Gradient Descent

Gradient descent is a widely used algorithm in machine learning, playing a key role in tasks ranging from optimizing linear regression models to training complex neural networks. Its primary objective is to minimize a given function, regardless of the number of parameters involved.

https://upload.wikimedia.org/wikipedia/commons/transcoded/4/4c/Gradient_Descent_in_2D.webm/Gradient_Descent_in_2D.webm.720p.vp9.webm

Gradient Descent in 2D:

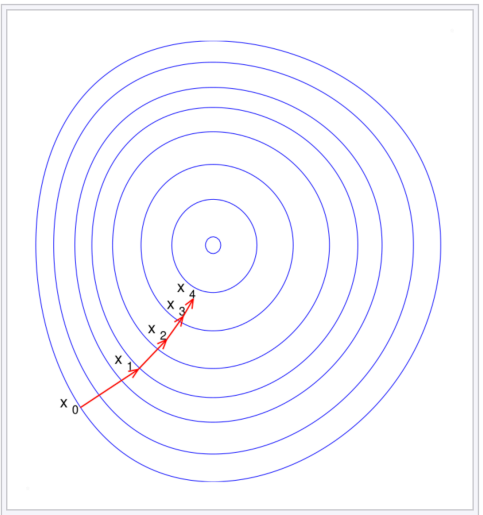
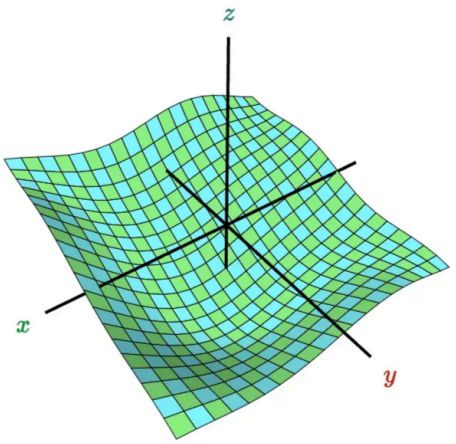


Illustration of gradient descent on a series of level sets

▼ Definning Model

Understanding Gradient Descent

- Gradient descent is a *systematic method* to find **optimal values for parameters (w and b)** that *minimize* the cost function (j).
- It is applicable **not only** to linear regression but also to more complex models, including deep learning

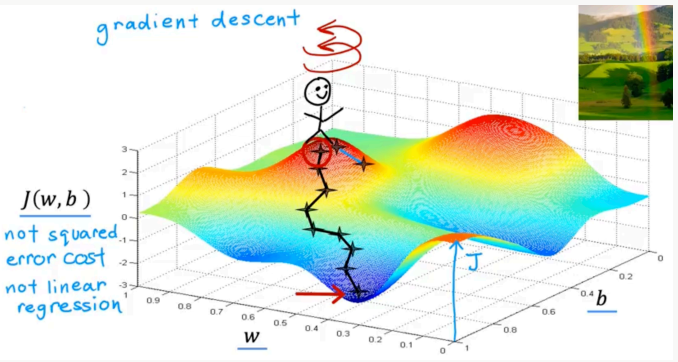


Source: Khan Academy

Process of Gradient Descent

- Start with initial guesses for parameters, often set to 0 $(= 0)$.
- Iteratively adjust parameters in the direction of steepest descent to reduce the cost function until reaching a minimum.
- Let's outline on a high level, how this algorithm works:
  - Start with some parameters w, b
  - Computes gradient using a single Training example.
  - Keep changing the values for w, b to reduce the cost function $J(w, b)$.
  - Continue until we settle at or near a minimum. Note, *some functions may have more than 1 minimum*.

💡 **Example:** an analogy is used to explain gradient descent through a **hilly landscape**

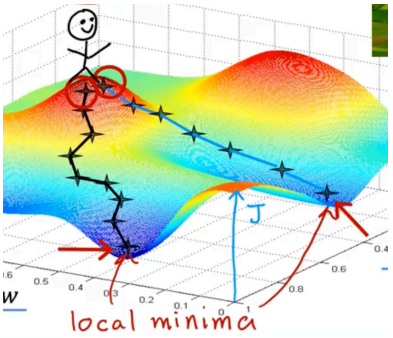


- **Imaginary Hill**: You stand on a hill, with high points as hills and low points as valleys.
- **Finding the Path**: You look around to find the steepest descent and take a small step in that direction.
- **Iterative Process**: Repeat this process until you reach the valley (local minimum).

This example illustrates how gradient descent works by continuously adjusting your position based on the steepest descent until the minimum cost is found.

Local Minima in Gradient Descent

- Different starting points can lead to different local minima, meaning the algorithm may converge to various solutions based on initial values.
- This property highlights the importance of the starting point in the optimization process.



Local Minima

▼ Defining Algorithm

The gradient descent algorithm is defined as a repeated convergence for each input parameter,

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

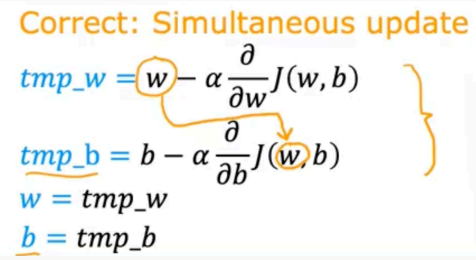
$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

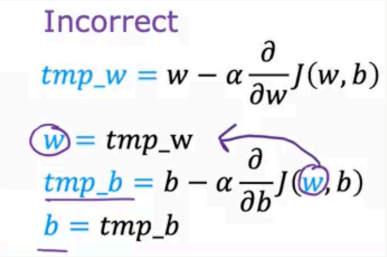
α = ***Learning rate***, which controls the size of steps taken during optimization. Smaller values (between 0 and 1) lead to more gradual updates, while larger values result in bigger steps.

$\frac{\partial}{\partial w} J(w, b)$ = ***Derivative*** of the cost function, which determines the direction to take each step. It is also calculated as the slope of the graph at a particular point.

💡 **NOTE:** ***Simultaneous Updates***

- Both w and b should be updated simultaneously to ensure ***accurate calculations***.
- The correct implementation involves calculating updates for both parameters before applying them, ***avoiding errors*** that can arise from non-simultaneous updates.

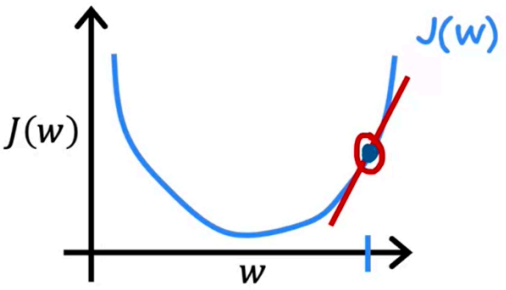




▼ Gradient Descent Intuition

Understanding the Derivative

- The derivative indicates the slope of the cost function at a given point, guiding the direction of the parameter update.
- A positive slope results in decreasing the parameter w, while a negative slope increases w, both aiming to minimize the cost function.



$$w = w - \alpha \cdot (negative\ number)$$

negative slope increases w

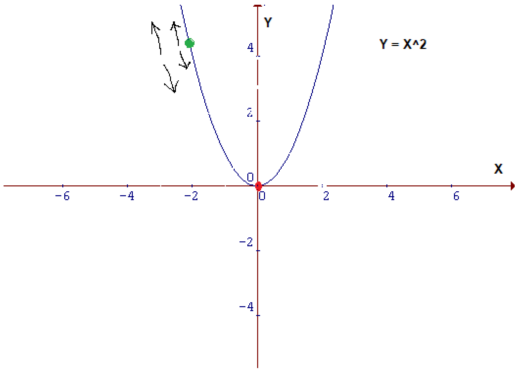
$$w = w - \alpha \cdot (positive\ number)$$

positive slope increases w

Example

source: TowardsDataScience

Consider that you are walking along with the graph below, and you are currently at the '**green**' dot. You aim to reach the minimum, i.e., the 'red' dot, but from your position, you are unable to view it.



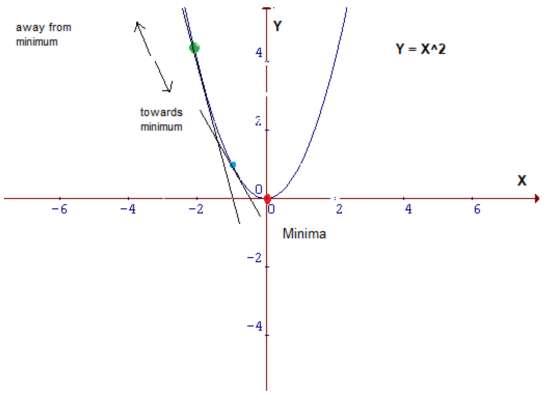
Possible actions would be:

- You might go *upward* or *downward*
- If you decide which way to go, you might take a bigger step or a little step to reach your destination.

> Essentially, there are two things that you should know to reach the minima, i.e. ***which way to go*** and ***how big a step to take.***

⇒ **Gradient Descent Algorithm:** helps us make efficient decisions using **derivatives**, which represent the slope of a graph at a point. This ***slope***, shown by a *tangent line*, guides us **toward the minimum**.

The Minimum Value and Steep Slope



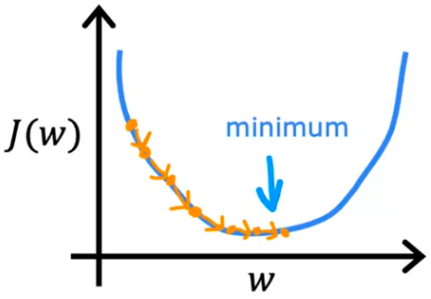
A tangent at the **green** point shows we're moving away from the minimum if going upward. Its ***steep slope*** means ***larger steps***, while the ***gentler slope*** at the blue point means ***smaller steps*** toward the minimum.

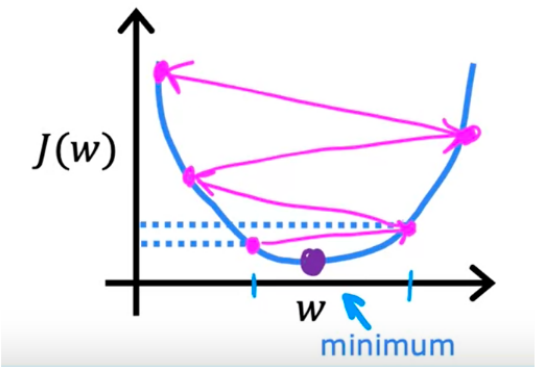
▼ Choosing Learning Rate

The learning rate (α) significantly affects the efficiency of gradient descent.

Too small α:

- Leads to very slow convergence.
- Requires many steps, increasing computation time.





Too large α:

- Can overshoot the minimum.
- May cause the algorithm to diverge away from the minimum (fail to converge).

The challenge increases with complex graphs:

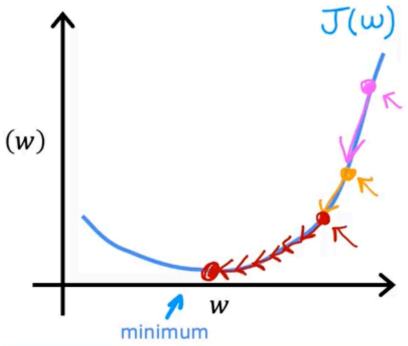
- Multiple *local minima* and *maxima* make optimization harder.

Choosing the right α is crucial for stable and effective convergence.

Can reach local minimum with fixed Learning Rate α

Near a local minimum,

- Derivative automatically gets smaller.
- Update steps become smaller.



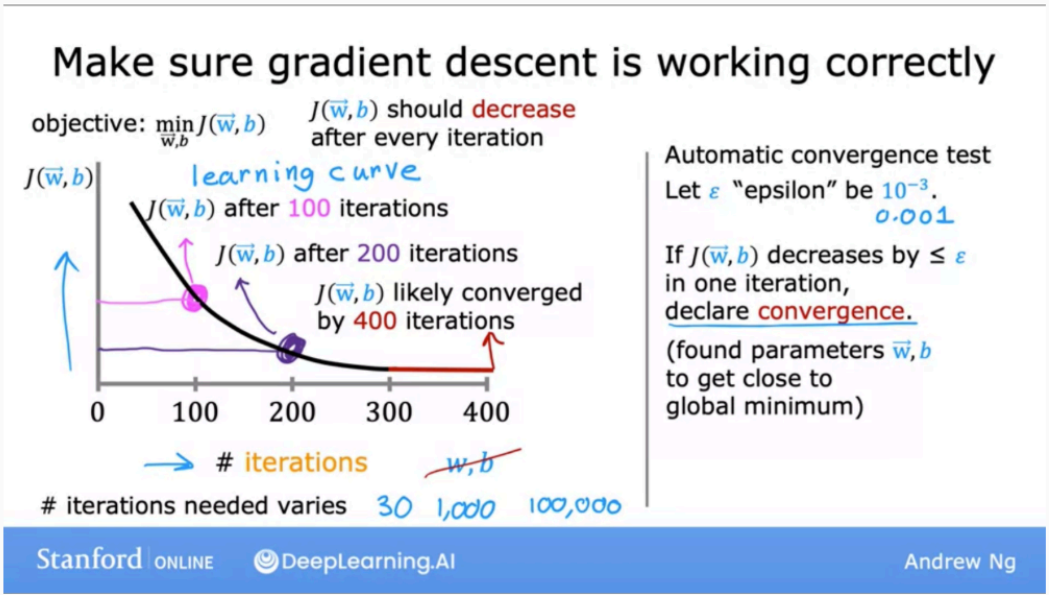
Optimizations

source: GitHub

How can we check gradient descent is working correctly?

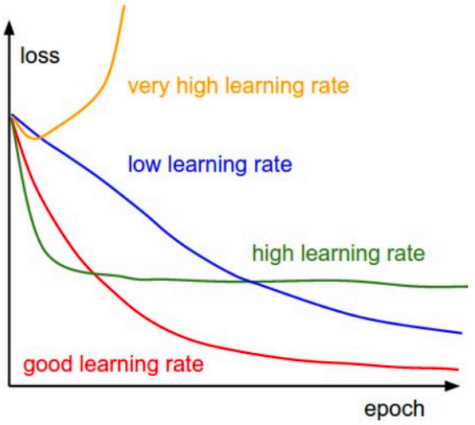
source: Vinija_Notes

- We can have 2 ways to achieve this. We can plot the cost function J, which is calculated on the training set, and plot the value of J at each iteration (aka each simultaneous update of parameters w, b) of gradient descent.
- We can also use an *Automatic convergence test*. We choose an ϵ to be a very small number. If the cost J decreases by less than ϵ on one iteration, then you're likely on this flattened part of the curve, and you can declare convergence.



Learning curve

Gradient descent aims to minimize the cost function $J(w, b)$. Plotting J over iterations gives the **learning curve**, which ideally should drop quickly and converge to 0. If J increases at any point, it may indicate a poor learning rate α or a bug. The required number of iterations varies by application, making it hard to predict convergence time.



Automatic convergence test

An automatic convergence test can stop training when the drop in cost J(w, b) between iterations is by $<= \epsilon$. Selecting the right ϵ can be challenging, so it's best used together with a learning curve.

Debugging

Although a cost function that fails to decrease or fluctuates is often due to a high learning rate, it may also indicate a coding bug. Using a very small learning rate can help reveal bugs, as the algorithm should still behave unexpectedly if an issue exists.

▼ Gradient Descent for Linear Regression

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

(The squared error cost function)

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)^2$$

Pre-derived gradient descent algorithm

repeat until convergence:

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

We have:

$$\frac{\partial}{\partial w} J(w,b) \quad \Rightarrow \quad \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i) x^i$$

$$\frac{\partial}{\partial b} J(w,b) \quad \Rightarrow \quad \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)$$

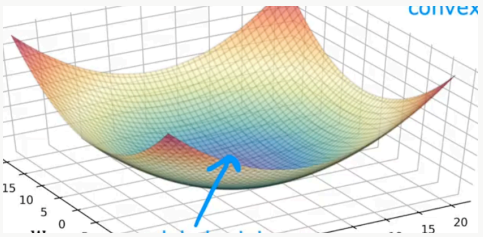
Final gradient descent algorithm

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i) x^i$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)$$

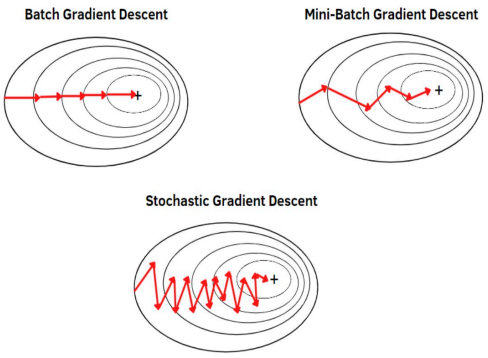
💡 **Convex function** is cost function of linear regression

- The squared error cost function is convex, ensuring that gradient descent will always converge to a **global minimum**, avoiding local minima issues.



▼ Batch Gradient Descent

- Batch gradient descent uses the entire training dataset to compute updates at each step, ensuring a comprehensive approach to optimization.
- This method contrasts with other versions of gradient descent that utilize smaller subsets of data.



https://statusneo.com/efficientdl-mini-batch-gradient-descent-explained/
23/05/2025

- When we try to find the minimum in the contour plot: The graph is like following: