Exercise Sheet 6

Due: 01.12.25 14h00

Christof Gehrig, Gerrit Gerhartz

General Regulations.

- Please hand in your solutions in groups of two (preferably from the same tutorial group).
- Your solutions to theoretical exercises can be either handwritten notes (scanned), or typeset using LaTeX. For scanned handwritten notes, please ensure they are legible and not blurry.
- For the practical exercises, always provide the (commented) code as well as the output, and don't forget to explain/interpret the latter.
- Please hand in a **single PDF** that includes both the exported notebook and your solutions to the theoretical exercises. Submit the PDF to the Übungsgruppenverwaltung once per group, making sure to include the names of both group members in the submission.
- You can find all the data in the GitHub Repository.

1 Forward- and reverse-mode differentiation

We explore the computation of derivatives on general acyclic computational graphs. Consider the following function:

$$y = \exp\left[\exp(x) + \exp(x)^2\right] + \sin\left[\exp(x) + \exp(x)^2\right]$$

(a) Write down the most efficient computational graph with intermediate functions f_i, where $+, \exp, \cdot^2$ and sin are the primitive computational steps. (1 pt)

(b) Compute the derivative $\frac{\partial y}{\partial x}$ by *forward-mode differentiation*. In other words, compute the derivatives of your intermediate functions starting from the x-node and going *forward* to the y-node. Use the chain rule in each case to make use of the derivatives you already computed. (2 pts)

(c) Compute the derivative $\frac{\partial y}{\partial x}$ by *reverse-mode differentiation*. In other words, compute the derivatives of your intermediate functions starting from the y-node and going *reverse* to the x-node. Use the chain rule in each case to make use of the derivatives you already computed. (2 pts)

2 ADAM optimizer

(a) Write down the formulae for the Adaptive Moment Estimation (ADAM) optimizer, and write a short sentence describing what each line does. (1 pt)

(b) Show that in the very first iteration, the update of the parameters is reduced to the sign of the gradient g. (2 pts)

(c) How could this potentially unwanted behavior be avoided? (1 pt)

(d) Consider an MLP with a set of weights w trained with Adam and L2-regularization. Does it make a difference whether the L2-penalty $\|w\|_2^2$ is included in the loss, or whether weight decay is applied to the weights directly? Can you argue why one may be better than the other? *Hint: AdamW.* (1 pt)

3 Heteroscedastic regression

Use the dataset `x_y.csv` in the data folder. It contains 1,000 data points with 1-dimensional inputs, $\boldsymbol{x}$, and 1-dimensional outputs, y, as shown in the following figure

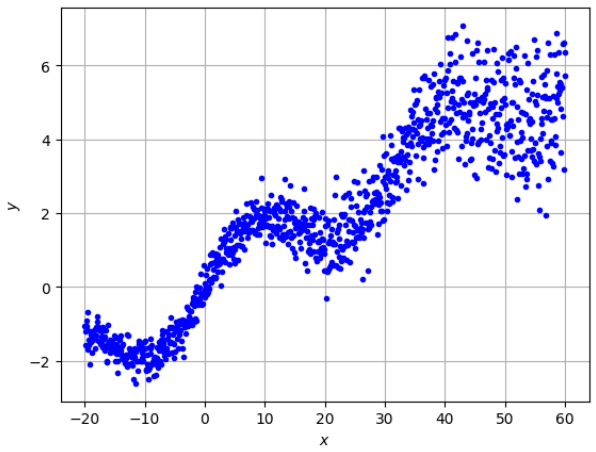

Figure 1: Scatter plot of the dataset contained in `x_y.csv`

(a) Load the dataset, and randomly split it into a training and a test set with ratio (66/33%). (1 pt)

(b) We model the distribution of outputs as

$$p(y|\boldsymbol{x},\boldsymbol{\theta}) = \mathcal{N}(y|f_\mu(\boldsymbol{x}),\sigma^2),$$

where $f_\mu(\boldsymbol{x})$ will be a multilayer perceptron (MLP), and we assume that all measurement errors are identical (i.e., homoscedastic regression), $\sigma(\boldsymbol{x}) = \sigma$. Explain why the mean-squared error is a reasonable log-likelihood function. (1 pt)

(c) Build an MLP with the following architecture: 3 fully-connected linear layers with hidden dimensions 32 and 64 (i.e., the dimensionality of your network should yield $1 \to 32 \to 64 \to 1$). Connect them with the ReLU activation function. Optimize the log-likelihood on the training set. Feel free to use any optimizer (e.g., stochastic gradient descent or Adam). Plot the resulting model on the test set together with the reference data points. Are you underfitting, overfitting? (2 pts)

(d) Play around with the architecture (number of layers / number of neurons / activation functions / regularization / ...) and try to improve the performance of the model. Plot train and validation errors for

- a learning rate that is too big,
- a learning rate that is too small and
- a learning rate scheduler: linear ramp up with cosine annealing. (2 pts)

(e) One standard way to address the prediction of heteroscedastic data for regression is to predict both the mean and the variance of a Normal distribution: $f_\mu(\boldsymbol{x}) = E[y|\boldsymbol{x},\boldsymbol{\theta}]$ and $f_{\sigma^2}(\boldsymbol{x}) = \mathrm{Var}[y|\boldsymbol{x},\boldsymbol{\theta}]$. Derive a log-likelihood function for this heteroscedastic problem. (2 pts)

(f) Modify the MLP to output two values: the mean and the variance. Optimize the model. Plot the mean and variance predicted by your model on the test set together with the reference data points. Are you underfitting, overfitting? (2 pts)

Hint: In case you use PyTorch, you may find the following functions useful to reshape tensors and arrays: `numpy.reshape` and `torch.squeeze`.

4 Bonus: ResNet properties

Consider a ResNet with L residual blocks. Each block takes an input x and outputs

$$T(x) = x + F(x),$$

and the full network is the composition of these blocks:

$$f = T_L \circ \cdots \circ T_1.$$

Assume that each block function F is 1-Lipschitz (i.e., $\|F(x) - F(y)\| \leq \|x - y\|$) and that there is no batch normalization or any other operation that changes the Lipschitz constant.

(a) Explain why the mapping $T(x) = x + F(x)$ is still Lipschitz. Derive an upper bound for $\mathrm{Lip}(T)$. (1 bonus pt)

(b) Derive an upper bound on the Lipschitz constant of the full network f as a function of depth L. (2 bonus pts)

(c) Explain why including batch normalization complicates the guarantee of a given Lipschitz constant at test time. (2 bonus pts)