UNIVERSITY OF
CENTRAL FLORIDA

Handwritten Character recognition

Zhendong Zhang

Outline for Section 1

Outline for Section 1.1

Introduction: Background I

Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "offline" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition.

Outline for Section 1.2

Introduction: Problem Description I

What's this problem?

- This is a supervised classification problem;
- And identification accuracy is the goals.

We decided to further investigate by following questions:

1. How to process image type data?
2. What is the best way to identifying hand written text?

Outline for Section 2

Outline for Section 2.1

Data Preparation: Data Source I

Identifying hand written text will be a hard task. To answer this question, we made use of MNIST data for Digits. [1].

Outline for Section 2.2

Data Preparation: Data Content

- The MNIST database (Modified National Institute of Standards and Technology database) of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples;
- It is a subset of a larger set available from NIST;
- The records are 28 by 28 pixel black and white images, that is 784 variables;
- All variable are introduced grayscale levels, which is a number from 0 to 255;
- There are total 10 for Digits (i.e. 0 to 9).

Here's some example of data, Fig. 1

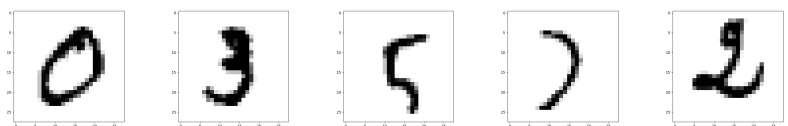


Figure 1: Example of data

Outline for Section 2.3

Data Preparation: Data transformation

- Besides, the data can't be normalized since there are some constant valued columns (some columns are always 0).
- Alternatively, We will map these values into an interval from $[0.01, 1]$ by multiplying each pixel that values p with (1).

$$p^{new} = p * \frac{0.99}{255} + 0.01 \tag{1}$$

  - It will speed the computation if we map the data to $[0.01, 1]$, rather than $[0, 1]$.

Outline for Section 3

Data visualization I

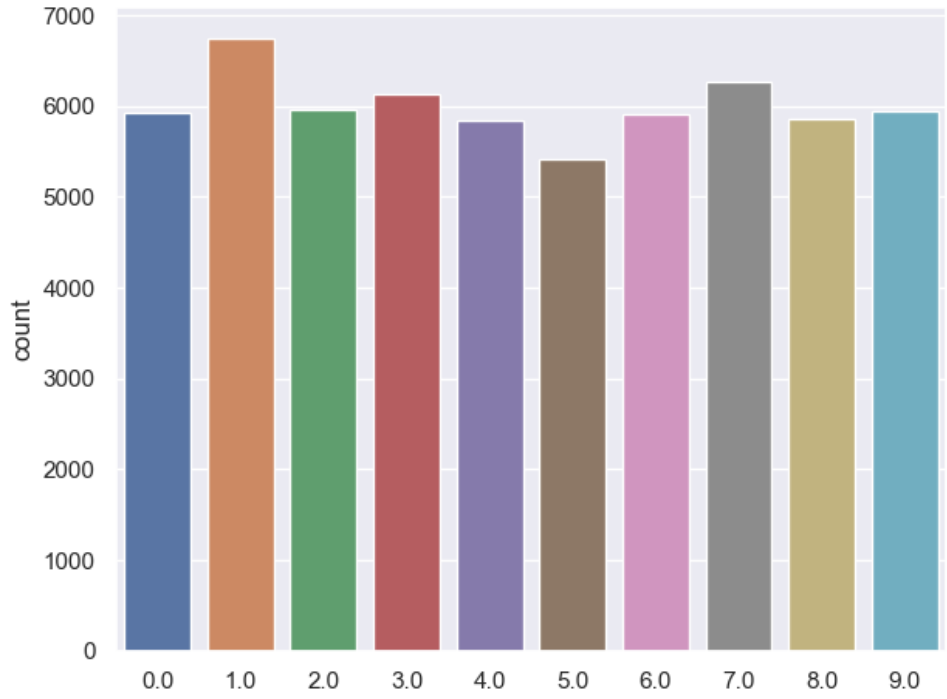


Figure 2: Distribution of MNIST data

- From Fig. 2, we can conclude that the data size of different class are basically equal, which means that our data is balanced.

Outline for Section 4

Used Language I

- We are originally use R language as our tool to model
  - by the limited computing ability, we only randomly pick 6,000 train and 1,000 test data from all numerate data from (0-9) to train our model, which is 1/10 of the MNIST data set.
  - However, even if we reduced the data size, we still find the running time is extremely long when tuning model of Neural Network in 7.
- Since Python is up to 8 times faster than R. Therefore, we only use R as our visualization programming language and Python as our model building language.

Outline for Section 5

Support Vector Machines I

- In machine learning, support-vector machines (SVMs, also support-vector networks[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.
- Since it's time consuming to use quadratic programing in SVM, so we adapted Stochastic Sub-Gradient Descent into Linear Kernel SVM.

Outline for Section 5.1

Support Vector Machines: Linear Kernel I

SGD optimization

First we use support vector machines of linear kernel.

$$K\left\langle x, x' \right\rangle = \left\langle x, x' \right\rangle$$

We added Elastic-Net regularization term (See (2) for the math formula of Elastic-Net) into our SVM model which linearly combines the L1 and L2 penalties of the lasso and ridge methods, and included two parameters:

- $l_{1_{ratio}}$: weight of L_1 regularization term;
- α: overall penalization term.

$$R(w) := \alpha\left(\frac{1 - l_{1_{ratio}}}{2}\sum_{i=1}^{n} w_i^2 + l_{1_{ratio}}\sum_{i=1}^{n}|w_i|\right) \tag{2}$$

Applying this regularization term into SVM model, and tuning these 2 parameters, the results are as Fig. 3.

Support Vector Machines: Linear Kernel II

SGD optimization

From figure, we can easily see that a smaller α tend to be better, which means that there is barely improvement on test error by using regularization term. Therefore, we don't consider regularization term in the following SVM models.

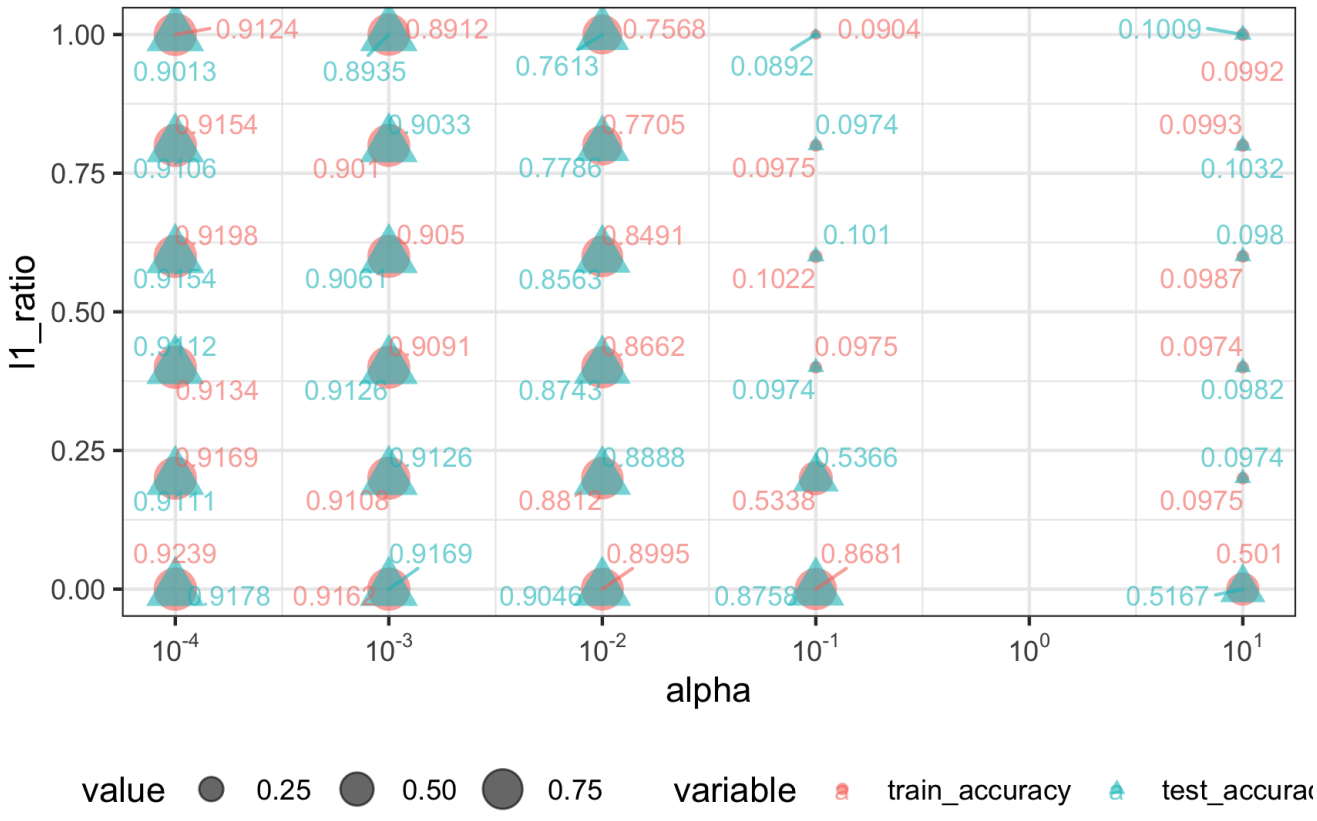


Figure 3: SGD SVM Accuracy rate, alpha versus l1 ratio

Support Vector Machines: Linear Kernel III

SGD optimization

We pick the model of best test accuracy, which is 0.9178, and the confusion table is as TABLE 1.

Table 1: SVM confusion table

	0	1	2	3	4	5	6	7	8	9	
0	953	0	1	2	0	16	3	4	1	0	0.9724
1	0	1106	3	1	1	5	4	2	13	0	0.9744
2	7	3	899	19	9	20	15	17	39	4	0.8711
3	6	0	16	899	2	50	4	11	13	9	0.8901
4	1	1	7	0	926	0	7	4	4	32	0.9430
5	6	1	0	22	12	814	14	4	14	5	0.9126
6	11	3	3	2	5	30	901	1	2	0	0.9405
7	1	5	19	4	4	4	1	969	2	19	0.9426
8	9	7	4	24	24	83	11	18	784	10	0.8049
9	8	4	0	10	47	22	1	38	9	870	0.8622
	0.9511	0.9788	0.9443	0.9145	0.8990	0.7797	0.9376	0.9073	0.8899	0.9168	0.9178

Outline for Section 5.2

Support Vector Machines: RBF Kernel I

Tuning with 20% data set

In order to get a non-linear boundary, we use RBF kernel to train model.

$$K\left\langle x, x' \right\rangle = \exp\left(-\gamma \left\| x - x' \right\|^2\right)$$

Considering the computation complexity of RBF Kernel, we only use 20% data set to tune our model. The hyper-parameter field are as follows.

- γ : $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$.

Support Vector Machines: RBF Kernel II

Tuning with 20% data set

It takes total 13,578.077 seconds to train those 4 model with 4-core CPU. And we gain best model of $\gamma = 0.01$ with a test accuracy of 0.9375, training this model takes 2,749.068 seconds.

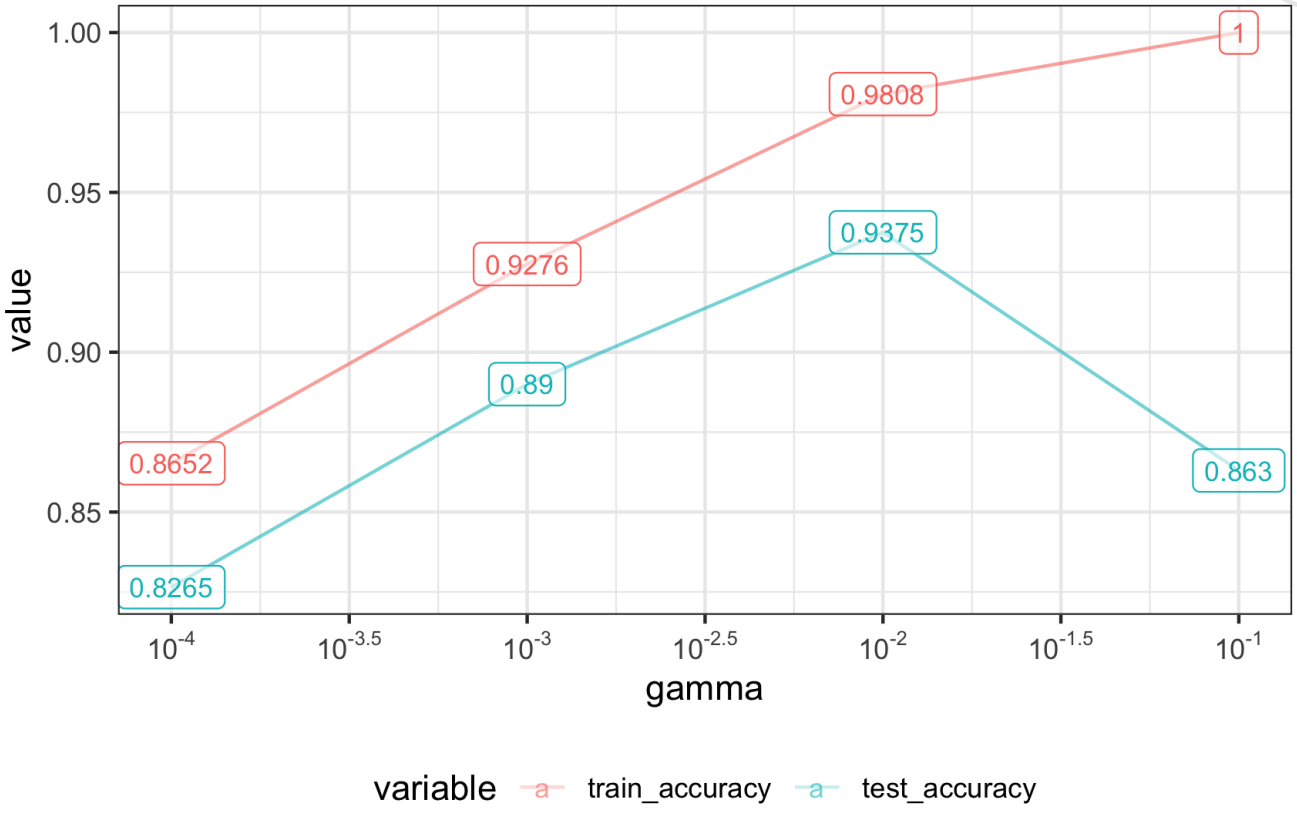


Figure 4: RBF SVM Accuracy versus gamma

Support Vector Machines: RBF Kernel I

Train best model with whole data set

Applying best model of $\gamma = 0.01$ to whole data set. It takes total 2,749 seconds to train this model with 4-core CPU. We get a train accuracy of 0.9983 and test accuracy of 0.9852, and the confusion table is as TABLE 2.

Table 2: RBF SVM confusion table

	0	1	2	3	4	5	6	7	8	9	
0	1014	0	2	0	0	2	2	0	1	3	0.9902
1	0	1177	2	1	1	0	1	0	2	1	0.9932
2	2	2	1037	2	0	0	0	2	5	1	0.9867
3	0	0	3	1035	0	5	0	6	6	2	0.9792
4	0	0	1	0	957	0	1	2	0	3	0.9927
5	1	1	0	4	1	947	4	0	5	1	0.9824
6	2	0	1	0	2	0	1076	0	4	0	0.9917
7	1	1	8	1	1	0	0	1110	2	4	0.9840
8	0	4	2	4	1	6	0	1	1018	1	0.9817
9	3	1	0	7	5	2	0	4	9	974	0.9692
	0.9912	0.9924	0.9820	0.9820	0.9886	0.9844	0.9926	0.9867	0.9677	0.9838	0.9852

Outline for Section 5.3

Support Vector Machines: Polynomial Kernel I

Tuning with 20% data set

We now use Polynomial kernel to train model.

$$K\left\langle x, x' \right\rangle = \left(\gamma \left\langle x, x' \right\rangle\right)^d$$

- Considering the computation complexity of Polynomial Kernel, we only use 20% data set to tune our model.
- The hyper-parameter field are as follows.
  - $\gamma : 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$;
  - $d = 1, 2, 3, 4, 5$.

Support Vector Machines: Polynomial Kernel II

Tuning with 20% data set

It takes total 4,916.98 seconds to tune the model with 4-core CPU. And we gain best model of $\gamma = 0.1, d = 2$. We get a test accuracy of 0.9535.

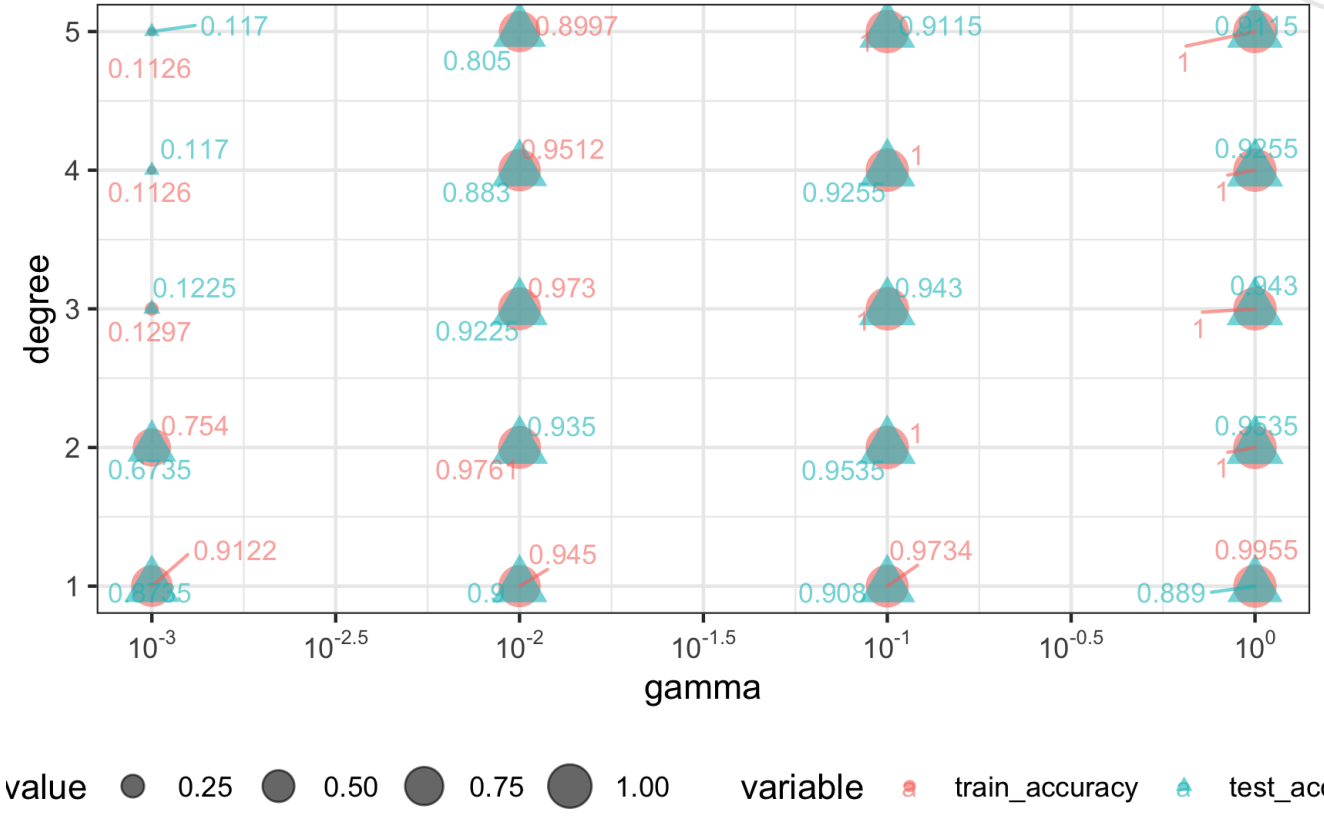


Figure 5: Polynomial SVM Accuracy degree versus gamma

Support Vector Machines: Polynomial Kernel I

Train best model with whole data set

Applying best model of $\gamma = 0.1, d = 2$ to whole data set. It takes total 1,439.43 seconds to train this model with 4-core CPU. We get a train accuracy of 0.9989 and a test accuracy of 0.9806, and the confusion table is as TABLE 3.

Table 3: Polynomial SVM confusion table

	0	1	2	3	4	5	6	7	8	9	
0	973	0	1	2	0	1	1	0	2	0	0.9929
1	0	1128	2	1	0	1	1	1	1	0	0.9938
2	7	1	1008	1	1	0	4	6	4	0	0.9767
3	0	0	3	987	0	5	0	5	7	3	0.9772
4	1	0	4	0	966	0	2	0	0	9	0.9837
5	2	0	0	9	1	872	3	1	2	2	0.9776
6	5	2	2	0	2	5	940	0	2	0	0.9812
7	0	6	9	1	1	0	0	1002	1	8	0.9747
8	4	0	2	4	3	2	1	4	951	3	0.9764
9	2	4	0	4	9	4	0	4	3	979	0.9703
	0.9789	0.9886	0.9777	0.9782	0.9827	0.9798	0.9874	0.9795	0.9774	0.9751	0.9806

Outline for Section 6

Outline for Section 6.1

Boosting: Ada Boosting I

Tuning with 20% data set

It takes total 1,044 seconds to tune the model with 4-core CPU. And we gain best model of learning rate = 0.1.

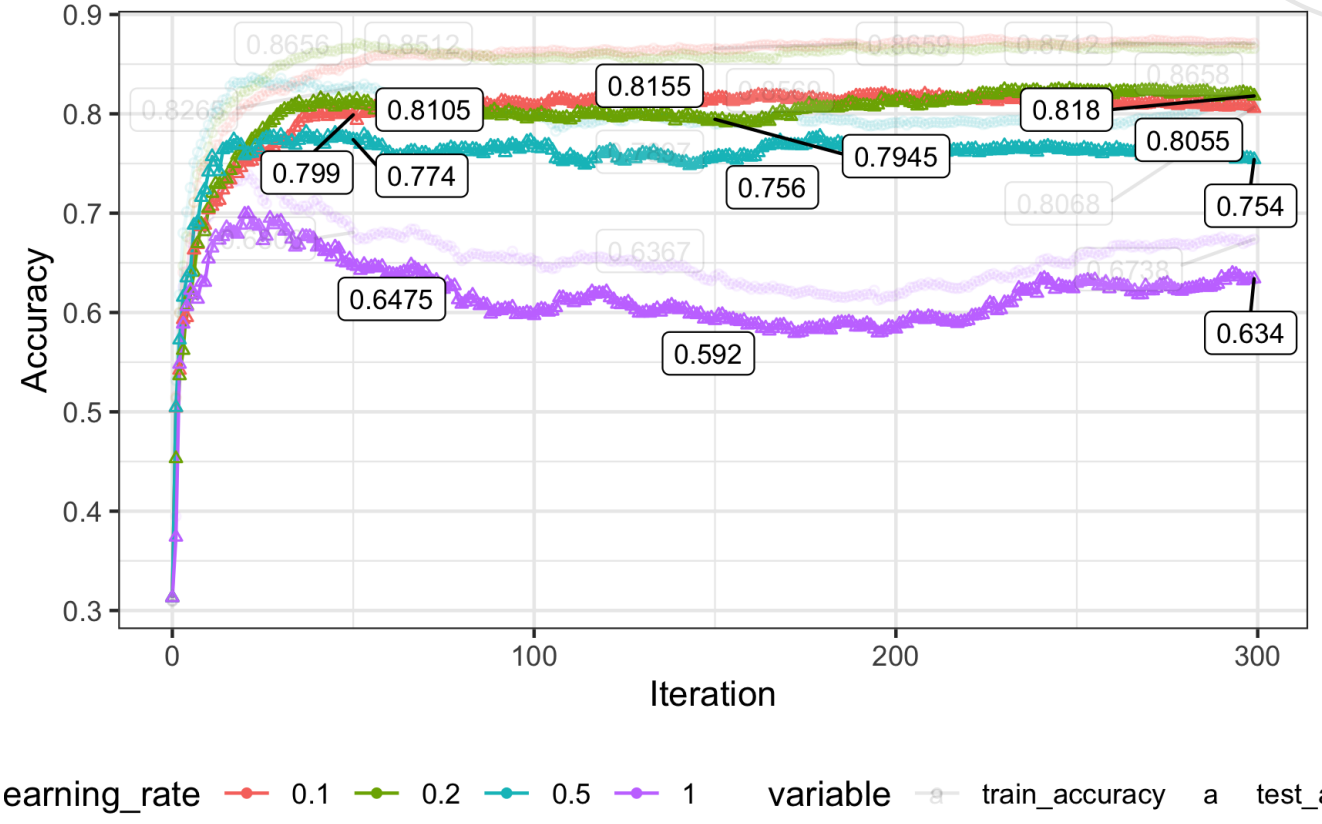


Figure 6: Ada Boosting Accuracy By Iteration

Boosting: Ada Boosting I

Train best model with whole data set

- Now we apply the best model of learning rate $= 0.1$ into whole data
- it takes 1402 seconds to train the model
- The test accuracy is 0.8902, train accuracy is 0.8911, and the confusion table is as TABLE 4.
- However from Fig. 7, we can see that there are barely differences between the test and train accuracy in each iteration, which means that our model is under-fitted. Try to increase the iteration number and decrease the learning rate might increase accuracy.

Boosting: Ada Boosting II

Train best model with whole data set

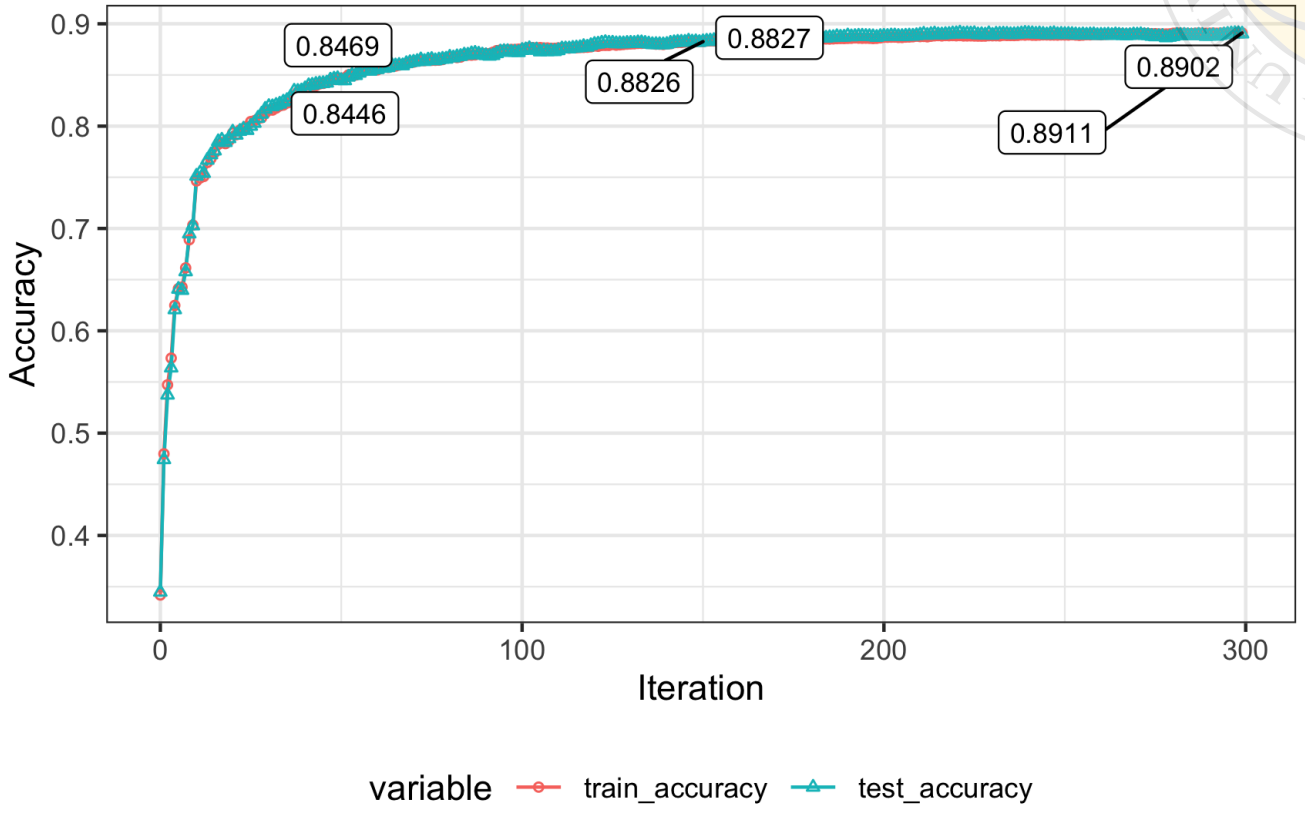


Figure 7: Final Ada Boosting Accuracy By Iteration

Boosting: Ada Boosting III

Train best model with whole data set

Table 4: Ada Boosting confusion table

	0	1	2	3	4	5	6	7	8	9	
0	902	0	13	0	0	45	13	0	4	3	0.9204
1	0	1114	5	3	0	1	1	1	8	2	0.9815
2	12	14	852	16	8	5	38	9	76	2	0.8256
3	20	0	13	894	0	30	0	7	43	3	0.8851
4	1	0	6	0	902	1	2	1	8	61	0.9185
5	16	3	3	46	3	754	13	1	38	15	0.8453
6	10	3	44	0	31	29	833	0	8	0	0.8695
7	0	3	27	4	6	0	0	889	8	91	0.8648
8	12	10	3	18	8	11	4	3	891	14	0.9148
9	4	8	7	12	56	2	0	12	24	884	0.8761
	0.9232	0.9645	0.8756	0.9003	0.8895	0.8588	0.9215	0.9632	0.8042	0.8223	0.8911

Outline for Section 6.2

Boosting: Gradient Boosting I

Tuning with 20% data set

- Even though we only training model with 20% data, Gradient Boosting still need an average of 15 minutes to build one model, which is significantly slower than ada boosting method.
- It takes 12,129 seconds to tune parameters.

