Training the neural network of a hidden layer based on the Backpropagation Algorithm

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import math
import random
```

Importing and using needed libraries to solve the question:

pandas:

To call and use the “Dataframe” function.

numpy:

To use its features and functions dealing with matrixes.

matplotlib:

To show and plot the input data’s graph and to draw the model.

sklearn:

To call the “two moons” function and plot it as the dataset.

To separate the training and testing datasets with one percent specific segregation.

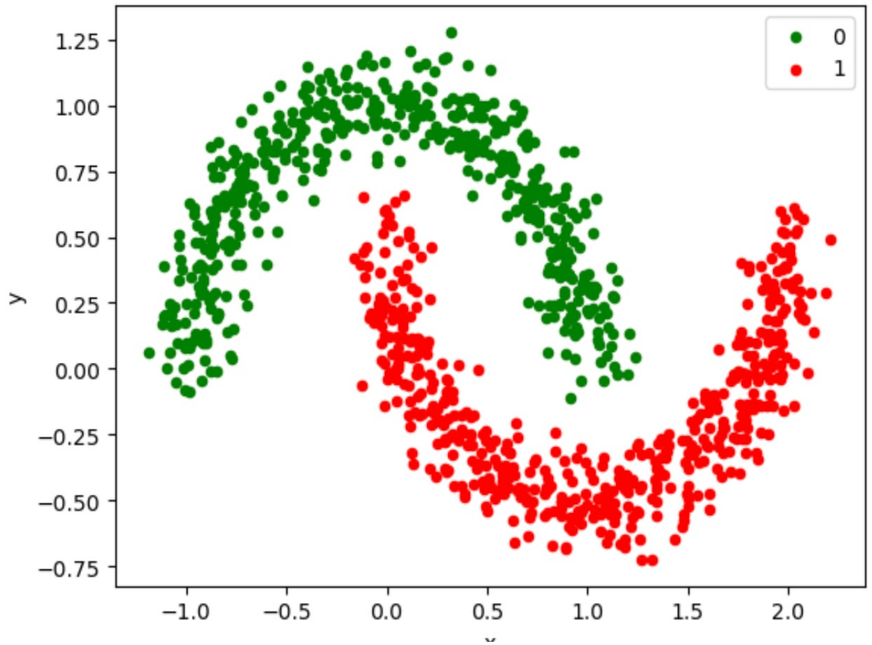
math:

To use the infinite number.

random:

To use an extreme value for weighting the hidden neuron layers.

```
x,y = make_moons(n_samples=1000,noise=0.1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 80)

df=pd.DataFrame(dict(x=x[:,0],y=x[:,1], label =y))
colors ={0:'green' , 1:'red'}
fig, ax = plt.subplots()
grouped = df.groupby('label')
for key , group in grouped:
    group.plot(ax=ax , kind = 'scatter' , x= 'x' ,y='y' , label=key ,color=colors[key])
plt.show()
```



To call and use the “sklearn” from the “make moon” library.

After importing the corresponding function, there can be a dataset with one thousand samples and a noise of 0/1.

After the formation of the corresponding dataset by using the “sklearn”, this dataset is split into training and testing data in ratios of respectively 0/3 and 0/7.

In the end, using a graph, plot, and scatter, each internal point of the dataset is shown as a spot in the figure.

```python
def square_error_loss(y_true, y_pred):
    return (y_true - y_pred) ** 2

def mean_square_error(y_true, y_pred):
    return np.sum(square_error_loss(y_true, y_pred)) / len(y_pred)

def sig_af(z):
    return 1/(1 + np.exp(-z))

def tanh_af(z):
    return np.tanh(z)
```

```python
def set_params(n_input,n_hidden,n_out,hidden_af,output_af):

    W_hide = np.random.randn(n_hidden, n_input+1)
    W_out = np.random.randn(n_out, n_hidden+1)
    delta_W_hide_ago = np.zeros((n_hidden, n_input+1))
    delta_W_out_ago = np.zeros((n_out, n_hidden+1))
    dw_out = np.zeros((n_out,n_hidden+1))
    dw_hide = np.zeros((n_hidden,n_input+1))

    parameters = {"W_hide": W_hide,
                  "W_out":W_out,
                  "delta_W_hide_ago":delta_W_hide_ago,
                  "delta_W_out_ago":delta_W_out_ago,
                  "act_func_hide":hidden_af,
                  "act_func_out":output_af}

    return parameters
```

Using these two functions, the MSE error of testing the model is estimated.

These two functions are the activation functions of a person's neural network.

This function receives the number of Neurons in the first and the hidden layers and forms these layers' weight matrixes randomly so that the number of rows equals the number of neurons in each layer. It also forms the weight matrixes of the previous step which are used in the momentum method. Then, this function returns all of these data as a variable of the function it has called in a dictionary.

```python
def forward_propagation(X, parameters):

    X_b = np.append(np.ones((X.shape[0],1)),X,axis=1)
    z_hide = np.dot(parameters["W_hide"],X_b.T)

    if parameters['act_func_hide'] =='tanh':
        A_hide = tanh_af(z_hide)
    else :
        A_hide = sig_af(z_hide)

    A_hide_b = np.append(np.ones((1,X.shape[0])),A_hide,axis=0)
    z_out = np.dot(parameters["W_out"],A_hide_b)

    if parameters['act_func_out']=='liner':
        A_out = z_out
    elif parameters['act_func_out']=='tanh':
        A_out = tanh_af(z_out)
    else:
        A_out = sig_af(z_out)

    prev_params = {"z_hide": z_hide,
                   "A_hide": A_hide,
                   "z_out": z_out,
                   "A_out": A_out}

    return A_out,prev_params
```

The fist row of each of the inputs are being added by the value 1. This is done to recognize the bias.

The activity function is recognized with one condition.

The first row of each of the inputs of the last layer are being added by value 1. This is done to recognize the bias.

Using this function, the output value of the neural network can be estimated.

The neural network inputs along with the network parameters are given to the function. This function first enters the outputs of the first layer which are determined by an internal multiplication into the inputs of the hidden layer, and then after calculating the outputs of the hidden layer using the corresponding activity function and calling it to calculate their output, it returns them in the output.

It should be noted that the amount of previous iteration are being returned to estimate the momentum in the output along with the output of the network.

```
def backward_propagation(parameters,prev_params, X, Y):

    m = X.shape[0]
    X = np.append(np.ones((X.shape[0],1)),X,axis=1)
    dL = -(Y-prev_params['A_out'])

    if parameters['act_func_hide'] =='tanh':
        dz_out = 1-np.power(prev_params['A_out'],2)
    else:
        dz_out = prev_params['A_out']*(1.0-prev_params['A_out'])
    dw_out = (1/m) * (np.dot(dL*dz_out,np.append(np.ones((1,X.shape[0])),prev_params['A_hide'],axis=0).T))


    if parameters['act_func_out']=='liner':
        dz_hide = prev_params['A_hide']
    elif parameters['act_func_out']=='tanh':
        dz_hide = 1-np.power(prev_params['A_hide'],2)
    else:
        dz_hide = prev_params['A_hide']*(1.0-prev_params['A_hide'])


    buffer = dL*np.dot(parameters['W_out'][:,1:].T,dz_out)
    buffer = buffer * dz_hide
    dw_hide = (1/m) * np.dot(buffer,X)
    grads = {"dw_out": dw_out,
             "dw_hide": dw_hide,}

    return grads
```

At first, the number of samples is saved in the variable “m” based on the number of their rows. Then, the derivative of the error function is estimated and saved in the variable “dL”.

Then, “dL” and “dz_out” are inner multiplied and the bias value is added to the result for the next calculations.

Then, after transposing the result, the two matrixes are multiplied with matrix multiplication. Finally, by dividing this matrix by m, all layers are divided by the number of samples, which is the derivative of the output layer.

To use the chain rule and find the derivative of the hidden layer, first, the derivative of the activation function is calculated and saved in dz_hide, then W_out (which is transposed without the weight of the bias neuron) is multiplied by dz_out and the result is multiplied by dL, and in the end, the result is saved in the buffer.

Now, the buffer is multiplied by dz_hide and the result is saved again in the buffer.

And at last, the buffer is multiplied by the inputs, is divined to m (the number of samples), and is saved in dw_hide.

A dictionary named “grads” is made that contains the derivative of each layer’s weight (which has been estimated). This dictionary is returned in the function’s output to update the next step.

This function is used after estimating the error to update the weight matrixes and edify them to decrease the error.

This function takes the derivative according to the operation function of the same network:

This is the derivative of the Sigmoid operation function:

(y_pred)(y*-y_pred)

This is the derivative of the tanh operation function:

$1 - \tanh^2$

This function is used to learn the two approaches of learning: Batch and online learning. The difference between them is that in the Batch learning approach, the value of the derivative should be divided by the number of samples.

This function is used to update the parameters of the neural network.

In this function, layers' weights are updated using the derivative of the weight matrixes of the previous steps. Now, the weights of the previous step are subtracted from the derivative of the weights and the results are multiplied by the learning rate (if it is constant). If the learning rate is not constant, the declining factor reaches the power of this epoch and is multiplied by the derivative of weight matrixes. In the end, the updated weights are saved in the matrix of parameters.

