Statistical Methods for Machine Learning 2015, Assignment III:

Assignment 3: Neural Networks and Support Vector Machines

Christian Igel and Kim Steenstrup Pedersen
Department of Computer Science
University of Copenhagen

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

You have to pass this and the other 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 written exam assignment which must be solved individually and is graded (7-point scale).

The deadline for this assignment is **March 17, 2015**. You must submit your solution electronically via the Absalon home page. 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, having made a *serious* attempt, you may get a second chance of submitting a new solution.

A solution consists of:

- Your solution source code (Matlab / R / Python scripts or C / C++ / Java 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, etc.
- 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 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.

III.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 files `sincTrain25.dt` and `sincValidate10.dt` . The data has been generated from a sinc : $\mathbb{R} \to \mathbb{R}$ function

$$\operatorname{sinc}(x) = \frac{\sin(x)}{x} \tag{III.1.1}$$

with additive normally distributed noise. This is a frequently used benchmark function for regression tasks. Here we use this artificial example because it can be easily visualized and therefore helps you to find possible mistakes in your implementation.

III.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)

$$h(a) = \frac{a}{1+|a|} \ . \tag{III.1.2}$$

Prove that its derivative is

$$h'(a) = \frac{1}{(1+|a|)^2} \ . \tag{III.1.3}$$

(Hint: One way to go is to consider the cases $a < 0$ and $a > 0$ separately and then to discuss what happens at $a = 0$.)

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 (III.1.2)).

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 $[\boldsymbol{w}]_i$ by computing

$$\frac{\partial E(\boldsymbol{w})}{\partial [\boldsymbol{w}]_i} \approx \frac{E(\boldsymbol{w} + \epsilon \boldsymbol{e}_i) - E(\boldsymbol{w})}{\epsilon} \tag{III.1.4}$$

for small positive $\epsilon \ll 1$. Here, the vector $\boldsymbol{w}$ is composed of all neural network parameters (weights w_{ij} and bias parameters w_{i0} for all i and j), the ith component of $\boldsymbol{w}$ is denoted by $[\boldsymbol{w}]_i$, 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

III.1.2 Neural network training

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

For all experiments in this part of the assignment, use the sample data in `sincTrain25.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 [9, 7], see also the slides of the "Neural Networks" lecture).

Train neural networks with 2 and 20 hidden neurons using all the data in `sincTrain25.dt`. Use batch learning until the error on the training data stops decreasing significantly (make sure that you watch it long enough). The validation data is used to monitor the training process, but not for gradient-based optimization of the network weights. 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.

What happens for very small learning rates? What happens for very large learning rates? Plot the mean-squared error on the training set as well as on the validation data set `sincValidate10.dt` 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.

Plot both the function (III.1.1) 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).

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 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

III.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 (e.g., see chapter 4.3 in [5] for algorithms or [2] to solve the SVM optimization problem), but you are welcome to use existing SVM software.

We recommend using LIBSVM [3], 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, package such as E1071 (recommended) and KERNLAB are available. If you are comfortable with C++, you are encouraged to use the SVM implementation within the SHARK machine learning library [6]. Get the latest snapshot from `http://image.diku.dk/shark`.[1]

For this exercise, use Gaussian kernels of the form

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{z}\|^2) \ . \tag{III.2.1}$$

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.

[1]Carefully study what the SVM implementation you use is doing under the hood. Some implementations consider C/ℓ instead of C in the SVM objective function, where C denotes the regularization parameter. The SVM from the Matlab Bioinformatics Toolbox may by default use different regularization parameters depending on the class and the class frequency.

Application: Diagnosing Parkinson's disease voice signals

We consider the medical application of diagnosing Parkinson's disease from a person's voice. We consider the data from [8], which can be obtained from the well-known UCI benchmark repository [4].

The data were collected from 31 people, 23 suffering from Parkinson's disease. Several voice recordings of these people were processed. Each line in the data files corresponds to one recording. The first 22 columns are features derived from the recording, including minimum, average and maximum vocal fundamental frequency, several measures of variation in fundamental frequency, several measures of variation in amplitude, two measures of ratio of noise to tonal components in the voice status, two nonlinear dynamical complexity measures, a measure called signal fractal scaling exponent, as well as nonlinear measures of fundamental frequency variation [8, 4]. The last column is the target label indicating whether the subject is healthy (0) or suffers from Parkinson's disease (1). The data were split in to a training and test set, `parkinsonsTrainStatML.dt` and `parkinsonsTestStatML.dt`, respectively.

III.2.1 Data normalization

As seen in a previous assignment, data normalization is an important preprocessing step. A basic normalization is to generate mean free, unit variance input data. You may reuse the code from your previous assignments.

Consider the training data in `parkinsonsTrainStatML.dt`. Compute the mean and the variance of every input feature (i.e., of every component of the input vector). Find the affine linear mapping $f_{\text{norm}} : \mathbb{R}^{22} \to \mathbb{R}^{22}$ that transforms the training data such that the mean and the variance of every feature in the transformed data are 0 and 1, respectively (verify by computing these values).

Use the function f_{norm} to also encode the test data. Compute the mean and the variance of every feature in the transformed test data.

The normalization is part of the model building process. Thus, you may only use the training data for determining f_{norm} (always remember that you are supposed to not know the test data).

Deliverables: Mean and variance of the training data; mean and variance of the transformed test data

III.2.2 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 γ). Adapting these *hyperpa-*

rameters is referred to as SVM *model selection*, as already discussed in a previous assignment.

Repeat the following process for the raw (i.e., not transformed) and for the normalized data.

Use grid-search to determine appropriate SVM hyperparameters γ and C. Look at all combinations of $C \in \{c_1, c_2, \ldots, c_7\}$ and $\gamma \in \{g_1, \ldots, g_7\}$, where you have to choose proper grid points for yourself. Consider values around $C = 10$ and $\gamma = 0.1$ of different orders of magnitude. It is common to vary the values on a logarithmic scale (e.g., $0.001, 0.01, 0.1, 1, 10, 100$). 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 an SVM with these hyperparameters using the complete training dataset. Only the training data must be used in the model selection process.

Report the values for C and γ 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. Briefly discuss the differences between using the raw and the normalized data.

Deliverables: Description of software used; a short description of how you proceeded (e.g., did the cross-validation); training and test errors as well as the best hyperparameter configuration for normalized and raw data; short discussion of the effect of normalization

III.2.3 Inspecting the kernel expansion

In this exercise, we will inspect the trained SVM models. Use the hyperparameters you found in the previous section, either for the normalized or the raw data.

III.2.3.1 Support vectors

Count the number of free and bounded support vectors. A support vector x_i is bounded if the corresponding coefficient in the kernel expansion (usually denoted by α_i) has an absolute value of C.

Let us consider the effect of the regularization parameter C. How do the numbers of bounded and free support vectors change if C is drastically increased and decreased? Why?

Deliverables: Numbers of bounded and free support vectors; short discussion of how the SVM model changes depending on the choice of C

III.2.3.2 Scaling behavior

Let us think about the scaling of SVM models with respect to the number of training examples. 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?

Deliverables: 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] L. Bottou and C.-J. Lin. Support vector machine solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 1–28. MIT Press, 2007.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions Intelligent Systems Technology*, 2(3):27:1–27:27, 2011.

[4] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

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

[6] C. Igel, T. Glasmachers, and V. Heidrich-Meisner. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[7] C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.

[8] M. Little, P. McSharry, E. Hunter, J. Spielman, and L. Ramig. Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. *IEEE Transactions on Biomedical Engineering*, 56(4):1015–1022, 2009.

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