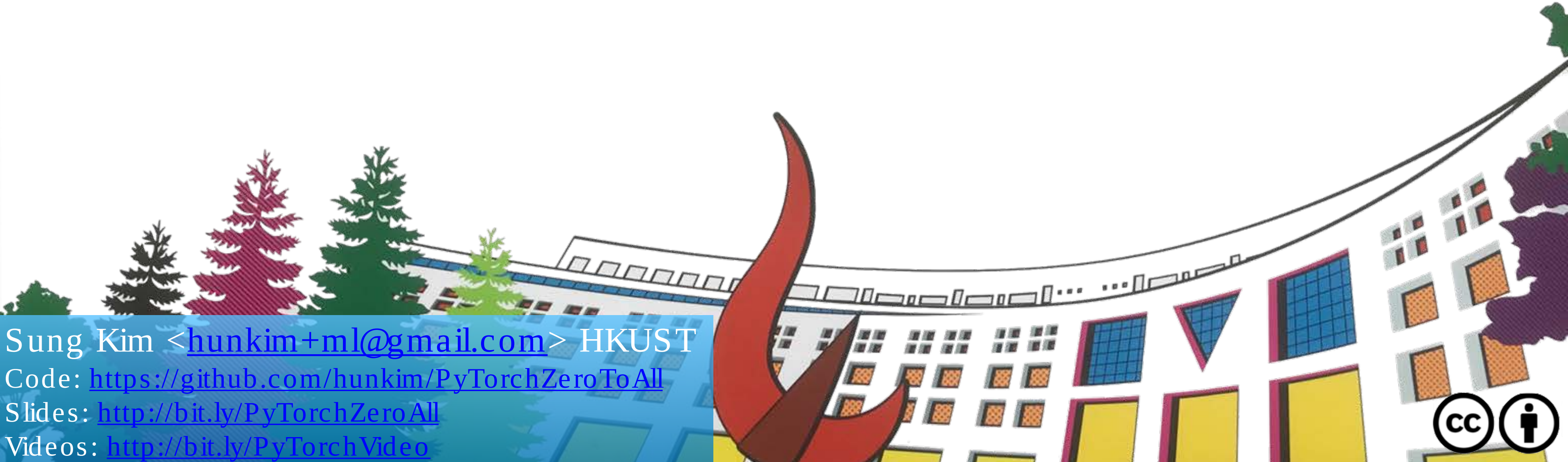
ML/DL for Everyone with PYTORCH

Lecture 4: Back-propagation



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

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

Slides: http://bit.ly/PyTorchZeroAll

Videos: http://bit.ly/PyTorchVideo

Call for Comments

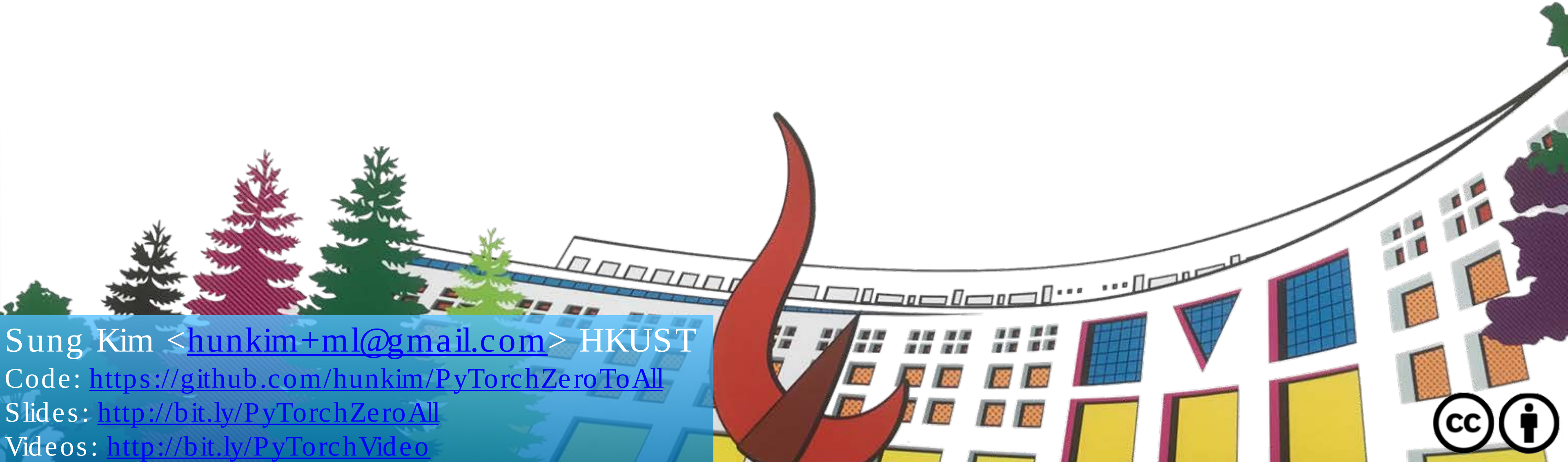
Please feel free to add comments directly on these slides.

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

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

ML/DL for Everyone with PYTORCH

Lecture 4: Back-propagation & Autograd



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

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

Slides: http://bit.ly/PyTorchZeroAll

Videos: http://bit.ly/PyTorchVideo

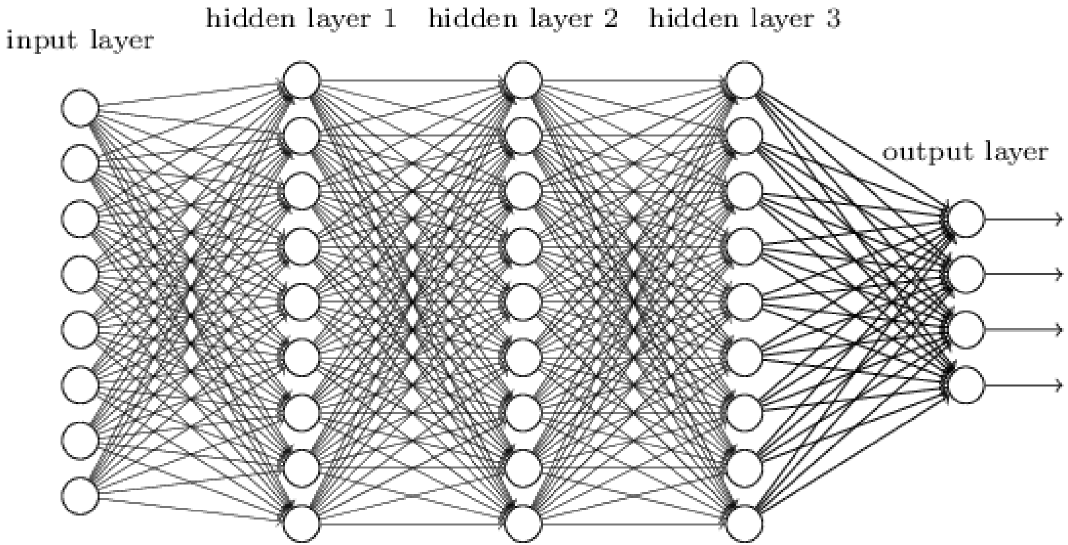
Computing gradient in simple network

Gradient of **loss**
with respect to ***w***

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

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

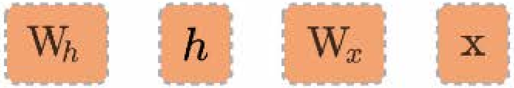
Complicated network?



