Practical 11: Write a program to demonstrate MultiLayer Perceptron.

17CP011 Multilayer perceptron is a part of Artificial Neural Network where there is a layer of neuron units between Input and Output Layer.

In [ ]:

```
# Importing Libraries
import numpy as np
import os
import matplotlib.pyplot as plt

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
```

In [ ]:

```
# Importing digits Dataset from sklearn
digits = load_digits()
```

In [ ]:

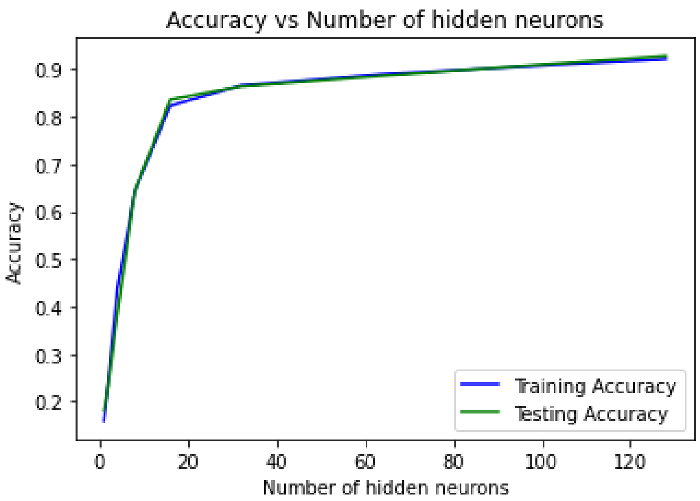
```
# split data to training and testing data
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.25)

# Standardise data, and fit only to the training data
scaler = StandardScaler()
scaler.fit(X_train)

# Apply the transformations to the data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```
# Training the data for varied hidden layer size
N = [pow(2, i) for i in range(8)]
Train_acc =[]
Test_acc =[]

for n in N:
    model = MLPClassifier(hidden_layer_sizes=n,
                          max_iter=200,
                          activation='relu',
                          solver='sgd',
                          batch_size = 32,
                          verbose=False,
                          early_stopping=True)

    model.fit(X=X_train_scaled,y=y_train)
    Train_acc.append(model.score(X_train_scaled, y_train))
    Test_acc.append(model.score(X_test_scaled, y_test))

# print("Training set score: %f" % model.score(X_train_scaled, y_train))
# print("Test set score: %f" % model.score(X_test_scaled, y_test))
```

In [ ]:

```
# Plotting for Number of neurons vs Accuracy
plt.plot(N, Train_acc, color='blue')
plt.plot(N, Test_acc, color='green')
plt.xlabel('Number of hidden neurons')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Number of hidden neurons')
plt.legend(['Training Accuracy','Testing Accuracy'])
plt.show()
```



> **Using Keras API for a tensorflow Model for Mnist Digits Dataset**

In [ ]:

```
from keras.datasets.mnist import load_data
from keras.utils import to_categorical
(X_train, y_train), (X_test, y_test) = load_data()

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

from keras import Model
from keras import layers

input_layer = layers.Input(shape=(28,28))
flatten_layer = layers.Flatten()(input_layer)
hidden_layer = layers.Dense(16, activation='relu')(flatten_layer)

# hidden_layer = layers.Dense(128, activation='relu')(hidden_layer)
# hidden_layer = layers.Dense(64, activation='relu')(hidden_layer)

output_layer = layers.Dense(10, activation='softmax')(hidden_layer)

keras_model = Model(inputs = input_layer, outputs=output_layer)
keras_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
keras_model.summary()
```

```
Model: "model_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_11 (InputLayer)        [(None, 28, 28)]          0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 16)                12560     
_________________________________________________________________
dense_25 (Dense)             (None, 10)                170       
=================================================================
Total params: 12,730
Trainable params: 12,730
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
history = keras_model.fit(X_train, y_train, epochs=20)
```

```
Epoch 1/20
1875/1875 [==============================] - 3s 1ms/step - loss: 5.9688 - accuracy: 0.278
4
Epoch 2/20
1875/1875 [==============================] - 2s 1ms/step - loss: 1.4132 - accuracy: 0.471
5
Epoch 3/20
1875/1875 [==============================] - 2s 1ms/step - loss: 1.1529 - accuracy: 0.573
0
Epoch 4/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.8725 - accuracy: 0.685
1
Epoch 5/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.7068 - accuracy: 0.755
8
Epoch 6/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.5360 - accuracy: 0.849
8
Epoch 7/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4834 - accuracy: 0.864
7
Epoch 8/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4464 - accuracy: 0.877
1
Epoch 9/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4376 - accuracy: 0.880
1
Epoch 10/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4193 - accuracy: 0.884
6
Epoch 11/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4037 - accuracy: 0.889
9
Epoch 12/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3944 - accuracy: 0.891
0
Epoch 13/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3922 - accuracy: 0.892
5
Epoch 14/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3818 - accuracy: 0.894
4
Epoch 15/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3706 - accuracy: 0.897
4
Epoch 16/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3652 - accuracy: 0.898
5
Epoch 17/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3663 - accuracy: 0.898
2
Epoch 18/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3645 - accuracy: 0.898
7
Epoch 19/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3633 - accuracy: 0.900
8
Epoch 20/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3587 - accuracy: 0.901
7
```