| EXP NO: 06<br>DATE: 13/09/2025 | **BRNN VS FFNN** |
|---|---|

**AIM:** To implement a Bidirectional Recurrent Neural Network (RNN) for predicting sequences in time-series data and compare its performance with a traditional Feed-Forward Neural Network (FFNN) using the Airline Passenger Dataset.

**ALGORITHM:**

- Import necessary libraries (NumPy, Pandas, TensorFlow, etc.) and set a random seed for reproducibility.
- Upload and load the Airline Passenger dataset and extract the passenger column.
- Normalize the data using MinMaxScaler to scale values between 0 and 1.
- Create time-series windows with a fixed lookback period (e.g., 12 months).
- Split the dataset into training, validation, and testing subsets.
- Build two models: Bidirectional LSTM-based RNN model, Feed-Forward Neural Network (FFNN).
- Train both models using Mean Squared Error (MSE) loss and Adam optimizer with early stopping.
- Predict and inverse-transform the outputs to original scale.
- Compute performance metrics (MSE, MAE, RMSE, MAPE) for comparison.
- Plot true vs predicted values and training loss curves to visualize model performance.

**CODE:**

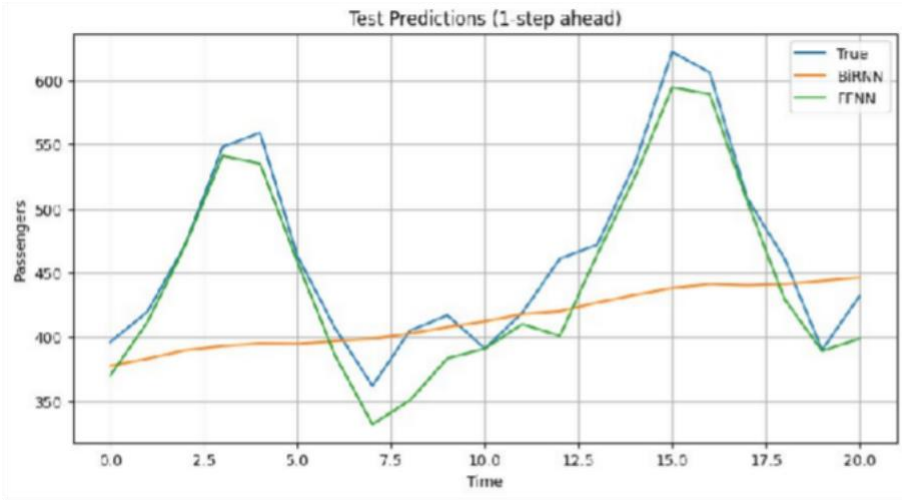
```
import numpy as np, pandas as pd, math, random, io

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.preprocessing import MinMaxScaler

from google.colab import files


SEED = 42

np.random.seed(SEED); random.seed(SEED); tf.random.set_seed(SEED)
```

