1. Implement a 3 layer multilayer perceptron neural network with 2-4-1 architecture and solve the EX-OR classification problem usingbackpropagation algorithm. Note: Don't consider bias at anyneuron.Use Sigmoid activation functionat every neuron.Train for 100 epochs. Plot the convergence graph.

In [33]:

```
import matplotlib.pyplot as plt
import numpy as np
X=np.array(([0,0],[0,1],[1,0],[1,1]), dtype=float)
y=np.array(([0],[1],[1],[0]), dtype=float)
```

In [2]:

```
def sigmoid(t):
    '''This will return the sigmoid value of the function'''
    return 1/(1+np.exp(-t))
```

In [3]:

```
def sigmoid_derivative(d):
    return d * (1 - d)
```

In [10]:

```
class NeuralNetworkSigmoid:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(self.input.shape[1],4)
        self.weights2 = np.random.rand(4,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2

    def backprop(self):
        '''Back propagation of the final hidden layers to initial layers'''
        derv_weights2 = np.dot(self.layer1.T, 2*(self.y -self.output)*sigmoid_derivativ
        derv_weights1 = np.dot(self.input.T, np.dot(2*(self.y -self.output)*sigmoid_der

        self.weights1 += derv_weights1
        self.weights2 += derv_weights2

    def train(self, X, y):
        self.output = self.feedforward()
        self.backprop()
```

In [13]:

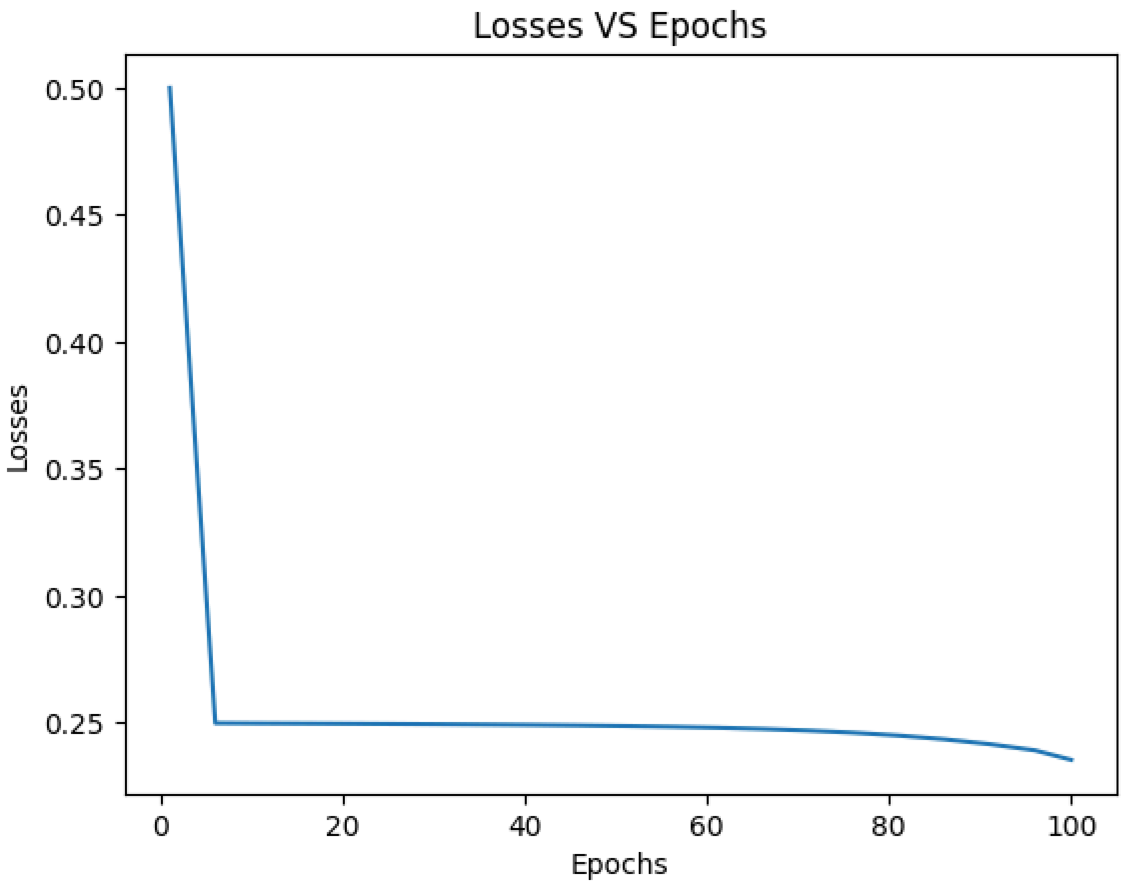
```
model=NeuralNetworkSigmoid(X,y)
iterations = 100
losses = []
ep = []
for i in range(iterations):
    if i % 5 == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
```

```
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(100)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.show()
```

```
For iteration # 100
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.45741481]
 [0.51573353]
 [0.53633454]
 [0.53165812]]
Loss:
0.23534708289181813
```



