Stochastic gradient methods

Waldemar Hebisch

January 18, 2022

1 Motivation

Motivation:

- Optimizing functions computed by Monte Carlo simulations
- randomness can improve convergence in non-convex case
- large scale machine learning
- noisy input data (treating it as random perturbation of true data)

1.1 Monte Carlo example

Consider simple Monte Carlo integration. We want to optimize

$$F(x) = \int f(x, y) d\mu(y)$$

where μ is a high-dimensional measure such that we can not compute the integral exactly. Instead we assume that we can generate random y_j with distribution μ. At core of Monte Carlo methods are procedures that can generate such (pseudo) random points with reasonable efficiency. This is rather nontrivial but for several interesting μ we know good methods.

We have

$$F(x) \approx \frac{1}{N} \sum_{j=1}^{N} f(x, y_i).$$

Since we only get approximate values of F optimizing it may look quite difficult. But we also have

$$\nabla F(x) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_x f(x, y_j)$$

so we can use approximate value of ∇F in say gradient descent.

1.2 Toy non-convex example

Consider $f(x) = x^2 + \sin(Nx)^2$ where $N = 10000$. Then $f'(x) = 2x + 2N\cos(Nx)\sin(Nx) = 2x + N\sin(2Nx)$. From this is easy to see that for $|x| < N/2$ distance to closest local minimum is of order 2π. So, once $|x| < N/2$ gradient descent is likely to converge to closest local minimum giving us value of goal function of order of $N^2/4$, which is rather large.

How can we improve this? Simple idea: add random number to gradient of f. That is instead of $\nabla f(x)$ use $\nabla f(x) + r$ where r is random variable. To have real effect r must be enough, roughly of order N. In such case instead of convergence to local minima we will get random walk.

With enough randomness we can hope that expectation of $\nabla f(x) + r$ will be close to average of $\nabla f(x)$ in a neighbourhood of x which in turn will be close to gradient of nonoscillatory part, that is x^2. Over long intervals average of gradient of part is 0. For concreteness we can take as r Gaussian random variable with $E(r^2) = \log(N)N^2$. To avoid very large random jumps we must take relatively small learning rate α, say $\alpha = N^{-2}$. Assuming for simplicity that expectation of $\nabla f(x) + r = \nabla x^2$ and using our later result we get

$$E(f(x_{[2\log(N)N^2]}) - f(x_\infty)) \le C\log(N)$$

where $[\cdot]$ means integer part.

At first the above look quite bad: we need $2\log(N)N^2$ steps. However, Lipschitz constant of f' is of order N^2. Second derivative f'' has indefinite sign, but regular part is $(x^2)'' = 2$ which suggests learning rate of order $1/N^2$. So using gradient descent we also expect large number of steps.

In one dimension simple alternative strategy is evaluate f at several points say at distance 1. Such strategy needs N steps for search of critical region $|x| < N/2$ and will give error not larger than 2. So in much smaller number of steps than method with randomized gradient we can get better result. However, we can build similar function on $\mathbb{R}^n$ by formula $g(x_1, \ldots, x_n) = f(x_1) + \cdots + f(x_n)$. Now, number of local minima is of order $(N/2\pi)^n$, so for $n > 2$ even naive randomized gradient works better.

We can substantially reduce number of steps needed in randomized approach by initially using α of order 1 and decreasing it after substantial decrease of goal function. Also, we can use variance reduction methods.

1.3 Machine learning

We want to fit model to data. x represent model parameters, y represent input points, z expected value. $f(x, y, z)$ measures how well model with parameters x fits to input y with value z. Given N input, value pairs we want minimize

$$\frac{1}{N}\sum_{j=1}^{N} f(x, y_j, z_j)$$

More concrete example could be linear regression where A is n by m matrix $y \in \mathbb{R}^m$, $z \in \mathbb{R}^n$ and

$$f(A, y, z) = \|Ay - z\|^2 + \phi(A)$$

where $\phi(A)$ is penalty for undesired features of A.

For large n and m evaluating $Ay - z$ may be large task. If N is also large, then single evaluation of f' may be quite expensive. Namely, single matrix-vector product has cost nm, so doing this N times costs nmN operations. Reasonable values:

- $m = 10000 = 10^4$
- $n = 1000 = 10^3$
- $N = 10000000 = 10^8$

Product $nmN = 10^{15}$, not small task even for modern machines. And we may expect several thousands of iterations for convergence, so clearly cheaper method is desirable.

