Optimization using Gradient Descent

Many of the statistical estimation problems that we have talked about ultimately come down to solving an optimization program,

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}), \tag{1}$$

for some $f : \mathbb{R}^N \to \mathbb{R}$. One approach to solving (1) is by using **gradient descent**. The essential algorithm, in the unconstrained case, is straightforward:
From initial point $\boldsymbol{x}_0$, we iterate over $k = 0, 1, \ldots$ with

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).$$

The α_k above is a stepsize that changes (in general) at every iteration. When f is *convex*, there is a lot we can say about the convergence of gradient descent to a global optima. When f is not convex, there is much less we can say; most results in this domain characterize the convergence to a local minima that depends on the starting point $\boldsymbol{x}_0$.

Choosing the stepsize

Many algorithms for optimization rely on choosing α_k adaptively. We are at starting point $\boldsymbol{x}_k$ and have decided to move in direction $\boldsymbol{d}_k$, and now we want to decide how far to move; we want to choose α_k for the update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$. The first thing we might think of is an **exact line search**, taking

$$\alpha_k = \underset{t>0}{\arg\min}\, f(\boldsymbol{x}_k + t\boldsymbol{d}_k).$$

And in fact, this is what we did previously in the least-squares case, as we had a closed form to compute the minimizer exactly. But in general, computing the solution to this 1D optimization program is not

straightforward. Moreover, finding the truly "optimal" step might not even be that beneficial, as $\boldsymbol{d}_k$ does not point exactly towards the solutions anyway.

More commonly, a step size is chosen through a **backtracking line search**. For a fixed initial step size t_0 and backtracking parameter $0 < \beta < 1$, do the following:

Initialize: $t = t_0$
while $f(\boldsymbol{x}_k + t\boldsymbol{d}_k)$ is "not good enough" **do**
    $t = \beta t$
end while
$\alpha_k = t$

Of course, the key thing here is defining "not good enough". You might be tempted to try to implement a condition such as $f(\boldsymbol{x}_k+t\boldsymbol{d}_k)$ being as least 10% smaller than $f(\boldsymbol{x}_k)$, but there is nothing that says that this has to be true for any t, especially when you are close to the solution. When $f(\cdot)$ is convex and the problem is unconstrained, there is a systematic criteria you can use that also gives great results, both in theory and in practice.

For a convex function, it will always be true that

$$f(\boldsymbol{x}_k + t\boldsymbol{d}_k) \geq f(\boldsymbol{x}_k) + t\langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)\rangle, \quad \text{for all } t \geq 0.$$

A good criteria for backtracking line search is to find a t such that

$$f(\boldsymbol{x}_k + t\boldsymbol{d}_k) < f(\boldsymbol{x}_k) + t\gamma\langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)\rangle,$$

for some fixed $\gamma \in (0, 0.5)$. So in the algorithm above, make the substitution

$f(\boldsymbol{x}_k+t\boldsymbol{d}_k)$ is "not good enough" $\leftarrow f(\boldsymbol{x}_k+t\boldsymbol{d}_k) > f(\boldsymbol{x}_k)+t\gamma\langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)\rangle$.

Here is a figure[1] (from [BV04, Chapter 9]):

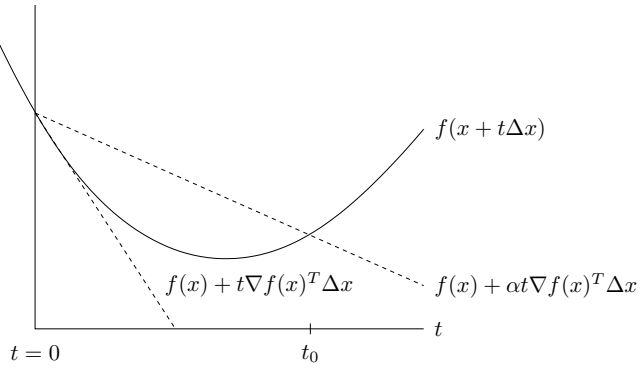

Figure 9.1 *Backtracking line search.* The curve shows f, restricted to the line over which we search. The lower dashed line shows the linear extrapolation of f, and the upper dashed line has a slope a factor of α smaller. The backtracking condition is that f lies below the upper dashed line, *i.e.*, $0 \leq t \leq t_0$.

The best values of β and γ vary by application; a good starting point is to take $\beta = 0.5$ and $\gamma = 0.05$.

Gradient descent with constraints

If there are constraints which we can specify through a restriction of the optimization variable to a set $\mathcal{C} \subset \mathbb{R}^N$,

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{x} \in \mathcal{C}, \tag{2}$$

then there is an analogous algorithm called **projected gradient descent**. For our discussion here, we will assume that the set $\mathcal{C}$ is convex as is the functional f.

[1] They are using 'α' in the figure label instead of the 'γ' we are using in these notes.

The projected gradient descent algorithm makes use of the the "closest point" operator $\mathrm{P}_{\mathcal{C}} : \mathbb{R}^N \to \mathbb{R}^N$,

$$\mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) = \arg\min_{\boldsymbol{z} \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{z}\|_2^2.$$

As the name suggests, this takes an arbitrary vector in $\mathbb{R}^N$ and returns the point in $\mathcal{C}$ that is closest. If $\mathcal{C}$ is closed and convex, then the closest point exists and is unique. For some sets $\mathcal{C}$ it is easy to compute $\mathrm{P}_{\mathcal{C}}(\boldsymbol{x})$, for others it is extremely hard.

Examples:

- $\mathcal{C}$ is a K-dimensional subspace spanned by basis vectors that are the columns of the $N \times K$ matrix $\boldsymbol{A}$. Then

  $$\mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}.$$

  If we have an orthobasis for the subspace concatenated as the columns of $\boldsymbol{U}$, this simplifies to

  $$\mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}.$$

  For example, if $N = 4$, and

  $$\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^4 \ : \ x[2] = 0, \ x[3] = 0\},$$

  then

  $$\mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) = \begin{bmatrix} x[1] \\ 0 \\ 0 \\ x[4] \end{bmatrix}.$$

- $\mathcal{C}$ is the unit ball for the ℓ_2 norm:

  $$\mathcal{C} = \{\boldsymbol{x} \ : \ \|\boldsymbol{x}\|_2 \leq 1\}.$$

Then

$$\mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}, & \|\boldsymbol{x}\|_2 \leq 1, \\ \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}, & \|\boldsymbol{x}\|_2 > 1. \end{cases}$$

- $\mathcal{C}$ is the positive orthant:

$$\mathcal{C} = \{\boldsymbol{x} \ : \ x[n] \geq 0, \ n = 1, \ldots, N\}.$$

Then

$$\hat{\boldsymbol{x}} = \mathrm{P}_{\mathcal{C}}(\boldsymbol{x}) \quad \text{obeys} \quad \hat{x}[n] = \max(x[n], 0).$$

- $\mathcal{C}$ is the set of $N \times N$ symmetric positive semi-definite matrices:

$$\mathcal{C} = \{\boldsymbol{X} \ : \ \boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}, \ \ \lambda_n \geq 0\}.$$

If $\boldsymbol{X}$ is an arbitrary symmetric matrix with eigenvalue decomposition

$$\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}} = \sum_{n=1}^{N} \lambda_n \boldsymbol{v}_n \boldsymbol{v}_n^{\mathrm{T}},$$

then[2]

$$\mathrm{P}_{\mathcal{C}}(\boldsymbol{X}) = \sum_{n=1}^{N} \max(\lambda_n, 0)\, \boldsymbol{v}_n \boldsymbol{v}_n^{\mathrm{T}}.$$

If $\boldsymbol{X}$ is an arbitrary (not necessarily symmetric) matrix, then we form

$$\boldsymbol{X}' = \frac{1}{2}(\boldsymbol{X} + \boldsymbol{X}^{\mathrm{T}})$$

with eigenvalue decomposition $\boldsymbol{X}' = \boldsymbol{V}'\boldsymbol{\Lambda}'\boldsymbol{V}'^{\mathrm{T}}$ and take

$$\mathrm{P}_{\mathcal{C}}(\boldsymbol{X}) = \sum_{n=1}^{N} \max(\lambda'_n, 0)\, \boldsymbol{v}'_n \boldsymbol{v}'^{\mathrm{T}}_n.$$

[2] That this is indeed the closest point isn't at all obvious, but we will forgo the proof here.

If we do indeed have a tractable projection operator for $\mathcal{C}$, then we can solve (2) using the iterations:

$$\begin{aligned} \boldsymbol{z}_k &= \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \mathrm{P}_{\mathcal{C}}(\boldsymbol{z}_k). \end{aligned}$$

The convergence properties of this algorithm, with the appropriate choice of step size α_k, are very similar to those for unconstrained gradient descent.

Linear constraints. When $\mathcal{C}$ is a subspace, then (as we saw above) the projection operator is itself linear. Thus in this case

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \mathrm{P}_{\mathcal{C}}\left(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)\right) \\ &= \mathrm{P}_{\mathcal{C}}(\boldsymbol{x}_k) - \alpha_k \, \mathrm{P}_{\mathcal{C}}(\nabla f(\boldsymbol{x}_k)) \\ &= \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k, \quad \boldsymbol{g}_k = \mathrm{P}_{\mathcal{C}}(\nabla f(\boldsymbol{x}_k)), \end{aligned}$$

where the last equality follows from the fact that $\boldsymbol{x}_k$ is, by construction, already in $\mathcal{C}$. For optimization with linear constraints, then, we can compute the gradient, project it, then choose a step size using exact or backtracking line search.

Interior point iterations

Another approach to handling constraints it to keep iterates in the "interior" of the set $\mathcal{C}$. We say $\boldsymbol{x}$ is an **interior point** for $\mathcal{C}$ if there exists an $\epsilon > 0$ such that

$$\boldsymbol{x} + \epsilon \boldsymbol{h} \in \mathcal{C}, \quad \text{for all } \; \boldsymbol{h} \in \mathbb{R}^D \text{ with } \|\boldsymbol{h}\|_2 \leq 1.$$

This basically means that the set $\mathcal{C}$ has non-zero volume. Not all sets $\mathcal{C}$ have an interior point — for example a subspace of dimension $K < N$ does not have any interior points.

Examples:

- If $\mathcal{C}$ is the positive orthant, then all $\boldsymbol{x}$ with $x[n] > 0$, $n = 1, \ldots, N$ are interior points.
- If $\mathcal{C}$ is the set of $N \times N$ symmetric positive semi-definite matrices, than all symmetric positive definite matrices (symmetric matrices that have eigenvalues that are strictly positive) are interior points.

If we start at an interior point $\boldsymbol{x}_0$, we can solve (2) by using gradient descent with α_k chosen using backtracking line search with the additional constraint that

$$\boldsymbol{x}_k + t\boldsymbol{d}_k \in \text{Interior}(\mathcal{C}).$$

This type of algorithm can be very effective for small problems, but tends to converge very slowly for large problems when the solution is on the boundary of $\mathcal{C}$ (where it tends to be, if the constraints are changing the problem in any way). Indeed, there is an entire class of interior point algorithms that are far more graceful, using smooth "barrier functions" to make the problem better posed.

A particular place where interior point iterations are interesting, and one that is relevant for Homework 10, is when

$$\mathcal{C} = \mathcal{X} \cap \mathcal{S},$$

where $\mathcal{S}$ is a subspace (e.g. constraining certain optimization variables to be zero), and $\mathcal{X}$ is a convex set with a non-empty interior[3] (e.g. the set of symmetric positive semi-definite matrices). Now, from

[3]We really only need that $\mathcal{X}$ has a non-empty interior relative to $\mathcal{S}$.

an initial point $\boldsymbol{x}_0 \in \mathcal{C}$, we can proceed via

$$\begin{aligned} \boldsymbol{g}_k &= \mathrm{P}_{\mathcal{S}}(\nabla f(\boldsymbol{x}_k)), \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k, \end{aligned}$$

where α_k is chosen small enough so that $\boldsymbol{x}_{k+1} \in \mathrm{Interior}(\mathcal{X})$.

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.