AI and Deep Learning

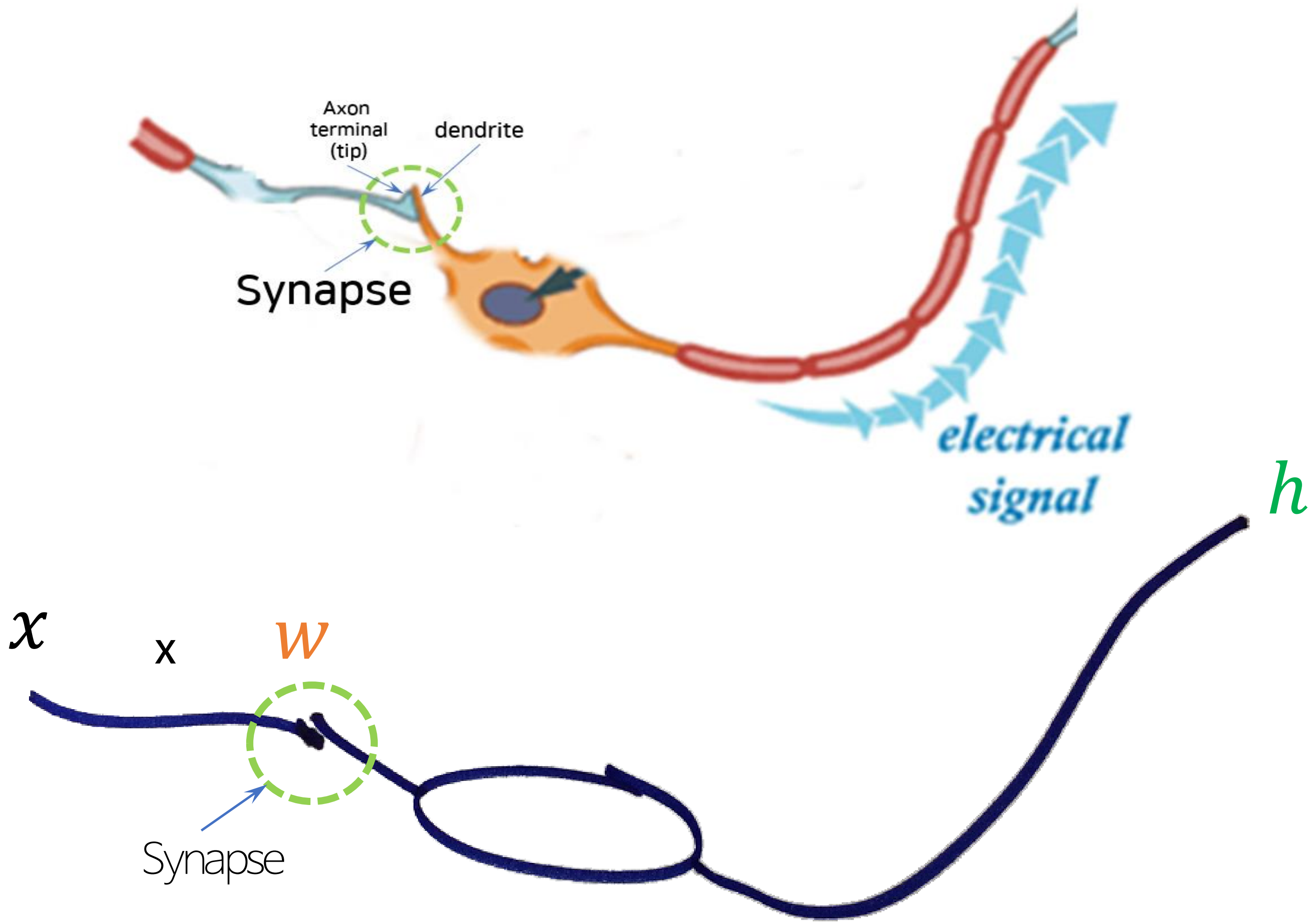
Logistic Regression & Classification

Jeju National University

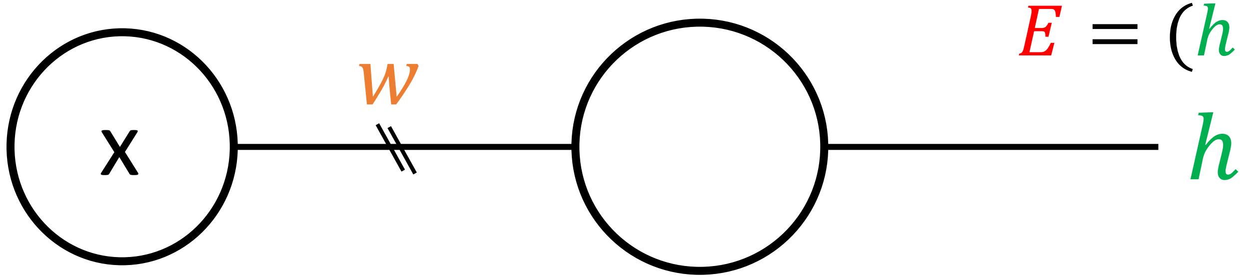
Yungcheol Byun

Agenda

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


Axon terminal (tip)
dendrite
Synapse
electrical signal
h
x
x
w
Synapse

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


x
w
h

Logistic Regression

The shape of regression is not linear but logistic.

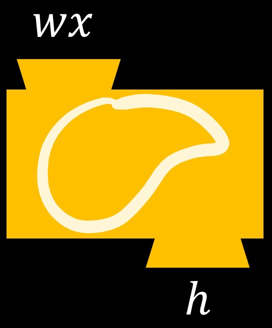
What does that mean?

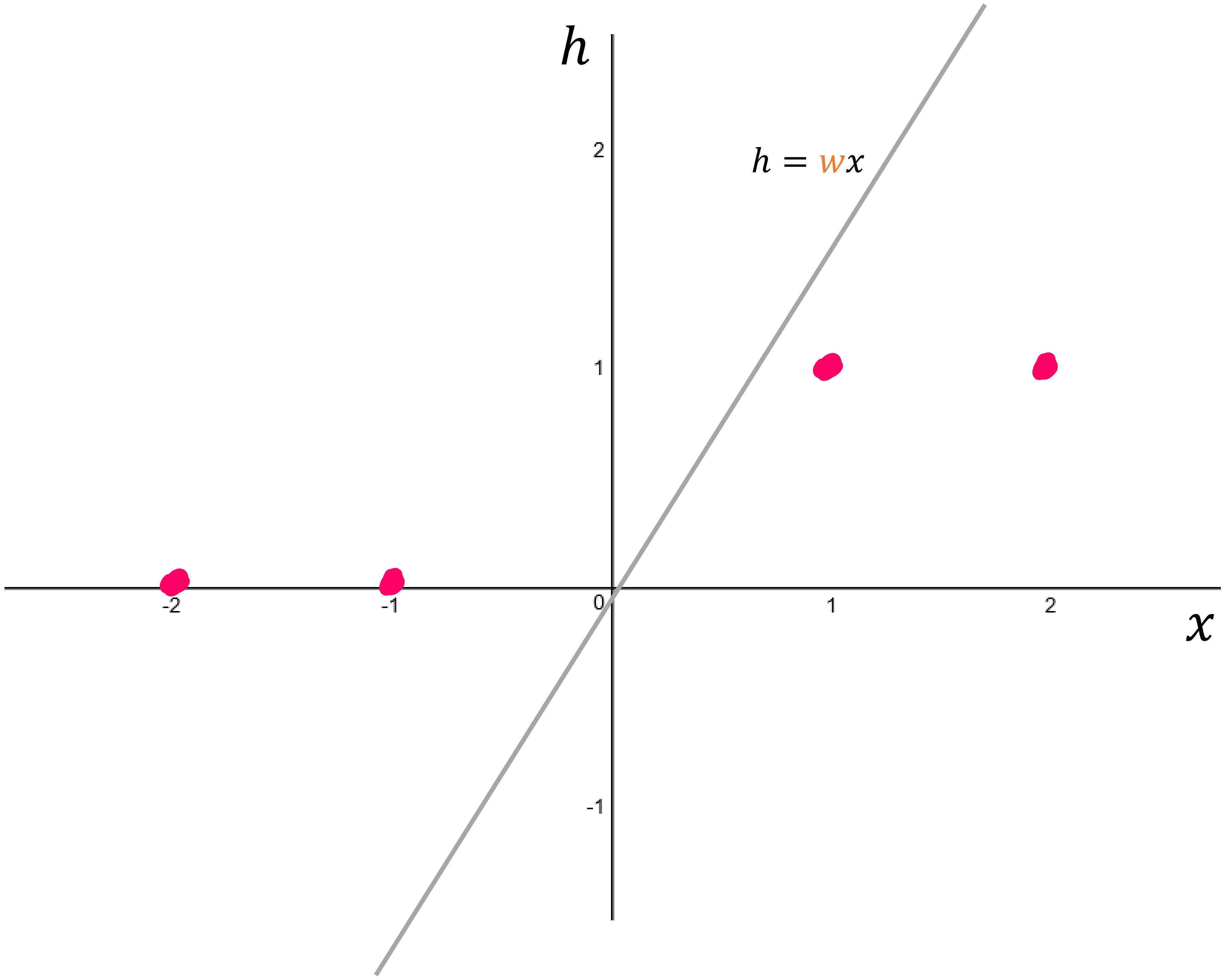
desmos

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

$$h = wx$$

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





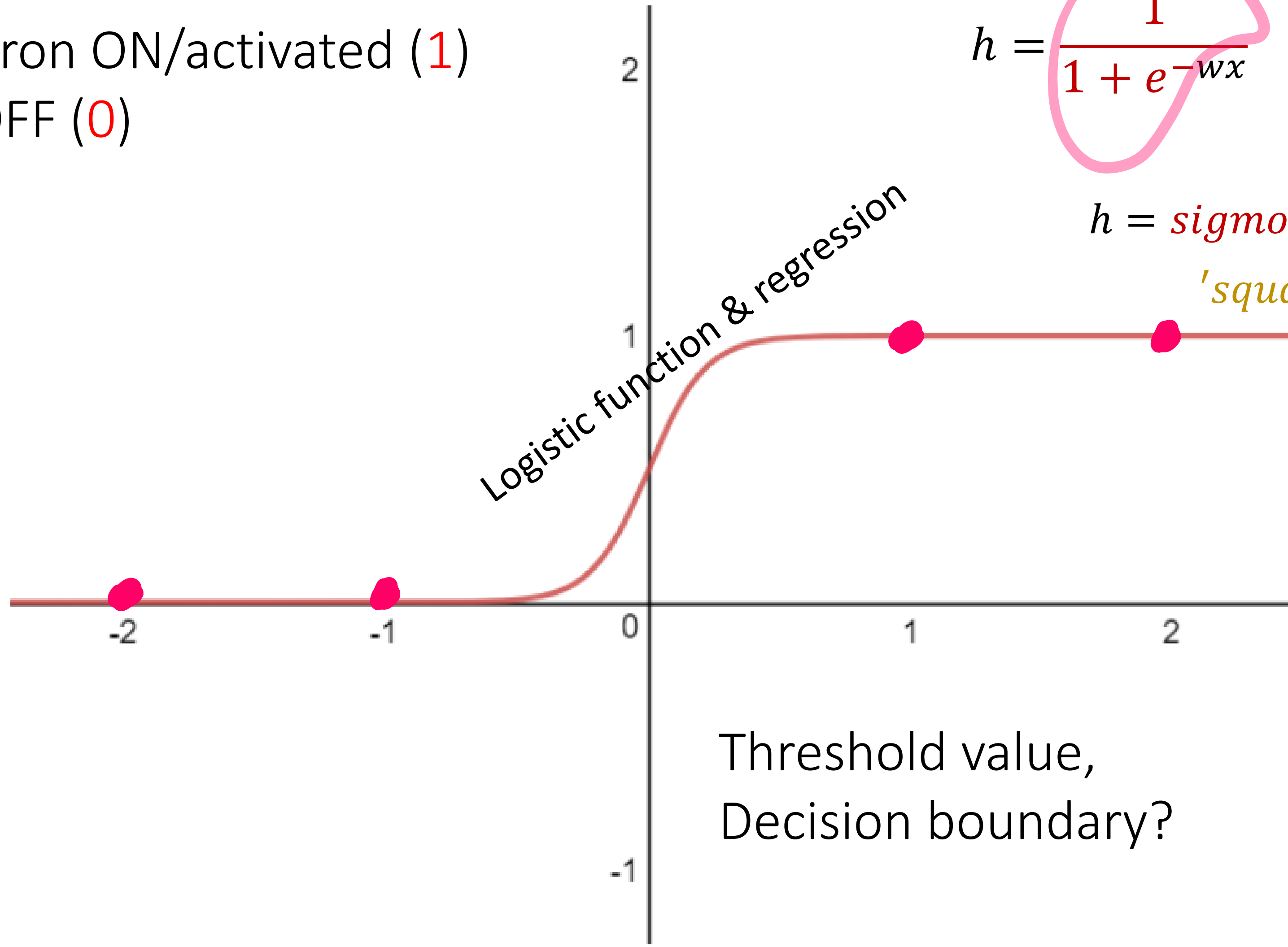
h
$h = wx$
2
1
0
-1
-2
-1
1
2
x

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

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

$$h = sigmoid(wx)$$

$'squash'$



Threshold value,
Decision boundary?

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

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

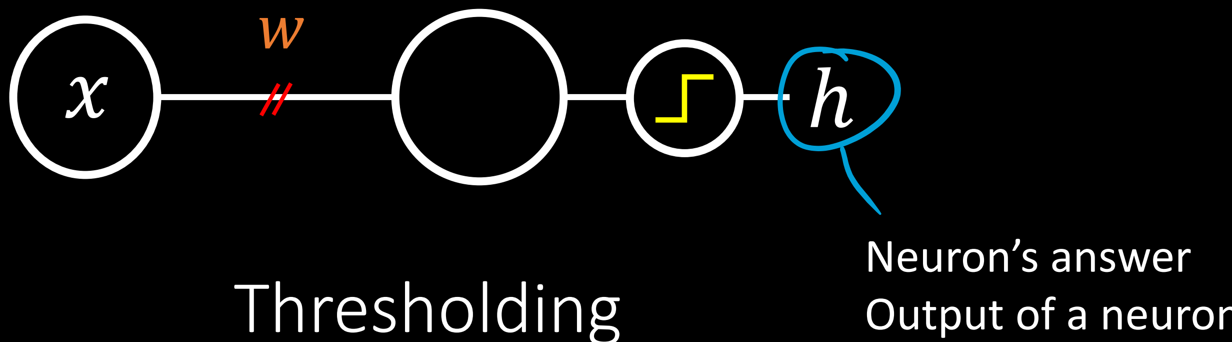
- e = the natural logarithm base (also known as Euler's number),
- x_0 = the x-value of the sigmoid's midpoint,
- L = the curve's maximum value, and
- k = the logistic growth rate or steepness of the curve.[1]

Revisited

Real operation of a neuron

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

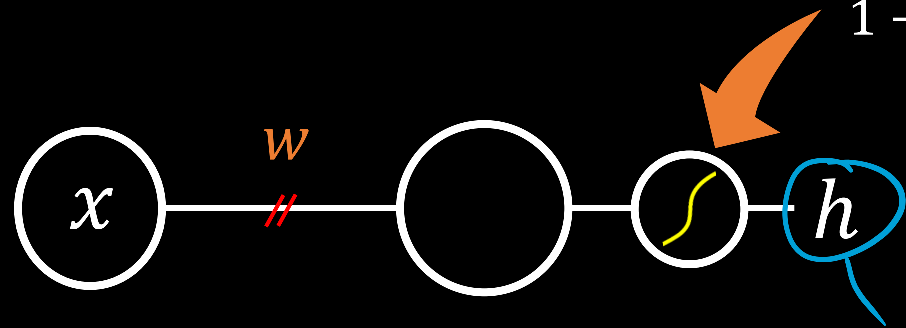
Revisited



Thresholding

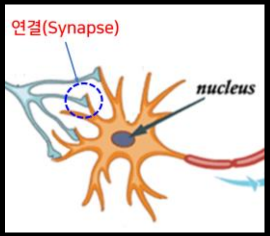
Revisited

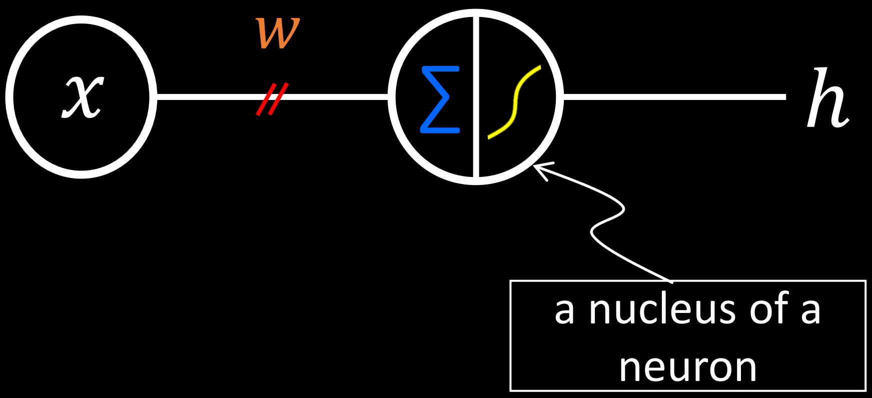
sigmoid $$\frac{1}{1+e^{-wx}}$$



Neuron's answer
Output of a neuron

Thresholding



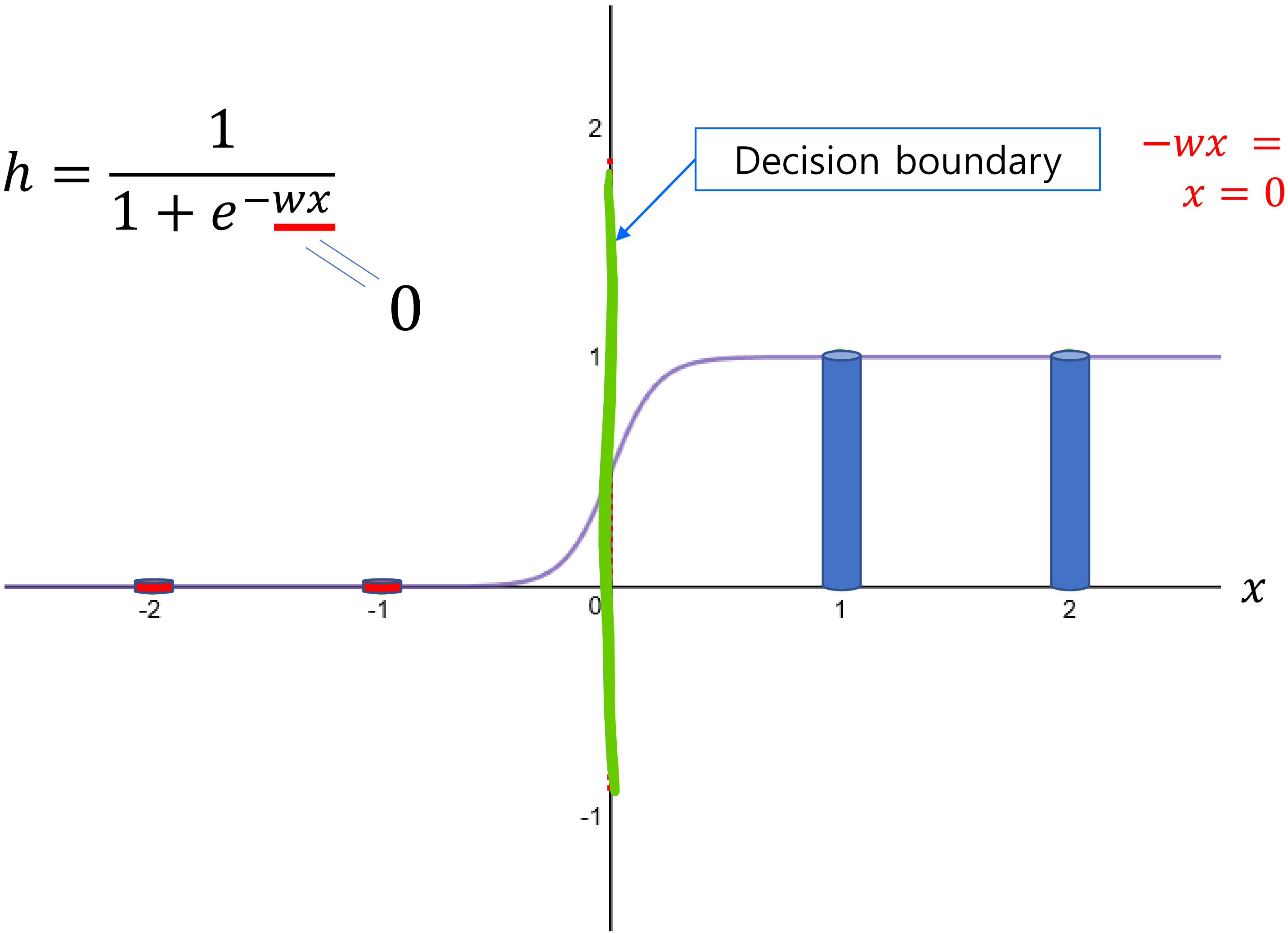


- Activated(1) or not(0) according to the input x
- Find the decision boundary to decide 1 or 0.

