

# Introduction to Deep Learning & Neural Networks

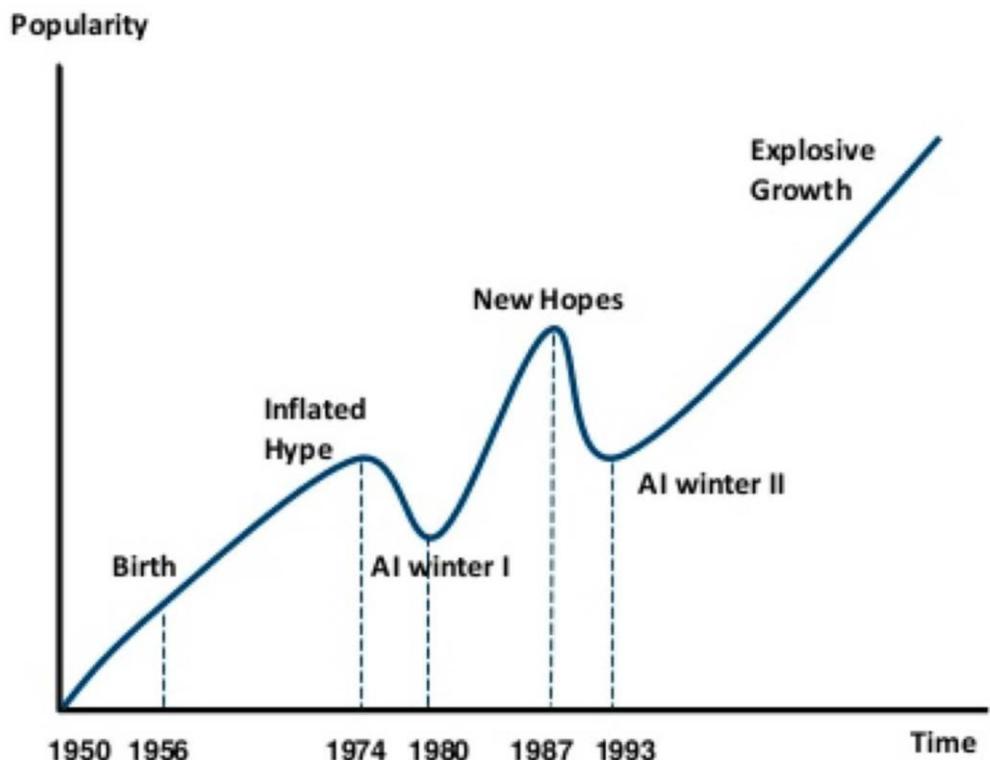


2001: A Space Odyssey(1968)



# History of AI

- 1943: Neural networks
- 1957-62: Perceptron
- 1970-86: Backpropagation, RBM, RNN
- 1979-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: “Deep Learning”, DBN
- 2009: ImageNet + AlexNet
- 2014: GANs
- 2016-17: AlphaGo, AlphaZero
- 2017: 2017-19: Transformers

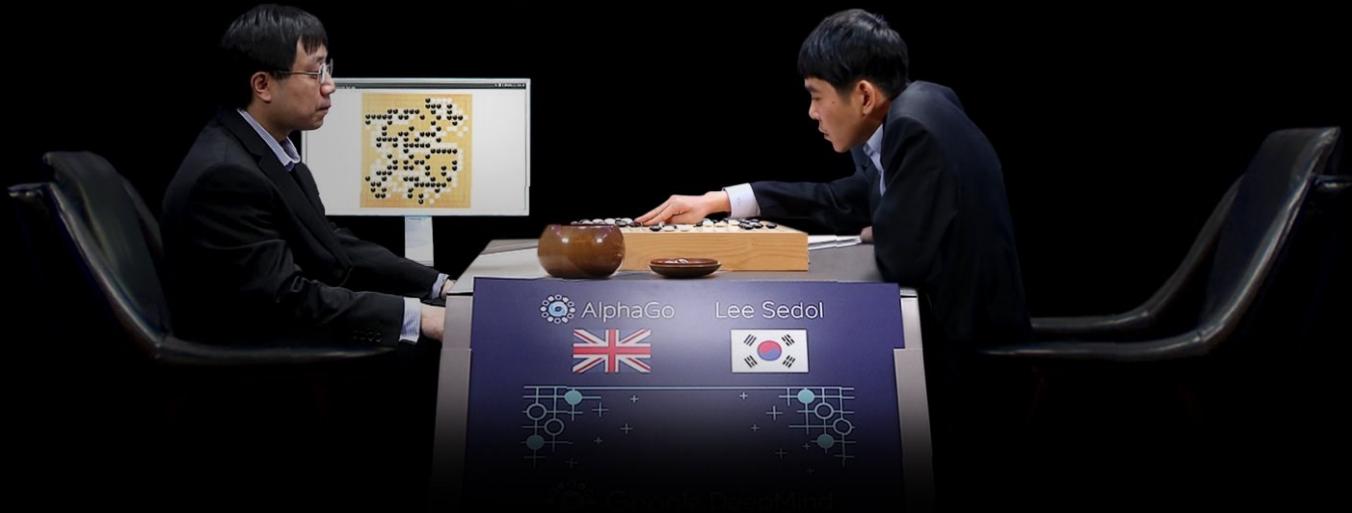


# AI의 현재

AI의 현재



# AI의 현재



# ALPHAGO

# AlphaGo

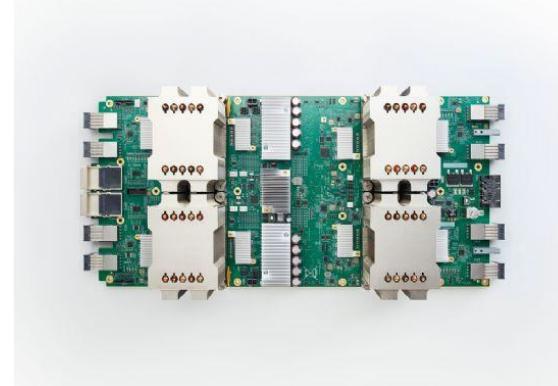
TPU Server used  
against Lee Sedol



AlphaGo Lee – 4승 1패



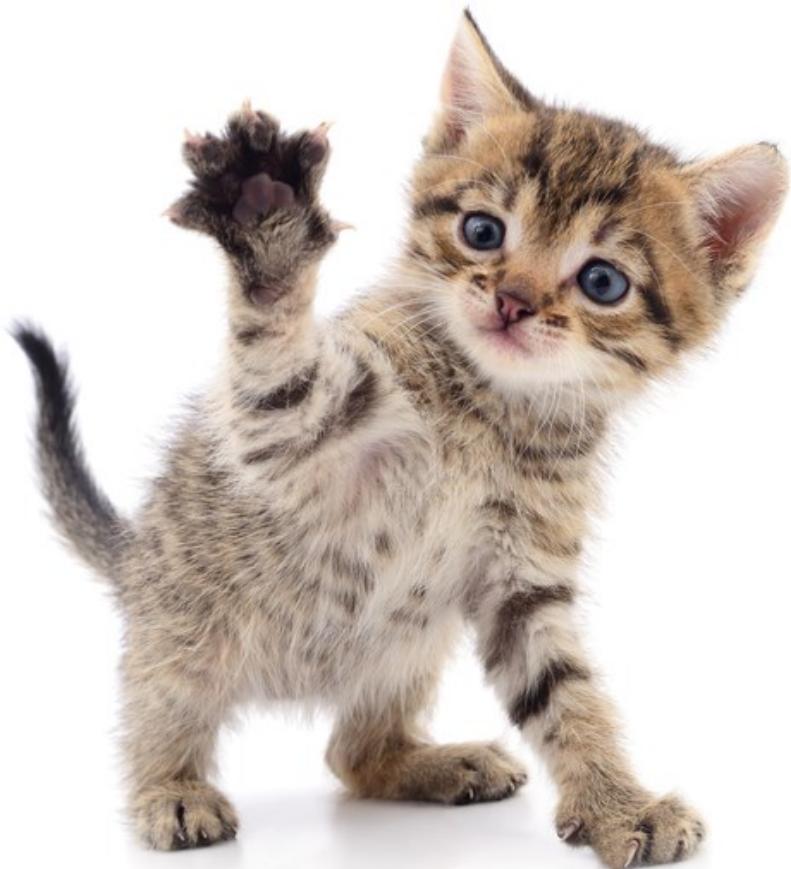
TPU Board used  
against Ke Jie



AlphaGo Master – 3승 0패



# 이것은 무엇인가요?



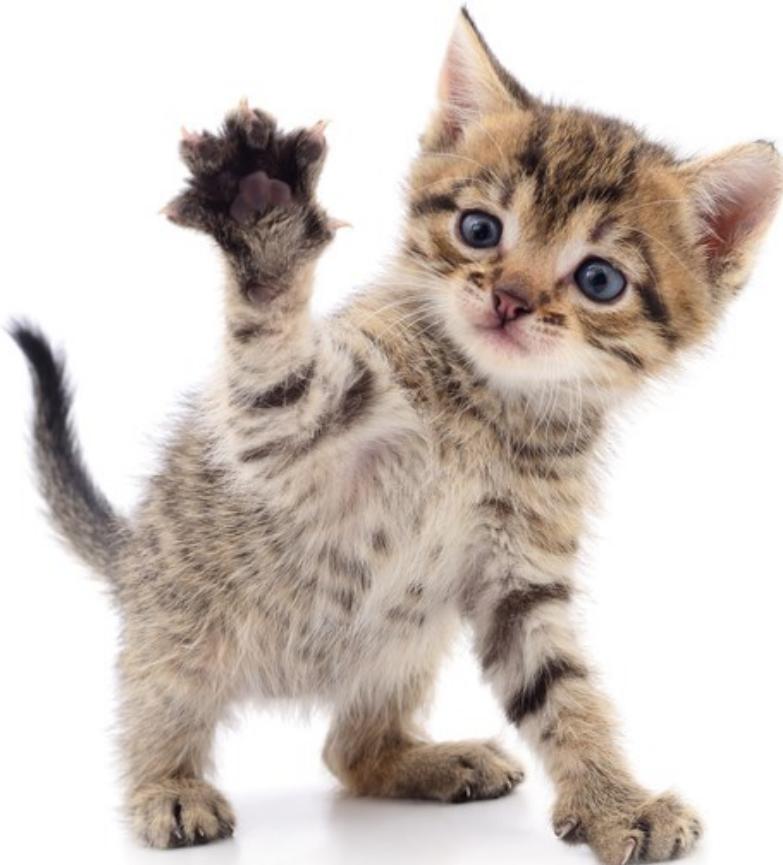
고양이!

# 이것은 무엇인가요?

```
array([[[0.03921569, 0.03529412, 0.02352941, 1.  
       [0.2509804 , 0.1882353 , 0.20392157, 1.  
       [0.4117647 , 0.34117648, 0.37254903, 1.  
       ...  
       [0.20392157, 0.23529412, 0.17254902, 1.  
       [0.16470589, 0.18039216, 0.12156863, 1.  
       [0.18039216, 0.18039216, 0.14117648, 1.  
  
       [[0.1254902 , 0.11372549, 0.09411765, 1.  
       [0.2901961 , 0.2509804 , 0.24705882, 1.  
       [0.21176471, 0.2      , 0.20392157, 1.  
       ...  
       [0.1764706 , 0.24705882, 0.12156863, 1.  
       [0.10980392, 0.15686275, 0.07843138, 1.  
       [0.16470589, 0.20784314, 0.11764706, 1.  
  
       [[0.14117648, 0.12941177, 0.10980392, 1.  
       [0.21176471, 0.1882353 , 0.16862746, 1.  
       [0.14117648, 0.13725491, 0.12941177, 1.  
       ...  
       [0.10980392, 0.15686275, 0.08627451, 1.  
       [0.0627451 , 0.08235294, 0.05098039, 1.  
       [0.14117648, 0.2      , 0.09803922, 1.  
  
       ...
```

고양이!

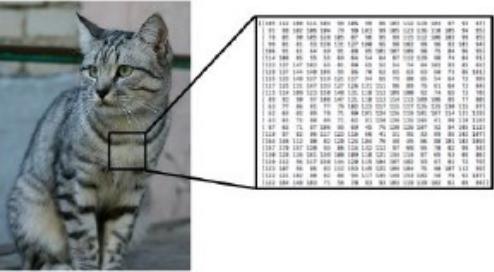
# 이것은 왜 고양이인가요?



- 귀가 삼각형 모양이라서
- 털이 많아서
- 몸집에 비해 꼬리가 길어서
- 다리가 4개라서
- ...

# 고양이란 무엇인가?

Viewpoint

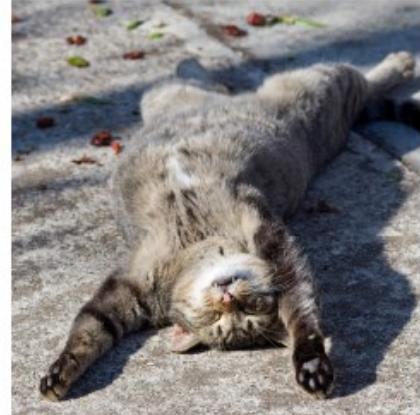


Illumination



[This image is CC0 1.0 public domain](#)

Deformation



[This image by Umberto Salvagnin is licensed under CC-BY 2.0](#)

Occlusion



[This image by jonsson is licensed under CC-BY 2.0](#)

Clutter



[This image is CC0 1.0 public domain](#)

Intraclass Variation



[This image is CC0 1.0 public domain](#)

# 강아지란 무엇인가?



Sheepdog or mop?



Chihuahua or muffin?

# AI, ML, DL

## ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior



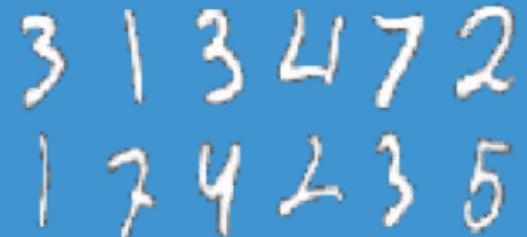
## MACHINE LEARNING

Ability to learn without explicitly being programmed



## DEEP LEARNING

Extract patterns from data using neural networks



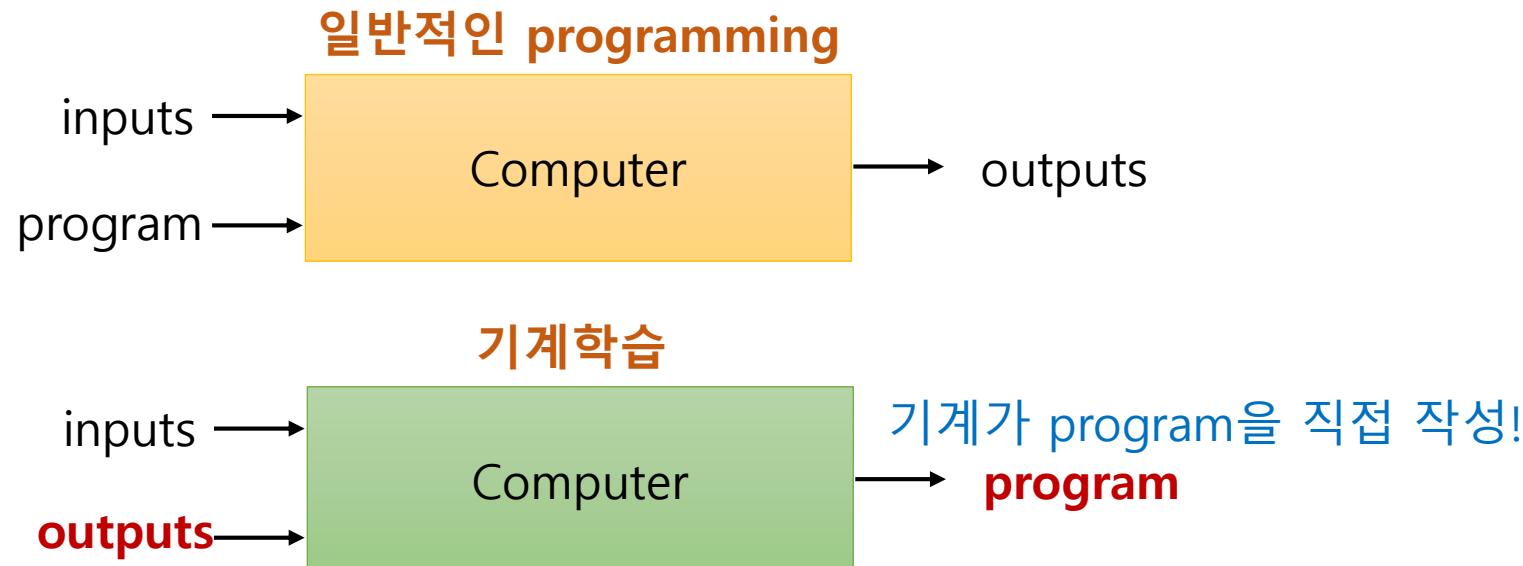
# 기계학습(Machine Learning)

## – Data Driven Approach

- 기계학습(Machine Learning)

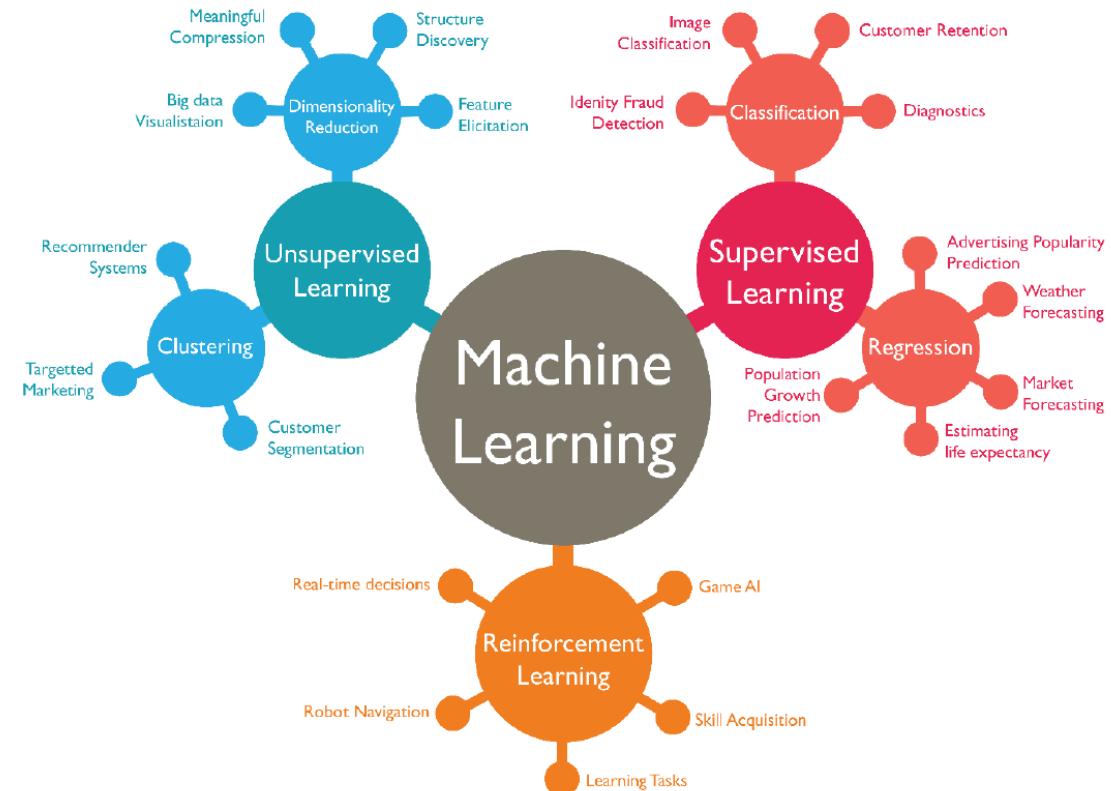
Machine learning is the subfield of [computer science](#) that "gives computers the ability to learn without being explicitly programmed"

기계 학습은 "컴퓨터에 명시적으로 프로그래밍하지 않고 학습 할 수 있는 능력을 부여하는" 컴퓨터 과학의 하위 분야입니다.



# Machine Learning의 종류

- Supervised Learning(지도학습)
  - 입력 data와 정답을 이용한 학습
  - 분류(classification), 회귀(regression)
- Unsupervised Learning(비지도학습)
  - 입력 data 만을 이용한 학습
  - 군집화(clustering), 압축(compression)
- Reinforcement Learning(강화학습)
  - Trial and error를 통한 학습
  - Action selection, policy learning



# Supervised Learning

## – Regression vs Classification



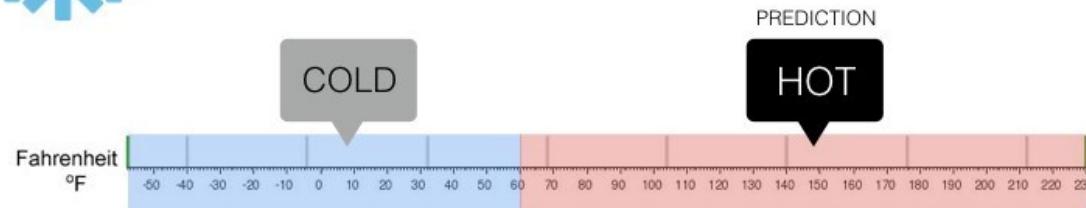
### Regression

What is the temperature going to be tomorrow?



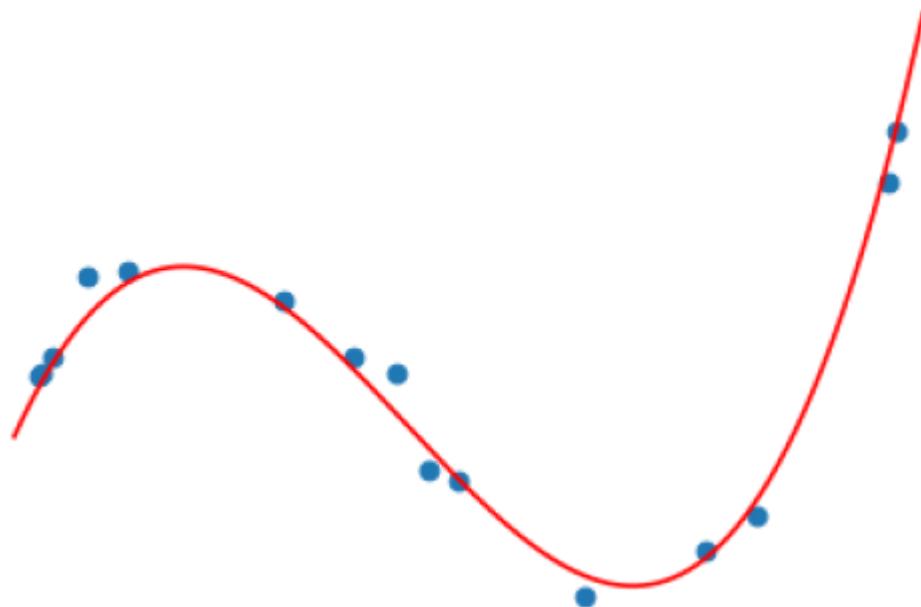
### Classification

Will it be Cold or Hot tomorrow?

