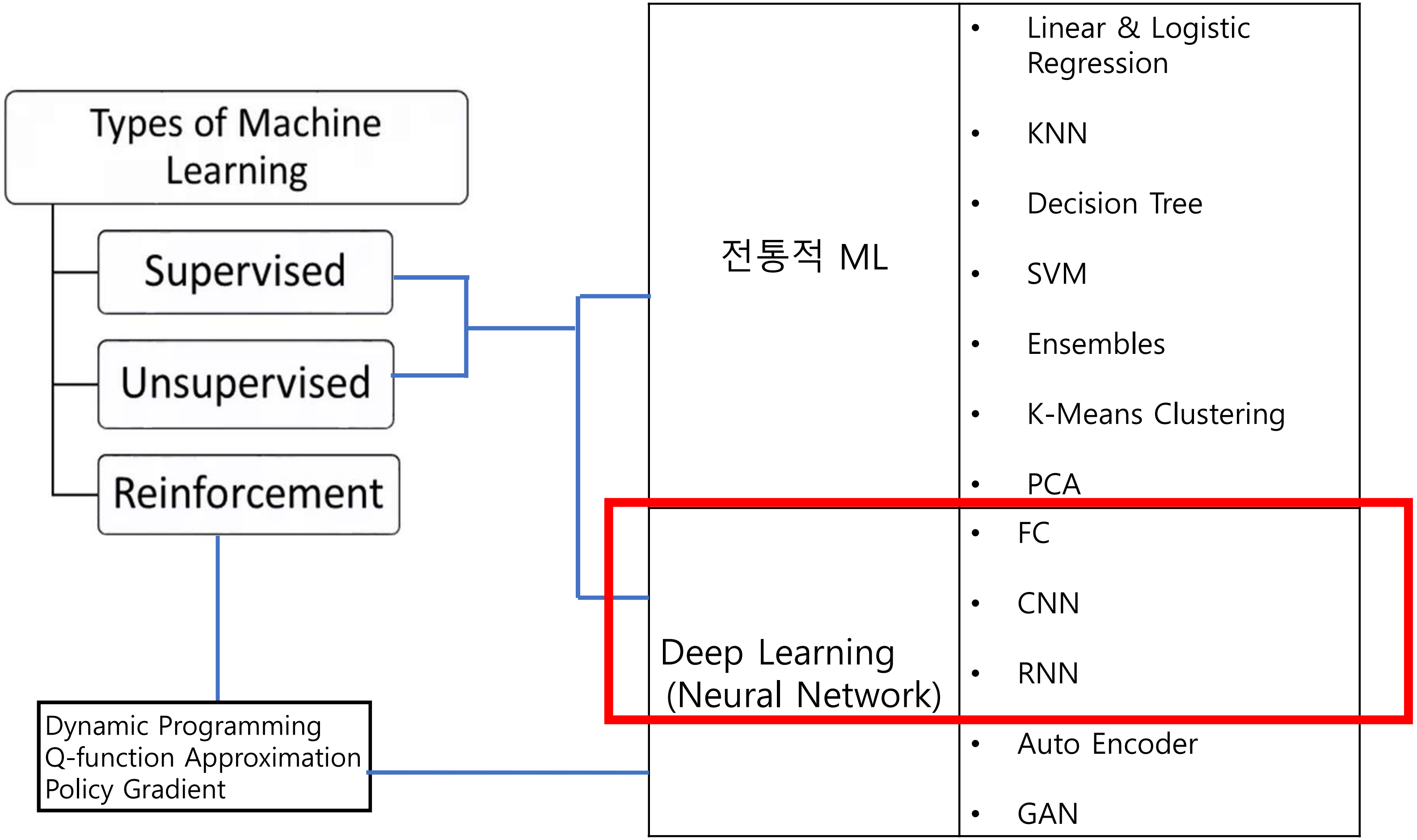
Neural Network
and
Deep Learning

Tensorflow Installation

- pip install --upgrade tensorflow

- import tensorflow as tf
- tf.__version__

Machine Learning 모델의 종류



Classical Machine Learning vs. Deep Learning

Machine Learning

-. Hunan-crafted features
-. Great fit for data mining applications

컴퓨터가 이해할 수 있도록 Domain 지식 및 통계학적 지식을 바탕으로 Feature 를 잘 만들어서 Data 를 구성

알고리즘 학습

80~90 % 의 비중

Domain 전문지식을 가진 석,박사급 인재 필요

10~20% 의 비중

각 feature 의 weight 를 optimize

Deep Learning

- 중요한 Feature 를 스스로 구분하여weight 를 부여

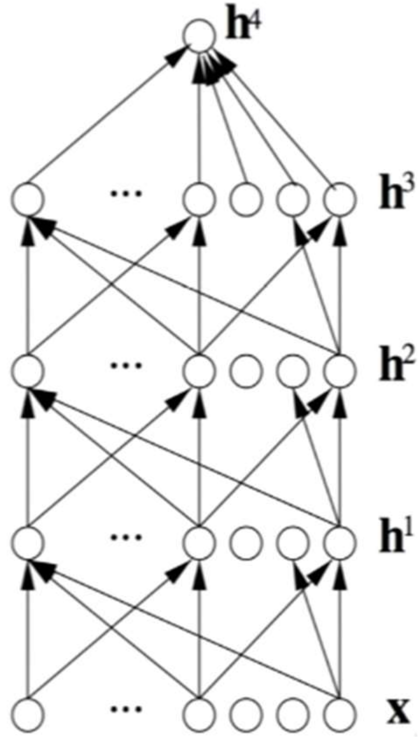
  - 사람이 manually 정해준 feature 는 over-specified, incomplete
  위험성 있고 작성에 많은 시간 소요

- 여러 층에 걸친 내부 parameter 를 스스로 학습
  - 적용하기 쉽고 빠르다.

