

Deep Learning

Kim Hye Kyung

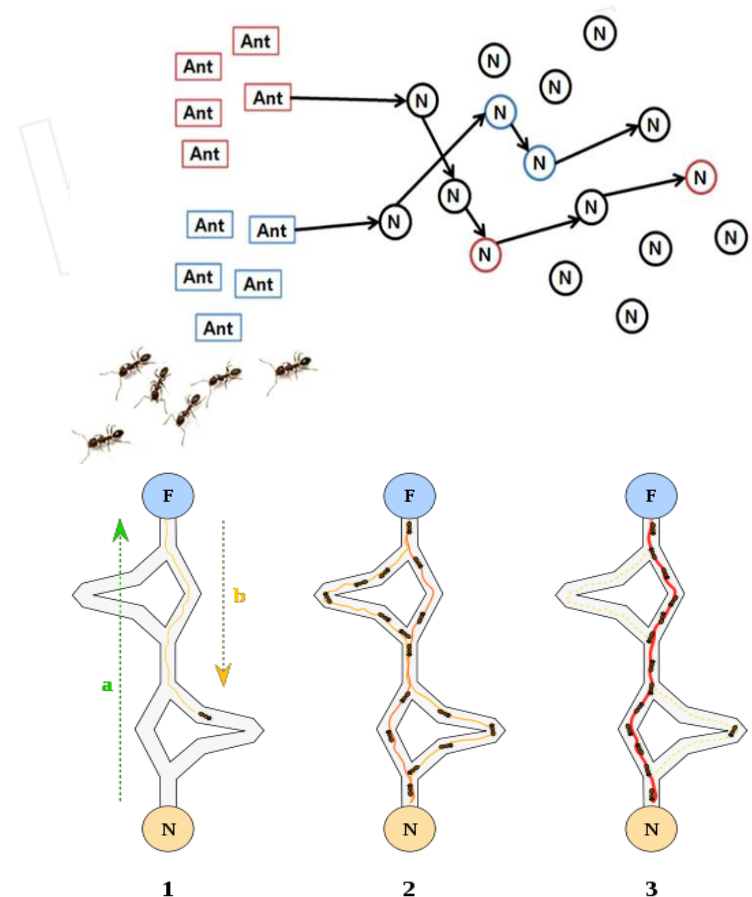
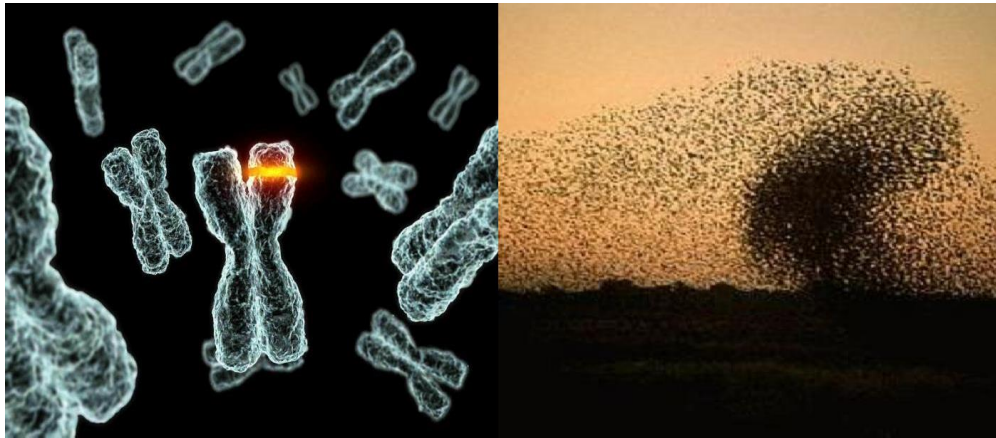
topickim@naver.com

- I 딥러닝
- II 개발 환경 구축
- III 딥러닝 동작 원리
- IV 선형 회귀로 본 딥러닝 필요 이론
- V 경사 하강법
- VI 로지스틱 회귀
- VII 오차 역전파[Back Propagation]
- VIII 신경망에서 딥러닝으로
- IX Keras
- X 컨볼루션 신경망(CNN)
- X I 순환 신경망(RNN)

딥러닝

왜 인공지능망인가?

- 최적화의 많은 알고리즘은 자연 현상의 관찰을 토대로 개발됨
 - 유전자 알고리즘, 개미군집 알고리즘, particle swarm optimization 등



인공지능 vs. 머신러닝 vs. 딥러닝

- 인공지능 vs. 머신러닝 vs. 딥러닝

- 인공지능

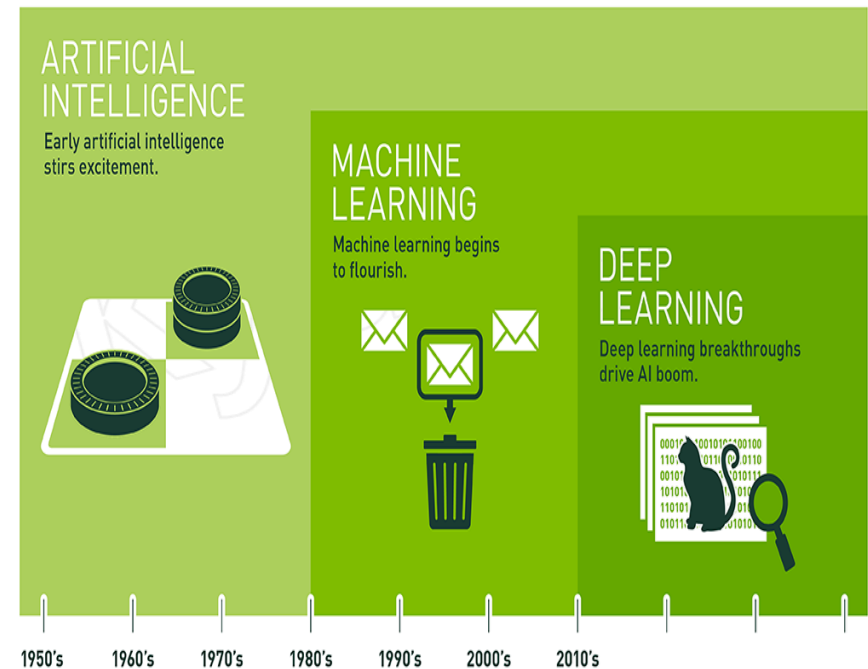
- 특정 분야를 지칭하는 것이 아닌, 지능적 요소가 포함된 기술을 총칭

- 머신러닝

- 학습 전용 데이터에서 규칙성 등을 '학습' 하고 미지의 데이터를 판별할 수 있는 알고리즘

- 딥러닝

- 심층 신경망을 이용한 머신러닝 기법
 - 머신러닝의 기초 필수
 - 수술 환자의 사망률 예측
 - 손으로 쓴 글씨 판별
 - 꽃 품종 판별



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>

딥러닝

- 딥러닝이란?

겉으로 드러나지 않는 '미니 판단 장치' 들을 이용해서 복잡한
연산을 해낸 끝에 **최적의 예측 값을 내놓는 작업**

딥러닝 개요

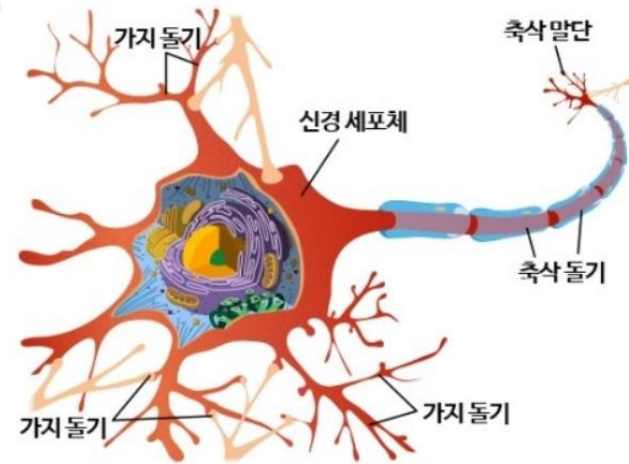
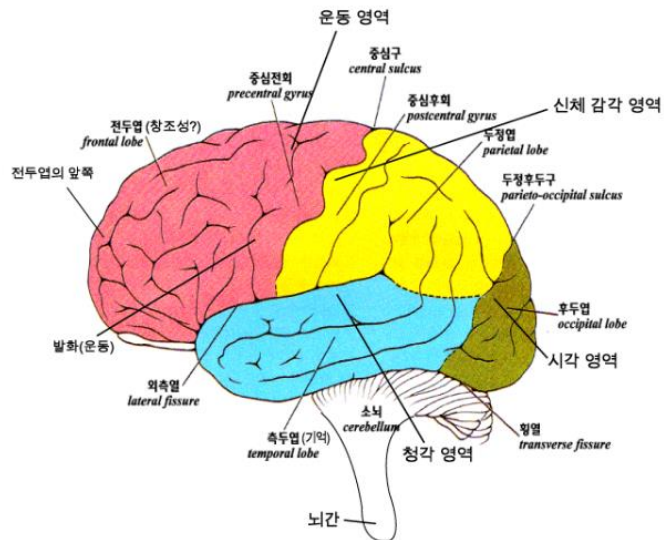
- 딥러닝이란?
 - 머신러닝 방법 중에 하나인 신경망 중에서 층(Layer)을 깊게(Deep) 만든 것을 의미
- 신경망(Neural Network)이란?
 - 신경 세포의 기능을 인공적으로 시뮬레이션한 것
 - 인공 신경망이라고도 함
- 비구조화 데이터를 쉽게 처리 할수 있음
 - 비구조화 데이터란?
 - 이미지 또는 음성처럼 특징이 "사람의 감각으로 파악 할 수 있는 데이터"
 - 가령 사진을 보면 고양이 모습? 목소리는 어떤 사람의 목소리? 등을 알 수 있게 해 주는 특징을 가진 데이터
 - 구조화 데이터란?
 - 털의 길이가 5cm, 주파수가 몇 Hz 처럼 수치를 엑셀과 같은 표로 정리할 수 있는 데이터 의미

딥러닝의 역사

시대	흐름
제 1차 인공지능 부흥기 (1950~1970년대)	<p>손으로 했던 작업과 계산을 자동으로 실행하는 규칙을 컴퓨터에게 적용하는 규칙 기반(가령, A이면 B이다 와 같은 규칙) 인공지능 연구가 활발</p> <p>딥러닝의 기본 단위인 신경세포(뉴런)의 기초 이론 활성화 퍼셉트론 발표(1958년 프랭크 로젠블라트, 신경세포 구조를 모방해 여러 개의 뉴런을 결합해서 복잡한 문제에 대응하는 지능 탄생 시도)</p> <p>XOR 문제를 풀수 없다는 사실을 증명하면서 점차 침체기에 돌입(마빈 마스크)</p>
제 2차 인공지능 부흥기 (1980년대)	<p>하드웨어 기술의 발전으로 수많은 데이터 축적이 가능해지고 정보를 고속으로 탐색 가능</p> <p>XOR 문제 해결 - 오차 역전파법(back-propagation)을 이용해서 다층 퍼셉트론을 이용한 해결법 제시 그러나 층을 쌓일수록 성능이 떨어지는 경사 소실 문제와 지역 최적해 문제 및 현실적인 시간내에 학습 불가능한 문 발생으로 인한 관심밖의 대상이 됨</p>
1990년대	컴퓨터 성능 향상과 더불어 지프리 힌튼의 연구로 4층 이상의 심층 신경망에서 지역최적화와 경사 소실 문제 해소
2012년	이미지 인식 성능 대화인 국제 대회 ILSVRC(ImageNet Large Scale Visual Recognition Challenge)에서 심층 신경망을 이용한 이미지 인식 성공률 입증
제 3차 인공지능 부흥기	오토인코더를 이용한 이미지 자동 생성, 자연어 생성, 자동 번역, 로봇 제어등 여러 분야에 이용되면서 인공 지능 부흥기

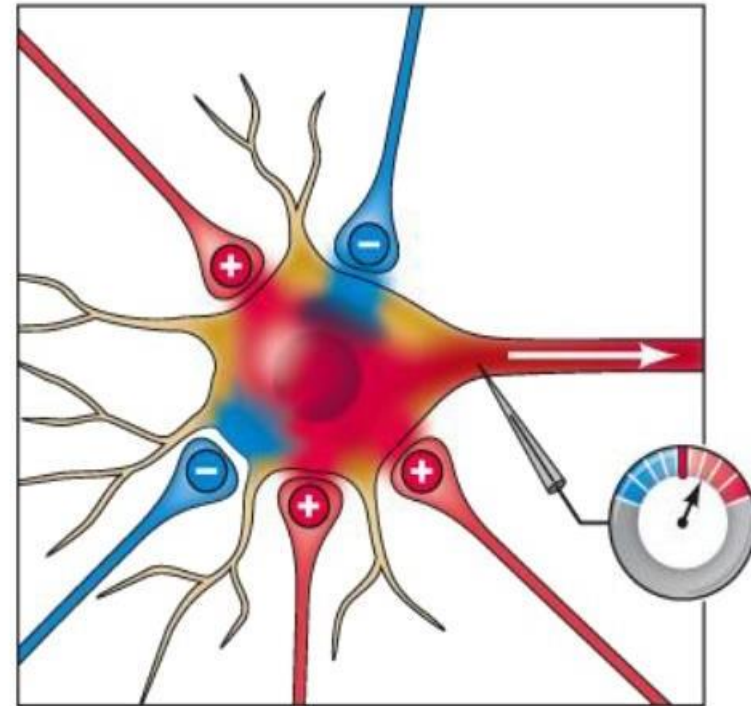
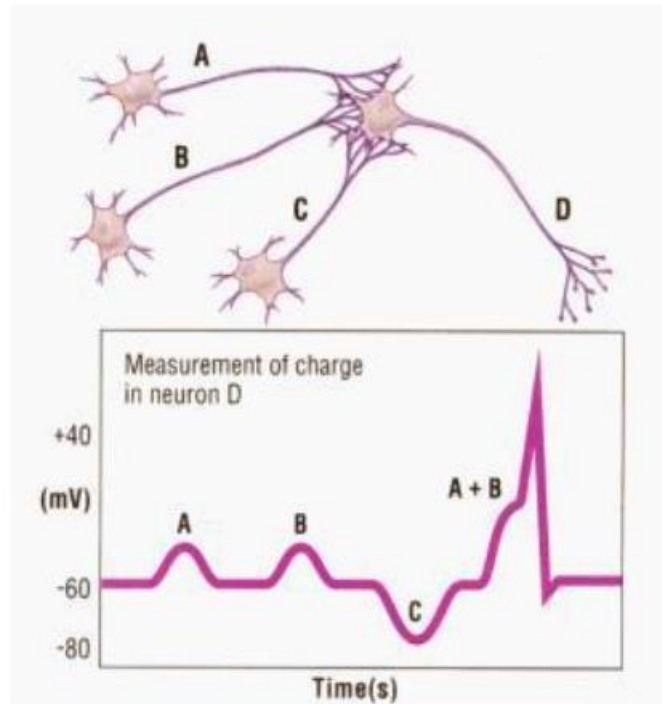
우리 뇌가 정보를 처리하는 방식

- 우리의 뇌가 작동하는 방식
 - 외부의 자극이 감지되면 자극 감지 기관으로부터 뇌의 특정 부분까지의 신경계가 활성화되면서 해당 정보를 처리
 - 신경세포들은 전기적 신호를 이용하여 메시지를 주고받음



우리 뇌가 정보를 처리하는 방식

- 뉴런의 작동 방식



뉴런은 계속해서 시그널을 받아
- 그 것을 조합- '*sum*' 하고,
- 특정 threshold 가 넘어서면 - '*fire*' 를 한다.

신경망

- 신경망이란?
 - 머신러닝 분야에서 연구되는 알고리즘의 하나
 - 이러한 뉴런의 작용을 수학적으로 추상화해 일정 단위의 인위적인 네트워크로 표현한 것
 - 뇌를 형성하는 뉴런의 집합체를 수학 모델로 나타내는 것이 신경망의 출발점

신경망과 딥러닝

- 생물학에서의 뉴런(신경세포)

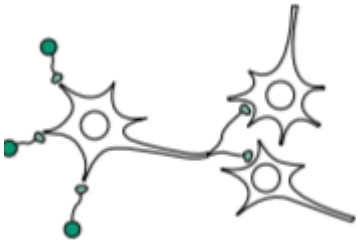
1. 뉴런은 네트워크를 형성

2. 여러 뉴런에 전달되는 신호의 합이 일정 크기(임계값)를 넘지 않으면, 뉴런은 전혀 반응하지 않음

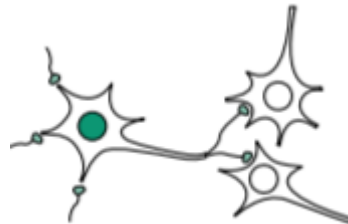
3. 여러 뉴런에 전달되는 신호의 합이 일정 크기(임계값)를 넘으면 뉴런이 반응하며, 다른 뉴런에 일정 강도의 신호를 전달

4. 2와 3에서 여러 뉴런에 전달되는 신호의 합은 신호마다 가중치가 다름

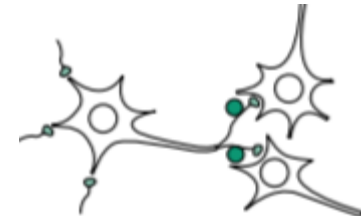
뉴런에 신호를 입력



세포체는 신호의 합 크기를 판단

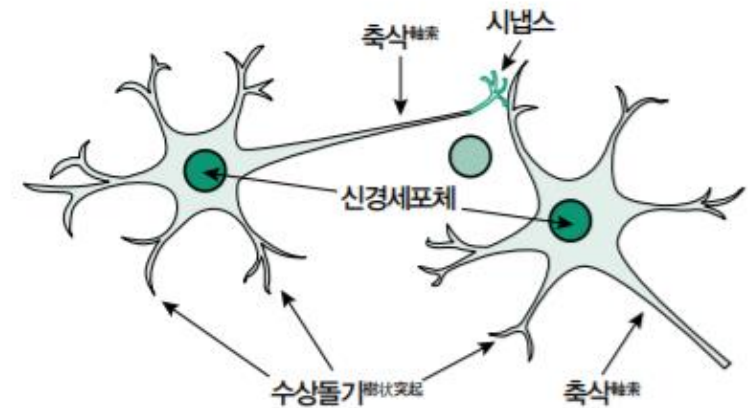


신호의 합이 임계값보다 크면 뉴런이 반응해서 근처 뉴런에 신호 전달



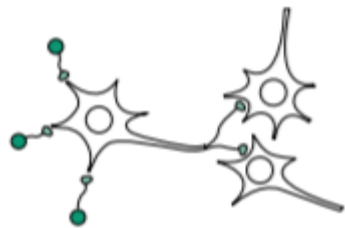
뉴런의 기능

- 사람의 뇌 속에는 다수의 뉴런(신경세포)이 네트워크 형성
- 뉴런 하나가 다른 뉴런에게 신호를 받거나 보낸다는 의미
- 뉴런의 모식도
 - 신경 세포체, 축삭, 수상 돌기로 구성
- 뉴런의 정보 전달 방법
 - 수상돌기 : 다른 뉴런에게 정보를 받는 돌기
 - 축삭 : 다른 뉴런에게 정보 발송하는 돌기
 - 수상돌기가 받은 자기 신호는 신경 세포체에서 처리 한후 축삭을 지나 다음 뉴런에게 전달
 - 뉴런은 시냅스를 매개로 결합해 네트워크를 형성

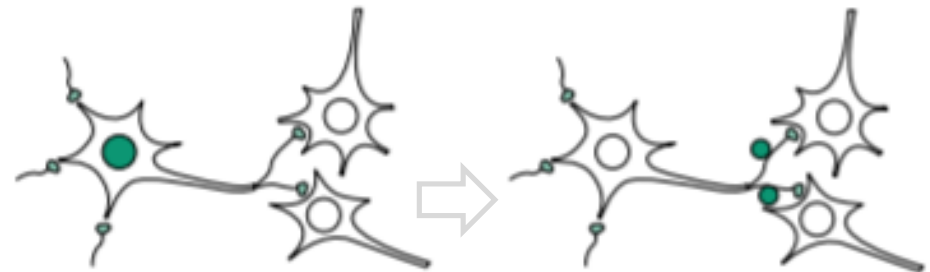
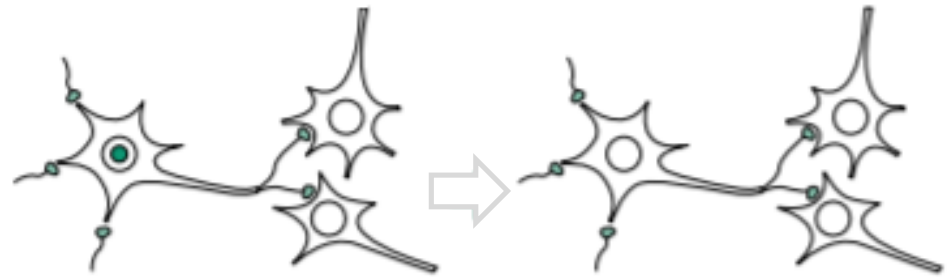


뉴런의 동작 원리

신경 세포는 신호를 합해서 신호의 합이 임계(경계)값보다 작을 경우 무시



뉴런에 신호 입력



신경 세포는 신호를 합해서 신호의 합이 임계값보다 클 경우 반응해 옆 뉴런에 신호 전달

뉴런의 활동을 수학으로 표현하기

- 뉴런의 반응 구조

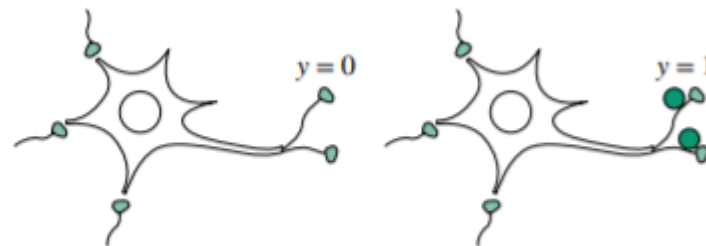
- 다른 여러 뉴런의 "신호 합" 이 뉴런의 입력
- "신호 합" 이 뉴런 고유의 임계값보다 크면 반응
- 뉴런의 출력 신호는 반응했는지를 0과 1의 디지털 신호로 표현
- 복수의 출력 신호가 있더라도 반응 여부를 0과 1이라는 값으로 표현

