

AI and Deep Learning

# Linear Regression & Back-propagation

Jeju National University

Yungcheol Byun

# Agenda

- Neuron and Regression
- Loss/Error/Cost Function
- Learning and Updating Weights
- Gradient/Slope
- Computation Graph
- Forward Propagation
- Backpropagation



NAT GEO  
**WILD**

“

After spending most of the  
**ocean** life, **salmons** go back  
home(**river**) where they  
were born.



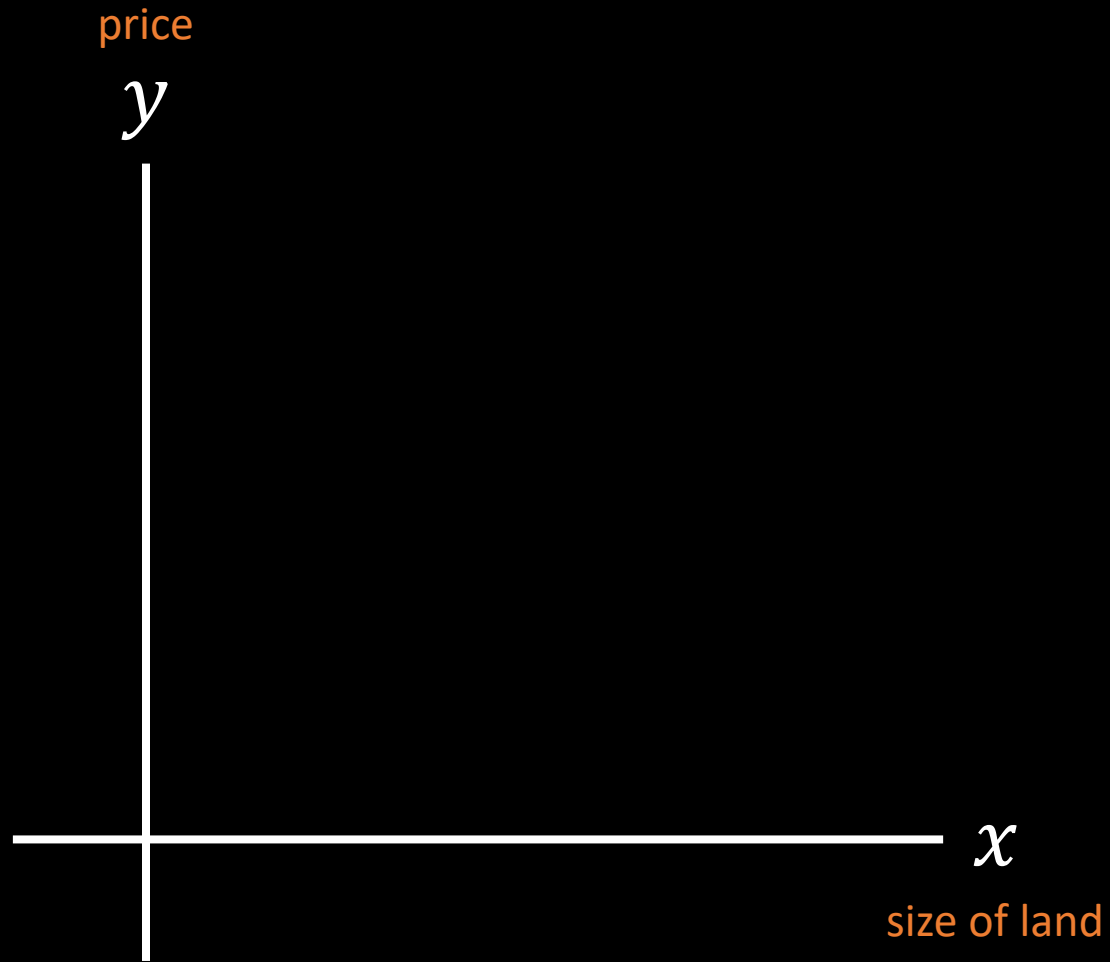
# Regression (회귀)

- Going back again.
- To describe a **natural phenomena**
- A term frequently used in anthropology(인류학) to present a **natural tendency**

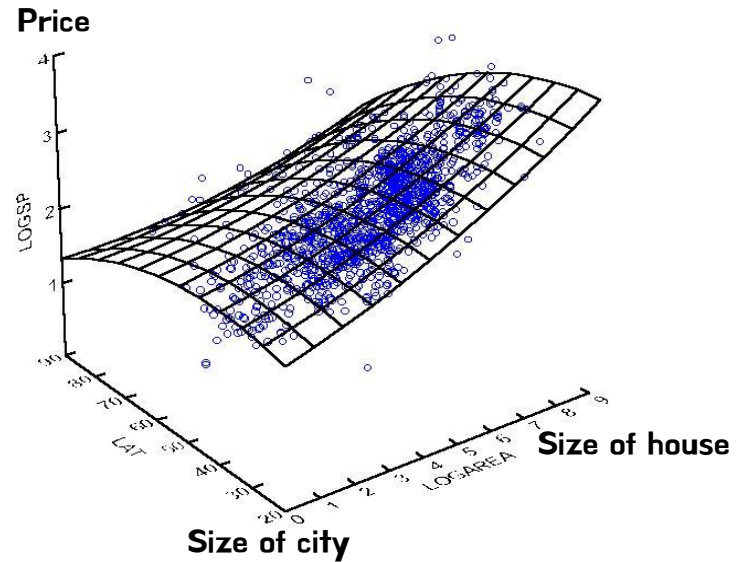
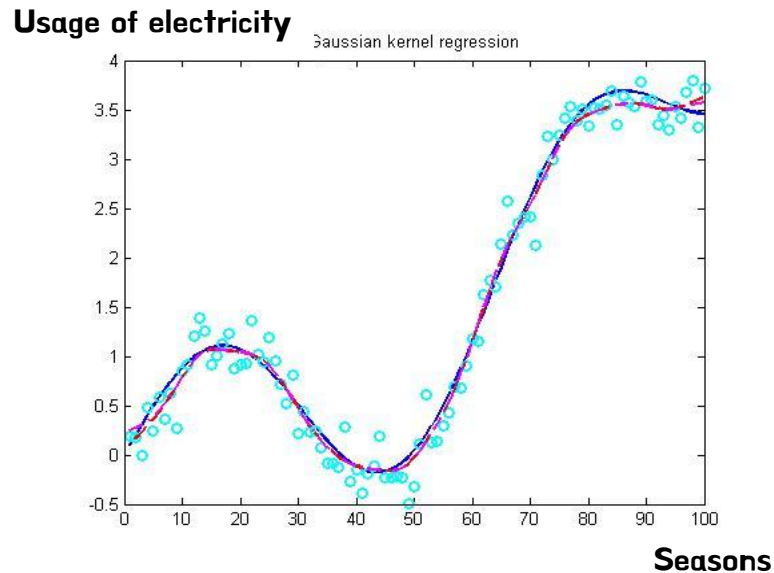
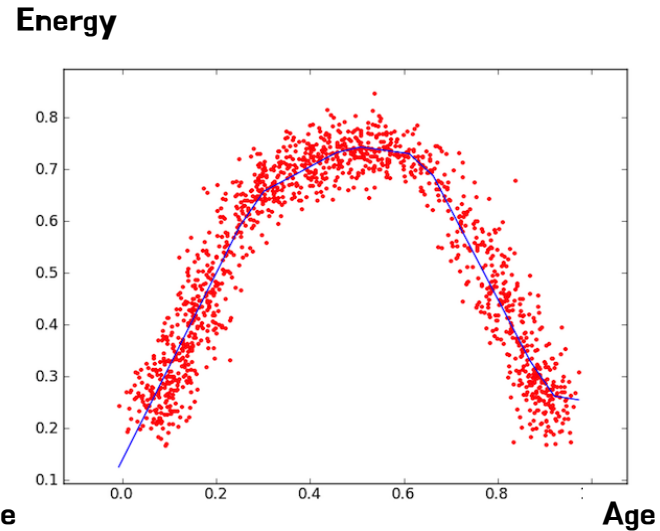
What is 'a proposed explanation for a phenomenon'?

# Regression (회귀)

- Statistical measure to determine the relationship between one dependent variable (usually denoted by  $Y$ , 종속변수) and a series of other independent variables  $X$  독립변수 .

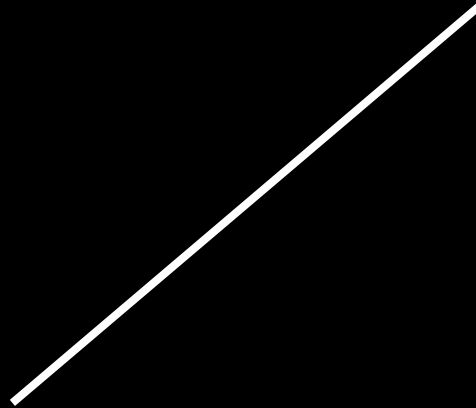


# Examples of Regression





# An example of Linear Regression



# Lab

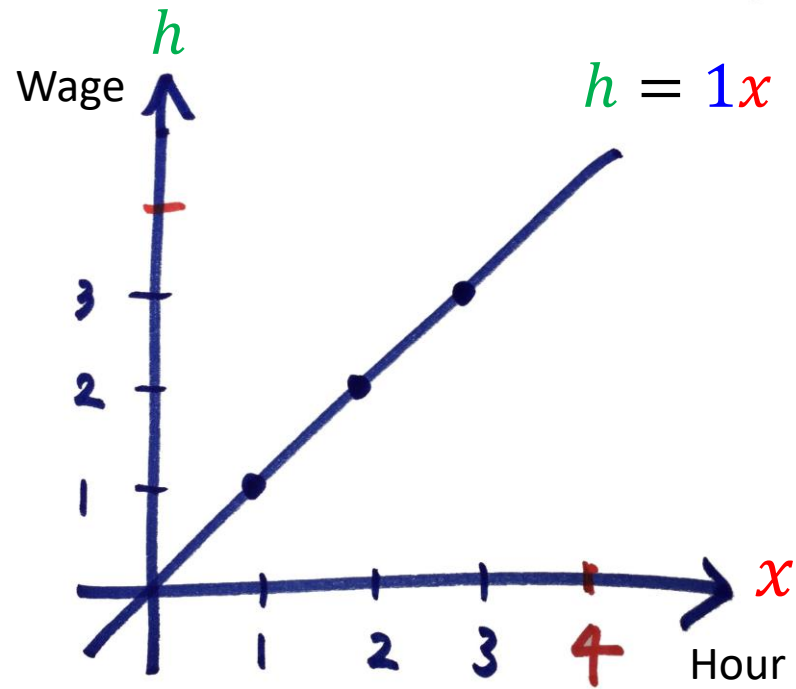
## Linear Regression

using  desmos

www.desmos.com

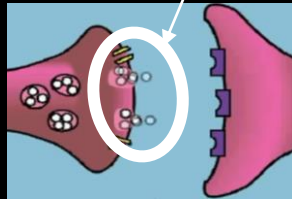
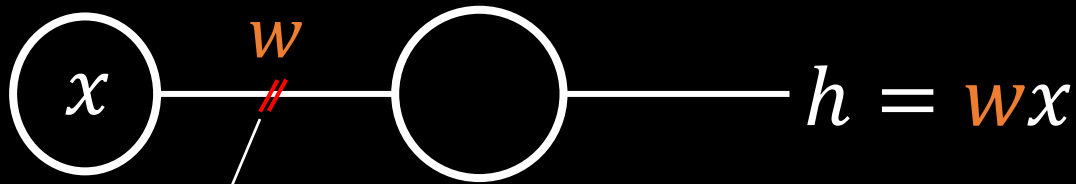
1. Draw a point(data) (1, 1)
2. Add (2, 2), (-1, -1), (-2, -2)
3.  $h = x$
4.  $h = 2x$
5.  $h = wx$  (**rotation**)
6. Move all of the points by adding 1 to y
7.  $h = wx + 1$  (**shifting**)
8.  $h = wx + b$  (**rotation** and **shifting**)

www.desmos.com



$$h = wx$$

# Neuron and regression



- $w$  : Neuro-transmitter
- if large / if small / if not exists (rotation)
- A neuron represents  
a linear regression

# Hypothesis

$$h = wx$$

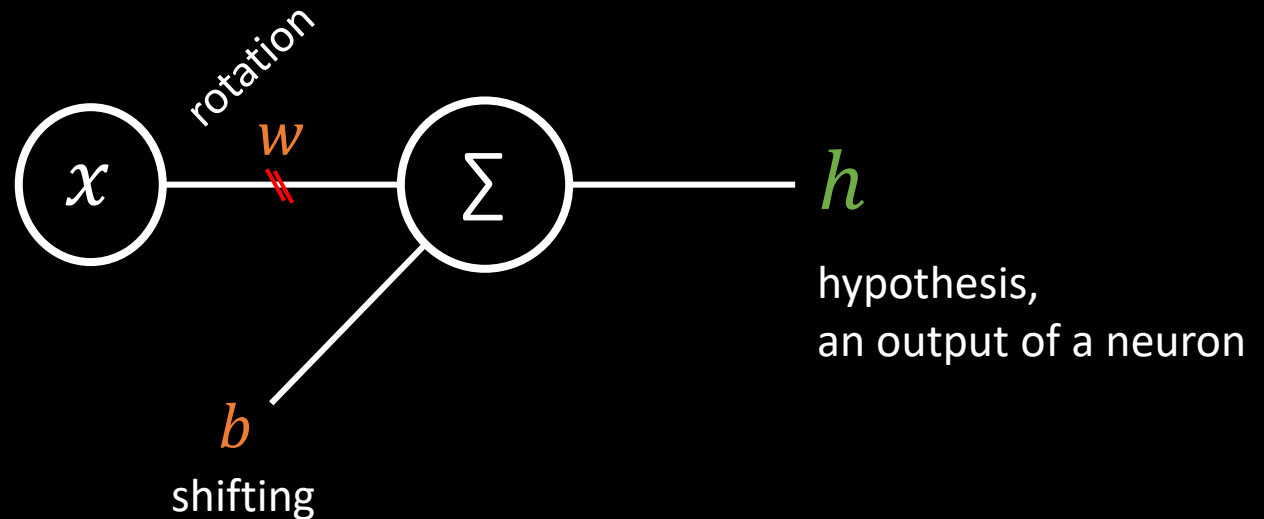
$$h = wx + b$$

- An answer by a neuron



- *h*ypothesis : a proposed explanation for a phenomenon (a regression).
- Not proved yet, but it can represents the regression after updating *w*.
- *b*? for better linear regression

# The role of $w$ and $b$



$$h = wx + b$$

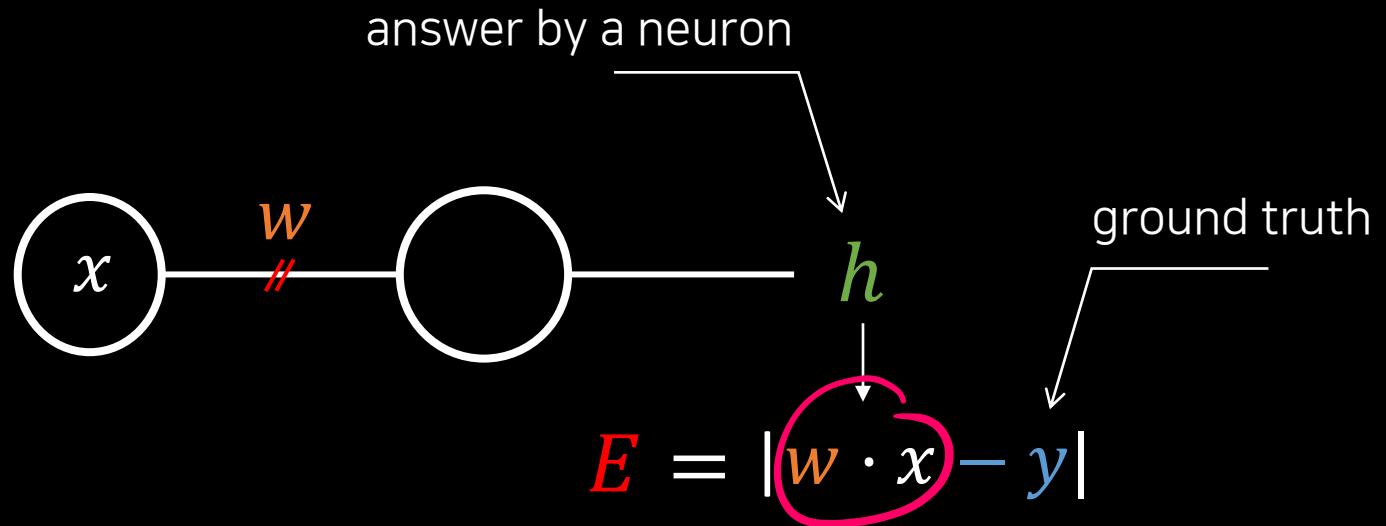


# How to learn (update $w$ )

- Scolding or blaming the neuron if it is wrong
- The neuron gets stress and automatically updates  $w$  to answer well next time so that the error(difference) decreases.
- How can we calculate the error?

# Error function

loss function  
difference function



Why absolute?

