Final Coursework Report

Yuliang Xiao
Student ID: 1929288

Abstract

In this final coursework, we aimed to explore handwritten digit recognition using two different approaches: a three-layer Multi-Layer Perceptron (MLP) and a Support Vector Machine (SVM), and we conducted our experiments on the MNIST dataset. The objective was to explore the impact of specific hyperparameter settings on the performance of these models. For the MLP, we investigated the effects of varying the learning rate, the number of hidden nodes on the model's performance, the method of weight initialization, and activation function. In parallel, we explored the performance of SVM, and to enhance the SVM's efficiency, we utilized Principal Component Analysis (PCA) to reduce the dimensionality of the input features. Throughout the experimentation process, we meticulously evaluated the performance of both models based on their accuracy in classifying handwritten digits. We compared the results obtained by varying the learning rate and the number of hidden nodes in the MLP, as well as the value of C in the SVM. By systematically analyzing these hyperparameters, we aimed to identify the optimal configurations for each model. After experiment, the highest accuracy for MLP is 95.7% while the highest accuracy is 98.5%. The code is made available at https://github.com/Regen2001/Numpy-MLP-handwritten-digital-recognition for MLP and https://github.com/Regen2001/SVM-handwritten-digital-recognition for SVM.

1 Introduction

The ability to accurately recognize and classify handwritten digits is a fundamental task in the field of machine learning. In this experiment, we explore the application of two popular algorithms, Multi-Layer Perceptron (MLP) and Support Vector Machines (SVM), for handwritten digit recognition using the MNIST dataset.

The first algorithm we employ is the Multi-Layer Perceptron (MLP), which is a type of artificial neural network. MLPs consist of multiple layers of interconnected nodes, known as neurons, and are particularly effective in handling complex classification tasks Mahesh (2020). In our experiments, we focus on two key factors that can significantly impact the performance of an MLP model: the learning rate and the number of hidden nodes. By varying these parameters, we can observe how they influence the accuracy and convergence of the model. In addition, we tested the different methods of weight initialization and activation function and compared their effects in the artificial neural network.

Furthermore, we also incorporate Support Vector Machines (SVM) into our experiment. SVMs are powerful and versatile machine learning models widely used for classification tasks Chauhan et al. (2019). To enhance the performance of the SVM model, we utilize Principal Component Analysis (PCA) to reduce the dimensionality of the feature space Tipping & Bishop (1999). By reducing the number of features, we aim to improve the computational efficiency and potentially enhance the accuracy of the SVM model.

In this experiment, we systematically evaluate the performance of MLP and SVM models on the MNIST dataset. We compare their accuracy, convergence rates, and computational efficiency under different settings. By examining the impact of varying learning rates and hidden nodes in MLP, as well as the effect of different values of the regularization parameter

(C) in SVM, we aim to identify the optimal configurations for achieving high accuracy and efficiency in handwritten digit recognition.

2 Methodology

2.1 Multi-layer Perception

Multi-Layer Perceptron (MLP) is a widely used neural network model that excels in capturing complex patterns. With its interconnected layers of neurons, MLPs have become essential in tasks like image recognition. Training an MLP involves forward propagation, where inputs pass through layers to generate predictions, and backward propagation, which computes gradients to update the network's parameters Mahesh (2020). In this experiment, we explore MLP's effectiveness in handwritten digit recognition using the MNIST dataset. Figure 1 is a schematic diagram for MLP.

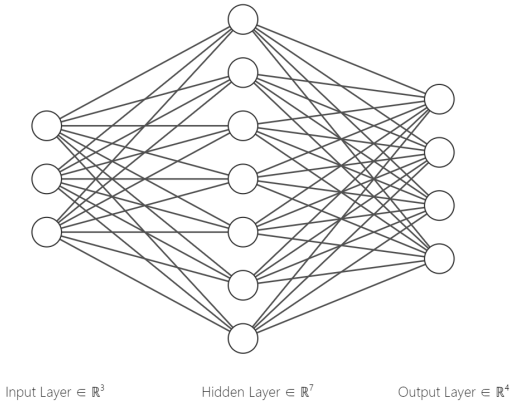


Figure 1: The schematic diagram for MLP

2.1.1 Initialization Methods and Activation Functions

Initialization methods in machine learning to refer to techniques used to assign initial values to the parameters of a model before training. In this experiment, we tested Xavier and He initialization methods, while as the initialization method is different, we have chosen the different activation functions.

Xavier initialization, a widely used technique in neural networks, sets the initial parameter values to maintain consistent variance across layers. It considers the number of input connections to appropriately scale the initialization, for example, $w_{ji} \in \left(-\sqrt{\frac{6}{D+L+1}}, \sqrt{\frac{6}{D+L+1}}\right)$ Glorot & Bengio (2010). By balancing the parameter values, Xavier initialization helps prevent vanishing or exploding gradients and facilitates stable and efficient training. Thus, we have chosen $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ as its activation function, while its derivatives is shown as $f'(x) = 1 - f^2(x)$.

He initialization is an initialization method for neural network weights. It addresses the vanishing/exploding gradient problem by setting initial weights from a Gaussian distribution with zero mean and variance scaled by $2/n$, where n is the number of input connections. This technique facilitates efficient optimization and stability in deep learning models He et al. (2015). For this type of method, we have chosen ReLU function as its activation function, which is $f(x) = max(0, 1)$, while its derivatives is $f'(x) = 0,\ x < 0$ and $f'(x) = 1,\ x \geq 0$.

