task that involves the identification of opinions, attitudes, and emotions expressed in textual data. This technique is widely applied to assess customer feedback, product reviews, and social media posts. In this project, sentiment analysis is performed on movie reviews from the IMDB dataset to classify text as positive or negative. Traditionally, sentiment analysis is performed using rule-based approaches, but deep learning techniques offer a more advanced and accurate method for extracting meaningful patterns from complex, unstructured data.

- Role of Deep Learning in Sentiment Analysis:

  Deep learning models, especially neural networks, excel at identifying intricate patterns in large datasets. For sentiment analysis, models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have proven to be highly effective. These models are designed to handle sequential data, such as text, and are capable of learning dependencies across time steps, making them ideal for text classification tasks. LSTMs, in particular, address issues like the vanishing gradient problem by maintaining long-term memory, allowing the model to learn better contextual representations of text. Recurrent Neural Networks (RNNs) are a type of deep learning model designed for sequential data, where the output from the previous step influences the next step. This architecture is particularly suited for tasks like sentiment analysis, where understanding the sequence and context of words is crucial. LSTMs, a more advanced version of RNNs, are specifically built to capture long-term dependencies in the data, enabling the model to remember information for longer periods. These models can be trained on large datasets to identify sentiment with high accuracy, even when the sentiment is subtle or mixed.

2. Aim and Scope:

- Aim of the Project:

  The aim of this project is to develop an advanced sentiment analysis system using deep learning models, particularly a combined Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) architecture. The system will classify movie reviews from the IMDB dataset as either positive or negative. By combining RNN's ability to capture sequential dependencies with LSTM's strength in handling long-term memory, the model will achieve high accuracy in sentiment classification, even in complex or lengthy reviews. This system will be valuable for automating text analysis in various applications, including customer feedback, social media sentiment tracking, and market analysis.

- Scope of the Project:

  The scope of the project includes the following key components:

  1. Utilizing RNN for Sequential Data Processing
     The RNN will be employed to model the sequential nature of the text in the reviews. It will capture the immediate context of words and sentences, learning dependencies between words that help identify sentiment. The RNN model will process the IMDB dataset to perform basic sentiment classification tasks.

  2. Incorporating LSTM for Long-Term Dependency Learning
     The LSTM model will handle the long-term dependencies present in the text. It will help address the limitations of traditional RNNs by allowing the model to remember information over longer sequences. This is particularly useful for capturing sentiment in longer reviews where context from earlier parts of the text may be crucial for accurate classification.

  3. Combining RNN and LSTM for Enhanced Performance
     A hybrid architecture will be created by combining the RNN and LSTM layers. This will allow the model to benefit from both short-term sequential patterns (captured by RNN) and long-term dependencies (handled by LSTM). This combined model aims to improve the overall accuracy of sentiment classification, particularly in complex cases where both short and long-term context is important.

  4. Evaluating Model Performance
     The performance of the combined RNN + LSTM model will be evaluated using common metrics such as accuracy, precision, recall, and F1-score. A confusion matrix will also be used to assess the prediction quality and to identify any biases or weaknesses in classification.

5. Handling the IMDB Dataset
   The IMDB movie reviews dataset will be used for training and testing the sentiment analysis model. This dataset contains labelled reviews (positive or negative) and will be pre-processed—tokenized and vectorized—before being input into the neural network model.

- Research Paper Analysis and Objective

Objective: The objective of this project is to construct an advanced sentiment analysis system using a combined RNN and LSTM architecture to classify movie reviews into positive or negative categories. By combining the strengths of both models, the project aims to improve the accuracy and efficiency of sentiment classification, especially for more nuanced or complex reviews that require both short and long-term contextual understanding.

Key Components:

1. RNN (Recurrent Neural Networks)
   - Role: Processes sequential text data, capturing dependencies between adjacent words in a sentence.
   - Function: Helps the model learn local patterns and immediate context.
   - Strength: Effective for capturing short-term sequential dependencies in text data.
2. LSTM (Long Short-Term Memory Networks)
   - Role: Addresses the long-term dependency problem that traditional RNNs struggle with.
   - Function: Maintains memory over longer sequences of words, improving sentiment classification for longer reviews.
   - Strength: Ideal for handling complex sentiment analysis tasks where context over longer text spans is crucial.
3. Combined RNN + LSTM Hybrid Model
   - Role: Merges the advantages of both RNN and LSTM to process text data more effectively.
   - Function: The RNN layer captures immediate sequential dependencies, while the LSTM layer handles long-term memory and captures broader contextual information.
   - Strength: Provides a comprehensive approach for sentiment analysis by leveraging both local and long-term context.

