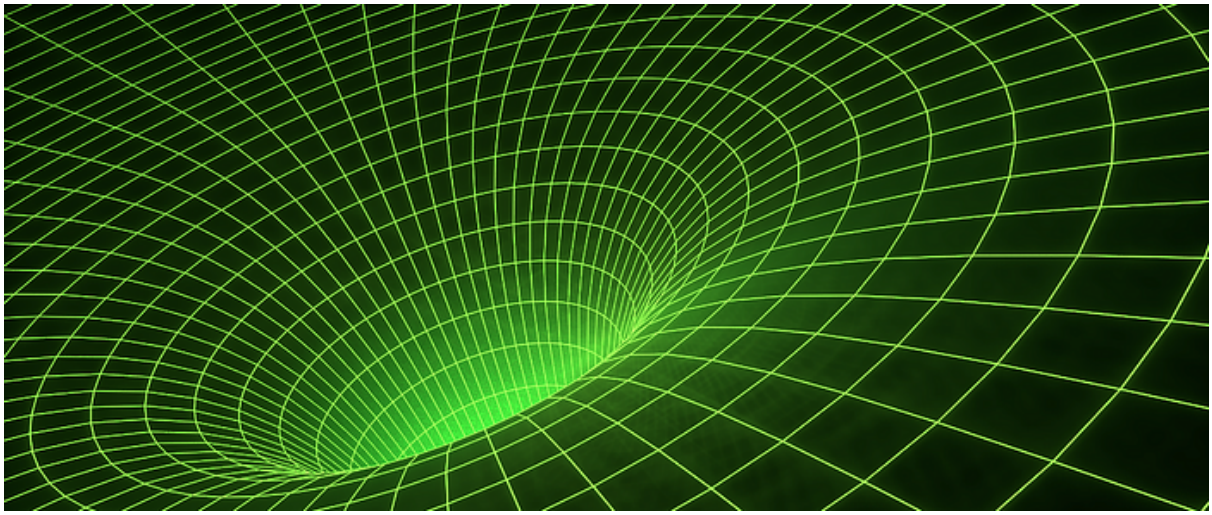
Gradient Descent Algorithm — a deep dive (by R. Kwiatkowski)

The Gradient Descent method lays the foundation for machine learning and deep learning techniques. Let's explore how does it work, when to use it and how does it behave for various functions.



1. Introduction

Gradient descent (GD) is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function. This method is commonly used in *machine learning* (ML) and *deep learning*(DL) to minimise a cost/loss function (e.g. in a linear regression). Due to its importance and ease of implementation, this algorithm is usually taught at the beginning of almost all machine learning courses.

However, its use is not limited to ML/DL only, it's being widely used also in areas like:

- control engineering (robotics, chemical, etc.)
- computer games
- mechanical engineering

That's why today we will get a deep dive into the math, implementation and behaviour of first-order gradient descent algorithm. We will navigate the custom (cost) function directly to find its minimum, so there will be no underlying data like in typical ML tutorials — we will be more flexible in terms of a function's shape.

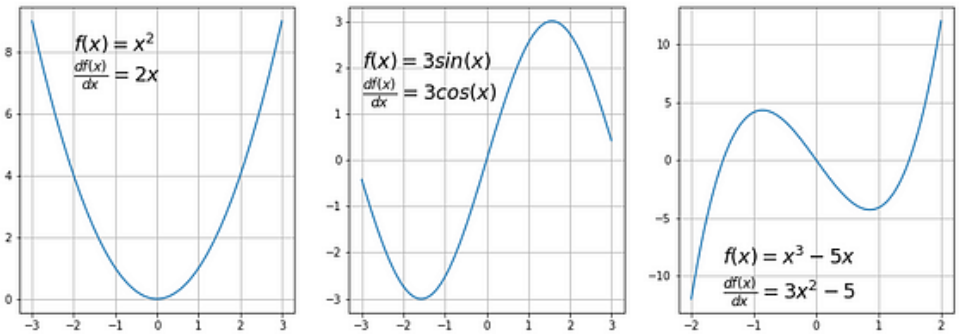
This method was proposed before the era of modern computers and there was an intensive development meantime which led to numerous improved versions of it but in this article, we're going to use a basic/vanilla gradient descent implemented in Python.

2. Function requirements

Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:

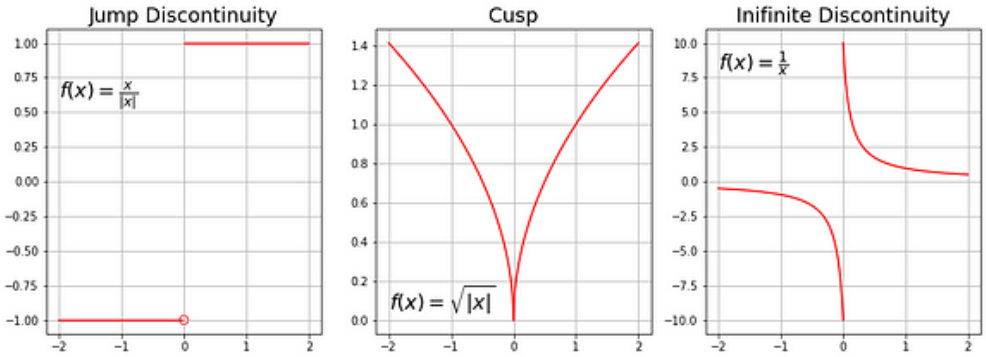
- **differentiable**
- **convex**

First, what does it mean it has to be **differentiable**? If a function is differentiable it has a derivative for each point in its domain — not all functions meet these criteria. First, let's see some examples of functions meeting this criterion:



Examples of differentiable functions; Image by author

Typical non-differentiable functions have a step a cusp or a discontinuity:



Examples of non-differentiable functions; Image by author

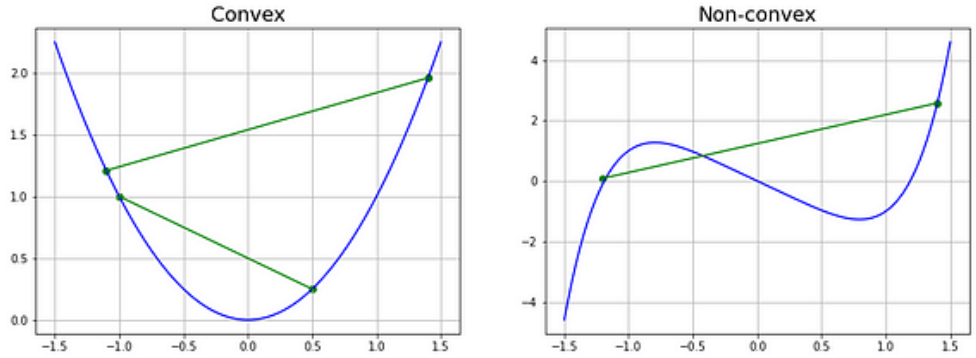
Next requirement — **function has to be convex**. For a univariate function, this means that the line segment connecting two function's points lays on or above its curve (it does not cross it). If it does it means that it has a local minimum which is not a global one.

Mathematically, for two points x1, x2 laying on the function's curve this condition is expressed as:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

where λ denotes a point's location on a section line and its value has to be between 0 (left point) and 1 (right point), e.g. λ=0.5 means a location in the middle.

Below there are two functions with exemplary section lines.



Exemplary convex and non-convex functions; Image by author

Another way to check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0.

$$\frac{d^2 f(x)}{dx^2} > 0$$

Let's do a simple example (*warning: calculus ahead!*).

GIF via giphy

Let's investigate a simple quadratic function given by:

$$f(x) = x^2 - x + 3$$

Its first and second derivative are:

$$\frac{df(x)}{dx} = 2x - 1, \quad \frac{d^2 f(x)}{dx^2} = 2$$

Because the second derivative is always bigger than 0, our function is strictly convex.

It is also possible to use **quasi-convex functions** with a gradient descent algorithm. However, often they have so-called **saddle points** (called also minimax points) where the algorithm can get stuck (we will demonstrate it later in the article). An example of a quasi-convex function is:

$$f(x) = x^4 - 2x^3 + 2$$
$$\frac{df(x)}{dx} = 4x^3 - 6x^2 = x^2(4x - 6)$$

Let's stop here for a moment. We see that the first derivative equal zero at x=0 and x=1.5. This places are candidates for function's extrema (minimum or maximum)— the slope is zero there. But first we have to check the second derivative first.

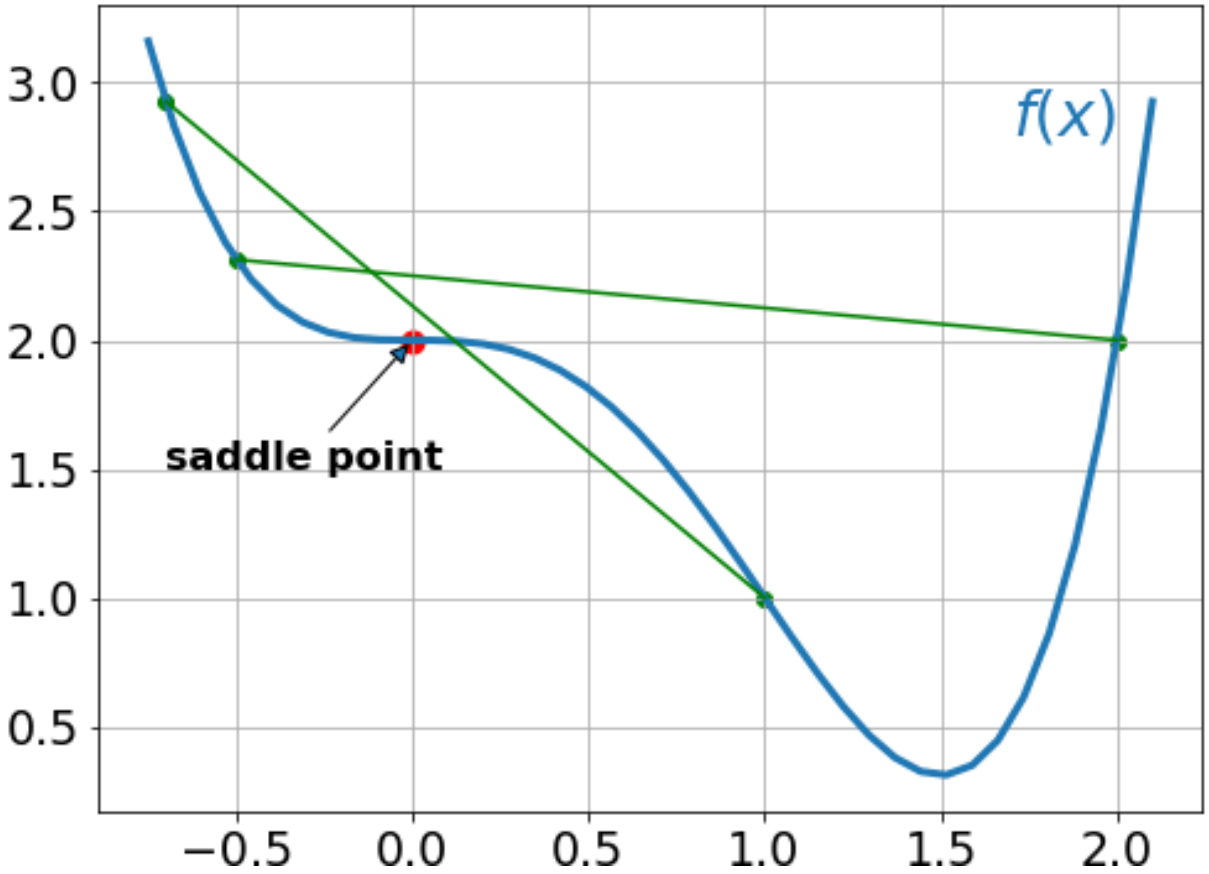
$$\frac{d^2f(x)}{dx^2} = 12x^2 + 12x = 12x(x - 1)$$

The value of this expression is zero for *x=0* and *x=1*. These locations are called an inflexion point — a place where the curvature changes sign — meaning it changes from convex to concave or vice-versa. By analysing this equation we conclude that :

- for x<0: function is convex
- for 0<x<1: function is concave (the 2nd derivative < 0)
- for x>1: function is convex again

Now we see that point *x=0* has both first and second derivative equal to zero meaning this is a saddle point and point x=1.5 is a global minimum.

Let's look at the graph of this function. As calculated before a saddle point is at *x=0* and minimum at *x=1.5*.

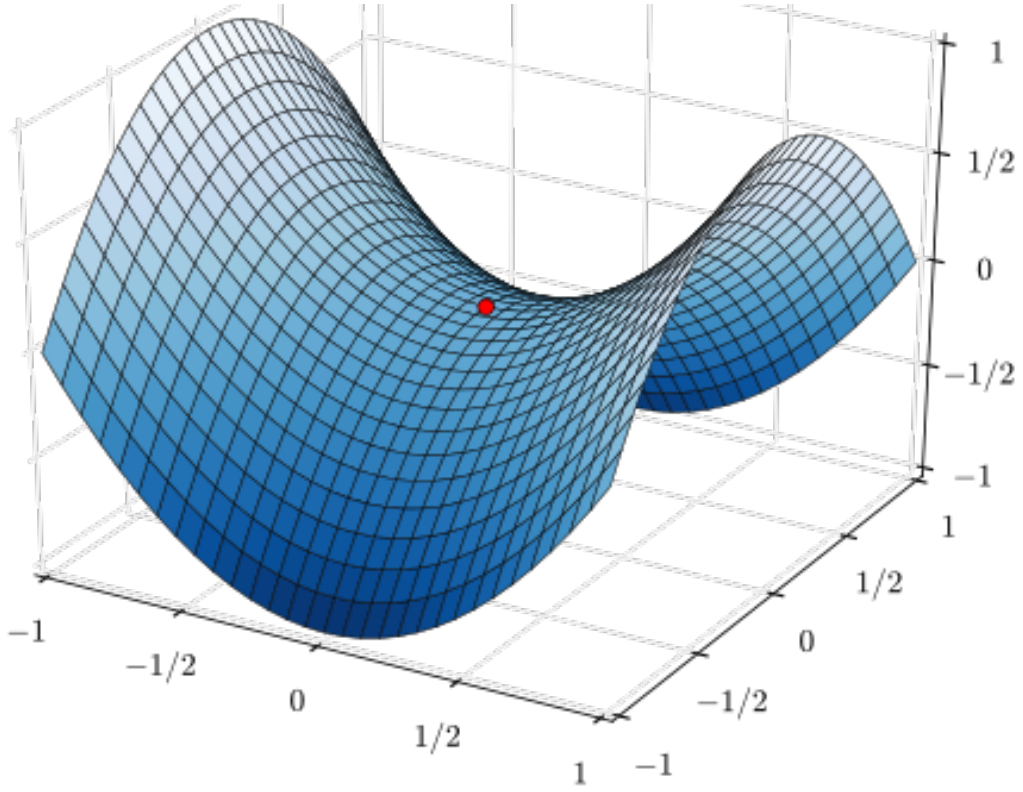


Semi-convex function with a saddle point; Image by author

For multivariate functions the most appropriate check if a point is a saddle point is to calculate a Hessian matrix which involves a bit more complex calculations and is beyond the scope of this article.

Example of a saddle point in a bivariate function is show below.

$$z = x^2 - y^2$$



Nicoguaro, CC BY 3.0, via Wikimedia Commons

3. Gradient

Before jumping into code one more thing has to be explained — what is a gradient. Intuitively it is a slope of a curve at a given point in a specified direction.

