Assignment 3

Numerical Optimization / Optimization for CS WS2021

Christian Kopf, `christian.kopf@icg.tugraz.at`
Lukas Erlbacher, `lukas.erlbacher@student.tugraz.at`
Thomas Wedenig, `thomas.wedenig@student.tugraz.at`

December 14, 2021

Deadline: Jan 11th, 2022 at 23:59

Submission: Upload your report and your implementation to the TeachCenter. Please use the provided framework-file for your implementation. Make sure that the total size of your submission does not exceed 50MB. Include **all** of the following files in your submission:

- `report.pdf`: This file includes your notes for the individual tasks. Keep in mind that we must be able to follow your calculation process. Thus, it is not sufficient to only present the final results. You are allowed to submit hand written notes, however a compiled LaTeX document is preferred. In the first case, please ensure that your notes are well readable.
- `main.py`: This file includes your python code to solve the different tasks. Please only change the marked code sections. Also please follow the instructions defined in `main.py`.
- `figures.pdf`: This file is generated by running `main.py`. It includes a plot of all mandatory figures on separate pdf pages. Hence, you do not have to embed the plots in your report.
- `model_GD.json`: Model parameters of the trained steepest descent model as JSON file.
- `model_NAG.json`: Model parameters of the trained Nesterov model as JSON file.

1 Species Classification with Neural Networks

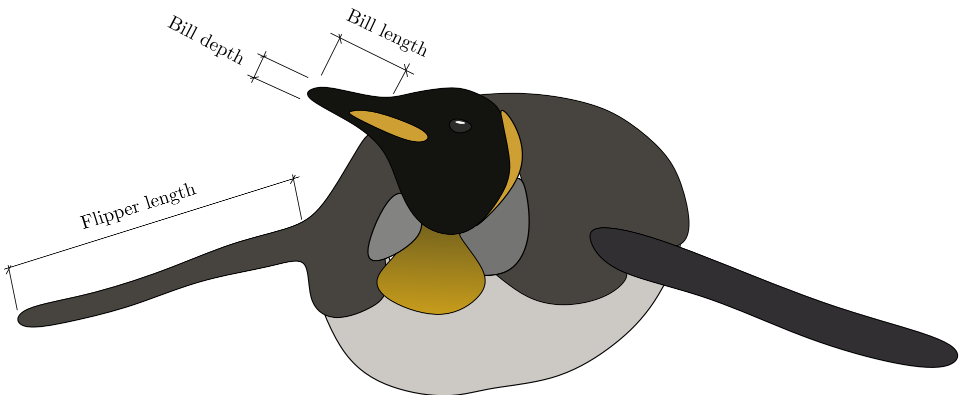

Figure 1: Illustration of the body measurements. The specimen correspondences to the labels are given by {Adelié: $y^s = 0$, Chinstrap: $y^s = 1$, Gentoo: $y^s = 2$} .

The task of this assignment is to classify penguin specimen given some body measurements. In particular you are given a modified version of the Palmer Penguin [1] dataset. To solve this non-linear problem, we use a small feed-forward neural network and a dataset comprised of 280 training samples and 40 test samples. Each sample is given as a tuple $(\mathbf{x}^s, y^s)$, where $\mathbf{x}^s = (x_1^s\ x_2^s\ x_3^s\ x_4^s)^\top \in \mathbb{R}^4$ is the input and $y^s \in \{0, 1, 2\}$ is the target label of the sth sample. The

[1] `https://github.com/allisonhorst/palmerpenguins`

inputs x_1^s, x_2^s correspond to the bill length and depth in centimeters and x_3^s is the flipper length also in centimeters and x_4^s is the mass of the specimen in kilograms. An indication of the measurements are given in fig. 1.

2 Network Architecture

To achieve this task, we use a fully connected neural network with 1 hidden layer. In neural networks, the term fully connected means that every neuron from the previous layer is connected to every neuron of the subsequent layer. Using this approach, classifying multiple classes is typically performed in a lifted space that represents discrete probability distributions over the labels. Thus, the network predicts for *each* label how likely it is. To achieve this goal, we setup a neural network f that maps an input $\mathbf{x} \in \mathbb{R}^{N_I}$ (N_I is the input dimension) and the learnable network parameters ($\mathbf{W}^{(0)}$, $\mathbf{b}^{(0)}$, $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$) to the probability simplex $S^{N_O-1} = \{\mathbf{y} \in \mathbb{R}^{N_O} : y_i \geq 0, \sum_{j=0}^{N_O} y_i = 1\}$ (N_O is the output dimension), i.e.

$$f(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(0)}, \mathbf{b}^{(0)}) \coloneqq g\left(\mathbf{W}^{(1)} h\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right) + \mathbf{b}^{(1)}\right).$$

The activation function h applies the so called softpuls function to each element of the pre-activation of the input layer $\mathbf{z}^{(1)} = \mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)} \in \mathbb{R}^{N_H}$ (N_H is the hidden dimension)

$$\mathbf{a}^{(1)} = h(\mathbf{z}^{(1)}) \iff a_i^{(1)} = \ln(1 + \exp(z_i^{(1)}))\,.$$

The output of the network $\widetilde{\mathbf{y}} \in \mathbb{R}^{N_O}$ is eventually computed by applying the softmax function to the pre-activation of the output layer $\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)} \in \mathbb{R}^{N_O}$

$$\widetilde{\mathbf{y}} = g(\mathbf{z}^{(2)}) \iff \widetilde{y}_i = \frac{\exp(z_i^{(2)})}{\sum_{j=1}^{N_O} \exp(z_j^{(2)})}\,.$$

The elements of the output of the softmax function are all positive and sum up to one. Thus, the output of the neural network $\widetilde{\mathbf{y}}$ can indeed be interpreted as a discrete probability distribution over all labels. The actual prediction of the network is then defined as

$$\widetilde{y} = \arg\max_i \widetilde{y}_i,$$

i.e. the most likely label. Refer to fig. 2 for a complete overview of the network structure.

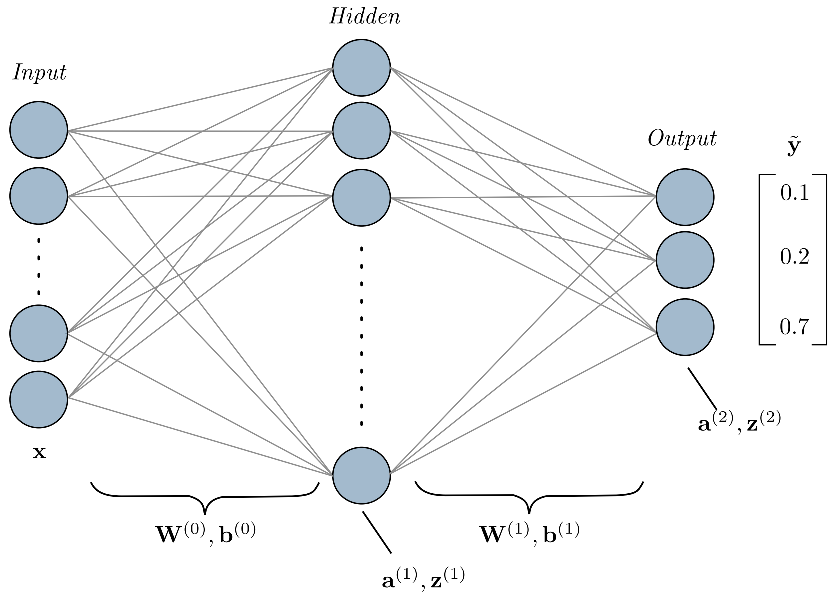
Hidden
Input
Output
$\tilde{\mathbf{y}}$
0.1
0.2
0.7
$\mathbf{x}$
$\mathbf{a}^{(2)}, \mathbf{z}^{(2)}$
$\mathbf{W}^{(0)}, \mathbf{b}^{(0)}$
$\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$
$\mathbf{a}^{(1)}, \mathbf{z}^{(1)}$

Figure 2: Structure of a feed forward neural network with one hidden layer.

3 Optimization

Assume we have given a set of S input-output samples $\{(\mathbf{x}^s, y^s) : s = 1, \ldots, S\}$, where $\mathbf{x}^s \in \mathbb{R}^4$ is an input image and $y^s \in \{0, 1, 2\}$ is the corresponding ground-truth label. First, we convert each ground-truth label y^s into a discrete target probability distribution $\mathbf{y}^s$. For example let $y^s = 2$, then $\mathbf{y}^s = (0,0,1)^\top$. Then, the training process consists of adapting the parameters of the network $\{\mathbf{W}^{(0)}, \mathbf{b}^{(0)}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}\}$ such that the output of the network $\widetilde{\mathbf{y}}^s = f(\mathbf{x}^s, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(0)}, \mathbf{b}^{(0)})$ is as close as possible to the ground-truth distribution $\mathbf{y}^s$ for all samples. This can be achieved by minimizing a cost function through back-propagation. A common loss function in multi-class classification is the cross-entropy loss which is in our case defined as

$$l(\widetilde{\mathbf{y}}^s, \mathbf{y}^s) = -\sum_{i=1}^{N_O} y_i^s \ln(\widetilde{y}_i^s)\,.$$

The total loss for the whole dataset is simply the average of each sample-loss l across the dataset , i.e.

$$\mathcal{L} = \frac{1}{S}\sum_{s=1}^{S} l(\widetilde{\mathbf{y}}^s, \mathbf{y}^s)\,.$$

In each iteration of the *training* process the objective is given by minimizing the cost function $\mathcal{L}$. Each iteration is comprised of a *forward* and a *backward* pass. During the forward pass the total loss L is computed for the all S samples. In the backward pass, all network parameters are updated through a gradient method based on the loss L. This is referred to as *back-propagation*.

3.1 Steepest Descent Algorithm

The simplest option to perform the back-propagation step is to apply the steepest descent algorithm, where you need to compute the gradient of the total loss w.r.t. to each learnable network parameter as presented in the exercise lecture. Let $\mathbf{p} \in \{\mathbf{W}^{(0)}, \mathbf{b}^{(0)}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}\}$ represent a network parameter. The k^{th} update step of the steepest descent algorithm for the parameter $\mathbf{p}$ reads as

$$\mathbf{p}^{k+1} = \mathbf{p}^k - t^k \nabla\mathcal{L}(\mathbf{p}^k),$$

where $\nabla\mathcal{L}(\mathbf{p}^k)$ is the gradient of the total loss w.r.t. $\mathbf{p}$ at the position $\mathbf{p}^k$

$$\nabla\mathcal{L}(\mathbf{p}^k) = \left.\frac{\partial\mathcal{L}}{\partial\mathbf{p}}\right|_{\mathbf{p}^k},$$

and t^k denotes the step size.

3.2 Nesterov Accelerated Gradient Method

A more complex gradient method is given by the Nesterov Accelerated Gradient method (NAG). For this method *momentum* is incorporated into the update process based on the steps stated in Algorithm 1. This algorithm computes an intermediate point $\mathbf{q}^k$ from the current and previous points $\mathbf{p}^k, \mathbf{p}^k$ and makes an extrapolated gradient step. t^k again denotes the step size.

Algorithm 1: NAG Algorithm

```
Set $\theta^1 = 0, \quad \mathbf{p}^1 = \mathbf{p}^0$
Choose $t^k > 0$
for $k > 0$ do
    $\theta^{k+1} = \frac{1+\sqrt{1+4(\theta^k)^2}}{2}$
    $\mathbf{q}^k = \mathbf{p}^k + \frac{\theta^k - 1}{\theta^{k+1}}\left(\mathbf{p}^k - \mathbf{p}^{k-1}\right)$
    $\mathbf{p}^{k+1} = \mathbf{q}^k - t^k\nabla\mathcal{L}(\mathbf{q}^k)$
end
```

4 Tasks

Pen & Paper:

- For the given dataset, what is the size of the matrices $\mathbf{W}^{(0)}$, $\mathbf{W}^{(1)}$, and the bias terms $\mathbf{b}^{(0)}$ and $\mathbf{b}^{(1)}$? Specify the number of learnable parameters of the neural network as a function of N_H.
- Compute the derivative of the total loss w.r.t. to all model parameters using the results from the practical exercise session.

In Python:

- Initialize the parameters using a normal distribution with $\mu = 0$ and $\sigma = 0.05$.
- Implement the forward pass of the network.
- Implement the backward pass by computing the gradients w.r.t. the model parameters.
- With one data sample verify your gradient using Scipy's `approx_fprime()`[2]. Report your findings.
- Implement the steepest descent algorithm.
- Implement Nesterov's method (NAG).
- For both gradient methods: train the neural network using the provided *training data* for 350 iterations with the steepest descent algorithm for $N_H = 16$. Select a suitable step size $\eta \in [0.1, 0.001]$ and keep it constant over the iterations.
- In a plot compare the training loss of both variants over the iterations. In another plot compare the training accuracy of both variants over the iterations. Use a y-logarithmic scale for the loss plot [3]. The accuracy is defined as the average number of correctly classified samples

$$\mathcal{A} = \frac{1}{S} \sum_{s=1}^{S} \delta(y^s, \widetilde{y}^s) \quad \text{with } \delta(u, v) = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{else,} \end{cases}$$

  where δ is the indicator function.
- Apply the learned network to the *test data* and add the *accuracy* over the iterations to their respective plot to compare both gradient methods.
- Discuss the plots regarding the convergence rate of both gradient methods. State and interpret the final accuracy w.r.t to the test set.
- Export the trained models (both variants) using the given function `export_model()`.

Show and describe all your results in the report!

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.approx_fprime.html`
[3] `https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.semilogy.html`