MINI PROJECT

Sentiment Analysis using Recurrent Neural Network (RNN)

PROBLEM STATEMENT:

In the age of digital media, understanding public sentiment about products, services, or media is critical for businesses to gain insights into customer opinions and experiences. Sentiment analysis, a subset of natural language processing (NLP), enables us to classify text data, such as reviews, as positive or negative, providing insights into public perception. This project focuses on creating a sentiment analysis model using a Recurrent Neural Network (RNN) to determine whether movie reviews are positive or negative. The project leverages the sequential nature of RNNs, which are well-suited to processing and learning dependencies in text data. The model is trained on movie review data to understand the context, tone, and patterns that typically indicate positive or negative sentiment. Such a system could be extended to assist streaming platforms, movie studios, and online review aggregators to automatically analyze reviews, providing immediate feedback on content reception and helping guide decisions based on audience sentiment. The use of an RNN here demonstrates the model's ability to process variable-length sequential data and provides a strong foundation for applications where understanding human emotions from text is essential.

DATASET USED:

The IMDB dataset, provided as part of the TensorFlow/Keras datasets library, is a widely used benchmark dataset for binary sentiment classification tasks. This dataset contains 50,000 movie reviews, labeled as either positive or negative, making it an ideal resource for training and evaluating sentiment analysis models. The data is divided evenly, with 25,000 reviews for training and 25,000 for testing, ensuring a balanced distribution of sentiment labels. Each review is represented by a sequence of integers, where each integer corresponds to a specific word in the IMDB vocabulary, allowing us to analyze the sequential patterns in text data. The reviews are preprocessed by Keras to contain only the most frequent words, simplifying model training and enhancing performance. This dataset serves as a strong benchmark due to its diversity in language and vocabulary, providing a variety of examples of both positive and negative sentiments. The IMDB dataset is freely available, and its size and balance make it a standard choice for testing sentiment analysis models.

IMPLEMENTATION AND CODE:

1. Importing the necessary libraries

```
import tensorflow as tf
from tensorflow.keras import layers, regularizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
```

2. Loading the dataset:

```
# Load and preprocess the IMDB dataset
vocab_size = 10000
max_length = 200
embedding_dim = 128
rnn_units = 128
```

```
# Load the IMDB dataset
(train_data, train_labels), (test_data, test_labels) = tf.keras.datasets.imdb.load_data(num_words=vocab_size)
```

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

3. Preprocessing the Dataset

```
# Create reverse word index for decoding reviews
word_index = tf.keras.datasets.imdb.get_word_index()
reverse_word_index = {value + 3: key for (key, value) in word_index.items()}
reverse_word_index[0], reverse_word_index[1], reverse_word_index[2] = "<PAD>", "<START>", "<UNK>"
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Decode a sample review
def decode_review(encoded_review):
    return ' '.join([reverse_word_index.get(i, '?') for i in encoded_review])
```

```
# Show a sample decoded review with its label
print("Sample review:", decode_review(train_data[0]))
print("Sample label:", "Positive" if train_labels[0] == 1 else "Negative")
```

```
Sample review: <START> this film was just brilliant casting location scenery story direction everyone's really suited the part
Sample label: Positive
```

4. Splitting the datasets

```
# Pad the sequences to ensure uniform input length
train_data = tf.keras.preprocessing.sequence.pad_sequences(train_data, maxlen=max_length, padding='post')
test_data = tf.keras.preprocessing.sequence.pad_sequences(test_data, maxlen=max_length, padding='post')
```

5. Building the RNN Model

```
# Define the RNN model
model = tf.keras.Sequential([
    layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    layers.SpatialDropout1D(0.2),
    layers.GRU(rnn_units, return_sequences=True, dropout=0.2, recurrent_dropout=0.2, kernel_regularizer=regularizers.l2(0.001)),
    layers.GRU(rnn_units, dropout=0.2, recurrent_dropout=0.2, kernel_regularizer=regularizers.l2(0.001)),
    (import) layers: Any
    layers.Dense(1, activation='sigmoid')
])

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

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
spatial_dropout1d_1 (SpatialDropout1D)	?	0 (unbuilt)
gru_2 (GRU)	?	0 (unbuilt)
gru_3 (GRU)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0 (unbuilt)
dense_3 (Dense)	?	0 (unbuilt)

6. Creating Callbacks:

```
# Callbacks for early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min')
```

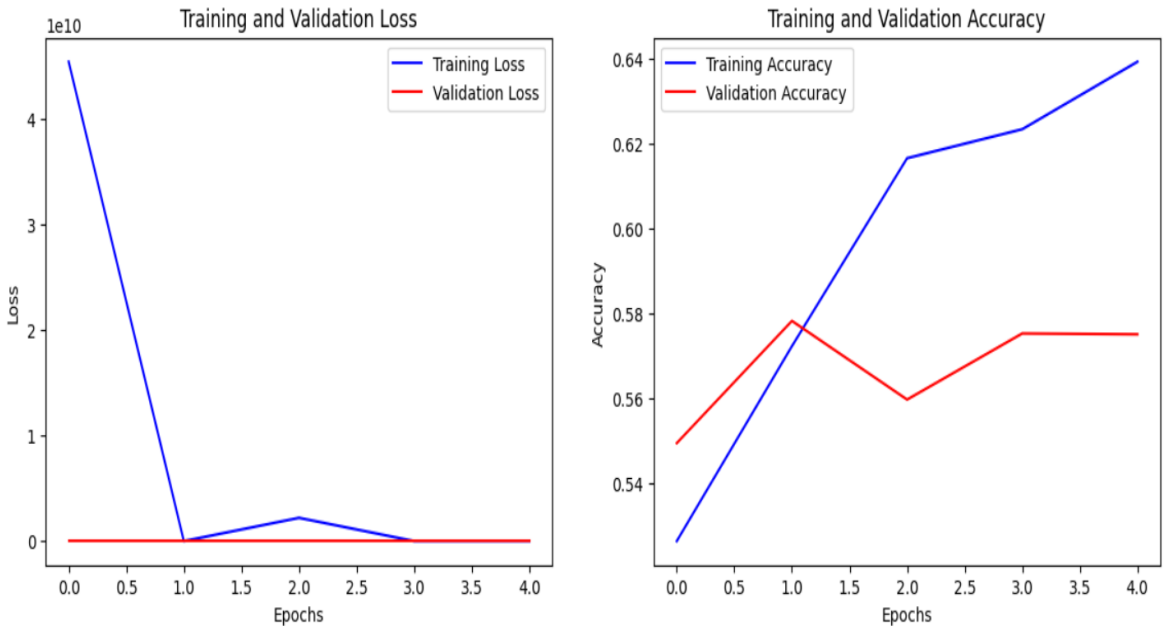
7. Training the model with Callbacks:

```
# Training the model with callbacks
epochs = 15
batch_size = 64

history = model.fit(
    train_data, train_labels, epochs=epochs, batch_size=batch_size,
    validation_data=(test_data, test_labels),
    callbacks=[early_stopping, model_checkpoint], verbose=1
)
```

```
Epoch 1/15
391/391 ━━━━━━━━━━━━━━━━━━━━ 478s 1s/step - accuracy: 0.5076 - loss: 231756864.0000 - val_accuracy: 0.5494 - val_loss: 0.7150
Epoch 2/15
391/391 ━━━━━━━━━━━━━━━━━━━━ 452s 1s/step - accuracy: 0.5561 - loss: 0.7377 - val_accuracy: 0.5782 - val_loss: 0.7141
Epoch 3/15
391/391 ━━━━━━━━━━━━━━━━━━━━ 456s 1s/step - accuracy: 0.6231 - loss: 248490672.0000 - val_accuracy: 0.5597 - val_loss: 0.7495
Epoch 4/15
391/391 ━━━━━━━━━━━━━━━━━━━━ 502s 1s/step - accuracy: 0.6229 - loss: 23.1383 - val_accuracy: 0.5753 - val_loss: 0.7441
Epoch 5/15
```

8. Plotting the Accuracy and Loss

