10-425/625: Introduction to Convex Optimization (Fall 2023)

Lecture 25: Parallel SGD / Nonconvex Optimization

Instructor:[1] *Matt Gormley* *December 6, 2023*

25.1 Parallel SGD

Recall from a previous lecture

25.1.1 Distributed Asynchronous SGD

A common alternative to distributed *synchronous* SGD is to allow a single parameter server to asynchronously receive *and apply* gradient updates. This comes at a great cost however: the asynchronous algorithm is no longer faithful to SGD, which is an inherently serial algorithm.

Algorithm 1 AsyncSGD-ParameterServer

Choose initial point $x^0 \in \mathbb{R}^n$
for $t = 1, 2, \ldots, T$ **do**
    Wait to receive $g_k^{(t)}$ from the next worker
    Immediately apply update $x^{t+1} = x^t - \eta_t g_k^{(t)}$

Algorithm 2 AsyncSGD-Worker-k

for $s = 1, 2, 3, \ldots$ **do**
    Request $x^{(s)}$ from parameter server
    Sample minibatch $I_k^{(s)} \subseteq \{1, \ldots, n\}$ of size b
    Compute $g_k^{(s)} = \sum_{i \in I_k^{(s)}} \nabla f_i(x^{(s)})$
    Send $g_k^{(s)}$ to parameter server

[1] These notes were originally written by Siva Balakrishnan for 10-725 Spring 2023 (original version: here) and were edited and adapted for 10-425/625.

25.1.2 Delayed SGD

Next we consider what happens if you define the algorithm to explicitly include a fixed amount of delay (staleness). The amount of delay τ is the number of iterations between when the update is made and which iterate was used to compute the gradient. The algorithm is called Delayed SGD (Zinkevich et al., 2009). We consider the full version which assumes the iterates are constrained to a convex set C.

Algorithm 3 AsyncSGD-ParameterServer

Choose initial points $x^{(0)}, \ldots, x^{(\tau-1)} = 0 \in \mathbb{R}^n$
for $t = \tau, \ldots, T - \tau - 1$ **do**
    Receive the next function f_t and incur loss $f_t(x^{(t)})$
    Compute gradient $g^{(t)} = \nabla f_t(x^{(t)}$
    Update $x^{(t+1)} = \text{argmin}_{x \in C} \|x - (x^{(t)} - \eta_t g^{(t-\tau)})\|_2$

The update rule is equivalent to $x^{(t+1)} = P_C(x^{(t)} - \eta_t g^{(t-\tau)})$ where P_C is the projection operator. Notice delay occurs because we are using stale gradients $g^{(t-\tau)}$.

Despite this we can obtain bounds on the regret as follows:

- For β-smooth, convex f_t, with an appropriate learning rate the regret can be bounded as $R(T) \in O(\tau\sqrt{T})$
- For β-smooth, α-strongly convex f_t, with an appropriate learning rate the regret can be bounded as $R(T) \in O(\tau + \log(T))$

25.2 Nearly-Nonconvex Optimization

Next, we'll discuss a few of the main ideas related to some non-convex problems where we know how to guarantee (sometimes local) convergence to a global optimum. These settings are still pretty far from things like optimization in deep learning, but they're already fairly interesting settings.

At a very high-level one should view both the results that we discuss as types of stability/robustness statements for GD, i.e. GD doesn't quite require convexity to be effective.

25.2.1 The Polyak-Lojasiewicz (PL) Condition

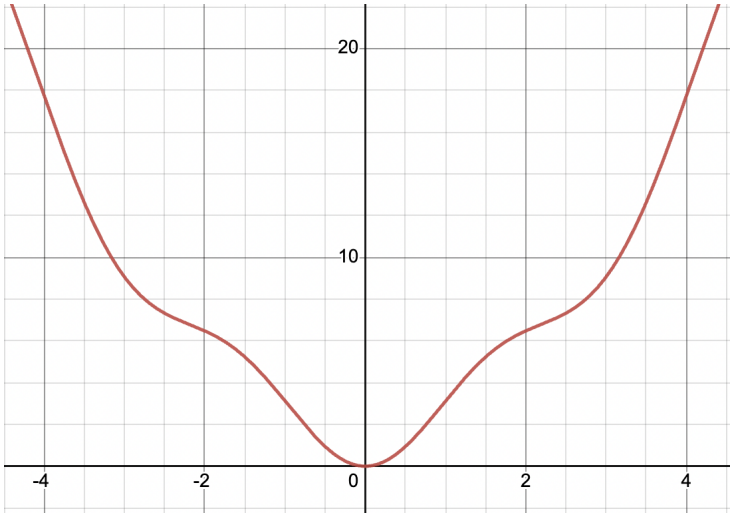
25.2.1.1 The PL Condition

Suppose we have a β-smooth function f, that we're interested in minimizing (unconstrained). We'll assume that a minimizer x^* exists (need not be unique). This function need not be convex. Suppose it instead satisfies the so-called μ-PL inequality, i.e. for some $\mu > 0$,

$$\frac{1}{2}\|\nabla f(x)\|_2^2 \geq \mu(f(x) - f(x^*)).$$

The way to interpret this condition is that it is some weakening of strong convexity. Recall, that when we discussed the linear convergence of GD we highlighted the key property that strong convexity gave us, if we're far away from optimal then strong convexity will ensure that the gradient is large. The PL condition is a more direct statement of that desirable property (but highlights the crucial fact that this is all we need for linear rates, i.e. we do not need convexity itself).

