ALG-ML SEMINAR

ACCELEARTION, COORDINATE DESCENT, AND ROTATION MARCH 1, 2016

LECTURER: ZEYUAN ALLEN ZHU SCRIBE: KIRAN VODRAHALLI

Contents

1 Introduction

We're going to describe several optimization talks under a similar framework.

Abstract: This talk is divided into three pieces so related to three papers. In Part 1, I will revisit my linear-coupling work that unifies gradient descent and mirror descent. This result has led to many recent breakthroughs in positive LP and SDP (including two-player games, bipartite matching, etc. In Part 2, I will talk about how to use this linear-coupling framework to obtain even faster accelerated coordinate-descent methods than Sidford-Lee. In Part 3, I will talk about how to train machine-learning problems faster by rotating the dataset. This part depends on Part 2.

2 Linear-Coupling of Gradient and Mirror Descent

This is about a new framework to help understand the connections between gradient descent and mirror descent. And we can analyze accelerated gradient descent with a much simpler analysis.

Let us redefine gradient and mirror descent. Consider $f(x)$ convex and L smooth. That is, $\|\nabla f(x) - \nabla f(y)\| \leq L \cdot \|x - y\|$. The gradient is not changing too much. Note that this definition, you get sub-second order derivatives for L-smooth. Then, $\|\nabla^2 f(x)\|_{\text{spectral}} \leq L$ implies this: The other way does not hold, but it does in "nice cases", i.e., cases where you actually have differentiability.

For smooth functions, we have gradient descent: We iteratively move from x to a different point $x' = x - \frac{1}{L}\nabla f(x)$. We just have a linear lowerbound in the hyperplane. Knowing that the function is smooth gives a very interesting property of the function, namely that there exists a quadratic upper bound on the function. It is precisely $f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2$ (Not we're using Euclidean norms throughout). So we get a linear lower bound and a quadratic upper bound. This is an equivalent definition of smoothness. We like steps of $1/L$ because it is precisely the minimizer of the quadratic upper bound. Now gradient descent satisfies this: By comparing the distance between $f(x)$ and $f(x')$: We are for sure decreasing the objective, and furthermore, we're descreasing it by at least $\frac{1}{2L}\|\nabla f(x)\|^2$: We will call this the gradient descent lemma:

Lemma 2.1. *Gradient descent lemma.*
With gradient steps $x' = x - \frac{1}{L}\nabla f(x)$ for convex smooth f, at any step we decrease the objective by this much:

$$f(x') \leq f(x) - \frac{1}{2L}\|\nabla f(x)\|^2$$

Now if we look at the zoo of first order methods, a lot of them are of this type of guarantee. But there is another category: Mirror descent.

What is mirror descent? Let us say we're at the point $z_{k+1} \leftarrow \text{argmin}\left\{\frac{1}{2}\|z - z_k\|^2 + \alpha \nabla f(z_k)^T z\right\} = z_k - \alpha \nabla f(z_k)$. In some special cases (Euclidean case), we actually see that it's the same as gradient descent. But in some cases, α does not guarantee you reduce the objective at each step. Furthermore, for different norms, like l_1 of l_∞, it's no longer the same again. Gradient descent looks like in the l_1 case

$$\text{argmin}_{x'}\left\{\frac{1}{2}\|x' - x\|_1^2 + \frac{1}{L}\nabla f(x)^T x'\right\}$$

where smoothness is defined as

$$\|\nabla f(x) - \nabla f(y)\|_\infty \leq L\|x - y\|_1$$

(note that these are dual norms). In general, the definition of **smoothness** is as follows:

Definition 2.2. Smoothness.

$$\|\nabla f(x) - \nabla f(y)\|_q \leq L \cdot \|x - y\|_p$$

where p is the primary norm and q is the dual norm.

This is no longer so easy to compute necessarily.

