Gradient Descent

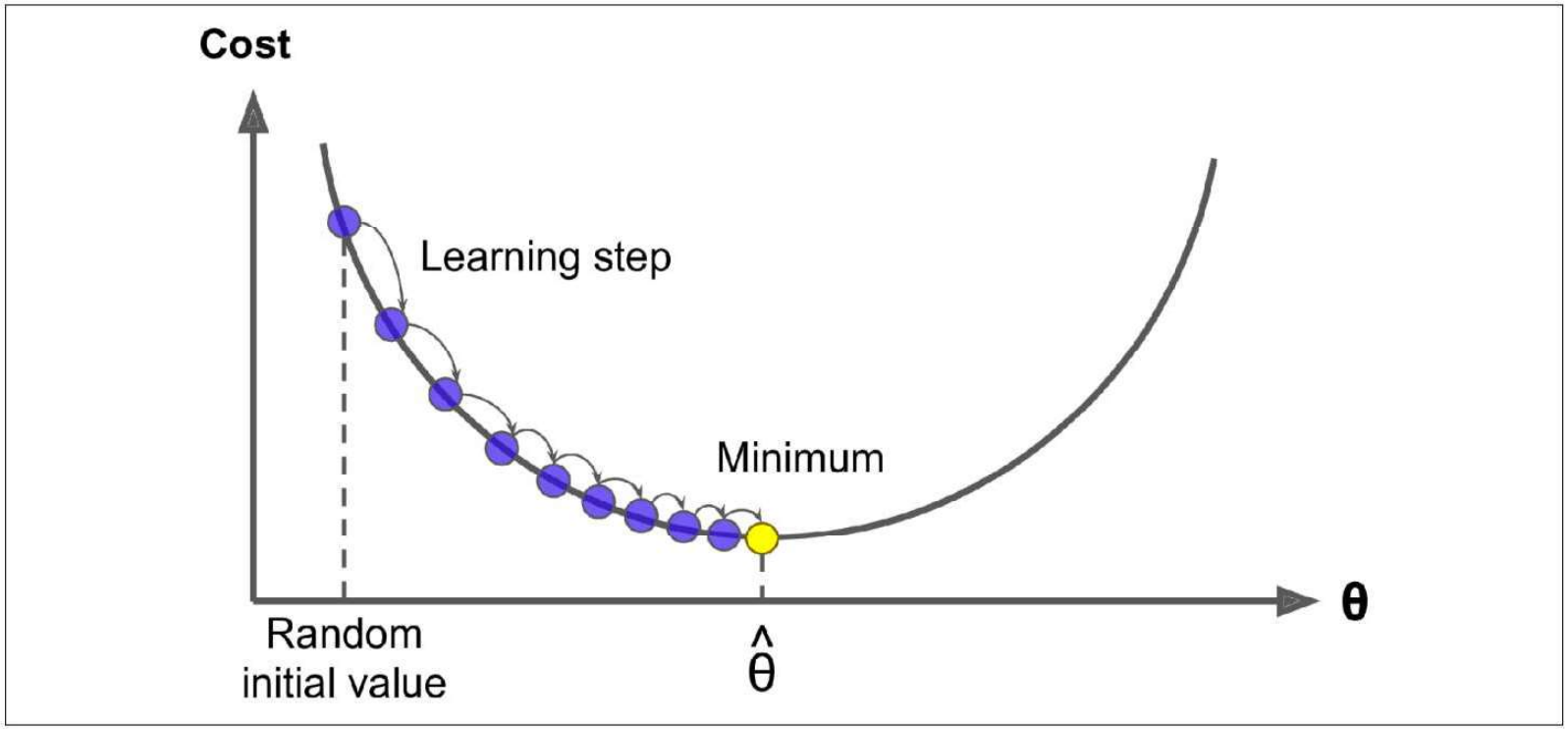
1. Batch Gradient
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

Gradient Descent

- **Gradient Descent** is a 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 to minimize a cost function.
- it measures the local gradient of the error function with regard to the parameter vector θ, and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum!.
- Concretely, you start by filling θ with random values (this is called random initialization). Then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum

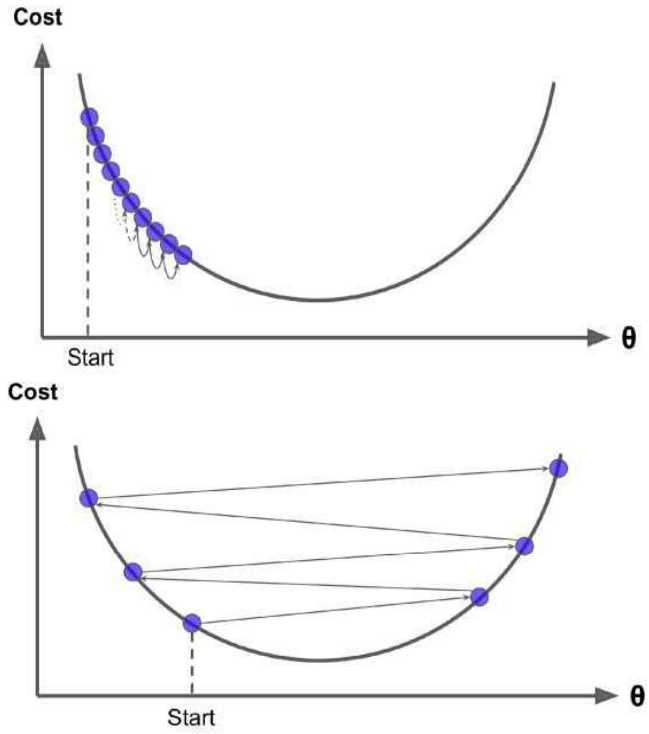
Gradient Descent

- In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function;
- the *learning step size* is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum.

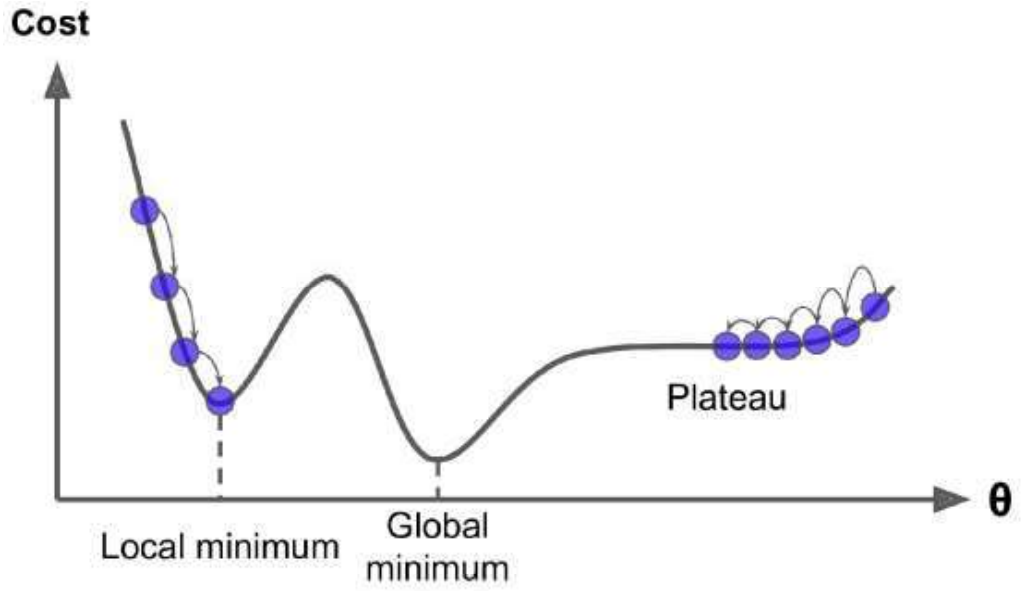


Gradient Descent

- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. 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.
- 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 up than you were before.
- 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 difficult.



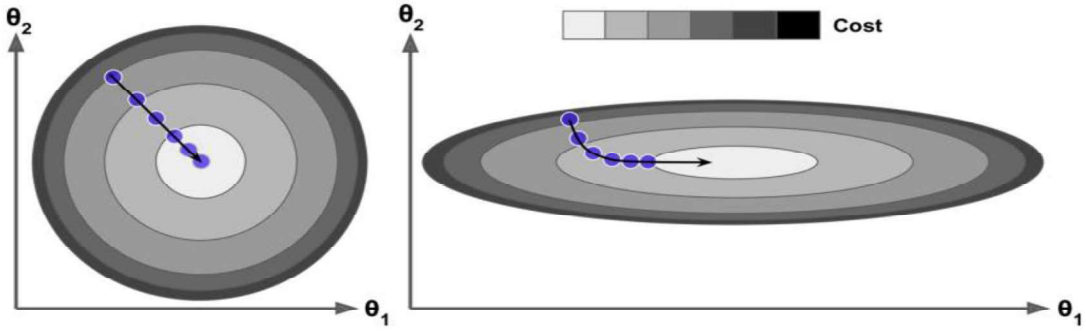
Gradient Descent



Two main challenges with Gradient Descent.

- If the random initialization starts the algorithm on the left, then it will converge to a **local minimum**, which is not as good as the global minimum.
- If it starts on the right, then it will take a very long time to cross the plateau. And if you stop too early, you will never reach the global minimum.

Gradient Descent



- Fortunately, the MSE cost function for a Linear Regression model happens to be a ***convex function***, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve.
- This implies that there are ***no local minima***, just one global minimum.
- It is also a ***continuous function*** with a slope that never changes abruptly. These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close to the global minimum.
- In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the ***features have very different scales***.
- When using Gradient Descent, you should ensure that all features have a similar scale.

Batch Gradient Descent

- To implement Gradient Descent, you need to compute the gradient of the cost function with regard to each model parameter θ_j.
- In other words, you need to calculate how much the cost function will change if you change θ_i just a little bit. This is called a **partial derivative**.

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

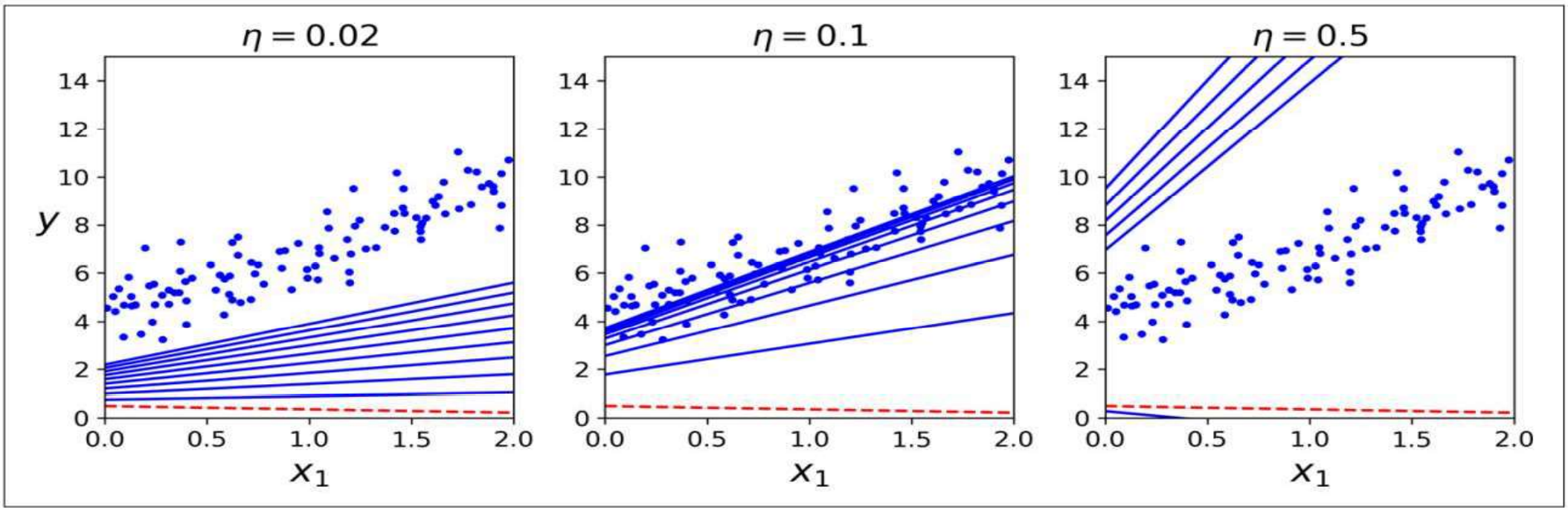
- The **gradient vector** $\nabla_\theta \text{MSE}(\theta)$, contains all the partial derivatives of the cost function (one for each model parameter).

$$\nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Batch Gradient Descent

- Once you have the gradient vector, which points uphill, just go in the opposite direction to go downhill. This means subtracting $\nabla_\theta \text{MSE}(\theta)$ from θ.
- This is where the learning rate $\boldsymbol{\eta}$ comes into play: multiply the gradient vector by $\boldsymbol{\eta}$ to determine the size of the downhill step.

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$



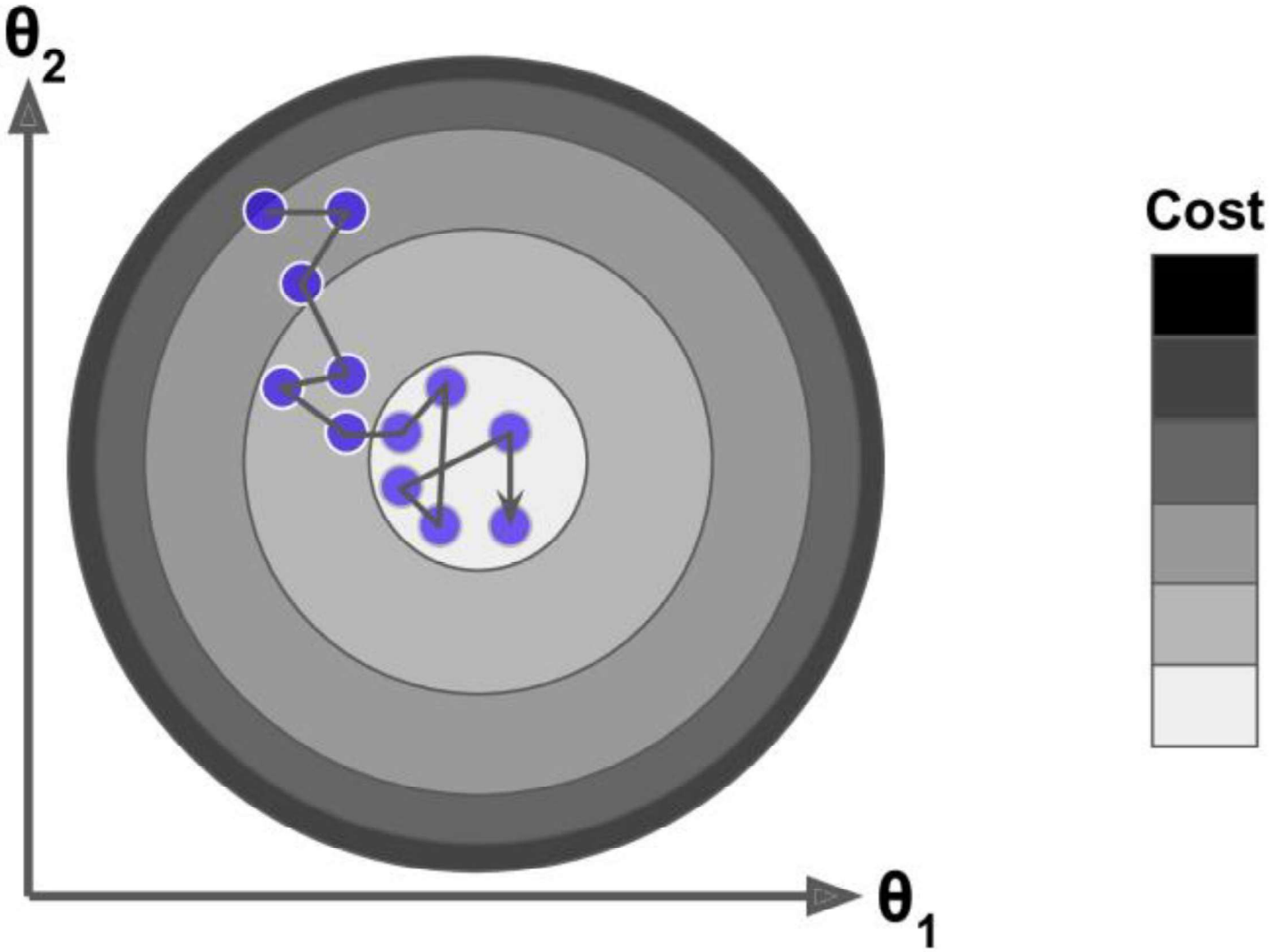
Batch Gradient Descent

- To find a good learning rate, you can use grid search.
- this formula involves calculations over the full training set X, at each Gradient Descent step! This is why the algorithm is called **Batch Gradient Descent**: it uses the whole batch of training data at every step.
- As a result, it is slow on very large training sets.
- A simple solution is to set a very large number of iterations but to interrupt the algorithm when the gradient vector becomes tiny—that is, when its norm becomes smaller than a tiny number ϵ (called the tolerance)—because this happens when Gradient Descent has (almost) reached the minimum.

Stochastic Gradient Descent

- Stochastic Gradient Descent picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- Obviously, working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration. It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration.
- Due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. So once the algorithm stops, the final parameter values are good, but not optimal.

Stochastic Gradient Descent



SGD

- **Advantages:**
  - Speed
  - Memory efficiency
  - Avoidance of Local Minima
- **Disadvantages**
  - Noisy updates
  - Slow Convergence
  - Sensitivity to Learning rate
  - Less Accurate

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, especially when using GPUs.