annotated version

Machine Learning Pre-Doc Summer School

Optimization

Jul 4+5, 2017

Martin Jaggi — http://mlo.epfl.ch

credits to Mohammad Emtiyaz Khan, Tao Lin, Frederik Kunstner, Anastasiia Koloskova

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

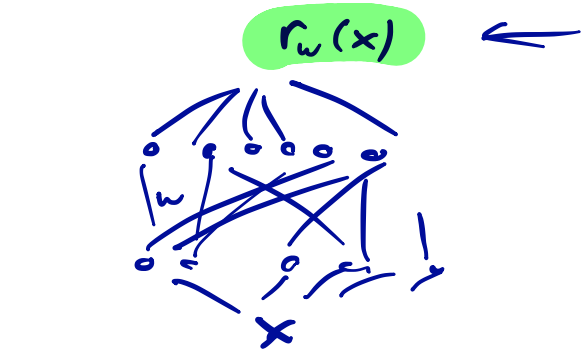
Learning / Estimation / Fitting

Given a cost function $f(\mathbf{w})$, we wish to find $\mathbf{w}^\star$ which minimizes the cost:

$$\min_{\mathbf{w}} f(\mathbf{w}) \quad \text{subject to} \quad \mathbf{w} \in \mathbb{R}^D$$

This means the *learning* problem is formulated as an optimization problem.

We will use an optimization algorithm to solve the problem (to find a good $\mathbf{w}$).



linear regr.

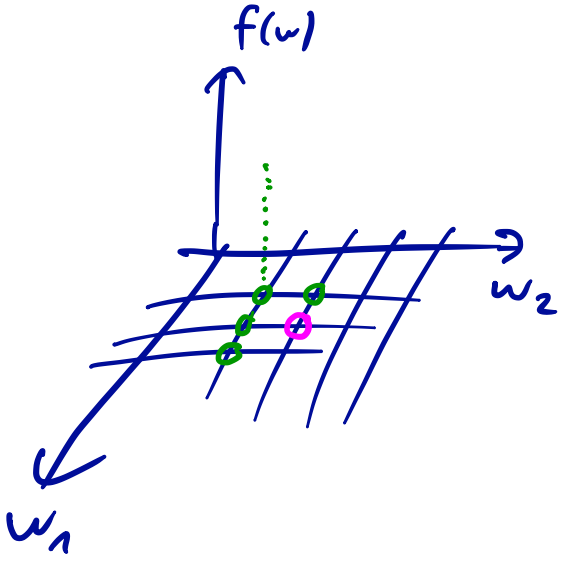
$r_w(x) = x^T w$

$f(w) = \frac{1}{N} \sum_{n=1}^{N} (y_n - r_w(x_n))^2$

Grid Search

Grid search is one of the simplest optimization algorithms. We compute the cost over all values $\mathbf{w}$ in a grid, and pick the best among those.

This is brute-force, but extremely simple and works for any kind of cost function when we have very few parameters and the cost is easy to compute.

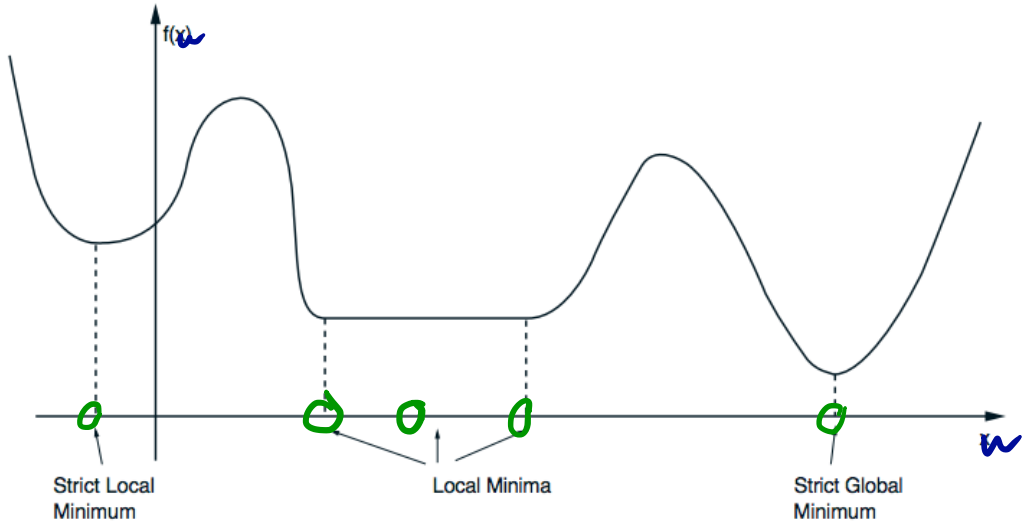


For a large number of parameters D, however, grid search has too many "for-loops", resulting in an exponential computational complexity:

If we decide to use 10 possible values for each dimension of $\mathbf{w}$, then we have to check 10^D points. This is clearly impossible for most practical machine learning models, which can often have $D \approx$ millions of parameters. Choosing a good range of values for each dimension is another problem.

Other issues: No guarantee can be given that we and up close to an optimum.

Optimization Landscapes



The above figure is taken from Bertsekas, Nonlinear programming.

A vector $\mathbf{w}^\star$ is a local minimum of f if it is no worse than its neighbors; i.e. there exists an $\epsilon > 0$ such that,

$$f(\mathbf{w}^\star) \leq f(\mathbf{w}), \quad \forall \mathbf{w} \text{ with } \|\mathbf{w} - \mathbf{w}^\star\| < \epsilon$$

A vector $\mathbf{w}^\star$ is a global minimum of f if it is no worse than all others,

$$f(\mathbf{w}^\star) \leq f(\mathbf{w}), \quad \forall \mathbf{w} \in \mathbb{R}^D$$

A local or global minimum is said to be strict if the corresponding inequality is strict for $\mathbf{w} \neq \mathbf{w}^\star$.

