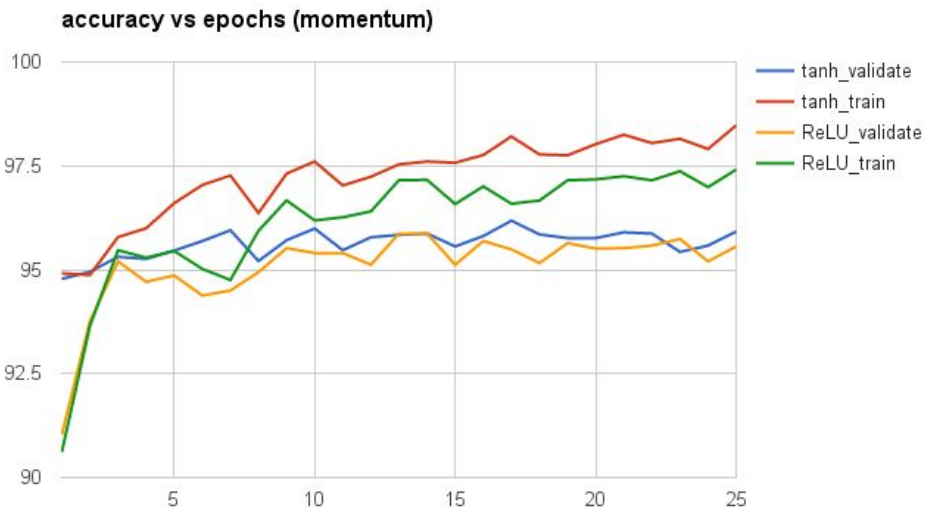
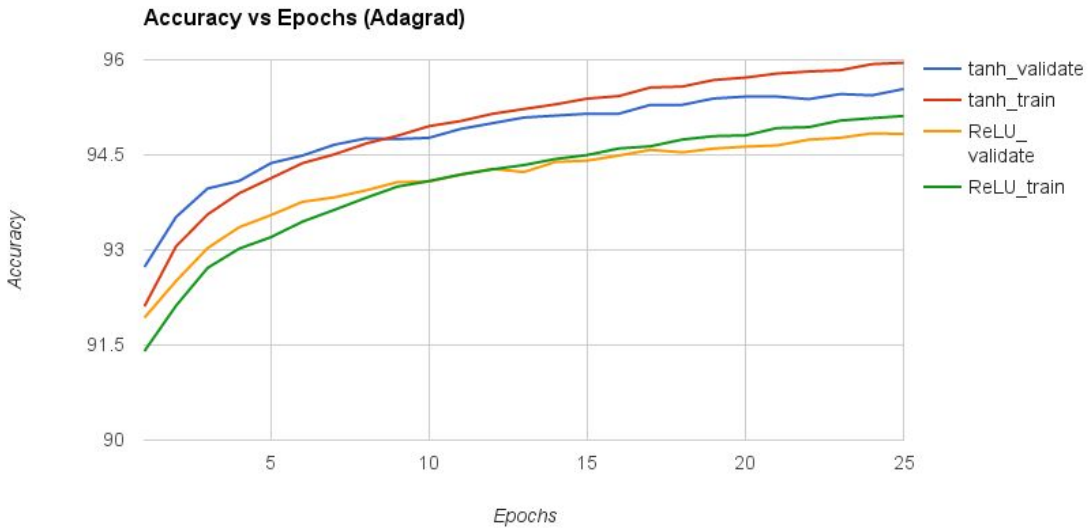
ABHINAV AGRWAL

