Good Step Size Policies Adjust Often, Look Ahead, and Don't Try to Minimize the Objective Function

ali, ying, moshe, tamas, sushant, rina, shankar, alan

August 2016

1 Idea for exposition 1

We're used to thinking of gradient descent as an algorithm whose convergence rate depends on the condition number of the optimization problem. For example, we've come to accept that to minimize $f(x) = \|A(x - x*)\|^2/2$ on $\mathbf{R}^d$, gradient descent requires $O(\kappa \log 1/\epsilon)$ steps to achieve a relative error of ϵ.

But if we pick the perfecct step sizing rule, gradient descent can optimize the quadratic in exactly d steps. This requres an impractical step sizing rule. Computing this step size is as hard as minimizing $f(x)$ itself, but the existence of the rule showcases how important step sizes are in optimization.

In the case of the deterministic quadratic, the step size at iteration t is the square reciprocal of the tth singular value of A. In other words, the iterations

$$x_{t+1} = x_t - \frac{1}{\sigma^2(A)} \nabla f(x_t)$$

reaches a stationary point in d iterations and $x_d = x^*$, the minimizer of f.

This observation is important because it points out gradient descent is a vast family of algorithms whose performance can vary dramatically from linear dependence on the condition number to one that converges in only d iterations. Even though in this case computing the step size is hard, it might be possible to approximate this step sizing rule to obtain fast rates for deterministic gradient descent.

2 Idea for exposition 2

Title of the paper: Scalar Preconditioners for Faster Gradient Descent

Scalar preconditioning sounds like a ridiculous concept: the whole point of a preconditioner is to shear the gradient with a matrix. A scalar preconditioning at best scales the gradient, serving more as a step size than a preconditioner.

But we show that a careful choice of a scalar preconditioner (or equivalently a step size), can improve the rate of convergence of Gradient Descent and its stochastic variant by a huge amount: the right scalar can completely eliminate the rate's dependence on the problem's condition number.

In the case of deterministic gradient descent, setting the step size to [...described above...] causes gradient descent to converge in just d steps. This precise step size is as hard to compute as the minimum itself, so the iterations are not a practical optimization procedure.

But if one has a priori knowledge about the spectrum of the problem, replacing the step sizes with these approximations results in fast convergence. For example, if there is reason to believe the spectrum of A decays as $1/t$, a periodic step sizing rule of $\lambda_t \alpha t^2 \text{mod} d$ speeds up convergence (conjecture!)

A similar result hold in the stochastic case. In that case, the step sizing rule is more easily expressed as a linear search. On its face, this linear search appears to be as hard to compute as the optimization problem itself, but approximations are possible. If $f(x) = \|Ax-b\|^2$, line search is on $\mathcal{L}(\alpha) = (x'-x^*)' \left(A'A\right)^{-1} (x'-x*)$ with $x' = x_t - \lambda \nabla f(x_t)$. This line search appears to require both the inverse of $A'A$ and x^* itself, but in fact, because we only need to extract an approximately minimizing α there is an easy way to compute it.

(spitballing here) The equivalent line problem is $\mathcal{L}(\alpha) = ...$ which is just the residual of the least squares problem .... Computing this residual is straightforward. Finding the minimizer over α requires a bisection search on this residual.

This result appears to violate known lower bounds on the performance of least squares algorithm that have oracle access to the gradient. But in the deterministic case, we relied on side information about the spectrum of the matrix. In the stochastic case, we had access to a bisection search on a surrogate problem that requires full knowledge of A.

3 Introduction

In deterministic numerical optimization, where search directions depend only on the current iterate, line searches offer a way to pick step sizes that guarantee fast convergence. In the stochastic setting, where search directions are random functions of the current iterate, relatively little seems to be know about line searches. What is the ideal step size in each iteration? Should one strive to approximate the true mean function and conduct line searches on it? Or is it better to perform a line search on the stochastic instances of this function? Is a line search even a good idea?

