Text classication using the Bidirectional versions on IMDB

Author: fchollet, modified by Priyanka A **Date created:** 2020/05/03 **Last modified:** 2024/11/29 **Original Description:** Train a 2-layer bidirectional LSTM on the IMDB movie review sentiment classification dataset.

Modified version: C1_RNN.ipynb: Sentiment Analysis using Bidirectional SimpleRNN, LSTM, and GRU. This script performs text classification on the IMDB dataset using RNN models, compares their performance, and visualizes key metrics.

Setup

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional,
SimpleRNN, LSTM, GRU, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Load and Prepare Data

```
# Load the IMDB dataset
# The dataset contains 25K training and 25K testing samples,
preprocessed into integer sequences.
max_features = 20000  # Maximum number of words in the vocabulary
maxlen = 200          # Maximum length of each review (padded or
truncated)

# Split data into training and testing sets
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)

# Pad sequences to ensure uniform input length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

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

Build Model

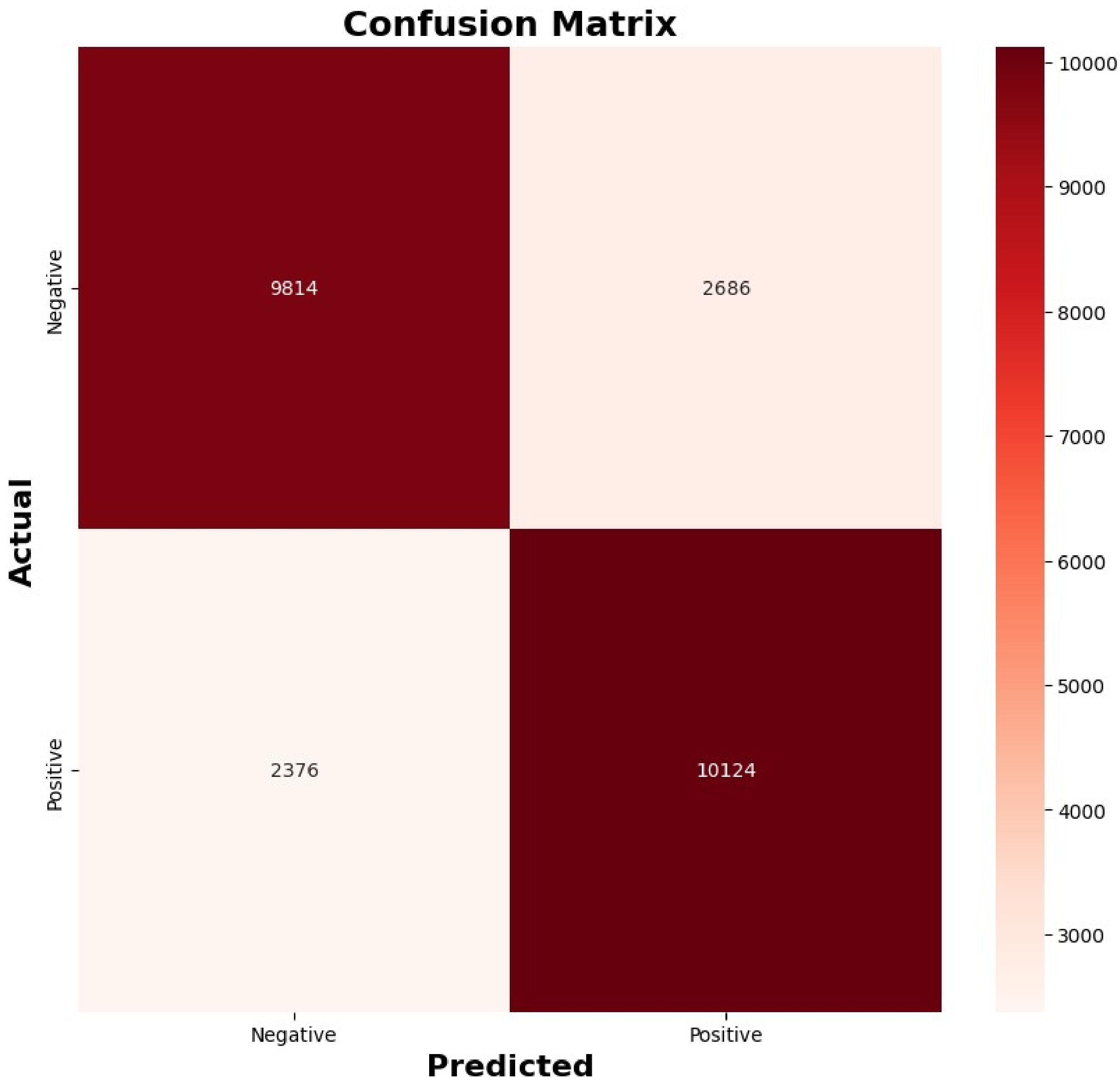
```
def create_model(rnn_layer):
    """
    Creates and compiles a sentiment analysis model with a
bidirectional RNN layer.

