# Abstract

This project presents a binary text classification model developed using **TensorFlow** and **Keras** to classify movie reviews from the **IMDB dataset** as either positive or negative. The dataset comprises 50,000 movie reviews, which are tokenized into sequences of word indices. The model leverages deep learning techniques to analyze the sentiment expressed in these reviews.

The methodology involves preprocessing the text data by padding/truncating sequences to a fixed length to ensure uniform input size. An **Embedding Layer** is used to convert word indices into dense vectors that capture semantic relationships between words. To capture the sequential nature of the text, a **Bidirectional Long Short-Term Memory (LSTM)** network is employed, allowing the model to learn contextual information in both forward and backward directions.

The architecture also includes **Dense Layers** for classification and **Dropout Layers** to prevent overfitting. The model is trained using the **Adam optimizer** and **binary cross-entropy loss** to optimize for accurate sentiment classification. The performance is evaluated using metrics such as accuracy, precision, recall, and F1-score.

The results are visualized through training and validation accuracy/loss curves, providing insights into model performance. This approach demonstrates how deep learning can be effectively applied to sentiment analysis and other text classification tasks, offering a reliable method for predicting sentiment in unstructured textual data.

# Objective

The primary objective of this project is to develop a robust **binary text classification model** using **TensorFlow** and **Keras** to accurately predict the sentiment (positive or negative) of movie reviews from the **IMDB dataset**. This involves:

1. **Preprocessing Text Data**:
    - Transform raw text data (movie reviews) into numerical representations suitable for input into a deep learning model.
    - Apply padding/truncation to ensure that all sequences have a uniform length.
2. **Building a Neural Network Model**:
    - Utilize an **Embedding Layer** to convert word indices into dense vectors that capture the semantic meaning of words.
    - Implement a **Bidirectional LSTM** to capture both past and future context of words in a review, allowing the model to understand the sequence dependencies in both directions.
    - Use **Dense Layers** with **Dropout** to perform classification and prevent overfitting during training.
3. **Training and Optimizing the Model**:
    - Train the model using the **Adam optimizer** and **binary cross-entropy loss** to efficiently minimize prediction errors and learn the underlying patterns in the data.
    - Regularly validate the model on unseen data (test set) during training to ensure generalization and avoid overfitting.
4. **Evaluating the Model**:
    - Evaluate the performance of the model using metrics such as **accuracy**, **precision**, **recall**, and **F1-score** to determine its effectiveness in correctly predicting sentiment.
    - Provide insights into how well the model can generalize to unseen data (test set).
5. **Visualizing Model Performance**:
    - Plot the training and validation accuracy and loss curves to track the model's performance and identify any signs of overfitting or underfitting during training.

The ultimate goal is to create a text classification model that effectively predicts whether a movie review is positive or negative, using deep learning techniques to understand the sequential and semantic relationships in the text

# Introduction

Text classification is a fundamental task in natural language processing (NLP) that involves categorizing text into predefined categories. One common use case is sentiment analysis, where the goal is to determine the sentiment (positive or negative) expressed in a text, such as a movie review.

In this project, we are building a binary text classification model using TensorFlow and Keras to classify movie reviews from the IMDB dataset as either positive or negative. The IMDB dataset contains 50,000 reviews, pre-split into 25,000 training and 25,000 test examples. Each review is already tokenized and represented as a sequence of integers, where each integer corresponds to a word in the dataset's vocabulary.

The primary challenge in text classification is the sequential nature of text and how to transform this unstructured data into something that a machine learning model can process. In this approach, we:

1. Preprocess the text data by padding/truncating sequences to a fixed length.
2. Use an Embedding Layer to transform word indices into dense vectors that capture semantic information.
3. Leverage Long Short-Term Memory (LSTM) layers, specifically a Bidirectional LSTM, to capture the relationships and context of words in both directions of the sequence (forward and backward).
4. Build a fully connected network for classification with Dropout layers to prevent overfitting.

By training the model using binary cross-entropy loss and Adam optimizer, we aim to build an efficient text classification system that can generalize well to unseen data. After training, the model's performance is evaluated on the test dataset using accuracy, precision, recall, and F1-score metrics.

This comprehensive pipeline provides a solid foundation for text classification tasks, showcasing how deep learning models can be applied effectively to natural language understanding.

# Methodology for Text Classification with TensorFlow

**1. Problem Definition**

The task is to build a text classification model to predict whether a given movie review (in the IMDB dataset) is positive or negative. This is a binary classification problem where:

- Class 0: Negative review.
- Class 1: Positive review.