We analyze line searches in the context of minimizing quadratic cost functions with stochastic gradient descent. *Be upfront that we don't actually offer a practical algorithm. We're trying to understand why our experiments didn't work.*

Surprises abound:

- Even if one had access to the true mean function, and one isn't charged for the line search at each iteration, a line search on the mean function would cause slower convergence than a fixed step size.

- SGD converges much faster with a line search on the random realizations of the function than on the mean function, despite the fact that the mean function is the objective of interest.

- One can represent the search for the optimal line search as a Markov Decision Process, and explicitly compute in closed form the optimal function on which to conduct the line search. This function turns out to be neither of the aforementioned candidates, but rather, a deterministic function whose Hessian is the inverse of that of the mean function's.

- The rate of convergence of SGD with this optimal line search has nearly the same asymptotic rate as Newton's method with exact Hessian and stochastic gradient. This rate depends only logarithmically on the condition number of the problem.

Here are the detail of the stochastic quadratic model we investigate: The value and gradients of a random quadratic on $\mathbf{R}^d$ are observed:

$$f_a(x) = \tfrac{1}{2}\left[a^\top(x - x^*)\right]^2, \qquad \nabla f_a(x) = aa^\top(x - x^*), \tag{1}$$

where a is a random vector drawn from a zero-mean Gaussian with covariance matrix A. The goal is to find x^*, the minimizer of the mean function,

$$\mathbb{E}_a f_a(x) = \tfrac{1}{2}(x - x^*)^\top A(x - x^*). \tag{2}$$

This model amounts to an overdetermined least squares problem where the independent variables are Gaussian vectors a and the dependent variable is $a^\top x^*$, a deterministic function of the dependent variables.

Stochastic gradient descent repeatedly applies the map

$$x_{t+1} \leftarrow x_t - \gamma \nabla f_a(x_t), \tag{3}$$

where the direction ∇f_a is stochastic.

Our concern is to find a good choice of the step size γ at each iteration that causes $\mathbb{E}_a f_a(x_t)$ to shrink quickly as a function of t. In practice, the cost of computing γ is an important consideration, but because we seek to characterize the behavior of SGD under an ideal γ, we allow ourselves significant leeway in the choice of algorithms for computing γ.

Consider two plausible strategies for picking γ. One is to minimize $\mathbb{E}f$ along the search direction:

$$\gamma_{mean}(x, a) \equiv \arg\min_{\gamma}\ \mathbb{E}_b f_b(x - \gamma \nabla f_a(x)).$$

where the direction ∇f_a is stochastic, but the expected value of the objective function is available. In the machine learning setting, the gradient would be computed on a mini-batch, and the line search would be conducted on the full dataset. This is obviously not a practical strategy, but one that others have instinctively sought to approximate.

Another strategy is to minimize f_a along the search direction:

$$\gamma_{sgd}(x,a) \equiv \arg\min_{\gamma} f_a(x - \gamma \nabla f_a(x)).$$

Here, both the direction ∇f_a and the value of the objective function are stochastic. In the machine learning setting, the line search and the stochastic gradient would be computed on the same batch.

More broadly, we're interested in a function $\mathcal{L}$ so that the line search

$$\gamma_{\mathcal{L}}(x,a) \equiv \arg\min_{\gamma} \mathcal{L}(x - \gamma \nabla f_a(x))$$

causes $\mathbb{E}_a f_a(x_T)$ to be small at the ultimate iterate T.

The empirical observation that prompted this note is this: With step size size γ_{mean}, the objective $\mathbb{E}_a f_a(x_t)$ decays more slowly over t than with step size γ_{sgd}. See Figure 1. This is surprising because γ_{mean} was explicitly chosen to minimize $\mathbb{E}_a f_a$, the very objective we wanted to minimize. Even more surprisingly, fixed step sizes also outperform γ_{mean}.

Furthermore, there is a step size that outperforms even γ_{sgd}. The optimal step size policy for the quadratic model is a line search over a deterministic function whose Hessian is the inverse of that of $\mathbb{E}f$. This function behavior in some ways in the exact opposite way of $\mathbb{E}f$. The reason γ_{sgd} outperforms γ_{mean} is that it is a better approximation to this optimal step sizing rule. See Figures 2 for comparisons.