1. Implement a 3 layer multilayer perceptron neural network with 2-6-1 architecture and solvethe EX-OR classification problem usingbackpropagation algorithm. Note: Don't consider bias at anyneuron.Use Sigmoid activation functionat every neuron.Train for 100 epochs. Plot the convergence graph.

In [34]:
```
class NeuralNetworkSigmoid:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(self.input.shape[1],6)
        self.weights2 = np.random.rand(6,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2

    def backprop(self):
        '''Back propagation of the final hidden layers to initial layers'''
        derv_weights2 = np.dot(self.layer1.T, 2*(self.y -self.output)*sigmoid_derivativ
        derv_weights1 = np.dot(self.input.T, np.dot(2*(self.y -self.output)*sigmoid_der

        self.weights1 += derv_weights1
        self.weights2 += derv_weights2

    def train(self, X, y):
```

```
        self.output = self.feedforward()
        self.backprop()
```

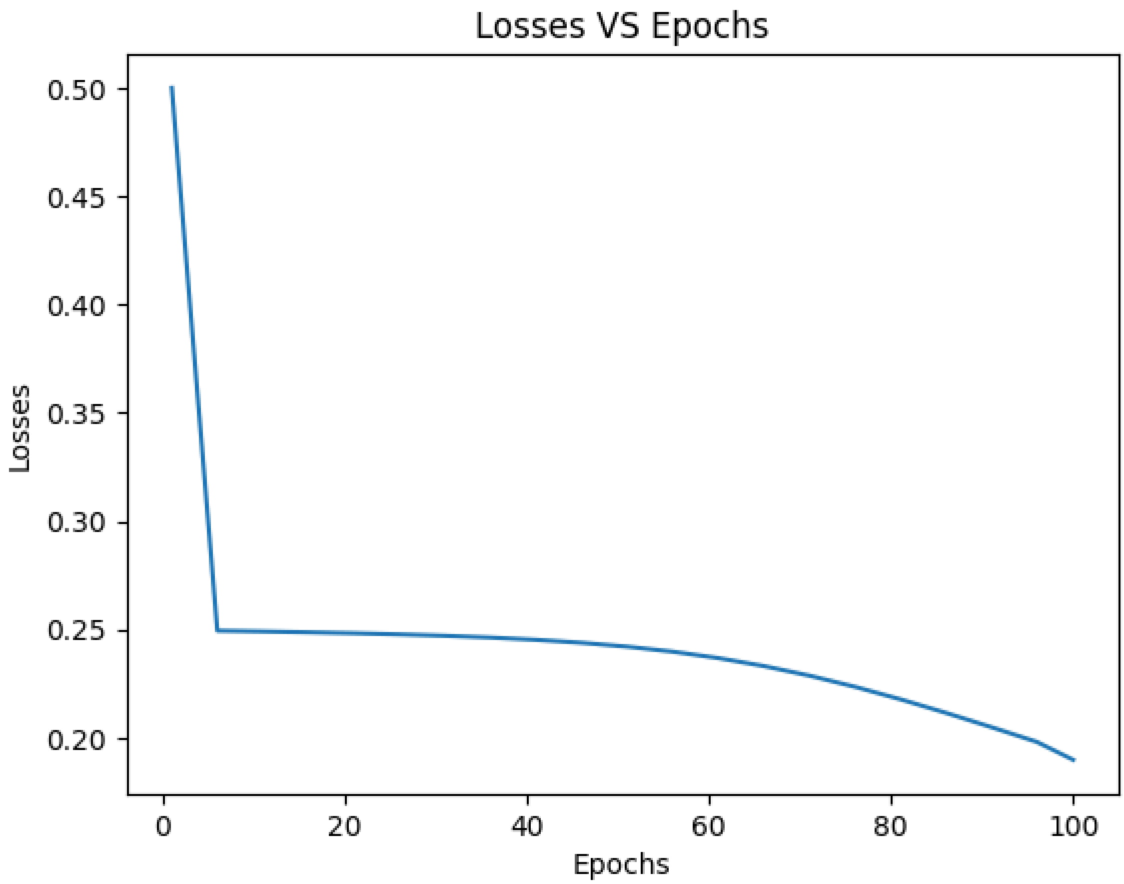
In [35]:

```
model=NeuralNetworkSigmoid(X,y)
iterations = 100
losses = []
ep = []
for i in range(iterations):
    if i % 5 == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(100)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.show()
```

```
For iteration # 100
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.37430272]
 [0.59784729]
 [0.55205842]
 [0.5069133 ]]
Loss:
0.18986052032027415
```



1. Implement a 3 layer multilayer perceptron neural network with 2-6-1 architecture and solvethe EX-OR classification problem usingbackpropagation algorithm. Note: Consider bias at everyneuron.Use Sigmoid activation functionat every neuron.Train for 100 epochs. Plot the convergence graph.

In [39]:

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

In [96]:

```
class NeuralNetworkSigmoid:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(2,6)
        self.weights2 = np.random.rand(6,1)
        self.bias1 = np.random.rand(1,6)
        self.bias2 = np.random.rand(1,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = sigmoid(np.dot(self.input, self.weights1) + self.bias1)
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2) + self.bias2)
        return self.layer2

    def backprop(self):
        '''Backpropagation of the final hidden layers to initial layers'''
        error = self.y - self.output
```

```
        d_weights2 = np.dot(self.layer1.T, 2 * error * sigmoid_derivative(self.output))
        d_bias2 = np.sum(2 * error * sigmoid_derivative(self.output), axis=0, keepdims=

        error_hidden_layer = np.dot(2 * error * sigmoid_derivative(self.output), self.w

        d_weights1 = np.dot(self.input.T, error_hidden_layer * sigmoid_derivative(self.
        d_bias1 = np.sum(error_hidden_layer * sigmoid_derivative(self.layer1), axis=0)

        self.weights1 += d_weights1
        self.weights2 += d_weights2
        self.bias1 += d_bias1
        self.bias2 += d_bias2


    def train(self, X, y):
        self.output = self.feedforward()
        self.backprop()
```

In [97]:

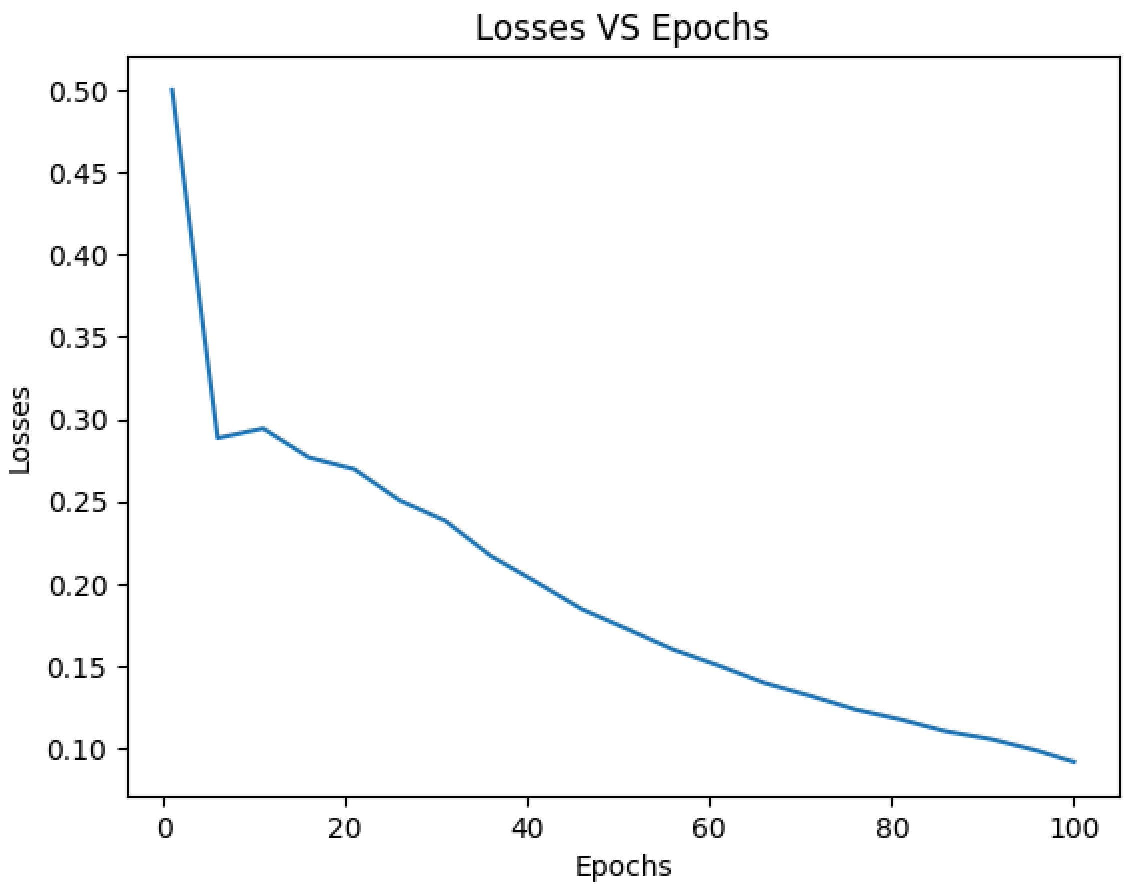
```
model=NeuralNetworkSigmoid(X,y)
iterations = 100
losses = []
ep = []
for i in range(iterations):
    if i % 5 == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(iterations)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.show()
```

```
For iteration # 100
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]
 [0. 0.]
 [1. 0.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]
 [0.]
```

```
 [1.]]
Predicted Output:
[[0.07397883]
 [0.42734192]
 [0.78915378]
 [0.3518793 ]
 [0.07397883]
 [0.78915378]]
Loss:
0.09193571686383777
```



1. Implement a 3 layer multilayer perceptron neural network with 2-6-1 architecture and solvethe EX-OR classification problem usingbackpropagation algorithm. Note: Consider bias at everyneuron.Use ReLUactivation functionat hidden layer neurons andSigmoid activation function at output layer neuron.Train for 100 epochs. Plot the convergence graph.

In [90]:

```
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

In [129…

```
class NeuralNetwork:
    def __init__(self, x,y):
```

```
        self.input = x
        self.weights1= np.random.rand(2,6)
        self.weights2 = np.random.rand(6,1)
        self.bias1 = np.random.rand(1,6)
        self.bias2 = np.random.rand(1,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = relu(np.dot(self.input, self.weights1) + self.bias1)
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2) + self.bias2)
        return self.layer2

    def backprop(self):
        '''Backpropagation of the final hidden layers to initial layers'''
        error = self.y - self.output

        d_weights2 = np.dot(self.layer1.T, 2 * error * relu_derivative(self.output))
        d_bias2 = np.sum(2 * error * relu_derivative(self.output), axis=0, keepdims=Tru

        error_hidden_layer = np.dot(2 * error * sigmoid_derivative(self.output), self.w

        d_weights1 = np.dot(self.input.T, error_hidden_layer * sigmoid_derivative(self.
        d_bias1 = np.sum(error_hidden_layer * sigmoid_derivative(self.layer1), axis=0)

        self.weights1 += d_weights1
        self.weights2 += d_weights2
        self.bias1 += d_bias1
        self.bias2 += d_bias2

    def train(self, X, y):
        self.output = self.feedforward()
        self.backprop()
```

In [141…

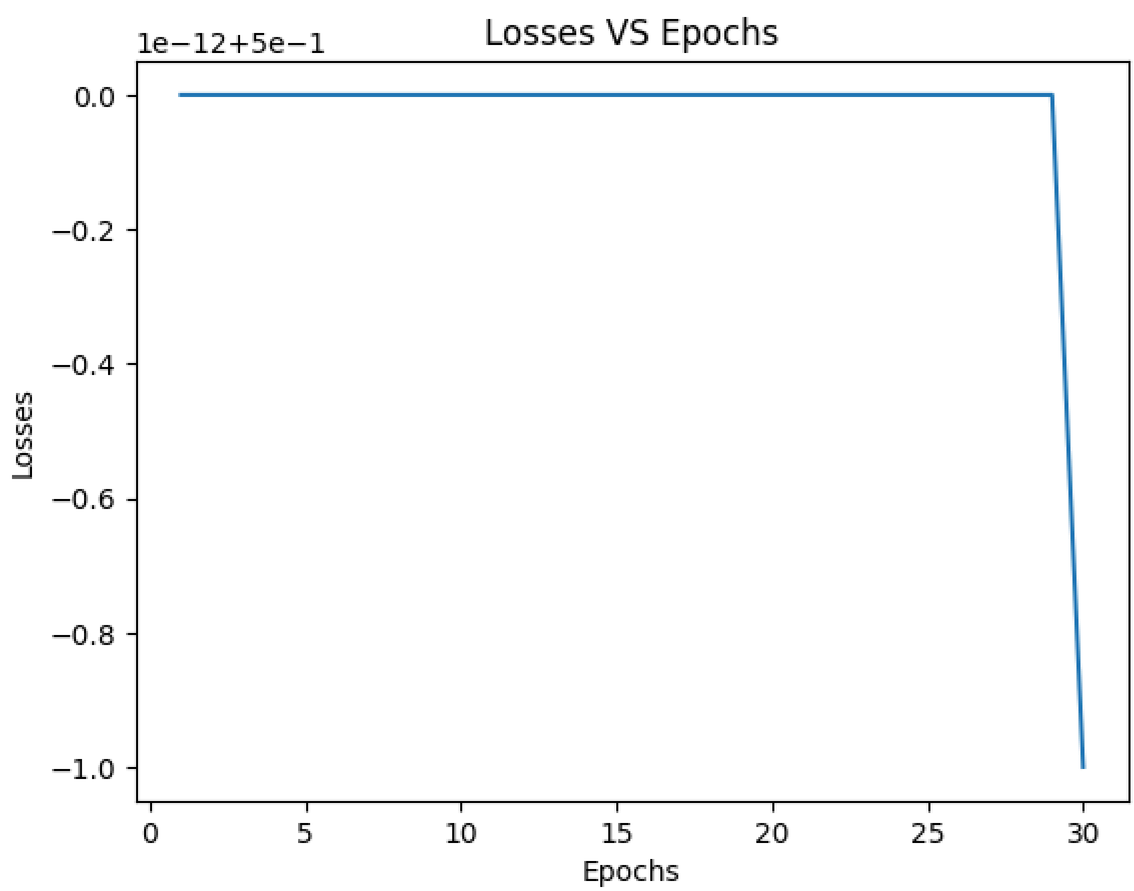
```
model=NeuralNetwork(X,y)
iterations = 30
intervals = 2
losses = []
ep = []
for i in range(iterations):
    if i % intervals == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(iterations)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
```

```
plt.ylabel('Losses')
plt.show()
```

```
For iteration # 30
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]
 [0. 0.]
 [1. 0.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]]
Predicted Output:
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
Loss:
0.49999999999900063
```



1. Implement a 3 layer multilayer perceptron neural network with 2-6-1 architecture and solvethe EX-OR classification problem usingbackpropagation algorithm. Note: Consider bias at everyneuron.Use Sigmoidactivation functionat hidden layer neurons andReLU activation function at output layer neuron.Train for 100 epochs. Plot the convergence graph.

In [121…

```
class NeuralNetwork:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(2,6)
        self.weights2 = np.random.rand(6,1)
        self.bias1 = np.random.rand(1,6)
        self.bias2 = np.random.rand(1,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = sigmoid(np.dot(self.input, self.weights1) + self.bias1)
        self.layer2 = relu(np.dot(self.layer1, self.weights2) + self.bias2)
        print(self.layer2)
        return self.layer2

    def backprop(self):
        '''Backpropagation of the final hidden layers to initial layers'''
        error = self.y - self.output

        d_weights2 = np.dot(self.layer1.T, 2 * error * sigmoid_derivative(self.output))
        d_bias2 = np.sum(2 * error * sigmoid_derivative(self.output), axis=0, keepdims=

        error_hidden_layer = np.dot(2 * error * relu_derivative(self.output), self.weig

        d_weights1 = np.dot(self.input.T, error_hidden_layer * relu_derivative(self.lay
        d_bias1 = np.sum(error_hidden_layer * relu_derivative(self.layer1), axis=0)

        self.weights1 += d_weights1
        self.weights2 += d_weights2
        self.bias1 += d_bias1
        self.bias2 += d_bias2

    def train(self, X, y):
        self.output = self.feedforward()
        self.backprop()
```

In [123…

```
model=NeuralNetwork(X,y)
iterations = 7
losses = []
ep = []
for i in range(iterations):
    if i % 5 == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
```

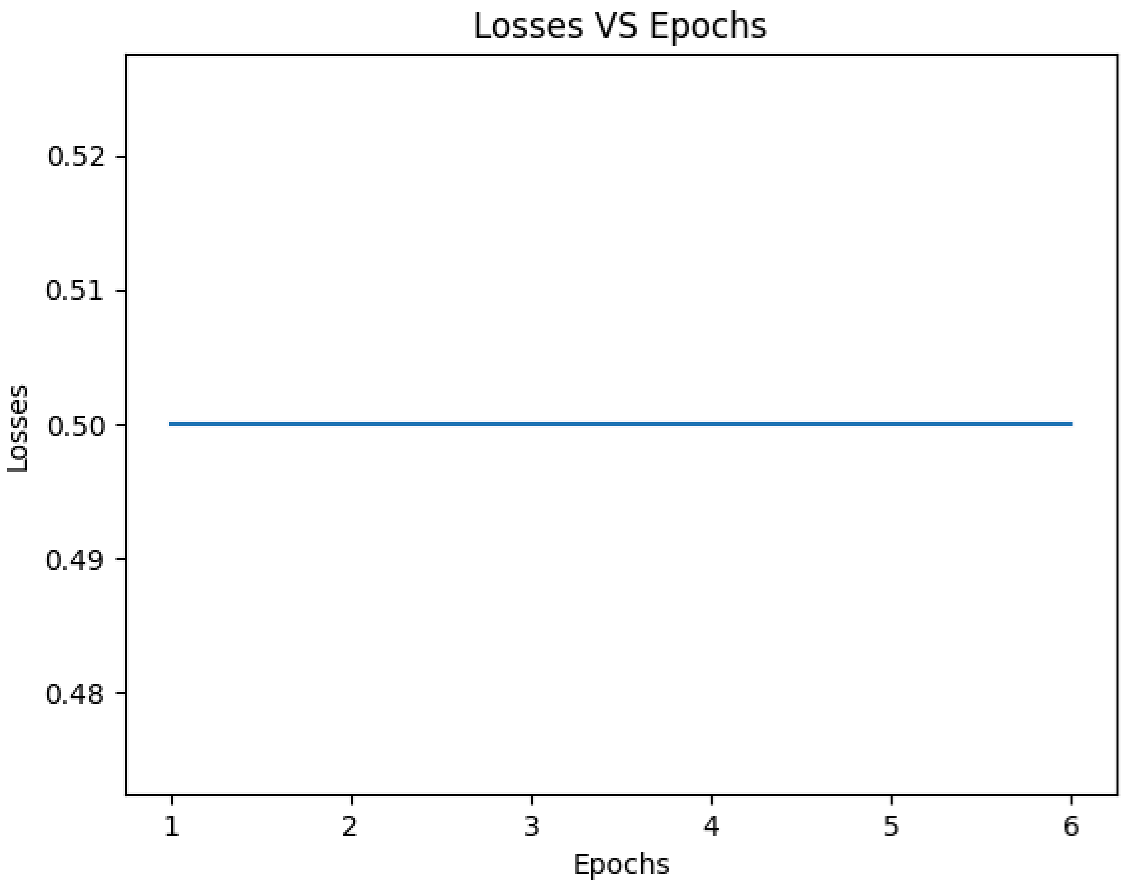
```
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(iterations)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.show()
```

```
[[2.56067223]
 [2.67570006]
 [2.86324808]
 [2.95924424]
 [2.56067223]
 [2.86324808]]
[[145.17613325]
 [144.16077078]
 [135.8383588 ]
 [135.58266172]
 [145.17613325]
 [135.8383588 ]]
[[32890861.85757602]
 [32890861.85757602]
 [32890861.85757602]
 [32890861.85757602]
 [32890861.85757602]
 [32890861.85757602]]
[[4.26979464e+23]
 [4.26979464e+23]
 [4.26979464e+23]
 [4.26979464e+23]
 [4.26979464e+23]
 [4.26979464e+23]]
[[9.34119005e+71]
 [9.34119005e+71]
 [9.34119005e+71]
 [9.34119005e+71]
 [9.34119005e+71]
 [9.34119005e+71]]
[[9.78110385e+216]
 [9.78110385e+216]
 [9.78110385e+216]
 [9.78110385e+216]
 [9.78110385e+216]
 [9.78110385e+216]]
[[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
For iteration # 7
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
```

```
 [1. 1.]
 [0. 0.]
 [1. 0.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]]
[[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
Predicted Output:
[[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
[[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
Loss:
nan
```

```
C:\Users\Laptop\AppData\Local\Temp\ipykernel_28232\2418103403.py:3: RuntimeWarning: over
flow encountered in exp
  return 1/(1+np.exp(-t))
C:\Users\Laptop\AppData\Local\Temp\ipykernel_28232\1397129429.py:2: RuntimeWarning: over
flow encountered in multiply
  return d * (1 - d)
```



1. Implement a 3 layer multilayer perceptron neural network with 2-6-1 architecture and solvethe EX-OR classification problem usingbackpropagation algorithm. Note: Consider bias at everyneuron.Use ReLUactivation functionat everyneuron.Train for 100 epochs. Plot the convergence graph.

In [124…

```
class NeuralNetwork:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(2,6)
        self.weights2 = np.random.rand(6,1)
        self.bias1 = np.random.rand(1,6)
        self.bias2 = np.random.rand(1,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        '''This will perform the forward propagation for the next 2 layers'''
        self.layer1 = relu(np.dot(self.input, self.weights1) + self.bias1)
        self.layer2 = relu(np.dot(self.layer1, self.weights2) + self.bias2)
        print(self.layer2)
        return self.layer2

    def backprop(self):
        '''Backpropagation of the final hidden layers to initial layers'''
        error = self.y - self.output

        d_weights2 = np.dot(self.layer1.T, 2 * error * relu_derivative(self.output))
        d_bias2 = np.sum(2 * error * relu_derivative(self.output), axis=0, keepdims=Tru
```

```
        error_hidden_layer = np.dot(2 * error * relu_derivative(self.output), self.weig

        d_weights1 = np.dot(self.input.T, error_hidden_layer * relu_derivative(self.lay
        d_bias1 = np.sum(error_hidden_layer * relu_derivative(self.layer1), axis=0)

        self.weights1 += d_weights1
        self.weights2 += d_weights2
        self.bias1 += d_bias1
        self.bias2 += d_bias2

    def train(self, X, y):
        self.output = self.feedforward()
        self.backprop()
```

In [126…

```
model=NeuralNetwork(X,y)
iterations = 40
losses = []
ep = []
for i in range(iterations):
    if i % 5 == 0:
        losses.append(np.mean(np.square(y - model.output)))
        ep.append(i+1)
    model.train(X, y)

print("For iteration #", iterations)
print ("Input : \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n" + str(model.feedforward()))
loss = np.mean(np.square(y - model.feedforward()))
print ("Loss: \n" + str(loss))
losses.append(loss)
ep.append(iterations)
print ("\n")

plt.plot(ep, losses)
plt.title('Losses VS Epochs')
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.show()
```

