AE3MLE CW Report

The University of Nottigham Ningbo China,
School of Computer Science

AE3MLE

Yichen PAN (6514897)

December 15, 2016

1 Introduction

The report presents the results of an investigation to compare the performances of the Multilayer Perceptron (MLP) and the K Nearest Neighbor (KNN) classifier for MNIST handwritten digit recognition problem. The comparison is about the recognition performance and the computational requirements of each classifier. The dataset used here is derived from the original MNIST database [6], and the experiment environment is set in Matlab (2015b) with Neural Networks Toolbox 4.0.1.

2 Task1: Multilayer Perceptron

Considering the unreasonable split of original dataset, the dataset is further changed into 7000 samples for training and 3000 samples for testing. Additionally, in order to further investigate the effects of different parameters, 8 more experiments are conducted. The summary and plots of total 18 experiments are in the Appendix. A.

2.1 Learning Rate, epochs and training functions

The learning rate is a value which controls the pace of training. It can make move to the local minimum more quickly. It can also cause to jump over it. The learning rate is multiplied times the negative of the gradient to determine the changes to the weights and biases. The larger the learning rate, the bigger the step. If the learning rate is made too large, the algorithm becomes unstable. If the learning rate is set too small, the algorithm takes a long time to converge.

The larger learning rates appear to help the model locate regions of general, large-scale optima, while smaller rates help the model focus on one particular local optimum. As shown from the experiment 3, 4 and 5. A, a reasonable increase in the learning rate tends to improve the peroformance. Moreover, there is a fluctuation near 3200 epoches with 0.03 learning rate. This is because the model jumps out of the local optimum into another optimum due to the relatively large step.

The epoch is a measure of the number of times all of the training vectors are used once to update the weights. For batch training all of the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated. As shown from the experiment 1, 2, and 3, the choice of epochs directly affects the performance of the model and training time. As the epoch increases from 1000, 10000, to 50000, the performance improves significantly from 35.9%, 74.10% to 85.3%. This is because the model has not reached a stable state and on the way to the local/global optima (i.e. the training performance is always improving while training). Meanwhile, the training time also increases accordingly.

With regard to the training function, the most common one is called gradient descent backpropagation (”traingd” in Matlab), which updates weight and bias values in the direction of the negative gradient of the performance function. Another improved version of common gradient descent is gradient descent with momentum, implemented by ”traingdm” in Matlab, which allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a lowpass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network can get stuck in a shallow local minimum. With momentum a network can slide through such a minimum.

One observation is that it is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface. In this case, a third training function called gradient descent with adaptive learning rate is used here (implemented by ”traingda” in Matlab). Different from the two above, the learning rate is made responsive to the complexity of the local error surface. As shown form the experiment 10, 16, and 18 where models have the same parameter setiings apart from the training function, the model that uses ”traingda” has the best performance.

2.2 Topology

The choice of the number of hidden layers is related to the under/over fitting problem. Higher number of hidden layers will result in overfitting (network can model any variation in data) and lower number will often result in underfitting (most of variance in data cannot be modelled by the network). However, according to the Universal Approximation theorem [4], a single hidden layer is sufficient to compute a uniform epsilon approximation to a given training set, which means a neural network with few hidden layers (i.e. one or two) is sufficient for most tasks. As shown from the experiment 10, 13, and 14, when the number of hidden layer increases from 1 to 3, the performance is not improved significantly.

This is also the case for the number of nodes in the hidden layer. If there are too few hidden units, it is llikely to get high training error and high generalization error due to underfitting and high statistical bias. Meanwhile, if there are too many hidden units, there is much chance that it will bring low training error but still high generalization error due to overfitting and high variance. As shown from the experiment 3, 8, 9, 10, and 15, the model with only 5 hidden units has the worst performance (58.9% recognition rate), but when the number of hidden units increases from 10 to 25, the performance in not improved significantly (only increased from 85.3% to 90.1%).

2.3 Training Guideline

Some books and articles offer ”rules of thumb” for choosing an architecture; for example:

1. The size of this hidden layer should be somewhere between the input layer size and the output layer size [2].
2. The general rule of the size of hidden layer is (Number of inputs + outputs) * (2/3).
3. It is not required to have more than twice the number of hidden units as inputs in an MLP with one hidden layer [9].
4. One rule of thumb is that it should never be more than twice as large as the input layer [1].
5. It is required to specify as many hidden nodes as dimensions (principal components) needed to capture 70-90% of the variance of the input data set [3].

