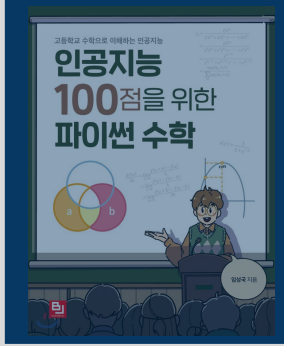
11. 경사와 미분

인공지능 100점을 위한 파이썬 수학



Contents

Contents

1. 미분의 수학적 정의

1. 미분의 수학적 정의

평균속도

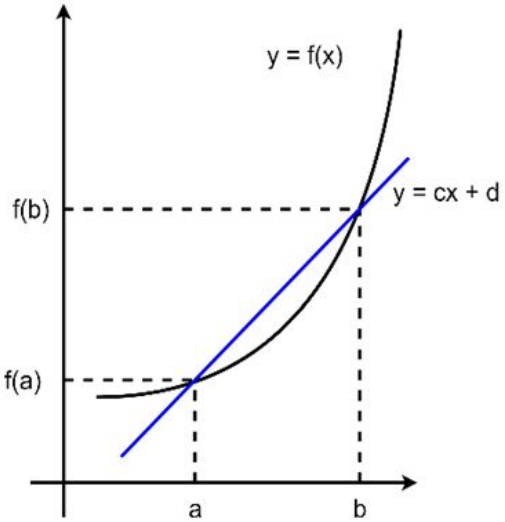
그리스의 철학자 제논은 운동이 불가능하다는 논증을 하면서 어느 한 짧은 순간에서의 위치의 변화는 일어나지 않는다는 것과 시간이 그 짧은 위치 변화가 일어나지 않는 순간들의 연속임을 말합니다. 즉, 움직이고 있다는 것은 모두 아주 짧은 찰나에 보면 정지해 있는 상태란 것입니다. 결국 정지한 상태의 연속일 뿐이니 움직임은 없다는 주장(궤변)이었습니다. 그로부터 2천 년 후 아이작 뉴턴과 라이프니츠는 움직이는 것은 어느 한순간을 보면 분명히 움직이지 않고 있지만 여전히 운동을 나타내는 무언가를 가지고 있다고 생각하고 그것을 찾기 시작했습니다. 그리고 그들이 찾아낸 것이 아주 짧은 시간 동안의 변화였습니다. 이 변화를 알기 위해 움직이고 있는 자동차를 생각해 보겠습니다.

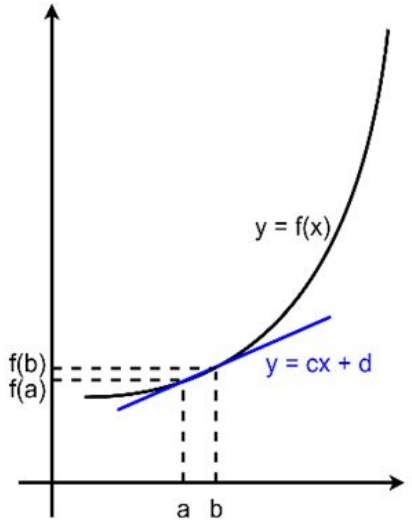
$$v_{av} = \frac{s_2 - s_1}{t_2 - t_1}$$

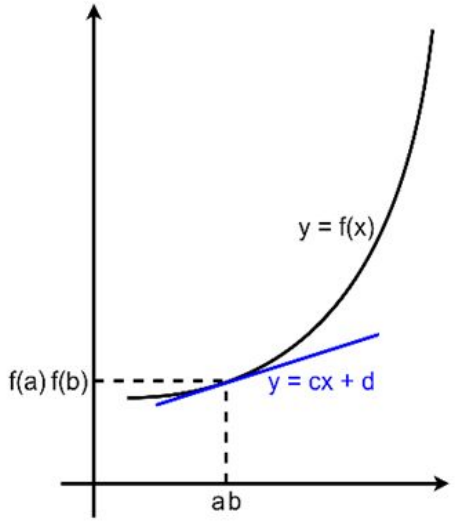
1. 미분의 수학적 정의

- 속도 = 거리 / 시간

$$c = \frac{f(b) - f(a)}{b - a}$$







1. 미분의 수학적 정의

- 순간속도

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

2. 수치미분

2. 수치미분

- 미분정의

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$= \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

