# Assignment 3: Neural Networks and Support Vector Machines

Christian Igel and Kim Steenstrup Pedersen

March 2012

The goal of this assignment is to get familiar with advanced non-linear supervised learning methods.

You have to pass this and the following mandatory assignments in order to be eligible for the exam of this course. There are in total 3 mandatory pass/fail assignments on this course, which can be solved individually or in groups of no more than 3 participants. The course will end with a larger exam assignment, which must be solved individually and is graded (7- point scale).

The deadline for this assignment is **Tuesday 20/3 2012**. You must submit your solution electronically via the Absalon homepage. Go to the assignments list and choose this assignment and upload your solution prior to the deadline. If you choose to work in groups on this assignment you should only upload one solution, but remember to include the names of all participants both in the solution as well as in Absalon when you submit the solution. If you do not pass the assignment (after having made a serious attempt), you will get a second chance of submitting a new solution. The deadline for the resubmission is one week after we have sent the notification that you did not pass.

A solution consists of:

- Your solution code (Matlab / R / Python / C++ source code) with comments about the major steps involved in each Question (see below).
- Your code should be structured such that there is one main file that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions / classes.
- Your code should also include a README text file describing how to compile and run your program, as well as list of all relevant libraries needed for compiling or using your code. If we cannot make your code run we will consider your submission incomplete and you may be asked to resubmit.

- A PDF file with notes detailing your answers to the non-programming questions, which may include graphs and tables if needed (**Max 10 pages** text including figures and tables). Do NOT include your source code in this PDF file.

# 1 Neural Networks

In this part of the assignment, we consider standard feedforward neural networks (also called multi-layer perceptrons) with a single hidden layer.

In the experiments, we use artificial toy data stored in the file `sincTrain50.dt`. The data has been generated from a sinc : $\mathbb{R} \to \mathbb{R}$ function

$$\operatorname{sinc}(x) = \frac{\sin(x)}{x} \tag{1}$$

with additive normally distributed noise. This is a frequently used benchmark function for regression tasks.

## 1.1 Neural network implementation

Implement a multi-layer neural network with a linear output neuron and a single hidden layer with non-linear neurons. All neurons should have bias (offset) parameters.

For the hidden neurons, use the non-linearity (transfer function, activation function)

$$\sigma(u) = \frac{u}{1+|u|} \quad . \tag{2}$$

Prove that its derivative is

$$\sigma'(u) = \frac{1}{(1+|u|)^2} \quad . \tag{3}$$

Consider the mean-squared error as loss/error function $E$. Implement backpropagation to compute the gradient of the error with respect to the network parameters (check the slides of the "Neural Networks" lecture and sections 5.1– 5.2.1, 5.2.3 –5.3.1 in [1]; the network shall have a structure similar to the network shown in Figure 5.1 on page 228 of [1]; going through the example 5.3.2 in [1] is helpful, remember to replace the tanh activation function by the activation function (2)).

Compute gradients of the network using some arbitrary sample data. For instance, you could use parts of the sinc data. To verify your implementation, calculate the numerically estimated partial derivatives of each network parameter $\theta_i$ by computing

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} \approx \frac{E(\boldsymbol{\theta} + \epsilon \boldsymbol{e}_i) - E(\boldsymbol{\theta})}{\epsilon} \tag{4}$$

for small positive $\epsilon \ll 1$. Here, the vector $\boldsymbol{\theta}$ is composed of all neural network parameters (weights, bias parameters) and $\boldsymbol{e}_i$ denotes a vector of all zeros except for the $i$th component that is 1. Compare the numerically estimated gradients with the analytical gradients computed using backpropagation. These should be very close (i.e., differ less than, say, $10^{-8}$) given a careful adjustment of $\epsilon$. See section 5.3.3 in [1] for a discussion of using finite differences instead of backpropagation.

*Deliverables:* source code of neural network with a single hidden layer including backpropagation to compute partial derivatives; verification of gradient computation using numerically estimated gradients; derivation of derivative of the transfer function

## 1.2 Neural network training

The goal of this exercise is to gather experience with gradient-based optimization of models and to understand the influence of the number of hidden units in neural networks and how early-stopping can prevent overfitting.

For all experiments in this part of the assignment, use the sample data in `sincTrain50.dt`.

Do not produce a single plot for every function you are supposed to visualize. Combine results in the plots in a reasonable, instructive way.

Apply gradient-based (batch) training to your neural network model. For the exercise, it is sufficient to consider standard steepest descent. In practice, more advanced gradient based optimization algorithms are advisable for batch training (e.g., RProp [3], see also the slides of the "Neural Networks" lecture).

Train neural networks with 2 and 20 hidden neurons using all the data in `sincTrain50.dt`. Use batch learning until the error on the training data stops decreasing significantly (make sure that you watch it long enough). Feel free to play around with the number of hidden neurons and different learning rates. It is not part of the assignment, but you are encouraged to consider other datasets and to explore the benefits of shortcut connections.

Plot both the function (1) and the output of your trained neural networks over the interval $[-10, 10]$ (e.g., by sampling the functions at the points -10, -9.95, -9.9, -9.85,…,9.95, 10).

What happens for very small learning rates? What happens for very large learning rates? Plot the mean-squared error on the training set over the course of learning. That is, generate a plot with the learning epoch on the x-axis and the error on the y-axis. Use a logarithmic scale on the y-axis. Briefly discuss the results. Comment on overfitting and how early-stopping can be used to prevent overfitting (see section 5.5.2 in [1] and the slides of the "Neural Networks" lecture).

*Deliverables:* plots of error trajectories and final solutions of neural networks with 2 and 20 neurons trained on the full dataset using steepest descent with different learning rates, respectively; brief discussion of the plots; discussion of overfitting and early-stopping in the context of the experiments

# 2 Support Vector Machines

In this part of the assignment, you should get familiar with support vector machines (SVMs). Therefore, you need an SVM implementation. You can implement it on your own (see chapter 4.3 in [2] for algorithms to solve the SVM optimization problem), but you are welcome to use existing SVM software.

We recommend using LIBSVM, which can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. Interfaces to LIBSVM for many programming languages exist including Matlab and Python. For R, the package KERNLAB is available. If you are comfortable with C++, you are encouraged to use the SVM implementation within the SHARK machine learning library. You can either download the most recent version of the library from `http://image.diku.dk/shark` or get the latest snapshot using: `svn co https://shark-project.svn.sourceforge.net/svnroot/shark-project/branches/SharkReworkNewInterface/Shark` (this is recommended). *Note that some implementations consider $C/\ell$ instead of $C$ in the SVM objective function, where $C$ denotes the regularization parameter.*

For this exercise, use Gaussian kernels of the form

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{z}\|^2) \ . \tag{5}$$

Here $\gamma > 0$ is a bandwidth parameter that has to be chosen in the model selection process. Note that instead of $\gamma$ often the parameter $\sigma = \sqrt{1/(2\gamma)}$ is considered.

We consider the artificial KNOLL problem you know from the previous assignment. In this section, repeat all exercises independently for the training datasets `knollC-train100.dt`, `knollC-train200.dt`, as well as `knollC-train400.dt`, which are drawn from the same distribution but differ in size. This should give you some idea about the scaling of SVMs w.r.t. the number of training patterns. Present the results in tables with a row for each dataset.

## 2.1 Model selection using grid-search

The performance of your SVM classifier depends on the choice of the regularization parameter $C$ and the kernel parameters (here $\gamma$). Adapting these *hyperparameters* is referred to as SVM *model selection.*

Use grid-search to determine appropriate SVM hyperparameters $\gamma$ and $C$. Look at all combinations of

$$C \in \{0.1, 1, 10, 100, 1000, 10000\}$$

and $\gamma \in \{g_1, \ldots, g_7\}$, where you have to choose proper grid points for $\gamma$ yourself. For each pair, estimate the performance of the SVM using 5-fold cross validation (see section 1.3 in [1]). Pick the hyperparameter pair with the lowest average 0-1 loss (classification error) and train it using the complete training dataset. Report the values for $C$ and $\gamma$ you found in the model selection process, the 0-1 loss on the training data, as well as the 0-1 loss on the test data `knollC-test.dt`.

*Deliverables:* description of software used; a short description of how you proceeded (e.g., did the cross-validation); table with results for the three datasets including training and test errors as well as the best hyperparameter configuration

## 2.2 Inspecting the kernel expansion

In this exercise, we will inspect the trained SVM models. If not stated otherwise, use the hyperparameters you found in the previous section for training the SVMs.

### 2.2.1 Visualizing the SVM solution

First, consider the SVM model you got after training on the dataset `knollC-train200.dt`. Make a 2D plot of these data. In this plot, highlight by different colors the training data points that ended up as free support vectors and as bounded support vectors. A support vector $i$ is bounded if the corresponding coefficient in the kernel expansion (usually denoted by $\alpha_i$) has an absolute value of $C$.

*Deliverables:* plot of the data with visualization of the SVM model

### 2.2.2 Effect of the regularization parameter

Now let us study the effect of the regularization parameter $C$. To this end, retrain your SVM on `knollC-train200.dt` using values of $C$ that are 100 times larger and 100 times smaller, respectively, than the outcome of the model selection procedure. How does the SVM model change?

*Deliverables:* short discussion of how the SVM model changes depending on the choice of $C$

### 2.2.3 Scaling behavior

In this exercise, we consider the scaling with respect to the number of training examples.

Look at the SVM models you got after training on the datasets `knollC-train100.dt`, `knollC-train200.dt`, and `knollC-train400.dt` having 100, 200, and 400 data points, respectively. How many free and bounded support vectors have the solutions? How many training patterns violate the target margin?

Now we take a theoretical point of view. Given a distribution with a non-zero Bayes risk. How do you expect the number of support vectors to scale with the number of training examples in theory? Why? What implication does this have for large scale applications?

If the number of training data points increases, how should this affect the magnitude of $C$?

*Deliverables:* table showing the numbers of bounded and free support vectors for the three datasets; brief theoretical discussion of the scaling behavior of SVMs w.r.t. the number of training patterns

## References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[2] C. Igel. Machine learning: Kernel-based methods. Available from Absalon homepage, 2011.

[3] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons – From backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994.