q1

September 20, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: #inputs
        inputs = np.array([[0,0], [0,1],[1,0],[1,1]])
        exp_output = np.array([[0],[1],[1],[0]])
```

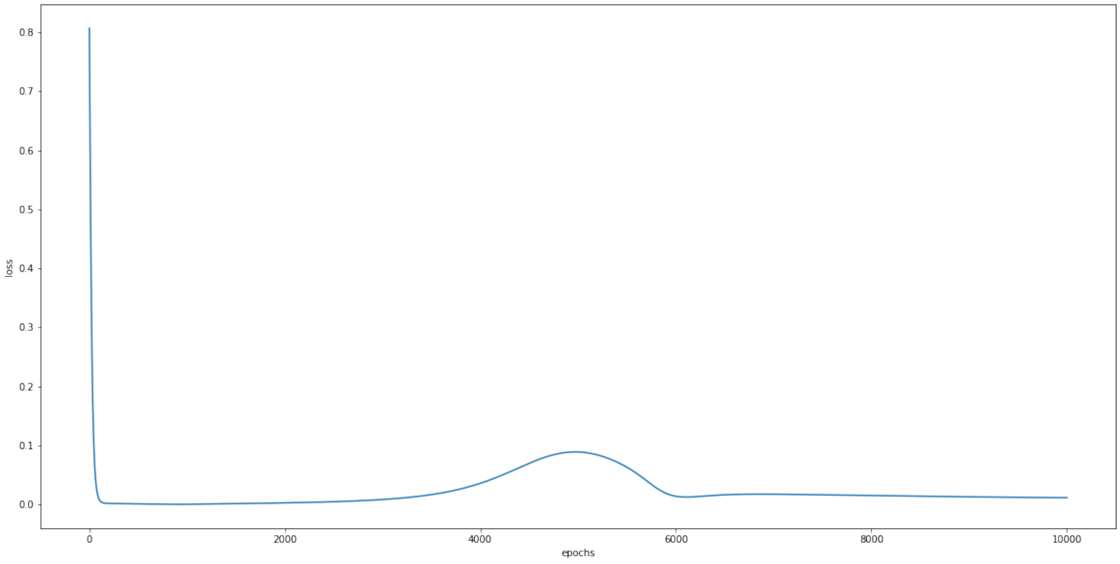
```
In [3]: def sigmoid (x):
            return 1/(1+np.exp(-x))
```

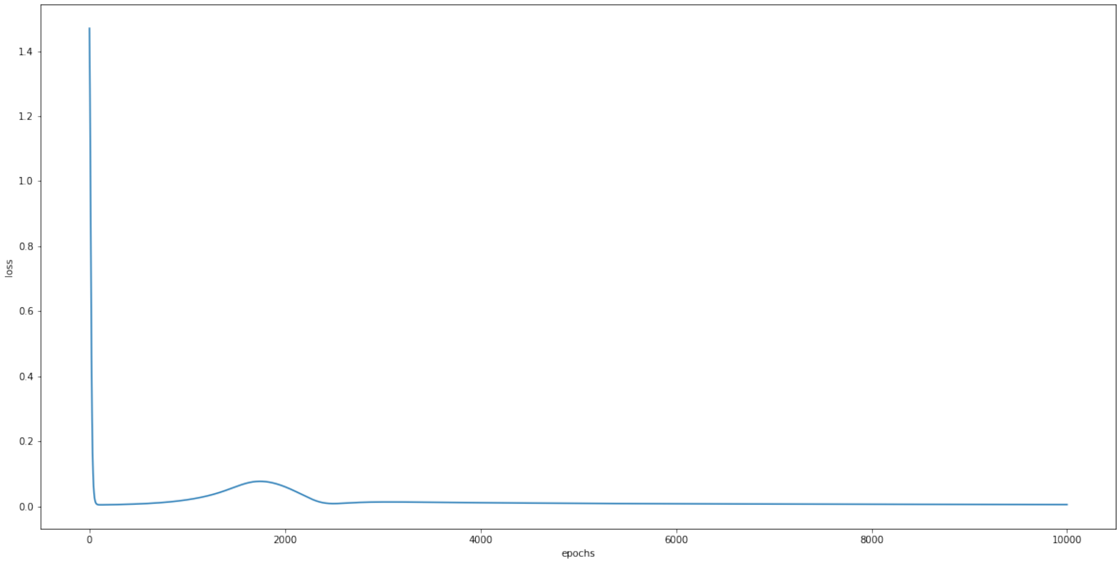
```
In [4]: def sigmoid_derivative (x):
            return x*(1-x)
```

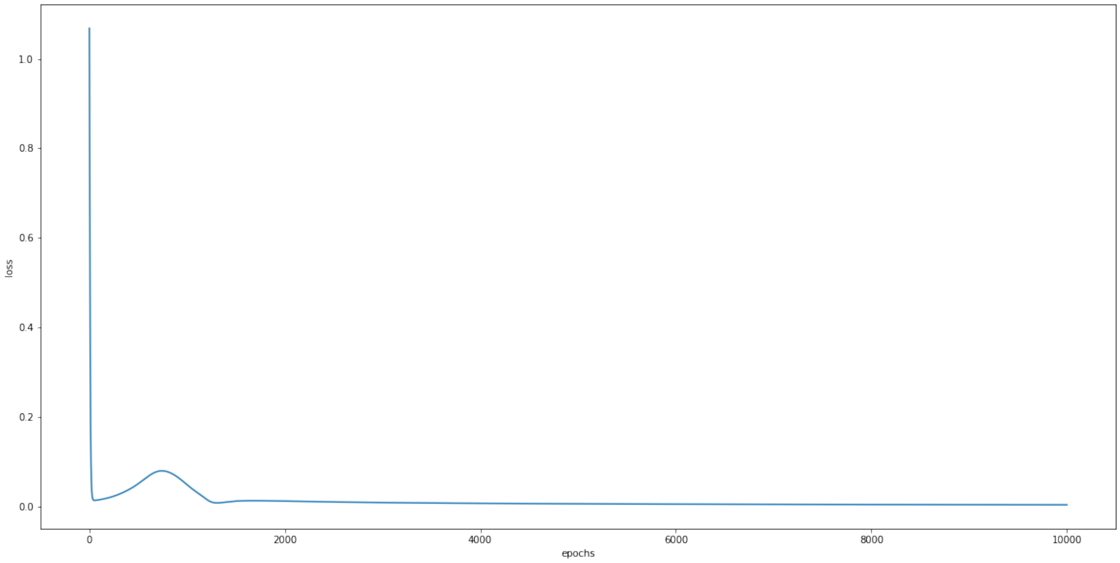
```
In [5]: def train(epochs = 10000, learning_rate = 0.1):
            input_layer_neurons = 2
            hidden_layer_neurons = 2
            output_layer_neurons = 1

