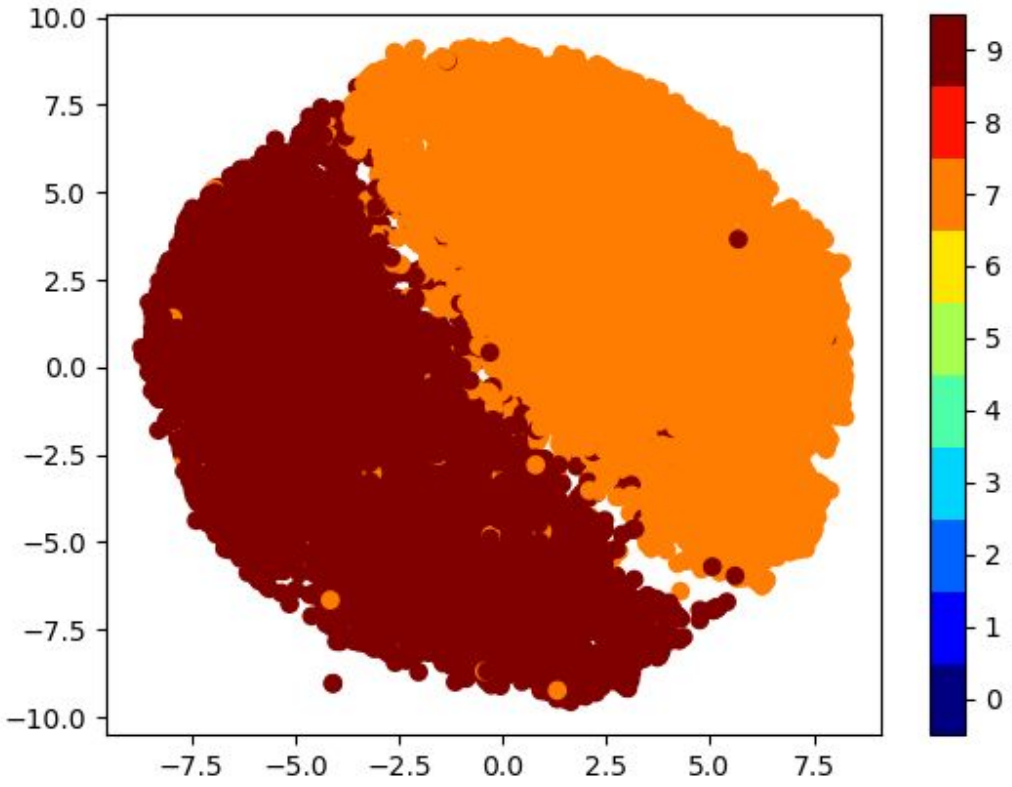
ML Homework 3 Report
Neha Jhamb
MT16037

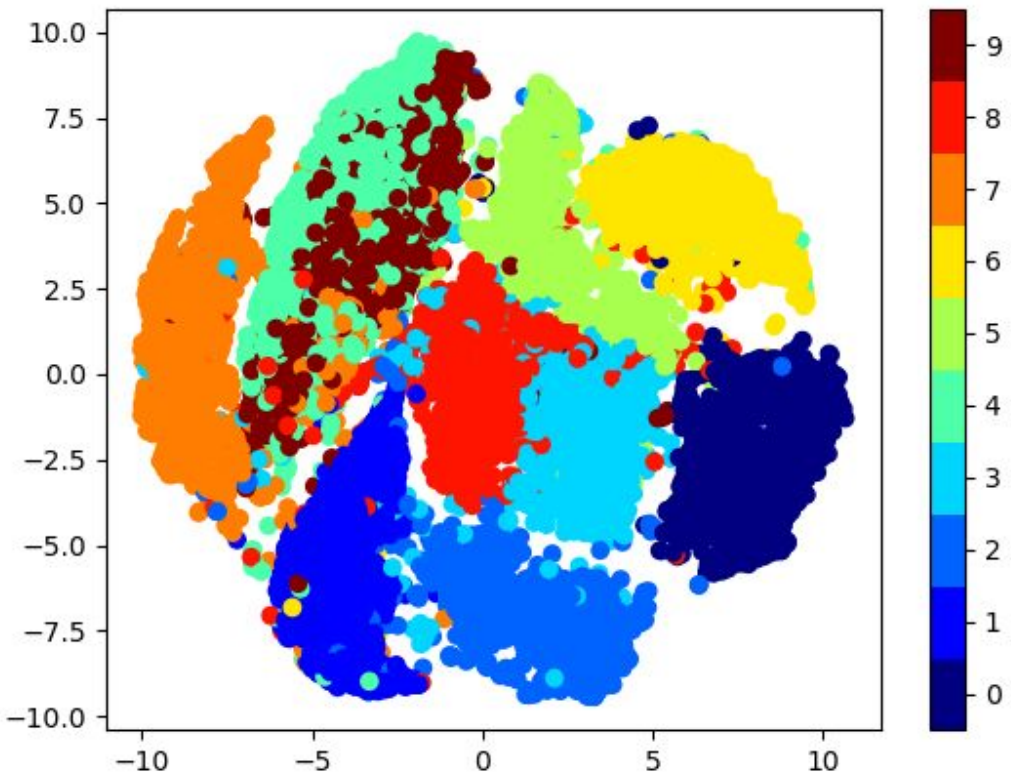
Visualization of smaller dataset:

The subset contains two classes only: 7 and 9. Therefor sigmoid activation suffices on outermost layer.



Visualization of MNIST dataset:

It contains 10 classes: 0 to 9. Therefore softmax activation is required on outermost layer to convert the values into probabilities. Note: I have taken a subset of size 12000 to for below plot.



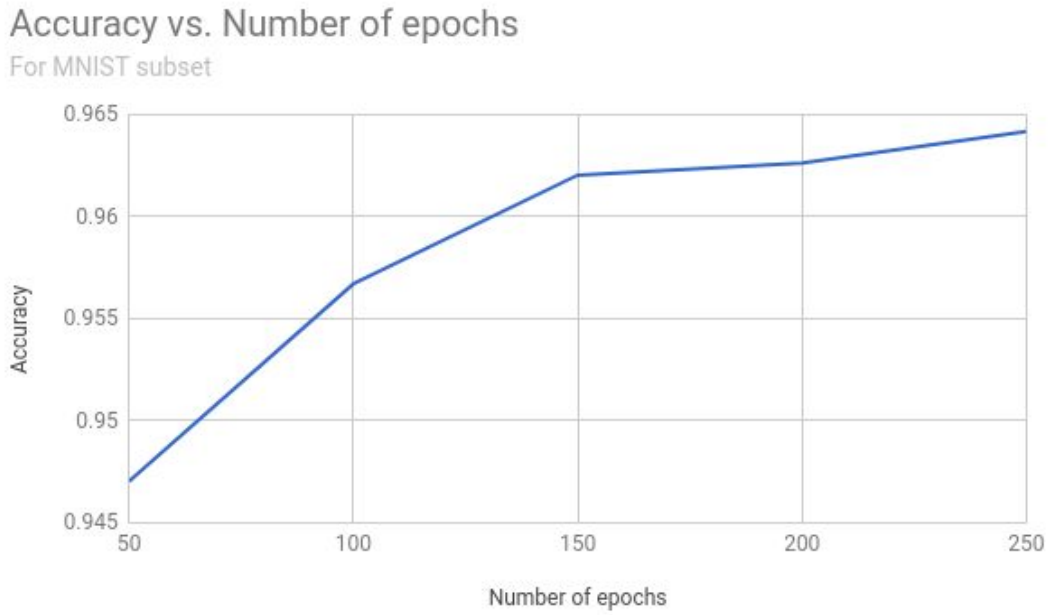
QUESTION 1

(a) Sigmoid activation:

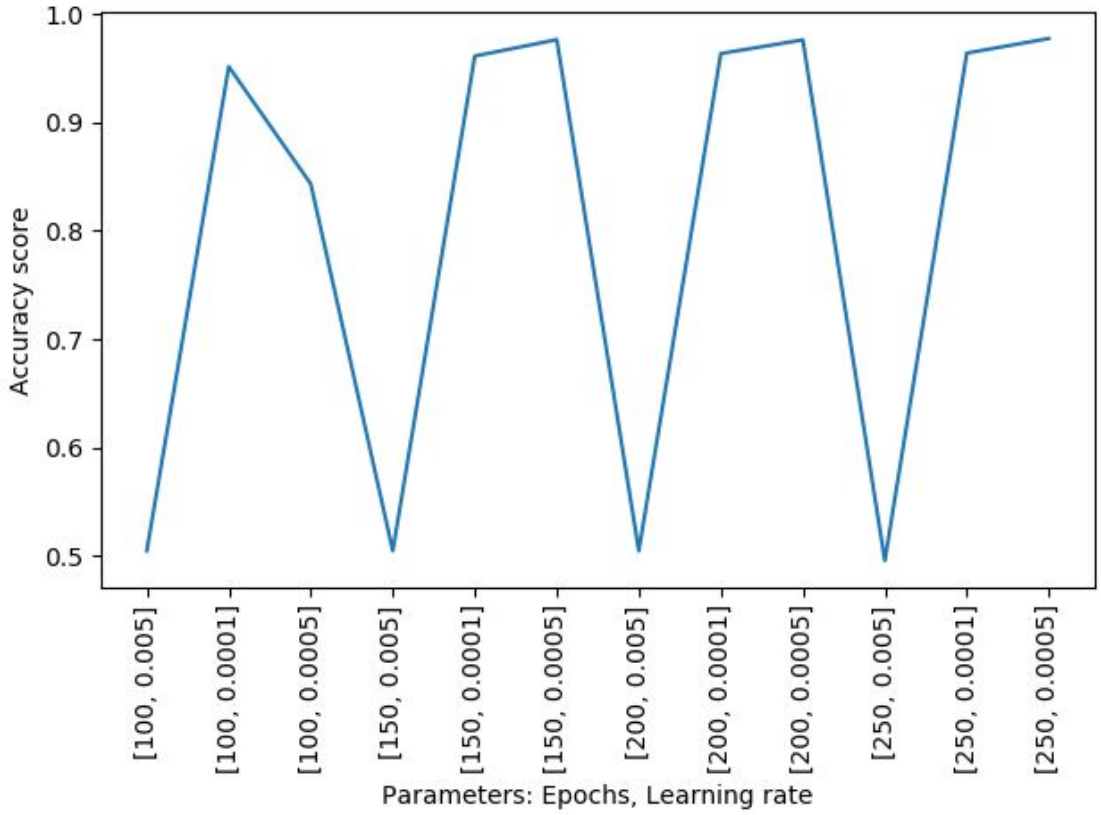
For MNIST subset:

learning rate: 0.0001

Number of epochs	Accuracy
50	0.94702130572
100	0.95670491553
150	0.9620378648
200	0.962616989428
250	0.966177920243