4 Optimal Step Sizes For Random Quadratics

Ultimately, the quantity one cares about is not the rate of decay of $\mathbb{E}_a f_a$ from one iteration to the next, but rather, the value of the function at the last iterate, $\mathbb{E}_a f_a(x_T)$. Finding the step sizing rule that minimizes this quantity requires planning ahead. Iterates with identical objective value can converge to x^* at dramatically different rates, even on the foregoing quadratic model. So as early as x_1, a bad step size can land x_2 into a region where progress is slow thereafter.

At each iteration, choose γ to minimize the value of the ultimate iterate, $\mathbb{E} f_a(x_T)$, and assume that subsequent step sizes are similarly chosen. The expectation here is over the entire sequence of a's leading to x_T. Let $\mathcal{L}_t(x_t)$ denote the objective value $\mathbb{E}_a f_a(x_T)$ starting from x_t and assuming γ is chosen optimally from time t onward:

$$\mathcal{L}_t(x_t) = \mathbb{E}_a \min_{\gamma} \mathcal{L}_{t+1}\left(x_t - \gamma \nabla f_a(x_t)\right) \tag{4}$$

$$\mathcal{L}_T(x_T) = \mathbb{E}_a f_a(x_T). \tag{5}$$

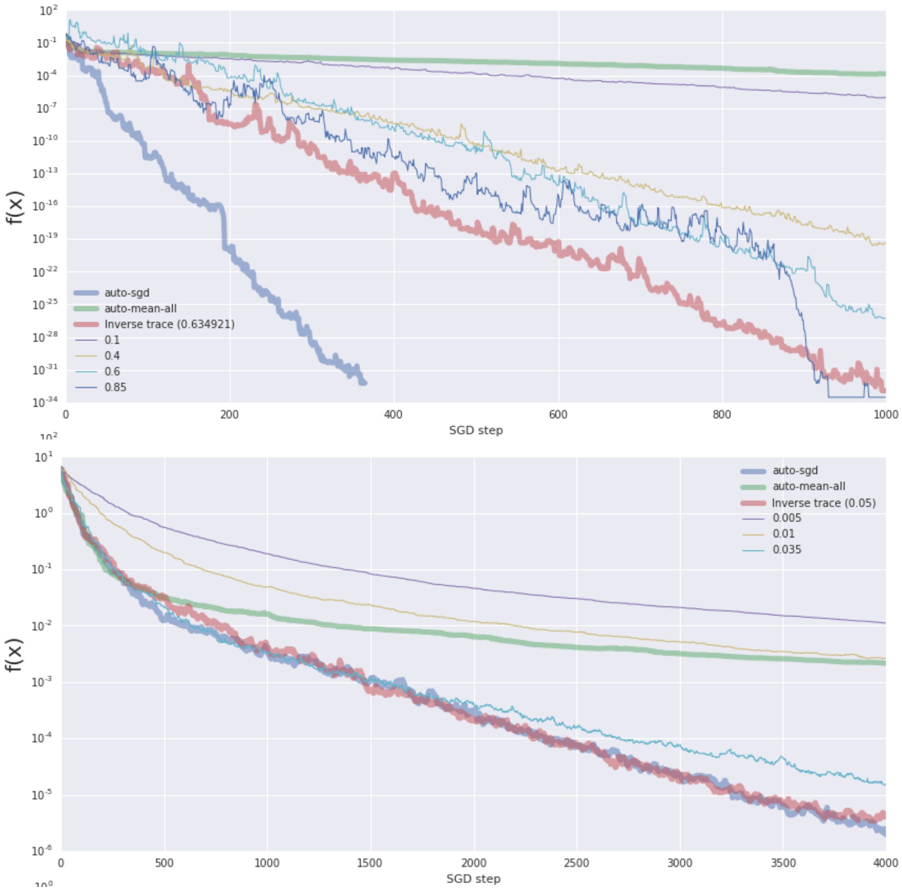

Figure 1: A line search over the mean function (`auto-mean-all`), which is the objective of interest, causes stochastic gradient descent to converge very slowly. SGD with a fixed step size (including `inverse-trace`, an automatically