

AI and Deep Learning

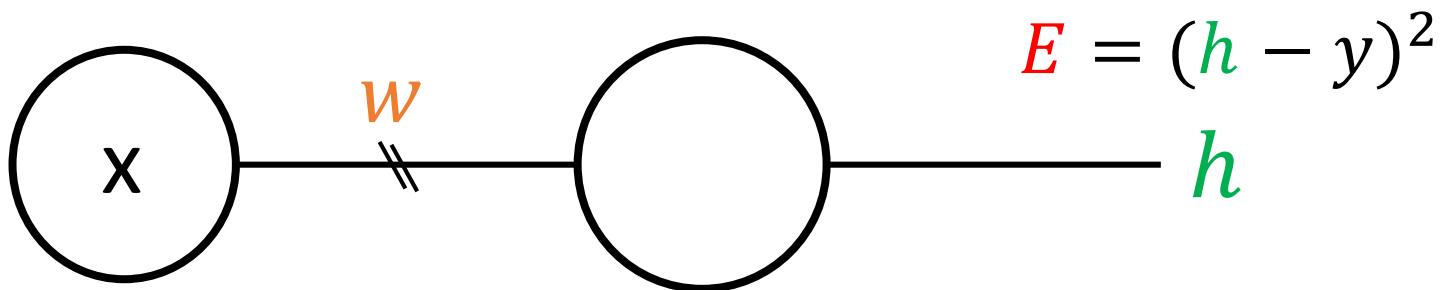
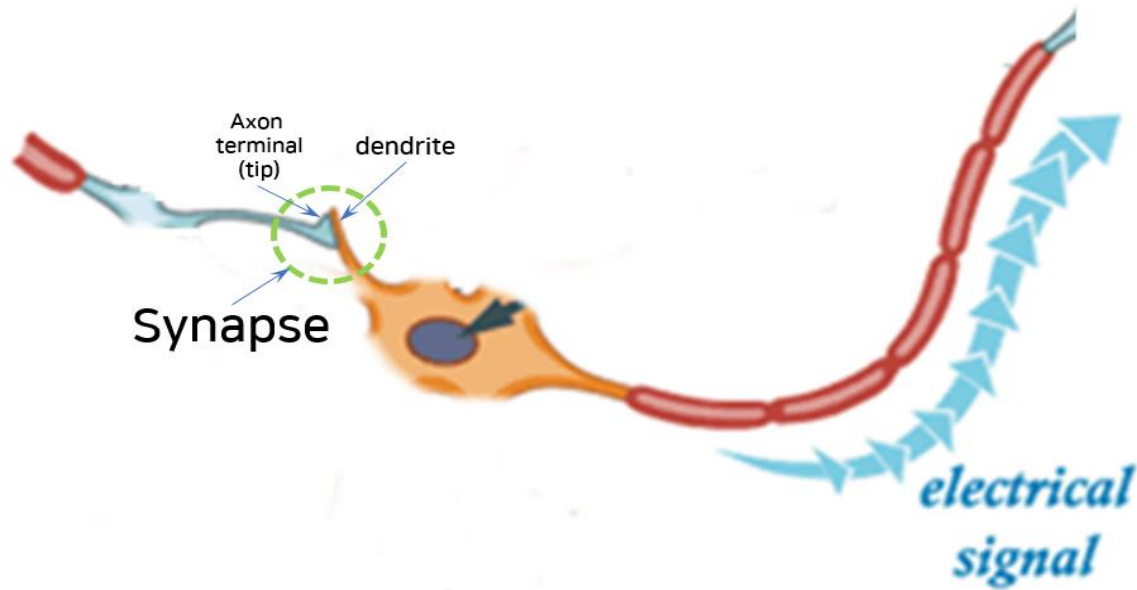
Logistic Regression & Classification

Jeju National University

Yungcheol Byun

Agenda

- Logistic regression and classification
- New loss/cost function
- Decision boundary
- Implementation using TensorFlow
- Multiple-class problem



Logistic Regression

The shape of regression is **not linear**
but logistic.

What does that mean?

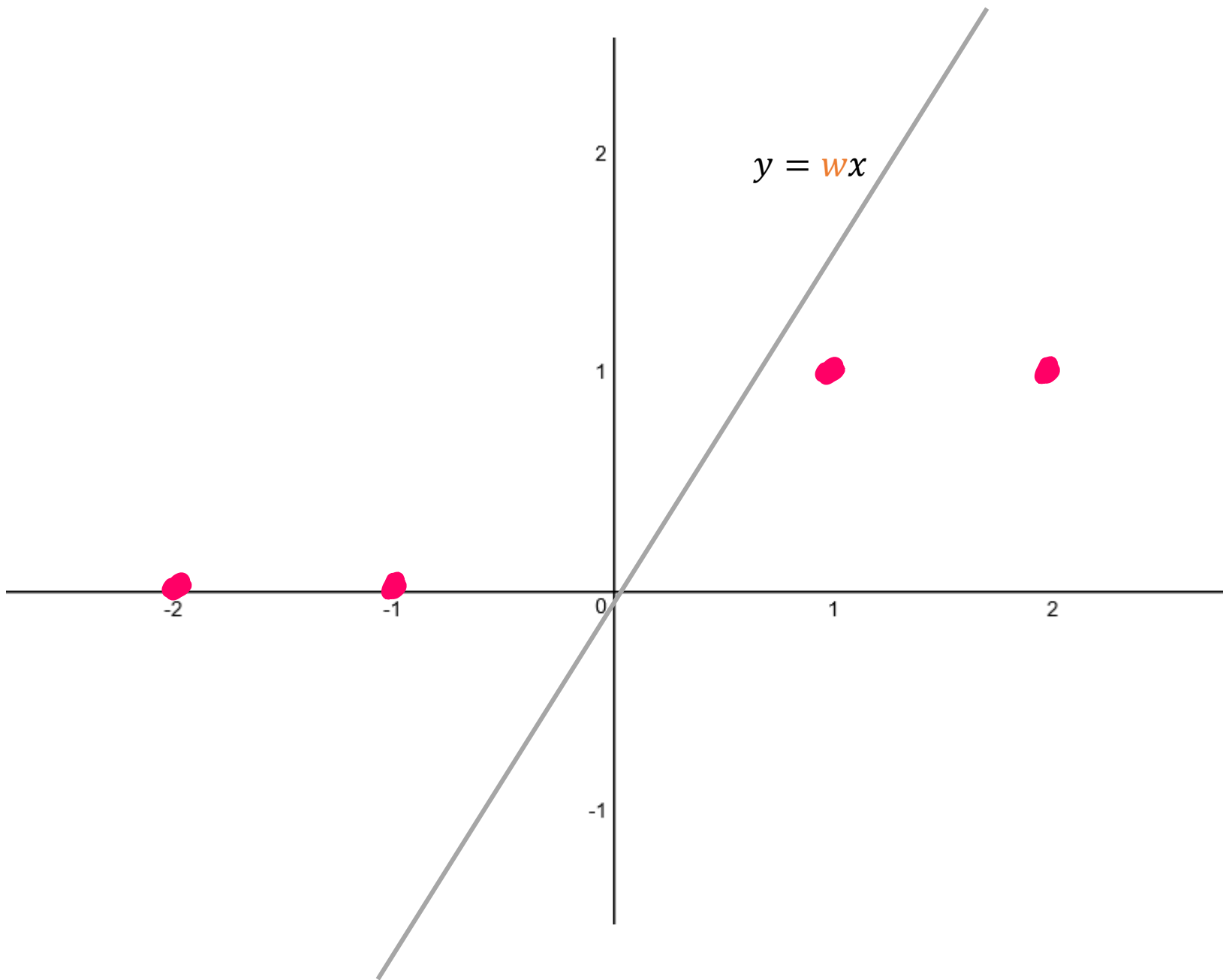


desmos

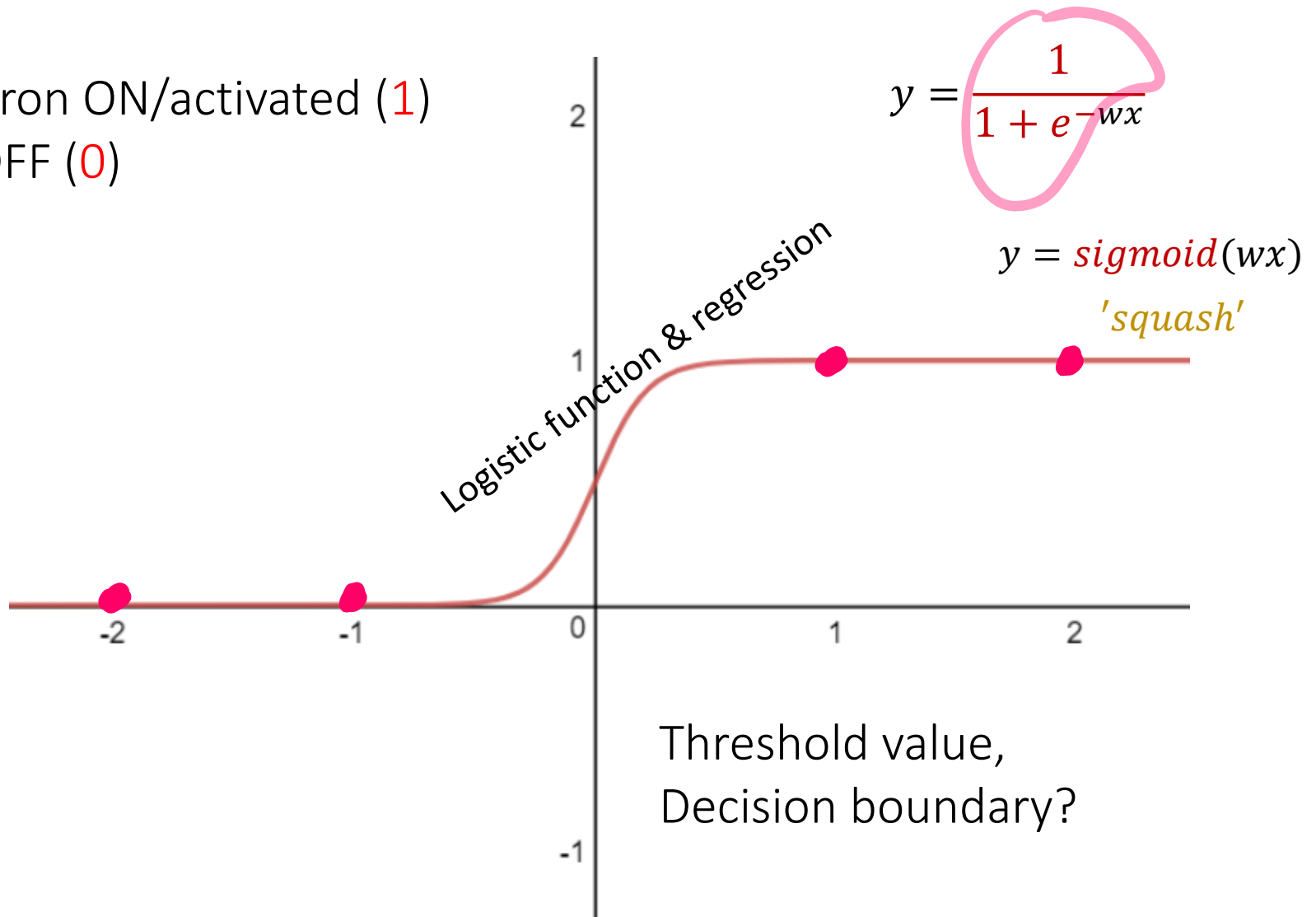
Draw $(-2, 0)$, $(-1, 0)$, $(1, 1)$, $(2, 1)$.

$$y = wx$$

$$y = \frac{1}{1 + e^{-wx}}$$



Neuron ON/activated (**1**)
or OFF (**0**)



$$y = \frac{1}{1 + e^{-wx}} \quad y = \frac{1}{1 + e^{-w(x-0)}}$$

Logistic function

From Wikipedia, the free encyclopedia

For the recurrence relation, see [Logistic map](#).

A **logistic function** or **logistic curve** is a common "S" shape ([sigmoid curve](#)), with equation:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

where

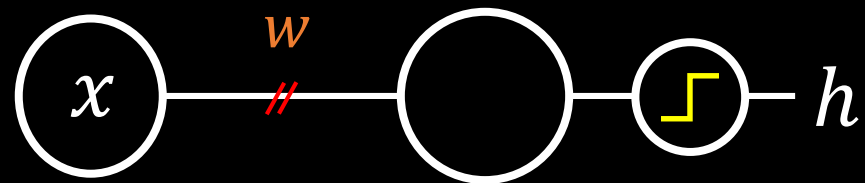
- e = the [natural logarithm](#) base (also known as [Euler's number](#)),
- x_0 = the x -value of the sigmoid's midpoint,
- L = the curve's maximum value, and
- k = the logistic growth rate or steepness of the curve.^[1]

Revisited

Real operation of a neuron

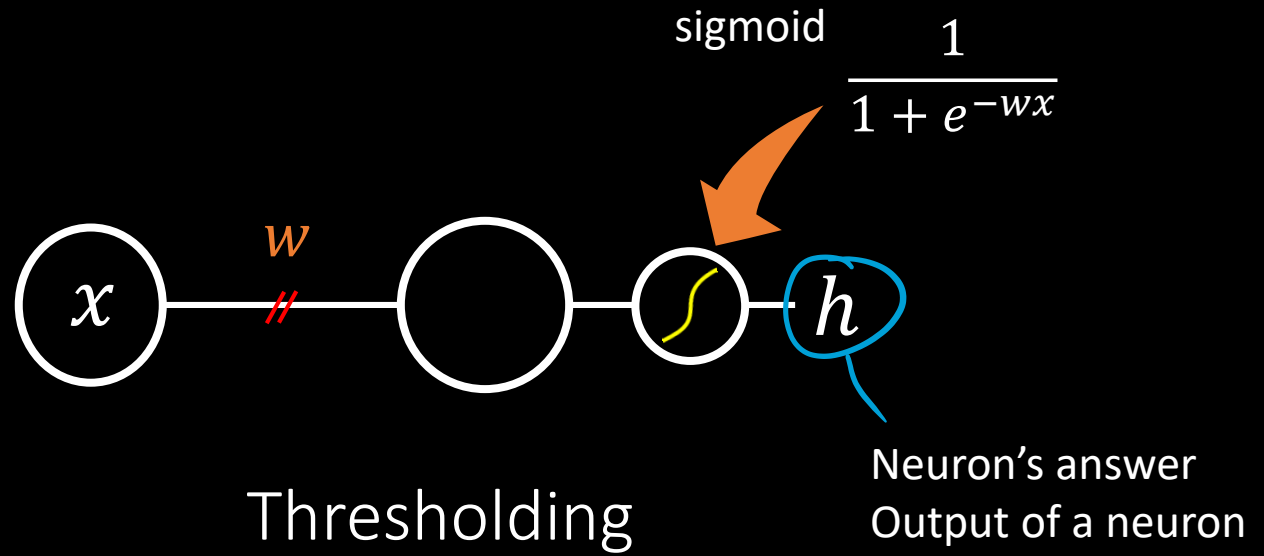
- signal **ON** if the weighted sum is greater than T
- otherwise signal **OFF**

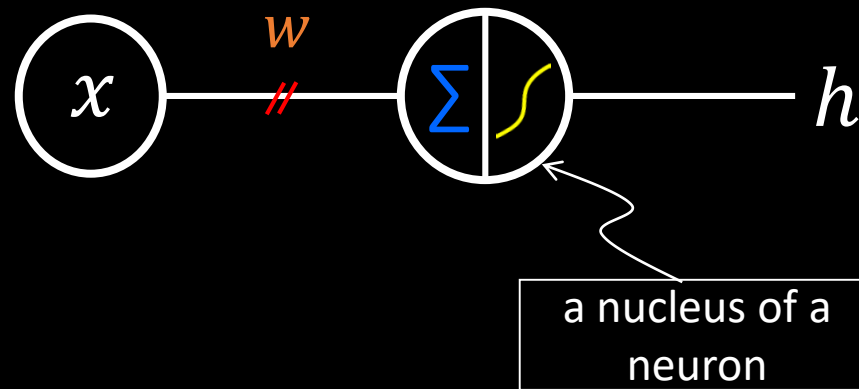
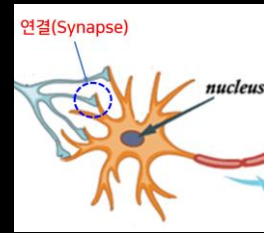
Revisited



Thresholding

Revisited



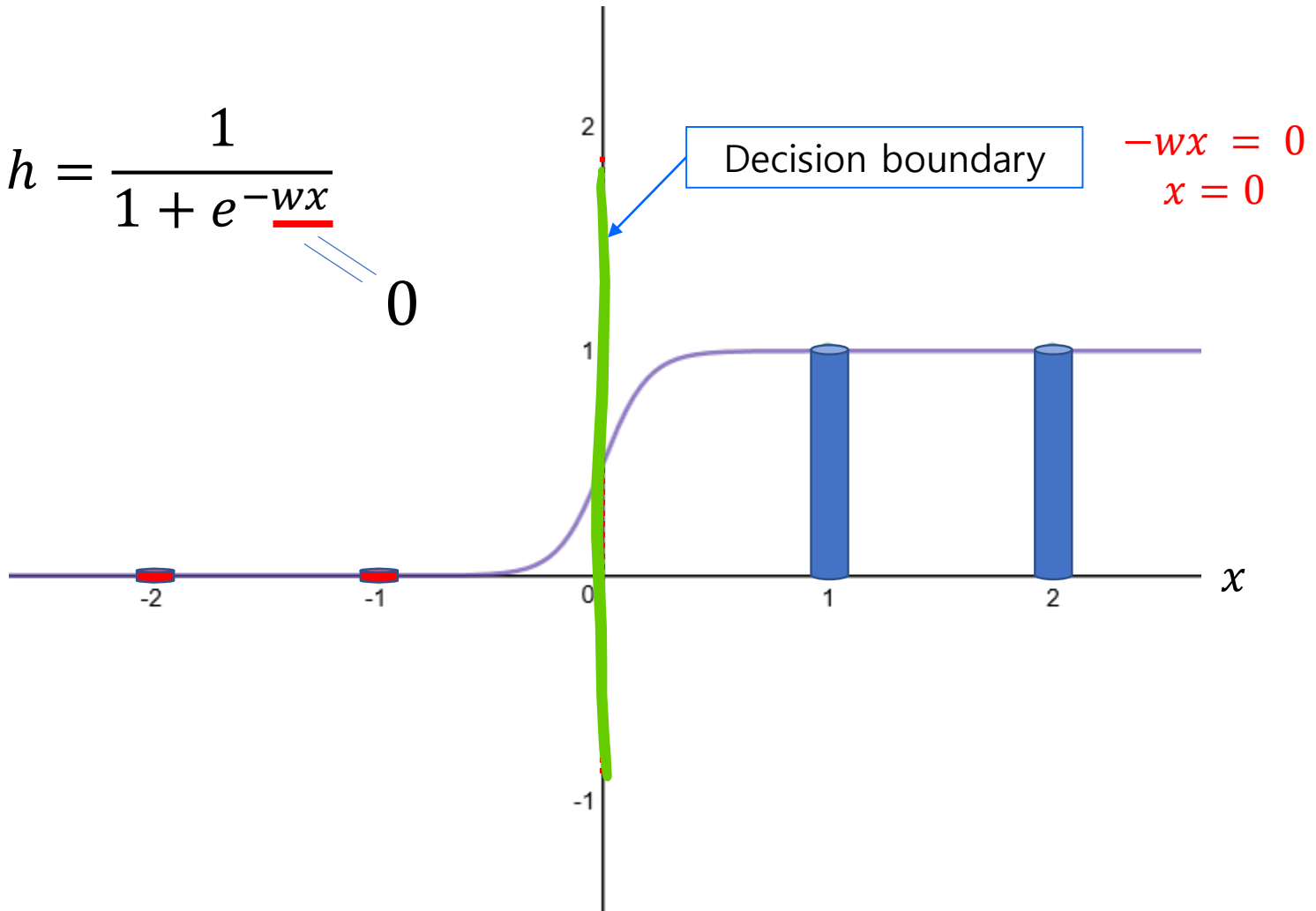


- Activated(1) or not(0)
according to the input x
- Let's find the **decision boundary** to decide 1 or 0.

