Optimization Algorithms

Matthieu R. Bloch

Assume that we wish to solve the problem $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ where $f:\mathbb{R}^d\to\mathbb{R}$. Very often one cannot obtain a closed form expression for the solution and one resorts to numerical algorithms to obtain the solution. We could spend an entire semester studying these algorithms in depth (and why they work), and we will only briefly review here important concepts.

1 Gradient descent

The idea of gradient descent is to find the minimum of f iteratively by following the opposite directions of the gradient ∇f. Intuitvely, gradient descent consists in "rolling down the hill" to find the minimum. A typical gradient descent algorithm would run as follows.

- Start with a guess of the solution $\mathbf{x}^{(0)}$.
- For $j\geqslant 0$, update the estimate as $\mathbf{x}_{j+1}=\mathbf{x}_j-\eta\nabla f(\mathbf{x}_j)$, where $\eta>0$ is called the *stepsize*.

Without further assumptions, there is no guarantee of convergence. In addition, the choice of step size η really matters: too small and convergence takes forever, too big and the algorithm might never converge. Very often in machine learning, the function f to optimize is a loss function $\ell(\theta)$ that takes the form

$$\ell(\theta)\triangleq\sum_{i=1}^{N}\ell_i(\theta), \tag{1}$$

where $\ell_i(\theta)$ is a function of θ and the data point $(\mathbf{x}_i,y_i)$ but not the other data points. When the number of data points N is very large, or when the data points cannot all be accessed at the same time, a typical approach is to not compute the exact gradient $\nabla\ell$ to perform updates but to only evaluate $\nabla\ell_i$. In its simplest form, *stochastic* gradient descent consists of the following rule.

- Start with a guess of the solution $\theta^{(0)}$.
- For $j\geqslant 0$, update the estimate as $\theta^{(j+1)}=\theta^{(j)}-\eta\nabla\ell_j(\theta^{(j)})$, where $\eta>0$ is the stepsize.[1]

Note that the update only depends on the loss function evaluated at a *single* point $(\mathbf{x}_j,y_j)$.

2 Newton's method

One drawback of the basic gradient descent sketched earlier is the presence of a parameter η, which has to be set *a priori*. There exists many methods to choose η adaptively, and Newton's method is one of many such methods. Specifically, consider a quadratic approximation $\tilde{f}$ of the function f at a point $\mathbf{x}$:

$$\tilde{f}(\mathbf{x}')\triangleq f(\mathbf{x})+\nabla f(\mathbf{x})(\mathbf{x}'-\mathbf{x})+\frac{1}{2}(\mathbf{x}'-\mathbf{x})^{\intercal}\nabla^2 f(\mathbf{x})(\mathbf{x}'-\mathbf{x}). \tag{2}$$

[1] The data index may be chosen to be different from the iteration step index.

The matrix $\nabla^2 f(\mathbf{x}) \in \mathbb{R}^{d\times d}$ is called the *Hessian* of f and is defined as

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \tag{3}$$

Note that the gradient of $\tilde{f}$ is $\nabla \tilde{f}(\mathbf{x}') = \nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x})\mathbf{x}' - \nabla^2 f(\mathbf{x})\mathbf{x}$.

Let us now assume that we decide to update our next point in the gradient descent update by choosing $\mathbf{x}_{j+1}$ as the minimum of the quadratic approximation of f at $\mathbf{x}_j$. Because $\tilde{f}$ is quadratic, we can find the minimum by solving $\nabla \tilde{f}(\mathbf{x}_{j+1}) = 0$:

$$\nabla \tilde{f}(\mathbf{x}_{j+1}) = 0 \Leftrightarrow \nabla f(\mathbf{x}_j) + \nabla^2 f(\mathbf{x}_j)\mathbf{x}_{j+1} - \nabla^2 f(\mathbf{x}_j)\mathbf{x}_j = 0 \tag{4}$$

$$\Leftrightarrow \mathbf{x}_{j+1} = \mathbf{x}_j - \left[\nabla^2 f(\mathbf{x}_j)\right]^{-1} \nabla f(\mathbf{x}_j). \tag{5}$$

This looks like the gradient descent update equation except that we have chosen the stepsize to be a matrix $\left[\nabla^2 f(\mathbf{x}_j)\right]^{-1}$; this adjusts how much we move as a function of the local curvature.

Newton's methods has much faster convergence than gradient descent, but requires the calculation of the inverse of the Hessian. This is feasible when the dimension d is small but impractical when d is large. In many machine learning problems, researchers therefore focus on gradient descent techniques that attempt to adapt the step size without having to compute a full Hessian.