computed fixed step-size that maximizes the greedy convergence rate) converges faster. A line search on the stochastic function (`auto-sgd`) converges even faster. Experiments are on a stochastic quadratic described by Equations (1) and (2). In the top panel, the covariance A is 3×3 matrix with condition number 100. In the bottom panel, A is 40×40 with condition number 10^6. The larger dimensionality causes the minimizer of the stochastic function to converge to the automatically computed fixed step size.

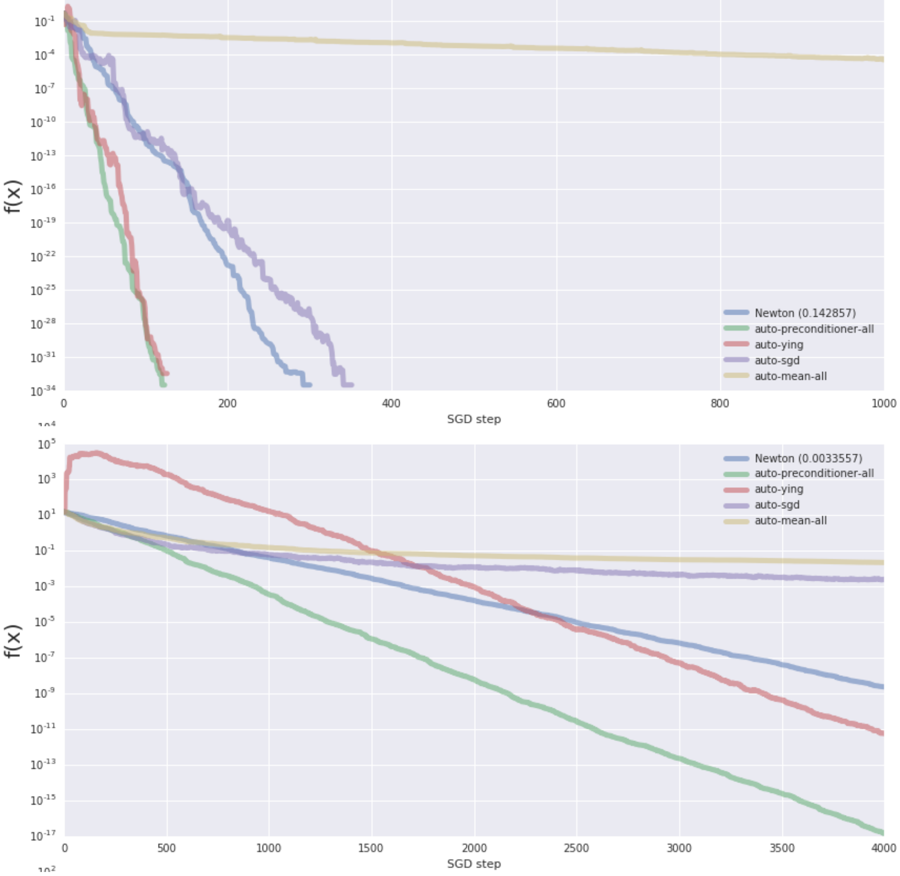

Figure 2: An optimal line search (`auto-ying`) causes stochastic gradient descent to converge nearly as faster as second order methods. The two second order methods in question are Newton step that rely on the exact Hessian and on the stochastic direction (`Newton`), and a preconditioned version of the problem, which amounts to Newton steps, but with the step sizes chosen by minimizing the stochastic function (`auto-preconditioner-all`). The top panel is the three-dimensional problem with condition number 20. The bottom panel is the 40-dimensional problem with condition number 10^6.

$\mathcal{L}_t(x_t)$ is large when the gradients at x_t are distributed so that no sequence of subsequent step sizes can ensure $\mathbb{E}_a f_a(x_T)$ is small. The choice of γ in the definition of $\mathcal{L}$ explicitly tries to avoid such bad regions.