Decision boundary

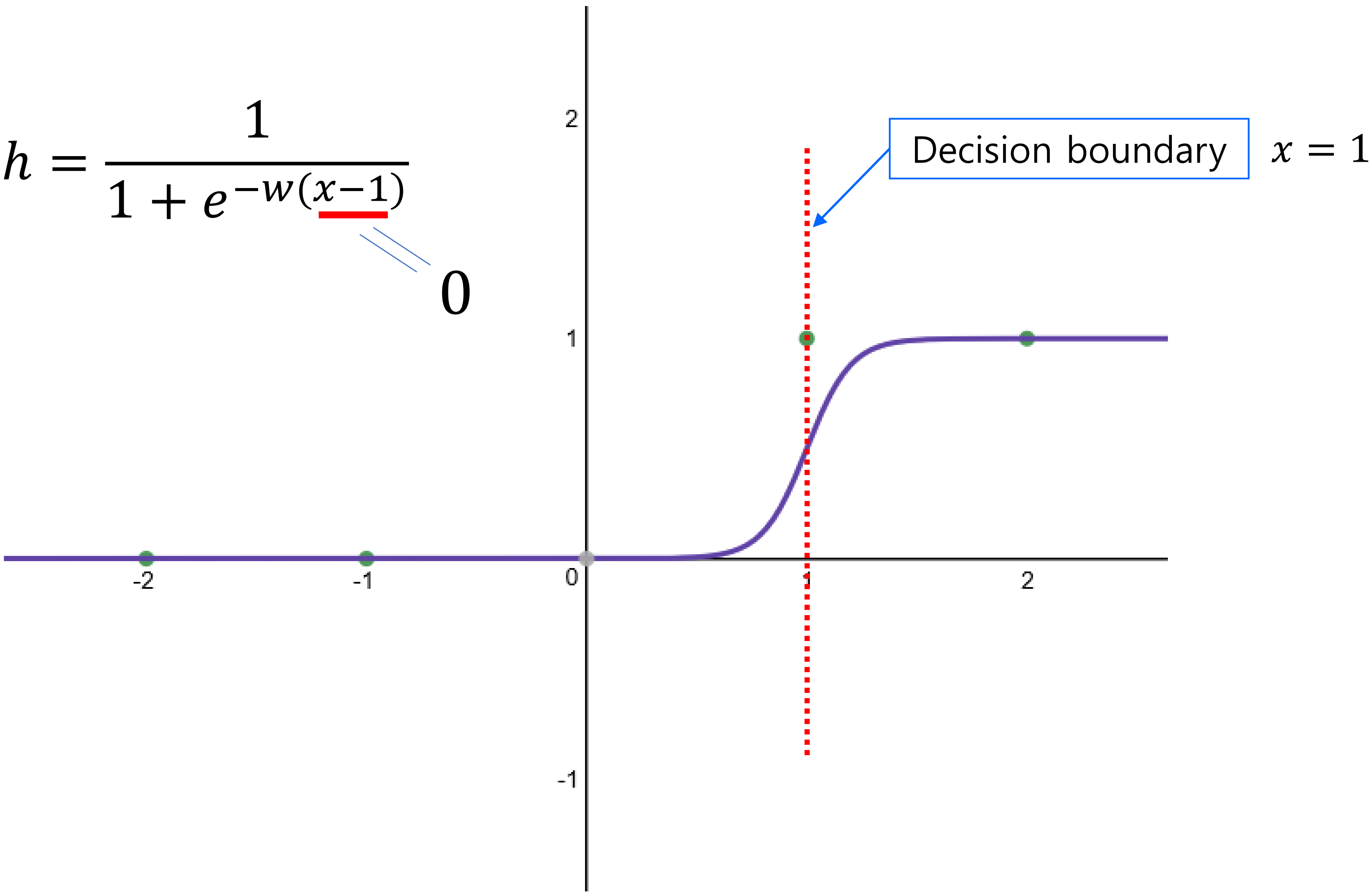
$$h = \frac{1}{1 + e^{-wx}}$$

$-wx = 0$

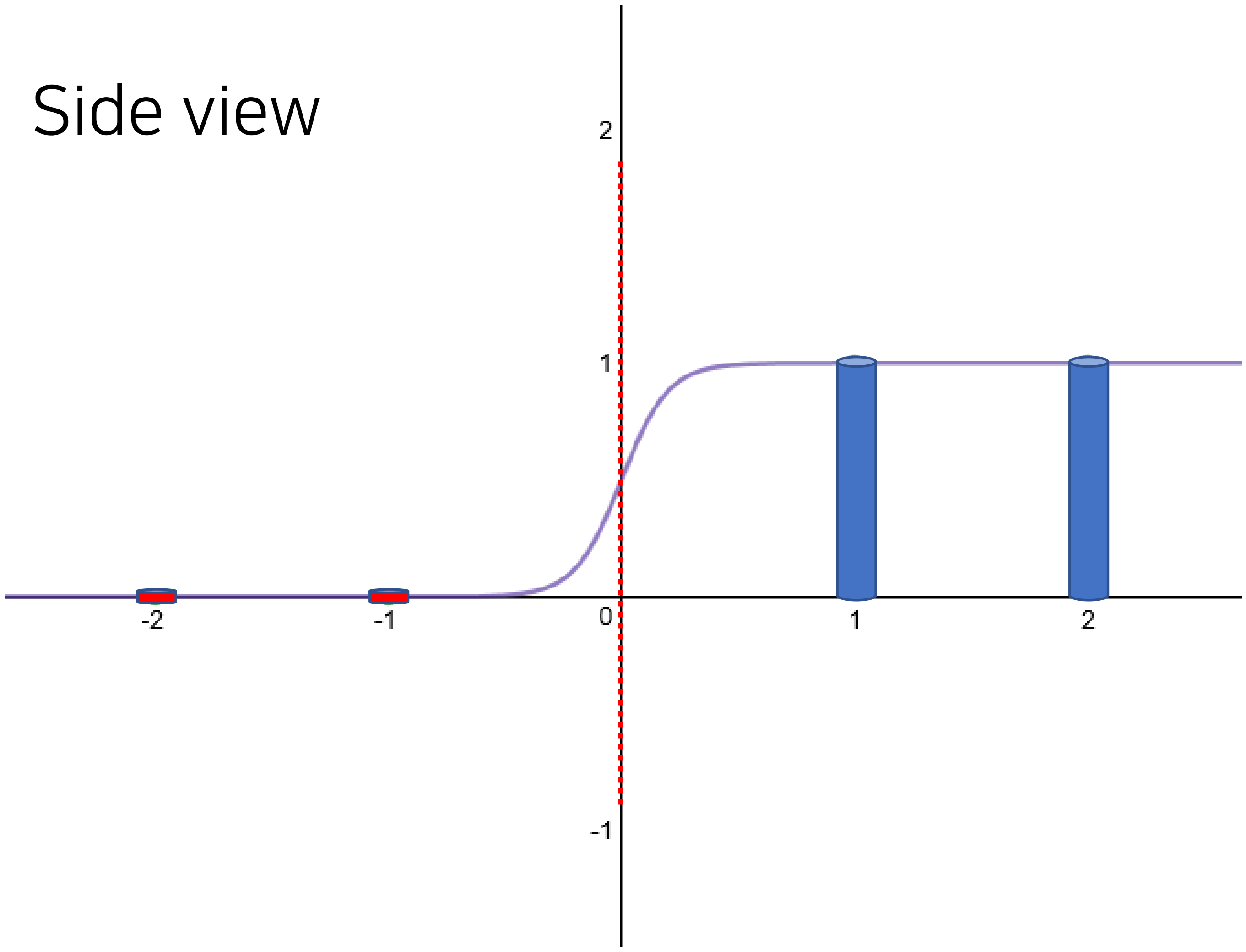


Decision boundary

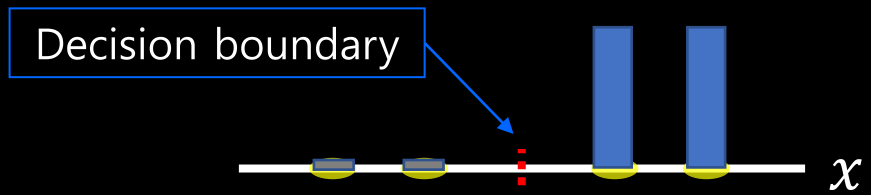
$$h = \frac{1}{1 + e^{-w(\underbrace{x-1}_{0})}}$$



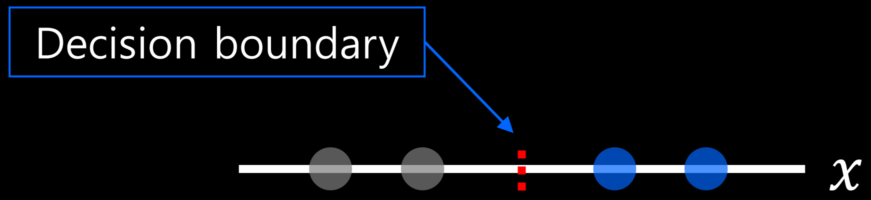
Side view



Decision Boundary



{Side view}

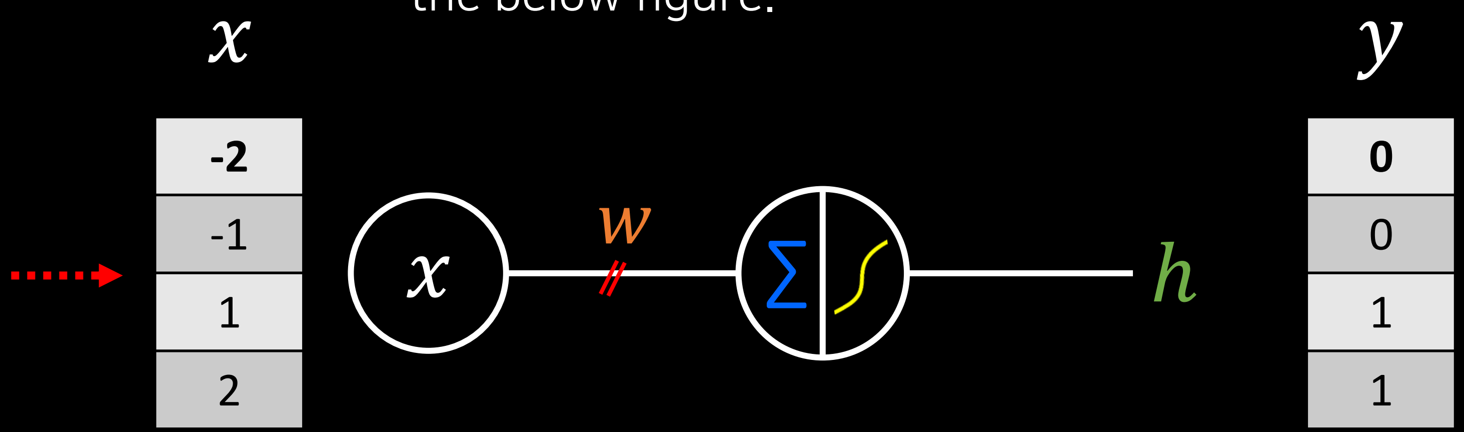


{View from above}

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

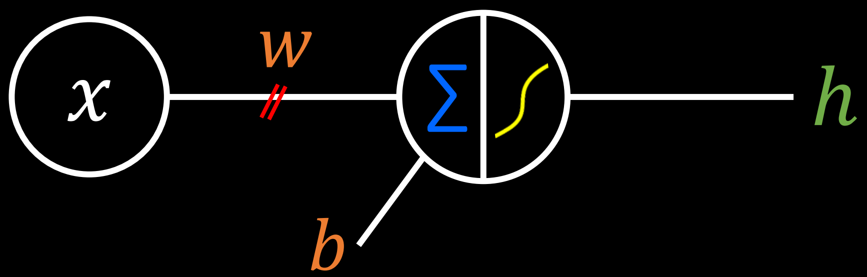
Guess the decision boundary from the below figure.



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

Guess the decision boundary from the below figure.

x	y
-2	**0**
-1	0
1	1
2	1



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

Hypothesis

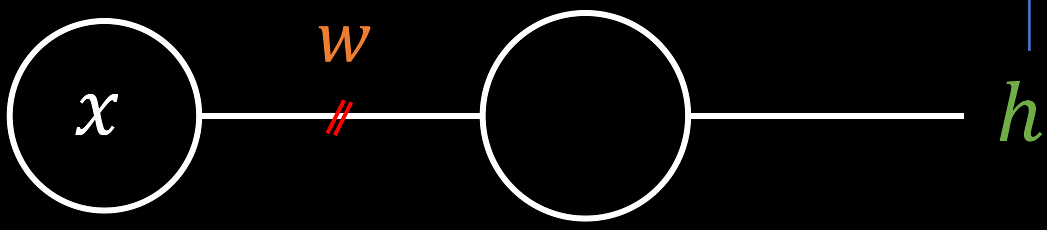
- What is hypothesis? The answer of a neuron
- Find decision boundary from the equation.

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

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

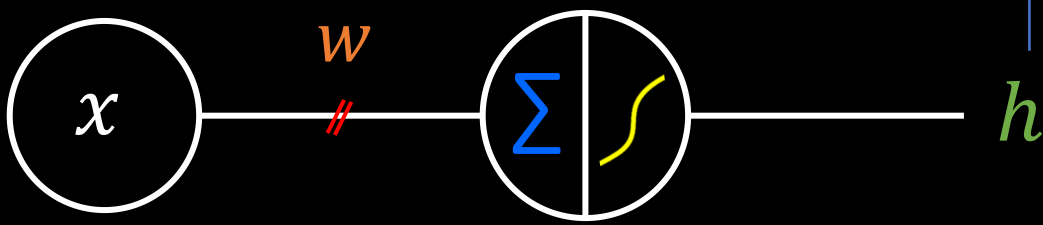
Cost/Error Function

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



Cost/Error Function

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



Does it work ?

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

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

if we have data (1, 1)

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

if we have data (1, 1)

desmos

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)

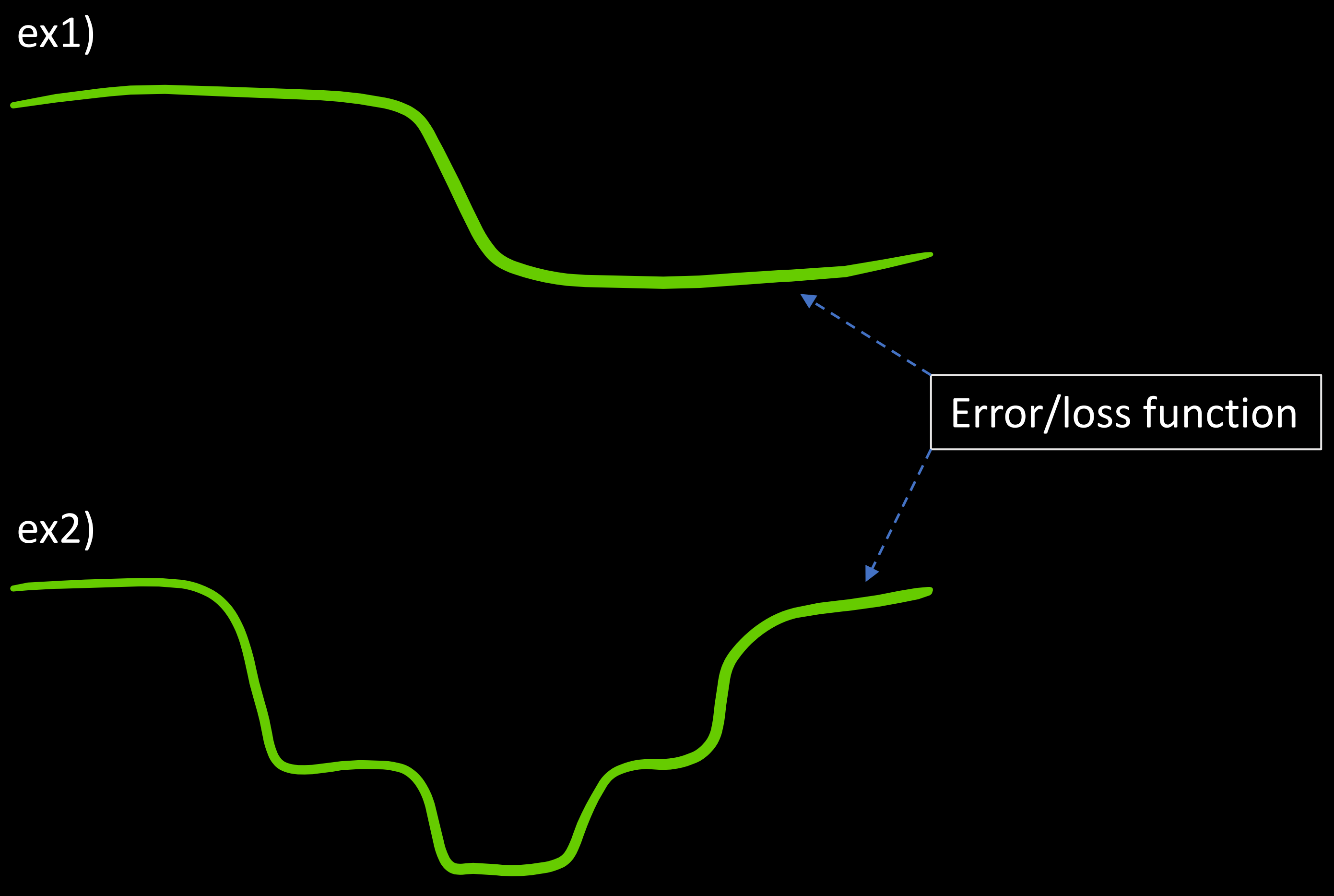
Draw 4 points: $(-1, 0), (1, 1), (-3, 0), (3, 1)$.