# Error function

The error is the difference between a neuron's answer and its ground truth.

$$E = |h_{\text{hypothesis}} - y|$$

$$E = |w \cdot x - y|$$

'1 hour, then 1 USD'

$x_i$	$y_i$
1	1

$$E = |w \cdot 1 - 1|$$

Supervised Learning

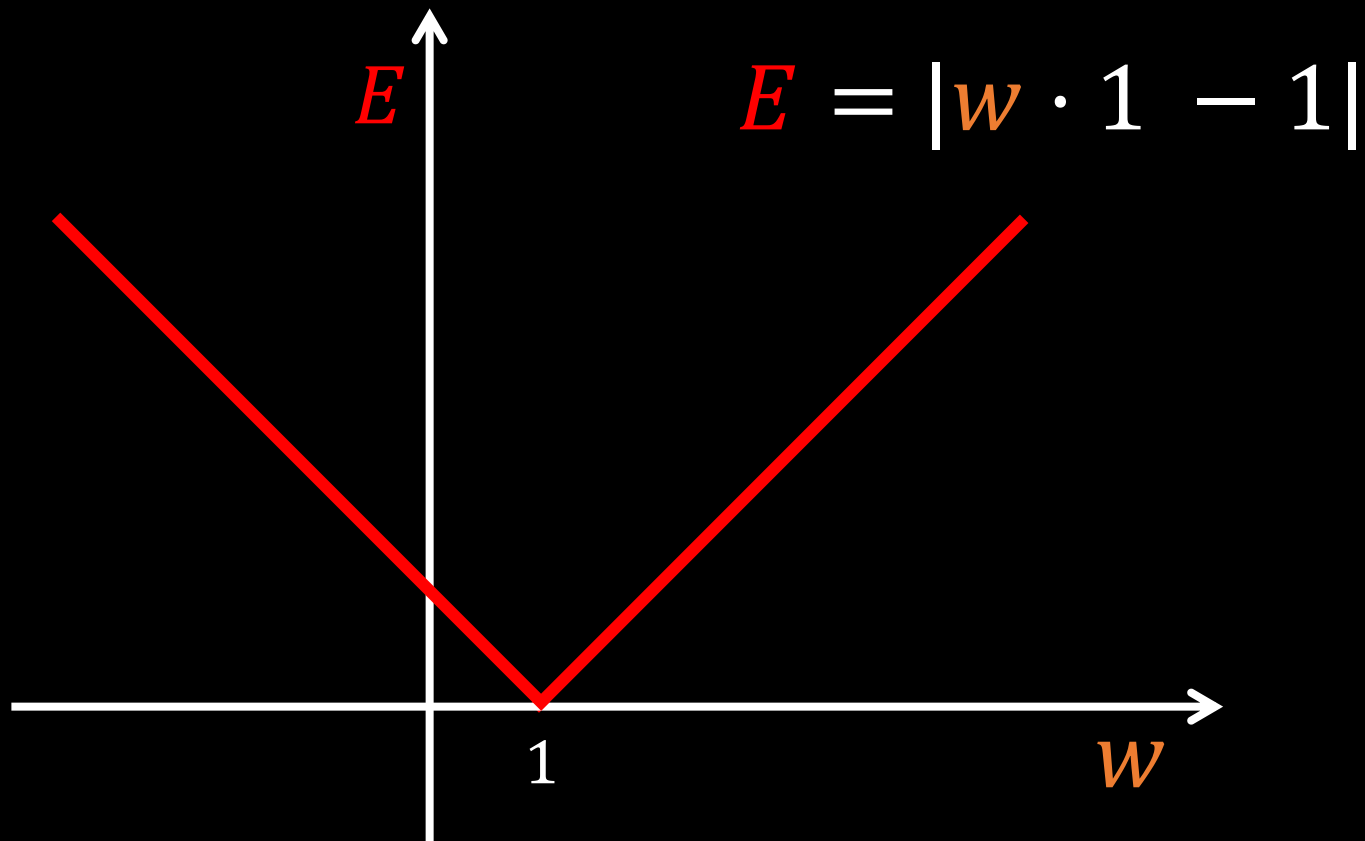
지도학습

www.desmos.com

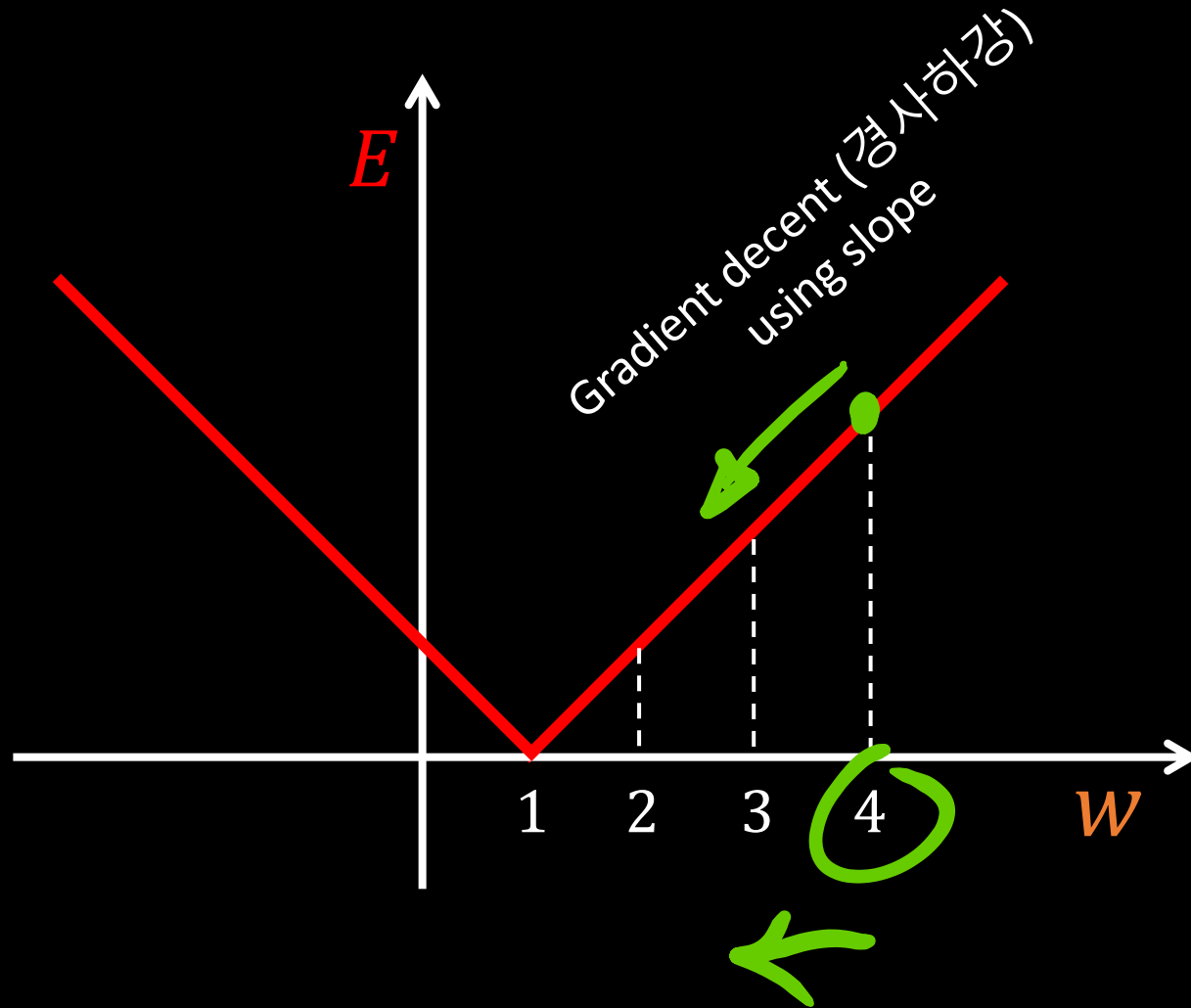
1. Mark  $(1, 1)$
2.  $h = w \cdot x$
3.  $E = w \cdot 1 - 1$
4.  $E = |w \cdot 1 - 1|$
5.  $(w, E)$



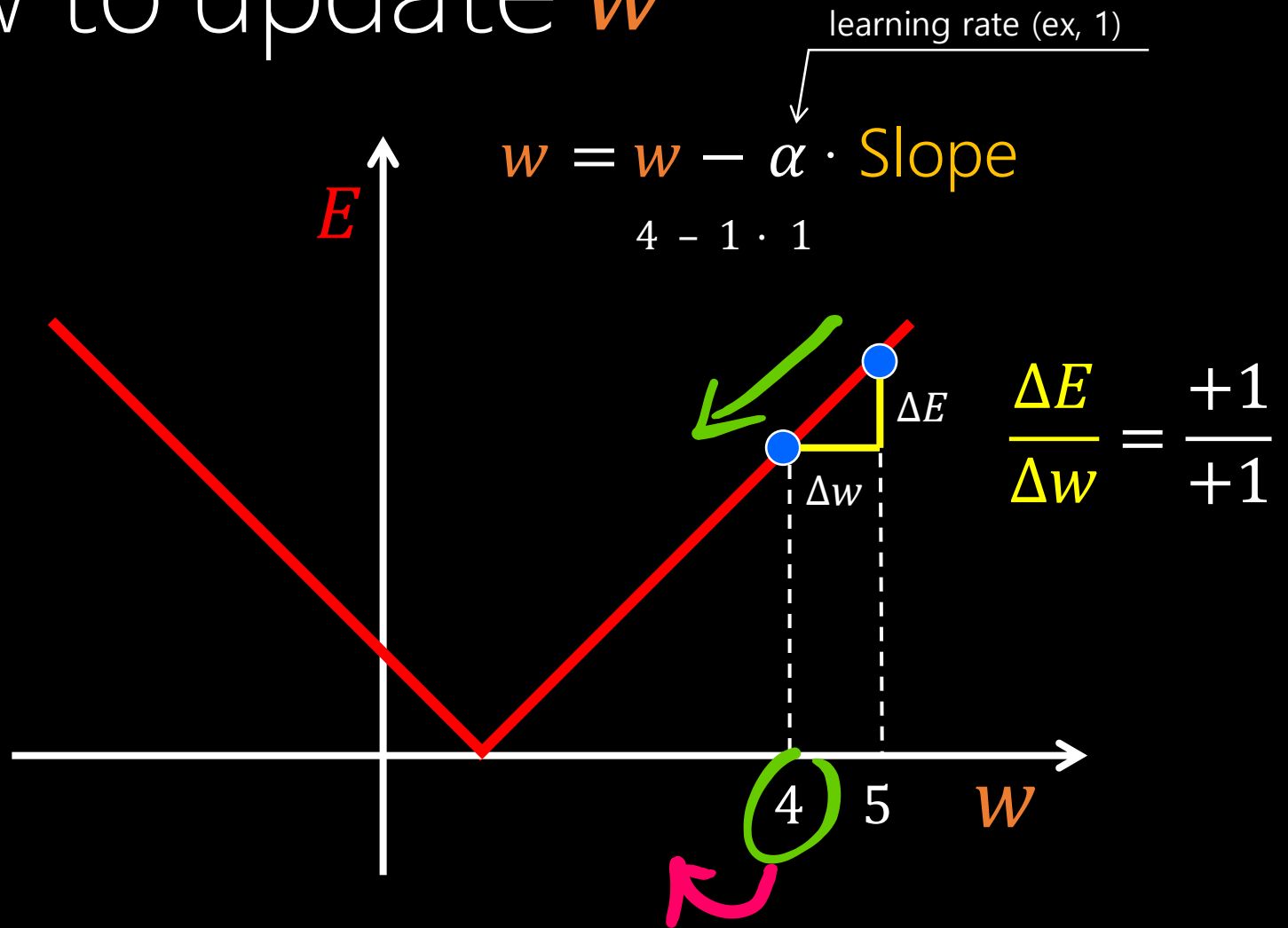
# Error Function of $w$



# How to update $w$



# How to update $w$



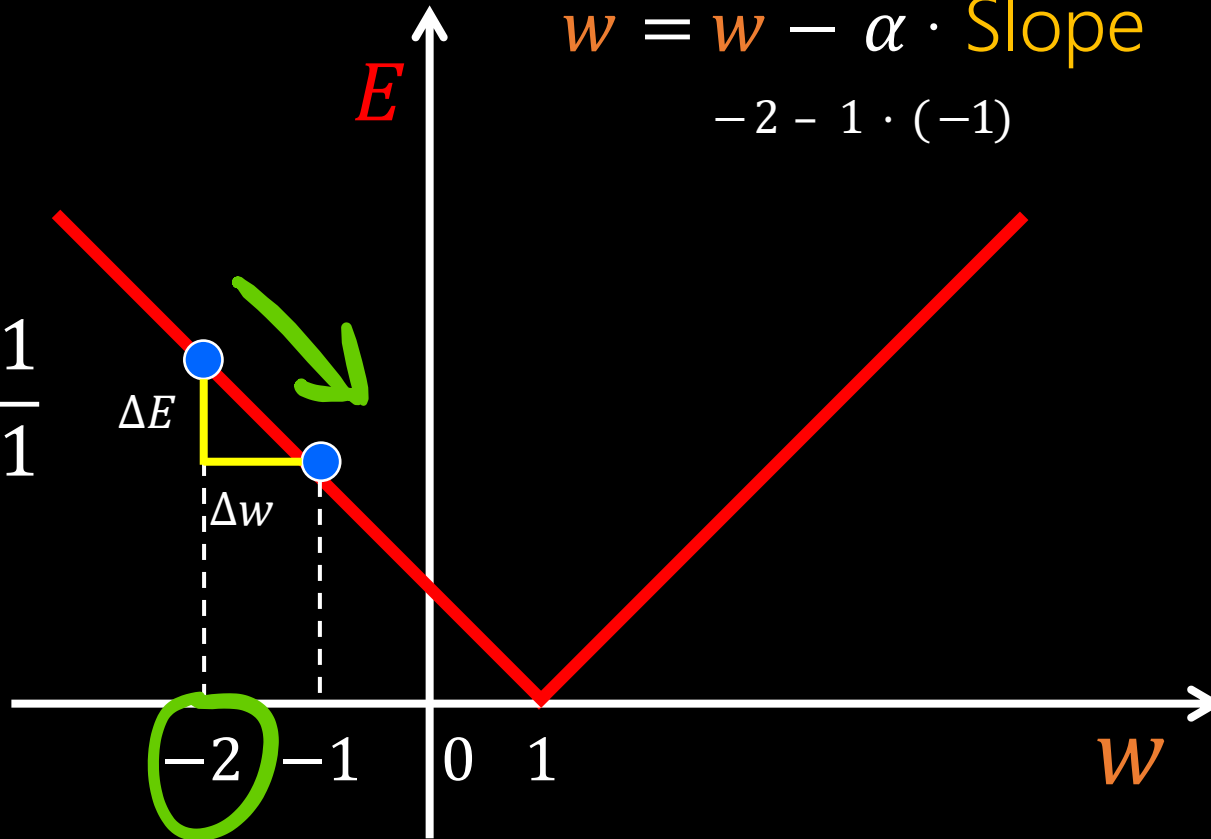


# How to update $w$

learning rate (ex, 1)

$$w = w - \alpha \cdot \text{Slope}$$
$$-2 - 1 \cdot (-1)$$

$$\frac{\Delta E}{\Delta w} = \frac{-1}{+1}$$



$$w = 4, \alpha = 1, \text{Slope} = 1$$

$$w = w - \alpha \cdot \text{Slope}$$

$$4 - 1 \cdot 1 \longrightarrow 3 \quad \text{Error } E = 2$$

$$3 - 1 \cdot 1 \longrightarrow 2 \quad \text{Error } E = 1$$

$$2 - 1 \cdot 1 \longrightarrow 1 \quad \text{Error } E = 0$$

$$w = -2, \alpha = 1, \text{Slope} = -1$$

$$w = w - \alpha \cdot \text{Slope}$$

$$-2 - 1 \cdot (-1) \rightarrow -1 \quad \text{Error } E = 2$$

$$-1 - 1 \cdot (-1) \rightarrow 0 \quad \text{Error } E = 1$$

$$0 - 1 \cdot (-1) \rightarrow 1 \quad \text{Error } E = 0$$

$$w = -2, \alpha = 2, \text{Slope} = -1$$

$$w = w - \alpha \cdot \text{Slope}$$

$$-2 - 2 \cdot (-1) \rightarrow 0 \quad \text{Error } E = 1$$

$$0 - 2 \cdot (-1) \rightarrow 2 \quad \text{Error } E = 1$$

$$2 - 2 \cdot (1) \rightarrow 0 \quad \text{Error } E = 1$$

$$0 - 2 \cdot (-1) \rightarrow 2 \quad \text{Error } E = 1$$

$$2 - 2 \cdot (1) \rightarrow 0 \quad \text{Error } E = 1$$

Absolute Error

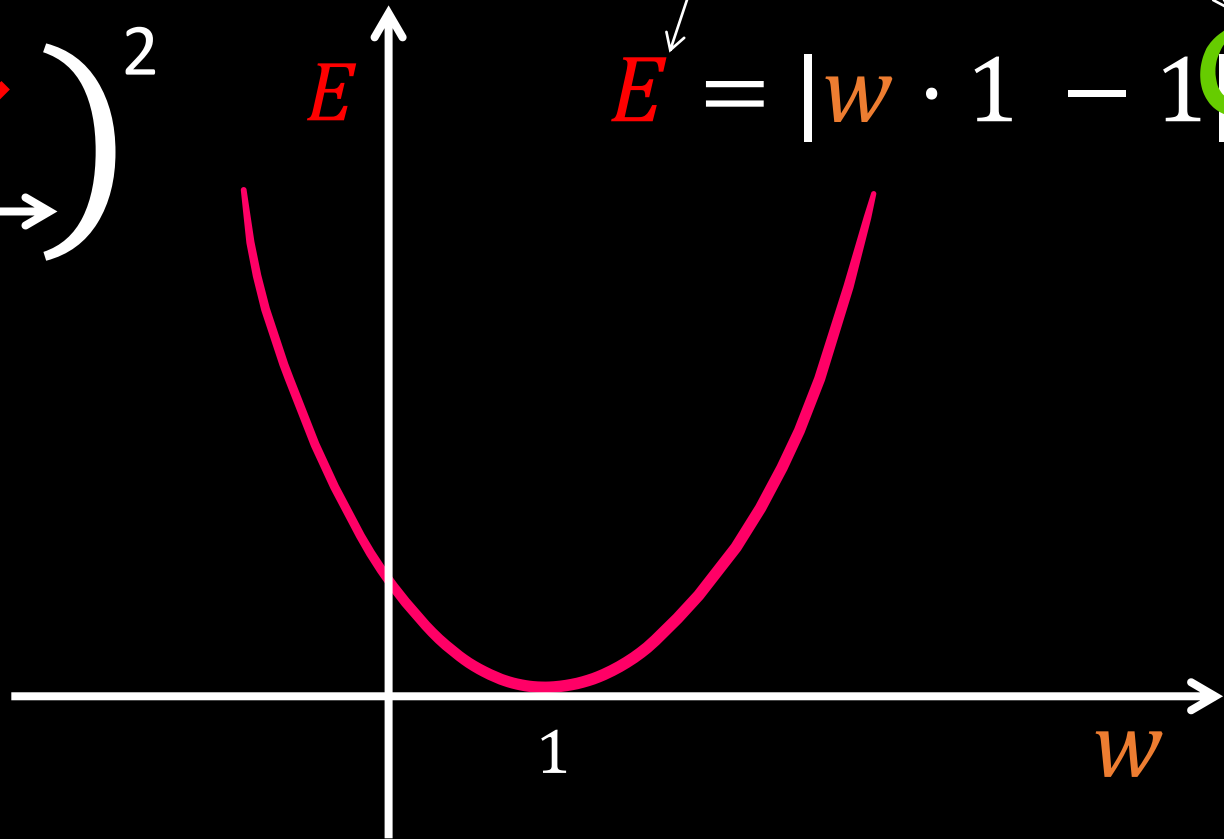
**L1 Loss function**

# Issues in the absolute error

- Always **the same slope** in the error graph regardless of the value of  $w$
- Therefore, **the same speed** in movement
- Not guarantee to get the proper value of  $w$  which gives 0 or almost 0 error.

