CSEP546 - HW3 Answers

Name: Abhijith Jagannath
Student ID: 1776317

Autumn 2017
11 / 19 / 2017

1.1. Code description:

Usage :

$ python3 main.py <training_images> <training_labels> <test_images> <test_labels>

Implementation details:

1. Normalising the data before training gave a very good boost in performance.
2. Matrix multiplication is used for storing and calculating everything
   WeightMatrix => NumWeightsPerNeuron X NumOfNeurons
   InputToNextNeuron=> 1 X (numInputs+1): 1x51 at input layer, 1x11 at hidden layer, 1x10 at output layer
3. Uniform (np.uniform) distribution of weights between -0.5 to 0.5 worked the best for most cases. Gaussian and random implementations are done. While that do not make much difference with sigmoid, uniform works better for ReLU
4. Learning rate scheme:

```
If current_error < lowest_error
        eeta = eeta + eeta / 10
else if current_error < last_error
        eeta = eeta + eeta / 100
else if current_error > last_error
        eeta = eeta - eeta / 100
        batch_size = next lowest batch size that can divide 30,000 ( to be in sync with graphing )
```

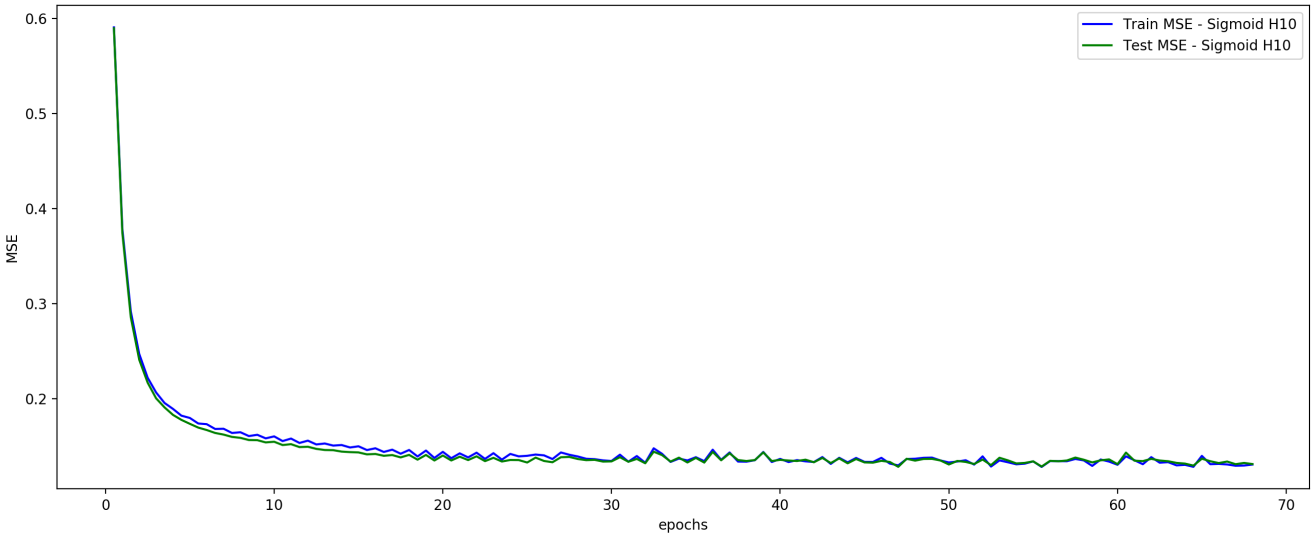
Initial learning rate of 0.01 works for all the sigmoid networks. However, for ReLU 0.0005 started to make some improvement. Also SGD converged faster and gave better results than any set batch_size. As there wasn't enough time to parallelize things, couldn't really take the advantage of batch size.

