README for Assignment 1 CS698U

abhinav garg

January 2017

Contents

1 Code Flow

The code contains following files. Their details are explained below:-

1. **MLP_Configuration_Driver.m** :- This is the main function which contains all the confuguration parameters and drives the program
2. **train_MLP.m** :- This is the main function which learns and trains the weight vector
3. **Activation.m** :- This file contains activation function and how to apply it.
4. **drev_Activation.m** :- Derivative of activation function.
5. **testMLP.m** :- This function is used to validate on test data and plot the graphs.
6. **evaluateMLP.m** :- Used to evaluate MLP after training it.

Below is description of each file:-

1.1 MLP_Configuration_Driver.m

This function first loads MNIST data Then sets appropriate parameters.

1.1.1 Parameters that can be set/Degrees of freedom

I have implemented many more features than were asked in the assignment and hence there are a lot of variables that can be tuned to obtain optimum performance. The code is highly modular and commented.

1. bias :- weather to use bias or not. 1 means ON and 0 is OFF
2. HiddenUnits :- Array for number of hidden layers and number of nodes in each layer. Eg:- two hidden layer with 12 , 20 layer $\implies$ HiddenUnits = [12 20]
3. learningRate :- learningRate for various methods
4. momentum :- momentum values for GD with momentum
5. b1,b2,epsi :- these are parameters for Adam traning. b1 corresponds to first momentum coefficient. b2 corresponds to second momentum coefficient.
6. methodToUse :- which trainig algorithm to use. 1 $\implies$ GD with Momentum , 2 $\implies$ Adam , 3 $\implies$ Adagrad
7. a :- this parameters is to be set in activation and drev_Avtivation. a=1 $\implies$ Relu , a = 0 $\implies$ tanh.
8. batchSize :- bacth size for batch GD.
9. epochs :- number of iterations of batchSize.

1.2 train_MLP.m

First we calculate input and output diemnsions. Then we modify HiddenUnits array so that it contains informtion of all the layers.

Then we create a cell Weights ith entry of which will store weight matrix from layer i to i+1.

OldWeights is used for momentum, DeltaWeights will store derivative of loss function w.r.t each weight.

m and v are the first and second momentum vector used for Adam.

Cache is used for Adagrad

Now we do forward pass and store inputs and outputs of each node in each layer.Then apply softmax to compute final error. Then we calculate backpropagation delta for each node using concept taught in class. Now we obtain Delta weights for each weight by multiplying delta and output of nodes connecting it.

We do the above procedure for the whole batch and then update as per method to use.

We also then plot training error.

Next commented methods are for the numeric gradient. next commented method is for validation errors.

1.3 testMLP.m and EvaluateMLP.m

These methods do testing and evaluation similar to forward pass.

2 Graphs and Findings

The Implemented neural network was tested thoroughly for 2 days over variations of all possible parameters.As the data obtained was quite high, I'm not mentioning each and every plot. However, below plots should be sufficient.