Decision boundary

$$h = \frac{1}{1 + e^{-wx}}$$

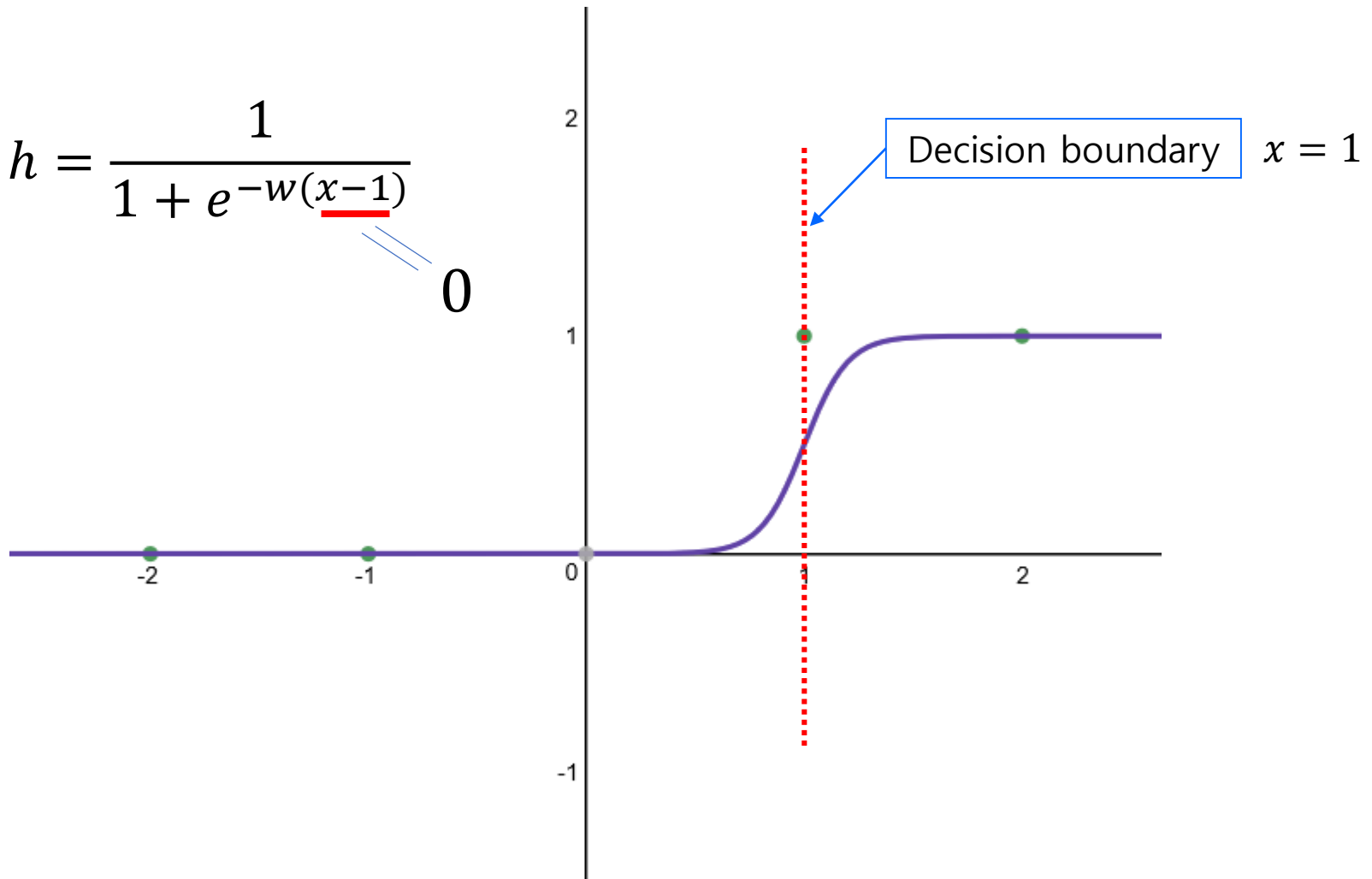
$\underline{-wx} = 0$



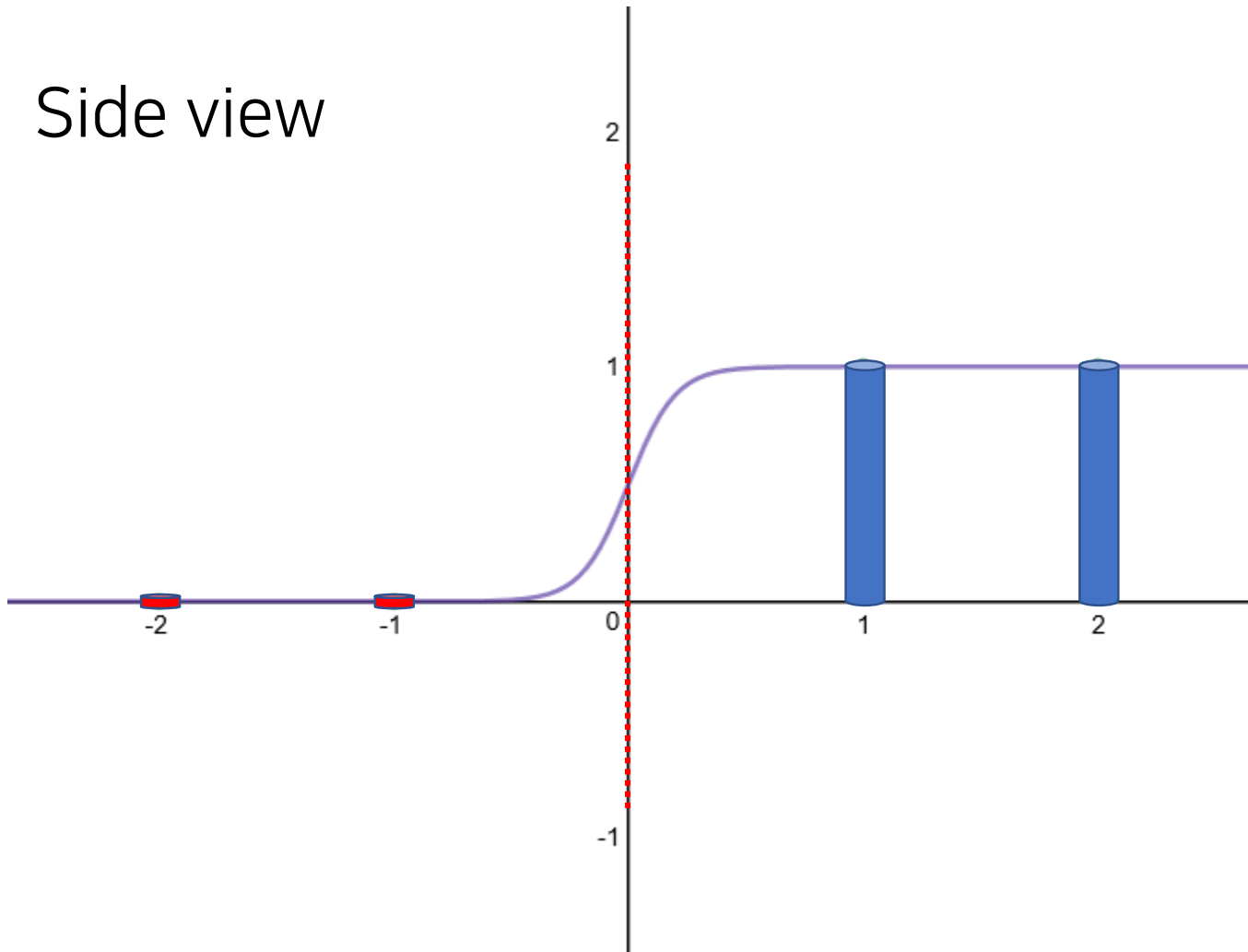
Decision boundary

$$h = \frac{1}{1 + e^{-w(x-1)}}$$

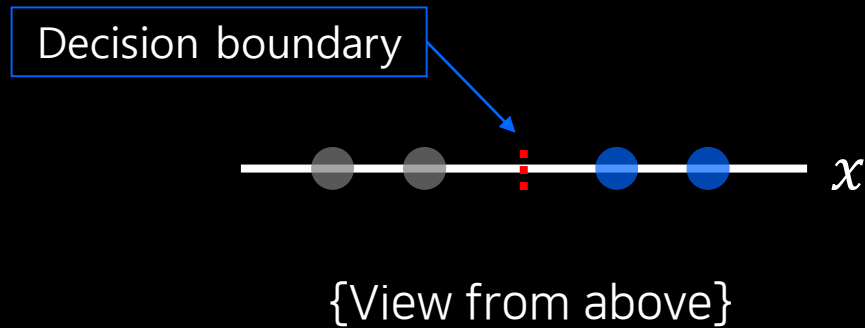
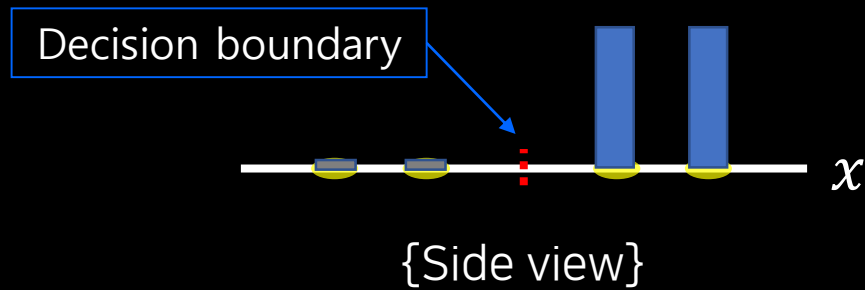
$\underline{x-1} = 0$



Side view



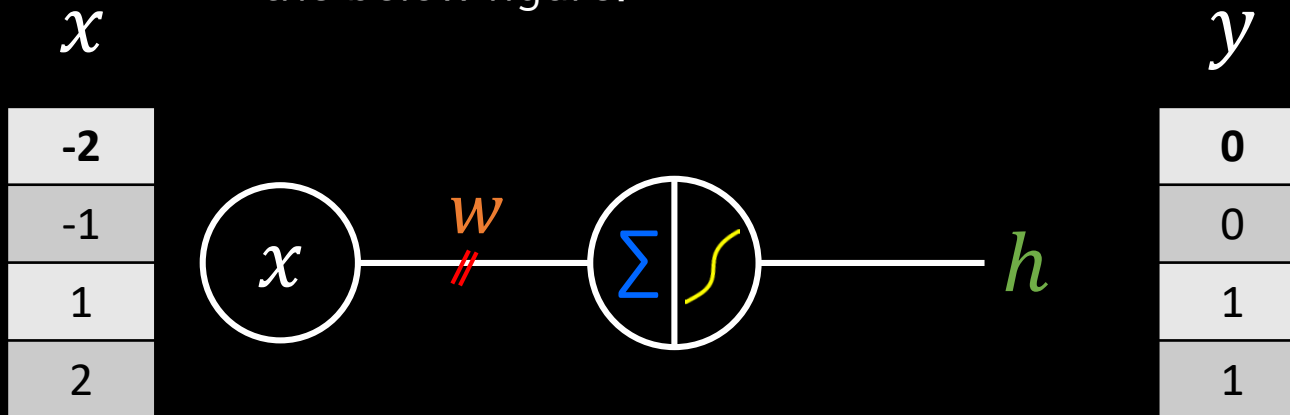
Decision Boundary



Classification

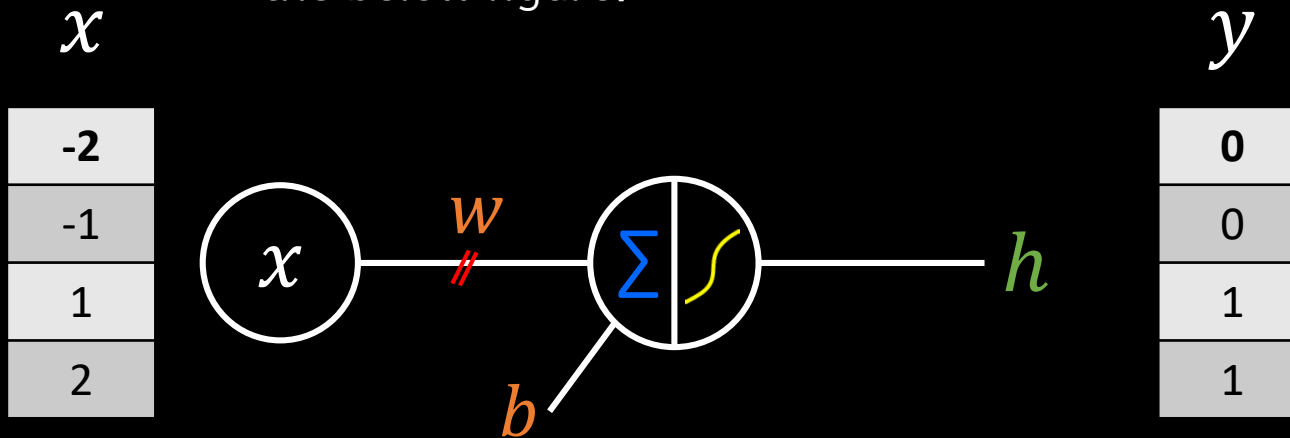
- Pass(1) or Fail(0)
- Spam(1) or Ham(0)
- Scam(fraud, 1) or not(0)
- Safe(1) or Dangerous(0)
- Intrusion/virus(1) or not(0)
- Cancer(1) or not(0)
- Binary classification -> Multiple classification

Guess the **decision boundary** from the below figure.



$$h = \begin{cases} 1 & \text{if } wx \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Guess the decision boundary from the below figure.



$$h = \begin{cases} 1 & \text{if } wx + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

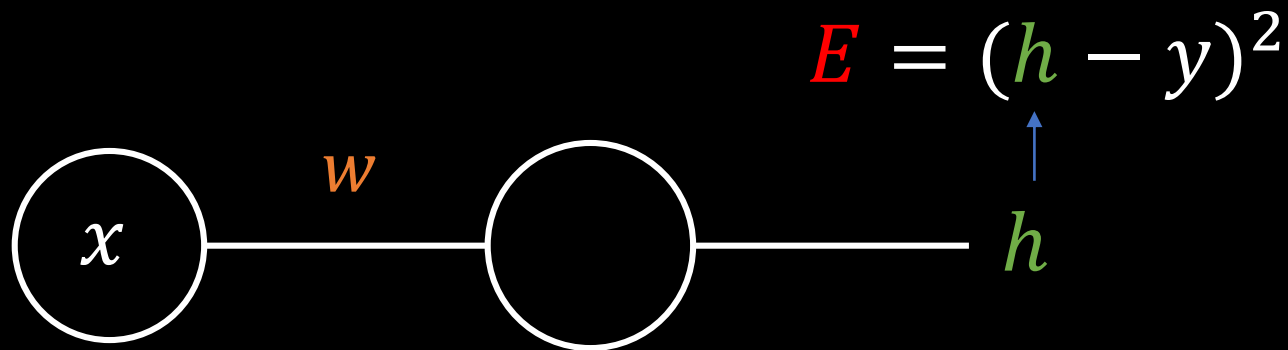
Hypothesis

- What is hypothesis? The answer of a neuron
- Find **decision boundary** from the equation.

$$h = \frac{1}{1 + e^{-wx}}$$

$$h = \frac{1}{1 + e^{-(wx+b)}}$$

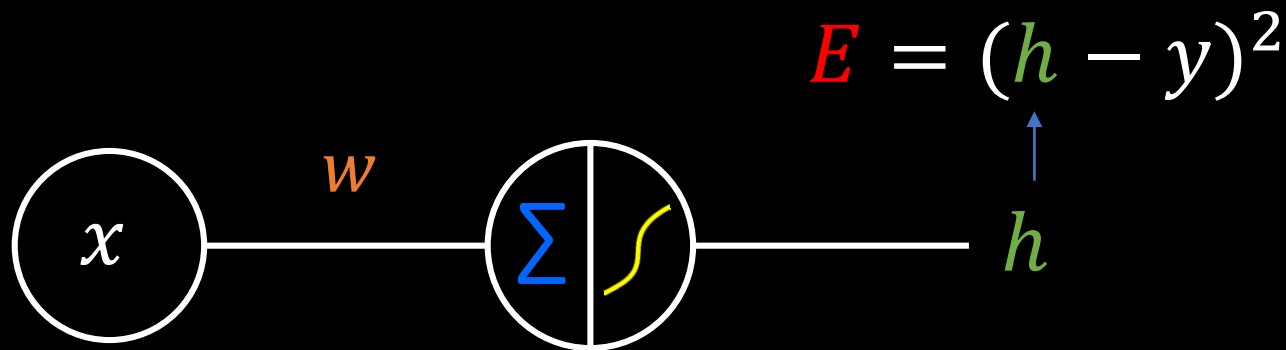
Cost/Error Function



Does MSE work?



Cost/Error Function



Does MSE work?





Draw $(-2, 0)$, $(-1, 0)$, $(1, 1)$, $(2, 1)$.

$$h = wx$$

$$h = \frac{1}{1 + e^{-wx}}$$

Draw $(1, 1)$ only.

$$E = \left(\frac{1}{1 + e^{-w \cdot 1}} - 1 \right)^2$$

$$(w, E)$$



desmos

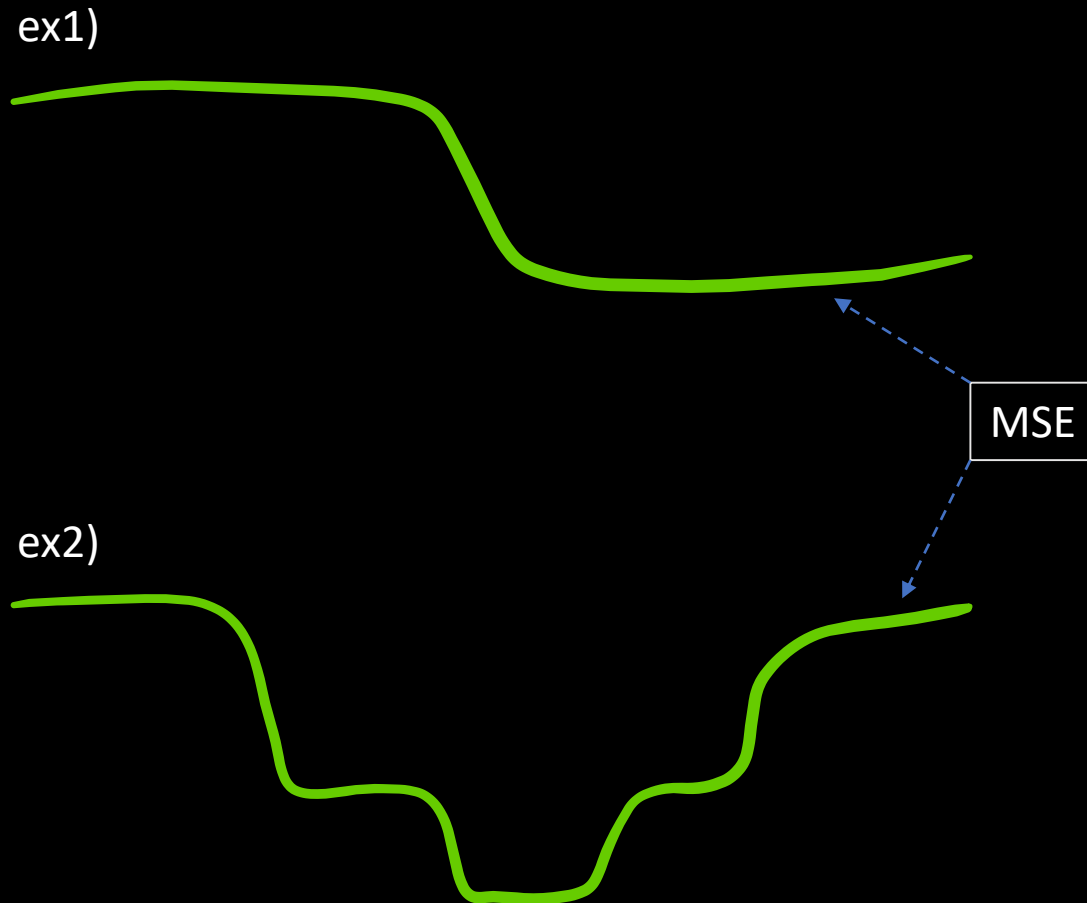
Draw 4 points: $(-1, 0)$, $(1, 1)$, $(-3, 0)$, $(3, 1)$.

$$E = \left(\frac{1}{1 + e^{-w(-1)}} - 0 \right)^2 + \left(\frac{1}{1 + e^{-w(1)}} - 1 \right)^2 + \\ \left(\frac{1}{1 + e^{-w(-3)}} - 0 \right)^2 + \left(\frac{1}{1 + e^{-w(3)}} - 1 \right)^2$$

Add bias b .