    Parameters:
    - rnn_layer: RNN layer instance (SimpleRNN, LSTM, or GRU)

    Returns:
    - Compiled Keras model
    """
    model = Sequential([
        Embedding(input_dim=max_features, output_dim=128),  #
Embedding layer for word vectors
        Bidirectional(rnn_layer),  # Bidirectional wrapper around the
RNN layer
        Dense(1, activation='sigmoid')  # Output layer for binary
classification
    ])
    # Compile the model with Adam optimizer and binary crossentropy
loss
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Train and Evaluate Models

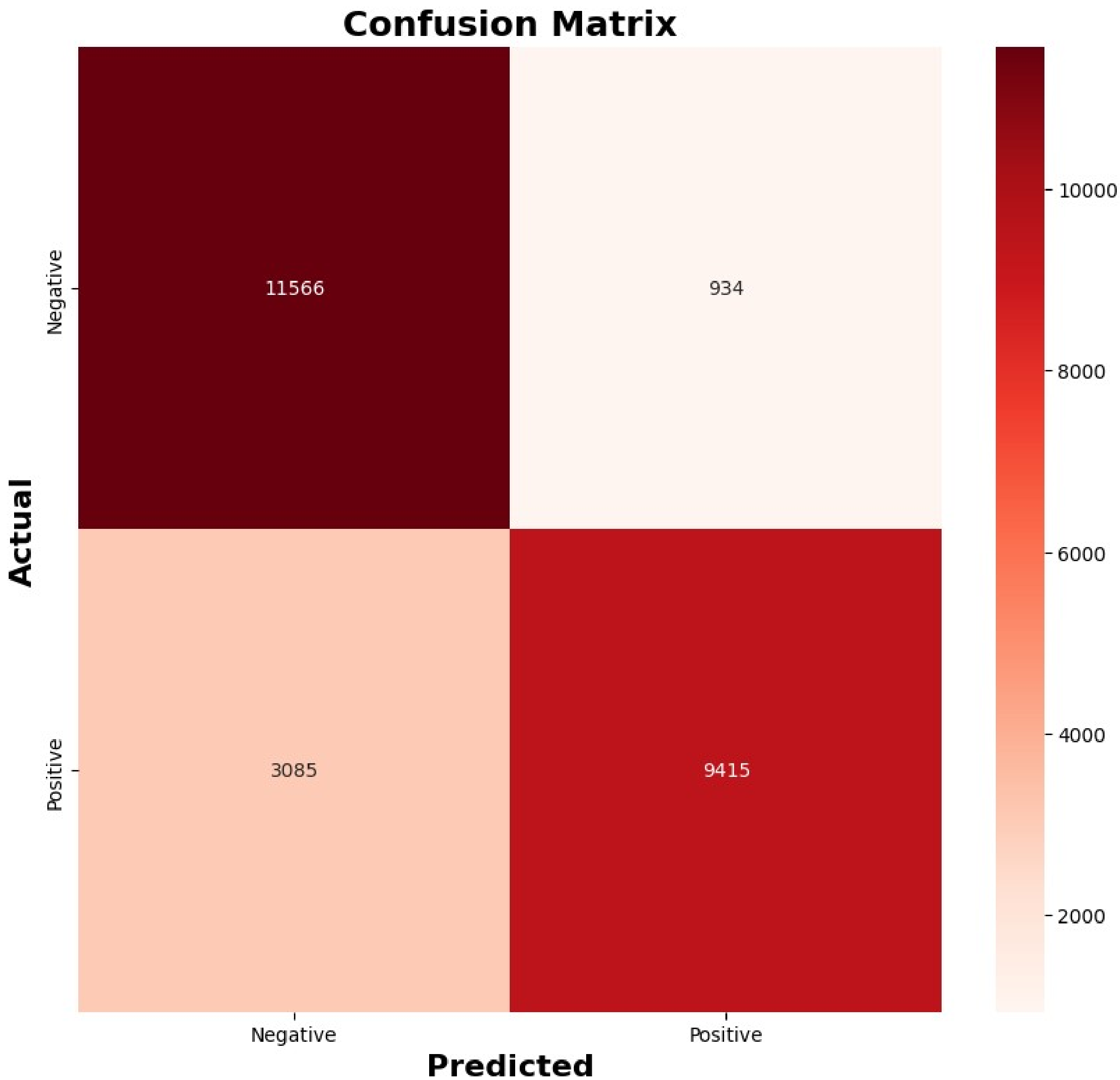
```
# Dictionary to store RNN variants for comparison
models = {
    "Bidirectional SimpleRNN": SimpleRNN(64, return_sequences=False),
    "Bidirectional LSTM": LSTM(64, return_sequences=False),
    "Bidirectional GRU": GRU(64, return_sequences=False)
}

# Dictionary to store results for each model
results = {}

# Train and evaluate each model
for name, layer in models.items():
    print(f"Training {name}...")
    model = create_model(layer)  # Create the model with the specified
RNN layer
    model.fit(x_train, y_train, epochs=3, batch_size=64,
validation_split=0.2, verbose=1)

    # Predict sentiment on the test set
```

```
    y_pred = (model.predict(x_test) > 0.5).astype("int32")

    # Generate and store performance metrics
    results[name] = classification_report(y_test, y_pred,
output_dict=True)

    # Print the confusion matrix for the current model
    print("\n")
    print(f"{name} Confusion Matrix:")
    cm= confusion_matrix(y_test, y_pred)


    # Plot confusion matrix
    plt.figure(figsize=(10, 9))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Reds",
xticklabels=["Negative", "Positive"], yticklabels=["Negative",
"Positive"])
    plt.xlabel('Predicted', fontsize=16, fontweight='bold')
    plt.ylabel('Actual', fontsize=16, fontweight='bold')
    plt.title('Confusion Matrix', fontsize=18, fontweight='bold')
    plt.show()
    print("\n")
    print("\n")
```

