AI TRAINING SERIES

Perceptron & ANN

AI/DL areas and applications: past, current, upcoming

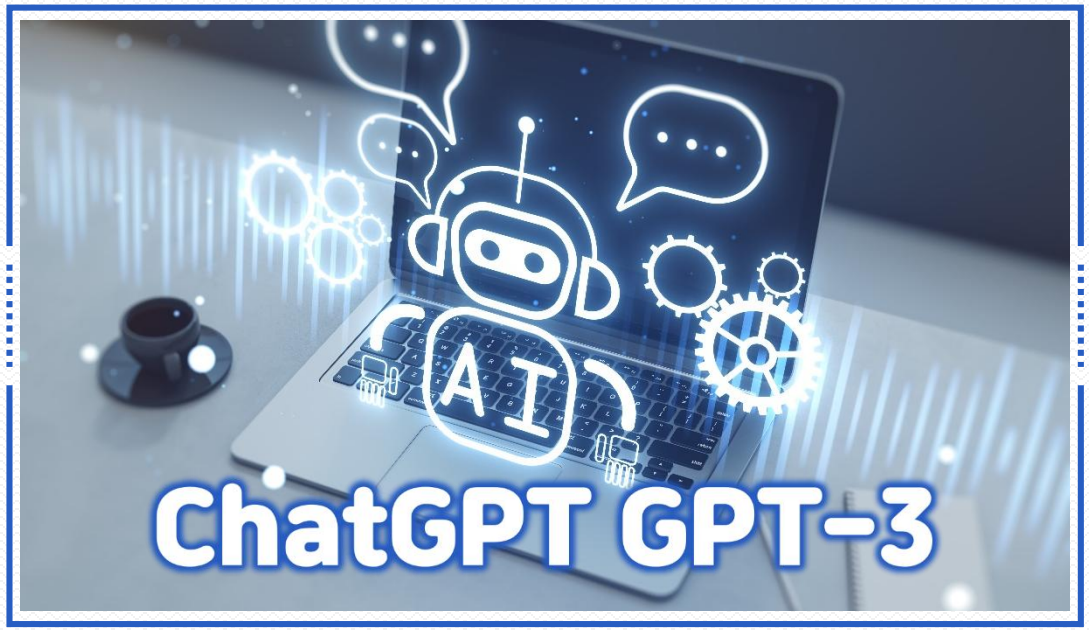
캄브리아기(Cambrian Period)

AI/DL Momentum

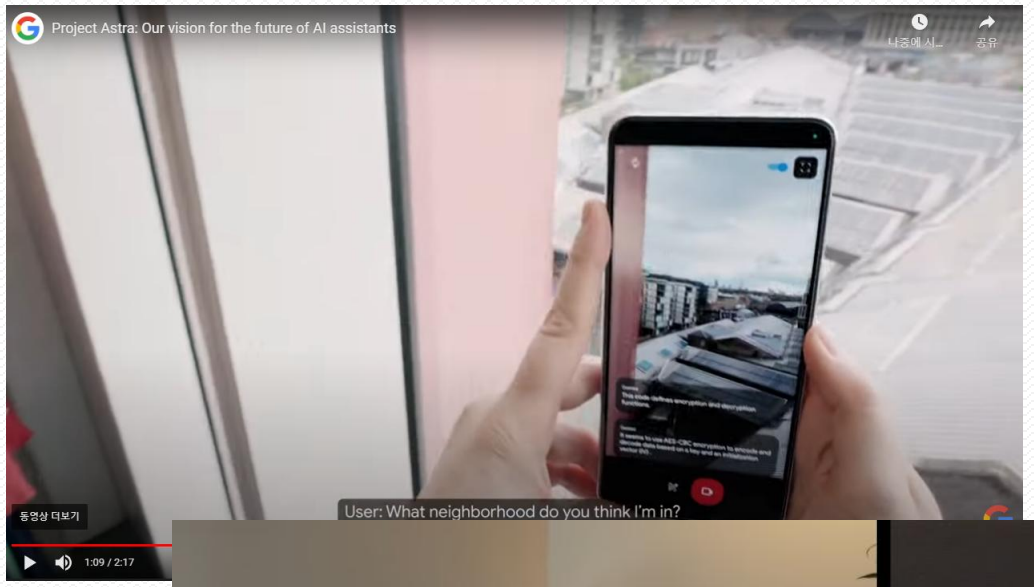
2016년 이세돌 vs. 알파고로 인해
대중들에게 AI 대한 관심도가 급상승함

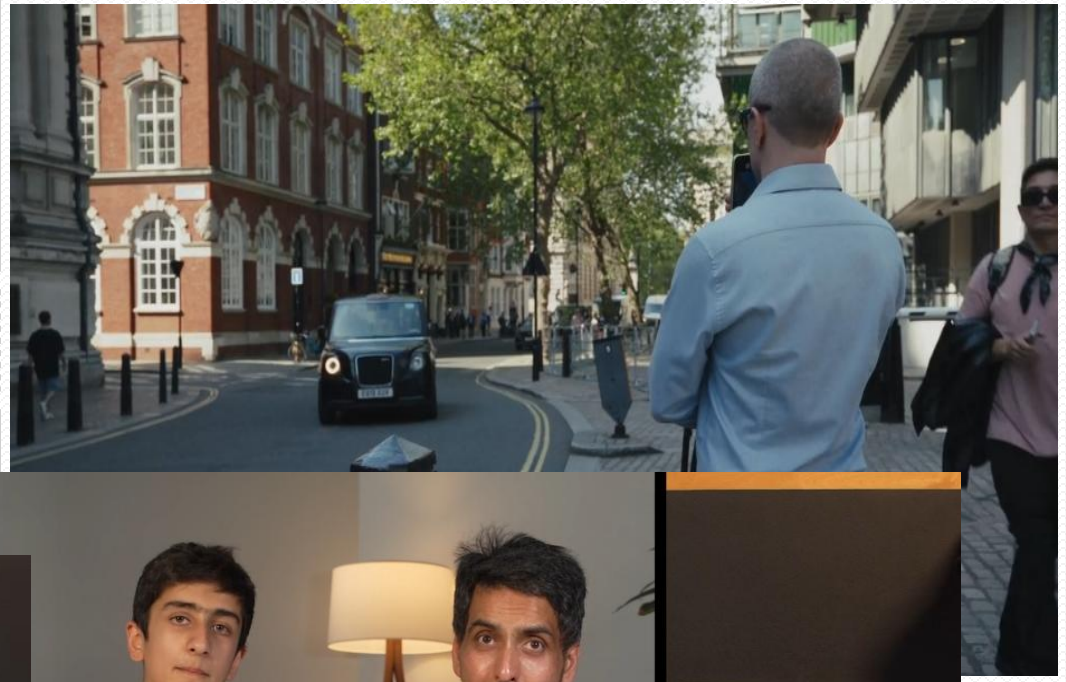
AI/DL Momentum

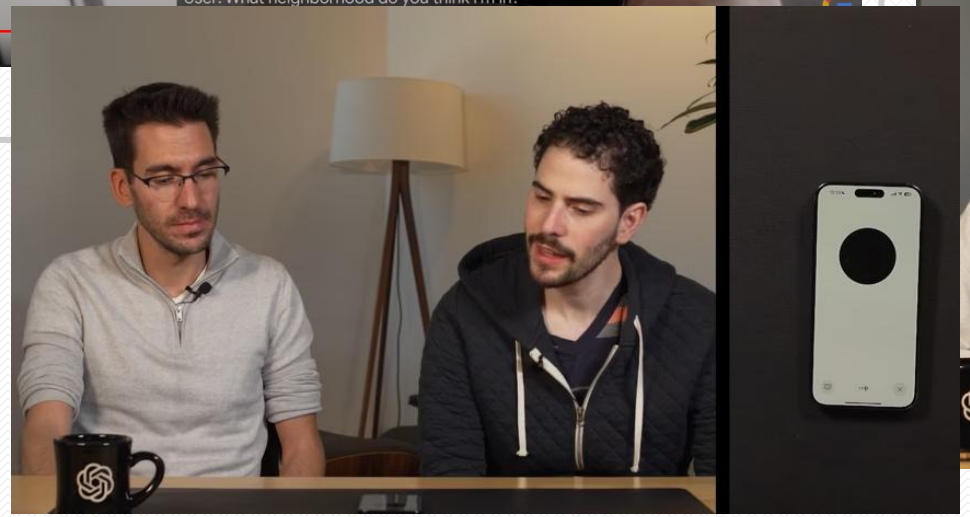
2022년 large langauge model(LLM) 기반
ChatGPT가 등장함



AI/DL Momentum : Be My eyes in London

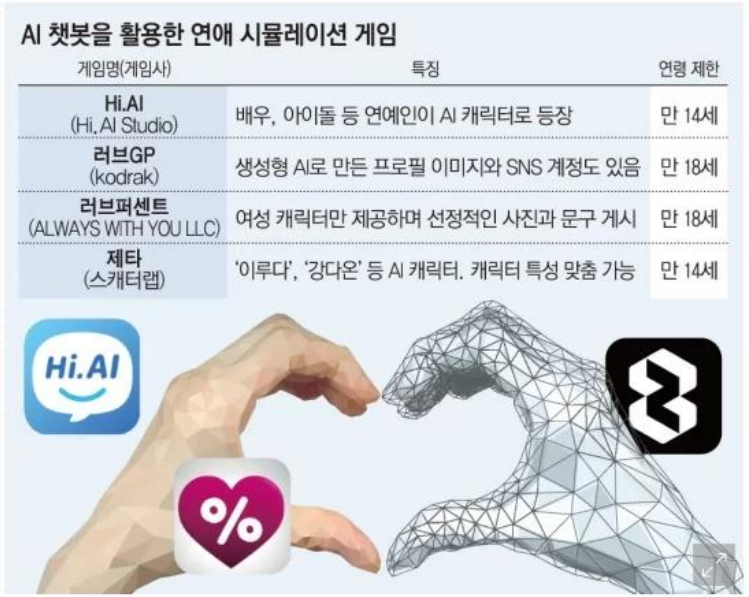


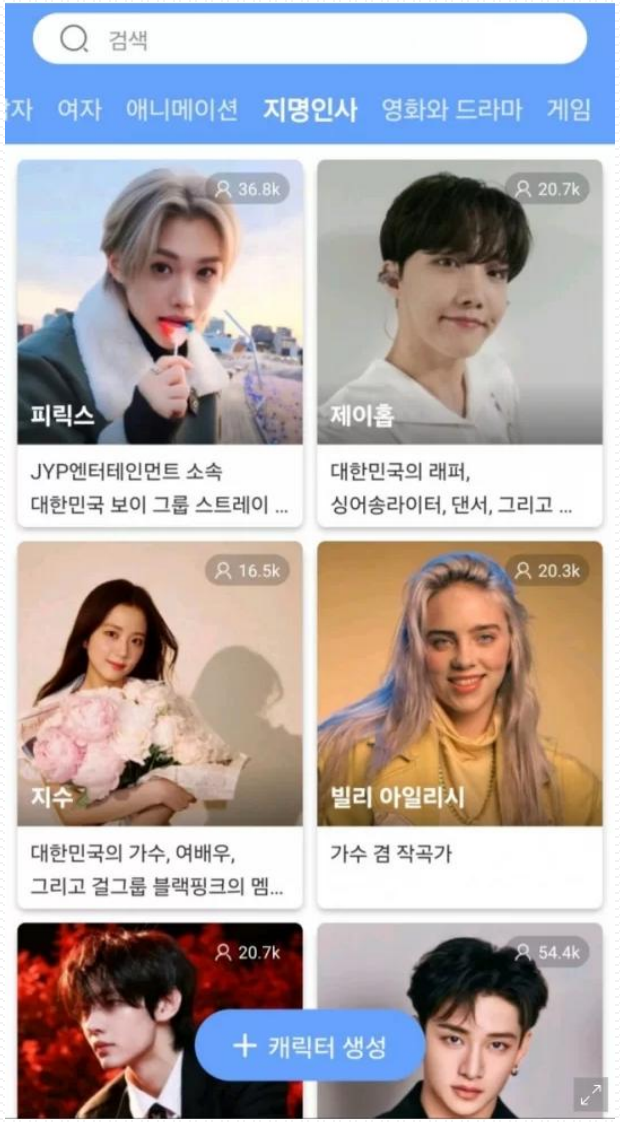


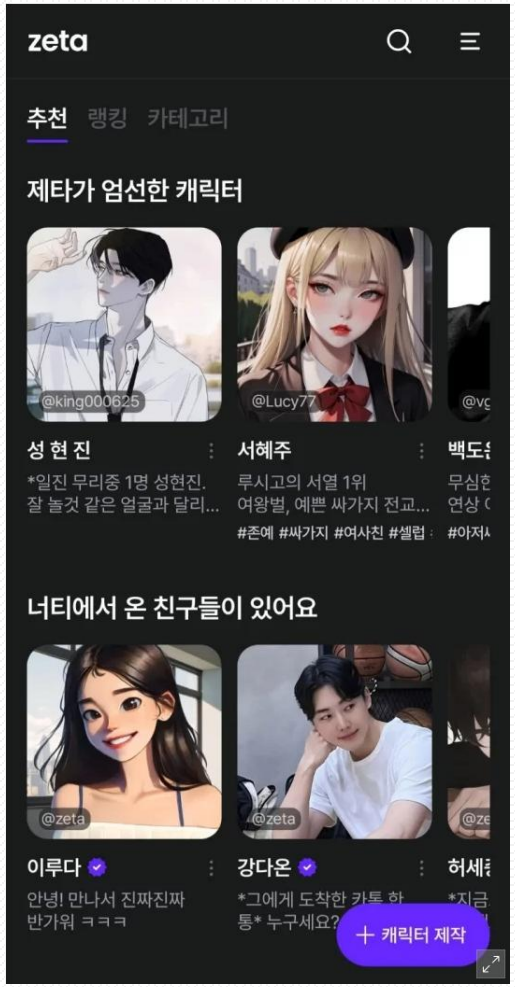


https://deepmind.google/technologies/gemini/

AI/DL Momentum : Her.





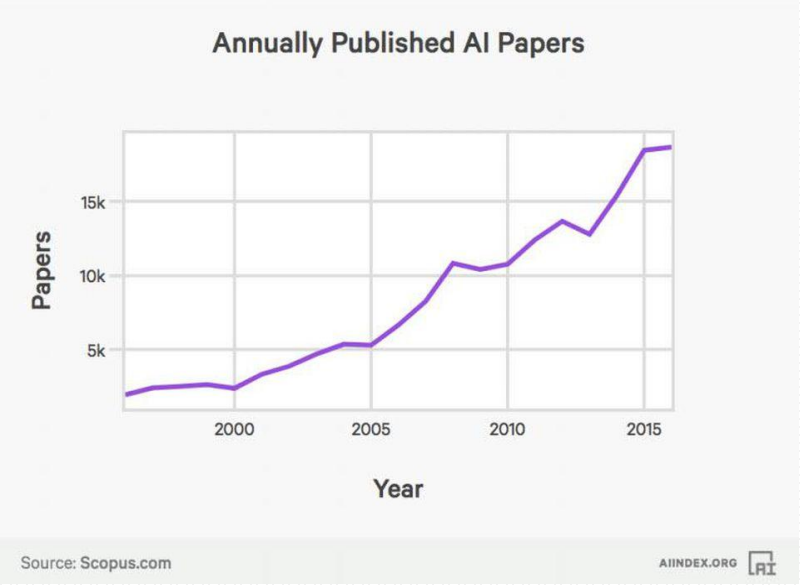


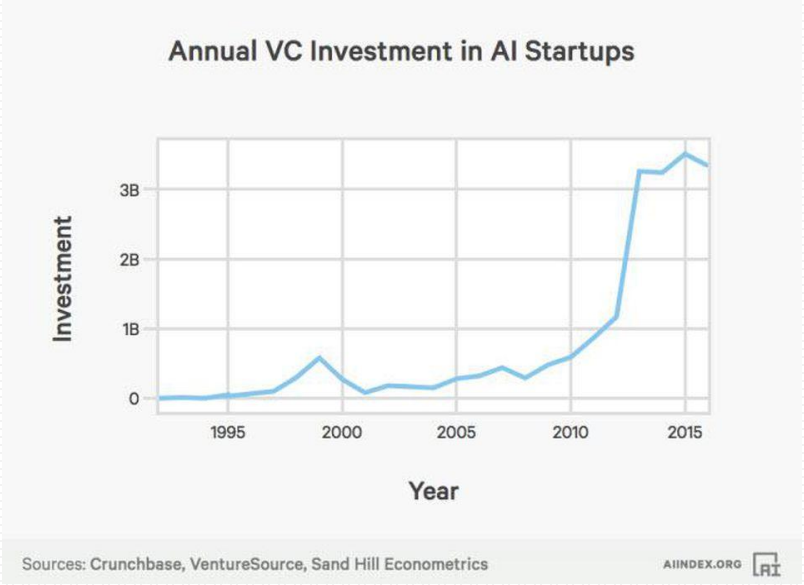
https://m.news.nate.com/view/20240517n01311

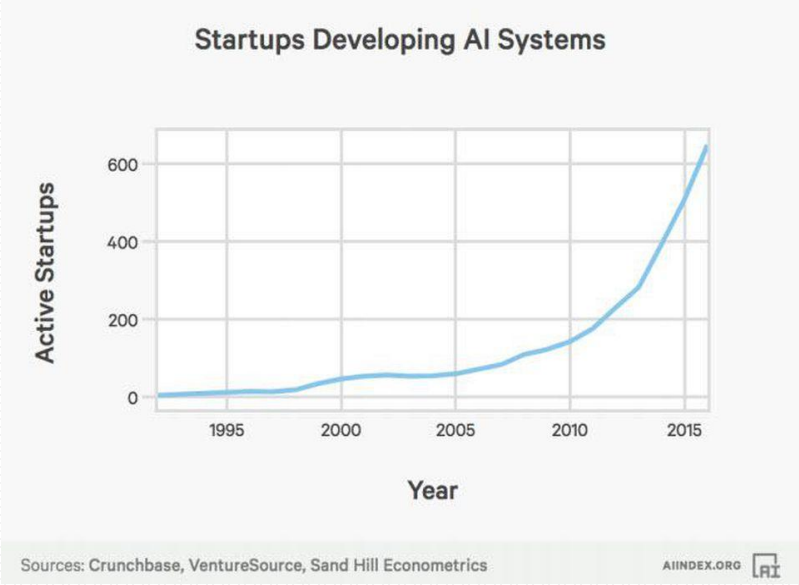
AI/DL Momentum : chatGPT app for iOS.

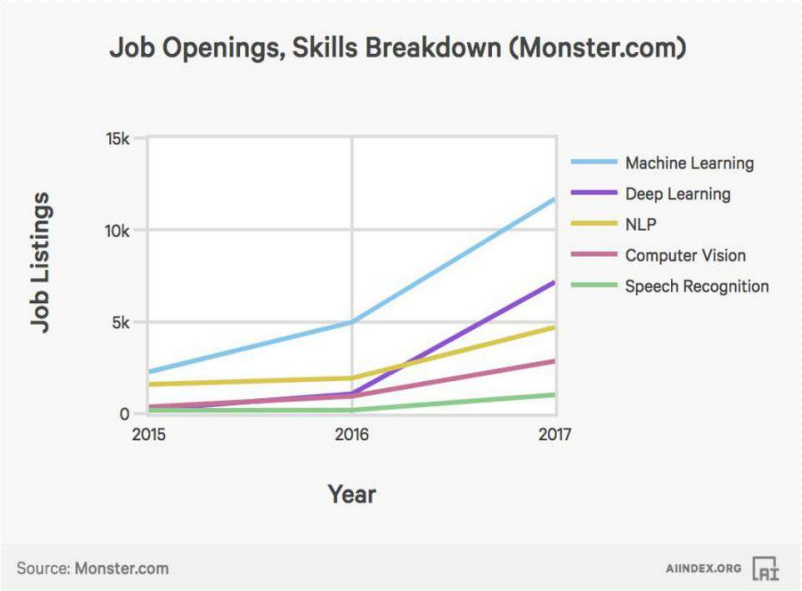
https://openai.com/index/introducing-the-chatgpt-app-for-ios/

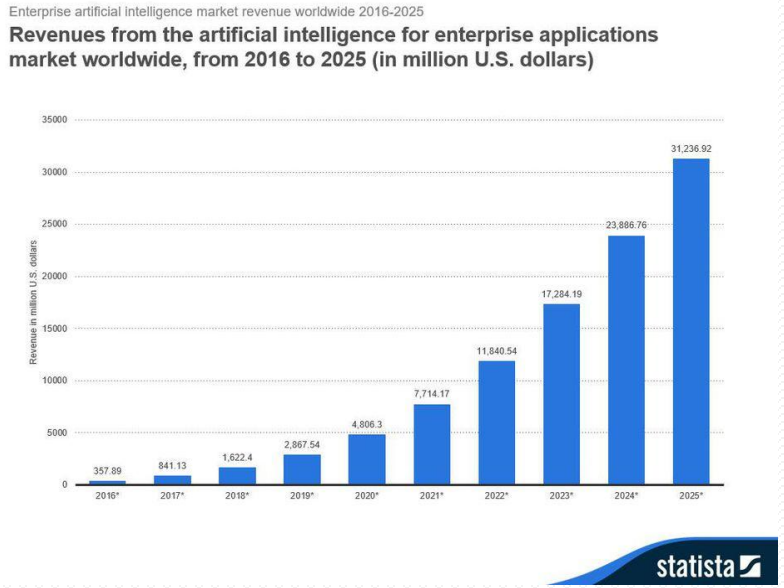
AI GROWTH





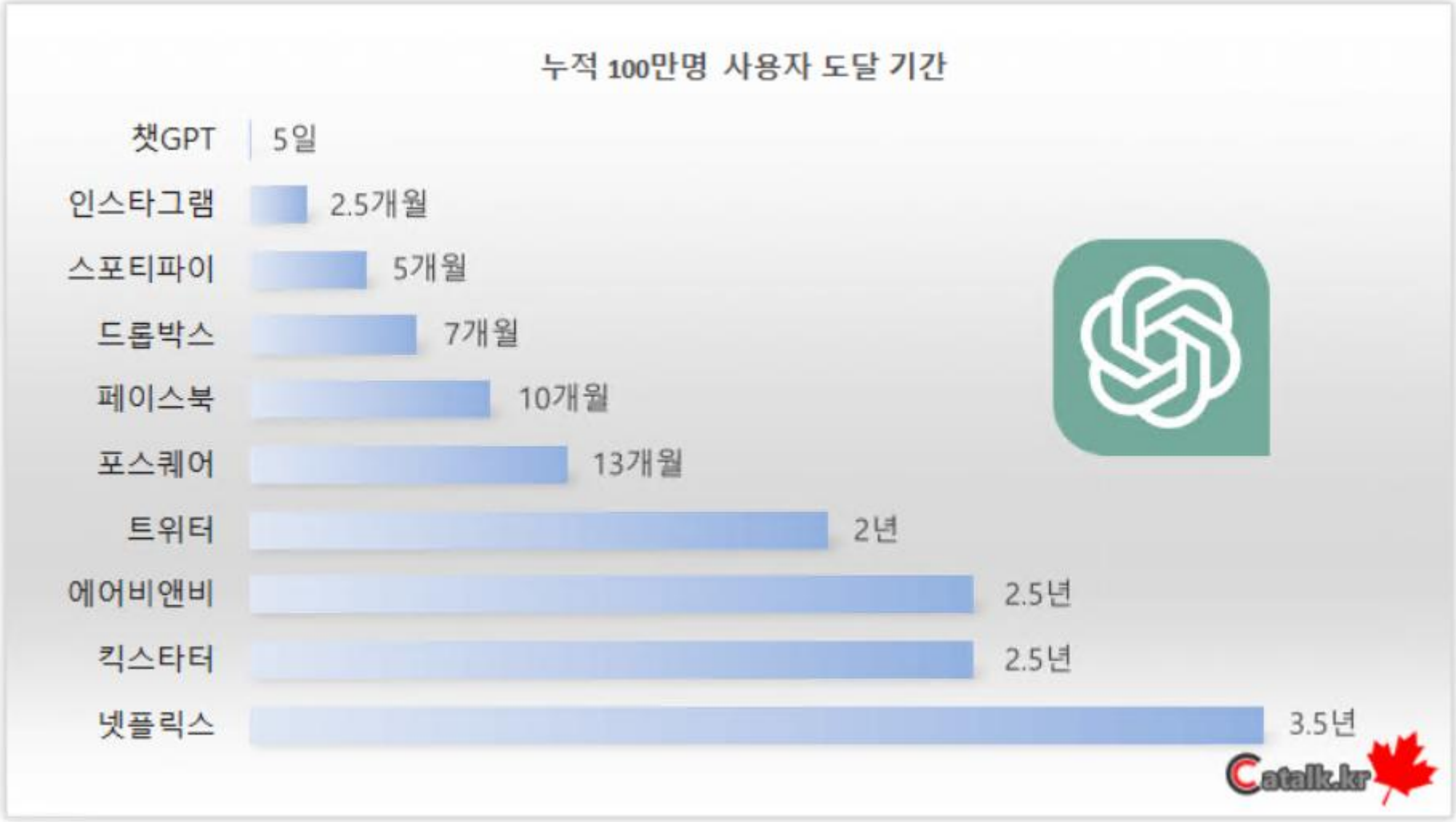






source

AI GROWTH


누적 100만명 사용자 도달 기간
챗GPT
5일
인스타그램
2.5개월
스포티파이
5개월
드롭박스
7개월
페이스북
10개월
포스퀘어
13개월
트위터
2년
에어비앤비
2.5년
킥스타터
2.5년
넷플릭스
3.5년
Catalk.kr

IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE ILSVRC

2010 - NEC-UIUC Lin et al.

2011 - XRCE Florent Perronnin, Jorge Sanchez

2012 - AlexNet

2013 - ZFNet

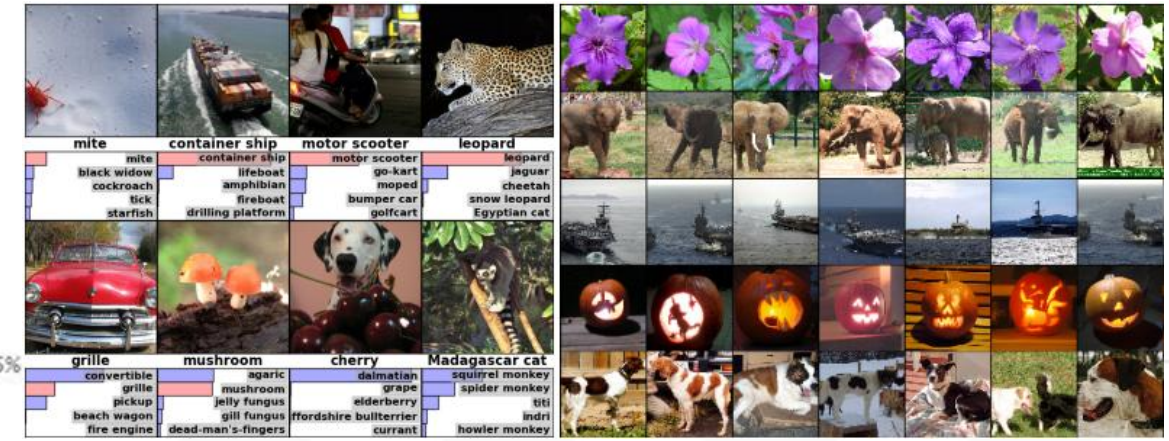
2014 – GoogLeNet

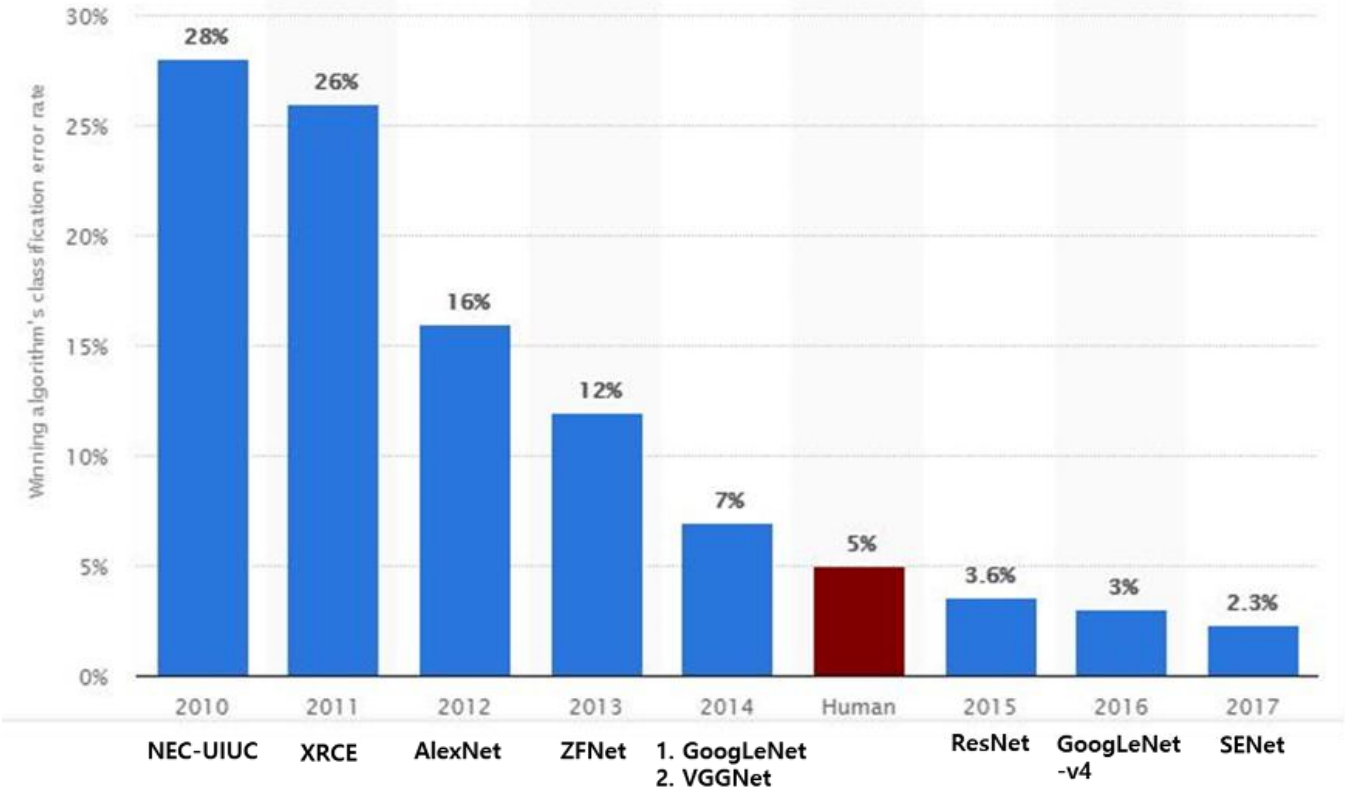
VGGNet Second Winner

2015 - ResNet

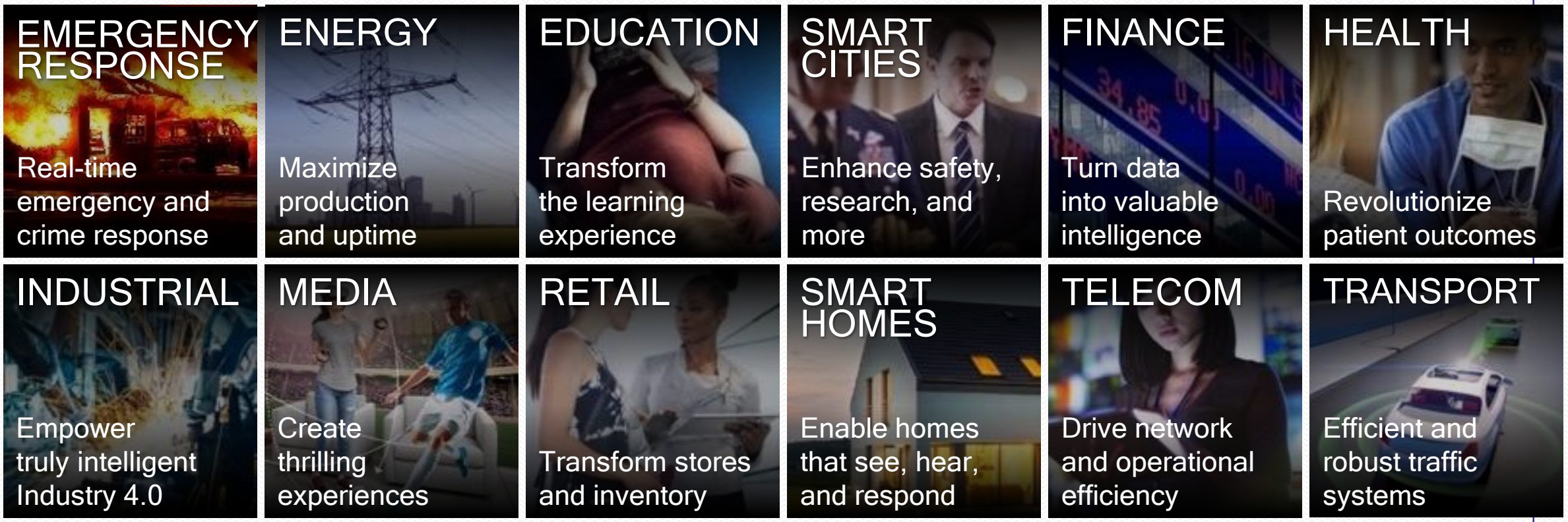
2016 - GoogLeNet-v4

2017 - SENet



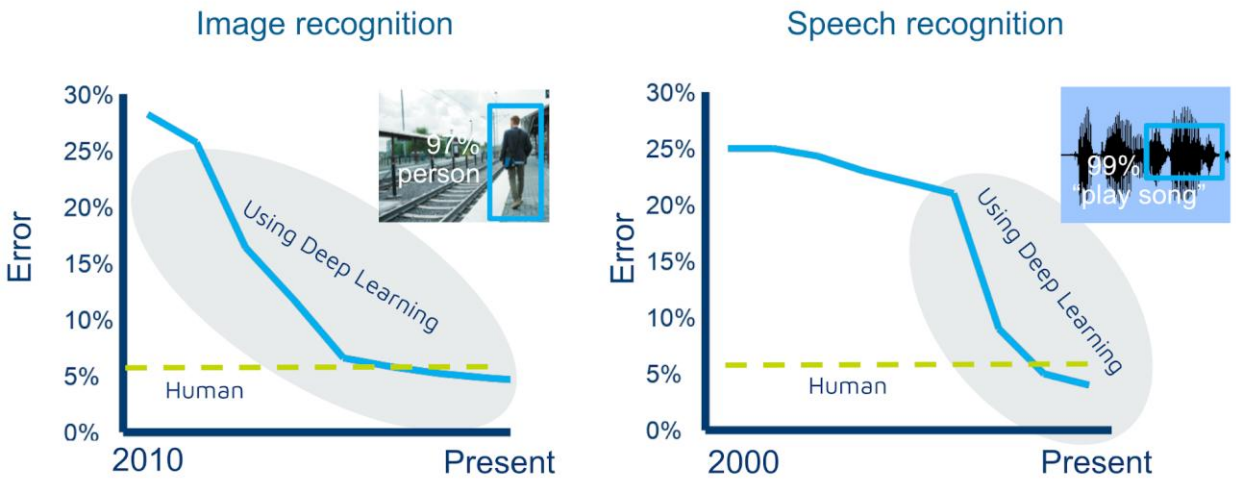


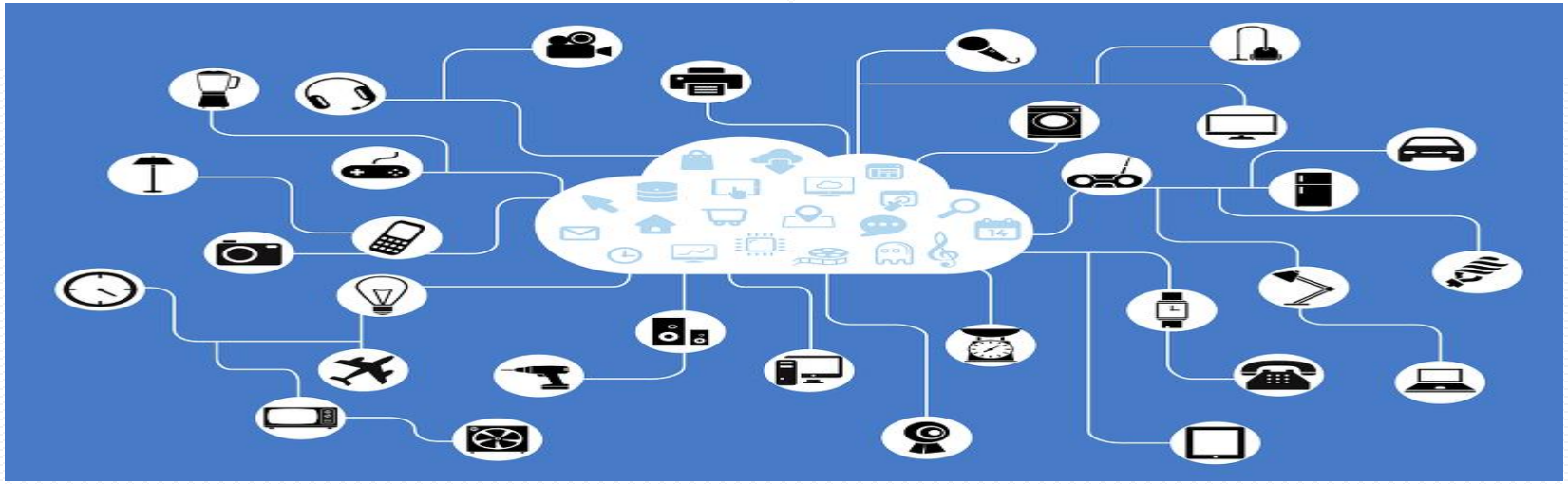
AI is changing every market



WHY NOW?

