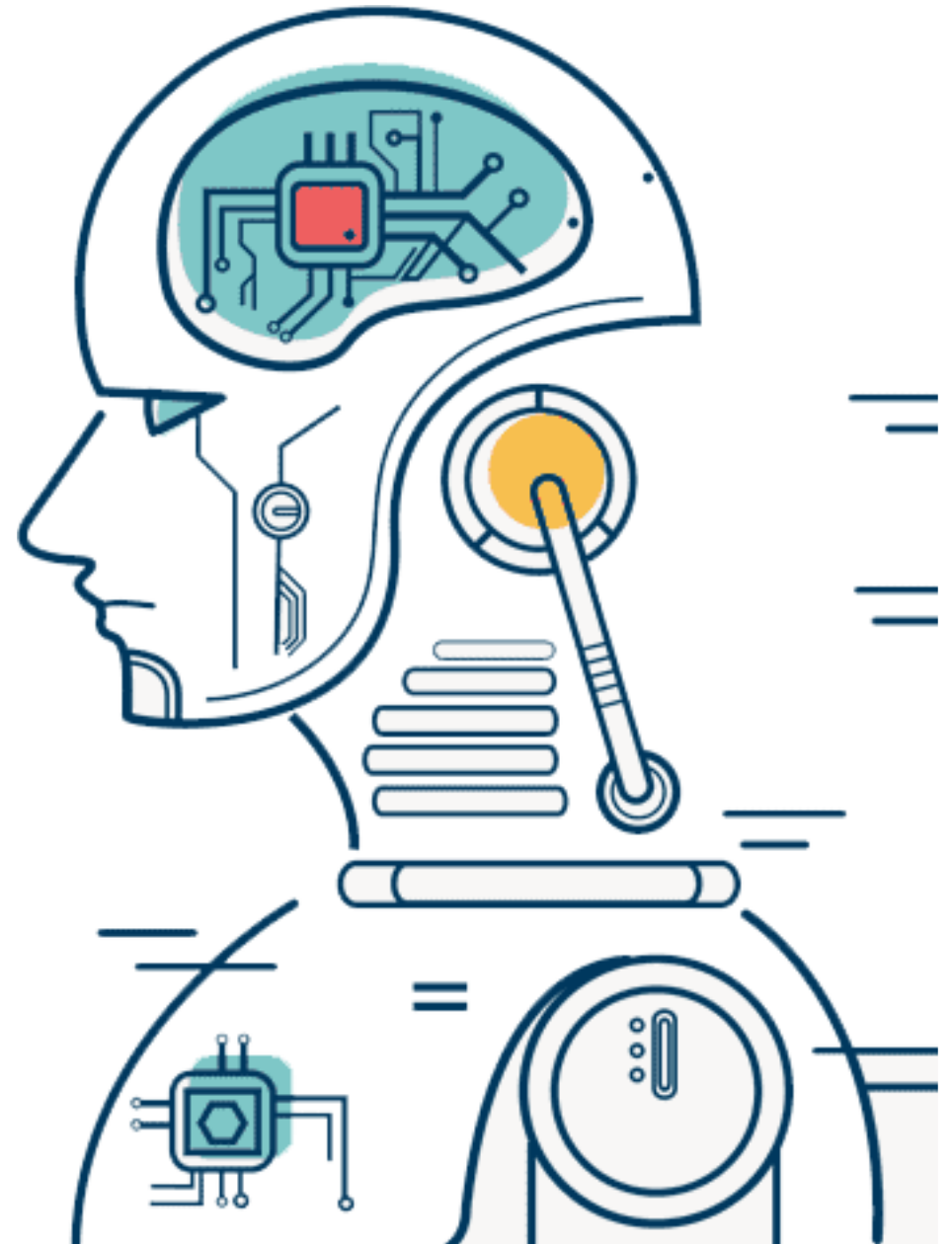




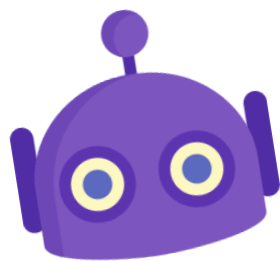
Smart Media  
스마트미디어인재개발원

# Deep Learning

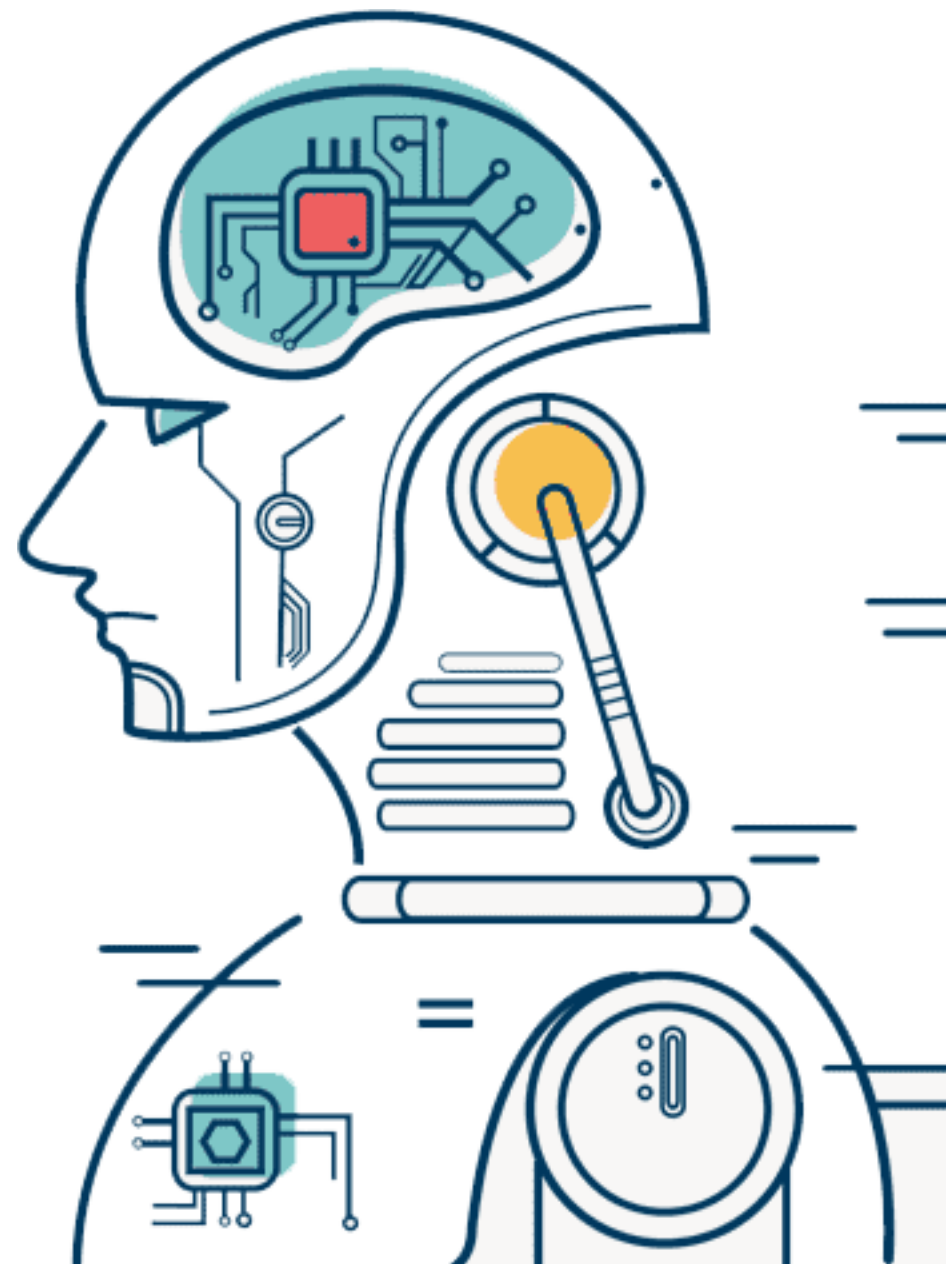
## Chapter 1 딥러닝 개요 (개념, 역사, 환경구축)

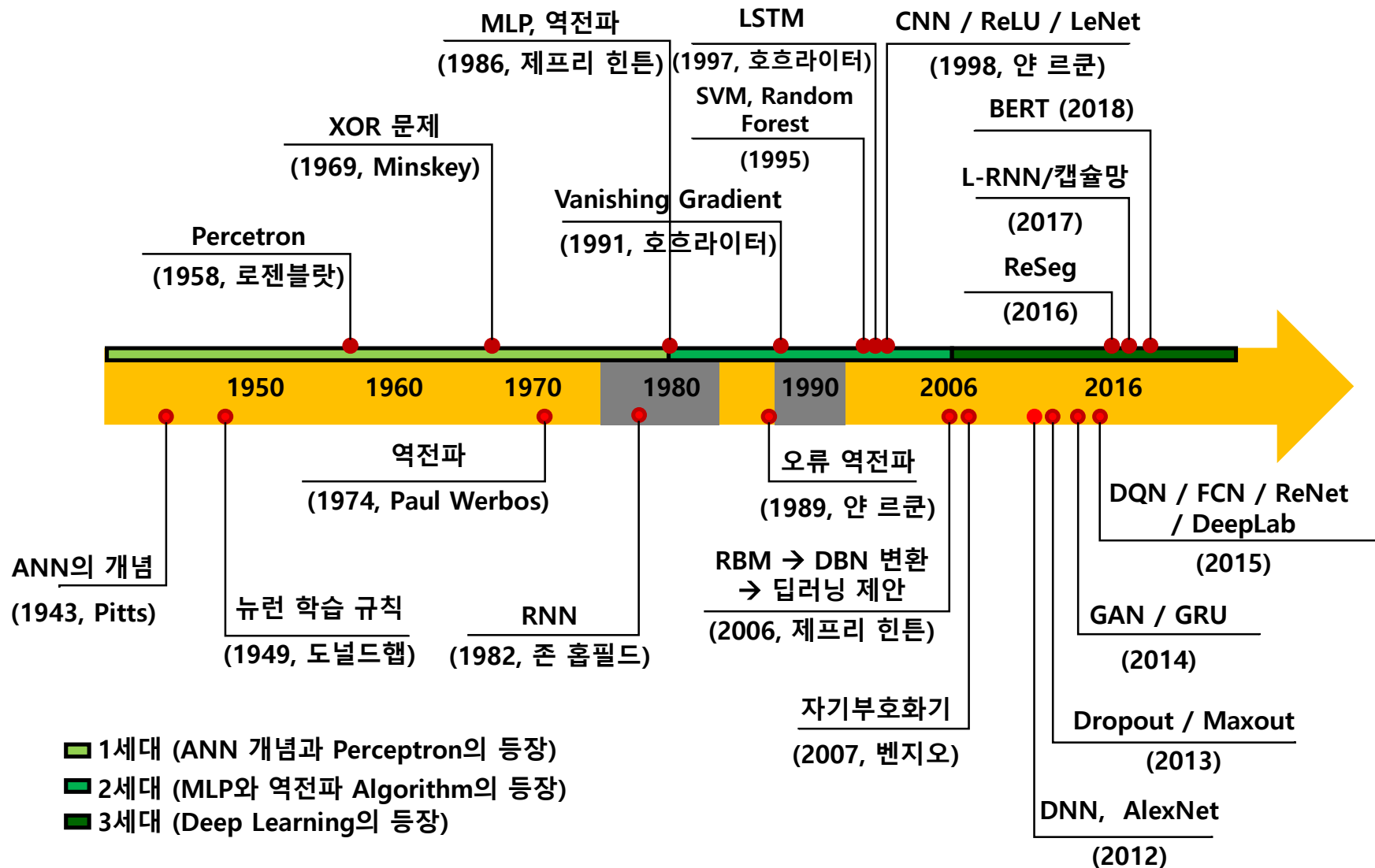


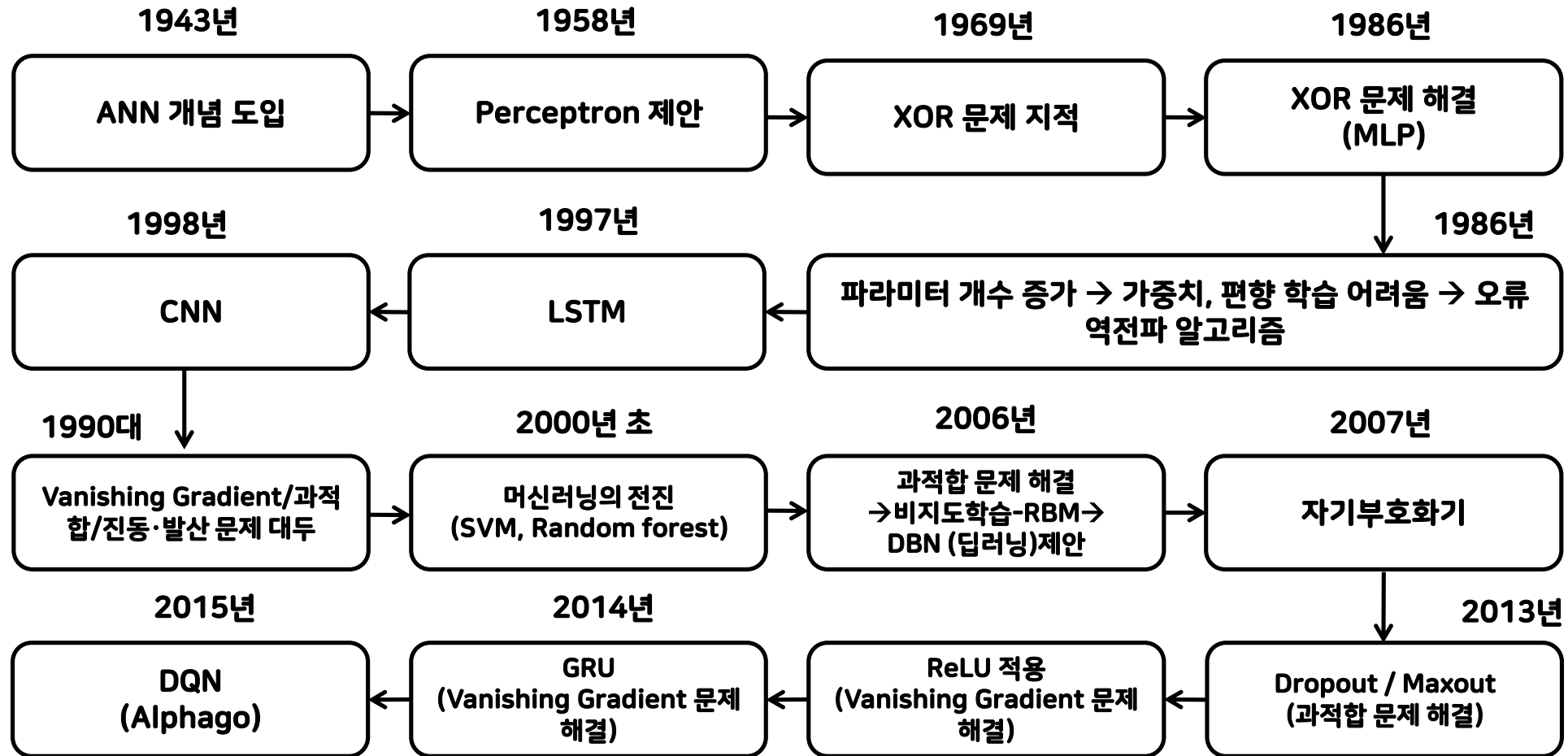
- **딥러닝의 개념을 이해 할 수 있다.**
- **딥러닝의 역사를 알 수 있다.**
- **딥러닝 개발환경을 구축 할 수 있다.**
- **Keras를 활용하여 간단한 딥러닝 학습을 수행할 수 있다.**

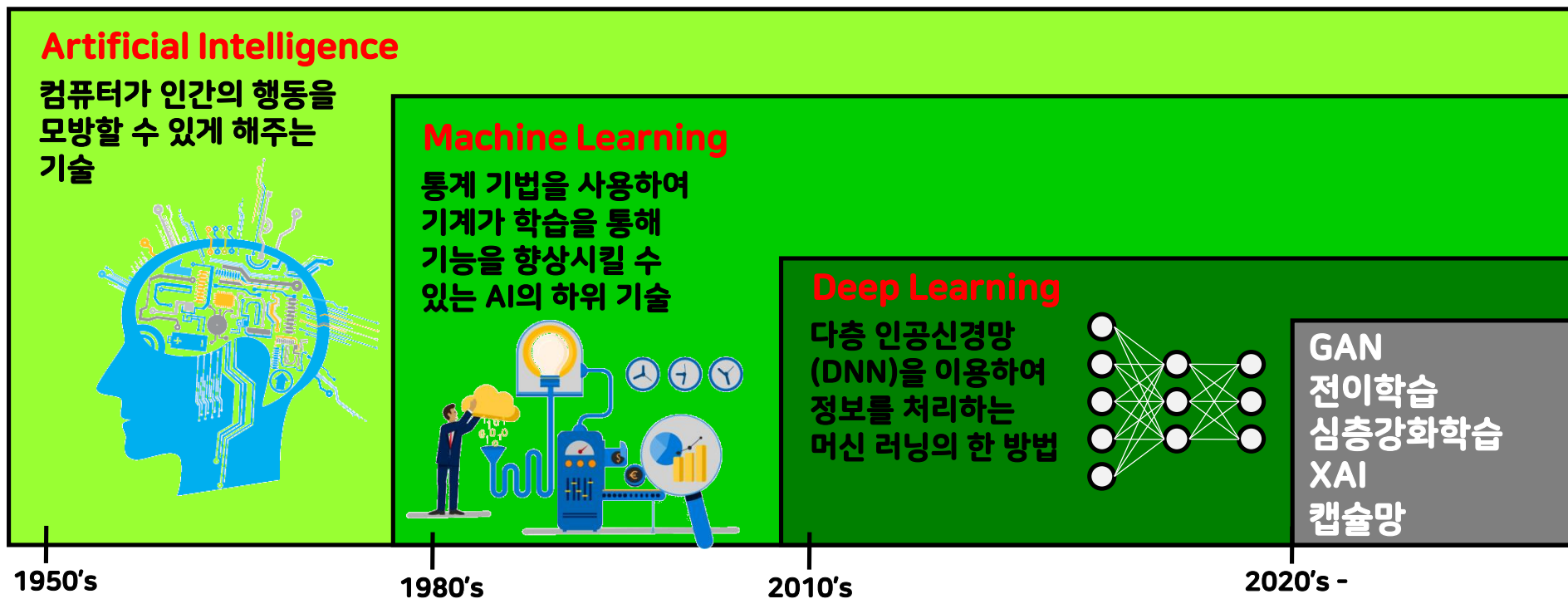


# 딥러닝 개요

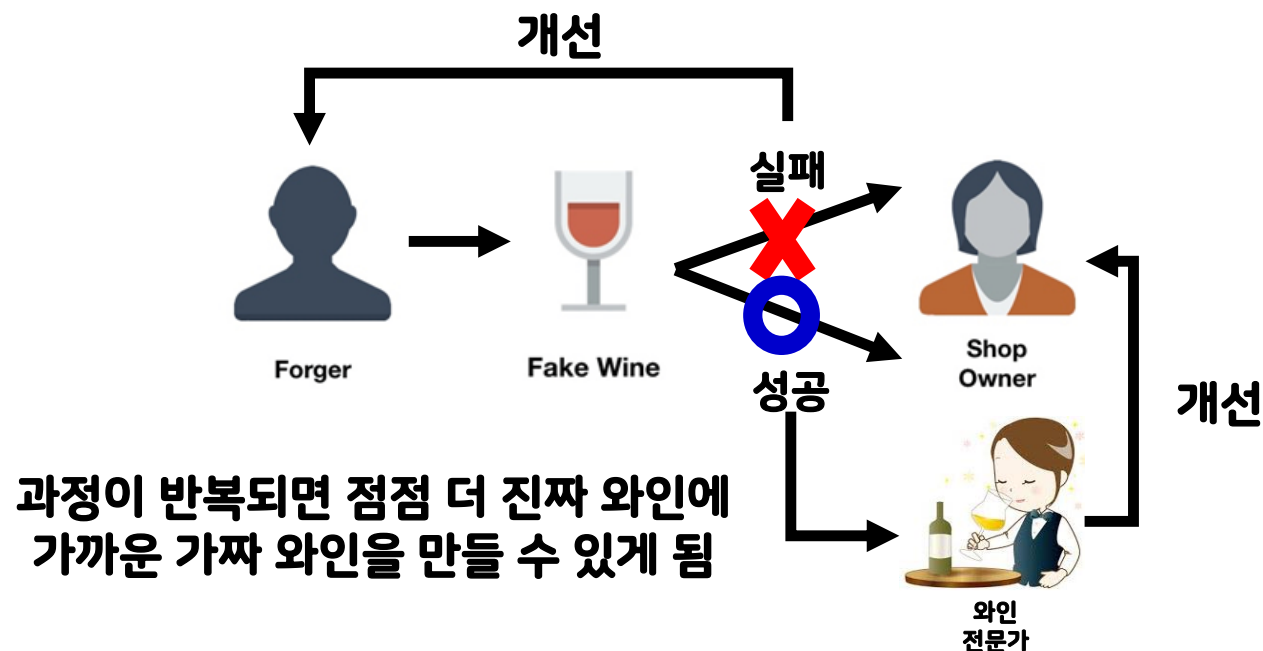




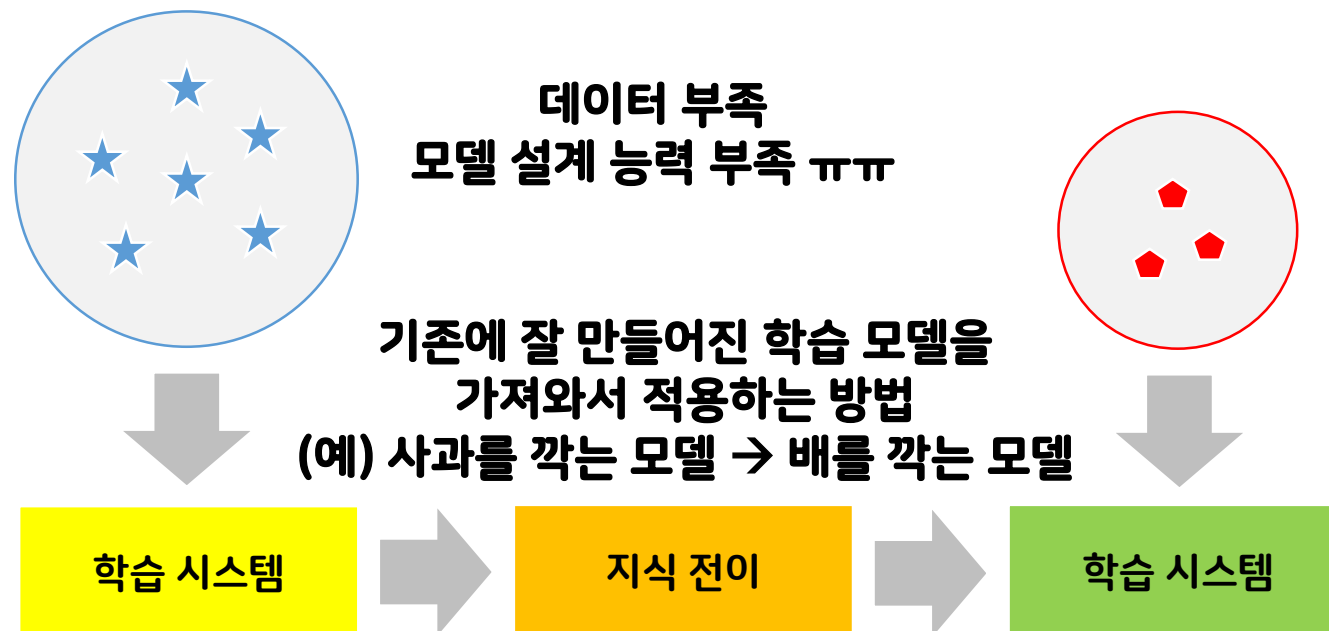




## GAN (적대적신경망, Generative Adversarial Networks)

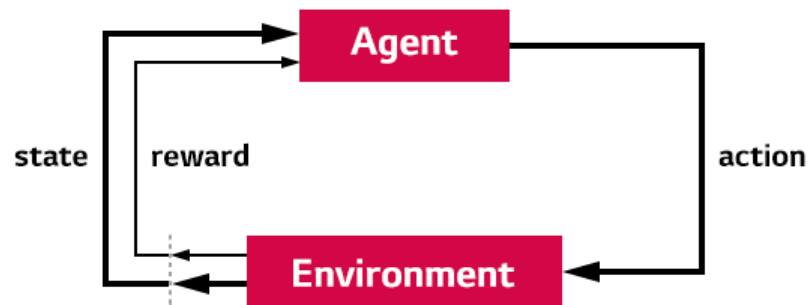


## 전이학습 (Transfer Learning)

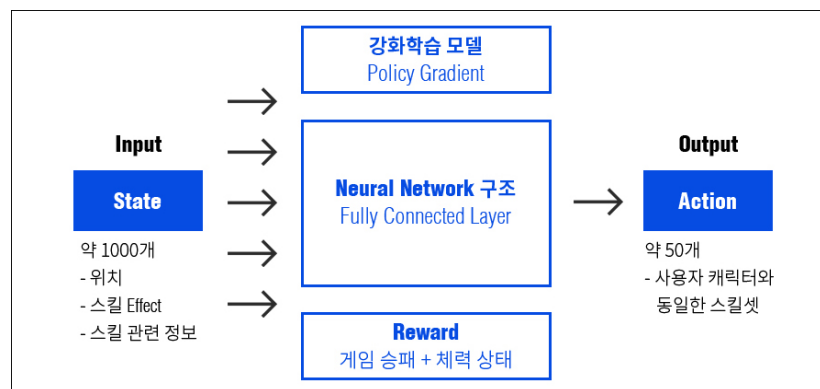




## 심층강화학습 (Deep Reinforcement Learning)

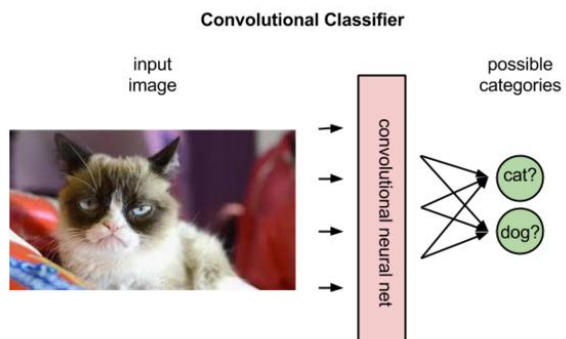


강화학습

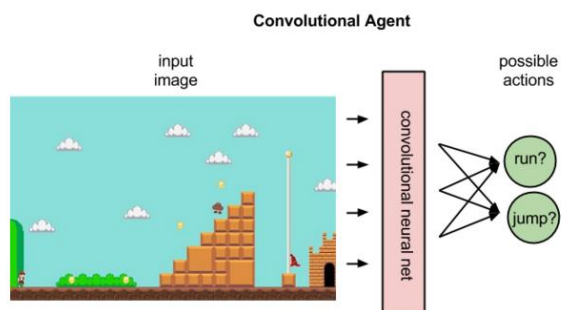


심층강화학습

## 심층강화학습 (Deep Reinforcement Learning)

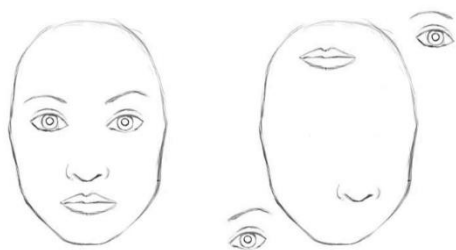


CNN은 이미지에 가장 잘 맞는  
(가능성이 높은, 순위가 높은)  
라벨을 선택  
(고양이 70%, 당나귀 40%, 개 30%)