2.1.2 Forward Propagation

Forward propagation in MLP involves passing input data through the network, computing weighted sums and applying activation functions to generate output predictions. It captures

the flow of information from input to output, enabling the network to make predictions based on the learned parameters Mahesh (2020).

Set the input layer l is $[x_1, x_2, x_3]$, while L_i means all the neurons of the layer, and the output is y_l, where the output of the jth node is $y_l^{(j)}$, the input of the node is $u_l^{(j)}$, the weight matrix connecting layer l and layer $(l1)$ is W_l, and the weight from the ith node of the previous layer $(l1)$ to the jth node of layer l is $w^{(ji)_l}$. Therefore, the output y_i can be written as Mahesh (2020)

$$\begin{cases} y_2^{(1)} = f(u_2^{(1)}) = f(\sum_{t=1}^{n} w_2^{1i} x_i + b_2^{(1)}) \\ y_2^{(2)} = f(u_2^{(2)}) = f(\sum_{t=1}^{n} w_2^{2i} x_i + b_2^{(2)}) \end{cases} \tag{1}$$

and then, convert it into matrix form

$$y_2 = \begin{bmatrix} y_2^{(1)} \\ y_2^{(2)} \end{bmatrix} = f\left(\begin{bmatrix} w_2^{11} & w_2^{12} & w_2^{13} \\ w_2^{21} & w_2^{22} & w_2^{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_2^{(1)} \\ b_2^{(2)} \end{bmatrix} \right) = f(W_2 X + b_2) \tag{2}$$

Furthermore, extent the equation of forward propagation from the last layer to any layer in the neural network, which is

$$\begin{gathered} y_l^{(j)} = f(u_l^{(j)}) = f(\sum_{i \in L_{(l-1)}}^{n} w_l^{ji} x_i + b_l^{(j)}) \\ y_l = f(u_l) = f(W_l y_{(l-1)} + b_l) \end{gathered} \tag{3}$$

where $f(\cdot)$ represent to activation function and $b_l^{(j)}$ is the bias of the jth node in layer l.

2.1.3 Backward Propagation

Back propagation, a fundamental algorithm in neural network training, plays a crucial role in optimizing model parameters by propagating error gradients backward through network layers. By iteratively adjusting weights and biases based on these gradients, backpropagation enables the network to learn from labeled data and improve its performance in solving complex problems LeCun et al. (2002).

In this experiment, the Mean Squared Error (MSE) loss is applied, which is calculated as

$$E_{MSE} = \frac{1}{2} \sum_{j \in L_k} (t^{(j)} - y_k^{(j)})^2 \tag{4}$$

In order to minimize the loss function, it is derived by gradient descent. There are $\frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} = f'(u_l^{(j)})$, $\frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = y_{(l-1)}^{(i)}$, and $\frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = 1$, so we have

$$\begin{cases} \frac{\partial E_l^{(j)}}{\partial w_l^{(ji)}} = \frac{\partial E_l^{(j)}}{\partial y_l^{(j)}} \frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \frac{\partial E_l^{(j)}}{\partial y_l^{(j)}} f'(u_l^{(j)}) y_{(l-1)}^{(i)} \\ \frac{\partial E_l^{(j)}}{\partial b_l^{(j)}} = \frac{\partial E_l^{(j)}}{\partial y_l^{(j)}} \frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \frac{\partial E_l^{(j)}}{\partial y_l^{(j)}} f'(u_l^{(j)}) \end{cases} \tag{5}$$

Also, each node of the next layer is related to all the nodes from the former layer, and the loss function can be considered as the function of the input of each node from next layer, and if we set $\delta_l^{(j)} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)})$, it is means

$$\frac{\partial E_l^{(j)}}{\partial w_l^{(ji)}} = \sum_{k \in L_{(l+1)}} \frac{\partial E}{\partial y_{(l+1)}^{(k)}} \frac{\partial y_{(l+1)}^{(k)}}{\partial u_{(l+1)}^{(k)}} w_{(l+1)}^{(kj)} = \sum_{k \in L_{(l+1)}} \delta_{(l+1)}^{(k)} w_{(l+1)}^{(kj)} \tag{6}$$

However, in view of the output layer, there is no "next layer" to be considered, and the output layer does not fit the equation above. The output of the final layer is related to the

error, and the derivation of the loss function can be considered directly. Therefore, multiply $f'(u_l^{(j)})$ to both sides and the equation can be written as

$$\delta_l^{(j)} = \begin{cases} f'(u_l^{(j)}) \sum_{k \in L_{(l+1)}} \delta_{(l+1)}^{(k)} w_{(l+1)}^{(kj)} & \text{if } l \text{ is hidden layer} \\ f'(u_l^{(j)}) \frac{\partial E}{\partial y_l^{(j)}} & \text{if } l \text{ is output layer} \end{cases} \tag{7}$$

Therefore, the gradient of loss function to each parameter is

$$\begin{cases} \frac{\partial E}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \delta_l^{(j)} y_{(l-1)}^{(i)} \\ \frac{\partial E}{\partial b_l^{(j)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \delta_l^{(j)} \end{cases} \tag{8}$$

Thus, the update equation of weight for each layer are LeCun et al. (2002)

$$\begin{cases} W_l = W_l - \eta \frac{\partial E}{\partial W_l} = W_l - \eta \delta_l y_{(l-1)}^T \\ b_l = b_l - \eta \frac{\partial E}{\partial b} = b_l - \eta \delta_l \end{cases} \tag{9}$$

2.2 Support Vector Machines

In support Vector Machines (SVM), the goal is to find an optimal hyperplane that separates the training data into different classes while maximizing the margin between the hyperplane and the nearest data points, known as support vectors Chang & Lin (2011). This margin acts as a measure of the model's robustness and its ability to generalize well to unseen data.

For linearly separable data, the SVM optimization problem can be formulated as follows Chang & Lin (2011)

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ \text{subject to} \quad &y_i(w^T \mathbf{x}_i + b) \geq 1, \quad \forall i = 1, 2, \ldots, N \end{aligned} \tag{10}$$

Then, the Lagrange multiplier method is used to optimize this equation, while the Lagrange function is Chauhan et al. (2019)

$$\begin{aligned} L(w, \lambda, a) &= f(w) + \sum_{i=1}^{n} \lambda_i h_i(w) \\ &= f(w) + \sum_{i=1}^{n} \lambda_i [g_i(w) + a_i^2] \quad \lambda_i \geq 0 \end{aligned} \tag{11}$$

For $\sum_{i=1}^{n} \lambda_i a_i^2 \geq 0$, the question can be rewritten as $\min L(w, \lambda)$, which is

$$L(w, \lambda) = f(w) + \sum_{i=1}^{n} \lambda_i g_i(w) \tag{12}$$

In practice, there are very few completely linearly separable samples, so the soft interval method is to allow individual samples to appear in the interval band. Then the original optimization goal of support vector machine becomes as Chauhan et al. (2019)

$$\begin{aligned} &\min_{w} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} \xi_i \\ &\text{s.t. } g_i(w,b) = 1 - y_i(w^T x_i + b) - \xi_i \leq 0, \quad \xi_i \geq 0 \end{aligned} \tag{13}$$

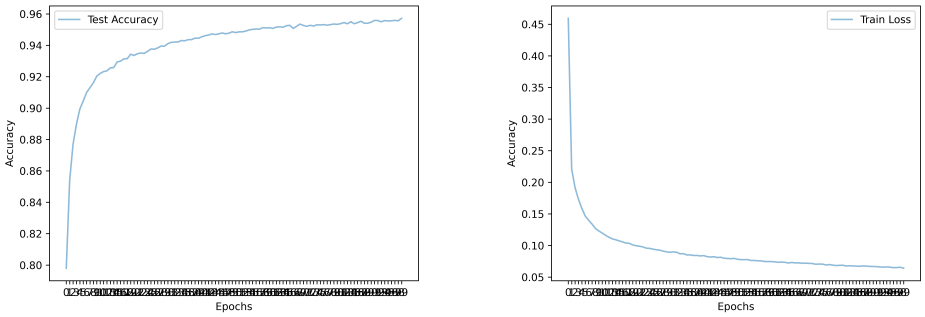
Then the corresponding Lagrange function is

$$\min_{w,b,\xi} \max_{\lambda,\mu} L(w, b, \xi, \lambda, \mu) = \frac{1}{2}\|w\|^2 + C \sum_{\substack{i=1 \\ \text{s.t. } \lambda_i \geq 0, \mu_i \geq 0}}^{m} \xi_i + \sum_{i=1}^{n} \lambda_i \left[1 - \xi_i - y_i \left(w^T x_i + b\right)\right] - \sum_{i=1}^{n} \mu_i \xi_i \tag{14}$$

3 Experiment Result and Analysis

3.1 Multi-layer Perception

In all experiment, we set the epoch is 100 and the value of gamma in stochastic gradient descent (SGD) is 0.7. Then, when we set the value of learning rate is 0.05, the number of hidden node is 64, initialization method is He initialization, and activation function is ReLU function, the final accuracy is 95.7%. Then, Figure 2(a) is the accuracy in each epoch and Figure 2(b) is the loss in each epoch. However, from the figure, it is found that if we continue to train the model, the accuracy can still increase.



(a) Accuracy (b) Loss

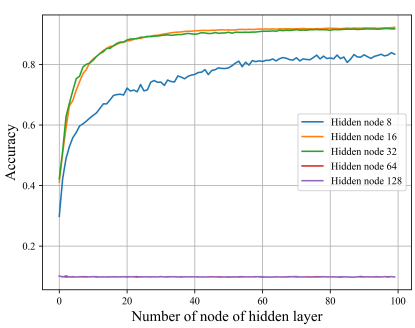
Figure 2: The training process of MLP for each epoch

However, this result was collected after we tested different value of learning rate and the number of hidden node, while we also evaluated the different method of initialization method and activation function. For example, when we replaced the initialization method with Xavier initialization, the activation function with Tanh, and the number of hidden node was 32, the final accuracy was 90.2%, which is lower than the above result. What is more, when we try to increase the number of hidden node, this model happened gradient explosion. For example, as we set number of hidden node as 64, its accuracy was only about 10% and its loss was infinite.

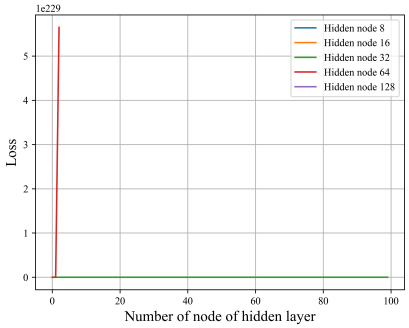
Therefore, we try to evaluated the effect of the number of hidden node which leads to gradient explosion when the initialization method is Xavier initialization and the activation function is Tanh. In the end, we found gradient explosion will happen if the number of hidden node is more than 60. Figure 3.1 is the result for this experiment as we changed the number of hidden node.