Bigger Datasets	Better Hardware	Smarter Algorithms
IMAGENET 10M labeled images YouTube 8M categorized videos	**Moore's Law** Cost / GB in 1995: $1000 Cost / GB in 2020: $0.02	**Recurrent Neural Networks** **Convolutional Neural Nets** **LSTM**





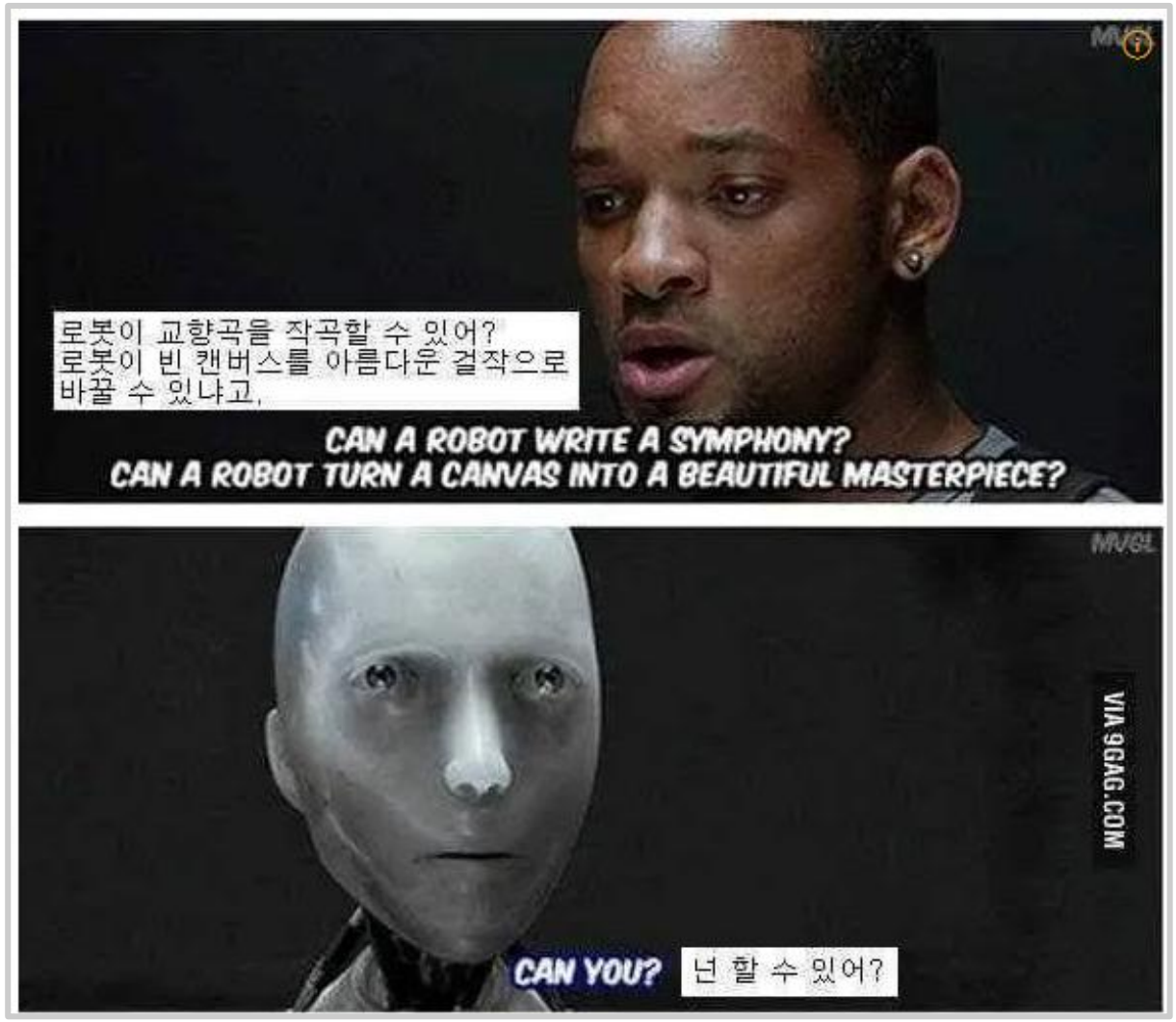
Session Break

AI TRAINING SERIES

AI

What is Artificial Intelligence? How Does AI Work?

AI 란 무엇일까?

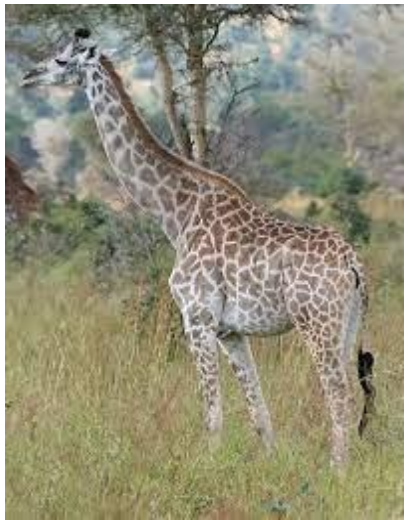

로봇이 교향곡을 작곡할 수 있어?
로봇이 빈 캔버스를 아름다운 걸작으로 바꿀 수 있냐고,
CAN A ROBOT WRITE A SYMPHONY?
CAN A ROBOT TURN A CANVAS INTO A BEAUTIFUL MASTERPIECE?
CAN YOU?
넌 할 수 있어?
VIA 9GAG.COM

자연현상에서 출발

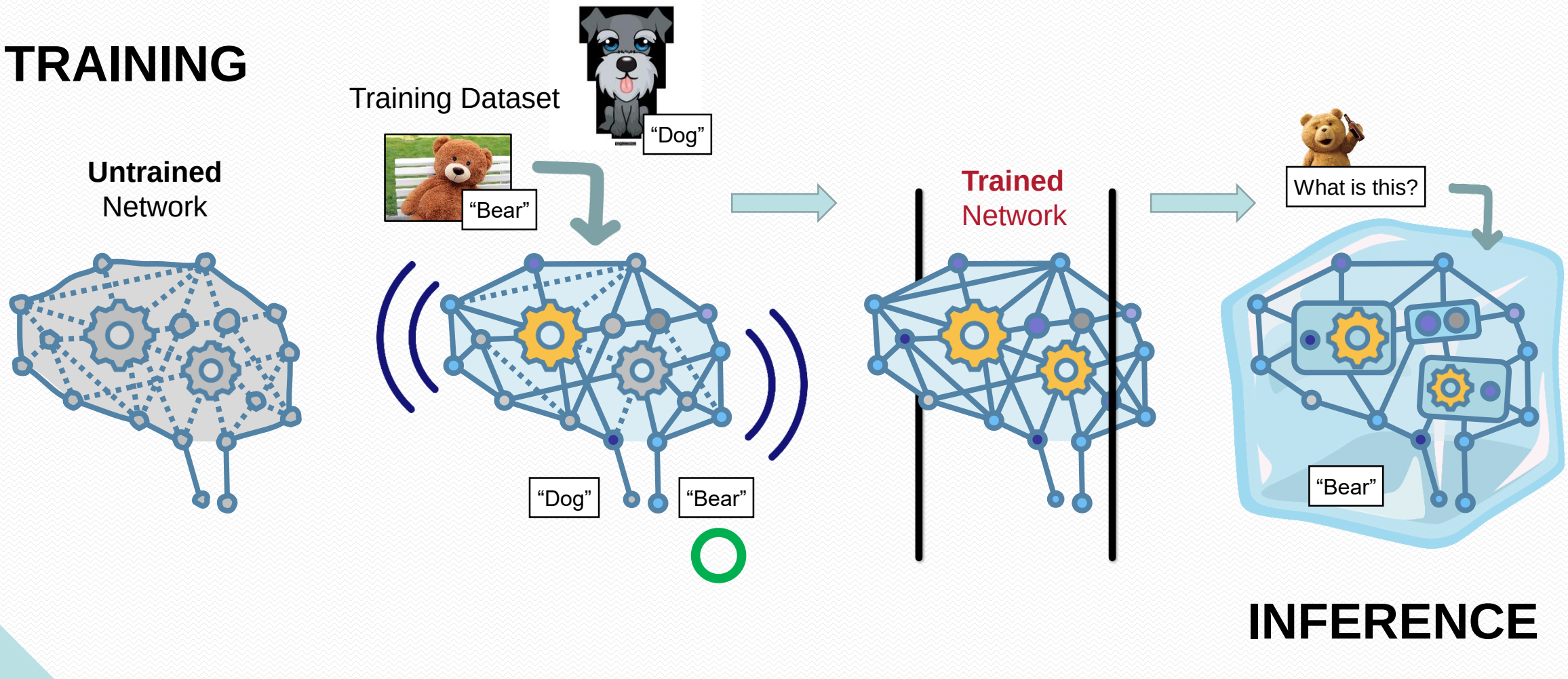
Sidewinder

Ultrasound

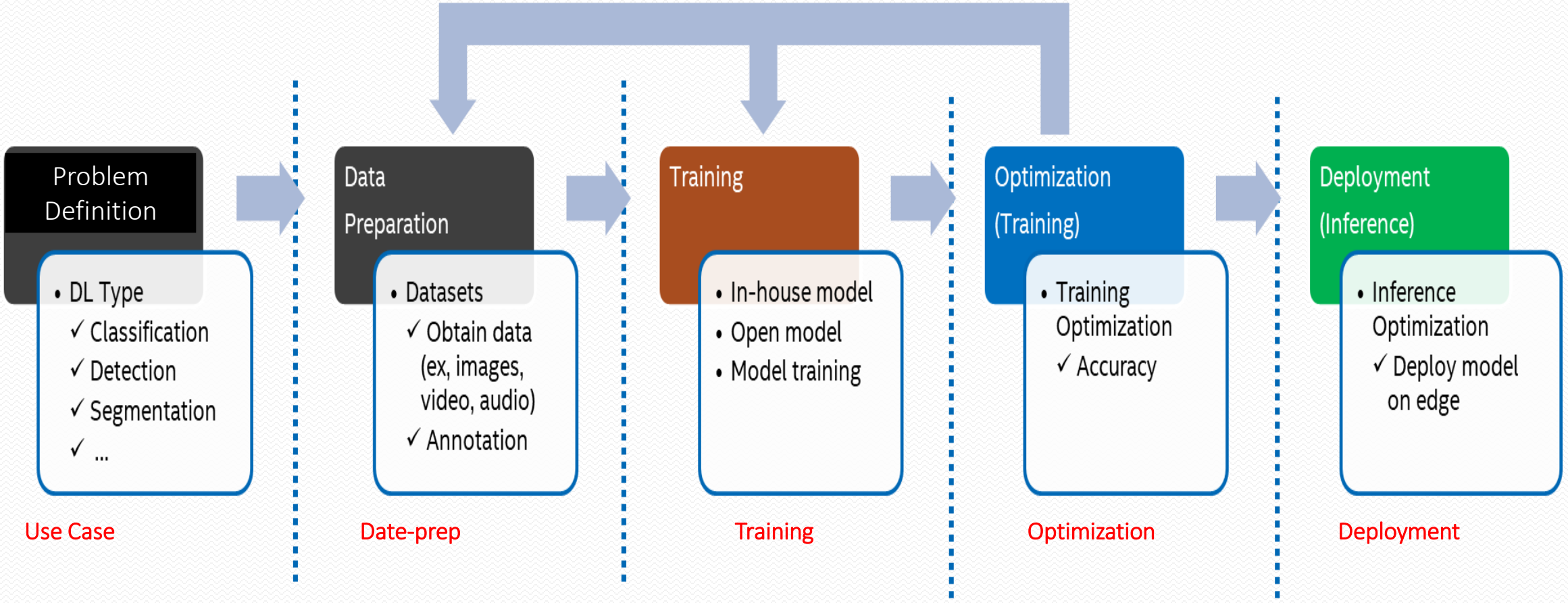
우리는 어떻게 생각했나



Deep Learning

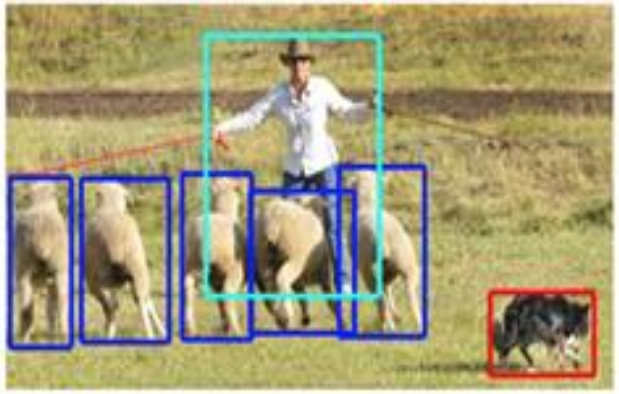


Deep Learning Workflow (생각하게 만들기)



1) Problem Definition

CLASSIFICATION



DETECTION

SEGMENTATION

NEURAL STYLE TRANSFER

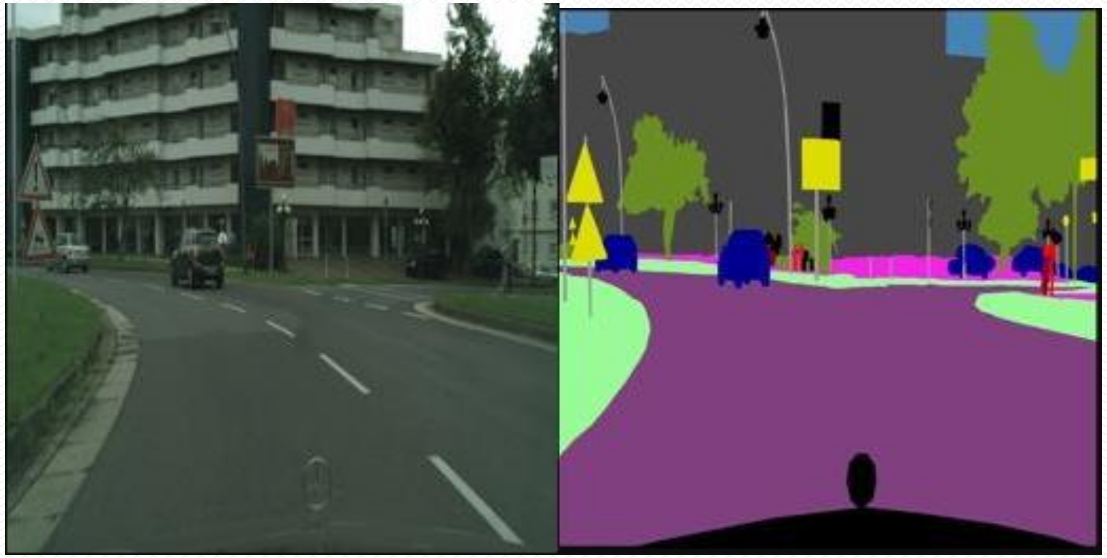
IMAGE CAPTION

COLORIZATION

2) Dataset preparation/annotation

- Cityscape/Camvid

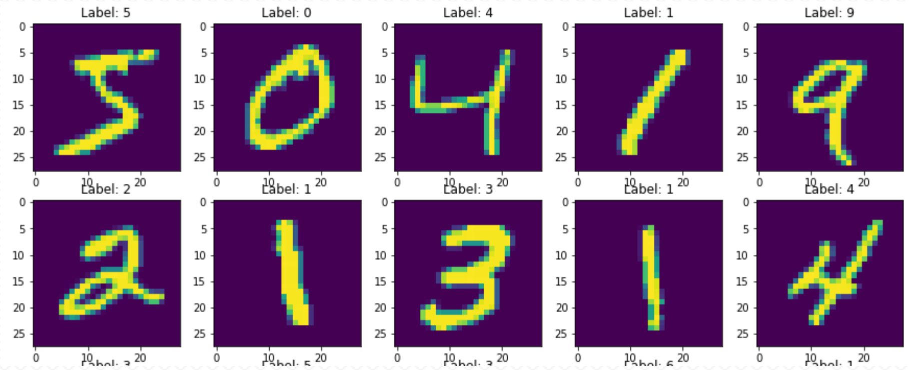
  Semantic Segmentation



- MNIST

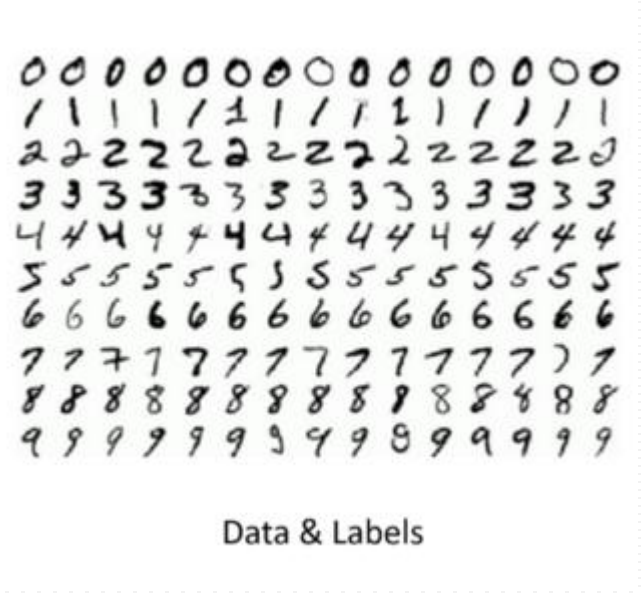
  Hand-written digit

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```



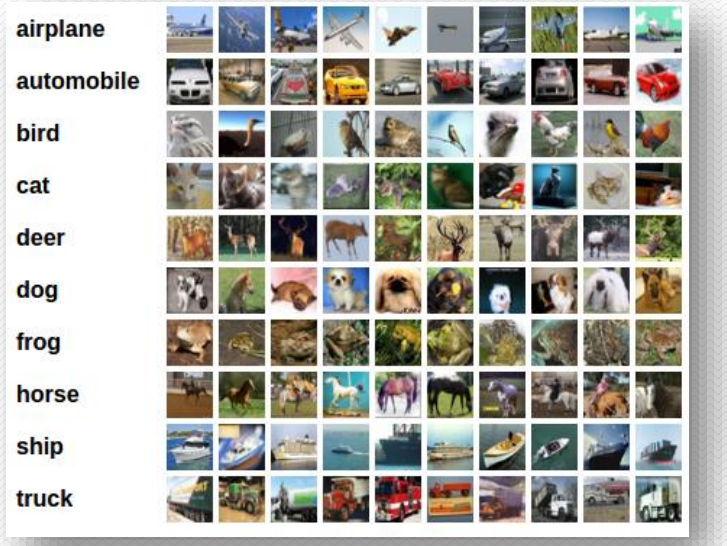
2) Dataset preparation/annotation – cont'd

- MNIST
  - Label : digit 0-9
  - Consists of 60,000 training image & 10,000 test image 28x28 gray scale image



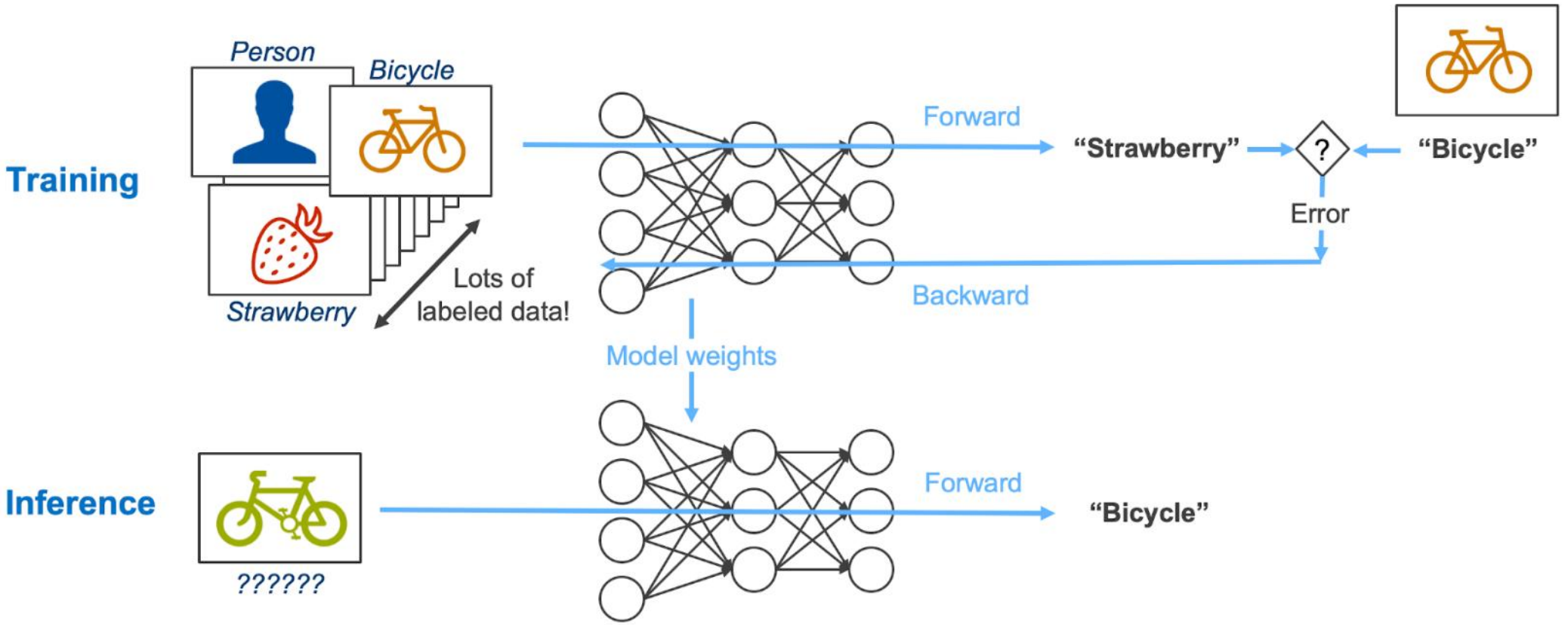
Data & Labels

- CIFAR-10 dataset
  - Label : 10 classes
  - Consists of 60,000 (32x32) color images, with 6000 images per class.

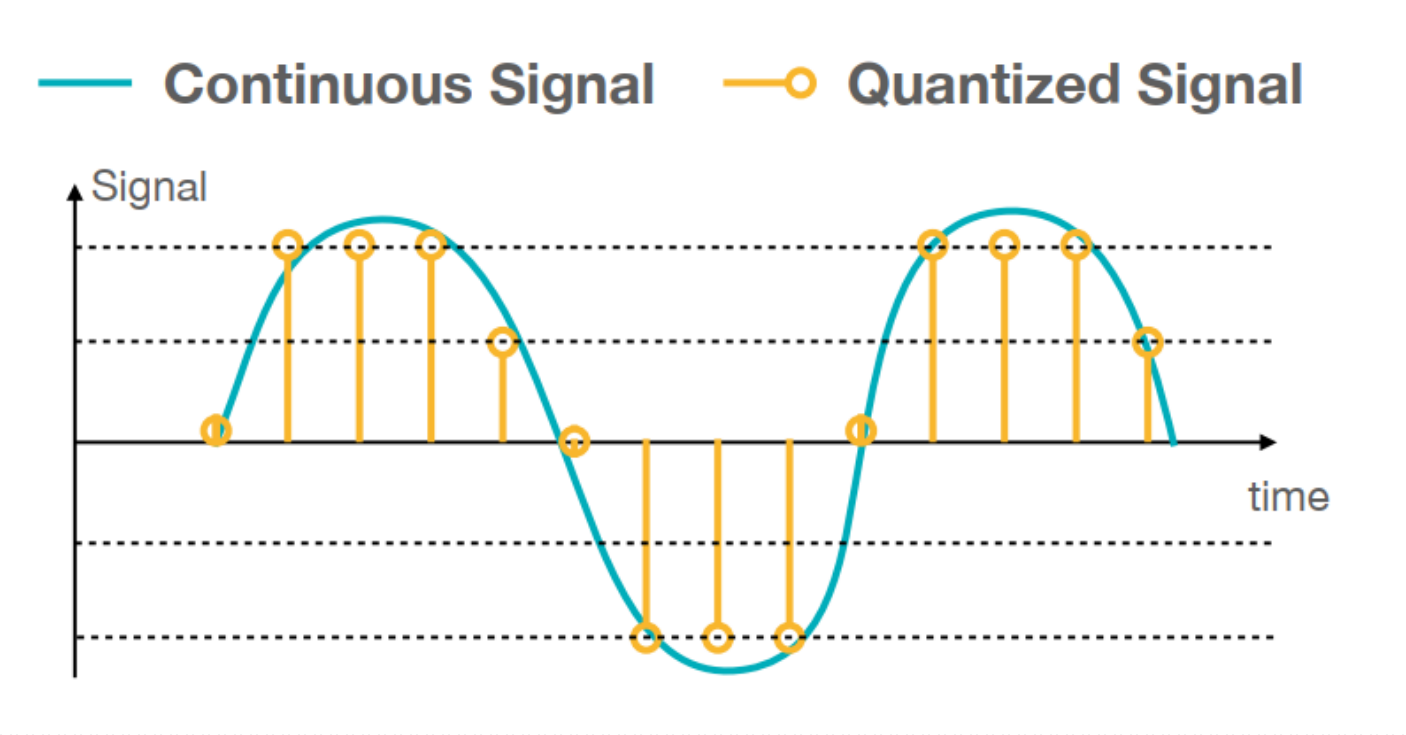


- IMAGENET
  - Label : 1000 classes
  - Consists of 800,000 (256x256 or 224x224) color Images, with 800 images per class.

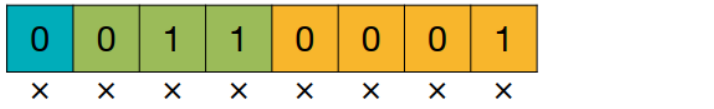
3) Model selection & Training



4) Optimization(Quantization)



***fixed-point Number**



***floating-point Number**

Sign 8 bit Exponent **23 bit Fraction** (significant / mantissa)

4) Optimization (Quantization)

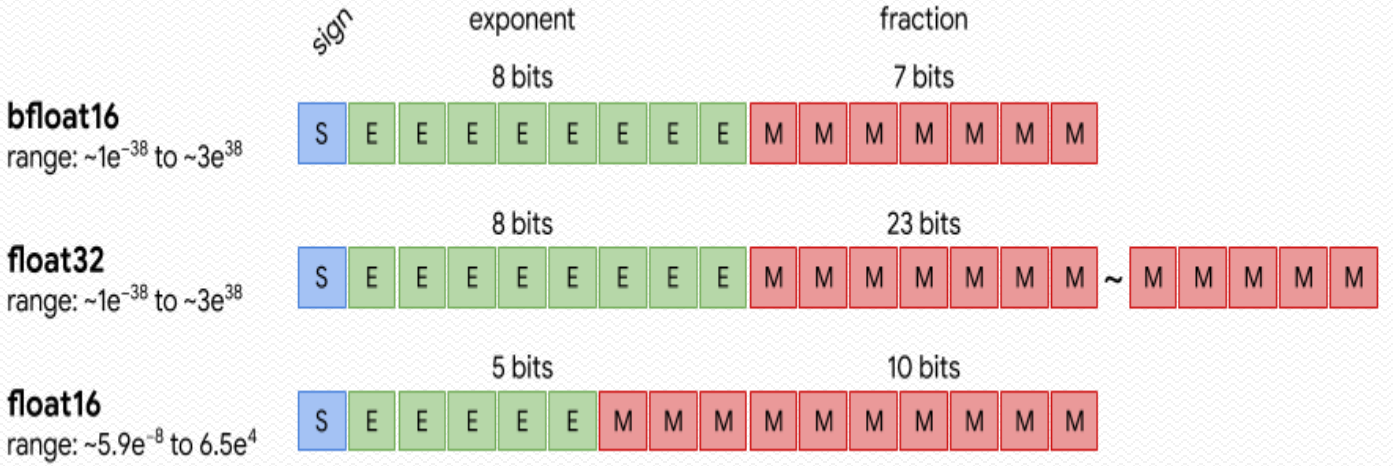
FP16 VS. FP32

FP16 improves speed (TFLOPS) and performance

FP16 reduces memory usage of a neural network

FP16 data transfers are faster than FP32

But converting from/to 32-bit floating-point can reduce accuracy



INT8 QUANTIZATION

Accelerate the performance of certain models

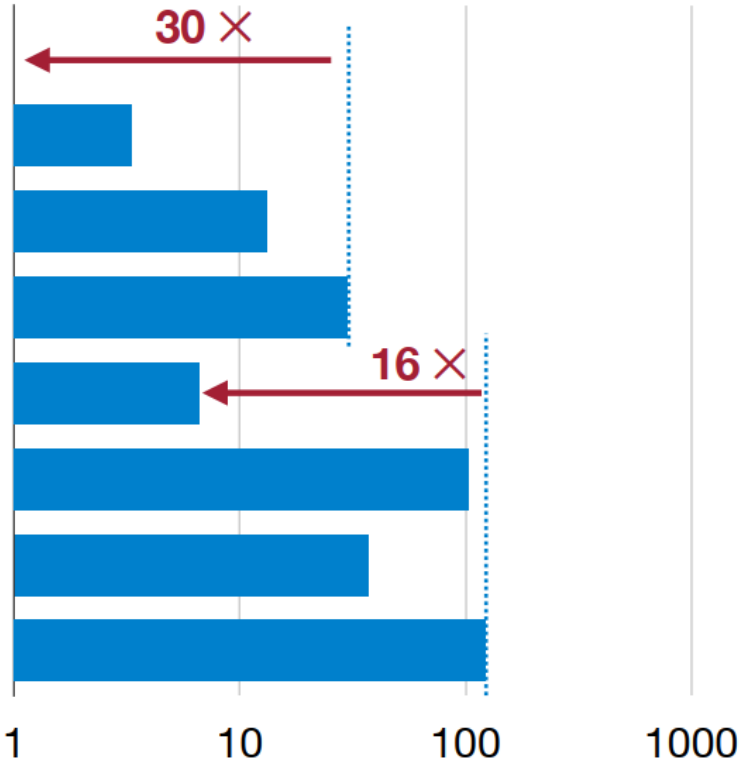
However, this comes at the cost of a small reduction in accuracy

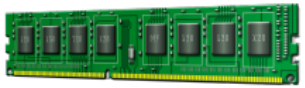
4) Optimization(Quantization)

Less bit → Less energy!!!

Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

Rough Energy Cost For Various Operations in 45nm 0.9V

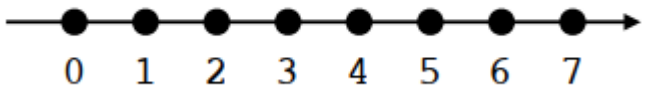


1  = 200 × +

4) Optimization(FP4 AND INT4)

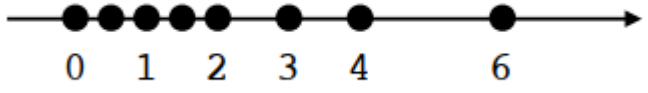
INT4

S				
0	0	0	1	=1
0	1	1	1	=7



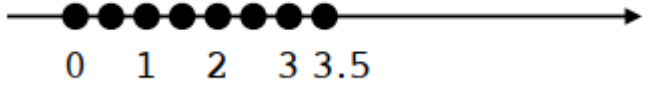
FP4 (E2M1)

S	E	E	M
0	0	0	1
0	1	1	1



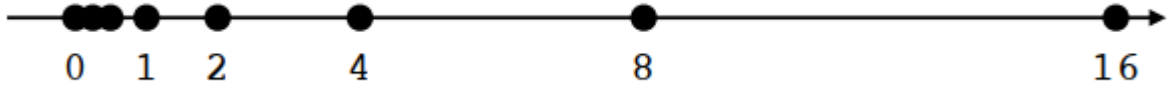
FP4 (E1M2)

S	E	M	M
0	0	0	1
0	1	1	1

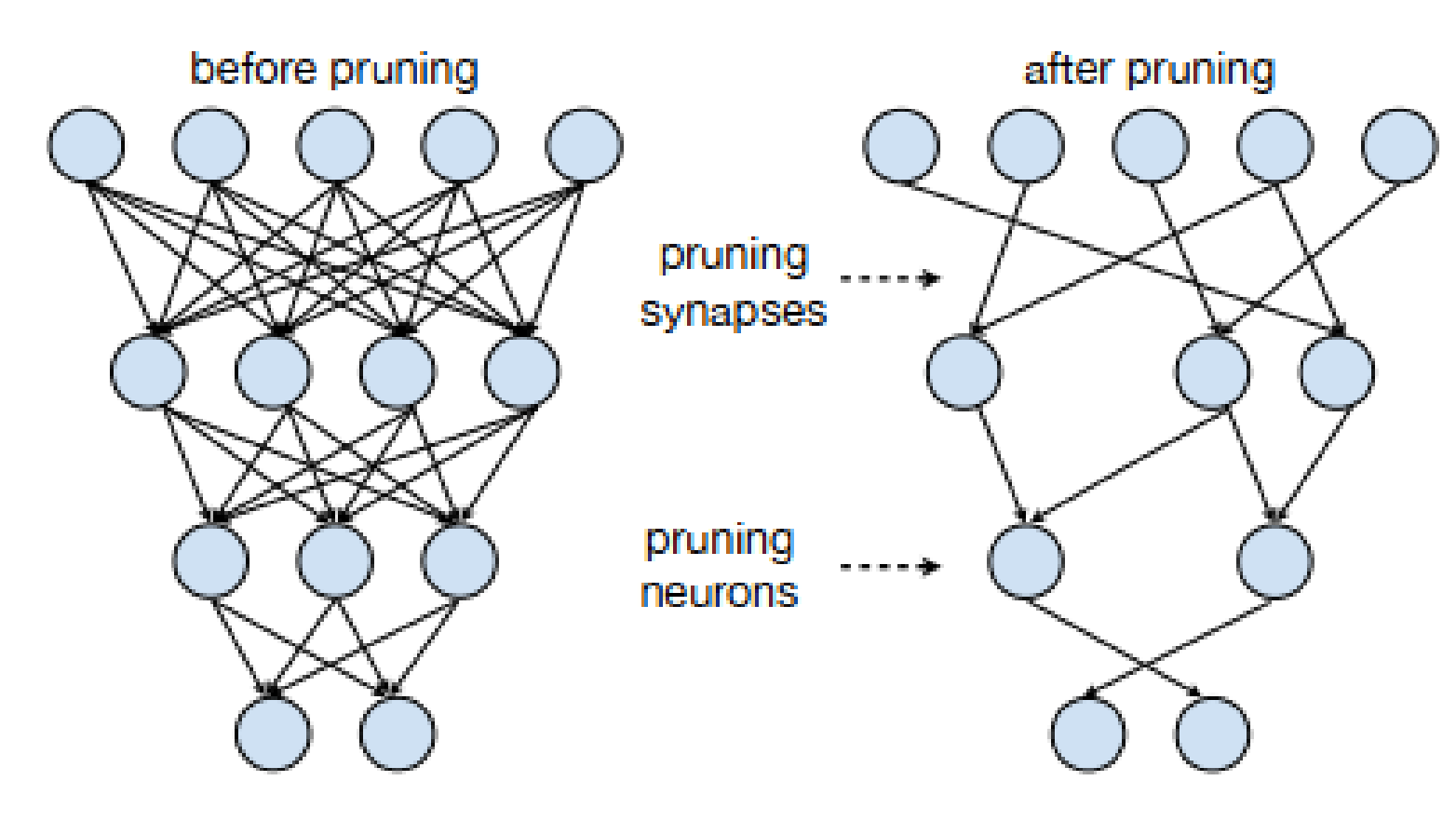


FP4 (E3M0)

S	E	E	E
0	0	0	1
0	1	1	1



4) Optimization(Pruning)

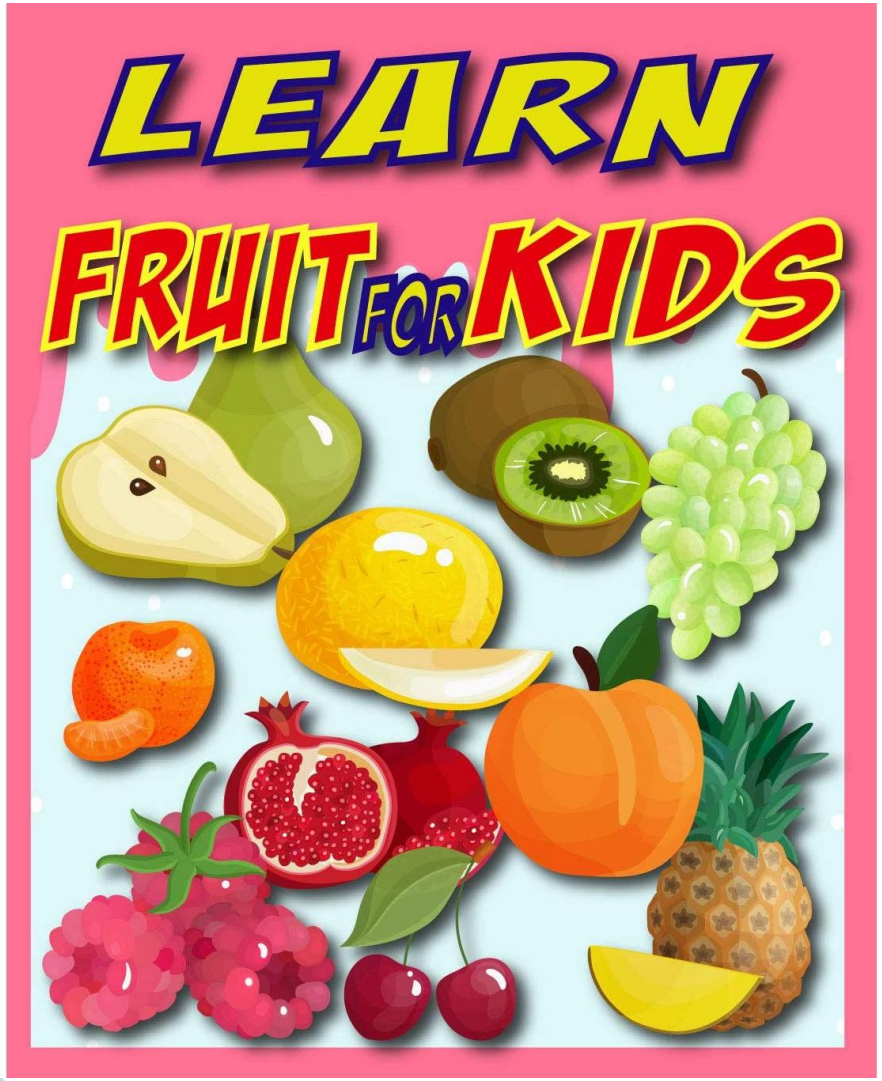


4) Optimization

사과!

