

Training Models by Stochastic Gradient Descent

36-473/673, Fall 2025

9 September 2025 (Lecture 5)

Contents

1 Steps towards Numerical Optimization	2
2 Gradient Descent	3
2.1 There's Too Much Data	3
3 Stochastic Gradient Descent	4
3.1 Wrinkles / Complications	5
4 “The Trade-offs of Large Scale Learning”	7
5 Stochastic Approximation, Recursive Estimation, and Online Learning	8
6 Flows (optional)	10
7 Further Reading	11
8 References	11

For most parametric probability models, there is no closed-form expression for the maximum likelihood estimator. We might be able to get a complicated, nonlinear set of equations for the gradient which needs to equal zero, $\nabla L_n(\hat{\theta}) = 0$, but solving those equations for the MLE $\hat{\theta}$ is itself hard; it might not even be feasible to get those equations explicitly, and we just have a function $L_n(\theta)$ to optimize. A very standard method for numerical optimization is **gradient descent**: start *somewhere*, $\theta^{(0)}$, find the gradient of the objective function there, $\nabla L_n(\theta^{(0)})$, move “downhill” in that direction a short distance, $\theta^{(1)} = \theta^{(0)} - \eta \nabla L_n(\theta^{(0)})$, and repeat the cycle at $\theta^{(1)}$, until we find a (local) minimum or get tired. This requires us to calculate the gradient ∇L_n , which means taking one derivative per parameter. That can be bad enough, if there are a lot of parameters, but the log-likelihood is a sum over n data points (and the cross-entropy is an average over n data points), which means calculating every derivative takes a time proportional to n . If, however, we pick *one* observation uniformly at random from the data set, the gradient of the negative log probability at that observation is an unbiased (but noisy) estimate of the gradient of the cross-entropy. (If the model is IID, an “observation” is a data point; if the model is a Markov chain, an “observation” is a transition from one state to the next.) **Stochastic gradient descent** is (basically) choosing a new random observation to calculate the gradient at each optimization step. This makes our calculation of the MLE sloppier than if we could exactly solve the equations, but this computational error can be made small in comparison to the statistical error which is intrinsic to *only* having n data points.

1 Steps towards Numerical Optimization

We have our favorite parametric model which gives us a probability, or probability density, for the data, $m(x; \theta)$. This in turn gives us a likelihood function for the parameters θ , treating the data as fixed, $L_n(\theta)$. In principle, we know how to find the maximum likelihood estimator: take the derivatives of L_n with respect to the coordinates of θ , set them all to zero, and solve:

$$\nabla L_n(\hat{\theta}_n) = 0 \tag{1}$$

In the sort of textbook situation you've encountered in your statistical education so far, including most of Lecture 4, you can actually solve these estimating equations, to get an explicit formula for the MLE. It's always very gratifying when this works, but the sad truth is that there are very few problems where things work out so neatly¹.

We are thus forced to think about how to solve estimating equations numerically. But if we are going to do *that*, we might as well think about numerical methods for optimizing functions in the first place.

The most basic approach is: start with a guess about the right parameter value; then make a change to the parameter value; if the change improved things, keep it, otherwise try again, or give up. Criteria for giving up include: “have tried too many changes”, “have been working at this for too long”, “the function is below some acceptable threshold” (**satisficing**), and/or “the last few changes to the function have been pretty small” (**convergence**). Of course, this raises the question of how we decide on what to change about the parameters, and by how much.

This is where we come back to the estimating equations, a.k.a. first order conditions. They tell us that at the optimum a *sufficiently small* change in any direction is going to make basically zero difference to L_n . In fact, the gradient tells us about changes at *any* point θ in the parameter space. Let's say we're currently at θ , and we contemplating moving a small distance $0 < \epsilon \ll 1$ in the direction $\vec{u} = (u_1, \dots, u_d)$, which we represent as a length-1 vector in the parameter space. Then

$$L_n(\theta + \epsilon \vec{u}) \approx L_n(\theta) + \epsilon \sum_{i=1}^d u_i \frac{\partial L_n}{\partial \theta_i}(\theta) \tag{2}$$

$$= L_n(\theta) + \epsilon \vec{u} \cdot \nabla L_n(\theta) \tag{3}$$

Since we want to minimize L_n , we'd like to make the change $L_n(\theta + \epsilon \vec{u}) - L_n(\theta)$ as negative as possible, so a little thought (or geometry) says that $\vec{u} = -\frac{\nabla L_n}{\|\nabla L_n\|}$. In words, the negative gradient $-\nabla L_n(\theta)$ gives the direction of **fastest descent** at θ . If we want to minimize the function by making small tinkering changes, but we want to do that quickly, a reasonable sounding idea is to find the gradient and start going in the *opposite* direction.