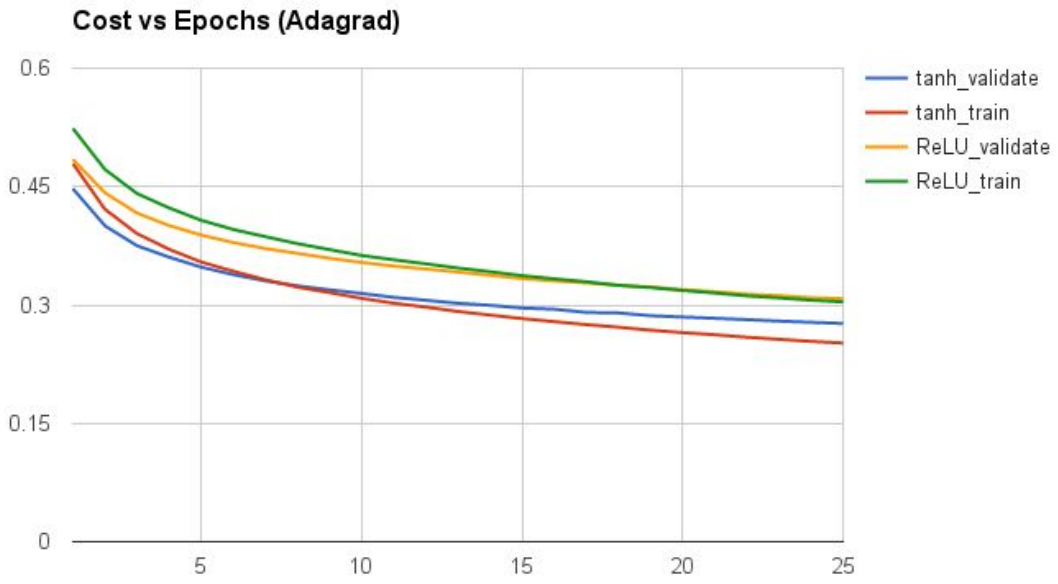
ROLL NO. : 14011

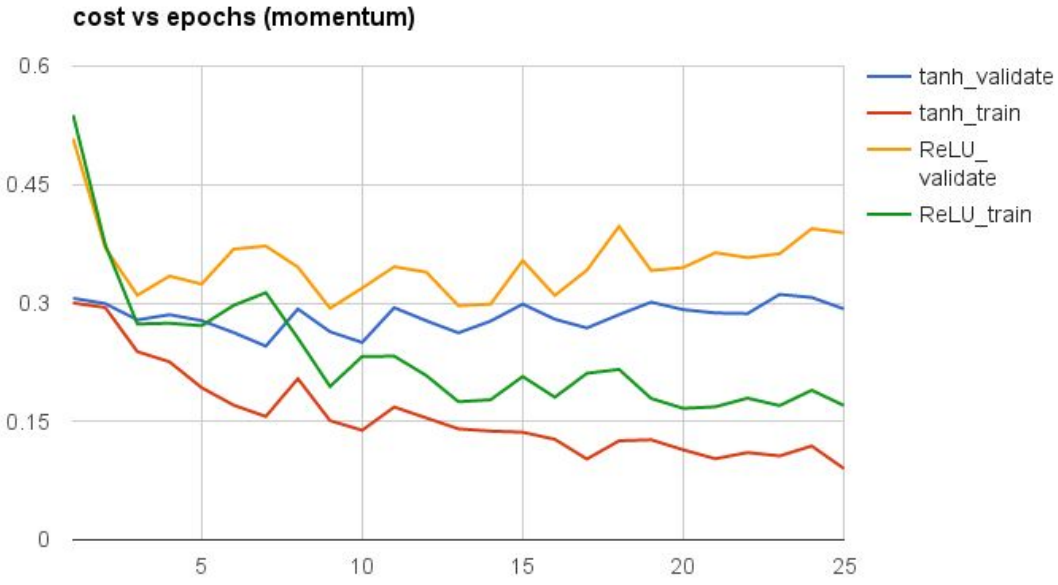
CS698A - ASSIGNMENT 1

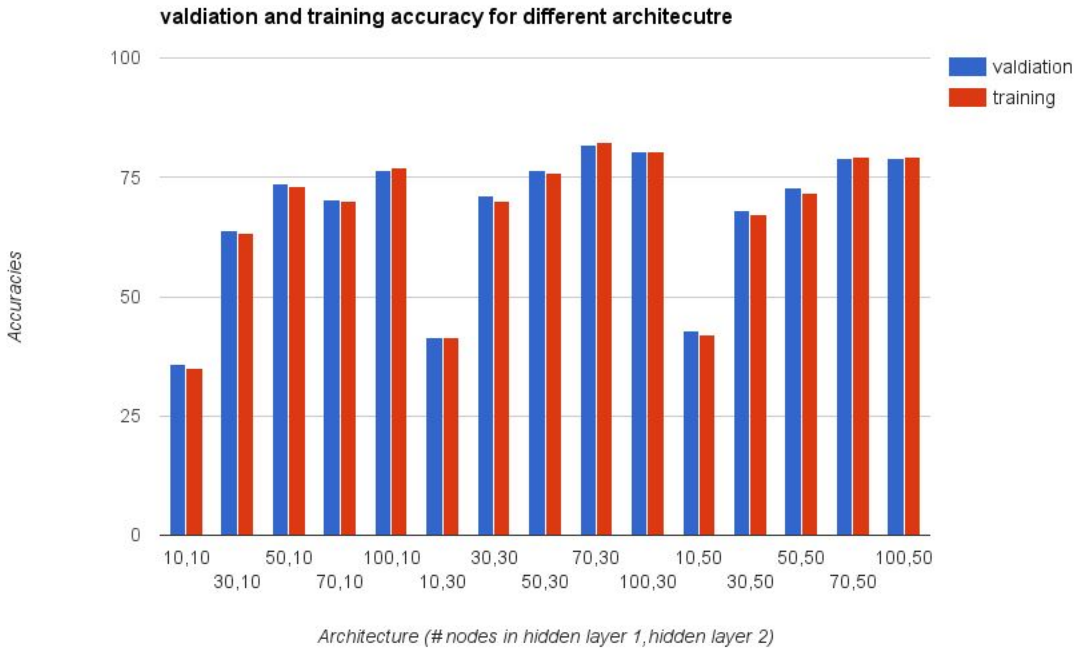
PLOTS:

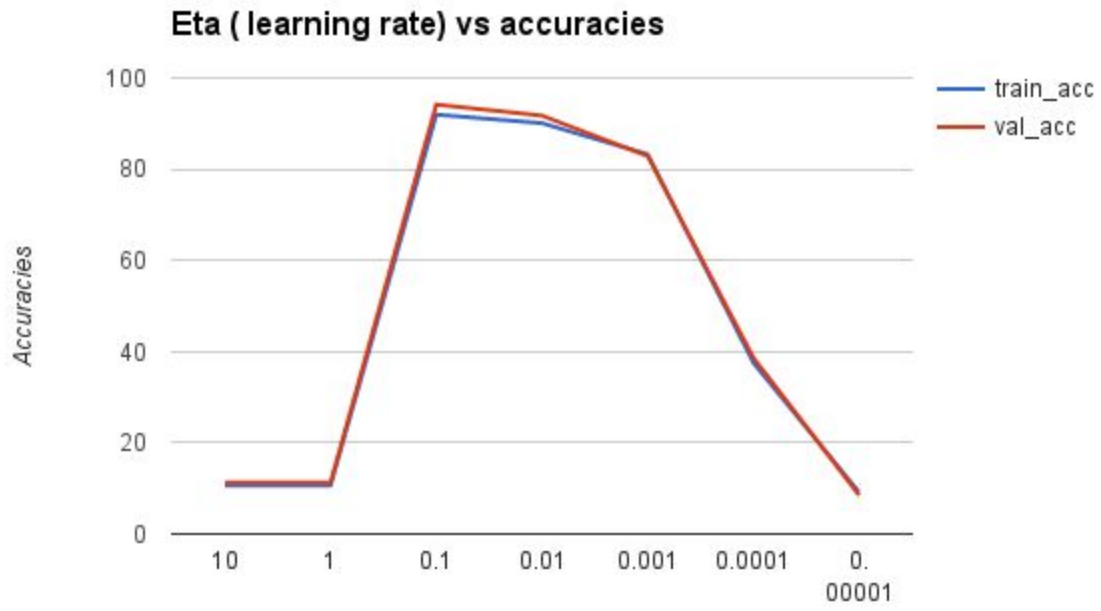





Cost vs Epochs (Adagrad)
0.6
0.45
0.3
0.15
0
5
10
15
20
25
tanh_validate
tanh_train
ReLU_validate
ReLU_train


cost vs epochs (momentum)
0.6
0.45
0.3
0.15
0
5
10
15
20
25
tanh_validate
tanh_train
ReLU_
validate
ReLU_train


valdiation and training accuracy for different architecutre
100
75
50
25
0
Accuracies
valdiation
training
10,10
30,10
50,10
70,10
100,10
10,30
30,30
50,30
70,30
100,30
10,50
30,50
50,50
70,50
100,50
Architecture (# nodes in hidden layer 1, hidden layer 2)


Eta (learning rate) vs accuracies
100
80
60
40
20
0
Accuracies
train_acc
val_acc
10
1
0.1
0.01
0.001
0.0001
0.
00001

FINDINGS:

The plots reveal that :

- Best choice for activation function is tanh (out of ReLU and itself) and the best way to update your gradients is momentum (out of momentum and adagrad), as it achieves higher accuracies with less number of epochs.
- The eta(learning rate) when varied over the a range represents a typical gaussian curve and the corresponding accuracy reaches maximum when eta is of the order of 0.1
  The eta=0.1 and the activation function=ReLU and the gradient = momentum for these findings. The network was trained with 10,000 examples and maximum accuracy out three epochs was used to come with data that is sufficient for intuitive results

- The study of different architecture validates the intuitive findings that the simple architecture (#no. of hidden nodes (10,10),(10,30) and so on) have limited abilities while the the architecture with sufficient number of parameters can learn with less number of epochs.
  The eta=0.1 and the activation function=tanh and the gradient = momentum for these findings. The network was trained with 10,000 examples and maximum accuracy out two epochs was used to come with data that is sufficient for intuitive results

