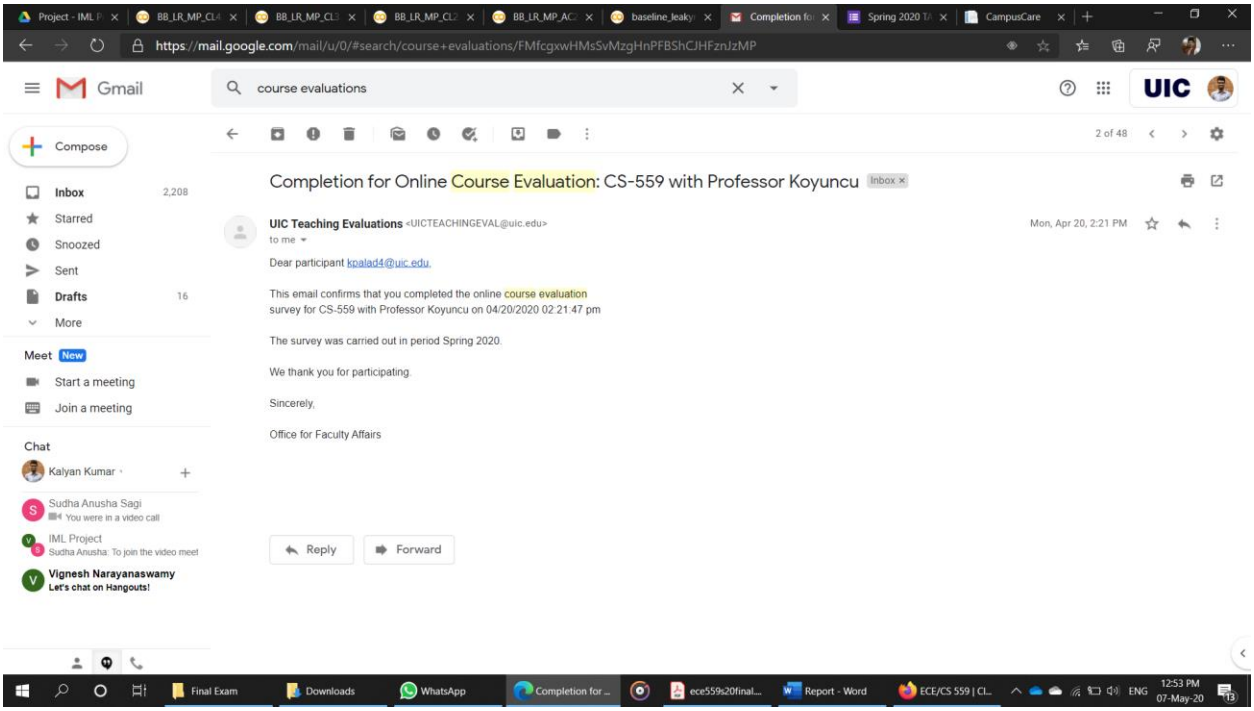
Name: Kalyan Kumar Paladugula,

NetId: kpalad4

UIN: 679025059



Name: [illegible]
NetID:- Kpalad4

CS559 - Neural Networks

Final Exam

Q1) Given

$$S_t = \tanh(w x_t + u S_{t-1}) \text{ where } t = 1, 2 \ldots K.$$

$S_0 = 0$

$S_1 = \tanh(w x_1)$ - Independent of u

$S_2 = \tanh(w x_2 + u S_1)$

Given $x_1, \ldots x_K$ we have to predict x_{K+1}

$$E = \frac{1}{2}\left(S_K - x_{K+1}\right)^2$$

a) $$\frac{\partial E}{\partial u} = \frac{1}{2} \cdot 2 \left(S_2 - x_3\right) \cdot \frac{\partial}{\partial u}(S_2)$$

As $\tanh'(x) = 1 - (\tanh(x))^2$

$$\Rightarrow \frac{\partial E}{\partial u} = \left(S_2 - x_3\right) \cdot \left(1 - S_2^2\right) \frac{\partial}{\partial u}\left(u S_1\right)$$

$$= \left(S_2 - x_3\right)\left(1 - S_2^2\right) S_1$$

Now, put $S_2 = \tanh(w x_2 + u S_1)$

$S_1 = \tanh(w x_1)$

$$\frac{\partial E}{\partial u} = \left[\tanh(wx_2 + u(\tanh(wx_1))) - x_3\right]$$

$$\cdot \left[1 - (\tanh(wx_2 + u(\tanh(wx_1))))^2\right]$$

$$\cdot \tanh(wx_1)$$

Now

$$\frac{\partial E}{\partial w} = \frac{1}{2} \cdot 2 (S_2 - x_3) \frac{\partial S_2}{\partial w}$$

$$= (S_2 - x_3)(1 - S_2^2)\left[x_2 + u \frac{\partial S_1}{\partial w}\right]$$

$$\frac{\partial E}{\partial w} = (S_2 - x_3)(1 - S_2^2)\left[x_2 + u(1 - S_1^2) x_1\right]$$

$$\frac{\partial E}{\partial w} = \left[\tanh(wx_2 + u(\tanh(wx_1))) - x_3\right] \cdot$$

$$\cdot \left[1 - (\tanh(wx_2 + u(\tanh(wx_1))))^2\right] \cdot$$

$$\left[x_2 + u\left(1 - (\tanh(wx_1))^2\right) x_1\right]$$

b) $E = \frac{1}{2}(S_K - T_{K+1})^2$

First let's find $\frac{\partial S_K}{\partial w}$ and $\frac{\partial S_K}{\partial u}$

$$\frac{\partial S_K}{\partial u} = (1 - S_K^2) \frac{\partial}{\partial u}[u S_{K-1}]$$

$$= (1 - S_K^2)\left[S_{K-1} + u \cdot \frac{\partial}{\partial u}(S_{K-1})\right]$$

$$= (1 - S_K^2)(S_{K-1}) + u(1 - S_K^2)\frac{\partial}{\partial u}(S_{K-1})$$

$$= (1 - S_K^2)(S_{K-1}) + u(1 - S_K^2)\left[(1 - S_{K-1}^2)(S_{K-2}) + u(1 - S_{K-1}^2)\frac{\partial}{\partial u}(S_{K-2})\right]$$

Generic Form:

$$\frac{\partial S_K}{\partial u} = \sum_{t=K}^{2} u^{K-t} S_{t-1} \prod_{j=t}^{K} (1 - S_j^2) \quad - (1)$$

$$\frac{\partial S_K}{\partial w} = (1-S_K^2)\left[x_K + u\frac{\partial}{\partial w}(S_{K-1})\right]$$

$$= (1-S_K^2)x_K + u(1-S_K^2)\left[(1-S_{K-1}^2)\left(x_{K-1} + u\frac{\partial}{\partial w}(S_{K-2})\right)\right]$$

$$= (1-S_K^2)x_K + u(1-S_K^2)(1-S_{K-1}^2)x_{K-1} + u^2(1-S_K^2)(1-S_{K-1}^2)\frac{\partial}{\partial w}(S_{K-2})$$

General Form

$$\frac{\partial S_K}{\partial w} = \sum_{t=K}^{1} u^{K-t} x_t \prod_{j=t}^{K}\left(1-S_j^2\right) \quad -(2)$$

Now

$$\Rightarrow \frac{\partial E}{\partial u} = \frac{1}{2} \cdot 2 \left(S_K - a_{K+1}\right)^{2} \frac{\partial}{\partial u}\left(S_K\right)$$

$$= \left(S_K - a_{K+1}\right) \frac{\partial}{\partial u}\left(S_K\right)$$

From ①

$$\Rightarrow \frac{\partial E}{\partial u} = \left(S_K - a_{K+1}\right)\left[\sum_{t=K}^{2} u^{K-t} S_{t-1} \prod_{j=t}^{K}\left(1 - S_j^{\cdot 2}\right)\right]$$

$$\Rightarrow \frac{\partial E}{\partial w} = \frac{1}{2} \cdot 2 \left(S_K - a_{K+1}\right) \cdot \frac{\partial S_K}{\partial w}$$

From ②

$$\frac{\partial E}{\partial w} = \left(S_K - a_{K+1}\right)\left[\sum_{t=K}^{1} u^{K-t} a_t \prod_{j=t}^{K}\left(1 - S_j^{\cdot 2}\right)\right]$$

In [1]:
```
import numpy as np
import pandas as pd
from math import *
import matplotlib.pyplot as plt
```

In [2]:
```
k = 5
z =[]
for i in range(0,k+200):
    z.append(sin(cos(i/10)+2*cos(i/5)))
```

In [3]:
```
x =[]
y =[]
t =[]
for i in range(0, k+195):
    p =[]
    for j in range(i,i+5):
        p.append(z[j])
    y.append(z[j+1])
    t.append(j+1)
    x.append(p)
```

In [4]:
```
z[0:6]
```

Out[4]:
```
[0.1411200080598672,
 0.18537683742241873,
 0.3140008437758883,
 0.5103441531044438,
 0.7359834803472819,
 0.9258978802988732]
```

In [5]:
```
x[0]
```

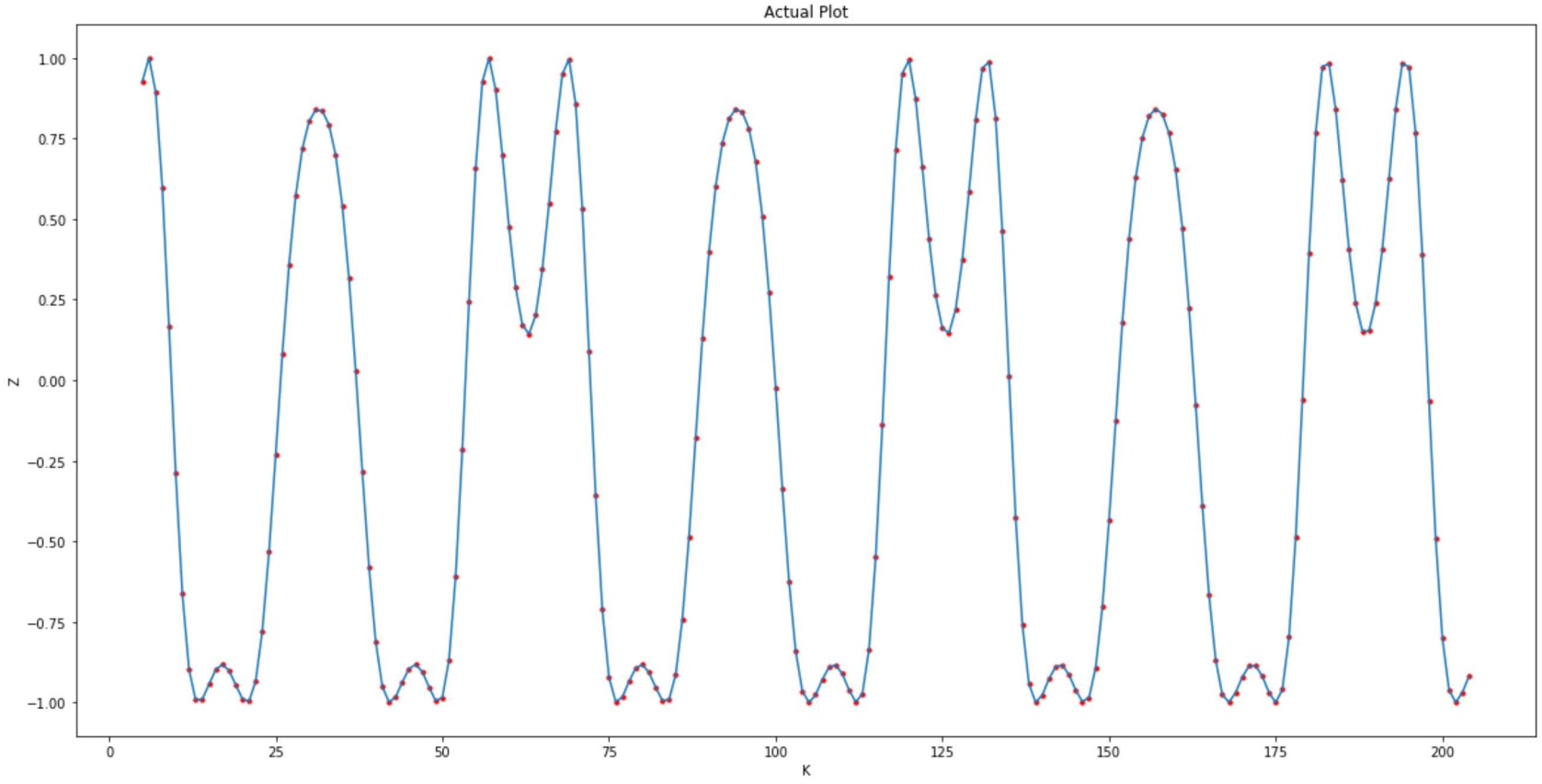
Out[5]:
```
[0.1411200080598672,
 0.18537683742241873,
 0.3140008437758883,
 0.5103441531044438,
 0.7359834803472819]
```

In [6]:
```
y[0]
```

Out[6]:
```
0.9258978802988732
```

In [23]:
```
plt.figure(figsize = (20,10))
plt.scatter(t,y, c='r', s=10)
plt.xlabel("K")
plt.ylabel("Z")
plt.title("Actual Plot")
plt.plot(t,y)
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x285023b7d88>]
```



In [8]:
```
features = pd.DataFrame(data= np.asarray(x))
```

In [9]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       200 non-null    float64
 1   1       200 non-null    float64
 2   2       200 non-null    float64
 3   3       200 non-null    float64
 4   4       200 non-null    float64
dtypes: float64(5)
memory usage: 7.9 KB
```

In [10]:
```
#The commented variables are suggestions so change them as appropriate,
#However, do not change the __init__(), train(), or predict(x=[]) function headers
#You may create additional functions as you see fit

import numpy as np
np.random.seed(100)


class Neural_Network():

    def __init__(self, x=[[]], y=[], nHiddenLayers = 0, nHiddenNodes =0, numOutputs = 0, eta = 1, iter = 0, p
rec = 0):
        newdata = []
        for i in range(len(x)):
            newdata.append(np.append(x[i],1))
        newdata = np.asarray(newdata)
        self.data = newdata
#         self.data = np.asarray(x)
        self.labels = y
        self.nInputNodes = self.data.shape[1]
        self.nHiddenLayers = nHiddenLayers
        self.nHiddenNodes = nHiddenNodes
        self.numOutputs = numOutputs
        self.eta = eta
        self.maxIt = iter
        self.prec = prec
        if self.numOutputs > 1:
             self.weights=[np.random.uniform(low = -2, high = 2, size = (self.nHiddenNodes, self.nInputNodes+
1))]
        else:
            self.weights=[np.random.uniform(low = -2, high = 2, size = (1, self.nInputNodes+1))]
        for i in range(self.nHiddenLayers-1):
            self.weights.append(np.random.uniform(low =-2, high = 2, size =(self.nHiddenNodes, self.nHiddenNo
des+1)))
        if self.numOutputs > 1:
             self.weights.append(np.random.uniform(low =-2, high = 2, size = (self.numOutputs, self.nHiddenNo
des+1)))

    # Tanh activation function for all layers except the output layer
    def tanh(self, s):
        return tanh(s)
```

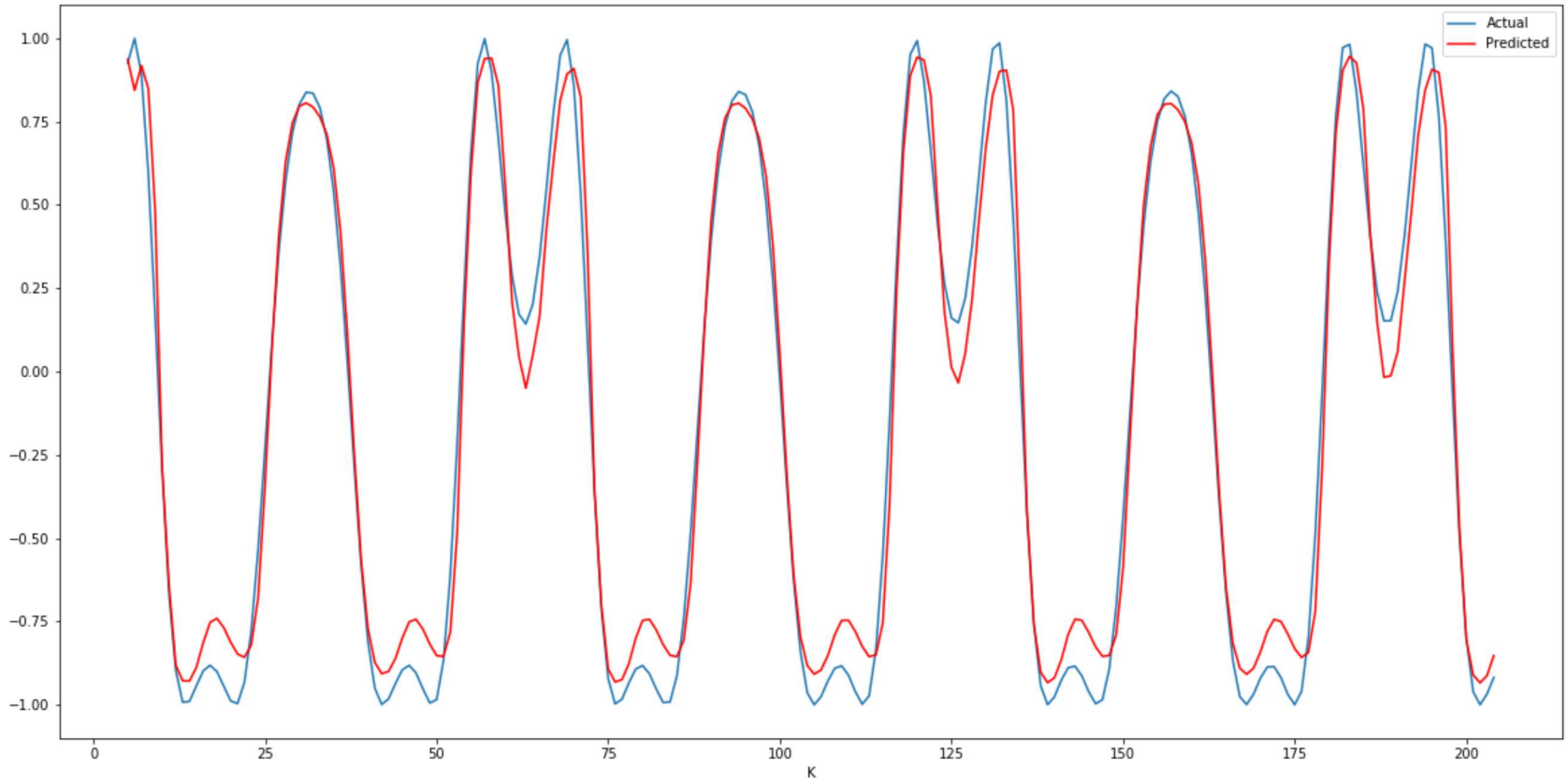
```
    # Derivative of tanh for all layers
    def tanhPrime(self, s):
        return (1 - s**2)

    # Activation function for all the input data
    def af_predict(self,t):
        t = tanh(t)
        return t

    # Activation function for the output layer of the feedforward graph
    def af(self,t):
        t = tanh(t)
        return t

    def predict(self, data=[]):
        seed = 0
        o =[]
        for i in range(len(data)):
#             print(data[i].shape)
#             print(np.asarray(self.weights)[0][self.nHiddenLayers][0:self.nInputNodes].shape)
#             print(f)
            temp_f = np.matmul(np.asarray(self.weights)[0][self.nHiddenLayers][0:self.nInputNodes], data[i].r
eshape(self.nInputNodes,1))
#             print(temp_f)
#             print(f)
            temp4 = self.af_predict(temp_f[0]+seed*np.asarray(self.weights)[0][self.nHiddenLayers][self.nInpu
tNodes])
            seed = temp4
            o.append(temp4)
        return o

    def feedforward(self, x, seed):
        self.r = np.append(x,seed)
#         print(self.r)
#         print(f)
        temp_f = np.matmul(np.asarray(self.weights)[0][self.nHiddenLayers][0:self.nInputNodes], x.T)
        temp4 =self.af(temp_f.T + seed*np.asarray(self.weights)[0][self.nHiddenLayers][self.nInputNodes])
        return temp4

    def backward(self, x, y, seed):
        self.gradient=[]
        self.o = self.feedforward(x, seed)
#         print(self.o)
```

```
#         print(f)
        self.out_error = y - self.o
        temp = self.out_error*self.tanhPrime(self.o)
#         print(self.r)
#         print(f)
        temp2 = ((self.r*temp)*2)/len(self.data)
        self.gradient.append(self.eta*temp2)
#         print(self.gradient)
#         print(self.weights[0])
#         print(self.weights[0]+self.gradient[0])
#         print(f)

        for i in range(len(self.weights)):
            self.weights[i][0] += self.gradient[i]

    def train (self):
        e = 0
        obj_training =[]
        epoch = []
        epoch.append(e)
        pred_train = self.predict(self.data)
        obj_training.append(((np.linalg.norm(self.labels - pred_train))**2)/len(self.data))
        mse = 100000000000
#         while e <= self.maxIt and mse >= self.prec:
        while e <= self.maxIt:
            seed = 0
            prev = mse
            e += 1
            for i in range(len(self.data)):
                self.backward(self.data[i].reshape(1,self.nInputNodes), self.labels[i], seed)
                seed = self.o
            pred_train = self.predict(self.data)
            mse = ((np.linalg.norm(self.labels - pred_train))**2)/len(self.data)
            obj_training.append(mse)
            epoch.append(e)
            if mse >= prev:
                self.eta = 0.9*self.eta
            prev = mse
        return self.weights, epoch, obj_training, self.predict(self.data)
```

In [11]:
```
iter = 100
prec =0.5
NN = Neural_Network(np.asarray(x), np.asarray(y), 0, 0, 1, 1, iter, prec)
w, epoch, obj_training, pred = NN.train()
```

In [22]:
```
plt.figure(figsize = (20,10))
# plt.scatter(t,y, c='r', s=10)
plt.plot(t,y, label = 'Actual')
plt.plot(t,pred, c ='r', label = 'Predicted')
plt.xlabel("K")
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x28503a12c08>



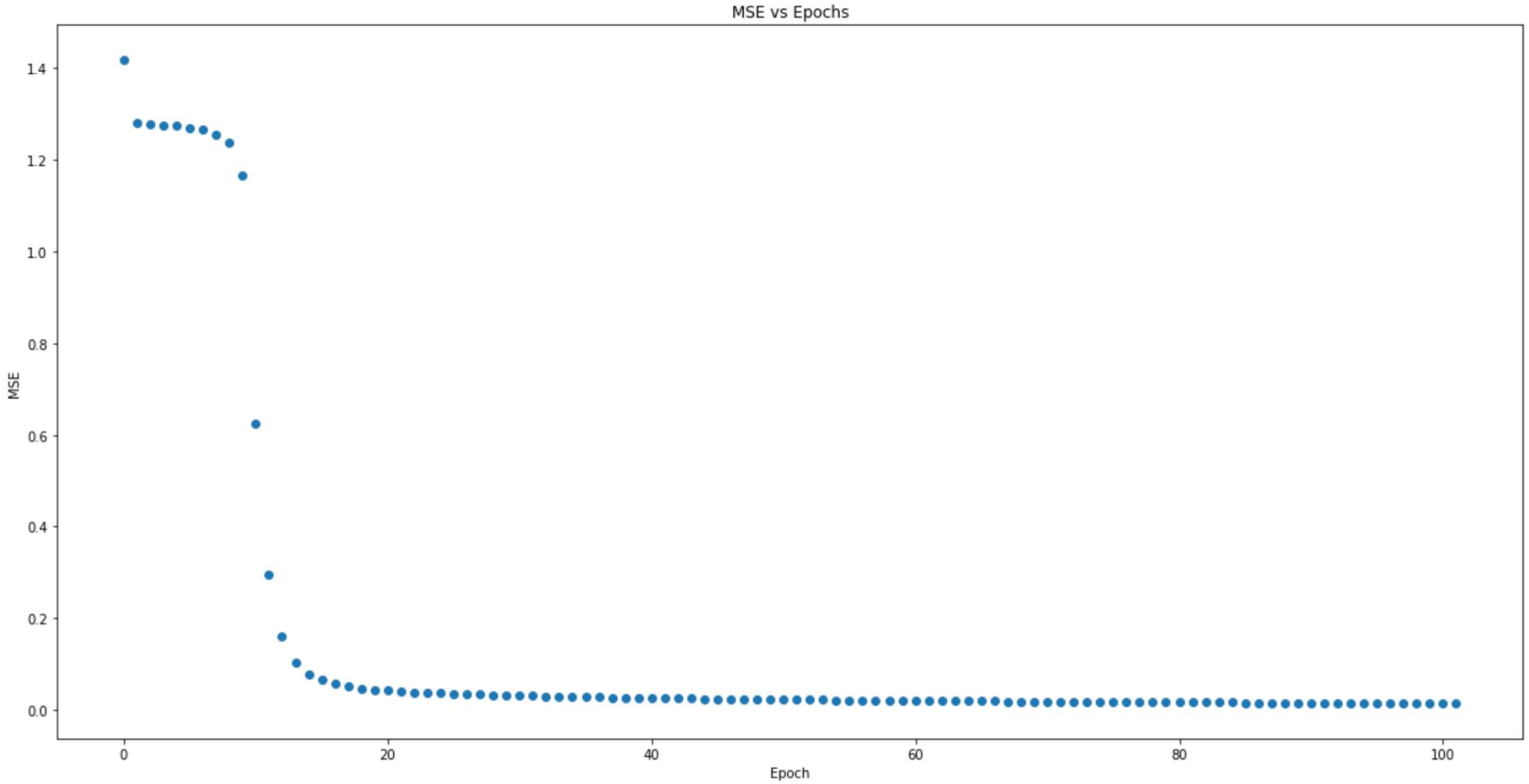
In [13]:
```
mse = []
for i in range(len(y)):
    mse.append(((y[i] - pred[i])**2)/2)
```

In [14]:
```
temp =[]
for i in range(k+1, k+201):
    temp.append(i)
```

Epoch Basis MSE Plot

In [21]:
```
plt.figure(figsize = (20,10))
plt.scatter(epoch,obj_training)
plt.xlabel("Epoch")
plt.ylabel("MSE")
plt.title("MSE vs Epochs")
```

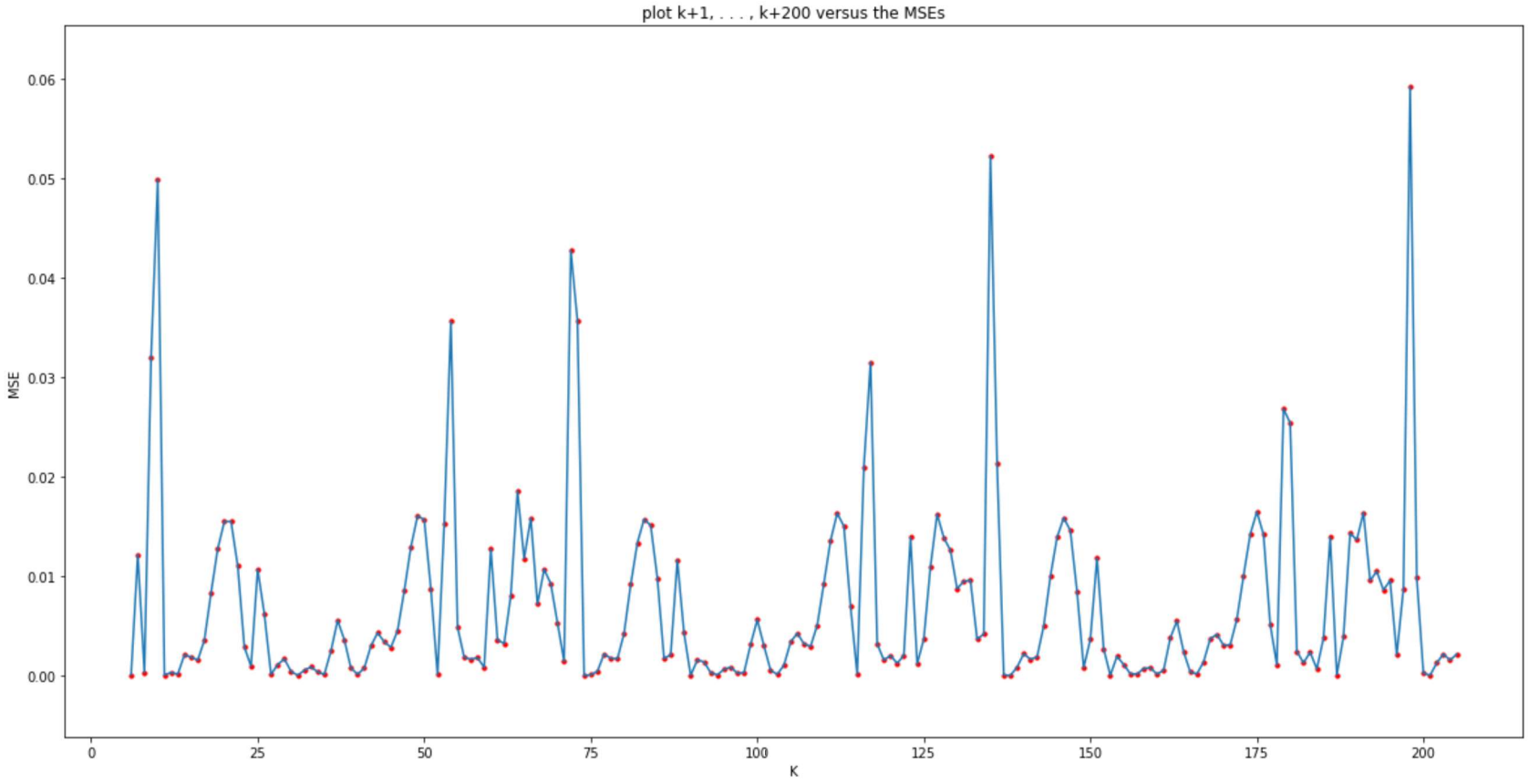
Out[21]: Text(0.5, 1.0, 'MSE vs Epochs')



Time Basis MSE Plot

In [20]:
```
plt.figure(figsize = (20,10))
plt.scatter(temp,mse, s=10, c='r')
plt.plot(temp,mse)
plt.xlabel('K')
plt.ylabel("MSE")
plt.title("plot k+1, . . . , k+200 versus the MSEs")
```

Out[20]: Text(0.5, 1.0, 'plot k+1, . . . , k+200 versus the MSEs')



Comments

On the predictions by the Neural Network

The predicted and actual curves are almost similar except at few points where the actual curve has troughs.

On the Tlme basis K vs MSE plot

I expected the time basis plot to be irregular because here the mse is calculated at each point rather than summing the mse's of all the points at the end

On the Epoch basis epoch vs MSE plot

I expected the epoch basis MSE plot should have a decreasing curve (not necessarily gradually decreasing) if the algorithm is correct. My plot has same nature which implies that my neural network code is correct.

In [ ]:

3) a) Given

Input size = 20×20

~~as No bias is there~~ No biases are there.

First Conv layer = 8 filters of size 3×3 each

So, No of parameters = 8×3×3 = 72.

There are no free parameters for pooling layer

Now,

Layer	o/p shape	parameters
Input	20×20	0
conV1 - 8×3×3	18×18×8	72.
Pool 2×2 ~~8=2~~	9×9×8	0
FC (10)	10×1	9×9×8×10 = 6480
		Total:- 6552

Total free parameters

= 6552

b) Even though we have Linear activation function in all layers, since we have max pooling Layer which is Non-Linear

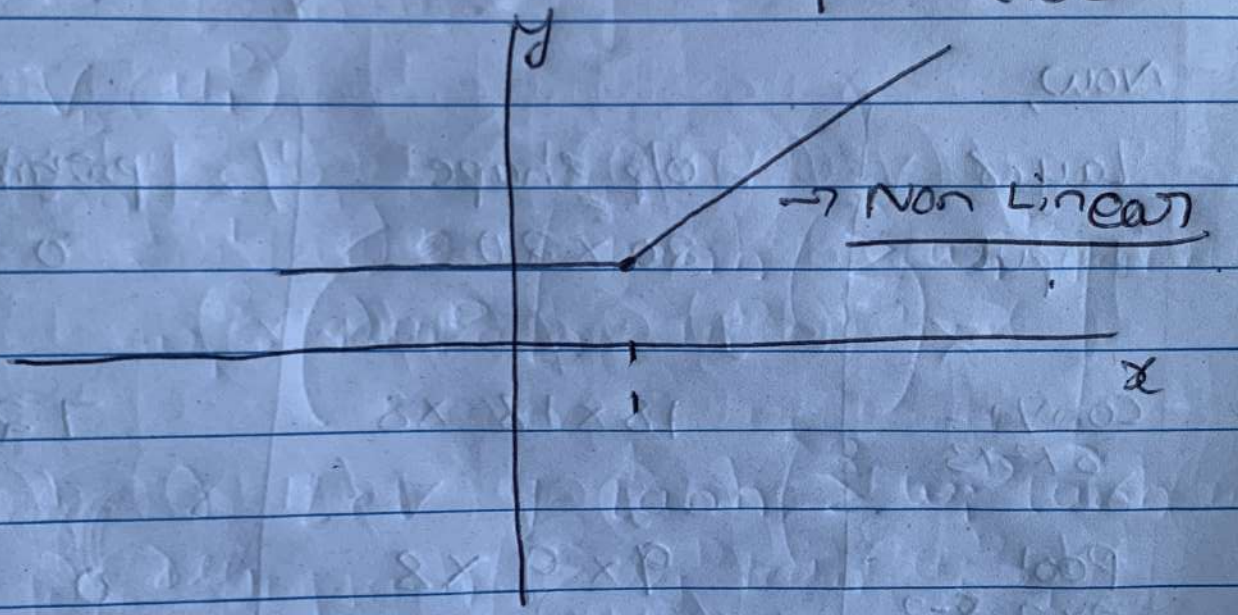
Ex:

The max() function is Non-Linear

consider

$y = \max(x, 1) = x$ when $x \geq 1$

else 1



So,

the output won't be a Linear function of Inputs

c) Consider a CNN with the given Max pooling layer as the first Layer and FC as second

Since, Max pooling is Non-Linear, the output won't be a Linear combination of Inputs.

d) Consider a CNN with the Convolutional layer and FC layer

Convolution is a Linear operation. So, since we have Linear activation functions in all the layers, output would be a Linear Combination

4) a) Given.

$$G(z) = \tanh\left(\sum_{i=1}^{2} w_i^G \tanh\left(u_i^G z\right)\right)$$

$$D(x) = \frac{1}{2}\left(1 + \tanh\left(\sum_{i=1}^{2} w_i^D \tanh\left(u_i^D x\right)\right)\right)$$

a) $\nabla_D(u_1^D)$

$$= \frac{1}{2} \frac{\partial}{\partial u_1^D}\left[\tanh\left(\sum_{i=1}^{2} w_i^D \tanh\left(u_i^D x\right)\right)\right]$$

$$= \frac{1}{2} \frac{\partial}{\partial u_1^D}$$ Let $y = \tanh\left(\sum_{i=1}^{2} w_i^D \tanh\left(u_i^D x\right)\right)$

Then, as $\left(\tanh(x)\right)' = 1 - \left(\tanh(x)\right)^2$.

$$\nabla_D(u_1^D) = \frac{1}{2}\left(1 - y^2\right) \frac{\partial}{\partial u_1^D}\left(\sum_{i=1}^{2} w_i^D \tanh\left(u_i^D x\right)\right)$$

~~Again assume~~

$$\Rightarrow \frac{1}{2}(1-y^2)\frac{\partial}{\partial u_1^D}\left(w_1^D \tanh(u_1^D x) + w_2^D \tanh(u_2^D x)\right)$$

$$\Rightarrow \frac{1}{2}(1-y^2)\, w_1^D \frac{\partial}{\partial u_1^D}\left(\tanh(u_1^D x)\right)$$

Let $p = \tanh(u_1^D x)$

$$\nabla_D(u_1^D) = \frac{1}{2}(1-y^2) w_1^D (1-p^2)\, x.$$

So, update equation of discriminator to update u_1^D is

$$\left[\frac{1}{2}\left(1 - \left(\tanh\left(\sum_{i=1}^{2} w_i^D \tanh(u_i^D x)\right)\right)^2\right) \cdot w_1^D \cdot \left(1 - (\tanh(u_1^D x))^2\right) \cdot x\right]$$

$$\frac{1}{m}\sum_{i=1}^{m}\left(\log\left(D(x_i)\right) + \log\left(1 - D(G(z_i))\right)\right)$$

where

x_i, $i = 1, \ldots m$ are input data

z_i, $i = 1, \ldots m$ are noise to Generator

$$\left[\frac{1}{2}\left(1-\left(\tanh\left(\sum_{i=1}^{2} w_i^D \tanh(u_i^D x)\right)\right)^2\right)\right.$$

$$\left. \cdot w_1^D \cdot \left(1-(\tanh(u_1^D x))^2\right) \cdot x\right]$$

$$\cdot \frac{1}{m}\sum_{i=1}^{m}\left[\log\left(D(x_i)\right) + \log\left(1-D\left(\tanh\left(\sum_{j=1}^{2} w_j^G \tanh(u_j^G z_i)\right)\right)\right)\right]$$

$$=$$

$$\left[\frac{1}{2}\left(1-\left(\tanh\left(\sum_{i=1}^{2} w_i^D \tanh(u_i^D x)\right)\right)^2\right)\right.$$

$$\left. \cdot w_1^D \cdot \left(1-(\tanh(-u_1^D x))^2\right) \cdot x\right]$$

$$\cdot \frac{1}{m}\sum_{i=1}^{m}\left[\log\left(\frac{1}{2}\left(1+\tanh\left(\sum_{j=1}^{2} w_j^D \tanh(u_j^D x_i)\right)\right)\right)\right.$$

$$\left. + \log\left(1-\frac{1}{2}\left(1+\tanh\left(\sum_{k=1}^{2} w_k^D \cdot \tanh\left(u_k^D \cdot \tanh\left(\sum_{l=1}^{2} w_l^G \tanh(u_l^G z_i)\right)\right)\right)\right)\right)\right]$$

b) $G(z) = \tanh\left(\sum_{i=1}^{2} w_i^G \tanh(u_i^G z)\right)$

$$\nabla_{w_2^G} = \frac{\partial G(z)}{\partial w_2^G}$$

$$\nabla_{w_2^G} = \frac{\partial}{\partial w_2^G}\left[\tanh\left(\sum_{i=1}^{2} w_i^G \tanh(u_i^G z)\right)\right]$$

$$= \left(1 - (G(z))^2\right) \frac{\partial}{\partial w_2^G}\left[\sum_{i=1}^{2} w_i^G \tanh(u_i^G z)\right]$$

$$= \left(1 - (G(z))^2\right) \tanh(u_2^G z)$$

Update equation of Generator to update w_2^G is

$$\left[\left(1 - (G(z))^2\right) \tanh(u_2^G z)\right] \times \frac{1}{m} \sum_{i=1}^{m} \left[\log(1 - D(G(z_i))\right]$$

$$\left[\left(1-(G_1(z))^2\right)\tanh\left(u_2^{G}z\right)\right]$$

$$\cdot\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(\tanh\left(\sum_{j=1}^{2}w_i^{G}\tanh\left(u_j^{G}z_i\right)\right)\right)\right)$$

$$=$$

$$\left[\left(1-(G_1(z))^2\right)\cdot\tanh\left(u_2^{G}z\right)\right]$$

$$\cdot\frac{1}{m}\sum_{i=1}^{m}\left[\log\left(1-\frac{1}{2}\left(1+\tanh\left(\sum_{j=1}^{2}w_j^{D}\tanh\left(u_j^{D}\cdot\tanh\left(\sum_{K=1}^{2}w_K^{G}\tanh\left(u_K^{G}z_i\right)\right)\right)\right)\right)\right)\right]$$

$$= \left[\left[1 - \left(\tanh \left(\sum_{k=1}^{2} w_k^G \tanh(u_k^G z) \right) \right)^2 \right] \cdot \tanh\left(u_2^G z\right) \right]$$

$$\cdot \frac{1}{m} \sum_{i=1}^{m} \left[\log \left(1 - \frac{1}{2} \left(1 + \tanh \left(\sum_{j=1}^{2} w_j^D \tanh \left(u_j^D \cdot \tanh \left(\sum_{l=1}^{2} w_l^G \tanh(u_l^G z_i) \right) \right) \right) \right) \right) \right]$$