Large-scale Optimization

Georgios Paschos

Director of Science, Amazon

Introduction

- '02–'06 PhD – **University of Patras**
- '07–'08 Postdoc – **VTT, Helsinki**
- '08–'12 Lecturer – **University of Thessaly, Volos**
- '12–'14 Researcher – **MIT, Cambridge**
- '14–'19 Principal Scientist – **Huawei Technologies, Paris**
- '19–now Director, Applied Science – **Amazon, Luxembourg**

Best way to reach out about the course: gpasxos@gmail.com

Mathematical Programming

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned}$$

- $x \in \mathbb{R}^n$ points available to choose
- $f : \mathbb{R}^n \to \mathbb{R}$ objective function to minimize
- $g_i(x) \leq 0$ the inequality constraints
- $h_j(x) = 0$ the equality constraints

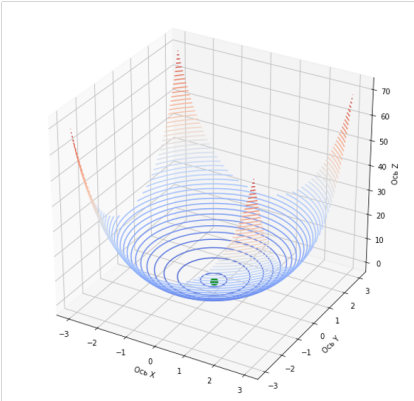


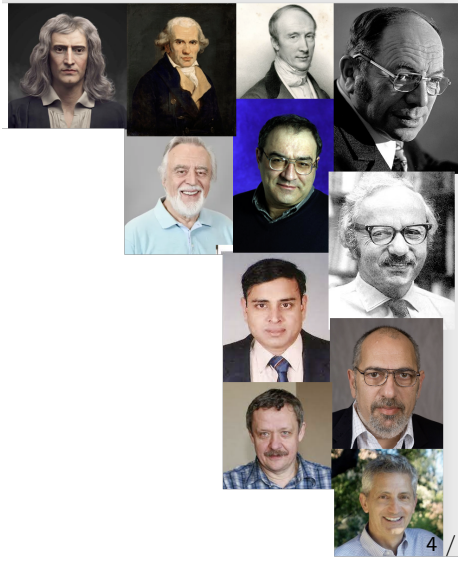
Fig from stackoverflow

Computer Sci, Engineering, Data Sci, Operation Research, Machine Learning, Economics, etc

amazon

History

- Newton 1669 (Newton's method)
- Monge 1781 (optimal transport)
- Cauchy 1847 (steepest descent)
- Kantorovich 1939 (linear programming)
- Dantzig 1947 (simplex method)
- Karush/Kuhn-Tucker 1951 (KKT)
- Robbins et al 1952 (SGD)
- Glowinski et al 1975 (ADMM)
- Nemirovski 1979 (mirror descent)
- Khatchiyan 1979 (ellipsoid)
- Karmarkar 1984 (interior point)
- Orlin 1994 (network simplex)



More: mathematical-tours.github.io/algorithms/

amazon

Where is it used today?

- Finance: portfolio optimization, utility theory
- Decision making with operation research
- Facility planning and flow optimization: trains, airports, petroleum industry, supply chain, etc
- Communication networks: information theory, congestion control protocols, medium access, etc
- Training ML models: backpropagation algorithm, regression
- Supervised/unsupervised learning: RL, clustering
- Transportation, logistics and engineering
- Robotics: model predictive control
- Image/signal processing
- **ADD YOUR FAVORITE...**

Quiz

- Which of the following is different from the rest?
  - Degree in computer science
  - Degree in engineering
  - Degree in pure mathematics
  - A large pizza

Quiz

- Which of the following is different from the rest?
  - Degree in computer science
  - Degree in engineering
  - Degree in pure mathematics
  - A large pizza

With the rest you can feed a family of 4!

Q: How do we solve optimization problems in real-life?

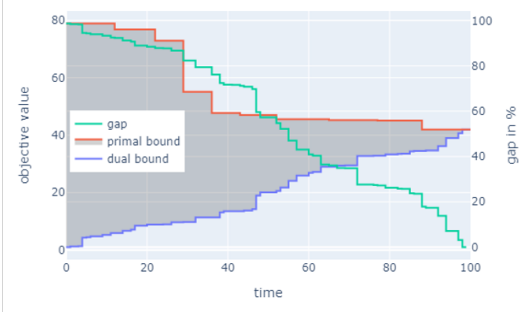
⇒ Cvx-opt - `https://cvxopt.org/`

⇒ SCIP

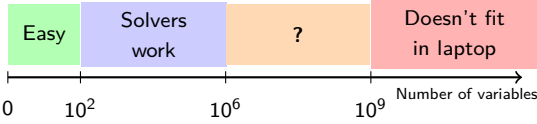
- AIMMS, AMPL, APMonitor, ASTOS, CPLEX
- Couenne – open source for MINLPs
- FICO Xpress, Galahad, GEKKO Python
- Gurobi - free for academic users
- LIONsolver, MIDACO, MINTO
- MOSEK – large scale optimization
- OptimJ, PottersWheel, Pyomo, WORHP

Sometimes solvers fail and you are needed

- LPs: today we solve ∼1M variables/1M constraints (Simplex, IPM, PDLP)

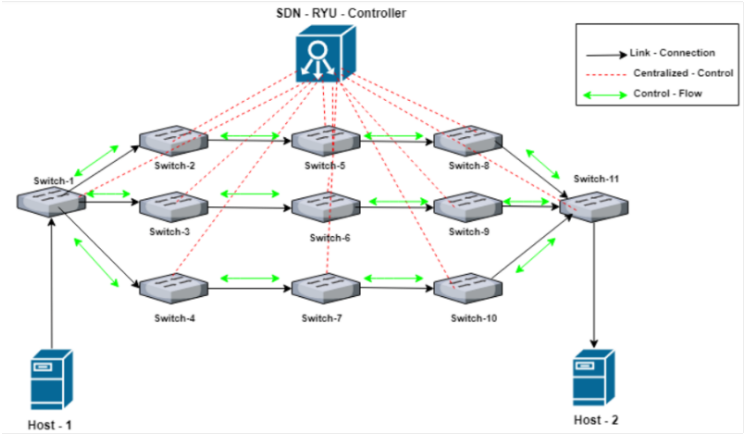


From Gurobi



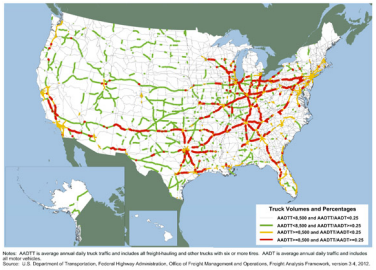
Large-scale example: Internet flow optimization

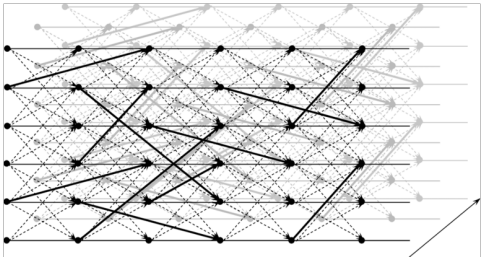
- Huawei SDN controllers: 10k od pairs, 10k links, **100M variables**



Large-scale example: truck scheduling

- Scheduling Amazon's trucks with time-expanded multicommodity flow in Europe
  - 10^3 commodities, 10^2 time slots, 10^2 nodes $\Rightarrow$ **100M var**





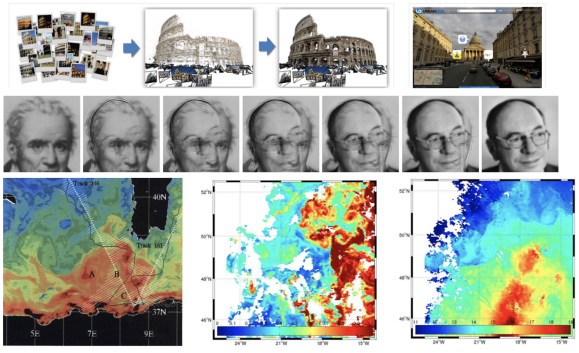
Question

- You have a linear program with 10 million variables representing Amazon's European truck routing.
- **Scenario A**: Gurobi gives you a solution in 2 hours that is provably optimal.
- **Scenario B**: A first-order method gives you a solution in 5 minutes that is 99.5% optimal.
- **Which would you choose and why?**
- **What factors influence this decision in practice?**

Consider: computational cost, business value, time constraints, solution quality needs.

Large-scale example: image processing

- Computer vision
  - 4Mpixels images $\Rightarrow$ $\mathbf{16}\times 10^{12}$ **variables** / pairs of pixels



reconstruction

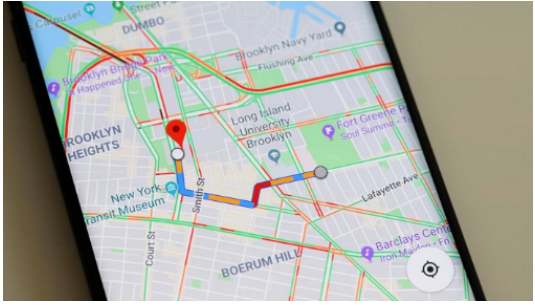
interpolation

weather model calibration

From Papadakis PhD

Large-scale example: Paths in Maps

- Finding a path in Google maps **>100M variables**
  - How does Google maps find the fastest path?



Visualization of a* algorithm https://www.youtube.com/watch?v=CgW0HPHqFE8

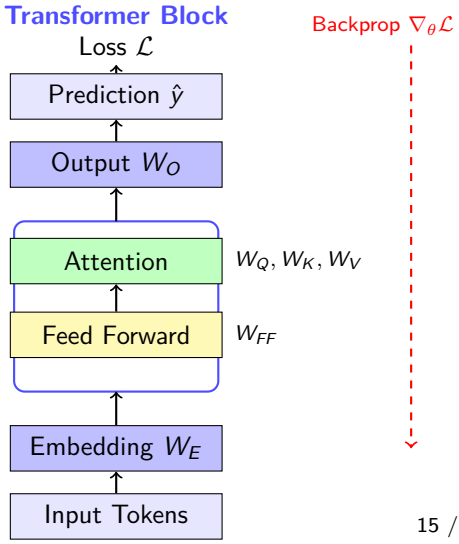
Large-scale example: training ChatGPT

Training = Optimization Problem

Find weights $\theta = \{W_E, W_Q, W_K, W_V, W_{FF}, W_O\}$ that minimize the loss:

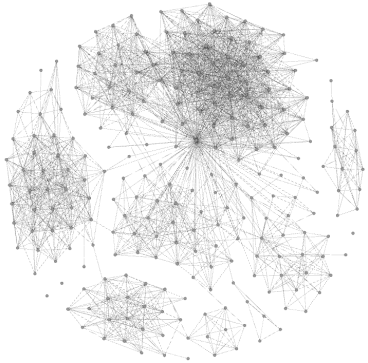
$$\underset{\theta}{\text{minimize}} \quad \mathcal{L}(\theta) = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i(\theta))$$

- GPT-3 davinci: **175 Billion** parameters
- LLaMA-70B: **70 Billion** parameters



Large-scale example: big data clustering

- Finding a community in the Facebook graph $>$**100M variables**



https://snap.stanford.edu/data/

amazon

Large-scale optimization course objectives

- Learn how to...
  - Recognize/classify an optimization problem
  - Solve optimization problems
  - Choose the right method at scale
  - Design new algorithms
- Focus on **convex programming**
- More broadly: how to employ mathematical modeling, algorithms, and python to solve problems.

Complexity map of optimization

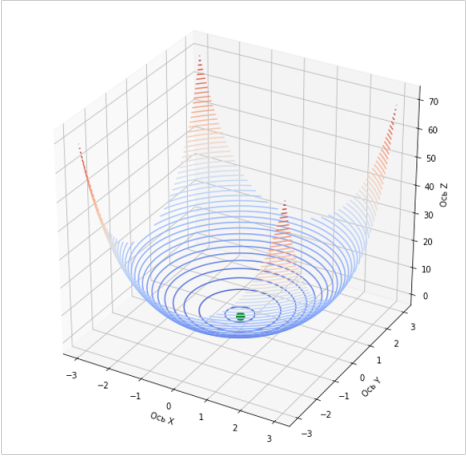
	Continuous	Discrete
Linear	**Linear** programming	**Integer** programming
Non-Linear but convex	**Convex** programming	
Non-Convex		

Question

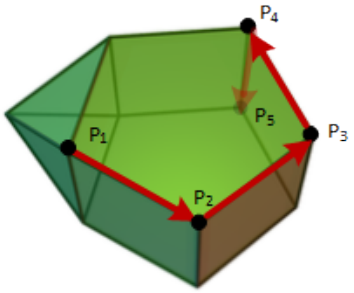
- Consider the problem of training a neural network with 1 billion parameters.
- **Where would you place this problem in our complexity map?**
  - Continuous or discrete?
  - Linear or non-linear?
  - Convex or non-convex?
- **Why is it still solvable in practice despite its classification?**

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{aligned}$$

- Introduction to convexity
- First-order methods
- Training neural networks
- Newton methods
- Semi-definite programming ← **Alexios**

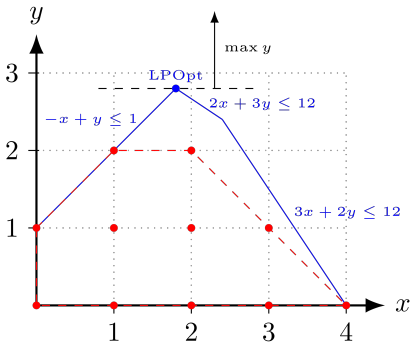


$$\begin{aligned} \underset{x \geq 0}{\text{maximize}} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$



- Simplex method
- LP duality
- Column generation
- **Interior point method** Newton-based
- **Primal-Dual (PDLP)** FOM

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & x \in \mathbb{Z}^n \\ & Ax \leq b \end{aligned}$$



- Complexity – NP-complete problems
- Total unimodularity and shortest paths
- Branch and bound
- Cutting planes and Bender's cut
- Lagrangian relaxation
- Submodularity and greedy (max cover)
- LP Rounding (set cover)
- Heuristics

amazon

Resources

- Presentations
  - **[Paschos]_lsopt_pres.pdf**
- Lecture notes
  - **[Paschos]_lsopt_notes.pdf** ongoing work - feedback welcomed
- Quizes
  - **Train multi-layer perceptron and achieve the best performance on MNIST**
  - **Implement PDHG and Interior Point for LP and compare them**
  - Submit quiz reports by 31/03/2026
- Python notebooks
  - https://github.com/gpasxos/large-scale-optimization
- External references available on Internet (**see next slide**)

References used in this course

[1] S. Boyd and L. Vandenberghe, "Convex Optimization", https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[2] R. Tibshirani, "Convex Optimization", https://www.stat.cmu.edu/~ryantibs/convexopt-F18/

[3] C. Caramanis, "Optimization algorithms", https://www.youtube.com/playlist?list=PLXsmhnDvpjORzPelSDs0LSDrfJcqyLlZc

[4] C. Papadimitriou and K. Steiglitz, "Combinatorial Optimization, Algorithms and Complexity"

[5] G. Peyre, "Numerical tours", https://www.numerical-tours.com/

[6] D. Bertsekas, "Non-linear programming"

[7] Y. Nesterov, "Introductory lectures on convex optimization"

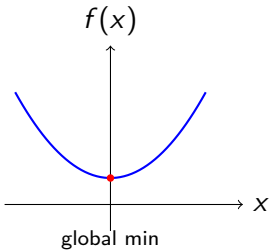
Linear Algebra refresher Prof. Caramanis: https://www.youtube.com/watch?v=FTzZff0uq9U

Part I

Introduction to Convexity

Convex vs Non-Convex

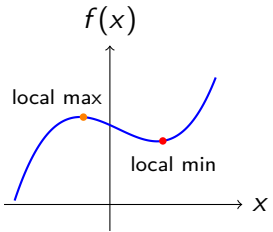
Convex



- Local min $\Rightarrow$ Global min
- Easy to find optimum
- Polynomial-time algorithms

Welcome to Paradise!

Non-Convex



- Can have many local extrema
- Can be very hard
- Worst-case exponential time

Abandon all hope!

Applications of Convex Programming

Finance & Economics:

- Portfolio optimization
- Optimal advertising
- Utility maximization

Machine Learning:

- Linear/Ridge/Lasso regression
- Logistic regression
- Support Vector Machines
- Neural network training

Engineering:

- Power allocation (wireless)
- Internet congestion control
- Network slicing
- Signal processing

Operations Research:

- Resource allocation
- Combinatorial relaxations

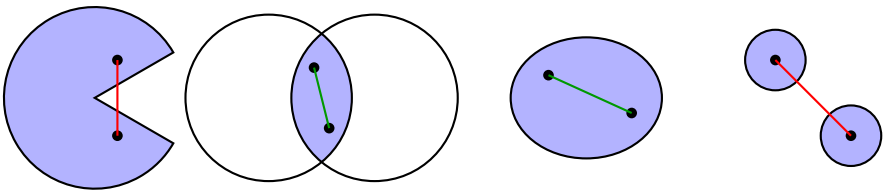
Outline

- **Convex sets** – definitions and examples
- **Convex functions** – definitions and examples
- **Operations preserving convexity**
- **Optimality conditions**
- **Smoothness and strong convexity**

Convex Sets

Definition. A set $\mathcal{X} \subseteq \mathbb{R}^n$ is **convex** if for any $x, y \in \mathcal{X}$:

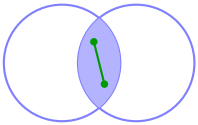
$$\theta x + (1-\theta)y \in \mathcal{X} \quad \text{for all } \theta \in [0,1]$$



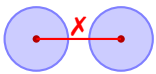
Non-convex Convex Convex Non-convex

Operations Preserving Set Convexity

Intersection: If $\mathcal{X}_1, \mathcal{X}_2$ are convex, then $\mathcal{X}_1 \cap \mathcal{X}_2$ is convex



Intersection is convex

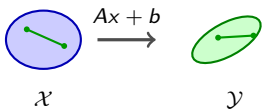


Union is NOT convex

Adding constraints to an optimization problem $\equiv$ intersecting with constraint sets

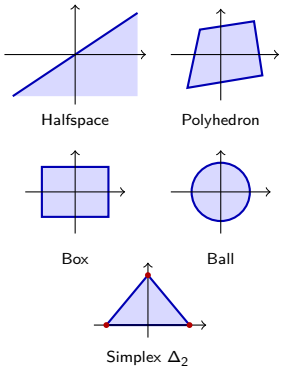
Affine maps: If $\mathcal{X}$ is convex and $y = Ax + b$, then

- $\mathcal{Y} = \{Ax + b : x \in \mathcal{X}\}$ is convex



Examples of Convex Sets

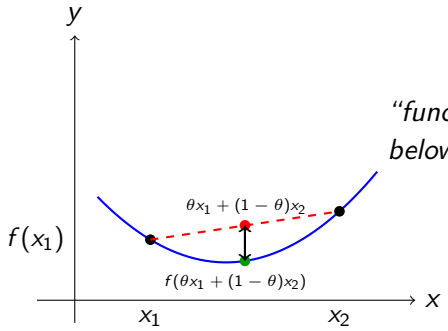
- **Hyperplane:** $\{x : a^\top x = b\}$
  Budget constraints, conservation laws, normalization
- **Halfspace:** $\{x : a^\top x \leq b\}$
  Capacity limits, resource constraints
- **Polyhedron:** $\{x : Ax \leq b,\ Cx = d\}$
  LP feasible regions, network flows, supply chains
- **Box:** $\{x : l \leq x \leq u\}$
  Variable bounds (prices, production levels)
- **Nonnegative orthant:** $\mathbb{R}^n_+ = \{x : x \geq 0\}$
  Physical quantities, costs, probabilities
- **Euclidean ball:** $\{x : \|x - x_0\|_2 \leq r\}$
  Regularization, trust regions, uncertainty sets
- **Simplex:** $\Delta_n = \{x : \sum_i x_i = 1,\ x \geq 0\}$
  Portfolio weights, probability distributions amazon



Convex Functions

Definition. A function $f : \mathcal{X} \to \mathbb{R}$ (where $\mathcal{X}$ is convex) is **convex** if:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y) \quad \text{for all } x, y \in \mathcal{X},\ \theta \in [0,1]$$



Q: *What if the gap is zero for all θ?*

Examples of Convex Functions on $\mathbb{R}$

Convex:

- Linear: $ax + b$
- Quadratic: $ax^2 + bx + c \quad (a > 0)$
- Exponential: e^{ax}
- Powers: x^a on $\mathbb{R}_+$ for $a \geq 1$ or $a \leq 0$
- Negative entropy: $x \log x$ on $\mathbb{R}_+$
- Piecewise linear: $\max_i (a_i x + b_i)$

Concave:

- Logarithm: $\log x$ on $\mathbb{R}_{++}$
- Powers: x^a on $\mathbb{R}_+$ for $0 < a < 1$
- Square root: $\sqrt{x}$

Both convex and concave:

- Affine: $ax + b$

Examples of Convex Functions on $\mathbb{R}^n$

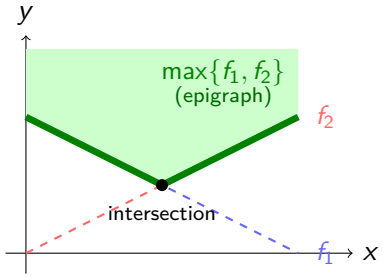
- **Affine:** $a^\top x + b$
- **Quadratic:** $\frac{1}{2}x^\top Qx + b^\top x + c$ where $Q \succeq 0$
- **ℓ_p-norms:** $\|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p}$ for $p \geq 1$
- **Max function:** $\max\{x_1, \ldots, x_n\}$
- **Log-sum-exp:** $\log\left(\sum_{i=1}^n e^{x_i}\right)$ (smooth approximation to max)
- **Negative entropy:** $\sum_i x_i \log x_i$

Concave:

- **Log-determinant:** $\log\det(X)$ on $\mathbb{S}^n_{++}$
- **Geometric mean:** $\left(\prod_i x_i\right)^{1/n}$ on $\mathbb{R}^n_+$

Operations Preserving Function Convexity

- **Nonnegative weighted sum:** $f = w_1 f_1 + \cdots + w_m f_m \quad (w_i \geq 0)$
- **Composition with affine:** $g(x) = f(Ax + b)$
- **Pointwise maximum:** $f(x) = \max\{f_1(x), \ldots, f_m(x)\}$
- **Pointwise supremum:** $f(x) = \sup_{y \in \mathcal{Y}} g(x, y)$



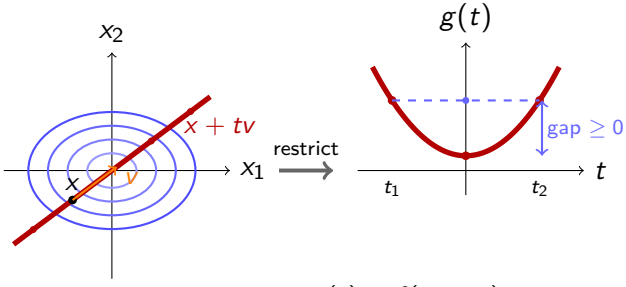
Epigraph of $\max\{f_1, f_2\}$ = intersection of epigraphs (both convex) $\Rightarrow$ convex

Restriction of f to a Line

Key insight: We can check convexity using 1D restrictions

Consider the function $g(t) = f(x + tv)$ where $t \in \mathbb{R}$, $x, v \in \mathbb{R}^n$

$$\boxed{g \text{ is convex in } t \text{ for all } x, v \iff f \text{ is convex}}$$



domain of f with level sets $g(t) = f(x + tv)$ **is convex**

We can disprove convexity in $\mathbb{R}^n$ by finding one violating point in $\mathbb{R}$

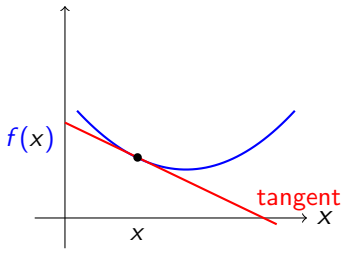
First-Order Condition for Convexity

Theorem. Let f be differentiable on convex $\mathcal{X}$. Then f is convex iff:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \text{for all } x, y \in \mathcal{X}$$

Interpretation:

- Function lies above all its tangent planes



Important consequence:
If $\nabla f(x^*) = 0$, then for all y:

$$f(y) \geq f(x^*) + 0 = f(x^*)$$

Hence $\nabla f(x^*) = 0$ is a sufficient condition for x^* to be **global minimum**!

