Sentiment Analysis Using LSTM & GRU on the IMDB Dataset

By Mahmoud AbuAwd

1. Introduction

This report presents the implementation of a Long Short-Term Memory (LSTM) & GRU modelS for sentiment analysis using the IMDB movie review dataset. The objective was to classify reviews as either "positive" or "negative" based on the text content. The model utilizes deep learning techniques to learn from the training data and evaluate its performance on unseen test data.

2. Methodology

2.1. Data Preparation

- **Dataset**: The dataset consists of IMDB reviews from kaggle : https://www.kaggle.com/code/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews/input?select=IMDB+Dataset.csv
- Where each review is labeled as either "positive" or "negative". The dataset was read using pandas:

```
df = pd.read_csv('/content/IMDB Dataset.csv')
```

- **Data Exploration**: Initial exploration checked for missing values and unique sentiment labels. The dataset shape was verified to understand its dimensions:

```
df.isna().sum()
df.head()
df.sentiment.unique()
```

```
print(df.shape)
```

- **Data Splitting**: The dataset was split into training and testing subsets using an 80-20 split:

```
train_df, test_df = train_test_split(df, test_size=0.2,
random_state=42)
```

2.2. Text Processing

- **Tokenization and Padding**:
  - A `Tokenizer` was created to convert text reviews into sequences of integers. The vocabulary size was set to 10,000.
  - The sequences were padded to a maximum length of 120 using `pad_sequences` to ensure uniform input size:

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oo
v_tok)
tokenizer.fit_on_texts(train_df['review'])
train_sequences = tokenizer.texts_to_sequences(train_df
['review'])
train_padded = pad_sequences(train_sequences, maxlen=max
_length, truncating=trunc_type)
```

- **Label Encoding**: Sentiment labels were converted to numerical values (1 for positive, 0 for negative):

```
train_labels = np.array([1 if sentiment == 'positive' el
se 0 for sentiment in train_df['sentiment']])
```

2.3. Model Architecture For LSTM

- The LSTM model architecture comprised:
  - An embedding layer for converting integer sequences into dense vectors.
  - A single LSTM layer with 64 units.
  - A dense layer with 24 units followed by a dropout layer for regularization.
  - An output layer with a sigmoid activation function for binary classification:

```
model_lstm = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
input_length=max_length),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

- **Compilation**: The model was compiled with binary cross-entropy loss and the Adam optimizer:

```
model_lstm.compile(loss='binary_crossentropy', optimizer
='adam', metrics=['accuracy'])
```

2.4. Training

- The model was trained for 10 epochs with an early stopping callback to prevent overfitting:

```
early_stopping = tf.keras.callbacks.EarlyStopping(monito
```

```
r='val_loss', patience=3, restore_best_weights=True)

history = model_lstm.fit(train_padded, train_labels, epo
chs=10, validation_data=(test_padded, test_labels), call
backs=[early_stopping])
```

