Pattern Classification and Machine Learning

Miniproject

Taha Elgraini (SCIPER ID: 238189), Fabian Brix (SCIPER ID: 236334)
Group 22, Python

December 16, 2013

1 Introduction

The imperative of this project was to apply two different machine learning techniques, the Multilayer Perceptron (henceforth referred to as "MLP") and the Support Vector Machine (henceforth "SVM") to the recognition of hand-written digits. More specifically error back-propagation gradient descent optimization had to be implemented for MLP and the Sequential Minimal Optimization (SMO) algorithm had to be implemented for SVM to solve problems posed with th0,00140e MNIST dataset. The MLP had to be trained on datasets of digits 3 and 5 as well as of 4 and 5, whereas the SVM classifier only had to be trained on the 3/5 problem. Both classifiers should accurately classify the digits in a test set provided labelling information of the training set. The focus thereby lay on a good understanding and clean implementation of the aforementioned algorithms as well as a sound evaluation of their performance.

2 Multilayer Perceptron

2.1 Methods

This section covers how we went about implementing the MLP back-progagation gradient descent algorithm and which parameters we chose to make to make it perform well on the validation set.

2.1.1 Treatment of Data

First of all, before passing it on to the algorithm and training, we had to process the raw data of labelled digits.

Splitting The data was already separated into training and test data containing the pattern vectors for the digits and the corresponding classification labels. All what was left to do according to the instructions received was to split the training data into a random $\frac{2}{3}$ training and a $\frac{1}{3}$ validation set. This was easily achieved by randomly permuting the indices of the training data patterns and then dividing both the set of patterns and the set of labels at the $\frac{2}{3}$ mark according to the permutation.

Preprocessing Before starting off with training the MLP it had to be ensured that both training and validation set were properly normalized. As given in the project description, we computed the min. and max. values for every component of the untouched training patterns beforehand and then applied the normalization to the training and the test data. This preprocessing was actually done before splitting the data in order to save the operation on the validation set.

2.1.2 MLP setup

The structure of the MLP we were advised to use through the project instructions is depicted in Fig. 1. The differences to the two-layer MLP discussed in the lecture lie in the following properties: Firstly, the first hidden layer has double the outputs than a "normal" MLP perceptron. Secondly, the gating function $g(a_{2k-1}, a_{2k}) = a_{2k-1}\sigma(a_2 k)$ takes pairs of values from the hidden layer accordingly. Lastly,

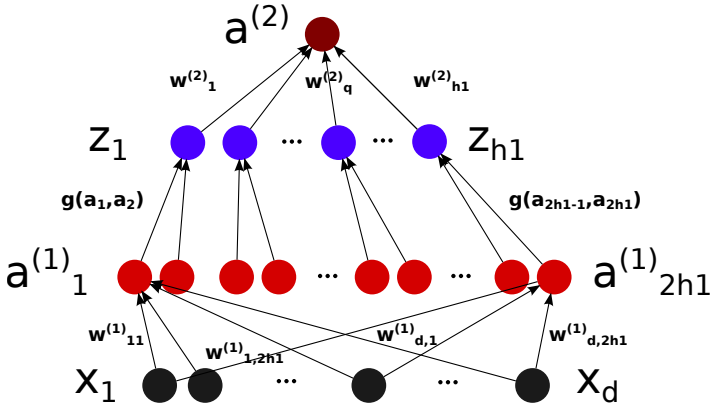
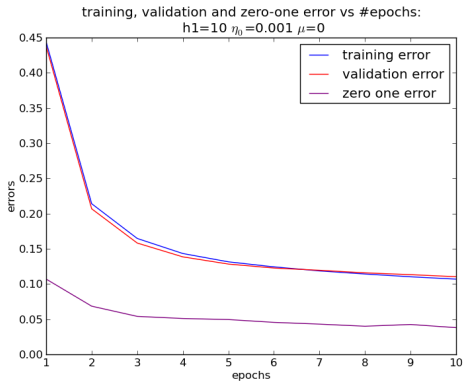


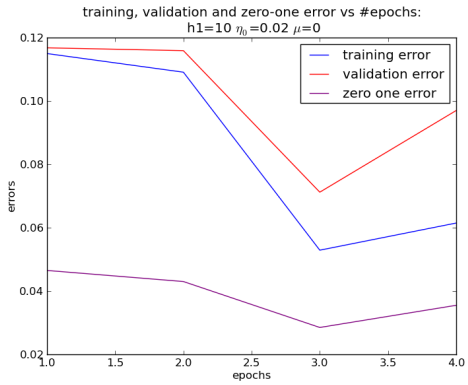
Figure 1: Schema of the MLP described in the project instructions

the activation value $a^{(2)}$ is not passed through another gating function. Once we had the code for this setup running in python, we left the number of hidden layers and the gating function untouched and proceeded with the optimization of the remaining parameters, the number of hidden inputs h_1, the learning rate η and the momentum term μ. The discussion of the effects of the adjustment of these parameters is discussed with early stopping meaning that the optimization stops as soon as the validation error starts increasing. The 0/1 errors displayed in the graphs of this section are always specific to the respective validation set. The graphs in this section were generated on the 3/5 subtask.

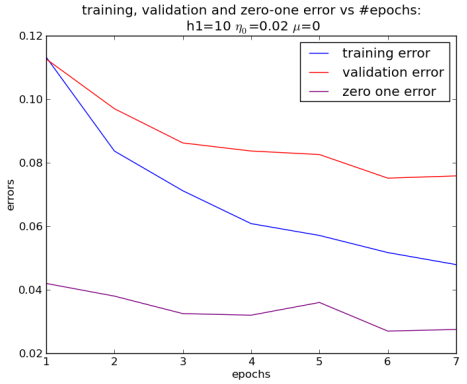
Effects of parameters on convergence speed and errors



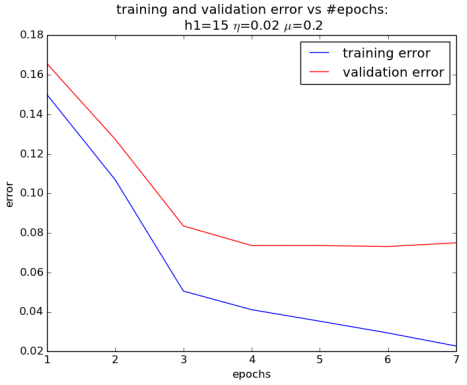
(a) Errors with small $\eta_0 = 0.001$



(b) Errors decrease with larger $\eta_0 = 0.02$



(c) Decreasing η to $\eta = \frac{\eta_0}{epochs}$ after every epoch



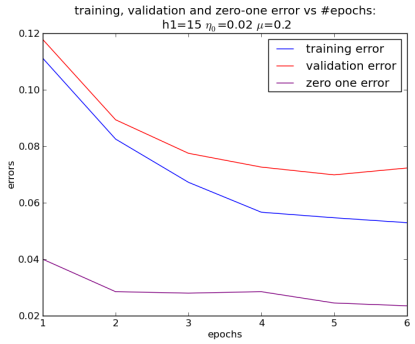
(d) Errors after introducing momentum term

Figure 2: This figure exemplifies the effects of increasing the initial learning rate η_0 and then decreasing it every epoch, as well as the effect of introducing a momentum term stabilizing the gradient descent.

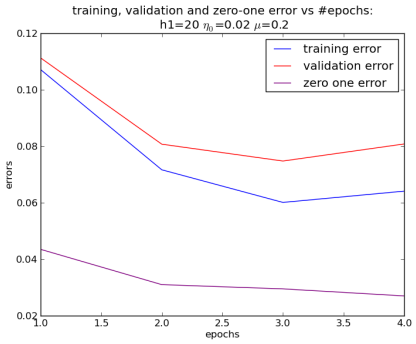
- In fig. 2 we show the effects of the learning rate η and the momentum term μ on the convergence

speed and the accuracy of the MLP classifier. By increasing the initial learning rate η_0 we were able to decrease the number of epochs until an increase in the validation error occurs and improve the validation error. After decreasing the learning rate in every epoch, the optimization again takes a some epochs more to converge. Introducing a standard momentum term $\mu = 0.2$ as in fig. 2d doesn't improve the convergence speed or the error values in our case. Before the decreasing the η term in the manner depicted in subfig. caption 2c we used momentum terms around 0.2. Generally a higher learning rate makes the algorithm converge faster, this effect is increased by introducing the momentum term which is the decay-term for averaging the previous the gradients.