Q: *Does that mean we can use $\nabla f(x) = 0$ to solve all convex programs?*

amazon

Second-Order Condition for Convexity

Theorem. Let f be twice differentiable. Then f is convex iff:

$$\nabla^2 f(x) \succeq 0 \quad \text{for all } x \in \mathcal{X}$$

where the Hessian is:

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

Interpretation:

- Hessian is symmetric: $\nabla^2 f(x) = \sum_i^n \lambda_i e_i e_i^T$
- All eigenvalues of Hessian are ≥ 0
- Curvature in every direction v is non-negative: $v^T \nabla^2 f(x) v = \sum_i^n \lambda_i v_i^2 \geq 0$
- If $\nabla^2 f(x) \succ 0$ (strictly positive definite), then f is **strictly convex**

Example: Log-Sum-Exp is Convex

Log-sum-exp: $f(x_1, x_2) = \log(e^{x_1} + e^{x_2})$ (smooth approximation to max)

Step 1: Gradient

$$\nabla f = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

where $p_i = \text{softmax}(x)_i, \quad p_1 + p_2 = 1$

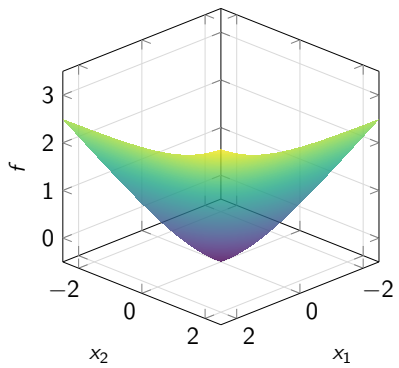
Step 2: Hessian

$$\nabla^2 f = \begin{bmatrix} p_1(1-p_1) & -p_1 p_2 \\ -p_1 p_2 & p_2(1-p_2) \end{bmatrix}$$

Since $p_1 + p_2 = 1$: $\quad \nabla^2 f = p_1 p_2 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

Step 3: Check $\nabla^2 f \succeq 0$ For any $v \in \mathbb{R}^2$:

$$v^\top \nabla^2 f \, v = p_1 p_2 (v_1 - v_2)^2 \geq 0 \ \checkmark$$



Surface curves upward in all directions

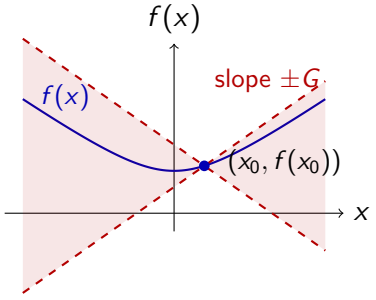
General case: $f(x) = \log \sum_{i=1}^{n} e^{x_i}$ has $\nabla^2 f = \text{diag}(p) - pp^\top \succeq 0$ (used in softmax, attention)

Lipschitz Continuity

Definition. Function f is G-**Lipschitz continuous** if:

$$|f(x) - f(y)| \leq G\|x - y\| \quad \text{for all } x, y \in \mathcal{X}$$

Equivalent condition (for differentiable f): $\|\nabla f(x)\| \leq G$



Bounded gradient norm $\Longrightarrow$ blanket assumption for most proofs

Examples: Lipschitz Continuity

Recall: f is G-Lipschitz if $|f(x) - f(y)| \leq G\|x - y\|$ (equivalently: $\|\nabla f(x)\| \leq G$)

1. **Linear:** $f(x) = a^\top x$

$$\nabla f = a \;\Rightarrow\; G = \|a\|$$

2. **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq \frac{1}{4}$$

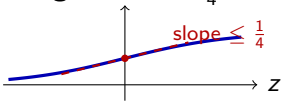
$\Rightarrow G = \frac{1}{4}$

3. **ReLU:** $f(z) = \max(0, z)$

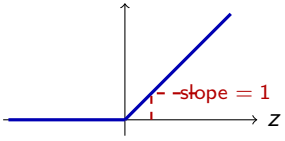
$$f'(z) \in \{0, 1\} \;\Rightarrow\; G = 1$$

4. **Softmax cross-entropy:**

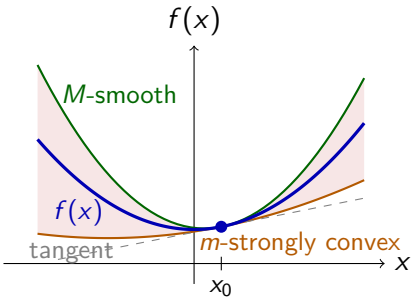
Sigmoid: $G = \frac{1}{4}$



ReLU: $G = 1$



Can we sandwich f within quadratic bounds?



M-**Smoothness:** $f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{M}{2}\|x - y\|^2$

m-**Strong convexity:** $f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{m}{2}\|x - y\|^2$

For $m = 0$: recover first-order convexity condition

Consequence of Strong Convexity

Useful bound: From strong convexity, minimizing the RHS over y:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{m}{2}\|x - y\|^2$$

Setting $\nabla_y(\text{RHS}) = 0$ gives $y^* = x - \frac{1}{m}\nabla f(x)$

Substituting back:

$$f(y) \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|^2 \quad \text{for all } y$$

Key insight:

$$f(x) - f^* \leq \frac{1}{2m}\|\nabla f(x)\|^2$$

Small gradient $\Rightarrow$ we're close to the minimum — provides stopping criterion!

Strong Convexity: Examples

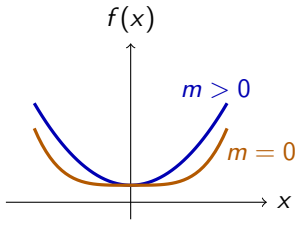
Recall: f is m-strongly convex if $\nabla^2 f(x) \succeq mI$ for all x

Strongly convex:

Function	m
$f(x) = \frac{1}{2}\|x\|^2$	1
$f(x) = \frac{1}{2}x^\top Qx, \quad Q \succ 0$	$\lambda_{\min}(Q)$
$f(x) = \|Ax - b\|^2, \quad A$ full rank	$2\lambda_{\min}(A^\top A)$

Not strongly convex:

$f(x) = \|Ax - b\|^2$	A rank-deficient
$f(x) = \|x\|_1$	not even smooth
$f(x) = \log \sum_i e^{x_i}$	$\lambda_{\min}(\nabla^2 f) = 0$



Blue: strongly convex (quadratic) Orange: convex, not strongly (x^4)

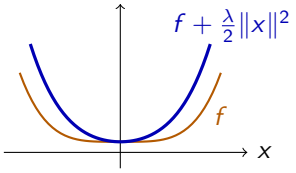
Regularization ⇒ Strong Convexity

Key trick: Adding $\frac{\lambda}{2}\|x\|^2$ to any convex f makes it strongly convex

Claim: If f is convex, then

$$g(x) = f(x) + \frac{\lambda}{2}\|x\|^2$$

is λ-strongly convex.



Proof:

$$\nabla^2 g(x) = \underbrace{\nabla^2 f(x)}_{\succeq 0} + \lambda I \succeq \lambda I \quad \checkmark$$

Ridge regression: $\|Ax - b\|^2 + \lambda\|x\|^2$

Weight decay: $\mathcal{L}(w) + \frac{\lambda}{2}\|w\|^2$

$\Rightarrow$ Unique minimum, faster convergence, better conditioning

Condition Number

Definition: The **condition number** of a smooth, strongly convex function:

$$\kappa = \frac{M}{m} \geq 1$$

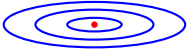
Well-conditioned (κ small):

- Nearly spherical level sets
- Fast convergence
- Algorithms work well

Ill-conditioned (κ large):

- Elongated level sets
- Slow convergence
- Algorithms struggle

$\kappa \approx 1$



$\kappa \gg 1$

Function Properties for Algorithm Analysis

Property	**Hessian**	**What it gives us**
Convex	$\nabla^2 f \succeq 0$	Local min = global min; no need to escape local minima
G-Lipschitz	$\|\nabla f\| \leq G$	Bounded gradients; controls step size in gradient & subgradient methods
M-smooth	$\nabla^2 f \preceq MI$	Bounded curvature; guarantees descent with step size $\eta = 1/M$; no line search needed
m-str. convex	$\nabla^2 f \succeq mI$	Unique minimum; linear convergence rate $O(\rho^k)$; stopping criterion

Condition number $\kappa = M/m$ determines convergence speed; smaller is better

Q: *Which is your favorite function now?*

amazon

Convex Programs

Definition 2.11 (Convex program). An optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{aligned}$$

is a convex program if:

- f is convex
- g_i are convex for all i
- Equality constraints are affine

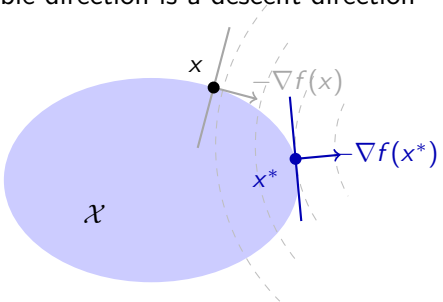
Key property: Every local optimum is a global optimum!

First-Order Optimality Condition for Convex Programs

Theorem. For a convex program with differentiable f, x^* is optimal if and only if:

$$\nabla f(x^*)^\top (y - x^*) \geq 0 \quad \text{for all feasible } y \in \mathcal{X}$$

Interpretation: No feasible direction is a descent direction



Special case – Unconstrained: Plugin $y = x^* - \nabla f(x^*)$ to get $\boxed{\nabla f(x^*) = 0}$

Optimality via $\nabla f = 0$: A System of Equations

Problem: $\min_{x \in \mathbb{R}^n} f(x)$ (unconstrained)

First-order necessary condition: $\nabla f(x^*) = 0$

System of n (generally nonlinear) equations

$$\begin{aligned} \frac{\partial f}{\partial x_1}(x^*) &= 0 \\ &\vdots \\ \frac{\partial f}{\partial x_n}(x^*) &= 0 \end{aligned}$$

Convex f:

- $\nabla f(x^*) = 0$ is **sufficient**
- Every solution is a **global minimum**

Non-convex f:

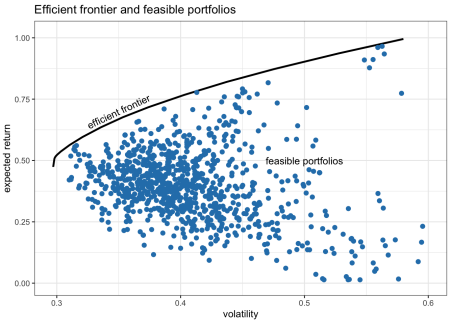
- $\nabla f(x^*) = 0$ is only **necessary**
- Solutions are **local extrema**

Q: *Why can't we just find all solutions to this system and compare?*

Example: Portfolio Optimization

Problem: Minimize risk for target return

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} x^\top \Sigma x \\ \text{subject to} \quad & \mathbf{1}^\top x = 1 \\ & \mu^\top x \geq r_{\min} \\ & x \geq 0 \end{aligned}$$



Efficient frontier: return vs. risk

- $\Sigma \succ 0$: covariance matrix
- μ: expected returns
- x_i: fraction in asset i

✓ Convex QP: $\Sigma \succ 0$

Example: Linear Regression

Least squares:

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2$$

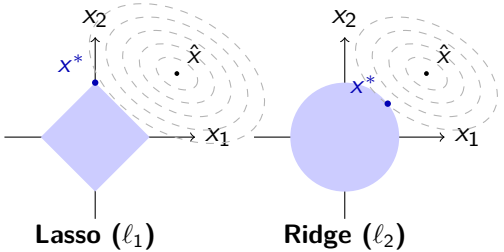
Ridge (L2 regularization):

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \lambda \|x\|_2^2$$

LASSO (L1, sparse solutions):

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1$$

✓ All three are convex!



LASSO vs Ridge constraint geometry

Geometry: Equivalent to $\min_x \|Ax - b\|^2$ s.t. $\|x\|_p \leq r$. Solution is where smallest ellipse touches the boundary. ℓ_1 ball has corners $\Rightarrow$ solution lands on axis $\Rightarrow$ **sparse!**

Example: Logistic Regression

Binary classification: $y_i \in \{0, 1\}$

Model:

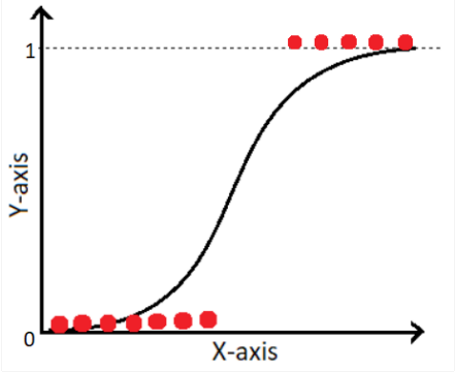
$$P(Y = 1 \mid x) = \frac{e^{a^\top x + b}}{1 + e^{a^\top x + b}}$$

Minimize negative log-likelihood:

$$\underset{a,b}{\text{minimize}} \sum_{i=1}^{m} [-y_i z_i + \log(1 + e^{z_i})]$$

where $z_i = a^\top x_i + b$

✓ Convex: $\log(1 + e^z)$ is convex



Sigmoid function & decision boundary

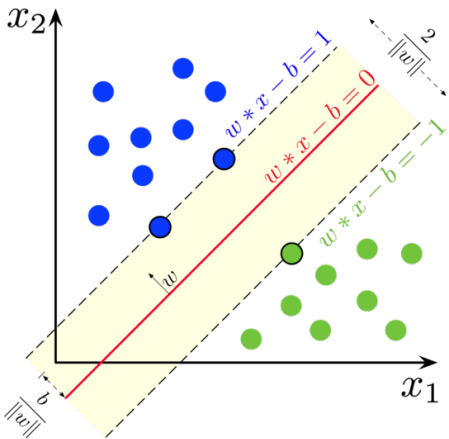
Example: Support Vector Machines

Maximum margin classifier:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

Equivalent (hinge loss):

$$\underset{w,b}{\text{minimize}} \quad \frac{1}{2}\|w\|^2 + C\sum_{i}[1 - y_i(w^\top x_i + b)]_+$$



Maximum margin & support vectors

✓ Convex QP

From Professor Winston: https://www.youtube.com/watch?v=_PwhiWxHK8o

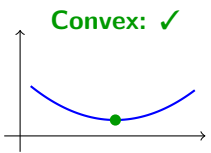
Why Convexity Matters

Convex programs are tractable:

- ✓ Local optima are global
- ✓ Polynomial-time algorithms
- ✓ Scalable to millions of variables
- ✓ Duality gives optimality certificates
- ✓ Rich theory for analysis

Recipe for success:

1. Recognize/formulate as convex program
2. Apply standard solver
3. Get guaranteed global optimum



Many local minima ⇒ hard to find global

Next: Gradient descent, Newton's method, Interior point methods

References for Convexity

[1] S. Boyd and L. Vandenberghe, "Convex Optimization", https://web.stanford.edu/~boyd/cvxbook/

[2] R. Tibshirani, "Convex Optimization", https://www.stat.cmu.edu/~ryantibs/convexopt-F18/

[3] Y. Nesterov, "Introductory Lectures on Convex Optimization"

[4] D. Bertsekas, "Convex Optimization Theory"

Part II

First-Order Methods for Unconstrained Optimization

Outline

Goal: Solve unconstrained convex optimization

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x)$$

using only function values $f(x)$ and gradients $\nabla f(x)$

- **Gradient descent** – the fundamental algorithm
- **Convergence analysis** – rates under different assumptions
- **Subgradient methods** – handling non-smooth functions
- **Lower bounds** – fundamental limits
- **Accelerated methods** – momentum and Nesterov's method

Iterative Algorithms

All methods generate a sequence $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$ converging to x^*

General update:

$$x^{(k+1)} = x^{(k)} + \eta^{(k)} d^{(k)}$$

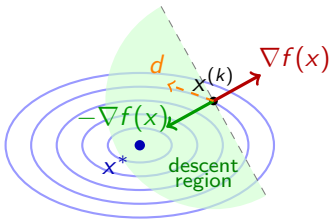
Key design choices:

- Direction $d^{(k)}$
- Step size $\eta^{(k)} > 0$

Key questions:

- Does it converge?
- How many iterations for $f(x^{(k)}) - f^* \leq \epsilon$?

Descent direction: $d^T \nabla f(x) < 0$



Any d with $d^T \nabla f(x) < 0$ decreases f; steepest descent uses $d = -\nabla f(x)$

Gradient Descent

$$x^{(k+1)} = x^{(k)} - \underbrace{\eta^{(k)}}_{\text{step size}} \underbrace{\nabla f(x^{(k)})}_{\text{direction } d}$$

Why negative gradient?

