Local, Unconstrained Function Optimization

COMPSCI 527 — Computer Vision

Outline

1. Motivation and Scope
2. First Order Methods
3. Gradient Descent
4. Step Size Selection Methods
5. Termination
6. Is Gradient Descent a Good Strategy?
7. Stochastic Gradient Descent

Motivation and Scope

- Most estimation problems are solved by optimization
- Machine learning:
  - Parametric predictor: $h(\mathbf{x} \,;\, \mathbf{v}) \;:\; \mathbb{R}^d \times \mathbb{R}^m \to Y$
  - Training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ and *loss* = $\ell(y_n, y)$
  - Risk: $L_T(\mathbf{v}) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, h(\mathbf{x}_n \,;\, \mathbf{v})) \;:\; \mathbb{R}^m \to \mathbb{R}$
  - Training: $\hat{\mathbf{v}} = \arg\min_{\mathbf{v}\in\mathbb{R}^m} L_T(\mathbf{v})$
- 3D Reconstruction:
  - Computer Graphics: $I = \pi(C, S)$ where I are (multiple) images, C are the camera positions and orientations, S is scene shape
  - Computer Vision: Given I, find $\hat{C}, \hat{S} = \arg\min_{C,S} \|I - \pi(C, S)\|$
- In general, "solving" the system of equations $E(\mathbf{z}) = 0$ can be viewed as

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} \|E(\mathbf{z})\|$$

Only *Local* Minimization

$\hat{\mathbf{z}} = \arg\min_{\mathbf{z} \in ?} f(\mathbf{z})$

- All we know about f is a "black box" (think Python function)
- For many problems, f has many local minima
- Start somewhere ($\mathbf{z}_0$), and take steps "down"

  $f(\mathbf{z}_{k+1}) < f(\mathbf{z}_k)$
- When we get stuck at a local minimum, we declare success
- We would like global minima, but all we get is local ones
- For some problems, f has a unique minimum...
- ... or at least a single connected set of minima

Gradient

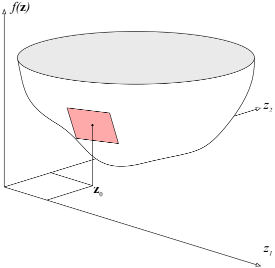
$$\nabla f(\mathbf{z}) = \frac{\partial f}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f}{\partial z_1} \\ \vdots \\ \frac{\partial f}{\partial z_m} \end{bmatrix}$$

- We worked with gradients for the case $\mathbf{z} \in \mathbb{R}^2$ (images)
- Now $\mathbf{z} \in \mathbb{R}^m$ with m possibly very large
- If $\nabla f(\mathbf{z})$ exists everywhere, the condition $\quad \nabla f(\mathbf{z}) = \mathbf{0}$ is necessary and sufficient for a stationary point (max, min, or saddle)
- Warning: only *necessary* for a minimum!
- Reduces to first derivative when $f : \mathbb{R} \rightarrow \mathbb{R}$

First Order Taylor Expansion

$$f(\mathbf{z}) \approx g_1(\mathbf{z}) = f(\mathbf{z}_0) + [\nabla f(\mathbf{z}_0)]^T(\mathbf{z} - \mathbf{z}_0)$$

approximates $f(\mathbf{z})$ near $\mathbf{z}_0$ with a (hyper)plane through $\mathbf{z}_0$



$\nabla f(\mathbf{z}_0)$ points to direction of steepest *increase* of f at $\mathbf{z}_0$

- If we want to find $\mathbf{z}_1$ where $f(\mathbf{z}_1) < f(\mathbf{z}_0)$, going along $-\nabla f(\mathbf{z}_0)$ seems promising
- This is the general idea of *gradient descent*

A Template

- Gradient descent methods fit the following template:

$k = 0$
while $\mathbf{z}_k$ is not a minimum
  compute the gradient $\mathbf{g}_k = \nabla f(\mathbf{z}_k)$
  compute a "learning rate" $\alpha_k > 0$
  $\mathbf{z}_{k+1} = \mathbf{z}_k - \alpha_k \mathbf{g}_k$
  $k = k + 1$
end

Design Decisions

$k = 0$
while $\mathbf{z}_k$ is not a minimum
    compute the gradient $\mathbf{g}_k$
    compute a *learning rate* $\alpha_k > 0$
    $\mathbf{z}_{k+1} = \mathbf{z}_k - \alpha_k \mathbf{g}_k$
    $k = k + 1$
end