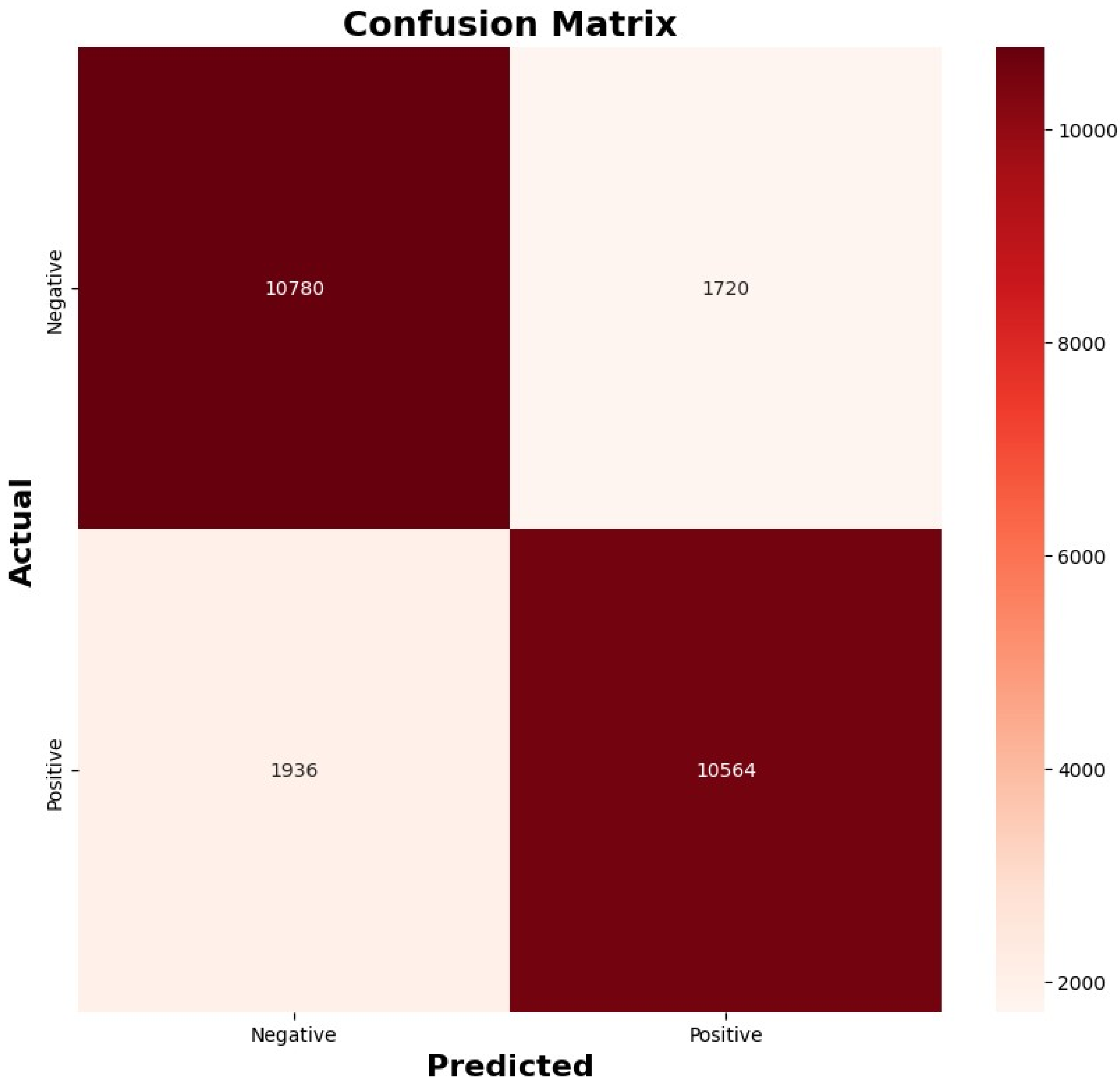
```
Training Bidirectional SimpleRNN...
Epoch 1/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 52ms/step - accuracy: 0.5566 - loss:
0.6751 - val_accuracy: 0.7768 - val_loss: 0.4831
Epoch 2/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 48ms/step - accuracy: 0.8237 - loss:
0.3993 - val_accuracy: 0.7888 - val_loss: 0.4648
Epoch 3/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 47ms/step - accuracy: 0.9360 - loss:
0.1766 - val_accuracy: 0.8022 - val_loss: 0.4487
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step
Bidirectional SimpleRNN Confusion Matrix:
```




```
Training Bidirectional LSTM...
Epoch 1/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 12s 24ms/step - accuracy: 0.6951 - loss:
0.5424 - val_accuracy: 0.8524 - val_loss: 0.3527
Epoch 2/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - accuracy: 0.9099 - loss:
0.2380 - val_accuracy: 0.8568 - val_loss: 0.3336
Epoch 3/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9550 - loss:
0.1306 - val_accuracy: 0.8506 - val_loss: 0.3725
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step
Bidirectional LSTM Confusion Matrix:
```



