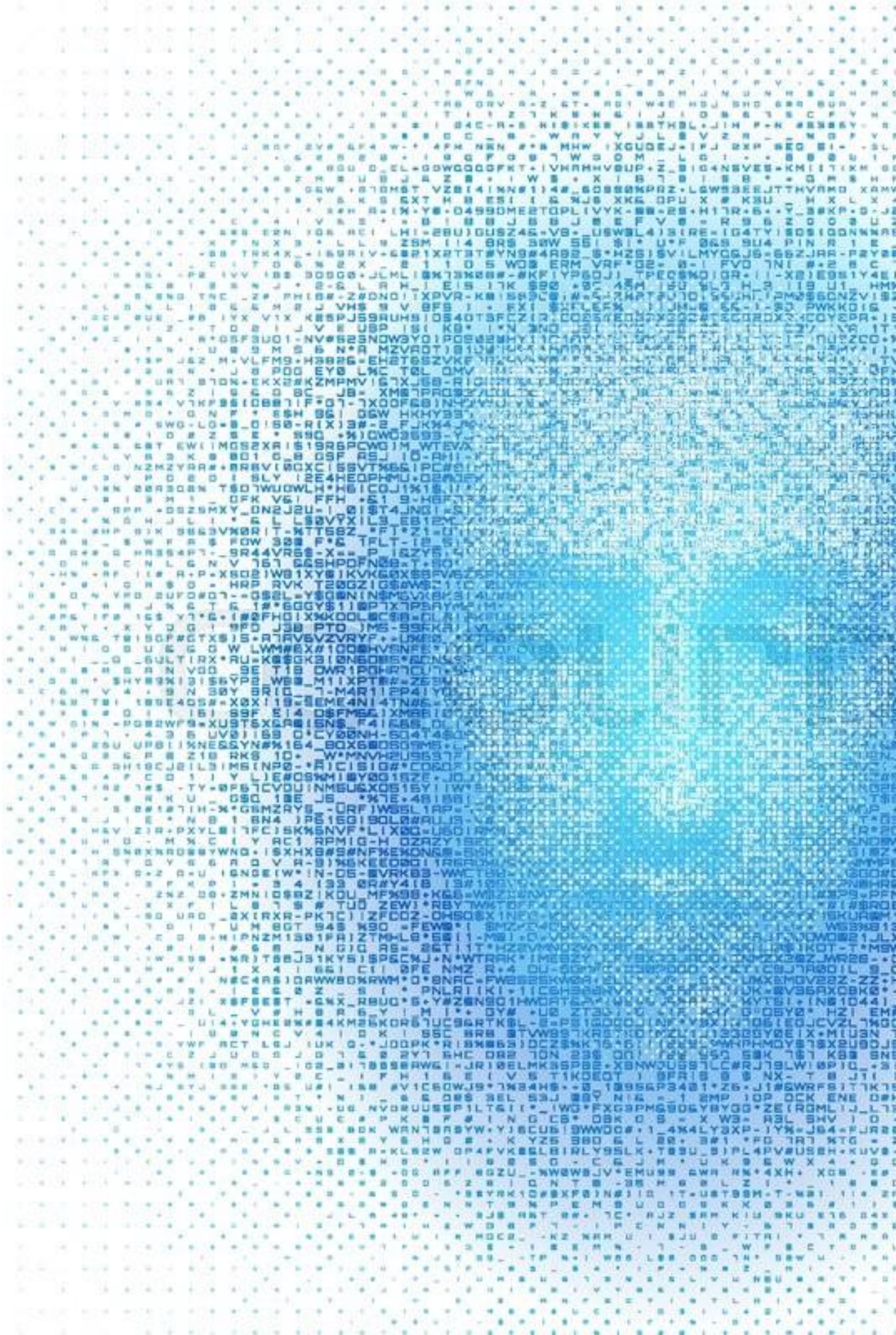
Introduction to Artificial Intelligence

KIAS, Computational Sciences
Research Fellow, Jeonggyu Huh



- CMAC (Yonsei University Center for Mathematical Analysis & Computation)
  Sep 2017 – Aug 2018

  Postdoctoral Researcher
  on Deep Learning to Finance

- KIAS (Korea Institute For Advanced Science)
  Sep 2018 – present

  Research Fellow
  on Deep Learning and Quantitative Finance

These photos were when I participated in 2019 Fuzhou World Go AI Competition. My team won the second prize.

Beginning of Artificial Intelligence (AI)

1936 - Alan Turing invented "Turing machine", which is known as one of fundamental theorems for computational science.

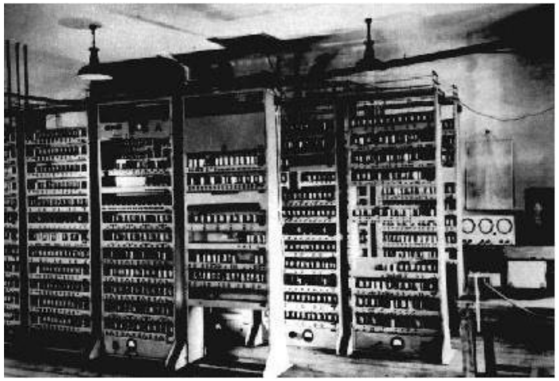
1946 - The first general-purpose computer, ENIAC, was developed in USA. It was the first machine to solve "a large class of problems" through reprogramming.

1949 - Besides, EDSAC was an early British computer inspired by von Neumann's draft. So the machine could store program instructions in electronic memory for the first time.

1950s - Many intellectuals began to think that human's brain is just like a electronic chip. It is generally considered that AI study started at this era.

Alan Turing

John von Neumann



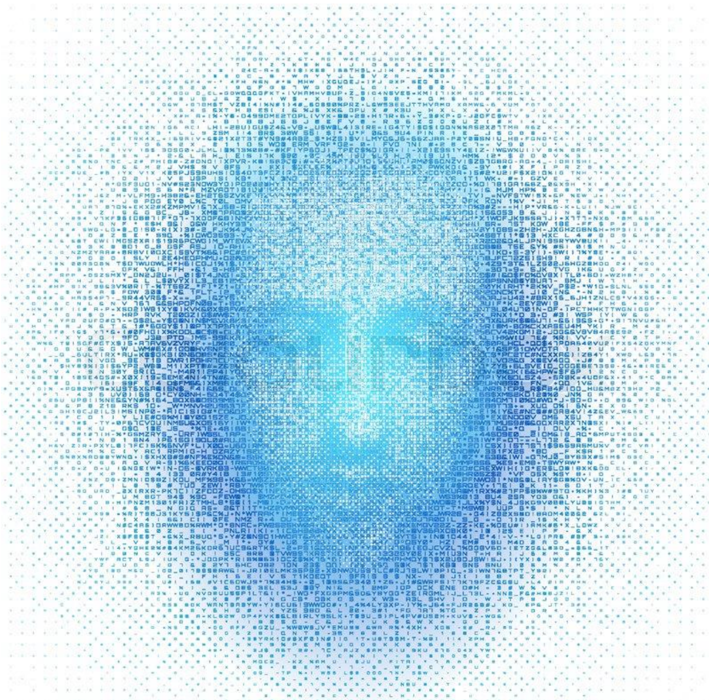
EDSAC

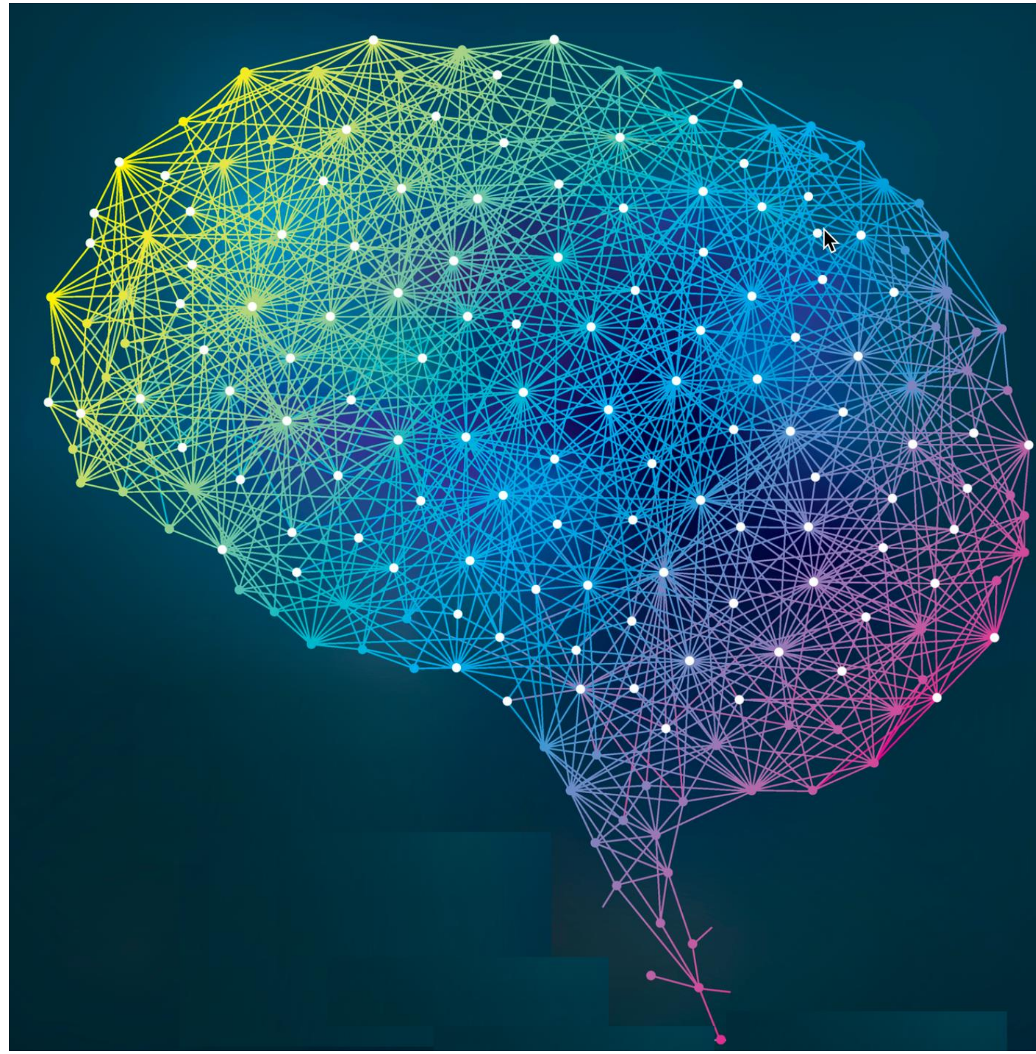
John McCarthy　Marvin Minsky　Ray Solomonoff

Dartmouth workshop, 1956

The word "artificial intelligence" was officially used for the first time.

Human's
Brain

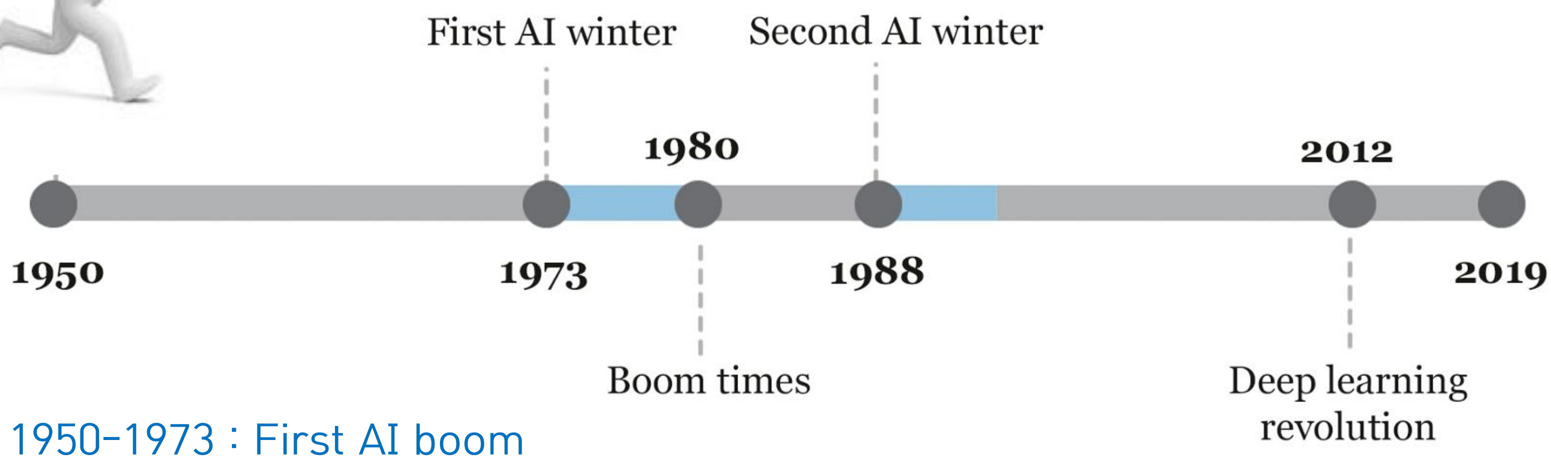




Can we imitate the brain artificially?

A Human brain consists of about 1000 billion of neurons.
The number of synapse, a connection between two neurons, is about 1000 trillion.

First AI Boom (1950s and 60s)



1950-1973 : First AI boom
1973-1980 : First AI winter

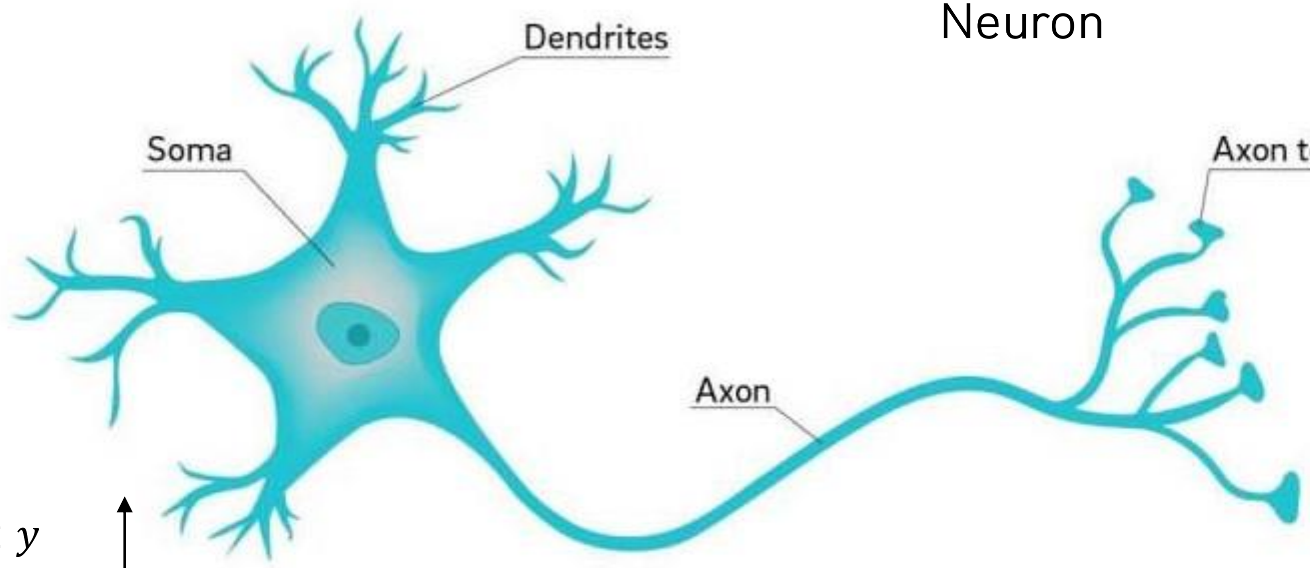
1980-1988 : Second AI boom
1988-2000 : Second AI winter

2000-2012 : Many researchers tried to revive the field of AI.