- Raw data 를 거의 그대로 사용 – computer vision, 언어처리 등
  (ex, image, sound, characters, words)

- Unsupervised, supervised learning 모두 가능

- Great fit for hard vision, speech, language problem



Artificial Neuron (Perceptron)

구성요소:

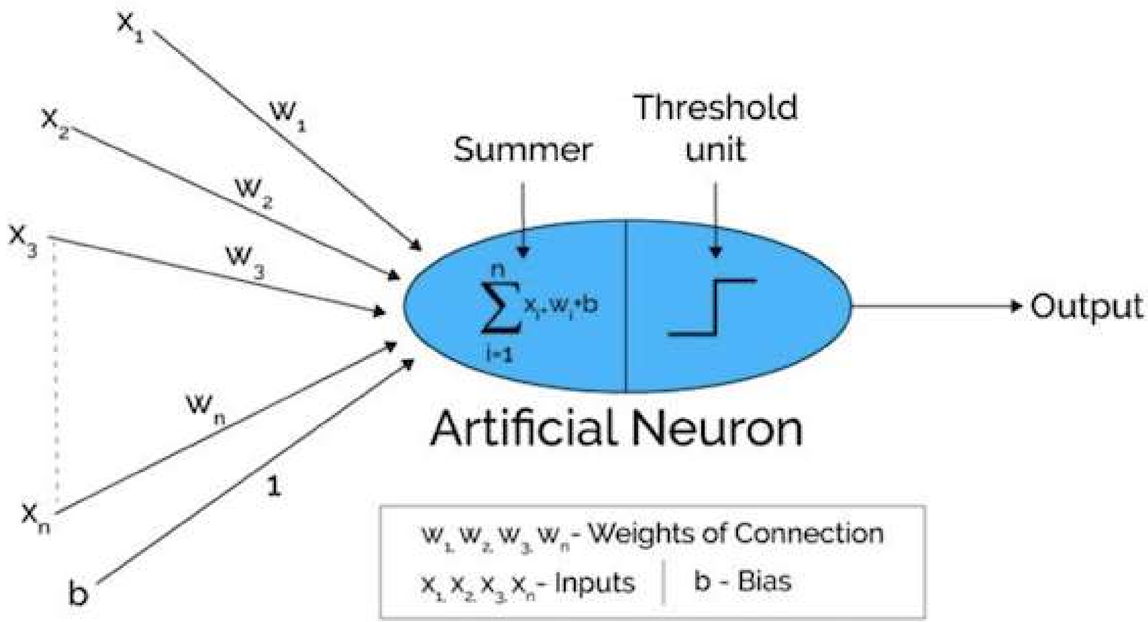
Pre-Activation :

a(x) = $b + \sum_i w_i x_i = b + w^T X$

Activation :

h(x) = g(a(x)) = g($b + \sum_i w_i x_i$)

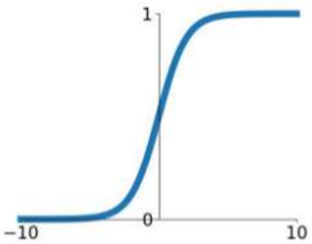
w : connection weights
b : bias
g : activation function



Activation Functions

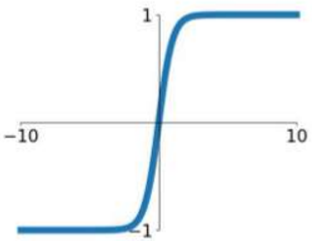
Sigmoid

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



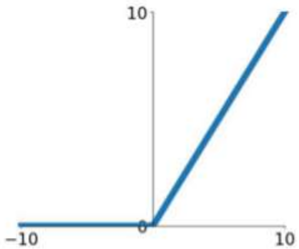
tanh

$\tanh(x)$



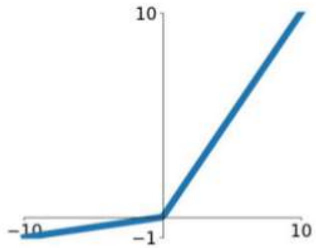
ReLU

$\max(0, x)$



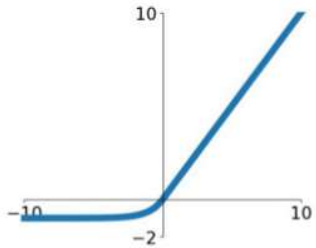
Leaky ReLU

$\max(0.1x, x)$



ELU

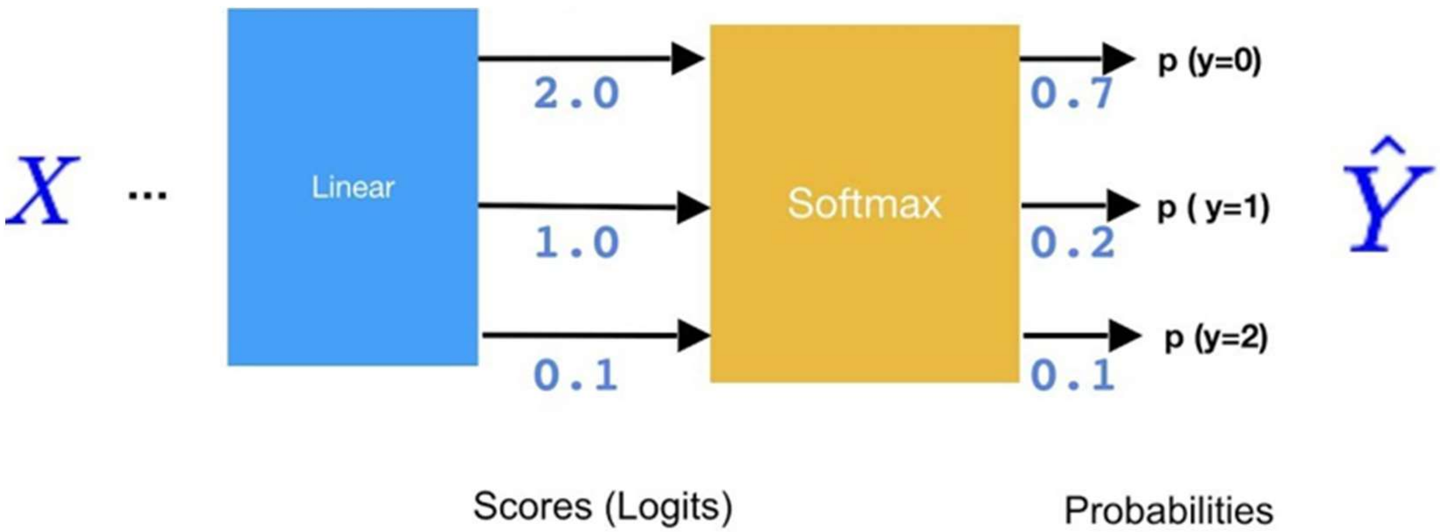
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