- 수학적 표현으로 바라본 반응 구조

- 입력 신호 변수 = x 라 가정
 - 입력 신호 없음 : $x = 0$
 - 입력 신호 있음 : $x = 1$



- 출력 신호 변수 = y 라 가정
 - 출력 신호 없음 : $y = 0$
 - 출력 신호 있음 : $y = 1$



수식으로 표현해 본 뉴런의 반응 여부

- 뉴런의 반응 여부

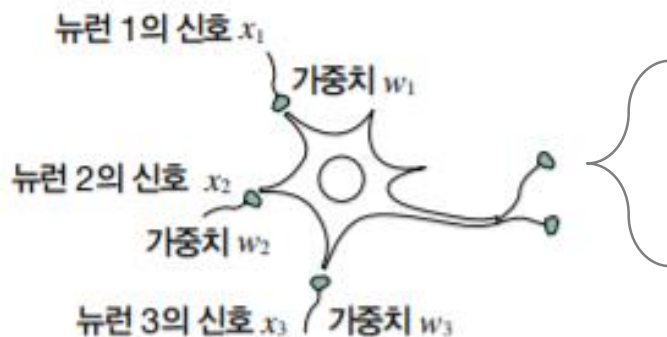
- 뉴런의 입력 신호의 합으로 판단

$$w_1x_1 + w_2x_2 + w_3x_3$$

- 식

입력 신호 : x_1, x_2, x_3

입력 신호의 가중치 : w_1, w_2, w_3

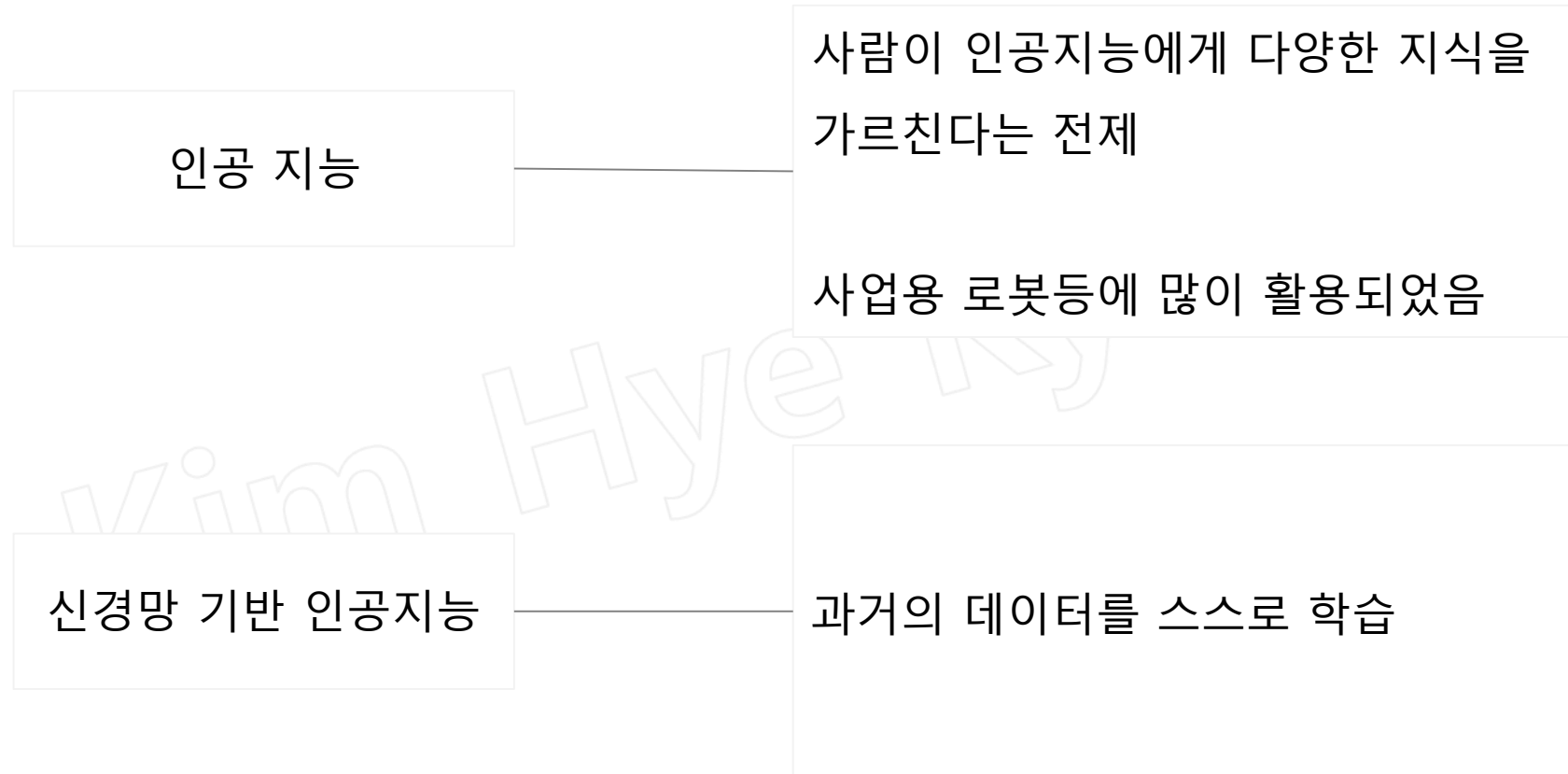


θ : 뉴런 고유의 임계값

출력 신호 없음($x=0$) : $w_1x_1 + w_2x_2 + w_3x_3 < \theta$

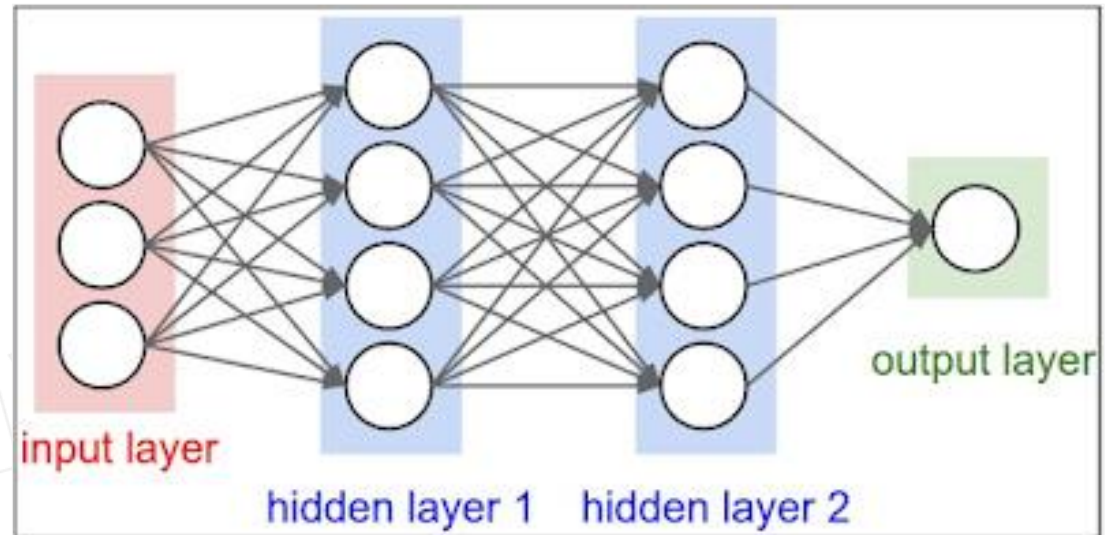
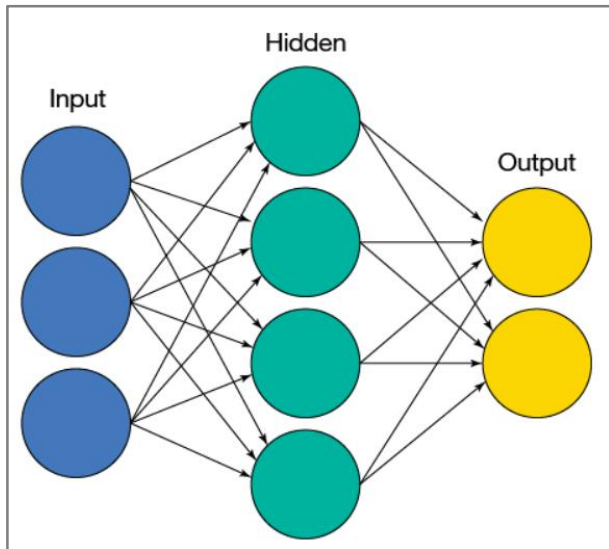
출력 신호 있음($x=1$) : $w_1x_1 + w_2x_2 + w_3x_3 \geq \theta$

기존 인공지능과 신경망 기반 인공지능의 차이



신경망 구분

- 일반적인 계층형 신경망과 딥러닝



- 기본 신경망
 - 입력층, 중간층(은닉층), 출력층의 3층 구조 네트워크로 구성된 기본 구조
- 딥러닝
 - 4개층 이상으로 깊은 네트워크 구조로 구성된 구조

신경망

입력층

신경망에 할당하는 입력 정보를 가져옴
데이터에서 얻은 값을 그대로 출력하는 단순한 유닛

은닉층

신경망에서 실제로 정보를 처리하는 부분

출력층

중간층과 마찬가지로 처리하면서 신경망에서 계산한 결과를 출력
신경망 전체의 출력이기도 함

| 개발 환경 구축

딥러닝 작업 환경 구축하기

- 아나콘다 설치
- 가상환경 구축
- Tensorflow 설치

Kim Hye Kyung

Tensorflow & Keras 설치

- 아나콘다 신버전에 설치 불가
- Python 버전을 3.6으로 변경 후 설치해야 함
- 참고 사이트
 - <https://www.tensorflow.org/install/pip>
- Python 가상 환경에 설치하기

```
>conda create -n my_tensorflow pip python=3.6
```

```
>conda activate my_tensorflow
```

```
>pip install tensorflow
```

```
>pip install keras
```

```
>pip install jupyter
```

가상 환경 구축

- 단계 : 가상 환경 설치
 - (base) > conda create -n my_tensorflow python=3.6 anaconda
- 2단계 : 가상 환경 활성화
 - (base) > conda activate my_tensorflow
 - (my_tensorflow) > python --version
- 3단계 : python package install
 - (my_tensorflow) > pip install --upgrade numpy matplotlib pillow
 - pillow 이미지 처리 라이브러리
- 4단계 : 가상 환경 비활성화
 - my_tensorflow > conda deactivate

```
# Requires the latest pip
$ pip install --upgrade pip

# Current stable release for CPU-only
$ pip install tensorflow

# Or preview build for CPU/GPU (unstable)
$ pip install tf-nightly
```

가상 환경 구축

```
C:\WINDOWS\system32\cmd.exe - conda create -n encorevir python=3.5 anaconda
```

```
Microsoft Windows [Version 10.0.17134.345]  
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Users\Playdata>conda create -n encorevir python=3.5 anaconda  
Solving environment: done
```

```
## Package Plan ##
```

```
environment location: C:\Users\Playdata\AppData\Local\Continuum\anaconda3\envs\encorevir
```

```
added / updated specs:
```

- anaconda
- python=3.5

```
The following NEW packages will be INSTALLED:
```

```
alabaster: 0.7.10-py35h3a808de_0  
anaconda: 5.2.0-py35_3  
anaconda-client: 1.6.14-py35_0  
anaconda-project: 0.8.2-py35h06aeb26_0  
asn1crypto: 0.24.0-py35_0  
astroid: 1.6.3-py35_0  
astropy: 3.0.2-py35h452e1ab_1  
attrs: 18.1.0-py35_0  
babel: 2.5.2-py35_0
```

```
zeromq: 4.2.5-hc6251cf_0  
zict: 0.1.3-py35hf5542#  
zlib: 1.2.11-h8395fce_#
```

```
Proceed ([y]/n)? y
```

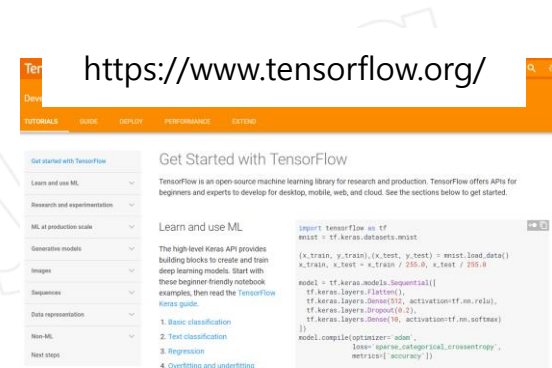
```
Preparing transaction: done
```

```
Verifying transaction: /
```

```
C:\Users\Playdata\AppData\Local\Continuum\anaconda3\envs\encorevir  
DEBUG menuinst_win32:create(320): Shortcut cmd is C:\Users\Playdata\AppData  
re ['C:\Users\Playdata\AppData\Local\Continuum\anaconda3\cwp.py',  
C:\Users\Playdata\AppData\Local\Continuum\anaconda3\envs\encorevir', 'C:\Users\Playdata\AppData\Local\Continuum\anaconda3\envs\encorevir',  
'C:\Users\Playdata\AppData\Local\Continuum\anaconda3\envs\encorevir']  
# To activate this environment, use:  
# > activate encorevir  
#  
# To deactivate an active environment, use:  
# > deactivate  
#  
# * for power-users using bash, you must source  
#
```


TensorFlow

- Google이 2015년에 공개한 머신러닝 오픈소스 Framework
 - 기본적으로 행렬 연산 사용되었으나, 머신러닝 모델링을 간단하게 할 수 있는 다양한 고수준 API 제공
 - 데이터 = Tensorflow, 데이터 흐름 = Graph 로 표현
 - 구성
 - 모델 정의 부분
 - 실제 계산 부분이 분리
- process
 - 기본적인 연산 정의 -> 정의한 연산으로 그래프 정의 -> 세션 정의 -> 연산 실행
- 주의사항
 - 윈도우인 경우 64bit 여야만 함



TensorFlow Mechanics

- graph를 build (노드들을 정의)
- sess.run을 통해서 graph를 실행
- graph 속에 있는 값들을 return

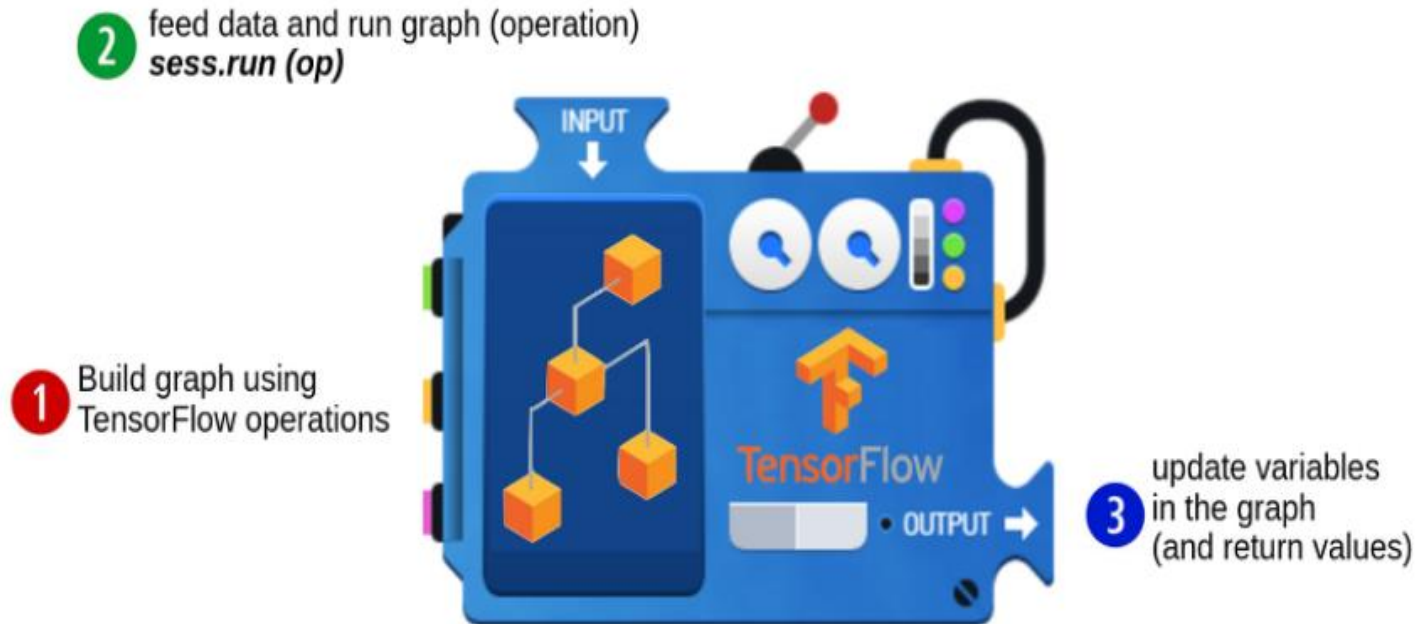
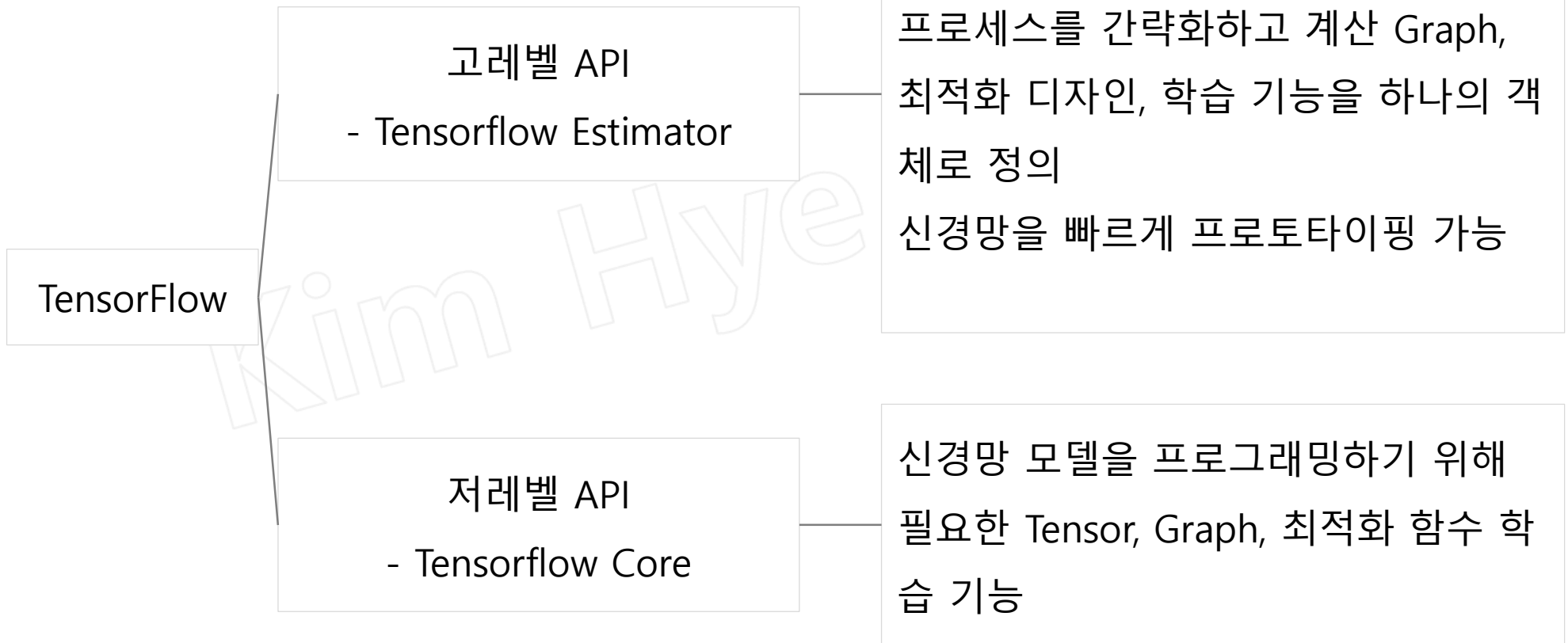


Image 1. TensorFlow Mechanics (src: www.mathwarehouse.com)

TensorFlow

- 자동 미분 기능 이용



TensorFlow 예제

- 튜토리얼 사이트의 예제 활용
 - <https://www.tensorflow.org/tutorials/>
- 구글 콜라보레이터에서 test 연동
- tensorflow 한글문서
 - <https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/g3doc/tutorials/linear/overview.html>

Tensor shapes & types

Rank	Math entity	Python example
0	Scalar (magnitude only)	<code>s = 483</code>
1	Vector (magnitude and direction)	<code>v = [1.1, 2.2, 3.3]</code>
2	Matrix (table of numbers)	<code>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</code>
3	3-Tensor (cube of numbers)	<code>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</code>
n	n-Tensor (you get the idea)	<code>....</code>

Tensor shapes & types

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

1

Tensor shapes & types

Data type	Python type	Description
DT_FLOAT	<code>tf.float32</code>	32 bits floating point.
DT_DOUBLE	<code>tf.float64</code>	64 bits floating point.
DT_INT8	<code>tf.int8</code>	8 bits signed integer.
DT_INT16	<code>tf.int16</code>	16 bits signed integer.
DT_INT32	<code>tf.int32</code>	32 bits signed integer.
DT_INT64	<code>tf.int64</code>	64 bits signed integer.

...

TensorFlow 기본 코드 이해하기 1

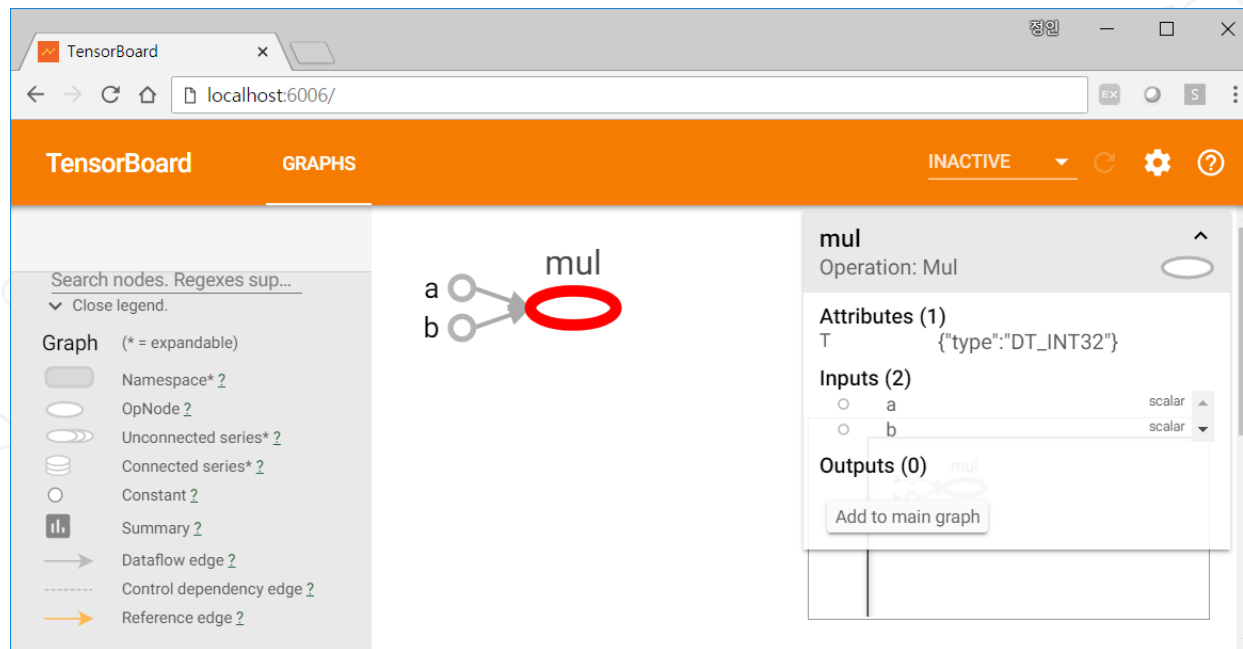
```
1 import tensorflow as tf
2
3 with tf.Graph().as_default():
4     # 상수 정의
5     x = tf.constant(10, name="x")
6     y = tf.constant(20, name="y")
7
8     # 더하기
9     data = tf.add(x, y)
10
11     # 연산 실행
12     with tf.Session() as sess:
13         result = sess.run(data)
14
15 print(result)
```


TensorFlow 기본 코드 이해하기 2

```
1 with tf.Graph().as_default():
2     # placeholder : 외부에서 입력받는 데이터를 활용 가능하게 하는 변수 선언
3     x = tf.placeholder(tf.int32, name="x")
4     y = tf.constant(20, name="y")
5
6     # 더하기
7     data = tf.add(x, y)
8
9     # 더하기 결과를 저장할 변수
10    z = tf.Variable(0, name="z")
11
12    # 더하기 결과를 변수에 넣기
13    data2 = tf.assign(z, data)
14
15    # 변수 초기화
16    init_op = tf.global_variables_initializer()
17
18    # 연산 실행
19    with tf.Session() as sess:
20        # 변수 초기화
21        sess.run(init_op)
22        result = sess.run(data2, feed_dict={x: 10})
23
24 print(result)]
```

TensorBoard

- TensorFlow의 계산과 학습 과정을 시각화 하는 도구
- 계산을 간단하게 모니터링할 수 있게 해주는 도구



TensorBoard

- Tensorboard 실행 명령
 - 1단계 : python 예제 실행
 - 2단계 : 실행 경로에 logdir 디렉토리 존재 여부 확인
 - 3단계 : logdir 폴더 하위에 log 파일 생성 여부 확인
 - 4단계 : 명령어로 tensorboard 실행
 - (my_tensorflow)C:\Windows\system32>tensorboard --logdir="log_dir이 존재하는경로\log_dir"
 - 5단계 : 브라우저 실행
 - <http://localhost:6006>

딥러닝 동작 원리

퍼셉트론

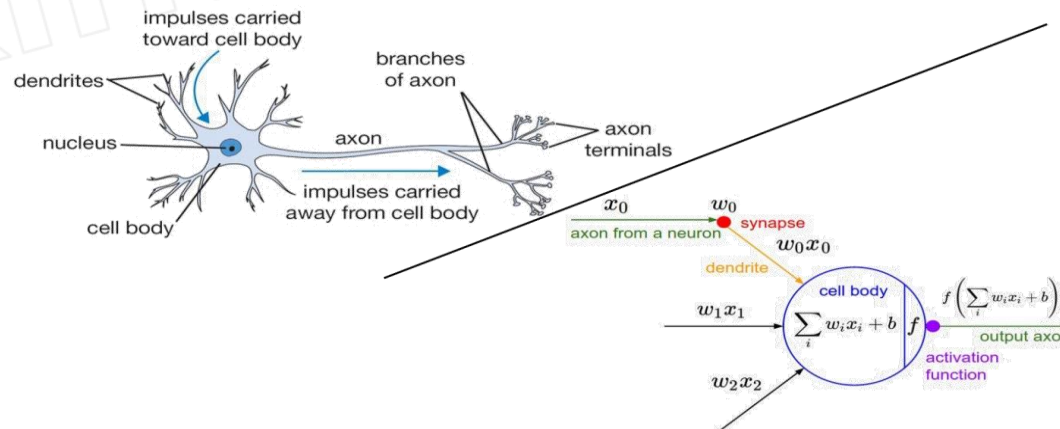
- AI를 위해 고안된 최초의 알고리즘
- 신경세포 구조를 모방해 여러 개의 뉴런을 결합해서 복잡한 문제에 대응하는 지능을 탄생시키려는 시도가 성행하면서 1958년 프랭크 로젠블라트(Frank Rosenblatt)는 신경망의 원형인 퍼셉트론 발표
- 퍼셉트론이란?
 - 뉴런을 연결해서 구성한 원시적인 신경망의 한 종류
 - 최초의 퍼셉트론은 단순 퍼셉트론이라고 부르며 입력층과 출력층의 2층이었지만, 좀 더 복잡한 문제를 풀기 위해 입력층, 중간층, 출력층의 3층으로 만든 것을 다층 퍼셉트론
 - 2-클래스 선형 식별 함수를 구하는 방법

다층 퍼셉트론

- 하나의 퍼셉트론
 - 굉장히 단순한 식별기
 - 복잡한 식별 경계를 만들 수 없음
 - 따라서 선형
- 다층 퍼셉트론
 - 비선형 퍼셉트론
 - 여러 층으로 쌓아 올린 퍼셉트론 의미
 - 신경망(Neural Network)이라고도 함

