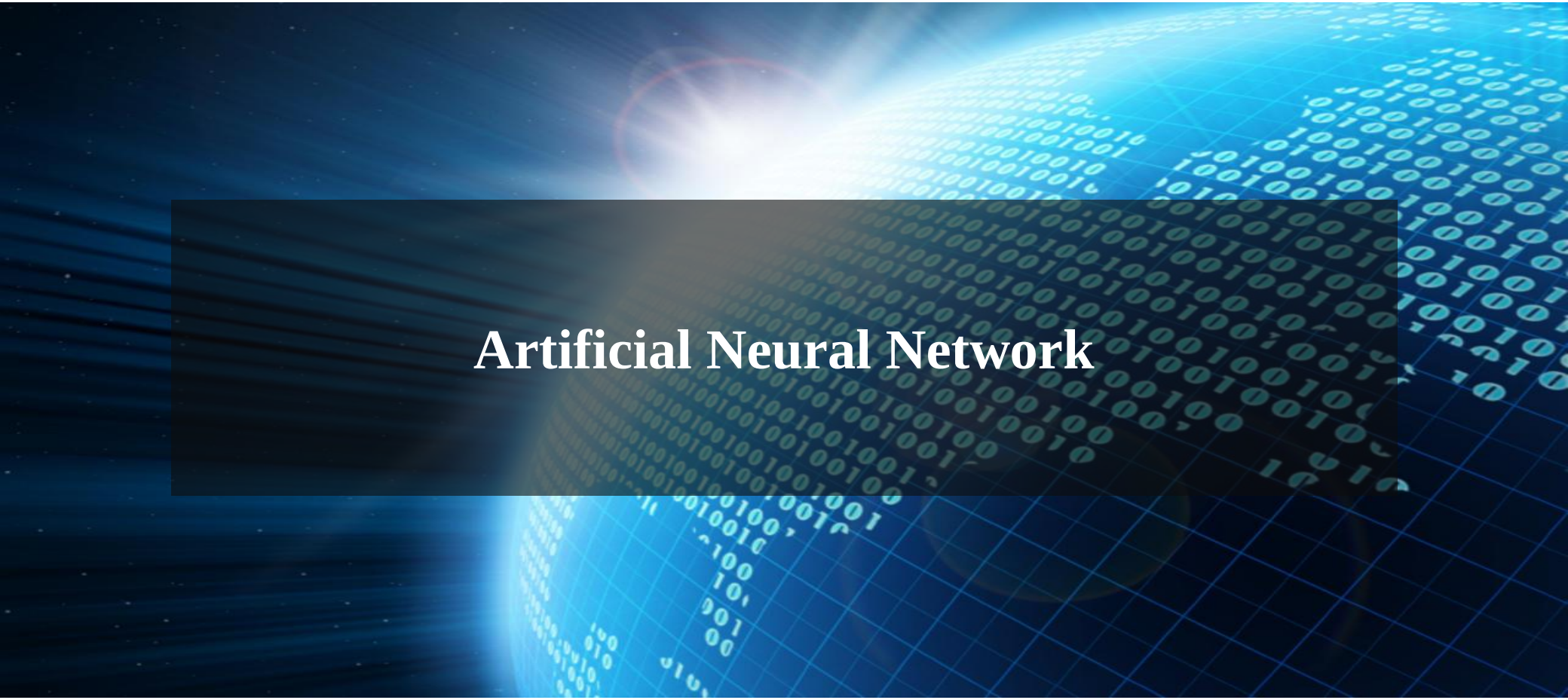


Artificial Neural Network

2023

Part I. Artificial Neural Network

I. 개요 및 현황

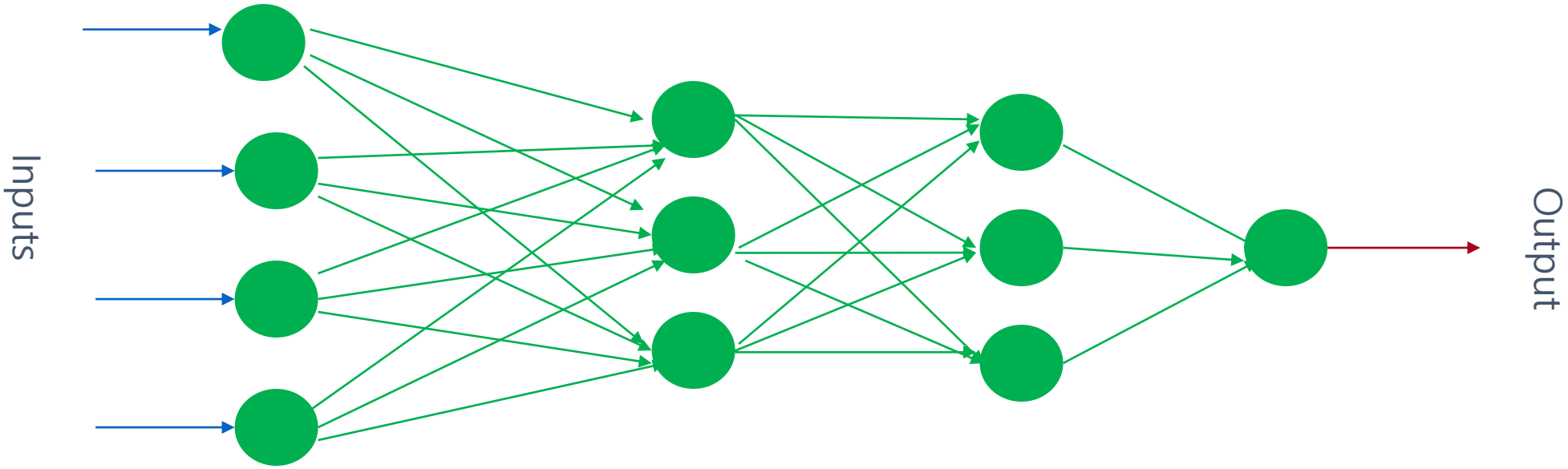
1. 인공신경망: 뉴런을 모방한 노드들이 각각 Input Layer, Hidden Layer, Output Layer로 구분되며 데이터를 입력받아 변환하여 원하는 결과로 출력하는 네트워크를 구축하는 것

2. 입력값에 대해 은닉층에서의 비선형 변환을 통해 출력을 예측하는 인공신경망 기법에서, 은닉층에서의 비선형변환이 여러 차례 반복되는 심층신경망에서의 학습기법이 딥러닝

3. 딥러닝은 다수의 은닉층에서도 잘 작동하며, 많은 Feature에 대해 많은 계산을 수행하며, 높은 예측력을 보여줌

- 인공신경망 (Artificial Neural Network, ANN): 데이터 기반의 AI
  - 사람 뇌의 정보처리 방식을 모사한 알고리즘
  - 데이터를 분석하여 예측, 분류 등 의사결정 문제를 해결하는데 적용할 수 있는 기법

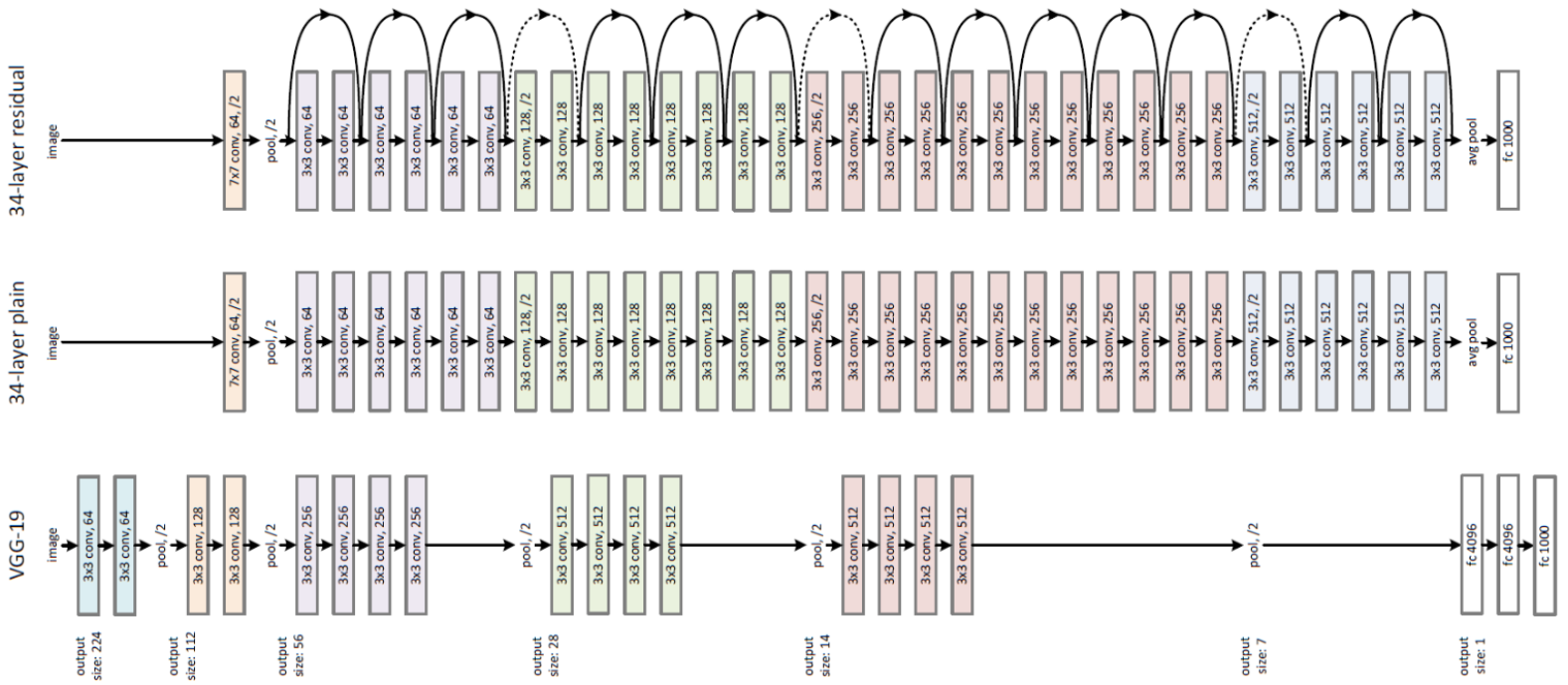
- Artificial Neural Network(ANN, 인공신경망)
  - 데이터를 입력받아 변환하여 원하는 결과로 출력: 각각 Input Layer, Hidden Layer, Output Layer로 구성
  - 예측 성능이 우수
  - 모형을 통한 추론은 어려움



인공신경망+깊이

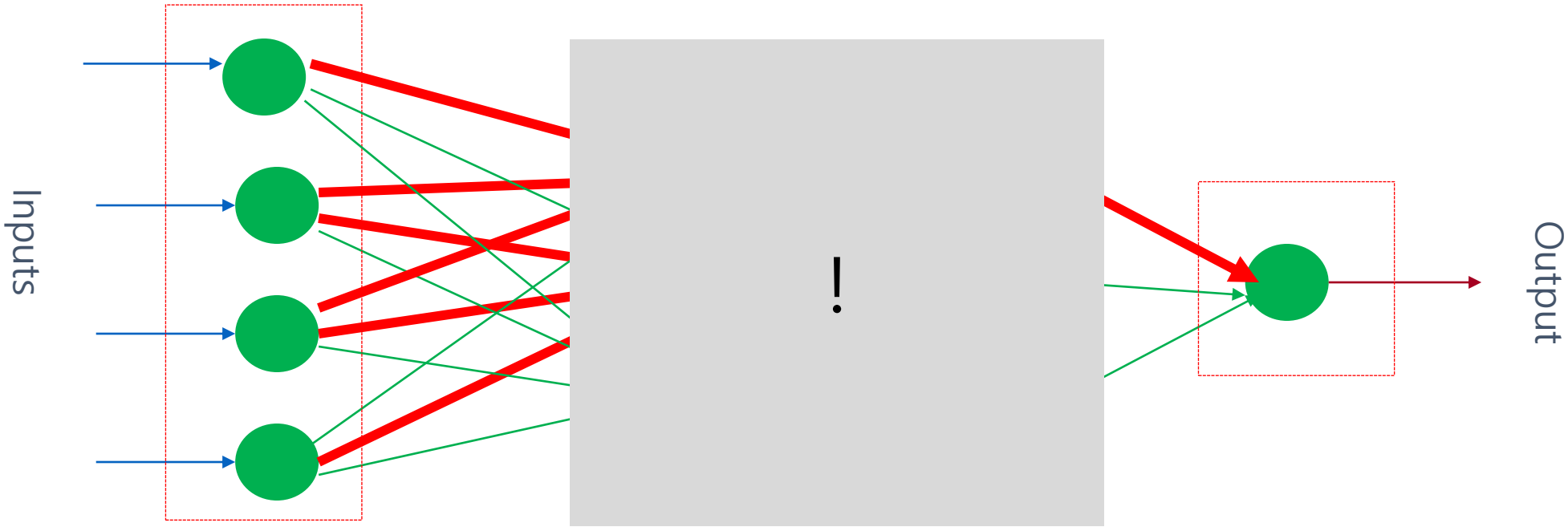
- Hidden layer가 2개 이상인 NN(Neural Network)을 Deep Neural Network(DNN)
- DNN에서의 학습을 딥러닝이라고 부름

비선형 변환과정들

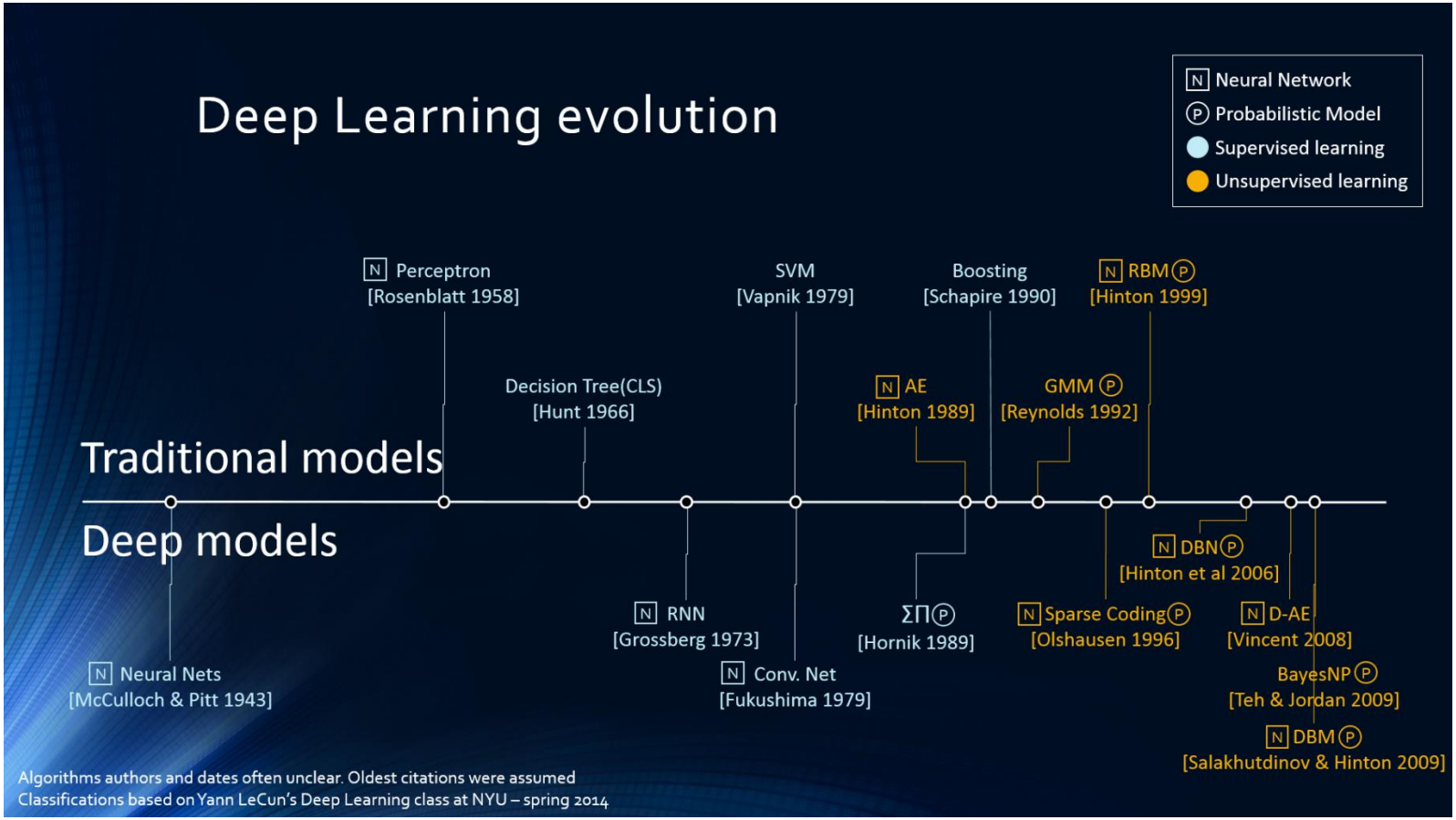


RESNET

- **딥러닝의 특성**
  - 다수의 은닉층 = 계산량



- **딥러닝 알고리즘의 발전**
  - 기존 ANN의 난제들을 해결
  - CNN, RNN, LSTM, Transformer, BERT 등으로 발전 중



II. Deep VS Shallow Learning

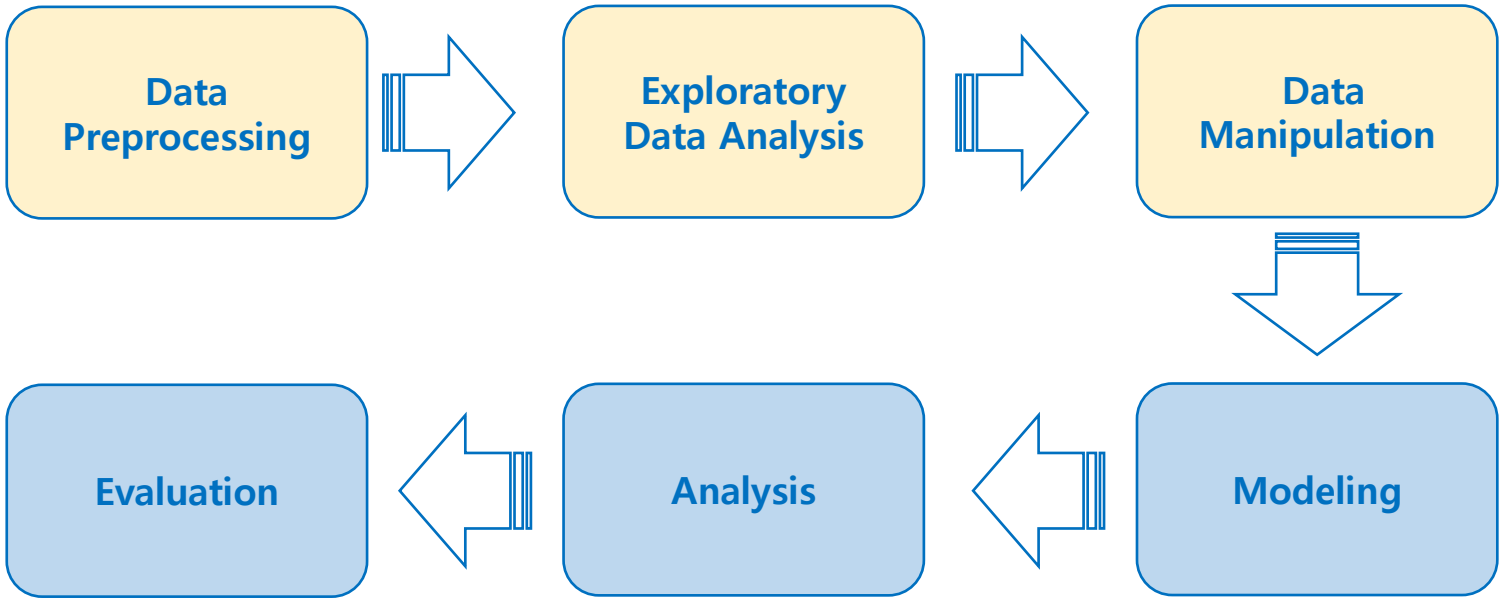
1. 변수는 관측된 개체들이 갖는 특성 또는 속성으로 데이터를 구성하며, 모형은 주어진 변수를 활용하여 데이터를 분석하고 문제를 해결하는 기법이나 알고리즘

2. Shallow Learning: 다수의 비선형 변환을 거치지 않고 주어진 변수로 부터 직접적으로 모형을 도출

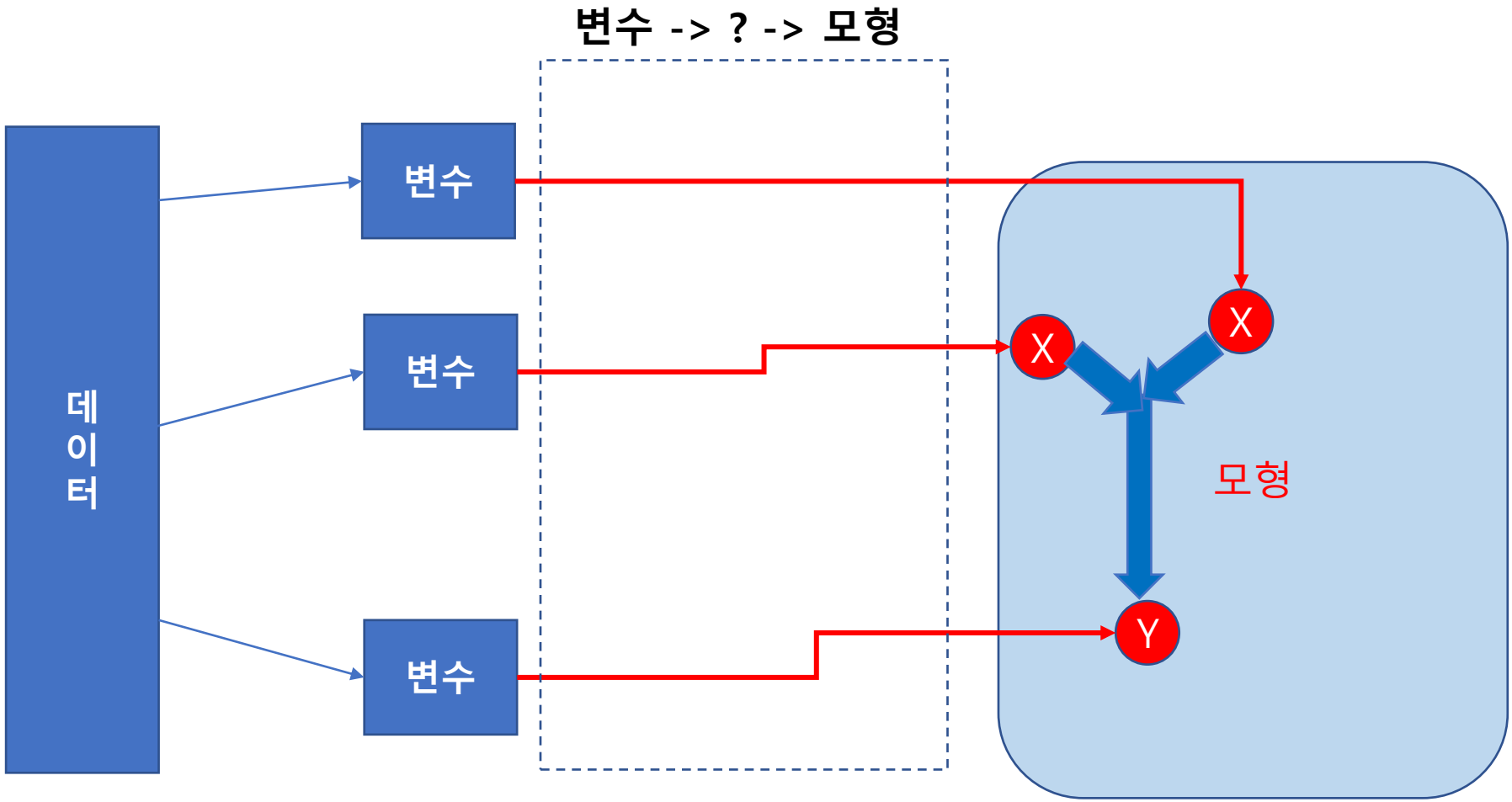
3. 분석의 목적이 설명인지, 예측인지에 따라, 변수를 다루는 방식이 다른 딥러닝과 샐로우러닝 중 선택해서 사용해야 함

변수
:관측된 개체들이 갖는 특성 또는 속성으로, 다양한 값을 갖을 수 있음, Feature라고도 지칭

모형
: 분석 목적에 맞게 데이터에서 인사이트를 찾아내는 적절한 기법이나 알고리즘



변수에서 모형으로



Shallow Learning VS Deep Learning

예:

IF

변수1 > 50,

TEHN

내일 비가 오는 것으로 예측!

예:

변수1 X 20 +10 = 변수3

예:

변수1의 20%, 변수2의 80%를 반영해서 합산한 값에
활성화함수를 통해 비선형 변환을 적용해서 얻어진 값이 새로 만든 변수 1

변수1의 40%, 변수2의 60%를 반영해서 합산한 값에
활성화함수를 통해 비선형 변환을 적용해서 얻어진 값이 새로 만든 변수 2

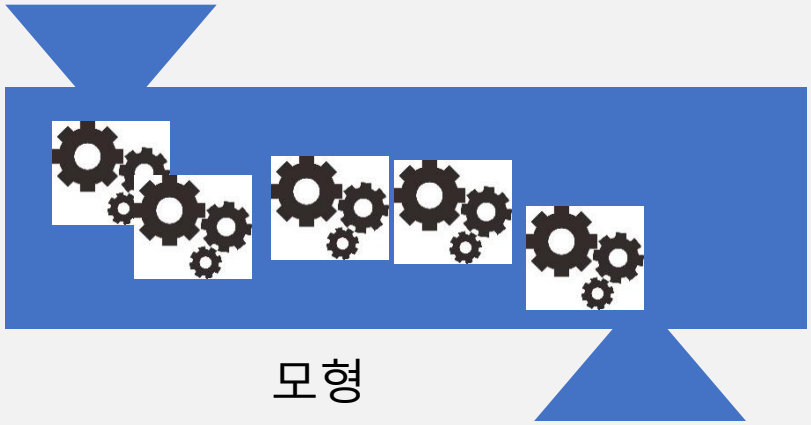
새로 만든 변수 1과 2에 대해 비선형 변환을 적용

....

그 결과로 Y변수를 모델링

분석의 목적: 설명? 예측?

딥러닝과 샐로우러닝의 선택!



모형

엄선된 적은 수의 변수 하나하나가
중요하고, 모형의 설명이 중요한 경우!

Shallow

장점: 설명 가능성
단점: 모형의 데이터 Fitting

?

Deep

단점: 설명 가능성
장점: 모형의 데이터 Fitting

변수(=feature)가 너무 많아 설명보다는
피팅이 중요한 경우!

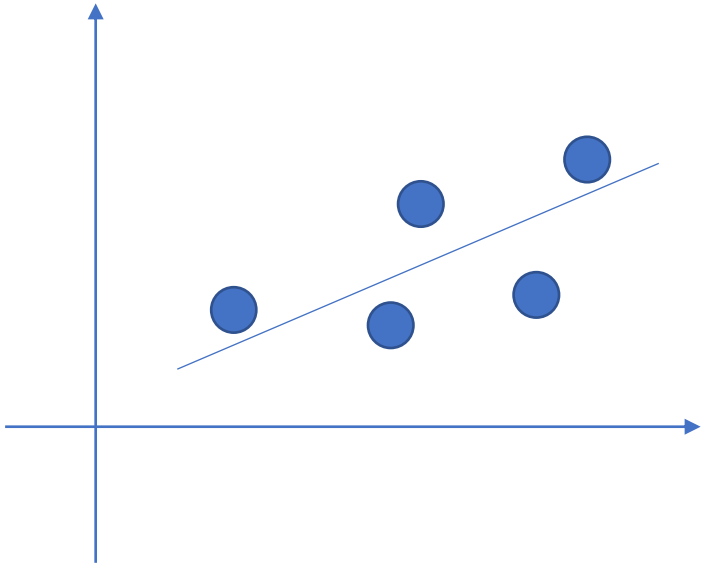
딥러닝의 활용 분야

많은 Feature가 제공되며, 예측을 잘 해야 하는 분야들!

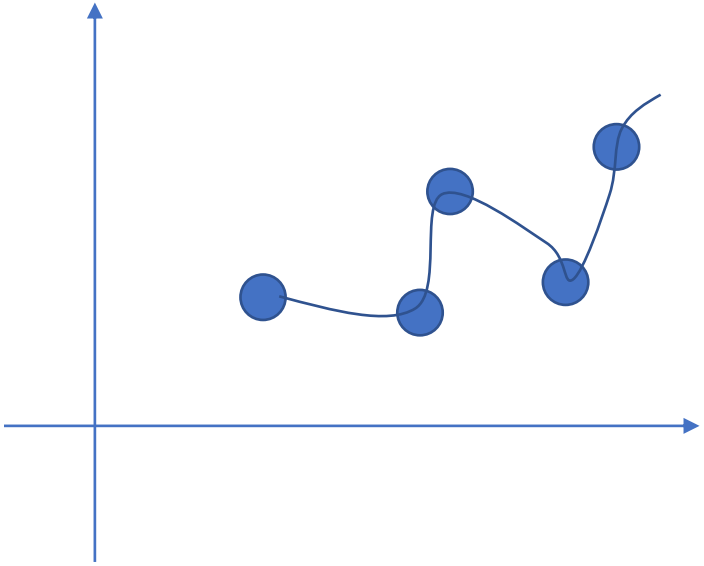
Overfitting?

- 피팅(Fitting, 적합)이란, 주어진 데이터를 모델링하는 과정
- 주어진 데이터에만 과도하게 피팅된 것이 오버피팅
- Feature가 많고, 데이터가 많으면 겪는 이슈

일반적인 Fitting

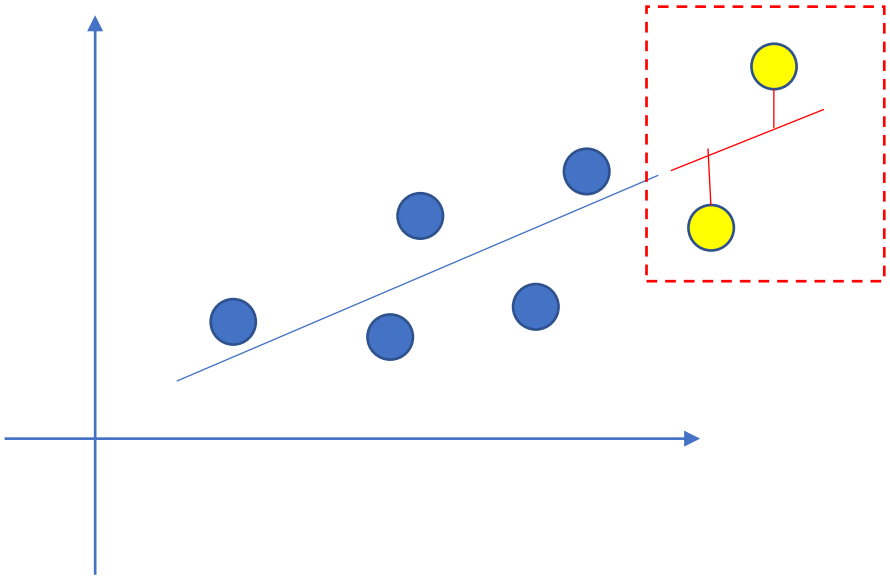


Over Fitting

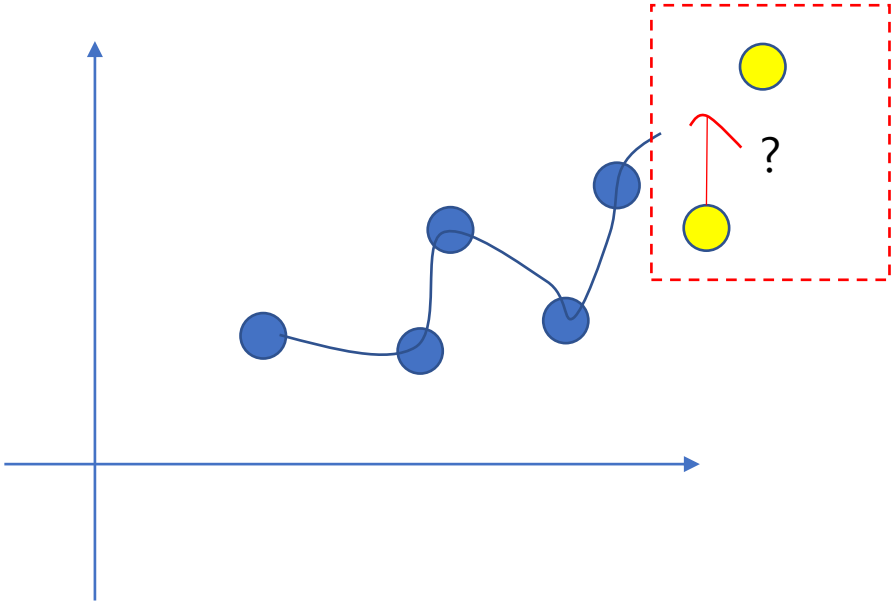


Overfitting?