The size of hidden layer is one of the most important factor for our three-layer MLP, so we assessed the result for the different size. Figure 3.1 shows the result when we changed the number of hidden node. It is clear that if the size of hidden layer is too small, the accuracy will be low. However, when the number is larger than 32, the accuracy will higher than 90% while the number is higher than 64, the accuracy will remain at 96%. Since with the same performance, the model with fewer parameters has a wider range of application scenarios, we believe that the number is 64 can be the best model. Figure 3.1 only shows part of experiment result, Figure 9 in Appendix is shown the more detailed result. However, contrary to expectations, our model did not appear to overfitting, which was considered that the dataset is large enught and out model is too shallow and simple.

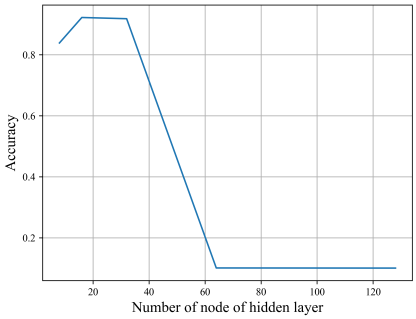
The learning rate plays an important role in the training phase since if this value is too big, the model cannot be convergent while if this value is too small, the epoch used to training will too big. Figure ?? shows the result if only learning rate be changed when we trained this MLP. It is seemed that if the learning rate is large, for example larger than 0.15, this model cannot be convergent, but if the learning rate is too small such as 0.01, the training



(a) Accuracy



(b) Loss



(c) Maximum accuracy

Figure 3: The training process for MLP while the number of hidden node is changed, and the initialization method is Xavier and activation function is tanh

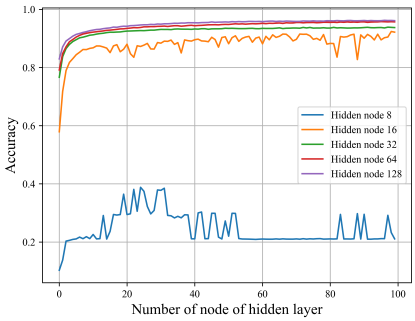
time will be extended, which is shown in Figure 12 in Appendix. Therefore, from this sub-experiment we found that, the suitablt learning rate is between 0.05 to 0.1, and we can get a high accuracy in 100 epoch.

In addition, the batch size will also influence the result of training while in SGD, the small batch size leads to a higher accuracy but longer training time, and a big batch size could reduce the training time but leads to a lower accuracy Keskar et al. (2016). Figure 6(a) is the accuracy change when batch size is 1024 in each epoch and the final accuracy is 94.7%, and then, Figure 6(b) is the result if we set the batch size as 5120 while its final accuracy is 91.8%. Compared with Figure 2(b), the curve for accuracy is more smooth while the batch size increasing.

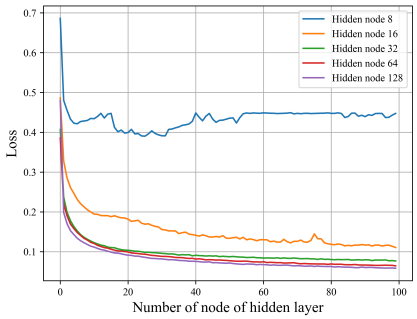
3.2 Support Vector Machines

As we choose SVM to realize classification task, select the suitable classify strategy and kernel is necessary since the different kernel will affect the accuracy and training time seriously, which is shown in Table 1 while we set the value of C is 2.0 and the slack variables is $\xi = 1/N_{\text{feature}}$, and tested by i9-10900X CPU Chang & Lin (2011). It is found that the influence from classify strategy is small but, the effect of kernel cannot be ignored. In these experiments, the rbf kernel shows the best performance, which includes the highest accuracy and minimum time. The sigmoid kernel shows the worst performance that the lowest accuracy and long training time. Furthermore, we applied PCA to decrease the features before fitting by SVM, and it is shown that the training time reduced significantly with the same accuracy Deepu et al. (2004).

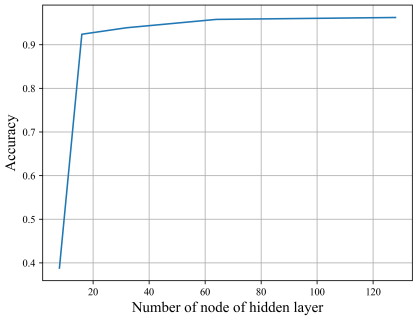
Most time, the slack variable in SVM will be calculated as $\xi = \frac{1}{N}$ where N is the number of feature, so in this experiment, we did not test the influence of slack variable but evaluated



(a) Accuracy



(b) Loss



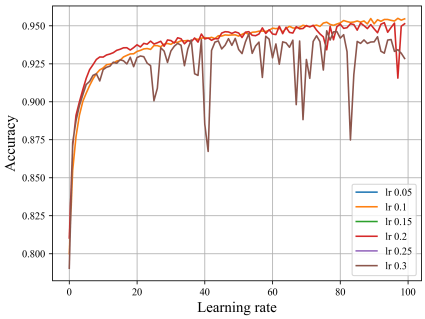
(c) Maximum accuracy

Figure 4: The training process for MLP while the number of hidden node is changed, and the initialization method is He and activation function is ReLU