Grid search on epochs and learning rate:

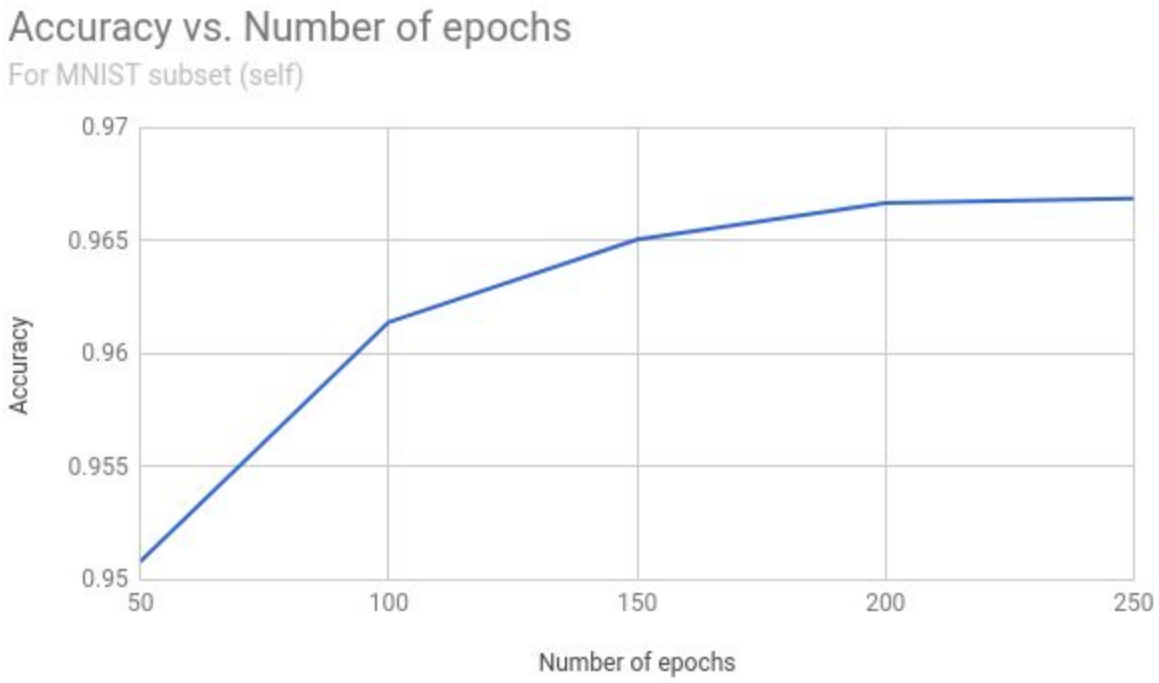


(b) Softmax at outermost layer and Sigmoid in others:

For MNIST subset:

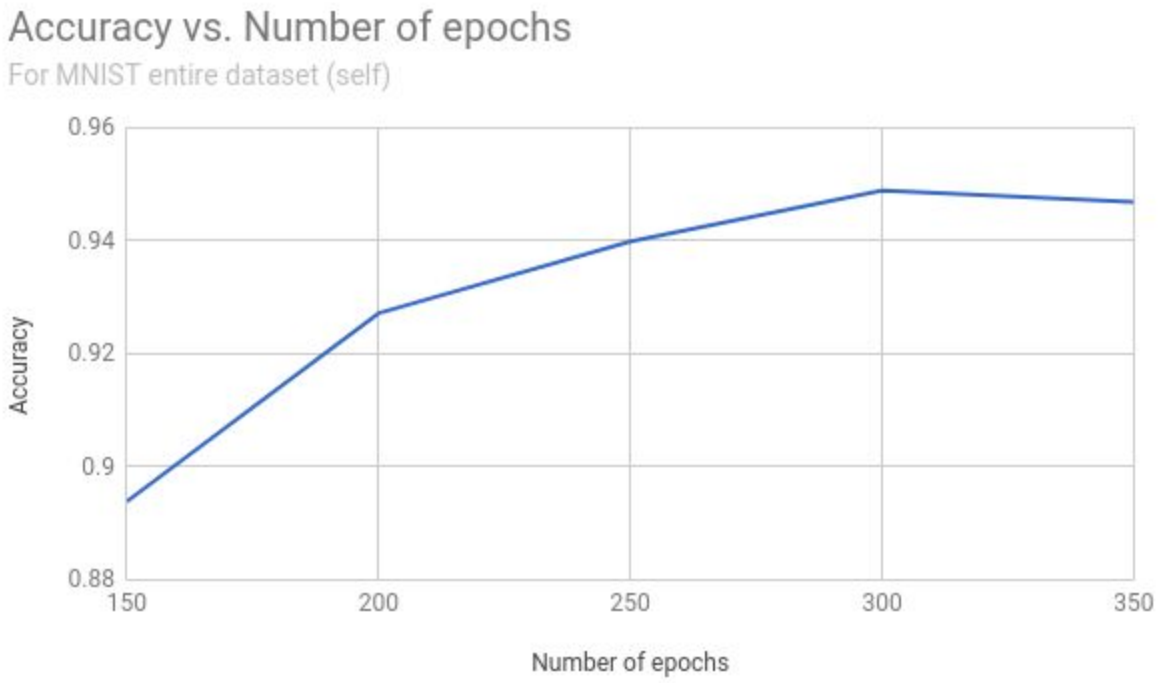
Learning rate: 0.00015

Number of epochs	Accuracy
50	0.950810671078
100	0.961406315396
150	0.965055063986
200	0.966669010444
250	0.966879576151



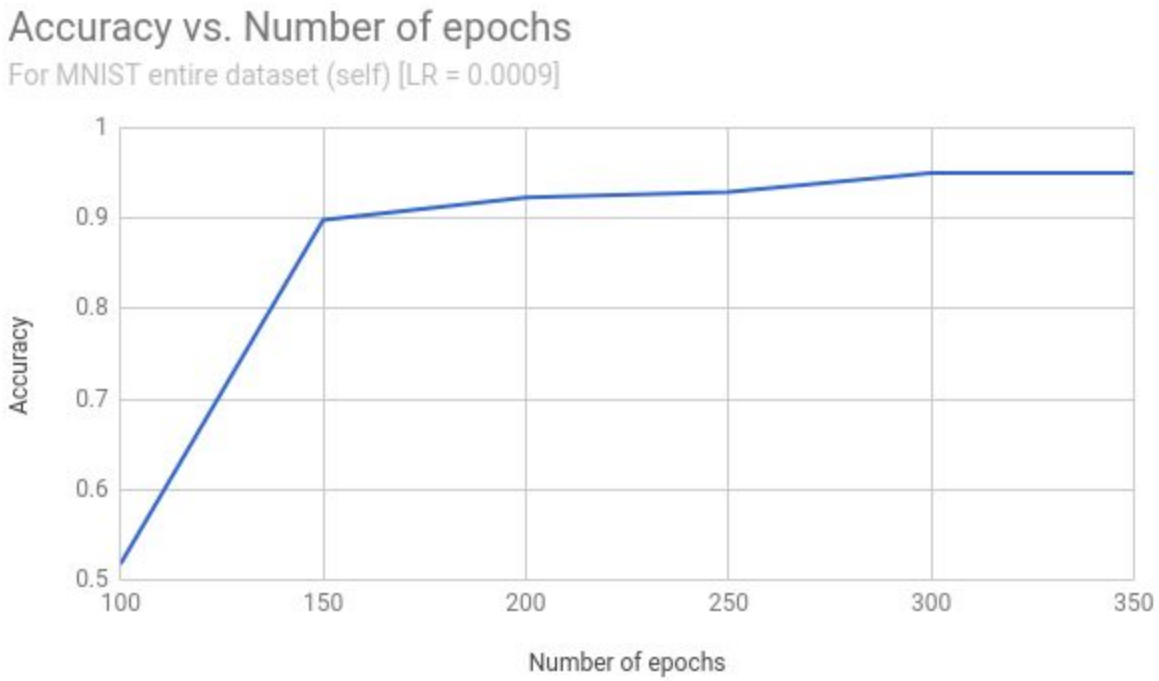
For entire MNIST dataset:
Learning rate: 0.00085

