

# ML/DL for Everyone with PYTORCH

## Lecture 3: Gradient Descent

Sung Kim <[hunkim+ml@gmail.com](mailto:hunkim+ml@gmail.com)> HKUST

Code: <https://github.com/hunkim/PyTorchZeroToAll>

Slides: <http://bit.ly/PyTorchZeroAll>

Videos: <http://bit.ly/PyTorchVideo>



# Call for Comments

Please feel free to add comments directly on these slides.

Other slides: <http://bit.ly/PyTorchZeroAll>



# ML/DL for Everyone with PYTORCH

## Lecture 3: Gradient Descent

Sung Kim <[hunkim+ml@gmail.com](mailto:hunkim+ml@gmail.com)> HKUST

Code: <https://github.com/hunkim/PyTorchZeroToAll>

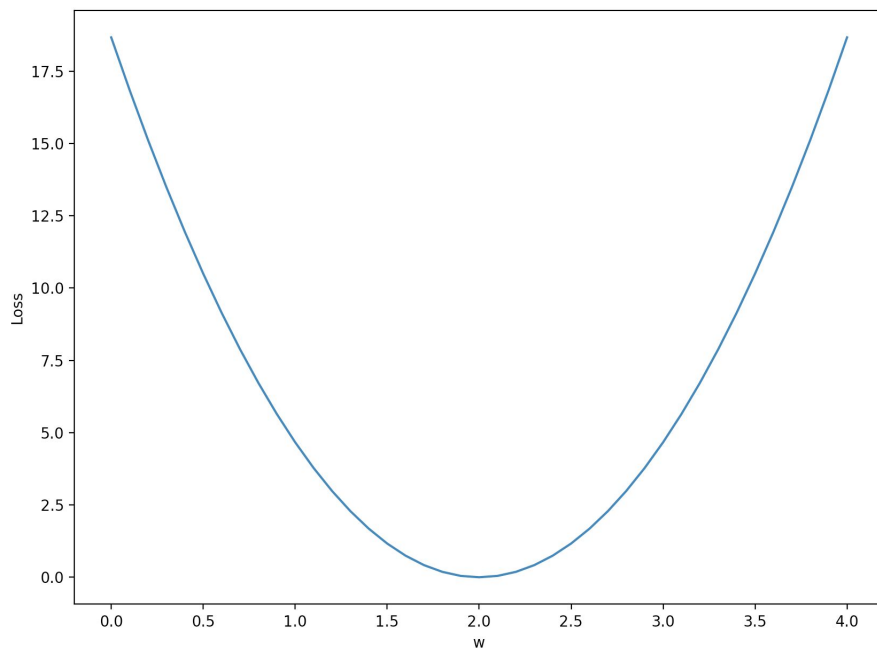
Slides: <http://bit.ly/PyTorchZeroAll>

Videos: <http://bit.ly/PyTorchVideo>



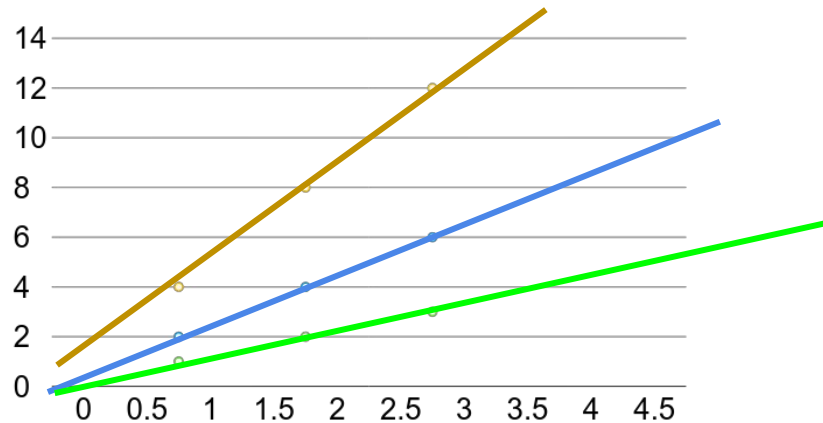
# Loss graph

Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
mean=56/3=18.7	mean=14/3=4.7	mean=0	mean=14/3=4.7	mean=56/3=18.7



