Diogo Ribeiro

ESMAD - Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

dfr@esmad.ipp.pt 0009-0001-2022-7072

January 14, 2026

Outline

What is Optimization?

Optimization Problem

Finding the best solution from a set of feasible alternatives:

$$\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and $\mathcal{X}$ is the feasible set.

Components:

- **Decision variables:** $\boldsymbol{x} = (x_1, \ldots, x_n)$
- **Objective function:** $f(\boldsymbol{x})$ (cost, loss, error)
- **Constraints:** $g_i(\boldsymbol{x}) \leq 0$, $h_j(\boldsymbol{x}) = 0$

Goal: Find $\boldsymbol{x}^*$ that minimizes (or maximizes) $f(\boldsymbol{x})$.

Optimization in Data Science

Machine Learning:

- Training models: minimize loss function
- Linear regression: minimize $\sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$
- Logistic regression: minimize cross-entropy
- Neural networks: backpropagation + gradient descent

Statistics:

- Maximum likelihood estimation
- Bayesian inference (MAP estimation)
- Robust estimation

Operations Research:

- Resource allocation
- Scheduling and routing
- Portfolio optimization

Hyperparameter Tuning:

Types of Optimization Problems

By Variable Type:

- **Continuous:** $\boldsymbol{x} \in \mathbb{R}^n$
- **Discrete:** $\boldsymbol{x} \in \mathbb{Z}^n$
- **Mixed-integer:** Both types

By Constraints:

- **Unconstrained:** $\mathcal{X} = \mathbb{R}^n$
- **Constrained:** $\mathcal{X} \subset \mathbb{R}^n$

By Objective:

- **Linear:** $f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$
- **Quadratic:** $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\mathbf{Q}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$
- **Convex:** f is convex
- **Non-convex:** General case

By Number of Objectives:

- **Single-objective**
- **Multi-objective**

Complexity

Convex problems are "easy" (polynomial time). Non-convex and discrete problems are generally NP-hard.

Convex Sets and Functions

Convex Set

A set $\mathcal{C}$ is convex if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ and $\theta \in [0, 1]$:

$$\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{y} \in \mathcal{C}$$

Convex Function

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for any $\boldsymbol{x}, \boldsymbol{y}$ and $\theta \in [0, 1]$:

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

Equivalent conditions (differentiable f):

- **First-order:** $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x})$
- **Second-order:** $\nabla^2 f(\boldsymbol{x}) \succeq 0$ (Hessian is positive semidefinite)

Properties of Convex Functions

Key properties:

- **Local minimum is global minimum**
- **Sublevel sets are convex:** $\{\boldsymbol{x} : f(\boldsymbol{x}) \leq \alpha\}$
- **Closed under positive combinations:** If f_1, f_2 convex, then $\alpha f_1 + \beta f_2$ convex for $\alpha, \beta \geq 0$
- **Composition rules:** $f(g(\boldsymbol{x}))$ may be convex under certain conditions

Common convex functions:

- Linear: $\boldsymbol{a}^T\boldsymbol{x} + b$
- Quadratic (with $\mathbf{Q} \succeq 0$): $\boldsymbol{x}^T\mathbf{Q}\boldsymbol{x}$
- Norms: $\|\boldsymbol{x}\|_p$ for $p \geq 1$
- Exponential: e^{ax}
- Logarithm: $-\log(x)$ on $\mathbb{R}_{++}$
- Maximum: $\max\{x_1, \ldots, x_n\}$

Fundamental Property

For convex f over convex set $\mathcal{X}$: any local minimum is a global minimum

Optimality Conditions

First-Order Optimality (Unconstrained)

For differentiable f, $\boldsymbol{x}^*$ is a minimum if and only if:

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0}$$

Second-Order Optimality

Necessary: If $\boldsymbol{x}^*$ is a local minimum, then:

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \quad \text{and} \quad \nabla^2 f(\boldsymbol{x}^*) \succeq 0$$

Sufficient: If $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}^*) \succ 0$ (positive definite), then $\boldsymbol{x}^*$ is a strict local minimum.

Convex Optimization Problem

Standard Form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leq 0, \quad i = 1, \ldots, m \\ & \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \end{aligned}$$

where f, g_i are convex functions.

Special cases:

- **Linear Programming (LP):** f and g_i are affine
- **Quadratic Programming (QP):** f quadratic, g_i affine
- **Second-Order Cone Programming (SOCP):** Generalization of QP
- **Semidefinite Programming (SDP):** Matrix optimization

Gradient Descent

Idea: Iteratively move in the direction of steepest descent.

Gradient Descent Algorithm

Starting from $\boldsymbol{x}_0$, iterate:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

where $\alpha_k > 0$ is the step size (learning rate).

Convergence conditions:

- For convex, L-Lipschitz continuous ∇f: converges with $\alpha_k = \frac{1}{L}$
- Convergence rate: $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) = O(1/k)$

Step size selection:

- **Constant:** $\alpha_k = \alpha$
- **Decreasing:** $\alpha_k = \frac{\alpha_0}{k}$ or $\alpha_k = \frac{\alpha_0}{\sqrt{k}}$
- **Backtracking line search:** Adaptively find good step size

Batch vs Stochastic Gradient Descent

Batch Gradient Descent:

- Use full dataset to compute gradient
- For $f(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i)$:

$$\nabla f(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i)$$

- Exact gradient, but expensive for large n

Stochastic Gradient Descent (SGD):

- Use single random sample to estimate gradient

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla \ell(y_i, \boldsymbol{w}_k^T \boldsymbol{x}_i)$$

- Fast updates, but noisy gradient estimates

Mini-batch SGD:

- Compromise: use batch of b samples
- Most common in practice (typical b: 32, 64, 128, 256)

Momentum Methods

Problem: Standard GD can be slow in ill-conditioned problems.

Gradient Descent with Momentum

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k - \alpha \nabla f(\mathbf{x}_k)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1}$$

where $\beta \in [0, 1)$ is the momentum coefficient (typically 0.9).

Intuition:

- Accumulate velocity in directions of consistent gradient
- Dampens oscillations in high-curvature directions
- Accelerates progress in low-curvature directions

Nesterov Accelerated Gradient (NAG)

Adaptive Learning Rate Methods

AdaGrad (Adaptive Gradient):

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{\alpha}{\sqrt{\boldsymbol{G}_k + \epsilon}} \odot \nabla f(\boldsymbol{x}_k)$$

where $\boldsymbol{G}_k = \sum_{i=0}^{k} (\nabla f(\boldsymbol{x}_i))^2$ (element-wise), $\epsilon \approx 10^{-8}$

- Adapts learning rate per parameter
- Good for sparse gradients
- Problem: Accumulated gradients can make learning rate too small

RMSProp:

$$\boldsymbol{G}_k = \beta \boldsymbol{G}_{k-1} + (1-\beta)(\nabla f(\boldsymbol{x}_k))^2$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{\alpha}{\sqrt{\boldsymbol{G}_k + \epsilon}} \odot \nabla f(\boldsymbol{x}_k)$$

- Uses exponential moving average
- Fixes AdaGrad's aggressive learning rate decay

Adam Optimizer

Adam (Adaptive Moment Estimation)

Combines momentum and adaptive learning rates:

$$\begin{aligned}
\boldsymbol{m}_k &= \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\nabla f(\boldsymbol{x}_k) && \text{(first moment)}\\
\boldsymbol{v}_k &= \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)(\nabla f(\boldsymbol{x}_k))^2 && \text{(second moment)}\\
\hat{\boldsymbol{m}}_k &= \frac{\boldsymbol{m}_k}{1-\beta_1^k} && \text{(bias correction)}\\
\hat{\boldsymbol{v}}_k &= \frac{\boldsymbol{v}_k}{1-\beta_2^k} && \text{(bias correction)}\\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \frac{\alpha}{\sqrt{\hat{\boldsymbol{v}}_k}+\epsilon} \odot \hat{\boldsymbol{m}}_k
\end{aligned}$$

Default hyperparameters: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Other Modern Optimizers

AdamW:

- Decoupled weight decay from gradient-based updates
- Better regularization than Adam with L2 penalty
- Becoming preferred over Adam for many applications

RAdam (Rectified Adam):

- Addresses early training instability
- Adaptive learning rate warm-up

Lookahead:

- Meta-optimizer that can wrap any base optimizer
- Maintains slow and fast weights

Second-order methods:

- **Newton's method:** Use Hessian information
- **L-BFGS:** Limited-memory quasi-Newton
- Fast convergence but expensive per iteration