배

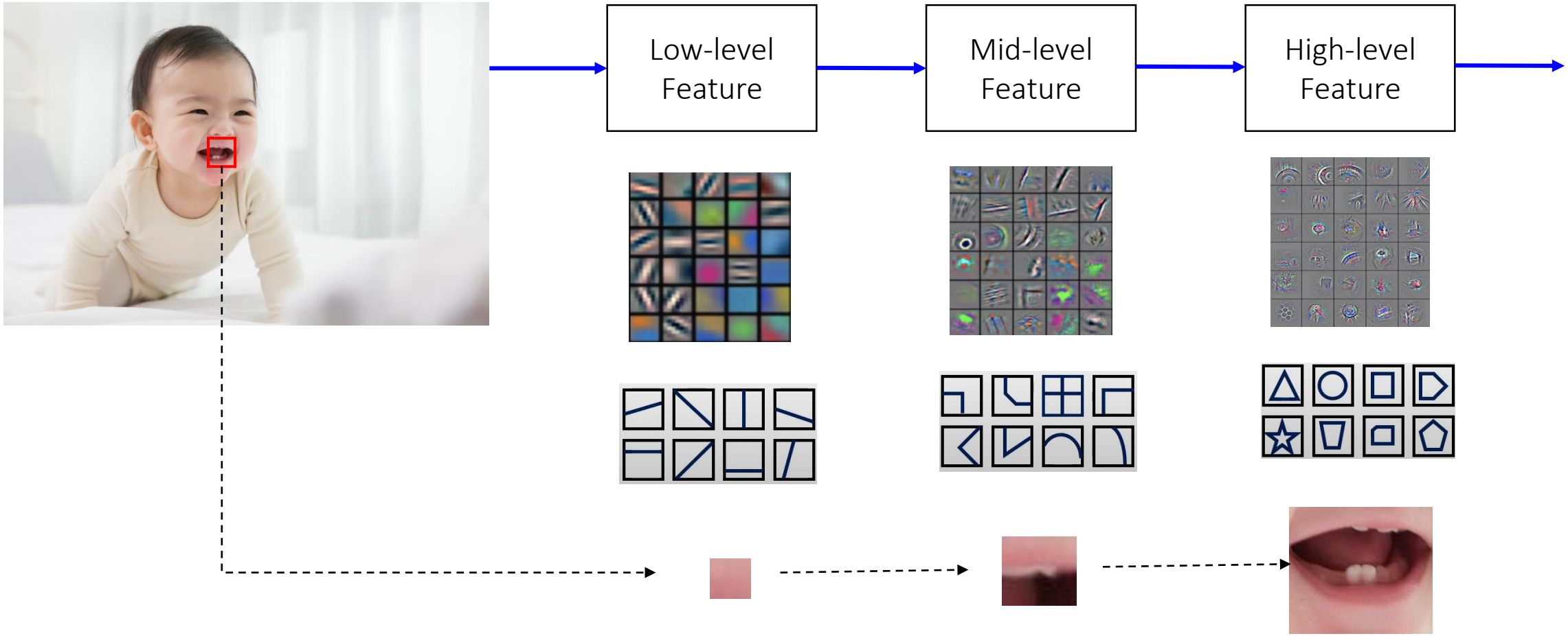
4) Optimization



Apple!

??Pear???

4) Optimization : transfer learning



Session Break

AI TRAINING SERIES

History

How DL and AI have evolved?

인공지능

artificial intelligence

초기 인공지능이 흥미를
불러일으킴

사고나 학습 등 인간이 가진 지적 능력을 컴퓨터를 통해 구현하려고 함

machine learning

머신러닝이 번성하기 시작

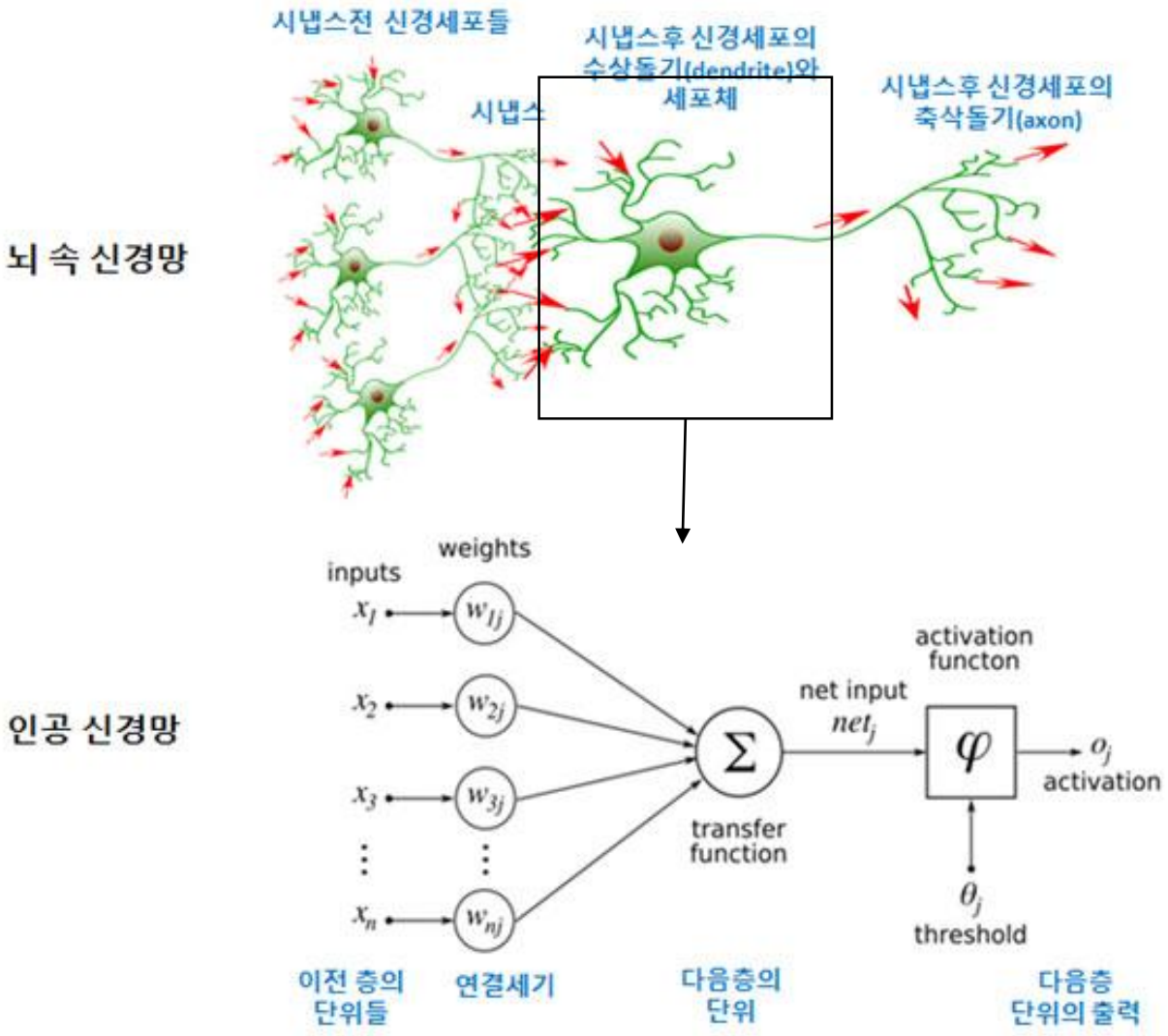
컴퓨터 강화학습을 통해 원하는 성능을
향상시키는 방법

deep learning

딥러닝 혁신이 AI붐을 일으키며,
인간을 뛰어넘는 성능을
나타내기 시작

1950's 1960's 1970's 1980's 1990's 2000's 2010's

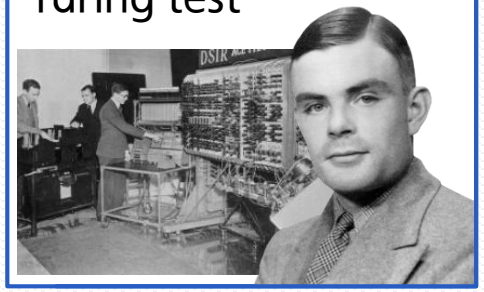
인공 신경망



http://scienceon.hani.co.kr/397536

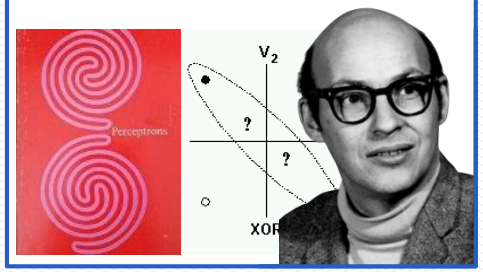
HISTORY OF A.I.

Alan Turing
Turing test



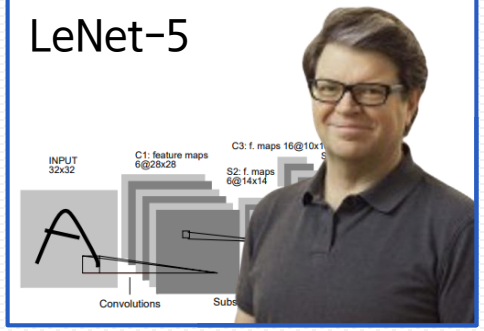
1950년

Marvin Minsky
a book of 'Perceptrons'



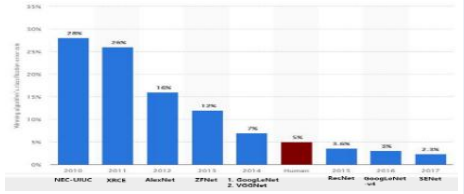
1969년

Yann Lecun
LeNet-5



1998년

ResNet
Accuracy is higher than human



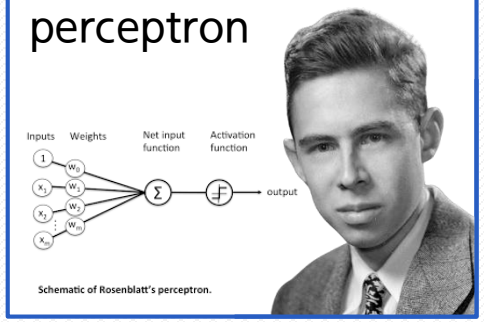
2015년

AI winter

AI winter

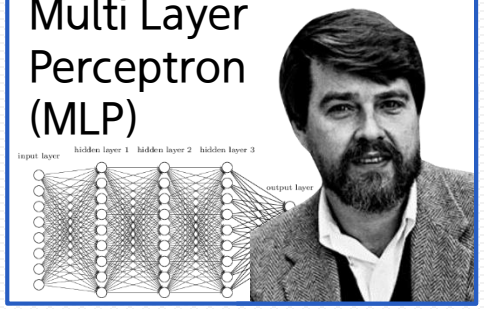
1958년

Frank Rosenblatt
perceptron



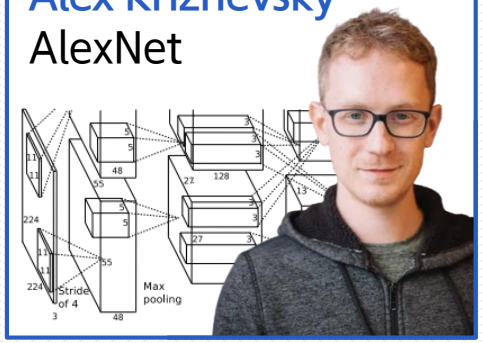
1986년

David Rumelhart
Multi Layer Perceptron (MLP)



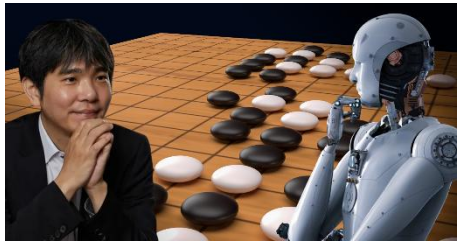
2012년

Alex Krizhevsky
AlexNet



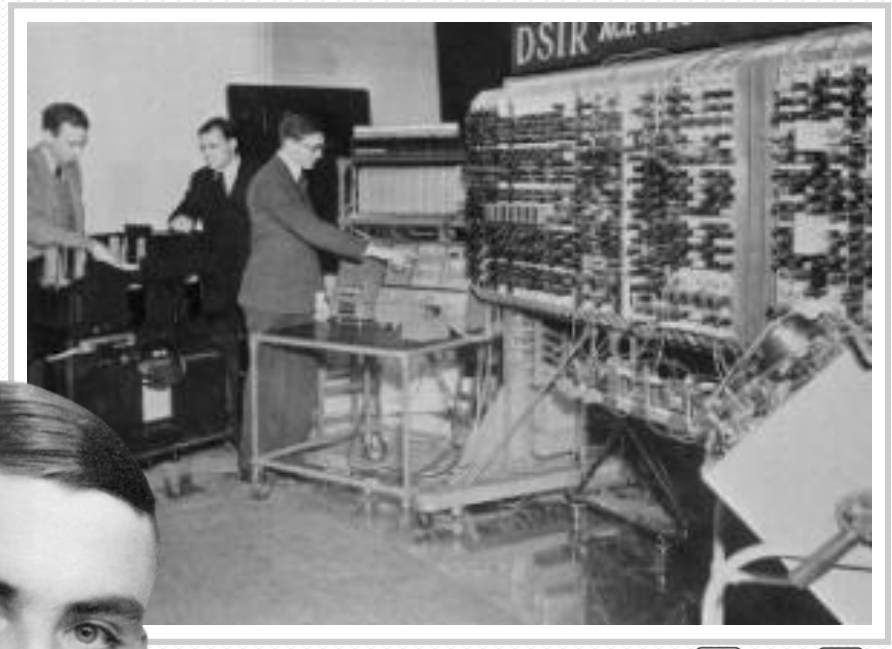
2016년

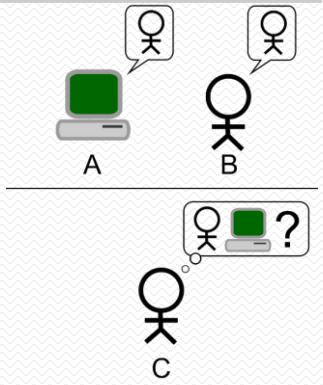
Google Brain
AlphaGo



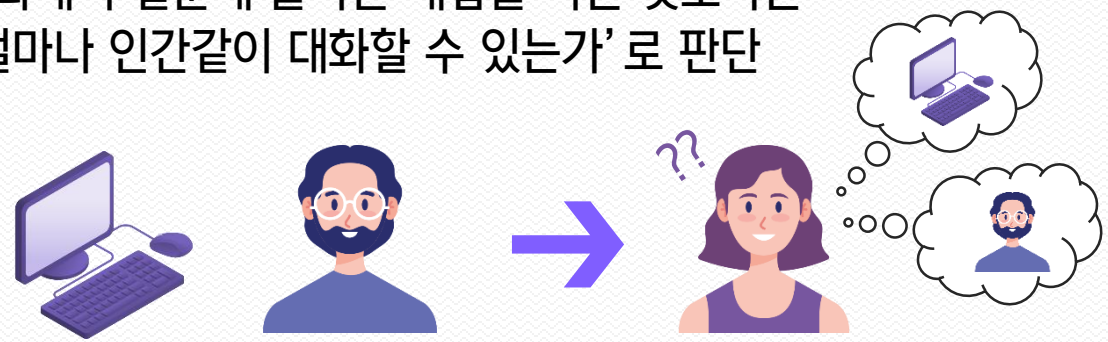
HISTORY OF A.I.

1950년 Alan Turing ‘Turing test’





- 튜링 테스트(Turing test)의 의미
  - 기계가 생각하는 것이 가능한가?
  - computing machinery and intelligence 논문
- 각각 분리된 채, 하나는 인간, 다른 하나는 기계와 대화를 하는 test
- 5분 동안 기계와 대화한다는 것을 인지하지 못하면 성공
- 대화에서 질문에 올바른 대답을 하는 것보다는 ‘얼마나 인간같이 대화할 수 있는가’로 판단



HISTORY OF A.I.

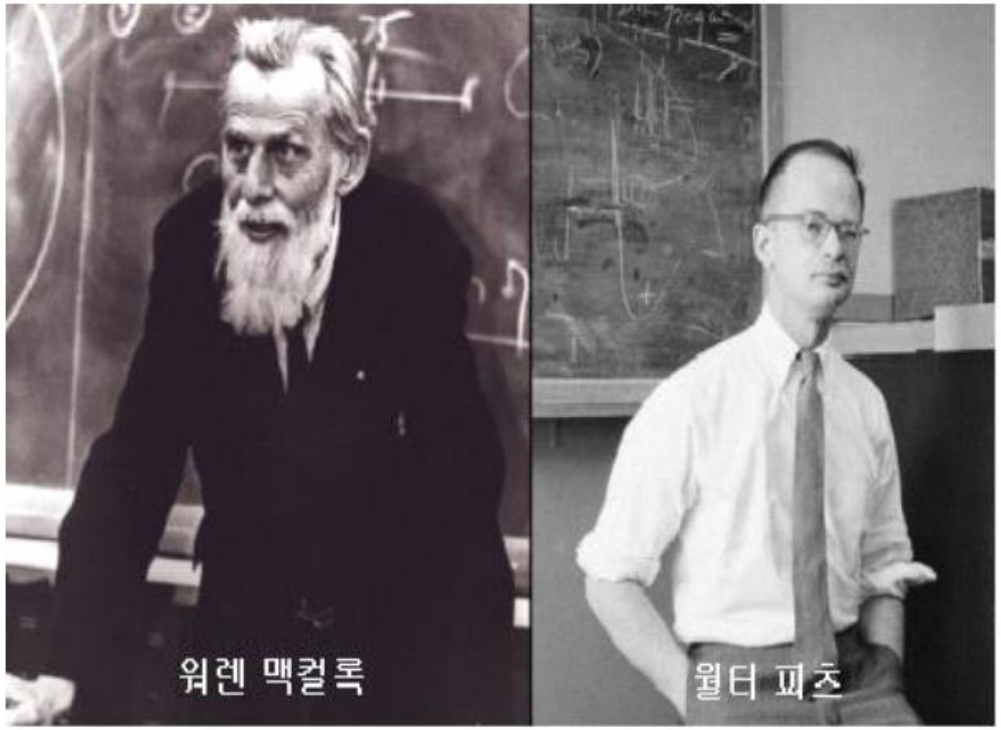
인공지능(AI) 최초 사용 (존 메카시, John McCarthy)

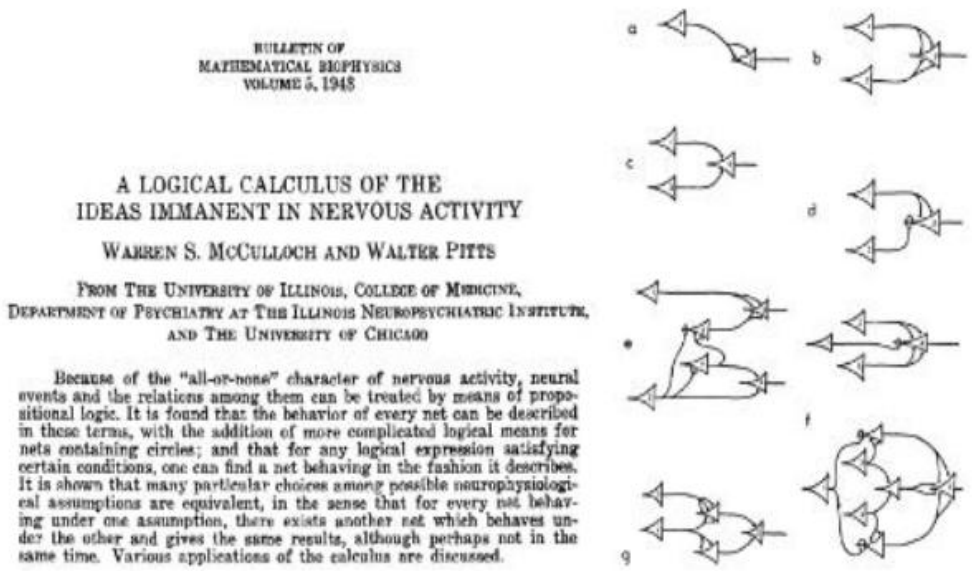
논문 “지능이 있는 기계를 만들기 위한 과학과 공학” (1955)

“인공지능(AI)” 이란 단어를 최초 사용함.

HISTORY OF A.I.

인공신경망의 시초 (월터 피츠, Walter Harry Pitts Jr.)





BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

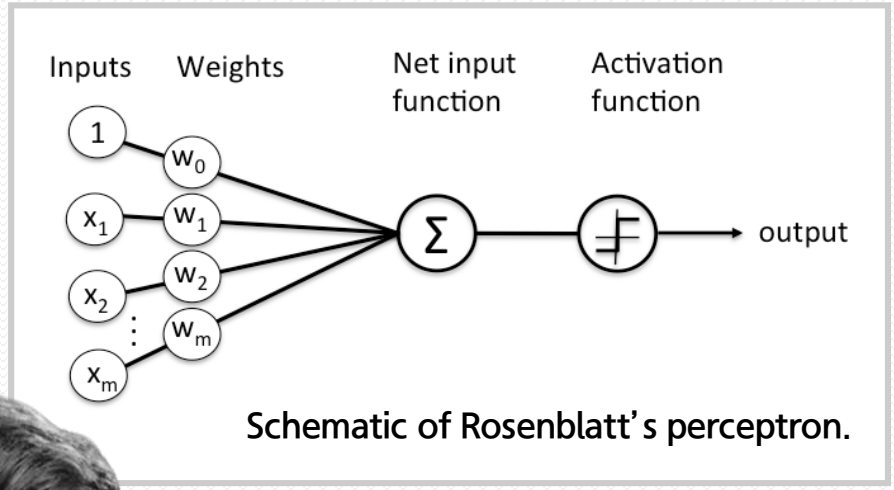
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

논문 <신경 활동에 있어서 관념의 논리적 미적분>과 명제논리로 나타낸 맥컬록-피츠 모델

- 인간 뉴런의 작동 원리를 최초로 수학적으로 모델링
- 인공신경망, 기계학습, 딥러닝 연구의 기원

HISTORY OF A.I.

1958년 Frank Rosenblatt ‘perceptron’



Schematic of Rosenblatt’s perceptron.

- 프랭크 로젠블렛은 신경 생물 과학자로, 심리학 전공을 했으며, 이미지 인식에 많은 투자와 관심을 가져 성공함
  - Feed Forward Network 발표 논문
    - » *THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN*

〈출처21, 22〉

- **If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental question.**

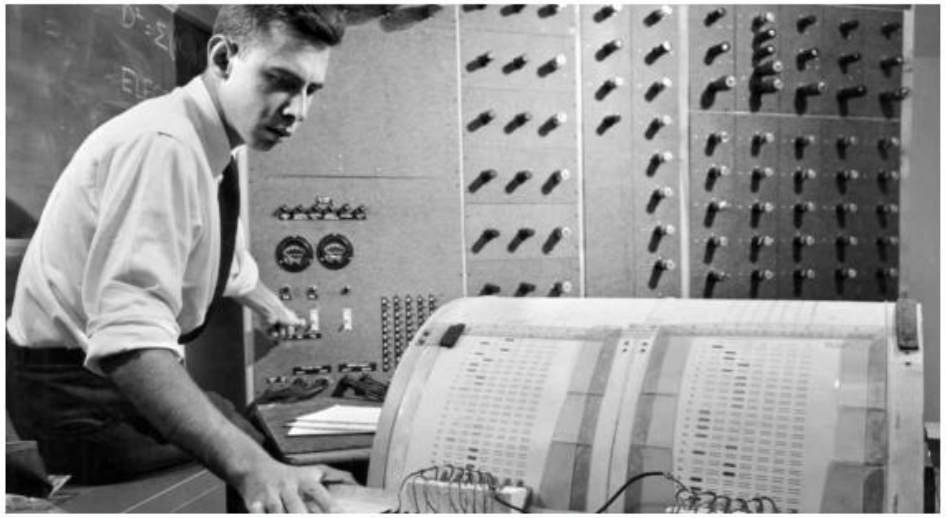
01 How is information about the physical world sensed, or detected, by the biological system?

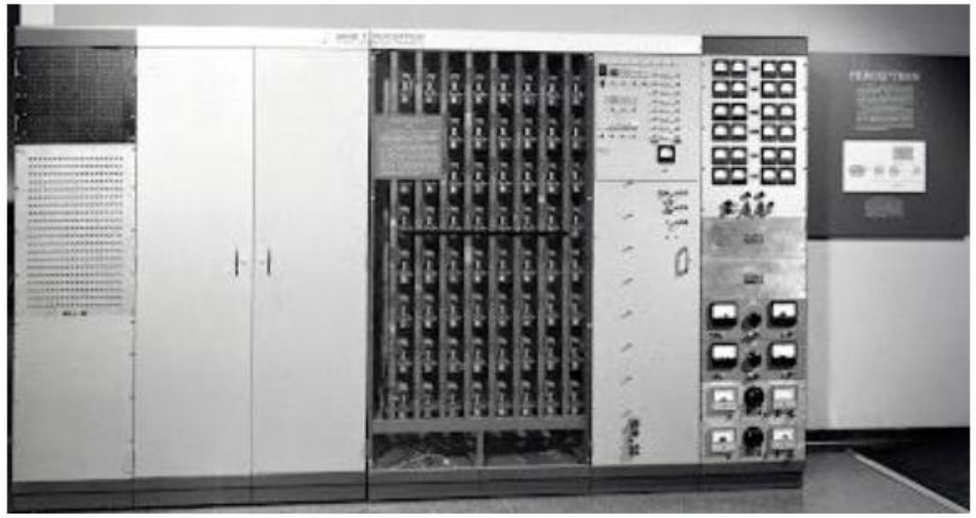
02 In what form is information stored, or remembered?

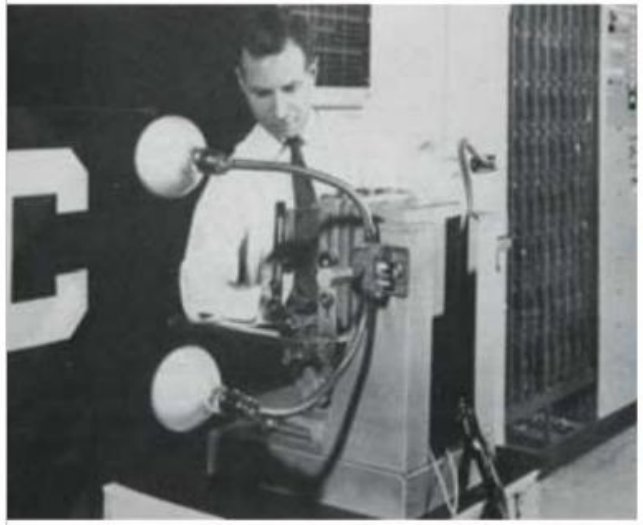
03 How does information contained in storage, or in memory, influence recognition and behavior?

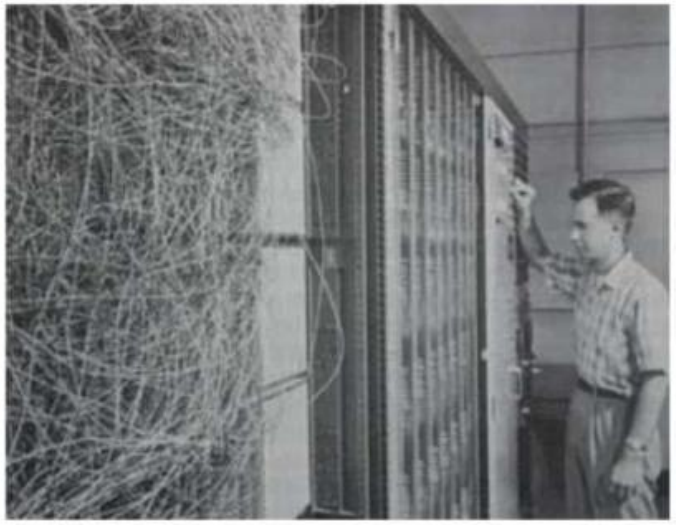
HISTORY OF A.I.

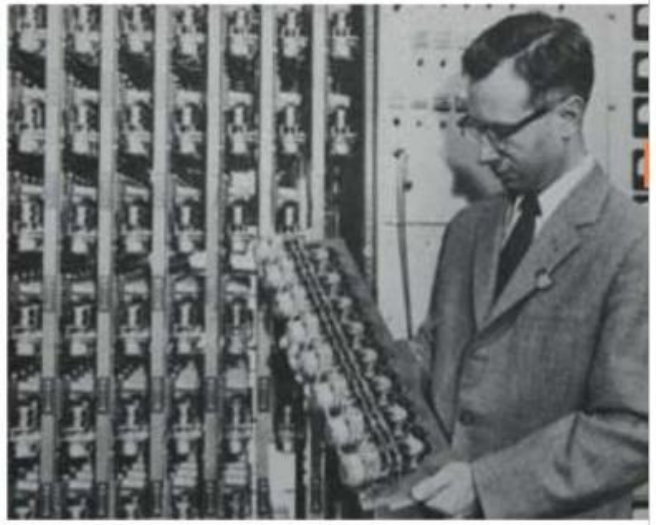
1958년 Frank Rosenblatt ‘perceptron’





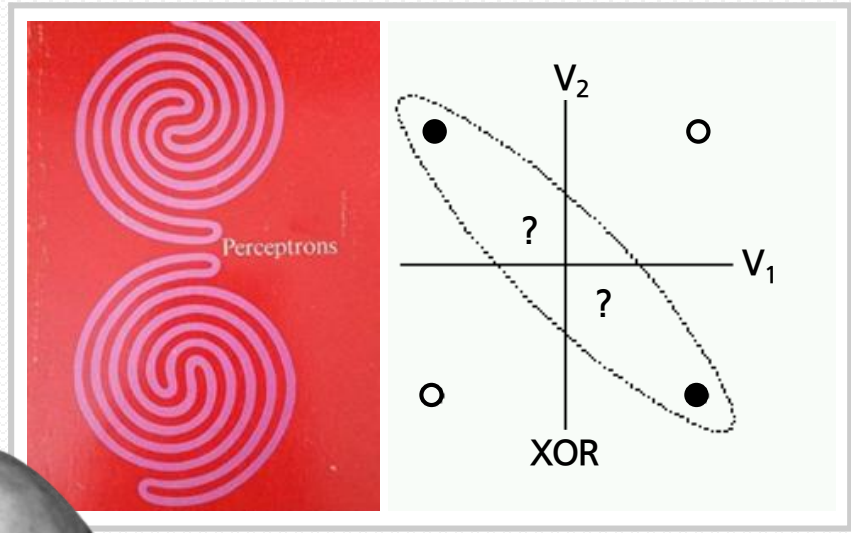




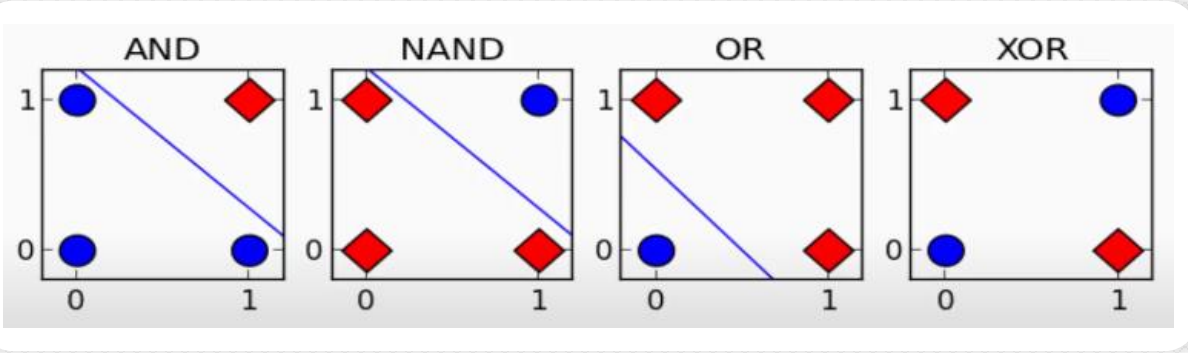


HISTORY OF A.I.

1969년 **Marvin Minsky** ‘A book of ‘Perceptrons’’

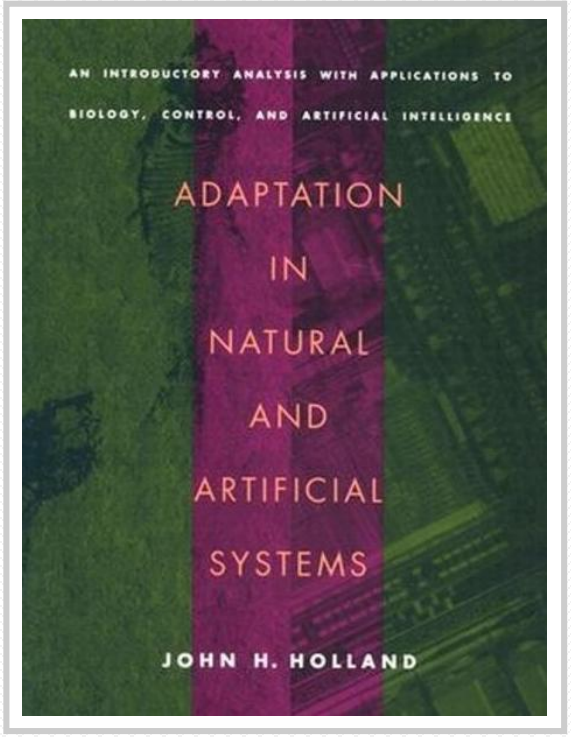


- *Perceptrons*
  - Minsky & Papert가 출간함
  - 퍼셉트론 모델에 대해 철저한 분석 및 그 한계에 대해서도 논리정연하게 분석함



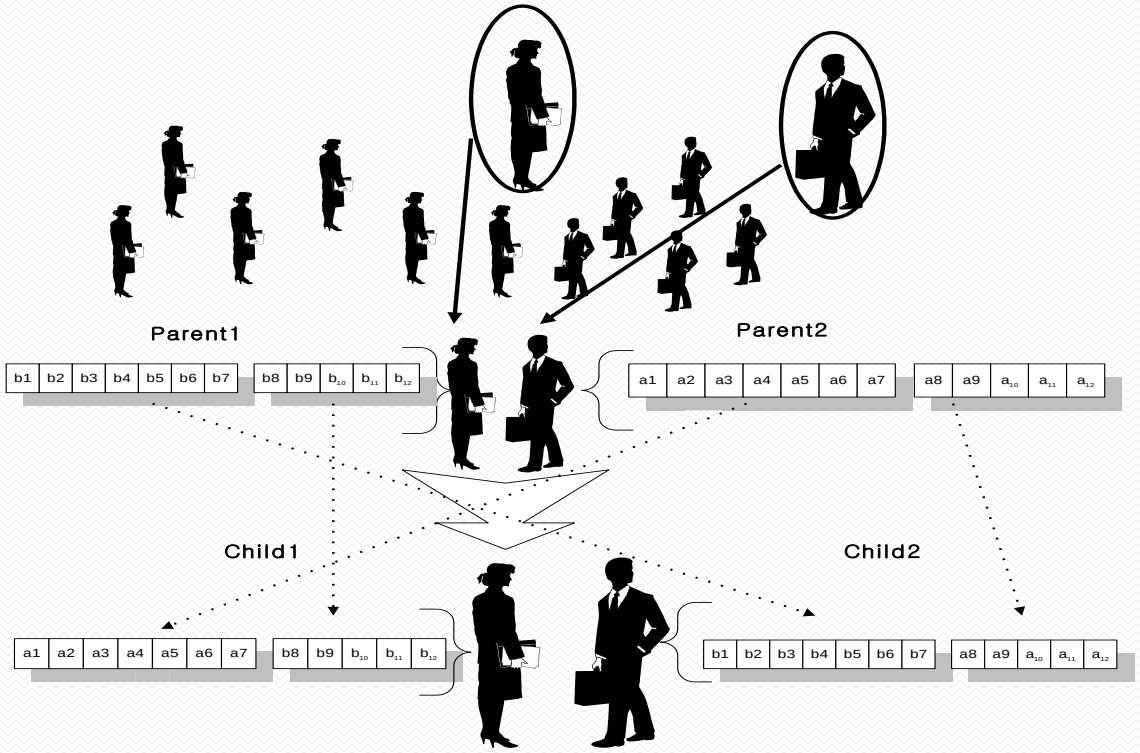
HISTORY OF A.I.

