Simple Techniques for Improving SGD

CS6787 Lecture 2 — Fall 2018

Step Sizes and Convergence

Where we left off

- Stochastic gradient descent (with fixed learning rate)

$$w_{t+1} = w_t - \alpha \nabla f(w_t; x_{i_t})$$

- Much faster per iteration than gradient descent
  - Because we don't have to process the entire training set

- But converges to a noise ball (for strongly convex problems)

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha \mu^2}$$

Controlling the accuracy

- Want the noise ball to be as small as possible for accurate solutions

- Noise ball **proportional to the step size/learning rate**

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2 \right] \le \frac{\alpha M}{2\mu - \alpha \mu^2} = O(\alpha)$$

- So **should we make the step size as small as possible?**

Effect of step size on convergence

- Let’s go back to the convergence rate proof for SGD
  - From the previous lecture, we have

$$\mathbf{E}\left[\|w_{t+1} - w^*\|^2\right] \le (1 - \alpha\mu^2)\mathbf{E}\left[\|w_t - w^*\|^2\right] + \alpha^2 M.$$

- If we’re far from the noise ball i.e. $\mathbf{E}\left[\|w_t - w^*\|^2\right] \ge \frac{2\alpha M}{\mu}$

$$\mathbf{E}\left[\|w_{t+1} - w^*\|^2\right] \le (1 - \alpha\mu^2)\mathbf{E}\left[\|w_t - w^*\|^2\right] + \frac{\alpha\mu}{2}\mathbf{E}\left[\|w_t - w^*\|^2\right].$$

Effect of step size on convergence (continued)

$$\begin{aligned}
\mathbf{E}\left[\|w_{t+1} - w^*\|^2\right] &\le (1 - \alpha\mu^2)\mathbf{E}\left[\|w_t - w^*\|^2\right] + \frac{\alpha\mu}{2}\mathbf{E}\left[\|w_t - w^*\|^2\right] \\
&\le \left(1 - \frac{\alpha\mu}{2}\right)\mathbf{E}\left[\|w_t - w^*\|^2\right] \qquad\qquad \text{if } \alpha\mu < 1 \\
&\le \exp\left(-\frac{\alpha\mu}{2}\right)\mathbf{E}\left[\|w_t - w^*\|^2\right].
\end{aligned}$$

- So to contract by a factor of **C**, we need to run **T** steps, where

$$1 = \exp\left(-\frac{\alpha\mu T}{2}\right) C \Leftrightarrow T = \frac{2}{\alpha\mu}\log C$$

The Full Effect of Step Size

- Noise ball **proportional to the step size**

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2} = O(\alpha)$$

- Convergence time **inversely proportional to the step size**

$$T = \frac{2}{\alpha\mu} \log C$$

- So **there's a trade-off!**

Demo

Can we get the best of both worlds?

- When do we want the step size to be large?
    - **At the beginning of execution!** Near the end? Both?

- When do we want the step size to be small?
    - At the beginning of execution? **Near the end!** Both?

- What about using a **decreasing step size scheme?**

SGD with Varying Step Size

- Allow the step size to vary over time

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t; y_{i_t})$$

- Turns out this is the standard in basically all machine learning!

- Two ways to do it:
  - **Chosen a priori step sizes** — step size doesn't depend on measurements
  - **Adaptive step sizes** — choose step size based on measurements & heuristics

Optimal Step Sizes for Convex Objectives

- Can we use math to choose a step size?
  - Start with our previous bound

$$\begin{aligned}\mathbf{E}\left[\|w_{t+1} - w^*\|^2\right] &\le (1 - \alpha_t \mu)^2 \mathbf{E}\left[\|w_t - w^*\|^2\right] + \alpha_t^2 M \\ &\le (1 - \alpha_t \mu) \mathbf{E}\left[\|w_t - w^*\|^2\right] + \alpha_t^2 M \quad (\text{for } \alpha_t \mu < 1)\end{aligned}$$