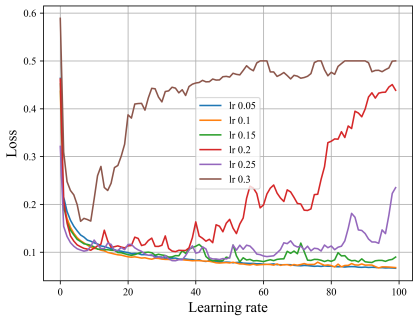
Table 1: The accuracy for SVM while the parameters are changed

The value of K in PCA	Classify Strategy	Kernel	Accuracy	Training Time / s
39	ovr	rbf	0.985	25.745
		linear	0.929	38.887
		poly	0.983	27.743
		sigmoid	0.798	36.598
	ovo	rbf	0.985	24.459
78	ovr	rbf	0.985	36.257
		linear	0.937	54.854
		poly	0.984	50.747
		sigmoid	0.828	46.587
	ovo	rbf	0.985	37.115
784	ovr	rbf	0.985	288.900
		linear	0.939	397.708
		poly	0.984	487.625
		sigmoid	0.848	257.107
	ovo	rbf	0.985	288.007

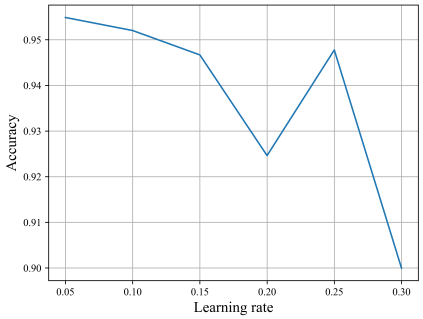
the effect for C. Figure 7 shows the accuracy when the value of C is changing, while to avoid the influence of PCA algorithm, we used the original data fitting. It is seemed that after C is morn than 2.0, the accuracy is higher than 0.985 while even though the value of C increases, the accuracy always approaches to 0.985. Additionally, Figure 16 in Appendix shows the more detailed result.



(a) Accuracy

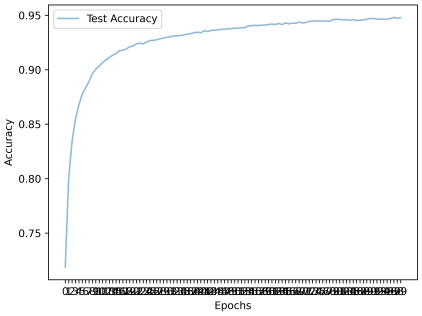


(b) Loss

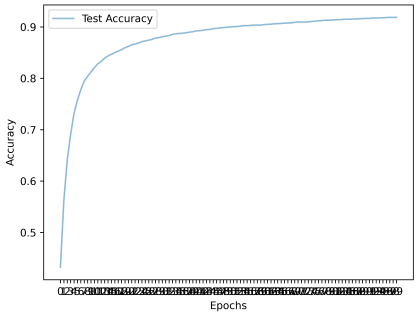


(c) Maximum accuracy

Figure 5: The training process for MLP while the learning rate is changed, and the initialization method is He and activation function is ReLU



(a) The accuracy change when batch size is 1024



(b) The accuracy change when batch size is 5120

Figure 6: The training process for MLP while the batch size is changed, and the initialization method is He and activation function is ReLU

Furthermore, since we applied PCA algorithm in this experiment, the value of K in PCA also influence the result of training and the training time. Figure 3.2 is result for this test and Figure ?? shows the training time for the different value of K, while owing to avoid the effect of C, we set $C = 1.0$. In the experiment, when the number of feature be saved was 1 percent for original data, which means $K = 8$, the accuracy is 0.913, higher than 90%. In addition, most time, the value of K is set as $k = 0.01N_{\text{feature}}$ Minka (2000). Then, if we saved 5% feature for original data, which means $K = 39$, the accuracy is 0.985, same with

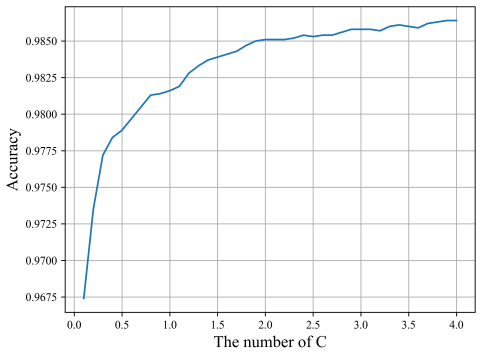
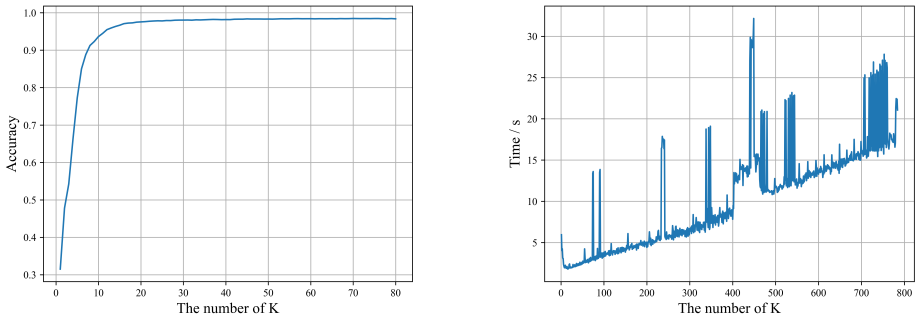


Figure 7: The accuracy change while the value of C is increasing

the result that just fitting the original data. However, from the curve in Figure 3.2, after the value of K is higher than 28, the accuracy will be 0.980. Since Figure ?? shows that the training time will decrease as the value of K reduce, we believe that $K = 39$ can be the best value. In addition, Figure 15 in Appendix shows the detailed result that we tested the effect of K from 1 to 784.



