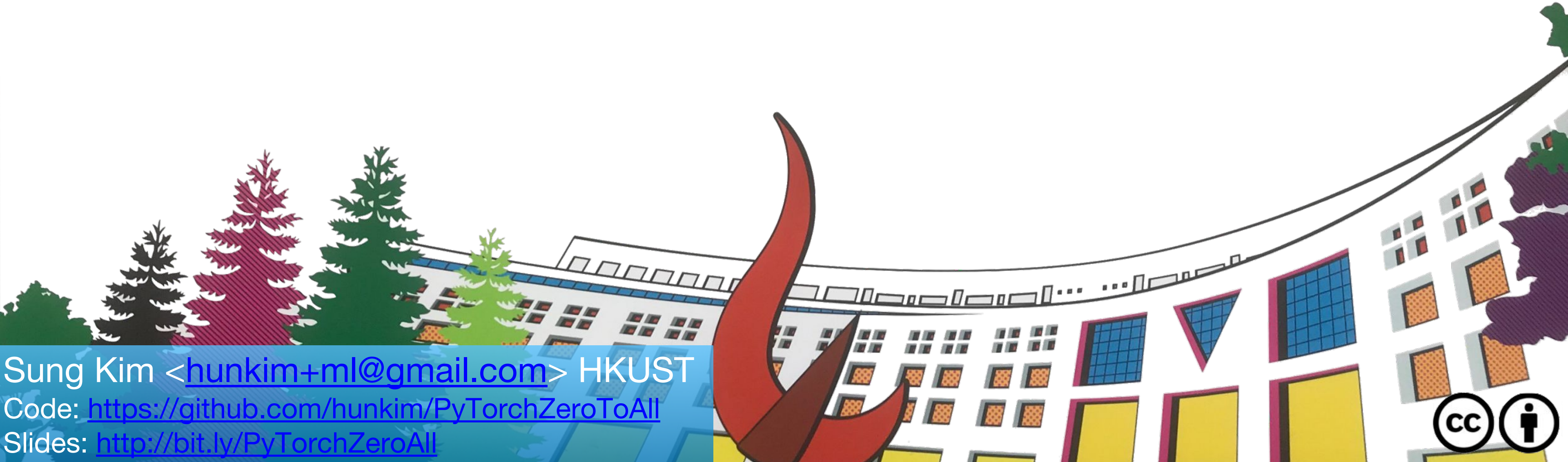
ML/DL for Everyone with PYTORCH

Lecture 3: Gradient Descent



Sung Kim <hunkim+ml@gmail.com> HKUST

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

Slides: http://bit.ly/PyTorchZeroAll

Call for Comments

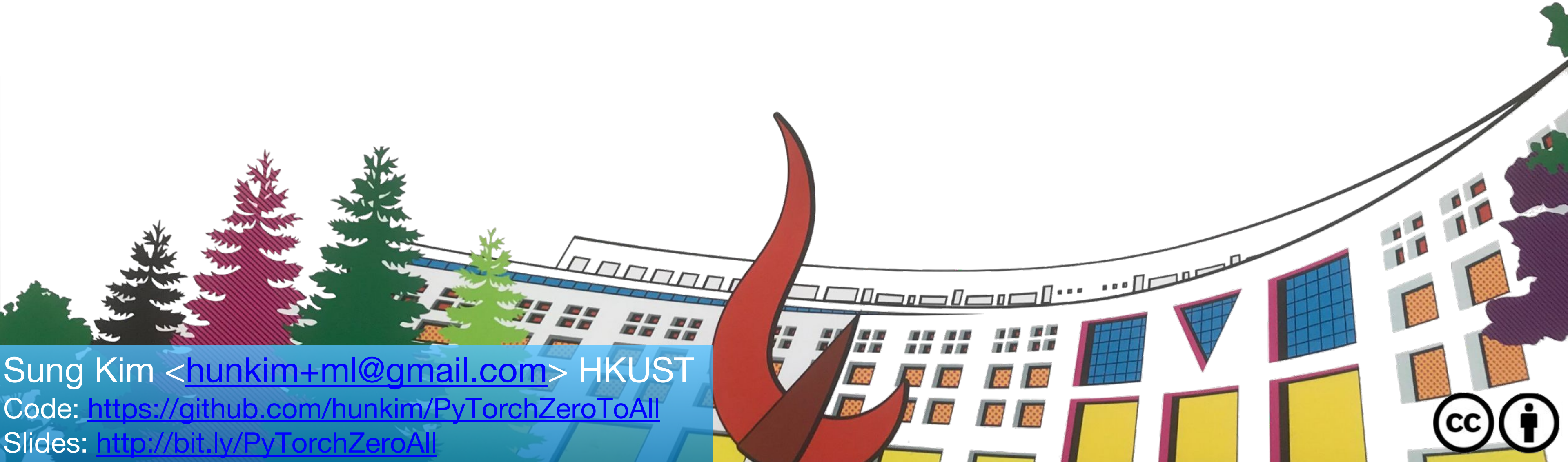
Please feel free to add comments directly on these slides.

