Gradient Descent with Momentum

Gradient Descent Algorithm

1. Predict an initialization point $\theta = \theta_0$.
2. Update θ until acceptable results are achieved:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

   With $\nabla_\theta J(\theta)$ is the derivative of the loss function at θ.

Gradient from a physical perspective

The GD algorithm is often compared to the effect of gravity on a marble placed on a valley shaped surface like the figure below. Regardless whether we place the marble at A or B, the marble will eventually roll down and end up at position C.

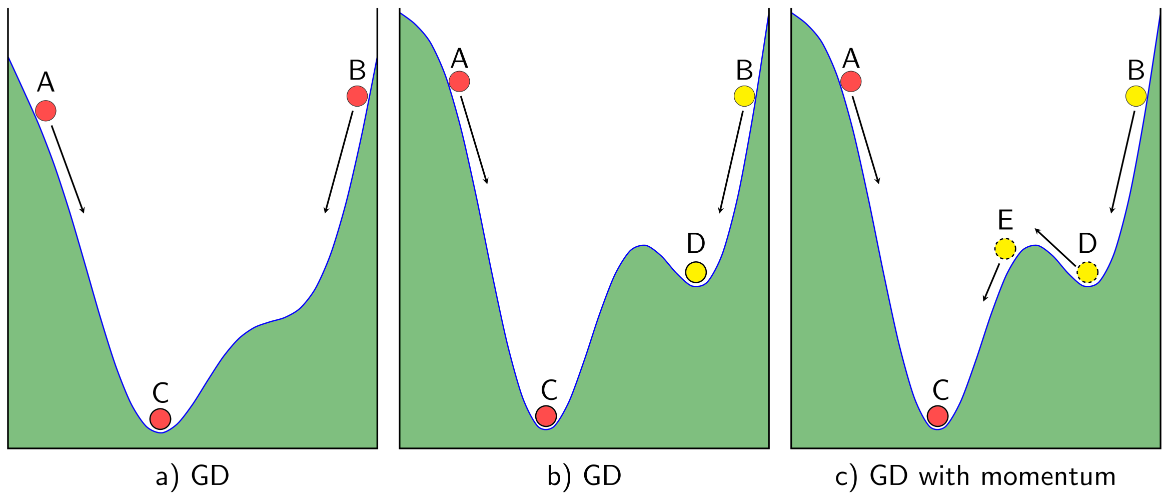

Figure 1: Comparison of Gradient Descent with Physical Phenomena

However, if the surface has two valley bottoms as shown in Figure 1b), depending on whether the ball is placed at A or B, the final position of the ball will be at C or D. Point D is a local minimum point that we do not expect.

If you think about it more physically, still in Figure 1b), if the initial velocity of the ball at point B is large enough, when the ball rolls to point D, following the momentum, the ball can continue to move up the slope to the left side of D. And if the initial velocity is assumed to be even larger, the ball can go uphill to point E and then roll down to C as shown in Figure 1c). This is exactly what we want.

Readers may ask whether the ball rolling from A to C follows *the momentum of* rolling to E and then to D. The answer is that this is more unlikely because compared to the DE slope, the CE slope is much higher.

Based on this phenomenon, an algorithm was created to overcome the problem of GD's solution falling into an undesirable local minimum point. That algorithm is called Momentum.

Gradient Descent with Momentum

How can we represent momentum mathematically?

In GD, we need to calculate the **amount of change** at time point t to update the new position for the solution (i.e. the marble). If we think of this quantity as **velocity** v_t, in physics, the marble's new position would be $\theta_{t+1} = \theta_t - v_t$. The minus sign represents having to move against the derivative.

Our job now is to compute the **velocity** v_t so that it both carries the information about `slope` (i.e. the derivative) and also carries information about the `momentum`, which is the previous velocity v_{t-1} (consider that the initial velocity $v_0 = 0$).

In the simplest way, we can add (weighted) these two quantities:

$$\text{velocity}\;\; v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

Where :

- γ is usually chosen as a value around 0.9
- v_{t-1} is the velocity at the previous time point
- $\nabla_\theta J(\theta)$ is the slope of the previous point

Then the new position of the marble is determined as follows:

$$\begin{aligned}\theta &= \theta - v_t \\ &= \theta - \gamma v_{t-1} + \eta \nabla_\theta J(\theta)\end{aligned}$$

This simple algorithm proves to be very effective in practical problems (in high dimensional space, the calculation method is completely similar). Below is an example in one-dimensional space.

A simple example

Let’s consider a simple function with two local minimum points, of which 1 point is the global minimum:

$$f(x) = x^2 + 10\sin(x)$$

The derivative is: $f'(x) = 2x + 10\cos(x)$.

Figure 2 below shows the difference between the GD algorithm and the GD algorithm with momentum:

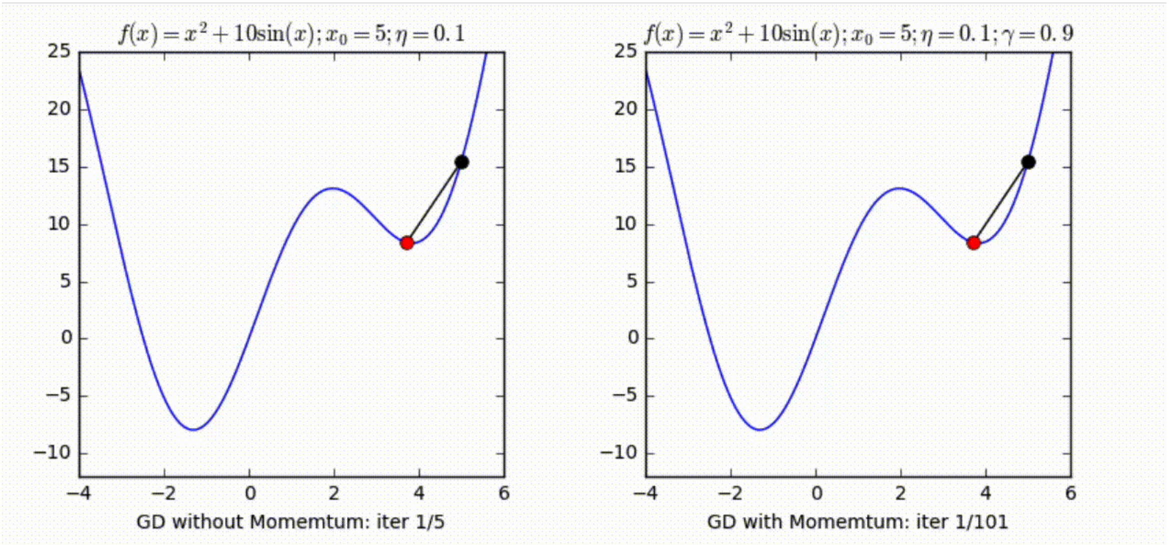

Figure 2: Illustration of GD algorithm with Momentum.

The figure on the left is the path of the solution when not using `momentum`, the algorithm converges after only 5 iterations but the solution found is a **local minimum.**

The figure on the right is the path of the solution when using `momentum`, the marble was able to climb the slope to the area near the global minimum point, then oscillate around this point, decelerate and finally reach the destination. Although it takes more iterations, GD with Momentum gives us a more accurate solution.

Remember the above formula to compute velocity v_t:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

And then use that velocity to update the position of the marble:

$$\theta = \theta - v_t$$

If we know the initial ball placement point theta, the derivate of loss function at a random point grad(theta), the informations of the previous velocity gamma, and the learning rate eta, we can write the function GD_momentum as follows:

```
# check convergence
def has_converged(theta_new, grad):
    # Check if the norm of the gradient of theta_new is less than epsilon
    # This indicates that the gradient is close to zero, and the algorithm has converged
    return np.linalg.norm(grad(theta_new)) / len(theta_new) < 1e-3


def GD_momentum(theta_init, grad, eta, gamma):
    theta = [theta_init]                # path of the marble (theta_init = marble placement point)
    v_old = np.zeros_like(theta_init)   # velocity vector initialization

    for it in range(100):
        v_new = gamma * v_old + eta * grad(theta[-1]) # update the velocity v_t
        theta_new = theta[-1] - v_new  # update the marble position

        if has_converged(theta_new, grad):
            break

        # update path history and velocity
        theta.append(theta_new)
        v_old = v_new
    return theta

# this variable theta includes all points in the path
```