TC2: Optimization for Machine Learning

Master of Science in AI and Master of Science in Data Science

@ UPSaclay

2025/2026.

Faïcel Chamroukhi

université PARIS-SACLAY

SystemX INSTITUT DE RECHERCHE TECHNOLOGIQUE

chamroukhi.com

week 6 : December 11, 2025.

Stochastic optimization, Non-convex optimization
(Stochastic Gradient, The EM Algorithm)

Stochastic Optimization

- Stochastic optimization refers to optimization techniques that incorporate randomness to handle uncertainty in :
  - Data (e.g., large-scale datasets).
  - Models (e.g., probabilistic or latent variable models).
  - The optimization process itself.
-
  - **Data Sampling :** Operates on random subsets of data (e.g., Stochastic Gradient Descent).
  - **Data Distribution :** Estimating the distribution of the data (potentially unobserved variables , e.g., Expectation-Maximization).
- Unlike deterministic methods, stochastic optimization uses probabilistic techniques to find optimal solutions

Eg. :

- **Gradient Descent :** Handles large datasets by using sampled gradients.
- **EM Algorithm :** Handles naturally and explicitly latent variables : Alternates between estimating latent variables and optimizing parameters.

Stochastic Optimization

université PARIS-SACLAY

- Stochastic Gradient Descent

Stochastic Gradient Descent

université PARIS-SACLAY

- Consider minimizing an average of functions :

$$\min_x \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

- Gradient Descent Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(x^{(k)})$$

- Stochastic (or Incremental) Gradient Descent (SGD) Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \nabla f_{i_k}(x^{(k)})$$

- i_k is chosen at each iteration, using :
  - **Randomized Rule :** Choose i_k uniformly at random.
  - **Cyclic Rule :** Iterate over $i_k = 1, 2, \ldots, m$ cyclically.

Stochastic Gradient Descent

université PARIS-SACLAY

- Consider minimizing an average of functions :

$$\min_x \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

- Gradient Descent Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(x^{(k)})$$

- Stochastic (or Incremental) Gradient Descent (SGD) Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \nabla f_{i_k}(x^{(k)})$$

- i_k is chosen at each iteration, using :
  - **Randomized Rule :** Choose i_k uniformly at random.
  - **Cyclic Rule :** Iterate over $i_k = 1, 2, \ldots, m$ cyclically.

Stochastic Gradient Descent

université PARIS-SACLAY

- Consider minimizing an average of functions :

$$\min_x \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

- Gradient Descent Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(x^{(k)})$$

- Stochastic (or Incremental) Gradient Descent (SGD) Update :

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \nabla f_{i_k}(x^{(k)})$$

- i_k is chosen at each iteration, using :
  - **Randomized Rule :** Choose i_k uniformly at random.
  - **Cyclic Rule :** Iterate over $i_k = 1, 2, \ldots, m$ cyclically.

Choosing the index i_k in SGD

université PARIS-SACLAY

Two rules for choosing i_k at iteration k :

- **Randomized Rule :** Choose $i_k \in \{1, \ldots, m\}$ uniformly at random.
- **Cyclic Rule :** Choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$.

- The **Randomized Rule** is more common in practice.
- For the randomized rule :

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x),$$

meaning SGD uses an unbiased estimate of the gradient at each step.
(see next slide)

Main appeal of SGD :

- Iteration cost is independent of m (number of functions).
- Saves memory by processing one sample (or function) at a time.
  Avoids storing the entire dataset in memory.

Choosing the index i_k in SGD

université PARIS-SACLAY

Two rules for choosing i_k at iteration k :

- **Randomized Rule :** Choose $i_k \in \{1, \ldots, m\}$ uniformly at random.
- **Cyclic Rule :** Choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$.

- The **Randomized Rule** is more common in practice.
- For the randomized rule :

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x),$$

meaning SGD uses an unbiased estimate of the gradient at each step.
(see next slide)

Main appeal of SGD :

- Iteration cost is independent of m (number of functions).
- Saves memory by processing one sample (or function) at a time.
  Avoids storing the entire dataset in memory.

SGD with randomized choice

université PARIS-SACLAY

- SGD Objective and gradient :
  $f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x); \nabla f(x) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x).$
- Randomized rule, i.e choosing i_k uniformly, i.e. $i_k \sim \mathcal{U}([1, 2, \ldots, m])$:
  $\mathbb{P}(i_k = i) = \frac{1}{m}, \forall i \in \{1, 2, \ldots, m\}.$
- **Expected value of the Stochastic Gradient**
  - The stochastic gradient $\nabla f_{i_k}(x)$ is a random variable because i_k is selected randomly.
  - Its expectation : $\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^{m} \mathbb{P}(i_k = i)\nabla f_i(x).$
  - Substituting $\mathbb{P}(i_k = i) = \frac{1}{m}$, we have :
    $\mathbb{E}[\nabla f_{i_k}(x)] = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x) = \nabla f(x).$
- Hence $\nabla f_{i_k}(x)$ is an *unbiased estimator* of the full gradient $\nabla f(x)$.

  $\hookrightarrow$ (but the variance ... !)
- $\hookrightarrow$ Instead of calculating the full gradient $\nabla f(x)$, SGD approximates it using a single component gradient $\nabla f_{i_k}(x)$, where i_k is chosen randomly at each iteration k.

SGD with randomized choice

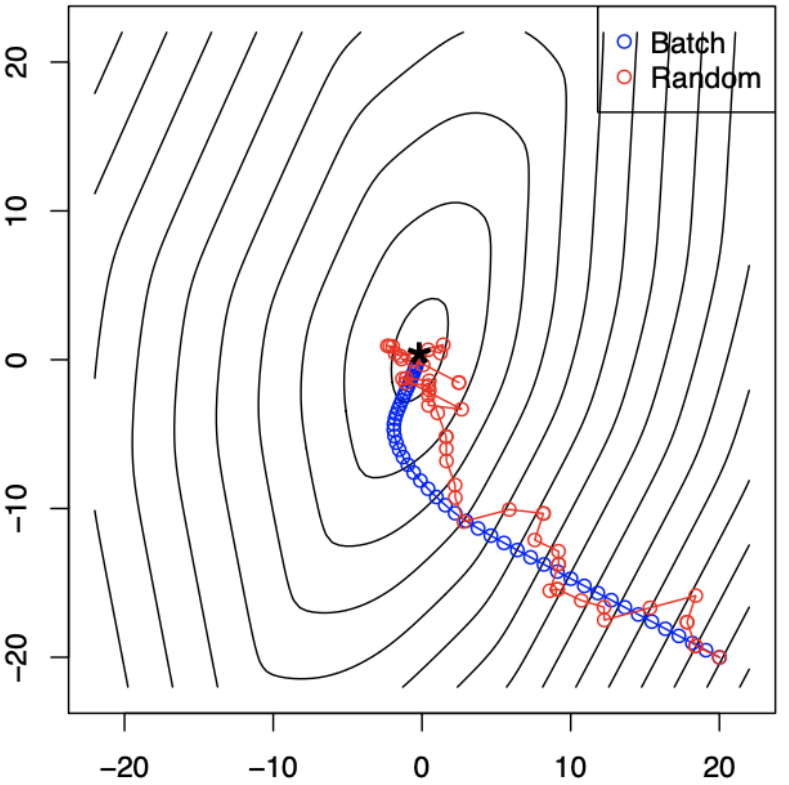
université PARIS-SACLAY

- SGD Objective and gradient :
  $f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x); \nabla f(x) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x).$
- Randomized rule, i.e choosing i_k uniformly, i.e. $i_k \sim \mathcal{U}([1, 2, \ldots, m])$:
  $\mathbb{P}(i_k = i) = \frac{1}{m}, \forall i \in \{1, 2, \ldots, m\}.$
- **Expected value of the Stochastic Gradient**
  - The stochastic gradient $\nabla f_{i_k}(x)$ is a random variable because i_k is selected randomly.
  - Its expectation : $\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^{m} \mathbb{P}(i_k = i)\nabla f_i(x).$
  - Substituting $\mathbb{P}(i_k = i) = \frac{1}{m}$, we have :
    $\mathbb{E}[\nabla f_{i_k}(x)] = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x) = \nabla f(x).$
- Hence $\nabla f_{i_k}(x)$ is an *unbiased estimator* of the full gradient $\nabla f(x)$.
  $\hookrightarrow$ (but the variance ... !)
- $\hookrightarrow$ Instead of calculating the full gradient $\nabla f(x)$, SGD approximates it using a single component gradient $\nabla f_{i_k}(x)$, where i_k is chosen randomly at each iteration k.

SGD with randomized choice

- SGD Objective and gradient :
  $f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x); \nabla f(x) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x).$
- Randomized rule, i.e choosing i_k uniformly, i.e. $i_k \sim \mathcal{U}([1, 2, \ldots, m])$:
  $\mathbb{P}(i_k = i) = \frac{1}{m}, \forall i \in \{1, 2, \ldots, m\}.$
- **Expected value of the Stochastic Gradient**
  - The stochastic gradient $\nabla f_{i_k}(x)$ is a random variable because i_k is selected randomly.
  - Its expectation : $\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^{m} \mathbb{P}(i_k = i)\nabla f_i(x).$
  - Substituting $\mathbb{P}(i_k = i) = \frac{1}{m}$, we have :
    $\mathbb{E}[\nabla f_{i_k}(x)] = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x) = \nabla f(x).$
- Hence $\nabla f_{i_k}(x)$ is an *unbiased estimator* of the full gradient $\nabla f(x)$.

  $\hookrightarrow$ (but the variance ... !)
- $\hookrightarrow$ Instead of calculating the full gradient $\nabla f(x)$, SGD approximates it using a single component gradient $\nabla f_{i_k}(x)$, where i_k is chosen randomly at each iteration k.



example with $n = 10$, $p = 2$ dimensions, to show the behaviour for batch versus stochastic gradient

Stochastic methods generally :

- perform well *far from the optimum* : fast progress with noisy but informative gradients.
- perform poorly *near the optimum* : high variance in gradients causes oscillations and slower convergence.

Example : SGD for Logistic Regression

université PARIS-SACLAY

Problem : Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$, logistic reg. objective :

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left[-y_i x_i^T \theta + \log\left(1 + \exp(x_i^T \theta)\right)\right]}_{f_i(\theta)}.$$

Gradient computation :

- $\nabla f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - p_i(\theta)\right) x_i$, where $p_i(\theta) = \frac{\exp(x_i^T \theta)}{1+\exp(x_i^T \theta)}$.
- Feasible when n (number of data points) is moderate.
- Computationally expensive when n is very large.

Cost Comparison :

- Full gradient (batch update) : $O(np)$.
- Stochastic gradient update : $O(p)$.
- Eg., Computing much more Stochastic steps is significantly more affordable than computing the full gradient for each update.
- But slower convergence rate (ie. stochastic noise)

Example : SGD for Logistic Regression

université PARIS-SACLAY

Problem : Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$, logistic reg. objective :

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left[-y_i x_i^T \theta + \log\left(1 + \exp(x_i^T \theta)\right)\right]}_{f_i(\theta)}.$$

Gradient computation :

- $\nabla f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - p_i(\theta)\right) x_i$, where $p_i(\theta) = \frac{\exp(x_i^T \theta)}{1+\exp(x_i^T \theta)}$.
- Feasible when n (number of data points) is moderate.
- Computationally expensive when n is very large.

Cost Comparison :

- Full gradient (batch update) : $O(np)$.
- Stochastic gradient update : $O(p)$.
- Eg., Computing much more Stochastic steps is significantly more affordable than computing the full gradient for each update.
- But slower convergence rate (ie. stochastic noise)

Example : SGD for Logistic Regression

université PARIS-SACLAY

Problem : Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$, logistic reg. objective :

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left[-y_i x_i^T \theta + \log\left(1 + \exp(x_i^T \theta)\right)\right]}_{f_i(\theta)}.$$

Gradient computation :

- $\nabla f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - p_i(\theta)\right) x_i$, where $p_i(\theta) = \frac{\exp(x_i^T \theta)}{1+\exp(x_i^T \theta)}$.
- Feasible when n (number of data points) is moderate.
- Computationally expensive when n is very large.

Cost Comparison :

- Full gradient (batch update) : $O(np)$.
- Stochastic gradient update : $O(p)$.
- Eg., Computing much more Stochastic steps is significantly more affordable than computing the full gradient for each update.
- But slower convergence rate (ie. stochastic noise)

Step Sizes in SGD

high variance in SGD leads to :
- Noisy updates
- The need for decreasing step sizes or variance-reduction methods

Step size (α_k) :

- Step size (α_k) controls the magnitude of each update in stochastic gradient descent (SGD).
- Standard practice : Use **diminishing step sizes** : Common forms : $\alpha_k = \frac{1}{k}$, $\alpha_k = \frac{\alpha_0}{1+\lambda k}$ or $\alpha_k = \frac{\alpha_0}{k}$ with (α_0) to be tuned
- Diminishing step sizes :
  - Gradually reduce the impact of noisy gradients.
  - Ensure that the difference between stochastic and full gradient steps vanishes over time.

Convergence Rates of SGD

université PARIS-SACLAY

- For convex $f(x)$, SGD with diminishing step sizes satisfies :

$$\mathbb{E}[f(x^{(k)})] - f^* = O\left(\frac{1}{\sqrt{k}}\right) \quad \text{(sublinear rate for convex SGD)}$$

- When f is μ-strongly convex and has L-Lipschitz gradients :

$$\mathbb{E}[f(x^{(k)})] - f^* = O\left(\frac{1}{k}\right) \quad \text{(faster sublinear rate for strongly convex SGD)}$$

- Due to gradient noise, SGD achieves only sublinear convergence (unlike deterministic methods).
- **Comparison with Gradient Descent (GD) :**
  - GD (with fixed step size) under strong convexity achieves linear convergence :

$$f(x^{(k)}) - f^* = O(\rho^k), \quad \rho \in (0, 1)$$

  - SGD lacks this linear rate due to noisy updates.
- $\hookrightarrow$ Noisy gradient estimates introduce variance and slow down convergence near the optimum.

Strategies to Improve SGD :

- for example **Mini-Batching :** Reduces variance by using a small batch of data points.

Mini-Batches in SGD

In **Mini-batch** stochastic gradient descent, we choose a random subset $I_k \subset \{1, ..., m\}$, with $\#I_k = b \ll m$, and update :

- Updating rule : $x^{(k+1)} = x^{(k)} - \alpha_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k)})$
- Benefits :
  - Reduces gradient variance by a factor of $\frac{1}{b}$.
  - Enables parallel computation (e.g., GPUs).
- Trade-off :
  - Mini-batches are b-times more expensive per iteration.
  - But reduce variance and stabilize convergence.
- Convergence rate :

$$\mathbb{E}[f(x^{(k)})] - f^* = O\left(\frac{1}{\sqrt{bk}}\right)$$

- (Sublinear rate ; faster than standard SGD by a factor of $\sqrt{b}$)
- In the strongly convex and smooth case, the rate improves to $O\left(\frac{1}{bk}\right)$.

Summary

- SGD is efficient for large-scale optimization.
- Convergence rates are slower than full gradient methods.
- Mini-batches and (other techniques eg. early stopping) are practical techniques for improving SGD.

SGD is widely used in machine learning for its simplicity and scalability.

Stochastic Optimization

université PARIS-SACLAY

- The EM algorithm

The EM Algorithm

université PARIS-SACLAY

Purpose :

- Solve maximum likelihood estimation (MLE) problems for latent variable models : probabilistic models with parameters θ, observed variables X, latent variables Z

  Goal :

  $$\hat{\theta} = \arg\max_{\theta} \log p(X \mid \theta),$$

  where $\log p(X \mid \theta)$ is the observed data log-likelihood.

- Iteratively optimize the likelihood function $\log p(X \mid \theta)$.

Key Idea : Exploit the observed data X and latent (unobserved) data Z in the construction of the optimization process :

- Alternately estimate :
  1. Compute an expectation of the log-likelihood assuming the latent variables Z are available
  2. Maximize the resulting expectation w.r.t the model parameters θ.

The EM Algorithm

université PARIS-SACLAY

Purpose :

- Solve maximum likelihood estimation (MLE) problems for latent variable models : probabilistic models with parameters θ, observed variables X, latent variables Z

  Goal :

  $$\hat{\theta} = \arg\max_{\theta} \log p(X \mid \theta),$$

  where $\log p(X \mid \theta)$ is the observed data log-likelihood.

- Iteratively optimize the likelihood function $\log p(X \mid \theta)$.

Key Idea : Exploit the observed data X and latent (unobserved) data Z in the construction of the optimization process :

- Alternately estimate :
  1. Compute an expectation of the log-likelihood assuming the latent variables Z are available
  2. Maximize the resulting expectation w.r.t the model parameters θ.

The EM Algorithm

Steps in the EM Algorithm

université PARIS-SACLAY

1. **Initialize :** Start with an initial estimate $\theta^{(0)}$.
2. **Repeat until Convergence :**
   - **E-Step :** Compute the expected complete-data log-likelihood :

$$Q(\theta \mid \theta^{(k)}) = \mathbb{E}_{Z \sim p(Z|X,\theta^{(k)})}[\log p(X, Z \mid \theta)].$$

   - **M-Step :** Maximize $Q(\theta \mid \theta^{(k)})$ to update θ :

$$\theta^{(k+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(k)}).$$

Convergence Properties of EM

université PARIS-SACLAY

- The EM algorithm ensures that the observed data log-likelihood $\log p(X \mid \theta)$ increases at every iteration.
- EM converges to a stationary point of the log-likelihood (not necessarily a global maximum).

Advantages :

- Handles missing or latent data efficiently.
- Straightforward to implement for many problems.

Advantages and Limitations of EM

université PARIS-SACLAY

Advantages :

- Handles latent variables naturally.
- Straightforward implementation for many probabilistic models.
- Widely used in probabilistic machine learning

Limitations :

- Generally used for non-convex problems
- Converges but may converge to a local optimum instead of the global optimum.
- Slow convergence near the optimum.
- Sensitive to initialization of parameters.

Finite Mixture Models

$f(x;\theta) = \sum_{j=1}^{m} \pi_j f_j(x;\theta_j)$ with $\pi_j > 0 \ \forall j$ and $\sum_{j=1}^{m} \pi_j = 1$.

Maximum-Likelihood Estimation

$$\widehat{\theta} \in \arg\max_{\theta} \log L(\theta)$$

log-likelihood : $\log L(\theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{m} \pi_j f_j(x_j;\theta_j)$.

The EM algorithm

$$\theta^{new} \in \arg\max_{\theta \in \Omega} \mathbb{E}[\log L_c(\theta) | \mathcal{D}, \theta^{old}]$$

completed-data log-likelihood : $\log L_c(\theta) = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log [\pi_j f_j(x_i;\theta_j)]$
where Z_{ij} is such that $Z_{ij} = 1$ if $Z_i = j$ and $Z_{ij} = 0$ otherwise.

Finite Mixture Models

$f(x;\theta) = \sum_{j=1}^{m} \pi_j f_j(x;\theta_j)$ with $\pi_j > 0 \ \forall j$ and $\sum_{j=1}^{m} \pi_j = 1$.

Maximum-Likelihood Estimation

$$\widehat{\theta} \in \arg\max_\theta \log L(\theta)$$

log-likelihood : $\log L(\theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{m} \pi_j f_j(x_j;\theta_j)$.

The EM algorithm

$$\theta^{new} \in \arg\max_{\theta\in\Omega} \mathbb{E}[\log L_c(\theta)|\mathcal{D}, \theta^{old}]$$

completed-data log-likelihood : $\log L_c(\theta) = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log [\pi_j f_j(x_i;\theta_j)]$
where Z_{ij} is such that $Z_{ij} = 1$ if $Z_i = j$ and $Z_{ij} = 0$ otherwise.

Finite Mixture Models

$f(x;\theta) = \sum_{j=1}^{m} \pi_j f_j(x;\theta_j)$ with $\pi_j > 0 \ \forall j$ and $\sum_{j=1}^{m} \pi_j = 1$.

Maximum-Likelihood Estimation

$$\widehat{\theta} \in \arg\max_\theta \log L(\theta)$$

log-likelihood : $\log L(\theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{m} \pi_j f_j(x_j;\theta_j)$.

The EM algorithm

$$\theta^{new} \in \arg\max_{\theta\in\Omega} \mathbb{E}[\log L_c(\theta)|\mathcal{D}, \theta^{old}]$$

completed-data log-likelihood : $\log L_c(\theta) = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log [\pi_j f_j(x_i;\theta_j)]$
where Z_{ij} is such that $Z_{ij} = 1$ if $Z_i = j$ and $Z_{ij} = 0$ otherwise.

Gaussian mixture models (GMMs)

The finite Gaussian mixture density is defined as :

$$f(x_i;\theta) = \sum_{j=1}^{m} \pi_j \mathcal{N}(x_i;\mu_j,\Sigma_j)$$

with $\mathcal{N}(x_i;\mu_j,\Sigma_j) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}\left(x_i-\mu_j\right)^T \Sigma_j^{-1}\left(x_i-\mu_j\right)\right)$,
$\pi_j > 0\ \forall j$ and $\sum_{j=1}^{m} \pi_j = 1$.

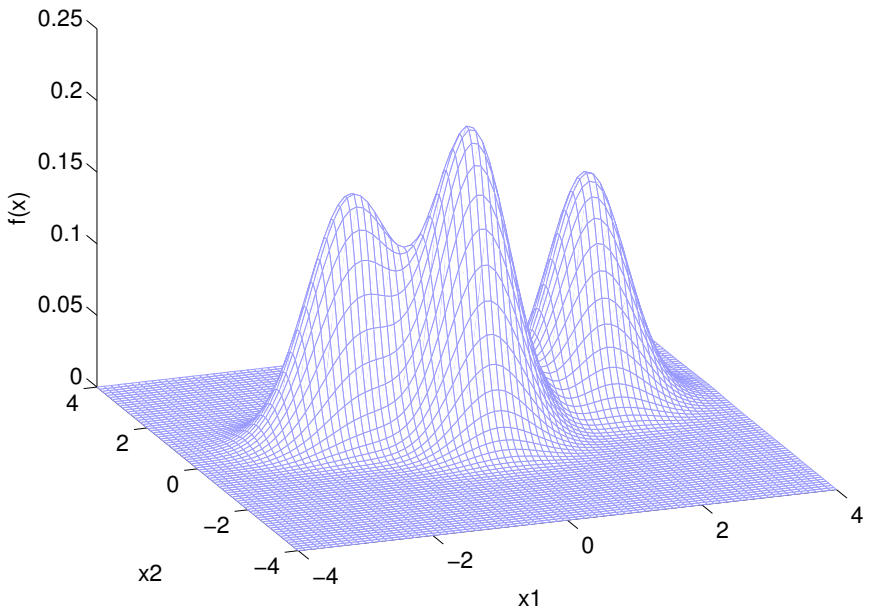


FIGURE – An example of a three-component Gaussian mixture density in $\mathbb{R}^2$.

EM for Gaussian mixture models

université PARIS-SACLAY

1. **E-Step** : calculates the posterior component memberships :

$$\tau_{ij}^{(k)} = \mathbb{P}(Z_i = j | x_i, \theta^{(k)}) = \frac{\pi_j \mathcal{N}(x_i; \mu_j^{(k)}, \Sigma_j^{(k)})}{\sum_{\ell=1}^{m} \pi_\ell \mathcal{N}(x_i; \mu_\ell^{(k)}, \Sigma_\ell^{(k)})}$$

that x_i originates from the kth component density.

2. **M-Step** : parameter updates :

$$\begin{aligned}
\pi_j^{(k+1)} &= \frac{\sum_{i=1}^{n} \tau_{ij}^{(k)}}{n} = \frac{n_j^{(k)}}{n}, \\
\mu_j^{(k+1)} &= \frac{1}{n_j^{(k)}} \sum_{i=1}^{n} \tau_{ij}^{(k)} x_i, \\
\Sigma_j^{(k+1)} &= \frac{1}{n_j^{(k)}} \sum_{i=1}^{n} \tau_{ij}^{(k)} (x_i - \mu_j^{(k+1)})(x_i - \mu_j^{(k+1)})^T.
\end{aligned}$$

Proofs : as an exercice

EM for Gaussian mixture models

université PARIS-SACLAY

1. **E-Step** : calculates the posterior component memberships :

$$\tau_{ij}^{(k)} = \mathbb{P}(Z_i = j | x_i, \theta^{(k)}) = \frac{\pi_j \mathcal{N}(x_i; \mu_j^{(k)}, \Sigma_j^{(k)})}{\sum_{\ell=1}^{m} \pi_\ell \mathcal{N}(x_i; \mu_\ell^{(k)}, \Sigma_\ell^{(k)})}$$

that x_i originates from the kth component density.

2. **M-Step** : parameter updates :

$$\begin{aligned}
\pi_j^{(k+1)} &= \frac{\sum_{i=1}^{n} \tau_{ij}^{(k)}}{n} = \frac{n_j^{(k)}}{n}, \\
\mu_j^{(k+1)} &= \frac{1}{n_j^{(k)}} \sum_{i=1}^{n} \tau_{ij}^{(k)} x_i, \\
\Sigma_j^{(k+1)} &= \frac{1}{n_j^{(k)}} \sum_{i=1}^{n} \tau_{ij}^{(k)} (x_i - \mu_j^{(k+1)})(x_i - \mu_j^{(k+1)})^T.
\end{aligned}$$

Proofs : as an exercice

The EM algorithm

$$\theta^{new} \in \arg\max_{\theta \in \Omega} \mathbb{E}[\log L_c(\theta) | \mathcal{D}, \theta^{old}]$$

completed-data log-likelihood :

$\log L_c(\theta) = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log [\pi_j f_j(x_i; \theta_j)]$

where Z_{ij} is such that $Z_{ij} = 1$ if $Z_i = j$ and $Z_{ij} = 0$ otherwise.

completed-data log-likelihood :

$\log L_c(\theta) = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log \left[\pi_j \mathcal{N}(x_i; \mu_j^{(k)}, \Sigma_j^{(k)})\right] =$

$\sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log \pi_j + \sum_{i=1}^{n} \sum_{j=1}^{m} Z_{ij} \log \mathcal{N}(x_i; \mu_j^{(k)}, \Sigma_j^{(k)})$

So

- **E-Step :** Compute the expected complete-data log-likelihood :

$$Q(\theta \mid \theta^{(k)}) = \mathbb{E}_{Z \sim p(Z|X,\theta^{(k)})}[\log p(X, Z \mid \theta)].$$

$= \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbb{E}[Z_{ij} \mid x_i, \theta^{(k)}] \log \pi_j$

$+ \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbb{E}[Z_{ij} \mid x_i, \theta^{(k)}] \log \mathcal{N}(x_i; \mu_j^{(k)}, \Sigma_j^{(k)})$

M-Step

université PARIS-SACLAY

- **M-Step :** Maximize $Q(\theta \mid \theta^{(k)})$ to update θ :

$$\theta^{(k+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(k)}).$$

The M-step maximizes the expected complete-data log-likelihood, which is often easier than directly maximizing the marginal log-likelihood.

Proofs left as an exercice

université PARIS-SACLAY

E-Step (Expectation)

M-Step (parameter update :)

- For the mixture proportions π_j's : a constrained optimization problem
- for the mean and the covariance matrix : a weighted estimation of the standard multivariate gaussian

  Hints :

- For the mixture proportions π_j's, use Lagrange multipliers
- For the means μ_j's, use the fact that $\frac{\partial x^T A x}{x} = (A + A^T)x$
- For the covariance matrices Σ_j's, use standard results
  - $\frac{\partial \log |A|}{\partial A} = A^{-1}$
  - $x^T A x = \text{ trace } (x^T A x)$
  - $\text{trace}(x^T A x) = \text{trace}(x x^T A)$
  - $\frac{\partial \text{trace}(BA)}{A} = B^T$

Proofs left as an exercice

université PARIS-SACLAY

E-Step (Expectation)

M-Step (parameter update :)

- For the mixture proportions π_j's : a constrained optimization problem
- for the mean and the covariance matrix : a weighted estimation of the standard multivariate gaussian

  Hints :

- For the mixture proportions π_j's, use Lagrange multipliers
- For the means μ_j's, use the fact that $\frac{\partial x^T A x}{x} = (A + A^T)x$
- For the covariance matrices Σ_j's, use standard results
  - $\frac{\partial \log |A|}{\partial A} = A^{-1}$
  - $x^T A x =$ trace $(x^T A x)$
  - trace$(x^T A x) =$ trace$(x x^T A)$
  - $\frac{\partial \text{trace}(BA)}{A} = B^T$

Thank you for your attention !