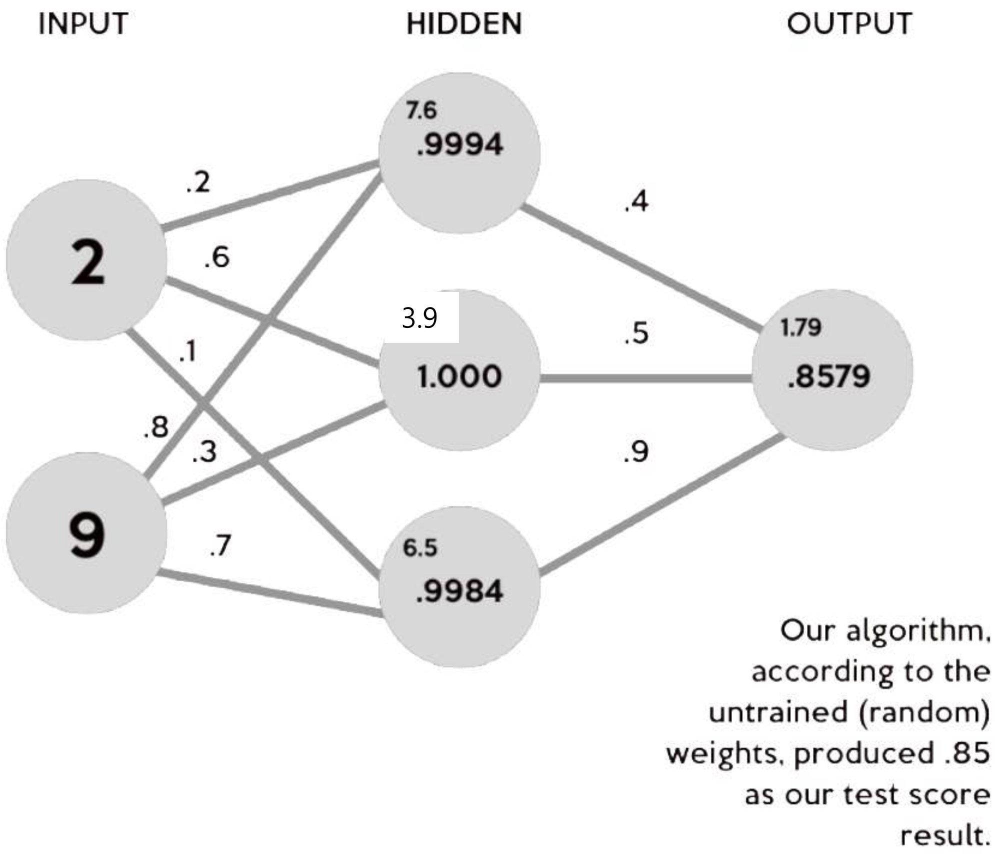
Softmax

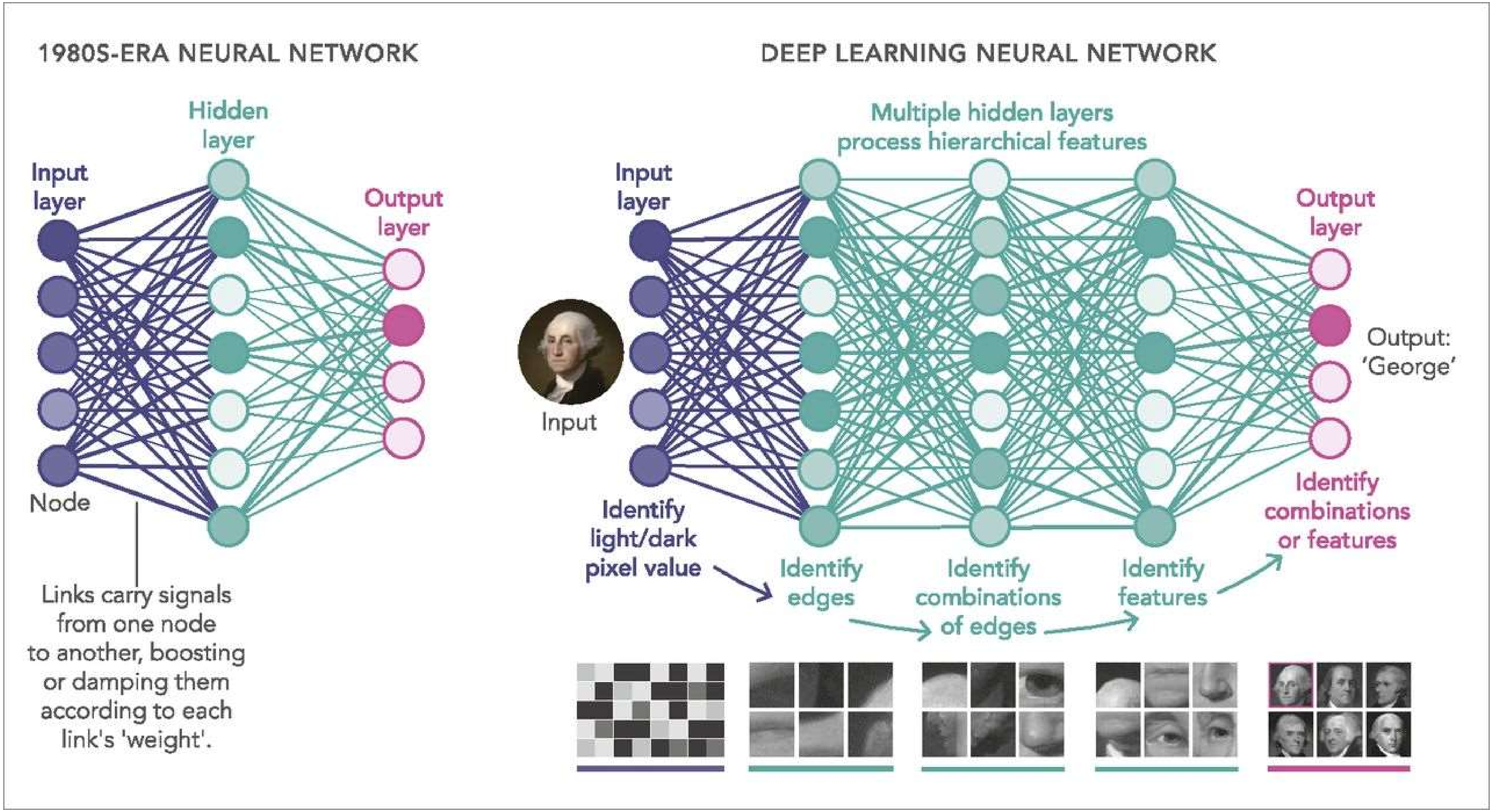
- 출력값의 class 분류를 위하여 출력값에 대해 **정규화 → 확률 분포 출력**

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



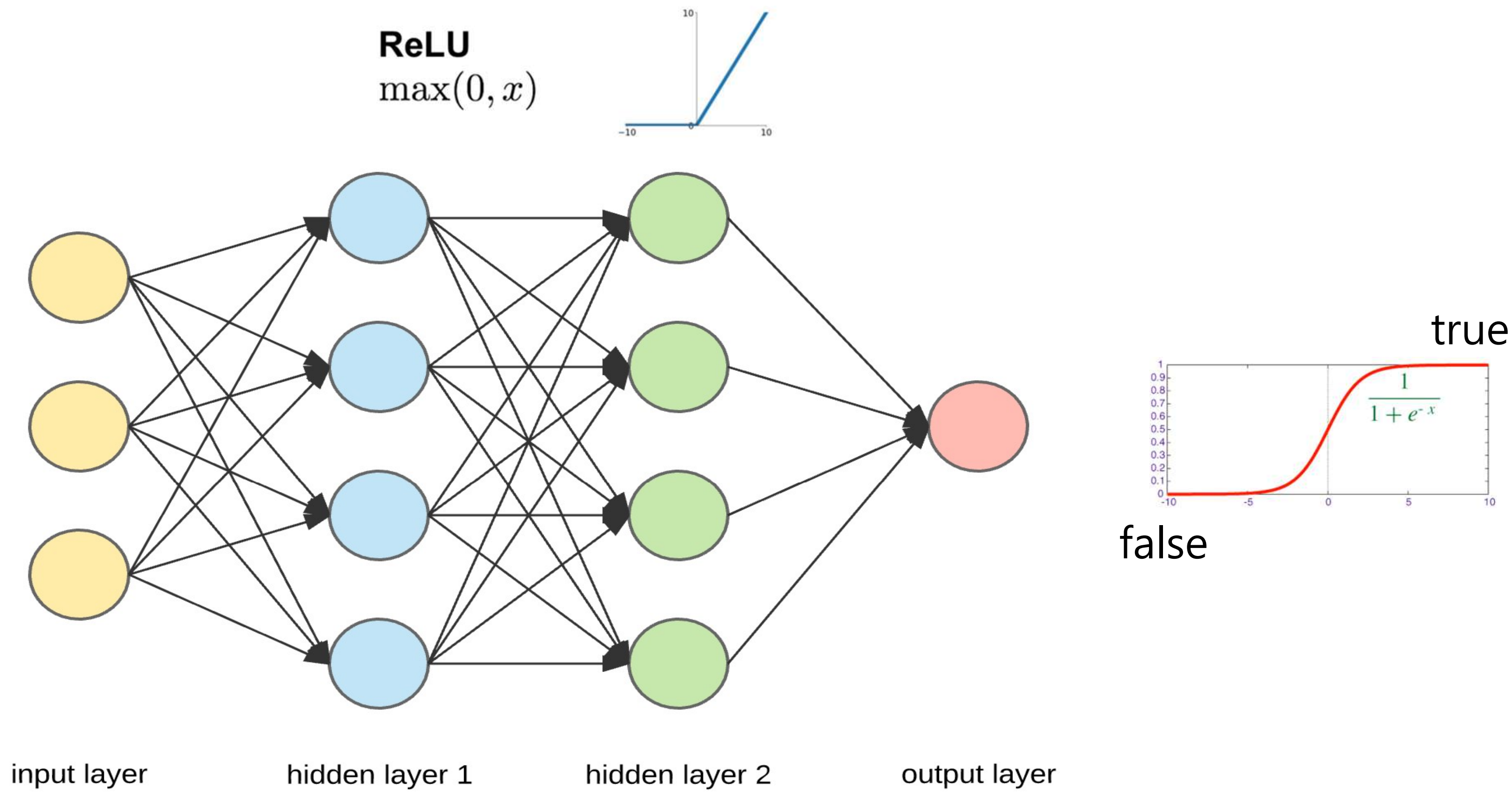
Neural Network 의 작동 원리



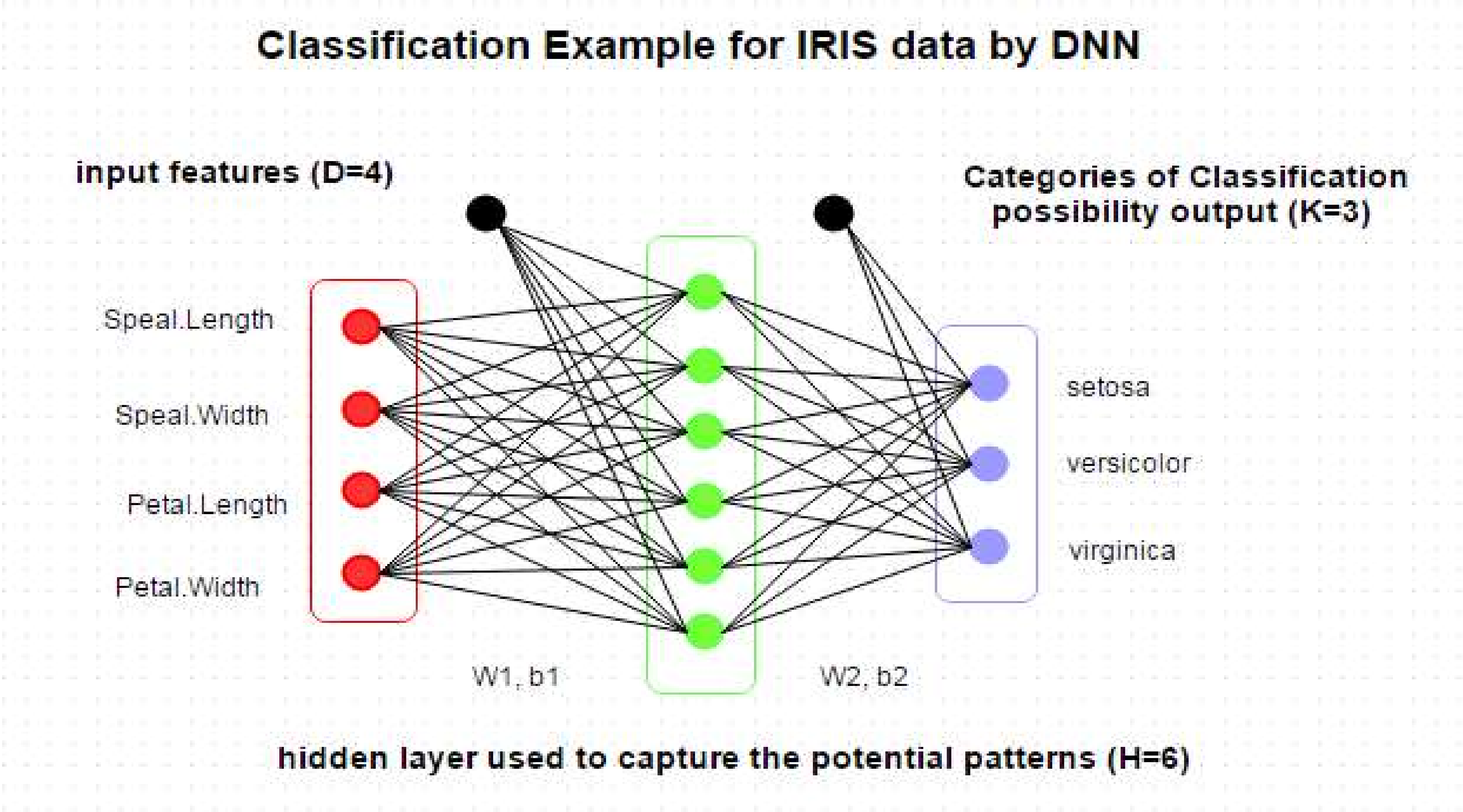


1980S-ERA NEURAL NETWORK
Hidden layer
Input layer
Output layer
Node
Links carry signals from one node to another, boosting or damping them according to each link's 'weight'.
DEEP LEARNING NEURAL NETWORK
Multiple hidden layers process hierarchical features
Input layer
Output layer
Input
Output: 'George'
Identify light/dark pixel value
Identify edges
Identify combinations of edges
Identify features
Identify combinations or features

Binary Classification (Sigmoid)