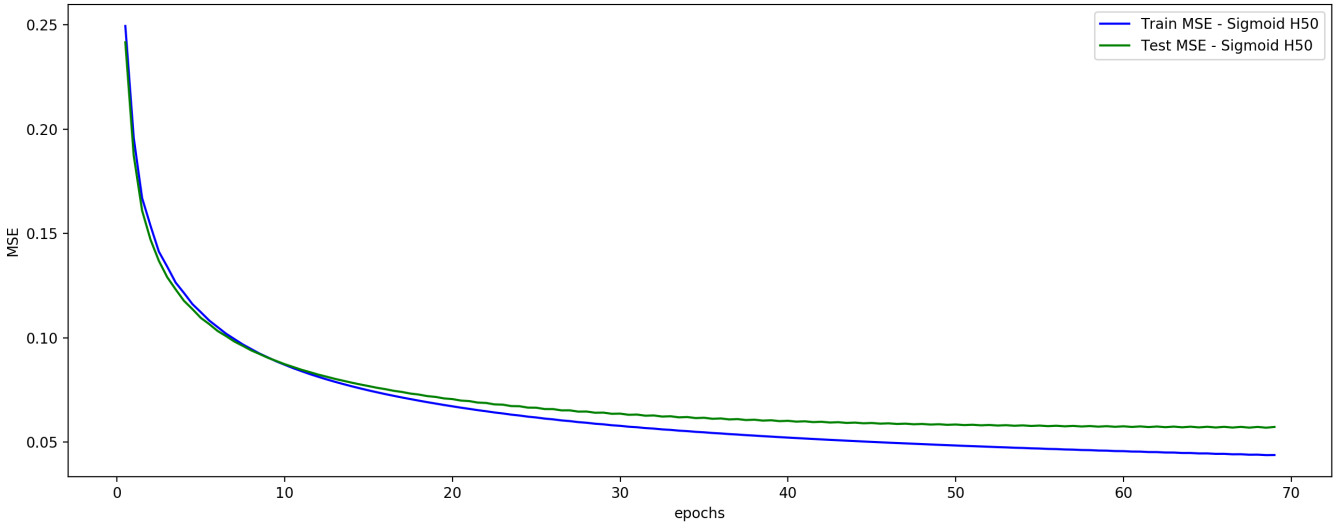
5. Exit strategy:
   Store the network with least error so far
   If the network did not improve from its lowest point even after max_epoch_distance, we stop!
   Max_epoch_distance : number of epochs after lowest error after which network did not improve.
   After stopping, we use the network which had lowest error (kind of post pruning!)

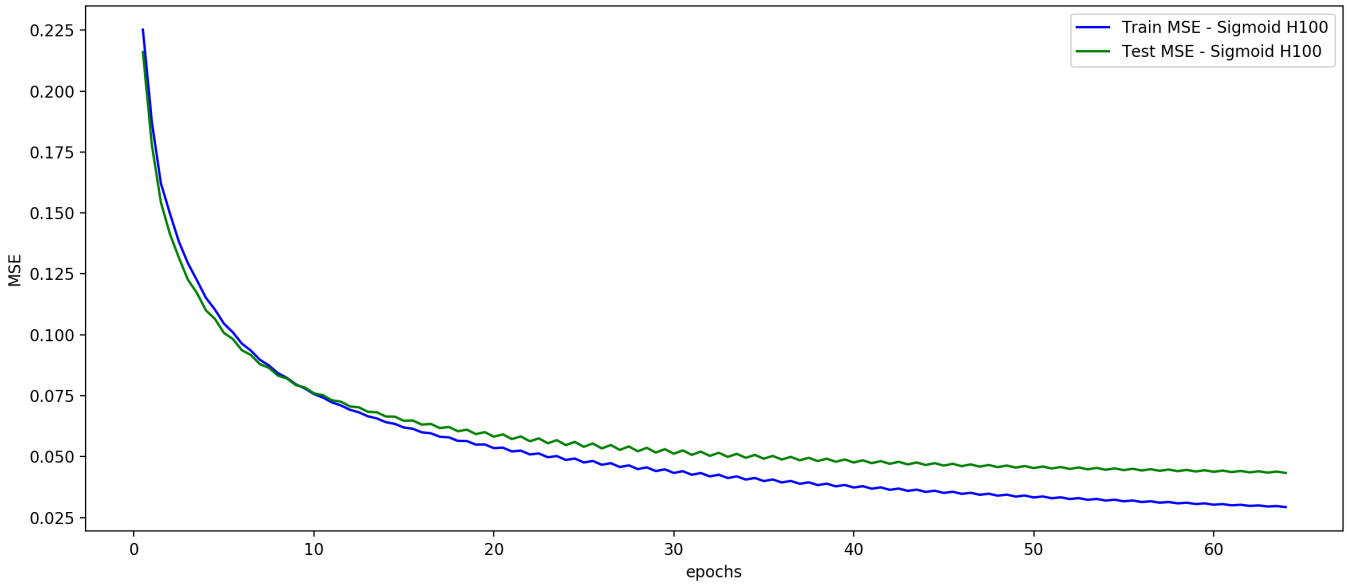
1.4 : Results:

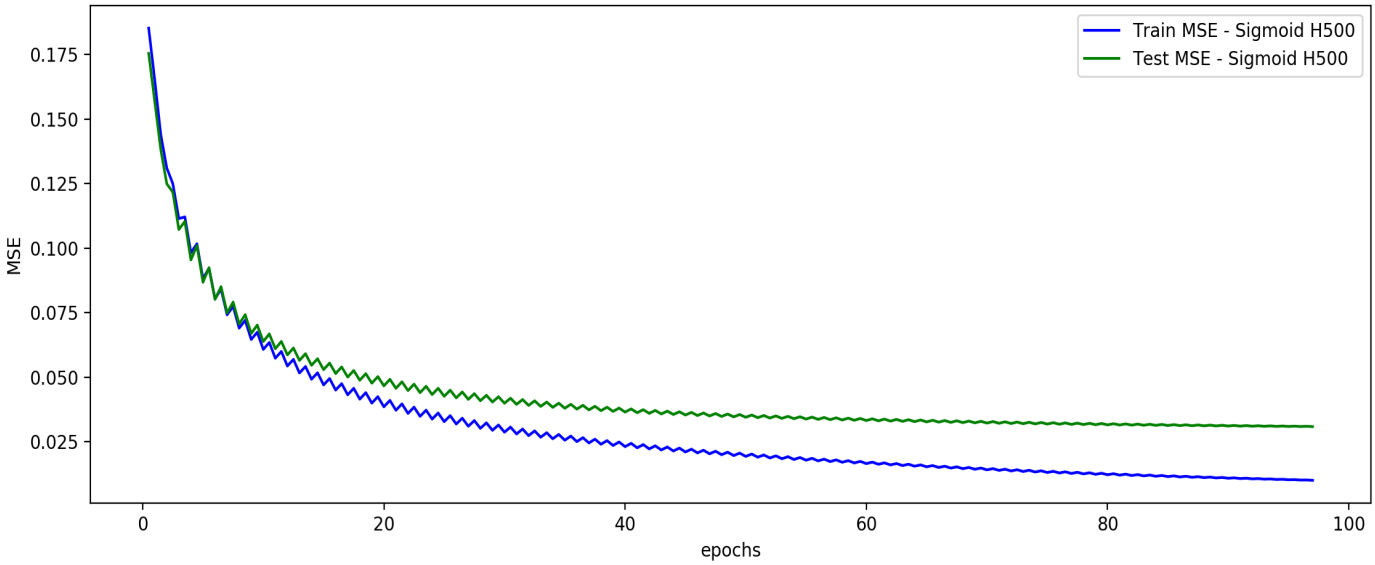
Networks	TRAIN MSE	TEST MSE	TRAIN 1/0	TEST 1/0
Sigmoid 10	0.130	0.128	7.882	7.620
Sigmoid 50	0.046	0.058	2.343	3.120
Sigmoid 100	0.032	0.044	1.542	2.310
Sigmoid 500	0.011	0.031	0.475	1.530
Sigmoid 1000	0.009	0.030	0.352	1.490
ReLU 500	0.136	0.143	4.517	4.800

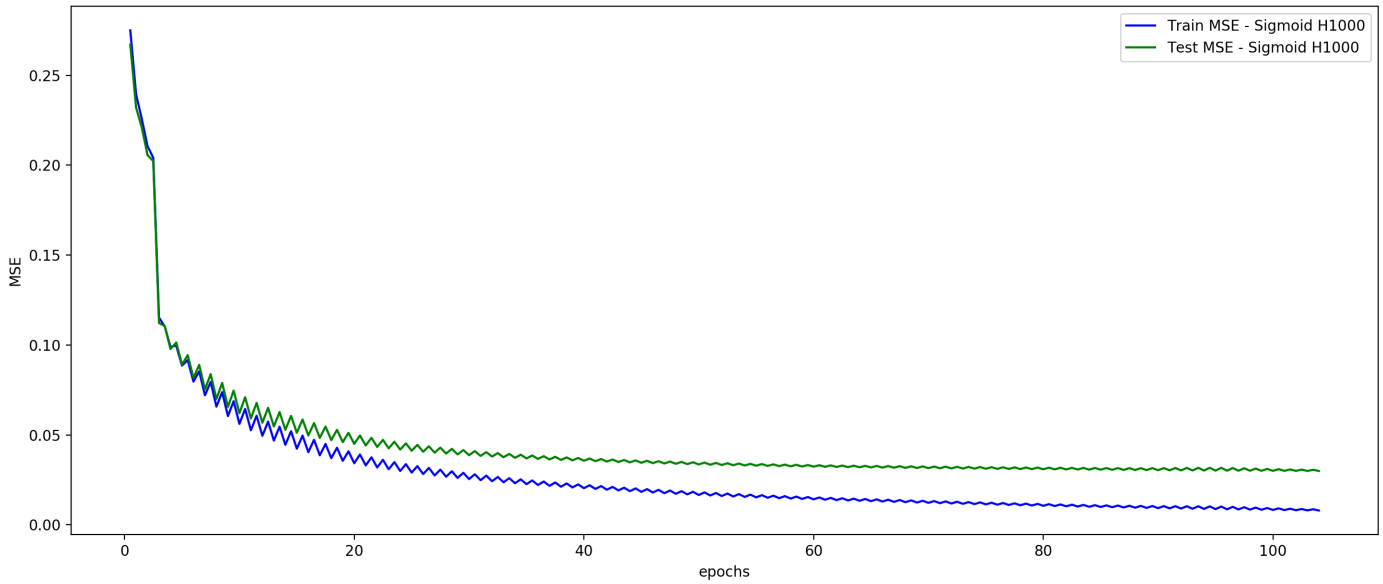
1.2. :

