Machine Learning 2019-2020

Home Assignment 6

Yevgeny Seldin **Christian Igel**
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **14 January 2020**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your source code in the PDF file.
- A .zip file with all your solution source code with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF.
- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in the speed grader. Zipped pdf submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) 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 include a README text file describing how to compile and run your program, as well as a list of all relevant libraries needed for compiling or using your code.
- Handwritten solutions will not be accepted, please use the provided latex template to write your report.

Neural Networks

In this 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

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

with additive normally distributed noise. This is a frequently used toy example for regression tasks. Here we use this artificial problem (instead of a more exciting real-world data set) because it can be easily visualized and therefore helps you to find mistakes in your implementation. Thereafter, you can apply you implementation to more intersting problems, for example starting with the data sets from previous assignments.

1 Neural network implementation

The first task is to implement a feed-forward neural network with a linear output neuron and a single hidden layer with non-linear neurons. The implementation should allow to vary the number of hidden neurons.

Why? In practice, one would in most cases use a machine learning library providing a neural network implementation, for example TensorFlow as introduced in the recommended course *Large-scale Data Analysis*. However, implementing a neural network is one way to gain a better understanding of what a neural network really does (and how neural network libraries work). Furthermore, if you do more advanced machine learning, you will have to implement more complex models yourself and need to know, for instance, how to check that the gradients are correct.

All neurons should have bias (offset) parameters. For the hidden neurons, use the non-linearity (transfer function, activation function)

$$h(a) = \frac{a}{1 + |a|} \tag{2}$$

with derivative

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

(Exercise not for submission: Check that the derivative is correct. To this end, consider the cases $a < 0$ and $a > 0$ separately and then 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.

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

$$\frac{\partial E(\boldsymbol{w})}{\partial [\boldsymbol{w}]_i} \approx \frac{E(\boldsymbol{w} + \epsilon \boldsymbol{e}_i) - E(\boldsymbol{w})}{\epsilon} \tag{4}$$

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.

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

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.

You should use your neural network code implemented for the first question of this assignment. *If you do not manage to get your neural network code running, you may use some software library. In this case – which should be avoided – describe in the report the number of network parameters and show in the report the lines of code you used to retrieve/compute the gradient magnitude.*

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).

Train a neural networks with 20 hidden neurons using all the data in `sincTrain25.dt`. Use batch learning. Now, how long should you train? Monitor

the training error (i.e., the error on the training data set) and the norm of the error gradient,

$$\|\nabla E(\boldsymbol{w})\| = \sqrt{\sum_i \left(\frac{\partial E(\boldsymbol{w})}{\partial [\boldsymbol{w}]_i}\right)^2} \ , \tag{5}$$

in every iteration. A good stopping condition can be if the training error is not decreasing anymore for a long time or the norm of the gradient falls below a certain threshold. Make sure that you watch the training long enough.

Additionally, you should compute the error on the validation data in every iteration. The validation data is used to monitor the training process, but not for gradient-based optimization of the network weights. Thus, for every iteration, you are supposed to measure the training error, the validation error, and the gradient norm.

The learning crucially depends on the leanring rate. Vary the learning rate η over several orders of magnitude, say, $\eta = 1, 0.1, 0.01, 0.001, \ldots$. What happens for very small learning rates? What happens for very large learning rates? Find a learning rate that is clearly too large, a learning rate that is clearly too small, and a learning rate η_{nice} you regard as OK.

Plot the mean-squared error on the training set as well as on the validation set `sincValidate10.dt` over the course of learning for each of the three learning rates. Generate plots with the learning epoch/iteration on the x-axis and the error on the y-axis. Use a logarithmic scale on the y-axis. Provide three plots – one for each learning rate – of the training and validation error. Additionally, visualize the corresponding gradient norms (either in three separate plots or you can add them to the error plots). Briefly discuss the plots in the report.

Now take the model you got after training with η_{nice} and visualize the learnt function. Plot both the function (1) and the output of your trained neural networks over the interval $[-15, 15]$ (e.g., by sampling the functions at the points -15, -14.95, -14.9, -14.85,. . . ,14.95, 15).

Comment on overfitting and how early-stopping can be used to prevent overfitting.

It is not part of the assignment, but you are encouraged to consider other network architecture (e.g., what happens when you use only two hidden neurons?) and datasets and to explore the benefits of shortcut connections.

Deliverables: Plots of error trajectories of neural networks neurons using steepest descent with three different learning rates; plots of gradient norms; plot of the final model; brief discussion of the plots; discussion of overfitting and early-stopping in the context of the experiments