KAZAKH-BRITISH TECHNICAL UNIVERSITY

Introduction to Machine Learning
Week 7

Olivier JAYLET

School of Information Technology and Engineering

AllianceFrançaise
Almaty

Open house
Saturday 1st March, from 11am to 6pm
Adress : Dostyk 160, Almaty

Introduction

A machine learning algorithm can be viewed as an optimization program.
Today, we will have a look at a common algorithm used to find the parameters that minimize a known cost function $f(\cdot)$: **the gradient descent algorithm.**

- **vanilla** version of the gradient descent algorithm.
- stochastic gradient
- mini-batch gradient

Let us consider a very general model:

$$y = f(X) + \varepsilon,$$

where y is a variable to predict (i.e. target variable or response variable), $f(\cdot)$ is an unknown model, X is a set of p predictors (i.e. features, or inputs, or explanatory variables) and ε is an error term.

For the example, we assume that the response variable is linearly dependent on the set of explanatory variables:

$$y = X\beta + \varepsilon.$$

We do not know the true generating data process and only observe some realizations of y and X for n observations.

When estimating a linear regression, we assume that the error term is normally distributed with zero mean and variance σ^2.

The vector of coefficients β can be estimated with **Ordinary Least Squares** (OLS).

The problem is to estimate the coefficients of vector β which **minimize an objective function**:

$$\arg\min_{\beta} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f(X_i)\right),$$

Here, with OLS, an analytical solution exists:

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T y.$$

Time complexity

Time complexity is a measure of the time used by an algorithm, expressed as a function of the size of the input. Time counts the number of calculation steps before arriving at a result.

OLS Complexity

Each steps of the solution are :

- $X^T X : O(nm^2)$,
- $(X^T X)^{-1} : O(m^3)$,
- $X^T : O(m^2 n)$,
- Final multiplication : $O(mn)$.

Dominant Term:

- For small n (small dataset): $O(m^3)$ (matrix inversion dominates).
- For large n (large dataset): $O(nm^2)$ (matrix multiplication dominates).

Effect of large dataset : $n \gg m$

When the number of samples (rows) n is much greater than the number of features m:

- The dominant time complexity is $O(nm^2)$, which comes from computing $X^T X$.
- This dependency on n makes OLS expensive for **large datasets**, as increasing n leads to a quadratic increase in computational cost.

Effect of High Dimensionality : $m \gg n$

When the number of features (columns m) is much greater than the number of samples (rows n):

- The dominant time complexity is $O(m^3)$, which comes from matrix inversion of $(X^T X)$, an $m \times m$ matrix.
- This makes dimensionality (number of features) a critical factor for computational feasibility.

Need of Numerical Solutions

Numerical optimization techniques are often used instead of analytical solutions :

Expensive Calculations of Analytical Solutions:

- Computing the closed-form solution involving operations like matrix multiplication and matrix inversion can be computationally expensive for large datasets or high-dimensional data.
- The complexity of these operations makes them infeasible for large-scale problems.

Need of Numerical Solutions

Applicability to Non-Linear Models:

- Analytical solutions are typically restricted to linear models. For non-linear models or cases where the loss function is not quadratic, numerical optimization methods, are required to minimize the loss function.
- These techniques are more flexible and can handle the iterative process needed to converge on a solution for non-linear and complex models.

Thus, numerical solutions provide scalability and adaptability for solving real-world machine learning problems efficiently.

Vanilla gradient descent

Let us illustrate this with a simple example.
Let us consider the following function:

$$f(x) = (x+3) \times (x-2)^2 \times (x+1).$$

The global minimum of that function is reached in

$$x = -1 - \sqrt{\frac{3}{2}}.$$

Let us generate some values from this process, for $x \in [-3, 3]$.

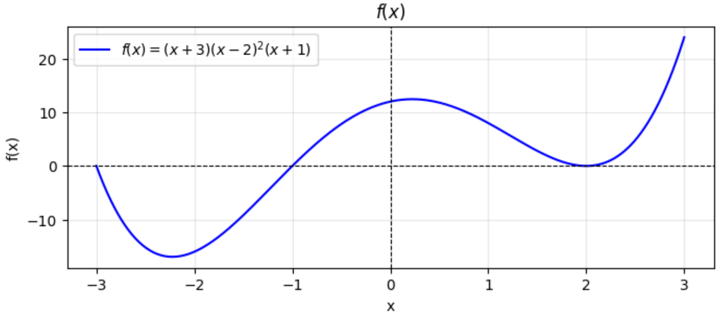

Figure: Function with a single input: a more complex function

Vanilla gradient descent

If we want to minimize this function using gradient descent, we can proceed as follows :

Initialization

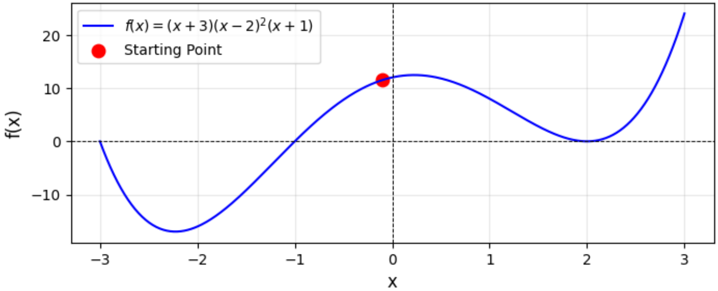

Figure: Initialize random point on the function

Vanilla gradient descent

Then, from that point, we need to decide on two things so as to reduce the objective function:

1. in which direction to go next (left or right)
2. and how far we want to go.

Direction

To decide the direction, we can compute the derivative of the function at this specific point of interest. The slope of the derivative will guide us:

- **if it is positive**: we need to shift to the left
- **if it is negative**: we need to shift to the right.

Direction

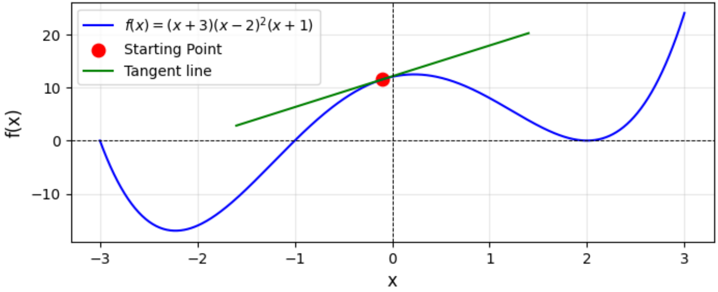

Figure: The derivative of $f(x) = (x+3) \times (x-2)^2 \times (x+1)$ at $x = -0.5$ is 12.5, i.e. the slope of the tangent is positive.

- Here, the slope is positive. We thus need to go left.
- We still need to decide how far we want to go, *i.e.*, we must decide the size of the step we will take.
- This step is called the **learning rate**:
    - On the one hand, if this learning rate is too small, we increase the risk of ending up in a local minimum.
    - On the other hand, if we pick a too large value for the learning rate, we face a risk of overshooting the minimum and keeping bouncing around a (local) minimum forever.

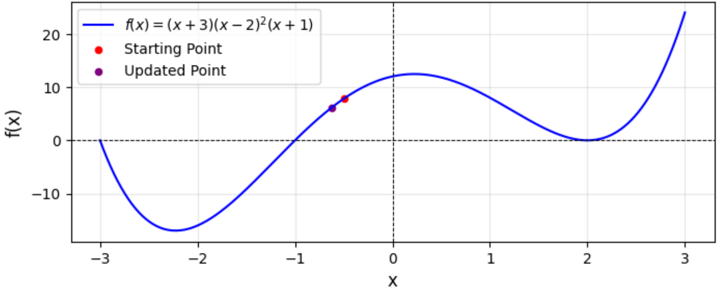

Figure: Create a new point

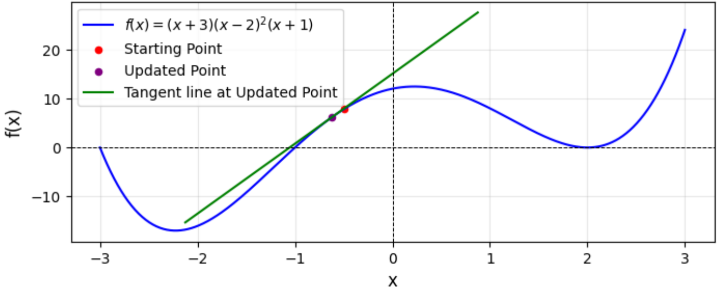

Figure: Update the derivative

Iterative Procedure

- Then, we can repeat the procedure multiple times through a loop.
- We will update our parameter from one iteration to the other and will stop either:
  - When a maximum number of iterations is reached
  - When the improvement (reduction in the objective function from one step to the next) is too small (below a threshold we will call **tolerance**).

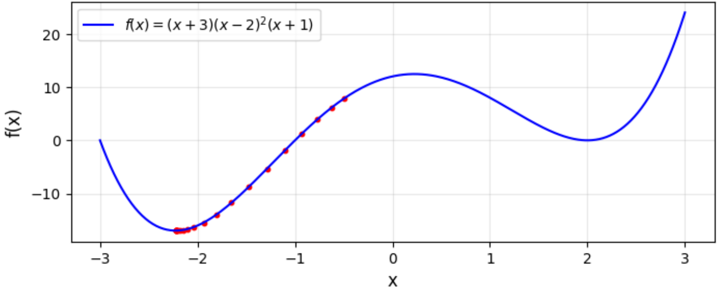

Figure: Process repeated into a loop

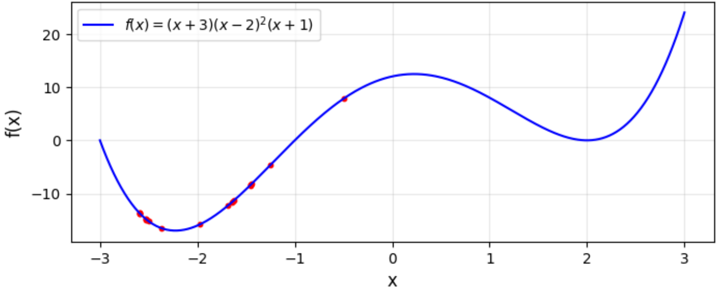

Figure: Too high learning rate (1)

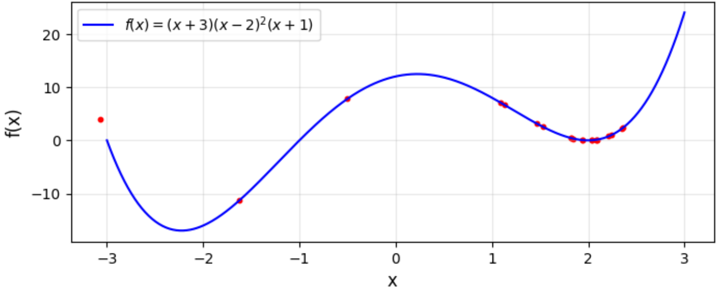

Figure: Too high learning rate (2)

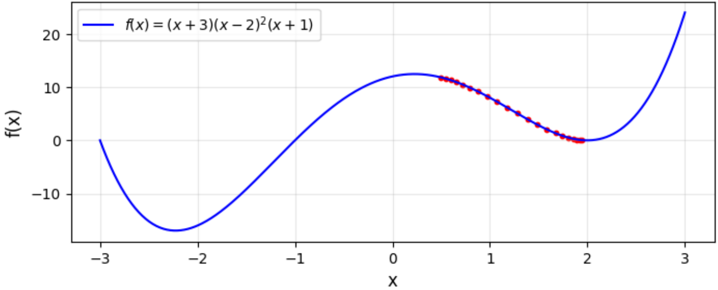

Figure: Dependence on the initial values

Higher Dimensions Optimization Problems

Let us consider the following data generating process:

$$f(x_1, x_2) = x_1^2 + x_2^2$$

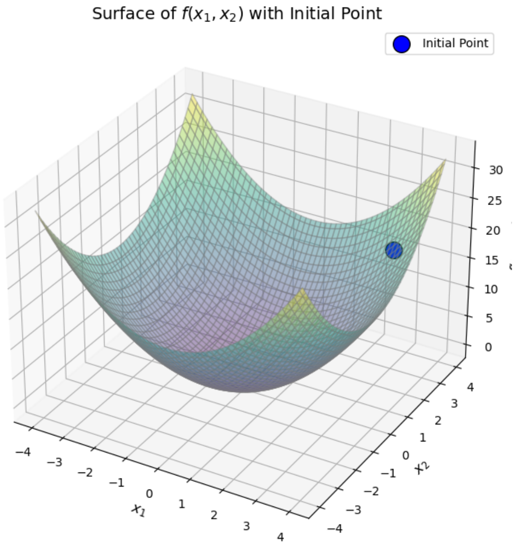

Figure: Finding a local minima

Intuition

From that point, we need to decide:

- **the direction** to go to
- **and the magnitude of the step** to take in that direction.

Computing the gradient

The direction is obtained by computing the first derivative of the objective function $f(\cdot)$ with respect to each argument x_1 and x_2 , at point $\boldsymbol{\theta}$. In other words, we need to evaluate the gradient of the function at point $\boldsymbol{\theta}$.

$$\nabla f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{\theta}) \\ \frac{\partial f}{\partial x_2}(\boldsymbol{\theta}) \end{bmatrix}$$

The values will give us the steepest ascent.

The updated value of the parameters after the end of the t th step will be:

$$\begin{bmatrix} x_1^{(t+1)} \\ x_2^{(t+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \end{bmatrix} - \eta \cdot \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1^{(t)}, x_2^{(t)}) \\ \frac{\partial f}{\partial x_2}(x_1^{(t)}, x_2^{(t)}) \end{bmatrix},$$

where

$\begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \end{bmatrix}$ is the current value of the vector of parameters,

$\eta \in \mathbb{R}^+$ is the learning rate, and

$\begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1^{(t)}, x_2^{(t)}) \\ \frac{\partial f}{\partial x_2}(x_1^{(t)}, x_2^{(t)}) \end{bmatrix}$ is the gradient of the function at point $\boldsymbol{\theta} = (x_1^{(t)}, x_2^{(t)})$.

In a more general context, the update rule becomes:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \cdot \nabla \mathcal{L}\big(\boldsymbol{\theta}^{(t)}\big),$$

where:

- η is the learning rate,
- $\nabla \mathcal{L}\big(\boldsymbol{\theta}^{(t)}\big)$ is the gradient of the loss function $\mathcal{L}$ at $\boldsymbol{\theta}^{(t)}$, and
- $\boldsymbol{\theta}$ denotes the vector of parameters being optimized.

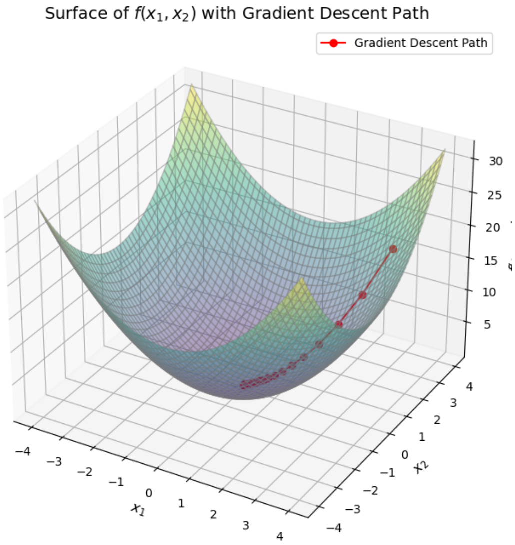

Figure: Finding a local minima

The gradient descent algorithm can be written as:

Algorithm Vanilla Gradient Descent

Input: $\mathcal{L}(x)$, η , $\theta^{(0)}$, T, ϵ
Initialize: $t \leftarrow 0$
repeat
    $g \leftarrow \nabla \mathcal{L}(\theta^{(t)})$
    $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \cdot g$
    $t \leftarrow t + 1$
until $||g|| < \epsilon$ OR $t > T$
Return: $\theta^{(t)}$

where, $\mathcal{L}(x)$ is a loss function, η is a learning rate, $\theta^{(0)}$ is the set of initial parameters, T is the maximum number of iterations, and ϵ is the tolerance.