Actually, when sufficient computing power is available, then much bigger values of parameters are useful.

Simple-minded savings:

1. use less data, that is smaller N
2. use simpler model, that is smaller m
3. stop optimization early to reduce number of iterations
4. lowering output size n usually is *not* an option, this is part of requirements

We need enough samples to get reasonable view of data. For noisy data we want enough samples to reduce error by averaging. So clearly, there is lower limit on N, such that with smaller N performance will be inadequate. In modern settings frequently we are able to collect large amount of data, much larger than minimal N, so we optimize N to make training easier — clearly we do not want too large N.

Bigger models allow better fit to data. To generalize we want information stored in the model to be much smaller than training data. So, with given data there is optimal model size. But with large amount of available data we want as big models as we can reasonably train.

Stopping optimization early, far from optimal value is frequently used. But we still may need large number of iterations to get reasonable performance.

Clearly, we would like optimization methods with lower costs for large models.

2 General stochastic gradient

To reduce computational cost we could try to approximate ∇F just using a sample of y_j, z_j. In extreme case in single step we just select j at random and use $\nabla_x f(x, y_j, z_j)$. Combined with gradient descent this gives stochastic gradient descent (SGD):

1. start from arbitrary x_0
2. select j randomly with uniform probability
3. $$x_{i+1} = x_i - \alpha_i \nabla_x f(x_i, y_j, z_j)$$
4. if not finished goto step 2

x_i produced by SGD are random process, in fact Markov chain. For Markov chains dependence (or independence) on the past plays significant role. Stochastic gradient method uses unbiased (with respect to past) estimate of gradient:

$$E(\nabla_x f(x, y_j, z_j)|x_0, \ldots, x_{i-1}) = \frac{1}{N}\sum_{j=1}^{N} \nabla_x f(x, y_j, z_j) = \nabla F(x).$$

Earlier approaches at reducing time used to divide training data into batches, and/or use sequential passes trough data updating x on the fly during pass. They have some use but randomness simplifies theoretical analysis and shows practical benefits.

We present now abstract setup that unifies analysis for Monte Carlo methods and machine learning methods like stochastic gradient descent.

Let $g(x)$ be *random* approximation to $\nabla F(x)$. Assume

$$mI \leq \nabla^2 F(x) \leq MI$$

$$\mathrm{E}(\|g(x)\|^2) \leq D + C\|\nabla F(x)\|^2,$$

$$\mathrm{E}(g(x)) = \nabla F(x).$$

and that $g(x_i)$ is independent from previously computed values.

Lemma 2.1 *Assume that F attains minimum at x_∞ and let x_0 be arbitrary. Under assumptions above, with constant $\alpha \leq \frac{1}{MC}$ for iteration*

$$x_{i+1} = x_i - \alpha g(x_i)$$

we have

$$\mathrm{E}(F(x_i) - F(x_\infty)) \leq \frac{\alpha DM}{2m} + (1 - \alpha\frac{m}{M})^i (F(x_0) - F(x_\infty)).$$

Consequences:

- initial convergence comparable to gradient descent
- accuracy limited by variance ($\frac{MD\alpha}{2m}$ term), to get good accuracy we need small α
- suggest strategy of decreasing α when convergence slows down

Viewing x_i as Markov chain we expect convergence to stationary distribution with variance of order

$$\frac{\alpha DM}{2m}$$

and expectation x_∞. Natural idea is to average several x_i-s to be closer to optimum.

Averaging turned out to be tricky in practice, combined with diminishing step sizes of order $\frac{1}{i}$ it did not give significant improvement.

Simply decreasing step size leads to algorithm with $O(1/\sqrt{i})$ error for large i in strongly convex case. Using clever averaging scheme and slowly decreasing step size Polyak and Juditsky reduced asymptotic rate to $O(1/i)$ in strongly convex case. Without strong convexity they got $O(1/\sqrt{i})$ for convex functions. In practice averaging may be undesirable and experiments showed that usually convergence rate without averaging was similar or better than with averaging. Later Shamir and Zhang showed that small modification to stochastic subgradient method has asymptotic error $O(\log(i+1)/i)$ in strongly convex case and $O(\log(i+1)/\sqrt{i})$ for convex functions, so now theoretical arguments for use of averaging are quite weak (theoretically by averaging we can get rid of $\log(i+1)$ factor.