People view this stuff different ways. Elad views it a bit differently:

You have some regularizer function R. Then the gradient ∇R is a vector mapping. In mirror descent, you say $\nabla R(x_{t+1}) \leftarrow \nabla R(x_t) + \eta \nabla f(x_t)$ where f is the function we're optimizing over: Applying gradient descent in some vector field, and this is arguably some vector field. If we set $R(x) = \|x\|^2$, we get gradient descent. If we set $R(x) = \text{entropy}(x)$, this is multiplicative weights. Why do we call it regularization? Well think of RFTL (regularized follow-the-leader). Then we can write

$$x_{t+1} \leftarrow \text{argmin}\left\{\sum_{t=1}^{T} \nabla f(x_t)^T x_t + R(x)\right\}$$

Now why is this interesting? Newton's method is a special case of this, when you take $R(x) = x^T H x$, where H is the Hessian $\nabla^2 f(x)$. So it's sort of a general thing. What Zeyuan is talking about is you can view the same thing as an argmin: This is more intuitive with respect to learning. These two things are identical if f is a linear function. Here in this explanation you don't need smoothness.

On the other hand, in mirror descent, we put something like **relative entropy** in place of a norm in the mirror descent, and you get something like the multiplicative weight update. We get

$$\text{argmin}_z\left\{\sum_i z_i \log(\frac{z_i}{z_{*,i}}) + \alpha \nabla f(z_k)^T z\right\}$$

In the Euclidean norm case, they look the same. But in the l_1 norm case, they look different. There are two concerns: One of them is convergence rate, which will depend on the smoothness parameter, which may change if we change on the underlying norms.

Whenever we talk about mirror descent, there's a step: We have to pick a regularizer. Now just think about whatever regularizer we picked. One way to write mirror descent update is like what we did before, but we can write it in the argmin sense:

We first define Bregman divergence:

Definition 2.3. Bregman divergence.
This is a non-symmetric function:

$$V_x(y) = R(y) - R(x) - \nabla R(x)^T(y - x)$$

where R is a regularizer we pick. So whenever the regularizer R is convex, we have

$$V_x(y) \geq 0$$

Now we can write mirror descent as the following:

Definition 2.4. Mirror descent.

$$z_{k+1} = \text{argmin}_z \left\{ V_{z_k}(z) + \alpha \nabla f(z_k)^T z \right\}$$

where V is the Bregman divergence of the regularizer R chosen before. This allows us to write down the mirror descent update rule for any regularizer R using Bregman divergence.

For instance, in the Euclidean norm case, $V_{z_k} = \frac{1}{2}\|z - z_k\|_2^2$. In most applications, R better be strongly convex with respect to the underlying norm. That means that we must satisfy

$$V_x(y) \geq \frac{1}{2}\|y - x\|^2_{\text{whatever norm}}$$

It will be clear why we need this strongly convex regularizer in a bit. The 2-norm squared regularizer is strongly convex (that's because it looks exactly the same as the difference between the two points in 2-norm squared). In the case for l_1 norm, then there is actually a very nice function which is strongly convex with respect to the l_1-norm in the simplex (i.e. probability space). If this is the only space of vector x that you ever care about, then the **entropy regularizer** $R_{\text{ent}}(x) = \sum_i x_i \log x_i$. It's probably the second most used in the literature, and really it boils down to exponentiated gradient (which is similar to multiplicative weights; the update rule is slightly different). This is applied to nonlinear things too.

Using this language, we can write down some lemmas about mirror descent. This regularization stuff is very interesting, but today for the rest of the talk, we will only focus on the Euclidean norm.