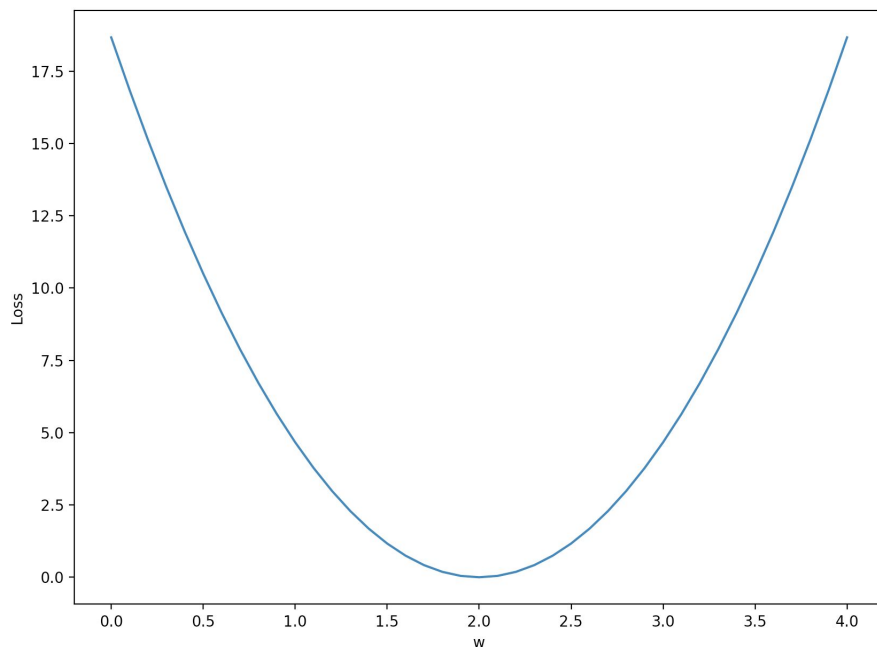
$$loss(w) = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n)^2$$

# Linear Regression error?



# What is the learning: find $w$ that minimizes the loss

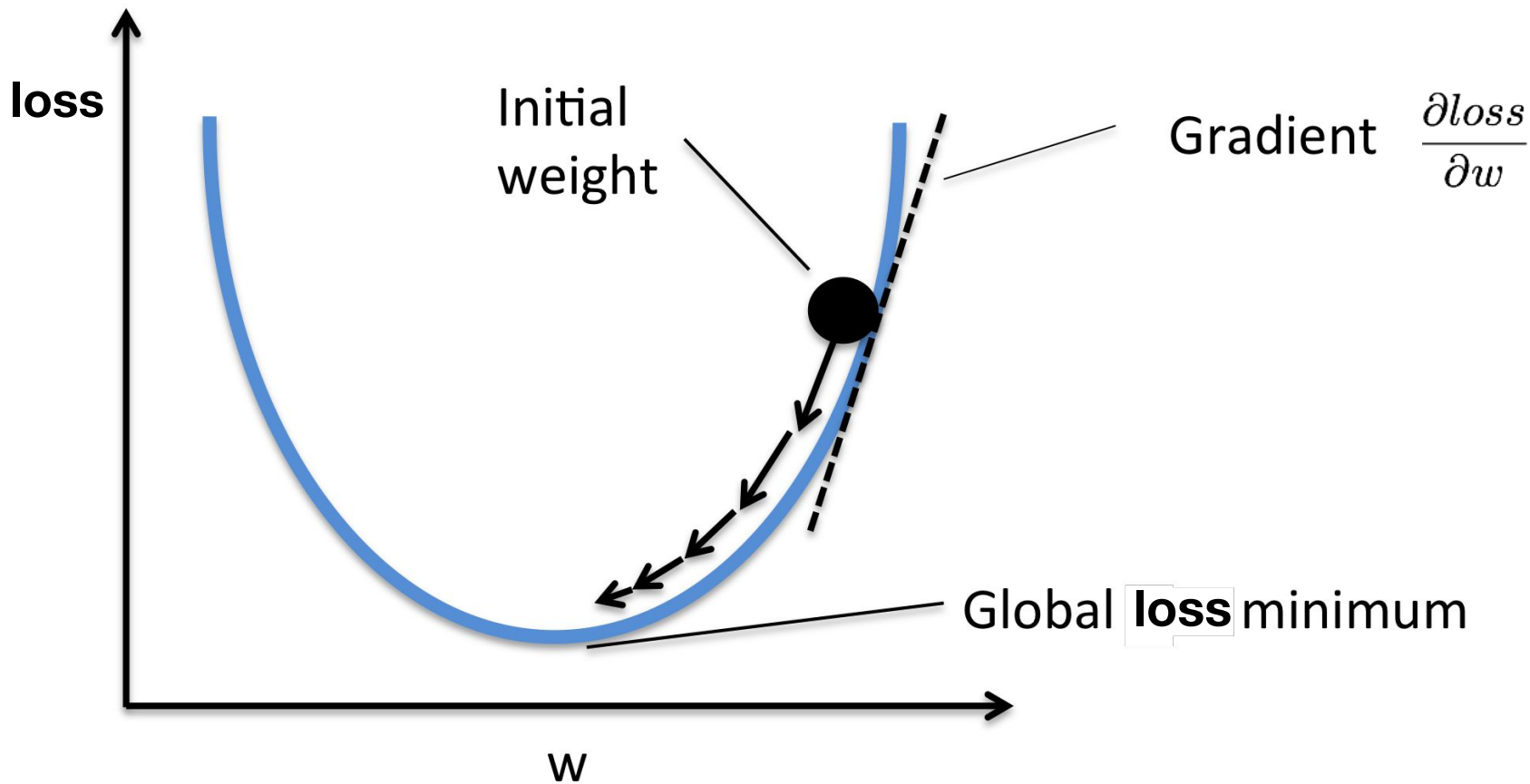
Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
mean=56/3=18.7	mean=14/3=4.7	mean=0	mean=14/3=4.7	mean=56/3=18.7



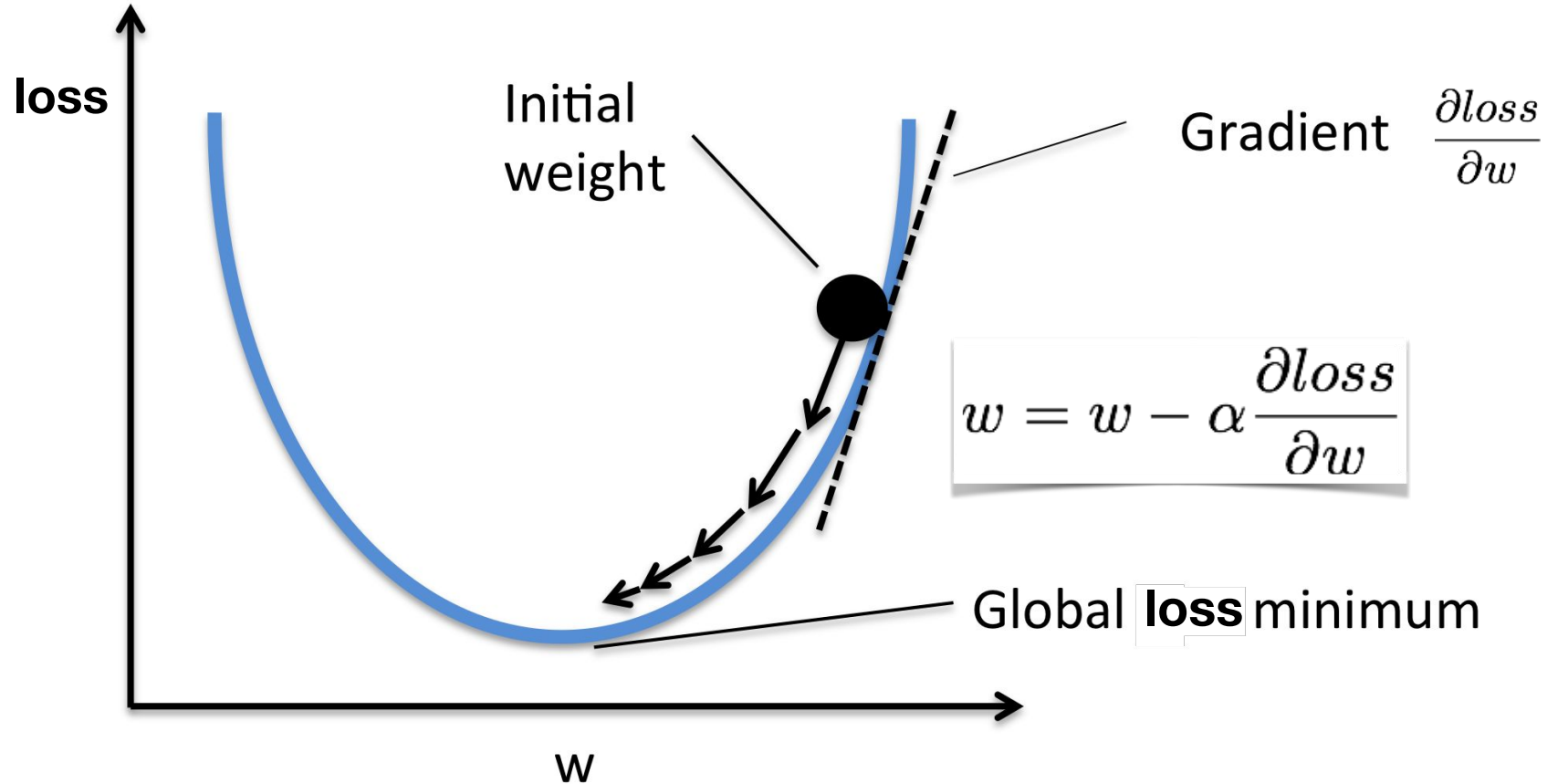
$$loss(w) = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n)^2$$

$$\arg \min_w loss(w)$$

# Gradient descent algorithm



# Gradient descent algorithm





# Derivative

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

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

# Derivative

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

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

$$\frac{\partial loss}{\partial w} = ?$$

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

$$\frac{\partial loss}{\partial w} = ?$$

# Derivative

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

$$\frac{\partial loss}{\partial w} = ?$$

YOUR INPUT:  
 $f(w) =$

$(xw - y)^2$

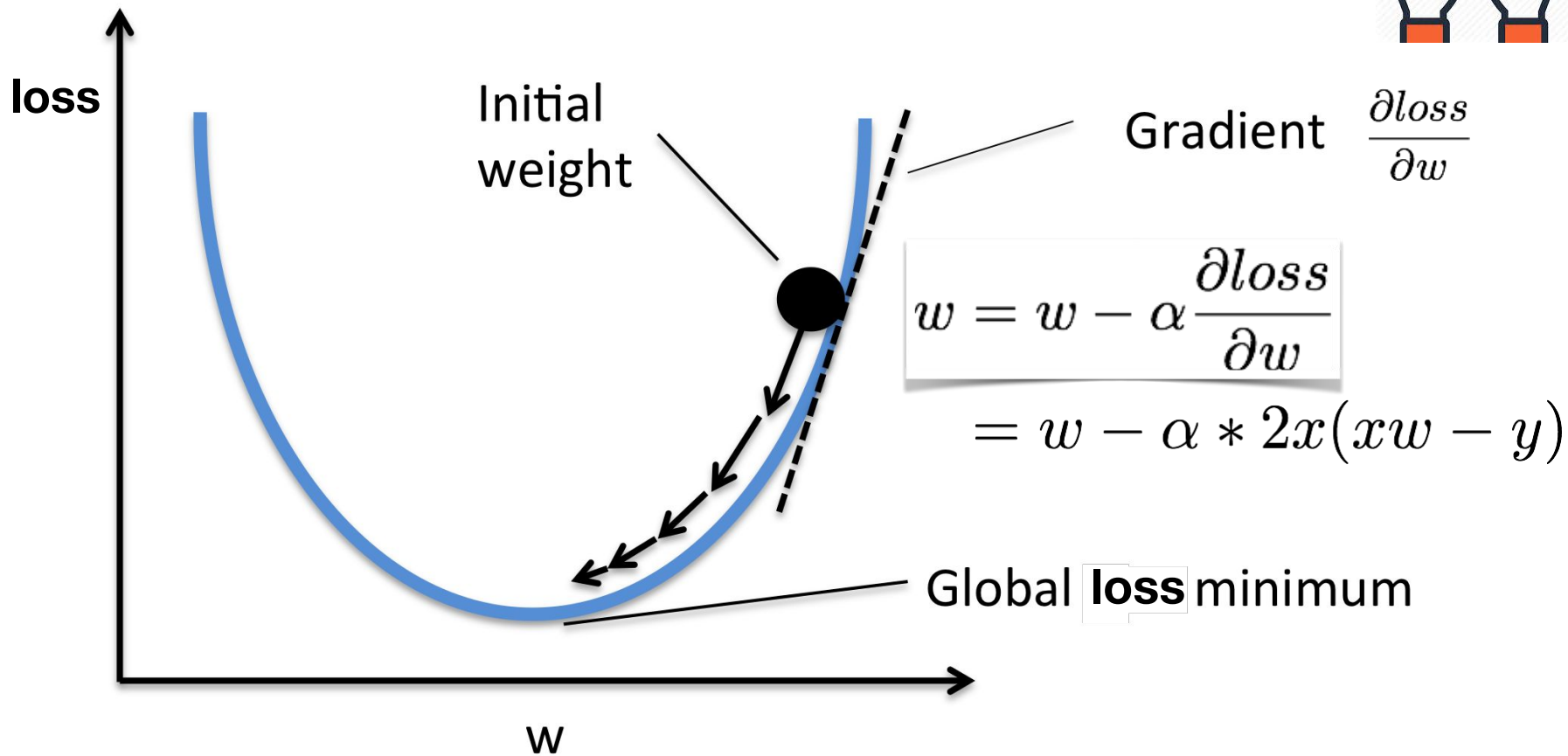
Simplify Roots/zeros

FIRST DERIVATIVE:  
 $\frac{d}{dw}[f(w)] = f'(w) =$

**The steps of calculation are displayed.**  
Move the mouse over a derivative  $\frac{d}{dw}[\dots]$  or tap it in order to show its calculation.

$$\begin{aligned} & \frac{d}{dw} [(xw - y)^2] \\ &= 2(xw - y) \cdot \frac{d}{dw} [xw - y] \\ &= 2 \left( x \cdot \frac{d}{dw} [w] + \frac{d}{dw} [-y] \right) (xw - y) \\ &= 2(1x + 0)(xw - y) \\ &= 2x(xw - y) \end{aligned}$$

# Let's implement!



# Data, Model, Loss, and Gradient



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

```
w = 1.0 # a random guess: random value
```

```
# our model forward pass
```

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

```
# Loss function
```

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

```
# compute gradient
```

```
def gradient(x, y): # d_loss/d_w
    return 2 * x * (x * w - y)
```

$$2x(xw - y)$$

# Training: updating weight



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

```
w = 1.0 # a random guess: random value
```

```
# our model forward pass
```

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

```
# Loss function
```

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

```
# compute gradient
```

```
def gradient(x, y): # d_loss/d_w
    return 2 * x * (x * w - y)
```

```
# Before training
```

```
print("predict (before training)", 4, forward(4))
```

```
# Training Loop
```

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

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

```
# After training
```

```
print("predict (after training)", "4 hours", forward(4))
```

**predict (before training) 4 4.0**

grad: 1.0 2.0 -2.0

grad: 2.0 4.0 -7.84

grad: 3.0 6.0 -16.23

progress: 0 w= 1.26 loss= 4.92

grad: 1.0 2.0 -1.48

grad: 2.0 4.0 -5.8

grad: 3.0 6.0 -12.0

progress: 1 w= 1.45 loss= 2.69

grad: 1.0 2.0 -1.09

grad: 2.0 4.0 -4.29

grad: 3.0 6.0 -8.87

progress: 2 w= 1.6 loss= 1.47

grad: 1.0 2.0 -0.81

grad: 2.0 4.0 -3.17

grad: 3.0 6.0 -6.56

..

progress: 7 w= 1.91 loss= 0.07

grad: 1.0 2.0 -0.18

grad: 2.0 4.0 -0.7

grad: 3.0 6.0 -1.45

progress: 8 w= 1.93 loss= 0.04

grad: 1.0 2.0 -0.13

grad: 2.0 4.0 -0.52

grad: 3.0 6.0 -1.07

progress: 9 w= 1.95 loss= 0.02

**predict (after training) 4 hours 7.80**

# Output

(from gradient numeric computation)



```
# Before training
```

```
print("predict (before training)", 4, forward(4))
```

```
# Training Loop
```

```
for epoch in range(100):
```

```
    for x_val, y_val in zip(x_data, y_data):
```

```
        grad = gradient(x_val, y_val)
```

```
        w = w - 0.01 * grad
```

```
        print("\tgrad: ", x_val, y_val, grad)
```

```
        l = loss(x_val, y_val)
```

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

```
# After training
```

```
print("predict (after training)", "4 hours", forward(4))
```



## Exercise 3-1: compute gradient

$$\hat{y} = x^2 w_2 + x w_1 + b$$

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

$$\frac{\partial loss}{\partial w_1} = ?$$

$$\frac{\partial loss}{\partial w_2} = ?$$

## Exercise 3-2: implement

$$\hat{y} = x^2 w_2 + x w_1 + b$$

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

$$\frac{\partial loss}{\partial w_1} = ?$$

$$\frac{\partial loss}{\partial w_2} = ?$$



## Lecture 4: Back-propagation