- Right side is minimized when

$$0 = -\mu \mathbf{E}\left[\|w_t - w^*\|^2\right] + 2\alpha_t M \Leftrightarrow \alpha_t = \frac{\mu}{2M} \mathbf{E}\left[\|w_t - w^*\|^2\right]$$

Optimal Step Sizes (continued)

Let $\rho_t = \mathbf{E}\left[\|w_t - w^*\|^2\right]$

$$\begin{aligned}
\rho_{t+1} &\le (1 - \alpha_t \mu)\,\rho_t + \alpha_t^2 M \\
&= \left(1 - \left(\frac{\mu}{2M}\rho_t\right)\mu\right)\rho_t + \left(\frac{\mu}{2M}\rho_t\right)^2 M \\
&= \rho_t - \frac{\mu^2}{2M}\rho_t^2 + \frac{\mu^2}{4M}\rho_t^2 \\
&= \rho_t - \frac{\mu^2}{4M}\rho_t^2
\end{aligned}$$

Optimal Step Sizes (continued)

Let $\rho_t = \mathbf{E}\left[\|w_t - w^*\|^2\right]$

$$
\begin{aligned}
\frac{1}{\rho_{t+1}} &\geq \left(\rho_t - \frac{\mu^2}{4M}\rho_t^2\right)^{-1} \\
&= \frac{1}{\rho_t}\left(1 - \frac{\mu^2}{4M}\rho_t\right)^{-1} \\
\text{(since } (1-z)^{-1} \geq 1 + z\text{)} \qquad &\geq \frac{1}{\rho_t}\left(1 + \frac{\mu^2}{4M}\rho_t\right) \\
&= \frac{1}{\rho_t} + \frac{\mu^2}{4M}.
\end{aligned}
$$

Optimal Step Sizes (continued)

Let $\rho_t = \mathbf{E}\left[\|w_t - w^*\|^2\right]$

$$\frac{1}{\rho_T} \geq \frac{1}{\rho_0} + \frac{\mu^2 T}{4M} \Rightarrow \rho_T \leq \frac{4M\rho_0}{4M + \mu^2 \rho_0 T}$$

- Sometimes called a **1/T rate.**
  - Slower than the linear rate of gradient descent, for convex problems.

Optimal Step Sizes (continued)

- Substitute back in to find how to set the step size:

$$\alpha_t = \frac{\mu}{2M} \cdot \frac{4M\rho_0}{4M + \mu^2 \rho_0 t} = \frac{2\mu\rho_0}{4M + \mu^2 \rho_0 t} = \frac{1}{\Theta(t)}$$

- This is a pretty common simple scheme
  - General form is

$$\alpha_t = \frac{\alpha_0}{1 + \gamma t}$$

Demo

Have we solved step sizes for SGD forever?

- **No.**

- We don't usually know what μ, L, M, and ρ_0 are
  - Even if the problem is convex

- This "optimal" rate **optimizes the upper bound** on the expected distance-squared to the optimum
  - But sometimes this bound is loose, and other step size schemes might do better

What if we don't know the parameters?

- One idea: still use a step size scheme of the form

$$\alpha_t = \frac{\alpha_0}{1 + \gamma t}$$

- Choose parameters α_0 and t via some other method
  - For example, **hand-tuning** — which doesn't scale

- Can we do this automatically?
  - **Yes!** This is an example of **hyperparameter optimization**

Other Techniques

- Decrease the step size in **epochs**
  - Still asymptotically $\alpha_t = \frac{1}{\Theta(t)}$ but step size decreases in discrete steps
  - Useful for parallelization and saves a little compute
- **Per-parameter** learning rates — e.g. AdaGrad
  - Replace scalar α with diagonal matrix **A**.

$$x_{t+1} = x_t - A \nabla f(x_t; y_{i_t})$$

  - Helps with poorly scaled problems

Mini-Batching

Gradient Descent vs. SGD

- Gradient descent: **all examples at once**

$$w_{t+1} = w_t - \alpha_t \frac{1}{N} \sum_{i=1}^{N} \nabla f(w_t; x_i)$$

- Stochastic gradient descent: **one example at a time**

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t; x_{i_t})$$

- Is it really **all or nothing**? Can we do something intermediate?