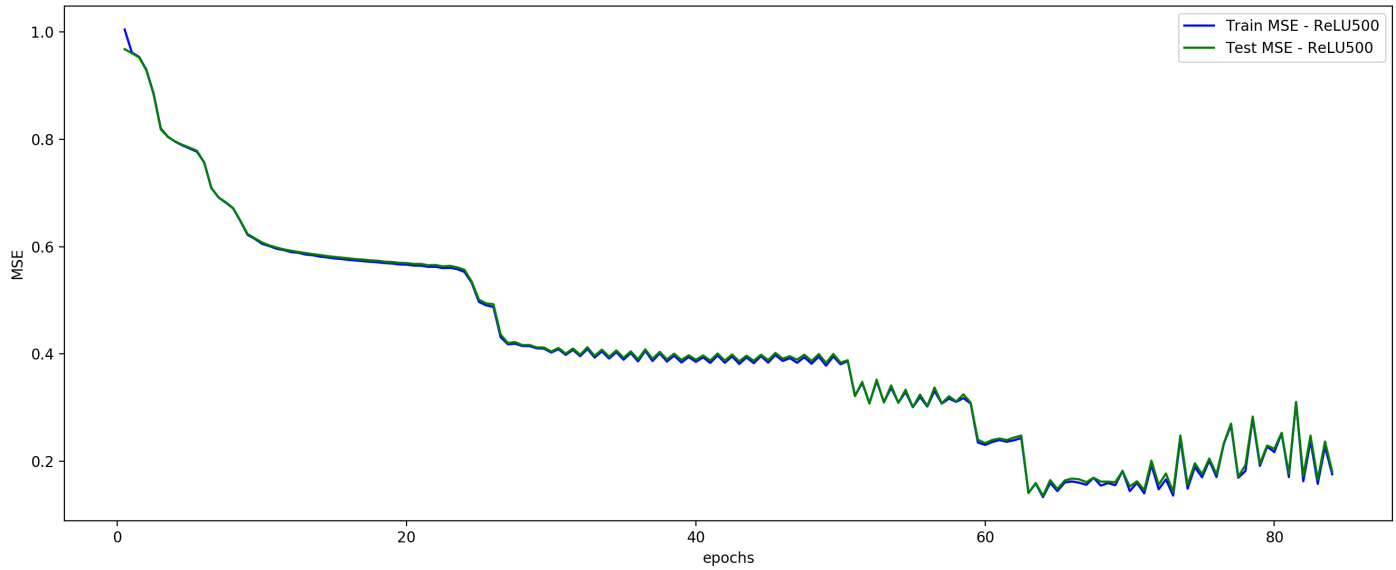

Train MSE - Sigmoid H10
Test MSE - Sigmoid H10
MSE
epochs


Train MSE - Sigmoid H50
Test MSE - Sigmoid H50
MSE
epochs

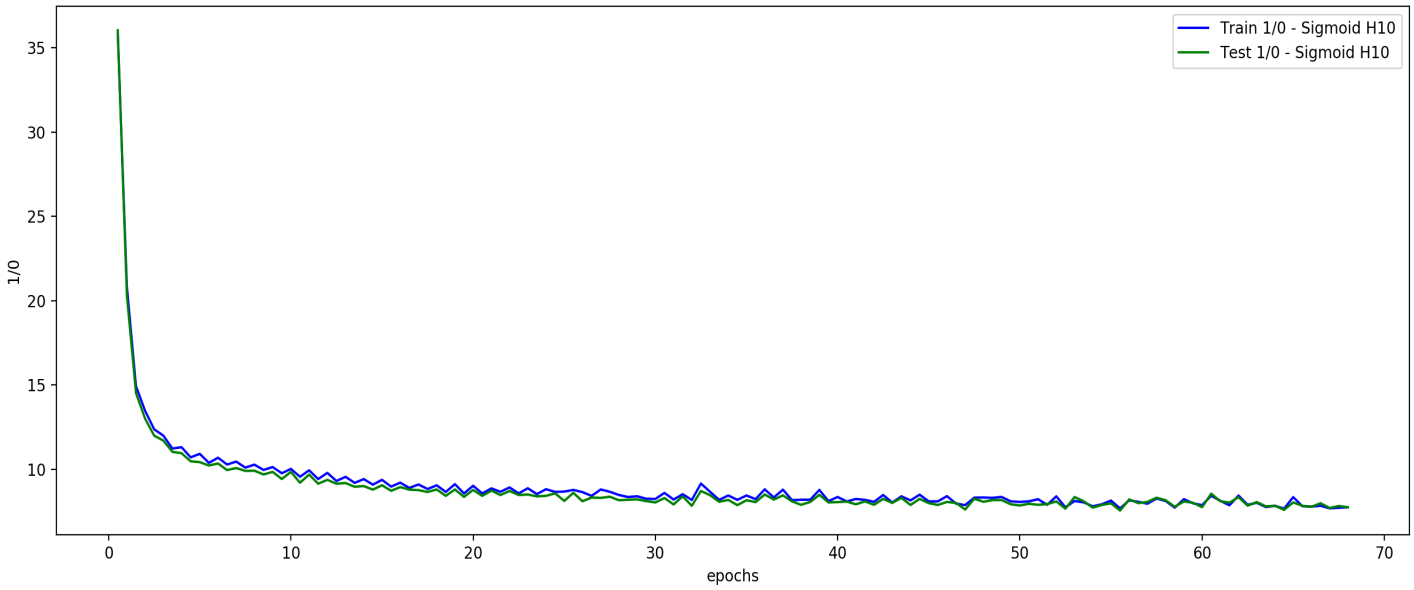

Train MSE - Sigmoid H100
