PyTorch Tutorial

03. Gradient Descent

Revision

- What would be the best model for the data?
- Linear model?

x (hours)	y (points)
1	2
2	4
3	6
4	?

Linear Model

$$\hat{y} = x * \omega$$

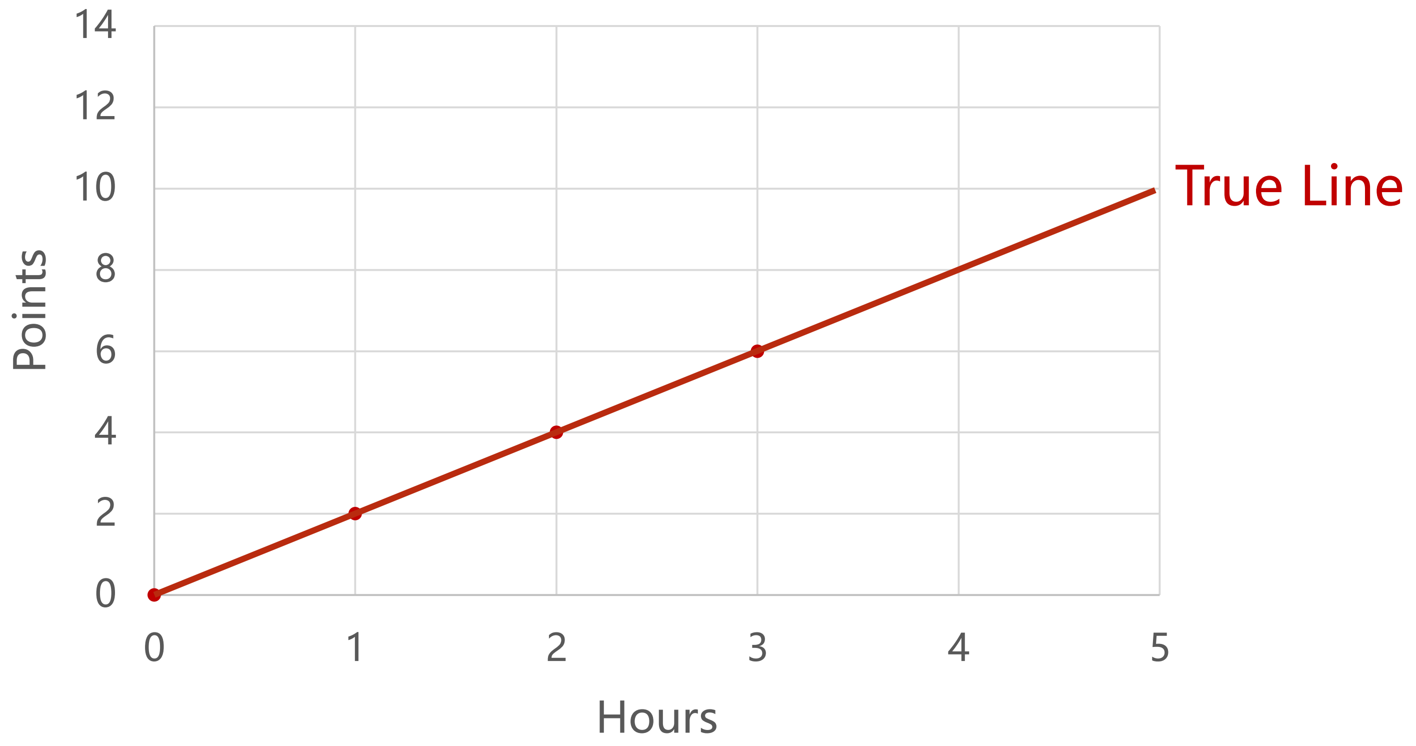
To simplify the model

Revision

Linear Model

$$\hat{y} = x * \omega$$

x (hours)	y (points)
1	2
2	4
3	6

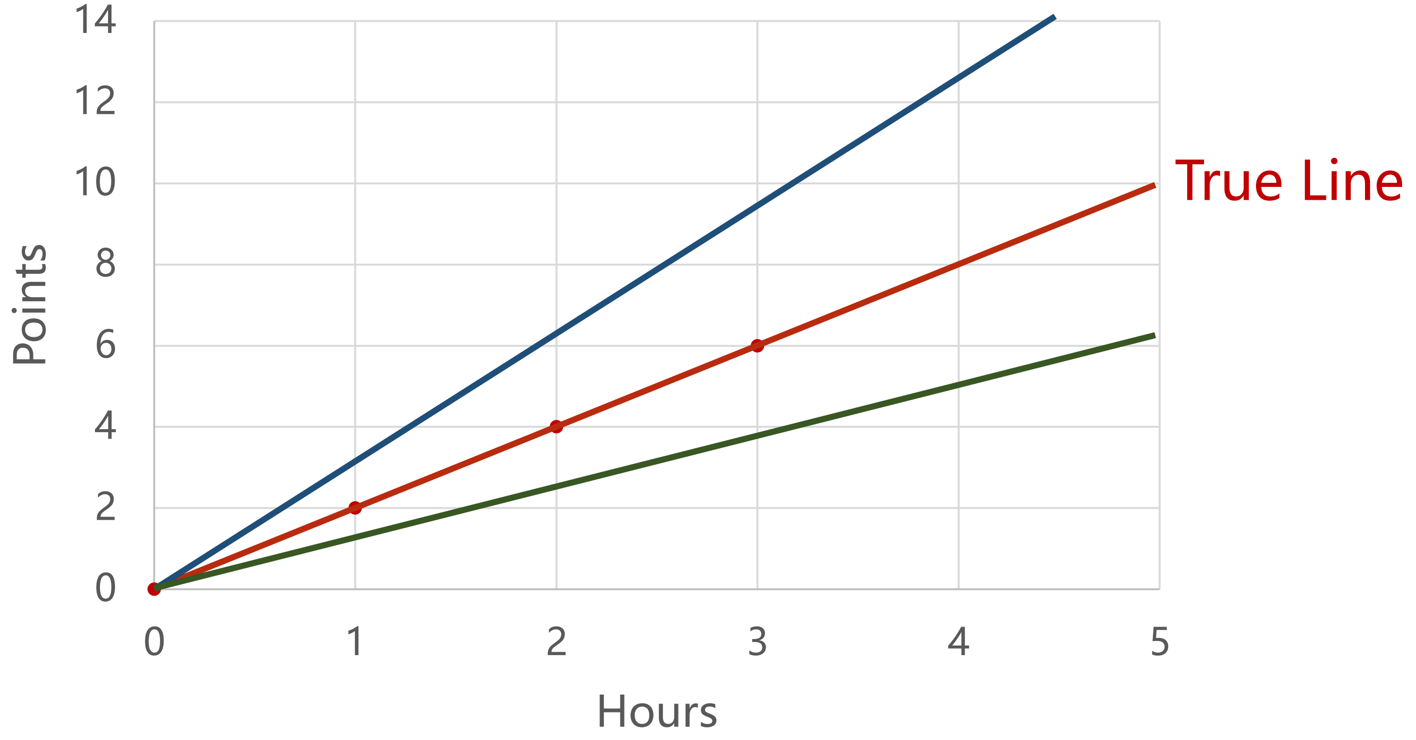


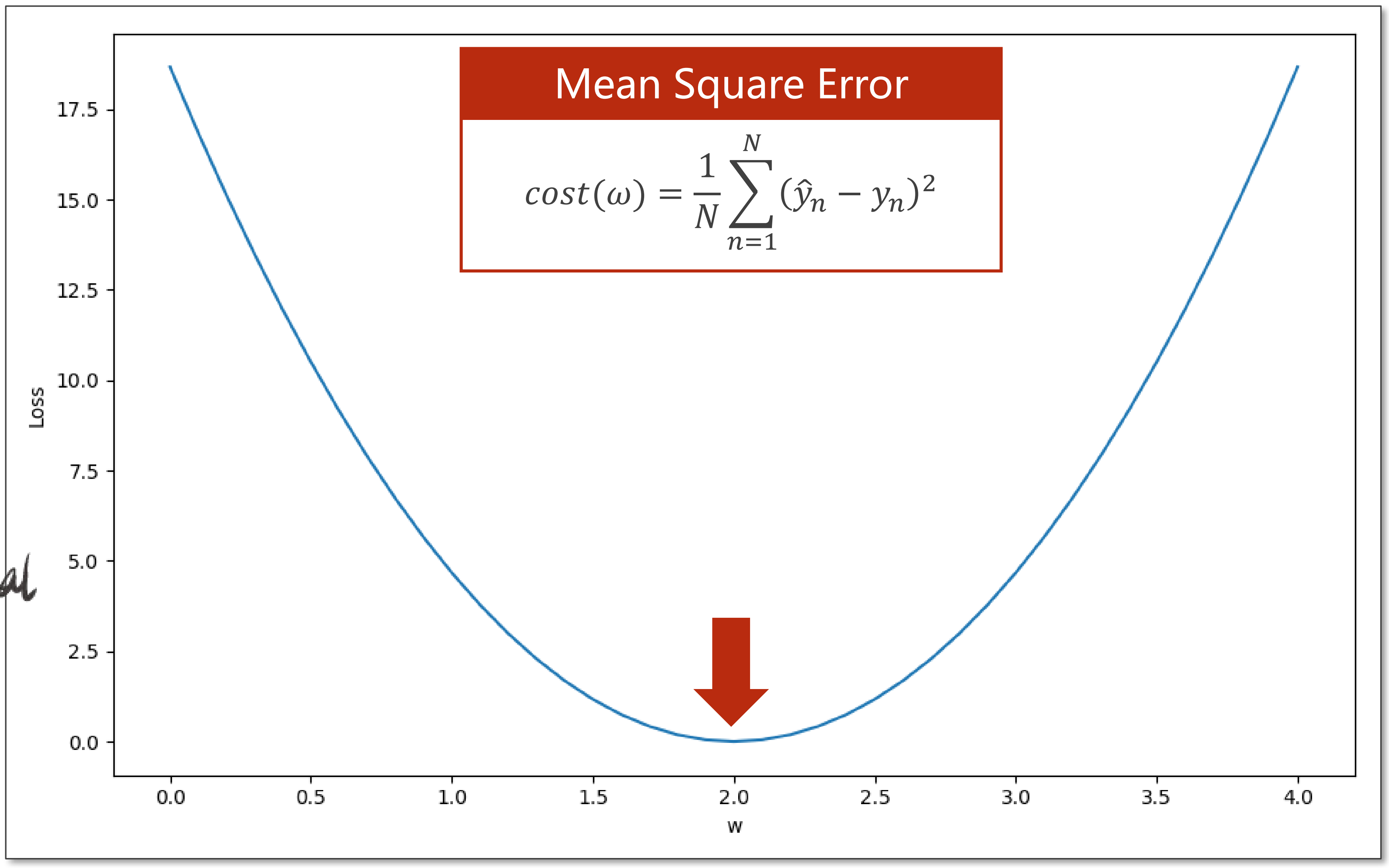
Linear Model

$$\hat{y} = x * \omega$$

x (hours)	y (points)
1	2
2	4
3	6

The machine starts with **a random guess**, ω = random value



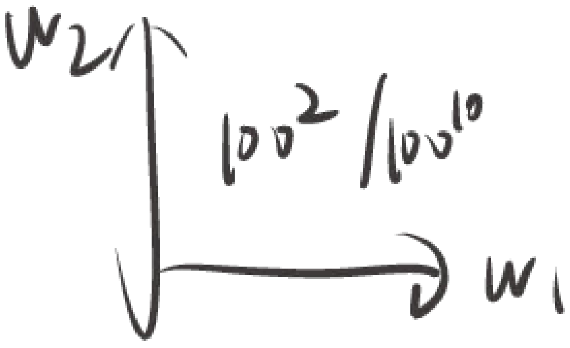


It can be found that when $\omega = 2$, the cost will be minimal.

will be minimal.

肉眼直接观察到minimal

一、穷举法



使用穷举法列举所有可能的权重 w_i

↓

求对应的目标损失函数 $loss_i$

↓

看哪一种权重组合的损失值最小

- 仅一个权重 w_1 → 100个数据，搜索量为100
- 两个权重 w_1, w_2 → $100 \cdots\cdots 100^2$
- 十个权重 $w_1, w_2 \ldots w_{10}$ → $100 \cdots\cdots 100^{10}$

二、分治法

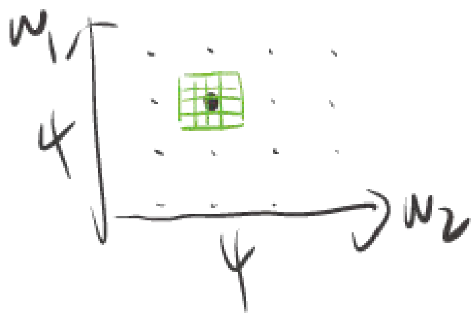
① 若搜索区间=100，不直接分为100份 (100×100=10000)

② 可先分为4份 (4×4=16，可选取16个点中的最低值)

再以16个点中的最低值为区间范围，继续划分为4份 (4×4=16)

两轮一共搜索 16+16=32

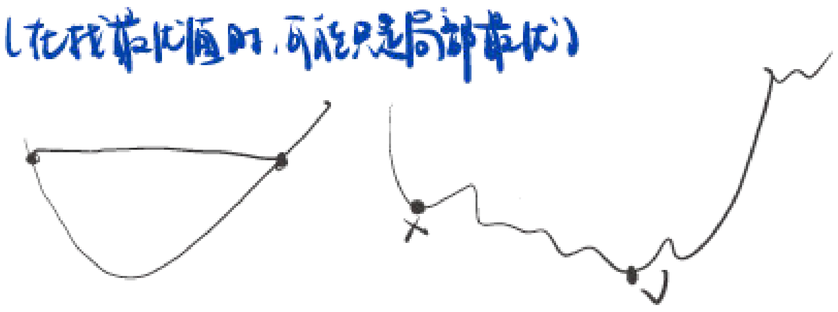
此时是在区间内搜索，否则搜索量=16×16=256



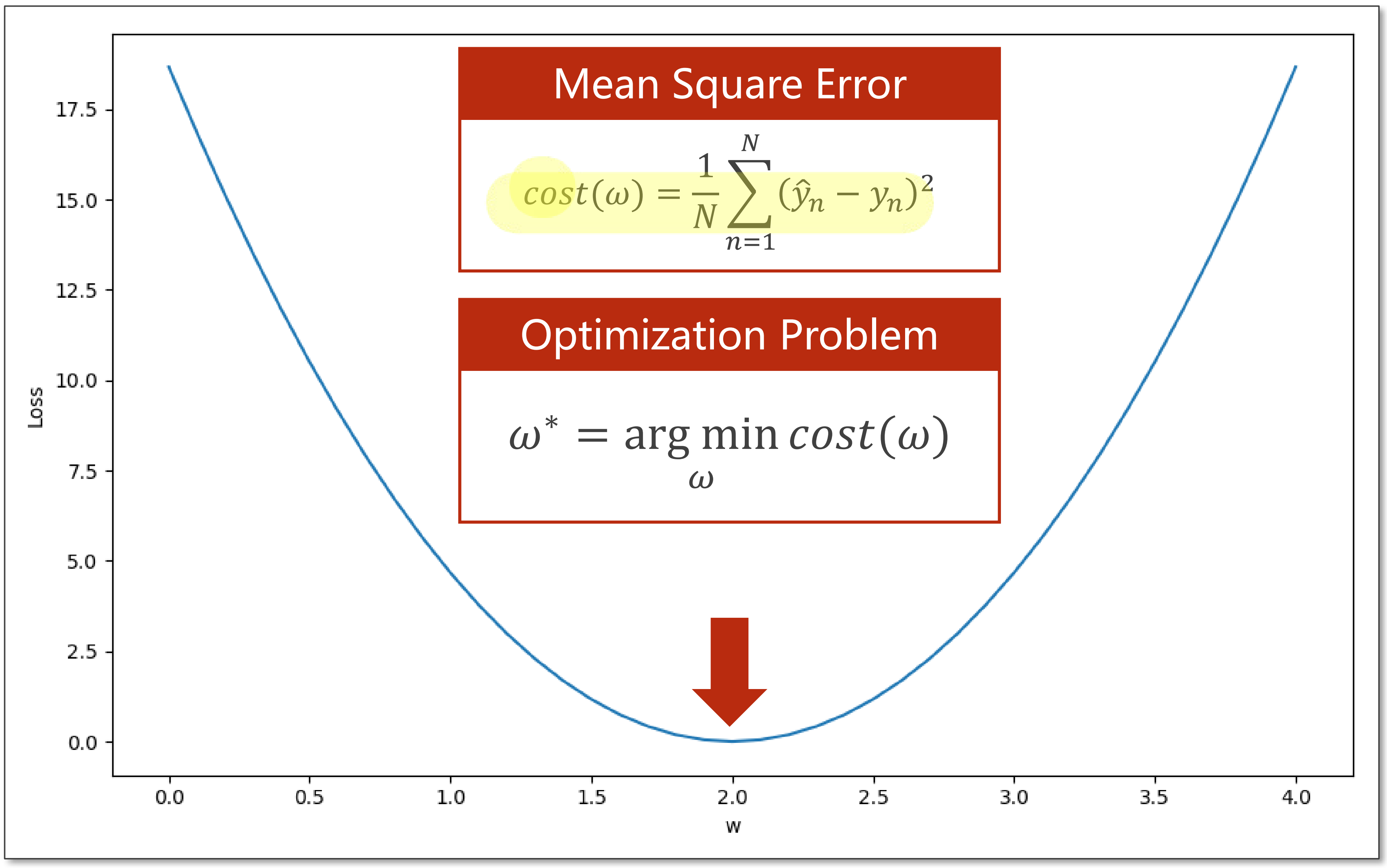
③ 分治算法可减小搜索量

- 有一定优势，
- 但我们的损失函数不一定就是平滑的凸函数

(在找最优值时，可能只是局部最优)



Optimization Problem

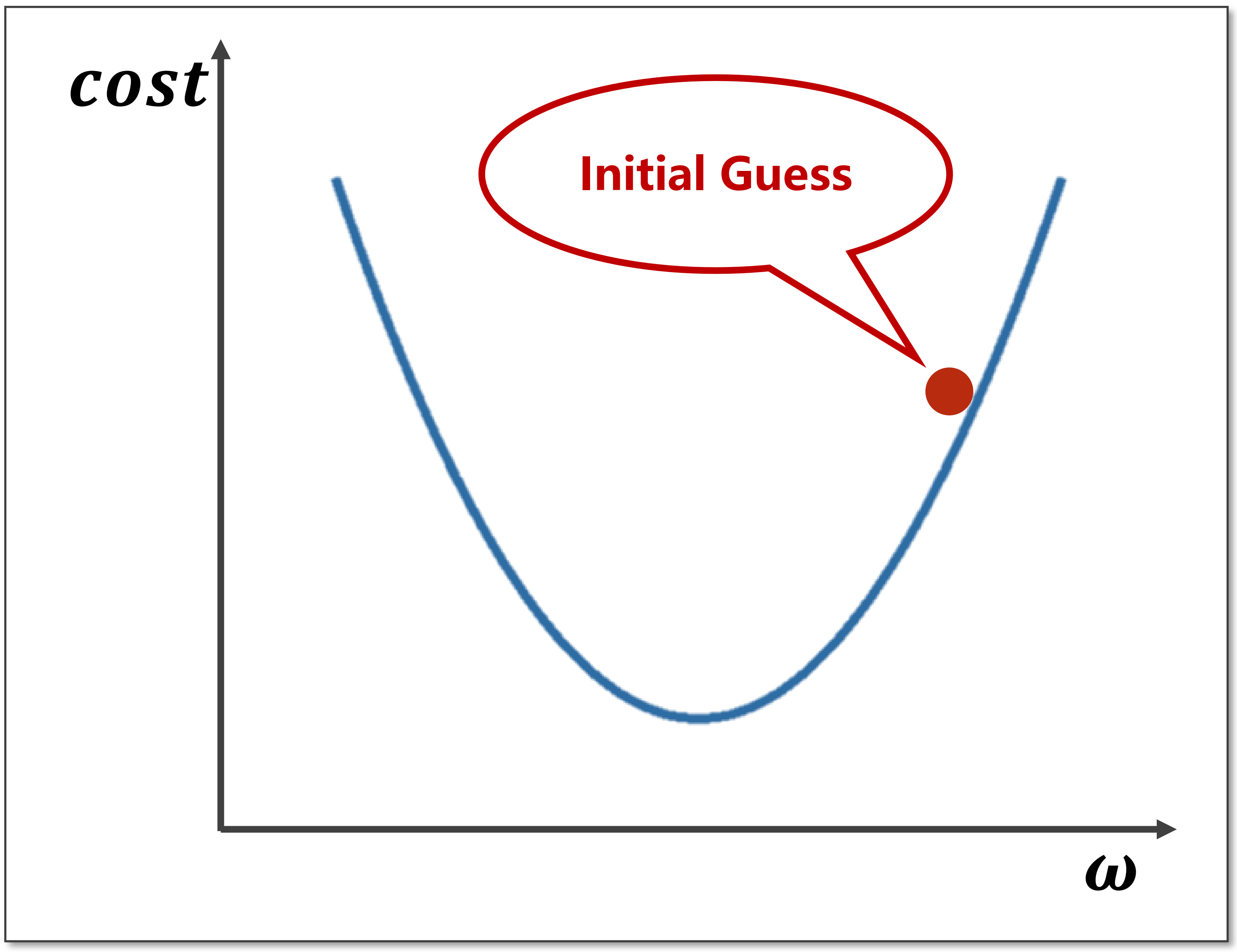


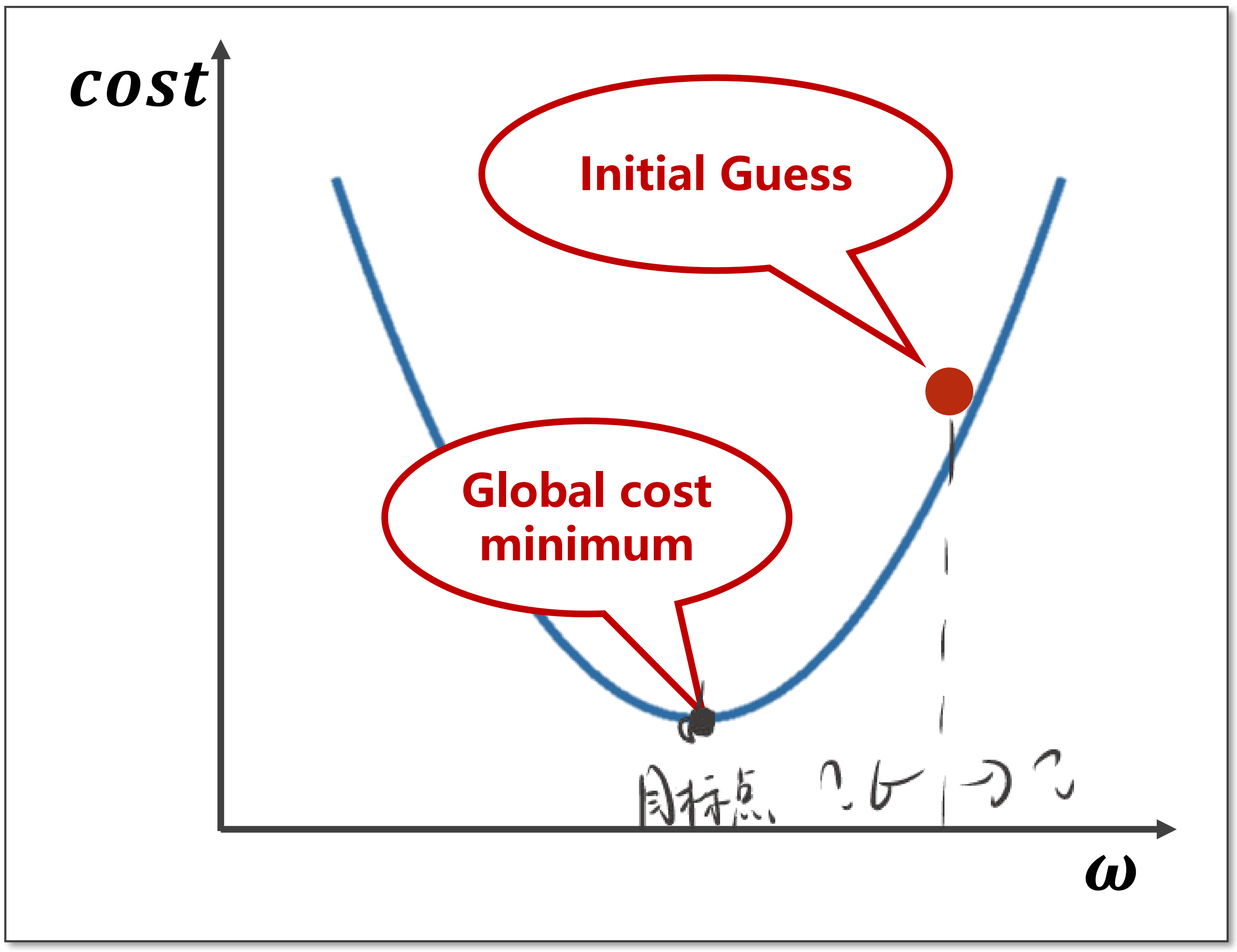
Mean Square Error

$$cost(\omega) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

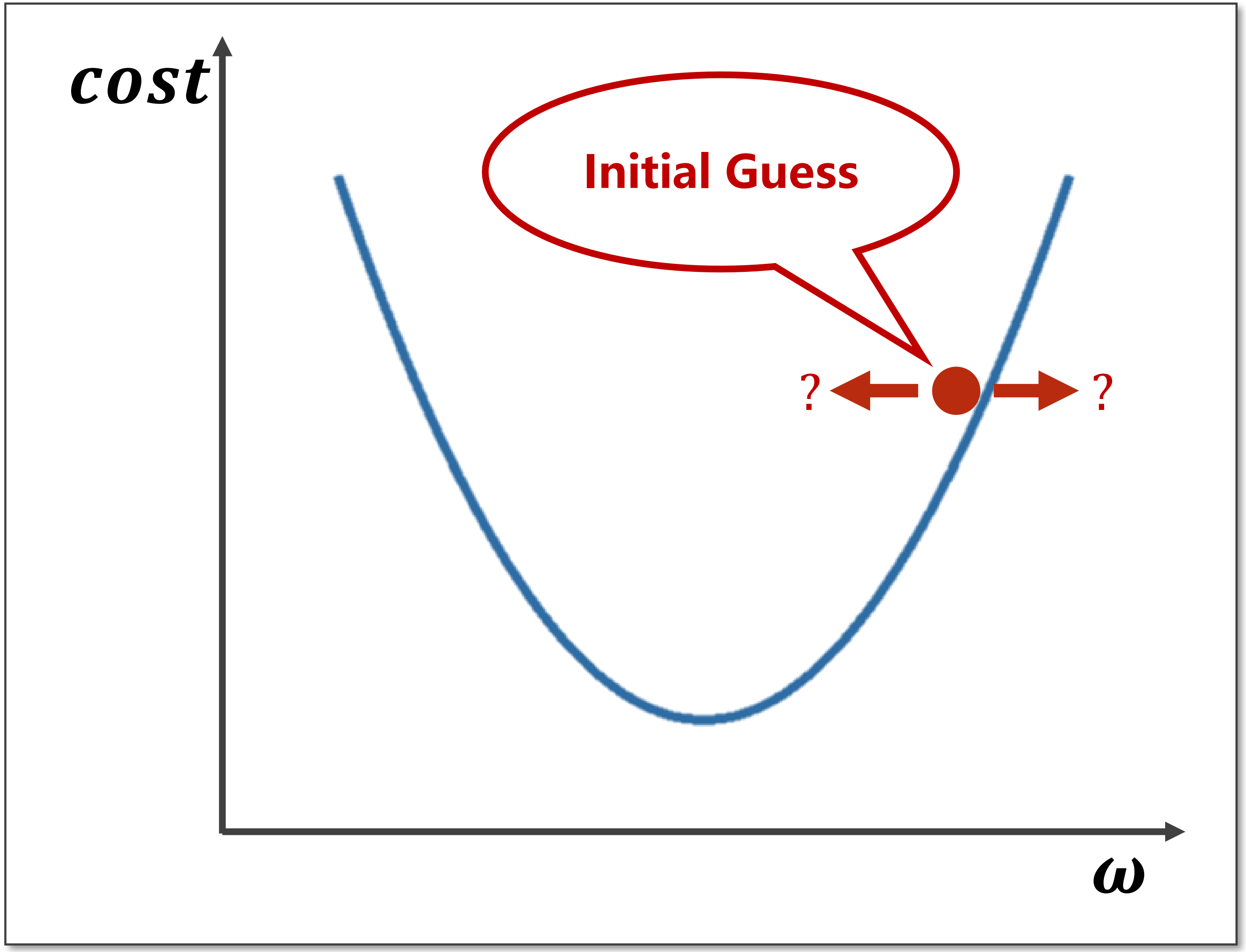
Optimization Problem

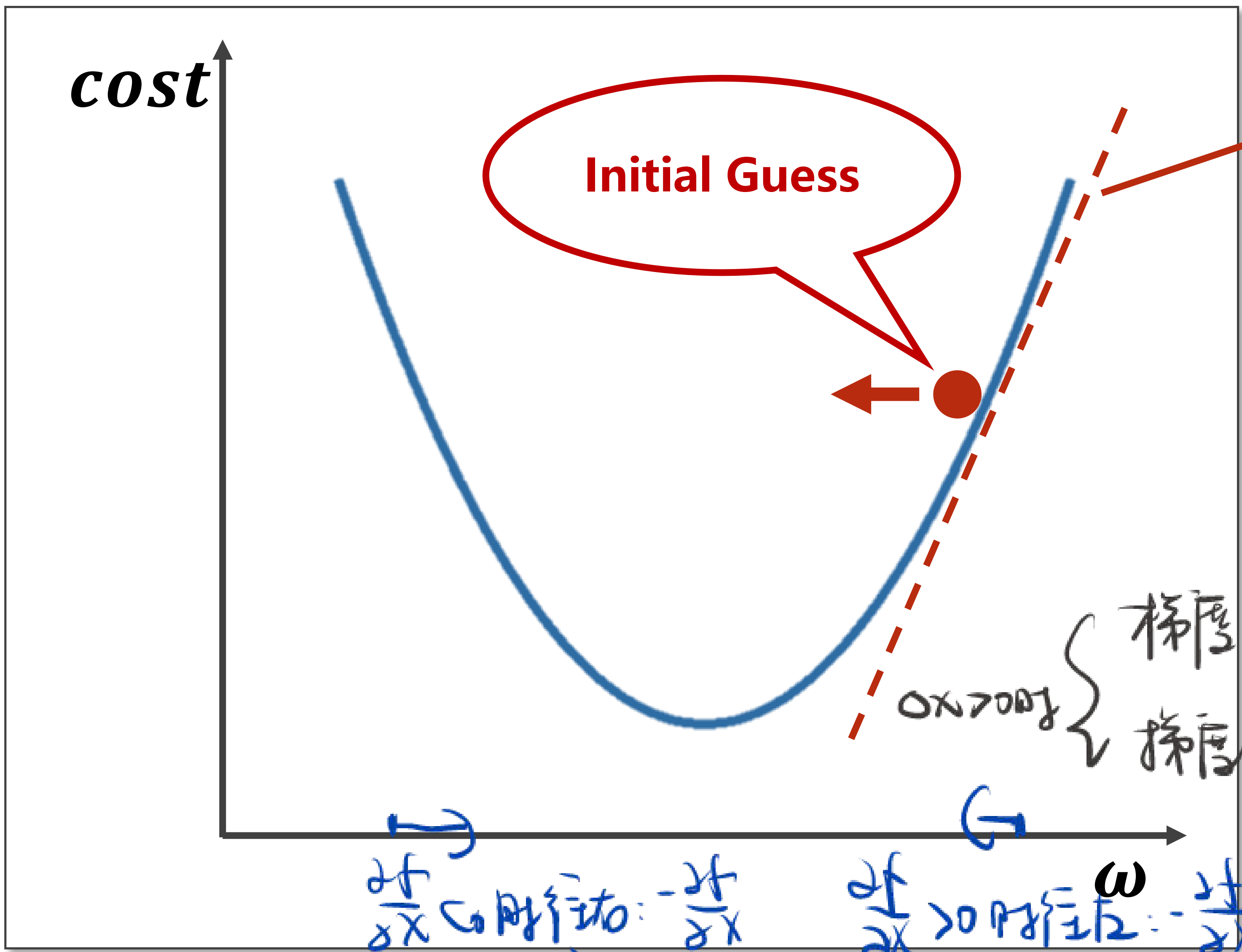
$$\omega^* = \underset{\omega}{\arg\min}\, cost(\omega)$$


cost
Initial Guess
ω


cost
Initial Guess
Global cost minimum
目标点
2.6
2
ω

Gradient Descent Algorithm





∂f/∂x < 0 时往右：$-\frac{\partial f}{\partial x} > 0$

∂f/∂x > 0 时往左：$-\frac{\partial f}{\partial x} < 0$

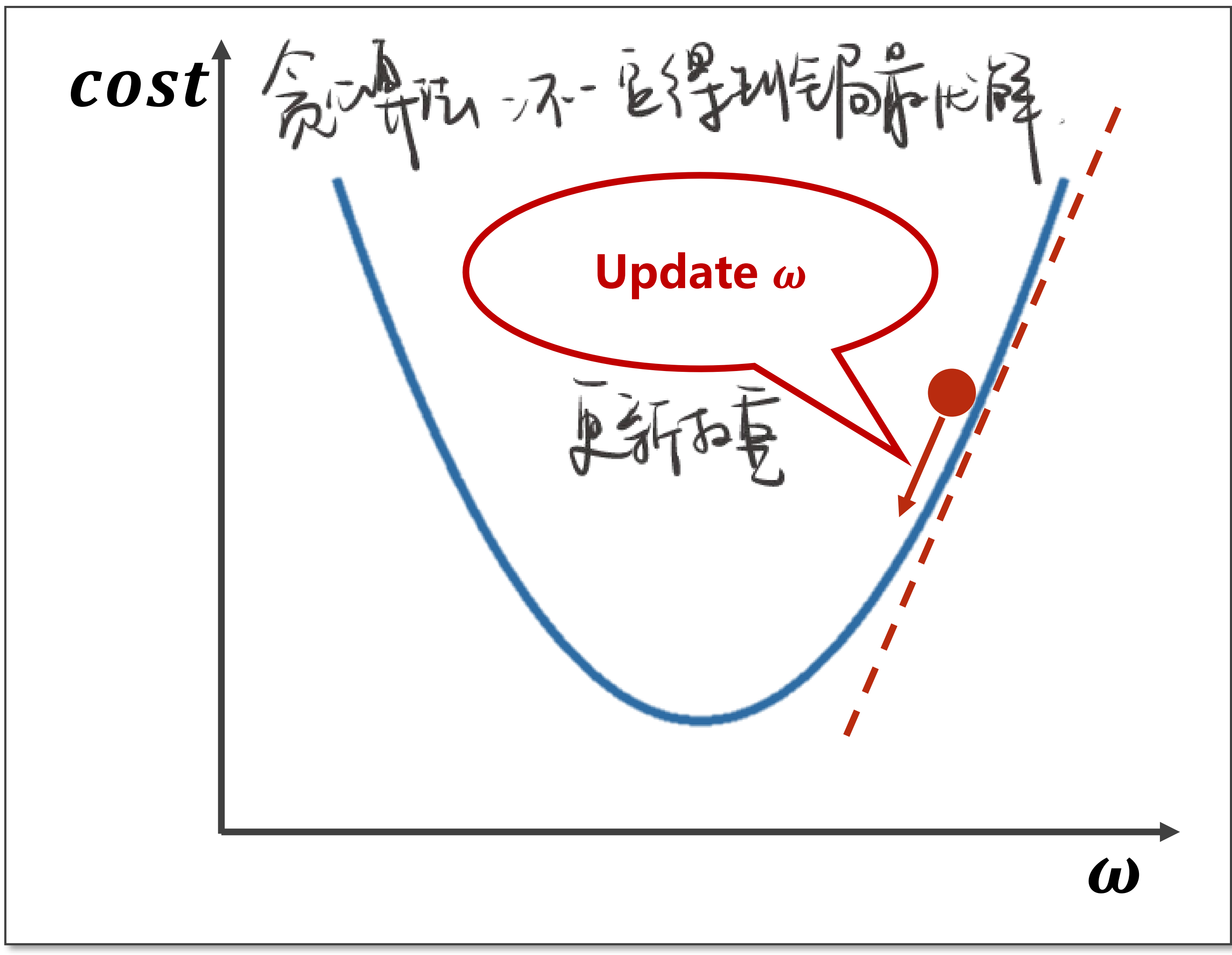
Gradient

$$\frac{\partial cost}{\partial \omega}$$

梯度：损失函数对权重的导数 = 该点的切线斜率

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Δx>0 时
- 梯度 $\frac{\partial f}{\partial x} > 0$ ⇒ 正方向 → 上升 ⇒ 负方向 $-\frac{\partial f}{\partial x}$ ↓
- 梯度 $\frac{\partial f}{\partial x} < 0$ ⇒ 负方向 ← 上升 ⇒ 正方向 $-\frac{\partial f}{\partial x}$ ↓



Gradient

$$\frac{\partial cost}{\partial \omega}$$

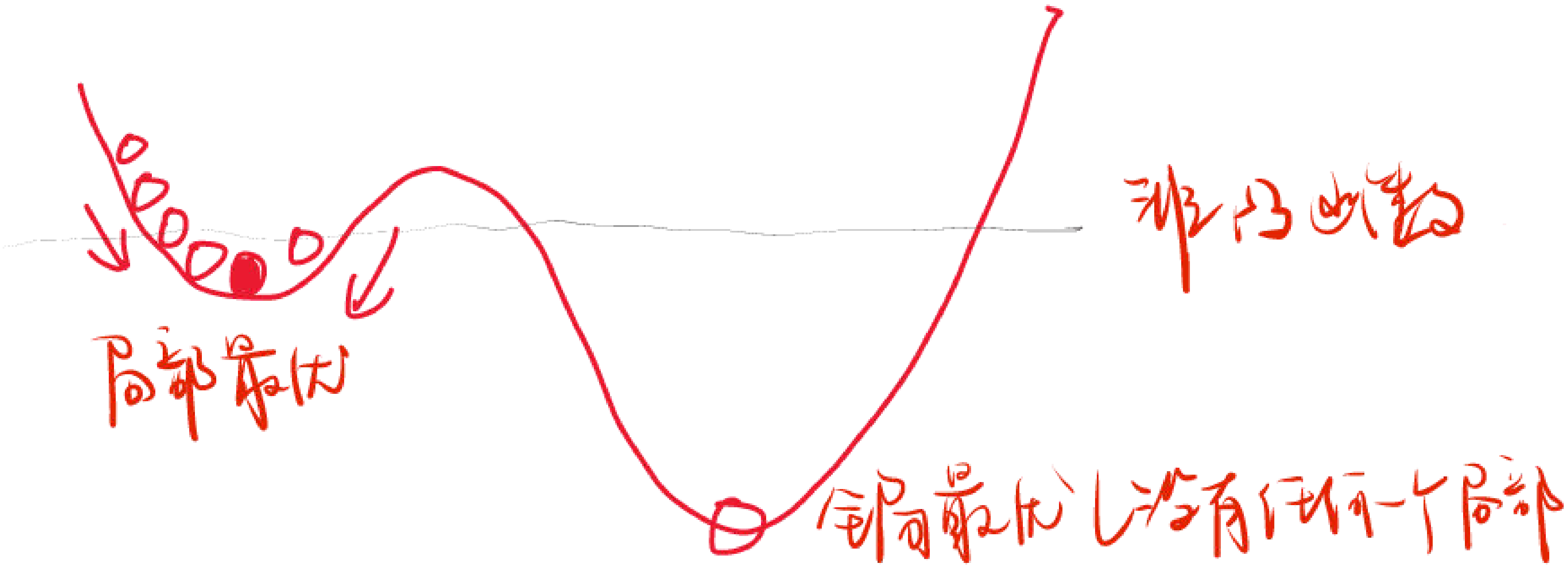
Update

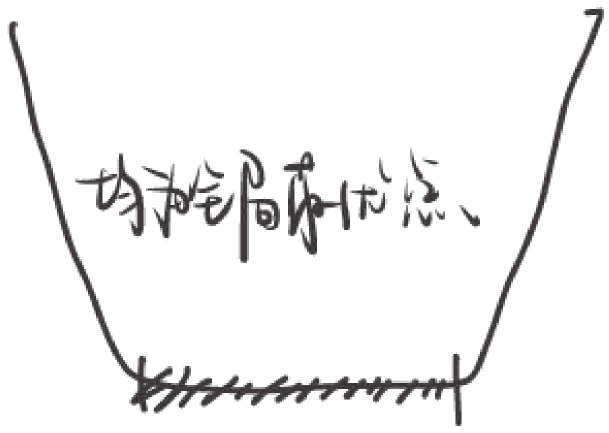
$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

α 学习率：表示往前走多远
一般要取得小一点，否则很难收敛

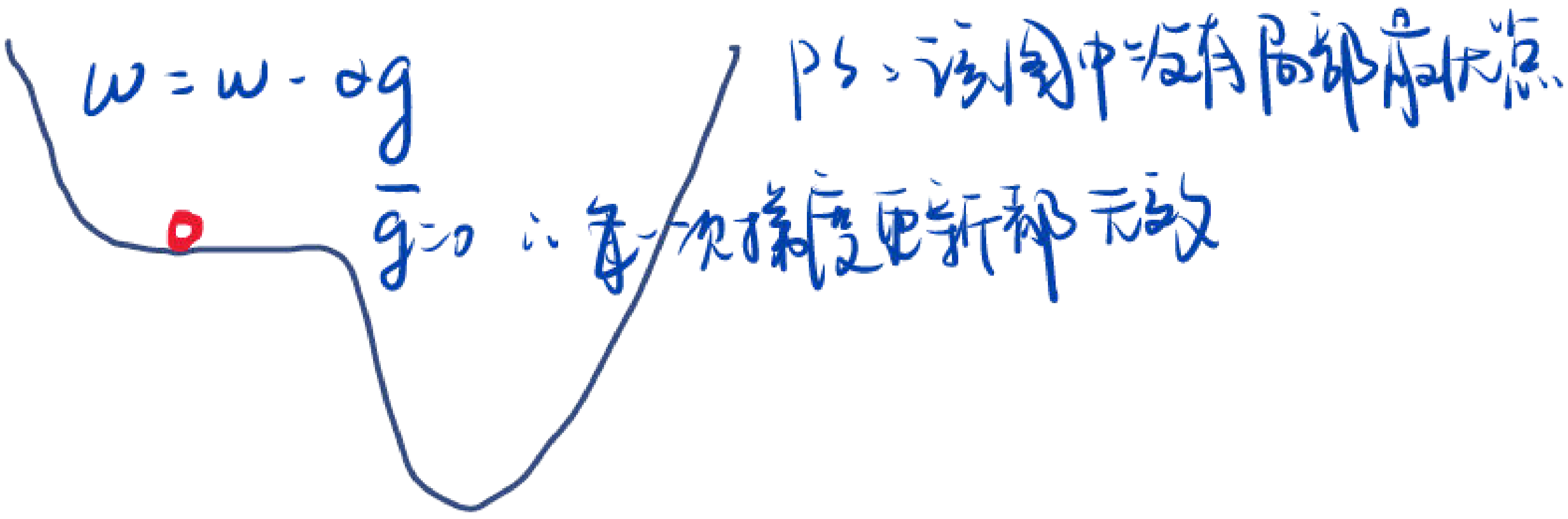
梯度下降：很难找到全局最优解，但深度学习中会大量使用梯度下降作为最基本的算法

以前大家认为是会：陷入局部最优点
但实际上深度神经网络中，损失函数并没有很多局部最优点

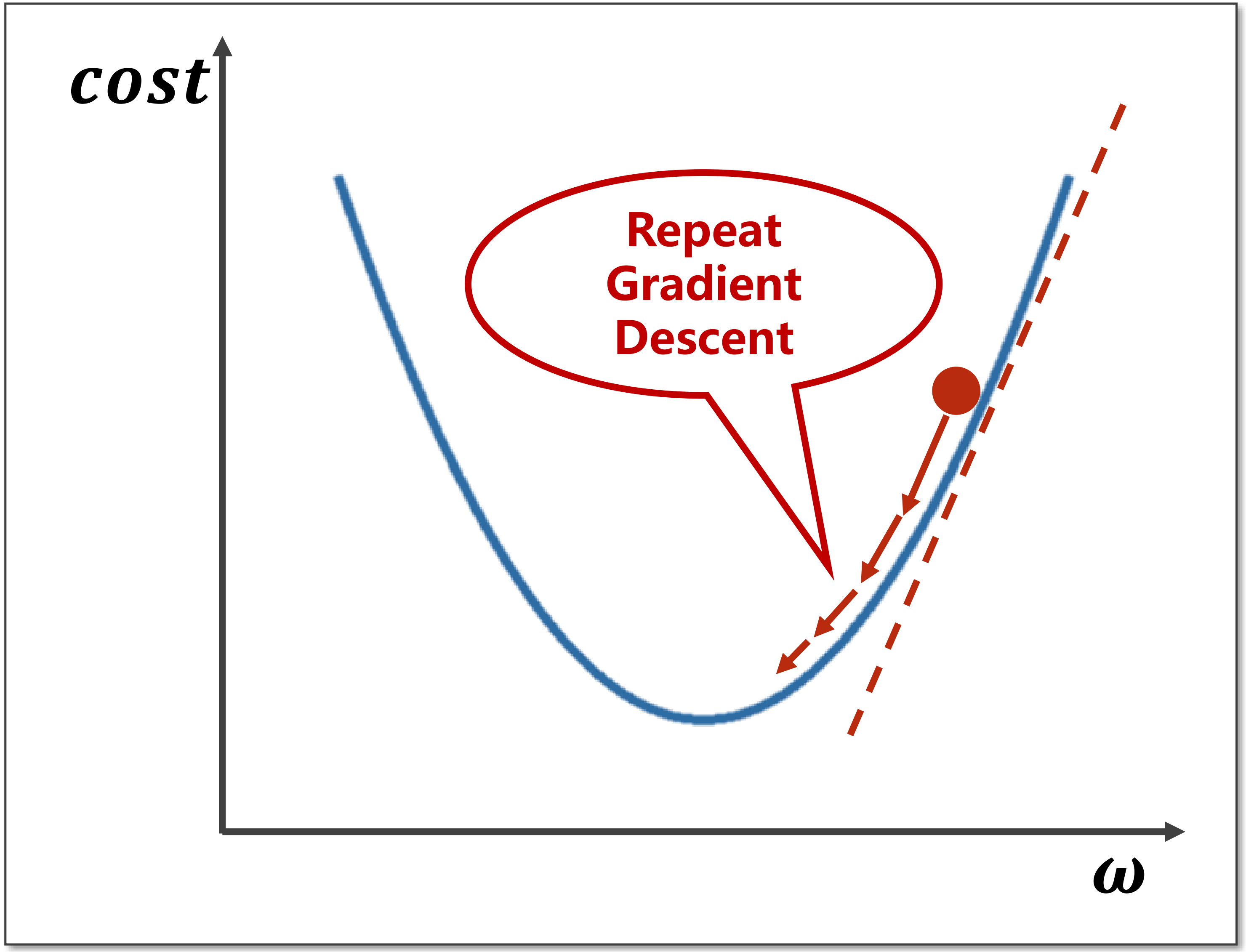




所以主要的问题是鞍点，无法继续迭代（梯度为0）



Gradient Descent Algorithm



Gradient

$$\frac{\partial cost}{\partial \omega}$$

Update

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

Gradient Descent Algorithm

Derivative

$\hat{y} = wx$ $\quad y$

$$\frac{\partial cost(\omega)}{\partial \omega} = \frac{\partial}{\partial \omega} \frac{1}{N} \sum_{n=1}^{N} (x_n \cdot \omega - y_n)^2$$

和的倒数=倒数的和

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

提公因式和以$\frac{1}{N}$

$$= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial \omega} \underbrace{(x_n \cdot \omega - y_n)^2}_{z}$$

链式求导法则

$$= \frac{1}{N} \sum_{n=1}^{N} 2 \cdot (x_n \cdot \omega - y_n) \frac{\partial (x_n \cdot \omega - y_n)}{\partial \omega}$$

$\frac{\partial z^2}{\partial w} = 2z \cdot \frac{\partial z}{\partial w}$

$$= \frac{1}{N} \sum_{n=1}^{N} 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Gradient

$$\frac{\partial cost}{\partial \omega}$$

Update

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

Derivative

$$\frac{\partial cost(\omega)}{\partial \omega} = \frac{\partial}{\partial \omega} \frac{1}{N} \sum_{n=1}^{N} (x_n \cdot \omega - y_n)^2$$

$$= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial \omega} (x_n \cdot \omega - y_n)^2$$

$$= \frac{1}{N} \sum_{n=1}^{N} 2 \cdot (x_n \cdot \omega - y_n) \frac{\partial (x_n \cdot \omega - y_n)}{\partial \omega}$$

$$= \frac{1}{N} \sum_{n=1}^{N} 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Gradient

$$\frac{\partial cost}{\partial \omega}$$

Update

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

Update

$$\omega = \omega - \alpha \frac{1}{N} \sum_{n=1}^{N} 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Implementation

```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

x

Y

```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
```

Prepare the training set.

准备训练集

Implementation

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

```python
w = 1.0
```

Initial guess of weight.

初始猜测的权重

Implementation

```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

```
def forward(x):
    return x * w
```

Define the model:

Linear Model

$$\hat{y} = x * \omega$$ 前馈模型

Implementation

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

```python
def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)
```

$\hat{y}$ $(\hat{y}-y)^2$

Define the cost function

Mean Square Error

$$cost(\omega) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

平均损失值 MSE

Implementation

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```


```python
def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)
```


Define the gradient function

Gradient

$$\frac{\partial cost}{\partial \omega} = \frac{1}{N}\sum_{n=1}^{N} 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$ 梯度

Implementation

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

∂cost/∂w

```python
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
```

Do the update

Update

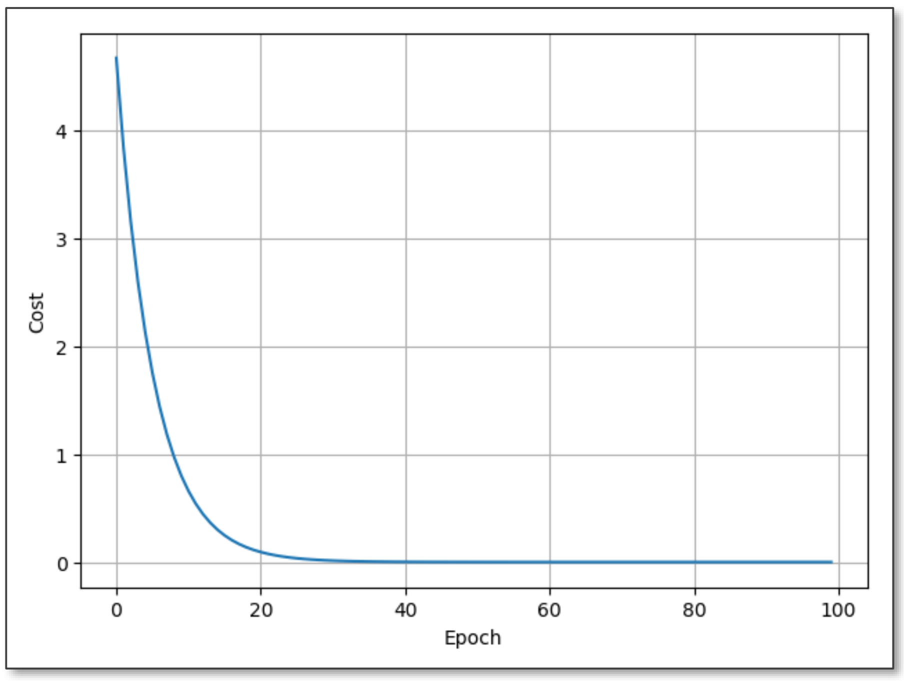
$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

训练过程即更新权重

Implementation

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_pred = forward(x)
        cost += (y_pred - y) ** 2
    return cost / len(xs)

def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, ys):
        grad += 2 * x * (x * w - y)
    return grad / len(xs)

print('Predict (before training)', 4, forward(4))
for epoch in range(100):
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= 0.01 * grad_val
    print('Epoch:', epoch, 'w=', w, 'loss=', cost_val)
print('Predict (after training)', 4, forward(4))
```

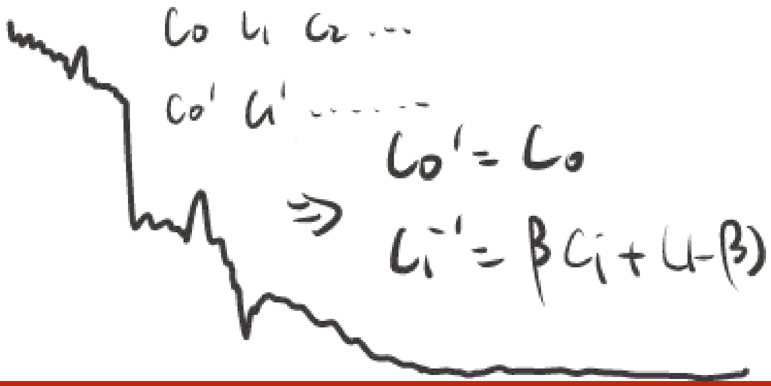
```
Predict (before training) 4 4.0
Epoch:  0 w= 1.09 cost= 4.67
Epoch:  1 w= 1.18 cost= 3.84
Epoch:  2 w= 1.25 cost= 3.15
Epoch:  3 w= 1.32 cost= 2.59
Epoch:  4 w= 1.39 cost= 2.13
Epoch:  5 w= 1.44 cost= 1.75
Epoch:  6 w= 1.50 cost= 1.44
Epoch:  7 w= 1.54 cost= 1.18
Epoch:  8 w= 1.59 cost= 0.97
Epoch:  9 w= 1.62 cost= 0.80
Epoch: 10 w= 1.66 cost= 0.66
...........
Epoch: 90 w= 2.00 cost= 0.00
Epoch: 91 w= 2.00 cost= 0.00
Epoch: 92 w= 2.00 cost= 0.00
Epoch: 93 w= 2.00 cost= 0.00
Epoch: 94 w= 2.00 cost= 0.00
Epoch: 95 w= 2.00 cost= 0.00
Epoch: 96 w= 2.00 cost= 0.00
Epoch: 97 w= 2.00 cost= 0.00
Epoch: 98 w= 2.00 cost= 0.00
Epoch: 99 w= 2.00 cost= 0.00
Predict (after training)  4  8.00
```



Cost in each epoch

$C_0 \quad C_1 \quad C_2 \cdots$

$C_0' \quad C_1' \cdots\cdots$



$$\Rightarrow \quad C_0' = C_0$$
$$C_i' = \beta C_i + (1-\beta) C_{i-1}'$$

指数加权均值：使曲线更平滑

训练发散：表示训练失败

可能鞍点太大

但不意味着局部最低点就是损失最小值。

Stochastic Gradient Descent

Gradient Descent

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

Derivative of Cost Function

$$\frac{\partial cost}{\partial \omega} = \frac{1}{N} \sum_{n=1}^{N} 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

所有损失的均值cost → 随机一个样本的损失值loss（对每个样本权重求导并更新）

因数据是有噪声的，引入一个随机噪声，使陷入鞍点的情况向前推动

Stochastic Gradient Descent

$$\omega = \omega - \alpha \frac{\partial loss}{\partial \omega}$$

Derivative of Loss Function

$$\frac{\partial loss_n}{\partial \omega} = 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) ** 2

def gradient(x, y):
    return 2 * x * (x * w - y)

print('Predict (before training)', 4, forward(4))

for epoch in range(100):
    for x, y in zip(x_data, y_data):
        grad = gradient(x, y)
        w = w - 0.01 * grad
        print("\tgrad: ", x, y, grad)
        l = loss(x, y)

    print("progress:", epoch, "w=", w, "loss=", l)

print('Predict (after training)', 4, forward(4))
```

cost → loss
求和 → 随机

```python
def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) ** 2
```

Calculate loss function:

Loss Function

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

Implementation of SGD

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) ** 2

def gradient(x, y):
    return 2 * x * (x * w - y)

print('Predict (before training)', 4, forward(4))

for epoch in range(100):
   for x, y in zip(x_data, y_data):
       grad = gradient(x, y)
       w = w - 0.01 * grad
       print("\tgrad: ", x, y, grad)
       l = loss(x, y)

   print("progress:", epoch, "w=", w, "loss=", l)

print('Predict (after training)', 4, forward(4))
```

梯度

求和→随机

```python
def gradient(x, y):
    return 2 * x * (x * w - y)
```

Calculate loss function:

Derivative of Loss Function

$$\frac{\partial loss_n}{\partial \omega} = 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Implementation of SGD

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) ** 2

def gradient(x, y):
    return 2 * x * (x * w - y)

print('Predict (before training)', 4, forward(4))

for epoch in range(100):
    for x, y in zip(x_data, y_data):
        grad = gradient(x, y)
        w = w - 0.01 * grad
        print("\tgrad: ", x, y, grad)
        l = loss(x, y)

    print("progress:", epoch, "w=", w, "loss=", l)

print('Predict (after training)', 4, forward(4))
```

→ 对每个样本都分别计算梯度并更新


```python
for epoch in range(100):
    for x, y in zip(x_data, y_data):
        grad = gradient(x, y)
        w = w - 0.01 * grad
        print("\tgrad: ", x, y, grad)
        l = loss(x, y)
```


Update weight by every grad of sample of train set.

梯度下降：效率更高（并行计算 $f(x_i)$、$f(x_{i+1})$，两者无相互依赖关系）

随机梯度下降：性能更高，更能找到最优点（但随机梯度下降是相互依赖的，所以不可并行）

$$w \mathrel{-}= \alpha \cdot \frac{\partial g(x_i)}{\partial w}$$

$$\downarrow$$

$$w \mathrel{-}= \alpha \cdot \frac{\partial g(x_{i+1})}{\partial w}$$

中和

梯度下降：批量的梯度下降

Batch

mini-batch

	Batch	mini-batch	随机
性能	低		高✓
时间复杂度	低✓		高

PyTorch Tutorial

03. Gradient Descent