```
[[2.89428029]
 [5.18400226]
 [4.47486359]
 [6.76458556]
 [2.89428029]
 [4.47486359]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

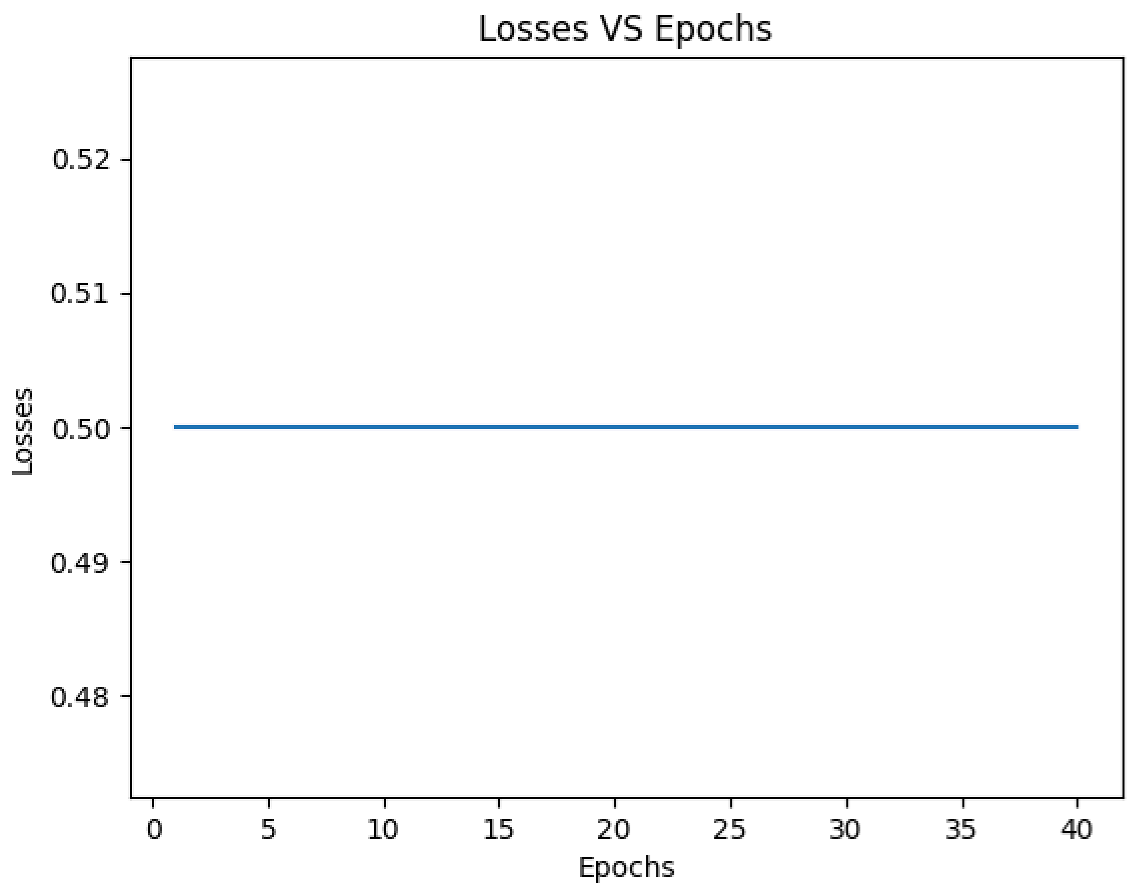
```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
For iteration # 40
Input :
[[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]
 [0. 0.]
 [1. 0.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]]
[[0.]
 [0.]
 [0.]
```

```
 [0.]
 [0.]
 [0.]]
Predicted Output:
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
Loss:
0.5
```



When having relu in layer 2 there is overflow of weight as relu does have an upper limit causing it to except the computer storage capacity. causing overflow in qno 5 and 6. But sigmoid limit is 0 - 1 so we dont face this issue in other questions