**Machine Learning Course - CS-433**

# Optimization

September 10+16, 2025

Robert West
Last updated on: September 9, 2025

credits to Martin Jaggi, Mohammad Emtiyaz Khan

## Learning / Estimation / Fitting

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

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

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 $\mathbf{w}$).

## 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.

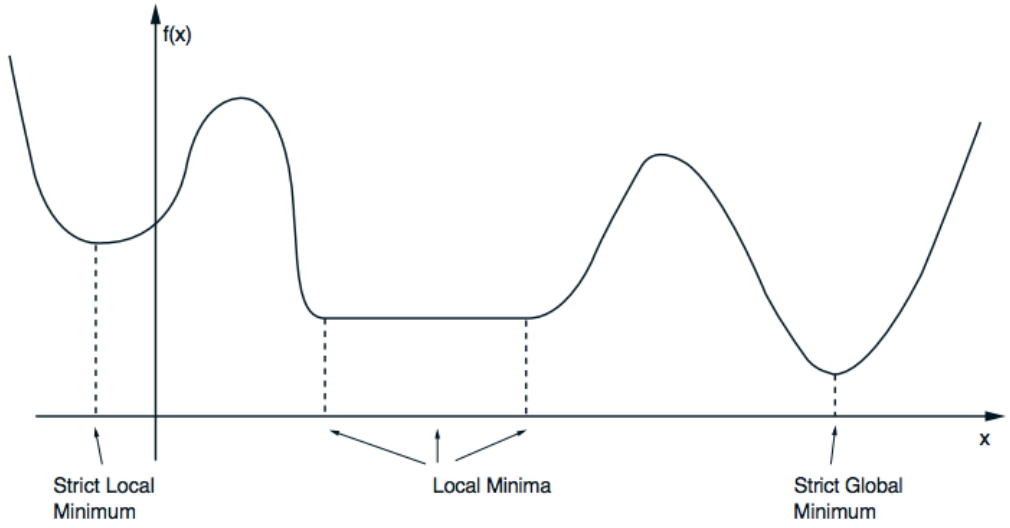
This is brute-force, but extremely simple and works for any kind of cost function when we have very few parameters and the cost is easy to compute.

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 end 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 $\mathcal{L}$ if it is no worse than its neighbors; i.e. there exists an $\epsilon > 0$ such that,

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

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

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

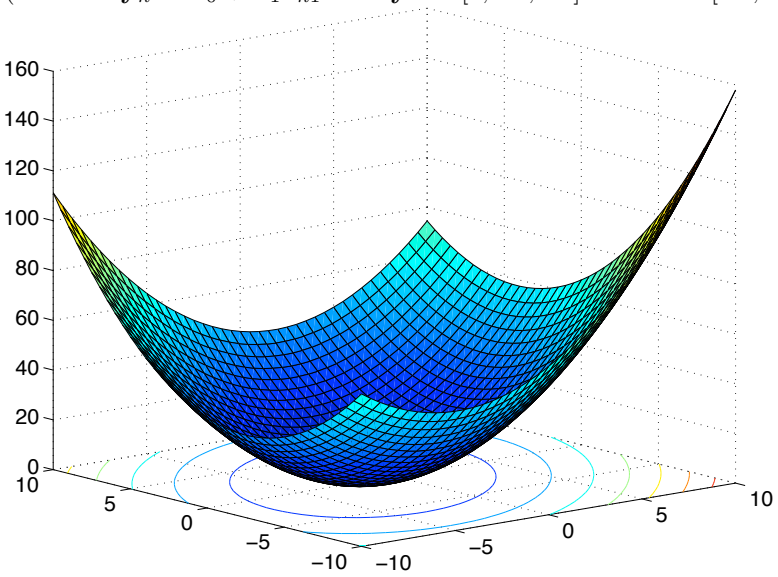
A local or global minimum is said to be strict if the corresponding inequality is strict for $\mathbf{w} \neq \mathbf{w}^\star$.

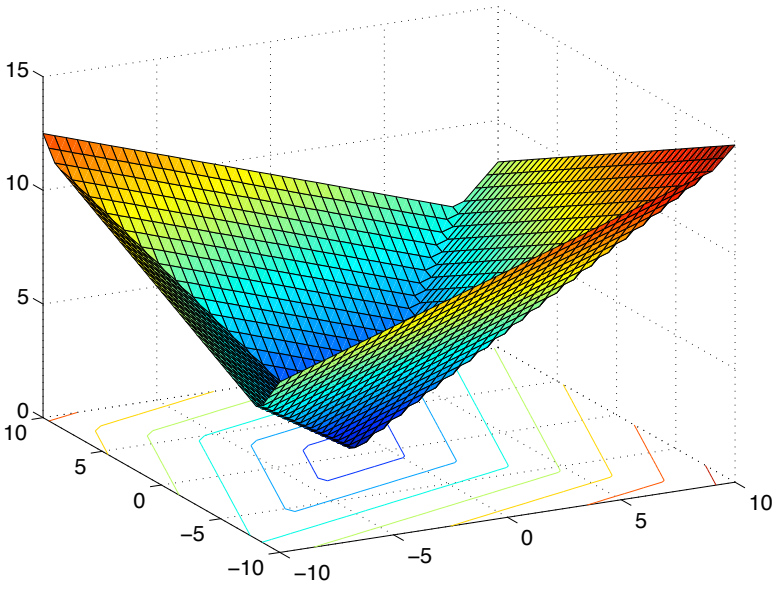
# Smooth Optimization

## 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 $\mathbf{x}^\top = [-1, 1, -1]$).



Definition of the gradient:

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

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

## Gradient Descent

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

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

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:

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

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

## Gradient Descent for Linear MSE

For linear regression

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

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

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

and MSE as follows:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &:= \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 \\ &= \tfrac{1}{2N} \mathbf{e}^\top \mathbf{e} \end{aligned}$$

then the gradient is given by

$$\nabla \mathcal{L}(\mathbf{w}) = -\tfrac{1}{N} \mathbf{X}^\top \mathbf{e}$$

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

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

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

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

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

# Stochastic Gradient Descent

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

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(\mathbf{w}) \ ,$$

where $\mathcal{L}_n$ is the cost contributed by the $n$-th training example.

**Q:** What are the $\mathcal{L}_n$ for linear MSE?

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

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

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

$$\mathbb{E}\left[\nabla \mathcal{L}_n(\mathbf{w})\right] = \nabla \mathcal{L}(\mathbf{w})$$

which is the true gradient direction. (check!)

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

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

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 $\nabla \mathcal{L}_n$, 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. $\mathbf{g} = \nabla \mathcal{L}$.

## SGD for Linear MSE

See Exercise Sheet 2.

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

# Variants of SGD

# SGD with Momentum

$$
\begin{aligned}
&\text{pick a stochastic gradient } \mathbf{g} \\
&\mathbf{m}^{(t+1)} := \beta_1 \mathbf{m}^{(t)} + (1-\beta_1)\mathbf{g} \qquad \text{(momentum term)} \\
&\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, \mathbf{m}^{(t+1)}
\end{aligned}
$$

- momentum from previous gradients (acceleration)

# Adam

$$
\begin{aligned}
&\text{pick a stochastic gradient } \mathbf{g} \\
&\mathbf{m}^{(t+1)} := \beta_1 \mathbf{m}^{(t)} + (1-\beta_1)\mathbf{g} && \text{(momentum term)} \\
&\mathbf{v}_i^{(t+1)} := \beta_2 \mathbf{v}_i^{(t)} + (1-\beta_2)(\mathbf{g}_i)^2 \quad \forall i && \text{(2nd-order statistics)} \\
&\mathbf{w}_i^{(t+1)} := \mathbf{w}_i^{(t)} - \frac{\gamma}{\sqrt{\mathbf{v}_i^{(t+1)}}}\, \mathbf{m}_i^{(t+1)} \quad \forall i
\end{aligned}
$$

- faster forgetting of older weights
- is a momentum variant of Adagrad
- coordinate-wise adjusted learning rate
- strong performance in practice, e.g. for self-attention networks

# SignSGD

$$
\begin{aligned}
&\text{pick a stochastic gradient } \mathbf{g} \\
&\mathbf{w}_i^{(t+1)} := \mathbf{w}_i^{(t)} - \gamma\, sign(\mathbf{g}_i) \qquad \forall i
\end{aligned}
$$

- only use the sign (one bit) of each gradient entry $\rightarrow$ communication efficient for distributed training
- convergence issues

## Non-Smooth Optimization

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

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

meaning that the function must always lie above its linearization.

## Subgradients

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

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

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

This definition makes sense for objectives $\mathcal{L}$ which are not necessarily differentiable (and not even necessarily convex).