Test MSE - Sigmoid H100
MSE
epochs

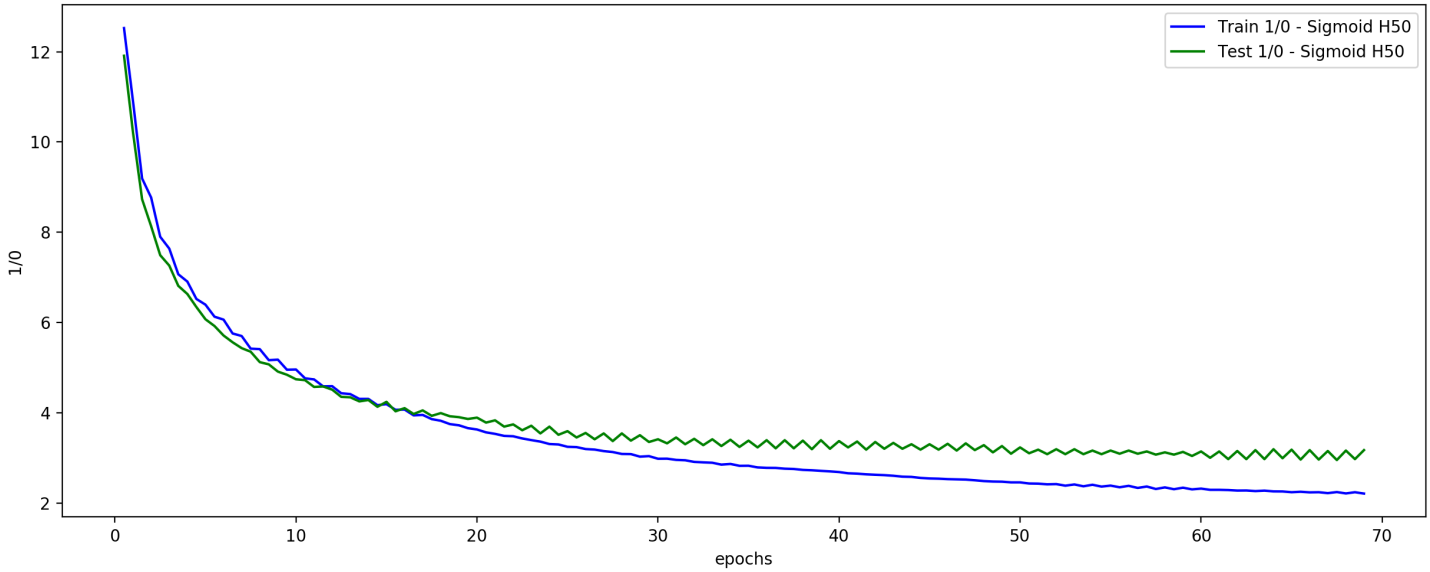

Train MSE - Sigmoid H500
Test MSE - Sigmoid H500
MSE
epochs
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
20
40
60
80
100