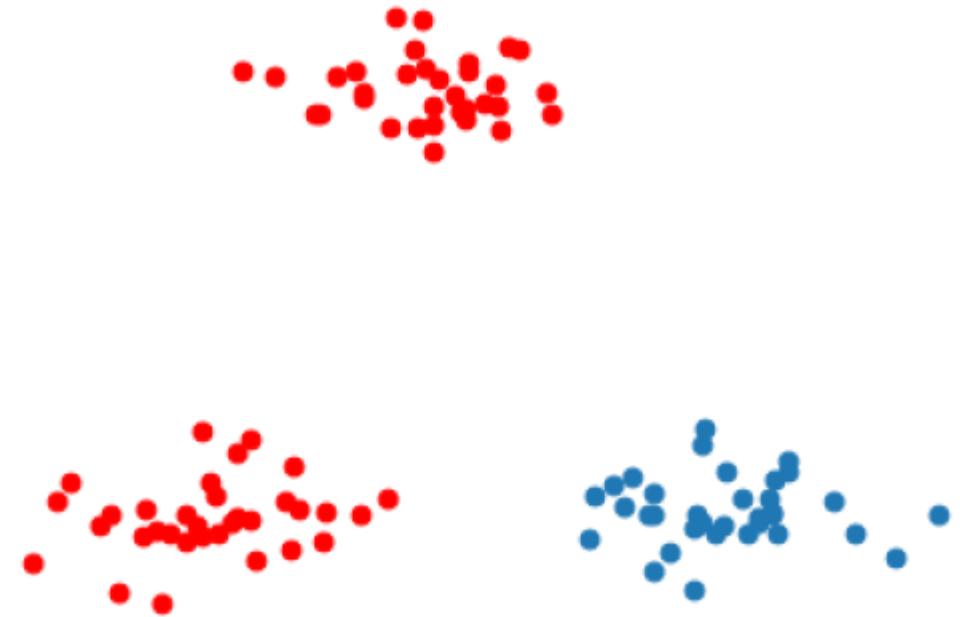


# Supervised Learning

## – Regression vs Classification

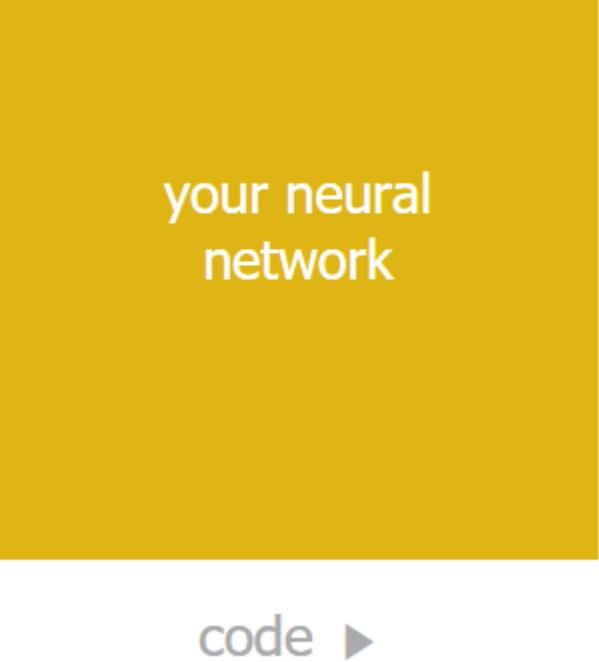


Regression aims at estimating relationships among  
(using continuous) variables



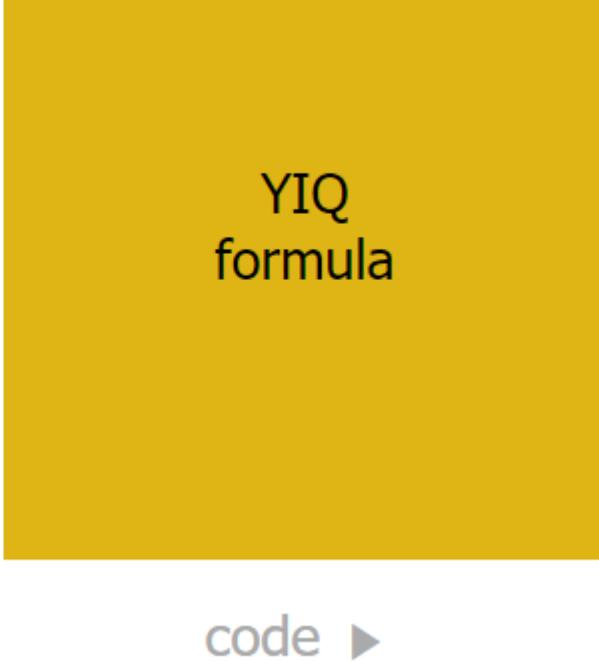
Classification consists in identifying a decision boundary  
between objects of distinct classes

# Supervised Learning



your neural  
network

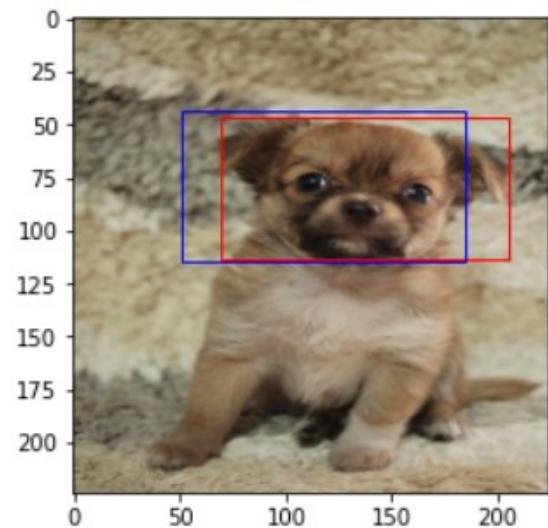
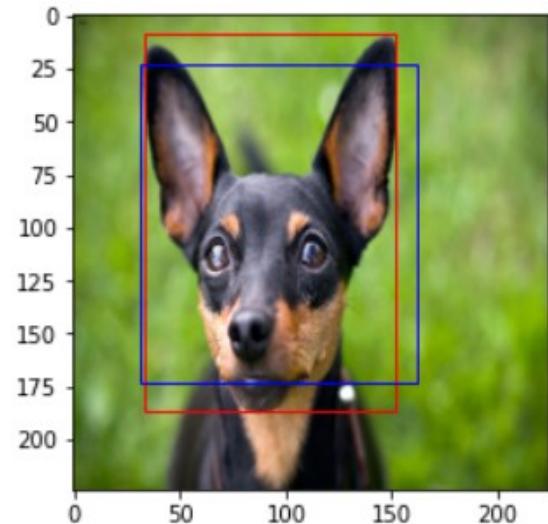
code ►



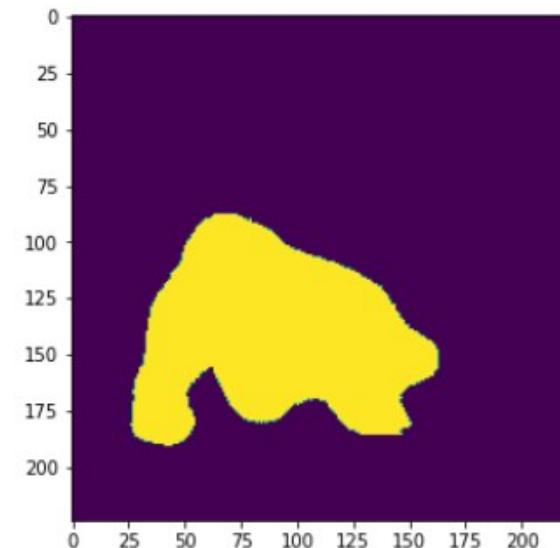
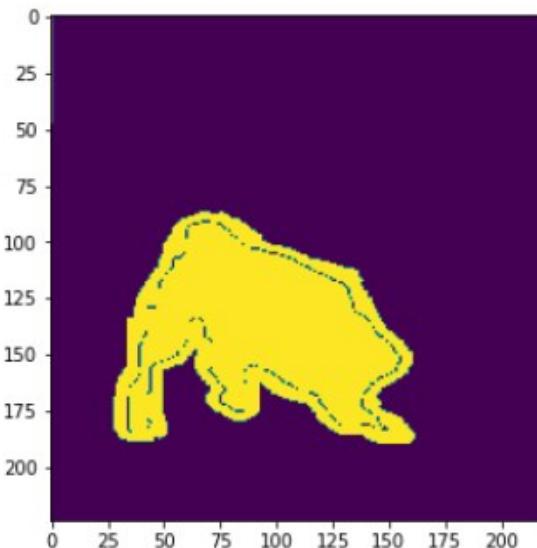
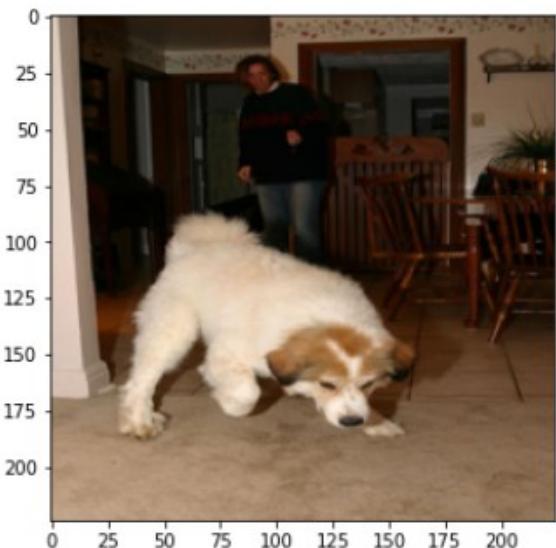
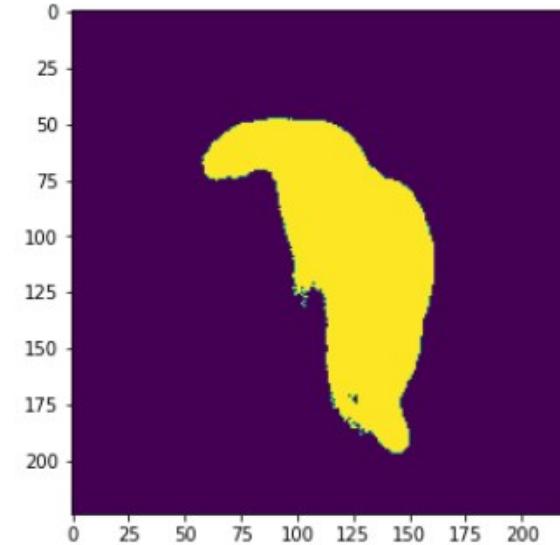
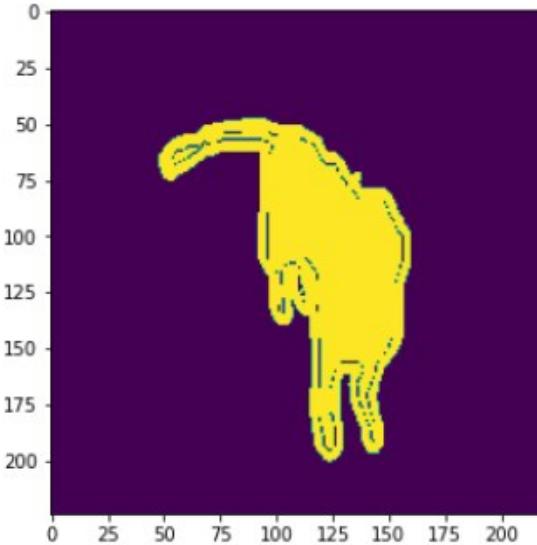
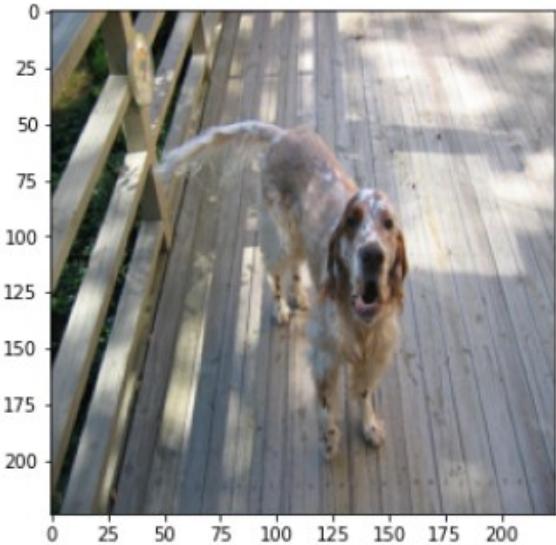
YIQ  
formula

code ►

# Object Detection/Localization



# Segmentation



# Machine Translation

영어 한국어 독일어 언어 감지 ▾

한국어 프랑스어 영어 ▾ 번역하기

옛날에 백조 한 마리가 살았다.

x

Once upon a time a swan lived.

◀ 🔍 ⌂ ▾

17/5000

수정 제안하기

yesnal-e baegjo han maliga sal-assda.

영어 한국어 독일어 언어 감지 ▾

한국어 프랑스어 영어 ▾ 번역하기

옛날에 백조 한 마리가 살았습니다.

x

The 100,000,000,000,001 lived long ago. ✓

◀ 🔍 ⌂ ▾

19/5000

수정 제안하기

yesnal-e baegjo han maliga sal-assseubnida.

# Text to Speech

<https://clovadubbing.naver.com/>

kakao



# Unsupervised Learning

- 아래 음식을 둘로 분류하면?



(1)



(2)



(3)



(4)



(5)



(6)

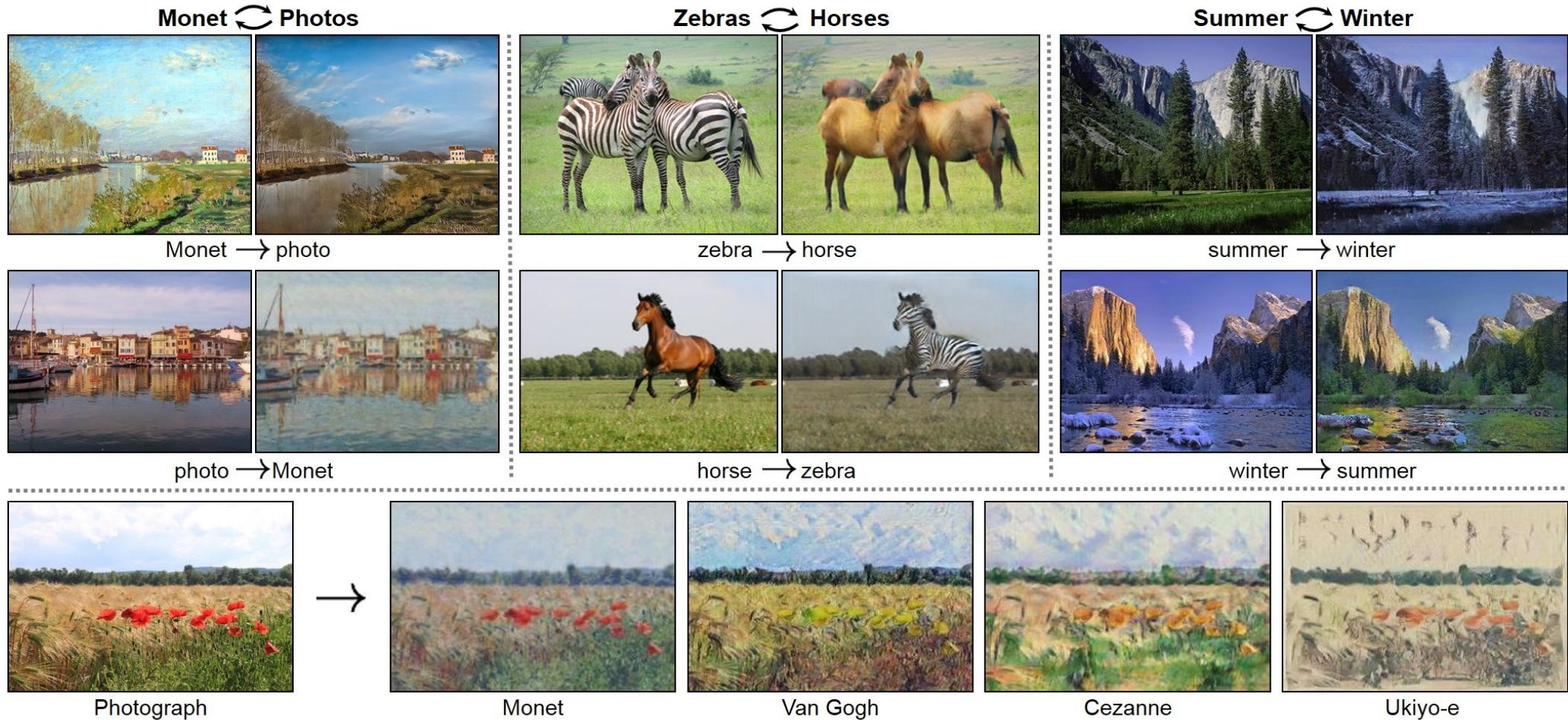


(7)



(8)