Accordingly, the best step size at time t is given by:

$$\gamma_t(x_t, a) = \arg\min_{\gamma} \; \mathcal{L}_{t+1}(x_t - \gamma \nabla f_a(x_t)).$$

At $T-1$, the optimal step size is then just a line search on the mean function. But things are very different earlier on.

Claim 1. *With $f_a(x) = \frac{1}{2}\left[a^\top(x - x^*)\right]^2$, and $a \in \mathbf{R}^d$ a stochastic sub-gaussian vector, $\mathcal{L}_1(x_1)$ is quadratic in x_1. Furthermore, with d large, as $T \to \infty$, there exists a sequence of positive scalars $c_1 \dots c_T$ so that*

$$\mathcal{L}_1(x_1) \approx c_T (x_1 - x^*)^\top A^{-1}(x_t - x^*).$$

The $\approx$ needs more work. It is only true if T and d are balanced.

Thus the optimal line search is on $\mathcal{L}_1(x) \propto \frac{1}{2}(x - x^*)^\top A^{-1}(x - x^*)$, and not on $\mathbb{E}_a f_a(x) = \frac{1}{2}(x - x^*)^\top A(x - x^*)$.

We'll need the following lemma to prove the claim:

Lemma 1. *Let $a \in \mathbf{R}^n$ be a vector with positive components. Consider the iterations*

$$x_i^{(t+1)} \leftarrow x_i^{(t)} - a_i \left[x_i^{(t)}\right]^2 / \sum_i a_i x_i^{(t)},$$

and the function

$$\ell(x) \equiv \frac{\max_i x_i a_i}{\min_i x_i a_i} - 1.$$

If the initial iterate $x^{(0)} \in \mathbf{R}^n$ has positive components, subsequent iterates satisfy

$$\ell(x^{(t)}) \leq \ell(x^{(0)}) \left(1 - \frac{1}{n-1}\right)^t.$$

The Lyapunov function ℓ measures the deviation between x_i and $1/a_i$. It attains its minimum when $x_i = \frac{c}{a_i}$ for any constant c. The lemma therefore implies $x_i^{(t)} \to \frac{c_t}{a_i}$ as $t \to \infty$ for some sequence of scalars c_t.

Proof of Claim 1. To simplify notiation, define $v_t \equiv x_t - x^*$, so that $\nabla f_a(x_t) = aa^\top v_t$ and $\mathbb{E}_a f_a(x_t) = \frac{1}{2}\|v_t\|_A^2$.

Assume the base case that $\mathcal{L}_{t+1}(x_{t+1})$ is a quadratic $\frac{1}{2}\|v_{t+1}\|^2_{B_{t+1}}$ for some symmetric matrix B_{t+1}. This is true for $\mathcal{L}_T$ with $B_T = A$. Then

$$\begin{aligned}\mathcal{L}_t(v) &= \mathbb{E}_a \min_{\gamma} \tfrac{1}{2}\|v - \gamma a a^\top v\|^2_{B_{t+1}} && (6)\\ &= \mathbb{E}_a \min_{\gamma} \tfrac{1}{2}\|v\|^2_{B_{t+1}} - \gamma(a^\top v)(a^\top B_{t+1} v) + \tfrac{1}{2}\gamma^2 (a^\top v)^2 (a^\top B_{t+1} a). && (7)\end{aligned}$$

Plugging back the minimizer $\gamma_t^* = \frac{a^\top B_{t+1} v}{(a^\top v)(a^\top B_{t+1} a)}$ gives

$$\mathcal{L}_t(v) = \tfrac{1}{2}\|v\|^2_{B_{t+1}} - \tfrac{1}{2}\mathbb{E}_a \frac{(a^\top B_{t+1} v)^2}{a^\top B_{t+1} a}.$$

