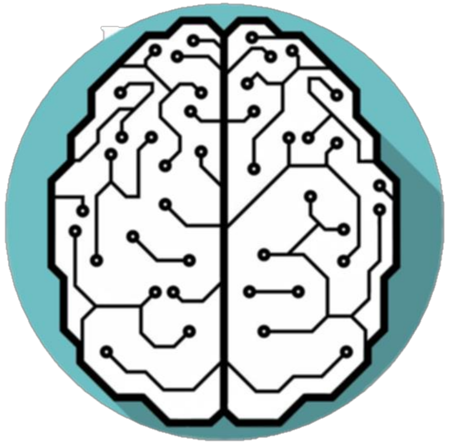
PyTorch Tutorial

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

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

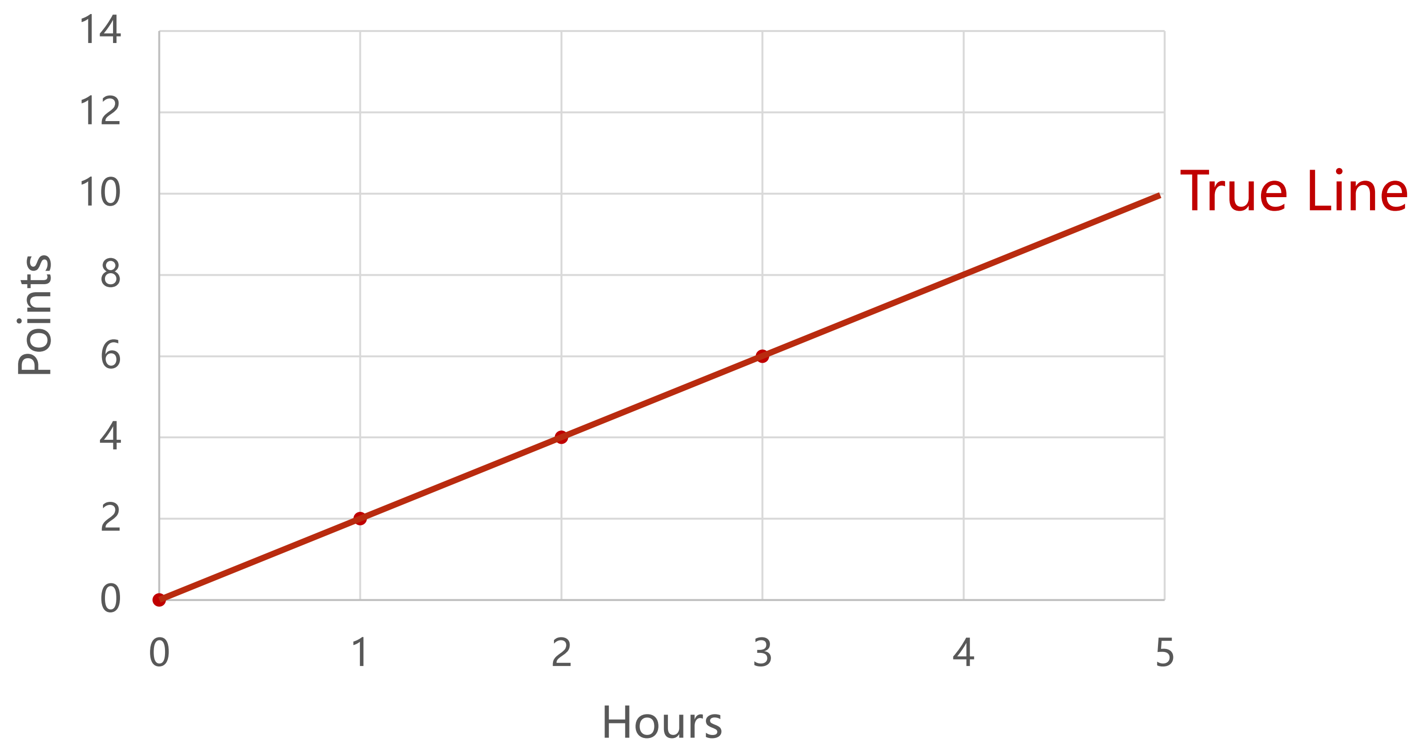
To simplify the model

Revision

Linear Model

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

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

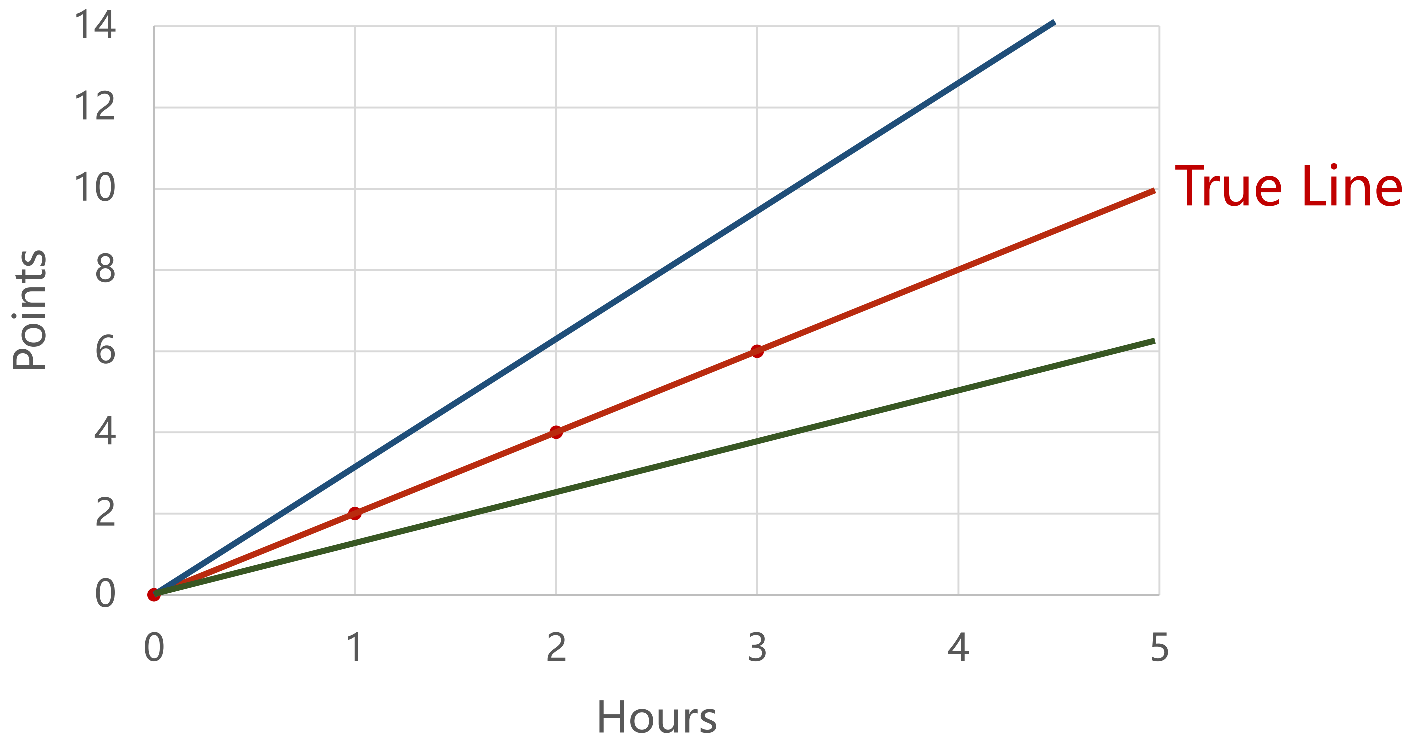


Linear Model

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

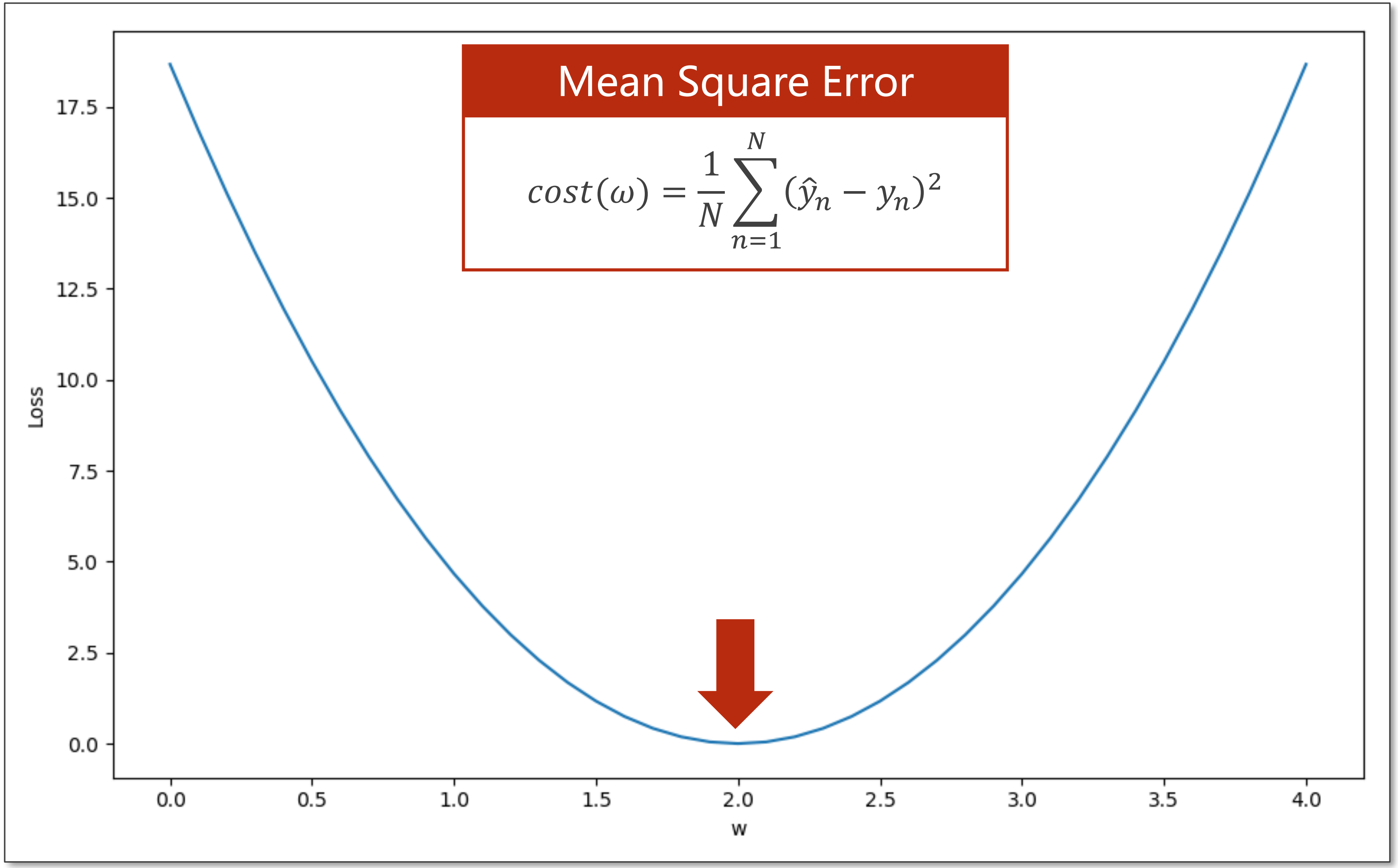
x (hours)	y (points)
1	2
2	4
3	6

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

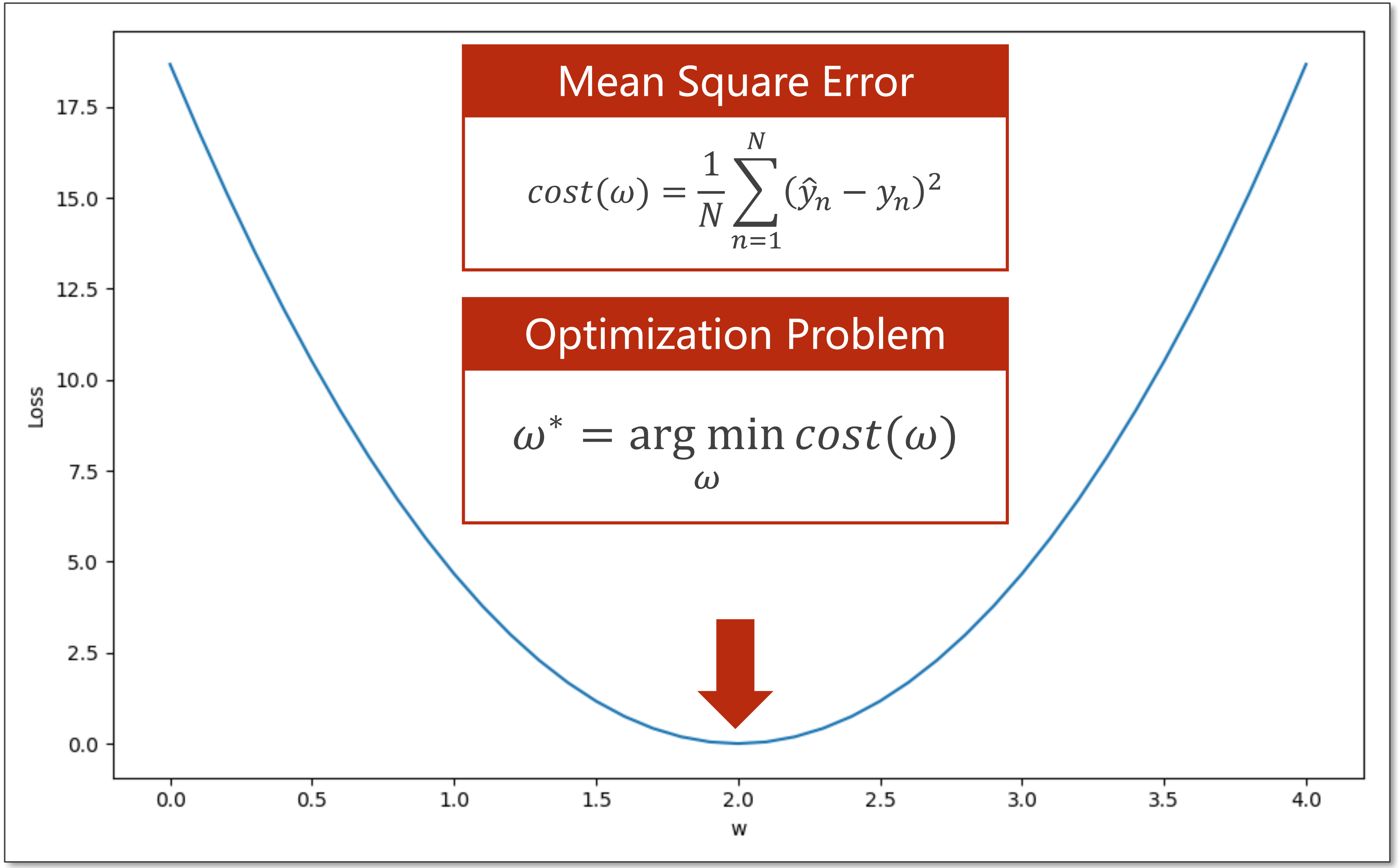


Revision

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



Optimization Problem



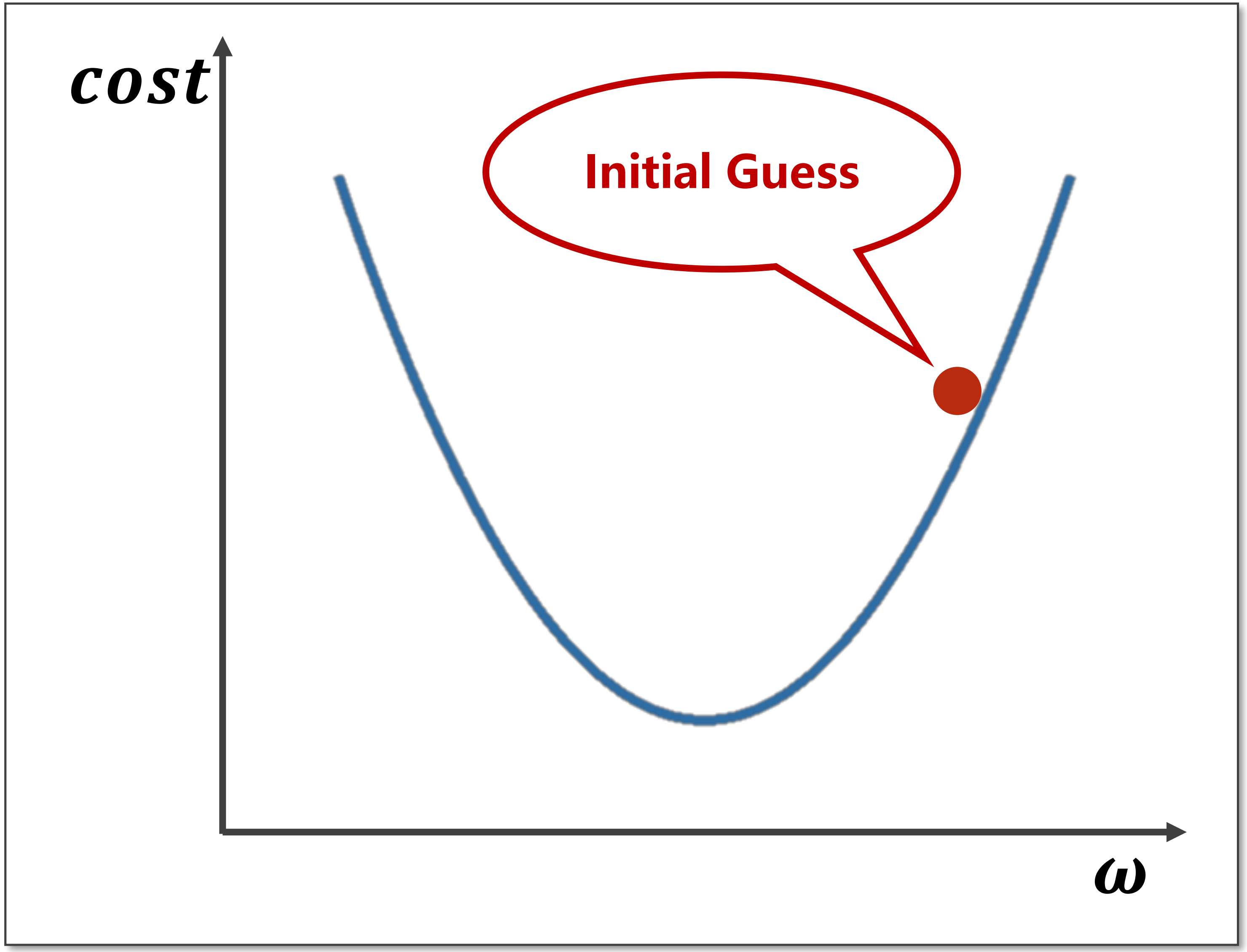
Mean Square Error

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

