Recognizing handwritten digits by using different neural networks

Rouyu Chen

Computer Science and Information Technology Building, Australian National University, Canberra ACT, Australia

u6539353@anu.edu.au

Abstract. This paper used the optical recognition of handwritten digits data set to distinguish 10 different digits on the bais of 64 input attributes. We used 5 different neural networks to estimate and improve the final accuracy. Firstly, we implemented a simple three-layer network and estimated the training and testing accuracy. Secondly, we changed the number of epoch and learning rate separately. And then we changed the optimiser from Stochastic Gradient Descent (SGD) to Momentum and Adaptive Moment Estimation (Adam). After that, we applied the shared weights and auto-assoiciative topologies. Finally, we built a Convolutional Neural Network (CNN). After each time of using method, we compared the results with the accuracy of the simple three-layer network. The result shows that changing the learning rate, using Adam optimiser and using CNN will improve the performance a lot.

Keywords: Handwritten Digits; Neural Network; Shared Weights; Auto-associative Network; Convolutional Neural Network; Overfitting.

1 Introduction

1.1 Data Set

The data set we used in this report is the optical recognition of handwritten digits data set (Alpaydin & Kaynak, 1998). There are 64 input attributes and 1 class attribute, the class attributes are the code from 0 to 9. Alpaydin and Kaynak (1998) had used a multistage method to test the real-world application of handwritten digit recognition. Xu, Krzyzak, and Suen (1992) tried four approaches to solve the problem of combining the classification powers of classifiers.

We chose this data set for three reasons. First, this dataset contains 5620 instances and 64 input attributes, which means we have sufficient data to train different networks. Second, this dataset was preprocessed once, the data had been separated to two parts, 68% for training and 32% for testing, so we don't need to split the training and testing data again. Third, this dataset comes from a series of handwritten digits images, but we don't need to process the images because the normalized bitmaps of images had been extracted, so we don't need to consider how to process the images.

1.2 Method Analysis

We used four approaches to analyze the results in this report. First, we used training accuracy, training accuracy is a good presentation of how well our neural network has trained. Second, we used testing accuracy, because we don't test the network after training, we may get an overfitting problem. Third, we used the confusion matrix. Confusion matrix could show us clearly how much attributes are classified right and how much are classified wrong (Brownlee, 2016). Finally, we used the loss to estimate the performance of some networks, the loss is especially useful when estimating the performance of regression process.

2. Method

2.1 Data Preprocessing

Firstly, the data will be processed by using pandas, and training data will only be transferred to numeric directly, because the target values are not in string type. Secondly, pandas dataframe will be converted to array, x_array denotes the first 64 input attributes, y_array denotes the target attributes. Finally, Tensors will be created to store both x_array and y_array, and then they will be wrapped by Variables and become the data being used in the network.

2.2 Implement a simple three-layer network

The first step is to implement a simple three-layer network. This network contains an input layer, a hidden layer and an output layer. There are 64 neurons, 10 neurons, and 10 neurons in them separately. The learning rate is 0.01, number of epoch is 500. Sigmoid is used as activation function. Stochastic Gradient Descent (SGD) is used as the optimizer. Cross-entropy is used as the tool to evaluate the performance.