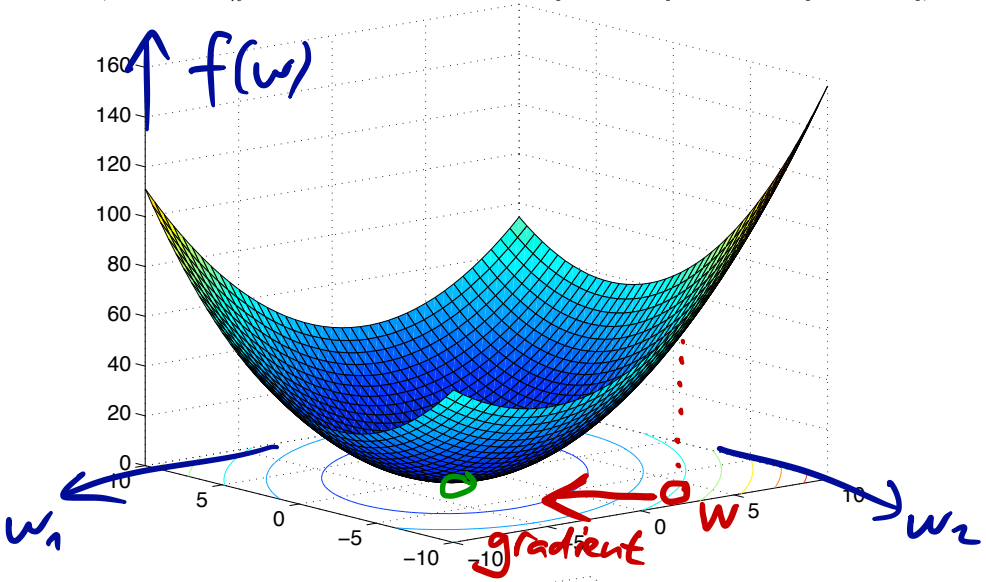
Differentiable

Smooth Optimization

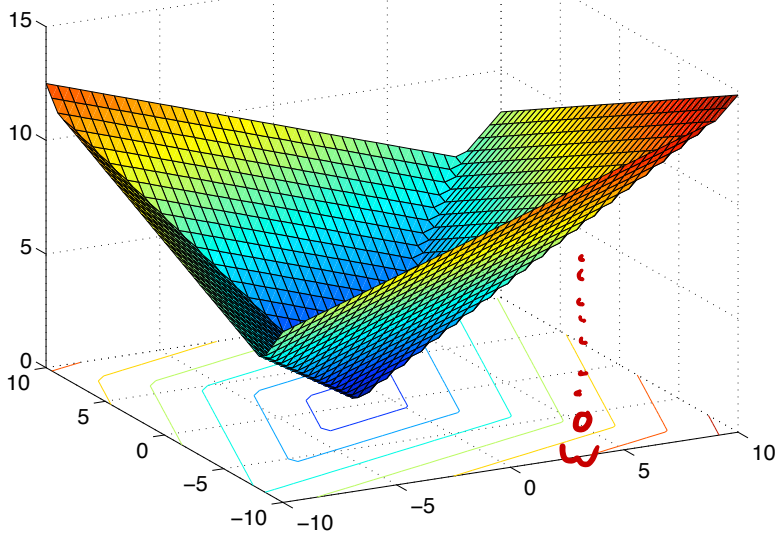
Follow the Gradient

A gradient (at a point) is the slope of the tangent to the function (at that point). It points to the direction of largest increase of the function.

For a 2-parameter model, MSE($\mathbf{w}$) and MAE($\mathbf{w}$) are shown below.
(We used $\mathbf{y}_n \approx w_0 + w_1 x_{n1}$ with $\mathbf{y}^\top = [2, -1, 1.5]$ and $\mathbf{x}^\top = [-1, 1, -1]$).



mean squared error



mean absolute error

$f(w) = \sum_n |y_n - r_w(x_n)|$

Definition of the gradient:

how fast does f change in coordinate direction

$$\nabla f(\mathbf{w}) := \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_D}\right]^\top$$

This is a vector, $\nabla f(\mathbf{w}) \in \mathbb{R}^D$.

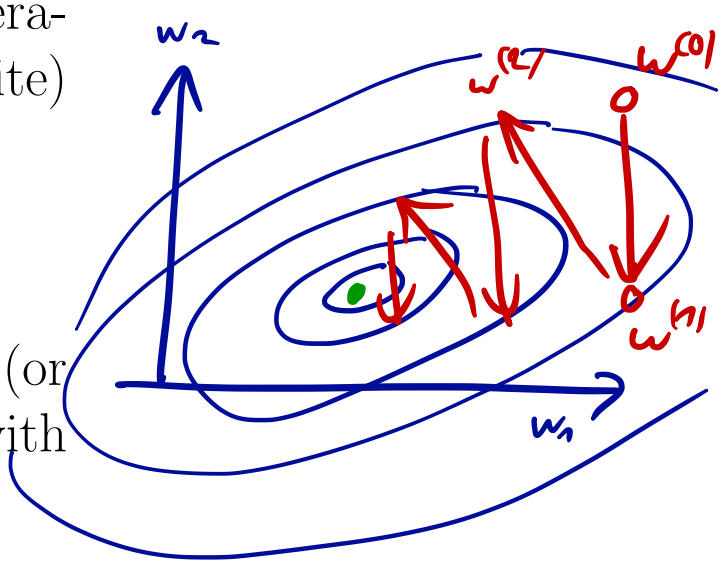
Gradient Descent (Batch)

start at $\mathbf{w}^{(0)}$

To minimize the function, we iteratively take a step in the (opposite) direction of the gradient

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla f(\mathbf{w}^{(t)})$$

where $\gamma > 0$ is the step-size (or learning rate). Then repeat with the next t.



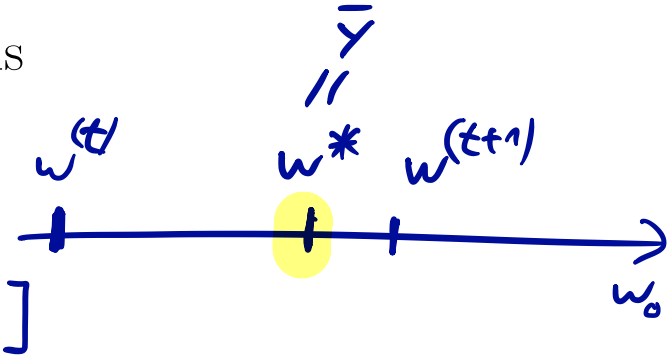
Example: Gradient descent for 1-parameter model to minimize MSE:

linear regression

$$f(w) = \frac{1}{N}\sum_{n=1}^{N}(y_n - w_0)^2$$

$$w_0^{(t+1)} := (1-\gamma)w_0^{(t)} + \gamma\bar{y}$$

Where $\bar{y} = \sum_n y_n/N$. When is this sequence guaranteed to converge?



stepsize $\gamma \in (0, 1]$

$r_w(x) = w_0 + \cancel{w_1 x + \ldots x^p}$

Gradient Descent for Linear MSE

For linear regression

features

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$$

$\in$ data example 1

$\in$ data example N

We define the error vector $\mathbf{e}$:

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w} \qquad \in \mathbb{R}^N$$

and MSE as follows:

$$f(\mathbf{w}) := \frac{1}{2N} \sum_{n=1}^{N} \underbrace{\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2}_{f_n(\mathbf{w})} = \frac{1}{2N} \mathbf{e}^\top \mathbf{e}$$

$$\frac{\partial}{\partial w_i} f \Big|_{\mathbf{w}}$$

then the gradient is given by

$$\nabla f(\mathbf{w}) = -\frac{1}{N} \mathbf{X}^\top \mathbf{e} \qquad \in \mathbb{R}^D$$

$$\nabla f_n(\mathbf{w}) = (y_n - \mathbf{x}_n^\top \mathbf{w})(-\mathbf{x}_n) \qquad \in \mathbb{R}^D$$

Computational cost. What is the complexity (# operations) of computing the gradient?

a) starting from $\mathbf{w}$ and

b) given $\mathbf{e}$ and $\mathbf{w}$?

compute $\mathbf{e}$

$N \cdot D + N \; + \; N \cdot D + D$

$= O(N \cdot D)$

$N \cdot D + D$

$= O(N \cdot D)$

Variant with offset. Recall: Alternative trick when also incorporating an offset term for the regression:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$$

constant feature

$r = w_0 + w_1 x_1 + \dots + w_D x_D = \tilde{x}^T w$

Stochastic Gradient Descent

Sum Objectives. In machine learning, most cost functions are formulated as a sum over the training examples, that is

$$f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} f_n(\mathbf{w}) \ ,$$

f_n: fit for datapoint n

where f_n is the cost contributed by the n-th training example.

Q: What are the f_n for linear MSE?

The SGD Algorithm. The stochastic gradient descent (SGD) algorithm is given by the following update rule, at step t:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla f_n(\mathbf{w}^{(t)}) \ .$$

$1 \le n \le N$

n chosen uniformly at random

Theoretical Motivation. *Idea:* Cheap but unbiased estimate of the gradient!
In expectation over the random choice of n, we have

$$\mathbb{E}_n[\nabla f_n(\mathbf{w})] = \nabla f(\mathbf{w})$$

which is the true gradient direction. (check!)

Mini-batch SGD. There is an intermediate version, using the update direction being

$$\mathbf{g} := \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\mathbf{w}^{(t)})$$

again with

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, \mathbf{g} \; .$$

In the above gradient computation, we have randomly chosen a subset $B \subseteq [N]$ of the training examples. For each of these selected examples n, we compute the respective gradient ∇f_n, at the same current point $\mathbf{w}^{(t)}$.

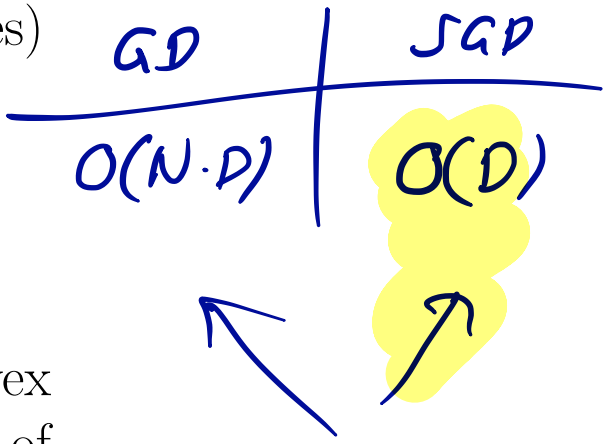
The computation of $\mathbf{g}$ can be parallelized easily. This is how current deep-learning applications utilize GPUs (by running over $|B|$ threads in parallel).
Note that in the extreme case $B := [N]$, we obtain (batch) gradient descent, i.e. $\mathbf{g} = \nabla f$.

SGD for Linear MSE

Exercise!

Computational cost. For linear MSE, what is the complexity (# operations) of computing the stochastic gradient?
(using only $|B| = 1$ data examples)

a) starting from $\mathbf{w}$?



Convergence Theory

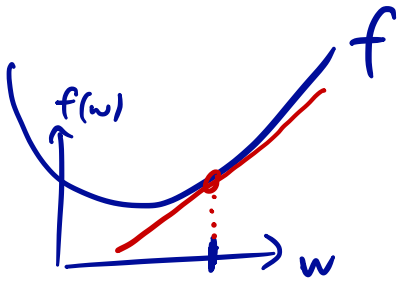
Theorem: For smooth convex problems, the convergence rate of (stochastic) gradient descent is

$$f(\mathbf{x}^{(t)}) - f(\mathbf{x}^\star) \leq \mathcal{O}(1/t)$$

(where $\mathbf{x}^\star$ is some optimal solution to the problem.)

caveat: SGD rate can be $1/\sqrt{t}$ if f is not strongly convex

Not differentiable

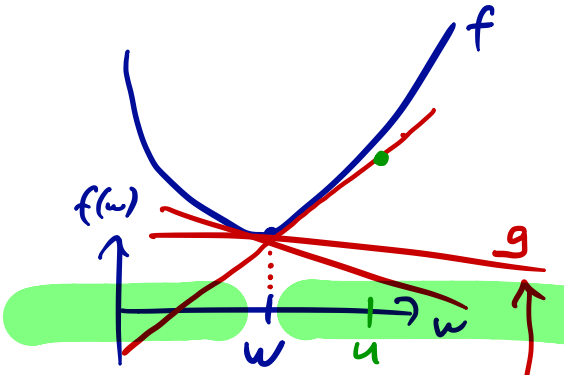


Non-Smooth Optimization

An alternative characterization of *convexity*, for differentiable functions is given by

$$f(\mathbf{u}) \geq f(\mathbf{w}) + \nabla f(\mathbf{w})^\top (\mathbf{u} - \mathbf{w}) \quad \forall \mathbf{u}, \mathbf{w}$$

meaning that the function must always lie above its linearization.

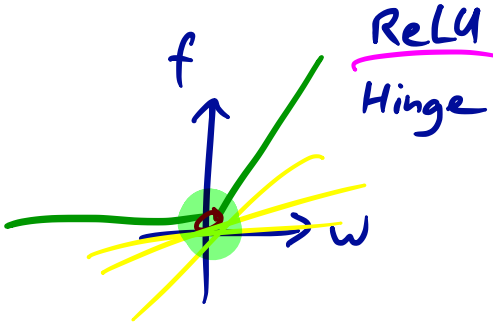


Subgradients

A vector $\mathbf{g} \in \mathbb{R}^D$ such that

$$f(\mathbf{u}) \geq f(\mathbf{w}) + \mathbf{g}^\top (\mathbf{u} - \mathbf{w}) \quad \forall \mathbf{u}$$

is called a subgradient to the function f at $\mathbf{w}$.



This definition makes sense for objectives f which are not necessarily differentiable (and not even necessarily convex).

If f is differentiable at $\mathbf{w}$, then the only subgradient at $\mathbf{w}$ is $\mathbf{g} = \nabla f(\mathbf{w})$.

Subgradient Descent

Identical to the gradient descent algorithm, but using a subgradient instead of gradient. Update rule

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \mathbf{g}$$

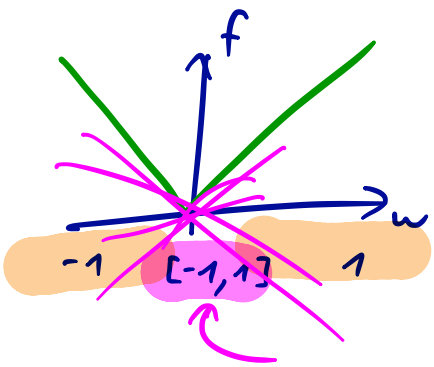
before: $\nabla f(w)$

for $\mathbf{g}$ being a subgradient to f at the current iterate $\mathbf{w}^{(t)}$.

Example: Optimizing Linear MAE

1. Compute a subgradient of the absolute value function

   $h : \mathbb{R} \to \mathbb{R}$, $h(e) := |e|$.



2. Recall the definition of the mean absolute error:

   $$f(\mathbf{w}) = \text{MAE}(\mathbf{w}) := \frac{1}{N} \sum_{n=1}^{N} |y_n - r(\mathbf{x}_n)|$$

   For linear regression, its (sub)gradient is easy to compute using the chain rule. Compute it!

(Exercise)

(convex) not differentiable

$$f(w) = \frac{1}{N}\sum_{n=1}^{N} f_n(w)$$

Stochastic Subgradient Descent

Stochastic SubGradient Descent (still abbreviated SGD commonly).

Same, $\mathbf{g}$ being a subgradient to the randomly selected f_n at the current iterate $\mathbf{w}^{(t)}$.

$$w^{(t+1)} := w^{(t)} - \gamma\, g_n$$

g_n: subgradient of $f_n(w^{(t)})$

Exercise: Compute the SGD update for linear MAE.

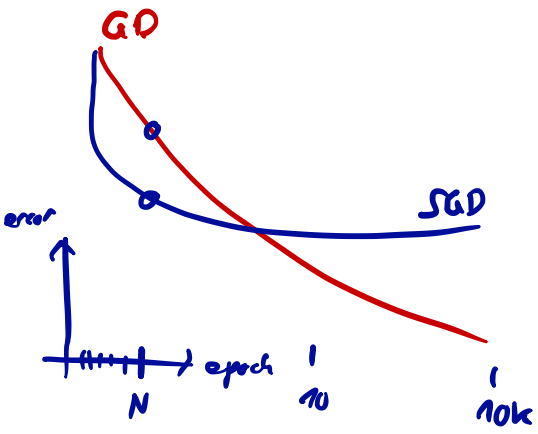
$$= \frac{1}{N}\sum_{n} |y_n - w^T x_n|$$

Convergence Theory

Theorem: For non-smooth convex problems, the convergence rate of (stochastic) subgradient descent is

$$f(\mathbf{x}^{(t)}) - f(\mathbf{x}^\star) \leq \mathcal{O}(1/\sqrt{t})$$

(where $\mathbf{x}^\star$ is some optimal solution to the problem.)



complexity	smooth (MSE)	non-smooth (MAE)
GD (sub)	$O(N \cdot D)$	$O(N \cdot D)$
10k		
SGD (sub)	$O(D)$	$O(D)$
10k		

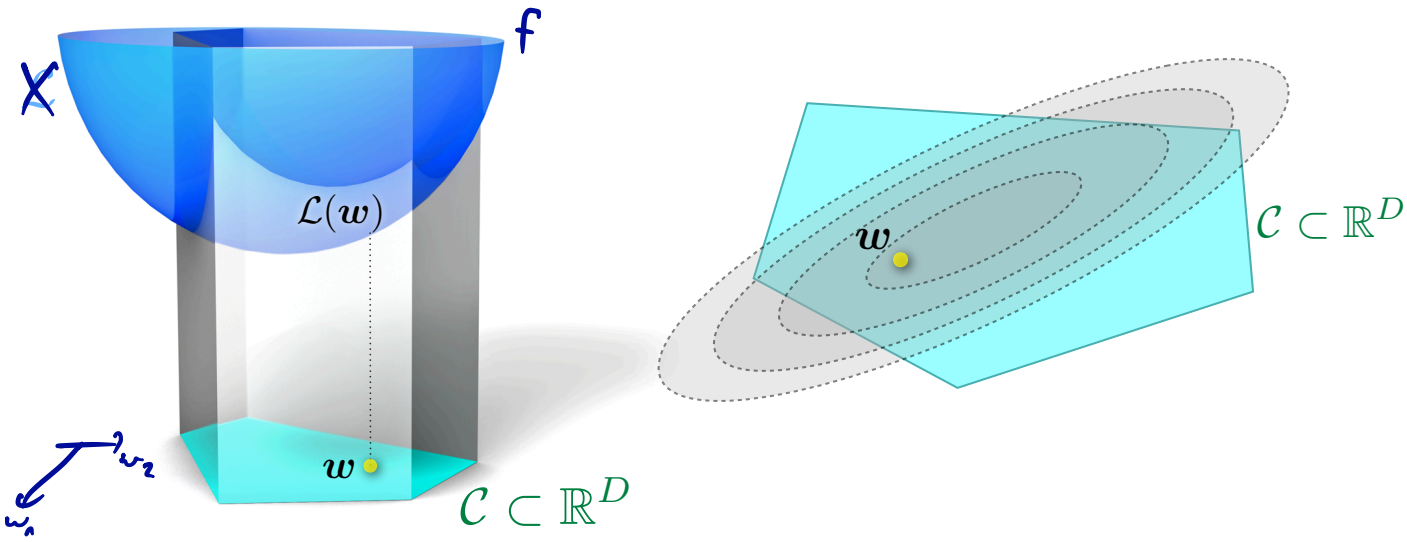
Constrained Optimization

Sometimes, optimization problems come posed with additional constraints:

$$\min_{\mathbf{w}} f(\mathbf{w}), \quad \text{subject to } \mathbf{w} \in \mathcal{C}.$$

examples:
$\|w\|_2 \leq r$
$w \geq 0$
$w_5 = 0$

The set $\mathcal{C} \subset \mathbb{R}^D$ is called the constraint set.



Solving Constrained Optimization Problems

A) Projected Gradient Descent

B) Transform it into an *unconstrained* problem

Convex Sets

A set $\mathcal{C}$ is convex *iff*
the line segment between any two points of $\mathcal{C}$ lies in $\mathcal{C}$, i.e., if for any $\mathbf{u}, \mathbf{v} \in \mathcal{C}$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta\mathbf{u} + (1-\theta)\mathbf{v} \in \mathcal{C}.$$

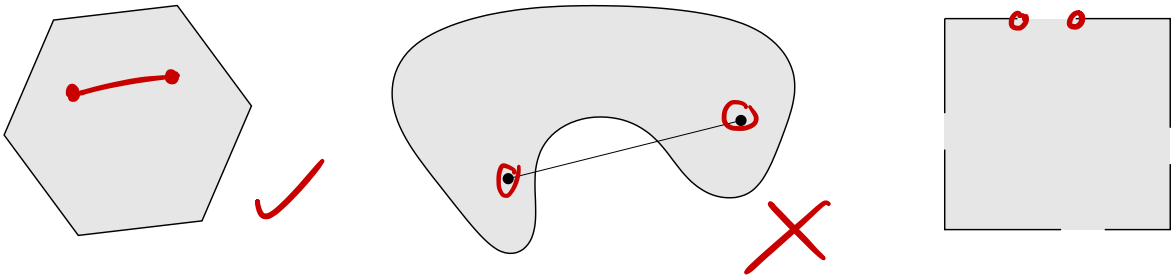
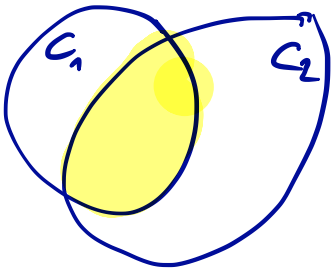


*Figure 2.2 from S. Boyd, L. Vandenberghe

Properties of Convex Sets



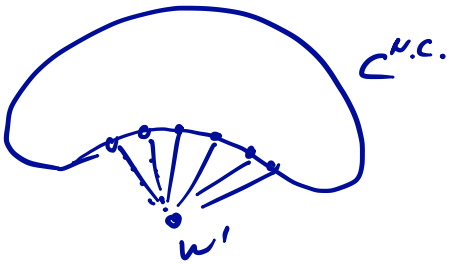
- Intersections of convex sets are convex
- Projections onto convex sets are *unique*.
  (and often efficient to compute)
  Formal definition:
  $P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v}\in\mathcal{C}} \|\mathbf{v} - \mathbf{w}'\|$.

(A)

Projected Gradient Descent

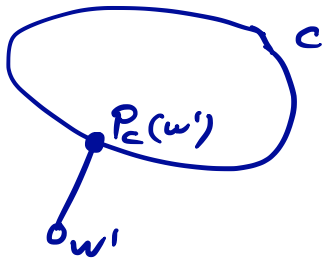
Idea: add a projection onto $\mathcal{C}$ after every step:

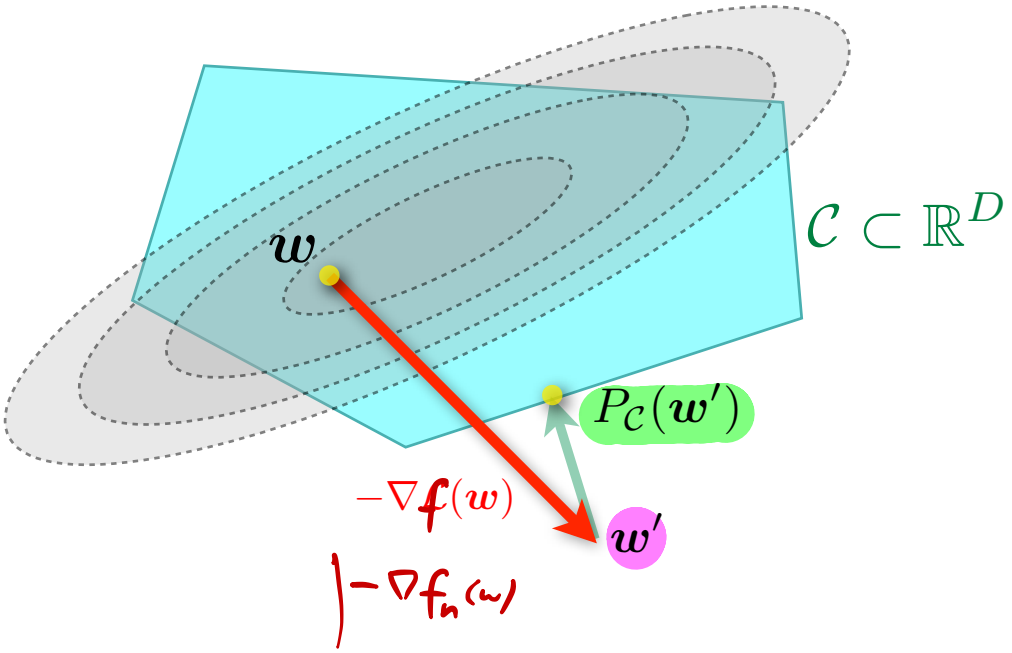
$$P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v}\in\mathcal{C}} \|\mathbf{v} - \mathbf{w}'\|_2 .$$



Update rule:

$$\mathbf{w}^{(t+1)} := P_{\mathcal{C}}\left[\mathbf{w}^{(t)} - \gamma \nabla f(\mathbf{w}^{(t)})\right] .$$





Projected SGD: Same SGD step, followed by the projection step, as above. Same convergence properties.

$\|w\|_2 \leq R$

Computational cost of projection? Crucial!

w'

$O(D)$

$\|w\|_1 \leq R$

B

Turning Constrained into Unconstrained Problems

(Alternatives to projected gradient methods)

Use penalty functions instead of directly solving $\min_{\mathbf{w}\in\mathcal{C}} f(\mathbf{w})$.

- "*brick wall*" (indicator function)

$$I_{\mathcal{C}}(\mathbf{w}) := \begin{cases} 0 & \mathbf{w} \in \mathcal{C} \\ \infty & \mathbf{w} \notin \mathcal{C} \end{cases}$$

$$\Rightarrow \min_{\mathbf{w}\in\mathbb{R}^D} f(\mathbf{w}) + I_{\mathcal{C}}(\mathbf{w})$$

(disadvantage: non-continuous objective)

- Penalize error. *Example:* $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^D \mid A\mathbf{w} = \mathbf{b}\}$

$$\Rightarrow \min_{\mathbf{w}\in\mathbb{R}^D} f(\mathbf{w}) + \lambda \|A\mathbf{w}-\mathbf{b}\|^2$$

- Linearized Penalty Functions (see Lagrange Multipliers)

$$\max_{\mathbf{z}\in\mathbb{R}^M} \min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{f(\mathbf{w}) + \mathbf{z}^T(A\mathbf{w}-\mathbf{b})}_{\mathcal{L}(\mathbf{w},\mathbf{z})}$$

M constraints $=0$

Advanced Methods

- Coordinate Descent
- Frank Wolfe
- Newton's Method

$\min_w f(w)$ s.t. $w_i \in C_i \ \forall i$

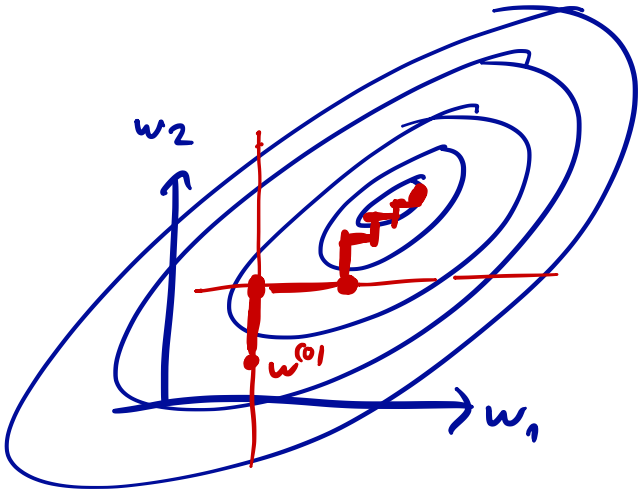
$w_i^{(t+1)} := \arg\min_{w_i} f(w_1^{(t)}, \dots, w_i, \dots, w_D^{(t)})$

i randomly

Block-Coordinate Descent

$\nabla^2 f \; (-\nabla f)$

$D \times D \quad D$

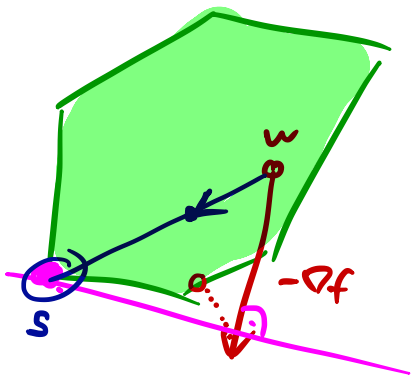


Ridge Regression

$\|Aw - y\|_2 + \lambda \|w\|_2^2$

Lasso .. $+ \lambda \|w\|_1^2$

Frank-Wolfe



$s = \arg\min_{s \in C} s^T \nabla f$

$w^{(t+1)} := (1-\gamma) w^{(t)} + \gamma s$

Implementation Issues

For Gradient Methods:

$0 = \nabla f = \sum_{n} \nabla f_n$

$\neq 0$

Stopping criteria: When $\nabla f(\mathbf{w})$ is (close to) zero, we are (often) close to the optimum.

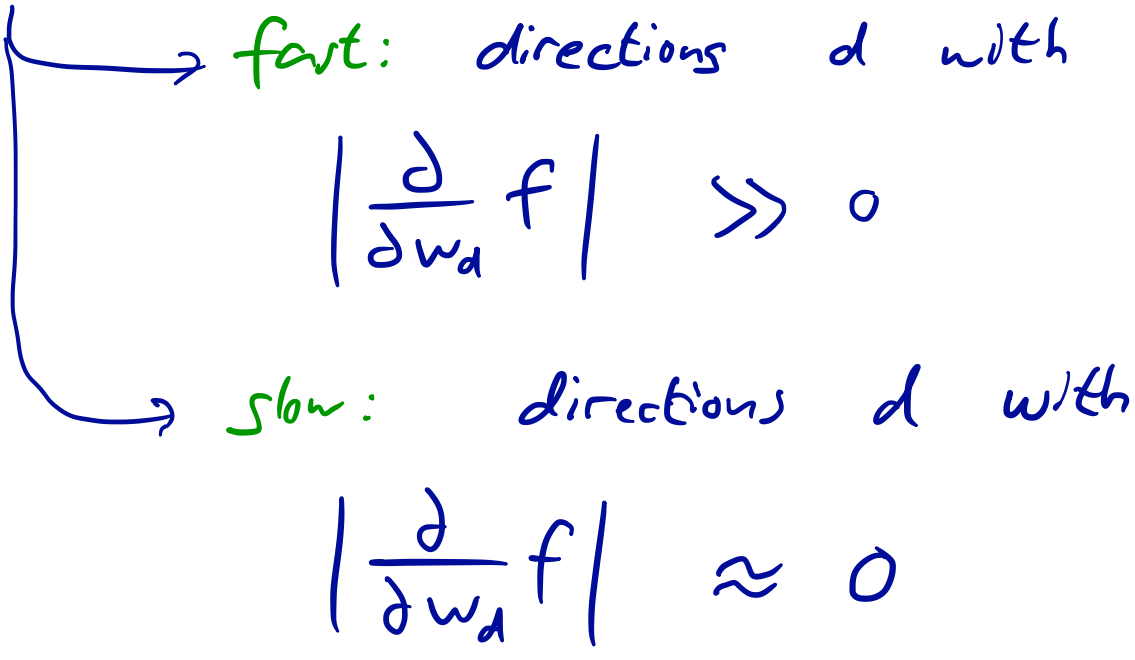
$f(w^{(t)}) - f(w^*) \leq \varepsilon$

Optimality: If the second-order derivative is positive (positive semi-definite to be precise), then it is a (possibly local) minimum. If the function is also convex, then this condition implies that we are at a global optimum. See the supplementary section on Optimality Conditions.

Step-size selection: If γ is too big, the method might diverge. If it is too small, convergence is slow. Convergence to a local minimum is guaranteed only when $\gamma < \gamma_{min}$ where γ_{min} is a fixed constant that depends on the problem.

Line-search methods: For some objectives f, we can set step-size automatically using a line-search method. More details on "backtracking" methods can be found in Chapter 1 of Bertsekas' book on "nonlinear programming".

Feature normalization and pre-conditioning: Gradient descent is very sensitive to ill-conditioning. Therefore, it is typically advised to normalize your input features. In other words, we pre-condition the optimization problem. Without this, step-size selection is more difficult since different "directions" might converge at different speed.



Non-Convex Optimization

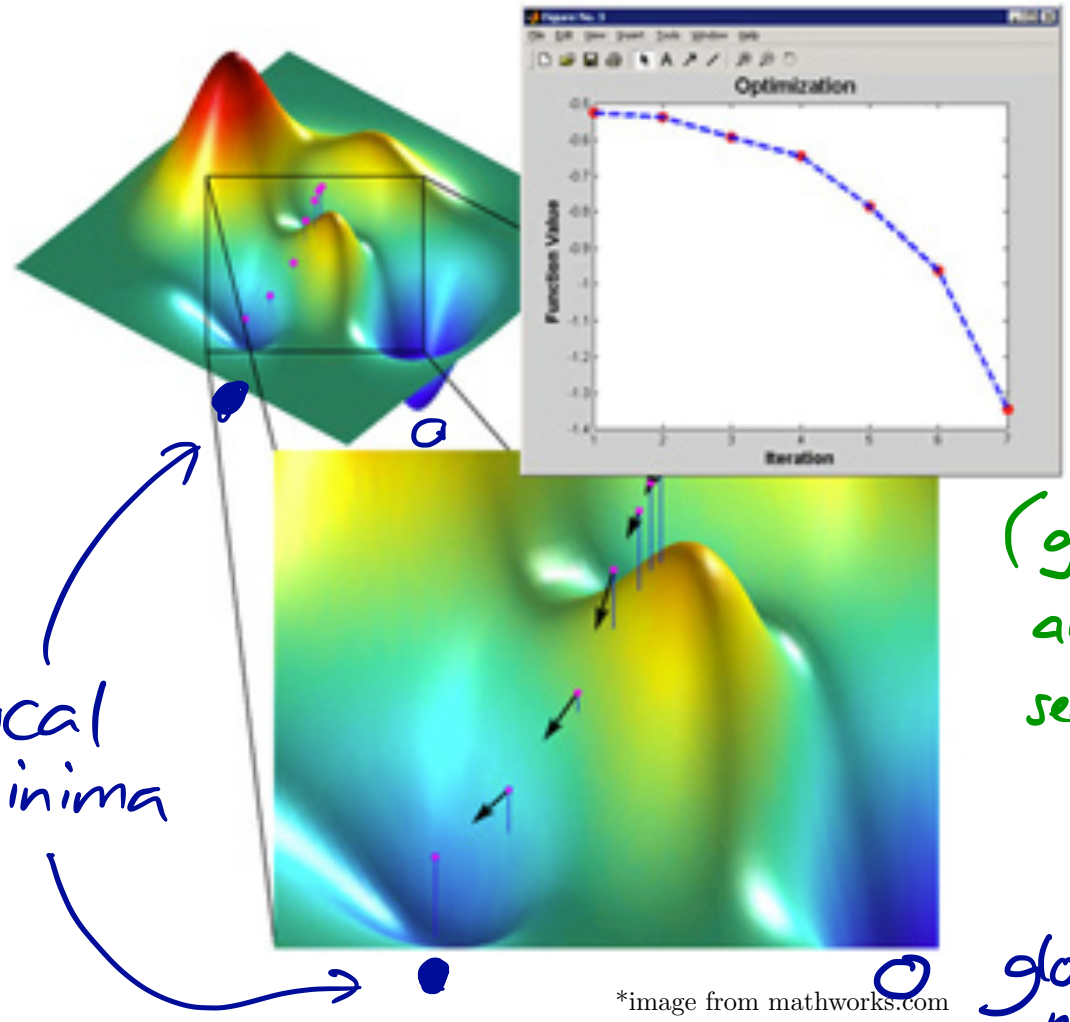


*image from mathworks.com

GD can get stuck in local minima

(grid search and random search doesn't)

Local minima

global minimum

Real-world problems are not convex!

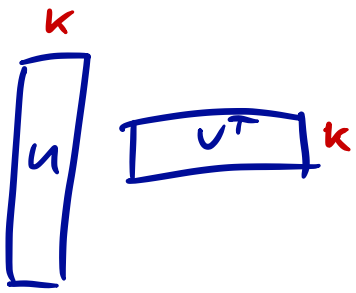
All we have learnt on algorithm design and performance of convex algorithms still helps us in the non-convex world.

Matrix Factorizations

General Formulation

$$\begin{aligned} \min_{\mathbf{U},\mathbf{V}} \quad & f(\mathbf{U},\mathbf{V}) \\ \text{s.t.} \quad & \mathbf{U} \in \mathbb{R}^{D\times K} \\ & \mathbf{V} \in \mathbb{R}^{N\times K} \end{aligned}$$

and assume $f(\mathbf{U},\mathbf{V}) = h(\mathbf{U}\mathbf{V}^\top)$ for some function $h : \mathbb{R}^{D\times N} \to \mathbb{R}$.



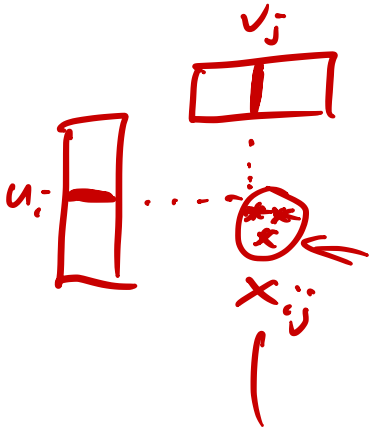
Examples:

- $f(\mathbf{U},\mathbf{V}) := \frac{1}{2}\|\mathbf{X} - \mathbf{U}\mathbf{V}^\top\|_F^2$.

  Has an explicit solution: Singular Value Decomposition (first K singular vector pairs)

  Unfortunately, this case is *a rare exception*!

- *Matrix Completion & Word Embeddings*



$$f(\mathbf{U},\mathbf{V}) := \frac{1}{|\Omega|}\sum_{(d,n)\in\Omega}\left[\mathbf{X}_{dn} - (\mathbf{U}\mathbf{V}^\top)_{dn}\right]^2$$

where Ω is the set of observed ratings

Matrix Factorizations are Typically Non-Convex

Even if we are given a *convex* objective function

$$h(\mathbf{W}) \quad : \quad \mathbb{R}^{D\times N} \to \mathbb{R},$$

the same objective function in its factorized form

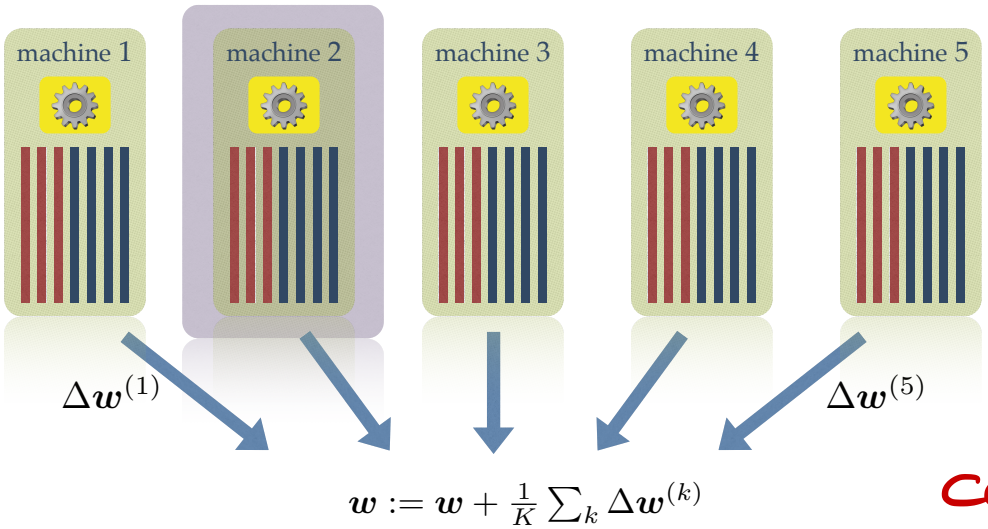
$$f(\mathbf{U},\mathbf{V}) := h(\mathbf{U}\mathbf{V}^\top) \quad : \quad \mathbb{R}^{(D+N)\times K} \to \mathbb{R}$$

is typically *not convex* (in its complete argument $(\mathbf{U},\mathbf{V})$).

Proof:
Identity function $h(w) := w$, and $D = N = 1$. The resulting objective $f(u,z) = uz$ is a saddle function over its two variables.

Applications and Research Directions

- Convergence Theory for (some) Non-Convex Problems
- Training Error vs. Generalization
- Distributed and Parallel Training



- Efficient Accelerated Methods
  $\mathcal{O}(1/t) \Rightarrow \mathcal{O}(1/t^2)$
- Efficient Second-Order Methods
- Importance Sampling

Additional Notes

Convex Functions

A function $f : \mathbb{R}^D \to \mathbb{R}$ is convex if $dom\, f$ is a convex set and if for all $\mathbf{x}, \mathbf{y} \in dom\, f$, and θ with $0 \leq \theta \leq 1$, we have

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y}).$$

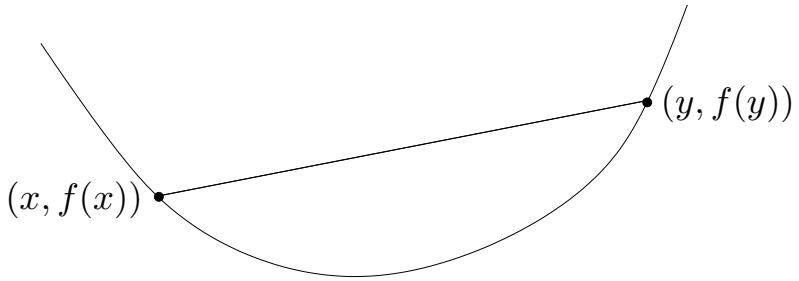


*Figure 3.1 from S. Boyd, L. Vandenberghe

Geometrically: The line segment between $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ lies above the graph of f.

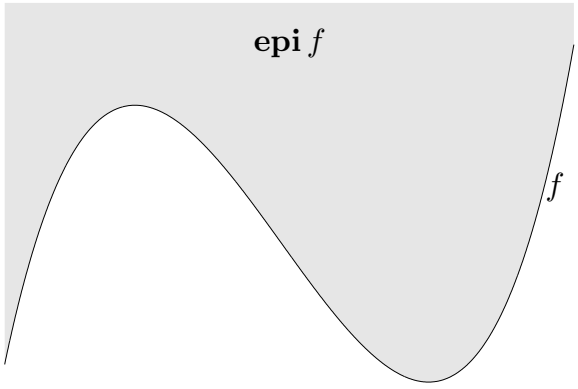
Epigraph: The *graph* of a function $f : \mathbb{R}^D \to \mathbb{R}$ is defined as

$$\{(\mathbf{x}, f(\mathbf{x})) \,|\, \mathbf{x} \in dom\, f\},$$

The *epigraph* of a function $f : \mathbb{R}^D \to \mathbb{R}$ is defined as

$$\{(\mathbf{x}, t) \,|\, \mathbf{x} \in dom\, f, f(\mathbf{x}) \leq t\},$$

A function is convex *iff* its epigraph is a convex set.



Convex?

*Figure 3.5 from S. Boyd, L. Vandenberghe

Examples of convex functions

- Linear functions: $f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x}$
- Affine functions: $f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} + b$
- Exponential: $f(\mathbf{x}) = e^{\alpha \mathbf{x}}$
- Norms. Every norm on $\mathbb{R}^D$ is convex.

Convexity of a norm $f(\mathbf{x})$
By the triangle inequality $f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ and homogeneity of a norm $f(a\mathbf{x}) = |a| f(\mathbf{x})$, a scalar:

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq f(\theta\mathbf{x}) + f((1-\theta)\mathbf{y}) = \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}).$$

We used the triangle inequality for the inequality and homogeneity for the equality.

Solving Convex Optimization Problems.

Convex Optimization Problems are of the form

$$\min \ f(\mathbf{x}) \qquad \text{s.t.} \qquad \mathbf{x} \in Q$$

where both

- f is a convex function
- Q is a convex set (note: $\mathbb{R}^D$ is convex)

Properties of Convex Optimization Problems

- Every local minimum is a *global minimum*

Optimality Conditions

For a *convex* optimization problem, the first-order *necessary* condition says that at *an* optimum the gradient is equal to zero.

$$\nabla f(\mathbf{w}^\star) = \mathbf{0} \tag{1}$$

The second-order *sufficient* condition ensures that the optimum is a minimum (not a maximum or saddle-point) using the Hessian matrix, which is the matrix of second derivatives:

$$\mathbf{H}(\mathbf{w}^\star) := \frac{\partial^2 f(\mathbf{w}^\star)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \quad \text{is positive semi-definite.} \tag{2}$$

The Hessian is also related to the convexity of a function: a twice-differentiable function is convex if and only if the Hessian is positive semi-definite at all points.

SGD Theory

As we have seen above, when N is large, choosing a random training example $(\mathbf{x}_n, y_n)$ and taking an SGD step is advantageous:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \gamma^{(t)} \nabla f_n(\mathbf{w}^{(t)})$$

For convergence, $\gamma^{(t)} \to 0$ "appropriately". One such condition called the Robbins-Monroe condition suggests to take $\gamma^{(t)}$ such that:

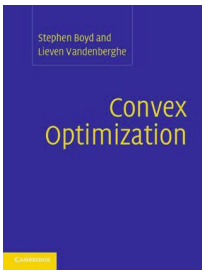
$$\sum_{t=1}^{\infty} \gamma^{(t)} = \infty, \qquad \sum_{t=1}^{\infty} (\gamma^{(t)})^2 < \infty \tag{3}$$

One way to obtain such sequences is $\gamma^{(t)} := 1/(t+1)^r$ where $r \in (0.5, 1)$.

More Optimization Theory

If you want, you can gain a deeper understanding of several optimization methods relevant for machine learning from this survey:

Convex Optimization: Algorithms and Complexity - by Sébastien Bubeck



(> 35 000 citations)

And also from the book of Boyd & Vandenberghe (both are free online PDFs)

Exercises

1. Follow the practical exercises described in the `handout.pdf`

2. Chain-rule

   If it has been a while, familiarize yourself with it again.

3. Derive the computational complexity of grid-search, gradient descent and stochastic gradient descent for linear MSE (# steps and cost per step).

4. Derive the gradients for the linear MSE and MAE cost functions.