**Date of the Session:__/__/__** **Time of the Session:____to______**

**SKILLING -1:**

1 .Design a one-layer Perceptron network to classify 4 classes.
Assume that the data set includes 25 samples and each sample is 10 dimensional. Print the weights and biases of the model

```
import numpy as np
class Perceptron:
    def __init__(self, num_classes,num_dimensions):
        self.num_classes = num_classes
        self.num_dimensions = num_dimensions
        self.weights = np.random.randn(num_classes,num_dimensions)
        self.biases = np.random.randn(num_classes)
    def set_weights(self,weights,biases):
        self.weights = weights
        self.biases = biases

    def train(self, X,y,alpha, num_iterations):
        for i in range(num_iterations):
            activations = np.dot(x, self.weights.T)+ self.biases
            y_pred = np.argmax (activations,axis = 1)
            error = y - y_pred
            self.weights = self.weights + alpha * np.dot (error,x)
            self.biases += alpha * error.sum(axis = 0)
```

```
def predict(self, x):
    activations = np.dot(x, self.weights.T) + self.biases
    return np.argmax(activations, axis=1)

num_classes = 4
num_samples = 25
num_dimensions = 10
alpha = 0.1
num_iterations = 1000
X = np.random.randn(num_samples, num_dimensions)
y = np.random.randn(0, num_classes, num_samples)
perceptron = Perceptron(num_classes, num_dimensions)
perceptron.train(x, y, alpha, num_iterations)
print("Weights: ", perceptron.weights)
print(" Biases:", perceptron.biases)
```

Output:-

Weights: [[-333.87 711.65 1224.36 -2022.58
591.34 -299.671 -208.07 -1246.65
-292.44 847.5626]

[-333.75 708.37 1224.33 -2020.16
592.11 -299.22 -207.79 -1246.62
-290.61 845.72]]

Biases: [701.063 701.106 698.628 698.685]

| Comment of the Evaluator (if Any) | Evaluator's Observation |
|---|---|
| | Marks Secured:______out of___________ |
| | Full Name of the Evaluator: |
| | Signature of the Evaluator Date of Evaluation: |

**Date of the Session:** ___/___/___ **Time of the Session:** ____to______

**SKILLING-2:**

Implement a feedforward neural network and write the backpropagation code for training the network. Use numpy for all matrix/vector operations. You are not allowed to use any automatic differentiation packages. This network will be trained and tested using the XOR input with one output And also with Fashion-MNIST dataset with each image size as 28 x 28. Train the MNIST model to classify the images into one of 10 classes.

```
import numpy as np

class NeuralNetwork:
    def __init__(self,input_size,hidden_size,output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.W1 = np.random.randn(self.input_size,
                                  self.hidden_size)
        self.b1 = np.zeros(self.hidden_size)
        self.W2 = np.random.randn(self.hidden_size,
                                  self.output_size)
        self.b2 = np.zeros(self.output_size)
    def sigmoid(self, x):
        return 1/(1+np.exp(-x))
    def sigmoid_derivative(self, x):
        return x*(1-x)
```

```
def  forward(self, x):
    self.z1 = np.dot(x, self.W1) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.a2 = self.sigmoid(self.z2)
    return self.a2

def  backward(self, X, y, learning_rate):
    m = len(x)
    self.dz2 = self.a2 - y
    self.dW2 = np.dot(self.a1.T, self.dz2)/m
    self.db2 = np.sum(self.dz2, axis=0)/m
    self.dz1 = np.dot(self.dz2, self.W2.T) * self.sigmoid_derivative
                    (self.a1)
    self.dW1 = np.dot(X.T, self.dz1)/m
    self.db1 = np.sum(self.dz1, axis=0)/m

    self.W1 -= learning_rate * self.dW1
    self.b1 -= learning_rate * self.db1
    self.W2 -= learning_rate * self.dW2
    self.b2 -= learning_rate * self.db2

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])
```



```
nn = NeuralNetwork(input_size=2, hidden_size=3,
                output_size=1)
```

**Output:**

[[0.00439025]
[0.97262309]
[0.97321272]
[0.0350386]]

| Comment of the Evaluator (if Any) | Evaluator's Observation<br>Marks Secured:______out of__________<br>Full Name of the Evaluator:<br>Signature of the Evaluator Date of Evaluation: |
|---|---|

**Date of the Session:___/___/___** **Time of the Session:____to_______**

**SKILLING-3:**

Implement a 2-class classification neural network with two hidden layers Use units with a non-linear activation function, such as tanh. Compute the cross entropy loss, Implement forward and backward propagation using python functions Use Planar data from Kaggle.

```
from sklearn.model_selection import train_test_split
from sklearn import datasets
import numpy as np
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.W1 = np.random.randn(self.input_size,
                                  self.hidden_size)
        self.b1 = np.zeros(self.hidden_size)
        self.W2 = np.random.randn(self.hidden_size,
                                  self.output_size)
        self.b2 = np.zeros(self.output_size)
    def tanh(self, x):
        return np.tanh(x)
    def tanh_derivative(self, x):
        return 1 - np.tanh(x)**2
```

```
def softmax(self,x):
    return exp_x/np.sum(exp_x, axis=1, keepdims=True)
def cross_entropy_loss(self, y_pred, y_true):
    m = len(y_true)
    loss = -(1/m) * np.sum(y_true * np.log(y_pred))
    return loss

def forward(self, x):
    self.z1 = np.dot(x, self.W1) + self.b1
    self.a1 = self.tanh(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.a2 = self.softmax(self.z2)
    return self.a2

def backward(self, X, y_true, learning_rate):
    m = len(x)
    self.dz2 = self.a2 - y_true
    self.dW2 = np.dot(self.a1.T, self.dz2) /m
    self.db2 = np.sum(self.dz2, axis=0)/m
    self.dz1 = np.dot(self.dz2, self.W2.T) * self.tanh_derivative
                                                    (self.a1)
    self.dW1 = np.dot(x.T, self.dz1)/m
    self.db1 = np.sum(self.dz1, axis=0)/m
    self.W1 -= learning_rate * self.dW1
    self.b1 -= learning_rate * self.db1
    self.W2 -= learning_rate * self.dW2
    self.b2 -= learning_rate * self.db2
```

```
iris = datasets.load_iris()
X = iris.data[iris.target != 2]
y = iris.target[iris.target != 2]
y_onehot = np.zeros((y.shape[0], 2))
```

**Output:**

```
Accuracy: 90.0
[[0.99     0.0048]
 [0.993    0.0063]
 [0.994    0.005]
 [0.991    0.0083]
 [0.0066   0.993]
 [0.0063   0.996]
 [0.00666  0.9933]]
```

| Comment of the Evaluator (if Any) | Evaluator's Observation |
|---|---|
| | Marks Secured:______ out of__________ |
| | Full Name of the Evaluator: |
| | Signature of the Evaluator Date of Evaluation: |