Pattern Classification and Machine Learning

Miniproject

Taha Elgraini (SCIPER ID: ...), Fabian Brix (SCIPER ID: 236334)

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 handwritten 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 to solve the problem with an SVM. Focus lay on a good understanding and clean implementation of the aforementioned algorithms as well as a sound evaluation of the performance of the implementation on problems posed with excerpts? the MNIST dataset.

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. **??**. 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, 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 paramters, 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.

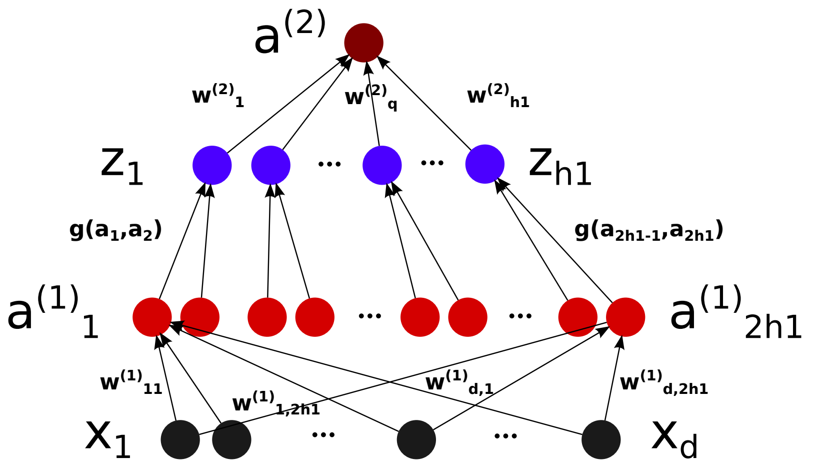
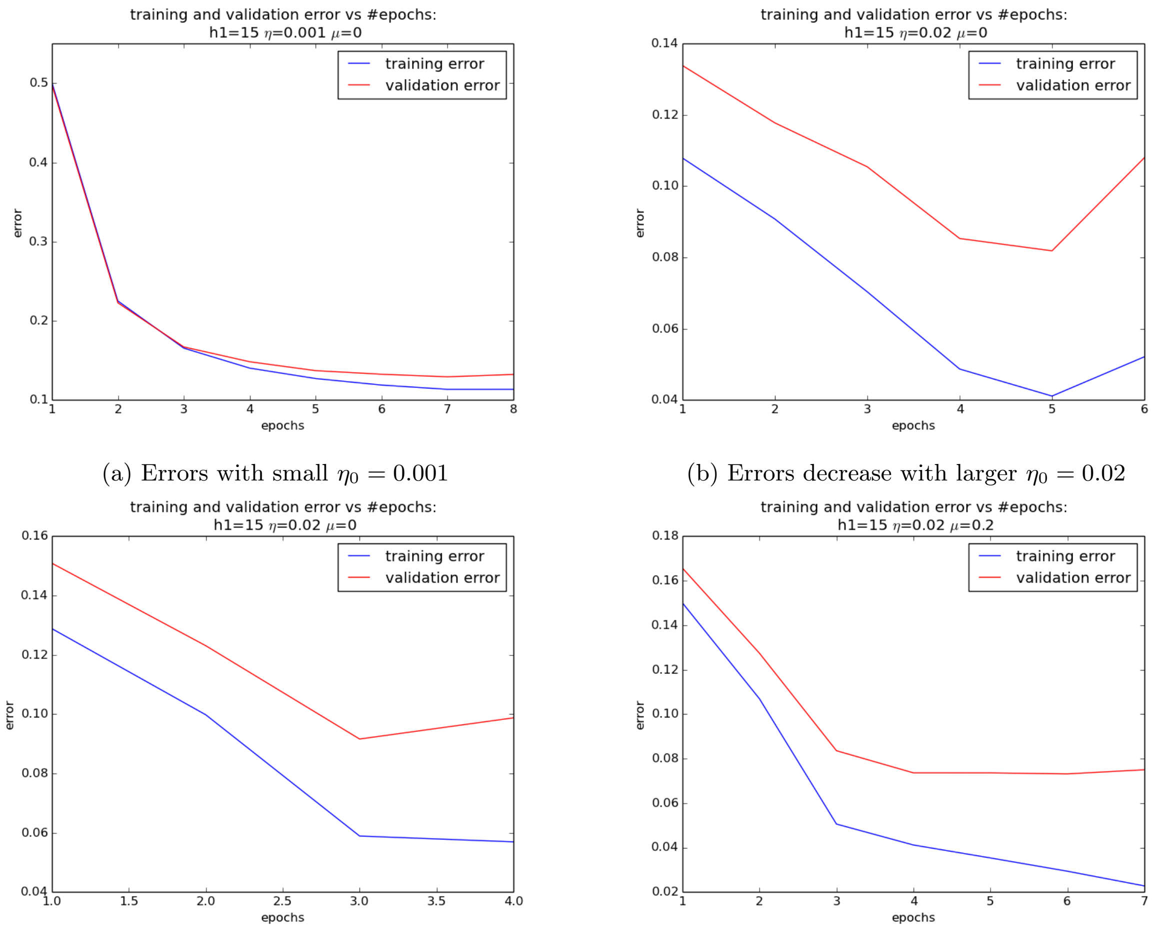


Figure 1: Schema of the MLP described in the project instructions

Effects of parameters on convergence speed and errors

- 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



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

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

(c) Decreasing η after every epoch

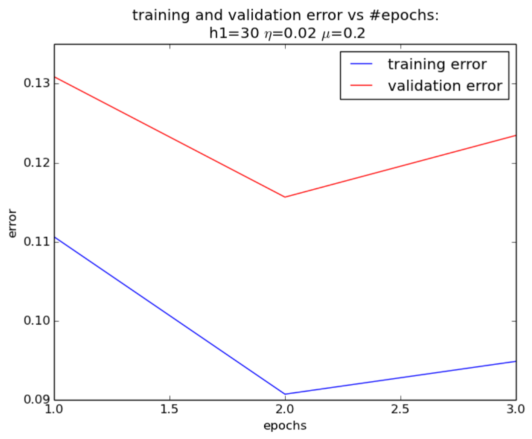
(d) Errors after introducing momentum term

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.

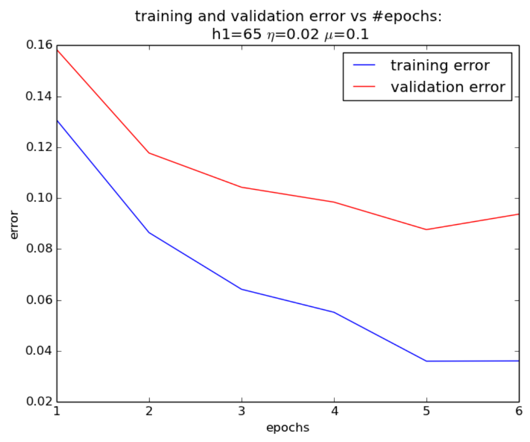
number of epochs, until an increase in the validation error occurs, to 6. After decreasing the learning rate in every epoch, the optimization converged already in the 4-th epoch. Introducing a momentum term $\mu = 0.2$

as in fig. 2d result in a faster decrease of error values.

