Alg-ML Seminar

LiSSA: Linear Time Second-Order Stochastic Algorithm — February 23, 2015

Lecturers: Naman Agarwal and Brian Bullins — Scribe: Kiran Vodrahalli

Contents

1 Introduction

This is a recent work with Elad Hazan.

The most basic problem in machine learning is you have a distribution over $\{(x, y)\}$ and you have some hypothesis class H, and you'd like to find the optimal $h \in H$ via $\text{argmin}_{h\in H}\mathbf{E}_D[l(y, h(x))]$, using some loss l. Hazan: Sometimes you get very lucky and this kind of combination of things may be a breakthrough in optimization.

So you can use ERM which is basically finding the hypothesis in H which minimizes error on sample set. However, ERM tends to overfit the data, so people typically add regularization to avoid overfitting, and you sometimes restrict H to linear functions, etc. We can reinterpret as follows:

$$\min_{\theta\in\mathbb{R}^d} f(\theta) = \min_{\theta\in\mathbb{R}^d}\left\{\frac{1}{m}\sum_{i=1}^{m} f_k(\theta) + R(\theta)\right\}$$

In SVM and logistic regression, f_k is convex, and $R(\theta)$ could be L_2 or L_1. We'll assume L_2 and f_k convex for the rest of the talk.

For convenience, redefine $f_k(\theta) := f_k(\theta) + \lambda\|\theta\|_2^2$. (Basically, we need that the regularizer be strongly convex).

1.1 History of Optimization

1. First-order methods: This is basically gradient descent. This is preferred in practice because gradients are cheap to compute. $O(md)$ per step, m is samples, d is dimension. You can also make it stochastic by subsampling the gradient, which is much faster (used in practice), but theoretically it has worse guarantees. This was the starting point of first order methods, but there are ways of obtaining faster

convergence while still maintaining linear time properties. Nesterov introduced acceleration (two or three gradients to get faster convergence). There is also adaptive regularization (AdaGrad: Duchi, Hazan, Singer). There's also the idea of variance reduction: doing mini-batches, SVRG: GD + SGD (mixing SGD and gradient steps). There's also dual coordinate descent, but it's really the same line of work of SVRG.

2. Second-order methods: Here we want to use ∇^2, the Hessian. Newton's method is the canonical example; however, Hessian computations are expensive. It has fast convergence (quadratic convergence near the optimum). The per step time is $\mathcal{O}(md^2 + d^3)$, which is too expensive in practice, where d^3 is the Hessian inversion step. More advanced second methods involving subsampling the Hessian, and get an unbiased estimator at any point. You can extend this to mini-batches of samples to get a better estimate (NewSamp paper from NIPS 2015). There are some issues with it though: The solution is not satisfying, since your estimator is no longer unbiased ($\mathbf{E}[\tilde{X^{-1}}] \neq \nabla^{-2}$). Moreover, you still need to invert the matrix you found with your new batch. They try to fix this with low rank projection, but finding one eigenvector is still $\mathcal{O}(d^2)$ time, and you will lose out on information.

1.2 The Problem

So the question is: Can we do Newton's method in linear time while obtaining similar guarantees. So we present LiSSA is a subsampled Newton's method which is linear time for usual ML applications, and enjoys linear convergence to the optimum.

Gradient descent gives linear for strongly convex function, but this result is still better (we will see why). There is a very simple way to perform fast multiplication of the inverse with the gradient.

We propose a novel unbiased estimator for this. Note that our notation for Hessian inverse is ∇^{-2}.

First recall Taylor expansion for PSD matrix A is $A^{-1} = \sum_{i=0}^{\infty}(I - A)^i$ where $\|A\| \leq 1$.

2 Estimator of the Inverse

Pick a probability distribution p_i over non negative integers. Select some $\hat{i} \sim p_i$. Then select $X_1, \cdots, X_{\hat{i}}$ indepdnent samples of the Hessian, and define

$$\tilde{\nabla}^{-2} f(x) = \frac{1}{p_{\hat{i}}} \prod_{j=1}^{\hat{i}} (I - X_j)$$

Basically we pick how many of the terms in the power series (where to truncate, this is $\hat{i}$). Then you sample $\hat{i}$ Hessians, but the distribution is **over the series**.

