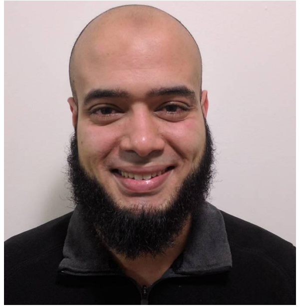
# Machine Learning
# Gradient Descent

**Mostafa S. Ibrahim**
*Teaching, Training and Coaching for more than a decade!*

*Artificial Intelligence & Computer Vision Researcher*
*PhD* from Simon Fraser University - Canada
*Bachelor / MSc* from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

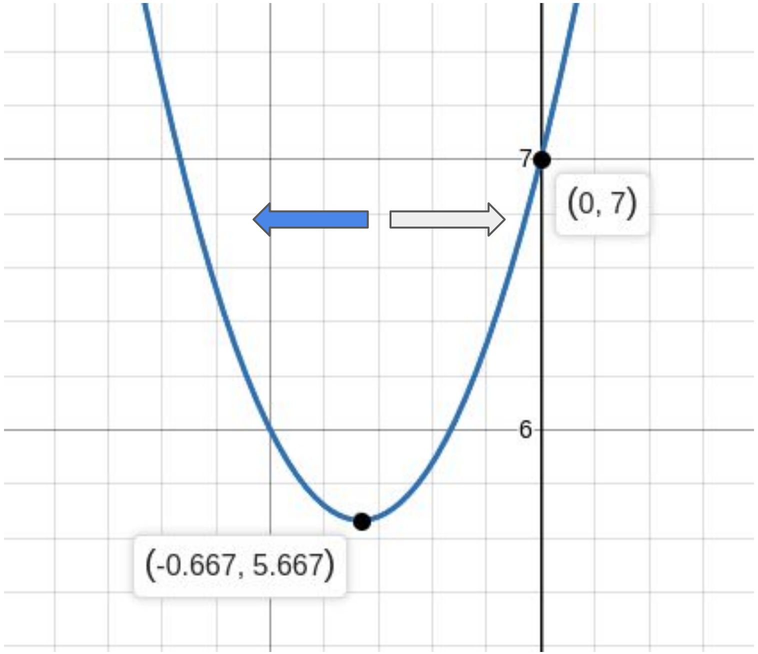


# Optimization

- An optimization problem is one that involves minimizing or maximizing a function
- Last time, we found the minimum of $f(x) = 3x^2 + 4x + 7$ using an **analytical solution**
  - Find derivative $f'(x)$
  - Set to zero $f'(x) = 0$
  - Find x: the minimum for a function with a global minimum
- The previous approach won't work for the more complex functions coming up!
- Let's find the minimum in an iterative way for the same $f(x)$

# Gradient Descent

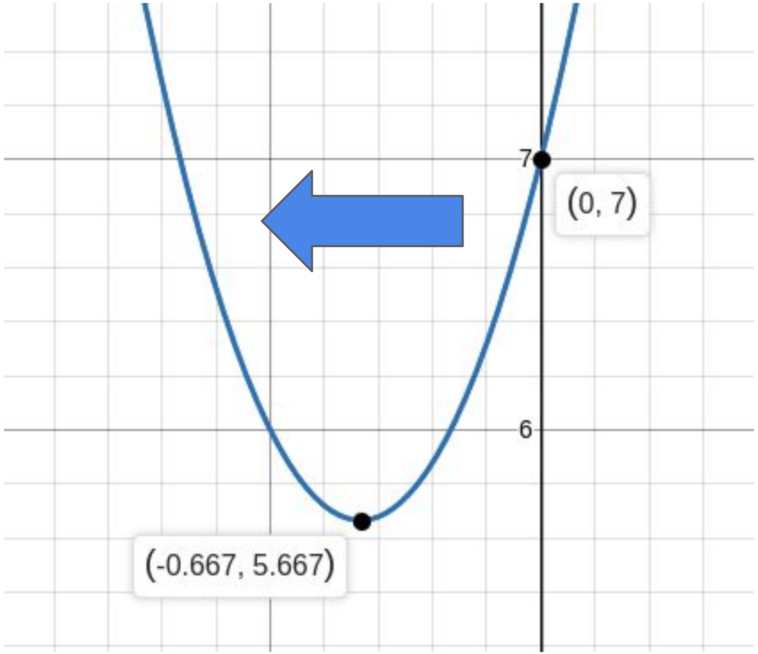
- Our goal is to find the x that has the **minimum** y (analytically x = -⅔)
- We'd like to find the answer, but in an **iterative** manner!
- Start from an **initial** location, e.g. x = 0
- **Strategy**: keep moving x with Δx = 0.01 in the **direction** that makes it **closer** to the optimal point!
- Should we go left or right? -0.01 vs 0.01