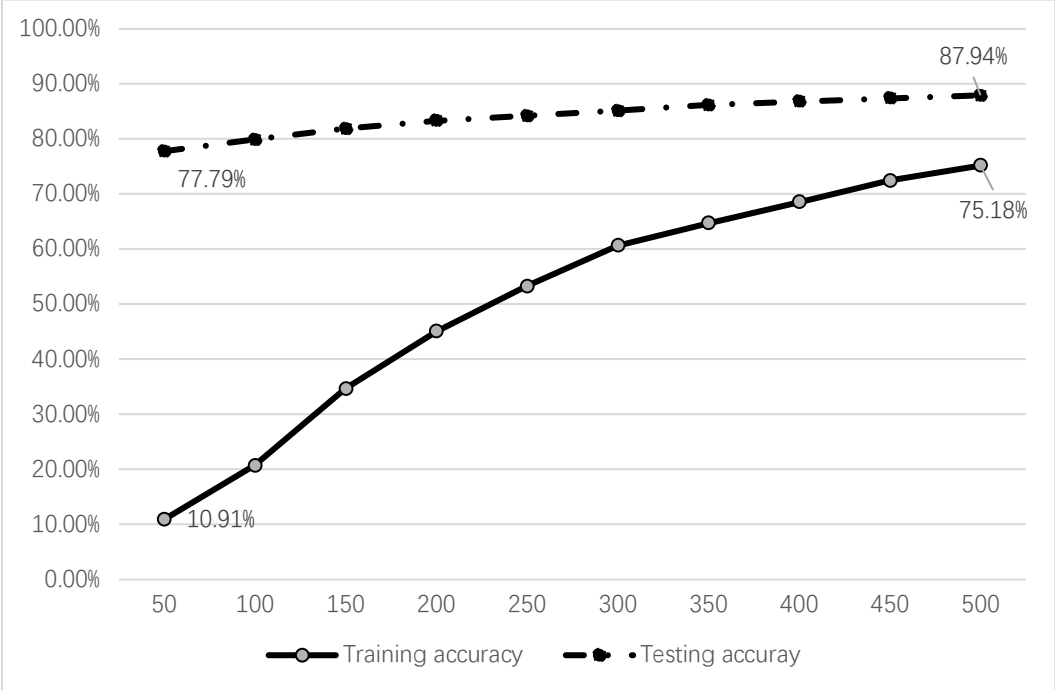

Chart 2.1 Accuracy of training and testing dataset

As is shown in the Chart 2.1, the training accuracy starts at a lower percentage (10.91%), while the testing accuracy starts at a higher one (77.79%). The rate of increase of the training dataset is higher than the testing one, we could see the steeper slope of the training accuracy. After 500 epochs, the gap between these two types of accuracy become smaller. In general, the testing accuracy is higher than the training accuracy. The confusion matrix of testing shows the result of testing. As we can see, most of the data are located in the diagonal, which means most actual digits are correctly classified. In conclusion, the network could classify the data for now.

172	1	2	0	1	0	0	0	0	1
0	163	13	0	1	0	2	0	2	2
1	1	168	3	0	1	4	1	1	0
0	0	0	179	0	2	0	0	0	4
0	2	1	0	158	0	11	4	0	3
1	2	3	6	0	152	2	0	0	6
0	1	2	0	0	0	178	0	0	0
0	0	0	1	2	0	0	179	0	3
2	29	2	4	5	3	2	1	117	1
1	3	0	23	7	1	0	12	0	142

Table 2.2 Confusion matrix for testing

2.3 Change the epoch and learning rate on the basis of 2.2

In order to increase the accuracy, we try to make some changes to the epoch and the learning rate on the basis of the simple three-layer network.

Firstly, we increased the number of epoch from 500 to 1500. As is shown in Chart x.x, the training accuracy was still lower than testing accuracy at the beginning. However, both of them showed a higher accuracy than before. The testing accuracy showed a smooth slope with higher rate, while the training accuracy had some fluctuations before it arrived at a steady level. The speed of growth decreased twice in around 100th and 350thepoch, then it increased again some epochs. Both of the accuracy rates were increased after changing the epochs.

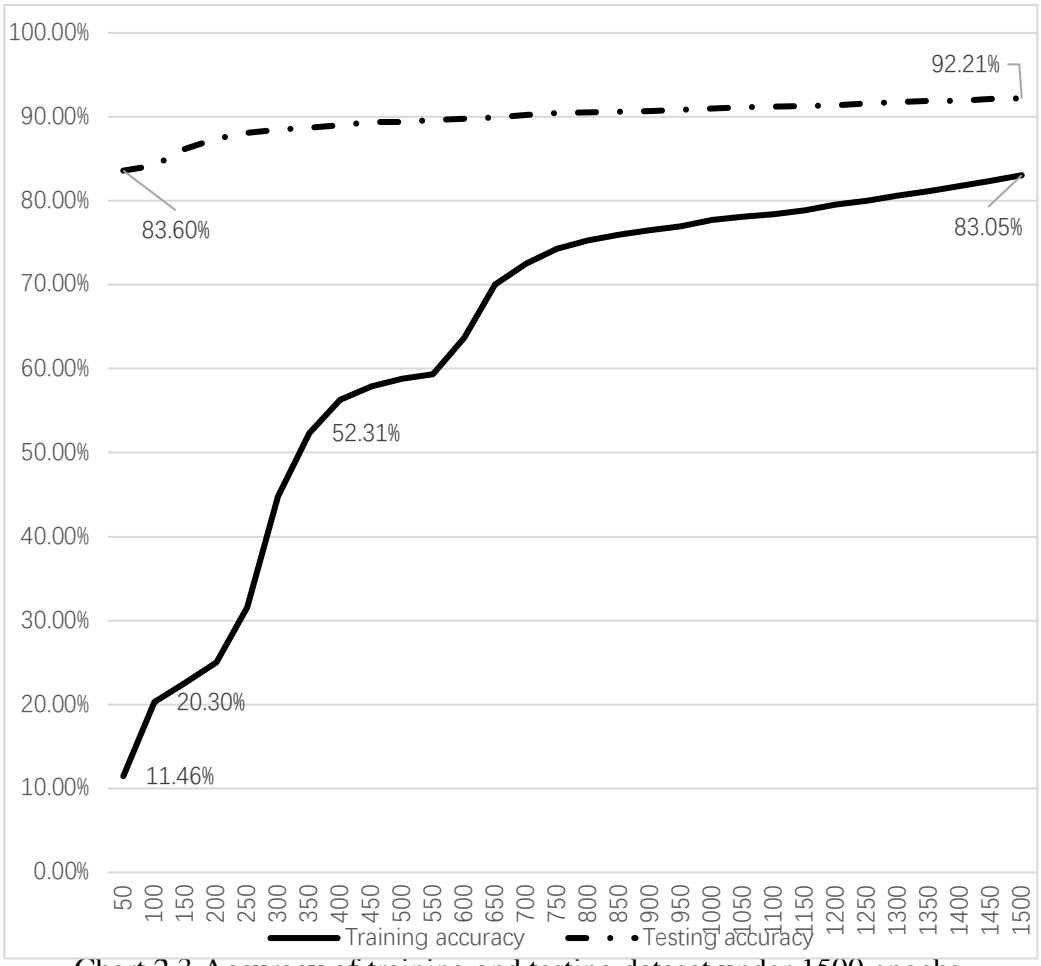

Chart 2.3 Accuracy of training and testing dataset under 1500 epochs

Secondly, we made the change to the learning rate. Zulkifli (2018) identifies that learning rate is an important hyper-parameter to train neural networks, it decides how quickly our model can reach to the best accuracy. If our learning rate is better, it could improve our performance of accuracy. For comparison, we will change the epoch back to 500 and then change the learning rate from 0.01 to 0.5.