The paper of Karimi-Nutini-Schmidt which re-popularized this condition and highlighted its usefulness also gives examples of some non-convex functions which satisfy the condition.



Example function satisfying α-PL condition for $\frac{1}{2\alpha} = 1/32$, $f(x) = x^2 + 3\sin^2(x)$ from Karimi et al. (2020).

Here is a lemma relating the PL condition to strong convexity:

Lemma 25.1. *An α-strongly convex function also satisfies the α-PL inequality.*

Proof: We know by strong convexity that for any y,

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{\alpha}{2}\|x-y\|_2^2.$$

We can now minimize both sides with respect to y to see that,

$$f(x^*) \geq f(x) - \frac{1}{2\alpha}\|\nabla f(x)\|_2^2,$$

which is precisely the α-PL inequality. ∎

25.2.1.2 GD Convergence under PL Condition

The reason why the PL condition is useful is that it is sufficient (together with smoothness) to ensure linear convergence of GD. Here is a theorem:

Theorem 25.2. *Suppose f is β-smooth, and μ-PL, then GD iterates with $\eta = 1/\beta$ converge linearly, i.e.*

$$f(x^k) - f(x^*) \leq \left(1 - \frac{\mu}{\beta}\right)^k (f(x^0) - f(x^*)).$$

Proof: The proof is in some sense a bit simpler than the proof of convergence with strong convexity (since we've already distilled its essence into the PL inequality). Recall our main descent lemma which holds under smoothness for our choice of step-size,

$$\begin{aligned} f(x^{t+1}) &\leq f(x^t) - \frac{1}{2\beta}\|\nabla f(x^t)\|_2^2 \\ &\leq f(x^t) - \frac{\mu}{\beta}(f(x^t) - f(x^*)). \end{aligned}$$

Re-arranging we get,

$$f(x^{t+1}) - f(x^*) \leq \left(1 - \frac{\mu}{\beta}\right)(f(x^t) - f(x^*)).$$

This yields the claimed result. ∎

1. This result is at the heart of many global convergence results in non-convex optimization (for instance, the analysis of GD for a randomly

initialized neural network in the so-called NTK regime, or the analysis of the policy gradient method for LQR). The main convenience is that convexity is some type of global property of a function that can be hard to verify in some examples, whereas the PL condition is often easier to verify. In some sense it is a type of 2-point property, i.e. we only need to verify some condition on the function at the current iterate, relative to the optimal point, in order to ensure that we make progress in the current iteration.

2. In our analysis of GD under strong convexity and smoothness we showed linear convergence of the iterates (not just of their associated function values). It turns out that something similar is true for PL functions (one needs to be a bit careful since x^* is not unique, so the distance to the set of the solutions decreases linearly). This proof is a bit involved, but one can show that PL functions exhibit something called quadratic growth (again the Karimi-Nutini-Schmidt paper is a great resource), i.e. letting x^* denote the closest optimal solution to x we have,

$$\frac{\mu}{2}\|x - x^*\|_2^2 \leq f(x) - f(x^*).$$

This in turn allows us show linear convergence of the iterates.

25.2.2 Local convergence to a Global Optimum of GD

We won't have time to belabour this point, but it's also worth noticing, that our proofs for convergence of GD under strong-convexity and smoothness, or under PL and smoothness don't require these conditions to hold globally.

In many cases, one can simply argue that the conditions hold locally around some optimal point x^*, i.e. in some ball of radius r around x^* (say). Our guarantees ensure that if we satisfy these conditions just within the ball and initialize within the ball, the iterates will (or can be shown in most cases) to stay within this ball, and converge linearly to the optimal solution. This is easier to explain with a picture.

The key takeaway is, if you can show some nice properties (things like smoothness, strong-convexity/PL) hold locally around x^* in some region, then you can usually make some type of statement about the local convergence to x^* (i.e. provided you initialize your algorithm smartly it will converge to the global optimum x^*).

25.3 Nonconvex Optimization and The Edge of Stability

25.3.1 Recall: GD on Smooth Possibly Non-Convex Functions

Definition 25.3 (β-Smooth)**.** *A function f is β-smooth, if its gradient is Lipschitz continuous with parameter β, i.e. for any $x, y \in dom(f)$,*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \beta \|x - y\|_2.$$

Recall from back in Lecture 9 that some of our first convergence results with gradient descent *did not require convexity*. We only assumed that the function was β-smooth (i.e. had Lipschiptz gradient with parameter β).

