Name : Chathumini B.G.D.T.

Index Number : 190107T

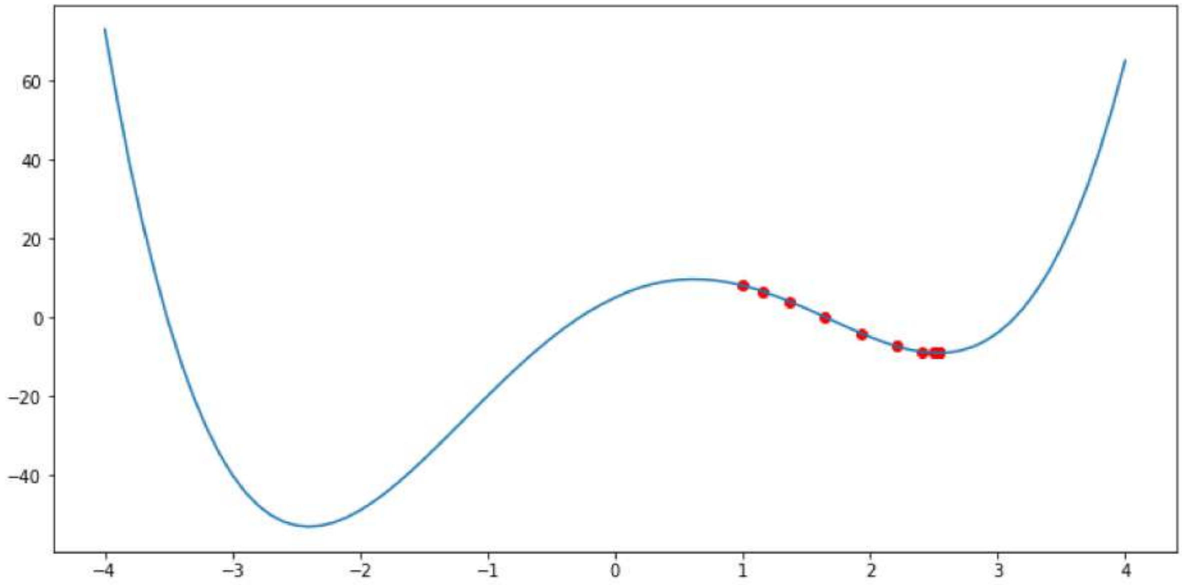
Github Repo: https://github.com/dulmi-19/Image-Processing-and-Machine-Vision

Question 1 (a)

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

def f(x):
    w = np.array([1,-1,-12,15,5])
    M = np.size(w)-1
    return np.sum([x**i*w[M-i] for i in range(0,M+1)], axis=0)

def g(x):
    w = np.array([1,-1,-12,15,5])
    M = np.size(w)-1
    return np.sum([i*x**(i-1)*w[M-i] for i in range(0,M+1)], axis=0)

alpha = 0.02 #Learning rate
x = 1 #Initial solution
x_hist = np.array(x)
fx_hist = np.array(f(x))
for i in range(20):
    x = x - alpha*g(x)
    x_hist= np.append(x_hist, x)
    fx_hist= np.append(fx_hist, f(x))

print('x= ',x,'f(x) = ',f(x))

fig = plt.figure(figsize = (12,6))
ax = plt.subplot(1,1,1)
delta = 0.1
x_ = np.arange(-4,4+delta,delta)
ax.plot(x_,f(x_))
ax.scatter(x_hist,fx_hist, c='r')
```

```
x=  2.533858129332268 f(x) =  -9.083837308516742
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x2008a78e860>
```



In [ ]:
```python
# finding a root close to x0
from scipy.optimize import fsolve
from scipy.optimize import minimize
x0=0.7
root = fsolve(g,x0) #gradient is zero ath this point
print('Innitial Solution: ', root)

#Using scipy to find minimum
minimum = minimize(f,x0)
print(minimum)
```

```
Innitial Solution:  [0.61654501]
      fun: -9.083837308515939
 hess_inv: array([[0.02625738]])
      jac: array([-7.62939453e-06])
  message: 'Optimization terminated successfully.'
     nfev: 16
      nit: 3
     njev: 8
   status: 0
  success: True
        x: array([2.53385792])
```

There is a hill in the function at x = 0.61654501. Since we have given x = 1 as the Initial solution ,we get (2.533858129332268 , -9.083837308516742) as the minimum point of the function. But this is a wrong minimum point.

If we assign x with a number greater than the innitial solution, we get a wrong minimum point for this function.

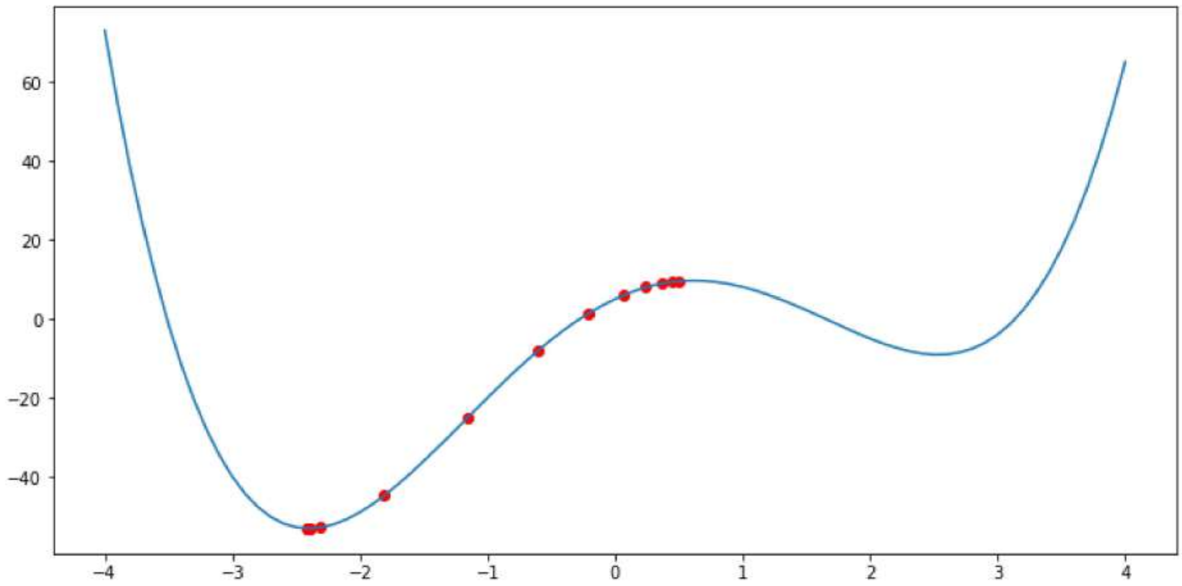
Lets assign x with a number less than the innitial solution(0.61654501)

Try x = 0.5 :

In [ ]:
```python
alpha = 0.02 #Learning rate
x = 0.5 #Initial solution
x_hist = np.array(x)
fx_hist = np.array(f(x))
for i in range(20):
    x = x - alpha*g(x)
    x_hist= np.append(x_hist, x)
    fx_hist= np.append(fx_hist, f(x))

print('x= ',x,'f(x) = ',f(x))

fig = plt.figure(figsize = (12,6))
ax = plt.subplot(1,1,1)
delta = 0.1
x_ = np.arange(-4,4+delta,delta)
ax.plot(x_,f(x_))
ax.scatter(x_hist,fx_hist, c='r')
```

```
x=  -2.400403139786586 f(x) =  -53.11840483801493
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x2008a802050>
```



Now we get the correct minimum point of the function.(-2.400403139786586 , -53.11840483801493) Therefore when using gradient descent to find the minimum, Innitial solution matters.

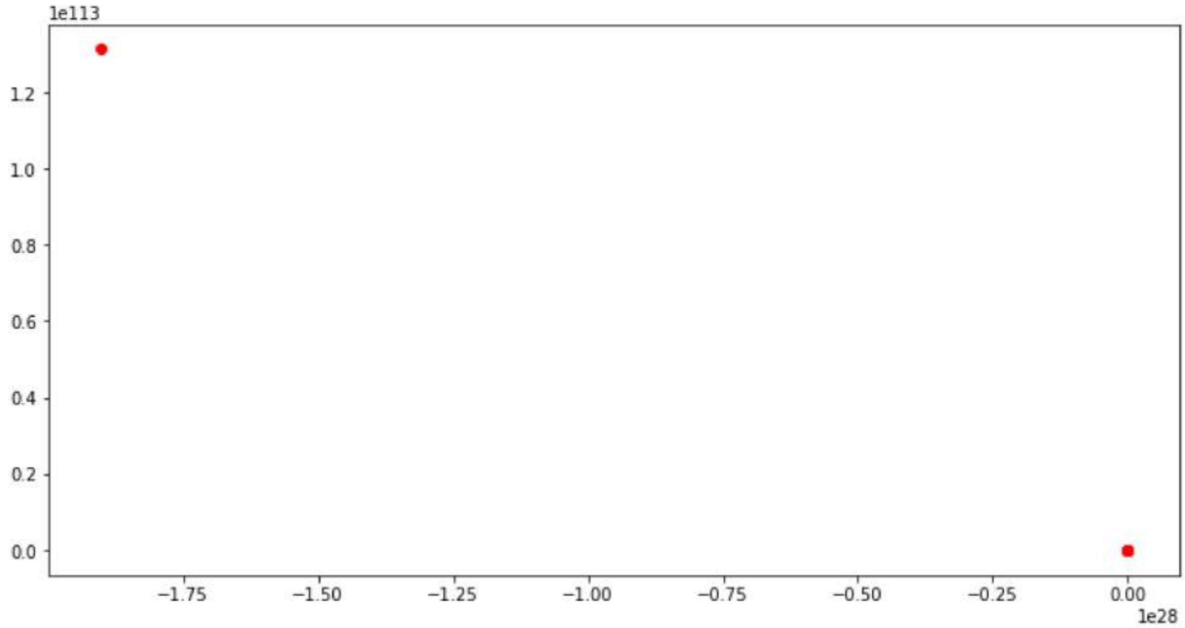
Question 1 (b)

Lets change Learning rate value to a larger value. (alpha = 0.2) :

In [ ]:
```
alpha = 0.2 #Learning rate
x = 0.5 #Initial solution
x_hist = np.array(x)
fx_hist = np.array(f(x))
for i in range(20):
    x = x - alpha*g(x)
    x_hist= np.append(x_hist, x)
    fx_hist= np.append(fx_hist, f(x))

print('x= ',x,'f(x) = ',f(x))

fig = plt.figure(figsize = (12,6))
ax = plt.subplot(1,1,1)
delta = 0.1
x_ = np.arange(-4,4+delta,delta)
ax.plot(x_,f(x_))
ax.scatter(x_hist,fx_hist, c='r')
```

```
x=  nan f(x) =  nan
C:\Users\Acer\AppData\Local\Temp\ipykernel_7756\1330090998.py:7: RuntimeWarning: o
verflow encountered in double_scalars
  return np.sum([x**i*w[M-i] for i in range(0,M+1)], axis=0)
c:\Python310\cv\lib\site-packages\numpy\core\fromnumeric.py:86: RuntimeWarning: in
valid value encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
C:\Users\Acer\AppData\Local\Temp\ipykernel_7756\1330090998.py:12: RuntimeWarning:
overflow encountered in double_scalars
  return np.sum([i*x**(i-1)*w[M-i] for i in range(0,M+1)], axis=0)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x20085df4430>
```



When lerning rate is increased to 0.2, we don't get a number as minimum point. (x= nan f(x) = nan). So this learning rate is too high.

Therefore, it is clear that the learning rate must be tuned carefully.

Question 2

In [ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib . pyplot as plt
from tensorflow.keras.datasets import cifar10 , mnist

(x_train,y_train) ,(x_test,y_test) = cifar10.load_data( )
# ( x_train , y_train ) , ( x_test , y_te st ) = mnist . load_data ( )
print(" x_train => " , x_train.shape)

Ntr = x_train.shape[ 0]
Nte = x_test.shape[ 0]
Din = 3072 # CIFAR10
 # Din = 784 # MINIST
x_train = x_train [range(Ntr),:]
x_test = x_test [range(Nte),:]
y_train = y_train [range(Ntr) ]
y_test = y_test [range(Nte)]

K = len(np.unique(y_train))

y_train = tf.keras.utils.to_categorical(y_train,num_classes=K)
y_test= tf.keras.utils.to_categorical(y_test,num_classes=K)

x_train = np.reshape(x_train,(Ntr,Din))
x_test= np.reshape(x_test,(Nte,Din))
x_train=x_train.astype(np.float32)
x_test = x_test.astype(np.float32)

x_train /= 255.
x_test /=255.
```

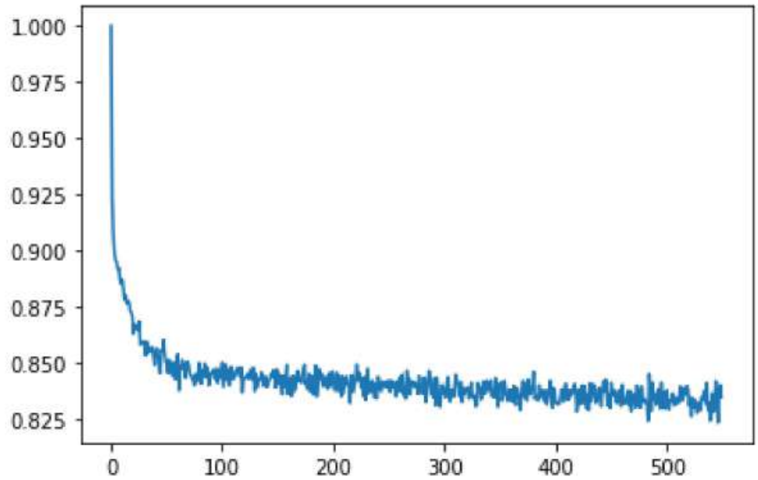
```
 x_train =>  (50000, 32, 32, 3)
```

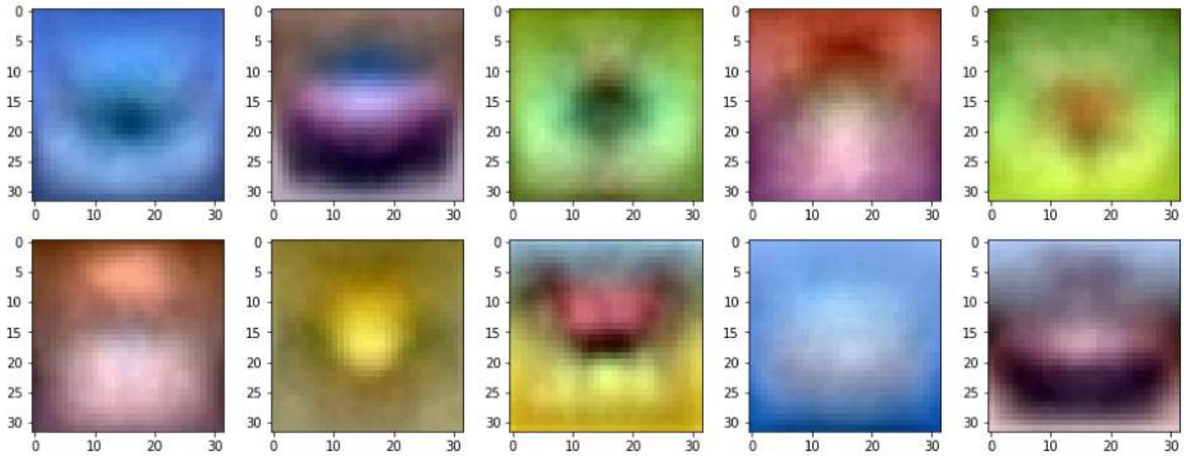
In [ ]:
```
def display(y_train, y_test, y_train_pred, y_test_pred, loss_history, w, showim = 
    plt.plot(loss_history)

     # For diapaying the weights matrix w as an image. 32*32*3 assumption is there
    if showim:
        f, axarr = plt.subplots(2, 5)
        f.set_size_inches(16, 6)
        for i in range(10):
            img = w[:, i].reshape(32, 32, 3)# CIFAR10
            # img = w1[:, i].reshape(28, 28)# MNIST
            img = (img - np.amin(img))/(np.amax(img) - np.amin(img))
            axarr[i//5, i%5].imshow(img)
        plt.show()

    train_acc = np.mean(np.abs(np.argmax(y_train, axis=1) == np.argmax(y_train_pre
    print("train_acc = ", train_acc)

    test_acc = np.mean(np.abs(np.argmax(y_test, axis=1) == np.argmax(y_test_pred, a
    print("test_acc = ", test_acc)
```

In [ ]:
```
std =1e-5
w = std*np.random.randn(Din,K)
b = np.zeros(K)
lr = 1e-3
lr_decay=0.1
epochs =11
batch_size=1000
loss_history = []
rng = np.random.default_rng(seed=0)

for e in range(epochs):
    indices = np.arange(Ntr)
    rng.shuffle(indices)
    for batch in range(Ntr//batch_size):
        batch_indices = indices[batch*batch_size:(batch+1)*batch_size]
        x =x_train[batch_indices]#Extract a batch of 100
        y = y_train[batch_indices]

        #Forward pass
        y_pred = x@w+b
        loss=1./batch_size*np.square(y_pred-y).sum()
        loss_history.append(loss)

        #backward pass
        dy_pred =1./batch_size*2.0*(y_pred-y)
        dw = x.T @ dy_pred
        db = dy_pred.sum(axis=0)*1
        w=w-lr*dw
        b = b-lr*db

    if e % 5==0:
        print("Iteration %d / %d: loss %f"%(e,epochs,loss))

    if e % 10==0:
        lr *= lr_decay
```

```
Iteration 0 / 11: loss 0.850465
Iteration 5 / 11: loss 0.836767
Iteration 10 / 11: loss 0.834917
```

In [ ]:
```
y_train_pred = x_train.dot(w)+b
y_test_pred = x_test.dot(w)+b
display(y_train,y_test,y_train_pred,y_test_pred,loss_history,w,showim=True)
```





```
train_acc =  0.33566
test_acc =  0.3349
```