The Randomized Newton Method for Convex Optimization

Vaden Masrani

UBC MLRG

April 3rd, 2018

Introduction

We have some unconstrained, twice-differentiable convex function $f : \mathbb{R}^d \to \mathbb{R}$ that we want to minimize:

$$x^* = \operatorname*{argmin}_{x \in \mathbb{R}^d} f(x)$$

e.g. quadratic loss, logistic loss, log-sum-exp, etc

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n \times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n \times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$
- Gradient descent
  - Iteration cost: $O(nd)$, convergence rate: $O(\log(1/\epsilon))$:

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n\times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$
- Gradient descent
  - Iteration cost: $O(nd)$, convergence rate: $O(\log(1/\epsilon))$:
- Stochastic gradient descent
  - Iteration cost: $O(d)$, convergence rate: $O(1/\epsilon)$:

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n\times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$
- Gradient descent
  - Iteration cost: $O(nd)$, convergence rate: $O(\log(1/\epsilon))$:
- Stochastic gradient descent
  - Iteration cost: $O(d)$, convergence rate: $O(1/\epsilon)$:
- Newton's method
  - Iteration cost: $O(nd^2 + d^3)$, convergence rate: $O(\log(\log(1/\epsilon)))$:

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n \times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$
- Gradient descent
  - Iteration cost: $O(nd)$, convergence rate: $O(\log(1/\epsilon))$:
- Stochastic gradient descent
  - Iteration cost: $O(d)$, convergence rate: $O(1/\epsilon)$:
- Newton's method
  - Iteration cost: $O(nd^2 + d^3)$, convergence rate: $O(\log(\log(1/\epsilon)))$:
- Randomized newton's method
  - Iteration cost: $O(nd\log(m) + md^2)$, convergence rate: $O(\log(1/\epsilon))$
  - with $m \ll n$

Introduction

How might you do it? For example, consider minimizing l2-regularized least squares with data matrix $A \in \mathbb{R}^{n \times d}$

- Exact:
  - Iteration cost: $O(nd^2 + d^3)$
- Gradient descent
  - Iteration cost: $O(nd)$, convergence rate: $O(\log(1/\epsilon))$:
- Stochastic gradient descent
  - Iteration cost: $O(d)$, convergence rate: $O(1/\epsilon)$:
- Newton's method
  - Iteration cost: $O(nd^2 + d^3)$, convergence rate: $O(\log(\log(1/\epsilon)))$:
- Randomized newton's method
  - Iteration cost: $O(nd \log(m) + md^2)$, convergence rate: $O(\log(1/\epsilon))$
  - with $m \ll n$
- Many, many other ways...

Newton's method

- We start with a derivation of the standard newton method.
- Assume we're at some iteration x^t
- We find x^{x+1} by minimizing the second-order taylor expansion $f(x) \approx g(x)$ around x^t:

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}}\ g(x)$$

- Where:
  - $g(x) = f(x^t) + \nabla f(x^t)^T(x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$
  - $H(x^t) = \nabla^2 f(x^t)$

Newton's method

- What happens if we ignore curvature information and set $H(x) = \frac{1}{\alpha_t} I$?

Newton's method

- What happens if we ignore curvature information and set $H(x) = \frac{1}{\alpha_t} I$?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\alpha_t} ||(x - x^t)||_2^2$

Newton's method

- What happens if we ignore curvature information and set $H(x) = \frac{1}{\alpha_t} I$?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\alpha_t} ||(x - x^t)||_2^2$
- Take derivative w.r.t x, set to zero:

Newton's method

- What happens if we ignore curvature information and set $H(x) = \frac{1}{\alpha_t} I$?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\alpha_t} ||(x - x^t)||_2^2$
- Take derivative w.r.t x, set to zero:

$$\begin{aligned}
\frac{\partial g(x)}{\partial x} &= 0 = \nabla f(x^t) + \frac{1}{\alpha_t}(x - x^t) \\
-\frac{1}{\alpha_t}(x - x^t) &= \nabla f(x^t) \\
x^{t+1} &= x^t - \alpha_t \nabla f(x^t)
\end{aligned}$$

Newton's method

- What happens if we ignore curvature information and set $H(x) = \frac{1}{\alpha_t} I$?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\alpha_t} ||(x - x^t)||_2^2$
- Take derivative w.r.t x, set to zero:

$$
\begin{aligned}
\frac{\partial g(x)}{\partial x} &= 0 = \nabla f(x^t) + \frac{1}{\alpha_t}(x - x^t) \\
-\frac{1}{\alpha_t}(x - x^t) &= \nabla f(x^t) \\
x^{t+1} &= x^t - \alpha_t \nabla f(x^t)
\end{aligned}
$$

- We recover gradient descent

Newton's method

- ... and if we use curvature information?

Newton's method

- ... and if we use curvature information?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$

Newton's method

- ... and if we use curvature information?
- $g(x) = f(x^t) + \nabla f(x^t)^T(x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$
- Again take derivative w.r.t x, set to zero:

Newton's method

- ... and if we use curvature information?
- $g(x) = f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$
- Again take derivative w.r.t x, set to zero:

$$
\begin{aligned}
\frac{\partial g(x)}{\partial x} &= 0 = \nabla f(x^t) + H(x^t)(x - x^t) \\
-H(x^t)(x - x^t) &= \nabla f(x^t) \\
x^{t+1} &= x^t - [H(x^t)]^{-1} \nabla f(x^t)
\end{aligned}
$$

Newton's method

- ... and if we use curvature information?
- $g(x) = f(x^t) + \nabla f(x^t)^T(x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$
- Again take derivative w.r.t x, set to zero:

$$
\begin{aligned}
\frac{\partial g(x)}{\partial x} &= 0 = \nabla f(x^t) + H(x^t)(x - x^t) \\
-H(x^t)(x - x^t) &= \nabla f(x^t) \\
x^{t+1} &= x^t - [H(x^t)]^{-1}\nabla f(x^t)
\end{aligned}
$$

- We get newton's method

Randomized newton's method

- Newton's method converges in superlinear time
- But Newton's method requires inverting the hessian, which is prohibitively expensive for large datasets
  - Have to solve linear system $Hx = \nabla f(x^t)$ at each iteration
- How does SGD reduce the cost per iteration?
  - Replace gradient with random vector d_t s.t. $E[d_t] = \nabla f(x^t)$:
    - $x^{t+1} = x^t - \alpha_t d_t$
- How does randomized newton reduce the cost per iteration?
  - Replace hessian with random matrix D_t s.t. $E[d_t] = H(x^t)$
    - $x^{t+1} = x^t - D_t^{-1} \nabla f(x^t)$
- Parallelization is trivial modification

Digression: Matrix Sketches

- In order to formalize the randomized newton method, we need to establish the concept of "matrix sketches".
- One can sample the rows of a matrix $A \in \mathbb{R}^{n \times d}$ by forming a random "sketch" matrix $S \in \mathbb{R}^{m \times n}$
  - Many variations are available
  - This work uses S s.t. $E[S] = 0$ and $E[S^T S] = I_n$
  - Using some tricks, SA can be formed in $O(nd \log m)$[1]

[1] See section 2.2 in Pilanci and Wainwright

Sketch example: Random row sampling

- Given a probability distribution $\{p_j\}_{j=1}^{n}$ over rows n = $\{1, 2, ..., n\}$, form S by sampling m rows with replacements, where each row i = $\{1, 2, ..., m\}$ takes on the value:

$$s_i^T = \frac{e_j}{\sqrt{p_j}}$$

- Sample (scaled) rows of A by matrix product SA

Sketch example: Random row sampling

$$S = \begin{bmatrix} 0 & 0 & \frac{1}{\sqrt{p_3}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{p_3}} & 0 \\ \frac{1}{\sqrt{p_1}} & 0 & 0 & 0 \end{bmatrix} \quad A = \begin{bmatrix} -a_1^T- \\ -a_2^T- \\ -a_3^T- \\ -a_4^T- \end{bmatrix}$$

$$SA = \begin{bmatrix} \frac{1}{\sqrt{p_3}} * a_3^T \\ \frac{1}{\sqrt{p_3}} * a_3^T \\ \frac{1}{\sqrt{p_1}} * a_1^T \end{bmatrix}$$

Back to randomized newton

- We will use a sketch matrix S to form a random vector d_t s.t $E[d_t] = H(x^t)$
- Many ways to do this, we will use the *Newton sketch* algorithm of Pilanci and Wainwright[2].
  - We are in the regime where $n > d$
  - I'll focus on the unconstrained case, but their work extends to constrained minimization as well.

[2] Mert Pilanci and Martin J Wainwright. "Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence". In: *arXiv preprint arXiv:1505.02250* (2015).

Randomized newton

Recall our setup:

$$x^{t+1} = \operatorname*{argmin}_{x \in \mathbb{R}^d} f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$$

Randomized newton

Recall our setup:

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$$

Now supposed we have some hessian square root matrix $L \in \mathbb{R}^{n \times d}$ ie. $L^T L = H(x)$

- Ex: Consider $f(x) = g(Ax)$, where $g : \mathbb{R}^n \to \mathbb{R}$ has the separable form $g(Ax) = \sum_i^n g_i(a_i^T x)$. In this case, $L = diag\{g_i''(a_i^T x)\}_{i=1}^n A$
- Pilanci and Wainwright give examples of L for linear program, GLMs, linear/logistic regression

Randomized newton

Then standard newton becomes:

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$$

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}||L(x - x^t)||_2^2$$

Randomized newton

Then standard newton becomes:

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \ f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}(x - x^t)^T H(x^t)(x - x^t)$$

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \ f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}||L(x - x^t)||_2^2$$

To randomized:

$$x^{t+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \ f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2}||S_t L(x - x^t)||_2^2$$

Where $S_t \in \mathbb{R}^{m \times n}$ is an independent realization of a sketching matrix at iteration t.

