In this lecture we discuss the common case in learning and statistics in which you would like to optimize a parameterized function over a large number of data points:

$$\theta^* = \arg \min_{\theta} f(\theta) = \arg \min_{\theta} \sum_{i=1}^{N} E_i(\theta)$$  \hspace{0.5cm} (1)

with \(N\) very large, and \(E_i\) describes the error at the \(i\)’th data point. These types of functions arise in stochastic optimization problems, in which where the error depends on some unknown stochastic component \(x\). The standard assumption in statistics that \(x\) is generated from some unknown probability distribution \(P(x)\), but we only observe a finite sample \(x_1, \ldots, x_n\). So, we make an approximation of the form:

$$f(\theta) = \int_x E(x; \theta)P(x)dx \approx \arg \min_{\theta} \sum_{i=1}^{N} \frac{1}{N}E_i(\theta; \theta).$$  \hspace{0.5cm} (2)

which is equivalent to (1) with:

$$E_i(\theta) = \frac{1}{N}E(x_i; \theta).$$  \hspace{0.5cm} (3)

[Note: the term stochastic optimization used here is not to be confused with the optimization techniques covered last lecture, which use stochasticity to help optimize and escape local minima.]

**vs. Batch Optimization.** A “batch” optimization optimizes the function \(f\) directly using descent methods or metaheuristic algorithms. However, this
may be expensive because each evaluation of the objective function, let alone its gradient or Hessian, requires $N$ evaluations of the individual error terms.

Stochastic optimization techniques take cheaper steps by only considering small subsets of the data at once. They are particularly useful for “online” intelligent agents that observe long streams of data, and need to maintain a model of the data in real time. For these agents it becomes impractical even to store, let alone optimize over long periods of observations.

1 Common Examples

**Generalized Least Squares.** Such functions appear in the generalized least squares problem, in which you wish to match a parametric model $h(x; \theta)$ to examples of an unknown function by minimizing the sum of squared errors:

$$f(\theta) = \sum_{i=1}^{N} (h(x_i; \theta) - y_i)^2.$$  

(4)

Here, the points $x_i$ are the datapoints at which you observe the “true” function values, and $y_i$ are the function values observed at those points (together $(x_i, y_i = f(x_i))$ is known as an example). By parametric, we mean that the $\theta$ values are “knobs” that can be tuned in order to get a better fit to the data. Both linear and nonlinear models can be represented this way. Neural networks are an example of popular nonlinear models.

**Maximum Likelihood Estimation.** Another common area is in maximum likelihood estimation, in which the parametric model $h$ describes the probability of an example $x$ appearing in the dataset. Assuming each example is independent, this requires maximizing a product of $h(x_i; \theta)$ over all examples. For numerical stability, you typically maximize the log-likelihood, or in other words minimize the negative log likelihood:

$$f(\theta) = -\log P(D; \theta) = -\sum_{i=1}^{N} \log h(x_i; \theta).$$  

(5)

2 Stochastic Gradient Descent

The most widely used method for stochastic optimization is stochastic gradient descent. It is often used to train neural networks and in other learning
algorithms. The basic idea is to take small steps in each of the directions of the negated gradients of the individual $E_i$'s, rather than the batch gradient $\nabla f$.

The update rule on each iteration is to pick some index $i$ and perform the update

$$x_{t+1} = x_t - \alpha_t \nabla E_i(x_t)$$  \hspace{1cm} (6)

where $\alpha_t$ is a step size. In other words, the step is taken in order to reduce the error for a single example. Each iteration is fast because this only requires computing the gradient of a single term. The index needs to be varied on subsequent iterations to cover all of the examples, and is typically chosen round-robin or random fashion. But the step size is a critical component of the success of this method. Let us examine the effects of step size.

2.1 Step Size

It is important to understand that a step to reduce the error on example $i$ is by no means guaranteed to reduce the error on other examples. In fact, it will probably increase error for a large fraction of the examples. In a very rough sense, at the true minimum of $f$ a step will increase error for approximately half of the examples (under many assumptions, but this should give some intuition about the method). So, large steps might be very counterproductive. On the other hand, if a large number of examples “agree” roughly on a good search direction, then larger steps will help speed convergence.

The typical method of choosing a step size is to decrease $\alpha$ inversely proportional to $t$, such as $\alpha_t = \alpha_0 \max(1, 1/(1 + \lambda t - \lambda t_0))$. Here $t_0$ is a value before which $\alpha_t$ stays constant, and $\lambda$ governs the rate of decrease of $\alpha$ (the lower $\lambda$, the slower the decrease). As we will see below, this strategy helps $\alpha$ tend toward zero in the limit, but not too quickly so that convergence stalls out (e.g., as would happen if the infinite series of $\alpha_t$ converged to a finite number).

In practice, the constants $t_0$ and $\lambda$ need to be tuned to a given problem.

2.2 Convergence

In the limit of small $\alpha$’s, stochastic gradient descent is guaranteed to converge to a local minimum (for random index selection convergence occurs with probability 1). In fact the sum of the $N$ descent directions approaches the batch descent direction as $\alpha$ grows small. We will prove this here.
Let us assume for simplicity that we select examples in round-robin fashion, and for simplicity we assume the same $\alpha_t$ for the next $N$ iterations. Without loss of generality assume that at time $t$ we start at the first $i=1$:

$$x_{t+N} = x_t + \alpha_t \sum_{i=1}^{N} \nabla E_i(x_{t+i-1})$$

Now perform a Taylor expansion of each $\nabla E_i(x)$ around $x_t$.

$$\nabla E_i(x_{t+i-1}) = \nabla E_i(x_t) + \nabla^2 E_i(x_t)(x_{t+i-1} - x_t) + O(||x_{t+i-1} - x_t||^2)$$

If $\nabla E_i$ is bounded for all $i$, then we can treat $x_{t+i-1} - x_t$ simply as $O(\alpha_t)$. So if $\nabla^2 E_i$ is bounded, then $\nabla E_i(x_{t+i-1}) = \nabla E_i(x_t) + O(\alpha_t)$.

Replacing this expression in the above equation we get

$$x_{t+N} = x_t + \alpha_t \sum_{i=1}^{N} [\nabla E_i(x_t) + O(\alpha_t)]$$

$$= x_t + \alpha_t \sum_{i=1}^{N} \nabla E_i(x_t) + O(\alpha_t^2)$$

$$\approx x_t + \alpha_t \sum_{i=1}^{N} \nabla E_i(x_t)$$

$$= x_t + \alpha_t \nabla f(x_t)$$

because $\alpha_t$ is assumed to be small.

3 Response Surface Methods

A more sophisticated method is response surface optimization, which is based on the response surface methodology developed in 1951 for optimizing design parameters for experiments performed by hand. The method requires sampling a set of $> n$ “test sites” around the current $x_t$, and performing “experiments” by picking one of the $E_i$’s to evaluate for each test site. A proxy function (usually a linear or quadratic) is then fit to the data via least-squares, and a step is taken toward the minimum of the proxy function. This approach works well when the objective function is taken to be the mean value of the $E_i$’s, and that the $E_i$’s at a given point are roughly distributed according to a Gaussian distribution, so that with a sufficient number of test sites, the proxy function approaches a good fit to the true mean.
4 Recursive Estimators

An idea that is related to stochastic optimization is that of a recursive estimator. In some simple cases, it is possible to perform an exact update of the solution to (1) as a new datapoint is observed. In other words, if we let the optimal solution for \( N \) points be denoted \( \theta_N \), we can derive an optimal solution for \( \theta_{N+1} \) once \( x_{N+1} \) is observed. Like stochastic optimizations, these are used in the on-line setting that deals with continuously streaming data.

**Recursive mean estimation.** The simplest case is that of the mean. The mean \( \bar{x}_N \) of a set of points \( \{x_1, \ldots, x_n\} \) is actually the least squares estimator

\[
\bar{x}_N = \arg \min_x \sum_{i=1}^N (x - x_i)^2.
\]  
(10)

(Confirm this fact by computing the gradient of the above expression.) If we now let \( \theta_N \equiv \bar{x}_N \), we can see that

\[
\theta_{N+1} = \theta_N + \frac{1}{N+1}(x_{N+1} - \theta_N)
\]  
(11)

which gives a way to maintain the mean of a set \( N \) while only storing two numbers: the current mean and \( N \).

It turns out that any expression of the form

\[
\theta_N = \theta_{N-1} + \alpha_N(x_N - \theta_{N-1})
\]  
(12)

with \( \alpha_N \) satisfying the two conditions

\[
\sum_{i=1}^\infty \alpha_N^2 < \infty
\]  
(13)

\[
\sum_{i=1}^\infty \alpha_N = \infty
\]  
(14)

also converges to the mean of the set as \( N \) grows large. This expression is often used in online learning, in particular reinforcement learning.

**Recursive least squares.** The solution to the least-squares problem

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^N (x_i^T \theta - y_i)^2
\]  
(15)
for examples \((x_i, y_i)\), with \(x_i\) an \(n\) dimensional vector, is given by the well-known expression

\[
\theta^* = (A^T A)^{-1} A^T b
\]  

(16)

with \(A\) being the \(N \times n\) matrix of \(x_i\)’s in its rows and \(b\) being the vector of \(y_i\)’s. The recursive least squares procedure gives us a way of computing an update to \(\theta^*\) given a new example \((x_{N+1}, y_{N+1})\). Let the original \(\theta^*\) be denoted \(\theta_N\), and the original \(A\) and \(b\) be denoted \(A_N\) and \(b_N\), respectively. After receiving the new example, \(A_N\) is augmented with a new row that contains to \(x_{N+1}\), and \(b_N\) is augmented with a new entry \(y_{N+1}\). Let these new matrices be denoted \(A_{N+1}\) and \(b_{N+1}\). We can express the new estimate:

\[
\theta_{N+1} = (A_{N+1}^T A_{N+1})^{-1} A_{N+1}^T b_{N+1}
\]  

(17)

in terms of the prior estimate as follows. First, we have

\[
A_{N+1}^T b_{N+1} = A_N^T b_N + x_{N+1} y_{N+1}.
\]  

(18)

Next, we have

\[
A_{N+1}^T A_{N+1} = A_N^T A_N + x_{N+1} x_{N+1}^T.
\]  

(19)

So, if we store the quantities \(q_N = A_N^T b_N\) and \(P_N = A_N^T A_N\), we get the efficient recursive expressions:

\[
q_{N+1} = q_N + x_{N+1} y_{N+1} \quad P_{N+1} = P_N + x_{N+1} x_{N+1}^T \quad \theta_{N+1} = P_{N+1}^{-1} q_{N+1}.
\]  

(20)

This eliminates the need to store \(N\) examples entirely. Even better, the Sherman-Morrison-Woodbury update can be used to store and update the inverse of \(P_N\), which leads to an \(O(n^2)\) update.

5 Exercises

1. 