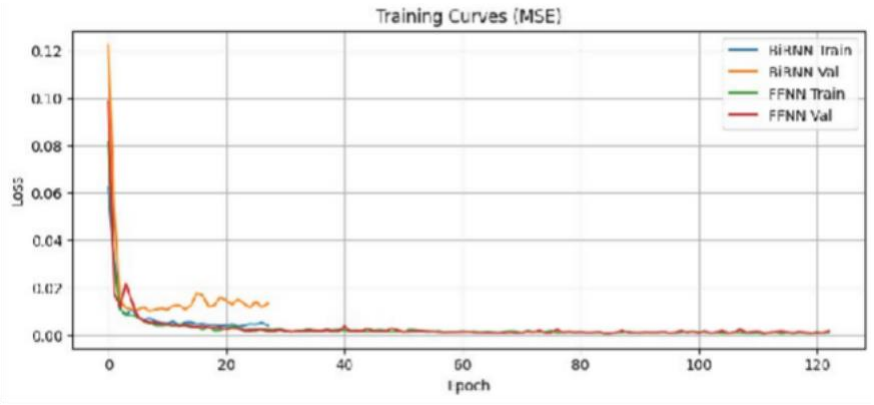
```
print("Upload CSV (e.g., AirPassengers.csv)")
uploaded = files.upload()
fname = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[fname]))
df.columns = [c.strip() for c in df.columns]
target_col = next((c for c in df.columns if c.lower()=="passengers"), None)
if target_col is None: target_col = [c for c in df.columns if
pd.api.types.is_numeric_dtype(df[c])][0]
series = df[target_col].astype("float32").to_numpy().reshape(-1,1)
lookback, horizon = 12, 1
scaler = MinMaxScaler(); series_scaled = scaler.fit_transform(series)
def make_windows(arr, lookback, horizon):
  X,y=[],[]
  for i in range(len(arr)-lookback-horizon+1):
    X.append(arr[i:i+lookback,0]); y.append(arr[i+lookback:i+lookback+horizon,0])
  return np.array(X), np.array(y)
X, y = make_windows(series_scaled, lookback, horizon)
n = len(X); n_train, n_val = int(0.7*n), int(0.15*n)
X_train, y_train = X[:n_train], y[:n_train]; X_val, y_val = X[n_train:n_train+n_val],
y[n_train:n_train+n_val]
X_test, y_test = X[n_train+n_val:], y[n_train+n_val:]
X_birnn_train, X_birnn_val, X_birnn_test = X_train[...,None], X_val[...,None],
X_test[...,None]

def build_birnn(lb):
  m = keras.Sequential([layers.Input((lb,1)),
    layers.Bidirectional(layers.LSTM(32)), layers.Dropout(0.2),
    layers.Dense(16,"relu"), layers.Dense(1)])
```

```
    m.compile("adam","mse"); return m
def build_ffnn(lb):
    m = keras.Sequential([layers.Input((lb,)), layers.Dense(64,"relu"), layers.Dropout(0.2),
        layers.Dense(32,"relu"), layers.Dense(1)])
    m.compile("adam","mse"); return m
birnn, ffnn = build_birnn(lookback), build_ffnn(lookback)

cb = [keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True,
monitor="val_loss")]
hist_birnn =
birnn.fit(X_birnn_train,y_train,validation_data=(X_birnn_val,y_val),epochs=300,batch_size=
16,verbose=0,callbacks=cb)
hist_ffnn =
ffnn.fit(X_train,y_train,validation_data=(X_val,y_val),epochs=300,batch_size=16,verbose=0,
callbacks=cb)

def inv(y_scaled): return scaler.inverse_transform(y_scaled).ravel()
def metrics(y_true, y_pred): return dict(MSE=np.mean((y_true-y_pred)**2),
MAE=np.mean(np.abs(y_true-y_pred)),
    RMSE=math.sqrt(np.mean((y_true-y_pred)**2)), MAPE=np.mean(np.abs((y_true-
y_pred)/(y_true+1e-8)))*100)
pred_birnn = inv(birnn.predict(X_birnn_test))
pred_ffnn = inv(ffnn.predict(X_test))
metrics_birnn = metrics(inv(y_test), pred_birnn)
metrics_ffnn = metrics(inv(y_test), pred_ffnn)
print("BiRNN:", metrics_birnn,"\nFFNN:", metrics_ffnn)



plt.figure(figsize=(10,5)); plt.plot(inv(y_test),label="True");
plt.plot(pred_birnn,label="BiRNN"); plt.plot(pred_ffnn,label="FFNN")
```

```
plt.title("Test Predictions (1-step ahead)"); plt.xlabel("Time"); plt.ylabel("Passengers");
plt.legend(); plt.grid(); plt.show()


plt.figure(figsize=(10,4)); plt.plot(hist_birnn.history["loss"],label="BiRNN Train");
plt.plot(hist_birnn.history["val_loss"],label="BiRNN Val")

plt.plot(hist_ffnn.history["loss"],label="FFNN Train");
plt.plot(hist_ffnn.history["val_loss"],label="FFNN Val")

plt.title("Training Curves (MSE)"); plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend();
plt.grid(); plt.show()


winner = "BiRNN" if metrics_birnn["RMSE"]<metrics_ffnn["RMSE"] else "FFNN"

print(f"\nWinner by RMSE: {winner}")
```

**OUTPUT:**




**RESULT:** The FFNN achieved lower error metrics and outperformed the Bidirectional RNN for sequence prediction tasks.