```
def update_parameters(parameters, grads):
    dw_hide = grads["dw_hide"]
    dw_out = grads["dw_out"]
    W_hide = parameters["W_hide"]
    W_out = parameters["W_out"]

    delat_W_hide = np.power(parameters['reduction_factor'],
    parameters['epoch']) * parameters['learning_rate'] * dw_hide + parameters['gama'] * parameters["delta_W_hide_ago"]


    delat_W_out = np.power(parameters['reduction_factor'],
    parameters['epoch']) * parameters['learning_rate'] * dw_out + parameters['gama'] * parameters["delta_W_out_ago"]


    W_hide = W_hide - delat_W_hide
    W_out = W_out - delat_W_out
    parameters["delta_W_hide_ago"] = delat_W_hide
    parameters["delta_W_out_ago"] = delat_W_out
    parameters['W_hide'] =  W_hide
    parameters['W_out'] =  W_out
    return parameters
```

```
def get_predict(parameters, X):

    A_out , prev_params = forward_propagation(X,parameters)
    for elm in A_out:
        elm = (elm > 0.5)

    return (elm *1)

def get_accurecy(parameters,X,y):

    true_detect = 0

    for index,x in enumerate(X):
        if get_predict(parameters,X[[index]]) == y[index]:
            true_detect += 1

    return true_detect/X.shape[0]
```

This function returns the outputs of neural network as True if they are more than 0/5 and as False if their below this number.

Using this function and comprising the labels and network's output, the accuracy of the neural network in the desired state is estimated and showed in the output.

```
def create_nn_model(X, y , nerun_hide,nerun_out, epochs = 20,act_hide='sig',
            act_out='sig',mode='online', print_cost=False ,
            learning_rate = 0.3,threshold=0.999,reduction_factor=1,gama=0):

    nerun_input = X.shape[1]
    parameters = set_params(nerun_input,nerun_hide,nerun_out,act_hide,act_out)
    parameters['learning_rate'] = learning_rate
    parameters['threshold'] = threshold
    parameters['reduction_factor'] = reduction_factor
    parameters['epochs'] = epochs
    parameters['gama'] = gama
    parameters_max_mse = parameters
    cost=math.inf
    mse_i=[]


#       Online Learning mode

    if mode == 'online':
        for i in range(parameters['epochs']):
            parameters['epoch'] = i
            for j in range(X.shape[0]):
                index = random.sample(range(X.shape[0]), 1)
                A_hide, prev_params = forward_propagation(X[index],parameters)
                grads = backward_propagation(parameters, prev_params, X[index], y[index])
                parameters = update_parameters(parameters, grads)

            cost = mean_square_error(A_hide, y,)
            mse_i.append(cost)
            if mean_square_error(A_hide, y,)< cost:
                parameters_max_mse = parameters
            a = get_accurecy(parameters,X,y)
            if a >= parameters['threshold']:
                break;
            if print_cost:
                print ("Cost after iteration %i: %f" % (i, cost))


#       Batch Learning mode
    else:
        for i in range(parameters['epochs']):
            parameters['epoch'] = i
            A_hide, prev_params = forward_propagation(X,parameters)
            grads = backward_propagation(parameters, prev_params, X, y)
            parameters = update_parameters(parameters , grads )
            cost = mean_square_error(A_hide, y,)
            mse_i.append(cost)
            if mean_square_error(A_hide, y,)< cost:
                parameters_max_mse=parameters
            a = get_accurecy(parameters,X,y)
            if a >= parameters['threshold']:
                    break;
            if print_cost:
                print ("Cost after iteration %i: %f" % (i, cost))


    plt.figure()
    plt.plot(np.arange(0, parameters['epoch']+1,1, dtype = int), mse_i)
    plt.xlabel('Itteration')
    plt.ylabel('Squared Error')
    plt.show()
    return parameters_max_mse
```

This function is used to form the neural network by receiving the training data and their labels, the number of neurons of hidden layer, the number of output neurons, and number of training replicates (if there is no replication, this number is considered 20). in addition, the function of both layers and the methods of learning such as online or Batch-based learning and whether or not to print the costs with the threshold and the decreasing factor will form the network.

the learning difference between online and Batch-based methods is that in online method, there are two loops. First loop checks the epochs and the second one studies the network input. Using every input, the network is trained. However, in the Batch-based method, the number of epochs and repetitions are estimated with only one loop. In each loop, all of the data are entered to the network as a package and are trained.

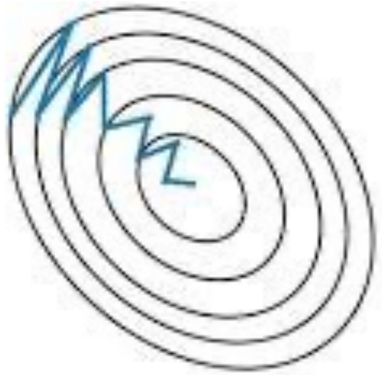
Also, with a condition that exists in both training methods, one checks the accuracy threshold limit and stops the learning when the accuracy threshold limit is reached.

```
def draw_decision_boundary(parameters, X, Y):

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    h = 0.01

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    grid_points = np.c_[xx.ravel(), yy.ravel()]

    Z = get_predict(parameters, grid_points)
    Z = Z.reshape(xx.shape)