¹In HW1, you worked out the MLE for the Pareto distribution, where $m(x; \alpha, x_0) \propto x^{-\alpha}$ for $x \geq x_0$ and is 0 otherwise, and where x is continuous. The corresponding discrete distributions is known as the **Zipf** distribution, and the normalizing factor is $\zeta(\alpha, x_0) = \sum_{k=x_0}^{\infty} k^{-\alpha}$, a special function rejoicing in the name of the “Hurwicz zeta function”. The MLE involves the ratio $\zeta'(\alpha, x_0)/\zeta(\alpha, x_0)$ (Seal 1952); you could call this a closed-form solution if you like. If we go back to continuous x but change to $m(x; \alpha, s) \propto (x + s)^{-\alpha}$, one gets a pair of simultaneous rational equations for $\hat{\alpha}$ and $\hat{s}$, or one transcendental equation for $\hat{s}$ (Maguire, Pearson, and Wynn 1952).

2 Gradient Descent

This gives the algorithm of **gradient descent**:

- Start at $\theta^{(0)}$, $t = 0$
- while ((making progress) and (not too tired))
 - Find $G_t = \nabla L_n(\theta^{(t)})$
 - Set $\Delta_t = -\eta G_t$
 - Set $\theta^{(t+1)} \leftarrow \theta^{(t)} + \Delta_t$
 - Increment t by 1
- Return final $\theta^{(t)}$

Here η is a control setting, known variously as the **step-size** or **learning rate**. Criteria for “not too tired” are basically “ t is not too big”, while “making progress” usually cashes out as some combination of “ G_t is not too close to zero” and/or “ $L_n(\theta^{(t)}) - L_n(\theta^{(t-1)})$ is not too close to zero”.

Setting η is a bit of an art. Bigger step-sizes can lead to faster convergence, but also to the possibility of over-shooting an optimum and having to back-track or zig-zag in on it, or even (for especially bad situations) orbiting around an optimum forever. Lowering η straightforwardly increases the number of iterations needed to find a well-behaved minimum².

Gradient descent uses the first derivatives of L_n , so it’s called a **first-order** optimization method. **Second-order** methods use second derivatives as well. The most important of these are **Newton’s method** and its relatives, which you can think of as a clever way of eliminating the “but how do I set η ?” problem: make the step-size the inverse of the second derivative. If the second derivative is small, so the first derivative isn’t changing rapidly, take big steps in the fastest-descent direction. If the second derivative is large, then the first derivative *is* changing rapidly, and take a much smaller step in the same direction³.

Second-order methods are incredibly common in practice, but are *not* much used for the sort of generative models we’re primarily interested in, because they have truly ridiculous numbers of parameters. If there are d parameters, calculating $\nabla L_n(\theta)$ is going to be at least an $O(d)$ operation. Calculating $\nabla \nabla L_n(\theta)$ is going to be an $O(d^2)$ operation, and usually we’d need to invert this matrix, which would take $O(d^3)$ time. (There are tricks to speed up inversion or avoid explicit inversion altogether, but none of them make a second-order step any faster than $O(d^2)$. If d is in the millions or billions, $O(d^2)$ is prohibitive, even with lots of parallelization. The growth in the number of parameters thus drives practice back towards first-order methods.

2.1 There’s Too Much Data

$L_n(\theta)$ is an average over n data points; so therefor is $\nabla L_n(\theta)$:

$$\nabla L_n(\theta) = -\nabla \frac{1}{n} \sum_{t=1}^n \log m(x_t | x_{0:t-1}; \theta) \quad (4)$$

$$= -\frac{1}{n} \sum_{t=1}^n \nabla \log m(x_t | x_{0:t-1}; \theta) \quad (5)$$

Thus the time-complexity of calculating $\nabla L_n(\theta)$ isn’t just $O(d)$, it’s really $O(nd)$. This means that as n grows, each iteration is getting more and more time-consuming. You *could* try parallelizing it, but if $n \approx 10^{12}$ that’s a lot of computers.

²If we imagine the limit of letting $\eta \rightarrow 0$ while simultaneously allowing the maximum number of iterations to go to infinity, we end up with an ordinary differential equation, $\frac{d\theta}{dt} = -\nabla L_n(\theta)$. Generally speaking, if you could solve that equation, you wouldn’t need to do numerical optimization, but some people (physicists) find this helps their intuition for what gradient descent is doing, and it can be useful theoretically.

³This is a bit of a simplification, since “the second derivative” is really a Hessian matrix, and one multiplies the gradient by the inverse Hessian, which not only alters the length but can also rotate the direction. But it does get the right idea across...

3 Stochastic Gradient Descent

It's at this point that we get to stochastic gradient descent. Suppose we pick on single random data point I , uniformly over $1 : n$. Let's evaluate the cross-entropy as though this were the only data point; we get

$$-\log m(x_I | x_{0:I-1}; \theta) \quad (6)$$

whose expectation value is precisely

$$-\frac{1}{n} \sum_{i=1}^n \log m(x_i | x_{0:i-1}; \theta) = L_n(\theta) \quad (7)$$

Similarly, if we take the *gradient* of the negative log probability at x_I ,

$$-\nabla \log m(x_I | x_{0:I-1}; \theta) \quad (8)$$

the expected value is

$$-\frac{1}{n} \sum_{i=1}^n \nabla \log m(x_i | x_{0:i-1}; \theta) = -\nabla L_n(\theta) \quad (9)$$

So the gradient at x_I is an unbiased estimate of the full-data gradient. (If we wanted to go there, the Hessian at x_I is an unbiased estimate of the full-data Hessian, and so on through all the higher derivatives.)

Here then is **stochastic gradient descent** (SGD):

- Start at $\theta^{(0)}$, $t = 0$
- while ((making progress) and (not too tired))
 - Pick I uniformly at random in $1 : n$
 - Find $G_t = -\nabla \log m(x_I | x_{0:I-1}; \theta^{(t)})$
 - Set $\Delta_t = -\frac{\eta}{t+1} G_t$
 - Set $\theta^{(t+1)} \leftarrow \theta^{(t)} + \Delta_t$
 - Increment t by 1
- Return final $\theta^{(t)}$

In homework 3, you will write simple, but working, code to implement SGD.

Notice that we've done two things differently than in ordinary gradient descent: we're using one randomly chosen data point (indexed by I) to calculate the gradient, rather than the full data, *and* we're multiplying that gradient by $\frac{\eta}{t+1}$ rather than just η , so we're making smaller and smaller changes at each step, all else being equal.

The traditional, and correct, explanation for why we introduce the factor of $1/(t+1)$ is to imagine what would happen if we *didn't* have that factor, but $\theta^{(0)}$ was, in fact, at the optimum⁴. (It could happen!) G_0 is an unbiased estimate of the true gradient, so G_0 would be zero *in expectation*, but it would still have some variance, which would kick us away from the optimum even if we'd landed on it. If we moved by ηG_t , then we'd do a random walk which would drift away from the optimum. If η is small, $\text{Var}[G_t] \approx \text{Var}[G_0]$ for not-too-big t , so we'd expect⁵ to end up a distance $\pm \eta \sqrt{\text{Var}[G_0] t}$ away from the optimum after t steps, *even if* we started there. The factor of $1/(t+1)$ in the step-size is, however, enough to suppress this behavior. (This is *especially* true if we don't start at the optimum, but reach it after some substantial number of steps.)

⁴Incidentally, you will see many sources which multiply by $1/t$, instead of $1/(t+1)$, because they start the optimizer's internal step index at 1 instead of 0. Even those who index from zero will often *talk* about "1/t scaling", because it's just easier to say.

⁵Consider a random walk in one dimension, where $X_t = \sum_{i=1}^t Z_i$ and the increments of the walk Z_i are all IID with mean 0 and variance σ^2 . A little character-building algebra shows that $\mathbb{E}[X_t] = 0$ for all t , but $\text{Var}[X_t] = t\sigma^2$, so the root-mean-square distance from the origin at time t , $\sqrt{\mathbb{E}[X_t^2]} = \sigma\sqrt{t}$. Thus a random walk drifts away from its starting point, with the typical distance after t steps being $\propto \sqrt{t}$. This humble little calculation turns out to explain a remarkably wide range of physical (Einstein 1956) and even biological (Berg 1984) phenomena, and to have deep mathematical ramifications (Spitzer 1964).

3.1 Wrinkles / Complications

The “vanilla” SGD I have just explained, and which you will implement in the homework, is too straightforward an idea to have been left alone. There are three sorts of elaborations or complications which people have imposed on it, and which are important enough to mention here. All three can be combined in various ways.

3.1.1 Mini-Batches (Sampling More Than One Observation)

One elaboration is to do what’s often called **mini-batch** gradient descent. Instead of sampling *one* random observation I , one samples $I_1, I_2, \dots, I_k$, for some fixed $k \ll n$, and takes the average of their gradients. This average will also be an unbiased estimate of the full-data gradient. The advantage of this over sampling a single observation is that the variance around that expected value is reduced by a factor⁶ of $1/k$. The disadvantage is that it takes more computing time and memory (by a factor of k).

3.1.2 Stochastic Second-Order Methods

Any second-order optimization method, which uses both first and second derivatives, has a stochastic counterpart. In a stochastic Newton’s method, for instance, we would calculate both the first and second derivative at the observation I , and scale a gradient step by the inverse Hessian. Second-order methods generally converge in fewer iterations, but each iteration is much more computationally expensive; that remains true for their stochastic versions.

3.1.3 Momentum

Rather than just using the current gradient at I , we can do a moving average of recent gradients, or even a moving average of *all* previous gradients:

- Start at $\theta^{(0)}$, $t = 0$
- while ((making progress) and (not too tired))
 - Pick I uniformly at random in $1 : n$
 - Find $G_t = -\nabla \log m(x_I | x_{0:I-1}; \theta^{(t)})$
 - Set $\Delta_t = -\frac{\eta}{t+1} G_t + \beta \Delta_{t-1}$
 - Set $\theta^{(t+1)} \leftarrow \theta^{(t)} + \Delta_t$
 - Increment t by 1
- Return final $\theta^{(t)}$

Here β is a control setting between 0 and 1. This has the effect of making Δ_t a weighted average of all previous randomly-sampled gradients, but with the weight on old gradients decaying exponentially fast as they age. This can give some of the advantages of mini-batching with less computational overhead, and *may* help the optimizer to (as it were) coast through local minima. In practice, β is often rather large, say 0.9.

For some time now, the most commonly-used optimizer for neural networks is a form of stochastic gradient descent with momentum known as “Adam”, introduced⁷ by Kingma and Ba (2015). It’s slightly more complicated than my pseudo-code above, since it uses some information about the second moments of gradients, but not by much.

⁶Strictly speaking, the variance goes down by a factor of exactly $1/k$ only if we do sampling with replacement, so the same observation could show up multiple times among the $I_1, \dots, I_k$. In practice, people prefer to do sampling without replacement, making the formula for the variance reduction more complicated. (The reason is that the lack of replacement *creates* correlations among the sample values. To see this, suppose $k = 2$, and the gradient at I_1 is *above* the full-data gradient. Then the [conditional] expected gradient at I_2 must be *below* the full-data gradient.) However, the upshot of the more complicated algebra is that the variance-reduction factor approaches $1/k$ as $k/n \rightarrow 0$. (If we draw a very small sample from a very big set, whether we do so with or without replacement hardly matters.)

⁷Somewhat notoriously, the proofs in that paper are flawed (Reddi, Kale, and Kumar 2018), but in practice it seems to work pretty well anyway.

These three principle ways of spicing up plain SGD can be combined (mini-batch stochastic Newton's method, etc.), and these are by no means exhaustive. But, as I said, these are the most important directions in which people have made SGD more complicated.

4 “The Trade-offs of Large Scale Learning”

What follows draws heavily on Bottou and Bousquet (2012), but tries to make it (even) simpler.

When we have to resort to numerical optimization, we should not expect that any algorithm will give us the *exact* optimum. Rather, we hope that the algorithm will return a solution whose value is close to the optimal value, and we should hope that the value will improve the more iterations we give the algorithm. We should be less concerned about the *location* of the optimum, for both mathematical⁸ and substantive⁹ reasons.

All this is true for any optimization problem, but there is an additional wrinkle for *statistical* optimization problems. This is because, when we optimize based on sampled data, the function we’re optimizing is not function we really *want* to optimize. We’re minimizing some function of the *data*, $L_n(\theta)$, but what we really want to minimize is our model’s divergence from the data-generating *process*, $\lambda(\theta)$. The data are just an imperfect, noisy proxy for the true process, full of useless details of no future relevance. Accordingly, past some point, extra effort spent optimizing L_n only advances our true goal by accident¹⁰.

This, it turns out, gives a powerful reason to prefer stochastic optimization methods for large data sets. For well-behaved (convex) problems with d parameters, gradient descent will return a value within δ of the optimal value in $O(\log 1/\delta)$ iterations, each of which takes $O(nd)$ computational operations. On the same problems, stochastic gradient descent takes $O(1/\delta)$ iterations, each of which takes $O(d)$ operations. On the other hand, the gap between the $L_n(\theta)$ we’re optimizing and the $\lambda(\theta)$ we wish we could optimize is generically of $O(\sqrt{d/n})$. (See Lecture 2.) This suggests that δ should be $O(\sqrt{d/n})$ as well; if it’s much smaller, we’re (probably) wasting our computational resources. The computational cost to optimizing to within that tolerance for gradient descent is thus

$$O(nd \log \sqrt{n/d}) = O(nd \log n - nd \log d) \quad (10)$$

(Recall that $\log x^\alpha = \alpha \log x$, and multiplicative constants get “eaten” by the big- O .) On the other hand, for stochastic gradient descent the computational cost is

$$O(d/\sqrt{d/n}) = O(\sqrt{nd}) \quad (11)$$

If we demand a tighter tolerance δ , say $\delta = O(d/n)$ (which would match the “optimism” or $\mathbb{E} [\lambda(\hat{\theta}_n) - L_n(\hat{\theta}_n)]$, see Lecture 2 again), we get a total computational cost for gradient descent of

$$O(nd \log (n/d)) = O(nd \log n - nd \log d) \quad (12)$$

versus

$$O(d/(d/n)) = O(n) \quad (13)$$

for stochastic gradient descent. Either way, we are looking at a considerable savings in computational effort by using *stochastic* optimization.

⁸Recall that near a well-behaved interior minimum at θ^* , the objective function will look quadratic, $L(\theta) \approx L(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T L''(\theta^*)(\theta - \theta^*)$. (I’m writing out the 1D version for simplicity, you can double-check everything with vector calculus on your own.) If we pick some small δ , $0 < \delta \ll 1$, and insist on coming within δ of the *value* of the optimum, that therefore implies $\frac{1}{2}(\theta - \theta^*)^T L''(\theta^*)(\theta - \theta^*) \leq \delta \Rightarrow \theta - \theta^* = O(\sqrt{\delta})$. But then $\sqrt{\delta} \gg \delta$ (because δ is small). Turned around, if we demand coming within ϵ of the *location* of the optimum θ^* , that requires finding a solution whose value is within $O(\epsilon^2)$ of the optimal value, and $\epsilon^2 \ll \epsilon$ (because ϵ is small). Either way, we see that close approximation of the location is harder than close approximation of the value, precisely because the gradient is zero at the optimum and so there’s a neighborhood around it where the value isn’t changing very rapidly.

⁹As Butler (1986) wrote, “in the predictive setting, all parameters are nuisance parameters”. Less epigrammatically, the models we’re concerned with in this course are deeply mis-specified, and none of their parameters have any sort of interesting or important real-world interpretations. (Or, at best, uncovering those interpretations is itself a difficult research project.)

¹⁰If L_n only matches λ to within ± 1 , coming within $\pm 10^{-6}$ of the optimum of L_n is unlikely to be *really* better than coming within, say $\pm 10^{-2}$.

5 Stochastic Approximation, Recursive Estimation, and Online Learning

Stochastic gradient descent belongs to a broad family of what have come to be called **stochastic approximation** algorithms, which began with Robbins and Monro (1951) and Kiefer and Wolfowitz (1952). There are of course many things which could be called “stochastic approximation”, but as a jargon term it’s come to mean “adjusting an estimate based on a sequence of random observations, without retaining the whole set of observations”.