These rules of thumb could provide empirical guidelines for choosing the number of hidden layers at the beginning. However, to some extent, they are nonsense because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function. Some other rules of thumb relate the total number of trainable weights in the network to the number of training cases. A typical recommendation is that the number of weights should be no more than 1/30 of the number of training cases. Such rules are only concerned with overfitting and are at best crude approximations. Also, these rules do not apply when regularization is used.

2.4 Overfitting and Generalization

As mentioned above, a complex model may result in overfitting. Apart from this, insufficient or unevenly distributed training sample, noise in the training sample, high dimension of features may also result in overfitting. In the experiment 19 where the model has 5 hidden layers and each hidden layer is composed of 40 hidden units, the problem of overfitting occurs. As shown from the performance plot, there is an obvious discrepancy between testing (generated from subset of training samples) and training performance in the training stage, which indicates the possible overfitting. As expected, in the testing stage, the MSE of testing is much worse than that of training (i.e. 0.0279 compared with 0.0032075). To be noticed, MSE value of 0.0032075 is smallest among all experiments where the average MSE value is 0.040558. In order to deal with problem of overfitting, two commonly used solutions are explained below.

2.4.1 Early Stopping

The msot commonly used method for improving generalization is called early stopping. In this technique the available training data is further divided into three subsets. The first subset is the training subset (70%), which is used for computing the gradient and updating the network weights and biases. The second subset is the validation set (15%). The error on the validation set is monitored during the training process. The validation error normally decreases during the initial phase of training, as does the training set error. However, when the network begins to overfit the data, the error on the validation set typically begins to rise. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned.

2.4.2 Regularization

Another method for improving generalization is called regularization. This involves modifying the performance function by adding a term that consists of the mean of the sum of squares of the network weights and bias. With this performance function, the network tends to have smaller weights and biases, and this forces the network response to be smoother and less likely to overfit.

The problem with regularization is that it is difficult to determine the optimum value for the performance ratio parameter. If you make this parameter too large, you might get overfitting. If the ratio is too small, the network does not adequately fit the training data. The solution is the Bayesian framework of David MacKay [MacK92]. In this framework, the weights and biases of the network are assumed to be random variables with specified distributions. The regularization parameters are related to the unknown variances associated with these distributions. You can then estimate these parameters using statistical techniques. Bayesian regularization has been implemented in the function trainbr.

3 Task2: K Nearest Neighbour Classifier

3.1 Pick the optimum k

In practice, cross-validation methods (see e.g., Lachenbruch and Mickey, 1968 [5]; Stone, 1977 [8]) are often used to estimate the misclassification rate for different values of k and choose that one which leads to the lowest estimate of misclassification rate.

Between a minimum k_{min} and maximum k_{max}, suppose that the training set has a cross validation variable with the integer values $1, 2, ..., V$, then the cross validation algorithm is as follows:

1. For each $k \in [k_{min}, k_{max}]$, compute the average error rate or sum-of square error of k: $CV_k = \sum_{v=1}^{V} e_v/V$, where e_v is the error rate or sum-of square error when we apply the Nearest Neighbor model to make predictions on the cases with $X = v$; that is, when we use the other cases as the training dataset.

2. Select the optimal k as: $k^{'} = arg\{minCV_k : k_{min}kk_{max}\}$.
(If multiple values of k are tied on the lowest average error, the smallest k is selected.)

Considering the fact that the value of k is non-parametric and a general rule of thumb in choosing the value of k is $k_{opt} = sqrt(N)/2$, where N stands for the number of training samples, k_{min} is set to $1/2 * k_{opt}$ and k_{max} is set to $3/2 * k_{opt}$.

3.2 Discussion

As seen from the Figure 3, the optimum k is always 1 no matter how many training samples are used. However, because it uses only the training point closest to the query point, the bias of the 1-nearest neighbor (1-NN) estimate is often low, but the variance is high. First of all, the bias of a classifier is the discrepancy between its averaged estimated and true function, whereas the variance of a classifier is the expected divergence of the estimated prediction function from its average value (i.e. how dependent the classifier is on the random sampling made in the training set). Hence, the presence of bias indicates something basically wrong with the model, whereas variance is also bad, but a model with high variance could at least predict well on average. However, since 1-NN is too dependent on the training set, the probability that it models the noise in the training data is really high. In this case, the performance of 1-NN classifier would be highly influenced if there is noise in the training set. In one of experiments where there are 1000 manually generated noise samples among the total 7000 training samples, the performance of 1-NN is much worse than 5-NN or 10-NN, as shown in Figure 4, and the performance further improves with the increase of k.

