One hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy.

XOR, perceptron, and Multi-layer Neural Networks

Why can't a perceptron model handle XOR,

but multi-layer neural networks can?

ENLP 2025 Spring
Xiulin Yang

XOR

XOR (exclusive OR) returns true if and only if one of the two conditions is true, but not both.

OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

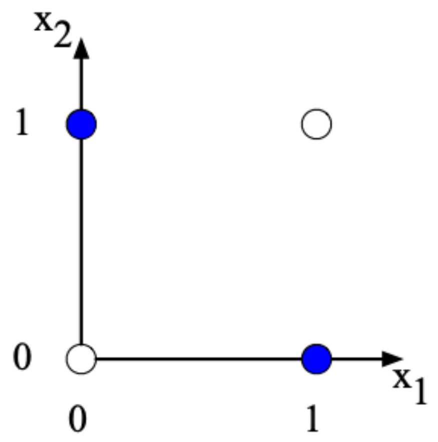
XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR

XOR (exclusive OR) returns true if and only if one of the two conditions is true, but not both.

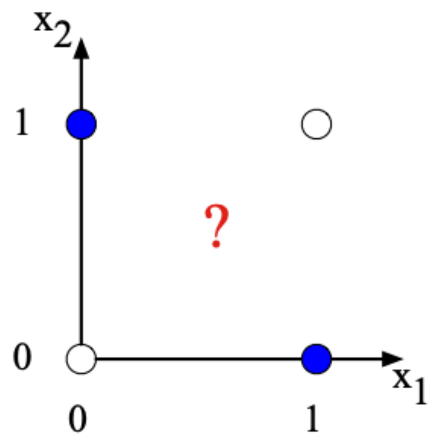
AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0



XOR

XOR (exclusive OR) returns true if and only if one of the two conditions is true, but not both.

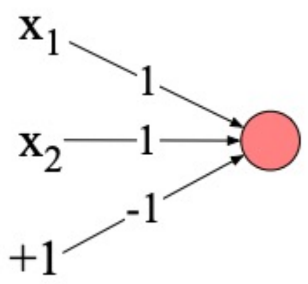
AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0



Why perceptron fails?

- A perceptron can handle ***and*** & ***or***
- This is because a perceptron is essentially a ***linear*** classifier.

perceptron



step function

$$y = \begin{cases} 0, & \text{if } \mathbf{w}\cdot\mathbf{x}+b \leq 0 \\ 1, & \text{if } \mathbf{w}\cdot\mathbf{x}+b > 0 \end{cases}$$

AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

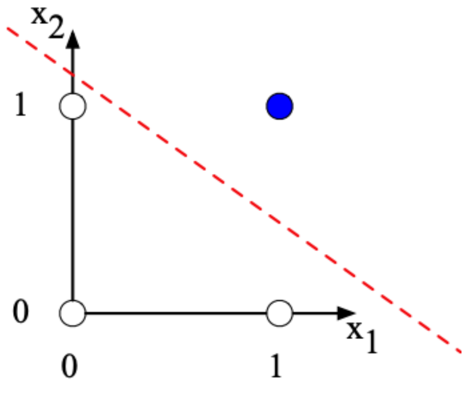
y = a(X1*w1+x2*w2+b)

0*1+0*1-1 =-1 →0

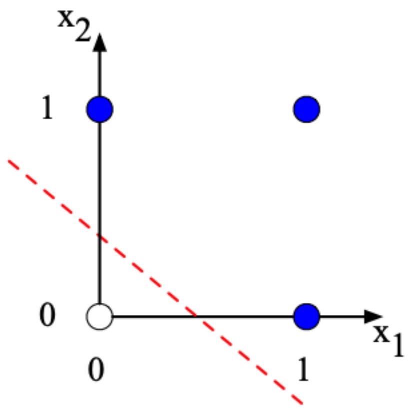
0*0+1*1-1 =0 →0

1*1+0*1-1 =0 →0

1*1+1*1-1 =1 →1



a) x_1 AND x_2

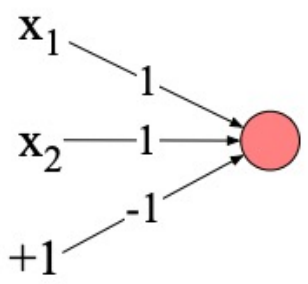


b) x_1 OR x_2

Why perceptron fails?