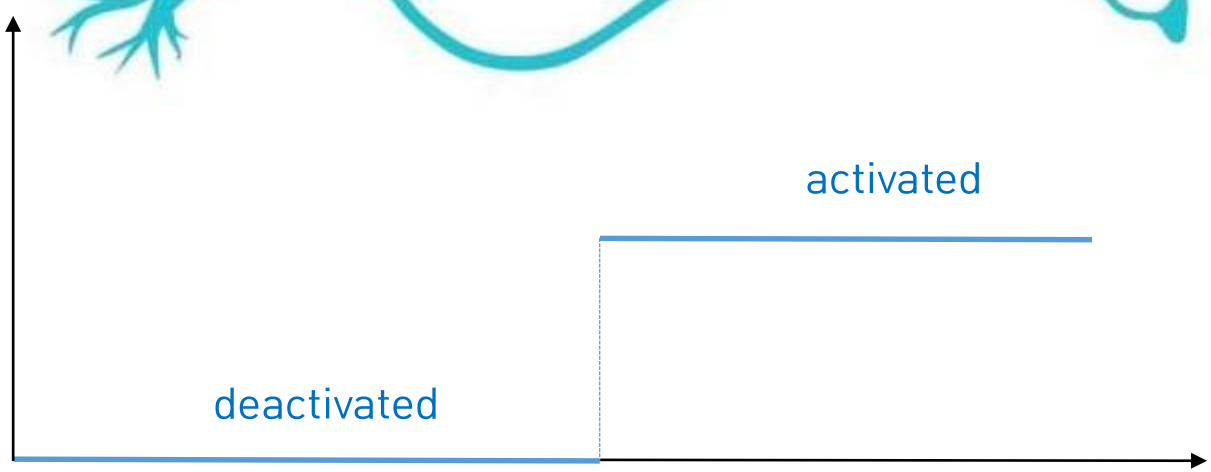
2012- : Third AI boom! Are another AI winter coming again? God knows...

input

Neuron

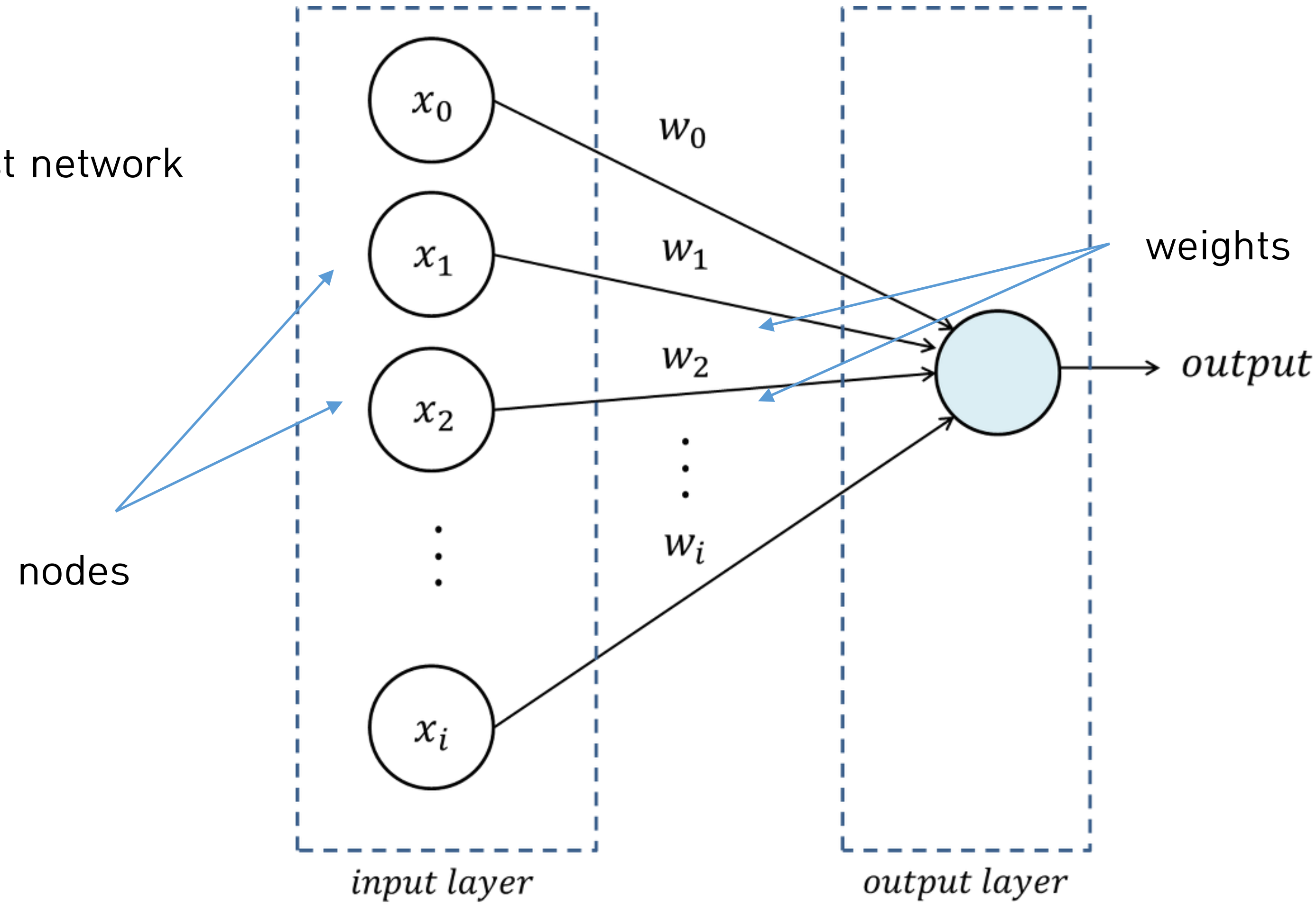


output



perceptron

: the simplest network



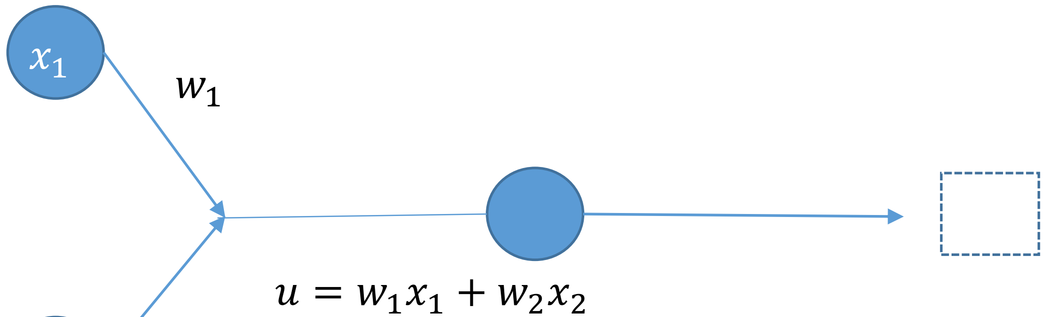
input signal

$$u = w_0x_0 + w_1x_1 + \cdots + w_ix_i$$

output signal

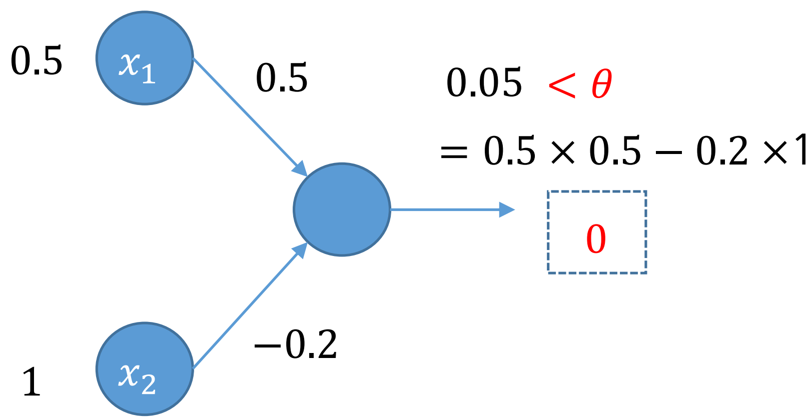
$$y = \begin{cases} 1 \text{ if } h(u) \geq \theta \\ 0 \text{ if } h(u) < \theta \end{cases}$$

h: activation function
θ: threshold

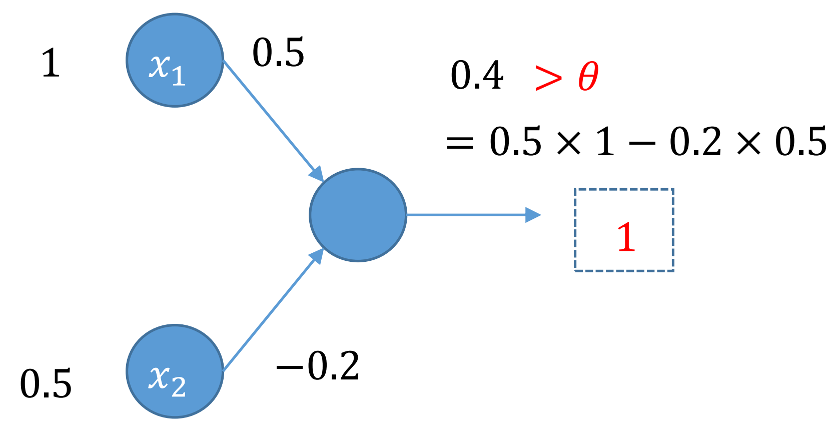


$\theta = 0.3$

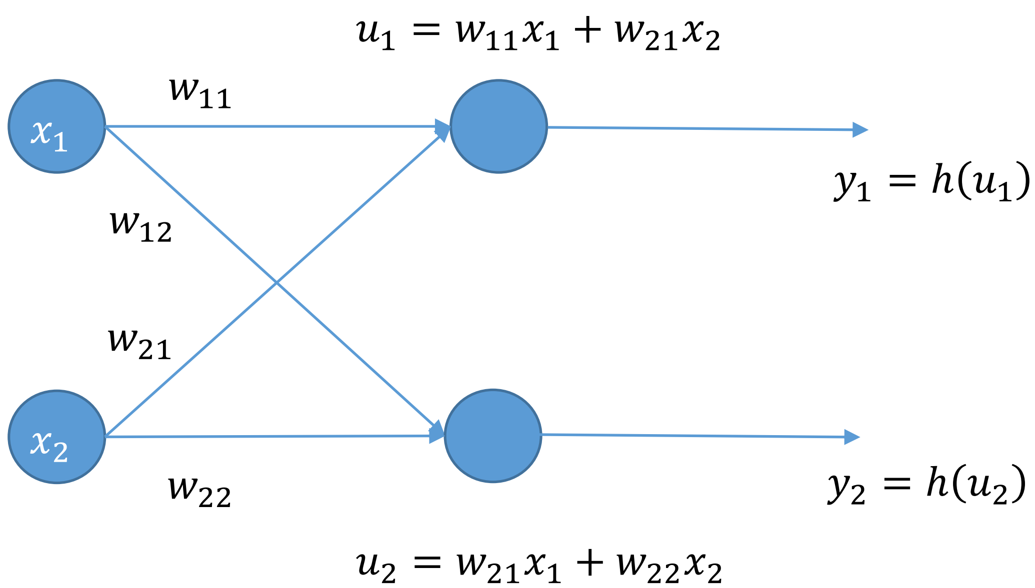
$$y = \begin{cases} 1 \text{ if } u \geq \theta \\ 0 \text{ if } u < \theta \end{cases}$$



deactivated

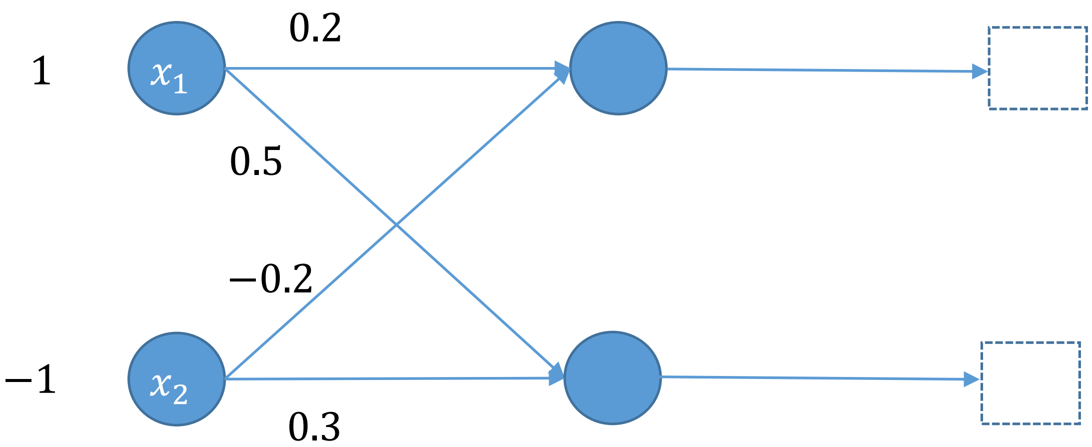


activated



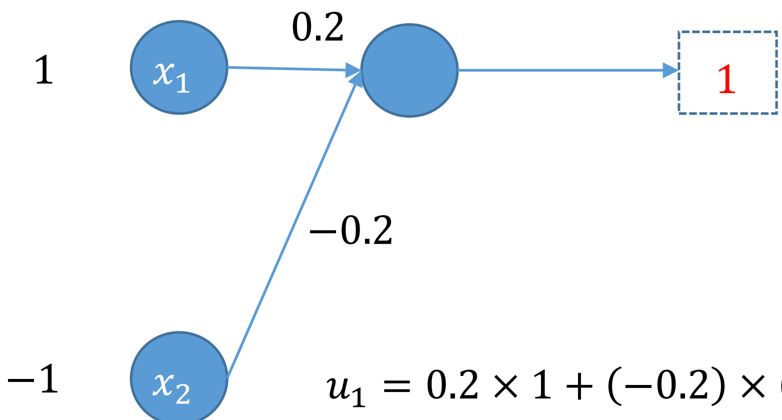
$$h(u) = \begin{cases} 1 \text{ if } u \geq \theta \\ 0 \text{ if } u < \theta \end{cases}$$

Of course, you can set the number of output neurons as you wish.