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

Add bias b.

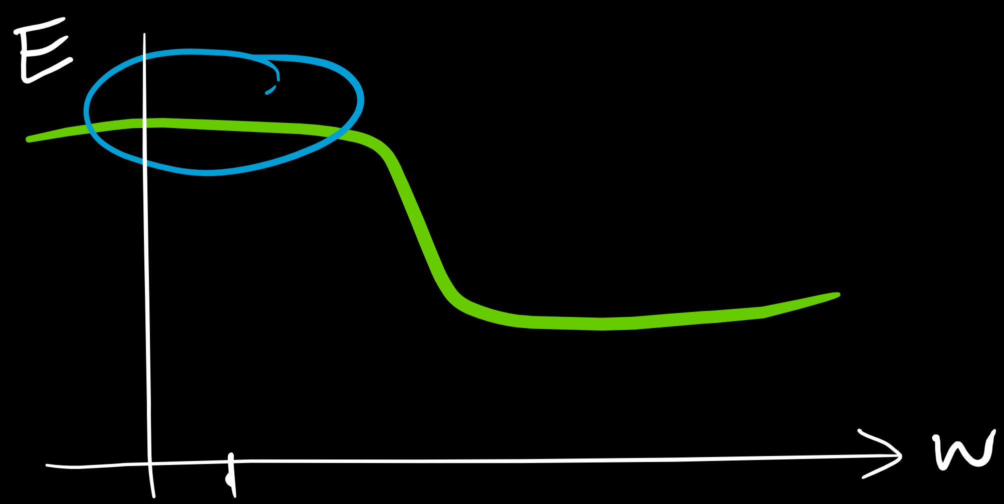
plot the error: (w, E)

Cost/Error Function



What problem in the error function?

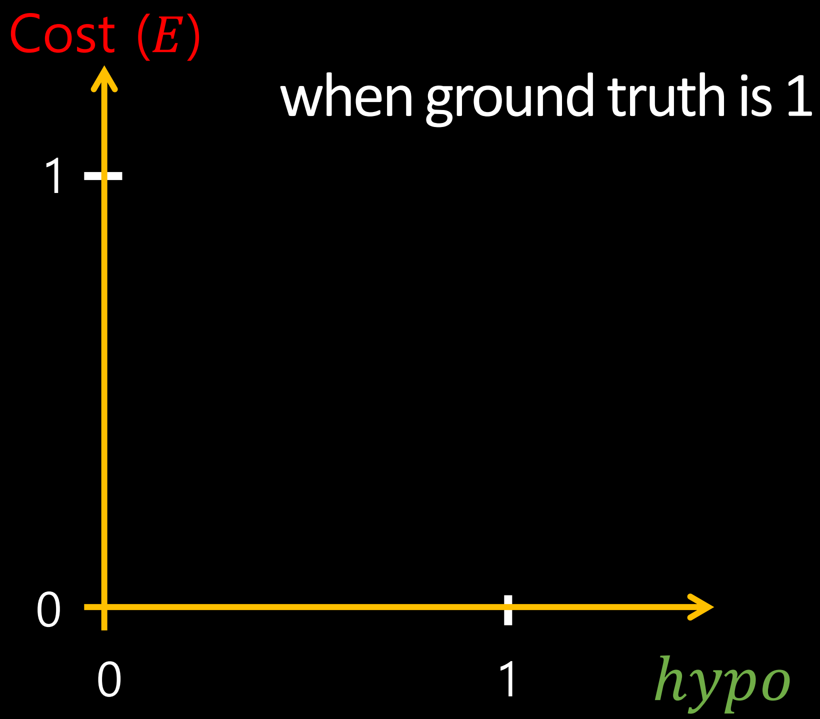
No gradient decent
in some parts



New Cost/Error Function

When ground truth is 1

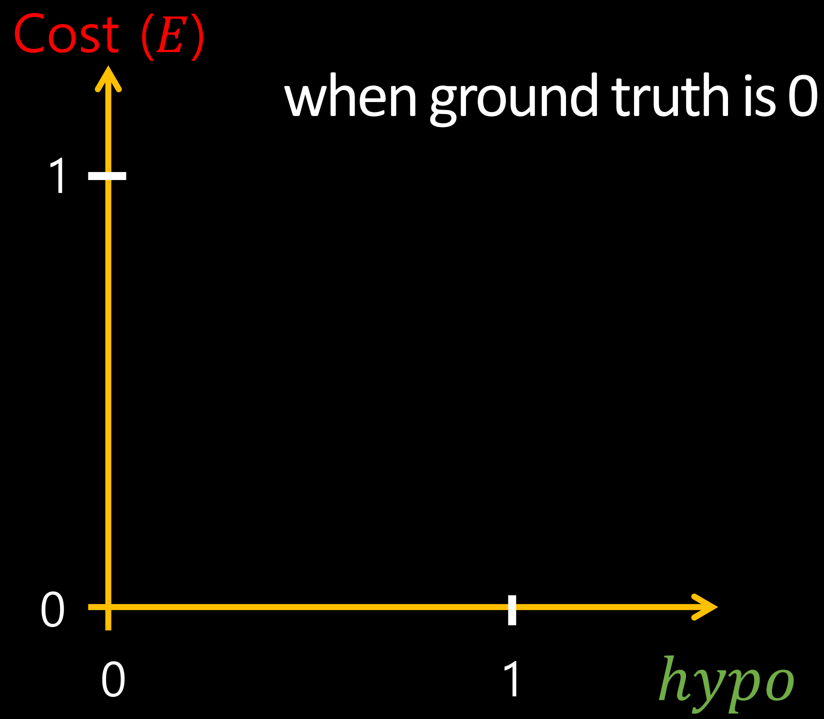
- if hypo is equal to 1, then error = 0
- if hypo is equal to 0 then error = ∞



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 = ∞



when ground truth is 1

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

when ground truth is 0

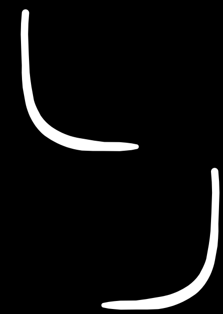
$$E = -\log(1-h)$$

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

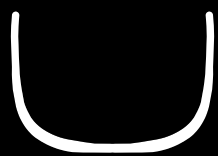
$$E = -\log\left(1-\frac{1}{1+e^{-wx}}\right)$$