Multi-Class Classification (Softmax)



Gradient Descent (경사하강법)

- Yhat = θX 의 θ 를 inference 하는 방법

  X = m x (n+1) matrix

  y = m dimensional vector

- OLS (Ordinary Least Squares) method 의 문제점

  1. OLS 는 Normal Equation 을 이용

     θ = $(X^T X)^{-1} X^T y$

  2. O(n^3) 의 complexity 를 가진다. (n : feature 수)
  3. large data set, large # of features 에는 부적합
  4. Regularization term 을 추가할 수 없음
  5. N > n 만큼의 Data 필요 (N: data 개수)

Gradient Descent for Linear Regression

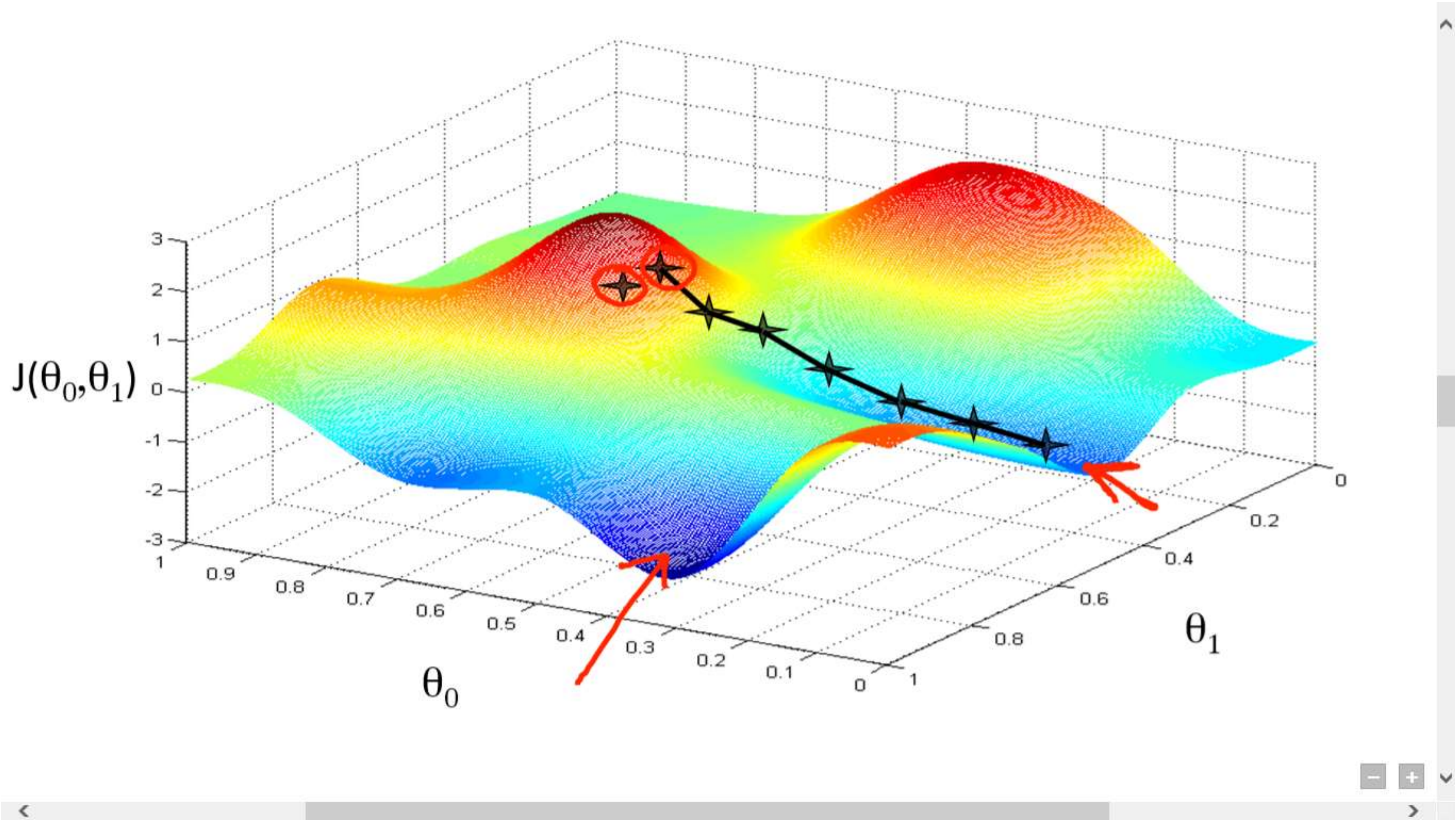
- Hypothesis : $h_\theta(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots\ldots + \theta_n x_n$
$= \theta^T X$