```
# Plotting the training and validation loss and accuracy
def plot_metrics(history):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    # Loss plot
    ax1.plot(history.history['loss'], label='Training Loss', color='blue')
    ax1.plot(history.history['val_loss'], label='Validation Loss', color='red')
    ax1.set_title('Training and Validation Loss')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Loss')
    ax1.legend()

    # Accuracy plot
    ax2.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
    ax2.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
    ax2.set_title('Training and Validation Accuracy')
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('Accuracy')
    ax2.legend()

    plt.show()

plot_metrics(history)
```



9. Evaluating the model on Test Data

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_data, test_labels, verbose=2)
print(f'Test Accuracy: {test_accuracy:.4f}, Test Loss: {test_loss:.4f}')

# Load the best saved model
model.load_weights('best_model.keras')
```

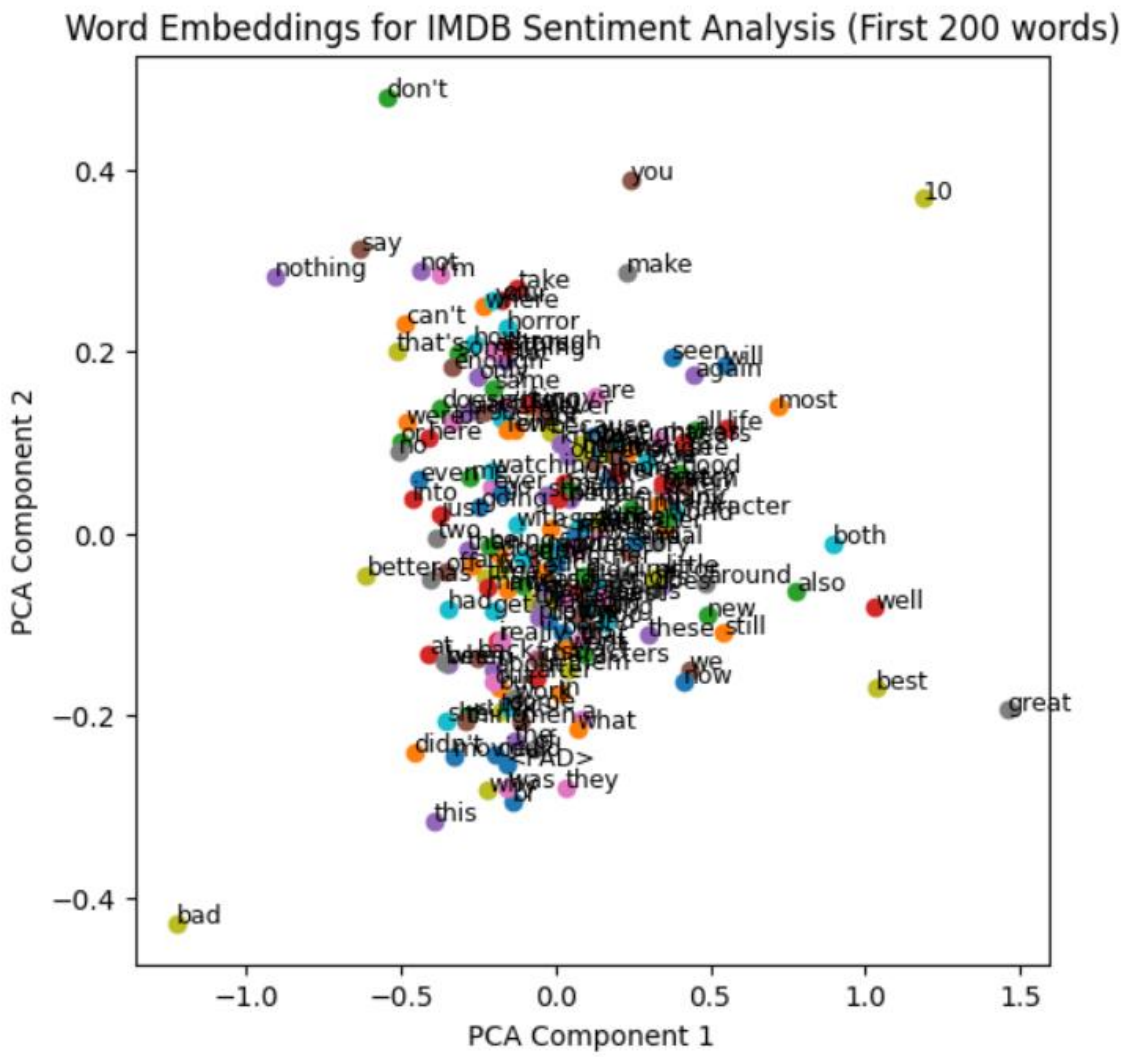
```
# Predict on test data and show predictions for a few reviews
predictions = model.predict(test_data[:5])

for i, prediction in enumerate(predictions):
    sentiment = "Positive" if prediction > 0.5 else "Negative"
    print(f"\nReview {i + 1} Prediction: {sentiment} (Confidence: {prediction[0]:.2f})")
    print(f"Actual Label: {'Positive' if test_labels[i] == 1 else 'Negative'}")
    print("Review text:", decode_review(test_data[i]))
    print("----------------------------------------------------")
```

