Assignment 3: Neural Networks and Support Vector Machines

Christian Igel and Kim Steenstrup Pedersen

March 2011

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, March 22 at 23:55. 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).
- Notes detailing your answer including graphs and tables (including proper axes labels, legends, and captions) if needed (max. 10 pages of text).

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 $\mathrm{sinc} : \mathbb{R} \to \mathbb{R}$ function

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

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)

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

Proof that its derivative is

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

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 θ_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 all neural network parameters (weights, bias parameters) and $\boldsymbol{e}_i$ denotes a vector of all zeros except for the ith 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 ϵ. 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.

Feel free to play around with the number of hidden neurons and learning rates. Although not part of the assignment, you are encouraged to consider other datasets and to explore the benefits of shortcut connections.

1.2.1 Empirical risk minimization

Apply gradient-based (batch) training to your neural network model. Both standard steepest descent as well as resilient backpropagation (Rprop [4], see also the slides of the "Neural Networks" lecture) should be implemented. A *sketch* of the basic Rprop method is given in Algorithm 1. Here the superscripts indicate the iteration, E denotes the error function, and all parameters w_{ij} are combined in the vector $\boldsymbol{w}$ (i.e., $\boldsymbol{w}$ corresponds to $\boldsymbol{\theta}$ as used in equation (4)).

Algorithm 1: sketch of Rprop algorithm

initialize $\boldsymbol{w}^{(0)}$ // parameter vector

$\forall i, j : \Delta_{ij} > 0$, $g_{ij}^{(0)} = 0$ // step sizes and gradients

$\eta^+ = 1.2$; $\eta^- = 0.5$ // step size increase/decrease factors

$t \leftarrow 1$ // iteration counter

while *stopping criterion not met* **do**

  $g_{ij}^{(t)} = \partial E(\boldsymbol{w}^{(t)}) / \partial w_{ij}^{(t)}$ // store derivatives

  foreach w_{ij} **do** // loop over parameters

    // update step size depending on change of sign of partial derivative

    if $g_{ij}^{(t-1)} \cdot g_{ij}^{(t)} > 0$ **then** $\Delta_{ij}^{(t)} \leftarrow \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$

    else if $g_{ij}^{(t-1)} \cdot g_{ij}^{(t)} < 0$ **then** $\Delta_{ij}^{(t)} \leftarrow \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$

    // parameter update; $\text{sign}(x)$ is 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$

    $w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} - \text{sign}\left(g_{ij}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

  $t \leftarrow t + 1$

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 changing significantly, say, for

50.000 epochs (i.e., iterations of the gradient-based optimization algorithm).

Plot both the function (1) as well as 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).

First use steepest descent. What happens for very small learning rates? What happens for very large learning rates? Then use Rprop for training. How do the results compare to steepest descent? 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.

Deliverables: plots of error trajectories and final solutions of neural networks with 2 and 20 neurons trained on the full dataset using Rprop and steepest descent, respectively; brief discussion of the plots

1.2.2 Early-stopping

Now we will consider early-stopping (see section 5.5.2 in [1] and the slides of the "Neural Networks" lecture).

Split the data in `sincTrain50.dt` into two sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{stop}}$, one for training and one for early-stopping. Use the first 40 points for training and the last 10 points for $\mathcal{D}_{\text{stop}}$.

Train the neural networks using $\mathcal{D}_{\text{train}}$ until the error on $\mathcal{D}_{\text{train}}$ stops changing significantly, say, for 50.000 epochs. In every epoch, monitor the mean-squared on $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{stop}}$. Store the network achieving the smallest error on $\mathcal{D}_{\text{stop}}$. This network is the model found by early-stopping.

Plot the mean-squared error on the reduced training $\mathcal{D}_{\text{train}}$ set as well as on the validation dataset $\mathcal{D}_{\text{stop}}$ 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 as above. Briefly discuss the results.

Plot both the function (1) as well as the output of your trained neural networks over the interval $[-10, 10]$. Show the outputs of the networks at the end of training and the outputs of the networks that gave the smalles error on $\mathcal{D}_{\text{stop}}$. That is, compare the models found with and without early-stopping. Briefly discuss the results.

Deliverables: plots of error trajectories on D_{train} and $\mathcal{D}_{\text{stop}}$ and final solutions with and without early-stopping of neural networks with 2 and 20 neurons trained on the reduced data set using Rprop and steepest descent; brief discussion of the plots

2 Support Vector Machines

In this part of the assignment, you should get familiar with support vector machines (SVMs). Therefore, you need a 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 an 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. 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://shark-project.sourceforge.net` or get the latest snapshot using: `svn co https://shark-project.svn.sourceforge.net/svnroot/shark-project/Shark .`

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 γ often the parameter $\sigma = \sqrt{1/(2\gamma)}$ is considered.

We reconsider the artificial KNOLL problem from the previous assignment. In this section, do all exercises independently using the training datasets `knollC-train100.dt`, `knollC-train200.dt`, as well as `knollC-train400.dt`, which simply 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 and Jaakkola's heuristic

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

Jaakkola's heuristic provides a reasonable initial guess for the bandwidth parameter σ or γ of a Gaussian kernel [3]. To estimate a good value for σ, consider all pairs consisting of an training input vector from the positive class and a training input vector from the negative class. Compute the difference in input space between all pairs. The median of these distances can be used as a measure of scale and therefore as a guess for σ. More formally, compute

$$G = \{\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \mid (\boldsymbol{x}_i, y_i), (\boldsymbol{x}_j, y_j) \in S \wedge y_i \neq y_j\} \tag{6}$$

based on your training data S. Then set σ_{Jaakkola} equal to the median of the values in G:

$$\sigma_{\text{Jaakkola}} = \text{median}(G) \tag{7}$$

Compute the bandwidth parameter γ_{Jaakkola} from σ_{Jaakkola} using the identity given above.

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

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

and

$$\gamma \in \left\{ \gamma_{\text{Jaakkola}} \cdot 2^i \,|\, i \in \{-3, -1, 0, 1, 3\} \right\} \ .$$

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 γ you found in the models 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: table with results for the three datasets

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 almost surely bounded support vectors.

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 may surely free and (almost surely) 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 effect the magnitude of C and σ?

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

2.3 Distances in feature space

Given a kernel k on X. Let k be normalized. That is, let $k(z, z) = 1$ for all $z \in X$. Derive an equation for efficiently computing the distance of the images of the two elements $x, z \in X$ in the kernel-induced feature space (using the kernel-induced metric).

Deliverables: derivation of a short formular for computing the distance in feature space induced by a normalized kernel

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] T. Jaakkola, M. Diekhaus, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In T. Lengauer, R. Schneider, P. Bork, D. Brutlad, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158. AAAI Press, 1999.

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