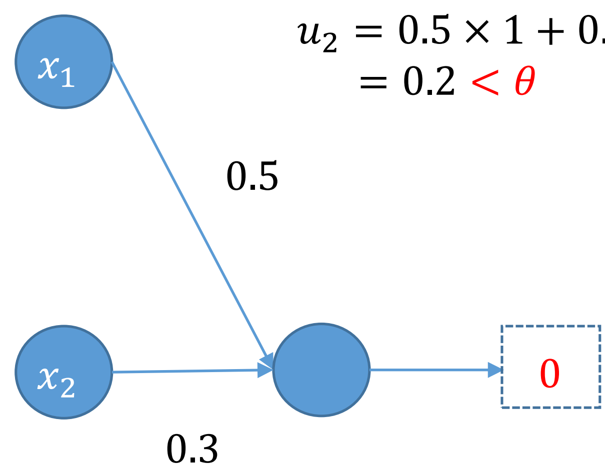


$\theta = 0.3$

1

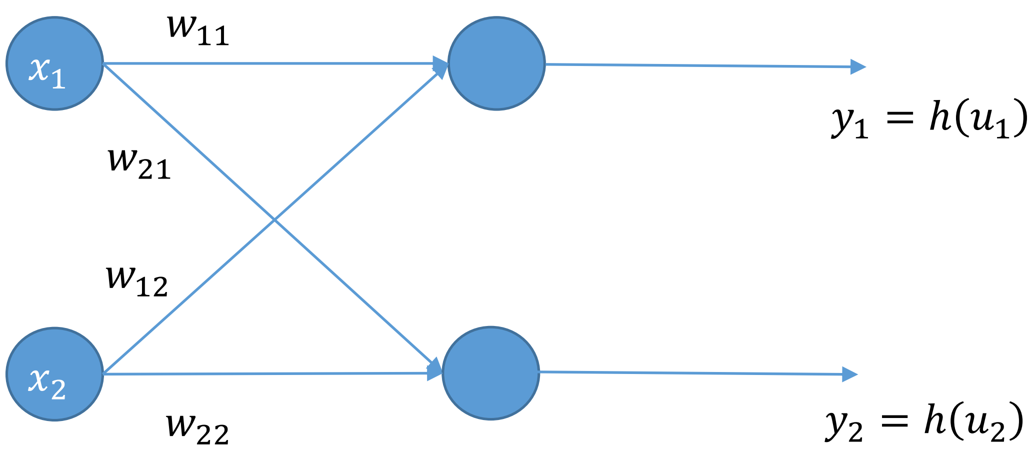


$$u_1 = 0.2 \times 1 + (-0.2) \times (-1)$$
$$= 0.4 > \theta$$



$$u_2 = 0.5 \times 1 + 0.3 \times (-1)$$
$$= 0.2 < \theta$$

$$u_1 = w_{11}x_1 + w_{12}x_2$$



$$u_2 = w_{21}x_1 + w_{22}x_2$$

$$h(u) = \begin{cases} 1 \text{ if } u \geq \theta \\ 0 \text{ if } u < \theta \end{cases}$$

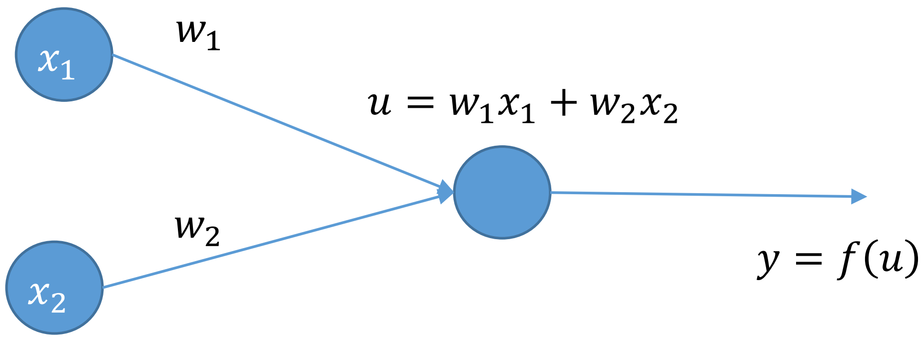
$$u_1 = w_{11}x_1 + w_{12}x_2 \qquad y_1 = h(u_1)$$

$$u_2 = w_{21}x_1 + w_{22}x_2 \qquad y_2 = h(u_2)$$

$$\vec{u} = W\vec{x} \qquad \vec{y} = h(\vec{u})$$

It is much more convenient to use a matrix vector notation.

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

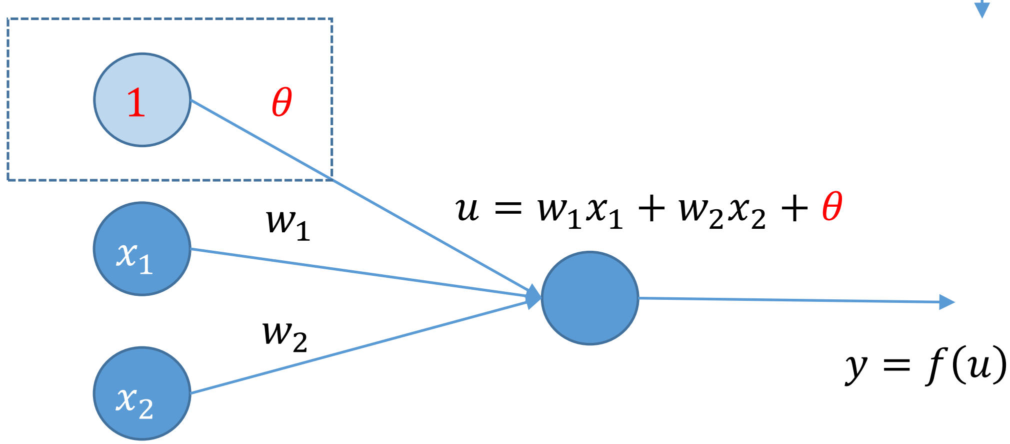


$$f(u) = \begin{cases} 1 \text{ if } u \geq \theta \\ 0 \text{ if } u < \theta \end{cases}$$

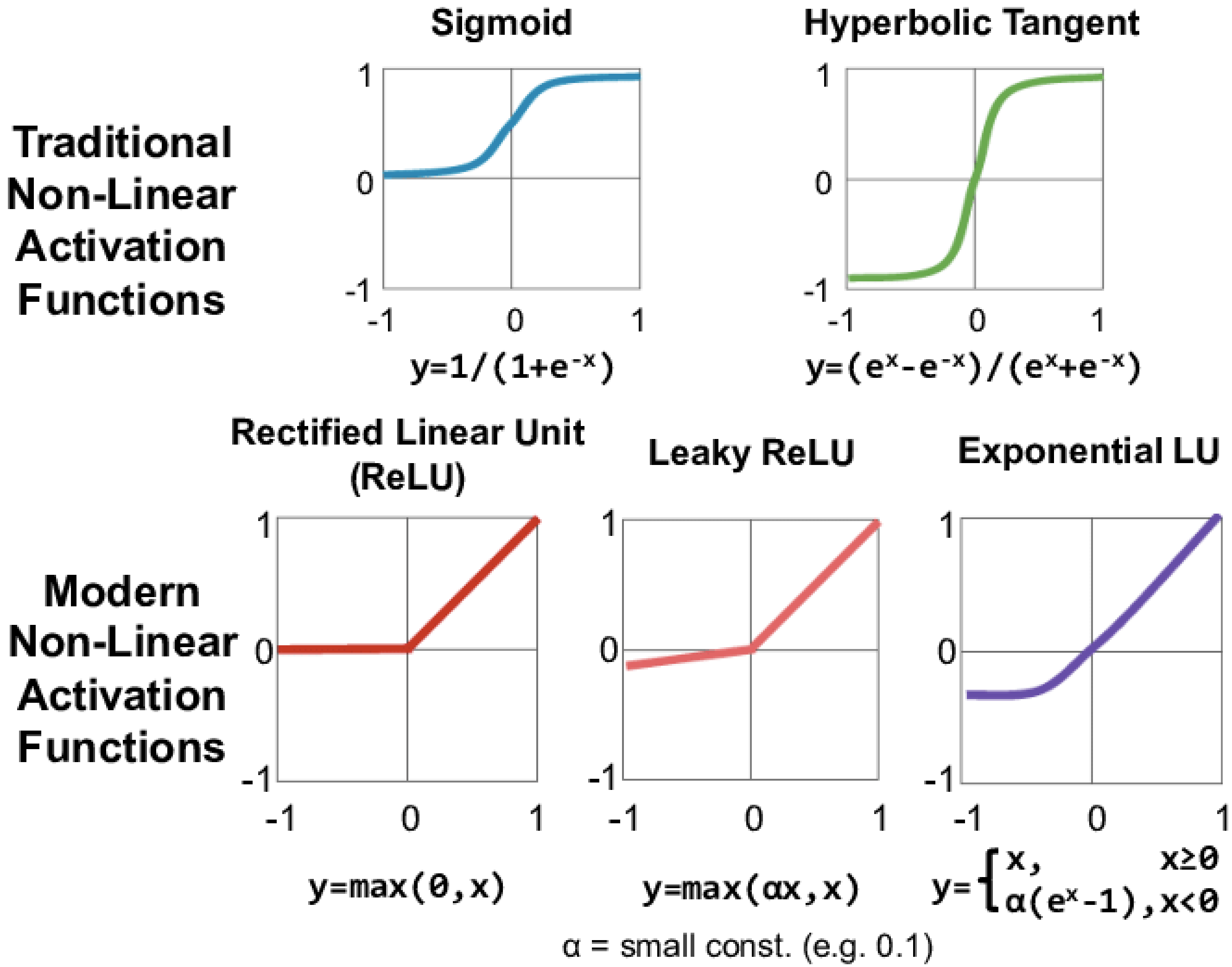
totally equivalent

bias node

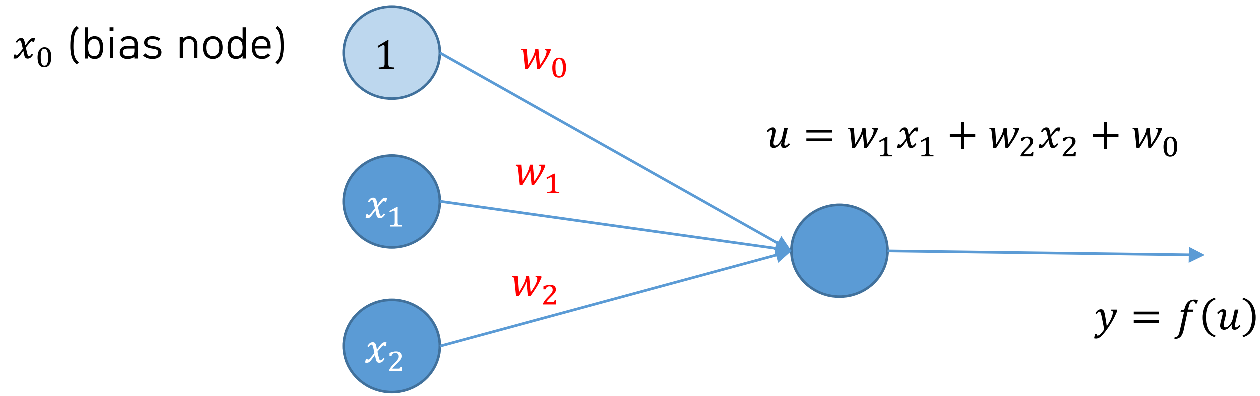
: it is supposed that it has always a virtual input 1

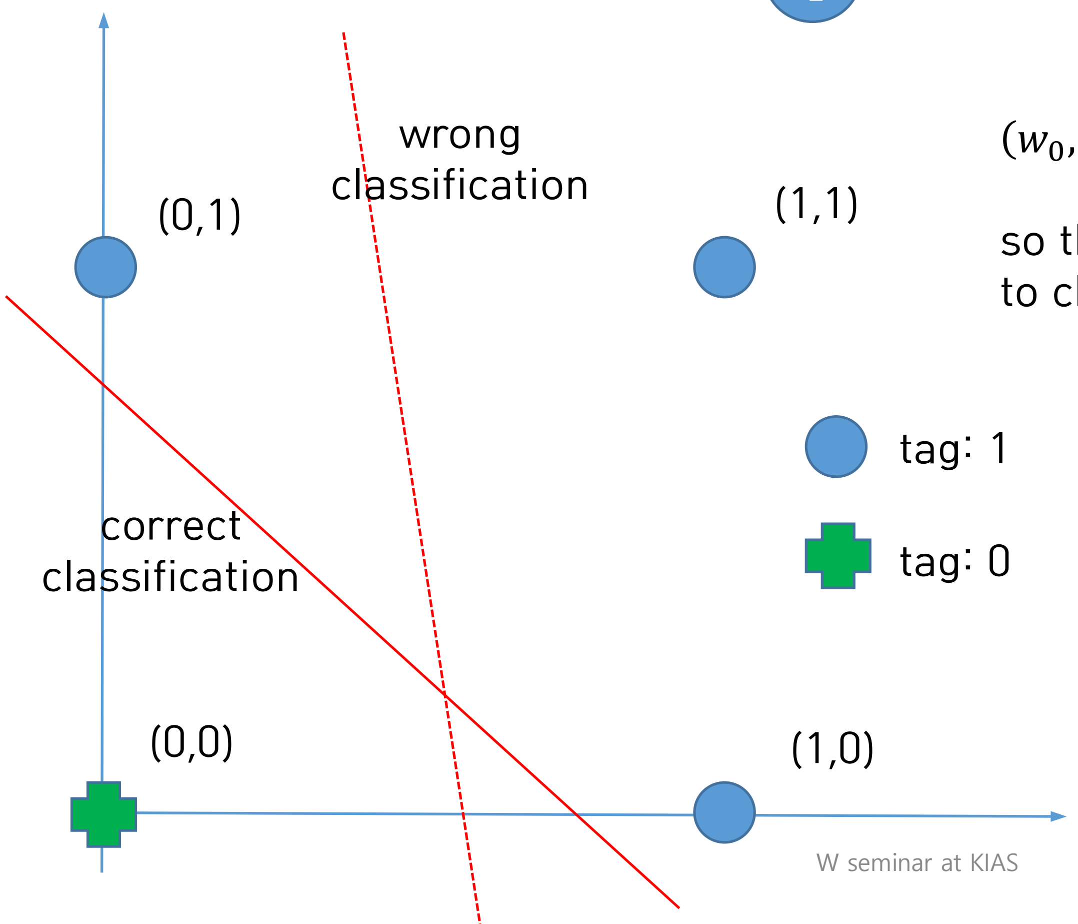


$$f(u) = \begin{cases} 1 \text{ if } u \geq 0 \\ 0 \text{ if } u < 0 \end{cases}$$



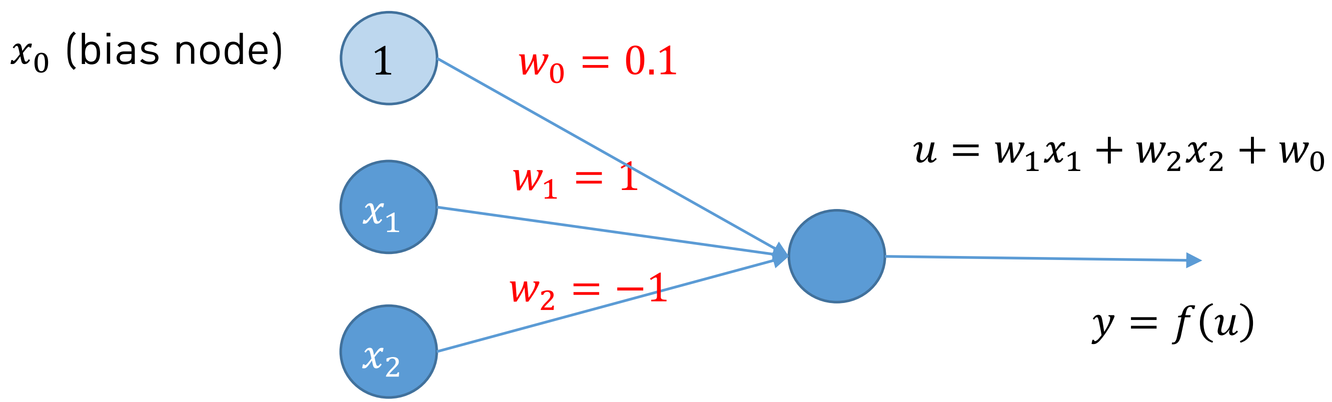
One does not need to use the unit step function only as your activation function.
Besides, there are many kinds of other activation functions such as sigmoid, tanh.
But the functions should be nonlinear and have known analytic forms for their derivatives.