# CycleGAN

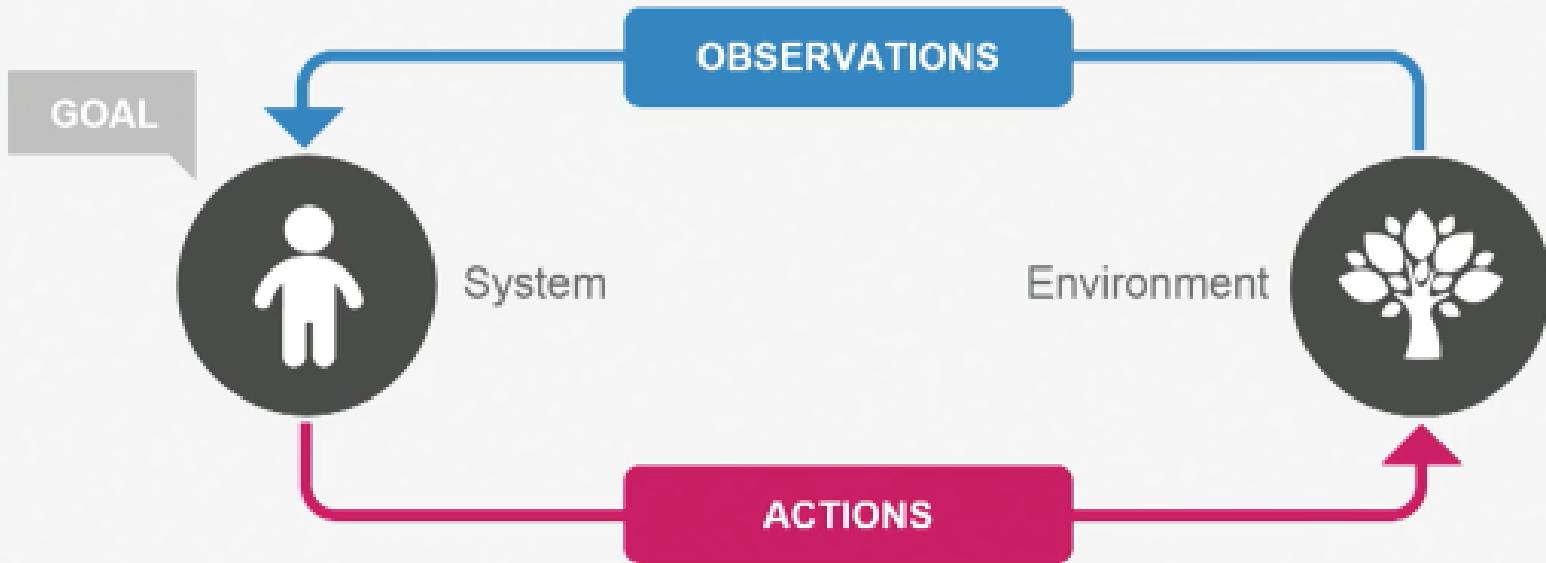


# Video Generation

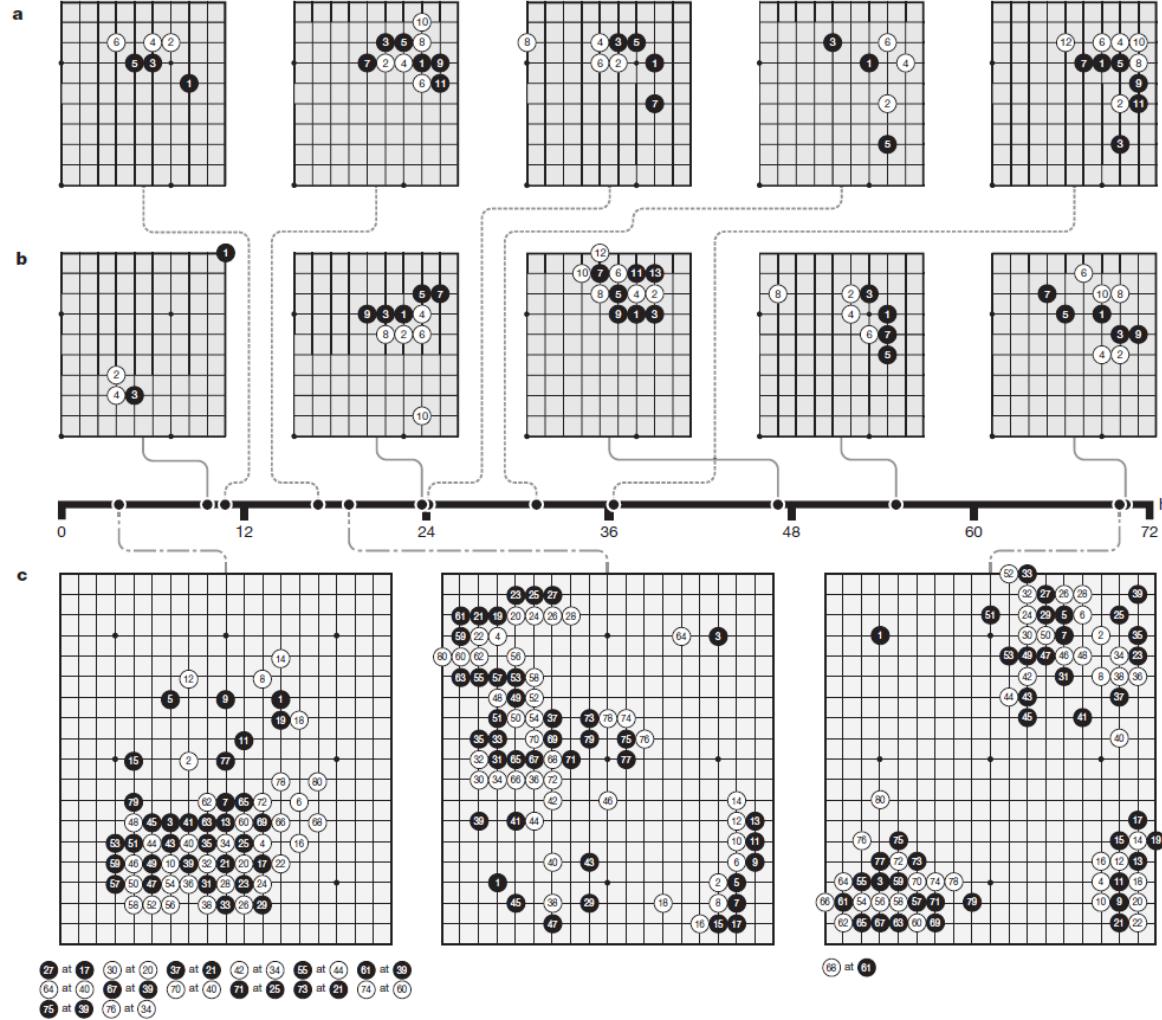


# Reinforcement Learning

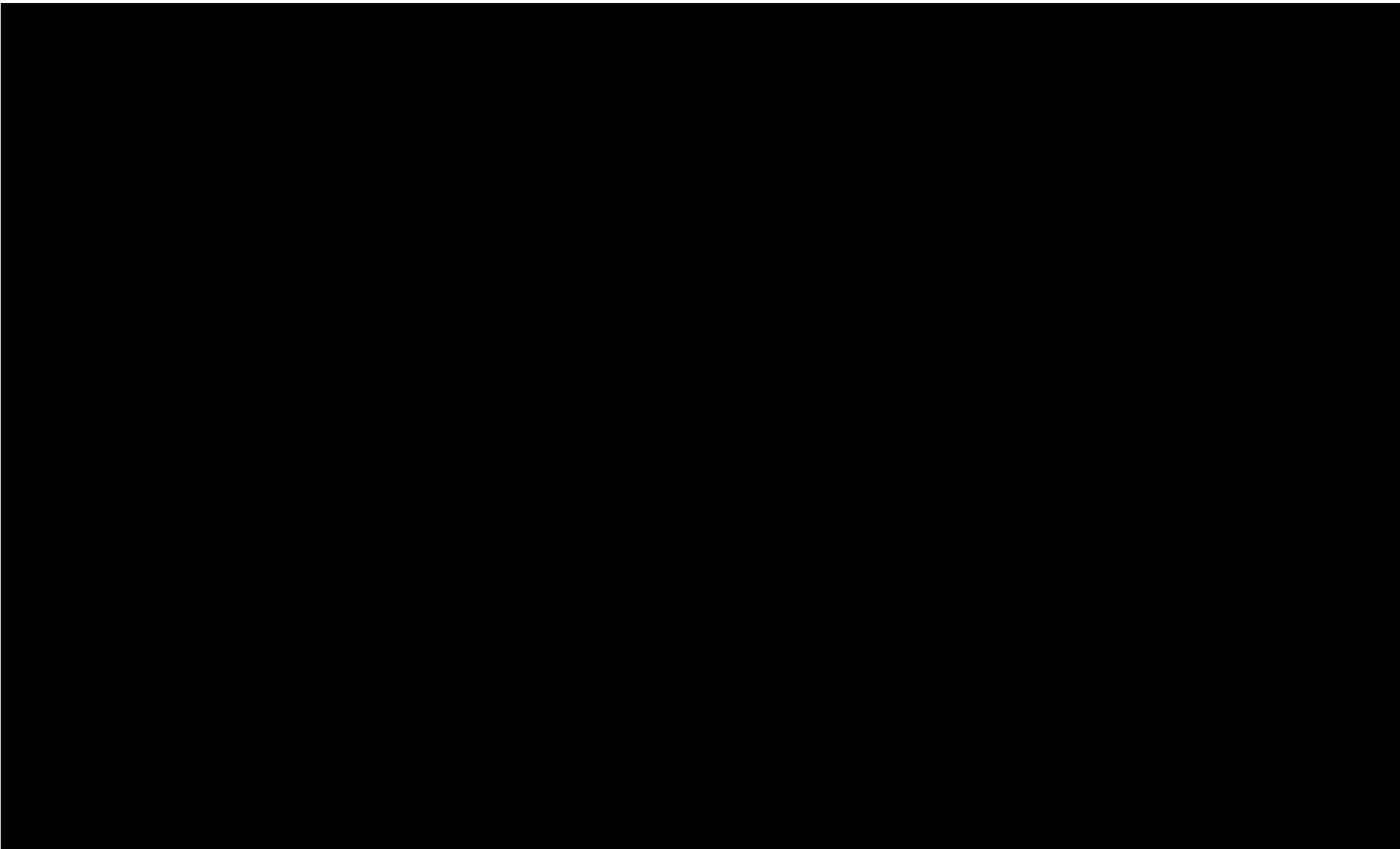
## Reinforcement Learning Framework



# AlphaGo Zero



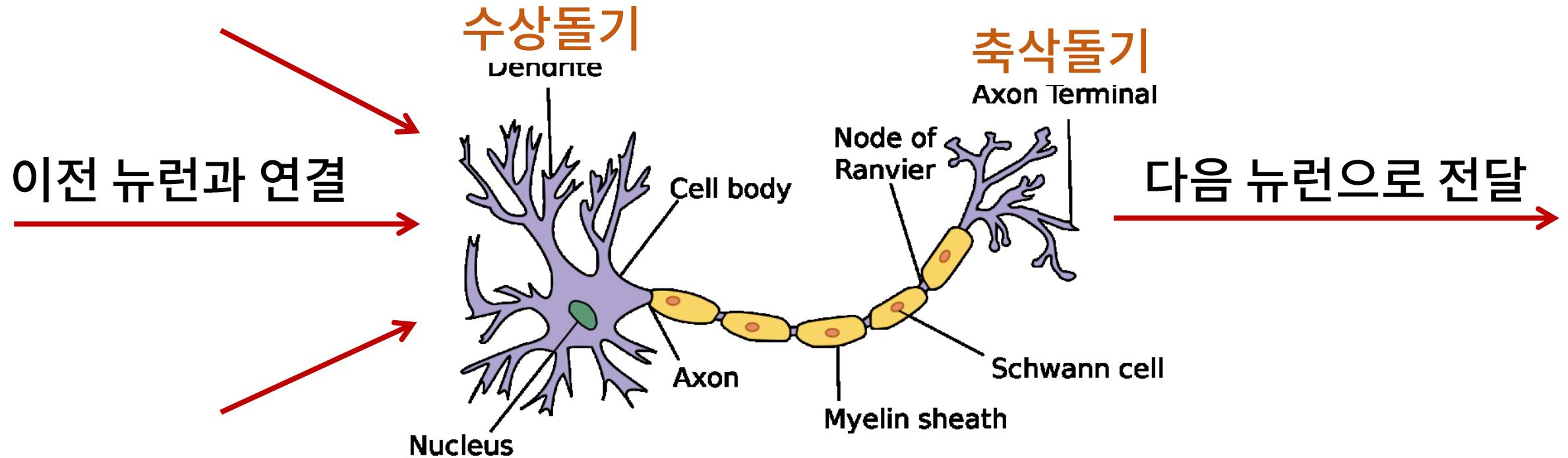
# Atari Games



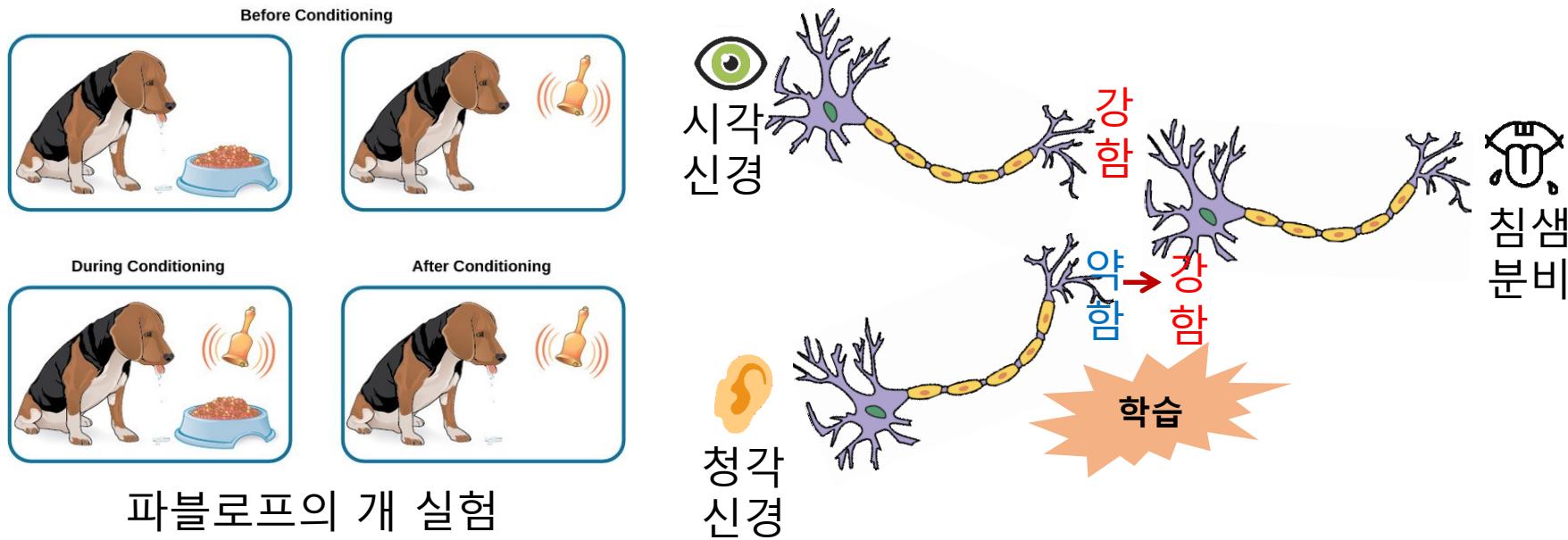
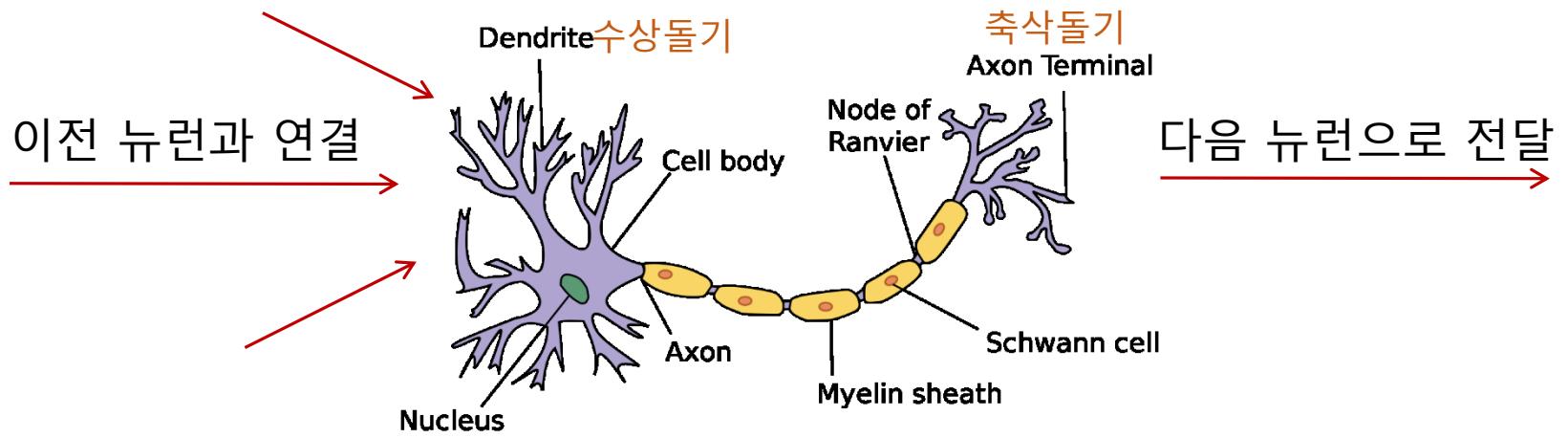
# Quiz

- □와 △에 들어갈 정수는?
  - $3 \times \square + 2 \times \triangle = 1$
  - $1 \times \square + 4 \times \triangle = -3$
  - $5 \times \square + 5 \times \triangle = 0$
  - $8 \times \square + 3 \times \triangle = 5$
- $\square = 1, \triangle = -1$
- $(3, 2), (1, 4), (5, 5), (8, 3)$  은 input data,  $1, -3, 0, 5$  는 label이다
- □와 △를 weight라고 하며 이 weight 값을 기계가 스스로 학습을 통해 찾아내도록 하는 것이 neural network를 이용한 기계학습이 하는 일

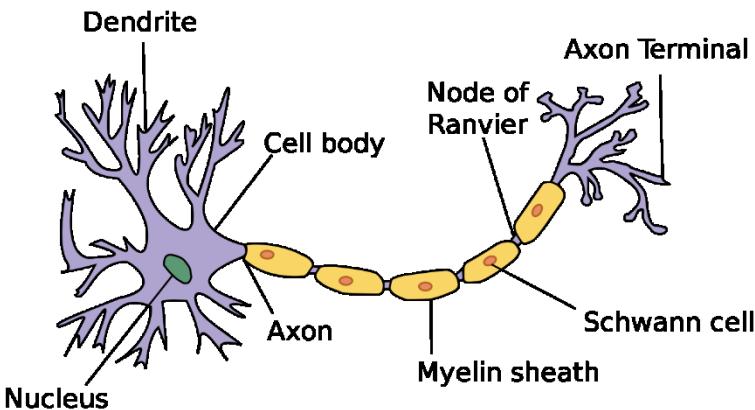
# 뉴런(Neuron)



# 뉴런과 사람의 학습



# Perceptron(Artificial Neural Network)

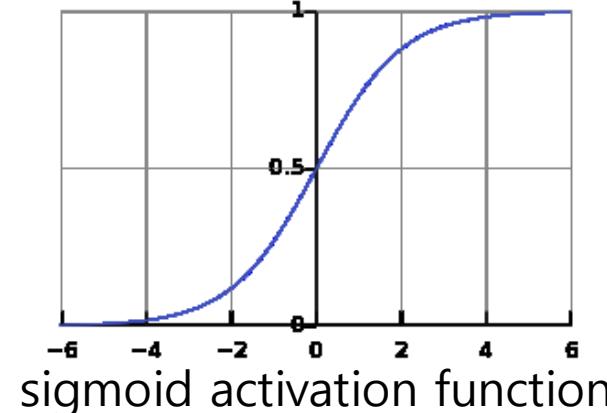


$$y = f(\mathbf{w}\mathbf{x} + b)$$

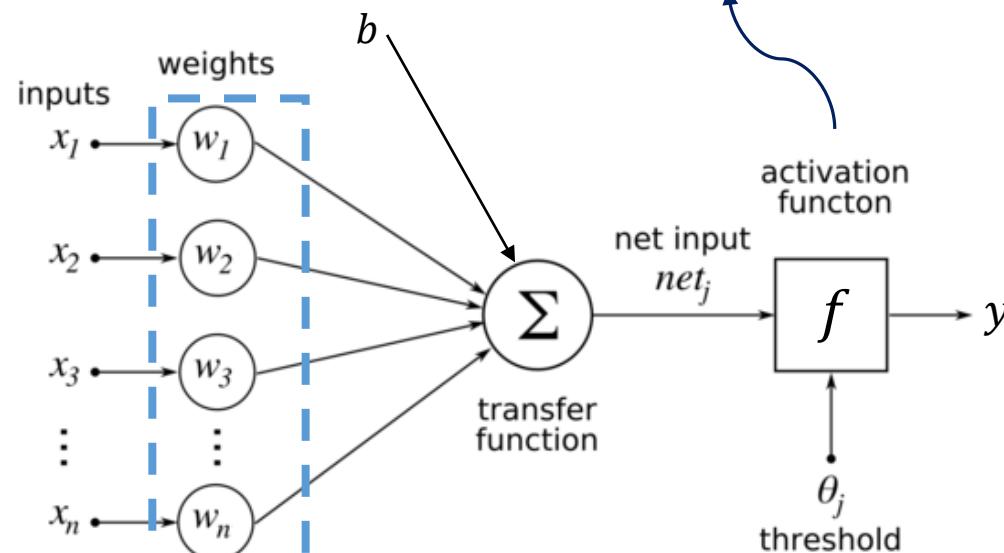
$$\mathbf{w} = [w_1 \ w_2 \ w_3 \ \dots \ w_n]$$

$$\mathbf{x} = [x_1 \ x_2 \ x_3 \ \dots \ x_n]^T$$

연결 강도를 표현



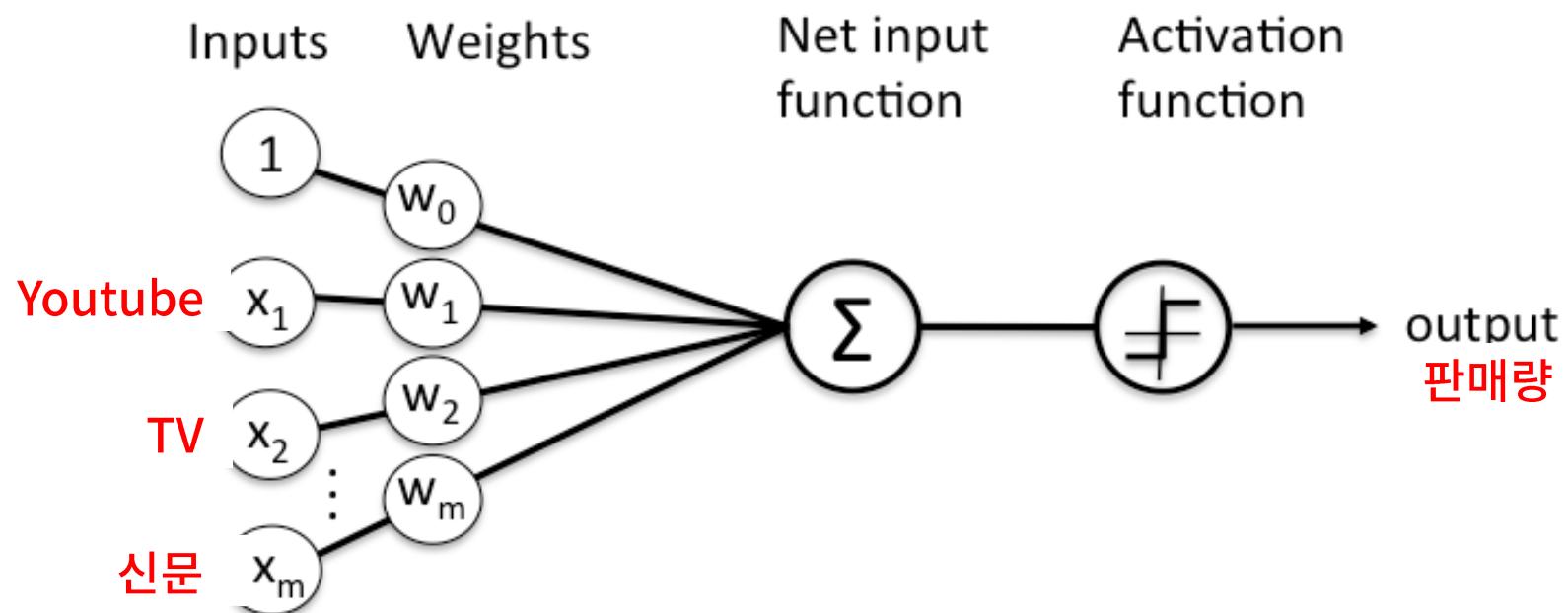
$$f(x) = \frac{1}{1 + e^{-x}}$$



<Perceptron>

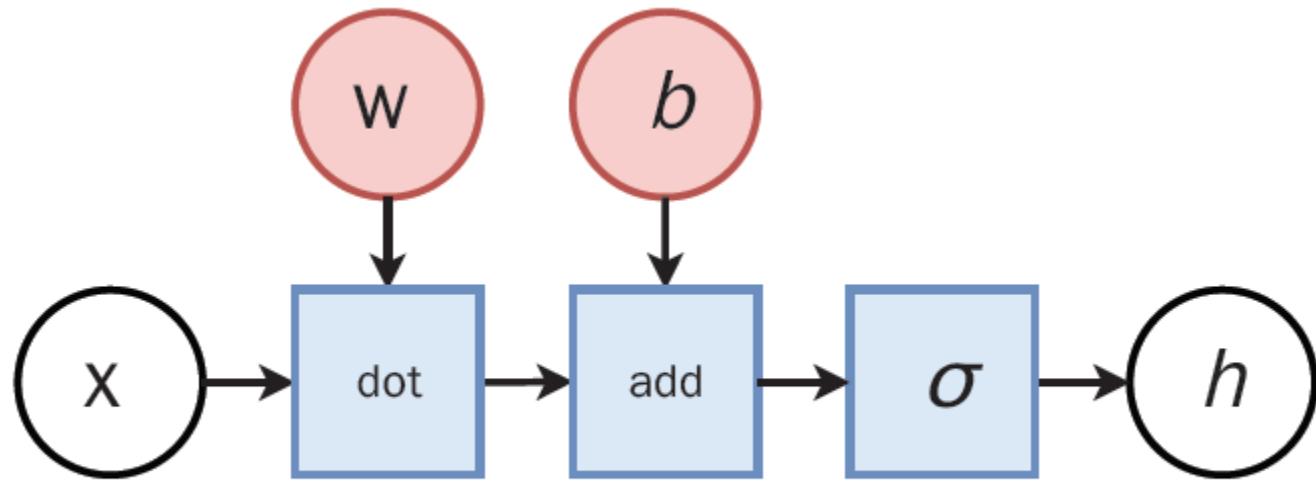
# An Example of Perceptron

어떤 상품을 판매하는 회사에서  
Youtube, TV, 신문 광고료에 따른 판매량을 예측



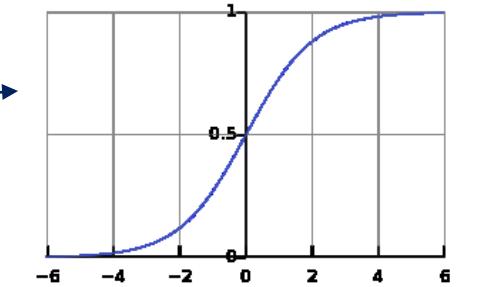
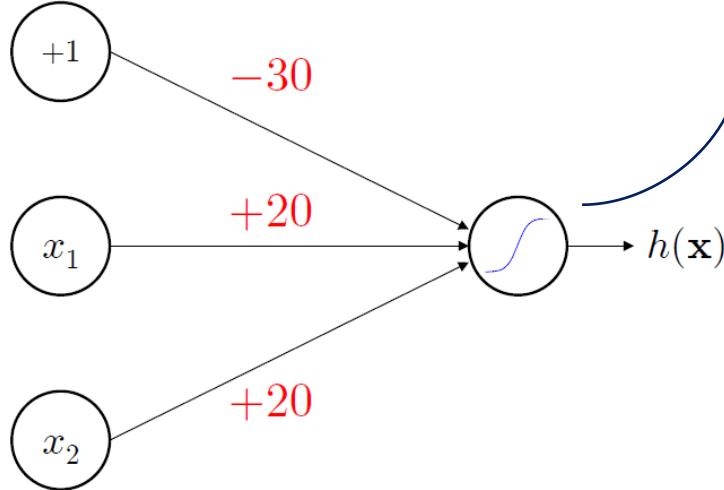
Youtube	TV	신문	판매량
130	70	55	221
44	40	45	109
17	46	70	73
81	40	57	181
52	12	11	90
9	49	61	52
57	33	23	122
30	30	25	100
...	...	...	...

# Computation Graph



This unit is the **lego brick** of all neural networks!

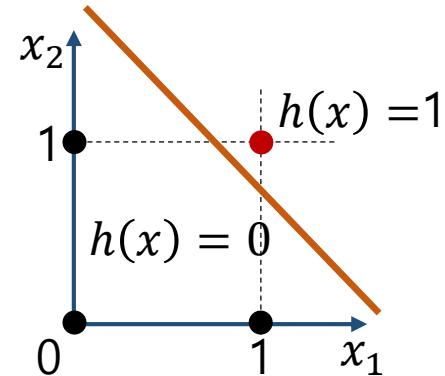
# Example of ANN(logical AND)



$x_1$	$x_2$	$h(\mathbf{x})$
0	0	$\sigma(-30) \approx 0$
0	1	$\sigma(-10) \approx 0$
1	0	$\sigma(-10) \approx 0$
1	1	$\sigma(10) \approx 1$

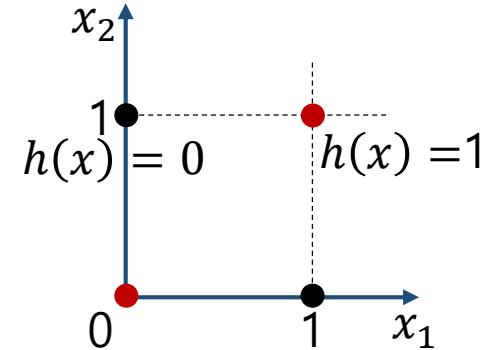
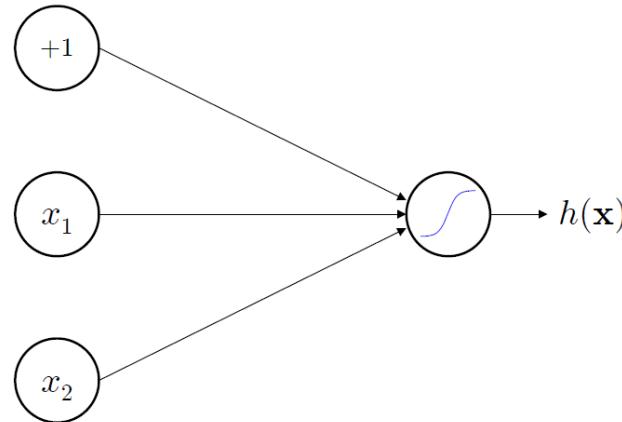
$$h(\mathbf{x}) = \sigma(-30 + 20x_1 + 20x_2)$$

학습이란 이러한 weight 값(-30, 20, 20)을  
기계 스스로 찾을 수 있도록 해주는 과정!

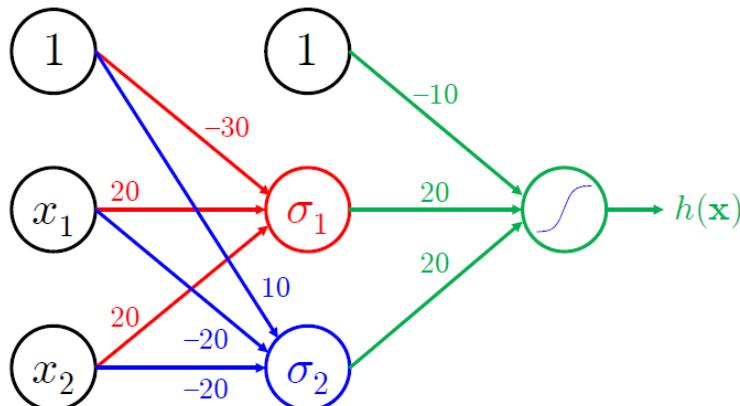


# Logical XNOR with Perceptron

- Is it possible(Linearly separable)?



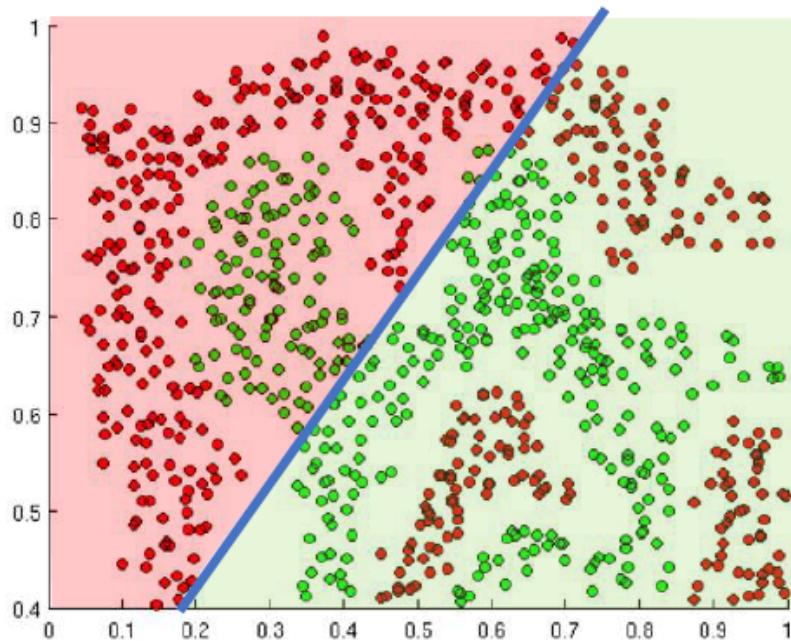
- We need more perceptrons and more layers



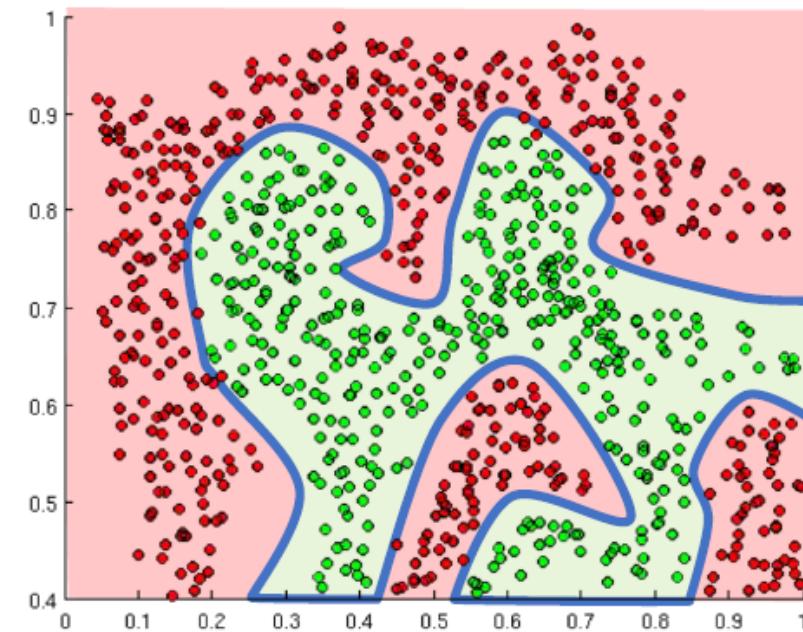
$x_1$	$x_2$	$\sigma_1$	$\sigma_2$	$h(\mathbf{x})$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

# Importance of Activation Functions

The purpose of activation functions is to *introduce non-linearities* into the network



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

# Universal Function Approximation

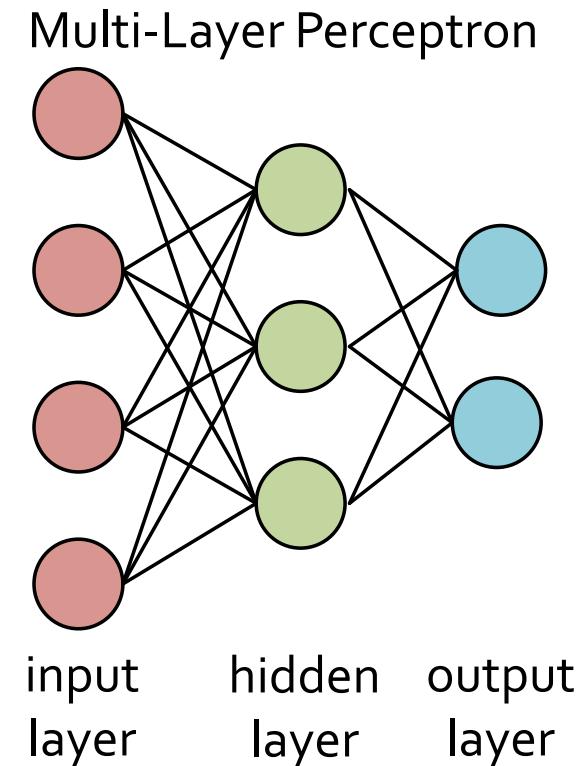
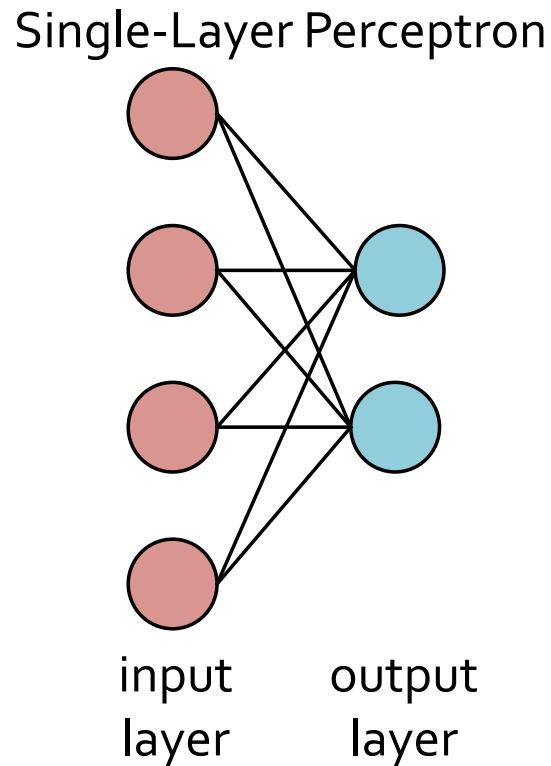
A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.

## Caveats:

The number of hidden units may be infeasibly large

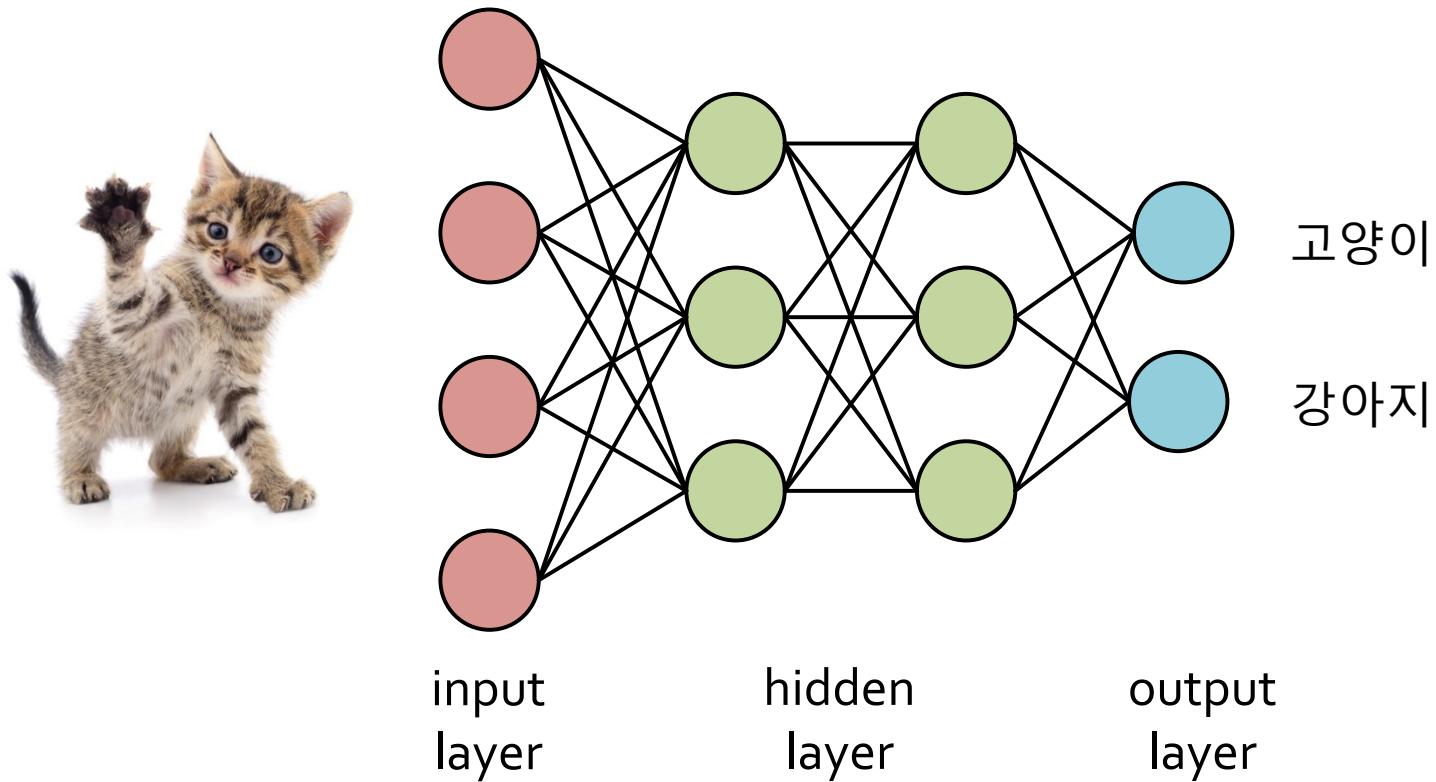
The resulting model may not generalize

# Single-Layer Perceptron & Multi-Layer Perceptron(MLP)

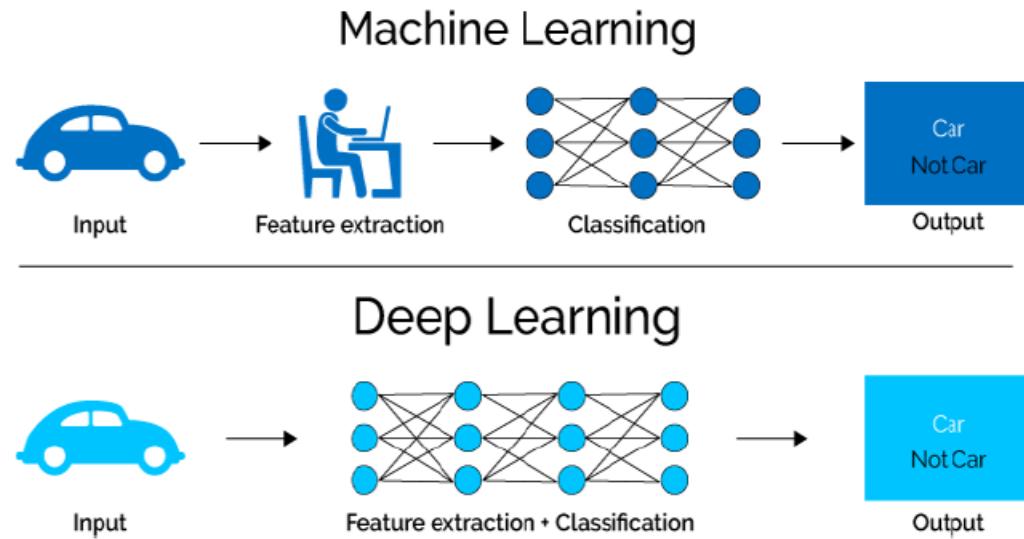


# Deep Learning

- Deep Neural Network을 이용한 Machine Learning 방법
  - Deep Neural Network : Hidden layer 수가 최소 2개 이상인 network



# Machine Learning vs Deep Learning



ML : Audio → Acoustic Model → Phonetic Model → Language Model → Text

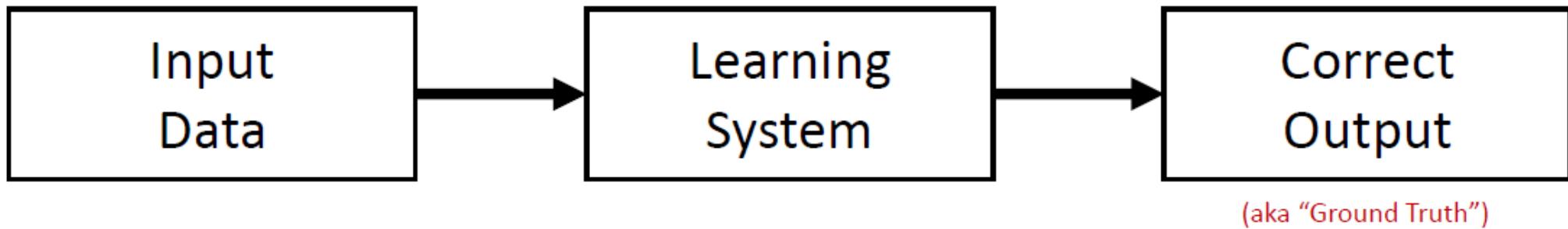
DL : Audio → Deep Networks → Text (End to End Learning)

ML : Pixels → Key Points → SIFT Features → Deformable Part Model → Label

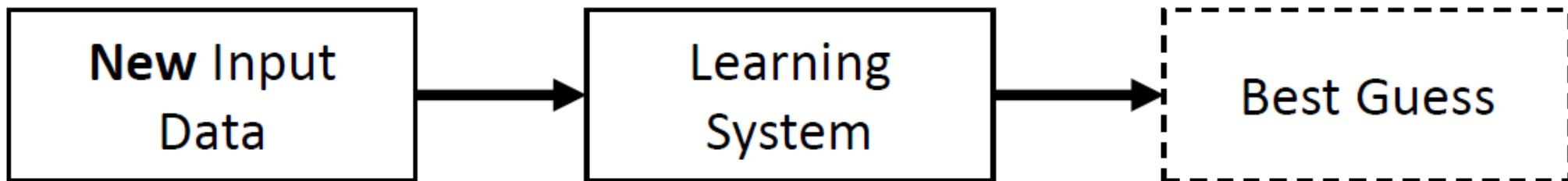
DL : Pixels → Deep Networks → Label (End to End Learning)

# Training and Testing

Training Stage:



Testing Stage:

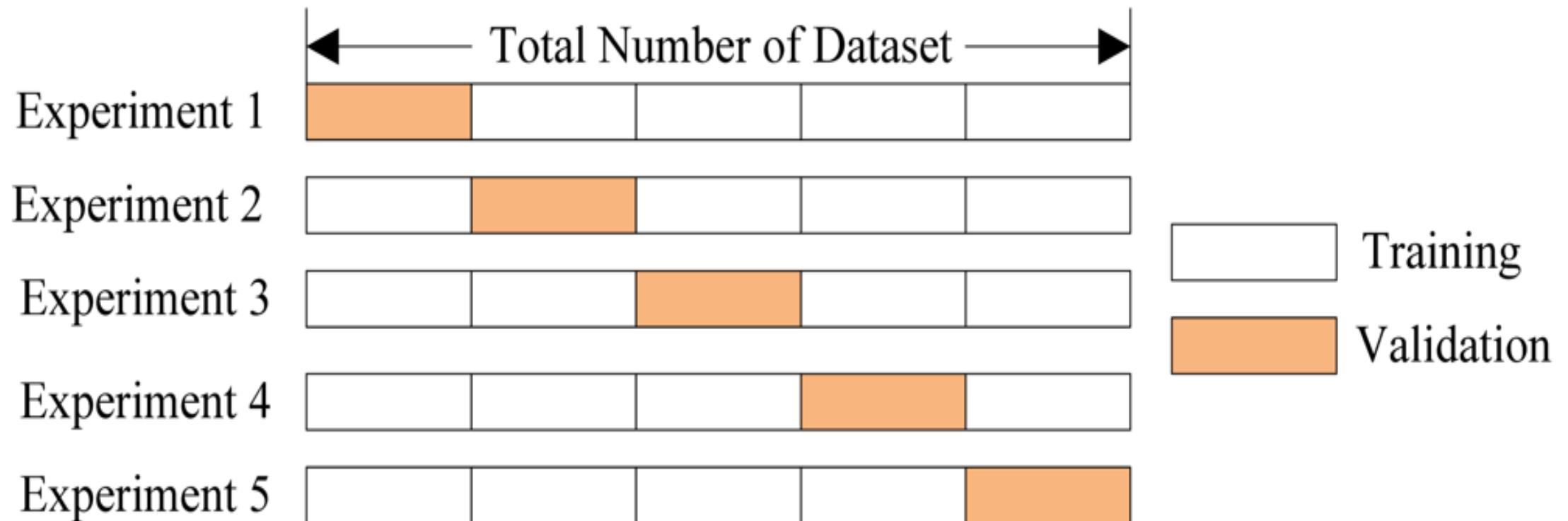


# Dataset

- Training set, Test set
  - or
- Training set, Validation set, Test set
- Training → Tuning(validation) → Test
- In any case, DO NOT USE the test set before the training is over!

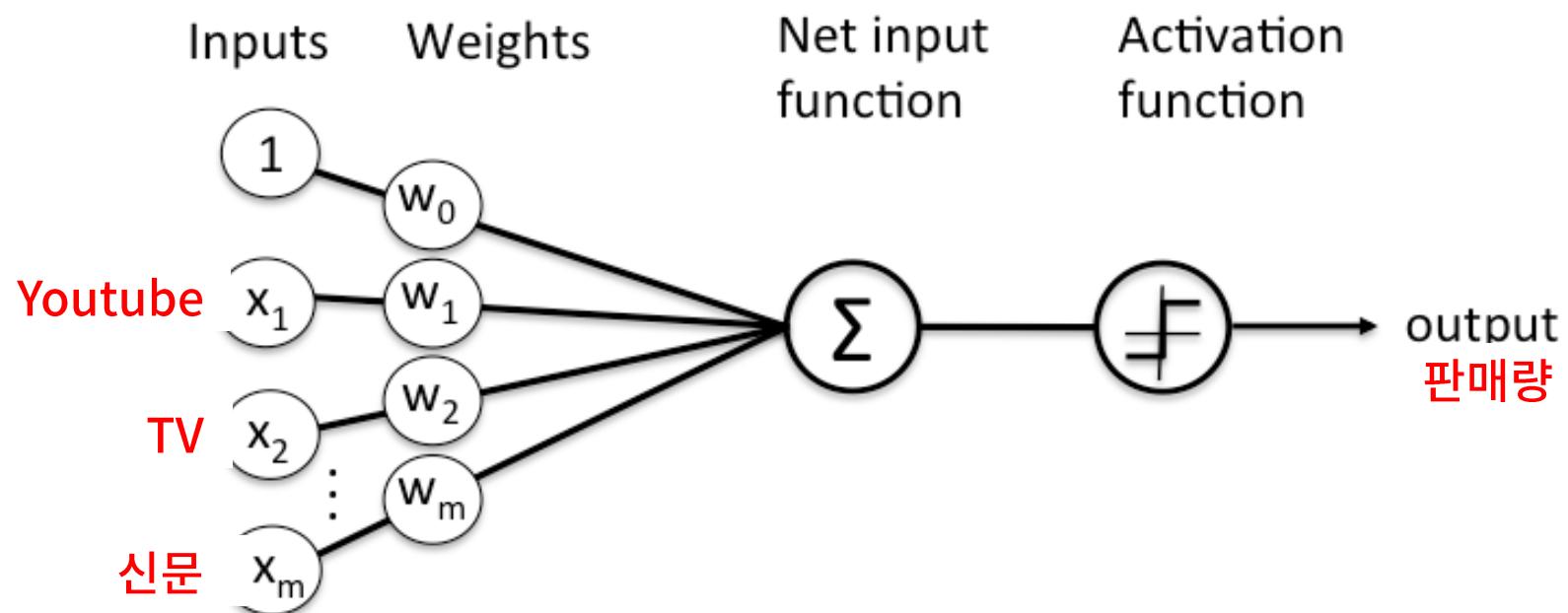


# Cross-Validation



# An Example of Perceptron

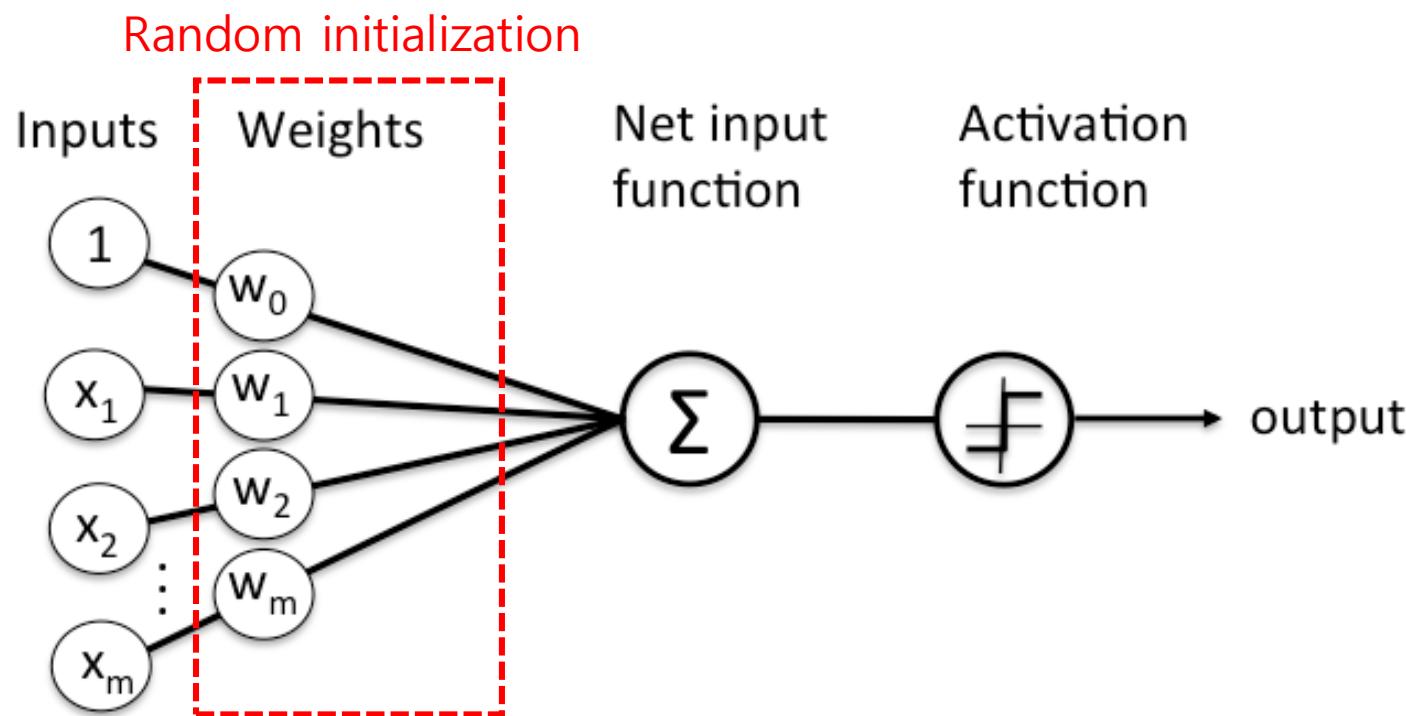
어떤 상품을 판매하는 회사에서  
Youtube, TV, 신문 광고료에 따른 판매량을 예측



Youtube	TV	신문	판매량
130	70	55	221
44	40	45	109
17	46	70	73
81	40	57	181
52	12	11	90
9	49	61	52
57	33	23	122
30	30	25	100
...	...	...	...