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

## Subgradient Descent

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

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

for $\mathbf{g}$ being a subgradient to $\mathcal{L}$ 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:

   $$\mathcal{L}(\mathbf{w}) = \text{MAE}(\mathbf{w}) := \frac{1}{N} \sum_{n=1}^{N} \left| y_n - f_{\mathbf{w}}(\mathbf{x}_n) \right|$$

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

See Exercise Sheet 2.

## Stochastic Subgradient Descent

Stochastic SubGradient Descent (still abbreviated SGD commonly).

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

*Exercise:* Compute the SGD update for linear MAE.

## Implementation Issues

**Step-size selection:** If $\gamma$ is too big, the method might diverge. If it is too small, convergence is slow. Convergence to a local minimum is guaranteed only when the step-size $\gamma$ follows a schedule becoming small enough, where 'small enough' depends on the problem (landscape) parameters.

**Feature normalization and pre-conditioning:** Gradient descent is very sensitive to ill-conditioning, that is the phenomenon when some coordinates (directions) have change rates vastly different from others.

Therefore, it is typically advised to normalize input features if possible, and also normalize each layer in a neural network (we will get to this in the later chapters on neural networks). Without this, step-size selection is much more difficult since different "directions" might converge (or fail to do so) at different speed. Rescaling the space in this sense is also known as *pre-conditioning* the optimization problem.

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

## Optimality Conditions

For a *smooth* optimization problem, the first-order *necessary* condition says that at *an* optimum the gradient is equal to zero. Points of zero gradient are called critical points.

$$\nabla \mathcal{L}(\mathbf{w}^\star) = \mathbf{0}$$

If $\mathcal{L}$ is convex and $\mathbf{w}^\star$ is a critical point, then $\mathbf{w}^\star$ is a global optimum.

We can use the second derivative to study if a candidate point is a local minimum (not a local maximum or saddle-point) using the Hessian matrix,

$$\nabla^2 \mathcal{L}(\mathbf{w}) := \frac{\partial^2 \mathcal{L}}{\partial \mathbf{w} \partial \mathbf{w}^\top}(\mathbf{w})$$

The second-order *sufficient* condition states that if

- $\nabla \mathcal{L}(\mathbf{w}) = \mathbf{0}$ (critical point)
- and $\nabla^2 \mathcal{L}(\mathbf{w}) \succ 0$ (positive definite),

then $\mathbf{w}$ is a local minimum.

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.