2.1 Figures

3 References

1. Numeric Gradient
2. GD methods
3. Adam
4. git I have consulted some repositories whenever stuck

Figure 1: numeric gradient for layer 1

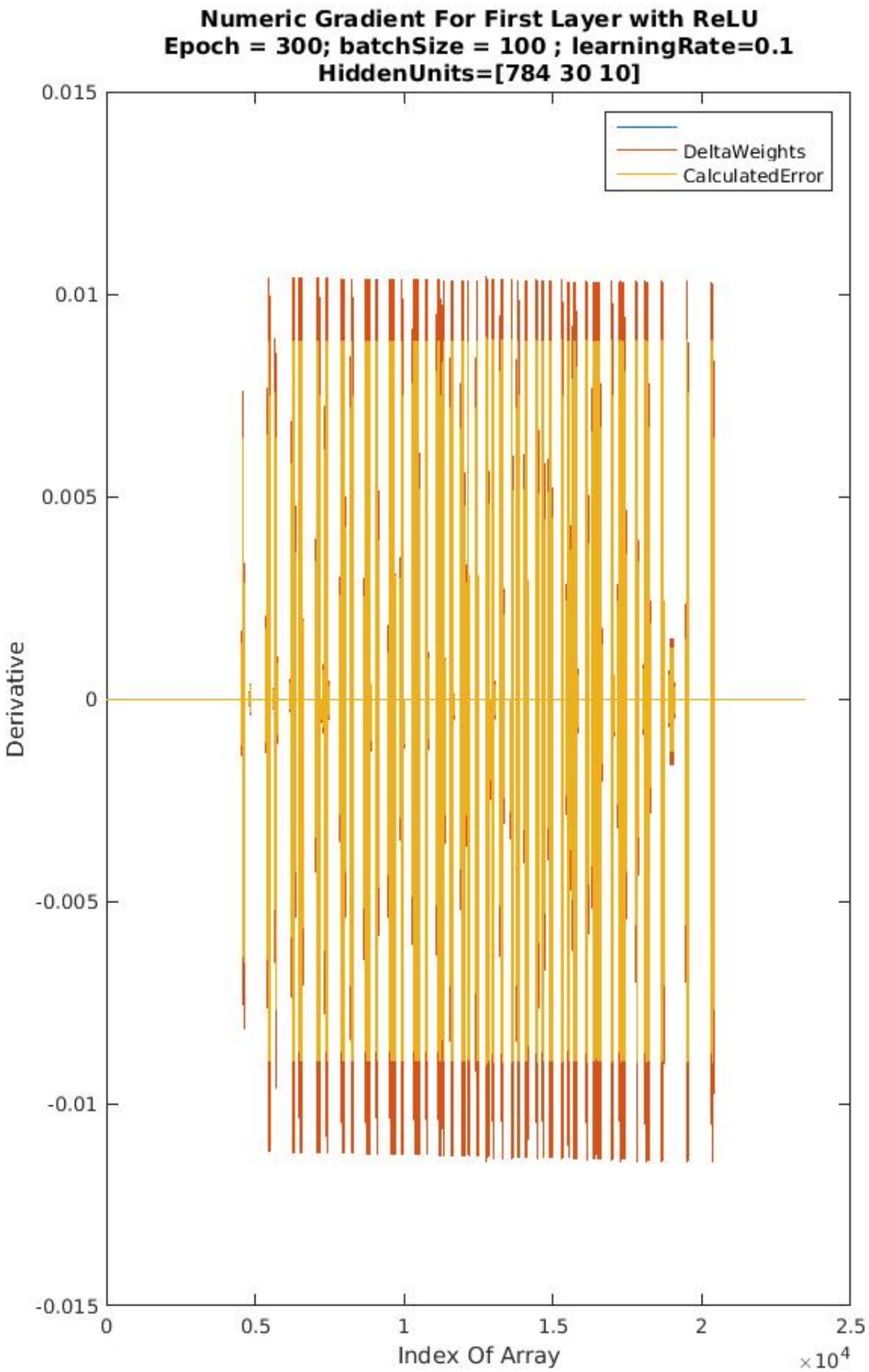

Figure 2: numeric gradient for layer 2