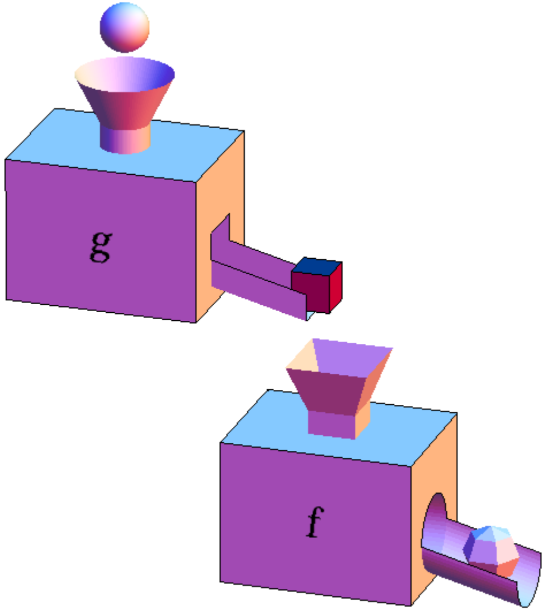
Gradient of **loss** with respect to ***w***

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

Better way? Computational graph + chain rule



Chain Rule



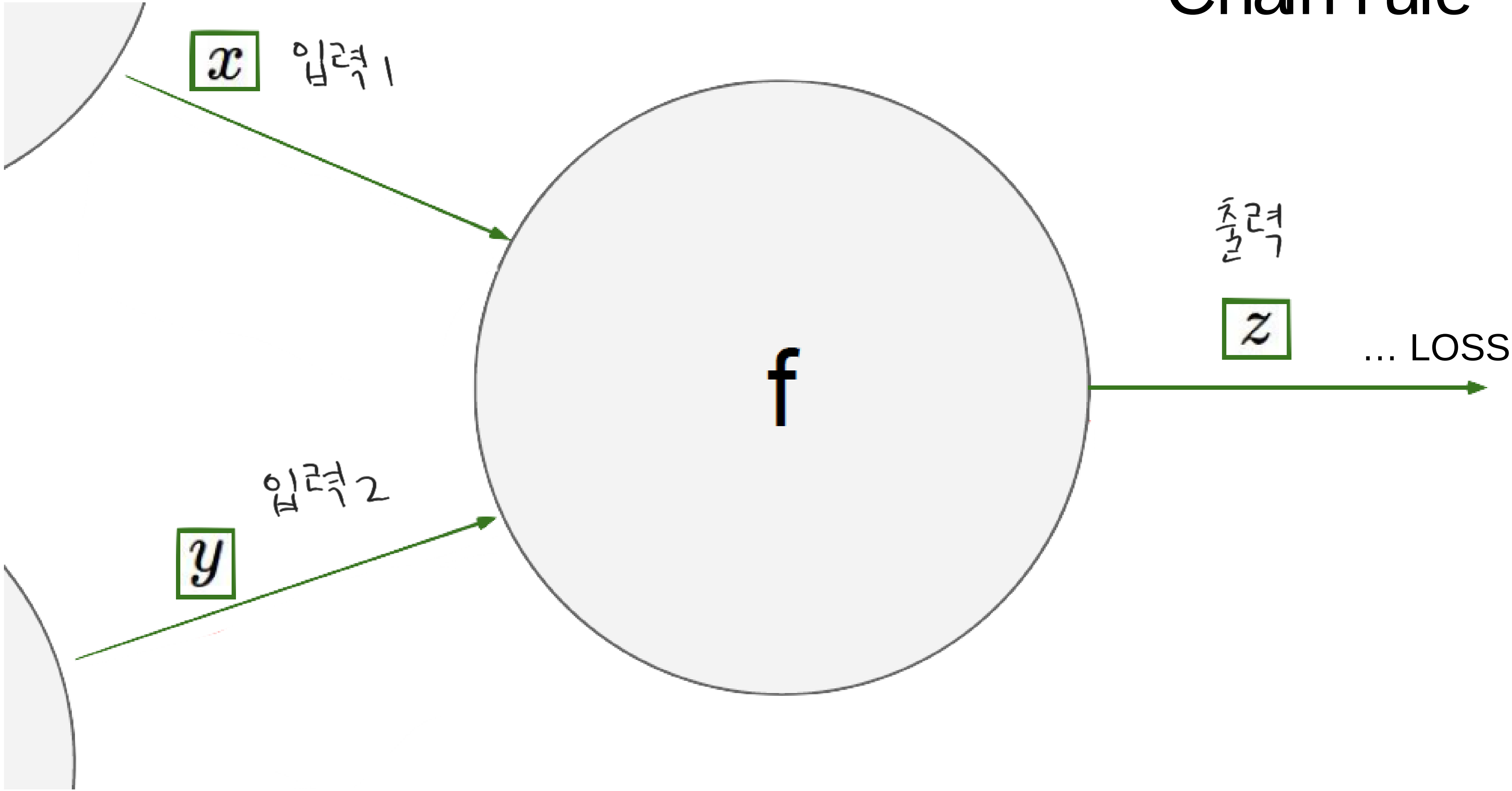
$$y = f(g)\ ; \quad g = g(x)$$

$$\frac{df}{dx} = \frac{df}{dg}\,\frac{dg}{dx}$$