# Non-Convex Optimization

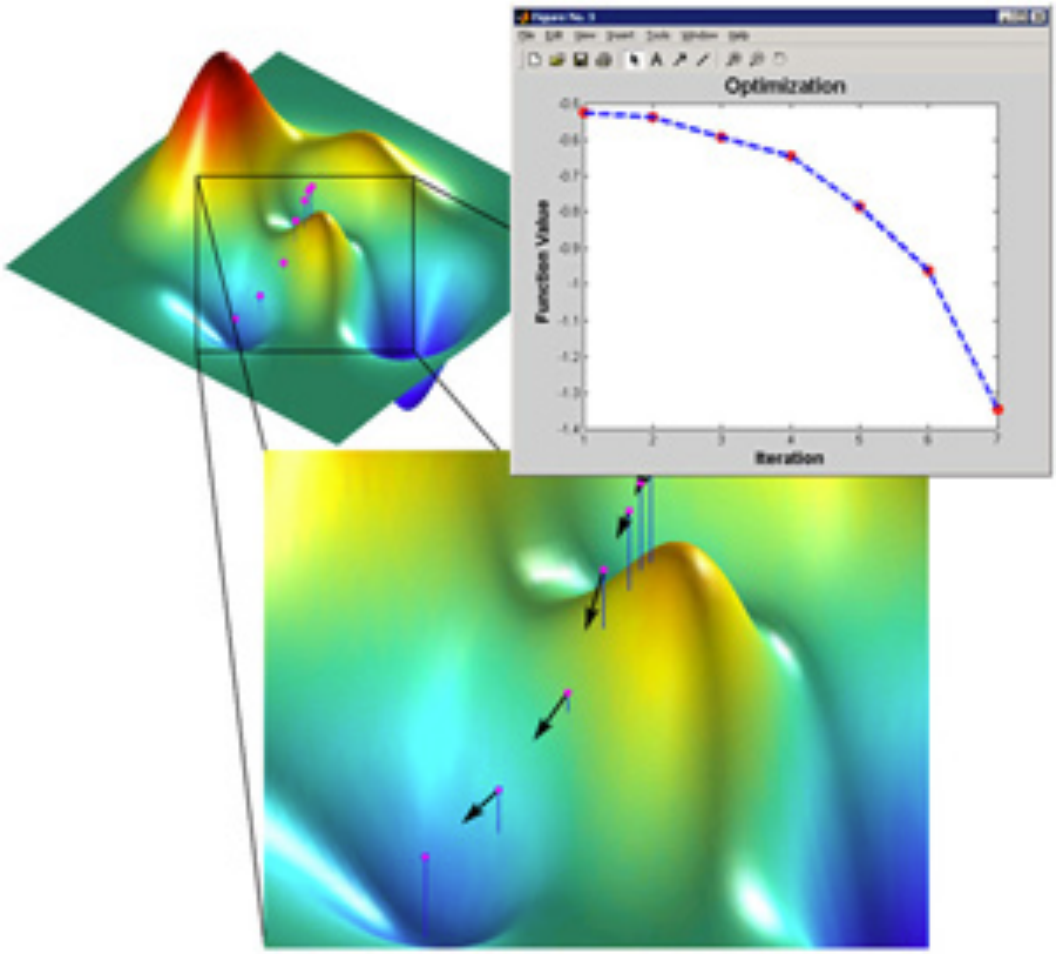



*image from mathworks.com

Real-world problems are not convex!

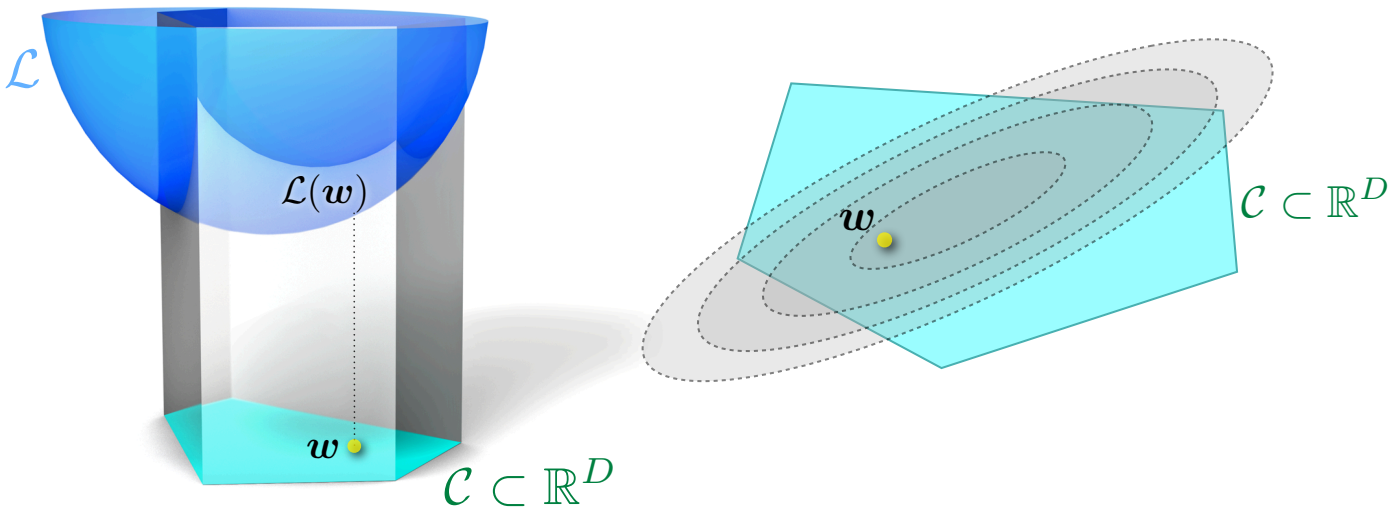
All we have learnt on algorithm design and performance of convex algorithms still helps us in the non-convex world.

# Constrained Optimization

Sometimes, optimization problems come posed with additional constraints:

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

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 $\theta$ with $0 \leq \theta \leq 1$, we have

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

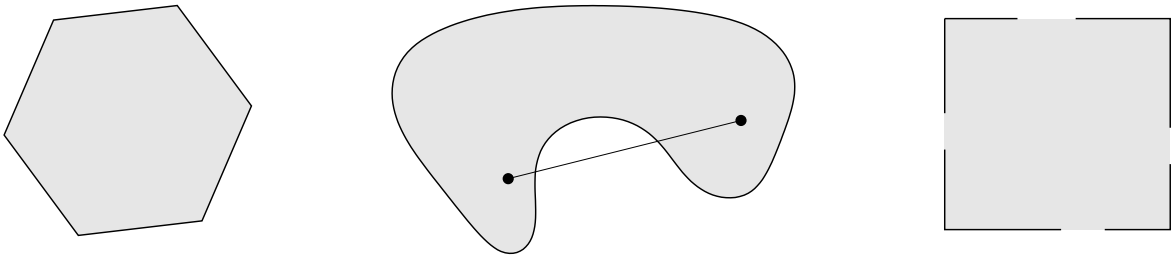


*Figure 2.2 from S. Boyd, L. Vandenberghe

# Properties of Convex Sets

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

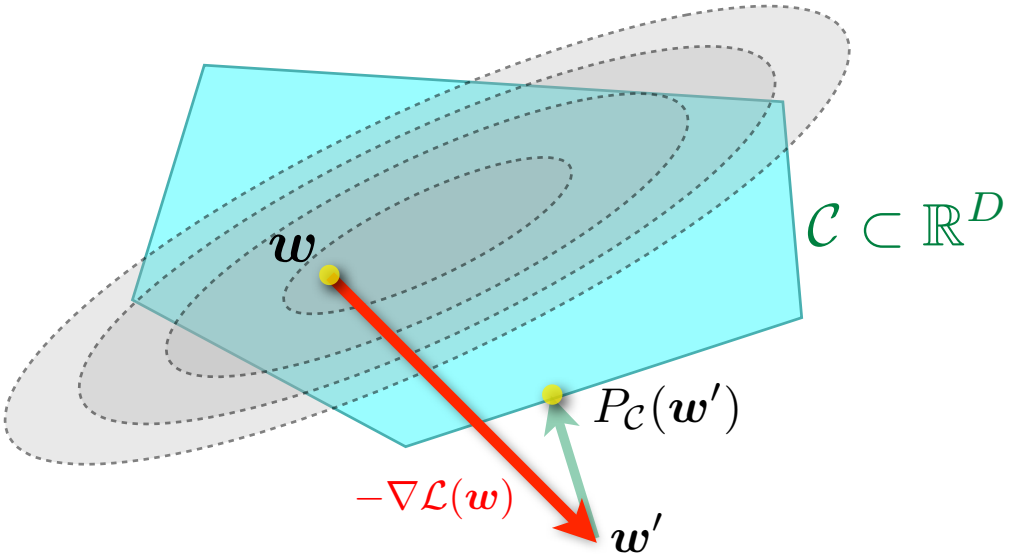
# Projected Gradient Descent

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

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

Update rule:

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




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

Computational cost of projection? Crucial!

# Turning Constrained into Unconstrained Problems

*(Alternatives to projected gradient methods)*

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

- "*brick wall*" (indicator function)

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

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

(disadvantage: non-continuous objective)

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

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

- Linearized Penalty Functions (see Lagrange Multipliers)

# Additional Notes

## Grid Search and Hyper-Parameter Optimization

Read more about grid search and other methods for "hyperparameter" setting:
en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search.

## Computational Complexity

The computation cost is expressed using the big-$\mathcal{O}$ notation. Here is a definition taken from Wikipedia. Let $f$ and $g$ be two functions defined on some subset of the real numbers. We write $f(x) = \mathcal{O}(g(x))$ as $x \to \infty$, if and only if there exists a positive real number $c$ and a real number $x_0$ such that $|f(x)| \leq c|g(x)|, \quad \forall x > x_0$.

Please read and learn more from this page in Wikipedia:
en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra .

- What is the computational complexity of matrix multiplication?
- What is the computational complexity of matrix-vector multiplication?

## 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:

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

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

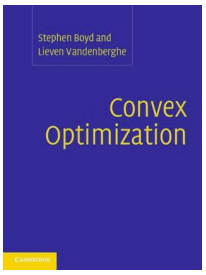
$$\sum_{t=1}^{\infty} \gamma^{(t)} = \infty, \qquad \sum_{t=1}^{\infty} (\gamma^{(t)})^2 < \infty \tag{1}$$

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




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

# Exercises


1. Chain-rule

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

2. Revise computational complexity ($\mathcal{O}$-notation).

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.

5. Implement gradient descent and gain experience in setting the step-size.

6. Implement SGD and gain experience in setting the step-size.