Number of epochs	Accuracy
150	0.8939
200	0.9272
250	0.9399
300	0.9489
350	0.9469

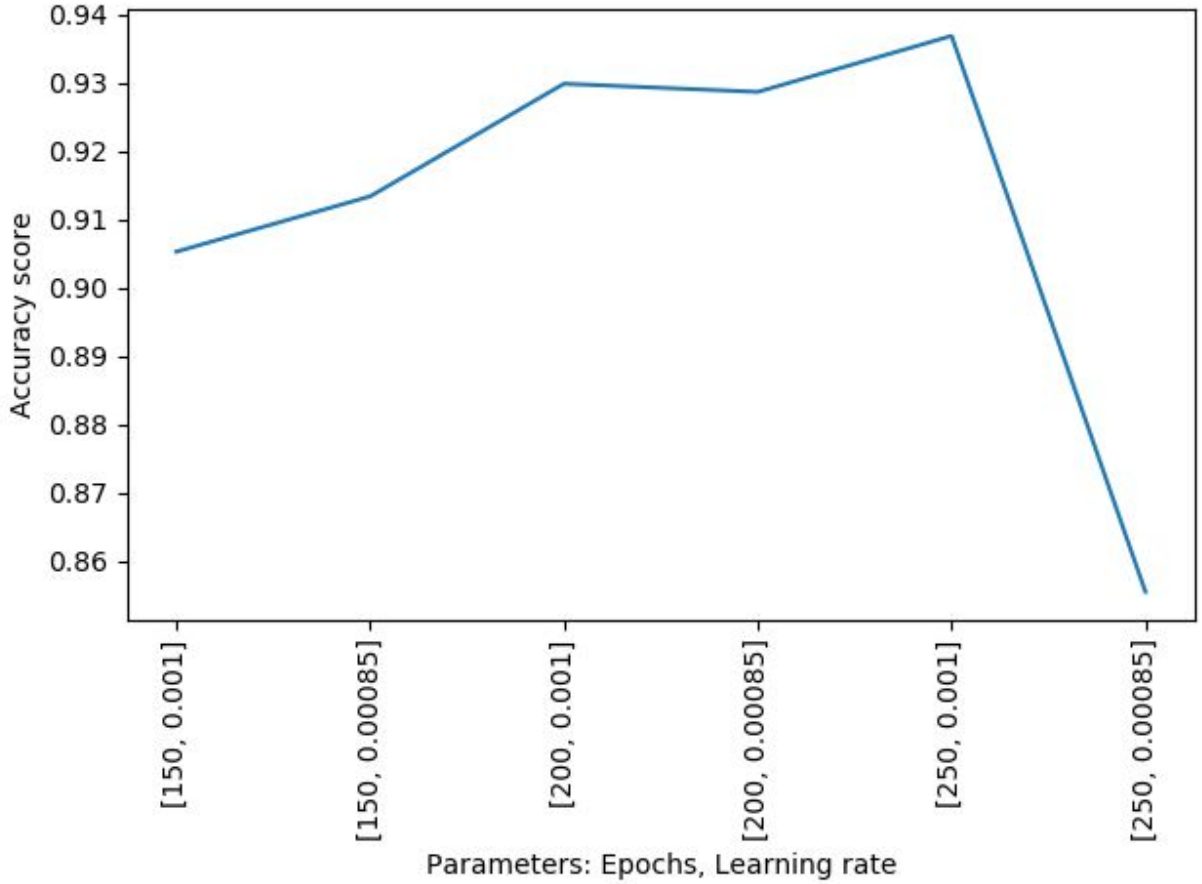


For entire MNIST dataset:
Learning rate: 0.0009

Number of epochs	Accuracy
100	0.5178
150	0.898
200	0.9231
250	0.9292
300	0.9503
350	0.9501



Grid search on epochs and learning rate:

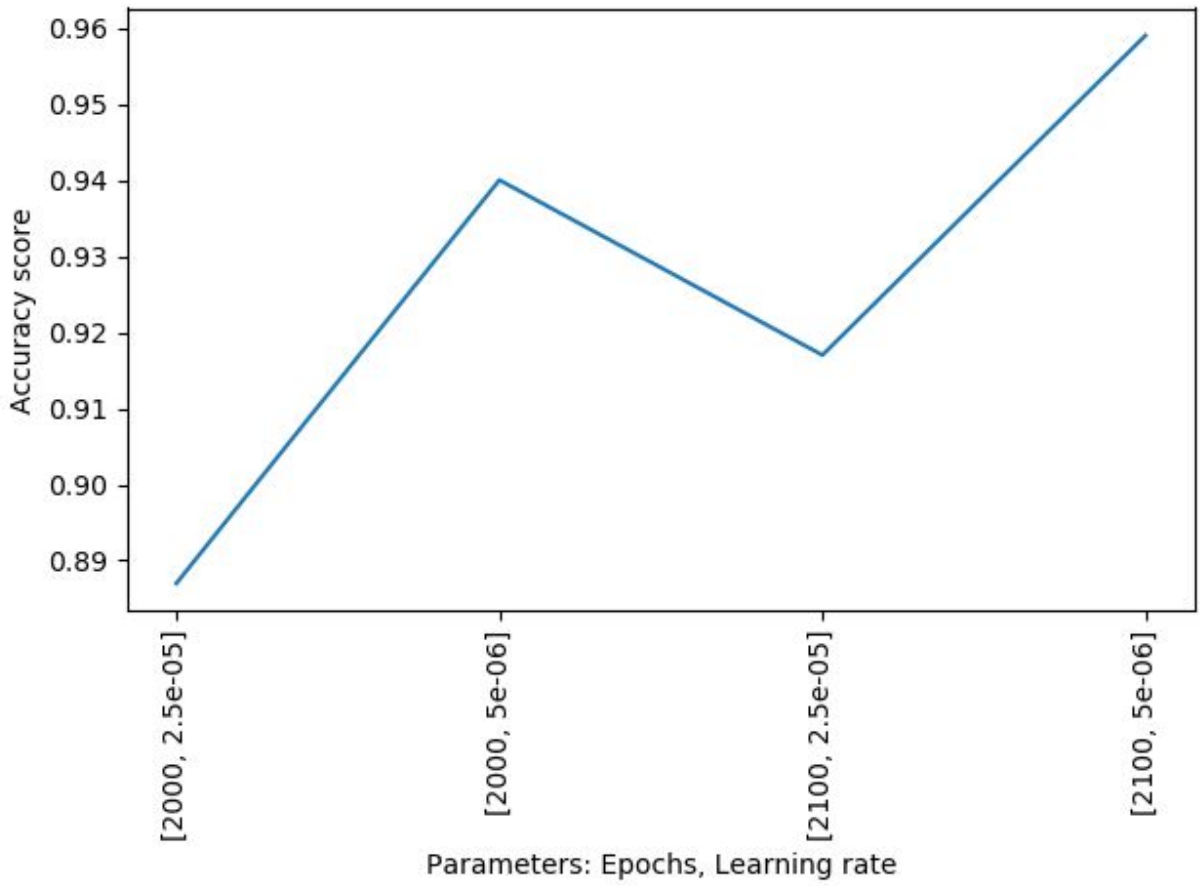


(c) Softmax at outermost layer and RELU in others:

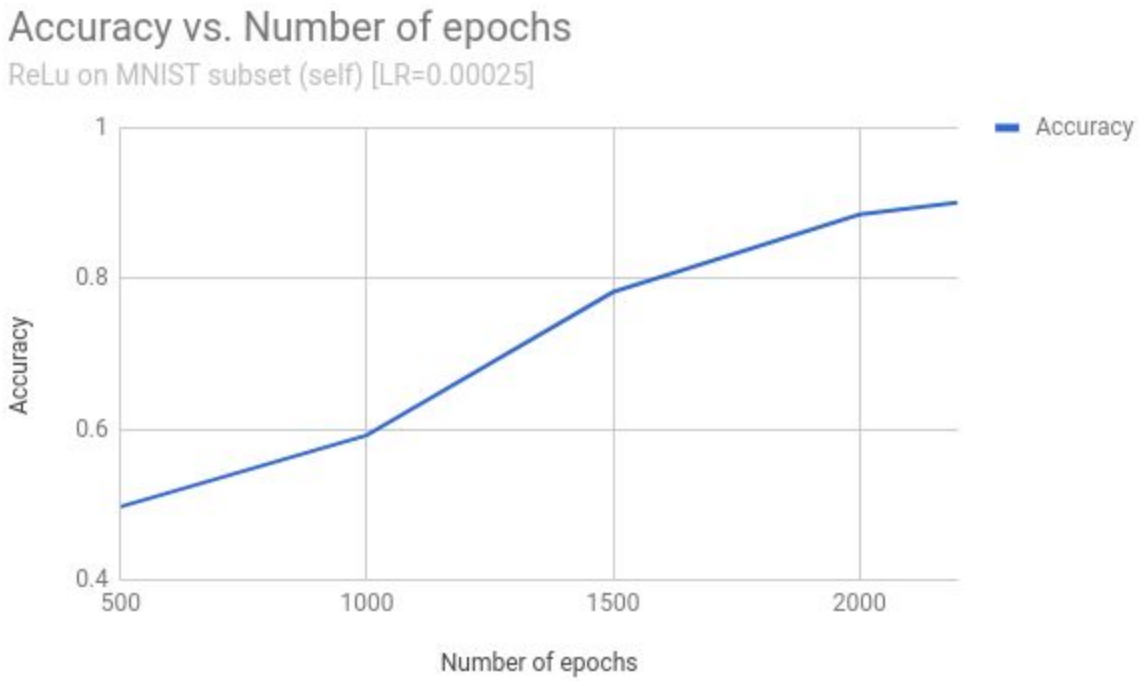
Grid search on epochs and learning rate:

For MNIST subset:
Learning rate: 0.000025

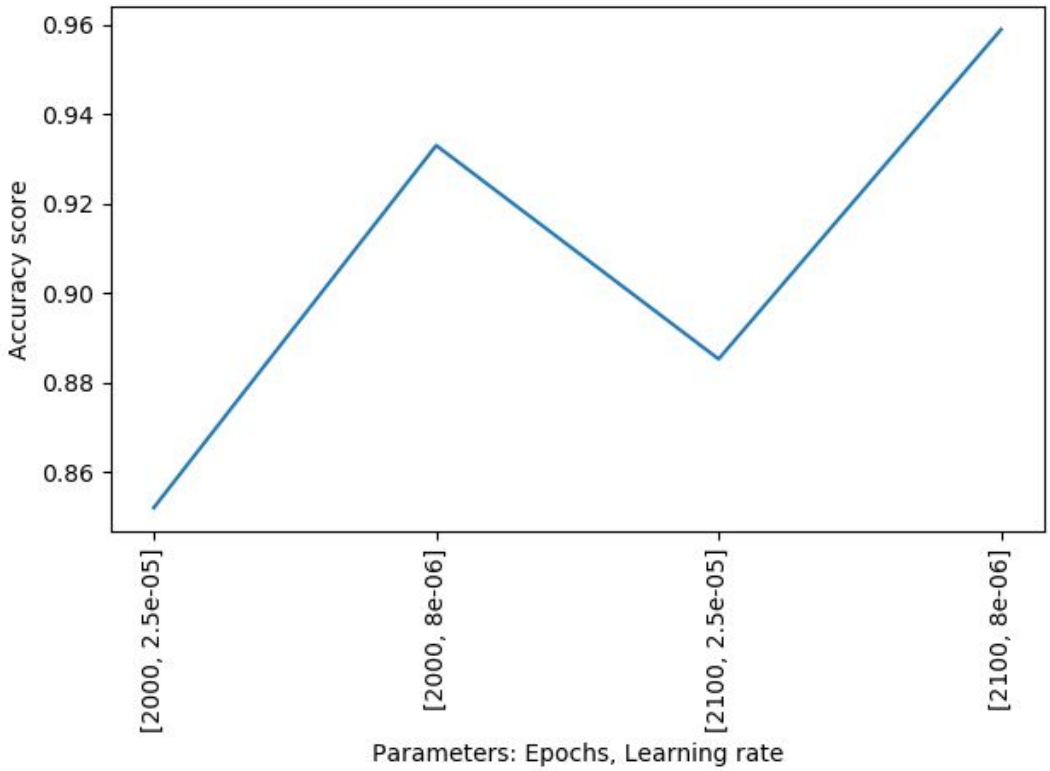
Grid search : epochs and learning rate



Number of epochs	Accuracy
500	0.497
1000	0. 641
1500	0.782
2000	0.885
2200	0.906

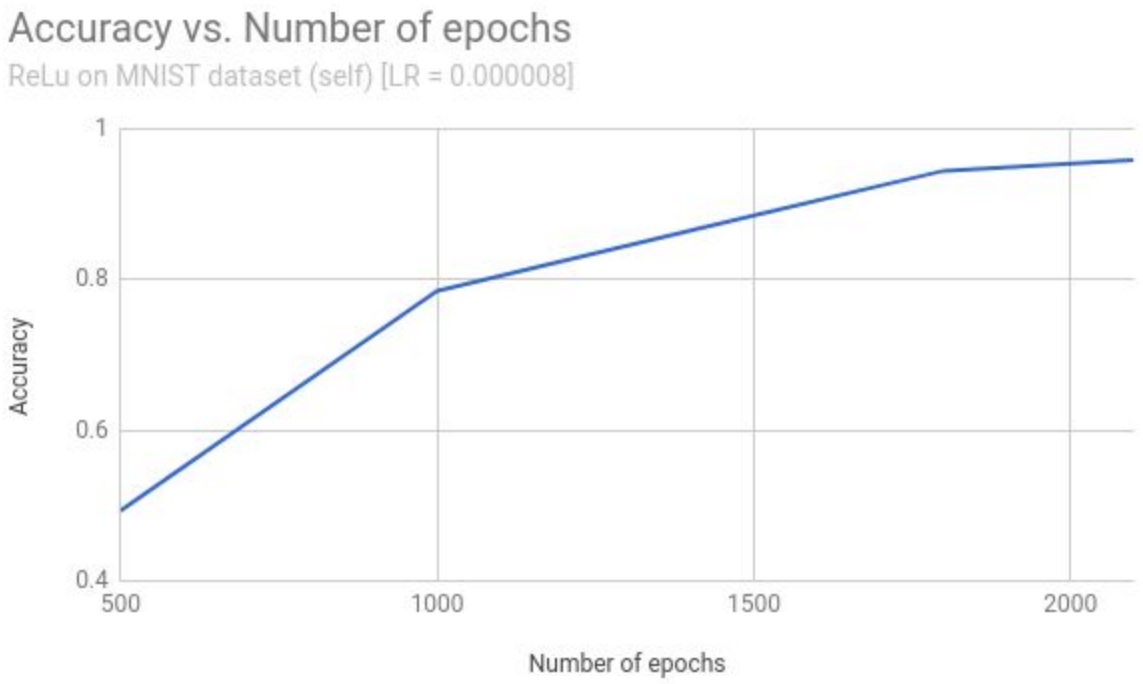


For entire MNIST dataset:



Learning rate: 0.000008

Number of epochs	Accuracy
500	0.493
1000	0.785
1500	0.8851
1800	0.9443
2100	0.9589

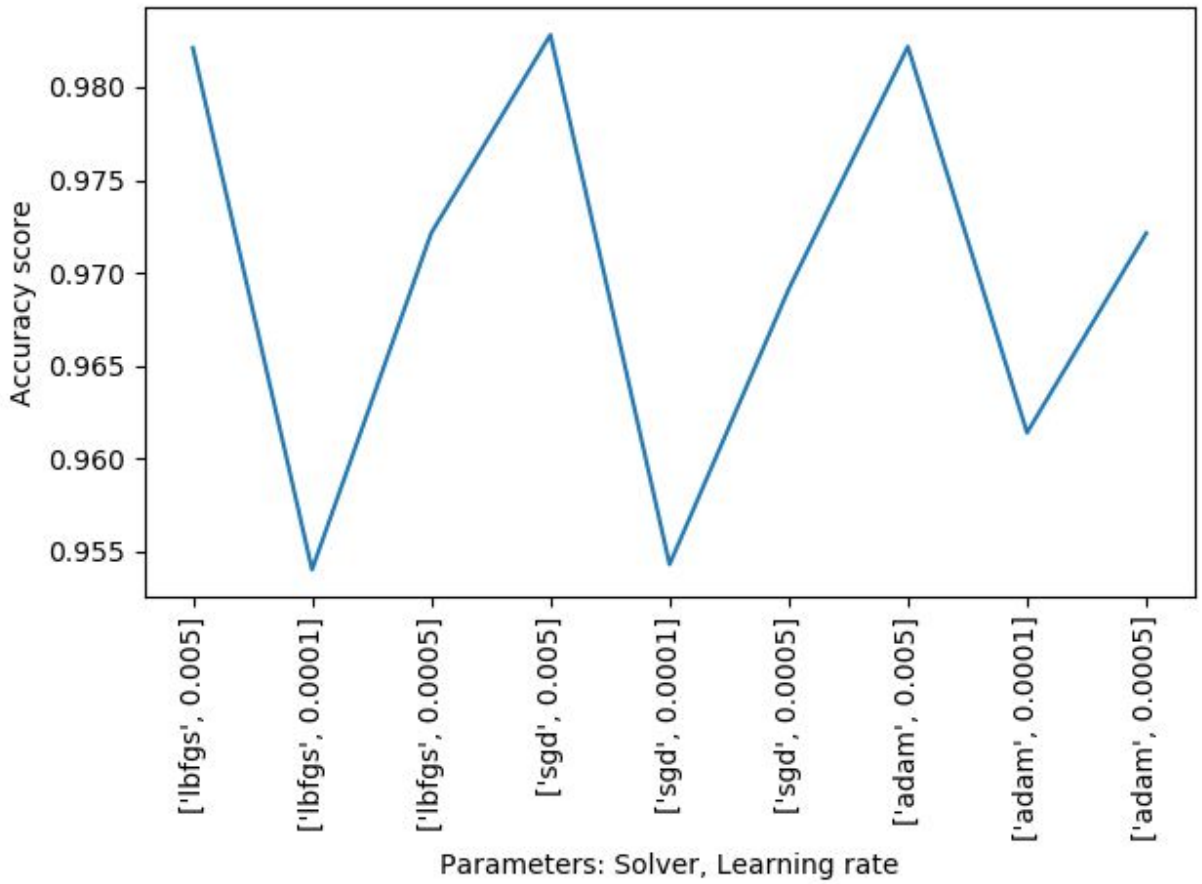


QUESTION 2

a) Sigmoid activation:

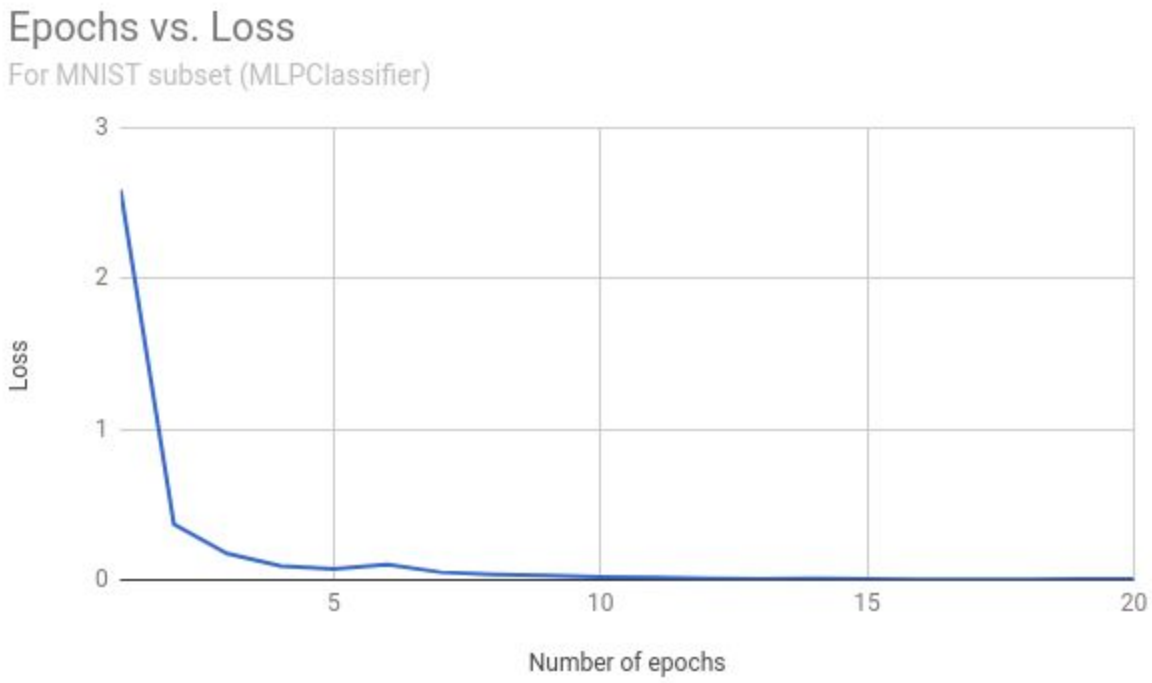
For MNIST subset:

Grid search on solver and learning rate:



Max accuracy: 0.98280701754385968
Best learning rate: 0.005)
Best solver: 'sgd'

Epochs vs. Loss: (Using sgd)

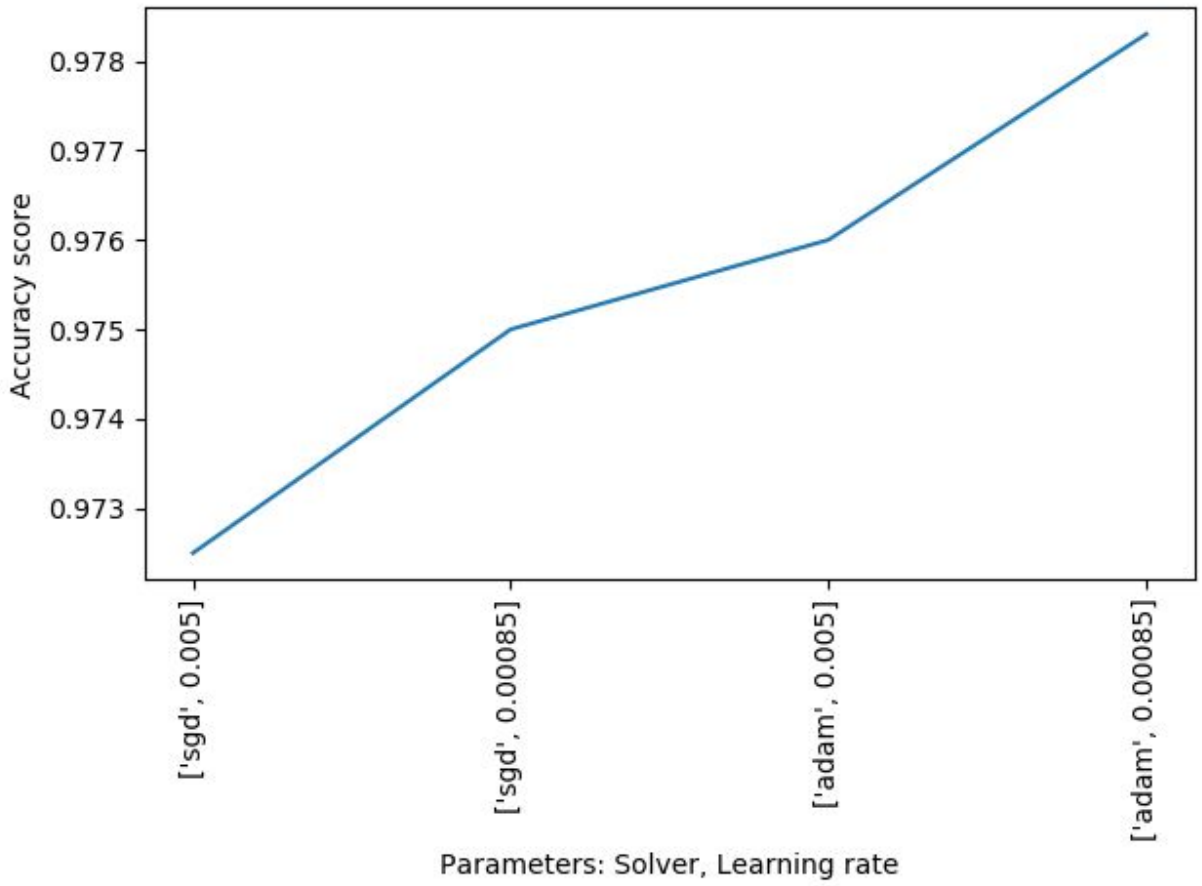


b) Softmax at outermost layer and Sigmoid in others:

For MNIST dataset:

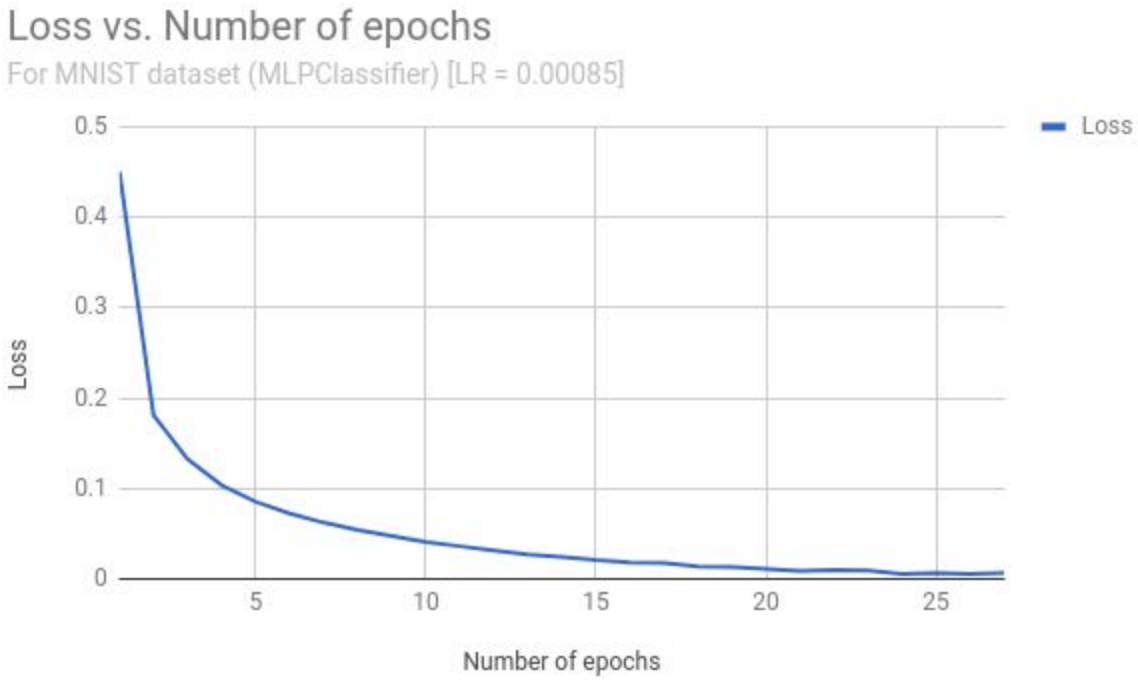
Grid search on solver and learning rate:

The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better. So, not using lbfgs for entire MNIST dataset.



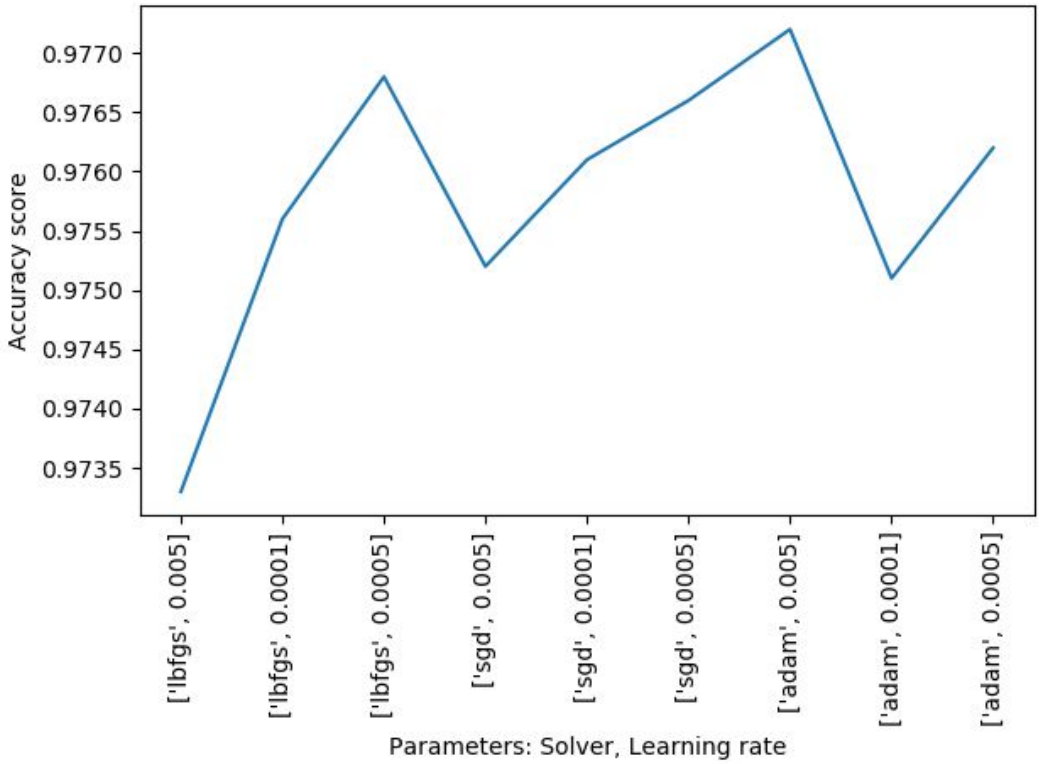
Max accuracy = 0.97829999999999995
Best solver: adam
Best learning rate: 0.00085