(a) Accuracy (b) Training time

Figure 8: The training process while the value of K is increasing

4 Conclusion

Compared with two methods used to realize handwritten digital recognition, it is seemed that SVM can be a better choice not only the higher accuracy, but also the less computing required. When we use R7-5800H tp CPU test, the training time for SVM was only $37s$ but this time for MLP was $903s$, while the accuracy for SVM is 98.5%, which is higher than the 95.7% accuracy for MLP. Furthermore, another problem for MLP is it need more computing resource to predict, which limits its application. For example, when we set the number of hidden node is 64, the number of parameter is 50816, but if we hope to achieve the same accuracy, so we set the value of K in PCA as 13, there are only 140 parameters. Even if we set K as 39, which leads to the accuracy is 98.5%, there are only 400 parameters. It is meaning SVM can be deployed in more device with low performance such as Raspberry Pi or some singlechips, and SVM can be deployed easier in the embedded development. Therefore, though this experiment, we found if we hope to realize some simple classification task, the traditional mathematical methods are more suitable than these machine learning methods.

References

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3):1–27, 2011.

Vinod Kumar Chauhan, Kalpana Dahiya, and Anuj Sharma. Problem formulations and solvers in linear svm: a review. Artificial Intelligence Review, 52(2):803–855, 2019.

V Deepu, Sriganesh Madhvanath, and AG Ramakrishnan. Principal component analysis for online handwritten character recognition. In Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., volume 2, pp. 327–330. IEEE, 2004.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pp. 1026–1034, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836, 2016.

Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In Neural networks: Tricks of the trade, pp. 9–50. Springer, 2002.

Batta Mahesh. Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9:381–386, 2020.

Thomas Minka. Automatic choice of dimensionality for pca. Advances in neural information processing systems, 13, 2000.

Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analyzers. Neural computation, 11(2):443–482, 1999.

Some Figure for the Detail of Experiment

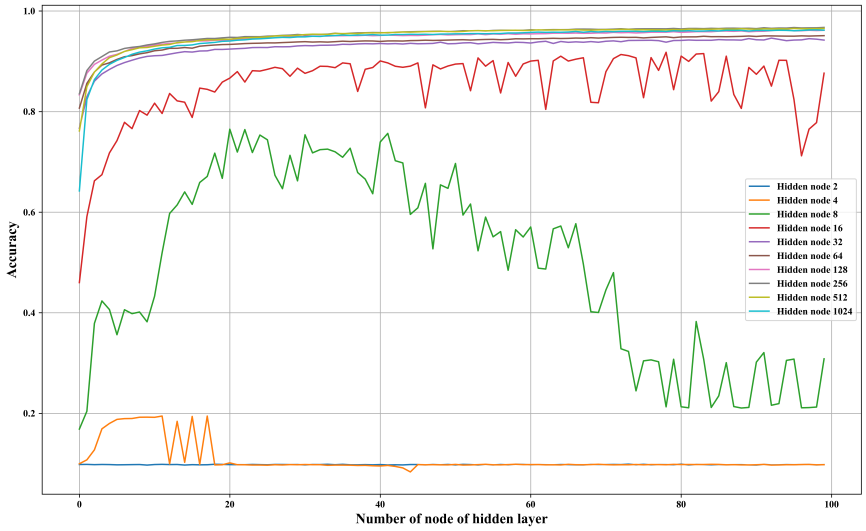


Figure 9: The accuracy for MLP while the number of hidden node is changed, and the initialization method is He and activation function is ReLU

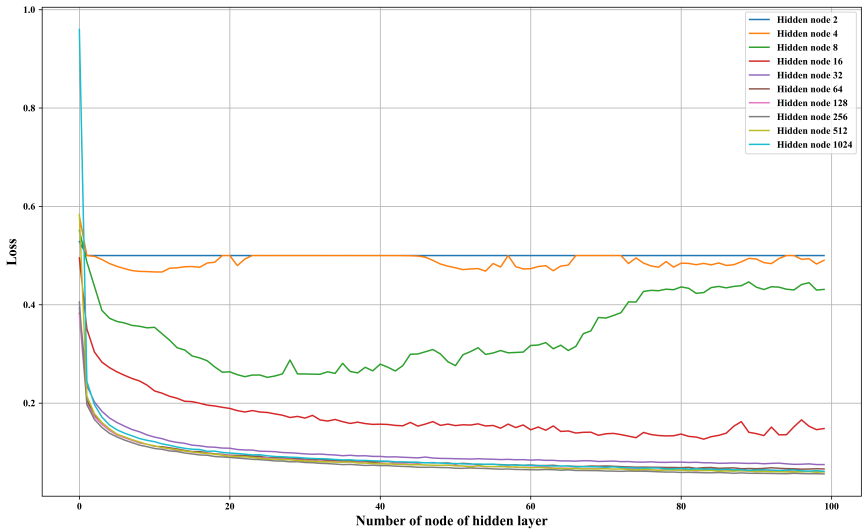


Figure 10: The loss for MLP while the number of hidden node is changed, and the initialization method is He and activation function is ReLU