**2. Data Collection and Loading**

The dataset used is the IMDB movie review dataset, which is included in TensorFlow's keras.datasets. This dataset contains 50,000 movie reviews, with:

- 25,000 reviews for training.
- 25,000 reviews for testing. The dataset is already tokenized, meaning each review is represented by a sequence of integers corresponding to word indices.
- Steps:
    - Load the dataset using imdb.load_data().
    - Limit the vocabulary size to the top 10,000 most frequent words (num_words=10000).
    - Split data into training and testing sets.

**3. Data Preprocessing**

Text data needs to be preprocessed into a format that the neural network can understand:

- Padding/Truncating Sequences: Reviews have different lengths, so they are padded to a maximum sequence length of 120 tokens. This ensures that each input to the model is the same size.
- The function pad_sequences from Keras is used to perform this operation, ensuring that all sequences are of uniform length by adding padding (post) to shorter reviews and truncating longer ones.

**4. Model Design**

The architecture of the model is designed to capture the semantics of the text and make accurate predictions.

Layers and Components:

- Embedding Layer: Converts each word in the sequence into a dense vector of fixed size (embedding dimension = 128). The embedding layer captures relationships between words based on the training data.

- Bidirectional LSTM Layer: Long Short-Term Memory (LSTM) layers are used to capture sequential dependencies in the data. A Bidirectional LSTM allows the model to learn from both the forward and backward context of the sequence. This helps in capturing both past and future information for each time step.
    - The first LSTM has 64 units and outputs sequences, which allows stacking more layers on top.
    - A Dropout layer is added after the LSTM to prevent overfitting by randomly dropping some connections during training.
    - A second LSTM layer with 32 units follows to further refine the features.
- Dense Layers: Fully connected layers are used for classification.
    - The first dense layer has 64 units with ReLU activation to introduce non-linearity and reduce dimensionality.
    - A second Dropout layer is added to reduce overfitting.
    - The final dense layer has 1 unit with a sigmoid activation function. This layer outputs a probability score between 0 and 1, which is interpreted as the likelihood of the review being positive.

**5. Model Compilation**

The model is compiled with:

- Adam optimizer: An adaptive learning rate optimization algorithm that is efficient for large datasets and high-dimensional spaces.
- Binary Cross-Entropy loss function: Since this is a binary classification problem, this loss function is appropriate for measuring the error between predicted and actual labels.
- Accuracy metric: Used to evaluate model performance on training and validation data.

**6. Model Training**

The model is trained on the training dataset with:

- 5 epochs: The entire dataset is passed through the network five times.
- Batch size of 64: The model is updated after processing 64 samples at a time. This helps in efficiently utilizing the GPU/CPU for faster training.
- Validation data: The model is evaluated on the test data during training to monitor generalization and check for overfitting.

**7. Model Evaluation**

After training, the model is evaluated on the test dataset to measure how well it generalizes to unseen data. The following metrics are used:

- Accuracy: The fraction of correct predictions out of total predictions.

- Test Loss: Measures how well the model is predicting the labels.

Additionally, the model's predicted outputs are used to generate:

- Classification Report: A detailed performance report using sklearn's classification_report() function, which includes metrics like precision, recall, and F1-score for both the positive and negative classes.

**8. Prediction**

The model makes predictions on the test dataset. Since the sigmoid activation function is used in the output layer, the predictions are probabilities. A threshold of 0.5 is used to convert these probabilities into class labels (0 or 1).

- Predicted Probability > 0.5: Positive review.
- Predicted Probability ≤ 0.5: Negative review.

**9. Result Visualization**

To assess the training process visually,  two graphs are plotted:

- Training and Validation Accuracy: Shows how the model's accuracy evolves over the epochs. If the validation accuracy increases steadily without diverging from training accuracy, the model is not overfitting.
- Training and Validation Loss: Helps in understanding whether the model's performance is improving over time. A  decreasing loss curve indicates that the model is learning.

# Code

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.datasets import imdb
from sklearn.metrics import classification_report
```

[4] ✓ 0.0s

```
# Step 1: Load and preprocess the IMDB dataset
# Load the IMDB dataset (binary sentiment classification)
vocab_size = 10000  # Use the top 10,000 most frequent words
max_length = 120    # Maximum sequence length (truncated/padded)
embedding_dim = 128 # Embedding dimension for word vectors
oov_tok = "<OOV>"  # Token for out-of-vocabulary words

# Load dataset from Keras, setting a vocabulary limit
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
```

[5] ✓ 7.3s

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
# Step 2: Padding the sequences to have consistent input length
# Pad training and testing data
x_train_padded = pad_sequences(x_train, maxlen=max_length, padding='post', truncating='post')
x_test_padded = pad_sequences(x_test, maxlen=max_length, padding='post', truncating='post')
```