Now, suppose we do the mirror descent update. The following lemma "Mirror Descent Lemma" is as follows:

Lemma 2.5. *Mirror Descent Lemma.*
For every $u \in \mathbb{R}^d$,

$$\alpha \nabla f(z_k)^T(z_k - u) \leq \frac{\alpha^2}{2}\|\nabla f(z_k)\|^2 + \frac{1}{2}\|z_k - u\|^2 - \frac{1}{2}\|z_{k+1} - u\|^2$$

*In particular we sometimes care about u which is optimal. The mirror descent lemma is doing something on the dual of the function. The LHS can be viewed in different ways. In online learning, it can be viewed as the regret. Here, we will think about it as the linear lower bound to the convex function $f(x)$. This lower bound is $g(u) = f(z_k) + \nabla f(z_k)^T(u - z_k)$, for arbitrary u. This function g is a linear lowerbound hyperplane. From the high level, at any point z, you have a linear lower bound hyperplane function $g(u)$. If you look at a different point of the function, you have another linear lower bound hyperplane. If you sample intelligently at a few different points, you keep getting linear lower-bounding hyperplanes. If you take the average of those hyperplanes, you get another linearly lowerbounding hyperplane. In short, mirror descent wants to intelligently decide which points you want to query so that the **average** linearly bounding-hyperplane is as close to average as possible. This mirror descent update and accompanying lemma ensure this: By picking a telescoping of the inequality with respect to all iterations k, and taking an average. This inequality has been famously used in mirror descent, multiplicative weights, online learning.*

Today, instead of proving this lemma, we'd like to talk about how to use that. Because f is convex, we have the bound

$$\alpha(f(z_k) - f(u)) \leq \alpha\langle \nabla f(z_k), z_k - u\rangle$$

and then we arrive at the average by telescoping

$$\frac{1}{T}\sum f(z_k) - f(u) \leq \frac{2\sum \|\nabla f(z_k)\|^2}{T} + \frac{\|z_0 - u\|^2}{\alpha T}$$

In mirror descent, they assume the function f is Lipschitz continuous (i.e., that $\|\nabla f(z)\|^2 \leq G$). Then we can replace the first term with

$$\frac{1}{T}\sum f(z_k) - f(u) \leq \alpha G + \frac{\|z_0 - u\|^2}{\alpha T}$$

Now by choosing α appropriately such that the terms equal each other, we then get

$$\frac{1}{T}\sum f(z_k) - f(u) \leq \sqrt{\frac{G\|z_0 - u\|^2}{T}}$$

Assuming we have this boundedness on the gradient, we have this guarantee.

At this point, we can think of u as the global minimizer of the convex function f. Then $f(\bar{z}) - f(u)$: By taking an average of the iterates, then we can obtain a point where the average is decreasing, with rate $\frac{1}{\sqrt{T}}$. This is the convergence rate of mirror descent with respect to Lipschitz continuous functions. This analysis corresponds to the picture we draw here. By taking an average with respect to the "linear lower bounding hyperplanes", we want to make sure the average is not too large (choosing α to make sure it's as small as possible). In the end we only care about the distance with respect to the global minimizer.

Now, we see two things: First, the mirror descent lemma, which corresponds to the mirror descent update. Second, we have a gradient descent update. The two lemmas look different from each other.

Now comes the interesting part. Given those two very classical results, I want to recover Nesterov's Accelerated Gradient Descent method as a coupling of these two procedures.

Now, whenever we see a large gradient in gradient descent, we can actually guarantee we are decreasing gradient by a large amount. So gradient descent prefers large gradients. On the other hand, mirror descent prefers small gradients. Whenever, G becomes smaller, we get better convergence because of the telescoping nature of things. We didn't talk about the convergence rate of gradient descent, but just by looking at the gradient descent lemma, we see that we prefer large gradients there.

Now, Nesterov's method is just doing a very clever linear combination of the two: Whenever the gradient descent is large, we want to do more gradient descent. Whenever gradient descent is small, we want to do more mirror descent. So at some x_k, we can query the gradient, and move it to some x_{k+1}. Maybe gradient is small, we can use mirror descent update to move to some other $\tilde{x}_{k+1}$. However, unless you apply each method for a substantial number of iterations, you don't see the effect of either of them. If you are **alternating** between the two, then in principle you may sometimes increase and sometimes decrease, and you may not have convergence anymore. Another concern is that the mirror takes the average as the optimal, while gradient descent takes the last point. This is also an issue of concern.