In another set of experiments, as shown in Figure 5, for different data splits, the performance is different - the performance improves with the increase of training data samples.

4 Comparison

4.1 Easy to use

One reason KNN is such a common and widely-known algorithm is its ease of implementation. Indeed, the actual implementation of the core algorithm is only of several lines of code. On top of that, KNN is pleasingly parallel, and inherently flexible. Unlike MLP which has a number of parameters including the learning rute, epoch, number of hidden layers and neurons, stopping condition and etc., KNN only equires few parameters,

4.2 Computational Complexity

In terms of KNN, assuming k is fixed, the runtime is $O(nd+kn)$ where n is the number of samples in the training set, and d is the dimenssion of the input. With regard to MLP, it is hard to give a accurate runtime calculation since it depends on the network efficiency settings that control the speed/storage tradeoff. Roughly, for the training session, the time complexity of the standard back-propagation algorithm is around $O(W^3)$ where W is the count of weights in the network. In this case, it is meaningless to simply compare time complexity of two algorithms without considering the actual implementation. In most cases, KNN takes multiple times more computations than does the MLP [7].

4.3 Recognition Performance

Two figures below show the best performance of two algorithms. The recognition performances of the MLP (784-40-40-40-10) classifier are given in Figure 6 (a). The recognition performances of the 1-NN classifierare given in Figure 6 (b). It turns out 1-NN performs relatively better than MLP (784-40-40-40-10) for the task of handwritten digits recognition. However, based on the experiment conducted by ROY [7], MLP should perform better than KNN. According to him, both the KNN classifier and the MLP can approximate arbitrary decision regions for classification of unknown patterns in the feature space. The performance of the k-NN classifier highly depends on the reference set and the value of k chosen for the work. The reference set again depends on the chosen distance metric. Thus the scope for fine tuning of the k-NN classifiers performance is much limited. By contrast, in terms of the MLP, the number of hidden layer and neurons and tunable connection weights of the MLP offer a wider range of parameters for fine tuning of the decision regions. Because of this, the recognition performances of the MLP can be theoretically better compared to those of the KNN when classifying the handwritten digits.