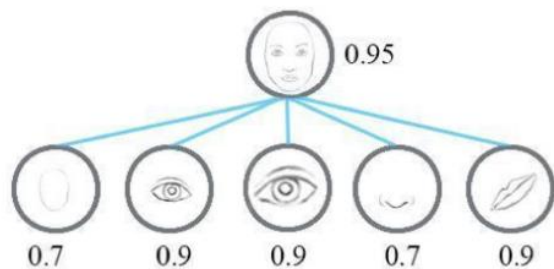
심층강화학습은 행동에 대해  
순위를 계산하고 점수(보상)을 결정  
(오른쪽 점프 5점, 점프 7점, 왼쪽  
점프 0점)

## 캡슐망 (Capsule Network)



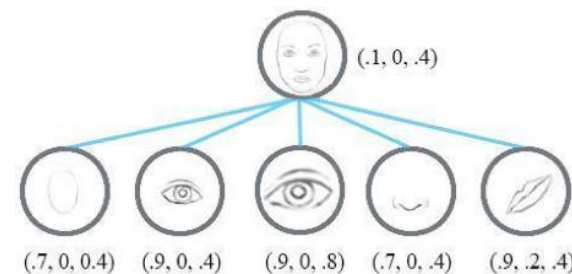
CNN은 유사한  
객체로 인식

탐색 속도 증가 → Maxpooling  
→ 공간 관계 상실 → 시점 변화에 취약



CNN

Maxpooling → 동적라우팅 (공간정보 포함)  
Scalar-output-dectector → Vector-ouput-capsule



캡슐망

## 딥러닝(Deep Learning)

사람의 신경망을 모방하여 기계가  
병렬적 다층 구조를 통해 학습하도록 만든 기술

기계 vs 사람

자동차입니다.  
자동차가 아닙니다.



기계는 판단하는 기준이 명확히 정해져 있다  
하지만 사람은 대상을 판단하는 경계가 느슨하다(추상적)

바퀴	있다
날개	없다
모양	사각형



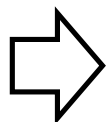
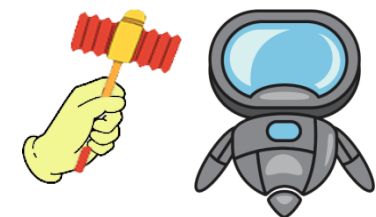
자동차를  
구분하는 기계



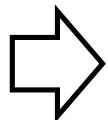
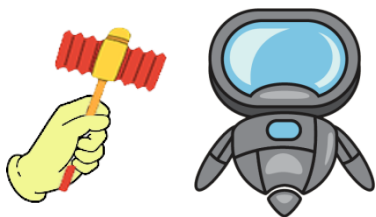
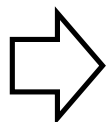
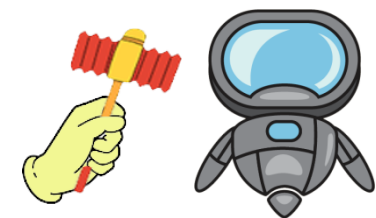
바퀴	있다
날개	없다
모양	찌그러진 사각형

## 기계 vs 사람

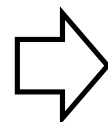
동일한 자극



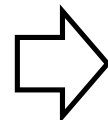
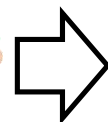
동일한 반응



동일한 자극

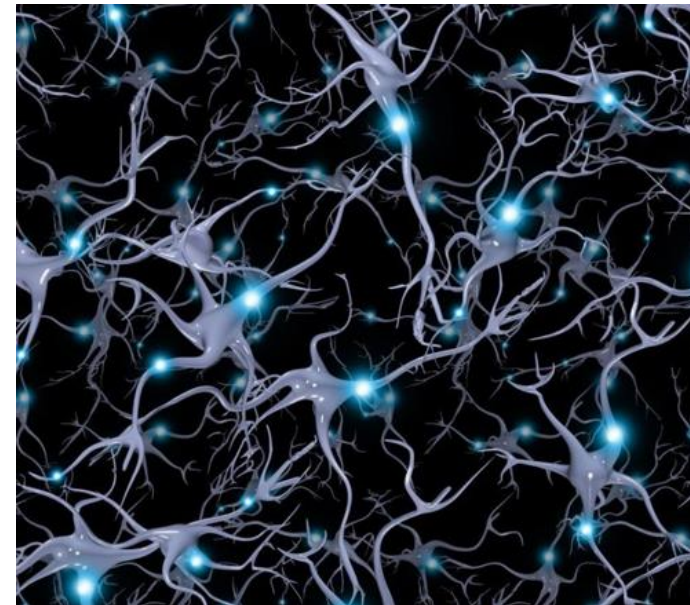
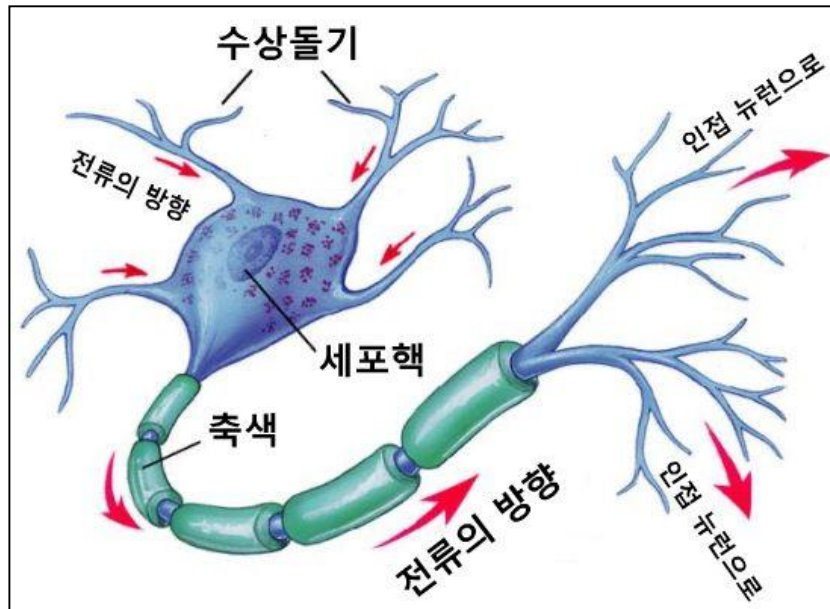


다양한 반응



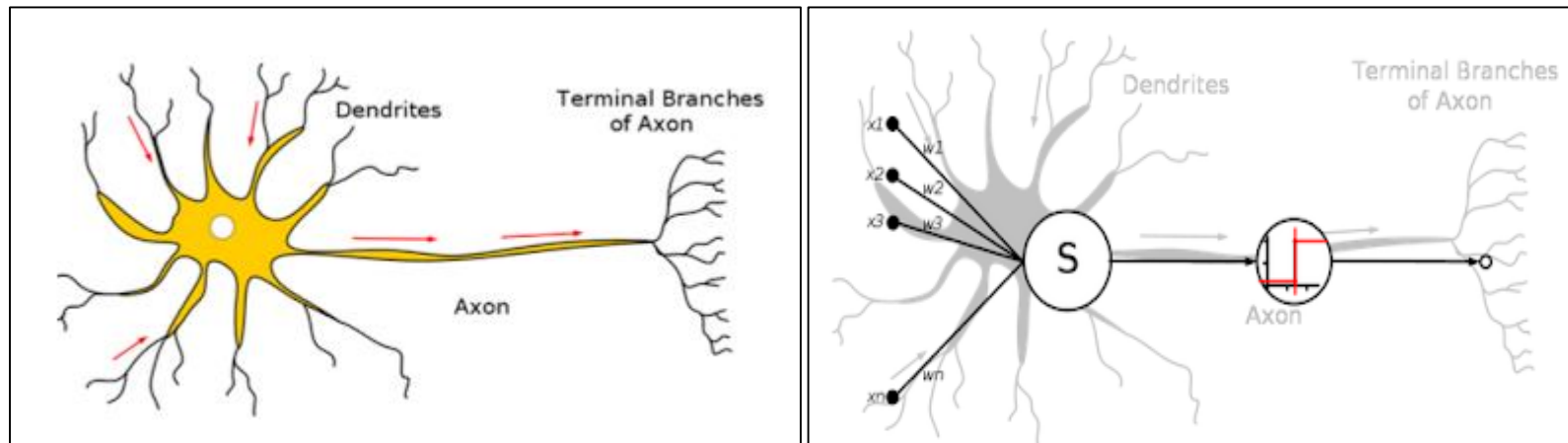
## 뉴런의 구조

- 성인의 뇌는 **850억 개의 뉴런**과  **$10^{14} \sim 10^{15}$ 개의 시냅스**로 이루어짐



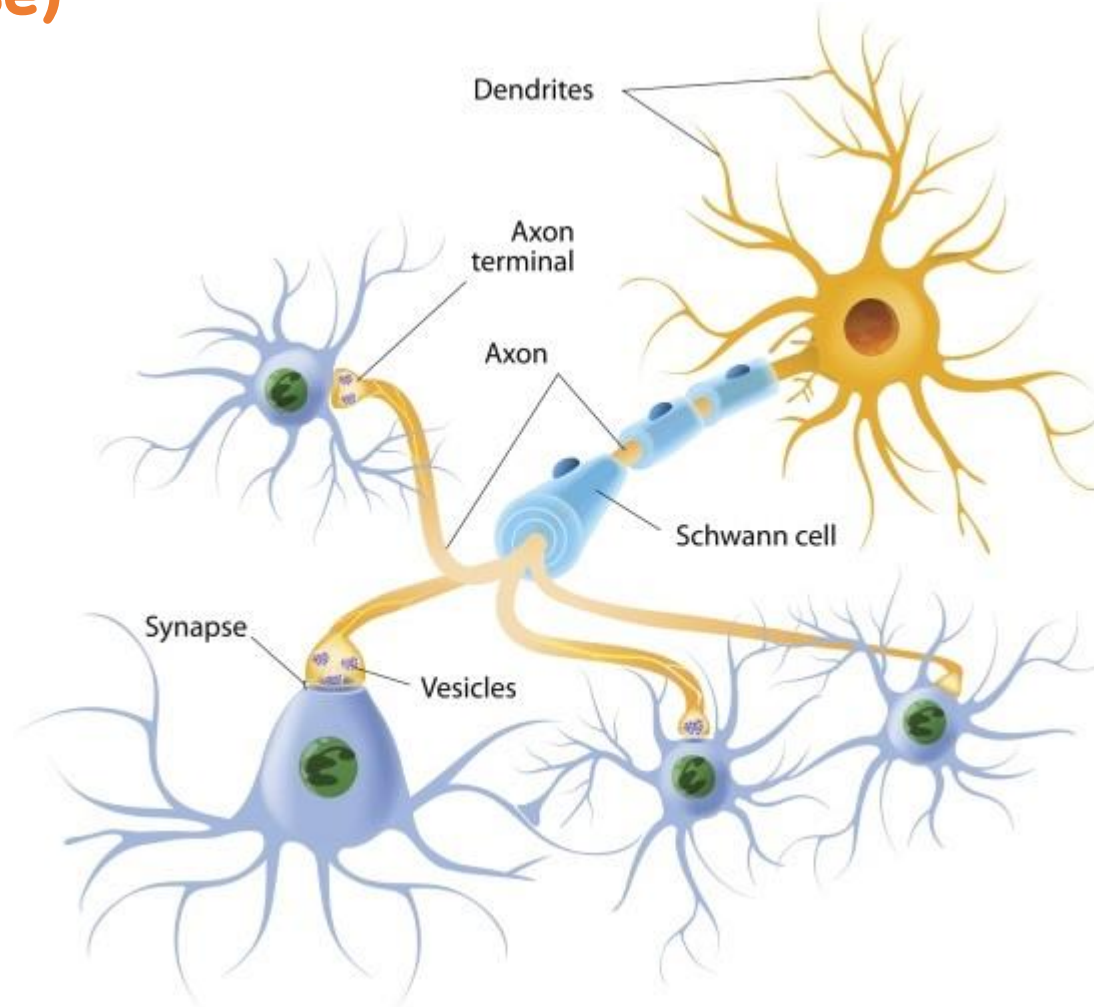
## 퍼셉트론 (Perceptron)

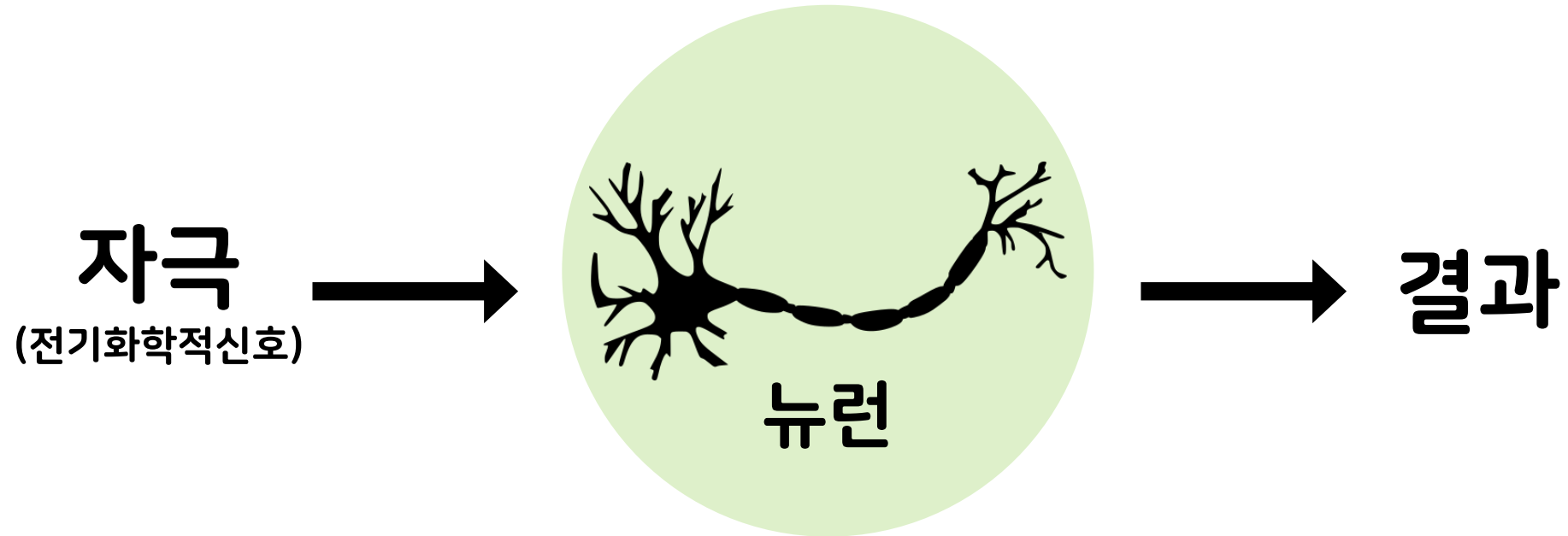
- 1957년 프랑크 로젠블라트에 의해 고안
- 신경세포는 **수상돌기(dendrite)**를 통해 다른 신경세포로부터 입력 신호를 받음 → **신경 세포체(cell body)**에서 정보 연산을 처리 → **축삭(axon)**을 통해 처리된 정보를 다른 신경세포로 전달
- 신경세포는 **시냅스**라는 가중 연결자(weighted connector)로 여러 층의 신경망을 구성
- 신경세포체는 입력 신호들의 합을 구함 → **임계치가 되면 축삭이 활성화되어 스파크**를 일으킴 → 다른 신경세포로 전기 신호를 전달
- 이 특성을 모방한 것이 **인공 신경망(ANN : Artificial Neural Network)**



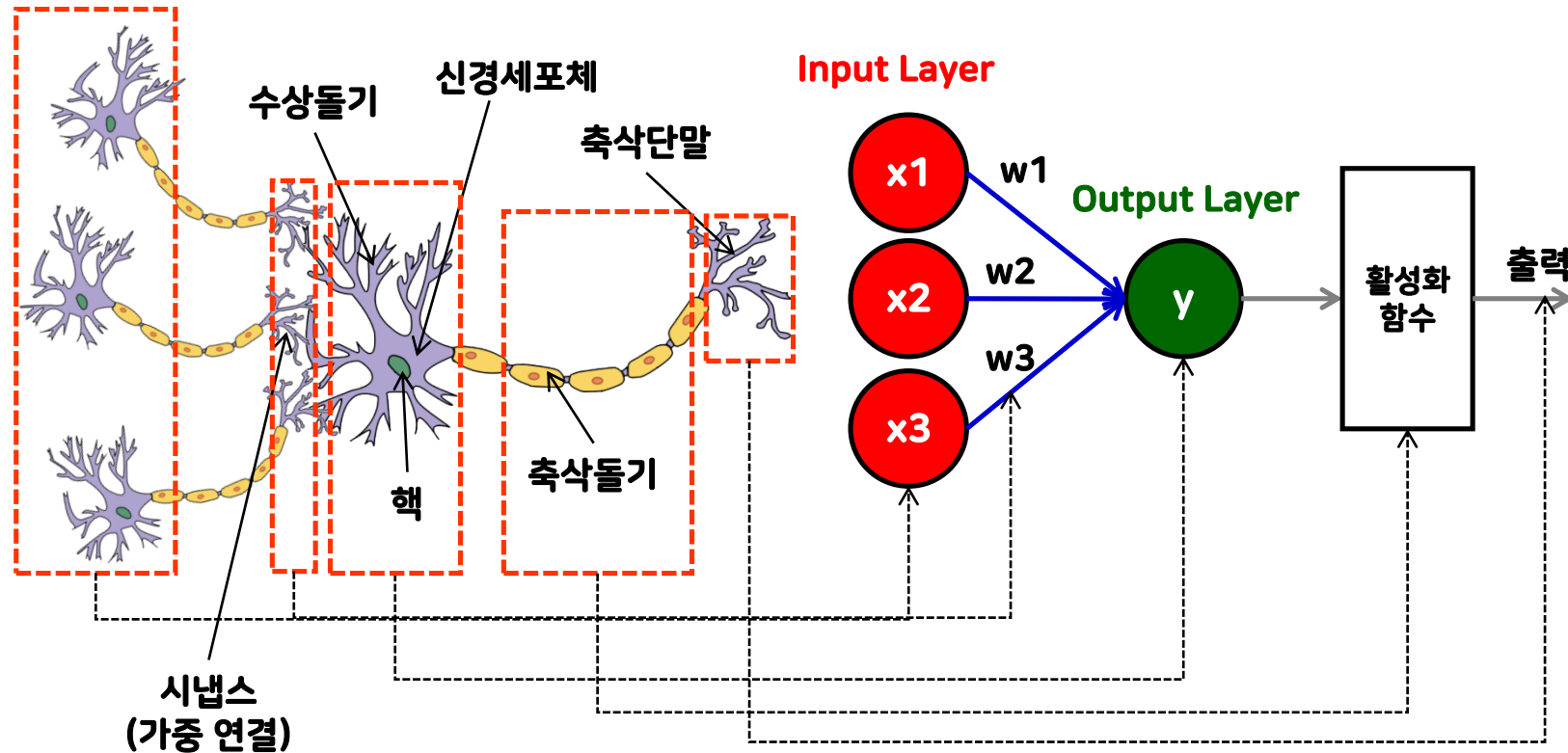


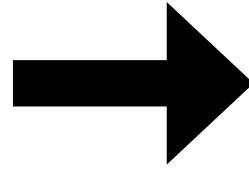
## 시냅스 (Synapse)





- 인간의 뉴런을 하나를 **노드** (인공 뉴런)으로 가상화하고 각 **노드의 특성** (가중치)를 **다르게 설정**하여 동일한 입력에 대해 다양한 반응을 발생하도록 하게 함.



