Hand Written Number Identification

1st Zhendong Zhang
dept. Mathematic Science
University of Central Florida
Orlando, United States of America
Zhendongzhang@knights.ucf.edu

***Abstract*—Handwritten Text Recognition is one of the most important techniques used in digital ages. In order to find best ways to recognize text, we explored several models with data mining methodologies in this report. We Compared accuracy, efficiency and characteristics of Support Vector Machine with 3 different Kernels (Linear, RBF and Polynomial), 2 Boost methods (Ada Boosting and Gradient Boosting) and several Neural Networks with different layers and activation functions. From the result, with consideration to the fallacy of time inefficiency of the most accurate model, we concluded the second most accurate model, which is the 2-layer neural network with 500 hidden nodes and ReLU activation function, as our recommendation for Handwritten Text Recognition application.**

***Index Terms*—Computer Vision, SVM, Kernel, Boost, Neural Networks**

I. Introduction

A. Background

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.

B. Problem Description

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?

II. Data Preparation

A. Data Source

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

B. 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

Fig. 1. Example of data

C. 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).

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

III. Data visualization

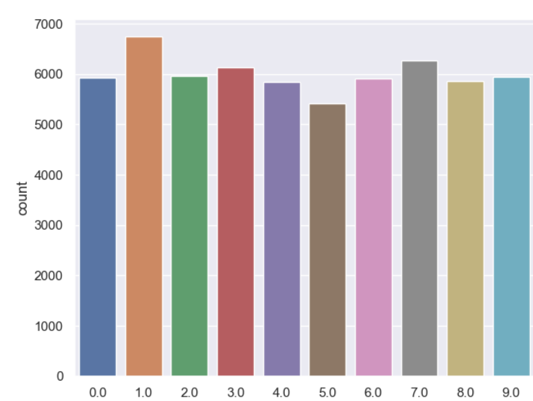

Fig. 2. Distribution of MNIST data

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

IV. USED PROGRAMMING LANGUAGE

We are originally use R language as our tool to model, and 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 VII.

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.

V. SUPPORT VECTOR MACHINES

In machine learning, support-vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

A. Linear Kernel

1) Global optimization: First we use support vector machines of linear kernel.

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

We added Elastic-Net regularization term [2](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. 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.

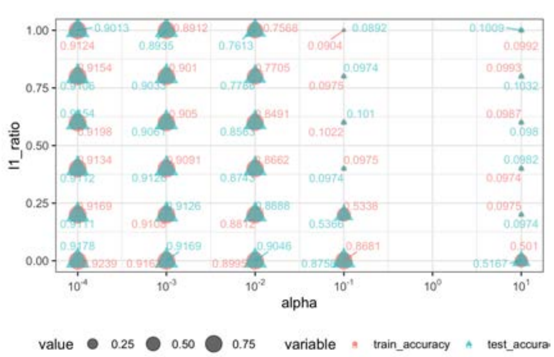

Fig. 3. SGD SVM Accuracy rate, alpha versus l1 ratio

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

TABLE I
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

B. RBF Kernel

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

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

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

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

It takes total 2,263.077 seconds to train those 4 model with 4-core CPU. And we gain best model of $\gamma = 0.01$. We get a test accuracy of 0.9375.

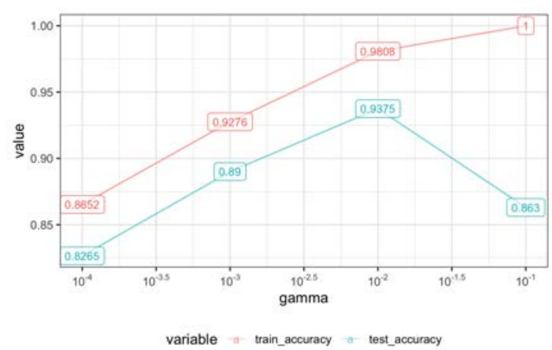

Fig. 4. RBF SVM Accuracy versus gamma

2) 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 II.

TABLE II
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

C. Polynomial Kernel

We now use Polynomial kernel to train model.

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

1) Tuning with 20% data set: Considering the computation complexity of Polynomial Kernel, we only use 20% data set to tune our model. The hyper-parameter field are as follows.

- $\gamma: 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$;
- $d = 1, 2, 3, 4, 5$.

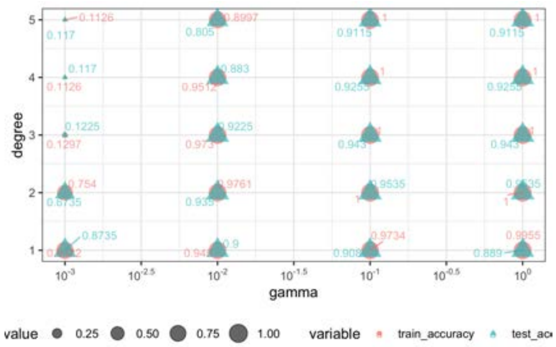

Fig. 5. Polynomial SVM Accuracy degree versus gamma

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.

2) 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 III.

TABLE III
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

VI. BOOSTING

A. Ada Boosting

1) 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 $= 0.1$.

2) Train best model with whole data set: Now we apply the best model of learning rate $= 0.1$ into whole data, and 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 IV. 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

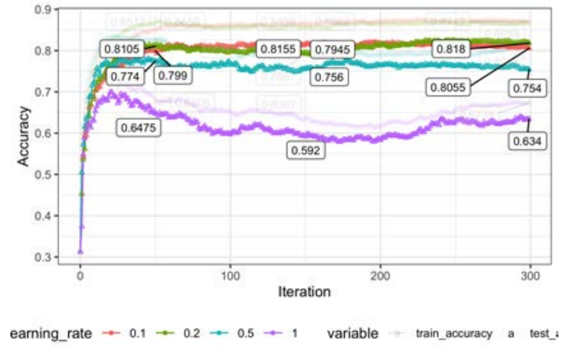

Fig. 6. Ada Boosting Accuracy By Iteration

increase the iteration number and decrease the learning rate might increase accuracy.

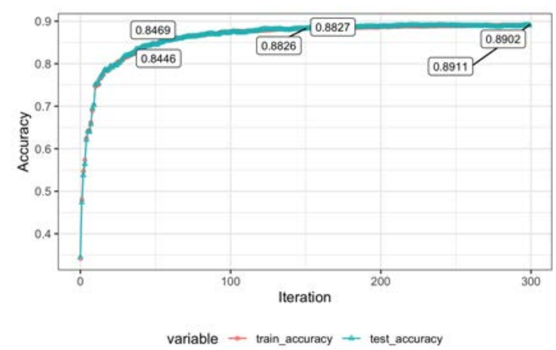

Fig. 7. Final Ada Boosting Accuracy By Iteration

TABLE IV
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

B. Gradient Boosting

1) 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. We used 12,129 seconds in tuning parameters.

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]

[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.

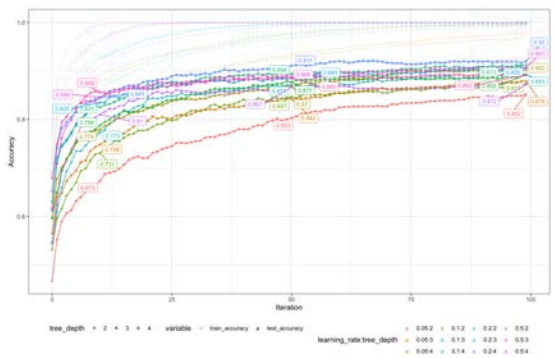

Fig. 8. Gradient Boosting Accuracy By Iteration

2) 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 0.9674, train accuracy is 0.9981, and the confusion table is as TABLE V.

TABLE V
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	0.9674

VII. NEUTRAL NETWORK

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 [3].

A. 1-layer Neutral Network

1) 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 [4].

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

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.

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.

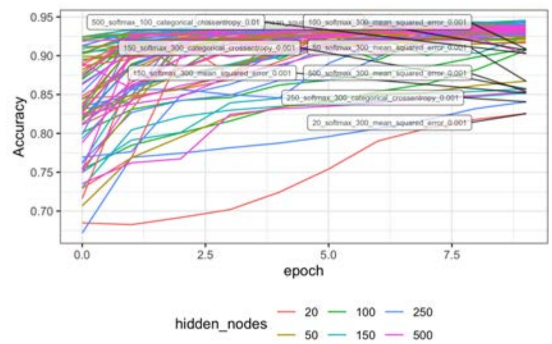

Fig. 9. 1-layer Neural Network Test Accuracy By Epoch

Next, we tune all the hyper-parameters. From Fig. 10, we can see that bias barely improves accuracy. 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. From Fig. 12, we can see that a learning rate of 0.01 tends to have better accuracy than 0.001. 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. From Fig. 13, we can see that a the use of different loss function don't impact the test accuracy a lot.

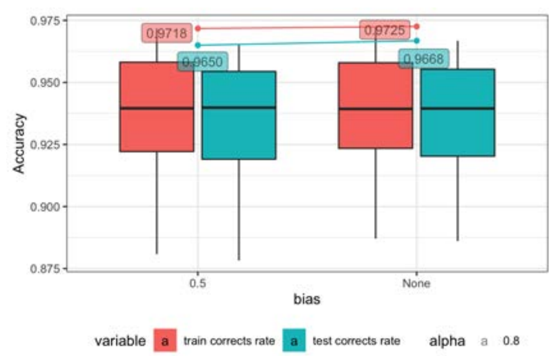

Fig. 10. 1-Hidden Layer NN Accuracy rate versus bias(Softmax)

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

[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.

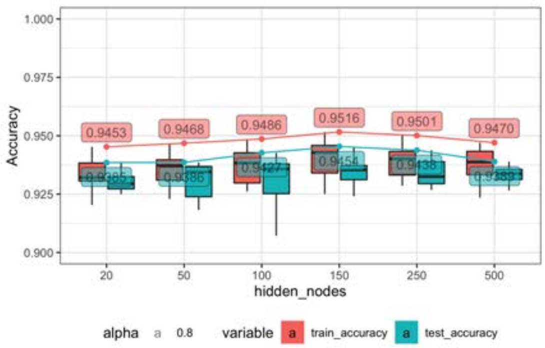

Fig. 11. 1-Hidden Layer NN Accuracy rate versus hidden nodes(Softmax)

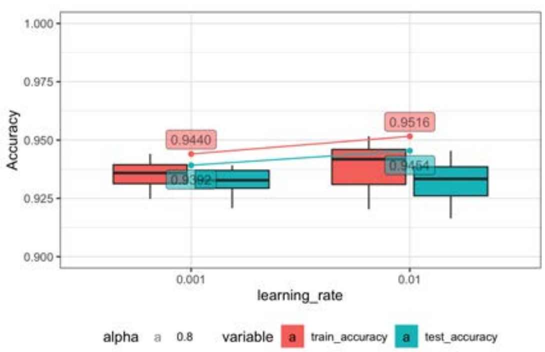

Fig. 12. 1-Hidden Layer NN Accuracy rate versus learning rate(Softmax)

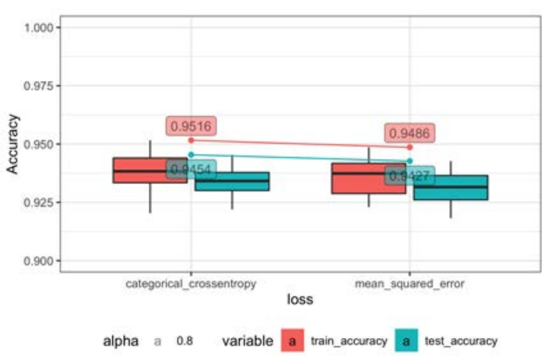

Fig. 13. 1-Hidden Layer NN Accuracy rate versus loss(Softmax)

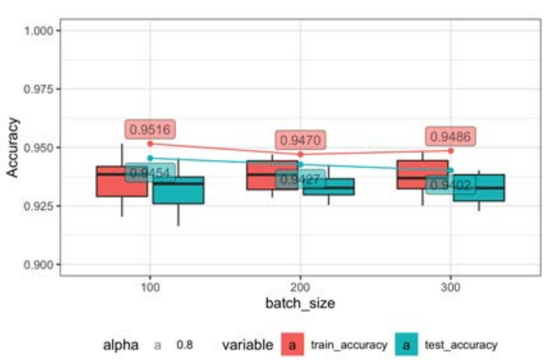

Fig. 14. 1-Hidden Layer NN Accuracy rate versus batch size(Softmax)

TABLE VI

1-LAYER NEUTRAL NETWORK BEST HYPER-PARAMETER(SOFTMAX)

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

TABLE VII

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

2) ReLU activation function: ReLU [5] 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 VII-A1, 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.

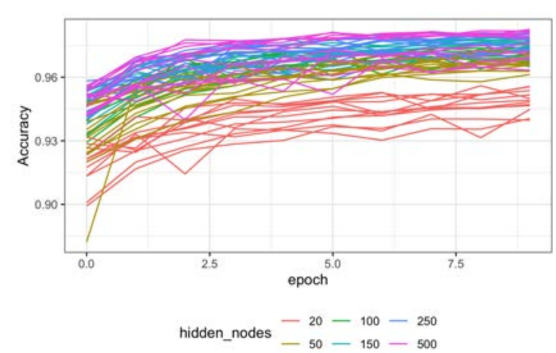

Fig. 15. 1-layer Neural Network Test Accuracy By Epoch(ReLU)

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.

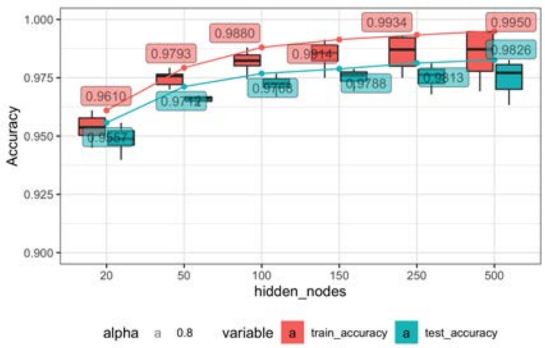

Fig. 16. 1-Hidden Layer NN Accuracy rate versus hidden nodes(ReLU)

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. 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. From Fig. 18, we conclude loss function barely impact accuracy. From Fig. 19, we conclude batch size barely impact accuracy.

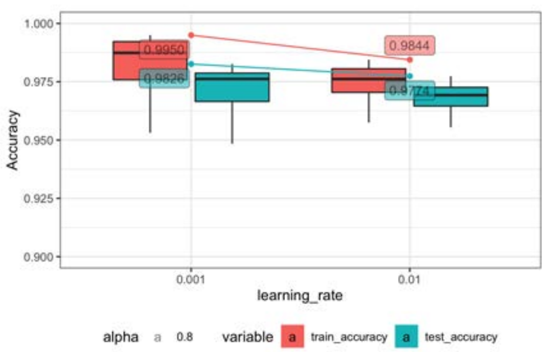

Fig. 17. 1-Hidden Layer Neural Network Accuracy rate versus learning rate(ReLU)

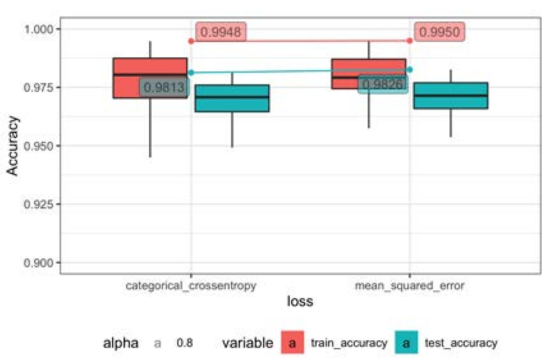

Fig. 18. 1-Hidden Layer NN Accuracy rate versus loss(ReLU)

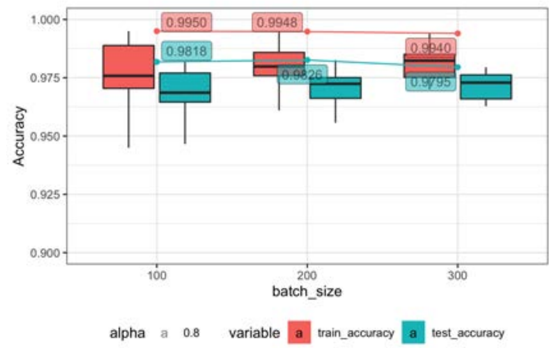

Fig. 19. 1-Hidden Layer NN Accuracy rate versus batch size(ReLU)

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

TABLE VIII
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 IX
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

3) Comparison: Now we compare the result of softmax activation function and ReLU activation function.