$$= \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

2. 수치미분

- 도함수

$$f(x) = x^n \quad \rightarrow \quad f'(x) = nx^{n-1}$$

$$f(x) = e^x \quad \rightarrow \quad f'(x) = e^x$$

2. 수치미분

- 곱셈과 덧셈의 도함수

f와 g는 함수이고, c가 상수일 때

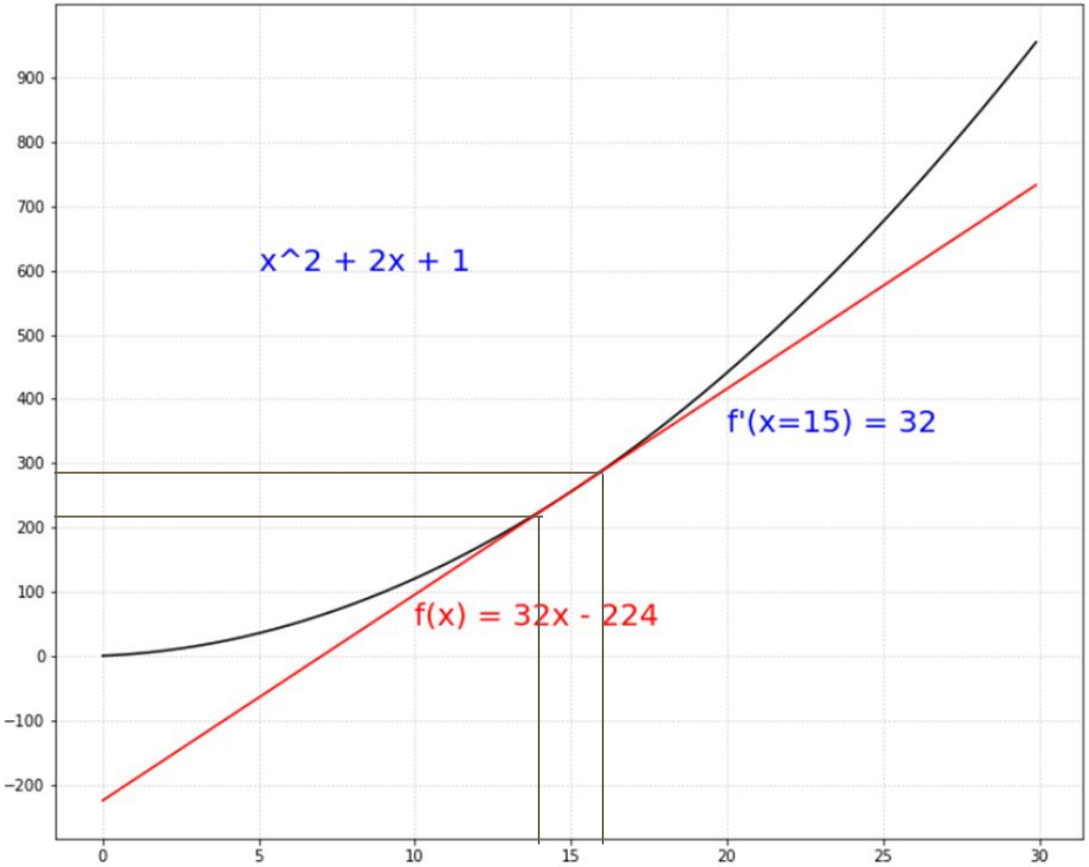
$$(cf)' = cf'$$

$$(f + c)' = f'$$

$$(f + g)' = f' + g'$$

$$(fg)' = f'g + fg'$$

2. 수치미분



$$f(x) = x^2 + 2x + 1$$

$$f'(x) = 2x + 2$$