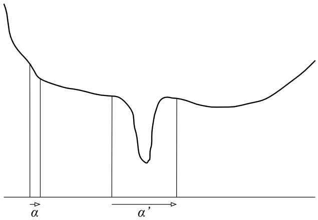
- In what direction to proceed ($-\mathbf{g}_k$)
- How long a step to take in that direction ($\alpha_k \|\mathbf{g}_k\|$)
- When to stop ("while $\mathbf{z}_k$ is not a minimum")
- Different decisions lead to different methods

Gradient Descent

- In what direction to proceed: $-\mathbf{g}_k = -\nabla f(\mathbf{z}_k)$
- "Gradient descent"
- Problem reduces to one dimension:
  $h(\alpha) = f(\mathbf{z}_k - \alpha \mathbf{g}_k)$
- $\alpha = 0 \Leftrightarrow \mathbf{z} = \mathbf{z}_k$
- Find $\alpha = \alpha_k > 0$ such that
  $f(\mathbf{z}_k - \alpha_k \mathbf{g}_k) < f(\mathbf{z}_k)$
- How to find α_k?

Step Size

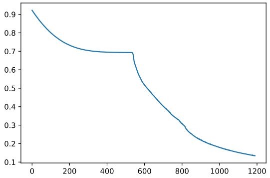
- Simplest idea: $\alpha_k = \alpha$ (fixed learning rate)
  - Small α leads to slow progress
  - Large α can miss minima



- Scheduling α:
  - Start with α relatively large (say $\alpha = 10^{-3}$)
  - Decrease α over time
  - Determine decrease rate of α by trial and error

Momentum

- Sometimes $\mathbf{z}_k$ meanders around in shallow valleys



$f(\mathbf{z}_k)$ versus k

- α is too small, direction is still promising
- Add *momentum*

$$\mathbf{v}_0 = \mathbf{0}$$

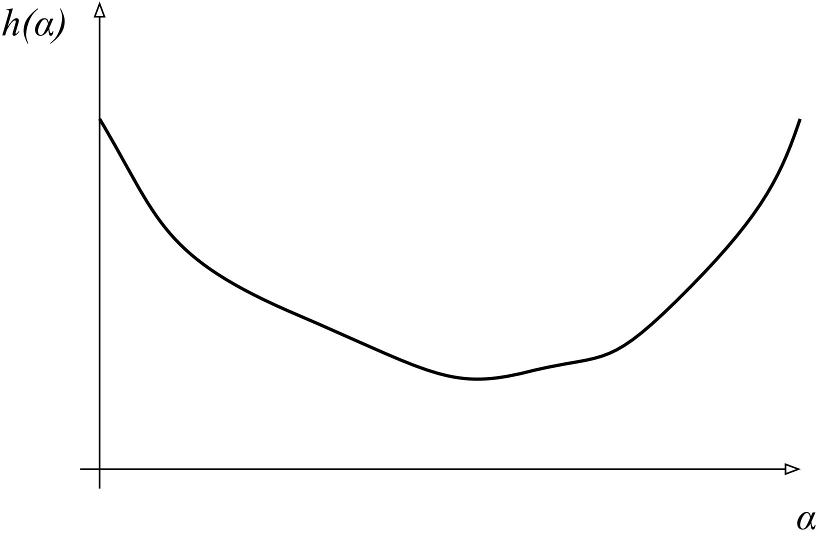
$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k - \alpha \nabla f(\mathbf{z}_k) \qquad (0 \leq \mu_k < 1)$$

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{v}_{k+1}$$

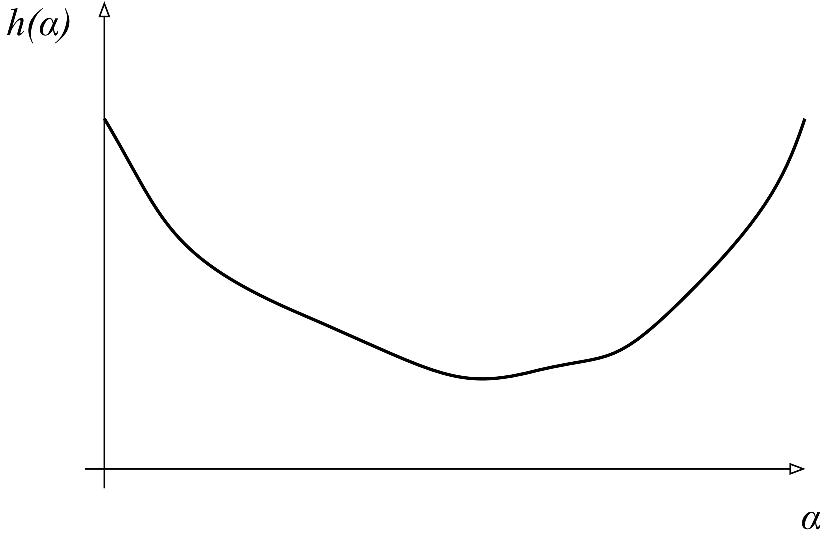
Line Search

- Find a local minimum in the search direction $\mathbf{p}_k = -\mathbf{g}_k$
  $h(\alpha) = f(\mathbf{z}_k + \alpha\mathbf{p}_k)$, a one-dimensional problem
- *Bracketing triple*:
- $a < b < c, \quad h(a) \geq h(b), \quad h(b) \leq h(c)$
- Contains a (local) minimum!
- Split the bigger of $[a, b]$ and $[b, c]$ in half with a point u
- Find a new, narrower bracketing triple involving u and two out of a, b, c
- Stop when the bracket is narrow enough (say, 10^{-6})
- Pinned down a minimum to within 10^{-6}

Phase 1: Find a Bracketing Triple



Phase 2: Shrink the Bracketing Triple



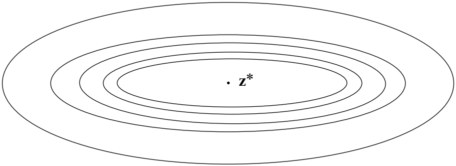
if $b - a > c - b$
    $u = (a + b)/2$
    if $h(u) > h(b)$
        $(a, b, c) = (u, b, c)$
    otherwise
        $(a, b, c) = (a, u, b)$
    end
otherwise
    $u = (b + c)/2$
    if $h(u) > h(b)$
        $(a, b, c) = (a, b, u)$
    otherwise
        $(a, b, c) = (b, u, c)$
    end
end

Termination

- Are we still making “significant progress”?
- Check $f(\mathbf{z}_{k-1}) - f(\mathbf{z}_k)$? (We want this to be strictly positive)
- Check $\|\mathbf{z}_{k-1} - \mathbf{z}_k\|$? (We want this to be large enough)
- Second is more stringent close the the minimum because $\nabla f(\mathbf{z}) \approx \mathbf{0}$
- Stop when $\|\mathbf{z}_{k-1} - \mathbf{z}_k\| < \delta$

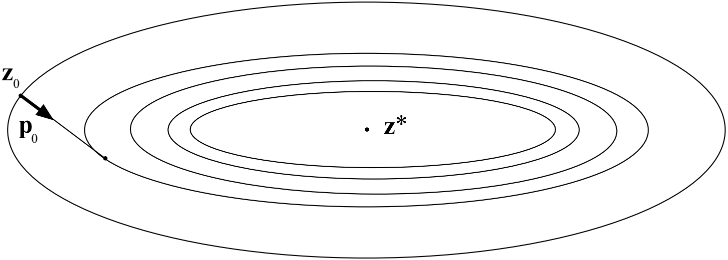
Is Gradient Descent a Good Strategy?

- "We are going in the direction of fastest descent"
- "We choose an optimal step size by line search"
- "Must be good, no?" *Not so fast!*
- An example for which we know the answer:
  $f(\mathbf{z}) = c + \mathbf{a}^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T Q\mathbf{z}$
  $Q \succcurlyeq 0$ (convex paraboloid)
- All smooth functions look like this close enough to $\mathbf{z}^*$



isocontours

Skating to a Minimum



- Many 90-degree turns slow down convergence
- There are methods that take fewer iterations, but each iteration takes more time and space
- We will stick to gradient descent
- See appendices in the notes for more efficient methods for problems in low-dimensional spaces

Stochastic Gradient Descent

- A special case of gradient descent, SGD works for *averages* of many terms (N very large):

$$f(\mathbf{z}) = \frac{1}{N} \sum_{n=1}^{N} \phi_n(\mathbf{z})$$

- Computing $\nabla f(\mathbf{z}_k)$ is too expensive
- Partition $B = \{1, \ldots, N\}$ into J random *mini-batches* B_j each of about equal size

$$f(\mathbf{z}) \approx f_j(\mathbf{z}) = \frac{1}{|B_j|} \sum_{n \in B_j} \phi_n(\mathbf{z}) \quad \Rightarrow \quad \nabla f(\mathbf{z}) \approx \nabla f_j(\mathbf{z}) .$$

- Mini-batch gradients are correct *on average*

SGD and Mini-Batch Size

- SGD iteration: $\mathbf{z}_{k+1} = \mathbf{z}_k - \alpha_k \nabla f_j(\mathbf{z}_k)$
- Mini-batch gradients are correct *on average*
- One cycle through all the mini-batches is an *epoch*
- Repeatedly cycle through all the data
  (Scramble data before each epoch)
- *Asymptotic* convergence can be proven with suitable step-size schedule
- Small batches $\Rightarrow$ low storage but high gradient variance
- Make batches as big as will fit in memory for minimal variance
- In deep learning, memory is GPU memory