3. Dataset and Experimental Setup:

- Dataset

  The dataset used in this project is the IMDB Movie Review Dataset, which contains 50,000 labelled movie reviews divided equally into positive and negative sentiment classes. It is a widely used benchmark dataset for binary sentiment classification tasks in Natural Language Processing (NLP).

  - Source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
  - Structure: The dataset is split into a training set (25,000 reviews) and a testing set (25,000 reviews), each containing balanced positive and negative reviews.
  - Purpose: To train models that can classify a given movie review as expressing either a *positive* or *negative* sentiment.

- Key Features

  Text Data: Raw movie reviews written by users, varying in length and vocabulary.

  Sentiment Labels:

  - 1 → Positive sentiment
  - 0 → Negative sentiment

  Preprocessing Applied:

  - Lowercasing
  - Removing punctuation
  - Removing stopwords
  - Lemmatization
  - Tokenization and padding (for LSTM)

  Input Format:

  - For Naive Bayes: Cleaned text is transformed using TF-IDF vectorization
  - For LSTM: Text is converted to sequences of integers and padded to a fixed length

Movie Review Sentiment Analysis

- Experimental Setup:

  1. Multinomial Naive Bayes Classifier:

     - Used as a classical baseline model for sentiment analysis.
     - TF-IDF vectorization is applied to convert text into numerical feature vectors.
     - The Naive Bayes model is trained on these vectors to classify sentiments.
     - Suitable for high-dimensional sparse data and performs well with word frequency-based features.

  2. LSTM (Long Short-Term Memory) Neural Network:

     - A sequential deep learning model used for understanding the context and order of words in the reviews.
     - Text is tokenized, padded, and then passed through an embedding layer followed by LSTM units.
     - LSTM is ideal for capturing long-range dependencies in text.
     - Final dense layers output the sentiment classification.

  3. Evaluation Metrics:

     - Accuracy
     - Classification Report (Precision, Recall, F1-Score)

- IDE Setup
  - Selected IDE: Jupyter Notebook / Google Colab (ideal for experimentation and visualization)
  - Python Version: Python 3.8+
  - Required Libraries:

    - numpy, pandas, matplotlib, seaborn
    - nltk, re, string
    - sklearn (for train-test split, evaluation)
    - keras, tensorflow (for LSTM model)
- Project Organization:
    - data – Contains raw and preprocessed data
    - notebooks – Jupyter notebooks for code and results
    - models – (Optional) Saved model weights
- Version Control: Optional Git repository initialized for tracking experiments

4. Methodology:

- LSTM:

Model Architecture:

1. Input Layer
    - Input Shape: A sequence of up to 200 integers (each representing a tokenized word from the review).
    - Input Description: These integers come from converting the review text into word indices using a tokenizer limited to the top 5000 most frequent words.
    - Purpose: Passes word index sequences into the network.

2. Embedding Layer:

    Embedding (input_dim=5000, output_dim=128, input_length=200)

    - What it does: Converts each integer (word index) into a dense 128-dimensional vector.
    - Output shape: (batch_size, 200, 128) — for every review of 200 tokens, each token is represented as a 128-length dense vector.
    - Purpose: Helps the model learn word representations (word embeddings) where semantically similar words have similar vectors.

3. LSTM Layer:

    LSTM (128, dropout=0.2, recurrent_dropout=0.2)

    - Type: A single LSTM (Long Short-Term Memory) layer with 128 memory units.
    - Input shape: (batch_size, 200, 128) (from the Embedding layer)
    - Output shape: (batch_size, 128) — The LSTM reads the full sequence and outputs a single 128-dimensional vector summarizing the review.
    - What it does:
    - Processes the sequence of embedded words step-by-step.
    - Remembers long-term dependencies (e.g., context of sentiment) using its internal memory and gating mechanisms.
    - dropout=0.2: Prevents overfitting by randomly dropping 20% of input units.
    - recurrent_dropout=0.2: Applies dropout to the LSTM's internal state transitions.