Gradient Descent Variants Comparison

Method	Momentum	Adaptive LR	Memory	Use Case
SGD	No	No	Low	Simple, well-understood
SGD+Momentum	Yes	No	Low	Standard baseline
AdaGrad	No	Yes	Medium	Sparse features
RMSProp	No	Yes	Medium	RNNs, online learning
Adam	Yes	Yes	Medium	General purpose
AdamW	Yes	Yes	Medium	With weight decay
L-BFGS	-	-	High	Small datasets

Practical Advice

- Start with Adam/AdamW with default parameters
- If overfitting, add weight decay or use AdamW
- For final tuning, try SGD with momentum and learning rate schedule
- Monitor training curves to detect issues

Lagrangian and Duality

Primal problem:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0, \quad i = 1, \ldots, m \\ & h_j(\boldsymbol{x}) = 0, \quad j = 1, \ldots, p \end{aligned}$$

Lagrangian

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \nu_j h_j(\boldsymbol{x})$$

where $\boldsymbol{\lambda} \geq \mathbf{0}$ (inequality multipliers) and $\boldsymbol{\nu}$ (equality multipliers).

Dual Function

KKT Conditions

Karush-Kuhn-Tucker (KKT) Conditions

For $\boldsymbol{x}^*$ to be optimal (assuming constraint qualifications hold):

1. Stationarity:

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \nu_j^* \nabla h_j(\boldsymbol{x}^*) = \mathbf{0}$$

2. Primal feasibility:

$$g_i(\boldsymbol{x}^*) \leq 0, \quad h_j(\boldsymbol{x}^*) = 0$$

3. Dual feasibility:

$$\lambda_i^* \geq 0$$

4. Complementary slackness:

$$\lambda_i^* g_i(\boldsymbol{x}^*) = 0, \quad \forall i$$

Penalty and Barrier Methods

Penalty Methods:

- Convert constrained problem to unconstrained
- Add penalty for constraint violations:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \rho \sum_{i=1}^{m} \max(0, g_i(\boldsymbol{x}))^2 + \rho \sum_{j=1}^{p} h_j(\boldsymbol{x})^2$$

- Increase penalty parameter ρ iteratively

Barrier Methods:

- Add barrier function to prevent constraint violations
- Logarithmic barrier:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) - \frac{1}{\mu} \sum_{i=1}^{m} \log(-g_i(\boldsymbol{x}))$$

- Decrease barrier parameter μ iteratively
- Basis of interior-point methods

Projected Gradient Descent

For problems with simple constraints:

$$\min_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x})$$

Projected Gradient Descent

$$\boldsymbol{x}_{k+1} = \Pi_{\mathcal{C}}(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k))$$

where $\Pi_{\mathcal{C}}(\boldsymbol{y}) = \arg\min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{y}\|_2$ is the projection onto $\mathcal{C}$.

Common projections:

- **Box constraints:** $\mathcal{C} = [\boldsymbol{l}, \boldsymbol{u}]$

$$[\Pi_{\mathcal{C}}(\boldsymbol{y})]_i = \max(l_i, \min(y_i, u_i))$$

- ℓ_2 **ball:** $\mathcal{C} = \{\boldsymbol{x} : \|\boldsymbol{x}\|_2 \leq r\}$

Proximal Methods

For problems with non-smooth terms:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + g(\boldsymbol{x})$$

where f is smooth, g is non-smooth but "simple" (e.g., ℓ_1 norm).

Proximal Operator

$$\text{prox}_{\alpha g}(\boldsymbol{y}) = \arg\min_{\boldsymbol{x}} \left\{ g(\boldsymbol{x}) + \frac{1}{2\alpha}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 \right\}$$

Proximal Gradient Method

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k g}(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k))$$

Evolutionary Algorithms Overview

Inspiration: Biological evolution and natural selection.

Key concepts:

- **Population:** Set of candidate solutions
- **Fitness:** Quality of a solution (objective function value)
- **Selection:** Choose better solutions for reproduction
- **Crossover:** Combine two solutions to create offspring
- **Mutation:** Random modifications to solutions
- **Replacement:** Update population with new generation

Advantages:

- Don't require gradient information
- Can handle non-convex, multimodal, discrete problems
- Naturally handle constraints and multiple objectives
- Global search capability

Disadvantages:

Genetic Algorithms (GA)

Genetic Algorithm

1. **Initialize:** Random population of size N
2. **Repeat until convergence:**
   1. **Evaluate:** Compute fitness $f(\boldsymbol{x})$ for each individual
   2. **Selection:** Select parents based on fitness
      - Roulette wheel: probability $\propto$ fitness
      - Tournament: best of random subset
      - Rank-based: based on rank, not absolute fitness
   3. **Crossover:** Create offspring from parents
      - One-point, two-point, uniform crossover
   4. **Mutation:** Randomly modify offspring
      - Bit flip (binary), Gaussian noise (real-valued)
   5. **Replacement:** Form new population

Evolution Strategies (ES)

Designed for continuous optimization:

- Represent solutions as real vectors
- Self-adaptive mutation rates
- Strong theoretical foundations

(μ, λ)-ES

- μ parents generate λ offspring
- Select best μ from offspring (no elitism)

$(\mu + \lambda)$-ES

- Select best μ from parents and offspring (elitism)

CMA-ES (Covariance Matrix Adaptation):

Particle Swarm Optimization (PSO)

Inspiration: Social behavior of bird flocking or fish schooling.

PSO Algorithm

For each particle i with position $\boldsymbol{x}_i$ and velocity $\boldsymbol{v}_i$:

1. Track personal best: $\boldsymbol{p}_i =$ best position particle i has visited
2. Track global best: $\boldsymbol{g} =$ best position any particle has visited
3. Update velocity:

$$\boldsymbol{v}_i \leftarrow \omega \boldsymbol{v}_i + c_1 r_1(\boldsymbol{p}_i - \boldsymbol{x}_i) + c_2 r_2(\boldsymbol{g} - \boldsymbol{x}_i)$$

   where $r_1, r_2 \sim \text{Uniform}(0, 1)$

4. Update position:

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$$

Other Metaheuristics

Simulated Annealing:

- Probabilistic local search with controlled randomness
- Accepts worse solutions with probability $\exp(-\Delta f / T)$
- Temperature T decreases over time
- Guarantees convergence to global optimum (in limit)

Differential Evolution (DE):

- Create offspring by adding scaled difference of random individuals
- Simple and effective for continuous optimization
- Few hyperparameters

Ant Colony Optimization (ACO):

- Inspired by foraging behavior of ants
- Good for combinatorial problems (TSP, routing)

Tabu Search:

- Local search with memory to avoid cycling

Bayesian Optimization for Hyperparameter Tuning

Problem: Optimize expensive black-box function $f(\boldsymbol{x})$.

Examples:

- Hyperparameter tuning (training neural network expensive)
- A/B testing (each evaluation requires collecting data)
- Engineering design (simulations/experiments costly)

Bayesian Optimization

1. **Surrogate model:** Probabilistic model of f (typically Gaussian Process)

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$$

2. **Acquisition function:** Determines next point to evaluate
   - Balance exploration vs exploitation
   - Examples: Expected Improvement (EI), Upper Confidence Bound (UCB)

Gaussian Processes for Bayesian Optimization

Gaussian Process

Distribution over functions specified by:

- Mean function: $\mu(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$
- Covariance (kernel) function: $k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}'))$

Common kernels:

- **RBF (Gaussian):** $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\ell^2}\right)$
- **Matérn:** $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{r}{\ell}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{r}{\ell}\right)$

Posterior after observing data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$:

$$f(\boldsymbol{x})|\mathcal{D} \sim \mathcal{N}(\mu_{\mathcal{D}}(\boldsymbol{x}), \sigma^2_{\mathcal{D}}(\boldsymbol{x}))$$

Key property: Uncertainty quantification - GP provides both prediction and confidence.

Acquisition Functions

Expected Improvement (EI):

$$\text{EI}(\boldsymbol{x}) = \mathbb{E}[\max(0, f(\boldsymbol{x}) - f(\boldsymbol{x}^+))]$$

where $\boldsymbol{x}^+ = \arg\max_{\boldsymbol{x}_i \in \mathcal{D}} y_i$ is current best.
Closed form for GP:

$$\text{EI}(\boldsymbol{x}) = \begin{cases} (\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+))\Phi(Z) + \sigma(\boldsymbol{x})\phi(Z) & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{x}) = 0 \end{cases}$$

where $Z = \frac{\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+)}{\sigma(\boldsymbol{x})}$, Φ is standard normal CDF, ϕ is PDF.

Upper Confidence Bound (UCB):

$$\text{UCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x})$$

where $\kappa > 0$ controls exploration-exploitation tradeoff.