# Square Error

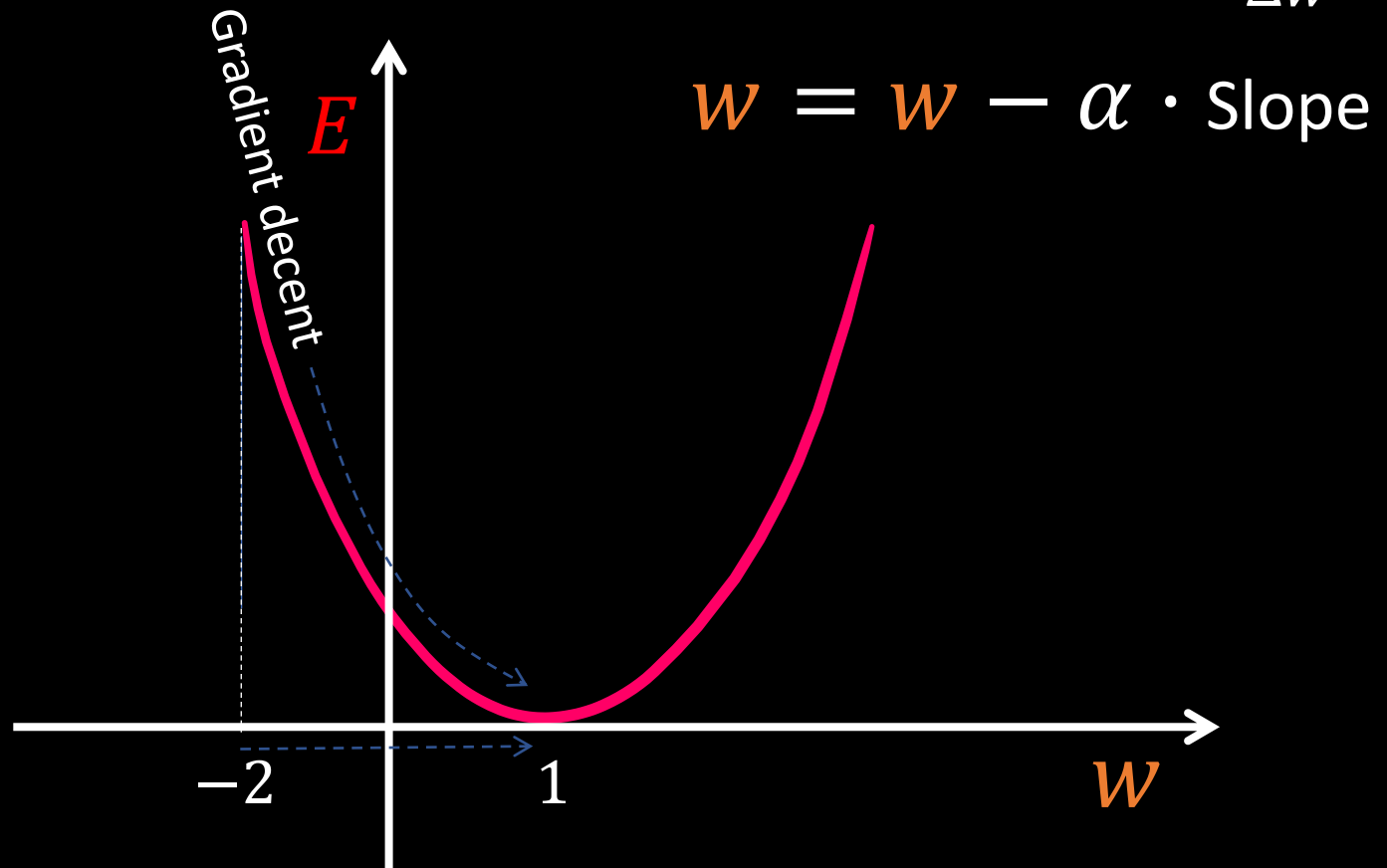
$$\left( \begin{array}{c} \swarrow \searrow \\ \downarrow \\ \swarrow \searrow \end{array} \right)^2$$



$$\overset{\text{error}}{E} = | \overset{\text{square (제곱)}}{w \cdot 1 - 1} |^2$$

