Stochastic Gradient Descent

Hoàng Nam Dũng

Khoa Toán - Cơ - Tin học, Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội

Last time: proximal gradient descent

Consider the problem

$$\min_x g(x) + h(x)$$

with g, h convex, g differentiable, and h "simple" in so much as

$$\text{prox}_t(x) = \text{argmin}_z \frac{1}{2t}\|x - z\|_2^2 + h(z)$$

is computable.

Proximal gradient descent: let $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = \text{prox}_{t_k}(x^{(k-1)} - t_k \nabla g(x^{(k-1)})), \quad k = 1, 2, 3, \ldots$$

Step sizes t_k chosen to be fixed and small, or via backtracking.

If ∇g is Lipschitz with constant L, then this has convergence rate $O(1/\varepsilon)$. Lastly we can accelerate this, to optimal rate $O(1/\sqrt{\varepsilon})$.

Outline

Today:

- Stochastic gradient descent
- Convergence rates
- Mini-batches
- Early stopping

Stochastic gradient descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x).$$

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent would repeat

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \dots$$

In comparison, stochastic gradient descent or SGD (or incremental gradient descent) repeats:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, \dots$$

where $i_k \in \{1, \dots, m\}$ is some chosen index at iteration k.

Stochastic gradient descent

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, ...m\}$ uniformly at random.
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Randomized rule is more common in practice. For randomized rule, note that

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x),$$

so we can view SGD as using an unbiased estimate of the gradient at each step.

Main appeal of SGD:

- Iteration cost is independent of m (number of functions).
- Can also be a big savings in terms of memory usage.

Example: stochastic logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$, recall logistic regression

$$\min_\beta f(\beta) = \frac{1}{n} \sum_{i=1}^n \underbrace{\left(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta))\right)}_{f_i(\beta)}.$$

Gradient computation $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - p_i(\beta)) x_i$ is doable when n is moderate, but not when n is huge.

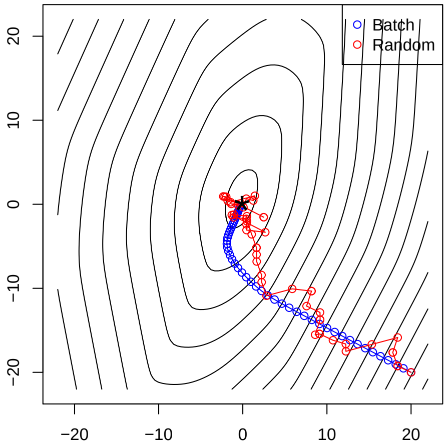
Full gradient (also called batch) versus stochastic gradient:

- One batch update costs $O(np)$.
- One stochastic update costs $O(p)$.

Clearly, e.g., 10K stochastic steps are much more affordable.

Batch vs. stochastic gradient descent

Small example with $n = 10, p = 2$ to show the "classic picture" for batch versus stochastic methods:



Blue: batch steps, $O(np)$

Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

Step sizes

Standard in SGD is to use diminishing step sizes, e.g., $t_k = 1/k$, for $k = 1, 2, 3, \ldots$

Why not fixed step sizes? Here's some intuition.

Suppose we take cyclic rule for simplicity. Set $t_k = t$ for m updates in a row, we get

$$x^{(k+m)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k+i-1)}).$$

Meanwhile, full gradient with step size t would give

$$x^{(k+1)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k)}).$$

The difference here: $t \sum_{i=1}^{m} [\nabla f_i(x^{(k+i-1)}) - \nabla f_i(x^{(k)})]$, and if we hold t constant, this difference will not generally be going to zero.

Convergence rates

Recall: for convex f, (sub)gradient descent with diminishing step sizes satisfies

$$f(x^{(k)}) - f^* = O(1/\sqrt{k}).$$

When f is differentiable with Lipschitz gradient, there holds for gradient descent with suitable fixed step sizes

$$f(x^{(k)}) - f^* = O(1/k).$$

What about SGD? For convex f, SGD with diminishing step sizes satisfies[1]

$$\mathbb{E}[f(x^{(k)})] - f^* = O(1/\sqrt{k}).$$

Unfortunately this does not improve when we further assume f has Lipschitz gradient.

[1] E.g., Nemirosvki et al. (2009), "Robust stochastic optimization approach to stochastic programming"

Convergence rates

Even worse is the following discrepancy!