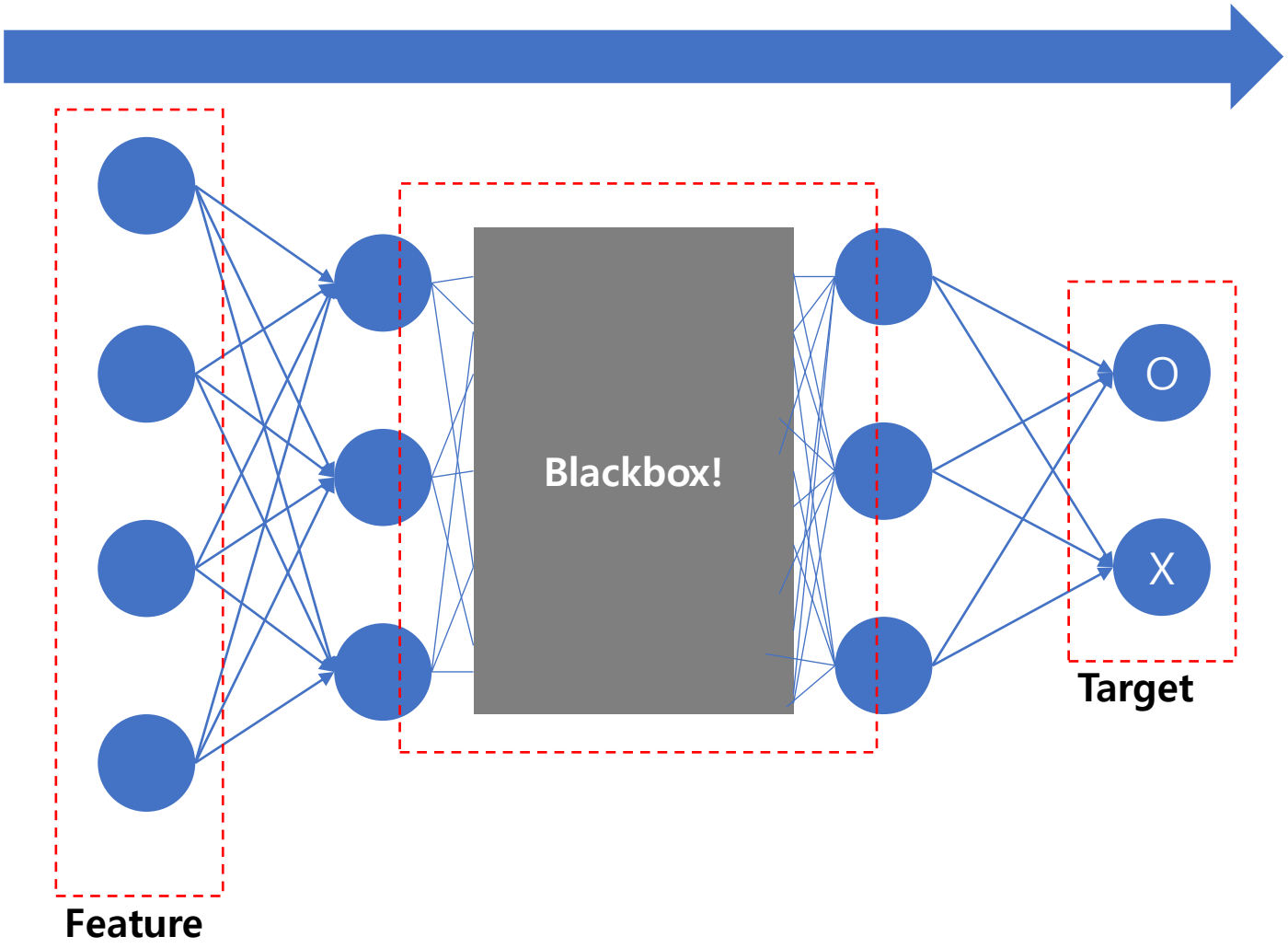
새로운 데이터에 대한 예측과 오차



Over Fitting 시
새로운 데이터에 대한 예측과 오차

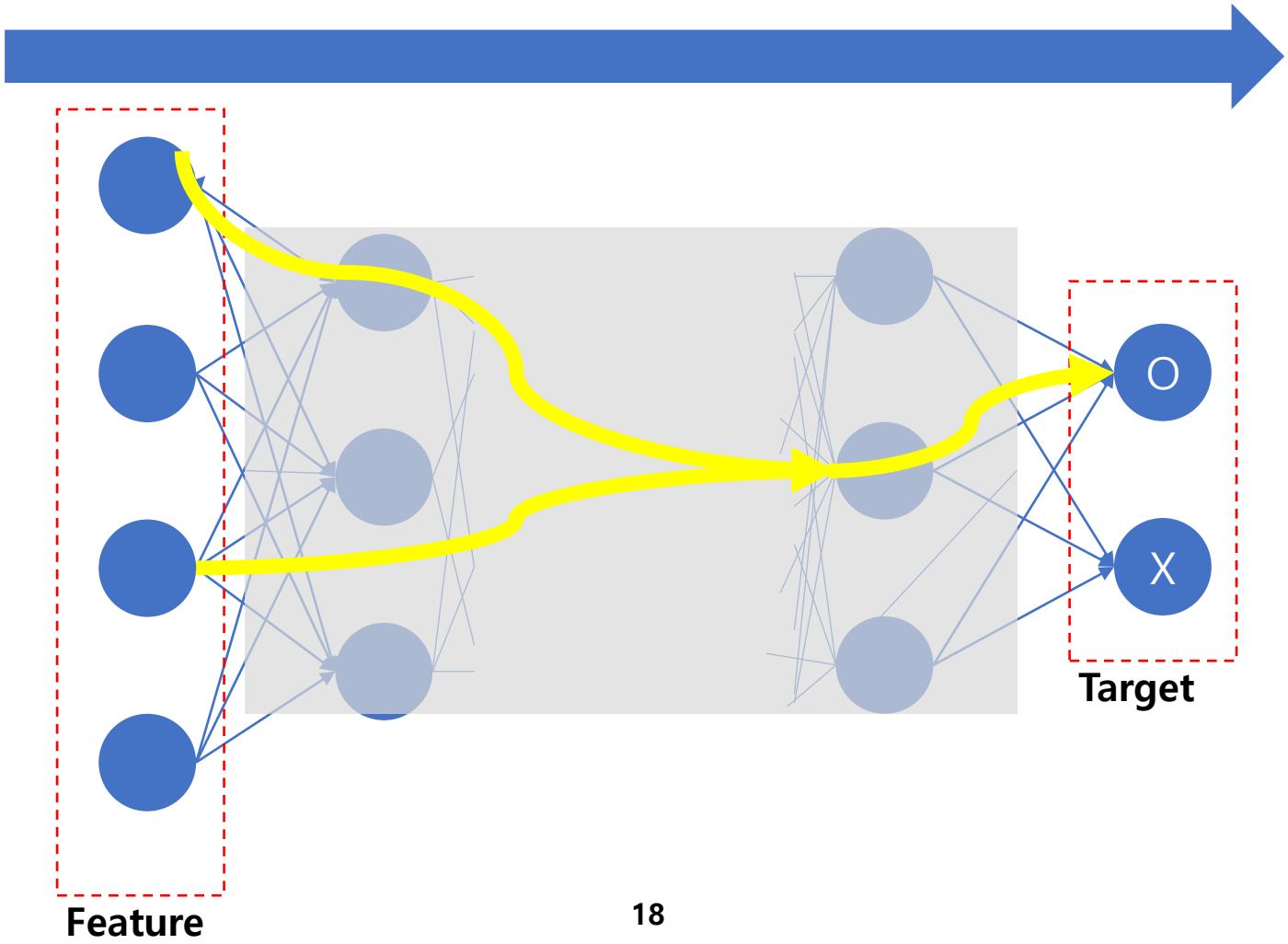


딥러닝=Blackbox model



설명 가능한 AI(eXplainable AI)

eXplainable AI로 XAI라고도 하며,
딥러닝과 같은 AI 모형의 예측 결과에 이르게 된 이유를 설명



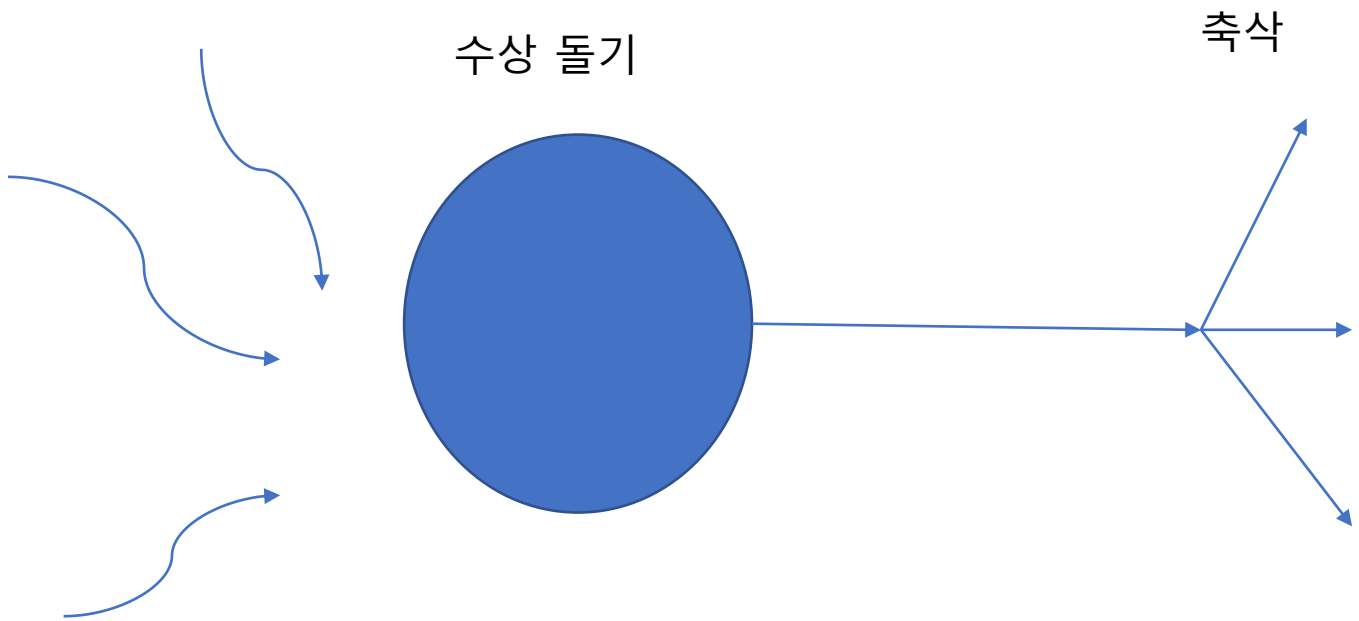
III. 인공신경망 구성

1. 퍼셉트론(Perceptron)이란: 1957년에 코넬 항공 연구소의 프랑크 로젠블라트에 의해 개발, 단순한 형태의 피드포워드 네트워크이며 선형분류기

2. 퍼셉트론의 작동 원리: 입력값들의 가중합을 활성화함수를 거쳐 분류하도록 함

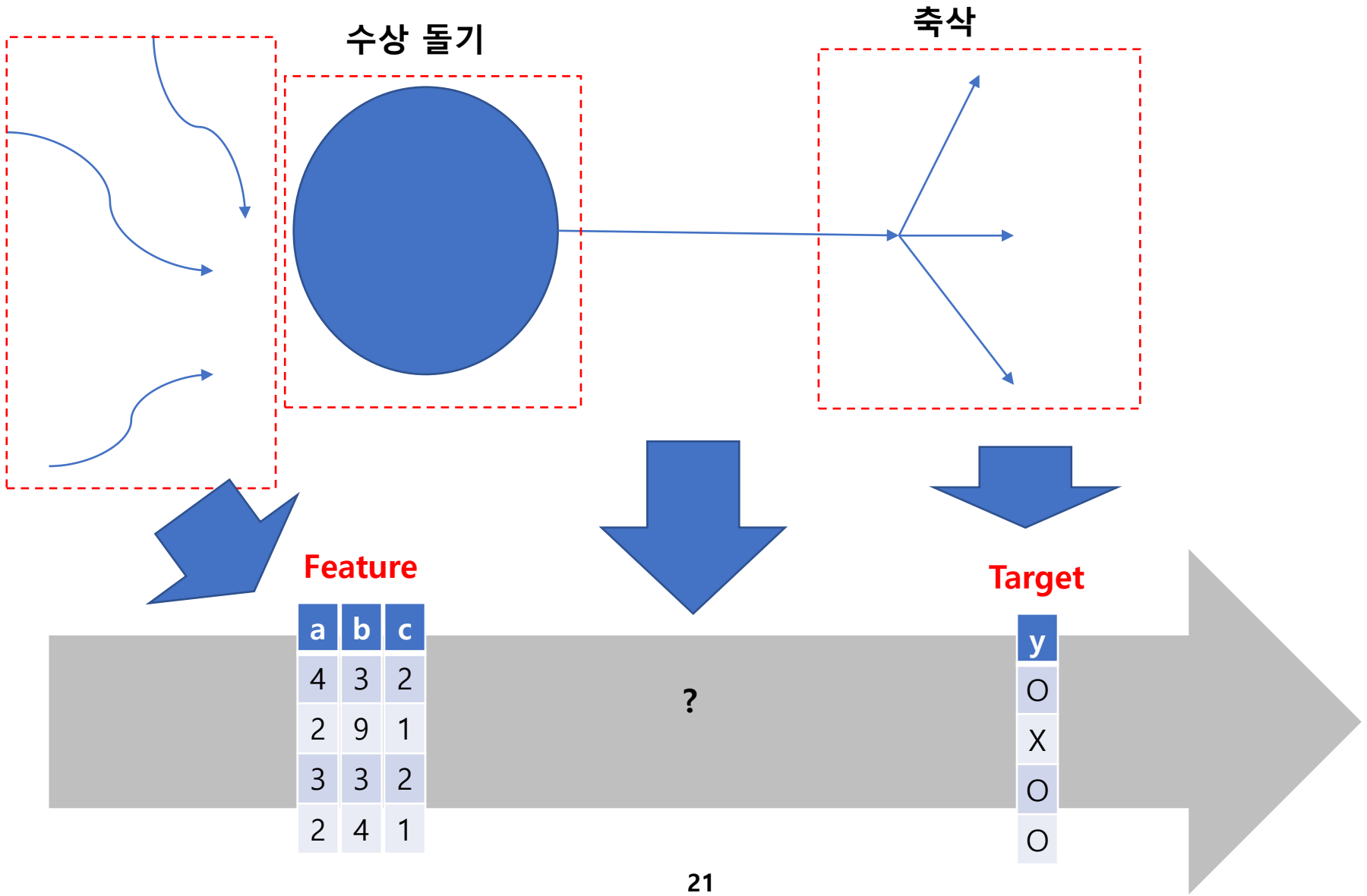
3. 퍼셉트론 한계 및 다층 퍼셉트론: 퍼셉트론은 비선형 분류에 한계가 있으며, 다층퍼셉트론으로 확장하면 비선형분류도 가능해짐.

뉴런?



III. 인공신경망 구성

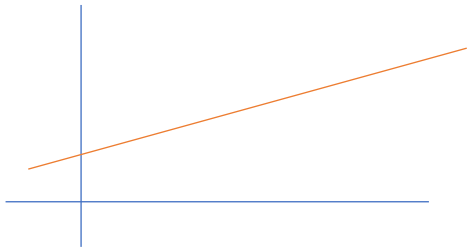
뉴런?



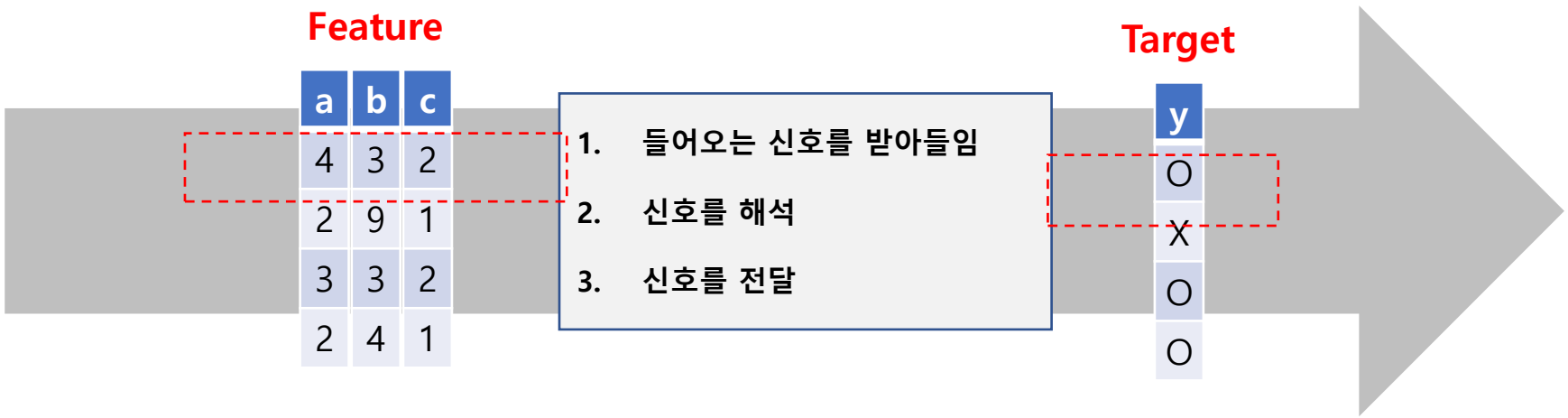
퍼셉트론

by
1957년
코넬 항공 연구소(Cornell Aeronautical Lab)
프랑크 로젠블라트 (Frank Rosenblatt)

간단한 형태의 선형분류기이자 인공신경망!



퍼셉트론: 신경세포와 같이 들어오는 신호를 바탕으로, Target을 계산



4, 3, 2 → O

범주

(4 X ?)+(3 X ?) + (2 X ?) →

만약 이 값이 어떤 수 보다 크면 O, 아니면 X

퍼셉트론

(4 X ?)+(3 X ?) + (2 X ?) → 만약 이 값이 어떤 수 보다 크면 O, 아니면 X

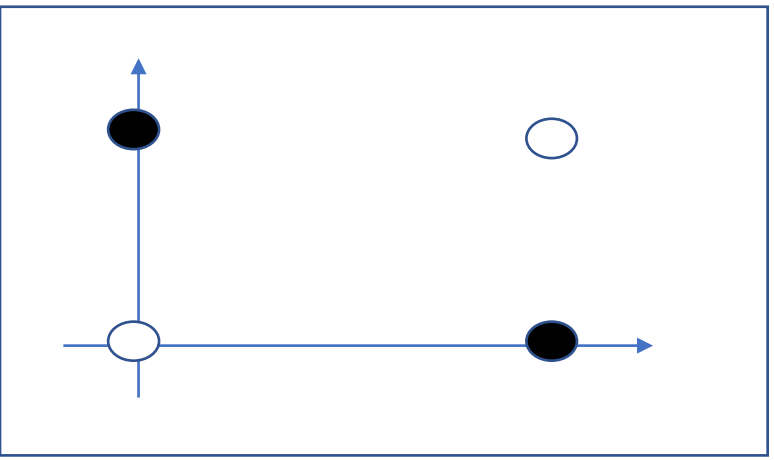
- 선형 분류기: 직선식을 통한 분류
- 직선식으로 분류를 못하는 경우에는 한계
- 예: XOR 형태의 데이터

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

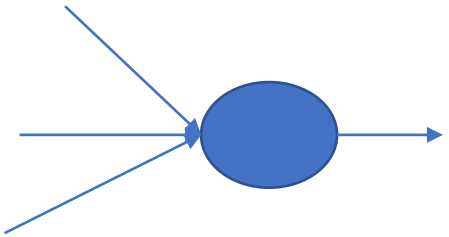
하나의 직선으로
검은 원과 하얀원
분류할 수 없음

다층 퍼셉트론으로 해결!

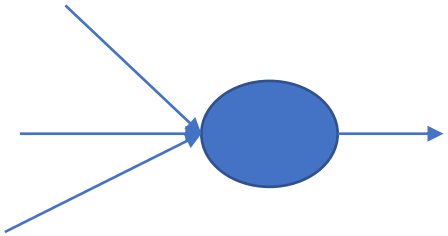
다층 퍼셉트론: 비선형 분류 가능!



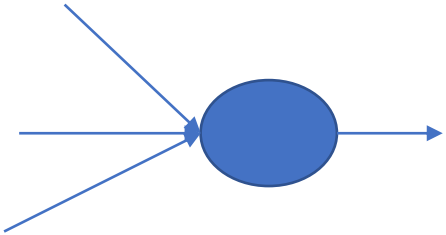
퍼셉트론



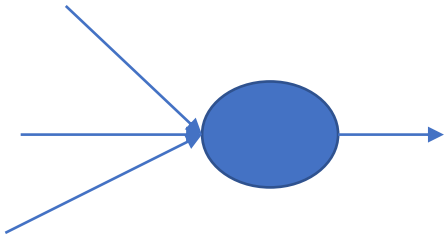
퍼셉트론

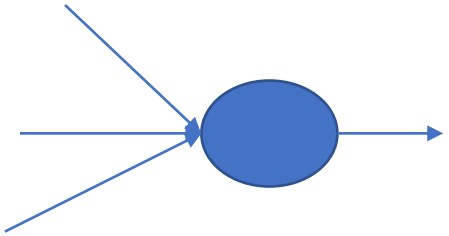


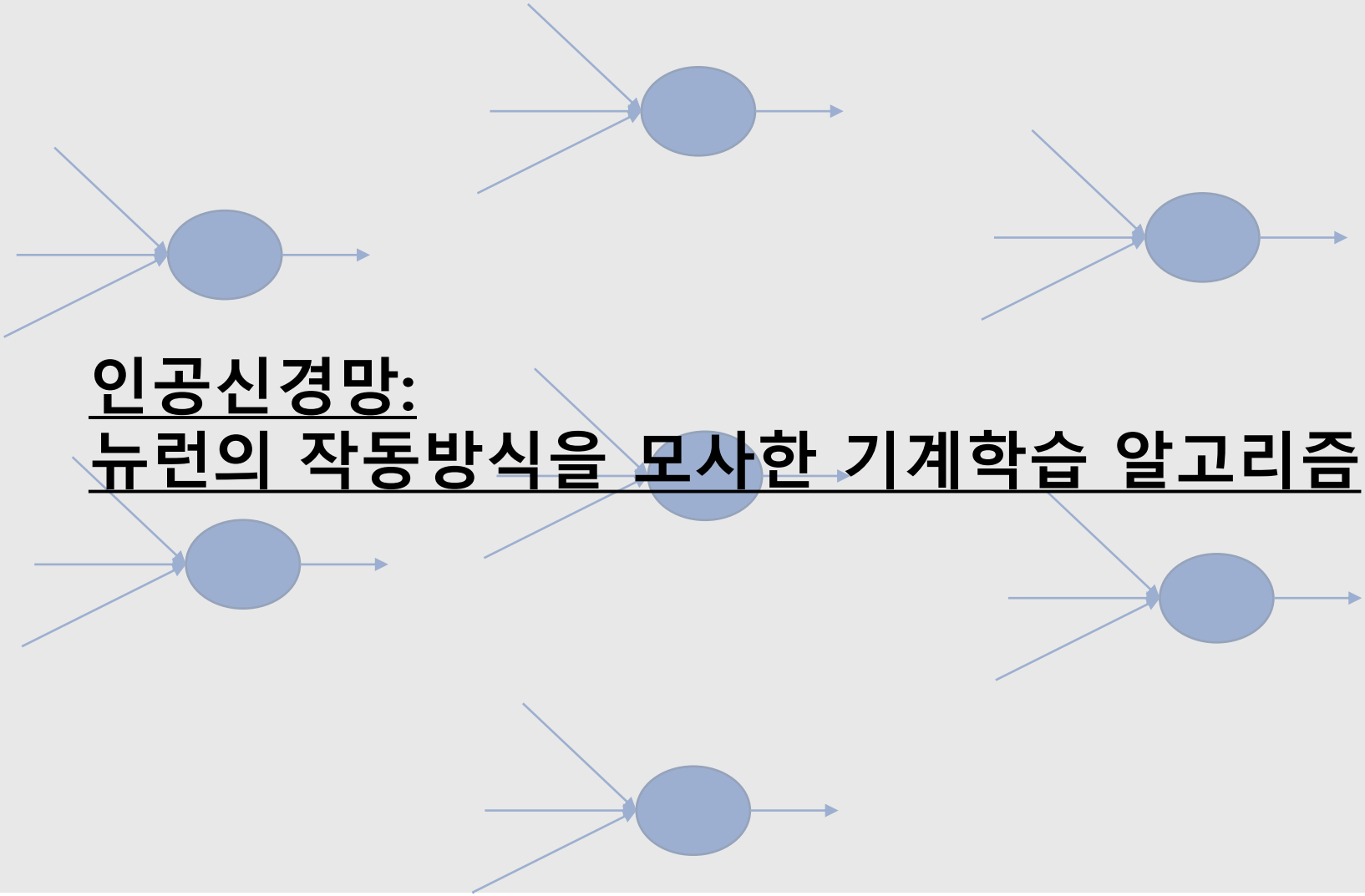
퍼셉트론



퍼셉트론







인공신경망:
뉴런의 작동방식을 모사한 기계학습 알고리즘

https://www.youtube.com/watch?v=vk4V-fl13d8

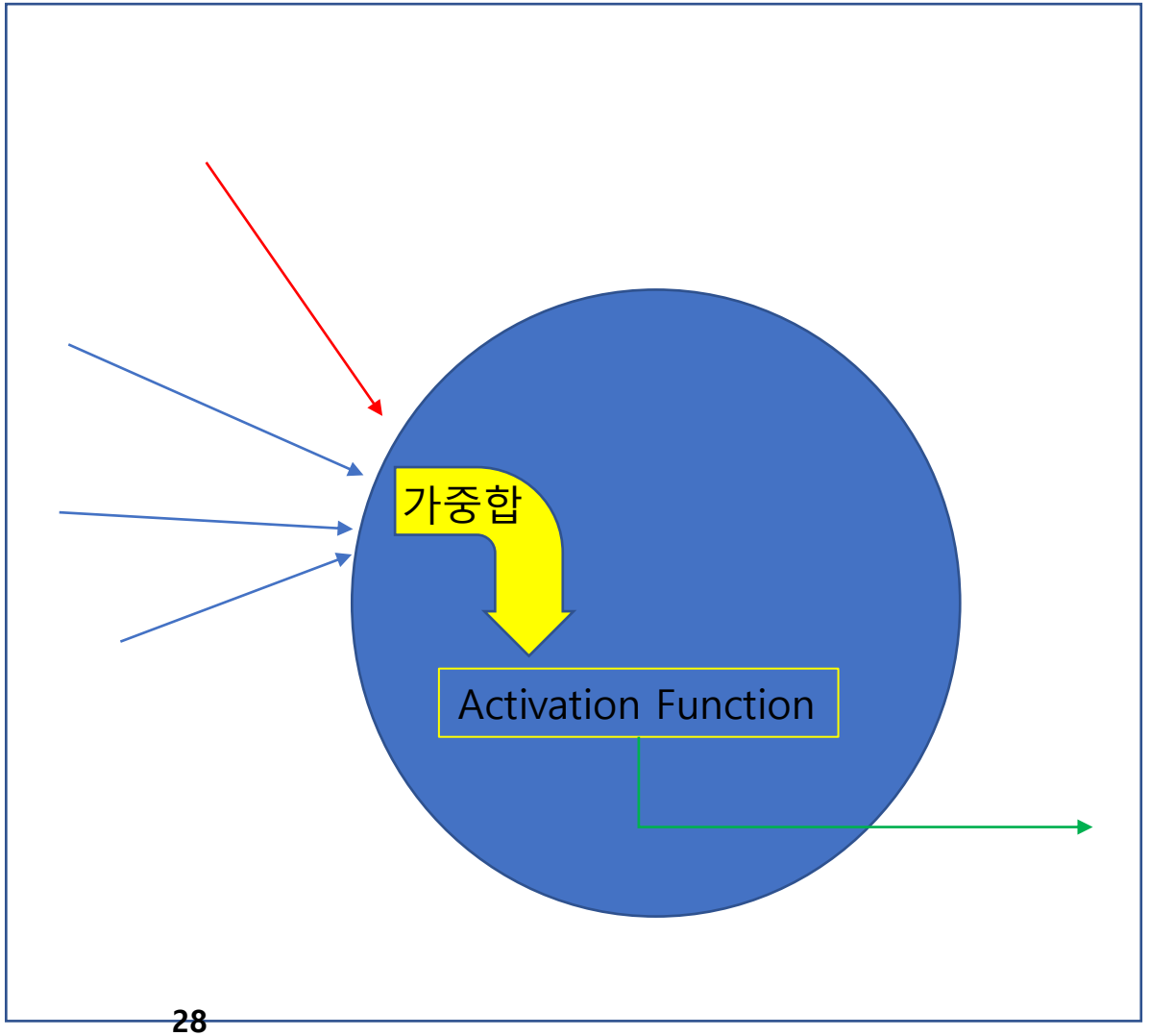
인공신경망 구성 요소

Node

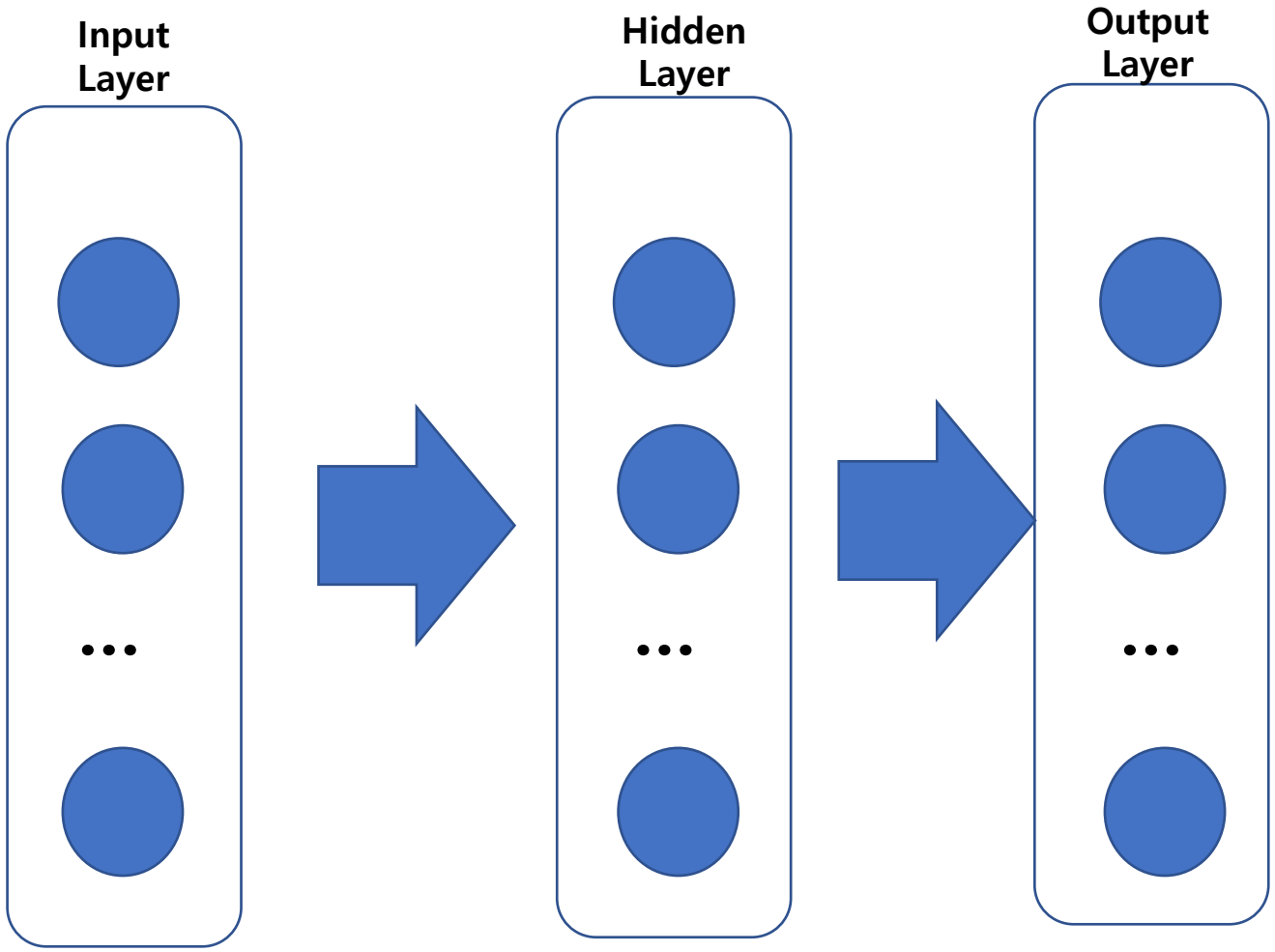
Weight

bias

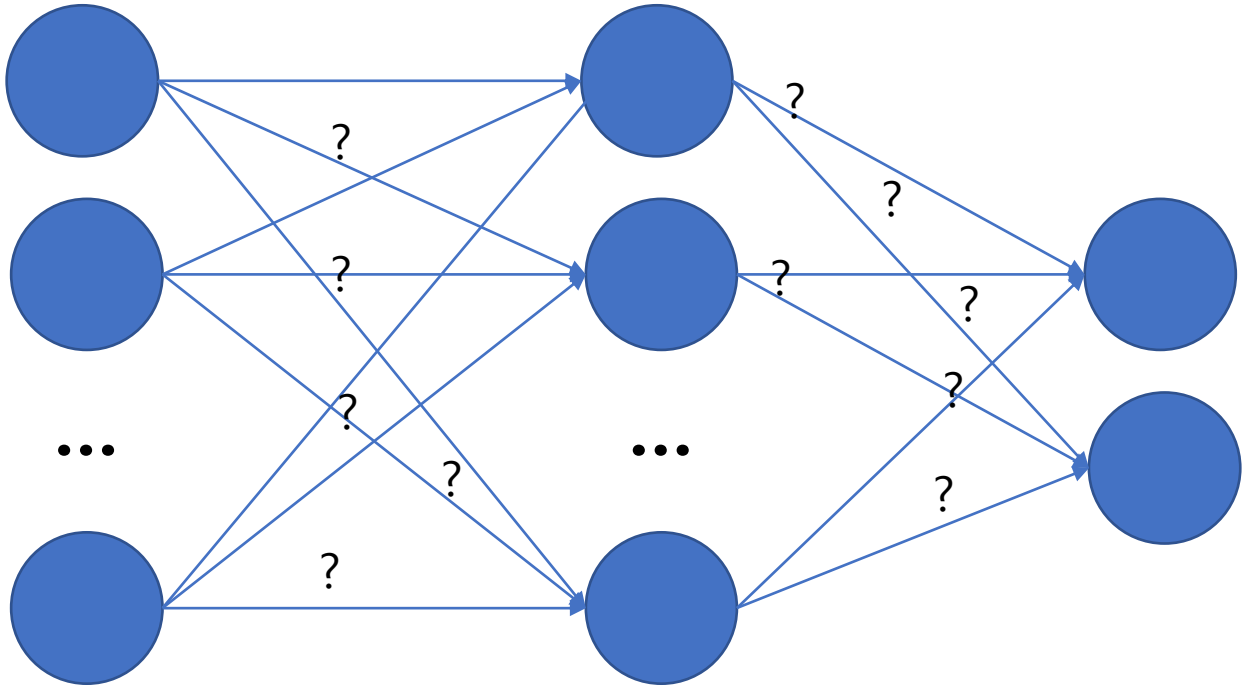
Activation Function 활성화함수



인공신경망 구성 요소



인공신경망: *가중치의 발견*



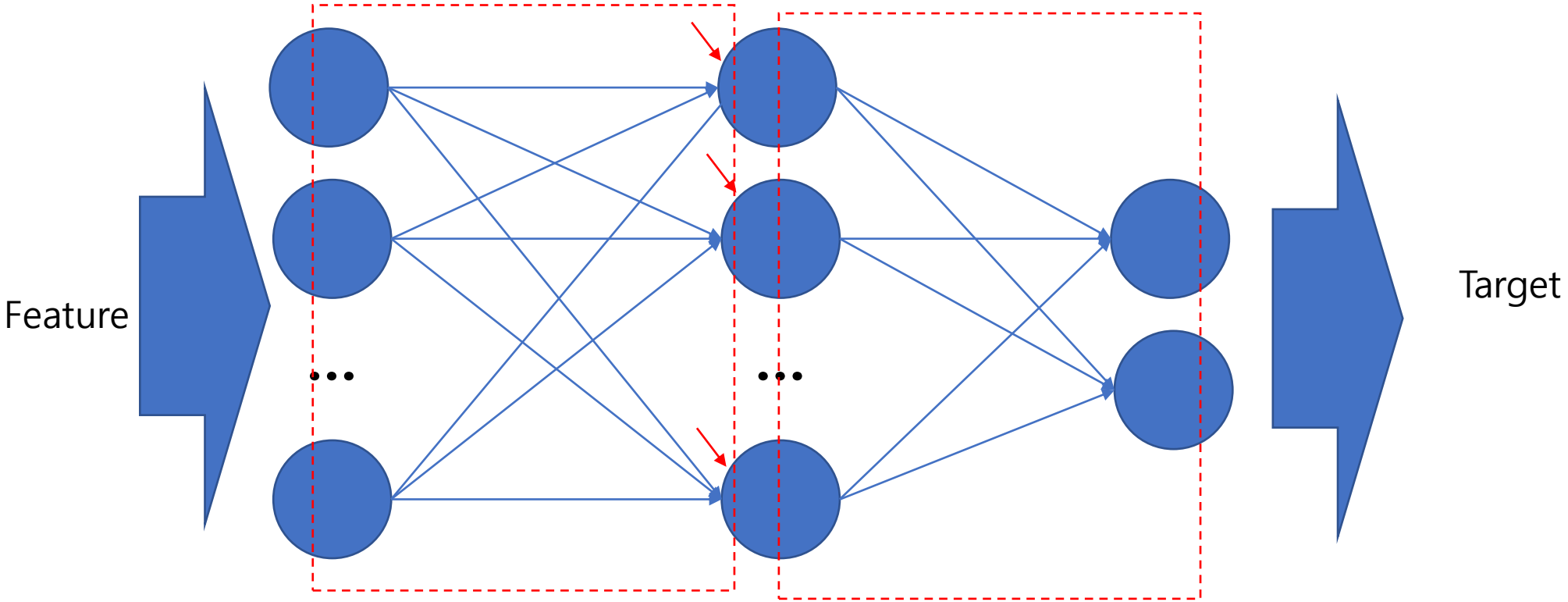
IV. 가중치와 행렬표현

1. 가중치: 원하는 값으로 변환하기 위해 입력값에 곱해지는 수치, Weight라고 하며 값을 강조할 경우 가중치를 증가시킬 수 있음.