New Cost/Error Function

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



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

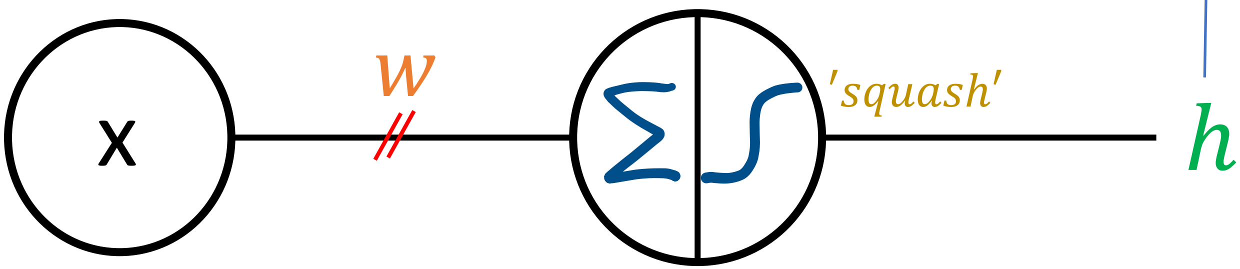


or, $$E = -(y\log(wx) + (1 - y)\log(1 - wx))$$

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

Binary Cross-Entropy Loss

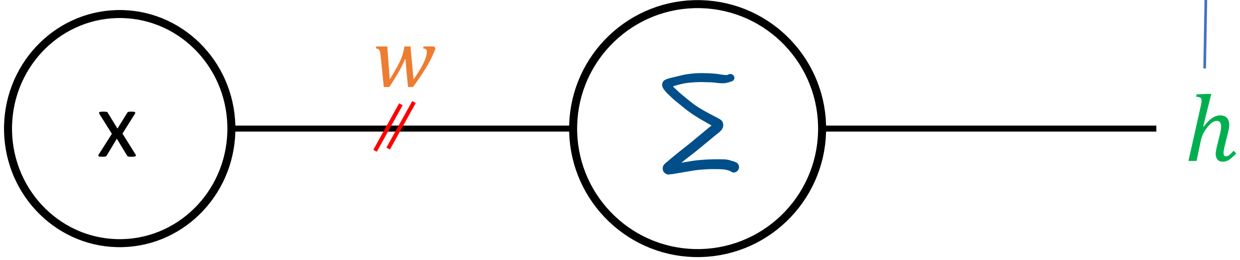
$$E = -(y\log(h) + (1-y)\log(1-h))$$



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

L2 Loss

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



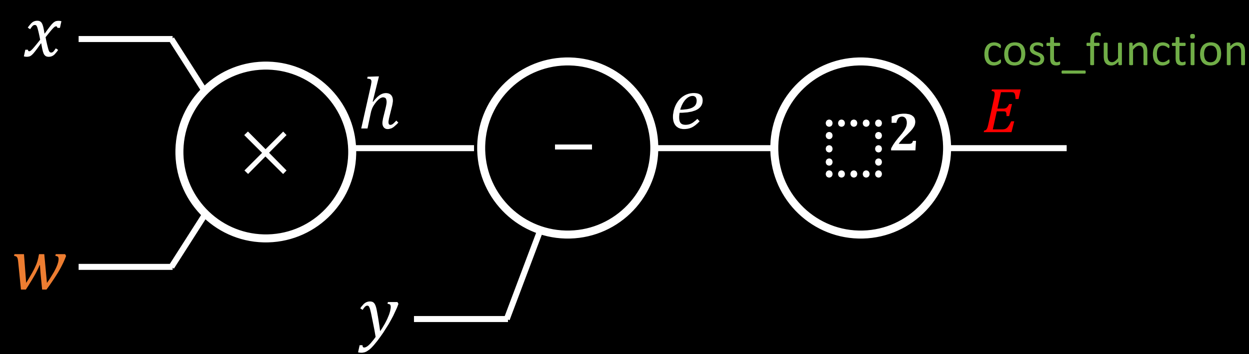
Computational graph

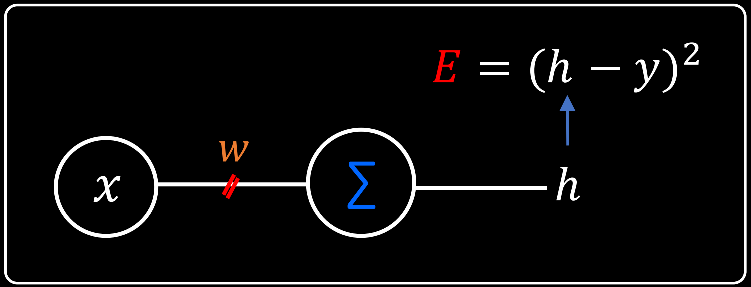
for the new cost function

Computational Graph

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

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



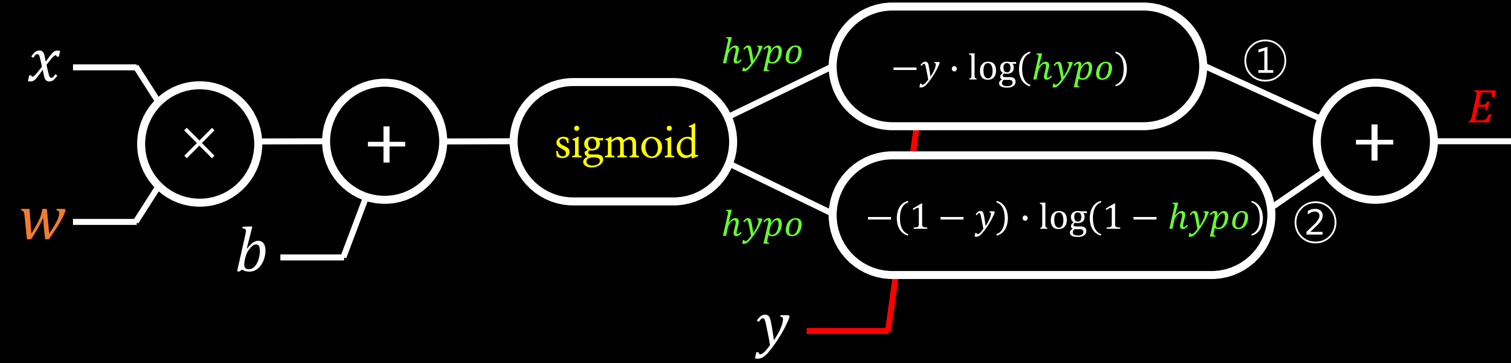


Computational Graph

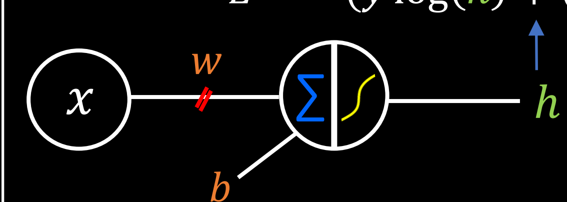
① ②

$$E = \boxed{-y \cdot \log(hypo)} \boxed{- (1-y) \cdot \log(1-hypo)}$$

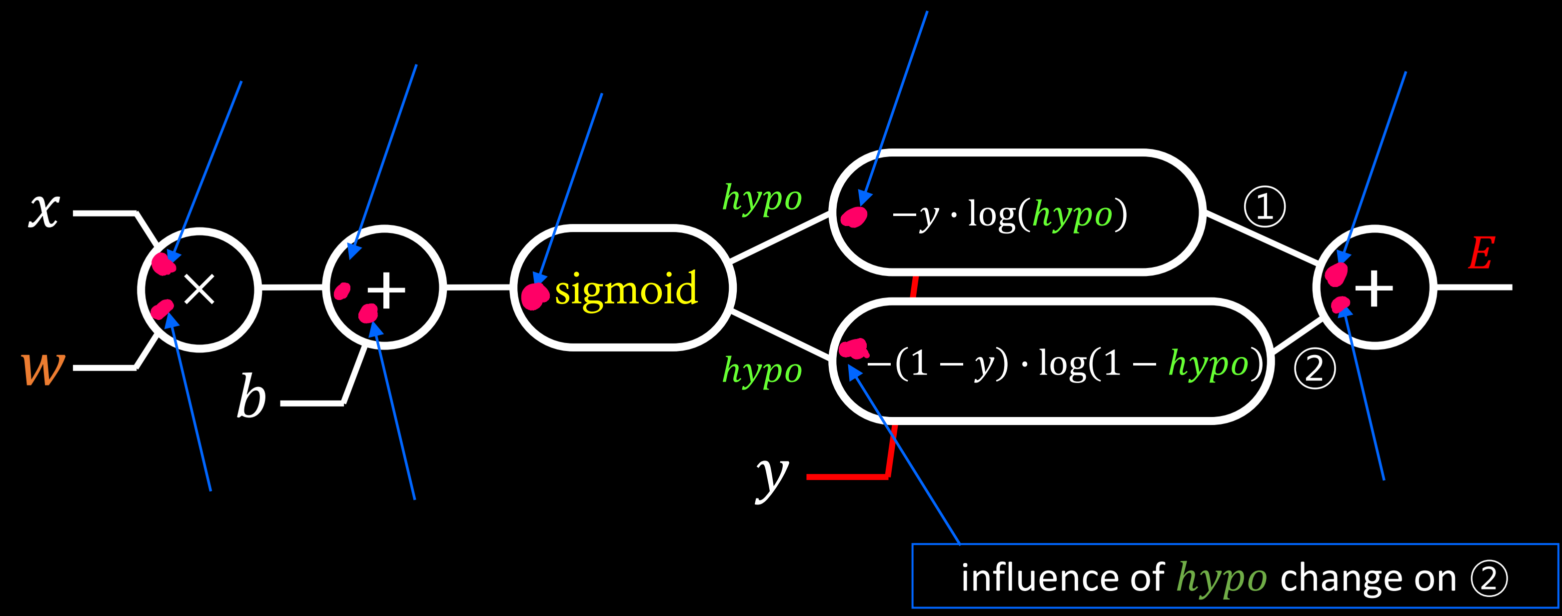
Binary Cross-Entropy Loss



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



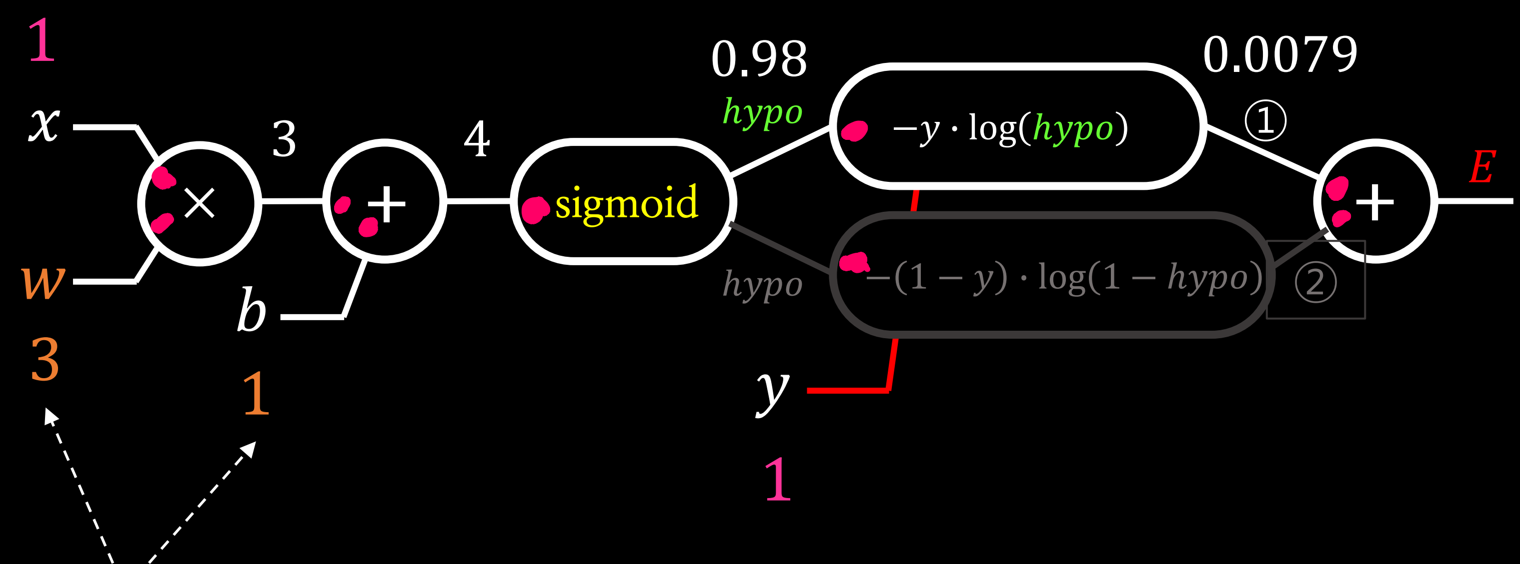
Local Gradients



$$\frac{\partial②}{\partial h}$$

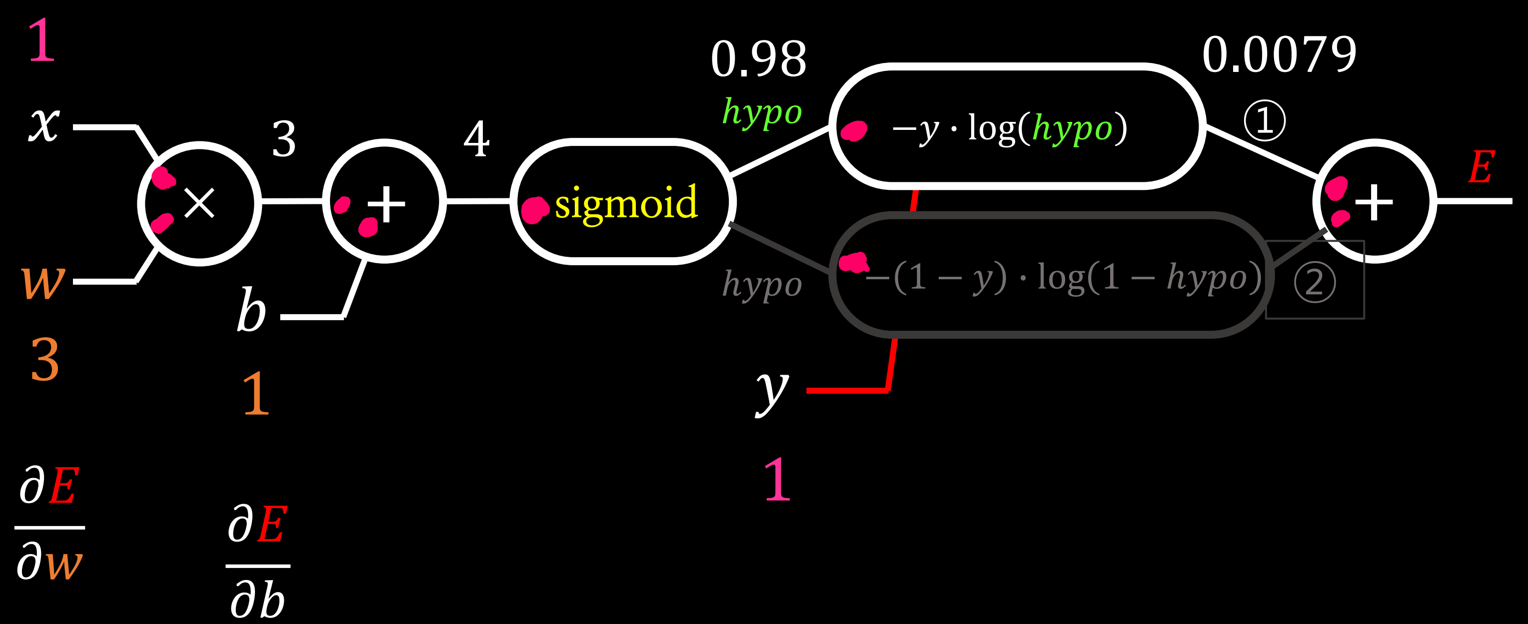
Forward propagation

$$(x, y) \longrightarrow (1, 1)$$



randomly initialized ⟶ will be optimized

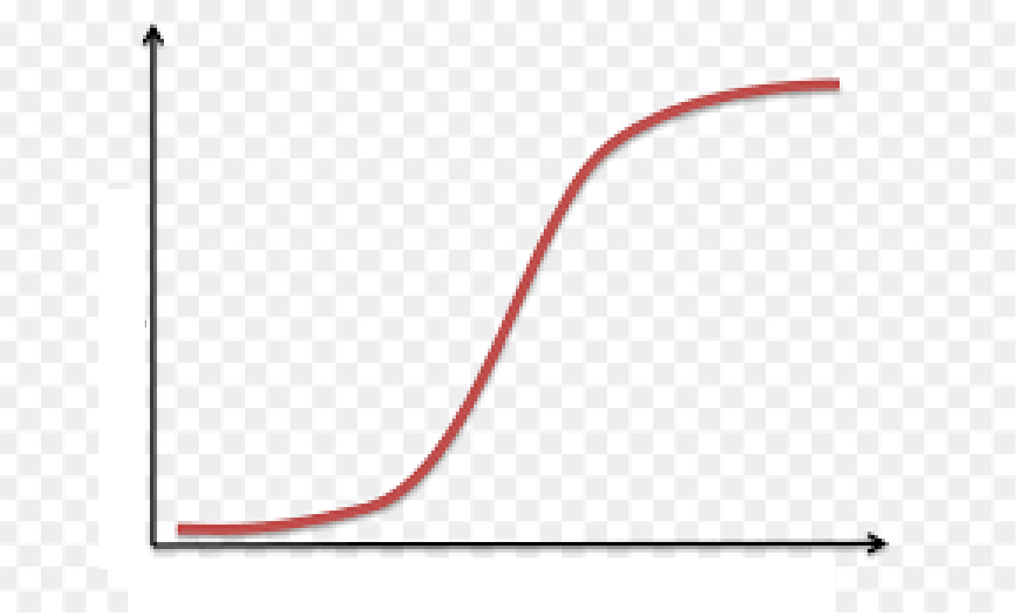
Back-propagation

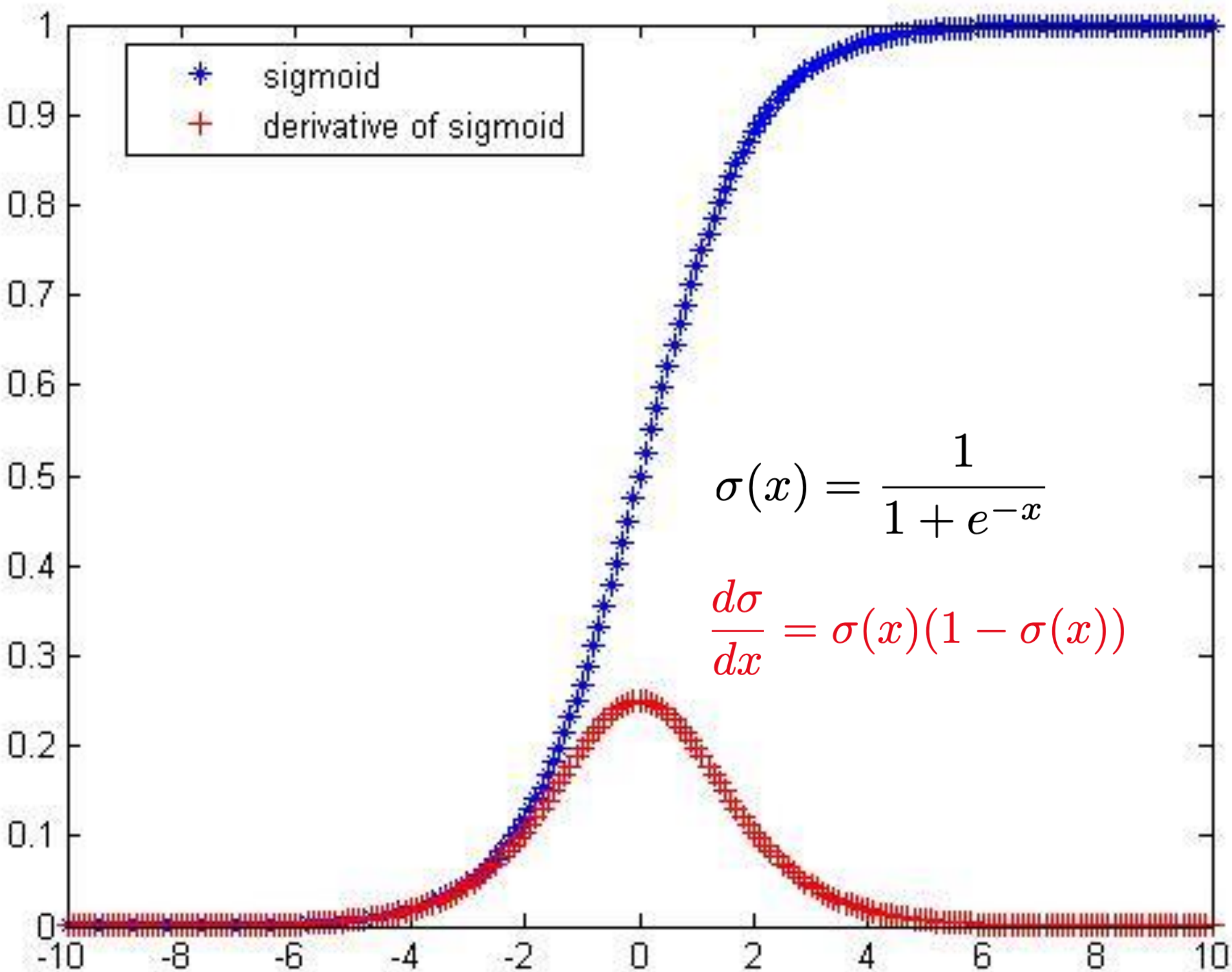


$$w = w - \propto \cdot \frac{\partial E}{\partial w}$$

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

Derivative of Sigmoid




sigmoid
derivative of sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
$\frac{d\sigma}{dx} = \sigma(x)(1-\sigma(x))$
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-8
-6
-4
-2
0
2
4
6
8
10

Desmos.com

Parameters(w, b) tuning for what?