[6] ✓ 0.3s

+ Code + Markdown

```
# Step 3: Define the text classification model using TensorFlow
# Initialize the model
model = Sequential()

# Embedding Layer: Converts word indices to dense vectors of a fixed size
model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))

# LSTM Layer: Adds Long Short-Term Memory (LSTM) network to capture sequential data
model.add(Bidirectional(LSTM(64, return_sequences=True)))  # Bidirectional LSTM for capturing context in both directions
model.add(Dropout(0.5))  # Regularization to prevent overfitting
model.add(LSTM(32))      # Another LSTM layer

# Dense Layers: Fully connected layers for classification
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))  # Additional dropout layer
model.add(Dense(1, activation='sigmoid'))  # Sigmoid activation for binary classification
```

✓ 0.0s

```python
# Step 4: Compile the model
# Adam optimizer and binary crossentropy loss for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model Summary
model.summary()
```

[8] ✓ 0.0s

**Model: "sequential_2"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_2 (Embedding) | ? | 0 (unbuilt) |
| bidirectional (Bidirectional) | ? | 0 (unbuilt) |
| dropout (Dropout) | ? | 0 (unbuilt) |
| lstm_3 (LSTM) | ? | 0 (unbuilt) |
| dense_4 (Dense) | ? | 0 (unbuilt) |
| dropout_1 (Dropout) | ? | 0 (unbuilt) |
| dense_5 (Dense) | ? | 0 (unbuilt) |

**Total params:** 0 (0.00 B)

**Trainable params:** 0 (0.00 B)

**Non-trainable params:** 0 (0.00 B)

```python
# Step 5: Train the model
# Train the model with training data and validate on test data
history = model.fit(x_train_padded, y_train, epochs=5, batch_size=64, validation_data=(x_test_padded, y_test), verbose=1)
```

[9] ✓ 3m 55.4s

```
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 50s 119ms/step - accuracy: 0.6576 - loss: 0.5982 - val_accuracy: 0.8156 - val_loss: 0.4270
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 45s 116ms/step - accuracy: 0.8703 - loss: 0.3408 - val_accuracy: 0.8156 - val_loss: 0.4476
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 118ms/step - accuracy: 0.8959 - loss: 0.2820 - val_accuracy: 0.8163 - val_loss: 0.4486
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 47s 119ms/step - accuracy: 0.9298 - loss: 0.1993 - val_accuracy: 0.7980 - val_loss: 0.5371
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 47s 121ms/step - accuracy: 0.9317 - loss: 0.1977 - val_accuracy: 0.8042 - val_loss: 0.5328
```

```python
# Step 6: Evaluate the model
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test_padded, y_test, verbose=2)
print('Test Accuracy:', test_acc)
```

[14] ✓ 11.9s

```
782/782 - 12s - 15ms/step - accuracy: 0.8042 - loss: 0.5328
Test Accuracy: 0.8042399883270264
```

```
# Step 7: Generate classification report
y_pred = (model.predict(x_test_padded) > 0.5).astype("int32")  # Predict classes using a threshold of 0.5
print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))
```

[15] ✓ 13.0s

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step
              precision    recall  f1-score   support

    Negative       0.77      0.86      0.81     12500
    Positive       0.84      0.75      0.79     12500

    accuracy                           0.80     25000
   macro avg       0.81      0.80      0.80     25000
weighted avg       0.81      0.80      0.80     25000
```

# Conclusion

In this project, a deep learning-based text classification model was successfully developed using **TensorFlow** and **Keras** to classify movie reviews from the **IMDB dataset** as positive or negative. The model effectively captured the sequential dependencies in the text using a **Bidirectional LSTM** network and represented words as dense vectors using an **Embedding Layer**.

The preprocessing steps, including padding and truncating sequences, ensured uniform input lengths, allowing the neural network to process reviews of varying lengths. With the inclusion of **Dropout layers** to prevent overfitting and the **Adam optimizer** for efficient learning, the model achieved strong generalization on unseen data, as evidenced by the high accuracy and other performance metrics on the test set.

The visualization of training and validation accuracy and loss provided insights into the model's learning process, highlighting areas where overfitting or underfitting might occur. By leveraging a sequential model architecture with LSTM layers, the project demonstrated how deep learning can effectively handle sentiment analysis tasks by capturing both the semantic meaning and contextual information of words in a sequence.

In conclusion, this project illustrates the power of neural networks, especially **LSTM** architectures, in understanding and classifying textual data, making it a valuable approach for tasks such as sentiment analysis, document classification, and other text-based predictions. The model's adaptability allows it to be further expanded to more complex NLP tasks, making it a versatile solution for a variety of text classification problems