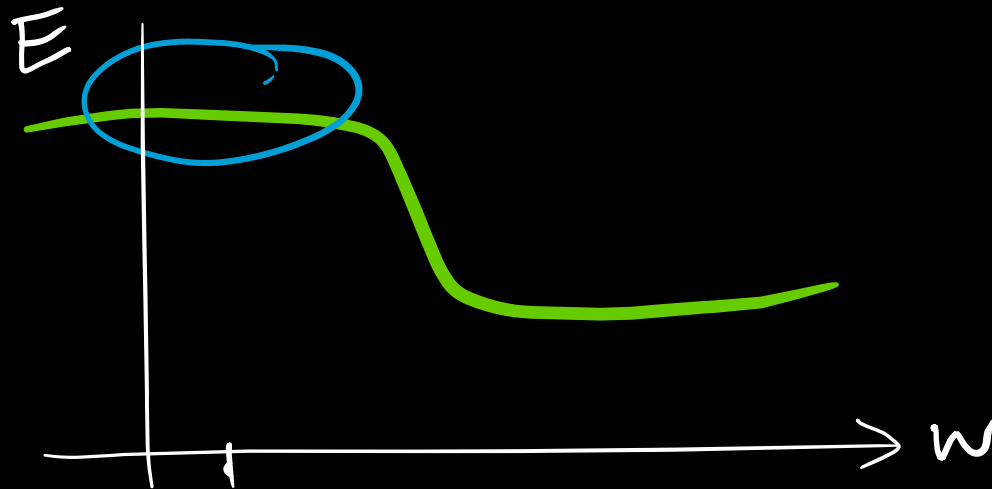
plot the error: $\left(w, \frac{E}{2} \right)$

Cost/Error Function



What problem in the error function?

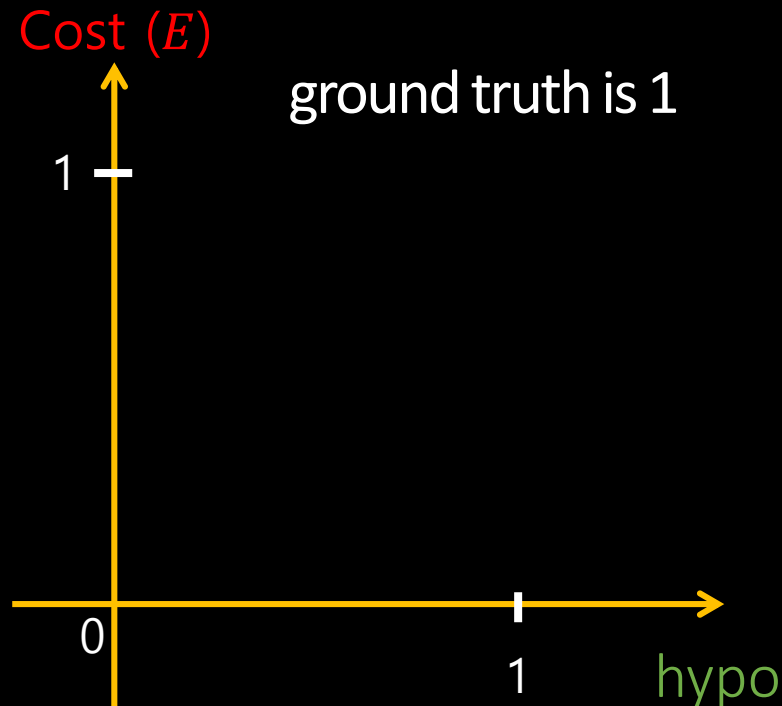
No gradient decent
in some parts



New Cost/Error Function

When ground truth is 1

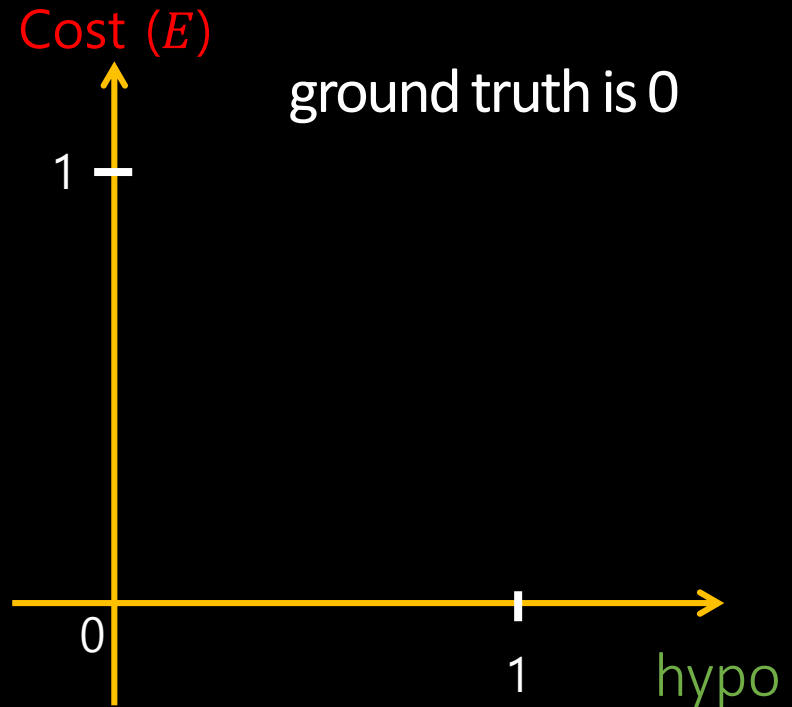
- if **hypo** is equal to 1, then error = 0
- if **hypo** is equal to 0 then error = ∞



New Cost/Error Function

When ground truth is 0

- if **hypo** is equal to 0, then error = 0
- if **hypo** is equal to 1 then error = ∞





desmos

$$E = -\log(h)$$

$$E = -\log(1 - h)$$

$$E = -\log\left(\frac{1}{1 + e^{-wx}}\right)$$

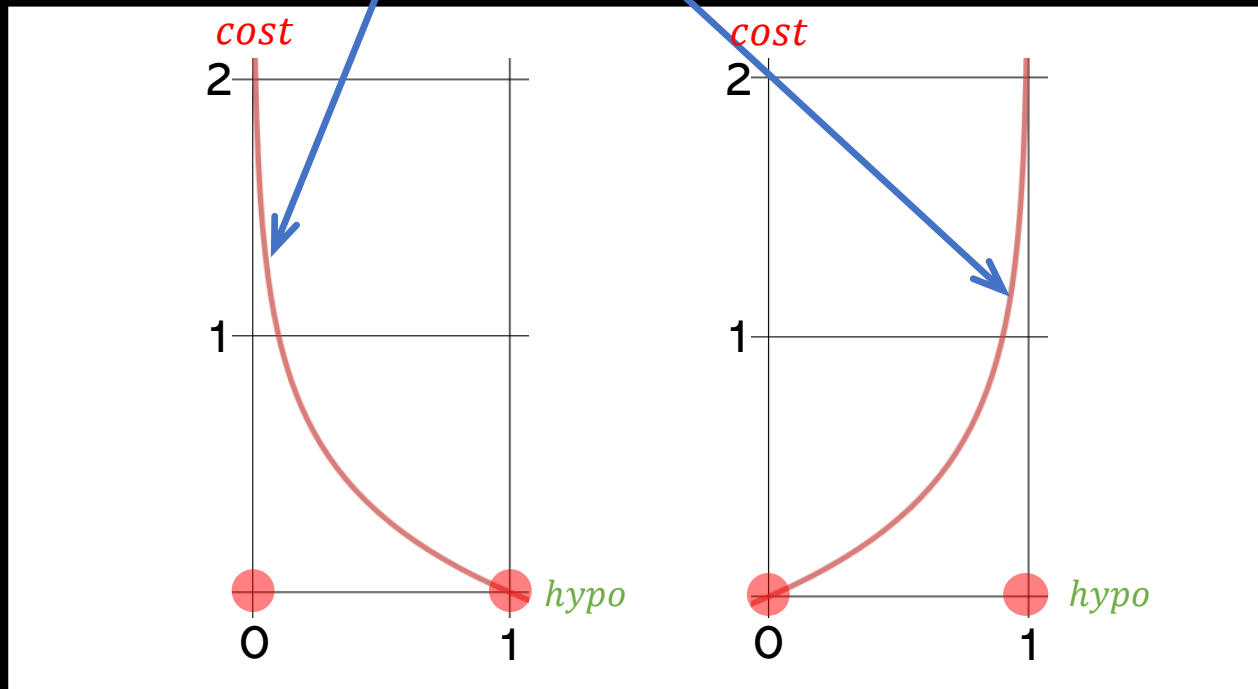
$$E = -\log\left(1 - \frac{1}{1 + e^{-wx}}\right)$$

New Cost/Error Function

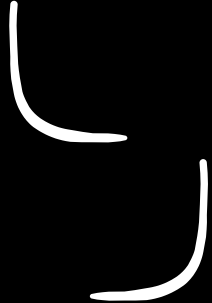
$$E = \begin{cases} -\log(h) & : y = 1 \\ -\log(1 - h) & : y = 0 \end{cases}$$


Prediction by a neuron

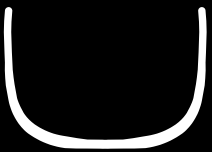
Correct answer



New Cost/Error Function

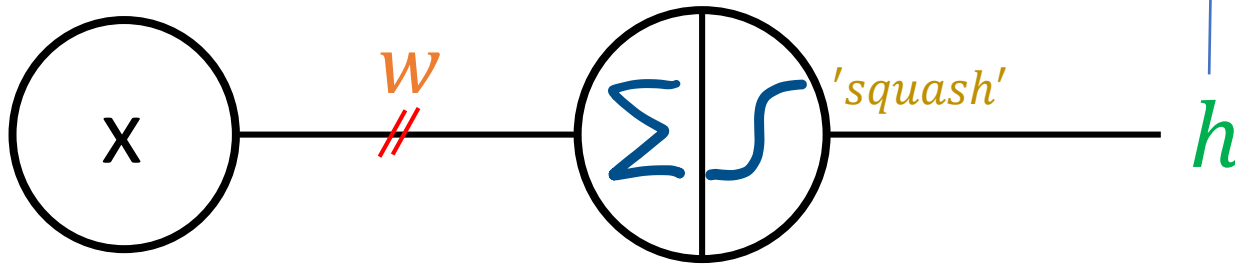
$$E = \begin{cases} -\log(wx) & : y = 1 \\ -\log(1 - wx) & : y = 0 \end{cases}$$



$$E = -y \log(wx) - (1 - y) \log(1 - wx)$$

$$E = -(y \log(wx) + (1 - y) \log(1 - wx))$$


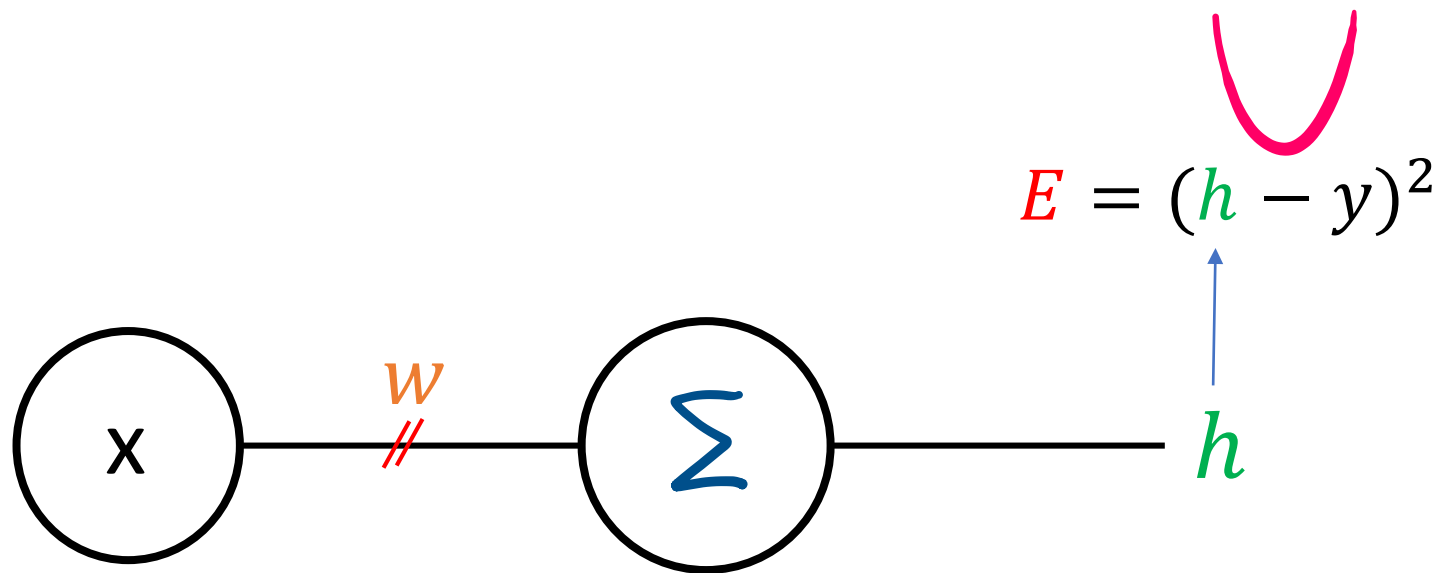
$$w = w - \alpha \cdot \frac{\partial E}{\partial w}$$

$$E = -(y \log(h) + (1 - y) \log(1 - h))$$



$$E = -\frac{1}{N} \sum (y \log(h) + (1 - y) \log(1 - h))$$

Binary Cross-Entropy Loss

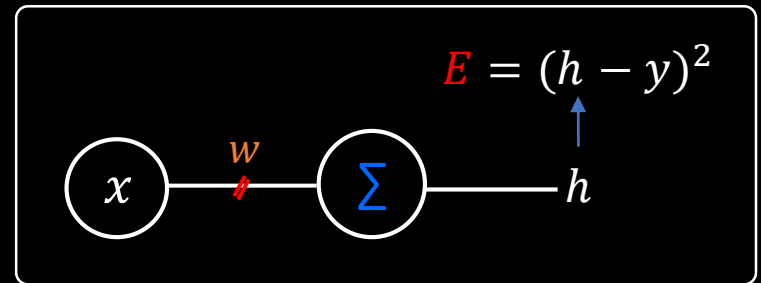
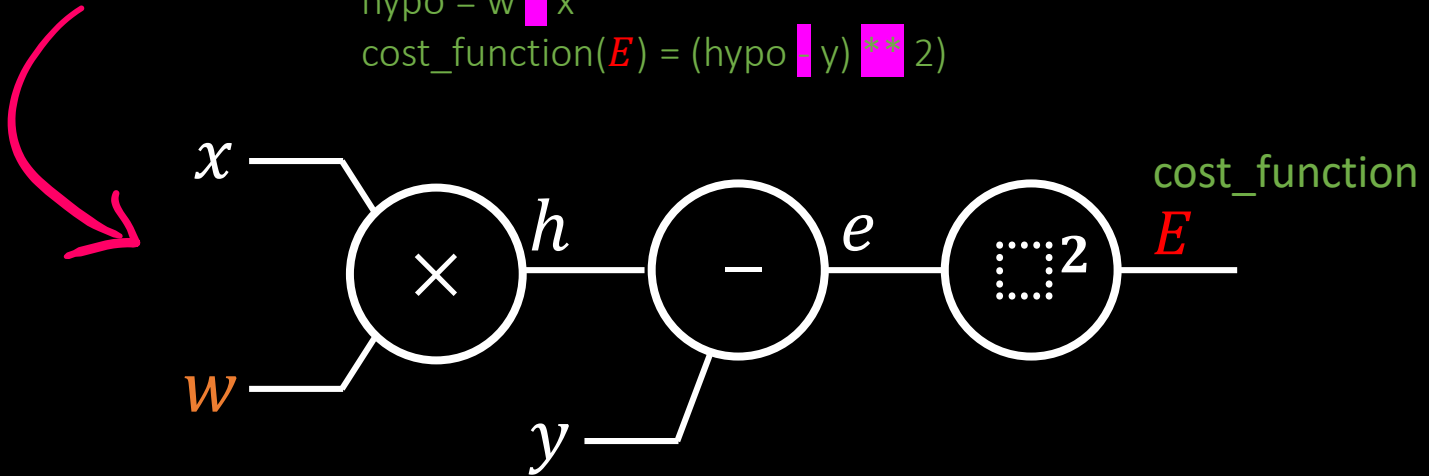


Computational graph
for the new cost function

Computational Graph

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

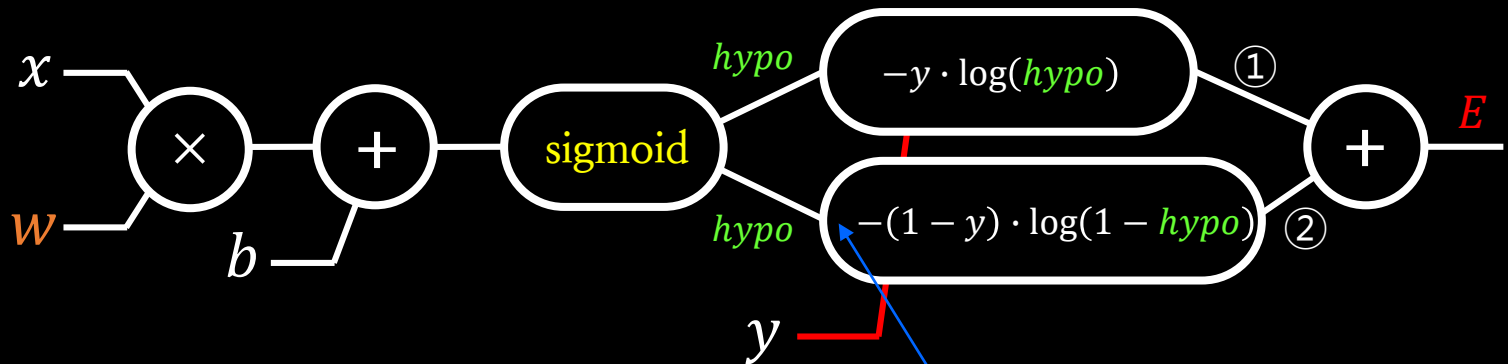
hypo = w * x
cost_function(E) = (hypo - y) ** 2



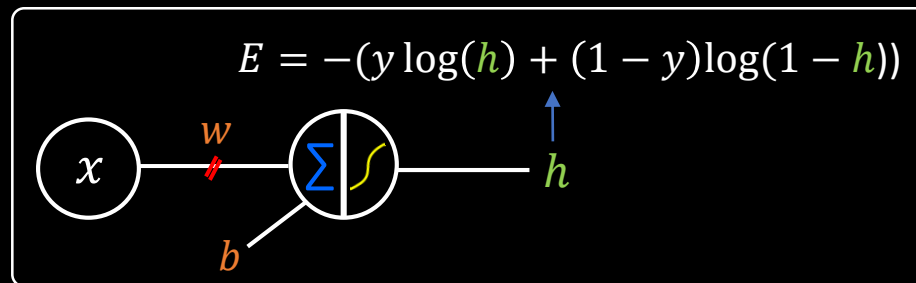
Computational Graph

$$E = \overset{\textcircled{1}}{-y \cdot \log(\text{hypo})} - \overset{\textcircled{2}}{(1 - y) \cdot \log(1 - \text{hypo})}$$

Binary Cross-Entropy Loss

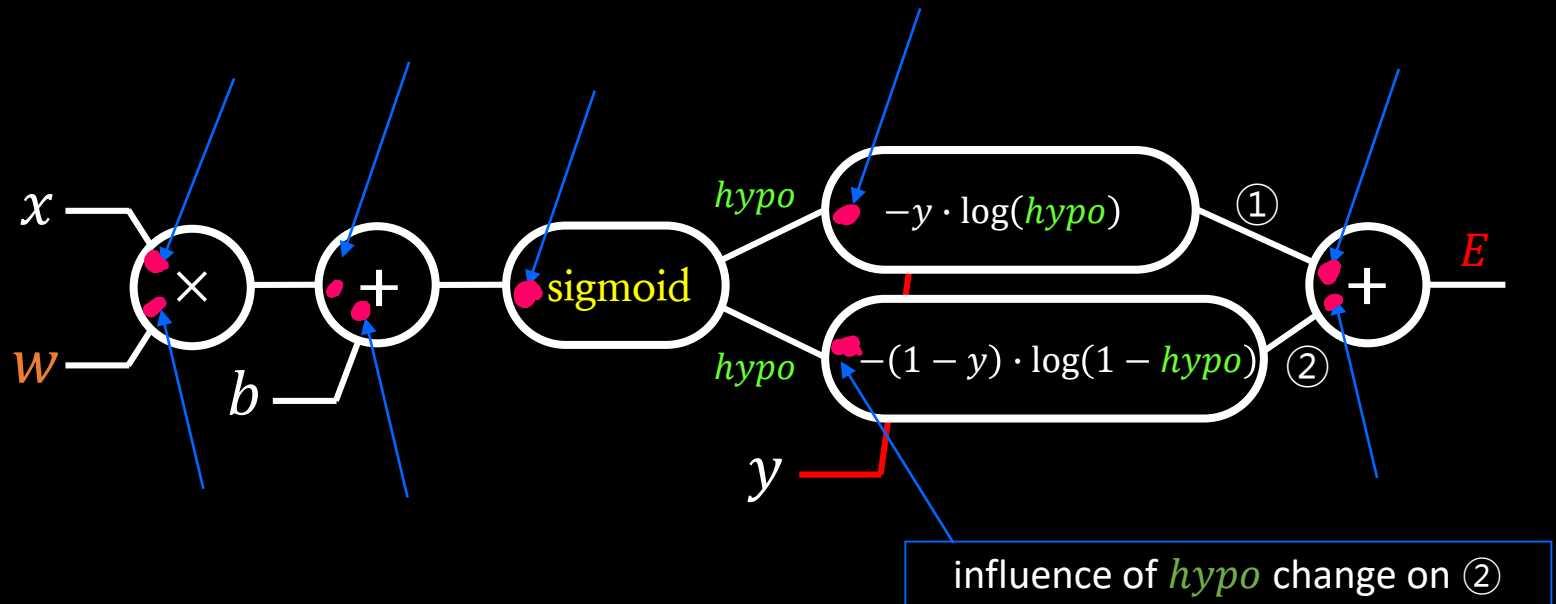


influence of *hypo* change on ②



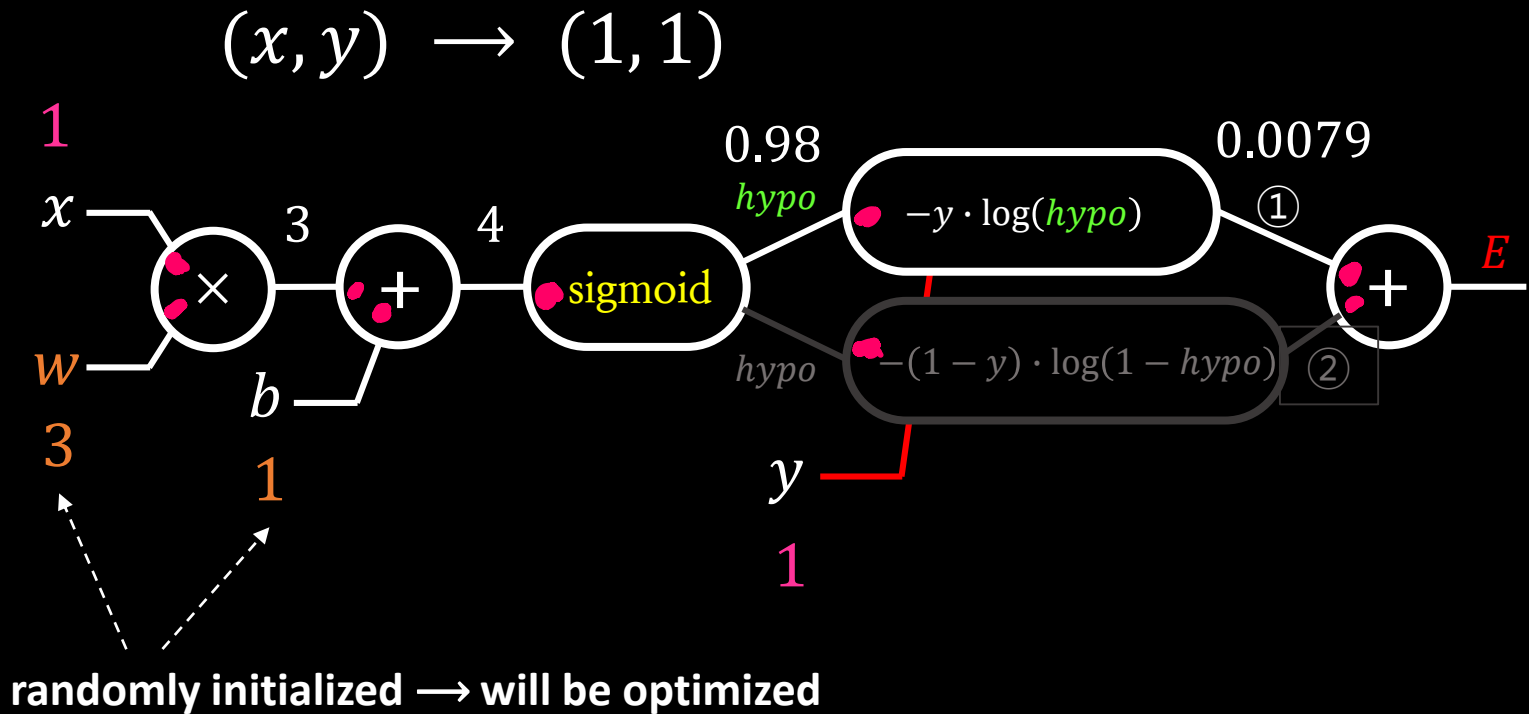
$$\frac{\partial \textcircled{2}}{\partial h}$$

Local Gradients

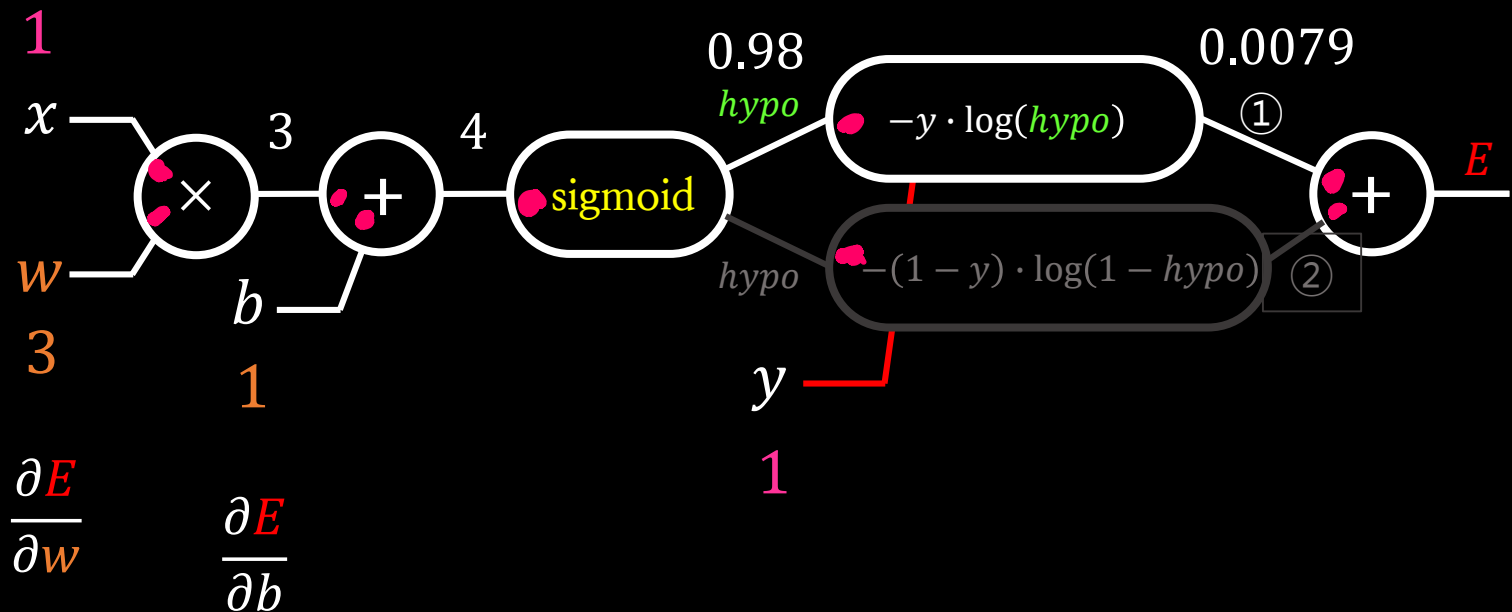


$$\frac{\partial ②}{\partial h}$$

Forward propagation



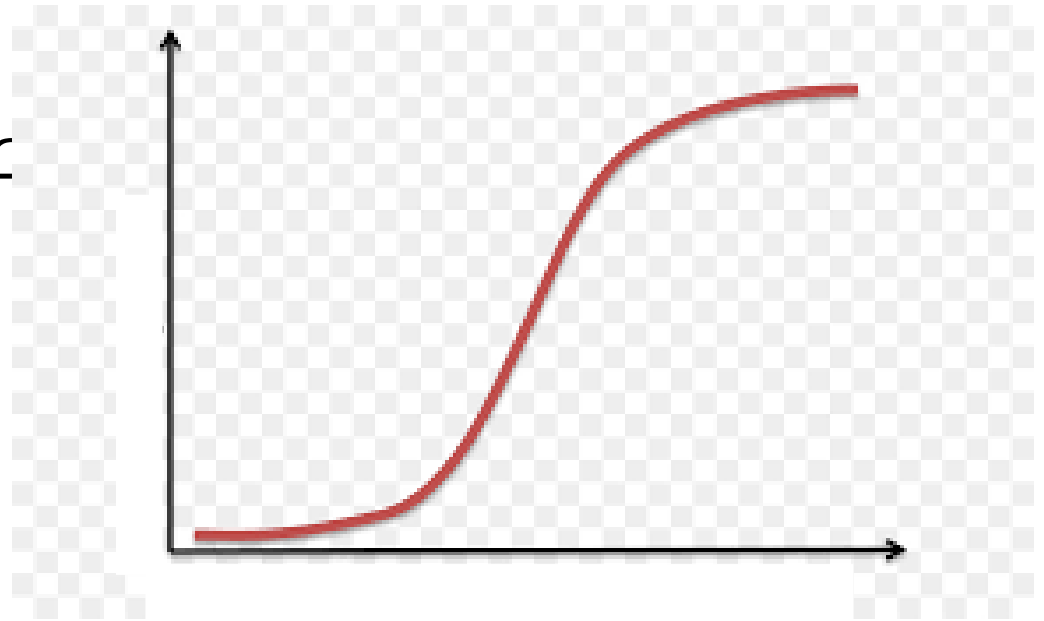
Back-propagation

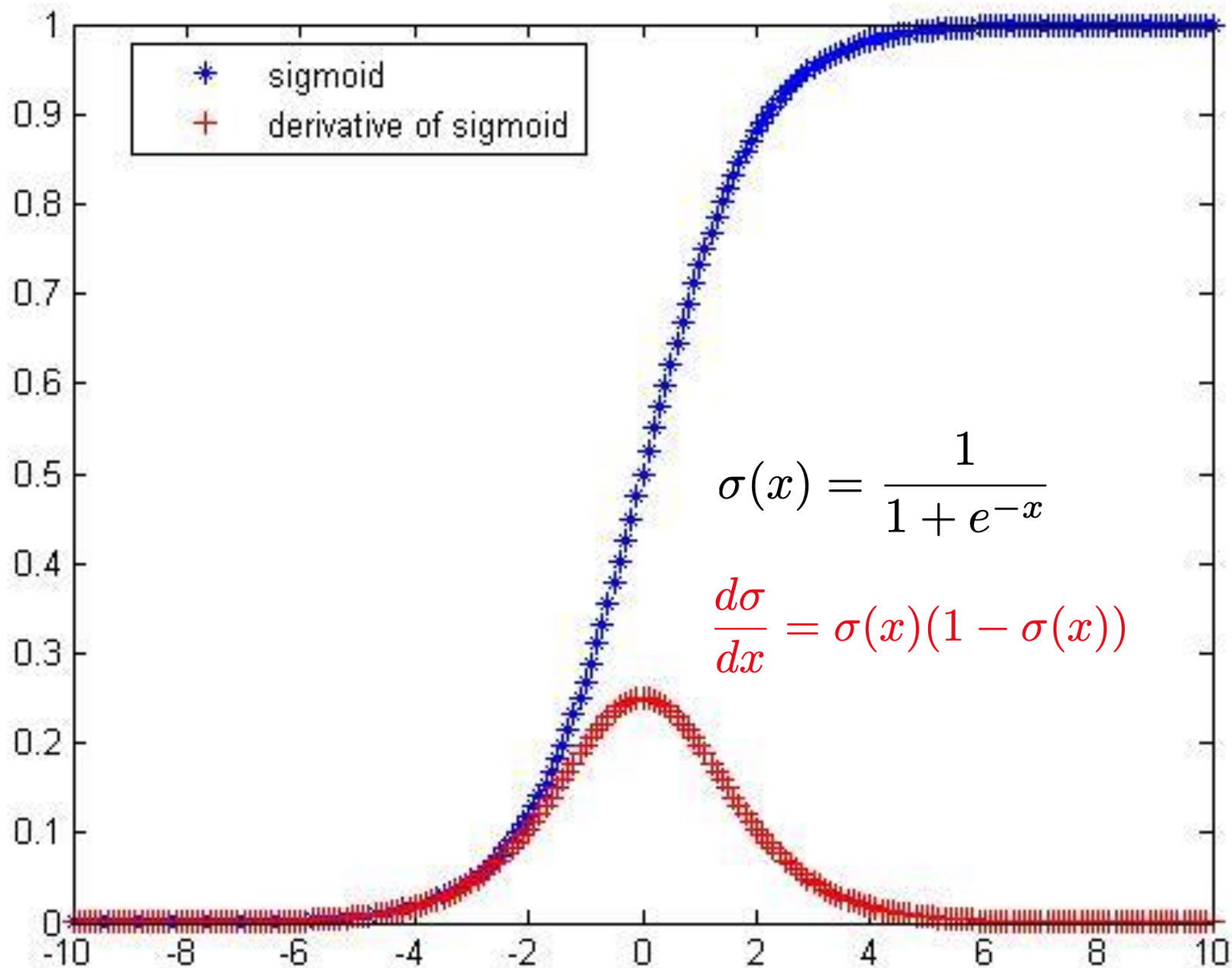


$$w = w - \alpha \cdot \frac{\partial E}{\partial w}$$

$$b = b - \alpha \cdot \frac{\partial E}{\partial b}$$

Derivative of
Sigmoid





Desmos.com

Parameters(w, b) tuning
for what?

decision boundary

$$wx + b = 0$$

for better decision boundary

Lab 11.py

Classification of
an input as 1 or 0

$$cost = -(y \log(H(X)) + (1 - y) \log(1 - H(X)))$$

```
x_data = [-2., -1, 1, 2]
y_data = [0., 0, 1, 1]
```

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

```
#----- learning
cost = -tf.reduce_mean(y_data * tf.log(hypo) +
                        tf.subtract(1., y_data) * tf.log(tf.subtract(1., hypo)))

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

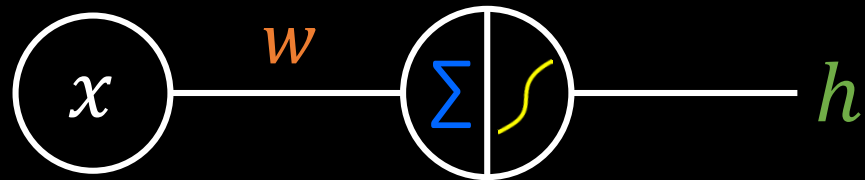
for step in range(5001):
    sess.run(train)
```

```
#----- testing(classification)
predicted = tf.cast(hypo > 0.5, dtype=tf.float32)
p = sess.run(predicted)
print("Predicted: ", p)
```

Lab 12.py
With a bias

1-Input Neuron

Guess a decision boundary.

