07-280 Optimization

Nihar B. Shah

Consider ERM:

$$\arg\min_{h \in \mathcal{H}} \sum_{i=1}^{N} \ell(h(x^{(i)}), y^{(i)})$$

This is an optimization problem: finding min / max / arg min / arg max. How to do this in a computationally efficient manner? We will start with a simple problem, then design an algorithm that is used widely in very general settings.

Recall linear regression with $\mathcal{X} = \mathbb{R}$. Let us set $b = 0$. Suppose $N = 1$. Then ERM:

$$\arg\min_{\omega} \underbrace{\left(y^{(1)} - x^{(1)} \cdot \omega\right)^2}_{\mathcal{J}(\omega),\, \mathcal{J}:\mathbb{R}\to\mathbb{R}}$$

What is the shape of $\mathcal{J}$? $\mathcal{J}(\omega)$ is a **parabola**. Even for general N, it is a parabola.

Observe that the derivative $\frac{d\mathcal{J}(\omega)}{d\omega}$ at points ω to the right of the minimum are all positive. And those to the left of the minimum are all negative.

Idea for an algorithm: Start with any point ω.

- If derivative of $\mathcal{J}$ is positive at that point, then move left.
- If derivative of $\mathcal{J}$ is negative at that point, then move right.
- Repeat until some stopping criterion is met.

How much to move? Observe that the magnitude of the derivative is large whenever the point is farther from the arg min.

Idea: Move proportional to magnitude of derivative.

Gradient Descent Algorithm

Algo:

- Initialize to some $\omega^{(0)} \in \mathbb{R}$
- For $t = 1, 2, \ldots$ (until some stopping criterion is met)
  - Set $\omega^{(t)} = \omega^{(t-1)} - \alpha \left.\frac{d\mathcal{J}(\omega)}{d\omega}\right|_{\omega=\omega^{(t-1)}}$
- Return $\omega^{(t)}$

This is the **"Gradient descent" (GD)** algorithm.

What are some stopping criteria? When performance isn't improving much or derivative isn't changing much or the iterate isn't moving much or you run out of compute.

α is called "learning rate" or 'step size'. A simple choice is just some constant (e.g., 0.01).

Let's do an example. Let $\mathcal{J}(\omega) = \omega^2$. Initialize $\omega^0 = 1$. Consider 3 iterations. What is $\omega^{(3)}$ when:

(i) $\alpha = 0.25$

(ii) $\alpha = 0.5$

(iii) $\alpha = 1$

(iv) $\alpha = 2$?

Observe that the learning rate matters regarding the performance. Another simple choice is to vary it with the iteration number t, e.g., as $\alpha_t = \frac{1}{\sqrt{t}}$. There are also more advanced methods for adaptive tuning of learning rate which we won't go into in this course.

The discussion so far was for a single sample for which we saw that $(y^{(i)} - x^{(i)}\omega)^2$ is a parabola. But $\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - x^{(i)} \cdot \omega)^2$ is also a parabola, so the same ideas apply here. This algorithm and its variants are, in fact, used in settings much more general than parabolas. We will come back to that later.

Extension to Vector Case

Now let us extend our ideas to a vector case: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$. Consider some function $\mathcal{J} : \mathbb{R}^d \to \mathbb{R}$. We want to find $\arg\min_{\theta \in \mathbb{R}^d} \mathcal{J}(\theta)$. For this we will need to establish some background.

Multi-variate Calculus

If $\mathcal{J} : \mathbb{R} \to \mathbb{R}$ (scalar case), $\frac{\partial \mathcal{J}(\theta)}{\partial \theta} \in \mathbb{R}$. Now, when $\mathcal{J} : \mathbb{R}^d \to \mathbb{R}$,

$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial \mathcal{J}(\theta)}{\partial \theta_1} \\ \frac{\partial \mathcal{J}(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathcal{J}(\theta)}{\partial \theta_d} \end{bmatrix} \in \mathbb{R}^d.$$

This is called the **gradient** of $\mathcal{J}$, denoted as $\nabla_\theta \mathcal{J}(\theta)$ or simply $\nabla \mathcal{J}$.

For instance:

(i) If $\mathcal{J}(\theta) = \theta^T v$ for some $v \in \mathbb{R}^d$ then $\nabla_\theta \mathcal{J}(\theta) = v$.

(ii) If $\mathcal{J}(\theta) = \theta^T A \theta$ for some $A \in \mathbb{R}^{d \times d}$ then $\nabla_\theta \mathcal{J}(\theta) = (A + A^T)\theta$.

Note that more generally, as you will encounter later in this course, you may have to compute $\frac{\partial \mathcal{J}(\theta)}{\partial \theta}$ where θ could be scalar, vector or matrix and the output of J could also be scalar, vector or matrix. Just remember that $\frac{\partial \mathcal{J}(\theta)}{\partial \theta}$ is simply a collection of scalar by scalar partial derivatives: the partial derivative of each coordinate of $\mathcal{J}(\theta)$ with respect to each coordinate of θ. These scalar partial derivatives are then arranged appropriately in some shape, like the vector in $\nabla_\theta \mathcal{J}(\theta)$. We will use "denominator layout" of arrangement (more popular in ML, as compared to "numerator layout.")

Back to Optimization

Now consider $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$. Recall ERM for linear regression:

$$\arg\min_\theta \|y - X\theta\|_2^2 == \arg\min_\theta (y - X\theta)^T (y - X\theta) = \arg\min_\theta \underbrace{y^T y - 2\theta^T X^T y + \theta^T X^T X\theta}_{\mathcal{J}(\theta)}$$

$\mathcal{J}$ is a parabola and we want to find θ such that $\nabla_\theta \mathcal{J}(\theta) = 0$.

Can use gradient descent. However linear regression is a special setting where something nice happens:

$$\nabla_\theta \mathcal{J}(\theta) = -2X^T y + 2X^T X\theta$$

Setting this to 0 gives a closed form expression for the optimal $\theta = (X^T X)^{-1} X^T y$.

In real life, most problems do not admit such a closed form solution. Gradient descent is a highly effective tool in such situations, even for functions that are not parabolas and may have crazy shapes (like in neural networks). Here is the gradient descent algorithm to attempt to find the argmin for a general function $\mathcal{J} : \mathbb{R}^d \to \mathbb{R}$:

- Initialize $\theta^{(0)} \in \mathbb{R}^d$
- For $t = 1, 2, \ldots$ (until some stopping criterion is met)
  - Set $\theta^{(t)} = \theta^{(t-1)} - \alpha \, \nabla_\theta \mathcal{J}(\theta)|_{\theta=\theta^{(t-1)}}$
- Return $\theta^{(t)}$

Note that gradient descent may not necessarily find the optimal solution unless the function takes some special form. It may get stuck in a "local minimum."

- Local Minimum: A value of $\theta \in \mathbb{R}^d$ which is the lowest in its neighborhood (but there could be points θ' farther away which are lower).
- Global Minimum: A value of θ that minimizes the function: $\mathcal{J}(\theta) \leq \mathcal{J}(\theta')$ for every $\theta' \in \mathbb{R}^d$.

Despite the fact that gradient descent may not always reach a global minimum (and may get stuck at a local minimum that is not a global minimum), in practice gradient descent and its variants are still very popular since they find a good solution in a computationally efficient manner.

At this point, let us come back to ERM:

$$\arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \ell\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

where $h_\theta(x) = \theta^T x$ under linear regression, but $h_\theta(x)$ could alternatively capture other types of hypothesis such as neural networks.

In the context of ERM, for reasons we will see below, gradient descent is also referred to as "batch" gradient descent – it process all N samples together as a batch. This causes a problem.

Problem: Suppose N is large. Then gradient descent has to compute $\nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$ for every $i \in \{1, \ldots, N\}$ in order to make any update to the iterate $\theta^{(t-1)}$. This makes it slow.

Idea: Take gradients of just one sample at a time for each update.

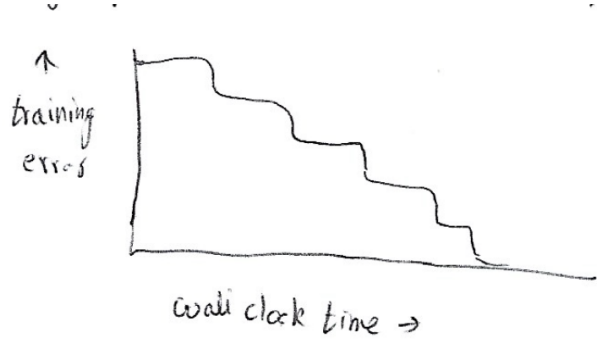

Figure 1: Training error of gradient descent.

Stochastic Gradient Descent (SGD)

- Initialize $\theta^{(0)} \in \mathbb{R}^d$
- For $t = 1, 2, \ldots$ (until some stopping criterion is met)
  - Choose $j \in$ Uniform-Random$\{1, 2, \ldots, N\}$
  - $\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla_\theta \ell(h_\theta(x^{(j)}), y^{(j)})$
- Return $\theta^{(t)}$

Very noisy version of full gradient, but fast to compute.

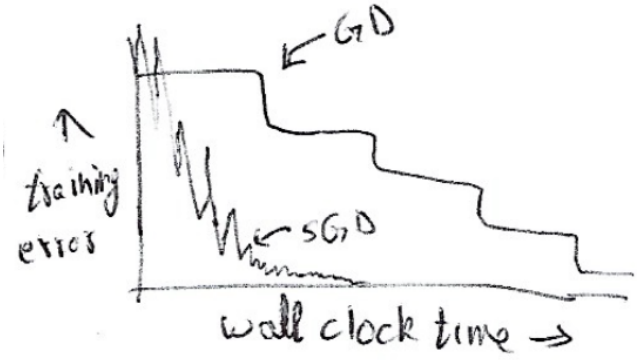

Figure 2: Training error of (batch) GD and SGD.

Intermediate version: Choose K $(1 < K < N)$ samples at random per iteration. Called "Mini-batch" GD.