    cs = plt.contourf(xx,yy,Z,colors=['#808080', '#A0A0A0', '#C0C0C0'], extend='both')
    cs.cmap.set_over('red')
    cs.cmap.set_under('green')
    plt.scatter(X[:, 0], X[:, 1], c=Y, cmap="gray")
    plt.ylabel('x2')
    plt.xlabel('x1')
    cs.changed()
```

Using this function and the maximum and minimum, the input samples are entered to the coordinate in the range of 0/01 and then the driven boundary is plotted.

Using the “contourf” function, the driven boundary is plotted in a way that the top of the line is colored in green and the bottom of the decision line is in red, just like our labels.

Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

The momentum method is used to increase the speed of convergence in the training process.

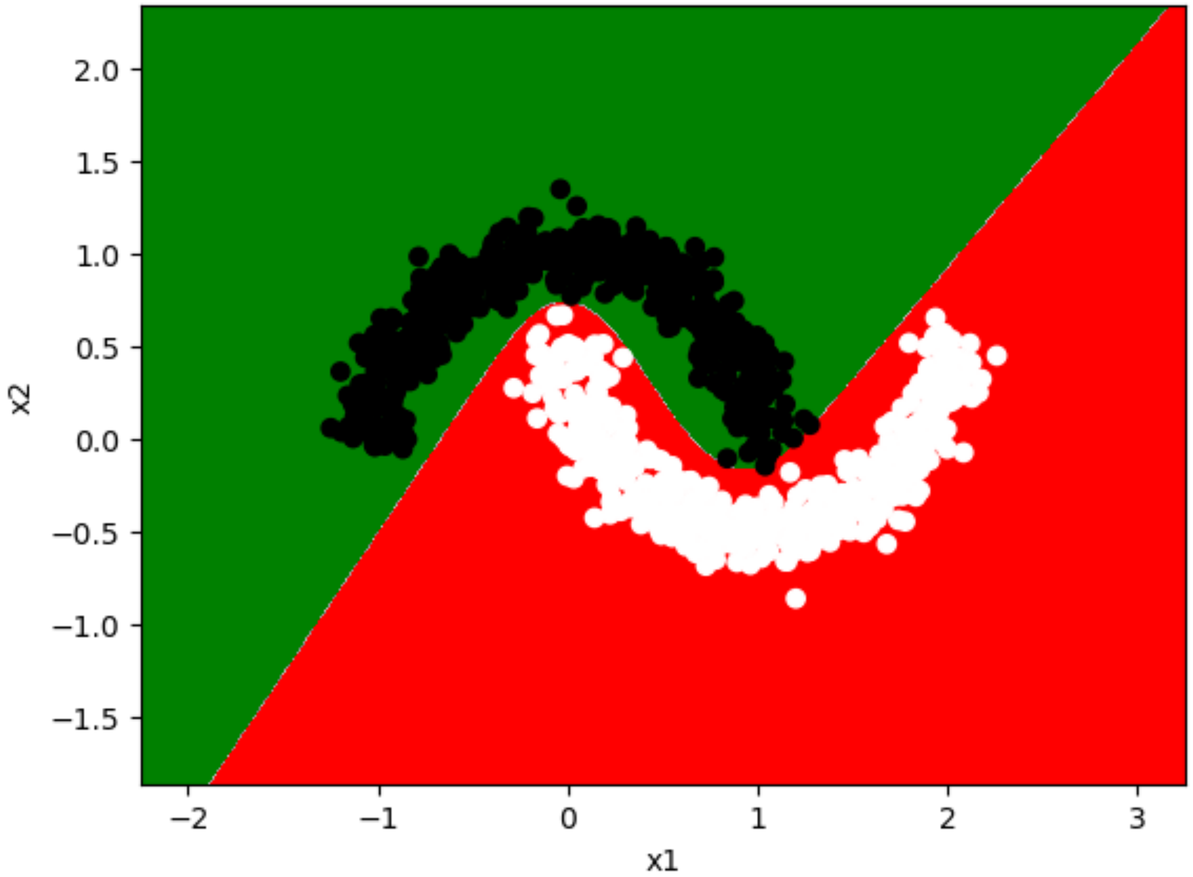
This method works like that in the training process, it affects and uses the number of changes of the weight matrix in the previous stage to train the weight matrix in the current stage in such a way that it multiplies the changes of the weight matrix in the previous stage by gamma in each stage and affects them to the changes of the current stage.

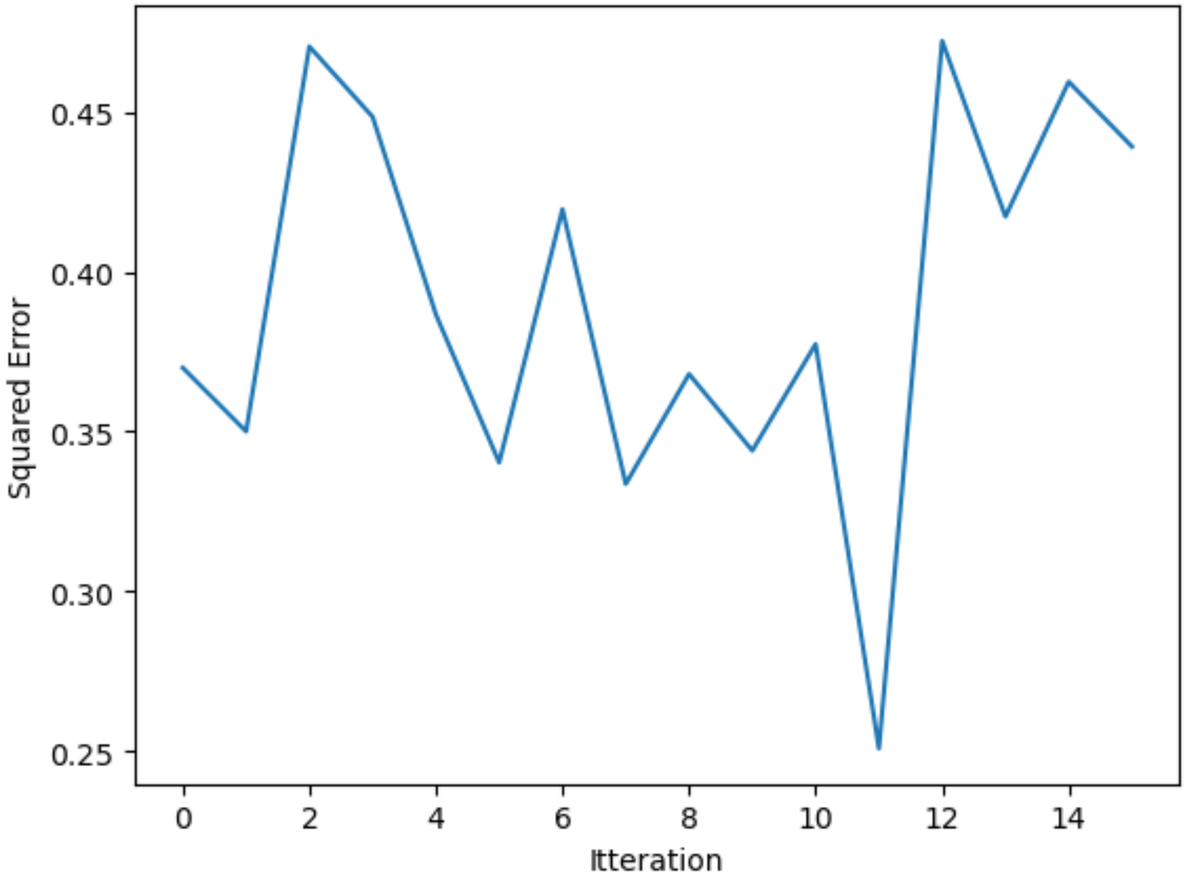
As is shown in the results of this report, using gamma input which starts the momentum action makes the minimum error possible with less repetition or epoch.

Plotting error function and decision boundary

Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a constant factor and online condition with the Sigmoid function in both layers