# 학습 방법

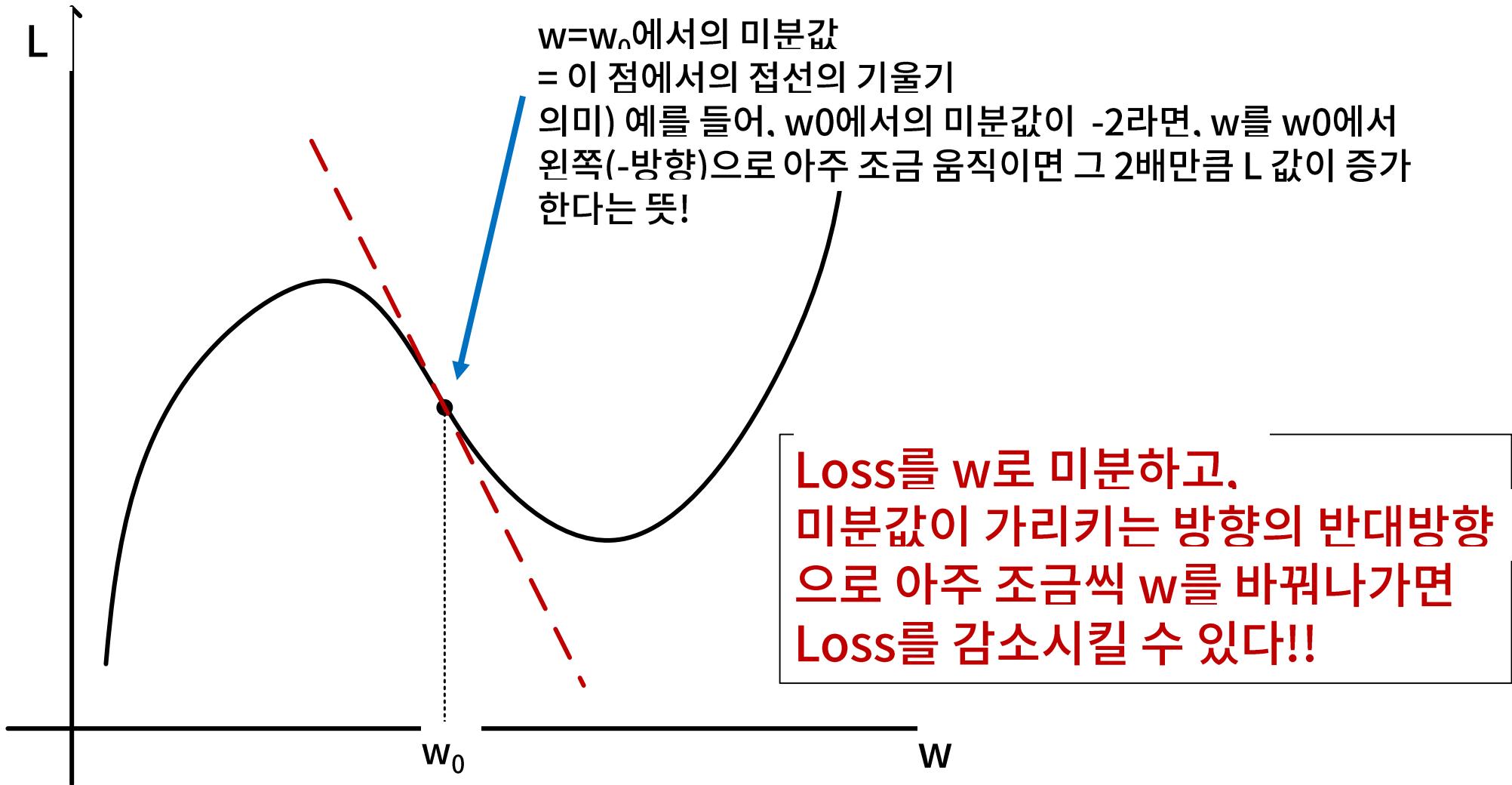
- 최적의 weight를 어떻게 찾을까?
- 아무것도 모르는 상태에서 시작하는데 weight의 초기값은 어떻게?
- 잘 모르겠으니 일단 아무 값이나 넣고 시작하자.



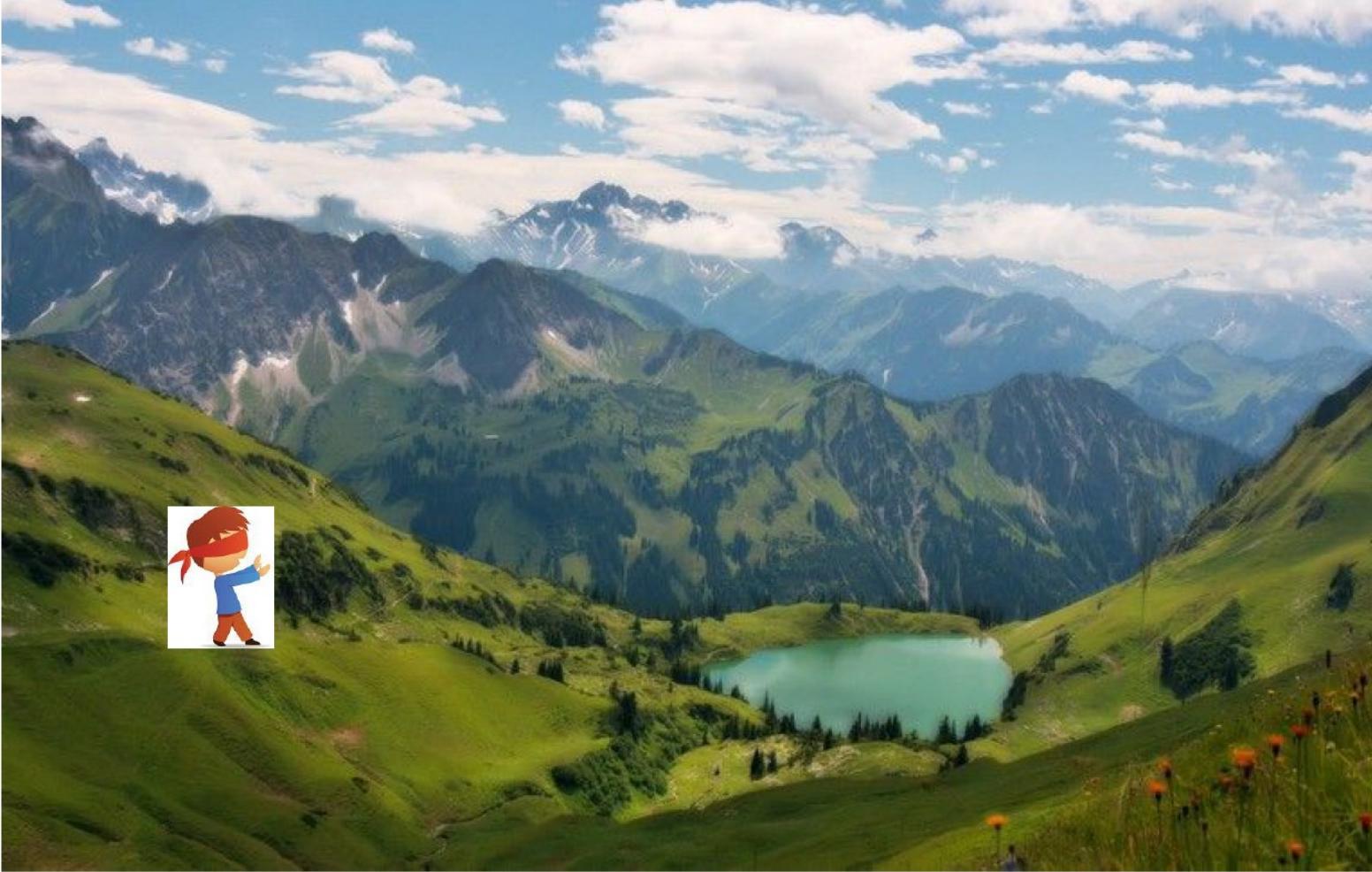
# 학습 방법

- Neural Network이 얼마나 잘하는지 or 얼마나 못하는지에 대한 척도가 필요함
  - Loss Function / Cost Function
- 많이 쓰는 방법 중 한가지
  - (Neural Network의 출력과 실제 정답의 차이)<sup>2</sup>
  - 예) 2가지 광고료 조합에 대하여 Neural Network이 예측한 판매량 : (100, 80), 실제 판매량 : (105, 78)일 때,
$$L = (105 - 100)^2 + (78 - 80)^2 = 29$$
- 이제 필요한 것은 Loss Function의 값이 줄어들도록 weight 값을 조금씩 바꾸는 것
  - Weight를 어떻게 바꿔야 Loss Function 값이 줄어들까? → 미분!

# 미분??



# Gradient Descent



# Gradient Descent

Loss Function의 미분(Gradient)를 이용하여 weight를 update하는 방법

$$w_{new} = w_{old} - \eta \nabla_w L$$

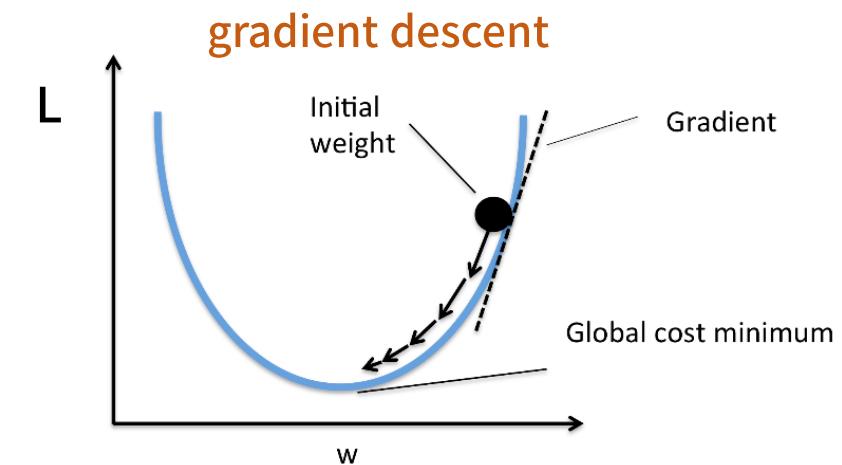
Weight update =  $w_{new} - w_{old}$

$$= -\eta \nabla_w L$$

Loss를 감소 시키는 방향  
(Descent)

아주 조금씩 이동  
(Learning Rate)

미분값  
(Gradient)



# Gradient Descent

- 내가 가진 모든 training data에 대하여 Neural Network의 출력과 실제 정답(label)을 비교하여 각각의 Loss를 계산하고 이를 모두 더해서 전체 Loss를 계산한다
- 그리고, 이 loss를 weight로 미분한 다음 그 미분값(gradient)가 가리키는 방향의 반대방향으로 weight 값을 아주 조금씩 바꿔나간다
- 그런데 내가 가진 data가 아주아주 많은데 weight 값을 아주 조금 바꾸기 위해서 모든 data에 대해서 loss를 다 계산해야 한다...
- Data가 더 많아질수록 학습이 더 느려진다...

# Batch / Stochastic / Mini-batch GD

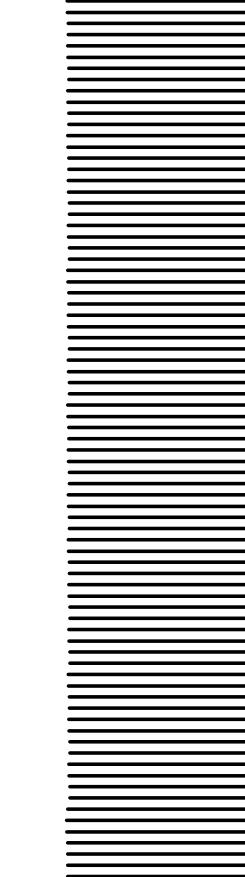
모든 training data에 대하여 loss를 다 계산하는 건 너무 오래 걸리니까 전체 data 중에 일부 data만 뽑고(sampling) 이 data들이 전체 training data를 잘 대표한다고 가정해서 이 data들로만 loss를 계산하여 weight를 update해도 되지 않을까?

- Batch Gradient Descent
  - 모든 data에 대해서 loss를 계산하여 다 더하고 이를 이용해서 GD
- Stochastic Gradient Descent
  - Data를 1개만 뽑고, 그 1개의 data에 대한 loss를 이용하여 GD
- Mini-batch Gradient Descent
  - Batch/Stochastic의 중간 형태로 data를 n개를 뽑고 그 n개의 data에 대한 loss를 계산하여 다 더한 뒤 이를 이용하여 GD → 제일 많이 쓰는 방법!

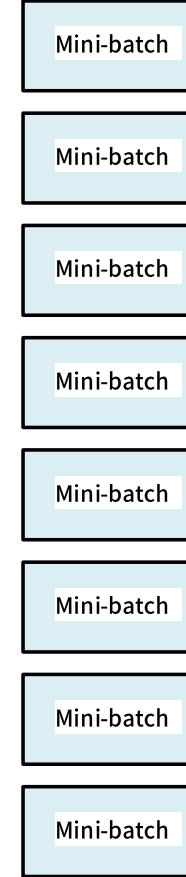
# Batch / Stochastic / Mini-batch GD



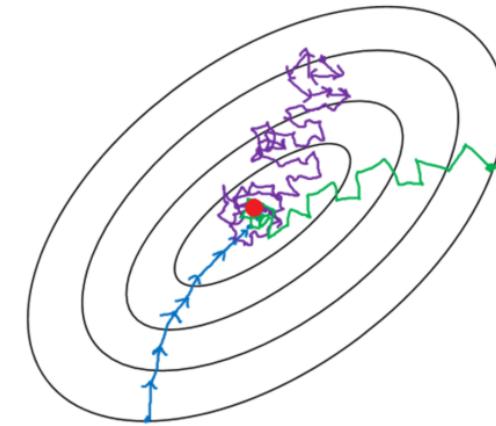
Batch  
Gradient Descent



Stochastic  
Gradient Descent



Mini-batch  
Gradient Descent



- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent

# 미분을 계산해봅시다!

$$z_{11} = x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13} + x_4 \cdot w_{14}$$

$$a_{11} = \sigma(z_{11}) = \frac{1}{1 + e^{-z_{11}}}$$

$$z_2 = a_{11} \cdot w_{21} + a_{12} \cdot w_{22} + a_{13} \cdot w_{23}$$

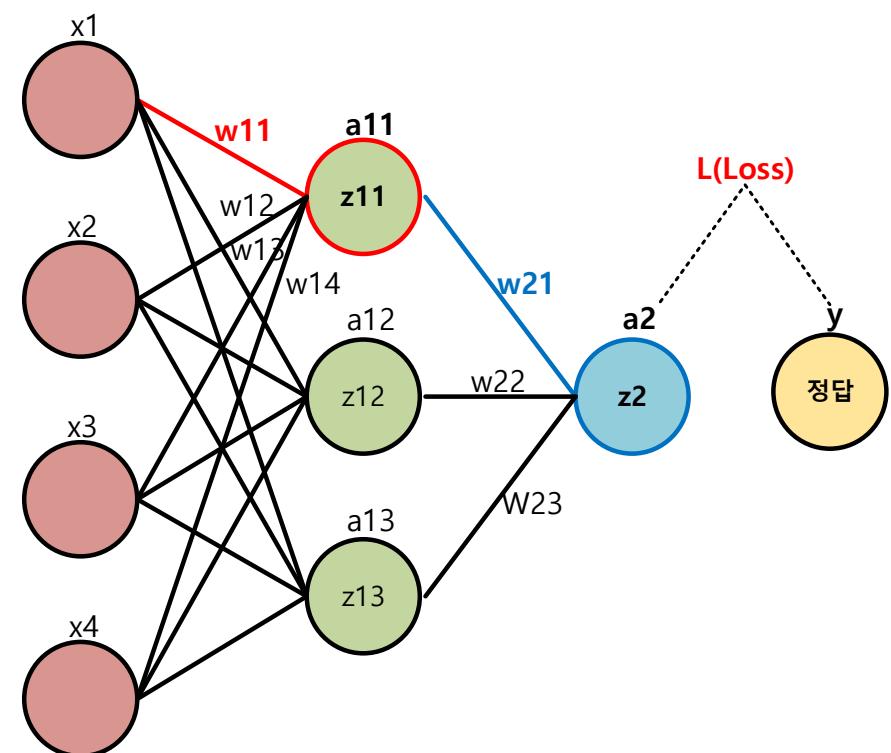
$$a_2 = z_2$$

$$L = (y - a_2)^2$$

일 때,

w<sub>11</sub>을 update 하기 위해 필요한 미분값

$$\frac{\partial L}{\partial w_{11}} = ??????$$



# Back Propagation

$$z_{11} = x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13} + x_4 \cdot w_{14}$$
$$a_{11} = \sigma(z_{11}) = \frac{1}{1 + e^{-z_{11}}}$$
$$z_2 = a_{11} \cdot w_{21} + a_{12} \cdot w_{22} + a_{13} \cdot w_{23}$$
$$a_2 = z_2$$
$$L = (y - a_2)^2$$

Loss부터 거꾸로 한 단계씩 미분을 해봅시다

$$\partial L / \partial a_2 = -2(y - a_2)$$

$$\partial a_2 / \partial z_2 = 1$$

$$\partial z_2 / \partial a_{11} = w_{21}$$

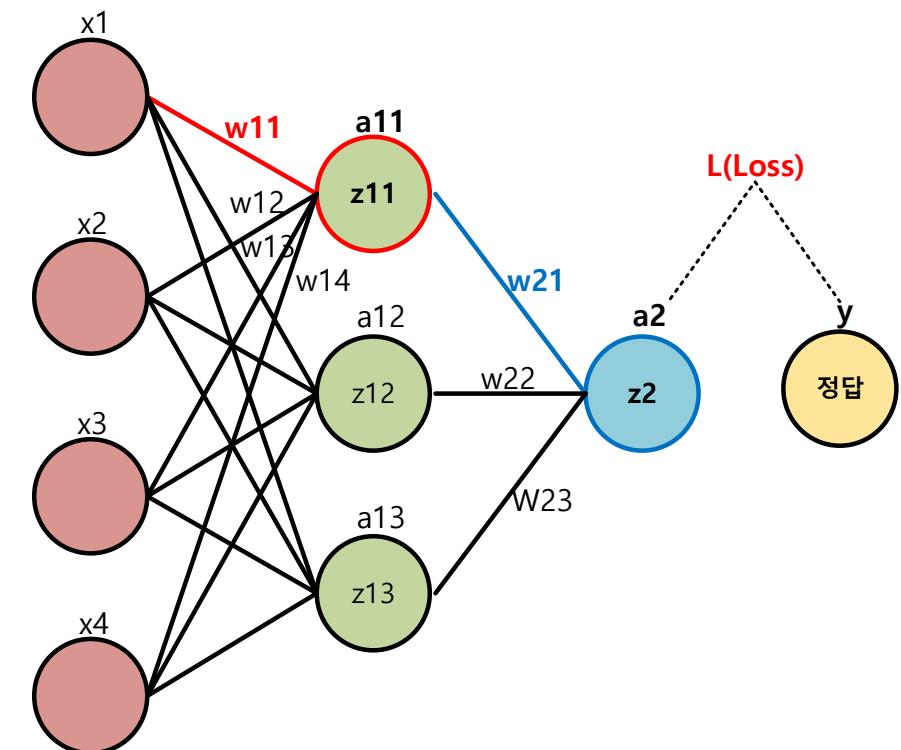
$$\partial a_{11} / \partial z_{11} = \sigma(z_{11}) \cdot (1 - \sigma(z_{11}))$$

$$\partial z_{11} / \partial w_{11} = x_1$$

이 미분들을 전부 각각 곱하면(chain rule),

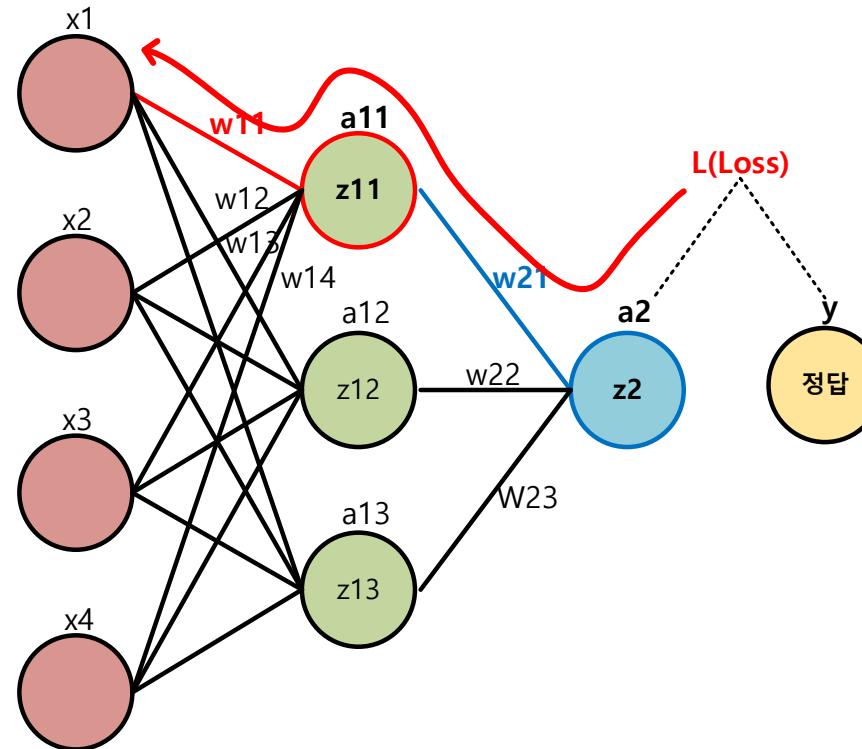
$$\frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_{11}} \cdot \frac{\partial a_{11}}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial w_{11}} = \frac{\partial L}{\partial w_{11}}$$

우리가 구하려고 했던 미분값



# Back Propagation

Loss로부터 거꾸로 한 단계씩 미분 값을 구하고 이 값들을 chain rule에 의하여 곱해가면서 weight에 대한 gradient를 구하는 방법

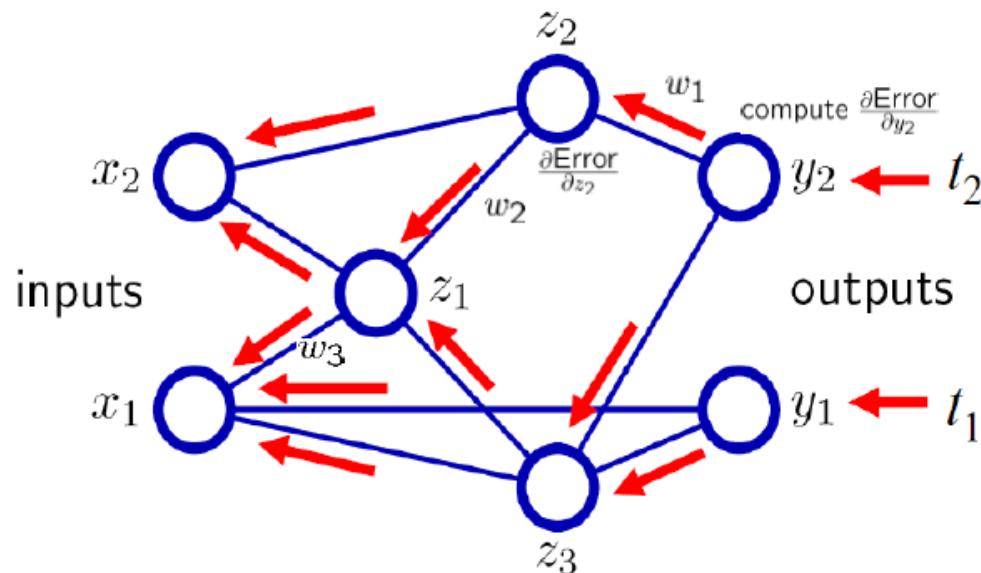
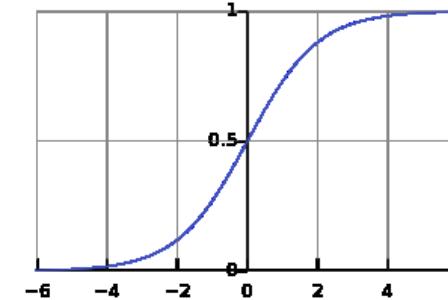


# 딥러닝을 어렵게 하는 것들

- Vanishing gradient problem
- Overfitting problem
- Get stuck in local minima

# Vanishing Gradient Problem

- Gradient 값이 뒤로 전달될 수록 점점 작아짐
- Sigmoid 사용으로 인하여(미분값의 최대 :  $\frac{1}{4}$ ) 아래쪽 layer는 학습이 이루어지지 않음



$$\frac{\partial \text{Error}}{\partial w_1} = \frac{\partial \text{Error}}{\partial y_2} \frac{\partial y_2}{\partial w_1} = \frac{\partial \text{Error}}{\partial y_2} \sigma'(\cdot) z_2$$

$$\frac{\partial y_2}{\partial w_1} = \sigma'(\cdot) z_2$$

$$\frac{\partial \text{Error}}{\partial w_2} = \frac{\partial \text{Error}}{\partial z_2} \frac{\partial z_2}{\partial w_2} = \frac{\partial \text{Error}}{\partial y_2} \sigma'(\cdot) w_1 \sigma'(\cdot) z_1$$

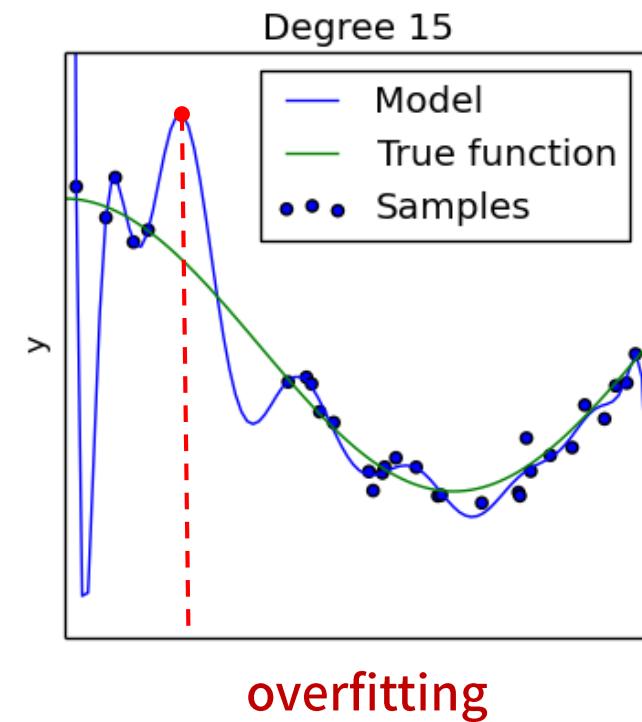
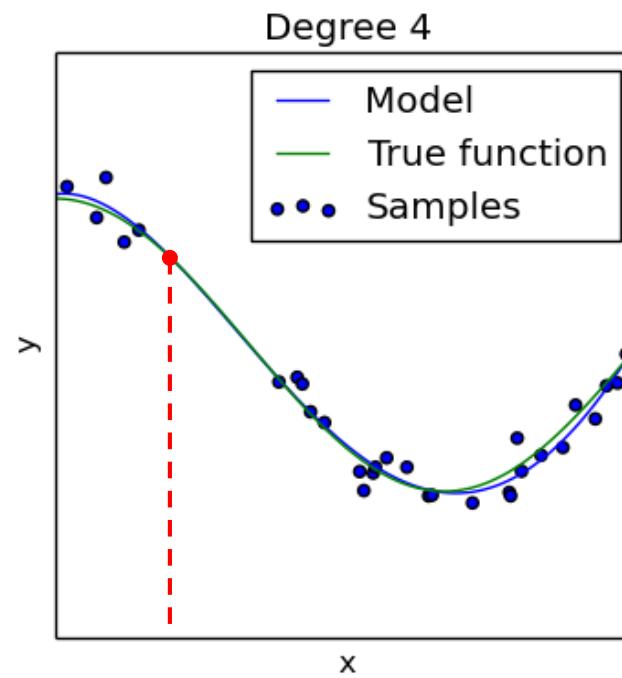
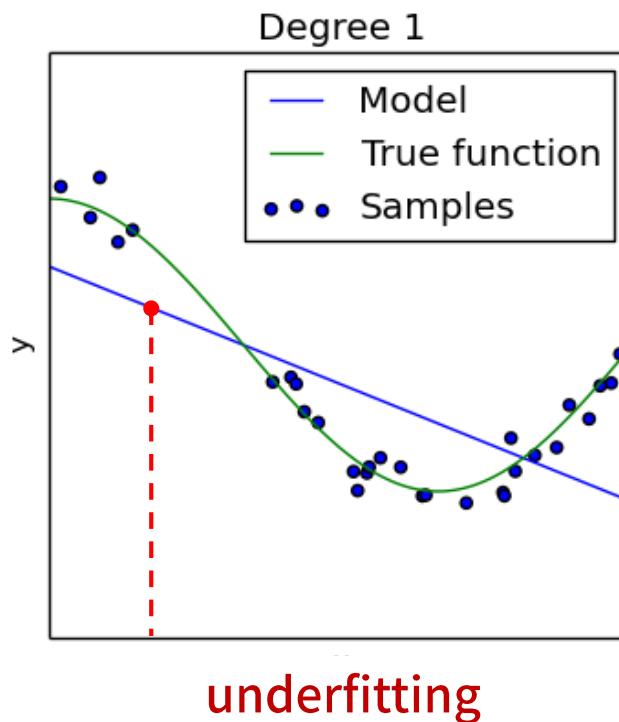
$$\frac{\partial z_2}{\partial w_2} = \sigma'(\cdot) z_1$$

$$\frac{\partial \text{Error}}{\partial w_3} = \sigma'(\cdot) \sigma'(\cdot) \sigma'(\cdot) C$$

$$\sigma'(\cdot) \approx 0$$

# Overfitting Problem

- Data가 많지 않은 경우에 발생할 수 있음
- Overfitting : Training data에 너무 최적화(fitting)를 함으로 인해서 일반화(generalization) 성능이 떨어지는 현상



# Overfitting Problem

열심히 Neural Network에게 고양이



를 가르쳤더니...



뚱뚱하니까 고양이 아님



갈색이니까 고양이 아님



귀 쳐졌으니까 고양이 아님

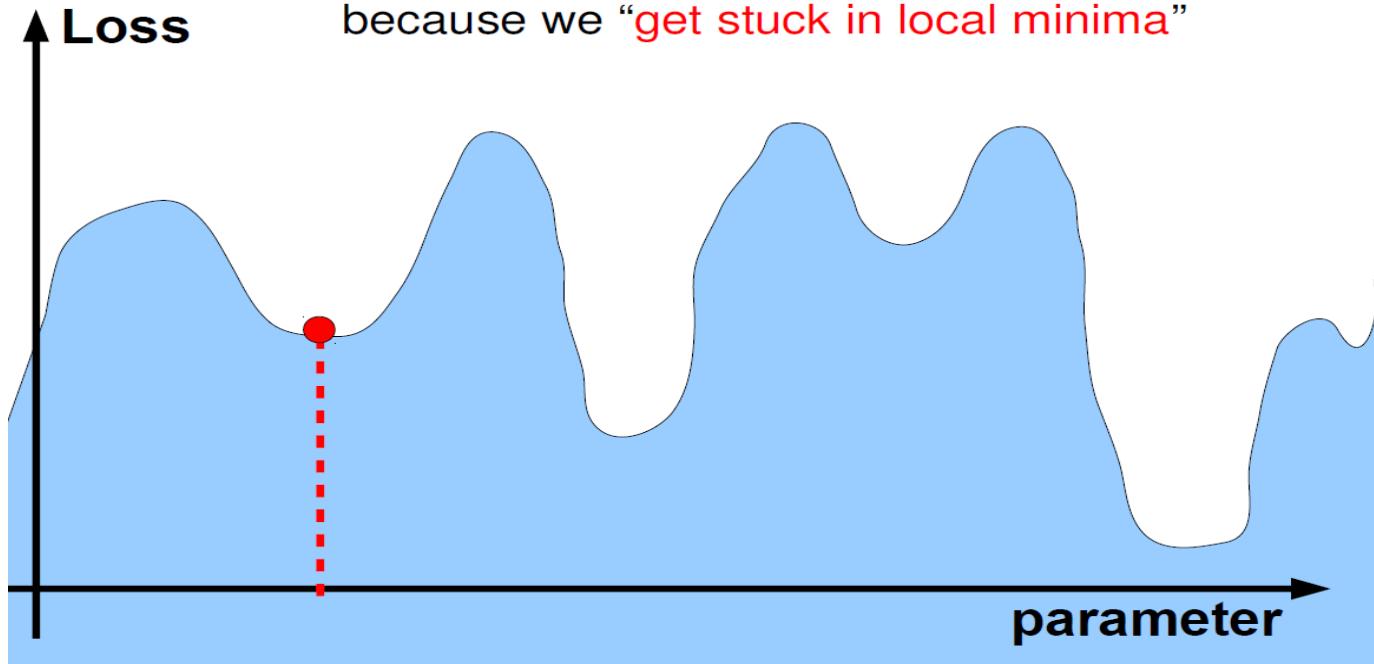
융통성이 하나도 없어짐 → Overfitting!!

# Local Minima

- 어디서 시작하느냐에 따라서 잘못하면 local minima에 빠질 위험이 존재

## ConvNets: till 2012

Common wisdom: training does not work because we “get stuck in local minima”

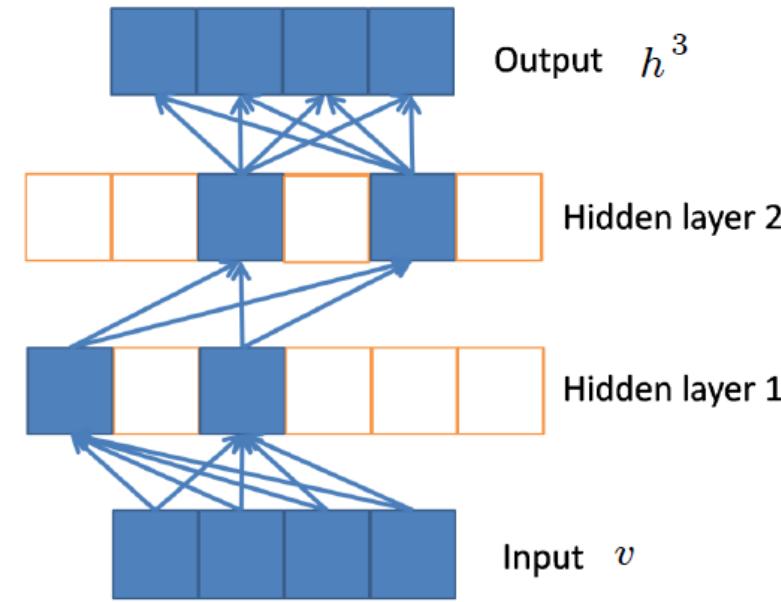
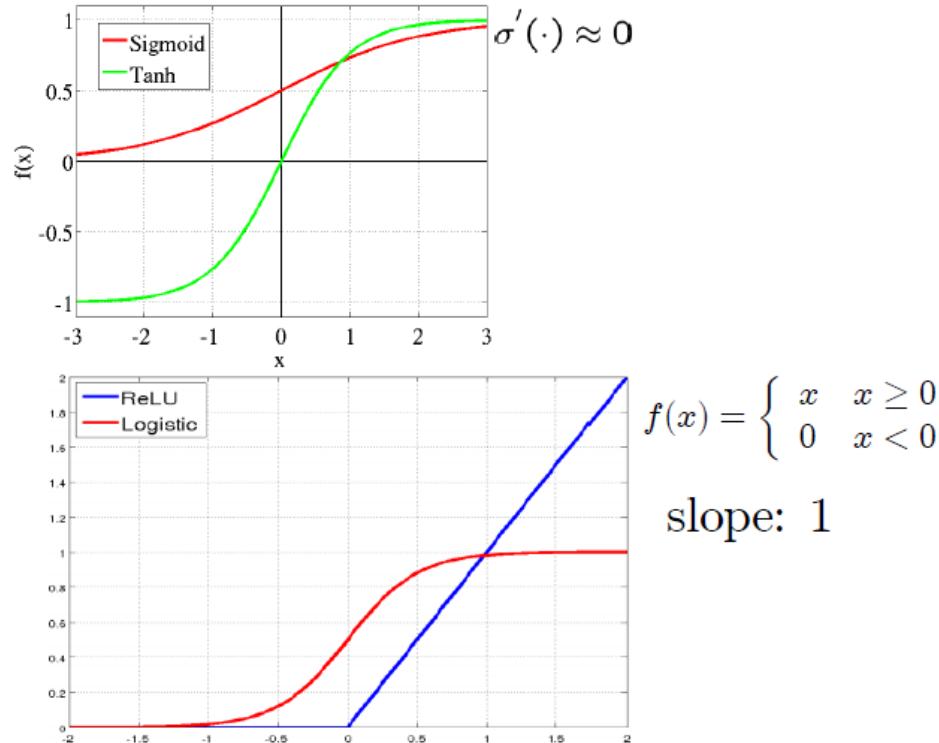


# Solutions

- Vanishing gradient problem  
→ Sigmoid 말고 ReLU를 쓰자
- Overfitting problem  
→ Regularization method를 쓰자(예 : dropout)
- Get stuck in local minima  
→ Local minima에 빠져도 괜찮다

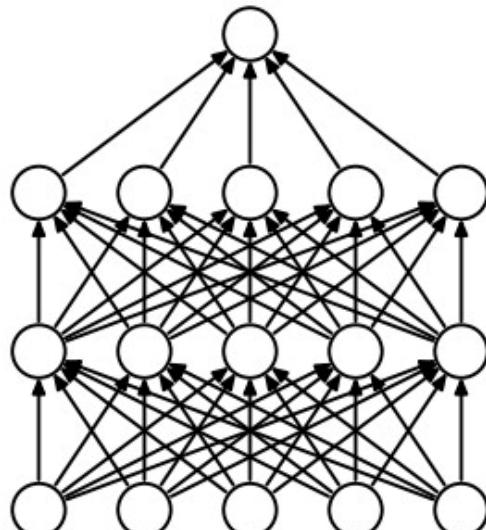
# ReLU : Rectified Linear Unit

- ReLU를 activation function으로 사용 → sparse activation
- ReLU는 미분값이 0 아니면 1 → vanishing gradient 해결

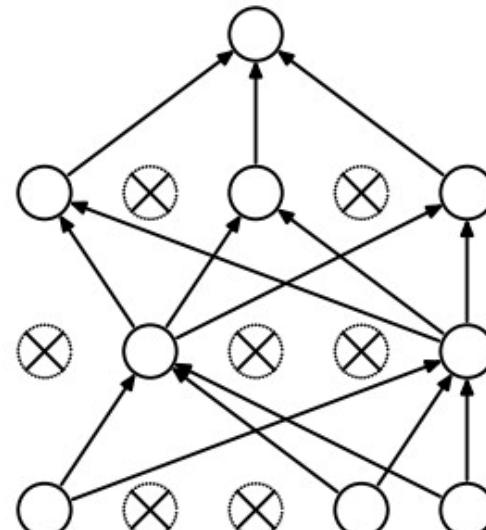


# Dropout(Regularization Method)

- 각 학습 단계마다, 특정 확률로(예 : 50%) random하게 hidden layer에 있는 unit들을 없애고 학습하는 방법
- Ensemble 개념을 적용
  - 여러 개의 model을 사용하여 평균값을 쓰면 하나의 model을 쓰는 경우보다 좋음
  - 하나의 model로 비슷한 효과를 낼 수 있는 방법



(a) Standard Neural Net



(b) After applying dropout.

# Other Regularization Methods

- Weight Decay(L2 Regularization)

$$E(w) = E_0(w) + \frac{1}{2} \lambda \sum_i w_i^2$$

- Batch Normalization

- Benefits of BN

- Increase learning rate
- Remove dropout
- Reduce L2 weight decay
- Remove LRN

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
Parameters to be learned:  $\gamma, \beta$

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

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

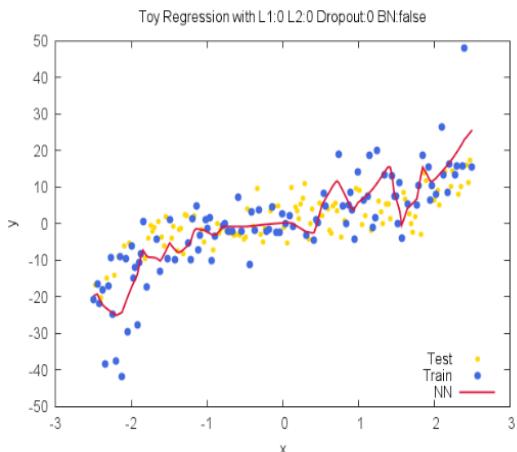
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

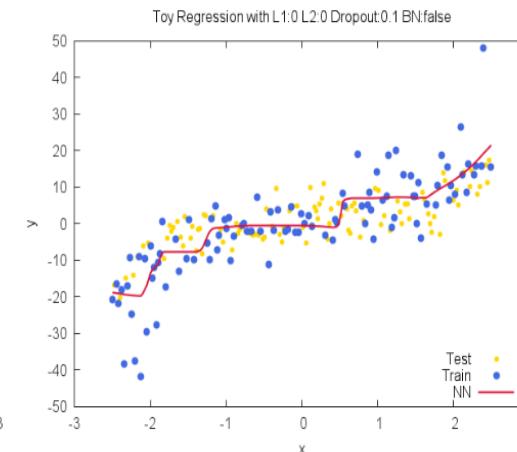
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Regularization Methods

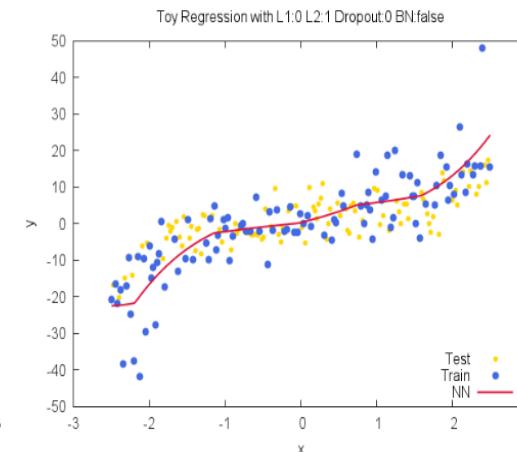
No Regularization



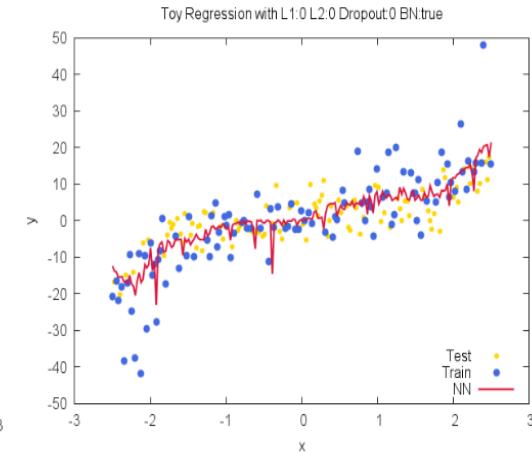
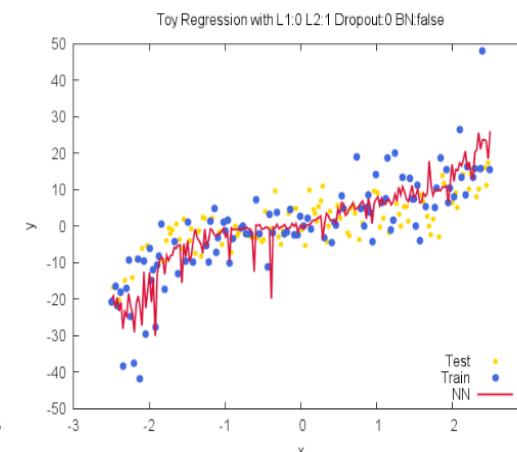
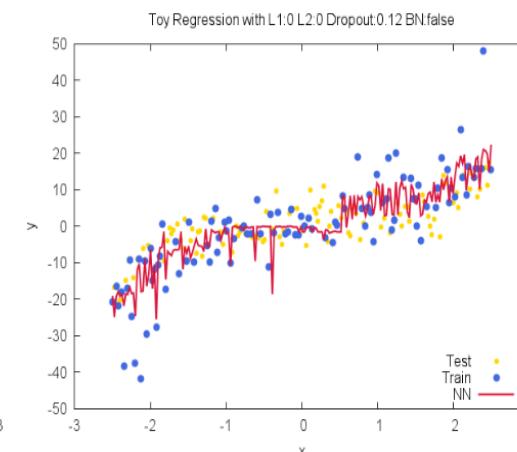
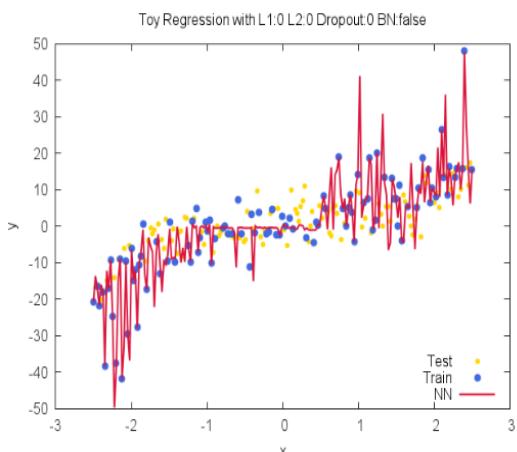
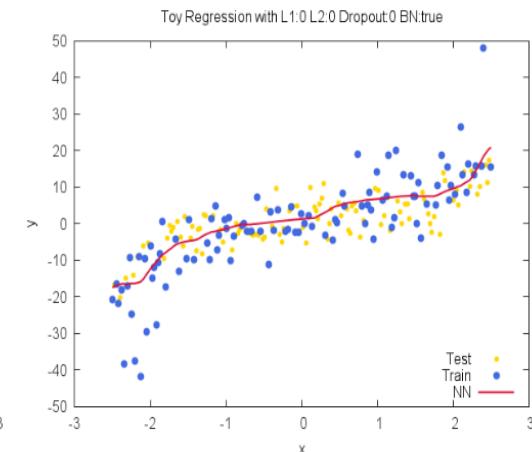
Dropout



L2 Regularization



Batch Normalization

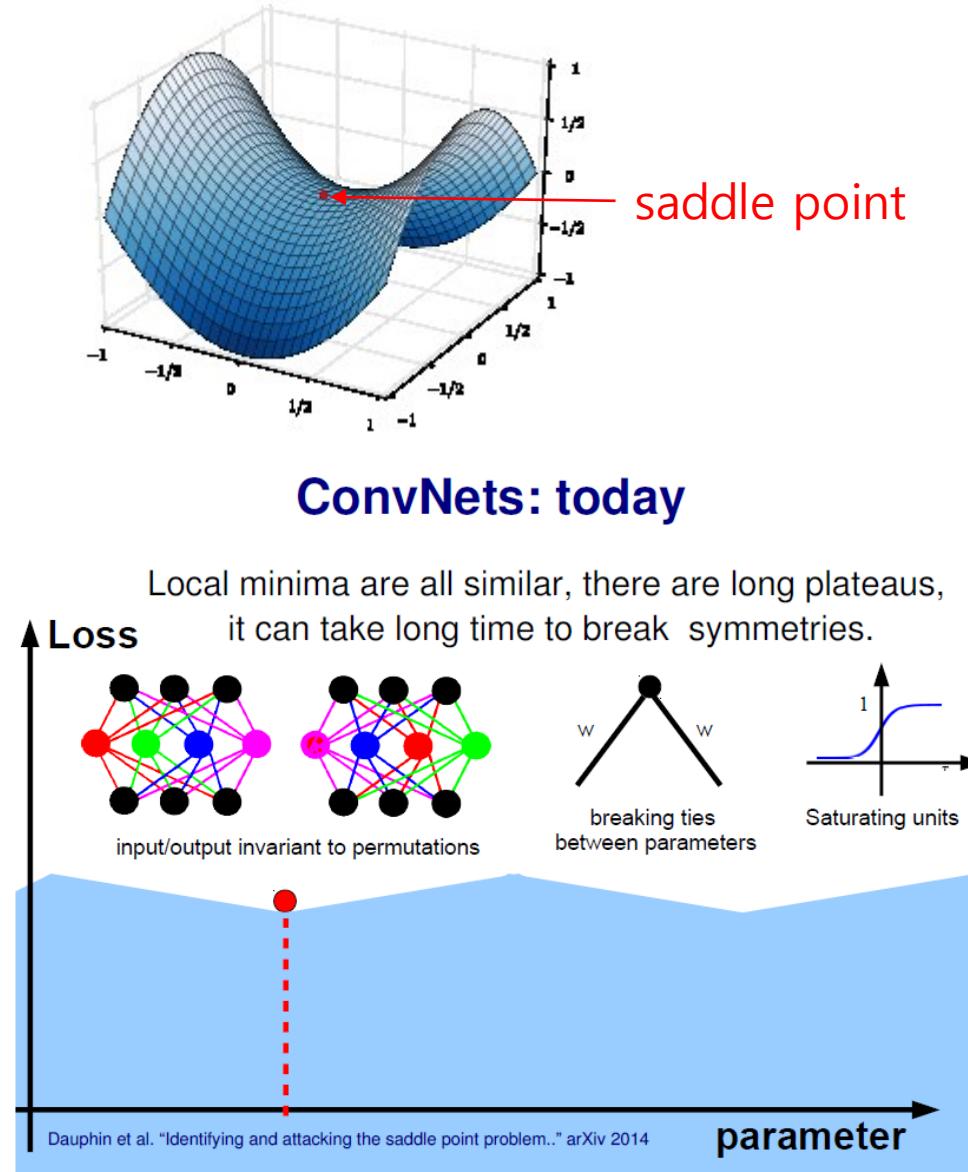


# Local Minima

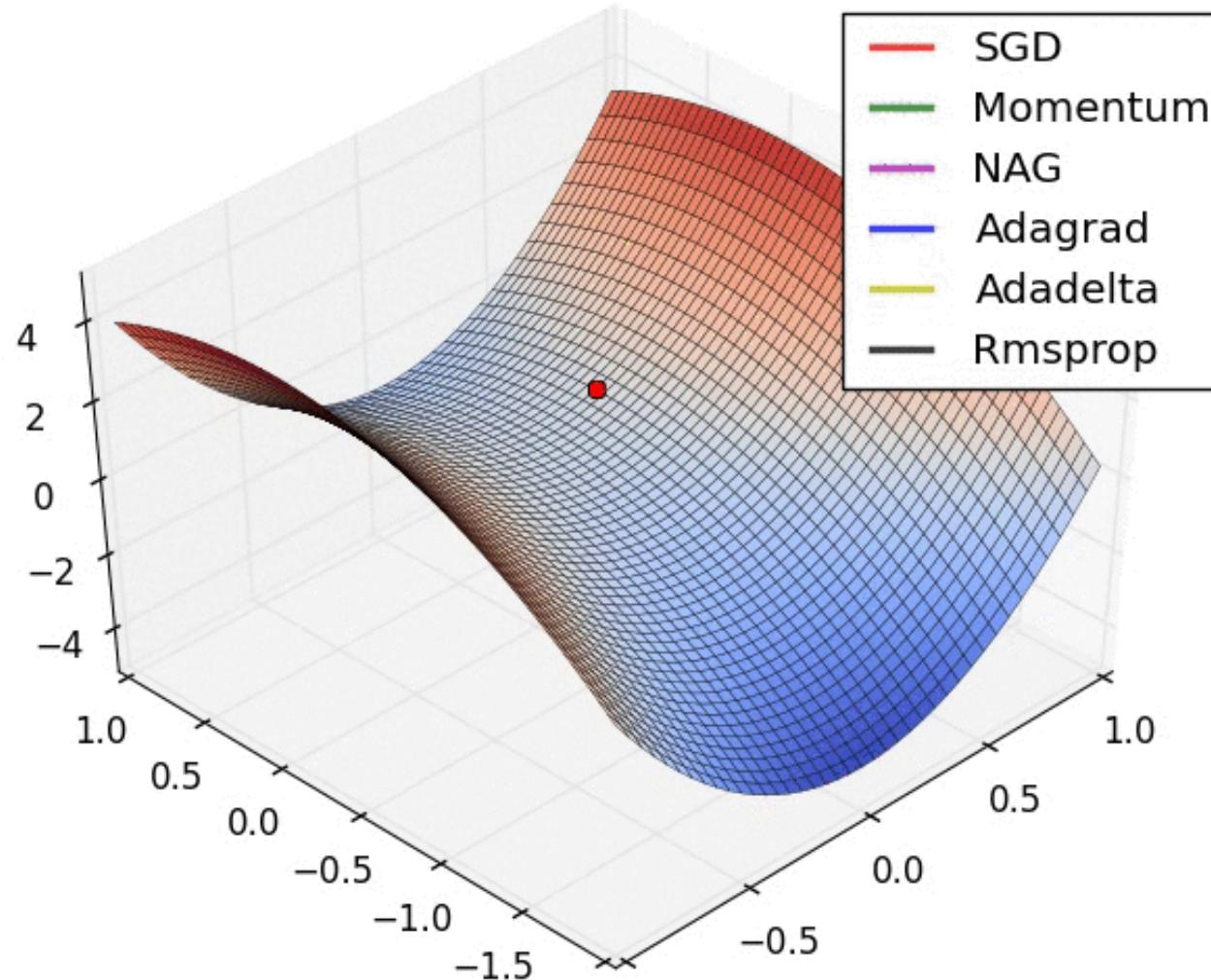


# Local Minima에 대하여

- minimum이라고 하는 것은 현재 차원에서 이동할 수 있는 모든 방향으로의 gradient 값이 증가하는 방향이어야 하는데 이런 경우는 확률적으로 희박함
- DNN과 같은 고차원 구조에서는 대부분은 local minima가 아니라 saddle point일 가능성이 높음
- 만약 실제 local minima가 존재한다면 그것은 global minimum과 거의 차이가 없을 가능성이 높음(neural network의 대칭성)



# Optimization Methods

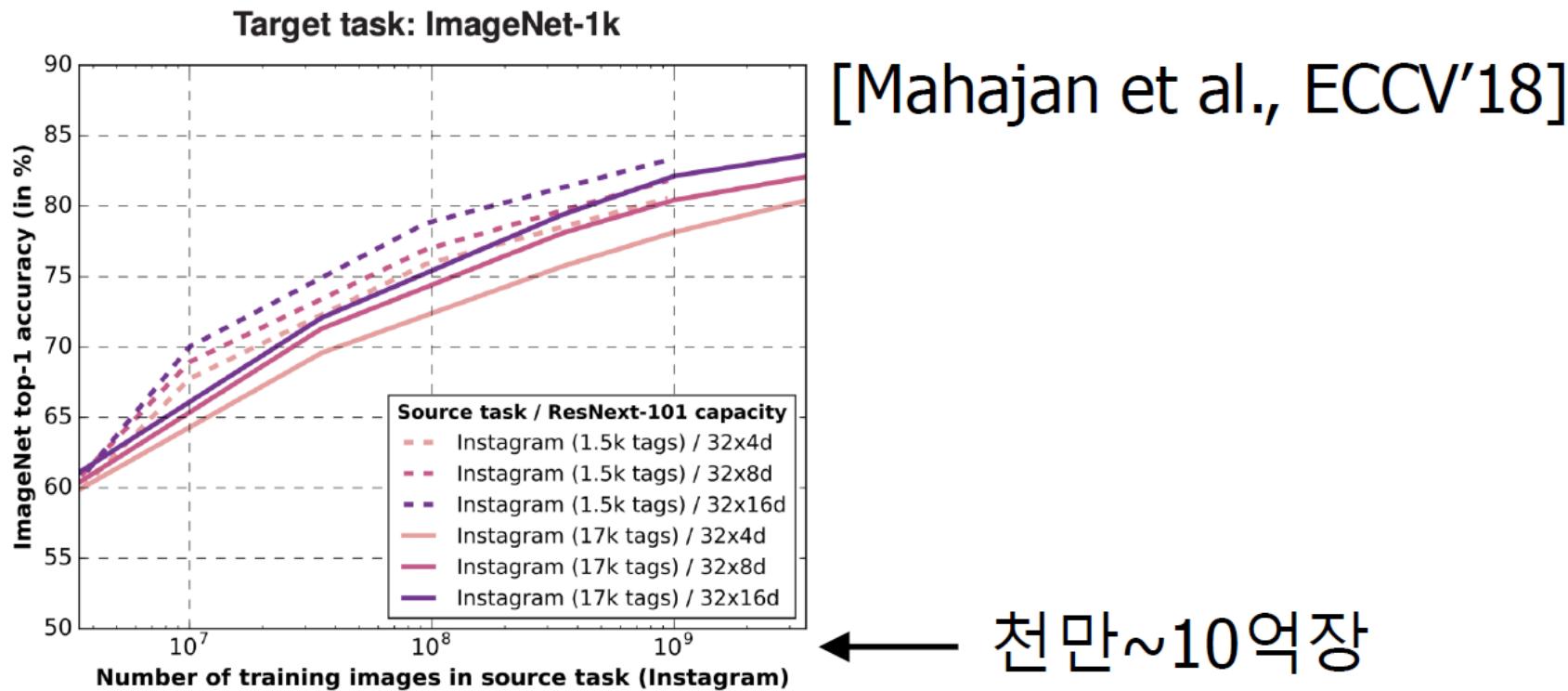


# 그 밖에 Deep Learning을 가능하게 했던 것들

- Hardware
  - CPUs, GPUs!, ASICs <https://youtu.be/-P28LKWTzrl>
- Organized Large Datasets
  - ImageNet
- Algorithms and Research
  - Backprop, CNN, LSTM
- Software and Infrastructure
  - Git, AWS, Amazon Mechanical Turk, TensorFlow, ...
- Financial Backing of large Companies
  - Google, Facebook, Amazon, ...

# Data is Important!

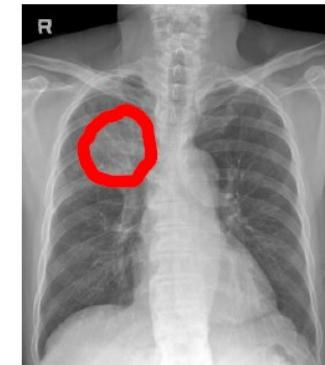
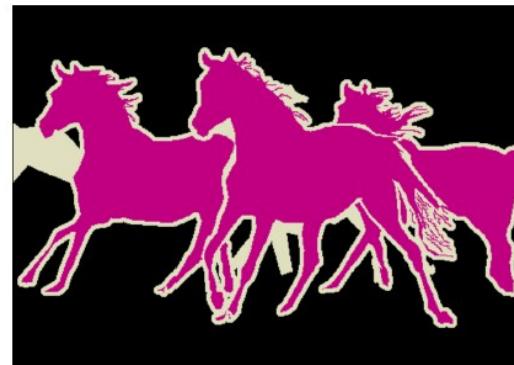
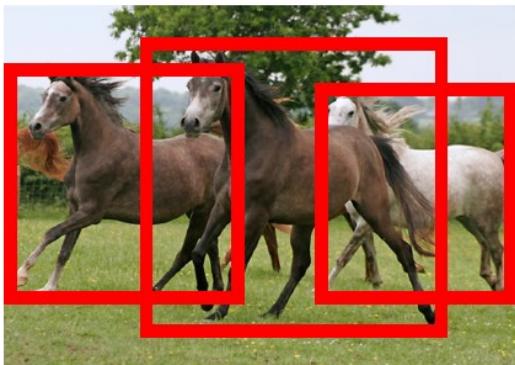
- It is not questionable that  
**more data still improves network performance**



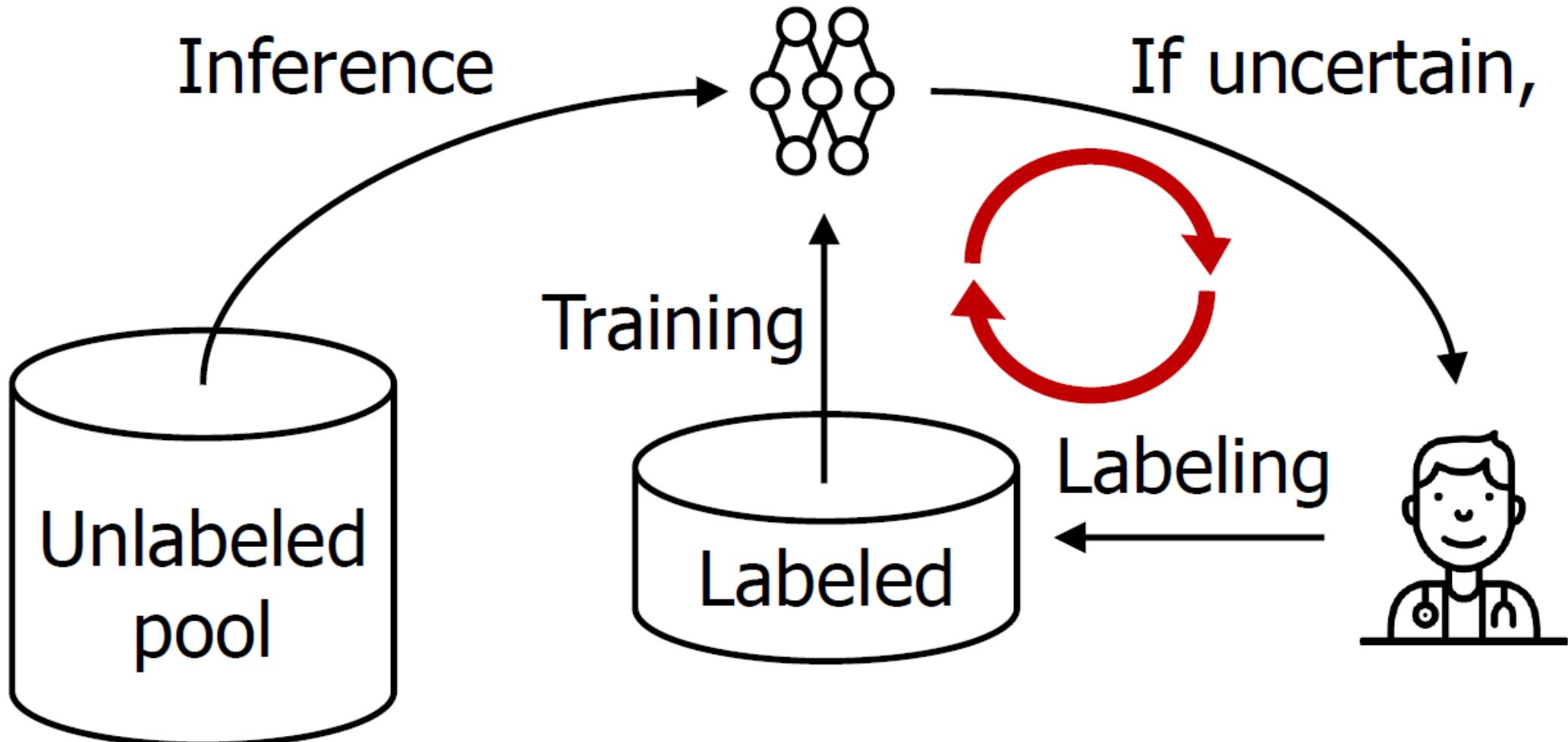
# Data is Expensive!



\$

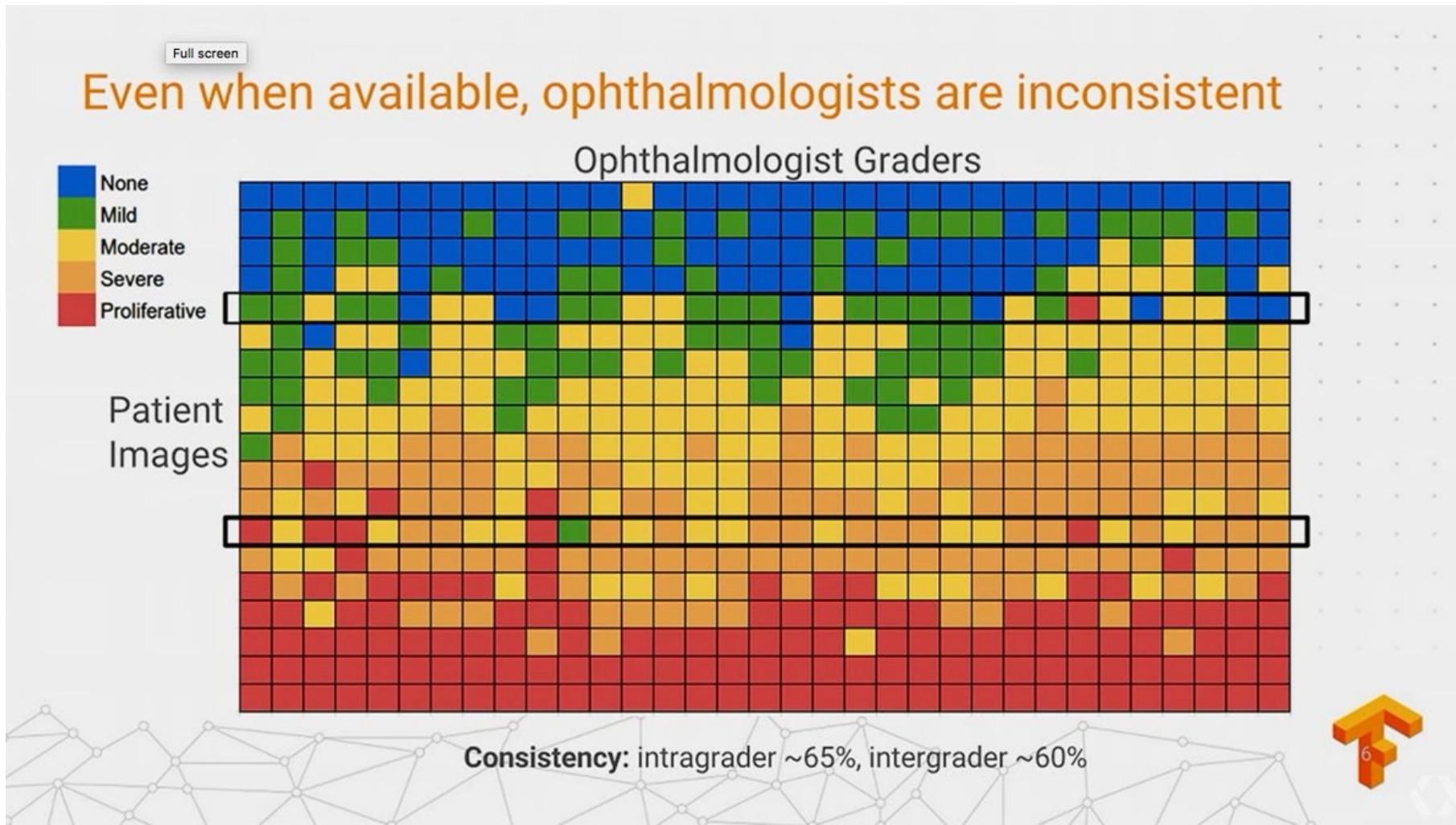


# Active Learning



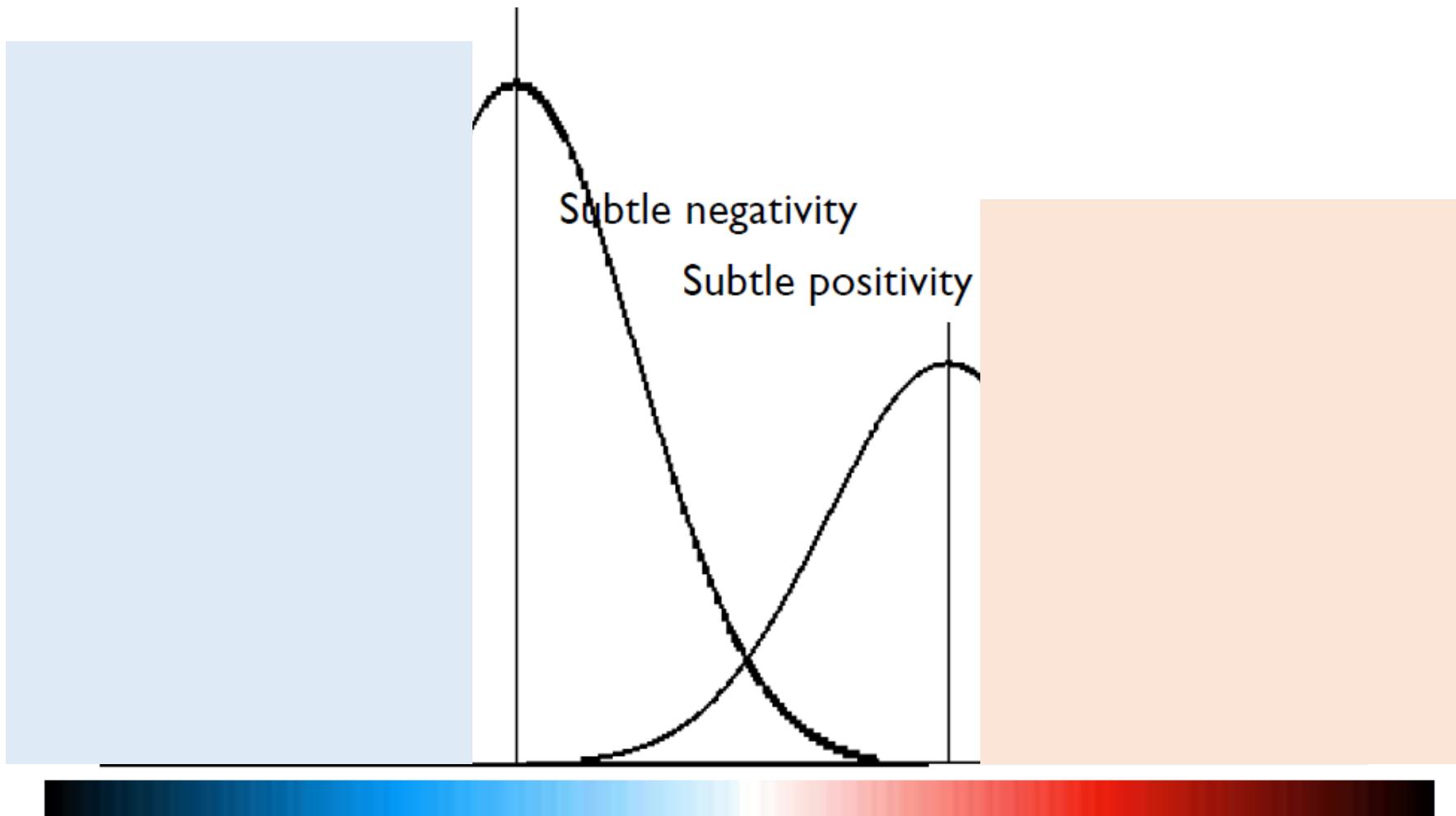
# Good Data vs Bad Data

- Our labels are perfect?



# Good Data vs Bad Data

- Our data is unbiased?



# Is Our Model Good Enough?

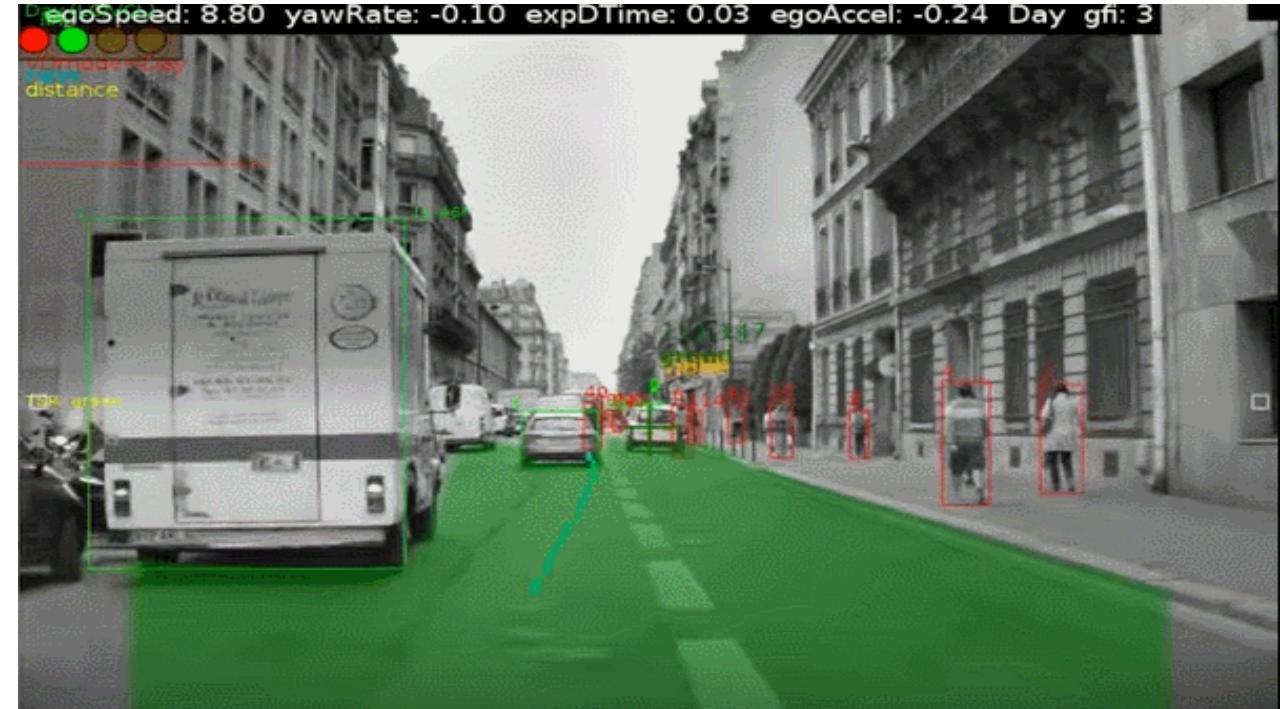
- Sometimes accuracy is not a good measurement

		Predicted class	
		$P$	$N$
Actual Class	$P$	True Positives (TP)	False Negatives (FN)
	$N$	False Positives (FP)	True Negatives (TN)

**Sensitivity**      **Recall**      **Precision**

**Specificity**

# Deep Learning's Challenges – Toward the Real World

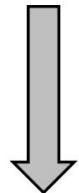


# Rethinking Generalization

- Training with random label



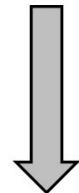
~~dog~~



banana



~~banana~~



dog



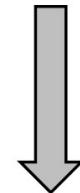
~~dog~~



tree



~~tree~~



dog

# Rethinking Generalization



# Unintended Consequences

## Human Play



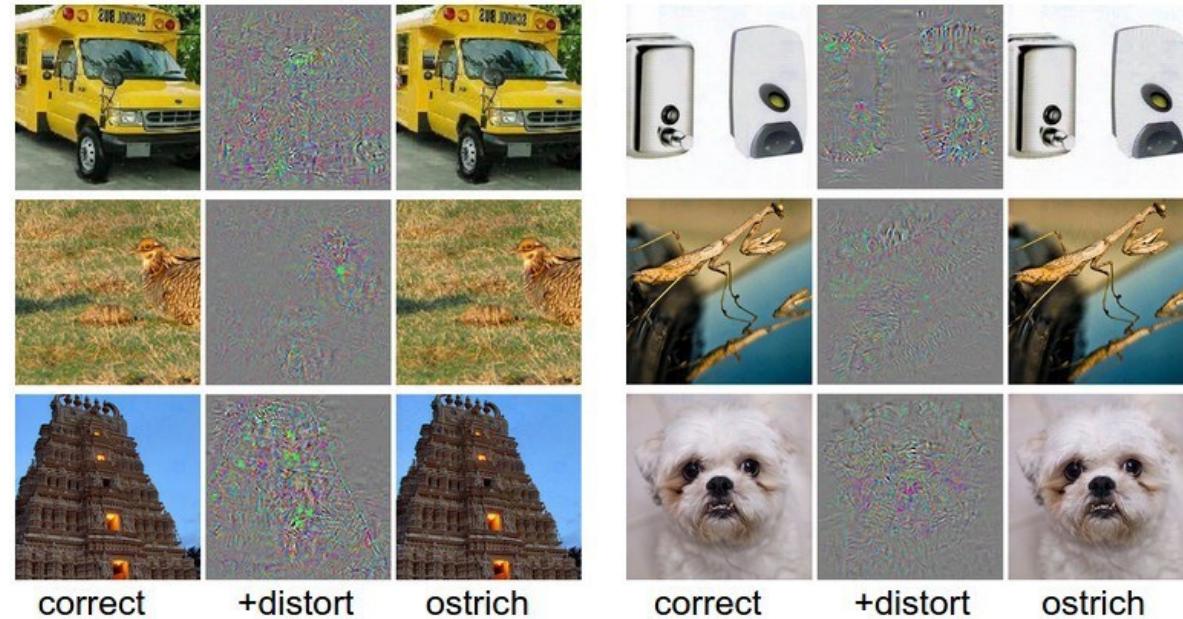
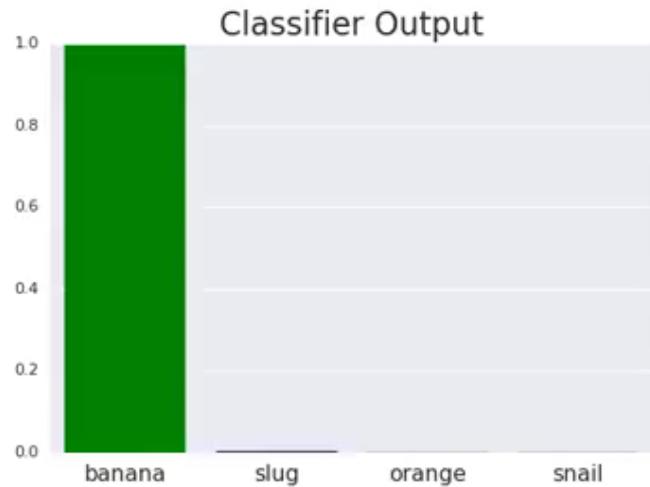
## AI Play



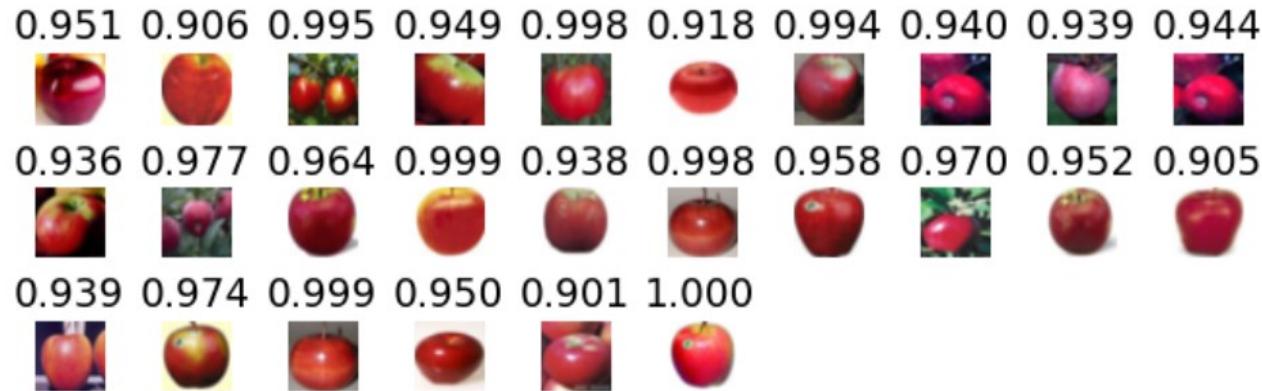
Player gets reward based on:

1. Finishing time
2. Finishing position
3. Picking up “turbos”

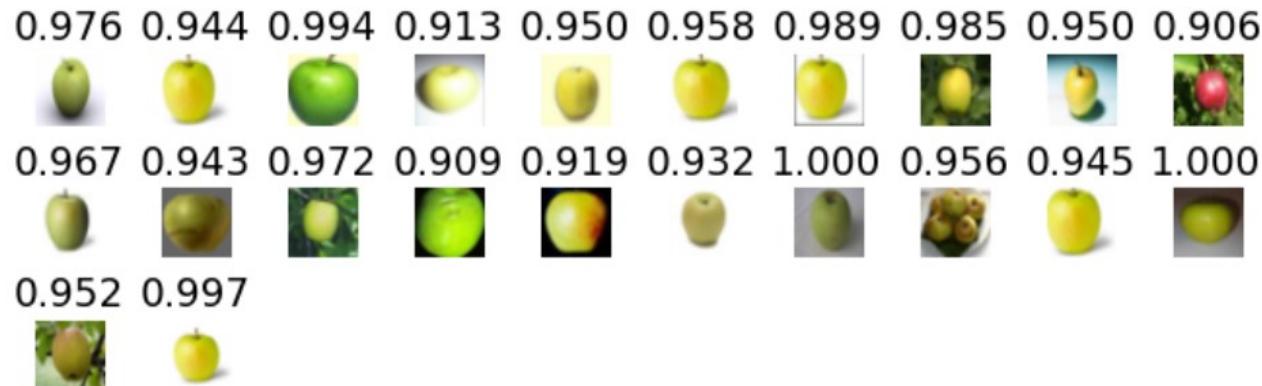
# Adversarial Attack



# Uncertainty

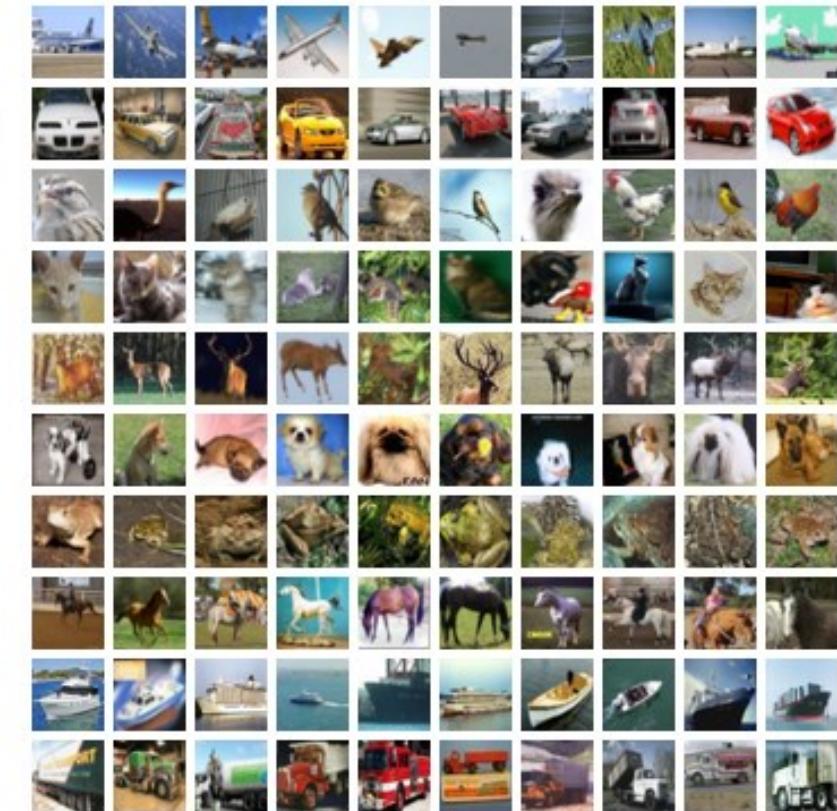


CIFAR-100's *apple* misclassified as CIFAR-10's *automobile* class with  
 $p > 0.9$ .

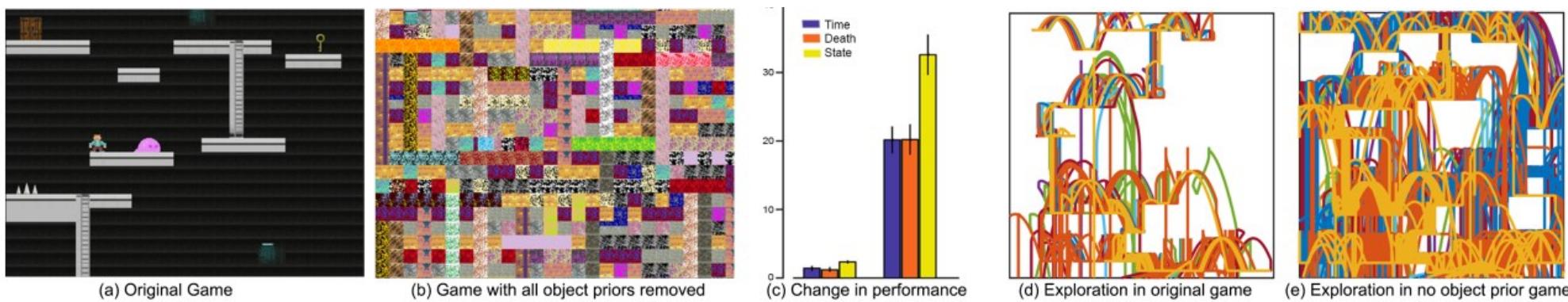
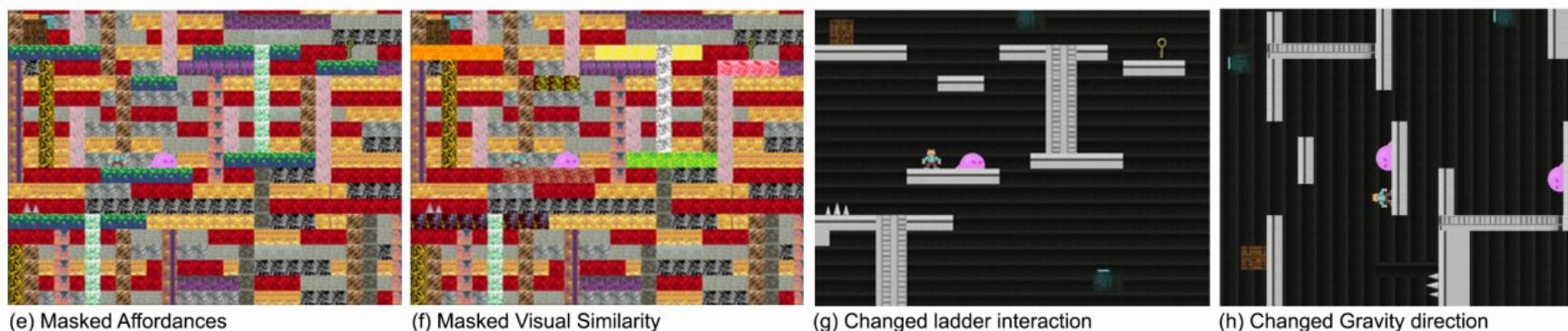
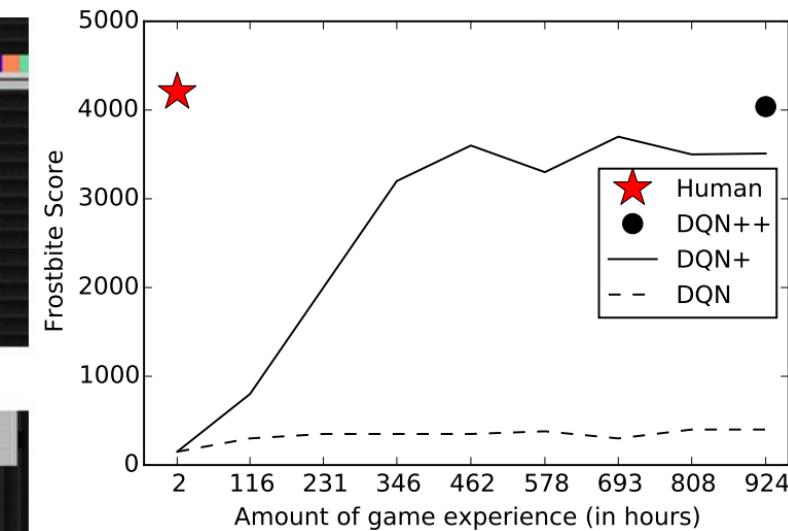
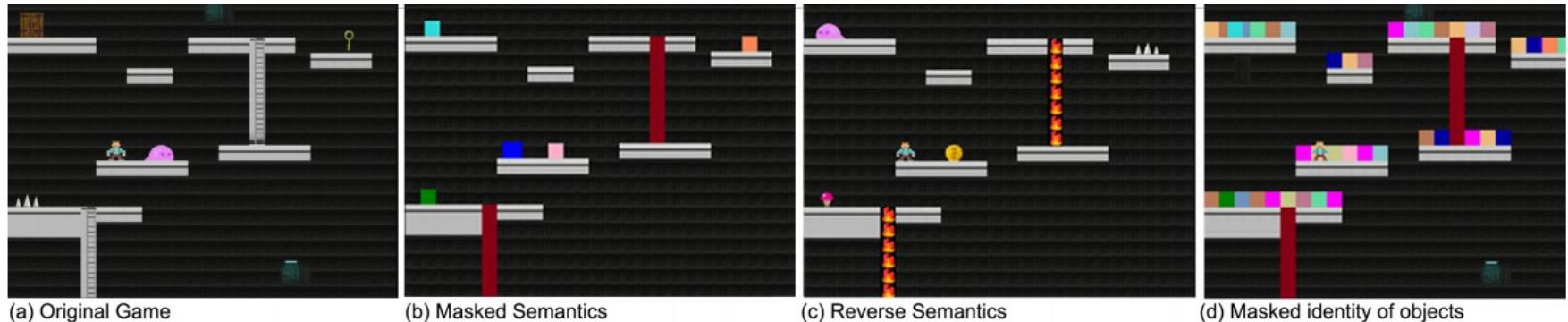


CIFAR-100's *apple* misclassified as CIFAR-10's *frog* class with  $p > 0.9$ .

Cifar 10 dataset



# Inefficient Training



# AI의 현재 – 모방



L'Avion III de Clément Ader, 1897  
(Musée du CNAM, Paris)

# AI의 현재 – 해석은 미래에...