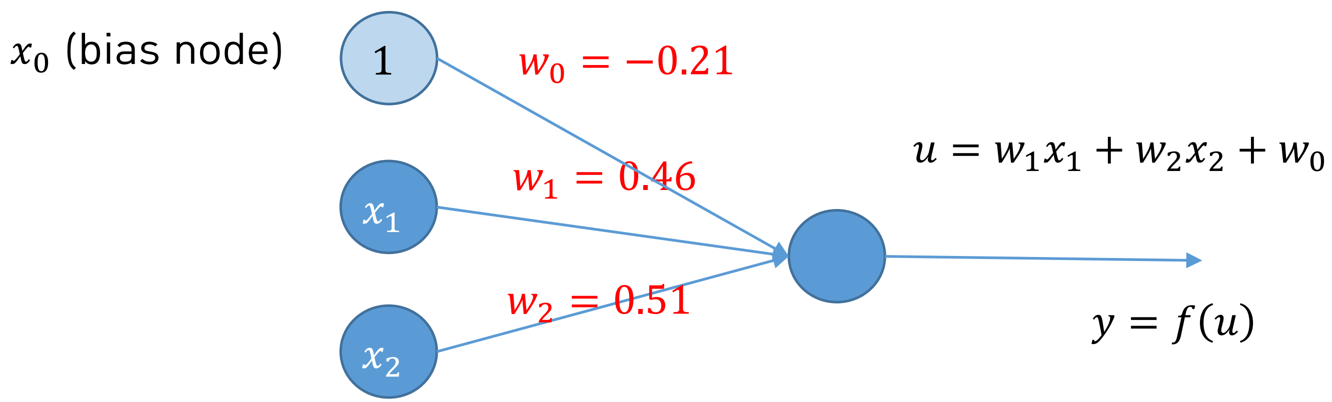
(w_0, w_1, w_2)?

so that the above perceptron
to classify the data on the left correctly



input (x_1, x_2)	u	y	answer	
(0,0)	0.1	1	0	disagree
(0,1)	-0.9	0	1	disagree
(1,0)	1.1	1	1	agree
(1,1)	0.1	1	1	agree

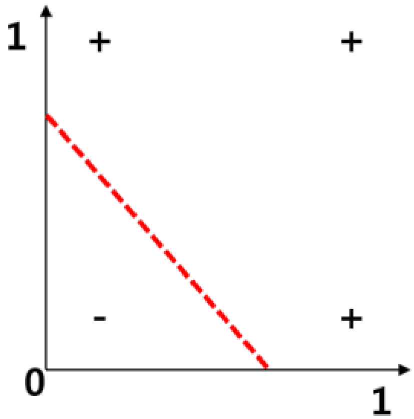
incorrect



input (x_1, x_2)	u	y	answer		
(0,0)	-0.21	0	0		agree
(0,1)	0.3	1	1		agree
(1,0)	0.25	1	1		agree
(1,1)	0.76	1	1		agree

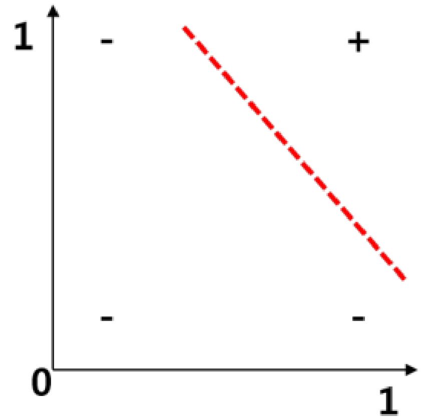
<u>correct</u>

OR



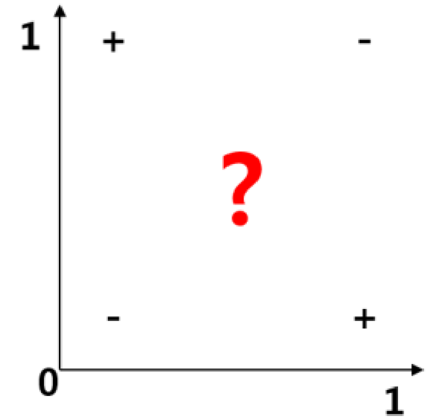
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

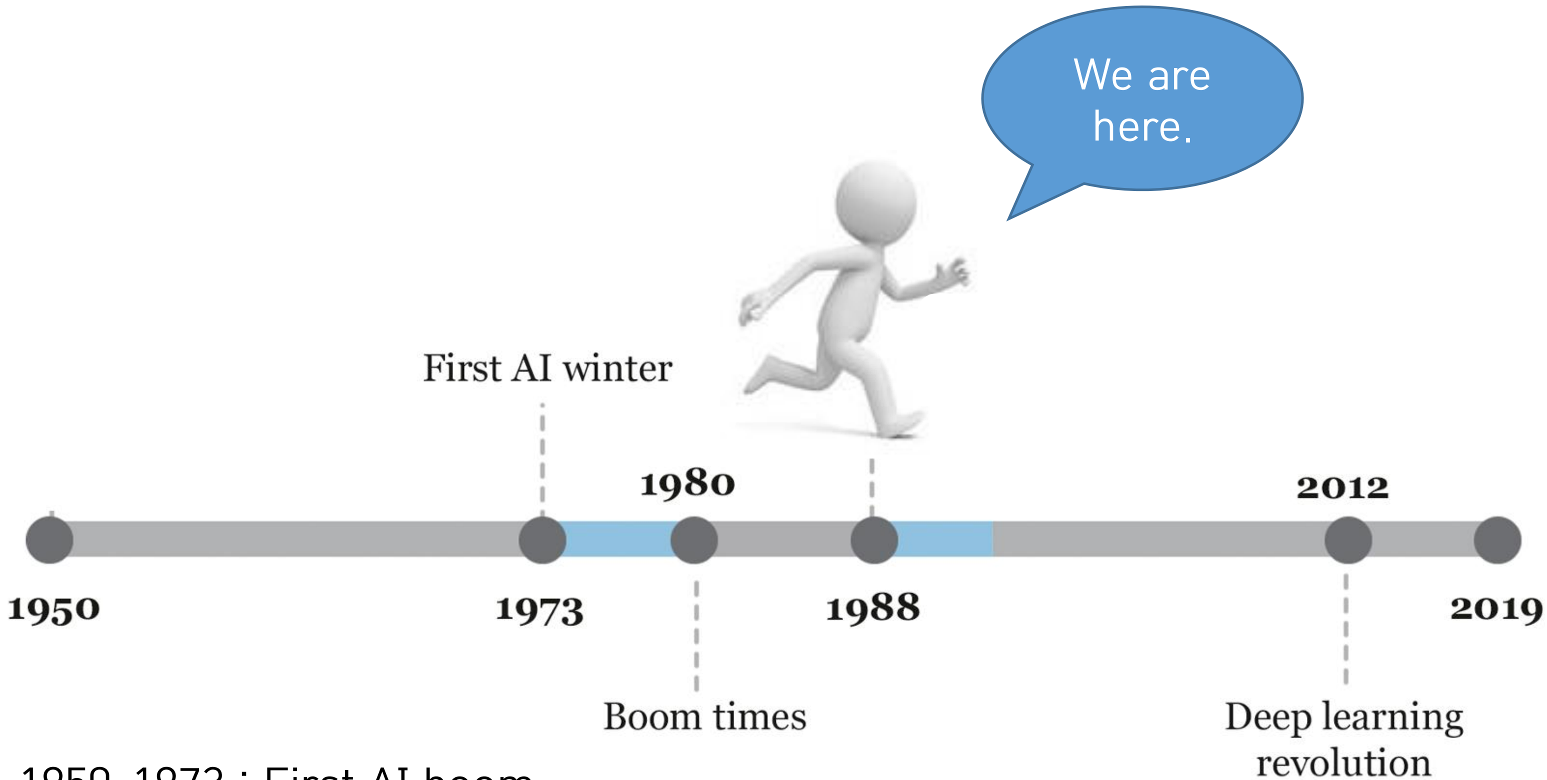
XOR



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

A perceptron is, however, unable to classify even the XOR problem.
And then, of course, there are a high probability that the network can not solve much more complex problems such as language translation.

Second AI Boom (1980s)



1950-1973 : First AI boom
1973-1980 : First AI winter

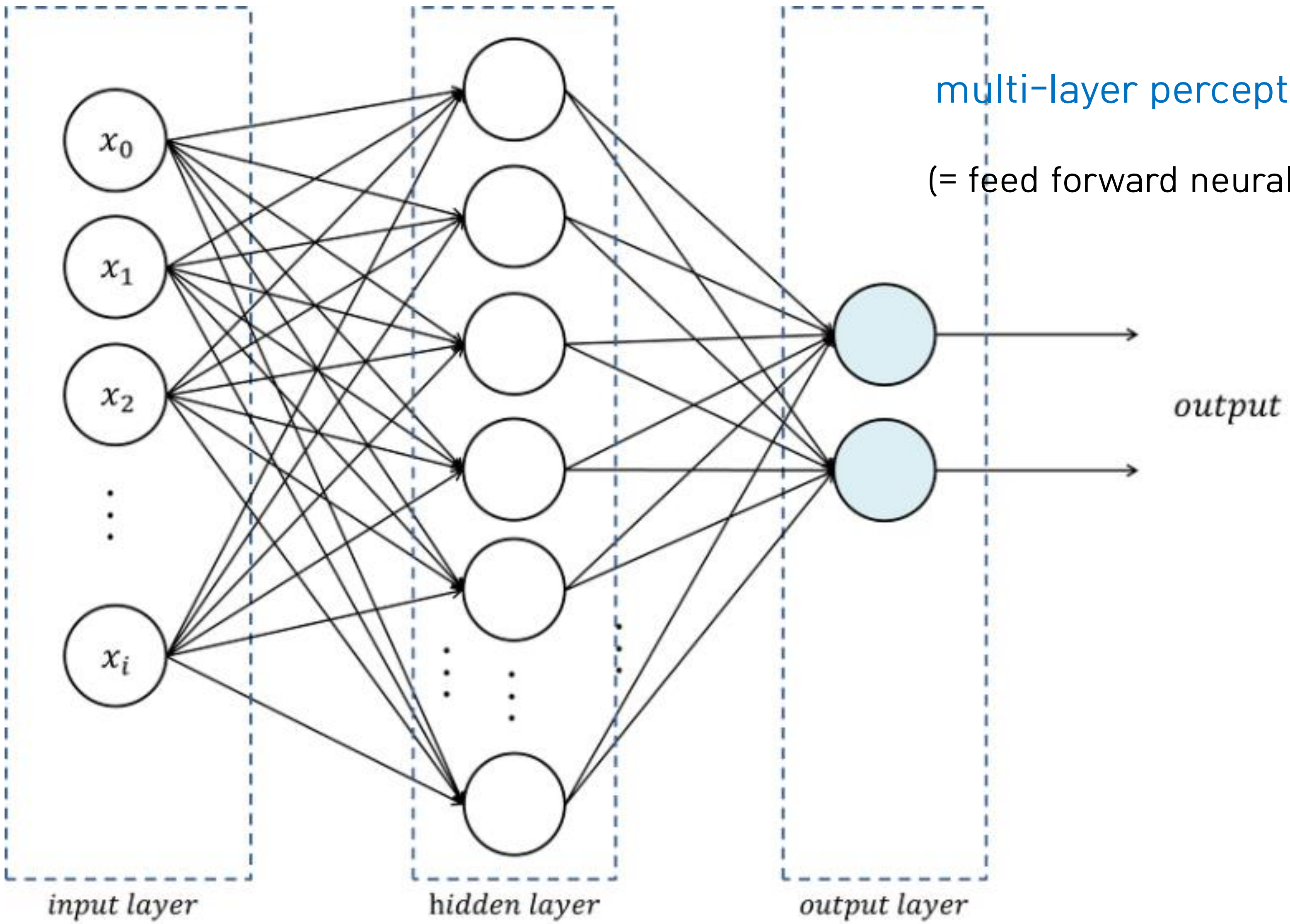
1980-1988 : Second AI boom
1988-2000 : Second AI winter

2000-2012 : Many researchers tried to revive the field of AI.

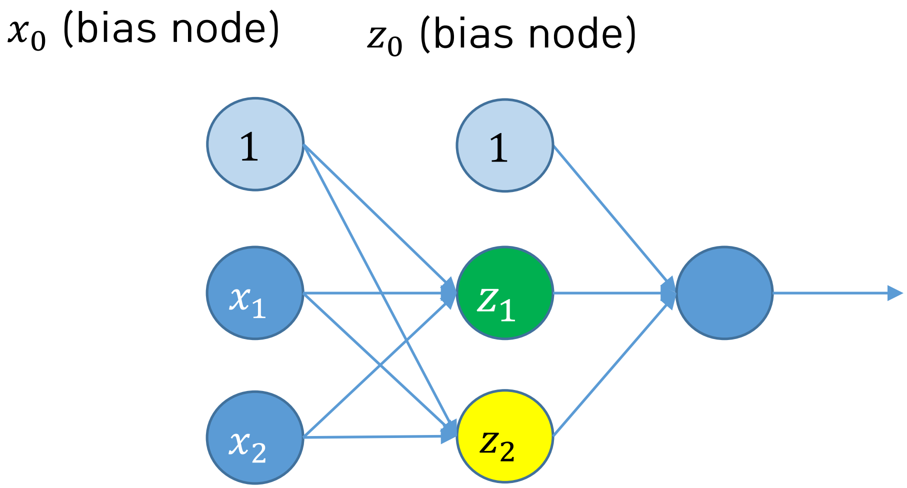
2012- : Third AI boom! Are another AI winter coming again? God knows...

multi-layer perceptron

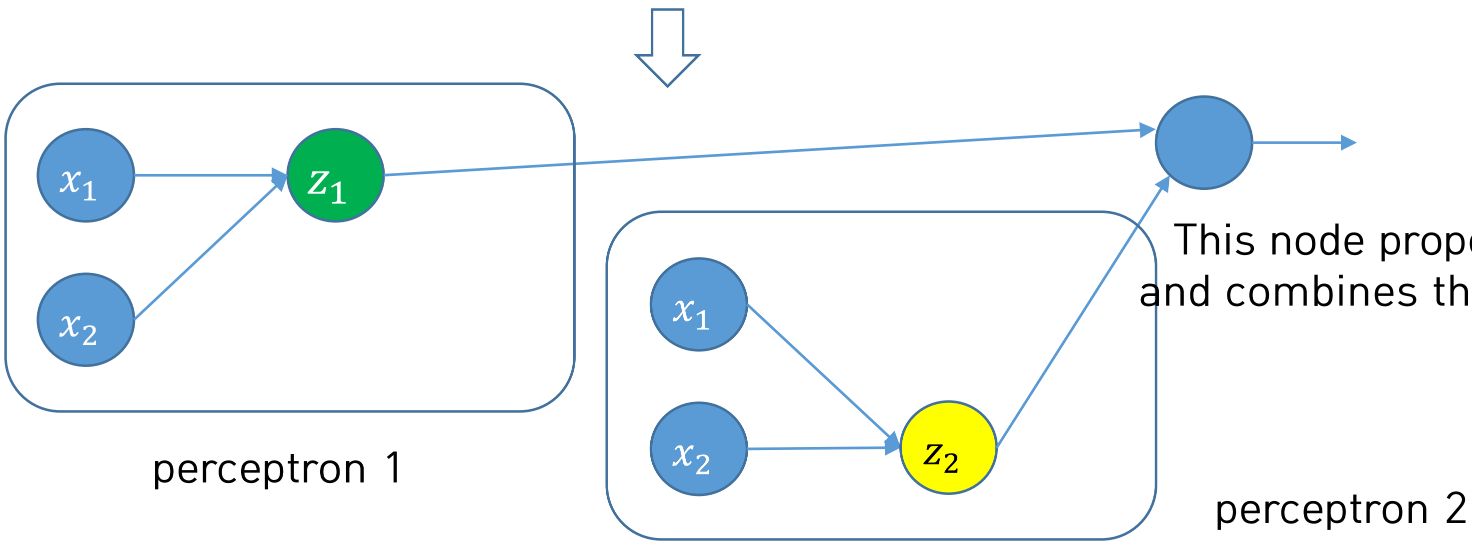
(= feed forward neural network)



Multi-layer perceptron (MLP) can have several hidden layers

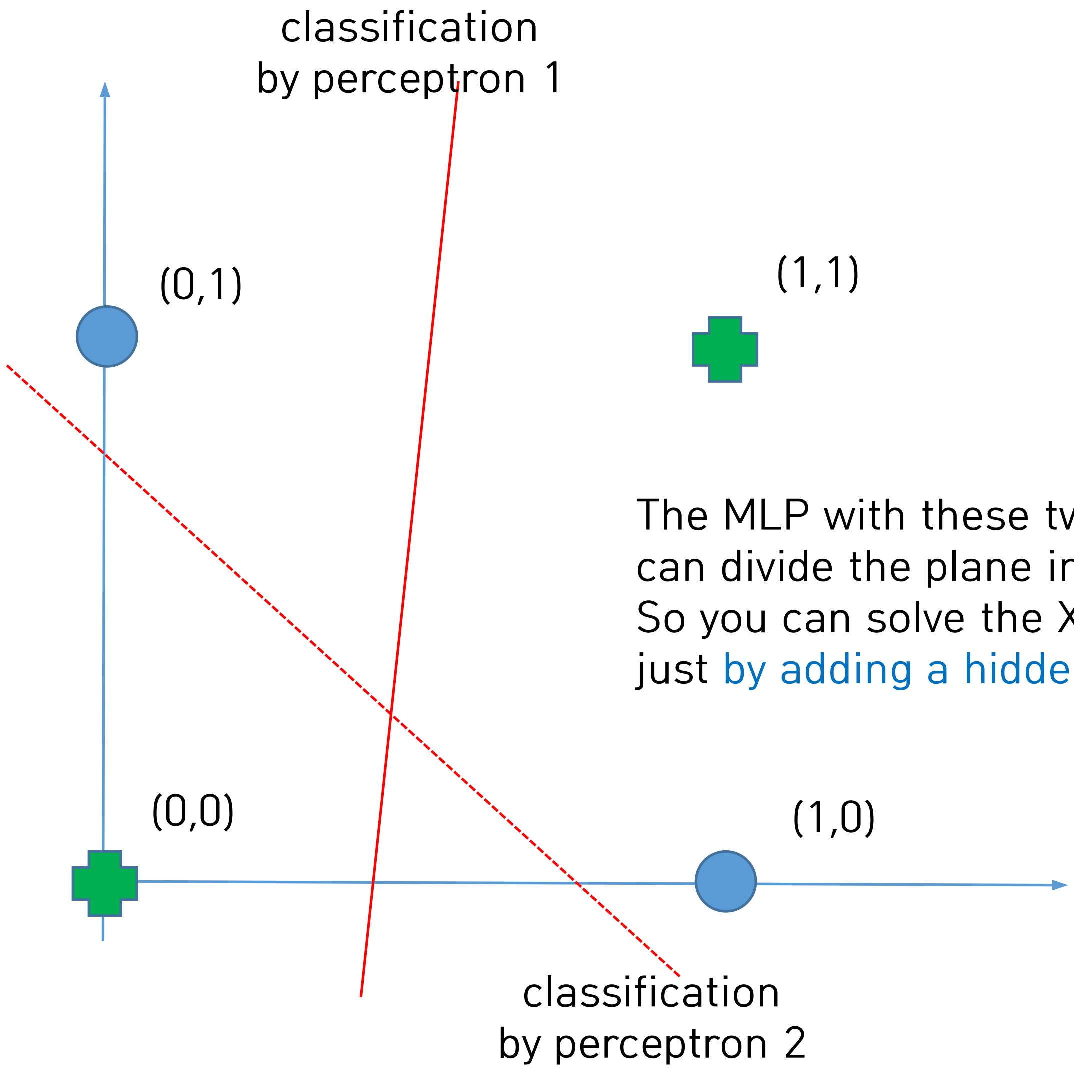


We can solve
the XOR problem with a MLP.



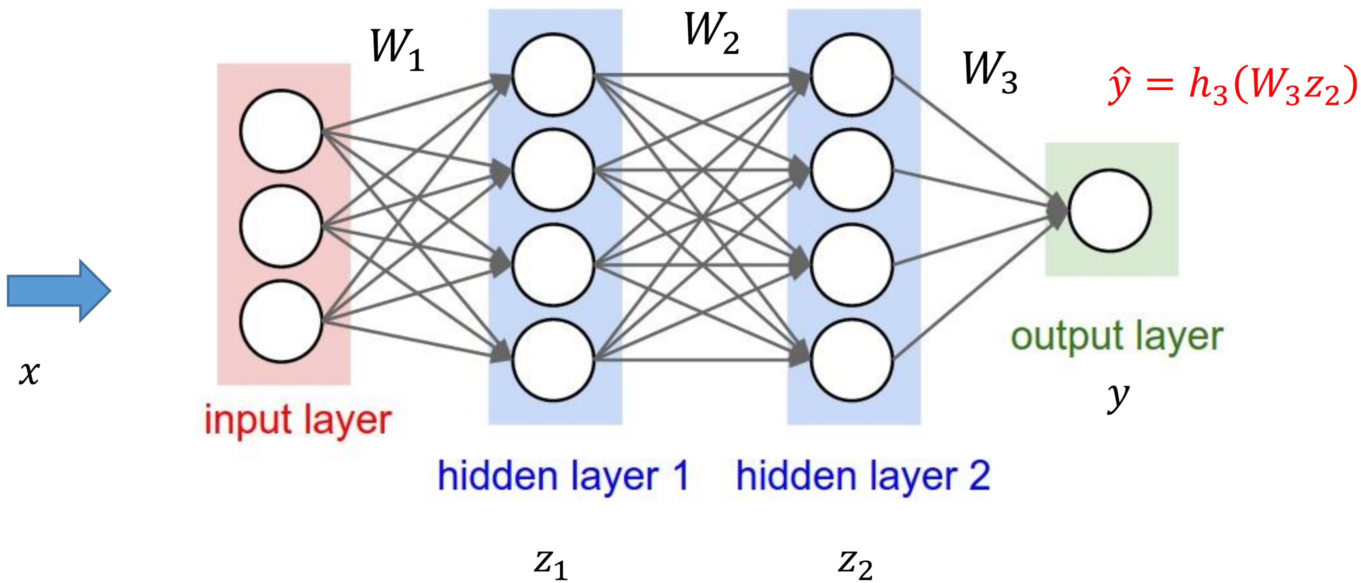
This node properly gathers
and combines the two results.

A MLP can be regarded as an ensemble of several perceptrons.



The MLP with these two perceptrons
can divide the plane into 4 pieces.
So you can solve the XOR problem
just by adding a hidden layer with two nodes.

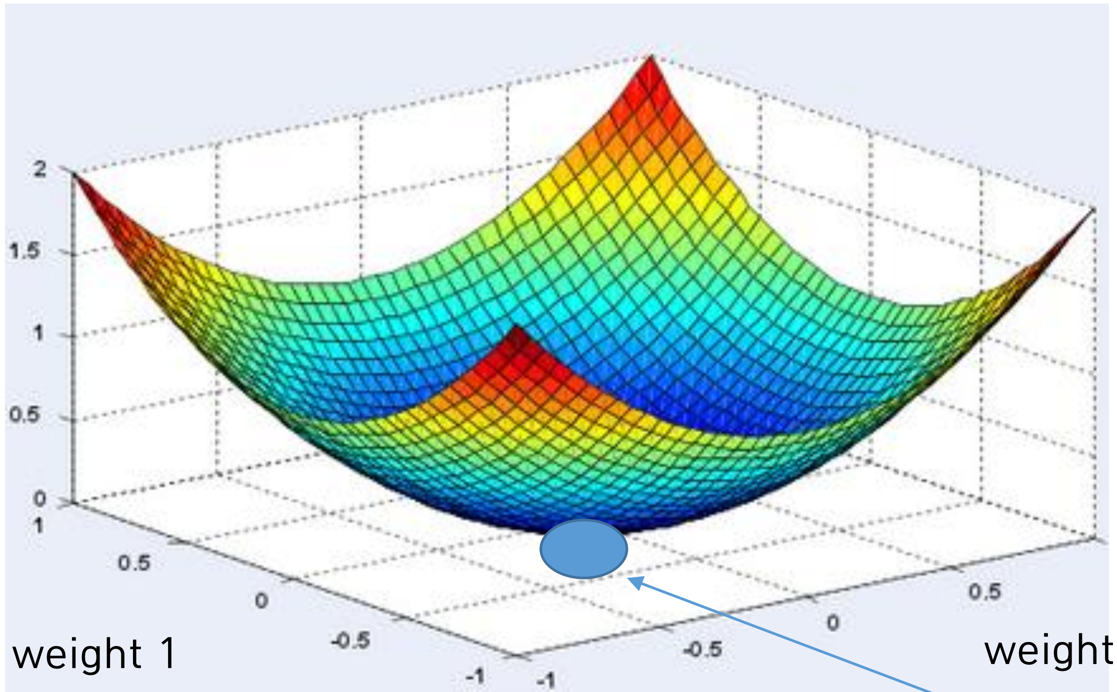
$z_1 = h_1(W_1 x)$ $\quad$ $z_2 = h_2(W_2 z_1)$



How can the weights of a network W_1, W_2 and W_3 be found?

Because mathematical closed solutions for them are rarely obtained, one should depend on numerical optimization using a computer.