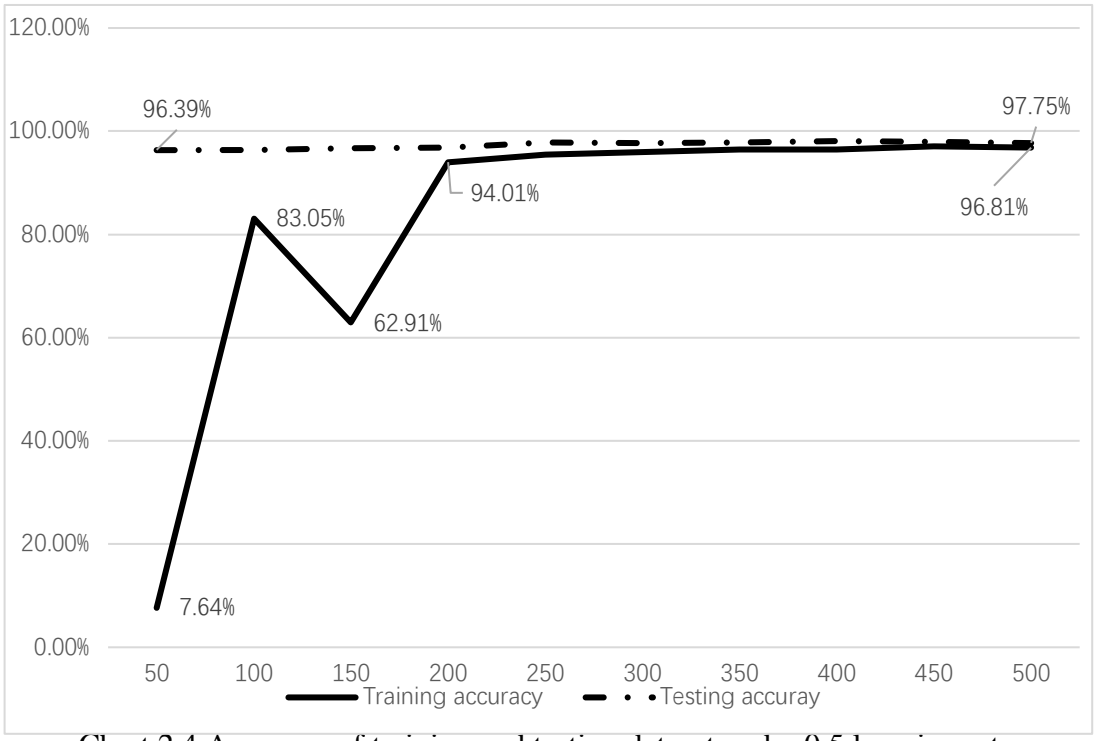

Chart 2.4 Accuracy of training and testing dataset under 0.5 learning rate

The chart shows that the gap between the training and testing accuracy at the beginning become larger. The slope of the testing accuracy kept smooth all the time, while the slope of the training accuracy was fluctuated before the 200th epoch. After 500 epochs, the training accuracy and testing accuracy reached 96.81% and 97.75% separately. Both of the accuracy rates were increased after increasing the learning rate from 0.01 to 0.5.

2.4 Change the optimizer on the basis of 2.2

According to Ruder (2016), the gradient descent is not always able to make good convergence, sometimes it will generate some problems. Some algorithms are used to optimize the gradient descent. To address the problem and improve the performance, we changed the optimizer from Stochastic Gradient Descent (SGD) to Momentum and Adaptive Moment Estimation (Adam). For better comparing the results, we kept the simple three-layer network structure unchanged.

Firstly, we changed the optimizer from Stochastic Gradient Descent (SGD) to the Momentum. According to MacLeod (2010), “Local Minima” is one of the best-known problems with back-propagation. If the local minima problem appears, we may not find the actual lowest error that we really want to find. One of the solutions is to add Momentum to the weight change. If the Momentum works, it will generate a better performance of the network.

