EX 6 BIDIRECTIONAL RNN VS FEEDFORWARD NN FOR TIME-SERIES PREDICTION

Problem Statement:

Implement a Bidirectional Recurrent Neural Network (BiRNN) to predict sequences in time-series data. Train the model and compare its performance with a traditional Feedforward Neural Network (FFNN) for sequence-based tasks.

Suggested Dataset: Airline Passenger Dataset

Objectives:

1. Understand the application of RNNs and FFNNs for time-series forecasting.
2. Train a bidirectional RNN model to capture sequential dependencies.
3. Compare prediction performance with a feedforward neural network.
4. Visualize and evaluate predictions using metrics such as Mean Squared Error (MSE).

Scope:

Recurrent Neural Networks are well-suited for tasks involving sequential data. This experiment demonstrates the power of BiRNNs in modeling time dependencies and compares them with simpler feedforward architectures, providing insight into the role of model memory in sequence modeling.

Tools and Libraries Used:

1. Python 3.x
2. pandas
3. numpy
4. matplotlib
5. scikit-learn
6. PyTorch

Implementation Steps:

Step 1: Import Necessary Libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

Step 2: Data Preparation

```
url='https://raw.githubusercontent.com/jbrownlee/Datasets/refs/heads/master/monthly
-airline-passengers.csv'
```

```
df = pd.read_csv(url, usecols=[1])
data = df.values.astype('float32')

scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)
```

Step 3: Create Sequences for Time-Series Prediction

```
SEQ_LENGTH = 10
X = np.array([data_scaled[i:i+SEQ_LENGTH] for i in range(len(data_scaled) - SEQ_LENGTH)])
y = np.array([data_scaled[i + SEQ_LENGTH] for i in range(len(data_scaled) - SEQ_LENGTH)])

train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Step 4: Prepare PyTorch Datasets and Loaders

```
import torch
from torch.utils.data import TensorDataset, DataLoader

X_train_tensor = torch.tensor(X_train)
y_train_tensor = torch.tensor(y_train)
X_test_tensor  = torch.tensor(X_test)
y_test_tensor  = torch.tensor(y_test)

train_loader = DataLoader(TensorDataset(X_train_tensor, y_train_tensor), batch_size=16, shuffle=True)
test_loader  = DataLoader(TensorDataset(X_test_tensor, y_test_tensor), batch_size=1)
```

Step 5: Define BiRNN and Feedforward Models

```
import torch.nn as nn

class BiRNN(nn.Module):
  def __init__(self, input_size=1, hidden_size=64, num_layers=1):
    super(BiRNN, self).__init__()
    self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
    self.fc = nn.Linear(hidden_size * 2, 1)

  def forward(self, x):
    out, _ = self.rnn(x)
    return self.fc(out[:, -1, :])

class FeedforwardNN(nn.Module):
  def __init__(self, input_size):
    super(FeedforwardNN, self).__init__()
```

```
        self.fc1 = nn.Linear(input_size, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        return self.fc2(self.relu(self.fc1(x)))
```

Step 6: Define Training and Evaluation Functions

```
def train_model(model, loader, epochs=100):
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    model.train()
    for epoch in range(epochs):
        loss_epoch = 0
        for seqs, targets in loader:
            optimizer.zero_grad()
            outputs = model(seqs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            loss_epoch += loss.item()
        if (epoch + 1) % 20 == 0:
            print(f"Epoch {epoch+1}/{epochs}, Loss: {loss_epoch / len(loader):.5f}")

def evaluate_model(model, X):
    model.eval()
    with torch.no_grad():
        return model(X).numpy()
```

Step 7: Train and Predict with Both Models

```
birnn = BiRNN()
train_model(birnn, train_loader)

ffnn = FeedforwardNN(input_size=SEQ_LENGTH)
train_model(ffnn, train_loader)

pred_birnn = evaluate_model(birnn, X_test_tensor)
pred_ffnn = evaluate_model(ffnn, X_test_tensor)
```

Step 8: Inverse Transform and Plot Results

```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

pred_birnn_inv = scaler.inverse_transform(pred_birnn)
pred_ffnn_inv = scaler.inverse_transform(pred_ffnn)
y_test_inv = scaler.inverse_transform(y_test)
```

```
plt.figure(figsize=(12, 5))
plt.plot(y_test_inv, label='Actual')
plt.plot(pred_birnn_inv, label='BiRNN Prediction')
plt.plot(pred_ffnn_inv, label='FFNN Prediction')
plt.legend()
plt.title('BiRNN vs Feedforward NN on Airline Passenger Data')
plt.xlabel('Time Step')
plt.ylabel('Passengers')
plt.show()

mse_birnn = mean_squared_error(y_test_inv, pred_birnn_inv)
mse_ffnn = mean_squared_error(y_test_inv, pred_ffnn_inv)

print(f"BiRNN MSE: {mse_birnn:.3f}")
print(f"FFNN MSE: {mse_ffnn:.3f}")
```

Output: