Ex-2: Design a Neural Network for Classifying Movie Reviews (Binary Classification) using IMDB Dataset

Objective:

To design and implement a deep learning model using **Neural Networks** for classifying movie reviews as **positive** or **negative** using the **IMDB dataset**.

Theory

1. IMDB Dataset Overview

- The **IMDB dataset** consists of **50,000** movie reviews labeled as **positive (1)** or **negative (0)**.
- It is divided into **25,000** training and **25,000** testing samples.
- The dataset contains the **top 10,000 most frequent words**, and each word is converted into an integer index.

2. Neural Network for Text Classification

A **neural network** is used for text classification, consisting of:

- **Embedding Layer:** Converts word indices into dense vectors.
- **LSTM Layer:** Captures sequential dependencies in text.
- **Fully Connected (Dense) Layers:** Used for final classification.
- **Activation Function:** sigmoid is used in the output layer for binary classification.
- **Loss Function:** binary_crossentropy since we are dealing with a binary classification problem.

Software & Libraries Required

- Python 3.x
- TensorFlow / Keras
- NumPy
- Matplotlib

Install dependencies using:

```
pip install tensorflow numpy matplotlib
```

Procedure

Step 1: Import Necessary Libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad_sequences

import numpy as np

import matplotlib.pyplot as plt

Step 2: Load and Preprocess the Data

Load dataset with only the top 10,000 words

vocab_size = 10000 # Top words to consider

max_length = 200   # Maximum words per review

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

Pad sequences to ensure uniform length

x_train = pad_sequences(x_train, maxlen=max_length, padding='post', truncating='post')

x_test = pad_sequences(x_test, maxlen=max_length, padding='post', truncating='post')

Step 3: Build the Neural Network Model

Define the model

model = keras.Sequential([

    keras.layers.Embedding(input_dim=vocab_size, output_dim=128, input_length=max_length),

    keras.layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2),

    keras.layers.Dense(64, activation='relu'),

```python
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')  # Sigmoid activation for binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()
```

Step 4: Train the Model

```python
# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

Step 5: Evaluate the Model

```python
# Evaluate model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

Step 6: Make Predictions

```python
# Example prediction
new_review = [[1, 45, 6, 200, 54]]  # Example tokenized input
new_review_sequence = pad_sequences(new_review, maxlen=max_length)
prediction = model.predict(new_review_sequence)
print("Positive" if prediction > 0.5 else "Negative")
```

Step 7: Plot Accuracy and Loss Curves

```python
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```

Observations/Expected Output:

1. The model trains efficiently and achieves an accuracy of **above 85%** on the test dataset.
2. The **LSTM** layer helps in capturing the sequential nature of text.
3. The **dropout layers** prevent overfitting and improve generalization.
4. Accuracy and loss curves provide insights into training progress.

Result

- We successfully designed a **Neural Network** for **binary classification** of movie reviews using **IMDB dataset**.
- The model can be further improved using **Bidirectional LSTMs, CNNs**, or **Transformer models**.

Viva Questions

1. What is the role of the **embedding layer** in the neural network?
2. Why do we use **LSTM** instead of a simple **Dense** network?
3. What is the significance of **dropout layers** in deep learning models?
4. Explain why **binary cross-entropy** is used as the loss function.
5. How can we improve the performance of the model further?

---||--

Here's Complete executable code in a single block:

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Load and Preprocess the Data
vocab_size = 10000  # Top words to consider
max_length = 200    # Maximum words per review
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences to ensure uniform length
x_train = pad_sequences(x_train, maxlen=max_length, padding='post', truncating='post')
x_test = pad_sequences(x_test, maxlen=max_length, padding='post', truncating='post')

# Step 3: Build the Neural Network Model
model = keras.Sequential([
    keras.layers.Embedding(input_dim=vocab_size, output_dim=128, input_length=max_length),
    keras.layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')  # Sigmoid activation for binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()

# Step 4: Train the Model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))

# Step 5: Evaluate the Model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

```python
# Step 6: Make Predictions
new_review = [[1, 45, 6, 200, 54]]  # Example tokenized input
new_review_sequence = pad_sequences(new_review, maxlen=max_length)
prediction = model.predict(new_review_sequence)
print("Positive" if prediction > 0.5 else "Negative")

# Step 7: Plot Accuracy and Loss Curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
C:\Users\91901\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `inpu
t_length` is deprecated. Just remove it.
  warnings.warn(
2025-02-06 18:45:14.072343: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use avai
lable CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)

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

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 43s 98ms/step - accuracy: 0.5150 - loss: 0.6904 - val_accuracy: 0.5982 - val_loss: 0.6485
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 38s 98ms/step - accuracy: 0.6270 - loss: 0.6239 - val_accuracy: 0.6001 - val_loss: 0.6473
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 37s 95ms/step - accuracy: 0.6613 - loss: 0.6063 - val_accuracy: 0.7260 - val_loss: 0.5649
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 38s 96ms/step - accuracy: 0.7924 - loss: 0.4951 - val_accuracy: 0.7915 - val_loss: 0.5219
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 37s 95ms/step - accuracy: 0.8259 - loss: 0.4352 - val_accuracy: 0.7570 - val_loss: 0.5409
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.7594 - loss: 0.5382
Test Accuracy: 0.7570
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 451ms/step
Negative
```