2. 인공신경망과 가중치: 인공신경망의 각 층의 노드를 연결하는 것이 바로 가중치이며, 이전 층의 노드값의 가중합이 다음 층의 노드의 입력값이 됨.

3. 인공신경망의 가중치 조절: 오차를 줄이거나 분류를 잘하는 것과 같이 원하는 목적을 달성하기 위해 가중치를 조절할 수 있음.

인공신경망과 가중치

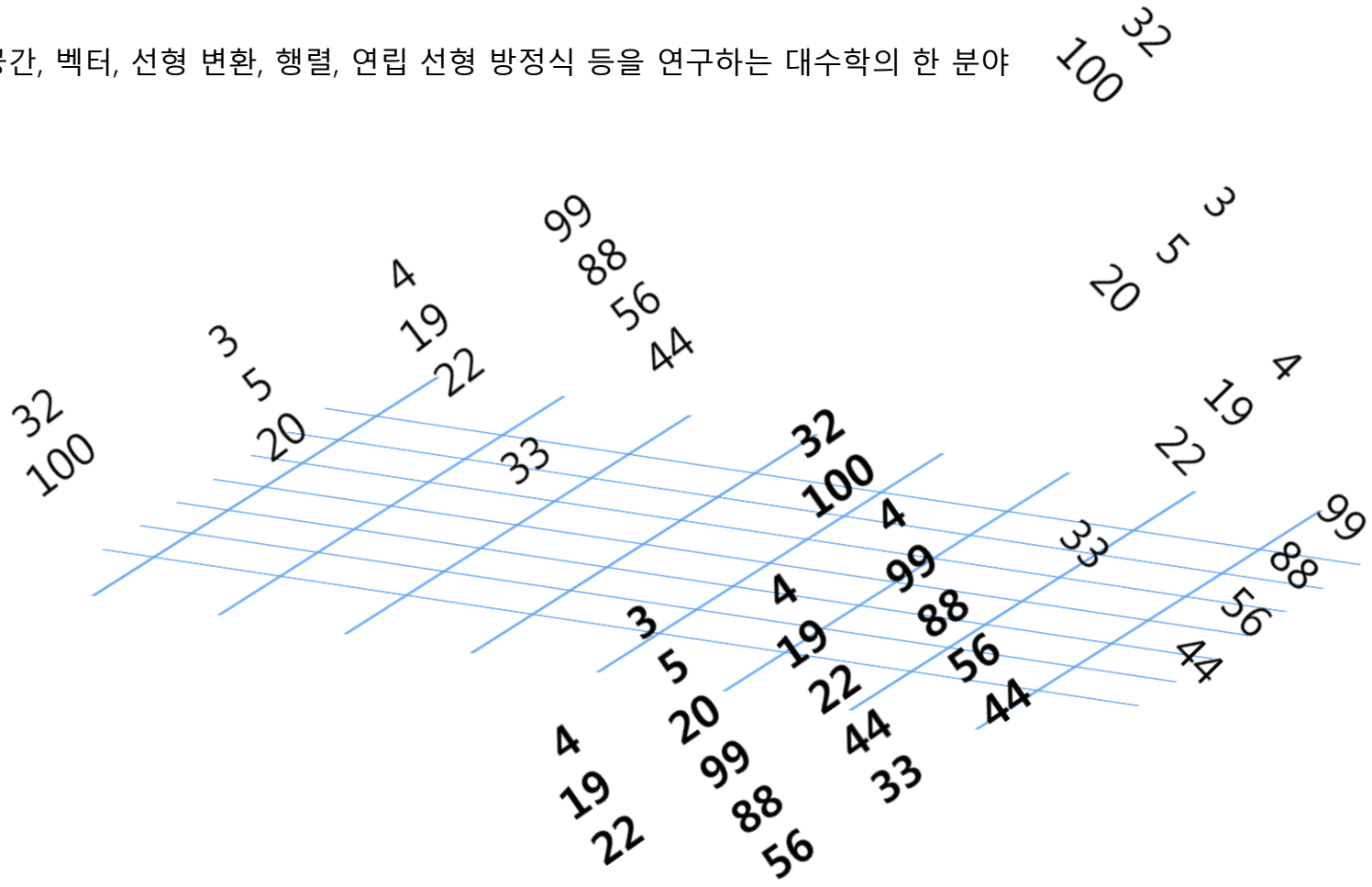


원하는 결과를 얻도록 가중치를 +-로 조정!

가중치와 관련하여 오늘 사용한 계산: 더하기, 곱하기, 크기 늘이기, 크기 줄이기

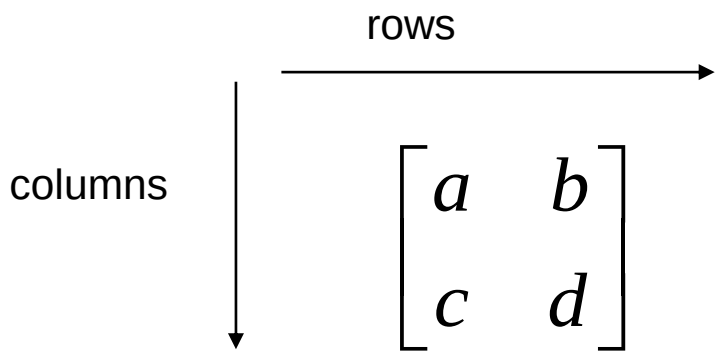
선형대수와 “행렬”

벡터 공간, 벡터, 선형 변환, 행렬, 연립 선형 방정식 등을 연구하는 대수학의 한 분야

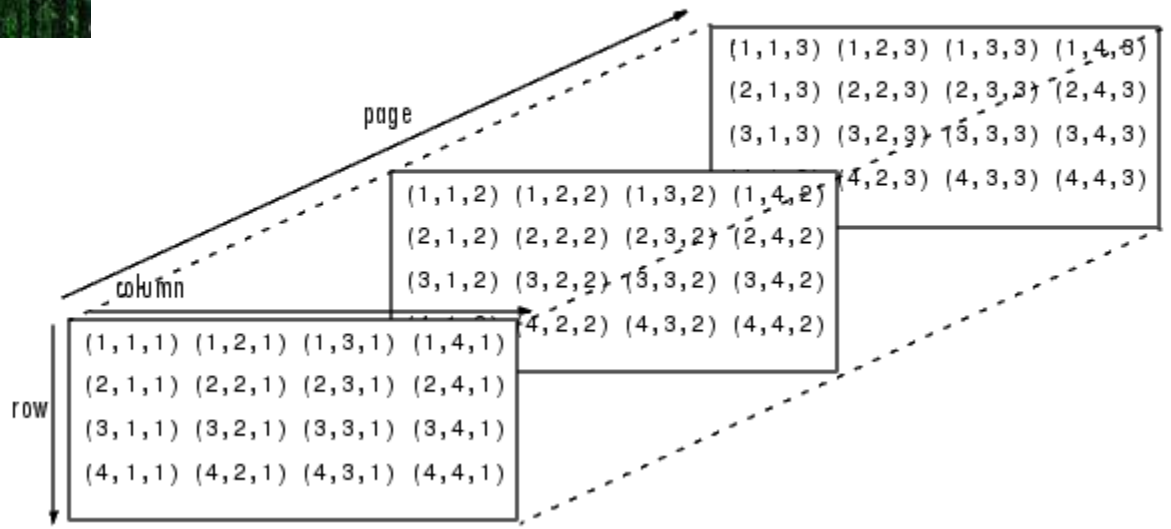


행렬이란: 어떤 값을 행과 열을 갖는 2차원 형태로 배열한 것

- 행렬
  - A matrix is a set of elements, organized into rows and columns
  - 대문자로 표현하며, 각 원소는 Ai,j로 표현



인공신경망을 행렬로 표현하기!



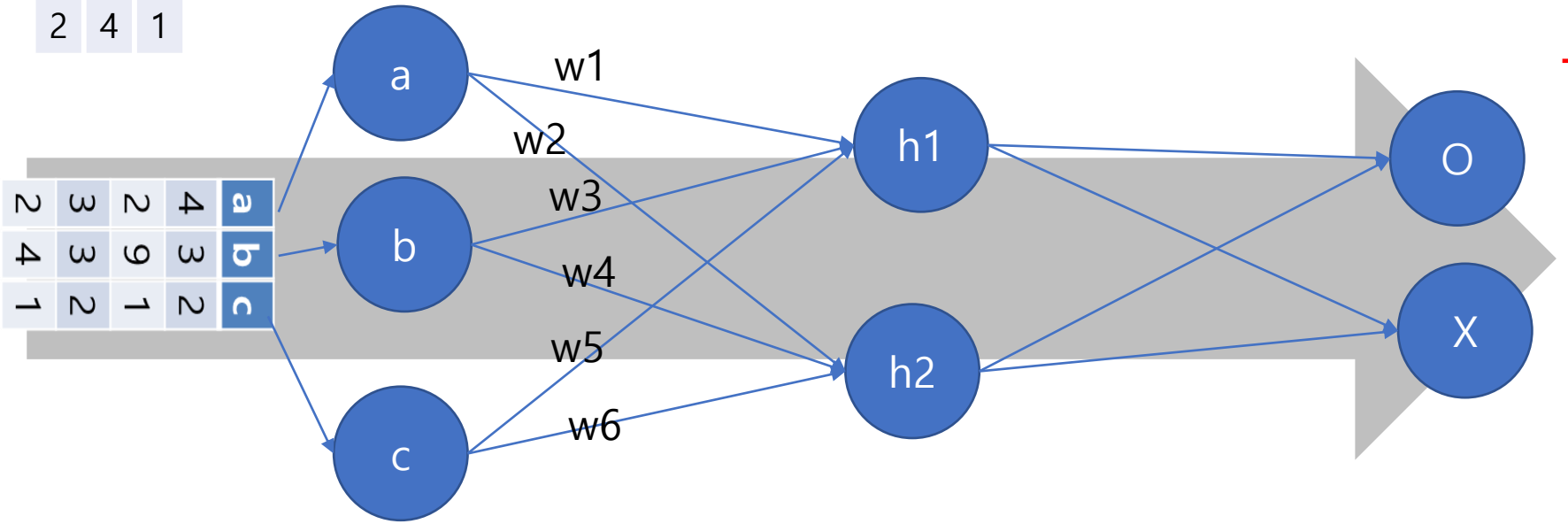
IV. 가중치와 행렬표현

인공신경망을 행렬로 표현하기!

1. *입력데이터*
2. *가중합*
3. *은닉층 표현*

Feature

a	b	c
4	3	2
2	9	1
3	3	2
2	4	1



Target

y
O
X
O
O

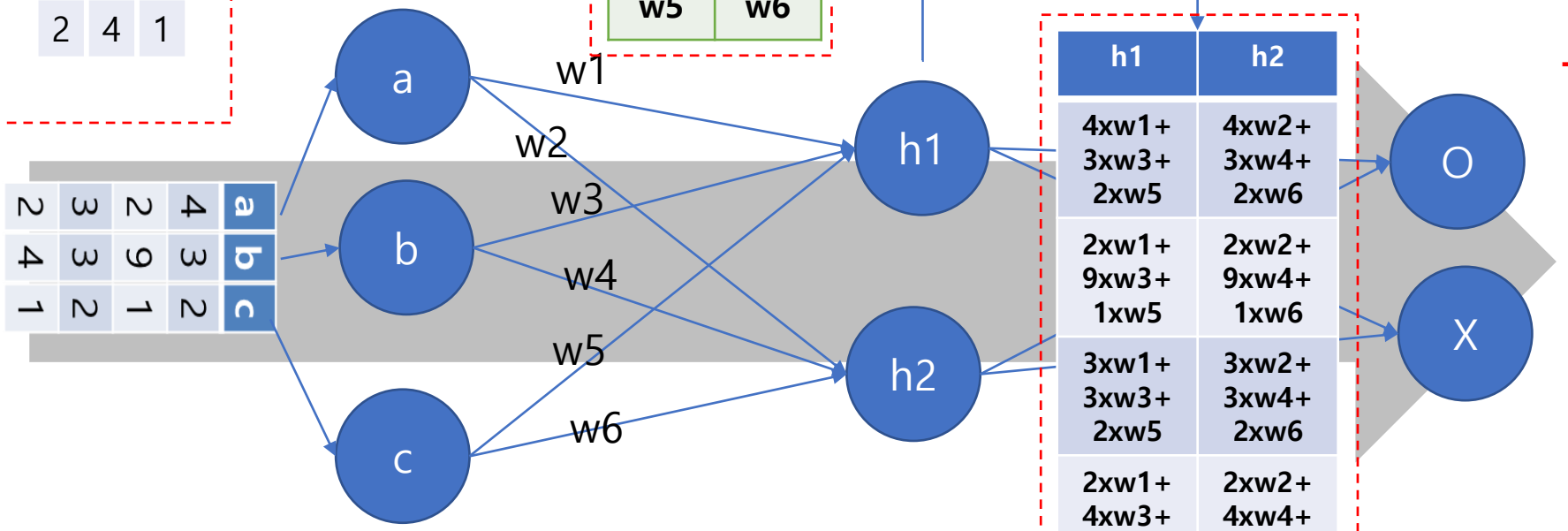
IV. 가중치와 행렬표현

인공신경망을 행렬로 표현하기!

Feature

a	b	c
4	3	2
2	9	1
3	3	2
2	4	1

w1	w2
w3	w4
w5	w6



a	b	c
2	3	2
4	3	9
1	2	1

h1	h2
4xw1+ 3xw3+ 2xw5	4xw2+ 3xw4+ 2xw6
2xw1+ 9xw3+ 1xw5	2xw2+ 9xw4+ 1xw6
3xw1+ 3xw3+ 2xw5	3xw2+ 3xw4+ 2xw6
2xw1+ 4xw3+ 1xw5	2xw2+ 4xw4+ 1xw6

Target

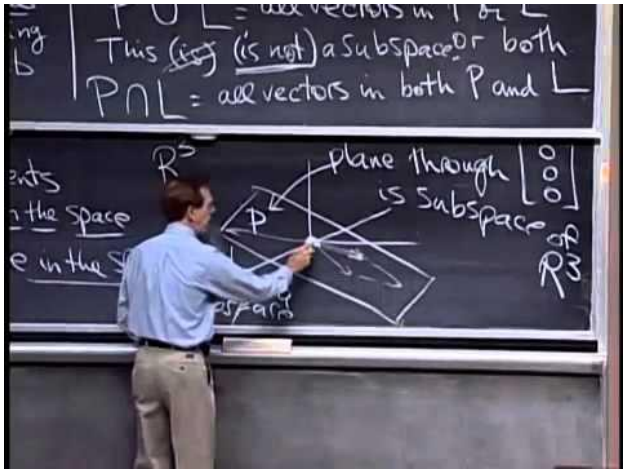
y
O
X
O
O

행렬로 할 수 있는 것들!

- 데이터 표현
  - 많은 데이터가 **행렬**을 통해 자연스럽게 표현
  - 정형 데이터, 비정형 데이터

- 데이터 변환
  - 다른 벡터 공간을 사용한 데이터 표현
  - 좌표계 변환
  - 차원변환 : 차원축소

- 데이터처리
  - 특징추출 - 행렬 분해
  - 수학적 기술의 편의성
  - 명확하고 간결한 표현



Linear Algebra has become as basic and as applicable as calculus, and fortunately it is easier.
Gilbert Strang, MIT

- **행렬 계산!**
  - 합/차/곱

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

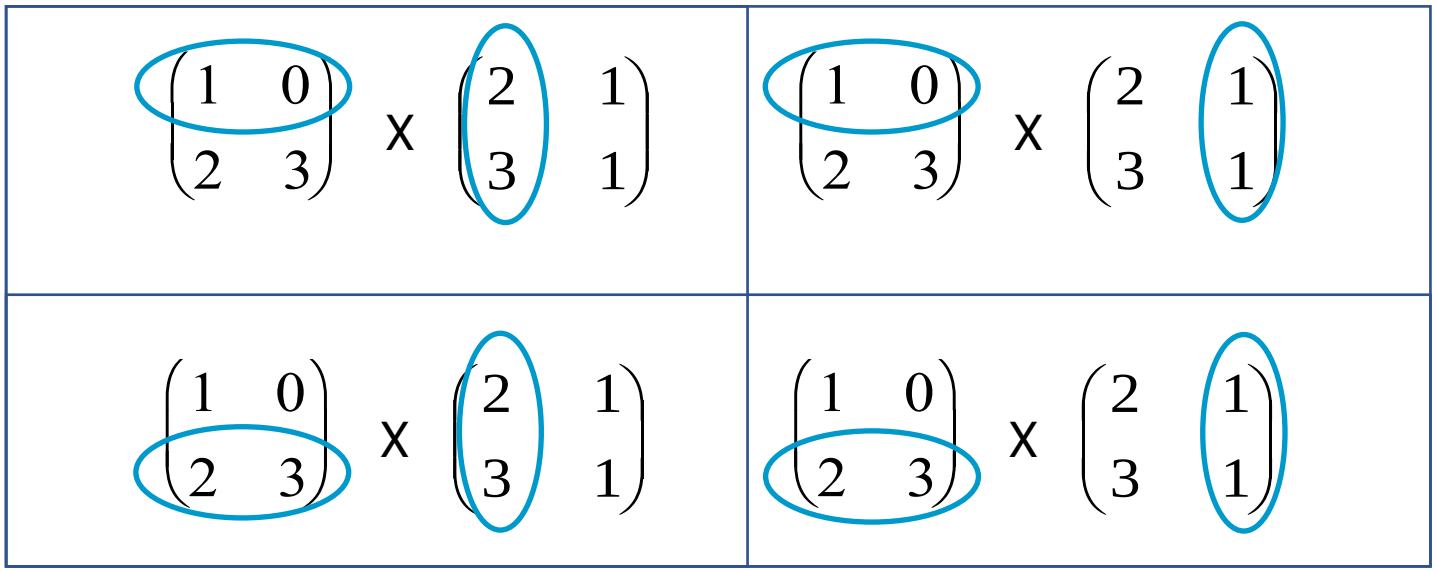
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

IV. 가중치와 행렬표현

- **행렬의 곱**
  - 두 행렬의 곱은 각 행/열의 곱의 합으로 계산

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} 2 & 1 \\ 3 & 1 \end{pmatrix} = \begin{bmatrix} (1\times2)+(0\times3) & (1\times1)+(0\times1) \\ (2\times2)+(3\times3) & (2\times1)+(3\times1) \end{bmatrix} = \begin{pmatrix} \mathbf{2} & \mathbf{1} \\ \mathbf{13} & \mathbf{5} \end{pmatrix}$$

A **B**



행렬곱의 두 행렬 중 앞 행렬의 열의 수가 뒤 행렬의 행의 수와 같아야 함

인공신경망의 가중합

Feature

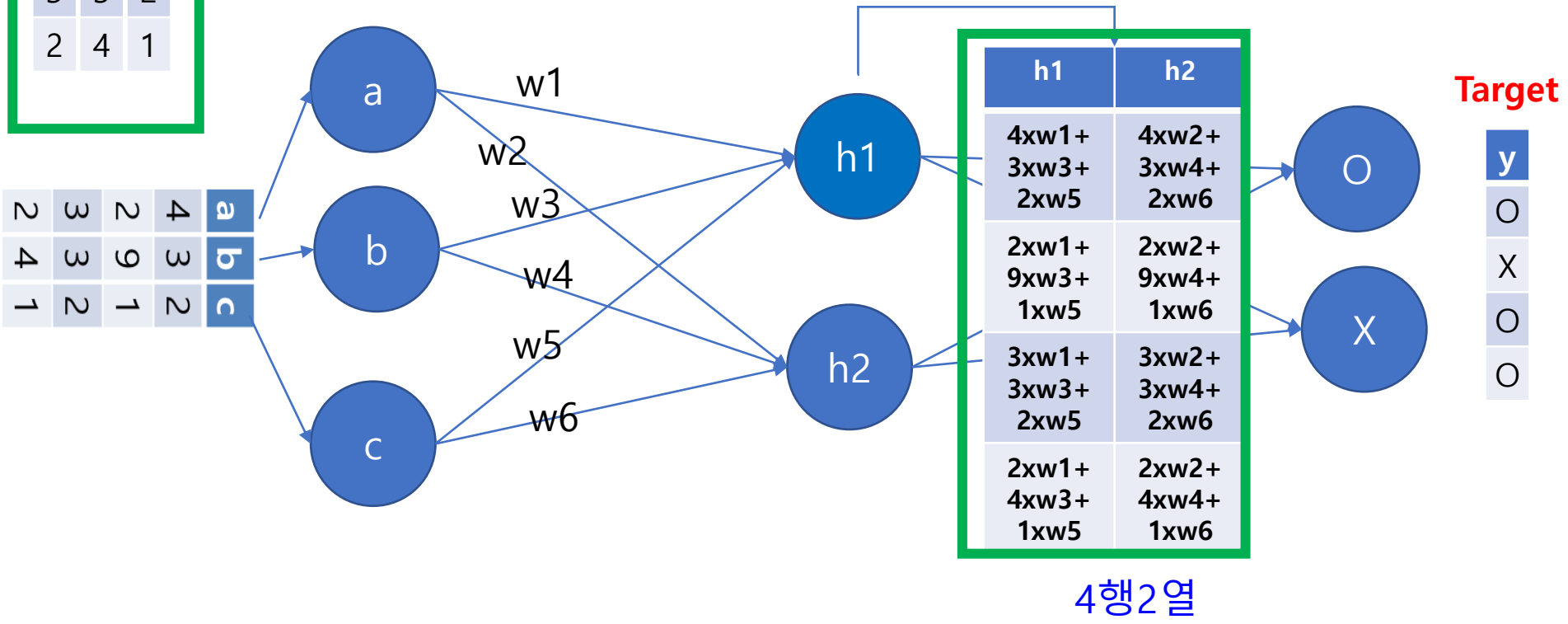
a	b	c
4	3	2
2	9	1
3	3	2
2	4	1

4행3열

3행2열

w1	w2
w3	w4
w5	w6

4Xw1+3Xw3+2Xw5

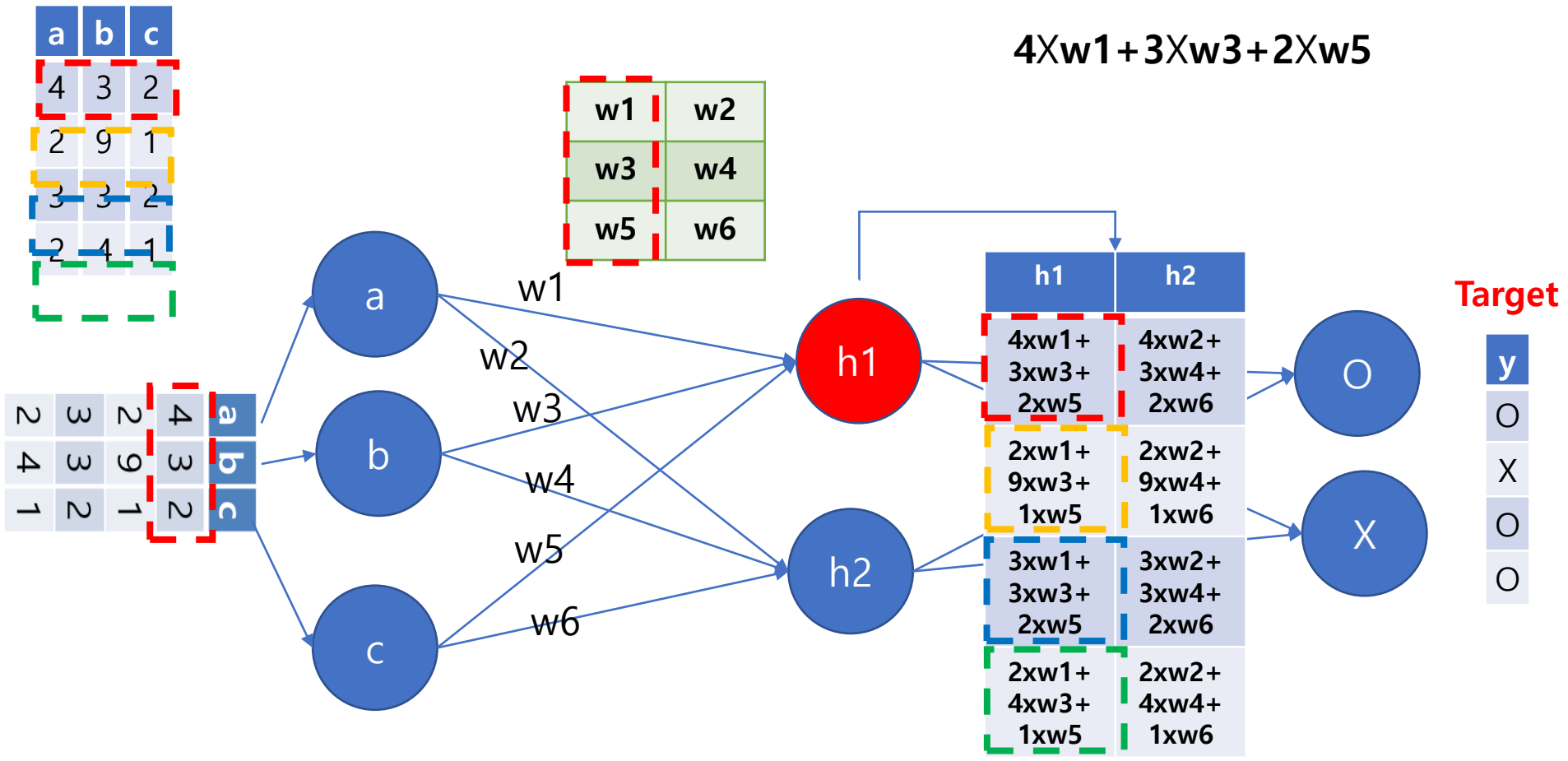


a	b	c
2	3	2
4	3	9
1	2	1

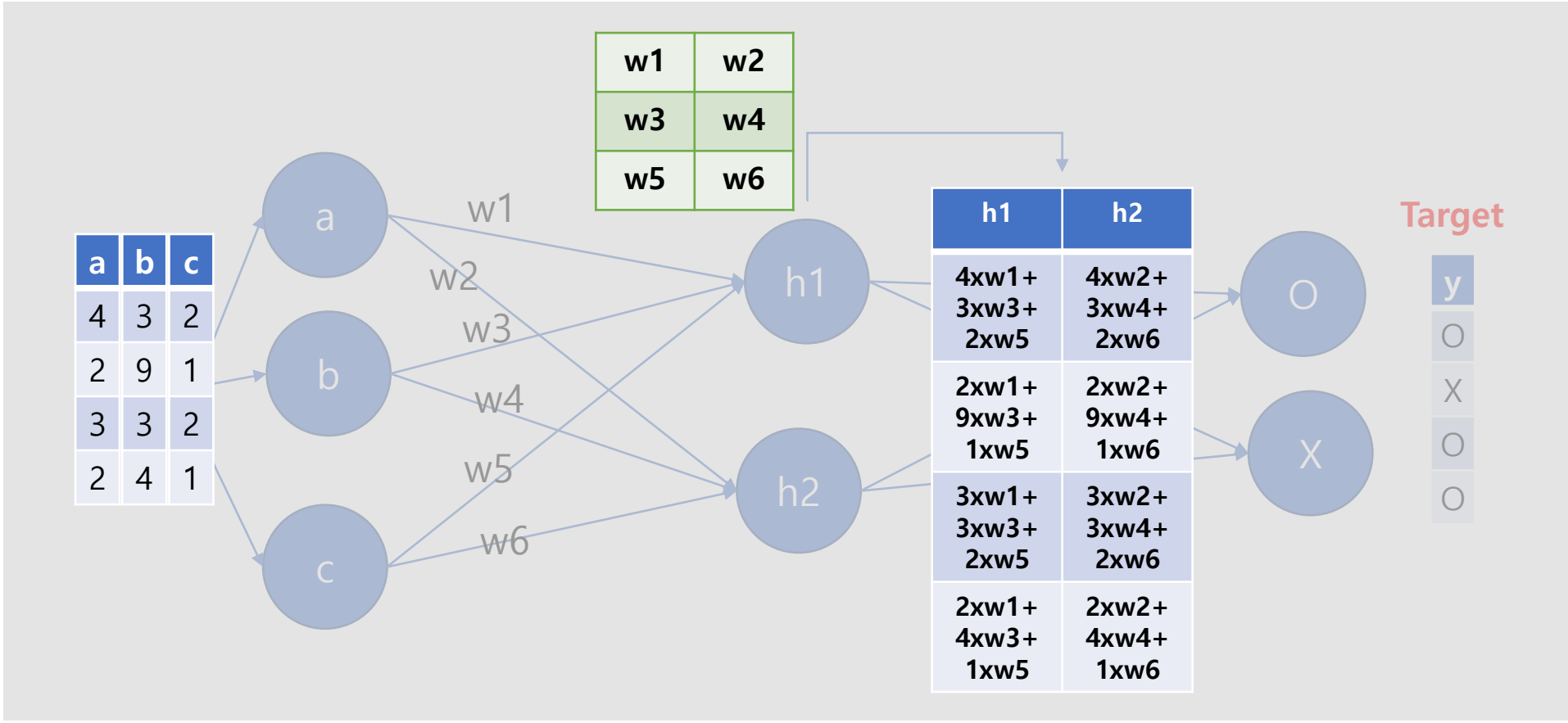
h1	h2
4xw1+ 3xw3+ 2xw5	4xw2+ 3xw4+ 2xw6
2xw1+ 9xw3+ 1xw5	2xw2+ 9xw4+ 1xw6
3xw1+ 3xw3+ 2xw5	3xw2+ 3xw4+ 2xw6
2xw1+ 4xw3+ 1xw5	2xw2+ 4xw4+ 1xw6

인공신경망의 가중합

인공신경망 가중합 다시보기: 인공신경망의 가중합이란 행렬곱 결과의 한 원소에 대한 계산과 동일



인공신경망의 행렬곱



a	b	c
4	3	2
2	9	1
3	3	2
2	4	1

w1	w2
w3	w4
w5	w6

h1	h2
4xw1+ 3xw3+ 2xw5	4xw2+ 3xw4+ 2xw6
2xw1+ 9xw3+ 1xw5	2xw2+ 9xw4+ 1xw6
3xw1+ 3xw3+ 2xw5	3xw2+ 3xw4+ 2xw6
2xw1+ 4xw3+ 1xw5	2xw2+ 4xw4+ 1xw6

