

# Introduction to Artificial Intelligence

## Gradient Descent Method

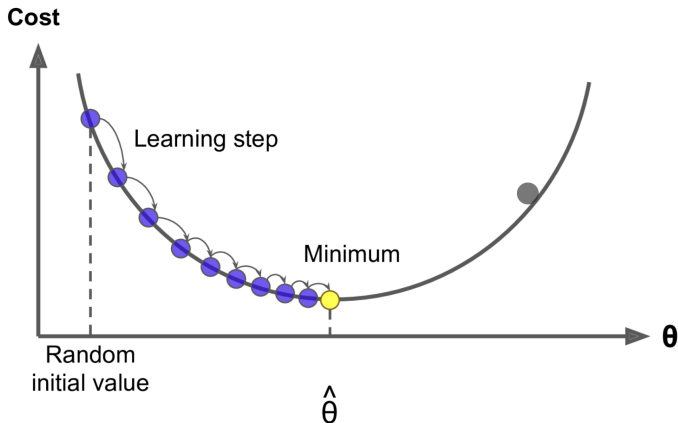
Ph.D. Gerardo Marx Chávez-Campos

Instituto Tecnológico de Morelia



# Gradient Descent

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to *tweak parameters iteratively in order to minimize a cost function*.



# Gradient (Grad)

The **Gradient** of a function  $f(x, y)$  in two dimensions is defined as:

$$\nabla f(x, y) = \frac{df}{dx} \hat{i} + \frac{df}{dy} \hat{j} \quad (1)$$

note that its component in the  $\hat{i}$  direction is the partial derivative of  $f$  with respect to  $x$ . This is the rate of change of  $f$  in the  $x$  direction since  $y$  is kept constant.

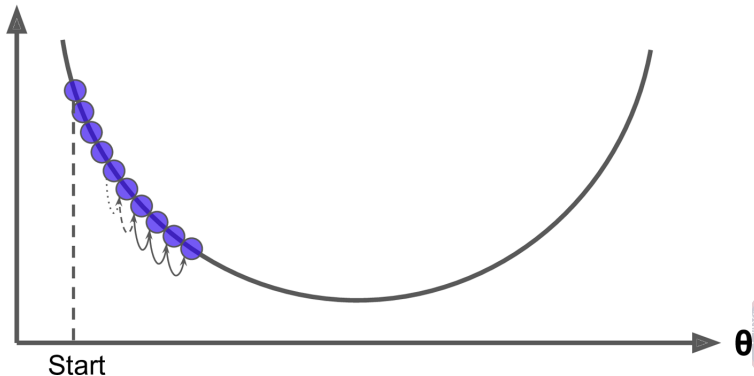


- ▶ Measures the local gradient (derivate) of the error function with regards to the parameter vector  $\theta$ , and it goes in the direction of descending gradient.
- ▶ Once the gradient is zero, you have reached a minimum.
- ▶ Concretely, you start with  $\theta$  with random values (this is called random initialization), and then you improve it gradually, taking one step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum.



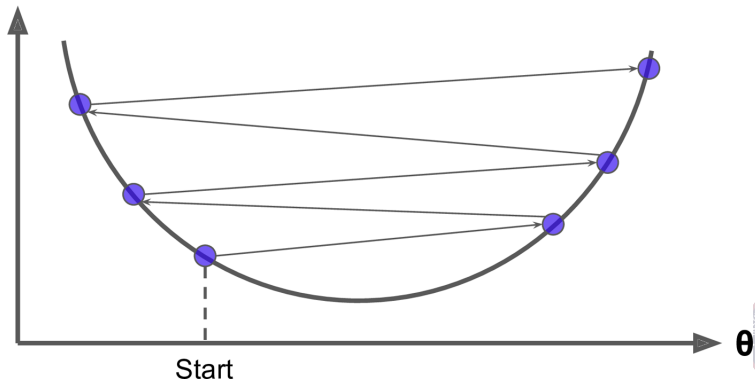
An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter  $\eta$ . If the learning rate is too small, then the algorithm will have to go through many iterations to **converge**, which will take a long time.

**Cost**



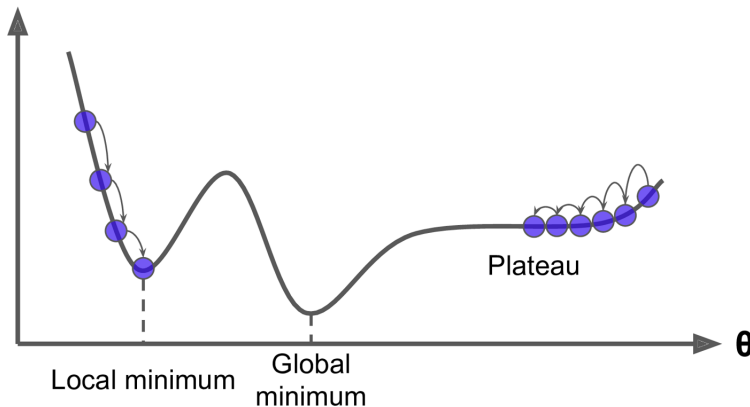
On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher than you were before. This might make the algorithm **diverge**, with larger and larger values, failing to find a good solution.

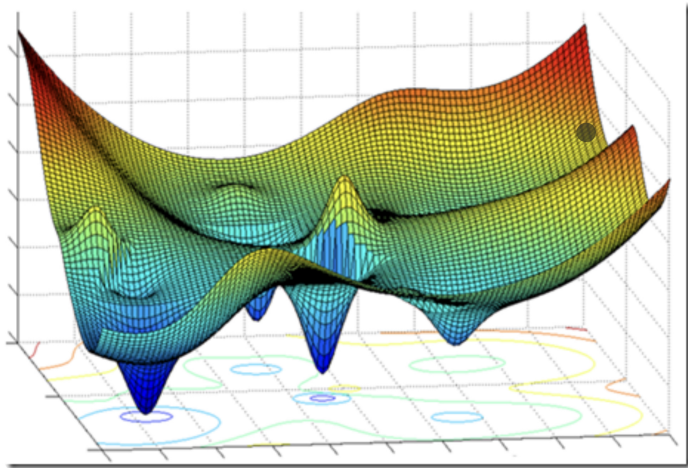
**Cost**



Finally, not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult.

**Cost**







# Batch Gradient Descent

To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter  $\theta_j$ . In other words, you need to calculate how much the cost function will change if you change  $\theta_j$  just a little bit.

$$\frac{\partial}{\partial \theta_j} MSE(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^m \left( \boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)} \quad (2)$$



## Batch Gradient Descent

Instead of computing these partial derivatives individually, you can use Equation to compute them all in one go. The gradient vector, noted  $\nabla_{\theta}MSE(\theta)$ , contains all the partial derivatives of the cost function (one for each model parameter).

$$\nabla_{\theta}MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial\theta_0}MSE(\theta) \\ \frac{\partial}{\partial\theta_1}MSE(\theta) \\ \frac{\partial}{\partial\theta_2}MSE(\theta) \\ \frac{\partial}{\partial\theta_3}MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}MSE(\theta) \end{bmatrix} = \frac{2}{m}X^T(X\theta - y) \quad (3)$$

Notice that this formula involves calculations over the full training set  $X$ , at each Gradient Descent step! it uses the whole batch of training data at every step (actually, Full Gradient Descent would probably be a better name).



# Special Homework

## Homework 25pts for unit 1

Develops the procedure to obtain the matrix/vector form of the batch gradient descent formula for the  $MSE(\theta)$  in details.

**Take a picture and send it by Teams**

$$\nabla_{\theta} MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} MSE(\theta) \\ \frac{\partial}{\partial \theta_1} MSE(\theta) \\ \frac{\partial}{\partial \theta_2} MSE(\theta) \\ \frac{\partial}{\partial \theta_3} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{bmatrix} = \frac{2}{m} X^T (X\theta - y) \quad (4)$$



# Stochastic Gradient Descent

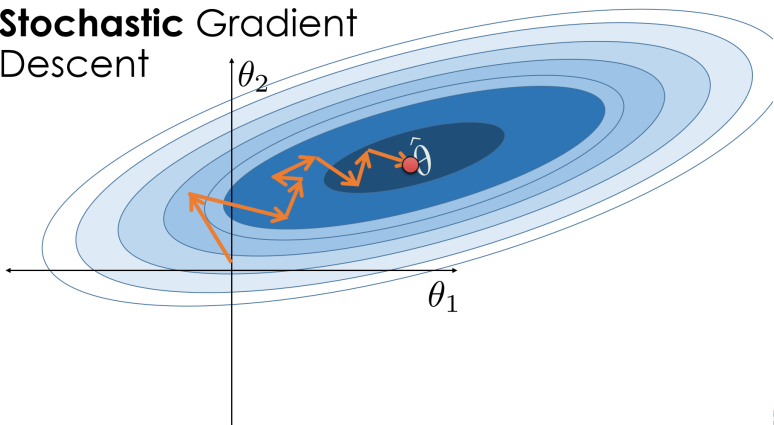
The main problem with Batch Gradient Descent (BGD) is that it uses the whole training set to compute the gradients at every step, making it very slow when the set is large.

At the opposite extreme, Stochastic Gradient Descent (SGD) just picks a random instance in the training set at every step and computes the gradients based only on that single instance.

This algorithm is much less regular than BGD. Thus, once the algorithm stops, the final parameter values are good but not optimal.



# Stochastic Gradient Descent



- ▶ When the cost function is very irregular, this can actually help the algorithm jump out of local minima. Hence, SGD has a better chance of finding the global minimum than BGD does.
- ▶ Therefore, randomness is good to escape from local optima but bad because it means that the algorithm can never settle at the minimum.
- ▶ One solution to this dilemma is to reduce the learning rate gradually. The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minimum.
- ▶ The function that determines the learning rate at each iteration is called the learning schedule.



## Mini-Batch Gradient Descent

Mini-batch GD computes the gradients on small random sets of instances called mini-batches. The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations when using GPUs.

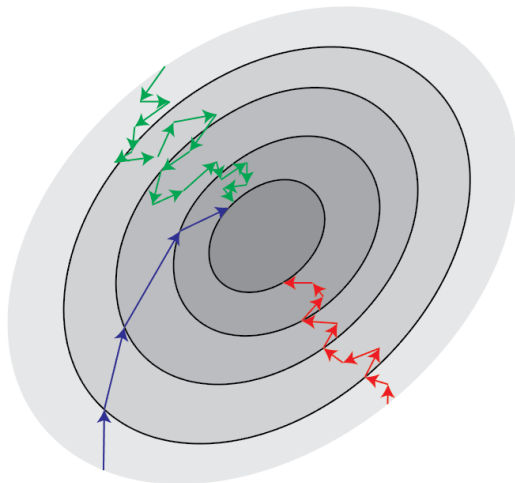
Mini-batch GD will end up walking around a bit closer to the minimum than SGD. But, on the other hand, it may be harder for it to escape from local minima.

They all end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around.

However, don't forget that Batch GD takes a lot of time to take each step, and Stochastic GD and Mini-batch GD would also reach the minimum if you use a good learning schedule.




- Batch
- Mini-batch
- Stochastic





# Referencias

-  Géron, Aurélien. "Hands-on machine learning with scikit-learn and tensorflow: Concepts." Tools, and Techniques to build intelligent systems (2017).