Appendices

A. Task 1

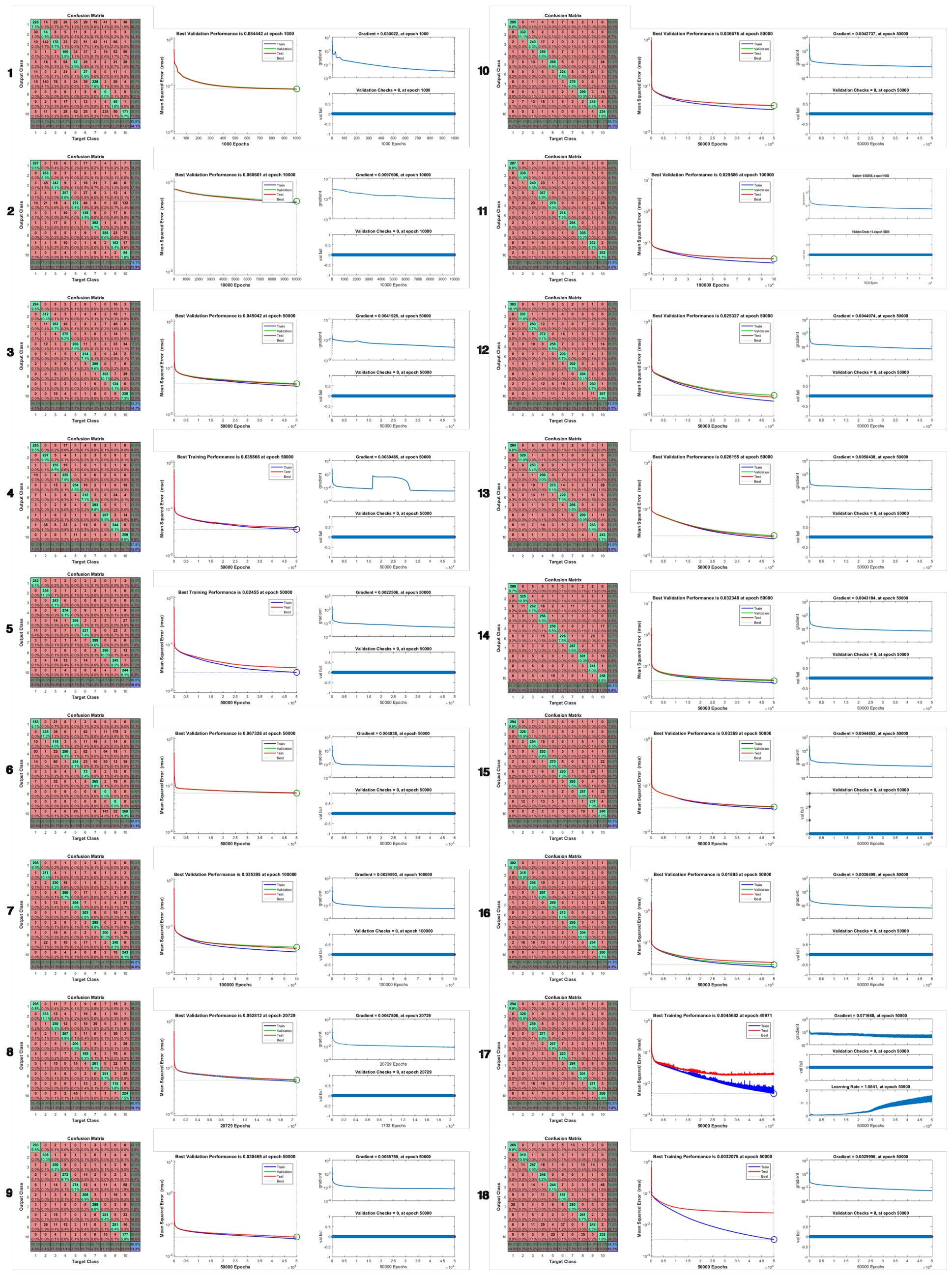


Figure 1: Plots of all 18 experiments

ID	Training Samples	Testing Samples	Training Function	Learning rate	Epoches	Hidden Layer	Time	Training Performance	Testing Performance	Recognition Rate	Precision	Recall
1	7000	3000	traingd	0.01	1000	1(10)	17''	0.084442	0.0844	35.90%	0.3172	0.2417
2	7000	3000	traingd	0.01	10000	1(10)	2'49''	0.060601	0.0692	74.10%	0.7757	0.441
3	7000	3000	traingd	0.01	50000	1(10)	15'56''	0.045042	0.0416	85.30%	0.8613	0.466
4	7000	3000	traingd	0.02	50000	1(10)	11'19''	0.035866	0.0393	87.4%%	0.8737	0.4693
5	7000	3000	traingd	0.03	50000	1(10)	12'14''	0.02455	0.0287	90.00%	0.8989	0.4757
6	7000	3000	traingd	0.01	100000	1(10)	29'19''	0.035395	0.0322	89.20%	0.893	0.4743
7	7000	3000	traingd	0.012	20000/50000	1(10)	> 74'38''	0.052812	0.0523	80.90%	0.8096	0.4518
8	7000	3000	traingd	0.01	50000	1(5)	16'52''	0.067326	0.0587	58.90%	0.503	0.3121
9	7000	3000	traingd	0.01	50000	1(15)	21'02''	0.038469	0.0381	86.80%	0.8746	0.4691
10	7000	3000	traingd	0.01	50000	1(20)	24'22''	0.036676	0.0345	89.50%	0.8952	0.4748
11	7000	3000	traingd	0.01	100000	1(20)	37'33''	0.029586	0.0297	91.40%	0.9154	0.4798
13	7000	3000	traingd	0.01	50000	2(20, 20)	21'22''	0.025327	0.0214	91.50%	0.9155	0.48
14	7000	3000	traingd	0.01	50000	3(20, 20, 20)	20'25''	0.026155	0.0235	91.00%	0.9106	0.4784
15	7000	3000	traingd	0.01	50000	1(25)	19'13''	0.032348	0.0329	90.10%	0.9002	0.4761
16	7000	3000	traingdm	0.01	50000	1(20)	97'38''	0.03369	0.0336	89.20%	0.8918	0.4738
17	7000	3000	traingd	0.01	50000	3(40, 40, 40)	43'32''	0.020643	0.021	92.70%	0.9263	0.4827
18	7000	3000	traingda	0.01	50000	1(20)	25'32''	0.0045682	0.0156	92.20%	0.925	0.482
19	1000	3000	traingd	0.01	50000	5(40, 40, 40, 40, 40, 40)	9'57''	0.0032075	0.0279	84.50%	0.85	0.4625