인공신경망의 행렬곱

인공신경망의 Feature와 가중치들은 모두 행렬 곱을 통해 계산됨

4행3열

a	b	c
4	3	2
2	9	1
3	3	2
2	4	1

X

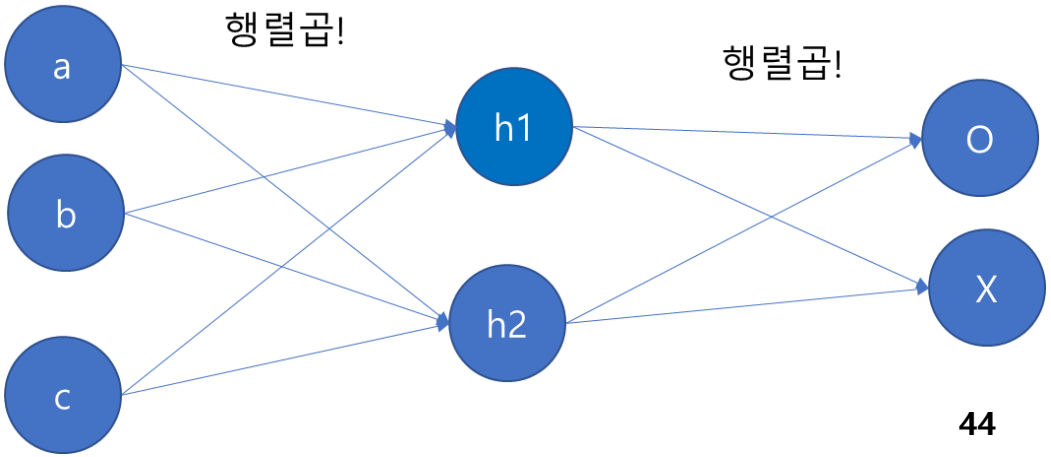
3행2열

w1	w2
w3	w4
w5	w6

=

4행2열

h1	h2
4xw1+ 3xw3+ 2xw5	4xw2+ 3xw4+ 2xw6
2xw1+ 9xw3+ 1xw5	2xw2+ 9xw4+ 1xw6
3xw1+ 3xw3+ 2xw5	3xw2+ 3xw4+ 2xw6
2xw1+ 4xw3+ 1xw5	2xw2+ 4xw4+ 1xw6



V. 활성화함수와 가중치 업데이트

1. 활성화함수: 은닉층의 노드에서 입력 신호에 대해 적절한 처리를 하여 출력해주는 함수

2. 다양한 활성화함수: Step, Sigmoid, Linear, ReLU 등 다양한 형태의 활성화 함수가 있음

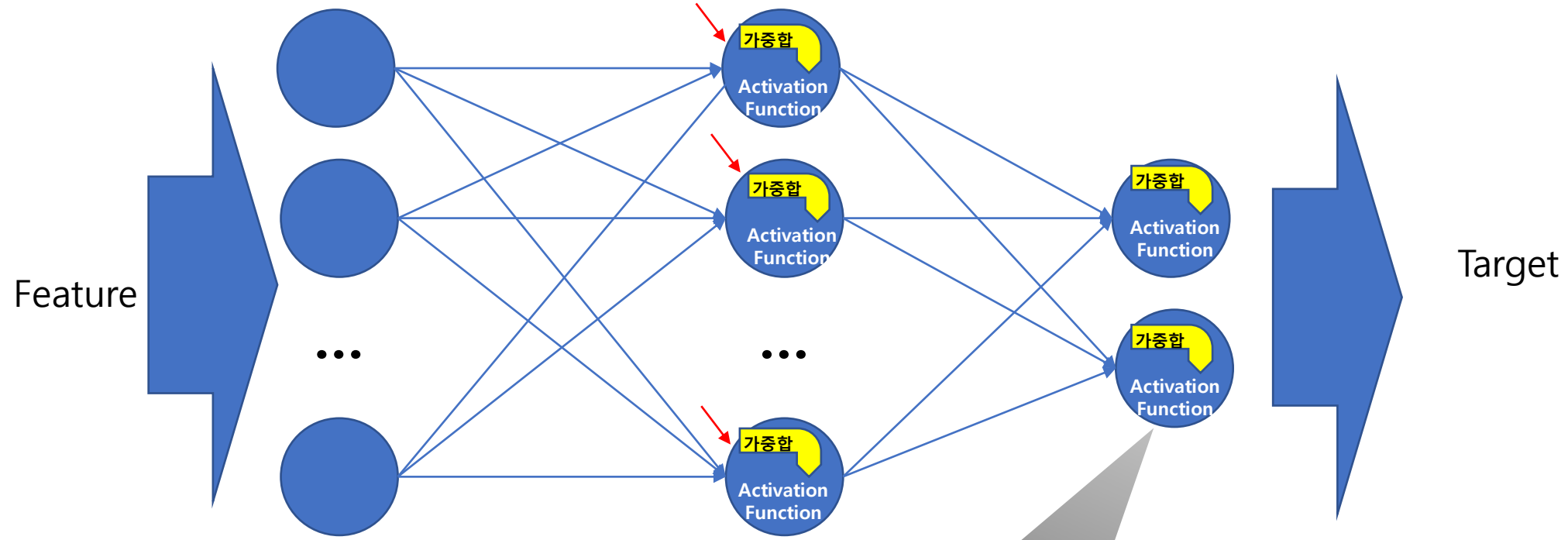
3. Feedforward 신경망: 노드 간의 연결에서 순환이나 루프가 없는 기본적인 인공신경망

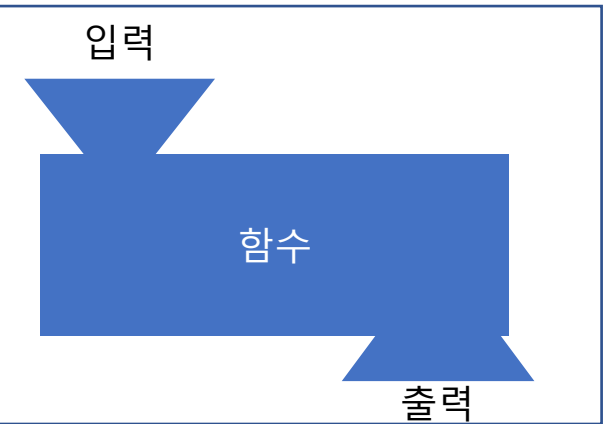
4. 가중치 업데이트: 랜덤하게 주어진 가중치부터 시작하여 인공신경망이 좋은 성능을 보이도록 최적화된 가중치를 찾는 과정

5. Epoch: 인공신경망에서 순전파(Forward Propagation)와 역전파(Back Propagation)를 마친 것을 의미

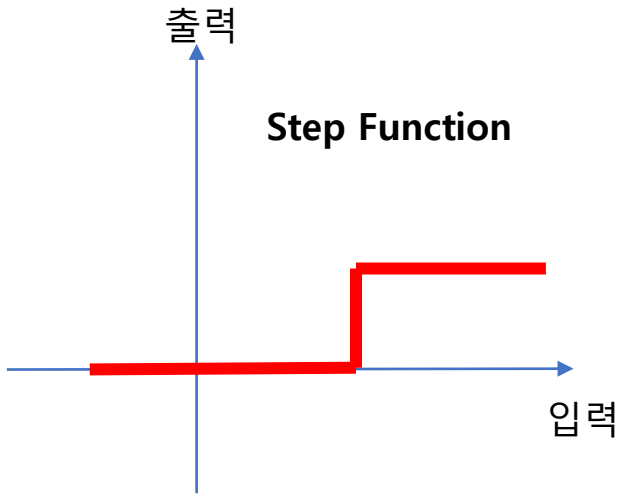
V. 활성화함수와 가중치 업데이트

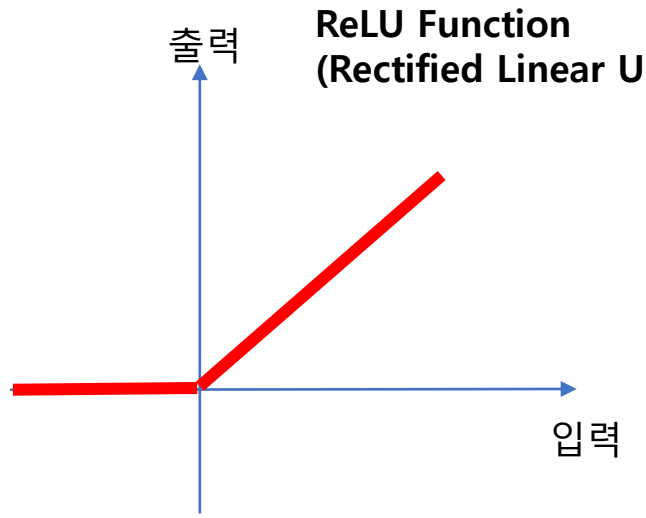
활성화 함수

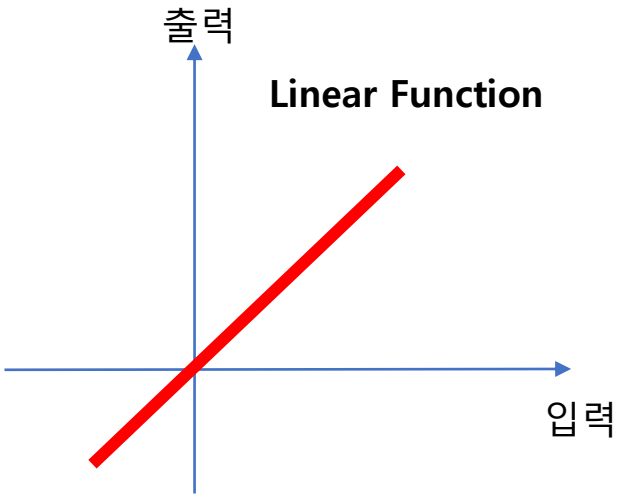


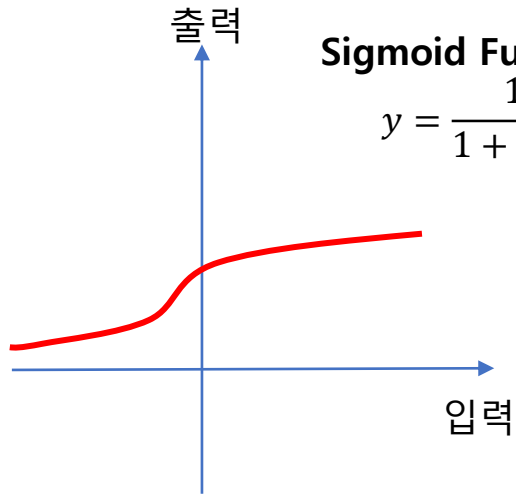


다양한 활성화 함수


출력
Step Function
입력


출력
ReLU Function
(Rectified Linear Unit)
입력


출력
Linear Function
입력


출력
Sigmoid Function
$y = \frac{1}{1+e^{-x}}$
입력

Feedforward Neural Network

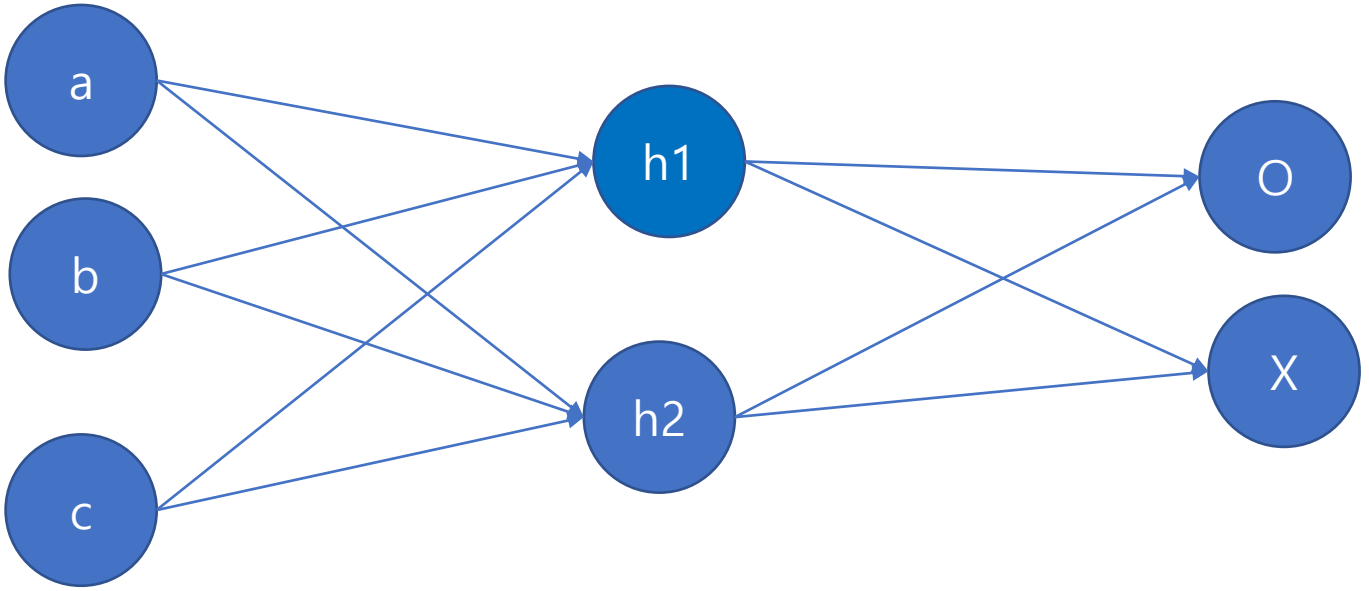
- 노드 간의 연결에서 순환이나 루프가 없는 기본적인 인공신경망

- Feedforward = 순방향

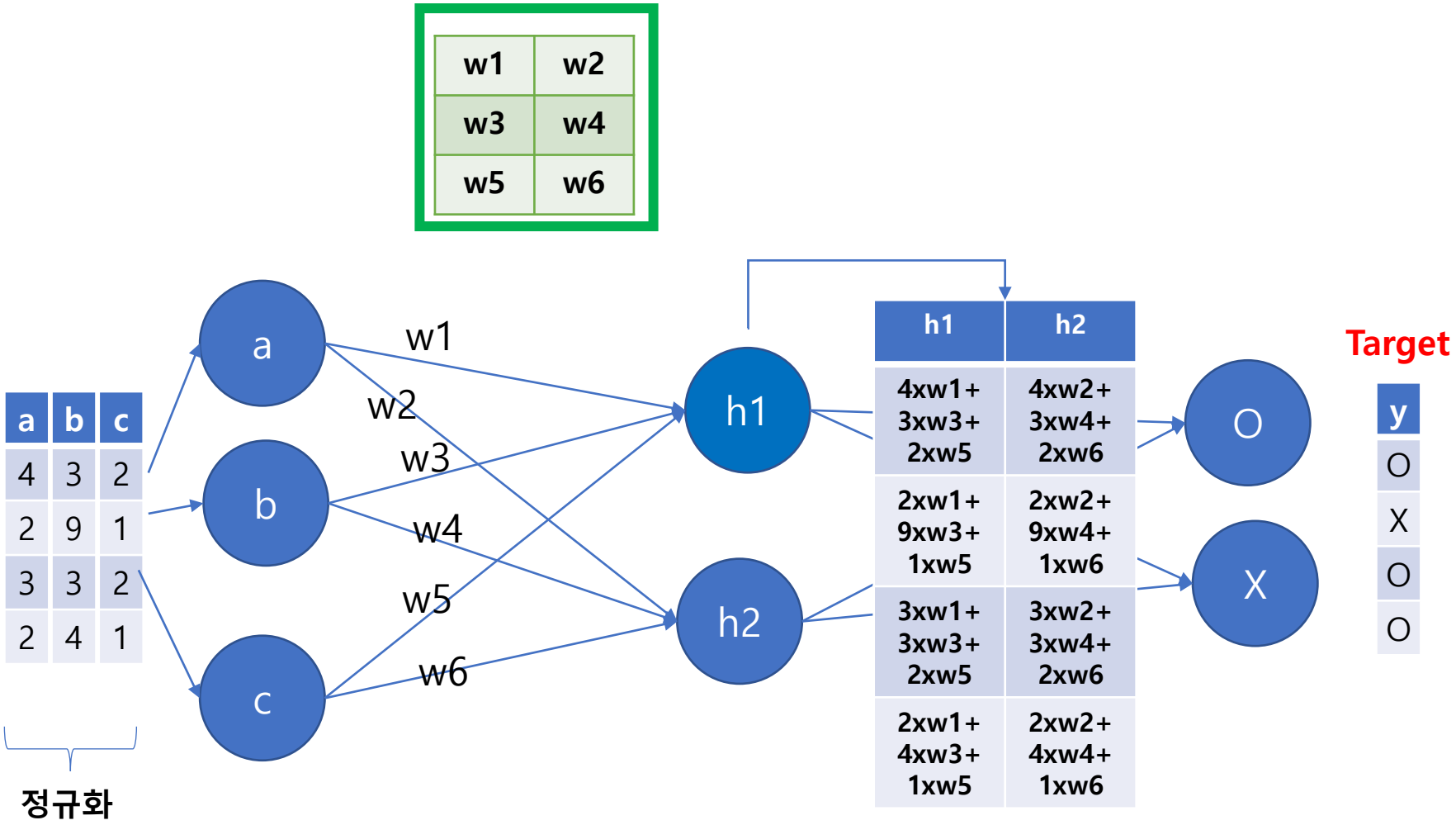
- Forward Propagation? Back Propagation?

Forward Propagation = Matrix Multiplication

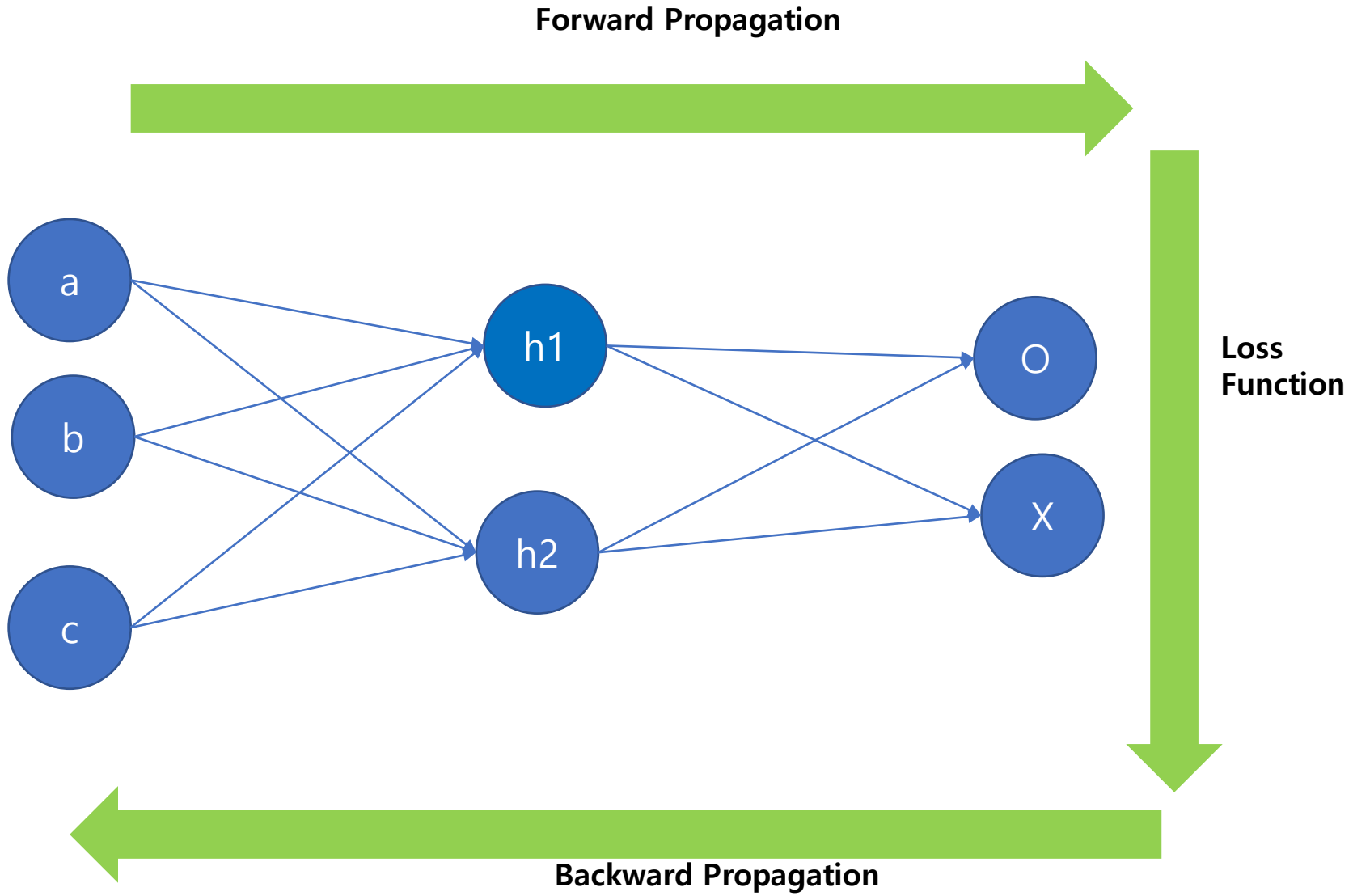
가중치!



가중치: 처음엔 Random (0~1)



Epoch(에포크): Forward Propagation + Back Propagation



VI. 역전파 알고리즘

1. 인공신경망의 오차: 인공신경망의 순전파 과정을 통해 계산된 예측값과 실제값과의 차이값

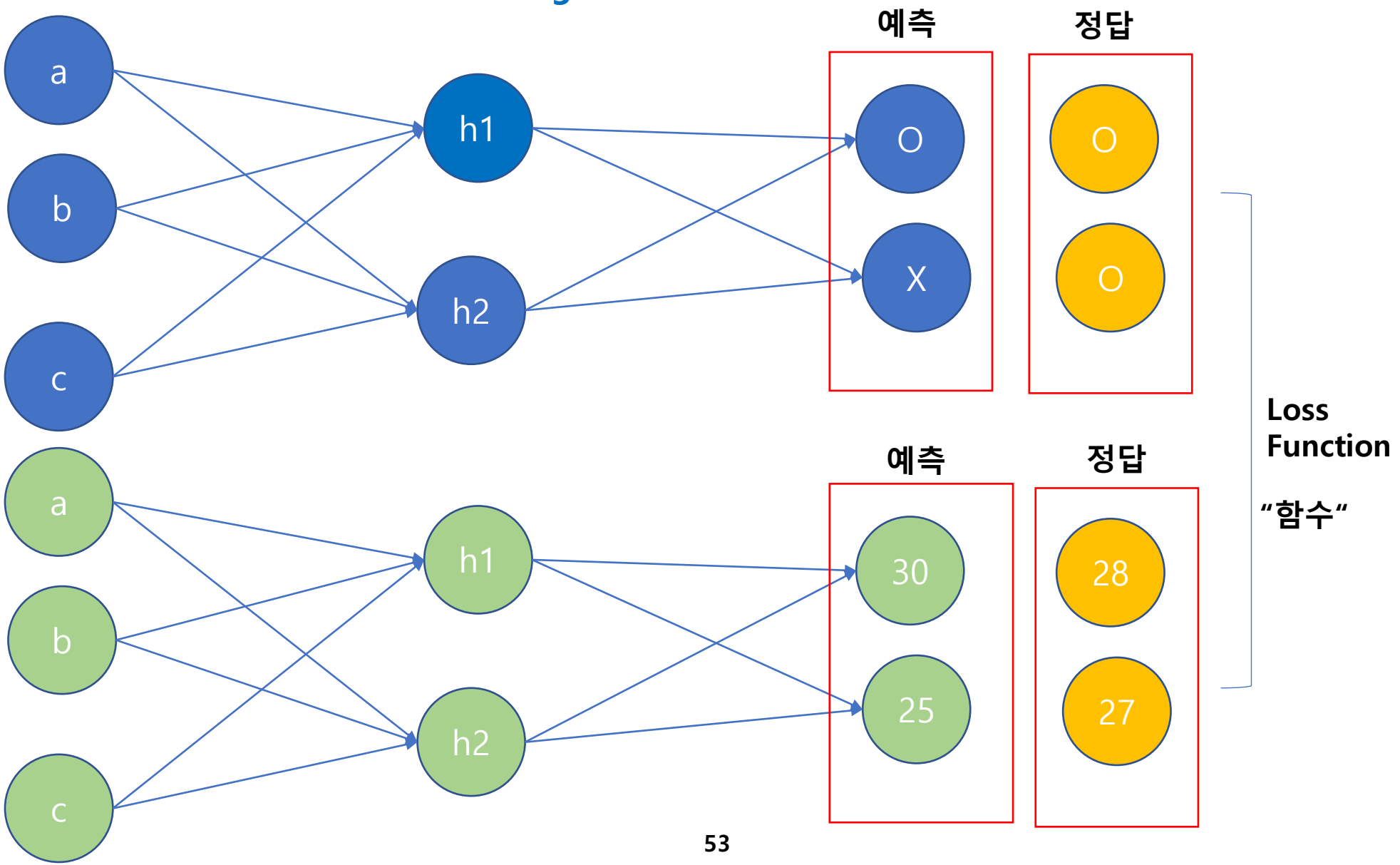
2. 역전파알고리즘: 오차를 활용하여 가중치를 업데이트하는 알고리즘

3. 경사하강법 : 가중치에 대한 오차 함수가 최소가 되는 지점을 발견하여 가중치를 최적화하는 방법

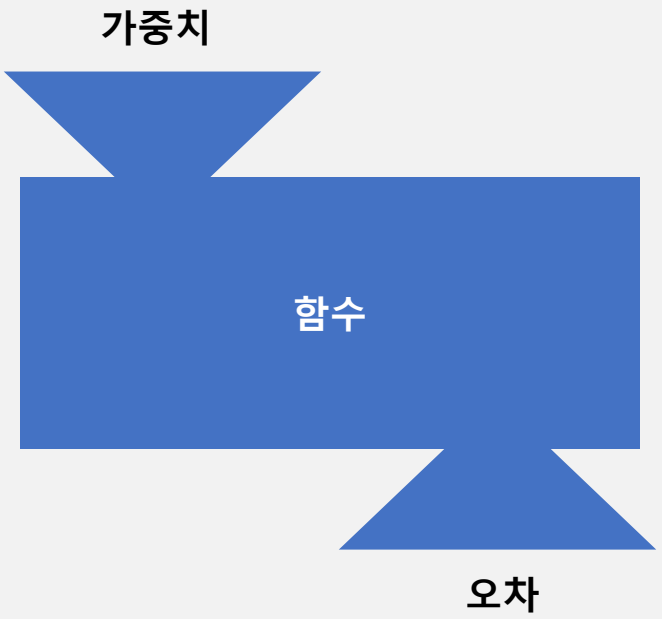
4. 학습율: 한 번의 에포크에서 얼마나 학습되는지를 알려주며, 이 값이 너무 크거나 작으면 최적의 가중치를 발견하기가 어려움

VI. 역전파 알고리즘

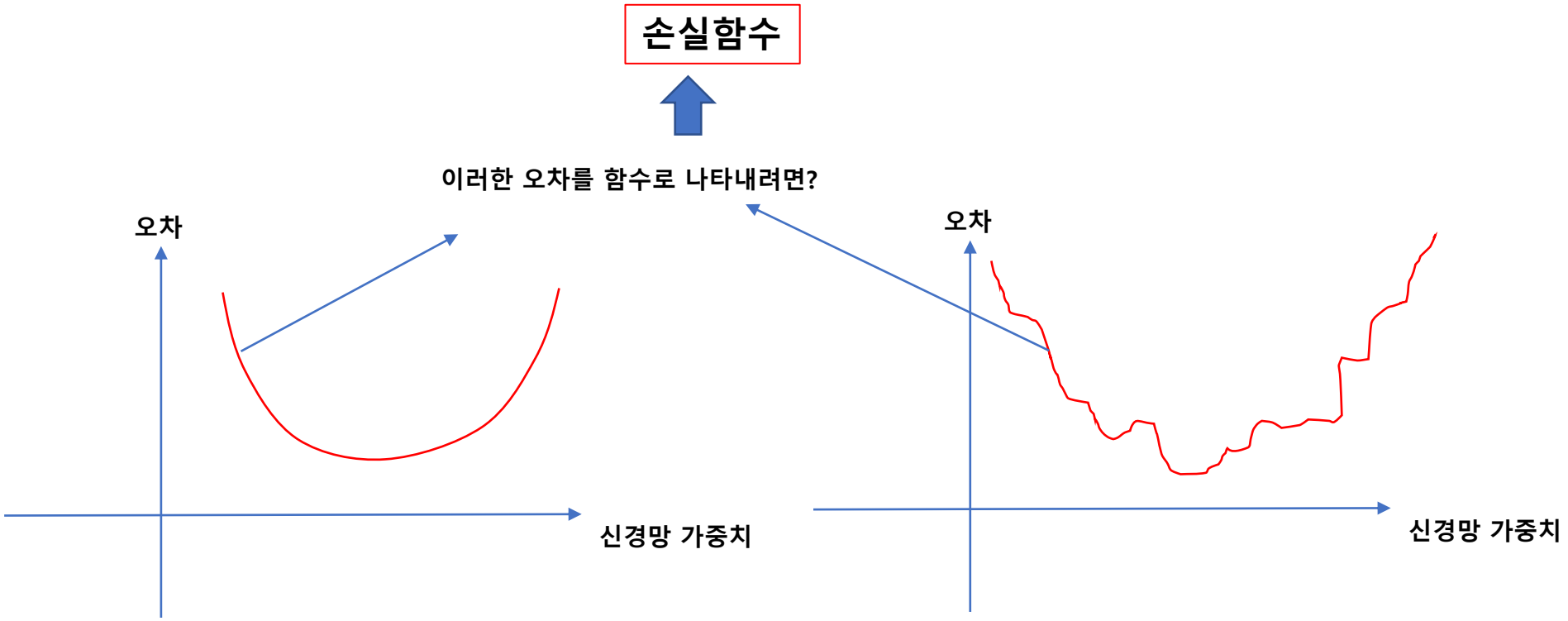
모형의 오차: Classification VS Regression



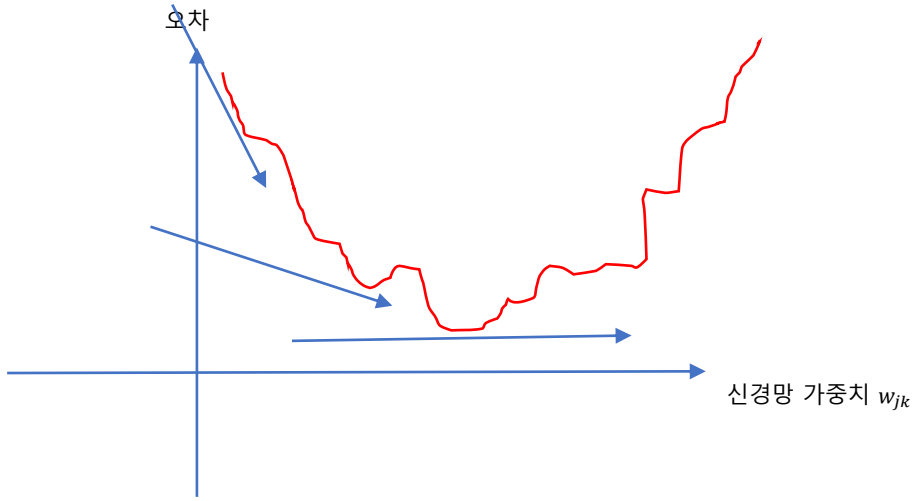
가중치와 오차의 함수



오차에 대한 함수



VI. 역전파 알고리즘



오차를 구하는 함수

$E = (\text{실제값} - \text{오차})^2$ → 기울기가 작아지거나 0이 되는 지점의 가중치

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_n (\text{목표값}_n - \text{출력값}_n)^2$$

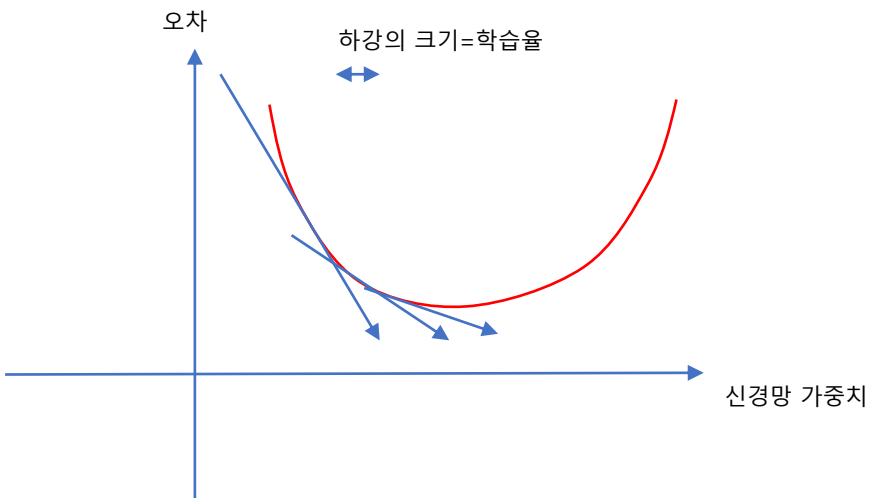
출력층 직전 은닉층이 n개의 노드
j: 은닉층의 노드 수
k: 결과층의 노드 수

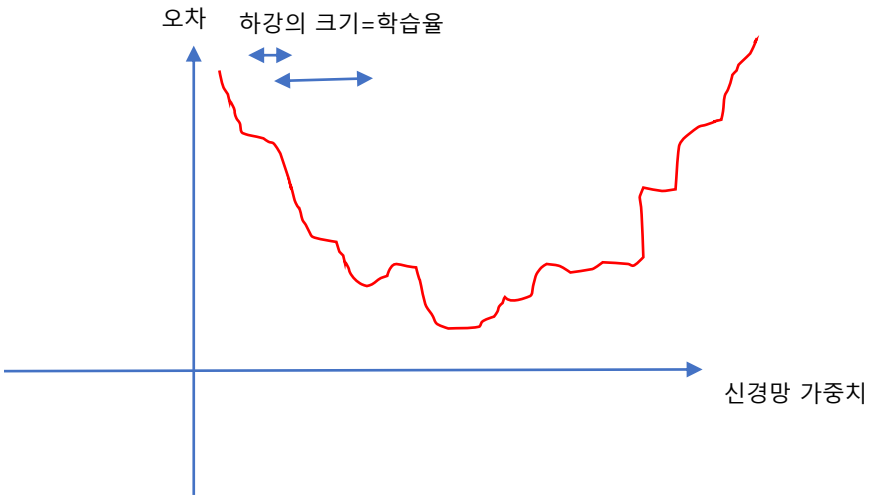
$$\frac{\partial E}{\partial w_{jk}} = \text{오차 함수를 미분한 기울기}$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) \times sigmoid\left(\sum_j w_{jk} o_j\right) \left(1 - sigmoid\left(\sum_j w_{jk} o_j\right)\right) \times o_j$$

VI. 역전파 알고리즘

- 신경망 가중치가 입력이고 오차가 출력인 함수 대상
- 경사하강법: 오차의 최소화
- 학습율: 하강의 크기





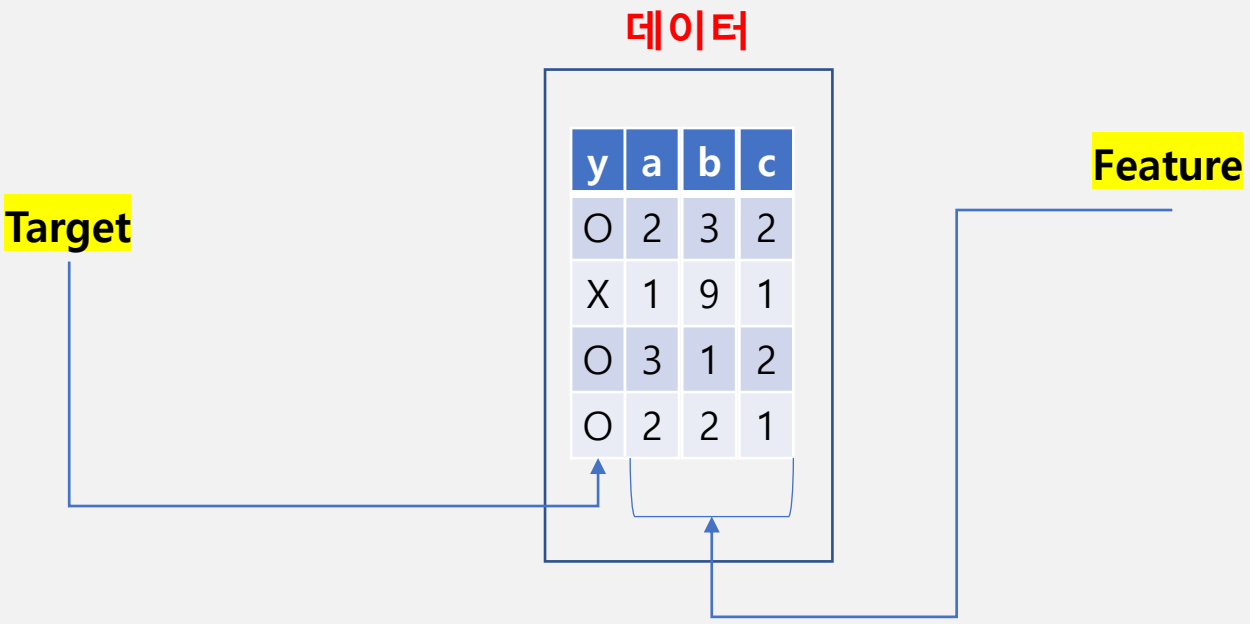
학습률(Learning Rate)

가중치 조정과 학습율

새로운 w_{jk} = 이전 $w_{jk} - \alpha \times \frac{\partial E}{\partial w_{jk}}$

- 이전 가중치에서 오차의 변화율을 빼주기
  - 양의 기울기(오차 증가시킴)이면 이전 가중치에서 빼주어 영향을 덜 받게 하고, 음의 기울기(오차 감소)이면 이전 가중치를 더 크게 해주는 효과(-의-)
- α는 가중치 변화하는 정도를 조정:학습률

ANN Step by Step



Supervised Learning-Classification

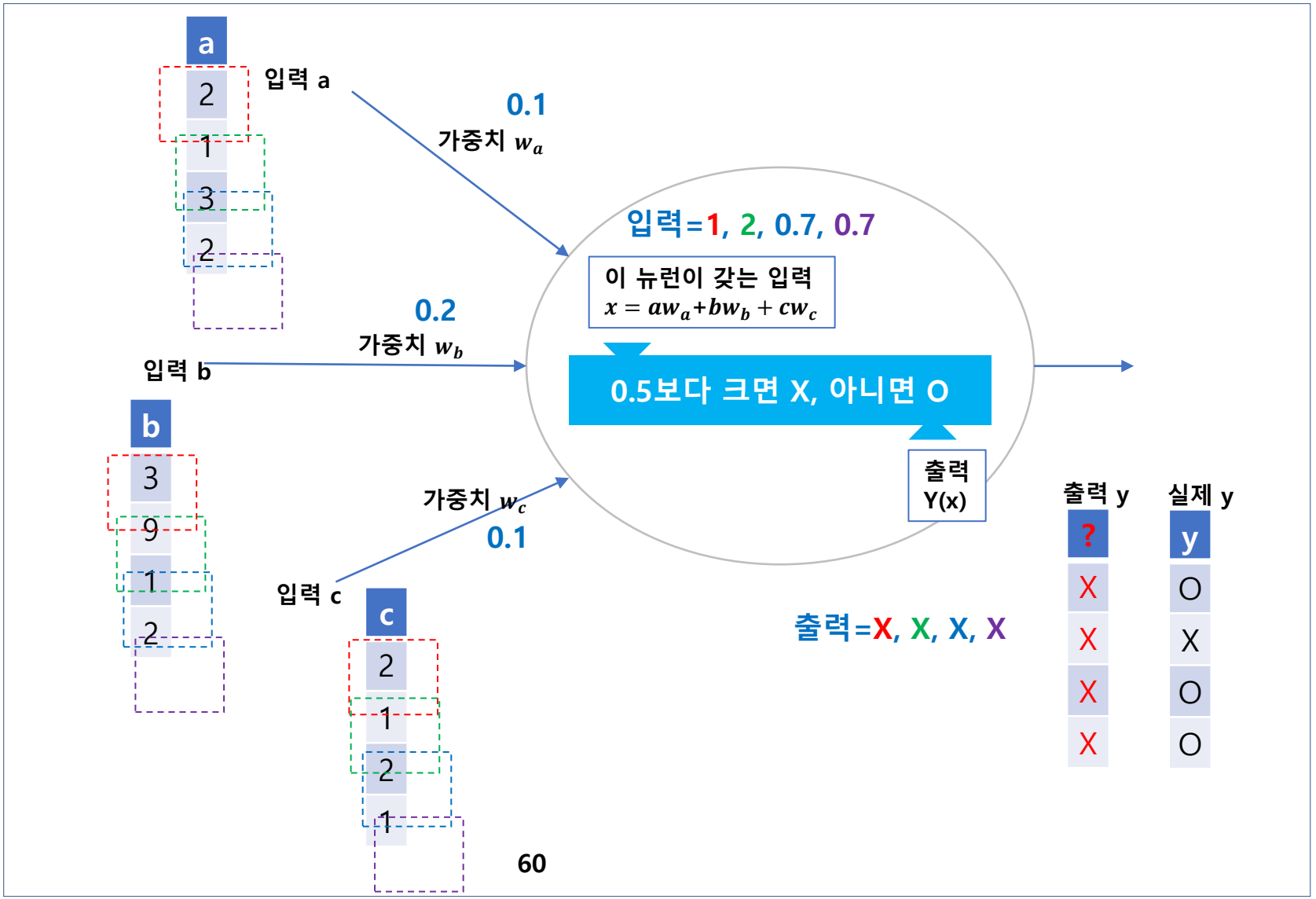
VI. 역전파 알고리즘

ANN Step by Step

인공신경망 1단계: 전파!

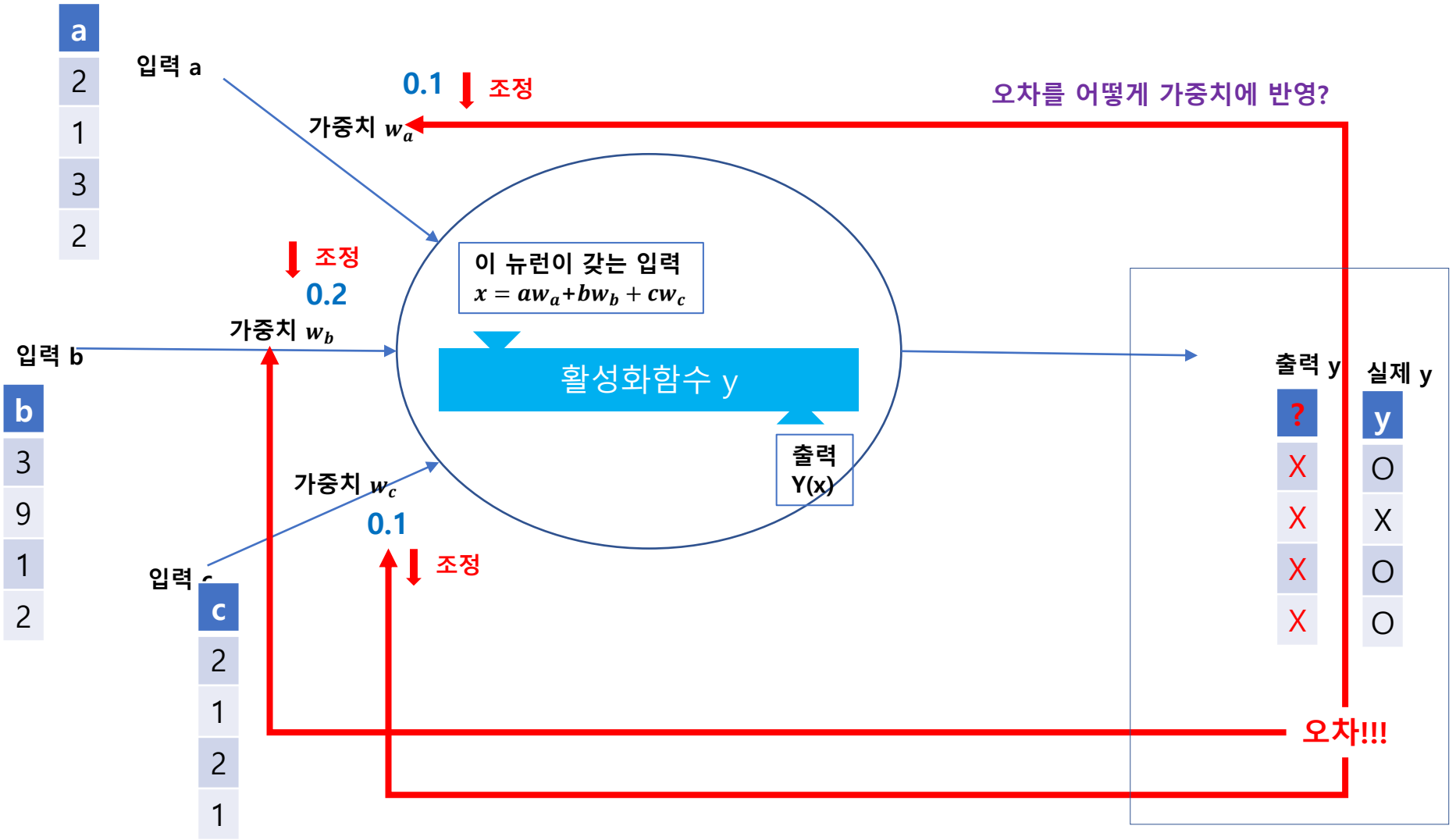
데이터

y	a	b	c
O	2	3	2
X	1	9	1
O	3	1	2
O	2	2	1



ANN Step by Step

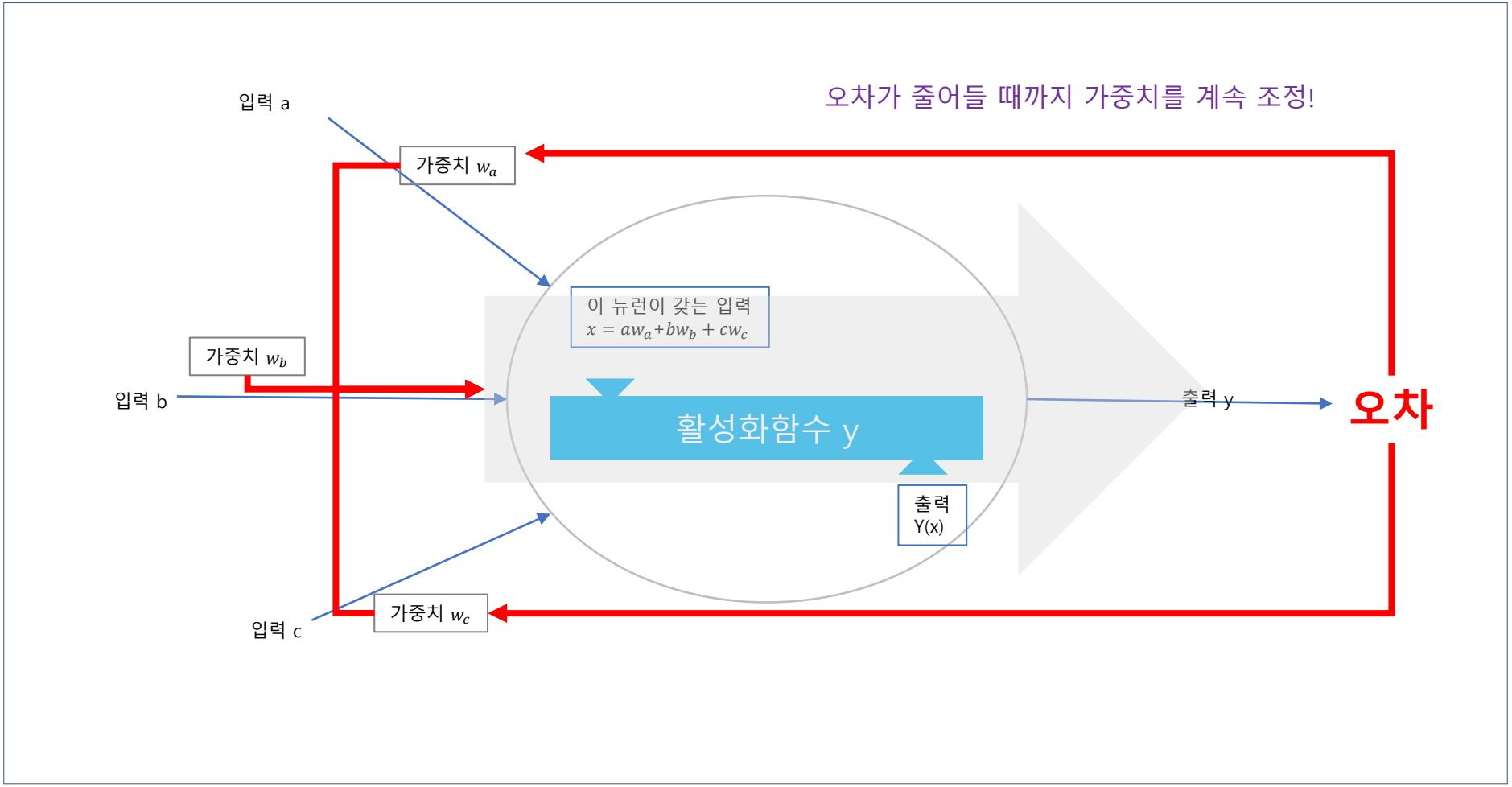
인공신경망 2단계: 오차의 역전파!



VI. 역전파 알고리즘

ANN Step by Step

잘 할 때까지 반복!



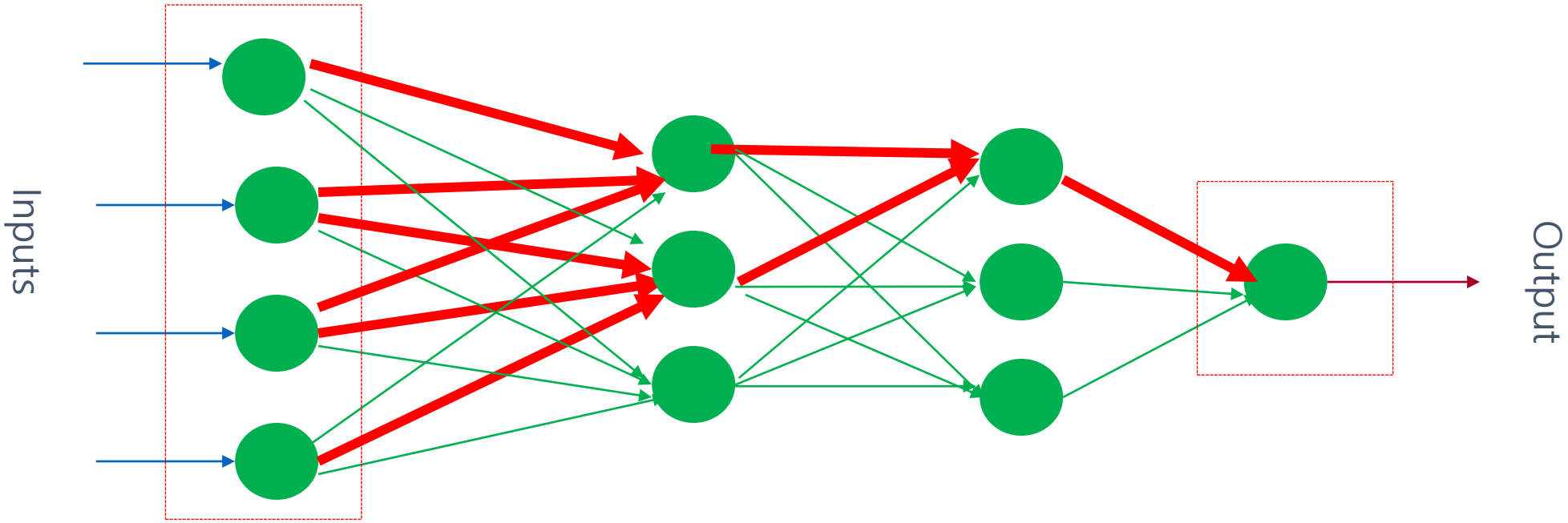
VII. ANN의 이슈와 성능 개선

1. ANN 난제1-Overfitting과 해석의 어려움: 주어진 데이터에 적합한 ANN의 가중치들로 인한 Overfitting, 가중치 해석의 어려움

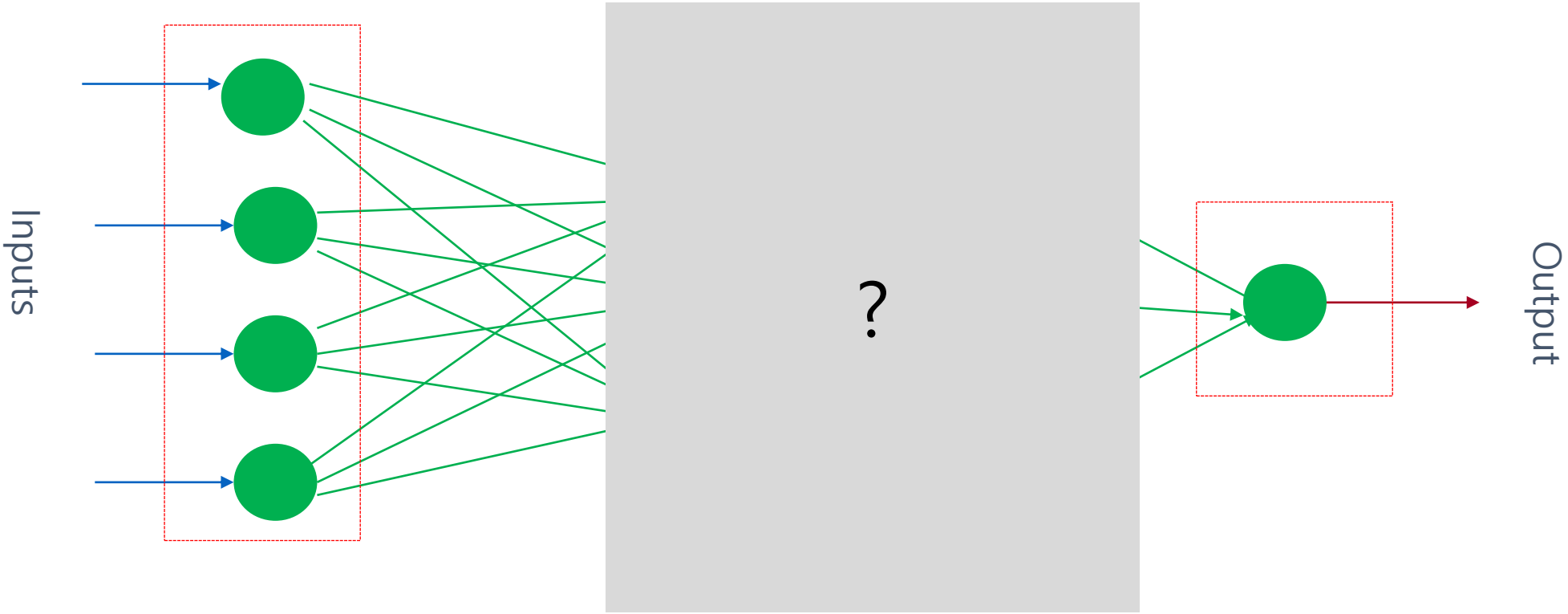
2. ANN 난제2-네트워크 구조: 입력층-은닉층-결과층으로 구성되는 ANN에서 은닉층의 노드의 수 등을 결정하기 위해 시행착오를 거치게 됨

3. ANN 난제3-Vanishing Gradient: 은닉층의 노드에서는 활성화함수를 통해 입력된 값이 변환되고, 역전파 과정에서 활성화함수의 미분값이 사용되며, 학습을 거듭할 수록 기울기가 작아지는 현상

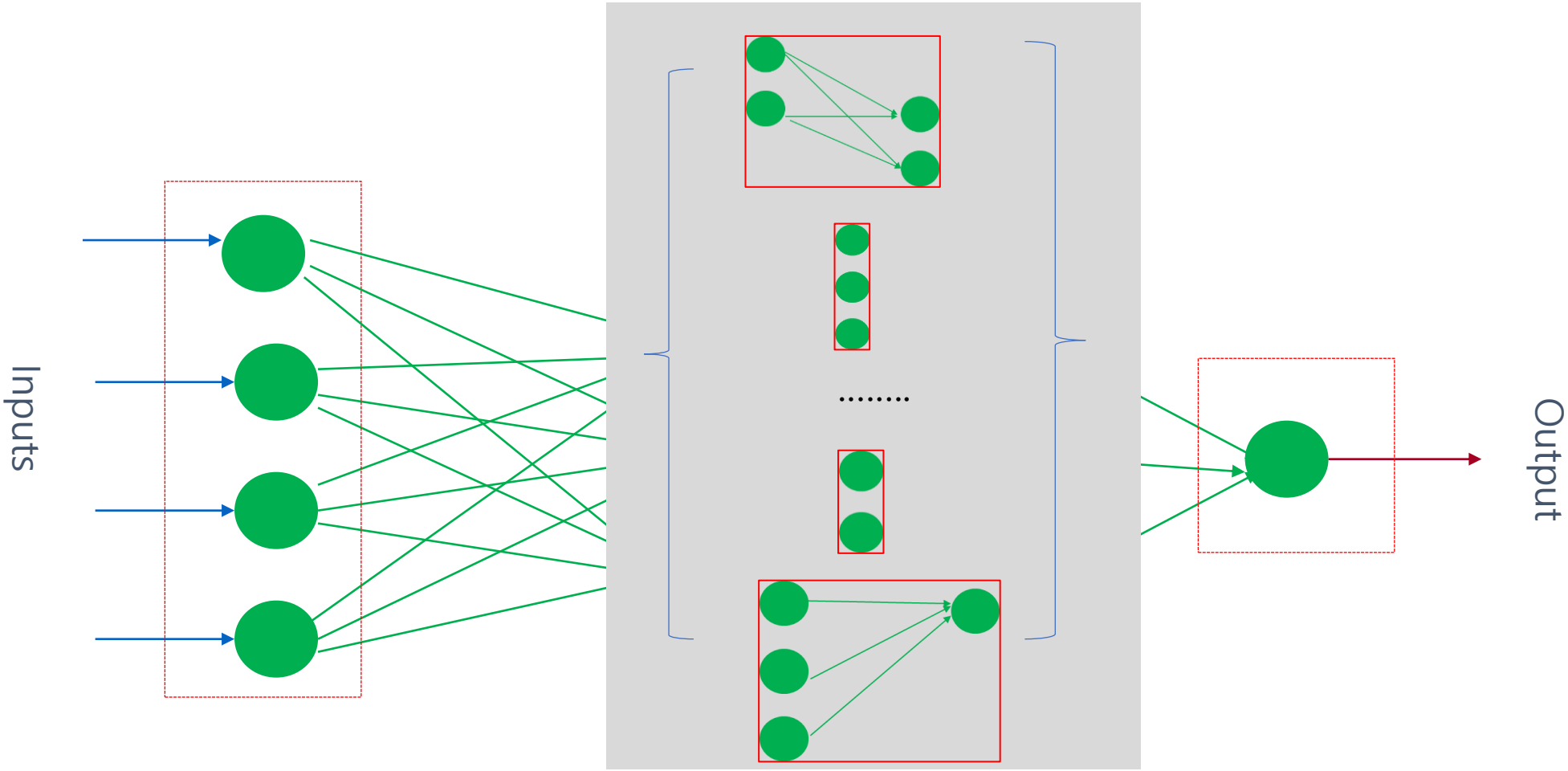
- **Artificial Neural Network(ANN, 인공신경망)**
  - 주어진 데이터에 적합한 다수의 가중치들이 최적값을 갖도록 학습
  - 어떤 X변수의 값의 단위크기 변화에 따른 결과에의 영향을 아는 것이 사실상 불가



- **Artificial Neural Network(ANN, 인공신경망) = Black Box**
  - ANN은 입력층-은닉층-결과층으로 이루어짐
  - 은닉층은 어떻게 구성되는지에 따라 ANN의 성능이 결정



- 은닉층 구성은 Trial and Error를 통해 접근해 나감 (Grid Search)

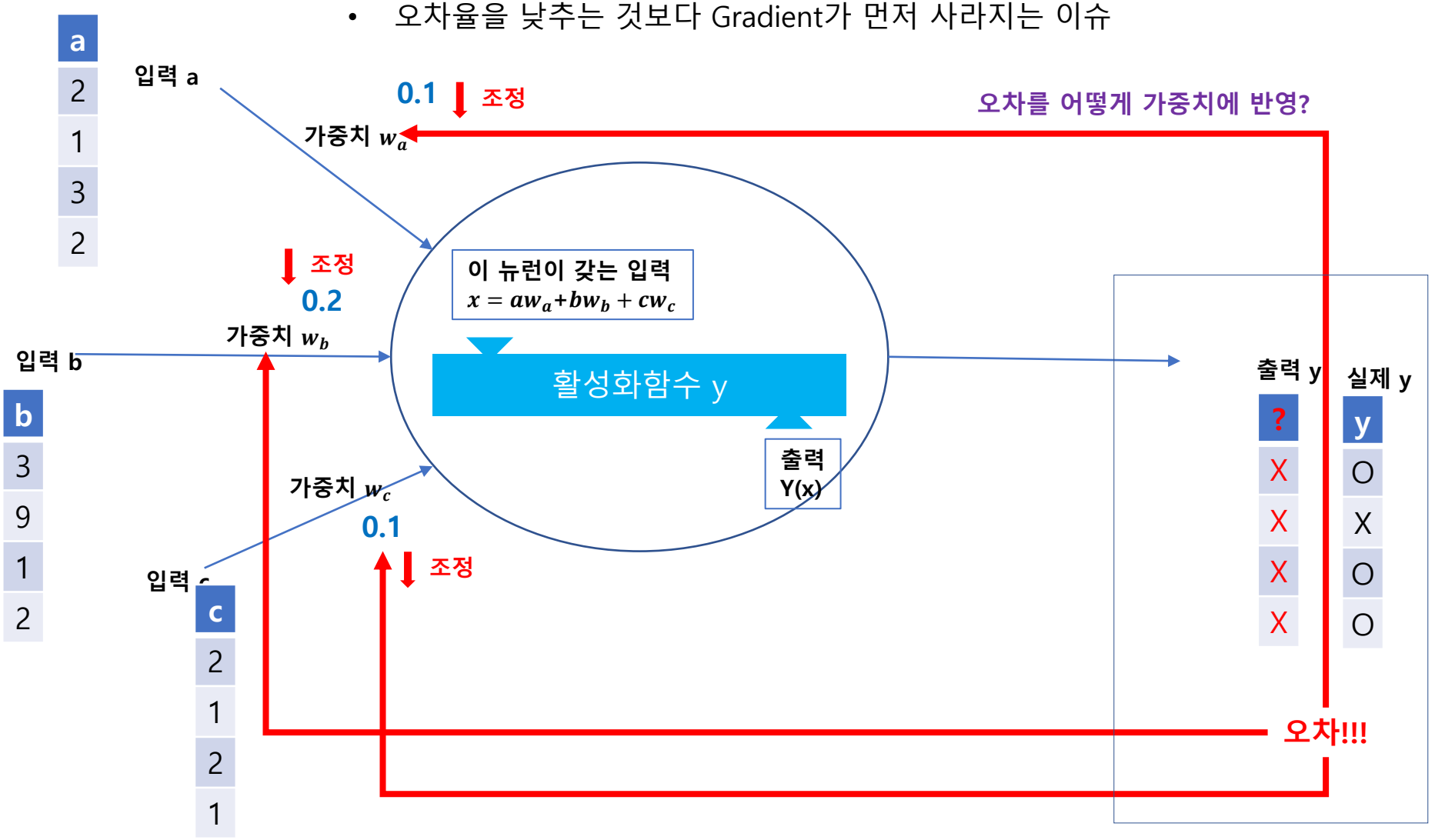


Vanishing Gradient

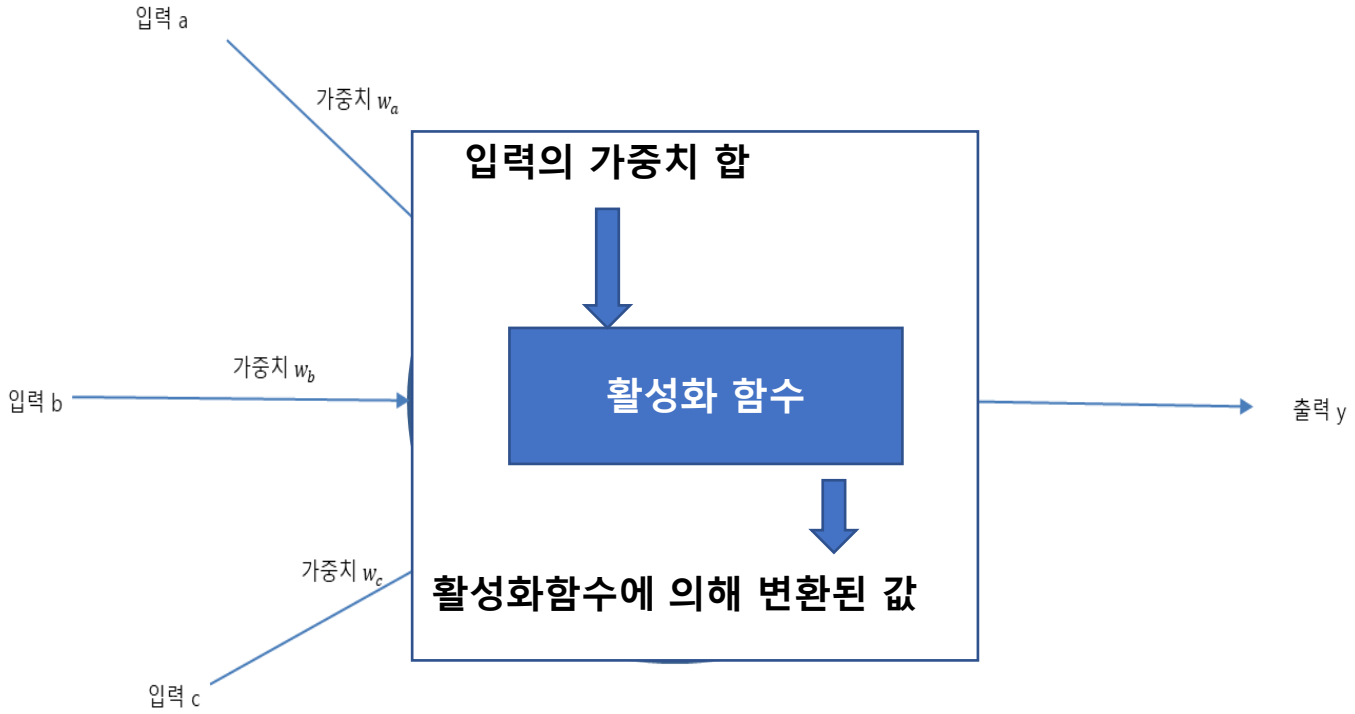
- Gradient? 미분된 함수의 기울기를 의미
- ANN의 활섬화 함수에 대해 기울기를 구할 수 있음
- 활성화 함수의 미분는 ANN 오차의 역전파(Back propagation) 과정에 사용

Vanishing Gradient

- 역전파를 통한 가중치 학습이 계속될 수록 오차가 감소
- 감소된 오차는 오차의 역전파 과정에서 작은 변화량만 전달
- 오차율을 낮추는 것보다 Gradient가 먼저 사라지는 이슈

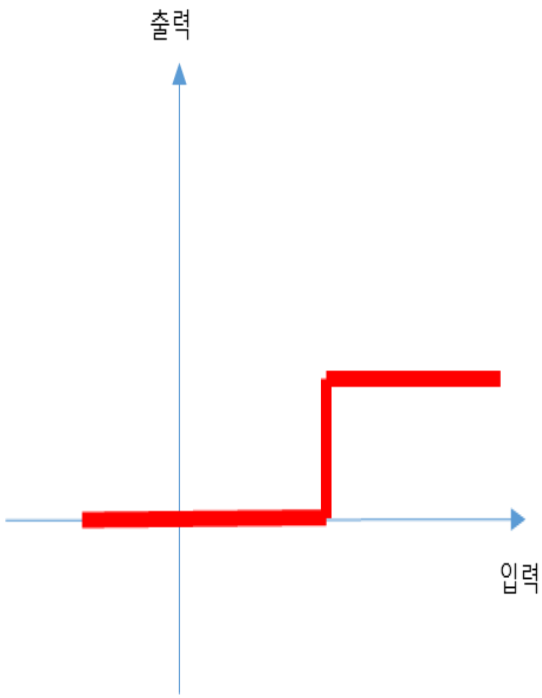


- **인공신경망 (Artificial Neural Network, ANN)의 활성화 함수**
  - 은닉층 노드에 입력되는 이전 단계 출력의 가중치 합을 변환시키는 함수
  - Activation function

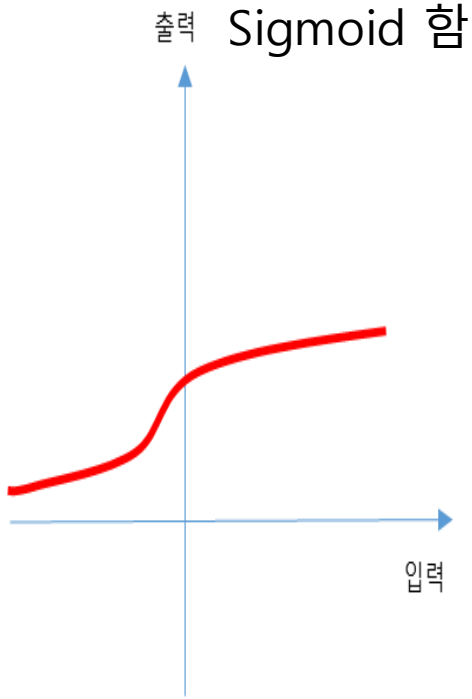


- 다양한 활성화 함수
  - 여러 종류의 활성화함수들이 사용

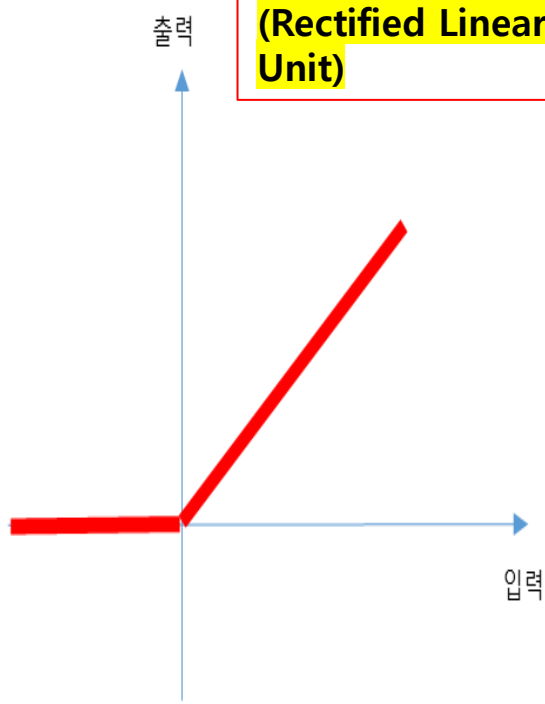
Step 함수



Sigmoid 함수



ReLU 함수 (Rectified Linear Unit)



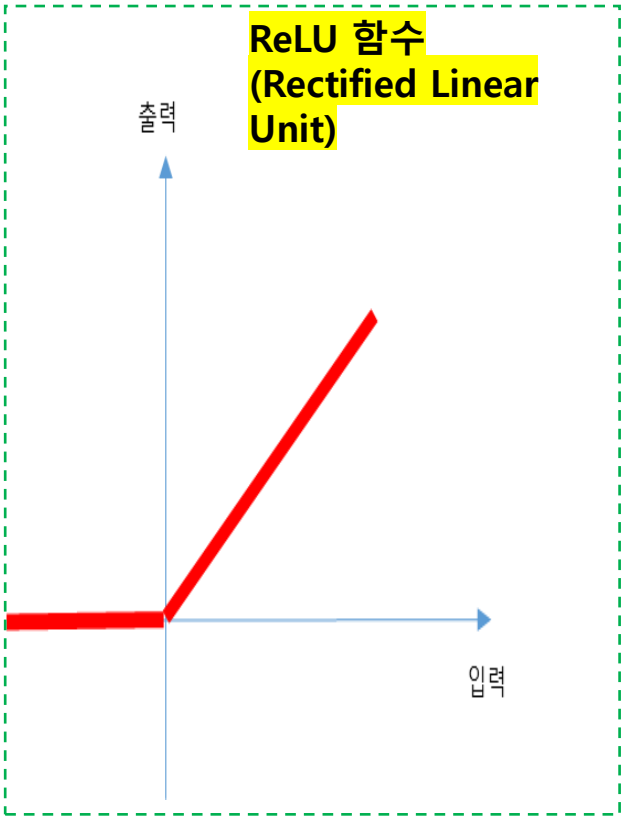
활성화 함수에 따라 가중치의 최적화도 영향을 받으며 모형의 성능에도 직결!