- $-\nabla f(x)$ points to locally steepest *decrease*
- The *direction* of $-\nabla f(x)$ minimizes $d^T \nabla f(x)$ over unit d

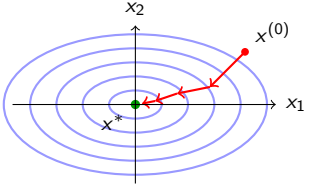
Motivation via quadratic model:

Minimize
$\hat{f}(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\eta} \|y - x\|^2$

Setting $\nabla_y \hat{f} = 0$: $\quad y = x - \eta \nabla f(x)$

$\Rightarrow$ Each step minimizes a local quadratic model

Augustin-Louis Cauchy (1847)



Example 1d: Gradient Descent on $f(x) = 2x^2 + 1$

Setup:

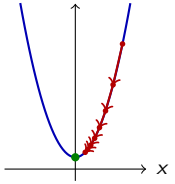
- Optimum: $x^* = 0$
- Gradient: $\nabla f(x) = 4x$
- Update: $x^{(k+1)} = (1 - 4\eta)x^{(k)}$

Convergence:

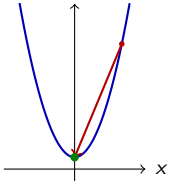
- $0 < \eta < \frac{1}{4}$: monotone
- $\eta = \frac{1}{4}$: one step!
- $\frac{1}{4} < \eta < \frac{1}{2}$: oscillates
- $\eta \geq \frac{1}{2}$: diverges

Key insight:

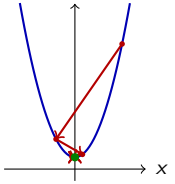
Step size must be small enough to converge, but larger $\Rightarrow$ faster



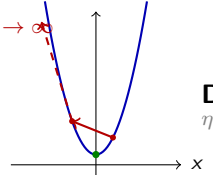
Too small
$\eta = 0.05$



Optimal
$\eta = 0.25$



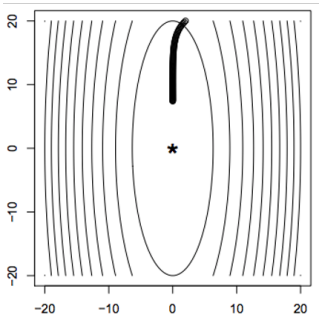
Oscillating
$\eta = 0.35$



Diverging
$\eta = 0.6$

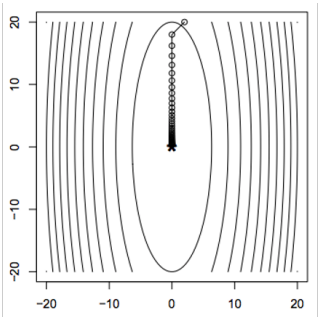
amazon

Effect of Step Size



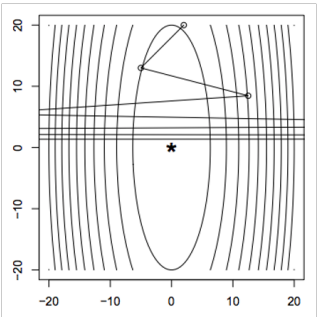
Too small

slow convergence



Just right

fast & stable



Too big

oscillates/diverges

Step Size Selection

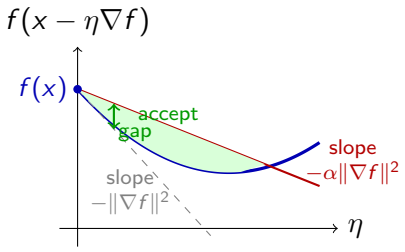
Four common strategies:

1. **Constant:** $\eta^{(k)} = \eta$ for all k
   - Simple, but requires knowing smoothness constant M
2. **Diminishing:** $\eta^{(k)} = c/\sqrt{k+1}$ or $\eta^{(k)} = c/(k+1)$
   - Conditions: $\sum_k \eta^{(k)} = \infty$ and $\sum_k (\eta^{(k)})^2 < \infty$
   - Essential for stochastic gradient descent
3. **Exact line search:** $\eta^{(k)} = \arg\min_{\eta \geq 0} f(x^{(k)} - \eta \nabla f(x^{(k)}))$
   - Optimal along gradient direction, but expensive
4. **Backtracking (Armijo's rule):** adaptive, practical
   - Start large, shrink until sufficient decrease

Armijo Rule (Backtracking Line Search)

Idea: Accept step η if it gives sufficient decrease:

$$f(x - \eta \nabla f(x)) \leq f(x) - \alpha \eta \|\nabla f(x)\|^2, \quad \alpha \in (0, 0.5)$$



Why terminates: Function slope $>$ Armijo slope at $\eta = 0$

$\Rightarrow$ gap always exists for small η

Backtracking:

1. Start with $\eta = 1$
2. While condition fails: $\eta \leftarrow \beta \eta$
3. Return η

Typical: $\alpha = 0.3$, $\beta = 0.5$

α	Effect
small	large steps
large	small steps
> 0.5	may reject $\frac{1}{M}$

Convergence Analysis: Setup

Recall function properties:

Property	Condition	Meaning
G-Lipschitz	$\|\nabla f(x)\| \leq G$	bounded gradients
M-smooth	$\nabla^2 f(x) \preceq MI$	bounded curvature
m-strongly convex	$\nabla^2 f(x) \succeq mI$	curved away from flat

Notation:

- $R = \|x^{(0)} - x^*\|$ (initial distance to optimum)
- $f_k^{\text{best}} = \min_{i \leq k} f(x^{(i)})$ (best value found so far)
- $\kappa = M/m$ (condition number)

More assumptions $\Rightarrow$ faster convergence rates

Convergence of Gradient Descent

Theorem (Convergence rates)

Let f be convex with bounded gradients $\|\nabla f(x)\| \leq G$.

1. **Lipschitz case:** With $\eta^{(k)} = R/(G\sqrt{k})$:

$$f_k^{\text{best}} - f^* \leq \frac{RG}{\sqrt{k}} \qquad \text{(convergence in } O(1/\epsilon^2) \text{ steps)}$$

2. **Smooth case:** If additionally M-smooth, with $\eta = 1/M$:

$$f_k^{\text{best}} - f^* \leq \frac{MR^2}{2k} \qquad \text{(convergence in } O(1/\epsilon) \text{ steps)}$$

3. **Strongly convex case:** If additionally m-strongly convex, with $\eta = 1/M$:

$$f(x^{(k)}) - f^* \leq \left(1 - \frac{1}{\kappa}\right)^k (f(x^{(0)}) - f^*) \quad (O(\kappa \log(1/\epsilon)) \text{ steps})$$

amazon

Proof Sketch: Lipschitz Case

Step 1: Track distance to optimum

$$\|x^{(k+1)} - x^*\|^2 = \|x^{(k)} - \eta^{(k)}\nabla f(x^{(k)}) - x^*\|^2 \qquad \text{expand norm} \|a-b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2$$

$$= \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}\nabla f(x^{(k)})^T(x^{(k)} - x^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Proof Sketch: Lipschitz Case

Step 1: Track distance to optimum

$$\|x^{(k+1)} - x^*\|^2 = \|x^{(k)} - \eta^{(k)}\nabla f(x^{(k)}) - x^*\|^2 \qquad \text{expand norm} \|a-b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2$$

$$= \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}\nabla f(x^{(k)})^T(x^{(k)} - x^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 2: Plug-in convexity $\nabla f(x^{(k)})^T(x^{(k)} - x^*) \geq f(x^{(k)}) - f^*$

$$\|x^{(k+1)} - x^*\|^2 \leq \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}(f(x^{(k)}) - f^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Proof Sketch: Lipschitz Case

Step 1: Track distance to optimum

$$\|x^{(k+1)} - x^*\|^2 = \|x^{(k)} - \eta^{(k)}\nabla f(x^{(k)}) - x^*\|^2 \qquad \text{expand norm} \|a-b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2$$

$$= \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}\nabla f(x^{(k)})^T(x^{(k)} - x^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 2: Plug-in convexity $\quad \nabla f(x^{(k)})^T(x^{(k)} - x^*) \geq f(x^{(k)}) - f^*$

$$\|x^{(k+1)} - x^*\|^2 \leq \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}(f(x^{(k)}) - f^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 3: Telescope and rearrange (using $\|\nabla f\| \leq G$, $\|x^{(0)} - x^*\| = R$)

$$f_k^{\text{best}} - f^* \leq \frac{R^2 + G^2\sum_{i=0}^{k-1}(\eta^{(i)})^2}{2\sum_{i=0}^{k-1}\eta^{(i)}}$$

Proof Sketch: Lipschitz Case

Step 1: Track distance to optimum

$$\|x^{(k+1)} - x^*\|^2 = \|x^{(k)} - \eta^{(k)}\nabla f(x^{(k)}) - x^*\|^2 \qquad \text{expand norm} \|a-b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2$$

$$= \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}\nabla f(x^{(k)})^T(x^{(k)} - x^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 2: Plug-in convexity $\nabla f(x^{(k)})^T(x^{(k)} - x^*) \geq f(x^{(k)}) - f^*$

$$\|x^{(k+1)} - x^*\|^2 \leq \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}(f(x^{(k)}) - f^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 3: Telescope and rearrange (using $\|\nabla f\| \leq G$, $\|x^{(0)} - x^*\| = R$)

$$f_k^{\text{best}} - f^* \leq \frac{R^2 + G^2\sum_{i=0}^{k-1}(\eta^{(i)})^2}{2\sum_{i=0}^{k-1}\eta^{(i)}}$$

Step 4: Optimize step size — for constant η:

$$f_k^{\text{best}} - f^* \leq \underbrace{\frac{R^2}{2k\eta}}_{\downarrow \text{ as } \eta\uparrow} + \underbrace{\frac{\eta G^2}{2}}_{\downarrow \text{ as } \eta\downarrow}$$

Proof Sketch: Lipschitz Case

Step 1: Track distance to optimum

$$\|x^{(k+1)} - x^*\|^2 = \|x^{(k)} - \eta^{(k)}\nabla f(x^{(k)}) - x^*\|^2 \qquad \text{expand norm} \|a-b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2$$

$$= \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}\nabla f(x^{(k)})^T(x^{(k)} - x^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 2: Plug-in convexity $\nabla f(x^{(k)})^T(x^{(k)} - x^*) \geq f(x^{(k)}) - f^*$

$$\|x^{(k+1)} - x^*\|^2 \leq \|x^{(k)} - x^*\|^2 - 2\eta^{(k)}(f(x^{(k)}) - f^*) + (\eta^{(k)})^2\|\nabla f(x^{(k)})\|^2$$

Step 3: Telescope and rearrange (using $\|\nabla f\| \leq G$, $\|x^{(0)} - x^*\| = R$)

$$f_k^{\text{best}} - f^* \leq \frac{R^2 + G^2\sum_{i=0}^{k-1}(\eta^{(i)})^2}{2\sum_{i=0}^{k-1}\eta^{(i)}}$$

Step 4: Optimize step size — for constant η:

$$f_k^{\text{best}} - f^* \leq \underbrace{\frac{R^2}{2k\eta}}_{\downarrow \text{ as } \eta\uparrow} + \underbrace{\frac{\eta G^2}{2}}_{\downarrow \text{ as } \eta\downarrow}$$

Setting $\frac{d}{d\eta}(\cdot) = 0$: $\quad \eta^* = \frac{R}{G\sqrt{k}} \quad \Rightarrow \quad \boxed{f_k^{\text{best}} - f^* \leq \frac{RG}{\sqrt{k}}}$

amazon

Summary: Gradient Descent Convergence Rates

Assumptions	**Error after k**	**Iters for ϵ**	**Step size**
Convex, G-Lipschitz	$O(1/\sqrt{k})$	$O(1/\epsilon^2)$	$\eta = R/(G\sqrt{k})$
Convex, M-smooth	$O(1/k)$	$O(1/\epsilon)$	$\eta = 1/M$
m-SC, M-smooth	$O((1-1/\kappa)^k)$	$O(\kappa \log(1/\epsilon))$	$\eta = 1/M$

Example derivation If $\epsilon = \frac{C}{\sqrt{k}}$, then the number of iterations we need are: $k = \frac{C^2}{\epsilon^2}$

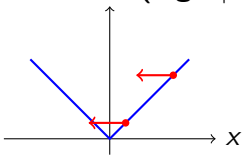
Interpreting the rates:

- $O(1/\sqrt{k})$: Slow – halving error requires $4\times$ iterations
- $O(1/k)$: Sublinear – halving error requires $2\times$ iterations
- $O(\rho^k)$: Linear – constant factor reduction per iteration

Why Smoothness Helps

Key insight: For smooth functions, the gradient "self-tunes"

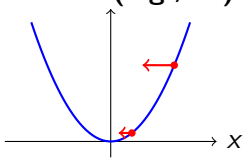
Non-smooth (e.g., $|x|$)



Gradient $= \pm 1$ always

Must use diminishing step size

Smooth (e.g., x^2)



Gradient $\to 0$ near x^*

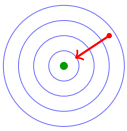
Fixed step size $\eta = 1/M$ works

From smoothness: $f(x^{(k+1)}) \leq f(x^{(k)}) - \frac{1}{2M}\|\nabla f(x^{(k)})\|^2$

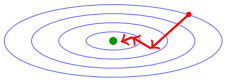
The gradient magnitude shrinks automatically as we approach the minimum!

Dependence on Condition Number

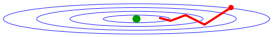
Condition number: $\kappa = M/m \geq 1$



$\kappa = 1$
circular, fast



$\kappa = 10$
elliptical, slower



$\kappa = 100$
narrow valley, zigzag

Convergence rate: $(1 - 1/\kappa)^k$

- $\kappa = 2$: error halves every ≈ 1.4 iterations
- $\kappa = 100$: error halves every ≈ 69 iterations
- $\kappa = 10000$: error halves every ≈ 6931 iterations

Ill-conditioned problems are hard for gradient descent!

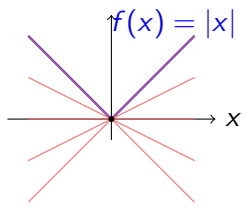
Subgradients for Non-Smooth Functions

Problem: f is convex but not differentiable everywhere

Definition: g is a **subgradient** of f at x if for all y:

$$f(y) \geq f(x) + g^T(y - x)$$

The **subdifferential** $\partial f(x)$ is the set of all subgradients at x



$\partial f(0) = [-1, 1]$

Examples:

- $f(x) = |x|$: $\partial f(0) = [-1, 1]$
- $f(x) = \|x\|_1$:
$$(\partial f(x))_i = \begin{cases} \text{sign}(x_i) & x_i \neq 0 \\ [-1, 1] & x_i = 0 \end{cases}$$
- $f(x) = \max_i(a_i^T x + b_i)$:

$$\partial f(x) = \text{conv}\{a_i : i \in I(x)\}$$

Optimality condition: $0 \in \partial f(x^*)$

where $I(x) = \{i : a_i^T x + b_i = f(x)\}$ (active set)

Subgradient Method

Algorithm:

$$x^{(k+1)} = x^{(k)} - \eta^{(k)} g^{(k)}, \quad \text{where } g^{(k)} \in \partial f(x^{(k)})$$

Warning

Subgradient direction is **not** necessarily a descent direction!
Function value may increase: $f(x^{(k+1)}) > f(x^{(k)})$
$\Rightarrow$ Track best iterate: $f_k^{\text{best}} = \min_{i \le k} f(x^{(i)})$

Theorem (Convergence)

With $\|g\| \le G$ for all $g \in \partial f(x)$ and step size $\eta^{(k)} = R/(G\sqrt{k})$:

$$f_k^{\text{best}} - f^* \le \frac{RG}{\sqrt{k}} \qquad \text{(convergence in } O(1/\epsilon^2) \text{ steps)}$$

Same rate as Lipschitz gradient descent – but cannot do better for non-smooth!

Lower Bounds: Fundamental Limits

Question: How fast can *any* first-order method converge?

Theorem (Nesterov's lower bounds)

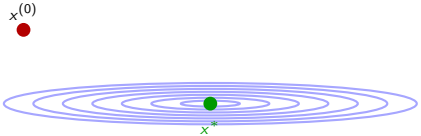
Using only $f(x)$ and $\nabla f(x)$ queries, to achieve $f(x^{(k)}) - f^* \leq \epsilon$:

Setting	**Lower bound**	**GD achieves**
G-Lipschitz convex	$\Omega(1/\epsilon^2)$	$O(1/\epsilon^2)$ ✓
M-smooth convex	$\Omega(1/\sqrt{\epsilon})$	$O(1/\epsilon)$ ✗
m-SC, M-smooth	$\Omega(\sqrt{\kappa}\log(1/\epsilon))$	$O(\kappa\log(1/\epsilon))$ ✗

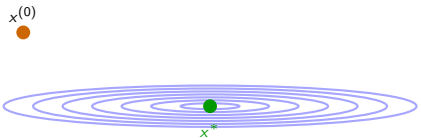
Key insight: GD is optimal for non-smooth, but suboptimal for smooth!

Can we close these gaps? **Yes!** $\rightarrow$ Accelerated methods

The Problem with Gradient Descent

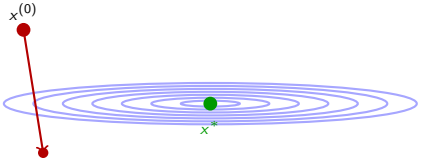


Gradient Descent

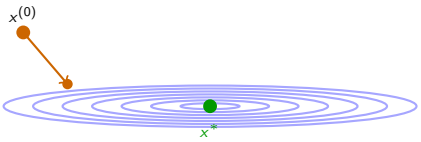


Momentum Method

The Problem with Gradient Descent

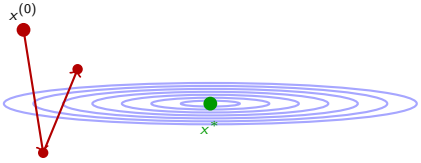


Gradient Descent

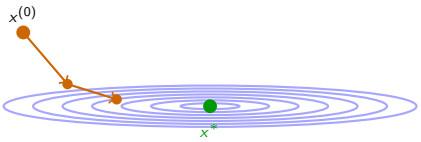


Momentum Method

The Problem with Gradient Descent

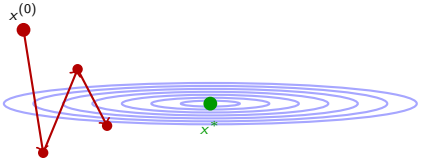


Gradient Descent

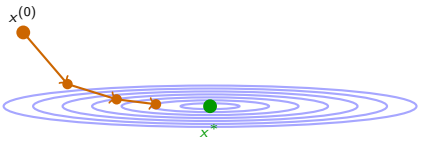


Momentum Method

The Problem with Gradient Descent

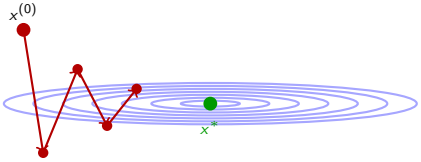


Gradient Descent

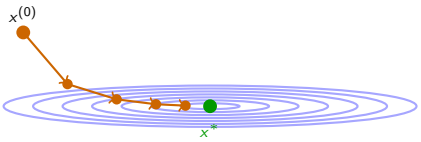


Momentum Method

The Problem with Gradient Descent

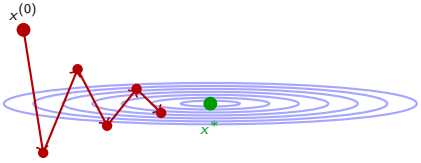


Gradient Descent

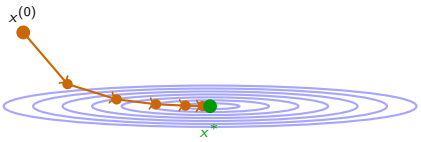


Momentum Method

The Problem with Gradient Descent

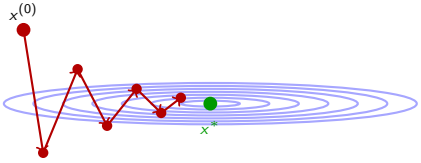


Gradient Descent

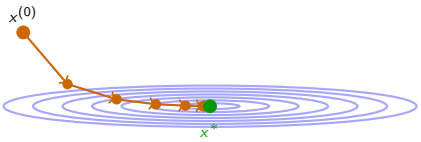


Momentum Method

The Problem with Gradient Descent

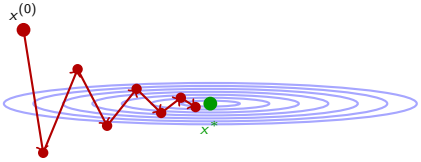


Gradient Descent

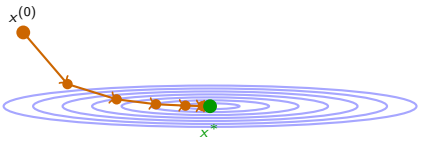


Momentum Method

The Problem with Gradient Descent

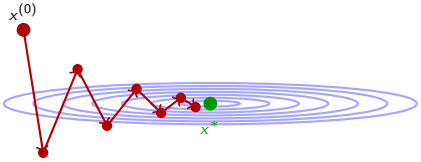


Gradient Descent



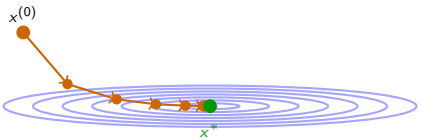
Momentum Method

The Problem with Gradient Descent



Gradient Descent

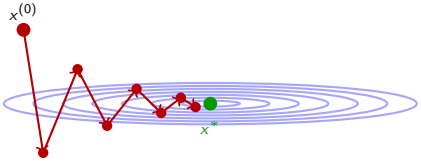
Gradient points to nearest contour, not toward optimum $\Rightarrow$ **zigzag**



Momentum Method

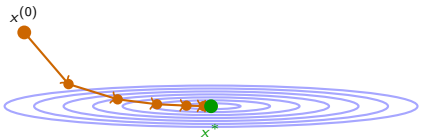
Momentum averages directions, cancels oscillations $\Rightarrow$ **smooth**

The Problem with Gradient Descent



Gradient Descent

Gradient points to nearest contour, not toward optimum $\Rightarrow$ **zigzag**



Momentum Method

Momentum averages directions, cancels oscillations $\Rightarrow$ **smooth**

Key insight: Momentum builds speed along consistent directions, dampens oscillations

Heavy Ball Method (Polyak, 1964)

$$\boxed{x^{(k+1)} = x^{(k)} - \eta \nabla f(x^{(k)}) + \beta(x^{(k)} - x^{(k-1)})}$$

new gradient step momentum term (previous)

Intuition: A heavy ball rolling downhill

- Without momentum ($\beta = 0$): gets bounced in valleys
- With momentum ($\beta > 0$): accumulates speed along consistent directions, cancels oscillations across inconsistent ones

Optimal parameters (for quadratics with eigenvalues in $[m, M]$):

$$\eta^* = \frac{4}{(\sqrt{M} + \sqrt{m})^2}, \quad \beta^* = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2$$

Nesterov's Accelerated Gradient (1983)

Key innovation: Evaluate gradient at *lookahead* point

$$\begin{aligned} x^{(k+1)} &= y^{(k)} - \eta \nabla f(y^{(k)}) && \text{(gradient step from lookahead)} \\ y^{(k+1)} &= x^{(k+1)} + \frac{k}{k+3}(x^{(k+1)} - x^{(k)}) && \text{(momentum/extrapolation)} \end{aligned}$$

Momentum coefficient: $\frac{k}{k+3}$ grows from 0 toward 1

- Early: rely on gradient information
- Later: trust momentum built from history

Theorem (Optimal rate for smooth convex)

With $\eta = 1/M$:

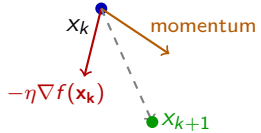
$$f(x^{(k)}) - f^* \leq \frac{2M\|x^{(0)} - x^*\|^2}{(k+1)^2} \qquad \text{(convergence in } O(1/\sqrt{\epsilon}) \text{ steps)}$$

amazon

From Heavy Ball to Nesterov's Accelerated Gradient

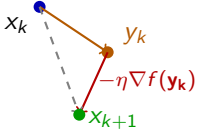
Key idea: Evaluate gradient at lookahead point instead of current point

Heavy Ball



$$x_{k+1} = x_k - \eta\nabla f(\mathbf{x_k}) + \beta(x_k - x_{k-1})$$

Nesterov



$$x_{k+1} = x_k - \eta\nabla f(\mathbf{y_k}) + \beta_k(x_k - x_{k-1})$$
where $y_k = x_k + \beta_k(x_k - x_{k-1})$

	Heavy Ball	**Nesterov**
Gradient at	current x_k	lookahead y_k
Momentum β	fixed (needs tuning)	$\frac{k-1}{k+2} \to 1$
Smooth + strongly cvx	$O\left(\left(1 - \sqrt{m/M}\right)^k\right)$	$O\left(\left(1 - \sqrt{m/M}\right)^k\right)$
Smooth only	may diverge	$O\left(\frac{1}{k^2}\right)$

Acceleration for Strongly Convex Functions

For m-strongly convex, M-smooth functions:

Nesterov variant: Use constant momentum $\beta = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$

$$f(x^{(k)}) - f^* \leq \left(1 - \frac{1}{\sqrt{\kappa}}\right)^k (f(x^{(0)}) - f^*)$$

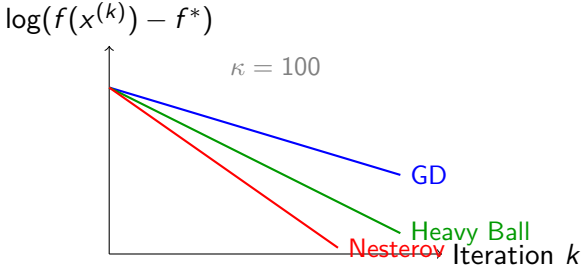
Comparison of rates:

Method	**Rate per iteration**	**Iters for ϵ**
Gradient Descent	$1 - \frac{1}{\kappa}$	$O(\kappa \log(1/\epsilon))$
Nesterov	$1 - \frac{1}{\sqrt{\kappa}}$	$O(\sqrt{\kappa} \log(1/\epsilon))$

Example: $\kappa = 10000$

- GD: $\approx 10000 \times \log(1/\epsilon)$ iterations
- Nesterov: $\approx 100 \times \log(1/\epsilon)$ iterations – 100× speedup!