4. Dense Output Layer:

   Dense (1, activation="sigmoid")

   - Units: 1
   - Activation: Sigmoid
   - Output: A single value between 0 and 1 — interpreted as the probability that the review is positive.
   - Purpose: Binary classification (positive vs. negative sentiment).

Pseudo code:

```
# Define LSTM model architecture
model = Sequential()

# Add embedding layer
model.add(Embedding(input_dim=vocabulary_size, output_dim=128, input_length=max_length))

# Add LSTM layer
model.add(LSTM(units=128, return_sequences=False))

# Add output layer
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

- RNN:

1. Input Layer:

   - Input shape: A sequence of up to 200 integers, each representing a word index in a tokenized IMDB movie review.
   - Purpose: This layer accepts the processed sequences and forwards them into the network for further embedding and analysis.

2. Embedding Layer:

   Embedding (input_dim=5000, output_dim=128, input_length=200)

   - Function: Transforms each word index into a 128-dimensional dense vector representation.
   - Input: A sequence of integers with maximum length 200.
   - Output shape: (batch_size, 200, 128)
   - Purpose: Provides a learned dense representation for each word, allowing the model to capture semantic meaning and context.

3. First SimpleRNN Layer (Stacked RNN):

   SimpleRNN(128, return_sequences=True, dropout=0.15, recurrent_dropout=0.15)

   - Units: 128
   - Return sequences: True, so the layer outputs the full sequence of hidden states (one for each timestep).
   - Input shape: (batch_size, 200, 128) from the embedding layer.
   - Output shape: (batch_size, 200, 128)
   - Purpose: Processes the sequence of word embeddings and produces a hidden state at each time step, maintaining context through recurrence. The output at each timestep is passed to the next RNN layer.

4. Second SimpleRNN Layer:

   SimpleRNN(64, dropout=0.15, recurrent_dropout=0.15)

   - Units: 64

- Input shape: (batch_size, 200, 128) from the previous RNN layer.
- Output shape: (batch_size, 64)
- Purpose: Further condenses the sequence of outputs from the first RNN layer into a single 64-dimensional vector that captures contextual information across the entire review.

5. Dense Output Layer:

   Dense (1, activation='sigmoid')

   - Units: 1
   - Activation: Sigmoid
   - Output: A single value between 0 and 1 indicating the probability that the review is positive.
   - Purpose: Converts the high-level features from the RNN into a binary classification output.

Pseudo code:

```
Initialize weights W_xh, W_hh, W_hy and biases b_h, b_y
Initialize learning rate (alpha)

For each epoch:
  For each training example (X, y):
    Initialize hidden state h_0 = 0

    For t = 1 to T (length of sequence):
      Compute hidden state:
        h_t = tanh(W_xh * x_t + W_hh * h_(t-1) + b_h)

    Compute the output:
      y_hat = sigmoid(W_hy * h_T + b_y)

    Compute the loss (binary cross-entropy):
      Loss = - (y * log(y_hat) + (1 - y) * log(1 - y_hat))

    Perform backpropagation:
      Compute gradients dL/dW_xh, dL/dW_hh, dL/dW_hy, dL/db_h, dL/db_y

    Update weights:
      W_xh -= alpha * dL/dW_xh
```

W_hh -= alpha * dL/dW_hh
W_hy -= alpha * dL/dW_hy
b_h -= alpha * dL/db_h
b_y -= alpha * dL/db_y

Repeat for specified number of epochs

Return trained model

- LSTM and RNN combined:

1. Input Layer:

   - Input shape: Each input is a sequence of up to 200 integers (word indices) representing a pre-processed movie review.
   - Purpose: This layer receives the tokenized and padded input data and forwards it into the embedding layer.

2. Embedding Layer:

   Embedding (input_dim=5000, output_dim=128, input_length=200)

   - Function: Converts each word index in the input sequence into a 128-dimensional dense vector.
   - Input shape: (batch_size, 200)
   - Output shape: (batch_size, 200, 128)
   - Purpose: Provides a learned representation of words in a continuous vector space, capturing semantic similarity.

3. LSTM Layer:

   LSTM (128, return_sequences=True, dropout=0.2, recurrent_dropout=0.2)

   - Units: 128
   - Return sequences: True — this means the LSTM outputs the hidden state for every time step in the sequence.
   - Input shape: (batch_size, 200, 128) from the embedding layer.
   - Output shape: (batch_size, 200, 128)

- Purpose: Processes the sequence using Long Short-Term Memory units, which are designed to handle long-range dependencies and remember relevant information over longer sequences. It outputs a full sequence to feed into the following RNN layer.

4. SimpleRNN Layer:

   SimpleRNN (64, dropout=0.2, recurrent_dropout=0.2)

   - Units: 64
   - Input shape: (batch_size, 200, 128) — the output from the LSTM layer.
   - Output shape: (batch_size, 64)
   - Purpose: Takes the processed sequence from the LSTM and further refines the temporal representation using a simpler RNN. This stacking of LSTM and RNN combines the LSTM's long-term memory handling with the RNN's simpler dynamics.

5. Dense Output Layer:

   Dense (1, activation='sigmoid')

   - Units: 1
   - Activation: Sigmoid
   - Output: A single scalar between 0 and 1 representing the probability of the input review being positive.
   - Purpose: Maps the learned 64-dimensional feature vector into a binary sentiment prediction.

Pseudo code:

Initialize weights W_xh, W_hh, W_hy, W_xi, W_xf, W_xo, W_xc, W_hi, W_hf, W_ho, W_hc and biases b_h, b_y, b_i, b_f, b_o, b_c

Initialize learning rate (alpha)

For each epoch:

  For each training example (X, y):

    Initialize hidden state h_0 = 0

    Initialize cell state c_0 = 0