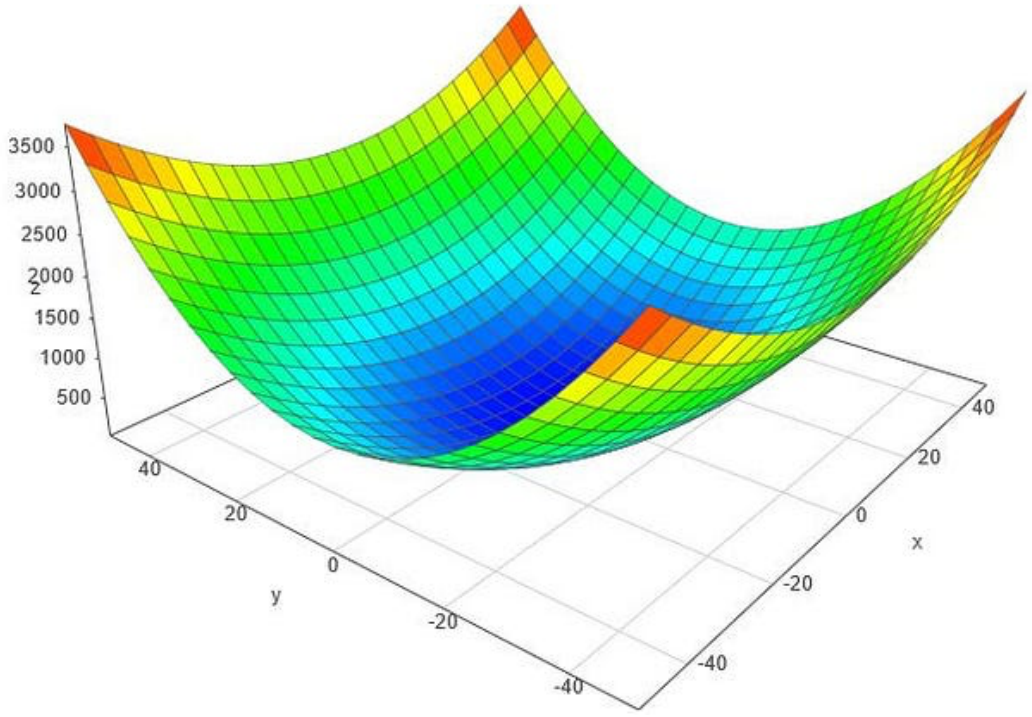
In the case of **a univariate function**, it is simply the **first derivative at a selected point**. In the case of **a multivariate function**, it is a **vector of derivatives** in each main direction (along variable axes). Because we are interested only in a slope along one axis and we don't care about others these derivatives are called **partial derivatives**.

A gradient for an n-dimensional function f(x) at a given point p is defined as follows:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

The upside-down triangle is a so-called *nabla* symbol and you read it “del”. To better understand how to calculate it let’s do a hand calculation for an exemplary 2-dimensional function below.

$$f(x) = 0.5x^2 + y^2$$



3D plot; Image by author

Let’s assume we are interested in a gradient at point p(10,10):

$$\frac{\partial f(x,y)}{\partial x} = x, \quad \frac{\partial f(x,y)}{\partial y} = 2y$$

so consequently:

$$\nabla f(x,y) = \begin{bmatrix} x \\ 2y \end{bmatrix}$$

$$\nabla f(10, 10) = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$

By looking at these values we conclude that the slope is twice steeper along the y axis.

4. Gradient Descent Algorithm

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, scales it (by a learning rate) and subtracts obtained value from the current position (makes a step). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

There's an important parameter **η** which scales the gradient and thus controls the step size. In machine learning, it is called **learning rate** and have a strong influence on performance.

- The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point
- If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

In summary, Gradient Descent method's steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point
3. make a scaled step in the opposite direction to the gradient (objective: minimise)
4. repeat points 2 and 3 until one of the criteria is met:

- maximum number of iterations reached
- step size is smaller than the tolerance (due to scaling or a small gradient).

Below, there's an exemplary implementation of the Gradient Descent algorithm (with steps tracking):

This function takes 5 parameters:

1. **starting poin**t - in our case, we define it manually but in practice, it is often a random initialisation

2. **gradient functio**n - has to be specified before-hand

3. **learning rate** - scaling factor for step sizes

4. maximum number of iterations

5. tolerance to conditionally stop the algorithm (in this case a default value is 0.01)

5. Example 1 — a quadratic function

Let's take a simple quadratic function defined as:
$f(x) = x^2 - 4x + 1$

Because it is an univariate function a gradient function is:
$$\frac{df(x)}{dx} = 2x - 4$$

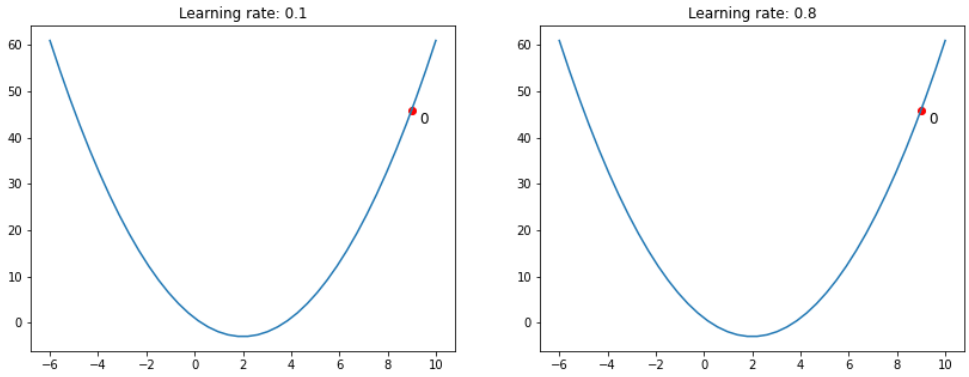
Let's write these functions in Python:

For this function, by taking a learning rate of 0.1 and starting point at x=9 we can easily calculate each step by hand. Let's do it for the first 3 steps:

$$x_1 = 9 - 0.1 \cdot (2 \cdot 9 - 4) = 7.6$$
$$x_2 = 7.6 - 0.1 \cdot (2 \cdot 7.6 - 4) = 6.48$$
$$x_3 = 6.48 - 0.1 \cdot (2 \cdot 6.48 - 4) = 5.584$$

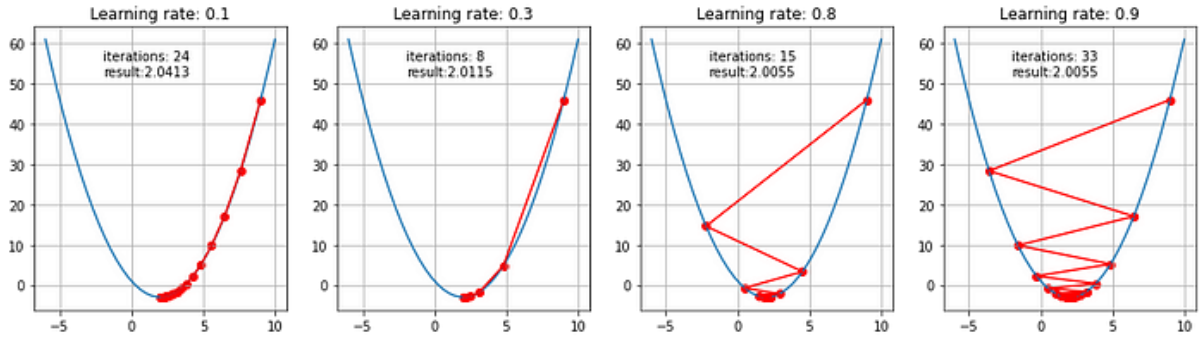
The python function is called by:

The animation below shows steps taken by the GD algorithm for learning rates of 0.1 and 0.8. As you see, for the smaller learning rate, as the algorithm approaches the minimum the steps are getting gradually smaller. For a bigger learning rate, it is jumping from one side to another before converging.



First 10 steps taken by GD for small and big learning rate; Image by author

Trajectories, number of iterations and the final converged result (within tolerance) for various learning rates are shown below:



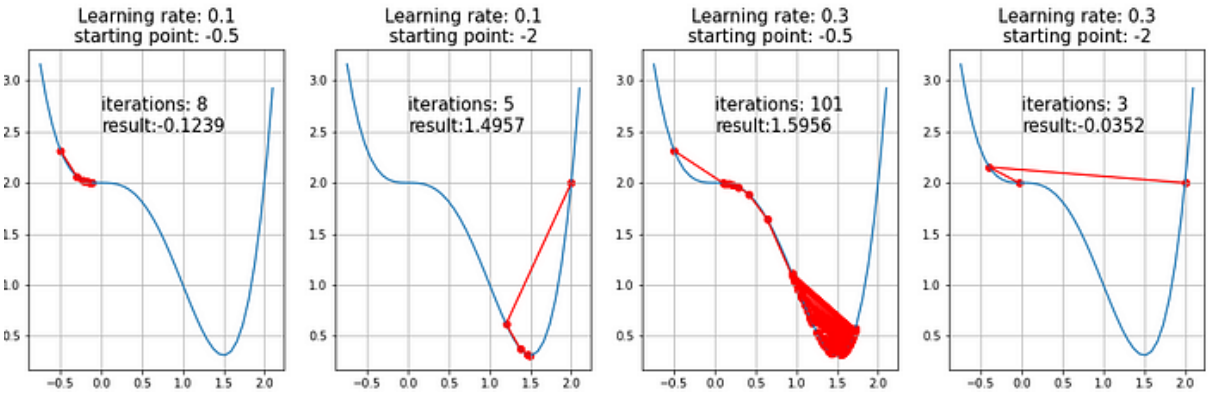
Results for various learning rates; Image by author

6. Example 2 — a function with a saddle point

Now let's see how the algorithm will cope with a semi-convex function we investigated mathematically before.

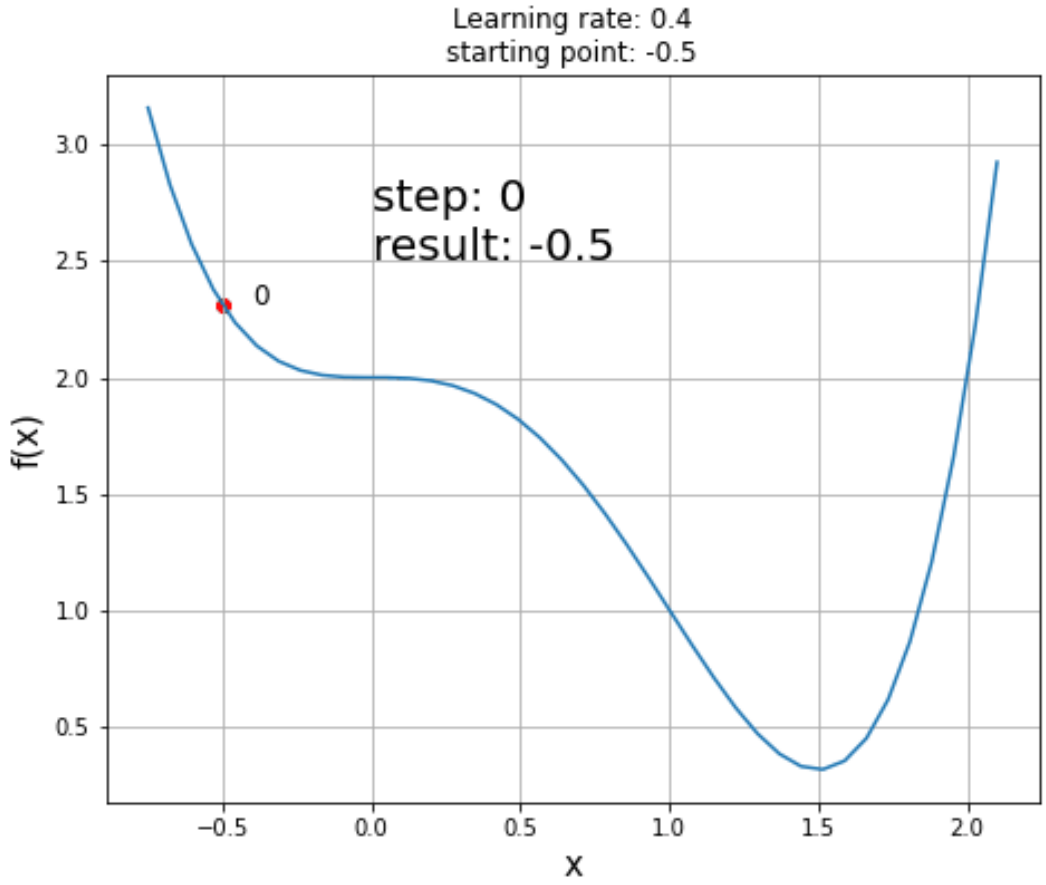
$$f(x) = x^4 - 2x^3 + 2$$

Below results for two learning rates and two different staring points.



GD trying to escape from a saddle point; Image by author

Below an animation for a learning rate of 0.4 and a starting point x=-0.5.



Animation of GD trying to escape from a saddle point; Image by author

Now you see that the existence of a saddle point imposes a real challenge for the first-order gradient descent algorithms like GD and

obtaining a global minimum is not guaranteed. Second-order algorithms deal with these situations better (e.g. Newton-Raphson method).

Investigation of saddle points and how to escape from them is a subject of ongoing studies and various solutions were proposed. For example, Chi Jin and M. Jordan proposed a Perturbing Gradient Descent algorithm — details you find in their blog post.

7. Summary

In this article, we checked how a Gradient Decent algorithm works, when can it be used and what are some common challenges when using it. I hope this will be a good starting point for you to explore more advanced gradient-based optimisation methods like Momentum or Nesterov (Accelerated) Gradient Descent, RMSprop, ADAM or second-order ones like the Newton-Ralphson algorithm.