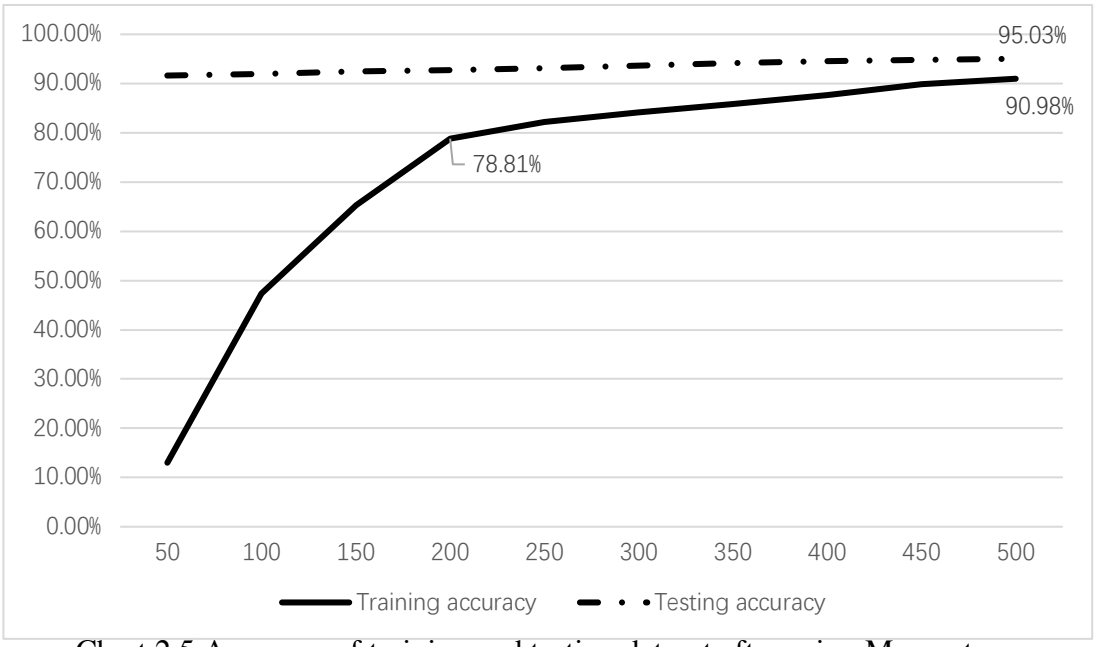

Chart 2.5 Accuracy of training and testing dataset after using Momentum

After using the Momentum optimizer, the training accuracy increased from 75.18% in the network in 2.2 to 90.98%, and the testing accuracy increased from 87.94% in the network in 2.2 to 95.03%. After around 200 epochs, the training accuracy reached 78.81%, and its growth speed started to slow down. Overfitting didn't show up. The performance of using Momentum is better than that of using Stochastic Gradient Descent (SGD).

Secondly, we changed the optimizer from Momentum to Adaptive Moment Estimation (Adam). Kingma and Ba (2015) report that Adaptive Moment Estimation (Adam) works practically better when comparing to other adaptive learning-method algorithms.

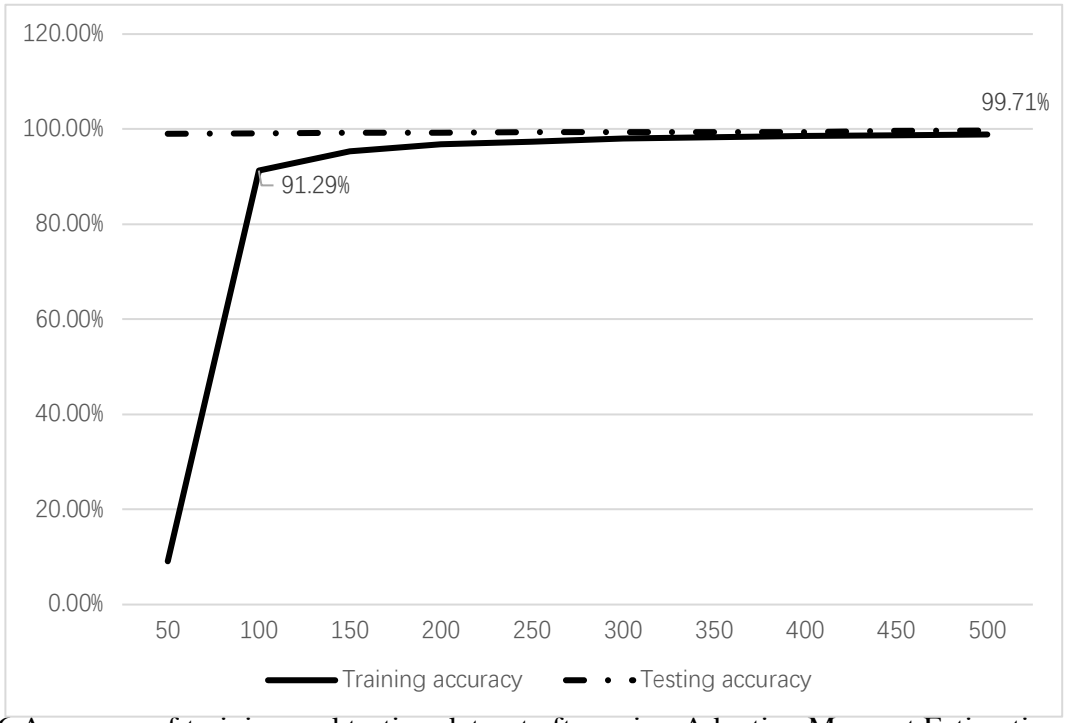

Chart 2.6 Accuracy of training and testing dataset after using Adaptive Moment Estimation (Adam)

With the Adaptive Moment Estimation (Adam) optimizer, both the testing and training accuracy were higher and reached to a very close percentage (99.71%). Comparing to the Momentum

optimizer, the Adaptive Moment Estimation (Adam) optimizer boosted the speed of training. After only around 100 epochs, the growth speed of training accuracy started to slow down. In addition, the Adaptive Moment Estimation (Adam) generated more increment in both training and testing accuracy comparing to the Momentum optimizer.