- Effects of number hidden units h_1



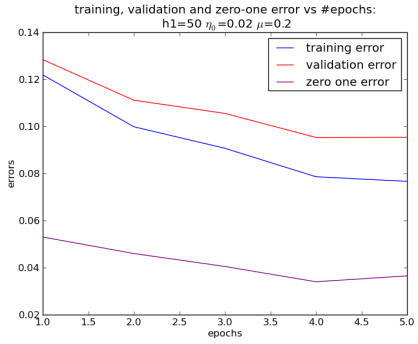
(a) $h_1 = 15$



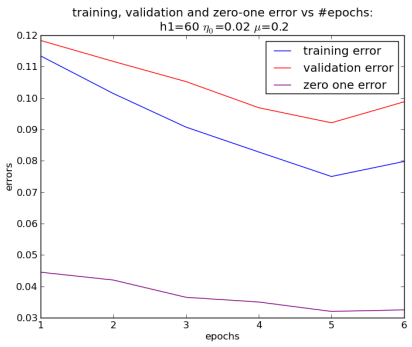
(b) $h_1 = 20$

(c) $h_1 = 30$

(d) $h_1 = 40$



(e) $h_1 = 50$



(f) $h_1 = 60$

Figure 3: Plotting graphs of the logistical error for different numbers of hidden units shows different performances and convergence in different numbers of epochs. No general pattern emerged in the depicted plots generated with early stopping. However, the model complexity should increase with the number of hidden units leading to an overfitting of the training set. More on this in the next section.

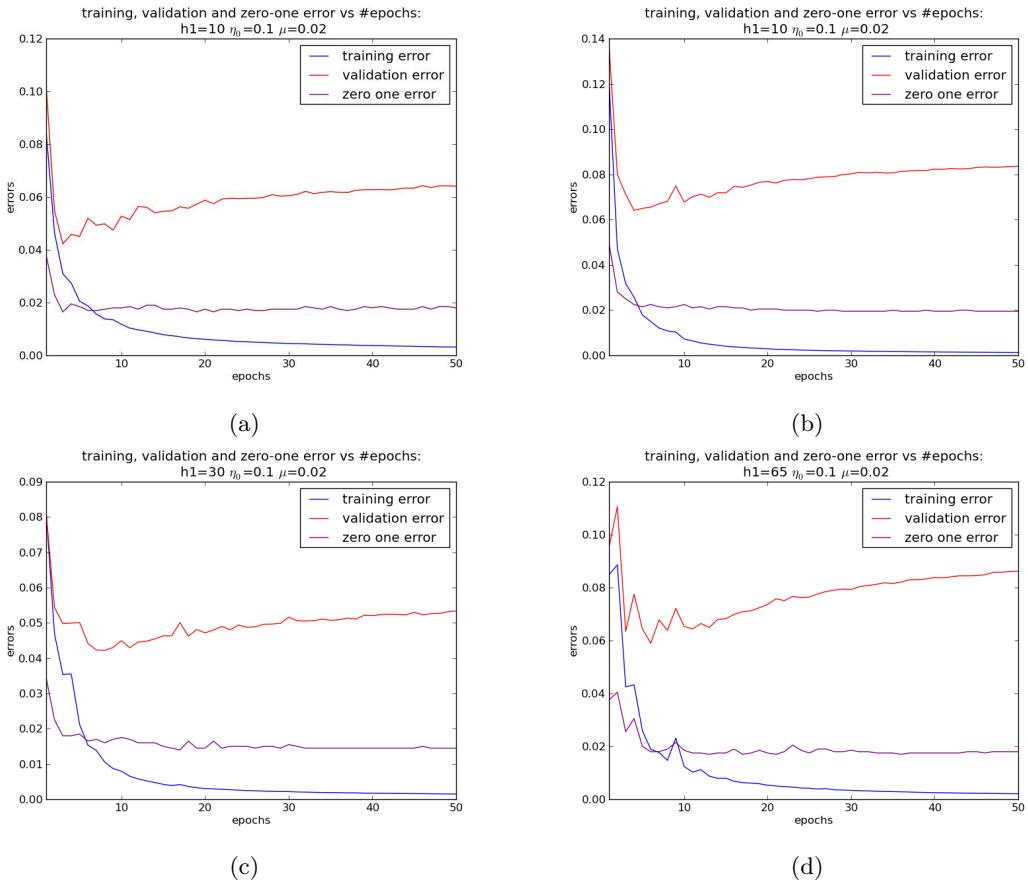


Figure 4: Example examples of overfitting for different numbers of hidden units

2.1.3 Effects of parameter choice on overfitting

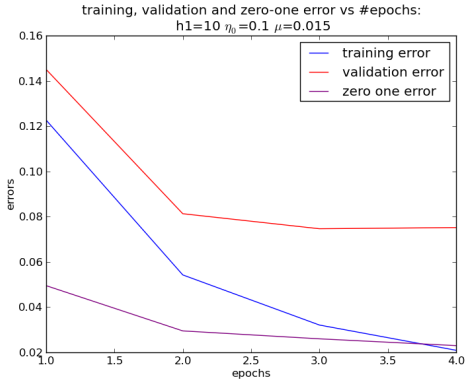
In this section we show examples of overfitting through changing the number of hidden units. We expected the validation error to increase more drastically over the number of epochs, while the training error decreases, for higher numbers of hidden units. However, we see in fig. 4a and fig. 4b that the result can already be very different for the same parameters. Additionally, for $h_1 = 30$ in fig. 4c the effect of overfitting does not appear to be more obvious than in the case of $h_1 = 10$. Finally we see that with 65 hidden units (fig. 4d) the validation error does increase more drastically over the 50 epochs indicating that with $h1$ around 65 and larger the now more complex network is more prone to overfitting.

2.2 Results

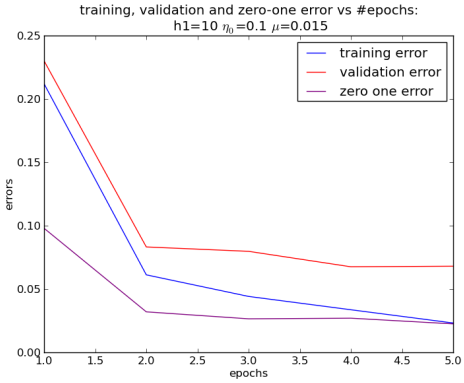
In this section we show the results obtained with the MLP classifier on both the 3/5 and 4/9 digit provided sets of the MNIST dataset and shortly discuss our choice of parameters for the subtasks.

2.2.1 Choice of parameters for subtasks and discussion of performance on test set

The best result we recorded on the 3/5 problem was a 0/1 error of 1.367% of a total of 1902 test points (26 in absolute numbers). To achieve this we used the following parameters: $h_1 = 10$, $\eta_0 = 0.1$, $\mu = 0.015$. We used the same parameters on the 4/9 problem where the best result on the testing set gave 2.26% misclassifications (45 in absolute numbers). We tried a number of different combinations of parameter values, however this combination consistently gave the best results for validation and testing for our setup.



(a) Sample graph for **mean** 0/1 error on 3/5 test-set recorded over 10 runs 0.01845 ± 0.00273

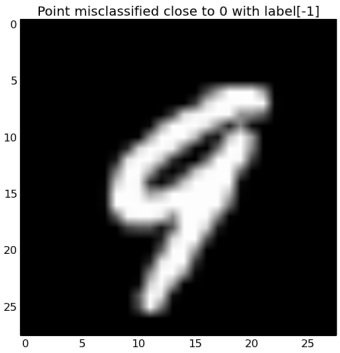


(b) Sample graph for **mean** 0/1 error on 4/9 test-set recorded over 10 runs 0.02395 ± 0.00140