$$\theta^T = [\theta_0, \theta_1, \ldots., \theta_n]$$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ x_n \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ . \\ . \\ y_n \end{bmatrix}$$

- Cost Function : $J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

➔ 미분 가능 / convex



3
2
1
0
-1
-2
-3
$J(\theta_0,\theta_1)$
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

Gradient Descent for Logistic Regression

- Hypothesis : $\sigma(\theta^T X) = \frac{1}{1+e^{-\theta^T X}}$

$$\theta^T = [\theta_0, \theta_1, \dots, \theta_n]$$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ x_n \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ . \\ . \\ y_n \end{bmatrix}$$

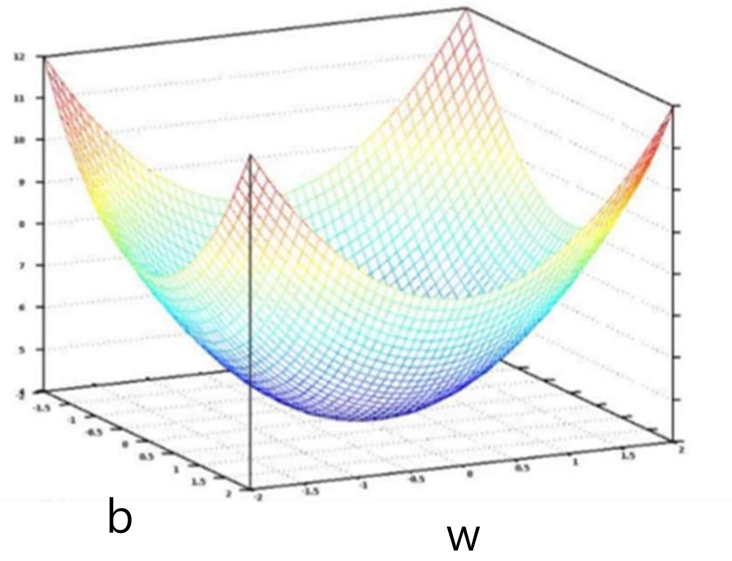
- Cost Function : $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log(\hat{y}^{(i)}) + \left(1 - y^{(i)}\right) \log(1 - \hat{y}^{(i)}))$

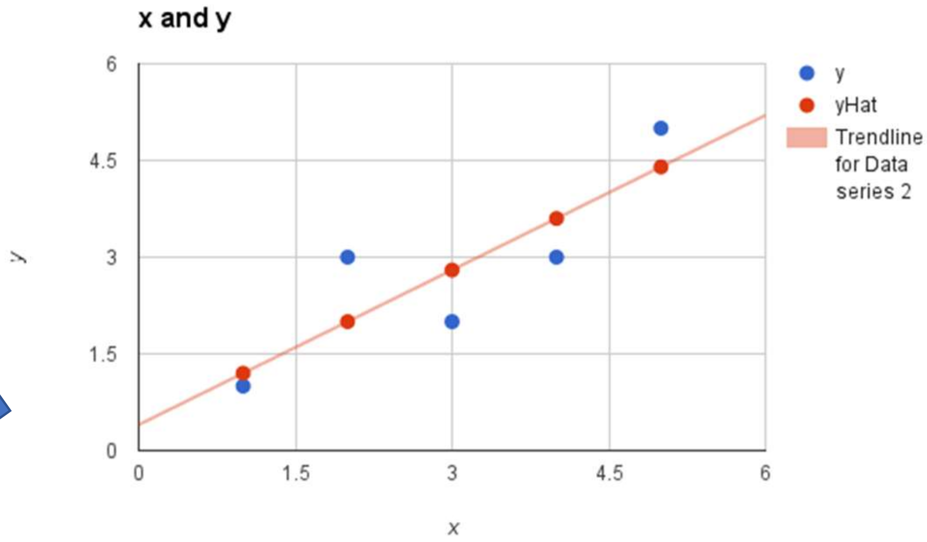
if y = 1 : $J(\theta) = -\log(\hat{y}^{(i)})$