Probability of Improvement (PI):

$$\text{PI}(\boldsymbol{x}) = \mathbb{P}(f(\boldsymbol{x}) > f(\boldsymbol{x}^+)) = \Phi(Z)$$

Practical Bayesian Optimization

Libraries:

- **scikit-optimize:** Simple BO for hyperparameter tuning
- **Optuna:** Modern, efficient hyperparameter optimization
- **Hyperopt:** Tree-structured Parzen estimator (TPE)
- **GPyOpt, BoTorch:** Advanced GP-based optimization
- **Ax/BoTorch (Facebook):** Production-ready BO platform

Best practices:

- Use log scale for hyperparameters spanning orders of magnitude
- Parallel evaluations: batch Bayesian optimization
- Early stopping: prune unpromising trials
- Transfer learning: Use prior knowledge from related tasks

When to use Bayesian optimization:

- Function evaluations are expensive
- Derivatives not available
- Low to moderate dimensionality ($d < 20$)

Choosing an Optimization Algorithm

Scenario	**Method**	**Reason**
Convex, smooth, small n	Newton/L-BFGS	Fast convergence
Convex, large n	SGD/Adam	Scalable
Deep learning	Adam/AdamW	Adaptive, robust
Non-smooth (e.g., Lasso)	Proximal gradient	Handles non-smoothness
Constrained, convex	Interior point	Polynomial time
Black-box, expensive	Bayesian opt.	Sample efficient
Non-convex, no gradients	CMA-ES, DE	Global search
Combinatorial	GA, ACO, tabu	Discrete optimization

Regularization in Optimization

Purpose: Prevent overfitting, improve generalization.

ℓ_2 **regularization (Ridge):**

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- Shrinks weights toward zero
- Closed-form solution for linear models

ℓ_1 **regularization (Lasso):**

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_1$$

- Induces sparsity (some weights exactly zero)
- Performs feature selection

Elastic Net:

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) + \lambda_1\|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{w}\|_2^2$$

Learning Rate Schedules

Constant: $\alpha_k = \alpha$

- Simple, but may not converge or converge slowly

Step decay: $\alpha_k = \alpha_0 \cdot \gamma^{\lfloor k/s \rfloor}$

- Reduce by factor γ every s epochs
- Common: $\gamma = 0.1$, $s = 30$ epochs

Exponential decay: $\alpha_k = \alpha_0 e^{-\lambda k}$

- Smooth decay

Cosine annealing: $\alpha_k = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min})(1 + \cos(\frac{k\pi}{K}))$

- Popular in deep learning
- Can restart periodically (SGDR)

Warm restarts: Periodically reset learning rate to high value

- Helps escape local minima

Debugging Optimization

Common Issues

1. **Loss not decreasing:**
   - Learning rate too high or too low
   - Bug in gradient computation (use gradient checking!)
   - Inappropriate initialization
2. **Loss oscillating:**
   - Learning rate too high
   - Reduce learning rate or use adaptive methods
3. **Slow convergence:**
   - Learning rate too low
   - Poor conditioning (try normalization, preconditioning)
   - Use momentum or adaptive methods

Monitoring and Visualization

What to plot:

- **Loss curves:** Training and validation loss vs epochs/iterations
- **Gradient norms:** Check for vanishing/exploding gradients
- **Parameter norms:** Monitor weight magnitudes
- **Learning rate:** Track current learning rate
- **Evaluation metrics:** Accuracy, F1, etc.

Tools:

- **TensorBoard:** Comprehensive visualization for TensorFlow/PyTorch
- **Weights & Biases:** Experiment tracking and visualization
- **MLflow:** Model tracking and management
- **matplotlib/seaborn:** Custom plotting

Best practices:

- Log frequently but not excessively (every N iterations)
- Save checkpoints regularly
- Track hyperparameters with results

Example: Training a Model with PyTorch

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Define model, loss, optimizer
model = MyNeuralNetwork()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)

# Training loop
for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        # Forward pass
        output = model(data)
        loss = criterion(output, target)

        # Backward pass and optimization
        optimizer.zero_grad()  # Clear gradients
        loss.backward()        # Compute gradients

        # Gradient clipping (optional, prevents exploding gradients)
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.step()       # Update parameters
        total_loss += loss.item()

```

Multi-Objective Optimization

Problem: Optimize multiple conflicting objectives simultaneously.

$$\min_{\boldsymbol{x}} \{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\}$$

Pareto Optimality

$\boldsymbol{x}^*$ is Pareto optimal if there exists no $\boldsymbol{x}$ such that:

- $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*)$ for all i
- $f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*)$ for at least one j

The set of all Pareto optimal solutions forms the **Pareto front**.

Methods:

- **Scalarization:** $\min \sum_i w_i f_i(\boldsymbol{x})$ with weights w_i
- ϵ**-constraint:** Optimize one objective, constrain others
- **NSGA-II:** Non-dominated Sorting Genetic Algorithm

Distributed and Parallel Optimization

Data parallelism:

- Split data across workers
- Each worker computes gradient on its subset
- Aggregate gradients and update parameters

Model parallelism:

- Split model across devices (for very large models)
- Pipeline parallelism: different layers on different devices

Synchronous vs asynchronous:

- **Synchronous SGD:** Wait for all workers before updating
- **Asynchronous SGD:** Workers update independently (may be stale)

Communication efficiency:

- Gradient compression
- Local SGD: multiple local updates before synchronization
- AllReduce operations for efficient gradient aggregation

Federated Learning

Setting: Optimize model across decentralized data (e.g., mobile devices).

Federated Averaging (FedAvg)

Server:

1. Initialize global model $\boldsymbol{w}_0$
2. **For each round** $t = 1, 2, \ldots$:
   - Select subset of clients
   - Send current $\boldsymbol{w}_t$ to selected clients
   - Receive local updates from clients
   - Average updates: $\boldsymbol{w}_{t+1} = \sum_k \frac{n_k}{n} \boldsymbol{w}_k^{(t+1)}$

Client k:

- Perform local SGD on local data for E epochs
- Send updated model to server

Neural Architecture Search (NAS)

Goal: Automatically find optimal neural network architecture.

Search space:

- Number of layers, layer types
- Number of neurons/filters
- Connections between layers
- Activation functions

Methods:

- **Reinforcement learning:** Train controller to generate architectures
- **Evolutionary algorithms:** Evolve architectures
- **Gradient-based:** DARTS (differentiable architecture search)
- **Bayesian optimization:** Model architecture performance

Efficiency techniques:

- Weight sharing across architectures
- Early stopping of poor architectures
- [illegible]

Summary

Key Concepts:

- Convex vs non-convex optimization
- Gradient descent and variants (SGD, momentum, Adam)
- Constrained optimization (Lagrangian, KKT conditions)
- Evolutionary and metaheuristic algorithms
- Bayesian optimization for hyperparameter tuning
- Practical considerations (regularization, learning rates, debugging)

The Optimization Hierarchy

1. If convex and small: Use Newton/L-BFGS
2. If convex and large: Use (mini-batch) SGD with momentum/Adam
3. If non-convex with gradients: Use Adam/AdamW with proper initialization
4. If black-box, expensive: Use Bayesian optimization
5. If no gradients or discrete: Use evolutionary/metaheuristic algorithms

Practical Advice

Best Practices

1. **Start simple:** Try Adam with default settings
2. **Normalize data:** Zero mean, unit variance
3. **Use validation set:** Monitor for overfitting
4. **Visualize:** Plot loss curves, gradients
5. **Gradient checking:** Verify implementation with finite differences
6. **Tune hyperparameters:** Learning rate most important
7. **Use regularization:** Weight decay, dropout, early stopping
8. **Be patient:** Deep models may need many epochs

Resources:

- Boyd & Vandenberghe: ”Convex Optimization” (2004)
- Nocedal & Wright: ”Numerical Optimization” (2006)

Future Directions

Emerging trends:

- **AutoML:** Automated machine learning pipelines
- **Neural ODEs:** Continuous-depth models, adjoint method
- **Meta-learning:** Learning to optimize, learned optimizers
- **Sparse optimization:** Efficient training of large models
- **Quantum optimization:** Quantum algorithms for optimization
- **Robust optimization:** Handling uncertainty and adversaries

Final Thought

Optimization is at the heart of machine learning and data science. Understanding optimization algorithms, their properties, and when to use them is essential for:

- Training models effectively
- Solving real-world problems
- Pushing the boundaries of what's possible with AI

Acknowledgments

- ESMAD for institutional support
- Mysense.ai for real-world optimization challenges
- The optimization research community
- Open-source contributors (PyTorch, TensorFlow, scikit-learn)

Generated with LaTeX Beamer
Theme: ESMAD Professional Academic Style

Contact Information

Diogo Ribeiro

ESMAD - Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

dfr@esmad.ipp.pt

0009-0001-2022-7072

github.com/diogoribeiro7

linkedin.com/in/diogoribeiro7