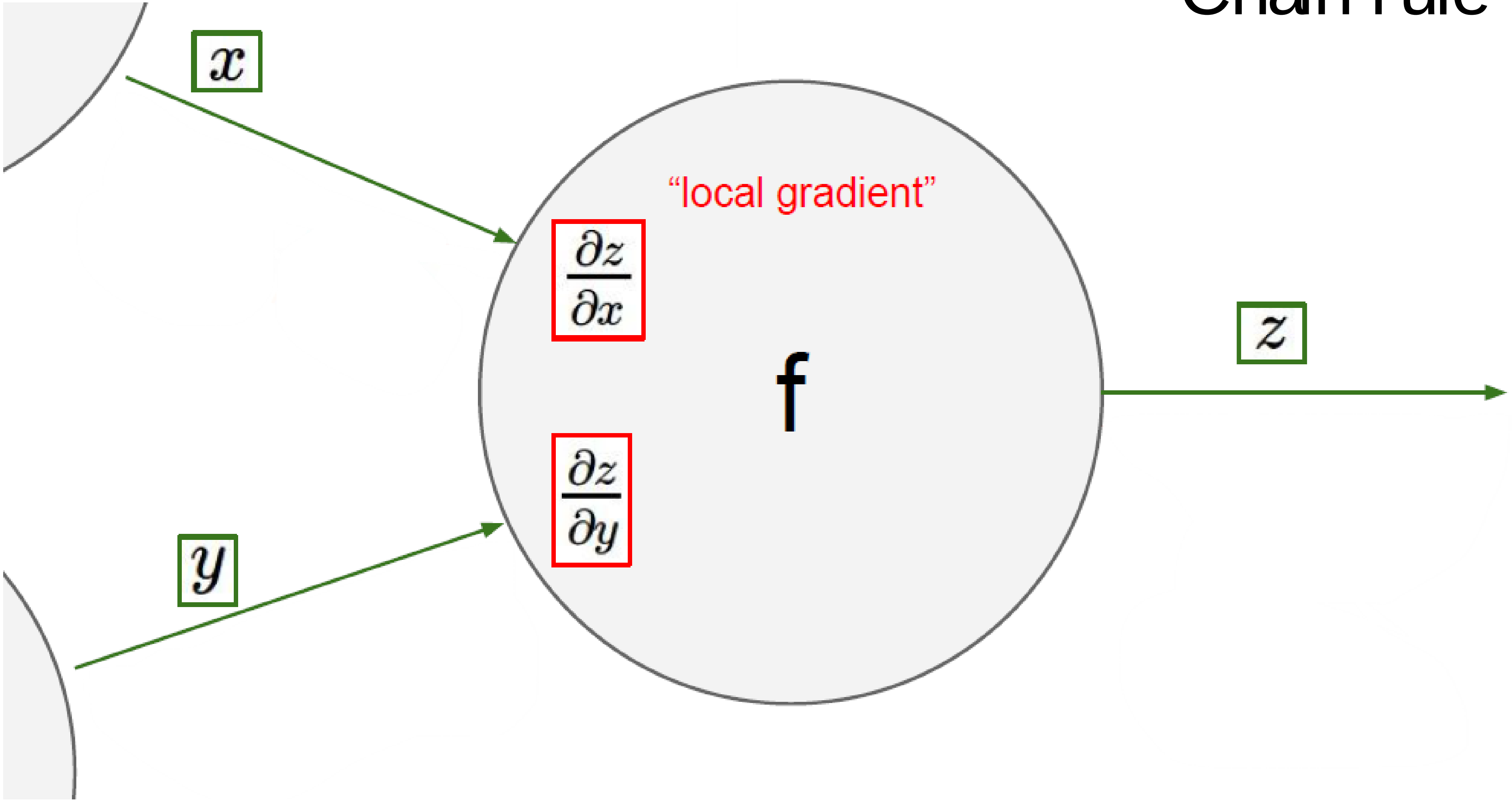
local gradient

http://mathinsight.org/image/function_machines_composed

Chain rule

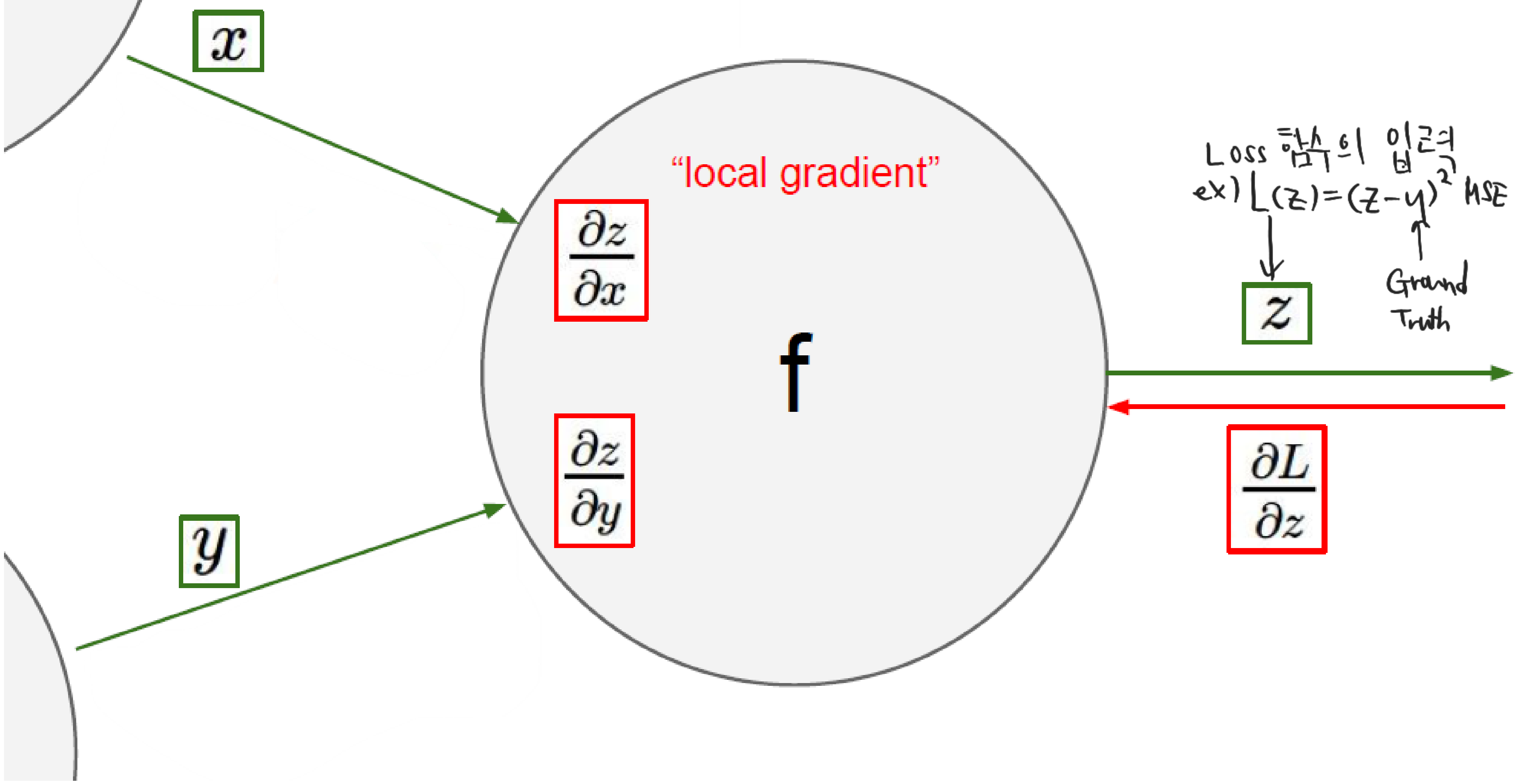


Chain rule



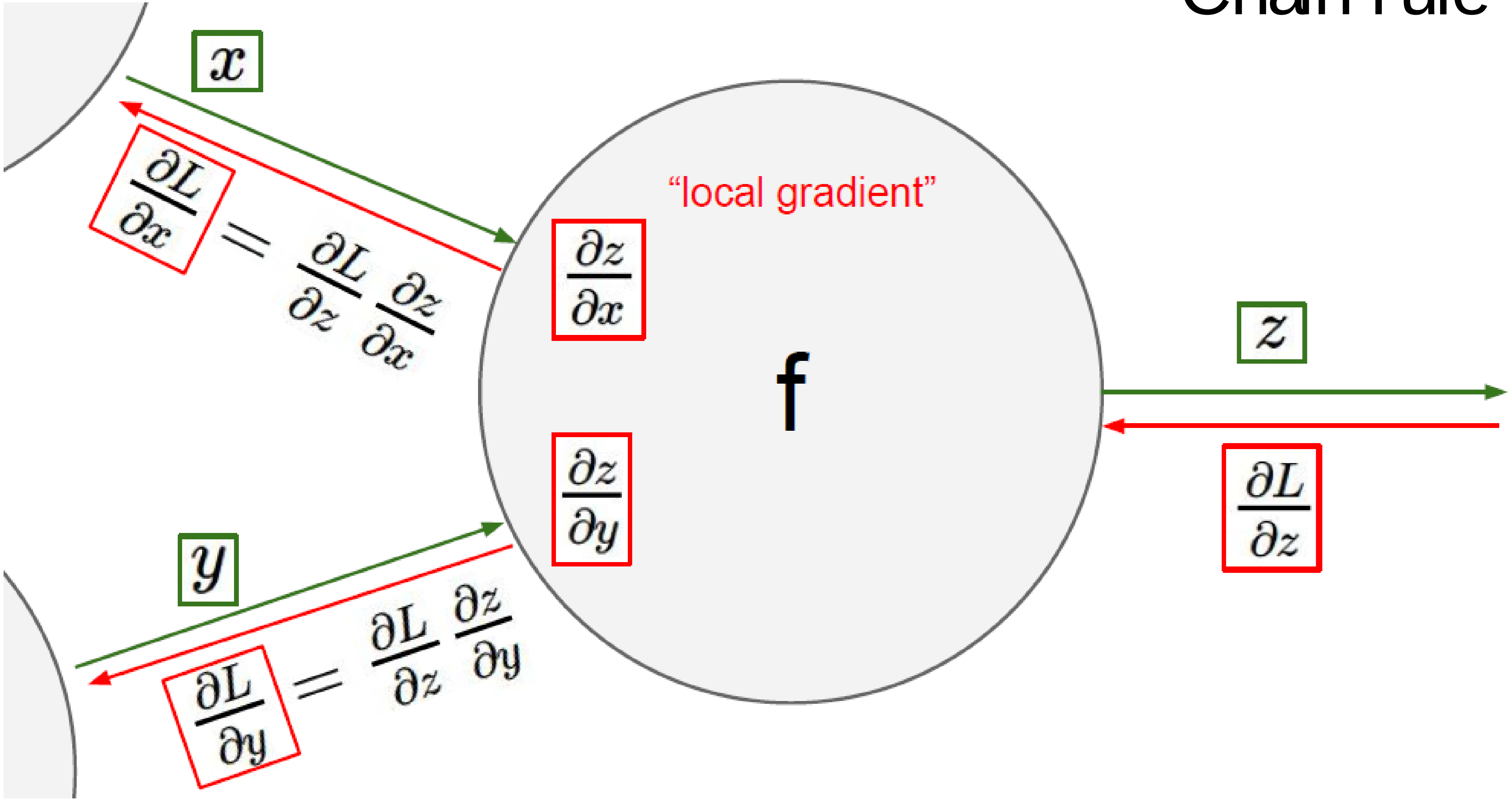
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