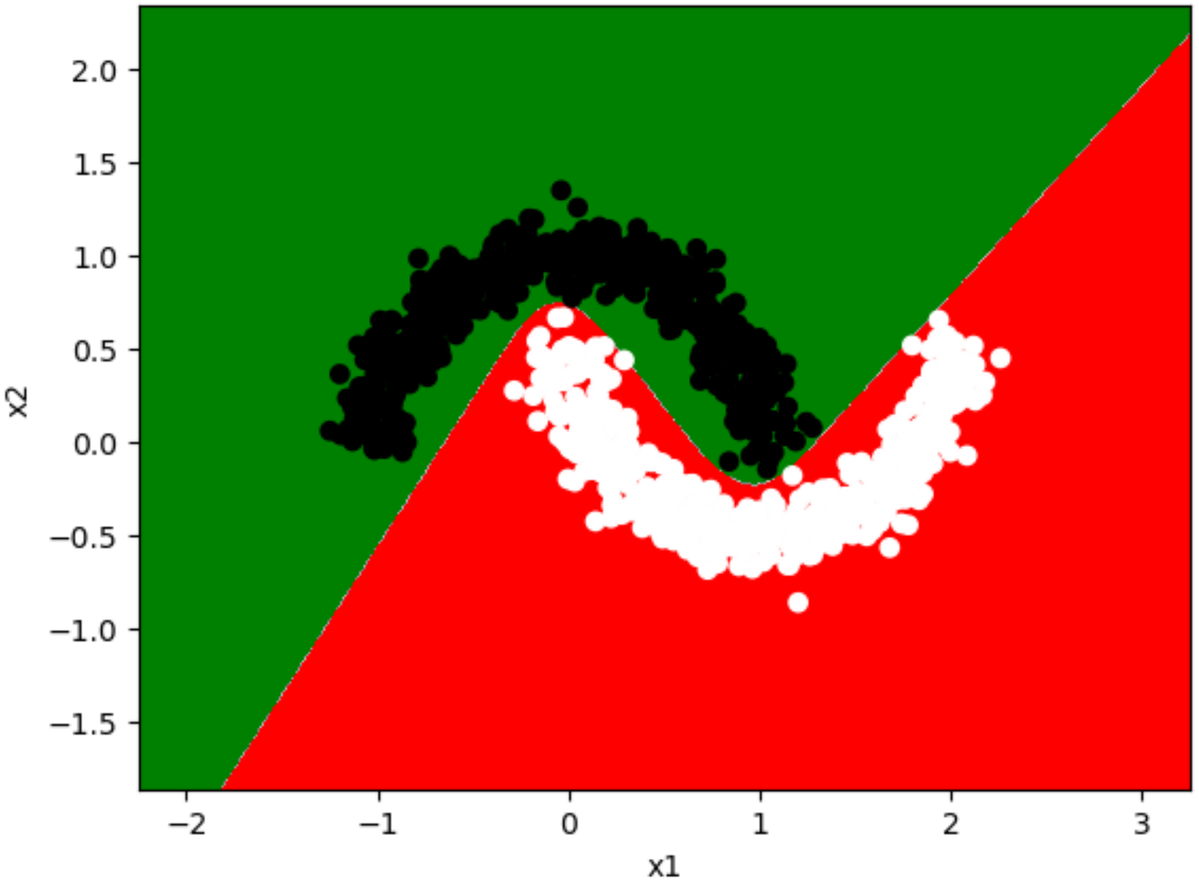
Accuracy = 1/0

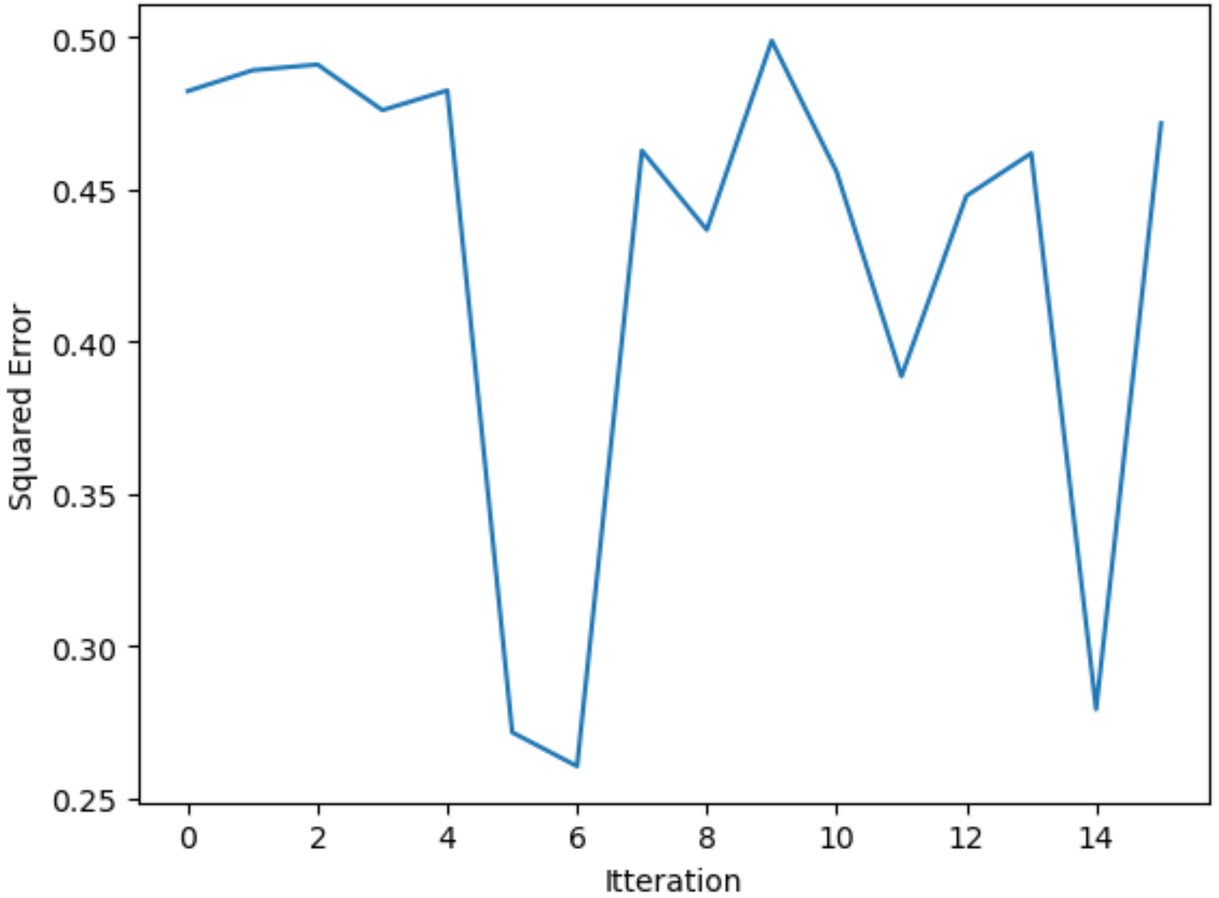




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a declining factor and online condition with the Sigmoid function in both layers

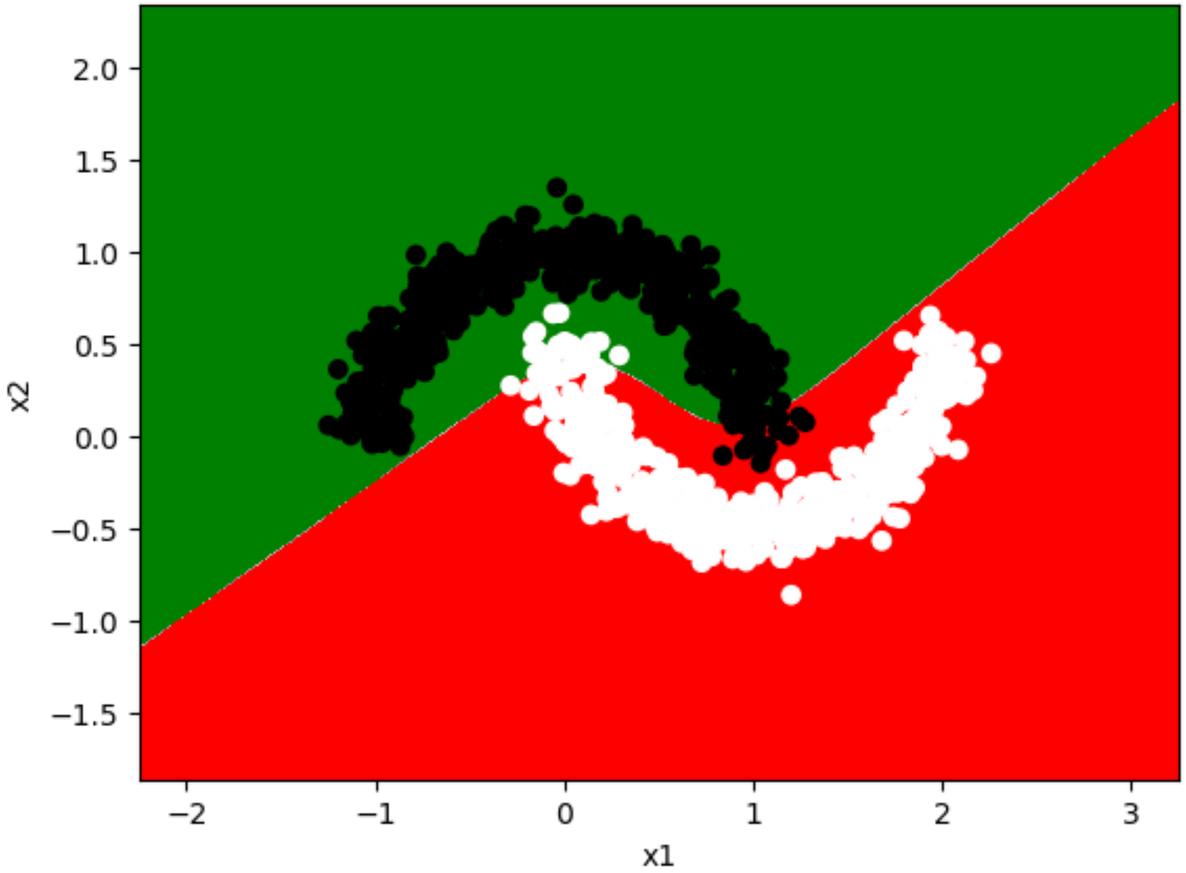
Accuracy = 1/0

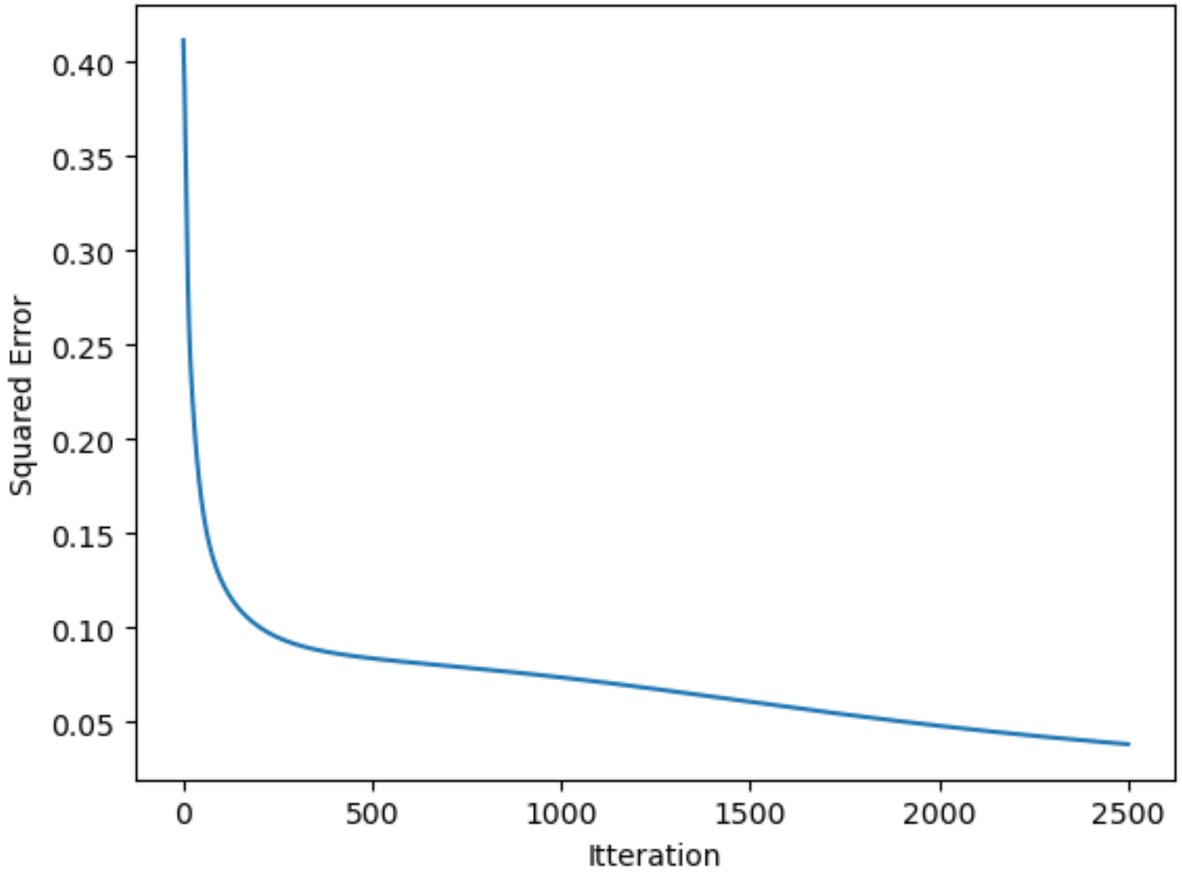




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a constant factor and Batch- condition with the Sigmoid function in both layers

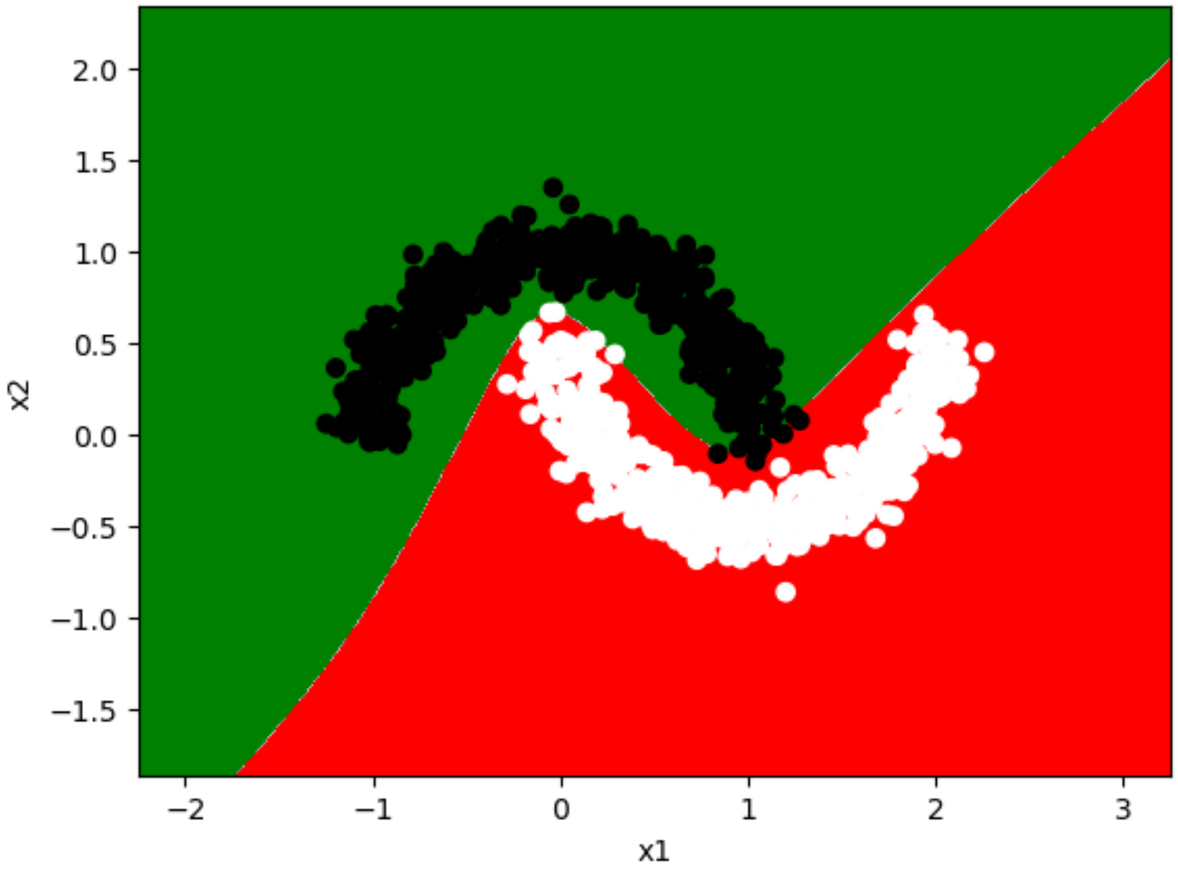
Accuracy = 0/96

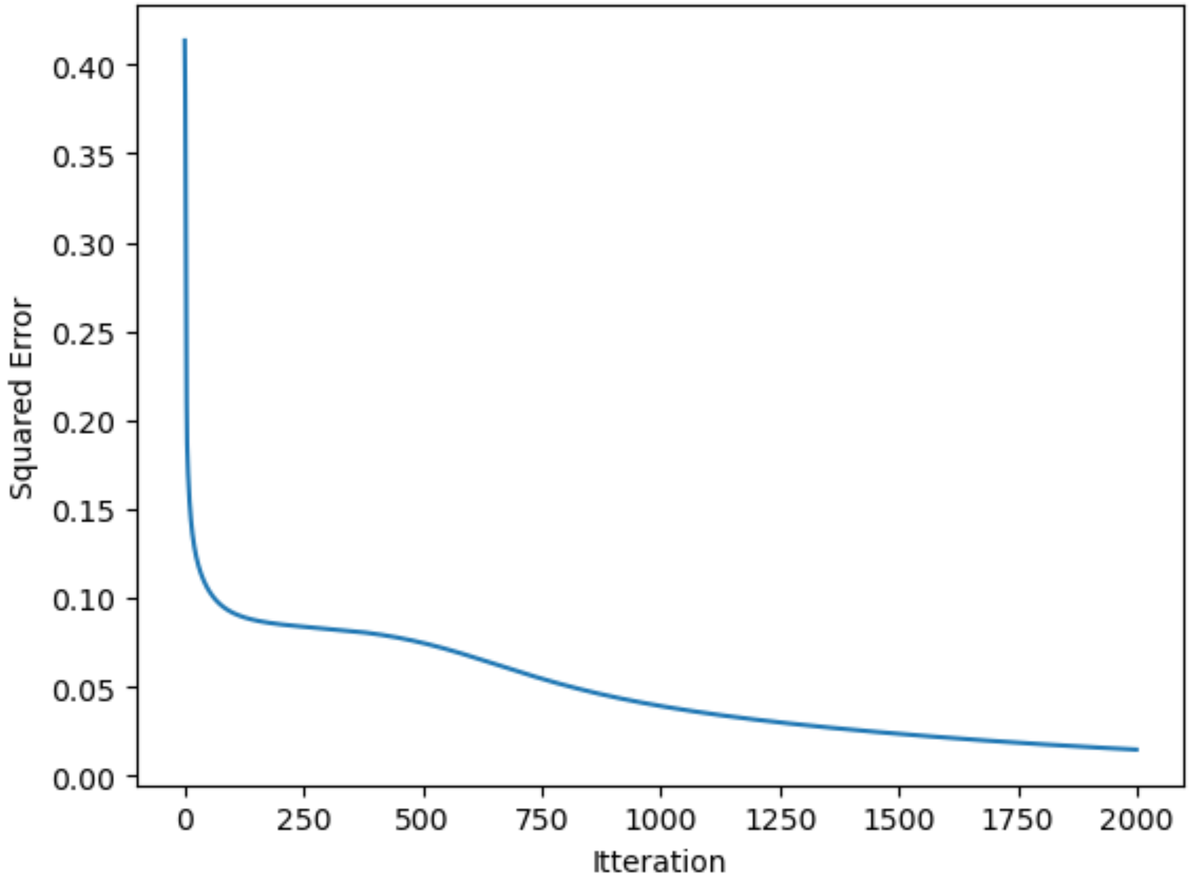




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a declining factor and Batch condition with the Sigmoid function in both layers

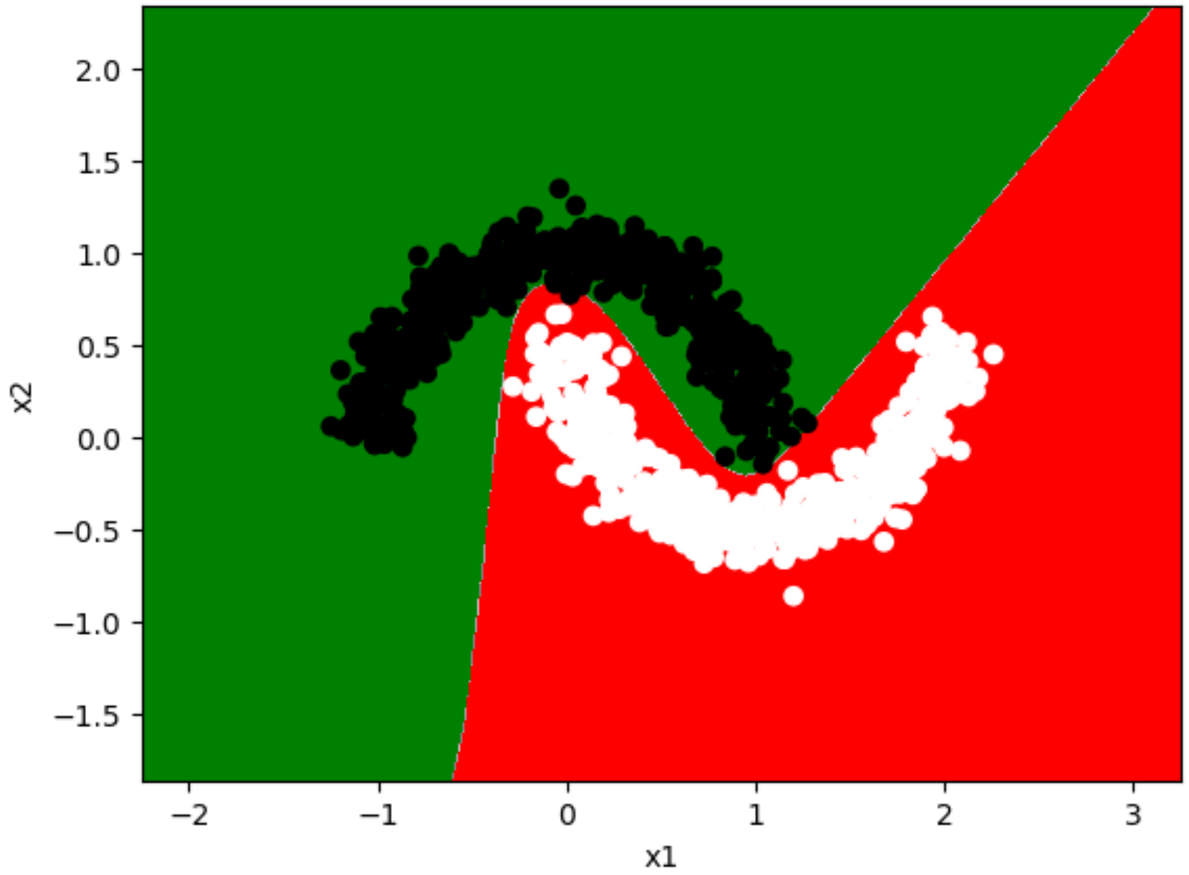
Accuracy = 0/99

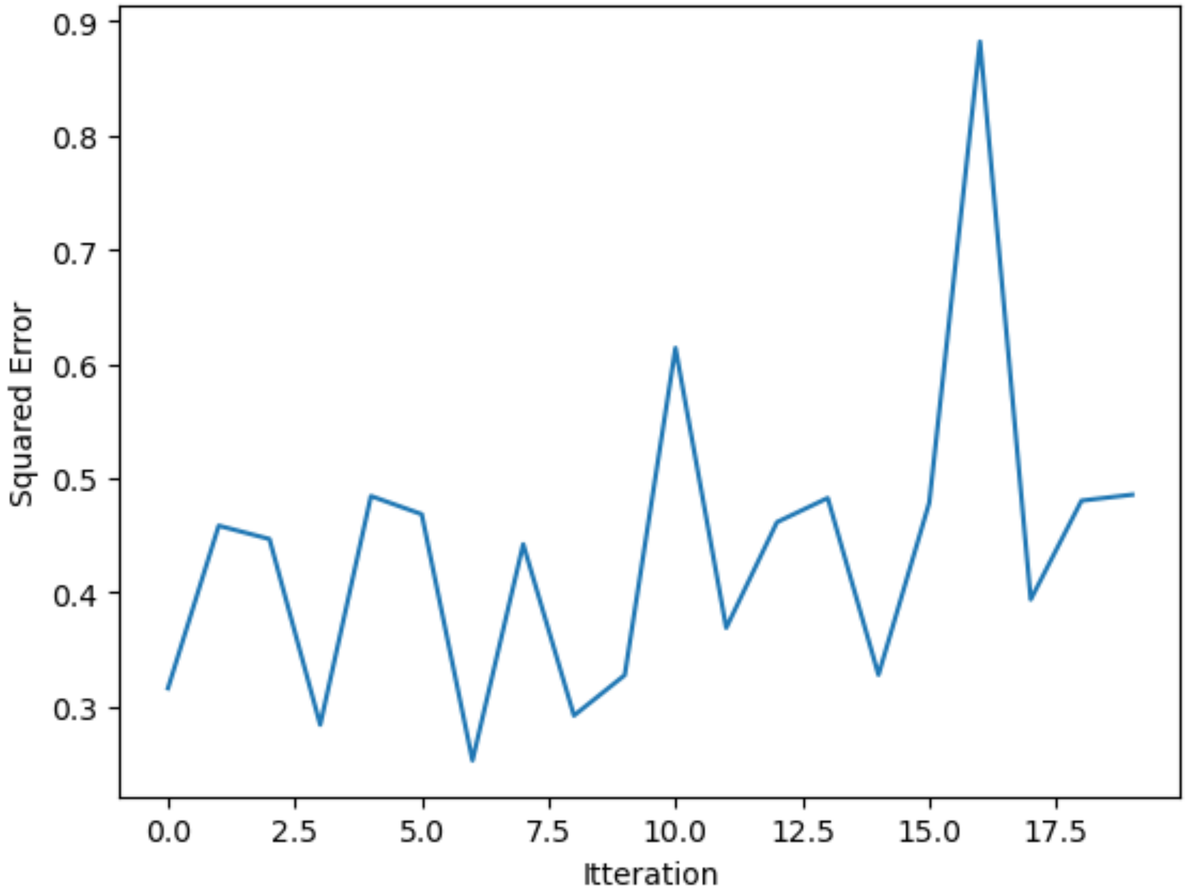




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a constant factor and online condition with the tangent-hyperbolic function in both layers

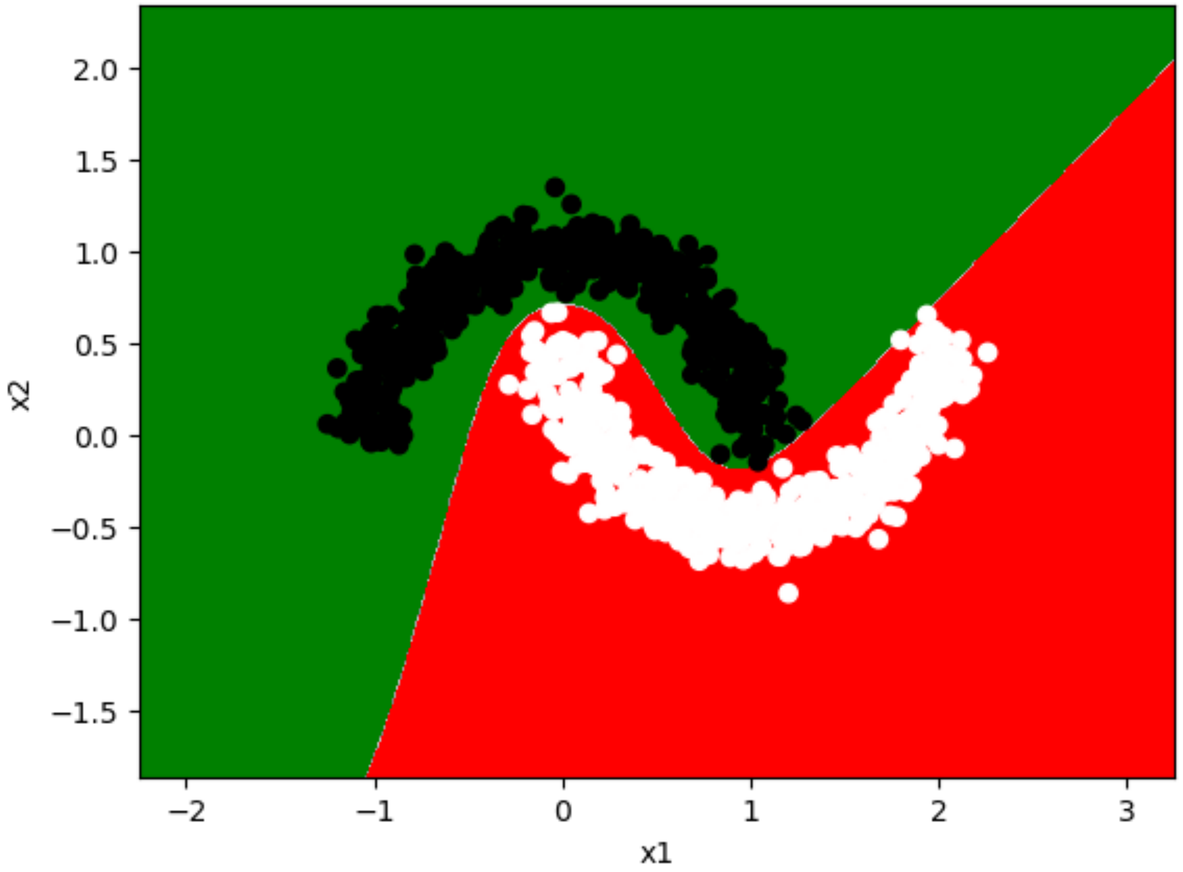
Accuracy = 0/99

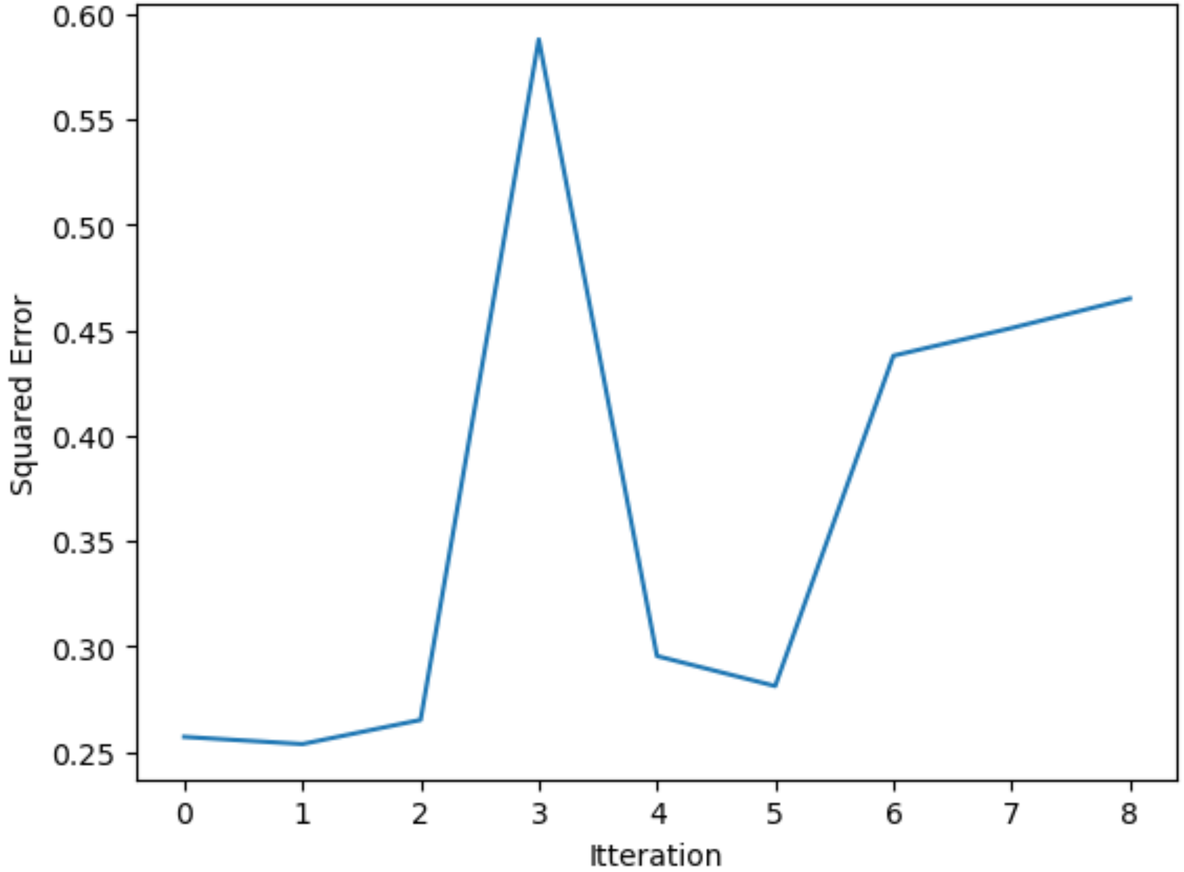




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a declining factor and online condition with the tangent-hyperbolic function in both layers

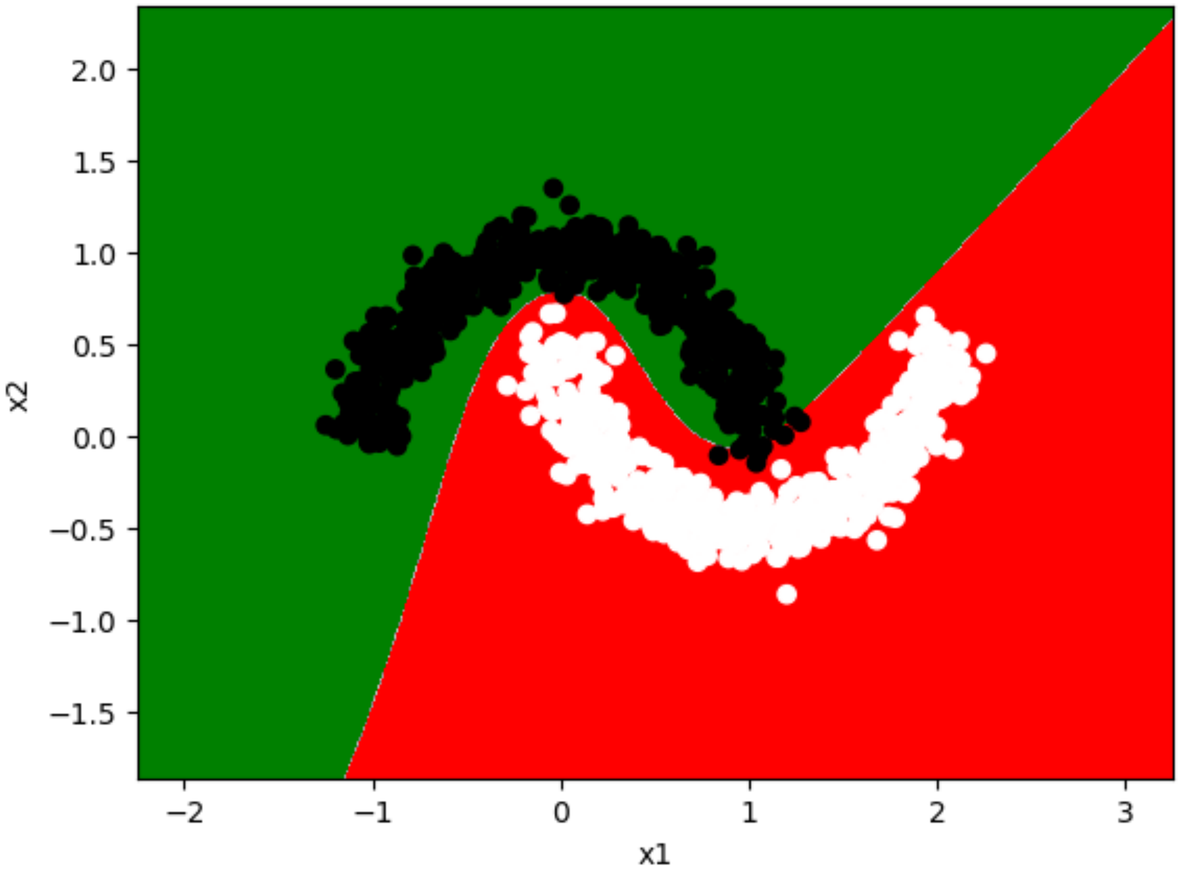
Accuracy = 1/0

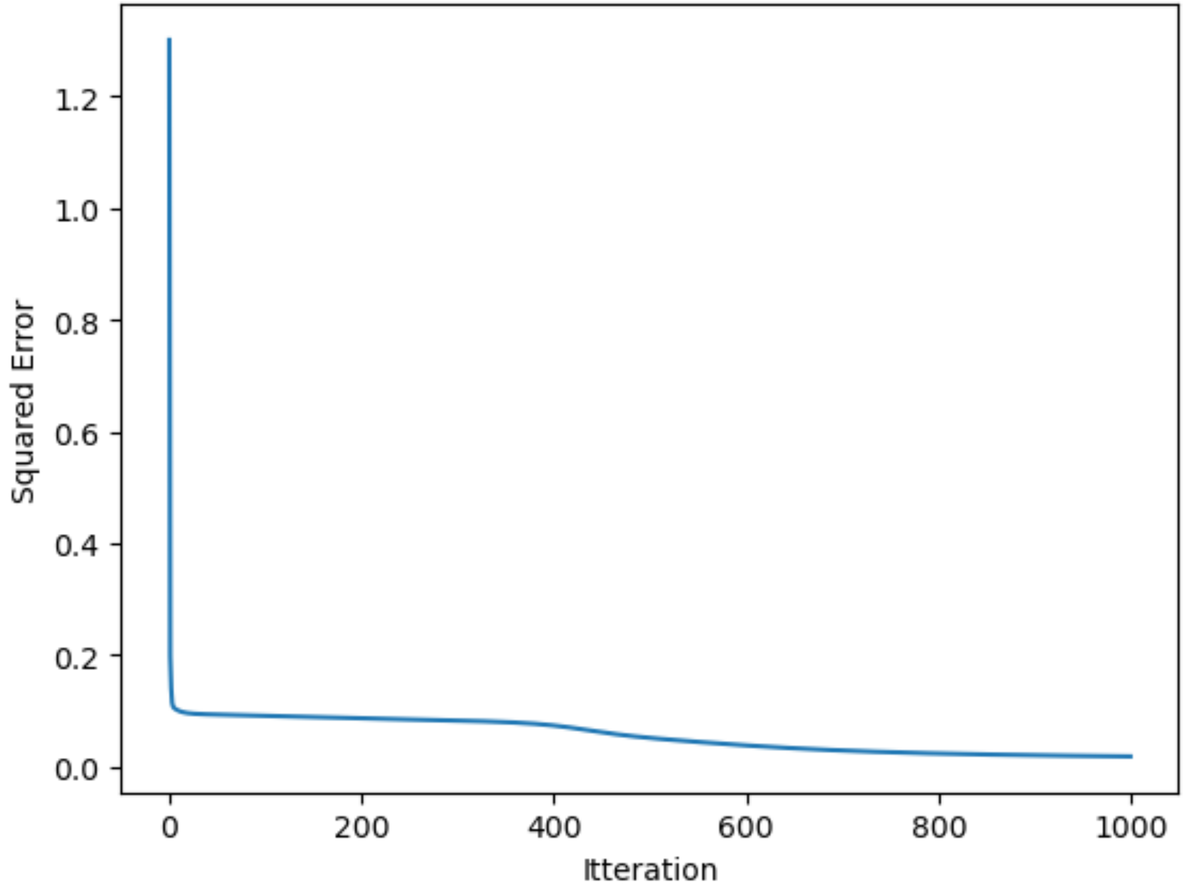




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a constant factor and Batch condition with the tangent-hyperbolic function in both layers