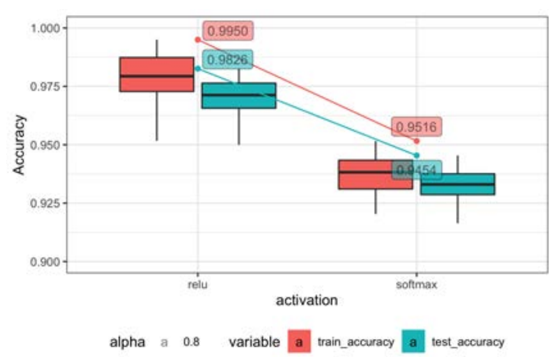

Fig. 20. 1-Hidden Layer Neural Network Accuracy rate versus activation

From Fig. 20, we can see that ReLU activation greatly improved the overall accuracy. From Fig. 21, we can see that ReLU activation greatly shortened the converge time of neural network.

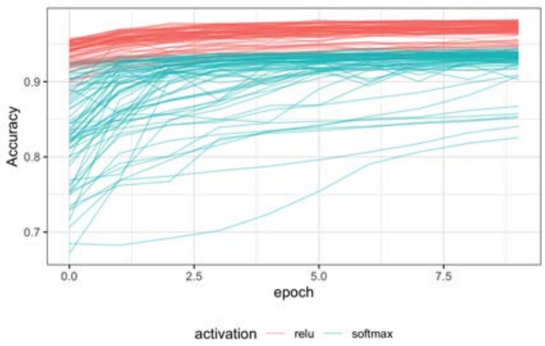

Fig. 21. 1-Hidden layer Neural Network Accuracy rate By Epoch

B. 2-layer Neutral Network

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.

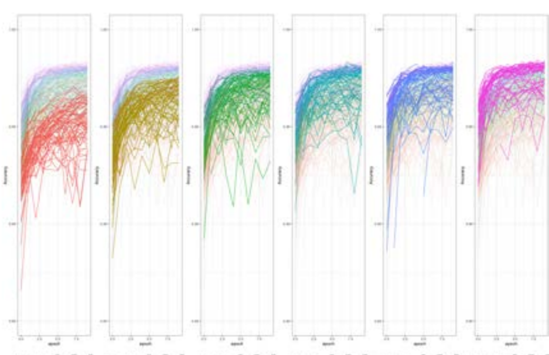

Fig. 22. 2-layer Neural Network Test Accuracy By Epoch(ReLU)-1

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. 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. From Fig. 26, we can see that a learning rate of 0.001 is better than 0.01. From Fig. 27, we conclude loss function barely impact

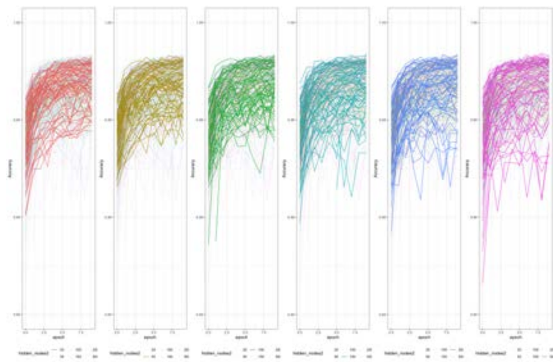

Fig. 23. 2-layer Neural Network Test Accuracy By Epoch(ReLU)-2

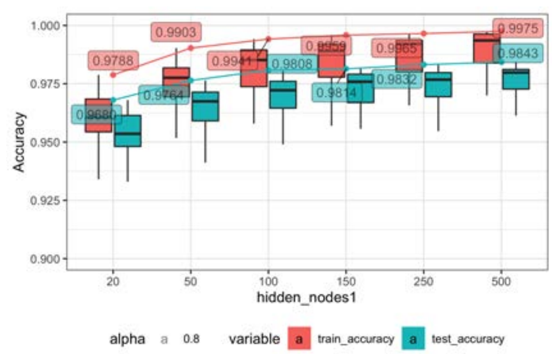

Fig. 24. 2-Hidden Layer NN Accuracy rate versus hidden nodes of 1st layer(ReLU)

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

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

VIII. Conclusion

From TABLE XII, 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

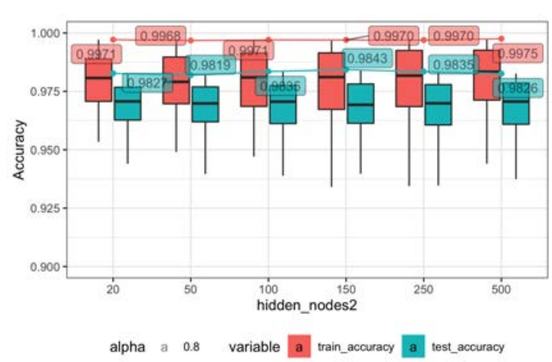

Fig. 25. 2-Hidden Layer NN Accuracy rate versus hidden nodes of 2nd layer(ReLU)

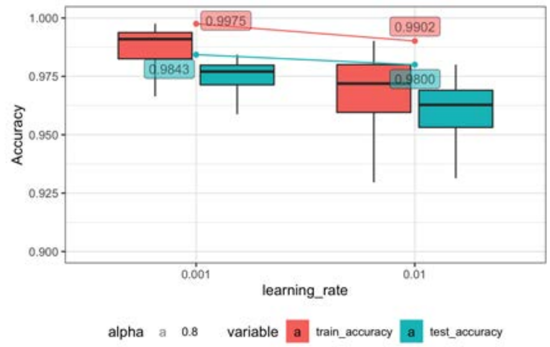

Fig. 26. 2-Hidden Layer Neural Network Accuracy rate versus learning rate(ReLU)

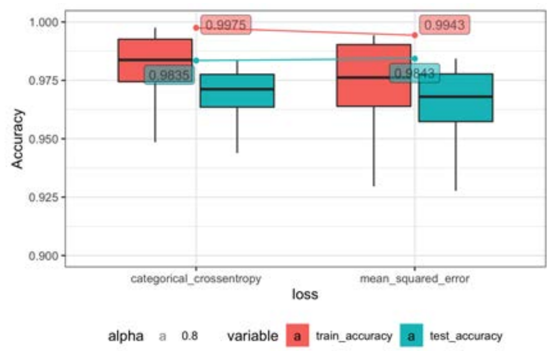

Fig. 27. 2-Hidden Layer NN Accuracy rate versus loss(ReLU)