            hidden_layer_weights = np.random.uniform(size=(input_layer_neurons, hidden_layer_ne
            hidden_bias = np.random.uniform(size=(1,hidden_layer_neurons))
            output_layer_weight = np.random.uniform(size=(hidden_layer_neurons,output_layer_neu
            output_bias = np.random.uniform(size=(1,output_layer_neurons))

            loss = []
            epochs_arr = []

            for epoch in range(epochs):
                hidden_layer_activation = np.dot(inputs, hidden_layer_weights)
                hidden_layer_activation += hidden_bias
                hidden_layer_output = sigmoid(hidden_layer_activation)

                output_layer_activation = np.dot(hidden_layer_output, output_layer_weight)
                output_layer_activation += output_bias

                predicted_output = sigmoid(output_layer_activation)

                #error
                error = exp_output - predicted_output
```

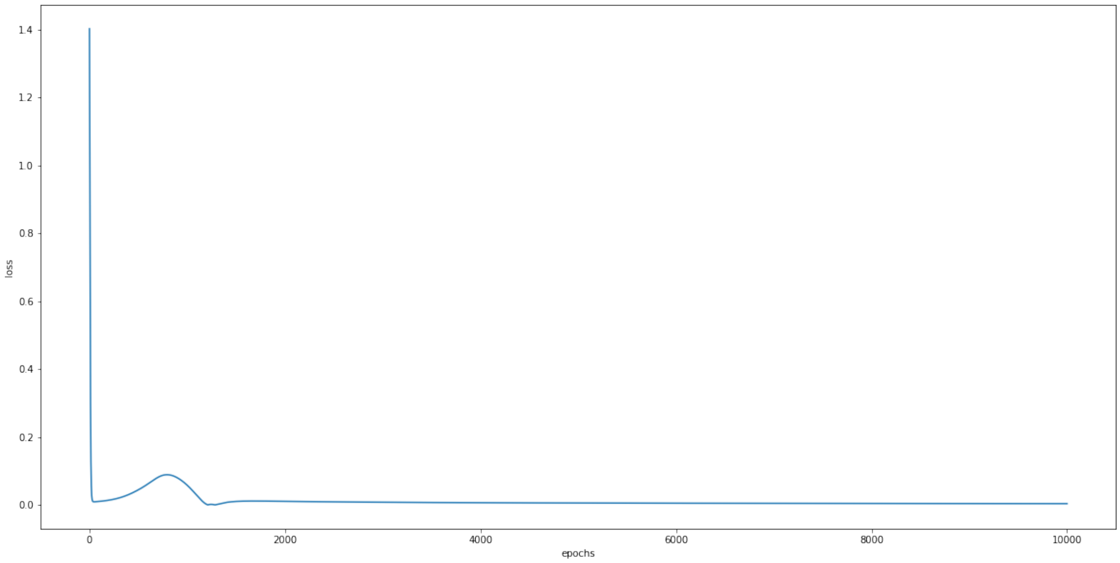
```
        derivative_predicted_output = error*sigmoid_derivative(predicted_output)

        error_hidden_layer = derivative_predicted_output.dot(output_layer_weight.T)
        derivative_hidden_layer_output = error_hidden_layer*sigmoid_derivative(hidden_

        output_layer_weight += hidden_layer_output.T.dot(derivative_predicted_output) *
        output_bias += np.sum(derivative_predicted_output, axis=0 , keepdims=True) * le
        hidden_layer_weights += inputs.T.dot(derivative_hidden_layer_output) *learning_
        hidden_bias += np.sum(derivative_hidden_layer_output, axis=0, keepdims=True) *l

        epochs_arr.append(epoch+1)
        loss.append(np.sum(error))

    return epochs_arr, loss
```

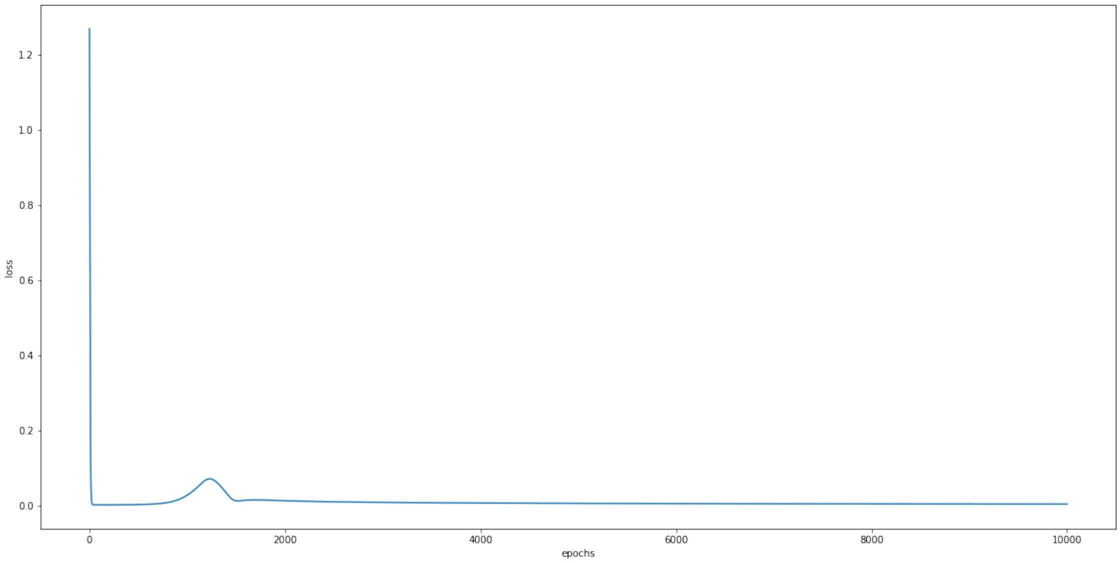
```
In [6]: loss_lr_data = []
        lr_val = []