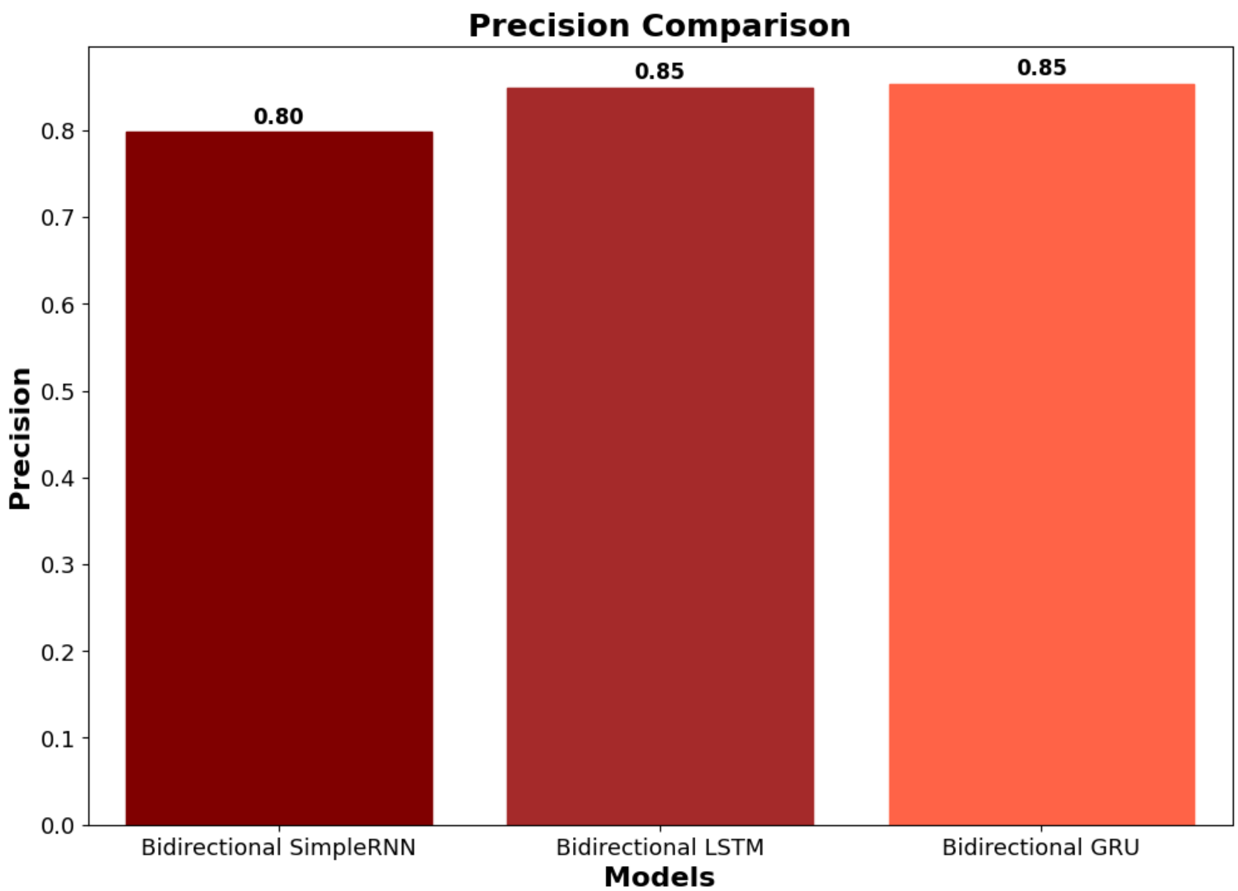

```
Training Bidirectional GRU...
Epoch 1/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - accuracy: 0.6773 - loss:
0.5625 - val_accuracy: 0.8302 - val_loss: 0.4263
Epoch 2/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 9s 20ms/step - accuracy: 0.9018 - loss:
0.2449 - val_accuracy: 0.8608 - val_loss: 0.3455
Epoch 3/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 22ms/step - accuracy: 0.9526 - loss:
0.1386 - val_accuracy: 0.8610 - val_loss: 0.3455
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step
Bidirectional GRU Confusion Matrix:
```