- A perceptron can handle ***and*** & ***or***
- This is because a perceptron is essentially a ***linear*** classifier.

perceptron



step function

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

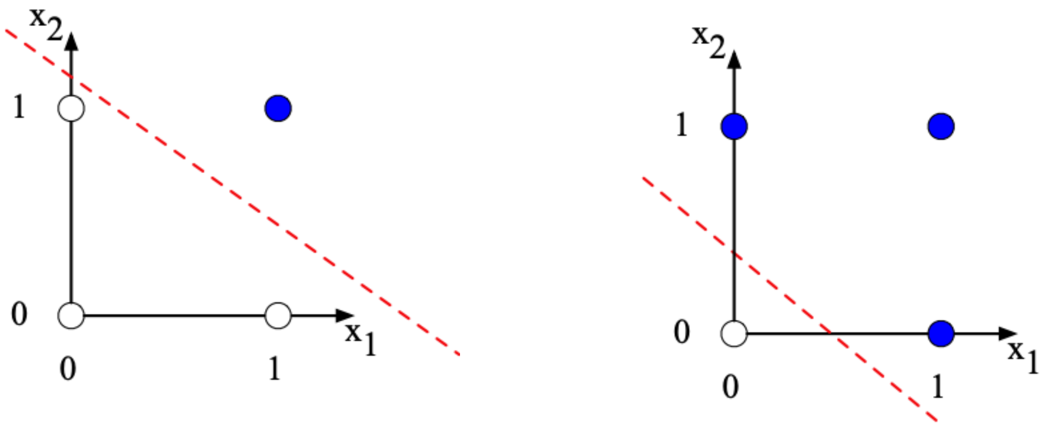
y = a(X1*w1+x2*w2+b)

0*1+0*1-1 =-1 →0

0*0+1*1-1 =0 →0

1*1+0*1-1 =0 →0

1*1+1*1-1 =1 →1



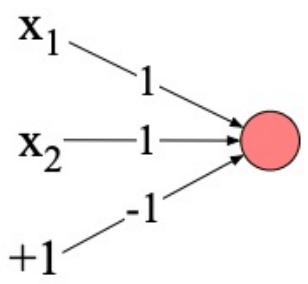
a) x_1 AND x_2

b) x_1 OR x_2

Why perceptron fails?

- A perceptron can handle ***and*** & ***or***
- This is because a perceptron is essentially a ***linear*** classifier.

perceptron



step function

$$y = \begin{cases} 0, & \text{if } \mathbf{w}\cdot\mathbf{x}+b \le 0 \\ 1, & \text{if } \mathbf{w}\cdot\mathbf{x}+b > 0 \end{cases}$$

AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

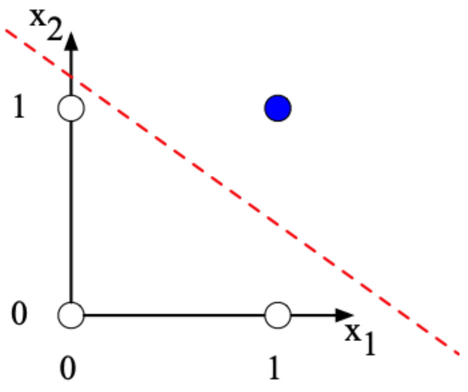
y = a(X1*w1+x2*w2+b)

0*1+0*1-1 =-1 →0

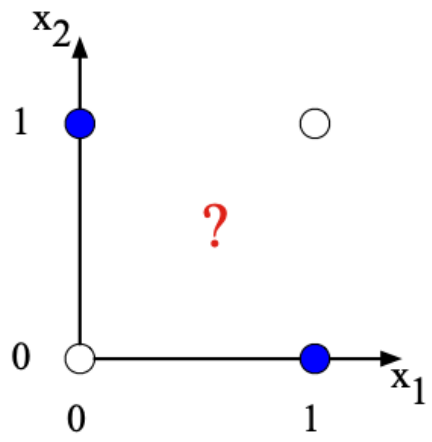
0*0+1*1-1 =0 →0

1*1+0*1-1 =0 →0

1*1+1*1-1 =1 →1



a) x_1 AND x_2



How can multilayer NNs solve the XOR problem?

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2\, W_{21}^{(1)} + b_1^{(1)}$)

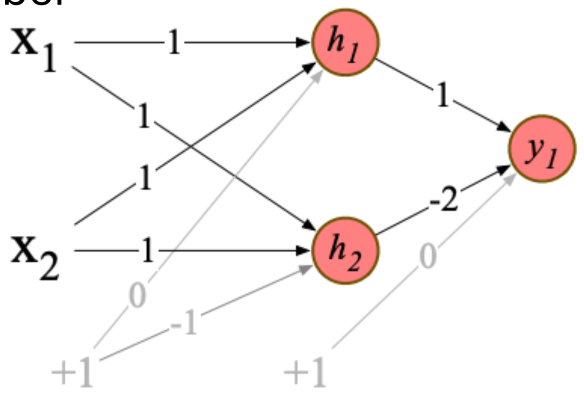
h2 = ReLU($x_1 * W_{12}^{(1)} + x_2\, W_{22}^{(1)} + b_2^{(1)}$)

y = ReLU($h_1 * W_{21}^{(2)} + h_2\, W_{22}^{(2)} + b_1^{(2)}$)

$$W_{ij}^{(l)}$$

The layer number

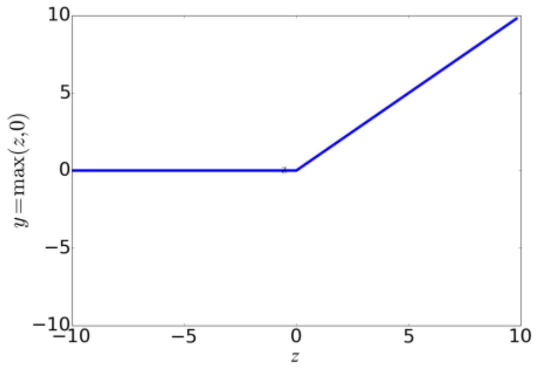
i: the neuron index from the previous layer; j the neuron index of the current layer



NN

Rectified linear unit, also called the ReLU

$$y = \mathrm{ReLU}(z) = max(z, 0)$$



ReLU

How can a multilayer NNs solve the XOR problem?

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2\, W_{21}^{(1)} + b_1^{(1)}$)

h2 = ReLU($x_1 * W_{12}^{(1)} + x_2\, W_{22}^{(1)} + b_2^{(1)}$)

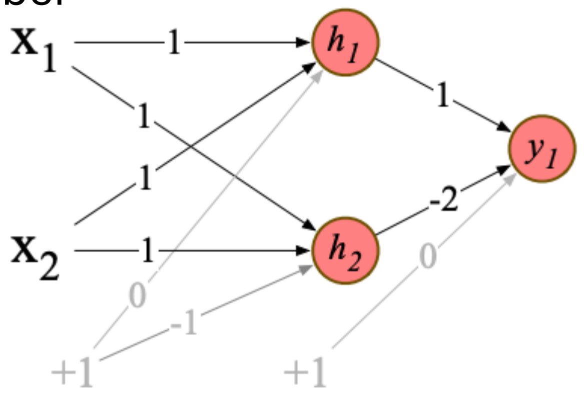
y = ReLU($h_1 * W_{21}^{(2)} + h_2\, W_{22}^{(2)} + b_1^{(2)}$)

h1 = ReLU(0*1+0*1+0)=0
h2 = ReLU(0*1+0*1-1)=0
y = ReLU(0*1+0*(-2)+0) = 0

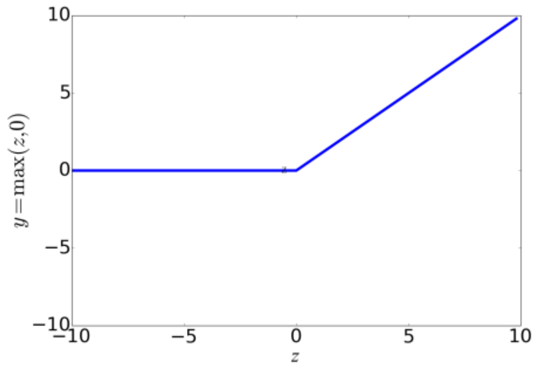
The layer number

$$W_{ij}^{(l)}$$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2 W_{21}^{(1)} + b_1^{(1)}$)

h2 = ReLU($x_1 * W_{12}^{(1)} + x_2 W_{22}^{(1)} + b_2^{(1)}$)

y = ReLU($h_1 * W_{21}^{(2)} + h_2 W_{22}^{(2)} + b_1^{(2)}$)