Robbins and Monro (1951) was concerned with the following problem (slightly modifying their notation): We want to find the solution θ^* to the equation $M(\theta) = \alpha$, for scalar-valued θ and M . The difficulty is that the function $M(\theta)$ is the *expected value* of something we observe when we adjust our control to θ , say $\mathbb{E}[Y(\theta)] = M(\theta)$. You could imagine taking *many* observations $Y_1(\theta), \dots, Y_n(\theta)$ at one θ and averaging them to get something $\approx M(\theta)$, then trying a new θ and taking many observations there, etc., to build up a (noisy) picture of the function $M(\theta)$, and then try to solve the equation (perhaps interpolating between the θ where data were collected or something). What Robbins and Munroe wanted to do instead was take *one* observation at each θ , but still have a sequence of θ s converging to θ^* . The procedure they proposed was iterative: Start with an arbitrary $\theta^{(0)}$. At step t , we observe $Y_t = Y(\theta^{(t)})$, a *single* draw from a distribution whose expectation is $M(\theta^{(t)})$, and then we compare that to the desired level α :

$$\theta^{(t+1)} = \theta^{(t)} + \eta_t(\alpha - Y_t) \quad (14)$$

Already in that first paper, they showed consistency, $\theta^{(t)} \rightarrow \theta^*$, for a wide range of M functions when $\eta_t = \eta/(t+1)$.

At the beginning of these notes, I reminded you that (interior) optima can be found by taking derivatives, setting them to zero, and solving. In particular, if $Y(\theta) = -\frac{d}{d\theta} \log m(x_I | x_{0:I-1}; \theta)$ for a randomly-chosen I in our data set, and the desired level $\alpha = 0$, then Robbins-Munroe gives us stochastic gradient descent.

(The first people to explicitly turn Robbins-Munroe-style stochastic approximation into an optimization method were Kiefer and Wolfowitz (1952). There the problem was to maximize $L(\theta)$, where we get to observe a $Z(\theta)$ such that $\mathbb{E}[Z(\theta)] = L(\theta)$, and every observation is statistically independent. (Again, θ was taken to be one-dimensional.) Their scheme was, in our notation,

$$\theta^{(t+1)} = \theta^{(t)} + \frac{\eta}{t+1} \frac{Z(\theta^{(t)} + c_t) - Z(\theta^{(t)} - c_t)}{c_t} \quad (15)$$

with $c_t = (t+1)^{-1/3}$ (or something proportional to that). This has the advantage, over SGD, of only requiring samples of values for the objective function, not its derivatives; as a consequence, this procedure can even handle some ill-behavior on the part of the objective function $L(\theta)$, e.g., non-differentiability at the optimum.)

The reason statisticians were interested in these problems is that they were ways of doing estimation *without* keeping all of the data around to do computations on. One only had to remember the last estimate, $\theta^{(t)}$, and the latest observation / random sample, Y_t . This was important because at the time, in the 1950s and 1960s, automatic computation was at once possible and quite expensive, and in particular data storage was expensive (and slow). Ways to simplify the computations, especially to reduce the amount of data that needed to be stored and recalled, thus had *considerable* value.

Accordingly, the early efforts of Robbins, Munroe, Kiefer, Wolfowitz and co. quickly bloomed into a thriving field of stochastic approximation. This considered modifications such as vector-valued θ and/or M , better techniques for optimization, and much more. (Albert and Gardener (1967) gives an incisive survey from the period.) In particular, they got applied to a lot of problems relating to dynamics and time series, such as determining the orbits of satellites (or rockets...) from successive observations (Albert and Gardener 1967, sec. 8.5).

This link of work in turn connects to what’s called **on-line learning**, or rather to one of the things known by that name, as distinct from offline, **batch learning**. Ordinarily, in statistics, we tend to think of having a *complete* data set available to analyze, and so are interested in fitting a model to an entire “batch” of

data. In contexts like signal processing, on the other hand, it makes more sense to think of the data arriving incremental, over time, and one needs *some* model to deal with it as it comes in¹¹. Statistically, on-line learning amounts to, or is a form of, **recursive estimation**, where the estimate we provide with n data points, $\hat{\theta}_n$, is a function of the new data x_n and the previous estimate $\hat{\theta}_{n-1}$, and not (directly) of $x_{1:n-1}$. Many forms of recursive estimation are *not* stochastic approximation (e.g., most of those discussed in Kulhavý (1996)), but all stochastic approximation *is* recursive estimation (Nevel’son and Has’minskiĭ, n.d.), as is on-line learning. The connections between on-line learning and stochastic approximation, especially stochastic gradient descent, were well-appreciated during the second wave of neural network research, in the 1980s and 1990s (Bottou 1998).

With fast enough computing, and in particular with very cheap data storage, there might seem to be no *need* for special attention to on-line learning, because one could always just apply a “batch-mode” estimation algorithm to all the data which have arrived to date. When a new data point arrives, just re-run the batch-mode estimator from scratch; problem solved. This, however, can run into trouble in (at least) three ways:

1. The batch-mode estimator might take longer to run than the time interval between data points arriving.
2. The batch-mode estimator might *slow down* as we get more data. (Think of running a linear regression: we need $(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$, and calculating both $\mathbf{x}^T \mathbf{x}$ and $\mathbf{x}^T \mathbf{y}$ takes $O(n)$ time. If we have to re-do this from scratch whenever we get a new observation [= new row in the data frame], the total time taken for the first n observations is $O(n^2)$.)
3. Keeping all the data in working memory, and/or retrieving it from storage, might be too slow and expensive.

I’ve talked already about how, in the 1950s and 1960s, working with more than *one* observation at a time was so expensive as to motivate stochastic approximation algorithms. As computing got faster, and in particular as memory and storage got cheaper and faster, we entered a phase where it was, in fact, practical to simply load the whole data set into memory and work on it all at once¹². This situation changed yet again in the 2010s, when the volume of data available for many analyses became *so* large that working with it all at once became impractical, *and* it imposed drastic slow-downs on batch-mode estimators. This coincided with the third, currently on-going wave of interest in neural networks (“deep learning”), and led to the renaissance of stochastic gradient descent. Whether the balance of computational considerations will tilt away from stochastic approximation again in the future is an interesting question, which fortunately we do not have to settle.

¹¹Of course there are statistical contexts where data arrive incrementally, such as sequential experiments where we can decide, based on the data, what the next experimental condition should be, or even whether to halt the experiment altogether.

¹²To give a sense of the transformation, Ripley (1981), p. 191–192, discusses ways to do statistical image analysis without having to load the entirety of a “vast” picture file, consisting of a whole 128 kb.

6 Flows (optional)

If we take gradient descent and make η very small, we should let it run for a very large number of steps. As we make η smaller and smaller, but run gradient descent for longer and longer, the paths traced out through parameter space are going to be more and more similar to each other. They will approach a limiting trajectory. (Heuristically, but only heuristically that limit is what would happen if we took infinitely many, infinitely small gradient-descent steps.) The limiting trajectory is given by the ordinary differential equation

$$\frac{d}{dt}\theta = -\nabla L_n(\theta) \quad (16)$$

Dynamical systems like this, where both time and state are continuous, and the rate of change in the state is a function of the current state, are called **flows**. This is an example, more specifically, of a **gradient flow**, where the rate of change¹³ of the state vector is the gradient of some scalar function (here, L_n). The **fixed points** of the flow are, precisely, the points where $\nabla L_n(\theta) = 0$. A **locally stable** fixed point is one where any trajectory which starts sufficiently near the fixed point will approach, and it's easy to convince yourself that those correspond to local minima. Because there's a lot of mathematical machinery for analyzing flows, especially gradient flows, considering this limit can be very helpful for working out the theoretical issues related to gradient descent at small (but > 0) levels of η .

The limiting flow is rarely a useful way of *computing* the optimum, however. Think about how you'd actually solve the differential equation numerically. The simplest ("Euler") scheme would be to pick some very small increment of time h and say

$$\theta(t+h) \approx \theta(t) - h\nabla L_n(\theta(t)) \quad (17)$$

but this is just gradient descent! There are more sophisticated ways of numerically solving differential equations (e.g., the Runge-Kutta method), but these rarely improve on simple gradient descent as ways of rapidly reaching a fixed point.

(The limit of *stochastic* gradient descent is a differential equation perturbed by random noise. We will look at such **stochastic differential equations** later in the course.)

¹³Incidentally, by re-defining the "time" variable, from t to $\tau = t/b$, we get $\frac{d}{d\tau}\theta = -b\nabla L_n(\theta)$, so we can have any (positive) proportionality constant we like on the right-hand side of the equation. This doesn't change the *curve* which will be traced out in parameter space from a given initial θ , just how "fast" the curve is traversed. Setting $b = 1$ is just a convention, but it does make book-keeping simpler.

7 Further Reading

There are many excellent reference works on optimization algorithms. For our purposes, Wright and Recht (2022) can be recommended as recent, clear, and specifically interested in applications to statistics and machine learning. (Chapter 5 is specifically about SGD.)

As remarked, the section “The Trade-offs of Large Scale Learning” is a (deliberate) simplification of the paper by the same title, Bottou and Bousquet (2012). I refer interested readers to that excellent work for some details and caveats, considerations of second-order methods, and more refined analyses that take into account learning-theoretic generalization bounds, instead of just my back-of-the-envelope formulas.

I am not aware of any serious history of stochastic approximation. I have tried to indicate some of the original context, as I am aware of it, but I suspect that there is a story to be told about Cold War and the Space Race, on both sides of the Iron Curtain. (Kalman filtering, a different form of recursive estimation, definitely had such a story — see McGee and Schmidt (1985); both Robbins and Monro (1951) and Kiefer and Wolfowitz (1952) were funded by the U.S. Office of Naval Research; etc.)

8 References

(The R Markdown bibliography process is screwing up dates for translated works)

Albert, Arthur E., and Jr. Gardener Leland A. 1967. *Stochastic Approximation and Nonlinear Regression*. Cambridge, Massachusetts: MIT Press. <https://doi.org/10.7551/mitpress/6464.001.0001>.

Berg, Howard C. 1984. *Random Walks in Biology*. Second. Princeton, New Jersey: Princeton University Press. <https://doi.org/10.1515/9781400820023>.

Bottou, Léon. 1998. “On-Line Learning and Stochastic Approximations.” In *Online Learning of Neural Networks*, edited by David Saad, 9–42. Cambridge, England: Cambridge University Press. <https://doi.org/10.1017/CBO9780511569920>.

Bottou, Léon, and Olivier Bousquet. 2012. “The Tradeoffs of Large Scale Learning.” In *Optimization for Machine Learning*, edited by Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, 351–68. Cambridge, Massachusetts: MIT Press. <https://doi.org/10.7551/mitpress/8996.003.0015>.

Butler, Ronald W. 1986. “Predictive Likelihood Inference with Applications.” *Journal of the Royal Statistical Society B* 48:1–38. <https://doi.org/10.1111/j.2517-6161.1986.tb01386.x>.

Einstein, Albert. 1956. *Investigations on the Theory of the Brownian Movement*. New York: Dover.

Kiefer, J., and J. Wolfowitz. 1952. “Stochastic Estimation of the Maximum of a Regression Function.” *Annals of Mathematical Statistics* 23:462–66. <https://doi.org/10.1214/aoms/1177729392>.

Kingma, Diederik P., and Jimmy Ba. 2015. “Adam: A Method for Stochastic Optimization.” In *3rd International Conference on Learning Representations [Iclr 2015]*, edited by Yoshua Bengio and Yann LeCun. <https://arxiv.org/abs/1412.6980>.

Kulhavy, Rudolf. 1996. *Recursive Nonlinear Estimation: A Geometric Approach*. Berlin: Springer-Verlag. <https://doi.org/10.1007/BFb0031830>.

Maguire, B. A., E. S. Pearson, and A. H. A. Wynn. 1952. “The Time Intervals Between Industrial Accidents.” *Biometrika* 39:168–80. <http://www.jstor.org/pss/2332475>.

McGee, Leonard A., and Stanley F. Schmidt. 1985. “Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry.” 86847. NASA Technical Memorandum. <https://ntrs.nasa.gov/citations/19860003843>.

Nevel’son, M. B., and R. Z. Has’minskii. n.d. *Stochastic Approximation and Recursive Estimation*. Providence, Rhode Island: American Mathematical Society. <https://doi.org/10.1090/mmono/047>.

- Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. 2018. “On the Convergence of Adam and Beyond.” In *6th International Conference on Learning Representations [Iclr 2018]*, edited by Yoshua Bengio and Yann LeCun. <https://arxiv.org/abs/1904.09237>.
- Ripley, Brian D. 1981. *Spatial Statistics*. New York: Wiley. <https://doi.org/10.1002/0471725218>.
- Robbins, Herbert, and Sutton Monro. 1951. “A Stochastic Approximation Method.” *Annals of Mathematical Statistics* 22:400–407. <https://doi.org/10.1214/aoms/1177729586>.
- Seal, H. L. 1952. “The Maximum Likelihood Fitting of the Discrete Pareto Law.” *Journal of the Institute of Actuaries* 78:115–21. <https://doi.org/10.1017/S0020268100052501>.
- Spitzer, Frank. 1964. *Principles of Random Walk*. Princeton: Van Nostrand.
- Wright, Stephen J., and Benjamin Recht. 2022. *Optimization for Data Analysis*. Cambridge, England: Cambridge University Press. <https://doi.org/10.1017/9781009004282>.