Ex No: 6 **Date: 30-08-2024**

BUILD A RECURRENT NEURAL NETWORK

AIM:

To build a recurrent neural network with Keras/TensorFlow.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a recurrent neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, SimpleRNN
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.datasets import imdb
import matplotlib.pyplot as plt


# Step 1: Download and load the dataset (IMDb dataset with 10,000 most frequent words)
vocab_size = 10000  # Use the 10,000 most common words
max_len = 100  # Maximum length of reviews


# Load the dataset
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Step 2: Perform analysis and preprocessing of the dataset
# Pad the sequences to ensure all inputs are of the same length
x_train = pad_sequences(x_train, maxlen=max_len, padding='post')
x_test = pad_sequences(x_test, maxlen=max_len, padding='post')

# Step 3: Build a simple RNN model using Keras/TensorFlow
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32, input_length=max_len),  # Embedding layer
    SimpleRNN(32, return_sequences=False),  # Simple RNN layer; can switch to LSTM if needed
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Step 4: Compile and fit the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model and capture the training history
history = model.fit(x_train, y_train, epochs=2, batch_size=64, validation_split=0.2)  # Train the model with 2 epochs

# Step 5: Perform prediction with the test dataset
predictions = model.predict(x_test)
```

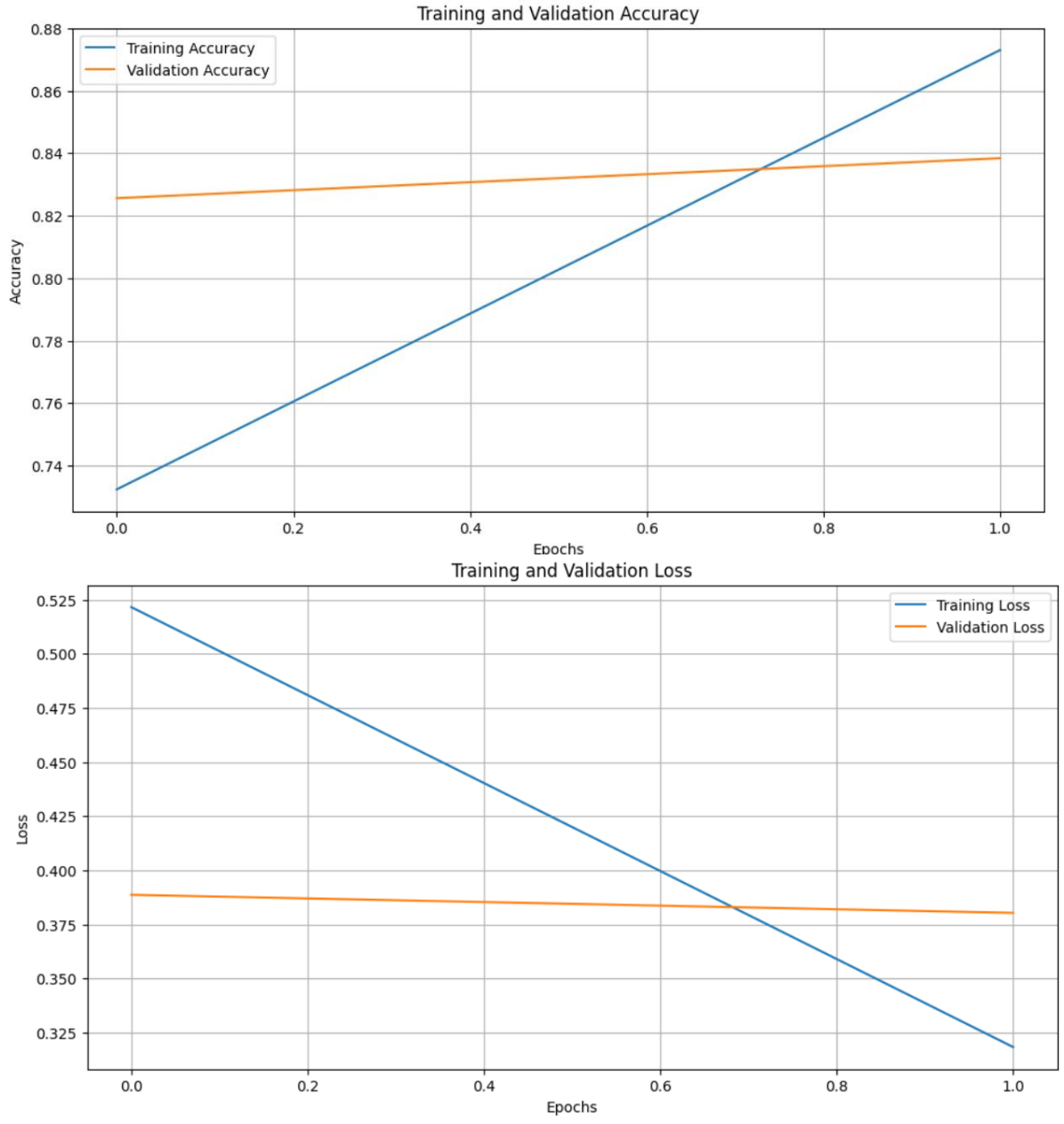
```
# Step 6: Calculate performance metrics
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')

# Plotting training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Plotting training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:

```
Epoch 1/2
313/313 [==============================] - 14s 38ms/step - loss: 0.5216 - accuracy: 0.7324 - val_loss: 0.3887 - val_accuracy: 0.8256
Epoch 2/2
313/313 [==============================] - 13s 41ms/step - loss: 0.3184 - accuracy: 0.8730 - val_loss: 0.3804 - val_accuracy: 0.8384
782/782 [==============================] - 8s 10ms/step
782/782 [==============================] - 6s 8ms/step - loss: 0.3848 - accuracy: 0.8395
Test accuracy: 0.8395
```



RESULT:

Thus, a recurrent neural network with Keras/TensorFlow was successfully implemented.