Comparing Momentum Methods



Method	Step size	Momentum	Rate
Gradient Descent	$1/M$	–	$(1 - 1/\kappa)^k$
Heavy Ball	$4/(\sqrt{M} + \sqrt{m})^2$	$(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^2$	$(1 - 1/\sqrt{\kappa})^k$
Nesterov	$1/M$	$\frac{k}{k+3}$ or fixed	$(1 - 1/\sqrt{\kappa})^k$

Source: Nice illustrations of momentum methods for machine learning

Summary: First-Order Methods

Method	Assumptions	Rate	Optimal?
Subgradient	G-Lipschitz	$O(1/\sqrt{k})$	✓
GD	G-Lipschitz	$O(1/\sqrt{k})$	✓
GD	M-smooth	$O(1/k)$	✗
GD	m-SC, M-smooth	$O((1-1/\kappa)^k)$	✗
Heavy Ball	m-SC, M-smooth	$O((1-1/\sqrt{\kappa})^k)$	✓
Nesterov	M-smooth	$O(1/k^2)$	✓
Nesterov	m-SC, M-smooth	$O((1-1/\sqrt{\kappa})^k)$	✓

Practical guidance:

- Non-smooth $\rightarrow$ Subgradient with diminishing step sizes
- Smooth $\rightarrow$ Nesterov (if you know M) or GD with backtracking
- Strongly convex + smooth $\rightarrow$ Heavy Ball or Nesterov variant

Practical Tips

Choosing a method:

1. Know your function properties (G, M, m if possible)
2. Start with GD + backtracking (robust baseline)
3. Try momentum if convergence is slow
4. Use Nesterov for smooth problems when M is known

Debugging convergence issues:

- Diverging? $\rightarrow$ Reduce step size
- Oscillating? $\rightarrow$ Reduce step size or add momentum
- Converging too slowly? $\rightarrow$ Increase step size or use acceleration
- Stuck at suboptimal point? $\rightarrow$ Check convexity!

Step size heuristics:

- Start with $\eta = 1$ and backtrack
- For quadratics: $\eta = 1/\lambda_{\max}(Q)$
- For neural networks: typical range 10^{-4} to 10^{-1}

References

[1] S. Boyd and L. Vandenberghe, "Convex Optimization", https://web.stanford.edu/~boyd/cvxbook/

[2] Y. Nesterov, "Introductory Lectures on Convex Optimization: A Basic Course", Springer, 2004

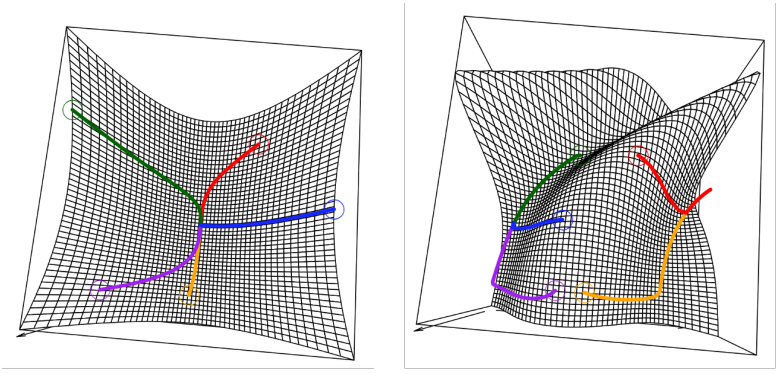
[3] B. Polyak, "Some methods of speeding up the convergence of iteration methods", USSR Computational Mathematics and Mathematical Physics, 1964

[4] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$", Soviet Mathematics Doklady, 1983

[5] A. Beck and M. Teboulle, "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems", SIAM J. Imaging Sciences, 2009

Convex vs Non-Convex Optimization

Convex | **Non-Convex**



GD on non-convex: converges to **stationary point** ($\nabla f = 0$), but no guarantee it's global optimum. Active research area in deep learning!

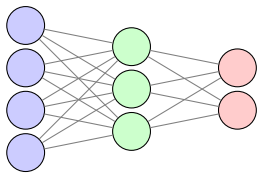
Part III

Training Neural Networks

Why This Matters

The Deep Learning Revolution:

- Image recognition
- Natural language processing
- Game playing (AlphaGo)
- Protein folding
- Large Language Models



At its core:

Training a neural network =
Solving an optimization problem

Find weights that minimize prediction error

Empirical Risk Minimization

Given n data points (a_i, b_i):

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_x(a_i), b_i)$$

- h_x = model with parameters x
- ℓ = loss function (e.g., squared error, cross-entropy)
- n = number of training samples

The Scalability Problem

Full gradient descent:

$$x^{(k+1)} = x^{(k)} - \eta \nabla f(x^{(k)}) = x^{(k)} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x^{(k)})$$

Cost per iteration: $O(n)$ gradient evaluations

Dataset	**Size** n
MNIST	60,000
ImageNet	1,200,000
Common Crawl	100,000,000,000+

Computing the full gradient once can take hours or days!

The Key Idea: Stochastic Gradients

Instead of computing the full gradient:

$$\nabla f(x) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) \quad \text{(expensive: } O(n)\text{)}$$

Use a random sample as an estimate:

$$\nabla f_{i_k}(x) \quad \text{where } i_k \sim \text{Uniform}\{1, \ldots, n\} \quad \text{(cheap: } O(1)\text{)}$$

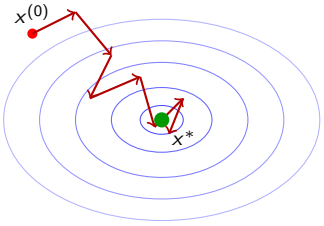
Why does this work?

$$\mathbb{E}_{i_k}[\nabla f_{i_k}(x)] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x)$$

Unbiased estimate! On average, we point in the right direction.

Stochastic Gradient Descent

1. **Input:** Initial $x^{(0)}$, step sizes $\{\eta^{(k)}\}$, iterations K
2. **For** $k = 0, 1, \ldots, K-1$:
   - Sample i_k uniformly from $\{1, \ldots, n\}$
   - $x^{(k+1)} \leftarrow x^{(k)} - \eta^{(k)} \nabla f_{i_k}(x^{(k)})$
3. **Return** average $\bar{x}^{(K)} = \frac{1}{K} \sum_{k=0}^{K-1} x^{(k)}$



Key properties:

- Cost per iteration: $O(1)$ instead of $O(n)$
- Noisy updates, but correct *on average*

Why return the average?

- In theory: full average; in practice: last few iterates

Mini-Batch SGD

Idea: Use a small batch of b samples instead of just one:

$$x^{(k+1)} = x^{(k)} - \eta^{(k)} \cdot \frac{1}{b} \sum_{j \in B_k} \nabla f_j(x^{(k)})$$

where $B_k \subset \{1, \ldots, n\}$ is a random subset of size b.

Why mini-batches?

Variance Reduction	**Parallelism**	**Hardware**
Averaging b samples reduces variance by factor b	b gradients computed in parallel on GPU	Modern GPUs optimized for batch operations

Typical batch sizes: 32, 64, 128, 256, 512

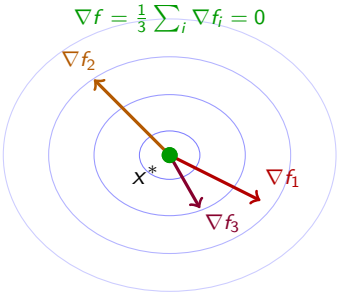
Convergence Analysis: The Variance Challenge

Key insight: At x^*, the full gradient vanishes but individual gradients do not!

Full gradient is the average:

$$\nabla f(x^*) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^*) = 0$$

But each $\nabla f_i(x^*) \neq 0$ in general!



Individual gradients cancel on average, but SGD sees only one at a time

With constant step η: oscillates in neighborhood of size $\sim \eta\sigma$. To converge $\Rightarrow$ **diminishing step sizes**

Convergence of SGD

Theorem. Let $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ with each f_i convex and M-smooth, and $\sigma^2 < \infty$.

***M*-smooth case:** With constant step size η:

$$\mathbb{E}[f(\bar{x}^{(K)})] - f^* \leq \underbrace{\frac{R^2}{\eta K}}_{\text{optimization}} + \underbrace{\eta\sigma^2}_{\text{noise}}$$

- If $\sigma^2 = 0$: set $\eta = \frac{1}{M} \Rightarrow O\left(\frac{MR^2}{K}\right)$ (recovers GD rate $O(1/K)$)
- If $\sigma^2 > 0$: set $\eta = \frac{R}{\sigma\sqrt{K}} \Rightarrow O\left(\frac{R\sigma}{\sqrt{K}}\right)$ (noise dominates)

***m*-strongly convex case:** With step size $\eta^{(k)} = \frac{2}{m(k+1)}$:

$$\mathbb{E}[f(\bar{x}^{(K)})] - f^* = O\left(\frac{\sigma^2}{mK}\right) \qquad \text{GD had rate: } O\left(\left(1 - \frac{m}{M}\right)^K\right)$$

Key insight: Noise σ^2 forces smaller step sizes, which slows convergence from $O(1/K)$ to $O(1/\sqrt{K})$.

SGD vs Full Gradient Descent (Convex + Smooth)

	Full GD	**SGD**	**Winner**
Cost per iteration	$O(n)$	$O(1)$	SGD
Iterations for ϵ accuracy	$O(1/\epsilon)$	$O(1/\epsilon^2)$	GD
Total cost for ϵ accuracy	$O(n/\epsilon)$	$O(1/\epsilon^2)$	**Depends!**

When does SGD win?

SGD is faster when: $\frac{1}{\epsilon^2} < \frac{n}{\epsilon}$, i.e., when $\boxed{n > \frac{1}{\epsilon}}$

Translation: SGD wins when we have many samples and don't need extreme precision.

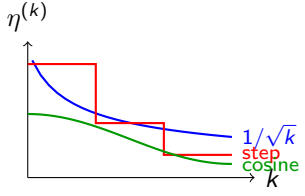
This is exactly the regime of modern machine learning!

Practical Considerations: Step Size Schedules

The step size $\eta^{(k)}$ is critical for SGD performance.

Common schedules:

- $1/\sqrt{k}$ **decay:** $\eta^{(k)} = \frac{\eta_0}{\sqrt{k+1}}$ (theoretically motivated)
- $1/k$ **decay:** $\eta^{(k)} = \frac{\eta_0}{k+1}$ (for strongly convex)
- **Step decay:** Reduce by factor 10 at fixed epochs (common in deep learning)
- **Cosine annealing:** $\eta^{(k)} = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min})(1 + \cos(\pi k/K))$

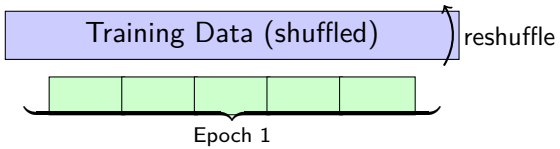


Epochs and Shuffling

Epoch: One complete pass through all n training samples

In practice:

1. Shuffle the data at the start of each epoch
2. Process mini-batches sequentially (not random sampling with replacement)
3. Repeat for multiple epochs



Why shuffle?

- Ensures batches are approximately independent
- Reduces variance compared to fixed ordering
- Required for theoretical guarantees

SGD with Momentum

Problem: SGD gradients are noisy $\Rightarrow$ oscillates, slow convergence
Solution: Use a smoothed gradient via **exponential moving average (EMA)**

$$m^{(k+1)} = \beta\, m^{(k)} + (1-\beta)\nabla f_{i_k}(x^{(k)}), \qquad x^{(k+1)} = x^{(k)} - \eta\, m^{(k+1)}$$

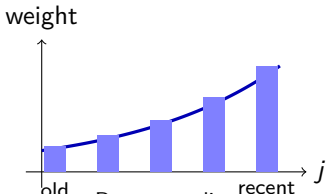
What does EMA compute? Unrolling the recursion:

$$m^{(k+1)} = (1-\beta)\sum_{j=0}^{k} \beta^{k-j}\nabla f_{i_j}(x^{(j)})$$

Weighted average of past gradients:

- Recent gradients: high weight
- Old gradients: exponentially forgotten
- Window $\approx \frac{1}{1-\beta}$ steps

$\beta = 0.9 \Rightarrow$ averages over ≈ 10 steps
$\beta = 0.99 \Rightarrow$ averages over ≈ 100 steps



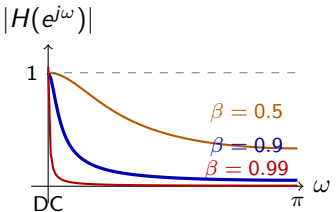
EMA is a Low-Pass Filter

Recall: $m^{(k+1)} = \beta\, m^{(k)} + (1-\beta)\, \nabla f_{i_j}(x^{(j)})$

This is a **first-order IIR filter** with transfer function:

$$H(z) = \frac{1-\beta}{1-\beta z^{-1}}$$

Frequency response:



Interpretation for SGD:

Signal	SGD meaning
Input	noisy gradient $g^{(k)}$
Output	momentum $m^{(k)}$
Low freq	true gradient
High freq	noise

Trade-off: large $\beta \Rightarrow$ filters out noise but adapts slowly. small $\beta \Rightarrow$ fast but noisy.

Typical $\beta = 0.9$: good balance for SGD momentum

Adam: Adaptive Moment Estimation

Idea: Momentum + **per-parameter adaptive learning rate**

$$m^{(k+1)} = \beta_1 m^{(k)} + (1-\beta_1)\, g \qquad \text{(EMA of gradient)}$$

$$v^{(k+1)} = \beta_2 v^{(k)} + (1-\beta_2)\, g \odot g \qquad \text{(EMA of squared gradient)}$$

$$x^{(k+1)} = x^{(k)} - \eta \cdot \hat{m}^{(k+1)} / (\sqrt{\hat{v}^{(k+1)}} + \epsilon) \qquad \text{(adaptive update)}$$

where $g \odot g = \left[\left(\frac{\partial f}{\partial x_1}\right)^2, \ldots, \left(\frac{\partial f}{\partial x_n}\right)^2 \right]^\top$ tracks one magnitude per parameter

Bias correction: Since $m^{(0)} = v^{(0)} = 0$, early estimates are biased toward zero:

$$\hat{m}^{(k)} = \frac{m^{(k)}}{1-\beta_1^k}, \qquad \hat{v}^{(k)} = \frac{v^{(k)}}{1-\beta_2^k}$$

Per-parameter adaptation: Each x_j has effective learning rate $\frac{\eta}{\sqrt{\hat{v}_j}+\epsilon}$

Large gradients $\Rightarrow$ large $\hat{v}_j \Rightarrow$ smaller step. Defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.001$

Kingma and Ba, "Adam: A Method for Stochastic Optimization", ICLR 2015. https://arxiv.org/abs/1412.6980

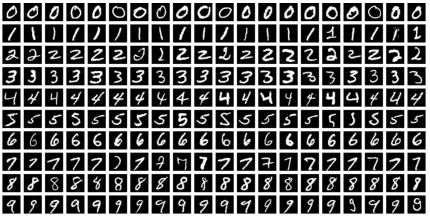
Problem Setup: MNIST Classification

The Dataset:

- 70,000 grayscale images
- Handwritten digits (0–9)
- Each image: 28×28 pixels
- Task: predict which digit
- https://en.wikipedia.org/wiki/MNIST_database

Why MNIST?

- Small enough to train on laptop
- Rich enough to illustrate key ideas
- The "Hello World" of ML



Sample digits from MNIST

$28 \times 28 = 784$ pixels per image

Mathematical Formulation

Input and Output:

- Input: $a \in \mathbb{R}^{784}$ (flattened image)
- Label: $y \in \{0, 1, \ldots, 9\}$ (true digit)
- Prediction: $\hat{p} \in \mathbb{R}^{10}$ (probability distribution over classes)

Goal: Learn a function $h_\theta : \mathbb{R}^{784} \to \mathbb{R}^{10}$ such that:

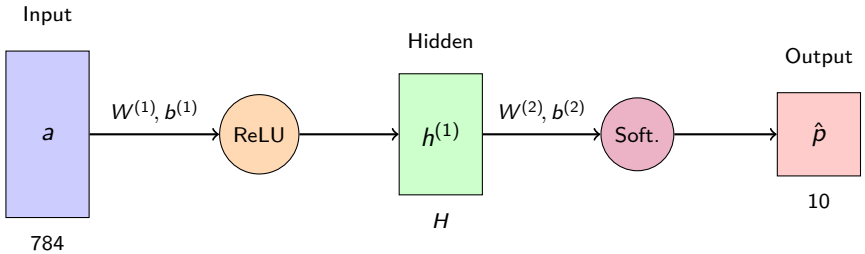
$$h_\theta(a) \approx \text{probability distribution peaking at true class } y$$

Key Questions:

1. What form should h_θ take? $\longrightarrow$ **Network Architecture**
2. How do we measure "good"? $\longrightarrow$ **Loss Function**
3. How do we find θ? $\longrightarrow$ **Optimization (SGD)**

The Multilayer Perceptron (MLP)

Core idea: Stack simple building blocks to create complex functions



Forward computation:

$$z^{(1)} = W^{(1)}a + b^{(1)} \quad \text{(linear transformation)}$$

$$h^{(1)} = \text{ReLU}(z^{(1)}) \quad \text{(nonlinear activation)}$$

$$z^{(2)} = W^{(2)}h^{(1)} + b^{(2)} \quad \text{(linear transformation)}$$

$$\hat{p} = \text{softmax}(z^{(2)}) \quad \text{(probabilities)}$$

amazon

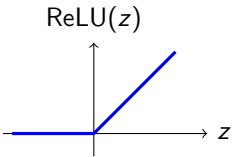
Network Components

Parameters to learn: $\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$

- $W^{(1)} \in \mathbb{R}^{H \times 784}$, $b^{(1)} \in \mathbb{R}^{H}$ first layer
- $W^{(2)} \in \mathbb{R}^{10 \times H}$, $b^{(2)} \in \mathbb{R}^{10}$ second layer

ReLU Activation:

$$\text{ReLU}(z) = \max(0, z)$$



Softmax:

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}$$

Converts scores to probabilities:

- All outputs in $(0, 1)$
- Sum to exactly 1

Example: $H = 128$ hidden units $\Rightarrow 784 \times 128 + 128 + 128 \times 10 + 10 = \mathbf{101{,}770}$ parameters

Loss Function for Multi-class Classification: Cross-Entropy

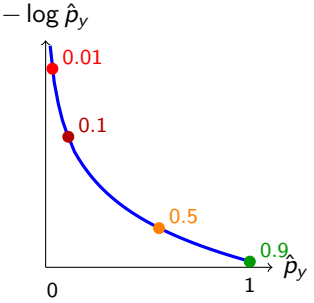
For true label y and predicted probabilities $\hat{p}$:

$$\ell(\hat{p}, y) = -\log \hat{p}_y = -\log \frac{e^{z_y^{(2)}}}{\sum_{k=0}^{9} e^{z_k^{(2)}}}$$

Example: Image shows a "3", so true label $y = 3$

Prediction	$\hat{p}_3$	**Loss**
Confident & correct	0.9	$-\log(0.9) = 0.1$
Uncertain	0.5	$-\log(0.5) = 0.7$
Unsure	0.1	$-\log(0.1) = 2.3$
Confident & wrong	0.01	$-\log(0.01) = 4.6$

Key property:
Confident mistakes are **severely** penalized!



The Optimization Problem

Given n training examples $\{(a_i, y_i)\}_{i=1}^n$, find weights that minimize:

$$\boxed{\underset{\theta}{\text{minimize}} \quad f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(h_\theta(a_i), y_i)}$$

where $\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$ collects all network parameters.

Two key observations:

1. This is **Empirical Risk Minimization** $\Longrightarrow$ **SGD** is the natural algorithm
2. The problem is **non-convex** due to nonlinear activations
   $\Longrightarrow$ No guarantee of global minimum, but SGD works well in practice

Backpropagation

Challenge: To apply SGD, we need to compute $\nabla_\theta \ell$ for each sample

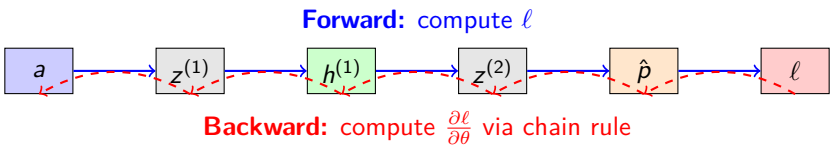
Chain rule: For composed functions $\ell = \ell(g(h(x)))$:

$$\frac{d\ell}{dx} = \frac{d\ell}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}$$

Our network is exactly this: a composition of simple functions

$$\ell = \ell\Big(\text{softmax}(W^{(2)} \cdot \text{ReLU}(W^{(1)} a + b^{(1)}) + b^{(2)}),\ y\Big)$$

Backpropagation: Systematically apply chain rule layer by layer starting from the end



Forward Pass: Compute and Store

Starting from input, compute all intermediate values:

1. **Input layer:** a (given)
2. **Hidden pre-activation:** $z^{(1)} = W^{(1)}a + b^{(1)}$
3. **Hidden activation:** $h^{(1)} = \text{ReLU}(z^{(1)})$
4. **Output pre-activation:** $z^{(2)} = W^{(2)}h^{(1)} + b^{(2)}$
5. **Output probabilities:** $\hat{p} = \text{softmax}(z^{(2)})$
6. **Loss:** $\ell = -\log \hat{p}_y$

Important: Store all intermediate values!
We need them for the backward pass.

Backward Pass

Goal: Compute all partial derivatives $\frac{\partial \ell}{\partial W^{(1)}}$, $\frac{\partial \ell}{\partial b^{(1)}}$, $\frac{\partial \ell}{\partial W^{(2)}}$, $\frac{\partial \ell}{\partial b^{(2)}}$ using chain rule

Step 1: Gradient w.r.t. output scores $z^{(2)}$
For cross-entropy + softmax, there's a clean formula:

$$\frac{\partial \ell}{\partial z^{(2)}} = \hat{p} - e_y$$

where e_y is one-hot encoding of true label y

Step 2: Gradients for $W^{(2)}$ and $b^{(2)}$ (chain rule)

$$\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} (h^{(1)})^\top, \qquad \frac{\partial \ell}{\partial b^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}}$$

Backward Pass: Hidden Layer

Step 3: Gradient w.r.t. hidden activations $h^{(1)}$

$$\frac{\partial \ell}{\partial h^{(1)}} = (W^{(2)})^\top \frac{\partial \ell}{\partial z^{(2)}}$$

Step 4: Gradient w.r.t. hidden pre-activations $z^{(1)}$

Apply chain rule through ReLU:

$$\frac{\partial \ell}{\partial z^{(1)}} = \frac{\partial \ell}{\partial h^{(1)}} \odot \underbrace{\mathbf{1}[z^{(1)} > 0]}_{\text{ReLU derivative}}$$

($\odot$ = elementwise multiplication; ReLU derivative is 1 where $z^{(1)} > 0$, else 0)

Step 5: Gradients for $W^{(1)}$ and $b^{(1)}$

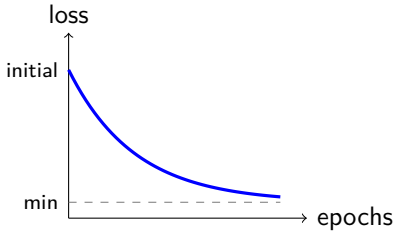
$$\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z^{(1)}} a^\top, \qquad \frac{\partial \ell}{\partial b^{(1)}} = \frac{\partial \ell}{\partial z^{(1)}}$$

Done! We have all four gradients needed for SGD update.

The Complete Training Algorithm

Mini-Batch SGD + Backprop:

1. Initialize weights randomly
2. **For each epoch:**
   - 2.1 Shuffle training data
   - 2.2 **For each mini-batch $\mathcal{B}$:**
     - Forward pass (compute ℓ)
     - Backward pass (compute $\nabla \ell$)
     - Average gradients over batch
     - Update: $\theta \leftarrow \theta - \eta \cdot \nabla$
3. Evaluate accuracy



Hyperparameters:

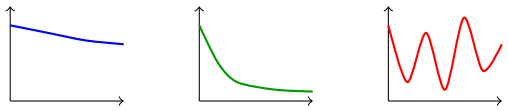
- Learning rate η
- Batch size B
- Number of epochs
- Hidden size H

Practical Considerations

Initialization matters:

- Too large $\Rightarrow$ exploding gradients
- Too small $\Rightarrow$ vanishing gradients
- Common: $W_{ij} \sim \mathcal{N}(0, 1/\sqrt{\text{fan-in}})$

Learning rate is critical:

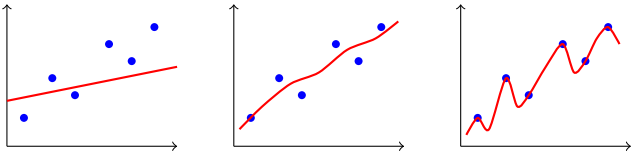


Shuffling: Ensures mini-batches are independent (required for SGD theory)

Regularization: Often add $\frac{\lambda}{2}\|\theta\|^2$ to prevent overfitting

Regularization: Preventing Overfitting

Problem: Network can memorize training data instead of learning patterns



Solution: Add a penalty for large weights

$$\underset{\theta}{\text{minimize}} \quad \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(h_\theta(a_i), y_i)}_{\text{fit the data}} + \underbrace{\frac{\lambda}{2}\|\theta\|^2}_{\text{keep weights small}}$$

Why it works:

- Large weights $\Rightarrow$ sharp, complex decision boundaries $\Rightarrow$ memorization
- Small weights $\Rightarrow$ smooth, simple functions $\Rightarrow$ generalization

amazon

Quiz 1: MNIST Classification Challenge / Submit by 31/03/2026

Task: Build an ML model to classify handwritten digits (0–9)

Model Interface:

- `model.fit(X, y)` – train on images $X \in \mathbb{R}^{n \times 784}$, labels $y \in \{0, ..., 9\}^n$
- `model.predict(X)` – return predicted labels

Rules:

1. Only `numpy` allowed
2. No pre-computed weights – training must happen in `fit()`
3. Time budget: 120s (hard stop: 180s)
4. Minimum accuracy: 97%

Evaluation:

- You develop with `seed=42`
- Grading uses an *unknown* seed $\Rightarrow$ your model must generalize

https://github.com/gpasxos/large-scale-optimization/blob/main/Quiz_2026.ipynb

amazon

Part IV

Duality

Outline

Goal: Solve (or bound) constrained convex optimization

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad g_i(x) \leq 0, \quad h_j(x) = 0$$

- **The Lagrangian** – combining objective and constraints
- **Weak and strong duality** – provide bounds
- **KKT conditions** – necessary and sufficient for optimality
- **Complementary slackness** – active constraints and multipliers
- **Fenchel duality** – conjugates and saddle points

Duality: every constrained problem has a "mirror" problem

Motivating Example: Constrained Minimum

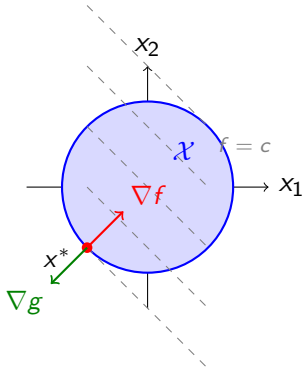
$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 2 \end{aligned}$$

Observations:

- Linear objective $\Rightarrow$ optimum on boundary
- At optimum: gradients are *parallel*
- $\nabla f(x^*) = \lambda \nabla g(x^*)$

Solution:

- $x^* = (-1, -1)$, $\lambda^* = 1/2$
- $f^* = -2$



At optimum: $\nabla f(x^*)$ and $\nabla g(x^*)$ are parallel (true for active constraints)

Why Must Gradients Be Parallel?

Suppose they weren't parallel at x^*...

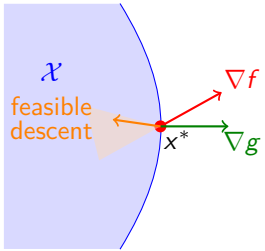
Then $-\nabla f(x^*)$ and $-\nabla g(x^*)$ span a cone of directions that:

- Decrease the objective ($d^T \nabla f < 0$)
- Stay feasible ($d^T \nabla g < 0$)

$\Rightarrow$ We could improve! Contradiction.

At a true minimum:

- Only way to decrease f is to leave feasible region: $\nabla f = \lambda \nabla g$ for some $\lambda \geq 0$



If gradients not parallel:

there exists feasible descent direction

The Lagrangian Function

Definition (Lagrangian)

For problem $\min_x f(x)$ s.t. $g_i(x) \leq 0$, $h_j(x) = 0$:

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x)$$

Dual variables:

- $\lambda_i \geq 0$: multiplier for $g_i(x) \leq 0$
- $\mu_j \in \mathbb{R}$: multiplier for $h_j(x) = 0$

Also called **Lagrange multipliers**

Interpretation:

- λ_i = "price" of violating constraint i
- Large $\lambda_i \Rightarrow$ violation is expensive
- At right λ: minimizing L gives constrained optimum

Stationarity: $\nabla_x L(x^*, \lambda^*, \mu^*) = 0 \Leftrightarrow \nabla f(x^*) + \sum_i \lambda_i^* \nabla g_i(x^*) + \sum_j \mu_j^* \nabla h_j(x^*) = 0$

The Lagrange Dual Function

Definition (Dual function)

$$D(\lambda, \mu) = \inf_x L(x, \lambda, \mu) = \inf_x \left[f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x) \right]$$

Key property:

Concavity

$D(\lambda, \mu)$ is **always concave**
(even if primal is nonconvex!)

Why? Pointwise infimum of affine functions in (λ, μ)

Implication:

- Dual problem is concave maximization
- Always "easy" (in principle)
- Can solve even when primal is hard

Computing D**:** Minimize L over x (often has closed form for structured problems)

Weak Duality: A Universal Lower Bound

Theorem (Weak Duality)

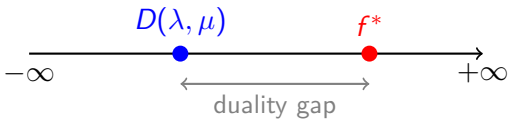
For any $\lambda \geq 0$ and any μ:

$$D(\lambda, \mu) \leq f^*$$

Proof: Let x^* be primal optimal. Then:

$$D(\lambda, \mu) = \inf_x L(x, \lambda, \mu) \leq L(x^*, \lambda, \mu)$$

$$= f(x^*) + \underbrace{\sum_i \lambda_i g_i(x^*)}_{\leq 0 \text{ (since } \lambda_i \geq 0, g_i(x^*) \leq 0)} + \underbrace{\sum_j \mu_j h_j(x^*)}_{=0} \leq f(x^*) = f^*$$



The Dual Problem

Question: What's the *tightest* lower bound?

Definition (Dual Problem)

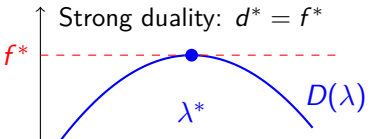
$$d^* = \underset{\lambda,\mu}{\text{maximize}} \; D(\lambda, \mu) \quad \text{subject to} \quad \lambda \geq 0$$

Properties:

- Concave maximization (tractable!)
- $d^* \leq f^*$ always (weak duality)
- $f^* - d^*$ = **duality gap**

Strong duality: $d^* = f^*$

- Gap is zero
- Dual finds the optimal primal value!
- When does this hold?



amazon

When Does Strong Duality Hold?

Definition (Slater's Condition)

A problem satisfies **Slater's condition** if there exists a **strictly feasible** point $\tilde{x}$:

$$g_i(\tilde{x}) < 0 \quad \forall i, \qquad h_j(\tilde{x}) = 0 \quad \forall j$$

Theorem (Strong Duality)

If the primal is convex and Slater's condition holds, then:

$$d^* = f^* \qquad \text{(strong duality)}$$

and the dual optimum is attained.

Slater satisfied:

- $\min\{x_1 + x_2 : x_1^2 + x_2^2 \leq 1\}$
- Take $\tilde{x} = (0, 0)$: $0 < 1$ ✓

Slater fails:

- $\min\{x : x^2 \leq 0\}$
- Only feasible point: $x = 0$

Complementary Slackness

Theorem

If strong duality holds with optimal (x^*, λ^*, μ^*), then:

$$\lambda_i^* g_i(x^*) = 0 \qquad \forall i = 1, \ldots, m$$

Equivalently:

$$\lambda_i^* > 0 \;\Rightarrow\; g_i(x^*) = 0 \qquad \text{and} \qquad g_i(x^*) < 0 \;\Rightarrow\; \lambda_i^* = 0$$

Economic interpretation:

- λ_i^* = "shadow price" of constraint i
- Slack constraint ($g_i < 0$): not binding, price $= 0$
- Tight constraint ($g_i = 0$): may have positive price
- Positive price $\Rightarrow$ constraint must be tight

Example: Plant Crops

Problem: x_1 acres of wheat (\$5k/acre profit) and x_2 acres of corn (\$4k/acre profit)

$$\begin{aligned} \underset{x_1, x_2 \geq 0}{\text{maximize}} \quad & 5x_1 + 4x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 6 \quad \text{(land: 6 acres)} \\ & x_1 \leq 4 \quad \text{(wheat irrigation capacity)} \\ & x_2 \leq 5 \quad \text{(corn irrigation capacity)} \end{aligned}$$

Solution: $x^* = (4, 2)$, profit $= \$28$k

	Land	**Wheat irr.**	**Corn irr.**
Used	6	4	2
Limit	6	4	5
Tight?	Yes	Yes	No
λ_i^*	4	1	0

$\lambda_1^*(6-6) = 0$ ✓ $\lambda_2^*(4-4) = 0$ ✓
$\lambda_3^*\underbrace{(2-5)}_{\neq 0} = 0$ ✓

Shadow prices:

- $\lambda_1^* = 4$: extra acre of land $\Rightarrow$ profit +\$4k
- $\lambda_2^* = 1$: extra wheat irrigation $\Rightarrow$ profit +\$1k
- $\lambda_3^* = 0$: extra corn irrigation $\Rightarrow$ **worthless** (3 acres already unused!)

Advice: Invest in land first (\$4k return per acre), then wheat irrigation (\$1k return).

The KKT Conditions

Definition (Karush-Kuhn-Tucker Conditions)

(x^*, λ^*, μ^*) satisfies the **KKT conditions** if:

1. **Stationarity:** $\nabla f(x^*) + \sum_i \lambda_i^* \nabla g_i(x^*) + \sum_j \mu_j^* \nabla h_j(x^*) = 0$
2. **Complementary slackness:** $\lambda_i^* g_i(x^*) = 0$ for all i
3. **Primal feasibility:** $g_i(x^*) \leq 0, \quad h_j(x^*) = 0$ for all i, j
4. **Dual feasibility:** $\lambda_i^* \geq 0$ for all i

Theorem (KKT for Convex Problems)

For convex problems with strong duality:
x^* is primal optimal, (λ^*, μ^*) is dual optimal $\Leftrightarrow$ KKT conditions hold

Key insight: KKT reduces constrained optimization to solving a system of equations!

Example: Equality constrained QP

Problem: $\min \frac{1}{2}\|x\|^2$ subject to $Ax = b$

Lagrangian: $L(x, \mu) = \frac{1}{2}x^T x + \mu^T(Ax - b)$

KKT conditions:

Stationarity: $x^* + A^T\mu^* = 0 \;\Rightarrow\; x^* = -A^T\mu^*$

Primal feas.: $Ax^* = b$

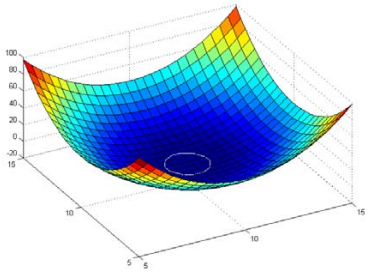
Solve: Substitute into feasibility:

$$A(-A^T\mu^*) = b \;\;\Rightarrow\;\; \mu^* = -(AA^T)^{-1}b$$

Solution:

$$\boxed{x^* = A^T(AA^T)^{-1}b}$$

Optimum is closest point to origin on the affine subspace $Ax = b$



Plot of $\frac{1}{2}\|x\|^2$ for $d = 2$.

Application: Power Allocation over Fading Channels

Problem: Allocate power p_i to n channels with noise levels $N_i > 0$

$$\underset{p \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \log(1 + p_i/N_i) \quad \text{s.t.} \quad \sum_{i=1}^{n} p_i = P$$

Lagrangian (with $\lambda_i \geq 0$ for $-p_i \leq 0$, and μ for equality):

$$L(p, \lambda, \mu) = -\sum_{i=1}^{n} \log(1 + p_i/N_i) - \sum_{i=1}^{n} \lambda_i p_i + \mu\Big(\sum_{i=1}^{n} p_i - P\Big)$$

Stationarity: $\dfrac{\partial L}{\partial p_i} = -\dfrac{1}{N_i + p_i} - \lambda_i + \mu = 0 \;\Rightarrow\; \lambda_i = \mu - \dfrac{1}{N_i + p_i}$

Comp. slackness: $\lambda_i \, p_i = 0 \;\; \forall \, i$

Primal feasibility: $p_i \geq 0, \quad \sum_i p_i = P$

Dual feasibility: $\lambda_i \geq 0 \;\; \forall \, i$

Substituting stationarity into complementary slackness: $p_i \left(\mu - \frac{1}{N_i + p_i}\right) = 0, \quad i = 1, \ldots, n$

For each channel: either $p_i = 0$ or $p_i = \frac{1}{\mu} - N_i$. **Q:** *Which one?*

amazon

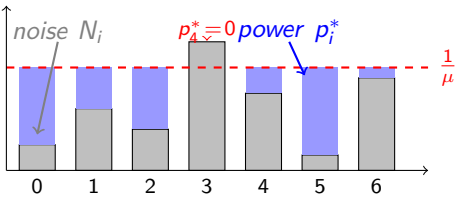
Solution: Water-Filling

Decide $p_i = 0$ or $p_i = \frac{1}{\mu} - N_i$ using KKT conditions:

Case 1: $1/\mu > N_i$ (good channel)
If we tried $p_i = 0$: stationarity gives $\lambda_i = \mu - 1/N_i < 0$ ✗ violates $\lambda_i \geq 0$
$\Rightarrow$ Must have $p_i = \frac{1}{\mu} - N_i > 0$

Case 2: $1/\mu \leq N_i$ (bad channel)
If we tried $p_i > 0$: need $\mu = \frac{1}{N_i + p_i} < \frac{1}{N_i}$
✗ contradicts $1/\mu \leq N_i$
$\Rightarrow$ Must have $p_i = 0$ (channel shut off)

Solution: $\boxed{p_i^* = \left(\frac{1}{\mu} - N_i\right)^+}$ where μ satisfies $\sum_{i=1}^n \left(\frac{1}{\mu} - N_i\right)^+ = P$



Intuition: Pour P water on top of N_i "rocks"

- Water level $1/\mu$ rises uniformly
- Good channels (low N_i) get more power
- Bad channels (high N_i) get **nothing** — better to reallocate!
- μ is the shadow price of power budget

amazon

From Constraints to Min-Max

Goal: Reformulate constrained problem as unconstrained

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0$$

Key observation: What happens when we maximize L over $\lambda \geq 0$?

$$\sup_{\lambda \geq 0} L(x, \lambda) = \sup_{\lambda \geq 0} [f(x) + \lambda g(x)] = \begin{cases} f(x) & \text{if } g(x) \leq 0 \\ +\infty & \text{if } g(x) > 0 \end{cases}$$

Why?

- If $g(x) \leq 0$: maximizing $\lambda g(x)$ over $\lambda \geq 0$ gives $\lambda^* = 0$, so $\sup = f(x)$
- If $g(x) > 0$: take $\lambda \to +\infty$, so $\sup = +\infty$

The sup over λ acts as an **infinite penalty** for constraint violation!

Constrained Optimization = Min-Max Game

Combining the insights:

$$\min_x \{f(x) : g(x) \leq 0\} \quad = \quad \min_x \sup_{\lambda \geq 0} L(x, \lambda)$$

Interpretation as a two-player game:

Player 1 (minimizer):

- Chooses x
- Wants to minimize cost
- Must respect constraints (or pay ∞)

Player 2 (maximizer):

- Chooses $\lambda \geq 0$
- "Adversary" or "referee"
- Penalizes constraint violations

The game:

1. Player 1 announces x
2. Player 2 responds with worst-case λ (infinite penalty if cheating)
3. Player 1's best strategy: choose feasible x that minimizes f

What If We Swap the Order?

Primal: Player 1 moves first, Player 2 responds

$$f^* = \min_x \sup_{\lambda \geq 0} L(x, \lambda)$$

Dual: Player 2 moves first, Player 1 responds

$$d^* = \sup_{\lambda \geq 0} \min_x L(x, \lambda) = \sup_{\lambda \geq 0} D(\lambda)$$

Who has the advantage?

- Moving **second** is always better (you see opponent's move)
- In primal: Player 2 moves second $\Rightarrow$ can punish violations
- In dual: Player 1 moves second $\Rightarrow$ can optimize against fixed λ

Max-Min Inequality or ... Weak Duality (always holds)

$$\underbrace{\sup_{\lambda} \min_x L(x, \lambda)}_{d^* \text{ (dual)}} \leq \underbrace{\min_x \sup_{\lambda} L(x, \lambda)}_{f^* \text{ (primal)}}$$

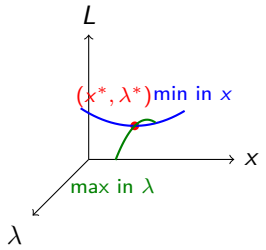
Saddle Points: When Order Doesn't Matter

Question: When does $\min \sup = \sup \min$? **Answer:** When there exists a **saddle point**

Definition (Saddle Point)

(x^*, λ^*) is a saddle point of L if:

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*) \quad \text{for all } x, \lambda \geq 0$$



At a saddle point:

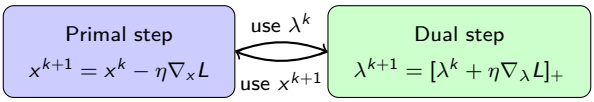
- x^* minimizes $L(\cdot, \lambda^*)$
- λ^* maximizes $L(x^*, \cdot)$
- Neither player wants to deviate
- **Nash equilibrium!**

Saddle point exists $\Leftrightarrow$ strong duality holds $\Leftrightarrow$ $d^* = f^*$

Why Saddle Points Matter for Algorithms

Saddle point formulation: $\min_x \max_{\lambda \geq 0} L(x, \lambda)$

Enables primal-dual algorithms:



Key benefits:

- **Decoupling:** Primal and dual updates can be simple (even closed-form)
- **Certificates:** Gap $f(x^k) - D(\lambda^k)$ measures distance to optimality
- **Parallelization:** Dual often decomposes across constraints

Examples: Uzawa's method, ADMM, Chambolle-Pock, Arrow-Hurwicz...

A Universal Template for Convex Optimization

Observation: Almost any convex problem can be written as:

$$\min_x \quad f(x) + g(Ax)$$

How? Use **indicator functions** $\delta_C(z) = \begin{cases} 0 & z \in C \\ +\infty & z \notin C \end{cases}$

Problem	**As** $\min_x f(x) + g(Ax)$
$\min f(x)$ s.t. $Ax = b$	$f(x) + \delta_{\{b\}}(Ax)$
$\min f(x)$ s.t. $Ax \leq b$	$f(x) + \delta_{\{z : z \leq b\}}(Ax)$
$\min f(x)$ s.t. $x \in C$	$f(x) + \delta_C(x)$ (take $A = I$)
LASSO	$\lambda\|x\|_1 + \frac{1}{2}\|Ax - b\|^2$

Constraints become functions, everything is $f + g \circ A$

Fenchel Duality for $f + g \circ A$

Since $\min_x f(x) + g(Ax)$ covers (almost) everything, let's derive its dual.

Fenchel dual: For any convex g: $g(z) = \max_y \{\langle z, y\rangle - g^*(y)\}$
Our template as a saddle-point:

$$\min_x \{f(x) + g(Ax)\} = \min_x \left\{ f(x) + \max_y [\langle Ax, y\rangle - g^*(y)] \right\}$$
$$= \min_x \max_y \{f(x) + \langle Ax, y\rangle - g^*(y)\}$$

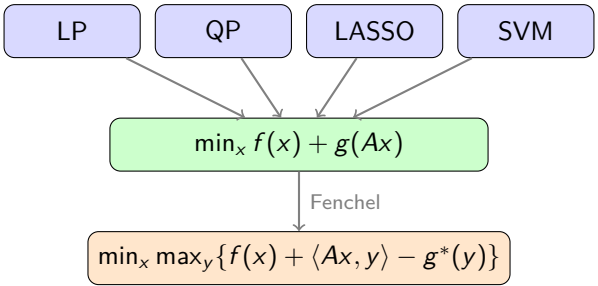
Swapping min and max gives the Fenchel dual:

$$\max_y \left\{ \underbrace{-f^*(-A^T y)}_{-\max_x[\langle -A^T y, x\rangle - f(x)]} \quad -g^*(y) \right\}$$

To use this, we need to know f^* and g^* — the **convex conjugates**

Why This Perspective is Powerful

One framework, many problems:



Benefits:

- **Unified theory:** One duality framework for all these problems
- **Unified algorithms:** Primal-dual methods work on the saddle point form
- **Modular:** Swap f, g, or A — same algorithm structure

Convex Conjugates: A Deeper Duality

To apply Fenchel duality, we need conjugates f^* and g^*

Definition (Convex Conjugate / Fenchel Conjugate)

$$f^*(y) = \sup_x \{\langle y, x\rangle - f(x)\}$$

Geometric interpretation:

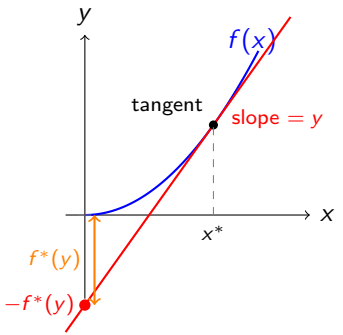
- Find $\ell(x)$, the tangent of f with slope y:

$$\ell(x) = yx - f^*(y)$$

- $f^*(y) =$ negative of y-intercept

Key properties:

- f^* is always convex
- $f^{**} = f$ (for convex, closed f)
- $f(x) + f^*(y) \geq \langle y, x\rangle$



For $f(x) = \frac{1}{2}x^2$: slope y tangent touches at $x^* = y$, giving $f^*(y) = \frac{1}{2}y^2$

Common Conjugates

Function $f(x)$	**Conjugate** $f^*(y)$
$\frac{1}{2}x^T Qx \quad (Q \succ 0)$	$\frac{1}{2}y^T Q^{-1} y$
$\frac{1}{2}\|x\|_2^2$	$\frac{1}{2}\|y\|_2^2$
$\|x\|$ (any norm)	$\delta_{\{\|y\|_* \leq 1\}}(y)$ (indicator of dual ball)
$\|x\|_1$	$\delta_{\{\|y\|_\infty \leq 1\}}(y)$
$\delta_{\mathcal{C}}(x)$ (indicator)	$\sigma_{\mathcal{C}}(y) = \sup_{x \in \mathcal{C}} \langle y, x \rangle$ (support fn)
e^x	$y \log y - y \quad (y > 0)$

Key pattern: Conjugate "inverts" the (derivative of the) function

- Quadratic with $Q \leftrightarrow$ Quadratic with Q^{-1}
- ℓ_1 norm $\leftrightarrow$ ℓ_∞ ball indicator
- $y = \nabla f(x) \Leftrightarrow x = \nabla f^*(y)$

amazon

From Saddle Point to Algorithm

Saddle point: $\min_x \max_y \{f(x) + \langle Ax, y\rangle - g^*(y)\}$

Algorithm: Alternate proximal gradient descent (in x) and ascent (in y)

$$\boxed{\begin{aligned} x^{k+1} &= \text{prox}_{\tau f}\left(x^k - \tau A^T y^k\right) \quad \text{(primal update)} \\ y^{k+1} &= \text{prox}_{\sigma g^*}\left(y^k + \sigma A x^{k+1}\right) \quad \text{(dual update)} \end{aligned}}$$

Ingredients you need:

1. $\text{prox}_{\tau f}$ — proximal operator of f (look up in table)
2. $\text{prox}_{\sigma g^*}$ — use identity: $\text{prox}_{\sigma g^*}(y) = y - \sigma \text{prox}_{g/\sigma}(y/\sigma)$
3. Step sizes: $\tau, \sigma > 0$ with $\tau\sigma\|A\|^2 < 1$

This is the **Primal-Dual Hybrid Gradient (PDHG)** or **Chambolle-Pock** algorithm.

When to Use Which Duality

Fenchel duality: Best when f, g have "simple" structure

Simple g	**Conjugate** g^*
$\|z\|_1$	$\delta_{\|y\|_\infty \leq 1}$
$\|z\|_2$	$\delta_{\|y\|_2 \leq 1}$
$\frac{1}{2}\|z\|^2$	$\frac{1}{2}\|y\|^2$
$\delta_{\{b\}}(z)$	$\langle b, y \rangle$
$\delta_{\{z \leq b\}}$	Easy (separable)

Lagrangian duality: Best for general nonlinear constraints $g(x) \leq 0$

- Dual variables λ handle constraints directly
- No need to compute conjugates of complicated sets

Rule of thumb: Linear constraints + structured objectives $\rightarrow$ Fenchel
Nonlinear constraints $\rightarrow$ Lagrangian

References

[1] S. Boyd and L. Vandenberghe, "Convex Optimization", Ch. 5
`https://web.stanford.edu/~boyd/cvxbook/`

[2] D. Bertsekas, "Nonlinear Programming", Ch. 3-5, Athena Scientific, 1999

[3] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming",
Proc. 2nd Berkeley Symposium, 1951

[4] W. Karush, "Minima of Functions of Several Variables with Inequalities as Side Constraints", M.Sc. Thesis, U. Chicago, 1939

[5] R. T. Rockafellar, "Convex Analysis", Princeton, 1970

Part V

First-Order Methods for Constrained Optimization

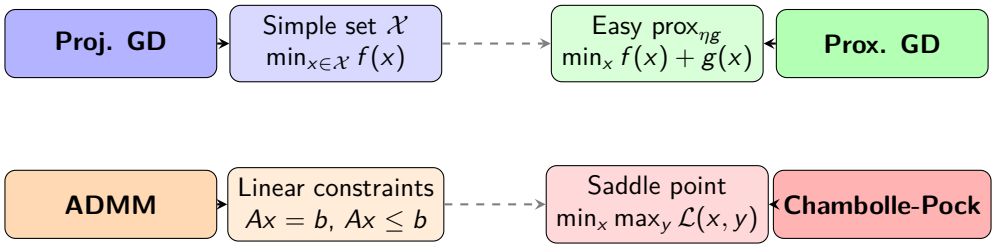
Goal: Solve constrained convex optimization

$$\underset{x}{\text{minimize}} \quad f(x) + g(x)$$

where f is smooth, g may be non-smooth (encodes constraints/regularization)

- **Projected gradient descent**
  $g(x) = \delta_{\mathcal{X}}(x)$ (indicator of set with cheap projection: box, ball, polyhedron)
- **Proximal gradient methods**
  $g(x) = \lambda\|x\|_1$ or other non-smooth regularizer with cheap prox
- **ADMM**
  $g(x) = h(Ax)$ or separable $g(x) = \sum_i g_i(x_i)$ (splitting structure)
- **Mirror descent**
  $g(x) = \delta_{\Delta_n}(x)$ (indicator of simplex constraints)
- **Primal-dual methods**
  $g(x) = h(Kx)$ with linear operator K (e.g., $K = \nabla$ for TV)

The Constrained Optimization Landscape



Key insight: Match algorithm to problem structure!

Projected Gradient Descent

Problem: $\min_{x \in \mathcal{X}} f(x), \quad \mathcal{X}$ closed convex

Algorithm:

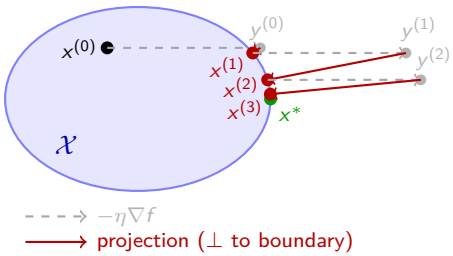
$$\boxed{x^{(k+1)} = \Pi_{\mathcal{X}}\left(x^{(k)} - \eta \nabla f(x^{(k)})\right)}$$

Two steps:

1. Gradient step (may leave $\mathcal{X}$)
2. Project back onto $\mathcal{X}$

Projection:

$$\Pi_{\mathcal{X}}(y) = \text{argmin}_{x \in \mathcal{X}} \|x - y\|^2$$



Projection finds the **nearest point** on boundary

Key property: Projection is **non-expansive**: $\|\Pi_{\mathcal{X}}(y_1) - \Pi_{\mathcal{X}}(y_2)\| \leq \|y_1 - y_2\|$

Common Projections

Set $\mathcal{X}$	Projection $\Pi_{\mathcal{X}}(y)$	Cost	
Box $[l, u]$	$\text{clip}(y, l, u)$	$O(n)$	✓
ℓ_2-ball $\|x\| \leq r$	$\begin{cases} y & \|y\| \leq r \\ r \cdot y/\|y\| & \text{else} \end{cases}$	$O(n)$	✓
Halfspace $a^T x \leq b$	$y - \frac{(a^T y - b)^+}{\|a\|^2} a$	$O(n)$	✓
Simplex Δ_n	Sort + threshold	$O(n \log n)$	✓
Affine $Ax = b$	$y - A^T(AA^T)^{-1}(Ay - b)$	$O(mn)^*$	✗
Polyhedron $Ax \leq b$	Solve QP	expensive	✗

Box projection:

$$[\Pi(y)]_i = \min(\max(y_i, l_i), u_i)$$

Ball projection:

$$\Pi(y) = \frac{r}{\max(r, \|y\|)} \cdot y$$

Projected GD: Convergence

Theorem (Convergence of Projected GD)

Let f be convex and M-smooth, $\mathcal{X}$ closed convex. With $\eta = 1/M$:

M-smooth:

$$f(x^{(k)}) - f^* \leq \frac{M\|x^{(0)} - x^*\|^2}{2k} \qquad (O(1/\epsilon) \text{ iterations})$$

m-strongly convex:

$$\|x^{(k)} - x^*\|^2 \leq \left(1 - \frac{m}{M}\right)^k \|x^{(0)} - x^*\|^2 \qquad (O(\kappa \log(1/\epsilon)) \text{ iterations})$$

Key insight: Same rates as unconstrained GD!

Why? Non-expansiveness of projection: constraint doesn't slow us down (if projection is cheap)

Projected Gradient Descent: Example Problems

1. Box-constrained regression

$$\min_{l \le x \le u} \frac{1}{2}\|Ax - b\|^2$$

Physical bounds on parameters (e.g., concentrations ≥ 0, probabilities ≤ 1). Projection: $O(n)$ clipping.

2. Portfolio optimization

$$\min_{x \in \Delta_n} x^T \Sigma x - \mu^T x$$

Weights on simplex $\Delta_n = \{x \geq 0, \mathbf{1}^T x = 1\}$. Projection: $O(n \log n)$ sort.

3. Low-rank matrix completion

$$\min_{\|X\|_F \le r} \sum_{(i,j) \in \Omega} (X_{ij} - M_{ij})^2$$

Frobenius ball: projection is $O(mn)$ scaling.

4. Signal recovery with energy budget

$$\min_{\|x\|_2 \le P} \|Ax - b\|^2$$

Power constraint in communications. ℓ_2-ball projection: $O(n)$ scaling.

5. SVM dual

$$\min_{0 \le \alpha \le C} \frac{1}{2}\alpha^T Q \alpha - \mathbf{1}^T \alpha$$

Box constraints on dual variables. Projection: $O(n)$ clipping.

6. Constrained least squares

$$\min_{\{x:\, a^T x \le b\}} \|Ax - c\|^2$$

Halfspace: projection is $O(n)$ closed-form.

Proximal Operators: Beyond Projection

Problem: $\min_x f(x) + g(x)$ where f smooth, g **non-smooth**

Definition (Proximal Operator)

$$\text{prox}_{\eta g}(y) = \text{argmin}_x \left\{ g(x) + \frac{1}{2\eta}\|x - y\|^2 \right\}$$

Interpretation:

- Balance minimizing $g(x)$...
- ...with staying close to y
- η controls the trade-off

Special case:
For $g = \delta_{\mathcal{X}}$ (indicator):

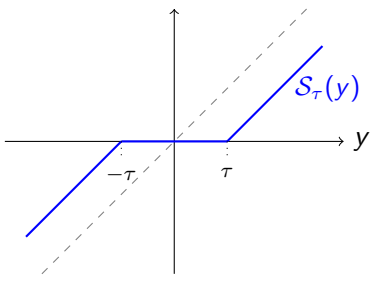
$$\text{prox}_{\eta \delta_{\mathcal{X}}}(y) = \Pi_{\mathcal{X}}(y)$$

Projection is a proximal operator!

Soft-Thresholding: Prox of ℓ_1 Norm

For $g(x) = \lambda\|x\|_1$:

$$\boxed{[\text{prox}_{\eta g}(y)]_i = \text{sign}(y_i) \cdot \max(|y_i| - \eta\lambda, 0) = \mathcal{S}_{\eta\lambda}(y_i)}$$



Effect:

- $\|x\|_1$ wants $x = 0$ ($\uparrow$ with λ)
- $\|x - y\|^2$ wants $x = y$ ($\downarrow$ with η)

Why it works:

- Values in $[-\tau, \tau] \to$ **zero**
- Other values shrunk by τ
- **Induces sparsity!**

Common Proximal Operators

Function $g(x)$	**Prox**$_{\eta g}(y)$	**Name**
$\delta_{\mathcal{X}}(x)$	$\Pi_{\mathcal{X}}(y)$	Projection
$\lambda\|x\|_1$	$\text{sign}(y) \odot (\|y\| - \eta\lambda)^+$	Soft-threshold
$\frac{\lambda}{2}\|x\|_2^2$	$\frac{y}{1+\eta\lambda}$	Shrinkage
$\lambda\|x\|_2$	$\left(1 - \frac{\eta\lambda}{\|y\|_2}\right)^+ y$	Group soft-thresh
$\delta_{\{\|x\|_2 \le r\}}(x)$	$\frac{r}{\max(r, \|y\|_2)} y$	Ball projection

Separability: If $g(x) = \sum_i g_i(x_i)$, then prox decomposes:

$$\left[\text{prox}_{\eta g}(y)\right]_i = \text{prox}_{\eta g_i}(y_i) \qquad \text{(solve } n \text{ independent 1D problems)}$$

Proximal Gradient Method (ISTA)

Problem: $\min_x f(x) + g(x)$ (f smooth, g non-smooth)

ISTA (Iterative Shrinkage-Thresholding Algorithm)

1: **for** $k = 0, 1, \ldots$ **do**
2: $\quad y^{(k)} \leftarrow x^{(k)} - \eta \nabla f(x^{(k)})$ // cheap forward step on f
3: $\quad x^{(k+1)} \leftarrow \text{prox}_{\eta g}(y^{(k)})$ // backward step on g (works for some functions)
4: **end for**

Interpretation: Minimize local quadratic model of f, plus exact g:

$$x^{(k+1)} = \text{argmin}_x \left\{ \underbrace{f(x^{(k)}) + \nabla f(x^{(k)})^T (x - x^{(k)}) + \frac{1}{2\eta} \|x - x^{(k)}\|^2}_{\text{quadratic approx of } f} + g(x) \right\}$$

Convergence

With $\eta = 1/M$: $\quad F(x^{(k)}) - F^* \leq \dfrac{M \|x^{(0)} - x^*\|^2}{2k}$

amazon

Why "Forward-Backward Splitting"?

Forward step (explicit, on f): Evaluate gradient at **current** point

$$y^{(k)} = x^{(k)} - \eta \nabla f(x^{(k)})$$

Backward step (implicit, on g): Optimality of $\text{prox}_{\eta g}$ gives

$$0 \in \partial g(x^{(k+1)}) + \tfrac{1}{\eta}(x^{(k+1)} - y^{(k)})$$

which rearranges to

$$x^{(k+1)} = y^{(k)} - \eta\, \partial g(x^{(k+1)})$$

Subgradient evaluated at the **next** (unknown) point!

ISTA = forward on f + backward on g:

$$x^{(k+1)} = \underbrace{\text{prox}_{\eta g}}_{\text{backward}}\Big(\underbrace{x^{(k)} - \eta \nabla f(x^{(k)})}_{\text{forward}}\Big)$$

FISTA: Accelerated Proximal Gradient (ISTA + Nesterov)

Algorithm 1 *

FISTA (Fast ISTA)

1: $y^{(0)} \leftarrow x^{(0)}$, $t_0 \leftarrow 1$
2: **for** $k = 0, 1, \ldots$ **do**
3: $\quad x^{(k+1)} \leftarrow \text{prox}_{\eta g}(y^{(k)} - \eta \nabla f(y^{(k)}))$
4: $\quad t_{k+1} \leftarrow (1 + \sqrt{1 + 4t_k^2})/2$
5: $\quad y^{(k+1)} \leftarrow x^{(k+1)} + \frac{t_k - 1}{t_{k+1}}(x^{(k+1)} - x^{(k)})$
6: **end for**

Optimal Convergence ($F = f + g$)

$$F(x^{(k)}) - F^* \leq \frac{2M\|x^{(0)} - x^*\|^2}{(k+1)^2}$$

	ISTA	**FISTA**
Rate	$O(1/k)$	$O(1/k^2)$
Iters for ϵ	$O(1/\epsilon)$	$O(1/\sqrt{\epsilon})$

Key insight: Momentum term $\frac{t_k - 1}{t_{k+1}} \approx \frac{k-1}{k+2}$ uses past iterates to "look ahead"; achieves optimal first-order rate for smooth + non-smooth composite problems.

Application: LASSO for Sparse Recovery

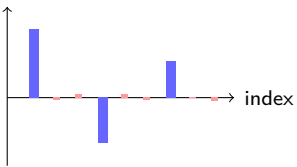
$$\min_x \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$$

ISTA for LASSO:

1. $y = x - \eta A^T(Ax - b)$
2. $x^+ = \mathcal{S}_{\eta\lambda}(y)$

Step size: $\eta = 1/\lambda_{\max}(A^TA)$

Each iteration: $O(mn)$ for matrix-vector products



Sparse signal + noise

Effect of λ:

- Small λ: fits noise
- Large λ: too sparse
- Just right: recovers signal

Dual Ascent

Problem: $\min_x f(x)$ s.t. $Ax = b$ f is strictly convex

Lagrangian: $L(x, \lambda) = f(x) + \lambda^T(Ax - b)$

Dual function: $D(\lambda) = \inf_x L(x, \lambda)$ (concave in λ)

Algorithm:

$$x^{(k+1)} = \text{argmin}_x L(x, \lambda^{(k)})$$

$$\lambda^{(k+1)} = \lambda^{(k)} + \eta(Ax^{(k+1)} - b)$$

Interpretation:

- Step 1: Find best x for current prices λ
- Step 2: Gradient ascent on dual $D(\lambda)$

Subgradient of dual

If $x^* \in \text{argmin}_x L(x, \lambda)$, then

$$g = Ax^* - b$$

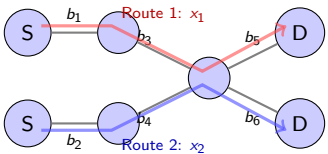
is a subgradient of D at λ.

The **constraint residual** tells us which direction to move λ!

Raise λ_i when constraint i is violated ($[Ax - b]_i > 0$); lower when slack

Application: Internet Congestion Control

Network: Links $\ell = 1, \ldots, L$ with capacities b_ℓ; Routes $r = 1, \ldots, R$ with rates $x_r \geq 0$



Optimization problem:

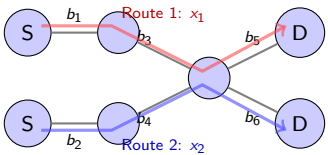
$$\max_{x \geq 0} \sum_r U_r(x_r) \quad \text{s.t.} \quad \sum_{r:\ell \in r} x_r \leq b_\ell \quad \forall \ell$$

- $U_r(x_r)$: utility of route r (concave, e.g. $\log x_r$)
- Capacity constraints couple all routes sharing a link

Challenge: Routes are controlled by **different users** who don't know each other's utilities!

Application: Internet Congestion Control

Network: Links $\ell = 1, \ldots, L$ with capacities b_ℓ; Routes $r = 1, \ldots, R$ with rates $x_r \geq 0$



Optimization problem:

$$\max_{x \geq 0} \sum_r U_r(x_r) \quad \text{s.t.} \quad \sum_{r:\ell \in r} x_r \leq b_\ell \quad \forall \ell$$

- $U_r(x_r)$: utility of route r (concave, e.g. $\log x_r$)
- Capacity constraints couple all routes sharing a link

Challenge: Routes are controlled by **different users** who don't know each other's utilities!

Lagrangian:

$$L(x, \lambda) = \sum_r U_r(x_r) - \sum_\ell \lambda_\ell \left(\sum_{r:\ell \in r} x_r - b_\ell \right) = \sum_r \Big[U_r(x_r) - x_r \underbrace{\sum_{\ell \in r} \lambda_\ell}_{q_r \text{ (path price)}} \Big] + \sum_\ell \lambda_\ell b_\ell$$

Lagrangian **decomposes by route** for fixed λ! Each user solves independently.

Congestion Control via Dual Ascent

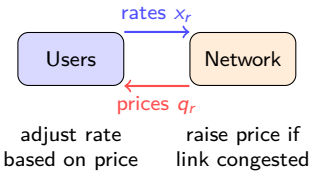
For fixed prices λ, each route independently solves:

$$x_r^*(\lambda) = \text{argmax}_{x_r \geq 0} \left\{ U_r(x_r) - x_r \cdot q_r \right\}, \qquad q_r = \sum_{\ell \in r} \lambda_\ell$$

For $U_r(x) = \log(x)$: $\quad x_r^* = 1/q_r \quad$ ("proportional fairness")

Distributed protocol:

1. **Network signals path delay:**
   Send price $q_r = \sum_{\ell \in r} \lambda_\ell$ to route r
2. **Users regulate rate:**
   Set rate $x_r = (U_r')^{-1}(q_r)$
3. **Rates generate link delays:**
   Update link prices:
   $\lambda_\ell^+ = \left[\lambda_\ell + \eta(\text{load}_\ell - b_\ell)\right]^+$



Q: *Why must we project λ_ℓ onto $\mathbb{R}_+$?*

This is how TCP works!

Dual Ascent: Dual Converges, But What About Primal?

Good news: $D(\lambda)$ is always concave, so dual ascent (= gradient ascent on D) converges to λ^* under standard conditions

Bad news: Having λ^* doesn't mean we can recover x^*!

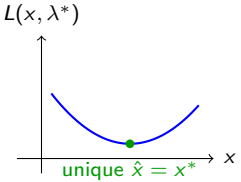
Primal recovery: Given λ^*, solve $\hat{x} \in \operatorname{argmin}_x L(x, \lambda^*)$ and hope $\hat{x} = x^*$

f strictly convex:

$\operatorname{argmin}_x L(x, \lambda^*)$ is a **unique point**

Since x^* must be in this set $\Rightarrow \hat{x} = x^*$

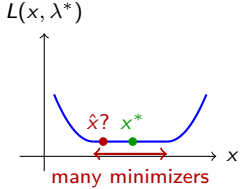
Moreover $x(\lambda)$ continuous, so $\lambda^{(k)} \to \lambda^*$ implies $x^{(k)} \to x^*$



f convex, not strictly:

$\operatorname{argmin}_x L(x, \lambda^*)$ has a **set** of minimizers which includes x^* and **infeasible** points!

Algorithm can pick: $\Rightarrow Ax^{(k)} \neq b$



What Goes Wrong Without Strict Convexity

Example: $\min_{x \in \mathbb{R}^3} \frac{1}{2}(x_1^2 + x_2^2)$ s.t. $x_1 + x_2 + x_3 = 1$

f is convex but **not** strictly convex (doesn't depend on x_3; Hessian $= \text{diag}(1, 1, 0)$)

What dual ascent does:

$L(x, \lambda) = \frac{1}{2}(x_1^2 + x_2^2) + \lambda(x_1 + x_2 + x_3 - 1)$

$\frac{\partial L}{\partial x_1} = x_1 + \lambda = 0 \Rightarrow x_1 = -\lambda$

$\frac{\partial L}{\partial x_2} = x_2 + \lambda = 0 \Rightarrow x_2 = -\lambda$

$\frac{\partial L}{\partial x_3} = \lambda$

If $\lambda \neq 0$: $\inf_{x_3} L = -\infty$ (unbounded!)

If $\lambda = 0$: minimizers $= \{(0, 0, x_3) : x_3 \in \mathbb{R}\}$

$\lambda^* = 0$, but minimizer set is a **line**

True solution: $x^* = (0, 0, 1)$ — but algorithm could pick $(0, 0, 47)$

Observations:

✗ $x(\lambda)$ **not unique**

x_3 is free $\Rightarrow$ infinitely many minimizers

✗ **Primal infeasible**

Algorithm picks arbitrary x_3

$\Rightarrow x_1 + x_2 + x_3 \neq 1$

✗ $D(\lambda)$ **non-smooth**

$$D(\lambda) = \begin{cases} -\lambda & \lambda = 0 \\ -\infty & \lambda \neq 0 \end{cases}$$

Gradient ascent can't even start!

$\Downarrow$

Need a fix!

Augmented Lagrangian: Fixing Dual Ascent

Idea: Add a quadratic penalty for constraint violation

Augmented Lagrangian:

$$\boxed{L_\rho(x, \lambda) = f(x) + \lambda^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|^2}$$

Method of Multipliers:

$$x^{(k+1)} = \text{argmin}_x L_\rho(x, \lambda^{(k)})$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \rho(Ax^{(k+1)} - b)$$

Note: Dual step size $= \rho$ (not a free parameter!)

This is gradient ascent on $D_\rho(\lambda) = \inf_x L_\rho(x, \lambda)$, which is **smooth** with Lipschitz constant $1/\rho$.

Why it fixes dual ascent:

- ✓ $D_\rho(\lambda)$ is **differentiable** (even if f not strictly cvx)
- ✓ Penalty **pushes** $x^{(k)}$ toward feasibility during iterations

amazon

Augmented Lagrangian: The Catch

Comparison:

	Dual Ascent	Aug. Lagrangian
Requires strict cvx	Yes	No
Dual smoothness	Not always	Yes
Primal feasibility	Not guaranteed	Encouraged
Convergence	Fragile	Robust
Decomposable?	**Yes!**	**No!**

The problem:

The penalty $\frac{\rho}{2}\|Ax - b\|^2$ **couples all variables** through A!

Can't distribute the x-update!

Q: *Can we get the best of both Aug. Lagrangian + Dual Ascent?*

$\Downarrow$

ADMM

ADMM: Splitting for Decomposition

Problem structure:

$$\min_{x,z} f(x) + g(z) \quad \text{subject to} \quad Ax + Bz = c$$

Key idea: Split problem into easier subproblems

Augmented Lagrangian:

$$L_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|^2$$

ADMM alternates:

$$x^+ \leftarrow \text{argmin}_x L_\rho(x, z, \lambda)$$
$$z^+ \leftarrow \text{argmin}_z L_\rho(x^+, z, \lambda)$$
$$\lambda^+ \leftarrow \lambda + \rho(Ax^+ + Bz^+ - c)$$

Why it works:

- x-update: only involves f
- z-update: only involves g
- Each may be easy!
- Penalty ρ couples them

Convergence: $O(1/k)$
Under strong cvx: linear rate

ADMM for LASSO

Reformulate: $\min_{x,z} \frac{1}{2}\|Ax - b\|^2 + \lambda\|z\|_1$ s.t. $x = z$

x-update: Ridge regression

$$x^+ = (A^TA + \rho I)^{-1}(A^Tb + \rho(z - u))$$

z-update: Soft-thresholding

$$z^+ = \mathcal{S}_{\lambda/\rho}(x^+ + u)$$

u-update: (scaled dual)

$$u^+ = u + x^+ - z^+$$

Advantages over ISTA:

- Can precompute $(A^TA + \rho I)^{-1}$
- Often faster for medium-sized problems
- Natural for distributed settings

Choosing ρ:

- Too small: slow convergence
- Too large: ill-conditioning
- Adaptive: balance residuals

ADMM for Distributed Consensus

Problem: $\min_x \sum_{i=1}^{B} f_i(x)$ (agent i knows only f_i)

Reformulation: Give each agent a local copy, enforce consensus

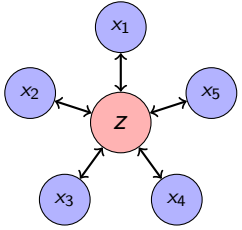
$\min_{x_1,\ldots,x_B,z} \sum_{i=1}^{B} f_i(x_i)$ s.t. $x_i = z \ \forall i$

ADMM updates:

$$x_i^+ = \text{argmin}_{x_i} \left\{ f_i(x_i) + \frac{\rho}{2}\|x_i - z + u_i\|^2 \right\}$$

$$z^+ = \frac{1}{B}\sum_{i=1}^{B}(x_i^+ + u_i)$$

$$u_i^+ = u_i + x_i^+ - z^+$$



Agents solve local problems
Coordinator averages

Communication: Each round, agents send x_i to coordinator, receive z

What About Nonlinear Constraints?

All dual/ADMM methods assumed **affine constraints** $Ax + Bz = c$. Why?

Augmented Lagrangian for $\min_x f(x)$ s.t. $g(x) \leq 0$, g convex:

$$L_\rho(x, \lambda) = f(x) + \lambda^T g(x) + \tfrac{\rho}{2}\|g(x)\|^2$$

Affine: $g(x) = Ax - b$

$\|Ax - b\|^2$ is convex quadratic in x ✓

$\Rightarrow$ x-subproblem is convex ✓

$\Rightarrow$ easy to solve ✓

Nonlinear convex: e.g., $g(x) = x^2 - 1$

$\|g(x)\|^2 = (x^2 - 1)^2 = x^4 - 2x^2 + 1$

$\frac{d^2}{dx^2} = 12x^2 - 4 < 0$ for $|x| < \frac{1}{\sqrt{3}}$ ✗

$\Rightarrow$ penalty is **nonconvex** ✗

What to do for nonlinear convex constraints:

Approach	Idea
Projection	If $\Pi_{\{g \leq 0\}}$ is cheap $\rightarrow$ use projected GD or proximal GD
Linearization	$g(x) \approx g(x^k) + \nabla g(x^k)^T(x - x^k) \leq 0 \rightarrow$ halfspace, then ADMM
Interior point	Barrier $-\sum_i \log(-g_i(x))$ keeps iterates strictly feasible (next Ch.)
Dual ascent	$\lambda^+ = [\lambda + \eta\, g(x^+)]^+$ still valid (dual always concave), but with above limitations

Mirror Descent: Convergence Depends on the Norm

Problem: $\min_{x \in \Delta_n} f(x)$ where $\Delta_n = \{x \geq 0, \sum_i x_i = 1\}$ (probability simplex)

Arises in: bandwidth allocation, portfolio weights, probability distributions, attention mechanisms

Key idea: Replace $\|x - y\|_2^2$ with a **Bregman divergence** $D_\phi(x, y)$:

$$x^{(k+1)} = \text{argmin}_{x \in \Delta_n} \left\{ \nabla f(x^{(k)})^T x + \tfrac{1}{\eta} D_\phi(x, x^{(k)}) \right\}$$

where ϕ is σ-strongly convex w.r.t. $\|\cdot\|_p$ on Δ_n.

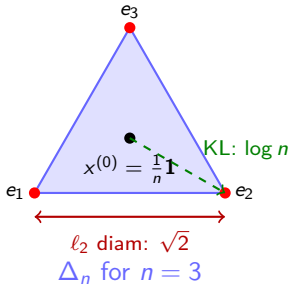
General convergence bound

$$f(\bar{x}^{(k)}) - f^* \;\leq\; L\sqrt{\frac{2\, n^{2-2/p} \cdot Z}{\sigma \cdot k}}$$

$n^{2-2/p}$ — **norm factor** — how gradient scales with dimension n

$Z = D_\phi(x^*, x^{(0)})$ — **initial distance** — Bregman diameter

σ — **strong convexity** of ϕ w.r.t. $\|\cdot\|_p$



Goal: Choose ϕ (and hence p, σ, Z) to minimize $n^{2-2/p} \cdot Z/\sigma$

amazon

Euclidean vs. Entropy: From n to $\log n$

Which norm p minimizes $n^{2-2/p} \cdot Z/\sigma$ over $\Delta_n = \{x \geq 0, \mathbf{1}^T x = 1\}$?

Euclidean ($p = 2$): $\phi = \frac{1}{2}\|x\|_2^2$

Norm factor: $n^{2-2/2} = \mathbf{n}$
Strong cvx: $\sigma = 1$
Diameter: $Z \leq 1$

Product: $\mathbf{n}$

$f(\bar{x}^{(k)}) - f^* \leq L\sqrt{2n/k}$

Iterations for ε: $O(n/\varepsilon^2)$

Entropy ($p = 1$): $\phi = \sum_i x_i \log x_i$

Norm factor: $n^{2-2/1} = n^0 = \mathbf{1}$
Strong cvx: $\sigma = 1$ (w.r.t. $\|\cdot\|_1$)
Diameter: $Z \leq \log n$

Product: $\log \mathbf{n}$

$f(\bar{x}^{(k)}) - f^* \leq L\sqrt{2\log n/k}$

Iterations for ε: $O(\log n/\varepsilon^2)$

	$n = 10^3$	$n = 10^5$	$n = 10^7$
Euclidean iters	$\propto 1000$	$\propto 100{,}000$	$\propto 10{,}000{,}000$
Entropy iters	$\propto 7$	$\propto 12$	$\propto 16$
Speedup	$140\times$	$8{,}300\times$	$620{,}000\times$

Exponentiated Gradient: The Update Rule

Mirror descent with entropy $\phi(x) = \sum_i x_i \log x_i$ on $\Delta_n = \{x \geq 0, \mathbf{1}^T x = 1\}$:

$$x_i^{(k+1)} = \frac{x_i^{(k)} \exp\big(-\eta \left[\nabla f(x^{(k)})\right]_i\big)}{\sum_j x_j^{(k)} \exp\big(-\eta \left[\nabla f(x^{(k)})\right]_j\big)}$$

Why it works:

- Multiply weight by $e^{-\eta \nabla_i f}$
  - Large gradient $\Rightarrow$ aggressive scaling
  - Small gradient $\Rightarrow$ gentle update
- Renormalize so $\sum_i x_i = 1$
- $x_i^{(k)} > 0$ always if $x_i^{(0)} > 0$
- **No projection needed!**

Per-iteration comparison:

	Proj. GD	**Exp. Grad.**
Compute ∇f	$O(n)$	$O(n)$
Project/update	sort $O(n \log n)$	exp $O(n)$
Total/iter	$O(n \log n)$	$O(n)$
Iters for ε	$O(n/\varepsilon^2)$	$O(\log n/\varepsilon^2)$

Also called: Multiplicative Weights Update, Hedge algorithm (online learning)

Init: $x^{(0)} = \mathbf{1}/n$ (minimizes max KL to any vertex)

Savings from **two sources**: fewer iterations ($n \to \log n$) **and** cheaper per-iteration ($n \log n \to n$)

Primal-Dual Methods: Saddle Point View

Problem: $\min_x f(x) + g(Kx)$ (f smooth, g convex, K linear operator)

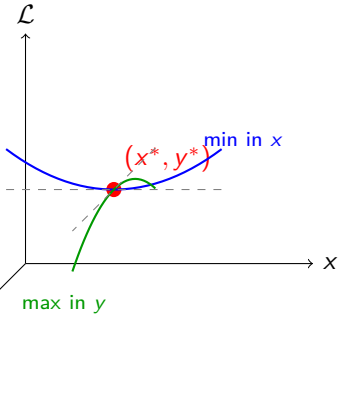
Saddle point reformulation: Using $g(z) = \max_y \{y^T z - g^*(y)\}$:

$$\min_x \max_y \underbrace{f(x) + \langle Kx, y\rangle - g^*(y)}_{\mathcal{L}(x,y)}$$

Examples:

LASSO	: $K = I$, $g = \lambda\|\cdot\|_1$
Equality constr.	: $g = \delta_{\{b\}}$, $g^* = b^T y$
TV denoising	: $K = \nabla$, $g = \lambda\|\cdot\|_1$
Inequality constr.	: $g = \delta_{\leq 0}$, $y \geq 0$

Key insight: Dual variable y acts as Lagrange multiplier for constraint $Kx \in \mathrm{dom}(g)$



From Saddle Point to Algorithms

Goal: $\min_x \max_y \mathcal{L}(x,y) = f(x) + \langle Kx, y\rangle - g^*(y)$

1. Simultaneous GD-Ascent ✗

$$x^{k+1} = x^k - \tau(\nabla f(x^k) + K^T y^k)$$

$$y^{k+1} = y^k + \sigma(Kx^k - \nabla g^*(y^k))$$

Both use **stale** info from step $k \Rightarrow$ oscillates/diverges

2. Simultaneous Proximal ✗

$$x^{k+1} = \text{prox}_{\tau f}(x^k - \tau K^T y^k)$$

$$y^{k+1} = \text{prox}_{\sigma g^*}(y^k + \sigma K x^k)$$

Prox handles f, g^*, but still simultaneous $\Rightarrow$ can diverge

3. Chambolle-Pock ✓

$$y^{k+1} = \text{prox}_{\sigma g^*}(y^k + \sigma K \bar{x}^k)$$

$$x^{k+1} = \text{prox}_{\tau f}(x^k - \tau K^T y^{k+1})$$

$$\bar{x}^{k+1} = x^{k+1} + \theta(x^{k+1} - x^k)$$

Fresh y^{k+1} + **extrapolated** $\bar{x} \Rightarrow$ converges!

4. ADMM ✓

$$x^{k+1} = \text{argmin}_x f(x) + \tfrac{\rho}{2}\|Kx - z^k + u^k\|^2$$

$$z^{k+1} = \text{prox}_{g/\rho}(Kx^{k+1} + u^k)$$

$$u^{k+1} = u^k + Kx^{k+1} - z^{k+1}$$

Quadratic penalty couples K and $x \Rightarrow$ converges!

Two fixes: Chambolle-Pock = alternate + momentum ADMM = augmentation (ρ-penalty)

Chambolle-Pock (PDHG)

Problem: $\min_x \max_y \; f(x) + \langle Kx, y \rangle - g^*(y)$

Algorithm 2 *

Chambolle-Pock (Primal-Dual Hybrid Gradient)

1: Initialize $x^{(0)}$, $y^{(0)}$, $\bar{x}^{(0)} = x^{(0)}$
2: **for** $k = 0, 1, \ldots$ **do**
3: $\quad y^{(k+1)} \leftarrow \text{prox}_{\sigma g^*}(y^{(k)} + \sigma K \bar{x}^{(k)})$
4: $\quad x^{(k+1)} \leftarrow \text{prox}_{\tau f}(x^{(k)} - \tau K^T y^{(k+1)})$
5: $\quad \bar{x}^{(k+1)} \leftarrow x^{(k+1)} + \theta(x^{(k+1)} - x^{(k)})$
6: **end for**

Step sizes: $\tau\sigma\|K\|^2 < 1$, $\theta = 1$

Key ingredients:

- Prox of f (primal)
- Prox of g^* (dual)
- Matrix-vector with K, K^T

Moreau identity:

$$\text{prox}_{\sigma g^*}(y) = y - \sigma \text{prox}_{g/\sigma}(y/\sigma)$$

Compute prox of g^* from prox of g!

Convergence: $O(1/k)$ for convex; linear under strong convexity.

Convergence Rates Summary

Solve: $\text{minimize}_x \quad f(x) + g(x)$

Method	f **Lip. Cvx**[1]	f **Sm. Cvx**	f **Sm.+Str.Cvx**	**Requires**
Subgradient	$O(1/\sqrt{k})$	$O(1/k)$	$O((1-m/M)^k)$	easy proj on $g(x) \leq 0$
ISTA	—	$O(1/k)$	$O((1-m/M)^k)$	∇f, easy prox_g
FISTA	—	$O(1/k^2)$	$O((1-\sqrt{m/M})^k)$	∇f, easy prox_g
Mirror Desc.	$O\left(\sqrt{\frac{\log n}{k}}\right)$[2]	—	—	suitable D_ϕ
ADMM	—	$O(1/k)$	linear	linear split, easy subprob
Chamb.-Pock	—	$O(1/k)$	linear	linear K, easy prox

[1] Non-smooth, bounded subgradient norm.
[2] On simplex Δ_n with entropy.

Key insight: Non-smooth $\Rightarrow O(1/\sqrt{k})$; smooth $\Rightarrow O(1/k)$; acceleration $\Rightarrow O(1/k^2)$.

Method Selection Guide

Problem Structure	**Best Method**	**Alternative**
Simple set (box, ball)	Projected GD	—
Simplex (high-dim)	Mirror descent	Projected GD
ℓ_1 regularization	Proximal GD / FISTA	ADMM
TV regularization	Chambolle-Pock	ADMM
Linear constraints $Ax = b$	ADMM	Chambolle-Pock
Distributed / consensus	ADMM	Dual ascent

References

[1] A. Beck and M. Teboulle, "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems", SIAM J. Imaging Sciences, 2009

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, "Distributed Optimization and Statistical Learning via ADMM", Found. & Trends in ML, 2011

[3] A. Chambolle and T. Pock, "A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging", J. Math. Imaging Vision, 2011

[4] A. Beck, "First-Order Methods in Optimization", SIAM, 2017

[5] S. Bubeck, "Convex Optimization: Algorithms and Complexity", Found. & Trends in ML, 2015

Part VI

Newton's Method and Interior Point Methods

Outline

Goal: Solve convex optimization with high accuracy

$$\underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad g_i(x) \leq 0, \quad Ax = b$$

- **Newton's method** – second-order information for fast convergence
- **Equality constraints** – Newton via KKT system
- **Interior point methods** – barrier functions for inequalities

Second-order methods: fewer iterations, but $O(n^3)$ per iteration

Newton's Method

Iterative algorithm:

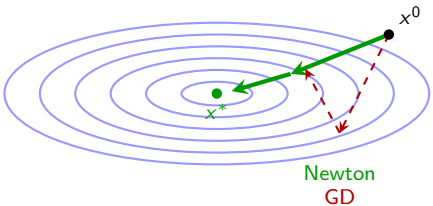
$$x^{k+1} = x^k - \underbrace{\eta^k}_{\text{step size}} \underbrace{[\nabla^2 f(x^k)]^{-1}}_{\text{Hessian inv.}} \underbrace{\nabla f(x^k)}_{\text{gradient}}$$

Newton step:

$$\Delta x_{\text{nt}} = -[\nabla^2 f(x^k)]^{-1} \nabla f(x^k)$$

Step size choices:

- **Pure Newton:** $\eta = 1$
- **Line search:** $\eta^* = \arg\min_\eta f(x^k + \eta \Delta x_{\text{nt}})$
- **Backtracking:** Armijo's rule



Newton takes direct path; GD zigzags on ill-conditioned problems

Newton's Method

Problem: $\text{minimize}_x\, f(x)$

Iterative algorithm:

$$x^{k+1} = x^k - \underbrace{\eta^k}_{\text{step size}} \underbrace{[\nabla^2 f(x^k)]^{-1}}_{\text{Hessian inv.}} \underbrace{\nabla f(x^k)}_{\text{gradient}}$$

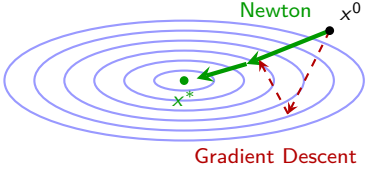
Newton step:

$$\Delta x_{\text{nt}} = -[\nabla^2 f(x^k)]^{-1} \nabla f(x^k)$$

Step size choices:

- **Pure Newton:** $\eta = 1$
- **Line search:** $\eta^* = \arg\min_\eta f(x^k + \eta \Delta x_{\text{nt}})$
- **Backtracking:** Armijo's rule

Isaac Newton (1643–1727)



Why Newton's Method?

Key idea: Use **second-order Taylor expansion** around x^k:

$$f(y) \approx f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}(y-x)^T\nabla^2 f(x)(y-x)$$

Find y that minimizes the approximation:

$$\nabla f(x) + \nabla^2 f(x)(y-x) = 0 \quad \Rightarrow \quad y^* = x - [\nabla^2 f(x)]^{-1}\nabla f(x)$$

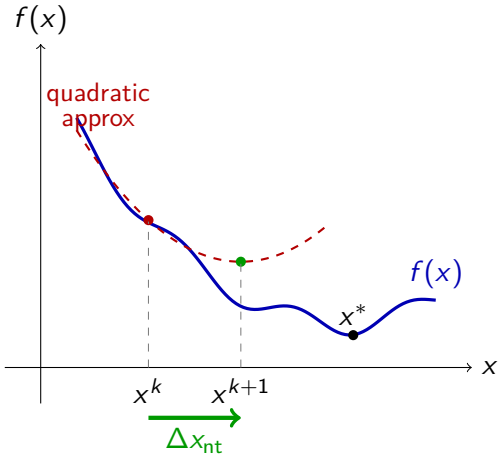
Properties of 2nd order Taylor model:

- **Exact** for quadratic functions
- **Good approximation** locally

Key Insight

Pure Newton step ($\eta = 1$) lands at the **exact minimum of the quadratic approximation**

Why Newton's Method? – Geometric View



Newton minimizes the local quadratic model, then repeats. For quadratic f, one step suffices!

Newton's Decrement

Q: *How much improvement do we expect from one Newton step?*

Newton's decrement: Substitute $y^* = x + \Delta x_{\text{nt}}$ into Taylor expansion:

$$f(x) - f(y^*) \approx \frac{1}{2}\nabla f(x)^T[\nabla^2 f(x)]^{-1}\nabla f(x) \triangleq \frac{1}{2}\lambda^2(x)$$

Definition (Newton Decrement)

$$\lambda(x) = \sqrt{\nabla f(x)^T[\nabla^2 f(x)]^{-1}\nabla f(x)}$$

Stopping Criterion

Stop when $\lambda^2(x)/2 \leq \epsilon$ (expected improvement is small)

Newton's Method with Backtracking

Algorithm 3 *

Newton's Method with Backtracking

Input: tolerance ϵ, $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$, initial x^0
for $k = 0, 1, 2, \ldots$ **do**
  Solve $\nabla^2 f(x^k)\Delta x = -\nabla f(x^k)$; $\lambda^2 \leftarrow -\nabla f(x^k)^T \Delta x$
  if $\lambda^2/2 \leq \epsilon$ **then return** x^k
  end if
  $\eta \leftarrow 1$
  while $f(x^k + \eta\Delta x) > f(x^k) - \alpha\eta\lambda^2$ **do** $\eta \leftarrow \beta\eta$
  end while
  $x^{k+1} \leftarrow x^k + \eta\Delta x$
end for

Typical parameters:

- $\alpha = 0.25$, $\beta = 0.5$
- $\epsilon = 10^{-6}$

Per iteration cost:

- $O(n^2)$: compute Hessian
- $O(n^3)$: solve linear system

Complexity of Newton's Method

Cost breakdown per iteration:

Operation	Cost	Comment
Compute gradient $\nabla f(x^k)$	$O(n)$ to $O(n^2)$	depends on f
Compute Hessian $\nabla^2 f(x^k)$	$O(n^2)$	storage + computation
Solve $\nabla^2 f(x^k)\Delta x = -\nabla f(x^k)$	$O(n^3)$	Cholesky factorization
Line search	$O(1)$	typically few steps

Main Bottleneck: $O(n^3)$ per iteration

- Manageable for n up to $\sim 10^4$
- Prohibitive for $n \sim 10^6 \Rightarrow$ use first-order methods
- Can exploit sparsity in Hessian for structured problems

Convergence of Newton's Method

Theorem (Two-Phase Convergence)

Suppose f is m-strongly convex, M-smooth, with H-Lipschitz Hessian. Then Newton with backtracking converges in two phases:

Phase 1 (Damped):

- When $\|\nabla f(x^k)\|$ is large
- **Linear convergence**
- Step size $\eta < 1$ (damped)

Iterations: $O\left(\frac{f(x^0)-f^*}{\gamma}\right)$
where γ depends on α, β, m, M

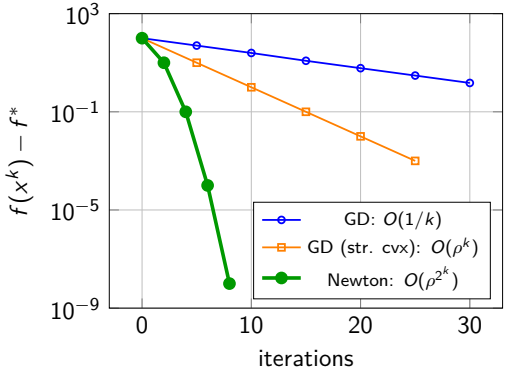
Phase 2 (Pure Newton):

- When $\|\nabla f(x^k)\|$ is small
- **Quadratic convergence**
- Step size $\eta = 1$ accepted

Iterations: $O(\log\log(1/\epsilon))$
Error halves *its exponent* each step!

Total: $O(1) + O(\log\log(1/\epsilon))$ iterations to reach ϵ-accuracy

Quadratic Convergence: Doubling Digits of Accuracy



Quadratic convergence:
Error at step $k+1$:

$$\epsilon_{k+1} \approx C \cdot \epsilon_k^2$$

Digits double each step:

$$\begin{aligned} \epsilon &= 10^{-2} \\ &\to 10^{-4} \\ &\to 10^{-8} \\ &\to 10^{-16} \end{aligned}$$

Machine precision in 3-4 Newton steps once in Phase 2!

amazon

Newton's Method is Affine Invariant

Key property: Newton's method is **independent of coordinate system**

Affine transformation: $y = Ax$ for invertible A, define $g(y) = f(A^{-1}y)$

Newton step in y-coordinates:

$$\begin{aligned}\Delta y &= -[\nabla^2 g(y)]^{-1}\nabla g(y) \\ &= -[A^{-T}\nabla^2 f(x)A^{-1}]^{-1}A^{-T}\nabla f(x) \\ &= A \cdot \underbrace{[-\nabla^2 f(x)]^{-1}\nabla f(x)}_{\Delta x} = A\Delta x\end{aligned}$$

Affine Invariance

Newton produces **identical iterates** (up to transformation) in any coordinate system!

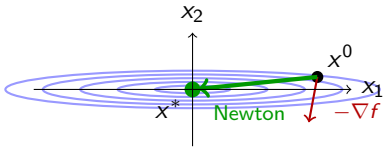
Contrast with GD: Gradient descent is **not** affine invariant — suffers from poor conditioning. Newton automatically adapts to the local geometry via the Hessian.

Example: Newton vs Gradient Descent

Problem: $f(x) = x_1^2 + 100x_2^2$ (condition number $\kappa = 100$)

Gradient and Hessian:

$$\nabla f = \begin{bmatrix} 2x_1 \\ 200x_2 \end{bmatrix}, \quad \nabla^2 f = \begin{bmatrix} 2 & 0 \\ 0 & 200 \end{bmatrix}$$



Newton step at $x^0 = [1, 1]^T$:

$$\Delta x = -\begin{bmatrix} 1/2 & 0 \\ 0 & 1/200 \end{bmatrix} \begin{bmatrix} 2 \\ 200 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

Hessian inverse rescales gradient to point directly at x^*

$\Rightarrow$ **One step to optimum!**

GD would need $O(\kappa) = O(100)$ iterations

Newton with Equality Constraints

Problem: minimize$_x$ $f(x)$ subject to $Ax = b$

Idea: Use KKT to find Newton step that is feasible (use Taylor approx.)

KKT system for Newton step:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \nu \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}$$

Variables:

- $\Delta x \in \mathbb{R}^n$: Newton step (primal)
- $\nu \in \mathbb{R}^p$: Lagrange multipliers (dual)

For feasibility we require:

$$\begin{cases} Ax^k = b \\ Ax^{k+1} = Ax^k + A\Delta x = b \end{cases} \Rightarrow A\Delta x = 0$$

Cost: Solve $(n + p) \times (n + p)$ linear system per iteration

Interior Point Methods: Motivation

Problem: $\min_x f(x) \quad \text{s.t.} \quad g_i(x) \leq 0, \quad i = 1, \ldots, m$

Reformulation with indicator:

$$\min_x \; f(x) + \sum_{i=1}^{m} I(g_i(x))$$

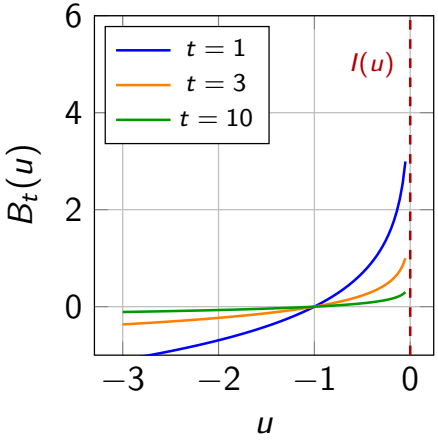
where $I(u) = \begin{cases} 0 & u \leq 0 \\ +\infty & u > 0 \end{cases}$

Problem: non-smooth, non-differentiable!

Idea: Approximate $I(.)$ with a smooth function:

$$B_t(u) = -\frac{1}{t}\log(-u)$$

✓ Smooth, convex, and $B_t(u) \to I(u)$ as $t \to \infty$



Barrier Problem Formulation

Original problem:

$$\underset{x}{\text{minimize}} \;\; f(x) \text{ subject to } g_i(x) \leq 0, \quad i = 1, \ldots, m$$

Barrier problem: (unconstrained!)

$$\underset{x}{\text{minimize}} \quad t \cdot f(x) + \phi(x)$$

where the **log barrier** is:

$$\phi(x) = -\sum_{i=1}^{m} \log(-g_i(x))$$

Properties:

- Domain: $\{x : g_i(x) < 0 \; \forall i\}$
- Barrier $\to \infty$ at boundary
- Solution approaches optimum as $t \to \infty$

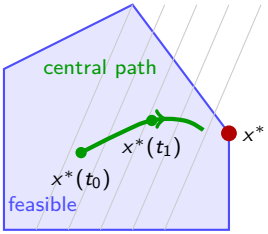
Key insight:

- Unconstrained problem!
- Can solve with Newton's method
- **Gradually increase** t

The Central Path

Definition: The **central path** is $\{x^*(t) : t > 0\}$ for increasing t where $x^*(t)$ solves:

$$\underset{x}{\text{minimize}} \quad t \cdot f(x) + \phi(x)$$



Properties:

- Path stays **strictly inside**
- As $t \to \infty$: $x^*(t) \to x^*$
- Smooth path (easy to follow)

Duality gap bound:

$$f(x^*(t)) - f^* \leq \frac{m}{t}$$

where $m = \#$ constraints

Double $t \Rightarrow$ halve the gap

The Barrier Method Algorithm

Algorithm 4 *

Barrier Method

1: **Input:** strictly feasible x^0, initial $t > 0$, growth factor $\mu > 1$, tolerance ϵ
2: **while** $m/t > \epsilon$ **do**
3: **Centering:** Starting from x, solve via Newton:

$$x \leftarrow \arg\min_x \{t \cdot f(x) + \phi(x)\}$$

4: **Increase:** $t \leftarrow \mu \cdot t$
5: **end while**
6: **return** x

Typical parameters:

- $\mu = 10$ to 20
- Initial t: moderate centering cost

Trade-off in μ:

- Large μ: fewer outer, more Newton
- Small μ: more outer, fewer Newton

Warm start: Previous $x^*(t)$ is excellent initial point for $x^*(\mu t)$

From KKT to Interior Point Methods

Original problem: $\min_x f(x) \quad \text{s.t.} \quad g_i(x) \leq 0, \quad i = 1, \ldots, m$

KKT conditions:

1. $\nabla f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(x^*) = 0$
2. $g_i(x^*) \leq 0$ (primal feas.)
3. $\lambda_i^* \geq 0$ (dual feas.)
4. $\lambda_i^* g_i(x^*) = 0$ (compl. slack.)

Problem: Condition 4 is **non-smooth** — either $\lambda_i = 0$ or $g_i = 0$, discrete choice!

Relaxed KKT conditions:

1. $\nabla f(x) + \sum_{i=1}^m \lambda_i \nabla g_i(x) = 0$
2. $g_i(x) < 0$ (strictly feas.)
3. $\lambda_i > 0$ (strictly dual feas.)
4. $\lambda_i(-g_i(x)) = 1/t$ (relaxed!)

✓ Condition 4 is now **smooth** — can solve with Newton's method!

As $t \to \infty$: relaxed condition $\lambda_i(-g_i) = 1/t \to 0$
$\Rightarrow$ recover exact complementary slackness!

Relaxed KKT $\Leftrightarrow$ Barrier Optimality

Step 1: Invert relaxed complementary slackness to get λ_i

$$\lambda_i(-g_i(x)) = \frac{1}{t} \qquad \Longrightarrow \qquad \boxed{\lambda_i = \frac{1}{-t \cdot g_i(x)}}$$

Step 2: Substitute into stationarity condition

$$\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) = 0 \qquad \Longrightarrow \qquad \nabla f(x) + \sum_{i=1}^{m} \frac{\nabla g_i(x)}{-t \cdot g_i(x)} = 0$$

Step 3: Multiply by t and recognize the gradient

$$t\nabla f(x) - \sum_{i=1}^{m} \frac{\nabla g_i(x)}{g_i(x)} = 0 \qquad \Longrightarrow \qquad \boxed{\nabla \left(t \cdot f(x) - \sum_{i=1}^{m} \log(-g_i(x)) \right) = 0}$$

Result: Relaxed KKT = optimality conditions for **barrier problem**!

$$\min_x \; t \cdot f(x) + \left(-\sum_{i=1}^{m} \log(-g_i(x)) \right)$$

amazon

IPM Complexity: Derivation

Barrier method: At outer iteration k, solve $\min_x \; t_k f(x) + \phi(x)$, where $\phi(x) = -\sum_{i=1}^{m} \log(-f_i(x))$

Step 1: Suboptimality from central path
The minimizer $x^*(t)$ satisfies (from duality):

$$f(x^*(t)) - f^* \leq \frac{m}{t}$$

So to achieve $f - f^* \leq \epsilon$, we need $t_{\text{final}} \geq m/\epsilon$.

Step 2: How many outer iterations to reach t_{final}?
We multiply t by $\mu > 1$ each iteration: $t_k = \mu^k t_0$. Need $\mu^k t_0 \geq m/\epsilon$:

$$k \;\geq\; \frac{\log(m/(\epsilon\, t_0))}{\log \mu}$$

Step 3: The tension — how large can μ be?

	Large μ	**Small μ**
Outer iterations	Few	Many
Centering (Newton)	Many steps (cold start)	Few steps (warm start)

The optimal μ balances these two effects $\longrightarrow$

IPM Complexity: The $\sqrt{m}$ Barrier

Step 4: Newton decrement after t-update
After multiplying $t \to \mu\, t$, the centering problem changes. One can show:

$$\lambda(x^*(t)) \text{ for the new problem at } \mu\, t \;\leq\; \frac{\mu - 1}{\mu} \cdot \sqrt{m}$$

(The barrier ϕ has m terms, each contributing $\sim 1/\sqrt{m}$ to the decrement shift.)
For Newton's method to converge in $O(1)$ steps (warm start), we need $\lambda \leq \beta$ for some constant $\beta < 1$:

$$\frac{\mu - 1}{\mu}\sqrt{m} \leq \beta \quad\Longrightarrow\quad \mu \;\leq\; \frac{1}{1 - \beta/\sqrt{m}} \;\approx\; 1 + \frac{\beta}{\sqrt{m}}$$

Step 5: Combine everything
With $\mu = 1 + \frac{c}{\sqrt{m}}$ and $\log(1 + c/\sqrt{m}) \approx c/\sqrt{m}$:

$$k \;=\; \frac{\log(m/\epsilon)}{\log \mu} \;\approx\; \frac{\log(m/\epsilon)}{c/\sqrt{m}} \;=\; \boxed{\frac{\sqrt{m}}{c}\log\left(\frac{m}{\epsilon}\right)}$$

Each outer iteration: $O(1)$ Newton steps $\times$ $O(n^3)$ per Newton step.

$$\text{Total: } \underbrace{O(\sqrt{m})}_{\text{outer iters}} \times \underbrace{O(1)}_{\substack{\text{Newton per}\\ \text{centering}}} \times \underbrace{O(n^3)}_{\substack{\text{per Newton}\\ \text{step}}} \times \underbrace{\log(m/\epsilon)}_{\text{accuracy}} = O(n^3\sqrt{m}\log(m/\epsilon))$$

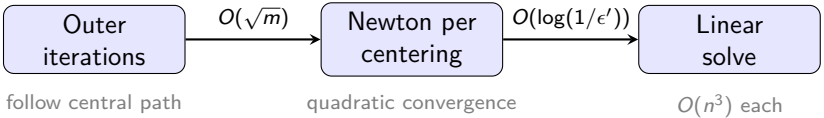
Insight: The $\sqrt{m}$ comes from how much the barrier shifts when t changes

Interior Point Method Complexity

Theorem (IPM Convergence)

The barrier method achieves $f(x) - f^* \leq \epsilon$ in $O\left(\sqrt{m}\log(m/\epsilon)\right)$ outer iterations, where $m =$ number of inequality constraints.

Complexity breakdown:



Total complexity:

$$O(n^3\sqrt{m}\log(m/\epsilon))$$

Polynomial in all parameters!

In practice:

- 20–50 Newton steps **total**
- Nearly independent of n, m
- Theoretically faster than simplex (in worst case)

Key: IPM solves LPs and convex programs in **polynomial time**

amazon

Summary: Second-Order Methods

Method	Problem	Per-iter	Convergence
Newton	Unconstrained	$O(n^3)$	Quadratic
Equality Newton	$Ax = b$	$O((n+p)^3)$	Quadratic
Barrier/IPM	Inequalities	$O(n^3)$	$O(\sqrt{m}\log(1/\epsilon))$

Use second-order when:

- Moderate dimension ($n \leq 10^4$)
- High accuracy needed
- Hessian available/cheap
- Inequality constraints (IPM)

Use first-order when:

- Large scale ($n > 10^6$)
- Low-moderate accuracy OK
- Special structure (sparsity)
- Hessian unavailable/expensive

References

[1] S. Boyd and L. Vandenberghe, "Convex Optimization", Ch. 9-11
https://web.stanford.edu/~boyd/cvxbook/

[2] Y. Nesterov and A. Nemirovskii, "Interior-Point Polynomial Algorithms in Convex Programming", SIAM, 1994

[3] J. Nocedal and S. Wright, "Numerical Optimization", Springer, 2006

[4] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming", Combinatorica, 1984