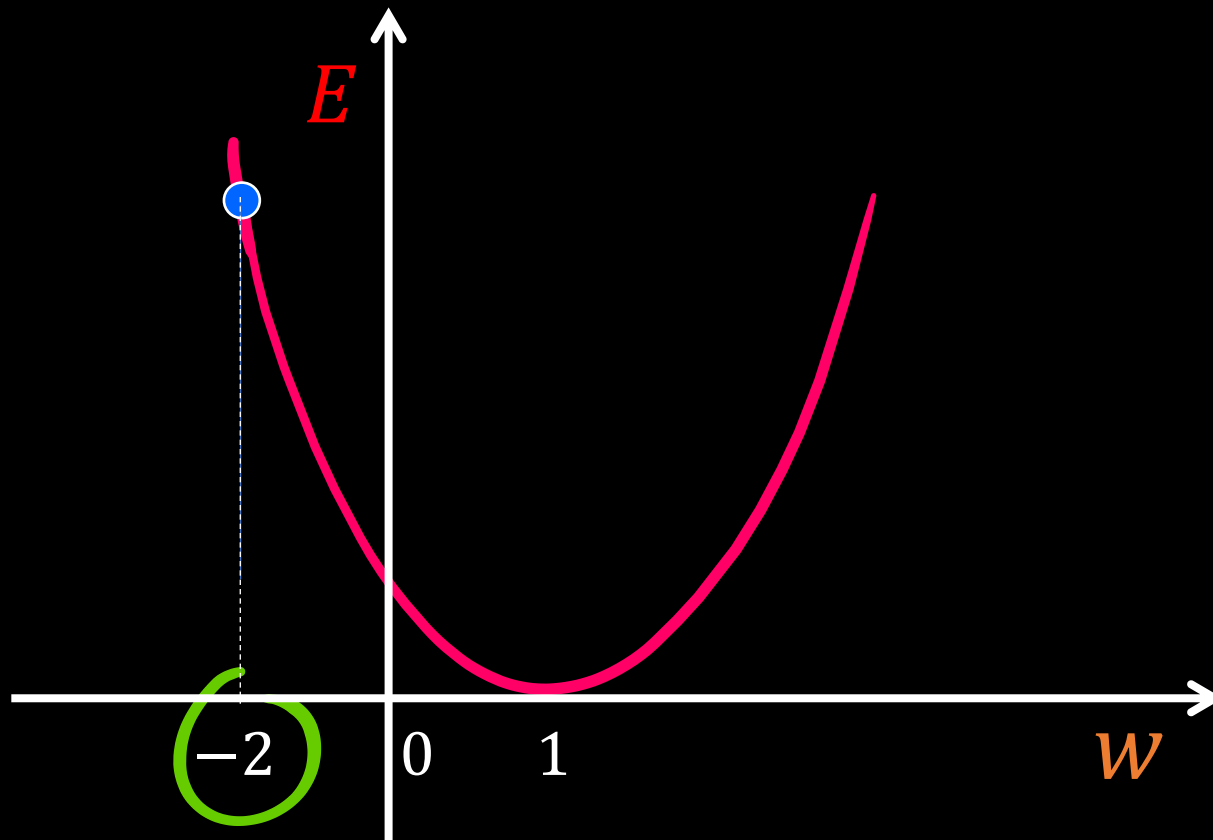
# How to update $w$

$$\frac{\Delta E}{\Delta w}$$

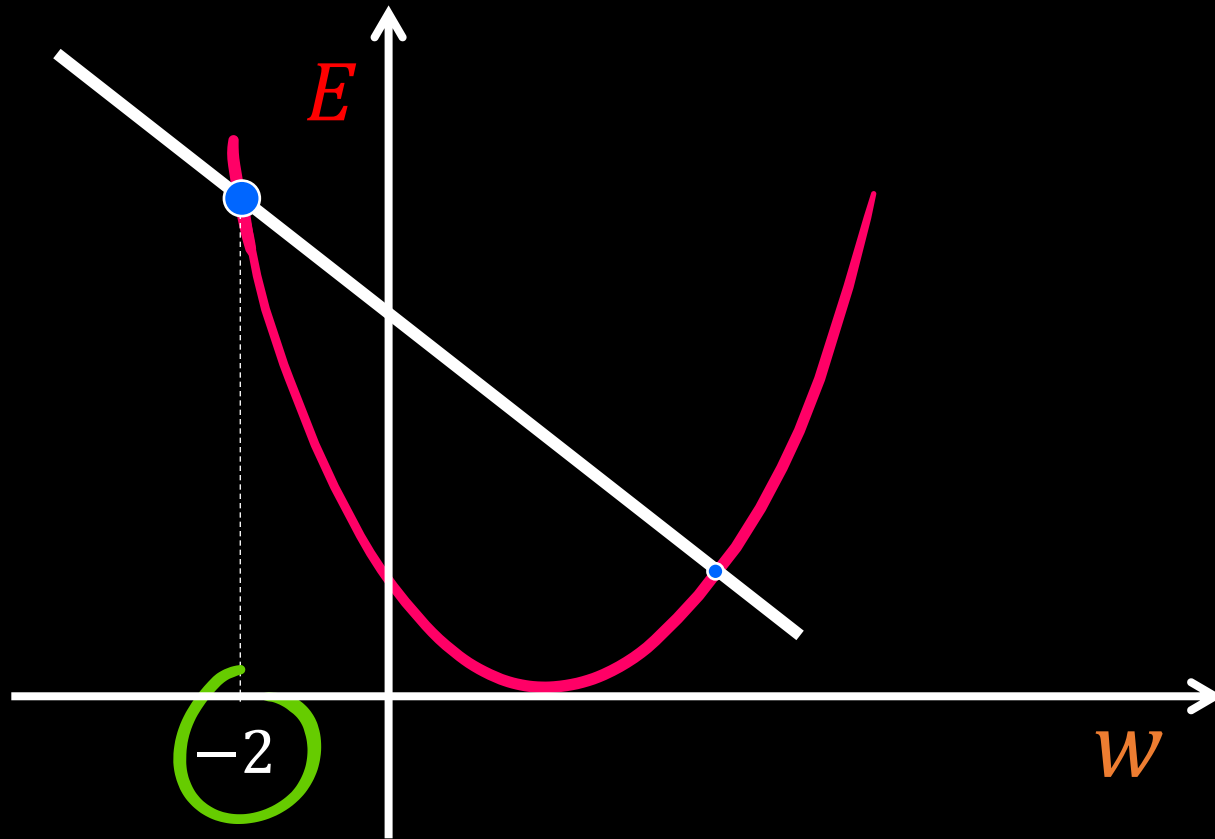




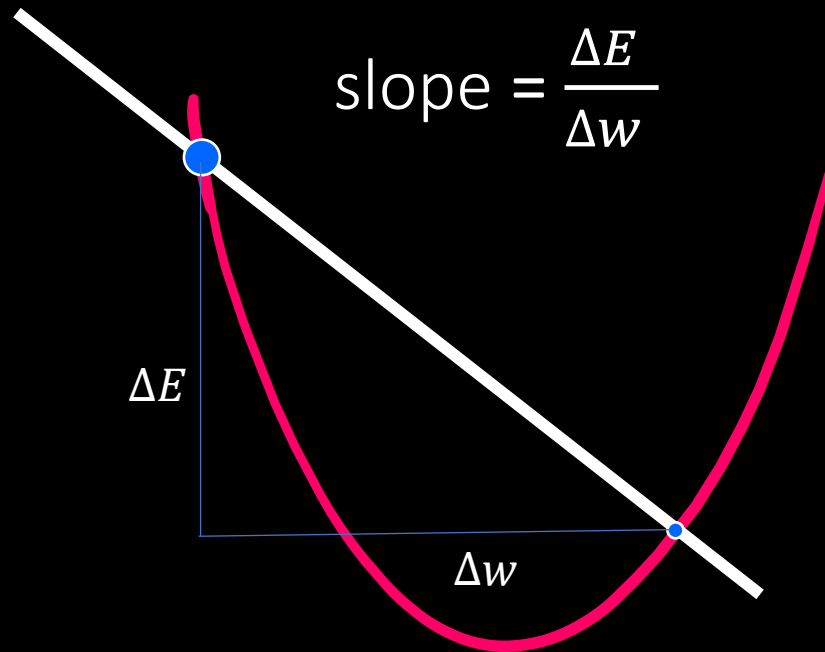
# How to update $w$



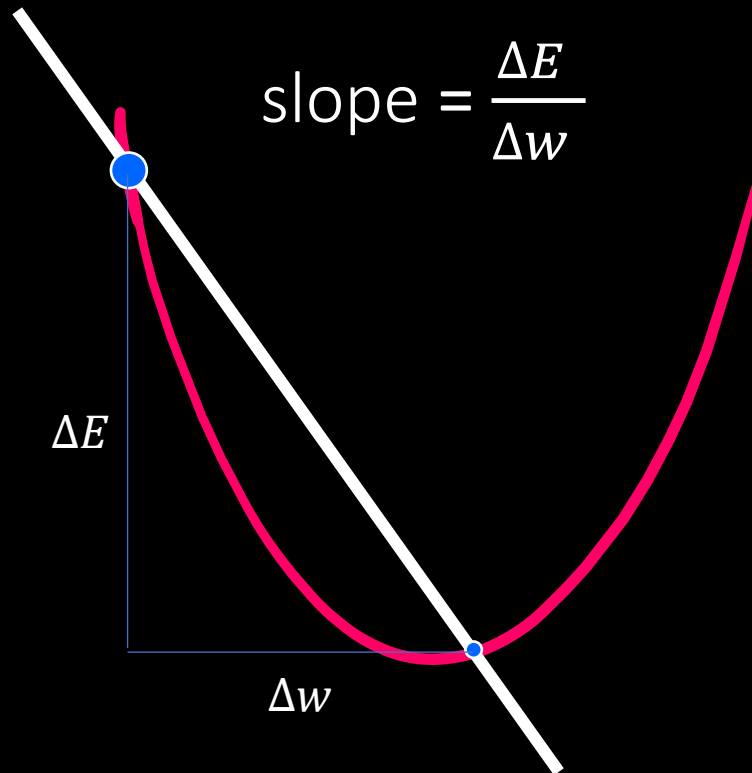
# How to update $w$



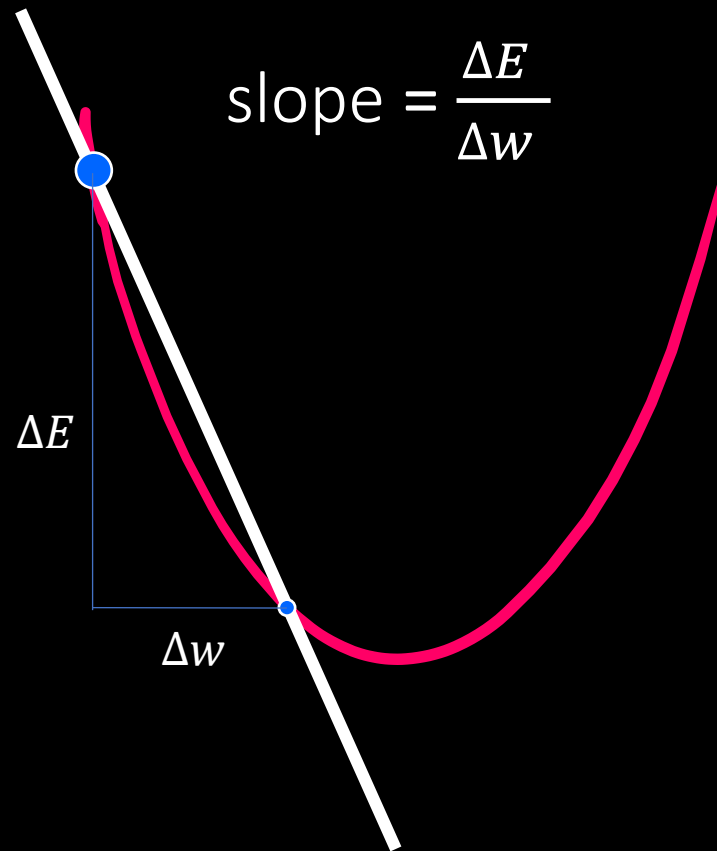
# How to update $w$



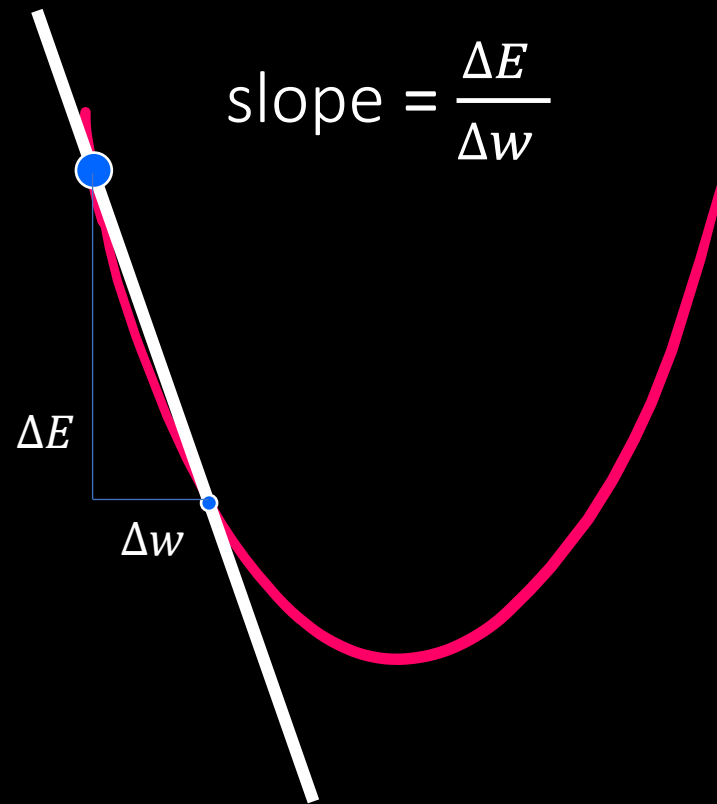
# How to update $w$



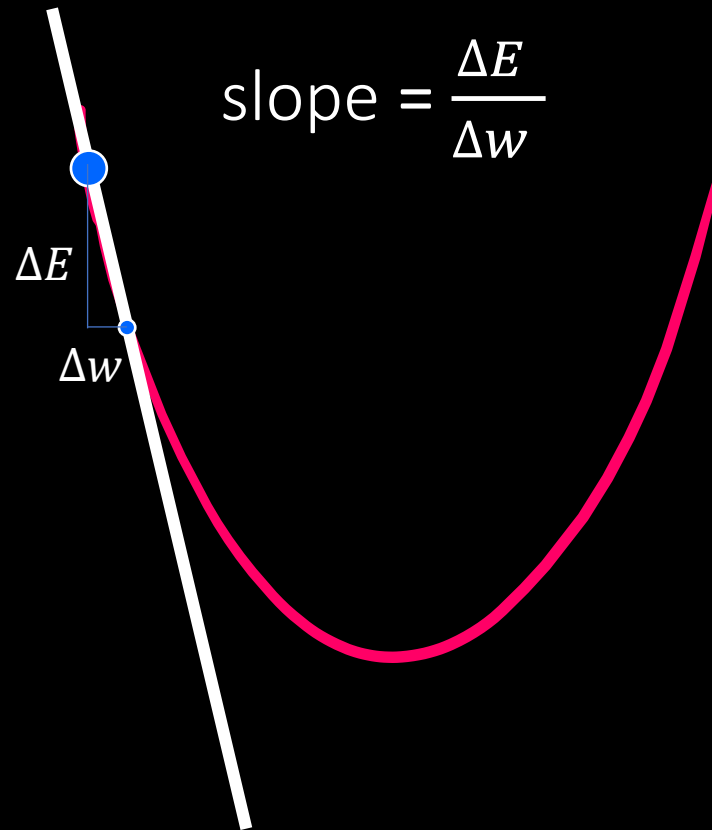
# How to update $w$



# How to update $w$

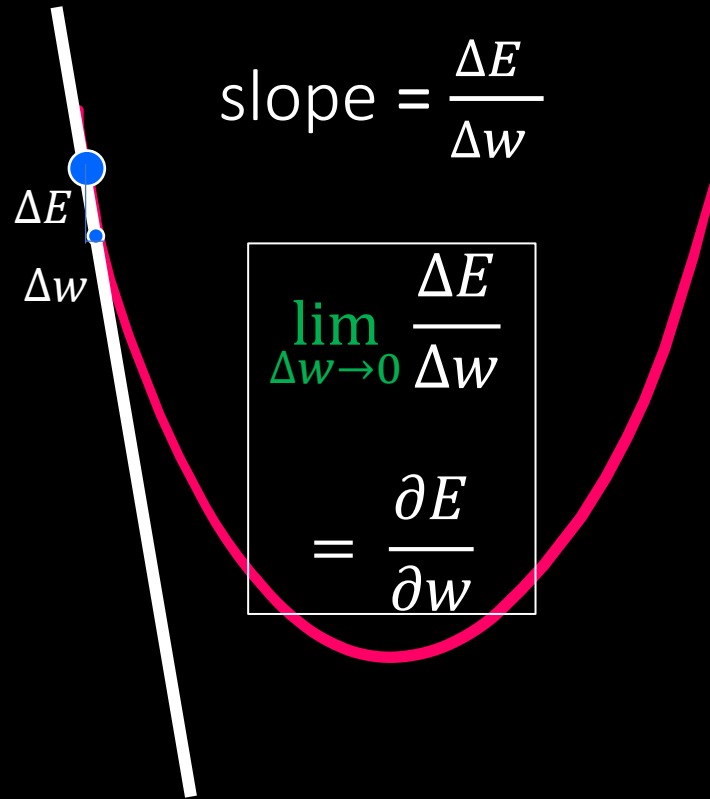


# How to update $w$



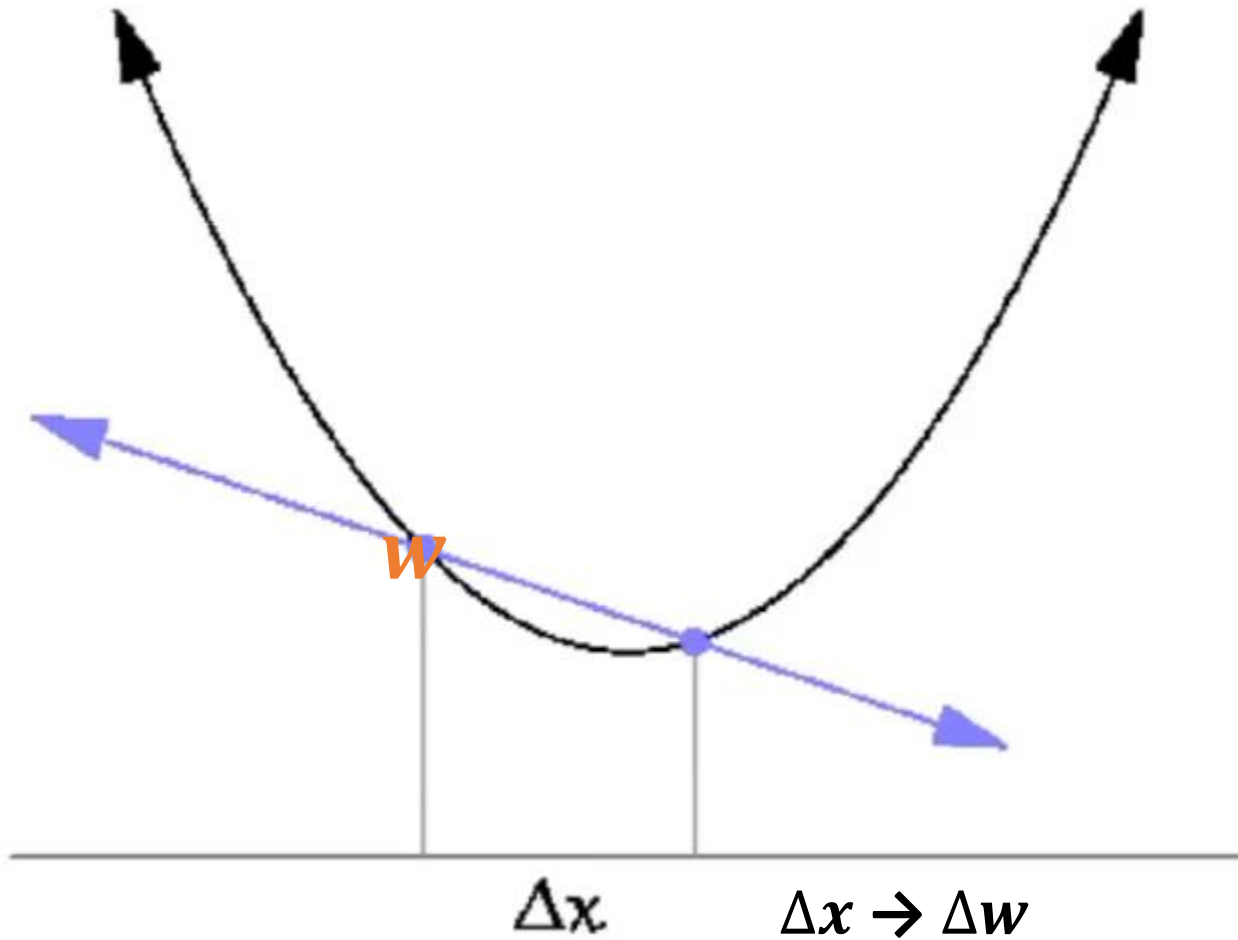
# How to update $w$

접선·Tangent line





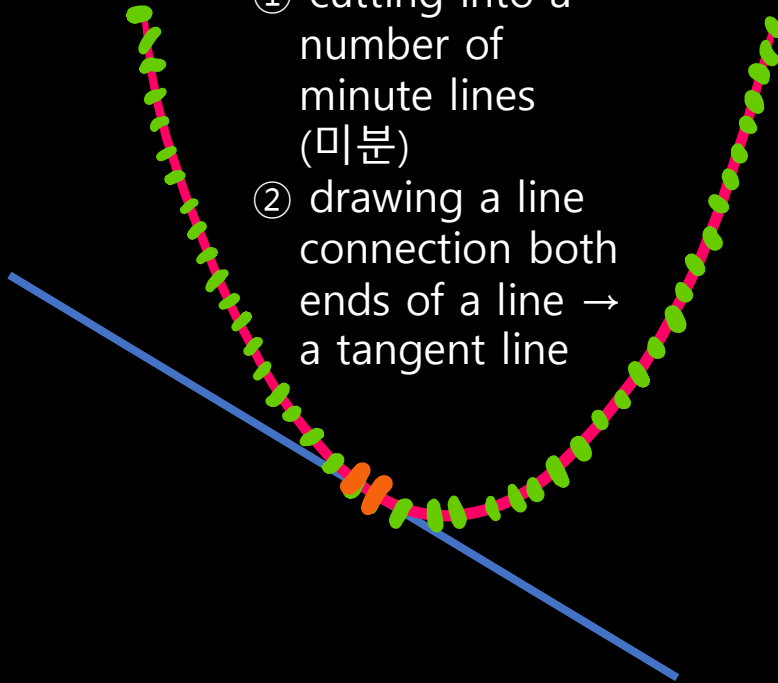
# How to update $w$



# How to update $w$

## Numerical differentiation

- ① cutting into a number of minute lines (미분)
- ② drawing a line connection both ends of a line  $\rightarrow$  a tangent line



$$\lim_{\Delta w \rightarrow 0} \frac{\Delta E}{\Delta w}$$

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

$$= \text{미분}$$

# How to update $w$

$$w = w - \alpha * \text{Slope}$$

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

$$\alpha = \text{learning rate (ex, 0.1)}$$

Squared Error

**L2 Loss function**

# Advantages

- Fast movement from both sides and fine tuning at the valley(center) area
- Different slope/gradient according to the value of  $w$
- Steep slope means that the error is big and  $w$  is far from the optimal area.
- We can get the slope(gradient) at any place(differentiable).

# In case of AE

- Always the same slope in the error graph regardless of the value of  $w$
- Therefore, the same speed in the movement
- Not sure to get the  $w$  value which gives 0 error or almost 0
- No way to guess the current value of  $w$
- Not differentiable when  $w$  is 1

# Multiple Data

For 3 instances of data

$x_i$	$y_i$
1	1
2	2
3	3

$$E = \frac{1}{3} \sum_{i=1}^3 (wx_i - y_i)^2$$



Add (2, 2), (3, 3)

$$E = \frac{1}{3} \sum_{i=1}^3 (wx_i - y_i)^2$$

Draw ( $w$ ,  $E$ )

# Multiple Data

In case of  $m$  instances,

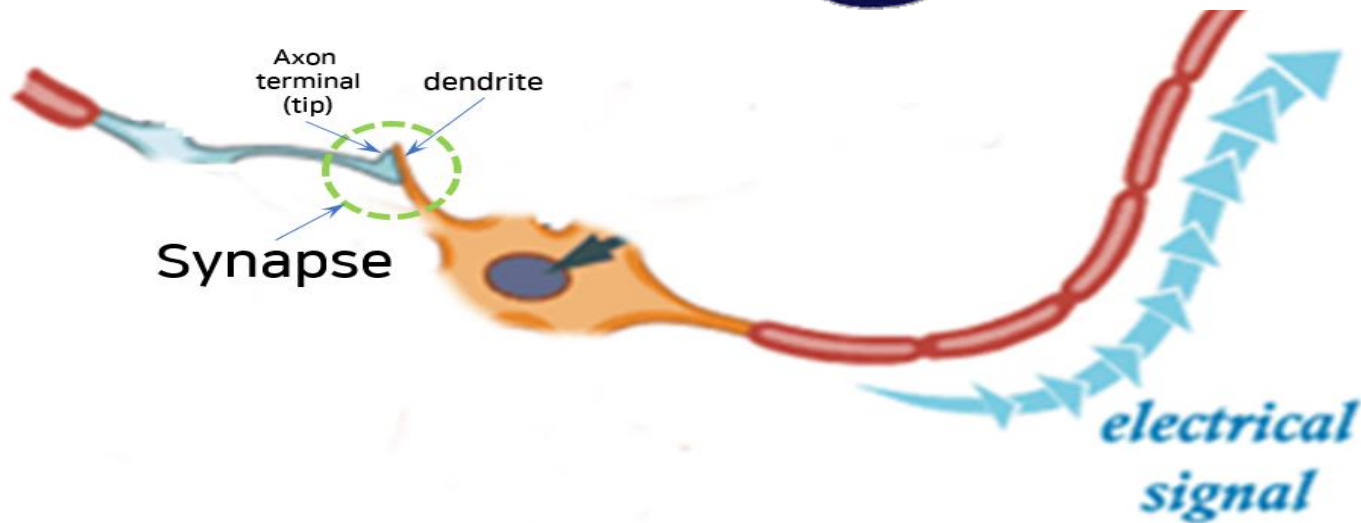
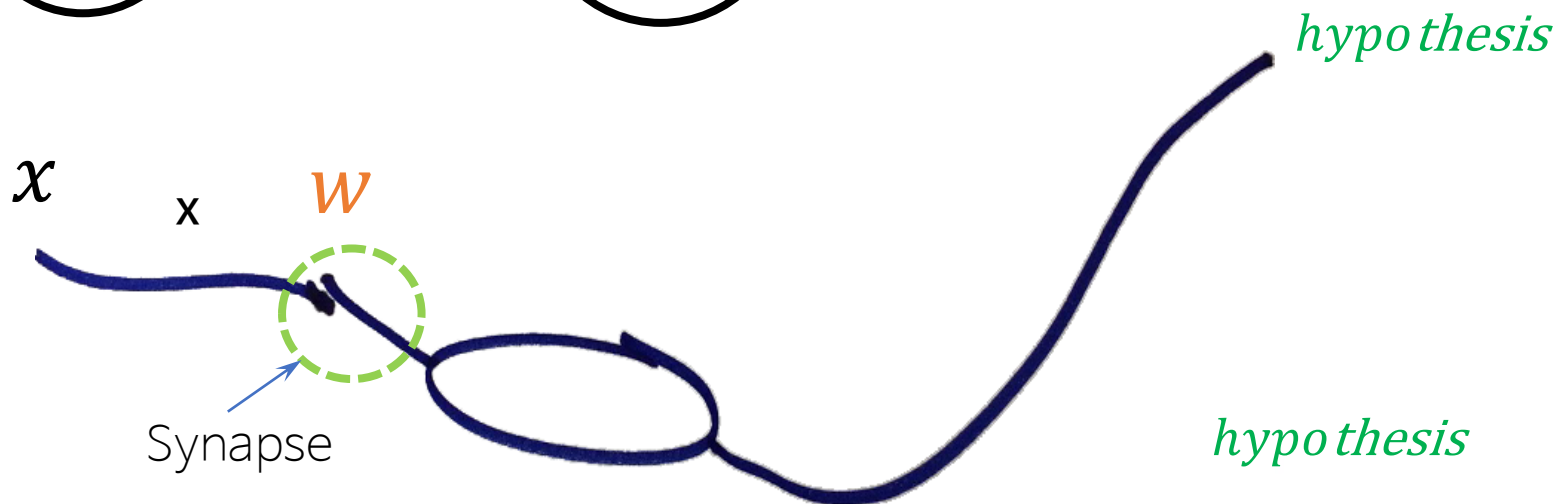
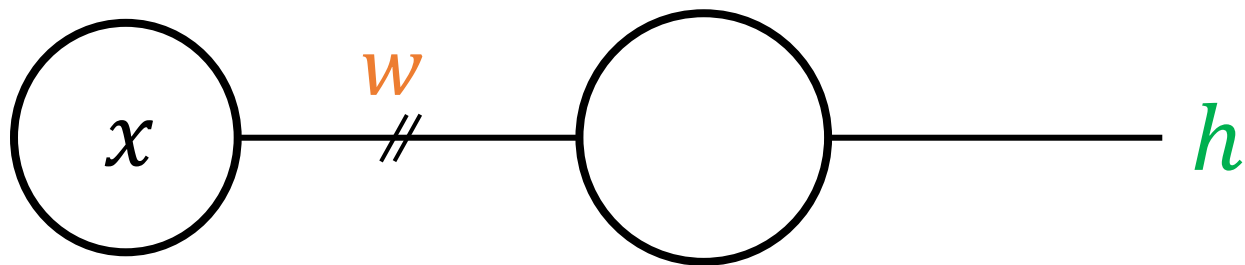
Mean Square  
Error

$$E = \frac{1}{m} \sum_{i=1}^m (\mathbf{w}x_i - y_i)^2$$

An answer by a neuron

Ground truth





# The meaning of slope

Steep slope  $\frac{\Delta E}{\Delta w}$   $\frac{8}{1}$

The error  $E$  will change drastically when we change  $w$ .

.....

Gentle slope  $\frac{\Delta E}{\Delta w}$   $\frac{1}{10}$

The error  $E$  changes just a little bit when we change  $w$ .

Therefore,

$$\lim_{\Delta w \rightarrow 0} \frac{\Delta E}{\Delta w} \rightarrow \frac{\partial E}{\partial w}$$

Slope/Gradient

means the influence of  $w$   
change on error  $E$ .

(Q) Compute the influence

$$E = (wx - y)^2$$

Let's assume that data  $(x, y)$  is  $(1, 1)$ ,  
then compute the influence of  $w$  change  
on  $E$  when the current  $w$  is equal to 3.

# Method1 numerical gradient

$$E = (w \cdot 1 - 1)^2$$

$$w: 3 \rightarrow E: 4$$

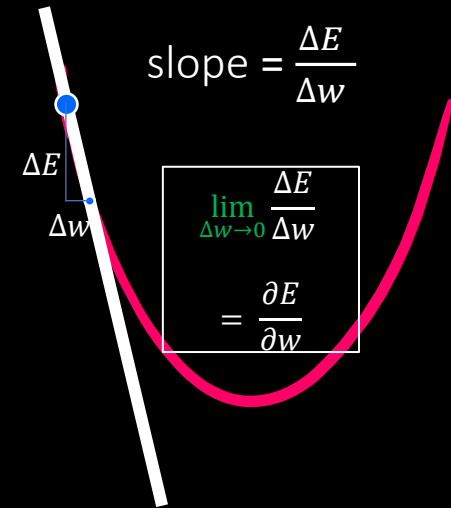
$$w: 3.00001 \rightarrow E: 4.00004$$

$$\Delta w = 0.00001$$

$$\Delta E = 0.00004$$

$$\frac{\Delta E}{\Delta w} = \frac{0.00004}{0.00001} = 4$$

Slope =  
Influence of  $w$  change = 4



# Method2 derivative, differential equation

$$E = (w \cdot 1 - 1)^2$$

$$\begin{aligned} \lim_{\Delta w \rightarrow 0} \frac{\Delta E}{\Delta w} &= \frac{\partial E}{\partial w} = \frac{\partial}{\partial w} (w \cdot 1 - 1)^2 \\ &= 2(w \cdot 1 - 1) \end{aligned}$$

Therefore, when  $w = 3$ ,  
the gradient is  $2(3 - 1) = 4$

# How to update $w$ (Learning)

1. Initialize  $w$  with a random value (ex, 3)

2. Get influence(slope) of  $w$  on  $E$

Loop

3. To decrease the error, update  $w$  using below eq:

Parameter Tuning

$$W = W - \alpha * \text{slope}$$

4. Go to step 2

# TensorFlow



- Machine learning framework by Google
- Tuning parameters including  $w$  automatically instead of us
- Define  $w$  inside of TensorFlow to be tuned (managed) by it
- Hypothesis and cost\_function( $E$ )



# Linear Regression using TF

③

`w = tf.Variable(tf.random_normal([1]))`

`hypo = w * x` ④

$h$

$y = [1]$

②

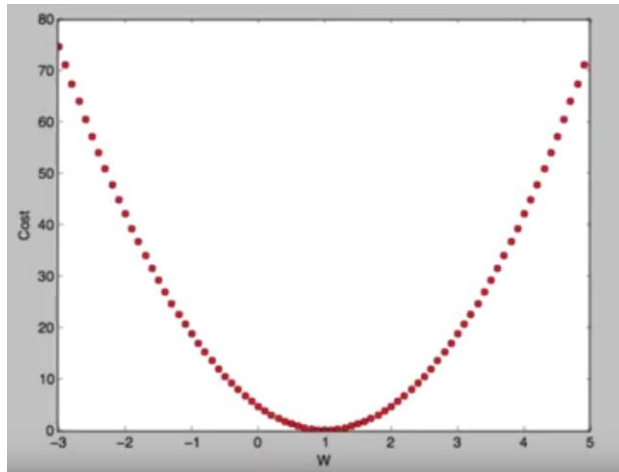
$x$

$x$

$w$

①

$x = [1]$



`cost_function = (hypo - y) ** 2`

⑤

$$E = (\text{hypo} - y)^2$$

# Download myml.git

<https://github.com/yungbyun/myml.git>

- 1) Run DOS prompt
- 2) git clone <https://github.com/yungbyun/myml.git>
- 3) Open using PyCharm (File | Open...)