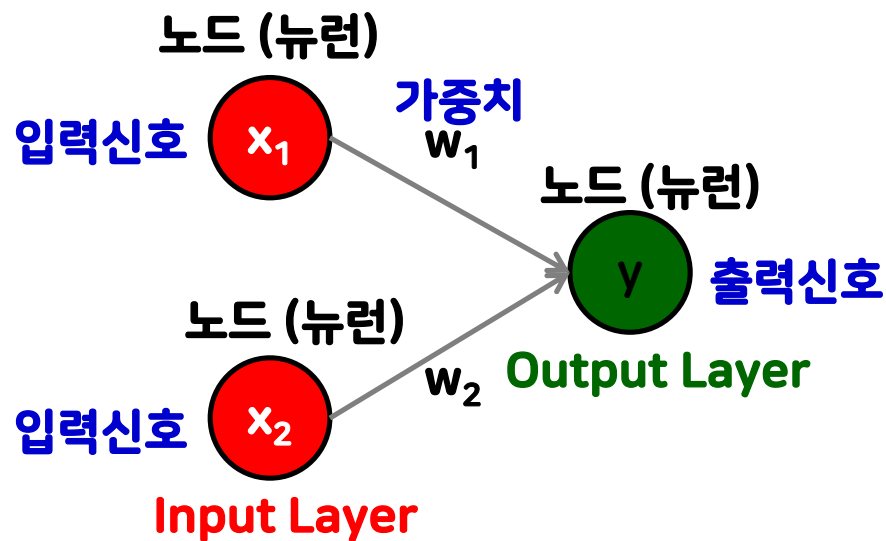


$$y = wx + b$$

선형모델

## 퍼셉트론 (Perceptron)

- 퍼셉트론은 다수의 신호를 입력받아 하나의 신호를 출력



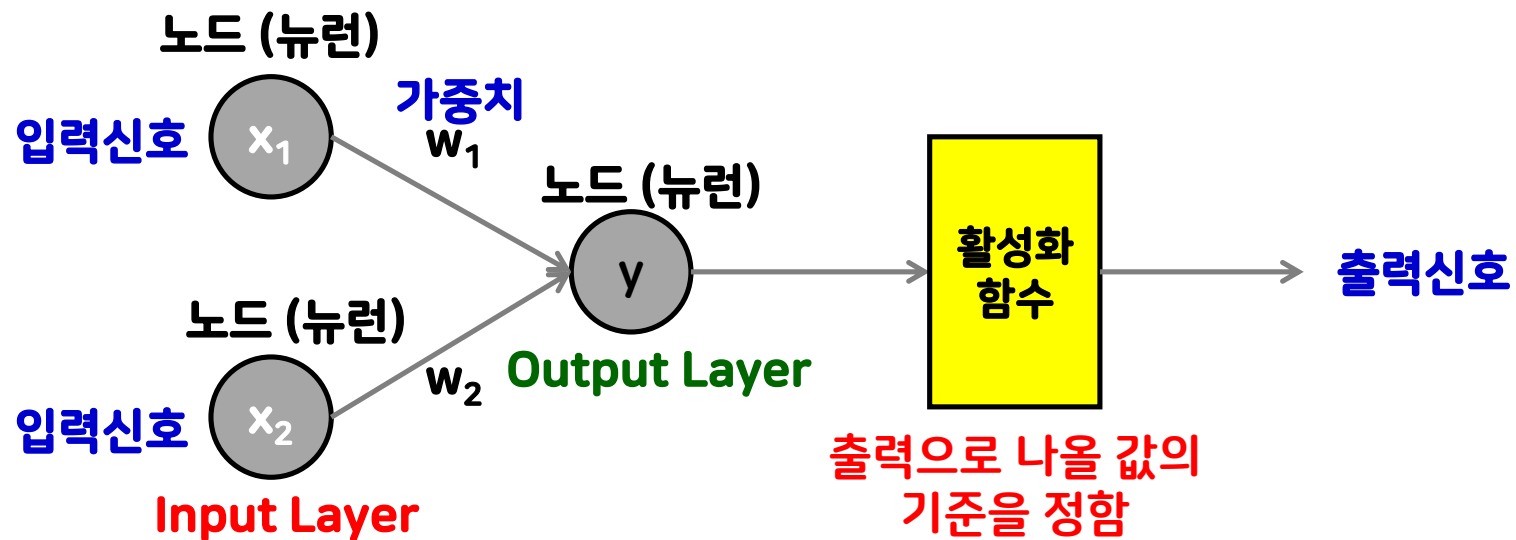
$$y = w_1 x_1 + w_2 x_2$$

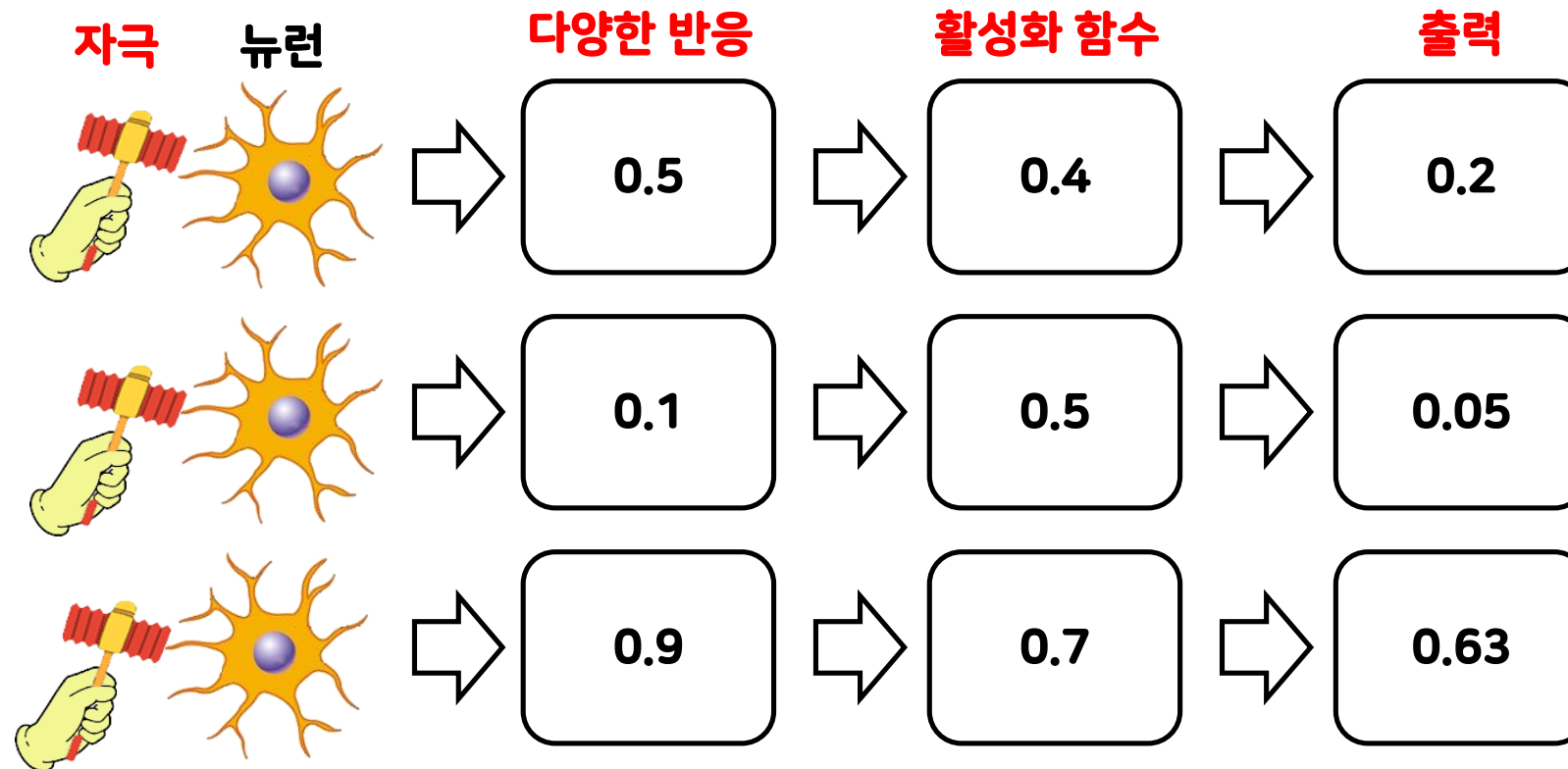
$$y = \sum_{i=0}^n w_i x_i$$

- 입력 신호가 뉴런에 보내질 때 각각의 고유한 가중치  $w_1, w_2$ 가 곱해지고 각 입력은 더해짐

## 퍼셉트론 (Perceptron)

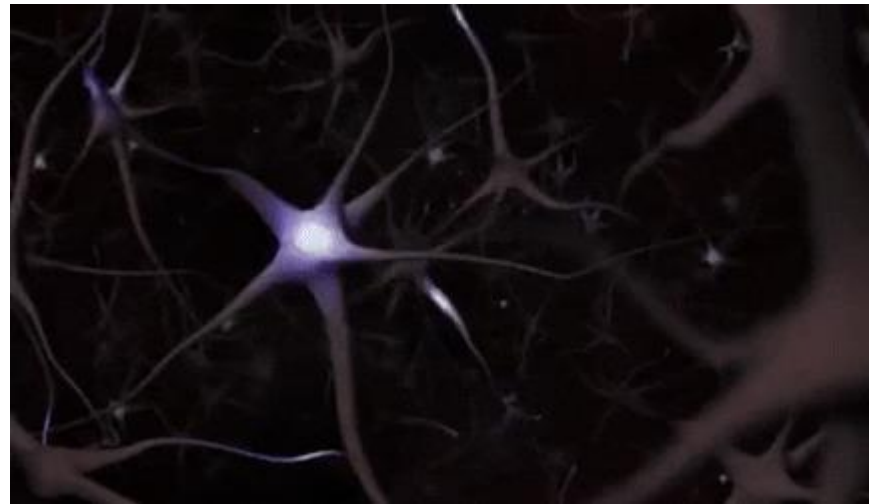
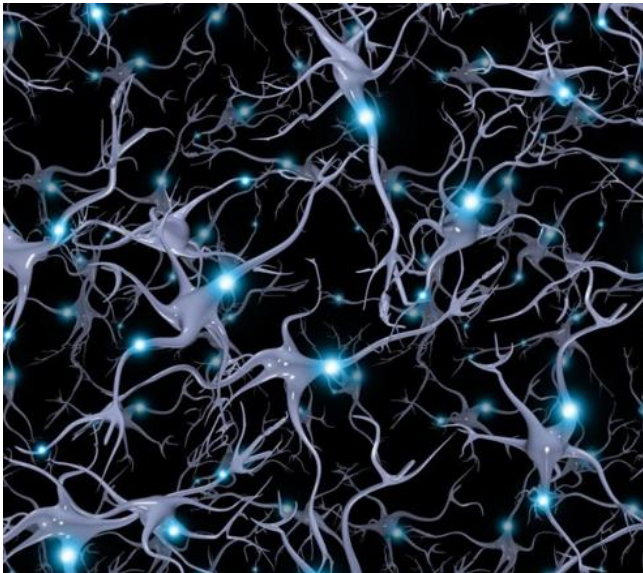
- **활성화 함수** : 출력을 다음 노드로 전달할 기준을 정하는 함수





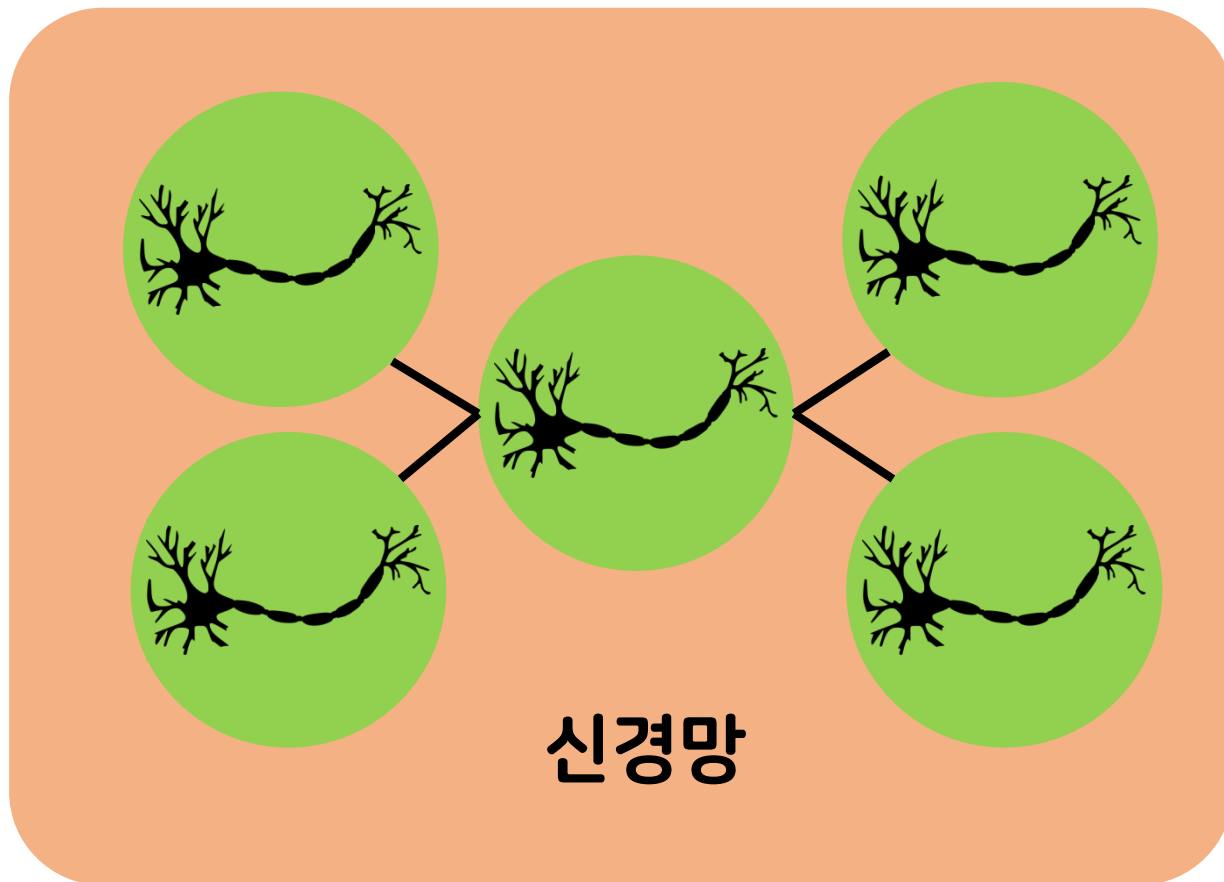
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- 신경망은 뇌의 신경세포 (뉴런)들이 시냅스로 연결되어 전기 신호를 통해 정보를 주고 받는 모습에서 착안한 것으로 **다층 퍼셉트론** (Multi-layer Perceptron)으로 부름





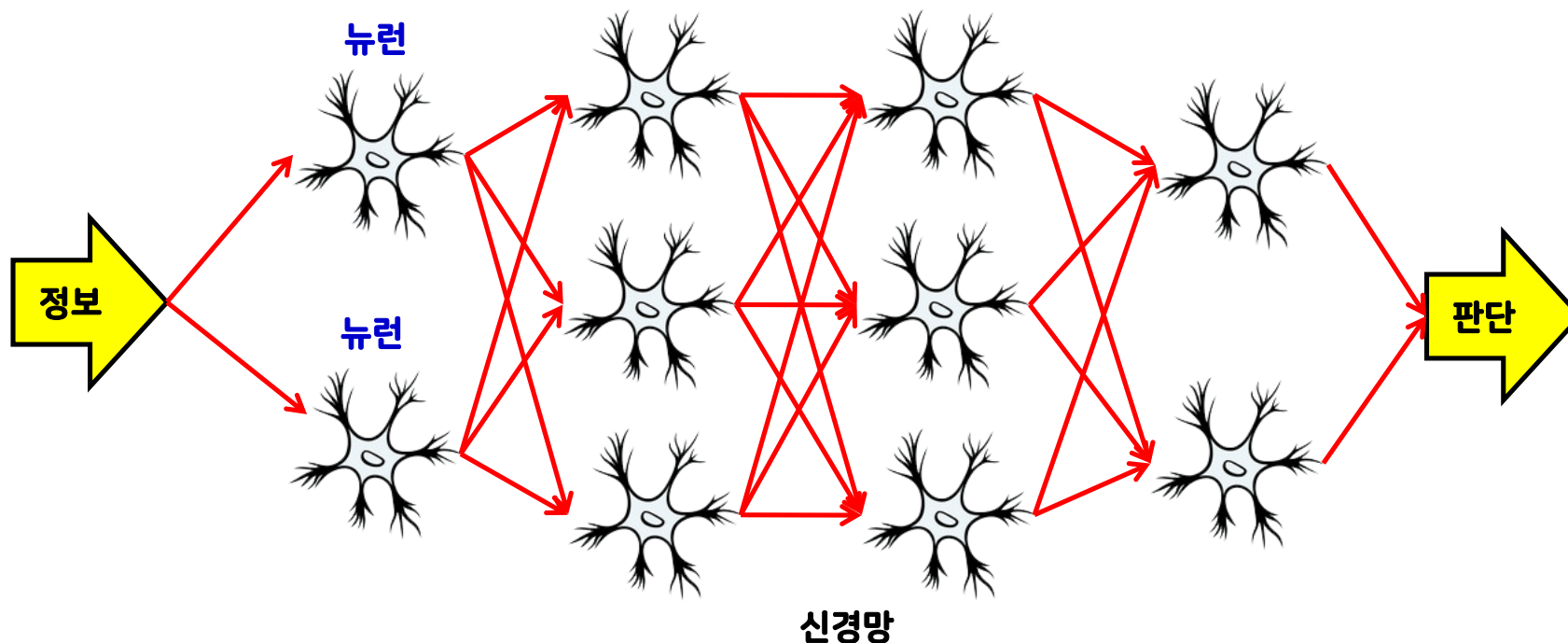
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)



많은 뉴런의 결과를  
종합하여 판단을 한다.

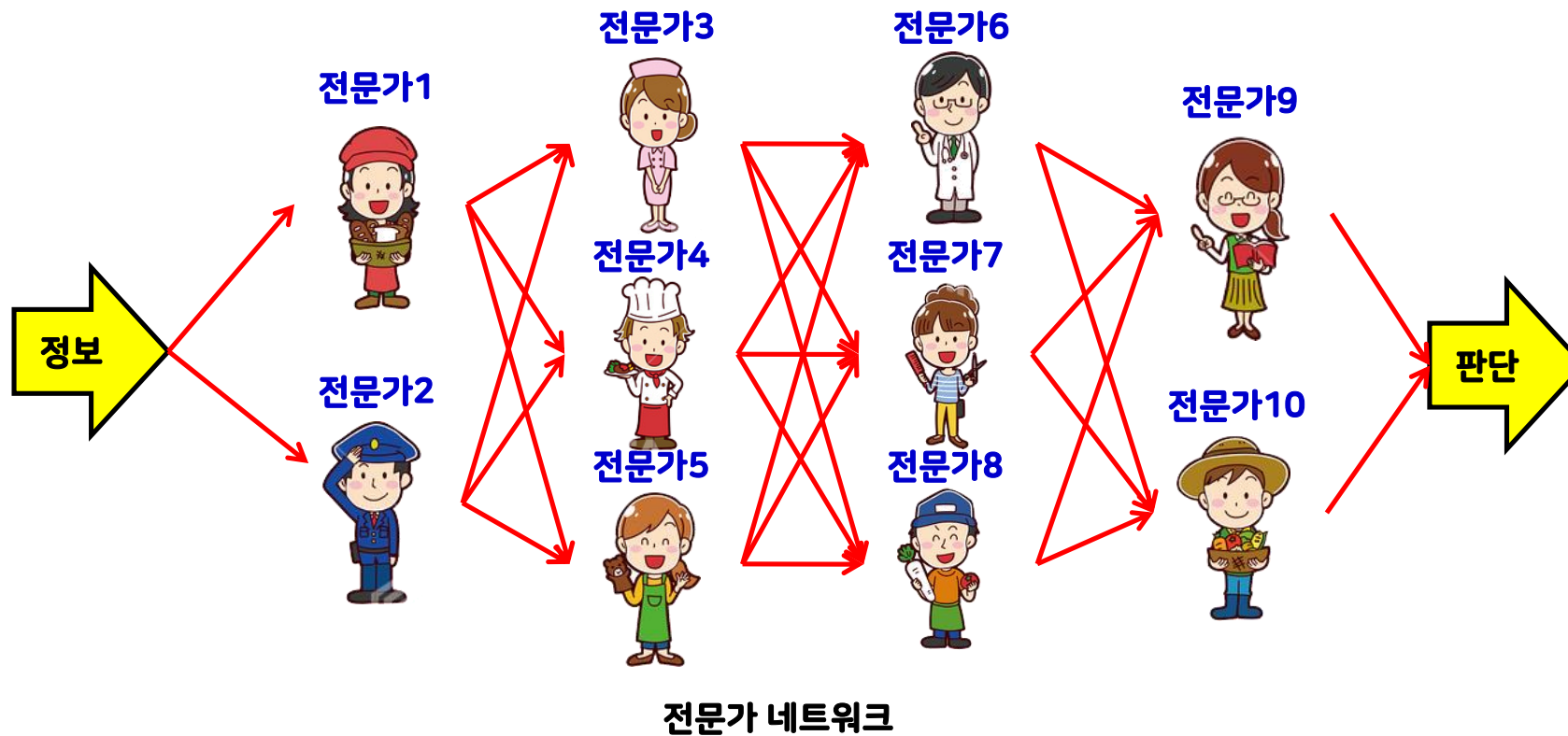
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- 사람은 대상이 무엇인지 **판단하는 정보들의 경계를 느슨하게** 가지고 있음 → 정확하지 않음 → 추상적
- 각 신경 세포 (뉴런)은 **정보에 대한 각기 다른 기준**을 가지고 있음
- 다층 신경망을 거치면서 정보를 판단하게 됨



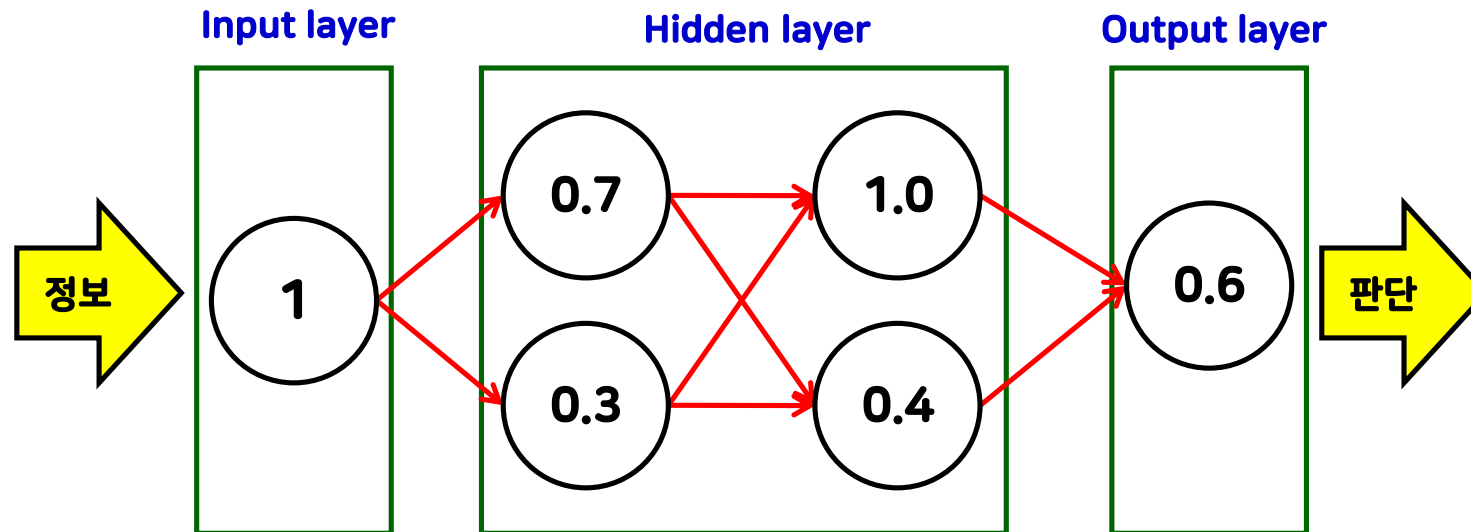
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- 다른 관점을 가진 전문가를 활용한 판단



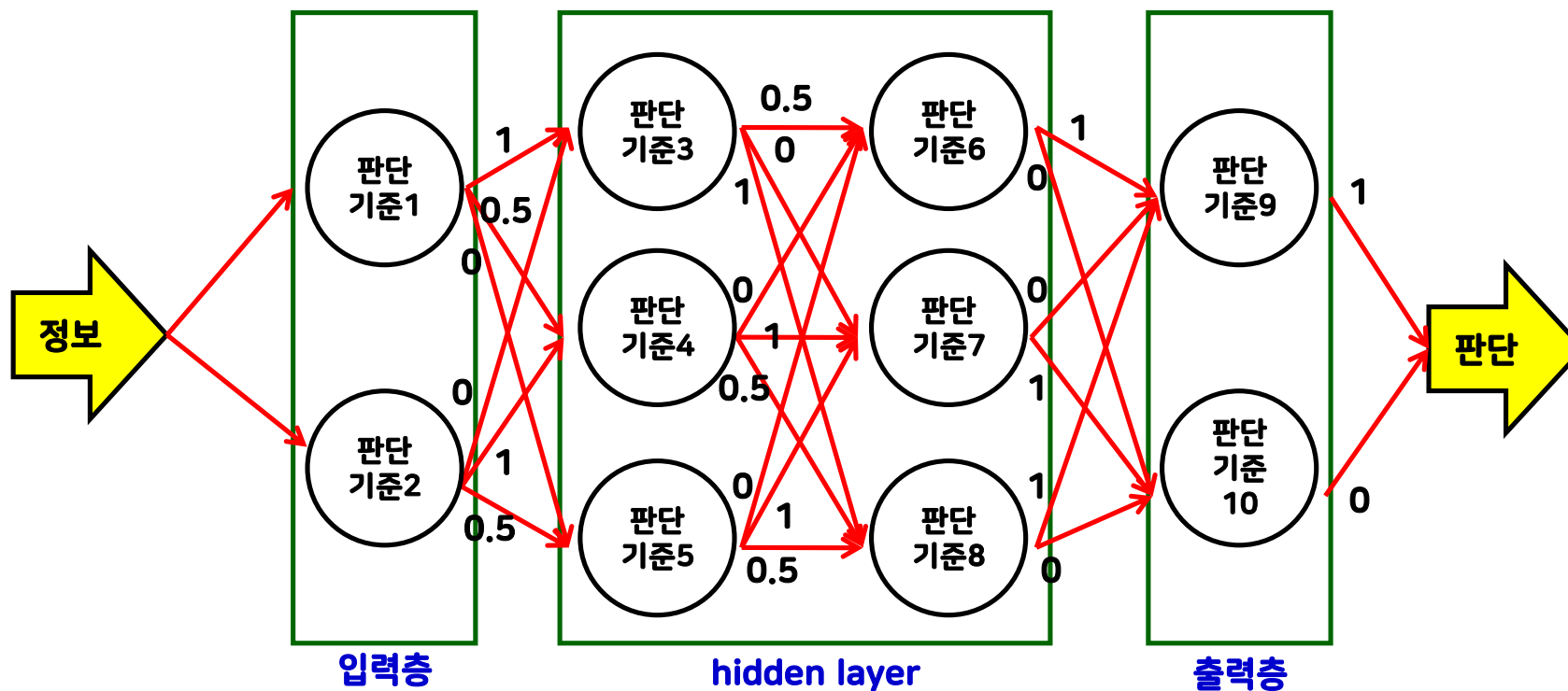
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- 하나의 입력에 대해 뉴런들은 다른 판단을 하고 다음 층으로 전달



