EXP-5: Train a Feed Forward Neural Network (FFNN)

Case 1: Using Pretrained Network

MLPRegressor is a Multi-Layer Perceptron (MLP) model in scikit-learn used for regression tasks — meaning it predicts continuous numeric values rather than discrete categories. It's part of sklearn.neural_network and implements a feedforward neural network trained with backpropagation.

```
from sklearn.neural_network import MLPRegressor
import numpy as np
X = [[1], [2], [3], [4]]
y = [2, 4, 6, 8]
model = MLPRegressor(
    hidden_layer_sizes=(5,),
    max_iter=5000,        # more iterations
    learning_rate_init=0.01, # higher learning rate
    solver='lbfgs',       # faster convergence for small datasets
    #solver='adam',
    #solver='sgd',
    random_state=42
)
model.fit(X, y)
preds = model.predict([[12], [7], [10]])
print(np.round(preds, 3))
```

Case 2: Using Manual Trained

```
import numpy as np
# 1. Training data
X = np.array([[1], [2], [3], [4]])  # inputs
y = np.array([[2], [4], [6], [8]])  # outputs (double the input)

# 2. Initialize weights and biases randomly
np.random.seed(1)
W1 = np.random.rand(1, 3)   # weights: input → hidden (1 input, 3 neurons in hidden)
b1 = np.zeros((1, 3))       # bias for hidden layer
W2 = np.random.rand(3, 1)   # weights: hidden → output
b2 = np.zeros((1, 1))       # bias for output layer

# 3. Activation function (ReLU)
def relu(x):
    return np.maximum(0, x)

# Derivative of ReLU
def relu_derivative(x):
    return (x > 0).astype(float)

# 4. Training loop
```

```
learning_rate = 0.01
epochs = 10

for epoch in range(epochs):
    # --- Forward pass ---
    z1 = X.dot(W1) + b1  # input to hidden layer
    a1 = relu(z1)        # activation
    z2 = a1.dot(W2) + b2 # hidden to output
    y_pred = z2          # prediction

    # --- Loss (Mean Squared Error) ---
    loss = np.mean((y_pred - y) ** 2)

    # --- Backpropagation ---
    dLoss_dy_pred = 2 * (y_pred - y) / y.size
    dLoss_dW2 = a1.T.dot(dLoss_dy_pred)
    dLoss_db2 = np.sum(dLoss_dy_pred, axis=0, keepdims=True)

    dLoss_da1 = dLoss_dy_pred.dot(W2.T)
    dLoss_dz1 = dLoss_da1 * relu_derivative(z1)
    dLoss_dW1 = X.T.dot(dLoss_dz1)
    dLoss_db1 = np.sum(dLoss_dz1, axis=0, keepdims=True)

    # --- Update weights and biases ---
    W1 -= learning_rate * dLoss_dW1
    b1 -= learning_rate * dLoss_db1
    W2 -= learning_rate * dLoss_dW2
    b2 -= learning_rate * dLoss_db2

    # Show progress every 2 steps
    if epoch % 1 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}")

# 5. Test the trained network
test_inputs = np.array([[5], [7], [10]])
z1 = test_inputs.dot(W1) + b1
a1 = relu(z1)
z2 = a1.dot(W2) + b2
predictions = z2

print("\nPredictions:")
for inp, pred in zip(test_inputs, predictions):
    print(f"{inp[0]} → {pred[0]:.2f}")
```

Experiment: Train a Feed Forward Neural Network

Aim:

To train a simple Feed Forward Neural Network (FFNN) using Keras to perform classification on a basic dataset.

Apparatus Required:

- Python installed (Anaconda/Miniconda or standard Python installation)
- Jupyter Notebook or any Python IDE
- Libraries: tensorflow (Keras), numpy, matplotlib

Theory:

A Feed Forward Neural Network (FFNN) is an artificial neural network where connections between the nodes do not form cycles. It is the simplest type of artificial neural network and consists of an input layer, one or more hidden layers, and an output layer.

Each neuron in a layer receives inputs, applies weights, adds a bias, and passes the result through an activation function. The network is trained using the backpropagation algorithm, which minimizes the loss function using optimization techniques like gradient descent.

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Generate a simple dataset
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Scale the data
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the FFNN model
model = Sequential()
model.add(Dense(10, input_shape=(2,), activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test),
verbose=1)

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

Result:

The Feed Forward Neural Network was successfully trained on the make_moons dataset. The accuracy graph demonstrated convergence over epochs, indicating the model learned to classify the data effectively.