```
# Pass through the LSTM layer
For t = 1 to T (length of sequence):
    Compute LSTM gates (i_t, f_t, o_t, c_t):
        i_t = sigmoid(W_xi * x_t + W_hi * h_(t-1) + b_i)
        f_t = sigmoid(W_xf * x_t + W_hf * h_(t-1) + b_f)
        o_t = sigmoid(W_xo * x_t + W_ho * h_(t-1) + b_o)
        c_t = f_t * c_(t-1) + i_t * tanh(W_xc * x_t + W_hc * h_(t-1) + b_c)
        h_t = o_t * tanh(c_t)

# Pass through the RNN layer
For t = 1 to T (length of sequence):
    Compute RNN hidden state:
        h_t = tanh(W_xh * x_t + W_hh * h_(t-1) + b_h)

# Compute the output
y_hat = sigmoid(W_hy * h_T + b_y)

# Compute the loss (binary cross-entropy)
Loss = - (y * log(y_hat) + (1 - y) * log(1 - y_hat))

# Perform backpropagation (BPTT)
Compute gradients dL/dW_xh, dL/dW_hh, dL/dW_hy, dL/dW_xi, dL/dW_xf, dL/dW_xo, dL/dW_xc, dL/dW_hi, dL/dW_hf, dL/dW_ho, dL/dW_hc, dL/db_h, dL/db_y, dL/db_i, dL/db_f, dL/db_o, dL/db_c

# Update weights and biases
W_xh -= alpha * dL/dW_xh
W_hh -= alpha * dL/dW_hh
W_hy -= alpha * dL/dW_hy
W_xi -= alpha * dL/dW_xi
W_xf -= alpha * dL/dW_xf
```

W_xo -= alpha * dL/dW_xo

W_xc -= alpha * dL/dW_xc

W_hi -= alpha * dL/dW_hi

W_hf -= alpha * dL/dW_hf

W_ho -= alpha * dL/dW_ho

W_hc -= alpha * dL/dW_hc

b_h -= alpha * dL/db_h

b_y -= alpha * dL/db_y

b_i -= alpha * dL/db_i

b_f -= alpha * dL/db_f

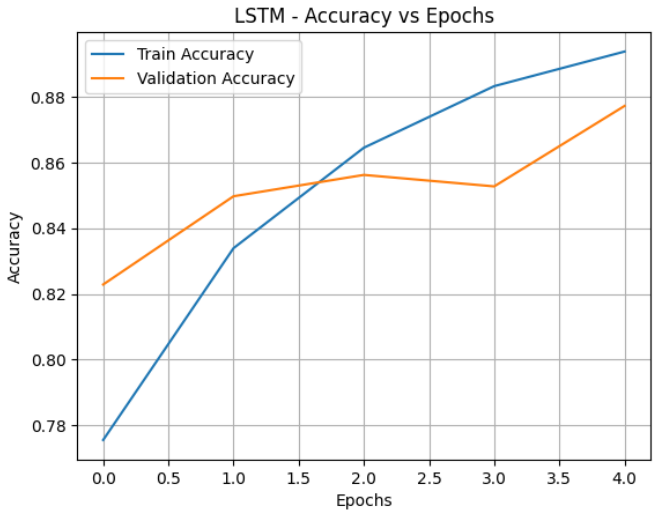
b_o -= alpha * dL/db_o

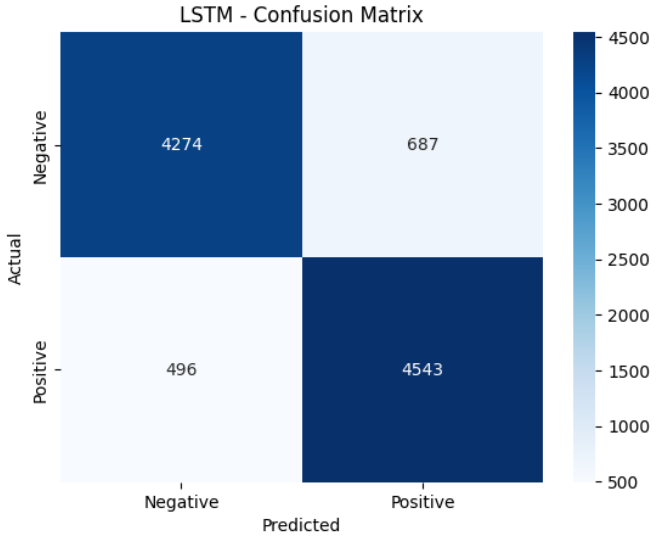
b_c -= alpha * dL/db_c

Repeat for specified number of epochs

Return trained model

5. Results:





- LSTM - Confusion Matrix

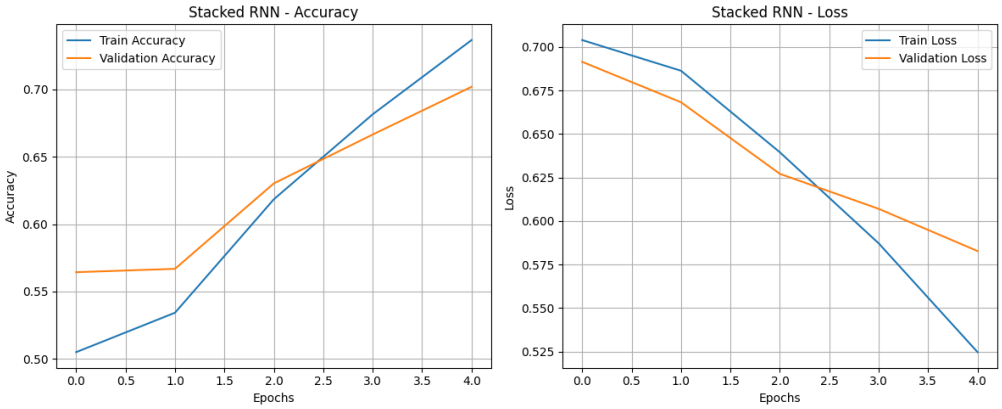
  True Negatives (TN): 4274, False Positives (FP): 687, False Negatives (FN): 496, True Positives (TP): 4543

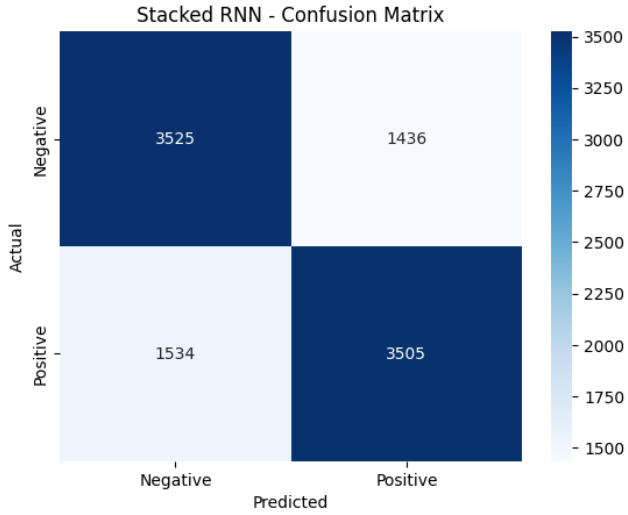
  Interpretation: LSTM performs the best among the three. It has the lowest false negatives and highest true positives, indicating strong sentiment classification capability, particularly in detecting positive reviews.

- LSTM - Accuracy vs Epochs

  Training Accuracy reaches ~0.895; validation ~0.88. Both curves are close, increasing consistently.

  Interpretation: LSTM demonstrates excellent learning and generalization. The smooth curve suggests minimal overfitting and a reliable performance on unseen data.





- Stacked RNN:

  Confusion Matrix: True Negatives (TN): 3525, False Positives (FP): 1436, False Negatives (FN): 1534, True Positives (TP): 3505

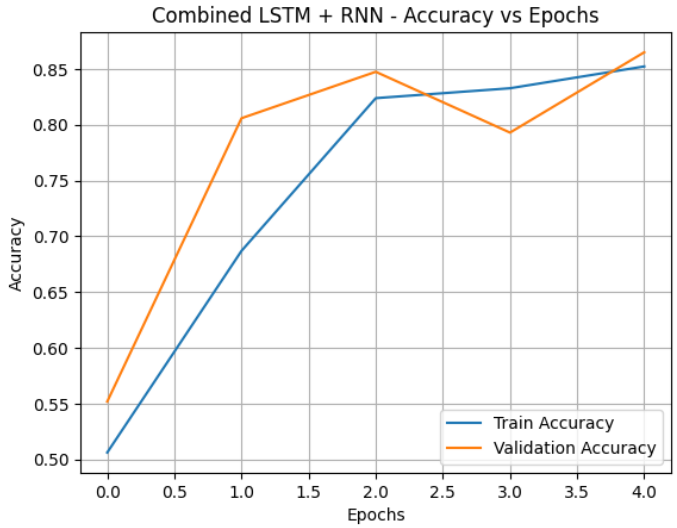