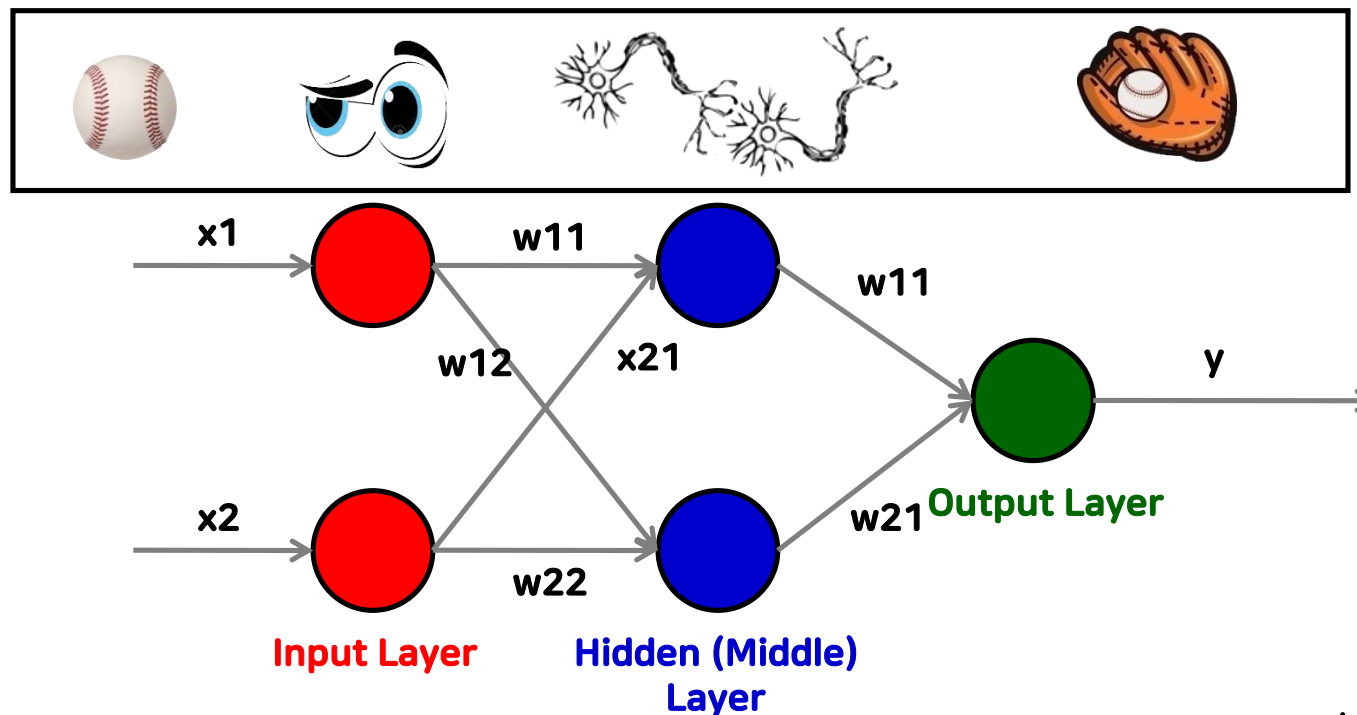
## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- hidden layer가 많을수록 (deep) 성능 향상 → **Deep Learning**
- 오차 증가 가능성 높아짐 → **오차감소 알고리즘 필요**


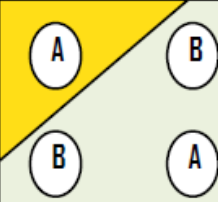
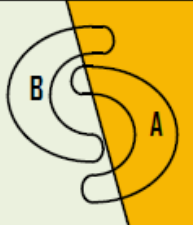
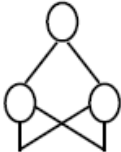
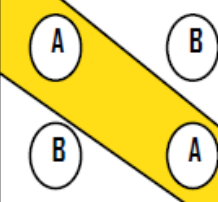
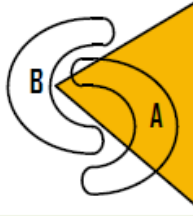
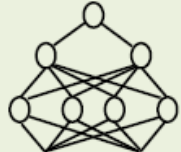
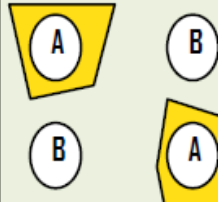



## 다층 퍼셉트론 (MLP : Multi Layer Perceptron)

- 비선형 데이터를 분리할 수 있다.
- 학습 시간이 오래 걸리고 파라미터 수가 많으므로 과대적합을 일으키기 쉬우며 가중치가 초기값에 민감하여 지역 최적점에 빠지기 쉽다



## 다층 퍼셉트론 (MLP : Multi Layer Perceptron) 용도

구조	결정 영역 형태	Exclusive-OR 문제	얹힌 결정 영역을 갖는 클래스들
<p>단층</p> 	<p>초평면에 의해 나뉘어지는 반평면 (Hyper plane)</p>		
<p>두 개층</p> 	<p>볼록한 모양 또는 닫힌 영역</p>		
<p>세 개층</p> 	<p>임의의 형태 (노드들의 개수에 따라 복잡도가 결정됨)</p>		

## 다층 퍼셉트론 (MLP : Multi Layer Perceptron) 용도

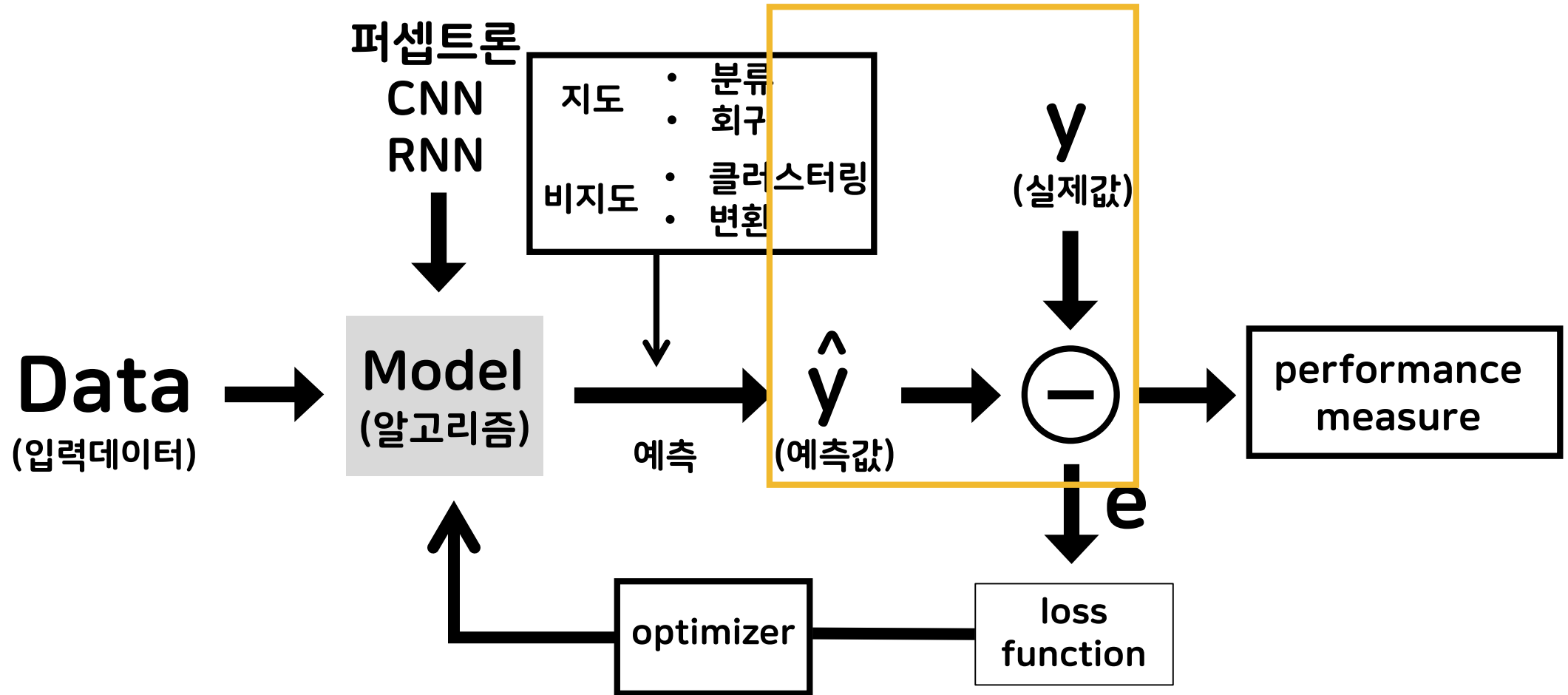
- (1) 음성인식, 영상인식, 자연어 처리 (챗봇), 가상실험 등 (일정하게 정해진 값이 존재하지 않는 데이터 → 같은 사람의 목소리라도 환경에 따라 달라짐)
- (2) 긴 학습 시간이 필요한 데이터
- (3) 학습 예제에 어려가 존재하는 경우
- (4) 여러 속성에 의해 표현되는 데이터 등



## 다층 퍼셉트론 (MLP : Multi Layer Perceptron) 동작



# 기존 머신러닝(선형모델)과 딥러닝의 공통점



## Rule-based expert system

**Data**

(입력데이터)



사람이  
직접 작성한 규칙



결과

## 기존 머신러닝

**Data**

(입력데이터)



사람이  
선택한 특성



SVM, KNN



결과

딥러닝 : feature engineering이 거의 필요 없다. (사람의 개입 최소화)

**Data**

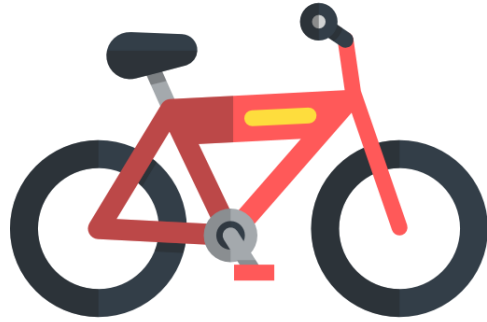
(입력데이터)



신경망(딥러닝)



결과



자전거



트럭

?

무엇을 타고갈까?



집 앞 편의점

## 딥러닝(Deep Learning)

컴퓨터비전, 음성인식, 자연어처리, 신호처리 등의 분야에 적용



모든 문제를 딥러닝으로 해결하지 않는다.  
기존 머신러닝 모델이 잘 동작하는 경우도 있다.

## (1) 비지도 학습을 이용한 전처리

- 군집 (Clustering)을 이용한 전처리로 잡음 제거하여 인공신경망 최적화 가능 (제프리 힌튼, 2006)

## (2) CNN의 진화

- 예측에 의해 뽑히던 feature들을 기계학습을 이용해 뽑는다는 것

## (3) RNN

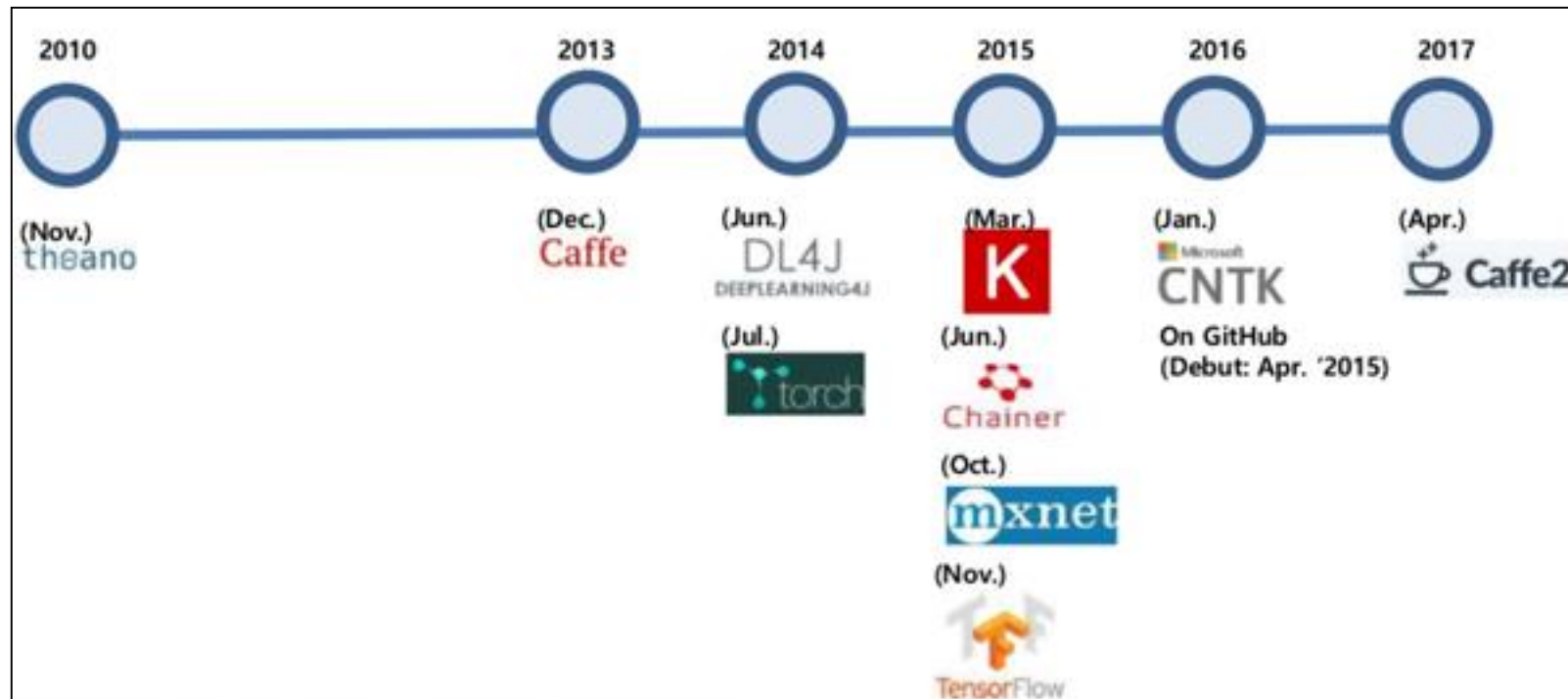
- 시계열 데이터(Time-series data)에서 탁월한 성능을 보이는 RNN → 가장 깊은 층을 가짐
- LSTM을 게이트 유닛마다 배치하여 Vanishing Gradient 문제 해결

## (4) GPU 병렬 컴퓨팅의 등장과 학습 방법의 진화

- GPGPU(General-Purpose computing on Graphics Processing Units)의 개념 개발 (병렬처리)
- 비선형 변환에 쓰이는 Rectified Linear Unit(ReLU)의 개발
- 거대 망을 선택적으로 학습하는 드롭아웃(Dropout)의 발견

## 1950년 전부터 있었던 신경망 개념이 최근에 갑자기 뜬 것일까요 ?







Tensorflow

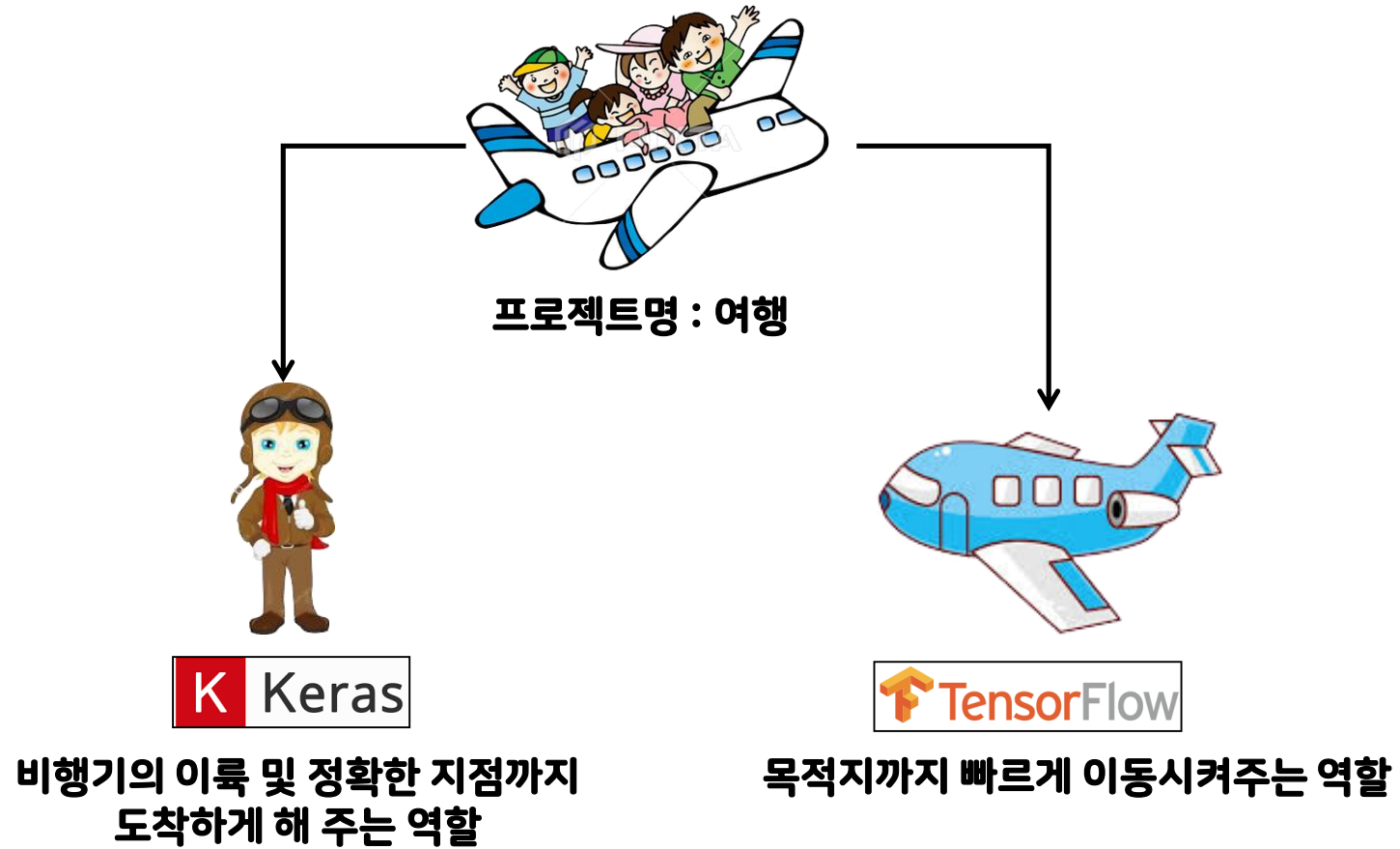


- 구글이 2015년 오픈 소스로 공개한 프레임워크
- 구글 거의 모든 제품에 적용, 다양한 언어 지원, 폭넓은 사용자층 → 가장 널리 사용
- 알파고의 AI 연산 가속기 TPU에도 적용
- 주요 개발 언어 : Python
- 단점 : GPU 버전은 리눅스만 지원, 다소 느림

## Keras



- 파이썬으로 작성된 오픈소스 신경망 프레임워크
- 비전문가도 쉽게 사용할 수 있음
- 실제로 Keras에서는 다양한 뉴럴 네트워크 모델을 미리 지원해주고 있으므로, 그냥 블록을 조립하듯이 네트워크를 만들면 되는 식이라, 전반적인 네트워크 구조를 생각하고 작성한다면 빠른 시간내에 코딩을 할 수 있는 엄청난 장점
- 현재는 tensorflow 위에서 keras가 동작하도록 설계되어 있고, 하위 모듈로 구현되어있음



Pytorch

PYTORCH

- 페이스북 인공지능 연구팀이 만든 파이썬 기반 오픈소스 머신러닝 프레임워크
- 파이토치는 절차가 간단하고 그래프가 동적으로 변화할 수 있음
- 코드 자체도 파이썬과 유사해 진입 장벽이 낮은 편
- 텐서플로우에 비해 사용자층이 얇고 관련 자료를 구하기 어려움

Caffe

Caffe



Caffe2

- 2013년 버클리 인공지능 연구소에서 개발된 프레임워크
  - 컴퓨터 비전 머신러닝에 특화, C/C++ 기반으로 사용
  - CUDA와 딥러닝 라이브러리 cuDNN을 활용해 성능 최적화 가능
  - Caffe2로 되면서 파이토치로 흡수
- 
- 컴퓨터 비전 분야 이외의 분야에서 타 프레임워크에 비해 성능이 떨어짐
  - 확장성이 낮고, 업데이트 느림, 분산 환경 지원 부족

## DL4J

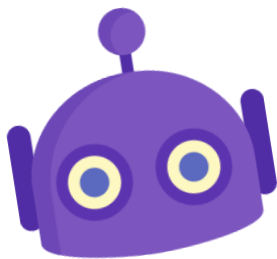


- Java와 JVM용으로 만들어진 딥러닝 프레임워크
- 빅데이터 분산 저장 및 분석에 특화된 아파치 하둡과 스파크와 연동 가능
- CUDA 커널이 포함
- 사용차 층이 적음
- 언어 자체가 딥러닝에 최적화되어 있는 않음

MXNet

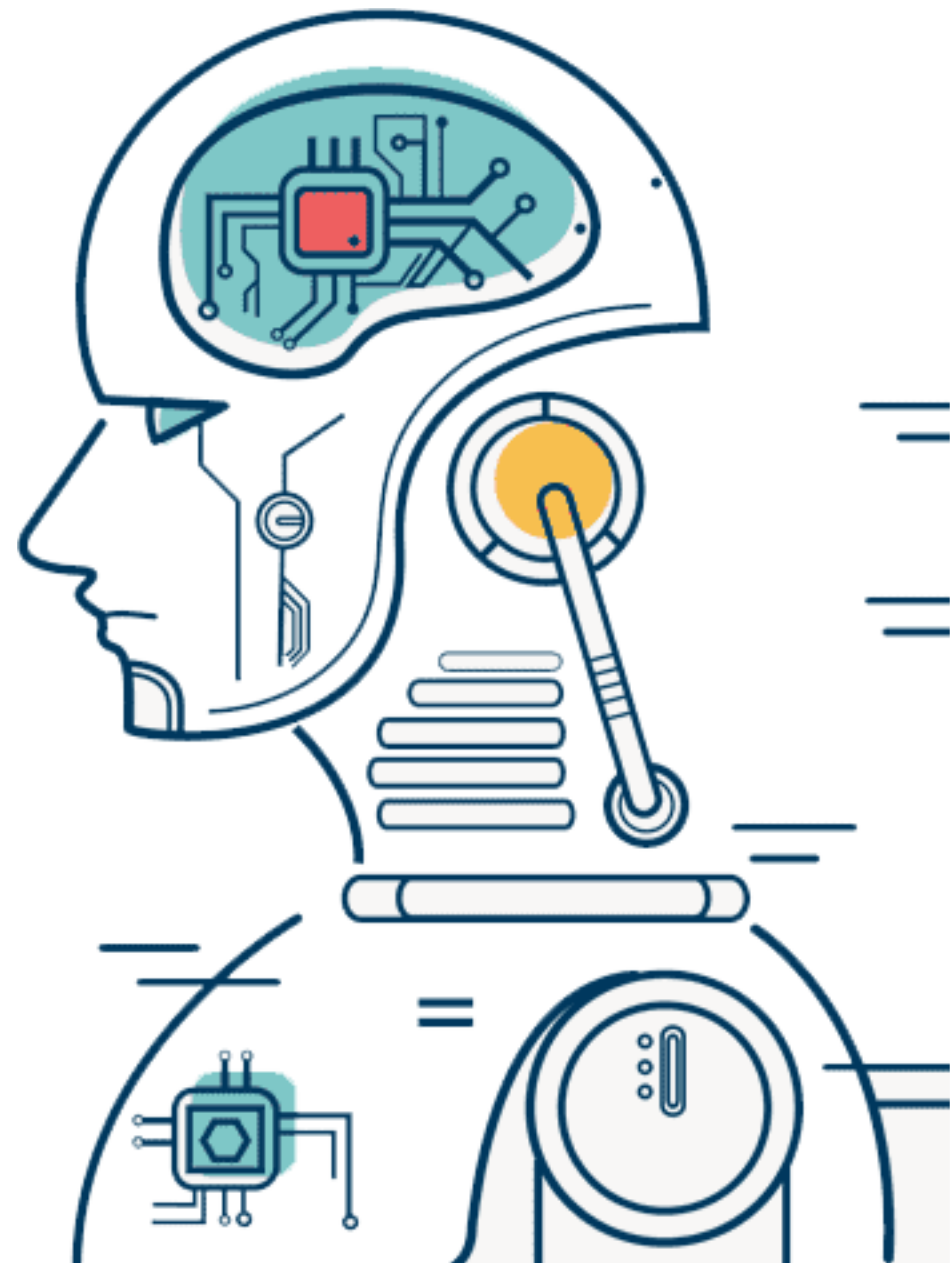


- 아마존이 공식 지원하는 오픈소스 프레임워크
- 폭넓은 언어 지원 (파이썬, C++, 스칼라, 줄리아, 매트랩, JS, Perl, R 등)
- 모바일 기기부터 서버 수준까지의 디바이스 사용 가능
- 속도가 느림, 로우 레벨 텐서 연산자가 적음



# 딥러닝 기초 실습

(환경설정, 딥러닝 따라하기)