For entire MNIST dataset:
Learning rate: 0.00085

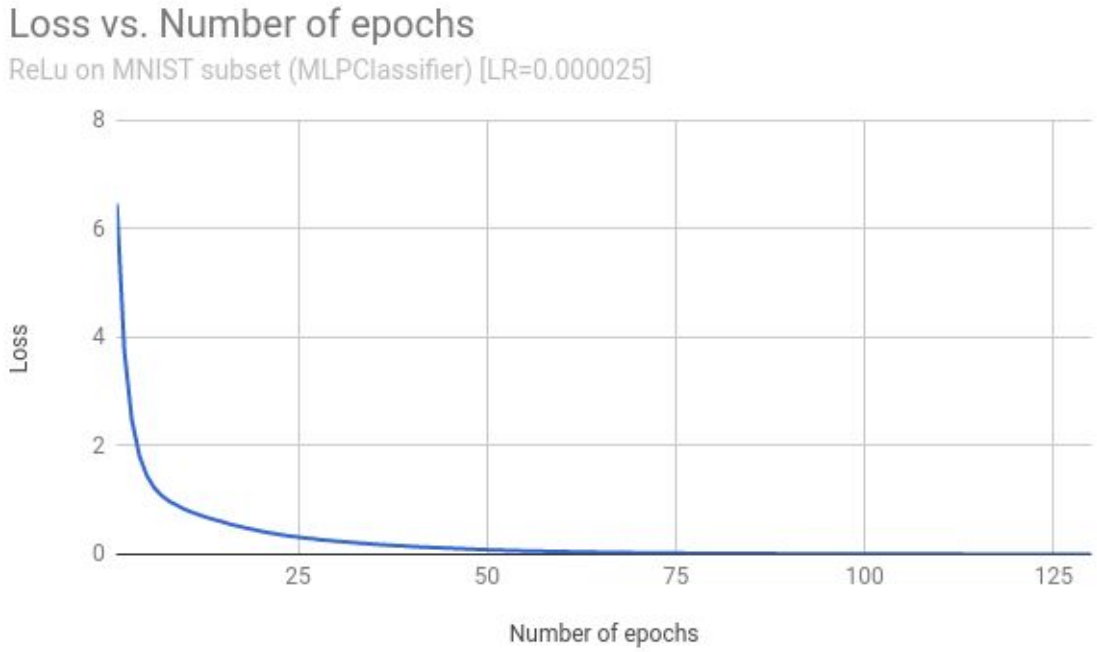


c) Softmax at outermost layer and RELU in others:

Grid search on solver and learning rate:

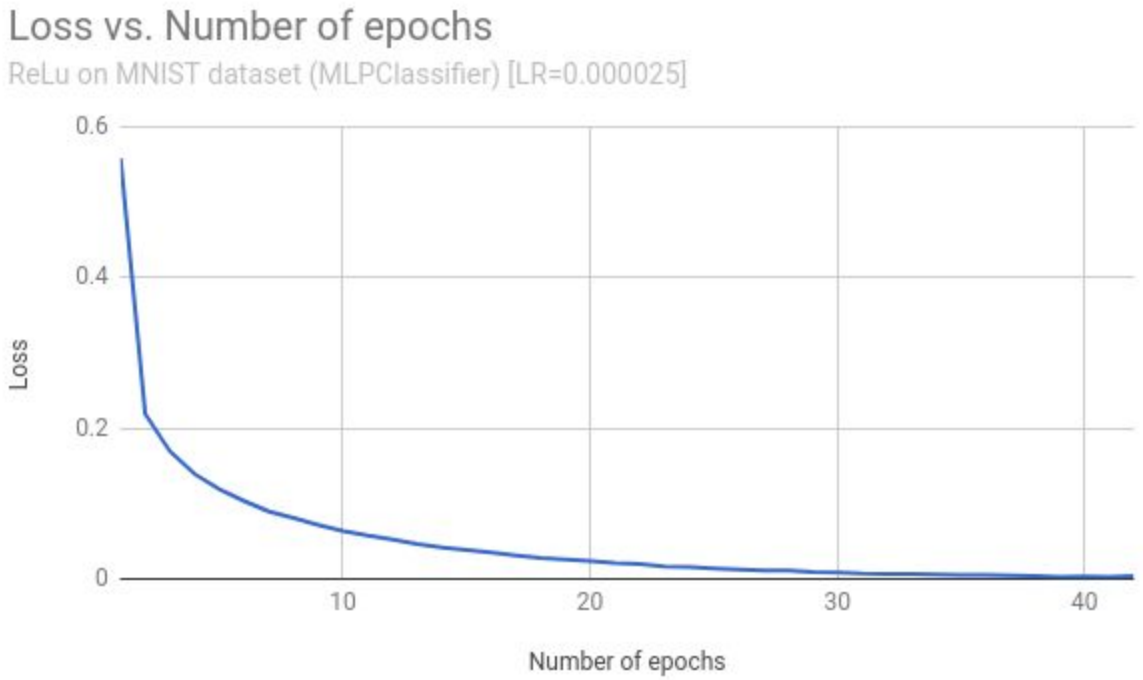


For MNIST subset:
Learning rate: 0.000025



Accuracy = 0.961122807018

For entire MNIST dataset:



Accuracy = 0.9748

Comparison of accuracies:
Model 1: Sigmoid on outermost for small dataset
Model 2: Softmax on outermost for large dataset
Model 3: Sigmoid on outermost and ReLu on hidden for small dataset
Model 4: Softmax on outermost and ReLu on hidden for large dataset

Model	**Accuracy on self implementation**	**Accuracy on MLPClassifier**
Model 1	0.966	0.98
Model 2	0.95	0.978
Model 3	0.906	0.961
Model 4	0.9589	0.9748

There is a difference in the accuracies between the self implementation and sklearn's MLPClassifier because MLPClassifier uses some techniques which give between accuracy:

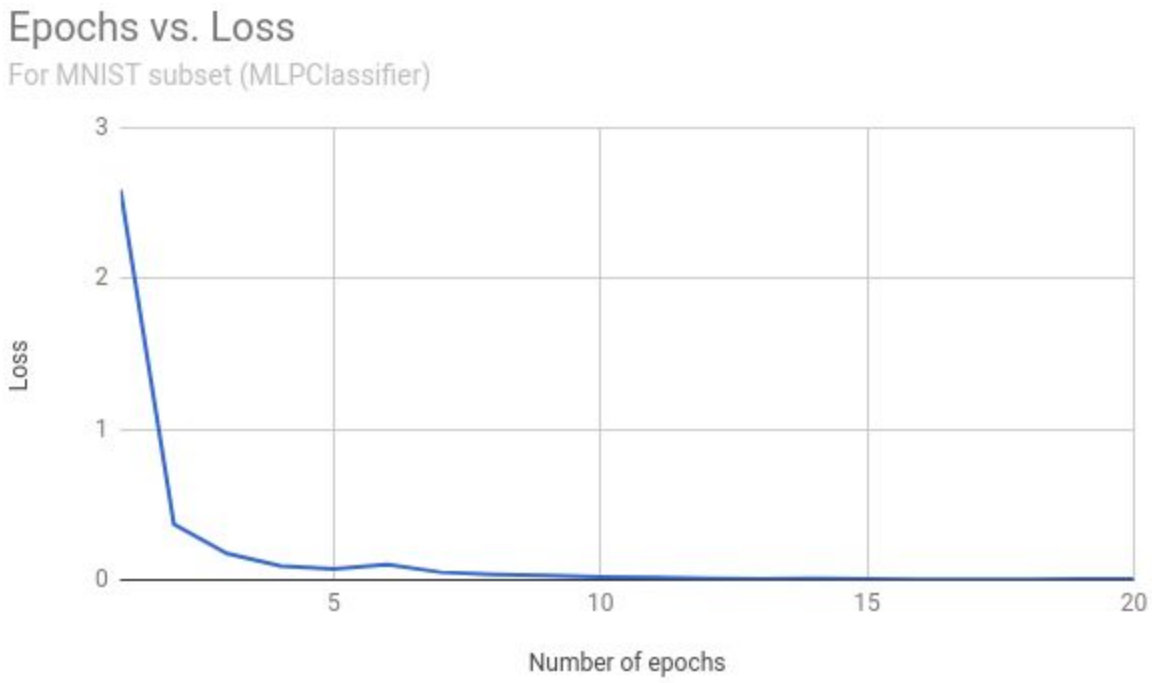
- It keeps decreasing the learning rate if the model is not converging.
- It gives the options to use various solvers: 'lbfgs', 'sgd', 'adam'. 'Lbfgs' is better for smaller datasets and others for larger datasets.
- It keeps on iterating till the time the model converges while my implementation requires specifying the number of epochs.
- ReLu has a significant amount of difference in accuracies as there was a lot of invalid value and overflow error encountered in self implementation due to which the learning rate had to be reduced to 0.000001. For such low learning rates, the number of epochs required were large and running it was time consuming. So, could reach an accuracy of around 0.88 in case of self implementation.
- Self tuning of parameters in case of MLPClassifier helps in achieving better accuracy.

Question 3:BONUS
MNIST subset

Network structure 1:

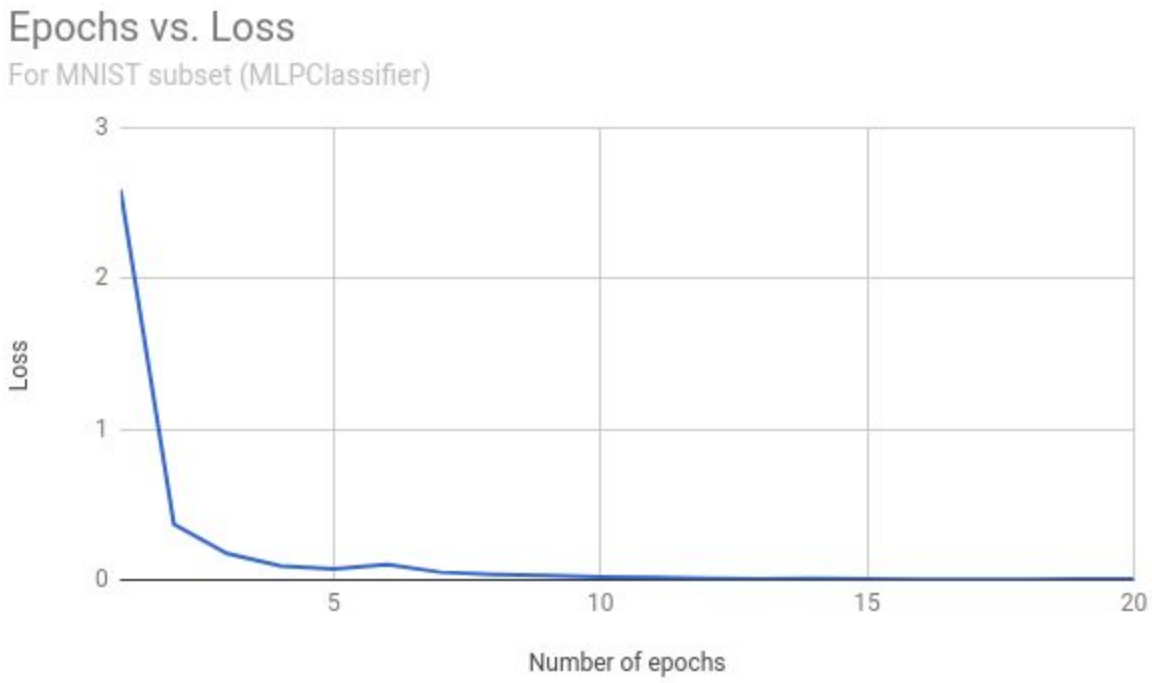
Max accuracy: 0.98280701754385968
Hidden layers: 100, 50
Best learning rate: 0.005
Best solver: 'sgd'