So instead, we have the following update rule: There are three steps: We look at $\{(x_k, y_k, z_k)\}_{k\geq 1}$.

1. The first step is just a linear weighted sum of two points.

$$x_{k+1} = \tau_k z_k + (1 - \tau_k) y_k$$

2. The second thing is essentially a gradient descent step:

$$y_{k+1} = x_{k+1} - \frac{1}{L}\nabla f(x_{k+1})$$

3. Finally, we have the mirror descent update:

$$z_{k+1} = z_k - \alpha \nabla f(x_{k+1})$$

Note that z_{k+1} starts from z_k, while y_{k+1} starts from x.

Again, here we're not assuming that f is directly strongly convex, which means there are only **linearly lower bounding** hyperplanes. If you had f strongly convex, then there are upper and lower quadratic bounds. Sometimes some of the geometric intuitions only works for strongly convex things and there are some limitations. The algebraic interpretation behind Nesterov's method gives a simpler way to handle acceleration in a not-so-geometric way, so we can apply it to cases beyond Nesterov's original scope.

We will apply the lemmas individually, and then combine them together on top of these rules to see why we have to design the method this way. In essence, we're trying to guess why Nesterov defined it this way.

As you will see, in the very end with this method, it's only a matter of parameter tuning to handle different versions of Nesterov's acceleration: i.e. smooth versus condition-number and so on. This method works in general with respect to these parameters.

Anyways, we can decrease

$$f(y_{k+1}) \leq f(x_{k+1}) - \frac{1}{2L}\|\nabla f(x_{k+1})\|^2$$

which follows from our previous Gradient Descent lemma. The second thing that follows is This is not a repetition of our previous Mirror Descent lemma, since that was for z_k. But actually, we can replace arbitrary the gradient with an arbitrary vector. Therefore, if we put the loss vector as being the gradient at point $k+1$, then we just change $f(z_k)^T(z_k - u)$ to $\nabla f(x_{k+1})$.

$$\alpha \nabla f(x_{k+1})^T(z_k - u) \leq \frac{\alpha^2}{2}\|\nabla f(x_{k+1})\|^2 + \frac{1}{2}\|z_k - u\|^2 - \frac{1}{2}\|z_{k+1} - u\|^2$$

Eventually we want to upper bound this by the objective difference. Recall that previously we lower bounded the inner product by the function difference. We can only do that when the gradient is with respect to the same point (z instead of x). So suppose now that we consider the really interesting thing in terms of mirror descent, then if we replace $\nabla f(x_{k+1})^T(z_k - u)$ with $\nabla f(x_{k+1})^T(x_{k+1} - u)$, we will incur another loss:

$$\alpha \nabla f(x_{k+1})^T(x_{k+1} - u) = \alpha \nabla f(x_{k+1})^T(x_{k+1} - z) + \alpha \nabla f(x_{k+1})^T(z_k - u)$$

We lose exactly $\alpha \nabla f(x_{k+1})^T(x_{k+1} - z)$ to get this term. So the idea is that we move from $x_{k+1} \to y_{k+1} \to x_{k+2} \to y_{k+2}$. So we lose from $x_{k+1} \to y_{k+1}$, gain something from y_{k+1} to x_{k+2}, and then lose again from $x_{k+2} \to y_{k+2}$. The idea is that these will "cancel out". This is where the magic happens: We can write

$$\alpha \nabla f(x_{k+1})^T(x_{k+1} - z) = \alpha \frac{1-\tau}{\tau} \nabla f(x_{k+1})^T(y_k - x_{k+1})$$

Then by using a convexity argument

$$\alpha \frac{1-\tau}{\tau} \nabla f(x_{k+1})^T(y_k - x_{k+1}) \leq \alpha \frac{1-\tau}{\tau}(f(y_k) - f(x_{k+1}))$$

And if we shift indices by 1, we get the same loss. So we're having a loss with respect to both mirror and gradient descents, and the two losses sum up to something at most 1: So they effectively cancel each other. We claim that there are only three steps behind Nesterov's acceleration. We have now seen all of the three, and it's only a matter of parameter tuning at this point.

