Experiment-10

Aim:

To compare the performance of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) on a sequence classification task using a sample dataset.

Apparatus Required:

- Python 3.x
- Jupyter Notebook or any Python IDE (VSCode, PyCharm, etc.)
- Libraries: numpy, tensorflow, keras, matplotlib
- GPU (optional for faster training)

Theory:

Convolutional Neural Networks (CNN):

CNNs are specialized deep learning models primarily used for spatial data such as images. They utilize convolutional layers to automatically and adaptively learn spatial hierarchies of features from input images. CNNs excel at capturing local spatial correlations through kernels/filters.

Recurrent Neural Networks (RNN):

RNNs are designed to handle sequential data by maintaining a hidden state that captures information from previous inputs. They are effective for time series, natural language processing, and other tasks where the order of data points matters. Variants like LSTM and GRU help alleviate issues such as vanishing gradients.

Comparison Points:

- **Input type:** CNNs work well with spatial data, RNNs with temporal/sequential data.
- **Architecture:** CNN uses convolution and pooling; RNN uses recurrent connections.
- **Use cases:** CNNs for image classification, object detection; RNNs for text, speech, sequence prediction.
- **Training complexity:** CNNs generally train faster; RNNs require careful tuning due to sequence dependencies.

Experiment Setup:

We will use a simple text classification dataset (IMDB movie review sentiment dataset) to compare CNN and RNN models on the same task.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,
Dense, LSTM, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
# Parameters
max_features = 10000  # Vocabulary size
max_len = 500         # Max length of each review
# Load IMDB dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=max_features)
# Pad sequences to the same length
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)
# CNN Model
def create_cnn():
    model = Sequential()
    model.add(Embedding(max_features, 128, input_length=max_len))
    model.add(Conv1D(64, kernel_size=5, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))  # Dropout for regularization
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=1e-4),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
# RNN Model (LSTM)
```

```
def create_rnn():
    model = Sequential()
    model.add(Embedding(max_features, 128, input_length=max_len))
    model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))  # Add dropout
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=1e-4),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=2,
restore_best_weights=True)
# Train CNN
cnn_model = create_cnn()
print("Training CNN Model...")
cnn_history = cnn_model.fit(
    X_train, y_train,
    epochs=15,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
# Train RNN
rnn_model = create_rnn()
print("Training RNN Model...")
rnn_history = rnn_model.fit(
    X_train, y_train,
    epochs=15,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
# Evaluate both models on test data
cnn_eval = cnn_model.evaluate(X_test, y_test, verbose=0)
rnn_eval = rnn_model.evaluate(X_test, y_test, verbose=0)
print(f"CNN Test Accuracy: {cnn_eval[1]*100:.2f}%")
print(f"RNN Test Accuracy: {rnn_eval[1]*100:.2f}%")
# Plot training and validation accuracy
plt.plot(cnn_history.history['accuracy'], label='CNN Train Acc')
plt.plot(cnn_history.history['val_accuracy'], label='CNN Val Acc')
plt.plot(rnn_history.history['accuracy'], label='RNN Train Acc')
plt.plot(rnn_history.history['val_accuracy'], label='RNN Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Result: Both CNN and RNN models were successfully trained, and their performance was evaluated in terms of accuracy on the IMDB sentiment classification task.