  Interpretation: The RNN model is weaker compared to others. It has higher misclassification, especially in detecting positive sentiments (high FN). This indicates that RNNs alone may struggle with capturing long-term dependencies in the reviews.

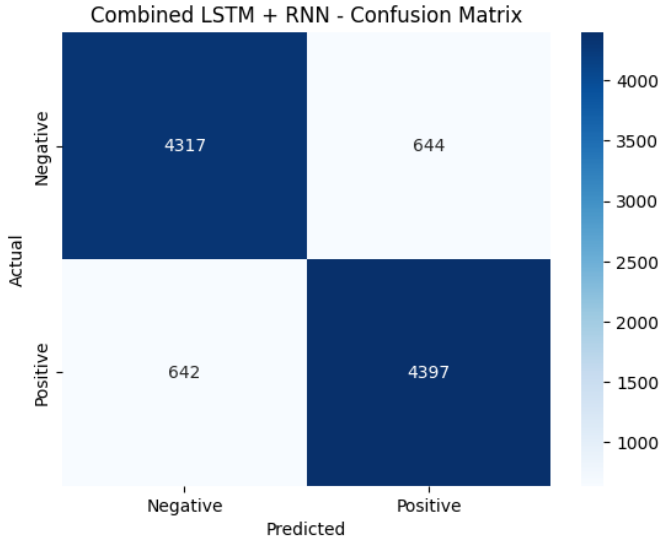
- Stacked RNN - Accuracy and Loss vs Epochs

  Accuracy improves gradually, reaching 0.71 (training) and 0.69 (validation).

  Loss steadily decreases, with a small gap between training and validation.

  Interpretation: The model improves over time but doesn't reach high accuracy. Still, the close curves suggest that it's not overfitting significantly—it simply underperforms compared to more advanced architectures.





- Combined LSTM + RNN - Confusion Matrix

  True Negatives (TN): 4317, False Positives (FP): 644, False Negatives (FN): 642, True Positives (TP): 4397

  Interpretation: This combined model performs well, showing a balanced classification performance with relatively low false predictions. The high TP and TN values indicate that the hybrid model is effective at recognizing both positive and negative sentiments.

- Combined LSTM + RNN - Accuracy vs Epochs

  Training Accuracy increases steadily to 0.86. Validation Accuracy spikes quickly to 0.85 by epoch 2, then shows slight fluctuation.

  Interpretation: The model learns fast and generalizes well without significant overfitting. It stabilizes early, indicating good convergence behaviour.

Movie Review Sentiment Analysis

Model Accuracy Summary:

LSTM standalone:	89.19%
RNN standalone:	70.59%
LSTM+RNN combined:	86.38%