When f is strongly convex and has a Lipschitz gradient, gradient descent satisfies

$$f(x^{(k)}) - f^* = O(c^k)$$

where $c < 1$. But under same conditions, SGD gives us[2]

$$\mathbb{E}[f(x^{(k)})] - f^* = O(1/k).$$

So stochastic methods do not enjoy the linear convergence rate of gradient descent under strong convexity.

What can we do to improve SGD?

[2]E.g., Nemirosvki et al. (2009), "Robust stochastic optimization approach to stochastic programming"

Mini-batches

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, \ldots, m\}$ of size $|I_k| = b \ll m$ and repeat

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Again, we are approximating full gradient by an unbiased estimate

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\right] = \nabla f(x).$$

Using mini-batches reduces the variance of our gradient estimate by a factor $1/b$, but is also b times more expensive.

Batch vs. mini-batches vs. stochastic

Back to logistic regression, let's now consider a regularized version:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \Big(-y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \Big) + \frac{\lambda}{2} \|\beta\|_2^2.$$

Write the criterion as

$$f(\beta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\beta), \quad f_i(\beta) = -y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) + \frac{\lambda}{2} \|\beta\|_2^2.$$

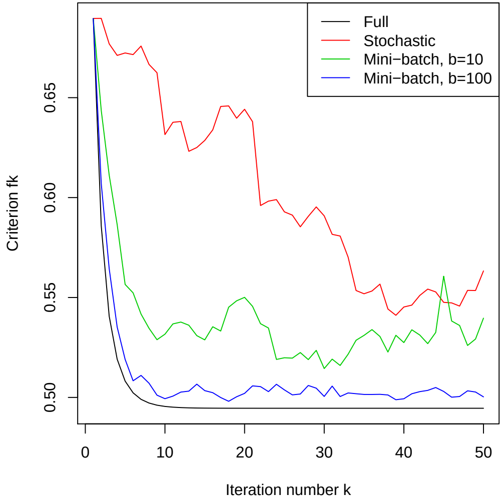
Full gradient computation is $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i + \lambda \beta$.

Comparison between methods:

- One batch update costs $O(np)$.
- One mini-batch update costs $O(bp)$.
- One stochastic update costs $O(p)$.

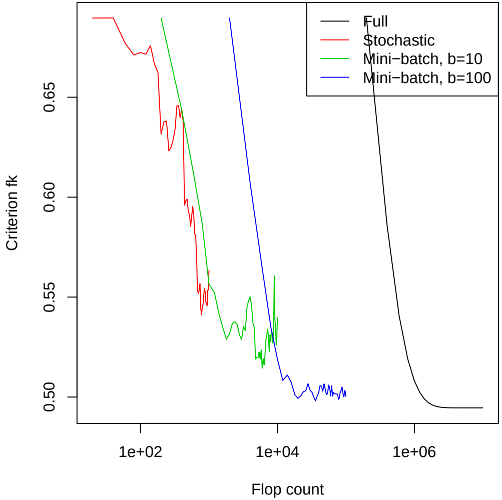
Batch vs. mini-batches vs. stochastic

Example with $n = 10,000, p = 20$, all methods use fixed step sizes



Batch vs. mini-batches vs. stochastic

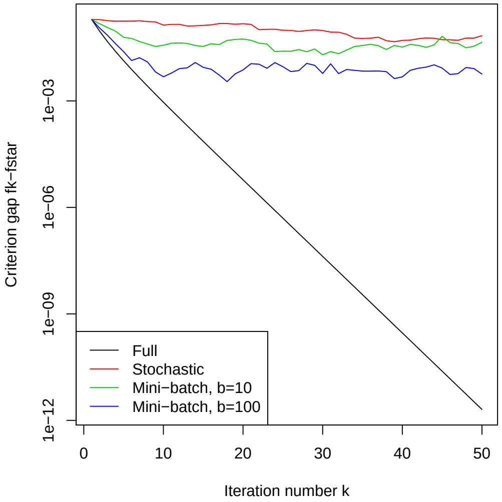
What's happening? Now let's parametrize by flops[3]



[3]flops: floating point operations per second

Batch vs. mini-batches vs. stochastic

Finally, looking at suboptimality gap (on log scale)



End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory.
- But SGD is slow to converge, can't adapt to strong convexity.
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice).

End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory.
- But SGD is slow to converge, can't adapt to strong convexity.
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice).