Output:

```
Test Accuracy: 0.8520, Test Loss: 0.3900


Review 1 Prediction: Positive
Actual Label: Positive
Review text: this film was just brilliant casting location scenery story direction everyon
```

10.Visualizing the words

```
# Visualizing word embeddings
# Extract embeddings and plot using PCA
embedding_layer = model.layers[0]
embeddings = embedding_layer.get_weights()[0]

pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(embeddings[:200])  # Take first 200 words for visualization

plt.figure(figsize=(6, 6))
for i in range(200):
    plt.scatter(reduced_embeddings[i, 0], reduced_embeddings[i, 1])
    plt.annotate(reverse_word_index.get(i, "<UNK>"), (reduced_embeddings[i, 0], reduced_embeddings[i, 1]), fontsize=9)
plt.title("Word Embeddings for IMDB Sentiment Analysis (First 200 words)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```




```
# Additional predictions on random test reviews with interpretation
print("\nAdditional Model Predictions on Random Test Samples:")
num_samples = 5
random_indices = np.random.choice(len(test_data), num_samples, replace=False)
random_reviews = test_data[random_indices]
random_labels = test_labels[random_indices]

for i, review in enumerate(random_reviews):
    prediction = model.predict(np.array([review]))[0][0]
    sentiment = "Positive" if prediction > 0.5 else "Negative"
    print(f"\nRandom Review {i + 1} Prediction: {sentiment} (Confidence: {prediction:.2f})")
    print(f"Actual Label: {'Positive' if random_labels[i] == 1 else 'Negative'}")
    print("Review text:", decode_review(review))
    print("----------------------------------------------------")

# Analysis of predicted sentiment distribution
positive_count = sum([1 for pred in predictions if pred > 0.5])
negative_count = len(predictions) - positive_count

print(f"\nDistribution of Sentiment Predictions (First 5 samples):")
print(f"Positive: {positive_count}, Negative: {negative_count}")
print("----------------------------------------------------")
```

OUTPUT:

```
Additional Model Predictions on Random Test Samples:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 407ms/step

Random Review 1 Prediction: Negative (Confidence: 0.49)
Actual Label: Positive
Review text: music were all good it's just that after you see a bunch of people doing things you can't t
--------------------------------------------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step

Random Review 2 Prediction: Negative (Confidence: 0.41)
Actual Label: Positive
Review text: has been partially <UNK> to change the <UNK> of his face in the original and as usual with
--------------------------------------------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step

Random Review 3 Prediction: Positive (Confidence: 0.60)
Actual Label: Positive
Review text: <UNK> of the evil <UNK> <UNK> but his dialogue is written with a very heavy swedish accent
--------------------------------------------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
```

RESULT:

Thus the Sentimental Analysis for IMDB Datasets implemented using RNN is built successfully and the output is verified.