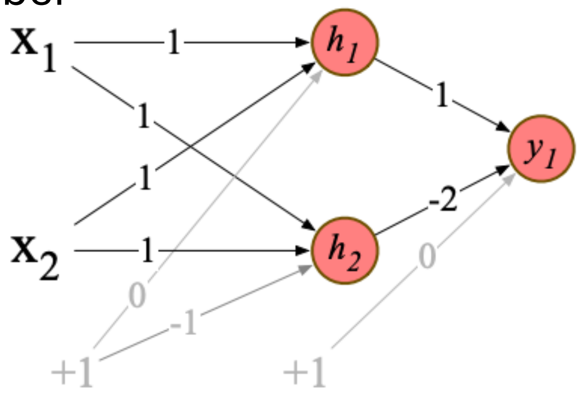
y = ReLU(0*1+0*(-2)+0) = 0

h1 = ReLU(0*1+1*1+0)=1
h2 = ReLU(0*1+1*1-1)=0
y = ReLU(1*1+0*(-2)+0) = 1

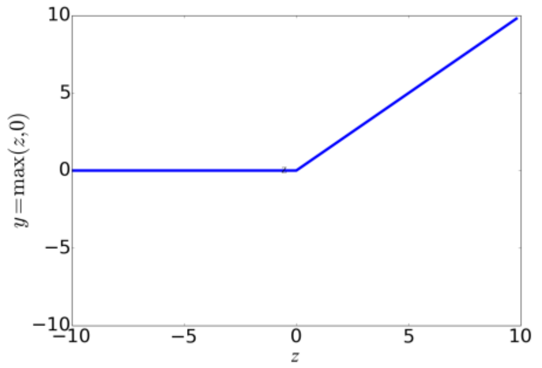
The layer number

$$W_{ij}^{(l)}$$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

$h1 = ReLU(x_1 * W_{11}^{(1)} + x_2\, W_{21}^{(1)} + b_1^{(1)})$

$h2 = ReLU(x_1 * W_{12}^{(1)} + x_2\, W_{22}^{(1)} + b_2^{(1)})$

$y = ReLU(h_1 * W_{21}^{(2)} + h_2\, W_{22}^{(2)} + b_1^{(2)})$

y = ReLU(0*1+0*(-2)+0) = 0

h1 = ReLU(0*1+1*1+0)=1
h2 = ReLU(0*1+1*1-1)=0
y = ReLU(1*1+0*(-2)+0) = 1

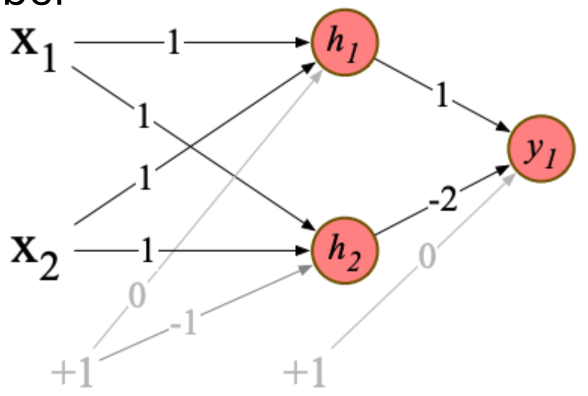
XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

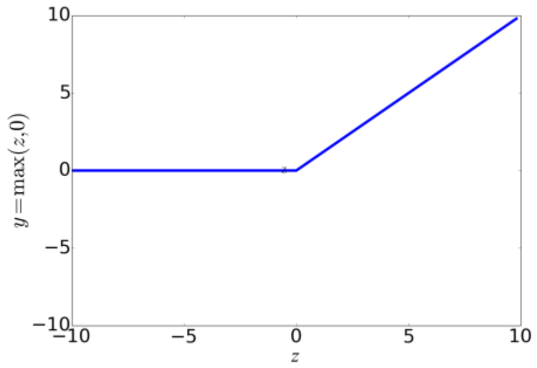
The layer number

$$W_{ij}^{(l)}$$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2 W_{21}^{(1)} + b_1^{(1)}$)

h2 = ReLU($x_1 * W_{12}^{(1)} + x_2 W_{22}^{(1)} + b_2^{(1)}$)

y = ReLU($h_1 * W_{21}^{(2)} + h_2 W_{22}^{(2)} + b_1^{(2)}$)

y = ReLU(0*1+0*(-2)+0) = 0

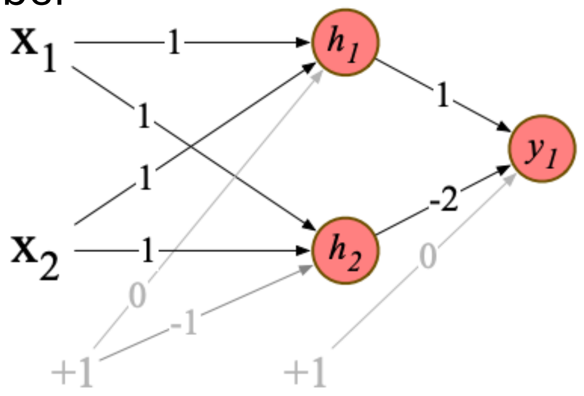
y = ReLU(1*1+0*(-2)+0) = 1

h1 = ReLU(1*1+0*1+0)=1
h2 = ReLU(0*1+1*1-1)=0
y = ReLU(1*1+0*(-2)+0) = 1

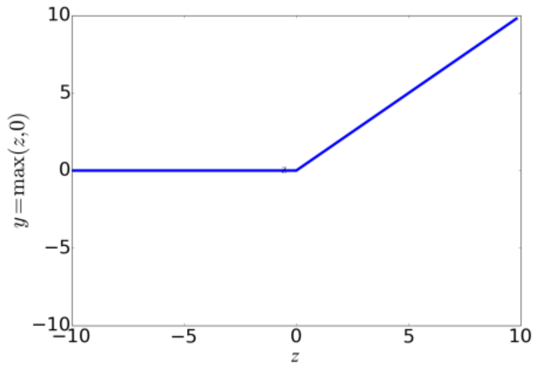
The layer number

$W_{ij}^{(l)}$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2\, W_{21}^{(1)} + b_1^{(1)}$)

h2 = ReLU($x_1 * W_{12}^{(1)} + x_2\, W_{22}^{(1)} + b_2^{(1)}$)

y = ReLU($h_1 * W_{21}^{(2)} + h_2\, W_{22}^{(2)} + b_1^{(2)}$)

y = ReLU(0*1+0*(-2)+0) = 0

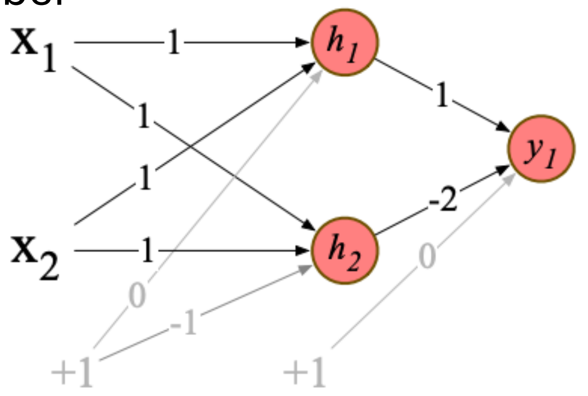
y = ReLU(1*1+0*(-2)+0) = 1

y = ReLU(1*1+0*(-2)+0) = 1

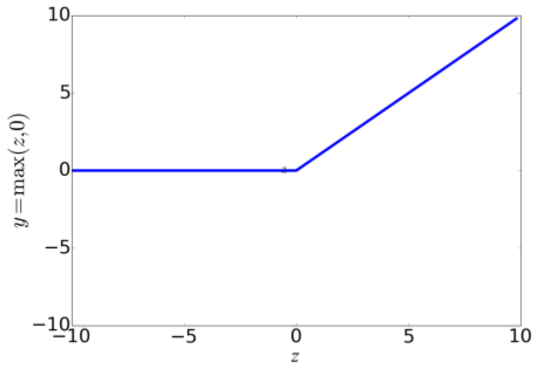
The layer number

$W_{ij}^{(l)}$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

h1 = ReLU($x_1 * W_{11}^{(1)} + x_2\, W_{21}^{(1)} + b_1^{(1)}$)

h2 = ReLU($x_1 * W_{12}^{(1)} + x_2\, W_{22}^{(1)} + b_2^{(1)}$)

y = ReLU($h_1 * W_{21}^{(2)} + h_2\, W_{22}^{(2)} + b_1^{(2)}$)

y = ReLU(0*1+0*(-2)+0) = 0

y = ReLU(1*1+0*(-2)+0) = 1

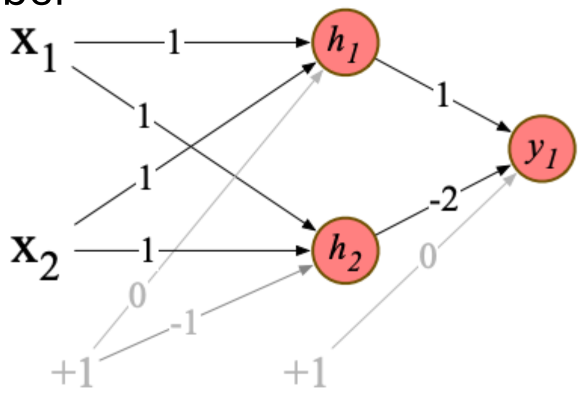
y = ReLU(1*1+0*(-2)+0) = 1

h1 = ReLU(1*1+1*1+0)=2
h2 = ReLU(1*1+1*1-1)=1
y = ReLU(2*1+1*(-2)+0) = 0

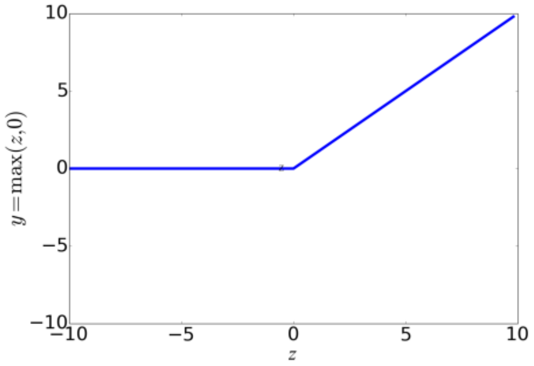
The layer number

$$W_{ij}^{(l)}$$

i: the neuron index from the previous layer; j the neuron index of the current layer



NN



ReLU

How can a multilayer NNs solve the XOR problem?