- $f(x) = 3x^2 + 4x + 7$
- $f'(x) = 6x + 4$

# Positive slope case

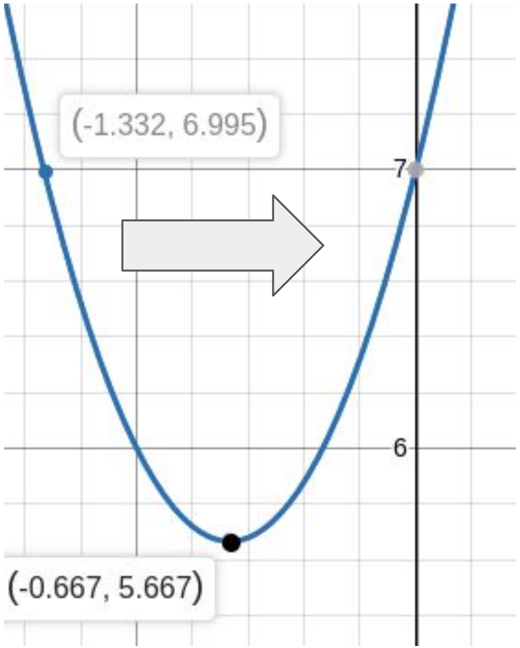
- Intuitively, we would like to move towards the left side (**negative** Δx)
- What is the slope sign at x = 0?
  - f'x(0) = 4: **positive slope**
- This means we need to move in the **opposite direction** of the slope!
- Then, **keep** moving towards the left until reaching a point with **zero slope**
  - The minimum!



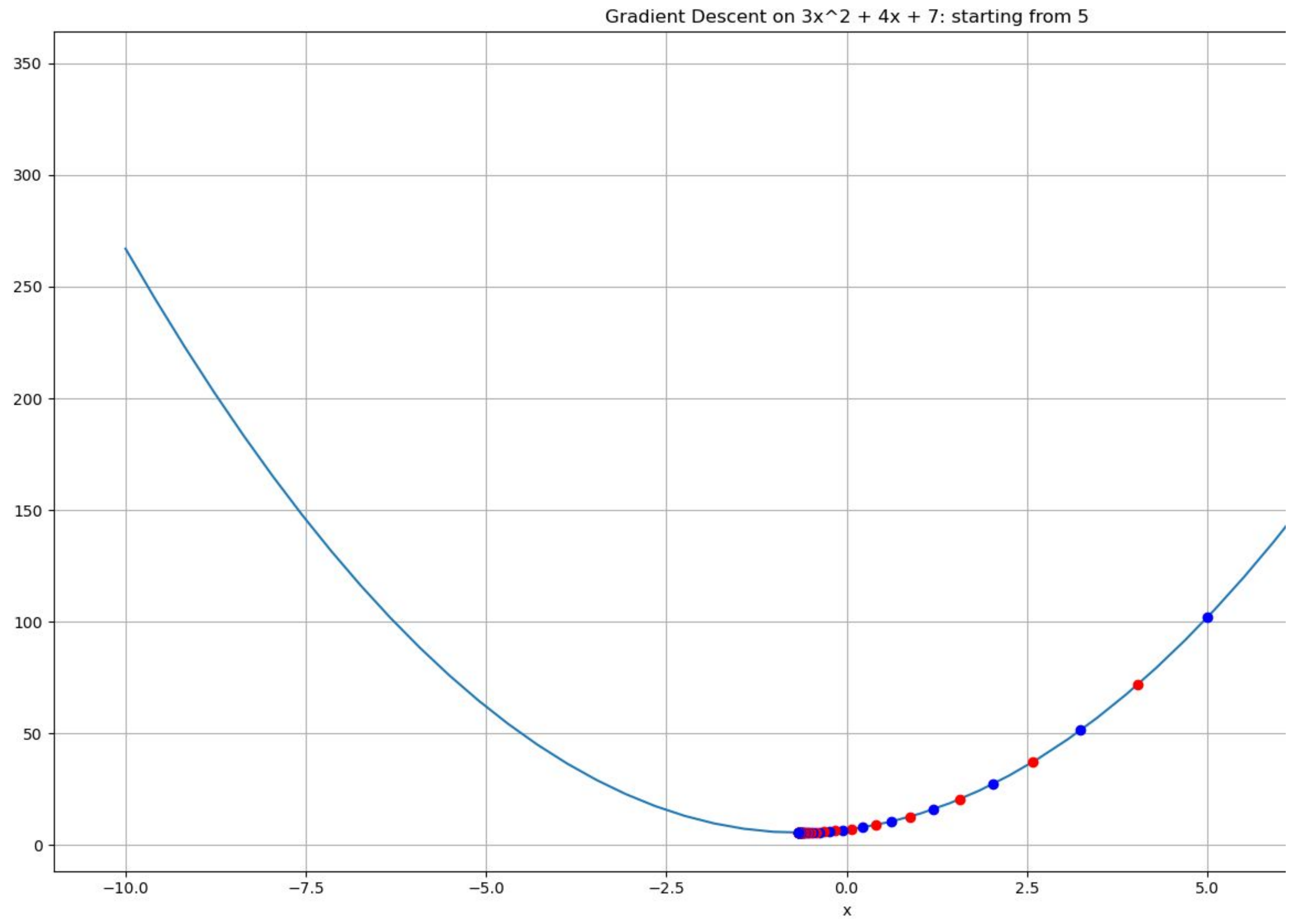

- **$f(x) = 3x^2 + 4x + 7$**
- **$f'(x) = 6x + 4$**

# Negative slope case

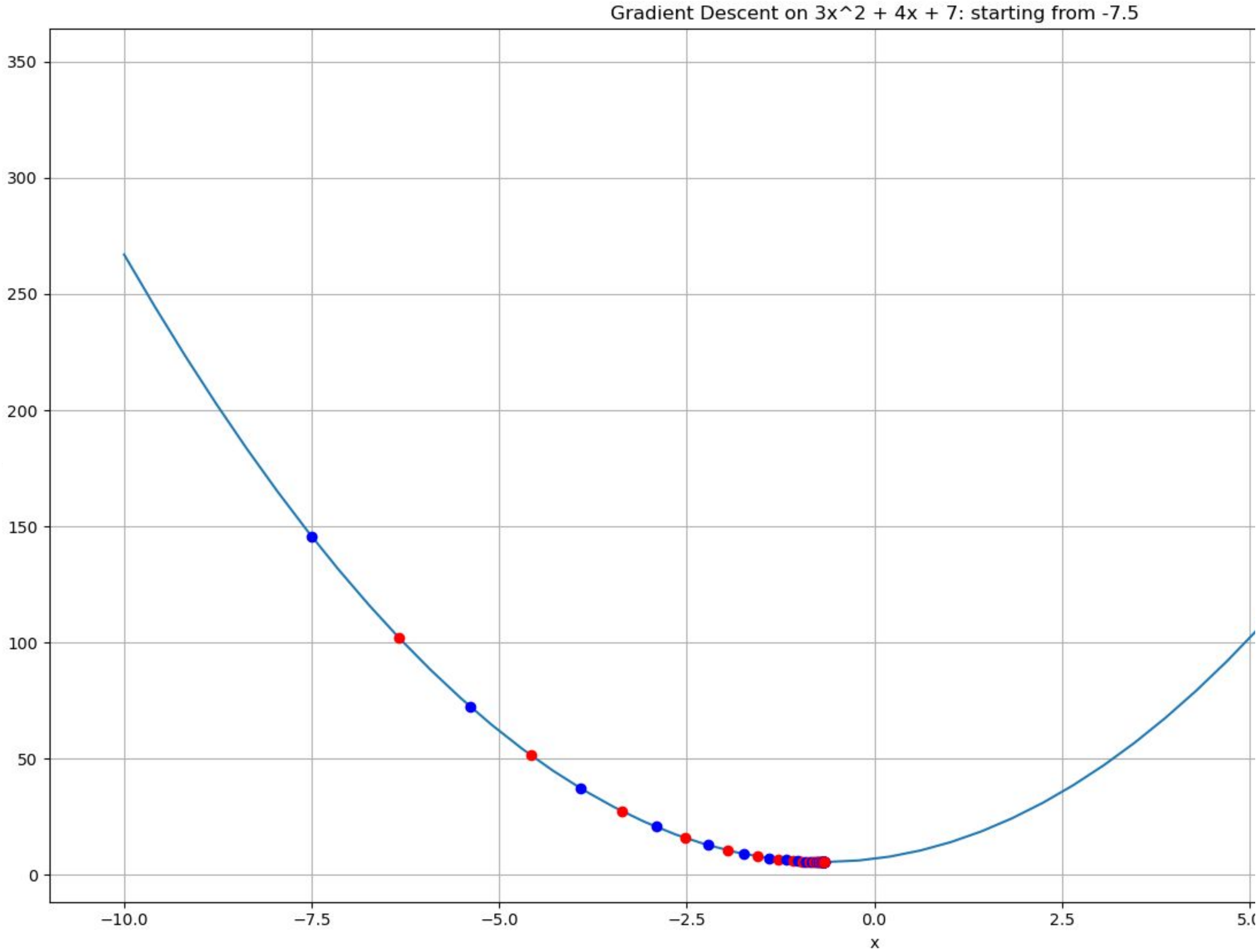
- What if we started from x = -1.3?
- Intuitively, we would like to move towards the right side (**positive** Δx)
- What is the slope sign at x = -1.3?
  - f'x(0) = -3.8: **negative slope**
- This means we need to move in the **opposite direction** of the slope!
- Then, **keep** moving towards the right until reaching a point with **zero slope**
  - The minimum!




- **$f(x) = 3x^2 + 4x + 7$**
- **$f'(x) = 6x + 4$**



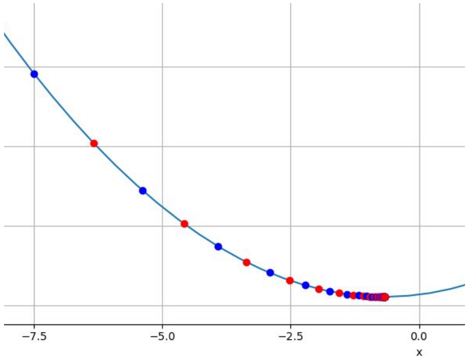
Gradient Descent on 3x^2 + 4x + 7: starting from 5
350
300
250
200
150
100
50
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
x



Gradient Descent on 3x^2 + 4x + 7: starting from -7.5
350
300
250
200
150
100
50
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
x

# Gradient Descent

- An **iterative** algorithm to find the **local** minimum
  - Start from an **initial** location
  - Keep moving in the **opposite direction** of the gradient
- So far we used **fixed** $\Delta x = 0.01$
  - But this constant can cause issues based on the steepness of the curve
  - It can be very slow. It can make big moves close to the minimum!
- How can we make it **dynamic**? Use the gradient value itself!
- However, this value itself has the potential to be very big
  - Let's multiply by a small value
  - Let's call it the **learning rate (lr)**. It is a hyperparameter
    - A **hyperparameter** is a parameter whose value is used to control the learning process

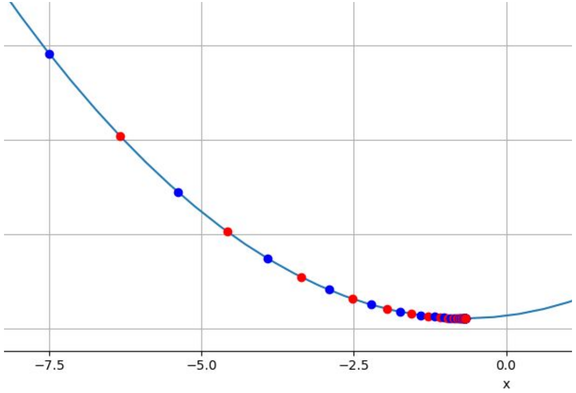


# Parameter and Hyperparameter

- Our model for a simple line has 2 parameters: m and c
  - We would like to **learn** these 2 lines
- We needed an extra parameter, the learning rate (and precision)
  - We call these hyperparameters
  - We typically don’t learn them
  - But we experimentally try different values to find suitable ones

# Stopping Criteria

- One simple criteria is the **number of iterations** (e.g. 100 iterations)
  - But what if we need more?
  - However, it's still good to force an end!
- Another way is the **precision**
  - At each iteration we have the old x and the new x
  - Once the 2 values are almost the same, stop the program
- *The best is to use both*

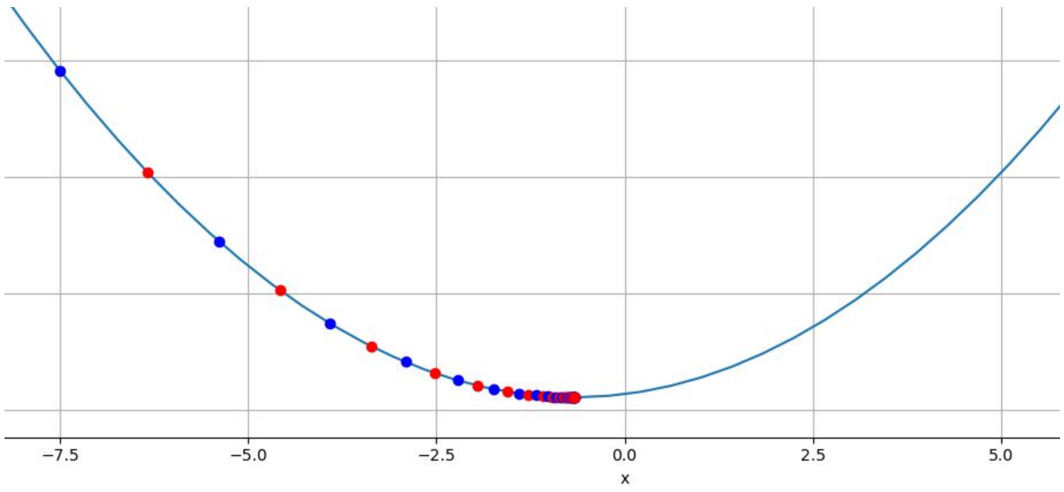


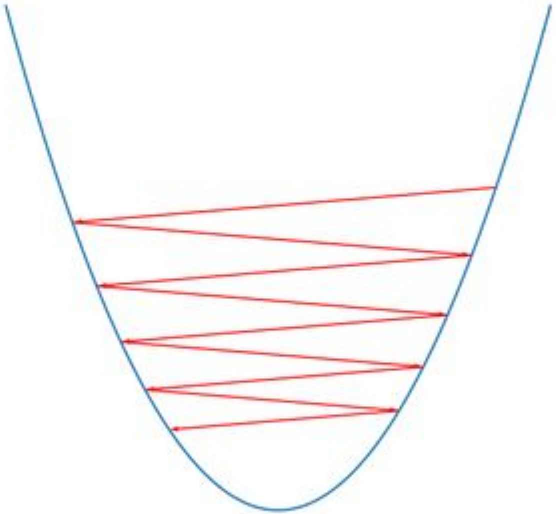

```python
step_size = 0.01          # learning rate
cur_x = -7.5

for iter in range(100):
    gradient = f_derivative(cur_x)
    cur_x -= gradient * step_size   # move in opposite direction
```

# Effect of **large** step size (learning rate)

- We may suffer from an **oscillating behaviour** around the minimum value
  - This means that it keeps missing the minimum, instead jumping to positive and negative directions on either side of the minimum