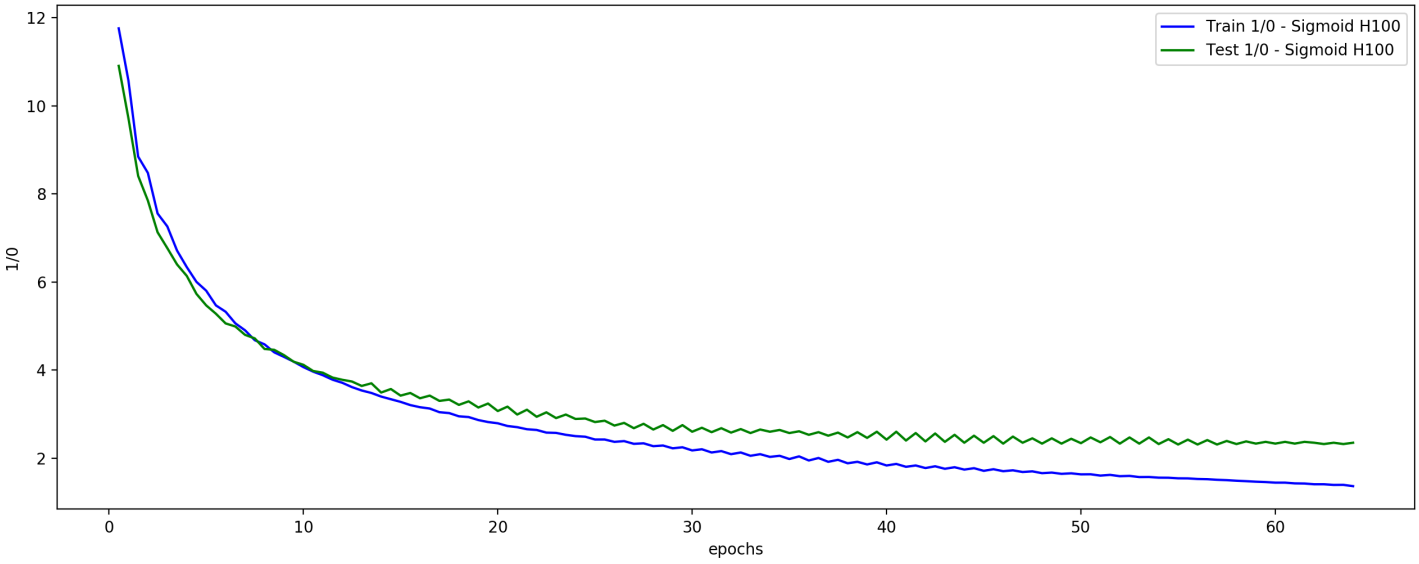

Train MSE - Sigmoid H1000
Test MSE - Sigmoid H1000
MSE
epochs
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