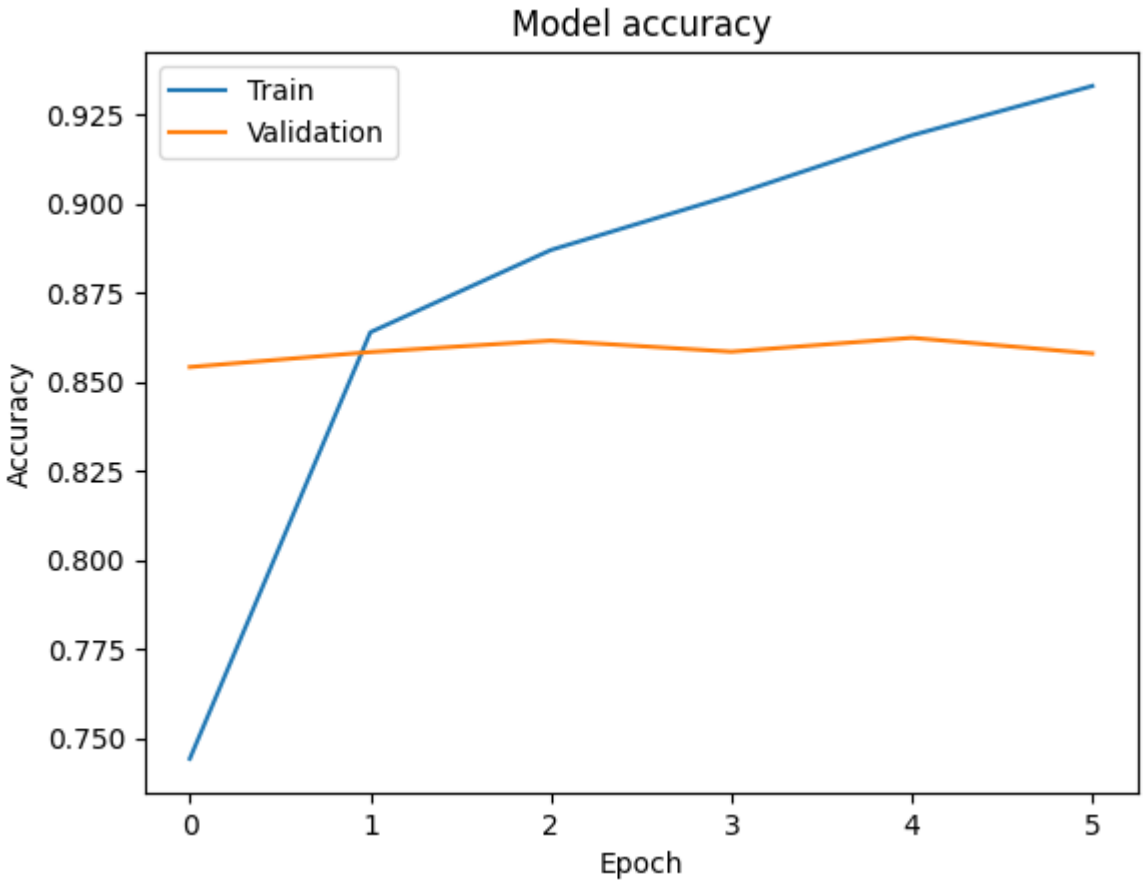
2.5. Evaluation

- **Performance Metrics**: After training, the model's performance was evaluated using accuracy, precision, recall, and F1-score:

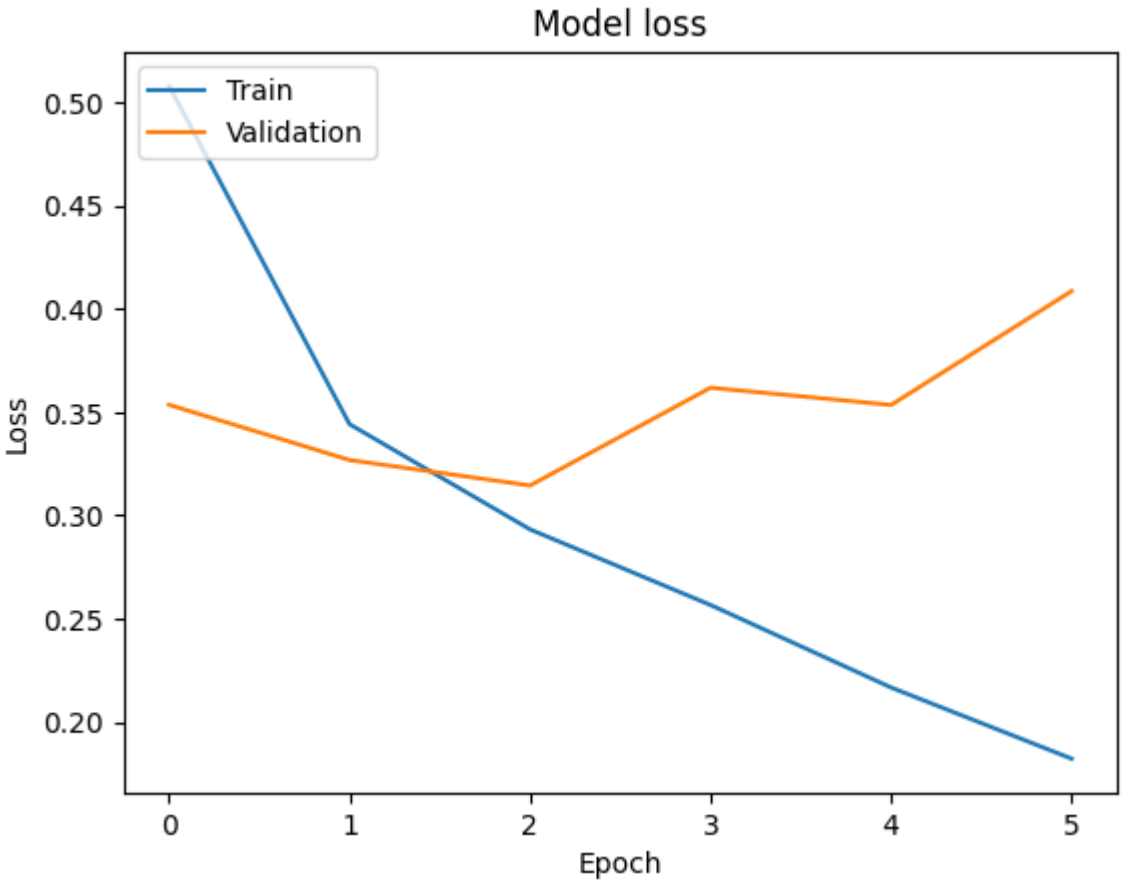
```
y_pred_prob = model_lstm.predict(test_padded)
y_pred = (y_pred_prob > 0.5).astype(int)
```