Then we clearly see

$$\mathbf{E}[\tilde{\nabla}^{-2} f(x)] = \sum_{i=0}^{\infty} p_i \left(\frac{1}{p_i} \prod_{j=1}^{i} \mathbf{E}[I - X_j] \right) = \sum_{i=0}^{\infty} (I - \nabla^2 f)^i$$

Now, you can pick many distributions over the i. It turns out the best estimator for variance over samples is picking uniform $p_i = 1/S$. This was our first attempt. This is picking out one term randomly in the series. But what if you want to go all terms **up to** a term.

If you want to pick a single term up to i, you need i samples, if you want all up to i, you need i^2 samples.

So we can recursively represent the Taylor series as

$$A^{-1} = I + (I - A)(I + (I - A)(...))$$

You could potentially use Chebyshev polynomial to write it in shorter form; you could then potentially accelerate things. But Chebyshev won't be noise stable. But a caveat is you cannot take a full estimate of the gradient. Taking the average of many gradient descents does work; this seems to be important. You're basically averaging at every step.

Then, set $X_0 = I$, and define

$$X_i = I + (I - \tilde{\nabla}^2 f)X_{i-1}$$

and via the recursive definition $\mathbf{E}[X_i] \to \nabla^{-2} f$, and for the same cost we get to use more terms of the series to get our estimator.

We will use this estimator for the rest of the talk.

3 The Algorithm

Now, let's just run Newton's method with the estimator we constructed. We will later explain why this is linear time. That is,

$$x_{t+1} = x_t - \tilde{\nabla}^{-2} f(x)\,()$$

The meat of the algorithm is in a concentration step. We step over time $t = 1, \cdots, T$: We're going to have S_1 samples of our estimator (concetration: making many samples of the same estimator). For each of these samples, we will sample S_2 copies of the Hessian (we're constructing the estimator). A critical aspect is that we're using **several** unbiased estimators of the product of the inverse, and then we're averaging those. To make it clear this is a linear time operation, we see that in our recursion, we already have the matrix inverse product is **already included**. For typical machine learning applications, this is a rank 1 matrix, which allows our product to be calculated in **linear time**.

We want to take S_2 large enough where ϵ is small enough. We only need S_2 samples to take a degree S_2 approximation of our Taylor series representation of the inverse.

3.1 Running Time

Note that $\tilde{\nabla}^2 f$ is of the form $\alpha I + \nabla^2 f_i$, the α is an artifact of the regularizer (condition number needs to not be too large). Commonly $\nabla^2 f_i = c x_i x_i^T$, where x_i is the i^{th} training sample. Recall that we restricted our hypothesis class to regularizations that are basically rank-1 losses: ridge regression, logistic, etc. The Hessian is rank 1 matrix for all of these cases. Therefore each update step $(I - \tilde{\nabla}^2 f)\nabla f$ can be done in linear time (via rank 1 matrix). It will be actually $\mathcal{O}(\text{sparsity})$ since we're just doing vector-vector products! So even if you're not rank 1 but you're sparse, you're set! $\mathcal{O}(d)$ is worst case (note that if you remove the rank 1 condition, you're at $\mathcal{O}(d^2)$ worst-case since that's all the time it takes to multiply vector by matrix, which is the form the estimator is cast in. This is **still** an improvement upon most other methods out there, which have a factor of at least $\mathcal{O}(d^3)$ due to the requirement of matrix inversion).

We make some extra assumptions:

1. $\|\nabla^2 f_i\| \leq 1$
2. $\lambda_{min}(\nabla^2 f_i) \geq \frac{1}{\kappa_{max}}$, where κ is the condition number. (Maybe this issue can be averaged out).