$h1 = \text{ReLU}(x_1 * W_{11}^{(1)} + x_2 W_{21}^{(1)} + b_1^{(1)})$

$h2 = \text{ReLU}(x_1 * W_{12}^{(1)} + x_2 W_{22}^{(1)} + b_2^{(1)})$

$y = \text{ReLU}(h_1 * W_{21}^{(2)} + h_2 W_{22}^{(2)} + b_1^{(2)})$

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

y = ReLU(0*1+0*(-2)+0) = 0

y = ReLU(1*1+0*(-2)+0) = 1

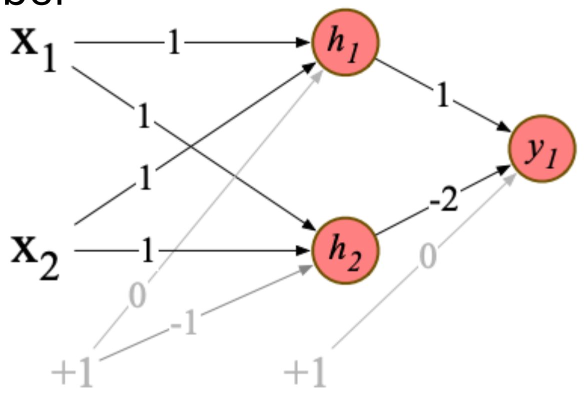
y = ReLU(1*1+0*(-2)+0) = 1

y = ReLU(2*1+1*(-2)+0) = 0

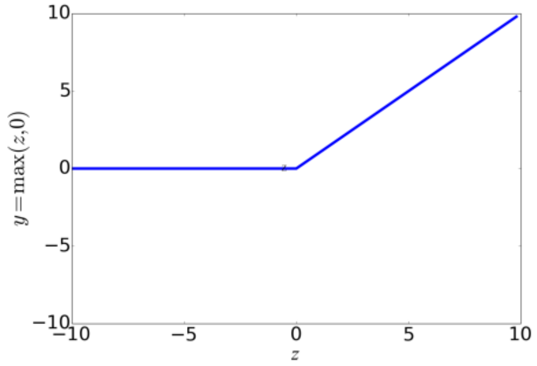
The layer number

$$W_{ij}^{(l)}$$

i: the neuron index from the previous layer; j the neuron index of the current layer

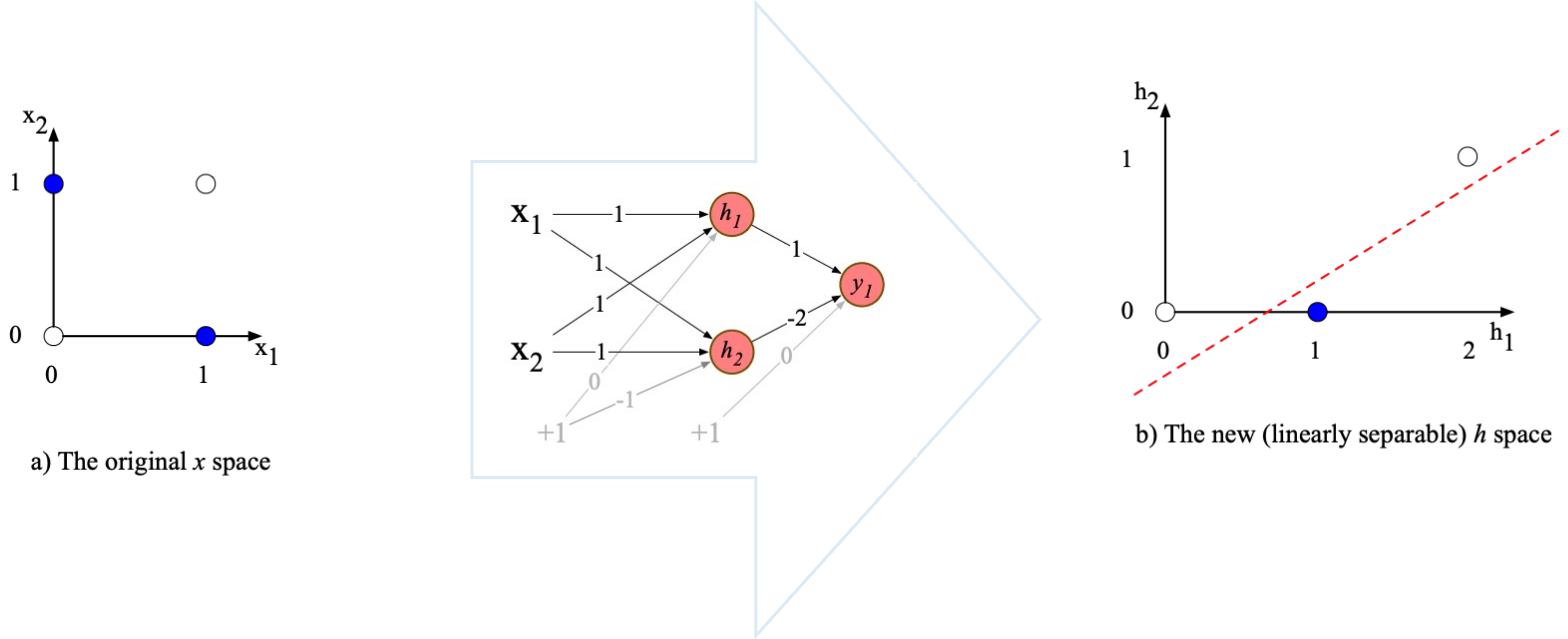


NN



ReLU

Why can a multilayer NNs solve the XOR problem?



- Code https://colab.research.google.com/drive/1Dkr6GSFfD6klrdpkH5bw0YeO5H9FzeAT?usp=sharing