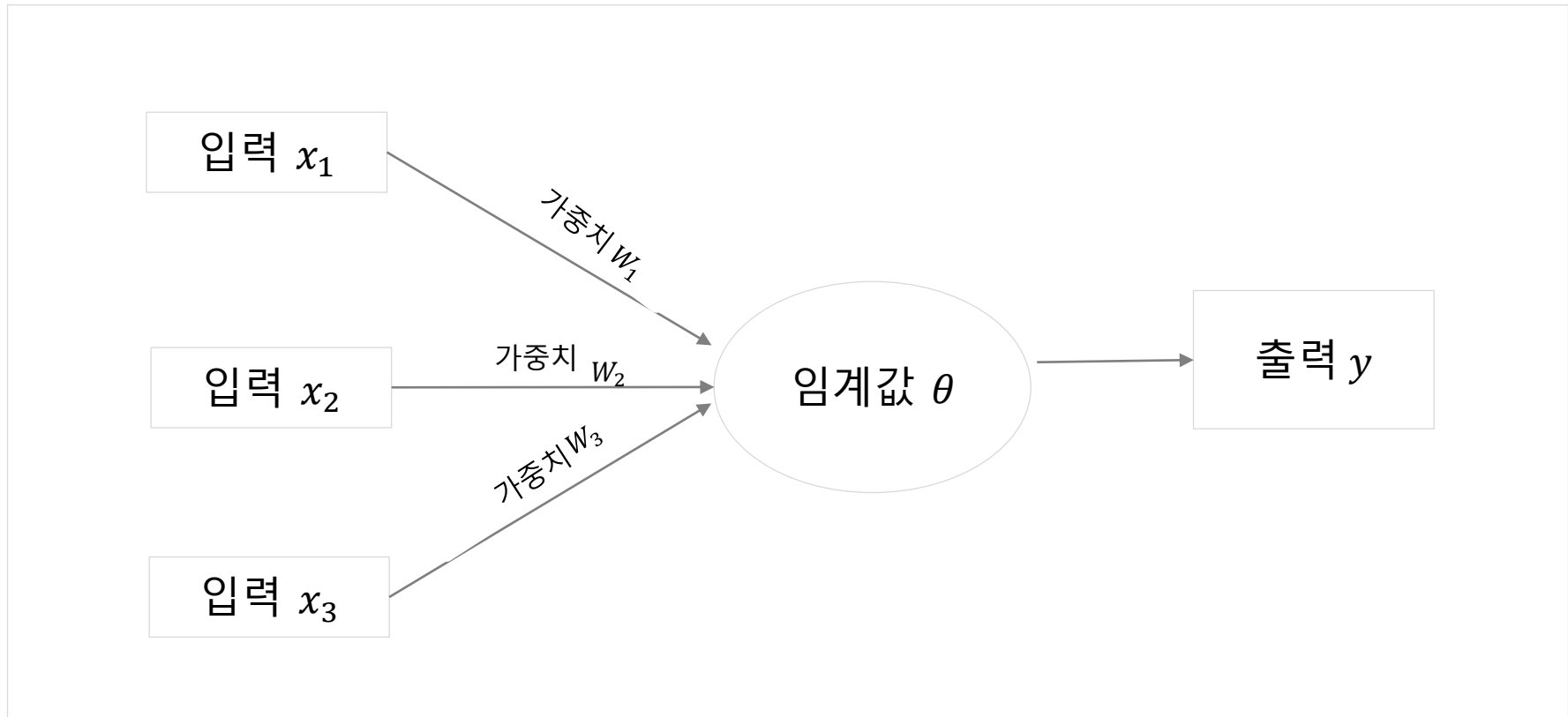
퍼셉트론 (Perceptron): 단층 신경망

- 퍼셉트론의 구조 및 역할
 - 단일 뉴런의 작동 원리를 모사
 - 뉴런: 시냅스로부터 탐지된 자극을 수상돌기를 통해 세포핵에 전달 후 역치를 넘어서는 자극에 대해서는 축색돌기를 이용하여 다른 뉴런으로 정보를 전달
- 퍼셉트론
 - 입력변수의 값들에 대한 가중합에 대한 활성화함수를 적용하여 최종 결과물 생성



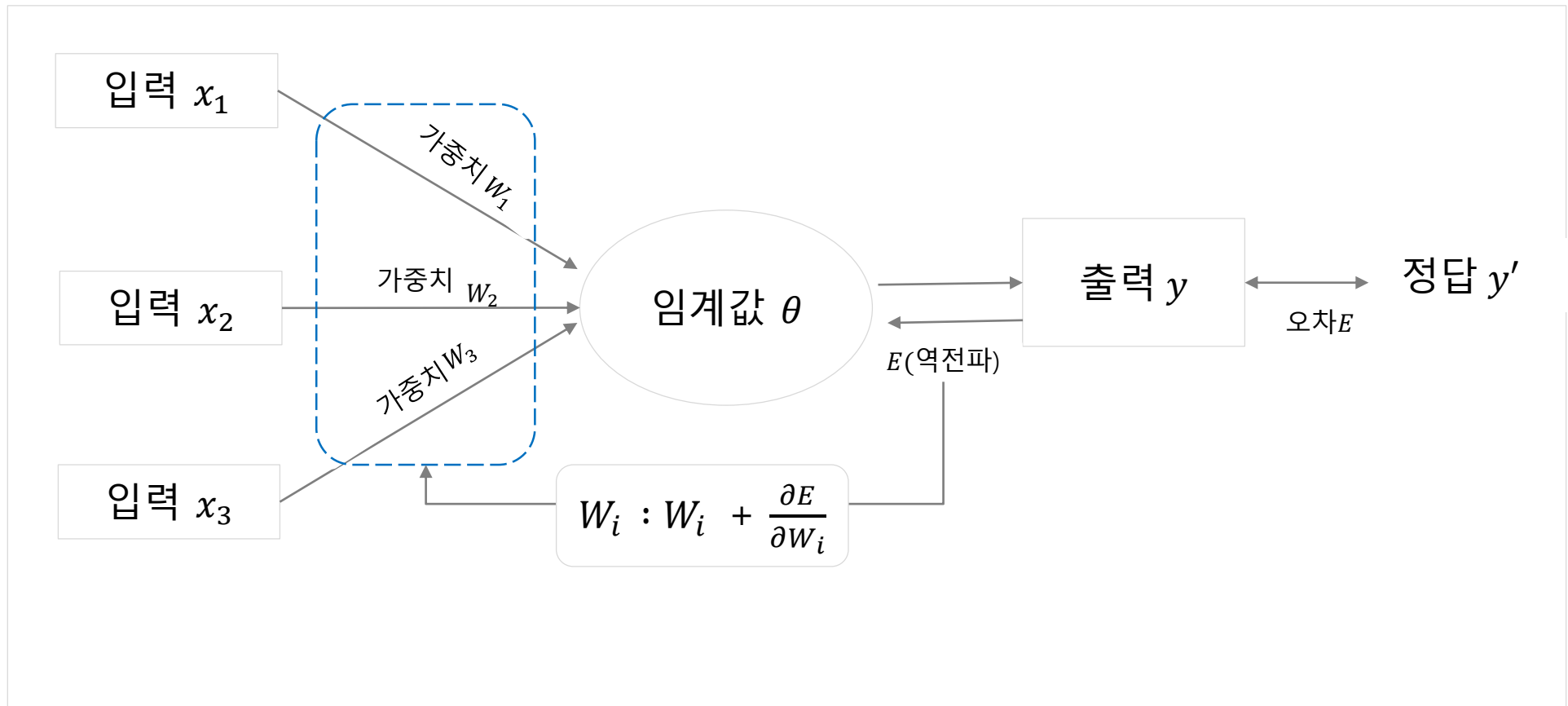
퍼셉트론(단층 신경망) 이해하기

- 가장 간단한 퍼셉트론(뉴런 형식) 구조
 - 1개의 뉴런으로 구성



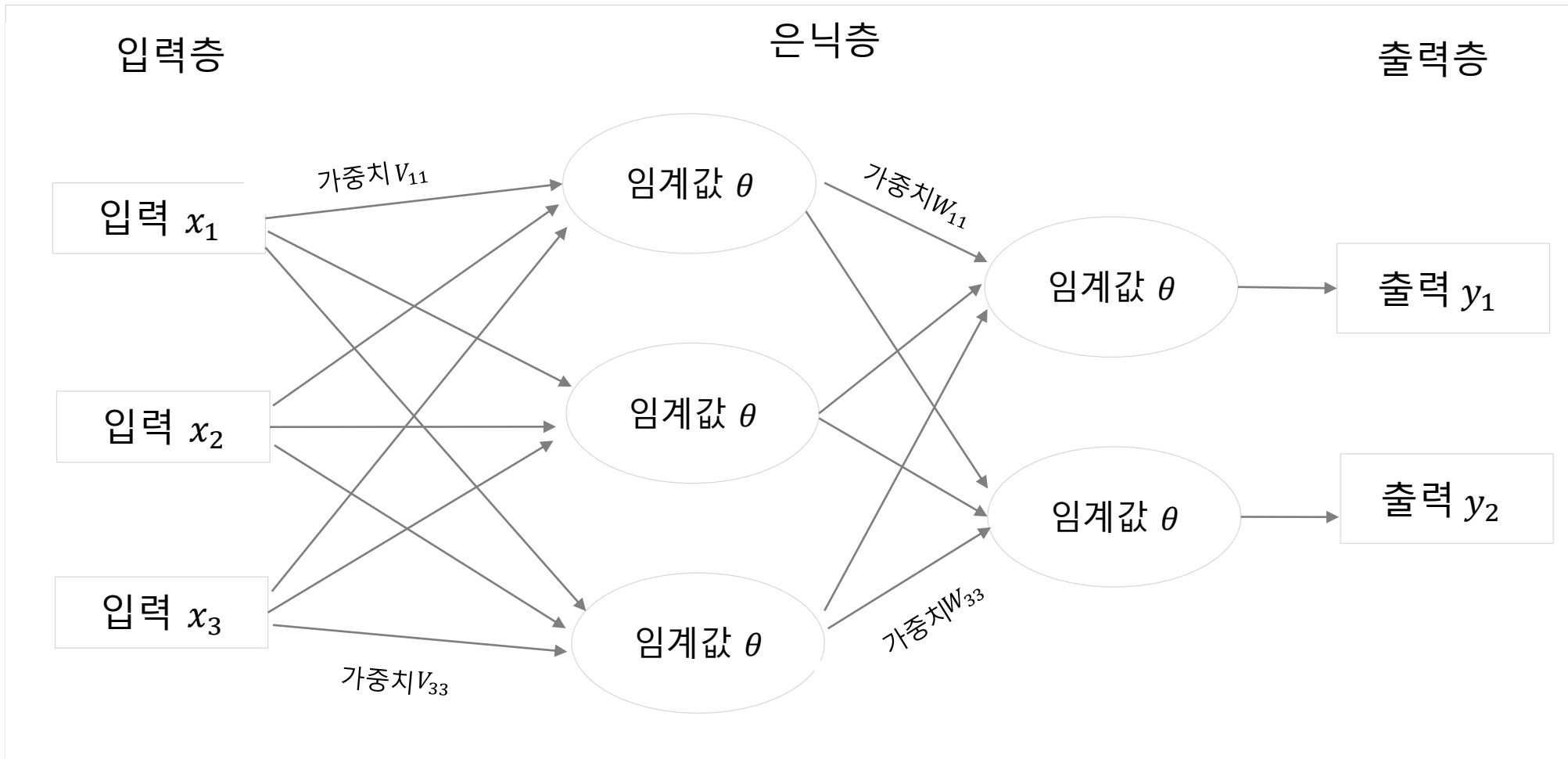
퍼셉트론 이해하기

- 가장 간단한 퍼셉트론(뉴런 형식)



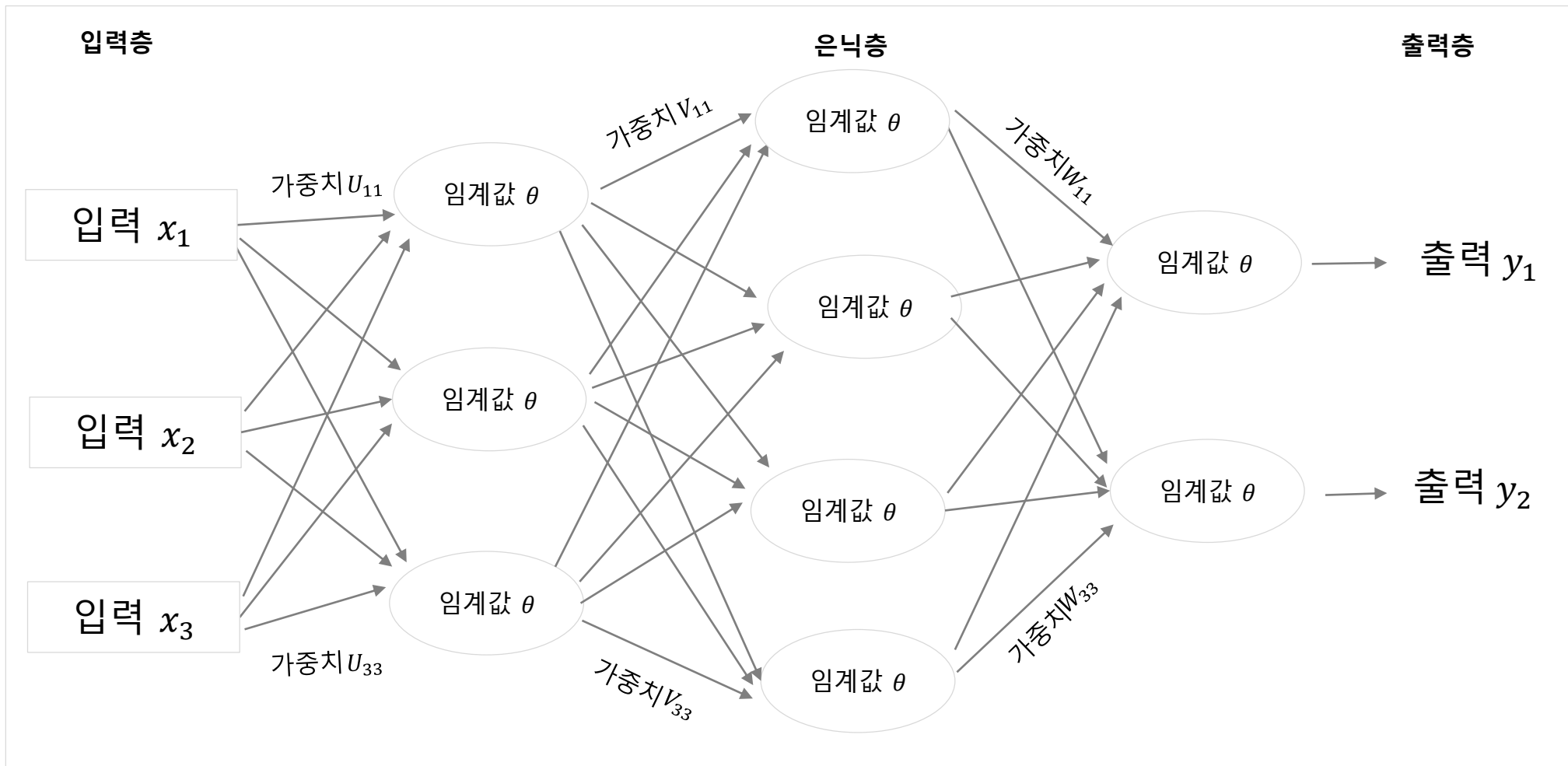
퍼셉트론 이해하기

- 단순 퍼셉트론(2층 퍼셉트론)



퍼셉트론 이해하기

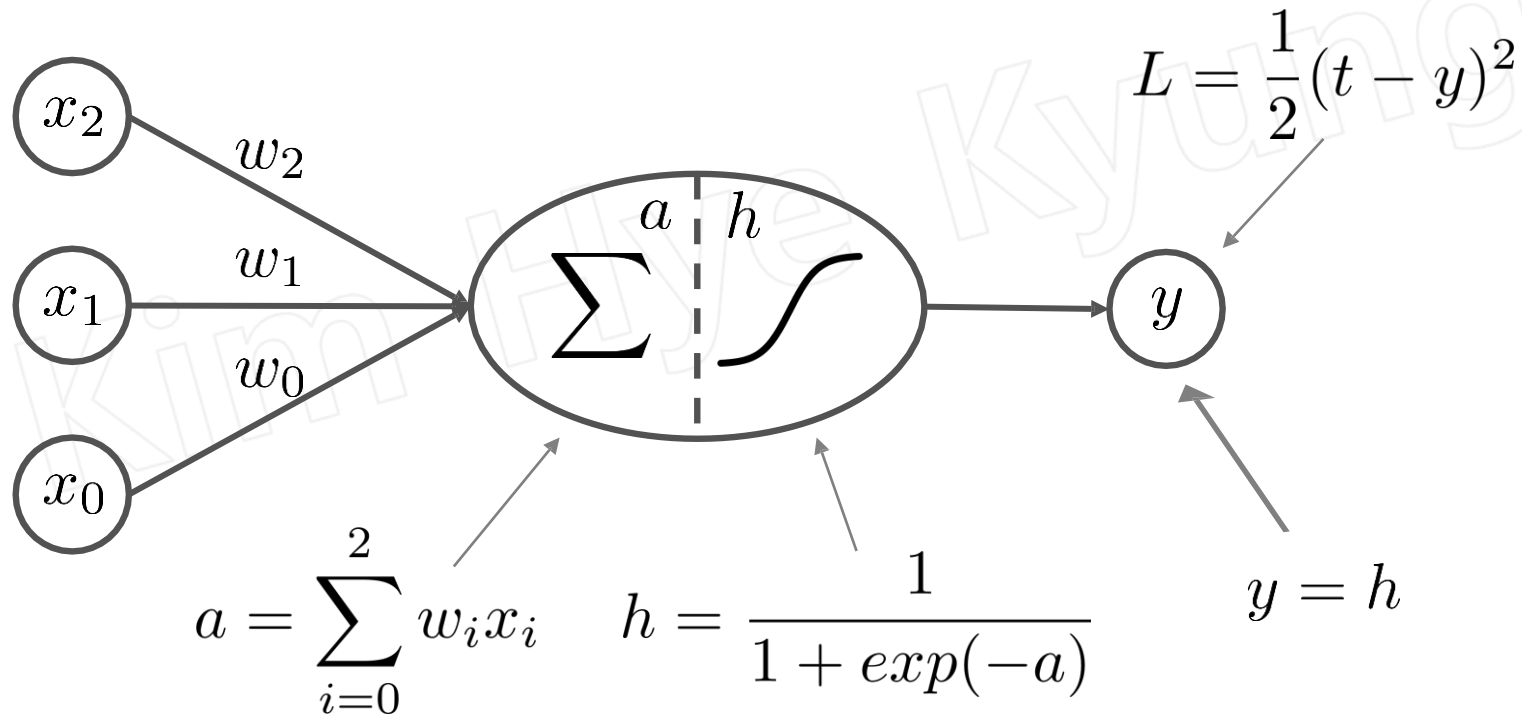
- 다층 퍼셉트론(3층 퍼셉트론)



퍼셉트론(단층 신경망) 이해하기

- 가장 간단한 퍼셉트론(뉴런 형식) 구조
 - 1개의 뉴런으로 구성
 - 식 적용

원하는 값(t)과 예측값(y)의 차이를 손실 함수로 정의

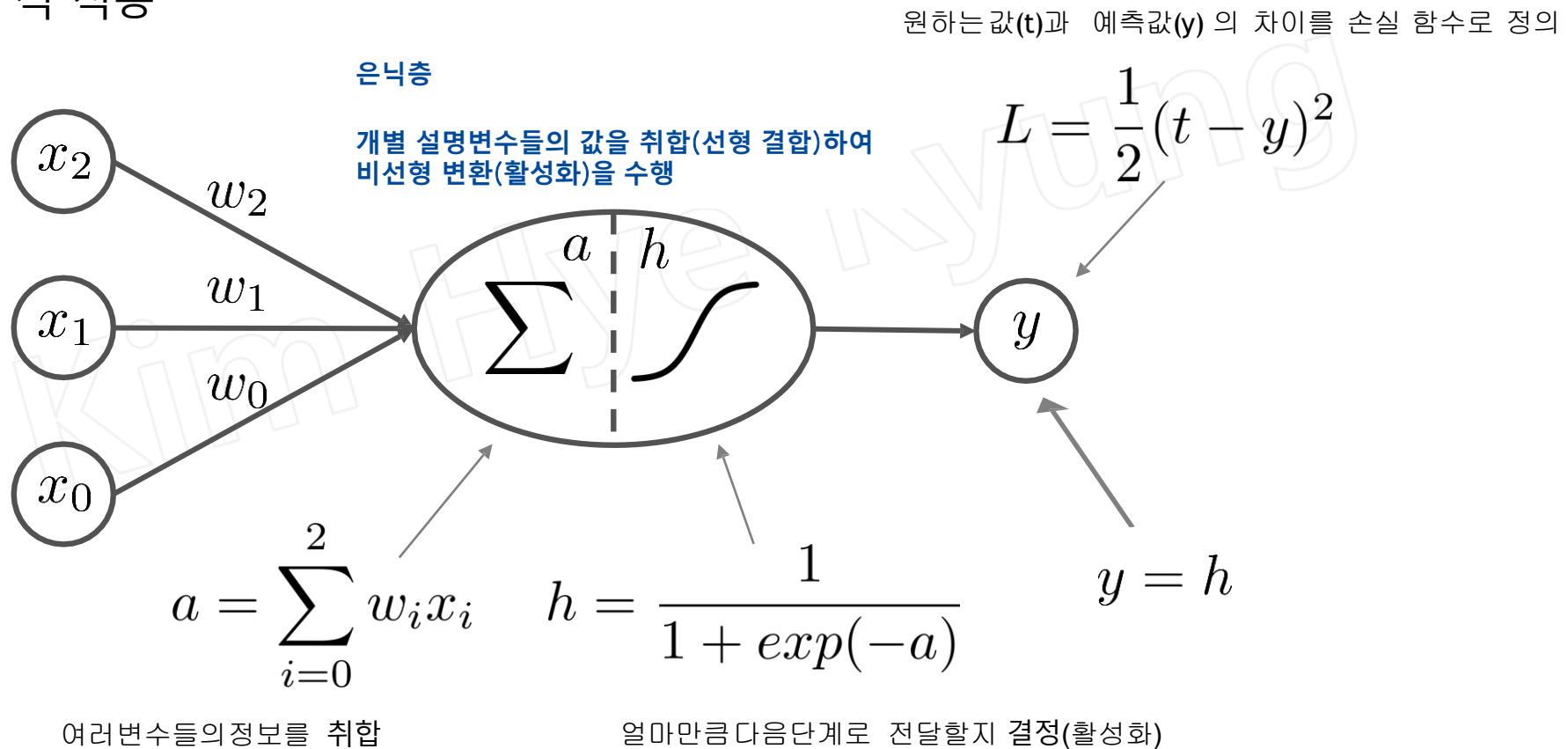


여러 변수들의 정보를 취합

얼마 만큼 다음 단계로 전달 할 지 결정(활성화)

퍼셉트론(단층 신경망) 이해하기

- 가장 간단한 퍼셉트론(뉴런 형식) 구조
 - 1개의 뉴런으로 구성
 - 식 적용



퍼셉트론 : 활성화 함수

- 은닉 노드의 활성화함수

입력을 활성화해서 다양한 출력을 구성

입력의 총합을 어떻게 활성화 해서 출력하는 지를 결정하는 함수

각 노드가 이전 노드들로부터 전달 받은 정보를 다음 노드에 얼마만큼 전달해 줄 것인가를 결정

가중치 값을 학습할 때 에러가 적게 나도록 도와주는 함수

퍼셉트론 : 활성화 함수

- 주요 활성화 함수 종류
 - 계단 함수
 - 결과값이 0 또는 1
 - 부호 함수
 - 결과가 -1 또는 1
 - 시그모이드 함수
 - 연속형 0~1
 - 소프트맥스 함수
 - 목표치가 다범주인 경우 각 범주에 속할 사후 확률 제공

퍼셉트론 : 활성화 함수

- 대표적 활성화 함수 종류

- Sigmoid

- 가장 일반적으로 사용되는 활성화 함수, $[0, 1]$ 의 범위를 가지며 학습 속도가 상대적으로 느림

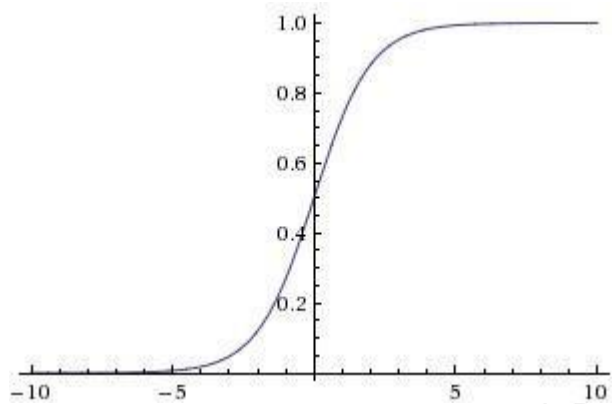
- Tanh

- Sigmoid 활성화 함수와 형태는 유사하나 $[-1, 1]$ 의 범위를 가져 학습 속도가 상대적으로 빠름

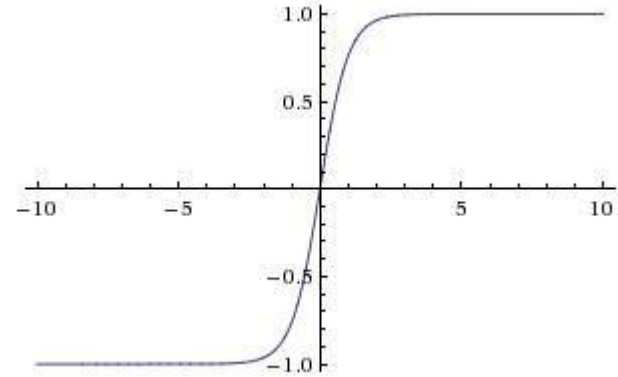
- ReLU(Rectified Linear Unit)

- 학습속도가 매우 빠르며 상대적으로 계산이 쉬움
 - (지수함수 형태를 사용하지 않으므로)

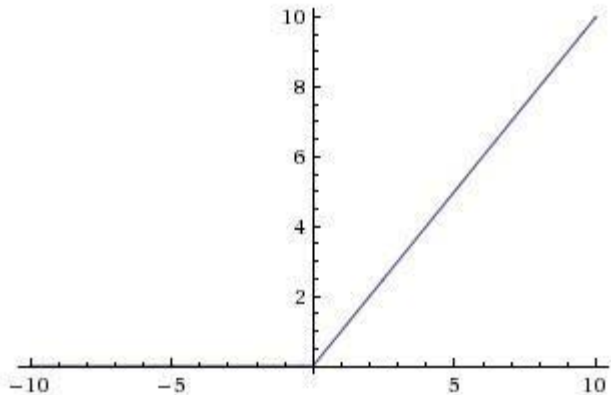
퍼셉트론 : 활성화 함수



Sigmoid $g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$



Tanh $g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$



ReLU $g(a) = \text{reclin}(a) = \max(0, a)$

퍼셉트론 : 목적 함수

- 퍼셉트론의 목적
 - 주어진 학습 데이터의 입력 정보(x)와 출력 정보(t)의 관계를 잘 찾도록 가중치 w를 조절하자!
- 관계가 잘 찾아졌는지는 어떻게 아는가?
 - 퍼셉트론의 결과물 y가 실제 정답 t에 얼마나 가까운지를 손실 함수(loss function)를 통해 측정 (손실 함수는 객체별로 산출)
 - 판정 결과가 정답인지, 오답인지 판정하고 그 오차를 평가하는 함수 의미
 - 회귀: 주로 squared loss 사용:
$$L = \frac{1}{2}(t - y)^2$$
 - 분류: 주로 cross-entropy loss 사용 (퍼셉트론은 이범주 분류만 가능):
$$L = \sum_{i=1}^2 t_i \log p_i$$
 - 전체 데이터 셋에 대해서 현재 퍼셉트론이 얼마나 잘못하고 있는지는 비용 함수(cost function)을 사용하며, 이는 모든 객체의 손실함수에 대한 평균값을 주로 사용함

퍼셉트론 : 목적 함수

- 비용(cost)/손실(loss) 함수

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

```
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

활성화 함수

- 입력을 활성화 해서 다양한 출력을 만듦
- 뉴런이 입력의 자극으로부터 임계값을 넘으면 발화하는 것처럼 어떤 조건을 만족하면 1, 만족하지 않으면 0을 출력하는 수식

Kim Hye Kyung

대표적인 신경망의 학습 규칙

헵의 규칙 (Hebbian Rule)

함께 발화한 뉴런 간의 시냅스 결합은 강해진다

$$\Delta w = \lambda xy$$

$$\Delta w = \begin{cases} \lambda x \\ 0 \end{cases}$$

가중치 w 는 입력 x 와 출력 y 가 동시에 발화한 빈도에 비례해 입력 x 의 λ 배만큼 증가하게 됨을 의미

델타 규칙 (Delta Rule)

정답과 출력의 차이가 클수록 가중치의 수정치(값)이 커진다
입력값이 클수록 가중치의 수정치(값)이 커진다

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i$$

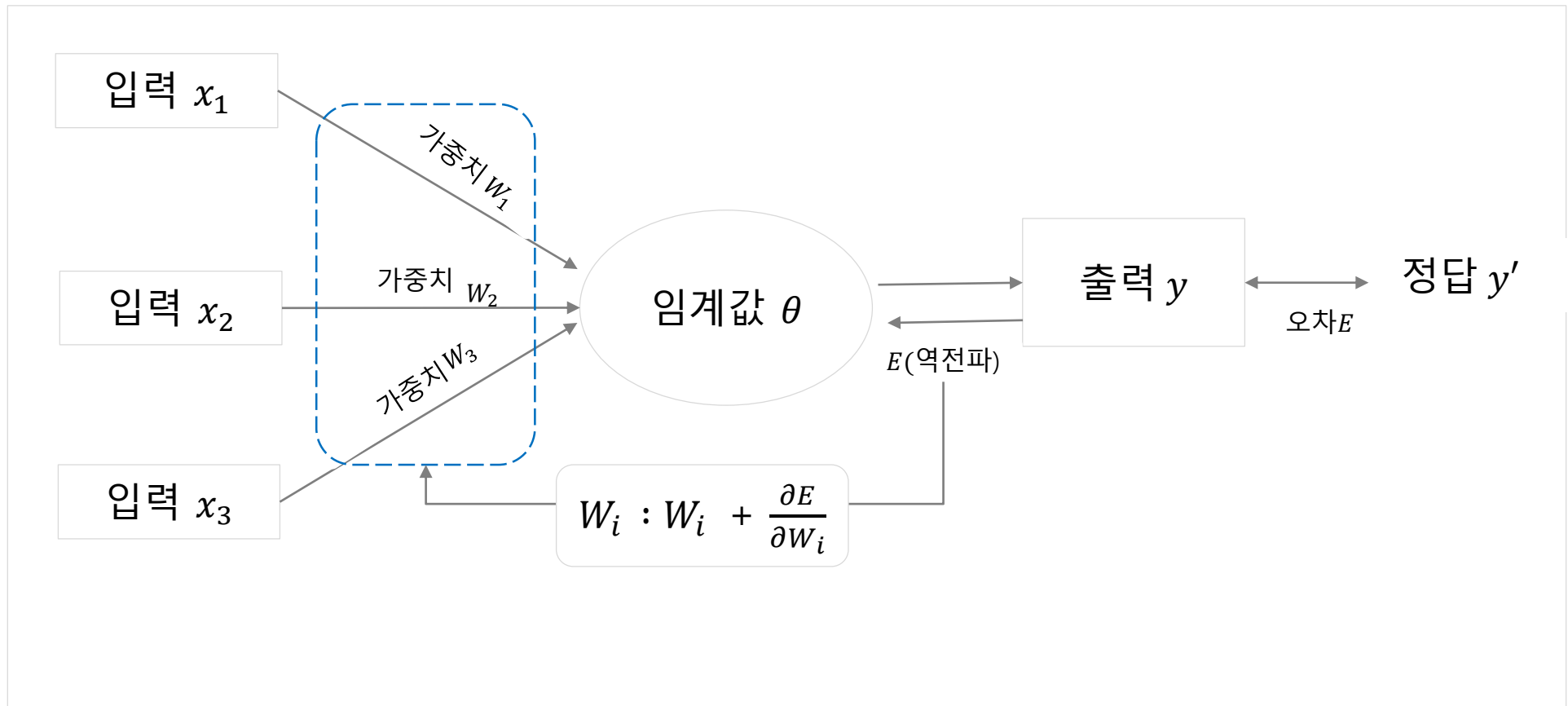
α : 학습속도라고 불리는 작은 상수,
 $g(x)$: 뉴런의 활성화 함수
 t_j : 원하는 목표 결과값
 y_j : 실제 결과값,
 x_i : i 번째 입력값

정답, 오답은 어떻게 판정?

- 학습 방법의 차이와 손실 함수
 - 지도/비지도/강화학습 차이
- 학습 프로세스로 보는 손실 함수
 - 지도 학습 process
 - 1단계 : 학습 데이터 준비
 - 2단계 : 데이터를 입력 데이터와 출력 데이터로 구분
 - 3단계 : 신경망에 데이터 입력
 - 4단계 : 신경망의 판정 결과와 출력 데이터 비교
 - 5단계 : 판정 결과와 출력 데이터의 차이를 피드백
 - 6단계 : 신경망의 파라미터를 갱신(3단계로 돌아감)
 - 판정 결과인 출력 데이터를 비교해서 오류(오차)가 거의 없거나 지정된 횟수나 시간이 지났을때 학습 종료

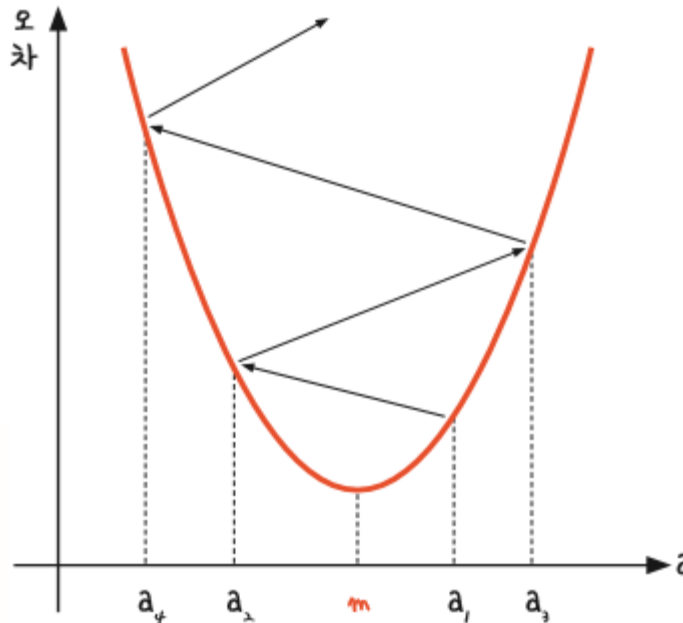
오차역전파법을 간략화한 수학적 모델

- 가장 간단한 퍼셉트론(뉴런 형식)



학습률[learning rate]

- 어느 만큼 이동시킬지를 정해주는 것 : 학습률(learning rate)
- 학습률을 너무 크게 잡으면 한 점으로 수렴하지 않고 발산
 - 기울기의 부호를 바꿔 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면 a 값이 한 점으로 모이지 않고 위로 치솟아 버림



| 선형 회귀로 본 딥러닝 필요 이론

선형 회귀

- 선형 회귀란?

- 독립 변수 x 를 사용해 종속 변수 y 의 움직임을 예측하고 설명하는 작업
- 선형 회귀는 곧 정확한 직선을 그려내는 과정
- 최적의 기울기 값과 절편 값을 찾아내는 작업

독립 변수

x 값이 변함에 따라 y 값도
변한다는 이 정의 안에서 독립적으로 변할 수 있는 x 를
의미

종속 변수

독립 변수에 따라 종속적으로
변하는 y 를 의미

훌륭한 예측이란?

- 딥러닝과 머신러닝의 '예측'이란?
 - 기존 데이터(정보)를 가지고 어떤 선이 그려질지를 예측한 뒤,
 - 아직 답이 나오지 않은 그 무언가를 그 선에 대입해 보는 것
 - 선형 회귀의 개념을 이해하는 것은 딥러닝을 이해하는 중요한 첫걸음

선형 회귀 종류

단순 선형 회귀
- Simple linear regression

하나의 x 값 만으로 y 값을 설명할 수 있는 경우

다중 선형 회귀
- Multiple linear regression

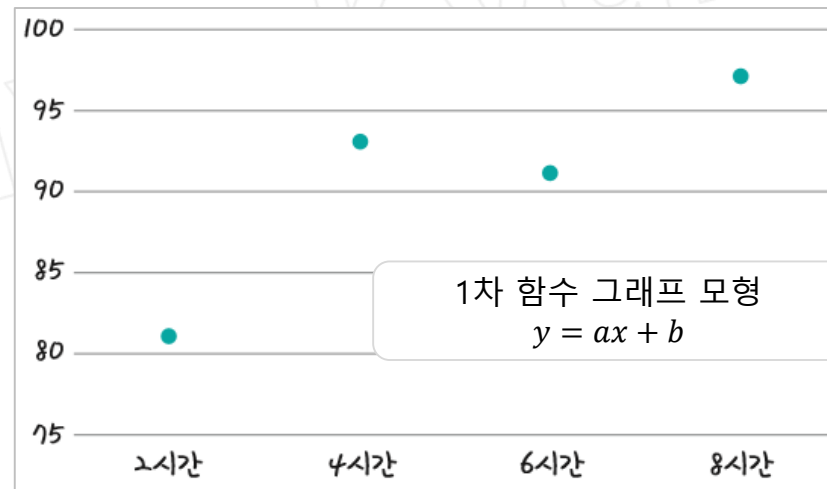
x 값이 여러 개 필요할 경우

학생들 성적 예측하기

- 독립 변수가 하나뿐인 단순 선형 회귀의 예

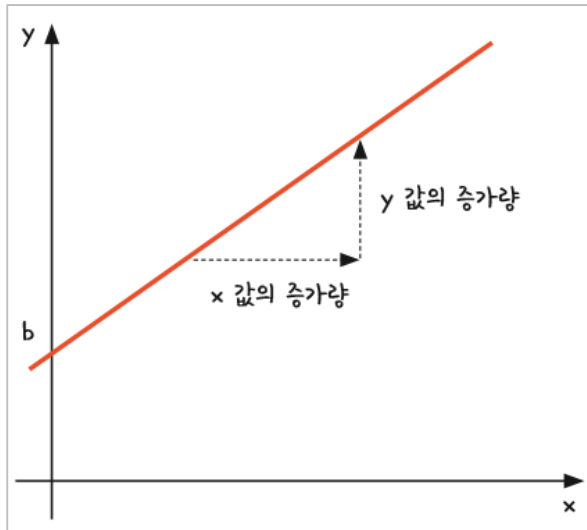
공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

- 공부한 시간
 - 집합 x
 - $x = \{2, 4, 6, 8\}$
- 성적
 - 집합 y
 - $y = \{81, 93, 91, 97\}$



공부한 시간과 성적을 좌표로 표현

학생들 성적 예측하기



1차 함수 그래프 모형
 $y = ax + b$

- x값 : 독립 변수, y값 : 종속 변수
 - x값에 따라 y 값이 달라짐
- a : 직선의 기울기, b : 절편
- 정확한 계산을 위한 a, b의 값 도출 방법
 - 최소 제곱법
 - 평균 제곱근 오차

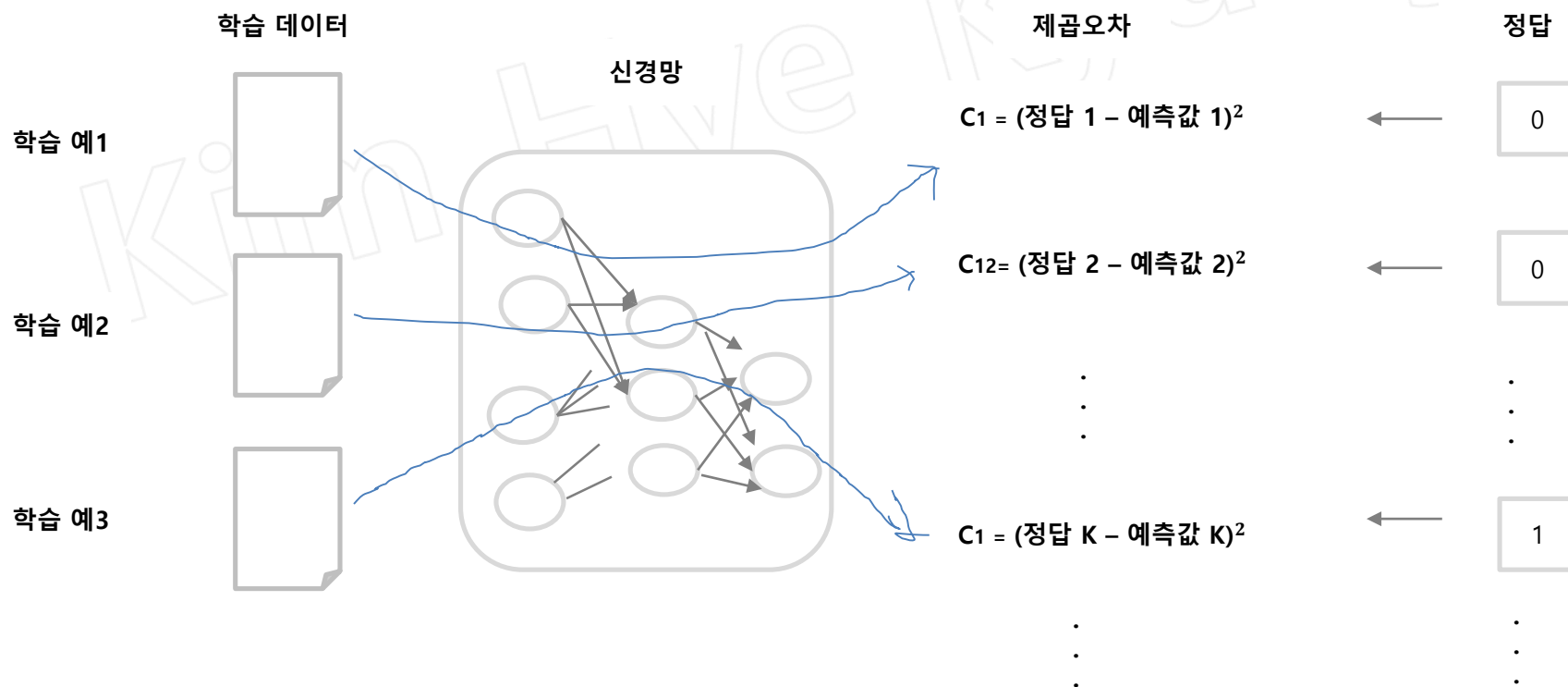
제곱 오차

- 제곱 오차란?

- 예측값과 정답 사이의 오차를 의미
- 학습 데이터 전체의 제곱 오차를 합한 것으로 정의
- 비용 함수라고도 함
- 제곱 오차를 이용하여 파라미터를 결정하는 방법을 수학에서 "최소제곱법" 이라고 함

최적화란?

제곱 오차의 합을 최소화하는
파라미터를 결정하는 방법



학생들 성적 예측하기 - 최소 제곱법(method of least squares)

- 회귀 분석에서 사용되는 표준 방식
- 정확한 기울기 a 와 정확한 y 절편의 값 b 를 알아내는 간단한 방법
- 실험이나 관찰을 통해 얻은 데이터를 분석하여 미지의 상수를 구할 때 사용되는 공식
- 보유한 정보가 x 값(입력 값, '공부한 시간')과 y 값(출력 값, '성적')일 때 최소 제곱법을 이용해 기울기 a 를 구하는 방법
 - 각 x 와 y 의 편차를 곱해서 이를 합한 값을 구함
 - x 편차 제곱의 합으로 나눔

$$a = \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균}) \text{의 합의 제곱}}$$

$$a = \frac{\sum_{i=1}^n (x - \text{mean}(x))(y - \text{mean}(y))}{\sum_{i=1}^n (x - \text{mean}(x))^2}$$

학생들 성적 예측하기 - 최소 제곱법(method of least squares)

- 성적(y)과 공부한 시간(x)으로 최소 제곱법을 이용해 기울기 a를 구할 경우?
 - x 값의 평균과 y 값의 평균을 각각 구함
 - 공부한 시간(x) 평균: $(2 + 4 + 6 + 8) \div 4 = 5$
 - 성적(y) 평균: $(81 + 93 + 91 + 97) \div 4 = 90.5$

$$\begin{aligned} a &= \frac{(2-5)(81-90.5) + (4-5)(93-90.5) + (6-5)(91-90.5) + (8-5)(97-90.5)}{(2-5)^2 + (4-5)^2 + (6-5)^2 + (8-5)^2} \\ &= \frac{46}{20} \\ &= 2.3 \quad \rightarrow \text{기울기는 2.3!} \end{aligned}$$

학생들 성적 예측하기 - 최소 제곱법(method of least squares)

- y 절편인 b를 구하는 공식

$$b = y\text{의 평균} - (x\text{의 평균} \times \text{기울기 } a)$$

- y의 평균에서 x의 평균과 기울기의 곱을 빼면 b의 값이 나옴

- 식
$$b = \text{mean}(y) - (\text{mean}(x) * a)$$

- y평균, x평균, 기울기 값을 적용한 식

$$\begin{aligned} b &= 90.5 - (2.3 \times 5) \\ &= 79 \quad \rightarrow \text{y 절편 } b \text{는 } 79 \end{aligned}$$

- 예측 값을 구하기 위한 직선의 일차 방정식 완성

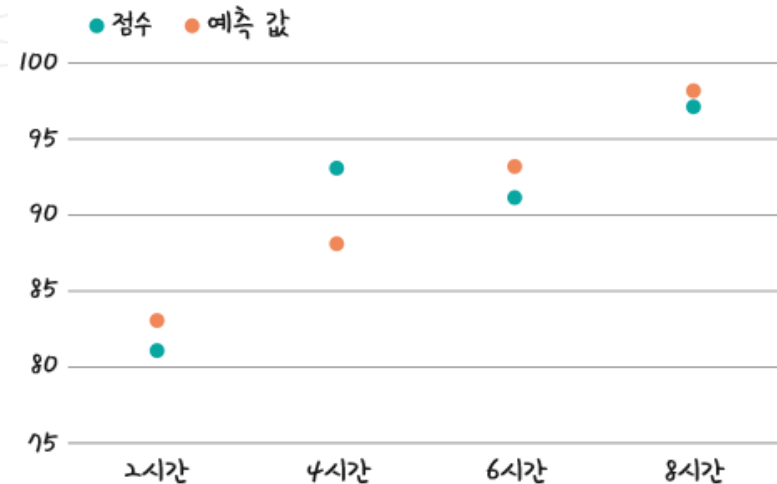
$$y = 2.3x + 79$$

학생들 성적 예측하기 - 최소 제곱법(method of least squares)

- 식에 x 를 대입했을 때 나오는 y 값을 ‘예측 값’

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	83.6	88.2	92.8	97.4

- 오차가 가장 적은 좌표 도출
- 따라서 x 값(공부한 시간)으로 공부량에 따른 성적 ‘예측’ 가능



공부한 시간과 성적, 예측 값을 좌표로 표현

평균 제곱근 오차(RMSE root mean square error)

- 최소 제곱법의 한계
 - '여러 개의 입력(x)'값이 있는 경우 이 공식만으로 처리 불가
 - 딥러닝은 대부분 입력 값이 여러 개인 상황
- 여러 개의 입력 값을 계산하는 방법
 - 가설을 하나 세운뒤 이 값이 주어진 요건을 충족하는지 판단하여 조금씩 변화를 주고, 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법
 - 임의의 선을 그리고 난 후
 - 이 선이 얼마나 잘 그려졌는지를 평가하여
 - 조금씩 수정해 가는 방법을 사용
- 이를 위해 주어진 선의 오차를 평가하는 오차 평가 알고리즘이 필요
 - 가장 많이 사용되는 방법: 평균 제곱근 오차(root mean square error)

평균 제곱근 오차(RMSE root mean square error)

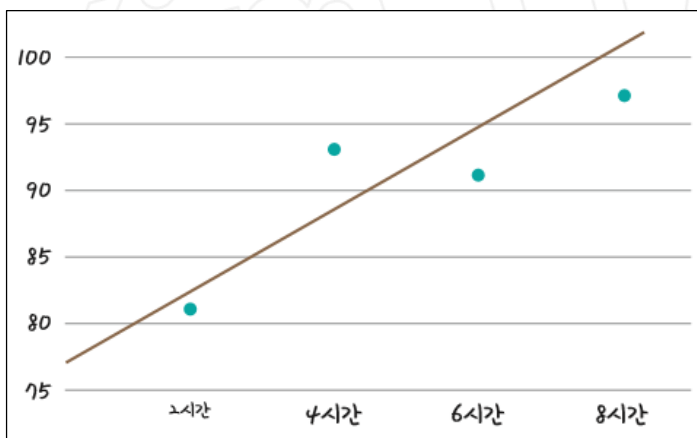
- 필요 단계
 - 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하기 위해 필요한 것은?
 - 각 선의 오차를 계산할 수 있어야 한다.
 - 이 오차가 작은 쪽으로 바꾸는 알고리즘이 필요하다

Kim Hye Kyung

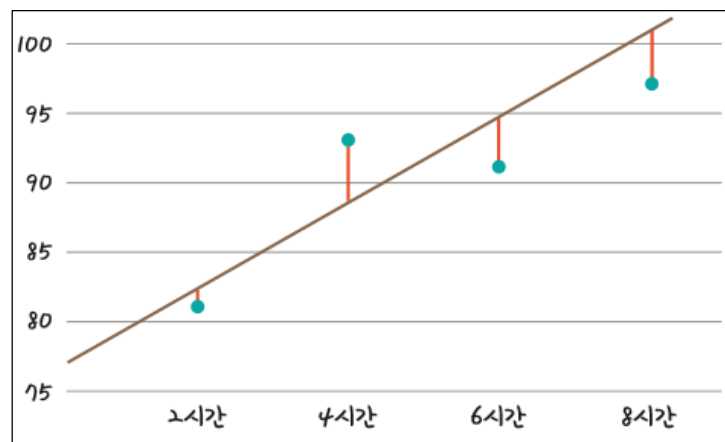
평균 제곱근 오차(RMSE root mean square error)

- 오차 계산 방법

- 선을 긋기 위해 기울기 a 와 절편 y 에 임의의 숫자 대입
- 임의의 직선이 어느 정도의 오차가 있는지를 확인하려면 각 점과 그래프 사이의 거리를 재면 됨
- 이 거리들의 합이 작을수록 잘 그어진 직선이고, 이 직선들의 합이 클수록 잘못 그어진 직선이 됨



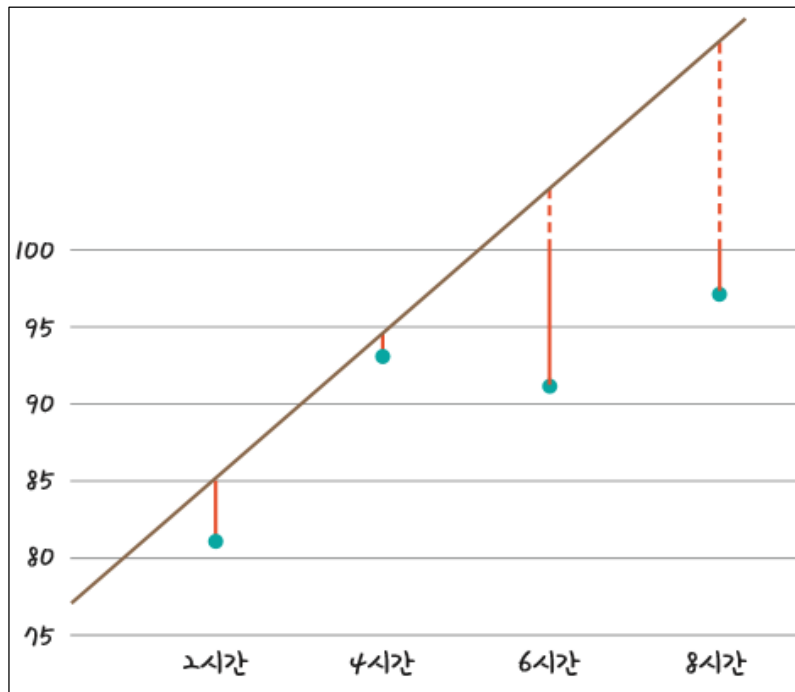
임의의 직선 그려보기



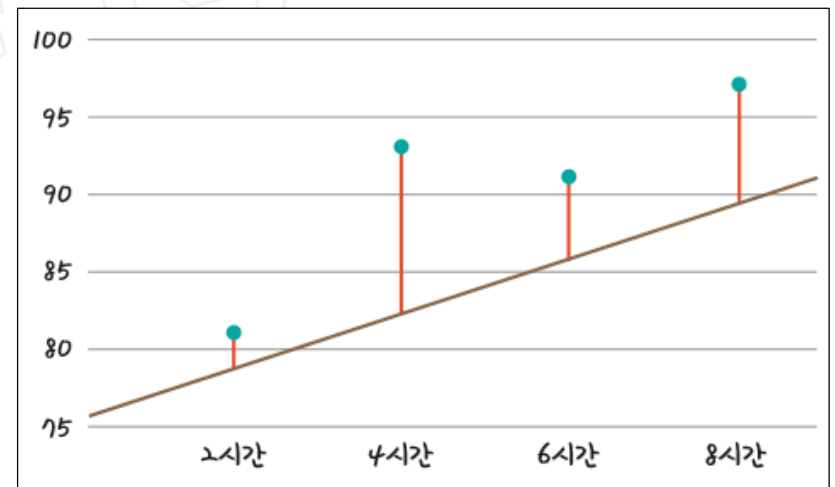
임의의 직선과 실제 값 사이의 거리

평균 제곱근 오차(RMSE root mean square error)

- 그래프의 기울기가 잘못 되었을 수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
- 만약 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있음



기울기를 너무 크게 잡았을 때의 오차



기울기를 너무 작게 잡았을 때의 오차

평균 제곱근 오차(RMSE root mean square error)

- 오차 구하는 방정식 오차 = 실제 값 - 예측 값
- $y = 3x + 76$ 인 경우의 오차

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	82	88	94	100
오차	1	-5	3	3

- 부호를 없애야 정확한 오차를 구할 수 있음
(양수, 음수가 섞여 있기 때문에 합이 0일수도 있음)
오차의 합을 구할 때는 각 오차의 값을 제곱해 줌
i : x가 나오는 순서
n : x 원소의 총 개수를 의미
 p_i : x_i 에 대응하는 '실제 값'
 y_i : x_i 가 대입되었을 때 직선의 방정식
(여기서는 $y = 3x + 76$)이 만드는 '예측 값'

$$\text{오차의 합} = \sum_{i=1}^n (p_i - y_i)^2$$

평균 제곱근 오차(RMSE root mean square error)

- 오차의 합을 n 으로 나누면 오차 합의 평균을 구할 수 있음

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2$$

- 평균 제곱 오차가 너무 커서 사용하기 불편한 경우 발생, 각 오차를 제곱한 값을 사용하므로 대용량 데이터를 이용할 때는 계산 속도가 느릴 가능성 보유
 - 해결책 : 제곱근 씹어줌
 - 평균 제곱근 오차**(Root Mean Squared Error, RMSE)라고 함
 - 계산 결과가 가장 작은 선을 찾는 작업

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2}$$

- 평균 제곱 오차 또는 평균 제곱근 오차는 오차를 계산해서 앞선 추론이 잘 되었는지 평가하는 대표적인 공식

평균 제곱근 오차(RMSE root mean square error)

- 잘못 그은 선 바로잡기는 곧 '평균 제곱근 오차'의 계산 결과가 가장 작은 선을 찾는 작업
- 선형 회귀란?
 - 임의의 직선을 그어 이에 대한 평균 제곱근 오차를 구하고
 - 이 값을 가장 작게 만들어 주는 a 와 b 값을 찾아가는 작업!

| 경사 하강법

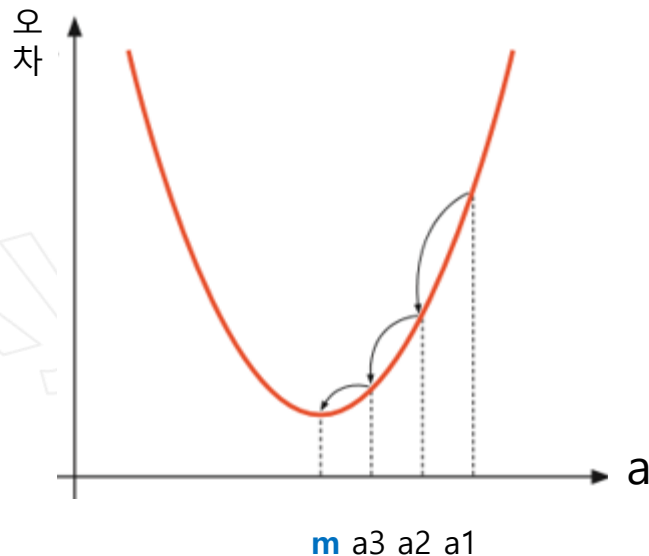
기울기 하강(Gradient Descent)

- 학습: 현재 퍼셉트론이 얼마나 틀렸는지를 알았으니 좀 더 잘 맞출 수 있도록 가중치 w 를 조절하는 것
- 기울기 하강: 눈을 가린 채로 산에서 가장 낮은 곳을 찾아가기

Kim Hye Kyung

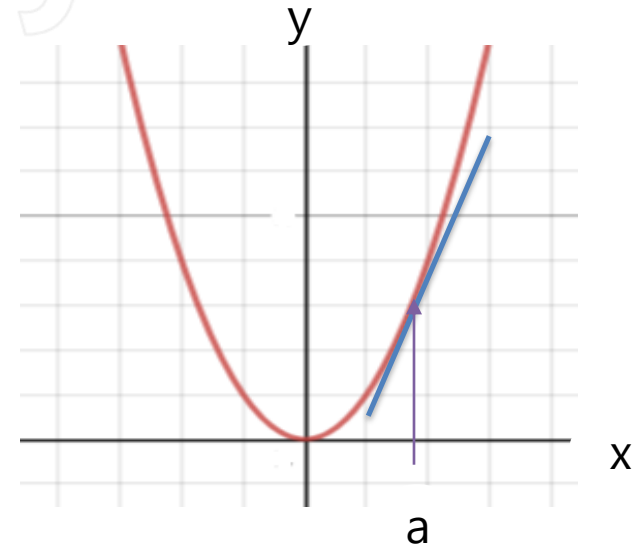
경사 하강법(Gradient Descent)

- 경사 하강법(gradient decent)이란?
 - 오차의 변화에 따라 이차 함수 그래프를 만들고 적절한 학습률을 설정해 미분값이 0인 지점을 구하는 것
- 기울기 a 와 오차와의 관계
 - 적절한 기울기를 찾았을 때 오차가 최소화



경사 하강법(Gradient Descent) : 미분의 개념

- 순간 변화율
 - ‘어느 쪽’이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선을 그릴 수가 있음
 - 변화량이 0에 가까울 만큼 아주 미세하게 변화했다면, y 값의 변화 역시 아주 미세해서 0에 가까울 것
 - 변화가 있긴 하지만, 그 움직임이 너무 미세하여 어느 쪽으로 ‘움직이려고 시도했다’는 정도의 느낌을 수학적으로 이름 붙인 것이 바로 ‘순간 변화율’
- 기울기
 - 순간 변화율이 ‘어느 쪽’이라는 방향성을 있고, 이 방향에 맞추어 그려지는 직선을 의미



경사 하강법(Gradient Descent) : 미분의 개념

- x 값이 아주 미세하게 움직일 때의 y 변화량을 구한 뒤, 이를 x의 변화량으로 나누는 과정
- **한 점에서의 순간 기울기**
- “함수 $f(x)$ 를 미분하라”의 표기 $\frac{d}{dx}f(x)$

$$\frac{d}{dx}f(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

① 함수 $f(x)$ 를 x 로 미분하라는 것은

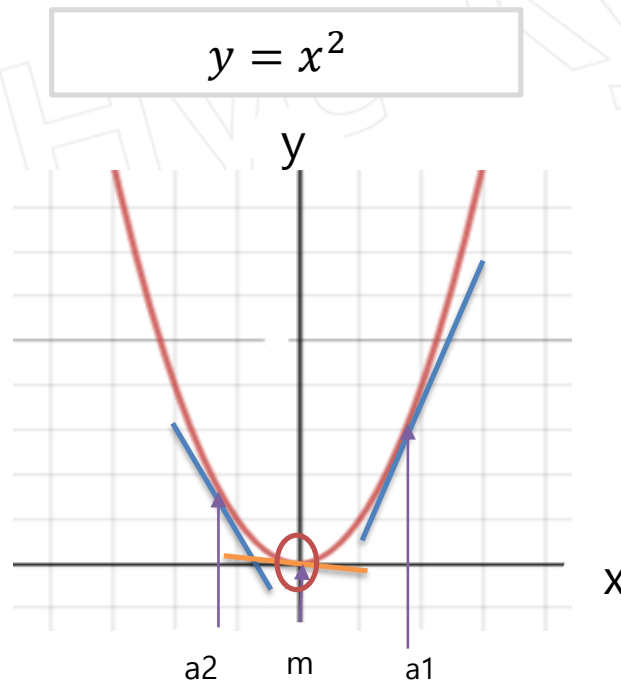
② x 의 변화량이 0에 가까울 만큼 작을 때

③ y 변화량의 차이를

④ x 변화량으로 나눈 값 (= 순간 변화율)을 구하라는 뜻

경사 하강법(Gradient Descent)

- 오차의 변화에 따라 이차 함수 그래프를 만들고 적절한 학습률을 설정해 미분값이 0인 지점은 구하는 것
- 미분은 한 점에서의 순간 기울기
- 순간 기울기가 0인 점 - 찾아야 하는 최솟값

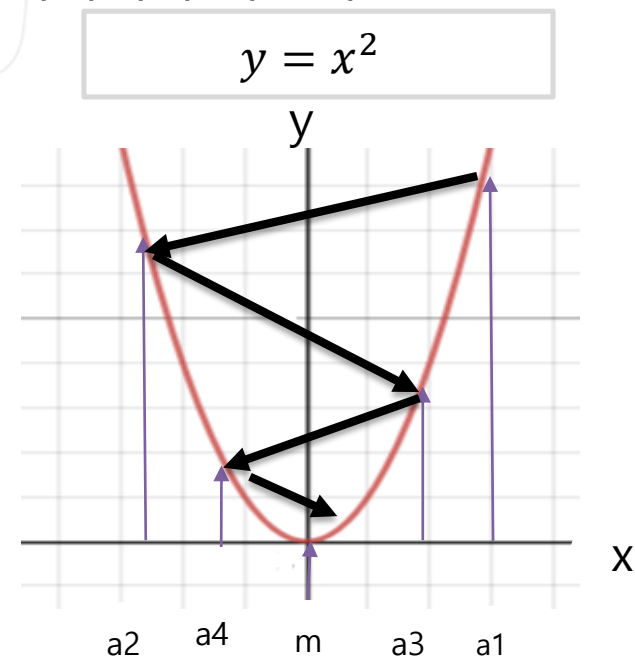


경사 하강법(Gradient Descent)

- 경사 하강법 구하기
 - 기울기 a 를 변화시켜서 m 의 값을 찾아내는 방법
- 기울기가 0인 최솟값 구하는 단계
 - 1단계 - a_1 에서 미분 구함
 - 2단계 - 구해진 기울기의 반대 방향으로 이동 시킨 위치에서 미분 구함
 - 3단계 - a_3 에서 미분 구하기
 - 4단계 - a_3 의 값이 0이 아니면 위 과정 반복
- 기울기가 0인 한 점(m)이 수렴

* 학습률이란? *

이동 거리를 정해주는 것이 학습률
어느 만큼 이동 시킬지를 신중히 결정해야 함



경사 하강법(Gradient Descent)

- 비용함수를 현재의 가중치 값(w)에 대해 1차 미분을 수행한 뒤 아래의 절차를 따름
 - 1차 미분 값(gradient)이 0인가?
 - 그렇다: 현재의 가중치 값이 최적! -> 학습 종료
 - 아니다: 현재의 가중치 값이 최적이지 않음 -> 좀 더 학습
 - 1차 미분 값(gradient)이 0이 아닐 경우 어떻게 해야 좀 더 잘하는 퍼셉트론을 만들 수 있는가?
 - 1차 미분 값(gradient)의 부호에 대한 반대 방향으로 가중치를 이동
 - 반대 방향으로 얼마나 움직여야 하는가?
 - 그건 잘 모름...
 - 조금씩 적당히(???) 움직여보고 그 다음에 다시 gradient 를 구해보자
 - 하다 보면 되겠지...

| 로지스틱 회귀

딥러닝이란?

겉으로 드러나지 않는 '미니 판단 장치' 들을 이용해서 복잡한
연산을 해낸 끝에 **최적의 예측 값을 내놓는 작업**

로지스틱 회귀(logistic regression)

- 로지스틱 회귀(logistic regression)란?
 - 참과 거짓중에 하나를 내 놓는 과정
- 원리

참, 거짓 구분하는 판단 장치 생성

주어진 입력 값의 특징 추출

추출된 특징으로 모델 생성

새로운 데이터로 모델 활용

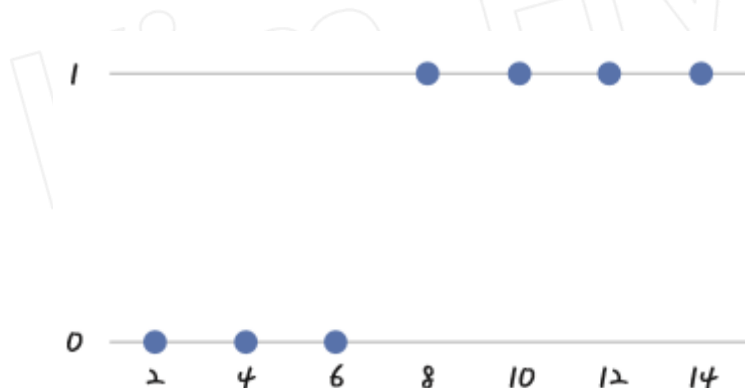
로지스틱 회귀

- 예시

- 점수가 아닌 오직 합격과 불합격으로만 발표되는 시험이 있다 가정

공부한 시간	2	4	6	8	10	12	14
합격 여부	불합격	불합격	불합격	합격	합격	합격	합격

- 합격 = 1 / 불합격 = 0 이라 가정

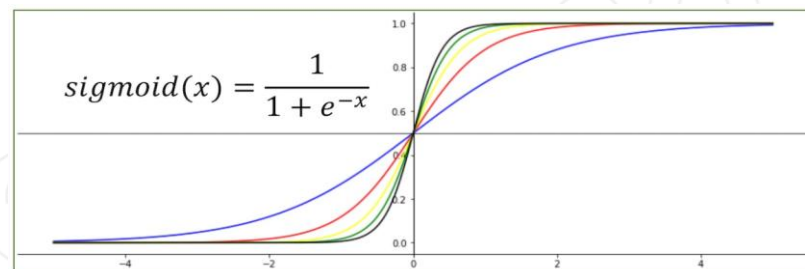
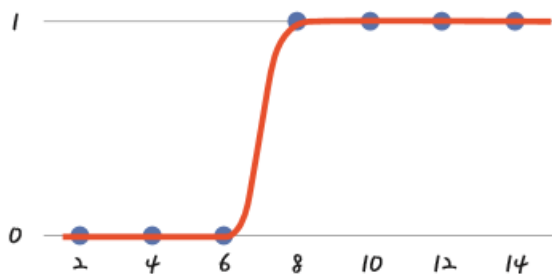


선을 그어 이 점의 특성을 잘 나타내는 일차 방정식을 만들 수 있을까?

이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기가 어려움

로지스틱 회귀

- 로지스틱 회귀는 선형 회귀와 마찬가지로 적절한 선을 그려가는 과정
- 다만, 직선이 아니라, 참(1)과 거짓(0) 사이를 구분하는 S자 형태의 선을 그려 주는 작업



- S자 형태의 그래프를 그리는 함수
 - 시그모이드 함수(Sigmoid Function)

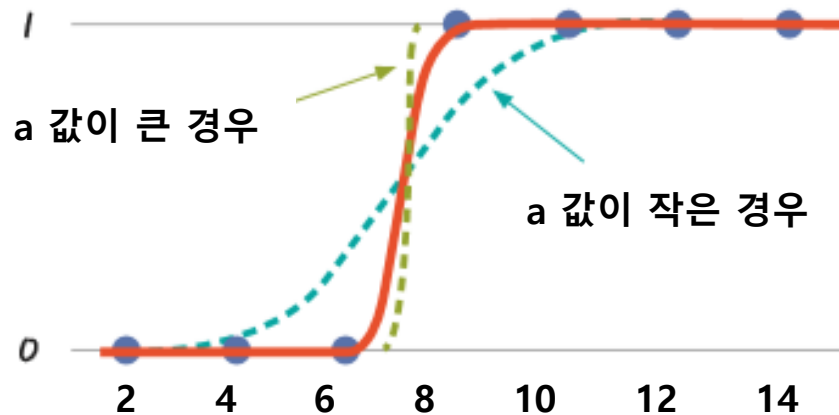
$$y = \frac{1}{1 + e^{(ax+b)}}$$

e는 자연 상수인 무리수로 값은 2.71828...
파이(π)처럼 수학에서 중요하게 사용되는 상수로 고정된 값이므로 우리가 따로 구해야 하는 값은 아님

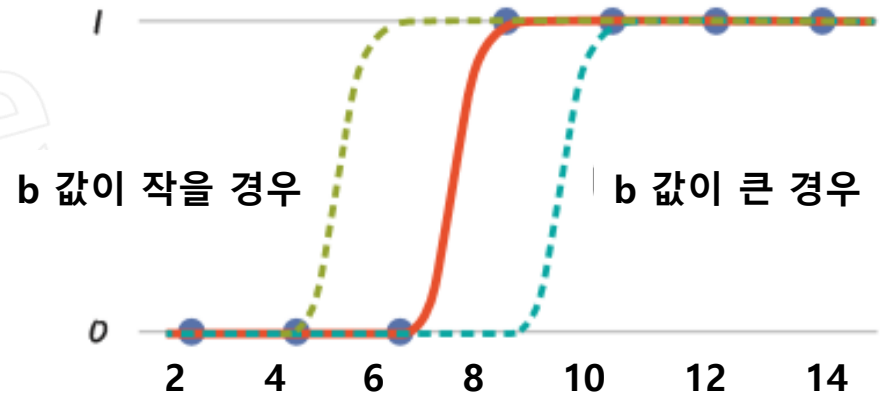
우리가 구해야 하는 값은 결국 $ax + b$

로지스틱 회귀

- 시그모이드 함수
 - a : 그래프의 경사도
 - B : 그래프의 좌우 이동



a 값이 커지면 경사가 커짐
 a 값이 작아지면 경사가 작아짐

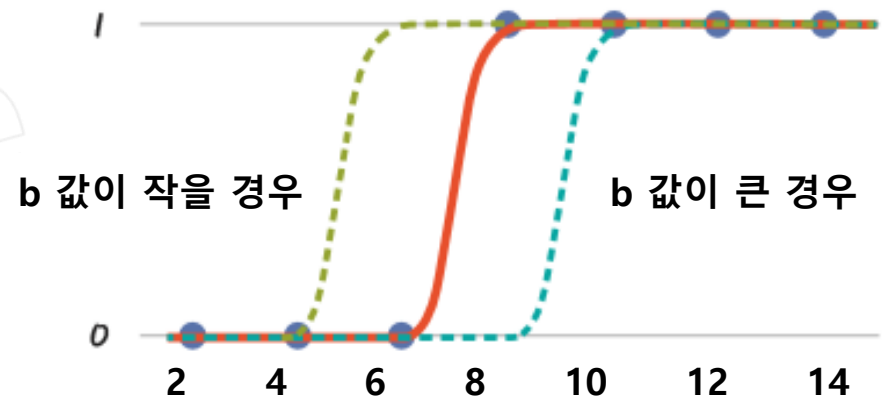
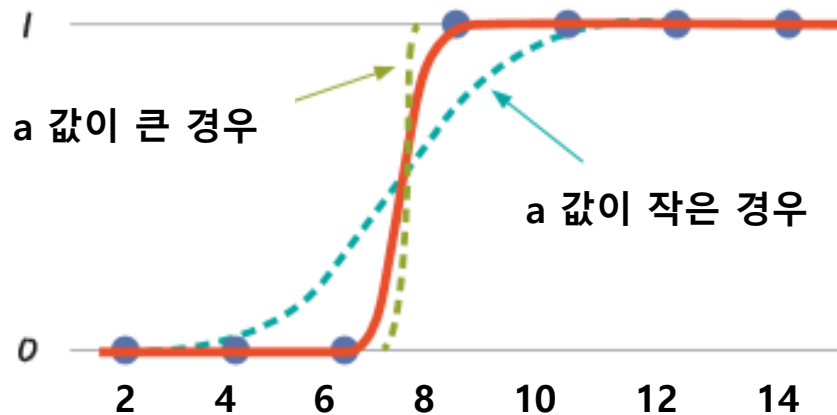


b 값이 크고 작아짐에 따라 그래프가 이동함

로지스틱 회귀

- 시그모이드 함수
 - a : 그래프의 경사도
 - b : 그래프의 좌우 이동

a 와 b 의 값이 클수록 오차가 생김



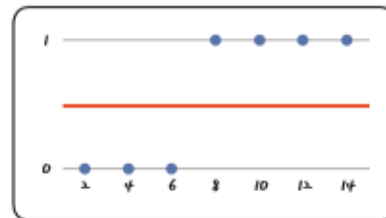
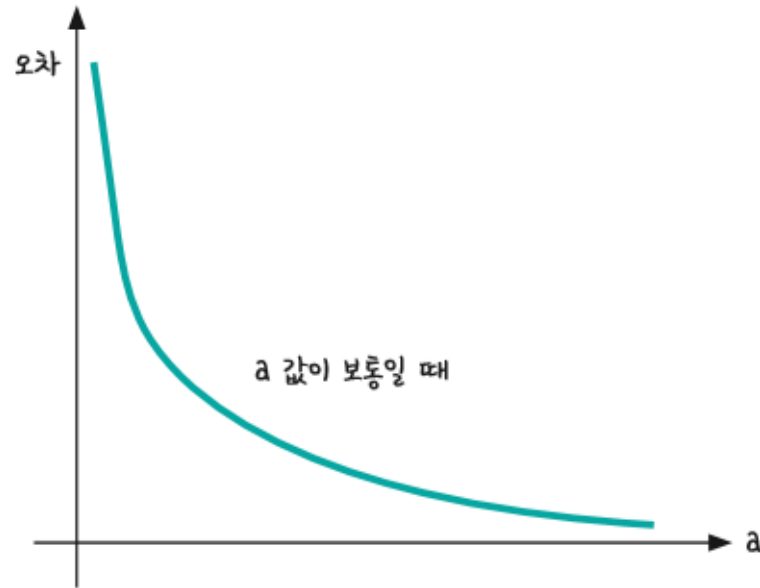
a 값이 커지면 경사가 커짐
 a 값이 작아지면 경사가 작아짐

b 값이 크고 작아짐에 따라 그래프가 이동함

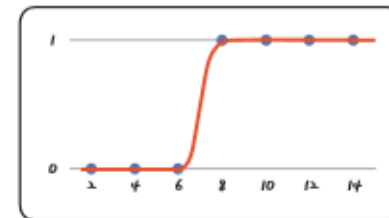
로지스틱 회귀

- 시그모이드 함수

- a 값이 크고 작아짐에 따라 오차는 다음과 같이 변함
- a 값이 작아지면 오차는 무한대로 커지지만, a 값이 커진다고 해서 오차가 무한대로 작아지지는 않음



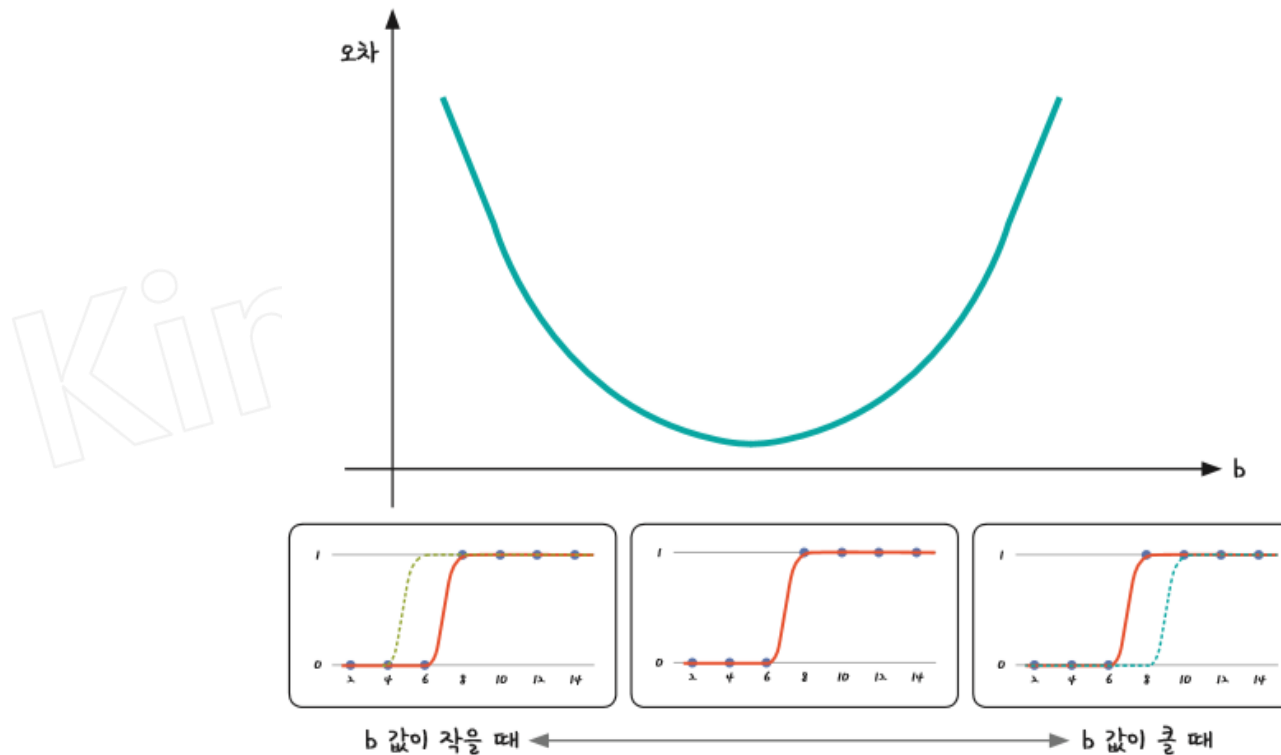
a 값이 작을 때



a 값이 클 때

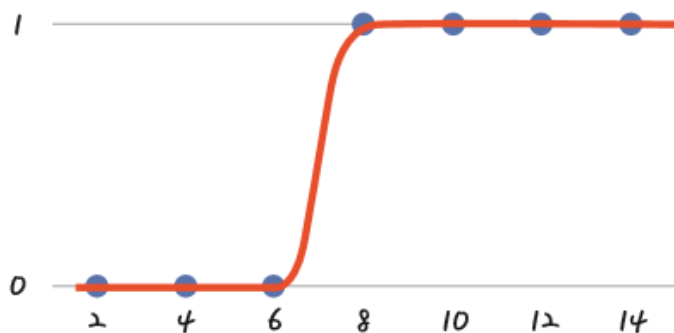
로지스틱 회귀

- 시그모이드 함수
 - b 값은 너무 크거나 작을 경우 오차가 무한대로 커짐
 - 이차 함수 그래프로 표현할 수 있음



로지스틱 회귀

- 시그모이드 함수에서 a 와 b 는 어떻게 구할 수 있나?
 - 해결책 : 경사 하강법
 - 오차를 구한 다음 오차가 작은 쪽으로 이동 시키는 방법
 - 예측 값과 실제 값의 차이, 즉 오차를 구하는 공식이 필요함

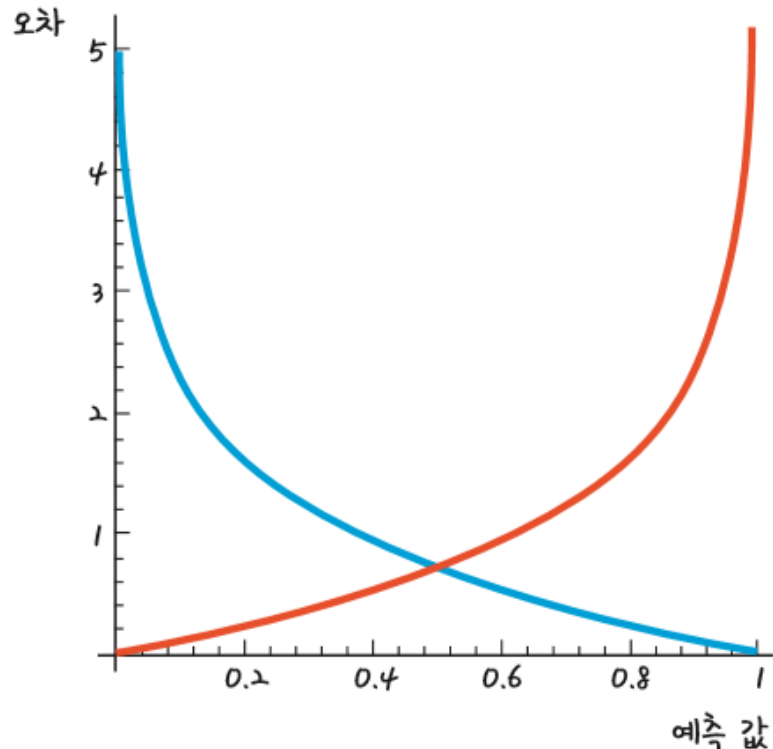


- 시그모이드 함수의 특징은 y 값이 0과 1 사이라는 것
 - 실제 값이 1일 때 예측 값이 0에 가까워지면 오차가 커져야 함
 - 반대로, 실제 값이 0일 때 예측 값이 1에 가까워지는 경우에도 오차는 커져야 함
 - 이를 공식으로 만들 수 있게 해 주는 함수: 로그 함수

로지스틱 회귀

- 로그 함수

- 실제 값이 1일 때(파란색)와 0일 때(빨간색) 로그 함수 그래프
- $y=0.5$ 에 대칭되는 두개의 함수



파란색 선

- 예측값이 1일때 오차는 0
- 예측값이 0에 가까울수록 오차는 커짐

빨간색 선

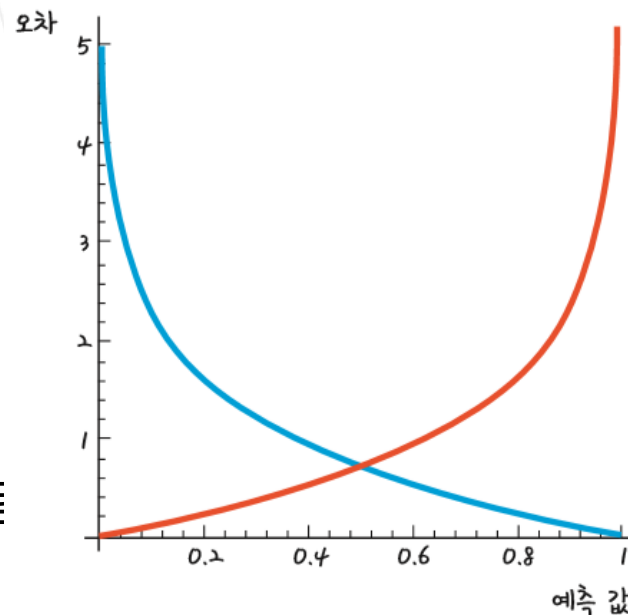
- 실제값이 0일때 사용할 수 있는 함수
- 예측값이 0일때 오차 없음
- 예측값이 1에 가까울수록 오차가 매우 커짐

로지스틱 회귀

- 로그 함수
- 파란색 그래프의 식 : $-\log h$
- 빨간색 그래프의 식 : $-\log(1 - h)$
 - y 의 실제 값이 1일 때 $-\log h$ 그래프를 쓰고, 0일 때 $-\log(1 - h)$ 그래프를 써야 함

$$-\underbrace{\{y \log h\}}_A + \underbrace{(1 - y) \log(1 - h)}_B$$

- 실제 값 y 가 1이면 B 부분이 없어짐
- 반대로 0이면 A 부분이 없어짐
- 따라서 y 값에 따라 빨간색 그래프와 파란색 그래프를 각각 사용할 수 있게 됨



로지스틱 회귀

- 시그모이드 함수 방정식을 넘파이 라이브러리를 이용해 다음과 같이 작성

$$y = \frac{1}{1 + e^{(ax+b)}}$$

```
y = 1/(1 + np.e**(a * x_data + b))
```

- 오차를 구하는 함수 역시 넘파이와 텐서플로를 이용해 다음과 같이 작성할 수 있음

$$-(y \cdot \log h + (1 - y) \log (1 - h))$$

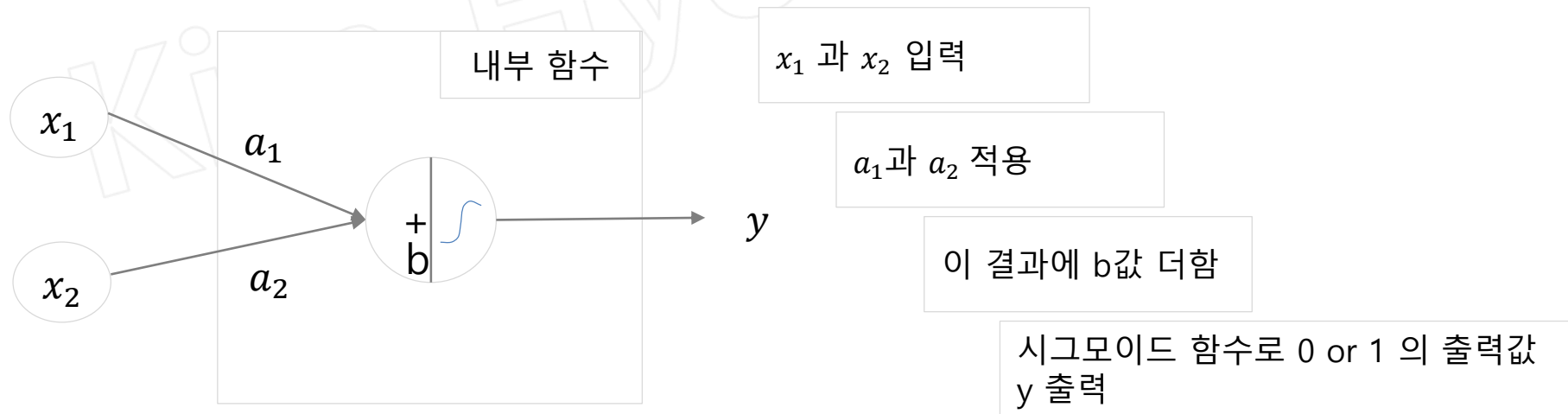
```
loss = -tf.reduce_mean(np.array(y_data) * tf.log(y) + (1 -  
np.array(y_data)) * tf.log(1 - y))
```

- 오차를 구하고 그 평균값을 loss 변수에 할당
- 평균을 구하기 위해 reduce_mean() 함수를 사용

로지스틱 회귀에서의 퍼셉트론

$$y = a_1x_1 + a_2x_2 + b$$

- 입력값(x_1, x_2)을 통해 출력값(y) 구함
- 출력값을 구하려면 a 와 b 필요
- 로지스틱 회귀를 퍼셉트론 방식으로 표현한 예

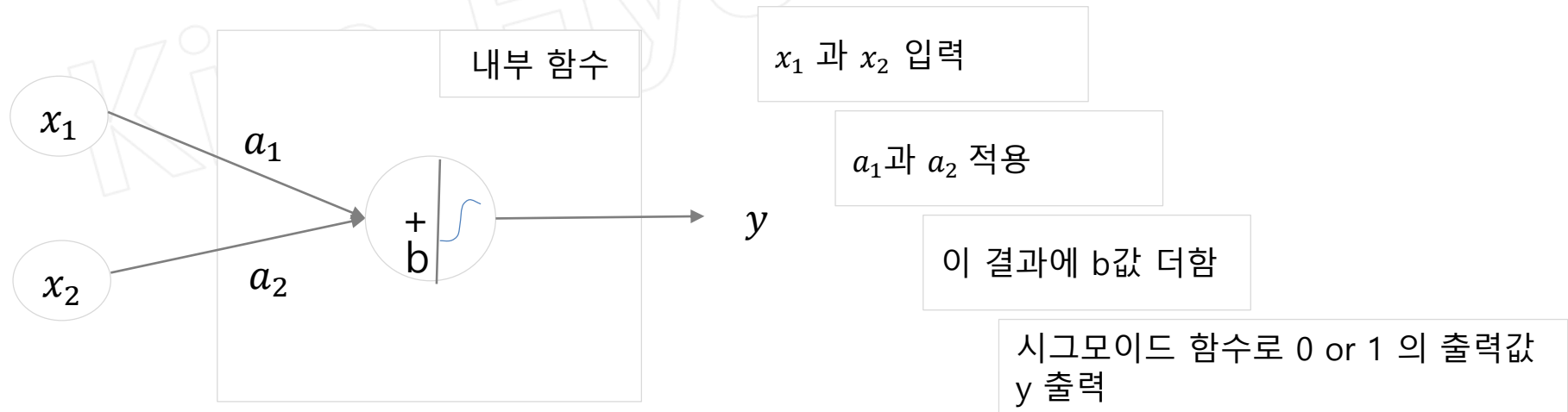


로지스틱 회귀에서의 퍼셉트론

$$y = a_1x_1 + a_2x_2 + b$$

- 입력값(x_1, x_2)을 통해 출력값(y) 구함
- 출력값을 구하려면 a 와 b 필요
- 로지스틱 회귀를 퍼셉트론 방식으로 표현한 예

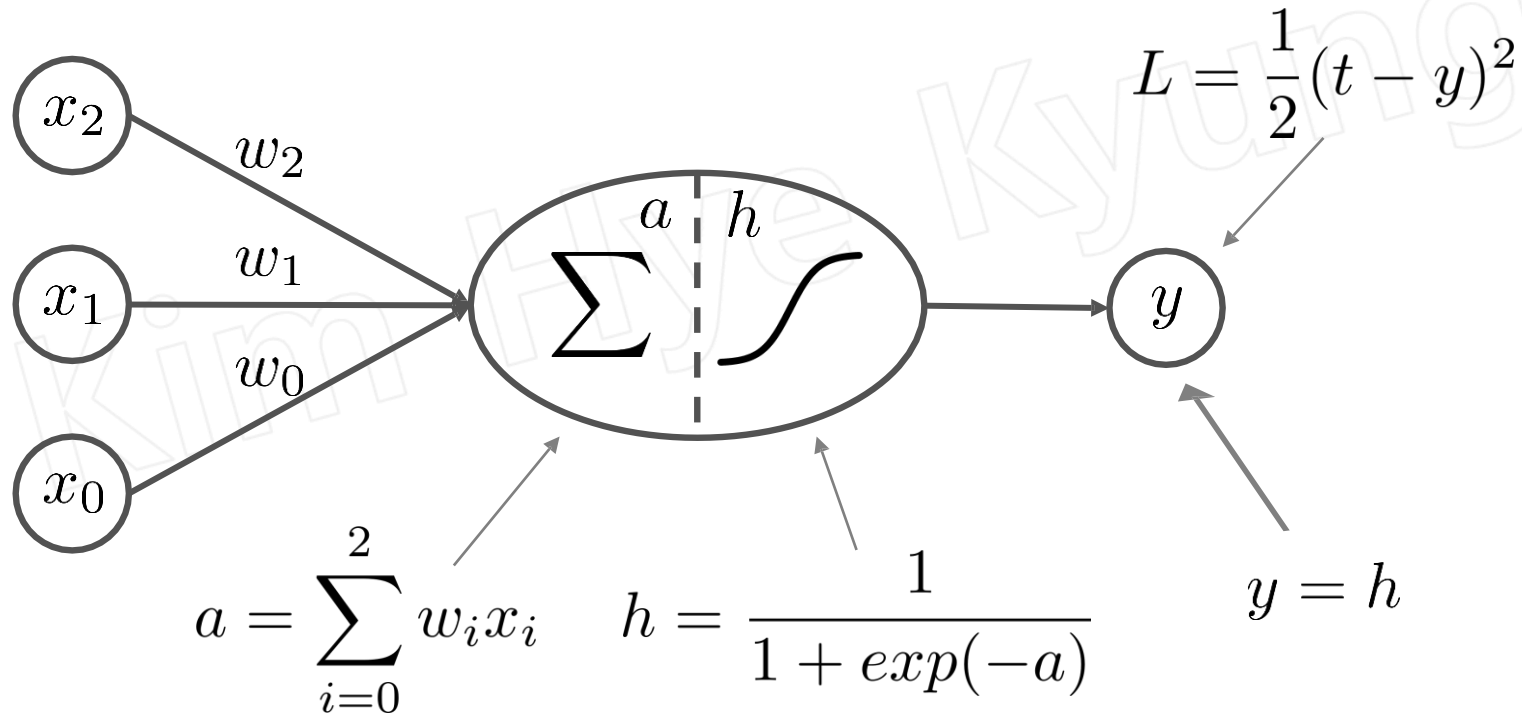
1957년도에 발표된
퍼셉트론과 동일한 개념



로지스틱 회귀에서의 퍼셉트론과 퍼셉트론(단층 신경망) 비교

- 가장 간단한 퍼셉트론(뉴런 형식) 구조
 - 1개의 뉴런으로 구성
 - 식 적용

원하는 값(t)과 예측값(y)의 차이를 손실 함수로 정의

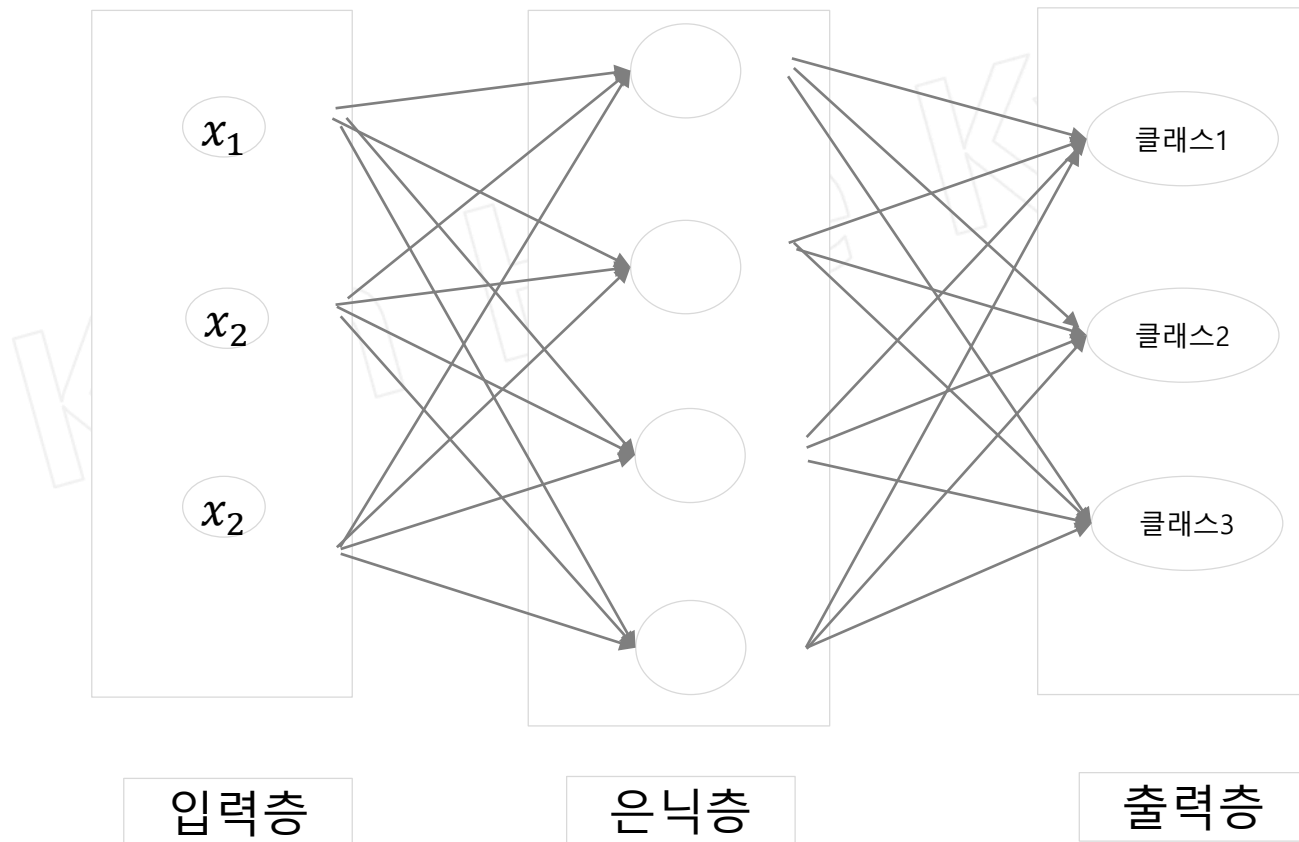


여러 변수들의 정보를 취합

얼마 만큼 다음 단계로 전달 할 지 결정(활성화)

N-클래스 식별

- N-클래스의 구조
 - 출력층이 하나 이상인 구조
 - 출력층의 유닛중에서 가장 큰 값을 출력한 유닛의 번호가 추론한 클래스



N-클래스 식별

- N-클래스
 - 신경망을 사용해서 N-클래스 문제를 풀 경우 one-hot-vector 형식 사용
 - one-hot-vector란?
 - 클래스 수 만큼의 길이를 가지는 배열
 - 배열은 내부에 요소 하나만 1, 나머지는 0

4-클래스			
0	1	2	3

어떤 레이블이 2인 경우

One-hot-vector

[0, 0, 1, 0]

N-클래스 식별

- One-hot-vector를 숫자 레이블로 변환하는 방법
 - `argmax()` 사용

하나만 변환하는 경우

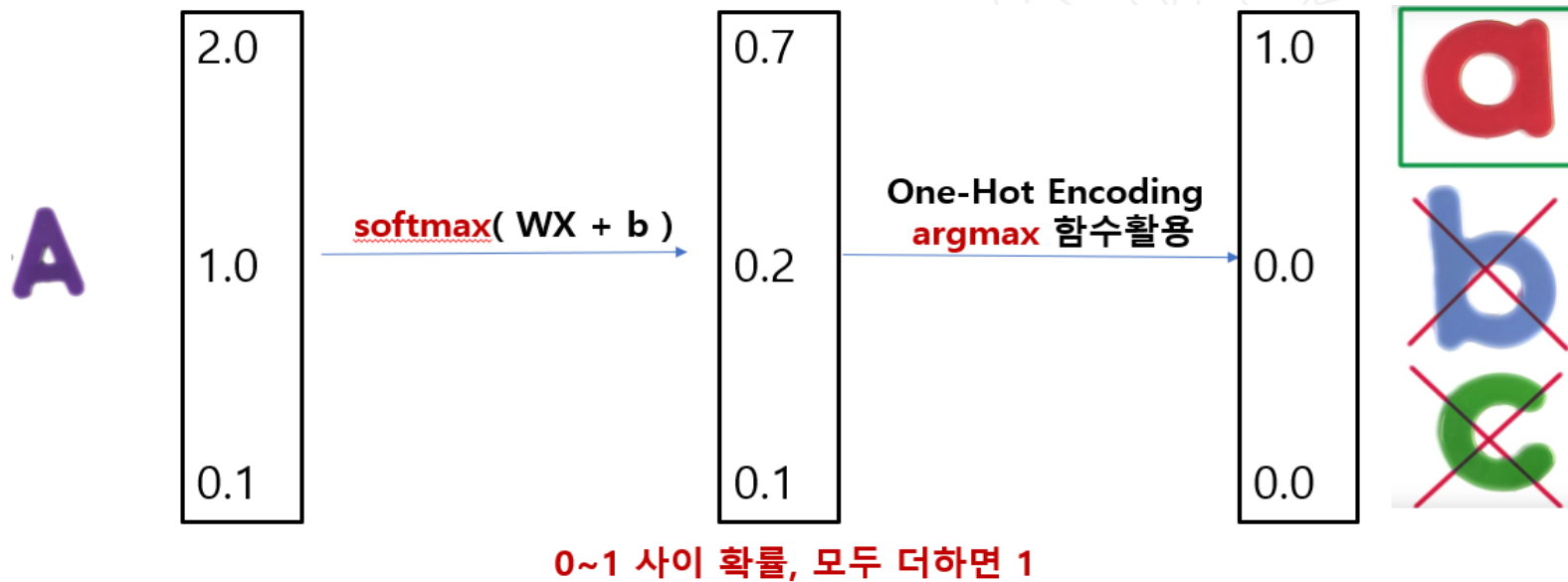
```
onehot = [0, 0, 1, 0]          # 4-클래스, 레이블은 2
label = np.argmax(onehot)      # label=2
```

여러개를 변환하는 경우

```
onehot = [ [1, 0, 0, 0],
            [0, 0, 1, 0] ]
label = np.argmax(onehot, axis=1) # label=[0, 2]
```

N-클래스 식별

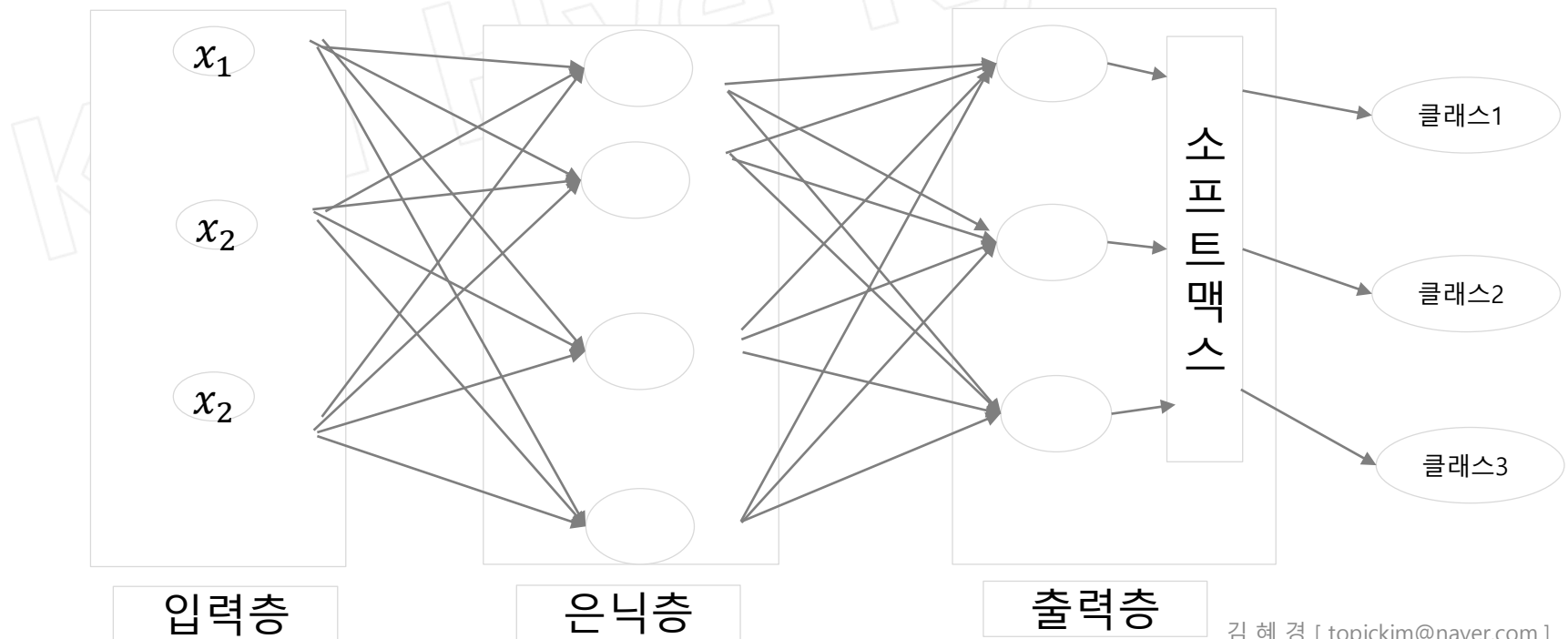
- 값이 아니라 A,B,C의 벡터 값으로 나옴
- softmax 함수를 이용하여 0~1사이의 값으로 p(확률)값으로 나오게 바꿀 수 있음
 - softmax 함수를 통과시킨 결과 값이 가장 큰 값이 결과



N-클래스 식별

- 소프트맥스 함수

- softmax 함수를 이용하여 0~1사이의 값으로 $p(\text{확률})$ 값으로 나오게 변경
- 출력층의 모든 출력(4개의 클래스라면 4개의 출력) 크기를 합해서 1인 되게 변환하는 활성화 함수
- 일반적인 활성화 함수는 하나의 유닛에 하나의 활성화 함수를 적용하나 소프트맥스 함수는 여러 개의 유닛을 모아 적용



N-클래스 식별

- 소프트맥스 함수
 - 4개의 유닛으로 구성되어 있는 출력층의 출력을 변환해 보기
 - **신경망의 출력층에 있으므로 추론 결과를 확률로 나타낼 수도 있음**

```
outputs = np.array([1, 2, 788, 789])
```

```
pred = softmax(outputs)
```

#출력 형식 맞추기

```
np.set_printoptions(formatter={'float' : '{:0.6f}'.format})
```

```
print(pred) # [0.000000, 0.000000, 0.268941, 0.731059]
```

각각 추론한 클래스의 값을 '예측 확률' 처럼 사용 가능

[0.000000, 0.000000, 0.268941, 0.731059]	정답 레이블
0%, 0%, 26,8941%, 73.1059%	추론 결과

N-클래스

- 소프트맥스 함수를 활성화 함수로 사용할 경우 선호하는 손실 함수
- 교차 엔트로피 오차
 - '어떤 그룹 A와 B가 어느 정도 다른가'를 나타냄
 - 출력 확률 중에서 정답을 얻을 확률을 뽑아낼 수 있기 때문에 그 확률을 대수화(log를 취함)한 것을 합하면 됨
 - 추론 결과에 log를 취하므로, 정답 확률이 낮아질수록 지수적으로 손실이 증가하는 특징이 있음
 - 따라서 손실이 클수록 학습이 더 많이 진행됨
 - 용어 정리
 - 교차
 - 개와 고양이의 예
 - 개와 고양이, 고양이와 개를 서로 비교
 - 엔트로피
 - 어떤 확률 P에서 발생할 사상의 정보량 의미
 - $-\log(P)$: P는 0~1의 범위 내에 있음

N-클래스

- **Cross-entropy** cost function :
1번씩 또는 2번씩 사용

```
logits = tf.matmul(X, W) + b  
hypothesis = tf.nn.softmax(logits)
```

1

```
# Cross entropy cost/loss
```

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

2







































```
# Cross entropy cost/loss
```

```
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,  
                                                  labels=Y_one_hot)
```

```
cost = tf.reduce_mean(cost_i)
```

N-클래스

- 동물의 16가지 특성 정보를 이용하여 동물 종 분류하기 예제
 - 다음과 같이 6 종류의 동물 종류가 있음

Birds	Insect	Fishes	Amphibians	Reptiles	Mammals
					 
				 	 
					 
					 
					 
	 			Kayla	    

data-04-zoo.csv 파일에 각 동물의 16가지 특성 정보가(feathers ,hair, milk 등) 있음

마지막 컬럼은 해당 동물이 7가지 종류(6가지와 기타 종) 중 몇 번에 속하는 동물 인지의 정보(0~5번 중 하나)

One Hot 인코딩이 필요하다

N-클래스

- 동물의 16가지 특성 정보를 이용하여 동물 종 분류 데이터

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

Deep Neural Network [DNN]

- [활성 함수: sigmoid 보다는 ReLu]
 - XOR 실습 예제의 hidden layer를 9개를 더 늘려보기

```
W1 = tf.Variable(tf.random_normal([2, 5], -1.0, 1.0), name='weight1')
W2 = tf.Variable(tf.random_normal([5, 5], -1.0, 1.0), name='weight2')
W3 = tf.Variable(tf.random_normal([5, 5], -1.0, 1.0), name='weight3')
W4 = tf.Variable(tf.random_normal([5, 5], -1.0, 1.0), name='weight4')
W5 = tf.Variable(tf.random_normal([5, 5], -1.0, 1.0), name='weight5')
W6 = tf.Variable(tf.random_normal([5, 1], -1.0, 1.0), name='weight6')
```

```
b1 = tf.Variable(tf.random_normal([5]), name='bias1')
b2 = tf.Variable(tf.random_normal([5]), name='bias2')
b3 = tf.Variable(tf.random_normal([5]), name='bias3')
b4 = tf.Variable(tf.random_normal([5]), name='bias4')
b5 = tf.Variable(tf.random_normal([5]), name='bias5')
b6 = tf.Variable(tf.random_normal([5]), name='bias6')
```

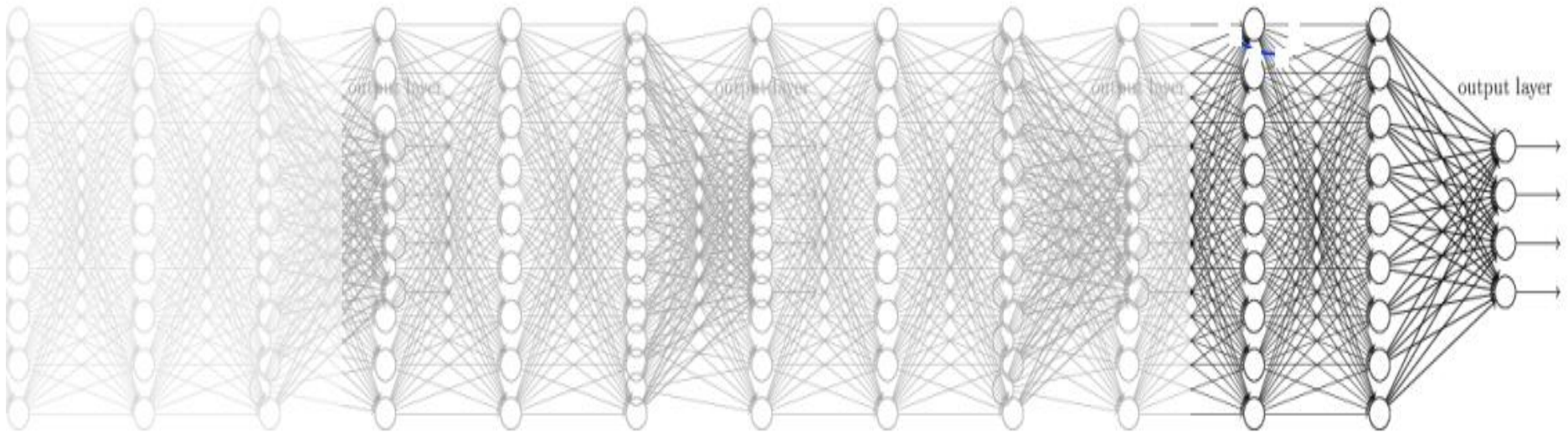
```
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
```

```
hypothesis = tf.sigmoid(tf.matmul(L5, W6) + b6)
```

```
196000 [0.69314718, array([[ 0.49999988],
[ 0.50000006],
[ 0.49999982],
[ 0.5          ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
[ 0.50000006],
[ 0.49999982],
[ 0.5          ]], dtype=float32)]
[array([[ 0.49999988],
[ 0.50000006],
[ 0.49999982],
[ 0.5          ]], dtype=float32), array([[ 0.],
[ 1.],
[ 0.],
[ 1.]], dtype=float32)]
Accuracy: 0.5
```

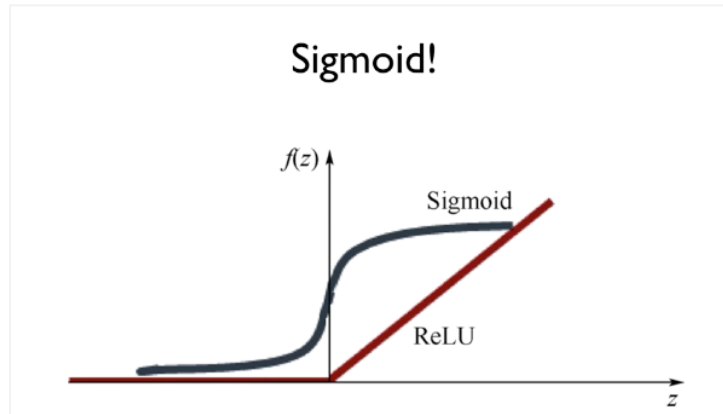
Deep Neural Network [DNN]

- 정확도가 많이 실망스럽습니다. 그 이유는?
 - Sigmoid 함수의 결과는 0~1 사이의 값이고, 그 값이 NN layer 지나가면서 사라지는 현상 때문 (Vanishing gradient)
 - (예) Sigmoid(Z) 결과가 0.01 이면 $0.01 * 0.01 * 0.02 \dots$)



Deep Neural Network [DNN]

- 딥러닝에서는 Sigmoid 함수보다 Relu 함수(0~무한대)를 대신 사용



- XOR 예제의 hidden layer 활성화 함수를 Relu 함수
- 마지막 단은 0~1사이의 값이어야 함
 - 이진 분류 문제이므로
#Sigmoid 사용

```
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L5 = tf.nn.relu(tf.matmul(L4, W5) + b5)
hypothesis = tf.sigmoid(tf.matmul(L5, W6) + b6)
```

Deep Neural Network [DNN]

- 어떤 활성화 함수를 써야 할까?

Hidden layer의 활성화함수

Relu

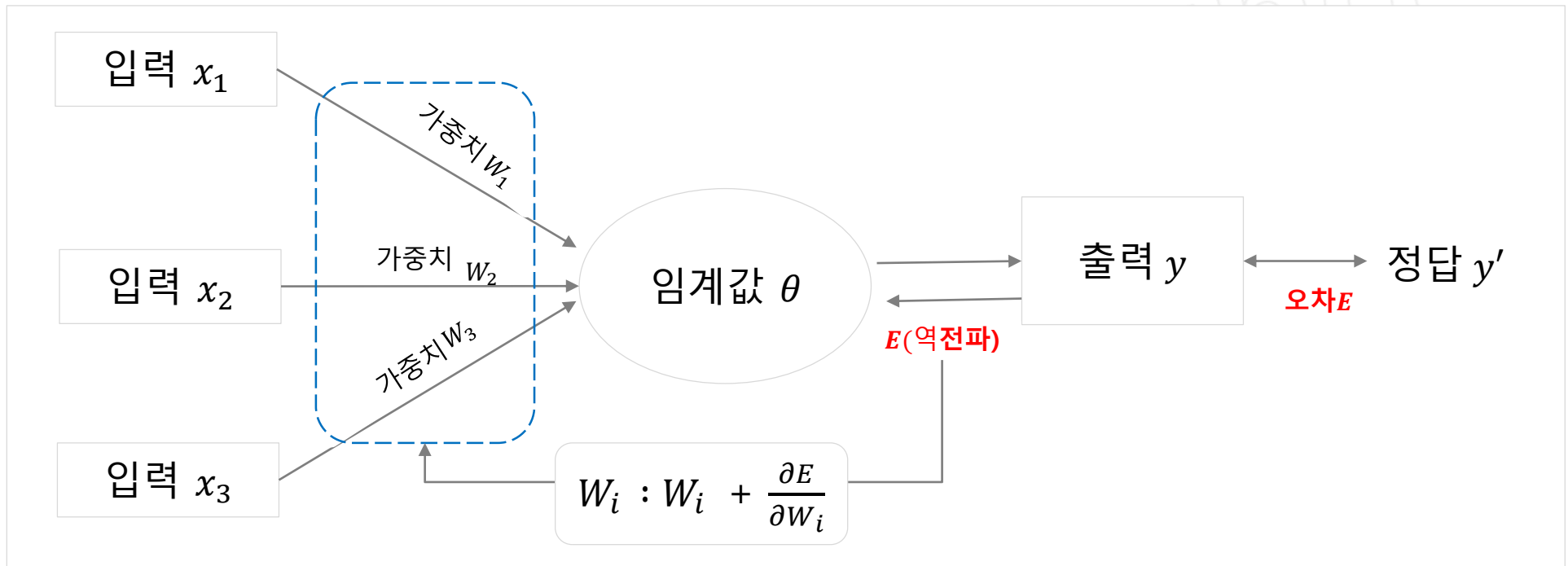
Output layer 의 활성화함수

이진 분류의 경우 Sigmoid, 다중 분류는 Softmax

| 오차 역전파[Back Propagation]

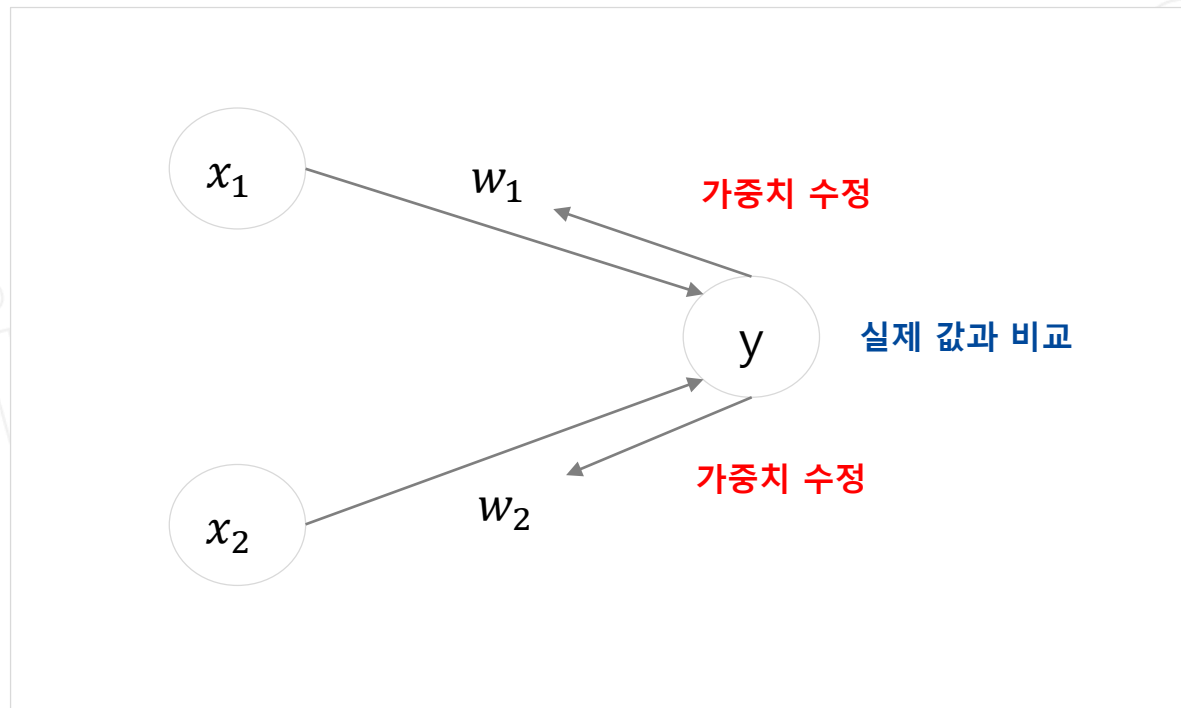
오차 역전파[Back Propagation]

- 출력층으로부터 하나씩 앞으로 되돌아 가며 각 층의 가중치를 수정하는 방법
 - 최적화의 계산 방향이 출력층에서 시작해 앞으로 진행



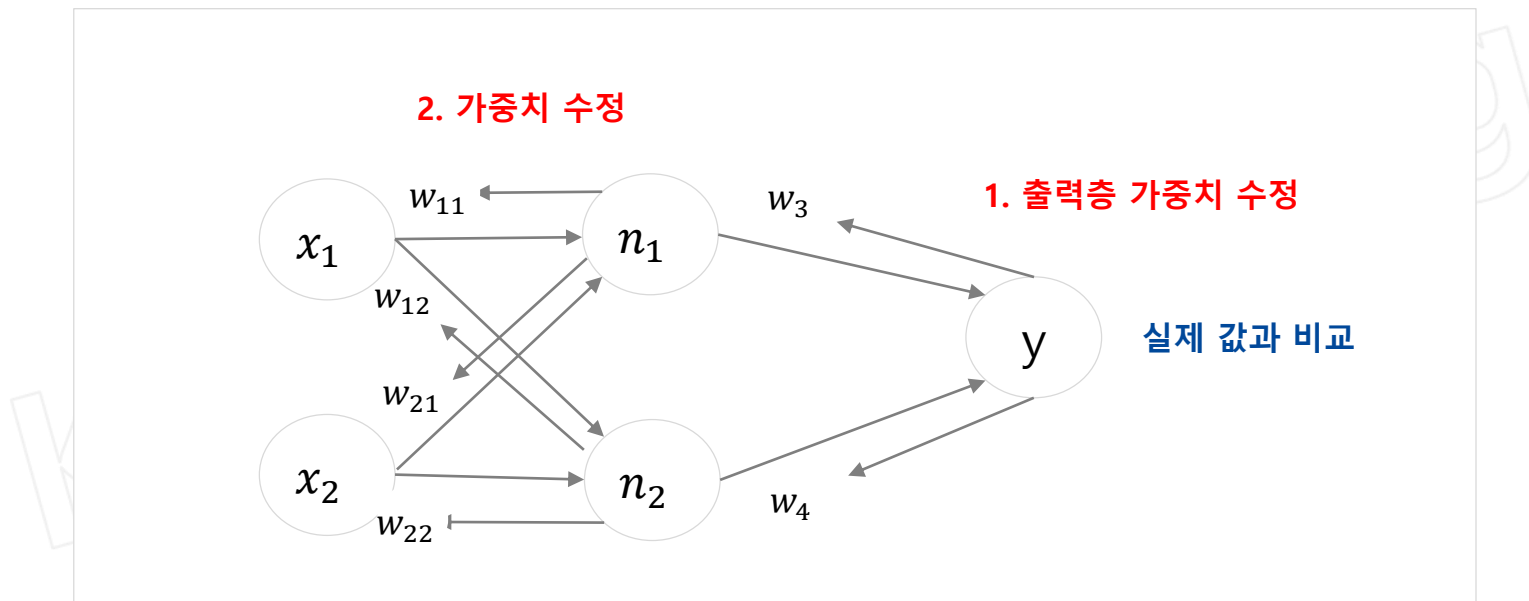
오차 역전파[Back Propagation]

- 단일 퍼셉트론에서의 오차 수정
 - 결과값을 얻으면 오차를 구해 앞 단계에서 정한 가중치를 조정



오차 역전파[Back Propagation]

- 다층 퍼셉트론에서의 오차 수정
 - 결과값을 얻으면 오차를 구해 하나 앞선 단계에서 정한 가중치를 조정



최적화의 계산 방향이 출력층에서 시작해 앞으로 진행
다층 퍼셉트론에서의 최적화 과정을 오차 역전파 라고 함

오차 역전파[Back Propagation]

- 구동 방식

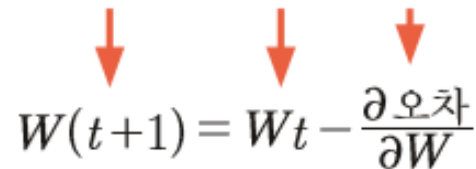
임의의 초기 가중치(W)를 준 뒤 결과를 계산

계산 결과와 우리가 원하는 값 사이의 오차를 구함

경사 하강법을 이용해 바로 앞 가중치를 **오차가 작아지는 방향**으로 업데이트

- 미분 값이 0에 가까워지는 방향으로 나아간다는 의미
- 가중치에서 기울기를 빼도 값의 변화가 없을 때까지 계속해서 가중치 수정 작업 반복하는 것

새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값

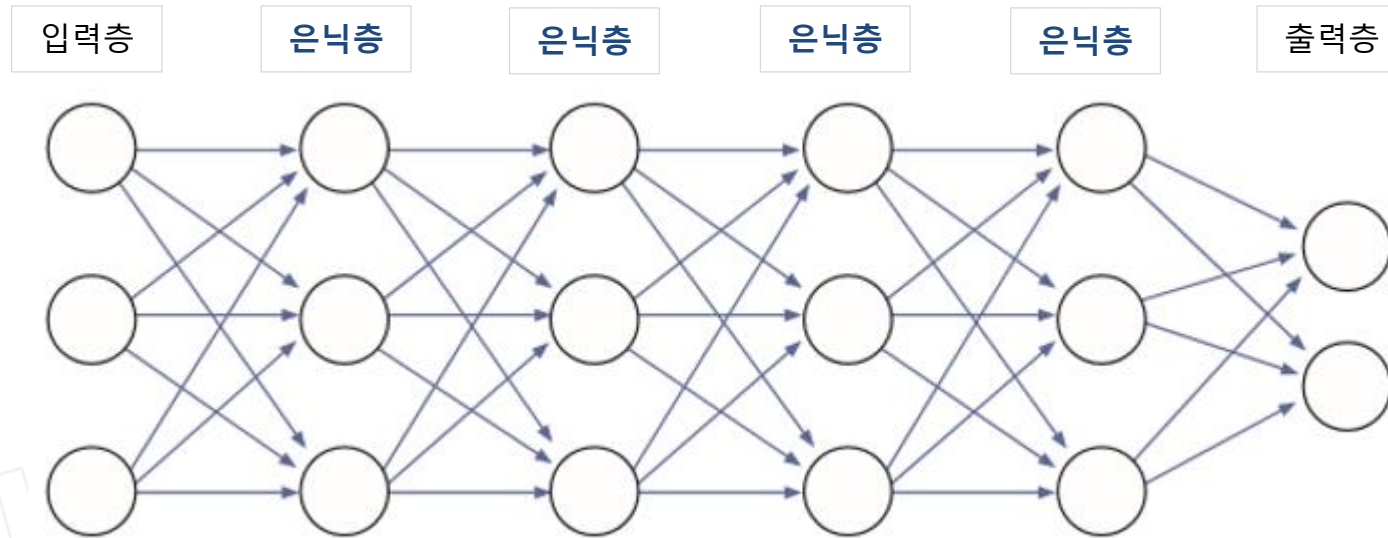

$$W(t+1) = W_t - \frac{\partial \text{오차}}{\partial W}$$

위 과정을 더 이상 오차가 줄어들지 않을 때 까지 반복

| 신경망에서 딥러닝으로

딥러닝시 고려 사항

- 딥러닝



- Layer가 많아 질수록?
 - 기대 : 예측의 정확도가 올라갈듯 하다
 - 현실 : 기대만큼 좋지 않더라...
 - 왜 이런가?

딥러닝시 고려 사항

- 현실 : 기대만큼 좋지 않더라...
- 왜 이런가?

기울기 소실 문제와 활성화 함수

- 층이 늘어 날수록 기울기가 중간에 0이 되어 버리는 기울기 소실 문제가 발생 될 수도 있음
- 시그모이드 함수를 사용하면 미분 할 수록 곱의 값이 0에 가까워짐
- 따라서 기울기가 사라짐
- 결론 가중치를 수정하기 어려움

경사 하강법

- 경사 하강법은 불필요하게 많은 계산량의 속도를 느리게 할 뿐 아니라, 최적의 해를 찾기 전에 최적화 과정이 멈출 수도 있음

딥러닝시 고려 사항

- 현실 : 기대만큼 좋지 않더라...
- 왜 이런가?

활성화 함수의 대체

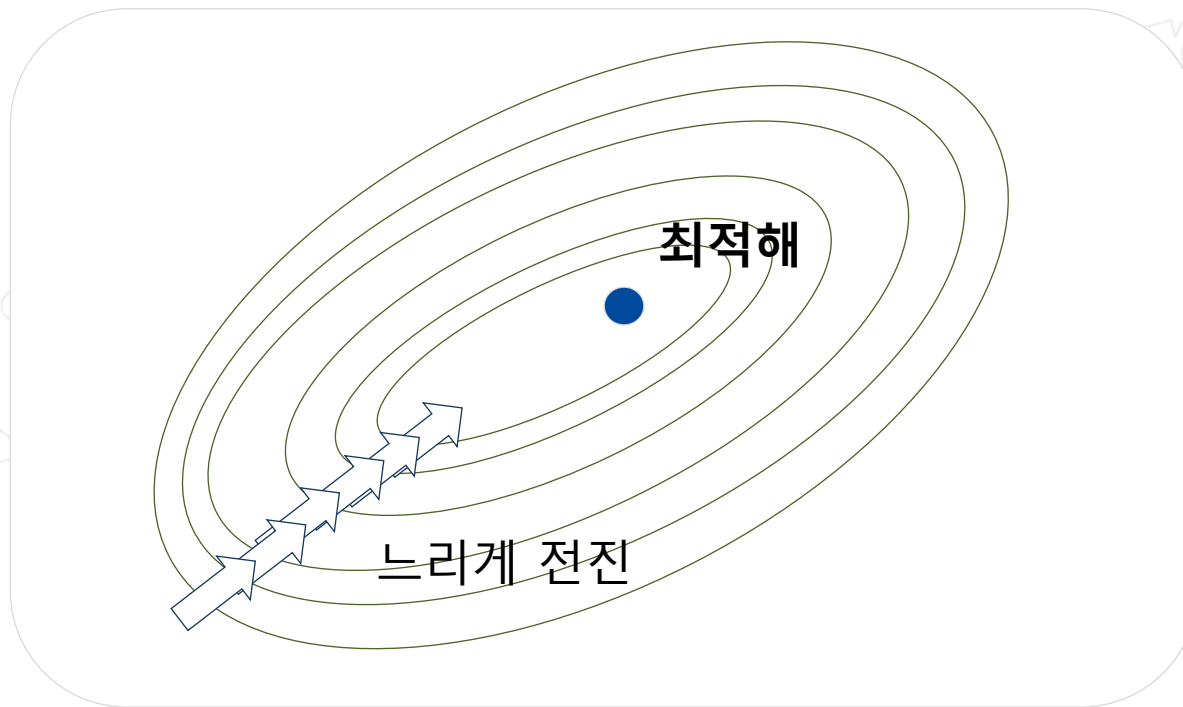
- 해결책 : 시그모이드 함수가 아닌 다른 함수로 대체

확률적 경사 하강법
(Stochastic Gradient Descent, SGD)

- 전체 데이터 대신 랜덤하게 추출한 일부 데이터를 사용

딥러닝시 고려 사항

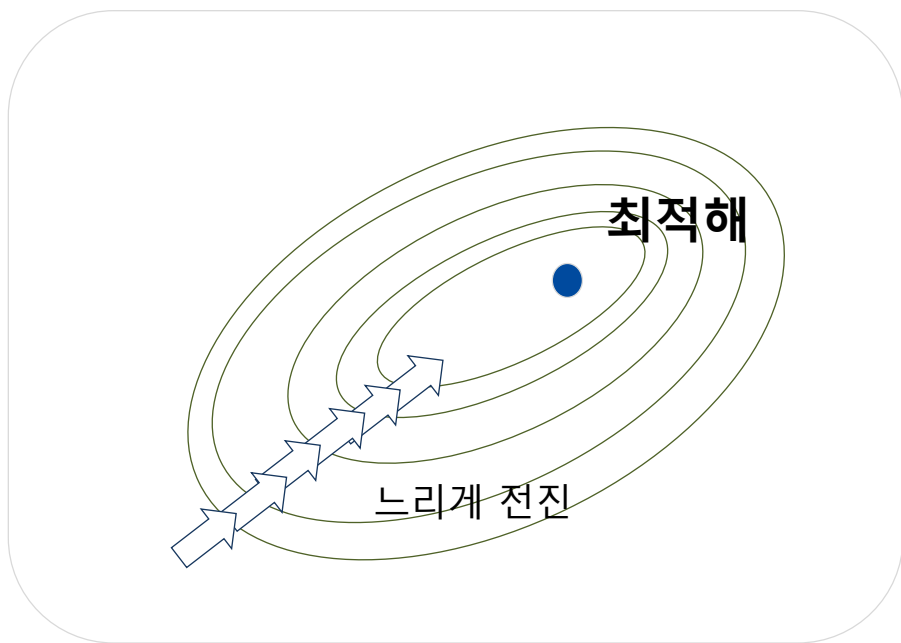
- 속도와 정확도 문제를 해결하는 고급 경사 하강법
 - 경사 하강법은 정확하게 가중치를 찾아가지만, 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있음



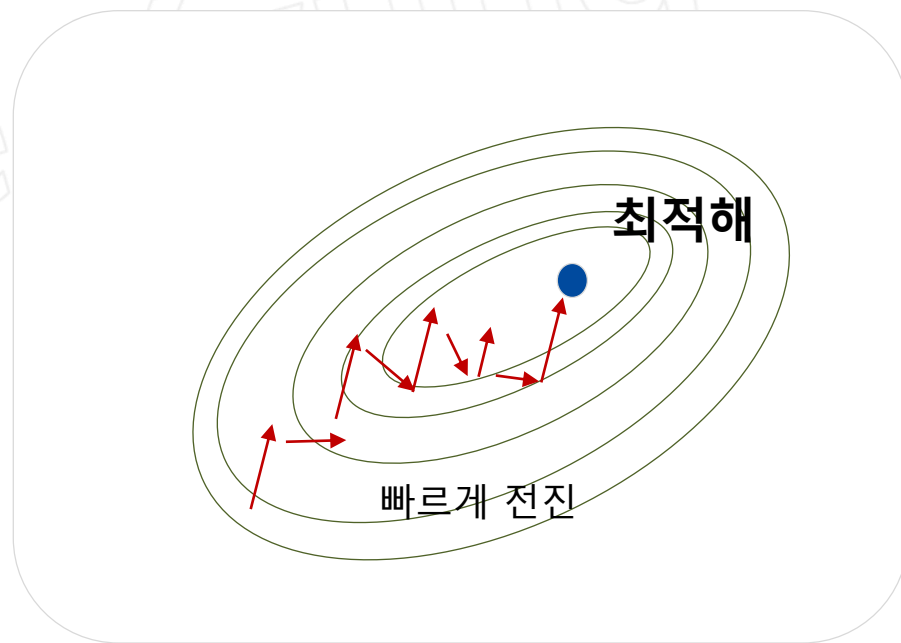
경사 하강법

딥러닝시 고려 사항

- 속도와 정확도 문제를 해결하는 고급 경사 하강법
 - 확률적 경사 하강법(SGD)
 - 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용
 - 일부 데이터를 사용하므로 더 빨리 그리고 자주 업데이트를 하는 것이 가능해짐



경사 하강법



확률적 경사 하강법

딥러닝시 고려 사항[식과 Keras API]

- 확률적 경사 하강법

- x_i 를 학습 샘플이라고 할 때 식으로 표현하면 다음과 같음
- 여기서 η 는 학습률을 나타냄
- 순수 파이썬 식과 Keras 함수 비교

$$W(t+1) = W(t) - \eta \frac{\partial \text{오차}}{\partial w} \quad \text{단, } x^{(i+n)}$$

```
self.weight[i] += - learning_rate * gradient
```

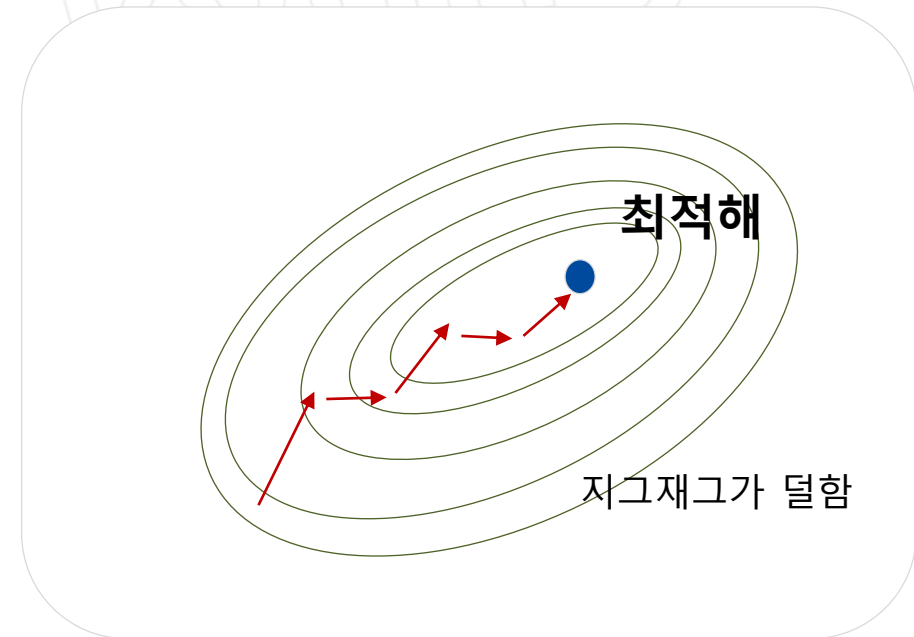
```
keras.optimizers.SGD(lr=0.1)
```

딥러닝시 고려 사항

- 속도와 정확도 문제를 해결하는 고급 경사 하강법
 - 모멘텀
 - 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

딥러닝시 고려 사항[식과 Keras API]

- 모멘텀(Momentum) 방법을 식으로 표현하면 다음과 같음
 - γ : 앞서 구한 오차를 어느 정도(%) 반영할지를 정하는 ‘모멘텀 계수’

$$V(t) = \gamma v(t-1) - \eta \frac{\partial \text{오차}}{\partial w}$$
$$W(t+1) = W(t) + V(t)$$

```
v = m * v - learning_rate * gradient  
self.weight[i] += v
```

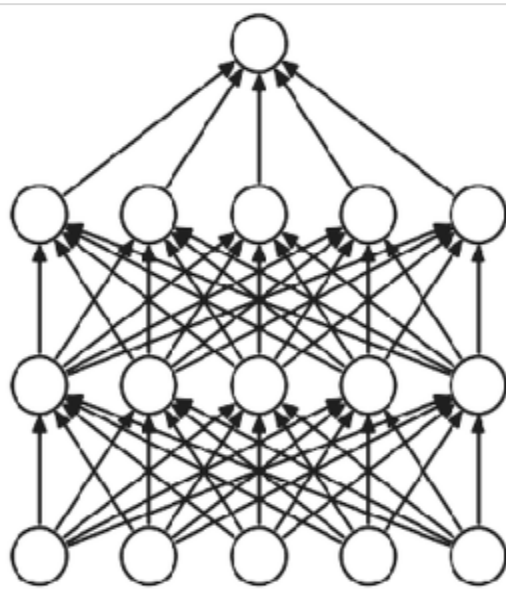
```
keras.optimizers.SGD(lr=0.1, momentum=0.9)
```

딥러닝시 고려 사항

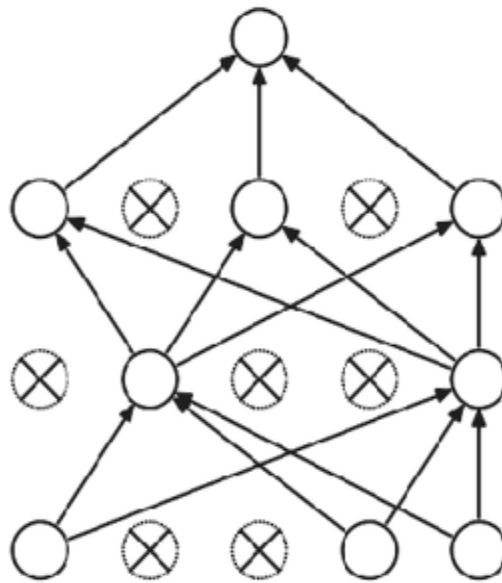
- 과적합(overfitting) :
 - 모델이 학습 데이터셋 안에서는 일정 수준 이상의 예측 정확도를 보이지만, 새로운 데이터에 적용하면 잘 맞지 않는 것을 의미
 - 학습한 데이터에만 종속적으로 맞춰져 있어 그 외 데이터는 잘 맞지 않는 상황을 의미
- Overfitting에 대한 해결책
 - **Dropout(망 부분 생략) 적용한 분류 DNN**
 - 드롭아웃(Dropout) :
 - 과적합을 해결하는 가장 효과가 좋은 방법 하나
 - 단순한 기법을 통해 신경망의 일반화 성능을 향상시킬 수 있는데, 학습시킬 때 무작위로 뉴런을 '드롭아웃(제외)'
 - 드롭아웃을 적용하면 실질적으로 많은 모델을 생성, 학습하고 그 안에서 예측을 실행하기 때문에 성능이 향상됨

딥러닝시 고려 사항

- 드롭아웃(Dropout)



(a) 일반 신경망



(b) 드롭아웃을 적용한 신경망

망에 있는 입력 layer나 hidden layer의 일부 뉴런을 생략(drop out)하고 줄어든 신경망을 통해 학습 수행

일정 mini-batch 구간 동안 생략된 망에 대한 학습을 끝내면 다시 무작위로 다른 뉴런들을 생략하면서 반복적으로 학습 수행



Keras

- Keras 설치
 - Tensorflow 가 설치되어 있는 환경에 설치

```
C:\Users\Playdata>activate my_tensorflow
(my_tensorflow) C:\Users\Playdata>pip list
Package            Version
-----
absl-py            0.6.1
astor              0.7.1
backcall          0.1.0
bleach            3.1.0
certifi           2018.11.29
colorama          0.4.1
cycler            0.10.0
decorator         4.3.0
defusedxml        0.5.0
entrypoints       0.3
gast              0.2.2
grpcio            1.17.1
h5py              2.9.0
ipykernel         5.1.0
ipython           7.2.0
ipython-genutils  0.2.0
ipywidgets       7.4.2
jedi              0.13.2
Jinja2            2.10
jsonschema        2.6.0
jupyter           1.0.0
jupyter-client    5.2.4
jupyter-console   6.0.0
jupyter-core      4.4.0
Keras-Applications 1.0.6
Keras-Preprocessing 1.0.5
kiwisolver        1.0.1
Markdown          3.0.1
```

>pip install keras

```
Keras                2.2.4
Keras-Applications  1.0.6
Keras-Preprocessing  1.0.5
```

Keras API

- 속도와 정확도 문제를 해결하는 고급 경사 하강법
 - 고급 경사 하강법과 케라스 내부에서의 활용법

고급 경사 하강법	개요	효과	케라스 사용법
1. 확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	<code>keras.optimizers.SGD(lr = 0.1)</code> 케라스 최적화 함수를 이용합니다.
2. 모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	<code>keras.optimizers.SGD(lr = 0.1, momentum = 0.9)</code> 모멘텀 계수를 추가합니다.
3. 네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	<code>keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True)</code> 네스테로프 옵션을 추가합니다.

Keras API

4. 아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	<code>keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6)</code> 아다그라드 함수를 사용합니다. ※ 참고: 여기서 <code>epsilon</code> , <code>rho</code> , <code>decay</code> 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 <code>lr</code> , 즉 <code>learning rate</code> (학습률) 값만 적절히 조절하면 됩니다.
5. 알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	<code>keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0)</code> 알엠에스프롭 함수를 사용합니다.
6. 아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	<code>keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0)</code> 아담 함수를 사용합니다.

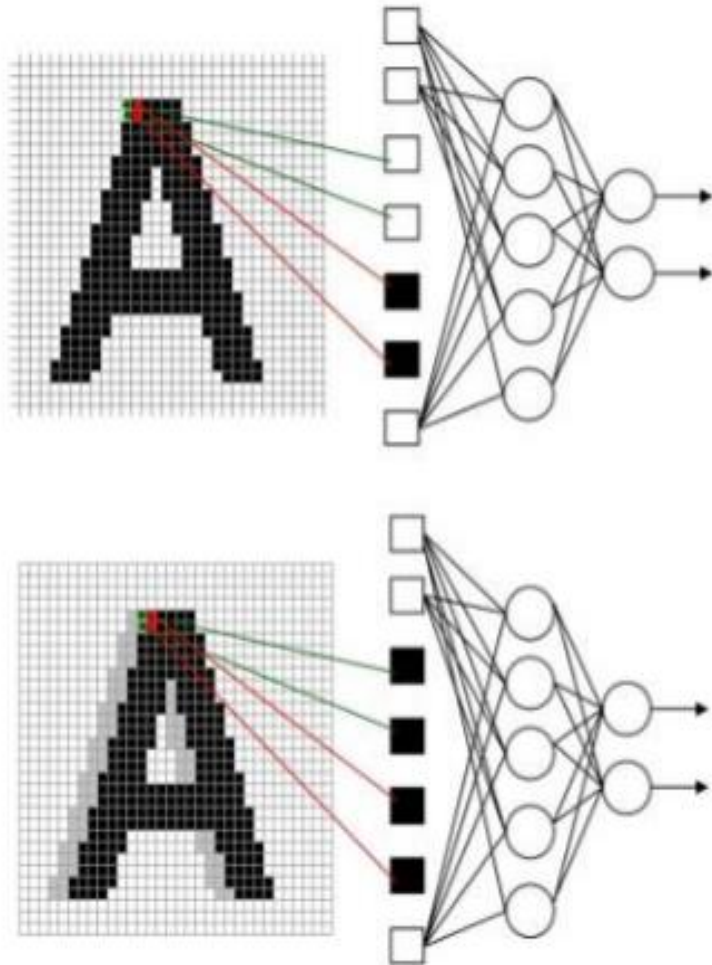
컨볼루션 신경망(CNN)

컨볼루션 신경망(CNN)

- 이미지 인식의 꽃
- 합성곱 신경망 (CNN) 은 이미지 인식 분야에서 강력한 성능을 발휘
- 음성인식이나, 자연어 처리 등에서 사용되며 활용성에서도 매우 뛰어남
- 일반 신경망의 경우, 이미지 데이터를 그대로 처리
- 즉, 이미지 전체를 하나의 데이터로 생각해서 입력으로 받아들이기 때문에, 이미지의 특성을 찾지 못하고 위와 같이 이미지의 위치가 조금만 달라지거나 왜곡된 경우에 올바른 성능을 내지 않음
- 해결책 :

합성곱 신경망(CNN)은 이미지를 하나의 데이터가 아닌, 여러 개로 분할하여 처리

컨볼루션 신경망(CNN)



컨볼루션 신경망(CNN)

- 실습 데이터 set
 - MNIST
 - 고등학생과 인구조사국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성되어 있음
 - 70,000개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋
 - 머신러닝을 배우는 사람이라면 자신의 알고리즘과 다른 알고리즘의 성과를 비교해 보고자 한 번씩 도전해 보는 가장 유명한 데이터 중 하나



컨볼루션 신경망(CNN)

- 컨볼루션 신경망은 입력된 이미지에서 다시 한번 특징을 추출하기 위해 마스크(필터, 윈도우 또는 커널이라고도 함)를 도입하는 기법
- 예 :입력된 이미지가 다음과 같은 값을 가지고 있음

1	0	1	0
0	1	1	0
0	0	1	1
0	0	1	0

2 x 2 마스크
각 칸에는 가중치 보유

가중치 값이 x 1, x 0이라 가정

x1	x0
x0	x1

컨볼루션 신경망(CNN)

x1	x0	1	0
x0	x1	1	0
0	0	1	1
0	0	1	0

적용된 부분은 원래 있던 값에 가중치의 값을 곱한
결과를 합하면 새로 추출된 값인 2가 됨

$$(1 \times 1) + (0 \times 0) + (0 \times 0) + (1 \times 1) = 2$$

컨볼루션 신경망(CNN)

- 마스크를 한 칸씩 옮겨 모두 적용

1x1	0x0	1	0
0x0	1x1	1	0
0	0	1	1
0	0	1	0

1	0x1	1x0	0
0	1x0	1x1	0
0	0	1	1
0	0	1	0

1	0	1x1	0x0
0	1	1x0	0x1
0	0	1	1
0	0	1	0

1	0	1	0
0x1	1x0	1	0
0x0	0x1	1	1
0	0	1	0
1	0	1	0
1	0	1	0
0	1x1	1x0	0
0	0x0	1x1	1
0	0	1	0
1	0	1	0
0	1	1	0
0	1	1	0
0x1	0x0	1	1
0x0	0x1	1	0
1	0	1	0
1	0	1	0
0	0x1	1x0	1
0	0x0	1x1	0
0	0	1x1	1x0
0	0	1x0	0x1

결과

2	1	1
0	2	2
0	1	1

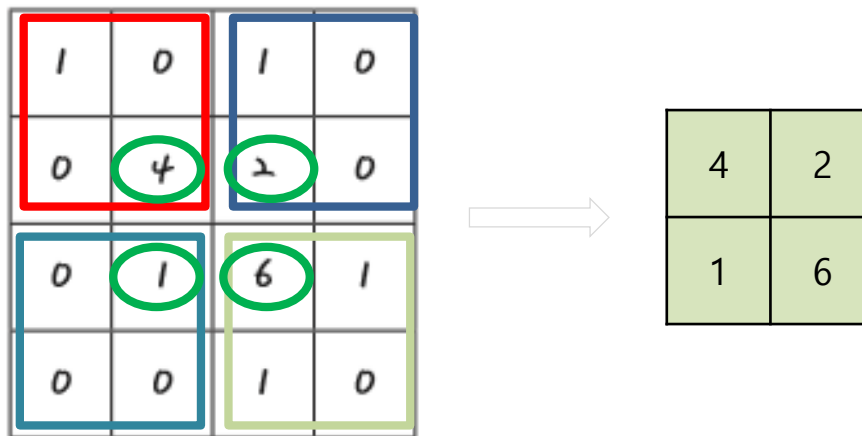
이처럼 새롭게 만들어진 층을 컨볼루션(합성곱)이라고 함

컨볼루션을 만들면 입력 데이터로부터 더욱 정교한 특징 추출 가능

마스크를 여러 개 만들 경우 여러개의 컨볼루션이 구성됨

컨볼루션 신경망(CNN)

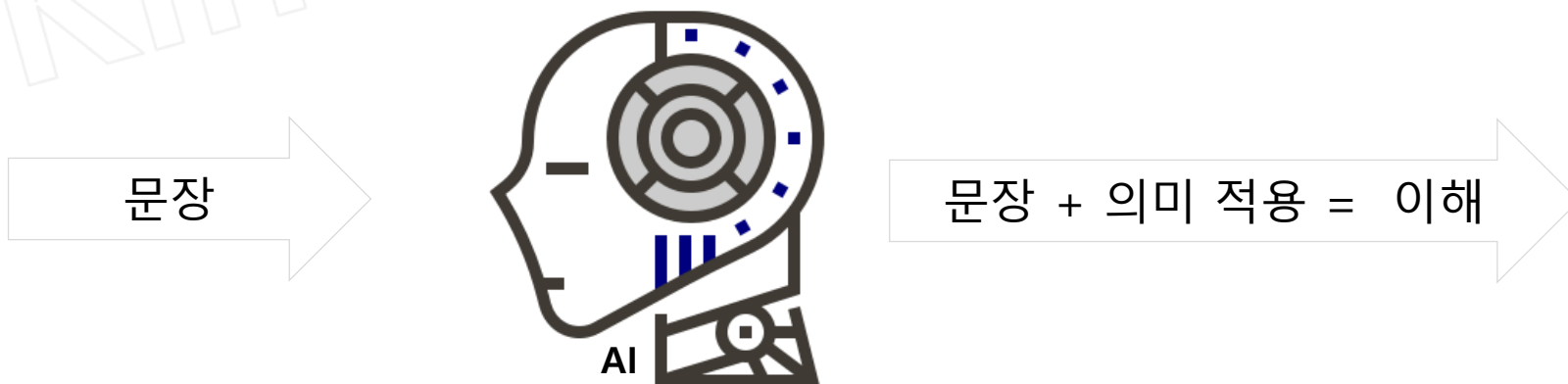
- 맥스 풀링 층을 추가
- 컨볼루션 층을 통해 이미지 특징을 도출 한 경우 그 결과가 여전히 크고 복잡하면 이를 다시 한번 축소해야 함
 - 과정을 풀링(pooling) 또는 서브 샘플링(sub sampling)이라고 함
 - 풀링 기법 중 가장 많이 사용되는 방법이 맥스 풀링(max pooling)
 - 정해진 구역 안에서 가장 큰 값만 다음 층으로 넘기고 나머지는 버림
 - 이 과정을 거쳐 불필요한 정보를 간추림
 - 맥스 풀링은 `MaxPooling2D()` 함수를 사용



| 순환 신경망(RNN)
Recurrent Neural Network

순환 신경망(RNN)

- AI[Artificial Intelligence, 인공지능]
 - 업무를 도와주고 삶의 질을 높여주는 인공지능 비서 서비스
 - 대화형 인공지능
 - 예시 : 네이버 클로바, 삼성의 빅스비, 애플의 시리, 구글의 어시스턴스등
 - 필요 사항
 - 사람의 언어를 이해하는 것
 - 문장을 듣고 무엇을 의미하는 지를 알아야 함



순환 신경망(RNN)

- 문장을 듣고 이해한다?
 - 많은 문장을 이미 학습해 놓았다는 의미
 - 과거에 입력된 데이터와 나중에 입력된 데이터 사이의 관계를 고려해야 함



순환 신경망(RNN)

- “문장을 듣고 이해한다?” 어떻게?
 - 순환 신경망(Recurrent Neural Network, RNN) 방법 고안

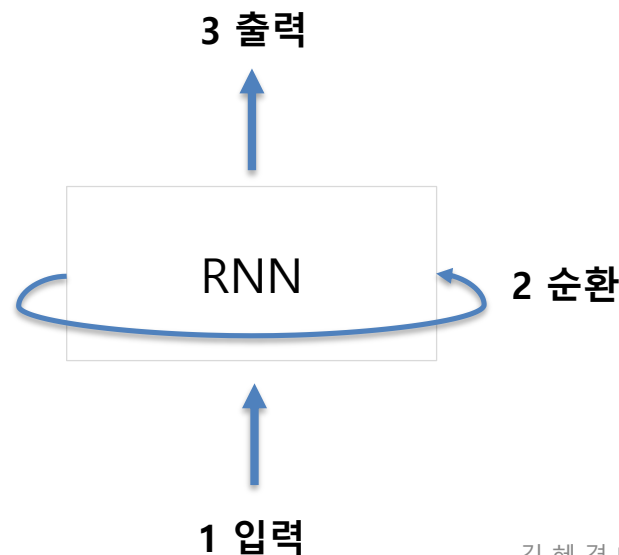


순환 신경망(RNN)

- 순환 신경망(Recurrent Neural Network, RNN) 이란?
 - 계층의 출력이 순환하는 인공신경망
 - 순환 방식
 - 은닉 계층의 결과가 다음 계층으로 넘어갈 뿐 아니라 자기 계층으로 다시 들어옴
 - 시계열 정보처럼 앞뒤 신호가 서로 상관도가 있는 경우 인공신경망의 성능을 더 높일 수 있음

순환 신경망(RNN)

- 여러 개의 데이터가 순서대로 입력되었을 때 앞서 입력 받은 데이터를 기억해 놓는 방법
- 모든 입력 값에 이 작업을 순서대로 실행하므로 다음 층으로 넘어가기 전에 같은 층을 맴도는 것처럼 보임
 - 같은 층을 맴도는 성질 때문에 순환 신경망이라 함
 - 일반 신경망과 순환 신경망의 차이



순환 신경망(RNN)

- RNN의 개선을 위한 노력

- 단점

- 한 층 내에서 반복을 많이 해야 하는 특성상 일반 신경망보다 기울기 소실 문제가 더 많이 발생

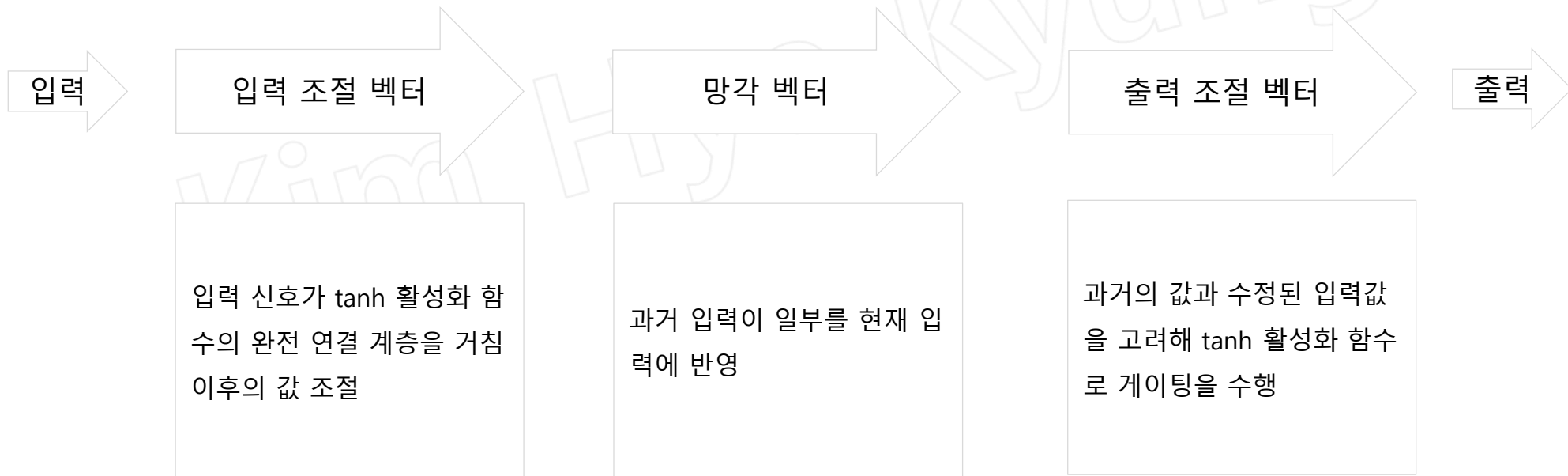
- 해결책

- LSTM[Long Short Term Memory]

- 반복되기 직전에 다음 층으로 기억된 값을 넘길지 안 넘기지를 관리하는 단계 추가

순환 신경망(RNN)

- LSTM
 - 입력과 출력 신호를 게이팅(gating)
 - 신호의 양을 조절해 주는 기법
 - LSTM 실행 구조



순환 신경망(RNN)

- RNN 응용 분야
 - 시퀀스 투 시퀀스 RNN
 - 날씨 예측 또는 다른 시계열 문제, 기계번역, 음악 생성 등
 - 시퀀스 투 벡터 RNN
 - 음악 샘플로 장르 구별, 책 후기에 대한 감성 분석, 사용자의 영화 시청 이력을 바탕으로 보고 싶어 할 영화의 확률을 예측
 - 벡터 투 시퀀스 RNN
 - 이미지 캡션 생성, 현재 아티스트를 기반으로 음악 플레이리스트 생성. 일련의 파라미터를 기반으로 한 멜로디 생성,

참고 문헌 및 서적

- 모두의 딥러닝[길벗] – 조태호 지음
- 모두를 위한 머신러닝/딥러닝 강의 (홍콩 과기대 김성훈 교수님 강좌)

Kim Hye Kyung