Bharatiya Vidya Bhavan’s
Sardar Patel Institute of Technology
Bhavan’s Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

BE-ETRX B **Sub- AIML Lab**
Name: Shubham Sawant **UID :2019110050**

Name of the Experiment: Multiple Layer Perceptron

Objective : To explore the multi-layer perceptron algorithm using back-propagation
Outcomes:
1. Identifying the algorithms for non-linear classification
2. Build the Multi-layer perceptron ML Model for the given data
3. Draw various plots and interpret them
4. Learn to use in built functions from libraries for training the NN

System Requirements: Windows with MATLAB

Dataset Description:

Number of Instances: 210

Number of Attributes: 7

Attribute Information:

1. area A,
2. perimeter P,
3. compactness C = 4*pi*A/P^2,
4. length of kernel,
5. width of kernel,
6. asymmetry coefficient
7. length of kernel groove.

Code:

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

BE-ETRX B **Sub- AIML Lab**
Name: Shubham Sawant **UID :2019110050**

+ Code + Text

```
[44] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Backprop on the Seeds Dataset
from random import seed
from matplotlib import pyplot as plt
from csv import reader
from math import exp
```

```
[46] def load_csv(filename):
         dataset = list()
         with open(filename, 'r') as file:
             csv_reader = reader(file)
             for row in csv_reader:
                 if not row:
                     continue
                 dataset.append(row)
         return dataset
```

```
[47] def str_column_to_float(dataset, column):
         for row in dataset:
             row[column] = float(row[column].strip())
```

+ Code + Text RAM Disk

```
[48]
# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup
```

```
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    print("Enter input layer to hidden layer weights and bias : ")
    hidden_layer = [{'weights': [float(input("Enter value for {}. row and {}. column: ".format(j + 1, i + 1))) for i in
                     range(n_inputs + 1)]} for j in range(n_hidden)]
    network.append(hidden_layer)
    print("Enter hidden layer to output layer weights and bias : ")
    output_layer = [{'weights': [float(input("Enter value for {}. row and {}. column: ".format(j + 1, i + 1))) for i in range(n_hidden + 1)]} for j in range(n_outputs)]
    network.append(output_layer)
    i = 1
    print("\n The initialised Neural Network:\n")
    for layer in network:
        j = 1
        for sub in layer:
            print("\n Layer[%d] Node[%d]:\n" % (i, j), sub)
            j = j + 1
        i = i + 1
    return network
```

```
[ ] # Calculate neuron activation (net) for an input

    def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights) - 1):
            activation += weights[i] * inputs[i]
        return activation
```

```
[51] def transfer(activation):
         return 1.0 / (1.0 + exp(-activation))
```

```
[52] # Forward propagate input to a network output
     def forward_propagate(network, row):
         inputs = row
         for layer in network:
             new_inputs = []
             for neuron in layer:
                 activation = activate(neuron['weights'], inputs)
                 neuron['output'] = transfer(activation)
                 new_inputs.append(neuron['output'])
             inputs = new_inputs
         return inputs
```

```
[53] def transfer_derivative(output):
       return output * (1.0 - output)
```

```
[54] # Backpropagate error and store in neurons
     def backward_propagate_error(network, expected):
         for i in reversed(range(len(network))):
             layer = network[i]
             errors = list()

             if i != len(network) - 1:
                 for j in range(len(layer)):
                     error = 0.0
                     for neuron in network[i + 1]:
                         error += (neuron['weights'][j] * neuron['delta'])
                     errors.append(error)
             else:
                 for j in range(len(layer)):
                     neuron = layer[j]
                     errors.append(expected[j] - neuron['output'])

             for j in range(len(layer)):
                 neuron = layer[j]
                 neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

```
# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
```

[56]

```
# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    print("\n Network Training Begins:\n")
    errors = []
    epoch_no = []
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        errors.append(sum_error)
        epoch_no.append(epoch)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    plt.plot(errors)
    plt.show()
    print("\n Network Training Ends:\n")
```

[57]

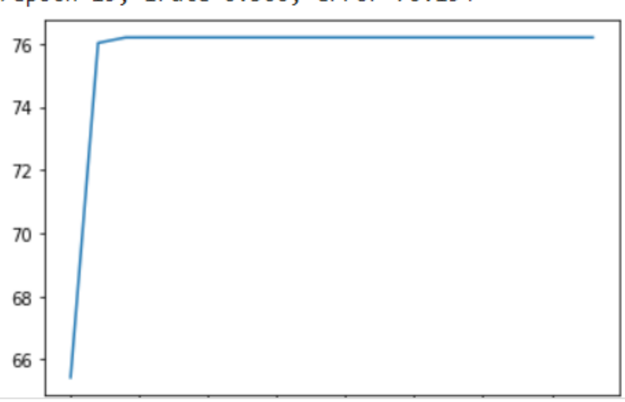
```
# Test training backprop algorithm
seed(2)
filename = '/content/drive/MyDrive/Colab Notebooks/wheat.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
  str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)

print("\n The input Data Set :\n", dataset)
n_inputs = len(dataset[0]) - 1
print("\n Number of Inputs :\n", n_inputs)
n_outputs = len(set([row[-1] for row in dataset]))
print("\n Number of Outputs :\n", n_outputs)
# Network Initialization
network = initialize_network(n_inputs, 2, n_outputs)

# Training the Network
train_network(network, dataset, 0.5, 20, n_outputs)

print("\n Final Neural Network :")

i = 1
for layer in network:
    j = 1
    for sub in layer:
        print("\n Layer[%d] Node[%d]:\n" % (i, j), sub)
        j = j + 1
    i = i + 1
```

BE-ETRX B **Sub- AIML Lab**
Name: Shubham Sawant **UID :2019110050**

Output:

```
The input Data Set :
[[15.26, 14.84, 0.871, 5.763, 3.312, 2.221, 5.22, 2], [14.88, 14.57, 0.8811, 5.554, 3.333, 1.018, 4.956, 2], [14.29, 14.0

Number of Inputs :
7

Number of Outputs :
3
Enter input layer to hidden layer weights and bias :
Enter value for 1. row and 1. column: 0.5
Enter value for 1. row and 2. column: 0.6
Enter value for 1. row and 3. column: 0.7
Enter value for 1. row and 4. column: 0.8
Enter value for 1. row and 5. column: 0.9
Enter value for 1. row and 6. column: 0.4
Enter value for 1. row and 7. column: 0.6
Enter value for 1. row and 8. column: 0.5
Enter value for 2. row and 1. column: 0.4
Enter value for 2. row and 2. column: 0.6
Enter value for 2. row and 3. column: 0.8
Enter value for 2. row and 4. column: 0.3
Enter value for 2. row and 5. column: 0.4
Enter value for 2. row and 6. column: 0.5
Enter value for 2. row and 7. column: 0.6
Enter value for 2. row and 8. column: 0.3
Enter hidden layer to output layer weights and bias :
Enter value for 1. row and 1. column: 0.8
Enter value for 1. row and 2. column: 0.9
Enter value for 1. row and 3. column: 0.4
Enter value for 2. row and 1. column: 0.6
Enter value for 2. row and 2. column: 0.6
Enter value for 2. row and 3. column: 0.4
Enter value for 3. row and 1. column: 0.5
Enter value for 3. row and 2. column: 0.6
```

```
Enter value for 3. row and 3. column: 0.2

 The initialised Neural Network:


 Layer[1] Node[1]:
 {'weights': [0.5, 0.6, 0.7, 0.8, 0.9, 0.4, 0.6, 0.5]}

 Layer[1] Node[2]:
 {'weights': [0.4, 0.6, 0.8, 0.3, 0.4, 0.5, 0.6, 0.3]}

 Layer[2] Node[1]:
 {'weights': [0.8, 0.9, 0.4]}

 Layer[2] Node[2]:
 {'weights': [0.6, 0.6, 0.4]}

 Layer[2] Node[3]:
 {'weights': [0.5, 0.6, 0.2]}

 Network Training Begins:

>epoch=0, lrate=0.500, error=65.426
>epoch=1, lrate=0.500, error=76.020
>epoch=2, lrate=0.500, error=76.193
>epoch=3, lrate=0.500, error=76.194
>epoch=4, lrate=0.500, error=76.194
>epoch=5, lrate=0.500, error=76.194
>epoch=6, lrate=0.500, error=76.194
>epoch=7, lrate=0.500, error=76.194
>epoch=8, lrate=0.500, error=76.194
>epoch=9, lrate=0.500, error=76.194
>epoch=10, lrate=0.500, error=76.194
>epoch=11, lrate=0.500, error=76.194
>epoch=12, lrate=0.500, error=76.194
>epoch=19, lrate=0.500, error=76.194
```



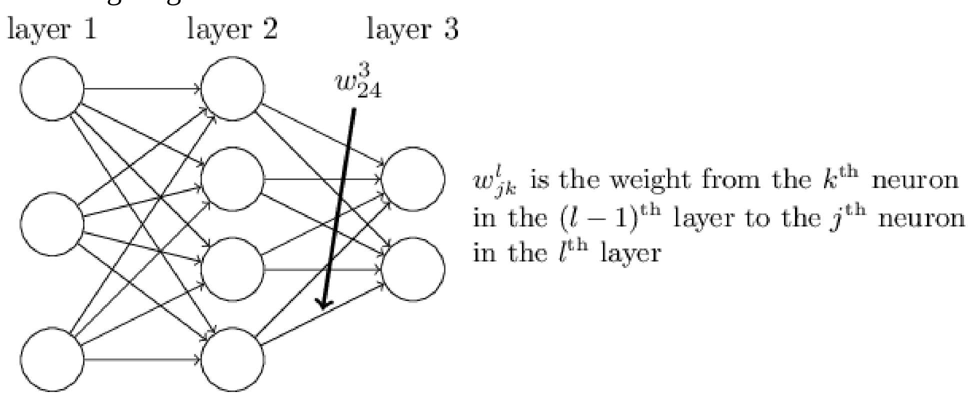
BE-ETRX B **Sub- AIML Lab**

Name: Shubham Sawant **UID :2019110050**

Application:

1. How does backpropagation work?

   Backpropagation is a process involved in training a neural network. It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.) Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.



2. Loss function

   A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs. It's a method of evaluating how well your algorithm models your dataset. If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll output a lower number.

3. Why do we need backpropagation?

   Backpropagation algorithms are used extensively to train feedforward neural networks in areas such as deep learning. They efficiently compute the gradient of the loss function with respect to the network weights. Backpropagation does not require any parameters to be set, except the number of inputs. Backpropagation is a flexible method because no prior knowledge of the network is required.

Conclusion:

1. Therefore, we can conclude that we identified the problem and worked towards creating a solution for the problem. We used sklearn to divide the data into test and train sets randomly.
2. We utilised TensorFlow to create the Multiple layer Perceptron network. We have used Relu and Sigmoidal activations here.
3. We got 76% test accuracy and the train losses kept on decreasing. We ran it for 20 epochs.
4. Thus, we learned how backpropagation works in a multiple layer perceptron.
5. We also implemented the algorithm and calculated our results.