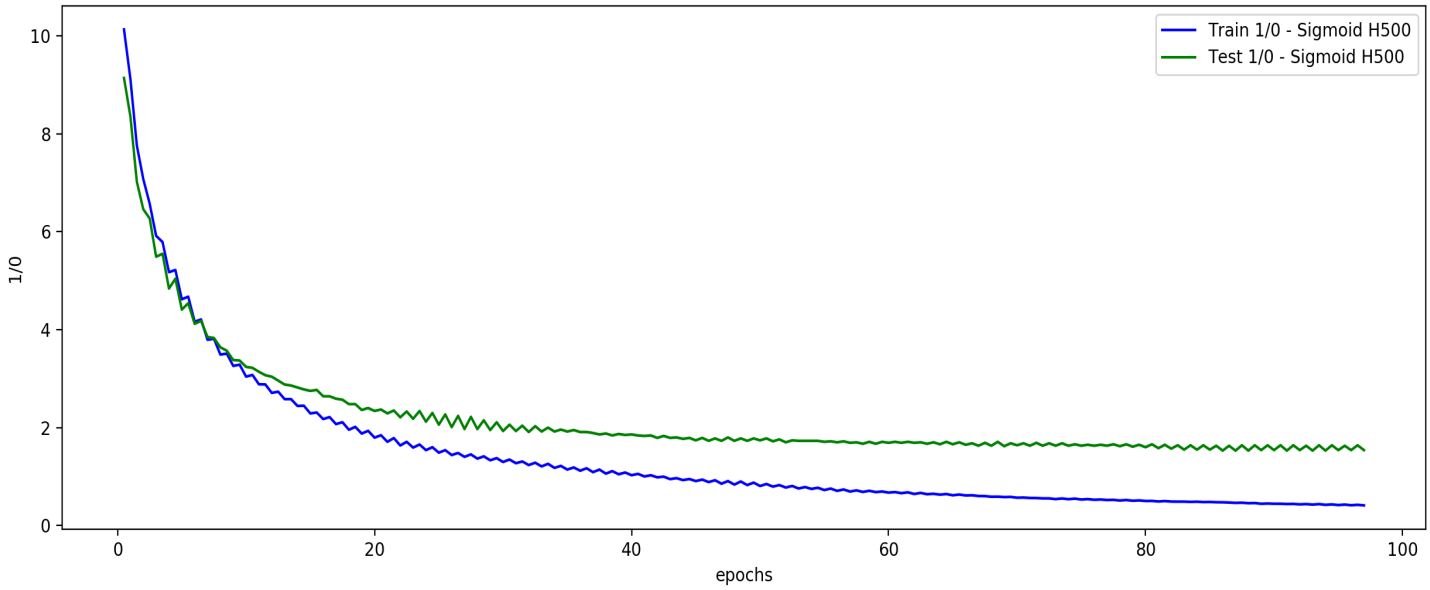

Train MSE - ReLU500
Test MSE - ReLU500
MSE
epochs
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80

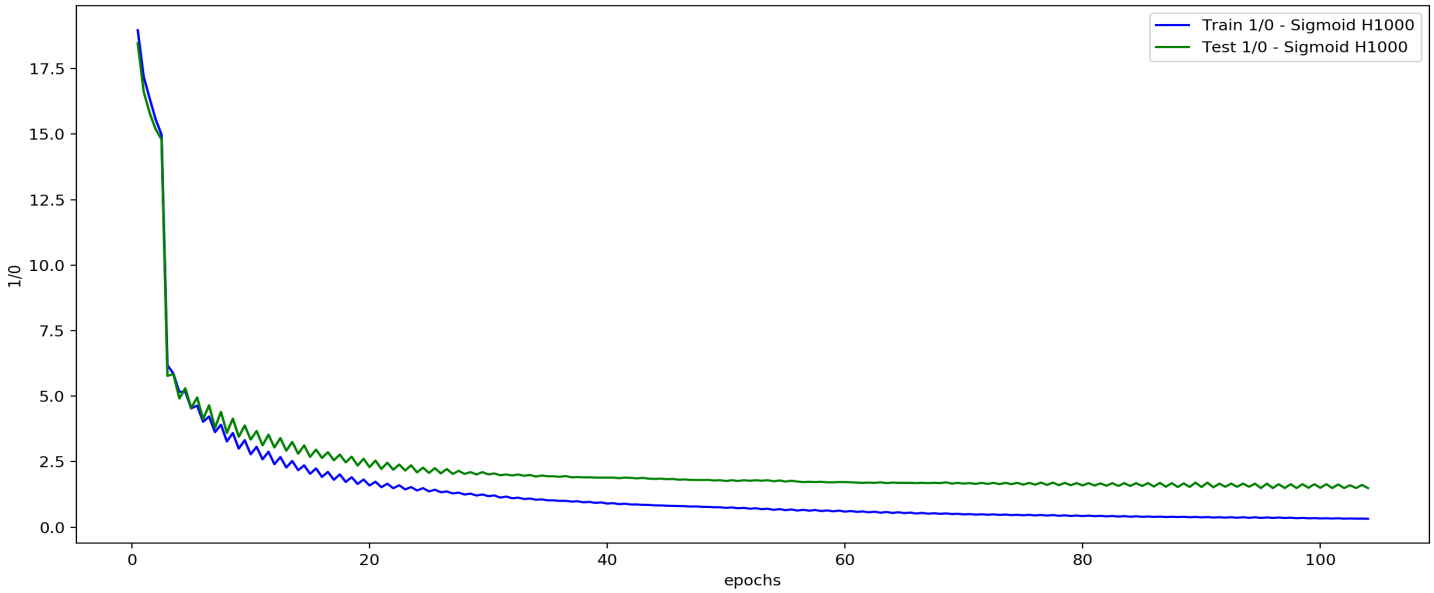
1.3:

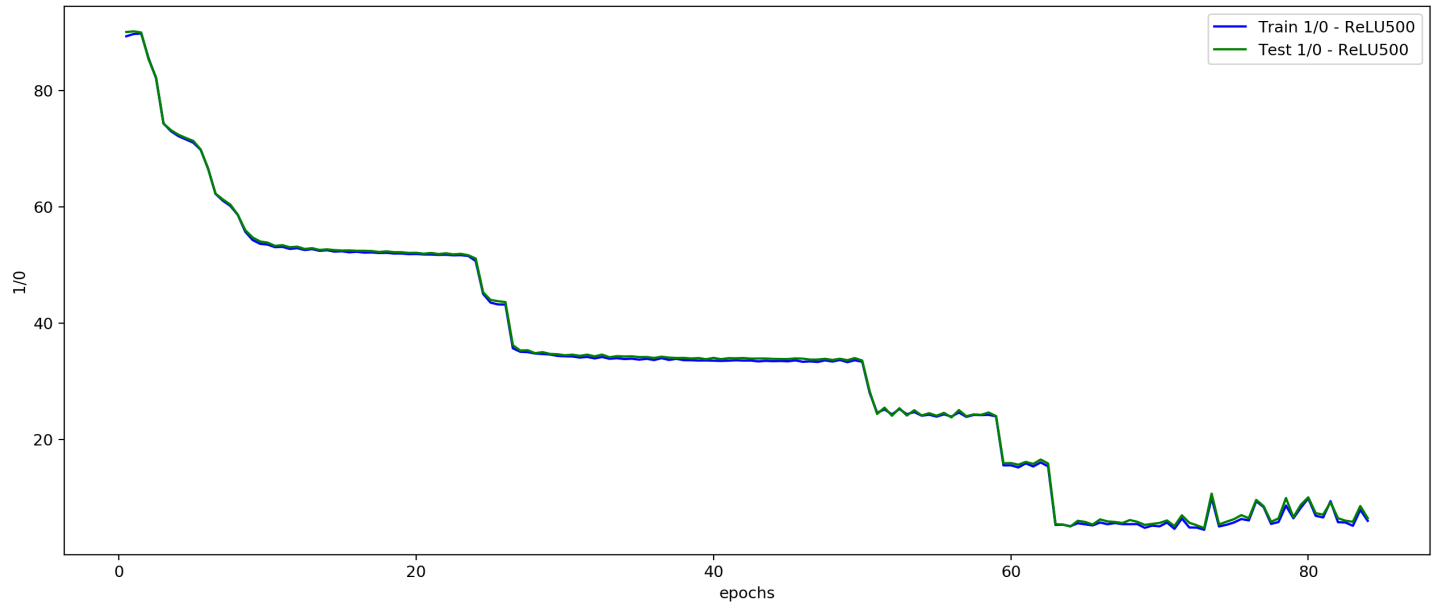

Train 1/0 - Sigmoid H10
Test 1/0 - Sigmoid H10
1/0
epochs


Train 1/0 - Sigmoid H50
Test 1/0 - Sigmoid H50
1/0
epochs


Train 1/0 - Sigmoid H100
Test 1/0 - Sigmoid H100
1/0
epochs


Train 1/0 - Sigmoid H500
Test 1/0 - Sigmoid H500
1/0
epochs


Train 1/0 - Sigmoid H1000
Test 1/0 - Sigmoid H1000
1/0
epochs


Train 1/0 - ReLU500
Test 1/0 - ReLU500
1/0
epochs

1.4: ReLu comparison with sigmoid observations:

Sigmoid	ReLU
Learning rate can be high and learns at every step	Initial learning rate should be low, else we risk dying of neurons
Gradient of the sigmoid function vanishes as we increase or decrease x	Gradient of the ReL function doesn't vanish as we increase x
Easy to tune	Hard to tune
Converges slowly	When it learns, it converges fast

Summary: ReLU can perform much better than sigmoid when we spend enough time tuning it. It also tests the patience of the programmer. As having higher learning rate with ReLU can make the neurons dead, there needs to be some learning strategy!

2.1 The ensemble learning algorithm with 'K' learned hypotheses will give an error when at least 'K/2' learning hypotheses fail. Error when exactly 'n' hypotheses fail is given by the binomial distribution formula :

$$P_{(K,n,p)} = (K\ Choose\ n)\ \ p^n\ (1-p)^{k-n}$$

Where, n = K/2, p = ε

To get the total ensemble error : we need cumulative distribution of above:

$$F_{(K,K/2,\varepsilon)} = \sum_{i=0}^{K/2} P(K,\ K/2\ +\ i,\ \varepsilon)$$

Error obtained for different values of K and ε are:

K / ε	0.1	0.2	0.4
5	0.00856	0.0579	0.3174
10	0.00163	0.0327	0.366
20	7.150×10^{-6}	0.00259	0.2446

Note: To calculate these values, I have written a python script included in the code.zip.
Usage: python3 cumulative_distribution.py K K/2 ε

2.2 If the assumption is removed, the probabilities become conditional and hence the errors of individual hypothesis become correlated. Thus, the ensemble error becomes worse than ε.
That is: If 1 of the K classifiers is at error, then it is possible that the error of ones which are correlated to it will be increased. And hence the total error of the system increases.