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

Picture from http://www.tssablog.org/archives/3280

ML/DL for Everyone with PYTORCH

Lecture 3: Gradient Descent



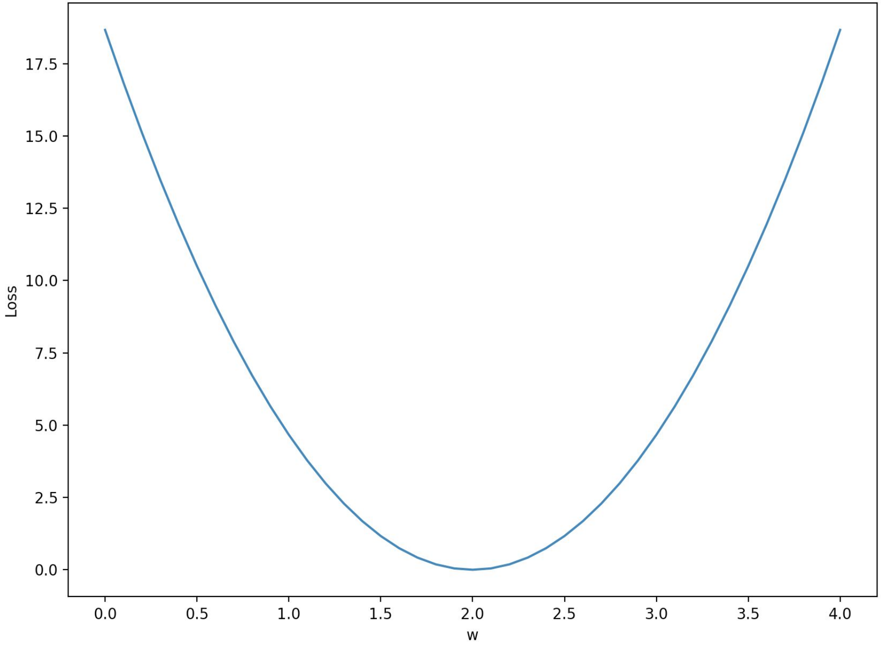
Sung Kim <hunkim+ml@gmail.com> HKUST

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

Slides: http://bit.ly/PyTorchZeroAll

Loss graph

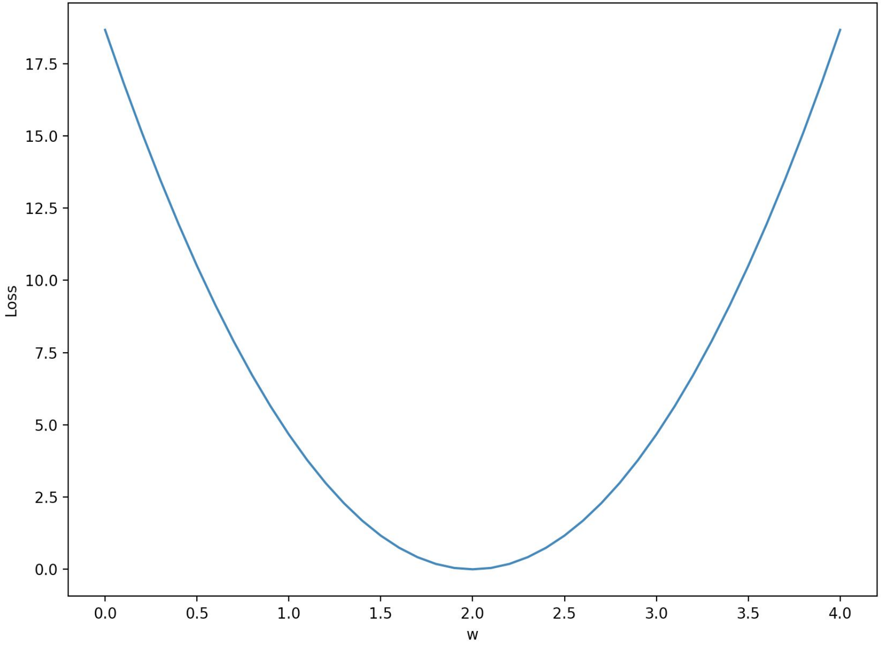
Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
mean=56/3=18.7	mean=14/3=4.7	mean=0	mean=14/3=4.7	mean=56/3=18.7



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

What is the learning: find **w** that minimizes the loss

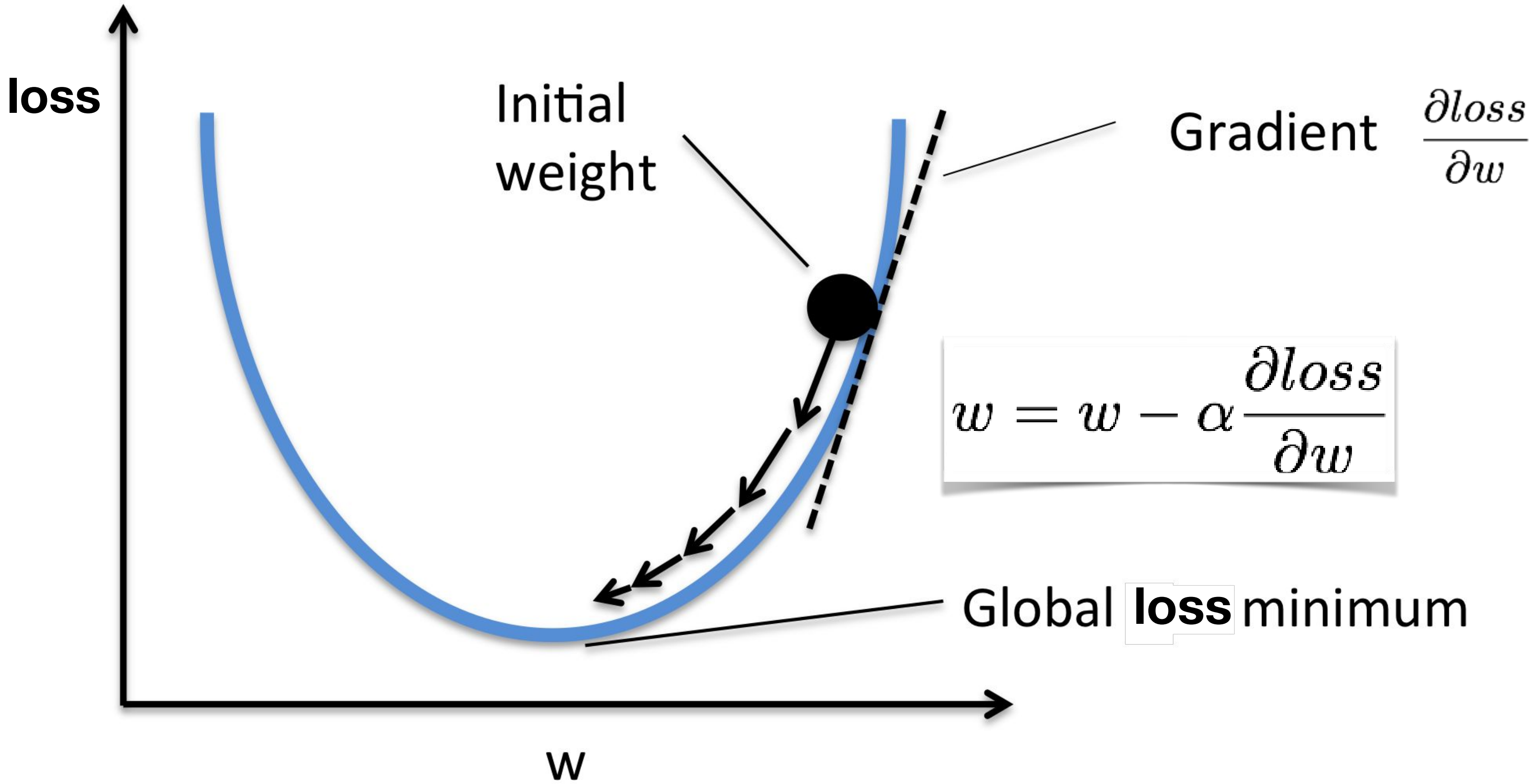
Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
mean=56/3=18.7	mean=14/3=4.7	mean=0	mean=14/3=4.7	mean=56/3=18.7