2.6 Results for LSTM

- **Training and Validation Accuracy**: The training and validation accuracy were plotted over the epochs, showcasing how the model learned:



- **Training and Validation Loss**: The loss values were similarly plotted, indicating model performance over epochs:



- **Evaluation Metrics**: The model's performance on the test set yielded the following results:
  - Accuracy: 0.8615
  - Precision: 0.895625812039844
  - Recall: 0.8207977773367732
  - F1-score: 0.8565807186496842

3. Model Architecture For GRU

- The GRU Model is similar to LSTM Model

```
model_gru = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
input_length=max_length),
    tf.keras.layers.GRU(64),  # Use GRU layer instead of
```

```
LSTM
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

- **Compilation**: The model was compiled with binary cross-entropy loss and the Adam optimizer:

```
model_gru.compile(loss='binary_crossentropy', optimizer
='adam', metrics=['accuracy'])
```

3.1. Training

- The model was trained for 10 epochs with an early stopping callback to prevent overfitting:

```
early_stopping = tf.keras.callbacks.EarlyStopping(monito
r='val_loss', patience=3, restore_best_weights=True)

history_gru = model_gru.fit(train_padded, train_labels,
epochs=10, validation_data=(test_padded, test_labels), c
allbacks=[early_stopping])
```

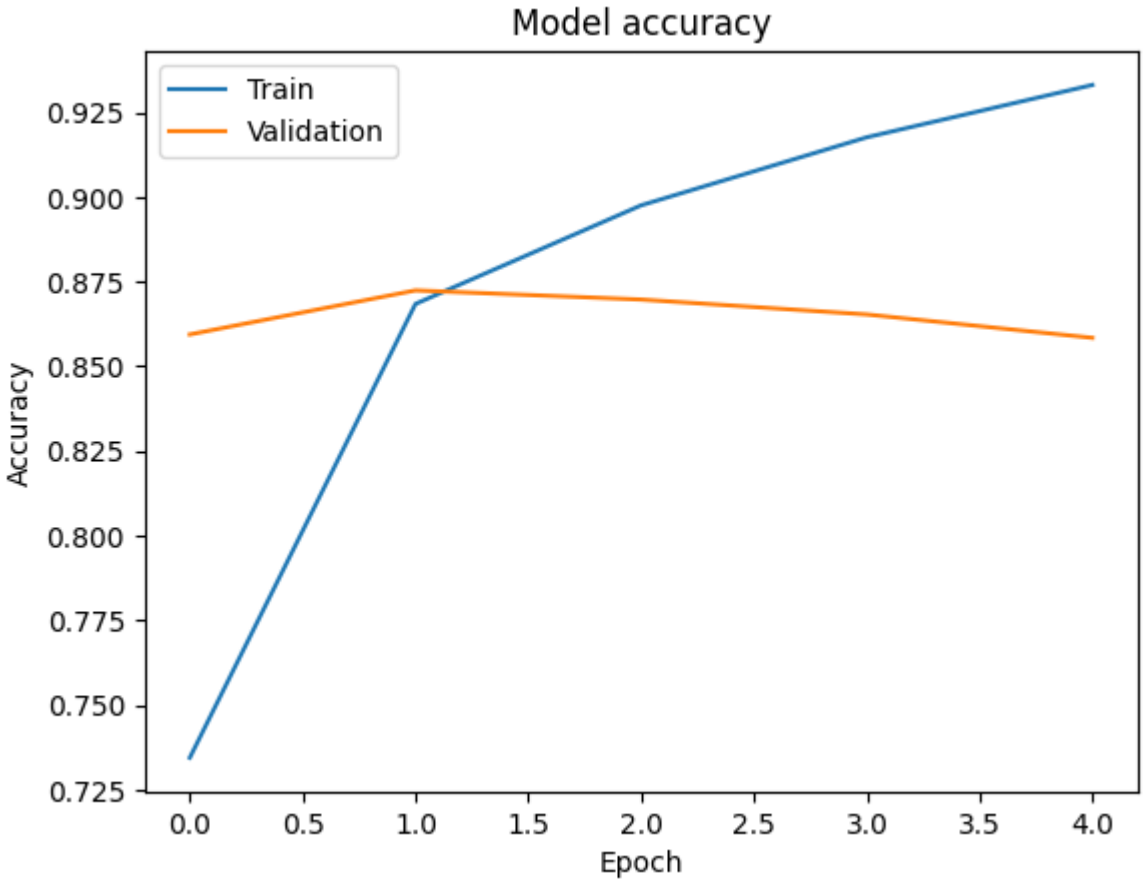
3.2. Evaluation

- **Performance Metrics**: After training, the model's performance was evaluated using accuracy, precision, recall, and F1-score:

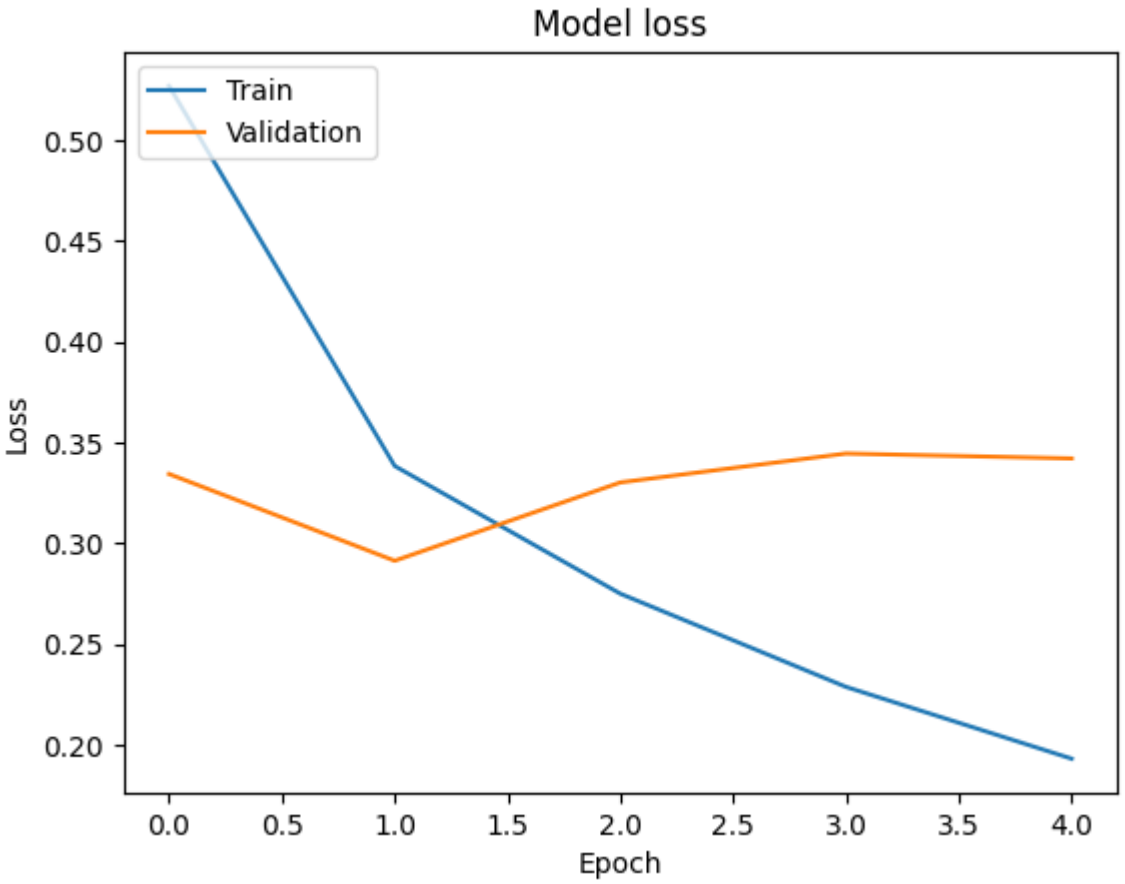
```
y_pred_prob = model_gru.predict(test_padded)
y_pred = (y_pred_prob > 0.5).astype(int)
```

3.3 Results for GRU

- **Training and Validation Accuracy**: The training and validation accuracy were plotted over the epochs, showcasing how the model learned:

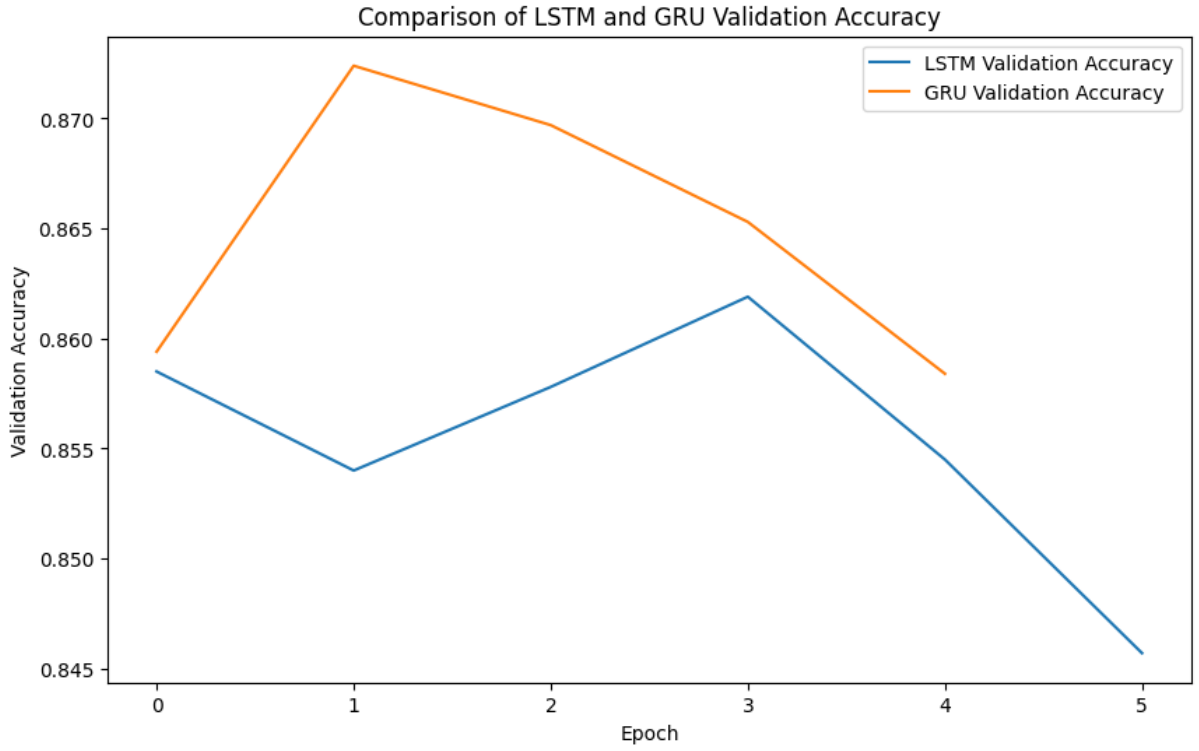


- **Training and Validation Loss**: The loss values were similarly plotted, indicating model performance over epochs:



- **Evaluation Metrics**: The model's performance on the test set yielded the following results:
  - GRU Accuracy: 0.8724
  - GRU Precision: 0.8463095895453708
  - GRU Recall: 0.9124826354435404
  - GRU F1-score: 0.8781512605042017

4. Comparing between 2 Models



GRU Model is better with an accuracy of: 0.8724