$e(\hat{y}, y)$: error function
($\hat{y}$: network output, y: real data)



optimal weights

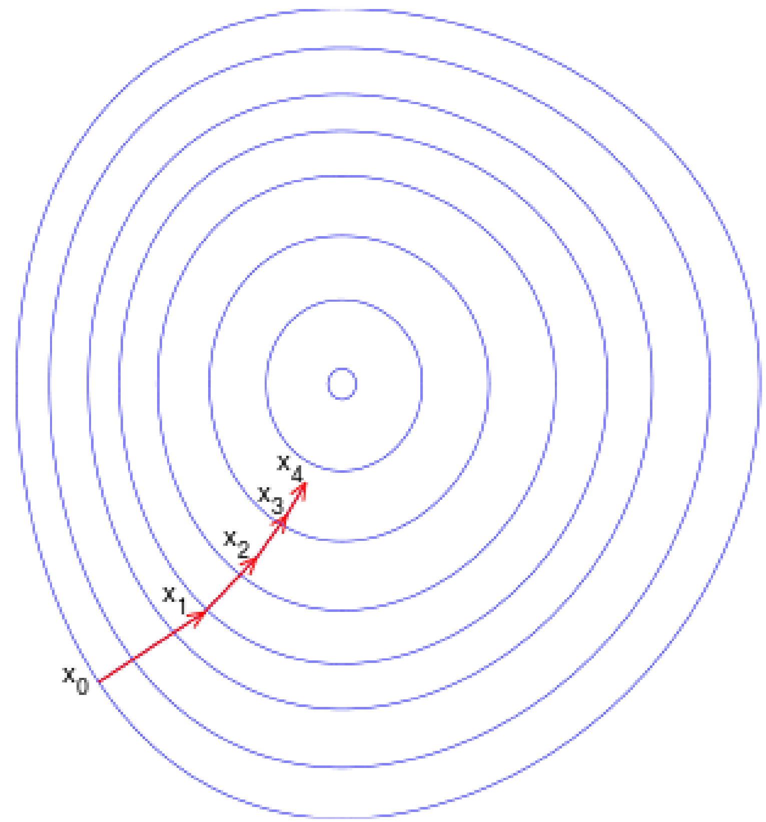
What is network learning?

a work using a computer to find the optimal weight W to minimize the error function between network output $\hat{y}$ and real data y

gradient descent method

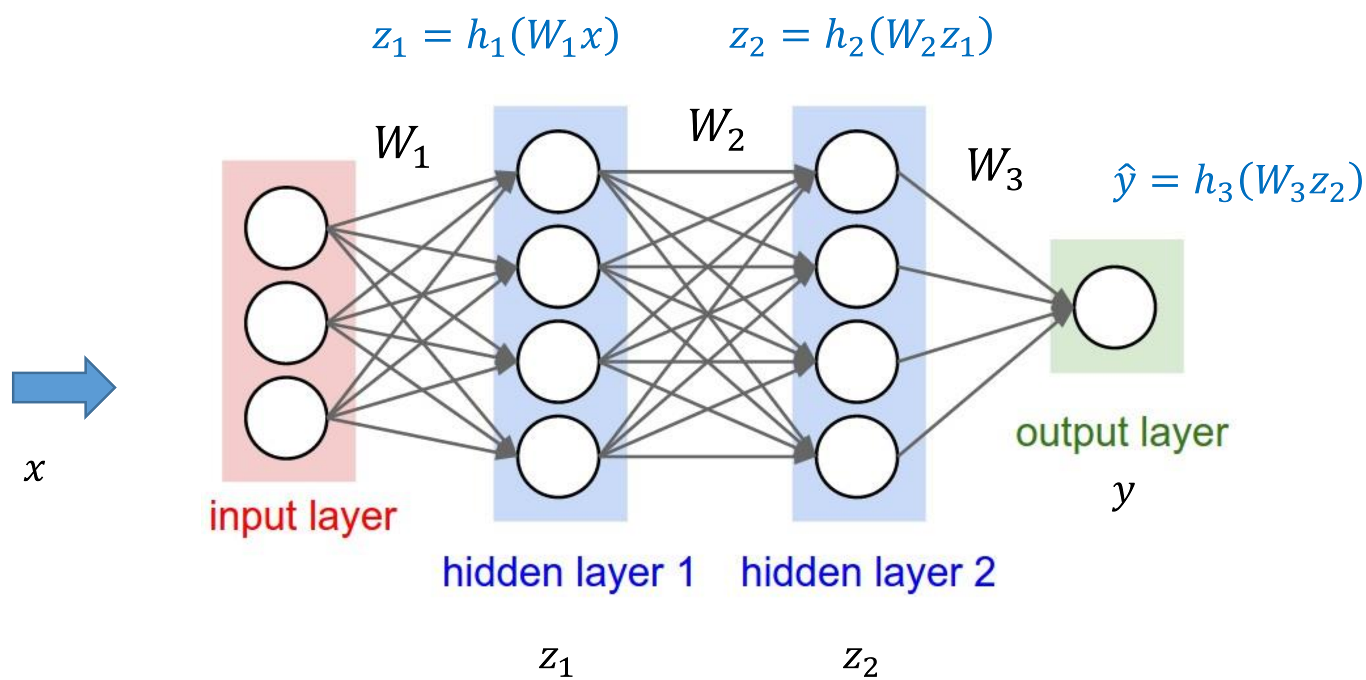
error function $e(w)$

$$w_{n+1} = w_n - \gamma \nabla e(w_n)$$



Weights are updated in the direction of greatest rate of decrease of the error function e

(i.e. the opposite direction of the gradient)



$$e = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(h_3(W_3 z_2) \ - y)^2 = \frac{1}{2}\big(h_3\big(W_3 h_2(W_2 z_1)\big) \ - y\big)^2 = \cdots$$

Chain rule gets to give a more complex gradient form as the number of hidden layers gets larger…

But such forms are so costly to compute…

$$\frac{\partial e}{\partial W_3} = (h_3(W_3 z_2) \ - y) h_3'(W_3 z_2) z_2$$

$$\frac{\partial e}{\partial W_2} = \big(h_3\big(W_3 h_2(W_2 z_1)\big) - y\big) h_3'\big(W_3 h_2(W_2 z_1)\big) W_3 h_2{}'(W_2 z_1) z_1$$

$$z_1 = h_1(W_1 x) \qquad z_2 = h_2(W_2 z_1) \qquad \hat{y} = h_3(W_3 z_2)$$

$$u_l := W_{l+1} z_l \qquad \delta_l := \frac{\partial e}{\partial u_l}$$

summary

$$\frac{\partial e}{\partial W_{l+1}} = \delta_l z_l$$

$$\delta_{l-1} = \delta_l W_l h_l{}'(u_{l-1})$$

$$\frac{\partial e}{\partial W_3} = (h_3(W_3 z_2) \ - y) h_3'(W_3 z_2) z_2$$

$$= (h_3(u_2) - y) h_3'(u_2) z_2 = \frac{\partial e}{\partial u_2} z_2 = \delta_2 z_2$$

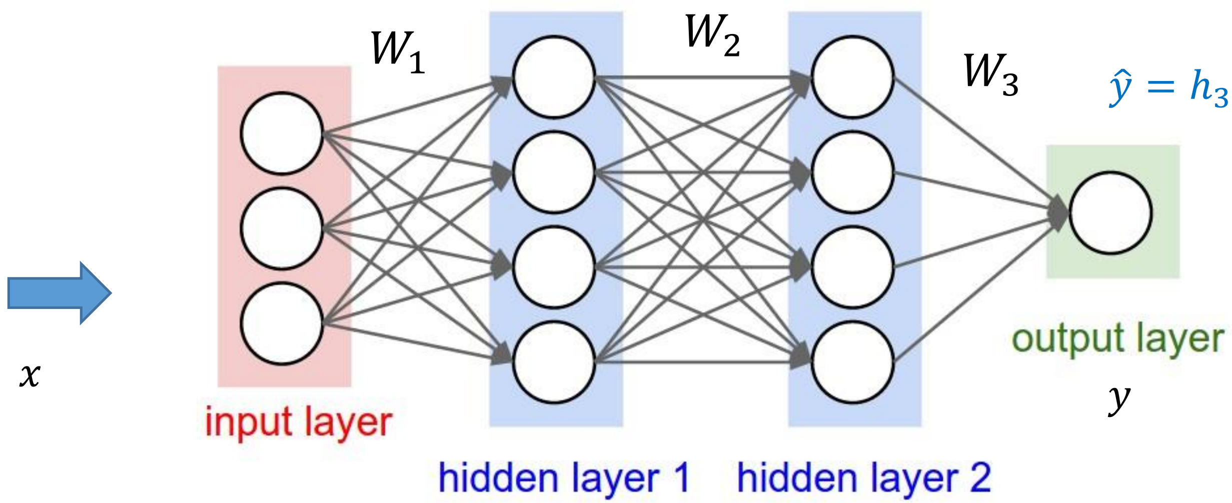
$$\frac{\partial e}{\partial W_2} = (h_3(W_3 z_2) - y) h_3'\big(W_3 h_2(W_2 z_1)\big) W_3 h_2'(W_2 z_1) z_1$$

$$= (h_3(u_2) - y) h_3'(u_2) W_3 h_2'(u_1) z_1 = \frac{\partial e}{\partial u_2} \frac{\partial u_2}{\partial u_1} z_1 = \frac{\partial e}{\partial u_1} z_1 = \delta_1 z_1$$

$$\delta_{l-1} = \frac{\partial e}{\partial u_l} \frac{\partial u_l}{\partial u_{l-1}} = \delta_l \frac{\partial\big(W_l h_l(u_{l-1})\big)}{\partial u_{l-1}} = \delta_l W_l h_l{}'(u_{l-1})$$

$z_1 = h_1(W_1 x)$ $z_2 = h_2(W_2 z_1)$

$\hat{y} = h_3(W_3 z_2)$



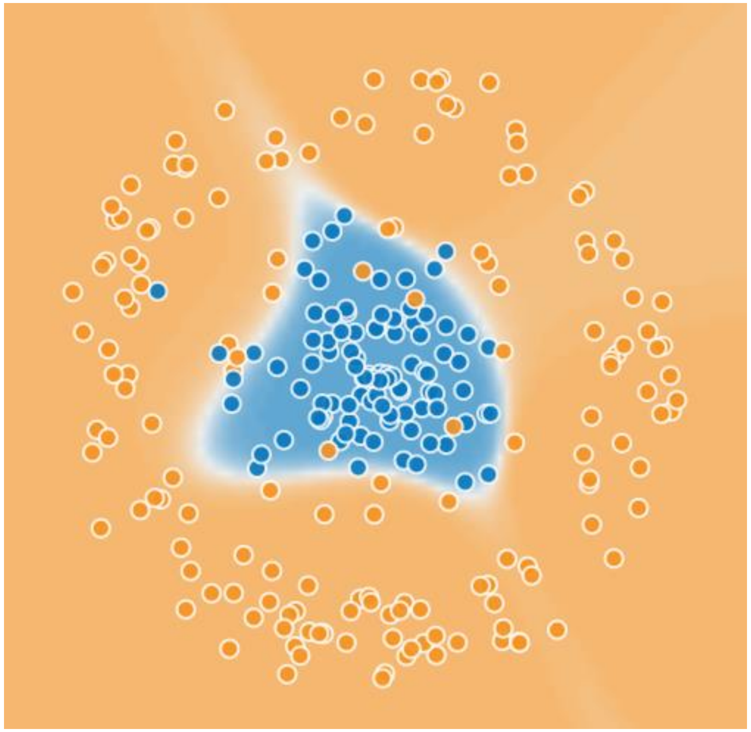
$\delta_1 = g_1(\delta_2)$ $\delta_2 = g_2(\delta_3)$ $g_{l-1}(\delta_l) := \delta_l W_l h_l'(u_{l-1})$

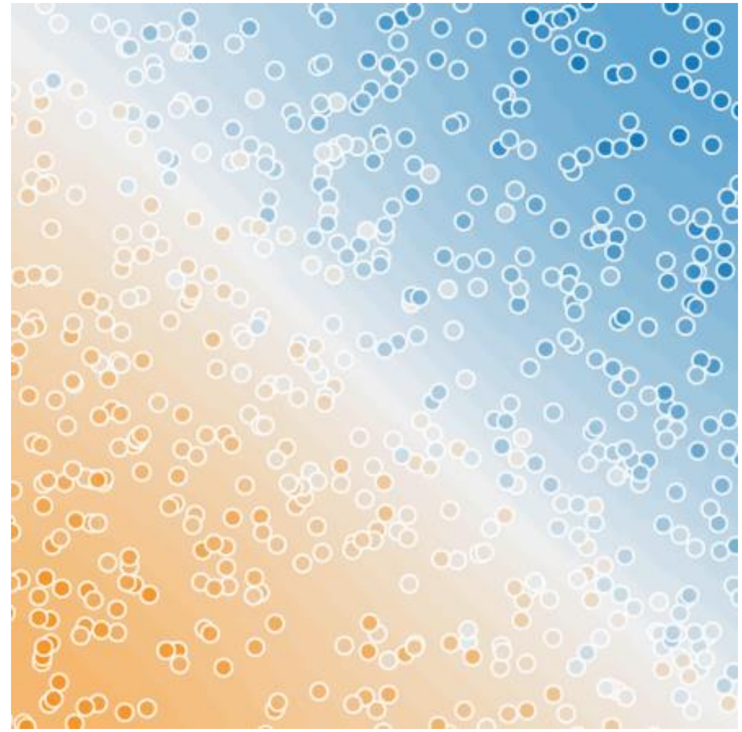
propagation

backpropagation

$$\frac{\partial e}{\partial W_{l+1}} = \delta_l z_l$$

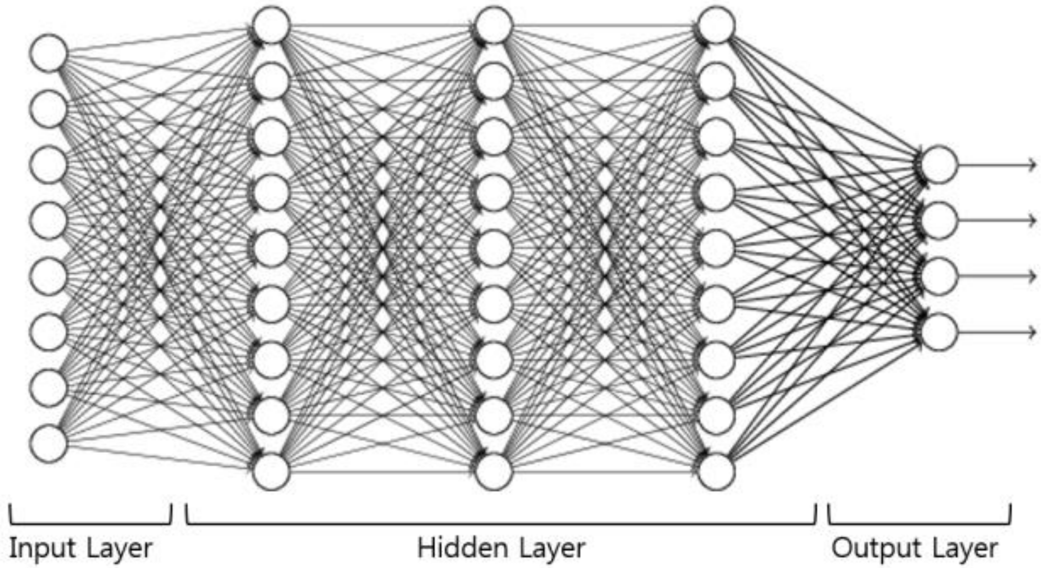
By backpropagation,
one can efficiently compute
the gradient for a update of weight W.





http://playground.tensorflow.org

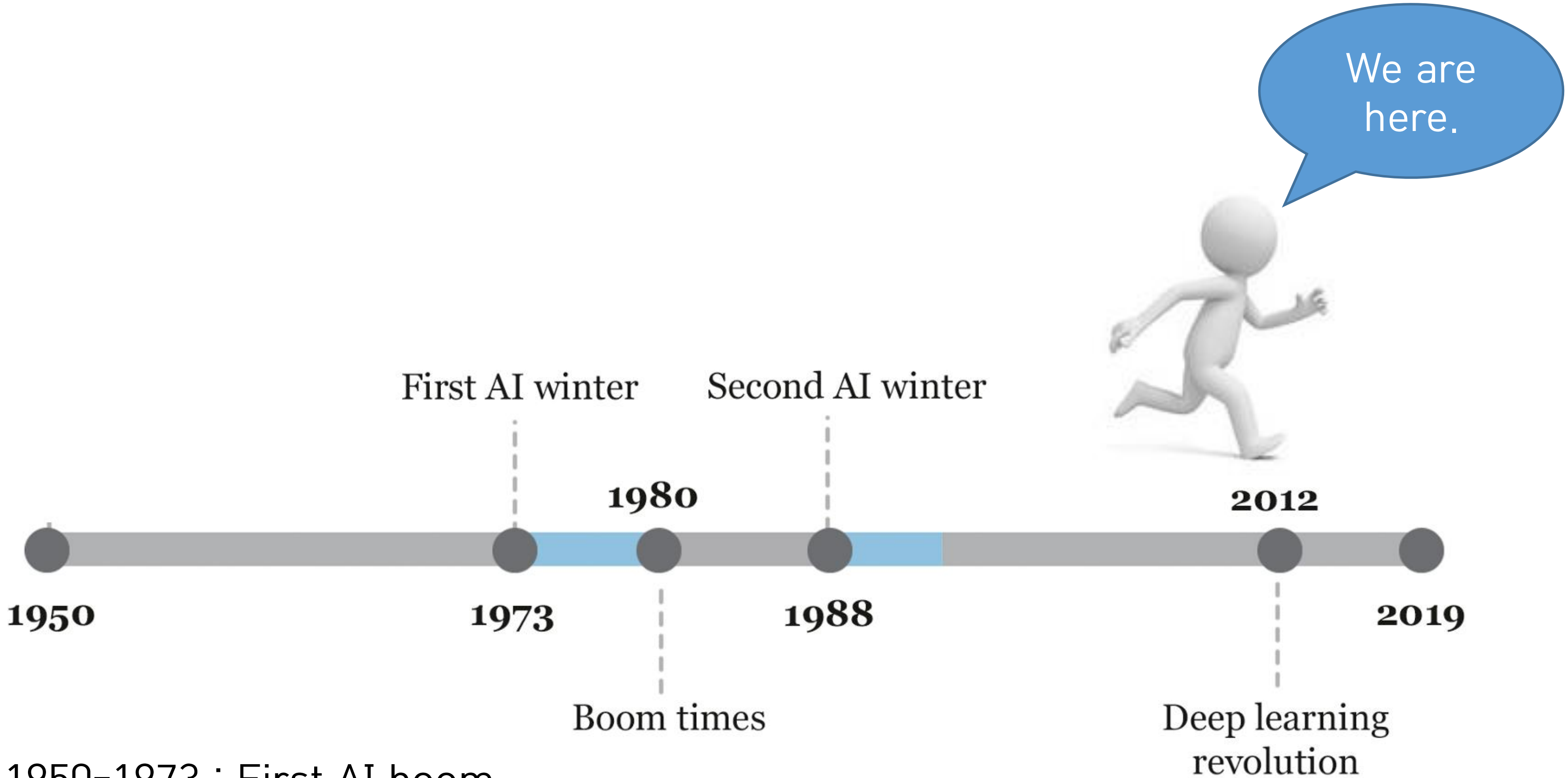
deep neural network



Vanishing Gradient Problem

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

Third AI Boom (2012-)

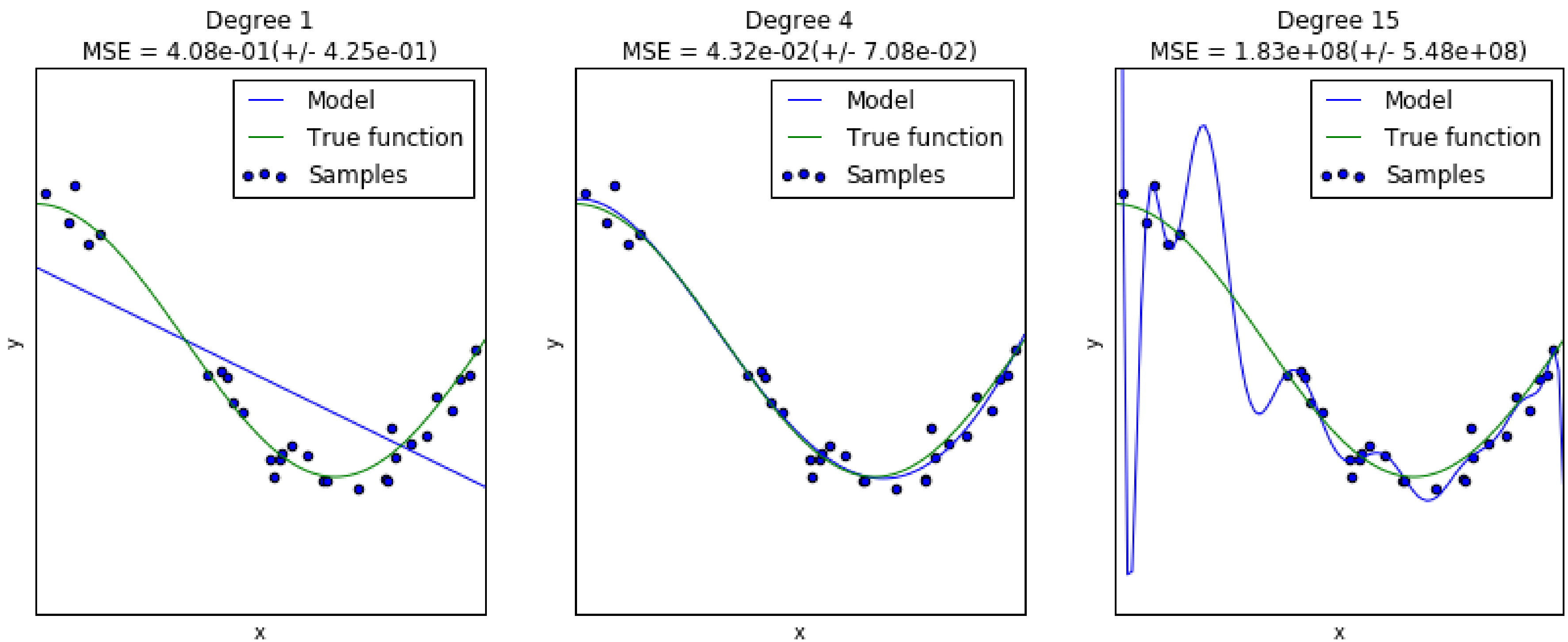


1950-1973 : First AI boom
1973-1980 : First AI winter

1980-1988 : Second AI boom
1988-2000 : Second AI winter

2000-2012 : Many researchers tried to revive the field of AI.

2012- : Third AI boom! Are another AI winter coming again? God knows…



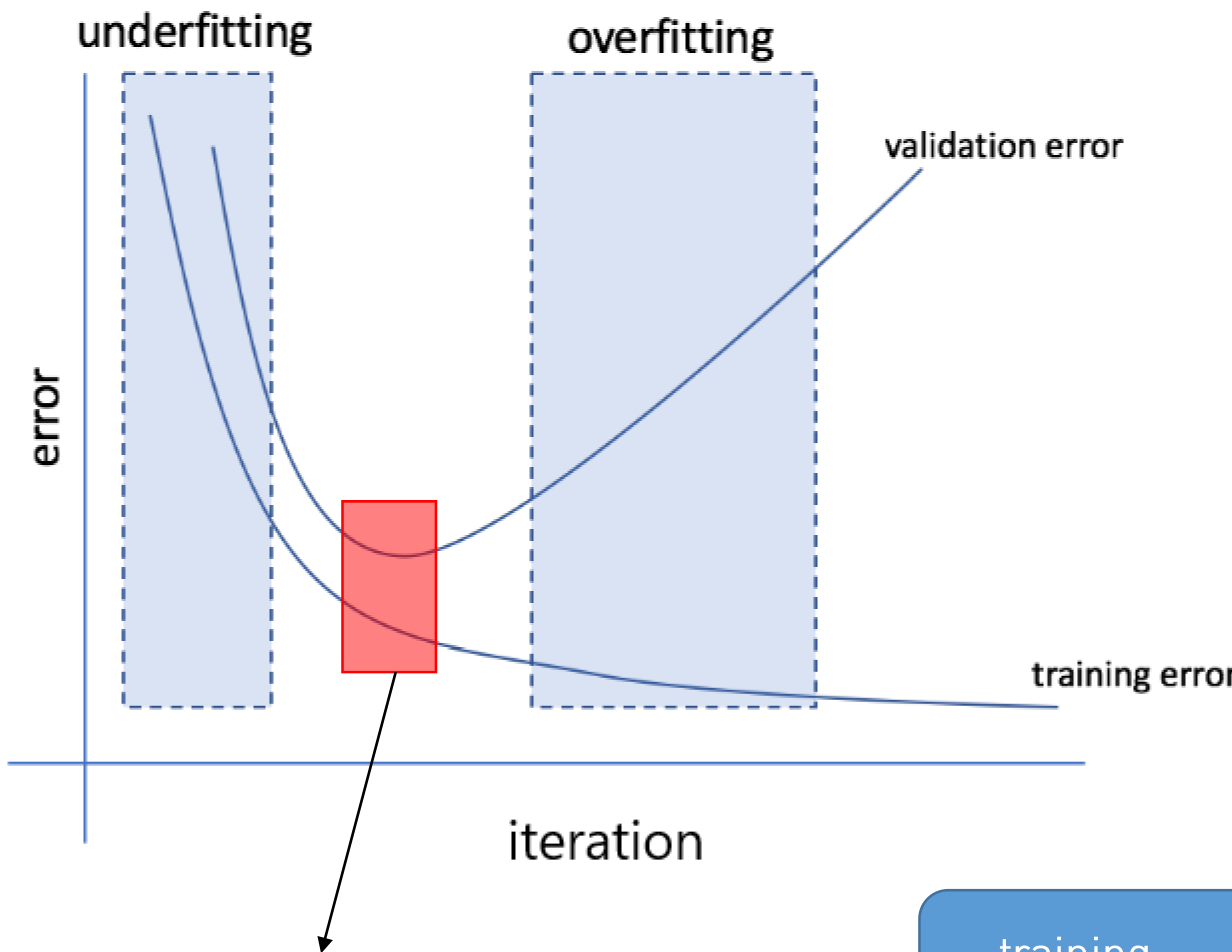
underfitting

good fitting

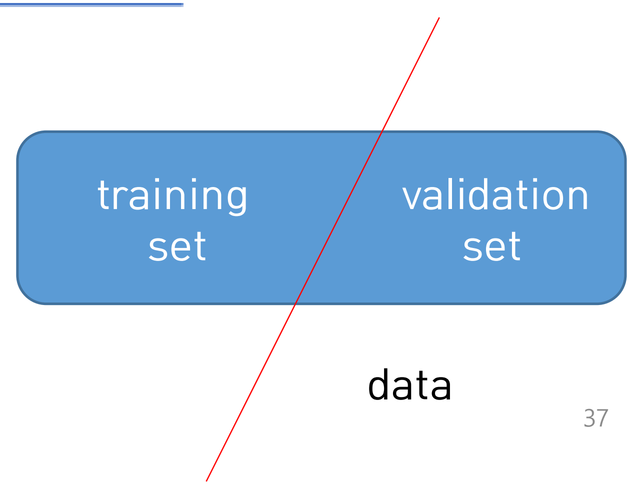
overfitting

Deep networks are hard to train because of vanishing gradients.

Because a deep network have numerous parameters, it tends to memorize the given data itself.



Theoretically, we only have to stop the optimization here.



underfitting

- new activation functions (ReLU, ELU etc.)
- network design to suit given data (CNN, RNN etc.)
- powerful computation tools (GPU etc.)

overfitting

- advanced optimization (stochastic gradient descent etc.)
- pre-training (DBN etc.)
- proper weight initialization (Xavier initialization etc.)
- dropout
- big data (exploding due to internet)

Andrew Ng
Baidu

Yann LeCun
Facebook

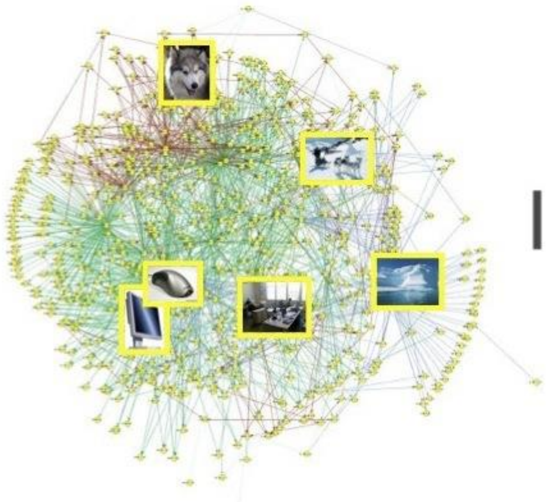
Geoffrey Hinton
University of Toronto

Big data

data sets that are too large or complex to be dealt with by traditional data-processing application software

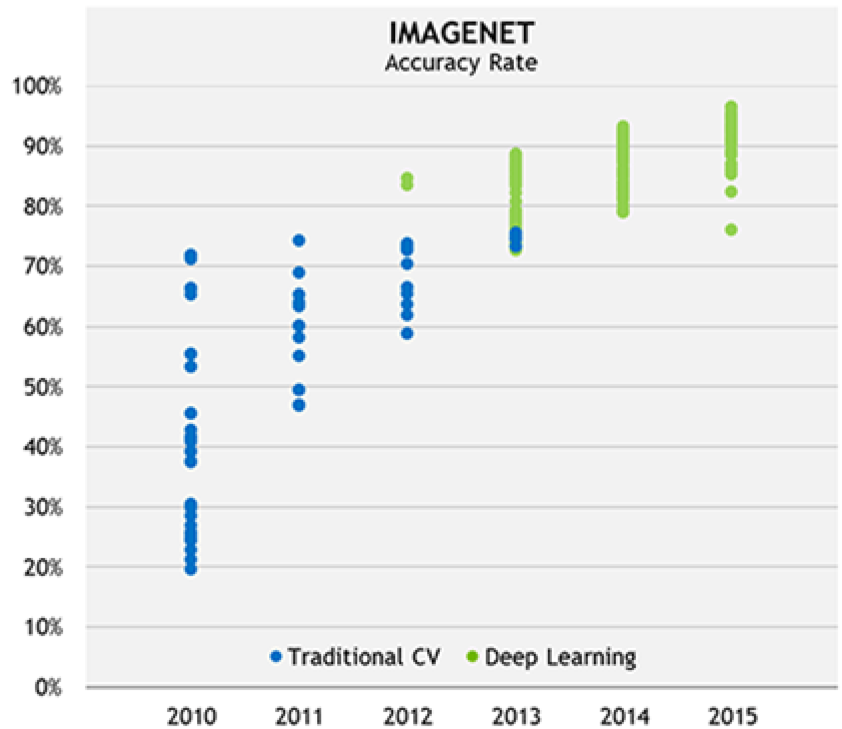
GPU

it enables networks to learn meaningful information from the big data due to its surprising powerful computational ability.