Chain rule

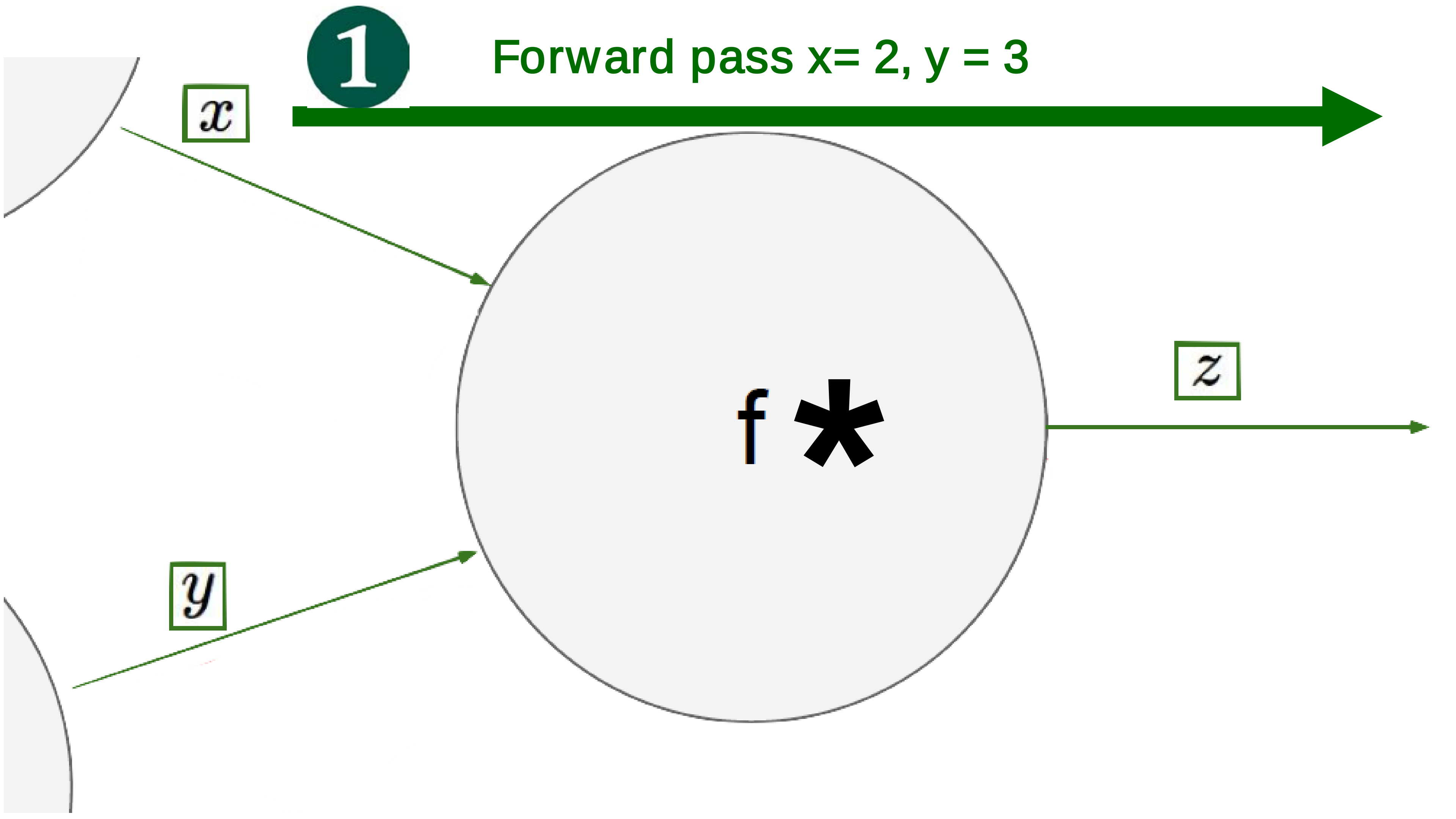


x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Loss 함수의 입력
ex) $L(z)=(z-y)^2$ MSE
z
Ground Truth
$\frac{\partial L}{\partial z}$

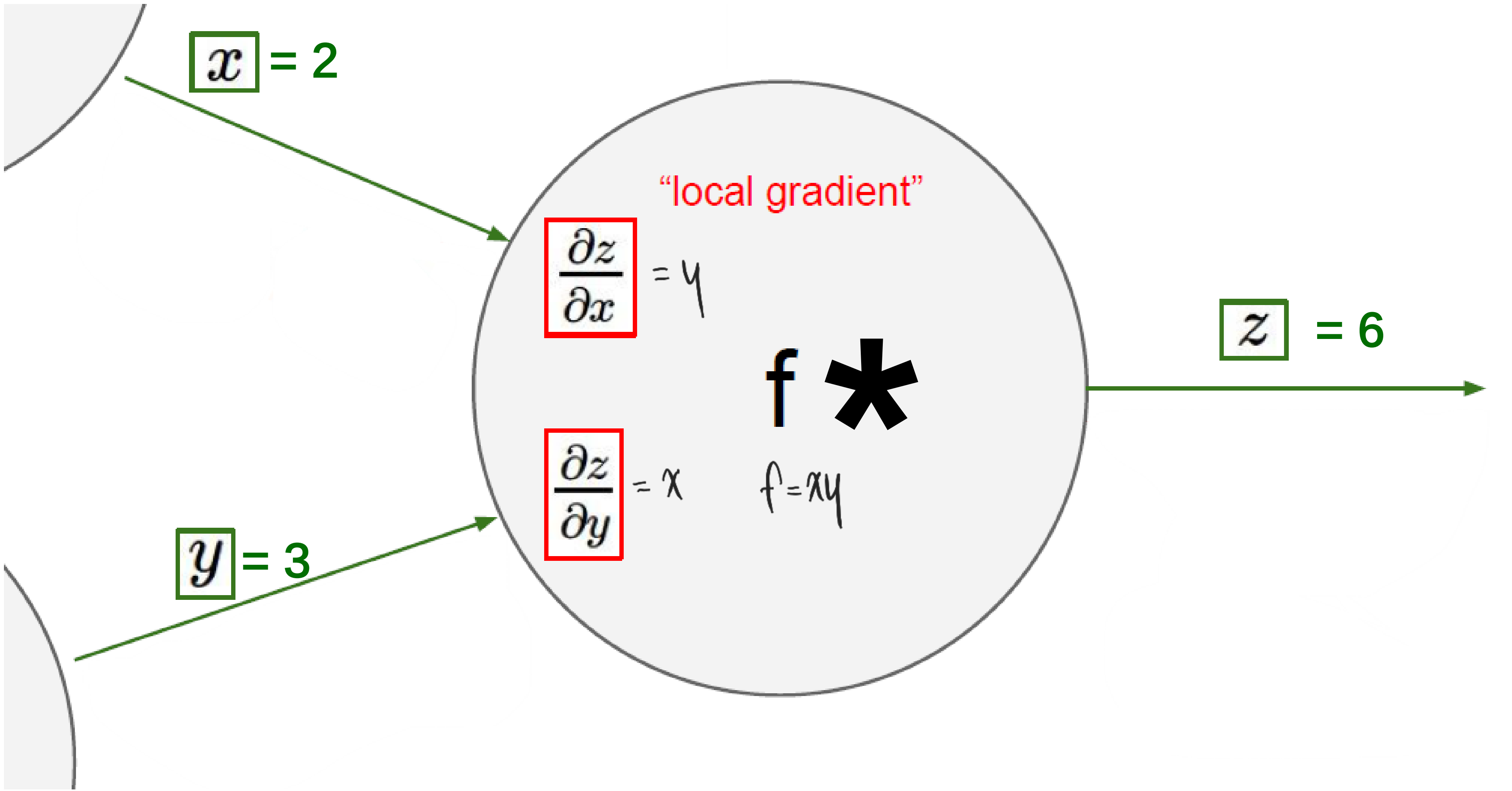
Chain rule



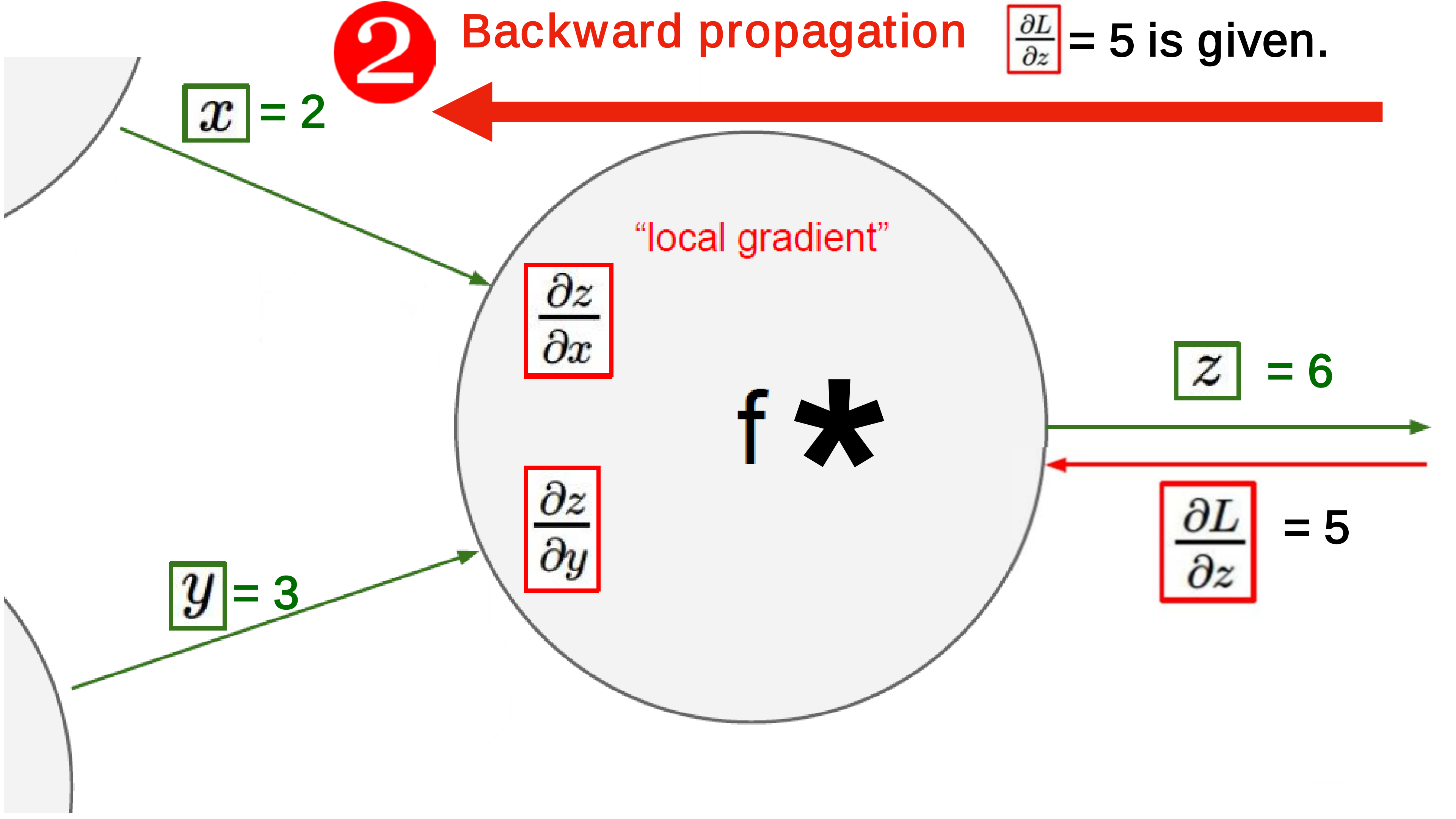
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$