1975년 John Holland ‘A book of ‘Adaptation’’



〈출처25〉

- 저서 ‘Adaptation in Natural and Artificial Systems’를 통해 제안함
- 생물의 진화를 모방한 진화 연산의 대표적인 기법



HISTORY OF A.I.

베이지안 이론

- 두 종류의 조건부 확률 사이의 관계를 정의함
- 다양한 분야에서 활용되는 대표적인 기법

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}$$

likelihood　class prior probability

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

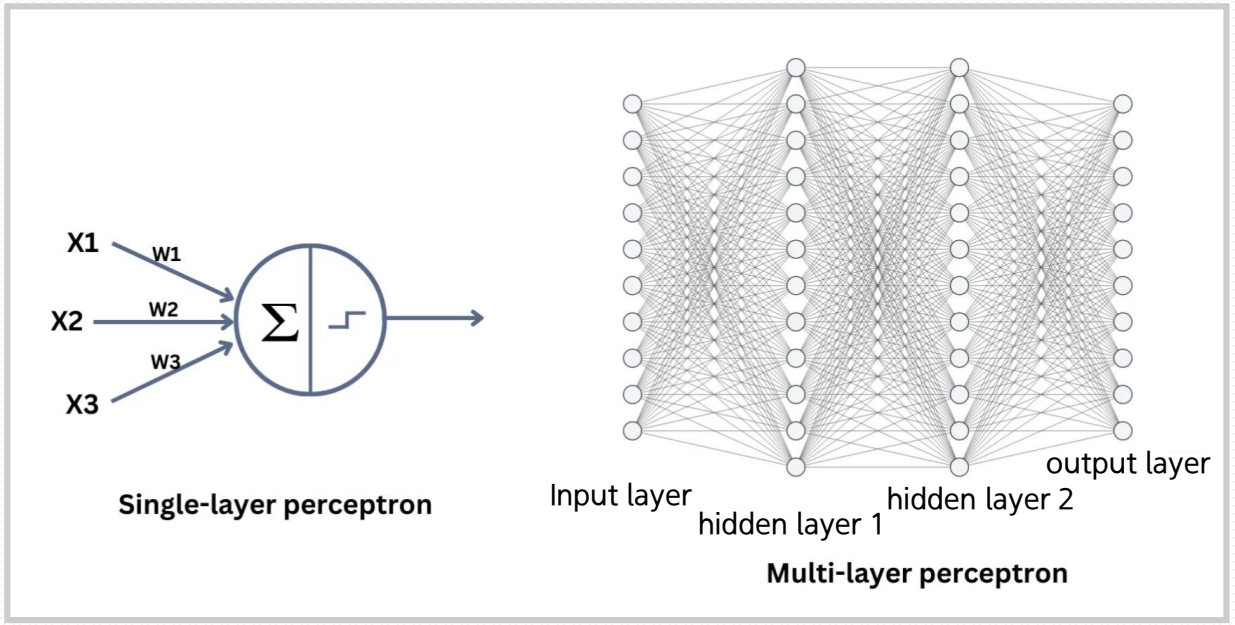
posterior probability　predictor prior probability

- $P(A)$, 사전확률(prior probability)
  : 결과가 나타나기 전에 결정되어 있는 A(원인)의 확률
- $P(B|A)$, 우도확률(likelihood probability)
  : A(원인)가 발생하였다는 조건하에서 B(결과)가 발생할 확률
- $P(A|B)$, 사후확률(posterior probability)
  : B(결과)가 발생하였다는 조건하에서 A(원인)가 발생했을 확률

$$P(\text{질병}|\text{양성}) = \frac{P(\text{양성}|\text{질병})P(\text{질병})}{P(\text{양성})} = \frac{0.99 \times 0.01}{0.01 \times 0.99 + 0.99 \times 0.10} = 0.091$$

HISTORY OF A.I.

1986년 **David Rumelhart** ‘Multi Layer Perceptron(MLP)’



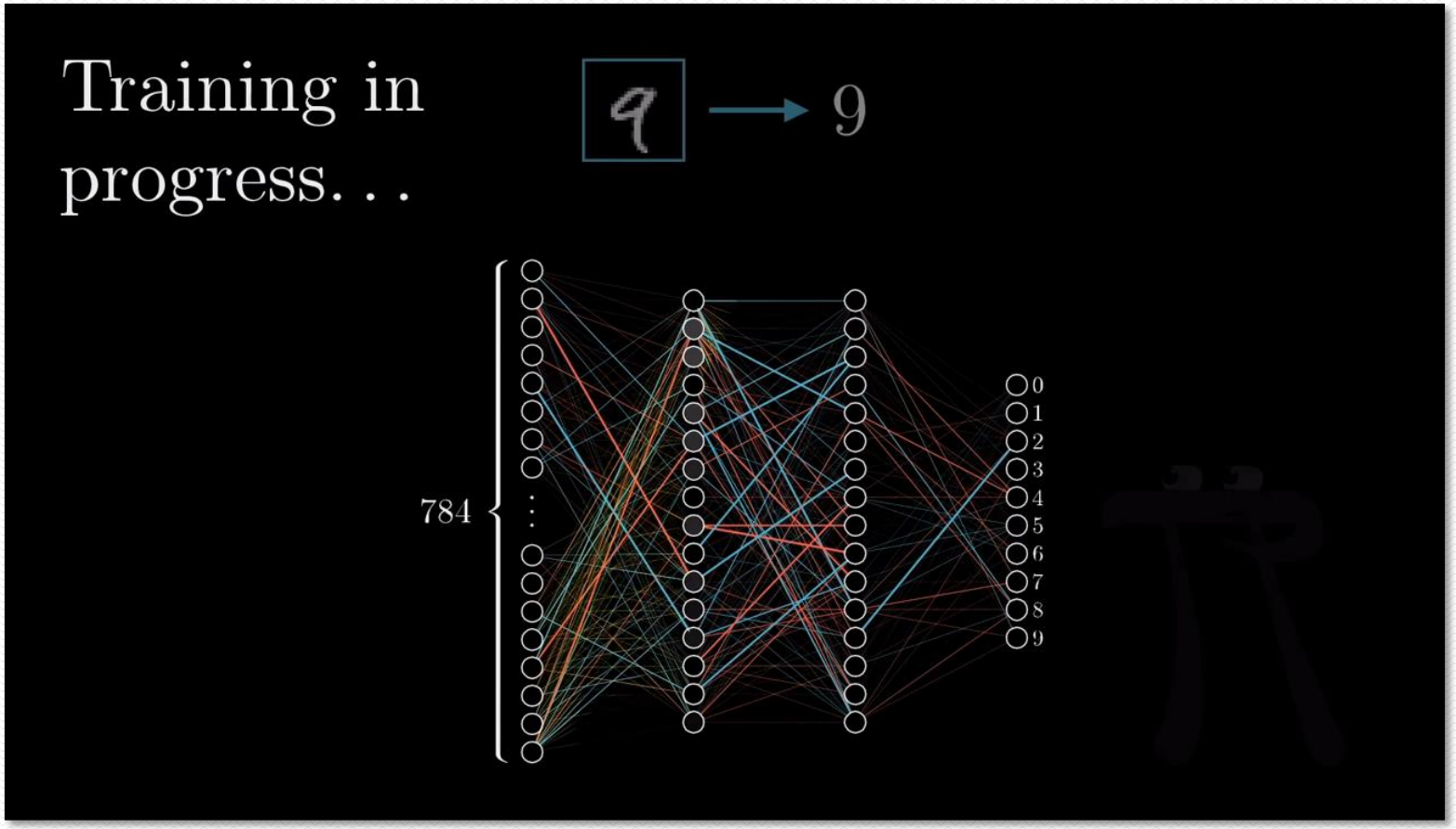
〈출처27〉

- Geoffrey Hinton, Ronald Williams와 함께 Back Propagation에 대한 논문을 통해 Multi Layer Perceptron을 Training을 할 수 있음을 언급함
- 다시 AI에 대한 열기를 가져옴

https://www.researchgate.net/figure/Deep-learning-backpropagation-in-a-nutshell-The-Economist-2016-Tim-Jones-2017_fig11_330382022

HISTORY OF A.I.

1986년 David Rumelhart ‘Multi Layer Perceptron(MLP)’

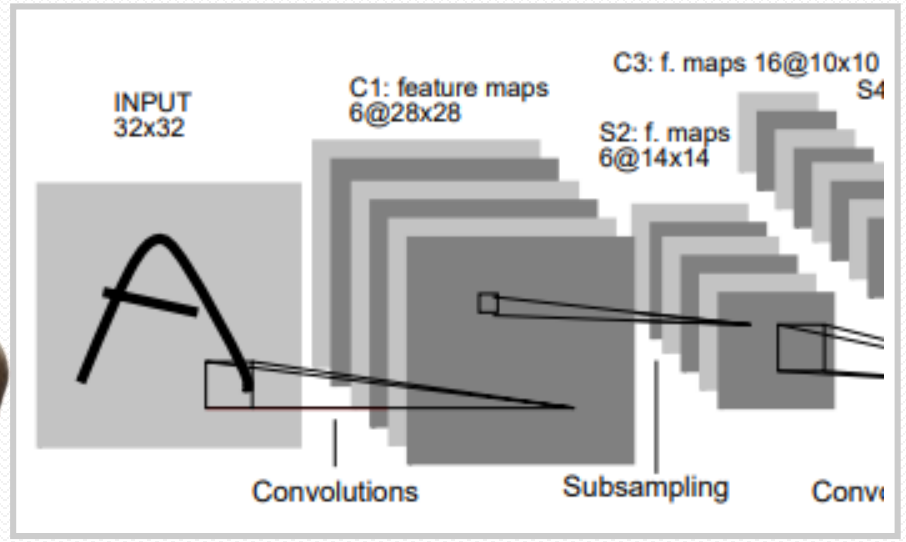


https://www.researchgate.net/figure/Deep-learning-backpropagation-in-a-nutshell-The-Economist-2016-Tim-Jones-2017_fig11_330382022

HISTORY OF A.I.

- 2번째 위기
  - Hidden layer가 깊어지면 back propagation이 제대로 안됨

1998년 Yann Lecun ‘LeNet-5’

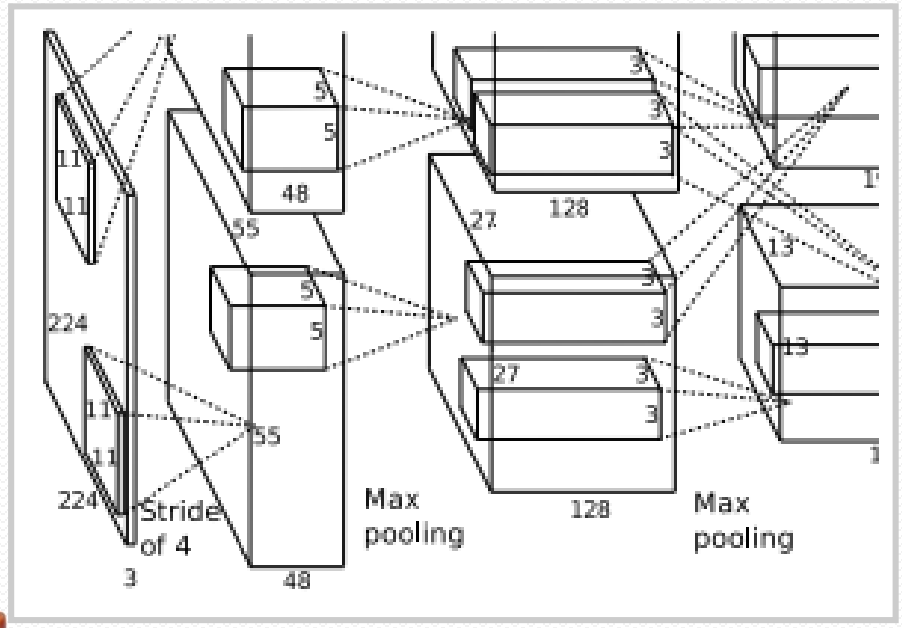


- Paper
  - Gradient-based learning applied to document recognition
- multilayer neural network의 한계를 극복하기 위해 CNN을 활용함
- MNIST Dataset

https://www.researchgate.net/figure/Deep-learning-backpropagation-in-a-nutshell-The-Economist-2016-Tim-Jones-2017_fig11_330382022

HISTORY OF A.I.

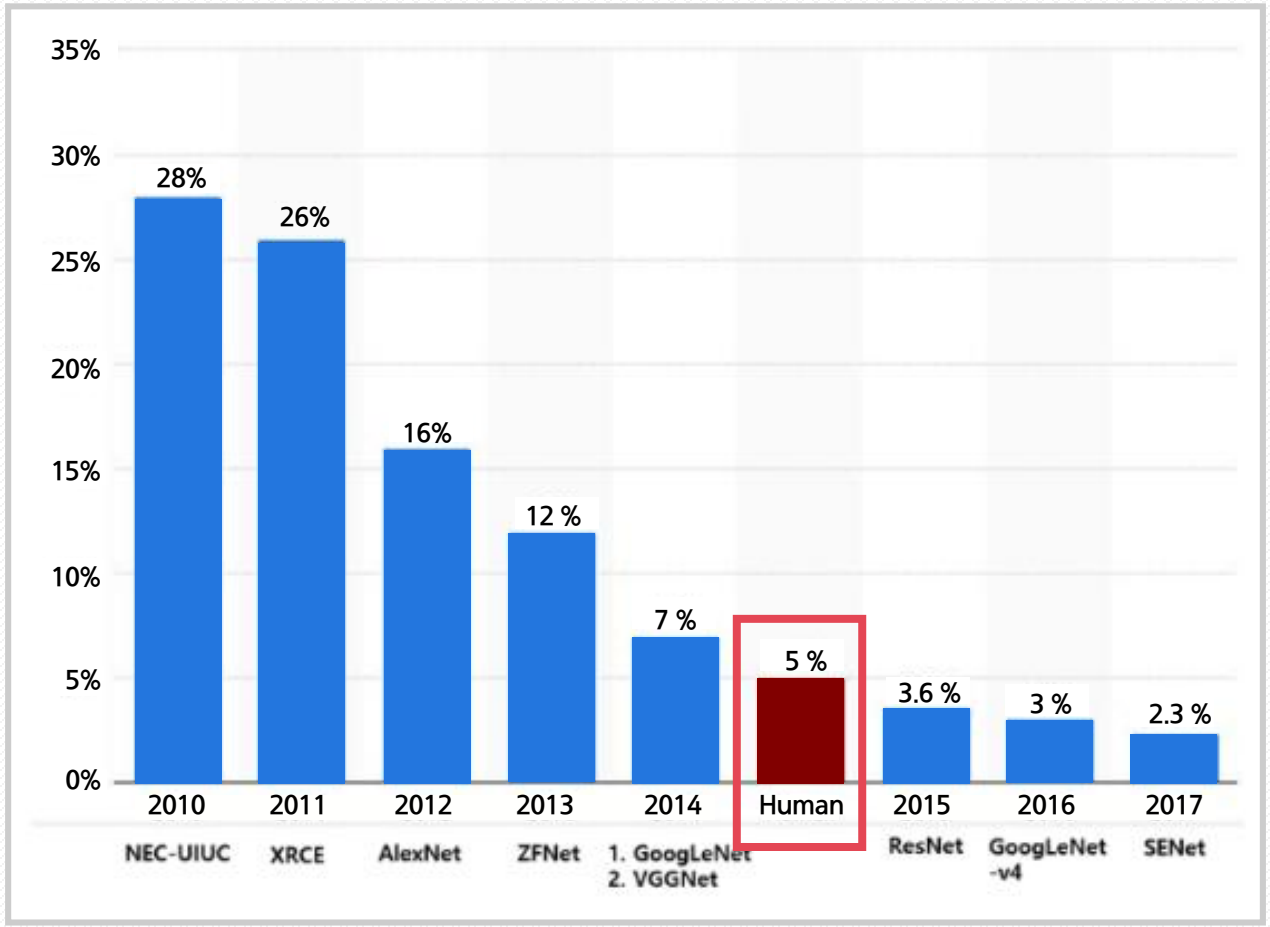
2012년 Alex Krizhevsky ‘AlexNet’



- Paper
  - ImageNet Classification with Deep Convolutional Neural Networks
- 2012년 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 우승

HISTORY OF A.I.

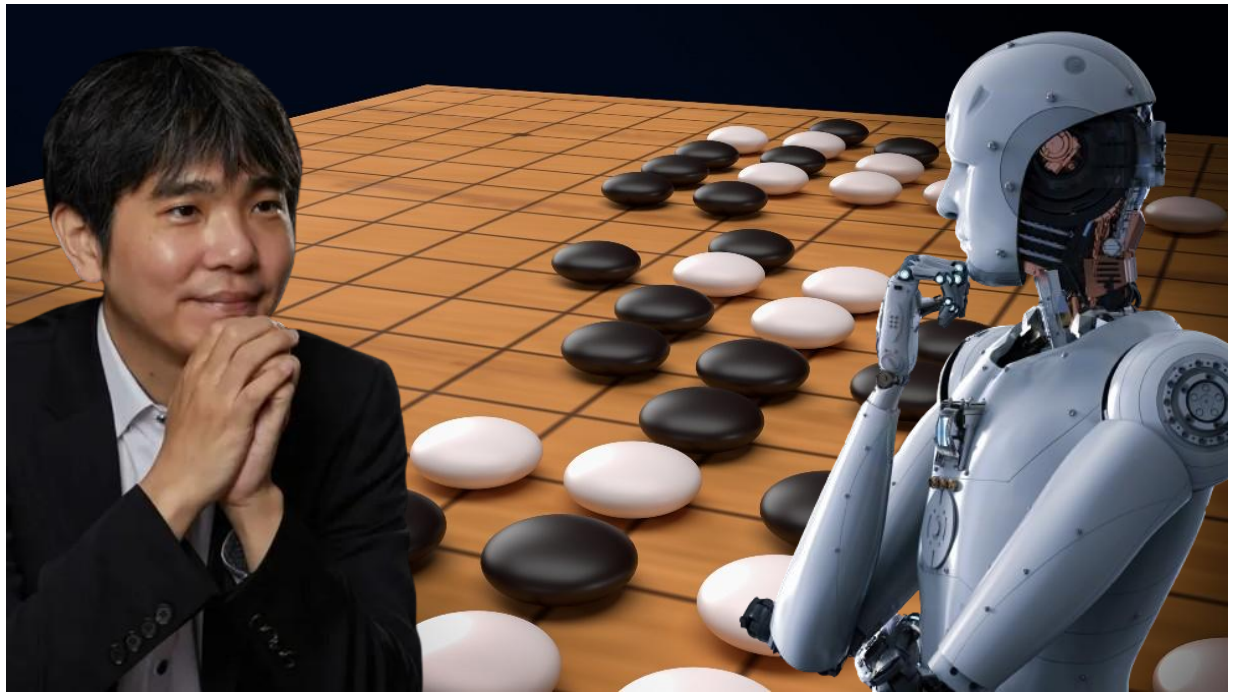
2015년 ResNet ‘Accuracy is higher than human’



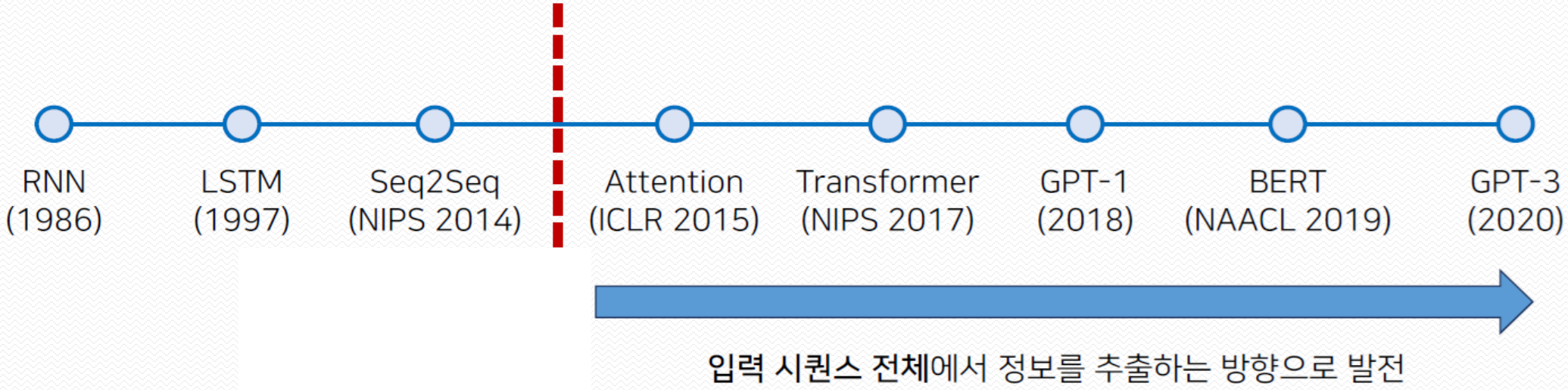
- 2015년 ILSVRC 우승
- Top-5 test error : 3.57%
- Residual neural network

HISTORY OF A.I.

2016년 Google Brain ‘AlphaGo’



HISTORY OF A.I. - 최근 언어 모델의 발전



AI TRAINING SERIES

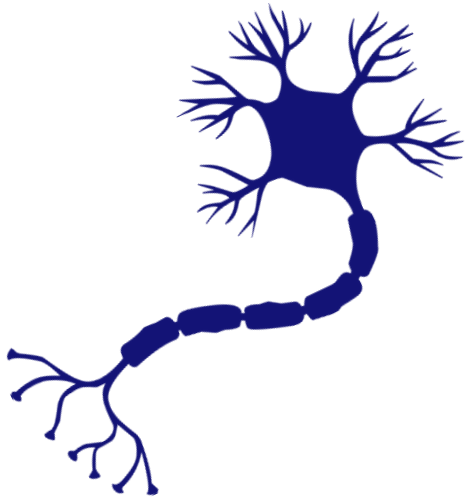
ANN

ARTIFICIAL NEURAL NETWORKS

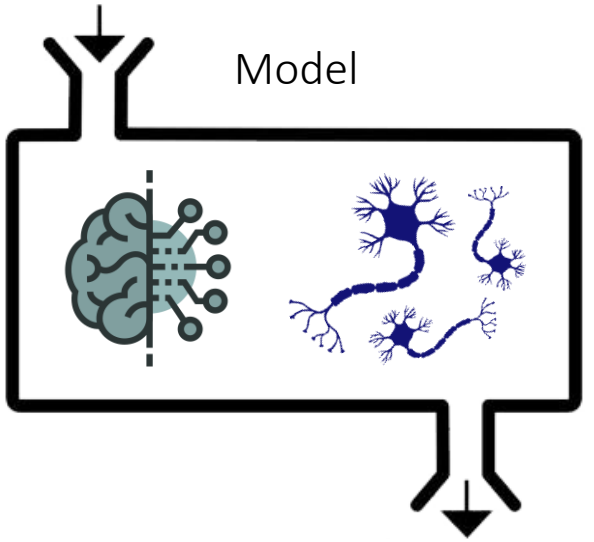
CONTENTS

NEURAL NETWORKS OVERVIEW

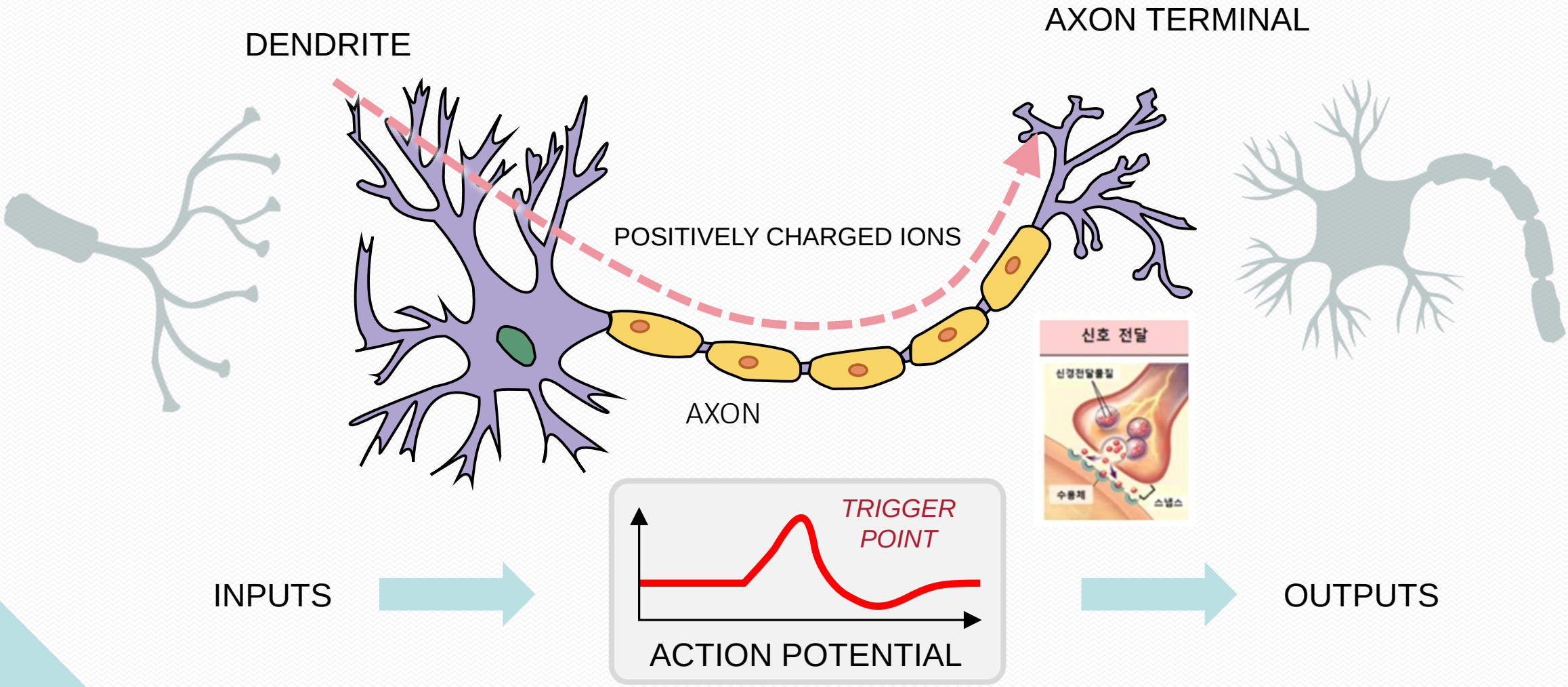
Neural networks



Neuron in human brain



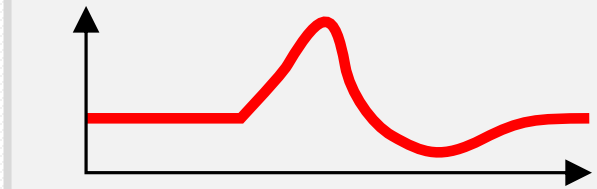
NEURON IN OUR BRAINS



ACTIVATION

NEURON

INPUTS



ACTION POTENTIAL

OUTPUTS

PERCEPTRON

INPUTS

CELL BODY

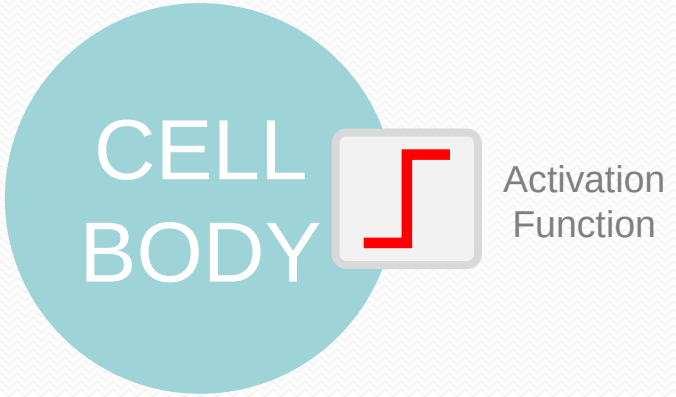
ACTIVATION FUNCTION

OUTPUT(S)

WEIGHTS AND BIASES

WEIGHTS

- CONTROL THE SIGNAL BETWEEN TWO NEURONS
- DETERMINES HOW THE INPUT AFFECTS THE OUTPUT



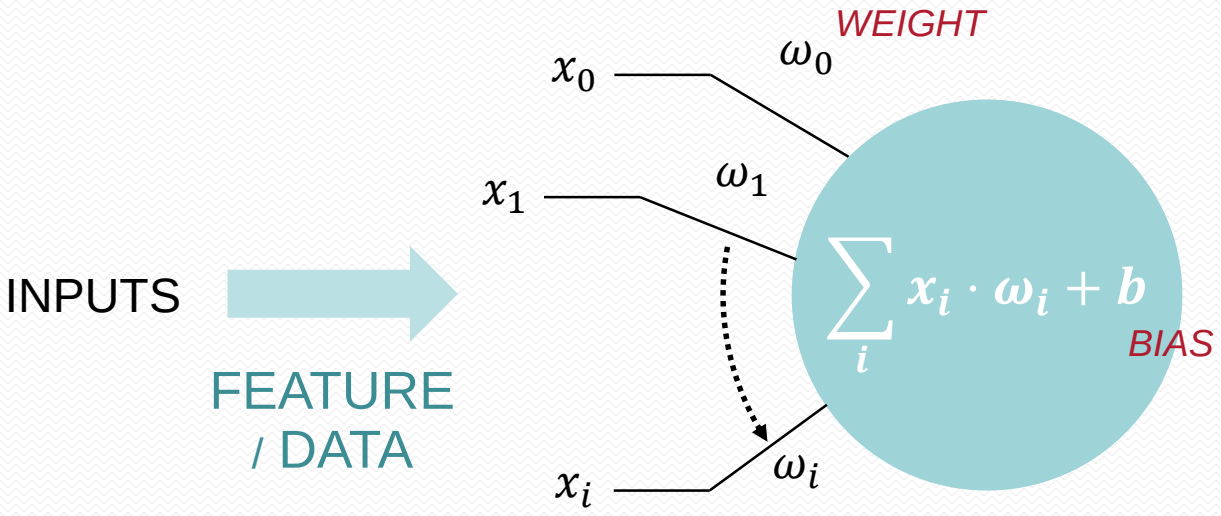
$$Y = Activation(\Sigma(weight \cdot input) + bias)$$

BIASES

- ADDITIONAL INPUT CONSTANT VALUES FOR NEXT NEURONS
- NOT INFLUENCED BY THE PREVIOUS NEURONS

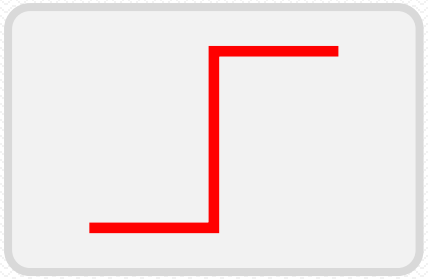
  These are the connections that go out of their own weights

PERCEPTRON



$H(x) = W_x + b$

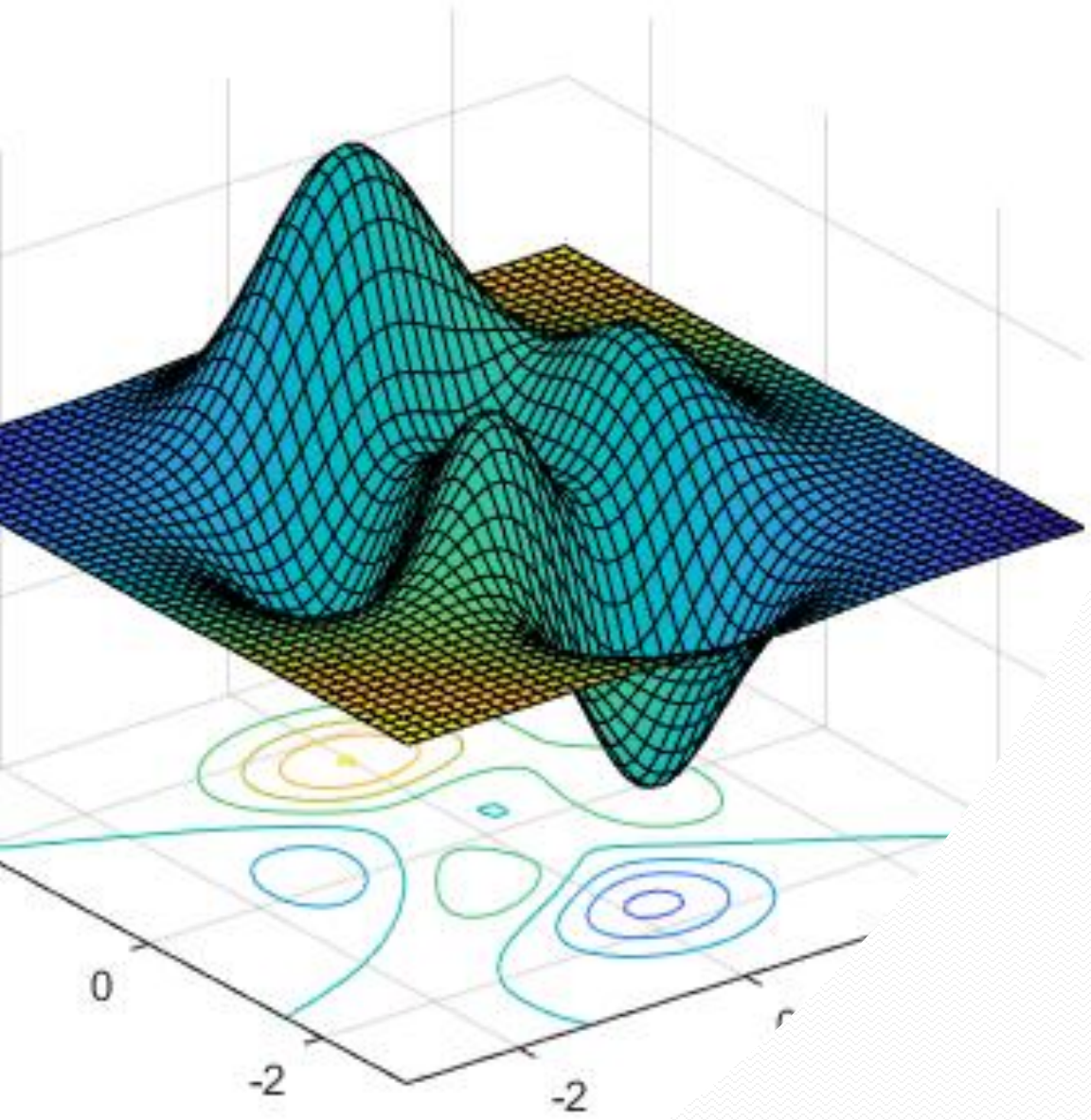
HYPOTHESIS



ACTIVATION FUNCTION
ex. Hard limit function

OUTPUT

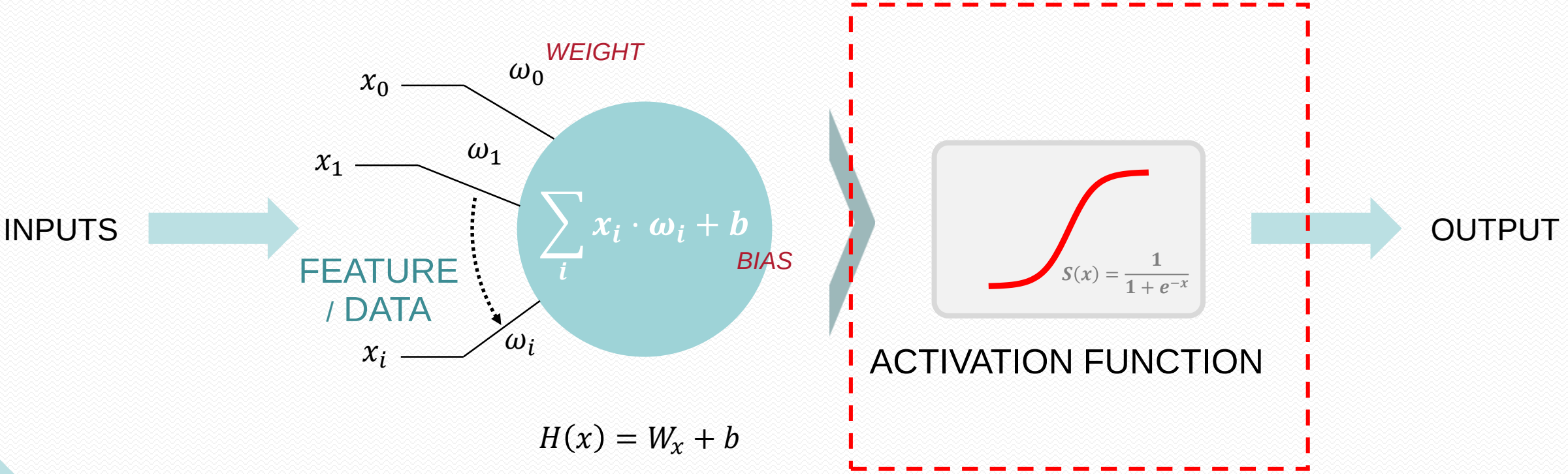
$$y = \begin{cases} 1 & \sum_{i=1}^{n} x_i w_i > 0 \\ 0 & otherwise \end{cases}$$



