**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

|  |  |
| --- | --- |
| **EXERCISE** | **5. MULTIPLE LAYER PERCEPTRON** |
| **DATE** | **27.08.2024** |

# AIM:

To design and implement feed forward multiple layer perceptron (MLP) trained using Back propagation algorithm to classify the given dataset.

# DESCRIPTION:

A MLP is a class of feedforward artificial neural network trained using backpropagation

algorithms. The architecture uses, at least, three layers of nodes: an input layer, a hidden layer

and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear

activation function. Its multiple layers and non-linear activation distinguish MLP from a

linear perceptron. It can be used to handle any non-linear data. It can be used for both

classification (to predict the class label of a given tuple) and numeric prediction (to predict a

continuous-valued output). MLP are sometimes referred to as vanilla neural networks,

especially when they have a single hidden layer.

Since MLPs are fully connected, each node in one layer connects with a certain weight wij to

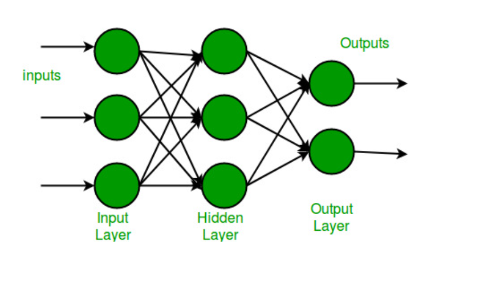
every node in the subsequent layer. Learning occurs in the perceptron by changing connection

weights after each row of data is processed, based on the amount of error in the output

compared to the expected result. This is an example of supervised learning, and is carried out

through backpropagation, a generalization of the least mean squares algorithm in the linear

perceptron.



1

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

**ALGORITHM**

**Step 1:** Select the number of input, hidden and output nodes

**Step 2:** Initialize the network and set the required hyperparameters like initial weights, hidden

nodes, learning rate, etc.

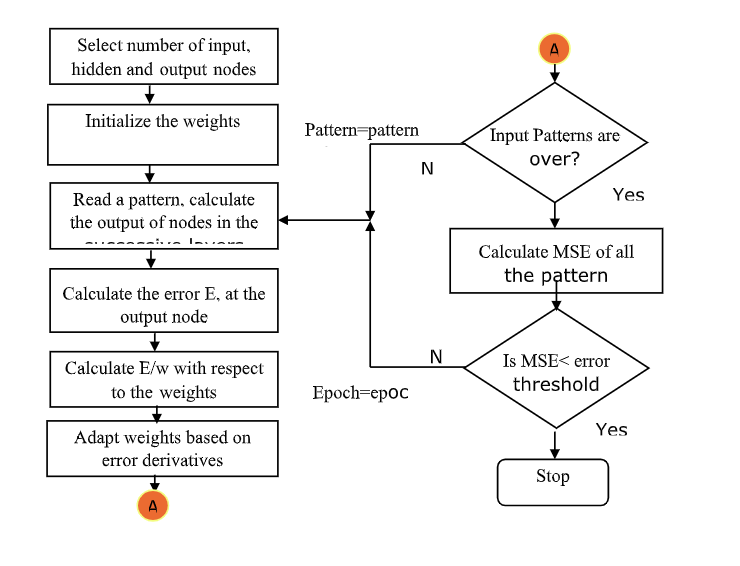
**Step 3:** Read a pattern, calculate the output of nodes in the successive layers

**Step 4:** Calculate the error E, at the output node.

**Step 5:** Propagate the error and adapt weights based on error derivatives

**Step 6:** Continue from step 3 for the required number of epochs.

**Step 7:** Evaluate the performance of the model using metrics and plot the hyperplane.



2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

**PERFORMANCE METRICS**

● Precision may be defined as the number of correct output returned by classification

model.

● Recall or Sensitivity may be defined as the number of positives returned by

classification model. It can be calculated from the confusion matrix

● Specificity, in contrast to recall, may be defined as the number of negatives returned

by the classification model.

● Support may be defined as the number of samples of the true response that lies in each

* F1 Score - This score will give us the harmonic mean of precision and recall.