decision boundary

$$wx + b = 0$$

for better <u>decision boundary</u>

Lab 11.py

Classification of an input as 1 or 0

```
import tensorflow as tf
```

↓

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

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

```
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

Adding a bias, b

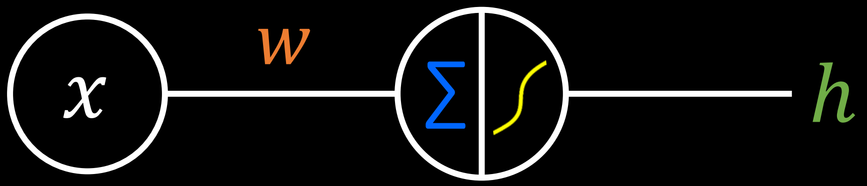
import tensorflow as tf

↓

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

1-Input Neuron

Guess a decision boundary.

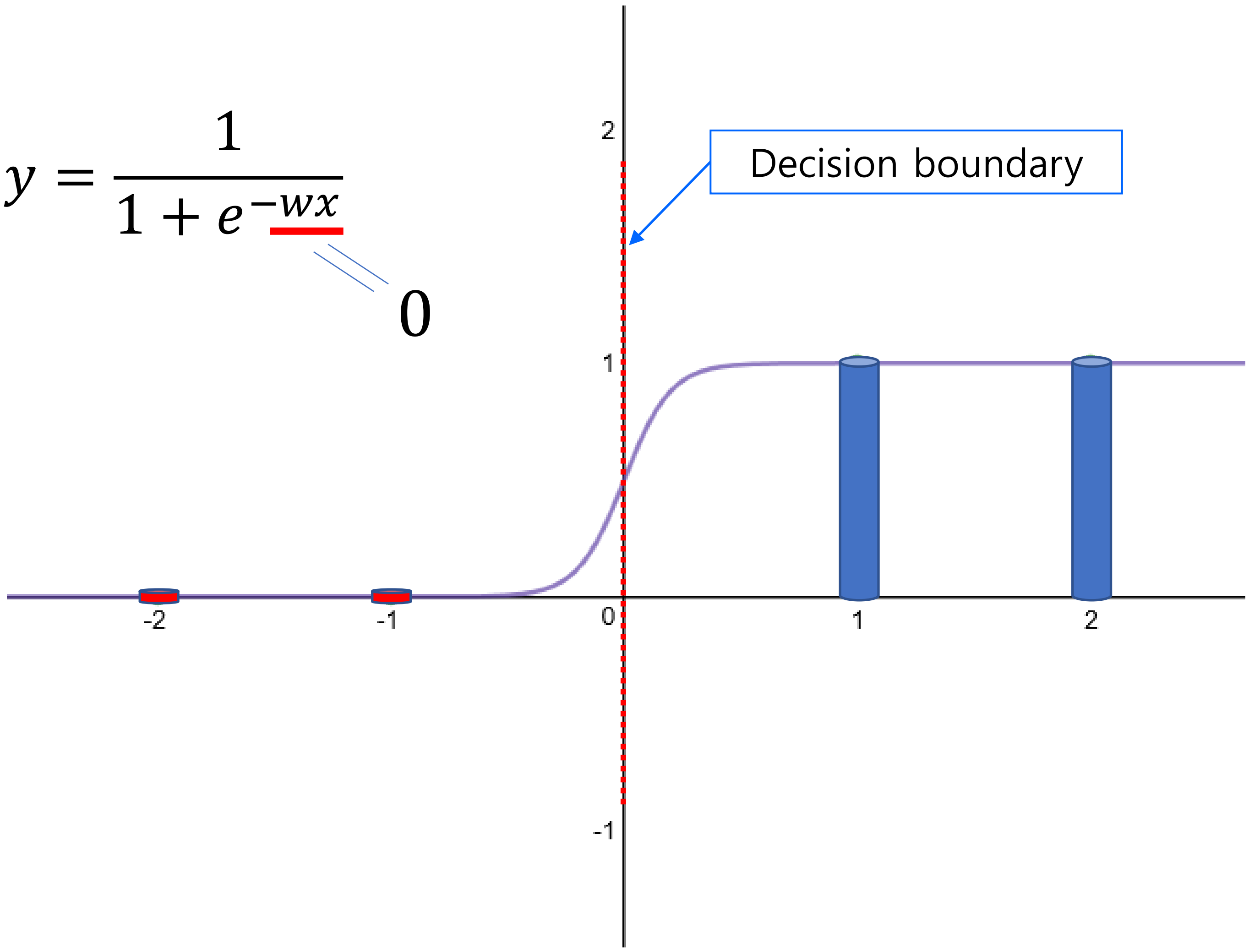


$$h = \frac{1}{1 + e^{-(wx)}}$$

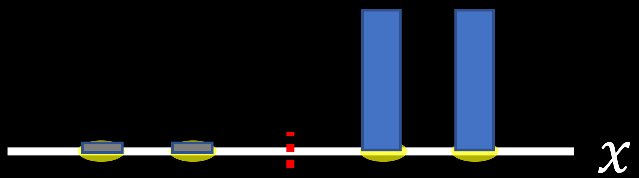
1-Input Neuron

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

0



1-Input(x) Neuron

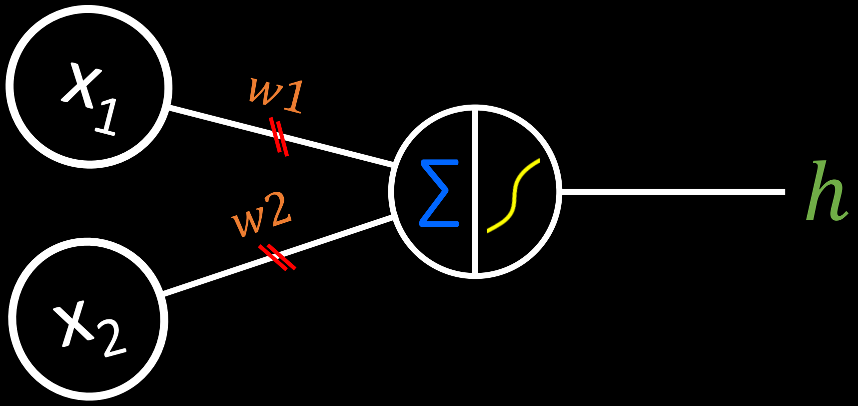


{Side view}

{View from above}

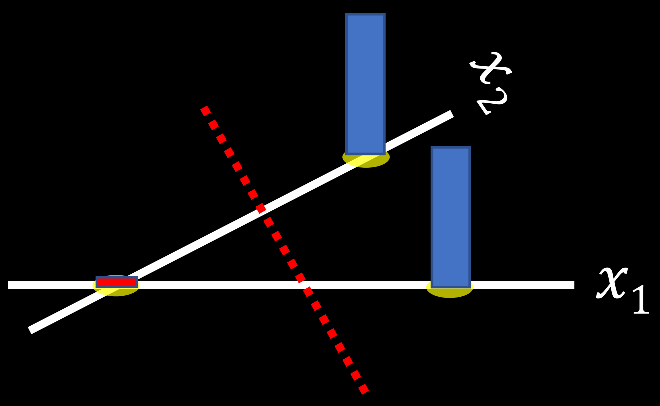
2-Input(x_1, x_2) Neuron

Guess a decision boundary.

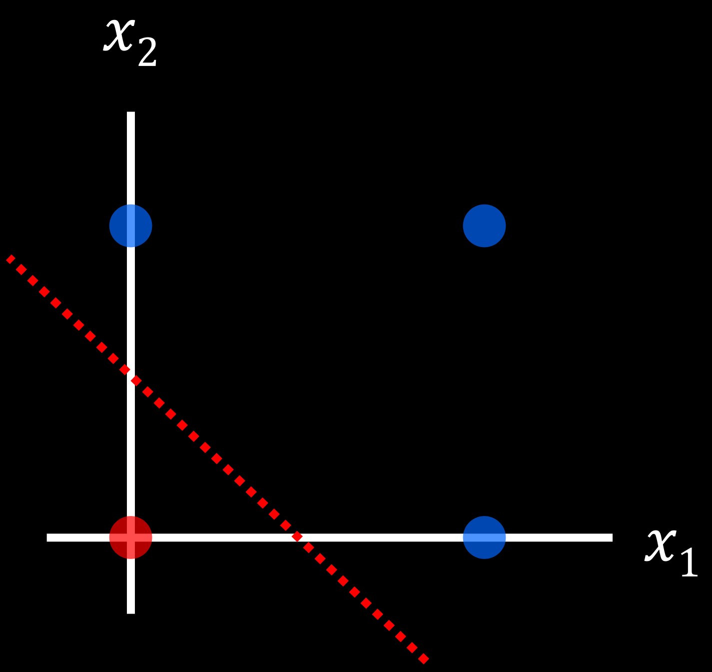


$$h = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}$$

2-Input(x_1, x_2) Neuron



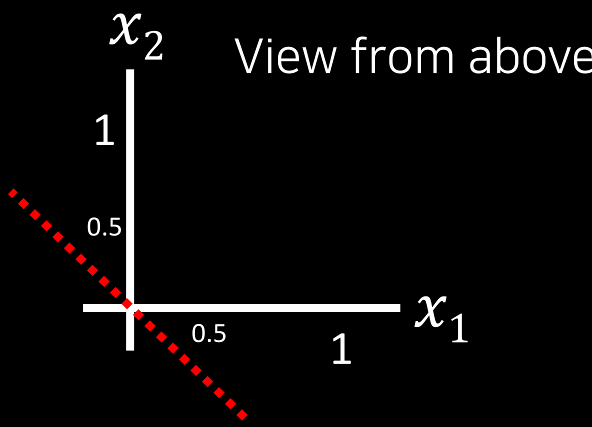
{Side view}



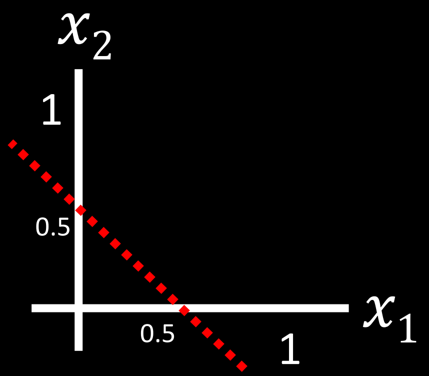
{View from above}

2-Input(x_1, x_2) Neuron

$$x_1 + x_2 = 0$$

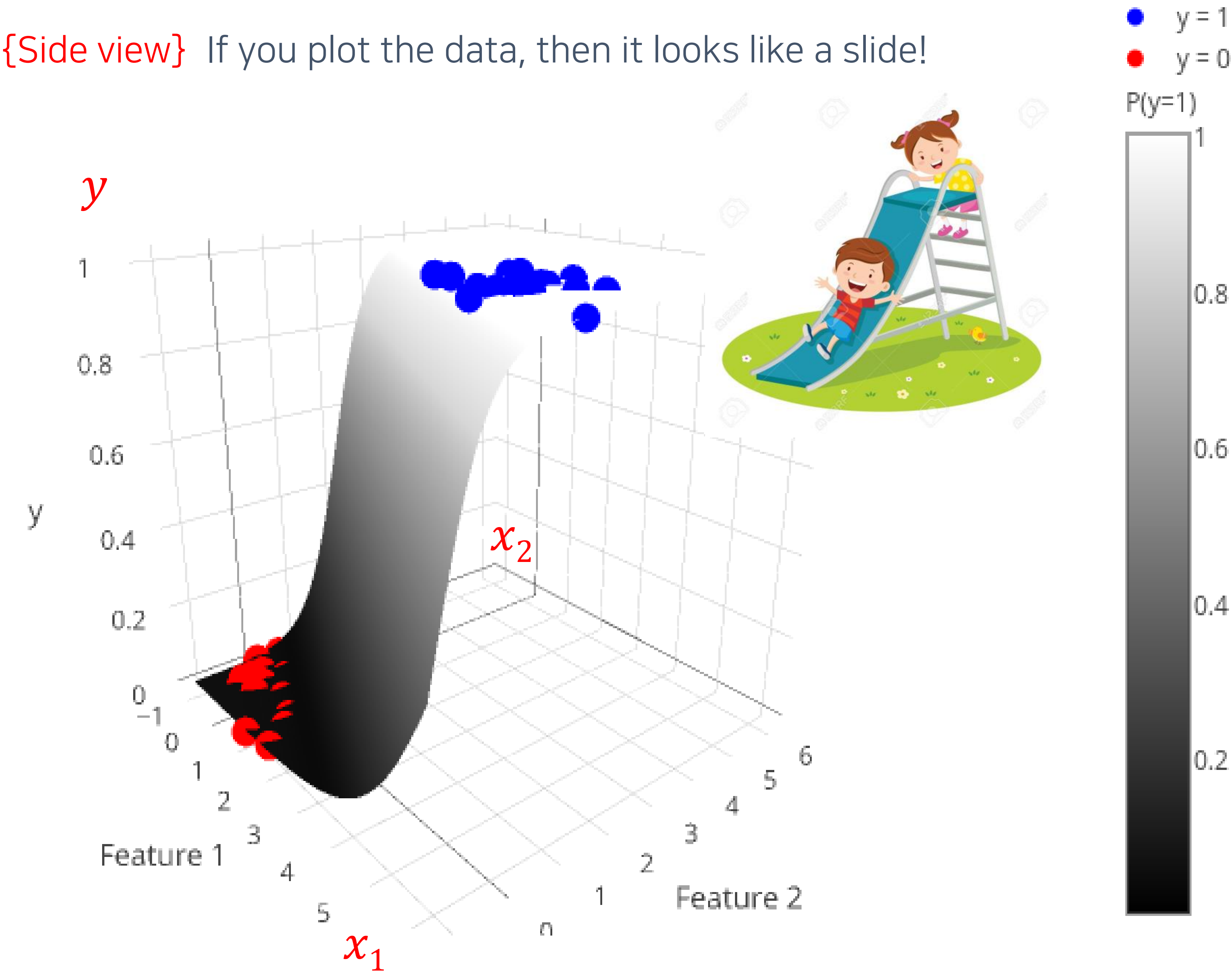


$$x_1 + x_2 - 0.5 = 0$$



Logistic Regression: 2 Features (Inputs)

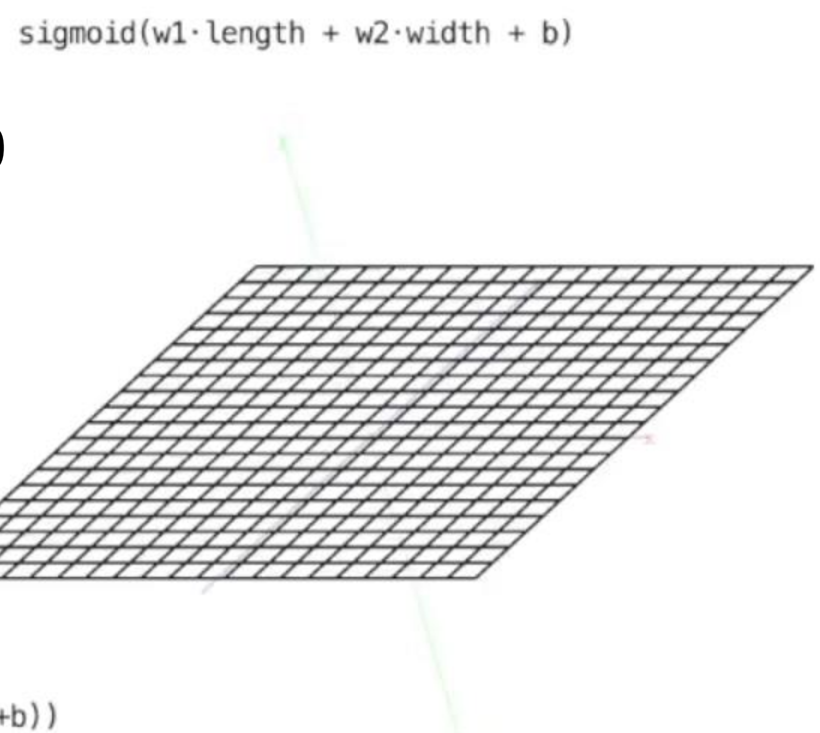
{Side view} If you plot the data, then it looks like a slide!



The meaning of parameters

db slope, db rotation, db shift

$$\underbrace{w_1}_{\text{db slope}} x_1 + \underbrace{w_2}_{\text{db rotation}} x_2 + \underbrace{b}_{\text{db shift}} = 0$$



surface(f(x,z)=sig(w1·x+w2·z+b))

slope → w1 = 0.00

rotation → w2 = 0.00

shift → b = 0.00

Lab 13.py

Implementation of OR gate with a neuron

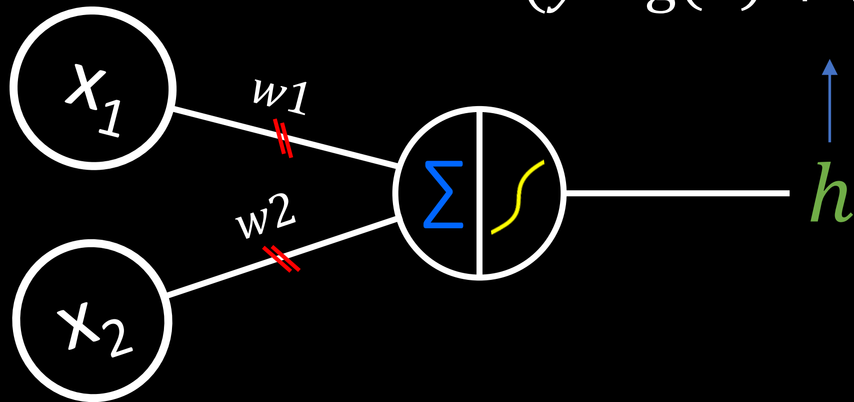
(a decision boundary)

