Implementation of the perceptron learning rule

Kian Anvari Hamdani

In [3]:
```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt


iris = datasets.load_iris()
inputs = iris.data[:,2:4]
labels = iris.target
```

In [2]:
```
for index in range(len(labels)):

    if labels[index] == 0:
        pass
    else :
        labels[index] = 1
```

In [3]:
```
inputs = np.array(inputs)
labels = np.array(labels)
labels = labels.reshape(150,1)

dataset = np.append(inputs, labels, axis=1)

print(dataset)
```

```
[[1.4 0.2 0. ]
 [1.4 0.2 0. ]
 [1.3 0.2 0. ]
 [1.5 0.2 0. ]
 [1.4 0.2 0. ]
 [1.7 0.4 0. ]
 [1.4 0.3 0. ]
 [1.5 0.2 0. ]
```

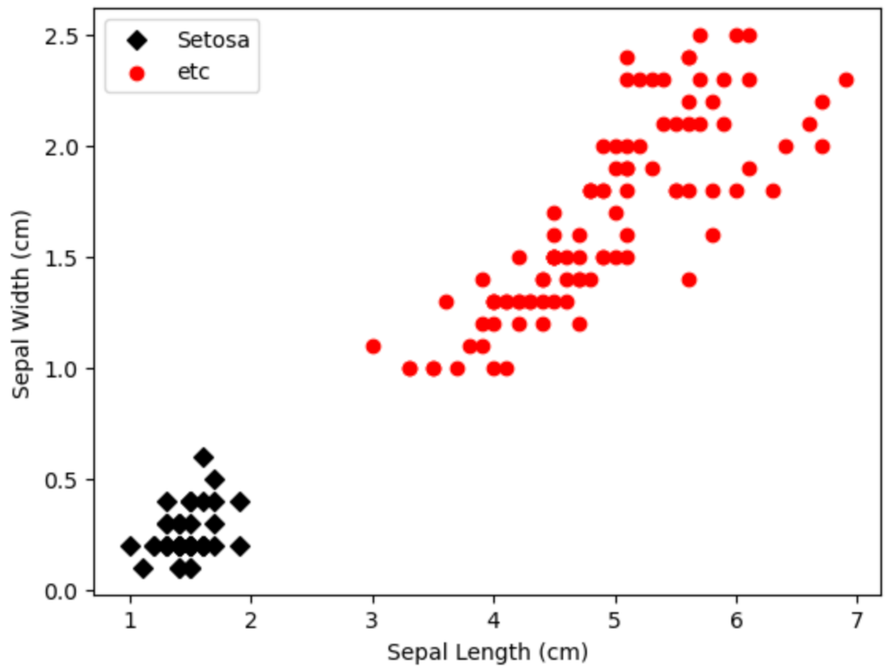
Use and import the required libraries to solve the question and implement it:

Sklearn - datasets:
Import the iris dataset to classify it

numpy:
To use its properties and functions in working with matrices

Matplotlib:
To show and draw the graph of the inputs and draw the obtained model
In the case of the question, it is said to classify the inputs using dimensions number and 3.
We store these dimensions in the inputs variable.
We save the labels of the flowers in the labels variable.
As mentioned in the question form, we combine two classes 1 and 2.
Finally, we rewrite the desired dataset to solve the question and store it in the dataset variable that contains the 2nd and 3rd dimensions of the inputs along with their labels.

In [4]:
```
plt.scatter(

                dataset[:,0][dataset[...,2]==0],
                dataset[:,1][dataset[...,2]==0],
                marker = "D",
                color = "Black",
                label = "Setosa"

)

plt.scatter(

                dataset[:,0][dataset[...,2]==1],
                dataset[:,1][dataset[...,2]==1],
                marker = "o",
                color = "Red",
                label = "etc"
)



plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend()
plt.show()
```



Using the scatter function in the Matplotlib library, we draw our inputs, which are from two classes, in the form of dots on the graph:

First, we put the dimensions of the inputs whose class or label is zero in black color and then the class 1 inputs in red color to the diagram to draw them:

In [5]:
```
def set_weights(n):

    weights = np.random.randn(3, n)
    return weights
```

In [6]:
```
def calculate_r(label, y):

    return label - y
```

In [7]:
```
def calculate_output(inputs,weights):

    net = np.dot(weights.T,inputs)
    return np.heaviside(net,1)
```

This function is written to create a matrix of weights with random values. In the case of a question, it is indicated to implement this question with m neurons. We take the number of neurons in the input of this function, which is for building the weights matrix. Finally, we make a matrix with 3 dimensions (one dimension for bias or passing from the origin) in the number of rows and columns.