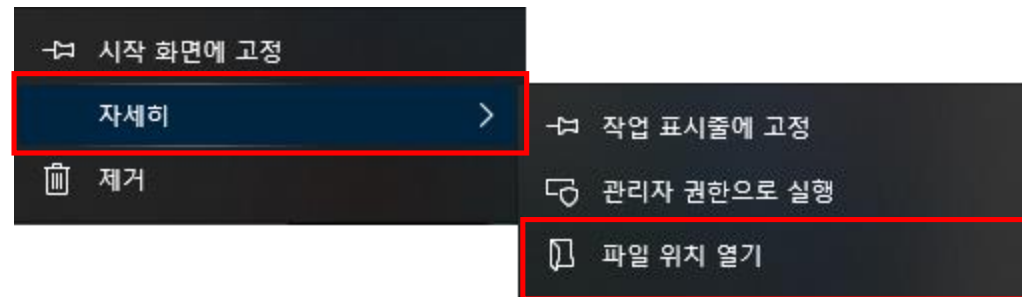
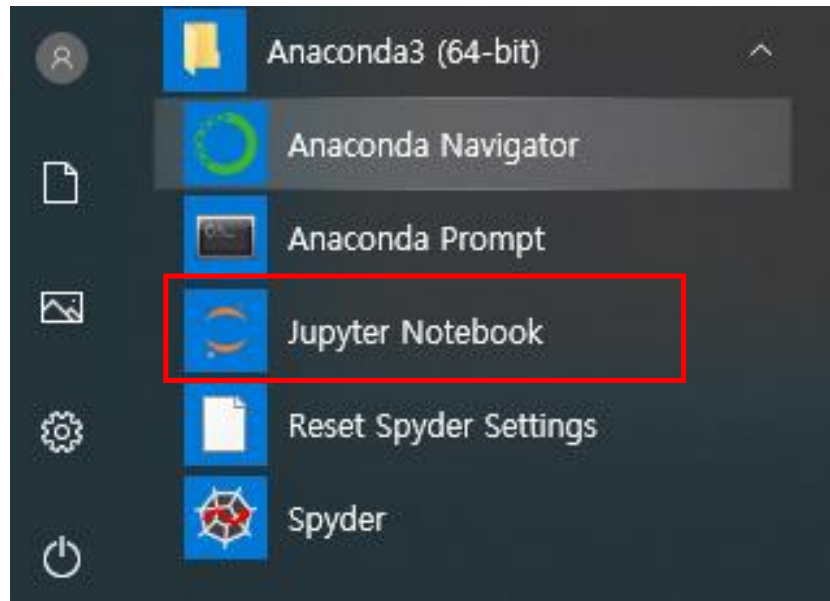


- <https://www.anaconda.com/products/individual#download-section>에서 Window 버전을 다운로드하여 설치







## Anaconda Installers

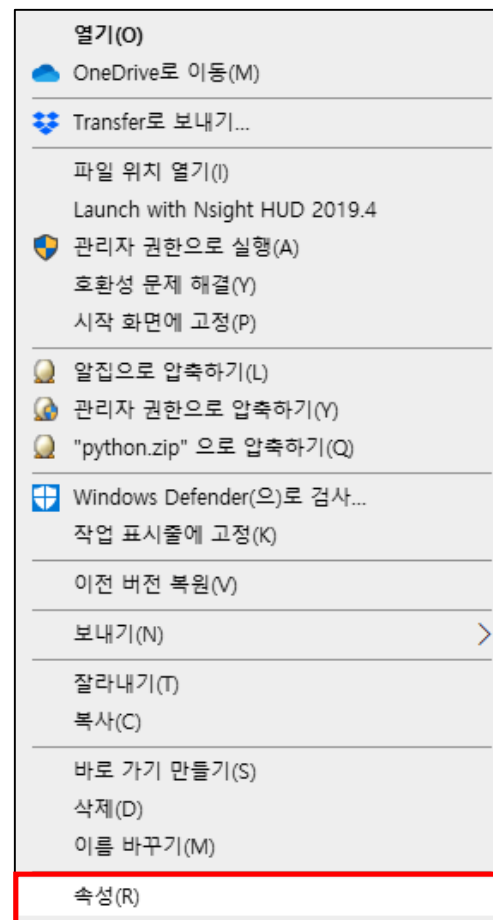
Windows 	MacOS 	Linux 
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

- 윈도우 아이콘을 클릭하고 Jupyter Notebook에서 오른쪽 마우스 버튼을 클릭
- 자세히 > 파일 위치 열기를 클릭

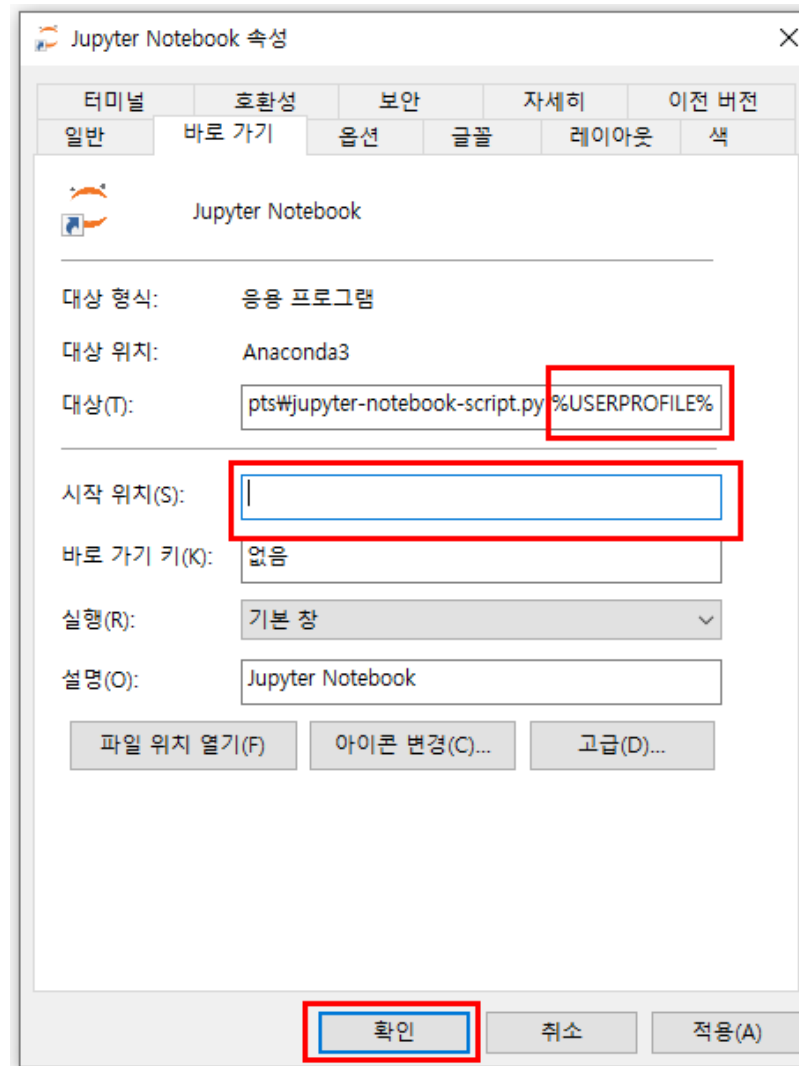


- Jupyter Notebook (anaconda3)에서 오른쪽 마우스 버튼을 클릭하고 **속성**을 선택

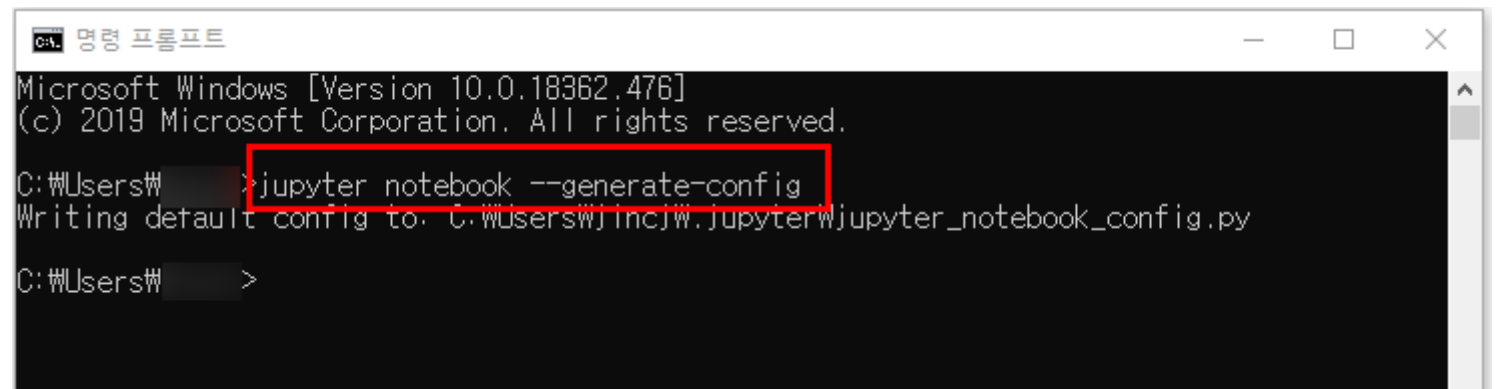
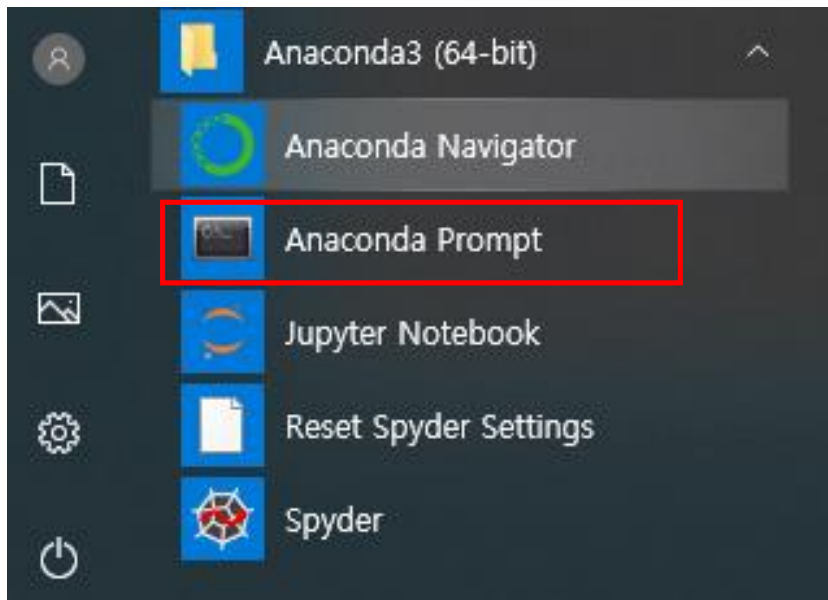
이름	수정한 날짜	유형
 Anaconda Navigator (anaconda3)	2020-09-20 오후 10:24	바로 가기
 Anaconda Powershell Prompt (anaconda3)	2020-09-20 오후 10:24	바로 가기
 Anaconda Prompt (anaconda3)	2020-09-20 오후 10:24	바로 가기
 Jupyter Notebook (anaconda3)	2020-09-20 오후 10:25	바로 가기
 Reset Spyder Settings (anaconda3)	2020-09-20 오후 10:24	바로 가기
 Spyder (anaconda3)	2020-09-20 오후 10:24	바로 가기



- 대상 입력란에 있는 %USERPROFILE%을 삭제
- 시작 위치 란은 공란으로 하고 확인

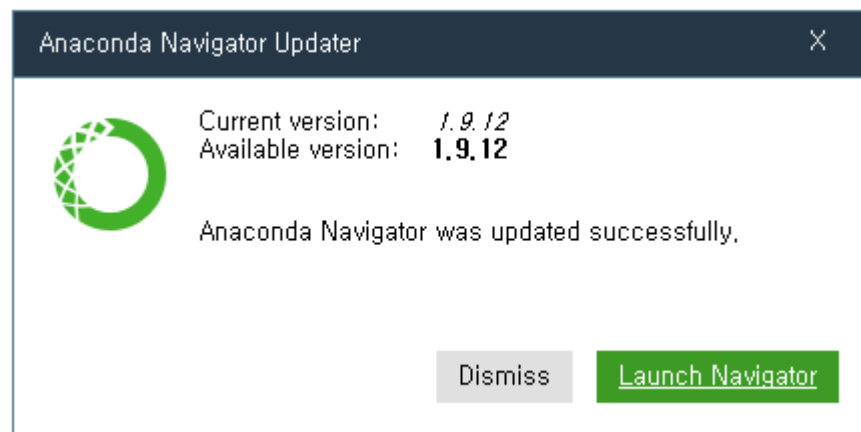
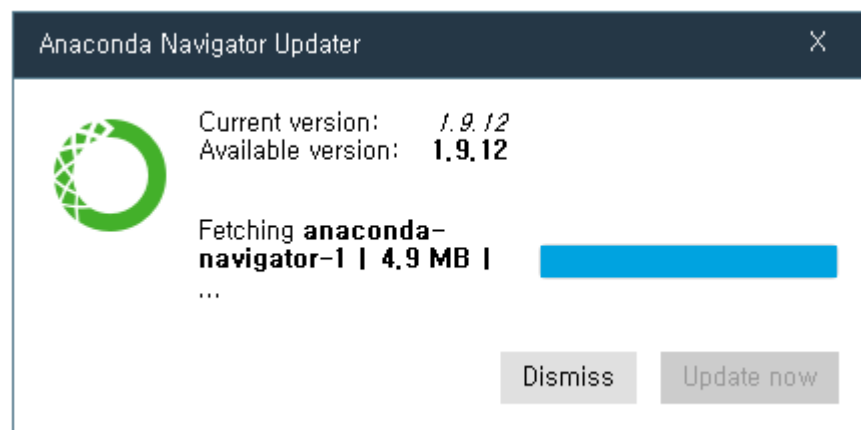
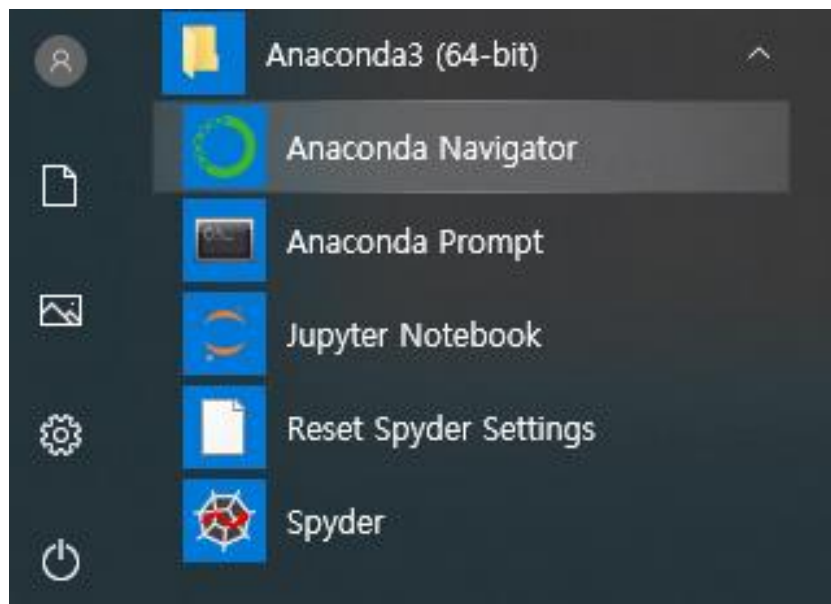


- 윈도우 아이콘을 클릭하고 Jupyter Notebook에서 오른쪽 마우스 버튼을 클릭
- Anaconda Prompt를 클릭
- `jupyter notebook --generate-config` 를 실행

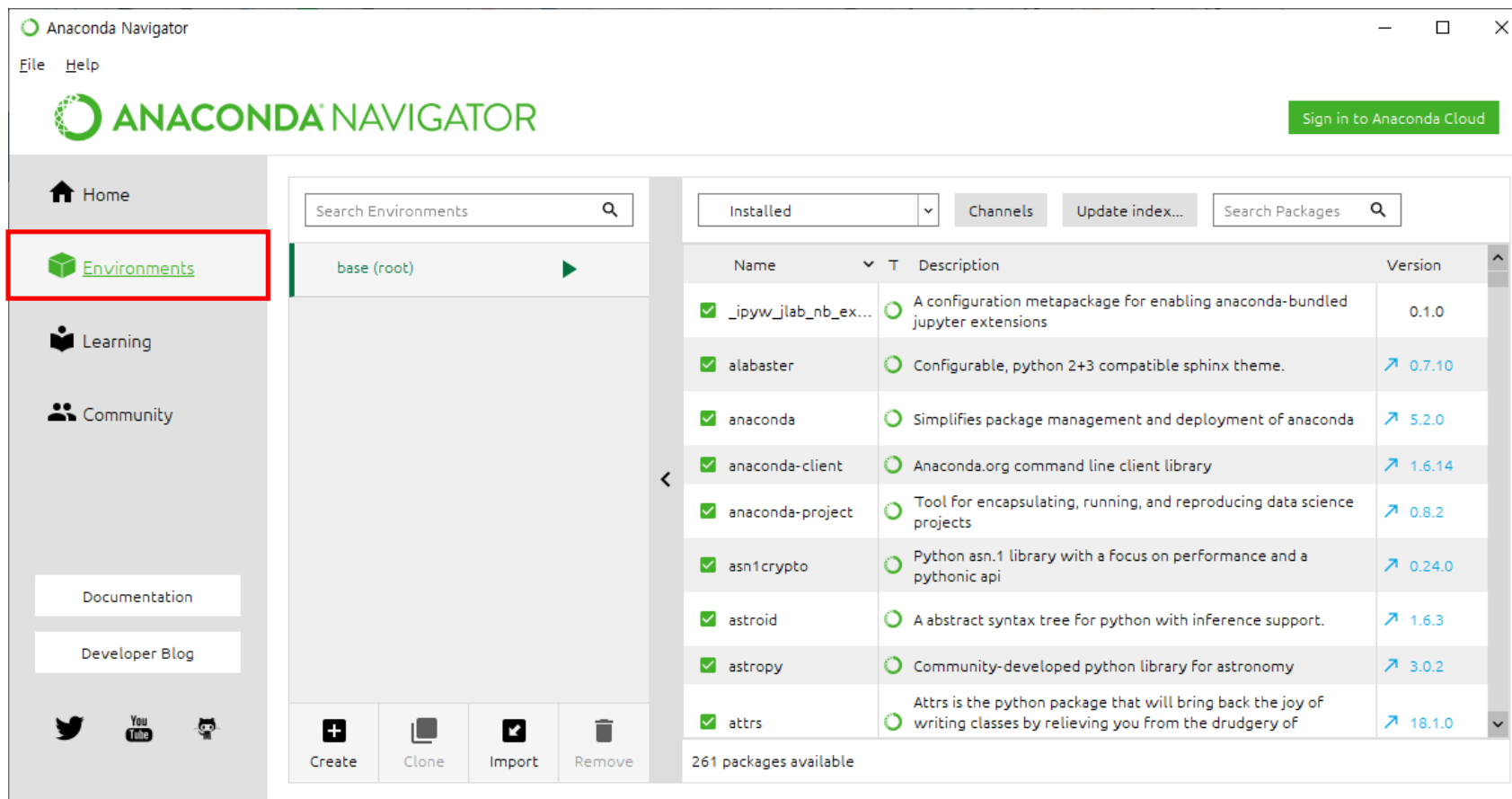


- 사용자 폴더 > .jupyter 폴더에 생성된 `jupyter_notebook_config.py`를 연다
- `#c.NotebookApp.notebook_dir` 를 검색
- `c.NotebookApp.notebook_dir = '시작 폴더의 경로'` 로 수정하고 저장
- Jupyter Notebook을 실행하여 해당 폴더에서 시작되는 것을 확인

- Anaconda Navigator를 실행 (Update 메시지가 뜨면 업데이트 후에 재실행)

