OPTIMIZATION FOR MACHINE LEARNING

December 5, 2024

Today: Last session!

- Short lab on sparsity
- Project presentation
- Last-minute questions and comments

Lab 4, question 2

$$\underset{x \in \mathbb{R}}{\text{minimize}} \quad f_1(x) = a(x-u) + \frac{L}{2}(x-u)^2 + \lambda |x|$$

$a \in \mathbb{R}$, $L > 0$, $u \in \mathbb{R}$, $\lambda > 0$

$$\partial f_1(x) = \begin{cases} \{a + L(x-u) + \lambda\} & \text{if } x > 0 \\ \{a + L(x-u) - \lambda\} & \text{if } x < 0 \\ [a + L(x-u) - \lambda,\ a + L(x-u) + \lambda] & \text{if } x = 0 \end{cases}$$

f_1 is convex

so $x^* \in \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} f_1(x) \iff 0 \in \partial f_1(x^*)$

Suppose that $x^* > 0$. Then $0 \in \partial f_1(x^*) \iff 0 = a + L(x^* - u) + \lambda$

$$\iff x^* = u - \frac{a}{L} - \frac{\lambda}{L}$$

only possible if

$$u - \frac{a}{L} > \frac{\lambda}{L}$$

Suppose that $x^* < 0$. Then, similarly,

$$0 \in \partial f_1(x^*) \iff 0 = a + L(x^* - u) - \lambda$$

$$\iff x^* = u - \frac{a}{L} + \frac{\lambda}{L}$$

only possible if

$$u - \frac{a}{L} < -\frac{\lambda}{L}$$

Suppose that $x^0 = 0$ Then, $0 \in \mathcal{G}_1(x^0) \Leftrightarrow 0 \in \left[a + L(x^0 - u) - \lambda,\ a + L(x^0 - u) + \lambda\right]$

$\Leftrightarrow 0 \in \left[a - Lu - \lambda,\ a - Lu + \lambda\right]$

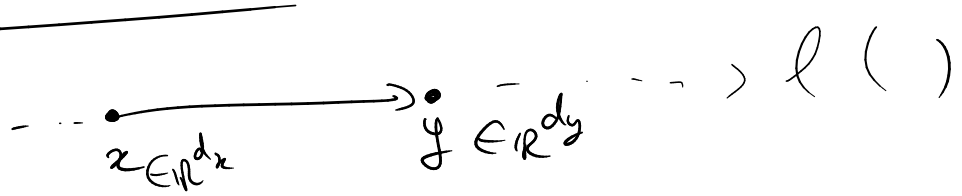
only possible if

$$\begin{cases} a - Lu - \lambda \leq 0 \\ a - Lu + \lambda \geq 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} \frac{a}{L} - u - \frac{\lambda}{L} \leq 0 \\ \frac{a}{L} - u + \frac{\lambda}{L} \geq 0 \end{cases}$$

$$\Leftrightarrow u - \frac{a}{L} \in \left[-\frac{\lambda}{L}, \frac{\lambda}{L}\right]$$

My take on JVP



$$\underbrace{\nabla_y \ell}_{\in \mathbb{R}^{d_y}} = \underbrace{\frac{\partial y}{\partial x}}_{\in \mathbb{R}^{d_y \times d_x}} \underbrace{\nabla_x \ell}_{\in \mathbb{R}^{d_x}}$$

$\Rightarrow \forall v \in \mathbb{R}^{d_y}$,

$$\langle \nabla_y \ell, v \rangle = \left\langle \frac{\partial y}{\partial x} \nabla_x \ell, v \right\rangle$$

To get $\nabla_x \ell$ from $\nabla_y \ell$, we seek a matrix A in $\mathbb{R}^{d_x \times d_y}$ such that

$$\langle \nabla_x \ell, v \rangle = \langle A \nabla_y \ell, v \rangle$$

partly because $\left\langle \frac{\partial y}{\partial x} u, v \right\rangle = \left\langle u, \left[\frac{\partial y}{\partial x}\right]^T v \right\rangle$, we can

show that $A = \left[\frac{\partial y}{\partial x}\right]^T \Rightarrow \nabla_x \ell = \left[\frac{\partial y}{\partial x}\right]^T \nabla_y \ell$

https://github.com/sriharikrishna/siamcse23

https://www.jmlr.org/papers/volume18/17-468/17-468.pdf

my nice reading / tutorial suggestions about auto. diff

Course project — Quasi-Newton methods

Motivation

- GD and related methods (SG, PG, CD, ...) is sensitive to the scale of problem
  → Not scale invariant

(1) minimize $f(x)$ $x \in \mathbb{R}^d$ and (2) minimize $f(Ay)$ $y \in \mathbb{R}^d$ where $A \in \mathbb{R}^{d \times d}$ invertible

GD on (1) $\quad x_{k+1} = x_k - \alpha_k \nabla f(x_k)$

GD on (2) $\quad y_{k+1} = y_k - \alpha_k A \nabla f(Ay_k)$

$\downarrow \; x_k = Ay_k$

$$x_{k+1} = x_k - \alpha_k A^2 \nabla f(x_k)$$

behavior of this iteration can be very different from that of GD on (1), especially if A is ill-conditioned

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ well conditioned $\quad \kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)} = 1$

$A = \begin{bmatrix} 10^{20} & 0 \\ 0 & 10^{-64} \end{bmatrix}$ badly conditioned $\quad \kappa(A) = 10^{84}$

→ Part of the philosophy of Adagrad, RMSProp, Adam, etc. consists in developing methods that are less sensitive to these scaling issues

→ Another way of making a method scale-invariant consists in using the second-order derivative. (assuming that it exists!)

⇒ Most famous method: Newton's method

$$x_{k+1} = x_k - \left[\nabla^2 f(x_k)\right]^{-1} \nabla f(x_k)$$

↑ Use inverse Hessian as "descent matrix"

⊕ Fast convergence near a solution in certain cases

⊖ Does not always converge

⊖ Not always well-defined unless $\nabla^2 f(x) \succ 0 \ \forall x$ (strictly convex)

⊖ Expensive to compute $\nabla^2 f(x) \in \mathbb{R}^{d \times d}$

In quasi-Newton methods, replace $\nabla^2 f(x_k)^{-1}$ by a matrix that is computed using only gradient information!

Quasi-Newton iteration (for us):

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$$

where $H_k \succ 0$ that is built using $\{y_j\}, \{s_j\}$

$$s_j = x_{j+1} - x_j$$
$$y_j = \nabla f(x_{j+1}) - \nabla f(x_j)$$

At every iteration, H_{k+1} is built from H_k using $s_k = x_{k+1} - x_k$, $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

Most popular technique for updating H_k: BFGS formula

→ BFGS proposed in late 1970s

→ BFGS + stochastic gradients ~2014

Project: • Implement BFGS and compare it with GD!

• Implement the stochastic version and compare it with SG/Adagrad

• Choose proximal BFGS or coordinate BFGS and compare with PG or CD

b) One issue with BFGS / QN in general: Eventually one stores

$H_k \in \mathbb{R}^{d \times d}$ as a dense matrix

⇒ The limited memory variant (L-BFGS) addresses this issue by building H_k using only the last m pairs

$\{s_i, y_i\}_{i=k-1,\dots,k-m}$ ($m = 5$ or $m = 10$)

Cost $2md \ll d^2$

⇒ Method from late 1980s, implemented in scikit-learn and PyTorch

NB: Default in scikit-learn for logistic regression