

EXERCISE	5. MULTIPLE LAYER PERCEPTRON
DATE	27.08.2024

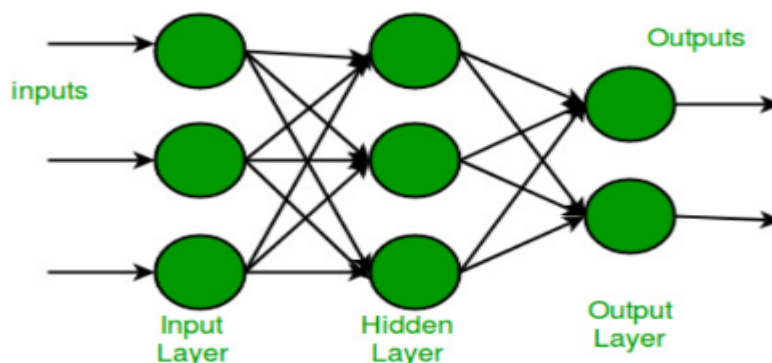
**AIM:**

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  $w_{ij}$  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.



**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.

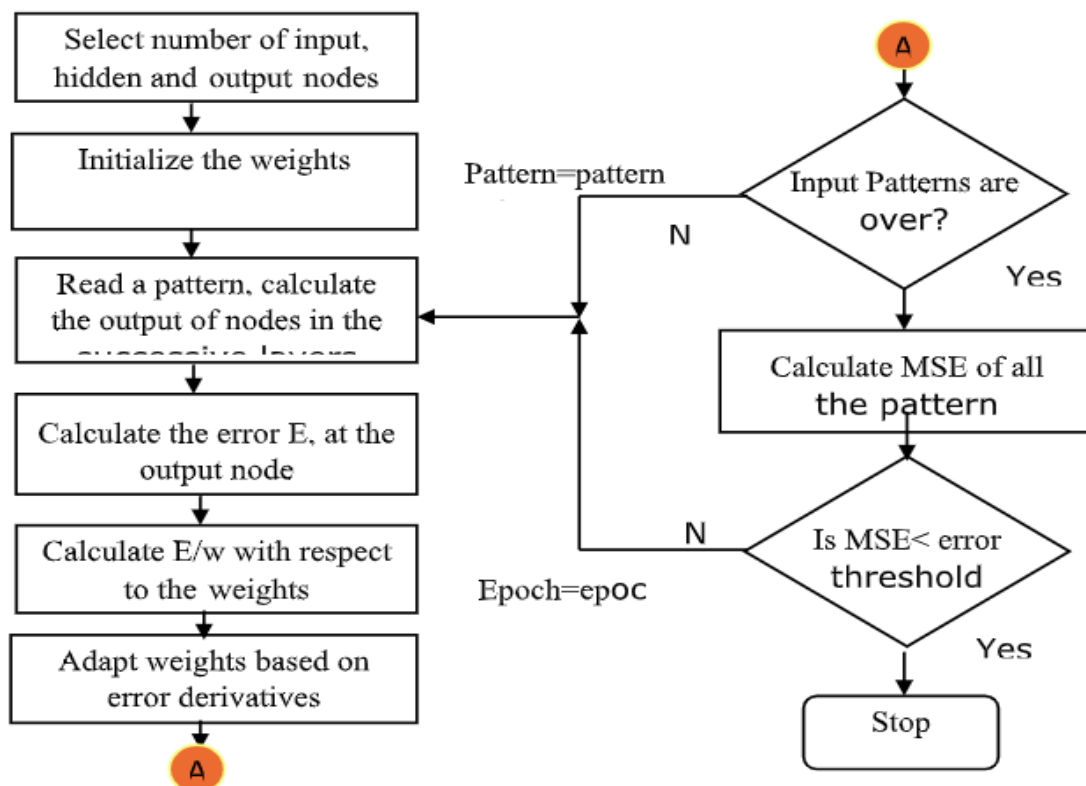
**Step 3:** Read a pattern, calculate the output of nodes in the successive layers

**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.



**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:

$$F1 = 2 * (precision * recall) / (precision + recall)$$

**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)
```

```
# 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:**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	0.69	0.82	13
2	0.76	1.00	0.87	13
accuracy			0.91	45
macro avg	0.92	0.90	0.89	45
weighted avg	0.93	0.91	0.91	45

**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
```

```
# 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)

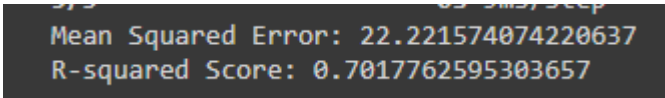
# Feature scaling
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 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:**

```
Mean Squared Error: 22.221574074220637
R-squared Score: 0.7017762595303657
```

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)
```

```
# 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:**

```
Mean Squared Error: 12.59321291413075
R-squared Score: 0.8309932929486097
```

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:**

```
Mean Squared Error: 24.971388169372414
R-squared Score: 0.6648724901433158
```

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))
```

**OUTPUT:**

```
Mean Squared Error: 24.971388169372414
R^2 Score: 0.6648724901433158
```

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:**

```
Mean Squared Error: 27.38779858377615
R-squared Score: 0.6324431514346205
```