y = 0 : $J(\theta) = -\log(1 - \hat{y}^{(i)})$

Cost Function - Linear Regression

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i} - Y_i)^2$$





y = Wx + b

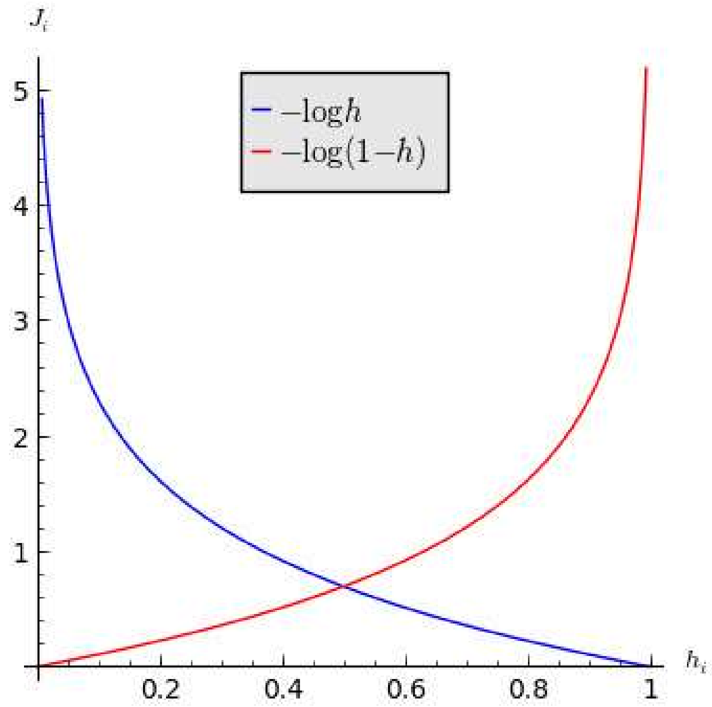
MSE 를 최소화 하는
W 와 b 를 optimize

Cost Function - Logistic Regression (Binary Cross-entroy)

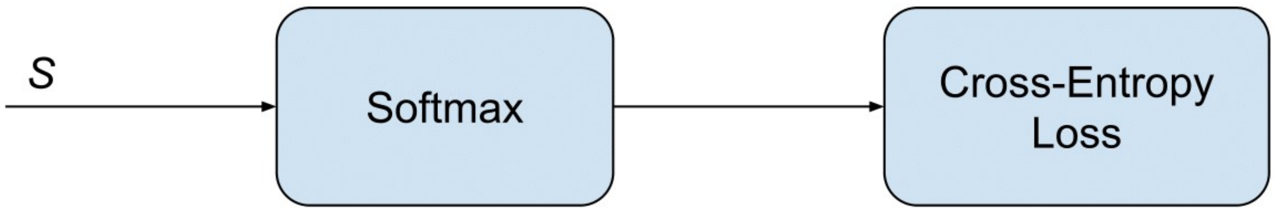
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_\theta(x^{(i)})) + y^{(i)}\log h_\theta(x^{(i)})\right]$$

If $y^{(i)} = 1$: $J(\theta) = -y^{(i)} log\ h_\theta (x^{(i)})$
where $h_\theta (x^{(i)})$ should be close to 1

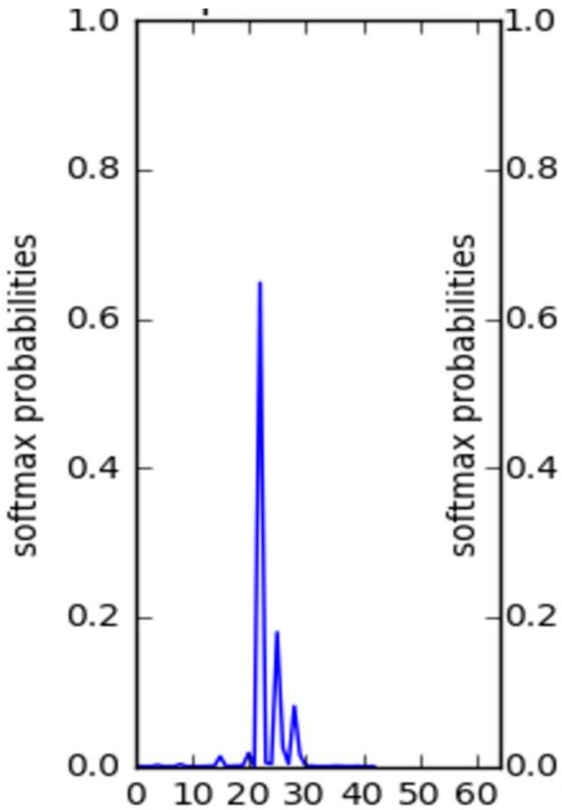
If $y^{(i)} = 0$: $J(\theta) = -\log(1 - h_\theta (x^{(i)}))$
where $h_\theta (x^{(i)})$ should be close to 0



Cost Function - Categorical Crossentroy (Softmax Loss)