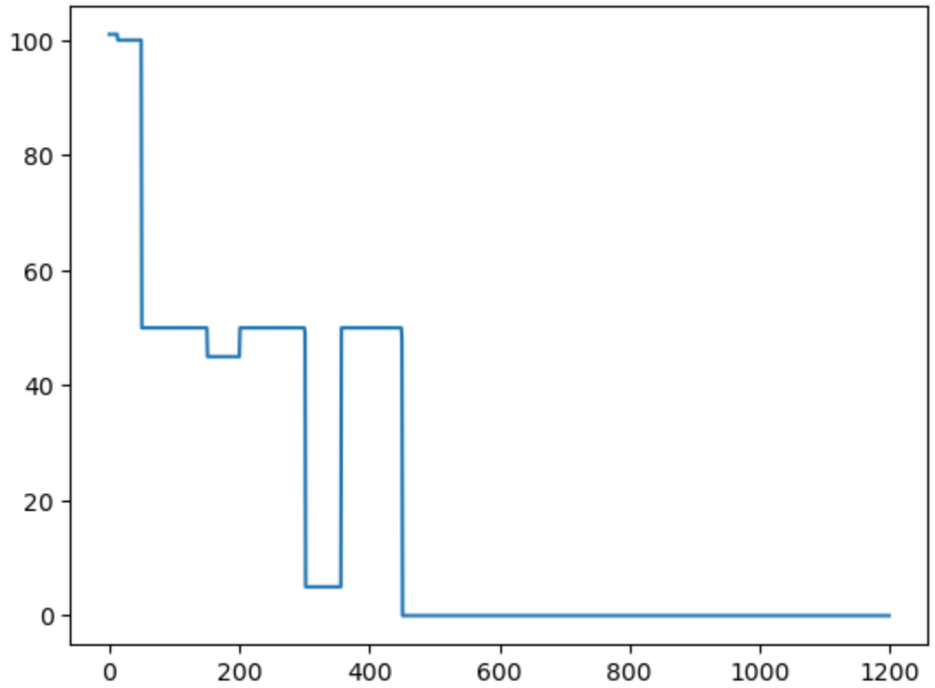
This function is intended to calculate the output error and the value produced by the input. It calculates the error relative to the label of the same input.
This function is written to calculate the output of the neuron.
It does this by multiplying the input values in the weights matrix and outputting the value of the inner multiplication to the step function for calculation.

In [8]:
```
def update_weights(inputs, weights, label, y):

    delta_W = calculate_r(label, y) * inputs

    weights = weights + delta_W

    return weights
```

```
n = 1
weights = set_weights(n)
epochs = 8
errors = []
for i in range(epochs):
    for index in range(len(inputs)):

        input_train = np.append(np.ones(1),inputs[index])
        input_train = input_train.reshape(3,1)
        output = calculate_output(input_train,weights)
        error = 0
        weights = update_weights(input_train, weights, labels[index], output)

        for j in range(len(inputs)):

            input_train = np.append(np.ones(1),inputs[j])
            input_train = input_train.reshape(3,1)
            output = calculate_output(input_train,weights)
            error += (labels[j][0]-output[0][0])**2

        errors.append(error)
```

This function is written to update and train the weight matrix, which also uses the error calculation function to do this.
We solve this question by a neuron and specifying its random weights.
The implementation is such that it repeats the training process a certain number of times (the number of repetitions is in the epochs variable)
The training is based on checking all the inputs and calculating the probability error resulting from the neuron to update the weights matrix to get a good discriminating model.
After correcting the weight after training the weights by entering an input - we save the error of all the inputs with that trained weight matrix so that we can draw the graph of the reduction of the error rate in the training in the next slide.

In [19]:
```
index = []

for i in range(0,150*epochs):
    index.append(i)


plt.plot(index,errors)
plt.show()
```



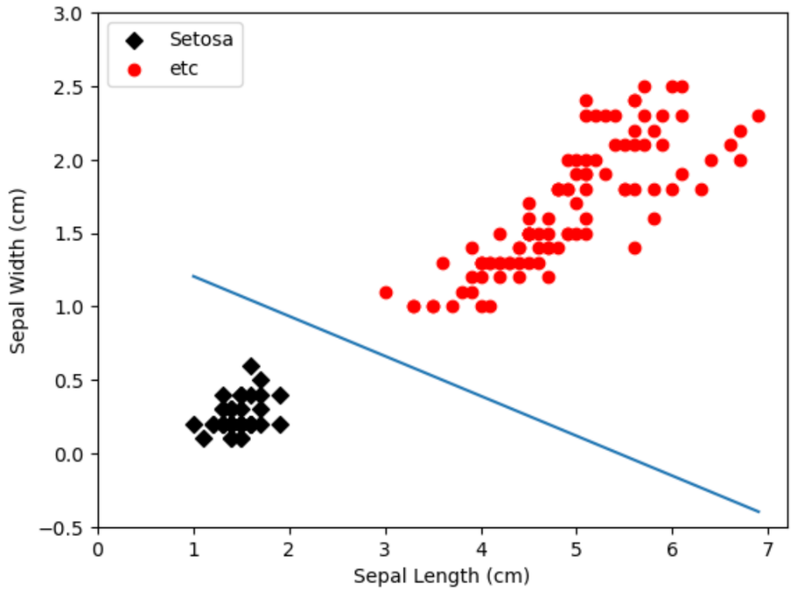
In [11]:
```
print(weights)
```

```
[[-4.83750957]
 [ 1.82452222]
 [ 1.42889695]]
```

You can see the error graph calculated in each step of the training. The resulting error is decreasing at each stage of training.

Finally, you can see the final weight matrix obtained after training.

In [11]:
```
plt.scatter(

                dataset[:,0][dataset[...,2]==0],
                dataset[:,1][dataset[...,2]==0],
                marker = "D",
                color = "Black",
                label = "Setosa"

)

plt.scatter(

                dataset[:,0][dataset[...,2]==1],
                dataset[:,1][dataset[...,2]==1],
                marker = "o",
                color = "Red",
                label = "etc"
)


# x0W0 + x1W1 + x2W2 = 0
x1 = [min(inputs[:,0]), max(inputs[:,0])]
m = -weights[1]/weights[2]
c = -weights[0]/weights[2]
x2 = m*x1 + c

plt.plot(x1, x2)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend()
plt.xlim([0, 7.2])
plt.ylim([-0.5, 3])

plt.show()
```



Using the scatter function in the Matplotlib library, we draw our inputs, which are from two classes, in the form of dots on the graph:

First, we put the dimensions of the inputs which class or label is zero in black color, and then the class 1 inputs in red color to the diagram to draw them:

Then, using the graph, we draw the separating line resulting from the training of the weights.