Exp-3: Designing a Neural Network for Classifying Newswires (Multi-Class Classification) Using the Reuters Dataset

Objective:

To design, implement, and train a **deep learning model** to classify **news articles** into multiple categories using the **Reuters dataset** with TensorFlow/Keras.

Prerequisites:

Before starting, ensure you have:

- Basic knowledge of **Neural Networks**, **Deep Learning**, and **Multi-Class Classification**.
- Familiarity with **Python**, **NumPy**, and **TensorFlow/Keras**.
- Installed the required libraries.

Install Dependencies:

```
pip install tensorflow numpy matplotlib
```

Step 1: Import Required Libraries

First, import the necessary Python libraries:

```
import numpy as np

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt
```

Step 2: Load the Reuters Dataset

The **Reuters dataset** is a collection of categorized news articles.

```
# Load the Reuters dataset

from tensorflow.keras.datasets import reuters


# Load dataset with the top 10,000 words

(x_train, y_train), (x_test, y_test) = reuters.load_data(num_words=10000)
```

Display dataset statistics

```
print(f"Training samples: {len(x_train)}")
print(f"Testing samples: {len(x_test)}")
print(f"Number of classes: {max(y_train) + 1}")
```

Step 3: Data Preprocessing

To process the dataset, convert sequences into numerical tensors.

Convert sequences into uniform length using padding

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

Define parameters

```
max_length = 200  # Maximum sequence length
```

Padding sequences to ensure uniform input size

```
x_train = pad_sequences(x_train, maxlen=max_length)
x_test = pad_sequences(x_test, maxlen=max_length)
```

Convert labels to one-hot encoding

```
y_train = to_categorical(y_train, num_classes=46)
y_test = to_categorical(y_test, num_classes=46)
```

Step 4: Building the Neural Network Model

We design a deep learning model using **an embedding layer, CNN + LSTM layers, and a dense output layer**.

Build the neural network model

```
model = keras.models.Sequential([
    keras.layers.Embedding(input_dim=10000, output_dim=128, input_length=max_length),
```

```
    keras.layers.Conv1D(64, 5, activation='relu'),
    keras.layers.MaxPooling1D(pool_size=2),
    keras.layers.LSTM(64, return_sequences=True),
    keras.layers.LSTM(64),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(46, activation='softmax')  # Output layer for multi-class classification
])

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

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

Explanation:

- **Embedding Layer:** Converts words into dense vector representations.
- **Conv1D Layer:** Extracts local patterns from word sequences.
- **MaxPooling1D Layer:** Reduces dimensionality and prevents overfitting.
- **LSTM Layers:** Capture long-term dependencies in the text.
- **Dense Layer:** Fully connected layer with ReLU activation.
- **Dropout Layer:** Reduces overfitting by randomly deactivating neurons.
- **Softmax Output Layer:** Provides probabilities for 46 categories.

Step 5: Training the Model

We train the model using **cross-entropy loss** and monitor validation performance.

```
# Train the model
```

```
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

Explanation:

- **Epochs:** Number of iterations over the dataset.
- **Batch Size:** Number of samples per training step.
- **Validation Split:** 20% of data is used for validation.

Step 6: Model Evaluation

Evaluate the model’s accuracy on test data.

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

Explanation:

- **Loss Function:** Measures the error in predictions.
- **Accuracy Metric:** Measures the percentage of correct predictions.

Step 7: Visualizing Training Performance

Plot training and validation accuracy/loss to analyze model behavior.

```
# Plot accuracy and loss
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation 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.legend()
plt.title('Model Loss')
plt.show()
```

Explanation:

- **Accuracy Graph:** Shows improvement in classification performance over epochs.
- **Loss Graph:** Helps identify overfitting or underfitting.

Step 8: Making Predictions

Perform inference on unseen data.

```
# Make predictions on test data
predictions = model.predict(x_test)


# Display sample prediction
sample_index = 5
predicted_class = np.argmax(predictions[sample_index])
true_class = np.argmax(y_test[sample_index])
print(f"Predicted Category: {predicted_class}")
print(f"Actual Category: {true_class}")
```

Explanation:

- **Predicts a class label** for an unseen news article.
- **Compares the predicted category** with the actual label.

Result

- Successfully built a **deep learning model** for classifying newswires into **46 categories**.
- Used **word embeddings, CNN + LSTM layers**, achieving high accuracy.
- Evaluated performance and visualized training progress.

Further Improvements:

✓ **Use Pretrained Embeddings** (GloVe, Word2Vec) for better word representation.
✓ **Hyperparameter Tuning** (batch size, learning rate, dropout rate).
✓ **Experiment with Transformer Models** (BERT, GPT) for enhanced accuracy.

OUT PUT:

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
conv1d (Conv1D)	?	0 (unbuilt)
max_pooling1d (MaxPooling1D)	?	0
lstm (LSTM)	?	0 (unbuilt)
lstm_1 (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 89ms/step - accuracy: 0.3423 - loss: 2.8723 - v
al_accuracy: 0.5253 - val_loss: 1.8037
Epoch 2/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 9s 78ms/step - accuracy: 0.5236 - loss: 1.8561 - va
l_accuracy: 0.5504 - val_loss: 1.6936
Epoch 3/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 9s 80ms/step - accuracy: 0.5657 - loss: 1.6632 - va
l_accuracy: 0.5442 - val_loss: 1.6701
Epoch 4/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 15s 132ms/step - accuracy: 0.5866 - loss: 1.5307 -
val_accuracy: 0.5949 - val_loss: 1.5623
Epoch 5/10
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 18s 105ms/step - accuracy: 0.6159 - loss: 1.4092 - val_accuracy: 0.6110 -
 val_loss: 1.5755
Epoch 6/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 9s 81ms/step - accuracy: 0.6509 - loss: 1.3055 - val_accuracy: 0.6299 - v
al_loss: 1.4496
Epoch 7/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 9s 79ms/step - accuracy: 0.6795 - loss: 1.1961 - val_accuracy: 0.6433 - v
al_loss: 1.4170
Epoch 8/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 10s 85ms/step - accuracy: 0.6991 - loss: 1.1332 - val_accuracy: 0.6349 -
val_loss: 1.4428
Epoch 9/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 10s 90ms/step - accuracy: 0.7215 - loss: 1.0468 - val_accuracy: 0.6483 -
val_loss: 1.4222
Epoch 10/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 10s 86ms/step - accuracy: 0.7498 - loss: 0.9557 - val_accuracy: 0.6644 -
val_loss: 1.4248
71/71 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.6530 - loss: 1.5072
Test Accuracy: 0.6460
```