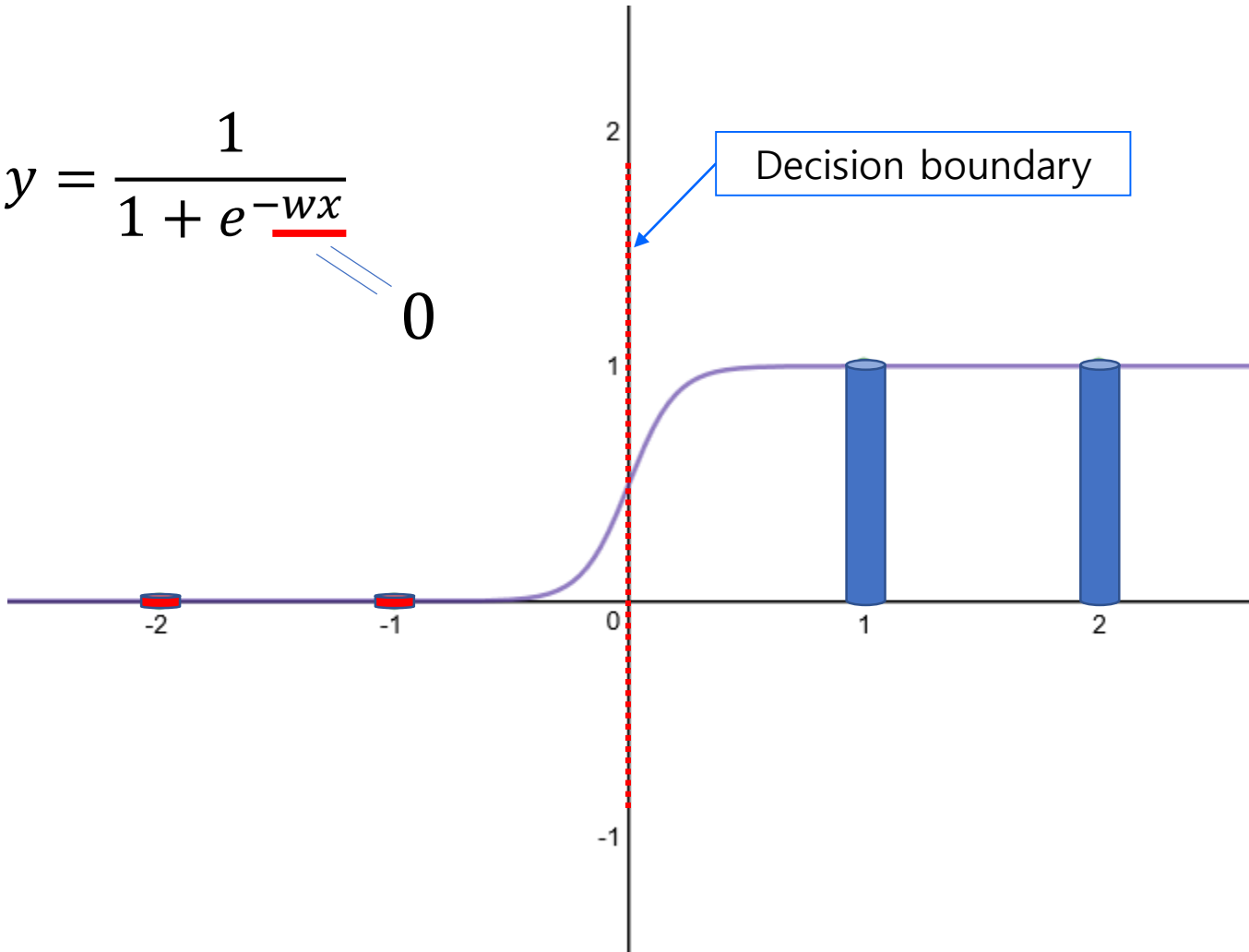


$$h = \frac{1}{1 + e^{-(wx)}}$$

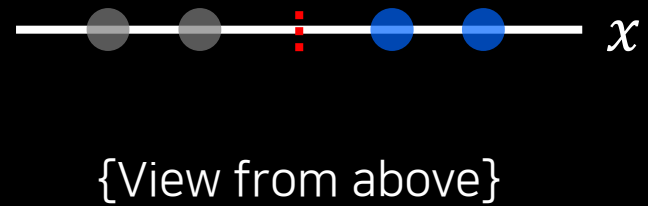
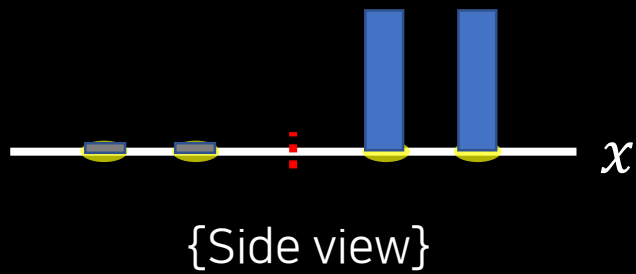
1-Input Neuron

$$y = \frac{1}{1 + e^{-wx}}$$

$-wx$ \Rightarrow 0

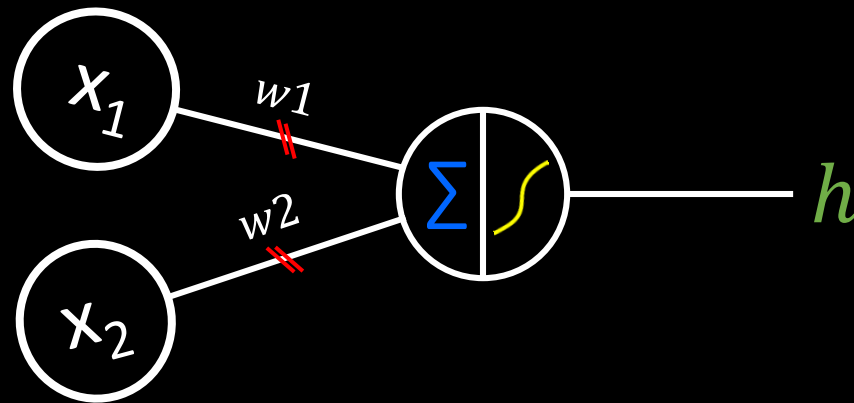


1-Input Neuron



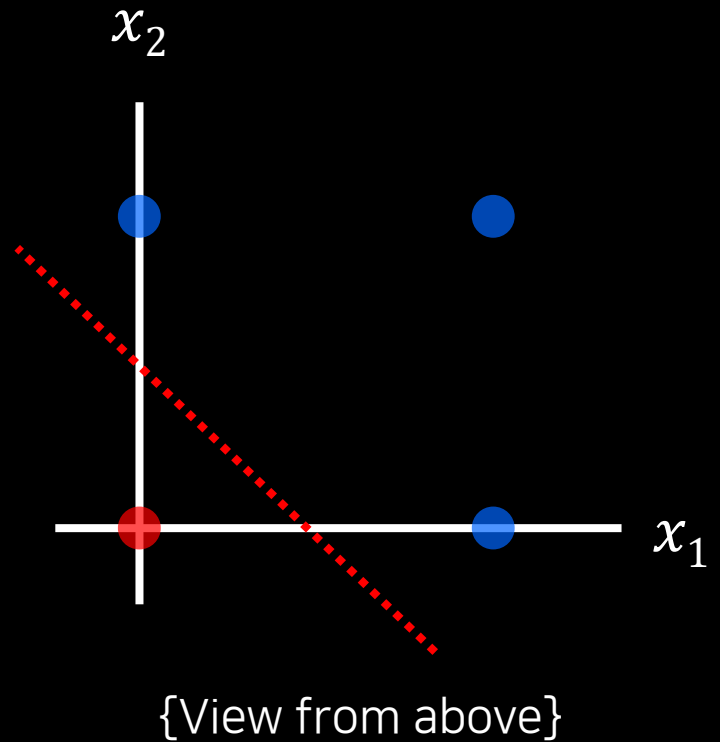
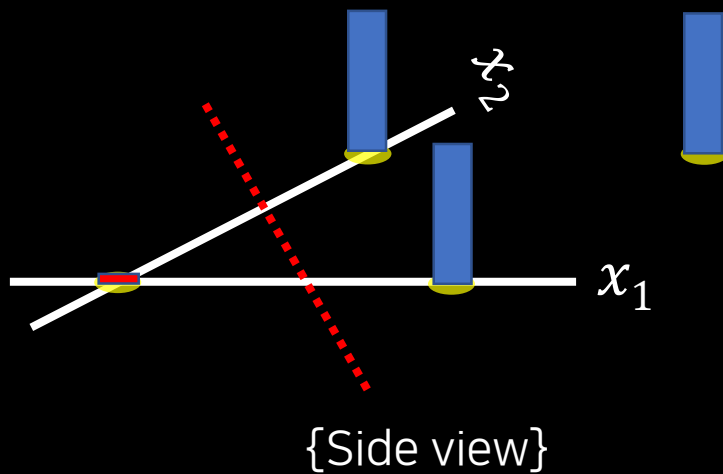
2-Input Neuron

Guess a decision boundary.



$$h = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}$$

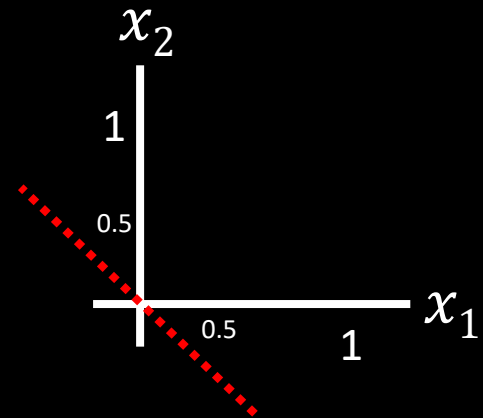
2-Input Neuron



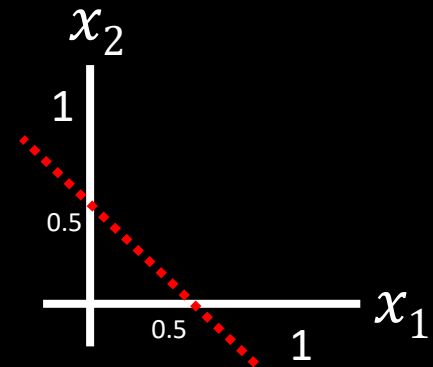
2-Input Neuron

$$x_1 + x_2 = 0$$

View from above

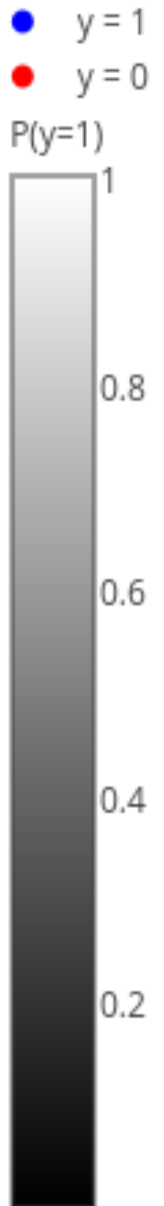
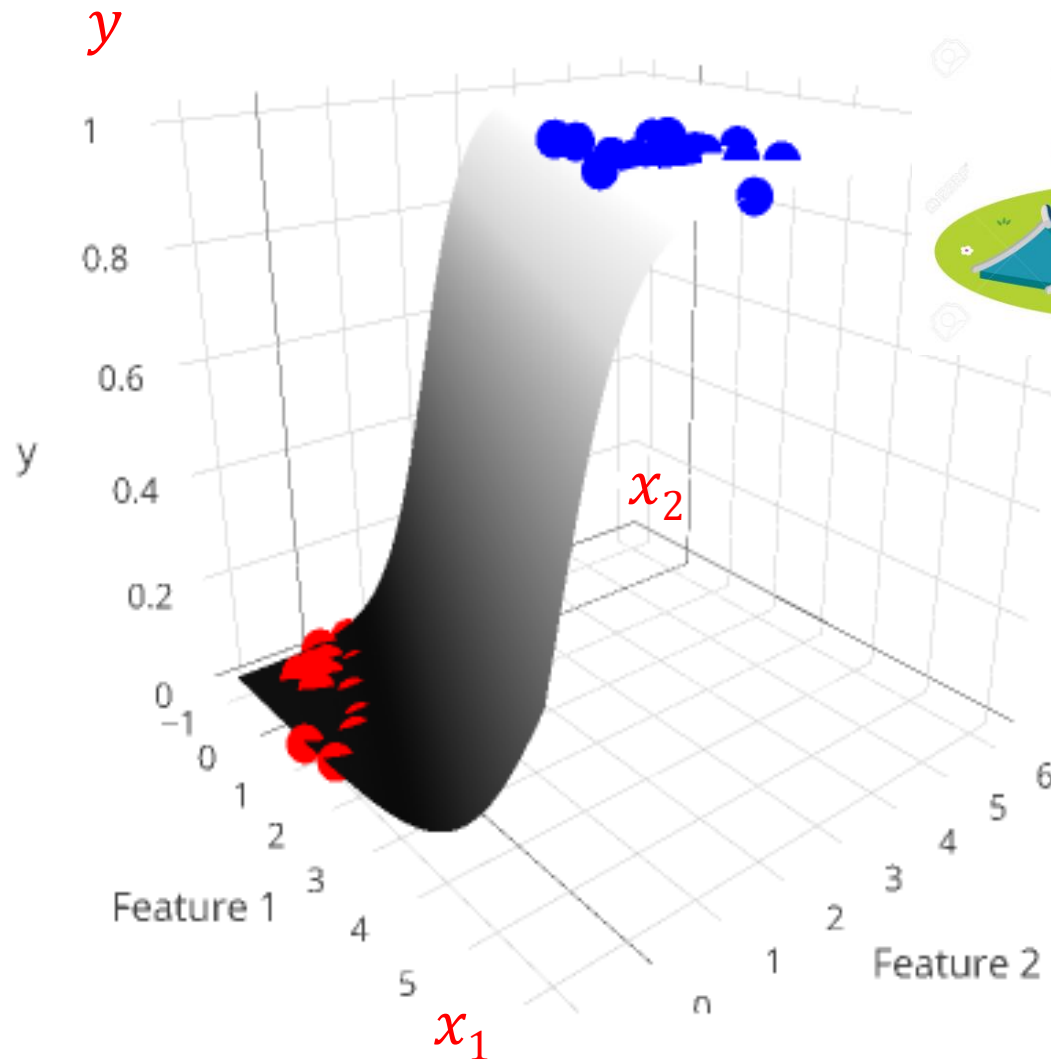


$$x_1 + x_2 - 0.5 = 0$$



Logistic Regression: 2 Features (Inputs)

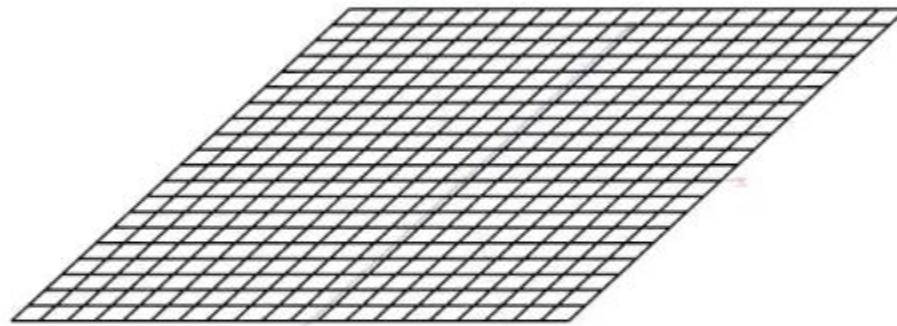
{Side view} looks like a slide!



The meaning of parameters

```
sigmoid(w1·length + w2·width + b)
```

$$w_1x_1 + w_2x_2 + b = 0$$

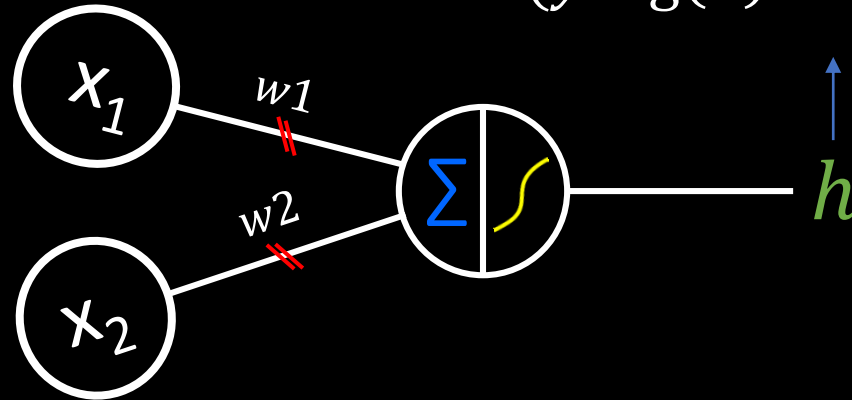


```
surface(f(x,z)=sig(w1·x+w2·z+b))  
w1 = 0.00  
w2 = 0.00  
b = 0.00
```

Lab 13.py

Implementation
of OR gate with a
neuron(a decision
boundary)

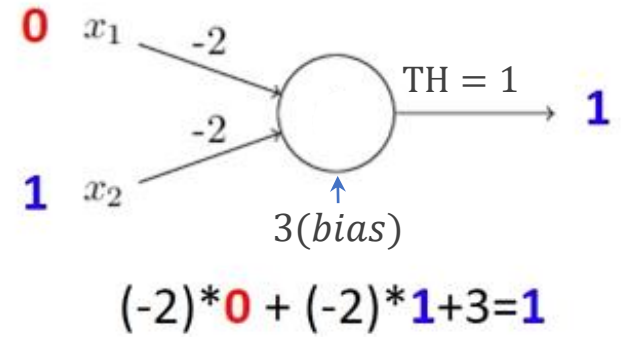
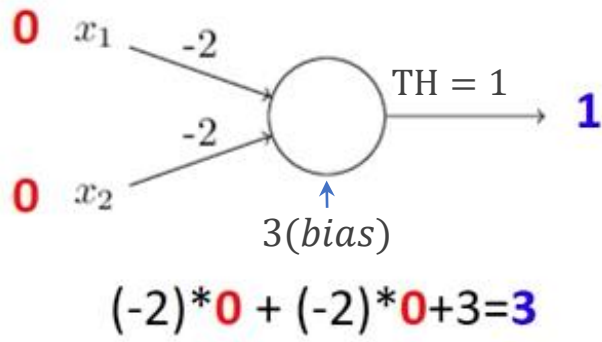
$$E = -(y \log(h) + (1 - y) \log(1 - h))$$



x_1	x_2	$AND(h)$
0	0	0
0	1	0
1	0	0
1	1	1

NAND

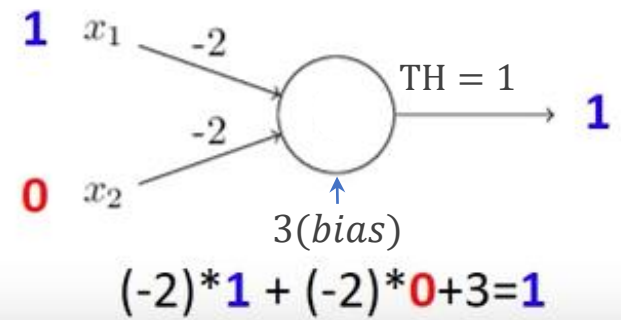
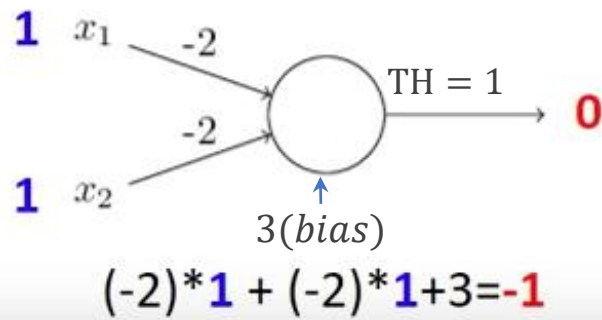
- NAND gates are functionally complete.
- We can build any logical functions out of them.