Now we come to the main theorem:

Theorem 3.1. *Set $S_1 = \tilde{\mathcal{O}}(\kappa_{max}^2)$ and $S_2 = \tilde{\mathcal{O}}(\kappa)$ (truncation point of Taylor series), then after a sufficient number of gradient descent steps, we have the following per step guarantee for LiSSA with probability $1 - \delta$:*

$$\|x_{t+1} - x^*\|_2 \leq \frac{\|x_t - x^*\|_2}{2}$$

We've done some work which can reduce κ_{max}^2. Note that this bound holds irrespective of whether or not the Hessian is rank 1. We are making use of second order information.

Corollary 3.2. *For $t > T_1$, LiSSA returns a point x_t such that with probability at least* $1 - \delta$

$$f(x_t) \leq \min_x f(x^*) + \epsilon$$

in total time $\log(1/\epsilon)\Big(\mathcal{O}(md) + \tilde{\mathcal{O}}(\kappa_{max}^2 \kappa d)\Big)$. *The number of gradient calls is independent of the condition number; it's* $\log(1/\epsilon)$.

Note that this is an impossible theorem to get for first order methods!

The runtime for LiSSA is $\mathcal{O}\left(m + \kappa\kappa_{max}^2 d \log(1/\epsilon)\right)$, which is much better than gradient descent, which is $m\kappa$, accelerated gradient descent, which is $m\sqrt{\kappa}$, SVRG (which behaves erratically near optimum) ($\mathcal{O}(m + \kappa)$).

We don't have at true Expectation result: Whatever you can get with expectation, you can get with high probability.

Remark 3.3. The number of iterations here is **independent of the condition number**, it only co mes into the sampling, which is already very pessimistic!

4 An Alternative Interpretation

We can think of Newton's method as minimzing local quadratic optimization. If we apply gradient descent to a quadratic function, we actually get

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

which is the exact same recursive formulation as our estimator.

You could then replace gradient descent with stochastic gradient descent, the Hessian with an estimator of the Hessian, so you can see that our estimator is a special case of performing a specific type of gradient descent (full) on the quadratic function $x^T(\nabla^2 f)x + \nabla f^T x + c$. So you can't get the same bounds with stochastic gradient descent via the analysis, since you need to do the averaging for the analysis. In the analysis, they just use spectral norm concentration bounds. If you use SGD, you won't get logarithmic convergence guarantee in the sample size bound. Lower bounds hold for the scenario where you use stochastic information about the gradient which would not allow this to work.

5 Experimental Results

Not surprisingly, Newton's method has a nice quadratic curvature (it converges quadratically in terms of iterations), but LiSSA still converges linearly in iterations. In time however, LiSSA is much better, since its per iteration complexity is much smaller. We tried out SVM on MNIST for our testing. NewSamp (2015 NIPS: Their theorems are weaker, and they still invert matrices (on a low rank projection - which loses information), and they only have expectation bounds instead of in high probability) is worse in both cases, and is much slower (you had to project down to a low number of dimensions even to get the slower algorithm!)

In practice, we take S_1 to be very small (recall that this is the number of models we draw for our spectral concentration bounds). Typically $1 \leq S_1 \leq 10$.

6 Comments

The importance of this result to some extent depends on how much you care about using second-order methods. The work demonstrates that LiSSA has better guarantees and empirical performance than other second-order methods; however, second-order methods take the same amount of time as stochastic gradient descent in practice when you only care about getting within $1/10$ or $1/100$ of your optimal value. The real gains show when you're aiming for 10^{-4} level of accuracy and above. Then, LiSSA converges a lot more quickly in terms of time rather than iterations. However, in many machine learning applications, you may not care about such accuracy convergence, since in practice you are training on a regularized objective in the first place which is only a proxy for the "true" correct objective.

However, the other important aspect of this result is it allows a fully second-order method to be considered feasible for other applications which before were too costly. There are problems with using Newton's method on non-convex functions, since Newton's method will tend towards saddle points and so on, so the Hessian information needs to be taken into account well before such second-order methods can be used for something like the training of neural networks. It's unclear whether or not there would be a benefit, as stochastic gradient descent seems to work well enough and is very quick. However, it is possible that advancements can be made to lead to escaping saddle points. Currently, Rong Ge authored the only theoretical work which tackles the problem of escaping saddle points. There may be more connections to be made here.