Lab 01.py

Finding  $w$  in  
linear regression

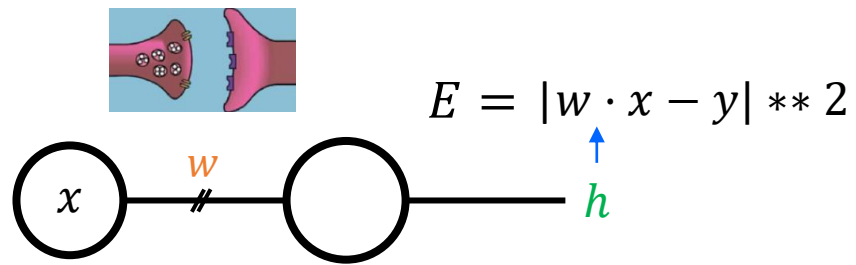
```
import tensorflow as tf
```

```
#----- training data  
x_data = [1]  
y_data = [1]
```

```
#----- a neuron / neural network  
w = tf.Variable(tf.random_normal([1]))  
hypo = w * x_data
```

```
#----- learning  
cost = (hypo - y_data) ** 2  
  
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)  
  
sess = tf.Session()  
sess.run(tf.global_variables_initializer())  
  
for i in range(1001):  
    sess.run(train) #1-run, 1-update of w -> 1001 updates  
  
    if i % 100 == 0:  
        print('w:', sess.run(w), 'cost:', sess.run(cost))
```

```
#----- testing(prediction)  
x_data = [2]  
print(sess.run(x_data * w))
```

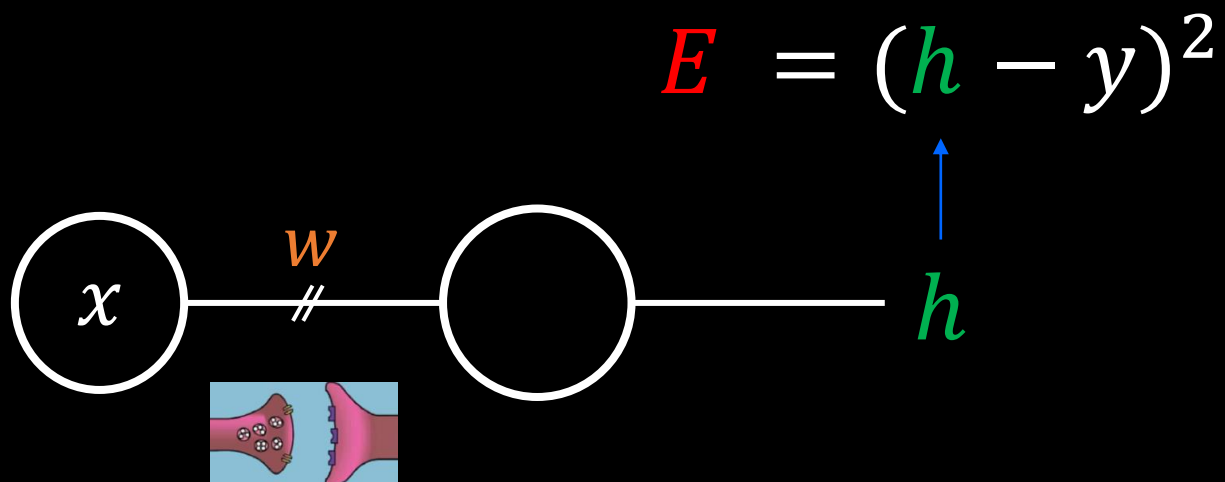


train operation to  
update  $w$  to minimize  
error( $E$ )

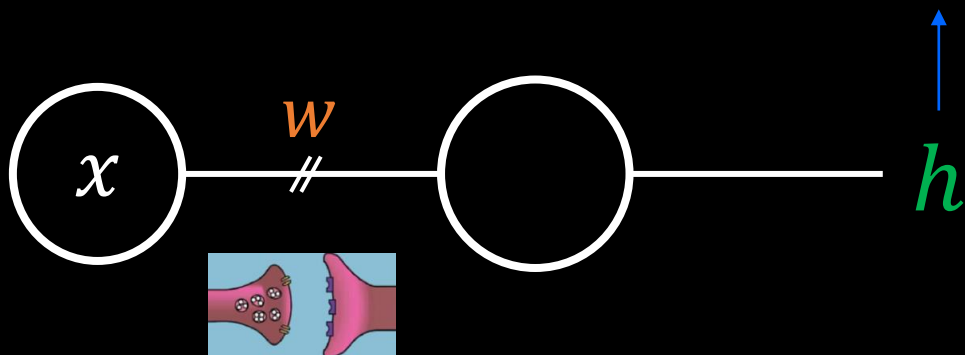
```
sess.run(train)
```

How to update  $w$  in  
TensorFlow

Computation Graph



$$E = (wx - y)^2$$



# Loss/error function

$$E = (wx - y)^2$$

- The part representing a neuron
- Where is a synapse?
- Which one is an input data?
- The output of a neuron
- Find hypothesis
- Find a correct answer or ground truth.
- Imagine  $E$  having multiple inputs.

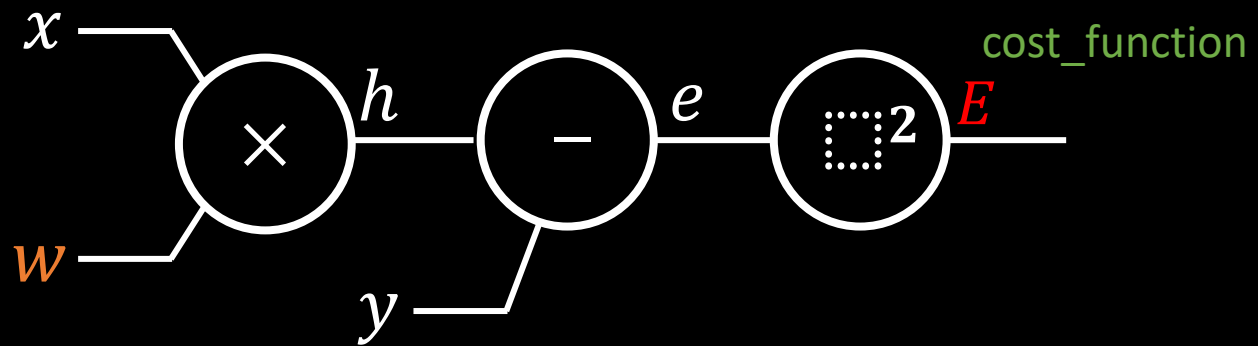


# Computation Graph

$$E = (w \cdot x - y)^2$$

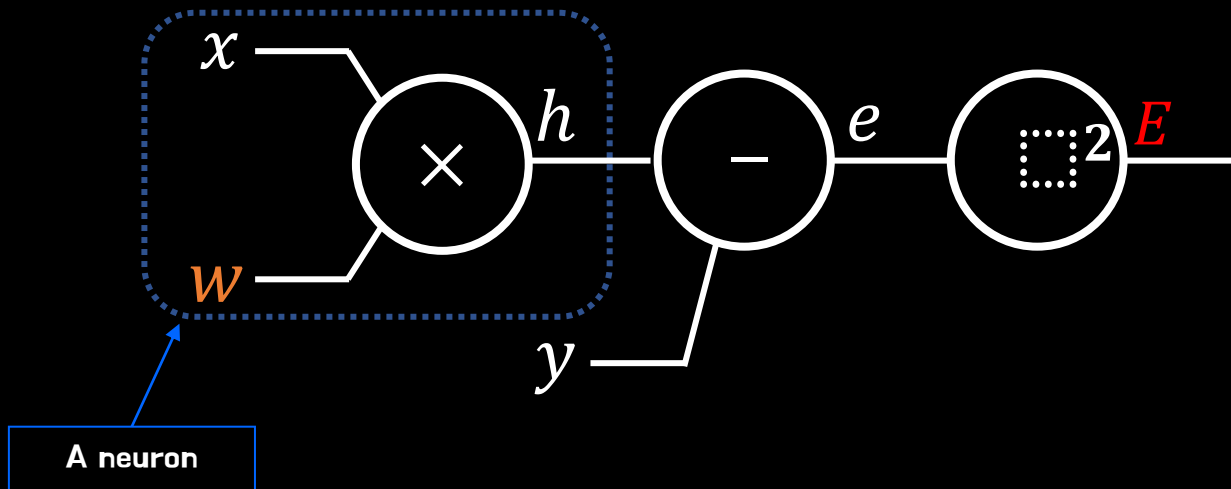
hypo = w \* x

cost\_function(E) = (hypo - y) \*\* 2



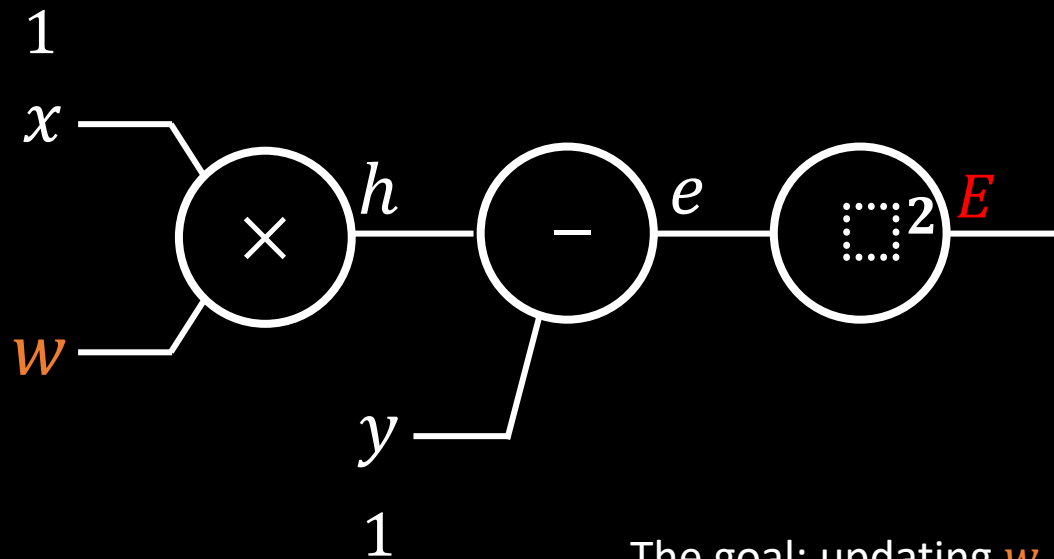
# Computation Graph

$$E = (w \cdot x - y)^2$$



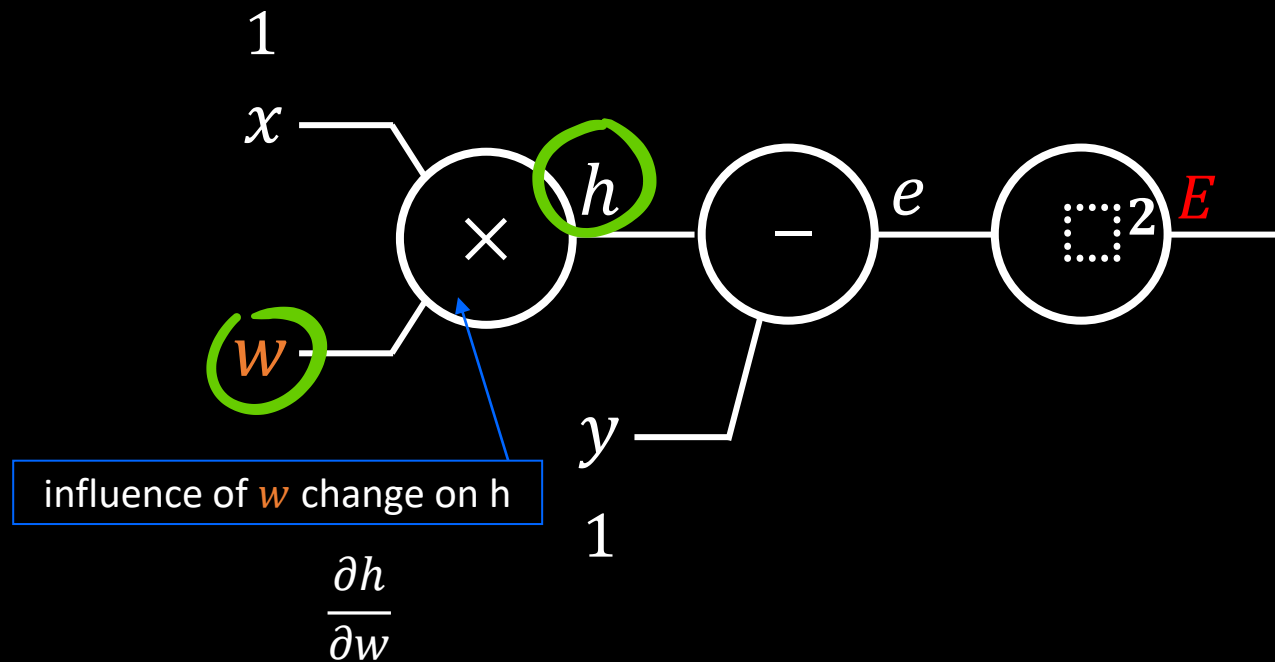
# Computation Graph

**Data**  $(x, y) \rightarrow (1, 1)$

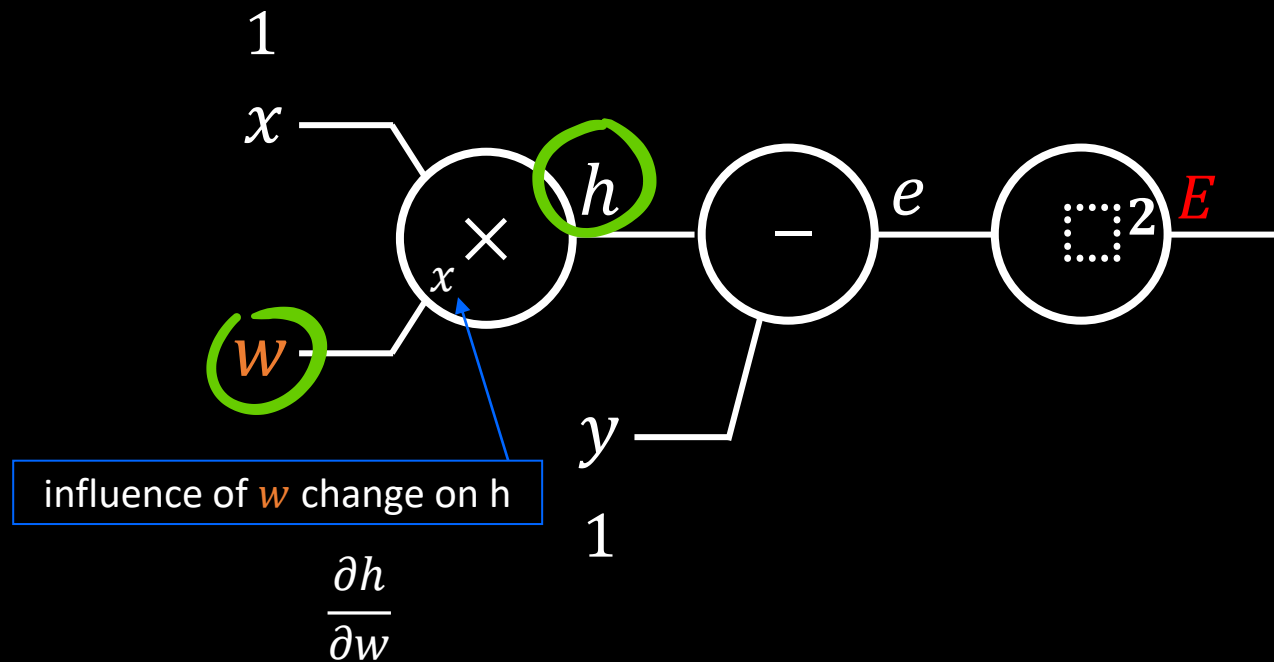


The goal: updating  $w$ , how?

