# Gradient Descent

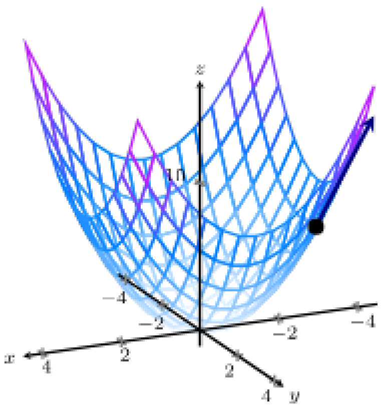
- What is a Gradient in math
- What is Gradient Descent
- Gradient Descent Algorithm
- Why Gradient Descent

## What is Gradient in math

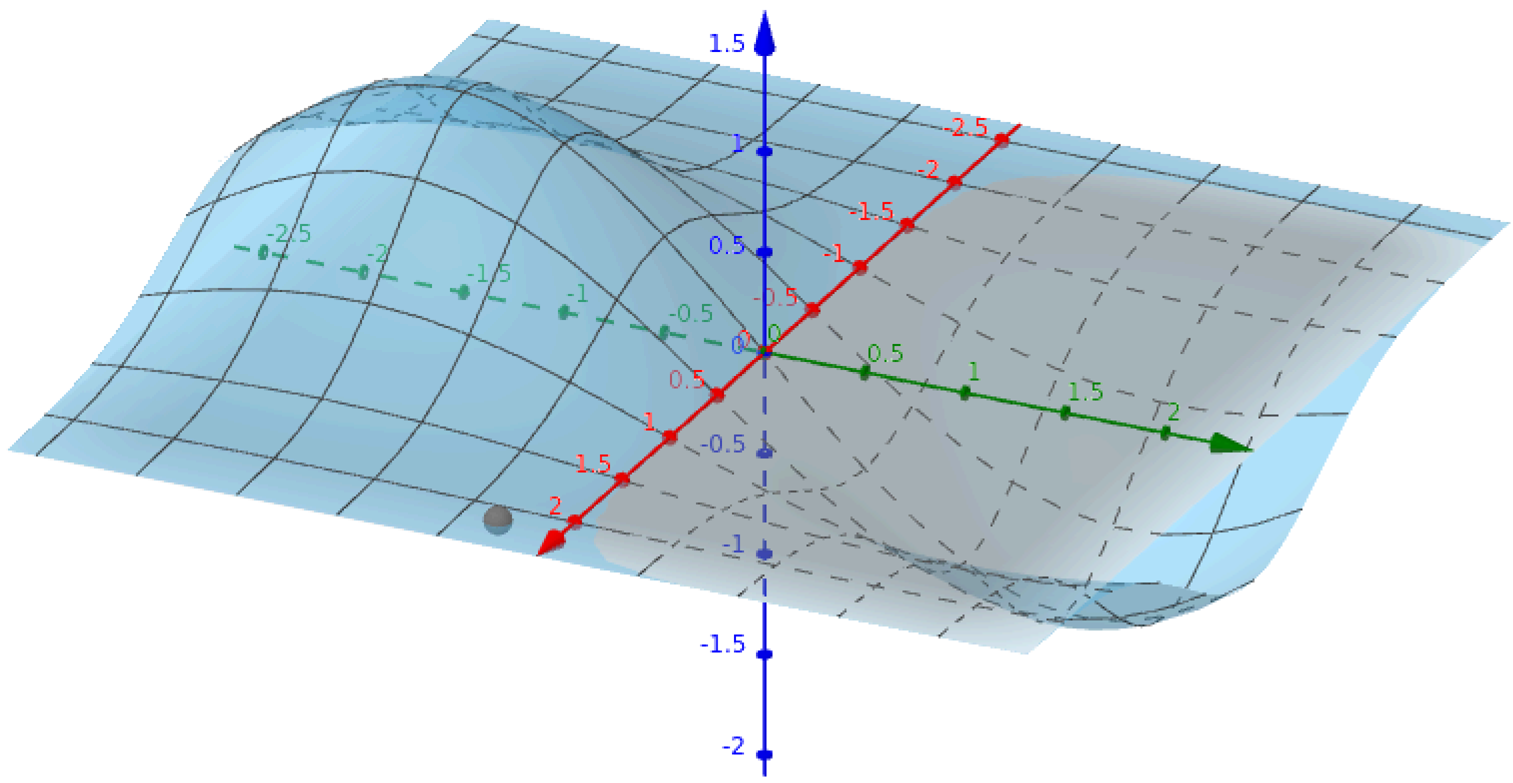
A gradient is a **vector** of partial derivatives for a **multivariate** function $f(x_1, x_2, ..., x_n)$ with respect for each variable $x_i$

$$\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots \frac{\partial f}{\partial x_n})$$

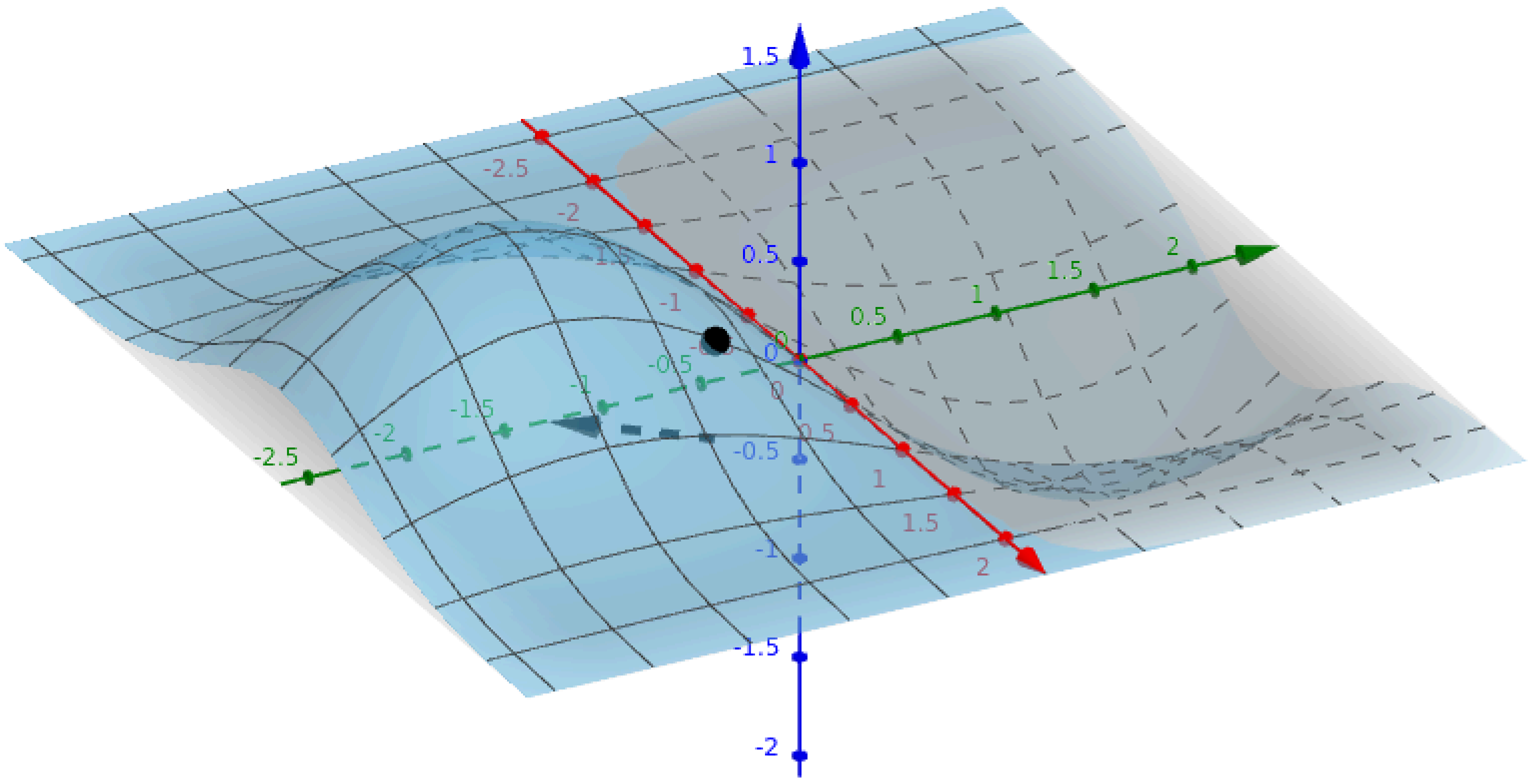
- The Gradient vector points in the direction of the **maximum** change (increase), formally **steepest ascent**
- Its magnitude is equal to the **maximum rate** of change



- A zero gradient $\nabla f = 0$ indicates a **critical point**



- The Gradient measures the "slop" in all directions our point here is on a *flat surface* the gradient vector is zero




- Unlike here where there is a slight slop the gradient vector $\nabla f$ points to the direction of the **steepest ascent**

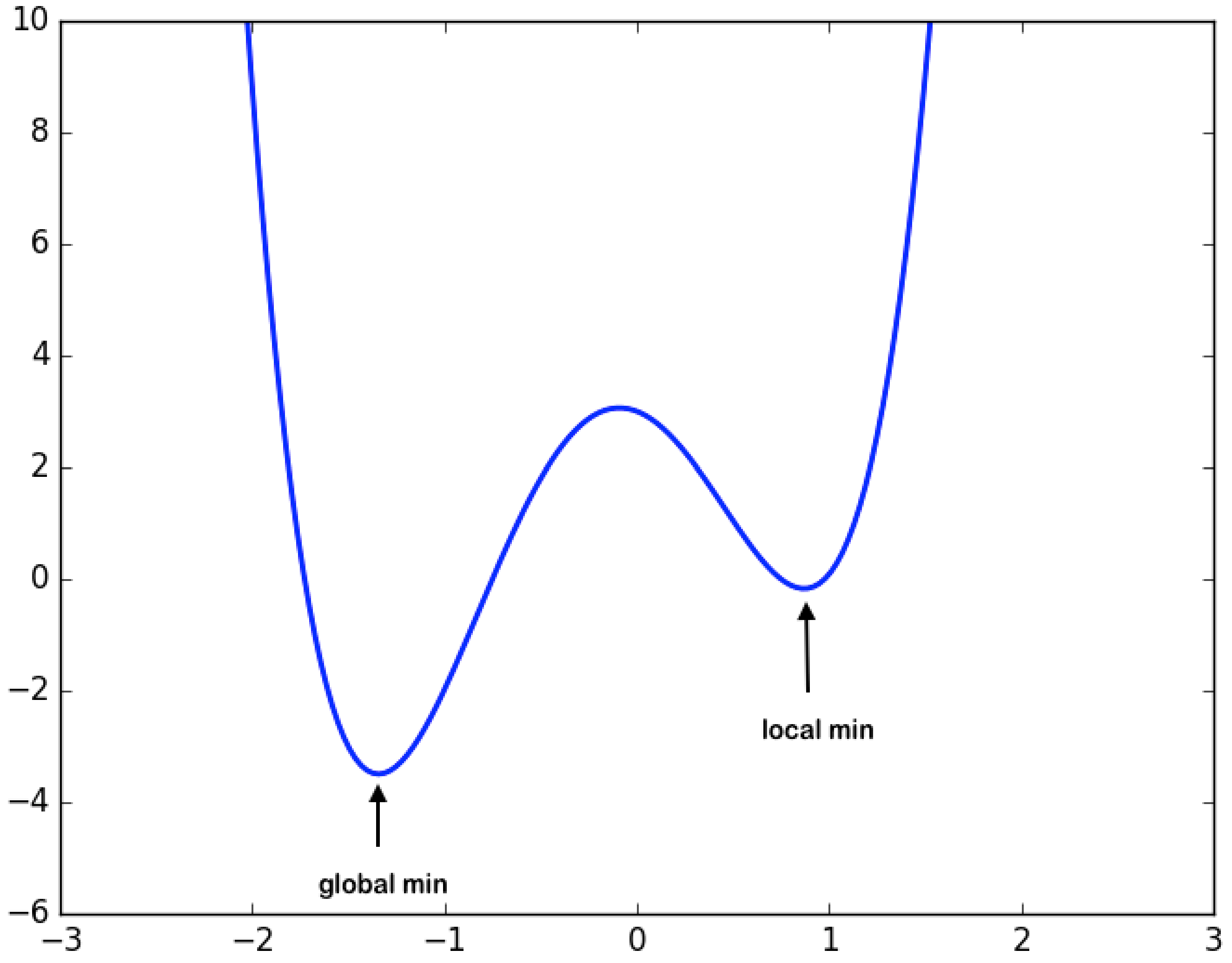
## Gradient Rules

1. Product Rule
   $\nabla(f.g) = f\nabla g + g\nabla f$
2. Quotient Rule
   $\nabla\left(\frac{f}{g}\right) = \frac{g\nabla f - f\nabla g}{g^2}$
3. Gradient of a Norm
   $\nabla||x|| = \frac{x}{||x||}$
4. Directional Derivative Connection
   $D_{\hat{u}}G = \nabla G.\hat{u}$

# What is Gradient Descent ?

Simply its an **optimization** algorithm used to **train machine learning** models, by optimizing the **parameters** of the model **iteratively** to find the **global minimum** of a loss function curve

More formally the Gradient Descent is an algorithm that essentially computes the gradient $\nabla$ of the cost loss function ($\mathrm{MSE}$ in Linear Regression) $*h*$




- That is the lowest point in a function curve

### Intuition :

Imagine a person(**gradient descent algorithm** ) is stuck in a foggy mountain(**loss function curve**) and he is trying to get down ( finding the **global minimum** ). Therefore the person need to use local information and **calculations** and what's visible to descent down , using the gradient descent which says look at your current position and goes into the direction of the steepest descent

- **Fog** $\rightarrow$ Limited, local information
- **Slop** $\rightarrow$ Gradient
- **Step size** $\rightarrow$ Learning rate

## Gradient Descent Algorithm

Generally the Gradient Descent follows These steps :

1. Initialize $\theta$ (randomly)
2. While not converged :
    1. Computer gradient : $\nabla J(\theta)$
    2. Update parameters : $\theta^{+} = \theta^{-} - \alpha \nabla_{\theta} J(\theta)$
    3. Check convergence **optional**
3. Return optimized $\theta$

**Note** :

- $J(\theta)$ is a the **Loss Function**
- $\alpha$ is the **Learning Rate** which is the step size
- **Batch size** $\rightarrow$ its the trade off between speed and stability

### Iterations vs Epochs

An **Epoch** $\rightarrow$ is going through all the batches

- In **SGD** the Epoch $= 1$ , cause we have one batch of the training example

An **iteration** is the number of **iterations** to complete one **Epoch**

- In **SGD** for example we need $5$ iterations to complete one **Epoch** which is one batch of training example

# Types Of Gradient Descent

1. Batch Gradient Descent
2. Stochastic Gradient Descent **SGD**
3. Mini-Batch Gradient Descent

## Batch Gradient Descent

This is the **Vanilla Gradient Descent**, and works by :

- Using all the data set to compute the gradient of the cost function $\nabla J(\theta)$ in each iteration
- In simple terms it calculate all the gradients for every training example and take the **mean gradient** to make just one step for one iteration
  **Pros** :
  Its a *Slow but perfect approach* and **deterministic** which means we will get the same values each time

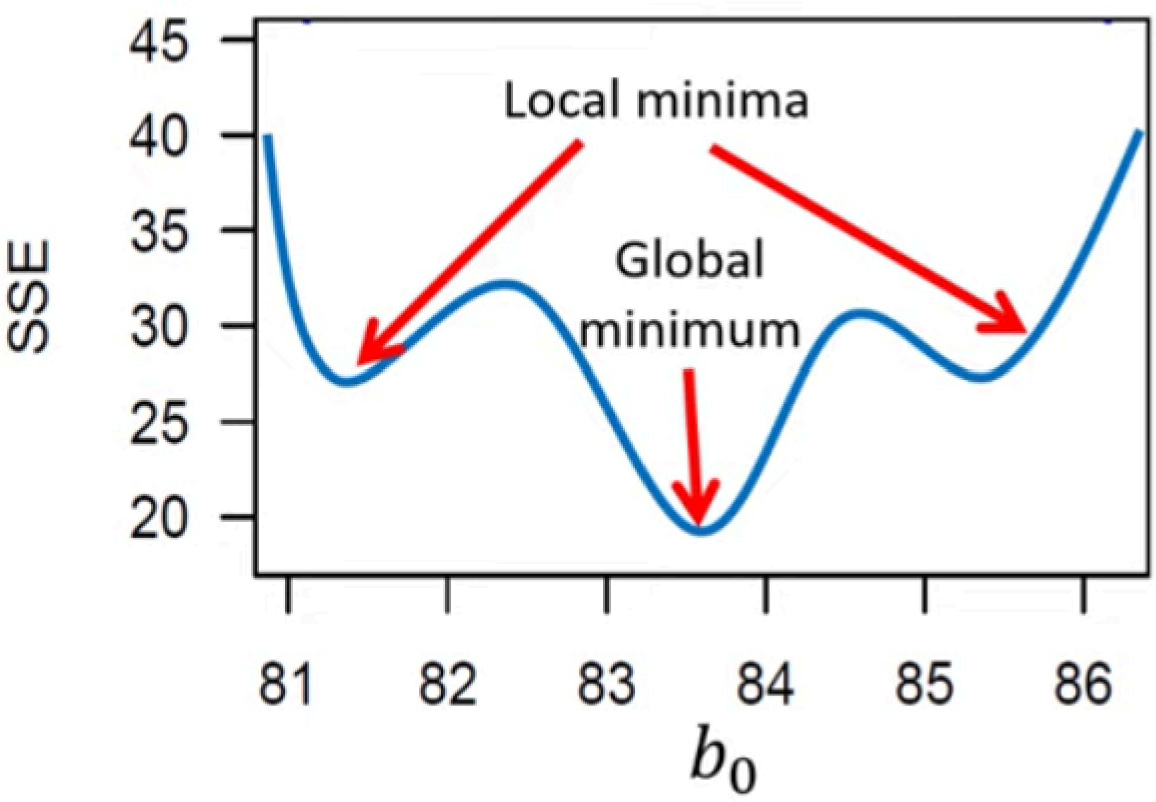
**Down sides** :

- Its **Computationally** expansive for large-medium data sets
- Can get stuck in **local minima** or **saddle points**

$$\theta^{+} = \theta^{-} - \alpha \nabla_{\theta} J(\theta)$$

## Stochastic Gradient Descent

Consider this the *Lazy Gradient Descent* which consider a random training examples from the data sets per iteration to compute the gradient




- Take this loss function for example using the **Full Batch Gradient Descent** it might take it a very long time to escape the local minima thinking its the minimum value for the loss function(cost function)

*Pros* :

- Here where the **Stochastic Gradient** can be useful cause its fluctuate due to randomness it escapes the **local minima** and results in faster updates which can be very effective in large data sets

*Down sides :*

- May never reach exactly the minimum of the cost function

- Also suffers from high variance

$$\theta^{+} = \theta^{-} - \eta \nabla_{\theta} J(\theta; x^{i}, y^{i})$$

Where $(x^i, y^i)$ is a single training example

## Mini-Batch Gradient Descent

Its a mixed approach between *Full Gradient Descent* and *Stochastic Gradient Descent* where instead of taking one example $(x^i, y^i)$ from the training data, we take a small batch typically around $32, 64, 128, 256 \dots$

*Pros :*

- Its a perfect mix between stability and efficiency, **Works faster than Full batch GD and less variance than SGD**
- Works well with GPU parallelization

*Down sides :*

- Just requires fine tuning the batch size

$$\theta^{+} = \theta^{-} - \eta \nabla_{\theta} J(\theta; B)$$

Where $B$ is the mini-batch of the training examples