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

$$\arg\min_{w} loss(w)$$

Gradient descent algorithm



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 $\frac{d}{dw}[\ldots]$ or tap it in order to show its calculation.

$$\frac{d}{dw}\left[(xw - y)^2\right]$$

$$= 2(xw - y) \cdot \frac{d}{dw}[xw - y]$$

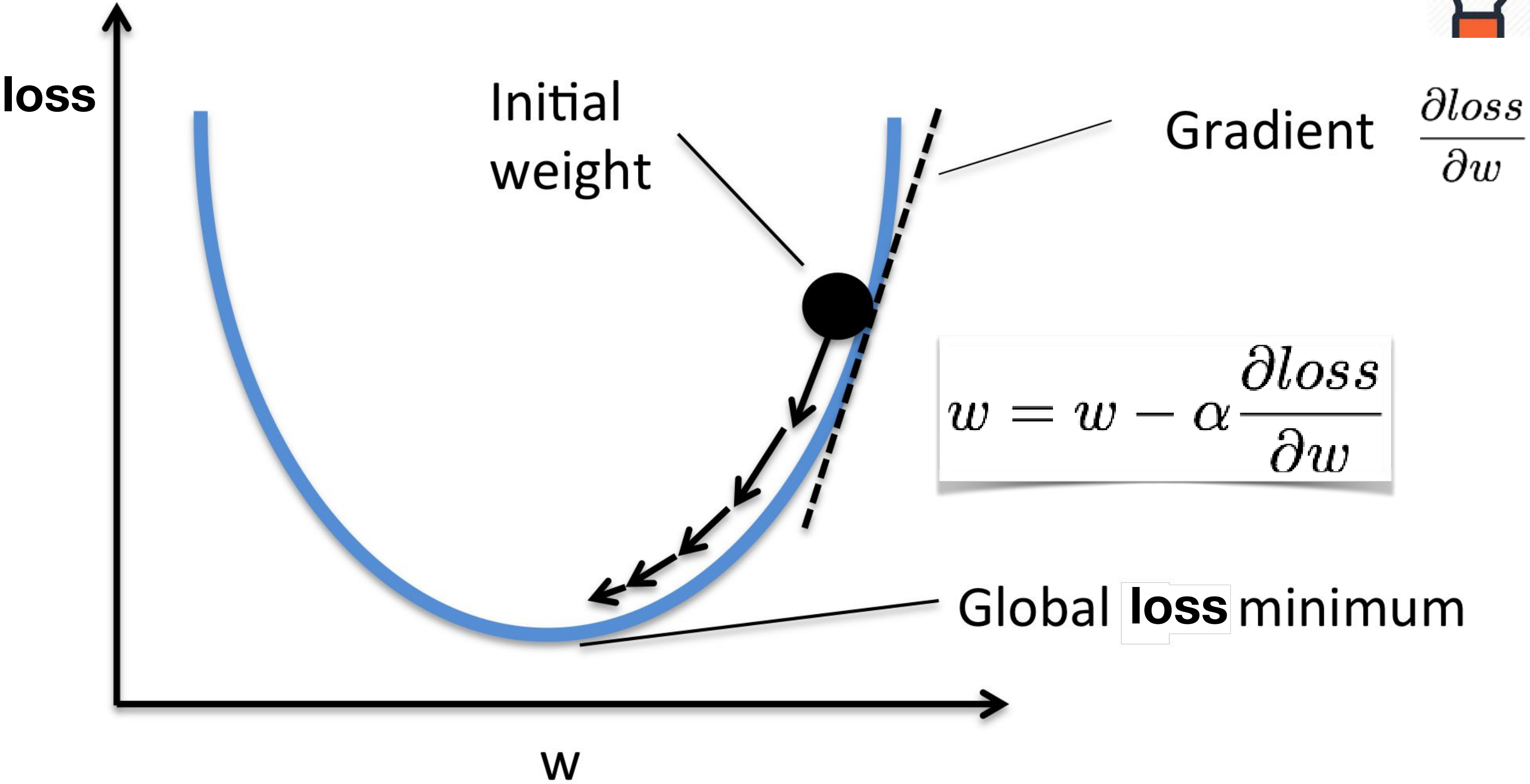
$$= 2\left(x \cdot \frac{d}{dw}[w] + \frac{d}{dw}[-y]\right)(xw - y)$$

