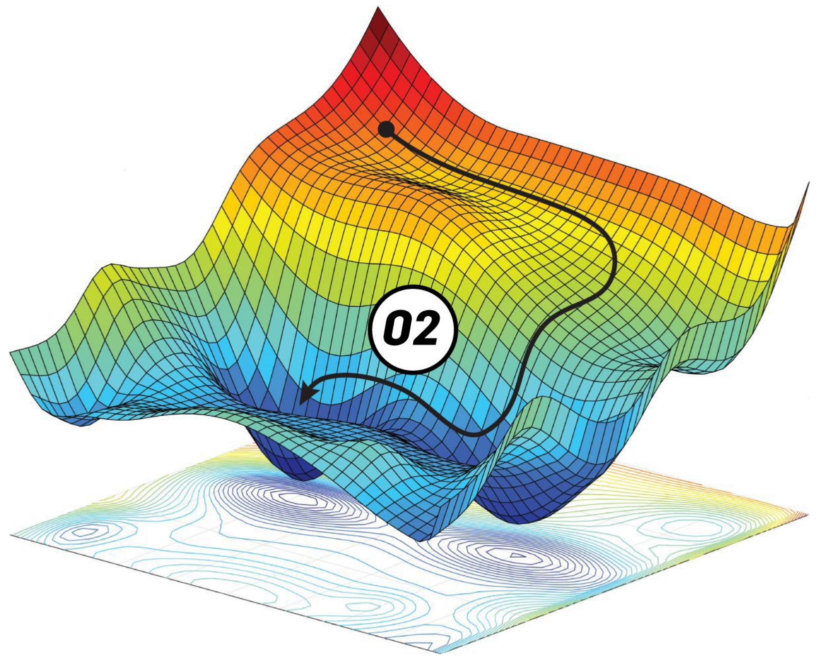
LESSON 4: GRADIENT DESCENT - OPTIMIZATION ALGORITHMS



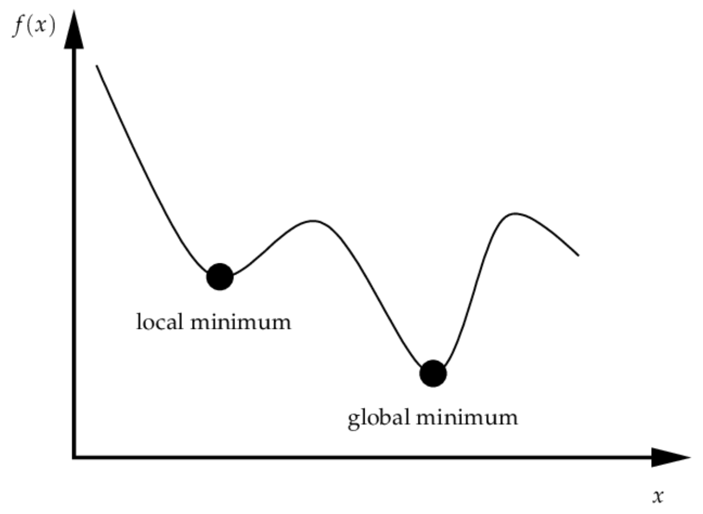
This lecture was refered by machinelearningcoban.com

1. Gradient descent introduction

In machine learning, we usually have to find the optimal point of a function (minimum or maximum).

For example: finding minimum of MSE loss function of Linear regression ...

In calculus, we split minimum point into 2 types: Local minimum and Global minimum



In theory, we can solve derivative equation to find all of local minimum points and choose the smallest to find global minimum point.

But, in reality, it's too hard (or maybe impossible) to solve the derivative equation because of the large-scale dataset, the complexity of the derivative equation or the high-dimension data points etc.

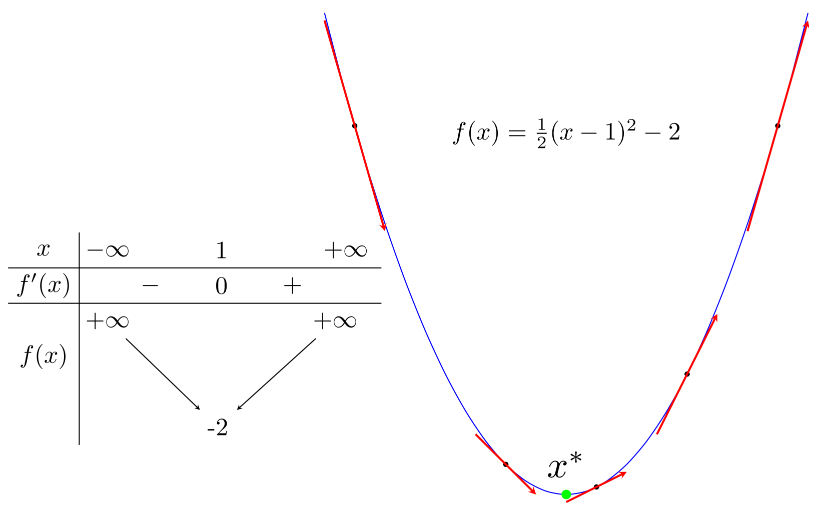
That's why we need to build an algorithm to gradually archieve local minimum point instead of finding it directly.

GRADIENT DESCENT solved the problem and become the most important part in lots of machine learning algorithms and deep learning models till now.

2. Gradient descent for singlevariate function

2.1. Formula

We have a simple singlevariate function and x^* is an minimum point.



Assume that we have x_t in the t iteration of gradient descent algorithm, our mission is to bring x_t near to x^*.

If $f'(x_t) > 0$, so $x_t > x^*$ and if $f'(x_t) < 0$, so $x_t < x^*$.

We have $x_{t+1} = x_t + \delta$ which delta has different sign of $f'(x_t)$ (if $f'(x_t) > 0$, so $\delta < 0$ and vice versa).

The larger distance between x_t and x^*, the higher value of $|f'(x_t)|$. So, $\delta = -\eta f'(x_t)$ and $x_{t+1} = x_t - \eta f'(x_t)$.

η is call ***learning rate*** which is really important in optimization. And the sign - is the reason why we call this algorithm ***Gradient descent***.

2.2. Implementation

We have an example:

$$f(x) = x^2 + 5\sin(x)$$
$$f'(x) = 2x + 5\cos(x)$$
$$x_{t+1} = x_t - \eta(2x + 5\cos(x))$$

In [1]:
```
import numpy as np
```

In [2]:
```
def my_function(x):
    return x ** 2 + 5 * np.sin(x)
```

In [3]:
```
def my_grad(x):
    return 2 * x + 5 * np.cos(x)
```

In [4]:
```
def my_gradient_descent(lr, x_0):
    x = [x_0]
    for it in range(100):
        # Caculate gradient and update into x
        x_new = x[-1] - lr * my_grad(x[-1])

        # Stopping criteria
        if abs(my_grad(x_new)) < 1e-3:
            break

        x.append(x_new)
    return x
```

In [5]:
```
solution_1 = my_gradient_descent(
    lr=0.1,
    x_0=-5
)
solution_1
```

Out[5]:
```
[-5,
 -4.141831092731613,
 -3.0434140487394945,
 -1.9371390635788721,
 -1.370609623535342,
 -1.1959138533062952,
 -1.1398126662660861,
 -1.1207324901805855,
 -1.1140974995041208,
 -1.1117718342401366,
 -1.1109543623859697,
 -1.1106667365268623]
```

In [6]:
```
solution_2 = my_gradient_descent(
    lr=0.1,
    x_0=5
)
solution_2
```

Out[6]:
```
[5,
 3.8581689072683867,
 3.463564567930569,
 3.2451582916682646,
 3.0934475688734215,
 2.9741786797296776,
 2.8723524342019475,
 2.7798685851337033,
 2.691538912182054,
 2.6034429924417726,
 2.512083663118539,
 2.413825273788166,
 2.3043909242955314,
 2.178284700900974,
 2.028031263811057,
 1.8431593967550366,
 1.6090315913519224,
 1.3063382475764564,
 0.9143774850440367,
 0.42636006838025553,
 -0.11415049832376267,
 -0.5880663503407275,
 -0.8864605464168875,
 -1.025247680965368,
 -1.0796417320111382,
 -1.0995355411928176,
 -1.1066334337506414,
 -1.1091439570842945,
 -1.1100292207856688,
 -1.1103410483948122]
```

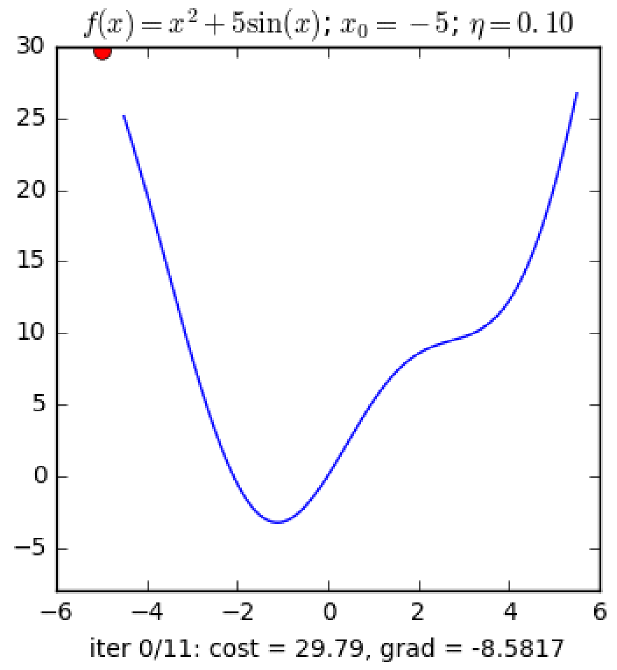
In [7]:
```
my_function(solution_1[-1])
```

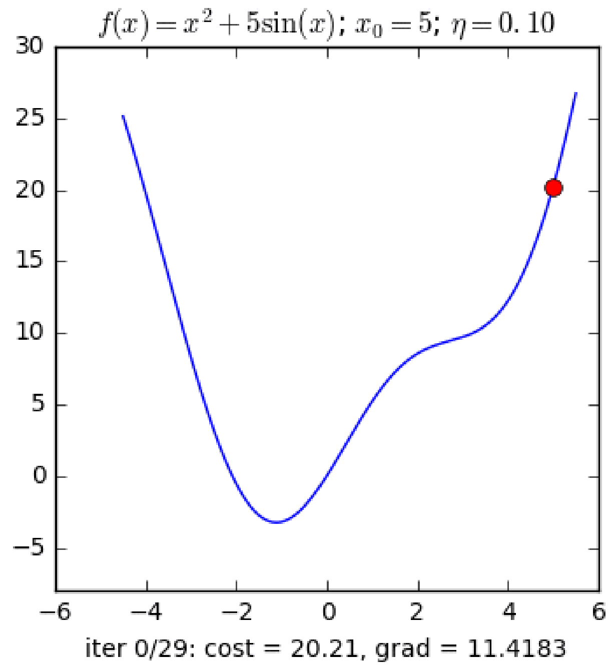
Out[7]:
```
-3.2463941936103735
```

In [8]:
```
my_function(solution_2[-1])
```

```
Out[8]: -3.246394179661889
```

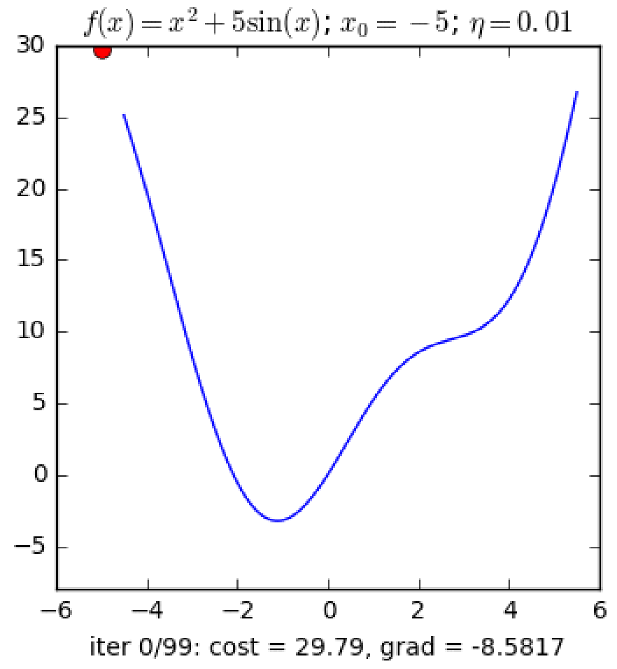
Compare initialization

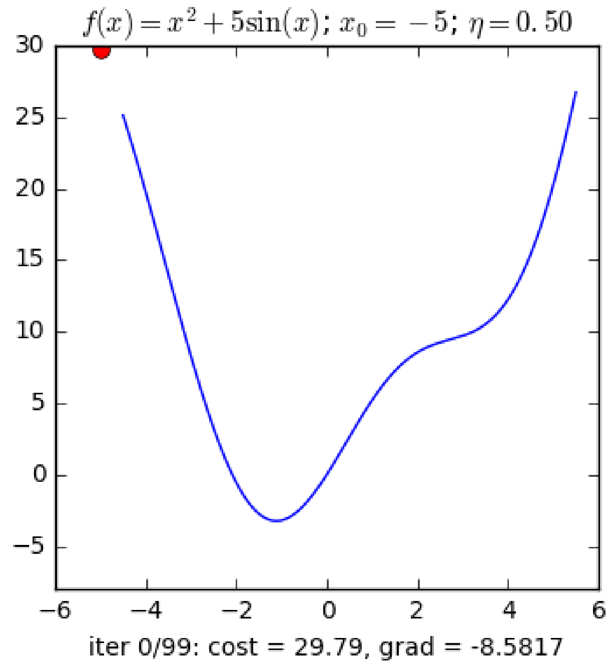




With initialization $x_0 = -5$, the solution is converged much faster than initialization $x_0 = 5$.

Compare learning rate





With learning rate $\eta = 0.01$, the solution is converged too slow (almost 10 times compare with learning rate $\eta = 0.1$)

With learning rate $\eta = 0.5$, the solution is converged faster but it cannot archieve the local minimum because of large step.

3. Gradient descent for multivariate function

3.1. Formula

One example of multivariate function is Linear regression.

The loss function of Linear regression,

$$MSE(w) = \frac{1}{N}\frac{1}{2}(Xw - y)^2$$

and the derivative of the loss function,

$$\frac{\partial MSE}{\partial w} = \frac{1}{N}X^T(Xw - y)$$

3.2. Implementation

Prepare linear regression data and model

In [9]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
```

In [10]:
```
df = pd.read_csv('../data/linear_regression_salary_data.csv')

X = np.array([df.YearsExperience.to_list()])
y = np.array([df.Salary.to_list()])
```

In [11]:
```
def prepare_X_ones(X):
    x_1 = np.ones_like(X)
    print('x_0.shape', x_1.shape)

    X = np.concatenate((x_1, X), axis=0).T
    print('X.shape', X.shape)

    return X
```

In [12]:
```
X_with_1 = prepare_X_ones(X)
```

```
x_0.shape (1, 30)
X.shape (30, 2)
```

In [13]:
```
sklearn_linear_regression = LinearRegression(fit_intercept=False)
sklearn_linear_regression.fit(X_with_1, y.T)
sklearn_linear_regression.coef_
```

Out[13]:
```
array([[25792.20019867,  9449.96232146]])
```

Implement gradient descent

In [14]:
```
def my_mse(w, X, y):
    N = X.shape[0]
    return (1 / N) * (1 / 2) * np.linalg.norm(y - X.dot(w), 2) ** 2
```

In [15]:
```
def my_mse_grad(w, X, y):
    N = X.shape[0]
    return 1 / N * X.T.dot(X.dot(w) - y)
```

In [16]:
```
def my_mse_gradient_descent(lr, w_init, X, y):
    w = [w_init]
    for it in range(2000):

        # Caculate gradient and update into x
        w_new = w[-1] - lr * my_mse_grad(w[-1], X, y)

        # Stopping criteria
        if np.linalg.norm(my_mse_grad(w_new, X, y)) / len(w_new) < 1e-3:
            break

        w.append(w_new)
    return w, it
```

In [17]:
```
w_init = np.array([[2], [1]])
```

In [18]:
```
solution_w_1, iteration_1 = my_mse_gradient_descent(
    lr=0.05,
    w_init=w_init,
    X=X_with_1,
    y=y.T
)
```

In [19]:
```
solution_w_1[-1]
```

Out[19]:
```
array([[25792.19080029],
       [ 9449.96371613]])
```

In [20]:
```
iteration_1
```

Out[20]:
```
1387
```

In [21]:
```
solution_w_2, iteration_2 = my_mse_gradient_descent(
    lr=0.5,
    w_init=w_init,
    X=X_with_1,
    y=y.T
)
```

```
<ipython-input-16-c92b5830d136>:6: RuntimeWarning: invalid value encountered in subtract
  w_new = w[-1] - lr * my_mse_grad(w[-1], X, y)
```

In [22]:
```
solution_w_2[-1]
```

Out[22]:
```
array([[nan],
       [nan]])
```

In [23]:
```
iteration_2
```

Out[23]:
```
1999
```

In [24]:
```
solution_w_3, iteration_3 = my_mse_gradient_descent(
    lr=0.01,
    w_init=w_init,
    X=X_with_1,
    y=y.T
)
```

In [25]:
```
solution_w_3[-1]
```

Out[25]:
```
array([[25446.65875162],
       [ 9501.23922556]])
```

In [26]:
```
iteration_3
```

Out[26]:
```
1999
```

In [27]:
```
sklearn_linear_regression.coef_
```

Out[27]:
```
array([[25792.20019867,  9449.96232146]])
```

4. Optimization algorithm based on Gradient descent

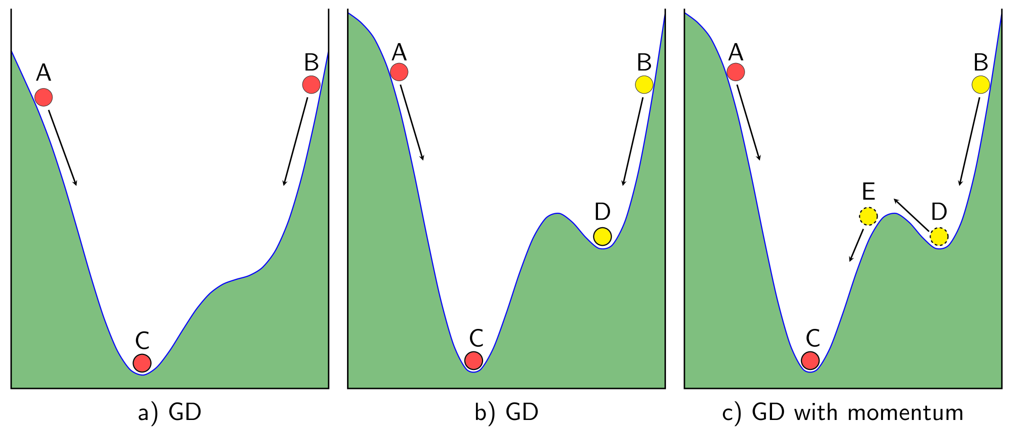
4.1. Momentum

In physical view, in the image a), a ball will come to C even it starts falling from A or B.

In the image b), a ball which is falling from A will come to C and a ball from B can be stucked at D.

If a ball from B has a high velocity, it can overcome D and fall into C.

This is the role of momentum in physics.



Applying this idea of momentum into gradient descent, instead of using only the slope,

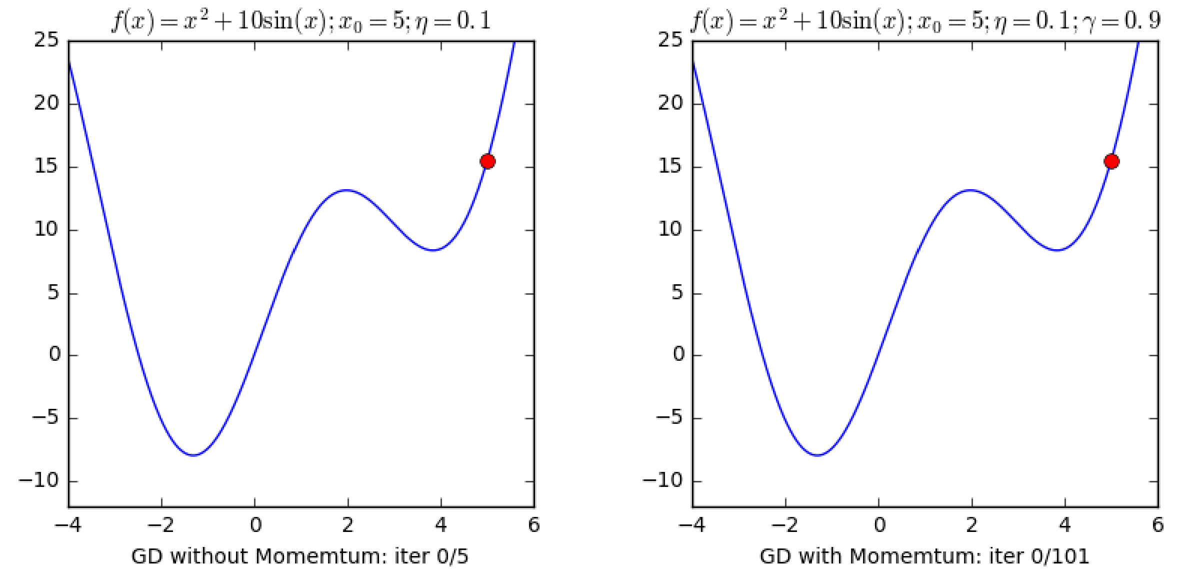
$$w_{t+1} = w_t - \eta \nabla_{w_t} L(w_t)$$

we add an additional value of v_{t-1} while calculating the value of w_{t+1}.

$$v_t = \gamma v_{t-1} + \eta \nabla_{w_t} L(w_t)$$
$$w_{t+1} = w_t - v_t$$

We have an example of single variate function

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



Without momentum on the left, the solution is converged into local minimum instead of global minimum with momentum on the right.

4.2. Nesterov accelerated gradient (NAG)

One remaining problem of Momentum is slow convergence rate. While getting closer to the minimum, the solution take lots of time to be really converged.

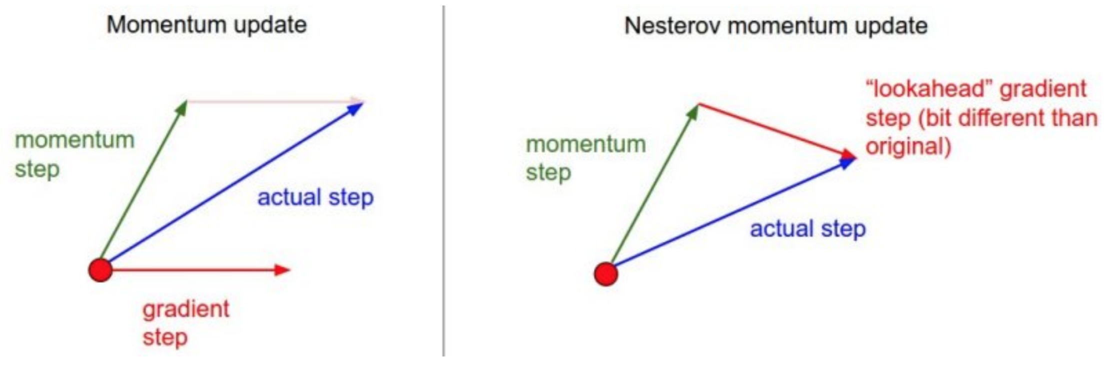
That's why we have Nesterov accelerated gradient.

With normal momentum, we calculate w_{t+1} base on the exact value of w_t.

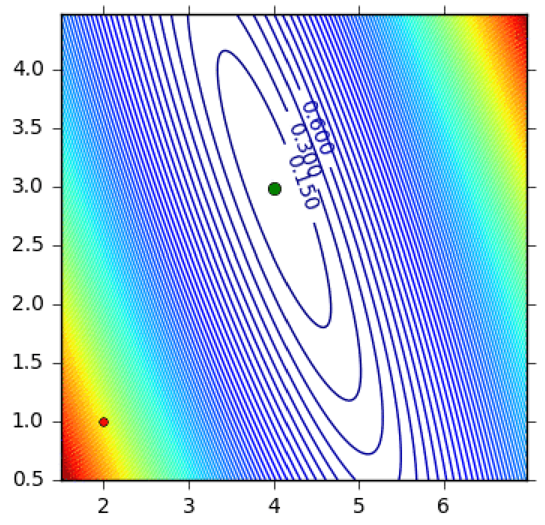
$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_{w_t} L(w_t) \\ w_{t+1} &= w_t - v_t \end{aligned}$$

With NAG, to calculate w_{t+1}, we base on the approximate value of w_{t+1} which calculate by $w_{t+1}^{\text{approx}} = w_t - \gamma v_{t-1}$.

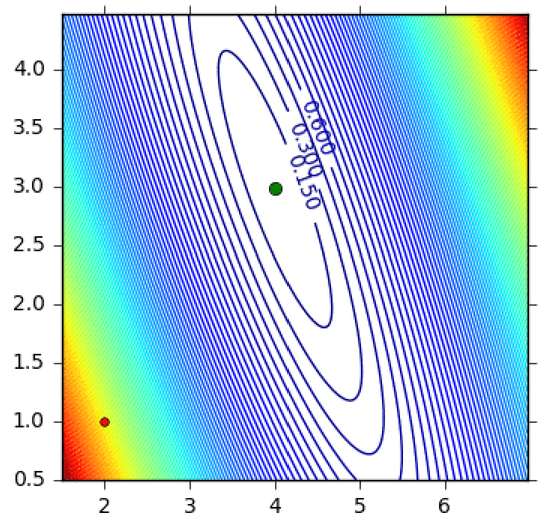
$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_{w_{t+1}^{\text{approx}}} L(w_{t+1}^{\text{approx}}) \\ &= \gamma v_{t-1} + \eta \nabla_{w_t} L(w_t - \gamma v_{t-1}) \\ w_{t+1} &= w_t - v_t \end{aligned}$$



NAG estimates the gradient of next step to modify the update to be more suitable.



η= 1; iter = 0/82; ||grad||_2 = 3.381



η= 1; iter = 0/30; ||grad||_2 = 3.381

Solution from NAG is converged much faster than Momentum without zigzag curve.

4.3. Stochastic gradient descent - Mini-batch gradient descent

Gradient descent, which we have talked about, work by calculating gradient by all data points in the dataset. And it's impossible while working with the dataset contains million or billion data points.

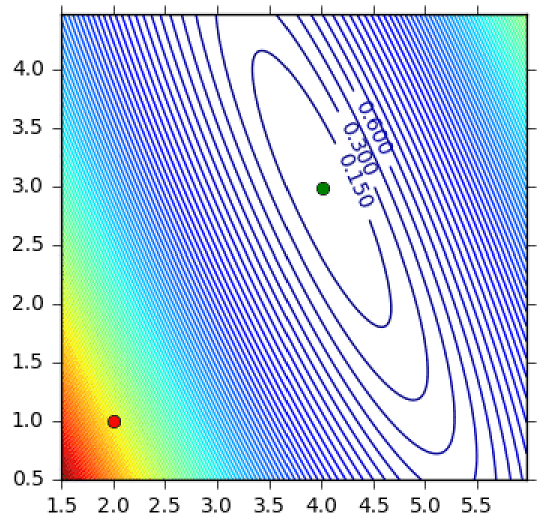
That's why we need another solution which called Stochastic gradient descent or SGD.

Instead of running with the whole dataset feature X and label Y,

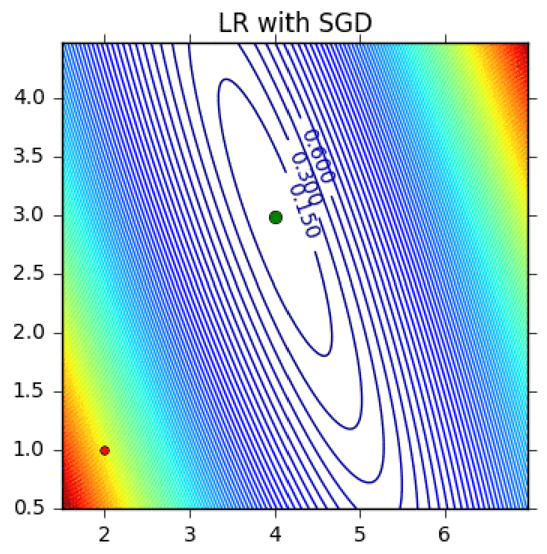
$$w_{t+1} = w_t - \eta \nabla_{w_t} L(w_t, X, y)$$

SGD update w for each data point in the dataset,

$$w_{t+1} = w_t - \eta \nabla_{w_t} L(w_t, x_i, y_i)$$



η= 1; iter = 0/49; ||grad||_2 = 3.381



η= 0.1; iter = 0/2911; ||grad||_2 = 3.381

SGD on the right is not stable like GD on the left, but solution from SGD still can be converged.

But, there is a problem of SGD. The outlier data point can makes the gradient update extremely large and break the training process.

That's why we need a solution called Mini-batch gradient descent.

Instead of running with only 1 data point feature x_i and its label y_i, we calculate gradient update by a batch of data point.

$$w_{t+1} = w_t - \eta \nabla_{w_t} L(w_t, x_{i:i+n}, y_{i:i+n})$$

while: n is batch size.

Both Stochastic gradient descent and Mini-batch gradient descent differ from original Gradient descent by number of data point to calculate gradient update. And both of them can be used with improved techniques such as Momentum, NAG etc.

```
In [ ]:
```