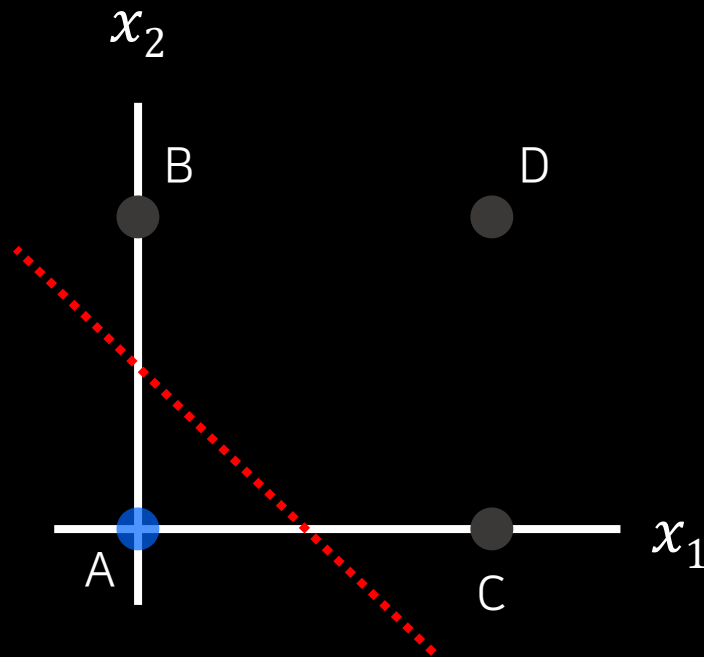
NAND

Truth Table

Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0



Decision boundary by a neuron



View from above

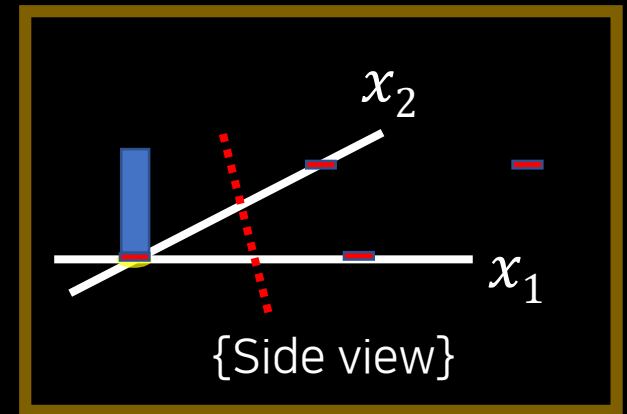
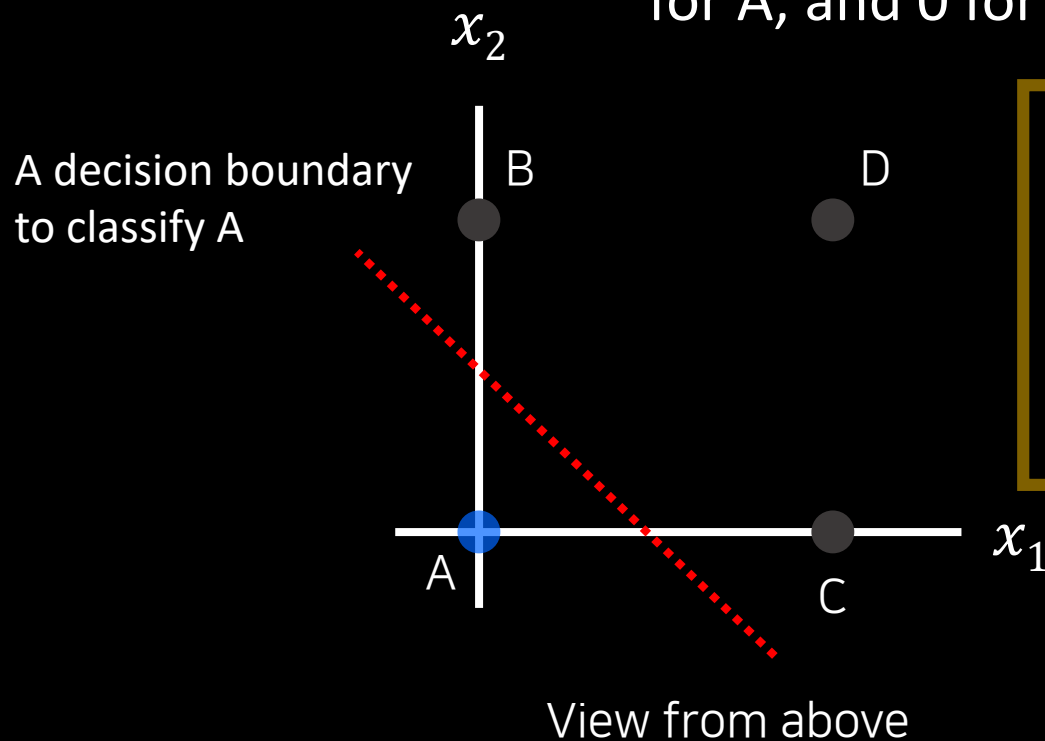
Decision boundary by a neuron

- A neuron, only 1 linear decision boundary
- A decision boundary yielding 2 classes (1 or 0)
- How to solve multiple classes more than 2

4-Class (A, B, C, D) Classification

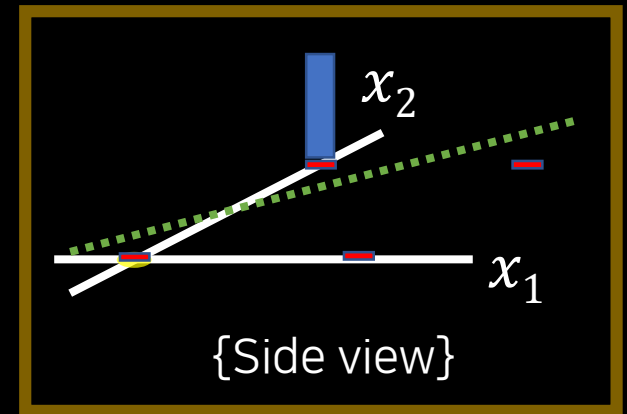
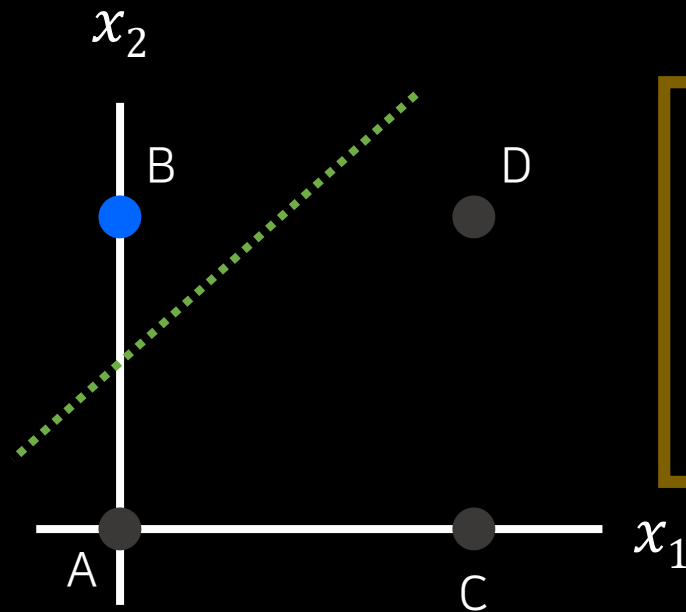
Neuron #1

The output of a neuron is 1 for A, and 0 for other cases.



Neuron #2

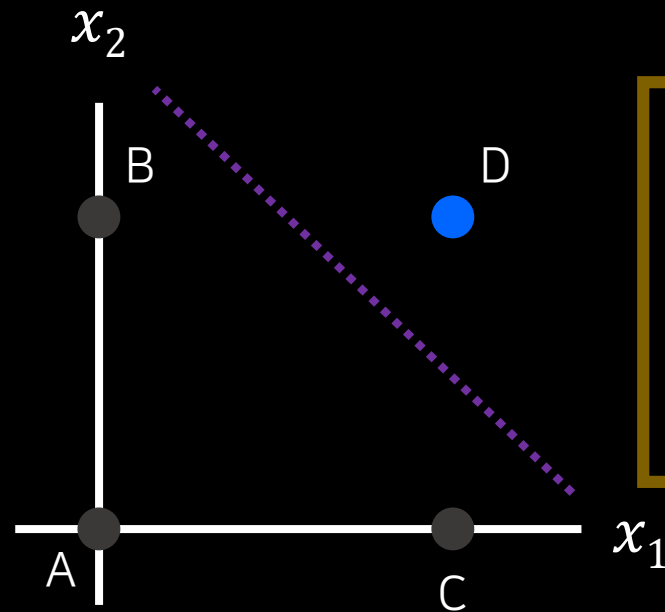
2nd neuron for 2nd decision
boundary to classify B



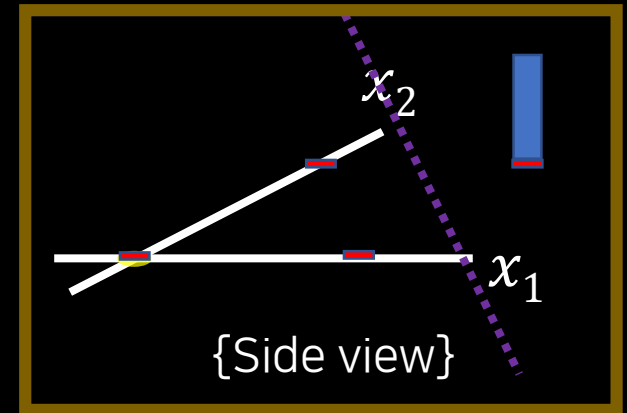
View from above

Neuron #3

3rd neuron for 3rd decision
boundary to classify D

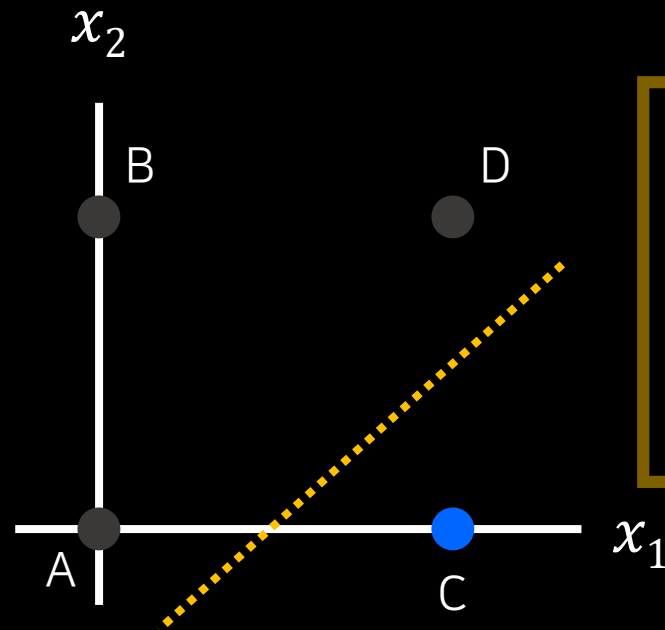


View from above

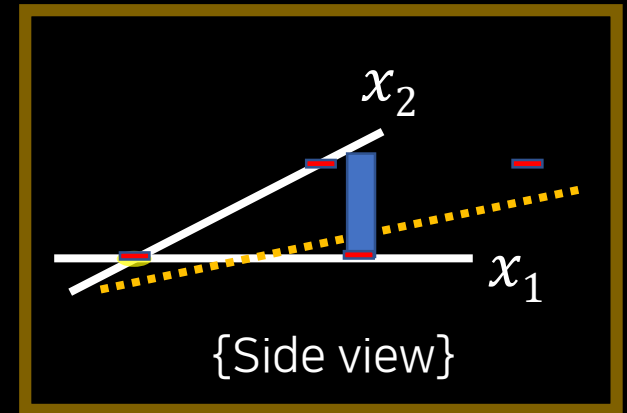


Neuron #4

4th neuron for 4th decision
boundary to classify C

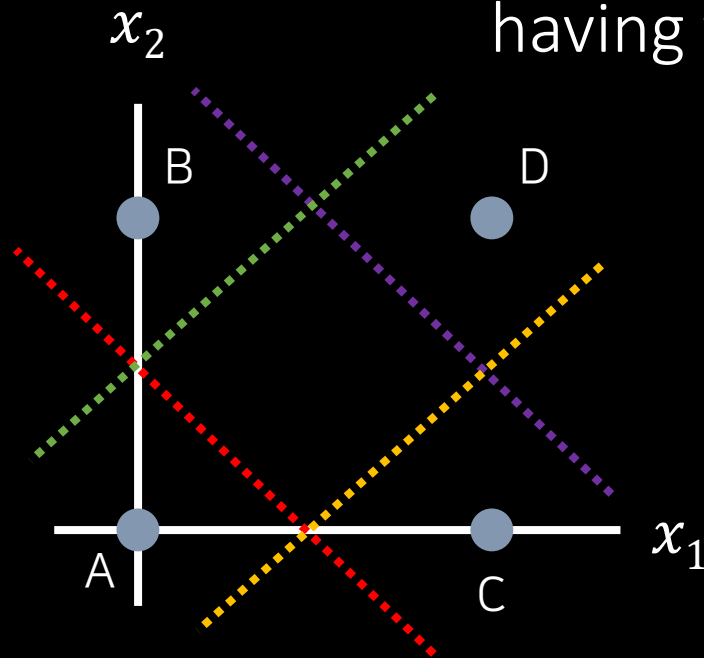


View from above



4 Neurons

4 neurons for
4 decision boundaries
having the same inputs

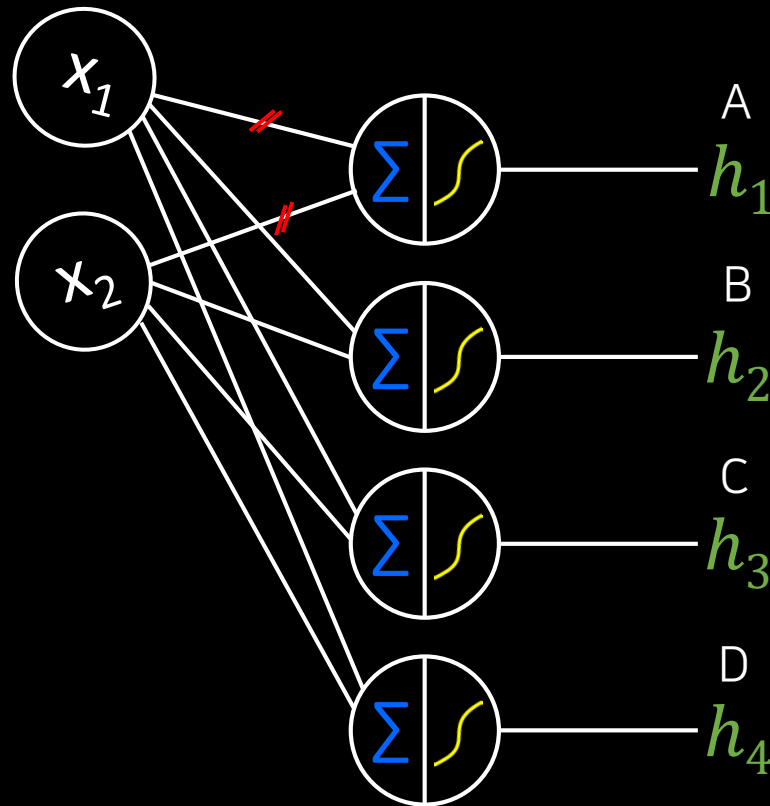


View from above

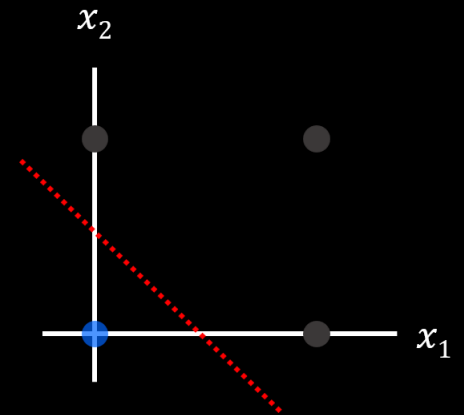
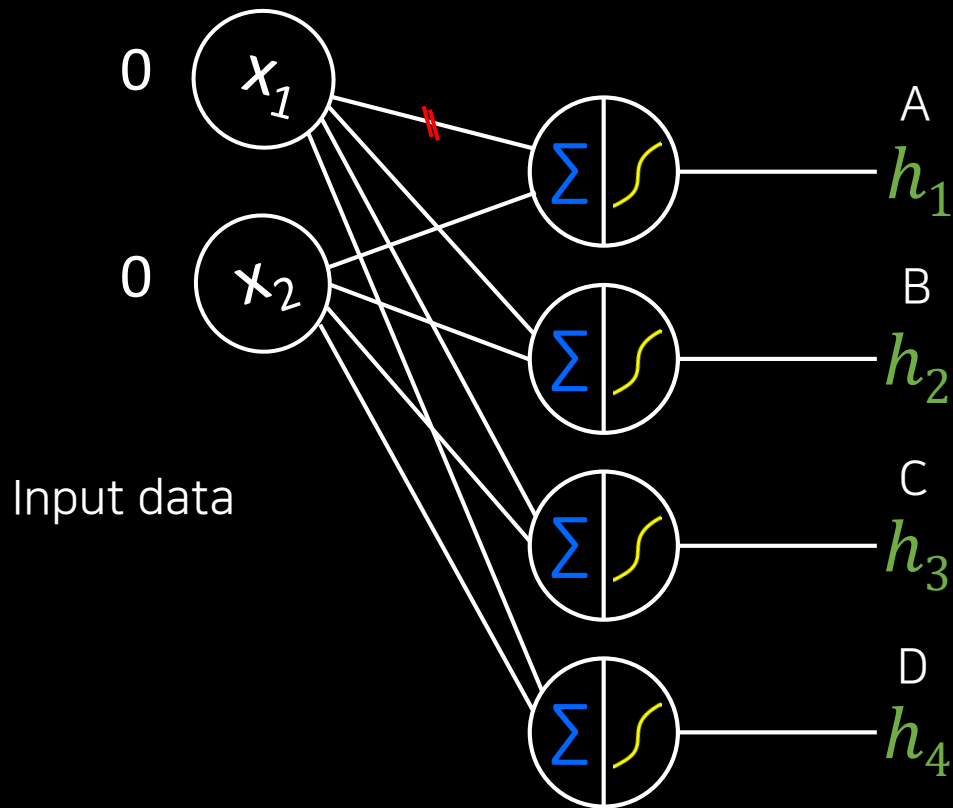
4 Neurons

Matrix notation

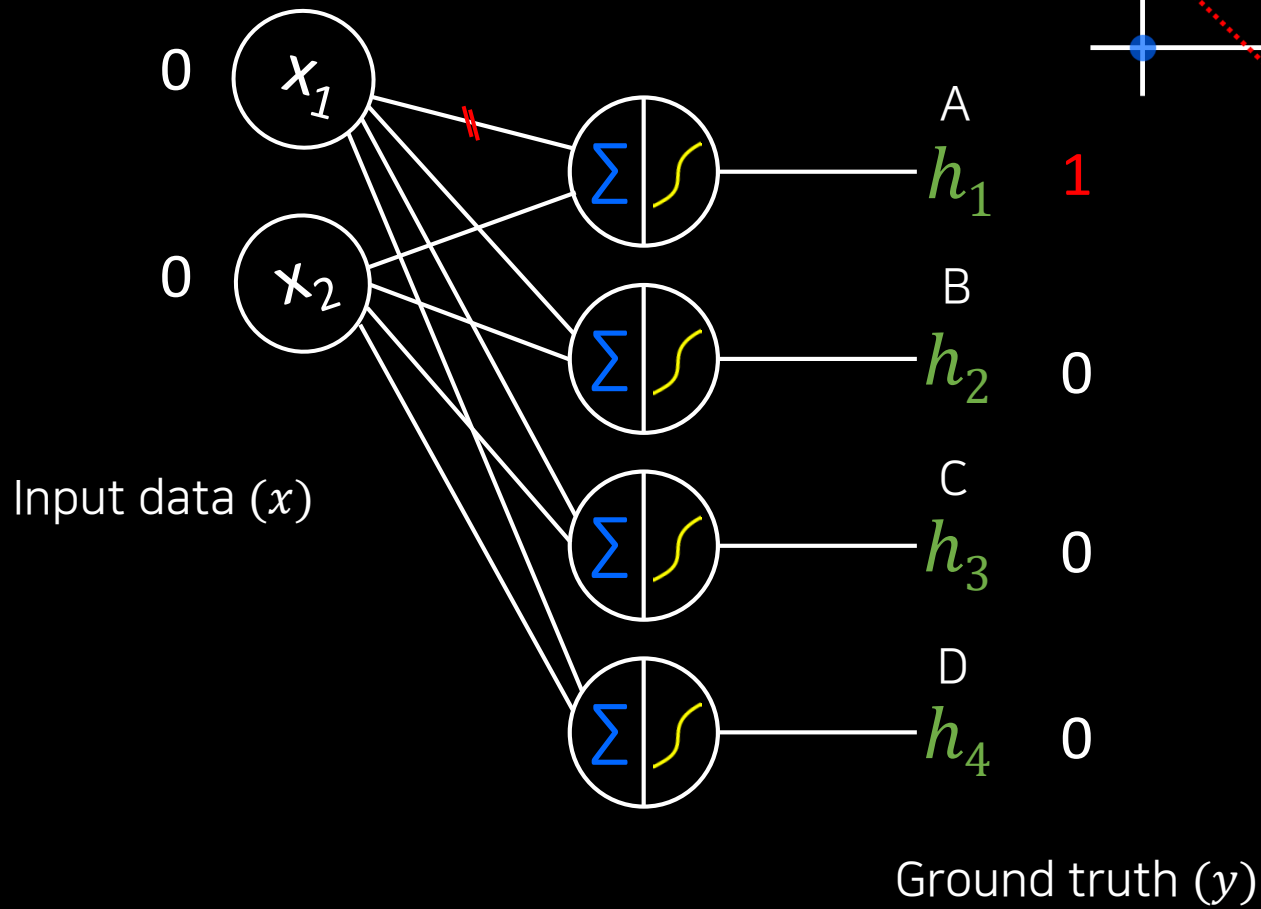
$$(x_1, x_2) \begin{pmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \end{pmatrix} \rightarrow (h_1, h_2, h_3, h_4)$$



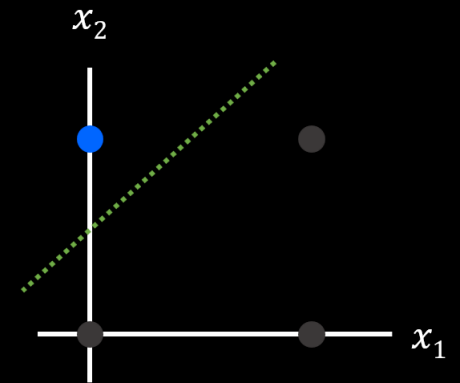
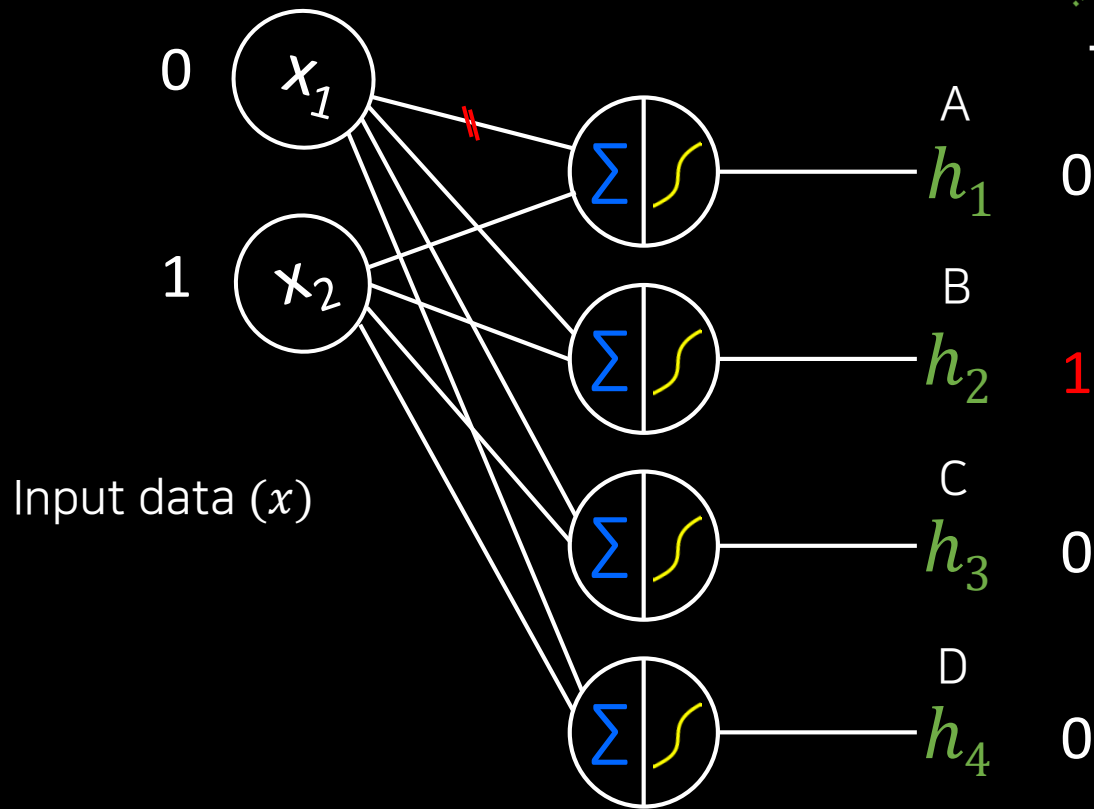
4 Neurons



4 Neurons

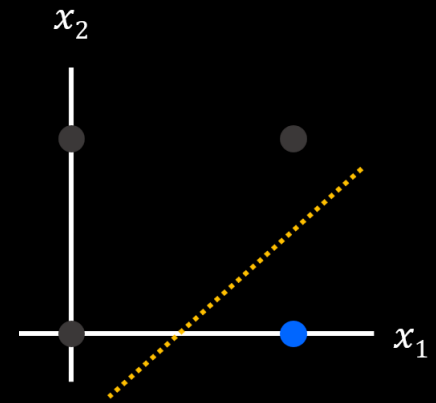
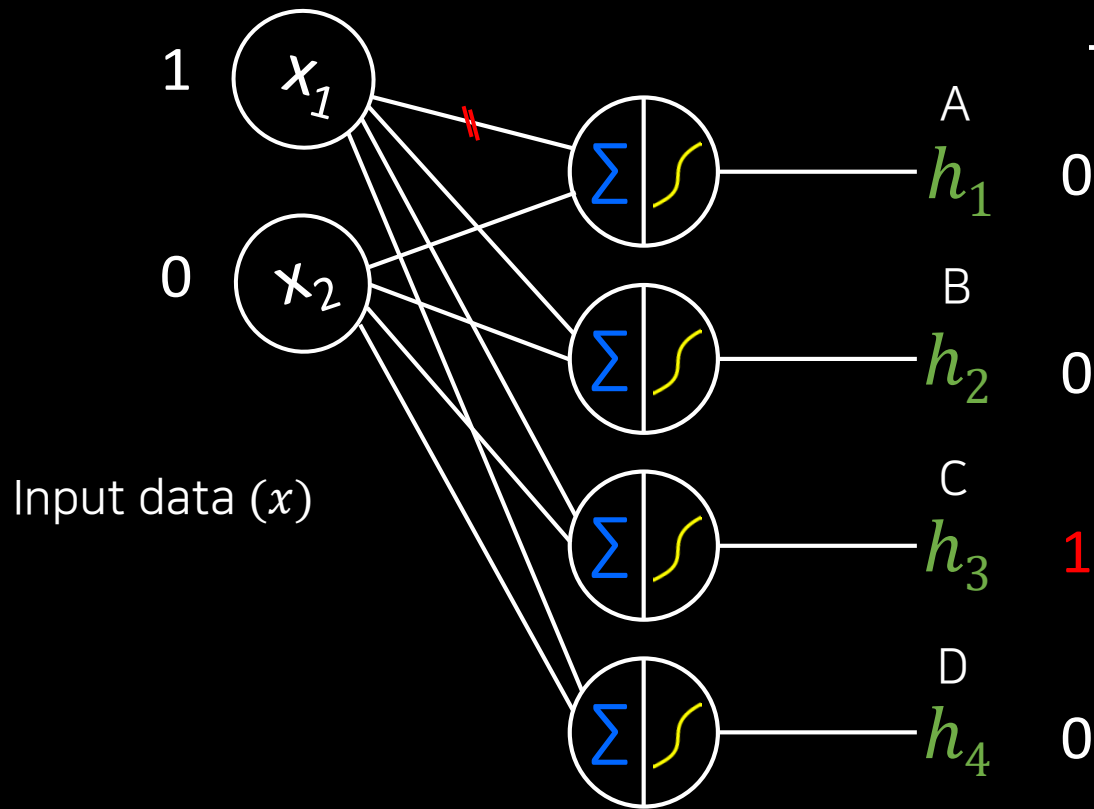


4 Neurons

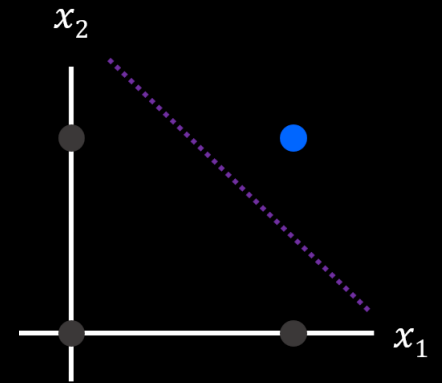
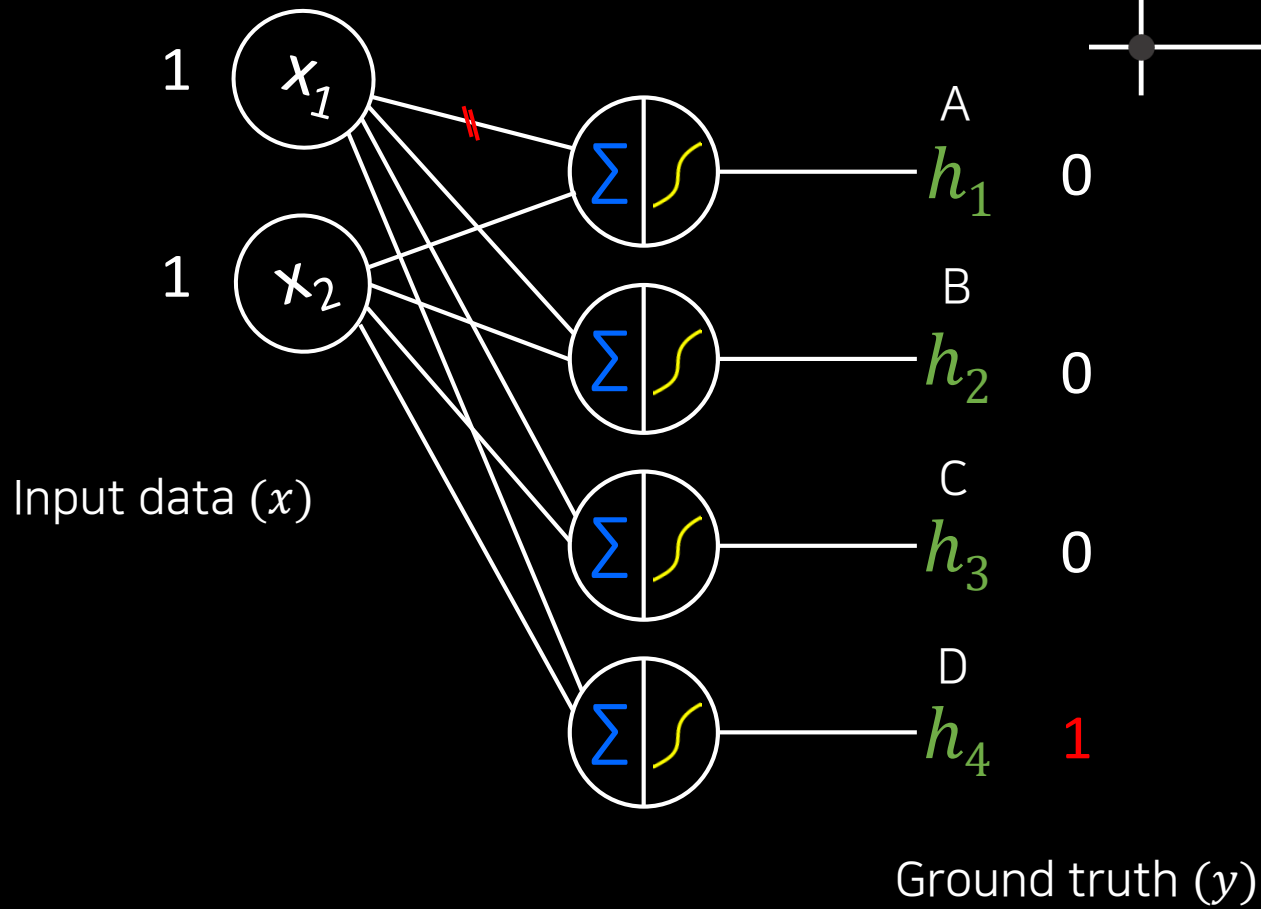


Ground truth (y)

4 Neurons



4 Neurons



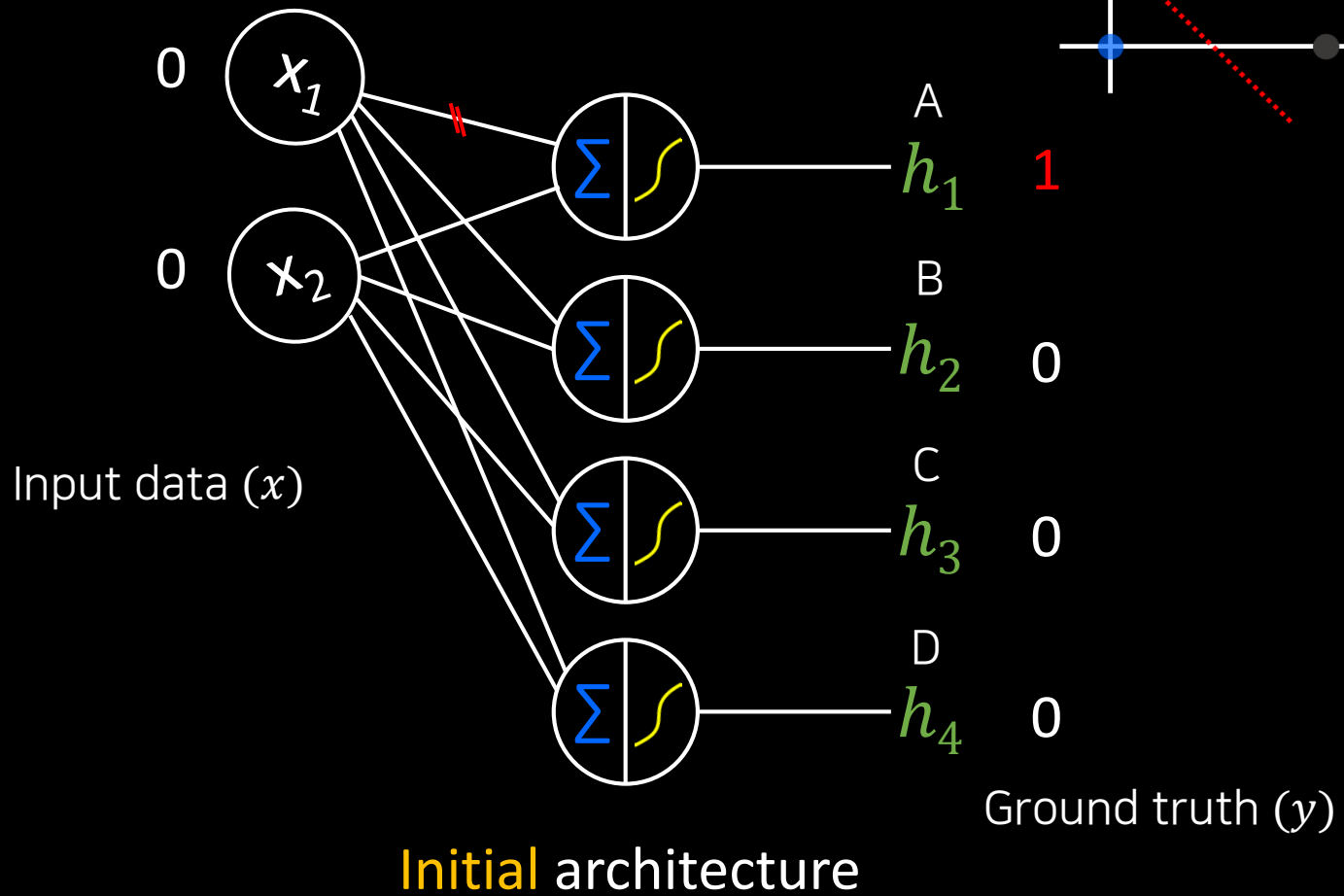
One-hot Encoding

- For the ground truth (y),
- setting only one as ON(1) and others as OFF(0)

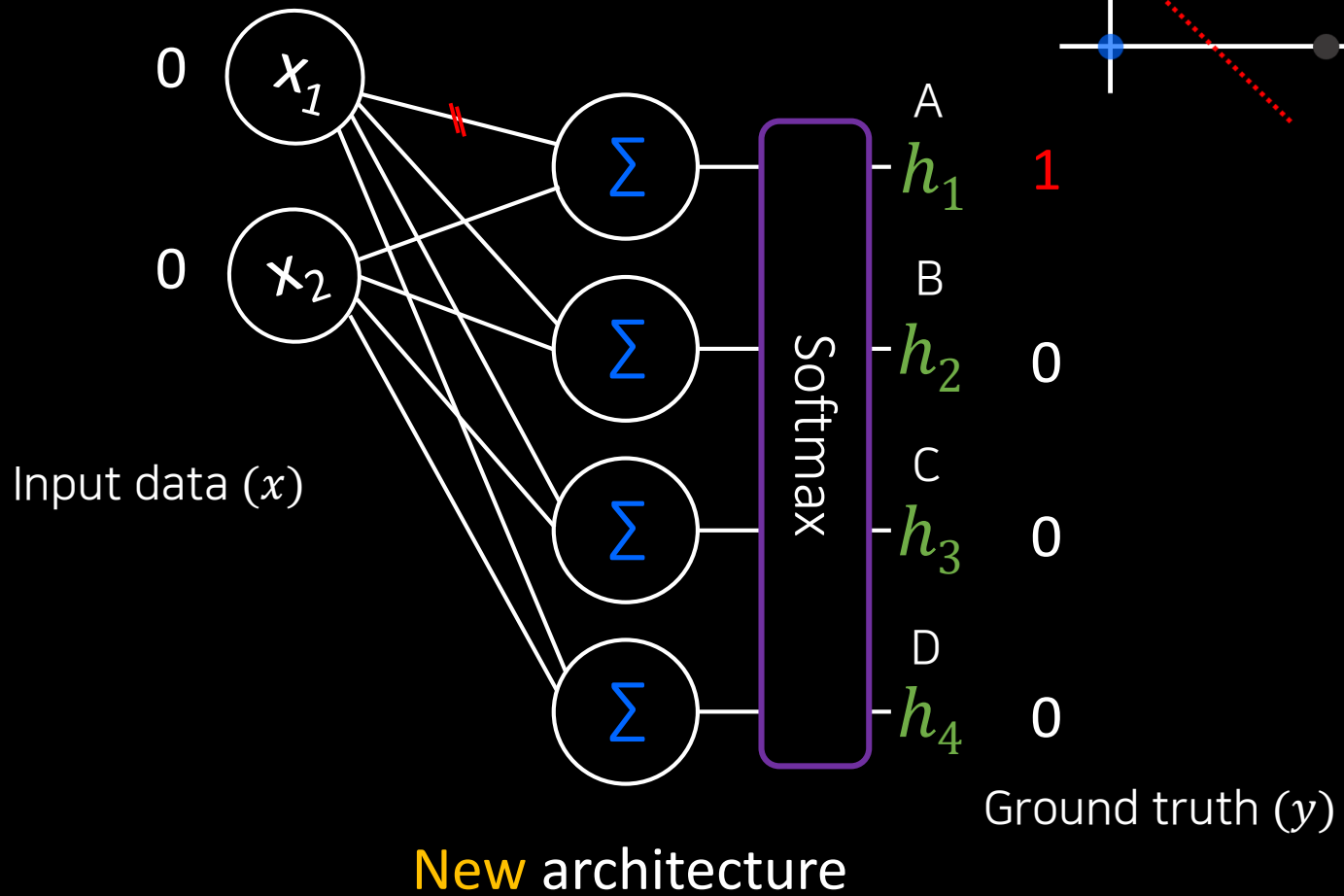
Considerations

- For the answers of four neurons, if a neuron's output is 1, and others must be 0.
- However, each neuron produce output independently.
- No way to control the 4 outputs together
- A **special function** introduced →
Softmax

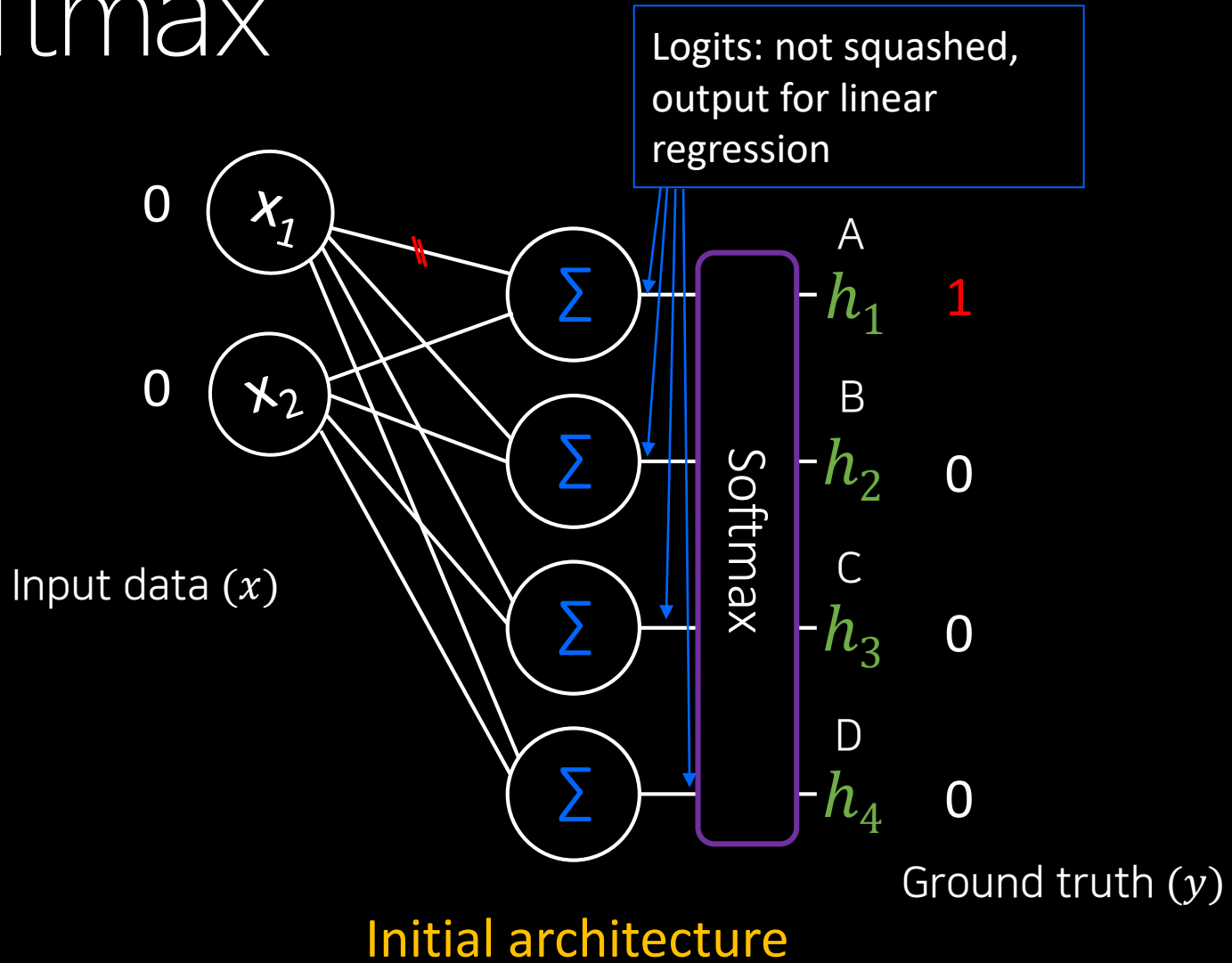
Softmax



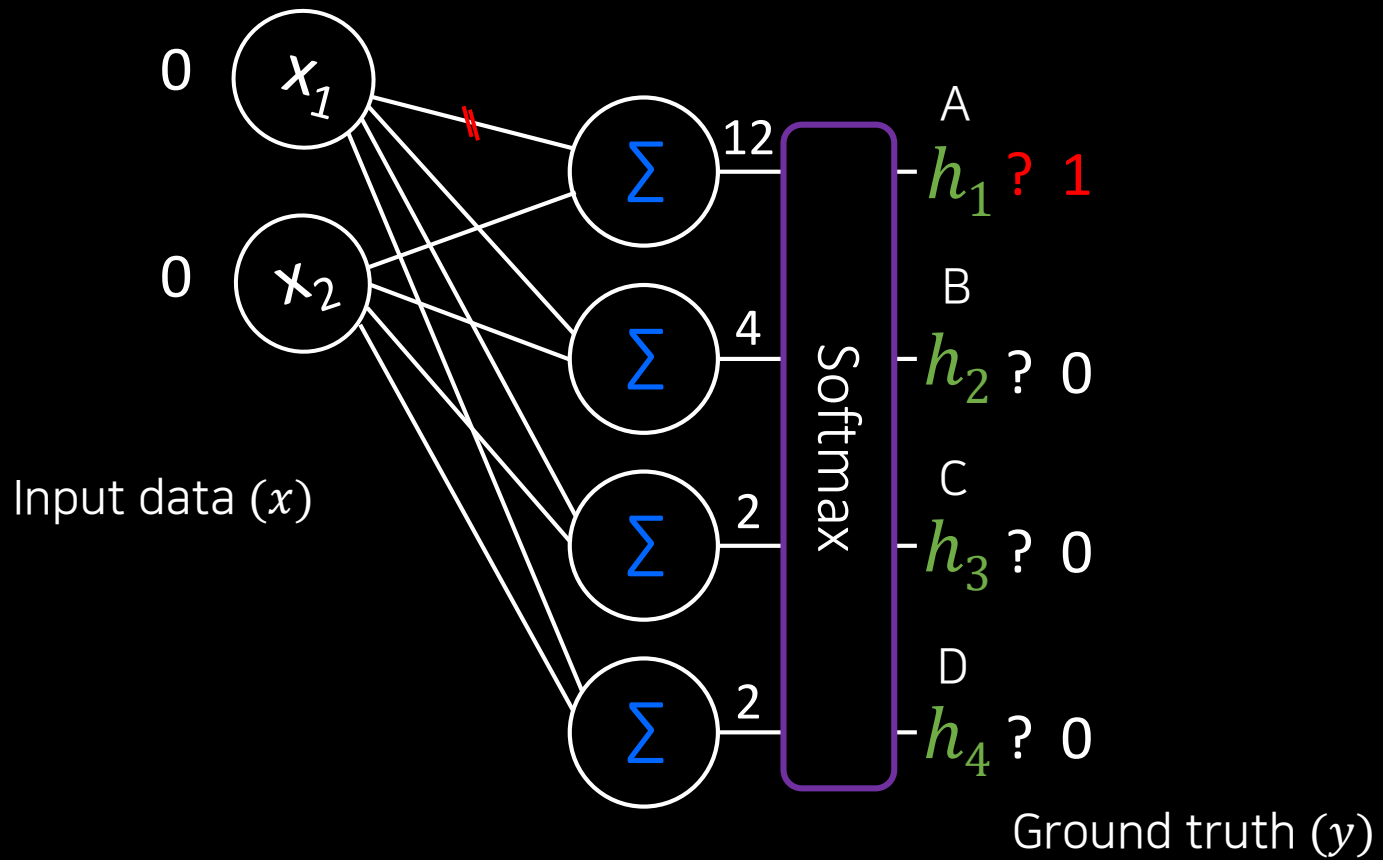
Softmax



Softmax



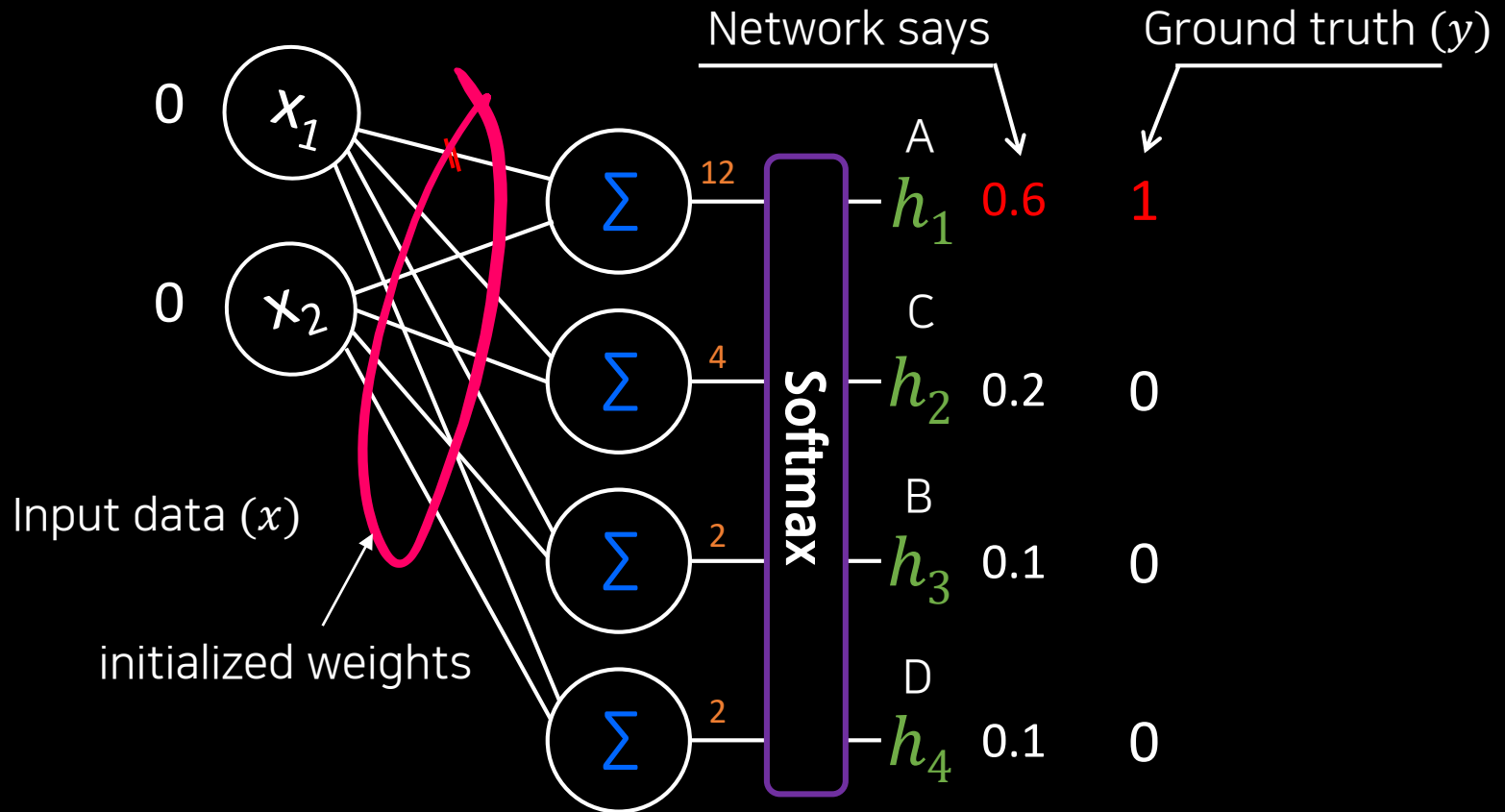
Softmax



Softmax

- For example, if the logits are 12, 4, 2, 2, then the Softmax function returns $\frac{12}{20}$, $\frac{4}{20}$, $\frac{2}{20}$, $\frac{2}{20}$ as results.
- Normalization of logits values
- Each value means the probability to be in the class.

Softmax

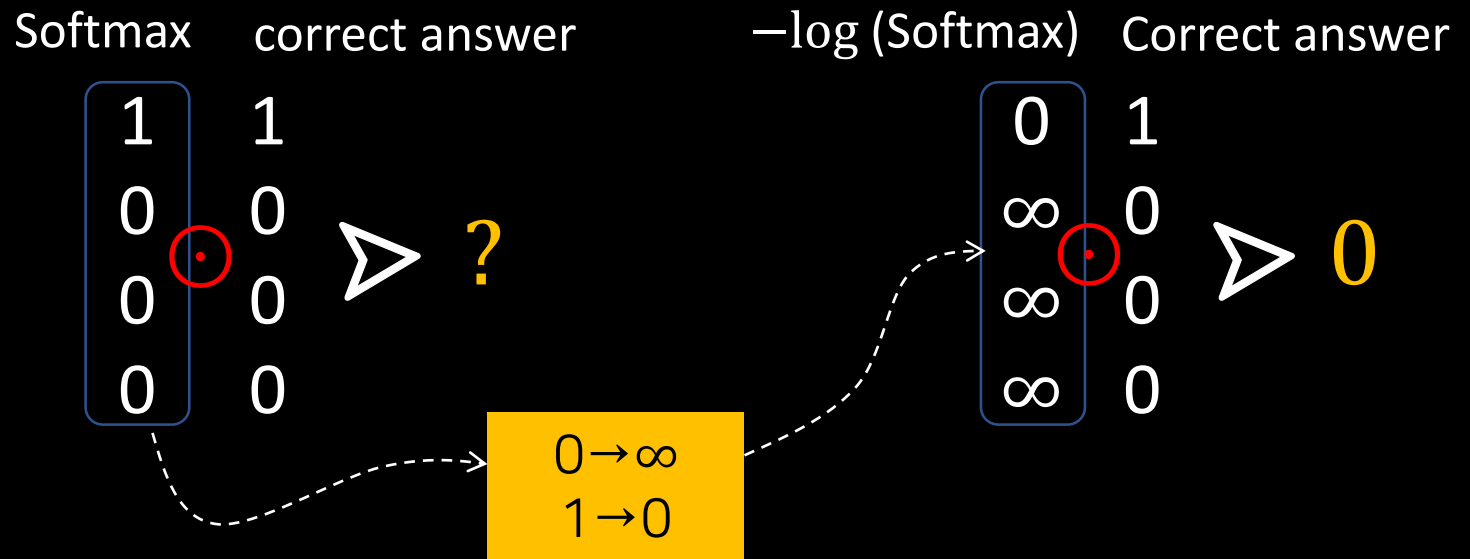


New Error Function

- Distance between the output of a network(softmax) and correct answer (ground truth)
- If answer correctly, then the distance is 0,
- If not(incorrect), then the distance is ∞

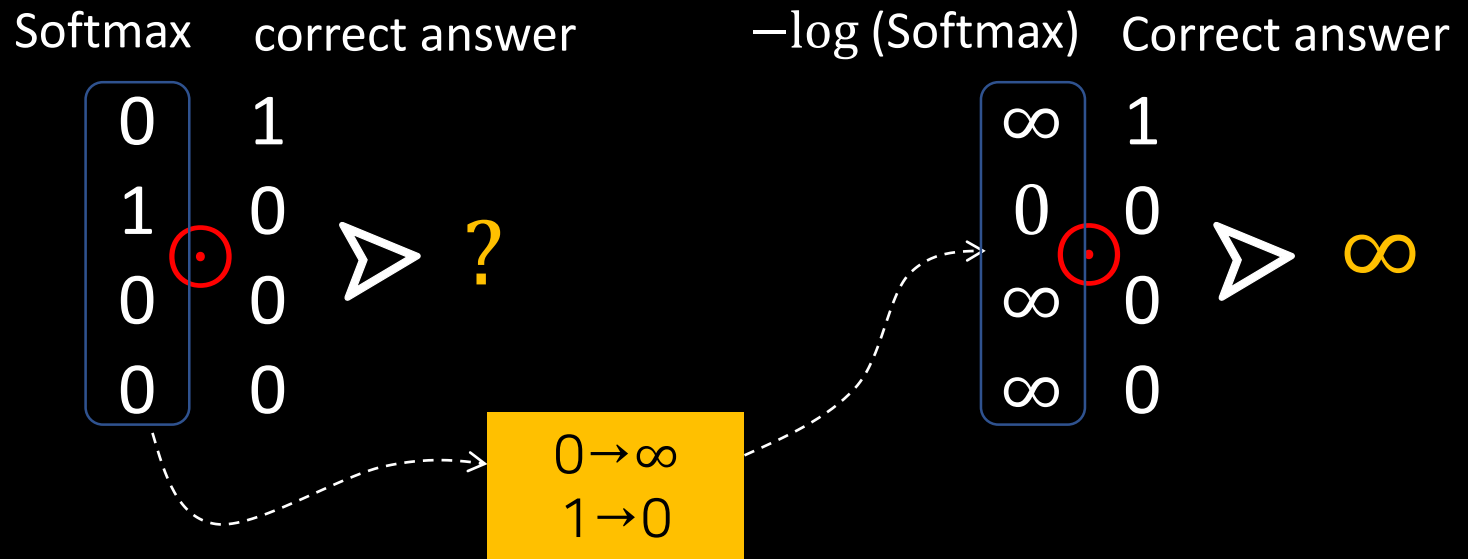
New Error Function

If answer correctly, then the distance(error) is 0.



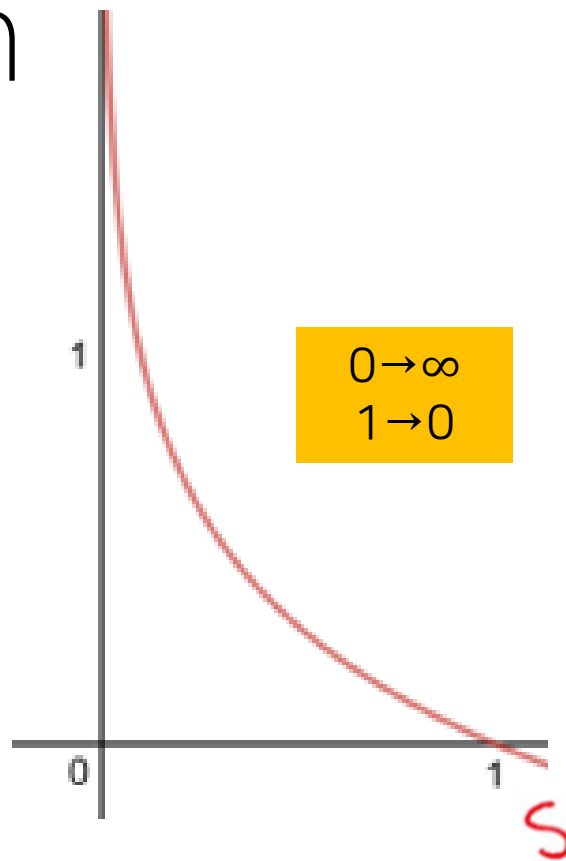
New Error Function

If incorrect, then the distance(error) is ∞ .



$-\log$ function

$$-\log(s)$$



The output of softmax

$$-L \log(S)$$

correct answer L

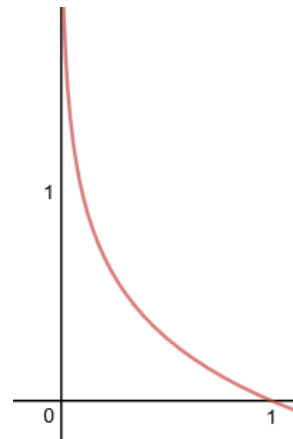
$$-\sum_i L_i \log(S_i)$$

New Error Function

$$D(\underset{\nearrow}{S}, \underset{\nwarrow}{L}) = - \sum_i L_i \log(S_i)$$

0.7
0.2
0.1
 $S(y)$

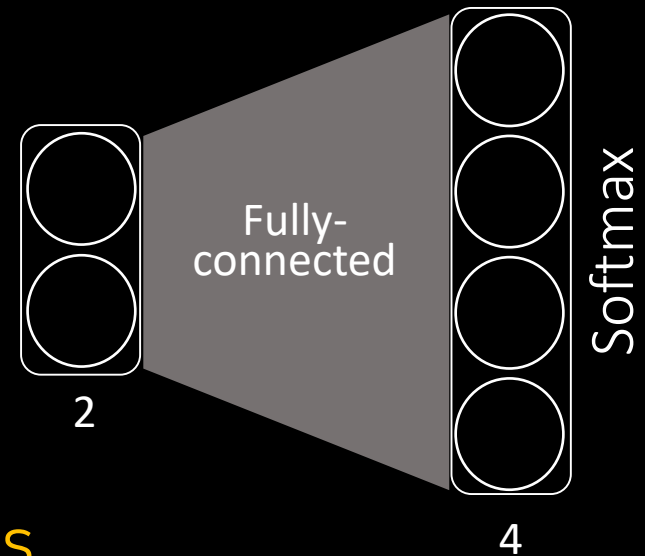
1.0
0.0
0.0
 L



`softmax_cross_entropy_with_logits(logits, y_data)`

- The function returns 0 if the answer is correct
- or returns ∞

Lab 14.py



- Classification into one of **four classes**
- 4 neurons where each has 2-input
- A bias for each neuron