# Computation Graph



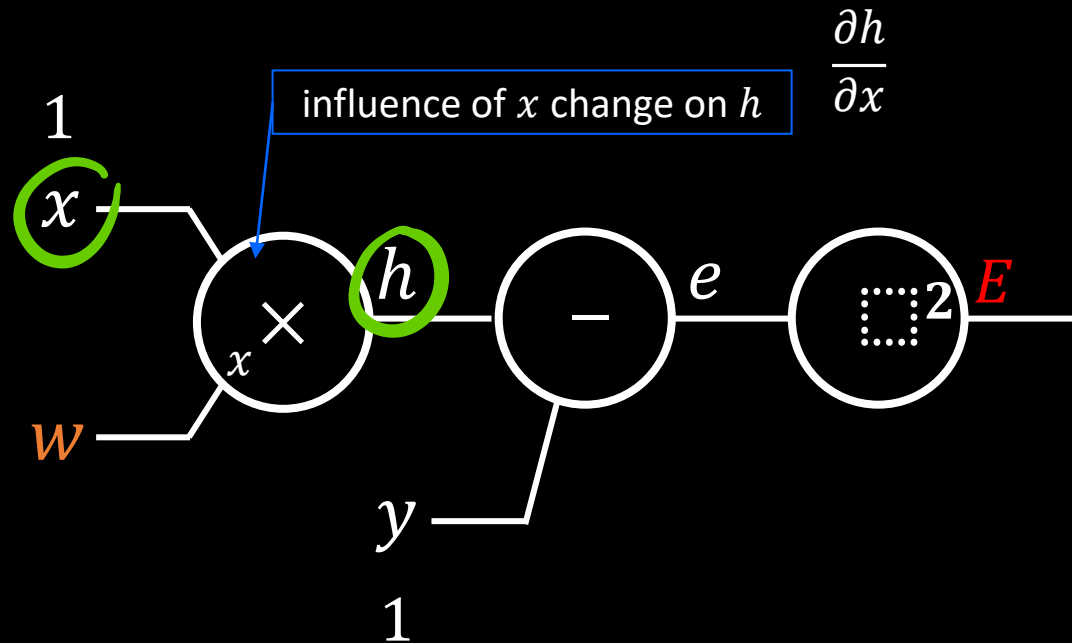
# Computation Graph



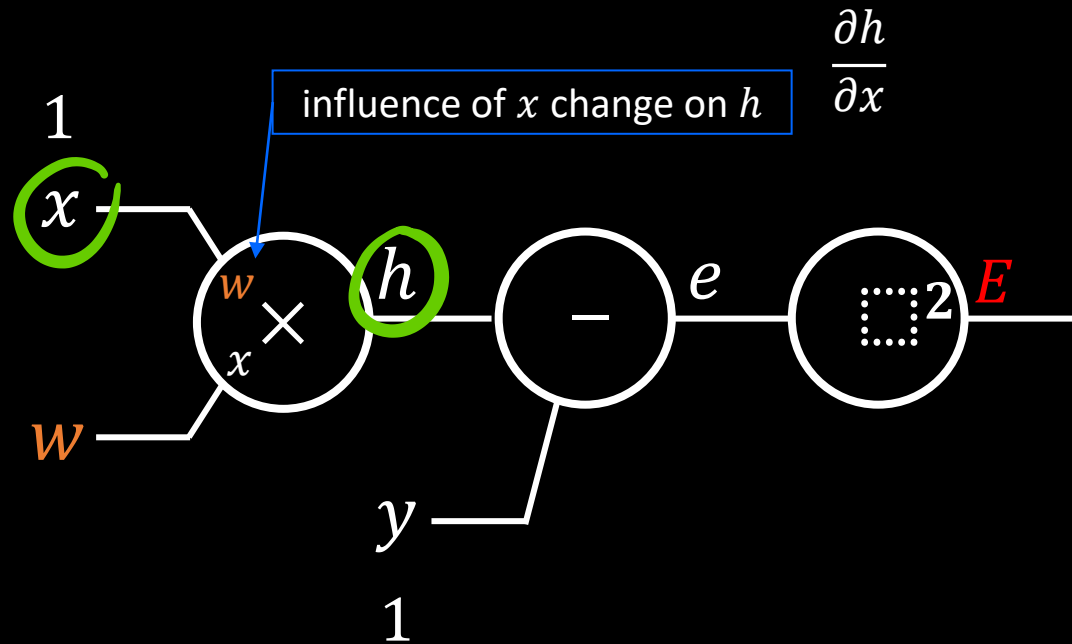
# Local gradient

부분/지역 기울기

# Computation Graph

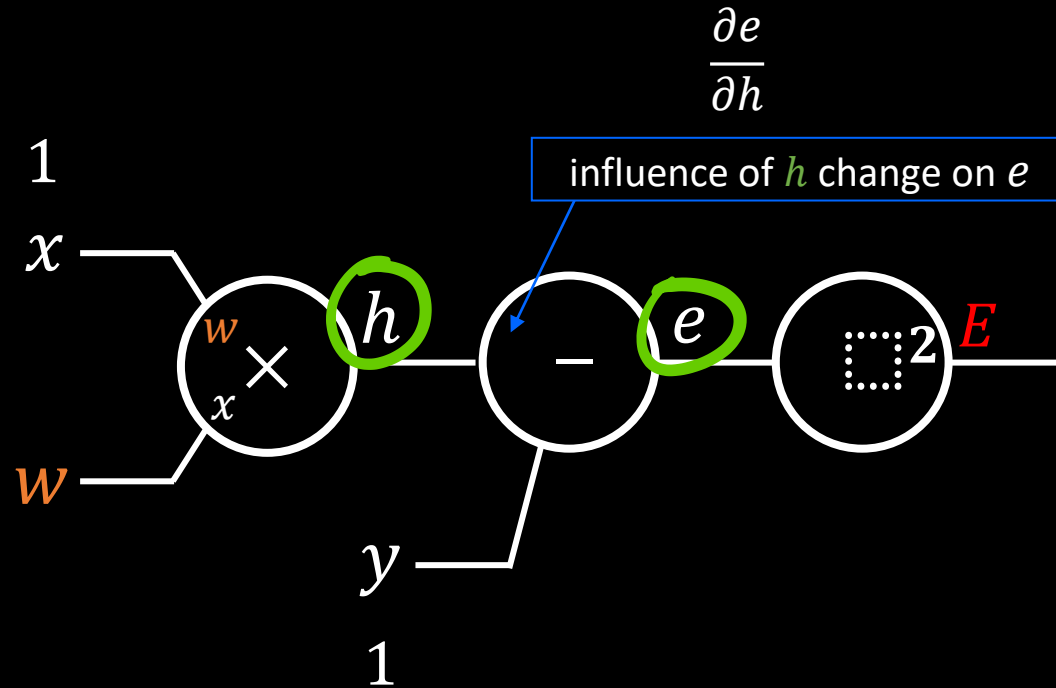


# Computation Graph

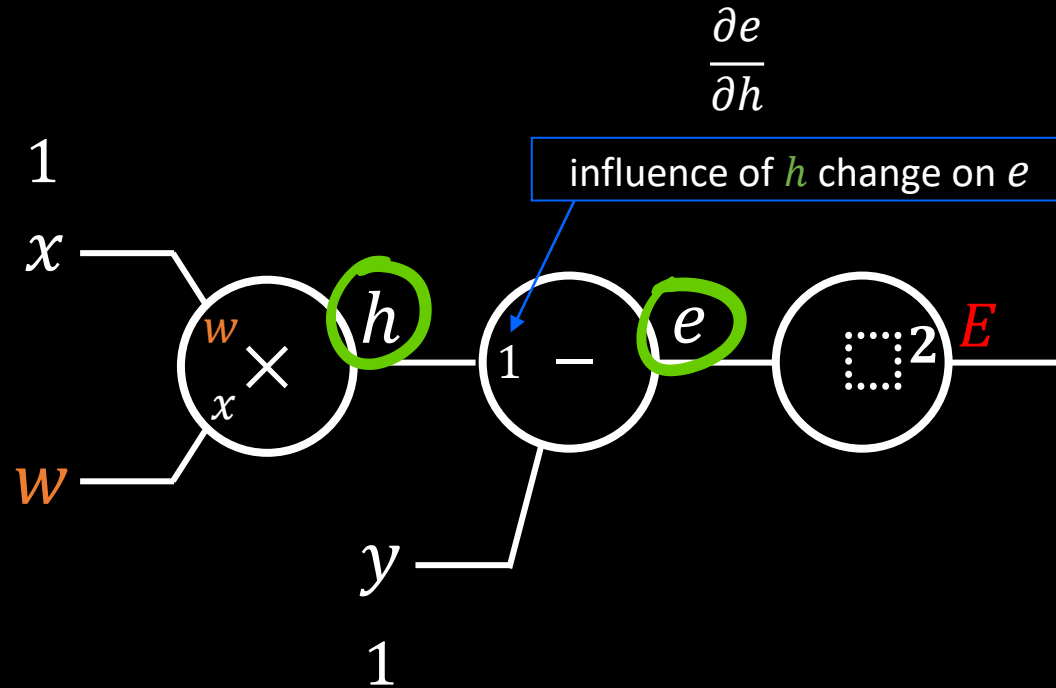




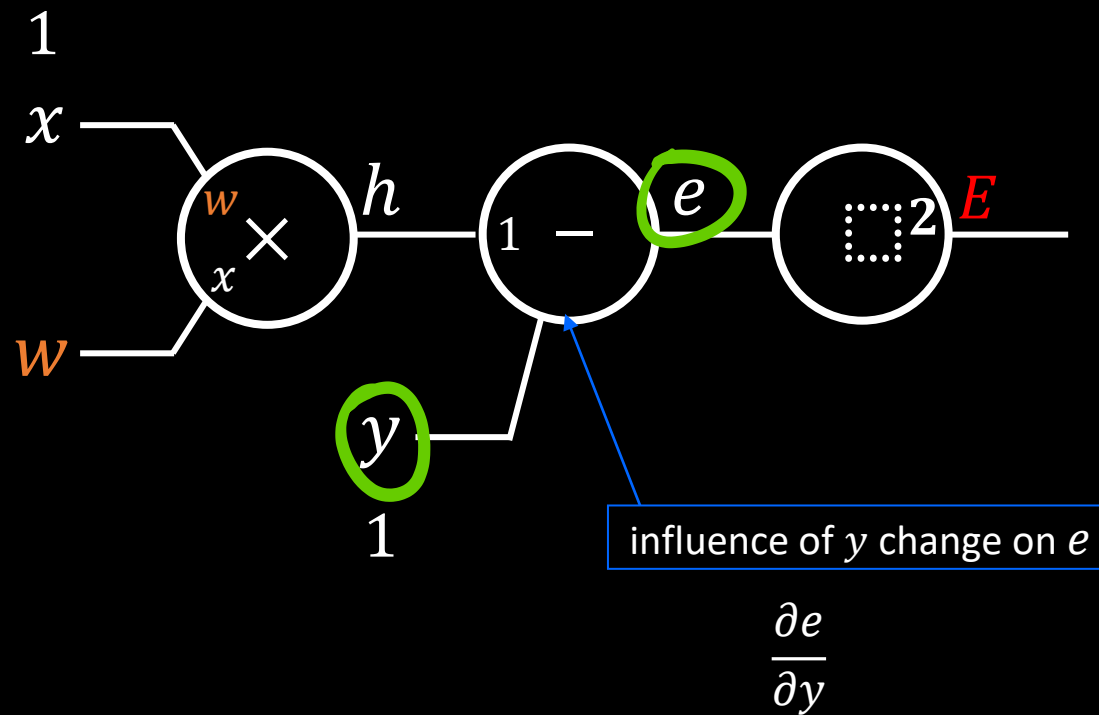
# Computation Graph



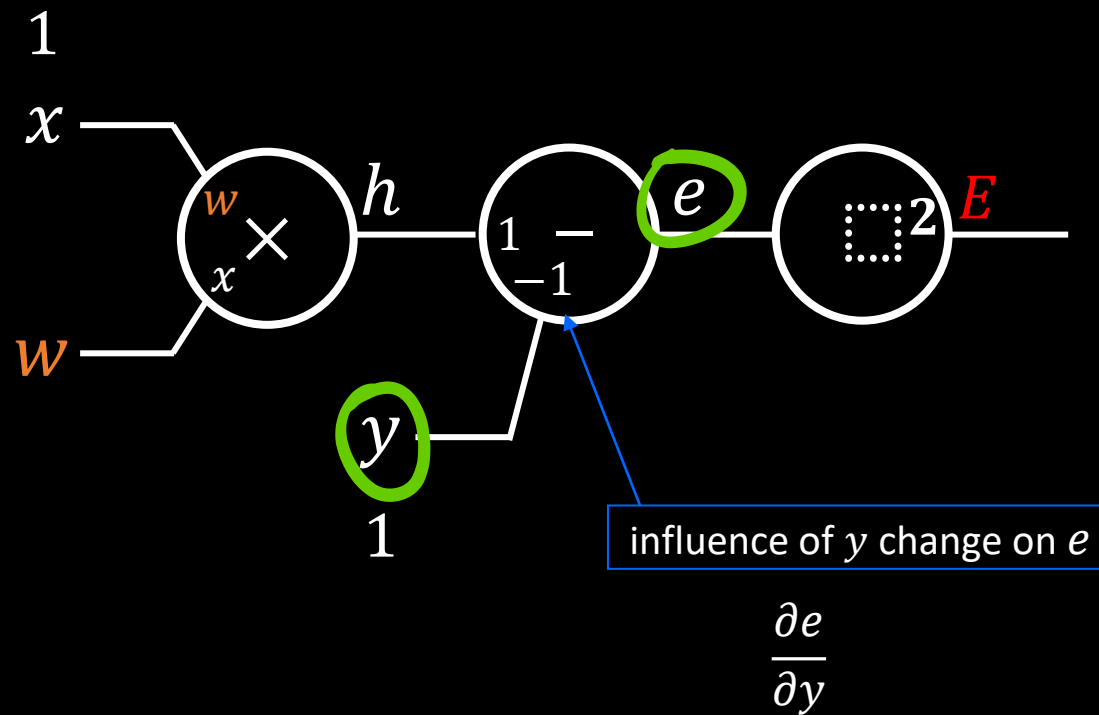
# Computation Graph



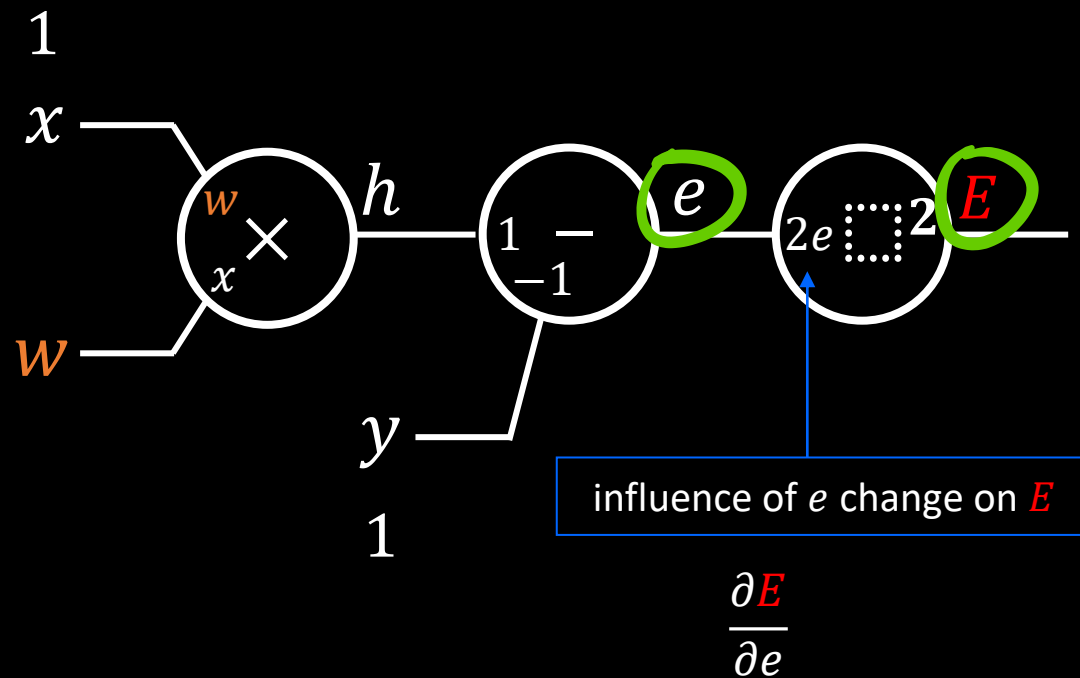
# Computation Graph



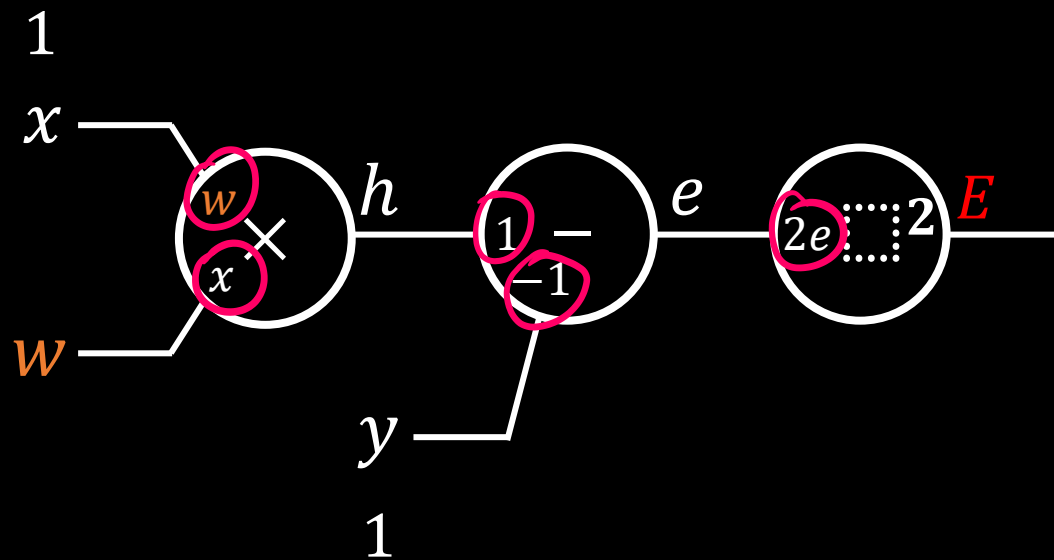
# Computation Graph



# Computation Graph

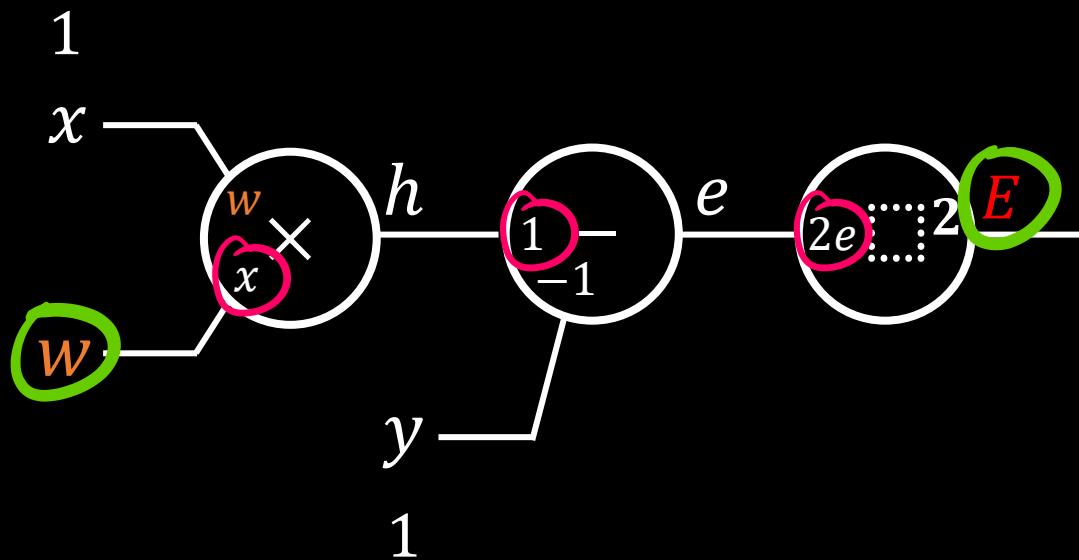


# 5 Local Gradients in gates



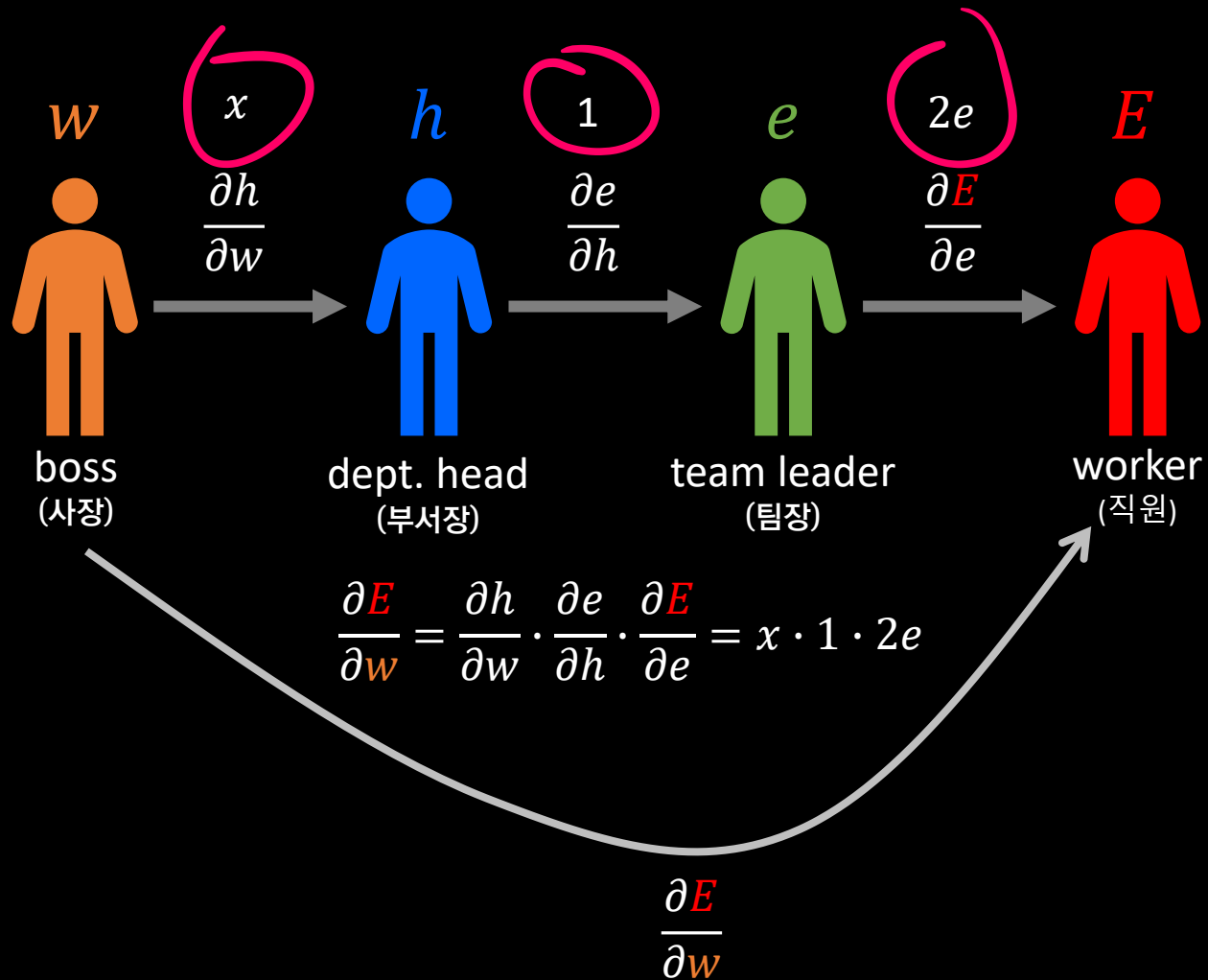
How can we get the influence  
of  $w$  change on  $E$ ?

