Lecture 2: Function Minimization

STAT 545: Intro. to Computational Statistics

Vinayak Rao

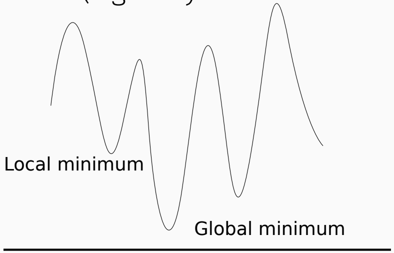
Purdue University

August 19, 2019

Global and local minima

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.
(maximization is just minimizing $-f$).

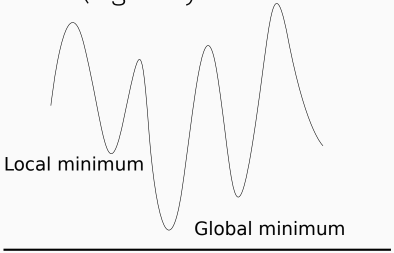
No global information (e.g. only function values, derivatives).



Global and local minima

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.
(maximization is just minimizing $-f$).

No global information (e.g. only function values, derivatives).



Finding global minima is hard! Usually settle for local minima.

Even finding local minima is not easy. Usually need iterative algorithms. (Exceptions?)

Gradient descent (iterative method)

Consider 1-d case. Let x_{old} be our current value.

Update x_{new} as $$x_{new} = x_{old} - \eta \, \frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_{old}}$$

The steeper the slope, the bigger the move.

Gradient descent (iterative method)

Consider 1-d case. Let x_{old} be our current value.

Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move.

η: ‘step-size’ or ‘learning rate’.

Gradient descent (iterative method)

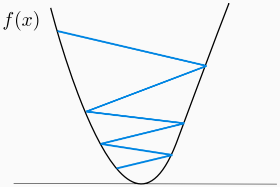
Consider 1-d case. Let x_{old} be our current value.

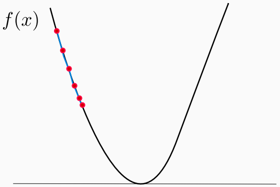
Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move.

η: ‘step-size’ or ‘learning rate’.

Choosing η requires care (not too large or too small):





Gradient descent (iterative method)

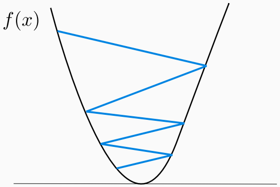
Consider 1-d case. Let x_{old} be our current value.

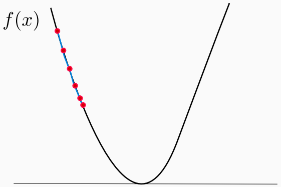
Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move.

η: ‘step-size’ or ‘learning rate’.

Choosing η requires care (not too large or too small):





Better methods adapt step-size according to the curvature of f.

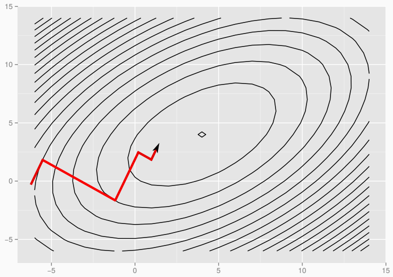
Gradient descent in higher-dimensions

Also applies to higher dimensions: $x_{new} = x_{old} - \eta \left.\nabla f\right|_{x_{old}}$

Again, need care choosing η

Alternately, at each step, set η by minimizing along ∇f

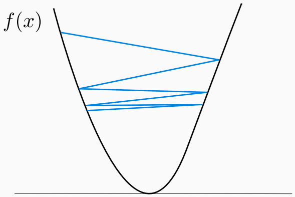
- Note: even the optimal step-size η can be inefficient:

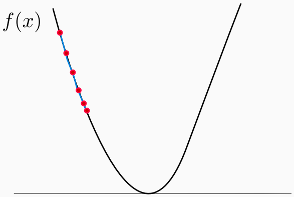


Rather than finding best step-size each step, save computation and find a decent solution

Wolfe conditions to decide step-size

Bad step sizes along direction **p** (for grad. descent, $\mathbf{p} = -\nabla f$):

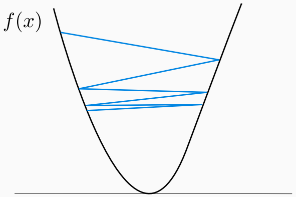


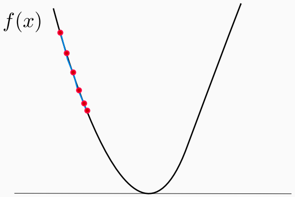


1) Big steps with little decrease 2) Small steps getting nowhere

Wolfe conditions to decide step-size

Bad step sizes along direction $\mathbf{p}$ (for grad. descent, $\mathbf{p} = -\nabla f$):





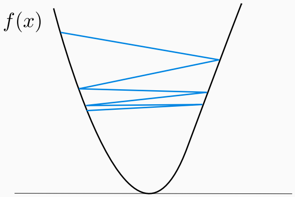
1) Big steps with little decrease 2) Small steps getting nowhere

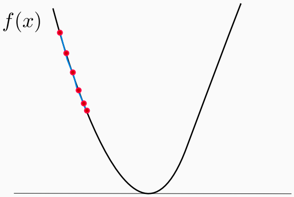
Avoid (1): Avg. decrease at least some fraction of initial rate:

$$f(\mathbf{x} + \eta\mathbf{p}) \leq f(\mathbf{x}) + \eta c_1(\nabla f \cdot \mathbf{p}), \qquad c_1 \in (0, 1) \ \textit{e.g.}\ 0.9$$

Wolfe conditions to decide step-size

Bad step sizes along direction $\mathbf{p}$ (for grad. descent, $\mathbf{p} = -\nabla f$):





1) Big steps with little decrease 2) Small steps getting nowhere

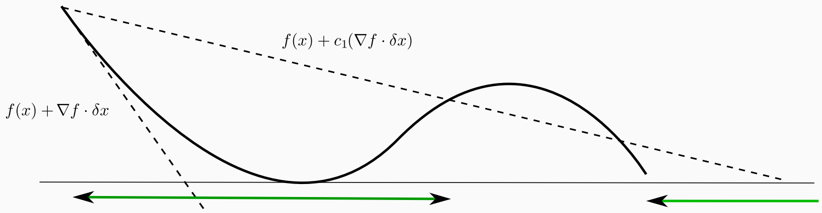
Avoid (1): Avg. decrease at least some fraction of initial rate:

$$f(\mathbf{x} + \eta\mathbf{p}) \leq f(\mathbf{x}) + \eta c_1(\nabla f \cdot \mathbf{p}), \qquad c_1 \in (0,1) \ e.g. \ 0.9$$

Avoid (2): Final rate is greater than some fraction of initial rate:

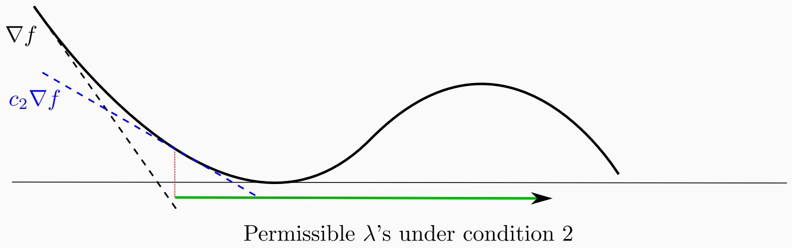
$$\nabla f(\mathbf{x} + \eta\mathbf{p}) \cdot \mathbf{p} \geq c_2 \nabla f(\mathbf{x}) \cdot \mathbf{p}, \qquad c_2 \in (0,1) \ e.g. \ 0.1$$

Wolfe conditions


$f(x) + c_1(\nabla f \cdot \delta x)$
$f(x) + \nabla f \cdot \delta x$

Permissible δx's under condition 1

Wolfe conditions



Wolfe conditions

A simple way to satisfy Wolfe conditions:

Set $\mathbf{p} = -\nabla f, c_1 = c_2 = .5$

Start with $\nu = 1$, and while condition i is not satisfied, set $\nu = \beta_i t$ (for $\beta_1 \in (0, 1), \beta_2 > 1$ and $\beta_1 * \beta_2 < 1$

Gradient flow

One way to understand/improve gradient descent is to view it as an approximation to ‘gradient flow’.

Write x_t for the position of a particle at time t, evolving as

$$\frac{\mathrm{d}\mathsf{x}_t}{\mathrm{d}t} = -\nabla f(\mathsf{x}_t), \quad \text{for some initialization at } t = 0.$$

x_t converges to x^*, the minimum of f as t increases

- At minimum, $\nabla f(\mathsf{x}^*) = 0$.

Typically, not easy to solve the differential eq. for x_t

Different algs can be seen as approximations to this ideal

Forward and backward methods

$$\frac{\Delta \mathsf{x}_t}{\Delta t} \approx \frac{\mathrm{d}\mathsf{x}_t}{\mathrm{d}t} = -\nabla f(\mathsf{x}_t) \quad \text{(forward Euler approximation)}$$

$$\implies \quad \mathsf{x}_{t+\Delta t} = \mathsf{x}_t - \Delta t \nabla f(\mathsf{x}_t)$$

This is just gradient descent with stepsize $\eta = \Delta t$

$$\mathsf{x}_{i+1} = \mathsf{x}_i - \eta \nabla f(\mathsf{x}_i)$$

Forward and backward methods

$$\frac{\Delta \mathbf{x}_t}{\Delta t} \approx \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -\nabla f(\mathbf{x}_t) \quad \text{(forward Euler approximation)}$$

$$\implies \quad \mathbf{x}_{t+\Delta t} = \mathbf{x}_t - \Delta t \nabla f(\mathbf{x}_t)$$

This is just gradient descent with stepsize $\eta = \Delta t$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_i)$$

Backward Euler approx $\implies \quad \mathbf{x}_{t+\Delta t} = \mathbf{x}_t - \Delta t \nabla f(\mathbf{x}_{t+\Delta t})$

For a step size η, the iterates are:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_{i+1})$$

The updates are implicit ($\mathbf{x}_{i+1}$ is on both LHS and RHS).
Why do we care?

Backward Euler method

Backward method: $\mathsf{x}_{i+1} = \mathsf{x}_i - \eta \nabla f(\mathsf{x}_{i+1})$

Claim: this is the same as solving

$$\mathsf{x}_{i+1} = \arg\min \; f(\mathsf{x}) + \frac{1}{2\eta}(\mathsf{x} - \mathsf{x}_i)^2$$

Backward Euler method

Backward method: $\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_{i+1})$

Claim: this is the same as solving

$$\mathbf{x}_{i+1} = \arg\min \; f(\mathbf{x}) + \frac{1}{2\eta}(\mathbf{x} - \mathbf{x}_i)^2$$

Also called a proximal point method

Backward Euler method

Backward method: $\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_{i+1})$

Claim: this is the same as solving

$$\mathbf{x}_{i+1} = \arg\min \; f(\mathbf{x}) + \frac{1}{2\eta}(\mathbf{x} - \mathbf{x}_i)^2$$

Also called a proximal point method

Now, we see that:

- $\mathbf{x}_{i+1} \leq \mathbf{x}_i$, unlike gradient descent. Has faster convergence.
- This works even if ∇f is not defined!
- Can be generalized to different distance functions:

$$\mathbf{x}_{i+1} = \arg\min \; f(\mathbf{x}) + \frac{1}{2\eta} d(\mathbf{x}, \mathbf{x}_i)$$

Other approaches to improving convergence

Newton's method: uses second derivatives/Hessians:

$$x_{i+1} = x_i - f'(x_i)/f''(x_i)$$

For vector-valued $\mathbf{x}$, writing $\mathrm{H}f(\mathbf{x}_i)$ for the Hessian of f at $\mathbf{x}_i$,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [\mathrm{H}f(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i)$$

Intuition:

- Stepsize is small when gradient is changing rapidly
- Each iteration uses $f(\mathbf{x}_i)$, $\nabla f(\mathbf{x}_i)$ and $\mathrm{H}f(\mathbf{x}_i)$ to construct a quadratic approximation to f, which is then minimized

Back to simple gradient descent

MLE: maximum likelihood estimation

Consider a set of observations $X = (x_1, \cdots, x_N)$.

Assume $x_i \sim p(x|\theta)$

$$\theta_{MLE} = \text{argmax}\, \ell(\theta) := \text{argmax} \sum_{i=1}^{N} \log p(x_i|\theta)$$

Back to simple gradient descent

MLE: maximum likelihood estimation

Consider a set of observations $X = (x_1, \cdots, x_N)$.

Assume $x_i \sim p(x|\theta)$

$$\theta_{MLE} = \text{argmax}\, \ell(\theta) := \text{argmax} \sum_{i=1}^{N} \log p(x_i|\theta)$$

The gradient of the log-likelihood is $\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$
(The average of the gradients of each datapoint.)

Back to simple gradient descent

MLE: maximum likelihood estimation

Consider a set of observations $X = (x_1, \cdots, x_N)$.

Assume $x_i \sim p(x|\theta)$

$$\theta_{MLE} = \text{argmax}\ \ell(\theta) := \text{argmax} \sum_{i=1}^{N} \log p(x_i|\theta)$$

The gradient of the log-likelihood is $\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$
(The average of the gradients of each datapoint.)

Starting with an initial θ_0, iterate:

$$\theta_{i+1} = \theta_i + \eta_i \nabla \ell(\theta_i)$$

Gradient descent (contd.)

$$\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$$

Cons:

- Calculating gradient requires evaluating likelihood N times. (Each iteration must cycle through all datapoints.)
- *Lots* of redundancy, esp. for large N.

Gradient descent (contd.)

$$\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$$

Cons:

- Calculating gradient requires evaluating likelihood N times. (Each iteration must cycle through all datapoints.)
- *Lots* of redundancy, esp. for large N.

Pros:

- Convergence is better understood.

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla \ell}$.

Typically split data into N/B batches of size B.
Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla \ell}(\theta) = \sum_{j \in B_i} \nabla \log p(x_j|\theta)$$

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla\ell}$.

Typically split data into N/B batches of size B.
Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla\ell}(\theta) = \sum_{j \in B_i} \nabla \log p(x_j|\theta)$$

Pros:

- Calculating the gradient is $O(B)$.
  (Often, each batch is just a single datapoint)
- Much faster convergence (just one sweep through the data can get you a decent solution).
- Often, you get better solutions.
- Useful for online systems, tracking θ that varies over time .

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla\ell}$.

Typically split data into N/B batches of size B.
Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla\ell}(\theta) = \sum_{j\in B_i} \nabla \log p(x_j|\theta)$$

Cons:

- Convergence analysis is harder.
- Noisy gradients mean the algorithm will never converge.
  Typically need to reduce the step size every iteration.
  We want

$$\eta_i \to 0, \quad \sum_{i=1}^{\infty} \eta_i = \infty$$

  E.g. $\eta_i = \frac{a}{b+i}$

Stochastic gradient descent with momentum

One way to accelarate convergence is to include a momentum term:

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla \ell}(\theta_i) + \beta_i \underbrace{(\theta_i - \theta_{i-1})}_{\text{momentum}}$$

Stochastic gradient descent with momentum

One way to accelarate convergence is to include a momentum term:

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla} \ell(\theta_i) + \beta_i \underbrace{(\theta_i - \theta_{i-1})}_{\text{momentum}}$$

More generally,

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla} \ell(\theta_i + \gamma(\theta_i - \theta_{i-1})) + \beta_i(\theta_i - \theta_{i-1})$$

Include many popular algorithms:

- Polyak's heavy ball method (HB): $\gamma = 0$
- Nesterov's accelerated gradient (NAG): $\gamma_i = \beta_i$

Adaptive methods

Adaptive methods accelerate convergence by using the entire history of iterates to determine step-sizes.

Adaptive methods

Adaptive methods accelerate convergence by using the entire history of iterates to determine step-sizes.

Often take the general form

$$\theta_{i+1} = \theta_i + \eta_i H_i^{-1} \widehat{\nabla \ell}(\theta_i + \gamma(\theta_i - \theta_{i-1})) + \beta_i H_i^{-1} H_{i-1}(\theta_i - \theta_{i-1})$$

where H_i is some combination of all previous gradients. E.g.

$$H_i = \mathrm{diag}\left(\sum_{j=1}^{i} g_j \circ g_j\right),$$

with $g_j = \widehat{\nabla \ell}(\theta_j + \gamma(\theta_j - \theta_{j-1}))$, and $\circ$ element-wise product.

Examples are AdaGrad, Adam etc.