Deep-dive

Activation Function

PERCEPTRON



$H(x) = W_x + b$

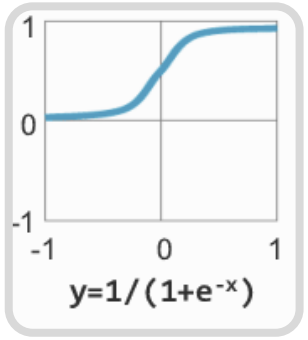
HYPOTHESIS

ACTIVATION FUNCTIONS

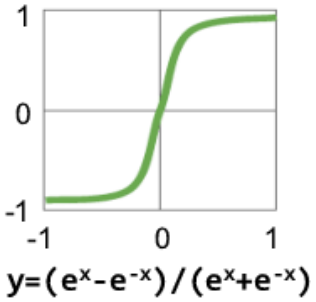
DEFINES THE OUTPUT OF A NODE USING A GIVEN INPUT OR SET OF INPUTS

Traditional Non-Linear Activation Functions

Sigmoid



Hyperbolic Tangent

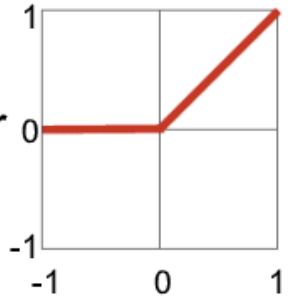


For binary classification

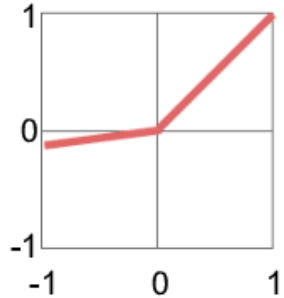
Widely used in hidden layers

Modern Non-Linear Activation Functions

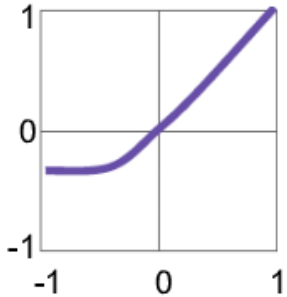
Rectified Linear Unit (ReLU)



Leaky ReLU



Exponential LU



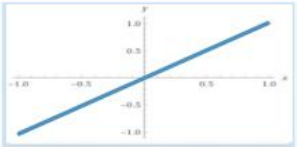
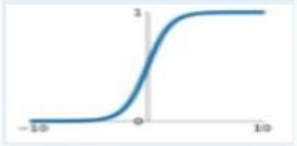
α = small const. (e.g. 0.1)

ACTIVATION FUNCTIONS

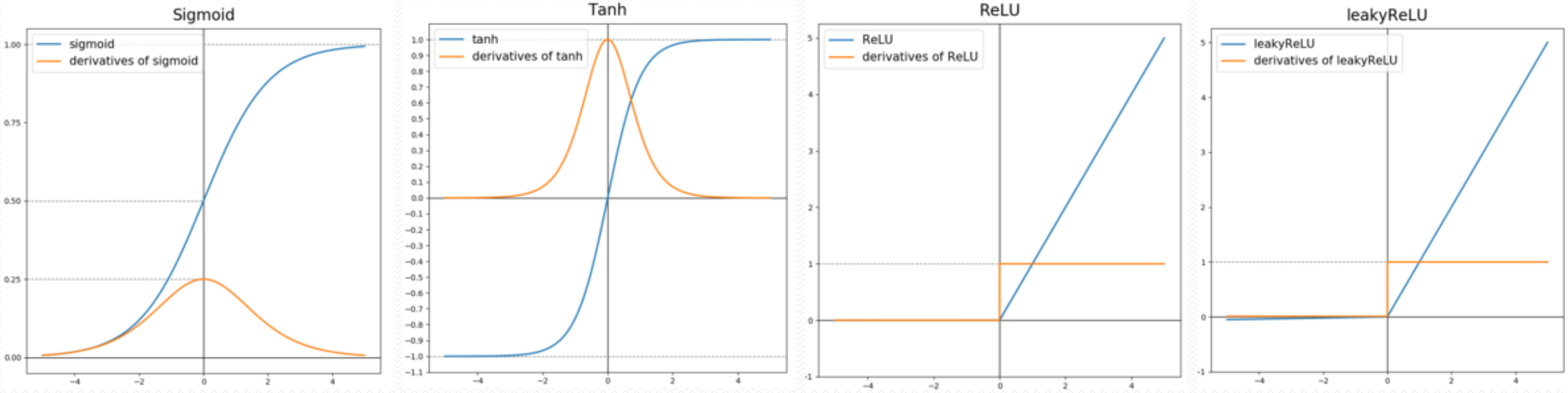
ACTIVATION FUNCTIONS

Why derivative/differentiation is used?

To know in which direction and how much to change or update the curve depending upon the slope.

Name	Plot	Equation	Derivative
Linear		$f(x) = x$	$f'(x) = 1$
Sigmoid		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$
Leaky ReLU		$f(x) = \begin{cases} ax \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$	$f'(x) = \begin{cases} a \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$

ACTIVATION FUNCTIONS

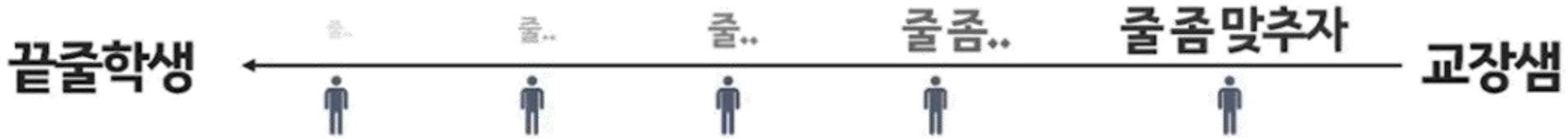


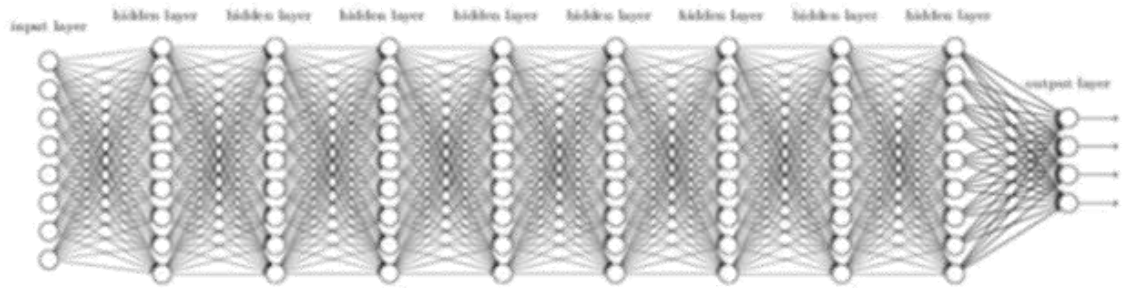
	Sigmoid	Tanh	ReLU	Leaky ReLU
Disadvantage	Vanishing Gradient Computational penalty	Vanishing Gradient	Dying ReLU problem	Result not consistent for negative
Advantage	1 or 0, binary value	Zero centered	Computational efficient Non-linear	Prevent negative Values become zero

https://ganghee-lee.tistory.com/32

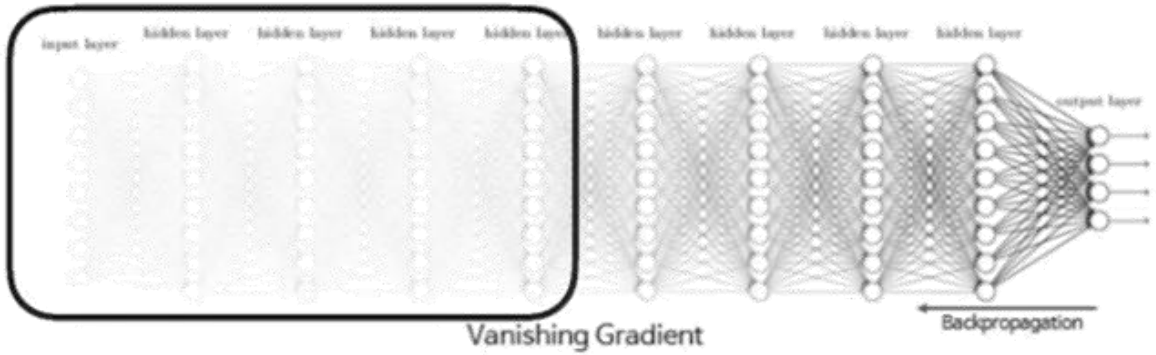
Vanishing gradient

레이어가 깊을 수록 업데이트가 사라져감.



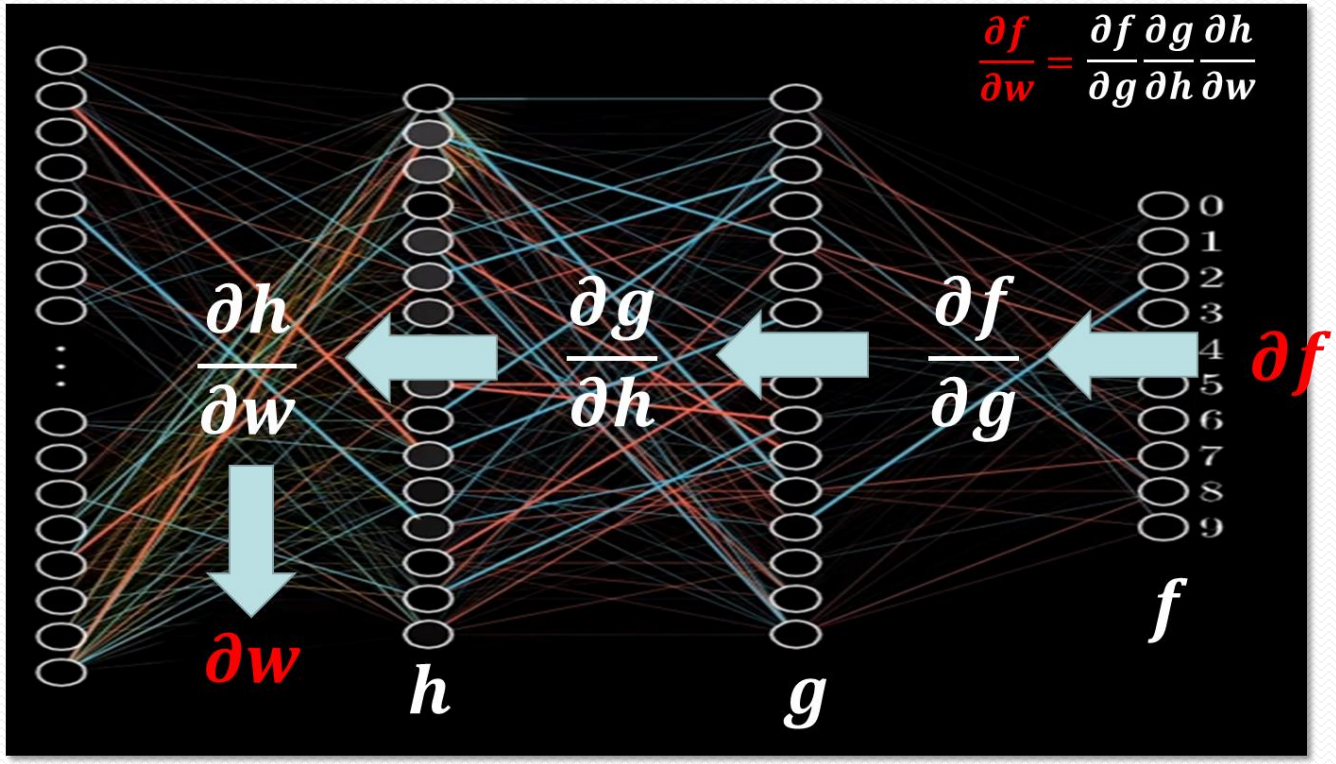


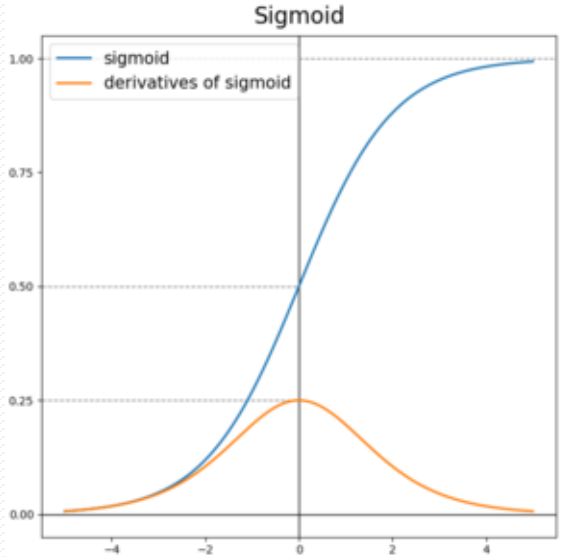
학습이 잘 안됨

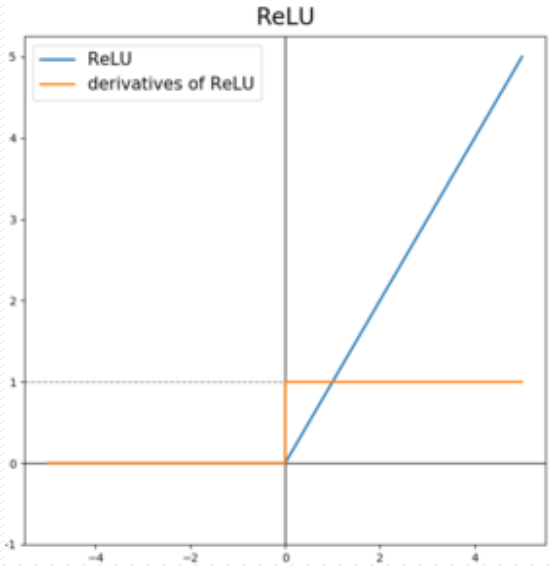


ACTIVATION FUNCTIONS: Vanishing Gradient

CALCULATE THE ∂w from ∂f

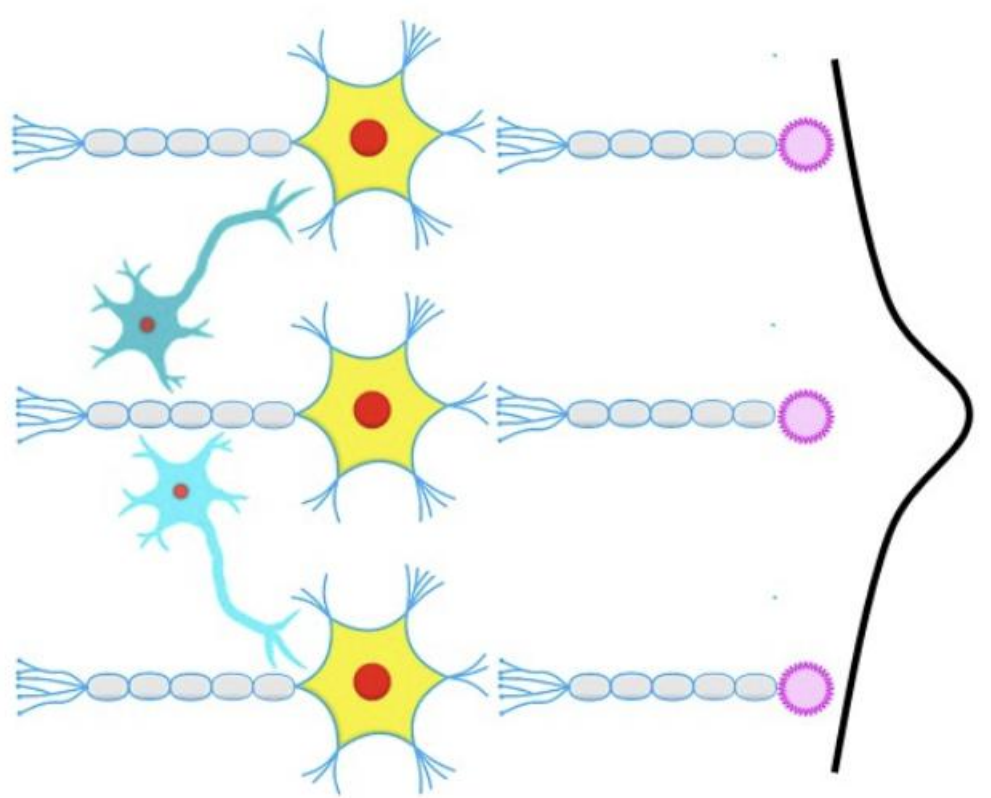


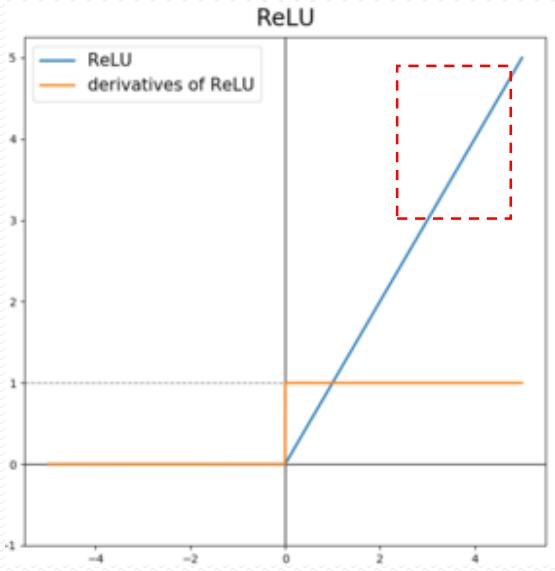




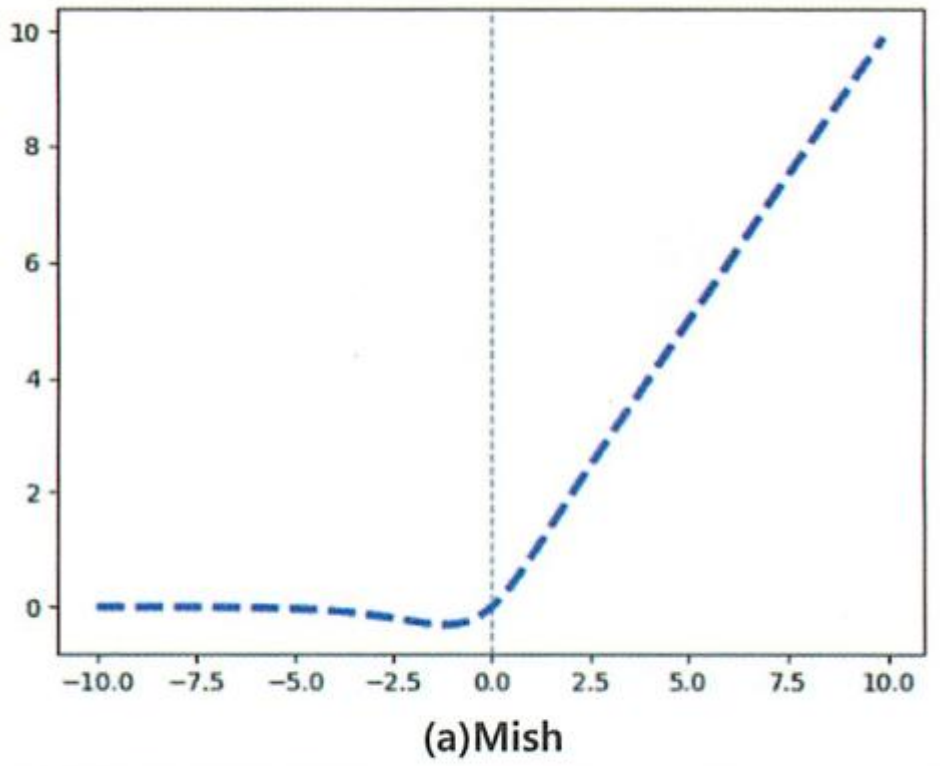
ACTIVATION FUNCTIONS:ReLu vs 측면억제

입출력단을 갖는 2개의 신경세포에서 각각의 신경세포의 입력 또는 출력이 서로 다른 신경세포의 입력 또는 출력에 의해 억제되는 현상.



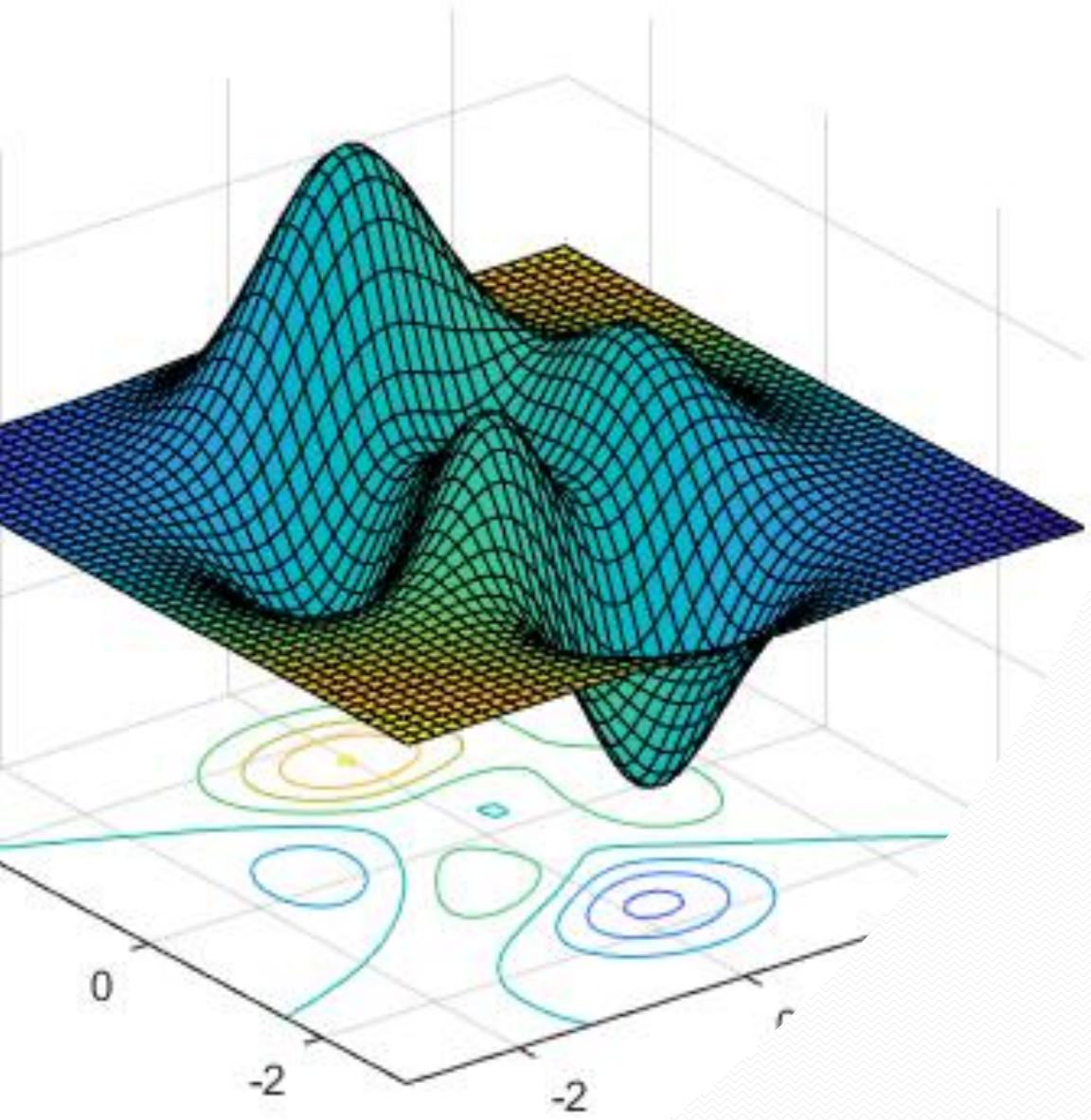


ACTIVATION FUNCTIONS:Yolov4 Mish func.



(a)Mish

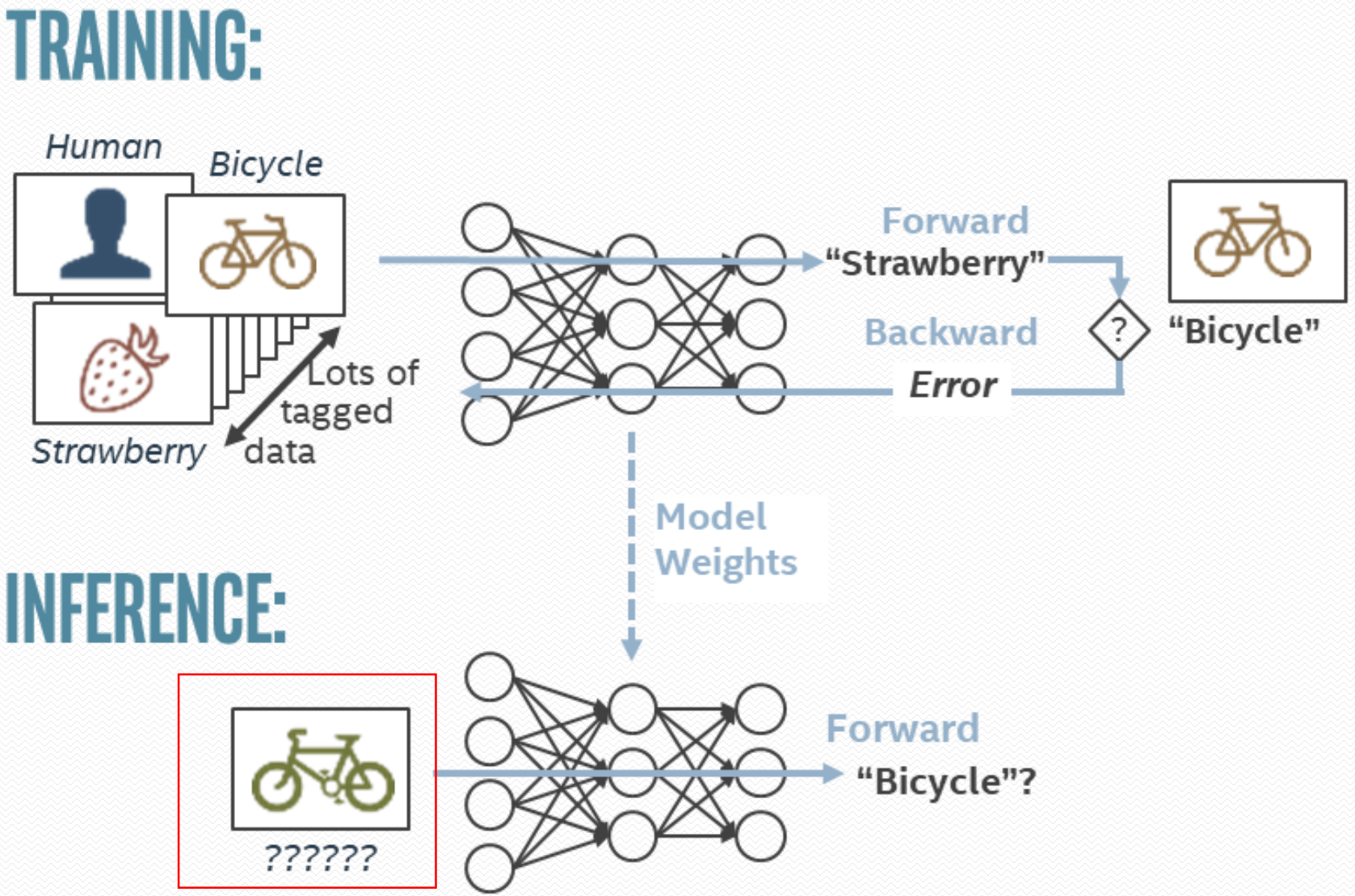
```
def MISH(x):
    return x* np.tanh(np.log(1 + np.exp(x)))
```



Deep-dive

ANN: technical in details

TRAINING & INFERENCING

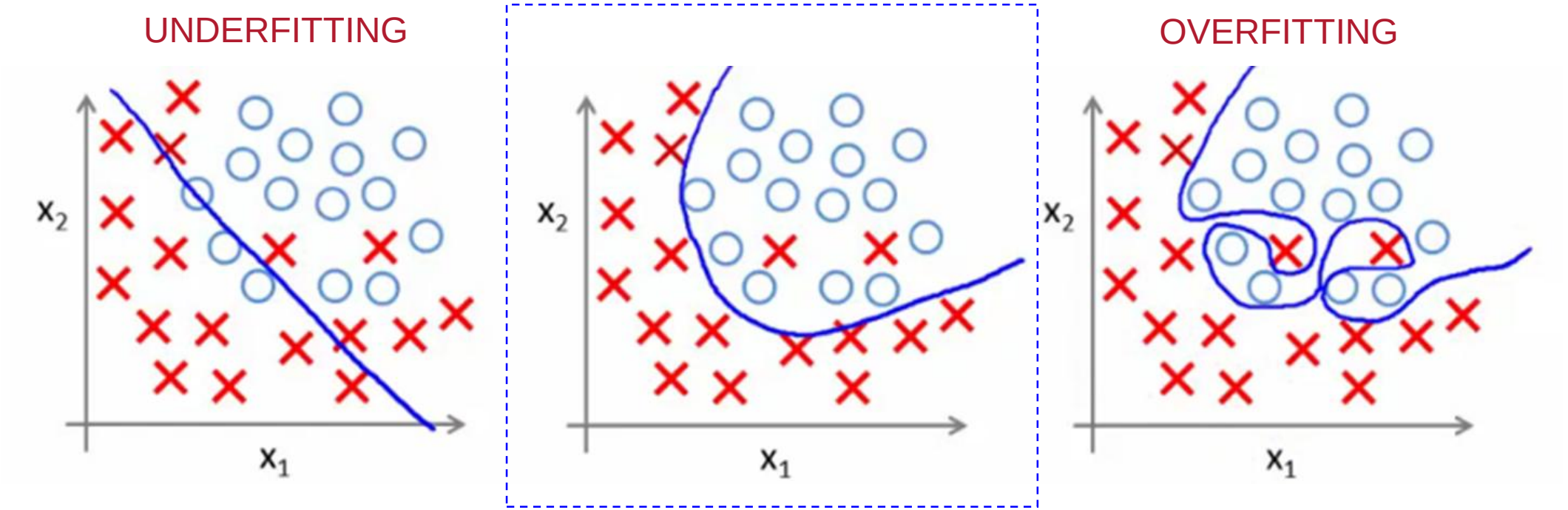


OVERFITTING / UNDERFITTING

HIGH COMPLEXITY MODEL = HIGH POSSIBILITY OF OVERFITTING

If the model is too specific, generalization is reduced

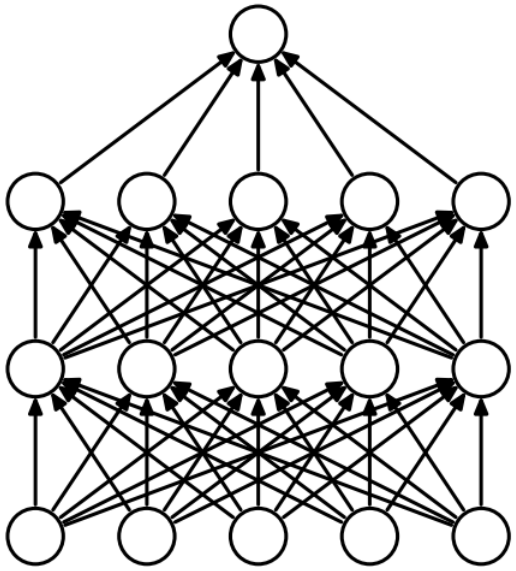
When new test data sets come in → they may not be well categorized



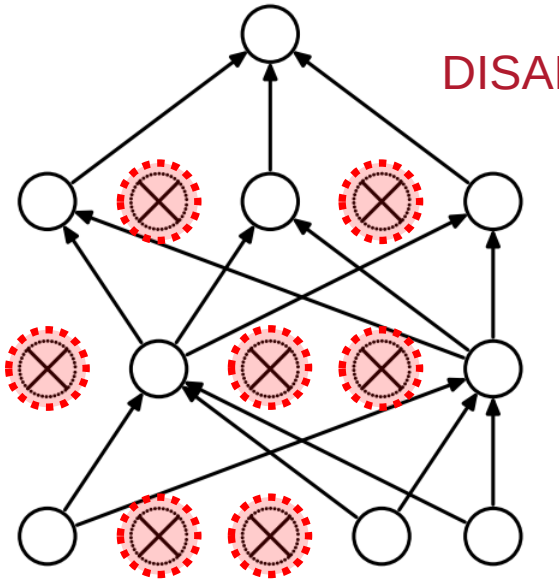
DROPOUT

RANDOMLY SET SOME NEURONS TO ZERO IN THE FORWARD PASS

Voting effect → makes the result not biased
Anti co-adaptation → creates a robust network



(a) Standard Neural Net



(b) After applying dropout.

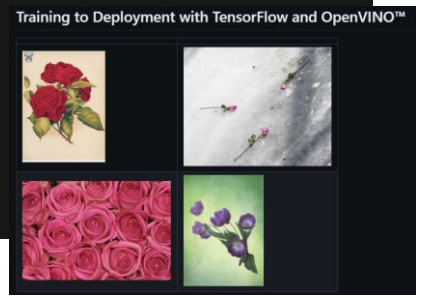
DISABLE RANDOMLY

DROPOUT

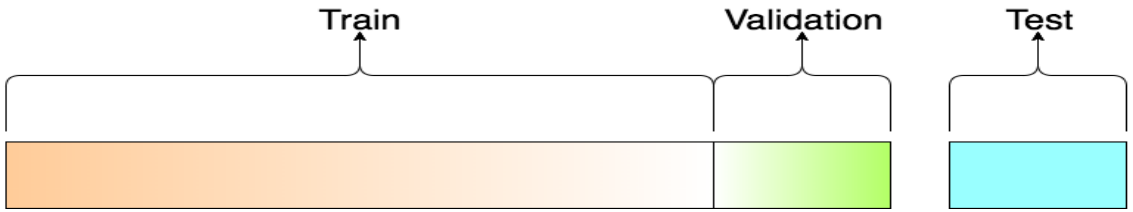
- dropout

```
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0

conv2d (Conv2D)              (None, 180, 180, 16)      448

max_pooling2d (MaxPooling2D  (None, 90, 90, 16)        0
)

conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640

max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)        0
2D)

conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496

max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)        0
2D)

flatten (Flatten)            (None, 30976)             0

dense (Dense)                (None, 128)               3965056

dense_1 (Dense)              (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

```python
# 모델 만들기
model = tf.keras.Sequential()
model.add(
    tf.keras.layers.Rescaling(
        1./255, input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)))
model.add(
    tf.keras.layers.Conv2D(16, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Conv2D(32, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Conv2D(64, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Flatten())
model.add(
    tf.keras.layers.Dropout(0.2)
)
model.add(
    tf.keras.layers.Dense(128, activation="relu")
)
model.add(
    tf.keras.layers.Dense(5))
model.summary()
```



DATASET



TRAIN

The actual dataset that we use to train the model (**weights** and **biases** in the case of a Neural Network).

VALIDATION

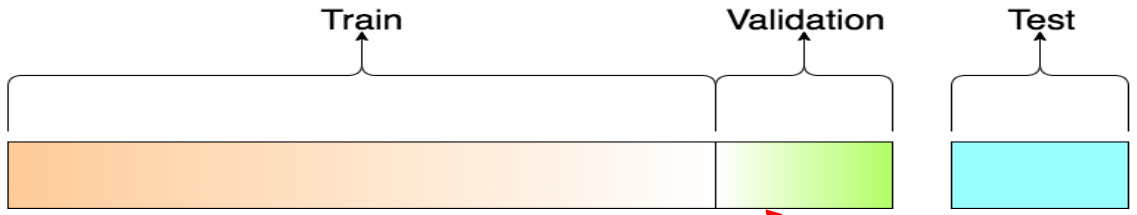
The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model **hyperparameters**.

TEST

The sample of data used to provide an **unbiased** evaluation of a final model fit on the training dataset.

DATASET

```
[ ]  epochs = 20
     history = model.fit(
         train_ds,
         validation_data=val_ds,
         epochs=epochs
     )
```




```
Epoch 1/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 7s 36ms/step - accuracy: 0.2904 - loss: 1.6266 - val_accuracy: 0.4782 - val_loss: 1.1946
Epoch 2/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 8s 29ms/step - accuracy: 0.5428 - loss: 1.0985 - val_accuracy: 0.5926 - val_loss: 1.0086
Epoch 3/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.6241 - loss: 0.9589 - val_accuracy: 0.6267 - val_loss: 0.9625
Epoch 4/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.6685 - loss: 0.8739 - val_accuracy: 0.6458 - val_loss: 0.8742
Epoch 5/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.6804 - loss: 0.8385 - val_accuracy: 0.6785 - val_loss: 0.8529
Epoch 6/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.6817 - loss: 0.8259 - val_accuracy: 0.6526 - val_loss: 0.9076
Epoch 7/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7187 - loss: 0.7595 - val_accuracy: 0.6798 - val_loss: 0.7931
Epoch 8/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7385 - loss: 0.7043 - val_accuracy: 0.6948 - val_loss: 0.8105
Epoch 9/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.7274 - loss: 0.6920 - val_accuracy: 0.7166 - val_loss: 0.7484
Epoch 10/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7636 - loss: 0.6486 - val_accuracy: 0.7098 - val_loss: 0.7514
Epoch 11/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7572 - loss: 0.6466 - val_accuracy: 0.7084 - val_loss: 0.7582
Epoch 12/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7673 - loss: 0.6086 - val_accuracy: 0.7275 - val_loss: 0.7308
Epoch 13/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7592 - loss: 0.6245 - val_accuracy: 0.7221 - val_loss: 0.7366
Epoch 14/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.8090 - loss: 0.5091 - val_accuracy: 0.6948 - val_loss: 0.8095
Epoch 15/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.7971 - loss: 0.5397 - val_accuracy: 0.7425 - val_loss: 0.7073
Epoch 16/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.8051 - loss: 0.5488 - val_accuracy: 0.7384 - val_loss: 0.7027
Epoch 17/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.7995 - loss: 0.5057 - val_accuracy: 0.7425 - val_loss: 0.7165
Epoch 18/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.8374 - loss: 0.4481 - val_accuracy: 0.7289 - val_loss: 0.7626
Epoch 19/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.8240 - loss: 0.4638 - val_accuracy: 0.7602 - val_loss: 0.7015
Epoch 20/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.8490 - loss: 0.4301 - val_accuracy: 0.7044 - val_loss: 0.8483
```

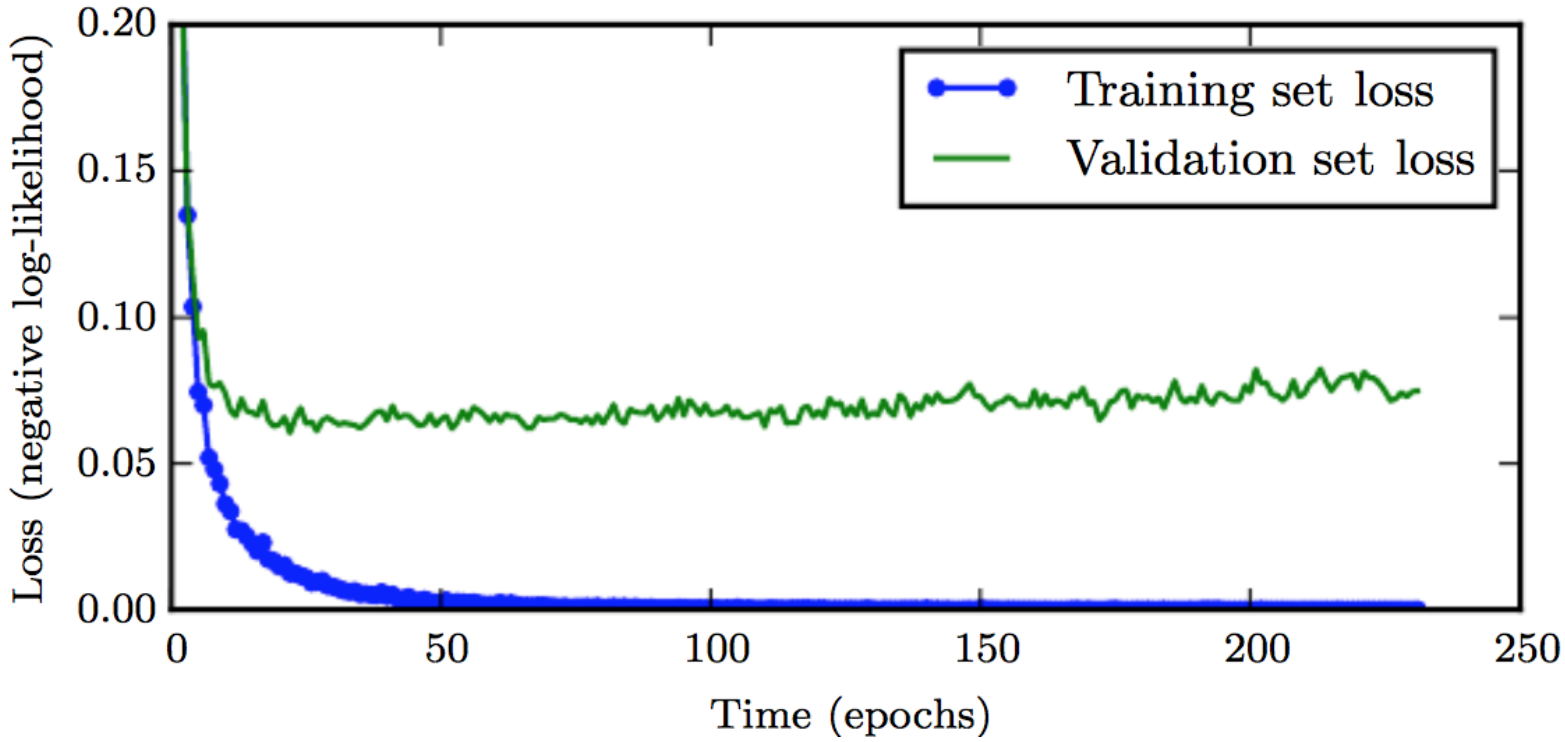
GENERALIZATION / RIZATION

EARLY STOPPING

Stop training when the gap between validation error and training error becomes large

Simple to do without additional hyperparameters, but mixing two problems

→ *UNDERFITTING AND OVERFITTING*



HYPERPARAMETERS

 # OF HIDDEN LAYER

determines how many hidden layers is in neural network

 # OF EPOCH

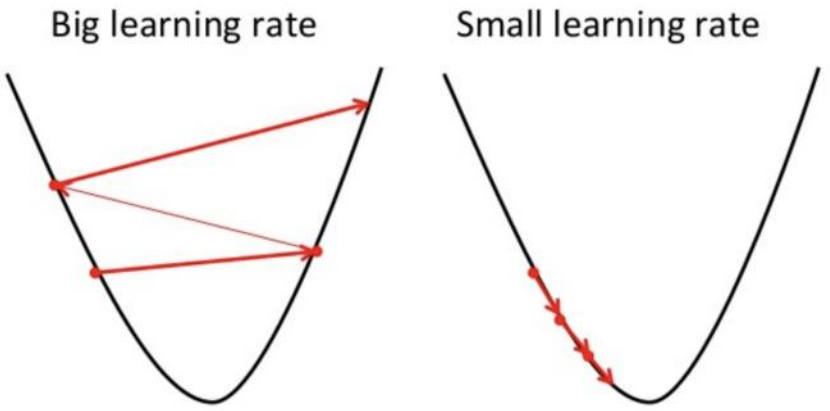
finds the boundary value of the epoch where the error is minimized

BATCH SIZE

means the size of the data to be divided into when learning

LEARNING RATE

determines how fast or slow to move per iteration



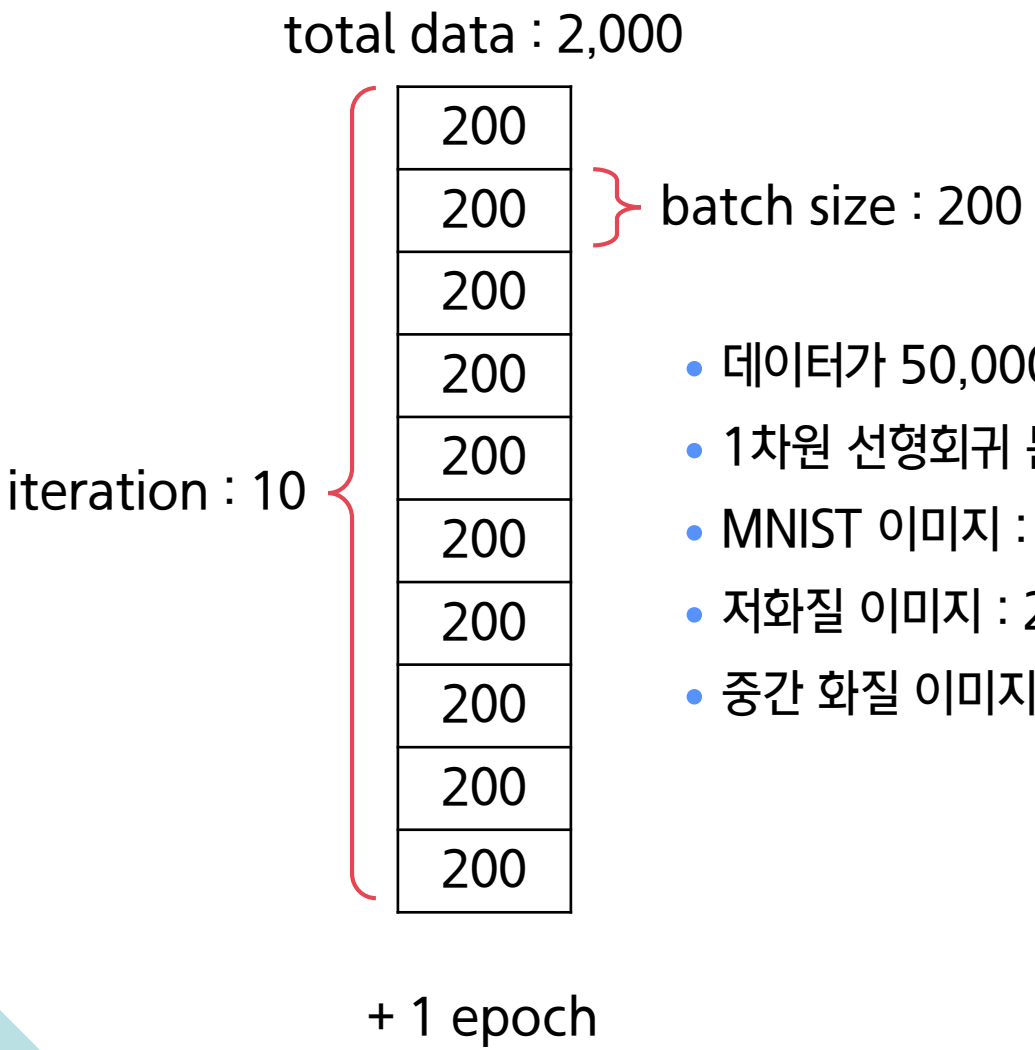
INITIAL VALUE OF EACH PARAMETER

determines the initial value when running iteration

LOSS / COST FUNCTION

Decides how to calculate between actual result and calculated output

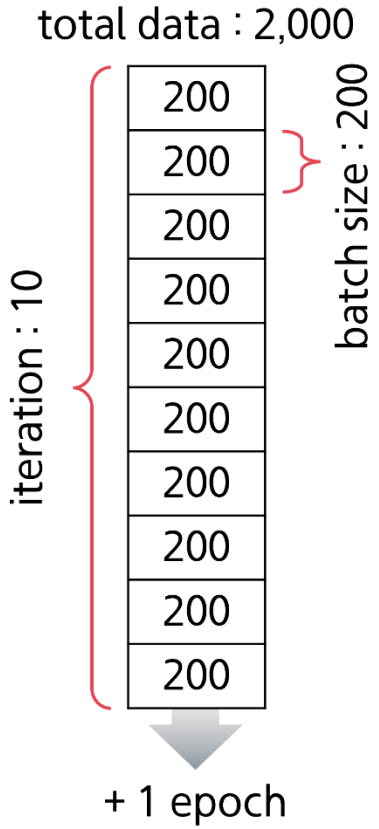
EPOCH & ITERATION & BATCH



- 데이터가 50,000개 있음
- 1차원 선형회귀 문제 : 50,000 × 4bytes = 200,000bytes = 0.2MB
- MNIST 이미지 : 28 × 28 × 50,000 × 4bytes = 156MB
- 저화질 이미지 : 280 × 280 × 50,000 × 4bytes = 15,600MB = 15.6GB
- 중간 화질 이미지 : 560 × 560 × 50,000 × 4bytes = 15,600MB = 62.4GB

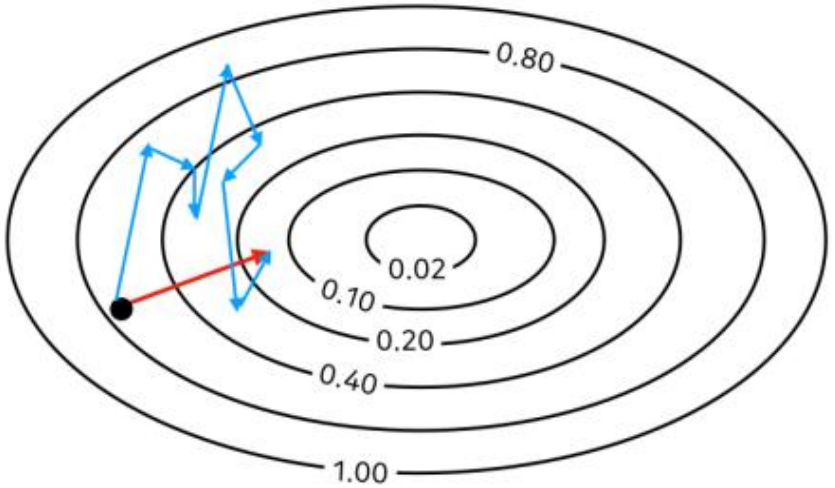
EPOCH & ITERATION & BATCH

일부분의 데이터(batch)를 사용하여 Gradient를 추정하면 되지 않을까?



If Batch size is Big(BGD)?

If Batch size is small(SGD)?



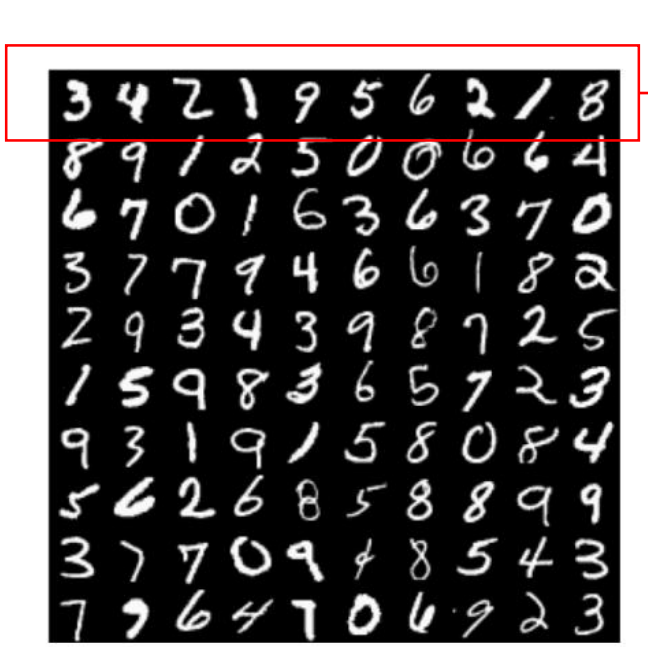
Then, Big batch size is more efficient?

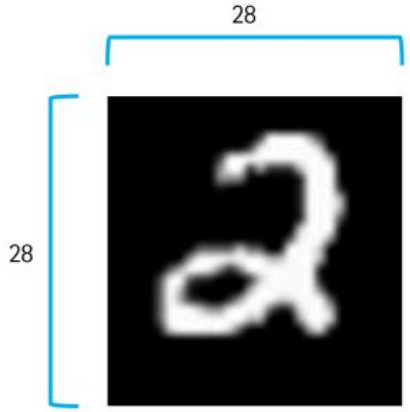
→requires too much memory

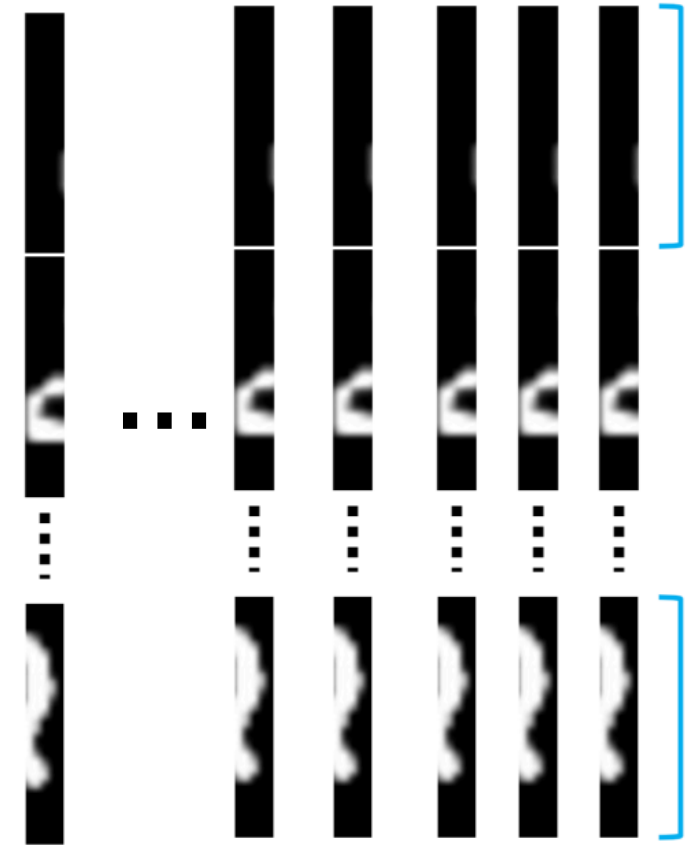
→Too many parameters makes slow learning speed

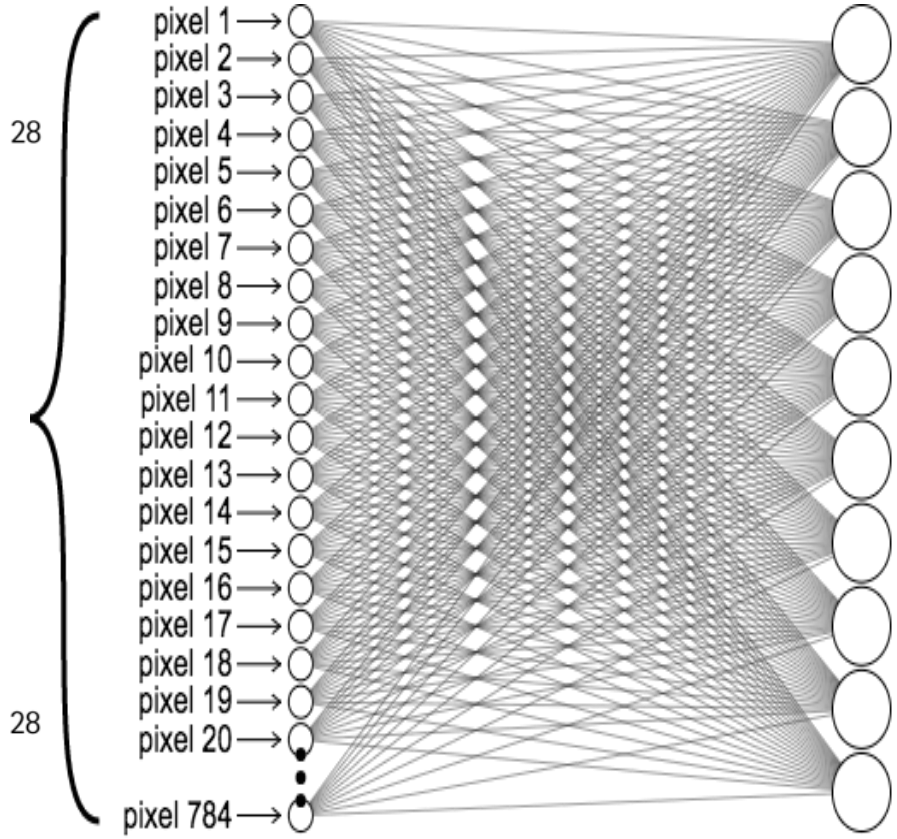
- 계산속도 증가
- 메모리 용량 감소
- Local minimu을 빠져나갈 가능성 증가
- 부정확한 Gradient
- 가장 빠른 방향으로 Search방향 설정 불가

EPOCH & ITERATION & BATCH





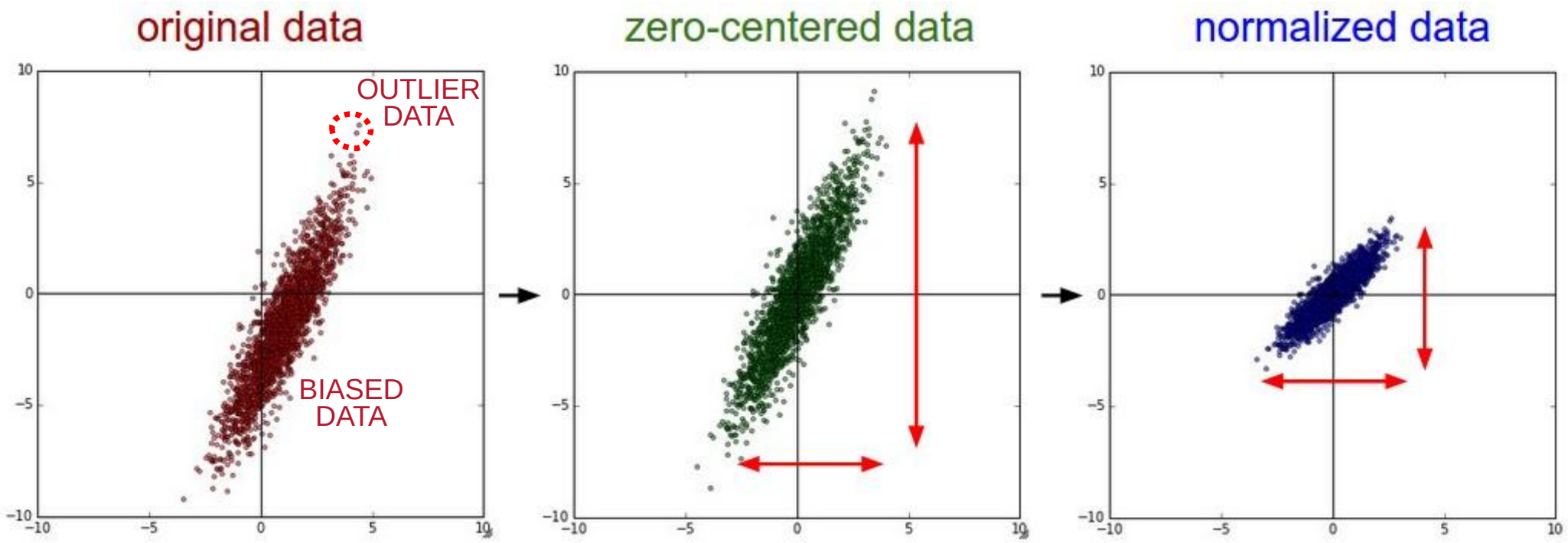




BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

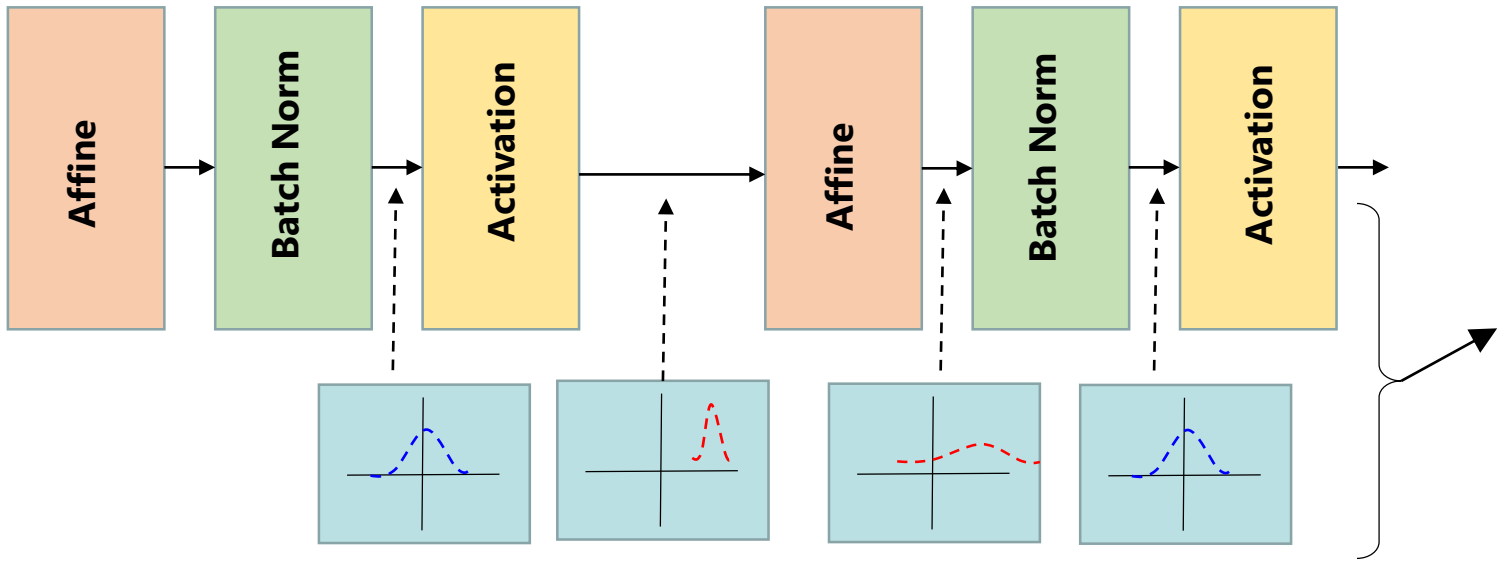
Method to help the network learn better

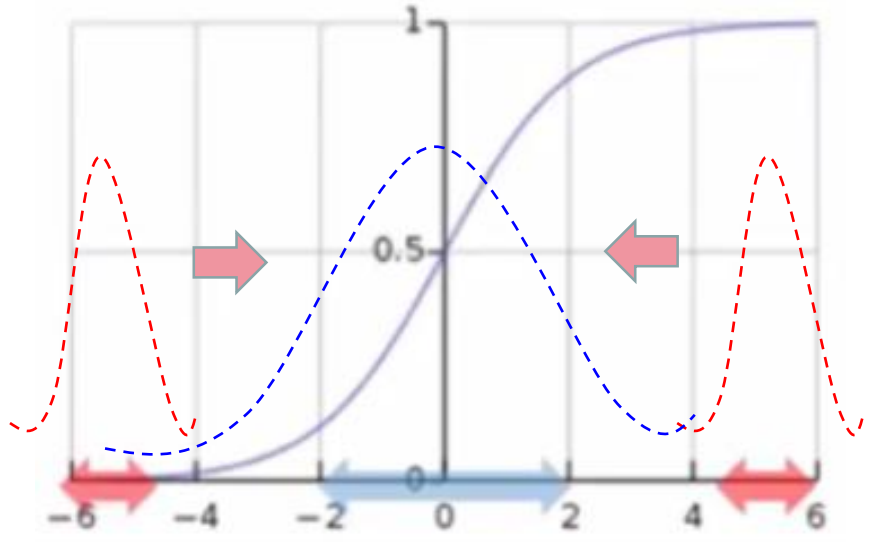


BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better



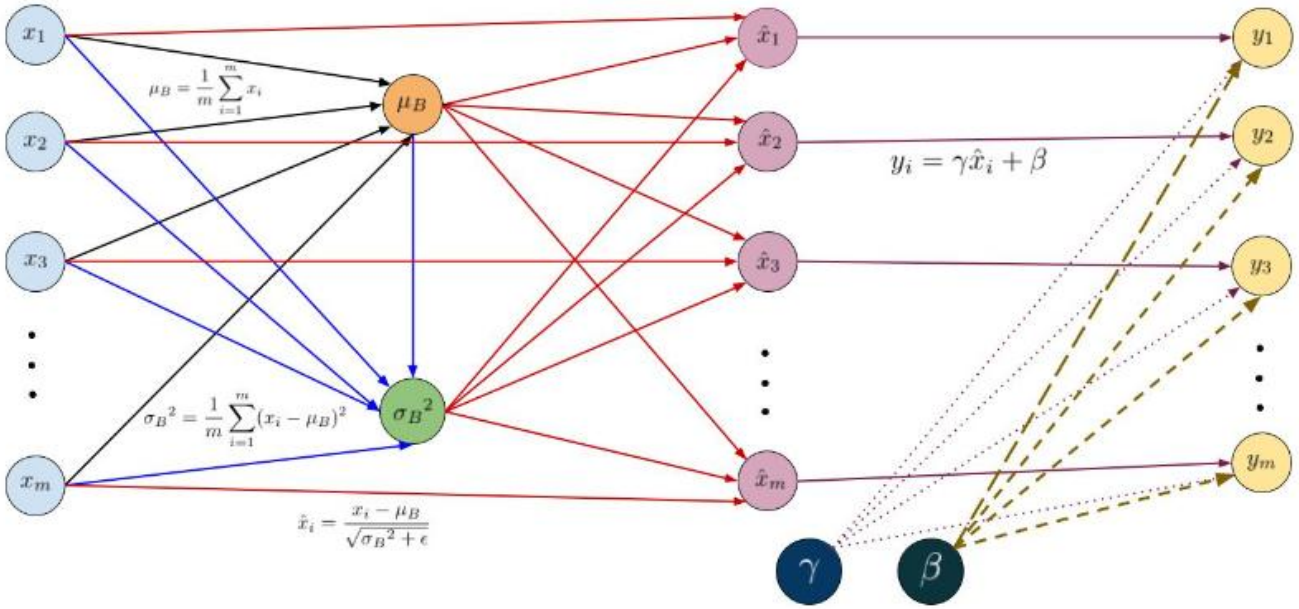


$$\gamma, \beta$$

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

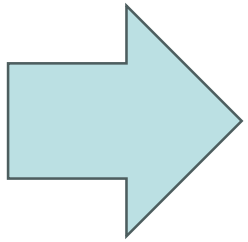
```python
def batchnorm_forward(x, gamma, beta, eps=1e-5):
    N, D = x.shape

    sample_mean = x.mean(axis=0)
    sample_var = x.var(axis=0)

    std = np.sqrt(sample_var + eps)
    x_centered = x - sample_mean
    x_norm = x_centered / std
    out = gamma * x_norm + beta

    cache = (x_norm, x_centered, std, gamma)

    return out, cache
```

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better

$$\gamma\left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) + \beta$$



scale and shift

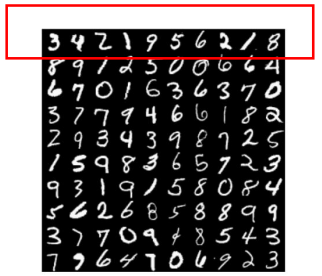
→ 분포재조정됨

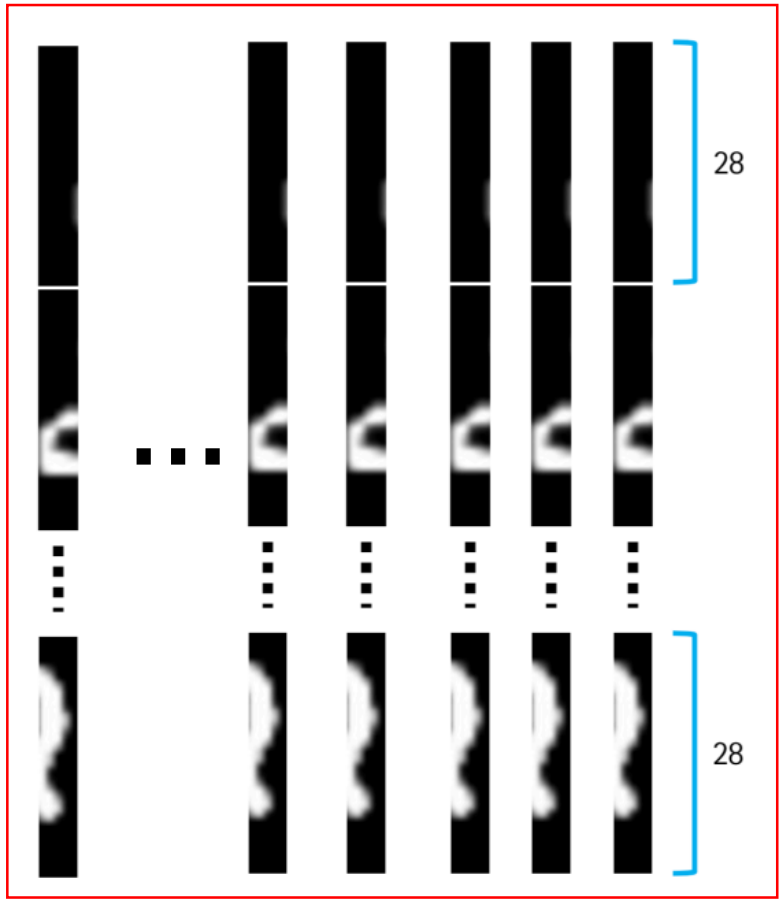
- 평균: β

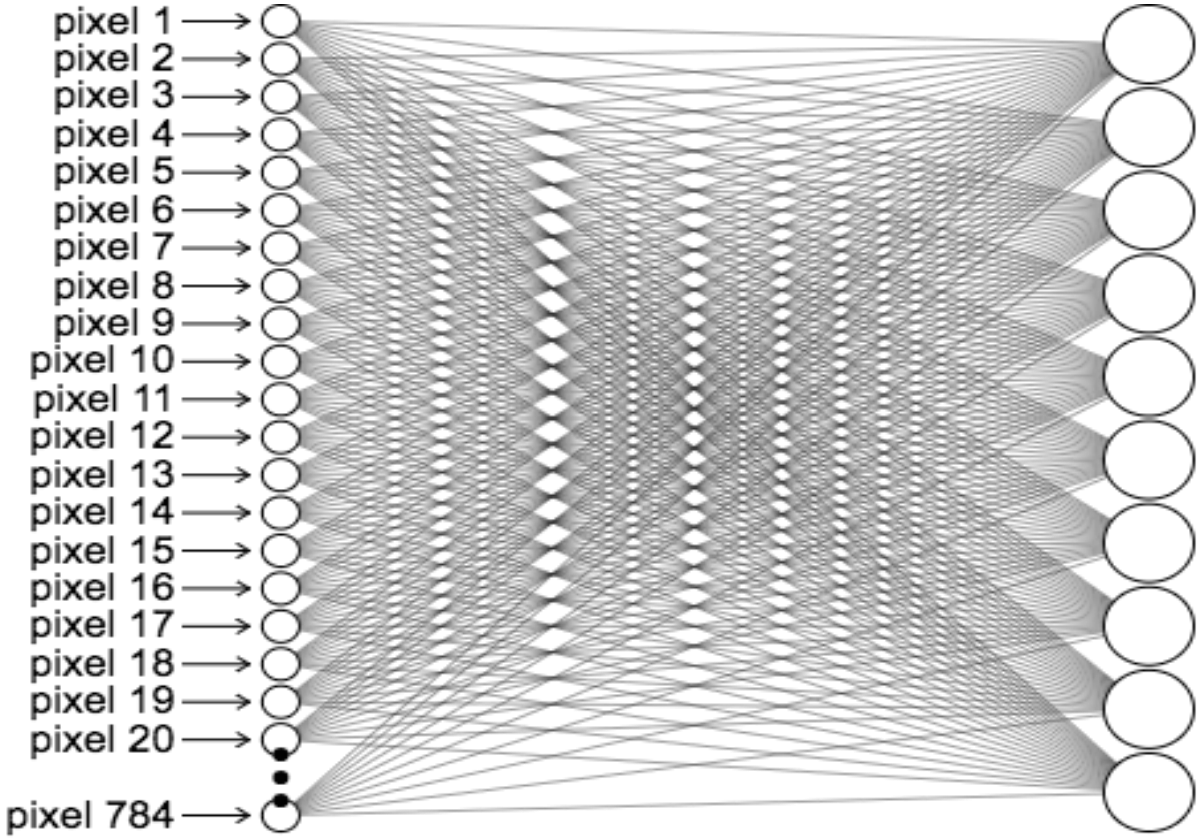
- 분산: γ^2

BATCH NORMALIZATION

Batch Normalization

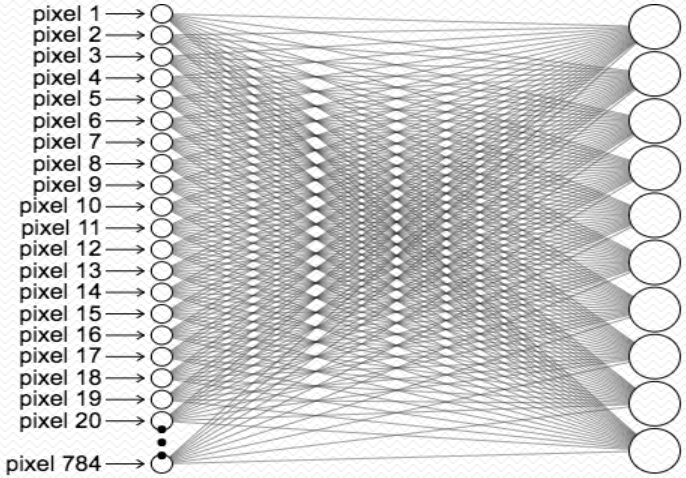


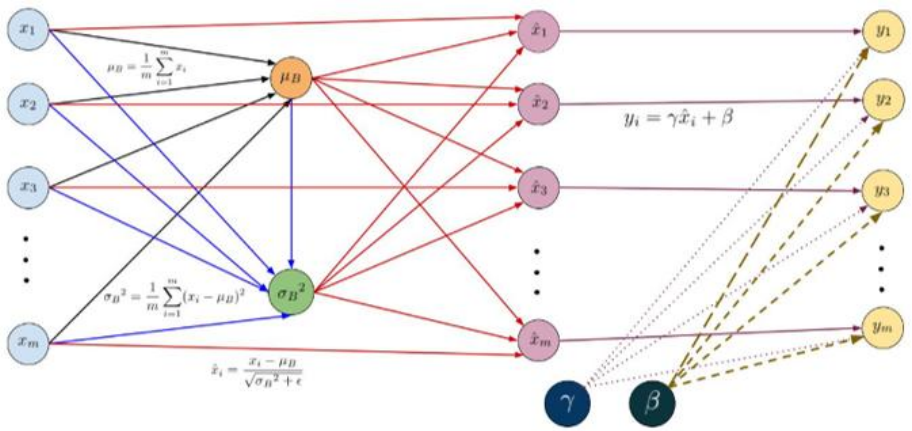


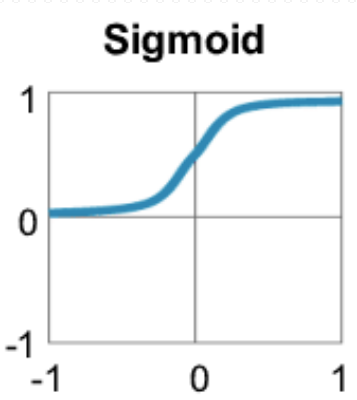


총 784차원 벡터

BATCH NORMALIZATION

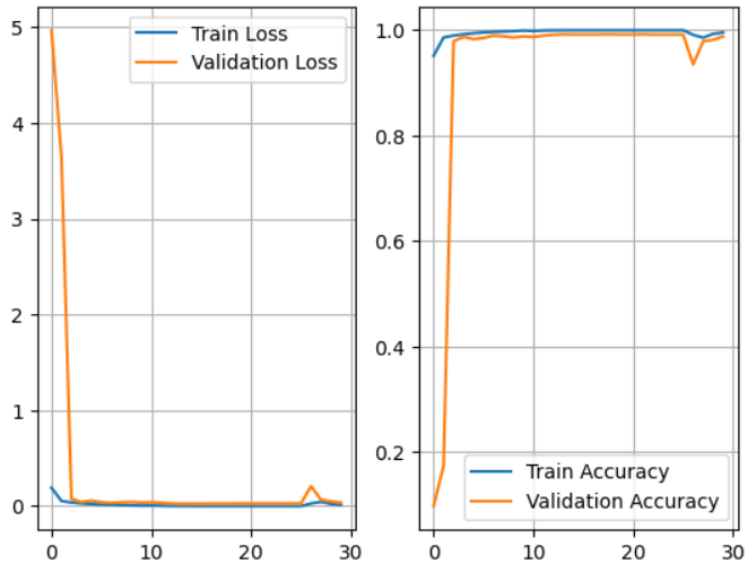




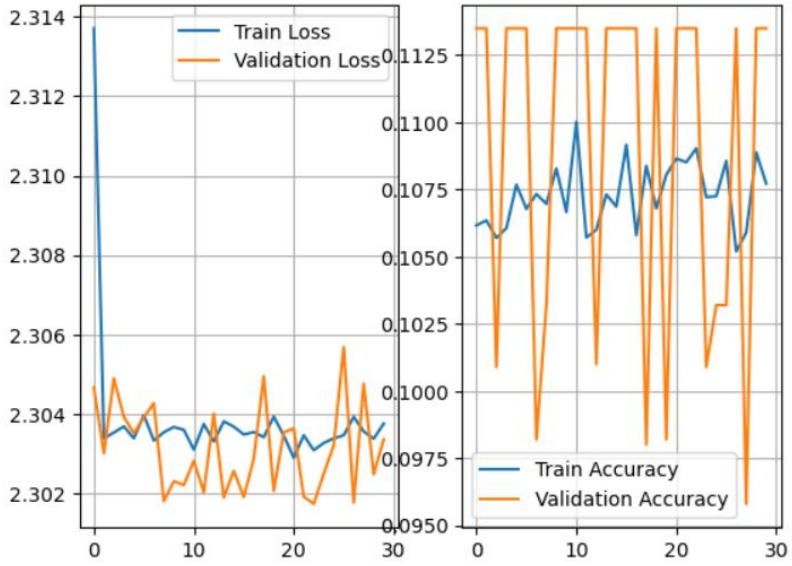


배치정규화 알고리즘

```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```




```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
# model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```



배치정규화 알고리즘

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 27, 27, 32)	160
batch_normalization (BatchNormalization)	(None, 27, 27, 32)	128
conv2d_1 (Conv2D)	(None, 26, 26, 64)	8256
batch_normalization_1 (BatchNormalization)	(None, 26, 26, 64)	256
conv2d_2 (Conv2D)	(None, 13, 13, 128)	32896
batch_normalization_2 (BatchNormalization)	(None, 13, 13, 128)	512
conv2d_3 (Conv2D)	(None, 12, 12, 32)	16416
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 32)	128
conv2d_4 (Conv2D)	(None, 11, 11, 64)	8256
batch_normalization_4 (BatchNormalization)	(None, 11, 11, 64)	256
conv2d_5 (Conv2D)	(None, 5, 5, 128)	32896
batch_normalization_5 (BatchNormalization)	(None, 5, 5, 128)	512
flatten (Flatten)	(None, 3200)	0
dense (Dense)	(None, 128)	409728
dense_1 (Dense)	(None, 10)	1290

Total params: 511690 (1.95 MB)
Trainable params: 510794 (1.95 MB)
Non-trainable params: 896 (3.50 KB)

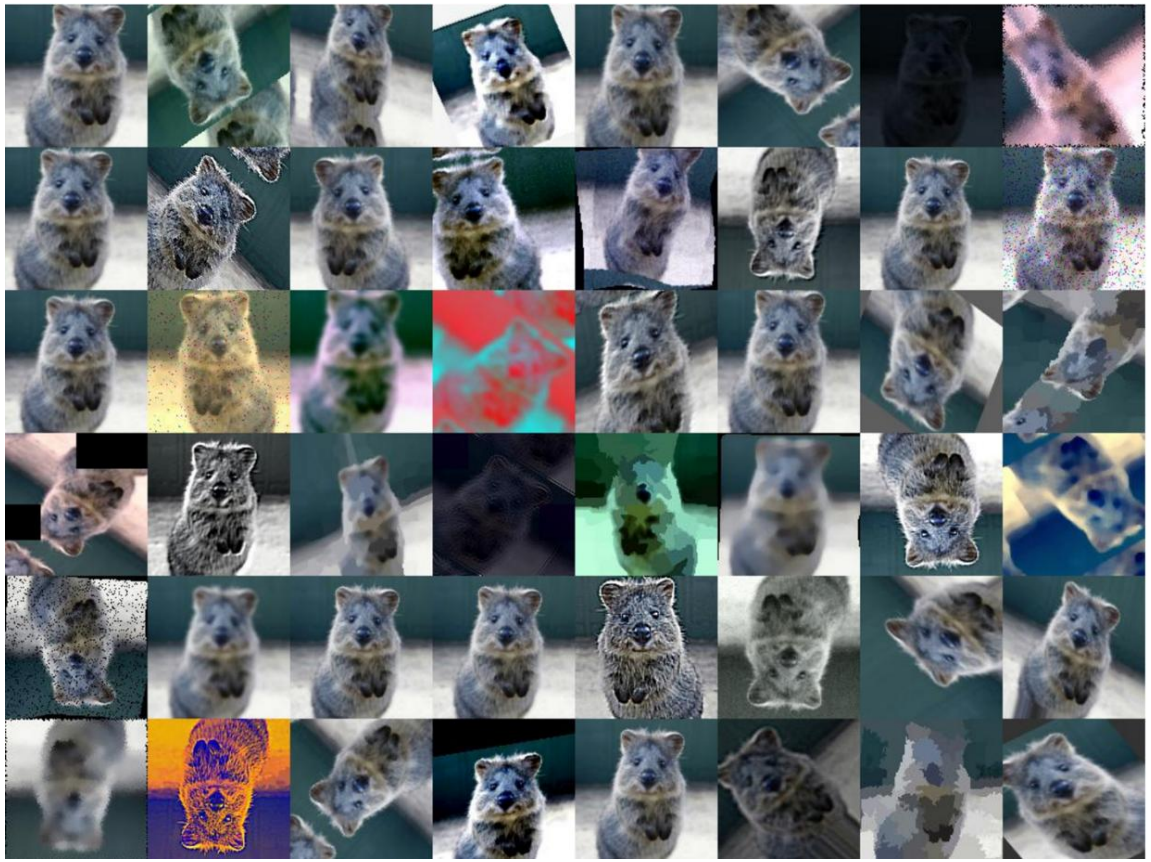
None

```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```

GENERALIZATION / REGULARIZATION

DATA AUGMENTATION

Strategy for improve CNN performance that enables user to significantly increase the diversity of data available for training models, without collecting new data



CALCULATE THE OUTPUT DATA / SOFTMAX

CALCULATE THE CONFIDENCE SCORE (Propbabilities)

- Suitable for classification problems
  multi-class classification
- Sum of scores will be 1
- Highest score is the prediction class

Hypothesis :

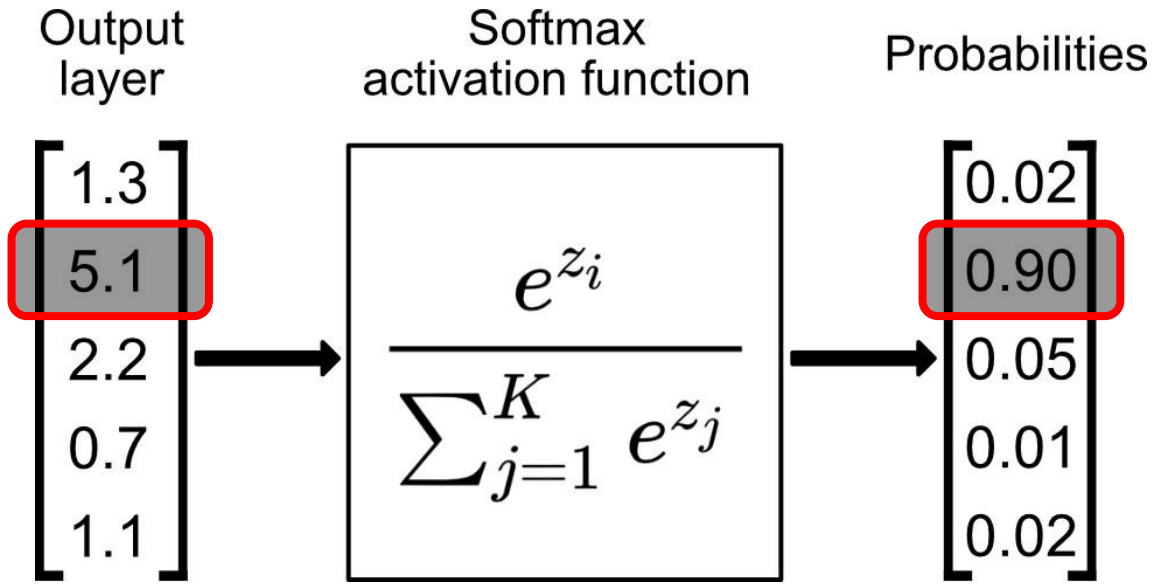
$$\hat{y} = wx + b$$

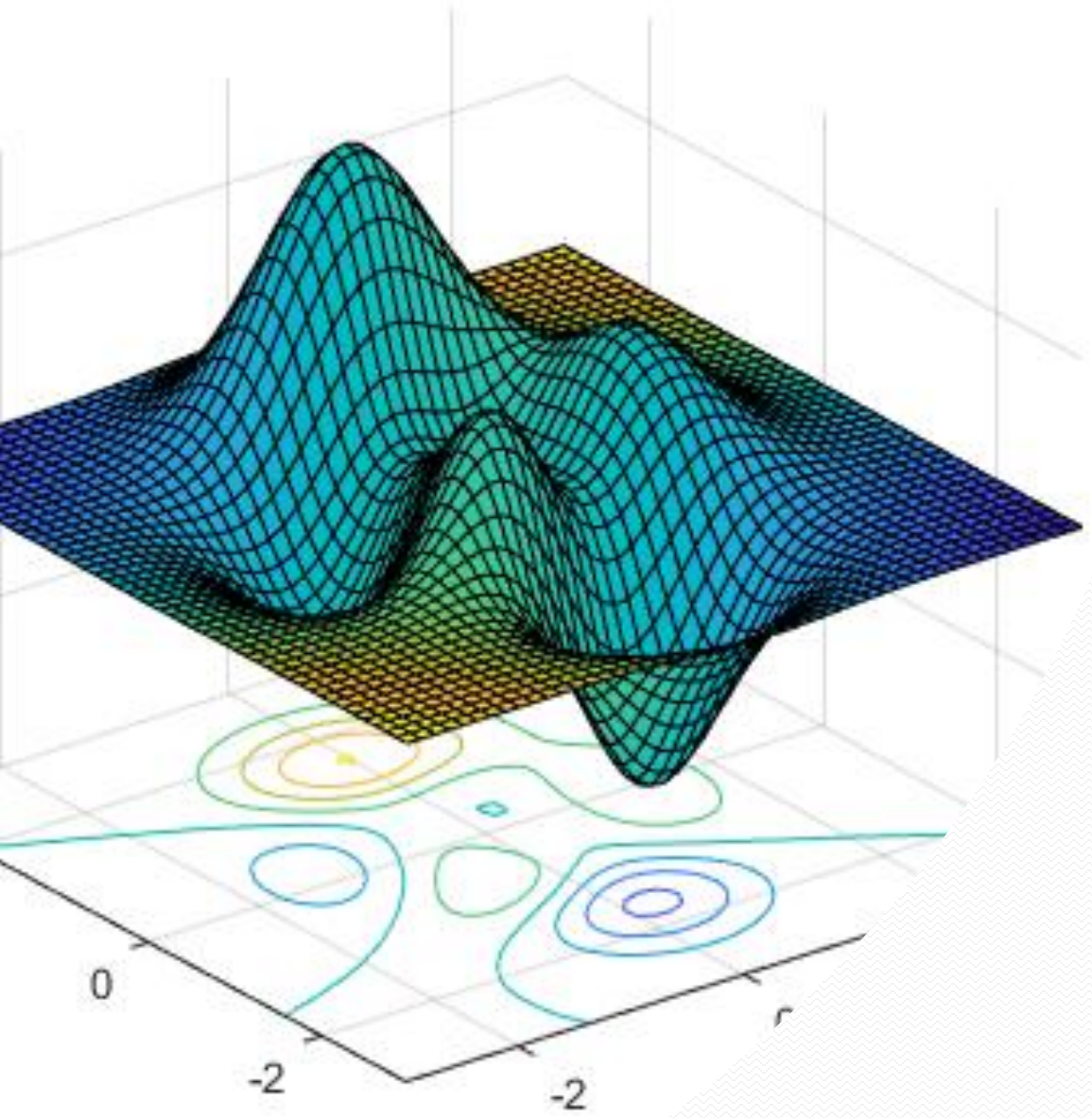
Parameters:

$$w, b$$

Cost(Loss) Function

$$\text{Ccost}(w, b) = \frac{1}{2}(\hat{y} - y)^2$$





Back Propagation

MLP TRAINING & INFERENCING

FEEDFORWARD

Predict

BACKPROPAGATE

Weight Update

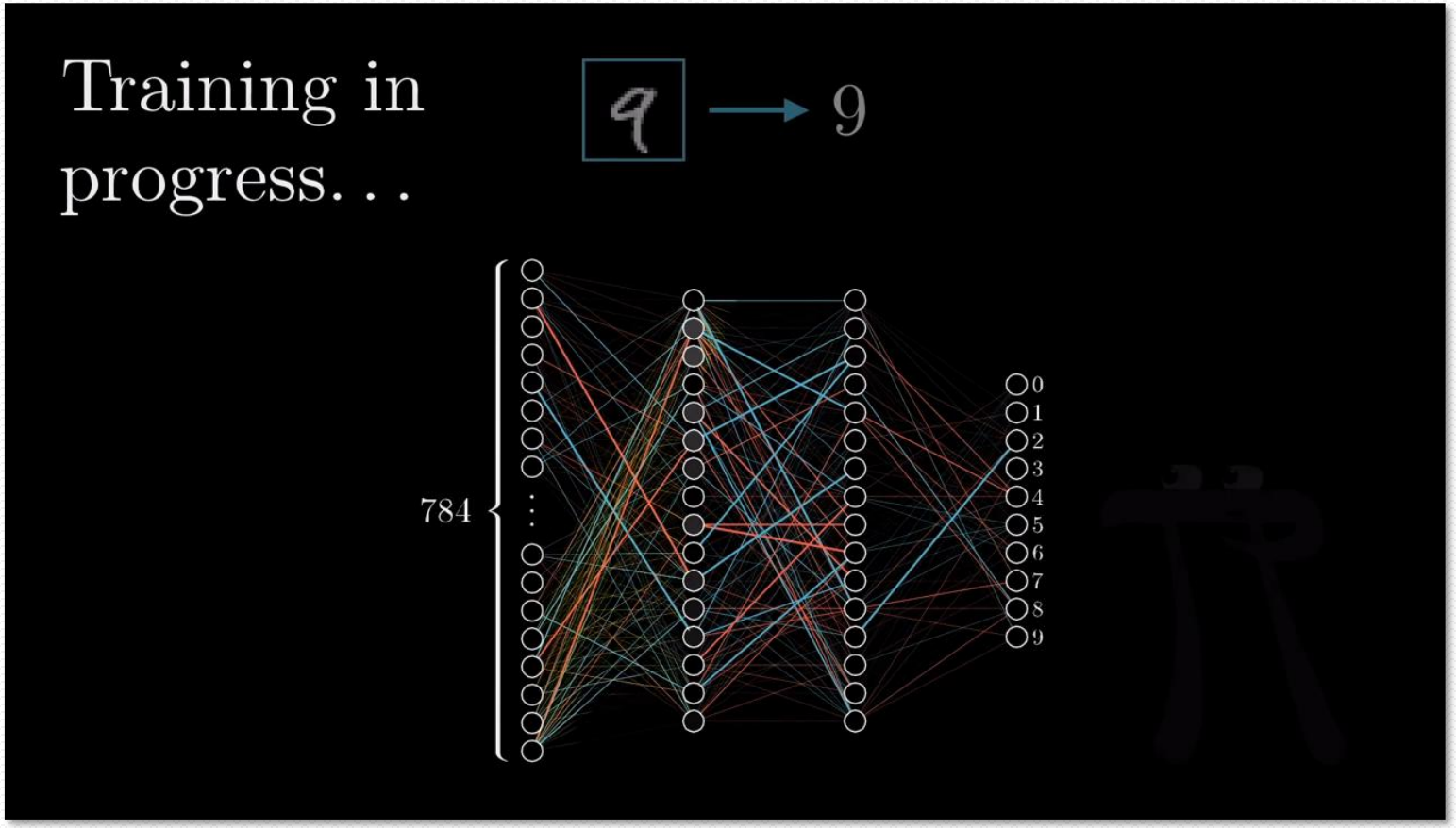
FEEDFORWARD

Predict

BACKPROPAGATE

Weight Update

...



chain rule

$$x \rightarrow f(x) = \left(x^2+1\right)^2$$

$$x \rightarrow x^2 \rightarrow x^2+1 \rightarrow \left(x^2+1\right)^2$$

$$\frac{d\left(\left(x^2+1\right)^2\right)}{dx} = \frac{d\left(\left(x^2+1\right)^2\right)}{d\left(x^2+1\right)} \frac{d\left(x^2+1\right)}{dx^2} \frac{dx^2}{dx}$$

$$f=\left(x^2+1\right)^2,\ g=x^2+1,\ h=x^2$$

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dh}\frac{dh}{dx}$$

$$\frac{d\left(\left(x^2+1\right)^2\right)}{dx} = \frac{df}{dg}\frac{dg}{dh}\frac{dh}{dx} = 2\left(x^2+1\right)\cdot 1 \cdot 2x = 4x\left(x^2+1\right)$$

chain rule

$$\frac{\partial f\left(g\left(h(x)\right)\right)}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x} = f'\left(g\left(h(x)\right)\right)g'\left(h(x)\right)h'(x) = (f \circ g \circ h)'$$

$$f(x) = \left(x^2 + 1\right)^2$$

$$f(g) = g^2 \qquad g(h) = h + 1 \qquad h(x) = x^2$$

$$f'(g) = 2g \qquad g'(h) = 1 \qquad h'(x) = 2x$$

Direct method

$$f(x) = \left(x^2 + 1\right)^2 = x^4 + 2x^2 + 1$$

$$f'(x) = 4x^3 + 4x = 4x\left(x^2 + 1\right)$$

Chain rule

$$f'(x) = 2\left(x^2 + 1\right) \cdot 1 \cdot 2x$$

$$= 4x\left(x^2 + 1\right)$$

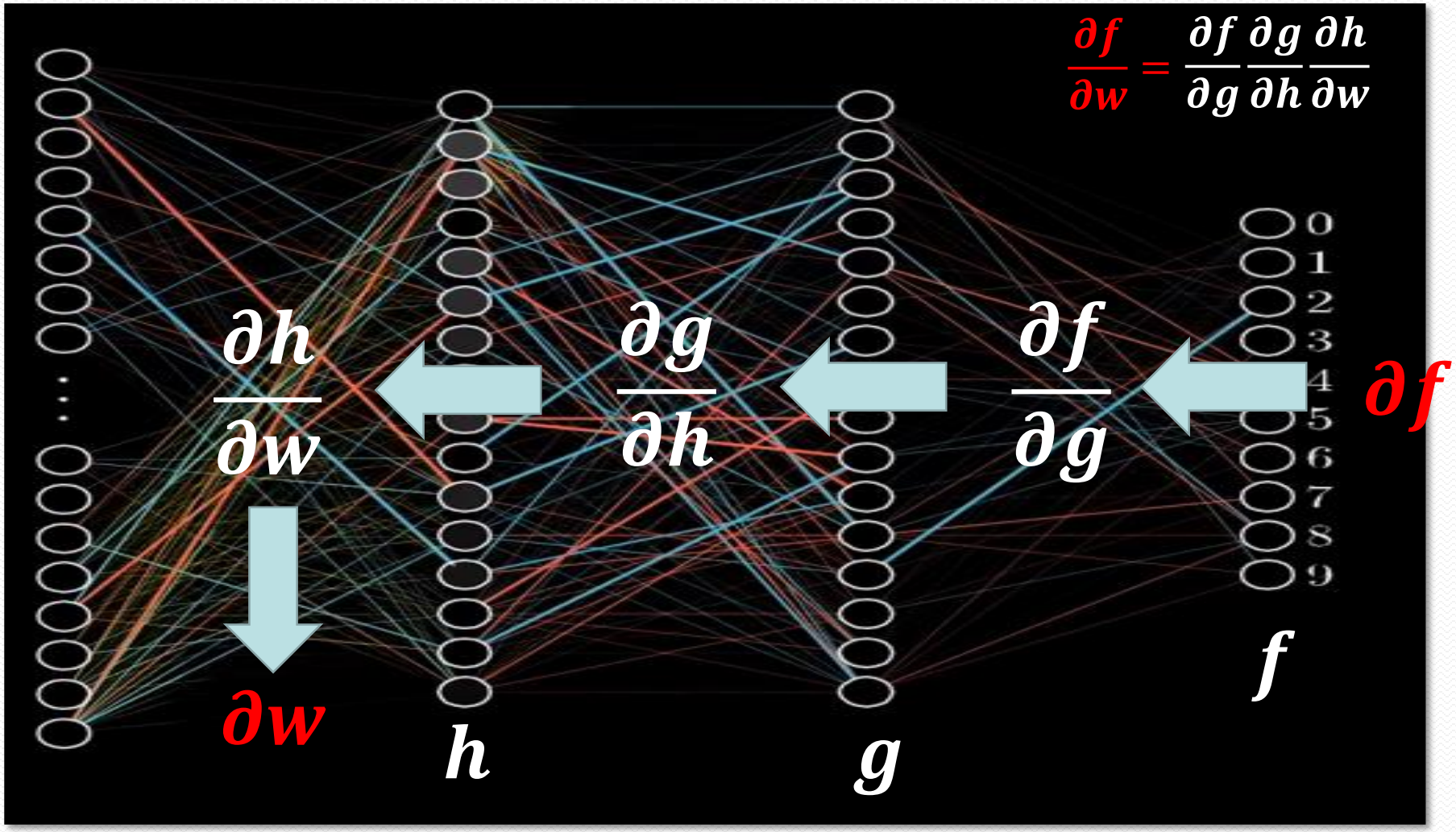
chain rule

$$(f \circ g \circ h)' = \frac{\partial f(g(h))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$$

- 체인룰은 변수가 여러 개일 때, 어떤 변수에 대한 다른 변수의 변화율을 알아내기 위해 쓰임
- Back propagation에서 주로 사용됨.

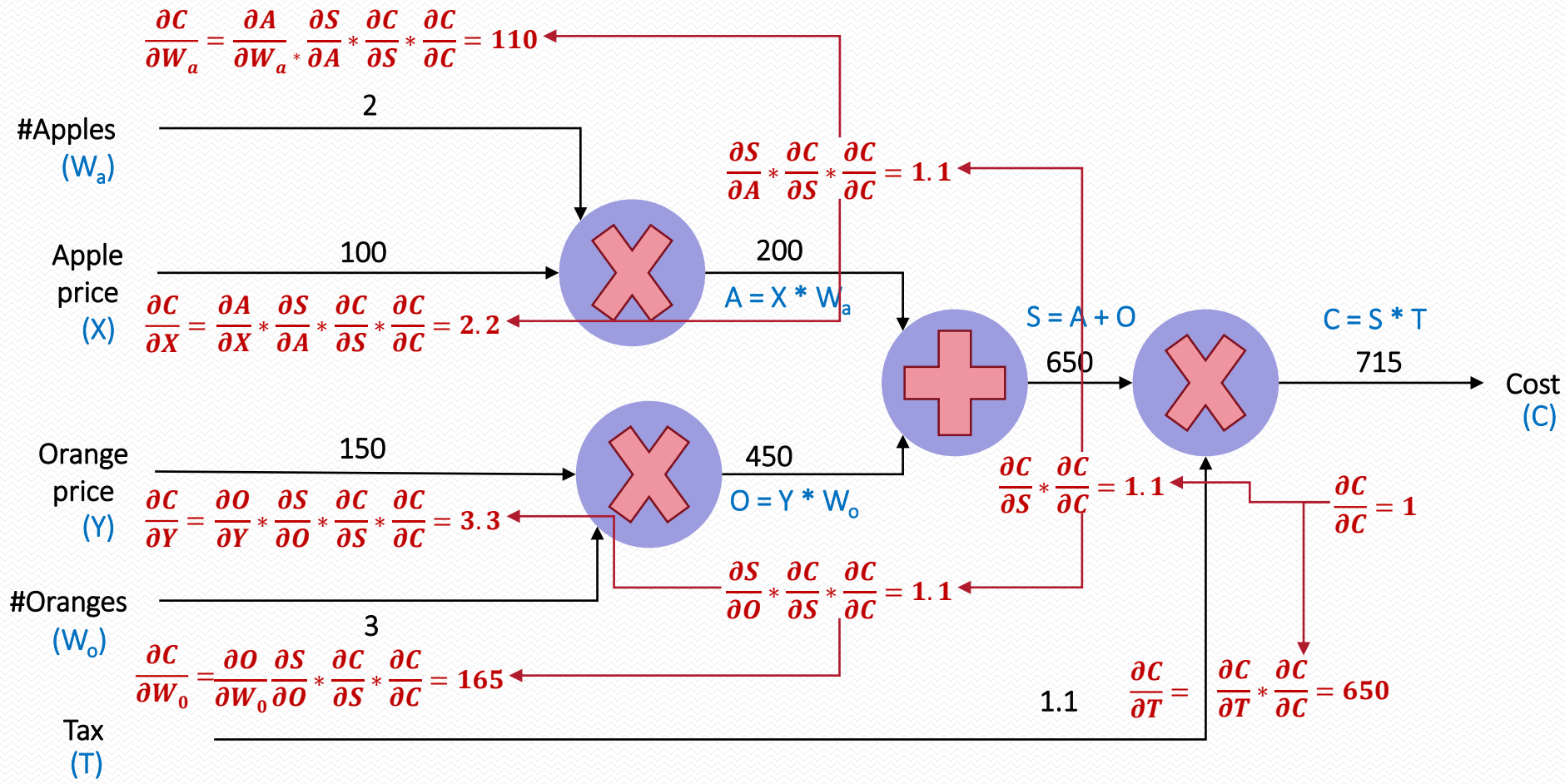
Error Back Propagation

CALCULATE THE ∂w from ∂f



Error Back Propagation

- Contributions of each parameter to final cost can be calculated from partial derivatives.

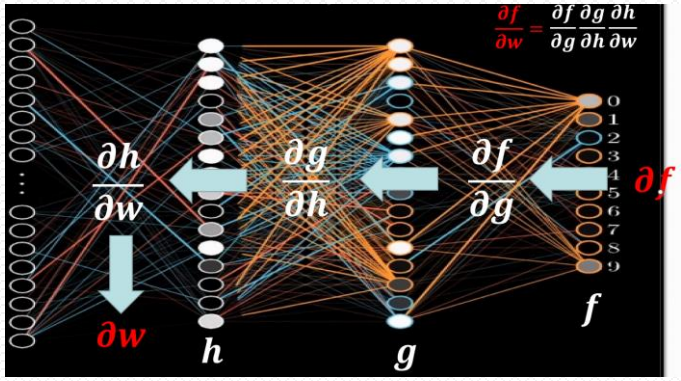


Back Propagation

```
[ ]  epochs = 20
     history = model.fit(
         train_ds,
         validation_data=val_ds,
         epochs=epochs
     )
```

```
Epoch 1/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 7s 36ms/step - accuracy: 0.2904 - loss: 1.6266 - val_accuracy: 0.5926 - val_loss: 1.0086
Epoch 2/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 8s 29ms/step - accuracy: 0.5428 - loss: 1.0985 - val_accuracy: 0.6267 - val_loss: 0.9625
Epoch 3/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.6241 - loss: 0.9589 - val_accuracy: 0.6458 - val_loss: 0.8742
Epoch 4/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.6685 - loss: 0.8739 - val_accuracy: 0.6785 - val_loss: 0.8529
Epoch 5/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.6804 - loss: 0.8385 - val_accuracy: 0.6526 - val_loss: 0.9076
Epoch 6/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.6817 - loss: 0.8259 - val_accuracy: 0.6798 - val_loss: 0.7931
Epoch 7/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7187 - loss: 0.7595 - val_accuracy: 0.6948 - val_loss: 0.8105
Epoch 8/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7385 - loss: 0.7043 - val_accuracy: 0.7166 - val_loss: 0.7484
Epoch 9/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.7274 - loss: 0.6920 - val_accuracy: 0.7098 - val_loss: 0.7514
Epoch 10/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7636 - loss: 0.6486 - val_accuracy: 0.7084 - val_loss: 0.7582
Epoch 11/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7572 - loss: 0.6466 - val_accuracy: 0.7275 - val_loss: 0.7308
Epoch 12/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.7673 - loss: 0.6086 - val_accuracy: 0.7221 - val_loss: 0.7366
Epoch 13/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7592 - loss: 0.6245 - val_accuracy: 0.6948 - val_loss: 0.8095
Epoch 14/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.8090 - loss: 0.5091 - val_accuracy: 0.7425 - val_loss: 0.7073
Epoch 15/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.7971 - loss: 0.5397 - val_accuracy: 0.7384 - val_loss: 0.7027
Epoch 16/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.8051 - loss: 0.5488 - val_accuracy: 0.7425 - val_loss: 0.7165
Epoch 17/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.7995 - loss: 0.5057 - val_accuracy: 0.7289 - val_loss: 0.7626
Epoch 18/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.8374 - loss: 0.4481 - val_accuracy: 0.7602 - val_loss: 0.7015
Epoch 19/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.8240 - loss: 0.4638 - val_accuracy: 0.7044 - val_loss: 0.8483
Epoch 20/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.8490 - loss: 0.4301
```

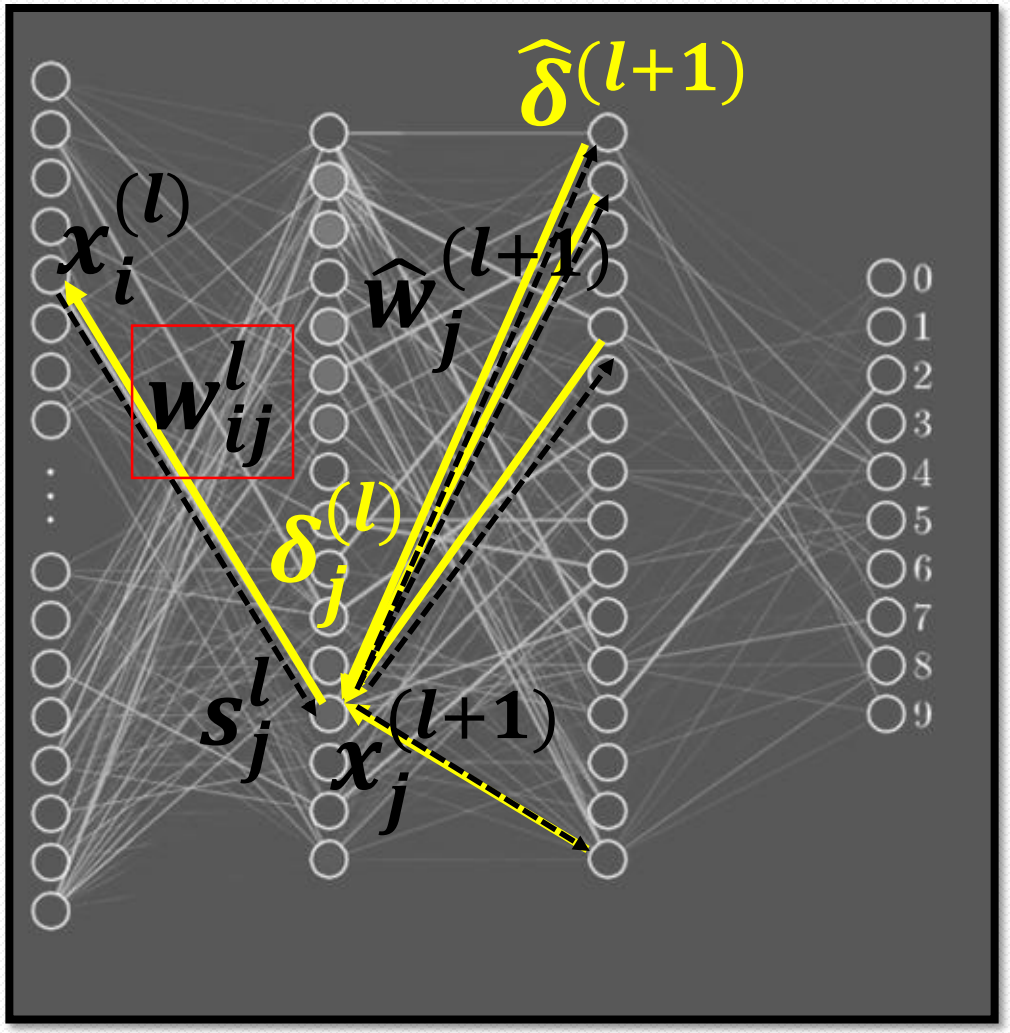
$$W_{t+1} = W_t + \alpha\left(-\frac{\partial F}{\partial W}\right)$$