So $f(y_k) - f(x_{k+1}$ is how much you gain from gradient descent. Before we choose τ, α, let me mention that there are only essentially three lemmas we have used: Gradient Descent, Mirror Descent, and a Convexity argument. If we combine all three, it's only a matter of parameter tuning. The first two are classical, so the only tricky one is the third method.

So now, if we combine these, we see that

$$\frac{\alpha^2}{2}\|\nabla f(x_{k+1})\|^2 \leq \alpha^2 L(f(x_{k+1}) - f(y_{k+1}))$$

Then, we can expand out again the third equation

$$\alpha\nabla f(x_{k+1})^T(x_{k+1}-u) \leq \alpha\frac{1-\tau}{\tau}(f(y_k)-f(x_{k+1}))+\alpha^2 L(f(x_{k+1})-f(y_{k+1}))+\frac{1}{2}\|z_k-u\|^2-\frac{1}{2}\|z_{k+1}-u\|^2$$

Note that we can do the telescoping in a few different ways. For instance, we could set the first two terms equal to each other. But there is a slight problem: You then have to take the average in the end, since you have an upper bound on the average. It's fine, but there will be a few issues. A slightly more convenient statement for choosing parameters is to choose τ is given as follows (Also note that every single parameter we've been talking about depends on k).

$$\frac{\alpha_k}{\tau_k} = \alpha_k^2 L$$

In this way, we claim that all of our terms will telescope in a slightly different way. So you will need epoching here. There are some unusual issues. In terms of producing the theorem, if we choose α, τ satisfying the previous equation, then we can write

$$\alpha\nabla f(x_{k+1})^T(x_{k+1}-u) \leq (\alpha_k^2 L-\alpha_k)(f(y_k)-f(x_{k+1}))+\alpha_k^2 L(f(x_{k+1})-f(y_{k+1}))+\frac{1}{2}\|z_k-u\|^2-\frac{1}{2}\|z_{k+1}-u\|^2$$

and then the x terms get canceled, leaving only

$$(\alpha_k^2 L - \alpha_k)(f(y_k) - f(u)) + \alpha_k^2 L(f(u) - f(y_{k+1})) + \frac{1}{2}\|z_k - u\|^2 - \frac{1}{2}\|z_{k+1} - u\|^2$$

Now we had better make sure the constant term of the next iteration cancels the constant term of the previous iteration. This corresponds to something like choosing $\alpha_k = \frac{k}{2L}$. Now let's state something about convergence.

After telescoping, we have

$$0 \leq \alpha_T^2 L(f(u) - f(y_t)) + \frac{1}{2}(\|z_0 - u\|^2)$$

and we can ignore the last term since we're talking about an upper bound. Then simplifying,

$$f(y_T) - f(u) \leq \frac{L\|z_0 - u\|^2}{T^2}$$

This is the accelerated converge for smooth functions, which is a $\frac{1}{T^2}$ convergence rate for smooth functions. This is because of our choice of α_k. There is no epoching here because of the way we choose the τs so that things canceled in the right way. In principle, we could cancel things the first way, take averages, and the epoch, and over different epochs change α and you will again end up with some $\frac{1}{T^2}$ rate. There will be some extra terms (think of weighted average over epochs) that will only disappear for strongly convex functions. But this other choice is much simpler. If we just use gradient descent, this convergence will just be $\frac{1}{T}$ for gradient descent. If your functions are actually strongly convex, there are other ways you can tune parameters.

3 Faster Accelerated Coordinate Descent

The second part of the talk is to deduce an even faster coordinate descent, built upon this framework. This trick will be easy to describe with the new framework.