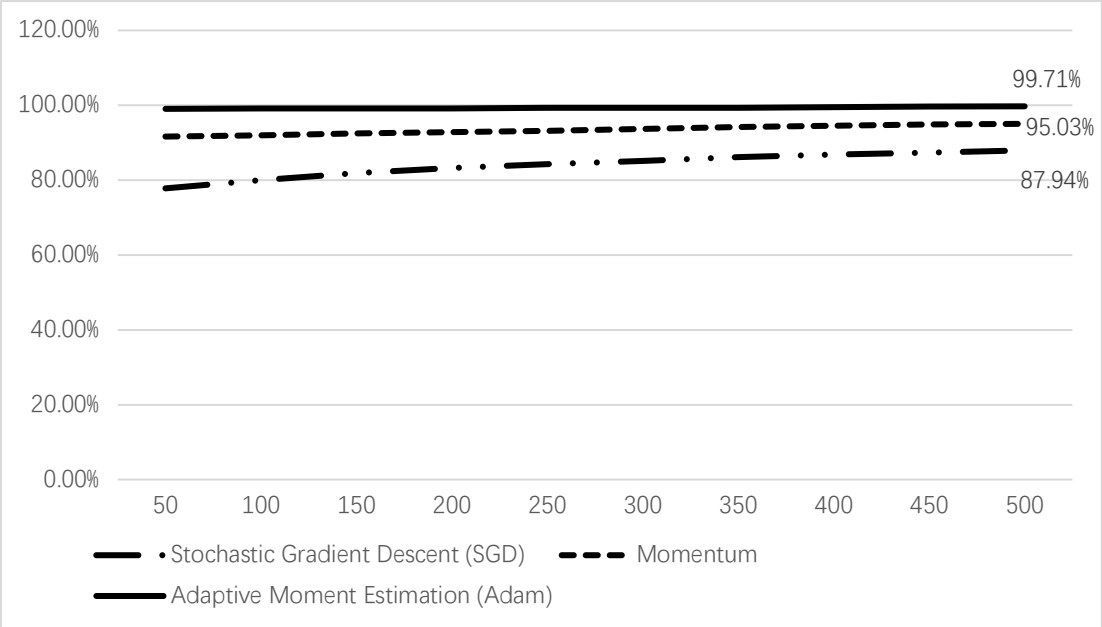

Chart 2.7 Comparing the accuracy of the testing dataset after using Stochastic Gradient Descent (SGD), Momentum and Adaptive Moment Estimation (Adam)

As is shown in Chart 2.7, by comparing the accuracy of the testing accuracy after using Stochastic Gradient Descent (SGD), Momentum and Adaptive Moment Estimation (Adam), we could find that Adaptive Moment Estimation (Adam) generated better results than other two optimizers.

2.5 Using auto-associative network with shared weights on the basis of 2.2

Shared weight and auto-associative topologies are used to compress image (Gedeon, Catalan, & Jin, 1997). We made some changes on the basis of these two topologies. We assumed that by using these two topologies, we could improve the performance of neural network. The hypothetical neural network is shown in Diagram 2.8.