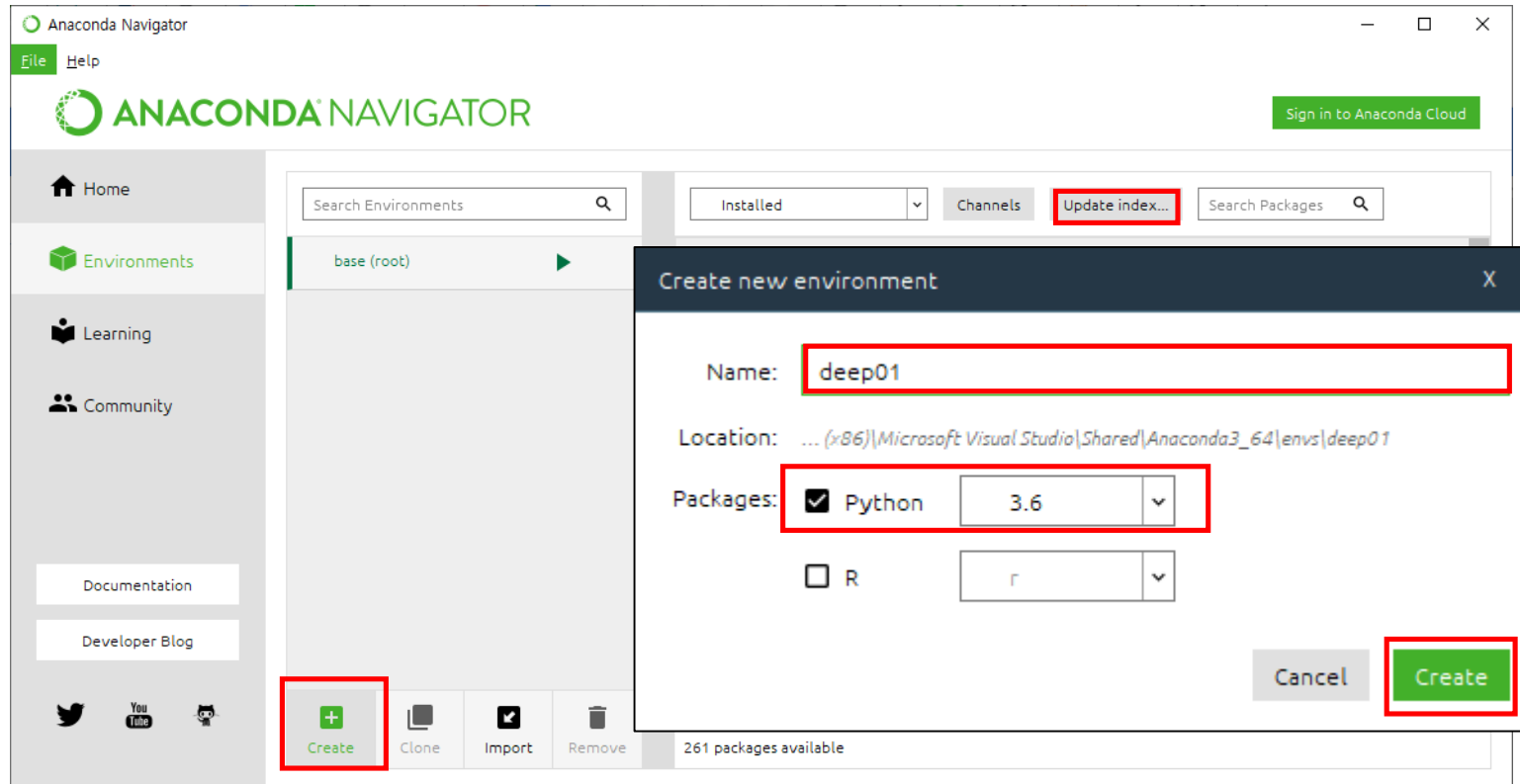


## - Environments 메뉴를 클릭

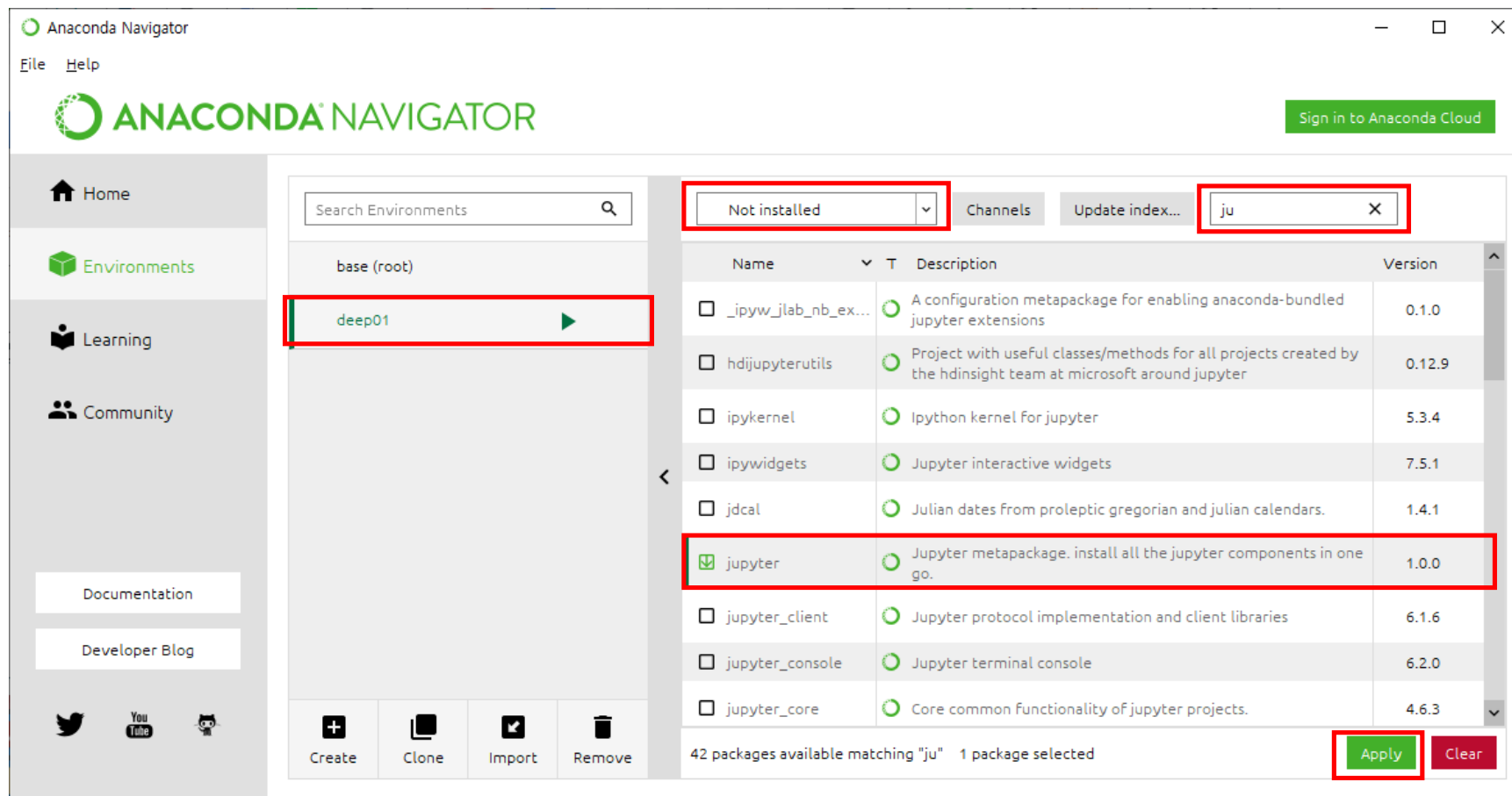




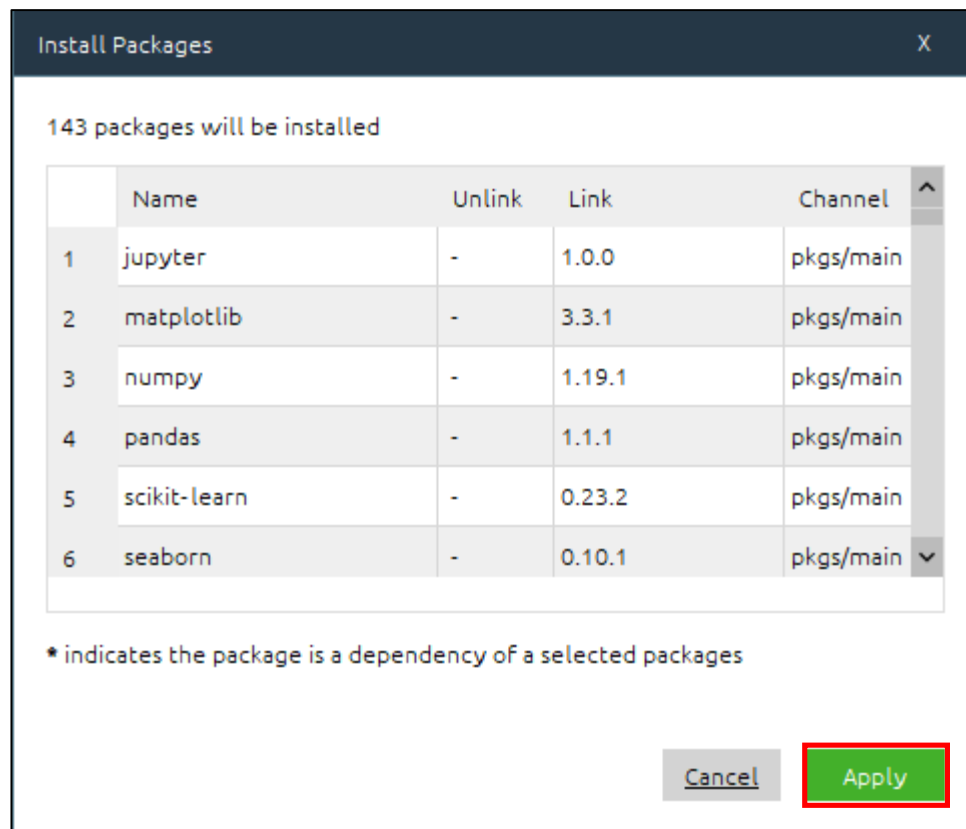
- Create 버튼을 클릭하고 새로운 환경 이름을 입력 (deep01)
- Packages를 Python으로 체크하고 버전을 3.6으로 선택 (없다면 Update index 버튼을 클릭하여 업데이트)



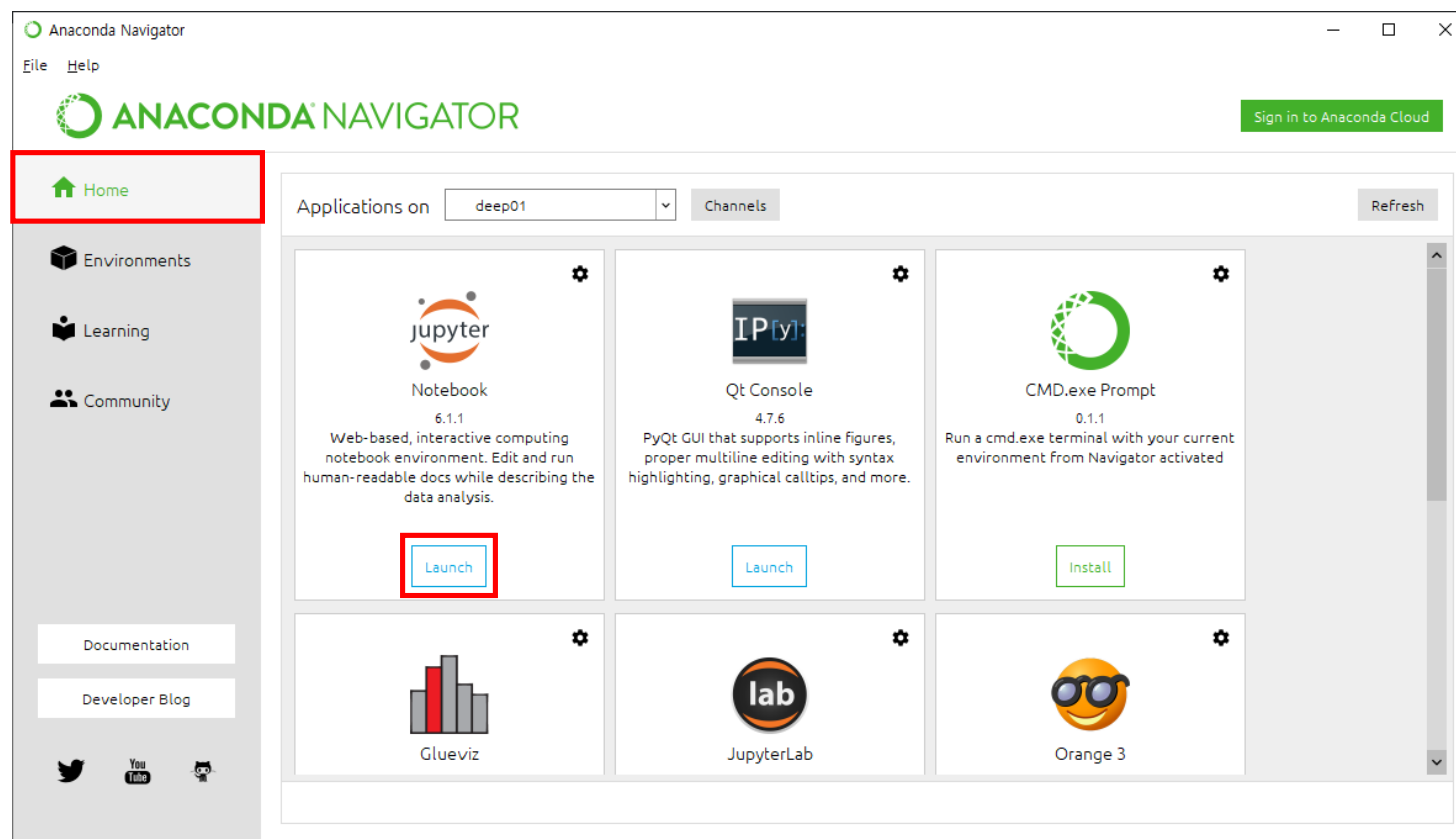
- 생성된 개발환경 이름을 선택하고 Not installed를 선택
- jupyter, tensorflow를 선택하고 Apply 버튼을 클릭



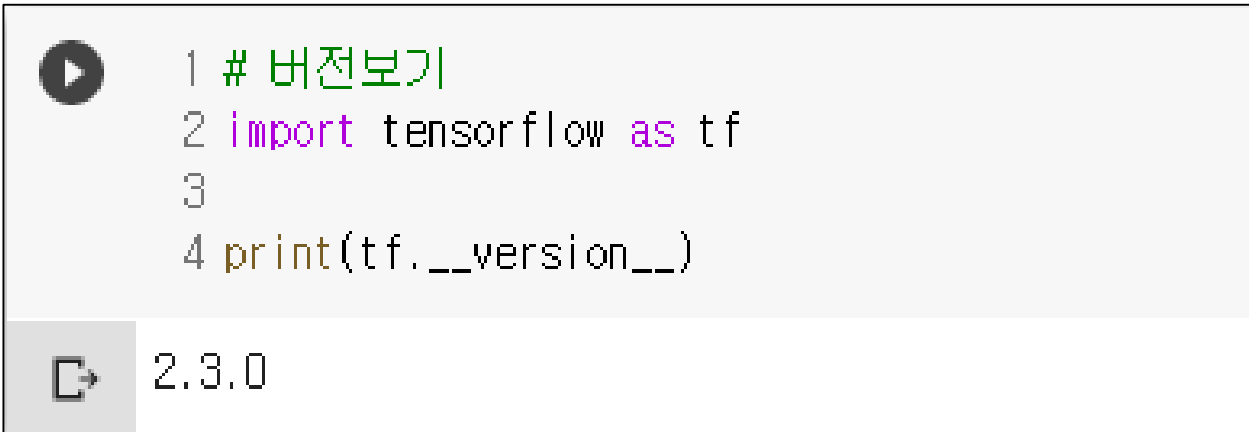
- 선택한 라이브러리를 확인하고 Apply 버튼을 클릭하여 설치



## - Home 메뉴를 클릭하고 Jupyter Notebook을 실행



- 새 파일을 만들어서 tensorflow 버전 확인 (Jupyter Notebook)



```
1 # 버전보기
2 import tensorflow as tf
3
4 print(tf.__version__)
```

2.3.0

(1) Google 회원 가입

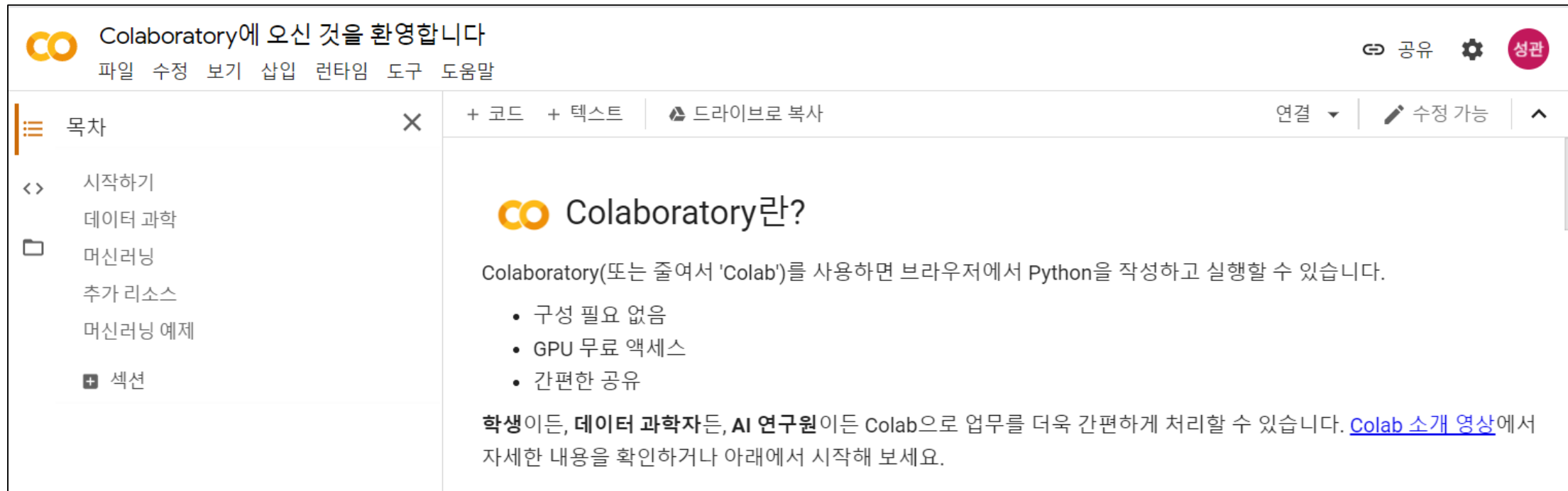
(2) 코랩 접속

<https://colab.research.google.com/>

(3) 구글 드라이브 접속

<https://drive.google.com/drive/my-drive>

## Colab 접속 - 로그인



The screenshot shows the Colaboratory web interface. At the top, there's a header with the Colab logo, a welcome message "Colaboratory에 오신 것을 환영합니다", and navigation links: "파일", "수정", "보기", "삽입", "런타임", "도구", "도움말". On the right of the header are icons for "공유" (Share), "설정" (Settings), and "성관" (Profile). Below the header is a sidebar on the left with a "목차" (Table of Contents) section containing links like "시작하기" (Get started), "데이터 과학" (Data science), "머신러닝" (Machine learning), "추가 리소스" (Additional resources), and "머신러닝 예제" (Machine learning examples). The main content area has a title "Colaboratory란?" (What is Colaboratory?) and a paragraph explaining that Colaboratory (or 'Colab') allows running Python in a browser. It lists three benefits: "구성 필요 없음" (No setup required), "GPU 무료 액세스" (Free GPU access), and "간편한 공유" (Easy sharing). At the bottom, it encourages students, data scientists, and AI researchers to use Colab for their work, with a link to "Colab 소개 영상" (Colab intro video).

Colaboratory에 오신 것을 환영합니다  
파일 수정 보기 삽입 런타임 도구 도움말

공유 설정 성관

목차

- 시작하기
- 데이터 과학
- 머신러닝
- 추가 리소스
- 머신러닝 예제
- 섹션

+ 코드 + 텍스트 | 드라이브로 복사

연결 | 수정 가능

### Colaboratory란?

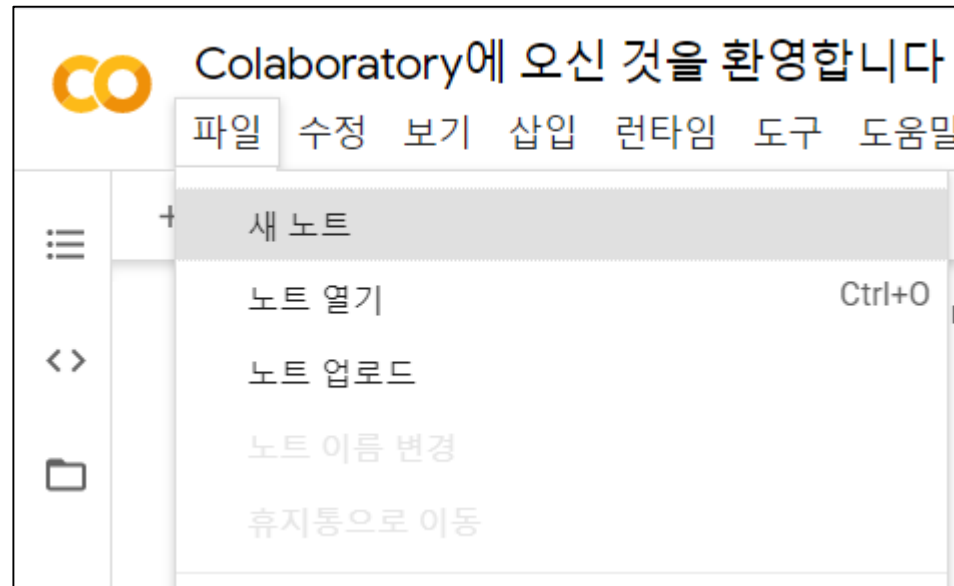
Colaboratory(또는 줄여서 'Colab')를 사용하면 브라우저에서 Python을 작성하고 실행할 수 있습니다.

- 구성 필요 없음
- GPU 무료 액세스
- 간편한 공유

학생이든, 데이터 과학자든, AI 연구원이든 Colab으로 업무를 더욱 간편하게 처리할 수 있습니다. [Colab 소개 영상](#)에서 자세한 내용을 확인하거나 아래에서 시작해 보세요.

## Colab - 새 노트 생성

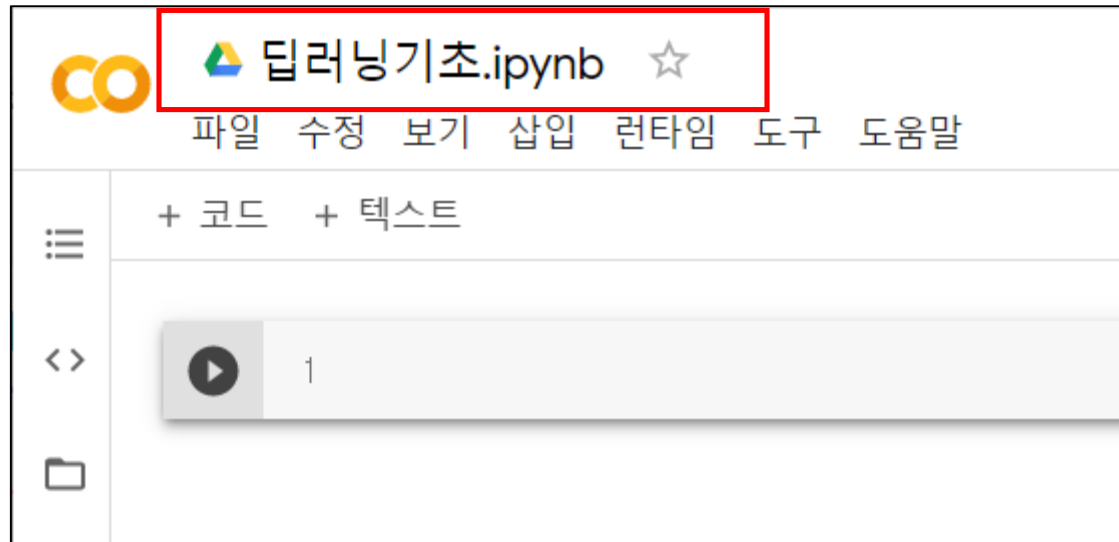
- 파일 > 새 노트 메뉴를 선택하여 새 파일 생성



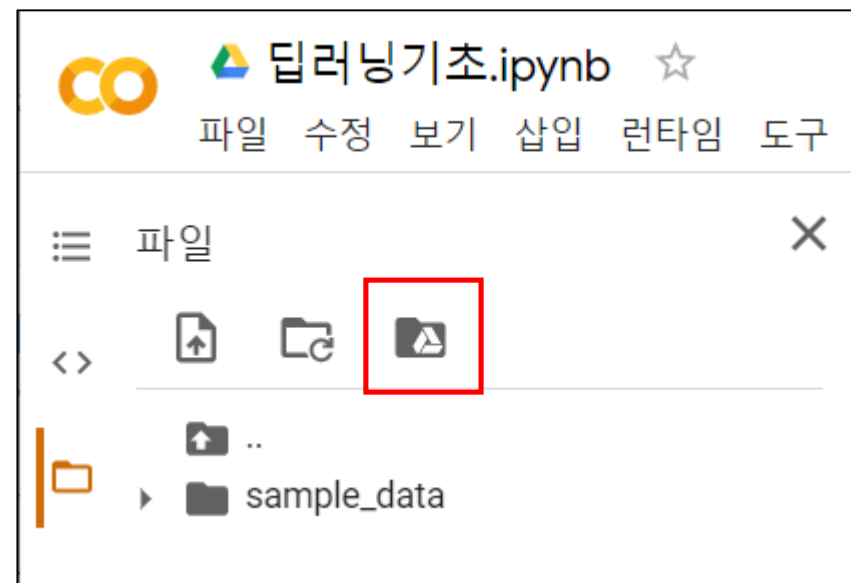
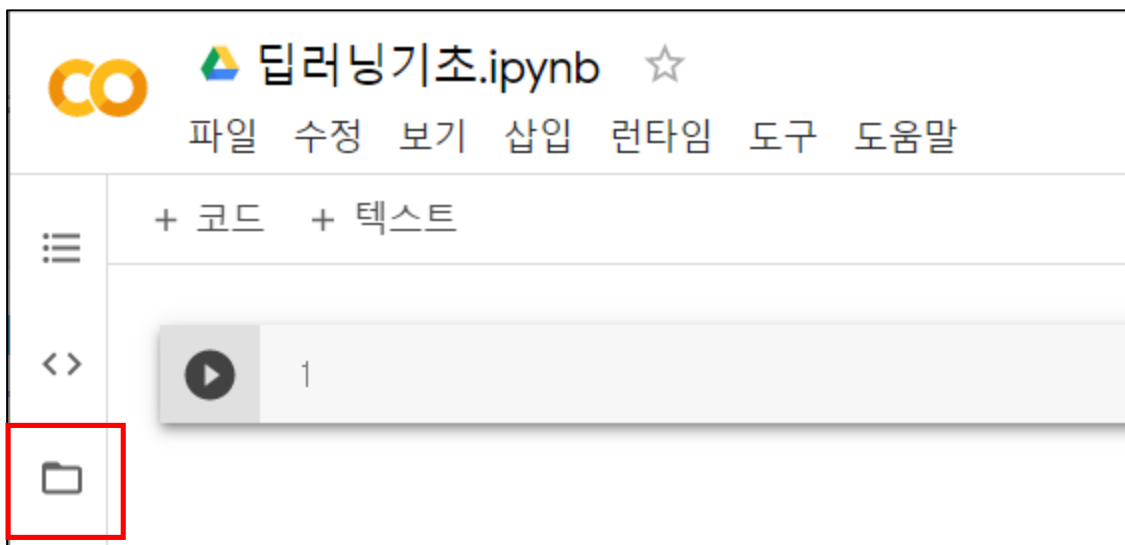


## Colab - 새 노트 생성

- 파일명을 클릭하여 이름을 수정



## Colab - 구글 드라이브 연동



## Colab - 구글 드라이브 연동

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

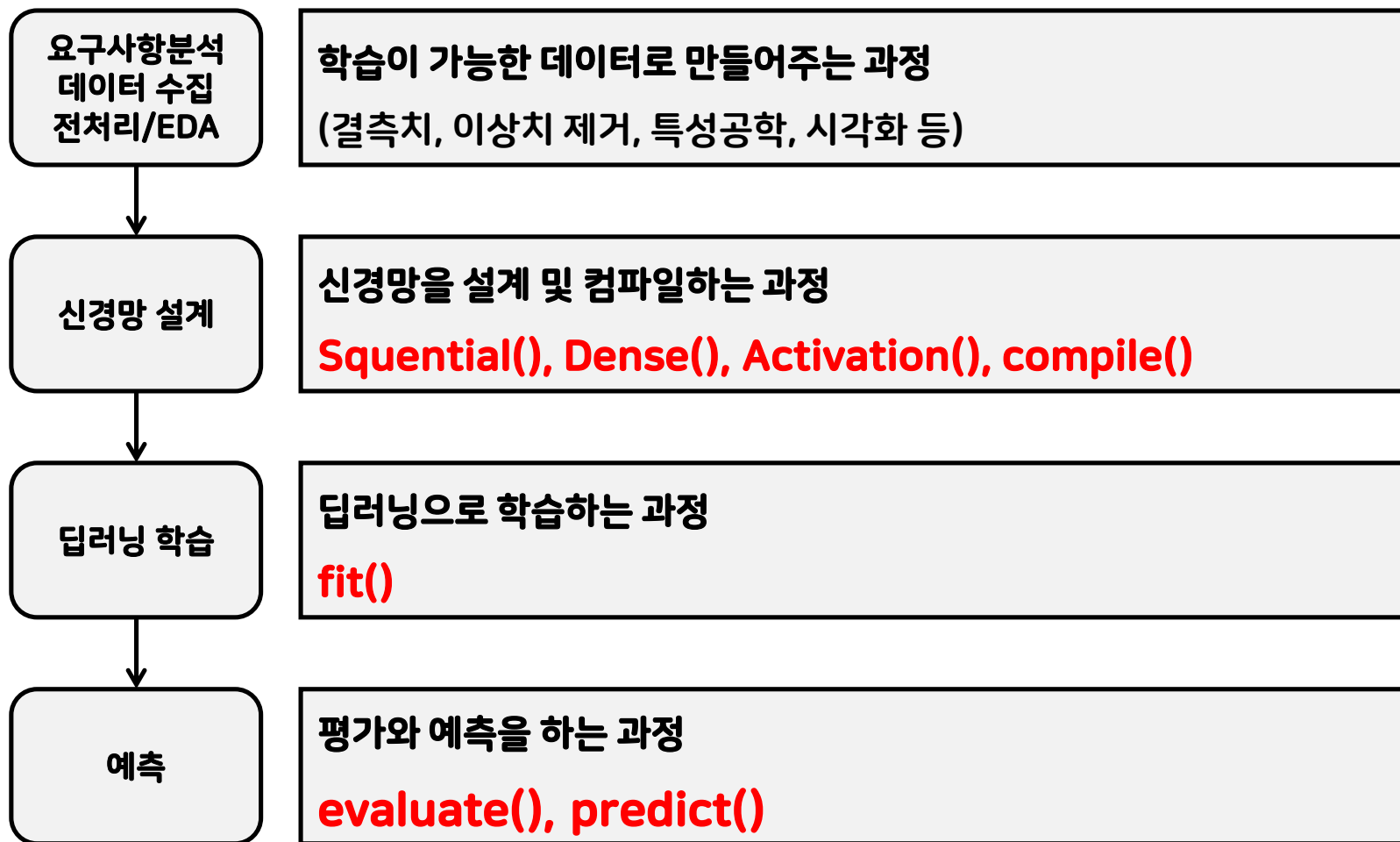
... Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-)

Enter your authorization code:

클릭 !!

인증코드 복사하여 붙여넣기

## 딥러닝 과정



## AND 논리 구현해보기

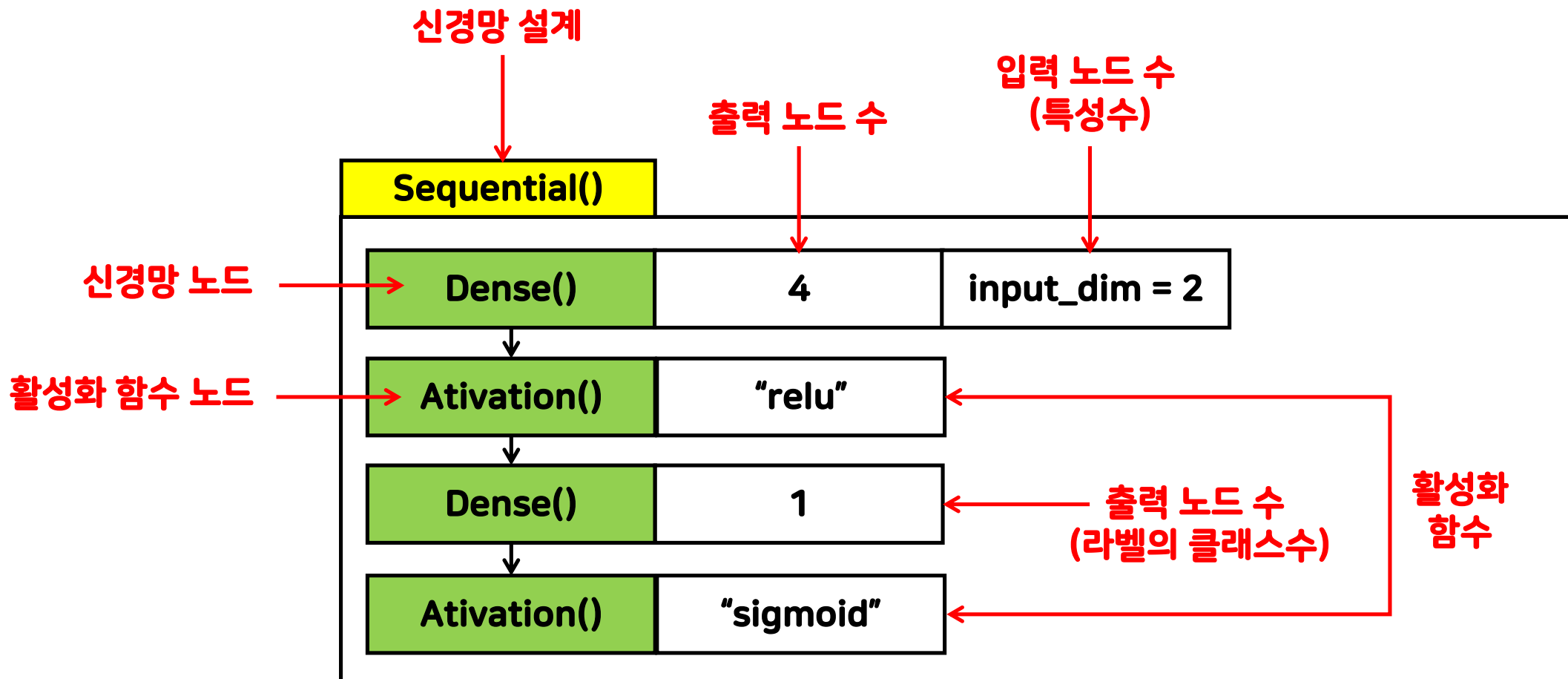
## 딥러닝 과정 - 설계 (활성화 함수를 함수를 이용하여 추가)

1	<code>model = Sequential()</code>
2	<code>model.add(Dense(4, input_dim=2))</code>
3	<code>model.add(Activation("relu"))</code>
4	<code>model.add(Dense(1))</code>
5	<code>model.add(Activation("sigmoid"))</code>

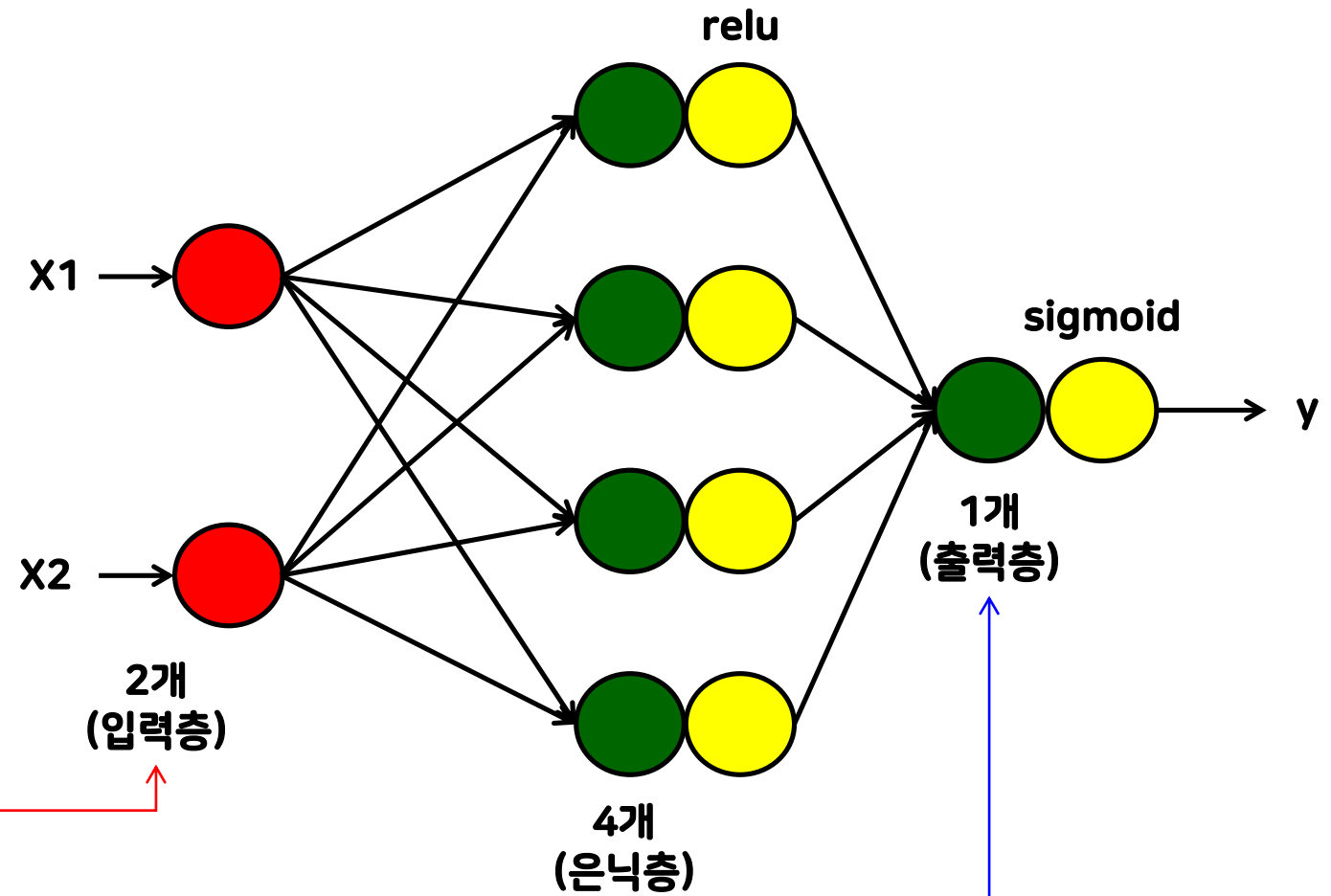
- **Sequential()** : 신경망을 한층 한층 쌓는 기능
- **add()** : 신경망 층을 추가
- **Dense()** : 실제 신경망 층을 설정하는 기능
- **Activation()** : 활성화 함수를 설정하는 기능

\* 활성화 함수 : 출력으로 나오게 하는 기준값 (sigmoid, tanh, relu, softmax 등)

## 딥러닝 과정 - 설계 (활성화 함수를 함수를 이용하여 추가)



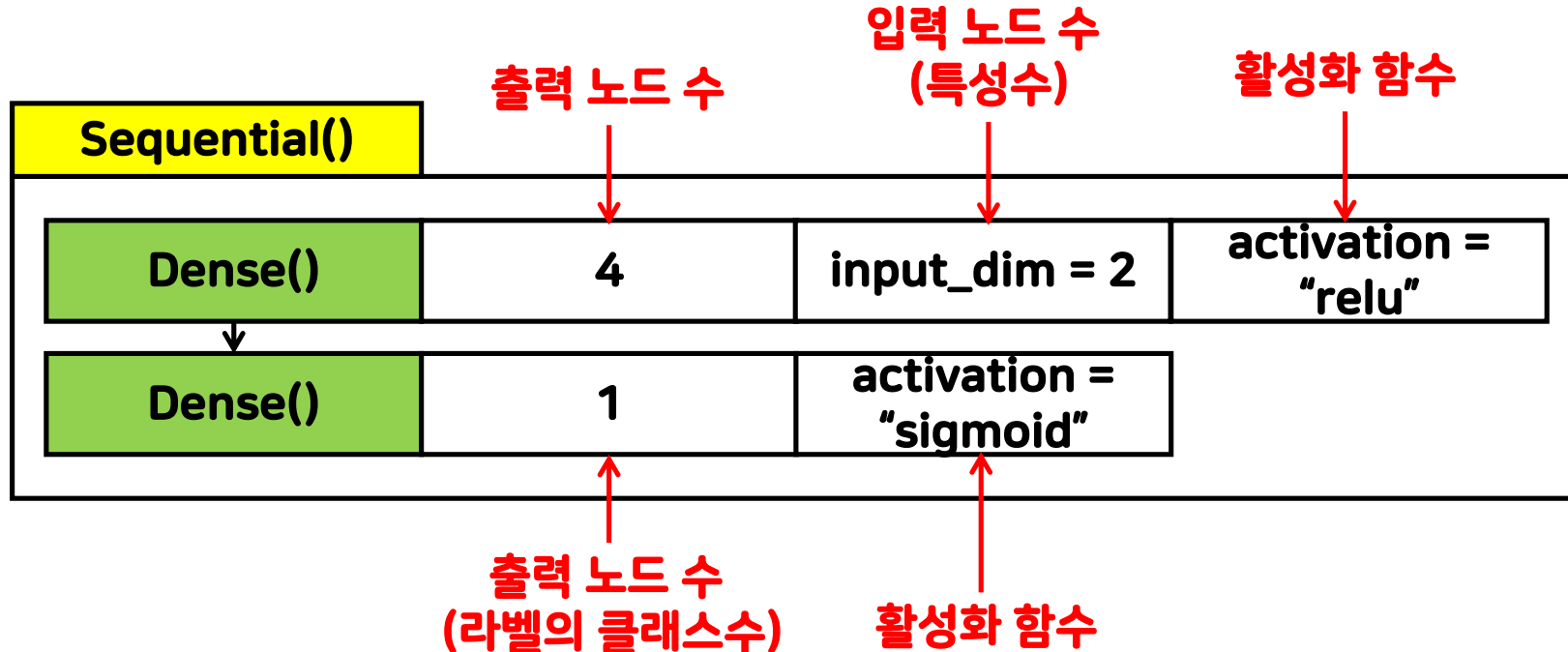
X1	X2	y
0	0	0
0	1	0
1	0	0
1	1	1



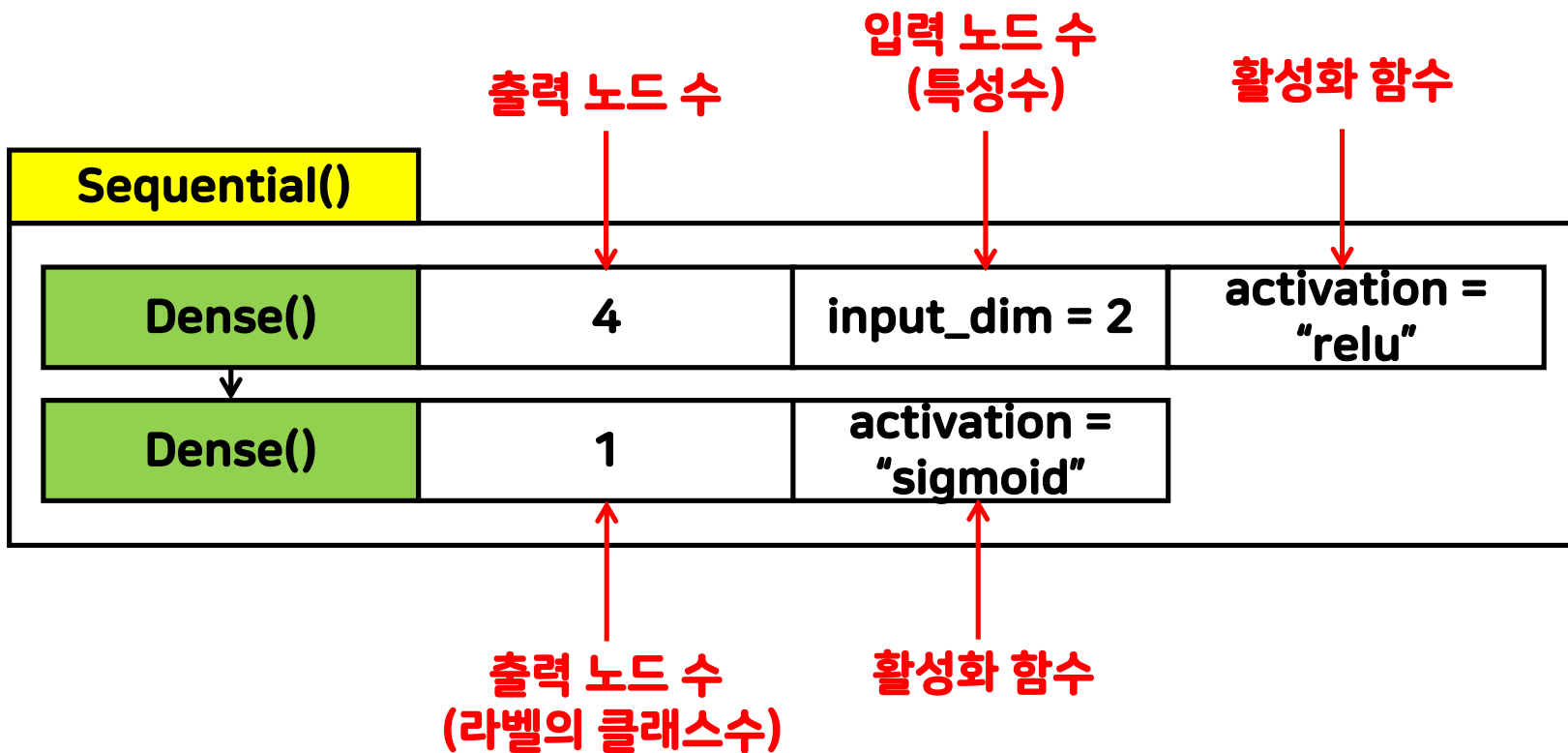


## 딥러닝 과정 - 설계 (활성화 함수를 파라미터로 설정)

1	<code>model = Sequential()</code>
2	<code>model.add(Dense(4, input_dim=2, activation='relu'))</code>
3	<code>model.add(Dense(1, activation='sigmoid'))</code>



## 딥러닝 과정 - 설계 (활성화 함수를 파라미터로 설정)

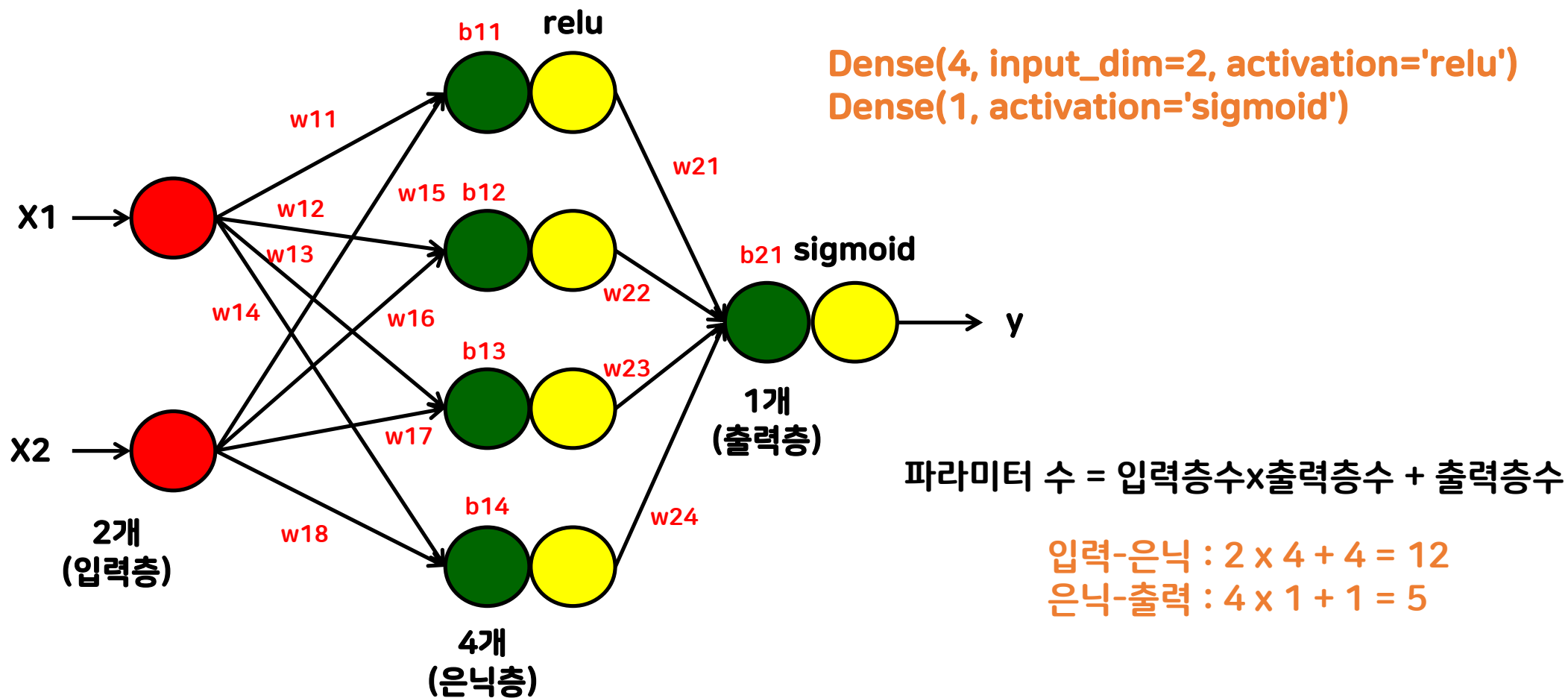


## 딥러닝 과정 - 신경망 구조 확인

1 `model.summary()`

```
Model: "sequential_19"
-----
Layer (type)                 Output Shape              Param #
=====
dense_51 (Dense)             (None, 4)                 12
-----
dense_52 (Dense)             (None, 1)                 5
=====
Total params: 17
Trainable params: 17
Non-trainable params: 0
-----
```

## 신경망의 파라미터 수



**은닉층을 추가해서 학습을 해보자**

## 딥러닝 과정 - 신경망 컴파일

1	<pre>model.compile(loss="binary_crossentropy",               optimizer="adam",               metrics=["acc"])</pre>
---	---

compile()	손실함수 (loss)	최적화 도구 (optimizer)	평가도구 (metrics)
-----------	----------------	-----------------------	-------------------

## 딥러닝 과정 - 컴파일

- **손실함수** : 오차 계산 방법 (MSE (최소자승오차), CEE (교차 엔트로피 오차) 등)
- **손실함수의 종류**

용도	손실함수
회귀	mean_square_error 또는 mse
이진 분류	binary_crossentropy
다중 분류	categorical_crossentropy

## 딥러닝 과정 - 컴파일

- **최적화 방법** : 정답을 찾아가는 방법 (SGD (확률적 경사하강법), 모멘텀, AdaGrad, Adam 등)
- 최적화 도구의 종류

rmsprop

adam

- 평가도구

['loss']

오차율

['accuracy']

['acc']

정확도



## 딥러닝 과정 - 추론

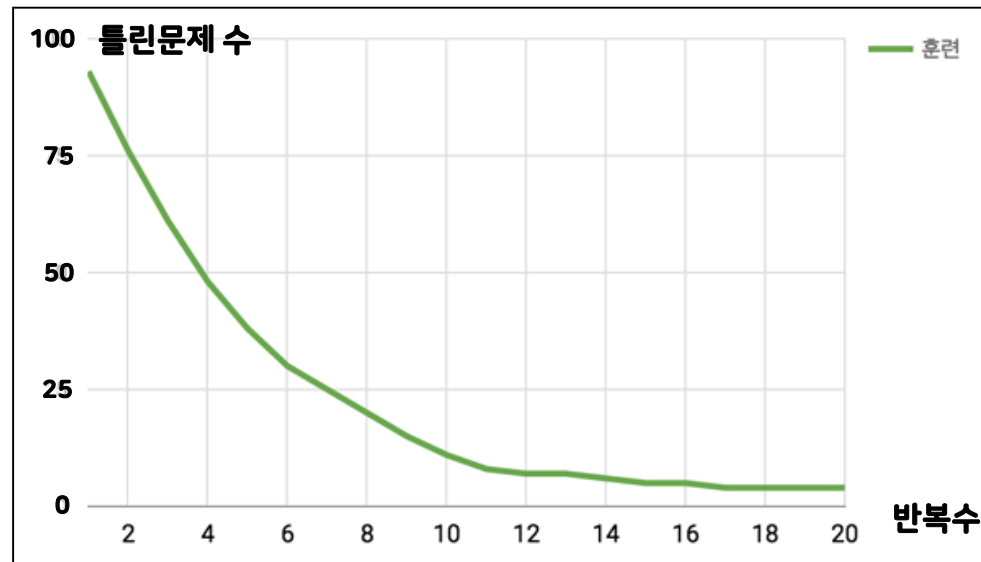
1	<code>model.fit(X, y, epochs=500, batch_size=4)</code>
---	--

<code>fit()</code>	특징데이터	라벨데이터	<code>epochs</code>	<code>batch_size</code>
--------------------	-------	-------	---------------------	-------------------------

- **epochs** : 학습 반복 회수
- **batch\_size** : 한 번 반복할 때 사용할 데이터의 수

## 딥러닝 과정 - batch\_size와 epochs

- **batch\_size** : 한번에 처리할 데이터의 수 (몇 문제를 풀고 해답을 확인하는 지를 의미)
- **epochs** : 학습을 몇 번 수행하는 지의 수 (문제를 몇 번 풀어 볼 것인지 의미)
- 반복 회수 (epoch) 마다 100문제씩 (batch) 풀 경우 틀린 문제의 수



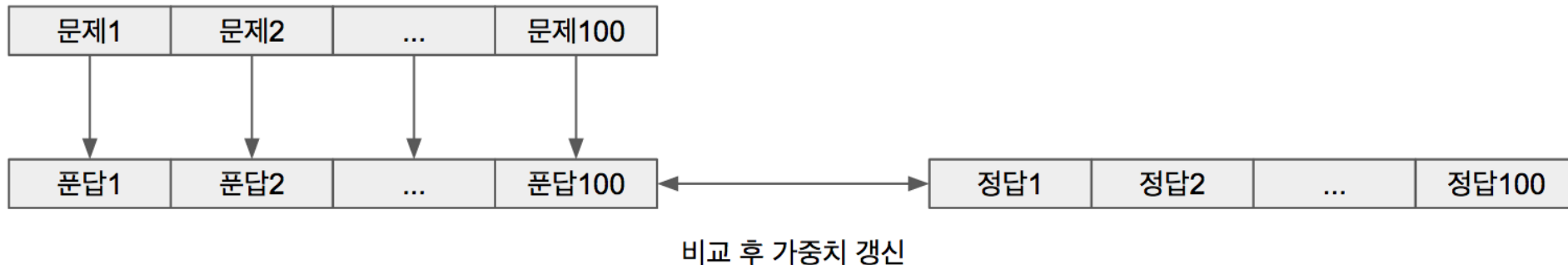
## 딥러닝 과정 - batch\_size와 epochs

- 모의고사 1회분을 가지고 학습하는 경우 (1회분은 100문항이 있고, 해답지도 제공)
  - 문제를 풀 뒤 해답지와 맞춰보면서 학습이 이루어지기 때문에 해답지가 없으면 학습이 안 됨.
- 데이터 : 100문항의 문제들
- 라벨 : 100문항의 답들

문제지	문제1	문제2	문제3	문제4	문제5	문제6	...	문제100
해답지	정답1	정답2	정답3	정답4	정답5	정답6	...	정답100

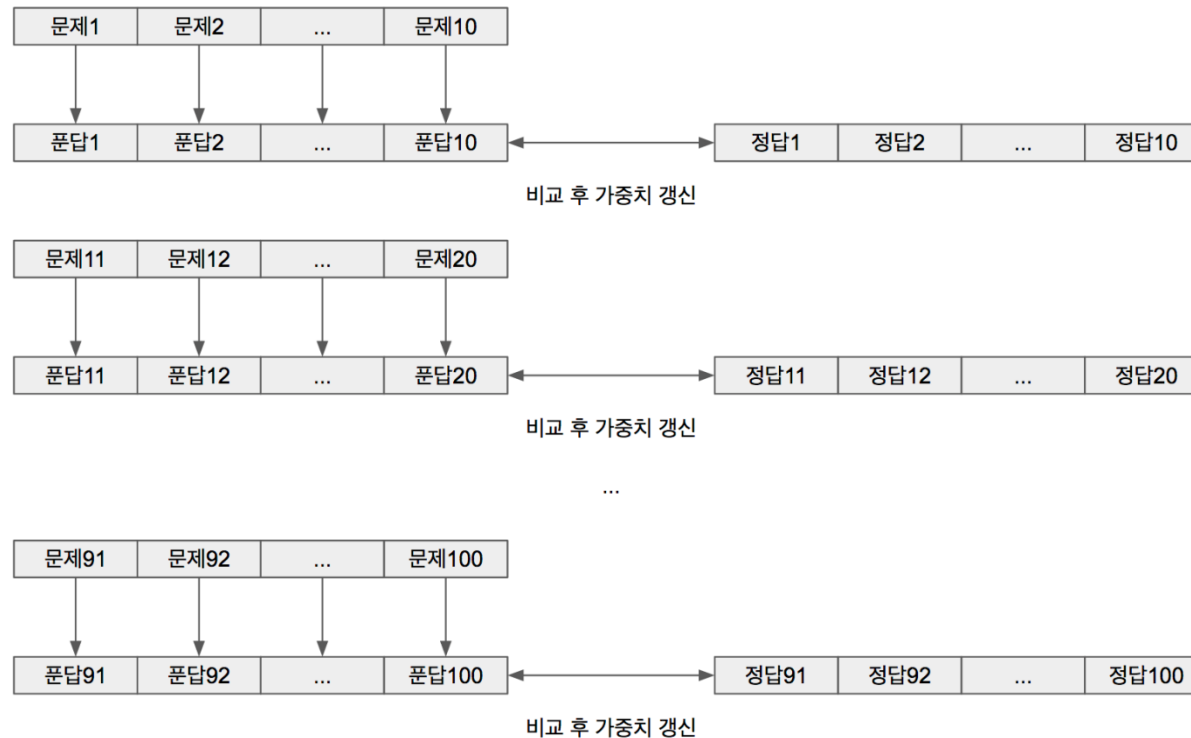
## 딥러닝 과정 - batch\_size

- 몇 문항을 풀고 해답을 맞추는 지를 의미
- 100문항일 때, 배치사이즈가 100이면 전체를 다 풀고 난 뒤에 해답을 맞춰보는 것
- 전체 문제를 푼 뒤 해답과 맞추므로 이 때 가중치 갱신은 한 번만 일어남
- 다 풀어보고 해답을 맞춰보기 때문에 한 문제를 틀릴 경우 이후 유사 문제를 모두 틀릴 경우가 많음
- 100문항 다 풀고 해답과 맞추어보려면 문제가 무엇이었는지 다 기억을 해야 맞춰보면서 학습이 됨  
→ 기억력(용량)이 커야 함



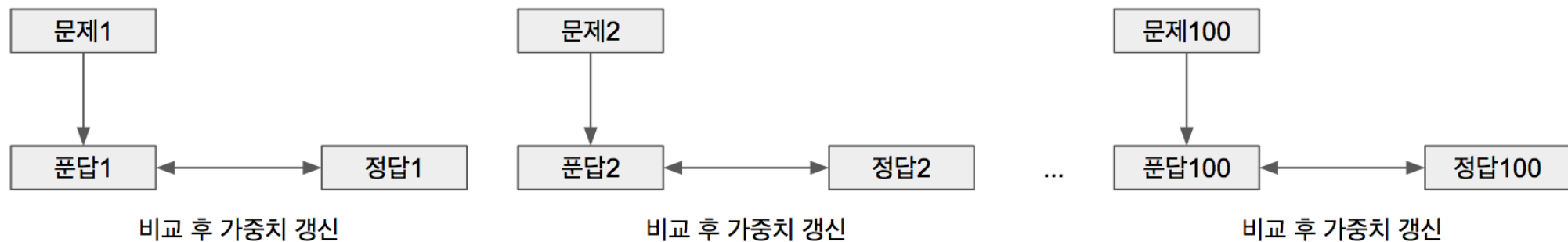
## 딥러닝 과정 - batch\_size

- 배치사이즈가 10이면 열 문제씩 풀어보고 해답 맞춰보는 것
- 100문항을 10문제씩 나누어서 10번 해답을 맞추므로 가중치 갱신은 10번 일어남.



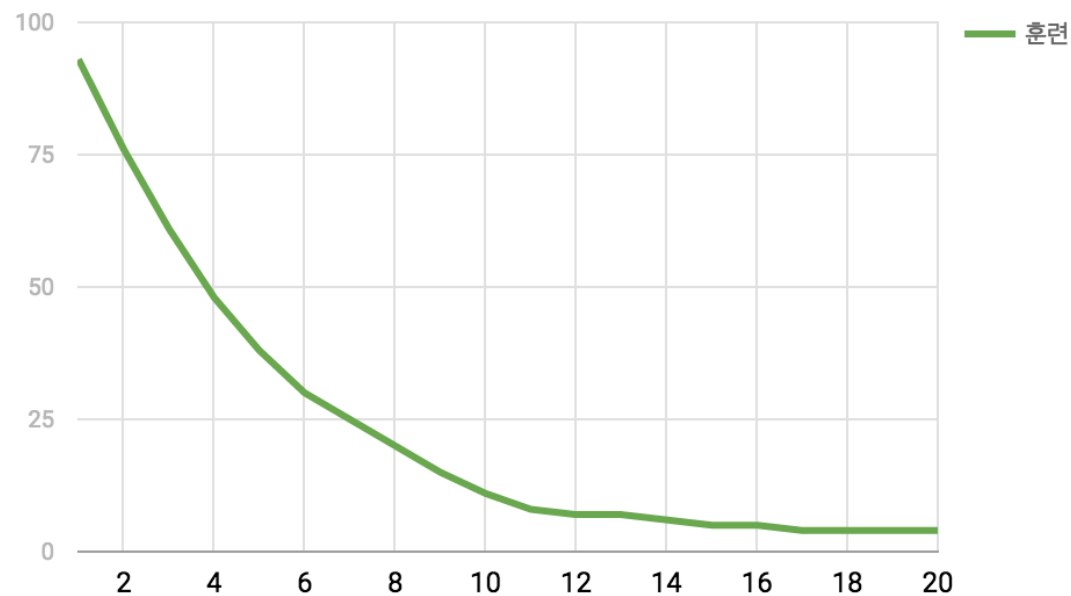
## 딥러닝 과정 - batch\_size

- 배치사이즈가 1이면 한 문제 풀고 해답 맞춰보고 또 한 문제 풀고 맞춰보고 하는 것
- 한 문제를 풀 때마다 가중치 갱신이 일어나므로 횟수는 100번
- 유사문제 중 첫 문제를 틀렸다고 하더라도 해답을 보면서 학습하게 되므로 나머지 문제는 맞추게 됨.
- 1문항씩 풀고 해답 맞추면 학습은 꼼꼼히 잘 되겠지만 시간이 너무 걸림



## 딥러닝 과정 - epochs

- 풀이를 반복할수록 틀린 개수가 적어짐
- 처음에는 틀린 개수가 확 적어지지만 반복이 늘어날수록 완만하게 틀린 개수가 줄어듬



## 딥러닝 과정 - epochs

- 모의고사 1회분을 20번 푸는 것과 서로 다른 모의고사 20회분을 1번 푸는 것과는 어떤 차이가 있을까요?

→ 분야에 따라 데이터특성에 따라 다를 것

→ 잡다한 문제를 많이 푸는 것보다 양질의 문제를 여러 번 푸는 것이 도움

→ 현실적으로 데이터를 구하기가 쉽지 않기 때문에 제한된 데이터셋으로 반복적으로 학습하는 것이 효율적



## 딥러닝 과정 - epochs

- epochs를 무조건 늘리면 좋을까요 ?

→ 하나의 문제집만 계속 학습하면 오히려 역효과가 발생

→ 피아노 칠 때 처음에 곡을 연습할 때는 악보를 보면서 치다가 다음엔 악보안보고도 치고, 나중엔 눈감고도 칠

→ 하지만 다른 곡은 ? → **오버피팅(overfitting)**

→ 악보보고 잘 치는 정도에서 그만 연습하는 것이 좋음

→ 실제로 모델을 학습할 때도 오버피팅이 일어나는 지 체크하다가 조짐이 보이면 학습을 중단

## 딥러닝 과정 - 학습

1	<code>pred = model.predict(X_new)</code>
---	--

`predict()`

추론용 특징데이터

## 딥러닝 과정 - 평가

1	<code>score = model.evaluate(X_new, y_new)[1]</code>
---	--

<code>evaluate()</code>	평가용 특징데이터	평가용 라벨데이터
-------------------------	-----------	-----------

- 첫 번째 요소([0])는 오차
- 두 번째 요소([1])는 정확도

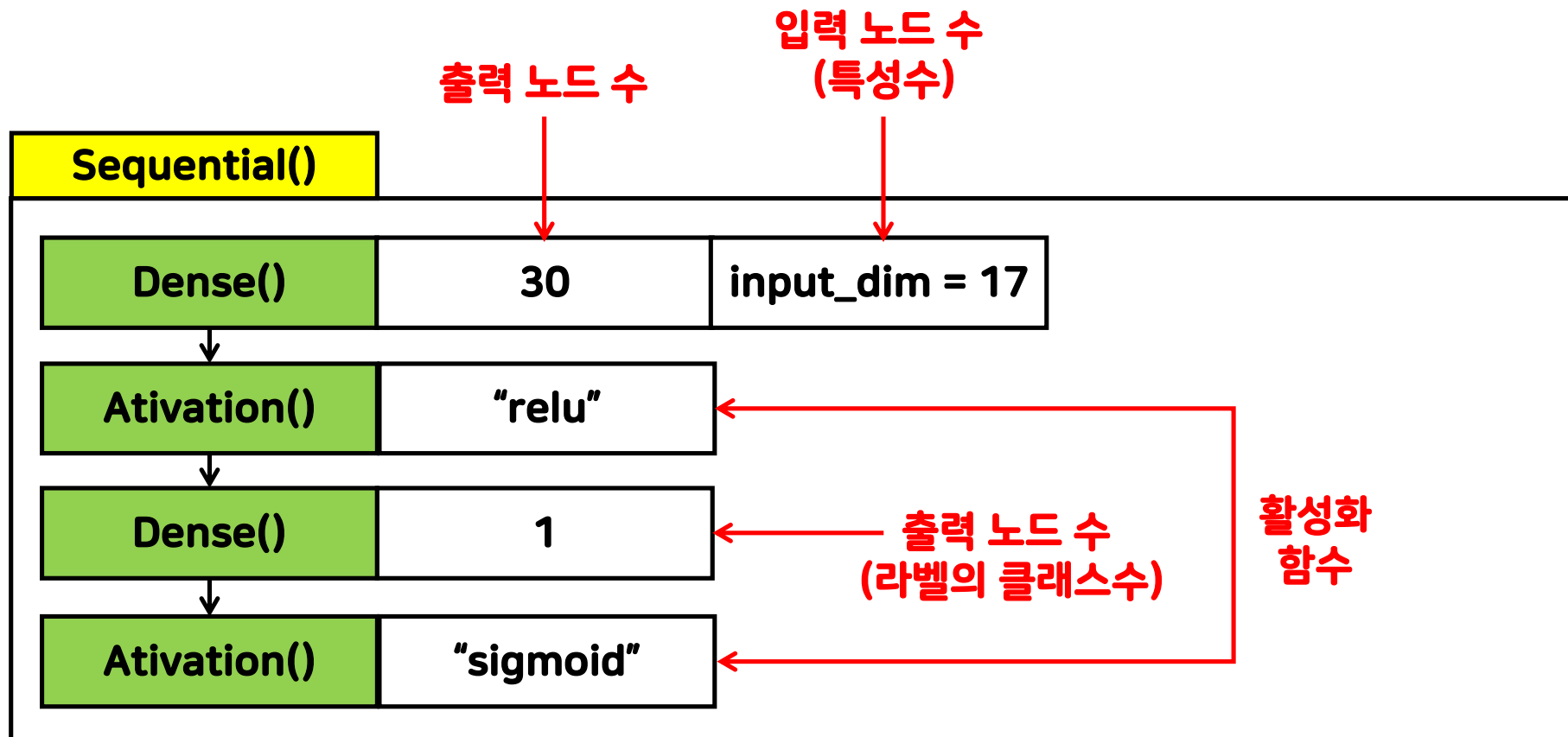
**폐암 수술 환자의 생존율 예측**  
**(종양유형, 폐활량, 흡연여부, 천식여부, 나이 등)**

## 폐암 수술환자 데이터셋 (ThoraricSurgery.csv)

- 폴란드 브로츠와프 의과대학에서 2013년 공개한 폐암 수술 환자의 수술 전 데이터와 수술 후 생존결과를 기록한 의료 기록 데이터
- 18개 항목으로 구성된 470개의 데이터로 구성되고 각 항목은 ,로 구분
- 종양 유형, 폐활량, 호흡곤란여부, 고통 정도, 기침, 흡연, 천식여부 등 17가지 환자 상태
- 18번째 항목은 수술 후 생존 결과 (1 : 생존, 0 : 사망)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	293	1	3.8	0	0	0	0	0	0	0	12	0	0	0	1	0	62	0
2	1	2	2.88	2.16	1	0	0	0	1	1	14	0	0	0	1	0	60	0
3	8	2	3.19	2.5	1	0	0	0	1	0	11	0	0	1	1	0	66	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
470	447	8	5.2	4.1	0	0	0	0	0	0	12	0	0	0	0	0	49	0

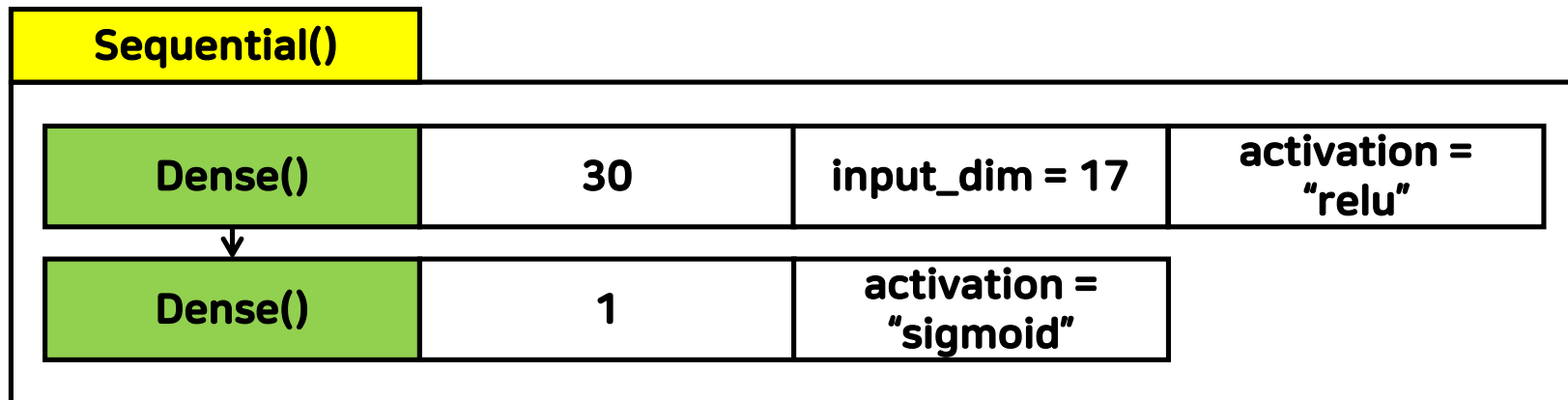
## 딥러닝 과정 - 설계



## 딥러닝 과정 - 설계

1	<code>model = Sequential()</code>
2	<code>model.add(Dense(30, input_dim=17))</code>
3	<code>model.add(Activation("relu"))</code>
4	<code>model.add(Dense(1))</code>
5	<code>model.add(Activation("sigmoid"))</code>

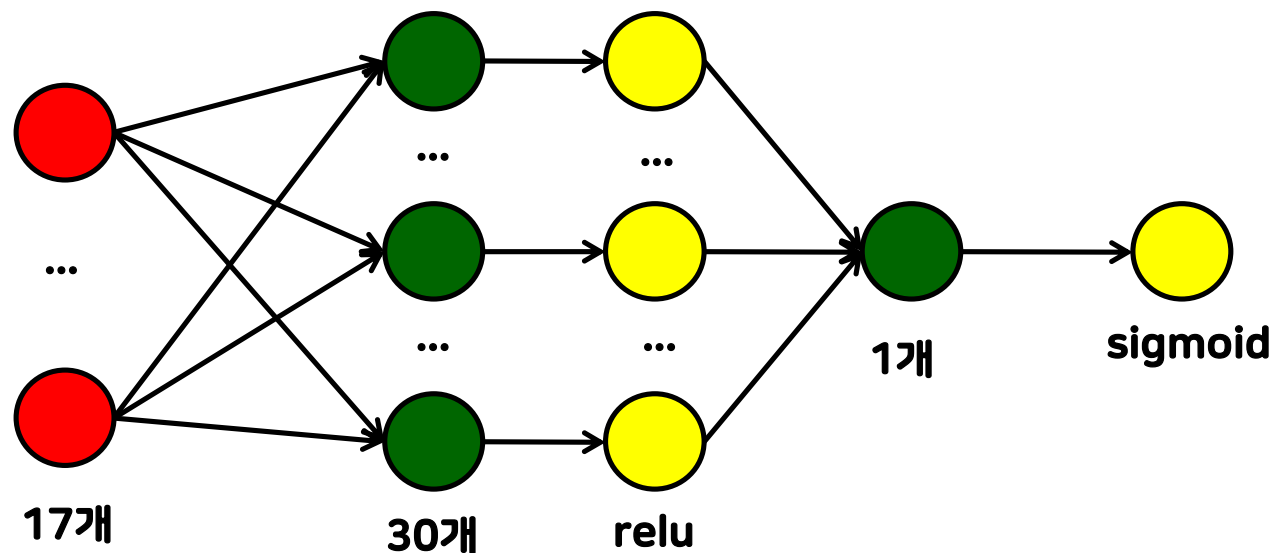
## 딥러닝 과정 - 설계



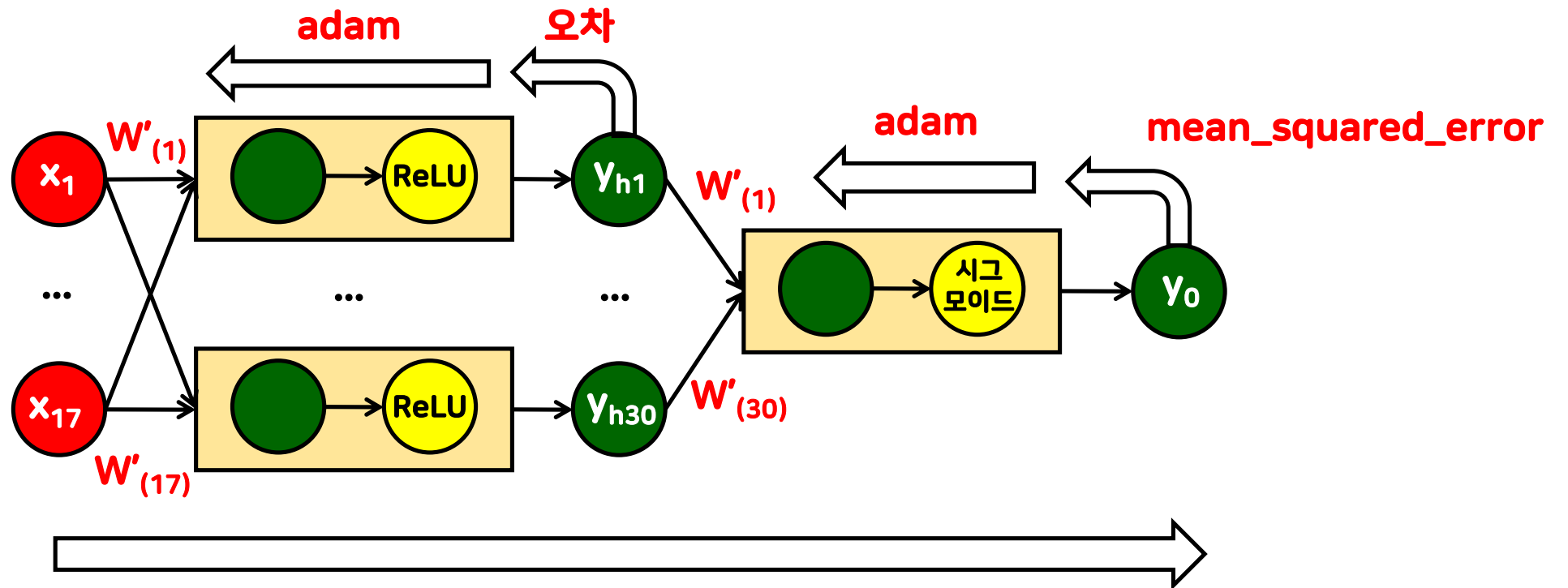


## 딥러닝 과정 - 설계

1	<code>model = Sequential()</code>
2	<code>model.add(Dense(30, input_dim=17, activation='relu'))</code>
3	<code>model.add(Dense(1, activation='sigmoid'))</code>



## 딥러닝 과정 - 학습



## 딥러닝 과정 - 신경망 컴파일

1	<code>model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc"])</code>
---	---

## 딥러닝 과정 - 학습

1	<code>model.fit(X, y, epochs=500, batch_size=4)</code>
---	--

**Wine 데이터셋으로 딥러닝을  
이용한 회귀 / 다중 분류 모델 학습을 해보자**

## wine 데이터셋

- 포트투칼의 비뉴 베르드 지방에서 만들어진 와인을 측정한 데이터
- 1,599개의 레드와인 데이터, 4,898개의 화이트와인 데이터 (총 6,497개 데이터)
- 12개의 정보와 1개의 클래스로 구성

	0	1	2	3	4	5	6	7	8	9	10	11	12
	주석산 농도	아세트 산농도	구연산 농도	진류당 분농도	염화나트 륨농도	유리 아 화산 농도	총 아 황산 농도	밀도	pH	황산 칼륨 농도	알코올 도수	와인맛 (5-9등 급)	레드 1/화 이트0
964	8.5	0.47	0.27	1.9	0.058	18	36	0.99518	3.16	0.85	11.1	6	1
664	12.1	0.4	0.52	2	0.092	18	54	1	3.03	0.66	10.2	5	1
1692	6.9	0.21	0.33	1.8	0.034	18	136	0.9899	3.25	0.41	12.6	7	0
5801	6.7	0.24	0.31	2.3	0.044	18	113	0.99013	3.29	0.46	12.9	6	0
6497	6	0.21	0.38	0.8	0.02	22	98	0.98941	3.26	0.32	11.8	6	0

## 앞으로 학습할 신경망

Conv1D()

Conv2D()

MaxPool2D()

MaxPool1D()

GlobalMaxPooling1D()

Flatten()

Dropout()

Embedding()

SimpleRNN()

LSTM()

GRU()

## 앞으로 학습할 신경망

