Aprendizagem Aplicada à Segurança

Optimization Algorithms for Machine Learning

Mário Antunes

September 19, 2025

Universidade de Aveiro

Table of Contents i

Optimization Concepts

- At its core, **optimization** is the process of finding the best possible solution to a problem.
- In mathematics, this means finding the **minimum** (or maximum) value of a function.
- In machine learning, “training” a model is an optimization problem:
  - We define a **Cost Function** (or Loss Function) that measures our model’s error.
  - We then use an optimization algorithm to find the model parameters (weights) that **minimize** this error.

- We will explore two families of optimizers:

1. **Gradient-Based:** Uses the function's derivative (e.g., Gradient Descent).
2. **Derivative-Free (Blind):** Does not require a derivative (e.g., Particle Swarm Optimization).

Blind/Derivative-Free Optimization

This is a type of “black-box” optimization.

Why use it? It’s essential for problems where the objective function $f(x)$ is:

- **Non-smooth** or discontinuous (no clear derivative).
- **Noisy** (the same input might give slightly different outputs).
- **Time-consuming** or expensive to evaluate.
- A “black box” where we don’t know the underlying equation, only the input-output.

Examples include Genetic Algorithms, Differential Evolution, and **Particle Swarm Optimization**.

Particle Swarm Optimization (PSO)

The Analogy: PSO is inspired by the social behavior of a flock of birds or a school of fish searching for food.

- The entire swarm is searching for a single point: the best food source (the global minimum).
- No single bird knows where the food is.
- By sharing information, the flock can find it.

The Algorithm:

- A “swarm” of “particles” is initialized with random positions and velocities in the search space.
- Each particle “flies” through the space, and its “fitness” (how good its solution is) is evaluated by the objective function.
- Each particle remembers its own **personal best** (`pbest`) position found so far.
- The swarm tracks the **global best** (`gbest`) position found by *any* particle in the swarm.
- Each particle updates its velocity based on these two pieces of information.

Gradient Descent (GD)

The Analogy: Imagine you are lost on a mountain in a thick fog. Your goal is to get to the bottom of the valley (the minimum). You can't see, but you can feel the slope of the ground beneath your feet.

The simplest strategy is:

1. Check the slope (gradient) in all directions.
2. Take one step in the direction of the **steepest descent**.
3. Repeat until the ground is flat (the gradient is zero).

The Concept: Gradient Descent is an iterative algorithm to find a **local minimum** of a function by repeatedly moving in the opposite direction of the gradient.

GD: The 1D Algorithm

For a simple function $f(x)$, the derivative $f'(x)$ gives the slope at any point x.

The algorithm's core is the **update rule**:

$$x_{t+1} = x_t - \alpha \times f'(x_t)$$

- $f'(x_t)$ is the derivative (slope) at the current point.
- α is the **Learning Rate**, a hyperparameter that controls how big of a step you take.
- **Too small α:** The algorithm is very slow.
- **Too large α:** The algorithm can "overshoot" the minimum and fail to converge.

GD: Multivariable Functions

For functions with many inputs, like $f(x, y)$, we can't use a simple derivative. We use the **Gradient**.

What is the Gradient (∇f)?

- It is a **vector** (a list of numbers) containing the **partial derivative** for each input variable.
- Example: $\nabla f = \left[\frac{\partial f}{\partial x} \frac{\partial f}{\partial y}\right]$
- The gradient vector always points in the direction of the **steepest ascent** (uphill).

The Algorithm: The update rule remains the same, but now we use vectors. We simply move in the **opposite** direction of the gradient.

$$\vec{x}_{t+1} = \vec{x}_t - \alpha \times \nabla f(\vec{x}_t)$$

GD vs. BO: Key Differences i

Feature	Gradient Descent (GD)	Particle Swarm Optimization (PSO)
Requirement	Needs the function's **gradient** (derivative).	Does **not** need the gradient.
Function Type	Best for smooth, continuous, and (ideally) convex functions.	Works well for noisy, non-smooth, non-convex functions.
Search	Deterministic. Follows the single steepest path downhill.	Stochastic (random). Uses a population to search many areas at once.

GD vs. BO: Key Differences ii

Local Minima	Can easily get **stuck** in a local minimum.	Good at “jumping out” of local minima to find the **global minimum**.
Analogy	A single hiker walking downhill.	A flock of birds searching for the lowest point in a valley.

Application 1: Linear Regression

The Goal: Fit a straight line ($y = mx + b$) to a set of data points.

1. The Model:

$$h(x, m, b) = m \times x + b$$

- The parameters we need to find are m (slope) and b (intercept).

2. The Cost Function (Mean Squared Error, MSE): We need a function that measures the total error. MSE is the average squared vertical distance between the data points and our line.

$$e(m, b) = \frac{1}{2n} \sum_{i=0}^{n} (y_i - h(x_i, m, b))^2$$

The Optimization Problem: Find the specific values for m and b that **minimize** the $e(m, b)$ cost function.

Solving Linear Regression

Method 1: Gradient Descent

1. We must calculate the partial derivatives of the MSE cost function:
   - $\frac{\partial e}{\partial m}$ (how the error changes with the slope)
   - $\frac{\partial e}{\partial b}$ (how the error changes with the intercept)
2. We apply the GD update rule, using these derivatives to iteratively "nudge" m and b towards their optimal values.

Method 2: Particle Swarm Optimization (PSO)

1. We do **not** need the derivatives.
2. Each "particle" is a candidate solution, a vector: $[m, b]$.
3. The "fitness" of a particle is its MSE: $e(m, b)$.
4. The swarm "flies" through the 2D search space of (m, b) values, looking for the pair that has the lowest MSE.

Key Idea: Both algorithms solve the *same problem* (minimizing MSE), but they use completely different strategies to do it.

Application 2: Logistic Regression

The Goal: Classify data into two categories (e.g., 0 or 1, "Spam" or "Not Spam").

1. The Model: We can't use a straight line. We need a function that outputs a probability (a number between 0 and 1).

1. We start with a linear model: $z = w \cdot x + b$
2. We "squash" its output using the **Sigmoid Function**: $\sigma(z) = \frac{1}{1+e^{-z}}$

- **Model Equation:** $h(x) = \sigma(w \cdot x + b)$

2. The Cost Function (Log Loss): MSE does not work well for classification. We use **Log Loss** (or Binary Cross-Entropy).

$$J(\theta) = -[y \log(h(x)) + (1 - y) \log(1 - h(x))]$$

This function heavily penalizes the model for being *confident and wrong*.

Solving Logistic Regression

The Optimization Problem: Find the specific values for the weights w and bias b that **minimize** the total **Log Loss** over all data points.

Method 1: Gradient Descent

1. We must calculate the partial derivatives of the **Log Loss function** with respect to each weight w_i and the bias b.
2. We apply the GD update rule to iteratively find the best parameters.

Method 2: Particle Swarm Optimization (PSO)

1. We do **not** need the derivatives.
2. Each “particle” is a candidate solution, a vector of all parameters: $[b, w_1, w_2, \dots]$.
3. The “fitness” of a particle is its Log Loss $J(\theta)$.
4. The swarm searches for the parameter vector that results in the lowest Log Loss.

Conclusion:

- Optimization is the core engine of ML.
- By defining a **model** and a **cost function**, we can use powerful, general-purpose optimizers like GD or BO to find the best possible version of that model.