PCML Mini-project Report

Group 18: Tasorn Sornnarong and Tian Guo

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
{tian.guo,sornnarong.tasorn}@epfl.ch

1 Multi-Layer Perceptron (MLP) part

1.1 Introduction

The aim of this miniproject is to get the experience on implementing two different machine learning methods for solving the same hand-written digits pattern recognition problem. We are supposed to understand and use both error back-propagation gradient descent optimization for a multi-layer perceptron (MLP) and a support vector machine (SVM) method, then apply to the MNIST dataset. So, we can compare the performance for each methods and discuss how and why the results behaved.

1.2 Methods

(1) The given .mat files contain training and test sets for the two binary classification. For the multi-layer perceptron (MLP), we are supposed to distinguish between digits 3 and 5 and between digits 4 and 9. Each image of a digit is stored as a vector of length 784. The provided files contain pattern matrices of size n×784, where n is the number of patterns. The training set is used for training the data and updating the parameter, and the test set is used only for evaluation of the final performance.

The next step for the multi-layer perceptron (MLP) is to split the training part into a training set (2/3) and a validation set (1/3). In the provided training data there are n=6,000 patterns, so after splitting the the size of the training set is 4,000×784 and the test set is 2,000×784.

The first thing we have to do for machine learning problem is the preprocessing such as loading the datasets in the directory path we created, and then normalizing all of the data to have coefficients between 0 and 1. To do this, we compute the maximum α_{max} and the minimum α_{min} over all coefficients of all training patterns, then substitute each input pattern by the following equation

$$x_{normalized} = \frac{1}{\alpha_{max} - \alpha_{min}}(x - \alpha_{min}\mathbf{1}) \tag{1}$$

For the test set, we also normalize all pattern by using the same coefficients α_{max} and α_{min} as stored in the training patterns because we cannot treat and compute the test data even for preprocessing.

(2) The final setup of the MLP: the network structure for this project, we use a two-layer MLP, with one hidden and one output layer, use the gating transfer function as the following

$$g(a_1, a_2) = a_1\sigma(a_2), \quad \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

for the hidden units. The binary classifier and the activation for each layer are given by

$$f(x) = \text{sgn}(a^{(2)}(x)), \quad a^{(2)}(x) = \sum_{q=1}^{h_1} w_q^{(2)} g(a_{2q-1}^{(1)}, a_{2q}^{(1)}) + b^{(2)} \tag{3}$$

$$a_k^{(1)} = \sum_{j=1}^{d} w_{kj}^{(1)} x_j + b_k^{(1)}, \quad k = 1, 2, \ldots, 2h_1 \tag{4}$$

In the code, we are supposed to minimize the logistic error function, which evaluated the whole training set, is given by

$$E_{log}(w) = \frac{1}{n} \sum_{i=1}^{n} log(1 + e^{-t_i a^{(2)}(x_i)}) \tag{5}$$

where the binary training dataset is $D = \{(x_i, t_i), i = 1, ..., n\}$ and t_i is -1 or 1. To do this, we have to evaluate by using the error back-propagation gradient descent optimization, which is to take the gradient with respect to the parameters (weight and bias). For the output layer, the residual is

$$\frac{\partial E_i}{\partial b^{(2)}} = r^{(2)} = \frac{\partial E_i}{\partial a^{(2)}} = -t_i \sigma(t_i a^{(2)}(x_i)) \tag{6}$$

$$\nabla_{w_q^{(2)}} E_i = r^{(2)} \cdot \frac{\partial a^{(2)}}{\partial w_q^{(2)}} = r^{(2)} \frac{\partial}{\partial w_q^{(2)}} (w_q^{(2)} g(a_{2q-1}^{(1)}, a_{2q}^{(1)}) + b^{(2)}) = g(a_{2q-1}^{(1)}, a_{2q}^{(1)}) \tag{7}$$

For the hidden layer, the residual is given by

$$\frac{\partial E_i}{\partial a^{(1)}} = \frac{\partial E_i}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial g} \cdot \frac{\partial g}{\partial a^{(1)}} = r^{(2)} w^{(2)} g' = r_k^{(1)} \tag{8}$$

The derivative of the transfer function g is $g'(a_1, a_2) = \begin{cases} \sigma(a_2) & \text{, k is odd} \\ a_1 \sigma(a_2)(1 - \sigma(a_2)) & \text{, k is even} \end{cases}$

As a result, the residual for the first layer is expressed as $r_k^{(1)} = \begin{cases} r^{(2)} w^{(2)} \sigma(a_2) & \text{, k is odd} \\ r^{(2)} w^{(2)} a_1 \sigma(a_2)(1 - \sigma(a_2)) & \text{, k is even} \end{cases}$

And the gradient with respect to weight $w_{kj}^{(1)}$ is thus equal to

$$\nabla_{w_{kj}^{(1)}} E_i = r_k^{(1)} \frac{\partial}{\partial w_{kj}^{(1)}} (w_{kj}^{(1)} x_j + b_k^{(1)}) = r_k^{(1)} x_j \tag{9}$$

For updating the new parameter, we ran the gradient descent with momentum which works as follows

$$\Delta w_i = -\eta(1 - \mu) \nabla_{w_i} E + \mu \Delta w_{i-1} \tag{10}$$

denoted by $\Delta w_i = w_{i+1} - w_i$ the update done in iteration i.

We can test if the backpropagation code for the gradient computation is bug-free or not. For the logistic error function E(w) of a single weight w, the gradient of error is $g(w) = E'(w) = dE/dw$ Gradient testing will be done by comparing the result of gradient g(w) with symmetric finite-difference approximation that is the evaluation of the error function at two other points that are very close to w, given by

$$g(w) \simeq \frac{E(w + \epsilon) - E(w - \epsilon)}{2\epsilon} \tag{11}$$

for a very small $\epsilon > 0$ (we used $\epsilon = 0.001$)
We have tested that the above equation satisfied. It means that our backpropagation code works.
To select the best parameters, we ran the MLP algorithm with the various parameters of the number of hidden units $h_1 = 10, 20, ..., 100$, the learning rate $\eta = 0.01, 0.03, 0.1, 0.3, 1, 3, 9$ and the momentum term $\mu = 0.1, 0.2, ..., 1.0$ and stored the logistic error in the form of matrices with 4-dimension depending on each parameters h1,η, μ and iterations (N=50). Next, we see the effects of each parameters by fixing the remaining parameters for every iterations. The experiment tells us that if we do not select the parameter appropriately, we will face some problems. According to the figure 1, we can see the error with different number of hidden units h_1. It showed about the overfitting problem obviously that the higher h_1 is, the larger the error is and the training and validation error are oscillated and its difference is pretty larger. Compared to $h_1 = 30$, there is almost no difference between two errors. Both are almost fit exactly and converge to 0.

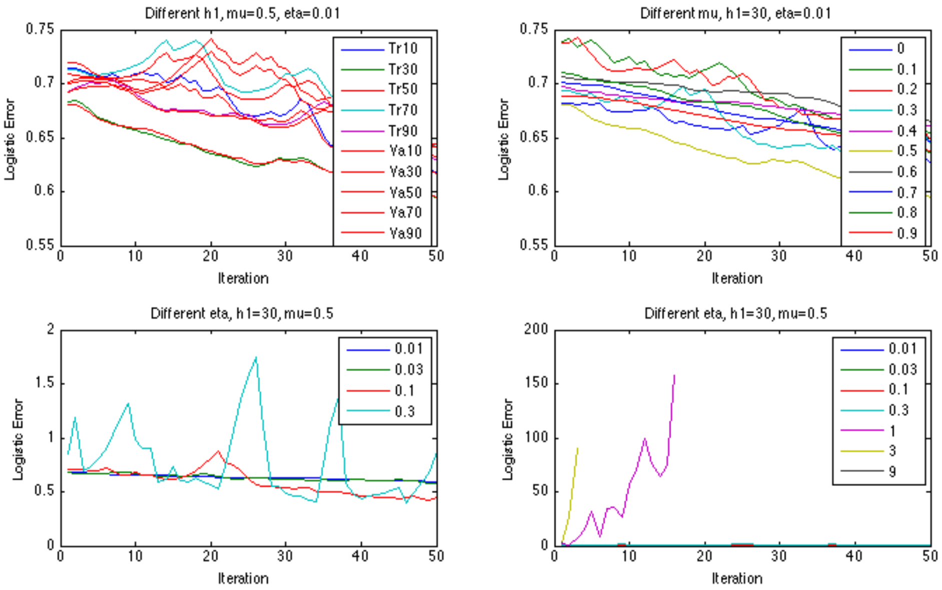

Fig. 1. Model Selection (1)different h_1 (2)different μ (3)different $\eta = 0.01 - 0.3$ (3)different $\eta = 0.01 - 9$ for digits 3and5

In the same figure, a small learning rate η like 0.01, the error decreases slowly, which means slow convergence during gradient descent. Also, while $\eta = 0.3$, it has a faster convergence. After a certain point, increasing η=3 or 9 will no longer increase the convergence speed. Similarly, if the momentum term μ is too small, the convergence speed is too fast and eventually does not converge due to oscillation (zig-zagging). The results follow from the theory we have learnt.
For each selection, we selected the best parameter that gives the minimum logistic error has no over fitting and not too fast convergence speed. As a result, for example we choose the number of hidden units $h_1 = 30$, the learning rate $\eta = 0.01$ and the momentum term $\mu = 0.5$ for digits 3 and 5 binary classification.

(3) For each binary subtasks, we used the different parameters. The method for model selection, and the chosen parameters and its plots for digits 3 and 5 has been shown in the previous part.

We conclude the chosen parameters as the following.
The 3 and 5 binary subtask, $h_1 = 30$, $\eta = 0.01$, $\mu = 0.5$.
The 4 and 9 binary subtask, $h_1 = 70$, $\eta = 0.01$, $\mu = 0.3$.

(4) To see one example of overfitting, let us use a large number of hidden units h_1 (says, 100) and do not use the early stopping. Then plot the graph to show the comparison between training and validation error as in figure 2

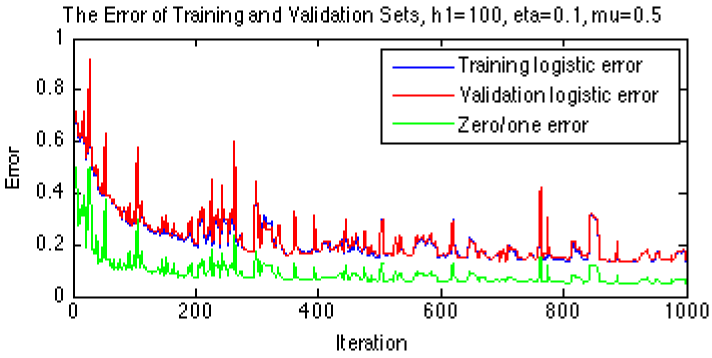

Fig. 2. The example of overfitting

1.3 Result and Discussion

1. By learning the multi-layer perceptron (MLP), we got the training set logistic error, the validation set logistic error, and the validation set (zero/one) error as a function of the epoch number

$$\frac{1}{m}\sum_{j=1}^{m} \mathbf{1}_{\{t_j a^{(2)}(x_j) \leq 0\}}. \tag{12}$$

   The error curves for digits 3 and 5, and 4 and 9 binary classification have been shown in figure 3(1) and 3(2) respectively.
2. After learning MLP, for 3and5 subtask the training error is 0.2005, so the accuracy is 99.7995%. for 4and9 subtask, the training error and the accuracy are 0.2095 and 99.7905%, we got the latest updated parameters (weight and bias). We used it to compute the error and the activation over the whole test dataset. Then, we evaluate the misclassified data by computing $t_i a^{(2)}(x_i)$ for each pattern. For digits 3 and 5 binary classification, the result is illustrated in figure 4(1), the standard deviation is $\sigma = 1.7550$ and the error of test set is 0.1855 (accuracy=99.8145%). For digits 4 and 9 binary classification, the result is illustrated in figure 4(2),the standard deviation is $\sigma = 2.4752$ and the error of test set is 1.1046 (accuracy=98.8954%). The figures showed that the more we trained the data, the more $t_i a^{(2)}(x_i)$ converges to zero. It means that the classification problem works more accurately.
3. Let us give the example of digits 3 and 5 binary subtask. For each run, in the patterns which are misclassified, we stored the values $t_i a^{(2)}(x_i)$ which has largest negative and close to zero. Then, we separately plot the graph in figure 5.

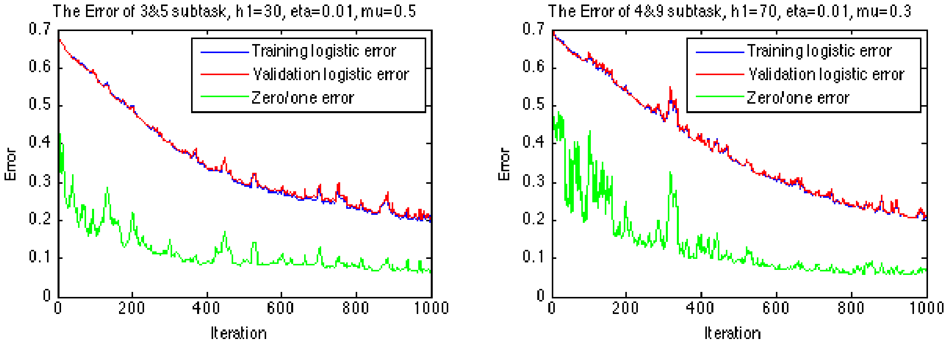

Fig. 3. The training set logistic error, the validation set logistic error, and the zero/one error for each iterations for digits 3and5 (1) and 4and9 (2)

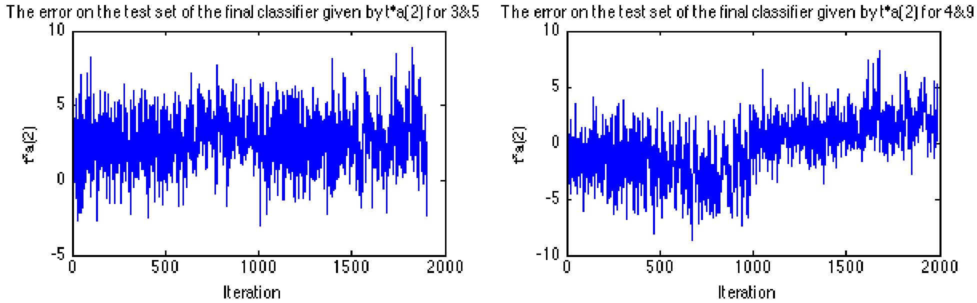

Fig. 4. The computation of the average of $t_i a^{(2)}(x_i)$ for each iterations for digits 3and5 (1) and 4and9 subtask (2)

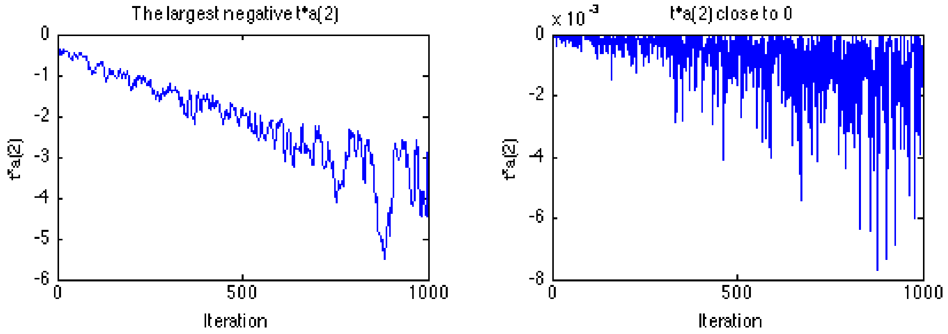

Fig. 5. $t_i a^{(2)}(x_i)$ which has (1) largest negative and (2) close to zero

2 SVM

2.1 Introduction

2.2 Methods

In this part, we discuss some details about the implementation of SMO.

1. First, we normalized the training and test data. Then permutate the training data set by iterating over the patterns and swap the current one to a random position of the data set.

2. In order to conduct 10-fold cross validation, we came up a training and validation set construction procedure over the permutated data set. The idea is that we divide the training set into ten equal-sized parts. For a specific setup of C and τ, the procedure sequentially extracts the i-th ($i = 1, \cdots, 10$) part as the validation set and combines the other parts as the training set.

In each training loop, we maintain a binary index to efficiently update the set I_{up} and I_{low}. Such binary index has the same number of entries as that of patterns in the training set. For instance, value 1 in the i-th entry of I_{up} index indicates the i-th pattern belongs to the set I_{up}. In this way, we only need to change the ownership of the chosen violation pair. Then we invoke the max (or min) function to get the most violated pair for next loop.

2.3 Results & Discussion

1.

Table 1. Matrix of 10-fold CV scores

	$\tau = 0.02$	$\tau = 0.04$	$\tau = 0.06$	$\tau = 0.08$
$C = 4$	1.613757	0.902137	0.582406	**0.406962**
$C = 5$	1.655328	0.901502	0.583424	0.407138
$C = 6$	1.652844	0.905430	0.583432	0.407138
$C = 7$	1.662291	0.902036	0.583348	0.407138
$C = 8$	1.674960	0.906631	0.583348	0.407138

Based on the experience of program tuning, we set the domains of parameter C and τ in $[4, 8]$ and $[0.02, 0.08]$ respectively to find the best configuration. In Tab. 1, we present the matrix of 10-fold cross validation scores. We can see that the winning configuration is C=4 and τ=0.08 which achieves the minimal CV score.

2.

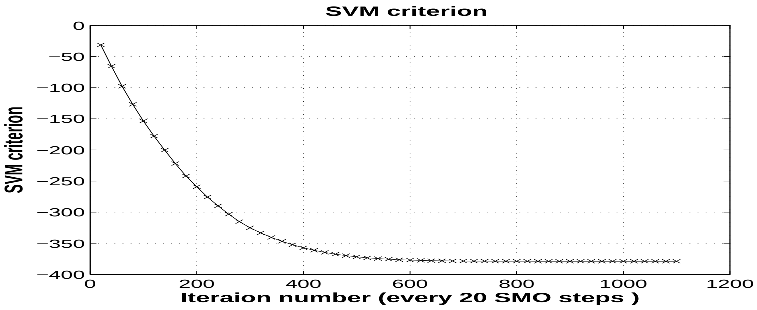

Fig. 6. SVM criterion as a function of iteration number

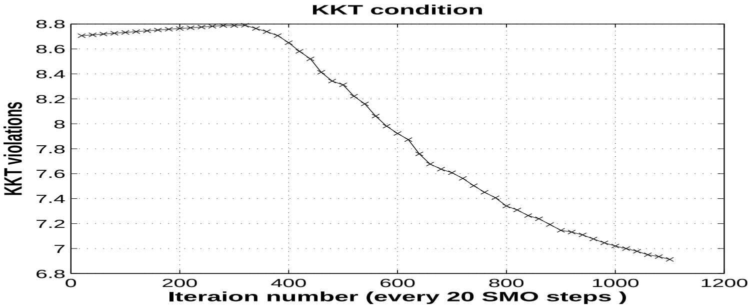

Fig. 7. KKT violations as a function of iteration number

In this part, we run our SMO algorithm on the full training set, with the parameters found by above cross-validation.

Fig. 6 presents the decreasing trend of SVM criterion as a function of SMO iterations. After 600 iterations, the SVM criterion decreases slowly.

Fig. 7 exhibits the variation of number of KKT condition violations in the training period. In the beginning phase, the KKT violations increases a little. This is because initially the set I_0 contains no patterns. Then when it has more elements, the violation number increases as the patterns in I_0 belong to both I_+ and I_-.

3. In this experiment, we make use of the final SVM classifier trained by our SMO algorithm with the configuration from cross validation to perform on the whole training and test set. The zero/one error on training set is zero, while the misclassification rate for test set is $\frac{17}{1991} = 0.85\%$

4. In the last experiment, we compare the classification performance of the final MLP and SVM on both the training and test data. Fig. 8 exhibits the results. We can see that the SVM outperforms the MLP on the training and test data sets. We leverage the Gaussian kernel which is an exponentially decaying function in the input feature space in SVM and therefore SVM works in a theoretically infinite feature space, which improves the separability of original data set. In the meanwhile, both MLP and SVM are more precise for training data than for test data. This is easy to understand, because the classifier are learnt from the training data with convergence threshold to ensure the precision.

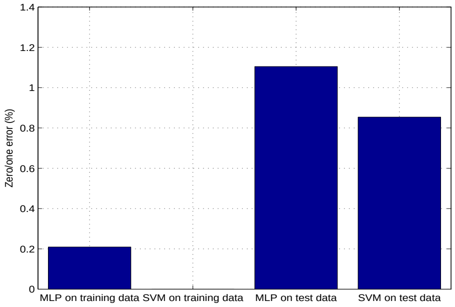

Fig. 8. Comparison of zero/one errors of final MLP and SVM on training and test data set