Epochs vs. Loss: (Using sgd)



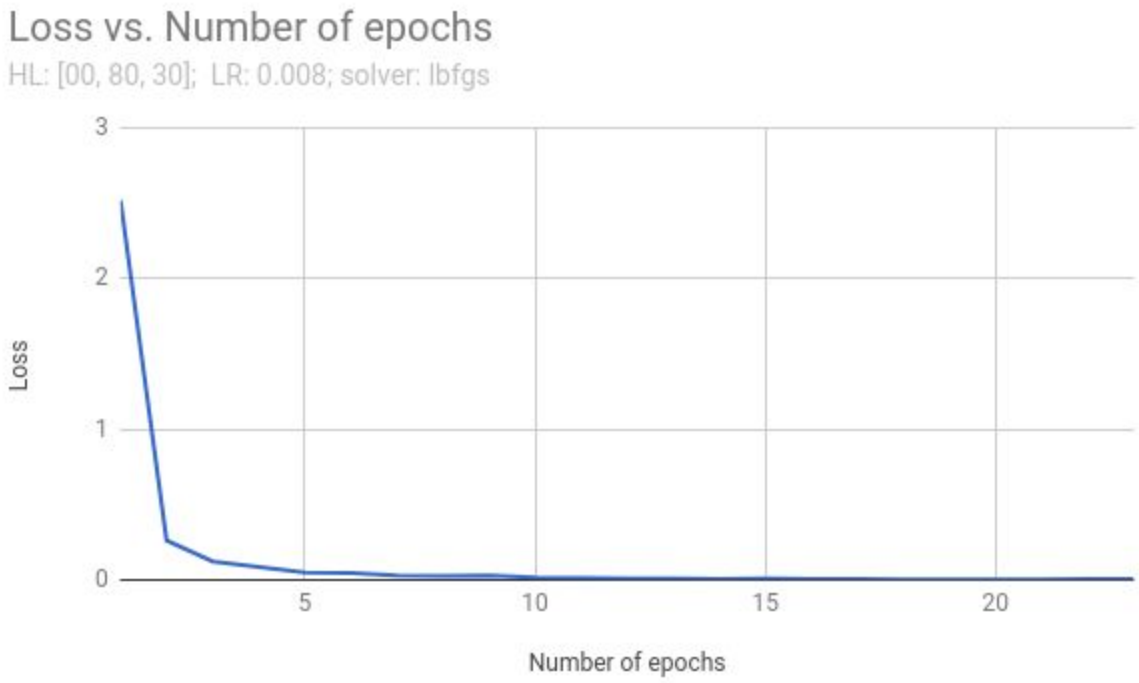
Network structure 2:
Max accuracy: 0.9817
Hidden layers: 100, 80
Best learning rate: 0.008
Best solver: lbfgs

```
Iteration 22, loss = 0.00615777
Iteration 23, loss = 0.00765611
Iteration 24, loss = 0.00388418
Iteration 25, loss = 0.00258425
Iteration 26, loss = 0.00132366
Iteration 27, loss = 0.00180115
Iteration 28, loss = 0.00136084
Iteration 29, loss = 0.00080138
Iteration 30, loss = 0.00088829
Iteration 31, loss = 0.00060169
Iteration 32, loss = 0.00051514
Iteration 33, loss = 0.00047522
Iteration 34, loss = 0.00046646
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
sigmoid + softmax
hidden layers: 100, 80
learning rate:  0.008
solver:  lbfgs
0.981754385965
```



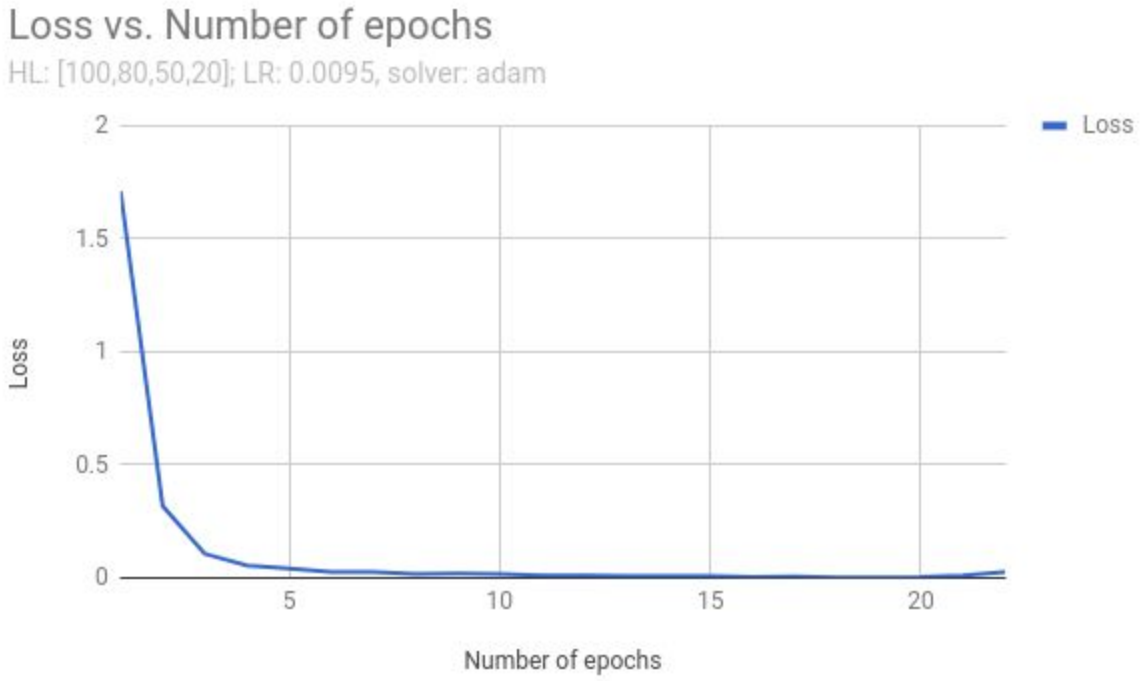
Network structure 3:
hidden layers: 100, 80, 30
learning rate: 0.008
solver: lbfgs
0.985403508772

```
Iteration 6, loss = 0.04724782
Iteration 7, loss = 0.02872706
Iteration 8, loss = 0.02621616
Iteration 9, loss = 0.02979023
Iteration 10, loss = 0.01481896
Iteration 11, loss = 0.01295301
Iteration 12, loss = 0.01024812
Iteration 13, loss = 0.00975676
Iteration 14, loss = 0.00576071
Iteration 15, loss = 0.00908104
Iteration 16, loss = 0.00739888
Iteration 17, loss = 0.00447123
Iteration 18, loss = 0.00396916
Iteration 19, loss = 0.00270570
Iteration 20, loss = 0.00143146
Iteration 21, loss = 0.00167427
Iteration 22, loss = 0.00724857
Iteration 23, loss = 0.00579310
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping
sigmoid + softmax
hidden layers: 100, 80, 30
learning rate:  0.008
solver:  lbfgs
0.985403508772
```



Network structure 4:
hidden layers: 100, 80, 50, 20
learning rate: 0.0095
solver: adam
0.987228070175