# 3 Local Gradients in gates





# Influence between persons

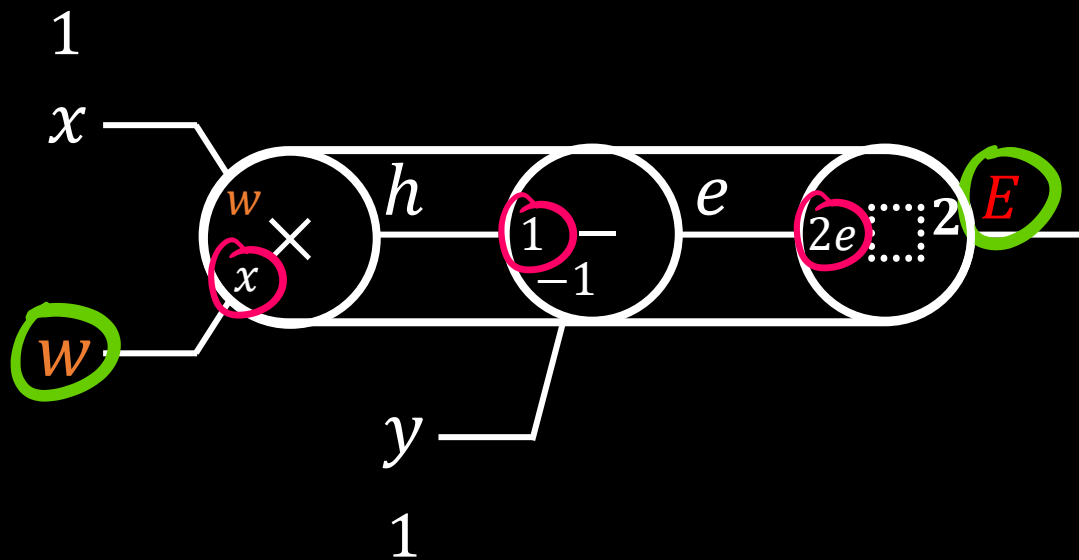


The influence of  $w$  change  
on  $E$

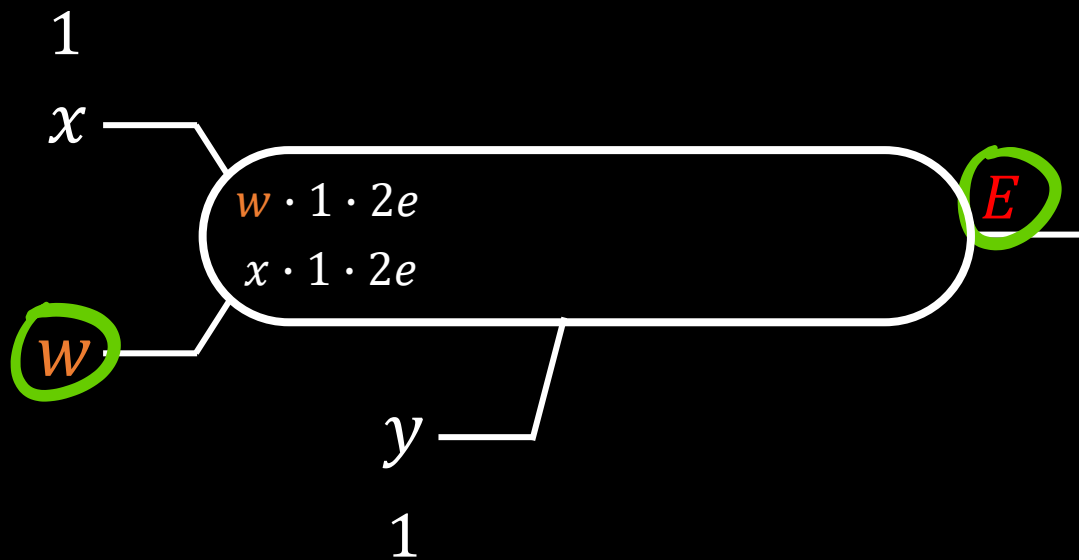
$$\frac{\partial E}{\partial w} = \frac{\partial h}{\partial w} \times \frac{\partial e}{\partial h} \times \frac{\partial E}{\partial e}$$

Chain rule!

# Merging Gate

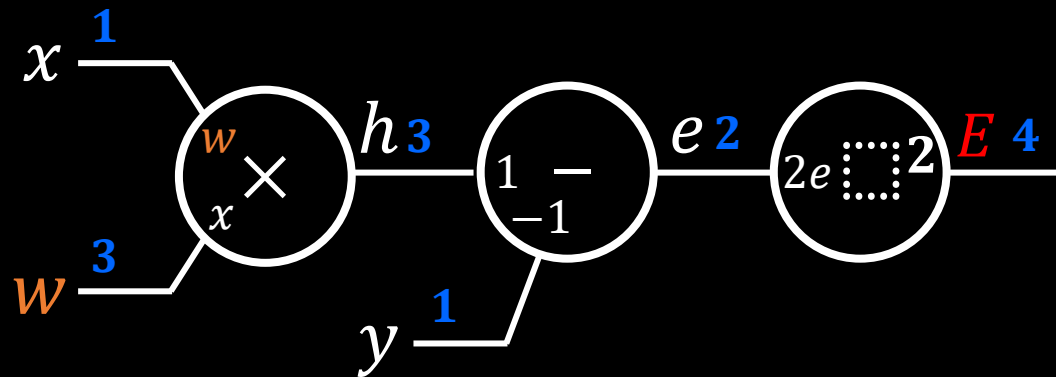


# *Composite Gate*



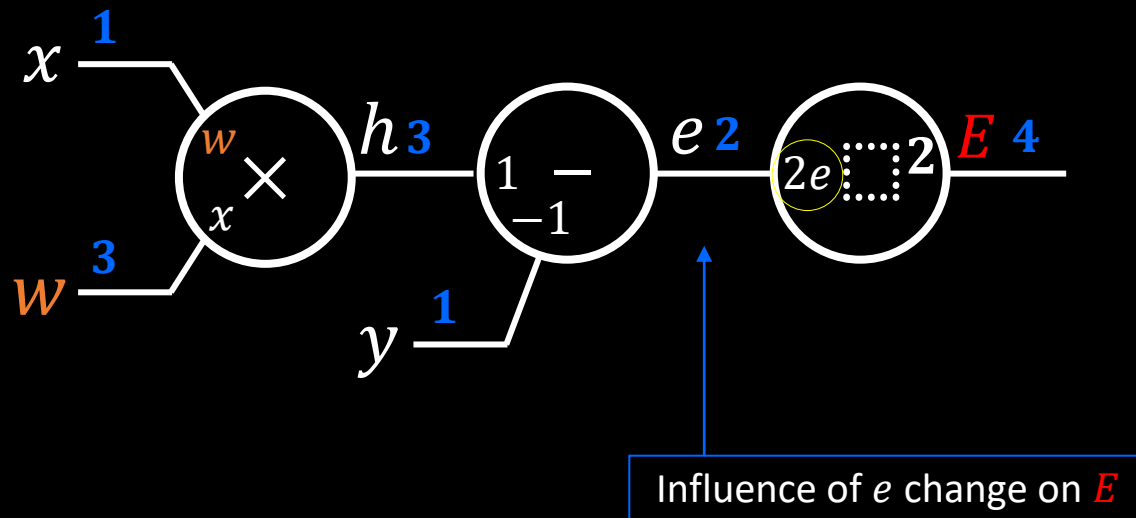
# *Forward* propagation

Let  $(x, y) = (1, 1)$  and  $w = 3$ , then compute  $E$ .

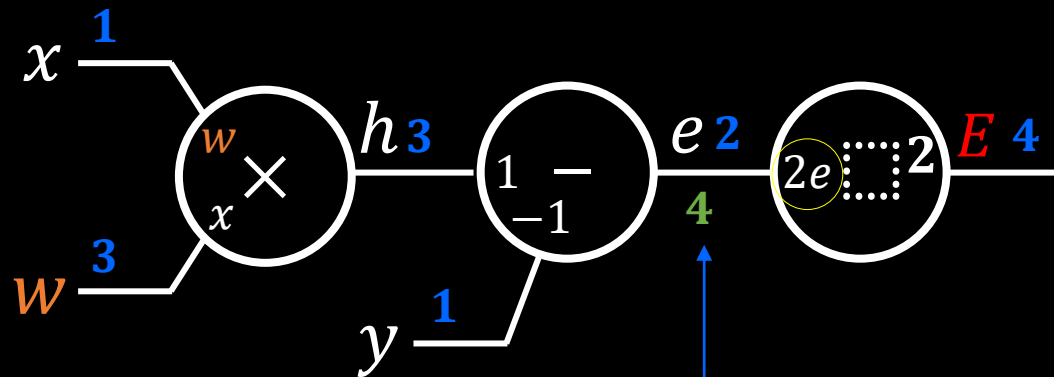


Error is big (4),  
so, let's update  $w$   
using *back*-propagation.

# Back-propagation



# Back-propagation

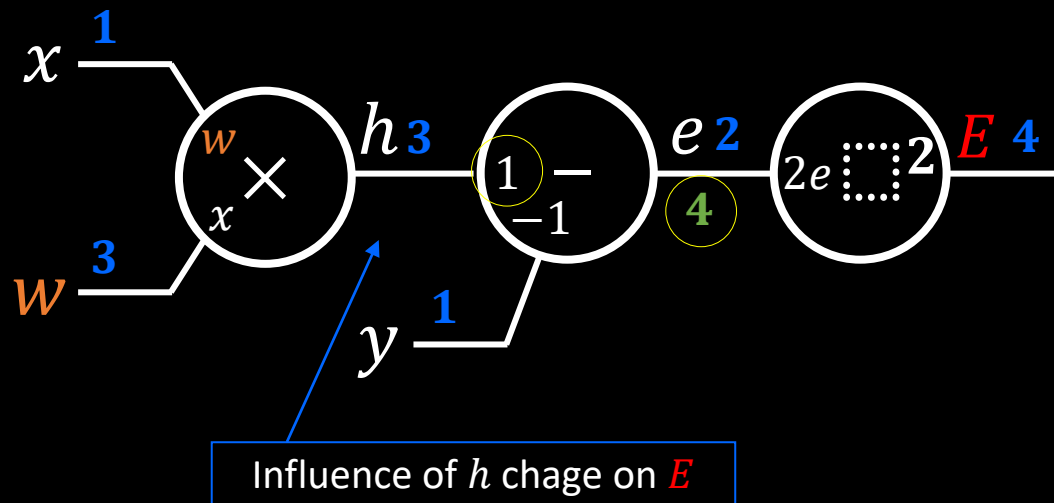


Influence of  $e$  change on  $E$

$$\frac{\partial E}{\partial e} = 2e$$

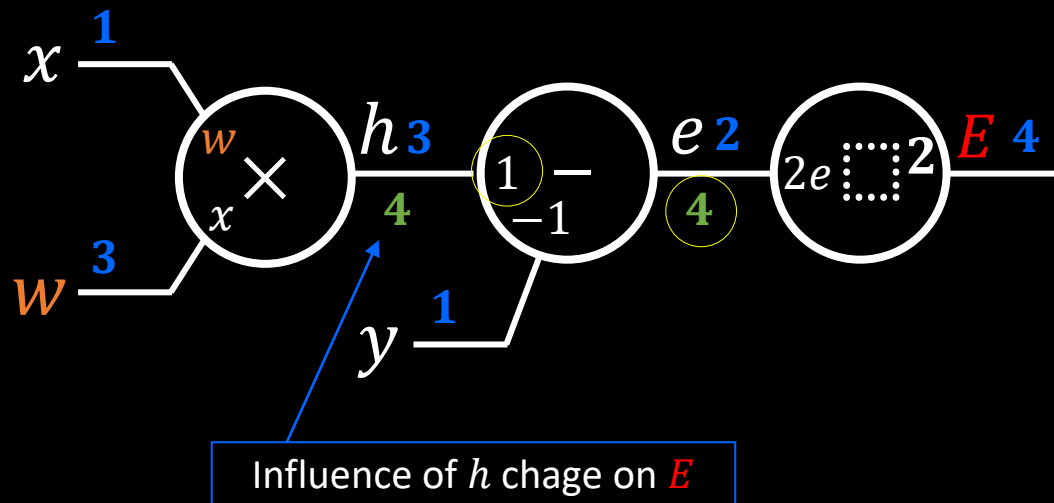


# Back-propagation

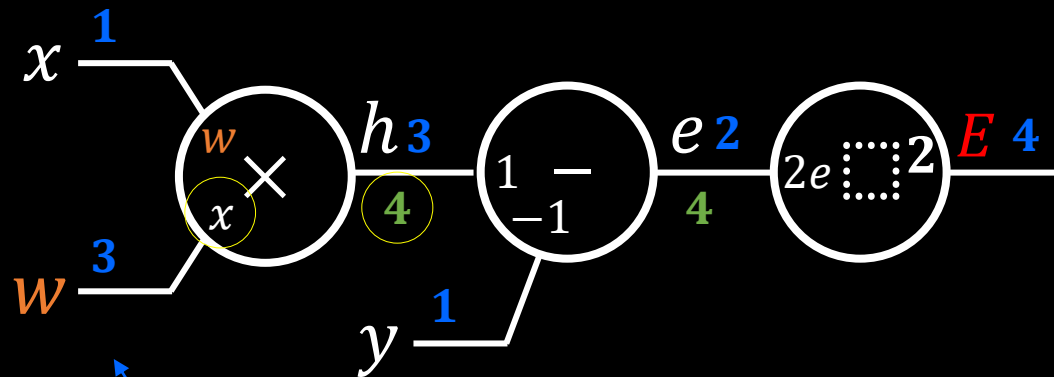


$$\frac{\partial E}{\partial h} = \frac{\partial e}{\partial h} \cdot \frac{\partial E}{\partial e} = 1 \cdot 4$$

# Back-propagation

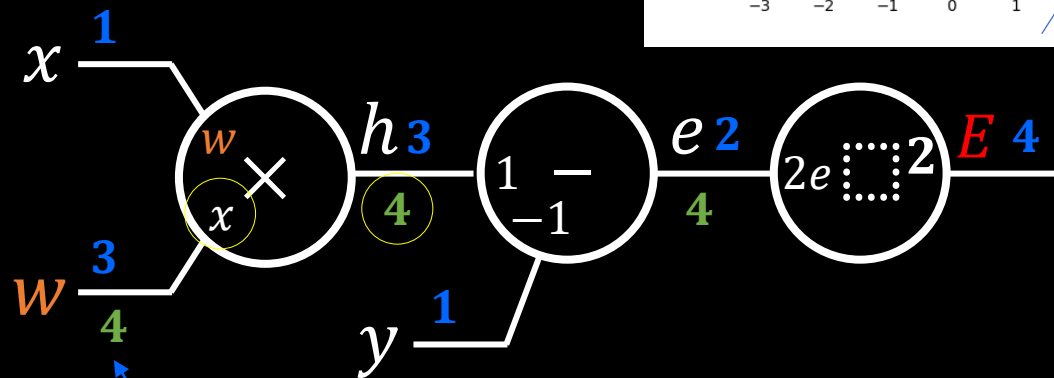
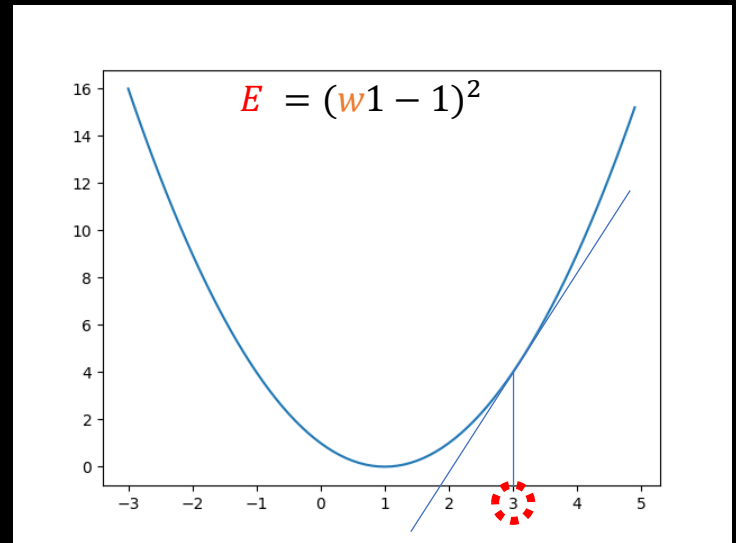


# Back-propagation



Influence of change  $w$  on  $E$

$$\frac{\partial E}{\partial w} = \frac{\partial h}{\partial w} \cdot \frac{\partial E}{\partial h} = 1 \cdot 4$$



Influence of change  $w$  on  $E$

$$\frac{\partial E}{\partial w}$$

Back-propagation,  
the process to **apply**  
**chain rules**

$$\frac{\partial E}{\partial w}$$

$$w = 3 - 0.1 * 4 \frac{\partial E}{\partial w}$$
$$w = 2.6$$



Tuned parameter after 1  
step learning

After enough number of steps(epochs),  
the parameter  $w$  will be optimized  
properly.

# *Back*propagation

by Paul Webros (1974, 1982) and  
Geoffrey Hinton (1986)




Prof. Univ. of Toronto,  
Google Brain  
Yann LeCun (his post doc)

```
import tensorflow as tf
```

```
#----- training data  
x_data = [1]  
y_data = [1]
```

```
#----- a neuron / neural network  
w = tf.Variable(tf.random_normal([1]))  
hypo = w * x_data
```

train operation to  
update  $w$  to  
minimize cost(error)



```
#----- learning  
cost = (hypo - y_data) ** 2  
  
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)  
  
sess = tf.Session()  
sess.run(tf.global_variables_initializer())  
  
for i in range(1001):  
    sess.run(train) # 1-run, 1-update of  $w$  -> 1001 updates  
  
    if i % 100 == 0:  
        print('w:', sess.run(w), 'cost:', sess.run(cost))
```

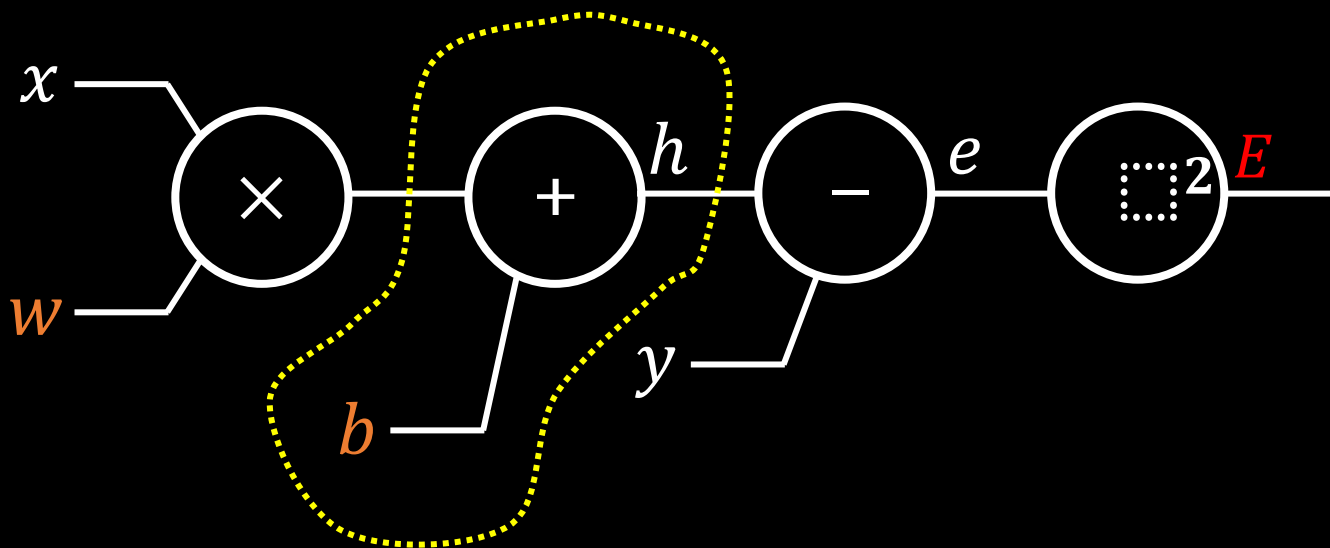
```
#----- testing(prediction)  
x_data = [2]  
print(sess.run(x_data * w))
```



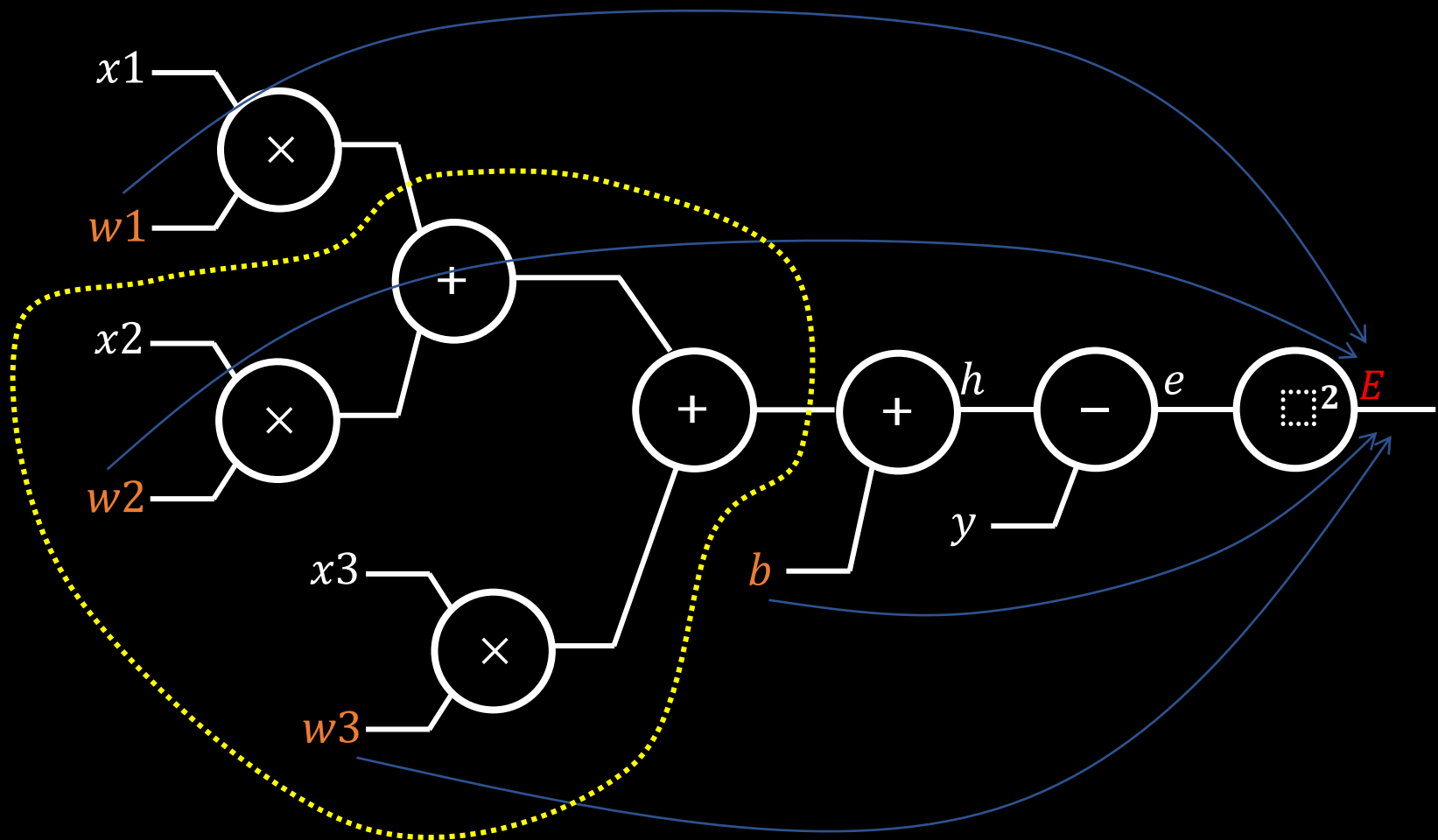
# Extension of the Graph

- adding bias  $b$  (one more plus gate)
- a neuron with 3 inputs (2 more + gate)
- two neurons

$$E = ((wx + b) - y)^2$$

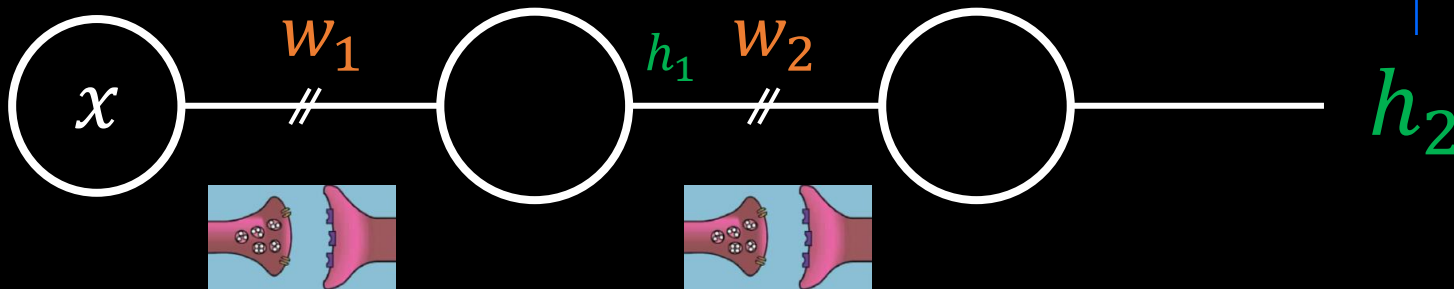


$$E = ((w1x1 + w2x2 + w3x3 + b) - y)^2$$



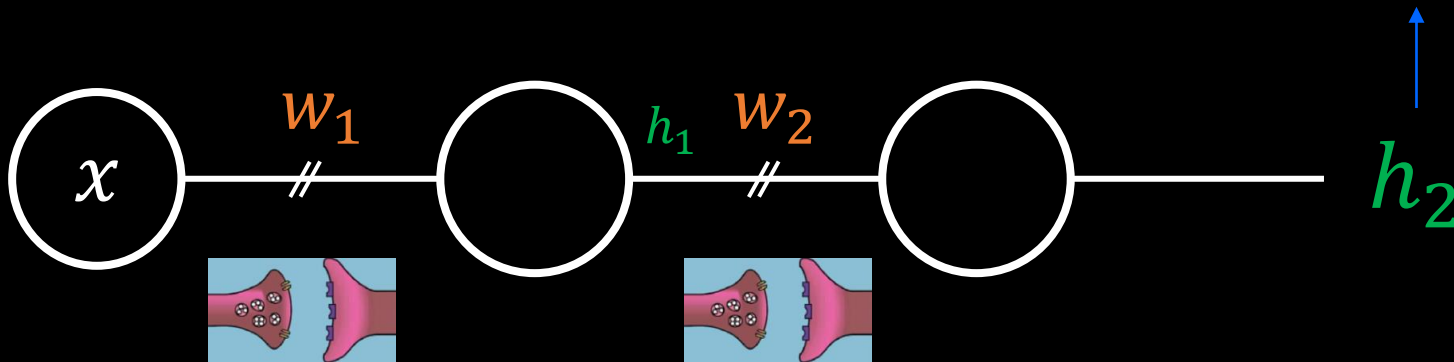
# Computation Graph for Two Neurons

$$E = (w_2 h_2 - y)^2$$



# Computation Graph for Two Neurons

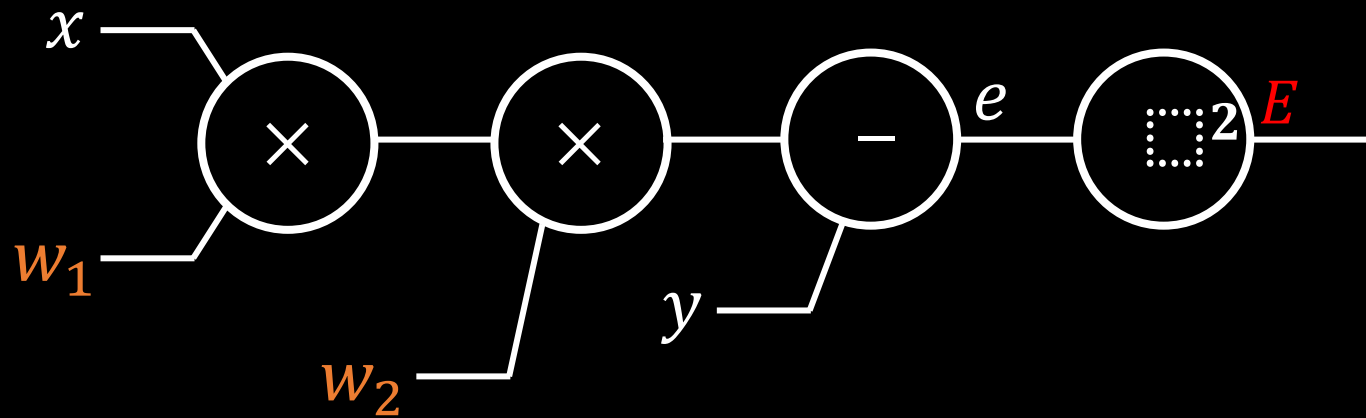
$$E = (w_2(w_1x) - y)^2$$



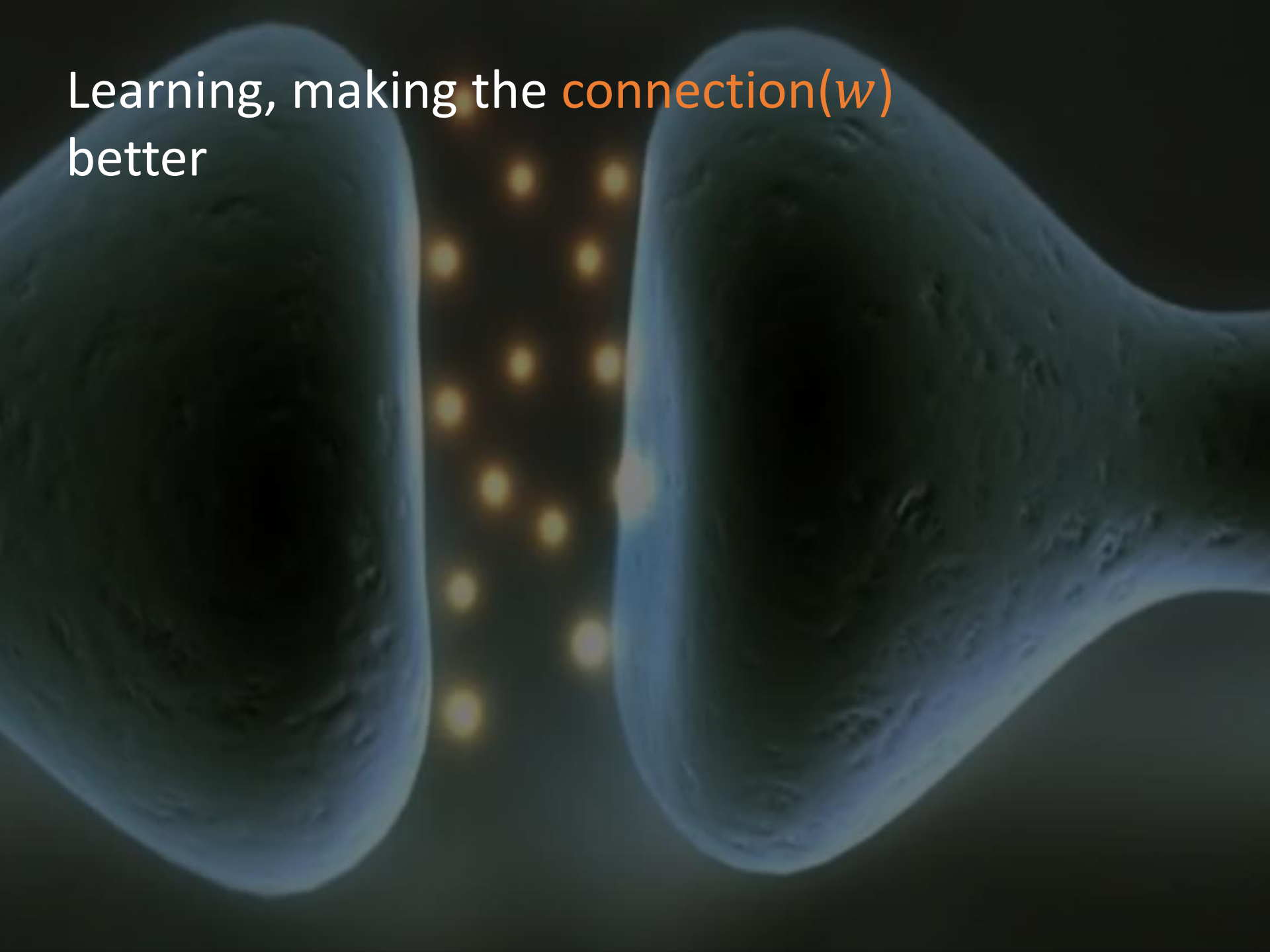
If the hypothesis( $h_2$ ) is the **linear** combination of coefficients( $w_1, w_2$ ), then it is a **linear** model.

# Computation Graph for Two Neurons

$$E = (w_2(w_1x) - y)^2$$

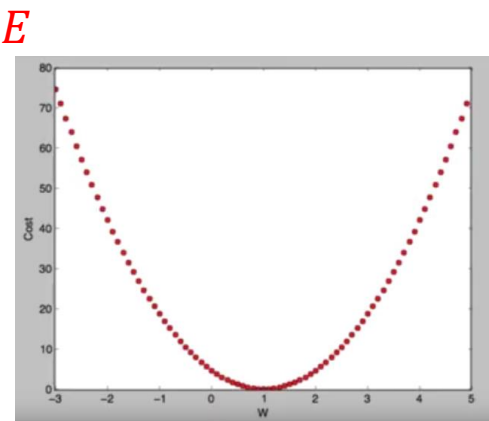
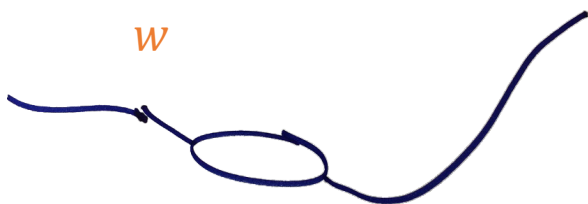


Learning, making the  $\text{connection}(w)$   
better



# Cost(Error) graph

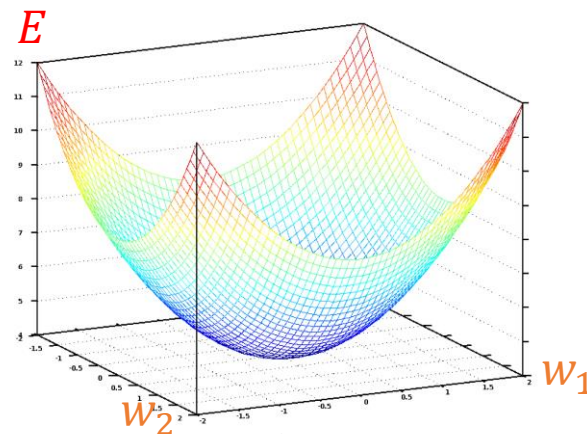
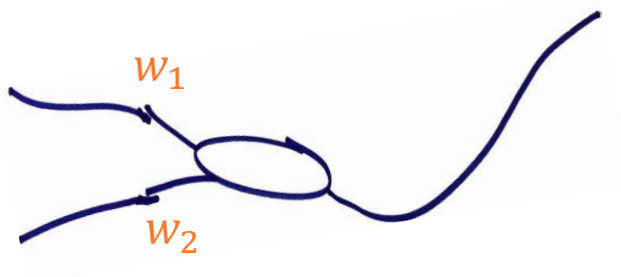
$$E = (w \cdot 1 - 1)^2$$



$w$

convex function

$$E = (w_1 \cdot 1 + w_2 \cdot 1 - 1)^2$$



convex function



Lab 02.with\_bias.py

Parameter tuning  
including bias

Lab 03.py

Using multiple  
data

Lab 04.py

Training a neuron  
having multiple  
inputs