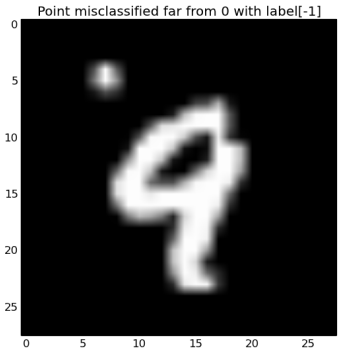
Figure 5: Sample graphs of a run of the final parametrization of the MLP on both subtasks annotated with the performance of test with mean and standard deviation

2.2.2 Misclassified patterns

Fig. 6a shows a misclassified pattern where $t_i a^{(2)}(x_i) < 0$ is close to 0. Its actual label is -1, meaning that the data pattern represents a 9. The fact that is was misclassified by little means that the model finds it slightly more similar to a 4 than to its actual value. Fig. 6b shows a misclassified point where



(a) Binary image of a 9 digit just about misclassified as a 4



(b) Binary image of a 9 digit **clearly** mistaken to be a 4 by the classifier

Figure 6: Examples of misclassified patterns for the 4-9 problem

$t_i a^{(2)}(x_i) < 0$ is far from 0. Its actual label is again -1, yet the fact that $a^{(2)}(x_i)a$ was greater than zero means the classifier sees it as a clear 4, assigning it the label 1. That means that in a feature space created from the data patterns the pattern would be an outlier of the "9 class" located in close proximity to patterns of the "4 class".

3 Support Vector Machine

3.1 Methods

3.1.1 Treatment of Data

Splitting For the implementation of the SMO algorithm the training data had not to be split into fixed training and validation sets. Instead the imperative was to implement 10-fold crossvalidation for parameter selection of τ and C. In order to achieve this task and save time we used the KFold() method of the python scikit library. This method thus splits the dataset into 10 consecutive folds without shuffling. Each fold is then used once as a validation set while the k-1 remaining folds form the training set.

Preprocessing Regarding preprocessing we applied the same normalization as with the Multilayer Perceptron.

3.1.2 SVM setup

To determine the optimal parametrization of the SVM we ran the SMO algorithm with 10-fold cross-validation for the following range of values of C and τ:

τ	0.002	0.004	0.008	0.016	0.032	0.064	0.128	0.256	
C	1	2	4	8	16	32	64	128	256

3.2 Results

The resulting scores from the cross.validation are depicted in fig. 7.

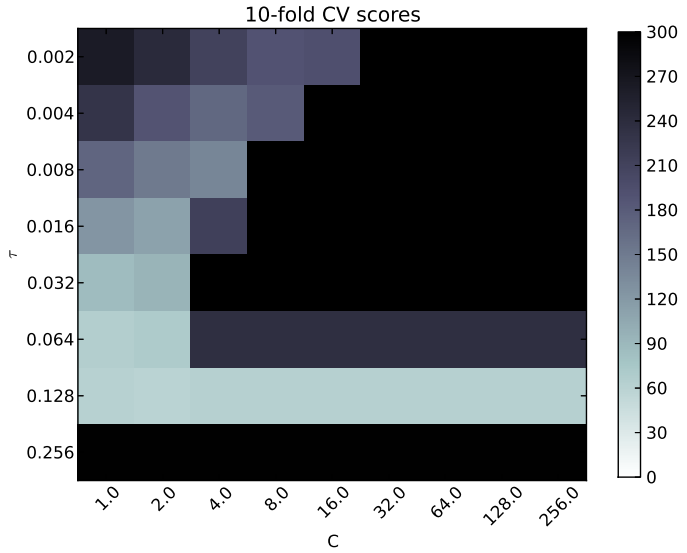


Figure 7: Summed up misclassification scores for range of parameters C and τ depicted after 10-fold crossvalidation. The smallest total score 59 was recorded for $\tau = .128$ and $C = 2$. The range of values is displayed up to a maximum score of 300 which represents 5% misclassifications on the validation data over the 10 folds.

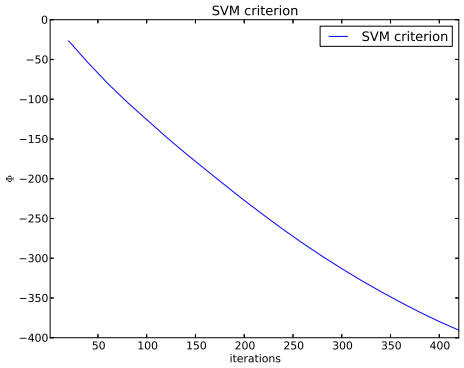
3.3 SVM criterion and Convergence criterion

Through running the SMO with the two parameters found through cross-validation on the whole training set, we obtained the graphs for the SVM criterion Φas a function of SMO iterations and the Convergence criterion $b_{up} - b_{low}$ based on KKT condition violations as shown in fig. 8. Subfig. 8a clearly shows how Φ is minimized according to the support vector coefficients α. The logarithmic scaling in subfig. 8b shows us that the convergence criterion decreases near-exponentially through the course of the optimization.

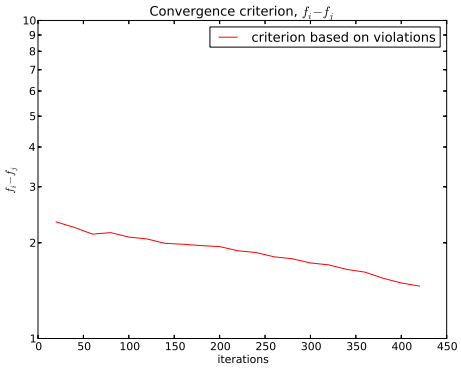
SVM Performance Training with the SMO as discussed above results in a 0/1 training error of 0.0502% and a classification error of 0.904%.

3.3.1 Performance comparison of SVM and MLP

In this section we make a comparison of the final setup of the Multilayer Perceptron and Support Vector Machine on the 4-9 classification problem. In fig. 9 we display the training and testing zero/one errors for the final parametrizations of the classification algorithms. One can see that SVM performs significantly better than the MLP, making less than 1% wrong classifications. Both algorithms are of



(a) SVM criterion $\Phi = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j t_i t_j - \sum_i \alpha_i$ over number of iterations



(b) Convergence criterion $f_i - f_j$ plotted over number of iterations of respective most violated pair

Figure 8: SMO run on full training set with parameters found through cross-validation

course powerful. Finding the optimal parameters for gradient descent of MLP may be faster than cross-validation for the SVM. However, the "kernel trick" makes nearly unbeatable in ways of complexity and fast setup, whereas changes in the architecture of the MLP can lead to different derivations of the gradient on the error function according to the weights of the network.

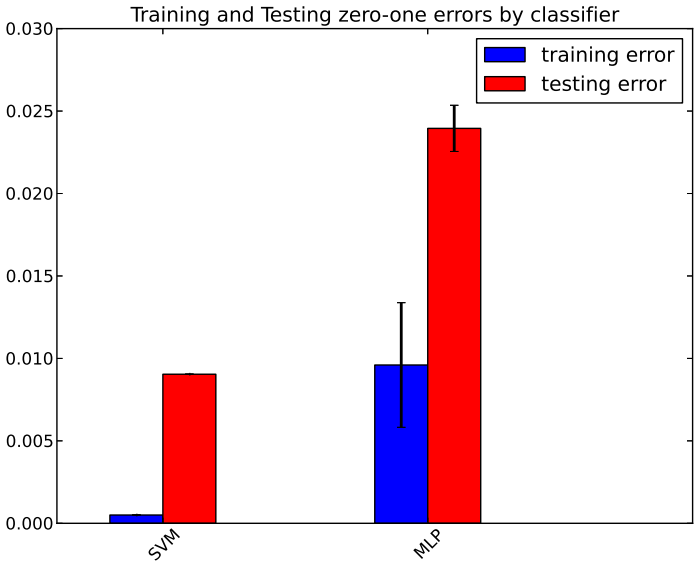


Figure 9: Bar plot of 0/1 error for training and testing comparing final parametrizations MLP and SVM (including the standard deviation for the MLP)