Mini-Batch Stochastic Gradient Descent

- An intermediate approach

$$w_{t+1} = w_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(w_t; x_i)$$

where B_t is sampled uniformly from the set of all subsets of {1, … , N} of size b.

  - The b parameter is the **batch size**
  - Typically choose b << N.

- Also called **mini-batch gradient descent**

How does runtime cost of Mini-Batch compare to SGD and Gradient Descent?

- Takes **less time to compute each update** than gradient descent
    - Only needs to sum up b gradients, rather than N

$$w_{t+1} = w_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(w_t; x_i)$$

- But takes **more time for each update** than SGD
    - So what's the benefit?

- It's more like gradient descent, so **maybe it converges faster** than SGD?

Mini-Batch SGD Converges

- Start by breaking up the update rule into expected update and noise

$$\begin{aligned} w_{t+1} - w^* = w_t - w^* - \alpha_t \left(\nabla h(w_t) - \nabla h(w^*) \right) \\ - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(w_t; x_i) - \nabla h(w_t) \right) \end{aligned}$$

- Second moment bound

$$\begin{aligned} \mathbf{E}\left[\| w_{t+1} - w^* \|^2 \right] = \mathbf{E}\left[\| w_t - w^* - \alpha_t \left(\nabla h(w_t) - \nabla h(w^*) \right) \|^2 \right] \\ + \alpha_t^2 \mathbf{E}\left[\left\| \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(w_t; x_i) - \nabla h(w_t) \right) \right\|^2 \right] \end{aligned}$$

Mini-Batch SGD Converges (continued)

Let $\Delta_i = \nabla f(w_t; x_i) - \nabla h(w_t)$

$$\mathbf{E}\left[\left\| \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(w_t; x_i) - \nabla h(w_t) \right) \right\|^2\right]$$

$$= \mathbf{E}\left[\left\| \frac{1}{|B_t|} \sum_{i \in B_t} \Delta_i \right\|^2\right] = \frac{1}{|B_t|^2} \mathbf{E}\left[\left\| \sum_{i=1}^{N} \beta_i \Delta_i \right\|^2\right]$$

$$= \frac{1}{|B_t|^2} \mathbf{E}\left[\left(\sum_{i=1}^{N} \beta_i \Delta_i \right)^T \left(\sum_{j=1}^{N} \beta_j \Delta_j \right)\right] = \frac{1}{|B_t|^2} \mathbf{E}\left[\sum_{i=1}^{N} \sum_{j=1}^{N} \beta_i \beta_j \Delta_i^T \Delta_j \right]$$

Mini-Batch SGD Converges (continued)

- Because we sampled B uniformly at random, for **i ≠ j**

$$\mathbf{E}\left[\beta_i \beta_j\right] = \mathbf{P}\left(i \in B \wedge j \in B\right) = \mathbf{P}\left(i \in B\right) \mathbf{P}\left(j \in B | i \in B\right) = \frac{b}{N} \cdot \frac{b-1}{N-1}$$

$$\mathbf{E}\left[\beta_i^2\right] = \mathbf{P}\left(i \in B\right) = \frac{b}{N}$$

- So we can bound our square error term as

$$\mathbf{E}\left[\left\| \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(w_t; x_i) - \nabla h(w_t)\right) \right\|^2\right] = \frac{1}{|B_t|^2} \mathbf{E}\left[\sum_{i=1}^N \sum_{j=1}^N \beta_i \beta_j \Delta_i^T \Delta_j\right]$$

$$= \frac{1}{b^2} \mathbf{E}\left[\sum_{i \neq j} \frac{b(b-1)}{N(N-1)} \Delta_i^T \Delta_j + \sum_{i=1}^N \frac{b}{N} \|\Delta_i\|^2\right]$$

Mini-Batch SGD Converges (continued)

$$\mathbf{E}\left[\left\|\frac{1}{|B_t|}\sum_{i\in B_t}\left(\nabla f(w_t;x_i)-\nabla h(w_t)\right)\right\|^2\right]=\frac{1}{bN}\mathbf{E}\left[\frac{b-1}{N-1}\sum_{i\neq j}\Delta_i^T\Delta_j+\sum_{i=1}^{N}\|\Delta_i\|^2\right]$$

$$=\frac{1}{bN}\mathbf{E}\left[\frac{b-1}{N-1}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta_i^T\Delta_j+\sum_{i=1}^{N}\left(1-\frac{b-1}{N-1}\right)\|\Delta_i\|^2\right]$$

$$=\frac{1}{bN}\mathbf{E}\left[0+\sum_{i=1}^{N}\left(1-\frac{b-1}{N-1}\right)\|\Delta_i\|^2\right]=\frac{N-b}{bN(N-1)}\mathbf{E}\left[\sum_{i=1}^{N}\|\Delta_i\|^2\right]$$

Mini-Batch SGD Converges (continued)

$$\mathbf{E}\left[\left\|\frac{1}{|B_t|}\sum_{i\in B_t}(\nabla f(w_t;x_i)-\nabla h(w_t))\right\|^2\right]=\frac{N-b}{b(N-1)}\mathbf{E}\left[\frac{1}{N}\sum_{i=1}^{N}\|\Delta_i\|^2\right]$$

$$=\frac{N-b}{b(N-1)}\mathbf{E}\left[\frac{1}{N}\sum_{i=1}^{N}\|\nabla f(w_t;x_i)-\nabla h(w_t)\|^2\right]$$

$$\le\frac{N-b}{b(N-1)}\cdot M$$

$$\le\frac{M}{b}$$

- Compared with SGD, **squared error term decreased by a factor of b**

Mini-Batch SGD Converges (continued)

- Recall that SGD converged to a noise ball of size

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2}$$

- Since mini-batching decreases error term by a factor of **b**, it will have

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2\right] \leq \frac{\alpha M}{(2\mu - \alpha\mu^2)b}$$

  - **Noise ball smaller** by the same factor!

Advantages of Mini-Batch (reprise)

- Takes **less time to compute each update** than gradient descent
  - Only needs to sum up b gradients, rather than N

$$w_{t+1} = w_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(w_t; x_i)$$

- Converges to a **smaller noise ball** than stochastic gradient descent

$$\lim_{T \to \infty} \mathbf{E}\left[\|w_T - w^*\|^2\right] \leq \frac{\alpha M}{(2\mu - \alpha\mu^2)b}$$

How to choose the batch size?

- **Mini-batching is not a free win**
  - Naively, compared with SGD, it takes **b** times as much effort to get a **b**-times-as-accurate answer
  - But we could have gotten a **b**-times-as-accurate answer by just running SGD for **b** times as many steps with a step size of $\alpha/\mathbf{b}$.

- But it still makes sense to run it for **systems** and **statistical** reasons
  - Mini-batching exposes more parallelism
  - Mini-batching lets us estimate statistics about the full gradient more accurately

- Another use case for **metaparameter optimization**

Mini-Batch SGD is very widely used

- Including in basically all neural network training

- **b = 32** is a typical default value for batch size
    - From “Practical Recommendations for Gradient-Based Training of Deep Architectures,” Bengio 2012.

Overfitting, Generalization Error, and Regularization

Minimizing Training Loss is Not our Real Goal

- Training loss looks like

$$h(w) = \frac{1}{N} \sum_{i=1}^{N} f(w; x_i)$$

- What we actually want to minimize is **expected loss on new examples**
  - Drawn from some real-world distribution ϕ

$$\bar{h}(w) = \mathbf{E}_{x \sim \phi} \left[f(w; x) \right]$$

  - Typically, assume the training examples were drawn from this distribution

Overfitting

- Minimizing the training loss **doesn't generally minimize the expected loss** on new examples
    - They are two different objective functions after all

- Difference between the empirical loss on the training set and the expected loss on new examples is called the **generalization error**

- Even a model that has high accuracy on the training set can have terrible performance on new examples
    - Phenomenon is called **overfitting**

Demo

How to address overfitting

- **Many, many techniques** to deal with overfitting
  - Have varying computational costs

- But this is a systems course…so what can we do **with little or no extra computational cost?**

- Notice from the demo that **some loss functions do better than others**
  - Can we **modify our loss function** to prevent overfitting?

Regularization

- Add an extra **regularization term** to the objective function

- Most popular type: **L2 regularization**

$$h(w) = \frac{1}{N}\sum_{i=1}^{N} f(w; x_i) + \sigma^2 \|w\|_2^2 = \frac{1}{N}\sum_{i=1}^{N} f(w; x_i) + \sigma^2 \sum_{k=1}^{d} x_k^2$$

- Also popular: **L1 regularization**

$$h(w) = \frac{1}{N}\sum_{i=1}^{N} f(w; x_i) + \gamma \|w\|_1 = \frac{1}{N}\sum_{i=1}^{N} f(w; x_i) + \gamma \sum_{k=1}^{d} \|x_k\|$$

Benefits of Regularization

- **Cheap to compute**
  - For SGD and L2 regularization, there's just an extra scaling

$$w_{t+1} = (1 - 2\alpha_t \sigma^2) w_t - \alpha_t \nabla f(w_t; x_{i_t})$$

- **L2 regularization makes the objective strongly convex**
  - This makes it easier to get and prove bounds on convergence

- **Helps with overfitting**

Demo

How to choose the regularization parameter?

- One way is to use an independent **validation set** to estimate the test error, and set the regularization parameter manually so that it is high enough to avoid overfitting
  - This is what we saw in the demo

- But doing this naively can be **computationally expensive**
  - Need to re-run learning algorithm many times

- Yet another use case for **hyperparameter optimization**

More general forms of regularization

- **Regularization** is used more generally to describe anything that helps prevent overfitting
  - By biasing learning by making some models more desirable *a priori*

- Many techniques that give throughput improvements also have a regularizing effect
  - Sometimes: a **win-win** of better statistical and hardware performance

Early Stopping

Asymptotically large training sets

- Setting 1: we have a distribution ϕ and we sample a very large (asymptotically infinite) number of points from it, then run stochastic gradient descent on that training set for only **N** iterations.

- Can our algorithm in this setting overfit?
    - **No, because its training set is asymptotically equal to the true distribution.**

- Can we compute this efficiently?
    - **No, because its training set is asymptotically infinitely large**

Consider a second setting

- Setting 1: we have a distribution ϕ and we sample a very large (asymptotically infinite) number of points from it, then run stochastic gradient descent on that training set for only **N** iterations.

- Setting 2: we have a distribution ϕ and we sample **N** points from it, then run stochastic gradient descent using each of these points exactly once.

- What is the difference between the output of SGD in these two settings?
  - **Asymptotically, there's no difference!**
  - So SGD in Setting 2 will also never overfit

Early Stopping

- Motivation: if we only use each training example once for SGD, then we can't overfit.

- So if we **only use each example a few times**, we probably won't overfit too much.

- **Early stopping**: just stop running SGD before it converges.

Benefits of Early Stopping

- **Cheap to compute**
  - Literally just does less work
  - It seems like the technique was designed to make systems run faster
- **Helps with overfitting**

How Early to Stop

- You'll see this in detail in one of **next Wednesday's papers**.
- Yet another application of **hyperparameter tuning**.

Questions?

- Upcoming things
  - Labor day next Monday — **no lecture**
  - Paper Presentation #1 **on Wednesday** — read paper before class