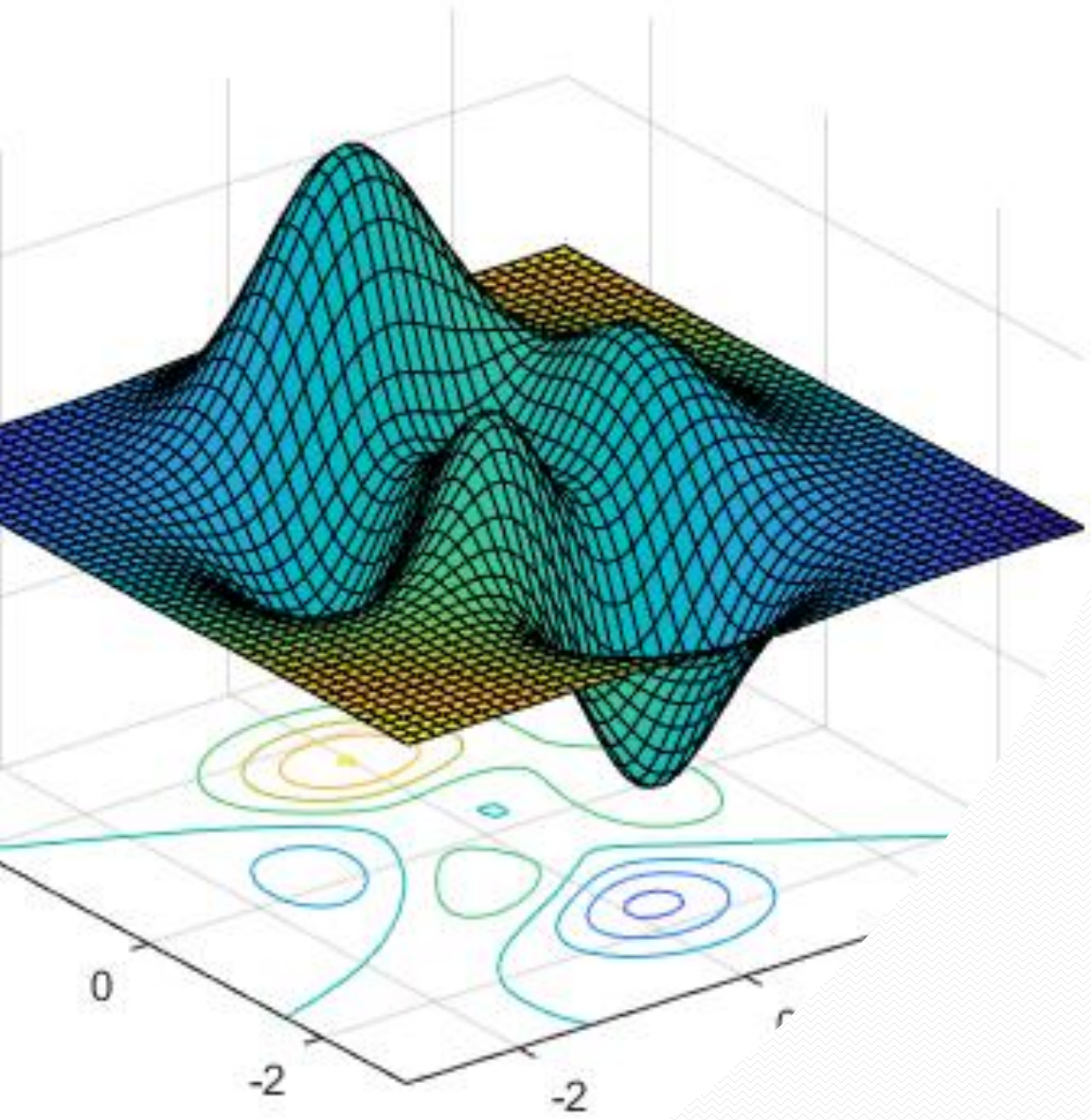


Back Propagation

$$\frac{\partial f}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} x_i^{(l)}$$

$$\delta_j^{(l)} = \left(\hat{\delta}^{(l+1)}\right)^T \hat{w}_j^{(l+1)} \frac{\partial x_j^{(l+1)}}{\partial s_j^{(l)}}$$





CONFUSION MATRIX

CONFUSION MATRIX

QUANTIFIABLE MEASURE THAT IS USED TO TRACK AND ASSESS THE STATUS OF A SPECIFIC PROCESS

Model classification & answer → TRUE / FALSE

- True Positive / TP
- False Positive / FP
- False Negative / FN
- True Negative / TN

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

		ACTUAL	
		POSITIVE	NEGATIVE
PREDICTED	POSITIVE	**TRUE** POSITIVE	FALSE POSITIVE
	NEGATIVE	FALSE NEGATIVE	**TRUE** NEGATIVE

PRECISION / RECALL

		ACTUAL	
		POSITIVE	NEGATIVEW
PREDICTED	POSITIVE	TRUE POSITIVE	FALSE POSITIVE
	NEGATIVE	FALSE NEGATIVE	TRUE NEGATIVE

Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\#\text{ predictions}}$$

Proportion of what the actual true among the model classifies as true

Accuracy rate

Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\#\text{ ground truths}}$$

Proportion of what the model predicted to be true among the actual true

Detection rate

Precision = 100%

9	0
1	

		ACTUAL	
		강아지	고양이
PREDICTED	강아지	9	0
	고양이	1	90

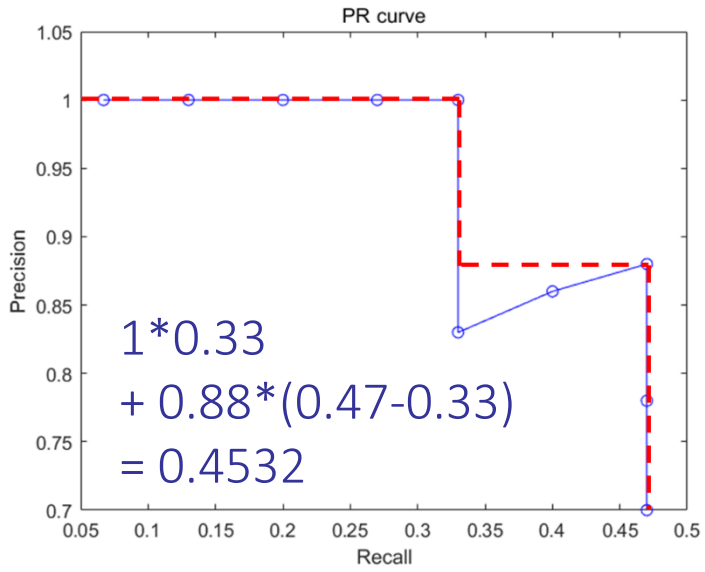
Recall = 90%

9	0
1	

AP AVERAGE PRECISION / MAP MEAN AP

Measurement that combines recall and precision for ranked retrieval results

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above

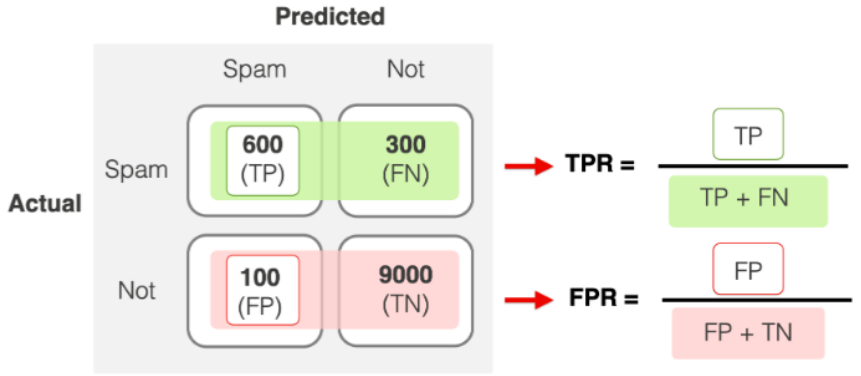


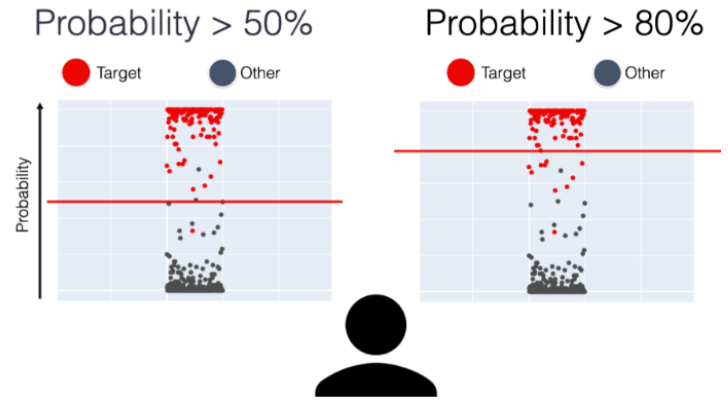
$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i$$

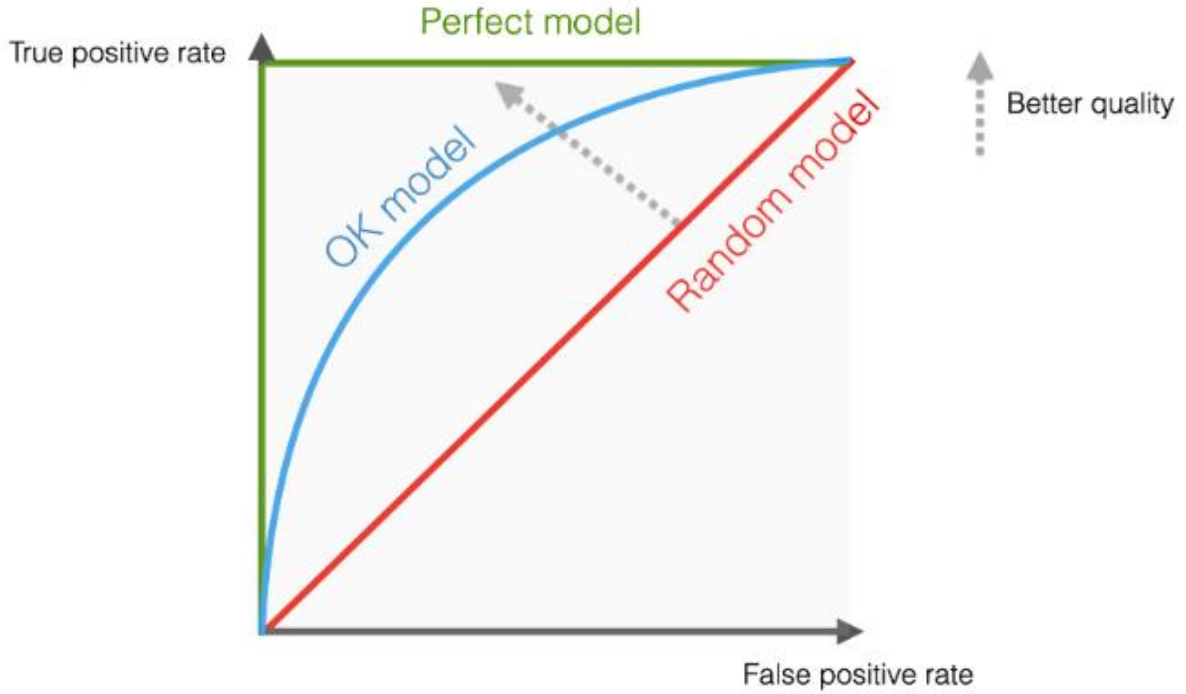
ROC Receiver Operation Characteristic Curve / AUC Area Under the Curve

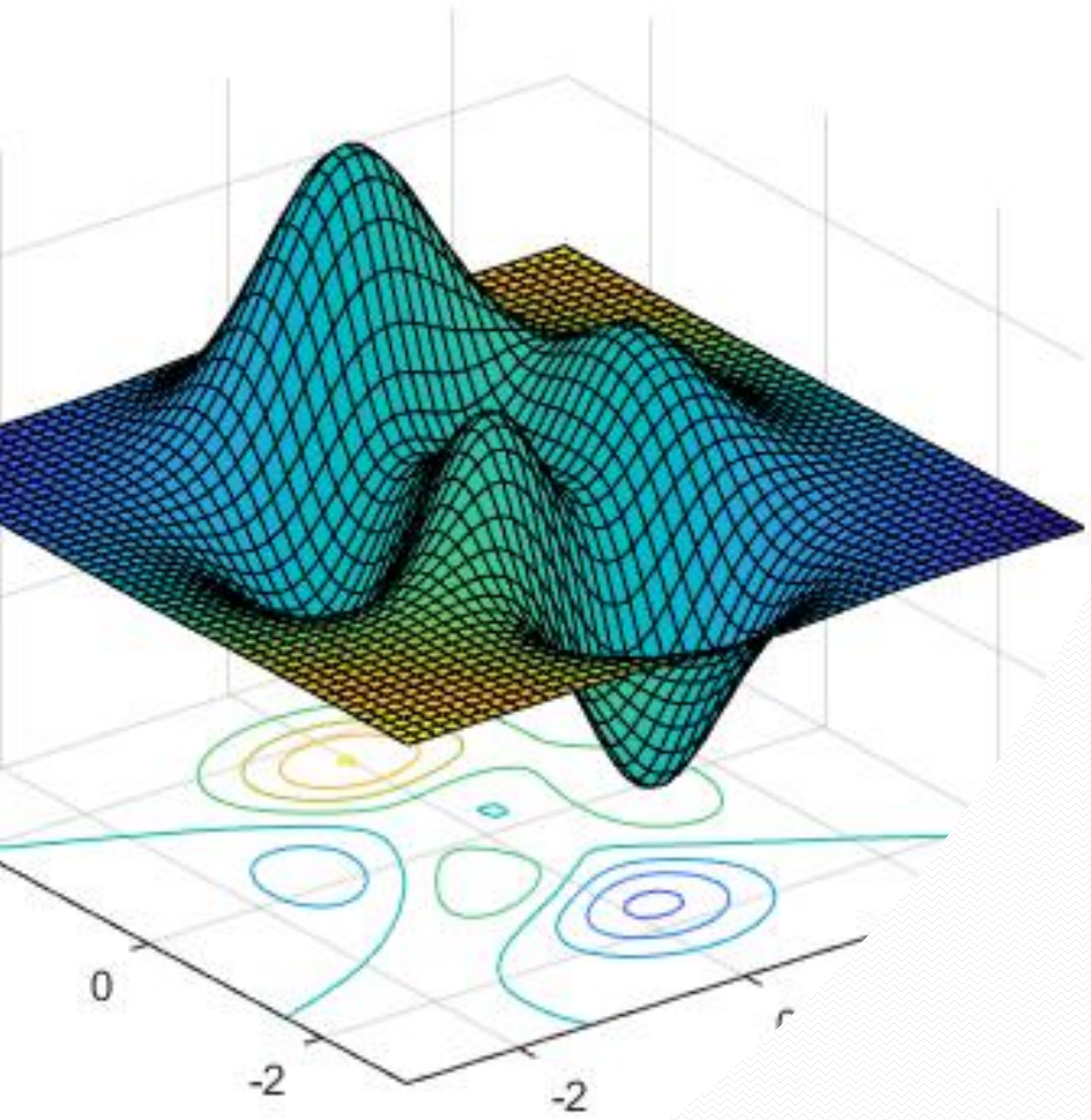
이진 분류 모델의 성능을 평가하는 지표로, 다양한 임곗값에서의 모델 성능을 시각적으로 보여줌

분류 모델의 성능을 평가하고, 최적의 임곗값을 설정하는 데 사용









Deep-dive

Optimizer

Optimizer: Gradient descent algorithm

```
# let's build the CNN model

cnn = Sequential()

#Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)))

#Pooling
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 2nd Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu"))

# 2nd Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# Flatten the layer
cnn.add(Flatten())

# Fully Connected Layers
cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'sigmoid', units = 1))

# Compile the Neural network
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

6) Derivatives : 1 variable → 2 variable

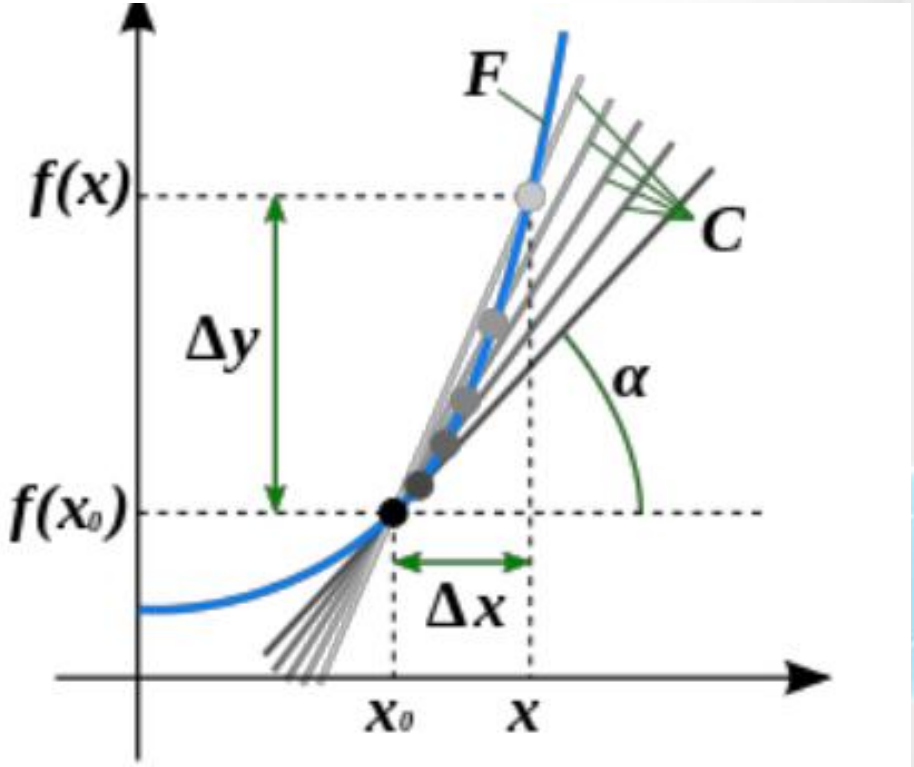
- First order

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

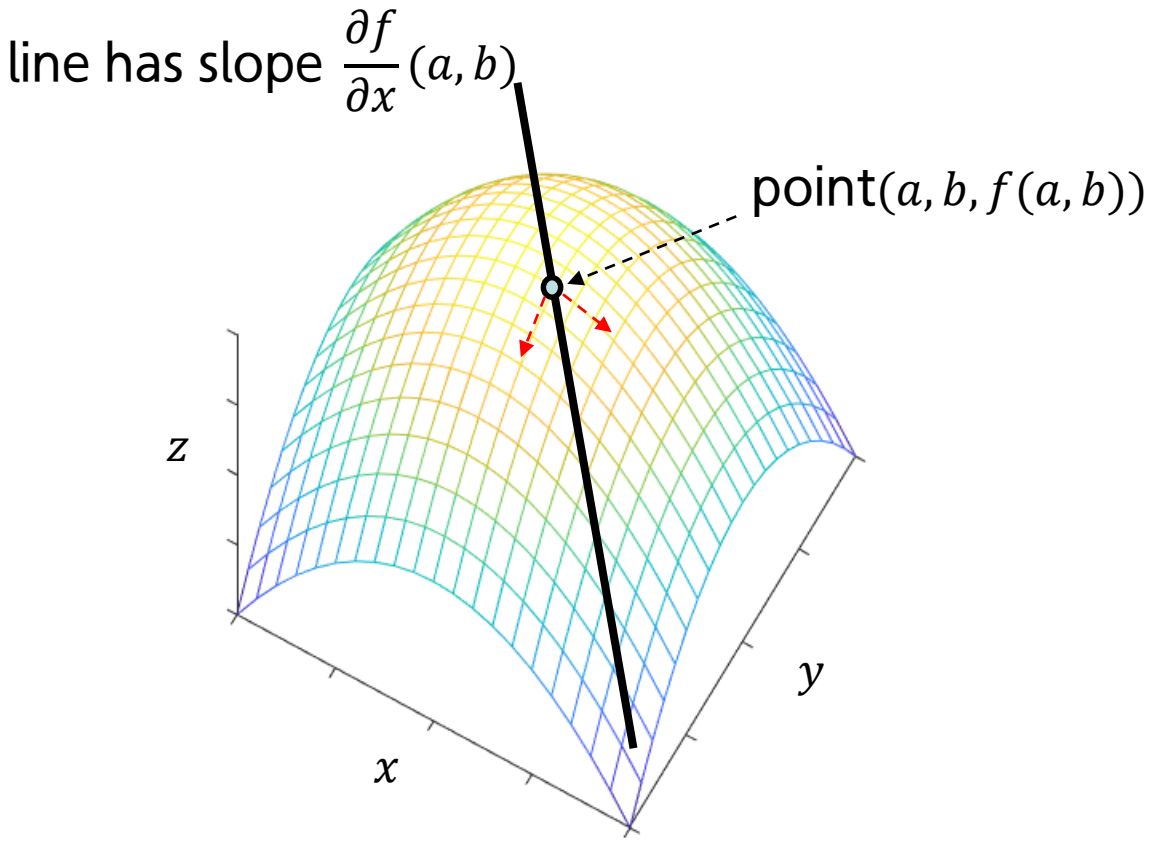
"편미분으로 바로 차원을 늘릴 수 있을까?"

$$\frac{\partial f(x_0, y_0)}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x, y_0) - f(x_0, y_0)}{\Delta x}$$

$$\frac{\partial f(x_0, y_0)}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x_0, y_0 + \Delta y) - f(x_0, y_0)}{\Delta y}$$



6) Derivatives : multivariable of partial derivatives



$$\nabla A(x,y) = \frac{\partial A}{\partial x}\hat{x} + \frac{\partial A}{\partial y}\hat{y}$$

"각 좌표축이 독립적이므로 편미분 사용가능"

$$\frac{\partial f(x_0,y_0)}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x_0+\Delta x, y_0) - f(x_0,y_0)}{\Delta x}$$

$$\frac{\partial f(x_0,y_0)}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x_0, y_0+\Delta y) - f(x_0,y_0)}{\Delta y}$$

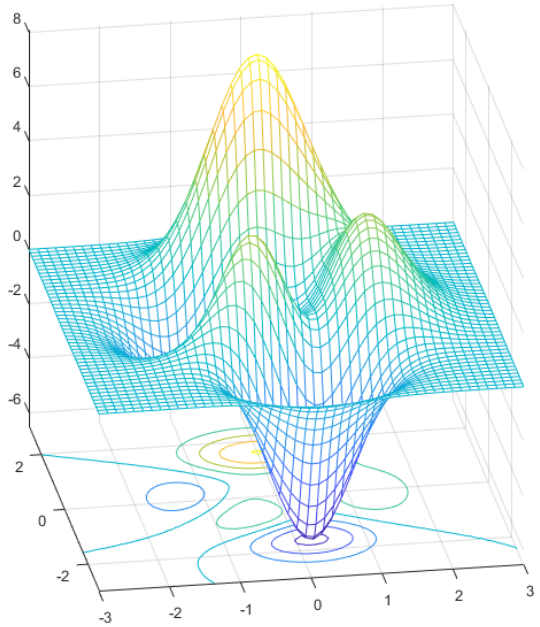
7) Gradient(∇) : Derivatives of vector

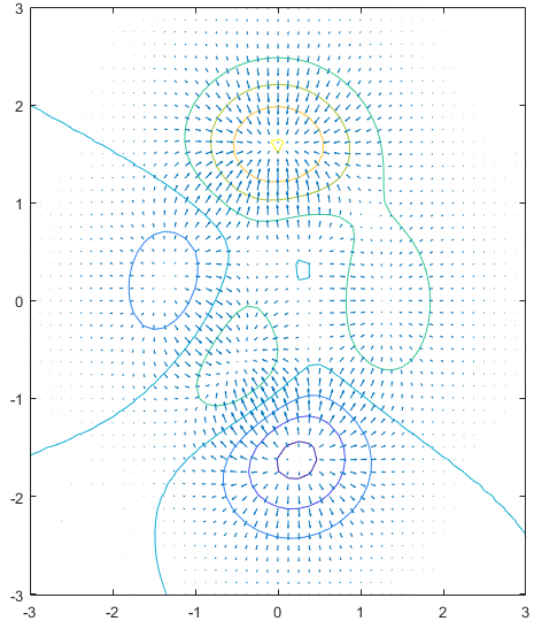
- **The gradient is a vector of partial derivatives representing the rate and direction of the steepest ascent of a function.**
  - gradient는 편미분값이 벡터 표현임

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

  - n차원에서는

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

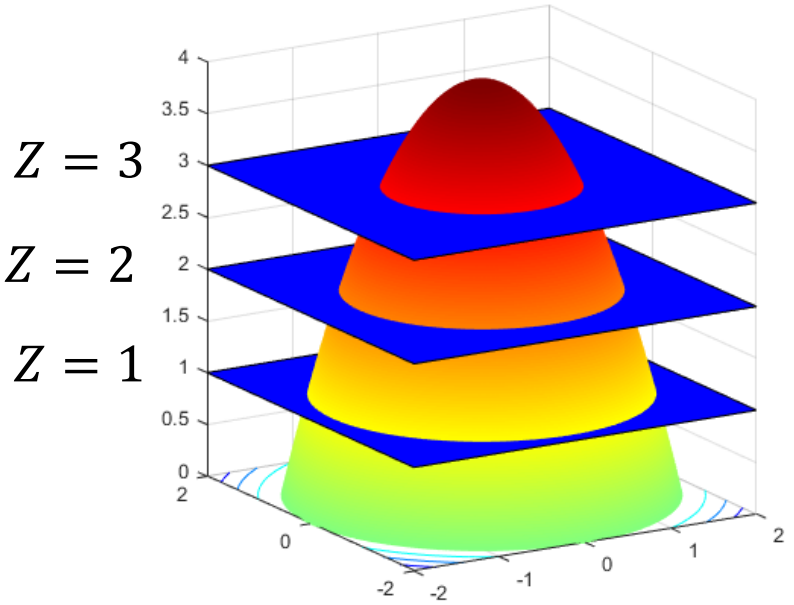


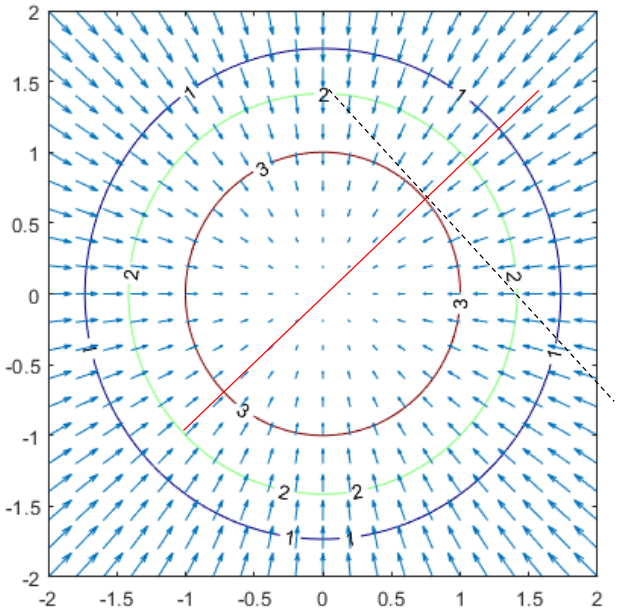


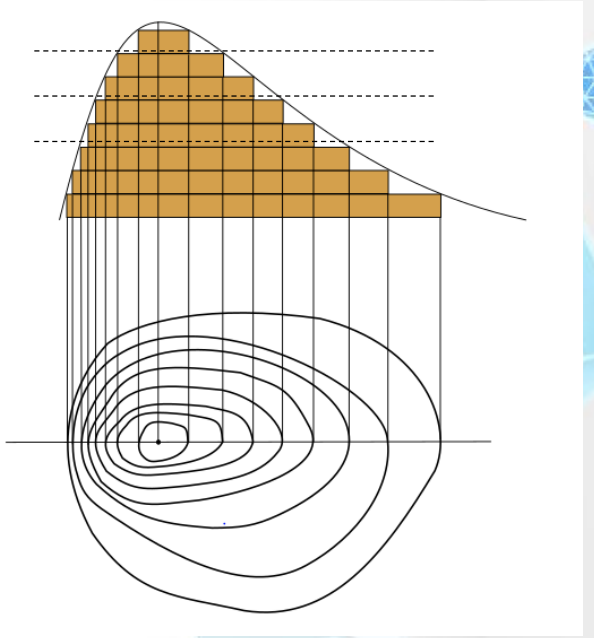
7) Gradient(∇) : Derivatives of vector

$$f(x, y) = 4 - x^2 - y^2$$

- $X = 1, y = 1$에서 가장 가파른 방향은 어느 방향일까?







Gradient descent algorithm

Solve the following minimization problem : for $x \in \mathbf{R}^n$,

$$\min_x f(x)$$

To solve the problem, we use descent method framework : For given initial $x^{(0)}$,

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} \Delta x^{(k)}$$

1. $\alpha^{(k)}$: Learning Rate / Step Size
2. $\Delta x^{(k)}$: Search Direction

→ α(constant)
$-\nabla f(x^k)$

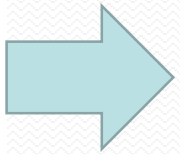
- 비용 함수 최소화
- Gradient descent는 주로 최적화 문제에 사용된다.
- 주어진 비용 함수, 비용 (W, b)에 대해 비용을 최소화하기 위해 W, b를 찾는다.
- 일반적인 함수 : 비용 (w1, w2,...)에 적용 가능

Gradient descent algorithm

Adaptive Learning rate scheme

- Learning rate가 작으면 안정적이지만 너무 느리고
- Learning rate가 크면 빠르지만 불안정
- 두가지 상황에서 단점을 빼고 장점만 가져가는 방법은?

1. $\alpha^{(k)}$: Learning Rate / Step Size
2. $\Delta x^{(k)}$: Search Direction



??? $\boldsymbol{\alpha}$

??? $-\nabla \boldsymbol{f}(\boldsymbol{x^k})$

Gradient descent optimization Algo.

BGD (Batch Gradient Descent)

- FULL TRAIN DATASET, REQUIRES A LOT OF COMPUTATION

SGD (Stochastic Gradient Descent)

- USE Some of training sample, calculate gradient

Momentum

- Method that helps accelerate SGD in the relevant direction and dampens oscillations by adding update vector of the past time step to the current update vector

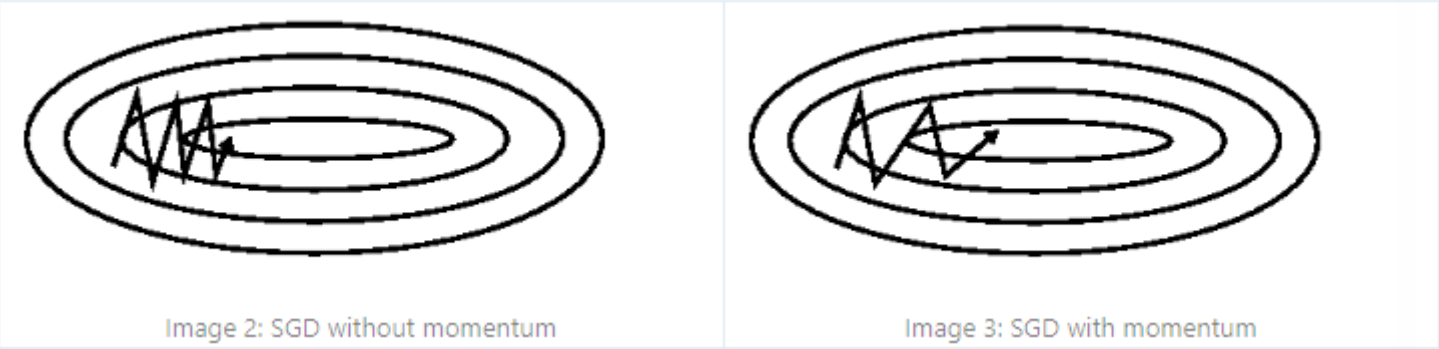


Image 2: SGD without momentum

Image 3: SGD with momentum

Gradient descent optimization Algo.

Adagrad (Adaptive Gradient)

- Adapts learning rate to the parameters, performing smaller updates(i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.

RMSProp

- Need to resolve Adagrad's radically diminishing learning rates. RMSprop divides the learning rate by an exponentially decaying average of squared gradients

Adam (Adaptive Moment Estimation)

- Another approach to compute adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients similar to momentum

Gradient descent algorithm

Momentum : 1945

- 방식은 진동하는 현상을 겪을 때 이를 해결하도록 해 줌
- 이전 gradient들도 계산에 포함해서 현재 파라미터를 업데이트
- 이전 gradient들을 모두 동일한 비율로 포함시키지는 않고 비율을 감소시켜줌

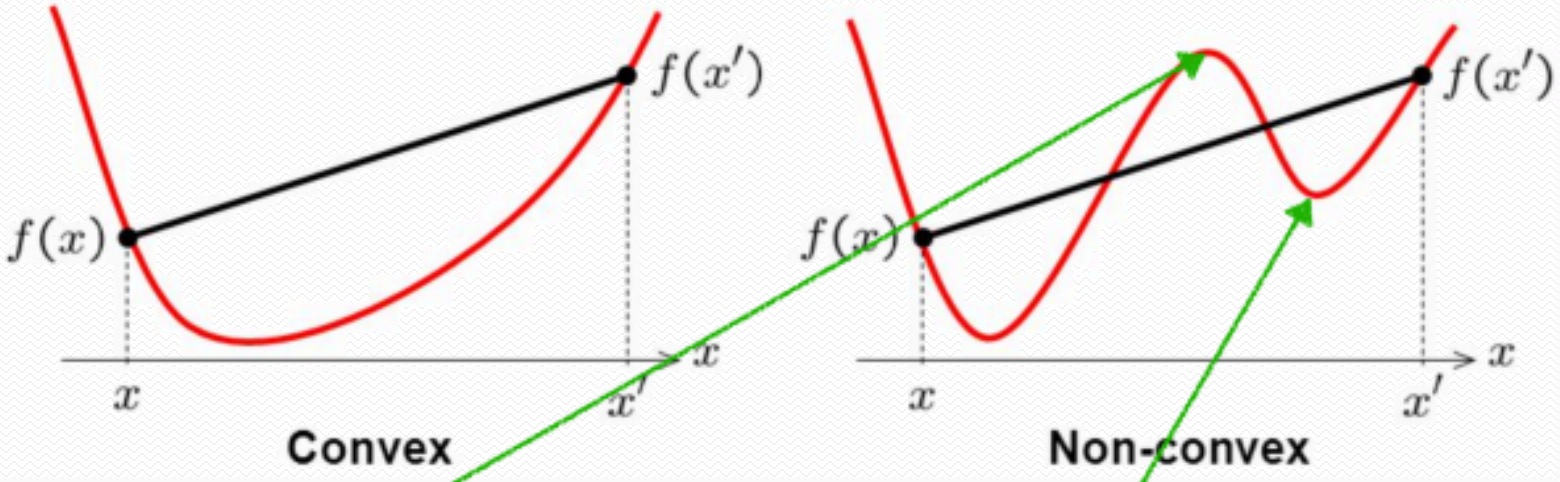
$$v_{k+1} = \alpha v_k - \epsilon \nabla f(x_k)$$
$$x_{k+1} = x_k + v_{k+1}$$

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Gradient descent algorithm

Momentum : 1945



- 미분값이 0이라고 무조건 최솟값은 아닙니다.
- 최댓값일 수도 있습니다.
- 최솟값은 아니지만 미분 값이 0인 곳을 Local Minimum이라고 합니다.

Gradient descent algorithm

Adagrad : 2011

- 각 weight마다 다른 learning rate를 줌
- 변화한 누적거리를 계속 저장하여 learning rate에 반영
- 변화한 누적 거리가 큰 weight는 작은 learning rate를 반영
- 변화한 누적 거리가 작은 weight는 큰 learning rate를 반영

$$r_{k+1} = r_k + \nabla f(x_k) \odot \nabla f(x_k)$$

$$x_{k+1} = x_k - \frac{\epsilon}{\delta + \sqrt{r_{k+1}}} \odot \nabla f(x_k)$$

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.

end while

Gradient descent algorithm

RMSprop : 2012

- Adagrad와 비슷한 알고리즘
- 아주 오래저 궤적의 영향을 줄이고 가까운 시간의 궤적의 영향을 높이기 위해 decay를 추가

$$r_{k+1} = \rho r_k + (1-\rho)\nabla f(x_k) \odot \nabla f(x_k)$$

$$x_{k+1} = x_k - \frac{\epsilon}{\sqrt{\delta + r_{k+1}}} \odot \nabla f(x_k)$$

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$.

Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.

end while

Gradient descent algorithm

Adam : 2014

- Momentum과 RMSprop를 Hybrid한 방법
- Hyper-parameter설정에 가장 robust하다고 알려짐 (Bengio, 2016)

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

$$
\begin{aligned}
s_{k+1} &= \rho_1 s_k + (1 - \rho_1)\nabla f(x_k) \\
r_{k+1} &= \rho_2 r_k + (1 - \rho_2)\nabla f(x_k) \odot \nabla f(x_k) \\
\hat{s}_{k+1} &= \frac{s_{k+1}}{1 - \rho_1^t} \\
\hat{r}_{k+1} &= \frac{r_{k+1}}{1 - \rho_2^t} \\
x_{k+1} &= x_k - \frac{\epsilon}{\sqrt{\hat{r}_{k+1}} + \delta} \hat{s}_{k+1}
\end{aligned}
$$

THANK YOU