Is this the end of the story for SGD?

End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory.
- But SGD is slow to converge, can't adapt to strong convexity.
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice).

Is this the end of the story for SGD?

For a while, the answer was believed to be yes. Slow convergence for strongly convex functions was believed inevitable, as Nemirovski and others established matching lower bounds ... but this was for a more general stochastic problem, where $f(x) = \int F(x, \zeta) dP(\zeta)$.

New wave of "variance reduction" work shows we can modify SGD to converge much faster for finite sums (more later?).

SGD in large-scale ML

SGD has really taken off in large-scale machine learning

- In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance.
- Thus (in contrast to what classic theory says) fixed step sizes are commonly used in ML applications.
- One trick is to experiment with step sizes using small fraction of training before running SGD on full data set ... many other heuristics are common[4].
- Many variants provide better practical stability, convergence: momentum, acceleration, averaging, coordinate-adapted step sizes, variance reduction ...
- See AdaGrad, Adam, AdaMax, SVRG, SAG, SAGA ... (more later?).

[4]E.g., Bottou (2012), "Stochastic gradient descent tricks"

Early stopping

Suppose p is large and we wanted to fit (say) a logistic regression model to data $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\},\ i = 1, \ldots, n$.

We could solve (say) ℓ_2 regularized logistic regression

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n \left(-y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \right) \text{ subject to } \|\beta\|_2 \le t$$

We could also run gradient descent on the unregularized problem

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n \left(-y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \right)$$

and stop early, i.e., terminate gradient descent well-short of the global minimum.

Early stopping

Consider the following, for a very small constant step size ε:

- Start at $\beta^{(0)} = 0$, solution to regularized problem at $t = 0$.
- Perform gradient descent on unregularized criterion

  $$\beta^{(k)} = \beta^{(k-1)} - \varepsilon \cdot \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i(\beta^{(k-1)}))x_i, \quad k = 1, 2, 3, \dots$$

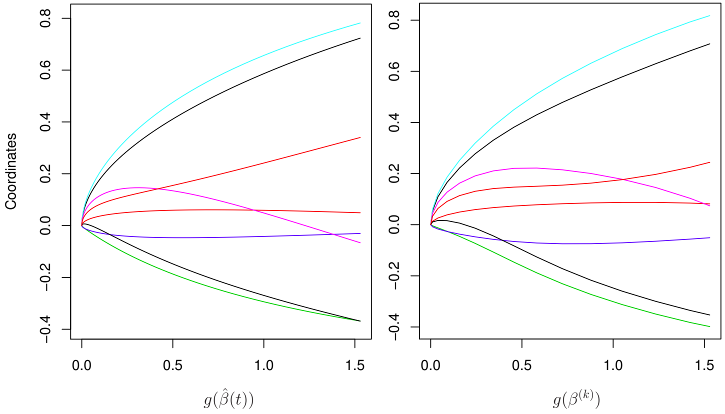
  (we could equally well consider SGD).
- Treat $\beta^{(k)}$ as an approximate solution to regularized problem with $t = \|\beta^{(k)}\|_2$.

This is called early stopping for gradient descent. Why would we ever do this? It's both more convenient and potentially much more efficient than using explicit regularization.

An intriguing connection

When we solve the ℓ_2 regularized logistic problem for varying t, solution path looks quite similar to gradient descent path!

Example with $p = 8$, solution and grad descent paths side by side



Lots left to explore

- Connection holds beyond logistic regression, for arbitrary loss.
- In general, the grad descent path will not coincide with the ℓ_2 regularized path (as $\varepsilon \to 0$). Though in practice, it seems to give competitive statistical performance.
- Can extend early stopping idea to mimick a generic regularizer (beyond ℓ_2)[5].
- There is a lot of literature on early stopping, but it's still not as well-understood as it should be.
- Early stopping is just one instance of implicit or algorithmic regularization, many others are effective in large-scale ML, they all should be better understood

[5] Tibshirani (2015), "A general framework for fast stagewise algorithms"

References and further reading

- D. Bertsekas (2010), *Incremental gradient, subgradient, and proximal methods for convex optimization: a survey*
- A. Nemirovski and A. Juditsky and G. Lan and A. Shapiro (2009), *Robust stochastic optimization approach to stochastic programming*
- R. Tibshirani (2015), *A general framework for fast stagewise algorithms*