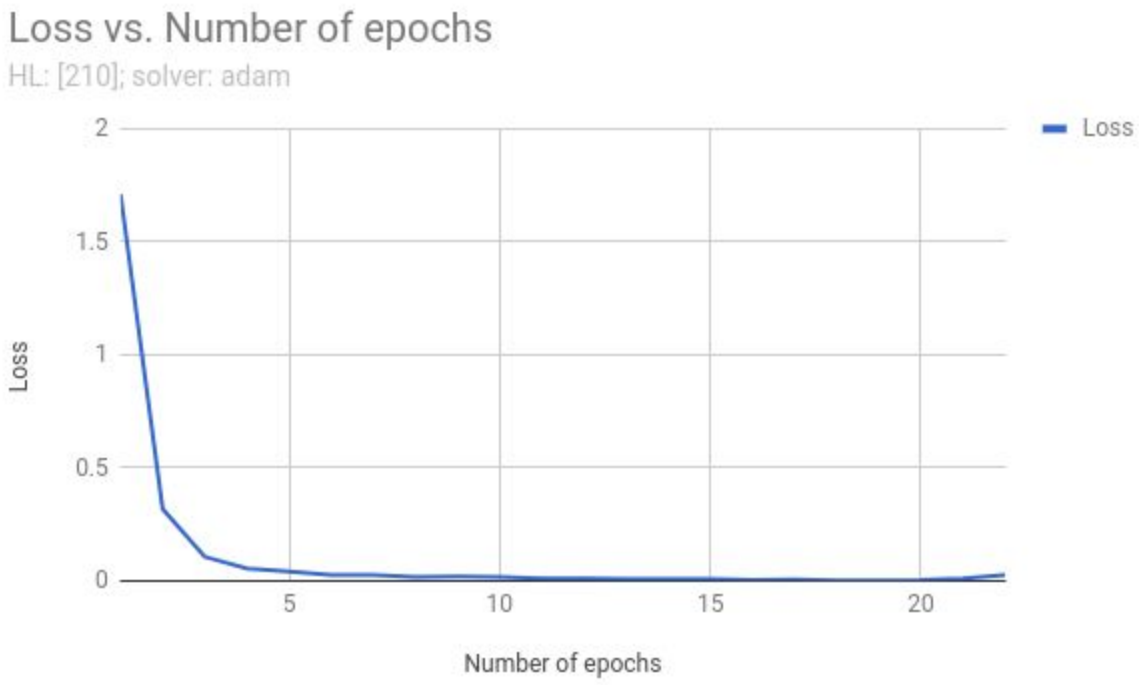
```
(14251, 784)
Iteration 1, loss = 2.78839332
Iteration 2, loss = 0.41449539
Iteration 3, loss = 0.12156596
Iteration 4, loss = 0.07760322
Iteration 5, loss = 0.06589469
Iteration 6, loss = 0.05771678
Iteration 7, loss = 0.07729049
Iteration 8, loss = 0.03597802
Iteration 9, loss = 0.02607139
Iteration 10, loss = 0.04731049
Iteration 11, loss = 0.03495158
Iteration 12, loss = 0.02017508
Iteration 13, loss = 0.01805925
Iteration 14, loss = 0.01212001
Iteration 15, loss = 0.00809473
Iteration 16, loss = 0.00879295
Iteration 17, loss = 0.01823432
Iteration 18, loss = 0.00804315
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Sto
sigmoid + softmax
hidden layers: 100, 80, 50, 20
learning rate:  0.0095
solver:  adam
0.987228070175
```



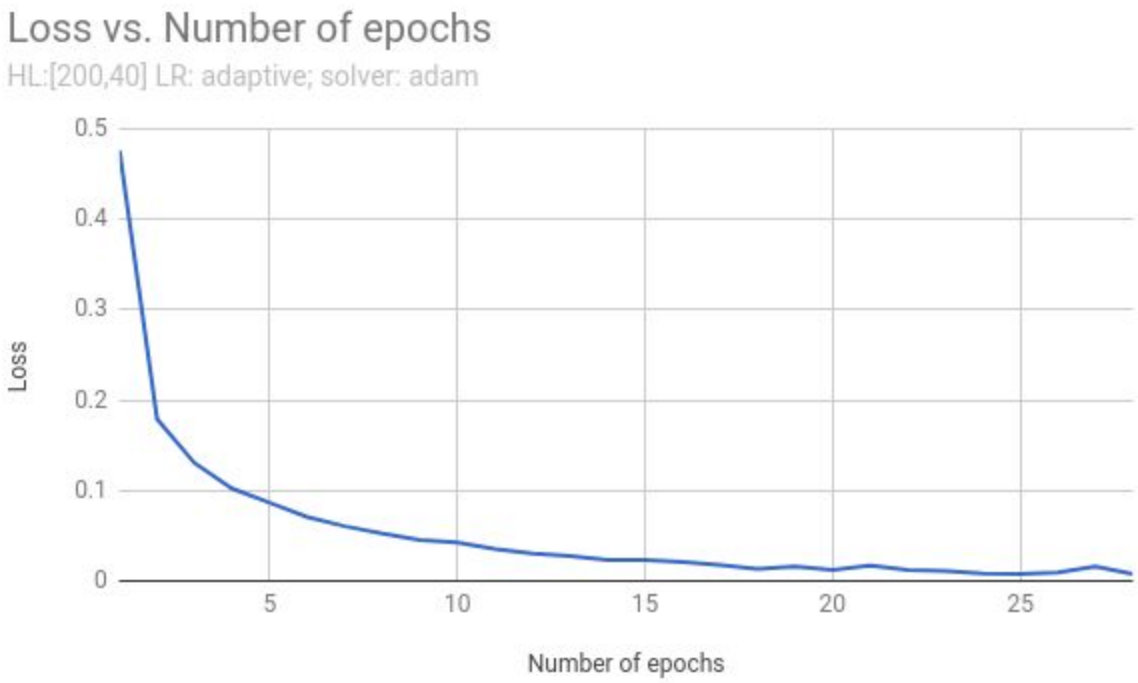
MNIST entire dataset

Network structure 1:
learning rate: adaptive
solver: adam
('Mean accuracy is: ', 0.979)
Hidden layers: 210

```
Iteration 14, loss = 0.01231085
Iteration 15, loss = 0.00833120
Iteration 16, loss = 0.00658578
Iteration 17, loss = 0.00622714
Iteration 18, loss = 0.00604770
Iteration 19, loss = 0.01218258
Iteration 20, loss = 0.01113766
Iteration 21, loss = 0.00702400
Training loss did not improve more than tol=0.000100 for two cons
0.9795
================================================
softmax on outer, sig on others
learning rate:  0.001
solver:  adam
================================================
('Mean accuracy is: ', 0.97950000000000004)
================================================
```

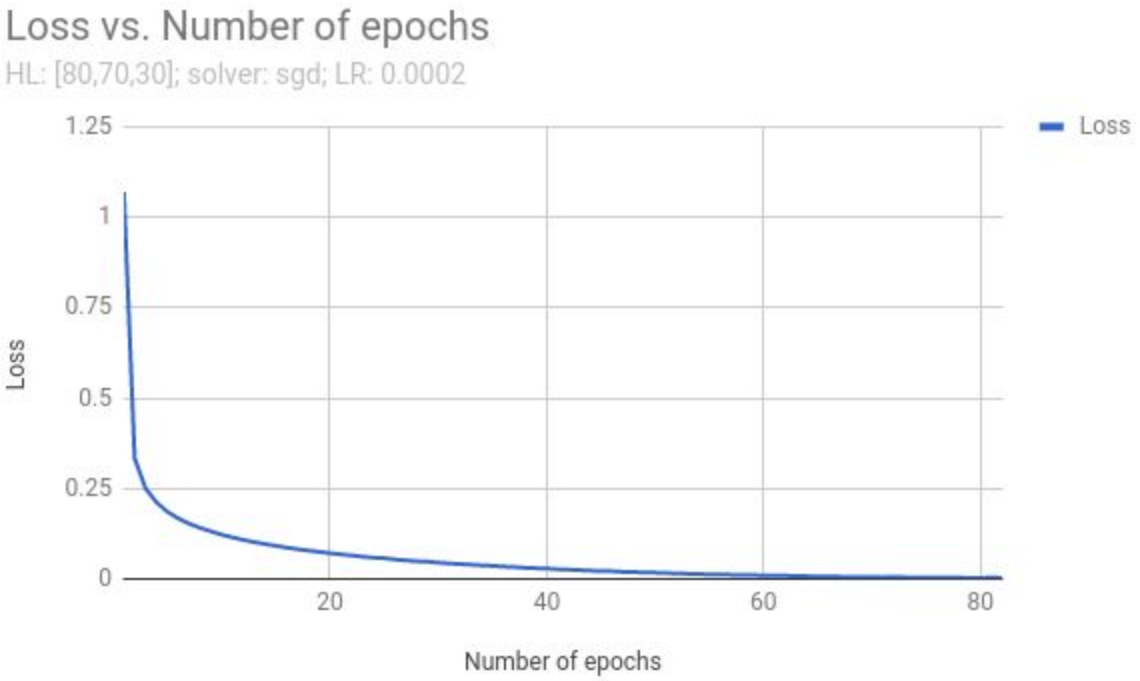


Network structure 2:
learning rate: adaptive
solver: sgd
Mean accuracy is: ', 0.98
Hidden layers: [200, 40]



Network structure 3:
learning rate: 0.0002
solver: sgd
('Mean accuracy is: ', 0.97640000000000005)

```
Iteration 71, loss = 0.00450141
Iteration 72, loss = 0.00489954
Iteration 73, loss = 0.00457957
Iteration 74, loss = 0.00420047
Iteration 75, loss = 0.00368951
Iteration 76, loss = 0.00321927
Iteration 77, loss = 0.00372611
Iteration 78, loss = 0.00356362
Iteration 79, loss = 0.00245346
Iteration 80, loss = 0.00272413
Iteration 81, loss = 0.00361339
Iteration 82, loss = 0.00292698
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
0.9764
==========================================
softmax on outer, sig on others
learning rate:  0.0002
solver:  sgd
==========================================
('Mean accuracy is: ', 0.97640000000000005)
```



Theory Ques 1:

It is not possible to use linear activation for making a classifier for XOR. Please refer to the solution in the folder Report-TheoryQues

---

Theory Ques 2:

Sol: This might be happening due to the **vanishing gradient problem**. When the number of layers is more, then after a point during back propagation, the partial derivative (or the gradient) of the sigmoid activation disappears due to which the model cannot be trained. The vanished gradient makes some of the neurons inactive (at which the gradient disappears).
This happens because the sigmoid derivative gives a maximum value of ¼. As more and more layers are used, the derivative values decrease exponentially.

Other possible problem can be **exploding gradient problem,** which occurs when the weights and bias are initialized with large values.

If ReLu is used instead, then the problem can be solved as explained below.

$$R(x)=\max(0,x)$$

ReLu function solves this problem as it makes the negative values zero. The gradient is either 1 or 0 due to which there is no vanishing gradient problem and hence it is preferred to use ReLu for deeper n layered networks.
However, there is a limitation of ReLu (**Rectified Linear Unit**) that it should only be used within hidden layers. Softmax should be used on the output layer to compute probabilities for the classes.

Other ways to solve the problem can be:

- Use smaller weight values lying between -1 and 1 to avoid the gradient problems in sigmoid activation.
- Normalize the data initially to avoid large values and to avoid exploding gradient problem. This can be done by dividing the data with 1000 (as the range of data is between 0 and 1000).
- Dimensionality reduction should be used if the number of dimensions is very large.

Theory Ques 3:
Please refer to the solution in the folder Report-TheoryQues