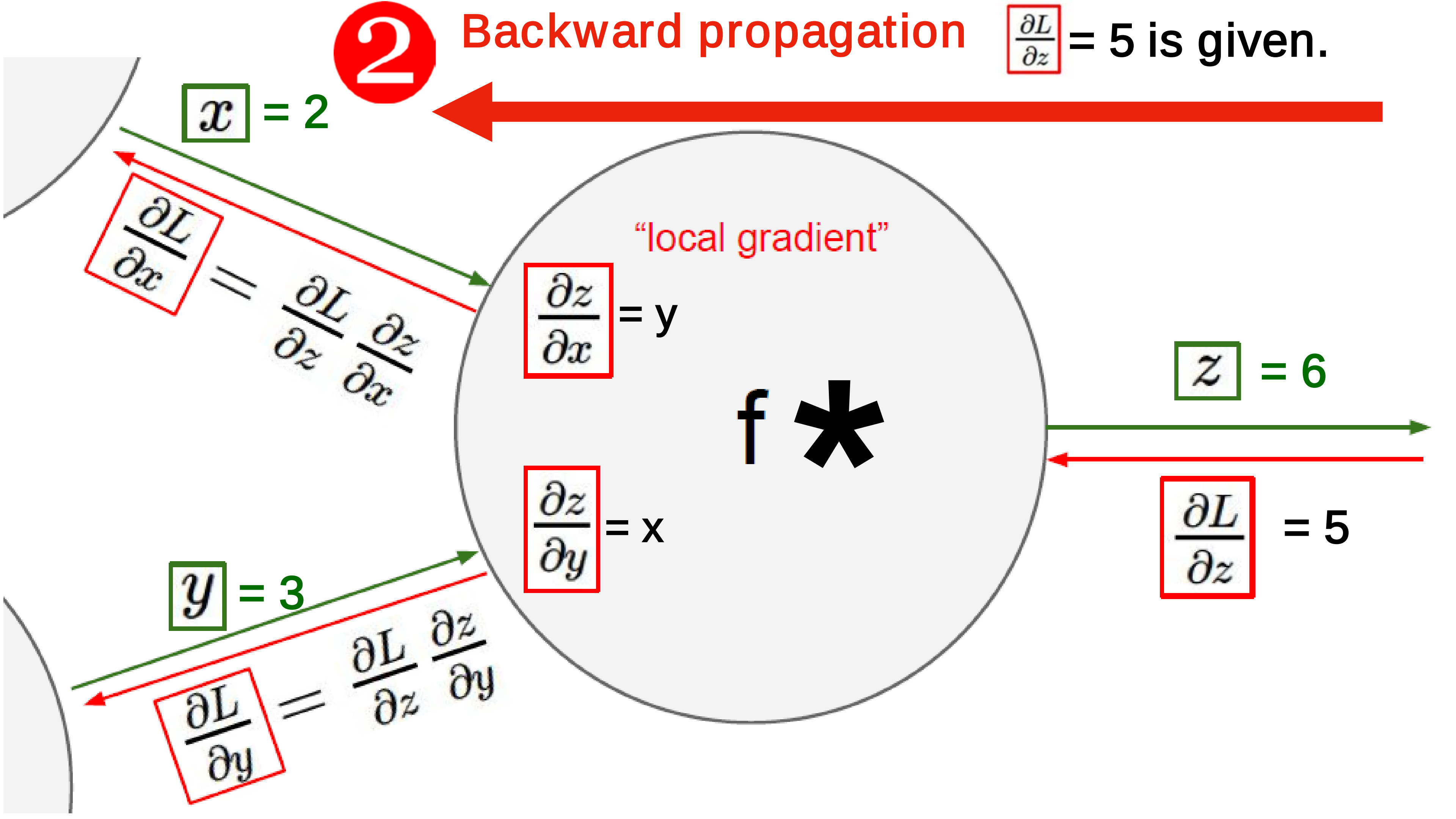
1
Forward pass x= 2, y = 3
x
y
z
f *



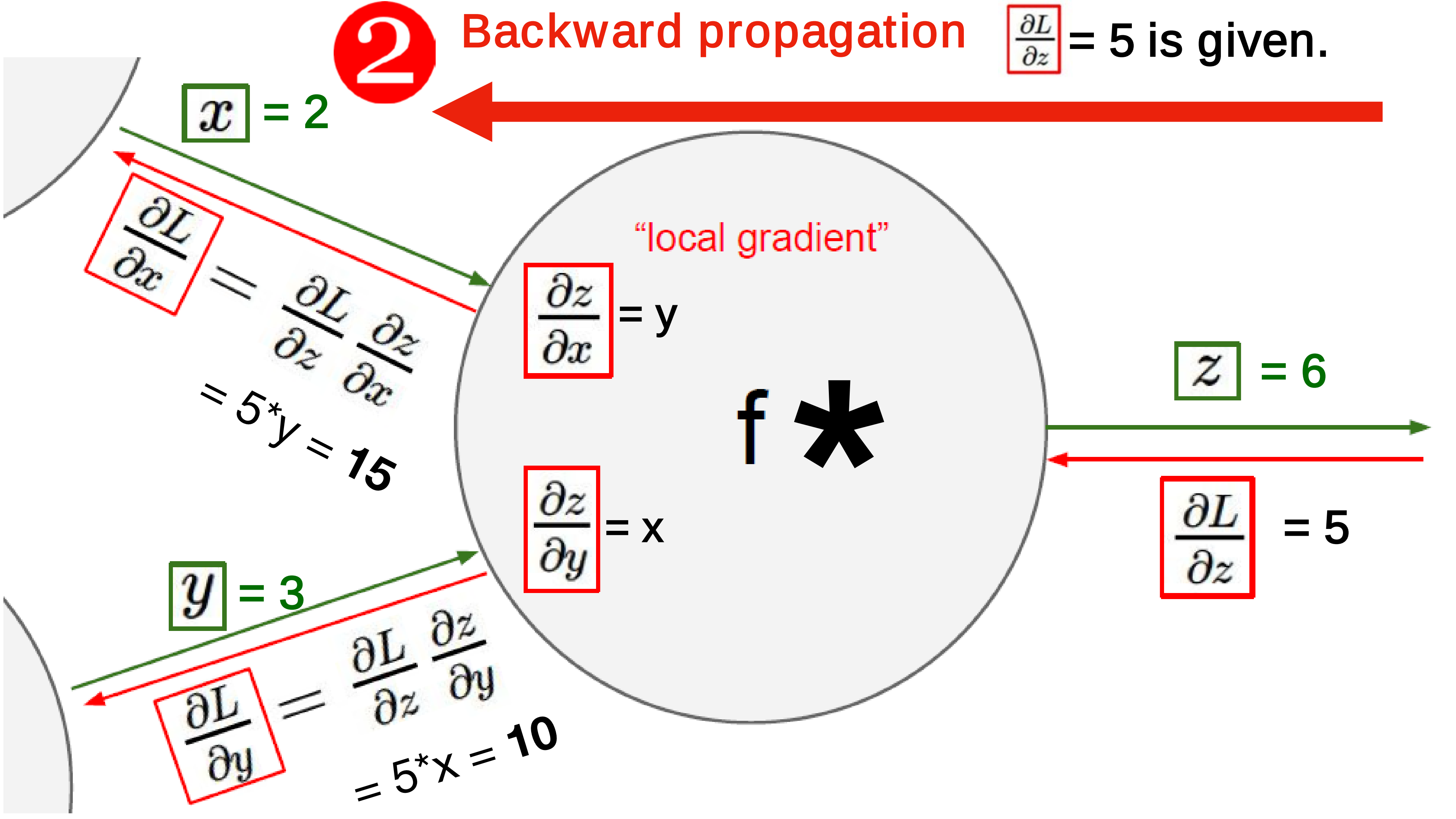
$\boxed{x}$ = 2
$\boxed{y}$ = 3
"local gradient"
$\frac{\partial z}{\partial x} = y$
$\frac{\partial z}{\partial y} = x$
f *
f=xy
$\boxed{z}$ = 6