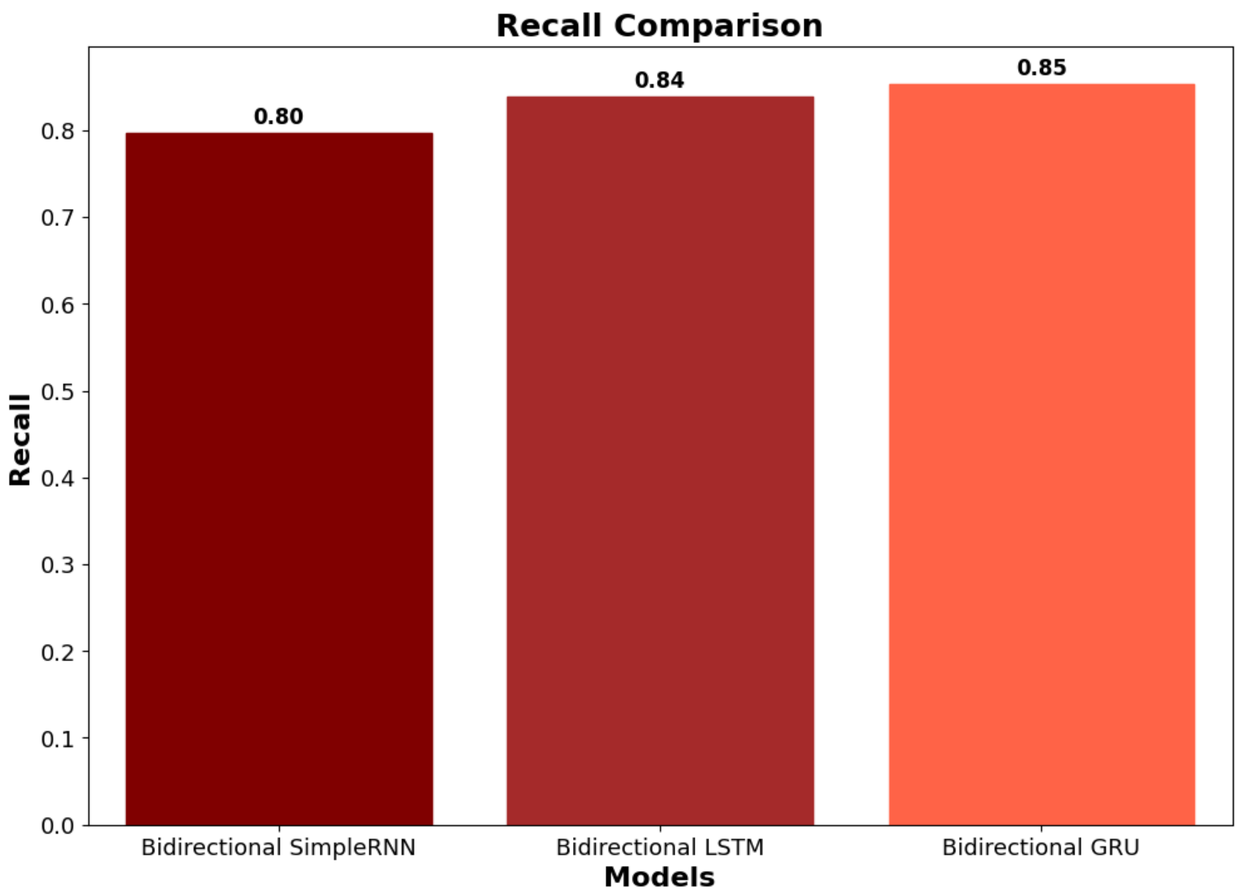


Inferences

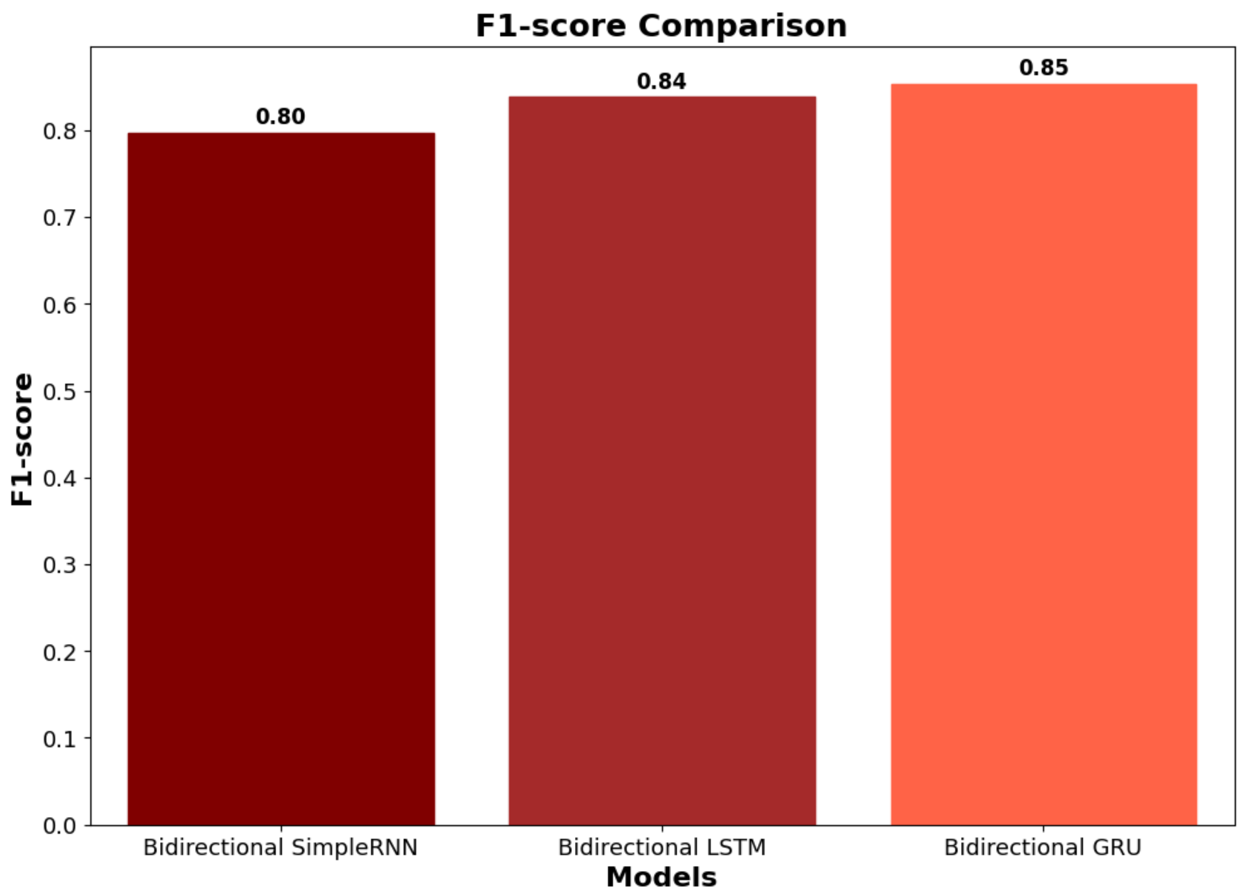
```python
def plot_metrics(results, metric, colors):
    """
    Plots a bar chart comparing a specified metric across models.

    Parameters:
    - results: Dictionary containing model performance metrics
    - metric: Metric to plot ('precision', 'recall', or 'f1-score')
    - colors: List of colors to use for the bars
    """
    values = [results[name]["weighted avg"][metric] for name in
```

```
models.keys()]
    plt.figure(figsize=(10, 7))  # Larger figure size for better
readability

    # Create the bar plot and manually set bar colors
    ax = sns.barplot(x=list(models.keys()), y=values)
    for bar, color in zip(ax.patches, colors):
        bar.set_color(color)

    # Add value labels on top of bars
    for i, v in enumerate(values):
        plt.text(i, v + 0.01, f"{v:.2f}", ha='center', fontsize=12,
color='black', fontweight='bold')

    # Adjust padding for visual appeal
    plt.tight_layout(pad=2)

    # Set plot titles and bold labels
    plt.title(f"{metric.capitalize()} Comparison", fontsize=18,
fontweight='bold')
    plt.ylabel(metric.capitalize(), fontsize=16, fontweight='bold')
    plt.xlabel("Models", fontsize=16, fontweight='bold')
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=13)

    plt.show()

# Define colors for the bars
colors = ['#800000', '#A52A2A', '#FF6347']

# Generate plots for precision, recall, and F1-score
for metric in ["precision", "recall", "f1-score"]:
    plot_metrics(results, metric, colors)
```


Precision Comparison
0.80
0.85
0.85
Precision
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Bidirectional SimpleRNN
Bidirectional LSTM
Bidirectional GRU
Models



Recall Comparison
0.80
0.84
0.85
Recall
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Bidirectional SimpleRNN
Bidirectional LSTM
Bidirectional GRU
Models


F1-score Comparison
F1-score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.80
0.84
0.85
Bidirectional SimpleRNN
Bidirectional LSTM
Bidirectional GRU
Models