Deep Learning Assignment I-A: Simple Regression

Felix Wiewel

February 10, 2021

1 Introduction

This programming exercise is the first of a series of exercises, which are intended as a supplement to the theoretical part of the Deep Learning course offered by the ISS. The goal is to introduce you to basic tasks and applications of methods you have encountered in the lecture. After completing the exercise you should be familiar with the basic ideas and one, possibly simple, way of solving the respective task. It is worth mentioning that most of the tasks can be solved in many different, not necessarily deep learning based, ways and the solution presented here is just one of them.

2 Regression

In this exercise we consider the problem of regression, where we are interested in modeling a functional dependence between different variables with typically noisy observations of input-output pairs. Mathematically such a dependence can be formulated as

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$ are the input and output observations, $f : \mathbb{R}^N \to \mathbb{R}^M$ is the function mapping inputs to outputs and $\boldsymbol{\epsilon} \in \mathbb{R}^M$ is a random vector, which models noise in our observations. Note that this assumes additive noise that only acts on the output and not on the input variable, which might not be true in all practical applications. For regression we are now interested in estimating the functional relationship f between the inputs and outputs. This can be done in many different ways, not just with neural networks, but for this exercise we focus on approximating this relationship with a neural network $g_{\boldsymbol{\theta}} : \mathbb{R}^N \to \mathbb{R}^M$ with parameter vector $\boldsymbol{\theta}$. The task is now to choose the parameters of the neural network in a way that results in a "good" approximation of f with $g_{\boldsymbol{\theta}}$.

In order to quantify how "good" our neural network can approximate f, we adopt a probabilistic view. For this we make the assumption that the noise $\boldsymbol{\epsilon}$ is

a random vector drawn from a known distribution, which enables us to derive a suitable cost function for training our neural network and also for quantifying a "good" approximation.

3 Mathematical formulation

If we assume that the noise $\boldsymbol{\epsilon}$ is drawn from a Gaussian distribution, e.g. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, we can use

$$\mathbf{y} = g_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon} \Rightarrow \mathbf{y} - g_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\epsilon} \tag{2}$$

to derive a log-likelihood of ϵ. Since the probability density function (pdf) of a generic multivariate normal distribution $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ is given by

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^D |\mathbf{C}|}} \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})^T}, \tag{3}$$

we get

$$\ln p(\boldsymbol{\epsilon}) = \ln\left(\frac{1}{\sqrt{(2\pi\sigma^2)^M}} \mathrm{e}^{-\frac{1}{2\sigma^2}\|\boldsymbol{\epsilon}\|_2^2}\right) = -\frac{1}{2\sigma^2}\|\boldsymbol{\epsilon}\|_2^2 - \frac{M}{2}\ln(2\pi\sigma^2). \tag{4}$$

Replacing $\boldsymbol{\epsilon}$ by $\mathbf{y} - g_{\boldsymbol{\theta}}(\mathbf{x})$ yields the log-likelihood for one particular input-output pair:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \ln p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}\|\mathbf{y} - \boldsymbol{g_\theta}(\mathbf{x})\|_2^2 - \frac{M}{2}\ln(2\pi\sigma^2) \tag{5}$$

This log-likelihood measures how likely the input-output pair is and we can use it to train our neural network. For this we maximize the expected log-likelihood over all input-output pairs under the assumption that the noise is independent and identically distributed (i.i.d.) over all input-output pairs. This corresponds to finding the parameters $\boldsymbol{\theta}^\star$ of our neural network, which maximize the log-likelihood for observing the corresponding input-output pairs. Mathematically the optimal parameters for our neural network are given by

$$\begin{aligned}\boldsymbol{\theta}^\star &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})\right] = \arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[-\|\mathbf{y} - \boldsymbol{g_\theta}(\mathbf{x})\|_2^2\right] \\ &\approx \arg\min_{\boldsymbol{\theta}} \frac{1}{N_D}\sum_{i=1}^{N_D} \|\mathbf{y_i} - \boldsymbol{g_\theta}(\mathbf{x_i})\|_2^2,\end{aligned} \tag{6}$$

where all terms, which are independent of $\boldsymbol{\theta}$, are ignored and the expectation operator is approximated by the mean over all N_D input-output pairs. In other words, we are maximizing the log-likelihood by minimizing the mean squared error (MSE) loss over all input-output pairs in our dataset, hence this approach is called Maximum Likelihood (ML) estimation. For solving this optimization

problem and obtaining the optimal network parameters, stochastic gradient descent (SGD) or one of it's many variants is typically used.

It is worth noting that choosing different distributions for the noise $\boldsymbol{\epsilon}$ leads to different log-likelihoods and therefore different cost functions for training the neural network. Another commonly used distribution for modeling the noise in regression tasks is the Laplace distribution. Deriving the log-likelihood and the corresponding cost function leads to the mean absolute error (MAE), which is given by the l_1-norm of the difference between observations and predictions of the neural network. This cost function is more robust against outliers since these have less influence on the MAE loss compared to the MSE because the square operation amplifies large errors.

4 Regularization

In this section we will explore the concept of regularization. As there is no theorem that can be used to determine the required size and structure of a neural network given a certain task, one has to find a suitable neural architecture by trial and error. This can result in choosing an architecture with a capacity that is higher than required for solving the given task and hence overfitting might occur. A common way to prevent large neural networks from overfitting is to employ some sort of regularization, e.g. weight norm penalty, dropout, early stopping and data augmentation. In this section we will focus on weight norm penalty as a regularization and derive a probabilistic interpretation for some of those.

We start by restating the conditional probability of the output $\mathbf{y}$ given the input $\mathbf{x}$ and the networks parameters $\boldsymbol{\theta}$ as

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}) = \frac{1}{\sqrt{(2\pi\sigma^2)^M}} \mathrm{e}^{-\frac{1}{2\sigma^2}\|\mathbf{y}-\boldsymbol{g}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2}. \tag{7}$$

If we have some prior knowledge about the parameters of the neural network, which is given by a pdf $p(\boldsymbol{\theta})$, we can use Bayes theorem to derive a posterior distribution

$$p(\boldsymbol{\theta}|\mathbf{x},\mathbf{y}) = \frac{p(\boldsymbol{\theta},\mathbf{y}|\mathbf{x})}{p(\mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y})}, \tag{8}$$

where $p(\boldsymbol{\theta})$ is the prior of $\boldsymbol{\theta}$. Similar to the derivation at the beginning of the exercise, we can use this posterior distribution to derive a cost function for training the neural network. In this case, however, we are not maximizing the log-likelihood but the posterior distribution over $\boldsymbol{\theta}$, hence this approach is called Maximum A Posteriori (MAP) estimation of the parameters. Mathematically we can formulate this as

$$\begin{aligned}\boldsymbol{\theta}^\star &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[p(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})\right] = \arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[\ln p(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})\right] \\ &= \arg\max_{\boldsymbol{\theta}} \ln p(\boldsymbol{\theta}) + \mathbb{E}\left[\ln p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})\right], \end{aligned} \tag{9}$$

where we used the fact that applying a strictly increasing function, e.g. $\ln$, does not change the position of the maximum of a cost function, ignored $p(\mathbf{y})$, since it is independent of the network parameters and dropped the expectation operator for $\ln p(\boldsymbol{\theta})$, since it is not depending on the random variable $\mathbf{x}$. When comparing the MAP estimate of the parameters with the ML estimate in Eq. 6, the only difference is the addition of $\ln p(\boldsymbol{\theta})$. This term describes a specific regularization, i.e. the weight norm penalty. Depending on the distribution over $\boldsymbol{\theta}$, it can have different forms. If we choose a standard normal distribution, i.e. $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we get

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \lambda \|\boldsymbol{\theta}\|_2^2 + \frac{1}{N_D} \sum_{i=1}^{N_D} \|\mathbf{y_i} - \boldsymbol{g_\theta}(\mathbf{x_i})\|_2^2, \tag{10}$$

where we have made the same simplifications as for the ML estimation and also introduced the parameter $\lambda = \sigma^2$, which is used to control the strength of the regularization. This form of regularization is commonly known as l_2-norm or weight decay regularization. Choosing a prior where the weights follow an i.i.d Laplacian distribution, i.e. $p(\boldsymbol{\theta}) = \prod_j \frac{1}{2} \mathrm{e}^{|\theta_j|}$, leads to

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \lambda \|\boldsymbol{\theta}\|_1 + \frac{1}{N_D} \sum_{i=1}^{N_D} \|\mathbf{y_i} - \boldsymbol{g_\theta}(\mathbf{x_i})\|_2^2, \tag{11}$$

where the strength of the regularization is again controlled by $\lambda = 2\sigma^2$. This type of regularization is known as l_1-norm and it has the property to induce sparsity in the parameters of the network.

5 Dataset

In this assignment a simple synthetic example is used in order to create a regression problem. For this, we consider the function

$$y = f(x) + 0.1 \cdot \epsilon = \sin(1 + x^2) + 0.1 \cdot \epsilon, \tag{12}$$

where $\epsilon \sim \mathcal{N}(0, 1)$ is normally distributed. In the provided code skeleton we train a neural network to learn an approximation of the function $f(x)$.

6 Tasks

In order to complete this assignment, please fill out the code skeleton provided with this task description and solve the following tasks.

1. Determine the total number of parameters and multiplications for the model `MyModel` in the code skeleton.

2. Determine the sizes of all hidden representations in the DNNs used in the code skeleton.

3. What activation function is used in these DNNs. Could other activation functions be used? Justify your answer.