Boosting: Gradient Boosting II

Tuning with 20% data set

From Fig. 8, we can see that with the increasing of tree depth, the accuracies are going up. The best hyper parameters are learning rate = 0.2, tree depth = 4. [1]

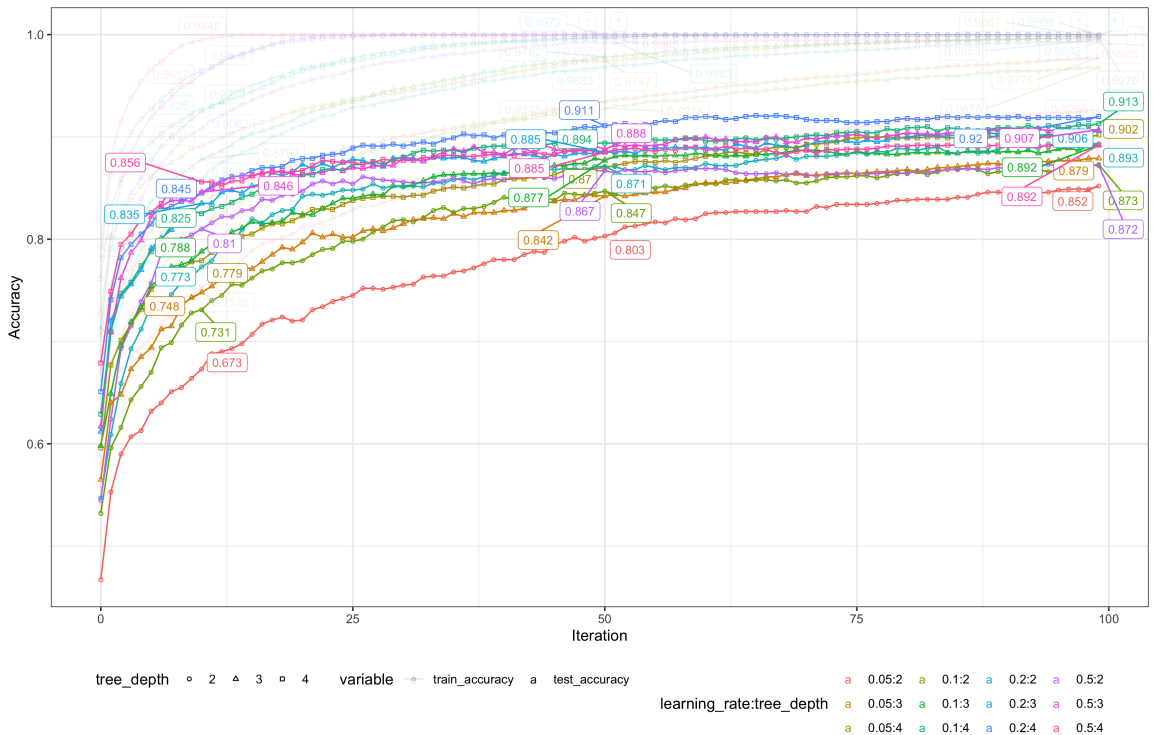


Figure 8: Gradient Boosting Accuracy By Iteration

[1] Since the size of tree depth we choose is the upper bound of our tuning field, therefore, we might gain a better result by using a tree depth that is more than 4.

Boosting: Gradient Boosting I

Train best model with whole data set

Now we train the model of learning rate $= 0.2$, tree depth $= 4$ with whole data set. It takes 7,321 seconds to train the model, the test accuracy is 0.9673, train accuracy is 0.9981, and the confusion table is as TABLE 5.

Table 5: Gradient Boosting confusion table

	0	1	2	3	4	5	6	7	8	9	
0	962	0	0	2	0	7	4	2	3	0	0.9816
1	0	1121	2	1	1	1	4	1	4	0	0.9877
2	5	2	995	7	3	2	1	8	8	1	0.9641
3	1	1	6	960	1	18	0	8	9	6	0.9505
4	1	0	1	0	960	1	5	0	3	11	0.9776
5	2	1	0	1	2	871	5	1	6	3	0.9765
6	7	3	1	0	4	14	922	1	6	0	0.9624
7	1	7	10	5	3	1	0	985	1	15	0.9582
8	2	1	3	4	5	7	2	6	940	4	0.9651
9	3	7	1	7	10	8	1	7	8	957	0.9485
	0.9776	0.9808	0.9764	0.9726	0.9707	0.9366	0.9767	0.9666	0.9514	0.9599	0.9673

Outline for Section 7

Neutral Network I

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. We use batch[2] and epoch[3] to update our parameters [2].

[2] Batch: a set of N samples. The samples in a batch are processed independently, in parallel. If training, a batch results in only one update to the model. A batch generally approximates the distribution of the input data better than a single input.

[3] Epoch: an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation.

Outline for Section 7.1

Neutral Network: 1-layer Neutral Network I

Softmax activation function

Softmax is a basic activation function because it not only maps our output to a [0,1] range but also maps each output in such a way that the total sum is 1.

$$\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \quad \text{for } j = 1, \ldots, K$$

We trained 216 1-layer Neutral Network models with different hidden nodes, learning rates and bias, for every type of model we trained 10 epochs. The hyper-parameter field are as follows.

- Bias: 1, None;
- Number of hidden nodes: 20, 50, 100, 150, 250, 500;
- batch size: 100, 200, 300;
- loss function: mean squared error, categorical cross entropy;
- Learning rate: 0.001, 0.01, 0.1.

Neutral Network: 1-layer Neutral Network II

Softmax activation function

Firstly, we need to know if all our model is converged. From Fig. 9, we get 9 models that are not converged. Therefore we need to add more epoch to those unconverged model until them converged, and then use the converged model's test accuracy as their final accuracy.

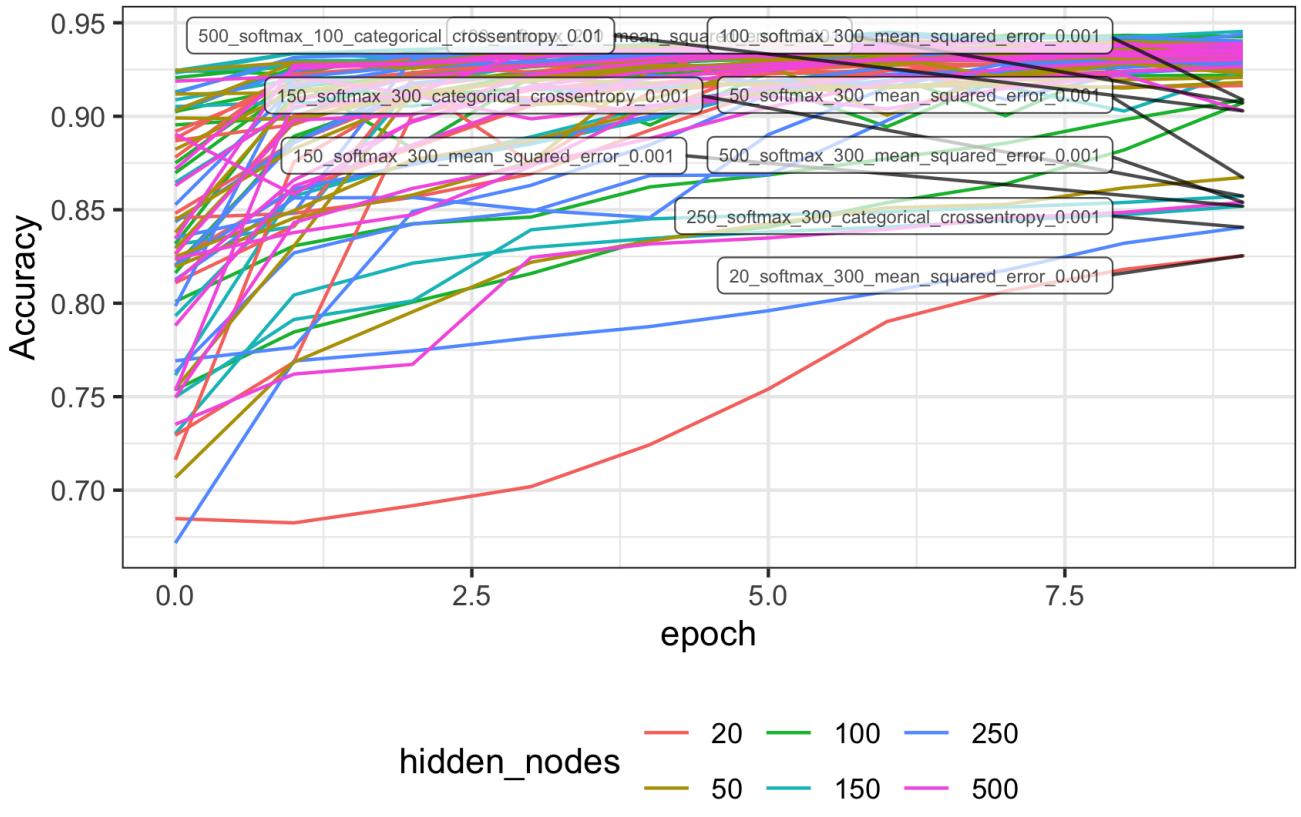


Figure 9: 1-layer Neural Network Test Accuracy By Epoch

Neutral Network: 1-layer Neutral Network III

Softmax activation function

Next, we tune all the hyper-parameters. From Fig. 10, we can see that bias barely improves accuracy.

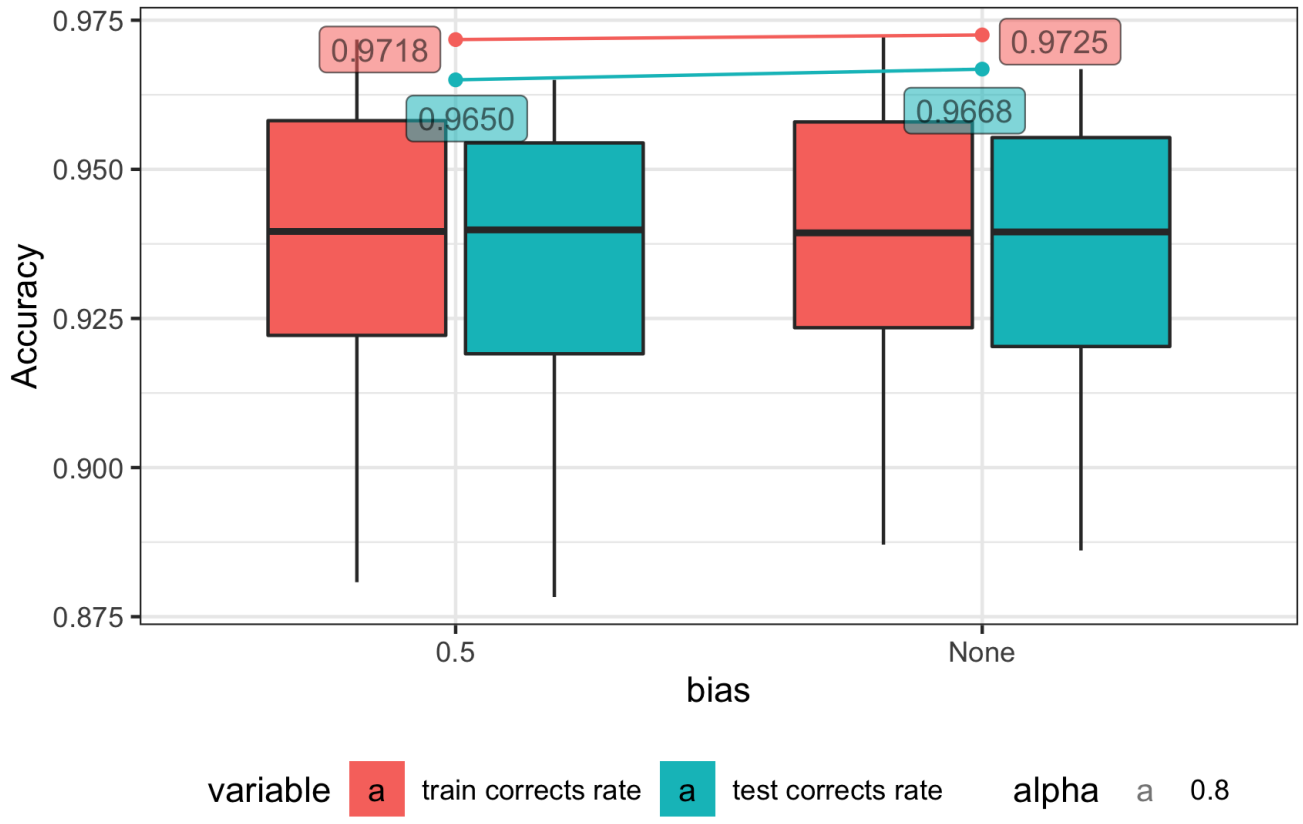


Figure 10: 1-Hidden Layer NN Accuracy rate versus bias(Softmax)

Neutral Network: 1-layer Neutral Network IV

Softmax activation function

From Fig. 11, we can see that a lager number of hidden nodes tends to have better accuracy. However, the best test accuracy is when number of hidden nodes equals 150.

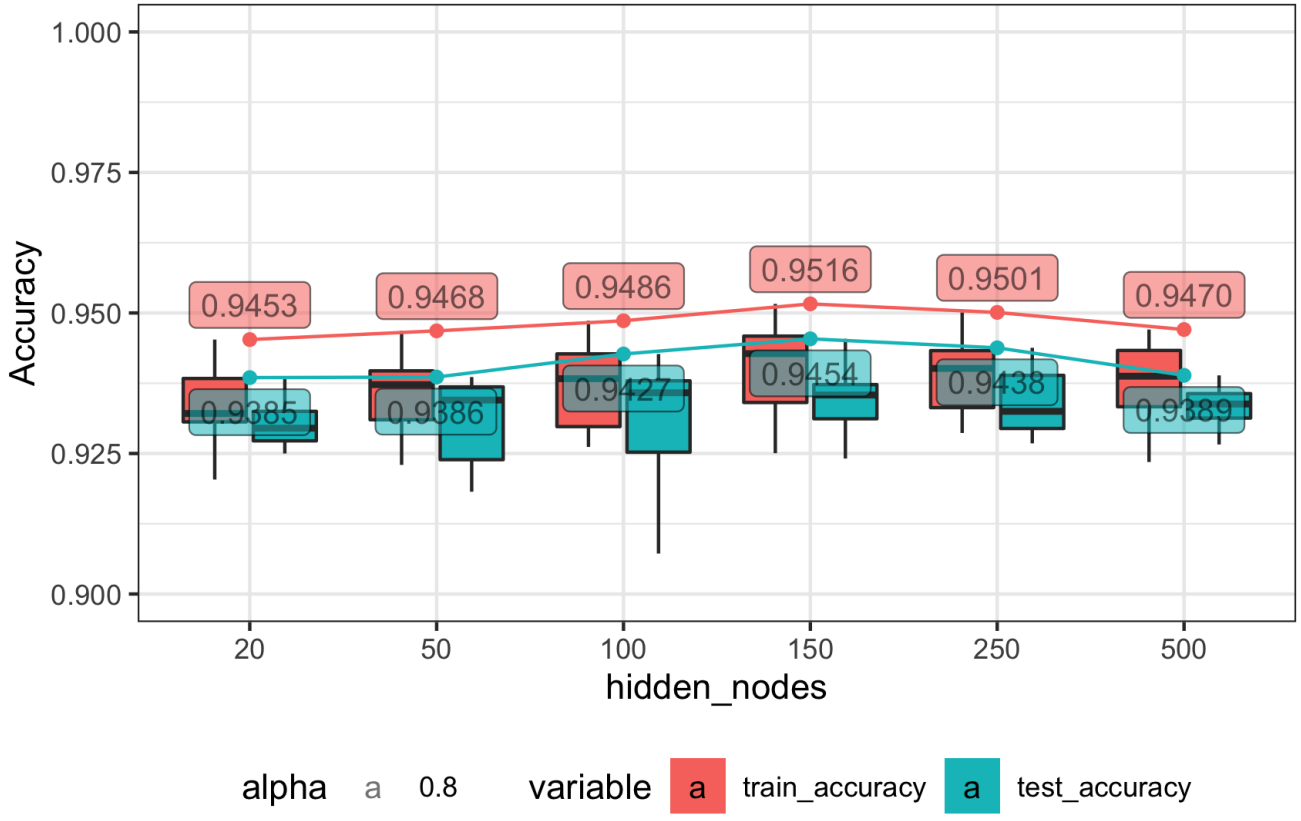


Figure 11: 1-Hidden Layer NN Accuracy rate versus hidden nodes(Softmax)

Neutral Network: 1-layer Neutral Network V

Softmax activation function

From Fig. 12[4], we can see that a learning rate of 0.01 tends to have better accuracy than 0.001.

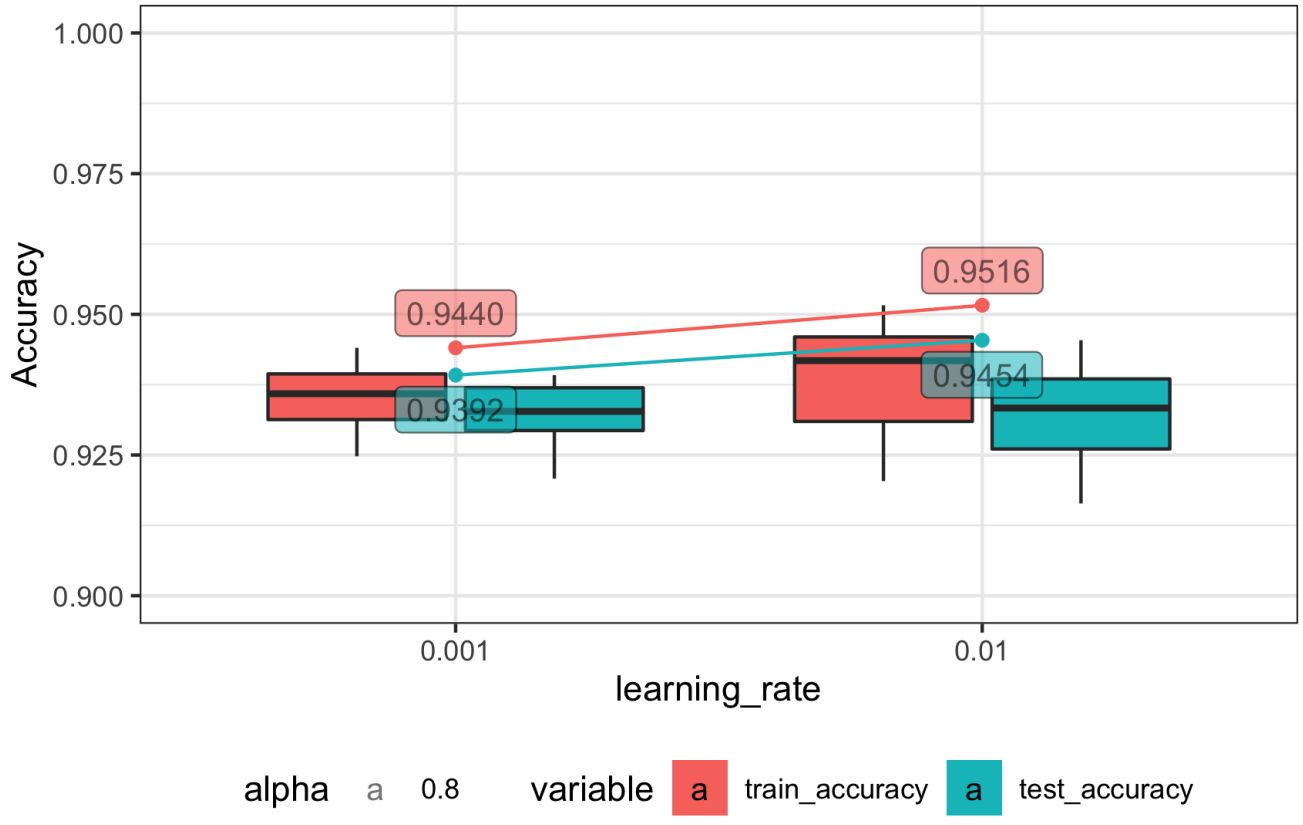


Figure 12: 1-Hidden Layer NN Accuracy rate versus learning rate(Softmax)

Neutral Network: 1-layer Neutral Network VI

Softmax activation function

From Fig. 13, we can see that a the use of different loss function don't impact the test accuracy a lot.

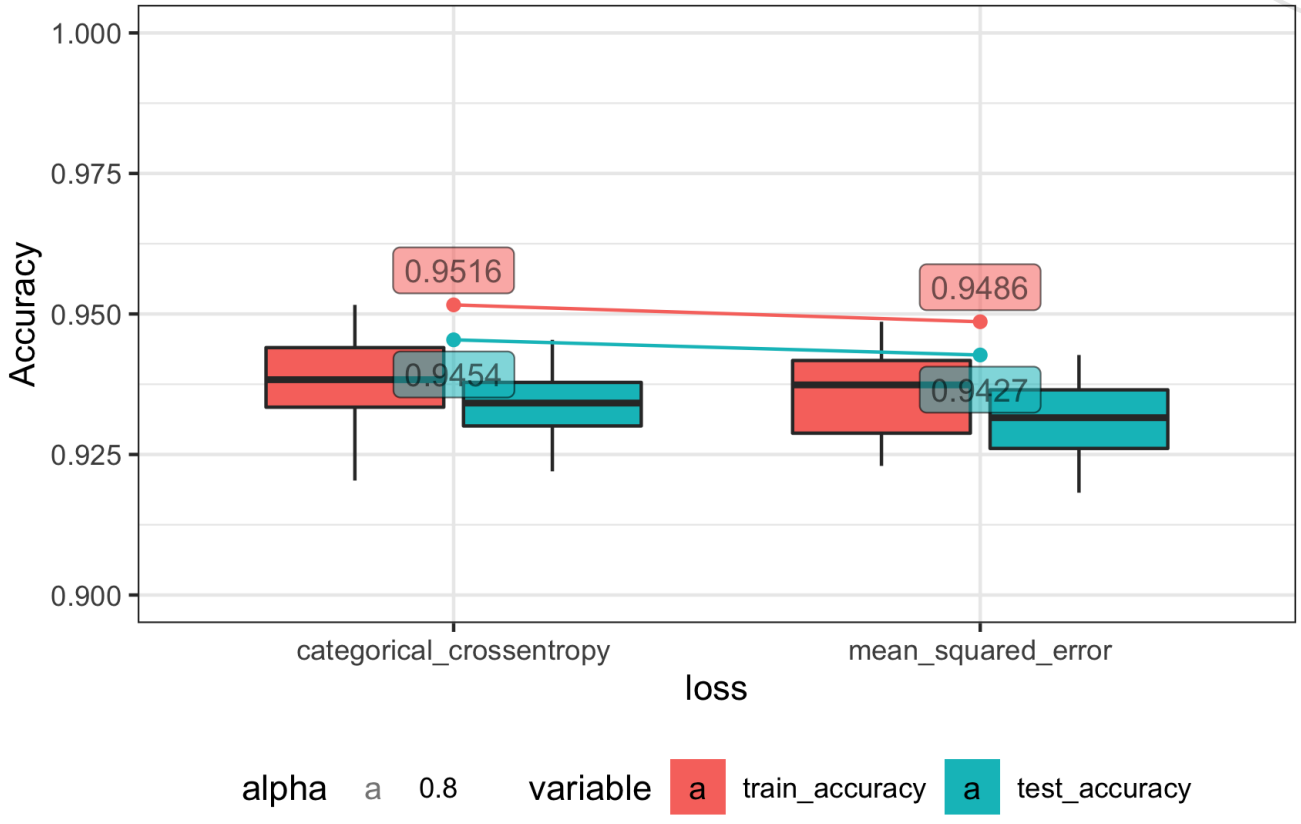


Figure 13: 1-Hidden Layer NN Accuracy rate versus loss(Softmax)

Neutral Network: 1-layer Neutral Network VII

Softmax activation function

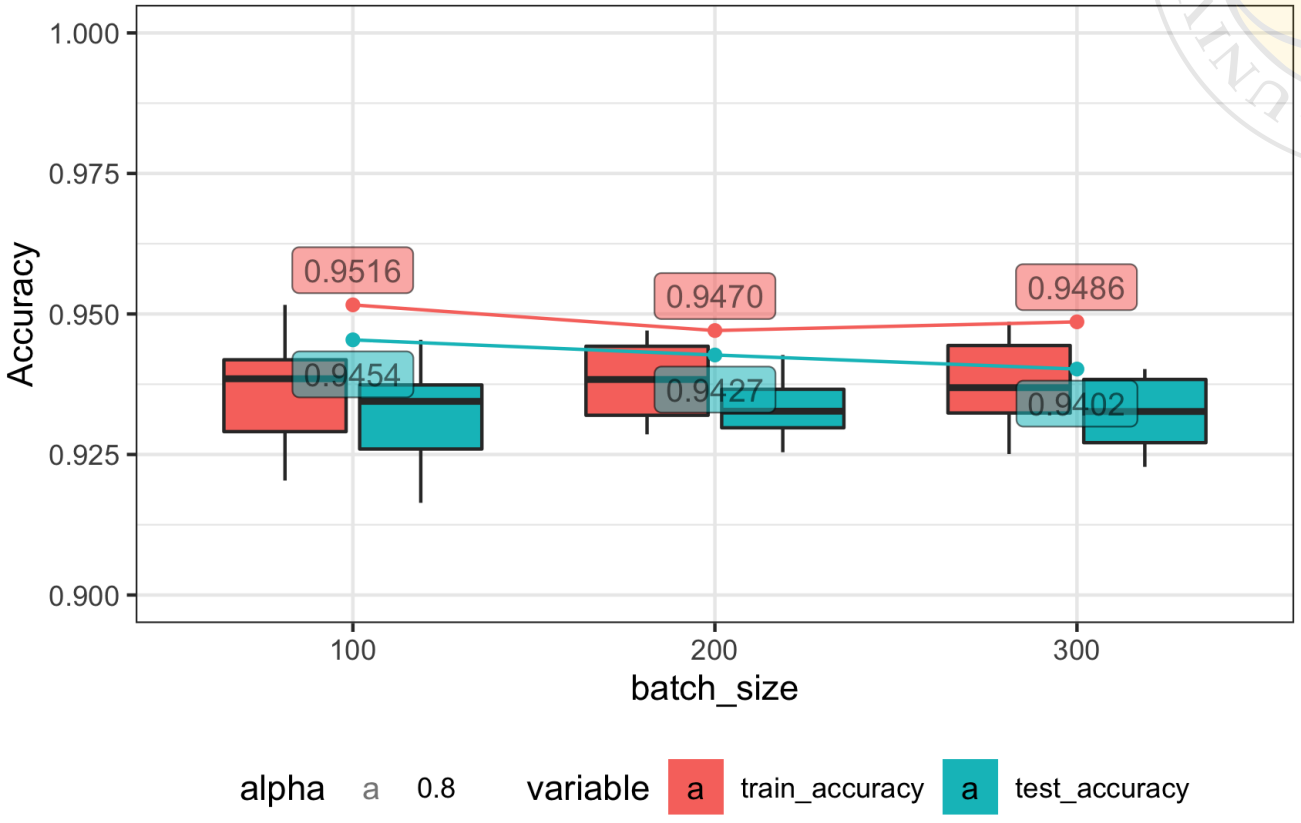


Figure 14: 1-Hidden Layer NN Accuracy rate versus batch size(Softmax)

We pick the model of best test accuracy, whose details are as in TABLE 6 and the confusion table is as TABLE 7.

Neutral Network: 1-layer Neutral Network VIII

Softmax activation function

Table 6: 1-layer Neutral Network best hyper-parameter(softmax)

hidden_nodes	activation	batch_size	loss
150	softmax	100	categorical_crossentropy
learning_rate	epoch	variable	value
0.01	9	test_accuracy	0.9454

Table 7: 1-layer Neutral Network confusion table(softmax)

	0	1	2	3	4	5	6	7	8	9	
0	965	0	9	0	2	5	6	1	5	2	0.9698
1	0	1126	4	0	0	1	4	8	4	6	0.9766
2	0	4	998	7	1	1	1	22	3	1	0.9615
3	1	1	5	977	0	10	0	10	23	9	0.9431
4	0	0	2	2	952	2	5	3	7	12	0.9665
5	3	1	0	10	0	853	13	0	7	1	0.9606
6	4	1	3	1	9	7	924	0	3	1	0.9696
7	1	1	7	4	0	2	0	966	5	5	0.9748
8	1	1	3	3	1	8	5	1	909	1	0.9743
9	5	0	1	6	17	3	0	17	8	971	0.9446
	0.9847	0.9921	0.9671	0.9673	0.9695	0.9563	0.9645	0.9397	0.9333	0.9623	0.9454

[4] Besides, we also used a learning rate of 0.1. However, the accuracy of training result was trapped around 10%, which means that a learning rate of 0.1 skipped the best parameter. Therefore, we didn't show the box plot of 0.1 learning rate here.

Neutral Network: 1-layer Neutral Network I

ReLU activation function

ReLU stands for rectified linear unit, and is a type of activation function:

$$\sigma(\mathbf{z})_j = \max(0, \mathbf{z}_j) \quad \text{for } j = 1, \ldots, K$$

Since we conclude bias term nearly have no affects in improving test accuracy in 1, we didn't consider bias term from now on.
We trained 108 1-layer Neutral Network models with different hidden nodes, learning rates and bias, for every type of model we trained 10 epochs. The hyper-parameter field are as follows, which are the same as the hyper-parameter field of 1-hidden layer neural networks with softmax activation function.

- Number of hidden nodes: 20, 50, 100, 150, 250, 500;
- batch size: 100, 200, 300;
- loss function: mean squared error, categorical cross-entropy;
- Learning rate: 0.001, 0.01, 0.1.

Neutral Network: 1-layer Neutral Network II

ReLU activation function

First, we need to know if all our model is converged. From Fig. 15, we can see that all of our models are converged. Besides, we can also see that those different group of model's iteration lines are approximately parallel, which implies that ReLU activation function can update the parameter more correctly, compared to Fig. 9.

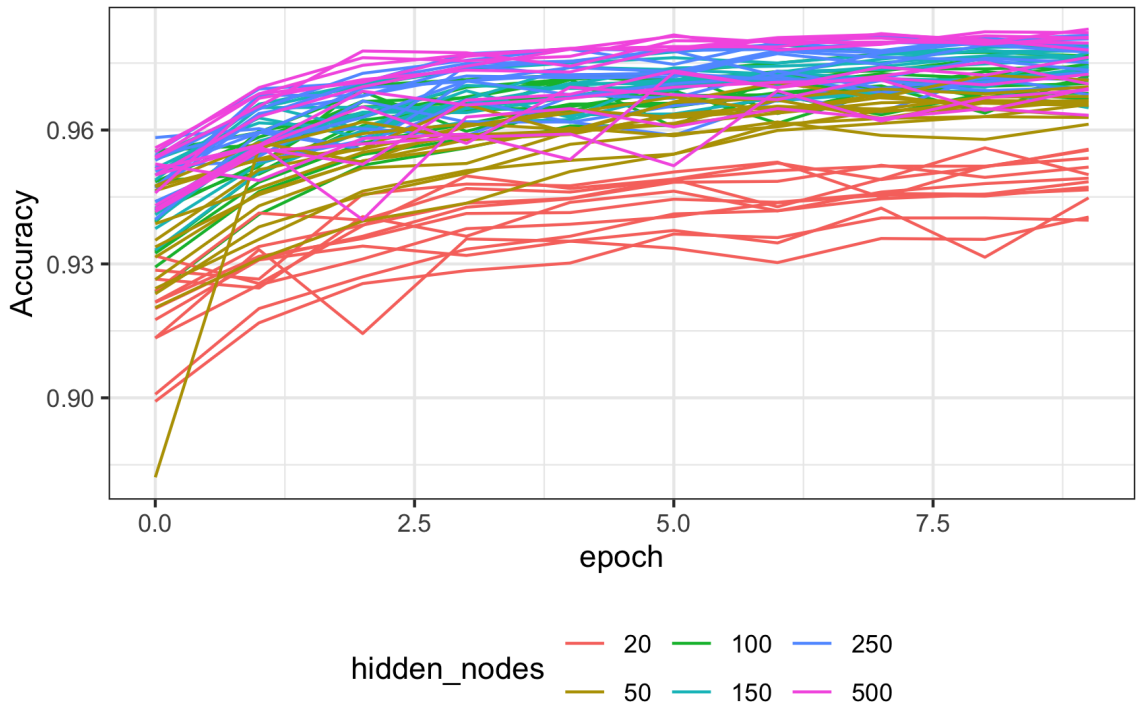


Figure 15: 1-layer Neural Network Test Accuracy By Epoch(ReLU)

Neutral Network: 1-layer Neutral Network III

ReLU activation function

Next, we tune all the hyper-parameters. From Fig. 16, we can see that the more hidden nodes, the better the maximum test accuracy is.

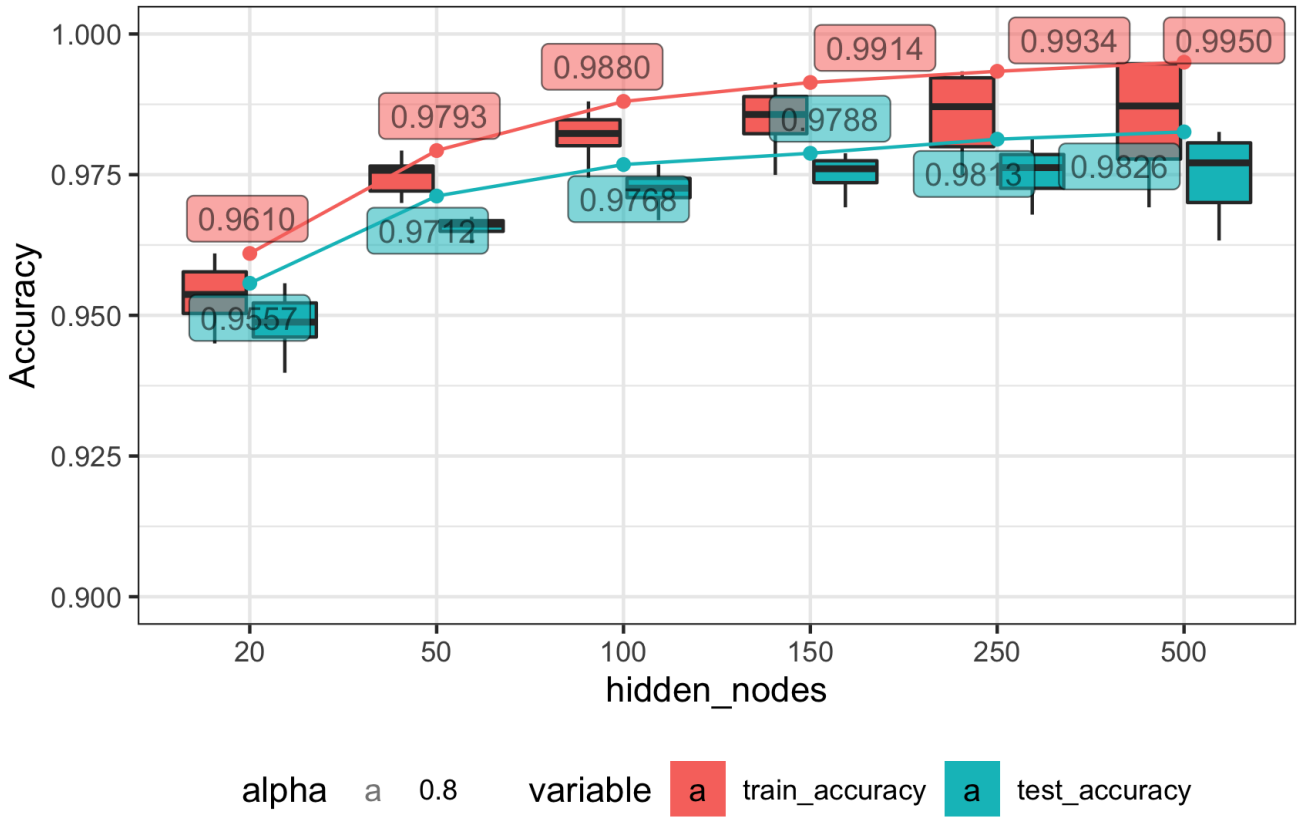
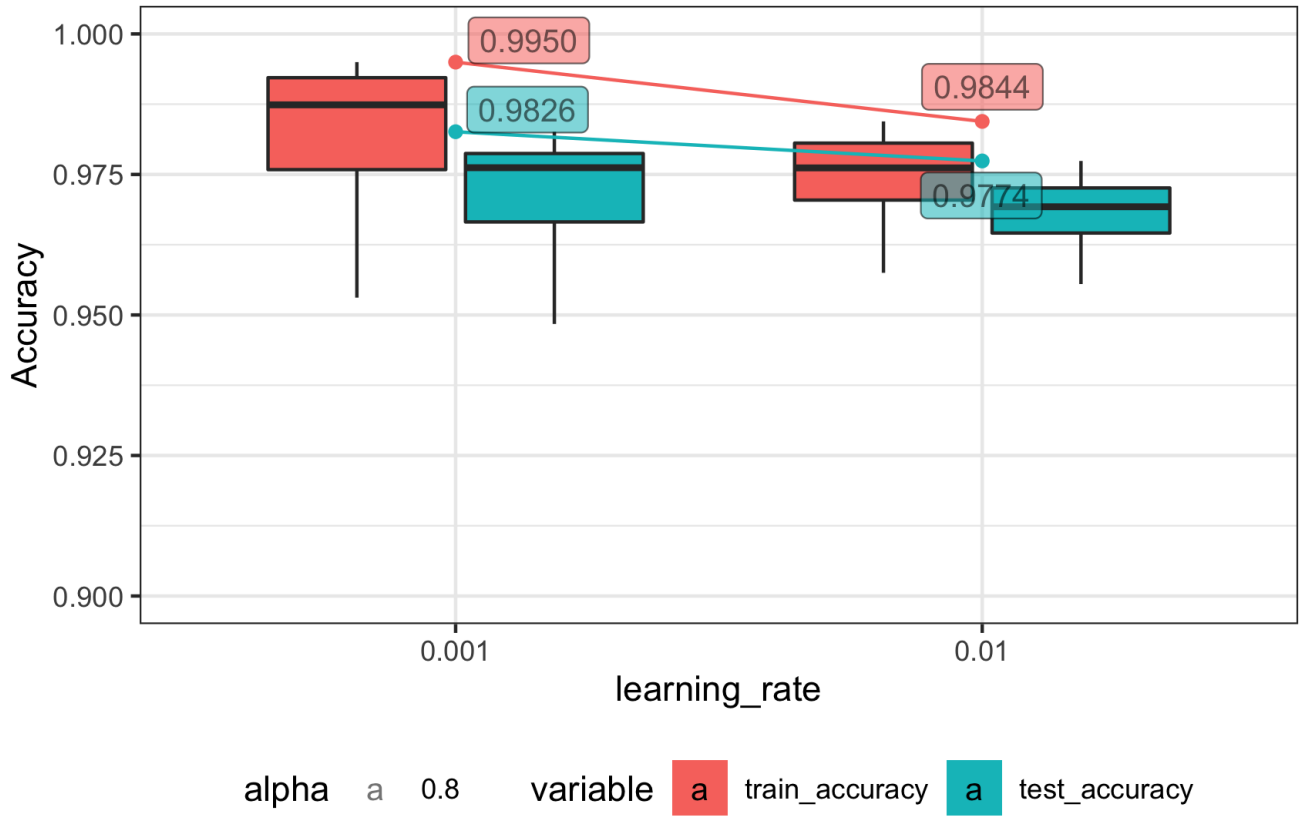


Figure 16: 1-Hidden Layer NN Accuracy rate versus hidden nodes(ReLU)

Neutral Network: 1-layer Neutral Network IV

ReLU activation function

From Fig. 17, we can see that a learning rate of 0.001 is better than 0.01. Besides, we also used a learning rate of 0.1. However, the accuracy of training result was trapped around 10%, which means that a learning rate of 0.1 skipped the best parameter. Therefore, we didn't show the box plot of 0.1 learning rate here.



Neutral Network: 1-layer Neutral Network V

ReLU activation function

Figure 17: 1-Hidden Layer Neural Network Accuracy rate versus learning rate(ReLU)

From Fig. 18, we conclude loss function barely impact accuracy.

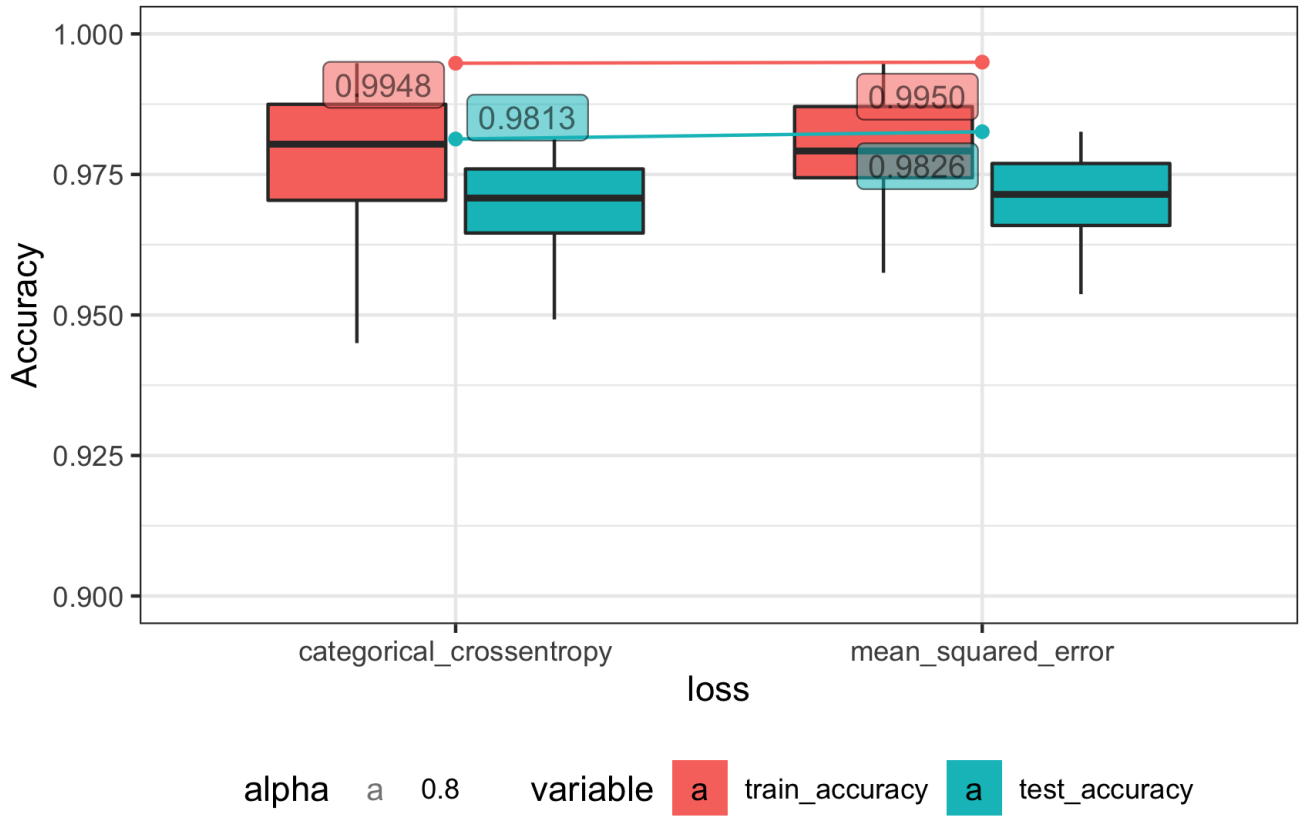


Figure 18: 1-Hidden Layer NN Accuracy rate versus loss(ReLU)

Neutral Network: 1-layer Neutral Network VI

ReLU activation function

From Fig. 19, we conclude batch size barely impact accuracy.

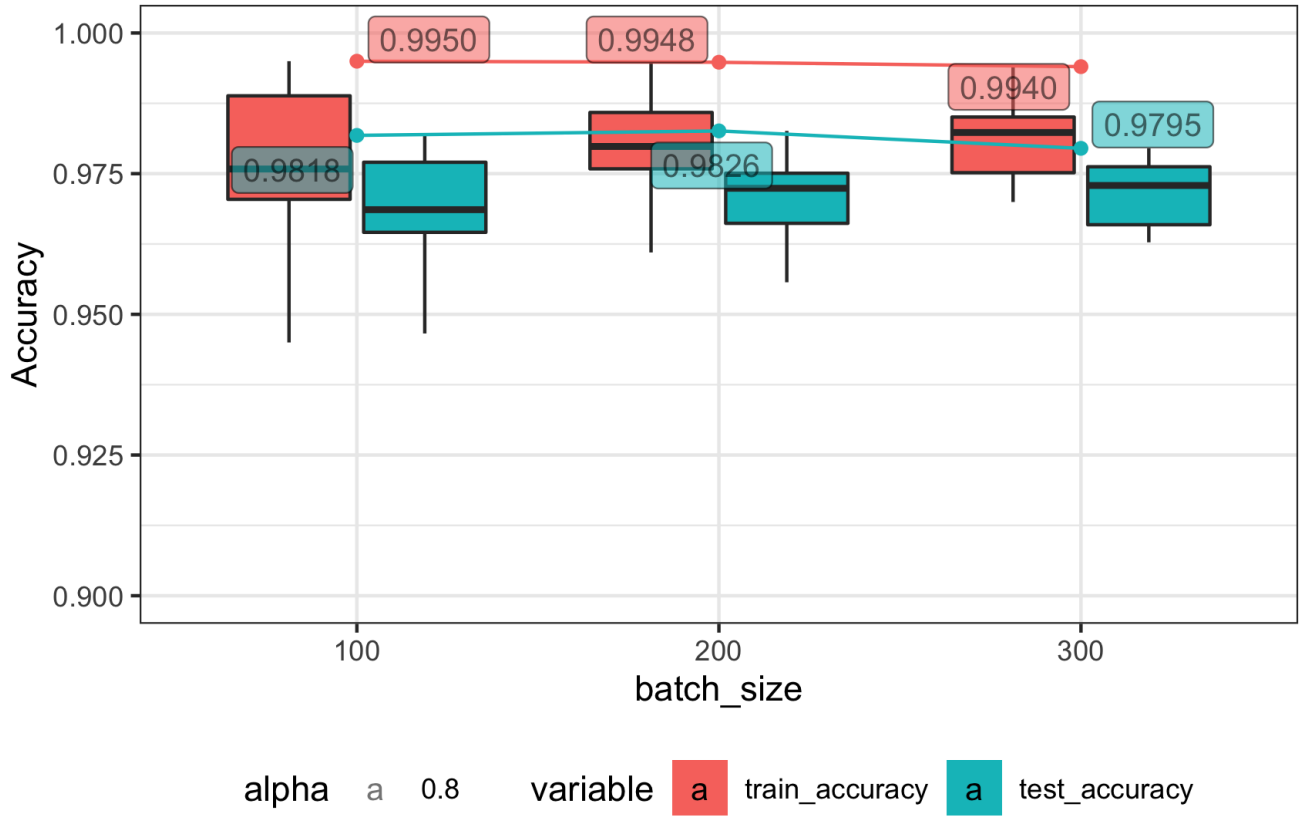


Figure 19: 1-Hidden Layer NN Accuracy rate versus batch size(ReLU)

Neutral Network: 1-layer Neutral Network VII

ReLU activation function

We pick the model of best test accuracy, whose details are as in TABLE 8 and the confusion table is as TABLE 9.

Table 8: 1-layer Neutral Network best hyper-parameter(ReLU)

hidden_nodes	activation	batch_size	loss
500	ReLU	200	mean_squared_error
learning_rate	epoch	variable	value
0.001	10	test_accuracy	0.9826

Table 9: 1-layer Neutral Network confusion table(ReLU)

	0	1	2	3	4	5	6	7	8	9	
0	971	0	0	1	0	5	0	1	2	0	0.9908
1	0	1126	3	2	0	0	2	1	1	0	0.9921
2	1	1	1018	4	0	0	1	4	3	0	0.9864
3	0	0	2	996	0	5	0	3	1	3	0.9861
4	0	0	5	1	960	2	5	1	0	8	0.9776
5	1	0	0	4	0	885	1	0	1	0	0.9922
6	4	2	1	1	2	10	937	0	1	0	0.9781
7	1	2	8	3	0	1	0	1005	3	5	0.9776
8	2	0	6	6	3	8	3	5	939	2	0.9641
9	1	2	2	12	7	10	0	4	3	968	0.9594
	0.9898	0.9938	0.9742	0.9670	0.9877	0.9557	0.9874	0.9814	0.9843	0.9817	0.9826

Neutral Network: 1-layer Neutral Network I

Comparison

Now we compare the result of softmax activation function and ReLU activation function. From Fig. 20, we can see that ReLU activation greatly improved the overall accuracy.

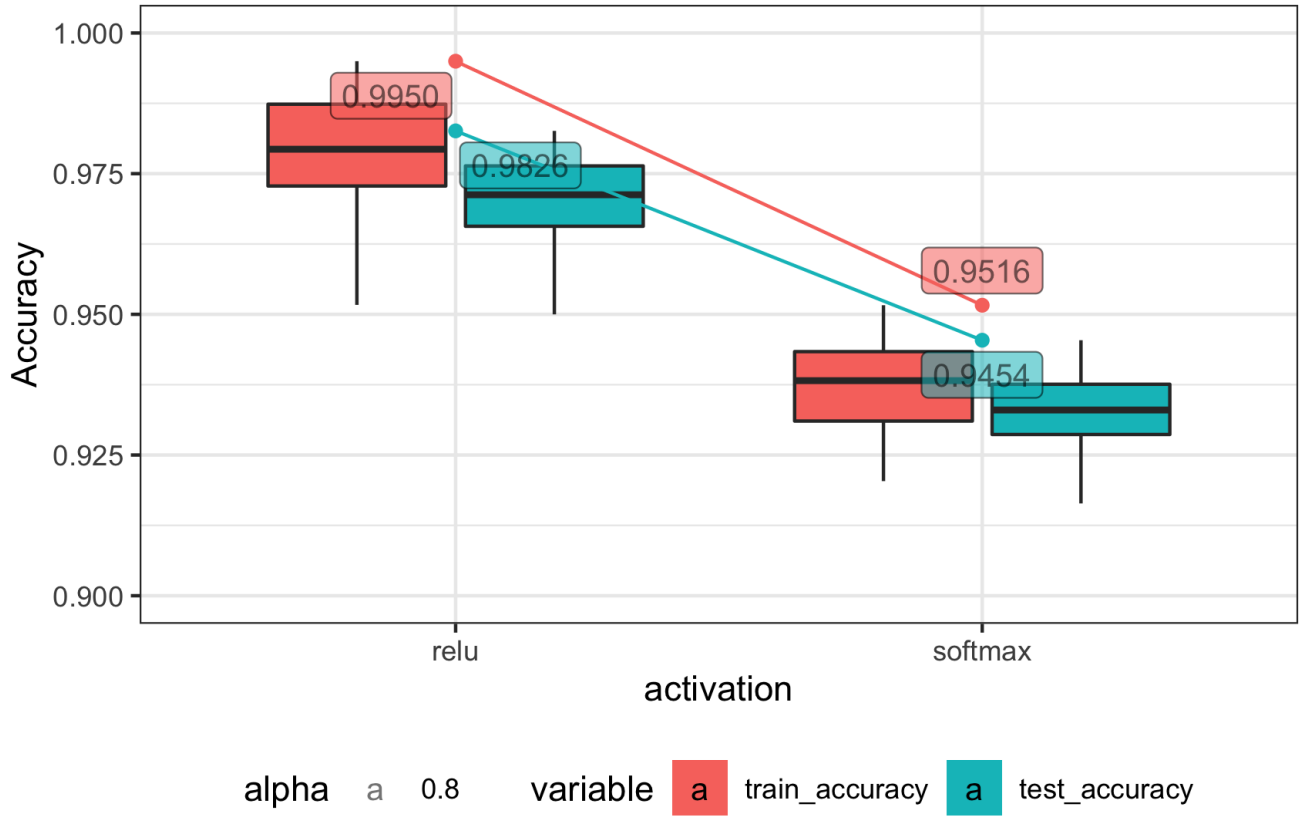


Figure 20: 1-Hidden Layer Neural Network Accuracy rate versus activation

Neutral Network: 1-layer Neutral Network II

Comparison

From Fig. 21, we can see that ReLU activation greatly shortened the converge time of neural network.

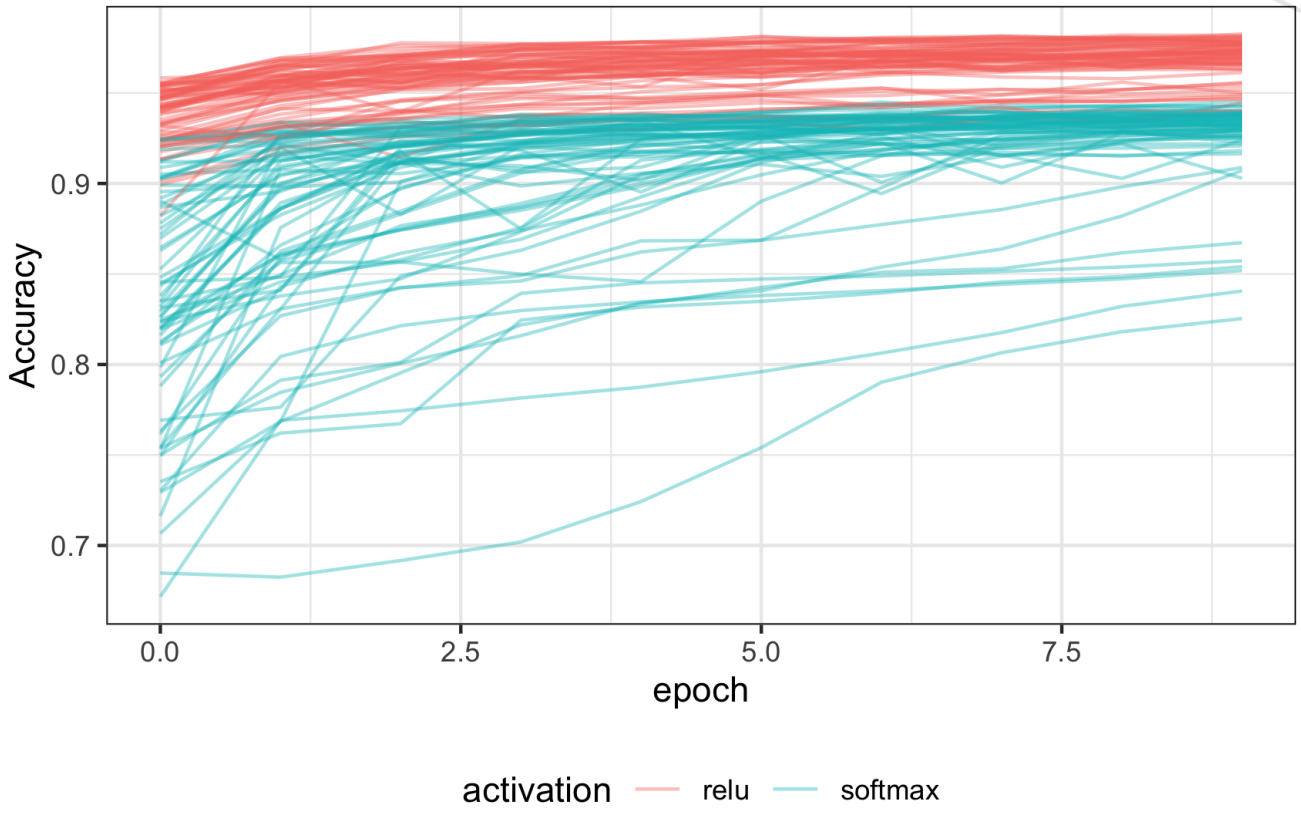


Figure 21: 1-Hidden layer Neural Network Accuracy rate By Epoch

Outline for Section 7.2

Neutral Network: 2-layer Neutral Network I

We didn't introduce bias term in this section and skipped the learning rate of 0.1 when tunning hyper-parameters and only uses ReLU activation function. We trained 648 2-layer Neutral Network models with different hidden nodes, learning rates and bias, and for every type of model we trained 10 epochs. The hyper-parameter field are as follows:

- Number of hidden nodes of 1st layer: 20, 50, 100, 150, 250, 500;
- Number of hidden nodes of 2nd layer: 20, 50, 100, 150, 250, 500;
- batch size: 100, 200, 300;
- loss function: mean squared error, categorical cross-entropy;
- Learning rate: 0.001, 0.01.

Neutral Network: 2-layer Neutral Network II

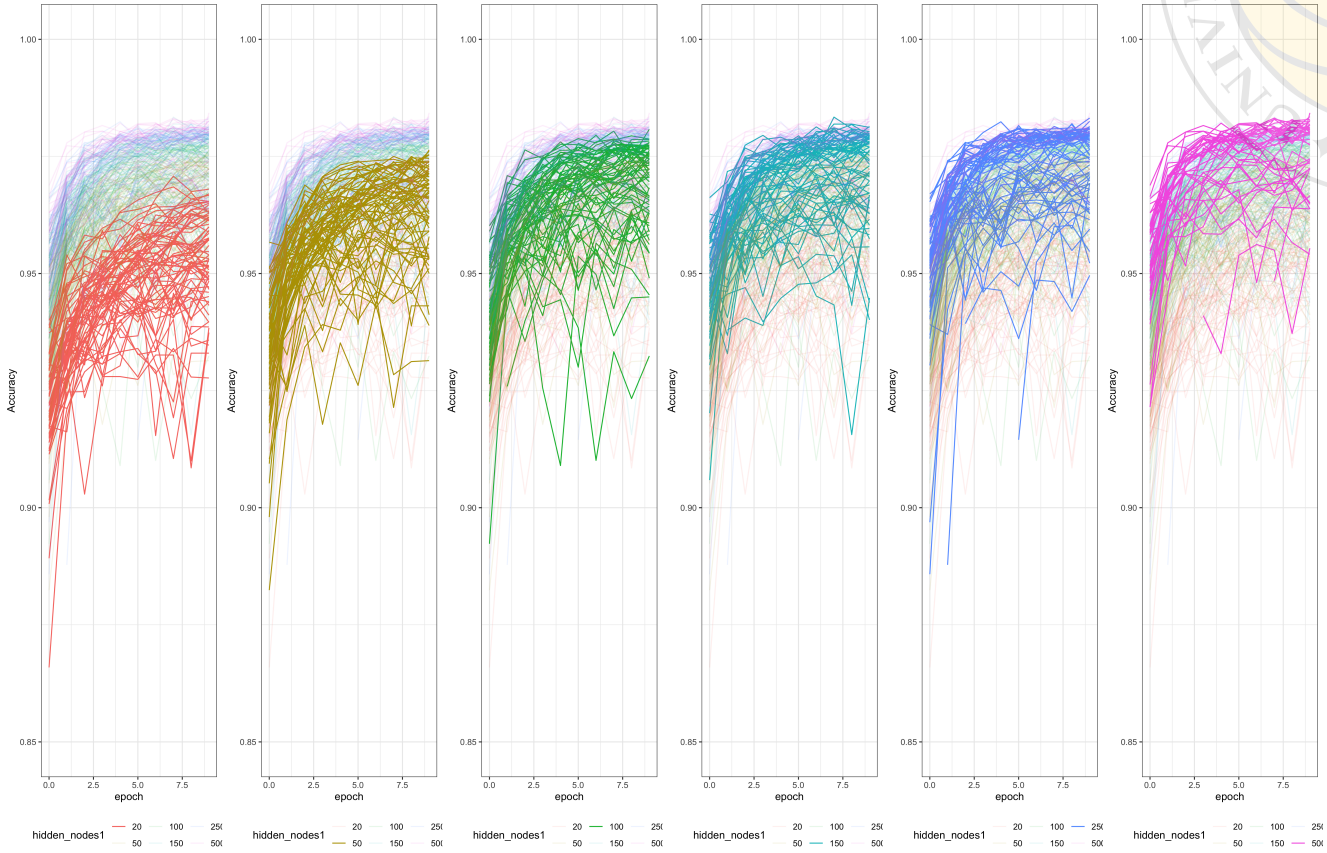


Figure 22: 2-layer Neural Network Test Accuracy By Epoch(ReLU)-1

Neutral Network: 2-layer Neutral Network III

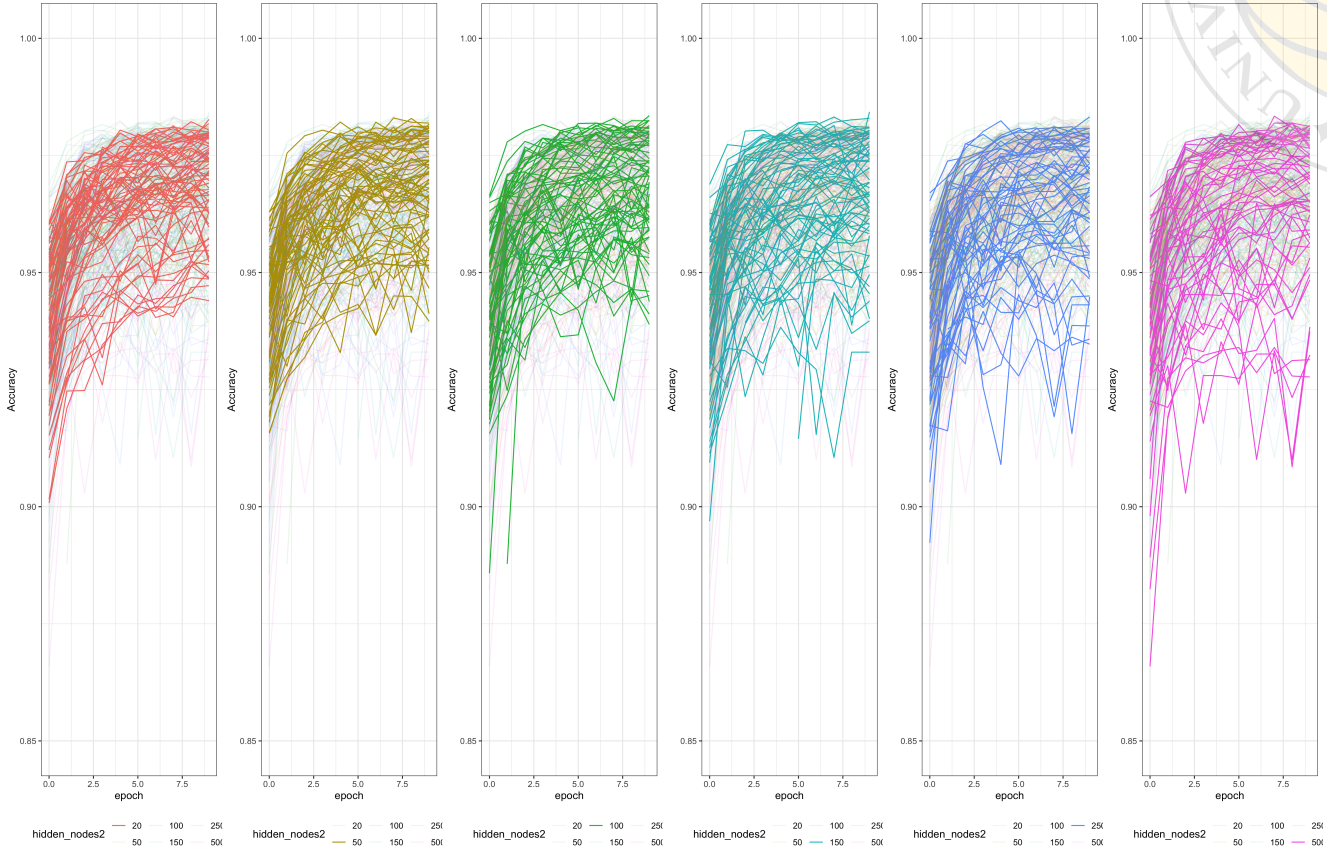


Figure 23: 2-layer Neural Network Test Accuracy By Epoch(ReLU)-2

Neutral Network: 2-layer Neutral Network IV

First, we need to know if all our model is converged. From Fig. 22 and Fig. 23, we can see that all of our models are converged. And from Fig. 22, we can see that with the number of 1st hidden layer nodes going up, the test accuracy going up.
Next, we tune all the hyper-parameters.

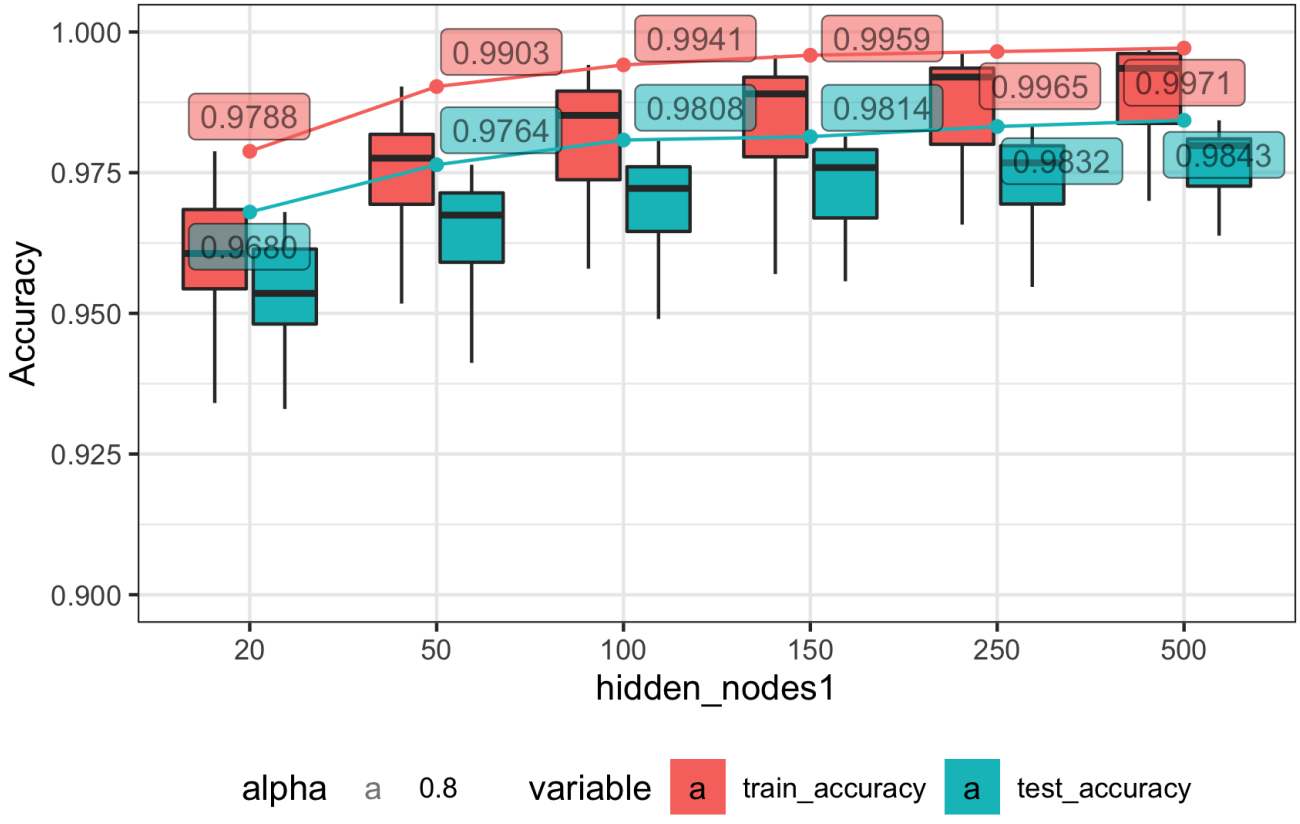
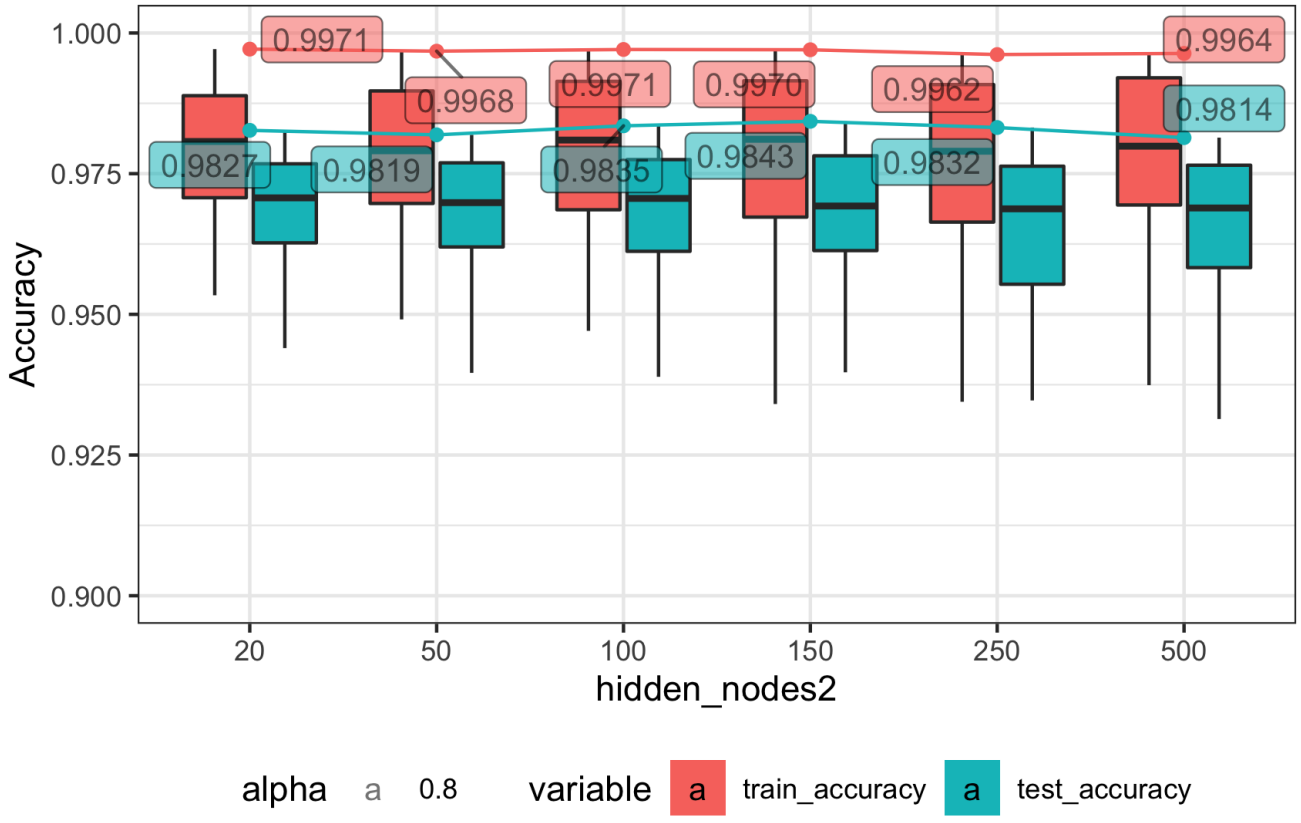


Figure 24: 2-Hidden Layer NN Accuracy rate versus hidden nodes of 1st layer(ReLU)

Neutral Network: 2-layer Neutral Network V

From Fig. 24 and Fig. 25, we can see that the more 1st layer hidden nodes, the better the maximum test accuracy is, the more 2nd layer hidden nodes, the larger the variance of test accuracy is.