By Rina and Ying, the denominator in the expectation concentrates, yielding

$$\begin{aligned}
\mathcal{L}_t(v) &\approx \tfrac{1}{2}\|v\|^2_{B_{t+1}} - \frac{v^\top B_{t+1} A B_{t+1} v}{2 \operatorname{tr} B_{t+1} A} && (8)\\
&= \tfrac{1}{2} v^\top \left(B_{t+1} - \frac{B_{t+1} A B_{t+1}}{\operatorname{tr} B_{t+1} A} \right) v, && (9)
\end{aligned}$$

or as a recurrence,

$$\begin{aligned}
B_T &= A && (10)\\
B_t &= B_{t+1} - \frac{B_{t+1} A B_{t+1}}{\operatorname{tr} B_{t+1} A} && (11)\\
\mathcal{L}_t(v) &= \tfrac{1}{2}\|v\|^2_{B_t}. && (12)
\end{aligned}$$

Now solve the recurrence assuming $T \to \infty$. Let $A = P D_A P^\top$ be the eigendecomposition of A. Then the eigendecomposition of B_t takes the form $P D_{B_t} P^T$, and the iterations imply $D_{B_{t+1}} = D_{B_t} - D^2_{B_t} D_A / \operatorname{tr} D_{B_t} D_A$. Write these in vector form and apply Lemma 1. □

Proof of Lemma 1. As a short hand, define $w_i \equiv a_i x_i^{(t)}$, and $w_i' \equiv a_i x_i^{(t+1)}$. The iteration becomes

$$w_i' \leftarrow w_i \cdot \left(1 - \frac{w_i}{\sum_k w_k} \right).$$

We will show that the iterations contracts $\frac{\max_i w_i}{\min_i w_i} - 1$.

Observe that the iteration leaves intact the total ordering of $w_1 \dots w_n$. Indeed, for any pair of indices i, j,

$$w_i' - w_j' = w_i - w_j - \frac{w_i^2 - w_j^2}{\sum_k w_k} = (w_i - w_j)\left(1 - \frac{w_i + w_j}{\sum_k w_k} \right).$$

Since $1 - \frac{w_i + w_j}{\sum_k w_k}$ is non-negative, $w_i - w_j$ and $w_i' - w_j'$ have the same sign for all pair i, j, so the ordering of the entire sequence $w_1 \dots w_n$ is preserved across iterations. In particular, if s and l are the indices of the smallest and largest element of $w_1, \dots w_n$, then w_s' and w_l' remain respectively the smallest and largest element of $w_1' \dots w_n'$.

Assume heretofore, without loss of generality, that the indices are sorted in ascending value of w, and consider the ratio $r \equiv w_n/w_1$:

$$\begin{aligned} r' \equiv \frac{w'_n}{w'_1} &= \frac{w_n(1 - w_n/\sum_k w_k)}{w_1(1 - w_1/\sum_k w_k}) = \frac{w_n}{w_1} \frac{\sum_k w_k - w_n}{\sum_k w_k - w_1} && (13) \\ &= \frac{w_n}{w_1} \left(1 - \frac{w_n - w_1}{\sum_k w_k - w_1} \right) && (14) \\ &\leq \frac{w_n}{w_1} \left(1 - \frac{w_n - w_1}{(n-1) w_n} \right) && (15) \\ &\leq r \left(1 - \frac{r-1}{(n-1)r} \right) = r - \frac{r-1}{n-1}. && (16) \end{aligned}$$

Therefore,

$$r' - 1 \leq (r - 1) \left(1 - \frac{1}{n-1} \right).$$

Remarking that $\ell(x) = r - 1$ proves the lemma. □

5 How Can a Scalar Step Size Strategy Achieve Newton-like Rates?

... where we explicitly write γ_{ying} and read the tealeaves to explain how it penalizes the bad spots.

bound the spectral radius of the iteration matrix, and show a rate of $1 - 1/d$ *on the inverse function.*

6 Experiment on a Real Network

... where i use a diagonal approximation to the hessian like RMSPROP's to compute γ_{ying} and run on CIFAR. Though i'm not sure we should do this. This note isn't about a practical algorithm for step sizing. It's about what not to approximate in step sizing.