2 Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given.
x = 2
"local gradient"
$\frac{\partial z}{\partial x}$
f *
z = 6
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$ = 5
y = 3



2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given.
x = 2
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$ = y
f *
z = 6
$\frac{\partial z}{\partial y}$ = x
$\frac{\partial L}{\partial z}$ = 5
y = 3
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$



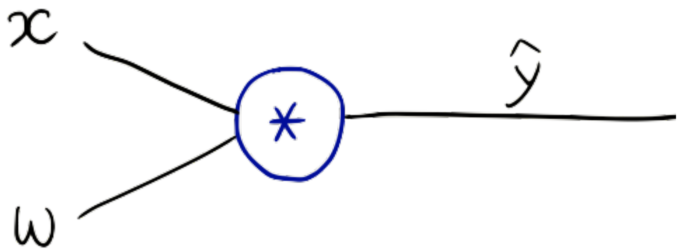
2
Backward propagation
$\frac{\partial L}{\partial z}$ = 5 is given.
x = 2
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
= 5*y = 15
"local gradient"
$\frac{\partial z}{\partial x}$ = y
$\frac{\partial z}{\partial y}$ = x
f *
z = 6
$\frac{\partial L}{\partial z}$ = 5
y = 3
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
= 5*x = 10

Computational graph

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

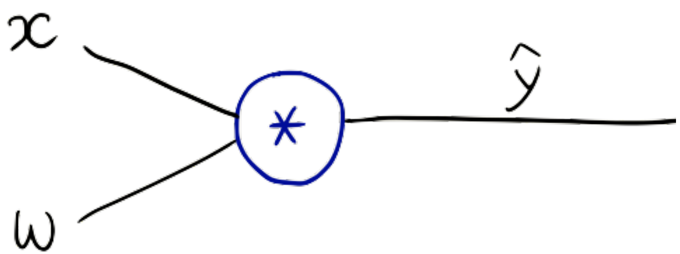
Computational graph

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



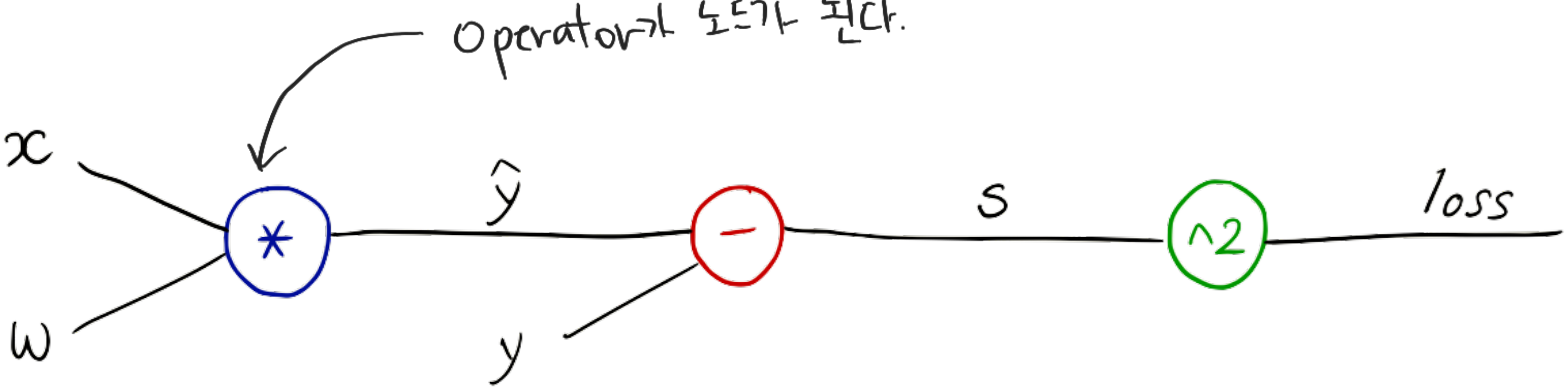
Computational graph

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

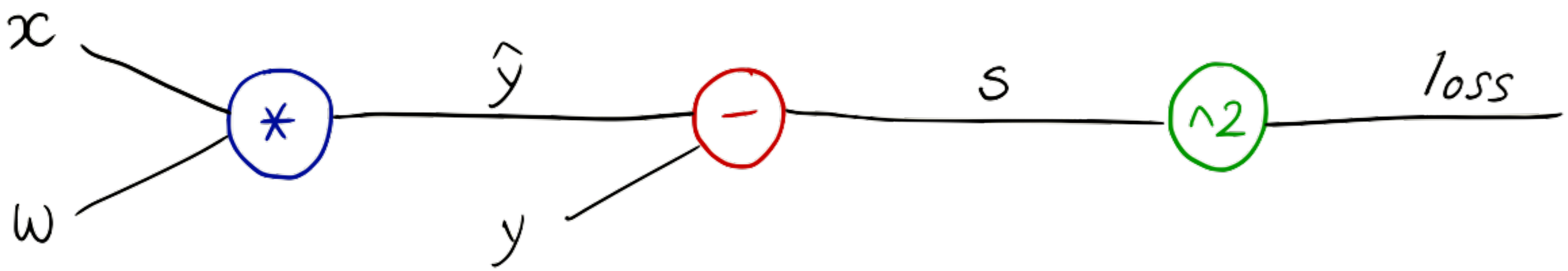


Computational graph

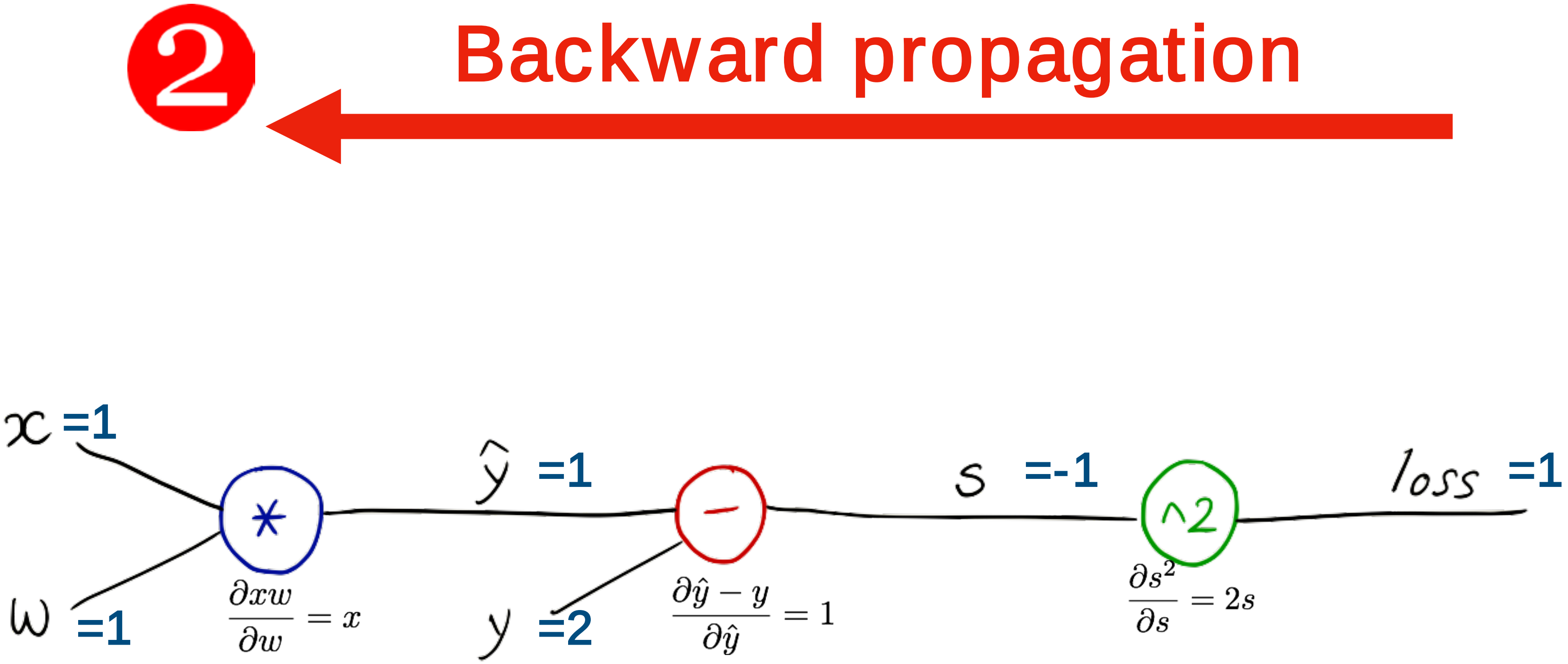
$$\hat{y} = x * w \qquad loss = (\hat{y} - y)^2 = (x * w - y)^2$$



