In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
np.random.seed = 69
```

In [2]:

```
#if something is sus check bias addition

def sigmoid(x):
    sig = 1 / (1 + math.exp(-x))
    return sig

def squish(z):
    a = []
    for row in z:
        temp = []
        for ele in row:
            temp.append(sigmoid(ele))
        a.append(temp)
    return(np.array(a))

def squishprime(z):
    f = []
    for row in z:
        temp = []
        for ele in row:
            temp.append(sigmoid(ele)*(1-sigmoid(ele)))
        f.append(temp)
    return(np.array(f))

def stackmult(X, Y): #mutliplies numbers of matrices index by index
    Z =[]
    for i in range(len(X)):
        temp = []
        for j in range(len(X[0])):
            temp.append(X[i][j]*Y[i][j])
        Z.append(temp)
    return np.array(Z)
```

In [3]:

```
class NeuralNetwork():
    #lis contains sizes of each layer
    def __init__(self, lis):

        self.n = len(lis) # number of layers in total

        self.reset()
        self.WML = [[[0]] ] #dummy weight matrix associated with first layer
        self.BL = [[[0]] ] #dummy bias matrix/list associated with first layer


        #np.random.seed = 34
        #initialize random weight matrices here...
        for i in range(len(lis)-1):
            #random weight matrix per layer
            wm = np.random.randint(-5, 6, (lis[i], lis[i+1])).astype(np.float32)
            self.WML.append(wm)
            #random biases per layer
            b = np.random.randint(-5, 6, (1, lis[i+1])).astype(np.float32)
            self.BL.append(b)

    def reset(self):
        self.AL = [] #list of layer activations
        self.FprimeL = [[] ] #AL0 does not have z value to find fprime
```

```
        self.DL = [[[]] ] #AL0 does not have delta values
        self.err =[[]]
        self.C =[]
        self.J = 0

    def forward(self, X):
        self.reset()
        self.AL.append(X)

        for i in range(self.n-1):
            wm = self.WML[i+1]
            b = self.BL[i+1]

            z = np.matmul(self.AL[i], wm) + b

            a = squish(z)
            self.AL.append(a)

            f = squishprime(z)
            self.FprimeL.append(f)

        return self.AL[-1]

    def backward(self, Y):

        self.err = self.AL[-1] - Y

        #some extra calc that can be moved around as seen fit
        err2 = stackmult(self.err, self.err)
        self.C = [sum(i) for i in err2] # C[t] gives cost of a sample
        self.J = sum(self.C) #total cost after running a batch


        #delta of layers by backprop
        #then loop through with E = np.matmul(Dl2, wm.T)
        for i in range(1, self.n):
            #i = 1 represents the last layer L
            #then you count backwards
            if(i == 1):
                E = self.err
            else:
                wm = self.WML[-i + 1]  #weights of l+1 layer
                E = np.matmul(self.DL[1], wm.T) #propogated errors

            f = self.FprimeL[-i]
            d = stackmult(f, E)
            self.DL.insert(1, d)

    def updateWeights(self, alpha):
        for i in range(1, self.n):
            al1 = self.AL[i-1]
            dl2 = self.DL[i]
            changeInWeights = np.matmul(al1.T, dl2)

            self.WML[i] = self.WML[i] - alpha*changeInWeights
            #print(f"Successfully update weights of layer {i}")


    def train(self, X, Y, epochs=10, alpha=0.2):
        Jprogress = []
        for epoch in range(epochs):
            self.forward(X)
            self.backward(Y)
            self.updateWeights(alpha)

            Jprogress.append(self.J)

        plt.plot(Jprogress, 'ro')
```

In [4]:

```
net = NeuralNetwork([3, 2, 2])
```

```python
#np.random.seed = 69
X = np.random.random(12).reshape(4,3)
Y = np.random.random(8).reshape(4,2)
```

In [4]:

```python
print("Required output : ")
print(Y)

print("\nInitialized neural network predicts : ")
print(net.forward(X))
```

```
Required output :
[[0.56205578 0.66316028]
 [0.23728235 0.16471085]
 [0.30116347 0.07575179]
 [0.22517367 0.63821744]]

Initialized neural network predicts :
[[0.00078673 0.0075441 ]
 [0.01179782 0.00319507]
 [0.00417952 0.0037232 ]
 [0.01037433 0.00299575]]
```

In [5]:

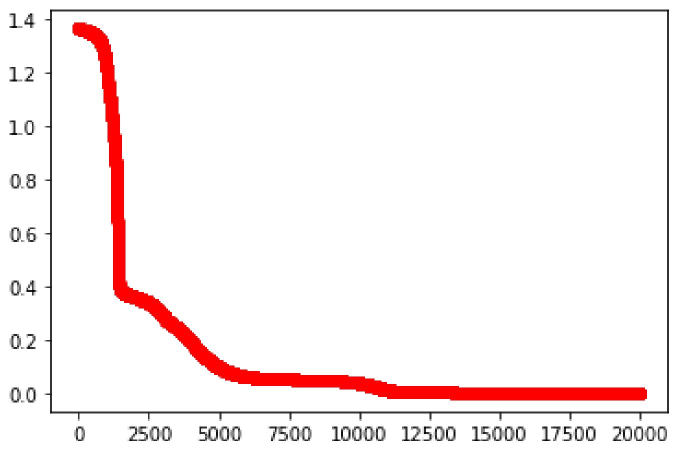
```python
e = net.forward(X) - Y
maxerr = max(e[0])
for rec in e:
    if(maxerr < max(rec)):
        maxerr = max(rec)
maxerr
```

Out[5]:

```
-0.07202858551258975
```

In [6]:

```python
net.train(X, Y, 20000)
```



In [8]:

```python
e = net.forward(X) - Y
maxerr = max(e[0])
for rec in e:
    if(maxerr < max(rec)):
        maxerr = max(rec)
print('max error = ', maxerr)

print("\nRequired output : ")
print(Y)

print('\nPrediction : \n')
print(net.forward(X))
```

```
max error =  0.016994398026884378

Required output :
[[0.56205578 0.66316028]
 [0.23728235 0.16471085]
 [0.30116347 0.07575179]
 [0.22517367 0.63821744]]

Prediction :

[[0.55142481 0.66626994]
 [0.22084067 0.17404616]
 [0.31431765 0.05819487]
 [0.24216806 0.63237453]]
```