**Perceptron Learning Algorithm Code**

In [1]:

**import** numpy **as** np

In [2]:

W **=** np**.**zeros(2**+**1)

W

Out[2]:

array([0., 0., 0.])

In [3]:

X**=**[2,3]

np**.**insert(X, 0, 1)

Out[3]:

array([1, 2, 3])

In [4]:

*# initialization code*

**def** \_\_init\_\_(self, input\_size, lr**=**1, epochs**=**10, bias**=**1):

self**.**W **=** np**.**zeros(input\_size**+**bias)

self**.**epochs **=** epochs

self**.**lr **=** lr

In [5]:

*# Activation function code which is a simple step function*

**def** activation\_fn(self, x):

*#return (x >= 0).astype(np.float32)*

**return** 1 **if** x **>=** 0 **else** 0

In [6]:

*# Calculating dot product of W and X (input vector) and applying step function*

**def** predict(self, x):

z **=** self**.**W**.**T**.**dot(x)

a **=** self**.**activation\_fn(z)

**return** a

*# Perceptron Learning code running all the samples for given epochs or iterations*

**def** fit(self, X, OutputLabel):

no\_of\_smaples**=**4

**for** \_ **in** range(self**.**epochs):

**for** i **in** range(no\_of\_samples):

y **=** self**.**predict(X[i])

e **=** OutputLabel[i] **-** y

self**.**W **=** self**.**W **+** self**.**lr **\*** e **\*** np**.**insert(X[i], 0, 1)

**The complete executable code of Perceptron model**

In [8]:

**class** Perceptron(object):

"""Implements a perceptron network"""

**def** \_\_init\_\_(self, input\_size, lr**=**1, epochs**=**100):

self**.**W **=** np**.**zeros(input\_size**+**1)

*# add one for bias*

self**.**epochs **=** epochs

self**.**lr **=** lr

self**.**loss\_lst **=** []

**def** activation\_fn(self, x):

*#return (x >= 0).astype(np.float32)*

**return** 1 **if** x **>=** 0 **else** 0

**def** predict(self, x):

z **=** self**.**W**.**T**.**dot(x)

a **=** self**.**activation\_fn(z)

**return** a

**def** fit(self, X, d):

**for** \_ **in** range(self**.**epochs):

**for** i **in** range(d**.**shape[0]):

x **=** np**.**insert(X[i], 0, 1)

y **=** self**.**predict(x)

e **=** d[i] **-** y

self**.**W **=** self**.**W **+** self**.**lr **\*** e **\*** x

self**.**loss\_lst**.**append(e)

*# Perceptron Learning code running all the samples for given epochs or iterations*

**def** fit(self, X, OutputLabel):

no\_of\_smaples**=**4

**for** \_ **in** range(self**.**epochs):

**for** i **in** range(no\_of\_samples):

y **=** self**.**predict(X[i])

e **=** OutputLabel[i] **-** y

self**.**W **=** self**.**W **+** self**.**lr **\*** e **\*** np**.**insert(X[i], 0, 1)

**AND GATE EXAMPLE WITH NO OF SAMPLES/RECORDS AS 4 AND EPOCHS AS 100**

In [9]:

**if** \_\_name\_\_ **==** '\_\_main\_\_':

X **=** np**.**array([

[0, 0],

[0, 1],

[1, 0],

[1, 1]

])

d **=** np**.**array([0, 1, 1, 1])

perceptron **=** Perceptron(input\_size**=**2)

perceptron**.**fit(X, d)

print(perceptron**.**W)

[-1. 1. 1.]

In [10]:

**import** matplotlib.pyplot **as** plt

x\_axis **=** [int(x) **for** x **in** range(100)]

y\_axis **=** perceptron**.**loss\_lst

plt**.**plot(x\_axis, y\_axis)

plt**.**xlabel("iteration")

plt**.**ylabel("loss")

plt**.**show()



**Using the AND gate data, we should get a weight vector of [-3, 2, 1]. This means that the bias is -3 and the weights are 2 and 1 for x\_1 and x\_2, respectively.**

**LETS TEST MANUALLY**

**let us test for x=[0, 1]**

In [11]:

x**=**[1, 0, 1]

y**=** **-**3**\***1**+**2**\***0**+**1**\***1

y

Out[11]:

-2

# since it is a negative value on applying activation function we get zero which is correct