1 Forward pass x=1, y = 2 where w=1



Backward propagation



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

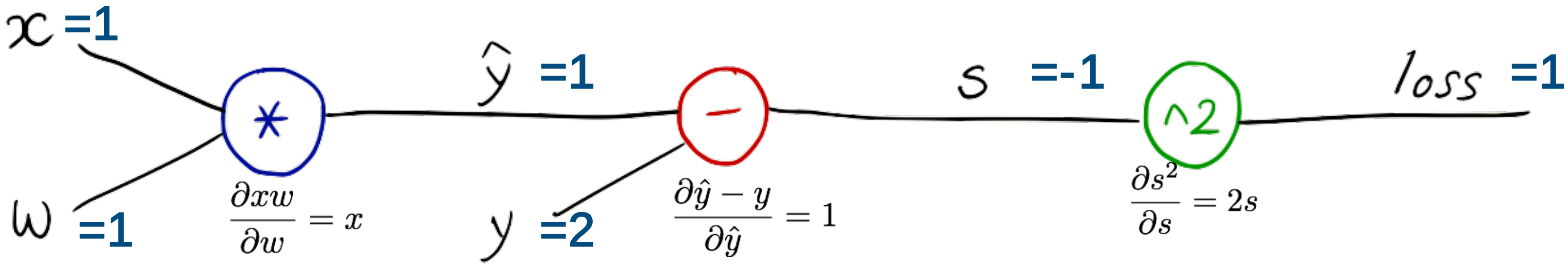
2

Backward propagation

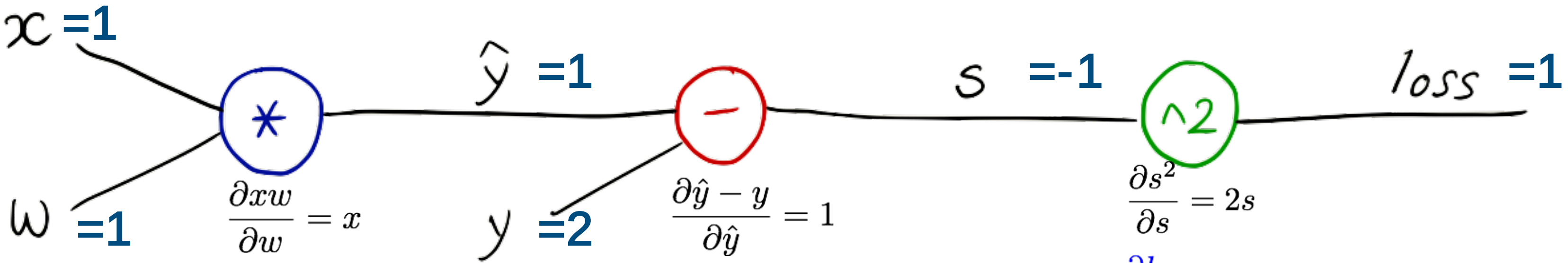
$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial s} = 2s = -2$$



Backward propagation



$$\frac{\partial xw}{\partial w} = x$$

$$\frac{\partial \hat{y} - y}{\partial \hat{y}} = 1$$

$$\frac{\partial s^2}{\partial s} = 2s$$

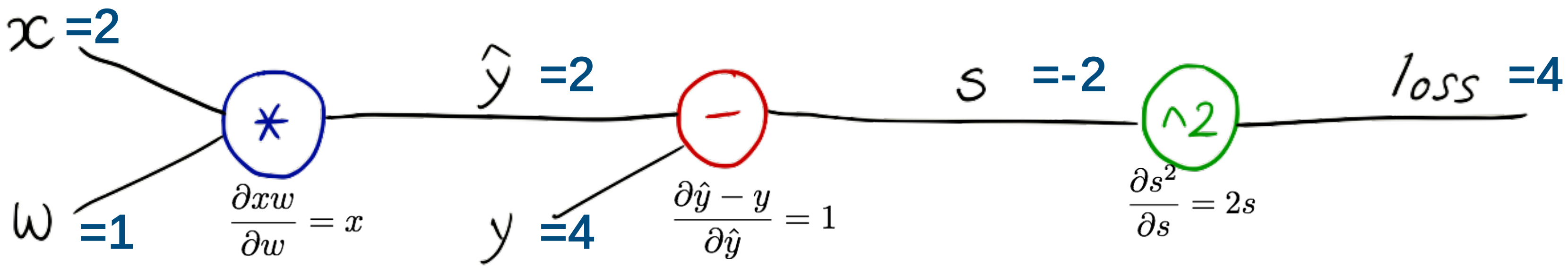
$$\frac{\partial loss}{\partial s} = 2s = -2$$

$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

```
    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
```

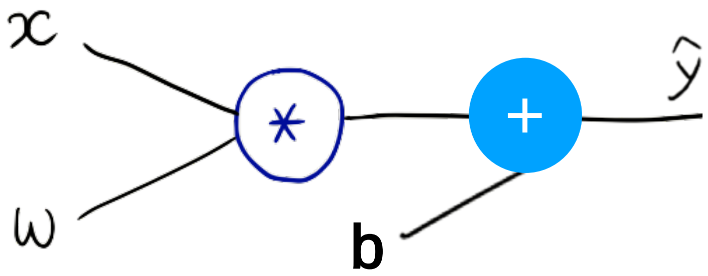
Exercise 4-1: x = 2, y=4, w=1



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

Exercise 4-2: x = 1, y=2, *w=1, b=2*

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



Data and Variable

```python
import torch
import numpy as np
import matplotlib.pyplot as plt

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

w = torch.tensor([1.0], requires_grad=True)
```

↳ auto-grad 활용 가능

http://pytorch.org/docs/master/notes/autograd.html?highlight=variable

Model and Loss

```
# our model forward pass
def forward(x):
    return x * w

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

Training: forward, backward, and update weight

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

batch mode

```python
for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        y_pred = forward(x_val) # 1) Forward pass
        l = loss(y_pred, y_val) # 2) Compute loss
        l.backward() # 3) Back propagation to update weights
```

gradient 자동 계산

```python
        print("\tgrad: ", x_val, y_val, w.grad.item())
        w.data = w.data - 0.01 * w.grad.item()

        # Manually zero the gradients after updating weights
        w.grad.data.zero_()
```

누적 방식으로 계산하기 때문에 초기화가 필요

```python
    print(f"Epoch: {epoch} | Loss: {l.item()}")

# After training
print("Prediction (after training)", 4, forward(4).item())
```

Output

```
for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        y_pred = forward(x_val) # 1) Forward pass
        l = loss(y_pred, y_val) # 2) Compute loss
        l.backward() # 3) Back propagation to update weights
        print("₩tgrad: ", x_val, y_val, w.grad.item())
        w.data = w.data - 0.01 * w.grad.item()

        # Manually zero the gradients after updating weights
        w.grad.data.zero_()

    print(f"Epoch: {epoch} | Loss: {l.item()}")
```

```
predict (before training) 4 4.0
	grad:  1.0 2.0 -2.0
	grad:  2.0 4.0 -7.840000152587891
	grad:  3.0 6.0 -16.228801727294922
progress: 0 7.315943717956543
	grad:  1.0 2.0 -1.478623867034912
	grad:  2.0 4.0 -5.796205520629883
	grad:  3.0 6.0 -11.998146057128906
progress: 1 3.9987640380859375
	grad:  1.0 2.0 -1.0931644439697266
	grad:  2.0 4.0 -4.285204887390137
	grad:  3.0 6.0 -8.870372772216797
progress: 2 2.1856532096862793
	grad:  1.0 2.0 -0.8081896305084229
	grad:  2.0 4.0 -3.1681032180786133
	grad:  3.0 6.0 -6.557973861694336
progress: 3 1.1946394443511963
	grad:  1.0 2.0 -0.5975041389465332
	grad:  2.0 4.0 -2.3422164916992188
	grad:  3.0 6.0 -4.848389625549316
progress: 4 0.6529689431190491
	grad:  1.0 2.0 -0.4417421817779541
	grad:  2.0 4.0 -1.7316293716430664
	grad:  3.0 6.0 -3.58447265625
progress: 5 0.35690122842788696
	grad:  1.0 2.0 -0.3265852928161621
	grad:  2.0 4.0 -1.2802143096923828
	grad:  3.0 6.0 -2.650045394897461
progress: 6 0.195076122879982
	grad:  1.0 2.0 -0.24144840240478516
	grad:  2.0 4.0 -0.9464778900146484
	grad:  3.0 6.0 -1.9592113494873047
progress: 7 0.10662525147199631
	grad:  1.0 2.0 -0.17850565910339355
	grad:  2.0 4.0 -0.699742317199707
	grad:  3.0 6.0 -1.4484672546386719
```

Output

(from numeric gradient 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))
```

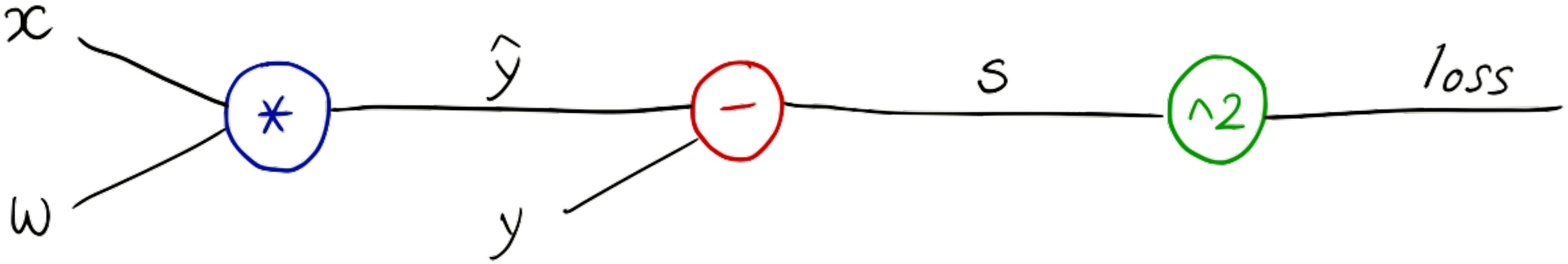
Output
(from numeric gradient computation)

```
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 -2.342216760409350
    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
```

Output
(computational graph)

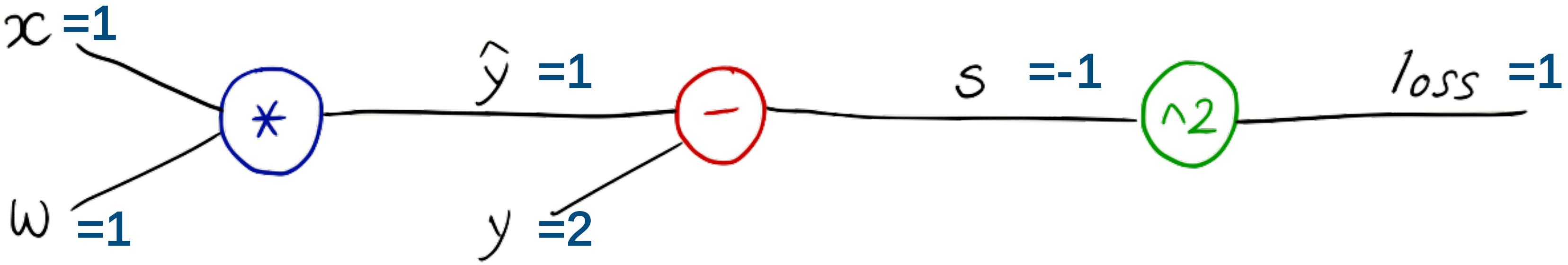

```
predict (before training) 4 4.0
    grad:  1.0 2.0 -2.0
    grad:  2.0 4.0 -7.84
    grad:  3.0 6.0 -16.2288
progress: 0 4.919240100095999
    grad:  1.0 2.0 -1.478624
    grad:  2.0 4.0 -5.796206079999999
    grad:  3.0 6.0 -11.998146585599997
progress: 1 2.688769240265834
    grad:  1.0 2.0 -1.093164466688
    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 -2.342216760409350
    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
```

PyTorch forward/backward



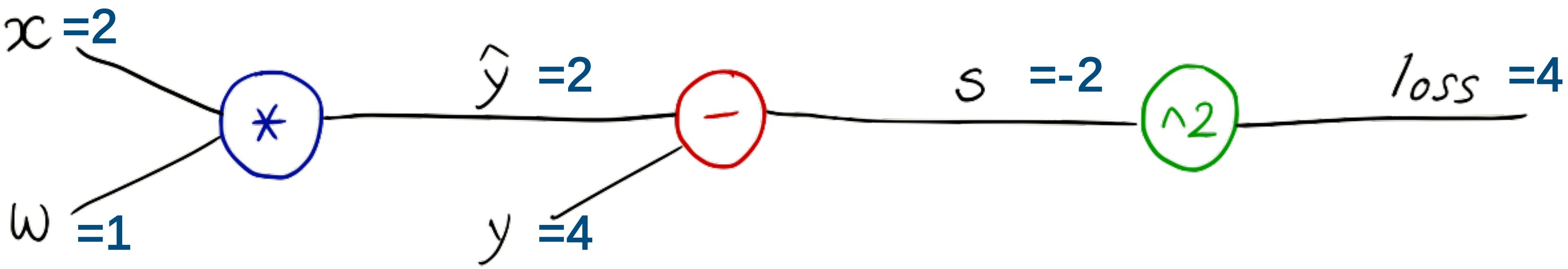
Forward pass

```
# Any random value
w = torch.tensor([1.0],  requires_grad=True)
l = loss(x=1, y=2)
```



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

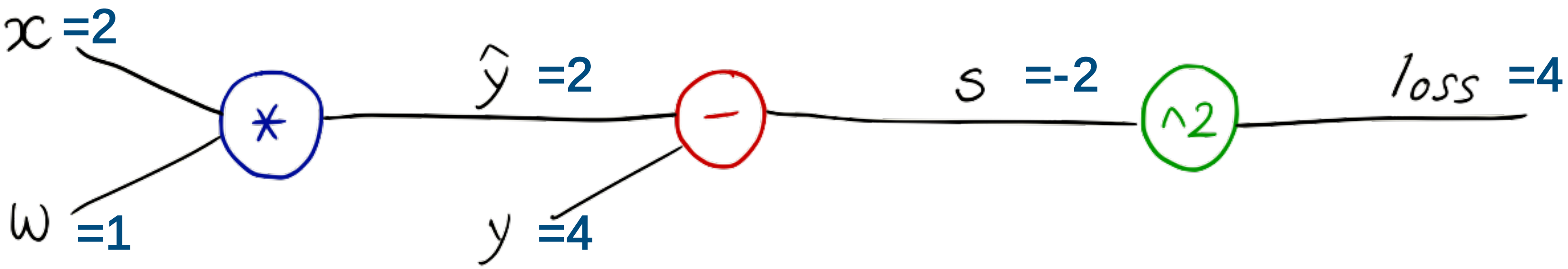
Back propagation: l.backward()



$$\frac{\partial loss}{\partial w} = \texttt{w.grad}$$

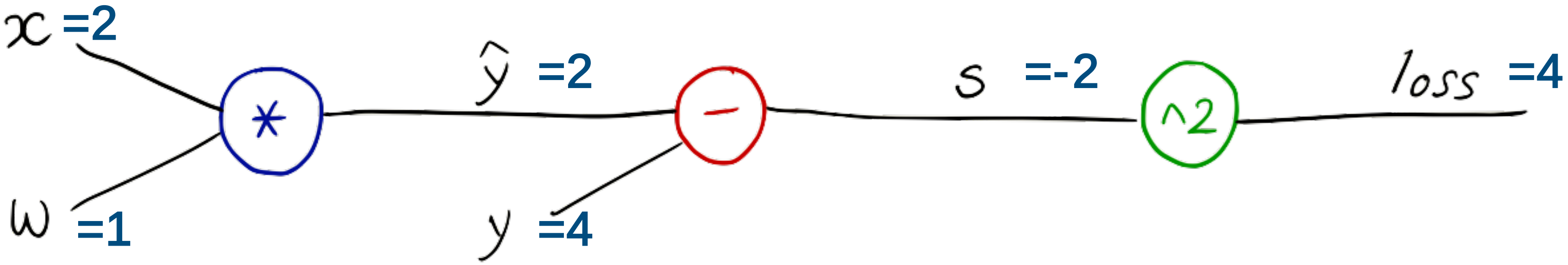
Weight update (step)

```
w.data = w.data - 0.01 * w.grad.data
```



$$\frac{\partial loss}{\partial w} = \text{w.grad}$$

Exercise 4-3: implement computational graph and backprop using NumPy



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

Exercise 4-4: Compute gradients using computational graph (manually)

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

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

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

Exercise 4-5: compute gradients using PyTorch

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

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

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

WHAT NEXT

Lecture 5: Linear regression in the PyTorch way