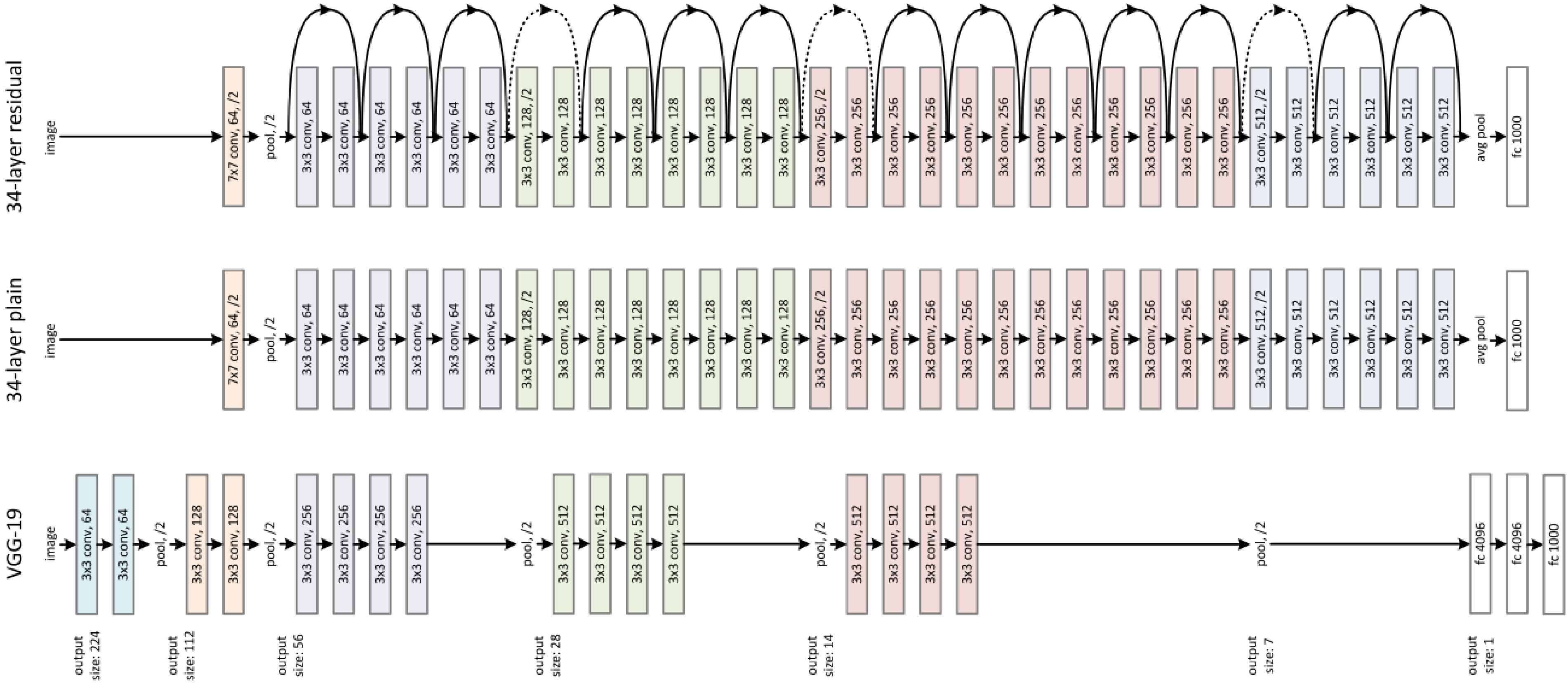


IMAGENET

2015: A MILESTONE YEAR IN COMPUTER SCIENCE



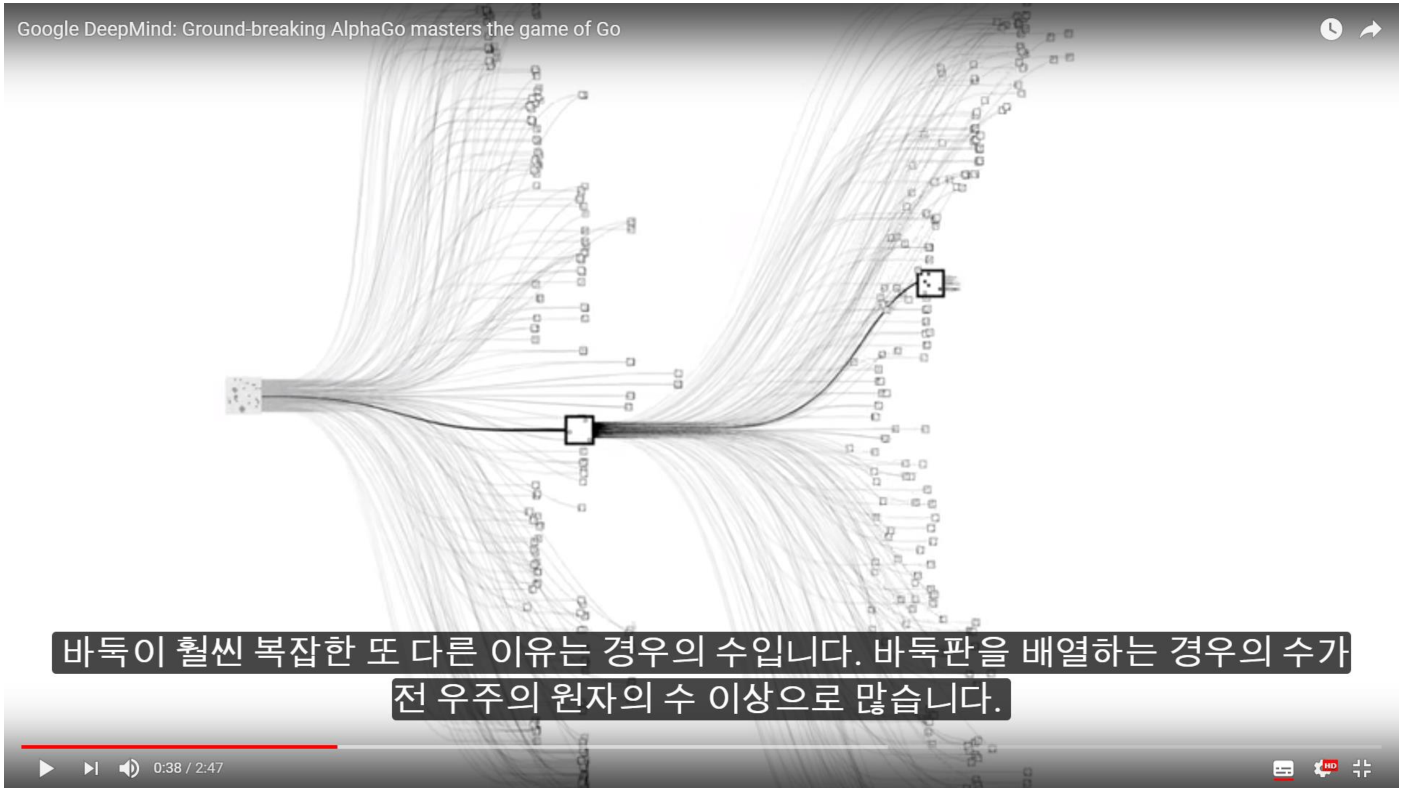
After Alex Krizhevsky won the first prize at Imagenet 2012, most algorithms for image recognition are based upon deep learning.



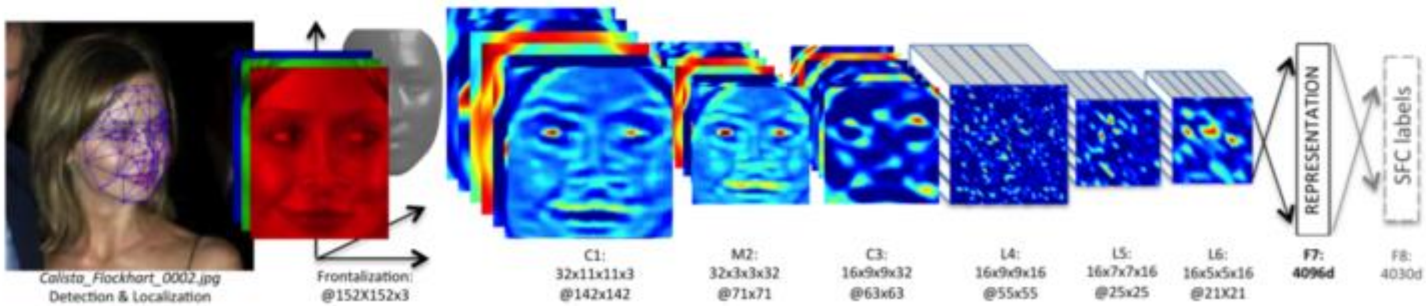
If a powerful computational resource and big data are prepared,
deep networks are generally better than shallow networks
in terms of all of accuracy and robustness.

So networks are getting deeper and deeper.

Google DeepMind Challenge Match 2016 AlphaGo

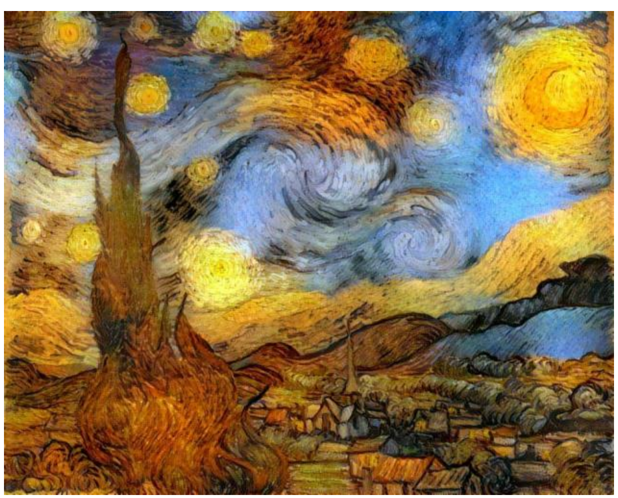


https://www.youtube.com/watch?v=SUbqykXVx0A&feature=youtu.be



Facebook ‘Deep Face’ for face recognition (2015)

Its accuracy 97.35% is almost same as the accuracy 97.53% by a human.
(Please be aware that the FBI's existing best system only achieved 85%.)



Google ‘Deep Dream’ (2015)

AI to draw abstract painting.

AI trader

Goldman Sachs + Kensho (2017)

After Kensho was introduced,
598 traders among 600 were fired.

AI translation

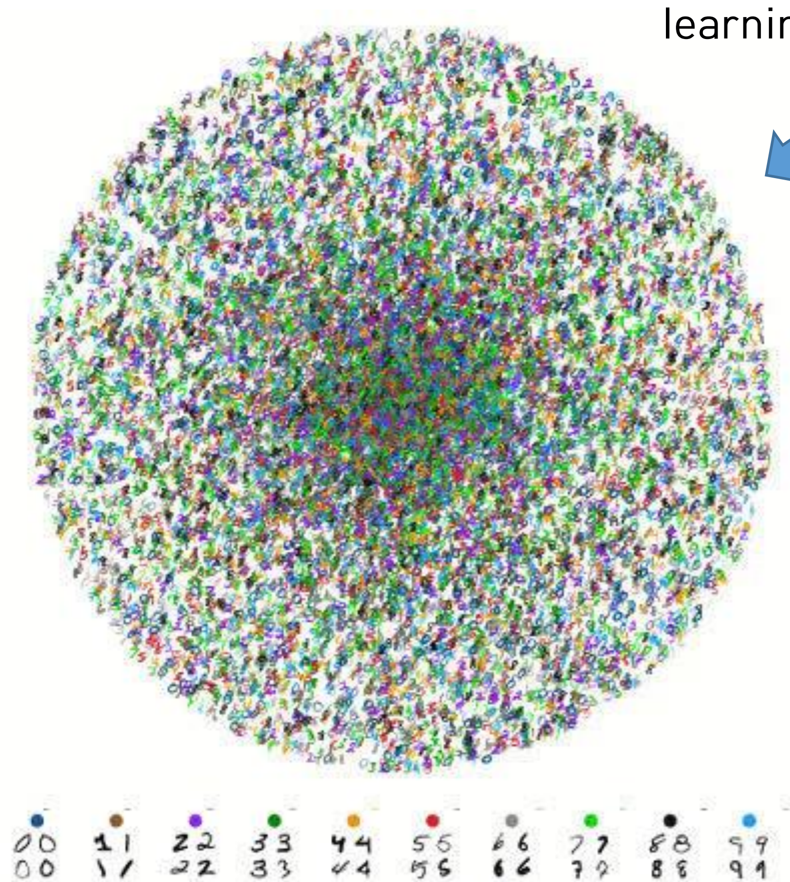
Naver Papago (2017)

the accuracy of language translation engine was improved two times due to deep learning.

generative network

(variational autoencoder,
generative adversarial networks)

learning



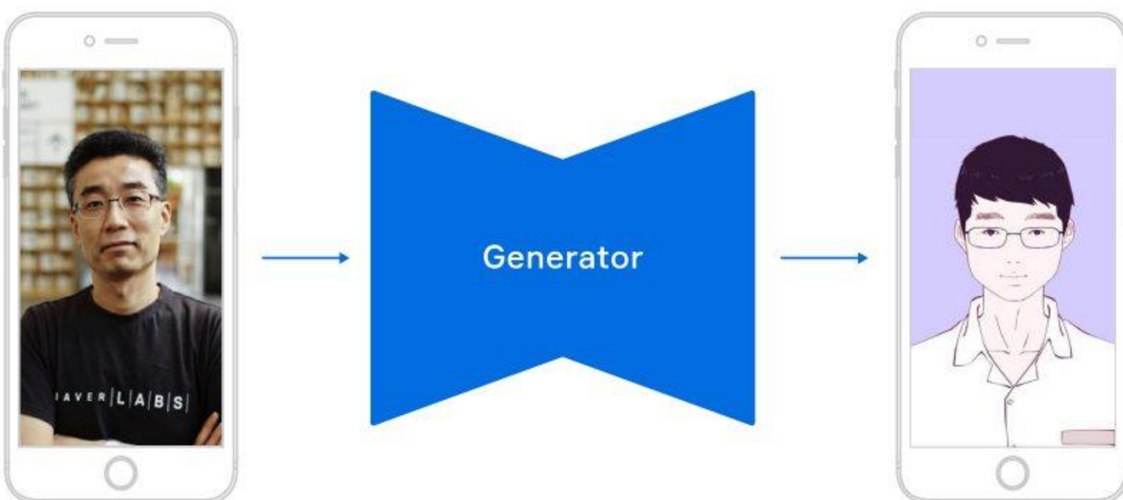
generation

https://www.youtube.com/watch?v=MVBe6_o4cMI&feature=youtu.be

Synthesizing Obama: Learning Lip Sync From Audio
2017, SUWAJANAKORN el.al., University of Washington

AI changing a rough sketch to a picture of Vincent Gogh

2017, Nvidia



AI changing a person in a photo to a comic character resembling the person.

2017, Naver

Why is deep learning awesome?

https://www.youtube.com/watch?v=C2FS9WVm7j4

a steel cut in the movie, “Her”

Thank you for listening