- ReLU 활성화 함수

Sigmoid 함수에 의한
Vanishing Gradient의 문제

->

ReLU함수를 통한 개선+빨라진 학습



GPU의 사용

- 다층 신경망의 성능 문제 해결
  - 2006년 힌턴(Hinton) 교수의 논문(A fast learning algorithm for deep belief nets)에서 해결의 단초가 제공, 이후 딥러닝이라는 이름
  - 딥러닝: 여러 개의 은닉층을 갖는 인공 신경망을 학습

- 다층 신경망의 성능 문제를 해결
  - 계산의 이슈!
    - 대용량의 계산, 그러나 각각의 계산이 아주 복잡하지는 않음
    - 굳이 CPU를 많이 사용할 필요는 없으며, GPU를 사용
    - GPU를 통한 딥러닝!


DEEP LEARNING
AUTOMOTIVE

Part II. Deep Learning

I. 딥러닝 알고리즘의 특징

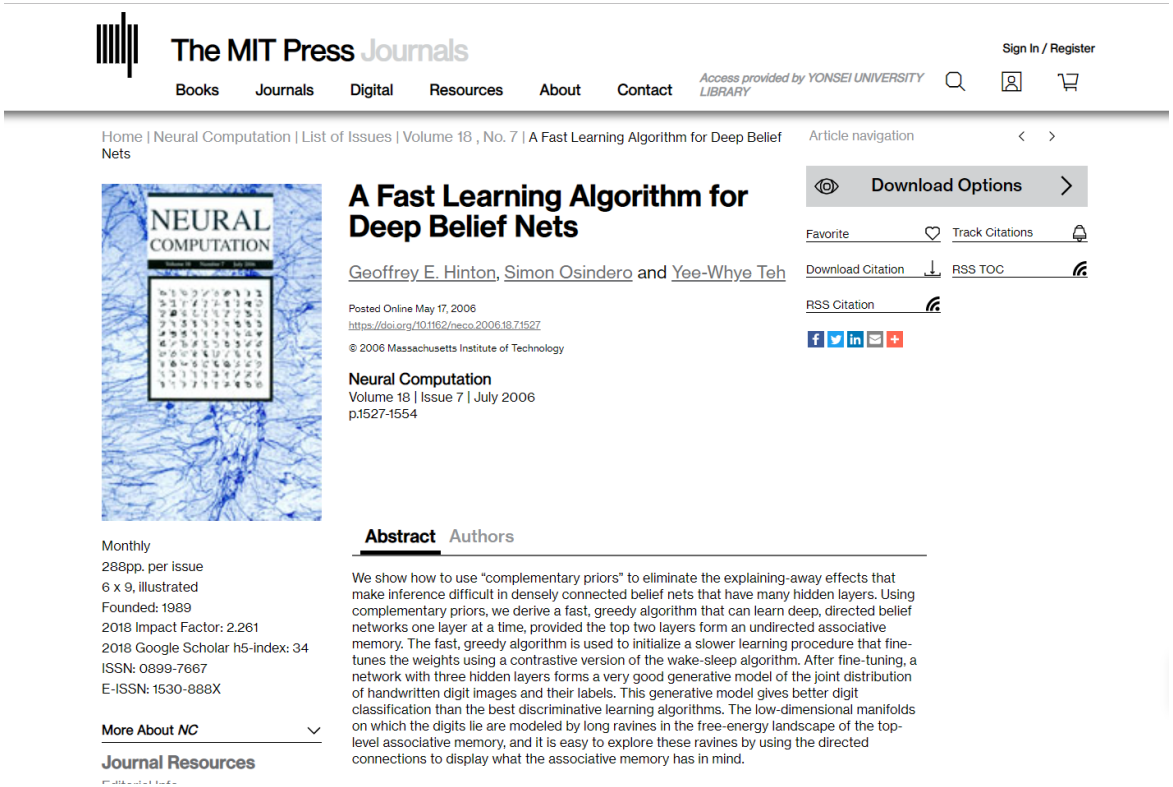
1. 딥러닝 알고리즘: 그라디언트 소실 등 기존 다층 신경망의 문제를 극복하면서 나타난 인공신경망에 대한 알고리즘

2. 딥러닝의 특성: 다수의 은닉층에서도 잘 작동하며, 많은 Feature에 대해 많은 계산을 수행하며, 높은 예측력을 보여줌

3. 딥러닝의 도구: 다양한 딥러닝의 도구를 사용할 수 있으며, 최근에는 tensorflow나 pytorch가 가장 많이 사용되고 있음

I. 딥러닝 알고리즘의 특징

- **딥러닝 알고리즘의 등장!**
  - 기존 다층신경망의 역전파 알고리즘의 극복: 은닉층이 많아질 수록 성능 저하의 이슈가 해결
  - ANN 이슈들을 해결
  - Convolution NN, Auto-encoder, Recurrent NN 등이 많이 활용



The MIT Press Journals

Sign In / Register

Books Journals Digital Resources About Contact Access provided by YONSEI UNIVERSITY LIBRARY

Home | Neural Computation | List of Issues | Volume 18 , No. 7 | A Fast Learning Algorithm for Deep Belief Nets

Article navigation

A Fast Learning Algorithm for Deep Belief Nets

Geoffrey E. Hinton, Simon Osindero and Yee-Whye Teh

Download Options

Favorite Track Citations

Download Citation RSS TOC

RSS Citation

Posted Online May 17, 2006
https://doi.org/10.1162/neco.2006.18.7.1527

Neural Computation
Volume 18 | Issue 7 | July 2006
p.1527-1554

Monthly
288pp. per issue
6 x 9, illustrated
Founded: 1989
2018 Impact Factor: 2.261
2018 Google Scholar h5-index: 34
ISSN: 0899-7667
E-ISSN: 1530-888X

More About *NC*

Journal Resources

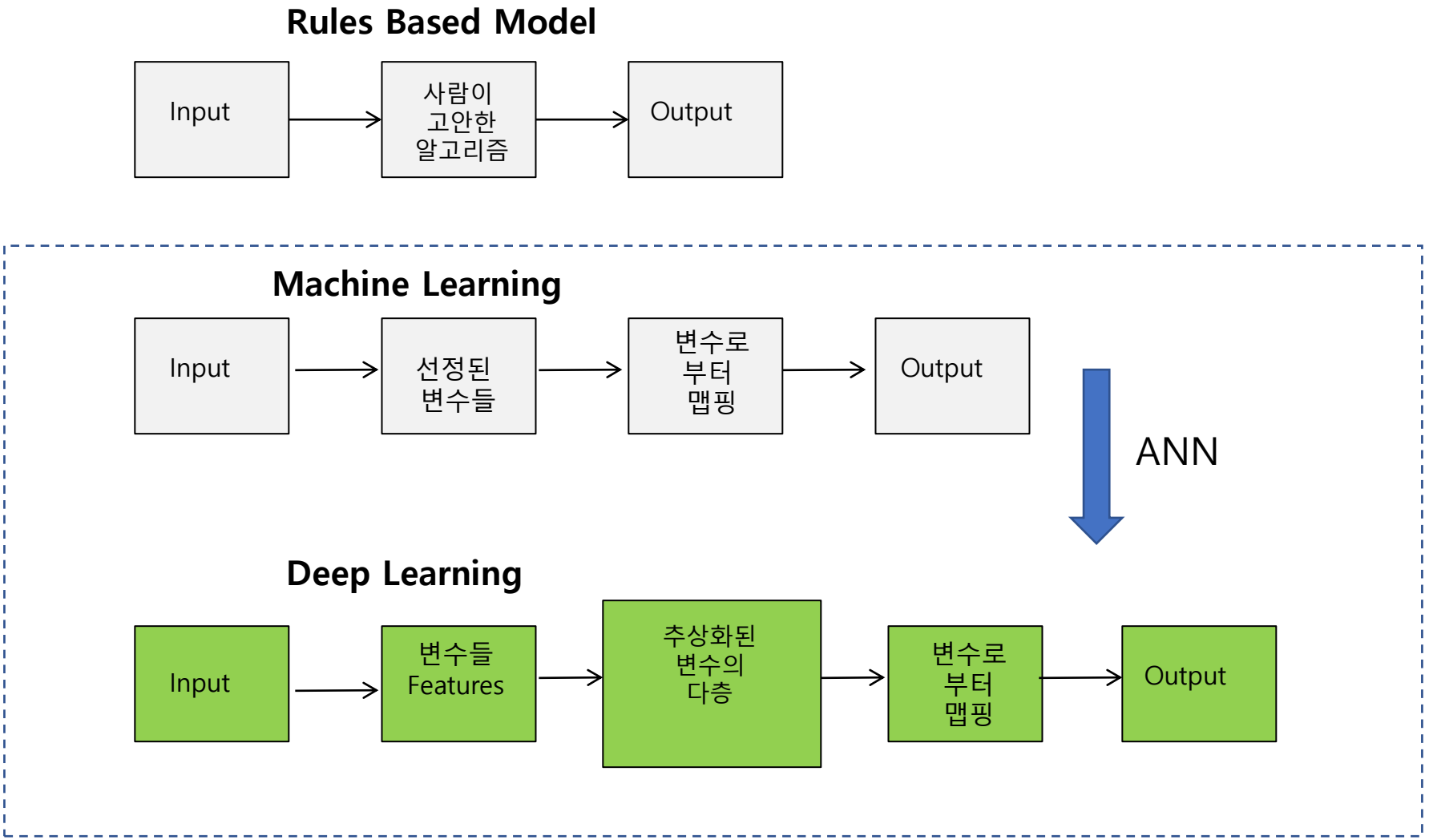
Abstract Authors

We show how to use "complementary priors" to eliminate the explaining-away effects that make inference difficult in densely connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modeled by long ravines in the free-energy landscape of the top-level associative memory, and it is easy to explore these ravines by using the directed connections to display what the associative memory has in mind.

다층신경망의 성능 문제의 해결방법이
2006년 Hinton의 "A fast learning algorithm for deep belief nets"를 통해 제시되면서, '딥러닝'으로 주목

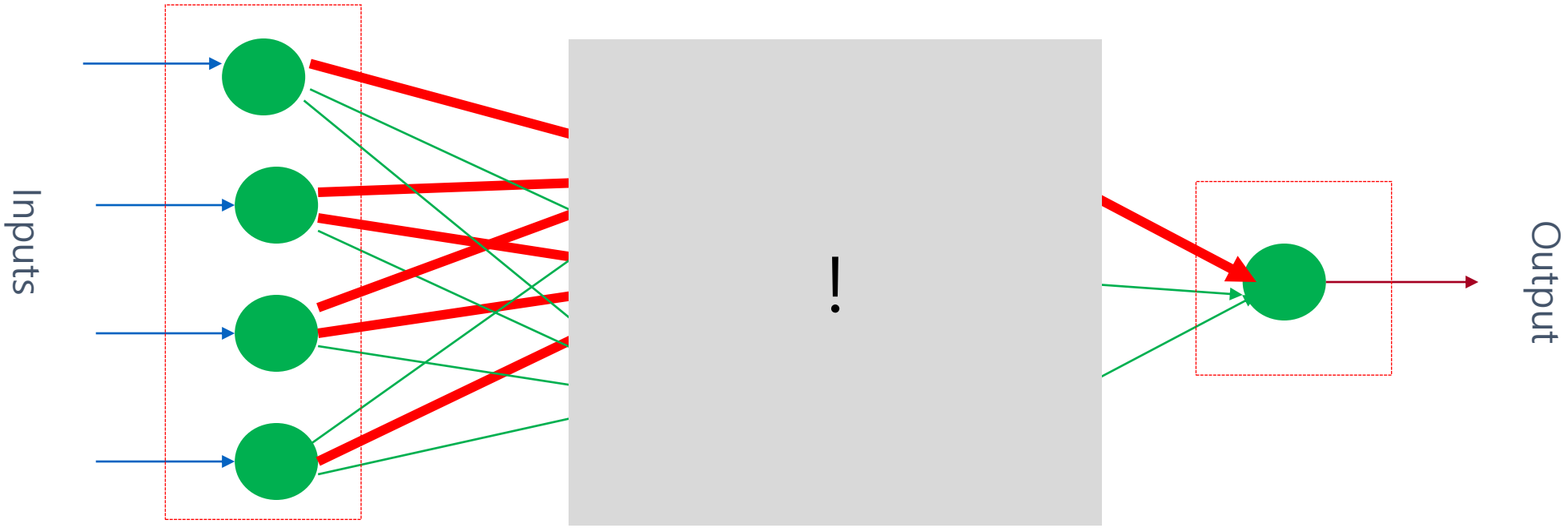
I. 딥러닝 알고리즘의 특징

• 딥러닝 특징



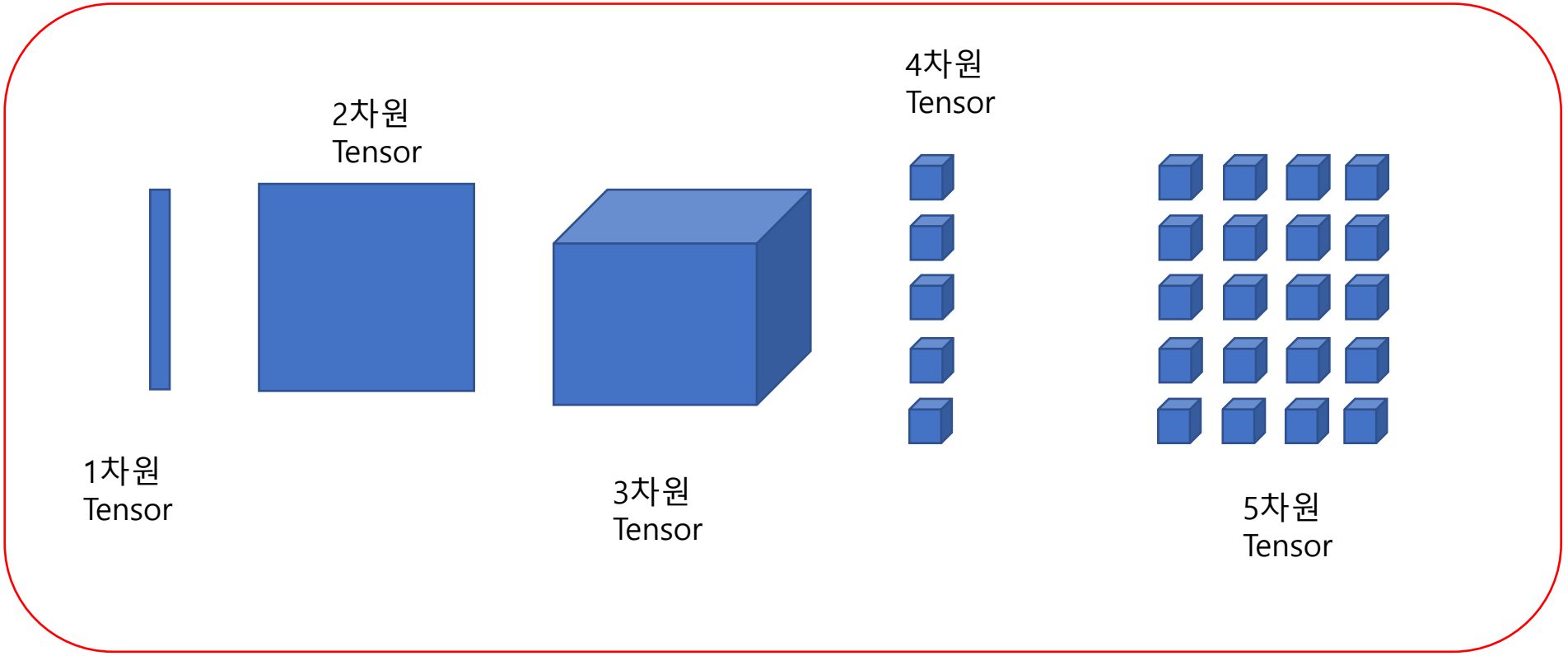
딥러닝의 특성

- 다수의 은닉층
- 활성화 함수를 통한 그라디언트 소실의 극복
- 그리고 계산량



I. 딥러닝 알고리즘의 특징

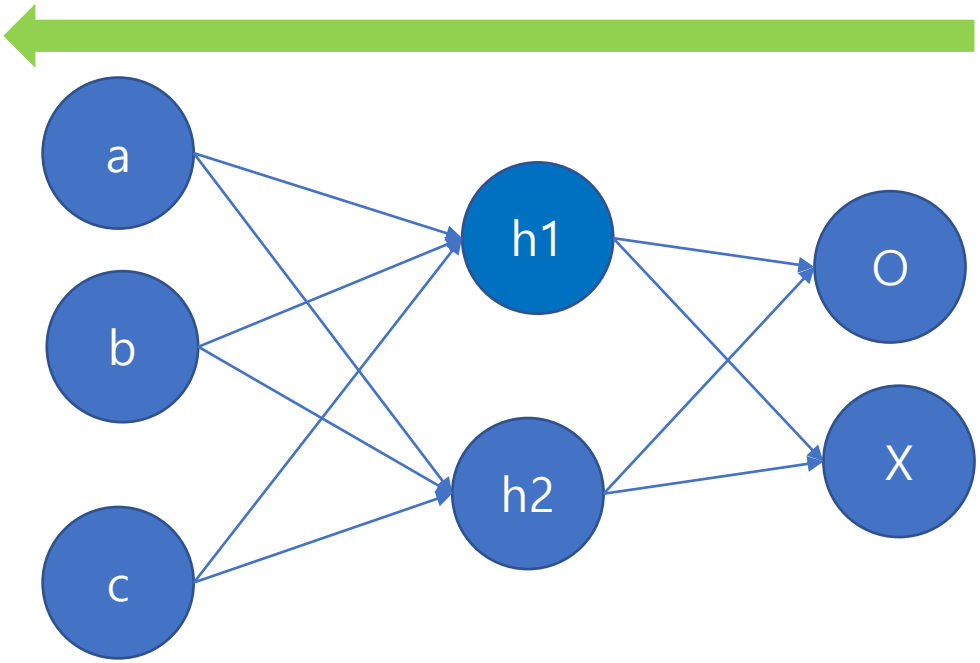
- **딥러닝의 기본 자료구조: Tensor**
  - 다차원 배열이나 리스트
  - Rank(차원의 수), shape(행과 열), type(값의 type)으로 구분

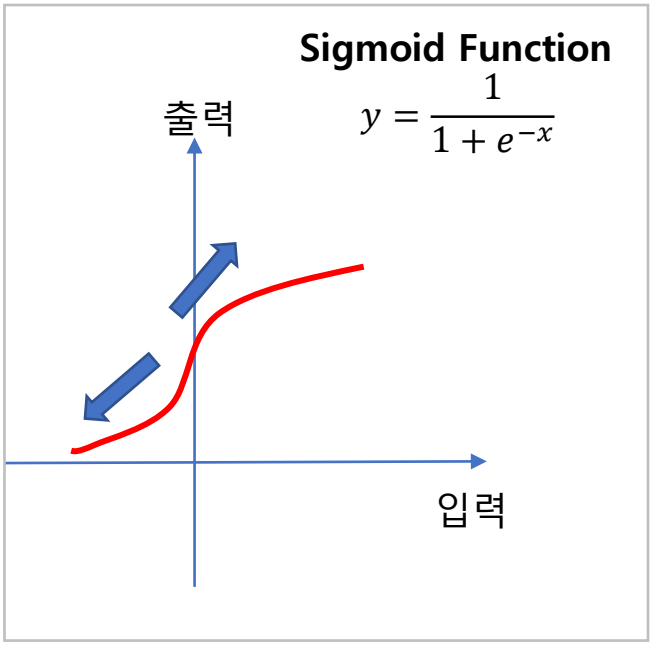


Vanishing Gradient

- *Gradient? 미분된 함수의 기울기*
- *Gradient는 역전파(Back propagation) 과정에 사용: 기존 활성화 함수(Sigmoid)를 거치면 원래의 값이 현저히 작아짐(작은 x는 더 작은 y로 변환, 큰 x는 더 큰 y로 변환)*

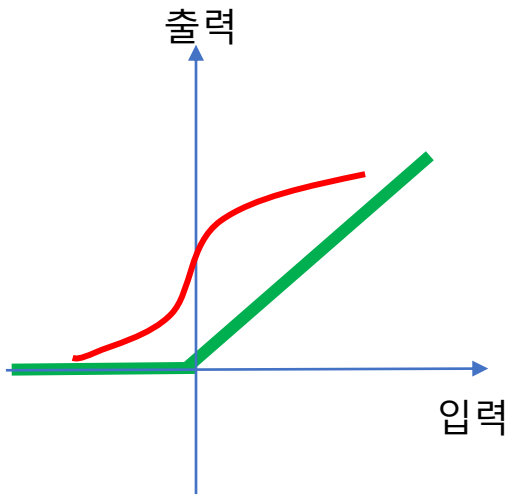
인공신경망의 두 번째 겨울 (1986-2006)





ReLU 활성화함수의 등장 Rectified Linear Unit

2006년 Jeffrey Hinton, "Vanishing Gradient의 문제는 Sigmoid 활성화 함수 때문!"



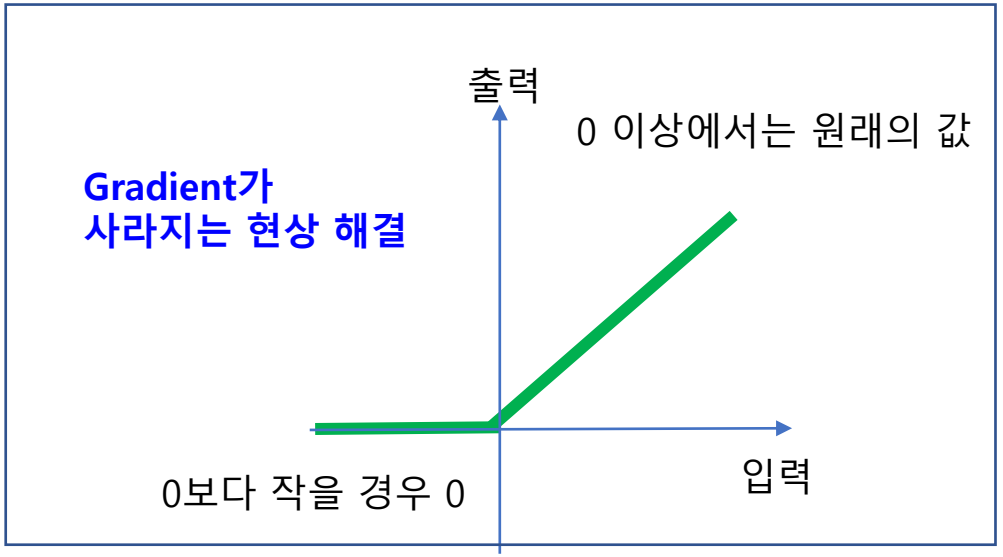
Sigmoid Function

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

ReLU Function
(Rectified Linear Unit)

ReLU 활성화 함수를 통한 인공신경망의 개선

- 여러 은닉층에 잘 작동!
- 계산 비용 낮추고 정확도 증가
- ReLU도 다양한 변형이 가능!



딥러닝의 도구들

- Deep Learning 알고리즘 및 Python 기능 소개

Deep Learning을 위한 Tool set

Tensorflow

- An open-source software library for Machine Intelligence
- GPU 지원 / Parallel Computing 지원

MXNet

- A Flexible and Efficient Library for Deep Learning
- Apache Incubator
- Gluon을 통한 Interfacing
- GPU 지원 / Parallel Computing 지원

pyTorch

- an open source machine library
- GPU 지원 / Parallel Computing 지원
- Facebook에서 개발
- 파이썬에서의 자연스러운 사용!

Keras

- An open source neural network library written in Python
- mxnet, tensorflow, cntk, deeplearning4j, theano를 Backend로 사용, tensorflow에서 Keras 지원하며, 인터페이스로 인식
- high-level neural networks API
- 쉽고 빠른 deep learning 프로토타이핑 지원

II. 딥러닝 파라미터의 이해

1. 딥러닝 파라미터 개요: 파라미터란 매개변수이며, 모델 설정을 위해 내부에서 데이터로 부터 추정된 값이 사용, 하이퍼파라미터는 모델링을 위해 설정해주는 값, 딥러닝에서는 신경망 구조 관련 부분, 학습율, 모멘텀, 배치사이즈 등 이 있음

2. 은닉층과 노드의 수: 인공신경망의 구조와 관련된 은닉층의 수와 노드의 수는 Trail and Error를 통해 발견

3. 최적화 알고리즘과 학습율: Gradient Descent를 위한 알고리즘과 학습율 선택을 통한 가중치 최적화

파라미터? 하이퍼파라미터?

파라미터(Parameter):

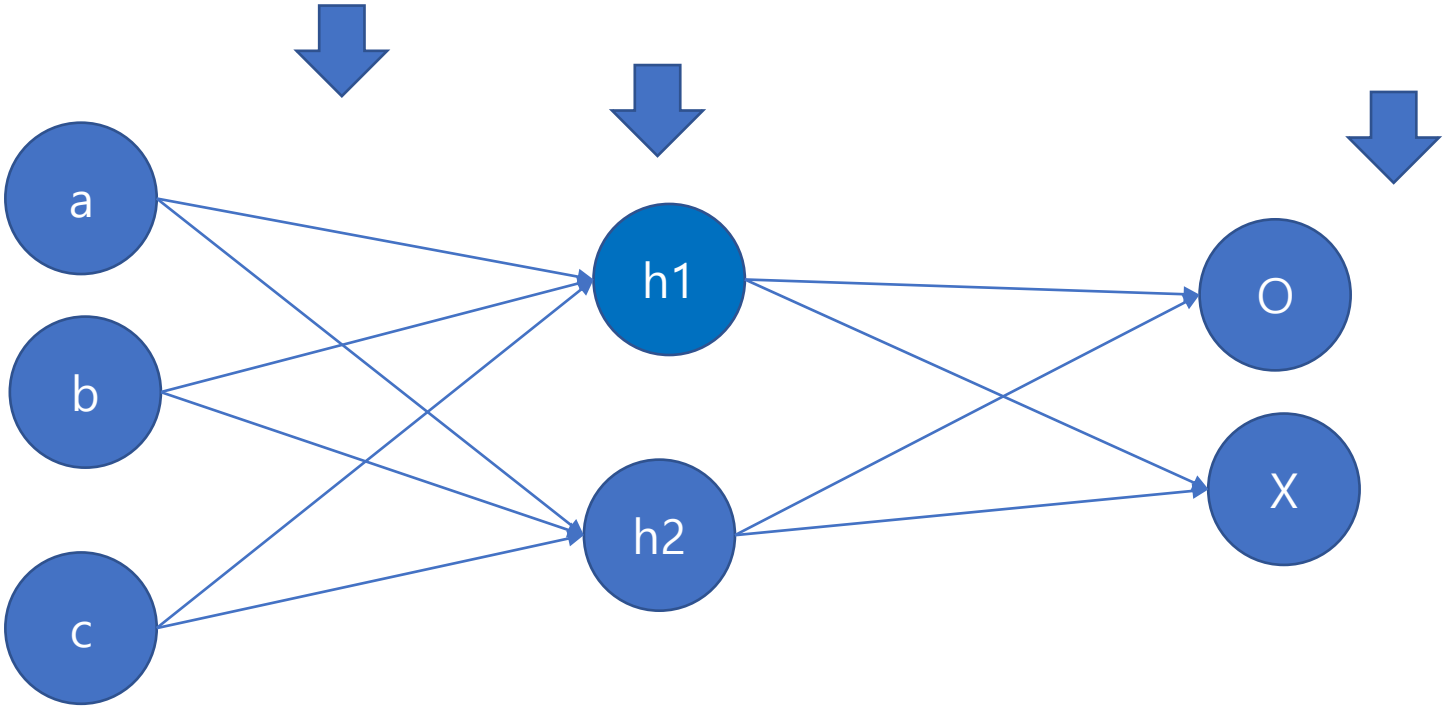
매개변수, 모델 설정을 위해 내부에서 데이터로 부터 추정된 값이 사용

하이퍼파라미터(Hyper Parameter):

모델링을 위해 설정해 주는 값, 딥러닝에서는 신경망 구조 관련 부분, 학습율, 모멘텀, 배치사이즈 등 이 있음

인공신경망의 Parameter 혹은 Hyper Parameter

은닉층의 수, 노드의 수, 학습율 등



인공신경망의 구성

- 은닉층과 노드의 수?
- 인공신경망의 구조와 관련된 은닉층의 수와 노드의 수는 Trail and Error를 통해 발견

입력변수의 수 > 은닉층 노드의 수

차원을 축소!

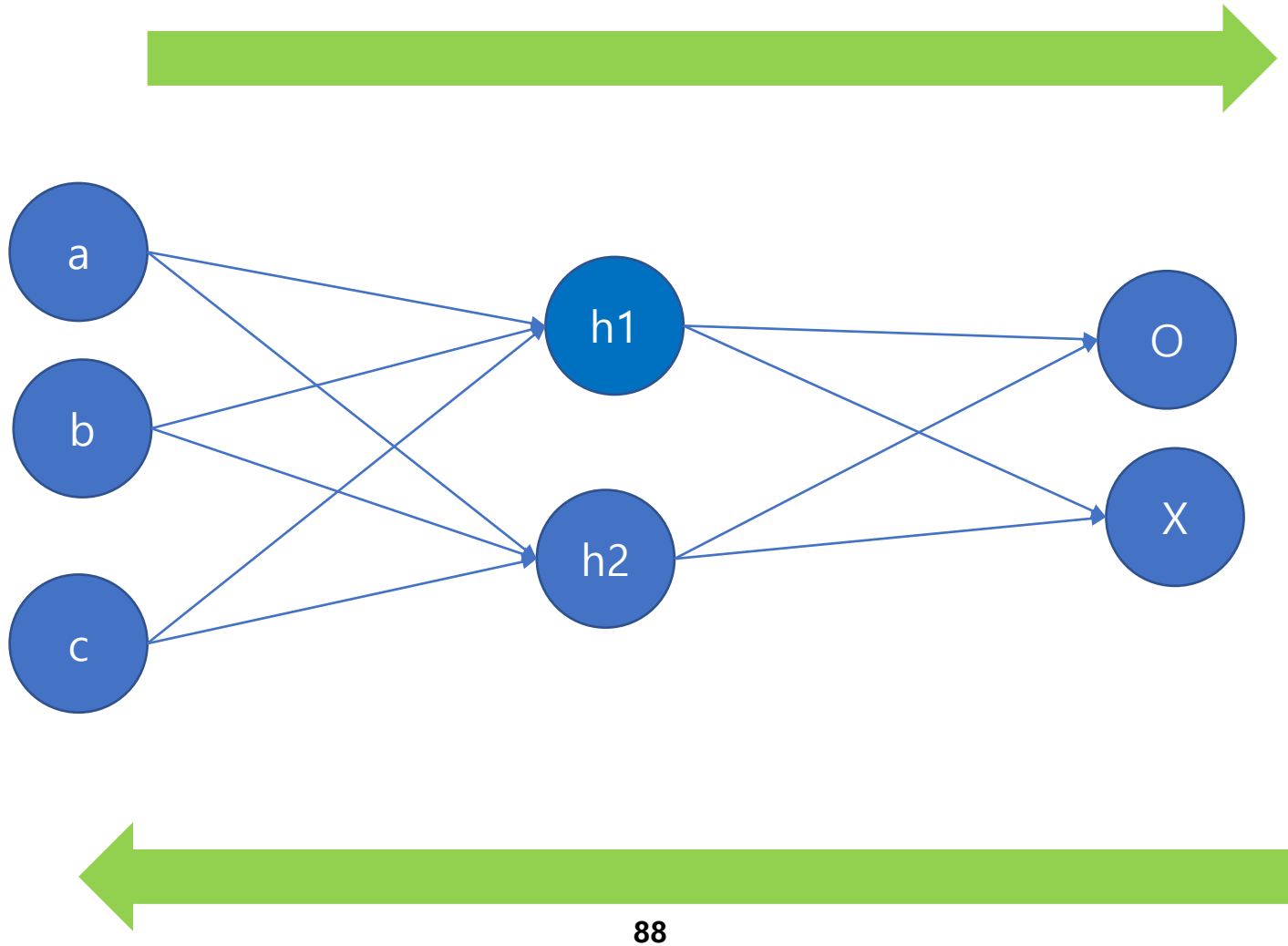
입력변수의 수 < 은닉층 노드의 수

차원을 확장!

은닉층이 많으면? 노드의 수가 많으면?

과적합, 계산 비용, 차원의 저주

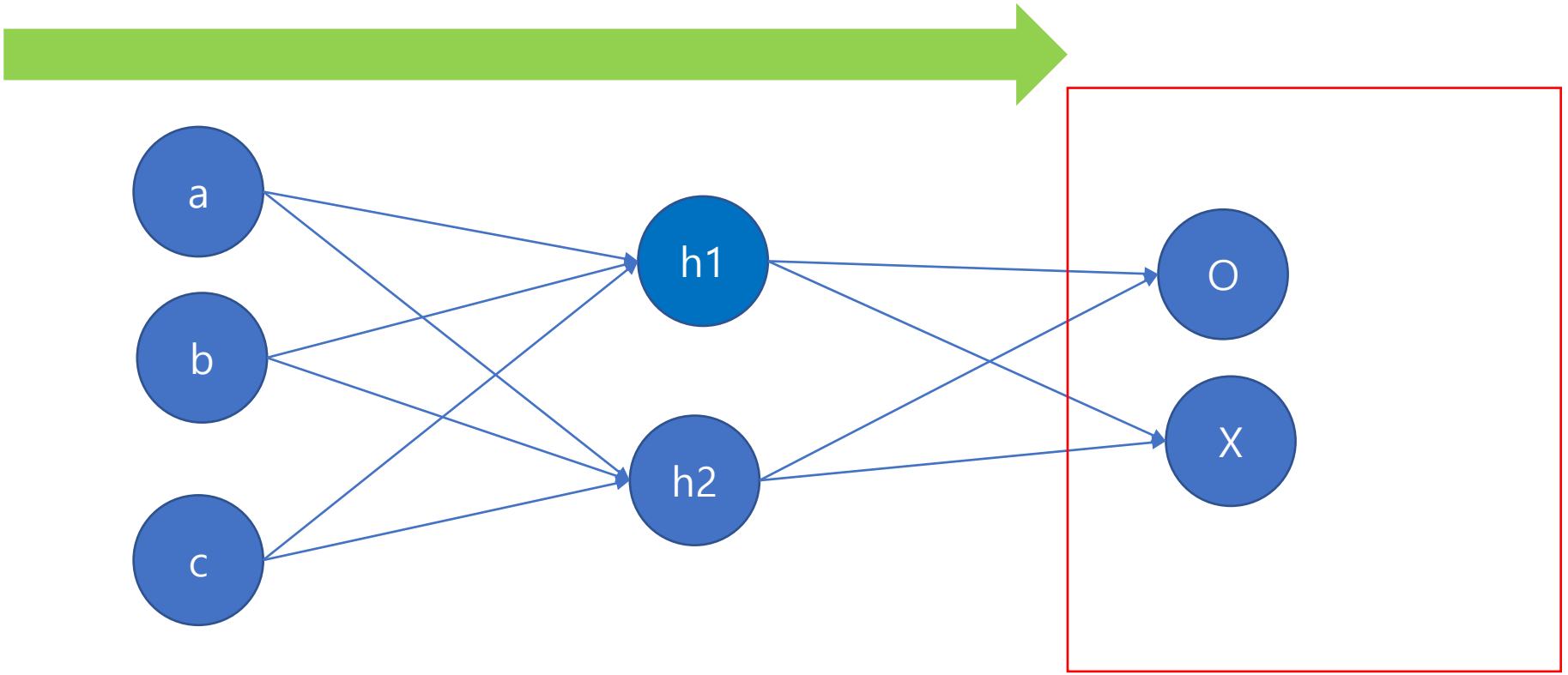
Epoch: Forward Propagation + Back Propagation



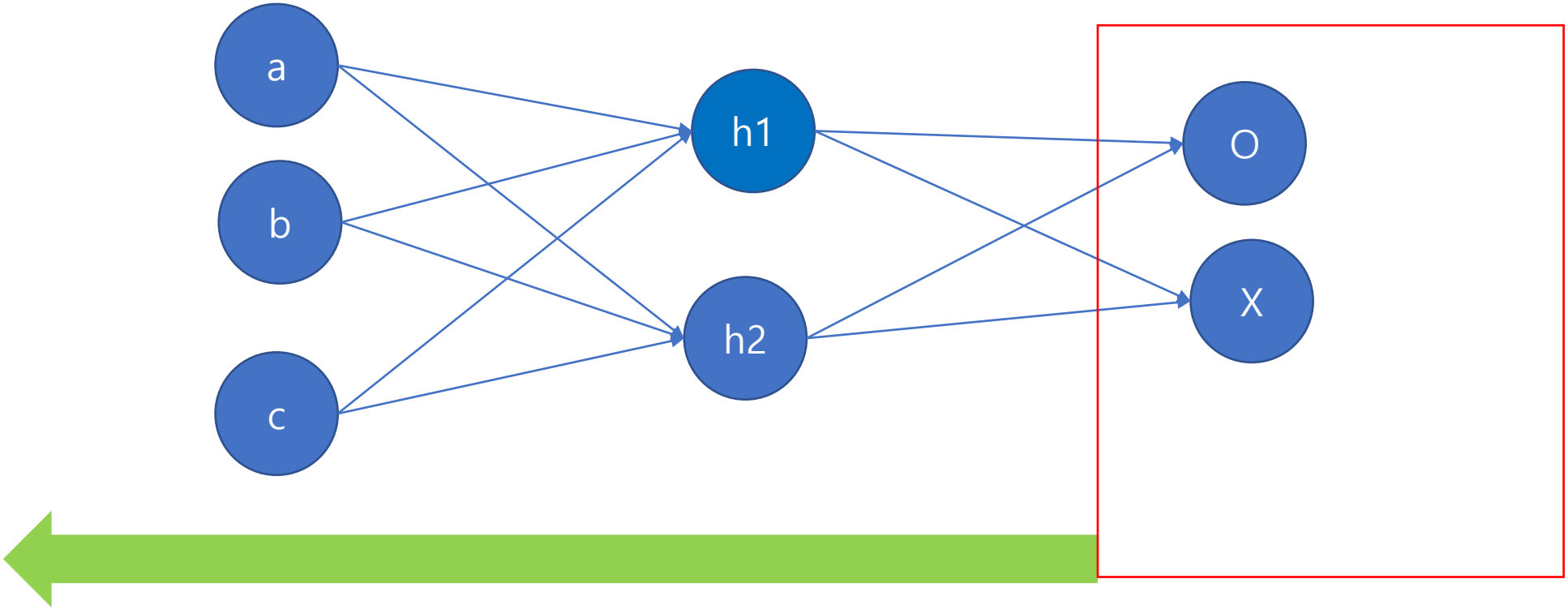
손실함수로 채점!

오차?

인공신경망의 순전파 과정을 통해 계산된 예측값과 실제값과의 차이를 측정



역전파알고리즘: 오차를 활용하여 가중치를 업데이트하는 알고리즘

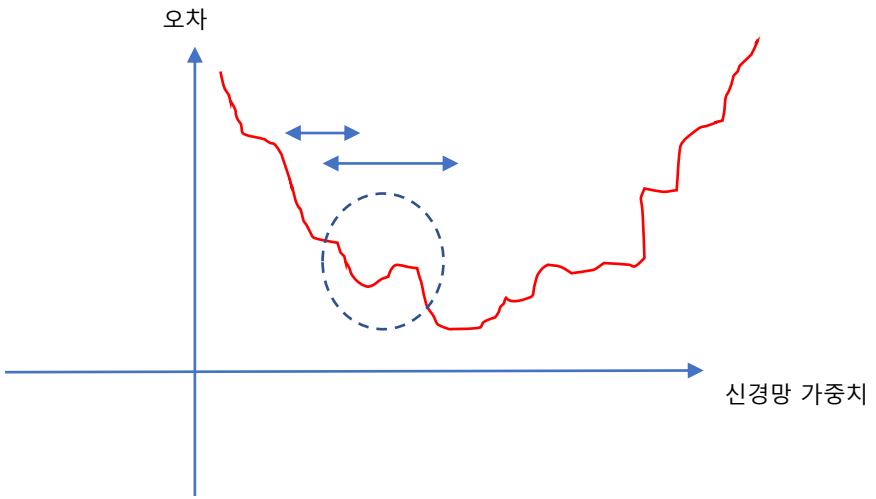


최적화 알고리즘과 학습율

학습율(Learning rate): 가장 중요한 하이퍼파라미터!

너무 크게 잡힌 경우!
오차가 크다!

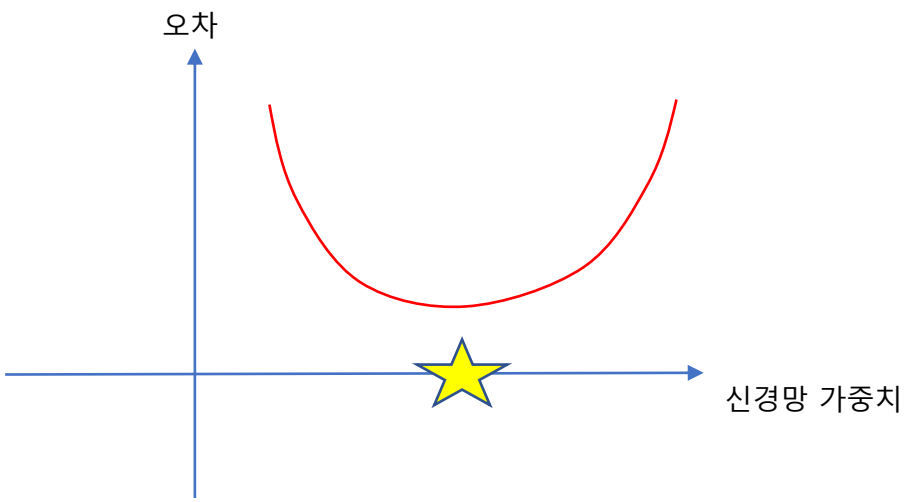
너무 작게 잡힌 경우!
최적이 아닌 가중치를 계산한다!

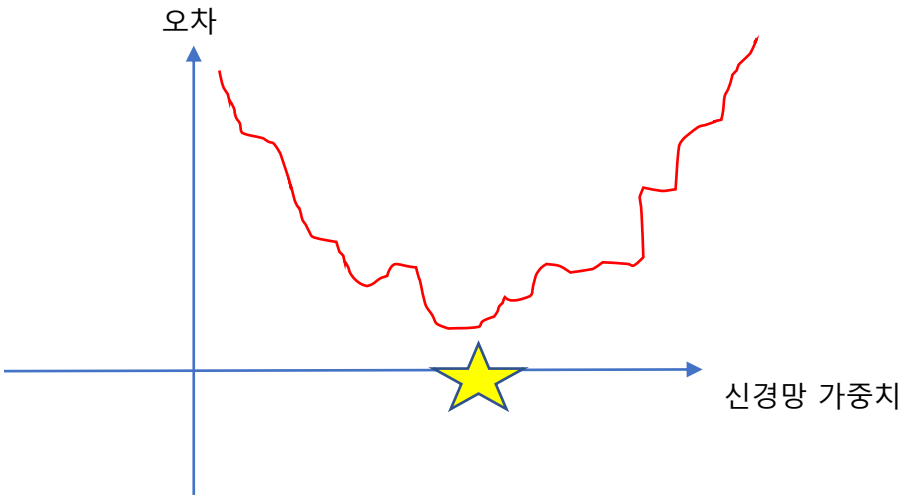


최적화 알고리즘과 학습율

- 최적화 알고리즘
  - Gradient Descent
    - Stochastic Gradient Descent
      - Momentum(관성 고려)
      - Adam(Momentum+RMSProp)
    - Adagrad(진행될 수록 변화 정도를 줄임)
      - RMSProp(상황을 봐가며 변화정도 조정)
        - Adam(Momentum+RMSProp)

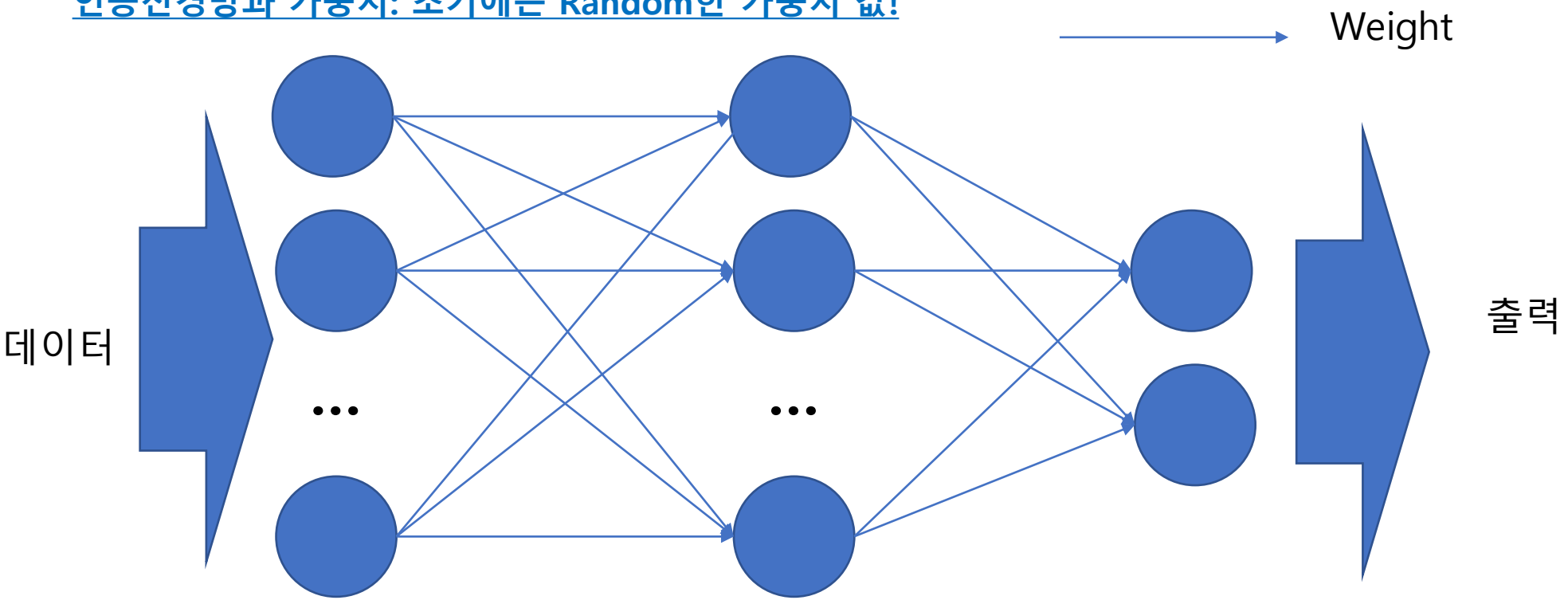
가중치 발견






그대로 다음 세대에도 넘었다

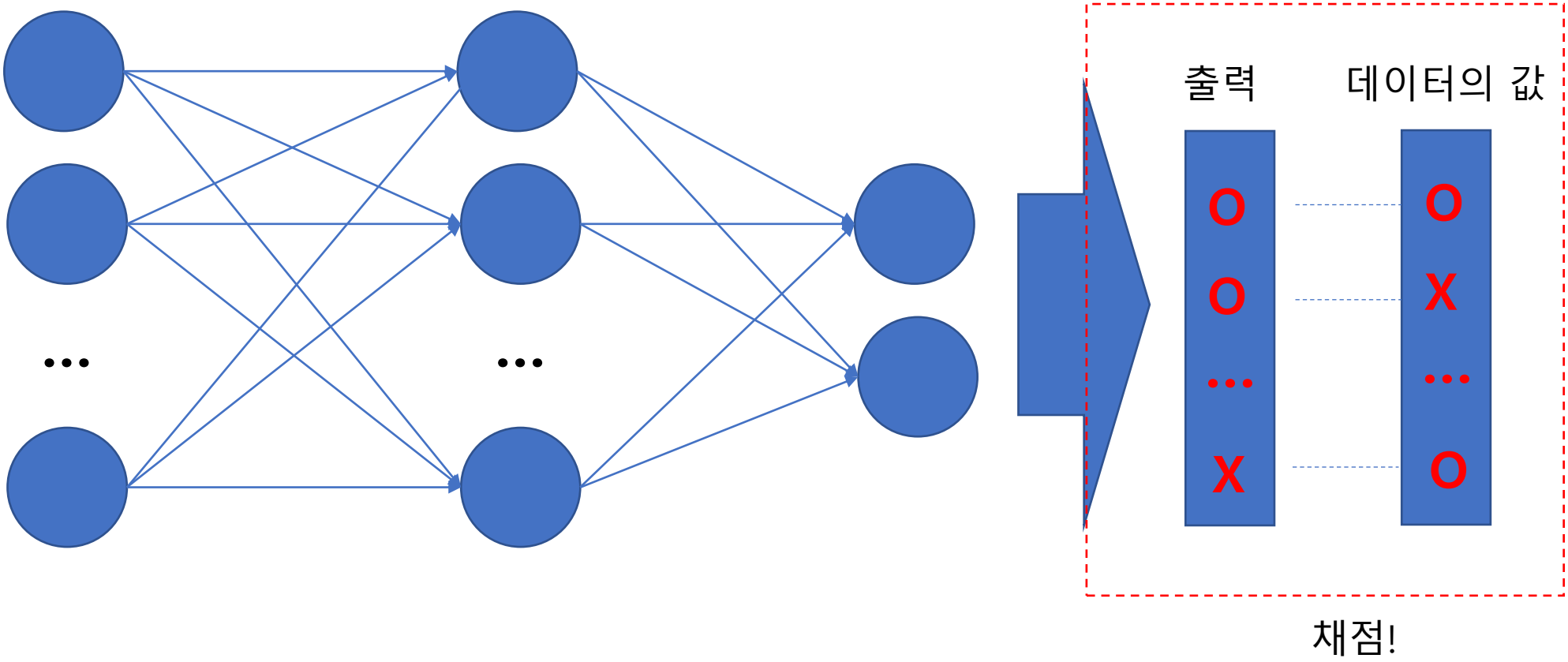
인공신경망과 가중치: 초기에는 Random한 가중치 값!



원하는 값으로 변환하기 위해 입력값에 곱해지는 수치, weight라고 하며 값을 강조할 경우 가중치를 증가시킬 수 있음.

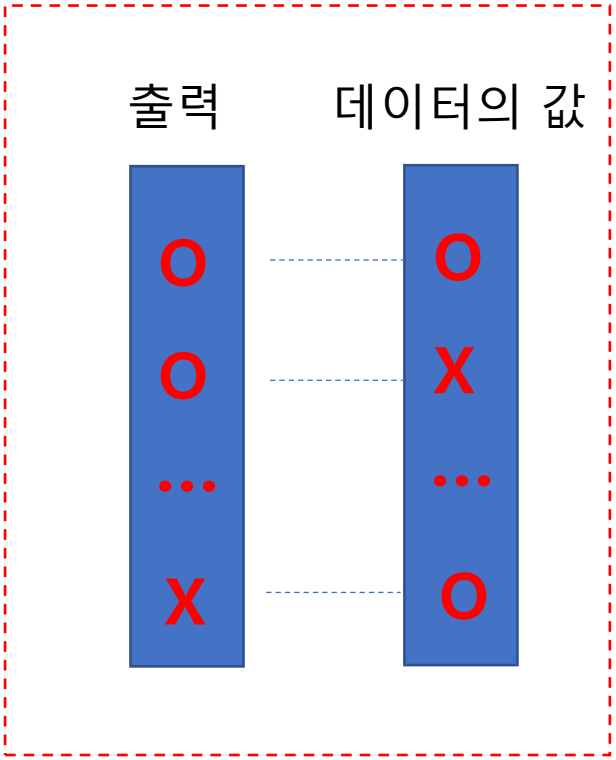
- 주어진 값을 2배 강조하고 싶은 경우? 주어진 값 X 2
- 주어진 값을 0.1배로 강조하고 싶은 경우? 값 X 0.1

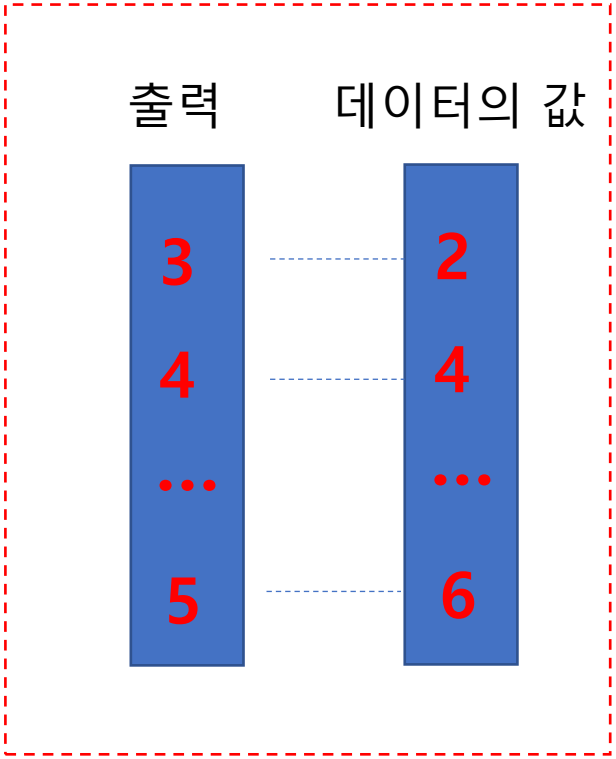
인공신경망과 가중치



인공신경망 출력의 채점: 손실함수

ANN이 가중치를 활용해 계산한 값과 원래 해당 값의 비교
출력값은 Category이거나 Numeric일 수 있음





III. 경사 하강과 미니배치

1. 함수와 기울기: 입력과 출력의 관계를 나타내는 함수의 변화 정도는 기울기로 측정

2. 경사하강(Gradient Descent): 함수의 기울기를 작게하면서 최소값을 구할때 까지 반복함

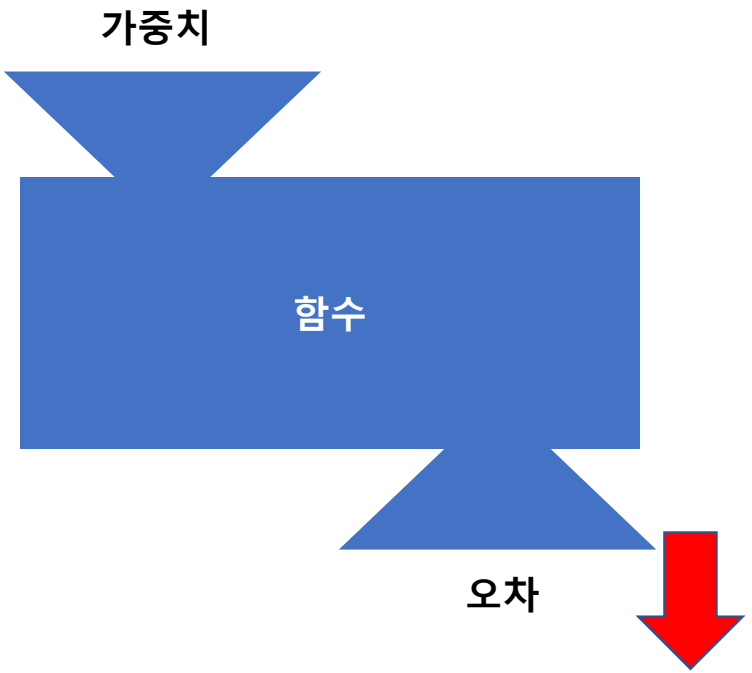
3. 확률적 경사하강(Stochastic Gradient Descent): 경사하강의 계산시간이 오래걸리는 점을 보완하기 위해 일부 데이터로 최소값을 발견

4. 배치(Batch)와 미니배치(Mini Batch): 배치는 공정에서의 묶음 단위이며, 딥러닝에서는 한 에포크에 학습되는 입력 데이터를 의미, 미니배치는 배치의 부분집합이며, 딥러닝의 속도를 개선

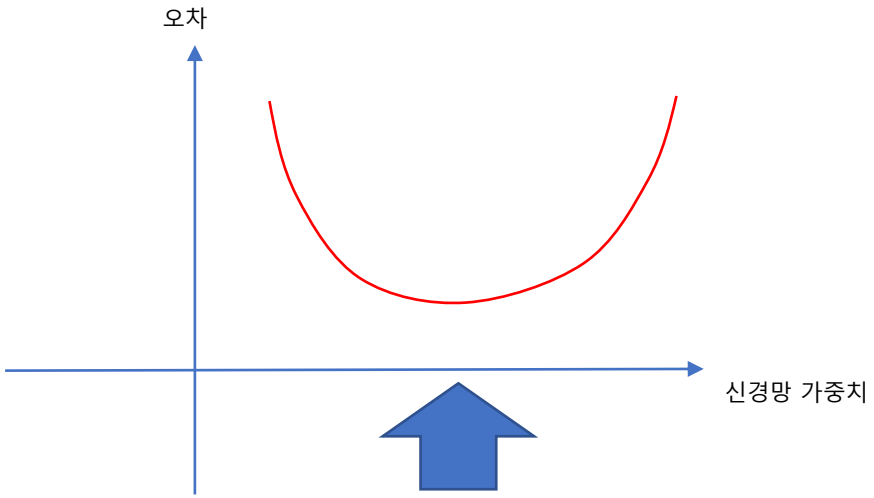
5. 미니배치의 특징: 계산 속도를 개선하며, 전체 입력데이터를 처리하는 장점

6. 미니배치와 에포크: 미니배치로 전체 데이터를 다 학습한 것이 한 에포크를 구성

함수와 기울기: 입력과 출력의 관계를 나타내는 함수의 변화 정도는 기울기로 측정

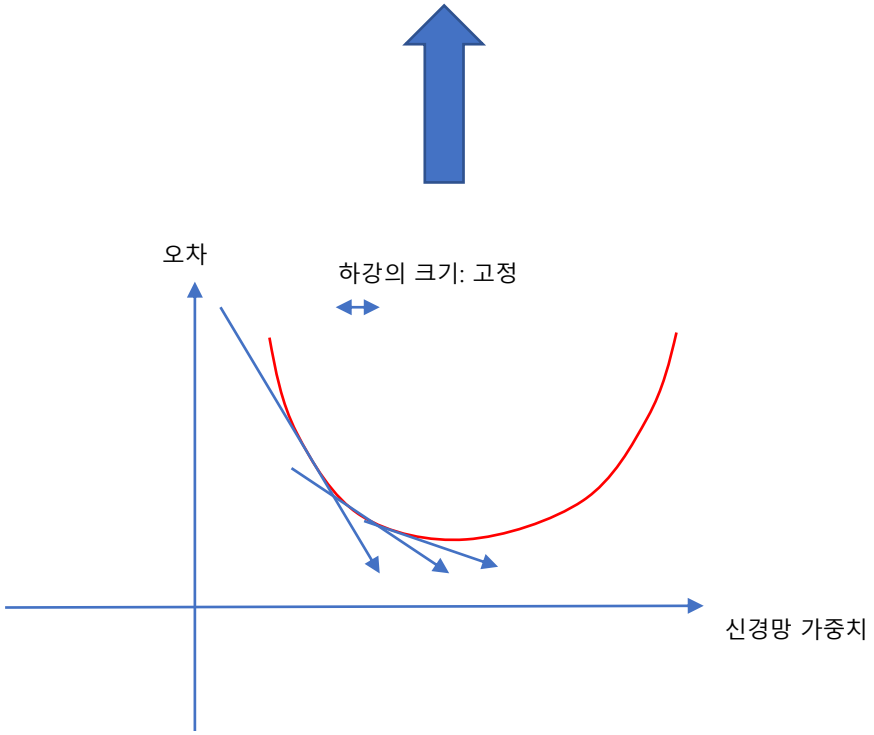


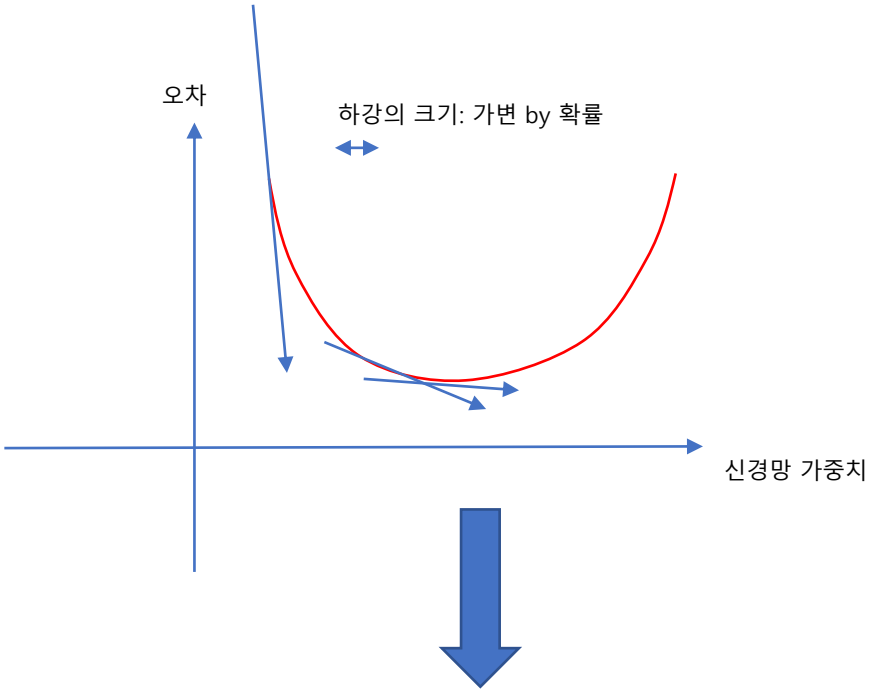
➢ **오차에 대한 함수**



이 곳이 최적!

- **경사하강 (Stochastic Gradient Descent):**
  - **함수의 기울기를 작게하면서 최소값을 구할때 까지 반복**
  - **최소값을 발견! 계산시간 소요!**





- **확률적 경사하강 (Stochastic Gradient Descent)**
  - **경사하강의 계산시간이 오래걸리는 점을 보완하기 위해 일부 데이터로 최소값을 발견**

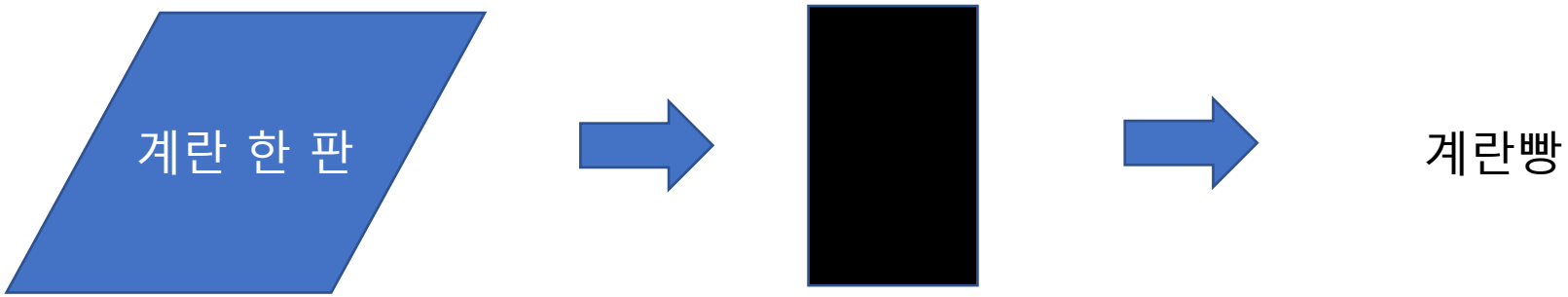
배치(Batch)?

공정에서의 묶음 단위

=

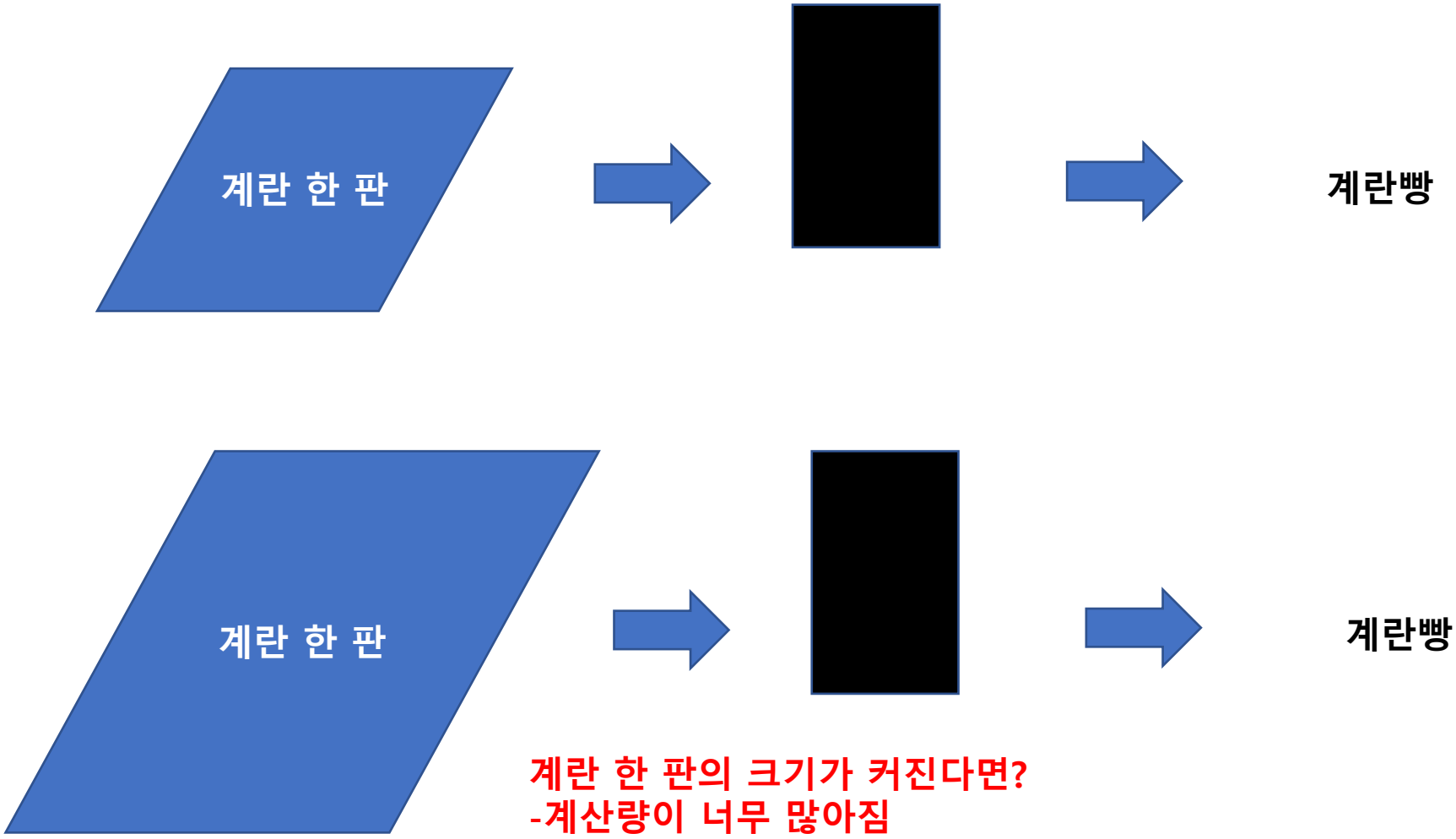
전체 데이터

경사하강을 통한 최적의 가중치!



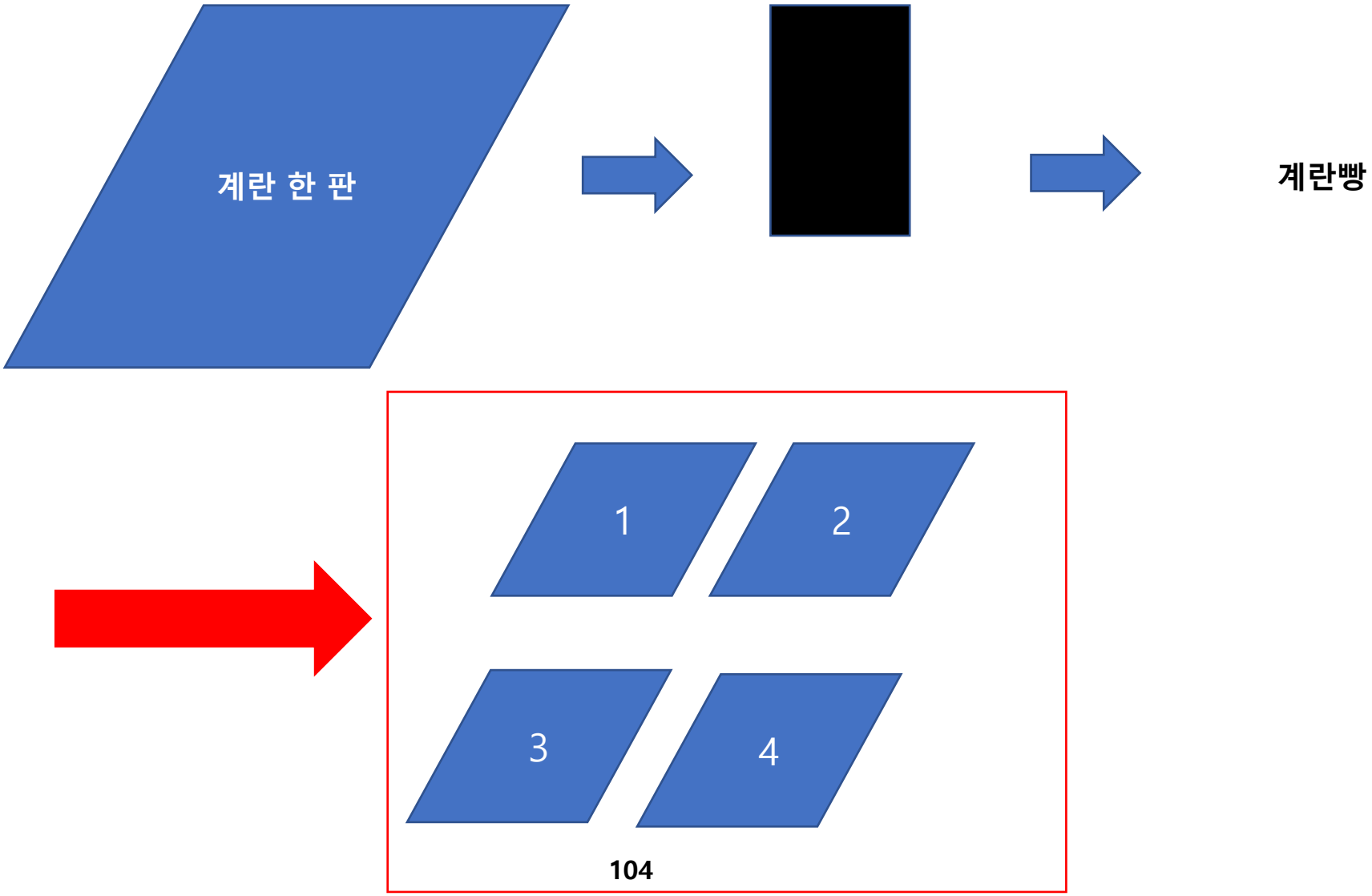
배치방식의 장점:
-개별로 처리하는 것보다는 배치로 처리하는 것이 효율적!
-안정적인 최적의 값을 발견할 수 있음

배치(Batch)가 큰 경우



계란 한 판의 크기가 커진다면?
-계산량이 너무 많아짐
-경사하강 등을 적용 시 속도가 느려질 수 있음!

배치(Batch)가 큰 경우: 미니배치(Mini Batch)!



미니배치(Mini Batch)의 특징

Batch: 전체 데이터

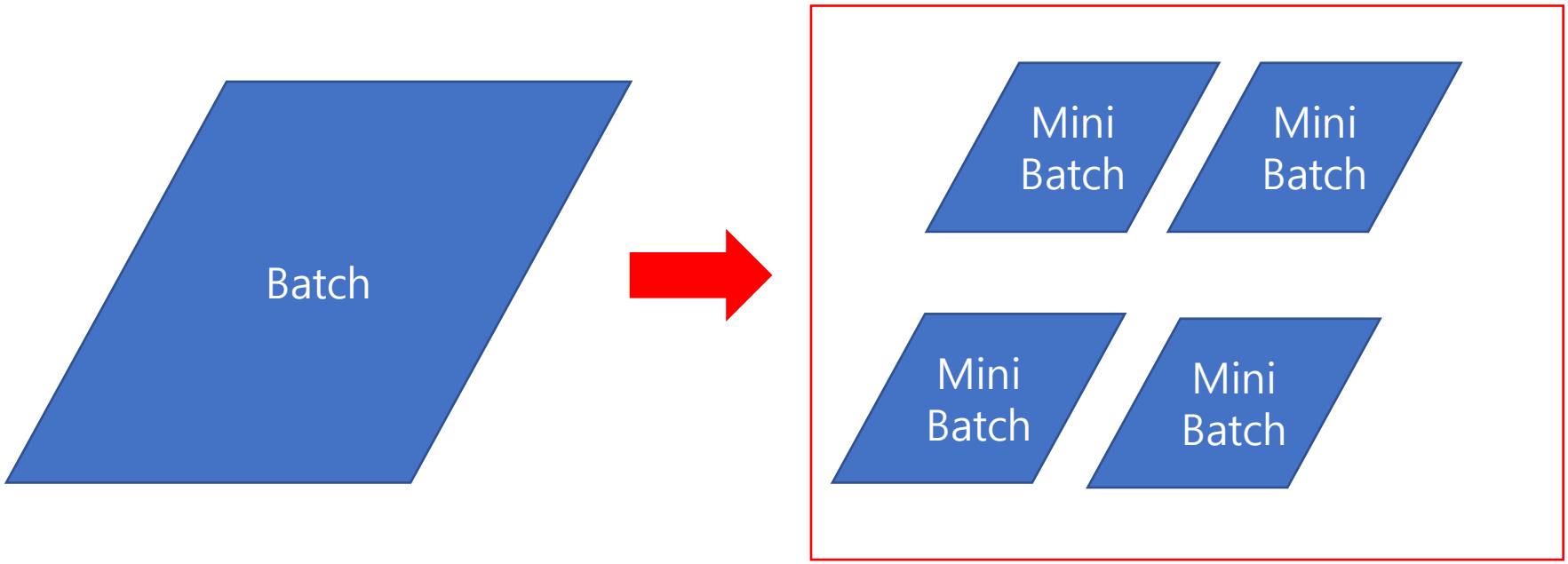
Mini Batch: 여러 개로 나누기

각각에 대한 경사하강 적용

- 배치의 장점을 계승
- 배치 사용 시 소요 시간 > 미니 배치 사용 시 소요 시간
- 여전히 전체 데이터를 학습!

Iteration

- 배치 크기: 각 미니배치에 포함된 데이터의 수



- 전체 데이터 학습->1 에포크

- Iteration: 한 에포크에서 미니배치수를 통한 학습의 횟수, 여기서는 4

IV. 과적합과 드롭아웃

1. 딥러닝과 과적합(Overfitting): 모형이 주어진 훈련데이터에 너무 맞게 만들어진 경우를 과적합이라고 하며, 이 경우 평가데이터에서 오차가 커질 수 있음

2. 과적합 방지를 위한 방안: 과적합 방지를 위해서는 모든 케이스를 갖는 대용량 데이터 사용, 정칙화, 드롭아웃 등을 사용

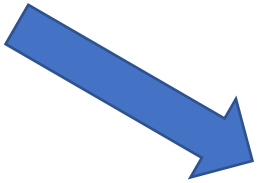
3. 드롭아웃(Drop Out): 인공신경망의 일부 노드를 랜덤하게 사용하지 않는 방식

딥러닝과 과적합

주어진 Training Data에만 잘 맞도록 모델링된 것을 Over Fitting(과적합)이라고 하며, Testing Data나 실제 적용 시 오차가 많이 발생할 수 있음

- **은닉층을 통한 입력 데이터의 비선형 변환**
- **우수한 성능**

- **과적합의 발생**



- **차라리 정말 많은 데이터**
- **가중치 제한(Regularization)**
- **드롭아웃**

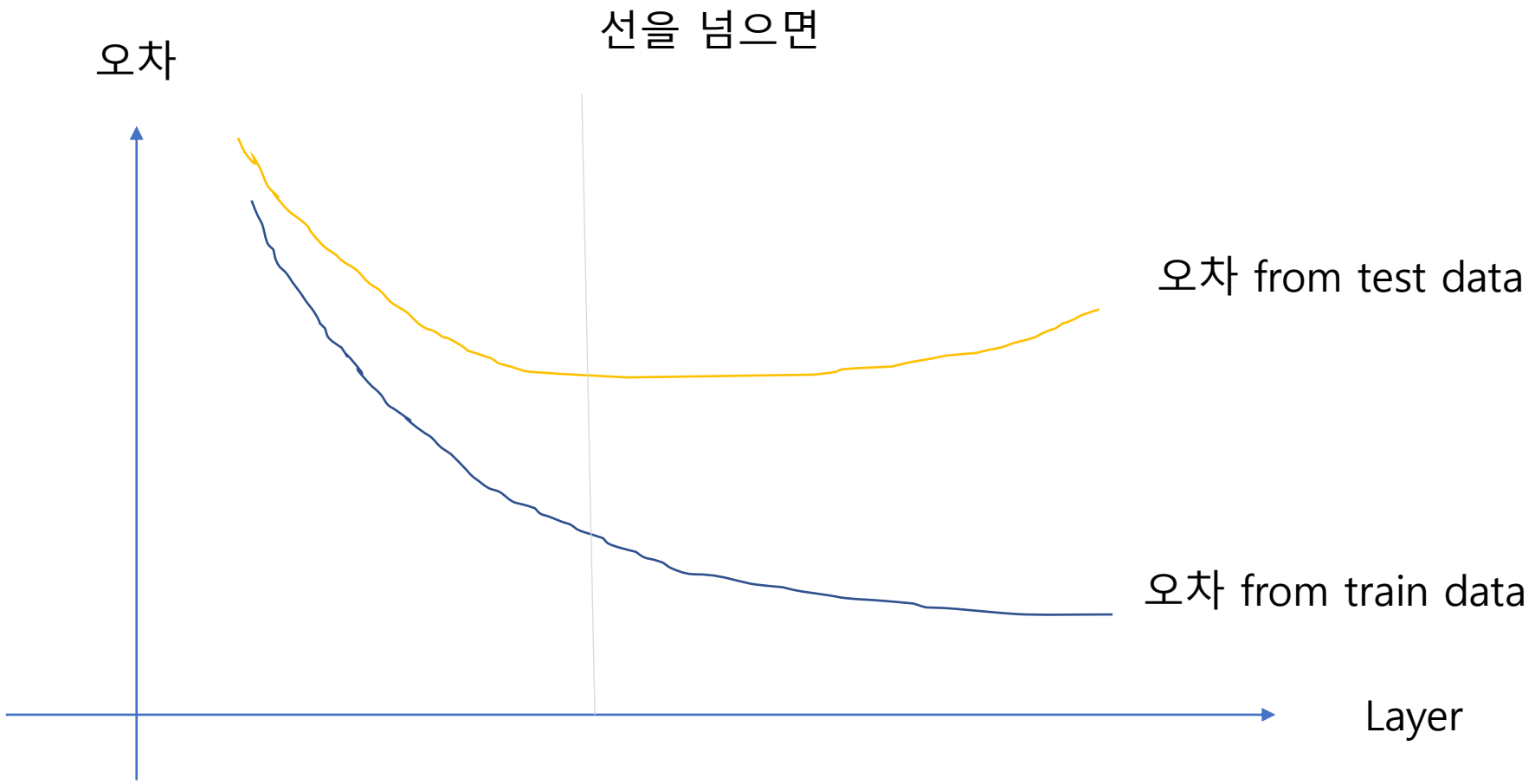
Deep Learning의 문제점: Overfitting!

Layer 1번 = 비선형변환 1번

오차를 줄이는 방향 & Training data에만 과적합!

주어진 Training Data에만 잘 맞도록 모델링

오차와 Overfitting!



Overfitting의 해결책?

- **모형 학습을 위한 더 많은 데이터!**

- **가중치 값에 제한(Regularization):**
  가중치가 너무 큰 값을 갖지 않도록 함!
  큰 가중치 값에 특정 값을 곱해서 해당 가중치 영향을 낮춤

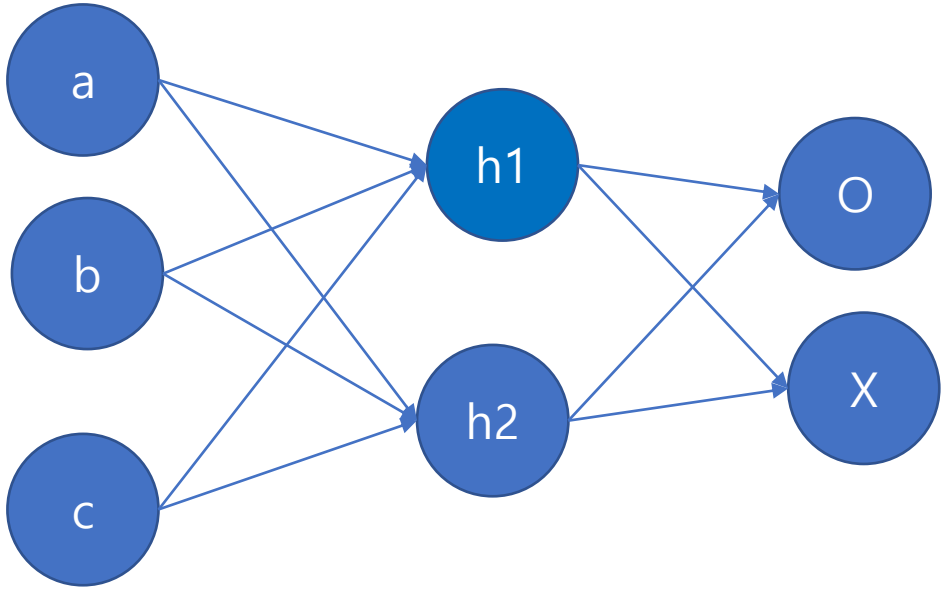
- **Drop Out**

Drop Out

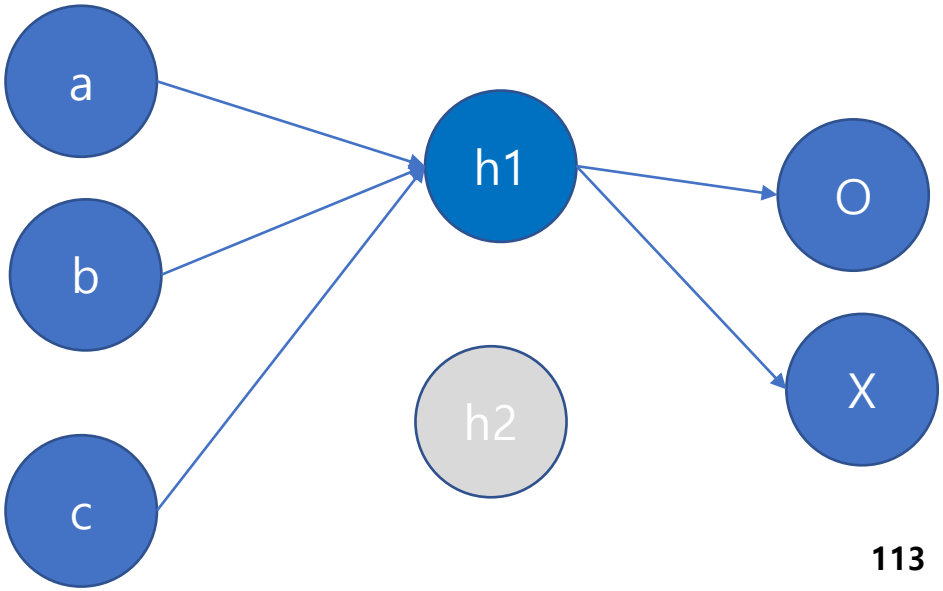
인공신경망의 학습에서 Random하게 은닉층의 특정 노드들을 사용하지 않는 것을 의미하며, Feature 및 은닉층 표현을 풍부하게 하여 성능을 개선할 수 있음

- **인공신경망의 Layer의 일부 Weight만 학습에 사용**
- **무작위로 특정 노드의 값을 0으로 지정!**

Drop Out 아닌 경우



Drop Out



Drop Out: 인공신경망의 Feature표현이 풍부

Drop Out Rate: 0~1

예: Drop Out rate, 0.7

각 weight가 Drop out 될 확률 70%로 지정

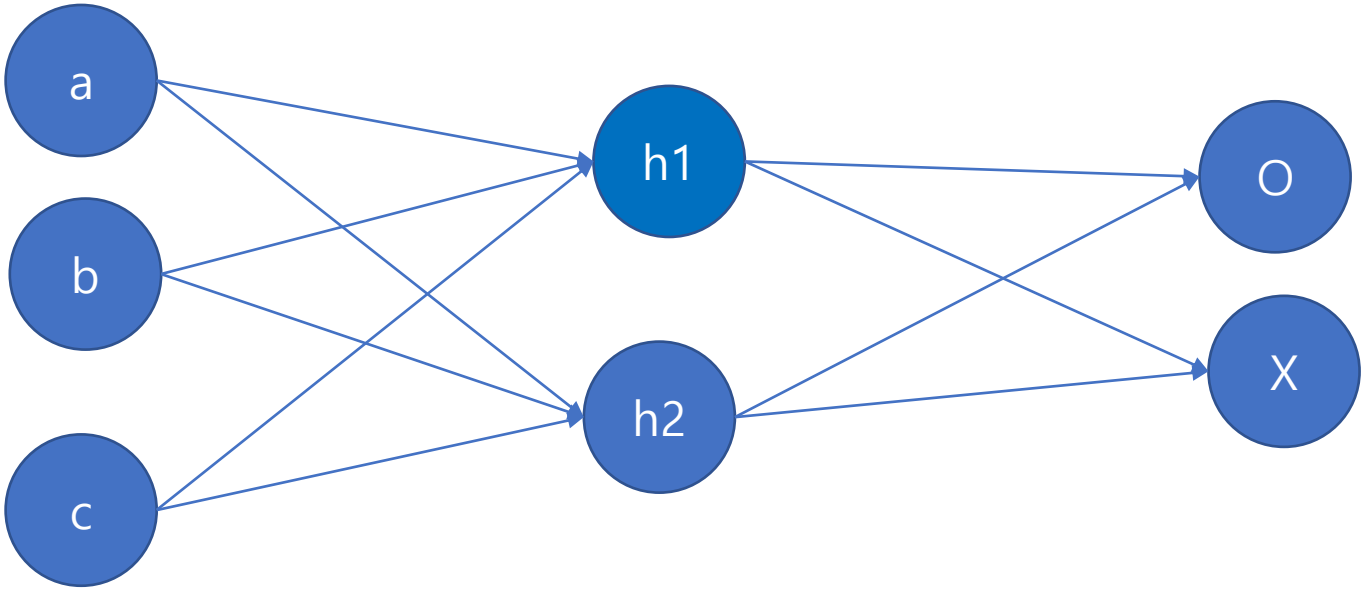
1. Hyper Parameter 튜닝: 학습율(Learning rate), 모멘텀, 은닉층 수와 노드의 수, 미치배치 크기, 최적화 알고리즘 관련 파라미터 등을 조정하여 최적화된 성능을 얻음

2. Grid Search VS Random Search: 모든 경우를 다 탐색하거나, 혹은 Random하게 탐색하는 경우가 있으며, 딥러닝에서는 Random한 하이퍼파라미터 탐색이 유리

3. 한 번에 하나 VS 한 번에 여러 개

하이퍼파라미터:

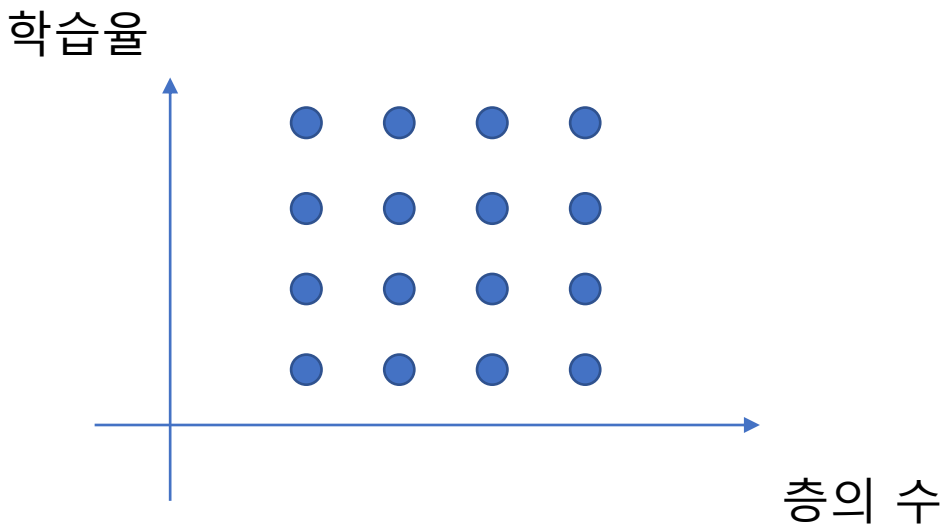
학습율(Learning rate), 모멘텀 파라미터, 은닉층과 노드의 수,
미니배치 크기, 학습율 감쇠 정도, 최적화알고리즘 파라미터 등



Grid Search?

모든 가능한 하이퍼파라미터의 조합을 체계적으로 하나씩 탐색하는 방식

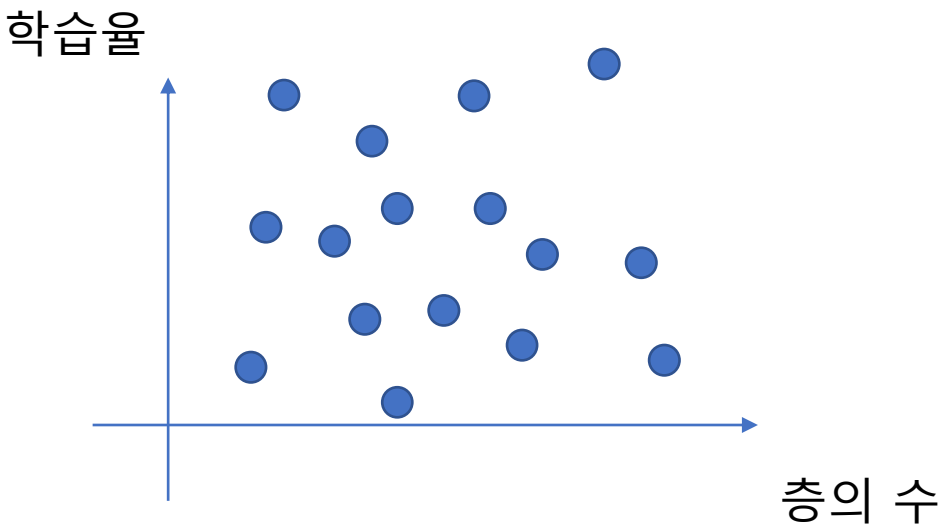
예: 2개의 파라미터의 조합



Random Search?

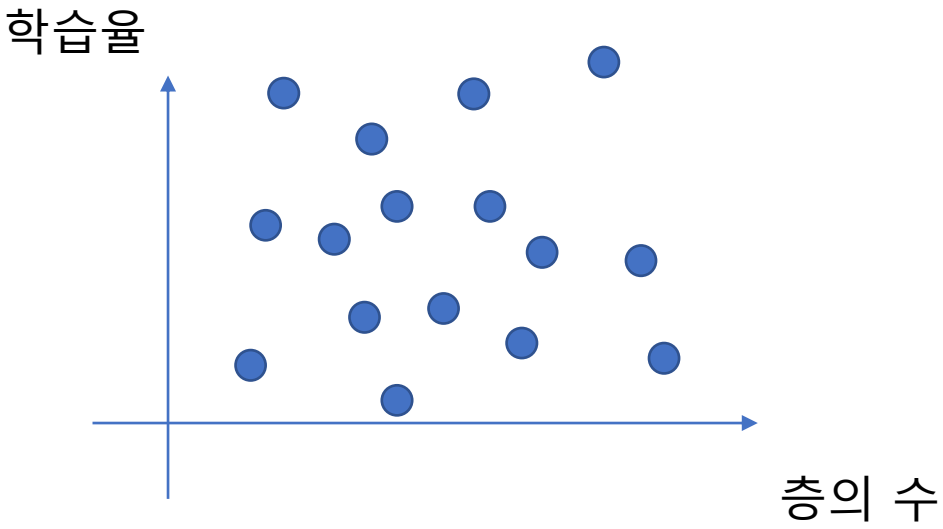
Random하게 하이퍼파라미터의 조합을 탐색

예: Random하게 생성되는, 파라미터 2개에 의한 값의 조합



Grid Search보다 우수할 수도!

Random Search?



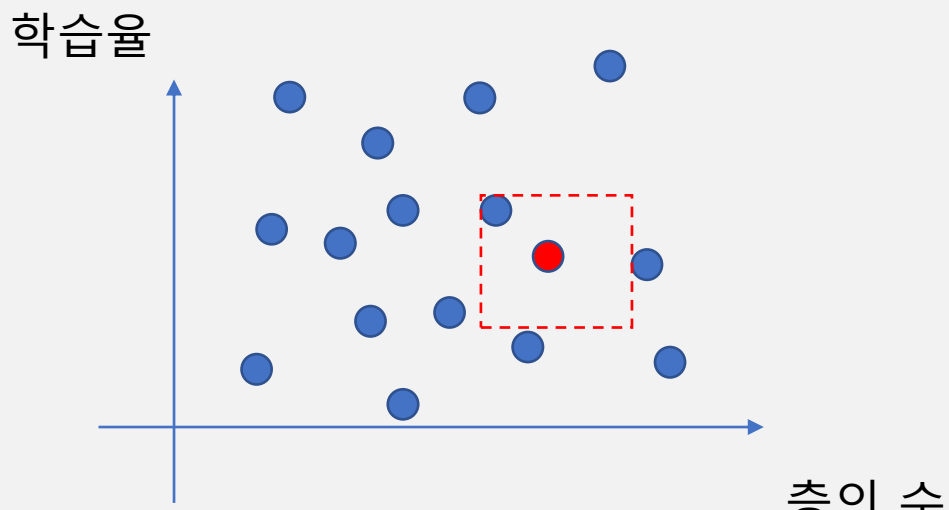
Don't use Grid!
from Andrew Ng

- 왜? 인공신경망에서는 어떤 파라미터가
- 성능에 어느 정도 영향을 주는지를 알 수 없음

- 중요한 하이퍼파라미터 관점에서 먼저 탐색!

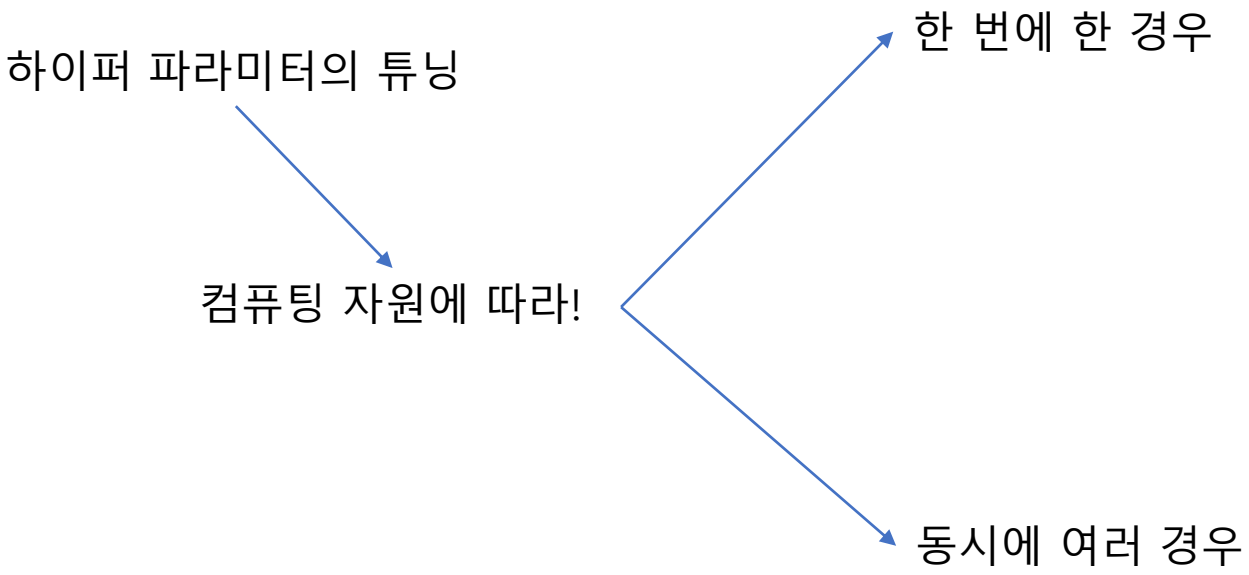
예: 두 조합 중 학습율이 있다면, 학습율 우선으로 고려,
이후 전체 조합의 개수 산출, 다른 파라미터 값을 검색

정밀화 접근!



Random한 탐색에서 발견한 최적의 하이퍼파라미터 주변을 더 탐색!

한 번에 하나! VS 한 번에 여러 개!



VI. 딥러닝의 발전

1. 딥러닝의 발전: ReLU 이후 딥러닝 알고리즘은 지속적으로 발전

2. 다양한 딥러닝 알고리즘: CNN, RNN, GAN, BERT 등 다양한 딥러닝 알고리즘의 등장

3. 딥러닝 적용의 확대: 다양한 분야로의 적용이 확대

인공신경망의 이슈들 (Jeffrey Hinton, 2006)

"Our labeled datasets were thousands of times too small"

"Our computers were millions of times too slow"

"We initialized the weights in a stupid way"

"We used the wrong type of non-linearity"

다양한 딥러닝 기법으로의 발전

입력 처리, 가중합, 활성화함수, 최적 가중치 찾기

CNN

BERT

RNN

GAN

다양한 딥러닝 기법으로의 발전: CNN

CNN

BERT

합성곱 신경망
- Convolutional Neural Network
- Feature를 처리하는 방식의 특징
- Convolution+Pooling을 거침
- 이미지 분류에 활용

RNN

GAN

다양한 딥러닝 기법으로의 발전: RNN

CNN

BERT

RNN

GAN

순환신경망
- Recurrent Neural Network
- 가중치 학습 시 Epoch사이의 순환 구조 포함
  - 이전 Epoch 학습내용을 이번 Epoch 학습 시 활용
- 시계열, 음성, 텍스트 등에 활용

다양한 딥러닝 기법으로의 발전: GAN

CNN

BERT

RNN

GAN

By Ian Goodfellow
- Generative Adversarial Network
- 입력 데이터의 분포를 만드는 모델
  - 원래 입력이 타겟이 되는 신경망
- 적대적? 두 모델을 통해 "진짜같은 가짜 " 입력데이터를 생성
  - Generator VS Discriminator

다양한 딥러닝 기법으로의 발전: BERT

CNN

BERT

- Bidirectional Encoder Representations from Transformers
- Encoder?
- 사전 훈련 언어 모델
- 사전 임베딩 결과가 Transfer+Fine Tune

RNN

GAN

QnA