Randomized newton

Solving for x:

$$x^{t+1} = x^t - D_t^{-1} \nabla f(x^t)$$
$$D_t = L^T S_t^T S_t L$$

We see:

$$E[D_t] = E[L^T S_t^T S_t L] = L^T E[S_t^T S_t] L = L^T L = H(x)$$

Each step of the newton sketch algorithm can be computed in $O(md^2)$ using conjugate gradient instead of $O(nd^2)$ of standard newton.

Convergence

If m is chosen to satisfy certain conditions[3], the unconstrained newton sketch algorithm achieves linear convergence:

$$f(x^t) - f(x^*) \leq \frac{\beta\gamma}{8L}(\frac{1}{2} + \epsilon\frac{\beta}{\gamma})^t$$

Where $\beta = \lambda_{\text{min}}(H(x^*))$, $\gamma = \lambda_{\text{max}}(H(x^*))$ and we assume the hessian is Lipschitz continuous, i.e. $||H(x) - H(y)|| \leq L||x - y||_2$.

[3] See eq'n 12 in Pilanci and Wainwright.

Parallel

Extending the newton sketch algorithm to the parallel setting is trivial. See for example "Parallel Stochastic Newton Method"[4] for convergence results:

Algorithm 1 PSN: Parallel Stochastic Newton Method

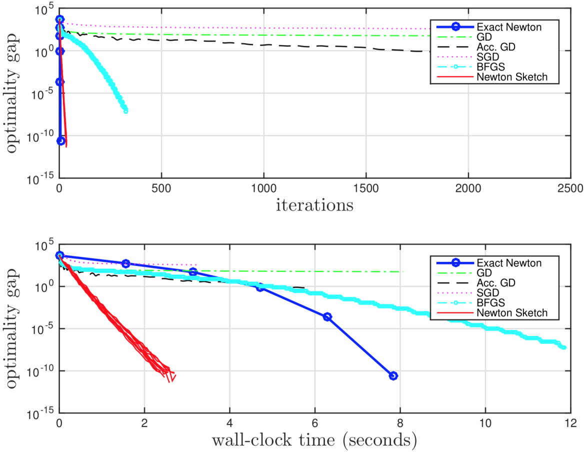
Parameters: sampling $\hat{S}$; data matrix $\mathbf{M}$; aggregation parameter b
Initialization: Pick $x^0 \in \mathbb{R}^n$

1: **for** $k = 0, 1, 2, \ldots$ **do**
2:     **for** $i = 1, \ldots, c$ in parallel **do**
3:         Independently generate a random set $\hat{S}_i^k \sim \hat{S}$
4:         $h_i^k \leftarrow \left(\mathbf{M}_{\hat{S}_i^k}\right)^{-1} \nabla f(x^k)$
5:     **end for**
6:     $x^{k+1} \leftarrow x^k - \frac{1}{b} \sum_{i=1}^{c} h_i^k$
7: **end for**

---

[4] Mojmír Mutnỳ and Peter Richtárik. "Parallel Stochastic Newton Method". In: *arXiv preprint arXiv:1705.02005* (2017).

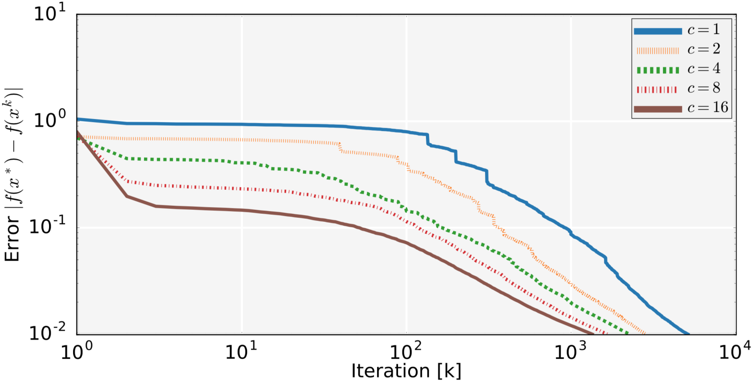
Sequential results[5]



Logistic regression, $d = 100$, $n = 16384$, $m = 6d$

[5] Pilanci and Wainwright, "Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence".

Parallel results[6]



Linear regression, synthetic data, $d = n = 10^3, m = 3, c =$ number of processors

[6] Mutný and Richtárik, "Parallel Stochastic Newton Method".

Mutný, Mojmír and Peter Richtárik. "Parallel Stochastic Newton Method". In: *arXiv preprint arXiv:1705.02005* (2017).

Pilanci, Mert and Martin J Wainwright. "Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence". In: *arXiv preprint arXiv:1505.02250* (2015).