$$\frac{f(15+0.1)-f(15-0.1)}{0.2}$$

3. 편미분

3. 편미분

출력의 비중을 조정

변수가 2개 이상인 함수의 미분

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

$$f'(x,y) -- X$$

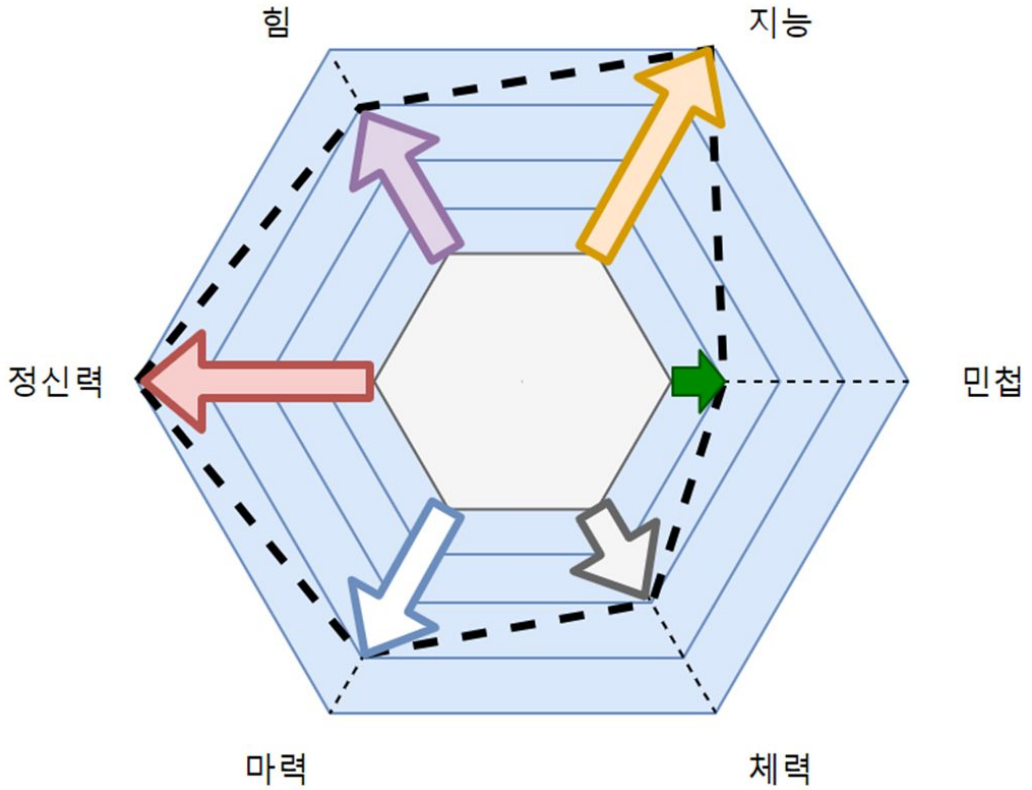
f(x,y) 를 x 에 대해서 미분 // f(x,y) 를 y 에 대해서 미분..

4. N차원

4. N차원

게임 캐릭터

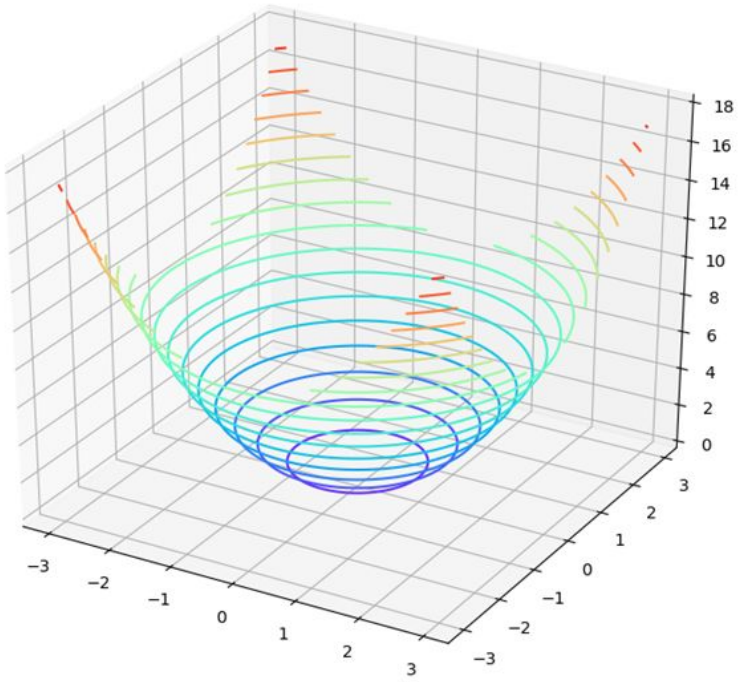
게임에서 캐릭터의 힘, 민첩, 지능, 체력, 마력, 정신력을 각각 하나의 변수로 하면 6개의 변수가 되고, 이 6개의 변수에 따라 캐릭터가 가지는 특성이 정해지게 됩니다.

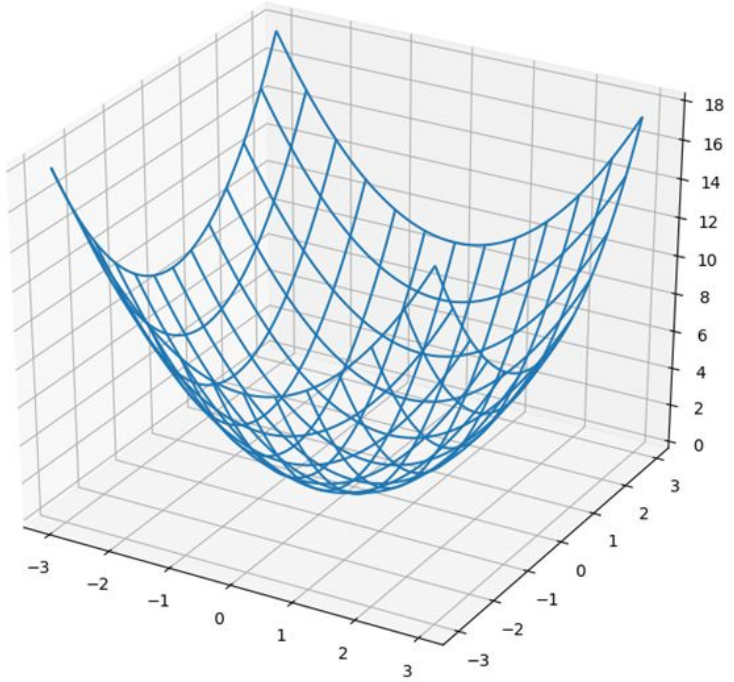


5. 2차원 함수의 그래프와 편미분

5. 2차원 함수의 그래프와 편미분

- $f(x_0, x_1) = x_0^2 + x_1^2$





5. 2차원 함수의 그래프와 편미분

편미분

이렇게 쓸 수 있습니다. $\partial f(x_0, x_1) / \partial x_0$는 x_0를 변수로 취급하면서 x_1은 상수로 취급합니다. 즉, $x_0^2 + x_1^2$을 미분할 때 x_1^2은 상수로 취급되기 때문에 x_1^2의 미분값은 0이 됩니다. 남는 것은 x_0^2 이고, x_0^2을 미분하면 $2x_0$가 됩니다.

$$f(x_0, x_1) = x_0^2 + x_1^2$$

$$\frac{\partial f(x_0, x_1)}{\partial x_0} = 2x_0$$

$$\frac{\partial f(x_0, x_1)}{\partial x_1} = 2x_1$$

6. 편미분 프로그래밍 코드

6. 편미분 프로그래밍 코드

변수가 2개인 함수의 미분, 편미분

f(x0, x1) = $x0^2$ + $x1^2$이고, x0, x1이 각각 1과 3일 때 편미분 값을 수치미분 방식으로 구해보겠습니다. 상미분 때에는 변수가 하나였지만 편미분은 변수를 2개 이상 사용합니다. 여기서는 x0, x1의 2개 입니다.

6. 편미분 프로그래밍 코드

편미분 파이썬 코드

CODE

```
h = 1e-5
def function_rx(x, y):
    return x**2 + y**2
def ndiff2(f, x, y):
    rf_rx = (f(x+h, y)-f(x-h, y))/(2*h)
    rf_ry = (f(x, y+h)-f(x, y-h))/(2*h)
    return [rf_rx, rf_ry]

ndiff2(function_rx, 1, 3)
```

7. numpy nditer

7. numpy nditer

7.1 for 반복문

CODE

```
market = ['apple', 'strawberry', 'grape']

for fruit in market:
    print(fruit)
```

