Optimization and Machine Learning

Sergio Peignier

What is Machine Learning?

Definition

Machine Learning (ML) studies algorithms that learn patterns from data.

Key Idea

Training a model = solving an optimization problem

$$\min_{\theta} \text{Loss}(\theta)$$

- θ: model parameters
- Loss: quality measures (e.g., prediction error, Likelihood)

Example: Linear Regression

Problem

Given data $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$

$$\mathcal{L}(w) = \|X \cdot w - y\|_2^2$$

- Goal: find w minimizing prediction error
- Objective is quadratic and convex

Linear Regression Solution

$$\nabla \mathcal{L}(w) = 2X^\top (X \cdot w - y)$$

$$X^\top X \cdot w = X^\top y$$

$$w^* = (X^\top X)^{-1} X^\top y$$

Insight

- Closed-form solution exists
- Unique solution if $X^\top X$ invertible

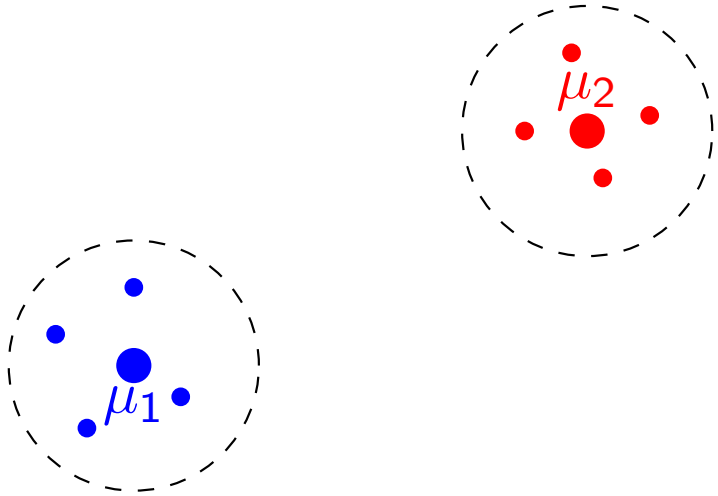
K-Means: Intuition

- Group data into K clusters
- Each cluster has a centroid
- Points assigned to closest centroid

Goal

Minimize intra-cluster distance

K-Means Illustration



K-Means as Optimization

$$\min_{\mu_k, c(i)} \sum_{i=1}^{n} \|x_i - \mu_{c(i)}\|^2$$

- $c(i)$: cluster assignment (discrete)
- μ_k: centroids (continuous)

Key Insight

Mixed optimization problem:

- discrete & continuous

K-Means Algorithm

1. Assign each point to closest centroid
2. Update centroids as means

Properties

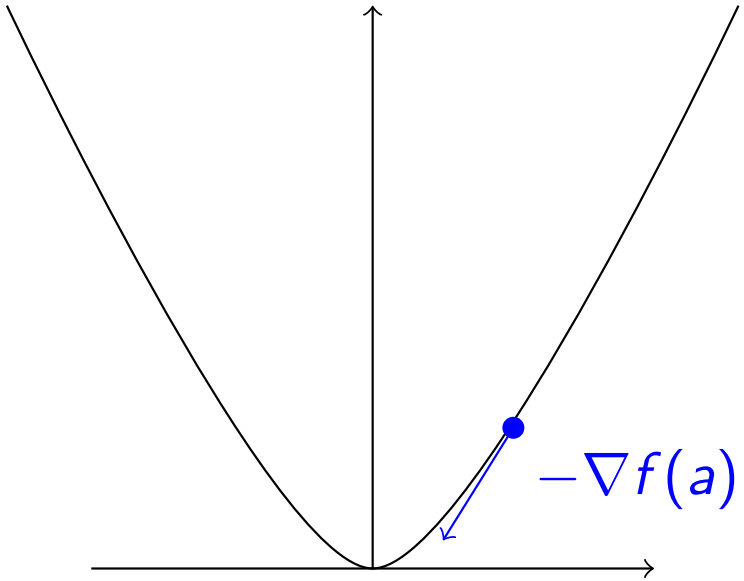
- Each step decreases objective
- Converges to local minimum

Gradient Descent: Intuition and Algorithm

Idea

Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient $\nabla f(a)$ points in the direction of **steepest increase**. To minimize f, we move in the **opposite direction** to gradient.

Example: Minimize $f(x) = x^2$



Gradient Descent: Update Rule

Update Rule

$$a^{(k+1)} = a^{(k)} - \gamma_k \nabla f(a^{(k)})$$

- $\gamma_k > 0$: learning rate (step size)
- Large γ_k: fast but risk of divergence
- Small γ_k: stable but slow convergence

Convergence (intuition)

- If f is **convex** $\Rightarrow$ converges to global minimum
- If f is **non-convex** $\Rightarrow$ may converge to local minimum
- Requires f smooth (differentiable)

Gradient Descent: When Do We Stop?

Termination Criteria

Gradient Descent is an iterative algorithm, so we need rules to stop it.

- **Maximum iterations** $k \geq K_{\max}$
- **Small gradient** (Close to a stationary point) $\|\nabla f(a_k)\| \leq \varepsilon$
- **Small change in parameters** $\|a_{k+1} - a_k\| \leq \varepsilon$
- **Small change in objective** $|f(a_{k+1}) - f(a_k)| \leq \varepsilon$

Practical Tip

Often combine several criteria for robustness.

Gradient Descent Algorithm

Input: $a_0,\ \gamma,\ K$

$a \leftarrow a_0$
for $k = 1 \ldots K$ **do**
  $g \leftarrow \nabla f(a)$
  $a \leftarrow a - \gamma g$

return a

- Start from an initial guess a_0
- Iteratively update using the gradient
- Produces an approximation of a minimum

Gradient Descent Variants

Classical Variants

- **Batch Gradient Descent** $\nabla f(a) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(a)$.
  Uses entire dataset. Stable but expensive
- **Stochastic Gradient Descent (SGD)** $\nabla f(a) \approx \nabla f_i(a)$
  Uses one sample at a time. Fast but noisy updates.
- **Mini-batch Gradient Descent**. Uses small batches of data.
  Trade-off: speed & stability

Deep Learning - Oriented Variants

- **Momentum**: smooth updates
- **RMSProp**: adaptive learning rates
- **Adam**: combines momentum + adaptation

Example: Applying Gradient Descent

Decision Function

$$f(a, b) = \cos(ax_1) - \sin(bx_2)$$

Apply Gradient Descent

- Compute the gradient with respect to parameters a and b
- Apply update rule until termination criteria

Example: Gradient Computation

Partial Derivatives

Using the chain rule:

$$\frac{\partial f}{\partial a} = -\sin(ax_1) \cdot x_1$$

$$\frac{\partial f}{\partial b} = -\cos(bx_2) \cdot x_2$$

Gradient Vector

$$\nabla f(a, b) = \begin{pmatrix} -\sin(ax_1)\, x_1 \\ -\cos(bx_2)\, x_2 \end{pmatrix}$$

Example: Apply Update steps

Gradient Descent Update

$$a \leftarrow a + \gamma \sin(a x_1)\, x_1$$

$$b \leftarrow b + \gamma \cos(b x_2)\, x_2$$

Simulated Annealing: Motivation

Problem with local search

Gradient descent / greedy methods:

- Get stuck in local minima
- Cannot escape poor regions

Key idea

Sometimes we must **accept worse solutions** to explore the search space.

Goal

Design a method that:

- explores globally at first
- exploits locally at the end

Simulated Annealing: Analogy with Physics

Annealing in metallurgy

- Heat metal to high temperature
- Atoms move freely (exploration)
- Slowly cool down
- System stabilizes in low-energy state

Optimization analogy

- State = candidate solution
- Energy = objective function $E(s)$
- Temperature T controls randomness

$$\text{High } T \Rightarrow \text{exploration}, \quad \text{Low } T \Rightarrow \text{exploitation}$$

Simulated Annealing Algorithm

Core idea

- Propose a variant
- Accept better solutions always
- Sometimes accept worse ones

Pseudocode

$s \leftarrow$ random initial state
for $k = 1 \dots K$ **do**
  $T \leftarrow T(k)$
  $s' \leftarrow \text{neighbor}(s)$
  $\Delta E = E(s') - E(s)$
  if $\Delta E \leq 0$ **then** $s \leftarrow s'$
  else accept with prob $e^{-\Delta E/T}$

Acceptance Rule and Behavior

Acceptance probability

If the new solution is worse:

$$P(\text{accept}) = e^{-(E(s')-E(s))/T}$$

- If $T \to \infty$: almost always accept
- If $T \to 0$: behaves like greedy descent

Key insight

Early: explore widely
Late: refine solution

Example: Traveling Salesman Problem

- **Problem**: Find the shortest tour visiting all cities exactly once.
- **Representation**
  - State s: permutation of cities
  - Energy $E(s)$: total tour length
  - Neighbor: swap two cities
- Example: Swap *LaPaz* and *Madrid*

$$(Lyon, Madrid, Mexico, LaPaz) \rightarrow (Lyon, LaPaz, Mexico, Madrid)$$

Why SA works here

- Can escape local minima
- Explores many permutations early
- Converges to near-optimal tours

Genetic Algorithms: Motivation

Why do we need them?

Many optimization problems are:

- non-convex
- discrete or combinatorial
- full of local minima

Key idea

Instead of improving one solution, we evolve a **population of solutions**.

Biological inspiration

- natural selection
- survival of the fittest
- genetic variation (mutation & recombination)

Intuition: Evolution as Optimization

Key analogy

- Individual = candidate solution
- Population = set of solutions
- Fitness = objective function
- Better solutions survive and reproduce

Evolution process

1. Evaluate fitness
2. Select best individuals
3. Recombine (crossover)
4. Random mutation

Genetic Algorithm: Core Algorithm

Pseudocode

$P \leftarrow$ random population
for $k = 1 \ldots K$ **do**
  evaluate fitness(P)
  $P_{\text{parents}} \leftarrow \text{selection}(P)$
  $P_{\text{children}} \leftarrow \text{crossover}(P_{\text{parents}})$
  $P_{\text{children}} \leftarrow \text{mutation}(P_{\text{children}})$
  $P \leftarrow P_{\text{children}}$

- Iterative improvement of a population
- Stochastic but guided by fitness

Genetic Algorithms: Selection

- **Goal of selection**: Select individuals with higher fitness to become parents.
- **Intuition**: Good solutions should have a higher probability of reproducing, but weaker solutions are not completely discarded.
- **Common methods**:
  - tournament selection
  - roulette wheel (fitness proportional)
  - rank-based selection
- Example (TSP)

$$s_1 = (\text{Lyon}, \text{Madrid}, \text{Mexico}, \text{La Paz})$$

$$s_2 = (\text{Madrid}, \text{La Paz}, \text{Lyon}, \text{Mexico})$$

If s_1 is shorter than s_2, then:

- s_1 has higher probability of being selected
- but s_2 can still be chosen

Genetic Algorithms: Crossover

- **Goal of crossover**: Combine two parent solutions to create new offspring.
- **Intuition**: Good partial structures from different solutions can be combined to produce an even better solution.
- **Key constraint** (TSP) Must preserve valid solutions
- Example (TSP) Parents:

$$p_1 = (\text{Lyon}, \text{Madrid}, \text{Mexico}, \text{La Paz})$$

$$p_2 = (\text{Madrid}, \text{La Paz}, \text{Lyon}, \text{Mexico})$$

Cut and recombine:

- take middle segment from p_1
- fill missing cities from p_2

$$\text{Child} = (\text{La Paz}, \text{Madrid}, \text{Mexico}, \text{Lyon})$$

Genetic Algorithms: Mutation

- **Goal of mutation**: Introduce random modifications to maintain diversity. Exploration mechanism of the algorithm
- **Intuition** Without mutation:
  - population becomes too similar
  - algorithm can stagnate in local optima
- Example TSP: swap mutation

  Initial: $s = (\text{Lyon}, \text{Madrid}, \text{Mexico}, \text{La Paz})$

  Swap two cities ("Madrid" and "La Paz"):

  Mutant: $s' = (\text{Lyon}, \text{La Paz}, \text{Mexico}, \text{Madrid})$