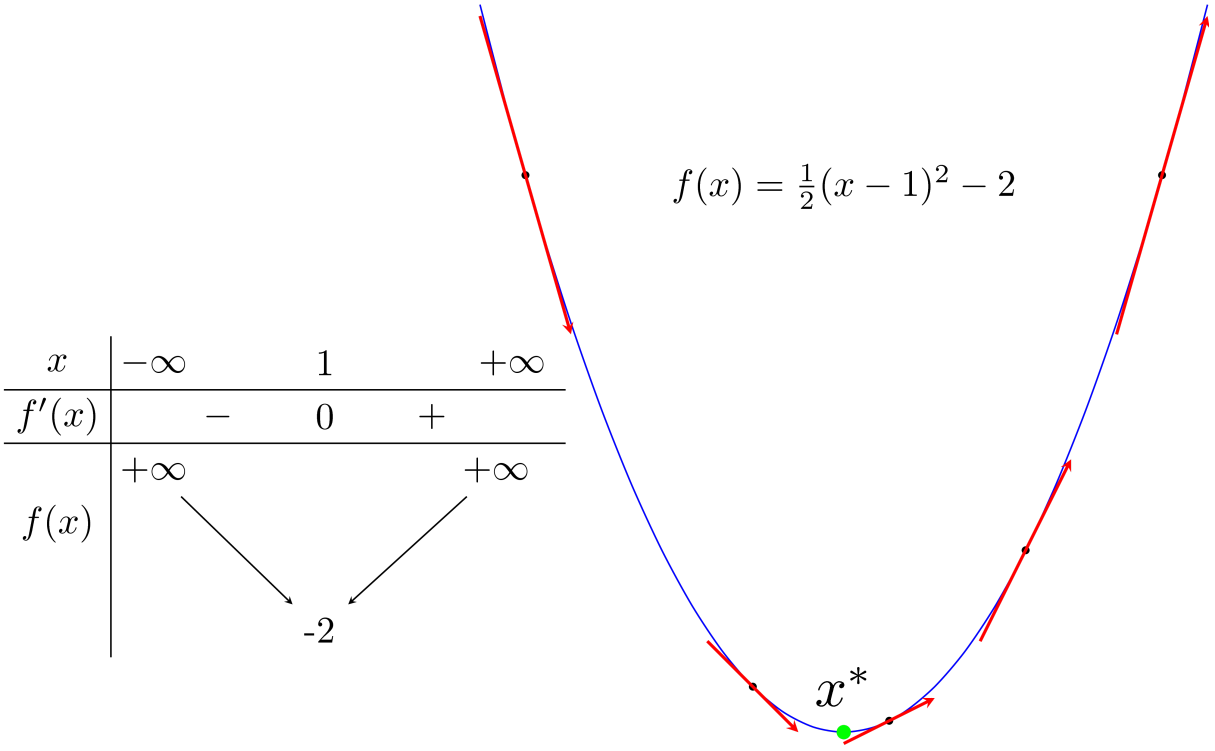
How do Gradients help guiding the Neural Network

The gradient of a loss function represent the rate of change of the loss with respect to a specific weight or bias. It indicates how much the loss will change in response to a small change in that parameter.



You might feel the above figure familiar.

The green point is a local minimum, it is also the point that makes the function reach the smallest value. From now on, I will use “local minimum” instead of “minimum point”, “global minimum” instead of “the point at which the function reaches the smallest value”.

Global minimum is a special case of Local minimum.

Suppose we are interested in a single-variable function that have derivative everywhere. Let me remind you of a few things that are already too familiar:

- The local minimum point x^* of the function is the point that make the derivative $f'(x^*)$ equal to 0. Moreover, in its neighborhood, the derivative of the points to the left of x^* is non-positive, the derivative of the points to the right of x^* is non-negative. (Look at the figure above)
- The tangent line to the graph of the function at any point has a slope equal to the derivative of the function at that point.

In the figure above, the `points to the left` of the green local minimum point have a `negative derivative`, the `points to the right` have a `positive derivative`.

And for this function above, the further to the left of the local minimum point, the more negative the derivative, and the further to the right of that point, the more positive the derivative is.

Gradient Descent

In Machine Learning and Optimization in general, we often need to find the minimum (or sometime maximum) value of a function. For example, the loss functions in the Linear Regression and K-means Clustering problems.

In general, finding the global minimum of the loss functions in ML is very complex, or even impossible. Instead, people often try to find local minimum points, and to some extent, consider them as the solution to the problem.

Local minimum points are solutions to the equation of the derivative equal to 0. If we can somehow find all (infinite) local minimum points, we just need to substitute each local minimum point into the function and then find the point that makes the function reach the smallest value.

However, in most cases, solving the equation of the derivative equal to 0 is impossible. The reasons can come from the complexity of the form of the derivative, from the fact that the data points have a large number of dimensions, or from the fact that there are too many data points.

The most popular approach is to start from a point that we consider to be close to the solution of the problem. Then use an iterative method to gradually approach the point we are looking for, until the derivative is close to 0. Gradient Descent and its variants are one of the most widely used methods.

Gradient Descent for Function of a Single Variable

Back to the original figure. Suppose x_t is the point we found after the t^{th} iteration. We need to find an algorithm to bring x_t closer to x^* as much as possible.

In the first figure, we have two more observations:

1. If the derivative of the function at x_t : $f'(x_t) \geq 0$ then x_t lies in the right hand side of x^* (and so to the opposite). To find the next point x_{t+1} that is closer to x^*, we need to move x_t to the left, i.e. to the negative side. In other words, **we need to move in the opposite direction of the derivative**.

   $$x_{t+1} = x_t + \Delta$$

   where Δ is a quantity with the opposite sign of the derivative
2. As x_t moves further to the right of x^*, $f'(x_t)$ becomes increasingly greater than 0 (and vice versa).

These above observations give us a simple way to update :

$$x_{t+1} = x_t - \eta f'(x_t)$$

In which η (pronounced as eta) is a positive number called the `learning rate`, define how large our steps are.
The minus sign indicates that we have to go against the derivative (This is also the reason why this method is called Gradient Descent - descent means going against).

The simple observation above, although not true for all problems, is the foundation for many optimization methods in general and Machine Learning algorithm in particular.

A simple example with Python

Consider the function $f(x) = x^2 + 5\sin(x)$ with derivative $f'(x) = 2x + 5\cos(x)$ (one reason I chose this function is because it is not easy to find the solution of the derivative equal to 0 like the function above). Assuming we start from a point x_0 somewhere, at the t^{th} iteration, we will update as follows:

$$x_{t+1} = x_t - \eta(2x_t + 5\cos(x_t))$$

```
import math
import numpy as np
import matplotlib.pyplot as plt

# function to compute the derivative
def grad(x):
  return 2*x + 5*np.cos(x)

# function to compute the value of the function (this function is used to check
# if the computation of derivative is right)
def cost(x):
  return x**2 + 5*np.sin(x)

# function to implement Gradient Descent
def GD(eta, x0):
  x = [x0]

  for it in range(100):
    # compute the next point of x
    x_new = x[-1] - eta*grad(x[-1])

    # stop when derivative is small enough
    if abs(grad(x_new)) < 1e-3:
      break

    x.append(x_new)
  return (x, it)
```

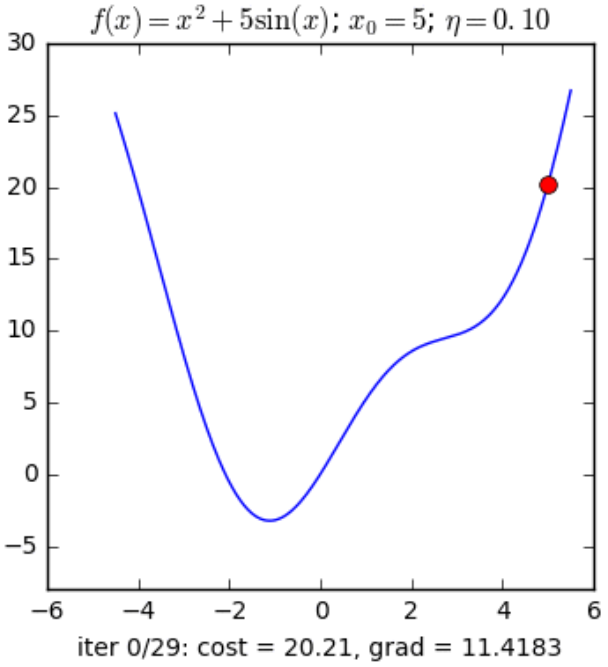
Different initialization points:

Try to find solutions with $x_0 = -5$ and $x_0 = 5$.

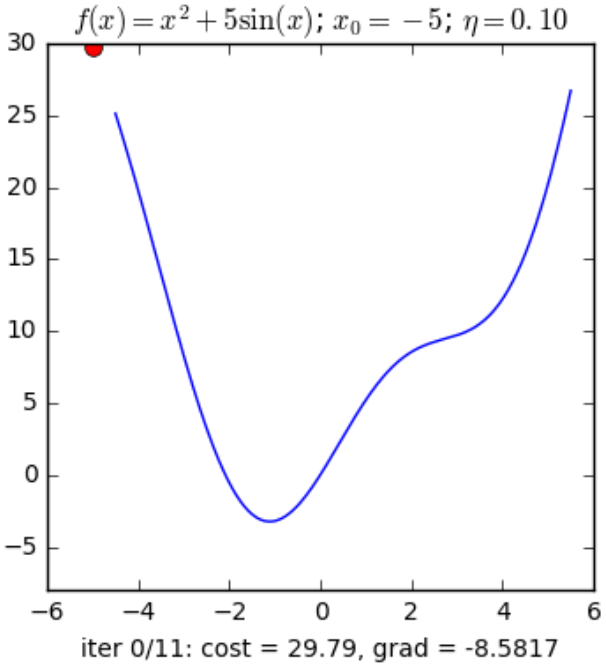
```
(x1, it1) = GD(.1, -5)
(x2, it2) = GD(.1, 5)

print('Solution x1 = %f, cost = %f, obtained after %d iterations'%(x1[-1], cost(x1[-1]), it1))
print('Solution x2 = %f, cost = %f, obtained after %d iterations'%(x2[-1], cost(x2[-1]), it2))
```

```
>> Solution x1 = -1.110667, cost = -3.246394, obtained after 11 iterations
   Solution x2 = -1.110341, cost = -3.246394, obtained after 29 iterations
```

So, with different initialization points, our algorithm finds similar solutions, although with different convergence rates.



iter 0/29: cost = 20.21, grad = 11.4183



iter 0/11: cost = 29.79, grad = -8.5817