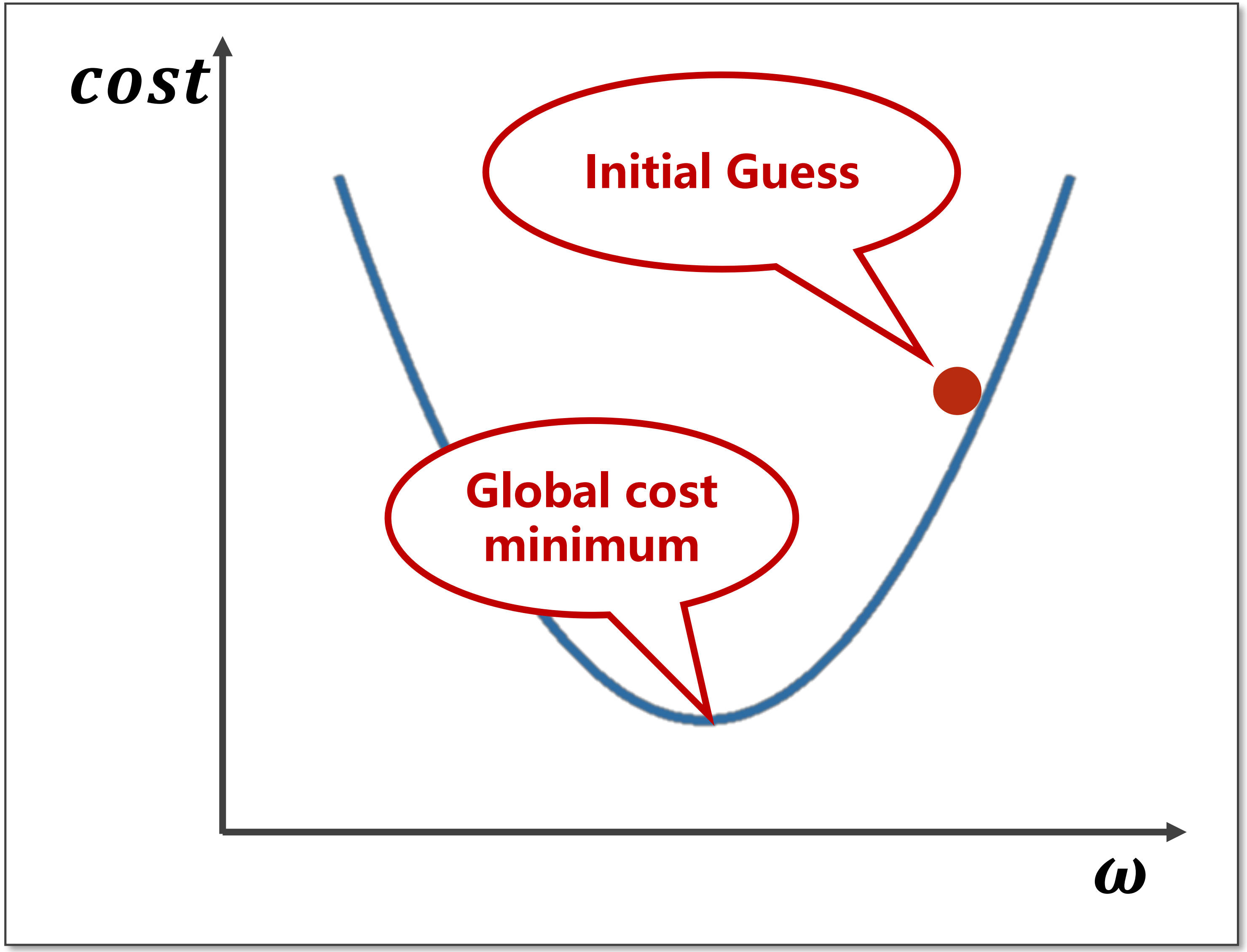
Optimization Problem

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

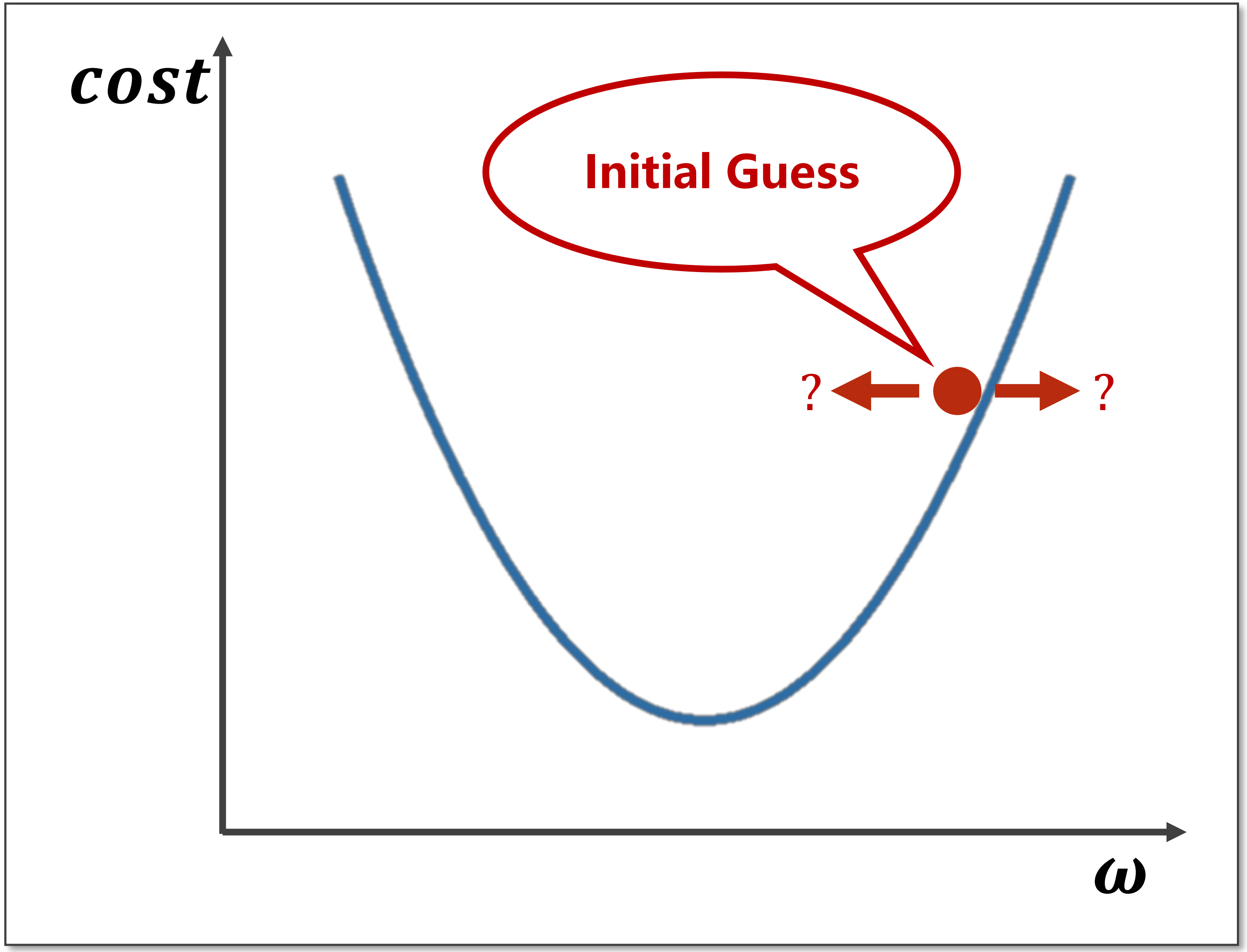
Gradient Descent Algorithm



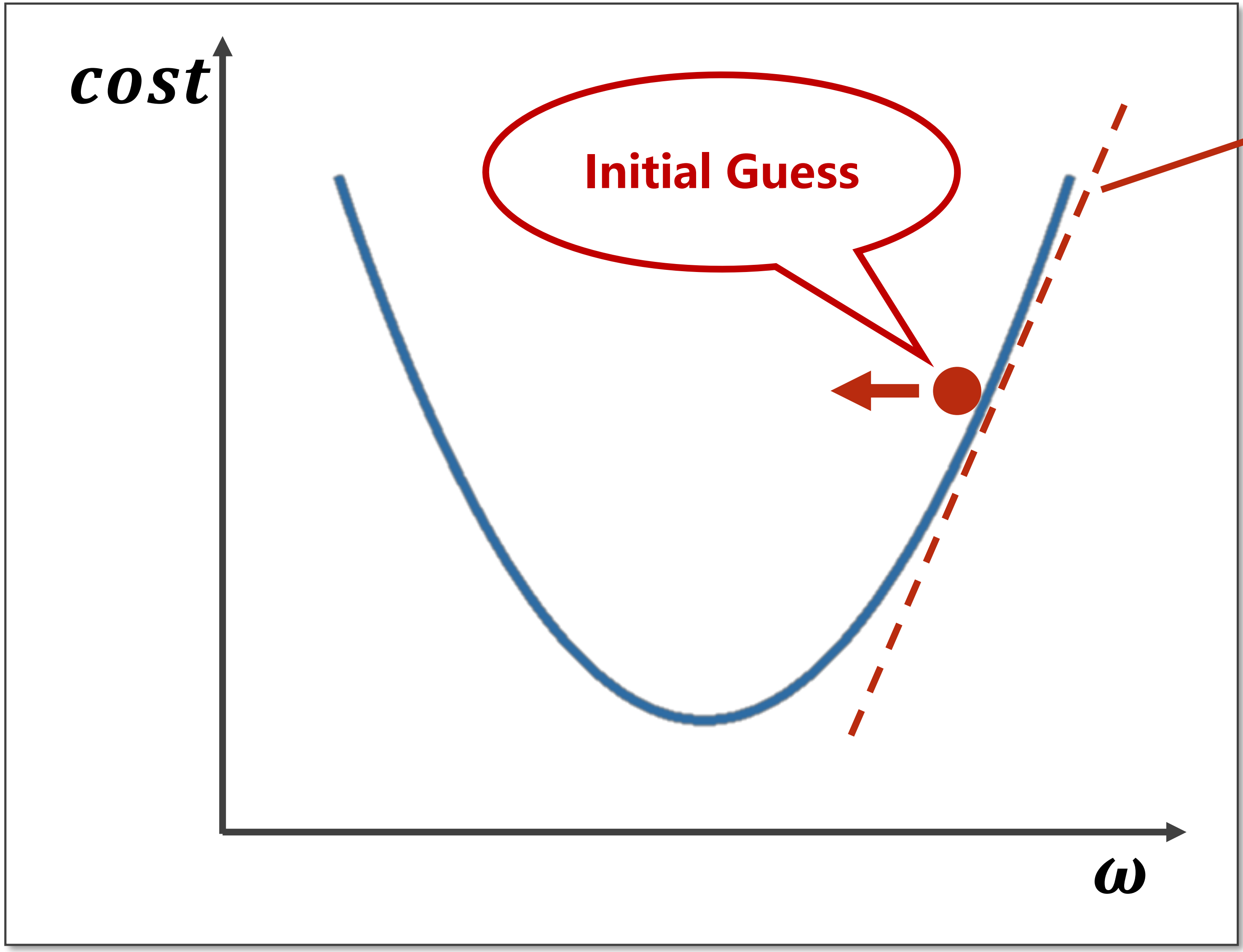
Gradient Descent Algorithm



Gradient Descent Algorithm



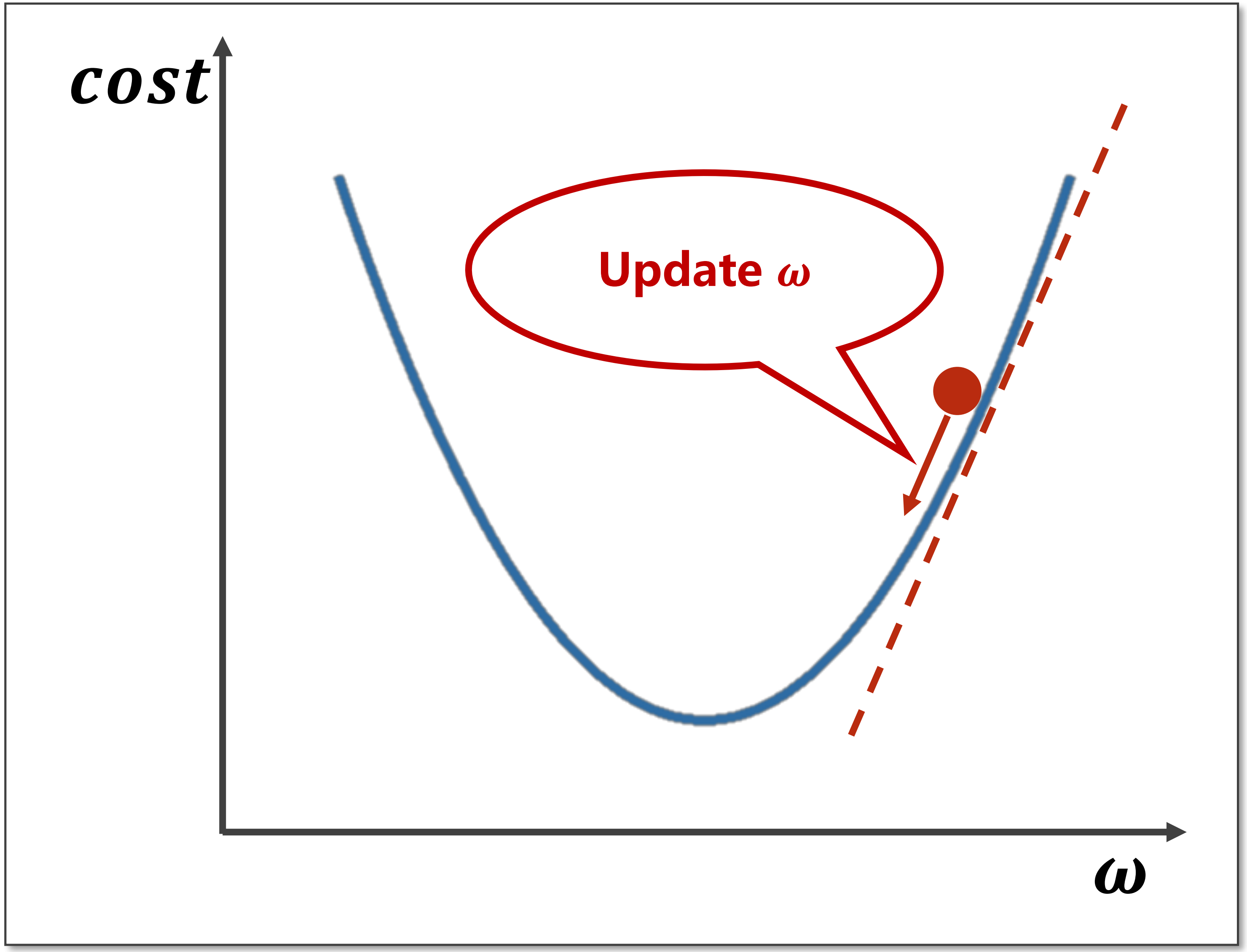
Gradient Descent Algorithm



Gradient

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

Gradient Descent Algorithm



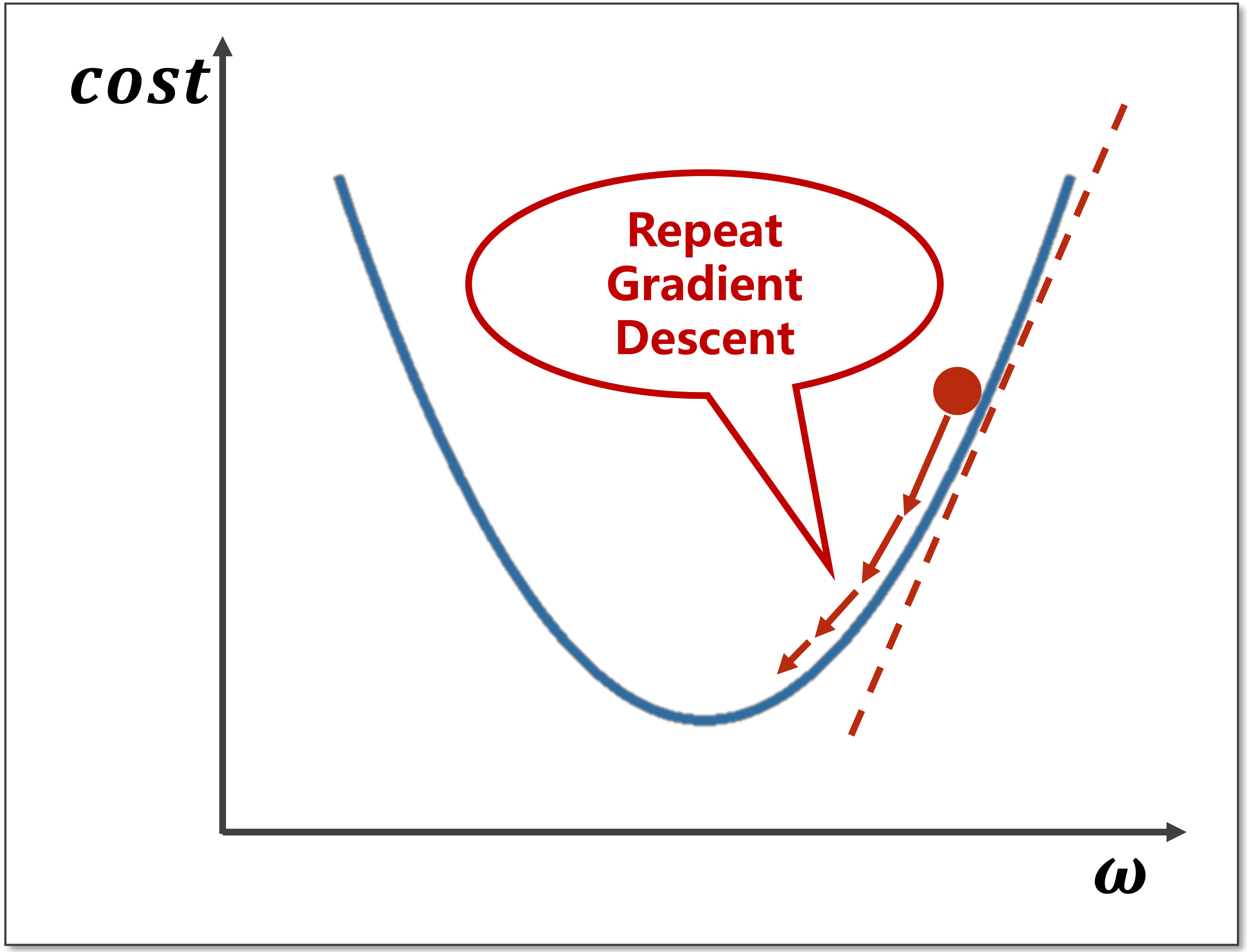
Gradient

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

Update

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

Gradient Descent Algorithm



Gradient

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

Update

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

Gradient Descent Algorithm

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

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

Gradient Descent Algorithm

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

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


Prepare the training set.

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))
```

```
w = 1.0
```

Initial guess of weight.

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

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

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)
```

Define the cost function

Mean Square Error

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

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

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

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
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

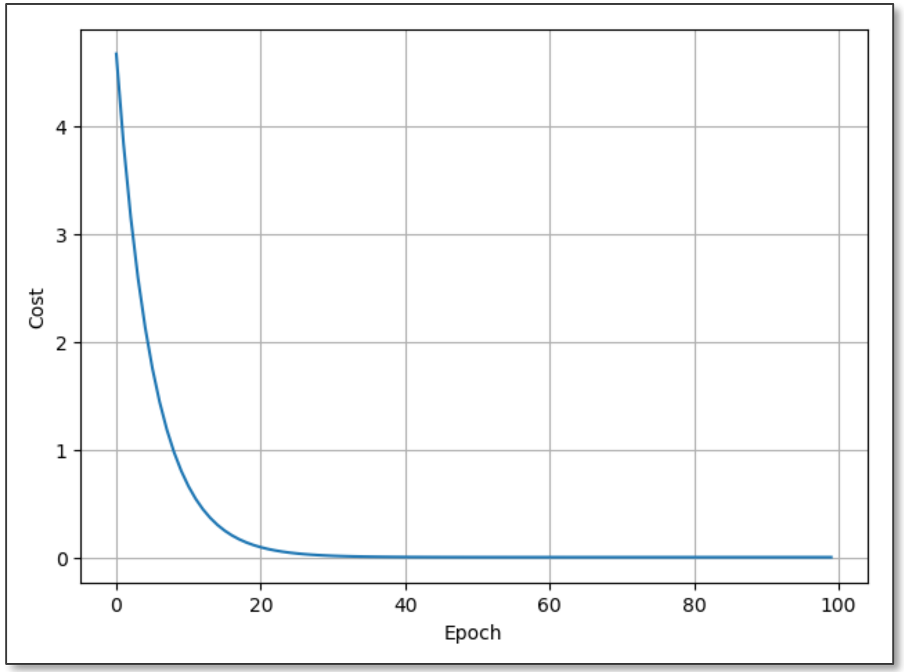
Update

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

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

Stochastic Gradient Descent

Gradient Descent

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

Derivative of Cost Function

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

Stochastic Gradient Descent

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

Derivative of Loss Function

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

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))
```

```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))
```


```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))
```


```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.

PyTorch Tutorial

03. Gradient Descent