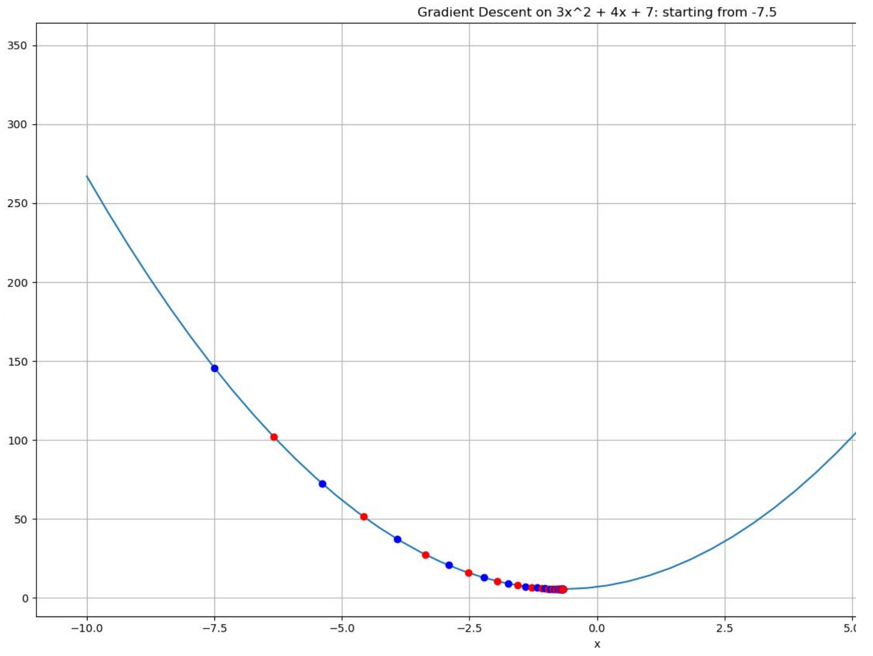
# Code Tour

# Question!

- Do we need to decrease the step size (LR) **over time** for the algorithm to **converge**?

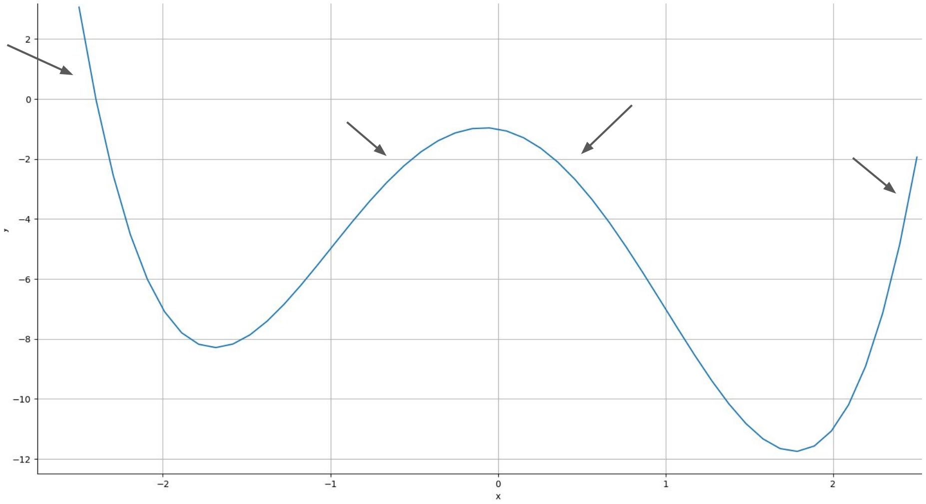
No, the gradient descent itself will take smaller steps when getting closer to the minima

```
for iter in range(100):
    gradient = f_derivative(cur_x)
    cur_x -= gradient * step_size
```



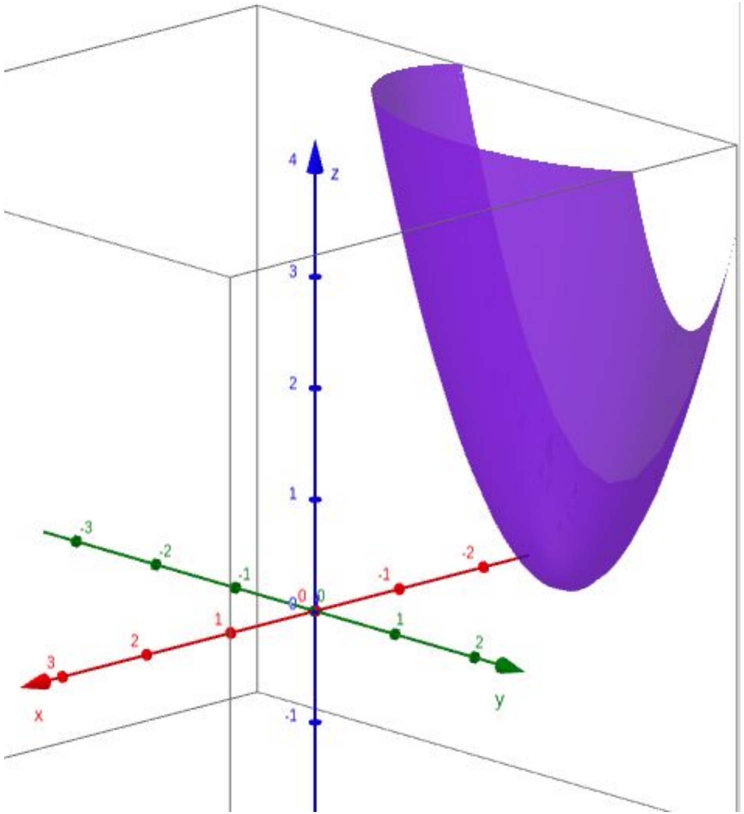
# Global vs Local

- When the function has a single u-shape, it will have a single **global minimum**
  - We call them **convex** functions
- However, if it has multiple u-shapes, it will have several minimums
  - We call them local **minima**, except the best the most minimum among all of them
- Consider the function
  $x^4 - 6x^2 - x - 1$
- There are 2 local minimum ys
  - X = -1.68, Y = -8.3
  - X = 1.17, Y = -11.7 (global)
- Where do we end if we start from: -2.4, -0.15, 0.1, 2.39
- Let's run it



# Graph of function of multivariate variables

- What about functions that have 2 variables? Or **multivariate**?!
- Consider for example:
  $f(x, y) = 3(x + 2)^2 + (y - 1)^2$
  - Now, both x and y should be updated
  - **Analogy**: To head to the south-west, people don't usually move directly from east-west, and then north-south. Typically, the shortest path (i.e. moving along both coordinates simultaneously) is preferred!
  - This function has a **global** minimum
- we can't **move both** x and y at the same time when looking at a **tangent**

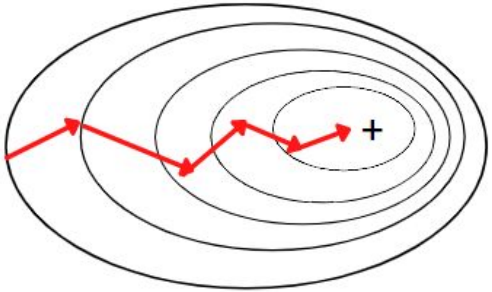


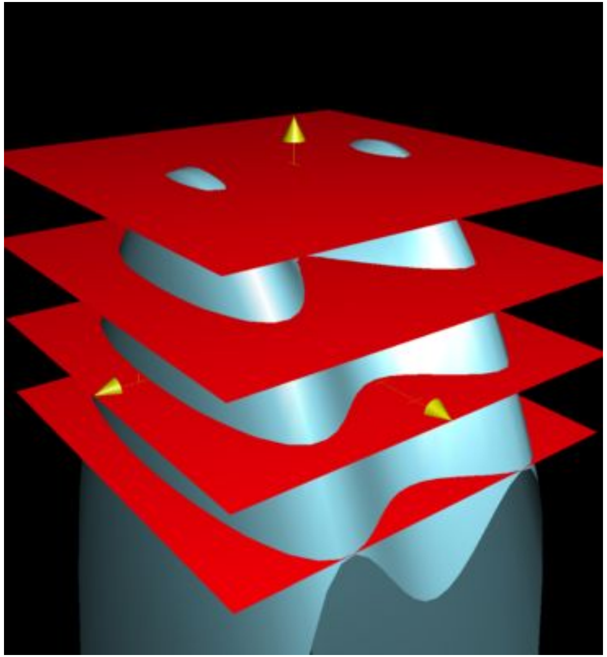
# Graph of function of multivariate variables

- A *typical* solution for multivariate problems that you can’t solve simultaneously is to try solving per variable and combining the results
  - Intuition: can we reduce it to 1D movements per variable?
- Therefore, we apply the **partial derivative** for each variable
  - Which mainly fixes on one variable (a 1D slice from this variable's perspective)
- The overall algorithm will just be extended from 1D to 2D
  - Compute a partial derivative for every variable
  - Make an update to all of them together

# Slicing a graph on output dimension (z)

- If you slice the graph with a plane that is parallel to the xy-plane, then all **z-values** will be equal
  - For example, if z represents some cost function, then all such (x, y) solutions will have the same cost
  - This is how we create contour plots

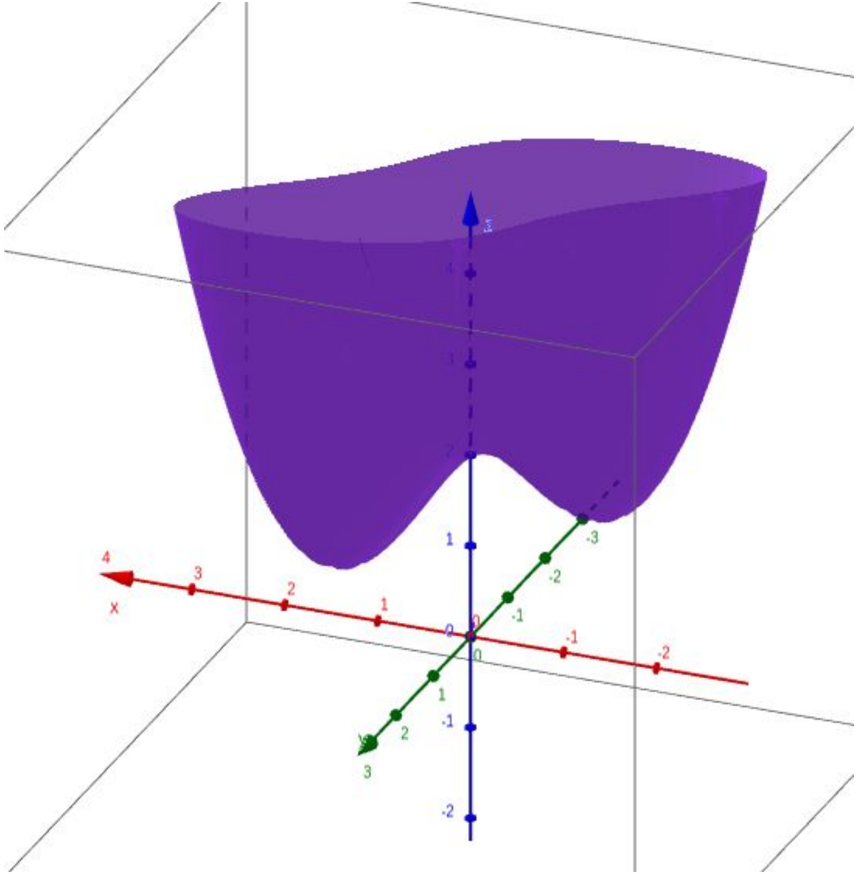




Img src

# $2x^2 - 4xy + y^4 + 2$

- This function has more than local minima
- **Coding common mistake**
  - Computing the partial derivative of each variable using the old state
  - Don’t use the changed state



# Terminology

- In programming, a **parameter** is a **variable in a method definition**.
  - The **arguments** are the values you pass when calling the function
  - Parameter = Variable. Argument = Value
- In ML/neural network, a **parameter** is a **weight value**
  - Parameter = Value (connection weight)
- A model is set of parameters (weights) that we save for later inference
  - (m, c) for the line equation
- A **hyperparameter** is a parameter whose value is used to control the learning process
  - Learning rate is one of the **most critical hyperparameters** to handle
    - Specially its initial value. Secondly, how to decay
  - We will comeback to it again later

# Gradient Descent

- Gradient descent is a **first-order iterative optimization** algorithm for finding a **local minimum** of a **differentiable** function
- Iterative Optimization
  - Start from an initial solution
  - Keep iterating to improve the last solution (*tail recursion*)
- First-order
  - Only first-order derivatives make use of this method
    - In other words, we compute derivatives **only once**
- Another alternative to Gradient Descent is Newton's method
  - Useful, but there are several impractical challenges

# Lectures flow so far

- We wanted to do House Price Prediction (Supervised / Regression)
- The data seems coming from a line
  - Let’s find the best line. This treatment is called linear regression
- Given data and line, we defined a cost (error) function
- We want to minimize the function (find parameters that gives min error)
- Gradient descent: general technique that given a function F, iteratively it finds its (local) minima
- Next: Let’s get back to linear regression + gradient descent

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."