Figure 2: Summary of all 18 experiments

B. Task 2

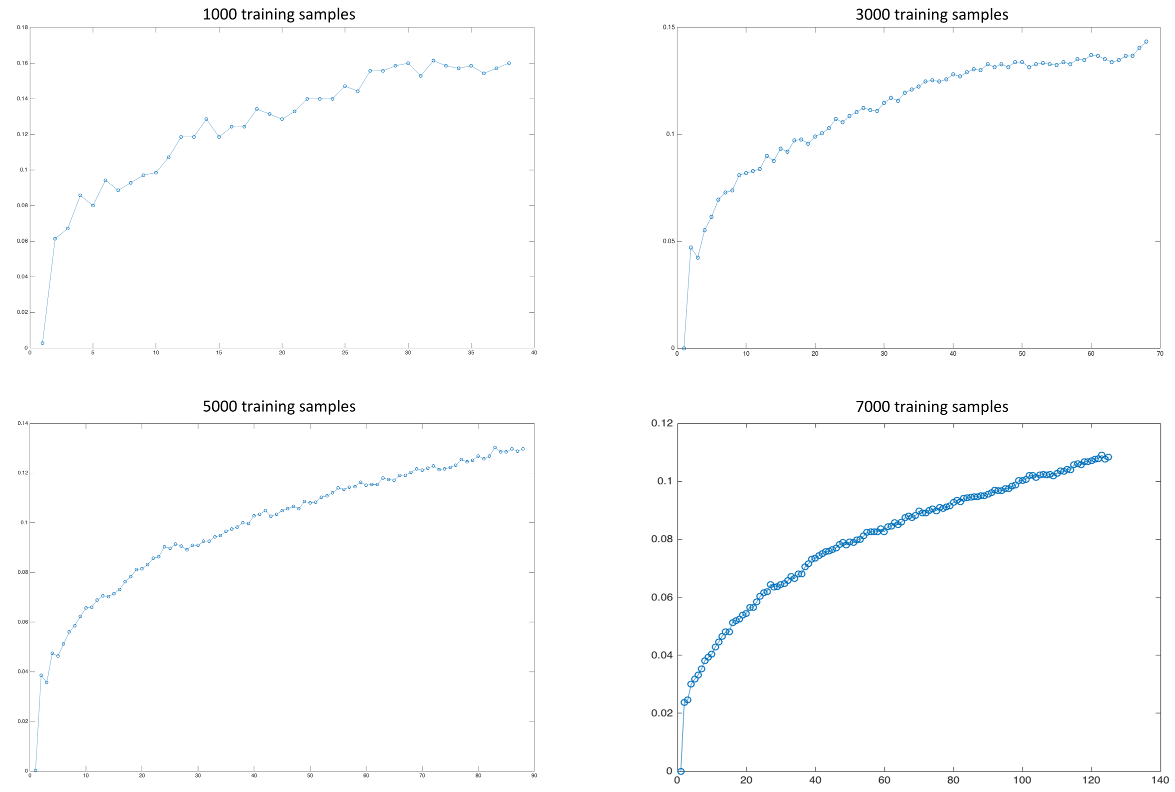


Figure 3: Choice of k and performance under different training samples

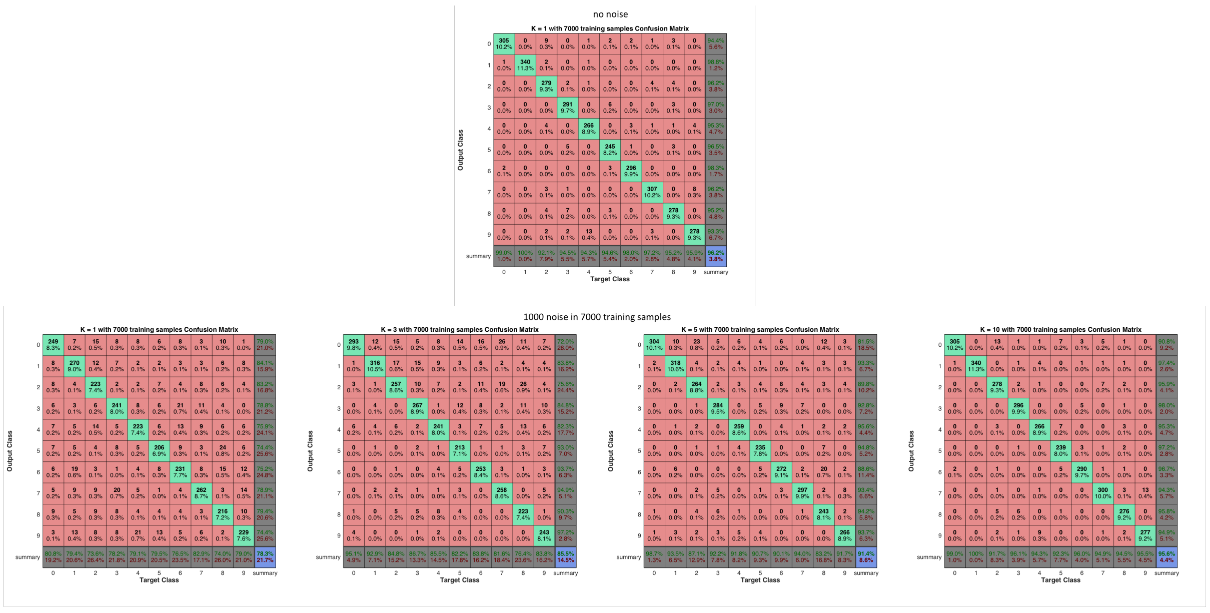
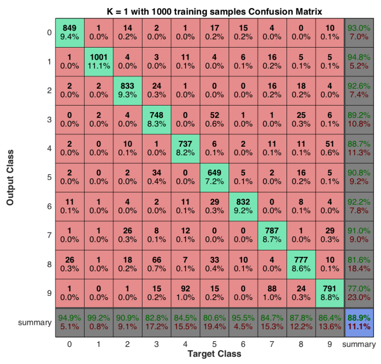
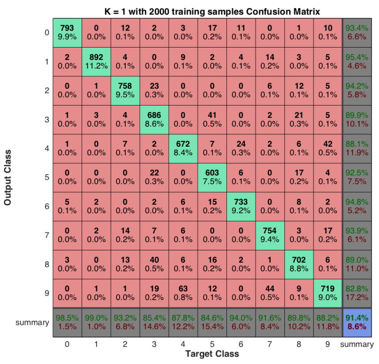
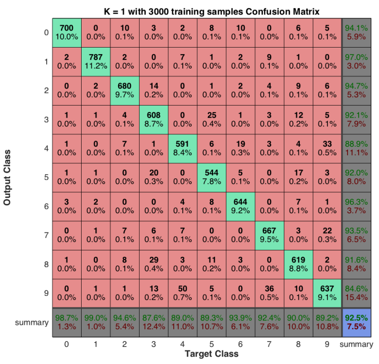
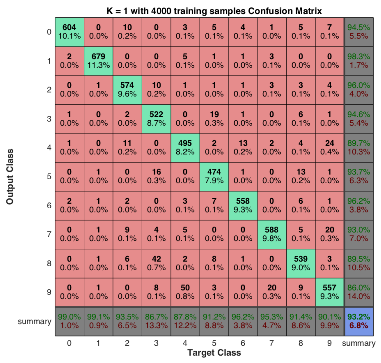


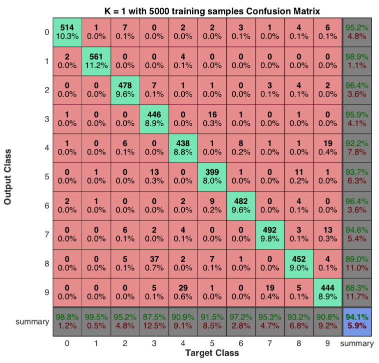
Figure 4: Performance with different k when noise in the training samples

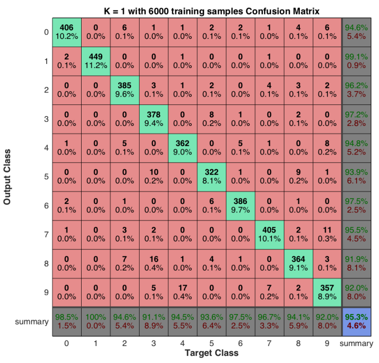


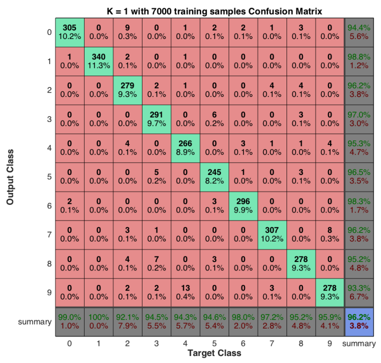


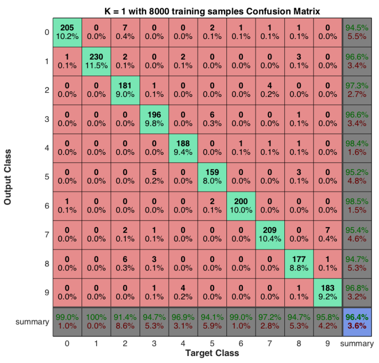












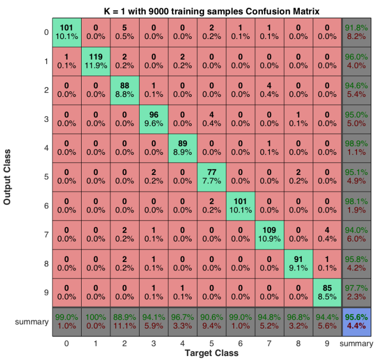
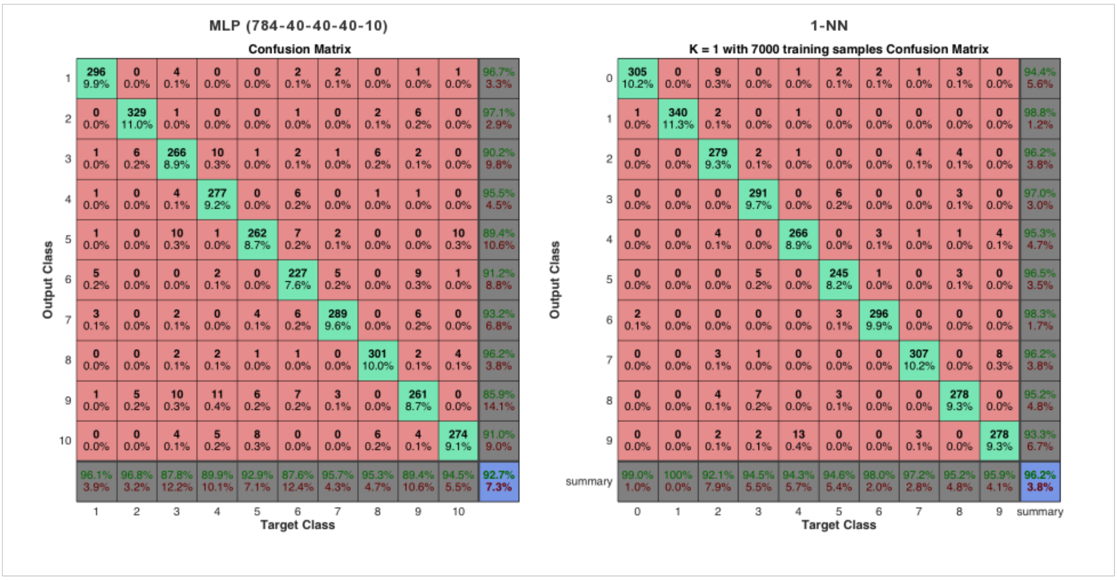


Figure 5: Performance with $k = 1$ under different data splits

C. Task 3



MLP (784-40-40-40-10)

Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	296 9.9%	0 0.0%	4 0.1%	0 0.0%	0 0.0%	2 0.1%	2 0.1%	0 0.0%	1 0.0%	1 0.0%	96.7% 3.3%
2	0 0.0%	329 11.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	2 0.1%	6 0.2%	0 0.0%	97.1% 2.9%
3	1 0.0%	6 0.2%	266 8.9%	10 0.3%	1 0.0%	2 0.1%	1 0.0%	6 0.2%	2 0.1%	0 0.0%	90.2% 9.8%
4	1 0.0%	0 0.0%	4 0.1%	277 9.2%	0 0.0%	6 0.2%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	95.5% 4.5%
5	1 0.0%	0 0.0%	10 0.3%	1 0.0%	262 8.7%	7 0.2%	2 0.1%	0 0.0%	0 0.0%	10 0.3%	89.4% 10.6%
6	5 0.2%	0 0.0%	0 0.0%	2 0.1%	0 0.0%	227 7.6%	5 0.2%	0 0.0%	9 0.3%	1 0.0%	91.2% 8.8%
7	3 0.1%	0 0.0%	2 0.1%	0 0.0%	4 0.1%	6 0.2%	289 9.6%	0 0.0%	6 0.2%	0 0.0%	93.2% 6.8%
8	0 0.0%	0 0.0%	2 0.1%	2 0.1%	1 0.0%	1 0.0%	0 0.0%	301 10.0%	2 0.1%	4 0.1%	96.2% 3.8%
9	1 0.0%	5 0.2%	10 0.3%	11 0.4%	6 0.2%	7 0.2%	3 0.1%	0 0.0%	261 8.7%	0 0.0%	85.9% 14.1%
10	0 0.0%	0 0.0%	4 0.1%	5 0.2%	8 0.3%	0 0.0%	0 0.0%	6 0.2%	4 0.1%	274 9.1%	91.0% 9.0%
	96.1% 3.9%	96.8% 3.2%	87.8% 12.2%	89.9% 10.1%	92.9% 7.1%	87.6% 12.4%	95.7% 4.3%	95.3% 4.7%	89.4% 10.6%	94.5% 5.5%	92.7% 7.3%

1-NN

K = 1 with 7000 training samples Confusion Matrix

Output Class \ Target Class	0	1	2	3	4	5	6	7	8	9	summary
0	305 10.2%	0 0.0%	9 0.3%	0 0.0%	1 0.0%	2 0.1%	2 0.1%	1 0.0%	3 0.1%	0 0.0%	94.4% 5.6%
1	1 0.0%	340 11.3%	2 0.1%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	98.8% 1.2%
2	0 0.0%	0 0.0%	279 9.3%	2 0.1%	1 0.0%	0 0.0%	0 0.0%	4 0.1%	4 0.1%	0 0.0%	96.2% 3.8%
3	0 0.0%	0 0.0%	0 0.0%	291 9.7%	0 0.0%	6 0.2%	0 0.0%	0 0.0%	3 0.1%	0 0.0%	97.0% 3.0%
4	0 0.0%	0 0.0%	4 0.1%	0 0.0%	266 8.9%	0 0.0%	3 0.1%	1 0.0%	1 0.0%	4 0.1%	95.3% 4.7%
5	0 0.0%	0 0.0%	0 0.0%	5 0.2%	0 0.0%	245 8.2%	1 0.0%	0 0.0%	3 0.1%	0 0.0%	96.5% 3.5%
6	2 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	3 0.1%	296 9.9%	0 0.0%	0 0.0%	0 0.0%	98.3% 1.7%
7	0 0.0%	0 0.0%	3 0.1%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	307 10.2%	0 0.0%	8 0.3%	96.2% 3.8%
8	0 0.0%	0 0.0%	4 0.1%	7 0.2%	0 0.0%	3 0.1%	0 0.0%	0 0.0%	278 9.3%	0 0.0%	95.2% 4.8%
9	0 0.0%	0 0.0%	2 0.1%	2 0.1%	13 0.4%	0 0.0%	0 0.0%	3 0.1%	0 0.0%	278 9.3%	93.3% 6.7%
summary	99.0% 1.0%	100% 0.0%	92.1% 7.9%	94.5% 5.5%	94.3% 5.7%	94.6% 5.4%	98.0% 2.0%	97.2% 2.8%	95.2% 4.8%	95.9% 4.1%	96.2% 3.8%

Figure 6: Best performance of MLP and KNN under 7000 training samples and 3000 testing samples

References

[1] Berry, M. J., and Linoff, G. *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc., 1997.

[2] Blum, A. *Neural networks in C++: an object-oriented framework for building connectionist systems*. John Wiley & Sons, Inc., 1992.

[3] Boger, Z., and Guterman, H. Knowledge extraction from artificial neural network models. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on* (1997), vol. 4, IEEE, pp. 3030–3035.

[4] Csáji, B. C. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary 24* (2001), 48.

[5] Lachenbruch, P. A., and Mickey, M. R. Estimation of error rates in discriminant analysis. *Technometrics 10*, 1 (1968), 1–11.

[6] LeCun, Y. C. C. C. J. B. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges. http://yann.lecun.com/exdb/mnist/, 2016. [Online; accessed 11-December-2016].

[7] Roy, K., Chaudhuri, C., Kundu, M., Nasipuri, M., and Basu, D. Comparison of the multilayer perceptron and the nearest neighbor classifier for handwritten digit recognition. *Journal of Information Science and Engineering 21*, 6 (2005), 1245–1257.

[8] Stone, M. Cross-validation: a review 2. *Statistics: A Journal of Theoretical and Applied Statistics 9*, 1 (1978), 127–139.

[9] Swingler, K. *Applying neural networks: a practical guide*. Morgan Kaufmann, 1996.