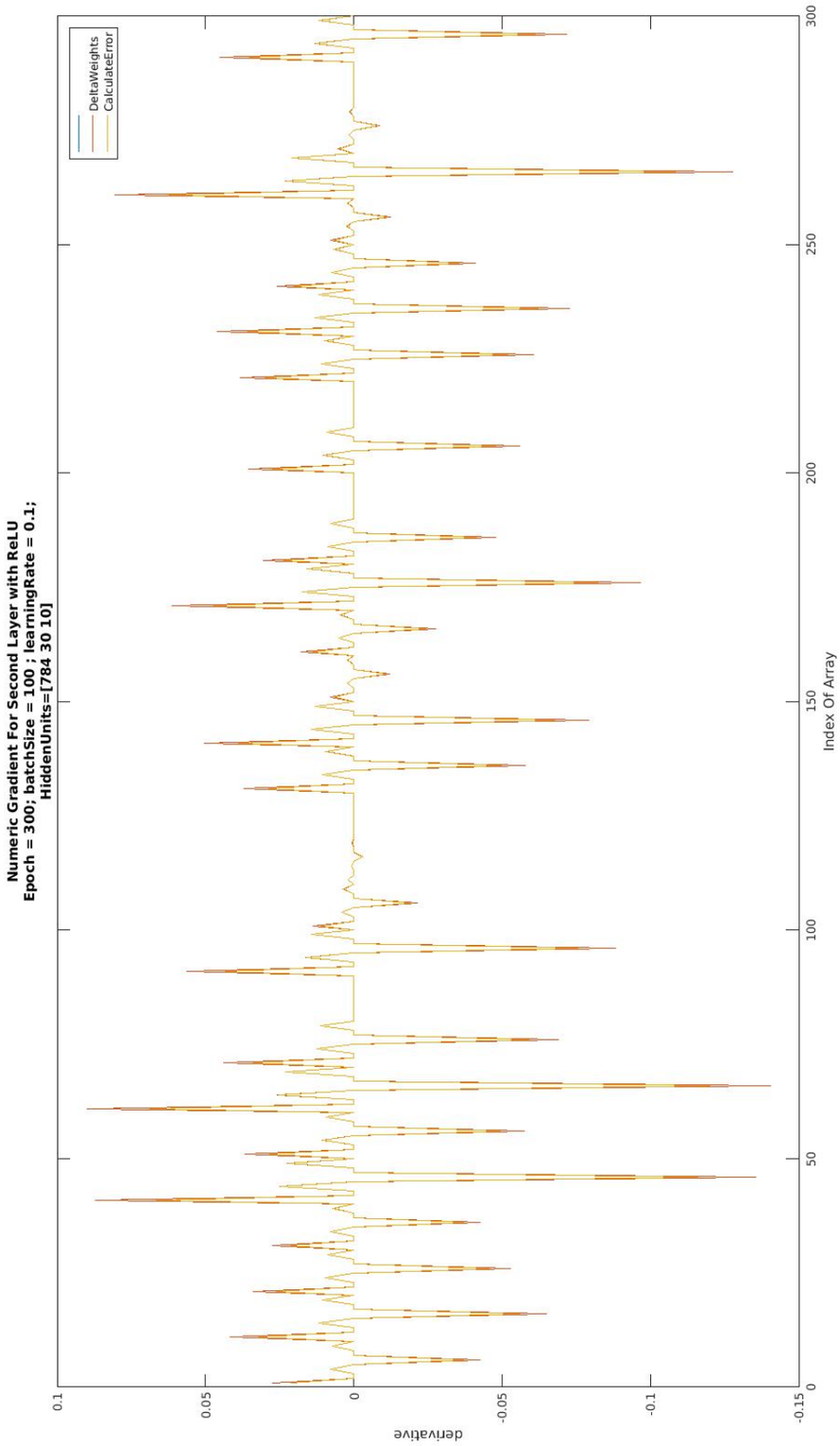

Figure 3: training accuracy for momentum

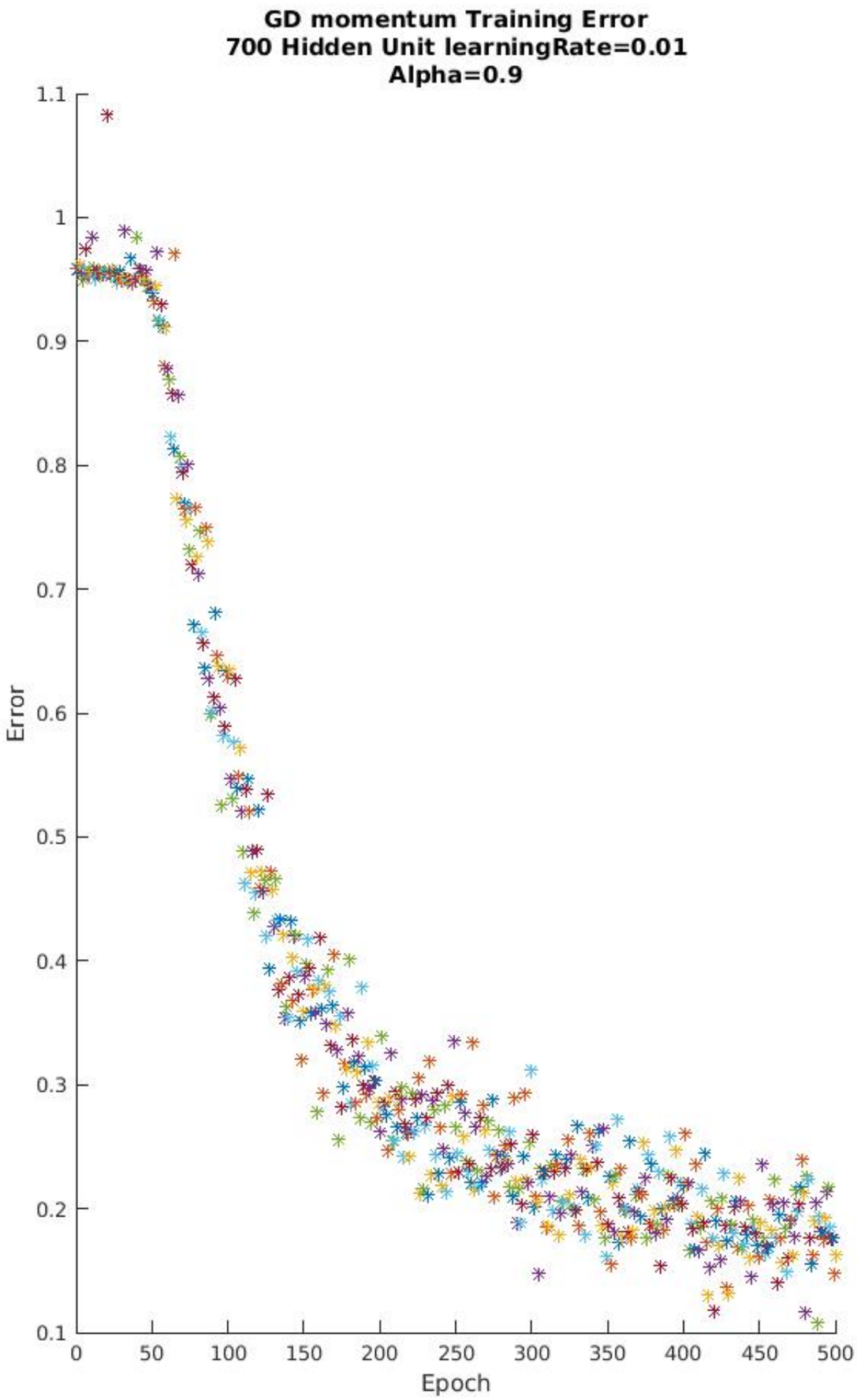

Figure 4: training accuracy for adagrad

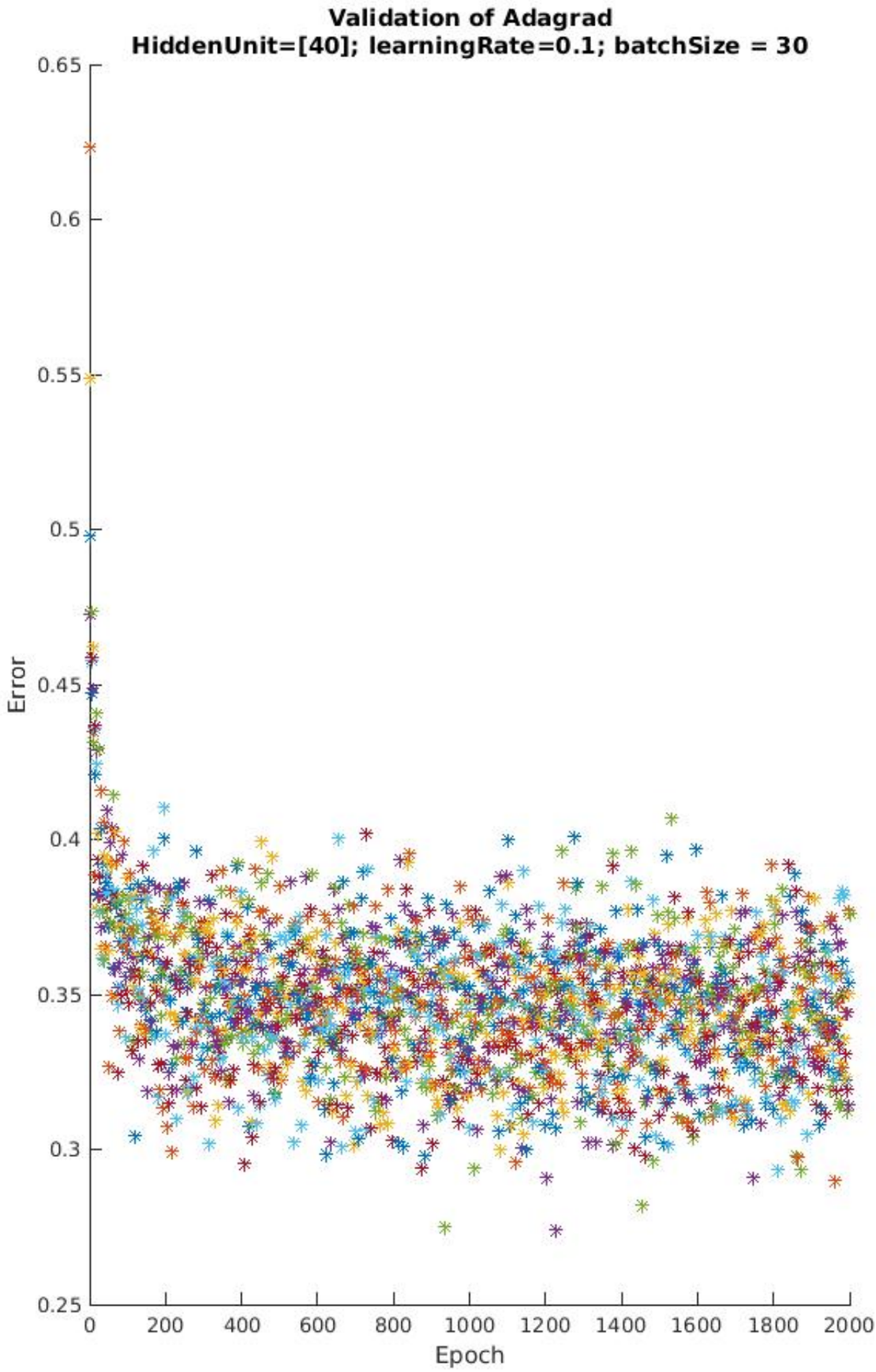

Figure 5: training accuracy for adam

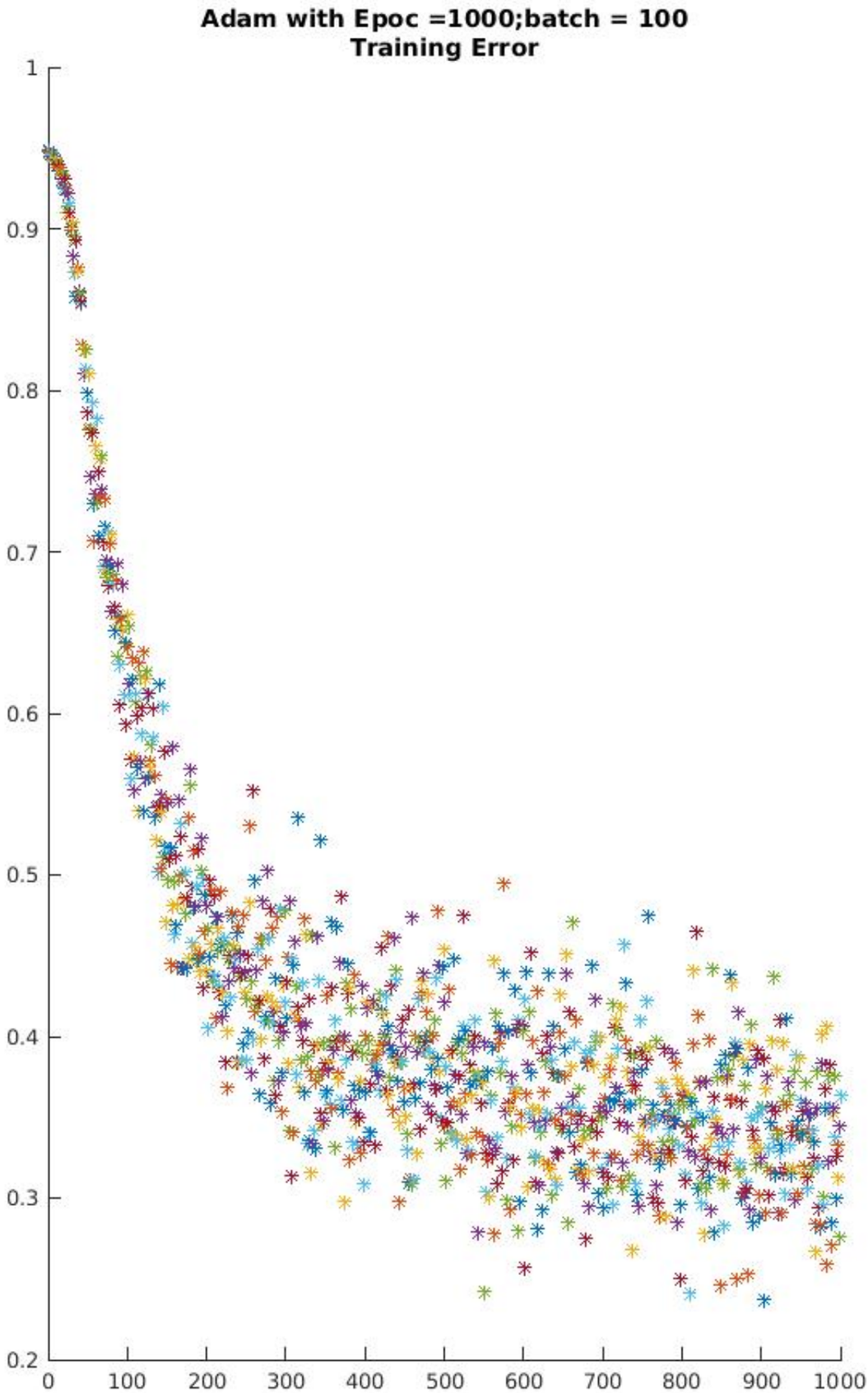

Figure 6: test accuracy for various methods

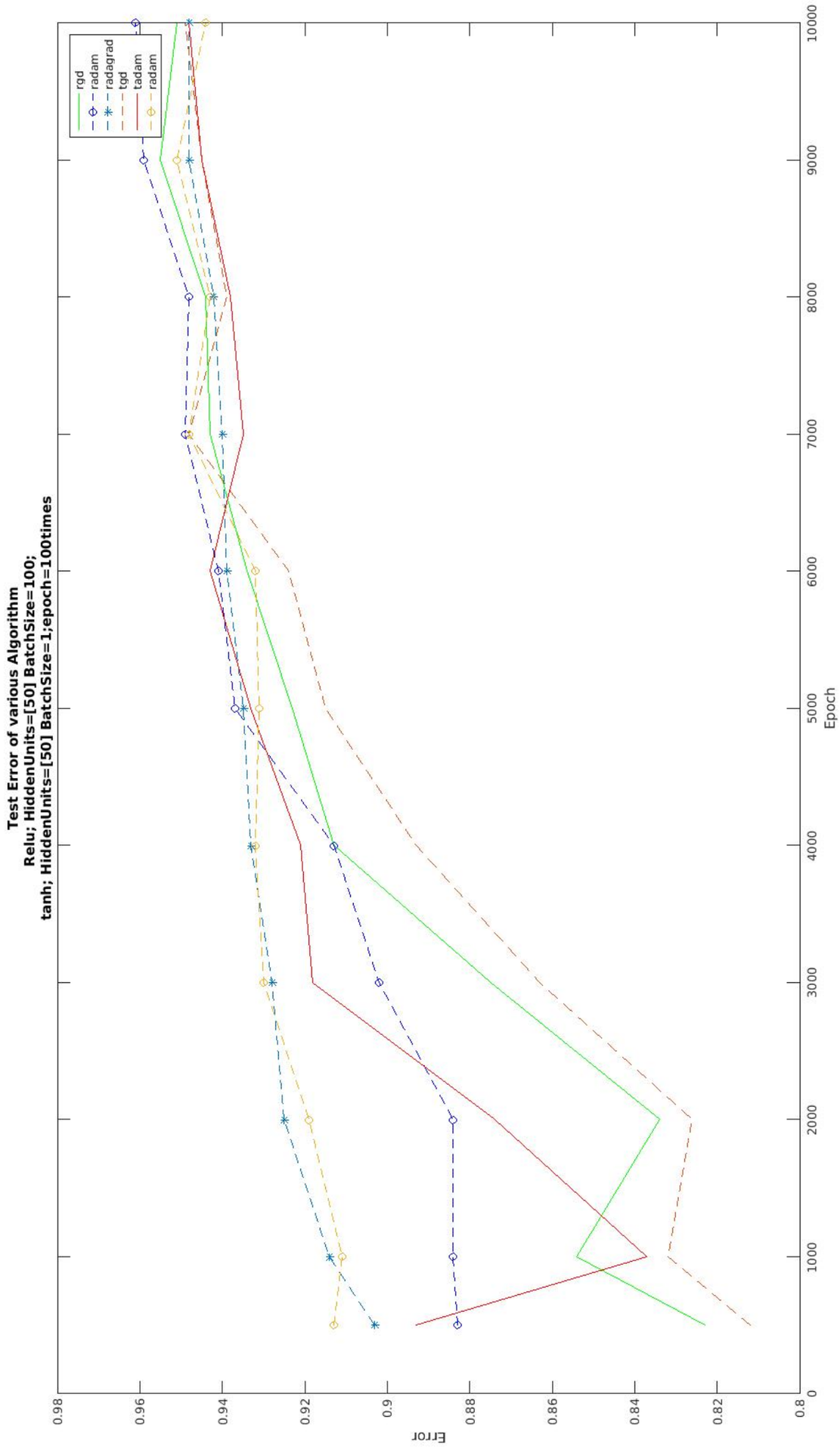

Figure 7: test accuracy for varying number of layers

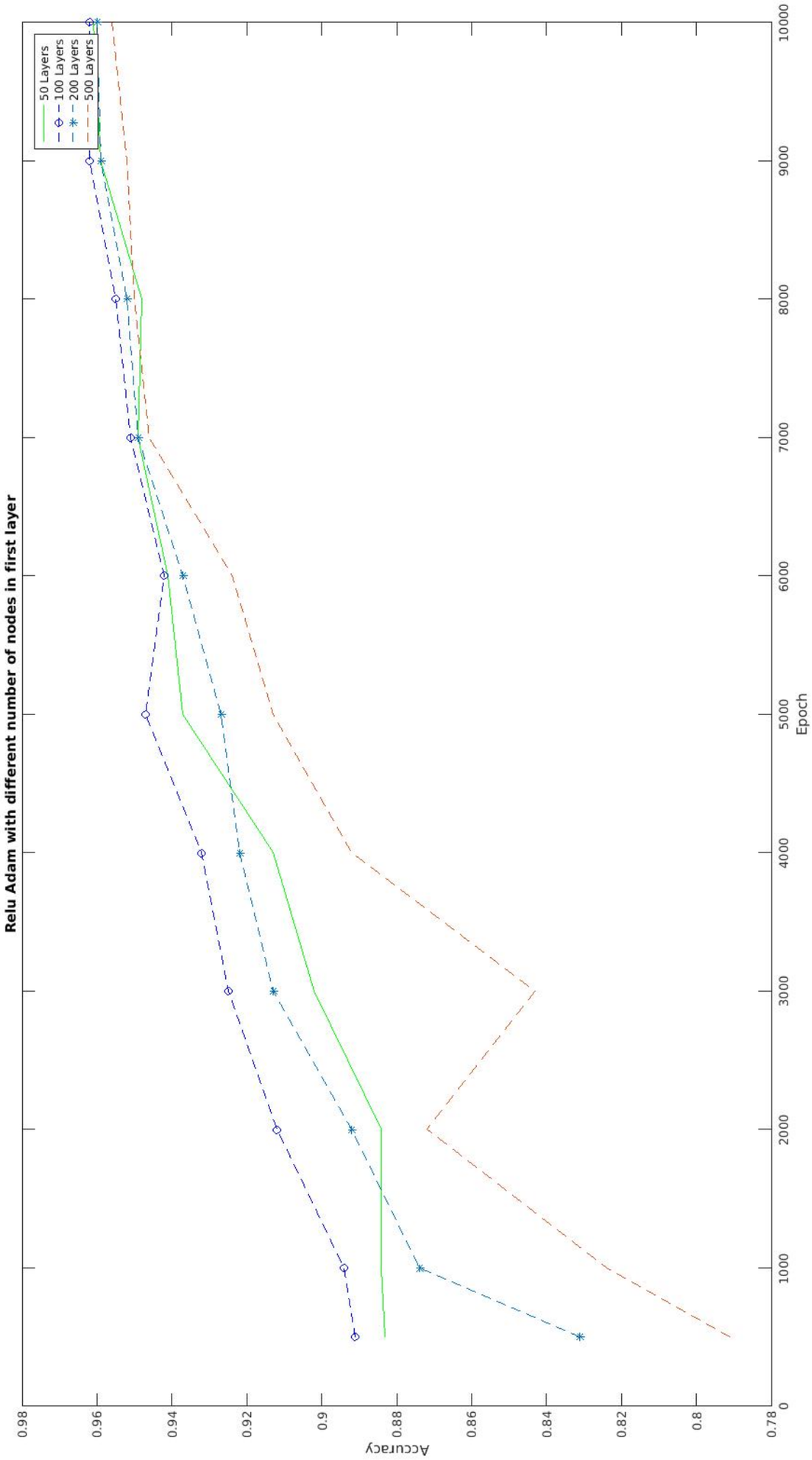

Figure 8: test accuracy for various number of hidden layers