```
import tensorflow as tf
```

↓

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

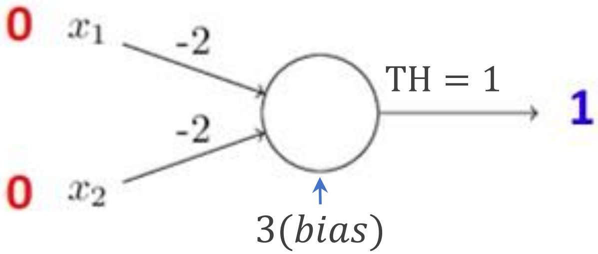
$$E = -(y \log(h) + (1 - y)\log(1 - h))$$



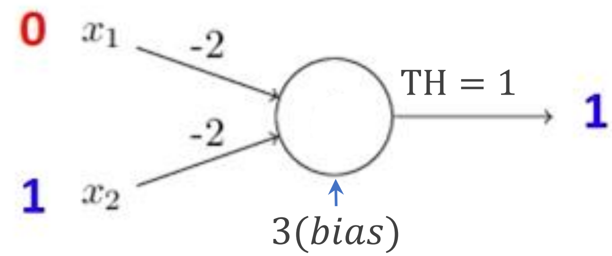
x_1	x_2	$AND(h)$
0	0	0
0	1	0
1	0	0
1	1	1

NAND

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



(-2)*0 + (-2)*0+3=3

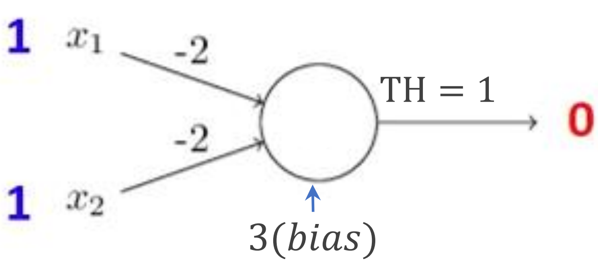


(-2)*0 + (-2)*1+3=1

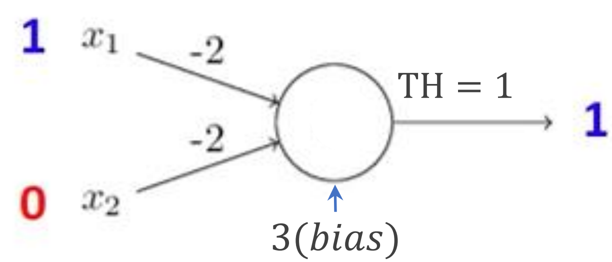
NAND

Truth Table

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



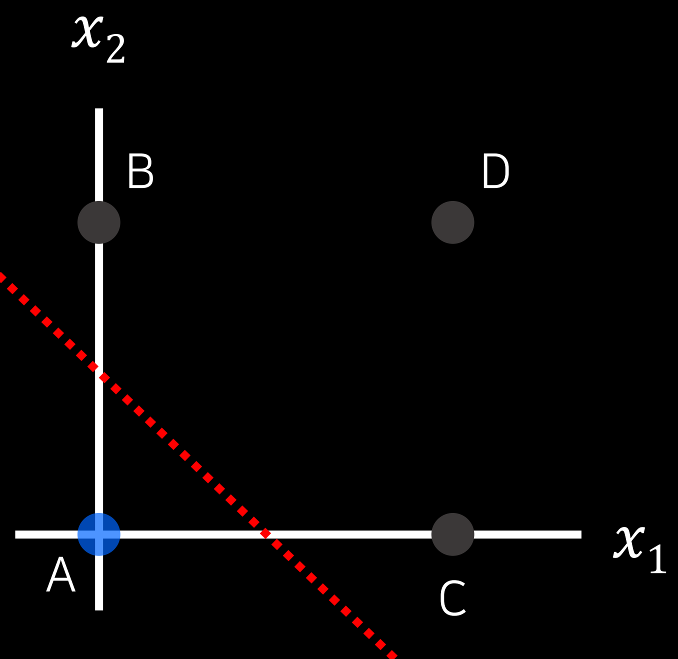
(-2)*1 + (-2)*1+3=-1



(-2)*1 + (-2)*0+3=1

Decision boundary by a neuron

two inputs



View from above

Decision boundary by a neuron

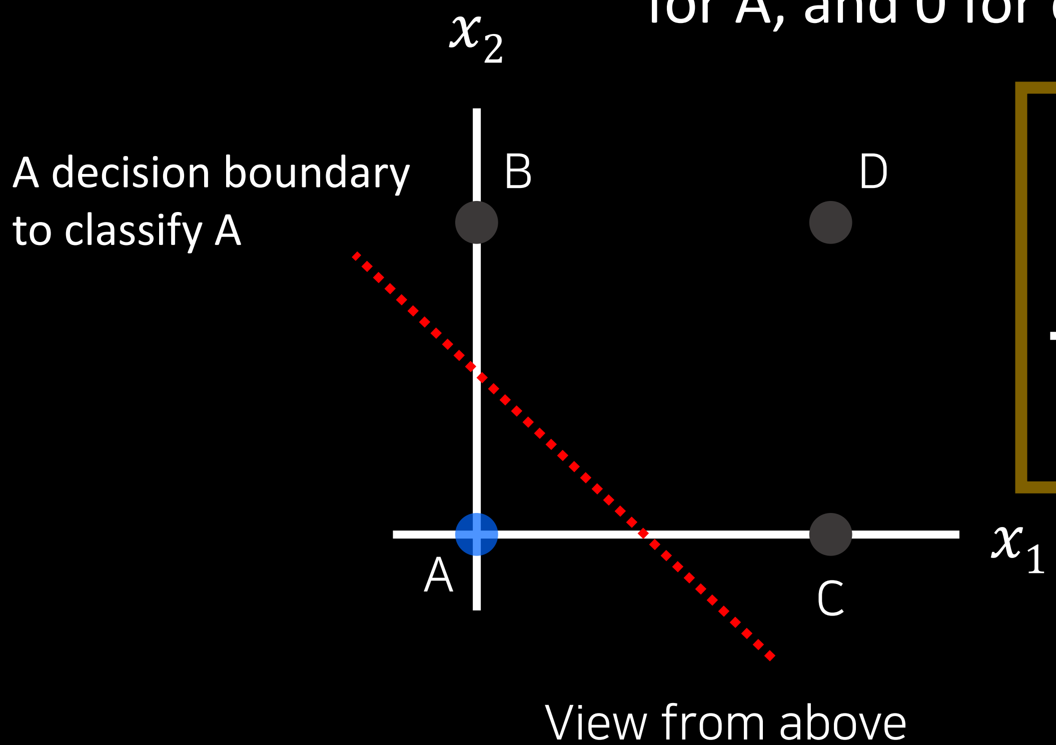
two inputs

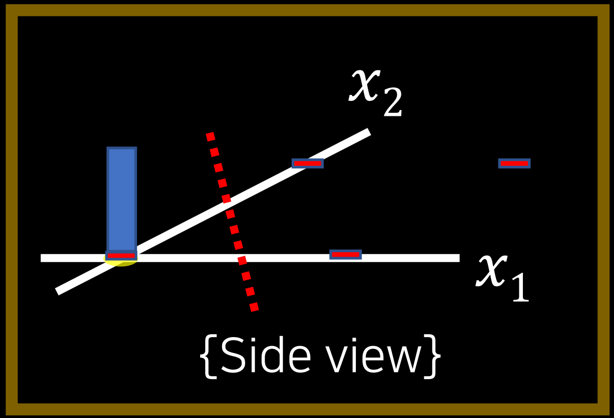
- A neuron, only 1 linear decision boundary
- A decision boundary yielding 2 classes (1 or 0)
- How to solve multiple classes more than 2

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

Neuron #1

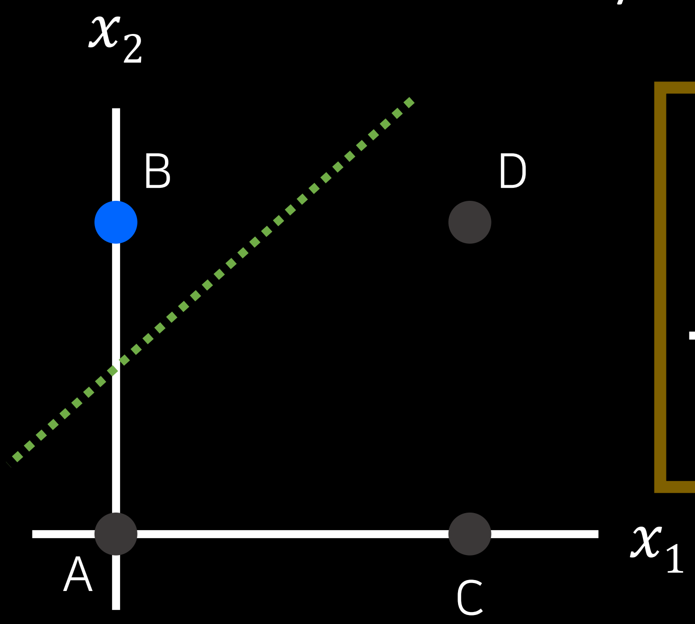
The output of a neuron is 1 for A, and 0 for other cases.

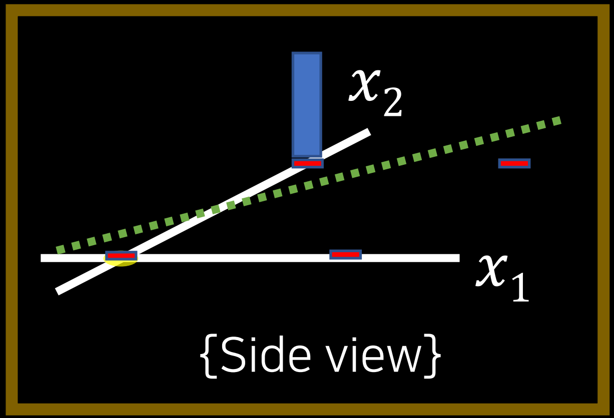




Neuron #2

2nd neuron for 2nd decision boundary to classify B

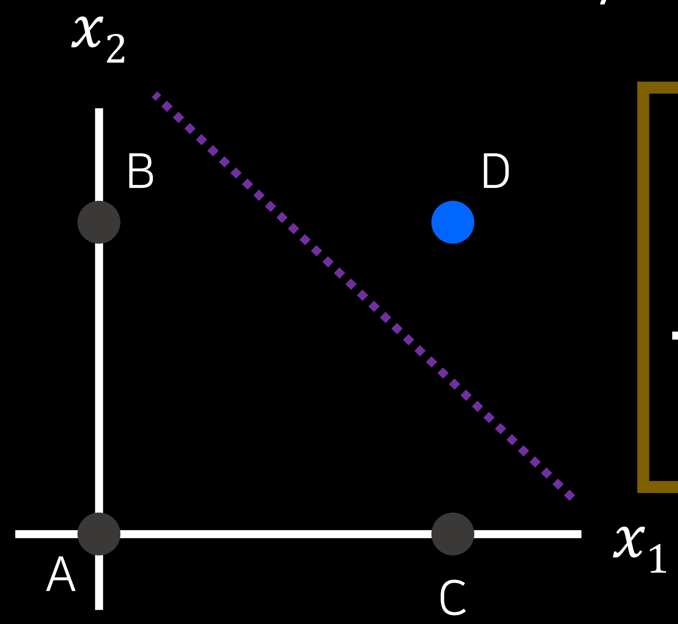




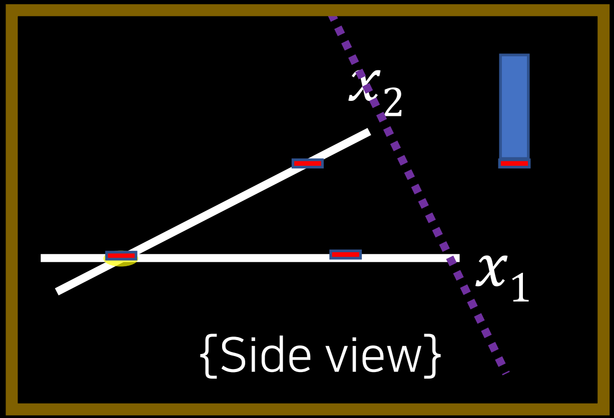
View from above

Neuron #3

3rd neuron for 3rd decision boundary to classify D

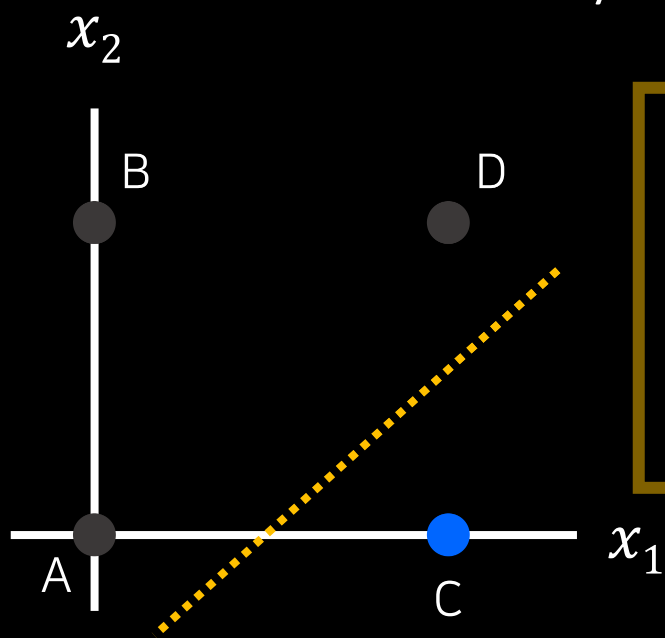


View from above

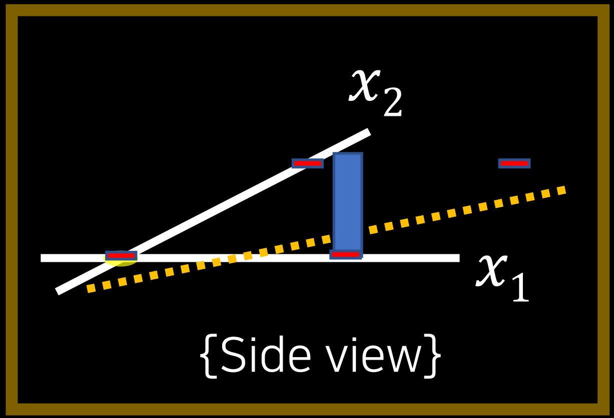


Neuron #4

4th neuron for 4th decision boundary to classify C

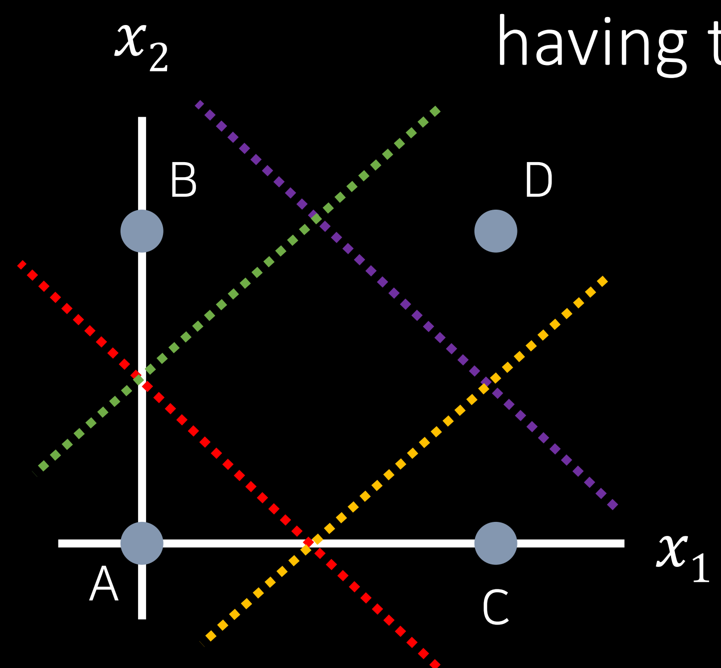


View from above



4 Neurons

4 neurons for
4 decision boundaries
having the same inputs

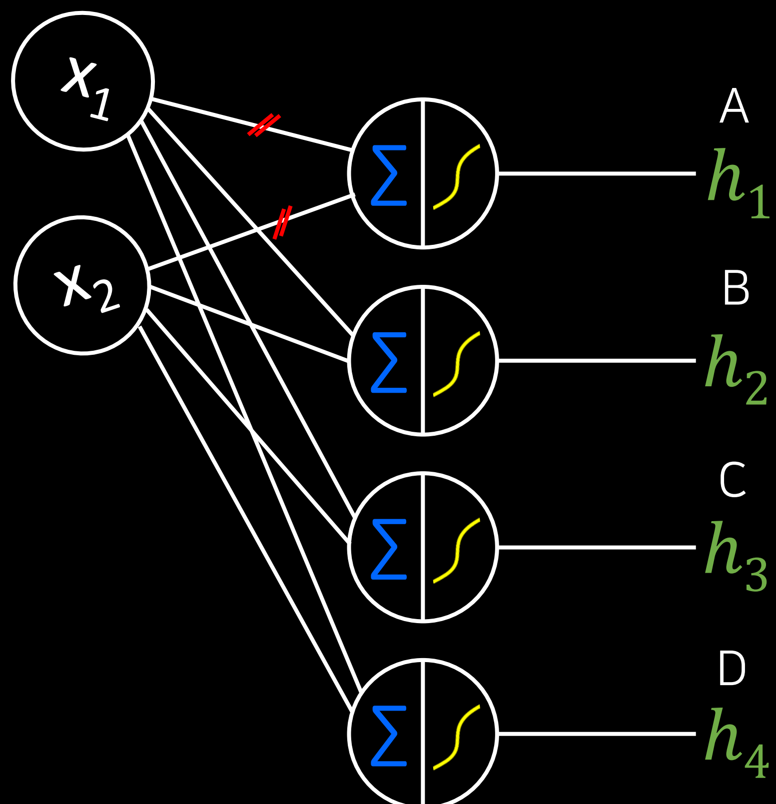


View from above

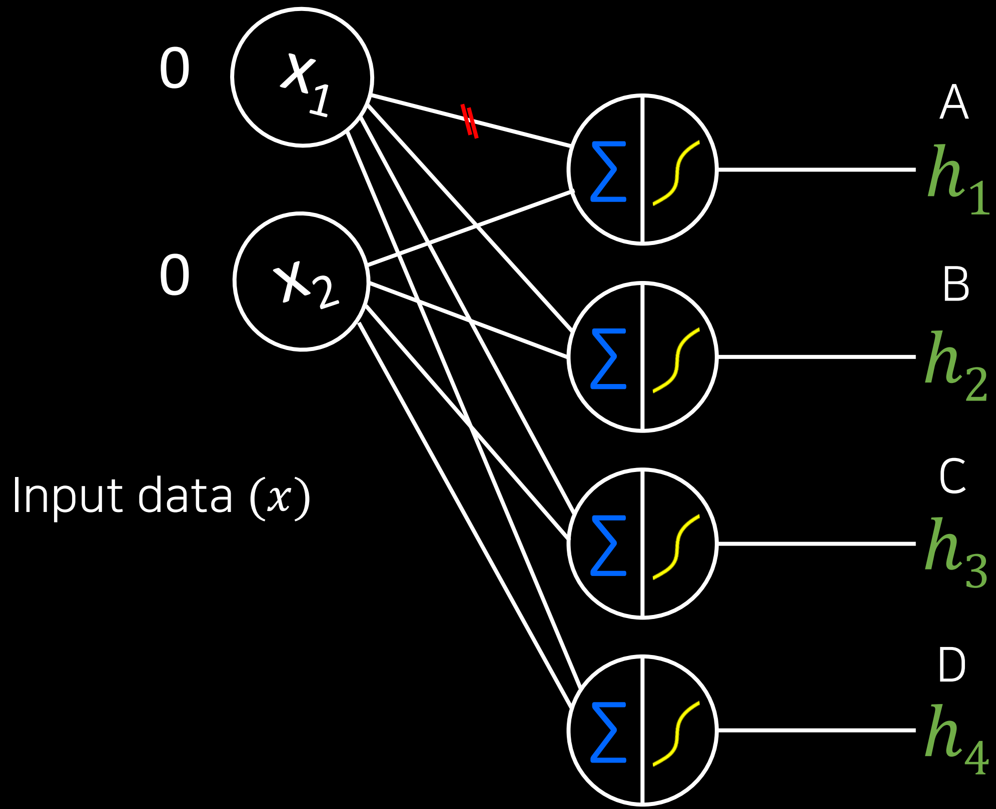
4 Neurons

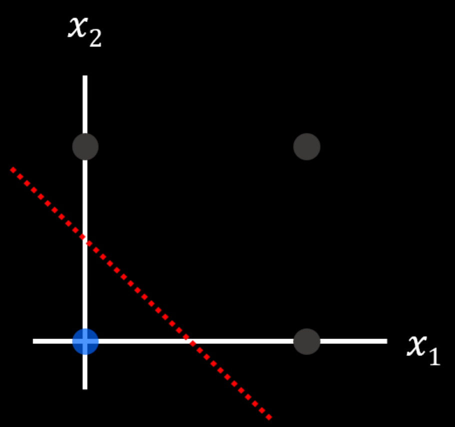
Matrix notation

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

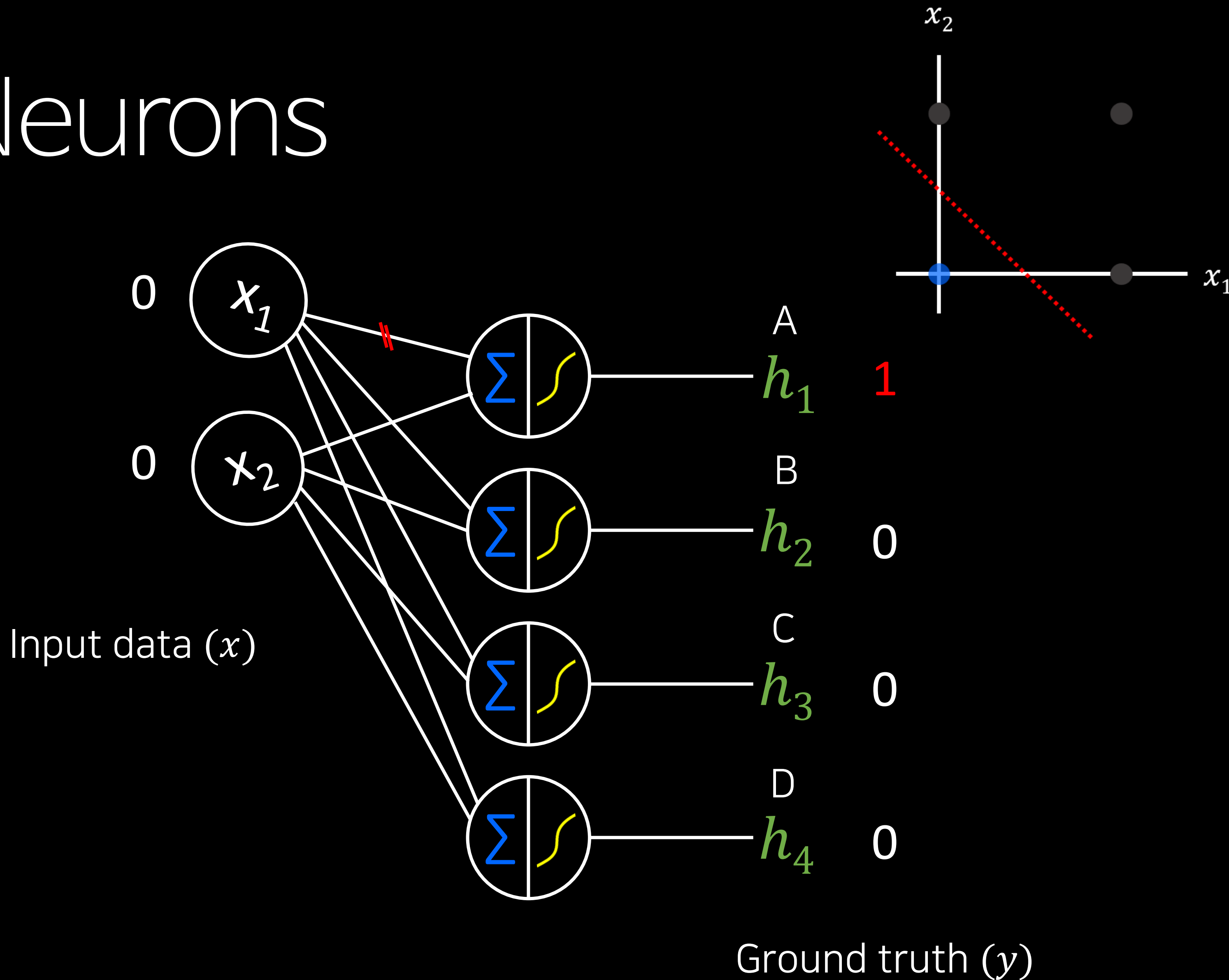


4 Neurons

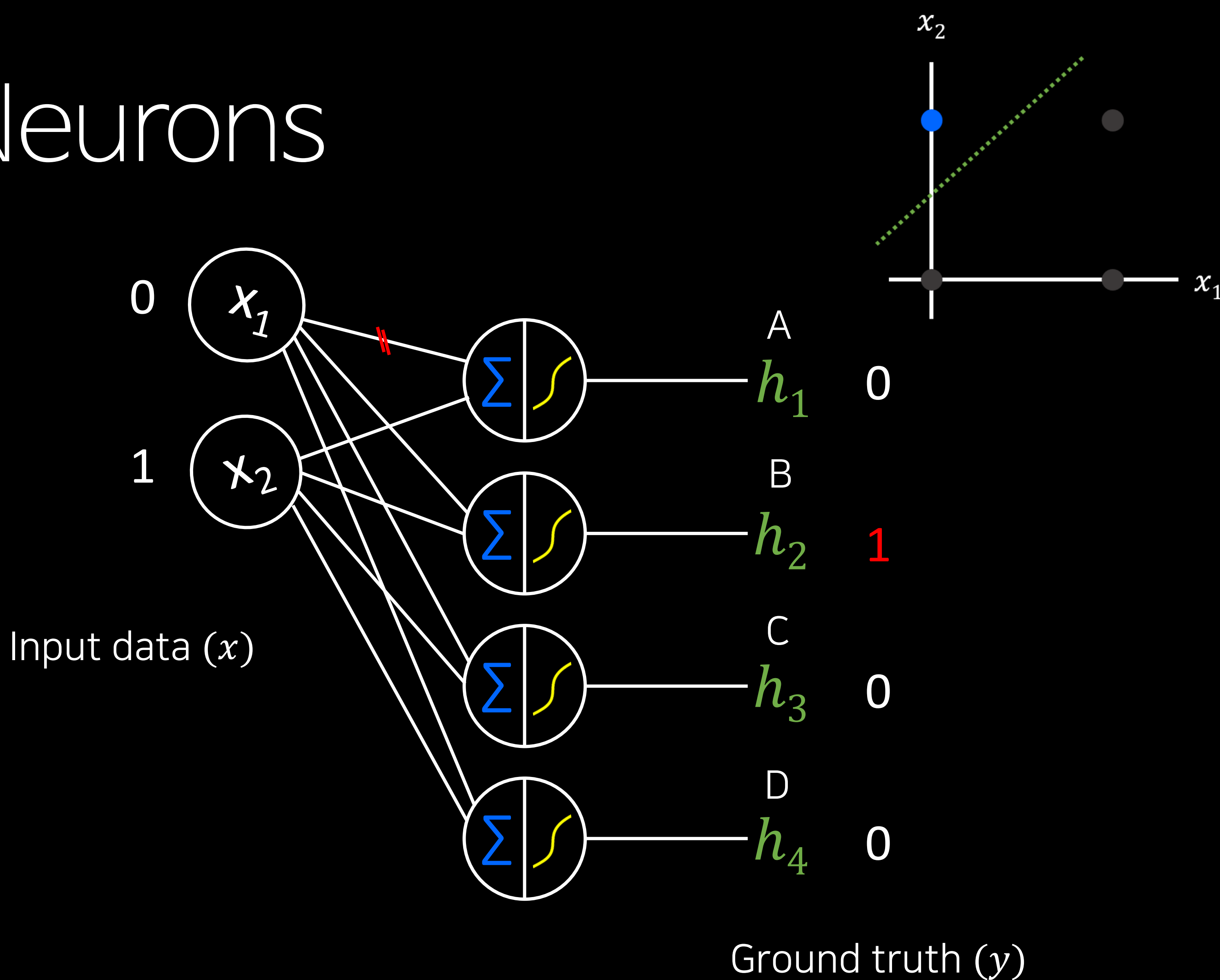




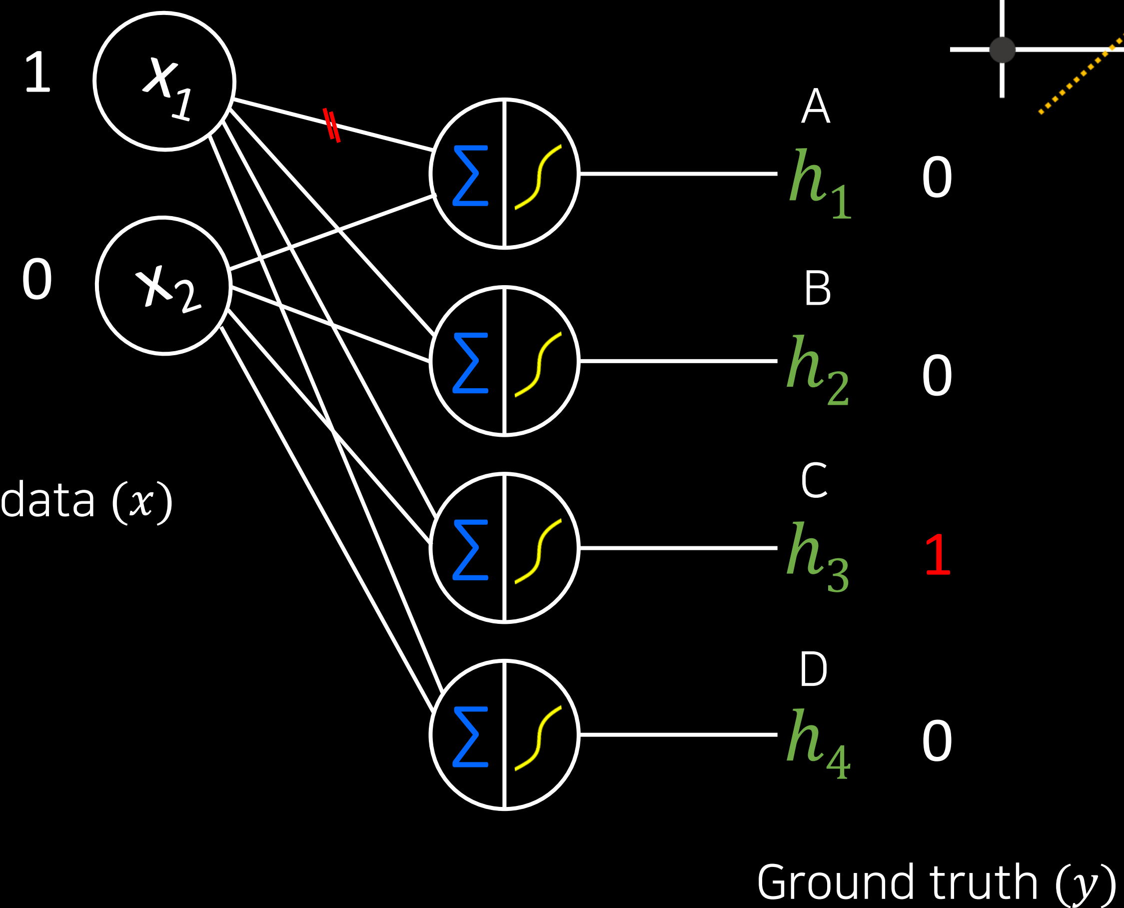
4 Neurons

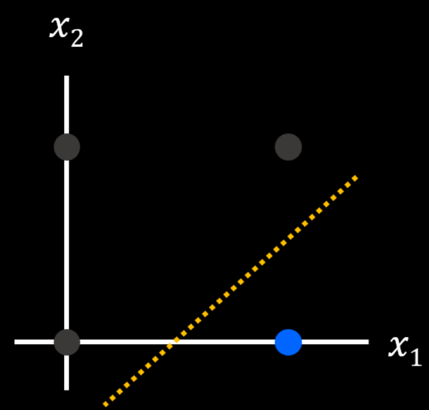


4 Neurons

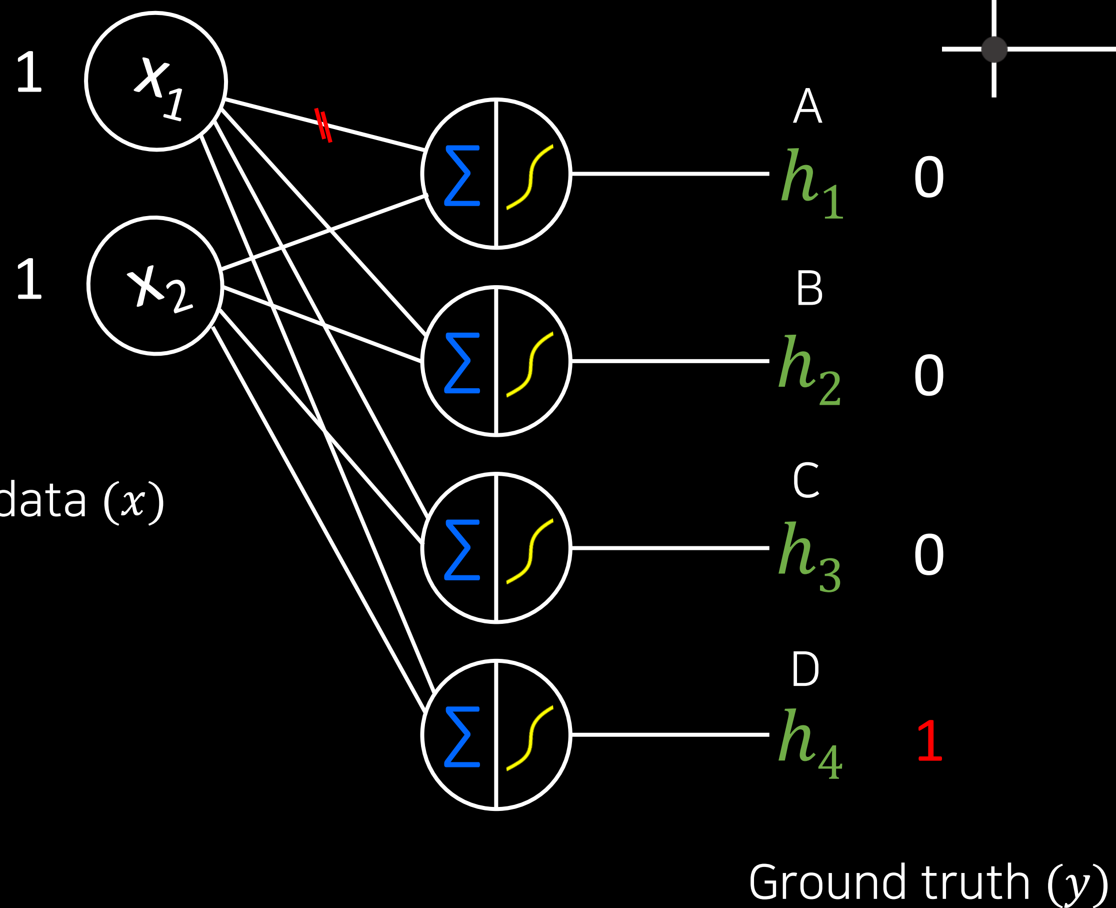


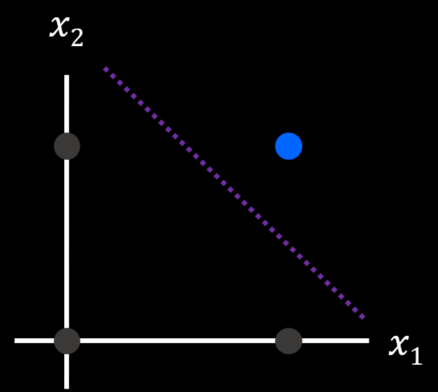
4 Neurons





4 Neurons





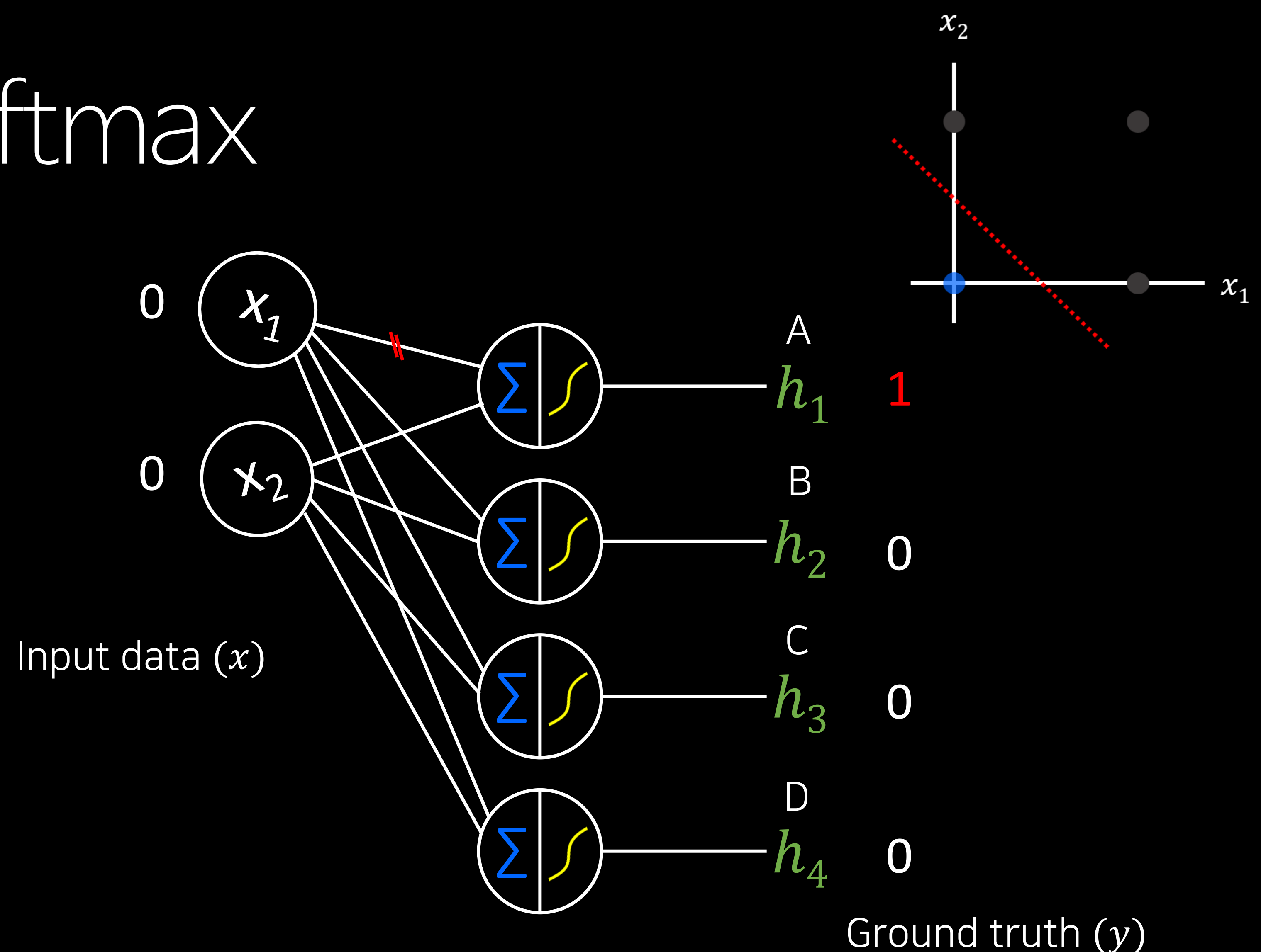
One-hot Encoding

- For the ground truth (y),
- setting only one output as ON(1) and others as OFF(0) → One-hot encoding

Considerations

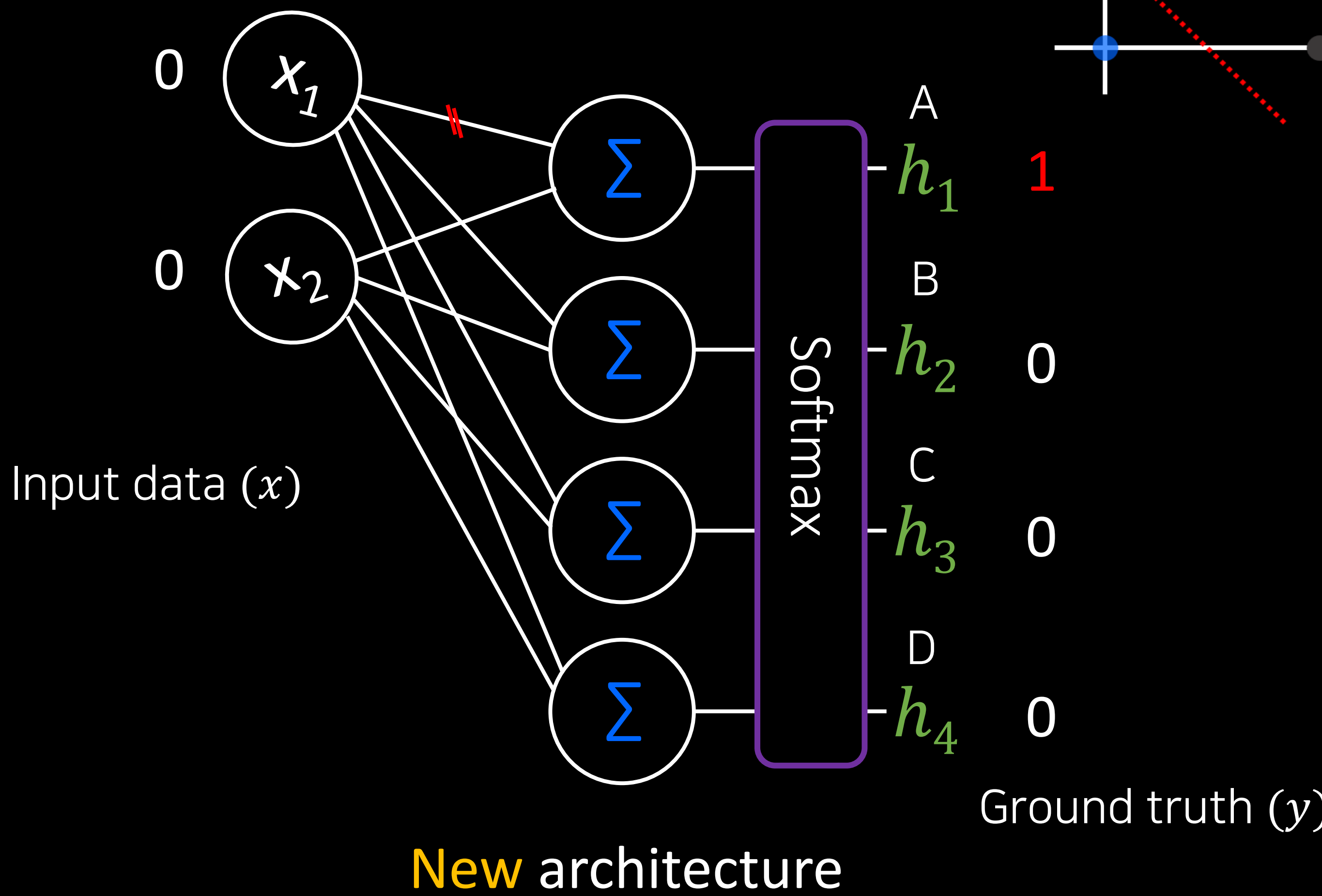
- If a neuron's output is 1, then others must be 0.
- However, each neuron produces output independently.
- No way to control the 4 outputs together
- A special function introduced → Softmax

Softmax

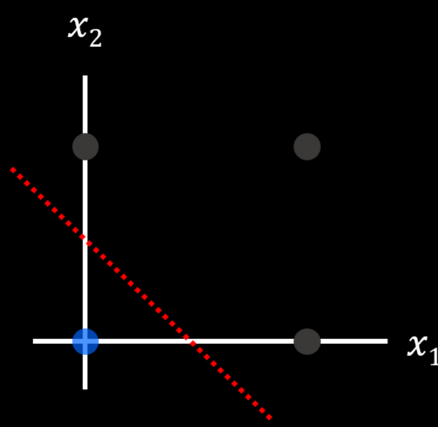


Initial architecture

Softmax

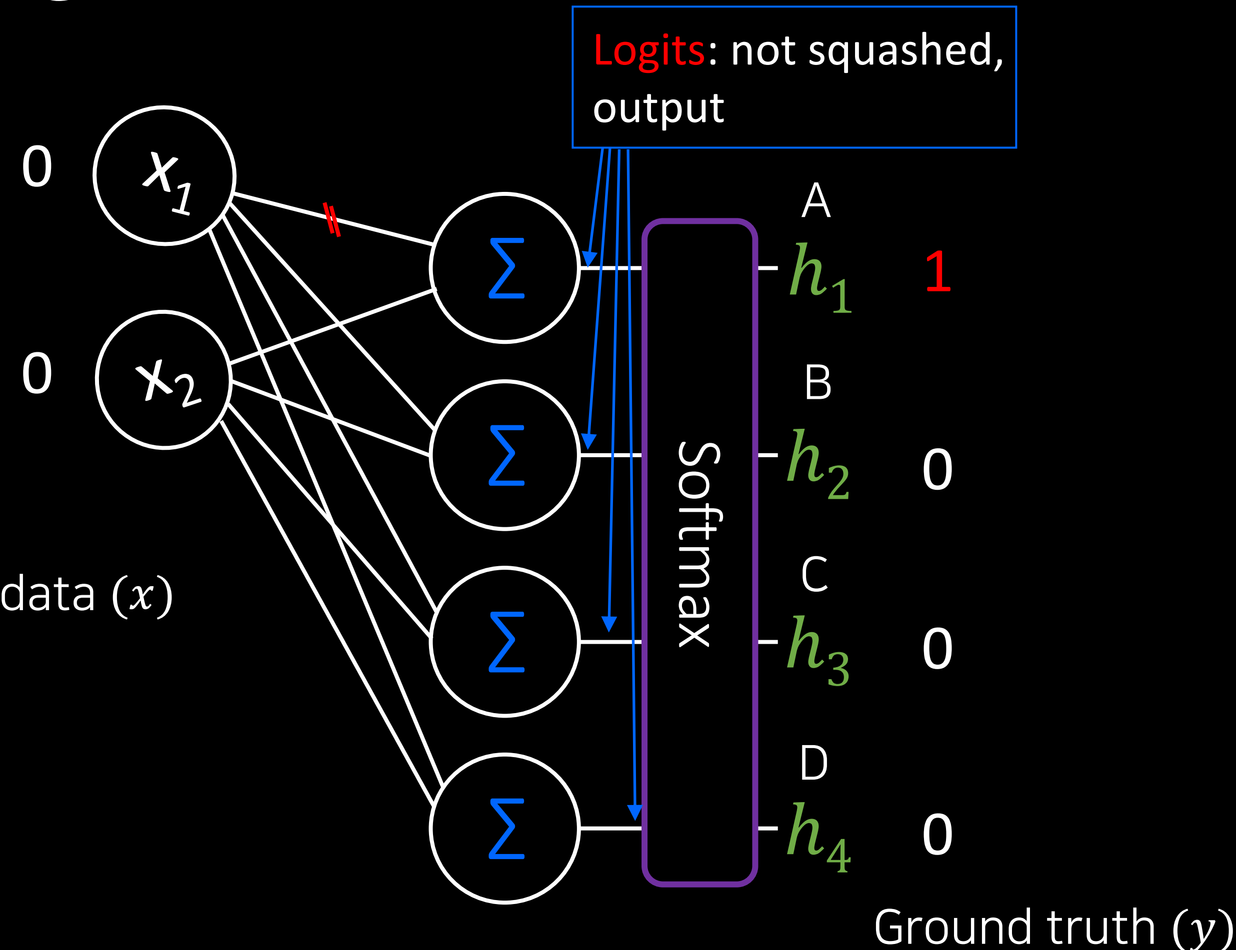


0
x_1
0
x_2
Input data (x)
Softmax
A
h_1
1
B
h_2
0
C
h_3
0
D
h_4
0
Ground truth (y)
New architecture



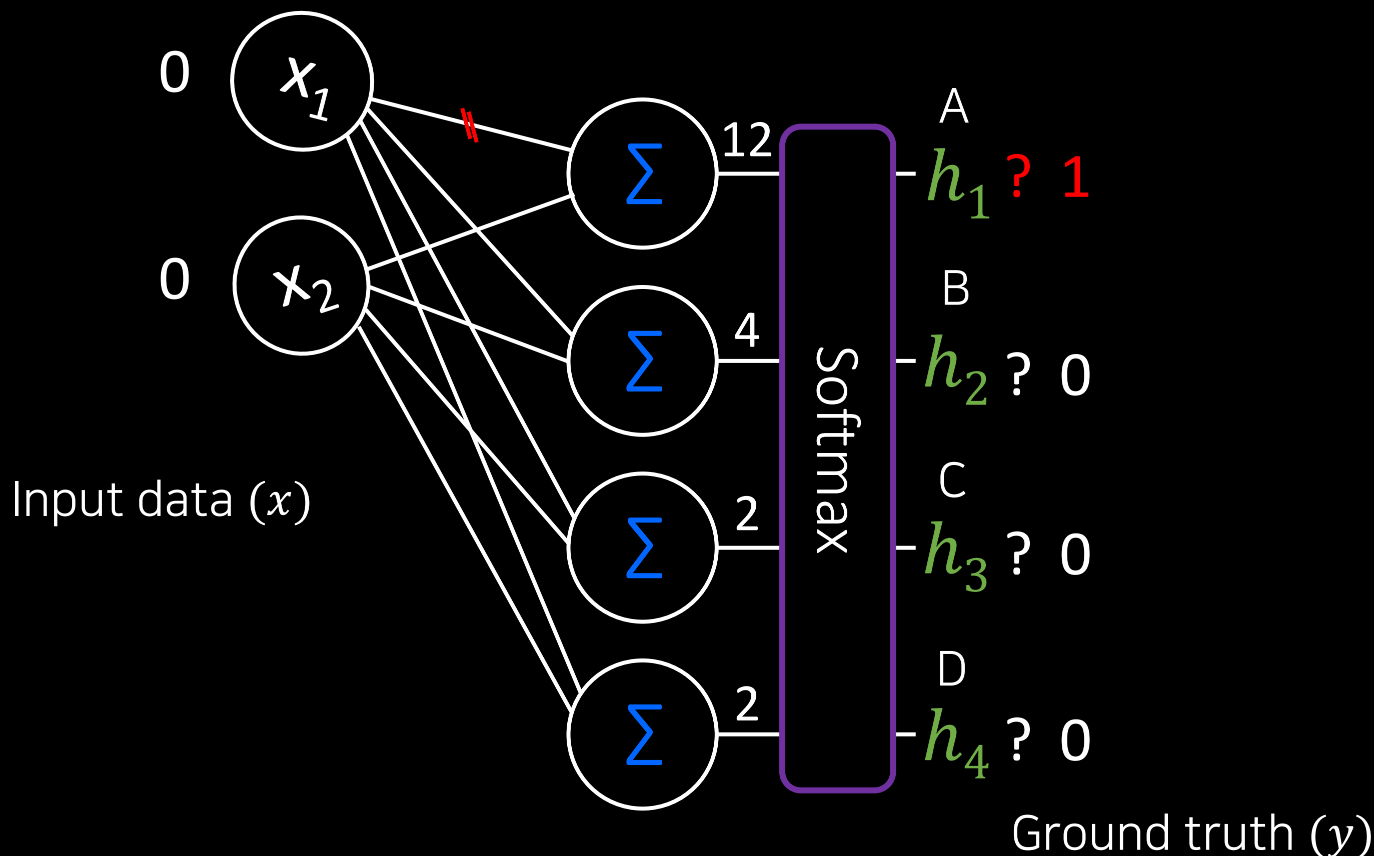
x_2
x_1

Softmax



New architecture

Softmax

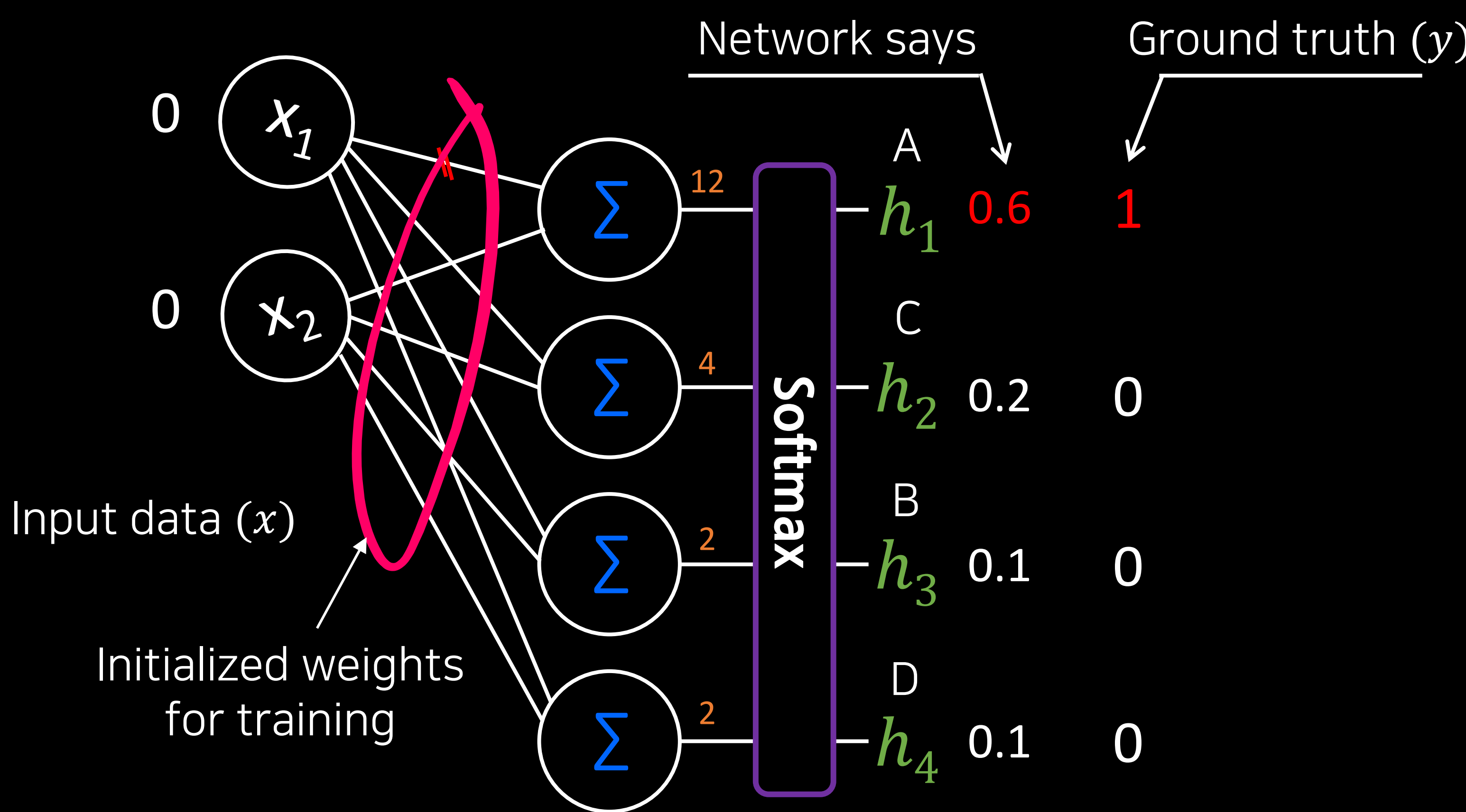


New architecture

Softmax

- for 12, 4, 2, 2, the Softmax function returns $\frac{12}{20}, \frac{4}{20}, \frac{2}{20}, \frac{2}{20}$.
- Normalization of logits values
- The probability for each class

Softmax



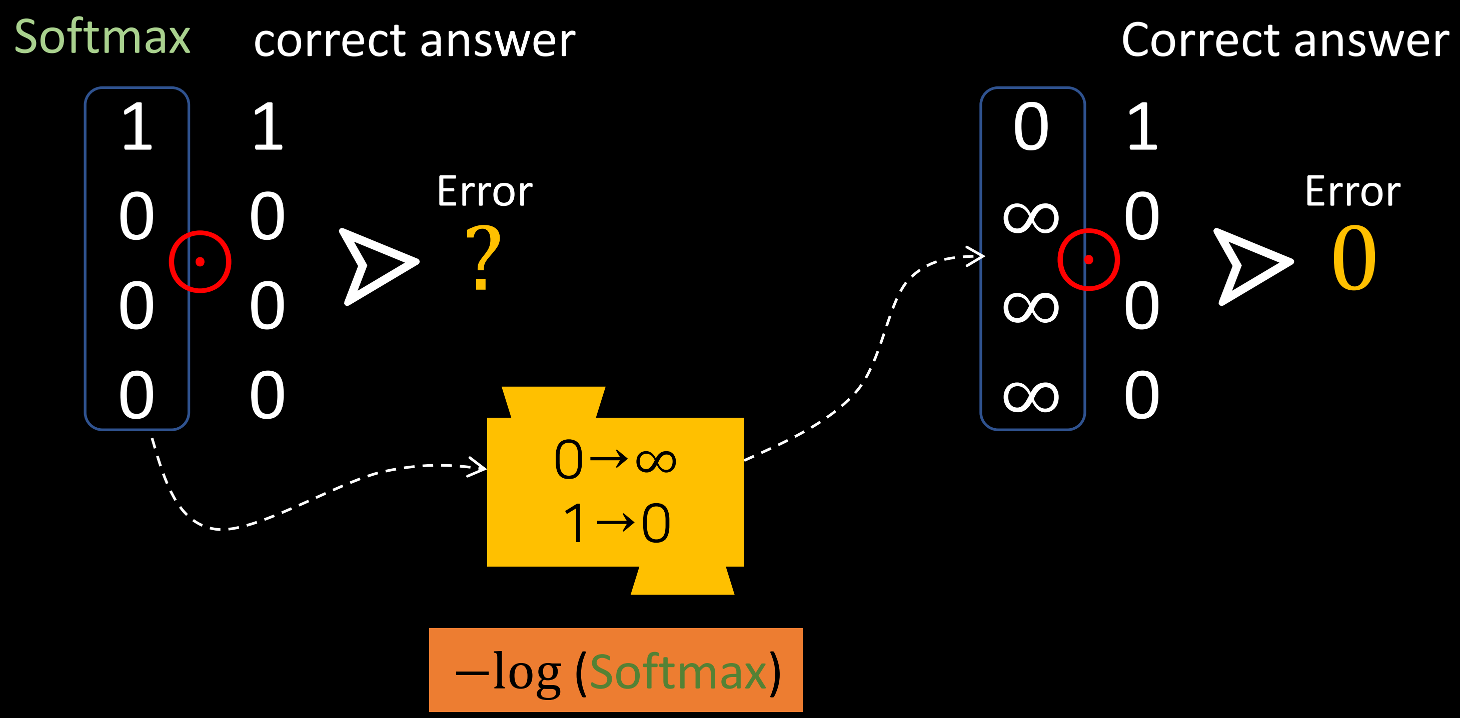
New architecture

New Error Function

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

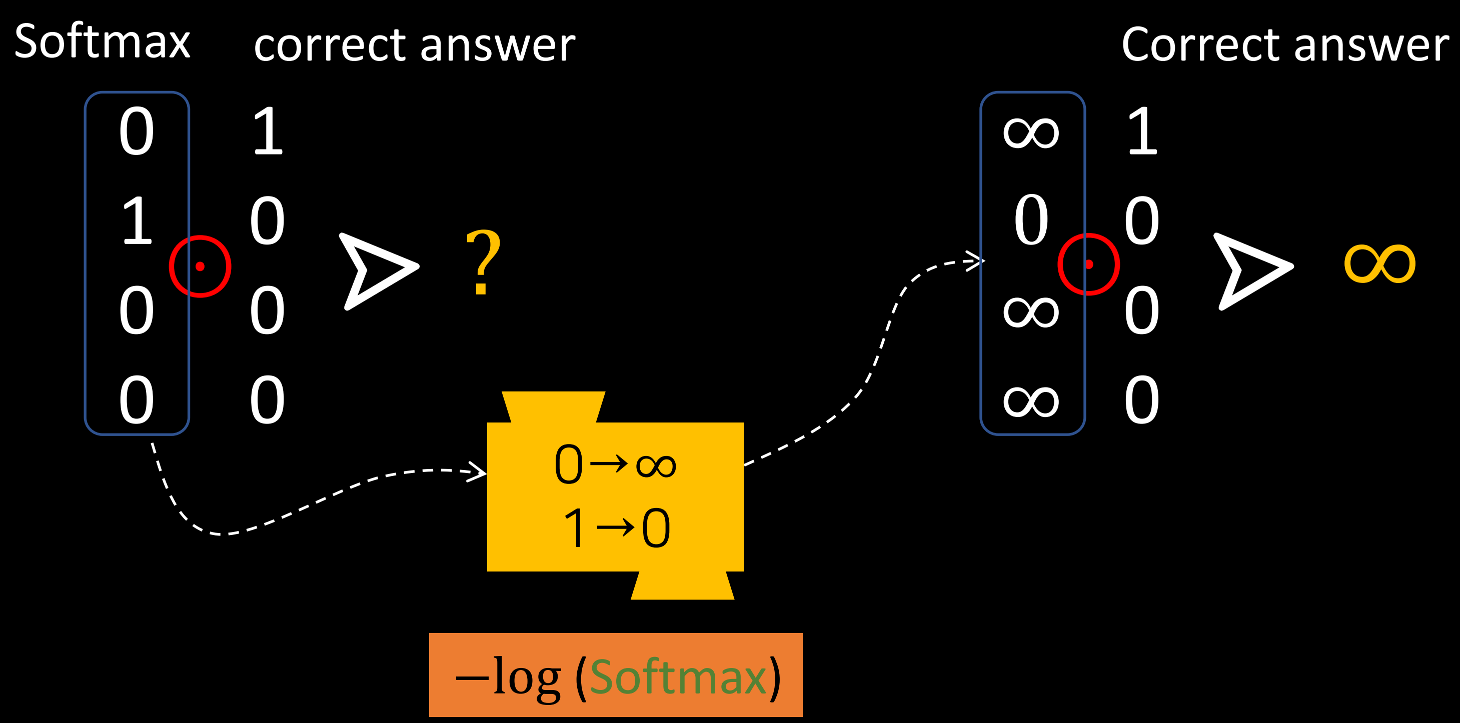
New Error Function

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



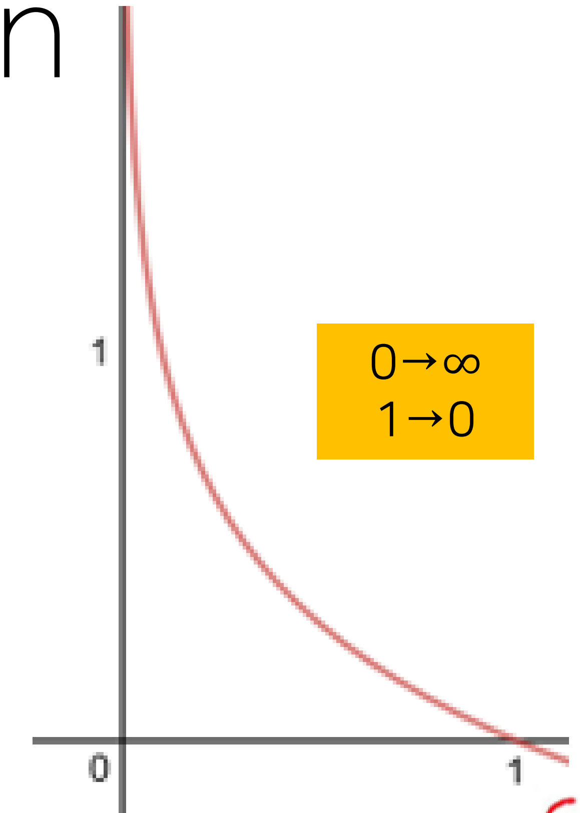
New Error Function

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



$-\log$ function

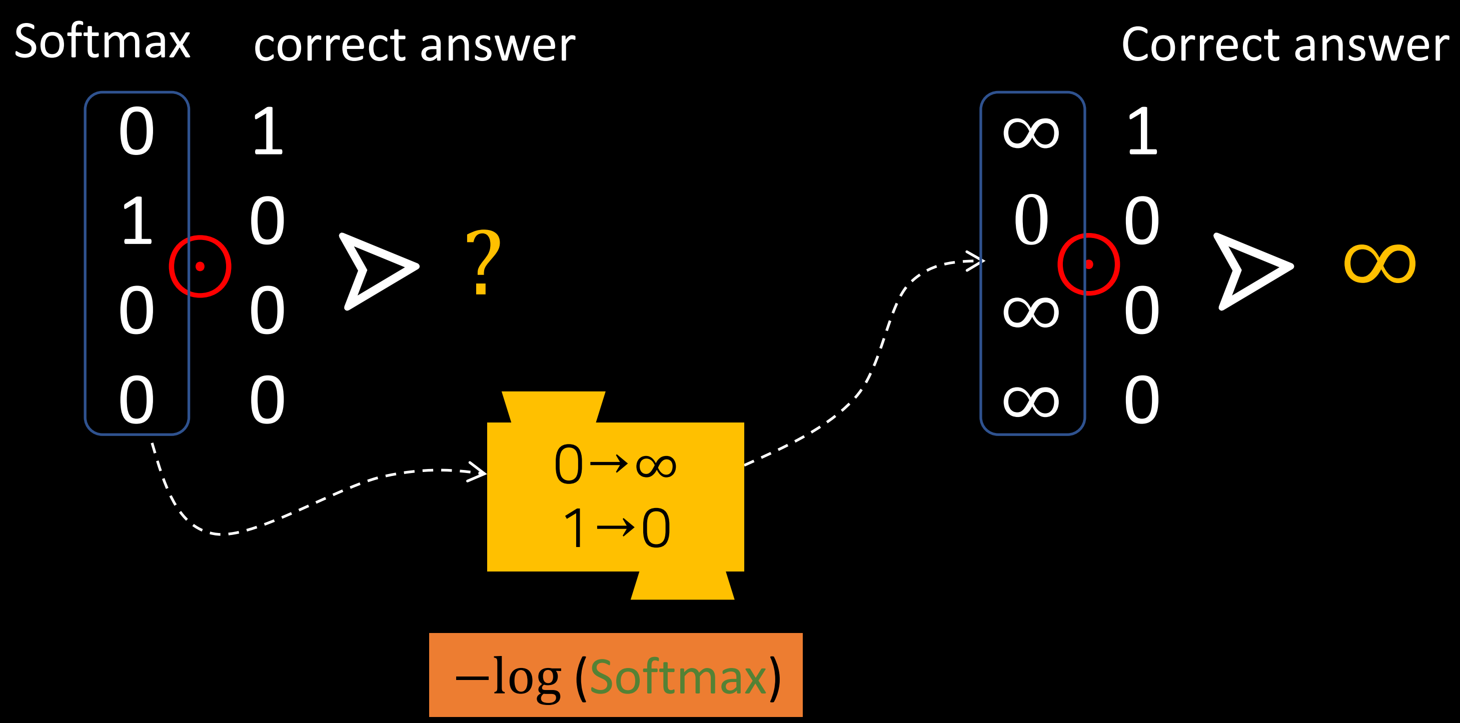
$$-\log(S)$$



S = Softmax

New Error Function

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



New Error Function

$$-L \log(S)$$

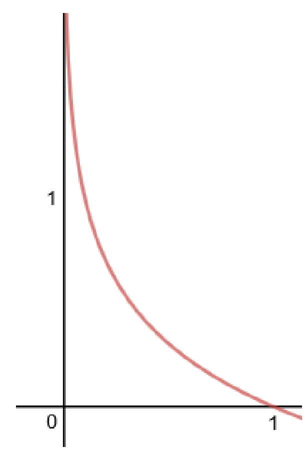
correct answer *L*

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

New Error Function

$$D(S, L) = -\sum_i L_i \log(S_i)$$

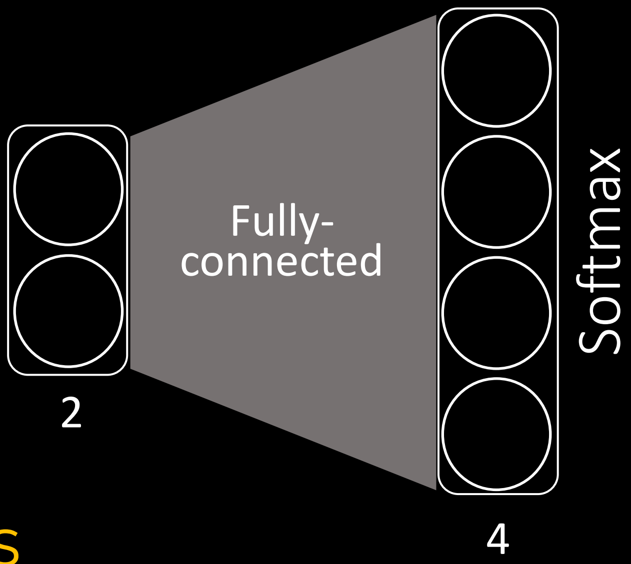
$$\underbrace{\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}}_{S(y)} \qquad \underbrace{\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}}_{L}$$



softmax_cross_entropy_with_logits(logits, y_data)

- The function returns 0 if the answer is correct,
- or returns ∞ if the answer is totally incorrect.

Lab 14.py



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

```
import tensorflow as tf
```

↓

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```