```python
import numpy as np
import matplotlib.pyplot as plt

# Class to create a neural network with single neuron
class NeuralNetwork(object):

    def __init__(self, num_params=blank1):
        # Using seed to make sure it'll generate same weights in every run
        # np.random.seed(1)
        # 3x1 Weight matrix
        self.weight_matrix = 2 * np.random.random((num_params+1, 1)) - 1
                                                #random weights between -1 and 1
        self.l_rate = 1

    # hard_limiter as activation fucntion
    def hard_limiter(self, x):
        outs=np.zeros(x.shape)
        outs[x>0]=1
        return outs

    # forward propagation
    def forward_propagation(self, inputs):
        outs=np.dot(inputs, self.weight_matrix)
        return self.hard_limiter(outs)

    # training the neural network.
    def train(self, train_inputs, train_outputs,
                              num_train_iterations=1000):

        # Number of iterations we want to perform for this set of input.
        for iteration in range(num_train_iterations):
            #updating the perceptron base on the misclassified examples
            for i in range(train_inputs.shape[0]):
              pred_i = self.pred(train_inputs[i,:])
              if pred_i!=train_outputs[i]:
                output = self.forward_propagation(train_inputs[i,:])
                # Calculate the error in the output.
                error = train_outputs[i] - output
                adjustment = self.l_rate*error*train_inputs[i]
                # Adjust the weight matrix
                self.weight_matrix[:,0] += adjustment


    #predicting the classes of new data points
    def pred(self,inputs):
      preds=self.forward_propagation(inputs)
      return preds


features=np.array([[2,2,1], [2,1,1], [1,2,1], [1,1,1]])
```

```
print(features)
labels=np.array([0,0,1,1])
print(labels)
classes=[0,1]
```

```
[[2 2 1]
 [2 1 1]
 [1 2 1]
 [1 1 1]]
[0 0 1 1]
```

```
bias = np.ones((features.shape[0],1)) #expanding the feature space by adding
                                        #the bias vector
print(bias)
print(bias.shape)
features=np.append(bias, features, axis=1)
print(features)
print(features.shape)
```

```
[[1.]
 [1.]
 [1.]
 [1.]]
(4, 1)
[[1. 2. 2. 1.]
 [1. 2. 1. 1.]
 [1. 1. 2. 1.]
 [1. 1. 1. 1.]]
(4, 4)
```

```
neural_network = NeuralNetwork()
print ('Random weights at the start of training')
print (neural_network.weight_matrix)
```

```
Random weights at the start of training
[[-0.9840514 ]
 [ 0.79595462]
 [-0.57000711]
 [ 0.46889876]]
```

```
neural_network = NeuralNetwork(blank2)
print ('Random weights at the start of training')
print (neural_network.weight_matrix)
neural_network.train(features, labels, blank3)

print ('New weights after training')
print (neural_network.weight_matrix)

# Test the neural network with training data points.
print ("Testing network on training data points ->")
print (neural_network.pred(features))
```

```
Random weights at the start of training
[[0.8878951 ]
 [0.46978031]
 [0.52788076]
 [0.63517545]]
New weights after training
[[ 0.8878951 ]
 [-1.53021969]
 [ 0.52788076]
 [ 0.63517545]]
Testing network on training data points ->
[[0.]
 [0.]
 [1.]
 [1.]]
```

✓ 0s completed at 12:17 PM