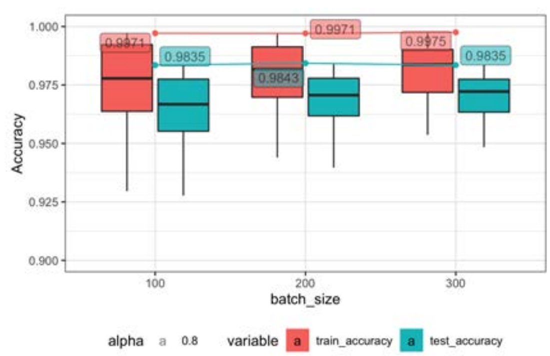

Fig. 28. 2-Hidden Layer NN Accuracy rate versus batch size(ReLU)

TABLE X
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 XI
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

TABLE XII
COMPARISON

Method	Train Accurancy	Test Accurancy	Yann LeCun	Time(Second)
SVM Linear Kernel	0.9239	0.9178	NA	344
SVM RBF Kernel	0.9983	0.9852	0.9860	2,749
SVM Polynomial Kernel	0.9989	0.9806	0.9890	1,439
Gradient Boosting	0.9981	0.9673	0.9874	7,321
Ada Boosting	0.8911	0.8902	0.923	1,402
1-layer NN 150 (Softmax)	0.9516	0.9454	0.953	30
1-layer NN 500 (ReLU)	0.9950	0.9826	0.984	40
2-Layer NN 500-150 (ReLU,ReLU)	0.9971	0.9843	0.9705	57

hidden nodes and ReLU activation function, and it only takes 40 seconds to train the model. The test accuracy difference of these 2 model is only 0.0026, 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 hidden nodes and ReLU activation function.

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my teacher who gave me the golden opportunity to do this wonderful project on the topic of this, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to him.

REFERENCES

[1] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[3] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[5] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with relu activation," in *Advances in neural information processing systems*, 2017, pp. 597–607.

Appendix A
Python Source Code

```python
import matplotlib.pyplot as plt
import numpy as np



image_size = 28 # width and length
no_of_different_labels = 10 # i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
data_path = "data/CSV/"
train_data = np.loadtxt(data_path + "mnist_train.csv",
                        delimiter=",")
test_data = np.loadtxt(data_path + "mnist_test.csv",
                       delimiter=",")

img = train_data[190,1:785].reshape((28,28))
plt.imshow(img, cmap="Greys")
plt.show()


fac = 0.99 / 255
train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01

train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])


# lr = np.arange(10)
#
# for label in range(10):
# one_hot = (lr==label).astype(np.int)
# print("label: ", label, " in one-hot representation: ", one_hot)

lr = np.arange(no_of_different_labels)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# we don't want zeroes and ones in the labels neither:
train_labels_one_hot[train_labels_one_hot==0] = 0.01
train_labels_one_hot[train_labels_one_hot==1] = 0.99
test_labels_one_hot[test_labels_one_hot==0] = 0.01
test_labels_one_hot[test_labels_one_hot==1] = 0.99

# Before we start using the MNIST data sets with our neural network, we will have a look at some images
for i in range(10):
    img = train_imgs[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.show()

import pickle

with open("data/pkl/pickled_mnist.pkl", "bw") as fh:
    data = (train_imgs,
            test_imgs,
            train_labels,
            test_labels,
            train_labels_one_hot,
            test_labels_one_hot)
    pickle.dump(data, fh)
```

Listing 1. Data Process

```python
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import svm
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist
import seaborn as sns
import numpy as np
import time
import pickle

sns.set()

(x_train, y_train), (x_test, y_test) = mnist.load_data()
num_classes = 10
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# n=0.2
# x_train=x_train[1:int(60000*n)+1,]
# x_test=x_test[1:int(10000*n)+1,]
# y_train=y_train[1:int(60000*n)+1,]
# y_test=y_test[1:int(10000*n)+1,]



print('Train size:', x_train.shape[0])
print('Test size:', x_test.shape[0])

# Following code is too slow consider SGD
# clf = svm.LinearSVC()

# with open("Result/nist_tests_SVM.csv", "w") as fh_out:
# for alpha in [0.0001,0.001,0.01,0.1,10]:
# for l1_ratio in [0,0.2,0.4,0.6,0.8,1]:
# print("*", end="")
# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3,alpha=alpha,penalty='elasticnet',l1_ratio=l1_ratio,learning_rate='optimal')
# clf_fit=clf.fit(train_imgs, np.ravel(train_labels))
# train_score = clf_fit.score(train_imgs, np.ravel(train_labels))
# test_score = clf_fit.score(test_imgs, np.ravel(test_labels))
#
# outstr = str(alpha) + " " + str(l1_ratio) + " "
# outstr += str(train_score) + " " + str(test_score)
#
# fh_out.write(outstr + "\n")
# fh_out.flush()

# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3)
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))
#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)


# clf = svm.SVC(kernel='linear')
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))

#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)
```

```python
with open("Result/nist_tests_SVM_rbf.csv", "w") as fh_out:
    for gamma in range(4):
        gamma = 10 ** -(gamma + 1)
        print("gamma=",gamma)
        clf = svm.SVC(kernel='rbf',gamma=gamma)
        start_time = time.time()
        clf_fit = clf.fit(x_train, y_train)
        training_time = time.time() - start_time
        print("--- %s seconds ---" % (training_time))

        outfile = 'SVMrbfmodel/' + str(gamma)
        with open(outfile, 'wb') as pickle_file:
            pickle_file = pickle.dump(clf, pickle_file)

        train_score = clf_fit.score(x_train, y_train)
        test_score = clf_fit.score(x_test, y_test)
        cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
        print('Train Accuracy=', train_score)
        print('Test Accuracy=', test_score)
        print(cm)

        outstr = str(gamma) + " " + str(train_score) + " " + str(test_score) +' '+str(training_time)
        fh_out.write(outstr + "\n")
        fh_out.flush()

with open("Result/nist_tests_SVM_poly.csv", "w") as fh_out:
    for degree in range(5):
        degree=degree+1
        for gamma in range(4):
            gamma=10**-(gamma)
            print("degree=" ,degree,"gamma=",gamma)
            clf = svm.SVC(kernel='poly', degree=degree,gamma=gamma)
            start_time = time.time()
            clf_fit = clf.fit(x_train, y_train)
            training_time=time.time() - start_time
            print("--- %s seconds ---" % (training_time))

            outfile = 'SVMpolymodel/' + str(degree) + '_' +str(gamma)
            with open(outfile, 'wb') as pickle_file:
                pickle_file = pickle.dump(clf, pickle_file)

            train_score = clf_fit.score(x_train, y_train)
            test_score = clf_fit.score(x_test, y_test)
            cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
            print('Train Accuracy=', train_score)
            print('Test Accuracy=', test_score)
            print(cm)

            outstr = str(degree) + " " + str(gamma) + " " + str(train_score) + " " + str(test_score)+' '+str(training_time)
            fh_out.write(outstr + "\n")
            fh_out.flush()
```

Listing 2. SVM

```python
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import svm
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist
import seaborn as sns
import numpy as np
import time
import pickle

sns.set()

(x_train, y_train), (x_test, y_test) = mnist.load_data()
num_classes = 10
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# n=0.2
# x_train=x_train[1:int(60000*n)+1,]
# x_test=x_test[1:int(10000*n)+1,]
# y_train=y_train[1:int(60000*n)+1,]
# y_test=y_test[1:int(10000*n)+1,]



print('Train size:', x_train.shape[0])
print('Test size:', x_test.shape[0])

# Following code is too slow consider SGD
# clf = svm.LinearSVC()

# with open("Result/nist_tests_SVM.csv", "w") as fh_out:
# for alpha in [0.0001,0.001,0.01,0.1,10]:
# for l1_ratio in [0,0.2,0.4,0.6,0.8,1]:
# print("*", end="")
# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3,alpha=alpha,penalty='elasticnet',l1_ratio=l1_ratio,learning_rate='optimal')
# clf_fit=clf.fit(train_imgs, np.ravel(train_labels))
# train_score = clf_fit.score(train_imgs, np.ravel(train_labels))
# test_score = clf_fit.score(test_imgs, np.ravel(test_labels))
#
# outstr = str(alpha) + " " + str(l1_ratio) + " "
# outstr += str(train_score) + " " + str(test_score)
#
# fh_out.write(outstr + "\n")
# fh_out.flush()

# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3)
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))
#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)


# clf = svm.SVC(kernel='linear')
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))

#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)


with open("Result/nist_tests_SVM_rbf.csv", "w") as fh_out:
    for gamma in range(4):
        gamma = 10 ** -(gamma + 1)
        print("gamma=",gamma)
        clf = svm.SVC(kernel='rbf',gamma=gamma)
        start_time = time.time()
        clf_fit = clf.fit(x_train, y_train)
        training_time = time.time() - start_time
        print("--- %s seconds ---" % (training_time))

```

```
        outfile = 'SVMrbfmodel/' + str(gamma)
        with open(outfile, 'wb') as pickle_file:
            pickle_file = pickle.dump(clf, pickle_file)

        train_score = clf_fit.score(x_train, y_train)
        test_score = clf_fit.score(x_test, y_test)
        cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
        print('Train Accuracy=', train_score)
        print('Test Accuracy=', test_score)
        print(cm)

        outstr = str(gamma) + " " + str(train_score) + " " + str(test_score) +' '+str(training_time)
        fh_out.write(outstr + "\n")
        fh_out.flush()

with open("Result/nist_tests_SVM_poly.csv", "w") as fh_out:
    for degree in range(5):
        degree=degree+1
        for gamma in range(4):
            gamma=10**-(gamma)
            print("degree=" ,degree,"gamma=",gamma)
            clf = svm.SVC(kernel='poly', degree=degree,gamma=gamma)
            start_time = time.time()
            clf_fit = clf.fit(x_train, y_train)
            training_time=time.time() - start_time
            print("--- %s seconds ---" % (training_time))

            outfile = 'SVMpolymodel/' + str(degree) + '_' +str(gamma)
            with open(outfile, 'wb') as pickle_file:
                pickle_file = pickle.dump(clf, pickle_file)

            train_score = clf_fit.score(x_train, y_train)
            test_score = clf_fit.score(x_test, y_test)
            cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
            print('Train Accuracy=', train_score)
            print('Test Accuracy=', test_score)
            print(cm)

            outstr = str(degree) + " " + str(gamma) + " " + str(train_score) + " " + str(test_score)+' '+str(training_time)
            fh_out.write(outstr + "\n")
            fh_out.flush()
```

Listing 3. Ada Boosting

```
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import svm
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist
import seaborn as sns
import numpy as np
import time
import pickle

sns.set()

(x_train, y_train), (x_test, y_test) = mnist.load_data()
num_classes = 10
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# n=0.2
# x_train=x_train[1:int(60000*n)+1,]
# x_test=x_test[1:int(10000*n)+1,]
# y_train=y_train[1:int(60000*n)+1,]
# y_test=y_test[1:int(10000*n)+1,]



print('Train size:', x_train.shape[0])
print('Test size:', x_test.shape[0])

# Following code is too slow consider SGD
# clf = svm.LinearSVC()

# with open("Result/nist_tests_SVM.csv", "w") as fh_out:
# for alpha in [0.0001,0.001,0.01,0.1,10]:
# for l1_ratio in [0,0.2,0.4,0.6,0.8,1]:
# print("*", end="")
# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3,alpha=alpha,penalty='elasticnet',l1_ratio=l1_ratio,learning_rate='optimal')
# clf_fit=clf.fit(train_imgs, np.ravel(train_labels))
# train_score = clf_fit.score(train_imgs, np.ravel(train_labels))
# test_score = clf_fit.score(test_imgs, np.ravel(test_labels))
#
# outstr = str(alpha) + " " + str(l1_ratio) + " "
# outstr += str(train_score) + " " + str(test_score)
#
# fh_out.write(outstr + "\n")
# fh_out.flush()

# clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3)
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))
#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)


# clf = svm.SVC(kernel='linear')
# start_time = time.time()
# clf_fit=clf.fit(x_train, y_train)
# print("--- %s seconds ---" % (time.time() - start_time))

#
# train_score=clf_fit.score(x_train,y_train)
# test_score=clf_fit.score(x_test,y_test)
# cm=confusion_matrix(np.ravel(y_test),clf_fit.predict(x_test))
# print('Train Accuracy=',train_score)
# print('Test Accuracy=',test_score)
# print(cm)


with open("Result/nist_tests_SVM_rbf.csv", "w") as fh_out:
    for gamma in range(4):
        gamma = 10 ** -(gamma + 1)
        print("gamma=",gamma)
        clf = svm.SVC(kernel='rbf',gamma=gamma)
        start_time = time.time()
        clf_fit = clf.fit(x_train, y_train)
        training_time = time.time() - start_time
        print("--- %s seconds ---" % (training_time))

        outfile = 'SVMrbfmodel/' + str(gamma)
        with open(outfile, 'wb') as pickle_file:
            pickle_file = pickle.dump(clf, pickle_file)

        train_score = clf_fit.score(x_train, y_train)
        test_score = clf_fit.score(x_test, y_test)
        cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
        print('Train Accuracy=', train_score)
        print('Test Accuracy=', test_score)
        print(cm)
```

```
        outstr = str(gamma) + " " + str(train_score) + " " + str(test_score) +' '+str(training_time)
        fh_out.write(outstr + "\n")
        fh_out.flush()

with open("Result/nist_tests_SVM_poly.csv", "w") as fh_out:
    for degree in range(5):
        degree=degree+1
        for gamma in range(4):
            gamma=10**-(gamma)
            print("degree=" ,degree,"gamma=",gamma)
            clf = svm.SVC(kernel='poly', degree=degree,gamma=gamma)
            start_time = time.time()
            clf_fit = clf.fit(x_train, y_train)
            training_time=time.time() - start_time
            print("--- %s seconds ---" % (training_time))

            outfile = 'SVMpolymodel/' + str(degree) + '_' +str(gamma)
            with open(outfile, 'wb') as pickle_file:
                pickle_file = pickle.dump(clf, pickle_file)

            train_score = clf_fit.score(x_train, y_train)
            test_score = clf_fit.score(x_test, y_test)
            cm = confusion_matrix(np.ravel(y_test), clf_fit.predict(x_test))
            print('Train Accuracy=', train_score)
            print('Test Accuracy=', test_score)
            print(cm)

            outstr = str(degree) + " " + str(gamma) + " " + str(train_score) + " " + str(test_score)+' '+str(training_time )
            fh_out.write(outstr + "\n")
            fh_out.flush()
```

Listing 4. Gradient Boosting

```
import numpy as np
import time
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop,SGD
from sklearn.metrics import confusion_matrix
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'



(x_train, y_train), (x_test, y_test) = mnist.load_data()

num_classes = 10
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

print('Train size:', x_train.shape[0])
print('Test size:', x_test.shape[0])

# model = Sequential()
# model.add(Dense(120, input_shape=(784,)))
# model.add(Activation('relu'))
# model.add(Dense(num_classes))
# model.add(Activation('softmax'))
#
# for l in model.layers:
# print(l.output.name, l.input_shape, '==>', l.output_shape)
# print(model.summary())
#
# batch_size = 200
# epochs = 1
#
# model.compile(loss='mean_squared_error',
# optimizer=RMSprop(),
# metrics=['accuracy'])
#
# history = model.fit(x_train, y_train,
# batch_size=batch_size,
# epochs=epochs,
# verbose=1,
# validation_data=(x_test, y_test))
#
# score = model.evaluate(x_test, y_test, verbose=100)
#
# print('Test loss:', round(score[0], 3))
# print('Test accuracy:', round(score[1], 3))
#
# plt.plot(history.history['accuracy'])
# plt.plot(history.history['val_accuracy'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
# plt.show()
#
epochs=10
# with open("Result/nist_tests_keras.csv", "w") as fh_out:
# for hidden_nodes in [20, 50, 100, 150, 250, 500]:
# for activation in ['softmax','relu']:
# for batch_size in [100, 200, 300]:
# for loss in ['mean_squared_error','categorical_crossentropy']:
# for learning_rate in [0.001,0.01]:
# model = Sequential()
# model.add(Dense(hidden_nodes, activation=activation, input_shape=(784,)))
# model.add(Dense(num_classes, activation='softmax'))
# model.compile(loss=loss,
# optimizer=RMSprop(learning_rate=learning_rate),
# metrics=['accuracy'])
#
# for l in model.layers:
# print(l.name, l.input_shape, '==>', l.output_shape, 'Activation=', activation)
# print('Loss=', loss)
# print('batch size=', batch_size)
# print(model.summary())
#
# history = model.fit(x_train, y_train,
# batch_size=batch_size,
# epochs=epochs,
# verbose=2,
# validation_data=(x_test, y_test))
#
#
# # score = model.evaluate(x_test, y_test, verbose=100)
# # print(history.history['accuracy'])
#
# for epoch in range(epochs):
# outstr = str(hidden_nodes) + " " + str(activation) + " " + str(batch_size )+ " " + str(loss)+ " " + str(learning_rate)
# outstr += " " + str(epoch) + " "+ str(history.history['accuracy'][epoch ]) + " "+ str(history.history['val_accuracy'][epoch])
# outstr += " "+ str(history.history['loss'][epoch]) + " "+ str(history.history['val_loss'][epoch])
# fh_out.write(outstr + "\n")
# fh_out.flush()


hidden_nodes1=50
activation='relu'
```

```
loss='mean_squared_error'
batch_size=200
learning_rate=0.001
model = Sequential()
model.add(Dense(hidden_nodes1, activation=activation, input_shape=(784,)))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='mean_squared_error',
              optimizer=RMSprop(learning_rate=learning_rate),
              metrics=['accuracy'])

for l in model.layers:
    print(l.name, l.input_shape, '==>', l.output_shape, 'Activation=', activation)
print('Loss=', loss)
print('batch size=', batch_size)
print(model.summary())

start_time = time.time()
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=2,
                    validation_data=(x_test, y_test))
training_time=time.time() - start_time
print("--- %s seconds ---" % (training_time))

def hot_to_cat(y_test):
    decoded_datum = np.zeros((len(y_test), 1), int)
    for i in range(len(y_test)):
        decoded_datum[i,] = np.argmax(y_test[i])
    return decoded_datum


cm=confusion_matrix(hot_to_cat(y_test), model.predict_classes(x_test))
print(cm)
```

Listing 5. 1-layer Neural Network

```
import numpy as np
import time
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop,SGD
from sklearn.metrics import confusion_matrix
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'



(x_train, y_train), (x_test, y_test) = mnist.load_data()

num_classes = 10
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

print('Train size:', x_train.shape[0])
print('Test size:', x_test.shape[0])

# model = Sequential()
# model.add(Dense(120, input_shape=(784,)))
# model.add(Activation('relu'))
# model.add(Dense(num_classes))
# model.add(Activation('softmax'))
#
# for l in model.layers:
# print(l.output.name, l.input_shape, '==>', l.output_shape)
# print(model.summary())
#
# batch_size = 200
# epochs = 1
#
# model.compile(loss='mean_squared_error',
# optimizer=RMSprop(),
# metrics=['accuracy'])
#
# history = model.fit(x_train, y_train,
# batch_size=batch_size,
# epochs=epochs,
# verbose=1,
# validation_data=(x_test, y_test))
#
# score = model.evaluate(x_test, y_test, verbose=100)
#
# print('Test loss:', round(score[0], 3))
# print('Test accuracy:', round(score[1], 3))
#
# plt.plot(history.history['accuracy'])
# plt.plot(history.history['val_accuracy'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
# plt.show()
#
# epochs=10
# activation='relu'
# with open("Result/nist_tests_keras_2.csv", "a") as fh_out:
# for hidden_nodes1 in [20, 50, 100, 150, 250, 500]:
# for hidden_nodes2 in [20, 50, 100, 150, 250, 500]:
# for batch_size in [100, 200, 300]:
# for loss in ['mean_squared_error','categorical_crossentropy']:
# for learning_rate in [0.001,0.01]:
# model = Sequential()
# model.add(Dense(hidden_nodes1, activation=activation, input_shape=(784,)))
# model.add(Dense(hidden_nodes2, activation=activation))
# model.add(Dense(num_classes, activation='softmax'))
# model.compile(loss=loss,
# optimizer=RMSprop(learning_rate=learning_rate),
# metrics=['accuracy'])
#
# for l in model.layers:
# print(l.name, l.input_shape, '==>', l.output_shape, 'Activation=', activation)
# print('Loss=', loss,'batch size=', batch_size)
# print(model.summary())
#
# history = model.fit(x_train, y_train,
# batch_size=batch_size,
# epochs=epochs,
# verbose=2,
# validation_data=(x_test, y_test))
#
#
# # score = model.evaluate(x_test, y_test, verbose=100)
# # print(history.history['accuracy'])
#
# for epoch in range(epochs):
# outstr = str(hidden_nodes1) + " " +str(hidden_nodes2) + " " + str(activation) + " " + str(batch_size)+ " " + str(loss)+ " " + str(learning_rate)
# outstr += " " + str(epoch) + " "+ str(history.history['accuracy'][epoch]) + " "+ str(history.history['val_accuracy'][epoch])
# outstr += " "+ str(history.history['loss'][epoch]) + " "+ str(history.history['val_loss'][epoch])
# fh_out.write(outstr + "\n")
# fh_out.flush()

epochs=10
```

```python
hidden_nodes1=500
hidden_nodes2=150
activation='relu'

loss='mean_squared_error'
batch_size=200
learning_rate=0.001
model = Sequential()
model.add(Dense(hidden_nodes1, activation=activation, input_shape=(784,)))
model.add(Dense(hidden_nodes2, activation=activation))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='mean_squared_error',
              optimizer=RMSprop(learning_rate=learning_rate),
              metrics=['accuracy'])

for l in model.layers:
    print(l.name, l.input_shape, '==>', l.output_shape, 'Activation=', activation)
print('Loss=', loss)
print('batch size=', batch_size)
print(model.summary())

start_time = time.time()
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=2,
                    validation_data=(x_test, y_test))
training_time=time.time() - start_time
print("--- %s seconds ---" % (training_time))

def hot_to_cat(y_test):
    decoded_datum = np.zeros((len(y_test), 1), int)
    for i in range(len(y_test)):
        decoded_datum[i,] = np.argmax(y_test[i])
    return decoded_datum


cm=confusion_matrix(hot_to_cat(y_test), model.predict_classes(x_test))
print(cm)
```