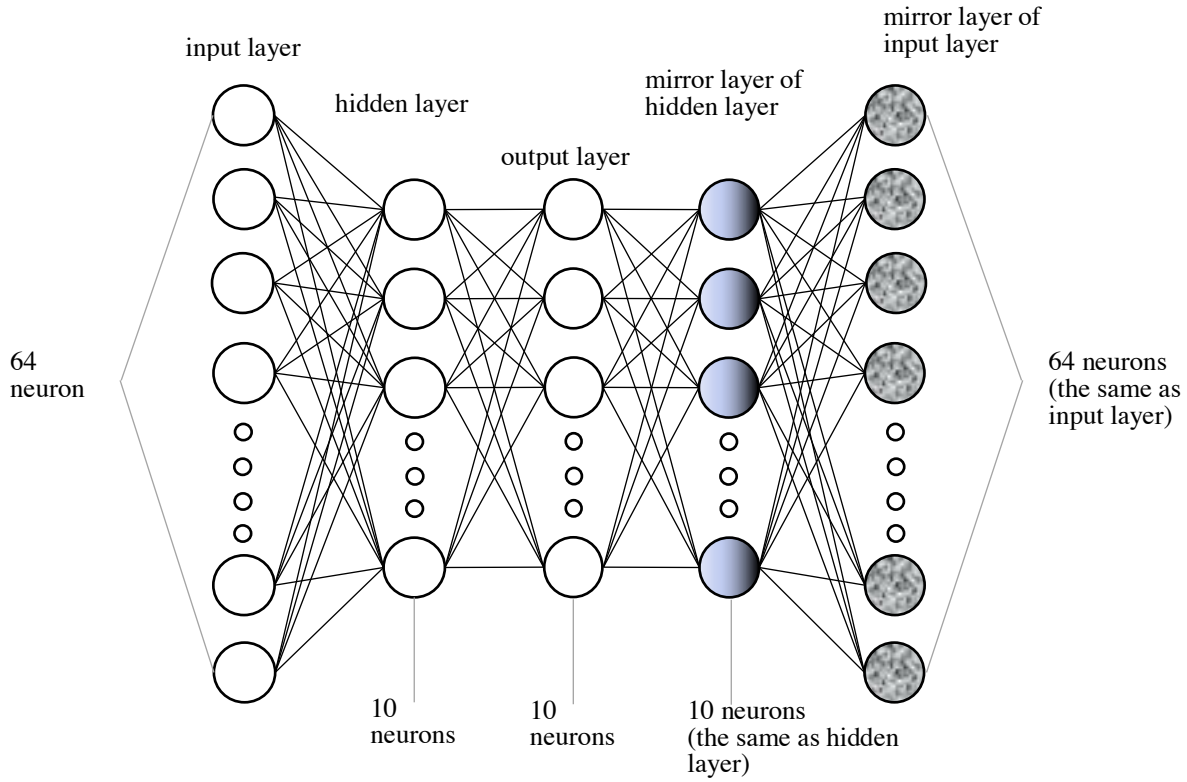

Diagram 2.8 the assumed network by using shared weights and auto-associative topologies

There are two steps to apply shared weights and auto-associative topologies. Firstly, we passed the 64 input attributes data to input and mirror output layers to guarantee an auto-associative network. The mirror input layer has the same neurons number as the input layer, the mirror hidden layer has the same neuron number as the hidden layer. After that, we defined our regression process to train the 5-layer neural network and used t function to let the actual layers and mirror layers have the same weights. Because the program couldn't run in a small epoch (500 epochs), so we used 2500 as the epoch number to train. Chart 2.9 shows the loss and Chart 2.10 shows the accuracy and testing accuracy under the shared weights and auto-associative topologies.

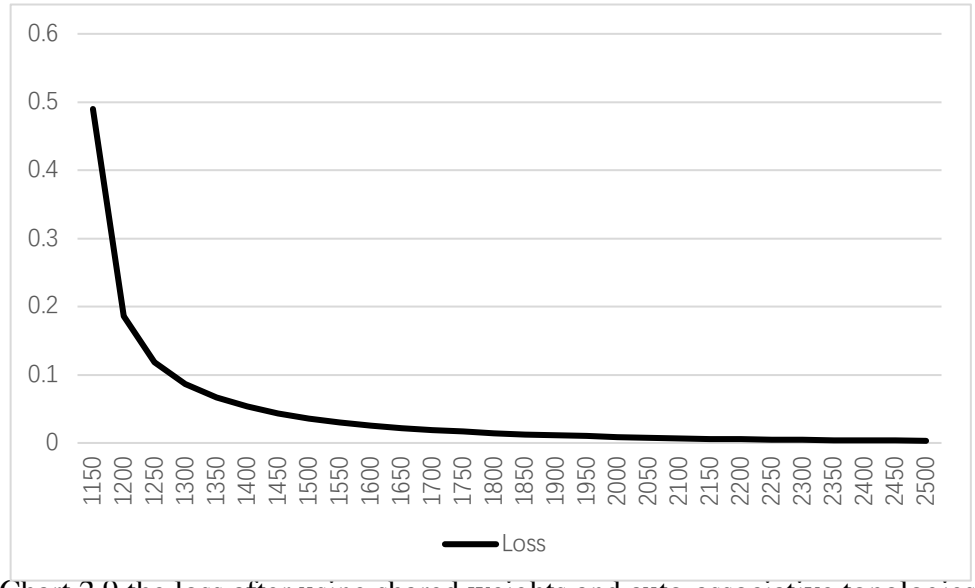

Chart 2.9 the loss after using shared weights and auto-associative topologies

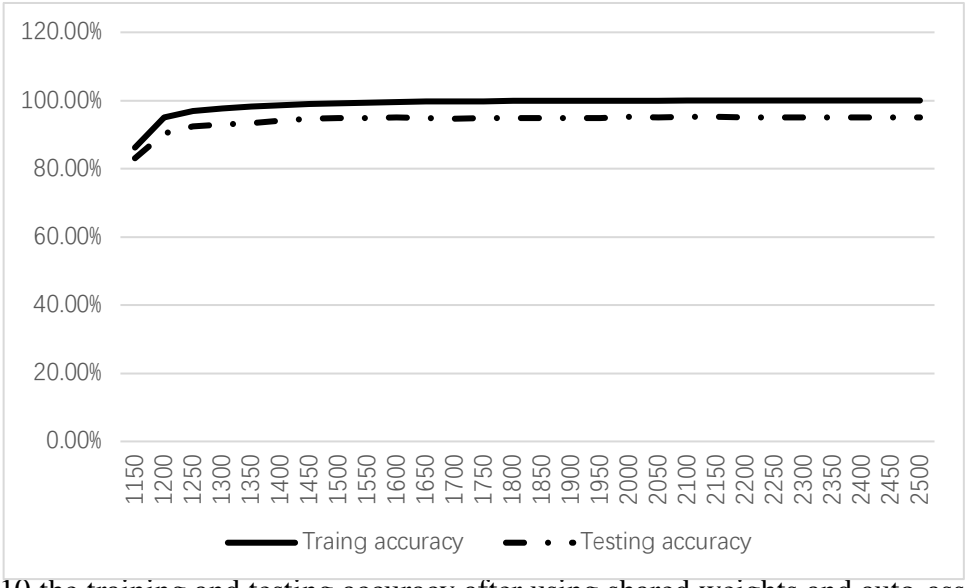

Chart 2.10 the training and testing accuracy after using shared weights and auto-associative topologies

As the charts show, the loss reached a very low level after using shared weights and auto-associative topologies. However, there is a gap between the training and testing accuracy after 2500 epochs, the training accuracy was higher than the testing accuracy. This means that there is an overfitting problem.

2.6 Building Convolutional Neural Network (CNN)

In 2.2, we used a simple three-layer network, but we only got 75.18% and 87.94% for testing and training accuracy separately. In 2.5, we tried to apply shared weights and auto-associative neural network, but we got an overfitting problem. According to Zhou (2016), Convolutional Neural Network (CNN) is a widely-used technique because it could generate a better estimation result in image and automatic speech recognition. To improve the results and mitigate the overfitting problem, we used Convolutional Neural Network (CNN) in this step.

For better comparing the results with the simple three-layer network, we set epoch as 500 and learning rate as 0.01. We processed the data in the same way as we did in building a three-layer network. We set the input channel as 1, the output channel as 4, kernel size as 5, stride as 1 and padding as 2 to build a Convolutional Neural Network (CNN). After 500 epochs of training, we could see the results in Chart 2.11 and Chart 2.12.

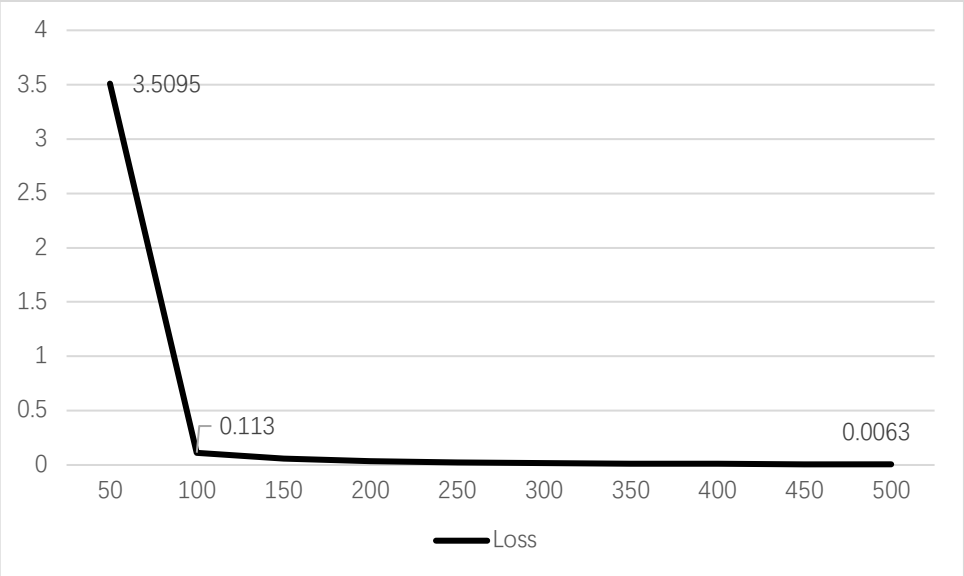

Chart 2.11 the loss after using Convolutional Neural Network (CNN)

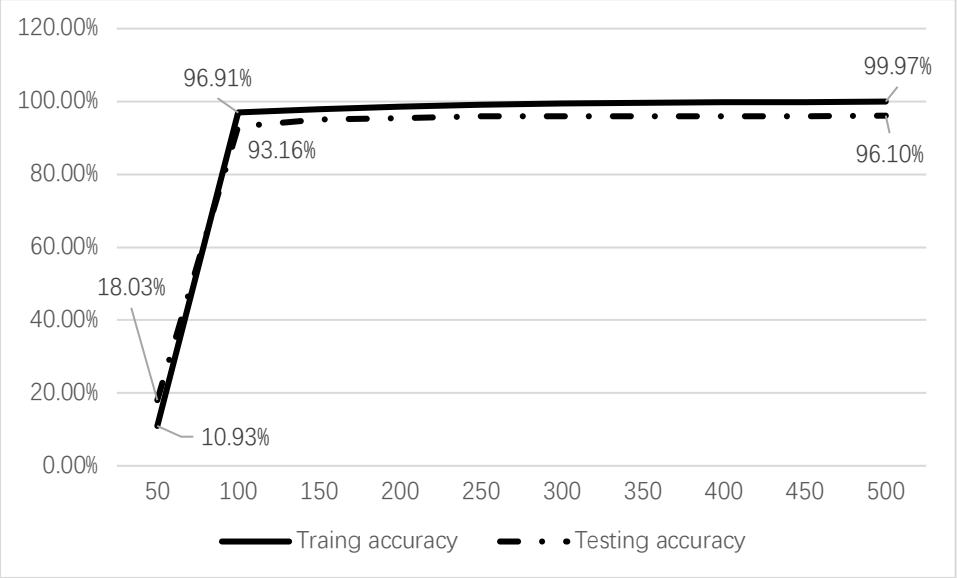

Chart 2.12 Accuracy of training and testing dataset after using Convolutional Neural Network (CNN)

As is shown in Chart 2.11, the loss started from 3.5095 and decreased quickly after 100 epochs. After 100 epochs, the slope of the loss become smooth, and the loss value reached 0.0063 finally. When we were using shared weights and auto-associative topologies, the loss started at a value around 5 and reached a relatively low value only after about 1700 epochs. As a result, with the Convolutional Neural Network (CNN) could reduce the loss better. However, the training accuracy reached 99.97% while the testing accuracy was only 96.10%, the network did better on the training set than on the testing set. It indicated that we had an overfitting problem ("Overfitting in machine learning", 2017). One of the solutions is to add more training data to improve the result, but we have no more training data for now, so we could use other methods to address the overfitting problem.

3 Results and Discussion

This report has implemented 5 different neural networks to train and test the handwritten digit dataset. During each process of implementing, every type of neural network shows its accuracy on training and testing. For better evaluating the results of this report, we found a related published paper to compare. Xu, Krzyzak, and Suen (1992) used the same dataset and applied four approaches to solve the problem of combining classifiers when recognizing handwritten digits. Finally, they reached a high testing accuracy of 98.9%, which is relatively higher than the result after using Convolutional Neural Network(CNN) in this report.

The first method we used was to implement a simple three-layer neural network, setting the epoch number as 500 and the learning rate as 0.01. The result of training and testing accuracy were 75.18% and 87.94% separately. Then we tried to increase the learning rate and change the epoch to see the change of the results. After implementing the second method in 2.3, the testing accuracy increased to 92.21% after changing epoch number to 1500, and it increased to 97.75% after changing the learning rate from 0.1 to 0.5. This result show that increasing epoch number and learning rate will help improve the result of accuracy. In 2.4, we used the third method to improve the performance. Optimizers Momentum and Adaptive Moment Estimation (Adam) were used to train the network instead of the Stochastic Gradient Descent (SGD). After 500 epochs, the testing accuracy increased to 95.03% after using the Momentum, and it increased to 99.71% after using the Adaptive Moment Estimation (Adam). However, after comparing the results of the three optimizers, we found that Adaptive Moment Estimation (Adam) generated better results than other two optimizers. By using the fourth method in 2.5, we applied the shared weights and auto-associative network topologies. Even though the loss could reach a low value after using these two topologies, there was an overfitting problem, the training accuracy was higher than the testing accuracy. To try to improve the performance and address the overfitting problem, we used a Convolutional Neural Network (CNN) in 2.6, even though the CNN could reduce the loss to a better extent, it still showed an overfitting problem.

Among the 5 methods we used, the best result of accuracy occurred when using Adaptive Moment Estimation (Adam) optimizer. It increased the accuracy rate and didn't generate an overfitting problem.

4 Conclusion and Future Work

This report has implemented 5 methods to improve the performance of the neural network. By comparing the results under the different methods, using Adaptive Moment Estimation (Adam) as the optimizer is the best way to increase the accuracy. Shared weights with auto-associative network and Convolutional Neural Network (CNN) could reduce the loss and increase the testing accuracy, but both of them could generate an overfitting problem.

We assumed one of the reasons why shared weights and Convolutional Neural Network (CNN) would generate an overfitting problem is that there is no sufficient data to train. By feeding more training data into the model, the overfitting problem may be solved. Another reason may be because the way of splitting dataset is not suitable, there are only 32% of data used in testing. If we feed more testing data to the network, the results of accuracy may be better.

For future work, it is important for us to consider how much data we need to train a neural network, and how to split training and testing data well before training network. Feeding more data into the network and adding more testing data could be beneficial for addressing the overfitting problem. As a result, the future work should focus on the data searching and data processing.

References

Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, 34(4), 369-374.

Alpaydin, E., & Kaynak, C. (1998, July 1). Optical recognition of handwritten digits data set. Retrieved May 30, 2018, from UCI machine learning repository: https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits

Brownlee, J. (2016, November). What is a confusion matrix in machine learning? *Machine Learning Mastery*. Retrieved from https://machinelearningmastery.com/confusion-matrix-machine-learning/

Gedeon, T. D., Catalan, J. A., & Jin, J. (1997). Image compression using shared weights and. bidirectional networks. *2nd International ICSC Symposium on Soft Computing (SOCO'97)*, (pp. 374-381).

Kingma, D. P., & Ba, J. (2015). Adam: a method for stochastic optimization. *ICLR*.

MacLeod, C.(2010). An introduction to practical neural networks and genetic algorithms. *The Robert Gordon University*. Retrieved from https://zh.scribd.com/document/359442318/Christopher-MacLeod-An-Introduction-to-Practical-Neural-Networks-and-Genetic-Algorithms-For-Engineers-and-Scientists-pdf

Overfitting in machine learning: what it is and how to prevent it (2017, September 7). *Elite DataScience*.

Ruder, S. (2016, January 19). An overview of gradient descent optimization algorithms [Blog post]. Retrieved from http://ruder.io/optimizing-gradient-descent/index.html#fn:15

Xu, L., Krzyzak, A., & Suen, C. Y. (1992). Methods of combining multiple classifiers and their. applications to handwriting recognition. *IEEE transactions on systems, man, and cybernetics,* 22(3), 418-435.

Zhou, M. (2016, November 03). What is Convolutional Neural Network(CNN) [Blog post]. Retrieved from https://morvanzhou.github.io/tutorials/machine-learning/tensorflow/5-03-A-CNN/

Zulkifli, H.(2018). Understanding learning rates and how it improves performance in deep learning. Retrieved from https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.