UNIVERSITE DE GENEVE — Section de Mathématiques
Faculté des sciences

Optimization with Applications I
Projects

One part of the final exam will be a presentation of your solution of one of these problems. You can pick whichever interests you most. Please make sure to document your code well and even more importantly create instructive visualizations that help you explain what is going on and that exemplify the correctness of your code. You will be asked to submit your code a few days before the actual exam.

In general these projects are open-ended. There are many correct ways of solving the problems and many ways to illustrate that some method is working. Here initiative and creativity is appreciated.

Problem 1 (Total Variation Smoothing)**.** A good way to denoise signals is total variation smoothing. In $1D$ we are given an input vector $\mathbf{y} \in \mathbb{R}^N$ and want to find

$$\arg\min_x ||\mathbf{y} - \mathbf{x}||_2^2 + \lambda \sum_{i=0}^{N-1} |x_i - x_{i+1}|.$$

Here we run this on data of the form $y_n = \mu_n + \epsilon_n$ with i.i.d. Gaussian ϵ_n and a piecewise constant function given by the values μ_n.

Implement an algorithm finding a solution and visualize examples of results. Compare this to other smoothing algorithms.

For a bonus solve this in the 2D-case. You can then test this on images that you add noise to.

Problem 2 (Lasso optimal descent)**.** Implement an algorithm solving the Lasso

$$\min_\alpha ||X\alpha - y||_2^2 + \lambda ||\alpha||_1$$

in the naive method and with optimal descent respectively. Compare the runtimes of the two algorithms in terms of CPU time as well as number of iterations.

The least squares minimization corresponds to data with an assumption of Gaussian noise. Adapt the Lasso to data with Poisson noise

Problem 3 (Backpropagation and Neural Networks)**.** For differentiable activation functions $\sigma_l : \mathbb{R} \to \mathbb{R}$ We define a collection of weights $w = (W_1, b_1, \dots, W_L, b_L)$ consisting of matrices W_i and vectors b_i. The neural network function $f(x, w) = \alpha^L(x, w)$ is then inductively defined by

$$\begin{aligned} z^{(0)}(x, w) &= x \\ z^{(l+1)}(x, w) &= W_{l+1}\alpha^{(l)}(x, w) + b_{l+1} \\ \alpha^l(x, w) &= \sigma_l(z^l(x, w)). \end{aligned}$$

We are looking for weights w minimizing some cost function $C(f(x, w), y)$. The backpropagation algorithm is an efficient method of calculating the gradient with respect to the w in such a network structure. It consists of a forward pass calculating and saving the derivatives of the subnetwork up to the point and a backwards pass applying the chain rule.

Without using prebuilt neural network libraries implement a network and the backpropagation algorithm using numpy. Make sure to make your code a modular as possible. You can compare the gradients your code creates with what reference implementations return.

A good way to demonstrate that your code works is to actually train your network on a data set like MNIST to classify images. Here you should research what typical initialization procedures and optimization algorithms are. A network with 2 hidden layers of size around 100 should already produce great results.

Problem 4 (Open)**.** Pick a problem yourself.

Choose one that you can broadly apply some of the methods we learned in class to and code away. Make sure to formalize the problem you are trying to solve and be able to explain the challenges along the way.

Deliverables

— **Code** : The code should be runnable, commented and reproducable.

— **Report** : A short summary of what you did. (One page is enough if your explanation fits there.)

You will be asked to submit the code and report before the exam. At the exam you will then present your solutions.

Of course we are available in case of questions or for guidance. If you are not sure about some project ideas do not hesitate to reach out.