PhD course “Nonlinear Optimization”
Assignment-2

The deadline to send reports is April 1, 2022. Don’t hesitate to ask me by email if something is not clear.

Choose one problem among three below and prepare your report in one of the following ways: (i) pdf file, (ii) Jupyter notebook, (iii) html file/link or ask me if you want to do something more exotic. Don’t expect me to run your code. You can use any programming language you want, no need to send me the actual code (but you can if you want). In the report additionally to plots and conclusions describe what you observed, what was surprising for you, what you did not understand, etc. It doesn’t need to be long, about 1 page is sufficient if it contains all what I asked for.

Send it as a one file (or archive), with your name in it.

The long description below is needed only for Problem 1 and Problem 2.

Logistic regression. Given dataset $(a_1, b_1), \dots, (a_n, b_n)$, with $a_i \in \mathbb{R}^d$, $b_i \in \{-1, 1\}$, we want to solve

$$\min_{x\in\mathbb{R}^d} f(x) := \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-b_i\langle a_i, x\rangle)) \qquad (1)$$

Motivation. (You can skip it if you don’t care). For example, a_i may represent images, and b_i whether the image a_i depicts a cat or a dog. After we solve problem (1) and obtain some x^* (approximation of a true minimizer), we can classify new data points $a_{n+1}, a_{n+2}, \dots$ by the following rule:

$$b_{n+1} = \operatorname{sign}\langle a_{n+1}, x^*\rangle$$

and so on.

Suggestions

(a) For computing gradients you are free to do whatever you want: analytical expressions or automatic differentiation are OK.

(b) If you compare $f(x_k) - f_*$, first run some method to find this f_*.

Data

You can choose either synthetic not interesting data or take some existing ML dataset:

(a) Synthetic non-interesting data.

$n = 100$, $d = 90$, $A \in \mathbb{R}^{m\times n}$ with Gaussian iid entries and set a_i to be the i-th row of A. Set b as signs of a vector in $\mathbb{R}^n$ also with Gaussian entries. In python it is done by

```
import numpy as np
n = 100
d = 90
A = np.random.randn(n,d)
b = np.sign(np.random.randn(n))
```

(b) Real datasets. You can use any ML dataset you find, for example check https://archive.ics.uci.edu/ml/datase
Alternatively, some of them you can find directly in python/julia/matlab, perhaps with some extra library.

Problems

1. **Gradient descent**

   - For the problem above and the dataset you have chosen implement gradient descent method with a fixed stepsize α. For this you need to choose right α. Either compute the Lipschitz constant L of ∇f or tune α by trial and error.
   - Implement an adaptive version of gradient descent with the update for α_k:

   $$\alpha_k = \min\left\{\sqrt{1+\frac{\alpha_{k-1}}{\alpha_{k-2}}}\alpha_{k-1}, \frac{\|x_k - x_{k-1}\|}{2\|\nabla f(x_k) - \nabla f(x_{k-1})\|}\right\}$$

   This update makes sense only after the first iteration, so for the first iteration run a standard step of GD with a small stepsize.
   - Compare both methods in terms of $\|\nabla f(x_k)\|$ or $f(x_k) - f_*$ vs. the number of iterations k. Make plot, write down some observations/conclusions.

2. **SGD**

   For the same problem implement SGD with different regimes for stepsizes:

   (a) $\alpha_k = \alpha$ for all k;

   (b) $\alpha_k = \frac{c_1}{c_2+k}$;

   (c) $\alpha_k = \frac{c_1}{c_2+\sqrt{k}}$;

   (d) $\alpha_k = \frac{c_1}{c_2+\sqrt{\lceil \frac{k}{n}\rceil}}$, where $\lceil a \rceil$ denotes the integer part of a.

   (e) $\alpha_k = \frac{\alpha}{\sqrt{10^{-6}+\sum_{i=0}^k \|g_i\|^2}}$, where g_i are stochastic gradients from previous iterations $0, \dots, k$

   Instead of some regimes above, you can choose your own. Don't be afraid to be creative! Your first goal is to find good constants c_1, c_2, α for every method. They don't need to be the best, just good enough. Then compare all 5+ methods. One iteration of SGD is quite cheap, so be ready to run the method for many iterations. To compare algorithms, compute $f(x_k) - f_*$ or $\|\nabla f(x_k)\|$ every n iterations. Here you don't need to solve problem to high accuracy.

3. **Linear system.**
   Given $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, we want to solve $Ax = b$. Let a_i be the i-th row of A. Generate any reasonable data, $m \geqslant 100$, $n \geqslant 100$, but make sure that the linear system has a solution.

   Kaczmarz's method:

   $$x_{k+1} = P_{\text{row } i} x_k = x_k - \frac{\langle a_i, x_k\rangle - b_i}{\|a_i\|^2} a_i,$$

   where $i = k \mod (m+1)$.

   Randomized Kaczmarz's method:

   $$\text{Sample } i \in \{1, \dots, m\} \text{ with probabilities } \frac{\|a_i\|^2}{\|A\|_F^2}$$
   $$x_{k+1} = P_{\text{row } i} x_k = x_k - \frac{\langle a_i, x_k\rangle - b_i}{\|a_i\|^2} a_i,$$

   (a) Implement Kaczmarz method and randomized Kaczmarz method

   (b) Find two instances of A and b (of any dimensions, random or deterministic), on which each method will show better performance. We are interested in the objective value $\|Ax_k - b\|^2$ with respect to the number of iterations (or epochs).

   (c) Can we interpret these methods as some variants of stochastic gradient descent?

   (d) Implement gradient descent and compare it with randomized Kaczmarz method on a data of your choice Be careful with the comparison: one iteration of Kaczmarz method is roughly n times cheaper than the one of GD.