Question 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Id | Y | D0 | hx1(x1>6) | Hx2(x2>6.5) | D1 | hx1(x1>8.5) | hx2(x2>9) |
| 1 | - | 1/10 | - | - | 1/4e | - | - |
| 2 | - | 1/10 | - | + | 1/4e | - | + |
| 3 | - | 1/10 | - | - | 1/4e | - | - |
| 4 | - | 1/10 | + | + | e/4 | - | - |
| 5 | - | 1/10 | - | - | 1/4e | - | - |
| 6 | - | 1/10 | - | - | 1/4e | - | - |
| 7 | + | 1/10 | + | + | 1/4e | - | - |
| 8 | + | 1/10 | - | + | e/4 | - | + |
| 9 | + | 1/10 | + | + | 1/4e | + | + |
| 10 | + | 1/10 | + | + | 1/4e | - | - |

For round 1:

error= 2/10=0.2

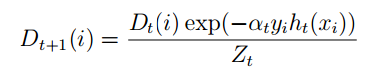
alpha= ½ln(1- error / error)

alpha= ½ln(1-0.2/0.2)

alpha= 1

Z1 = sqrt(error(1-error))

Z1=4/10



D(1,1) =1/10\*e^(-1\*-1\*-1)/(4/10)=1/4e

We can do similarly for D(1,i) where 1<= i <=10

For round 2:

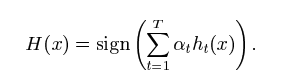
error= 3/10=0.3

alpha= ½ln(1- error / error)

alpha= ½ln(1-0.3/0.3)

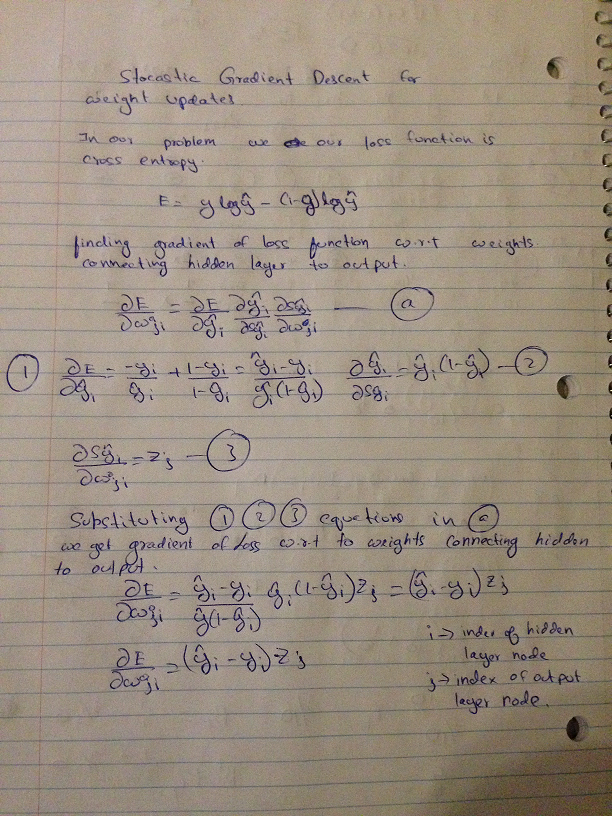
alpha= .6115

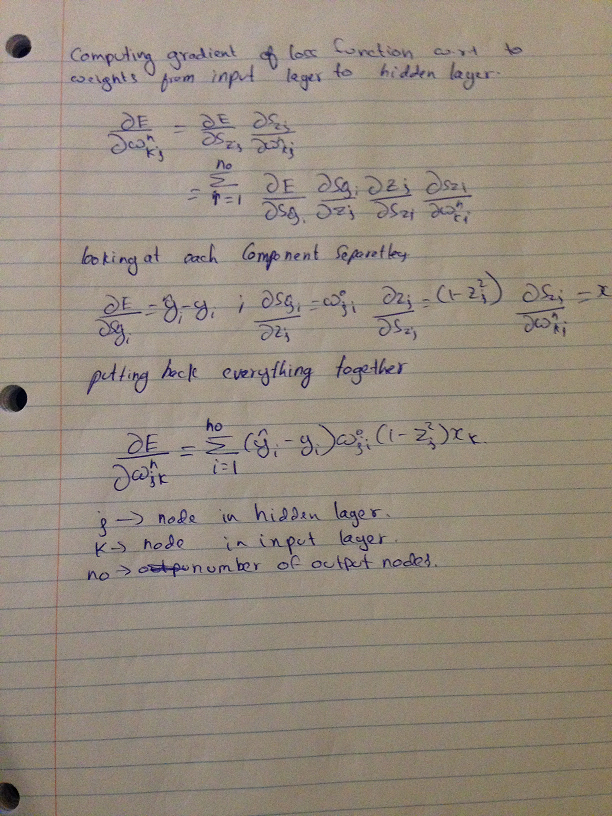
Final Hypothesis is based on



|  |  |
| --- | --- |
| Id | Y |
| 1 | - |
| 2 | - |
| 3 | - |
| 4 | + |
| 5 | - |
| 6 | - |
| 7 | + |
| 8 | - |
| 9 | + |
| 10 | + |

Question 2:





b.

|  |  |  |  |
| --- | --- | --- | --- |
| Learning Rate | Weight initialization | Halting criteria | Validation Accuracy |
| 0.0000063 | Gaussian distribution | 1000 epoch | 88.56 |
| 0.000004 | Gaussian distribution | 1000epoch | 86.85 |
| 0.00000006 | Gaussian distribution | 1000 epoch | 83.78 |
| 0.0000063 | Random initialization | 1000epoch | 10.53 |

The way I selected hyper parameters for my network is by selecting the combination of parameters that gave me the best validation accuracy. So from the above table I have decided to choose learning rate of 0.0000063, initialize weights from a Gaussian distribution and 1000 epoch of training

Plot of training vs validation

From the graph we can say that network is improving the performance as we increase the number of epochs. This plot is for 500 epochs.

The testing accuracy I get is 86.8.

Summary of Findings:

1. Identifying correct hyper parameters is crucial for neural network performance.
2. Network weights have to be initialized from a Gaussian distribution
3. If we initialize network weights randomly the network for some reason always guesses same label
4. Matrix operations have using batch learning the program runs very fast
5. Online learning for neural network coded on python is very very slow.
6. For some reason my network seems to converge at validation accuracy of 87%-89% and train accuracy of 91% and there is no improvement of accuracy beyond this point
7. I guess there is something wrong with how I am initializing my weights. I feel so because I think the gradient descent algorithm is finding a local minima and not global minima.