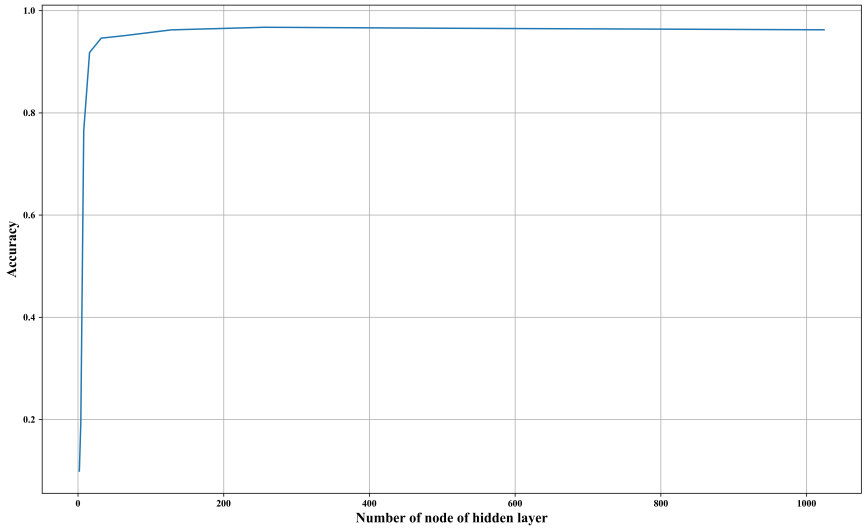


Figure 11: The maximum accuracy for MLP while the learning rate is changed, and the initialization method is He and activation function is ReLU

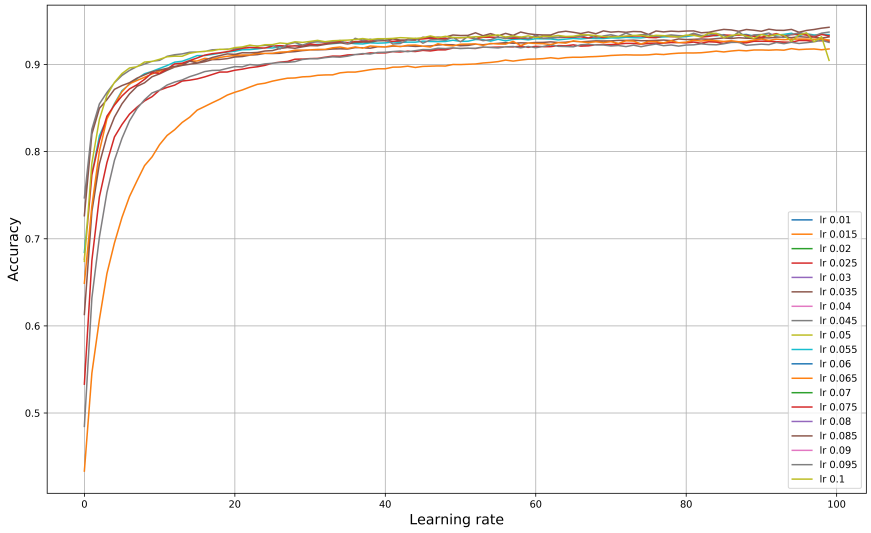


Figure 12: The accuracy for MLP while the learning rate is changed, and the initialization method is He and activation function is ReLU

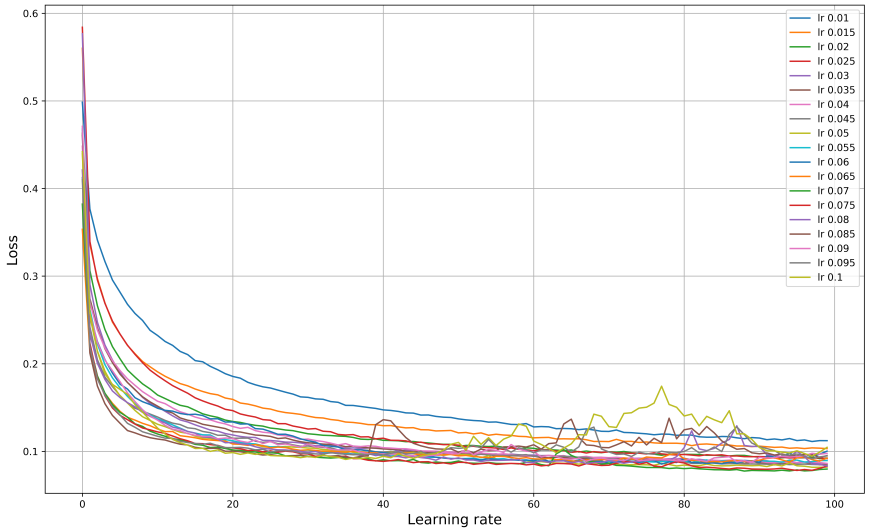


Figure 13: The loss for MLP while the learning rate is changed, and the initialization method is He and activation function is ReLU

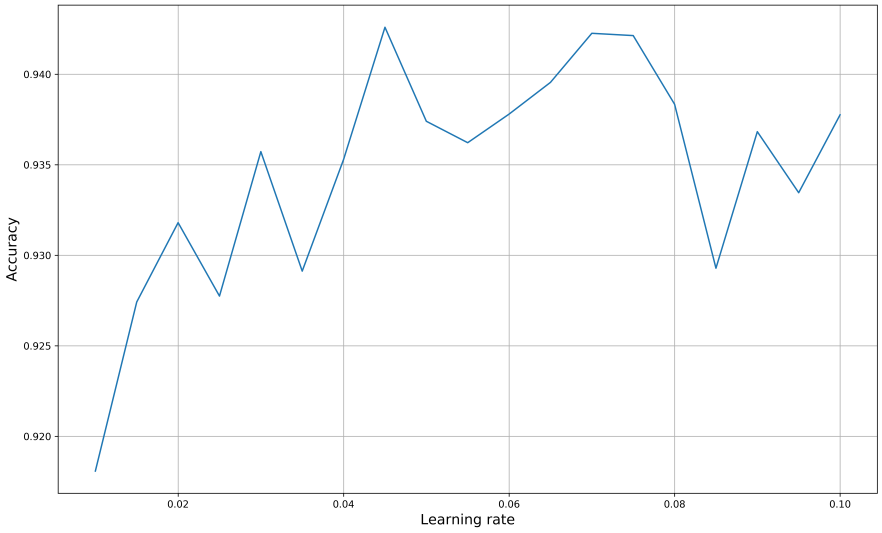


Figure 14: The maximum accuracy for MLP while the learning rate is changed, and the initialization method is He and activation function is ReLU

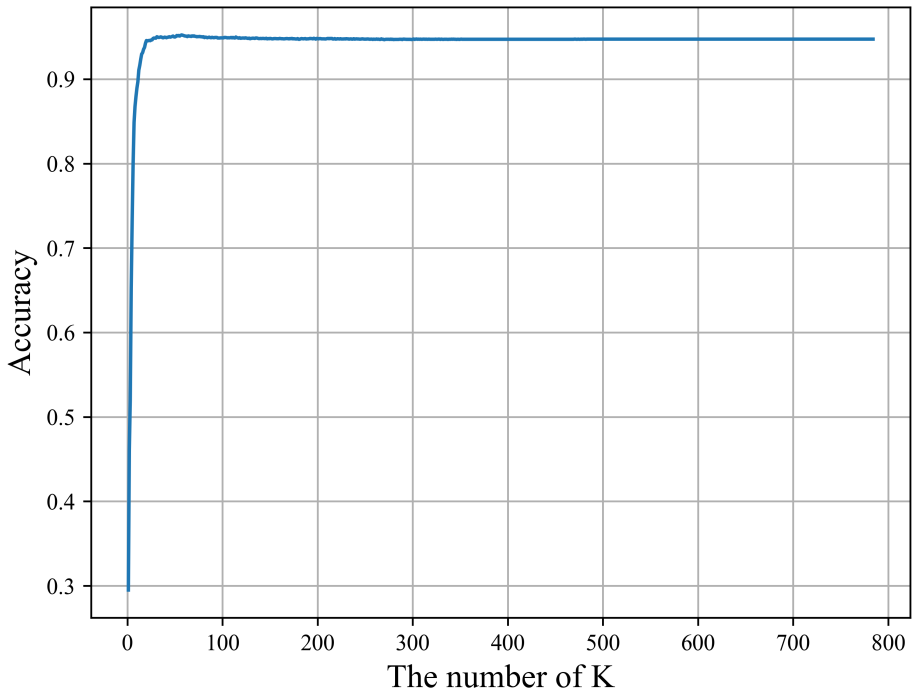


Figure 15: The accuracy change while the value of K is increasing

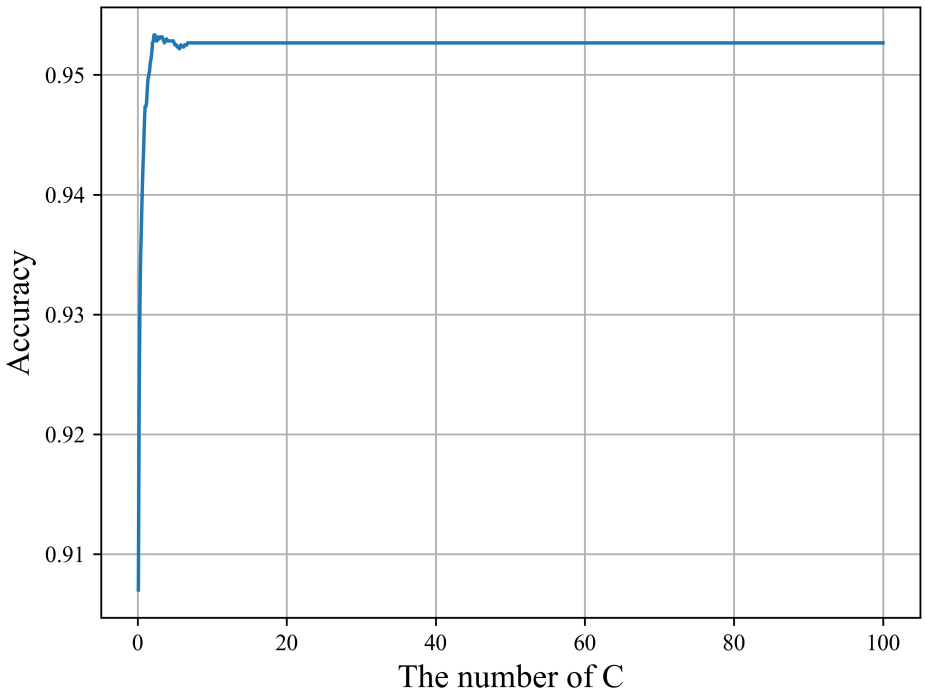


Figure 16: The accuracy change while the value of C is increasing