7. numpy nditer

7.2 for를 이용한 이중 반복문

CODE

```
market = np.array([['apple', 'strawberry', 'grape'], ['water', 'cola', 'orangejuice']])
for category in market:
    for item in category:
        print(item)
```

7. numpy nditer

7.3 nditer를 이용한 행렬의 멀티인덱스 처리

CODE

```
market = np.array([['apple', 'strawberry', 'grape'],
                   ['water', 'cola', 'orangejuice']])

it = np.nditer(market, flags=['multi_index'], op_flags=['readwrite'])
while not it.finished:
    idx = it.multi_index
    print(market[idx])

    it.iternext()
```

7. numpy nditer

nditer 사용법

```
it = np.nditer(market, flags=['multi_index'], op_flags=['readwrite'])
```

명령을 통해서 market이라는 행렬의 각 원소의 값과 인덱스를 가져올 수 있습니다. it은 nditer의 객체이고, it 안에 인덱스가 순서대로 들어갑니다. nditer의 객체인 it의 사용법은 아래 세 가지만 기억해 두세요.

- it.finished ------ 행렬의 끝이면 True를 아니면 False를 반환
- it.multi_index ------ 행렬의 인덱스를 반환
- it.iternext() ------ 현재의 인덱스에서 1 증가한 인덱스로 변환

8. 신경망 계산 과정에서의 미분 이해

8. 신경망 계산 과정에서의 미분 이해

- 노드

입력층 784개 ---- 은닉층 50개 ----- 출력층 10개

x(784) ----- h(50) ----- y(10)

여기서 x(784)는 상수입니다. 손으로 쓴 글씨의 가로 세로 픽셀값입니다. 이 값은 변하거나 수정되는 것이 아닙니다. 반면 h(50)의 값은 입력으로 들어온 상수 x와 변수 w0, b0에 따라서 결정됩니다. y(10)도 변수 w1과 b1과 h(50)에 따라서 결정됩니다.

8. 신경망 계산 과정에서의 미분 이해

- 편차 w0 와 바이어스 b0

입력과 은닉층 사이에 필요한 w0와 b0는

$$w0(784\ ,\ 50),\ b0(50)$$

입니다. w0는 총 39,200개의 변수를 가지고 있습니다. b0의 개수 50개를 더하면 w0와 b0는 총 39,250개의 변수로 이루어져 있습니다. 여기서 우리는 784개의 상수(x)와 39,250개의 변수(w0, b0)의 곱과 합으로 구성된 50개의 결과로 h가 만들어지는 것을 알 수 있습니다.

8. 신경망 계산 과정에서의 미분 이해

연산 순서

```
v = affine() ------- (1) w0, b0를 이용 [변수의 개수 39,250개]
h = relu(v) -------- (2) v를 h로 변환
y = affine() ------- (3) w1, b1을 이용 [변수의 개수 510개]
y = softmax(y) ----- (4) 확률값으로 변환
y = loss(y,t) ------ (5) 오차 계산
```

9. 네트워크 변수의 편미분값인 기울기

9. 네트워크 변수의 편미분값인 기울기

파라미터 크기

입력층(X)의 크기 : A
은닉층(H)의 크기 : B
출력층(Y)의 크기 : C

입력층과 은닉층 사이 w0 크기 : A×B
입력층과 은닉층 사이 b0 크기 : B

은닉층과 출력층 사이 w1 크기 : B×C
은닉층과 출력층 사이 b1 크기 : C

네트워크변수의 크기 : A×B + B + B×C + C

9. 네트워크 변수의 편미분값인 기울기

기울기

네트워크변수에서 w0의 첫 번째 변수인 w0(0,0) 변수 하나를 w0(0,0)-0.0001로 대체해서 앞 장의 신경망 계산 과정(1)에서 (5)까지를 순차적으로 진행한 후 오찻값을 계산합니다. 오찻값을 f1에 저장한 다음 w0(0,0)을 다시 w0(0,0)+0.0001로 대체하여 오찻값을 계산합니다. 이때의 오찻값을 f2에 저장합니다. (f2-f1)/(2×0.0001)을 계산하면 바로 미분값이 나옵니다.

$$grad = \frac{f2 - f1}{2 \times 0.0001}$$

9. 네트워크 변수의 편미분값인 기울기

- 기울기

변숫값 = 변숫값 - 기울기 × 변숫값 × 학습률

```
w0(0,1) = w0(0,1) - grad.w(0,1)*w0(0,1)*lr
        = 2.381 - 0.4*2.381*0.01
```

10. nditer 편미분 코드

10. nditer 편미분 코드

출력의 비중을 조정

```
import numpy as np

def numerical_diff(f, x):
    h = 1e-4    # 0.0001
    nd_coef = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        index = it.multi_index
        tmp = x[index]
```

10. nditer 편미분 코드

- 출력의 비중을 조정

```
        x[index] = tmp + h
        fxh2 = f()    # f(x+h)
        x[index] = tmp - h
        fxh1 = f()    # f(x-h)
        nd_coef[index] = (fxh2 - fxh1) / (2*h)
        x[index] = tmp
        it.iternext()

    return nd_coef
```

11. 경사하강법

11. 경사하강법

손실함수와 경사하강법

미분의 기울기를 이용해서 손실함수의 최솟값을 향해 변수를 변화시키는 것이 바로 '경사하강법'입니다. 마치 계곡의 경사를 따라 아래로 내려가면 산에서 마을로 갈 수 있는 것처럼, 정확한 산의 지리를 잘 모르더라도 낮은 곳을 향해 내려가면 가장 낮은 곳에 위치한 강이나 마을을 찾을 수 있습니다. η는 그리스 알파벳 소문자로 '에타(eta)'라고 읽고 학습률(learning rate)을 의미합니다.

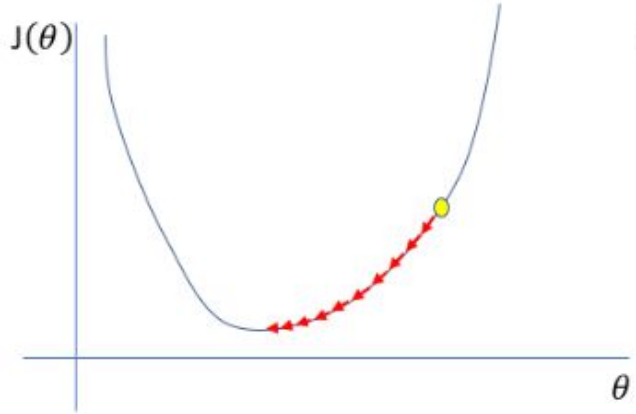
$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

11. 경사하강법

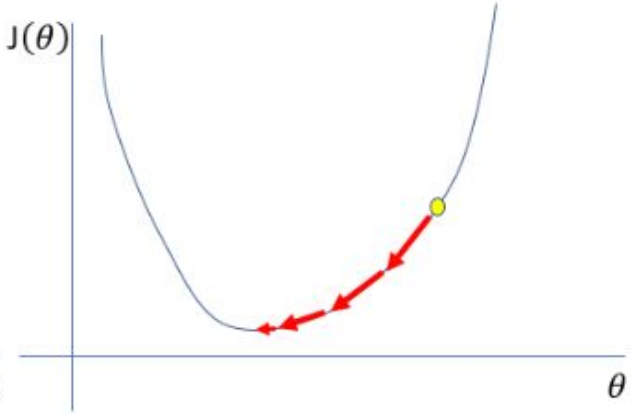
학습률

Too low



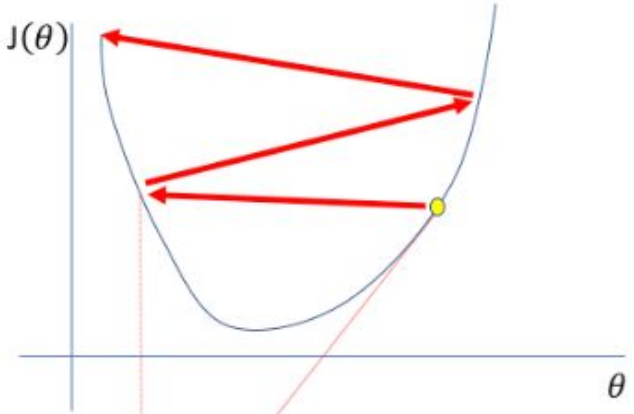
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors