Convex Optimization: Homework 2

Instructor: Yuanzhi Li
Carnegie Mellon University
Due: March 6th 2020 at 11:59PM

1 Projection to Convex Set

In this problem you want to show how to project onto a convex set.

1.1 The unit ball (20 points)

A d-dimensional unit ball $\mathcal{B}$ is a set in $\mathbb{R}^d$, defined as the set of points $x \in \mathbb{R}^d$ satisfying the following constraints:

$$\mathcal{B} = \{x \in \mathbb{R}^d \mid \|x\|_2^2 \leq 1\}$$

Show that

$$\Pi_{\mathcal{B}}(x) = \begin{cases} x & \text{if } \|x\|_2 \leq 1; \\ \frac{x}{\|x\|_2} & \text{if } \|x\|_2 > 1. \end{cases}$$

(Hint: On the contrary you can assume when $\|x\|_2 > 1$, $\Pi_{\mathcal{B}}(x) = z \neq \frac{x}{\|x\|_2}$, then $z' = \langle z, \frac{x}{\|x\|_2} \rangle \widehat{x}$ where $\widehat{x} = \frac{x}{\|x\|_2} \in \mathcal{B}$. What would happen to this point z'?)

1.2 The subspace (10 points)

Let a k-dimensional affine subspace $\mathcal{S}$ in $\mathbb{R}^d$ be defined as: For a matrix $U \in \mathbb{R}^{d \times k}$ such that $U^\top U = I$, and a vector $b \in \mathbb{R}^k$ such that:

$$\mathcal{S} = \{x \in \mathbb{R}^d \mid U^\top x = b\}$$

Give a closed-form formula for the projection to subspace: $\Pi_{\mathcal{S}}(x)$.

Hint: Use the Karush–Kuhn–Tucker conditions which are sufficient for optimality in linear programs.

1.3 The simplex (10 points)

The $d-1$-dimensional simplex Δ is defined as the set of points $x \in \mathbb{R}^d$ satisfying the following constraints:

$$\Delta = \{x \in \mathbb{R}^d \mid \forall i \in [d], x_i \geq 0, \quad \sum_{i \in [d]} x_i = 1\}$$

Where x_i is the i-th coordinate of x. Give a closed-form formula for the projection to simplex: $\Pi_\Delta(x)$ in terms of ρ, the number of non-zero elements of the vector Π_Δ.

Bonus (10 points): Give an efficient algorithm ($O(d \log d)$) to calculate the projection.

(Hint: write it as $\min\{\frac{1}{2}\|z - x\|_2^2, z \in \Delta\}$. Use Lagrange Multiplier to transfer a constrained optimization to unconstrained optimization, and find the closed-form solution.)

2 Stochastic gradient descent algorithm

2.1 Stochastic gradient descent with decaying learning rate (20 points)

In the lecture we learnt stochastic gradient descent with a fixed learning rate η, now you will consider stochastic gradient descent with different learning rates η_t at every iteration t:

Now, given a convex function $f : \mathbb{R}^d \to \mathbb{R}$, whose stochastic gradient $\widetilde{\nabla} f(x)$ satisfies for every $x \in \mathbb{R}^d$, $\mathbb{E}[\widetilde{\nabla} f(x)] = \nabla f(x)$ and $\mathbb{E}[\|\widetilde{\nabla} f(x)\|_2^2] \leq G$ for a fixed value G.

Consider the following update:

$$x_{t+1} = x_t - \eta_t \widetilde{\nabla} f(x_t)$$

Prove that for every $x \in \mathbb{R}^d$, it holds:

$$\frac{1}{\sum_{s=0}^{T-1} \eta_s} \sum_{t=0}^{T-1} (\eta_t \, \mathbb{E}[f(x_t)]) \leq f(x) + \frac{\|x_0 - x\|_2^2 + G \sum_{t=0}^{T-1} \eta_t^2}{2 \sum_{t=0}^{T-1} \eta_t}$$

(Hint: use the mirror descent analysis, when you do the telescoping sum, try to re-weight each term so the *old minus new*'s still cancel each other).

2.2 Stochastic gradient descent with better stochastic gradient norm bound (10 points)

Given a L-smooth convex function $f : \mathbb{R}^d \to \mathbb{R}$, whose stochastic gradient $\widetilde{\nabla} f(x)$ satisfies for every $x \in \mathbb{R}^d$, $\mathbb{E}[\widetilde{\nabla} f(x)] = \nabla f(x)$ and $\mathbb{E}[\|\widetilde{\nabla} f(x)\|_2^2] \leq 2\|\nabla f(x)\|_2^2$.

Consider the following update:

$$x_{t+1} = x_t - \eta \widetilde{\nabla} f(x_t)$$

Show that as long as $\eta \leq \frac{1}{2L}$, then for any $x \in \mathbb{R}^d$ farther away from x_0 than $x^\star$ it holds that:

$$\mathbb{E}[f(\overline{x}_T)] \leq f(x) + O\left(\frac{\|x - x_0\|_2^2}{\eta T}\right)$$

where $\overline{x}_T = \frac{1}{T} \sum_{t=0}^{T-1} x_t$.

(Hint: You just need to follow step-by-step the gradient descent analysis in the second lecture).

2.3 SGD Implementation (50 points)

Consider the following logistic regression objective: Given training data points and labels $\{x_i, y_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^d$ and $y_i \in [-1, 1]$, define the logistic regression function as:

$$f(w) = \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \langle w, x_i \rangle}\right)$$

Now, you will implement stochastic gradient descent algorithm to optimize this objective function. We will use the MNIST data set for classifying hand-written digits (0-9). We will pick all the

training data corresponding to digit 0 and 1 and y_i is -1 if the digit is 0 and 1 otherwise. There should be in total approximately $N = 12,000$ training examples (each digit has approximately 6,000 training examples).

Implement the stochastic gradient descent algorithm (starting from $w_0 = 0$) with batch size $1, 10, 100$, and learning rate $\eta = 1, 0.3, 0.1, 0.03$ (so there are 12 total combinations). Plot the training loss (over the whole dataset, not just your mini-batch) versus the number of iterations, and the training loss (over the whole dataset, not just your mini-batch) versus the running time. What do you discover? Does it agree with the theory in the lecture? Please give some simple explanations of your findings.

Clarifications:

1. Please code in Python. If you have a major problem with Python, please contact the TAs before using a different language without prior approval.
2. Please submit your code as part of your Gradescope submission; label these pages as part of this question.
3. Do not import any premade packages that implement SGD or other such gradient calculations.
4. Scale the features of your input data to be between 0 and 1.
5. Implement true SGD (sample each index using a uniform distribution).
6. Please show work for computing the gradient. It is fine to *check* your solution with an autodiff package, but you should also have a mathematical derivation of $\nabla_w f(w)$ included in your pdf submission, *and* your Python code should explicitly compute the gradient at each step.
7. Run for 500 iterations (i.e., 500 parameter updates).
8. Submit 5 plots for training loss vs. **number of iterations**: 1 plot for each learning rate (each of these plots should have 3 curves, 1 per batch size) and a 5th final plot with all 12 curves. Please include clear legends on your plots.
9. Submit 5 plots for training loss vs. **running time**: 1 plot for each learning rate (each of these plots should have 3 curves, 1 per batch size) and a 5th final plot with all 12 curves. Please include clear legends on your plots.

   Note: 10 plots total!

3 Distributed optimization

3.1 The Naive ADMM (10 points)

Suppose one does not introduce the dual variable to ADMM, and simply updates at each iteration:

$$W_j^{(t+1)} = \arg\min_W \left\{ f_j(W) + \lambda \|W - W^{(t)}\|_2^2 \right\}$$

then does an averaging:

$$W^{(t+1)} = \frac{1}{m} \sum_{j \in [m]} W_j^{(t+1)}$$

We will show that this update will not work and that ADMM requires the dual variable for convergence. Consider a one dimensional function $f(w) = f_1(w) + f_2(w)$ where

$$f_1(w) = 2(w-1)^2 - w^2$$
$$f_2(w) = 4(w+1)^2 - w^2,$$

and take $\lambda = 1$ and the initial iterate to be $w^{(0)} = 0$. Show mathematically that this "naive" ADMM update provably does not converge to the global minimum of f.

3.2 Dual Solution (10 points)

Consider now the distributed optimization problem

$$\min_{\forall w_j, w} \frac{1}{m} \sum_{j=1}^{m} \left(f_j(w_j) + \lambda \|w_j - w\|_2^2 \right)$$
$$\text{subject to } w_j = w \ \forall j,$$

for f, λ as specified in the previous problem. We will now use the dual to minimize this objective. Use the KKT conditions to solve for the dual solutions α_j^* and the primal solution w^*. What does this imply about the optimal dual solution and optimal primal solution?

3.3 ADMM Implementation (20 points)

Now implement ADMM to solve the distributed optimization problem given in the previous question. Be sure to solve the inner minimization problems exactly with closed form expressions. Initialize $\alpha_j^{(0)} = 0, \ \forall j$. Use learning rate $\eta = 2\lambda$.

Plot $\alpha_j^{(t)}$ to show that they are converging to the α_j^* you predicted in Q3.2. And plot $w_j^{(t)}$ to show that they are reaching consensus toward your predicted w^*.

Just like in Q2.3, code in Python and submit your code in Gradescope. *Please write your code in a separate file from Q2.3, and label the pages as part of this question.*

4 Proximal algorithm

4.1 The Lasso proximal mapping (20 points)

Show that the proximal mapping of $h(x) = \lambda|x|$ for $\lambda > 0$ is given as:

$$\operatorname*{prox}_h(x) = \begin{cases} x - \lambda\mathsf{sgn}(x) & \text{if } |x| > \lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Where $\mathsf{sgn}(x) = 1$ if $x \geq 0$ and $\mathsf{sgn}(x) = -1$ otherwise. Hint: Given f convex but not necessarily differentiable, if $\nabla f(x) = 0$ exists, then f is optimal at x.

4.2 The proximal gradient descent (20 points)

Show that for $h(x) = \lambda\|x\|_1$, the proximal gradient update $x_{t+1} = \text{prox}_{\eta h}(x_t)$ will converge to the *exact* minimizer 0 of h in $O\left(\frac{\|x_0\|_2}{\eta\lambda}\right)$ iterations, for *any* choice of $\eta > 0$.

What about the gradient descent algorithm $x_{t+1} = x_t - \eta\nabla h(x_t)$? Does it converge to the *exact* minimizer? If so, give an informal proof or argument. If not, give an explicit example of x_0, η and λ such that the iterates provably don't converge to 0.