

# CSE 6740 Lecture 22

## How Do I Optimize the Parameters? (Unconstrained Optimization)

Alexander Gray

`agray@cc.gatech.edu`

Georgia Institute of Technology

- ① Unconstrained Optimization: General
- ② Unconstrained Optimization: Sum-of-Squares

# Unconstrained Optimization: General

Optimizing an objective function without constraints, for general functions.

# Optimization

Learning or training the parameters of a model generally boils down to optimization. There is a general theory for this which covers many of the cases we need in machine learning.

The general form of optimization problem we'll consider is:

$$\text{Find } x^* = \arg \min_{x \in \mathbb{R}^D} f(x) \quad (1)$$

$$\text{subject to } c_i(x) = 0, \quad i = 1, \dots, M \quad (2)$$

$$c_i(x) \geq 0, \quad i = M + 1, \dots, N. \quad (3)$$

Maximization and minimization are interchangeable by negating  $f(x)$ , so we'll assume we're always minimizing.

# Unconstrained Optimization Cases

- Root-finding
- Univariate functions
- Multivariate smooth functions
  - Second-derivative methods
  - First-derivative methods
  - Non-derivative methods
- Multivariate non-smooth functions
- Sums of squares
- Latent-variable functions

# Smooth: Canonical Algorithm

Assume for now that  $f(x)$  is twice differentiable. Let  $x_k$  be the current estimate of  $x^*$ . A *descent method* imposes the *descent condition*, that on the  $k^{\text{th}}$  iteration  $f(x_k) < f(x_{k-1})$ . We'll use the shorthand  $f_k \equiv f(x_k)$ .

All of the methods we'll consider have this general form of iteration:

**Test for *convergence*.** If the conditions are satisfied, stop and return  $x_k$ .

**Compute a *search direction*.** A vector  $p_k$ .

**Compute a *step length*.** A scalar  $\alpha_k$ , such that  $f(x_k + \alpha_k p_k) < f(x_k)$ .

**Update the estimate of the *minimum*.** Set  $x_{k+1} = x_k + \alpha_k p_k$ .

# Smooth: Canonical Algorithm

Let's denote the gradient as

$$g(x) \equiv \nabla f(x) = \left( \frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_D} \right)^T, \quad (4)$$

and the Hessian as

$$H(x) \equiv \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_D} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_D \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_D^2} \end{pmatrix}. \quad (5)$$

The Hessian matrix of  $f(x)$  can also be described as the Jacobian matrix of  $g(x)$ . Higher-order derivatives are rarely needed in standard optimization methods.

# Smooth: Canonical Algorithm

To satisfy the descent condition,  $p_k$  and  $\alpha_k$  must have certain properties. One way to ensure it is to require that  $p_k$  be a *descent direction* at  $x_k$ , i.e. a vector satisfying

$$g_k^T p_k < 0, \quad (6)$$

roughly because by Taylor's Theorem,

$$f(x_k + p) \approx f_k + g_k^T p. \quad (7)$$

The step length  $\alpha_k$  must have the property that

$$f(x_k + \alpha_k p_k) < f(x_k). \quad (8)$$

# Convergence Rate

It can be shown under some conditions that the canonical algorithm converges, *i.e.* the sequence  $\{x_k\} \rightarrow x^*$ , but the efficiency of the method depends on the number of iterations required.

The sequence  $\{x_k\}$  is said to converge with *order*  $r$  when  $r$  is the largest number such that

$$0 \leq \lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} < \infty. \quad (9)$$

If  $r = 1$ , we say we have linear convergence. If  $r = 2$ , we say we have quadratic convergence.

# Convergence Rate

If the sequence  $\{x_k\}$  has order of convergence  $r$ , the *asymptotic error constant* is the value  $\gamma$  that satisfies

$$\gamma = \lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r}. \quad (10)$$

When  $r = 1$ ,  $\gamma$  must be strictly less than 1 in order for convergence to occur. If  $\gamma = 0$  when  $r = 1$ , we call it *superlinear* convergence. A convergence rate greater than 1 implies superlinear convergence. Quadratic convergence means, roughly, that eventually the number of correct digits in  $x_k$  doubles at each step.

# Smooth: Second-Derivative Methods

Suppose we have the second derivative of  $f$ . We can approximate  $f$  using Taylor's Theorem by

$$f(x_k + p) \approx f_k + g_k^T p + \frac{1}{2} p^T H_k p. \quad (11)$$

The minimum of the right-hand side will be achieved if  $p_k$  is a minimum of the quadratic function

$$\phi(p) = g_k^T p + \frac{1}{2} p^T H_k p. \quad (12)$$

# Smooth: Second-Derivative Methods

It can be shown that a stationary point, a point where the gradient is zero, of  $\phi(p)$  satisfies the linear system

$$H_k p_k = -g_k. \quad (13)$$

A method choosing  $p_k$  this way is called a *Newton method*.

Newton's method converges quadratically to  $x^*$  if  $x_0$  is sufficiently close to  $x^*$ , the Hessian matrix is positive definite at  $x^*$ , and the step lengths  $\{\alpha_k\}$  converge to 1.

## Smooth: Second-Derivative Methods

A symmetric matrix  $A$  is *positive definite* if all its eigenvalues are positive; then for any vector  $x$ ,  $x^T Ax > 0$ , and all the diagonal elements are positive. If all the eigenvalues are non-negative we say it is *positive semi-definite*. If some are positive and some negative we say it is *indefinite*.

If the Hessian is not positive definite, the quadratic model may not have a minimum and the method will not converge. In other words, this approximation is a poor one for  $f$ .

To deal with this, we can instead use a *related* matrix which is positive definite. Under certain conditions this will allow the method to converge. We can obtain such a matrix from factorizations such as the SVD and Cholesky factorization.

# Smooth: First Derivative Methods

What if we don't have the ability to use second derivatives?

Consider the first-order approximation

$$f(x_k + p) \approx f_k + g_k^T p. \quad (14)$$

Given a step size, we'd like to choose  $p$  so that  $g_k^T p$  is large and negative, but somehow not consider large multiples of the same direction. One way to find  $\min_p \frac{g_k^T p}{\|p\|}$ , which for the norm

$\|p\| \equiv \sqrt{p^T p}$  yields

$$p_k = -g_k. \quad (15)$$

Using this choice of direction yields the *steepest-descent method*.

# Smooth: First Derivative Methods

Unfortunately steepest-descent has linear convergence, and is slow in practice.

It turns out we can make a *discrete Newton algorithm* which is essentially equivalent, including quadratic local convergence, by approximating the Hessian matrix by finite-differences of the gradient.

A forward-difference approximation to the  $i^{\text{th}}$  column of  $H_k$  is given by the vector

$$z_i = \frac{g(x_k + he_i) - g(x_k)}{h} \quad (16)$$

where  $h$  is the *finite-difference interval* and the  $i^{\text{th}}$  unit vector  $e_i$  is the *finite-difference vector*.

# Smooth: First Derivative Methods

The matrix  $Z$  whose  $i^{\text{th}}$  column is  $z_i$  will not be symmetric in general, so to approximate the Hessian we use

$$\hat{H}_k = \frac{1}{2} (Z + Z^T). \quad (17)$$

How to choose  $h$ ? It should go to zero as  $\|g\|$  goes to zero, but not be so small as to cause numerical problems. It is its own art. However, it is not critical for approximating the Hessian.

# Smooth: First Derivative Methods

*Quasi-Newton methods* represent another approach, where the local curvature information provided by the Hessian matrix is computed on the fly without explicitly forming the Hessian matrix.

Let  $s_k \equiv x_{k+1} - x_k$  be the step taken from  $x_k$ , and let  $z_k \equiv g_{k+1} - g_k$  denote the change in gradient. Expanding the gradient function about  $x_k$ ,

$$g(x_k + s_k) = g_k + H_k s_k + \dots \quad (18)$$

The curvature of  $f$  along  $s_k$  is given by  $s_k^T H_k s_k$ , and can be approximated using only first-order information:

$$s_k^T H_k s_k \approx (g(x_k + s_k) - g_k)^T s_k. \quad (19)$$

# Smooth: First Derivative Methods

Based on this, we have the *quasi-Newton condition* for an approximate Hessian  $\hat{H}_k$ , which will start as the identity matrix and keep getting updated as we get curvature information:

$$\hat{H}_{k+1}s_k = z_k. \quad (20)$$

One derivation which enforces symmetry yields the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) update rule:

$$\hat{H}_{k+1} = \hat{H}_k - \frac{1}{s_k^T \hat{H}_k s_k} \hat{H}_k s_k s_k^T \hat{H}_k + \frac{1}{z_k^T s_k} z_k z_k^T. \quad (21)$$

# Smooth: First Derivative Methods

What if we don't even want to store a matrix like  $\hat{H}_k$  at all?

The *conjugate gradient* method can achieve this, by only using evaluations of the gradient vector.

It can also be viewed as a method for finding the solution of a linear system, without explicitly computing the elements of the matrix, only products of matrix and a vector.

Conjugate gradient is formally superlinear but behaves linearly in practice.

# Smooth: No-Derivative Methods

Suppose  $f$  is a smooth function, but we don't have the first derivative, perhaps because the function value is the result of a complex calculation, like a simulation.

We can replace the gradient by a finite-difference approximation, whose  $j^{\text{th}}$  component is defined by

$$\widehat{g}_j(x_k) = \frac{f(x_k + he_j) - f(x_k)}{h}. \quad (22)$$

In this setting, choosing a good  $h$  is important. Fortunately, good methods exist.

# Non-Smooth: No-Derivative Methods

Suppose  $f$  is not a smooth function, *i.e.* and this is why we don't have the first derivative.

We'll make an algorithm which only uses evaluations  $f(x)$  of the function, which we'll call the *polytope algorithm*, more often known as the “simplex method”.

At each iteration we have  $n + 1$  points  $x_1, x_2, \dots, x_{n+1}$ , which we can think of as vertices of a polytope, ordered such that  $f_{n+1} \geq f_n \geq \dots f_1$ .

# Non-Smooth: No-Derivative Methods

Let  $c = \frac{1}{n} \sum_i x_i$  denote the centroid of the best  $n$  points. On each iteration a new *reflection point*  $x_r$  is generated, for some  $\alpha > 0$  according to

$$x_r \equiv c + \alpha(c - x_{n+1}). \quad (23)$$

1. If  $x_r$  is a new best point, for some  $\beta > 1$  choose an *expansion point*

$$x_e \equiv c + \beta(x_r - c). \quad (24)$$

Replace  $x_{n+1}$  by whichever of  $x_r$  and  $x_e$  produced a smaller function value.

# Non-Smooth: No-Derivative Methods

2. If  $x_r$  is a new worst point, assume the polytope is too big, choosing for some  $0 < \gamma < 1$  a *contraction point*

$$x_c \equiv c + \gamma(x_r - c) \quad (25)$$

unless  $f_r \geq f_{n+1}$ , in which case use  $x_{n+1}$  instead of  $x_r$ .

3. Otherwise,  $x_r$  replaces  $x_{n+1}$ .
4. Repeat.

Surprisingly effective, though much less efficient than if we had the first derivative. A number of tweak parameters.

# Non-Smooth: No-Derivative Methods

The basic idea of *simulated annealing* is that we should always take downhill steps but occasionally also take uphill steps to avoid getting stuck in local minima.

A candidate “configuration”  $x_{k+1}$  is accepted with probability

$$\min(\exp\{- (f(x_{k+1}) - f(x_k)) / T\}, 1) \quad (26)$$

where  $T$  is the *temperature*, controlling the amount of random exploration. Notice that if  $f(x_{k+1}) < f(x_k)$ , this is 1.

There is an *annealing schedule* which decreases  $T$  by some amount  $\Delta T$  after each  $K$  samples. These are tweak parameters.

# Non-Smooth: No-Derivative Methods

To use this idea within a discrete problem, we need only specify how new solution candidates are chosen.

Here is one way to use this idea within a continuous problem.

We can modify the polytope algorithm so that we add a positive logarithmically distributed random variable, proportional to  $T$ , to the stored function value for each point in the polytope, and we subtract a similar random variable from the function value of each new candidate point. In the limit  $T \rightarrow 0$ , this is the usual polytope algorithm.

# Non-Smooth: No-Derivative Methods

In the world of discrete optimization problems, *hill-climbing* refers to the idea of looking locally around  $x_k$  and choosing from those points a  $x_{k+1}$  which moves downhill.

*Genetic algorithms* form another approach, in which a population of solution candidates is kept, and new solution candidates are chosen from them either by taking random local moves around existing candidates (“mutation”) or splicing together the vectors of two existing solution candidates (“cross-over”). There is a higher chance of choosing candidates with better function values (“fitness”).

*Genetic programming* refers to genetic algorithms which operate on solution candidates which are represented by variable-size objects such as trees, rather than fixed-length vectors.

# Non-Smooth: No-Derivative Methods

There is little to no rigorous work on these methods, either theoretically or empirically. For example it is not clear that they have been compared to the polytope method.

There are better methods for discrete (“combinatorial”) optimization.

These include WalkSAT, CHAFF, BDD-based search methods, and PIBL, but this is outside our current scope.

# Unconstrained Optimization: Sum-of-Squares

Now let's consider things more specific to statistics/learning.

# Sum-of-Squares Problems

Suppose we are minimizing squared-error loss (using our statistical notation) – then we want to find

$$\theta^* = \min_{\theta} \sum_{i=1}^N \left( \widehat{f}_{\theta}(x_i) - f_{\theta}(x_i) \right)^2. \quad (27)$$

Now using our optimization notation, this will be

$$x^* = \min_x \sum_{i=1}^N \left( \widehat{\phi}_x(t_i) - \phi_x(t_i) \right)^2. \quad (28)$$

# Sum-of-Squares Problems

We'll write that our objective function  $f$  has the form

$$f(x) = \sum_{i=1}^N \delta_i(x)^2 = \|\delta(x)\|_2^2 \quad (29)$$

where  $\delta_i(x) = \hat{\phi}_x(t_i) - \phi_x(t_i)$ .

We'll assume that  $\|\delta(x)\|_2^2$  will be “small”. If that is the case, methods specialized for this problem will be more effective than the general unconstrained optimization methods, because we can take advantage of the special structure of the gradient and Hessian matrix.

# Sum-of-Squares Problems

Denote the  $N \times D$  Jacobian matrix of  $\delta(x)$  by  $J(x)$ , and let the matrix  $H_i(x)$  denote the Hessian matrix of  $\delta_i(x)$ . Then

$$g(x) = J(x)^T \delta(x) \quad (30)$$

$$H(x) = J(x)^T J(x) + Q(x), \quad (31)$$

where  $Q(x) = \sum_{i=1}^N \delta_i(x) H_i(x)$ . Thus there is a decomposition of the Hessian into a first-order term  $J(x)^T J(x)$  and second-order term  $Q(x)$ . The first-order term dominates when  $\|\delta(x^*)\|^2$  is small, roughly speaking, comparable to the largest eigenvalue of  $J(x^*)^T J(x^*)$ .

# Sum-of-Squares Problems

Let's make a Newton method for this special case. The Newton direction is then the solution of

$$(J_k^T J_k + Q_k)p_k = -J_k^T \delta_k. \quad (32)$$

If  $\|\delta_k\|$  tends to zero as  $x_k$  approaches the solution, the matrix  $Q_k$  also tends to zero. Then the Newton direction can be *approximated* by the solution of

$$(J_k^T J_k)p_k = -J_k^T \delta_k, \quad (33)$$

or  $\min_p \|J_k p + \delta_k\|^2$ . Choosing this direction is called a *Gauss-Newton method*.

If  $J_k$  is full-rank and  $f(x^*) = 0$  this has quadratic convergence even though it only uses first derivatives.

# Sum-of-Squares Problems

One generalization is the *Levenberg-Marquardt method*. Here the search direction is the solution to

$$(J_k^T J_k + \lambda_k I) p_k = -J_k^T \delta_k, \quad (34)$$

where  $\lambda_k$  is a non-negative scalar. If  $\lambda_k$  is zero, we have the Gauss-Newton method. As  $\lambda_k \rightarrow \infty$ ,  $p_k$  becomes parallel to the steepest-descent direction.

$\lambda_k$  is chosen in a way which is intended to use steepest-descent when far from the minimum and Gauss-Newton near the minimum.

If Gauss-Newton-type assumptions are not applicable, we can also approximate the Hessian with a Quasi-Newton approximation to  $Q_k$ .

# Main Things You Should Know

- What Newton's method is
- What steepest descent is
- What the polytope method is
- What Levenberg-Marquardt is
- What quasi-Newton is