Listing 6. 2-layer Neural Network

Appendix B
R Source Code

```r
accuracies = read.csv('Result/nist_tests.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('hidden_nodes', 'learning_rate', 'bias', 'epoch', 'train corrects rate', 'train wrongs rate', 'test corrects rate', 'test wrongs rate')
accuracies = accuracies[, -c(6, 8)]
accuracies = reshape2::melt(accuracies, id.vars = c('hidden_nodes', 'learning_rate', 'bias', 'epoch'))
accuracies[c('hidden_nodes', 'learning_rate', 'bias')]=lapply(accuracies[c('hidden_nodes', 'learning_rate', 'bias')], factor)

library(ggplot2)
library(ggrepel)
dt=accuracies
for (x in c('bias')) {
  ggplot(dt,aes(dt[[x]],value,fill=variable))+
    geom_boxplot() +
    xlab(x)+
    stat_summary(fun.y=max, geom="line", aes(group=variable,color=variable))+
    stat_summary(fun.y=max, geom="point", aes(group=variable,color=variable))+
    stat_summary(geom="label_repel", fun.y=max,aes(group=variable,label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
    theme_bw()+ theme(legend.position="bottom")+ylab('Accuracy')
    # title('hidden_nodes=20, 50, 100, 120, 150\n learning_rate=0.01, 0.05, 0.1, 0.2\n bias=None, 0.5 epoch=1,2,3')
  ggsave(paste('1-Hidden Layer Neural Network Accuracy rate versus ',x,'.png',sep = ''), path ='../Report/figure', scale = 0.6)
}




# accuracies = read.csv('Result/nist_tests_Multiple.csv', sep = ' ', header = FALSE)
# colnames(accuracies) = c('layer_one_nodes','layer_two_nodes','train corrects rate','train wrongs rate','test corrects rate','test wrongs rate')
# accuracies = accuracies[, -c(4, 6)]
# accuracies = reshape2::melt(accuracies, id.vars = c('layer_one_nodes', 'layer_two_nodes'))
#
# ggplot(accuracies, aes(layer_one_nodes, layer_two_nodes,color=variable, shape = variable,size=value,label=round(value,4))) +
# geom_point(alpha=0.6)+
# ggrepel::geom_text_repel(size = 3,alpha=0.6,box.padding=3,segment.alpha=0.6)+
# ylab('Accuracy')+
# theme_bw()+ theme(legend.position="bottom")
# ggsave(paste('2-Hidden Layer Neural Network Accuracy rate, layer_one_nodes versus layer_two_nodes.png',sep = ''), path ='../Report/figure', scale = 0.6)


accuracies = read.csv('Result/nist_tests_keras.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('hidden_nodes','activation','batch_size','loss','learning_rate','epoch','train_accuracy','test_accuracy','train_loss','test_loss')
accuracies = reshape2::melt(accuracies,measure.vars = c('train_accuracy','test_accuracy','train_loss','test_loss'), id.vars = c('hidden_nodes','activation','batch_size','loss','learning_rate','epoch'))
accuracies=tidyr::unite(accuracies, col, 'hidden_nodes':'learning_rate', sep = "_", remove = FALSE, na.rm = FALSE)
accuracies[c('hidden_nodes','activation','batch_size','loss','learning_rate')]=lapply(accuracies[c('hidden_nodes','activation','batch_size','loss','learning_rate')], factor)
accuracies=subset(accuracies,learning_rate!=0.1)

dt=subset(accuracies, variable %in% c('test_accuracy')&activation=='softmax')
ggplot(dt, aes(epoch, value, group=col,color=hidden_nodes)) +
  geom_line()+
  xlim(0,9)+
  ggrepel::geom_label_repel(data=subset(dt, value<0.91&epoch==9),aes(epoch, value,label=col),color='black',force = 10,alpha=0.7, xlim=c(0, 8),size=2)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")
ggsave(paste('1-layer Neural Network Test Accuracy By Epoch.png',sep = ''), path ='../Report/figure', scale = 0.6)

dt=subset(accuracies, variable %in% c('test_accuracy')&activation=='relu')
ggplot(dt, aes(epoch, value, group=col,color=hidden_nodes)) +
  geom_line()+
  xlim(0,9)+
  ggrepel::geom_label_repel(data=subset(dt, value<0.91&epoch==9),aes(epoch, value,label=col),color='black',force = 10,alpha=0.7, xlim=c(0, 8),size=2)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")
```

```r
ggsave(paste('91-layer Neural Network Test Accuracy By Epoch.png', sep = ''), path ='../Report/figure', scale = 0.6)


dt=subset(accuracies, variable %in% c('train_accuracy','test_accuracy')& epoch==9&activation=='softmax')
for (x in c('hidden_nodes','batch_size','loss','learning_rate')) {
  ggplot(dt,aes(dt[[x]],value,fill=variable))+
    geom_boxplot(outlier.shape = NA) +
    xlab(x)+
    stat_summary(fun.y=max, geom="line", aes(group=variable,color= variable))+
    stat_summary(fun.y=max, geom="point", aes(group=variable,color= variable))+
    stat_summary(geom="label_repel", fun.y=max,aes(group=variable, label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
    ylim(0.9,1)+
    theme_bw()+ theme(legend.position="bottom")+ylab('Accuracy')
  ggsave(paste('1-Hidden Layer Neural Network Accuracy rate versus ' ,x,'.png',sep = ''), path ='../Report/figure', scale = 0.6)
}

dt=subset(accuracies, variable %in% c('train_accuracy','test_accuracy')& epoch==9&activation=='relu')
for (x in c('hidden_nodes','batch_size','loss','learning_rate')) {
  ggplot(dt,aes(dt[[x]],value,fill=variable))+
    geom_boxplot(outlier.shape = NA) +
    xlab(x)+
    stat_summary(fun.y=max, geom="line", aes(group=variable,color= variable))+
    stat_summary(fun.y=max, geom="point", aes(group=variable,color= variable))+
    stat_summary(geom="label_repel", fun.y=max,aes(group=variable, label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
    ylim(0.9,1)+
    theme_bw()+ theme(legend.position="bottom")+ylab('Accuracy')
  ggsave(paste('91-Hidden Layer Neural Network Accuracy rate versus ',x,'.png',sep = ''), path ='../Report/figure', scale = 0.6)
}

#########Acti Comparison######

dt=subset(accuracies, variable %in% c('train_accuracy','test_accuracy')& epoch==9)
x='activation'
ggplot(dt,aes(dt[[x]],value,fill=variable))+
  geom_boxplot(outlier.shape = NA) +
  xlab(x)+
  stat_summary(fun.y=max, geom="line", aes(group=variable,color= variable))+
  stat_summary(fun.y=max, geom="point", aes(group=variable,color= variable))+
  stat_summary(geom="label_repel", fun.y=max,aes(group=variable,label =sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
  ylim(0.9,1)+
  theme_bw()+ theme(legend.position="bottom")+ylab('Accuracy')
ggsave(paste('1-Hidden Layer Neural Network Accuracy rate versus ',x, '.png',sep = ''), path ='../Report/figure', scale = 0.6)

dt=subset(accuracies, variable %in% c('test_accuracy'))
ggplot(dt, aes(epoch, value, group=col,color=activation)) +
  geom_line(alpha=0.4)+
  xlim(0,9)+
  # ggrepel::geom_label_repel(data=subset(dt, value<0.91&epoch==9), aes(epoch, value,label=col),color='black',force = 10,alpha =0.7,xlim=c(0, 8),size=3)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")
ggsave(paste('1-Hidden layer Neural Network Accuracy rate By Epoch. png',sep = ''), path ='../Report/figure', scale = 0.6)























accuracies = read.csv('Result/nist_tests_keras_2.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('hidden_nodes1','hidden_nodes2','activation','batch_size','loss','learning_rate','epoch','train_accuracy','test_accuracy','train_loss','test_loss')
accuracies = reshape2::melt(accuracies,measure.vars = c('train_accuracy', 'test_accuracy','train_loss','test_loss'), id.vars = c('hidden_nodes1','hidden_nodes2','activation','batch_size','loss','learning_rate','epoch'))
accuracies=tidyr::unite(accuracies, col, 'hidden_nodes1','hidden_nodes2': 'learning_rate', sep = "_", remove = FALSE, na.rm = FALSE)
accuracies[c('hidden_nodes1','hidden_nodes2','activation','batch_size','loss','learning_rate')]=lapply(accuracies[c('hidden_nodes1','hidden_nodes2','activation','batch_size','loss','learning_rate')], factor)
accuracies=subset(accuracies,learning_rate!=0.1)

# library(plotly)
# plot_ly(x=temp, y=pressure, z=dtime, type="scatter3d", mode="markers", color=temp)
library(gridExtra)

dt=subset(accuracies, variable %in% c('test_accuracy')&activation=='relu')
p <- list()
for(i in 1:6){
  values = c(0.1,0.1,0.1,0.1,0.1,0.1)
  values[i] = 1
  p[[i]]=ggplot(dt, aes(epoch, value, group=col,color=hidden_nodes1, alpha=hidden_nodes1)) +
    geom_line()+
    xlim(0,9)+
    # stat_summary(geom="label_repel", fun.y=max,aes(group=variable, label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
    ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")+ ylim(0.85,1)+scale_alpha_manual(values = values)
}
ggsave(paste('2-layer Neural Network Test Accuracy By Epoch.png',sep = ''), plot=ggarrange(plotlist=p,ncol=6),path ='../Report/figure', scale = 2)

p <- list()
for(i in 1:6){
  values = c(0.1,0.1,0.1,0.1,0.1,0.1)
  values[i] = 1
  p[[i]]=ggplot(dt, aes(epoch, value, group=col,color=hidden_nodes2, alpha=hidden_nodes2)) +
  geom_line()+
  xlim(0,9)+
```

```
  # stat_summary(geom="label_repel", fun.y=max,aes(group=variable, label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")+ylim(0.85,1)+scale_alpha_manual(values = values)
}
ggsave(paste('2-layer Neural Network Test Accuracy By Epoch2.png', sep = ''), plot=ggarrange(plotlist=p,ncol=6),path ='../Report/figure', scale = 2)


dt=subset(accuracies, variable %in% c('train_accuracy','test_accuracy')& epoch==9&activation=='relu')
for (x in c('hidden_nodes1','hidden_nodes2','batch_size','loss','learning_rate')) {
  ggplot(dt,aes(dt[[x]],value,fill=variable))+
    geom_boxplot(outlier.shape = NA) +
    xlab(x)+
    stat_summary(fun.y=max, geom="line", aes(group=variable,color=variable))+
    stat_summary(fun.y=max, geom="point", aes(group=variable,color=variable))+
    stat_summary(geom="label_repel", fun.y=max,aes(group=variable, label=sprintf("%1.4f", ..y..),alpha=0.8), size=3.5)+
    ylim(0.9,1)+
    theme_bw()+ theme(legend.position="bottom")+ylab('Accuracy')
  ggsave(paste('2-Hidden Layer Neural Network Accuracy rate versus ',x,'.png',sep = ''), path ='../Report/figure', scale = 0.6)
}



















#########SVM##########
accuracies = read.csv('Result/nist_tests_SVM.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('alpha','l1_ratio','train_accuracy','test_accuracy')
accuracies = reshape2::melt(accuracies, id.vars = c('alpha', 'l1_ratio'))


ggplot(accuracies, aes(alpha, l1_ratio, color = variable, shape = variable, size=value,label=round(value,4))) +
  geom_point(alpha=0.6)+
  ggrepel::geom_text_repel(size = 3,alpha=0.6)+
  scale_x_log10(
    breaks = scales::trans_breaks("log10", function(x) 10^x),
    labels = scales::trans_format("log10", scales::math_format(10^.x))
  )+
  theme_bw()+ theme(legend.position="bottom")
ggsave(paste('SGD SVM Accuracy rate, alpha versus l1_ratio.png',sep = ''), path ='../Report/figure', scale = 0.6)




accuracies = read.csv('Result/nist_tests_SVM_rbf.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('gamma','train_accuracy','test_accuracy','time')
accuracies = reshape2::melt(accuracies, id.vars = c('gamma','time'))


ggplot(accuracies, aes(gamma, value, color = variable,label=round(value,4))) +
  geom_line(alpha=0.6)+
  geom_label(size = 3,alpha=0.6)+
  scale_x_log10(
    breaks = scales::trans_breaks("log10", function(x) 10^x),
    labels = scales::trans_format("log10", scales::math_format(10^.x))
  )+
  theme_bw()+ theme(legend.position="bottom")
ggsave(paste('RBF SVM Accuracy versus gamma.png',sep = ''), path ='../Report/figure', scale = 0.6)




accuracies = read.csv('Result/nist_tests_SVM_poly.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('degree','gamma','train_accuracy','test_accuracy','time')
accuracies = reshape2::melt(accuracies, id.vars = c('degree','gamma','time'))


ggplot(accuracies, aes(gamma, degree, color = variable, shape = variable, size=value,label=round(value,4))) +
  geom_point(alpha=0.6)+
  ggrepel::geom_text_repel(size = 3,alpha=0.6)+
  scale_x_log10(
    breaks = scales::trans_breaks("log10", function(x) 10^x),
    labels = scales::trans_format("log10", scales::math_format(10^.x))
  )+
  theme_bw()+ theme(legend.position="bottom")
ggsave(paste('Polynomial SVM Accuracy degree versus gamma.png',sep = ''), path ='../Report/figure', scale = 0.6)



















#######boost#######
accuracies = read.csv('Result/nist_tests_ada.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('n_estimators','learning_rate','Iteration','train_accuracy','test_accuracy','time')
accuracies = reshape2::melt(accuracies, id.vars = c('n_estimators','learning_rate','Iteration','time'))

dt=subset(accuracies,n_estimators==300,c('learning_rate','Iteration','value','variable'))
dt$learning_rate=factor(dt$learning_rate)
p=ggplot(dt, aes(Iteration,value , alpha= variable,shape = variable,color=learning_rate) )+
  geom_line()+
  geom_point(size=1)+
  ggrepel::geom_label_repel(data=subset(dt, value<0.91&Iteration%in%c(50,150,299)),aes(Iteration, value,label=round(value,4)),color='black',size=3)+
```

```
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")+ scale_shape( solid = FALSE)
ggsave(paste('Ada Boosting Accuracy By Iteration.png',sep = ''),p, path ='../Report/figure', scale = 0.6)

accuracies = read.csv('Result/nist_tests_ada_final.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('n_estimators','learning_rate','Iteration','train_accuracy','test_accuracy','time')
accuracies = reshape2::melt(accuracies, id.vars = c('n_estimators','learning_rate','Iteration','time'))

dt=subset(accuracies,n_estimators==300,c('learning_rate','Iteration','value','variable'))
dt$learning_rate=factor(dt$learning_rate)
p=ggplot(dt, aes(Iteration,value ,shape = variable,color=variable) )+
  geom_line()+
  geom_point(size=1)+
  ggrepel::geom_label_repel(data=subset(dt, Iteration%in%c(50,150,299)),aes(Iteration, value,label=round(value,4)),color='black',size=3)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")+ scale_shape( solid = FALSE)
ggsave(paste('Final Ada Boosting Accuracy By Iteration.png',sep = ''),p, path ='../Report/figure', scale = 0.6)




accuracies = read.csv('Result/nist_tests_gra.csv', sep = ' ', header = FALSE)
colnames(accuracies) = c('n_estimators','learning_rate','tree_depth','Iteration','train_accuracy','test_accuracy','time')
accuracies = reshape2::melt(accuracies, id.vars = c('n_estimators','learning_rate','Iteration','tree_depth','time'))

dt=subset(accuracies,n_estimators==100,c('tree_depth','learning_rate','Iteration','value','variable'))
dt$learning_rate=factor(dt$learning_rate)
dt$tree_depth=factor(dt$tree_depth)
p=ggplot(dt, aes(Iteration,value , alpha= variable, shape = tree_depth, color=learning_rate:tree_depth) )+
  geom_line()+
  geom_point(size=1)+
  ggrepel::geom_label_repel(data=subset(dt,Iteration%in%c(10,50,99)), aes(Iteration, value,label=round(value,4)),size=3)+
  # ggrepel::geom_text_repel(size = 3,alpha=0.6)+
  ylab('Accuracy')+theme_bw()+ theme(legend.position="bottom")+ scale_shape( solid = FALSE)
ggsave(paste('Gradient Boosting Accuracy By Iteration.png',sep = ''), p, path ='../Report/figure', scale = 1.2)
```

Listing 7. Result Visualization