        for lr in range(9):
            epochs_arr , loss = train(learning_rate=(lr+1)/10)
            loss = np.absolute(loss)

            lr_val.append((lr+1)/10)
            loss_lr_data.append(loss[-1])


            %matplotlib inline
            plt.rcParams['figure.figsize'] = [20,10]
            plt.xlabel('epochs')
            plt.ylabel('loss')
            plt.plot(epochs_arr, loss)
            plt.show()
```

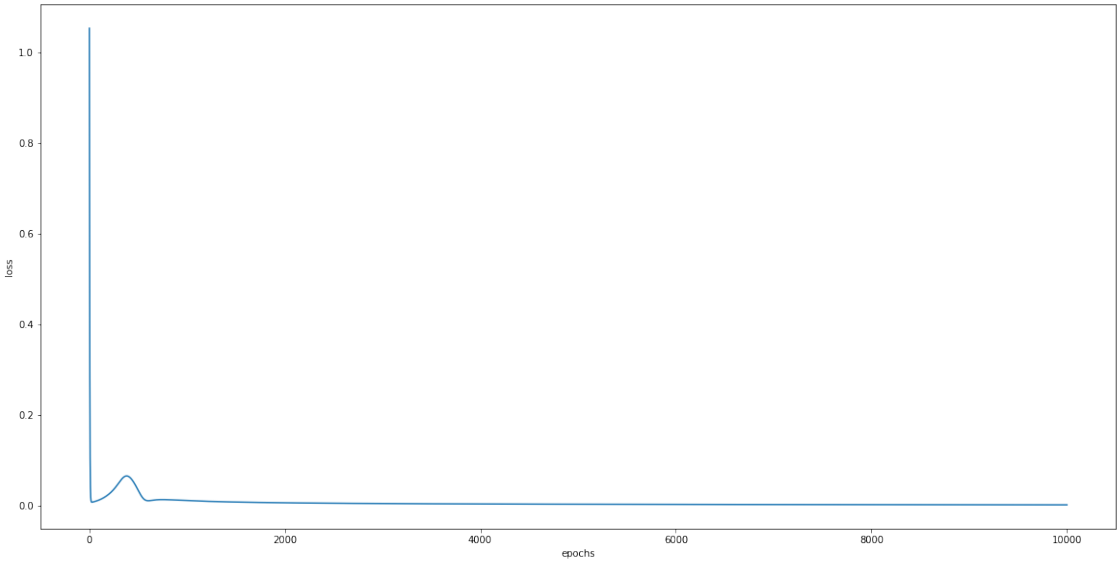


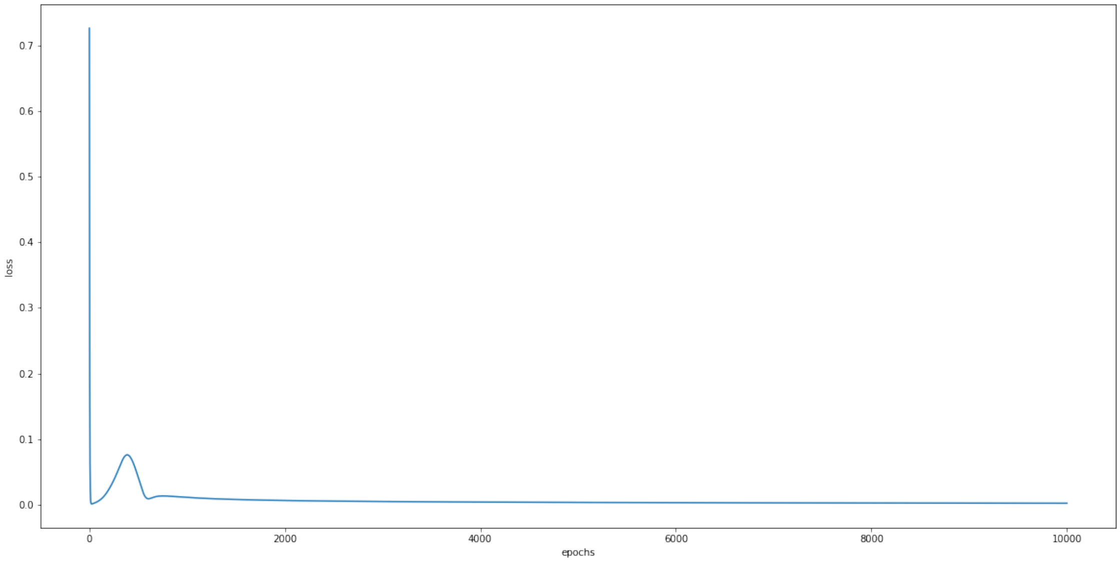

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
loss
0
2000
4000
6000
8000
10000
epochs

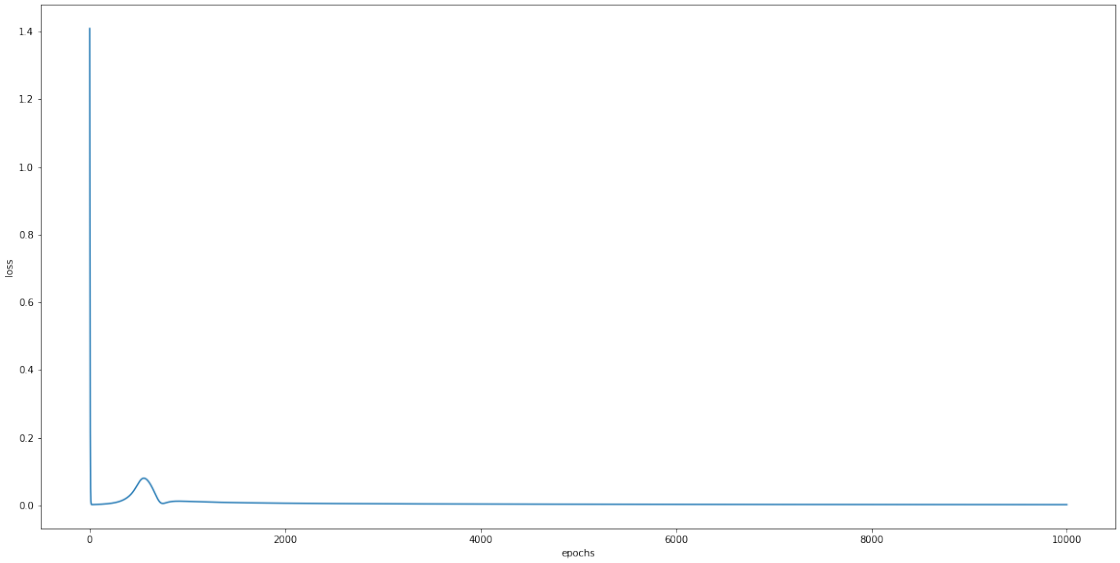

1.0
0.8
0.6
0.4
0.2
0.0
loss
0
2000
4000
6000
8000
10000
epochs

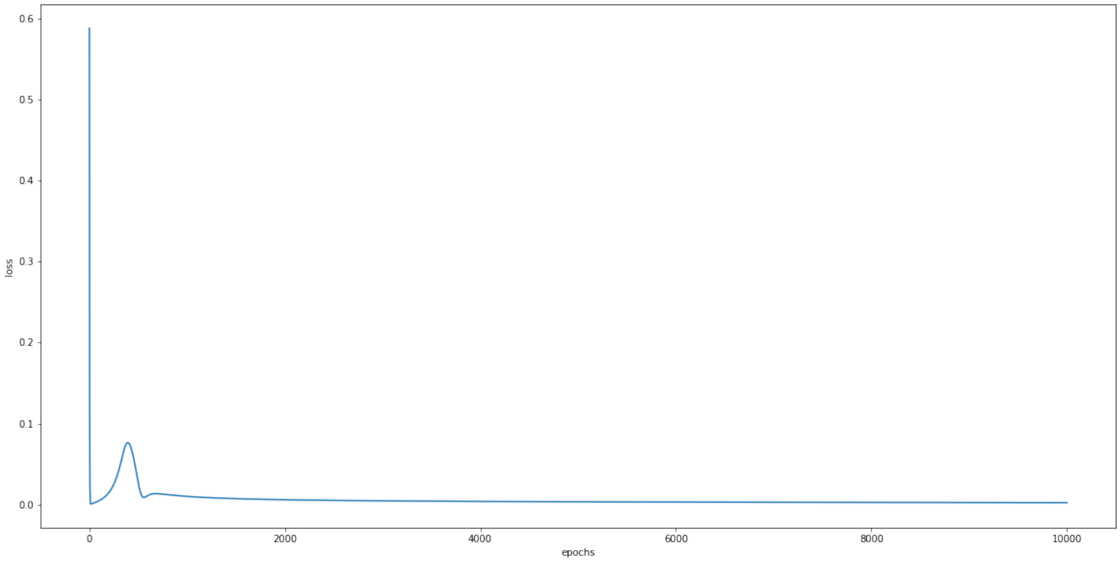

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
loss
0
2000
4000
6000
8000
10000
epochs


1.2
1.0
0.8
0.6
0.4
0.2
0.0
loss
0
2000
4000
6000
8000
10000
epochs

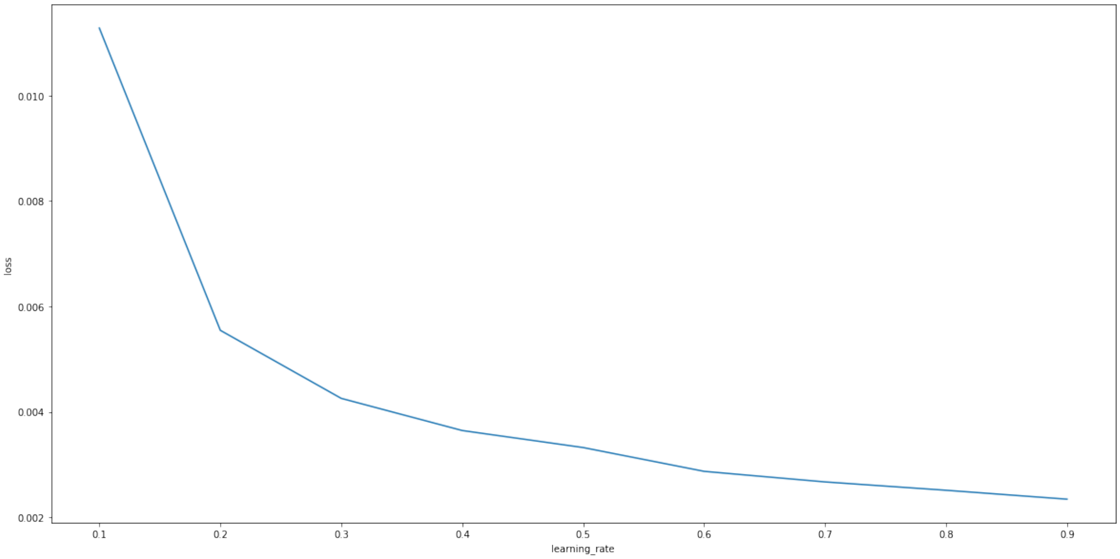

1.0
0.8
0.6
0.4
0.2
0.0
loss
0
2000
4000
6000
8000
10000
epochs


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
loss
0
2000
4000
6000
8000
10000
epochs





```
In [7]: %matplotlib inline
        plt.rcParams['figure.figsize'] = [20,10]
        plt.xlabel('learning_rate')
        plt.ylabel('loss')
        plt.plot(lr_val, loss_lr_data)
        plt.show()
```


0.010
0.008
0.006
0.004
0.002
loss
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
learning_rate