$$= 2(1x + 0)(xw - y)$$

$$= 2x(xw - y)$$

https://www.derivative-calculator.net/

Let’s implement!



Data, Model, Loss, and Gradient

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

w = 1.0 # any 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)
```

Training: updating weight

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

w = 1.0 # any 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)
```

```python
# Before training
print("predict (before training)",  4, forward(4))

# Training loop
for epoch in range(10):
    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, l)

# After training
print("predict (after training)",  4, forward(4))
```

Output

(from gradient numeric computation)

```
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 -16.2288
progress: 0 4.919240100095999
	grad:  1.0 2.0 -1.478624
	grad:  2.0 4.0 -5.796206079999999
	grad:  3.0 6.0 -11.998146585599997
progress: 1 2.688769240265834
	grad:  1.0 2.0 -1.093164466688
	grad:  2.0 4.0 -4.285204709416961
	grad:  3.0 6.0 -8.87037374849311
progress: 2 1.4696334962911515
	grad:  1.0 2.0 -0.8081896081960389
	grad:  2.0 4.0 -3.1681032641284723
	grad:  3.0 6.0 -6.557973756745939
progress: 3 0.8032755585999681
	grad:  1.0 2.0 -0.59750427561463
	grad:  2.0 4.0 -2.3422167604093502
	grad:  3.0 6.0 -4.848388694047353
progress: 4 0.43905614881022015
	grad:  1.0 2.0 -0.44174208101320334
	grad:  2.0 4.0 -1.7316289575717576
	grad:  3.0 6.0 -3.584471942173538
progress: 5 0.2399802903801062
	grad:  1.0 2.0 -0.3265852213980338
	grad:  2.0 4.0 -1.2802140678802925
	grad:  3.0 6.0 -2.650043120512205
progress: 6 0.1311689630744999
	grad:  1.0 2.0 -0.241448373202223
	grad:  2.0 4.0 -0.946477622952715
	grad:  3.0 6.0 -1.9592086795121197
progress: 7 0.07169462478267678
	grad:  1.0 2.0 -0.17850567968888198
	grad:  2.0 4.0 -0.6997422643804168
	grad:  3.0 6.0 -1.4484664872674653
progress: 8 0.03918700813247573
	grad:  1.0 2.0 -0.13197139106214673
	grad:  2.0 4.0 -0.5173278529636143
	grad:  3.0 6.0 -1.0708686556346834
progress: 9 0.021418922423117836
predict (after training) 4 7.804863933862125
```

```python
# Before training
print("predict (before training)",  4, forward(4))

# Training loop
for epoch in range(10):
    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, l)

# After training
print("predict (after training)",  4, 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

$$\hat{y} = x^2 w_2 + x w_1 + b$$
$$loss = (\hat{y} - y)^2$$

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

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

WHAT NEXT

Lecture 4: Back-propagation