ECE 901: Large-scale Machine Learning and Optimization **Fall 2016**

Lecture 9 — October 4

Lecturer: Dimitris Papailiopoulos *Scribe: Kyle Daruwalla*

Note: These lecture notes are still rough, and have only have been mildly proofread.

9.1 Overview

This lecture covers the following topics:

1. Choosing step sizes for SGD
2. Non-differentiable functions and subgradients
3. Contraint sets and projected gradient descent

9.2 Choosing Step Sizes for SGD

Previous lectures covered optimal step sizes that were useful for analyzing convergence rates, but not practical for implementation. Choosing the step size is not as rigid in practice, and there are several approaches to consider.

While the majority of this section focuses on SGD, there are options for choosing the step size, γ, in regular gradient descent:

- *Line Search:* This is the most direct method, and it essentially tries to solve

$$\gamma_{k+1} = \arg\min_{\gamma} f(x_k - \gamma \nabla f(x_k))$$

  Unfortunately, this method requires f to be re-evaluated for each potential γ_{k+1}, which can be expensive.

- Another option is back-tracking or the Armijo method. This is a smarter version of line search that finds the correct γ more efficiently using the Armijo-Goldstein condition. A simple overview can be found here.

9.2.1 Random/Grid Search

Assume we are trying to find a constant step size, γ, and that the data is split into training, validation, and test sets. The process for grid search is:

1. Choose n random γ between the bounds of γ (remember we can get reasonable bounds for γ from our convergence rate analysis)
2. Train n different models on the training data
3. Keep the model (and the γ) that has the lowest loss on the validation data

Random/grid search is the most straight forward method for choosing γ, but its simplicity comes at a cost. Instead of training a single model, it requires training n models.

9.2.2 Active Learning

Active learning starts off just like random/grid search. However, after a certain number of iterations, half of the models with the highest loss are stopped, and only the half with the lowest loss are allowed to continue. This event is called *partioning*, and the difficulty of active learning is knowing when to partition and how many times to partition. Figure 9.1 shows an example of active learning with two partitions. The process of finding the right number

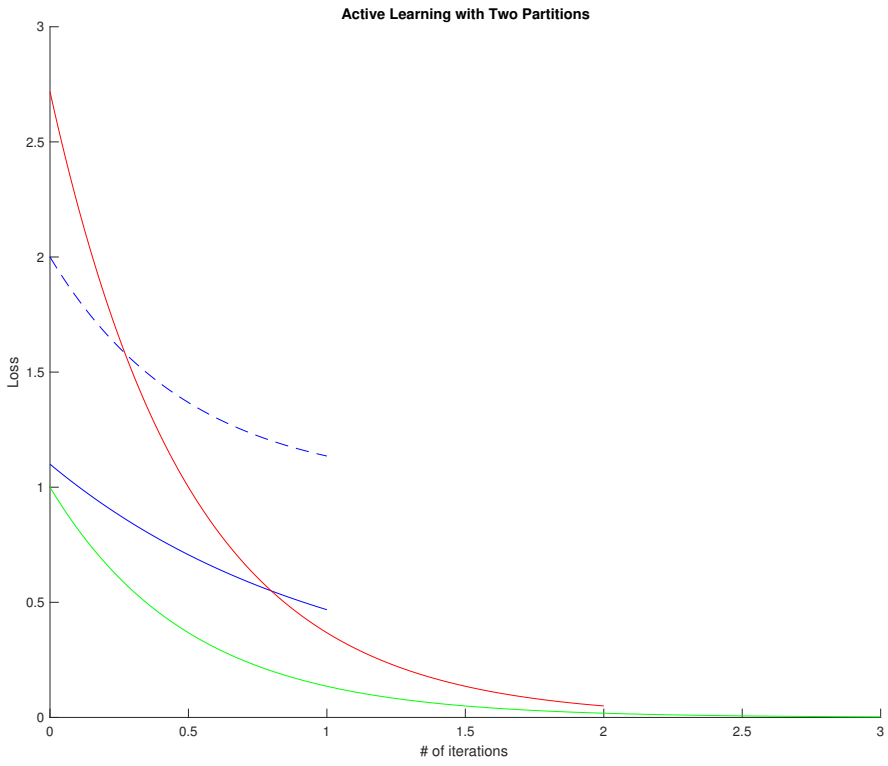

Figure 9.1. An example of active learning

of partitions is a type of hyperparameter optimization called "hyperband optimization." A quick code demo can be found here.

9.2.3 Approximate Polyak's Step Size

For this method, an optimization is run on a small subset of the data (ex. 10000 out of 1 million samples). The step size is given by

$$\gamma = \gamma_{\text{div}} - \varepsilon \tag{9.1}$$

Figure 9.2 shows an example of this method. As one can see, the black line corresponds to γ_{div} – the step size for which the loss diverges. We can use the convergent rate analysis to

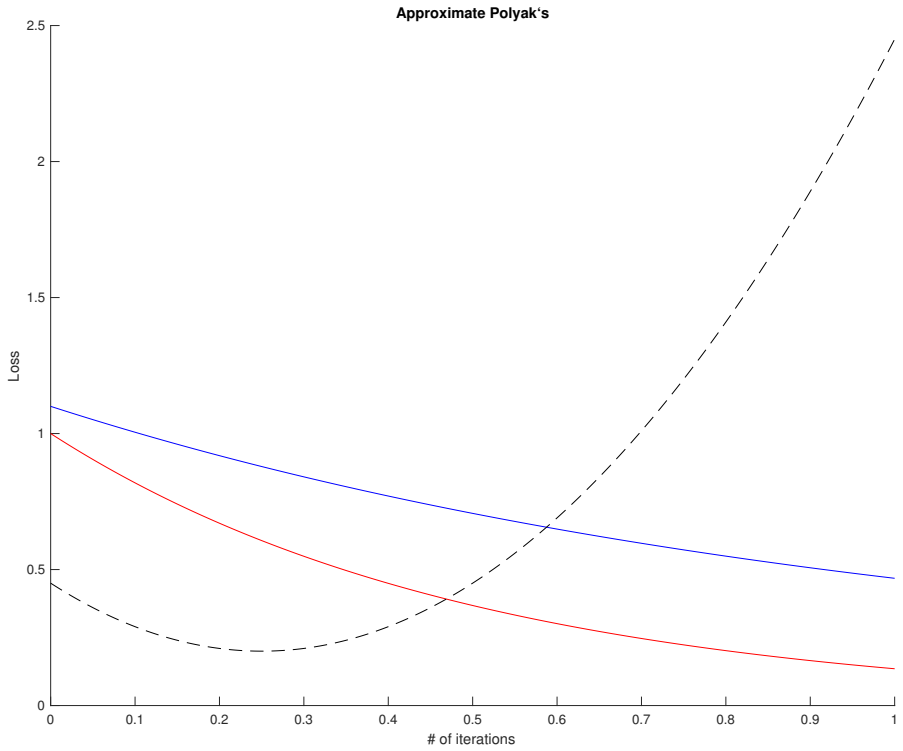

Figure 9.2. An example of approximate Polyak's step size

show the reasoning behind this method for selecting γ.
Suppose f has no constraints. According to SGD convergence,

$$\mathrm{E}\{\|x_{k+1} - x^*\|\} \leq \mathrm{E}\{\|x_k - x^*\|\} \underbrace{-2\gamma \mathrm{E}\{\langle \nabla f(x_k), x_k - x^* \rangle\} + \gamma^2 \mathrm{E}\{\|\nabla f_{s_k}(x_k)\|^2\}}_{\text{progress term}} \tag{9.2}$$

Our goal is to converge as fast as possible, so we want to maximize the progress term ($\max_\gamma \alpha\gamma + \beta\gamma^2$). For the k-th iteration, an maximizing γ can be found:

$$\gamma_k^{\text{optimal}} = \frac{\mathrm{E}\{\langle g_k, x_k - x^* \rangle\}}{\mathrm{E}\{\|g_k\|^2\}} \tag{9.3}$$

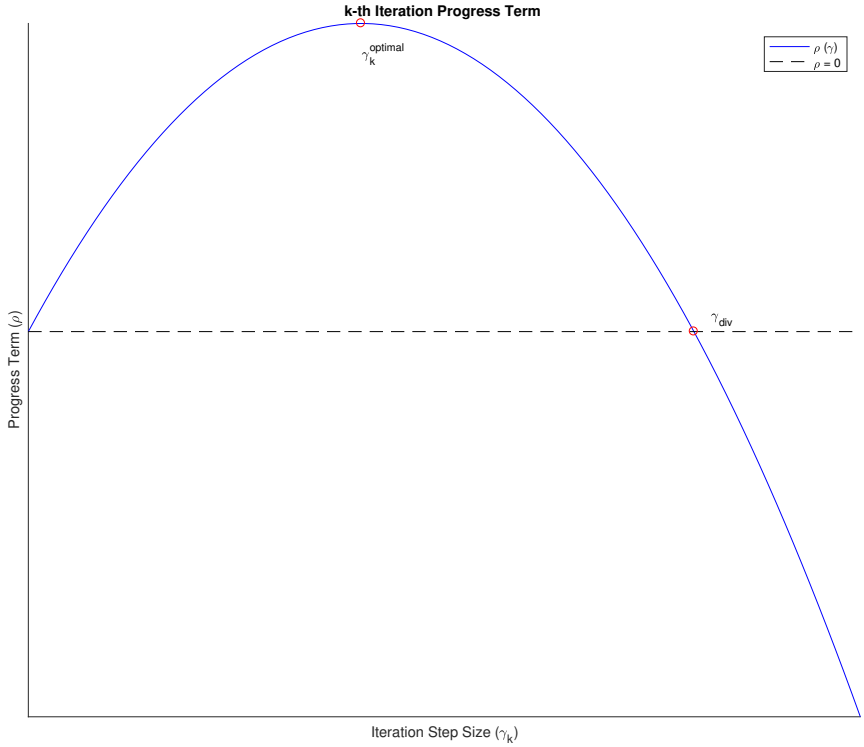

Figure 9.3. The progress term for the k-th iteration of SGD

where $g_k = \nabla f_{s_k}(x_k) = \nabla f(x_k)$.
Let $\rho(\gamma)$ be the progress term in Eq. 9.2. Then Figure 9.3 displays ρ for the k-th iteration as a function of γ_k. In particular, $\gamma_k^{\text{optimal}}$ is marked as the maximum, and γ_{div} shows where ρ switches signs from positive to negative. This figure illustrates the logic behind Eq. 9.1. By finding γ_{div} then moving back by ε, we can get close to $\gamma_k^{\text{optimal}}$.

9.3 Non-Differentiable Functions and Subgradients

Suppose $f_i(a_i^T x) = |a_i^T x - b|$. Then f_i is non-differentiable, and $\nabla f_i(x)$ does not exist for all x. However, for all f, there exists a subgradient (or under-approximator) for any $x \in X$. In particular, for convex functions, $f(x) \geq f(y) + \langle g_x, y - x \rangle$ for all $x, y \in X$, where g_x is the subgradient at x. Note that for a given point, there may exist multiple subgradients as shown in Figure 9.4. Additionally, the subdifferential is a closed and convex set defined as

$$\partial(x) = \{g \mid g \text{ is a subgradient at } x\} \tag{9.4}$$

Note that if f is L-Lipschitz, then $\|g\| \leq L$.

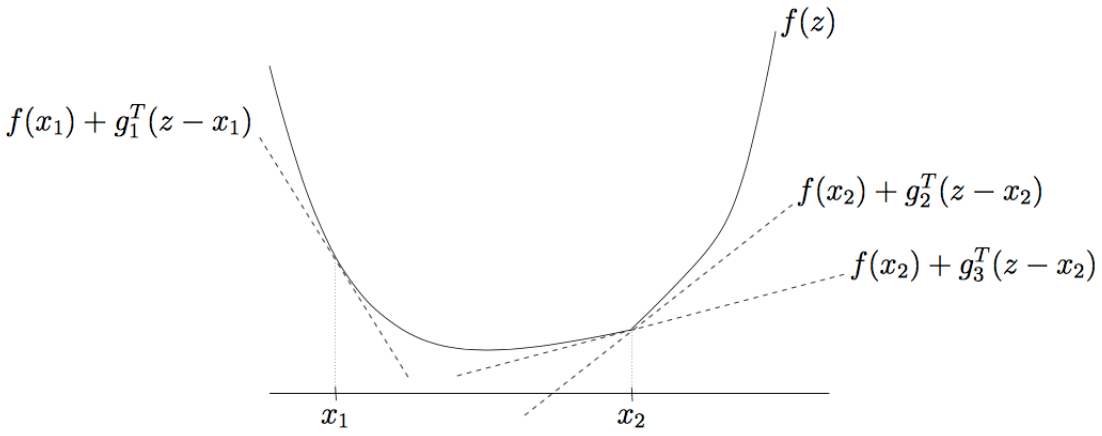

Figure 9.4. There can exist multiple subgradients at a point (from http://web.stanford.edu/class/ee364b/lectures/subgradients_notes.pdf)

Now that we have defined the subgradient to deal with non-differentiable functions, it remains to be seen whether the subgradient can be effeciently computed.
Consider the function $f = \max_{i=1\ldots n} f_i(x)$. Then the subdifferential is given by

$$\partial f(x) = \text{convex hull}\left(\bigcup_{i:f_i(x)=f(x)} \partial f_i(x)\right) \tag{9.5}$$

For any function that can be represented as a maximum of several other functions, the subdifferential can be computed according to Eq. 9.4.
This can be applied to common loss functions such as $f(x) = \|x\|_1$ (L_1-norm). Then the subgradient is given by

$$g_x(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} = \text{sign}(x)$$

This can be seen by considering a single x_i. $|x_i|$ is the absolute value function or the maximum of two lines. So,

$$\frac{\partial |x_i|}{\partial x_i} = \begin{cases} 1 & x_i > 0 \\ -1 & x_i < 0 \\ \{-1, 1\} & x_i = 0 \end{cases}$$

Similarly, the L_1-loss, $f_i(x) = |a_i^T x - b_i|$ has the subdifferential $\partial f_i(x) = a_i \text{sign}(a_i^T x - b_i)$.

Corollary 9.1. *For Lipschitz functions using subgradients, the same convergence rates hold.*

9.4 Projected (S)GD

Typically, given some loss functions $f(x)$, we try to solve $\min_x f(x)$. However, in some situations, x can have some constraints on it. Let $\mathcal{C}$ be a subset of X such that $x \in \mathcal{C}$ meets all the necessary constraints. We will call $\mathcal{C}$ the constraint set. As seen in Figure 9.5, the result of (S)GD can be outside the constraint set, so $x_k - s_k \nabla f(x_k)$ must be projected onto the constraint set. In particular, projected gradient descent is defined as

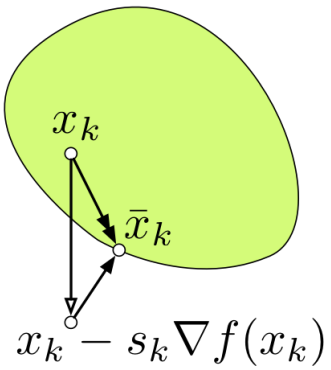

Figure 9.5. Projection of a solution onto the constraint set (from http://freemind.pluskid.org/machine-learning/projected-gradient-method-and-lasso/)

$$x_{k+1} = \text{proj}_{\mathcal{C}}(x_k - \gamma g_k) \tag{9.6}$$

where $\text{proj}_{\mathcal{C}}(x) = \arg\min_{x' \in \mathcal{C}} \|x' - x\|_2$. Note that $\|\text{proj}_{\mathcal{C}} x - x^*\| \leq \|x - x^*\|$ and $\|x_1 - x^*\| \leq \sup_{x \in \mathcal{C}} \|x - x^*\| = \text{diameter}(\mathcal{C})$. This means that projected (S)GD will not diverge, and it will have the same convergence rates as "non-projected" (S)GD.

However, projecting onto the constraint set is a small linear optimization problem at each time step. Sometimes, this can be costly.
Consider the constraint $\|x\|_2 \leq 1$. Then $\text{proj}_{\mathcal{C}} x = \frac{x}{\|x\|_2}$. In this case, $\text{proj}_{\mathcal{C}} x \sim \mathcal{O}(d)$ which is not too costly.
Suppose the constraint is $\|x\|_\infty \leq 1$. Then the i-th element of the projection is given by

$$[\text{proj}_{\mathcal{C}} x] = \begin{cases} \text{sign}(x_i) & \|x_i\| \geq 1 \\ x_i & \|x_i\| < 1 \end{cases}$$

So again, $\text{proj}_{\mathcal{C}} x \sim \mathcal{O}(d)$.
Similarly, for the L_1-norm, $\text{proj}_{\mathcal{C}} x \sim \mathcal{O}(d)$ (see Duchi et al).
Now suppose $\mathcal{C} = \{X_{n \times n} \mid X \succeq 0, \text{trace}(X) \leq 1\}$. Then

$$\text{proj}_{\mathcal{C}} X = \arg\min_{X' \in \mathcal{C}} \|X - X'\|_F \sim \mathcal{O}(d^3) = \mathcal{O}(n^3)$$

The cost of projection has scaled up by a factor of d^2. In a later lecture, we will see how to solve this problem using Frank-Wolfe (linear approximation to $\text{proj}_{m} athcal C x$).