Machine Learning 2020-2021

Home Assignment 7

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

The deadline for this assignment is **19 January 2021, 22:00**. 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. The programming language of the course is Python.
- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in 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.
- 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 (35 points)

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

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

(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 (35 points)

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

3 Early Stopping (30 points)

Early stopping is a widely used technique to avoid overfitting in models trained by iterative methods, such as gradient descent. In particular, it is used to avoid overfitting in training neural networks Goodfellow et al. (2016, Section 7.8)[1]. In this question we analyze several ways of implementing it. The technique sets aside a validation set S_{val}, which is used to monitor the improvement of the training process. Let $h_1, h_2, h_3, \ldots$ be a sequence of models obtained after $1, 2, 3, \ldots$ epochs of training a neural network (you do not have to know the details of the training procedure to answer the question). Let $\hat{L}(h_1, S_{\text{val}})$, $\hat{L}(h_2, S_{\text{val}})$, $\hat{L}(h_3, S_{\text{val}}), \ldots$ be the corresponding sequence of validation errors on the validation set S_{val}.

1. Let h_{t^*} be the neural network returned after training with early stopping. In which of the following cases is $\hat{L}(h_{t^*}, S_{\text{val}})$ an unbiased estimate of $L(h_{t^*})$ and in which cases is it not. Please, explain your answer.

   (a) Predefined stopping: the training procedure always stops after 100 epochs and always returns the last model $h_{t^*} = h_{100}$.

   (b) Non-adaptive stopping: the training procedure is executed for a fixed number of epochs T, and returns the model h_{t^*} with the lowest validation error observed during the training process, i.e., $t^* = \arg\min_{t \in \{1,\ldots,T\}} \hat{L}(h_t, S_{\text{val}})$.

   (c) Adaptive stopping: the training procedure stops when no improvement in $\hat{L}(h_t, S_{\text{val}})$ is observed for a significant number of epochs (this is the procedure proposed in Goodfellow et al. (2016, Algorithm 7.1) or `https://www.quora.com/How-does-one-employ-early-stopping-in-TensorFlow`). It then returns the best model observed ever during training.

2. Derive a high-probability bound (a bound that holds with probability at least $1 - \delta$) on $L(h_{t^*})$ in terms of $\hat{L}(h_{t^*}, S_{\text{val}})$, δ, and the size n of the validation set S_{val} for the three cases above. In the second case the bound may additionally depend on the total number of epochs T, while in the third case the bound may additionally depend on the index t^* of the epoch providing the optimal model. Please, solve the last case using the series $\sum_{t=1}^{\infty} \frac{1}{i(i+1)} = 1$.[2]

3. The adaptive approach suggests stopping when "no improvement in $\hat{L}(h_t, S_{\text{val}})$ is observed for a significant number of epochs". A natural way of redefining the stopping criterion once we have the generalization bound is to stop when "no improvement in the generalization bound is observed for a significant number of epochs". The adaptive approach does not limit the number of epochs in advance, but what is the maximal number of epochs $T_{\max}$, after which it makes no sense to continue training according to the

[1] `https://www.deeplearningbook.org/contents/regularization.html`

[2] We have $\sum_{i=1}^{\infty} \frac{1}{i(i+1)} = \sum_{i=1}^{\infty} \left(\frac{1}{i} - \frac{1}{i+1} \right) = 1$.

bound you derived in Point 2. Express $T_{\max}$ in terms of the number of validation samples n. It is sufficient to provide an order of magnitude of $T_{\max}$ in terms of n, you do not have to calculate the explicit constants.

4. How would your answer to the previous point change if you use the series $\sum_{i=1}^{\infty} \frac{1}{2^i} = 1$ for deriving the bound? (You should get that with these series you can run significantly less epochs in the adaptive approach compared to the series used in Point 2. Thus, unlike in the case of decision trees from an earlier assignment, here the choice of the series has a significant impact.)

5. In this question we compare the adaptive procedure with non-adaptive. Assume that the two procedures use the same initialization, so that the corresponding models at epoch t are identical, and assume that the adaptive procedure has reached $T_{\max}$ (but t^* may be smaller than $T_{\max}$). Show that the generalization bound for adaptive stopping in Point 2 is never much worse than the generalization bound for non-adaptive stopping, but in some cases the adaptive bound can be significantly lower.

   Guidance: Throughout the question we assume that the confidence parameter $\delta \leq \frac{1}{2}$. (In principle, it is possible to show a slightly weaker result also for $\delta > \frac{1}{2}$, but $\delta > \frac{1}{2}$ is such a low confidence that we ignore this case.)

   (a) First, assume that $T \leq T_{\max}$. Let t^* be the index of the epoch selected by the adaptive procedure and T^* be the index of the epoch selected by the non-adaptive procedure. Since the adaptive procedure has selected t^* we know that the adaptive bound for epoch t^* is lower than the adaptive bound for epoch T^*. We also know that $T^* \leq T$, where T is the number of epochs in the non-adaptive approach. Use this information and do some bounding to show that for any confidence parameter $\delta \leq \frac{1}{2}$, the adaptive bound can be at most a multiplicative factor of $\sqrt{2}$ larger than the non-adaptive bound.

   (b) Now consider the case $T > T_{\max}$. Show that in this case the adaptive bound is trivially 1 and the non-adaptive bound is at least $\frac{1}{\sqrt{2}}$. So in this case the adaptive bound also cannot exceed the non-adaptive bound by more than a multiplicative factor of $\sqrt{2}$.

   (c) You have shown that under the assumption that $\delta \leq \frac{1}{2}$ the adaptive bound never exceeds the non-adaptive bound by more than a multiplicative factor of $\sqrt{2}$. Now explain in which situations the adaptive bound can be significantly smaller than the non-adaptive bound. You should have two cases. In both cases you should have $T < T_{\max}$ and $\delta \leq \frac{1}{2}$.

Conclusion: depending on the data, the generalization bound for adaptive stopping can be significantly smaller than the generalization bound for non-adaptive stopping and at the same time it is guaranteed that it is never worse by more than a multiplicative factor of $\sqrt{2}$.

References

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.