Mathematically, F1 score is the weighted average of the precision and recall. The best

value of F1 would be 1 and worst would be 0. F1 score will be calculated with the help

of following formula:



**QUESTIONS**

1. Implement MLP to classify the given data set and analyse the performance of the

classifier.

import tensorflow as tf

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Feature scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

# Build the MLP model

model = tf.keras.Sequential([

    tf.keras.layers.Dense(10, input\_dim=4, activation='relu'),  # hidden layer

    tf.keras.layers.Dense(3, activation='softmax')  # output layer])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=10, validation\_split=0.1)

# Evaluate the model

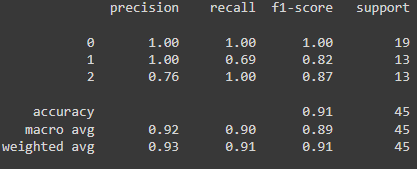
y\_pred = model.predict(X\_test)

y\_pred\_classes = tf.argmax(y\_pred, axis=1)

# Print classification report

print(classification\_report(y\_test, y\_pred\_classes))

**OUTPUT:**



2. Implement MLP for regression task

import pandas as pd

import numpy as np

from sklearn.metrics import mean\_squared\_error, r2\_score # import the missing functions

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

X = data

y = target

2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Feature scaling

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Build the MLP model

model = tf.keras.Sequential([

    tf.keras.layers.Dense(13, input\_dim=13, activation='relu'),

    tf.keras.layers.Dense(1)  # output layer for regression])

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=10, validation\_split=0.1)

# Evaluate the model

y\_pred = model.predict(X\_test)

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R-squared Score:", r2\_score(y\_test, y\_pred))

**OUTPUT:**



3. Improve the performance of the model by adjusting the hyperparameters such as

number of hidden nodes, learning rate parameter, momentum etc.

# Import the MLPRegressor

from sklearn.neural\_network import MLPRegressor

# 3. Reinitialize and train the MLP regressor with different hyperparameters

mlp\_optimized = MLPRegressor(hidden\_layer\_sizes=(20, 20), max\_iter=2000, learning\_rate\_init=0.01, random\_state=42)

mlp\_optimized.fit(X\_train, y\_train)

# Predict and analyze performance

y\_pred\_optimized = mlp\_optimized.predict(X\_test)

2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

# Use appropriate metrics for regression

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred\_optimized))

print("R-squared Score:", r2\_score(y\_test, y\_pred\_optimized))

**OUTPUT:**



4. Implement SVM to classify the given data set and analyse the performance of the

classifier.

from sklearn.svm import SVR # Changed SVC to SVR for regression

# Initialize and train the SVM regressor

svm = SVR(kernel='linear') # Changed SVC to SVR for regression

svm.fit(X\_train, y\_train)

# Predict and analyze performance

y\_pred\_svm = svm.predict(X\_test)

# Use appropriate metrics for regression

from sklearn.metrics import mean\_squared\_error, r2\_score # import metrics for regression

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred\_svm))

print("R-squared Score:", r2\_score(y\_test, y\_pred\_svm))

**OUTPUT:**



5. Implement SVM for regression task

from sklearn.svm import SVR

# Initialize and train the SVM regressor

svm\_reg = SVR(kernel='linear')

svm\_reg.fit(X\_train, y\_train)

# Predict and analyze performance

y\_pred\_svm\_reg = svm\_reg.predict(X\_test)

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred\_svm\_reg))

print("R^2 Score:", r2\_score(y\_test, y\_pred\_svm\_reg))

2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**

**20CS2032L – MACHINE LEARNING TECHNIQUES**

**URK22AI1015**

**OUTPUT:**



6. Improve the performance of the model by adjusting the hyperparameters such as

number of hidden nodes, learning rate parameter, momentum etc.

# Reinitialize and train the SVM regressor with different hyperparameters - Use SVR for regression

svm\_optimized = SVR(kernel='rbf', C=1, gamma=0.1) # Change model to SVR

svm\_optimized.fit(X\_train, y\_train)

# Predict and analyze performance

y\_pred\_svm\_optimized = svm\_optimized.predict(X\_test)

# Evaluate the model - Use metrics suitable for regression

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred\_svm\_optimized))

print("R-squared Score:", r2\_score(y\_test, y\_pred\_svm\_optimized)) # Change metrics to suit regression

**OUTPUT:**



2

Ex. 05 | **MULTIPLE LAYER PERCEPTRON**