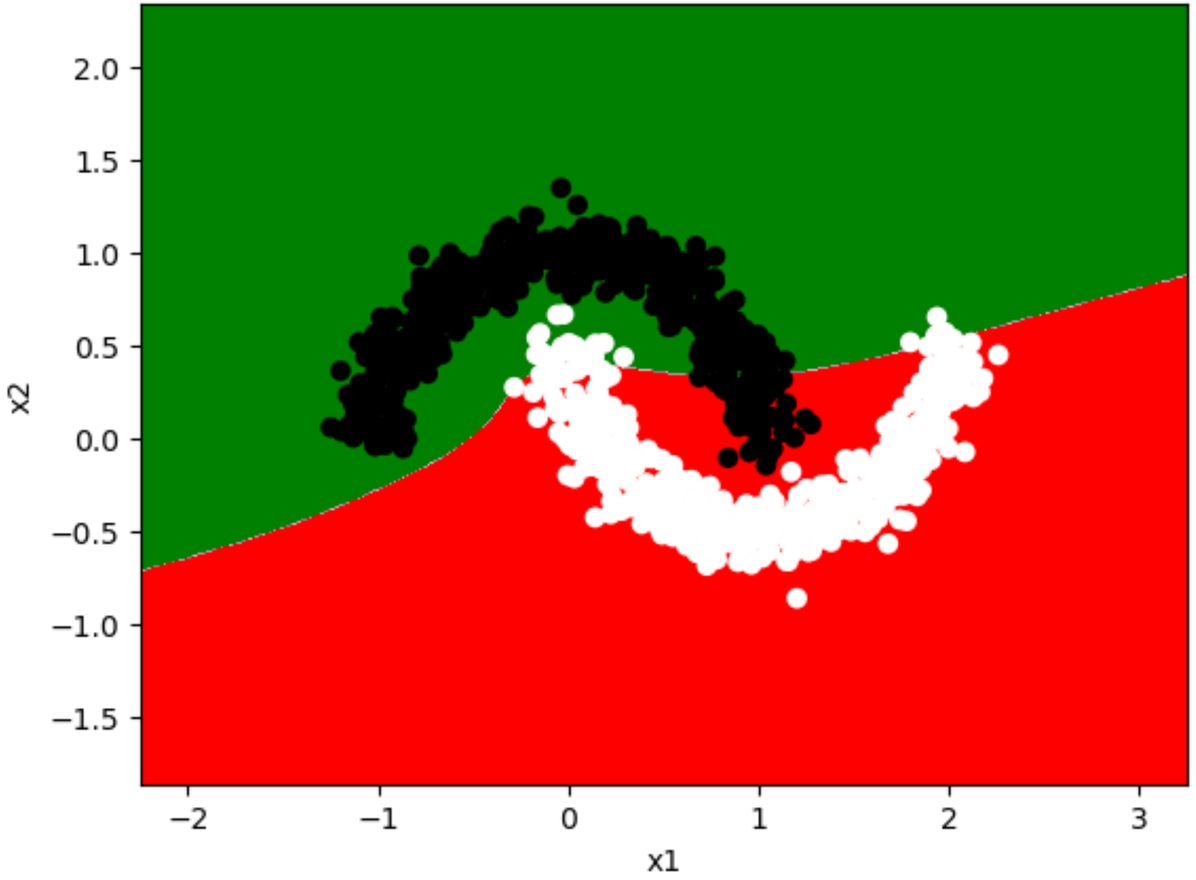
Accuracy = 0/99

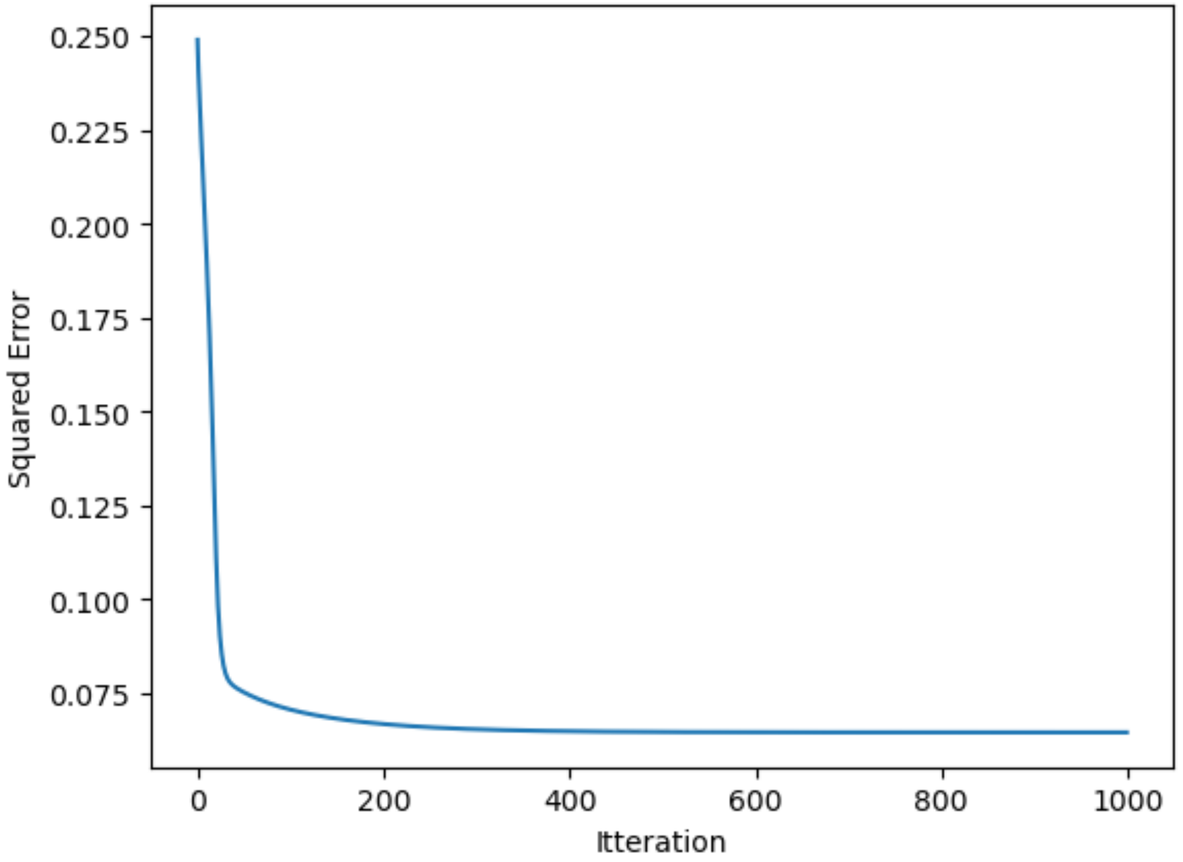




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a declining factor and Batch condition with the tangent-hyperbolic function in both layers

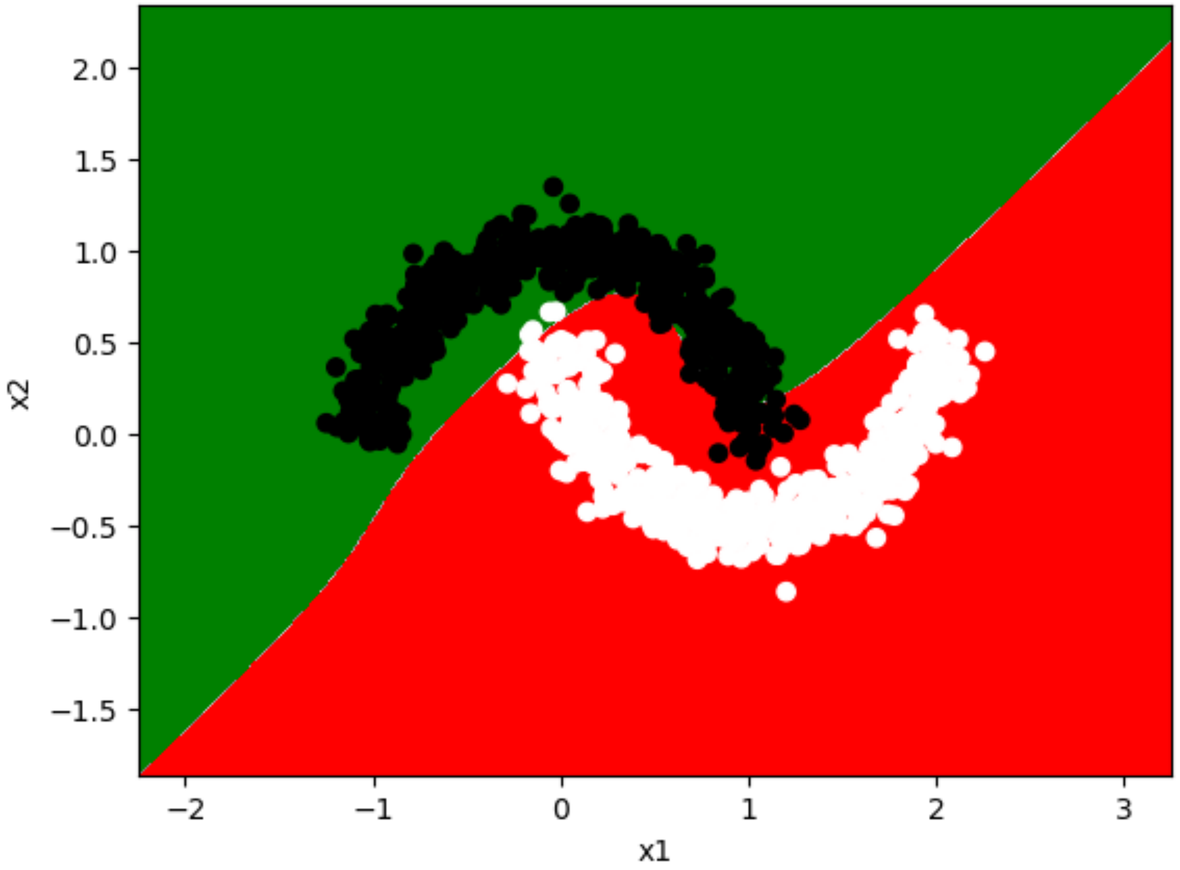
Accuracy = 0/91333

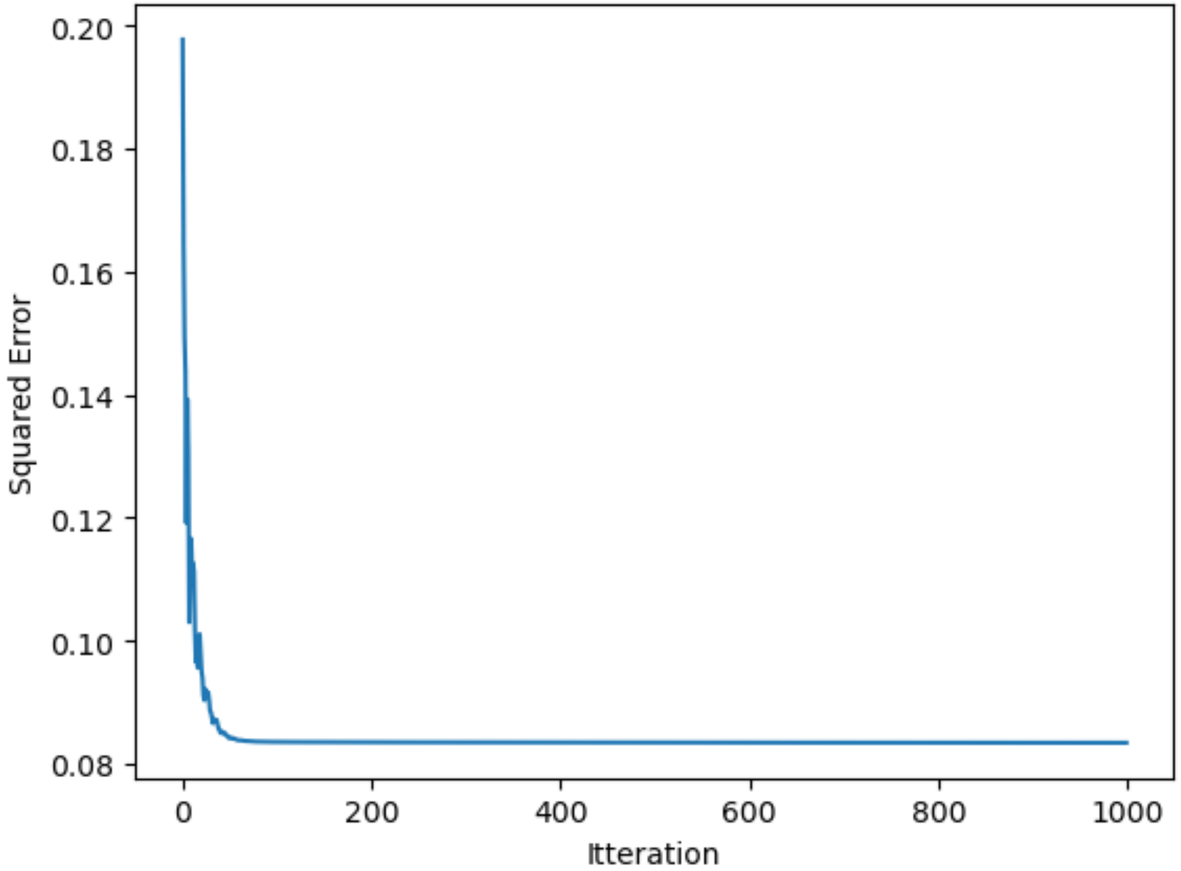




Plotting the dataset with the driven boundary of a neural network with a hidden layer and training with a constant factor and Batch condition with the tangent-hyperbolic function in both layers

Accuracy = 0/8833





As you can see, the graph of the errors resulting from testing the model in online mode does not have a complete descending mode. It is because when each input is entered, the model tries to learn. But in closed cases, you can see that the graph of MSE errors resulting from the model test is always downward and it is the most favorable indicator of the model. Also, in the closed mode, the number of neurons in the hidden layer is more than in the online mode.

When using the tanh activity function, it can be seen that the model errors converge to zero faster, and this shows the superiority of this activity function over the Sigmoid function. Also, when not using a constant coefficient and using a decreasing coefficient, it is observed that the model errors converge to zero faster with the initialization of a large growth rate.