Neutral Network: 2-layer Neutral Network VI

Figure 25: 2-Hidden Layer NN Accuracy rate versus hidden nodes of 2nd layer(ReLU)

From Fig. 26, we can see that a learning rate of 0.001 is better than 0.01.

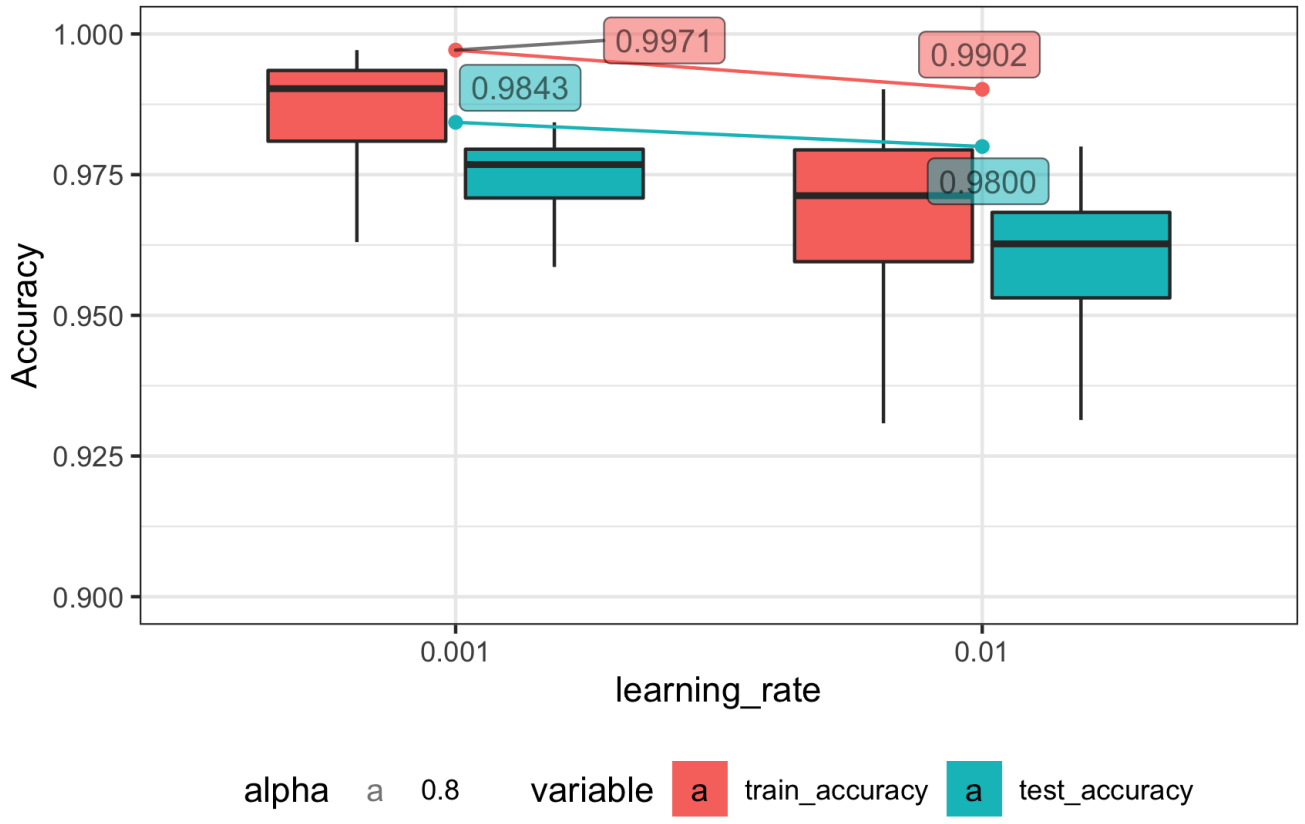


Figure 26: 2-Hidden Layer Neural Network Accuracy rate versus learning rate(ReLU)

Neutral Network: 2-layer Neutral Network VII

From Fig. 27, we conclude loss function barely impact accuracy.

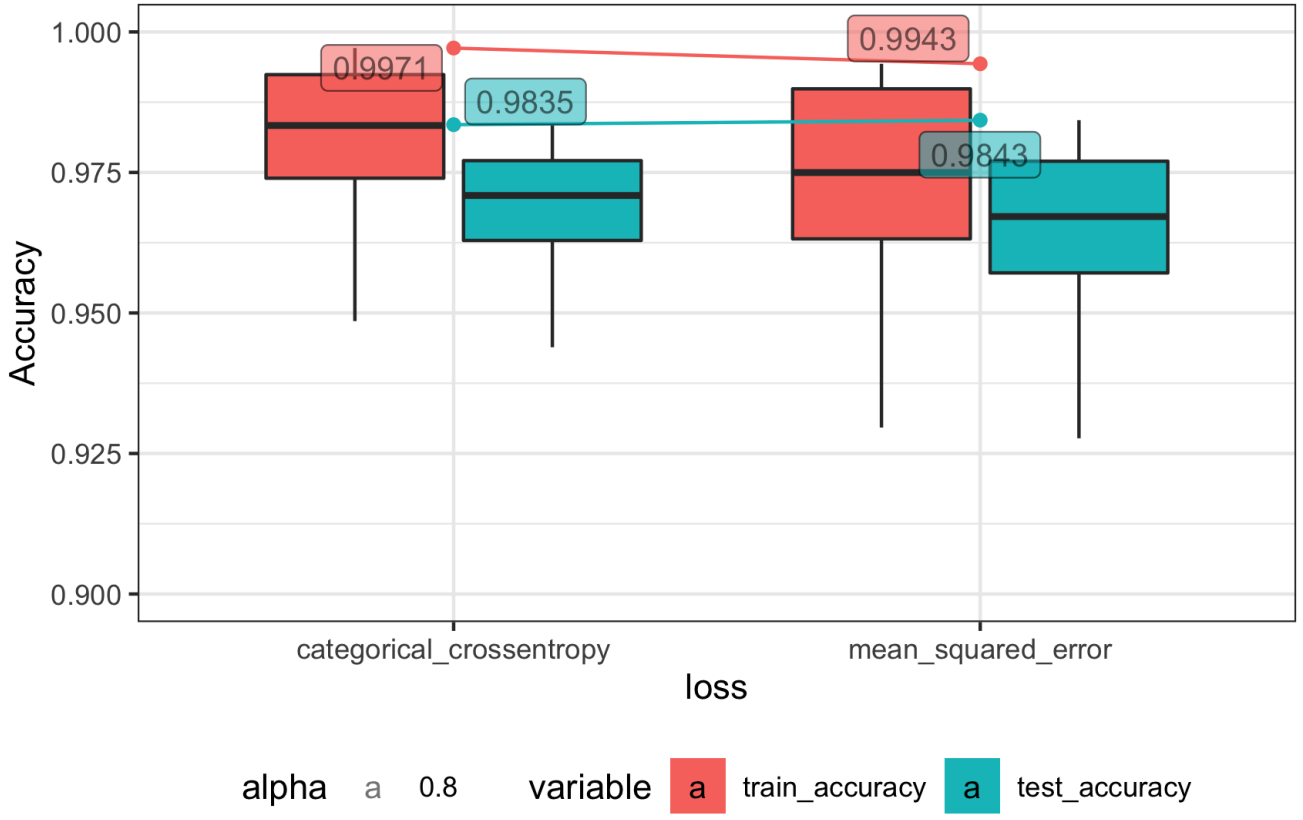


Figure 27: 2-Hidden Layer NN Accuracy rate versus loss(ReLU)

Neutral Network: 2-layer Neutral Network VIII

From Fig. 28, we conclude batch size barely impact accuracy.

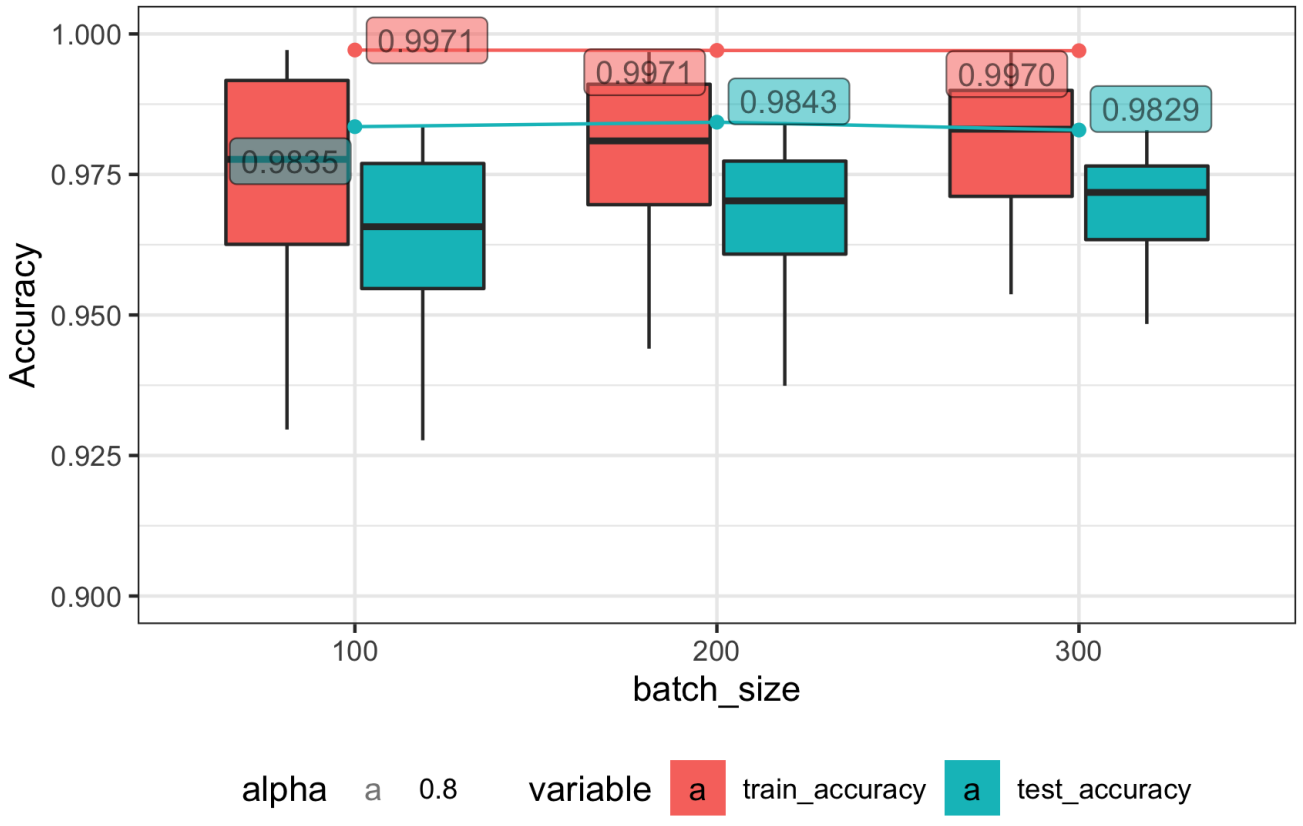


Figure 28: 2-Hidden Layer NN Accuracy rate versus batch size(ReLU)

Neutral Network: 2-layer Neutral Network IX

- The hyper-parameter of best model is as TABLE 10
- The test accuracy is 0.9843, train accuracy is 0.9971, and the confusion table is as TABLE 11.
- Here, we achieved a slight improvement than the 1 hidden layer neural network.

Table 10: 2-layer Neutral Network best hyper-parameter(ReLU)

hidden_nodes1 500	hidden_nodes2 150	activation ReLU	batch_size 200
learning_rate 0.001	epoch 10	variable test_accuracy	value 0.9843
loss mean_squared_error			

Table 11: 2-layer Neutral Network confusion table

	0	1	2	3	4	5	6	7	8	9	
	0	1	2	3	4	5	6	7	8	9	
0	975	1	0	0	1	1	1	1	1	0	0.9939
1	0	1129	1	2	0	1	2	0	0	0	0.9947
2	4	3	1002	3	5	0	4	6	4	0	0.9719
3	0	1	1	1000	0	4	0	4	4	1	0.9852
4	0	2	3	0	968	0	2	0	0	7	0.9857
5	3	1	0	5	1	867	7	0	2	1	0.9775
6	2	2	0	1	8	4	941	0	0	0	0.9823
7	1	5	4	2	0	0	0	1007	4	5	0.9796
8	2	1	2	3	3	1	4	3	953	2	0.9784
9	3	4	0	4	8	9	1	4	4	972	0.9633
	0.9848	0.9826	0.9891	0.9804	0.9738	0.9775	0.9782	0.9824	0.9805	0.9838	0.9843

Outline for Section 8

Conclusion I

Table 12: Comparison

Method	Train Accurancy	Test Accurancy	Time(Second)
SVM Linear Kernel	0.9239	0.9178	344
SVM RBF Kernel	0.9983	0.9852	2,749
SVM Polynomial Kernel	0.9989	0.9806	1,439
Gradient Boosting	0.9981	0.9673	7,321
Ada Boosting	0.8911	0.8902	1,402
1-layer NN 150 (Softmax)	0.9516	0.9454	30
1-layer NN 500 (ReLU)	0.9950	0.9826	40
2-Layer NN 500-150 (ReLU,ReLU)	0.9971	0.9843	57

- From TABLE 12, we can see that SVM with RBF kernel have the best test accuracy, but it takes a lot of time to train.
- The second best model is 2-layer neural network with 500-150 hidden nodes and ReLU activation function, and it only takes 57 seconds to train the model.

Conclusion II

- The test accuracy difference of these 2 model is only 0.0008, but the time difference is nearly 3,000 seconds.
- Therefore, we conclude that for MNIST data, the best model, considering test accuracy and training time, is 2-layer neural network with 500-150 hidden nodes and ReLU activation function.

Outline for Section 9

References

[1] Yann LeCun and Corinna Cortes.
MNIST handwritten digit database.
2010.

[2] François Chollet et al.
Keras.
https://github.com/fchollet/keras, 2015.

Outline for Section 10

Questions I

- Questions?