Let me explain in two minutes how to do accelerated **coordinate descent** in the correct way without any additional proof in the correct way. Suppose our function f is coordinate-wise smooth, which means the coordinate gradient with respect to a point:

Definition 3.1. Coordinate-wise smooth.
For any coordinate i,

$$|\nabla_i f(x) - \nabla_i f(x + \delta e_i) \leq L_i \cdot \delta$$

Now, we just change our gradient descent and mirror descent updates with respect to coordinate i, and almost identical analysis will go through. Now suppose we choose a probability distribution for choosing coordinate i, call it $(p_1, \cdots, p_i, \cdots, p_n)$. Then in expectation with respect to this distribution, we get the full gradient as desired before. Essentially, that's it! We can just combine the first and second update rules again. At this point, the correct distribution to choose coordinates in this language should be proportional to $\sqrt{L_i}$. Somehow this observation was just missing in previous analysis of coordinate gradient descent, but it becomes very clear in this framework. So you can get even faster accelerated coordinate descent.

Here's a quick summary of the work: It recovers Nesterov's method, and it's very nice to understand it. But the power of this framework/view of thinking about three things and combining them, is so powerful that it helps us to build way beyond the scope that Nesterov's method could originally talk about. For instance, if the function is not so smooth, what can we do? We can also talk about updates which will increase the update and then kill things. We can also obtain very surprising results.

We can even talk about non-convex cases: We can personally view this work submitted to ICML also as a linear coupling of gradient descent. There are basically two things: strong coupling and weak coupling. If we have three steps, we can combine them to get an accelerated method. But we sometimes have to talk about things globally. But for some applications, the third step can actually be ignored, which means you only have one sequence of points instead of three. But we view its update as a gradient descent update and a mirror descent. Surprisingly, we can combine the lemmas obtained this way and obtain faster running time than any one of them alone. One of them is the non-convex work, and also positive linear programming. So the coupling without the third step is "weak coupling", and with the third step is "strong coupling".

If you only have a gradient oracle, there is a lower bound: That is the real complexity of the problem. It's a very delicate thing. It's a very good analysis technique to know, since you can get robust results from this technique. It's because of the complimentary performances between gradient descent and mirror descent, which also works with a gradient oracle. There's just no intuition in the original Nesterov analysis.

Another one of the powers of this approach is that we postpone parameter tuning to the last step.

Bubeck has a good reference on his blog regarding accelerated gradient descent for strongly convex functions, where you have a quadratic lower bound and quadratic upper bound.

Here are a few other topics we could talk about:

1. One of them is perhaps non-convex SVRG: I will talk about this at Simons Meeting next Friday in NYC. Suppose the function f is strongly convex. Then we have a quadratic lower bound at any point. If the function is smooth (gradient doesn't change too much) but **non-convex**, there is a quadratic lower bound to the function (you can add a bit of noise to become smooth). If your function is strongly convex, then you can place the second lemma with a version of mirror descent. Geometrically, it corresponds to instaed of averaging linear lower-bounding hyperplanes, you want to lower bound by averaging quadratic functions. Then, you can actually develop the non-convex version of mirror descent using quadratic lower bounds. But then you can only combine a few things to get some speedup for non-convex local optimization.

Accelerated gradient descent in general is not robust to noise (if you only have a gradient estimate), so it is useless in practice for stochastic optimization. But this analysis is really good, since you can get a lot of other things out of this. The tuning of parameters is very delicate. Stochastic descent is very different from coordinate descent, though both are randomized. Coordinate descent is working on the dual of the objective, so the variance issue goes away. You can think of doing it separately for each coordinate. You can a priori say for each coordinate how many steps you will move. You can also view alternating minimization for coordinate descent. In coordinate descent, you're saying you have an exact estimate in each direction. This is not the case in the stochastic setting. You use the fact that the estimator is in a particular direction, since it's an exact thing along a chosen direction i. It's used also in the convexity part (minor point).

4 Rotation

The third part of the talk is about rotation: We see the data is bigger and bigger, but the data is not fully random. Can we speed up runtime when there's implicit structure in the data, say in Facebook graphs? Should we spend $100x$ more time in training data? It sounds irrelevant, but actually, it's rotation on top of accelerated methods.

I will talk about this some other time.