In short:

- smooth strongly convex F, simply decreasing step: $O(1/\sqrt{i})$
- smooth strongly convex F, Polyak and Juditsky: $O(1/i)$
- strongly convex (possibly non-smooth) F: $O(1/i)$ (the same rate as in smooth case but algorithm and analysis slightly different)
- convex (possibly non-smooth) F: $O(1/\sqrt{i})$

Asymptotic results would suggest that there nothing more to do: we know theoretically best method. However, situation in machine learning is different:

- we are interested in relatively small i, where asymptotic rate is irrelevant
- constants hidden in big O notation matter
- we have more information, in principle we have choice of computing ∇f exactly
- with limited data accuracy is limited by data noise
- with lot of data accuracy is limited by computer time

3 Stochatic gradient in machine learning, variance reduction

Let us now reformulate our model. We assume that F is a sum

$$F(x) = \frac{1}{N}\sum_{j=1}^{N} f_j(x)$$

This is a bit more general than our previous machine learning example. Namely when $f_j(x) = f(x, y_j, z_j)$ we get previous version but in principle dependence of f_j on j could be more complicated than change of numeric parameter.

3.1 SAG (Stochastic average gradient)

Stochastic gradient method uses simple unbiased estimate of gradient, the idea is to find (possibly biased) estimate with lower variance. One idea is to compute all ∇f_j at start and remember last computed ∇f_j for each j. Denote by $t(j,i)$ the last i for which we computed $\nabla f_j(x_i)$. We estimate derivative computing $\nabla f_j(x_i)$ for randomly selected j and using remembered values for other j:

$$g(x_i) = \frac{1}{N}\sum_{j=1}^{N} \nabla f_j(x_{t(j,i)})$$

At first glance this requires summing all ∇f_j, but we can perform update incrementally.

Namely denoting by l value of j selected at step i we have

$$\frac{1}{N}\sum_{j=1}^{N} \nabla f_j(x_{t(j,i)}) = \frac{1}{N}(\nabla f_l(x_l) - \nabla f_l(x_{t(j,i-1)})) + \frac{1}{N}\sum_{j=1}^{N} \nabla f_j(x_{t(j,i-1)})$$

so

$$g(x_i) = g(x_{i-1}) + \frac{1}{N}(\nabla f_l(x_i) - \nabla f_l(x_{t(j,i-1)}))$$

This leads to SAG (Stochastic average gradient) method:

1. start from arbitrary x_0, for j from 1 to N put $v_j = \nabla f_j(x_0)$
2. $$g_0 = \frac{1}{N}\sum v_j$$
3. put $x_1 = x_0 - \alpha_0 g_0$
4. select j at random, put
$$u = \nabla f_j(x_i),$$
$$g_i = g_{i-1} + \frac{1}{N}(u - v_j),$$
$$v_j = u$$
$$x_{i+1} = x_i - \alpha_i g_i$$

5. if not finished goto step 4

This is biased, that is g_i tends to be close to g_{i-1}, but has low variance which goes to 0 when i goes to infinity. To see why consider case when each f_j is strongly convex quadratic

$$f_j(x) = \frac{1}{2}\langle A_j x, x\rangle + \langle b_j, x\rangle$$

Then

$$\nabla f_j(x) - \nabla f_j(x_\infty) = A_j(x - x_\infty)$$

and

$$g_i = \frac{1}{N}\sum_{j=1}^{N}(\nabla f_j(x_{t(j,i)}) - \nabla f_j(x_\infty)) = \frac{1}{N}\sum_{j=1}^{N} A_j(x_{t(j,i)} - x_\infty)$$

where we could introduce $-\nabla f_j(x_\infty)$ terms because

$$0 = \nabla F(x_\infty) = \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_\infty)$$

Now, iterates of Markov chain tend to be weakly correlated so

$$E(\|g_i\|^2) \approx C\frac{1}{N}\sum_{j=1}^{N} E(\|x_{t(j,i)} - x_\infty\|^2)$$

which tends to 0 when iterates converge to 0.

Note: our reasoning above is circular (to show that $E(\|x_{t(j,i)} - x_\infty\|^2)$ is small we need to show that $E(\|g_i\|^2)$ is small), so it is not a proof (actual proof is more complicated). But our reasoning shows that there are no obstacle to convergence to 0. Namely, compare to SGD:

$$g_i = \nabla f_j(x_{i-1}) = A_j(x_i - x_\infty) + c_j$$

The c_j terms do not depend on x and when first term is very small c_j terms lead to nonzero variance

$$\mathrm{E}(\|g_i\|^2) \approx \mathrm{E}(\|c_j\|^2) = \frac{1}{N}\sum_{j=1}^{N}\|c_j\|^2.$$

Due to bias, convergence proofs for SAG are complicated, but there are strong theoretical convergence results and SAG works well in practice.