For a not necessarily convex problem, we should not expect to be able to find a point which is a global optimum. Instead we'll settle for finding a point with small gradient norm, i.e. a point x for which $\|\nabla f(x)\|_2 \leq \epsilon$ (say). These points are called ϵ-substationary. These points are called approximate saddle points (points where the gradient is 0 are called saddle points).

The main "descent" lemma for gradient descent:

Lemma 25.4. *For a β-smooth function f, and any step-size $\eta \leq 2/\beta$, the GD algorithm is a descent algorithm. For any $\eta \leq 1/\beta$ it further satisfies,*

$$f(x^{t+1}) \leq f(x^t) - \frac{\eta}{2}\|\nabla f(x^t)\|_2^2.$$

The main theorem for gradient descent:

Theorem 25.5. *For a β-smooth function f, let x^* be any minimizer of f, then GD with step-size $\frac{1}{\beta}$ has the property that within k iterations it will reach a point x such that*

$$\|\nabla f(x)\|_2 \leq \sqrt{\frac{2\beta}{k}(f(x^0) - f(x^*))}.$$

25.3.2 Implications for nonconvex, but β-smooth functions

Revisiting these results while considering that f could be nonconvex reveals that they are quite profound:

1. The lemma says that as long as the function f is (at least locally) β-smooth, then each iteration will yield a decrease in our objective; and that decrease will be larger when the gradient is larger.
2. The theorem tells us that even for nonconvex functions gradient descent will converge to a saddle point (i.e. a flat part of the function) at a rate of $O(\sqrt{k})$. So if we use gradient descent on a deep neural network, we can still guarantee something about its convergence. Of course, this isn't the guarantee we really *want*, but it is a good starting point.
3. The key requirement for both of these is that we choose the learning rate appropriately (i.e. proportional to the inverse of the smoothness of the function).

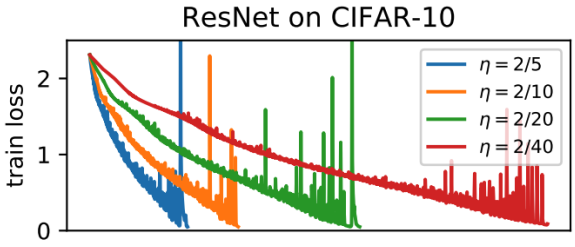
25.3.3 Nonconvexity and Deep Learning

Why do we care about nonconvexity? Because most of the objective functions that we optimize in machine learning are nonconvex. This is a simple property of the deep neural networks that dominate most of the applications of ML today.

25.3.4 Does Deep Learning Work Because We Picked Safe Learning Rates? No.

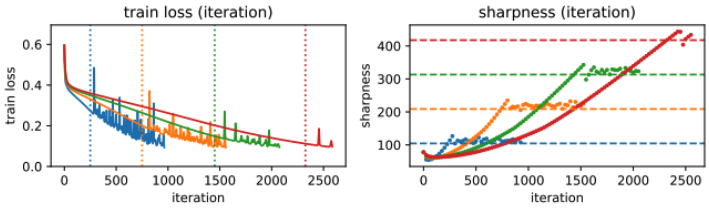
Cohen et al. (2022) show empirically that when training deep neural networks with gradient descent, the optimizer converges and yet the loss is *not* behaving according to the descent lemma: instead, it increases and decreases wildly during training. This means that the learning rates we use must not be $\eta \leq 1/\beta$ for the smoothness β of the objective.

Gradient descent convergence for ResNet on CIFAR-10 from Cohen et al. (2022).



They observe the same behavior when applying Nesterov momentum to a fully-connected neural network with tanh activations.

Gradient descent convergence for tanh network from Cohen et al. (2022).



We can measure the local smoothness of the function by the largest eigenvalue of the Hessian of the training loss, termed the *sharpness* of the function. When the sharpness is more than $2/\eta$, training for a locally quadratic function will become unstable. And, empirically, we see that the iterates where the objective function spikes occur are when the sharpness of the function as exceed the $2/\eta$ threshold (horizontal dashed line). Further even though we would expect the sharpness to continue to rise much higher than $2/\eta$ gradient descent somehow prevents it from doing so.

So when training a neural network, if the objective function suddenly spikes, the natural solution would be to decrease the learning rate. However, this would be a mistake in practice: in fact, retaining the same high learning rate, gradient descent continues to (overall) improve the objective. So (counter-intuitively) we want to use a learning rate that violates the main gradient descent lemma, since this will lead to the fastest convergence.

25.3.5 The Edge of Stability

Applied to deep learning, gradient descent (and other optimization algorithms) succeed in this regime where the learning rate is high, and the sharpness hovers just above the $2/\eta$ threshold. This space is called *the Edge of Stability*. To date, very little is understood about *why* gradient descent still

works in this way.