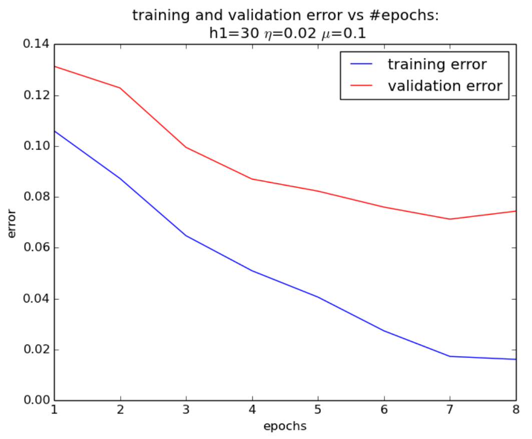
- Effects of number hidden units h_1



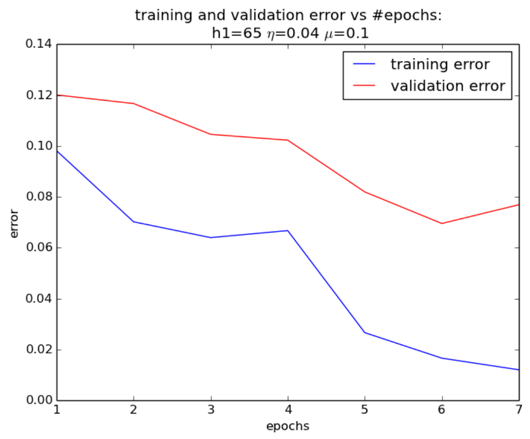
(a) example subcaption



(b) example subcaption



(c) example subcaption



(d) example subcaption

Figure 3: Example caption

2.1.3 Effects of parameter choice on overfitting

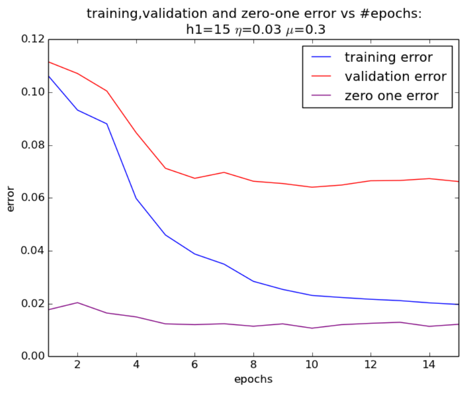
include TYPICAL example of overfitting by letting the validation error increase while the training error decreases for a few, say 3, rounds! show for which parameters training error decreases while validation increases yes, some epochs, not many... and you do this only for the report. in the learning algo, you would stop immediately okwe actually got better results when running once over 98what do you mean? over 98 is good! with early-stopping we nearly never get more than 97.75% so if you run 1 extra epoch after early stopping, it is better? yes, like two or threebecause it increases and goes down again that's a good finding! write it in your report! because there is no real danger of overfitting, because we use only 15 hidden inputsright? very good! in my case we used 64 units and we were overfitting if we would leave it for longer. We go more than 98 also with small dimentionality it generalizes more. It's not the same, but you can look at it learning in higher dimentionality and throwing away the least significant dimentions(which usually overfit) but it's not the same

2.1.4 Determining parameters for binary subtasks

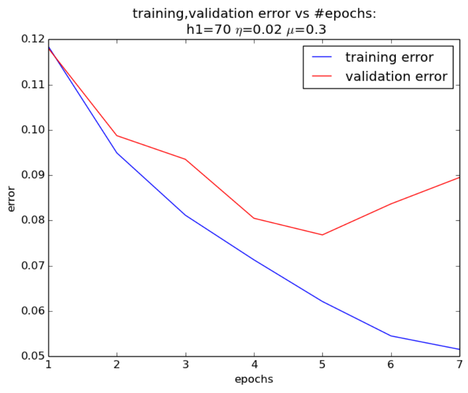
COMMENT ON FINDINGS **qualitatively**

3 Results

plots annotated by the result of the final classifier on the test set

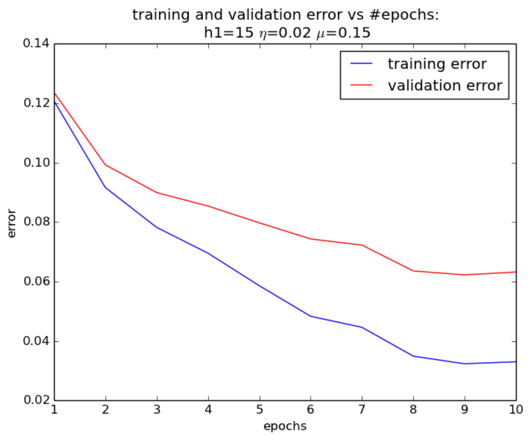


(a) example subcaption

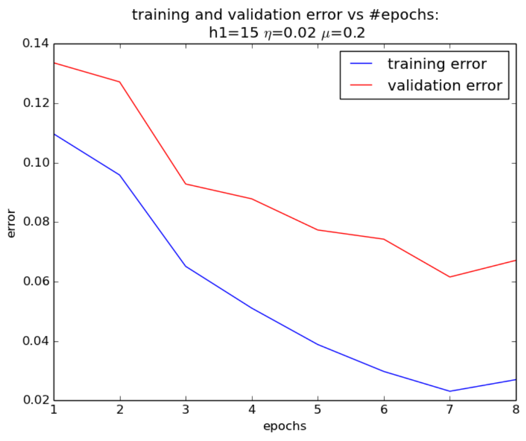


(b) example subcaption

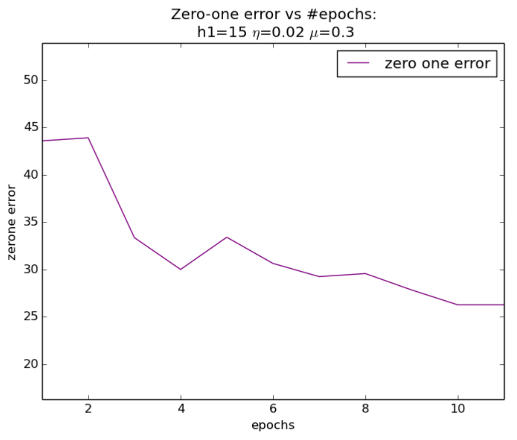
Figure 4: Example caption



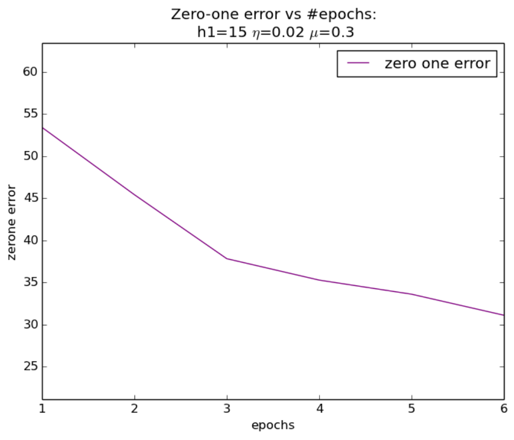
(a) example subcaption



(b) example subcaption



(c) example subcaption



(d) example subcaption

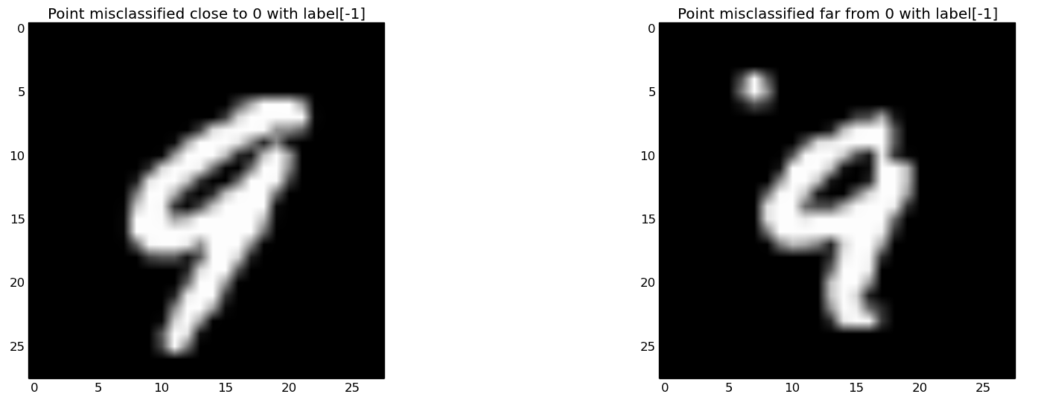
Figure 5: zerone

3.1 Discussion of performance on test set

3.2 Misclassified patterns

?? shows a misclassified point where ti*a2 is close to 0. The actual label is -1, meaning that the data point represents a 9. The fact that is was misclassified by little means that the model finds enough similarities with a 4. **??** shows a misclassified point where ti*a2 is far from 0. The actual label is -1, the data point if for a 9. yet, the fact that a2 was greater than zero means the classifier sees it as a clear 4. The point is actually an outlier from the

”4 class” to the ”9 class”.


Point misclassified close to 0 with label[-1]
Point misclassified far from 0 with label[-1]
0
5
10
15
20
25

(a) 9 just about misclassified as a 4 (b) 9 clearly mistaken as a 4 by classifier

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

4 Support Vector Machine

4.1 Methods

4.1.1 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.

4.1.2 SVM setup

To determine the optimal parametrization of the SVM we ran the SMO algorithm with 10-fold crossvalidation 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	0.512	
C	1	2	4	8	16	32	64	128	256	512

4.2 Results

Resulting scores from the crossvalidation

4.3 SVM criterion and Convergence criterion

Plot the SVM criterion as function of SMddO iterations (only every 20 SMO steps). Additionally, plot the value of the convergence criterion, based on KKT condition violations (see additional note). For the latter plot, the vertical axis should have a logarithmic scale.

5 Performance comparison of SVM and MLP

Comparison of the final setup of the Multilayer Perceptron and Support Vector Machine. In fig. .. we displayed the training and testing zero-one errors for the final parametrizations as required. One can see that SVM performs better overall...

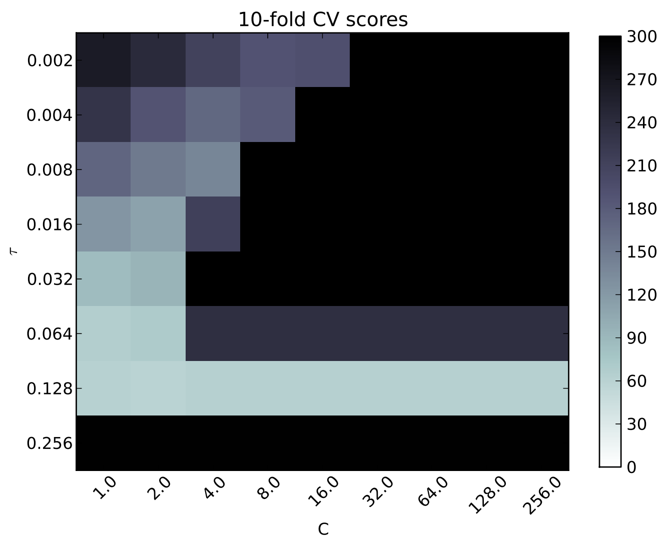
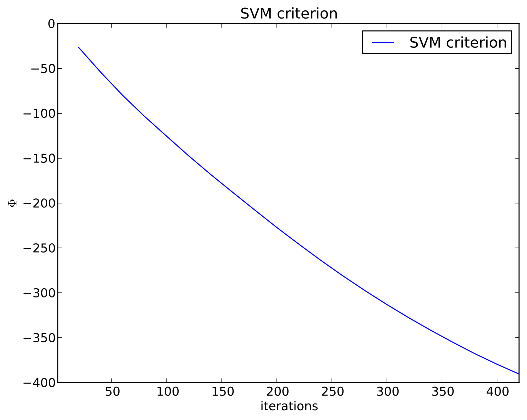
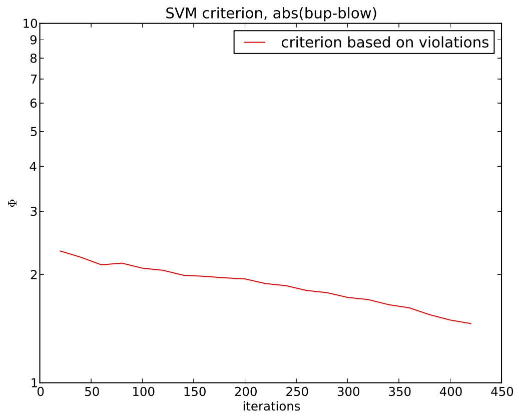


Figure 7: Summed up misclassification scores for range of parameters C and τ depicted after 10-fold crossvalidation. The smallest total score was recorded for $\tau = .128$ and $C = 2$. The range of values is displayed up to a maximum score of 300 which represents 5% misclassifications on the validation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



(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: Example caption

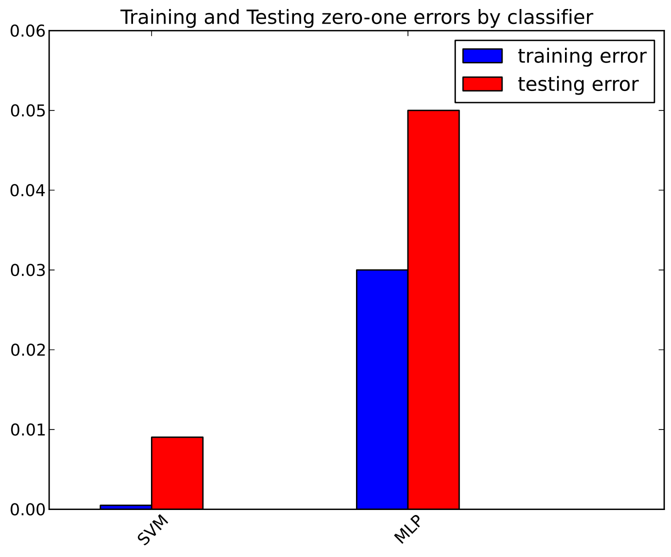


Figure 9: bar plot of zero-one error for training and testing comparing MLP (h1=.., eta=, mu=)and SVM (tau=.. and C=..)