3.2 SAGA

We can use slightly different update rule to get unbiased g_i. Namely, using formula

$$g_i = (\nabla f_j(x_i) - \nabla f_j(x_{t(j,i-1)}) + \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_{t(j,i-1)})$$

we get

$$E(g_i|x_0,\ldots,x_{i-1}) = E(\nabla f_j(x_i) - \nabla f_j(x_{t(j,i-1)})|x_0,\ldots,x_{i-1})$$

$$+\frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_{t(j,i-1)})$$

$$= \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_i) - \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_{t(j,i-1)}) + \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_{t(j,i-1)})$$

$$= \frac{1}{N}\sum_{j=1}^{N}\nabla f_j(x_i) = \nabla F(x_i).$$

This leads to SAGA, variation of SAG which is unbiased:

1. start from arbitrary x_0, for j from 1 to N put $v_j = \nabla f_j(x_0)$
2. $$w_0 = \frac{1}{N}\sum v_j$$
3. put $g_0 = w_0$, $x_1 = x_0 - \alpha_0 w_0$
4. select j at random, put
$$u = \nabla f_j(x_i),$$
$$g_i = (u - v_j) + w_i$$
$$w_i = w_{i-1} + \frac{1}{N}(u - v_j),$$
$$v_j = u$$
$$x_{i+1} = x_i - \alpha_i g_i$$
5. if not finished goto step 4

Convergence proofs for SAGA are simpler than for SAG. Variance of g_i is higher, but in similar way as for SAG we can argue that variance goes to 0 when x_i is close to optimum. In practice SAGA seem to be comparable to SAG.

Both in theory and in practice convergence of SAG and SAGA strongly depends on initial point: this is the only point where derivative was computed exactly. One can usually improve convergence by performing several iterations of SGD and then switching to SAG or SAGA.

3.3 SVRG

Both SAG and SAGA need to store past gradient for each f_j. When N is large this may require too much storage. SVRG (Stochastic Variance Reduced Gradient), trades a little extra computation to reduce storage use. SVRG works in stages (epochs). At start of each epoch SVRG computes and remembers full derivative. In following steps of epoch started at time l SVRG selects j at random and uses

$$g_i = \nabla f_j(x_i) - \nabla f_j(x_l) + g_l.$$

Like SAGA this means that g_i gives unbiased estimate of gradient.

To implement SVRG compared to SGD we need extra storage for x_l, g_l. In each iteration we need additionally to compute $\nabla f_j(x_l)$, so this is approximately twice the cost of SGD. Since SVRG recomputes full derivative at start of each epoch it suffer less than SAG or SAGA from bad initial point. Still, since SGD is faster per iteration it makes sense to run several iterations of SGD before switching to SVRG.

3.4 SARAH

Another algorithm is SARAH (Stochastic recursive gradient algorithm). It uses update rule

$$g_{i+1} = g_i + \nabla f_j(x_i) - \nabla f_j(x_{i-1})$$

where j is chosen at random. Like SAG it is biased, but there are convergence proofs.

3.5 Proximal versions

One can also consider proximal versions of algorithms above, that is assume

$$F(x) = \frac{1}{N}\sum_{j=1}^{N} f_j(x) + h(x)$$

and use proximal step to compute new x_i:

$$x_{i+1} = \mathrm{prox}_{\alpha_i h}(x_i - \alpha_i g_i).$$

For SAG and SARAH convergence of proximal version is not clear. For SGD, SAGA, SVRG proximal versions converge with similar rate as non-proximal case.

3.6 Acceleration

Since there is lower bound acceleration can not improve asymptotic convergence rate. For sufficiently regular functions we can hope for faster convergence before variance starts to play role. Also, acceleration may work to reduce variance (instead of averaging). At the moment situation is not clear: there are reports that in some cases acceleration gave big improvements, in other cases no improvement was observed.

3.7 Summary

New stochastic method appear regularly and one can combine existing methods in various ways. For example one can consider variant of SVRG that divides index set in subsets and remembers derivative for each subset. That would give faster update at cost of extra storage.

There is also theoretical activity, finding new properties of existing methods. So state of the art may change in few years.

4 Stochastic view of gradient descent

Typical convergence bounds we met were of worst case kind: convergence was warranted for all functions and all initial data satisfying our conditions. However, in practice we are interested in average cases. Method that is fast in most cases but may be slow in some exceptional cases may be preferable to method that consistently gives performance corresponding to worst case. For example, quicksort algorithm may have quite bad behaviour, heapsort works essentially as worst case bound predicts. In practice quicksort is usually preferred because it gives lower average execution time. So it makes sense to look at expected running time of algorithms that we met.

Today we will present simple average case result for gradient descent. To simplify analysis we consider only quadratic function

$$f(x) = \frac{1}{2}\langle Ax, x\rangle + \langle b, x\rangle + c$$

with positive definite A.

In theoretical analysis we look at $x - x_\infty$, so after shift we have simply

$$f(x) = \frac{1}{2}\langle Ax, x\rangle.$$

Now consider random initial data. For multidimensional problems it is reasonable to assume that initial data have normal distribution. The best we can hope is that expectation will be the same as optimal point x_∞. After shift above this leads to normal distribution with expectation 0.

If no coordinate plays distinguished role it it natural to assume that normal distribution has covariance matrix which is multiple of identity matrix. By simple rescaling, which changes numerical values but does not change qualitative picture we can assume that covariance matrix is identity. Now, positive definite matrix can be diagonalized using orthogonal matrices. Orthogonal matrices preserve normal distribution with 0 expectation and identity covariance matrix, so we can consider diagonal A. For concreteness take $1, 2, \ldots, n$ as eigenvalues.

It is an exercise to prove that gradient descent with constant α such that $0 < \alpha < \frac{2}{n}$ satisfies:

$$\mathrm{E}(\|x_i\|^2) = \sum_{j=1}^{n} \exp(2i\beta_j).$$

where $\beta_j = \log(|1 - j\alpha|) < 0$. Now, β_j depend in somewhat complicated way on α. But

$$\beta_j < -j\alpha$$

when $1 - j\alpha \geq 0$ and similar estimate is valid when $1 - j\alpha < 0$. So we can estimate sum of terms with $1 - j\alpha \geq 0$ from above by

$$\sum_{j=1}^{n} \exp(-2ij\alpha)$$

Assuming for simplicity that $|1 - n\alpha| \leq 1 - \alpha$ we can use the same estimate for terms with $1 - j\alpha < 0$.

So

$$\mathrm{E}(\|x_i\|^2) \leq 2 \sum_{j=1}^{n} \exp(-2ij\alpha)$$

This sum is a geometric progression and estimating it by infinite sum we get

$$\mathrm{E}(\|x_i\|^2) \leq 2 \exp(-2i\alpha) \frac{1}{1 - \exp(-2i\alpha)}$$

Now, for $i < n$ and α of order $1/n$ we see that $\exp(-2i\alpha)$ is of order 1 and $1 - \exp(-2i\alpha)$ is of order $2i\alpha$ so

$$\mathrm{E}(\|x_i\|^2) = O(\frac{n}{i})$$

Since $\mathrm{E}(\|x_0\|^2) = n$ we get

$$\mathrm{E}(\|x_i\|^2) = O(\frac{1}{i}) \mathrm{E}(\|x_0\|^2).$$

It is easy to see that terms with $2ij\alpha < 1/2$ give contribution comparable to the whole sum. For such terms already when $i = 1$ our upper estimate for β_j has matching lower bound of similar form. Similarly for other estimates that we did. So the estimate from above we got is unimprovable for i which are small compared to n. In other words, we can expect that with unfavourable conditioning our upper bound for gradient descent (the one without assuming strong convexity) in initial stage is close to optimal.

For $2i\alpha > 1$ we can rewrite our estimate as

$$\mathrm{E}(\|x_i\|^2) \leq C(1 - 2\alpha)^n \leq C(1 - 2\alpha)^n \frac{1}{n} \mathrm{E}(\|x_0\|^2)$$

so for large i compared to upper estimate we gain factor of $\frac{1}{n}$ and have slightly better constant in exponential term.

Remark: Strongly convex estimate for $2i\alpha \leq 1$ is weaker than estimate without strong convexity. Our estimate for expected value is blend of of estimate for case without string convexity with improvement for large i.