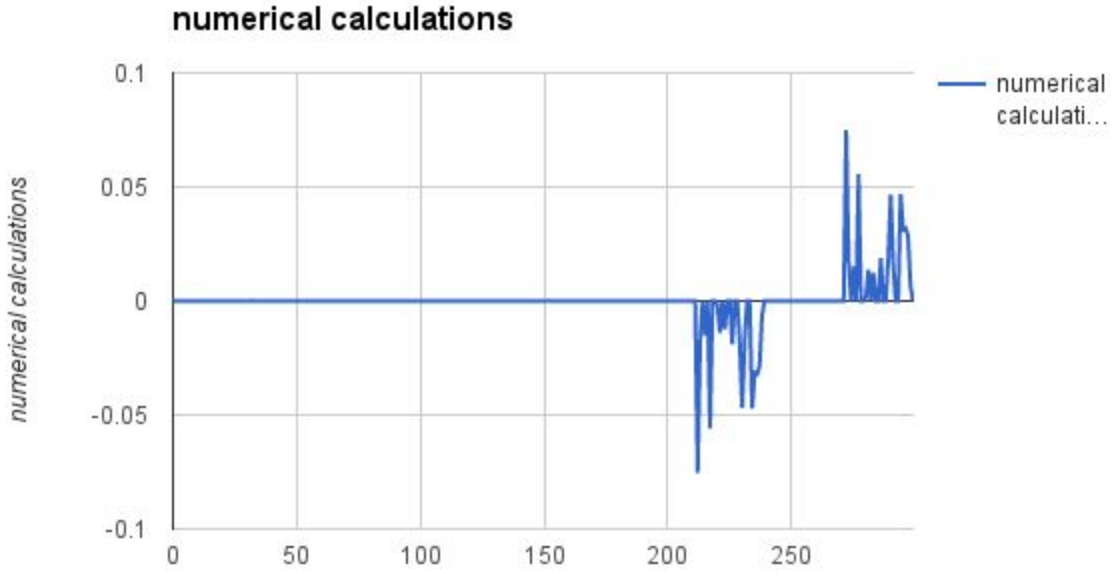
CODE:

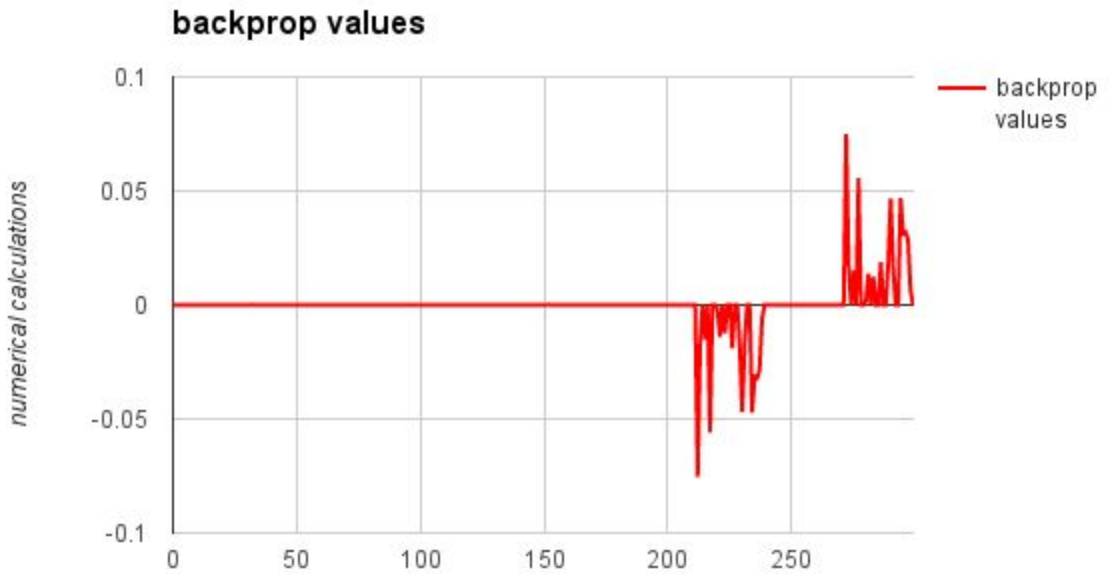
The code files are attached with the submission and a jupyter notebook is also attached to show how the play with the code.

GRADIENT CHECK:

The following graph shows the numerical and back-propagated gradients for a particular weight matrix and randomly chosen example. This was done after training for two epochs, using 10000 examples.

The function grad_check() was used for this and the details of implementation are in the jupyter notebook.


numerical calculations
0.1
0.05
0
-0.05
-0.1
0
50
100
150
200
250
numerical calculations
numerical calculati...


backprop values
0.1
0.05
0
-0.05
-0.1
0
50
100
150
200
250
numerical calculations
backprop values