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

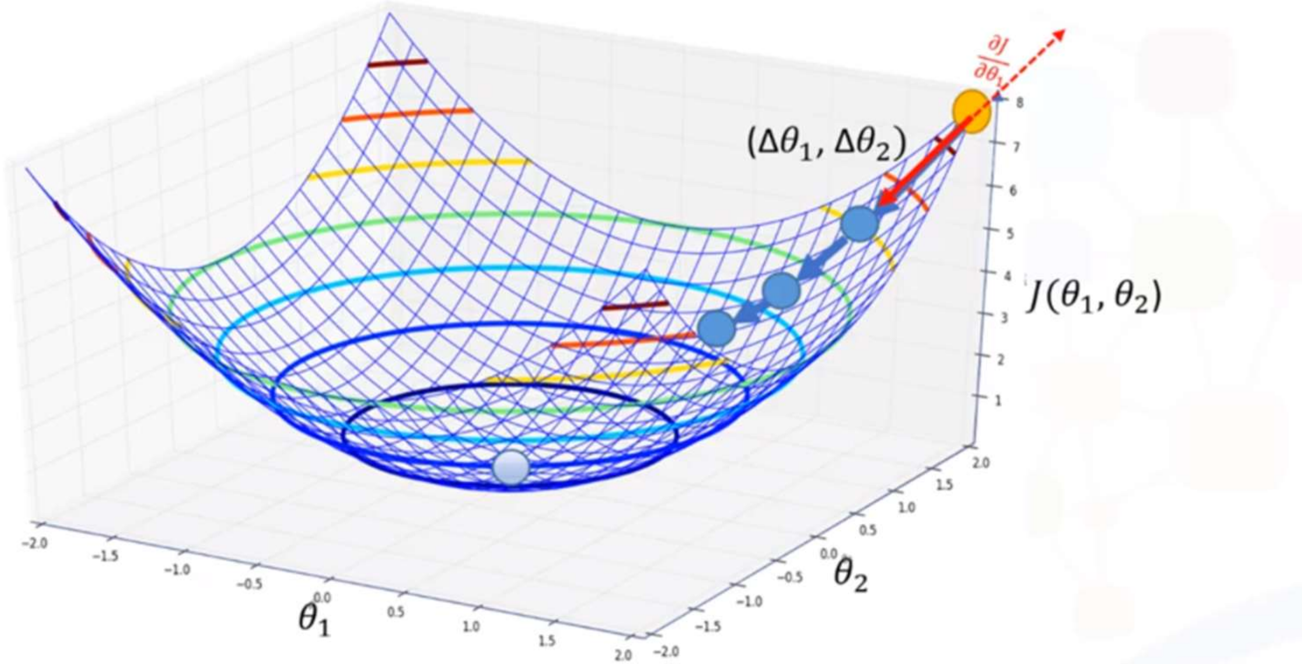


t_i : 0 이 아닌 target
(one-hot encoded 되어 있으므로 multi-class 중 오직 1 개만 1)

C: multi-classes

Goal : Minimize $J(\theta_0, \theta_1)$

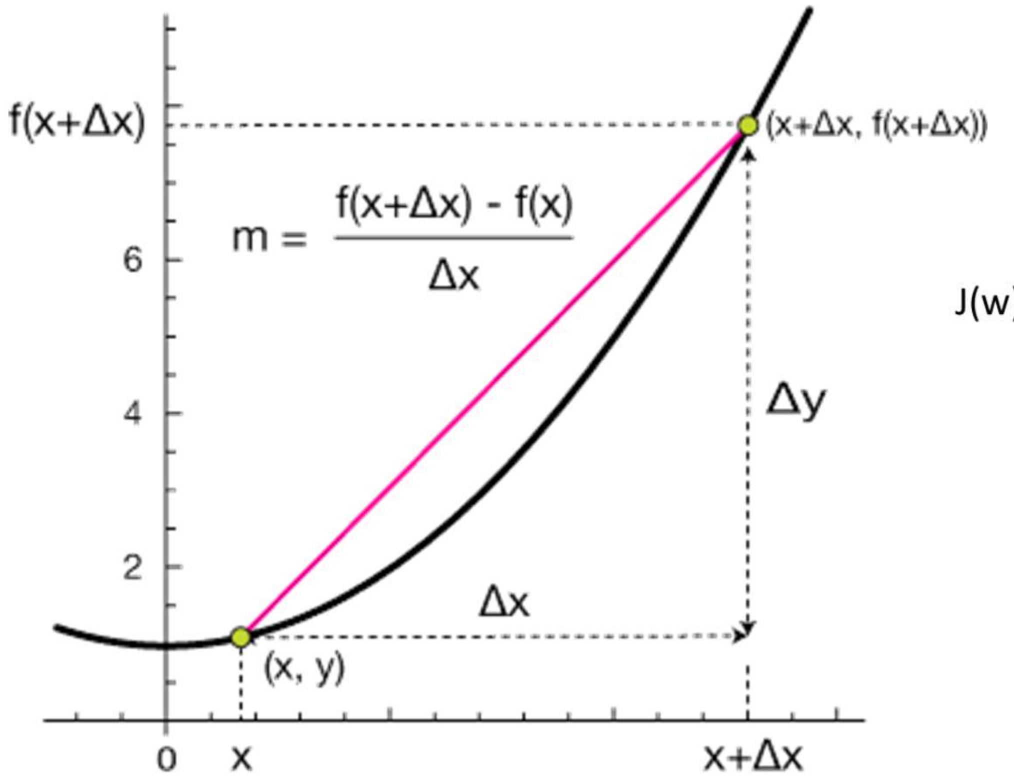
θ_0, θ_1

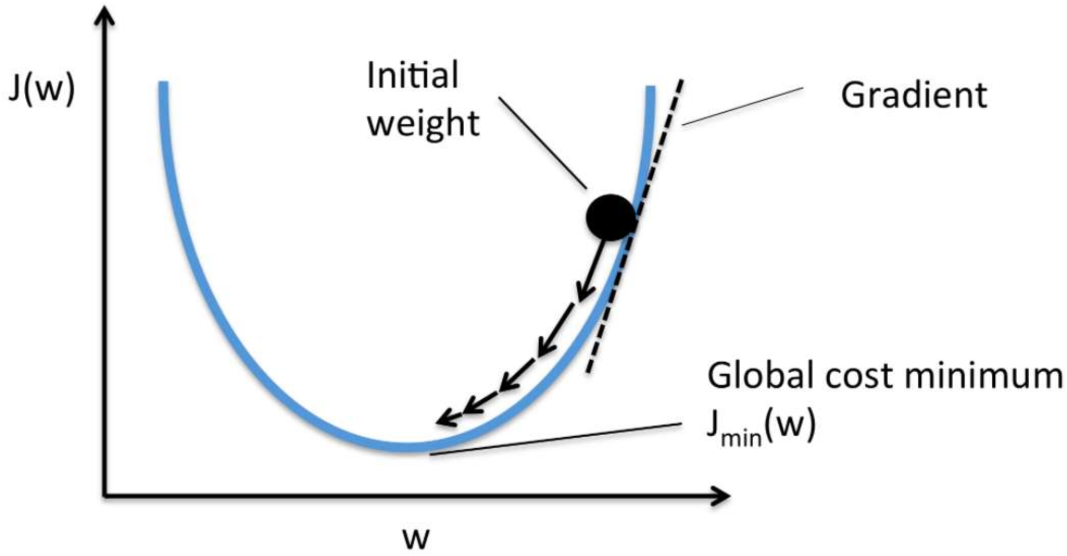


$\hat{y} = \sigma(\theta_1 x_1 + \theta_2 x_2)$

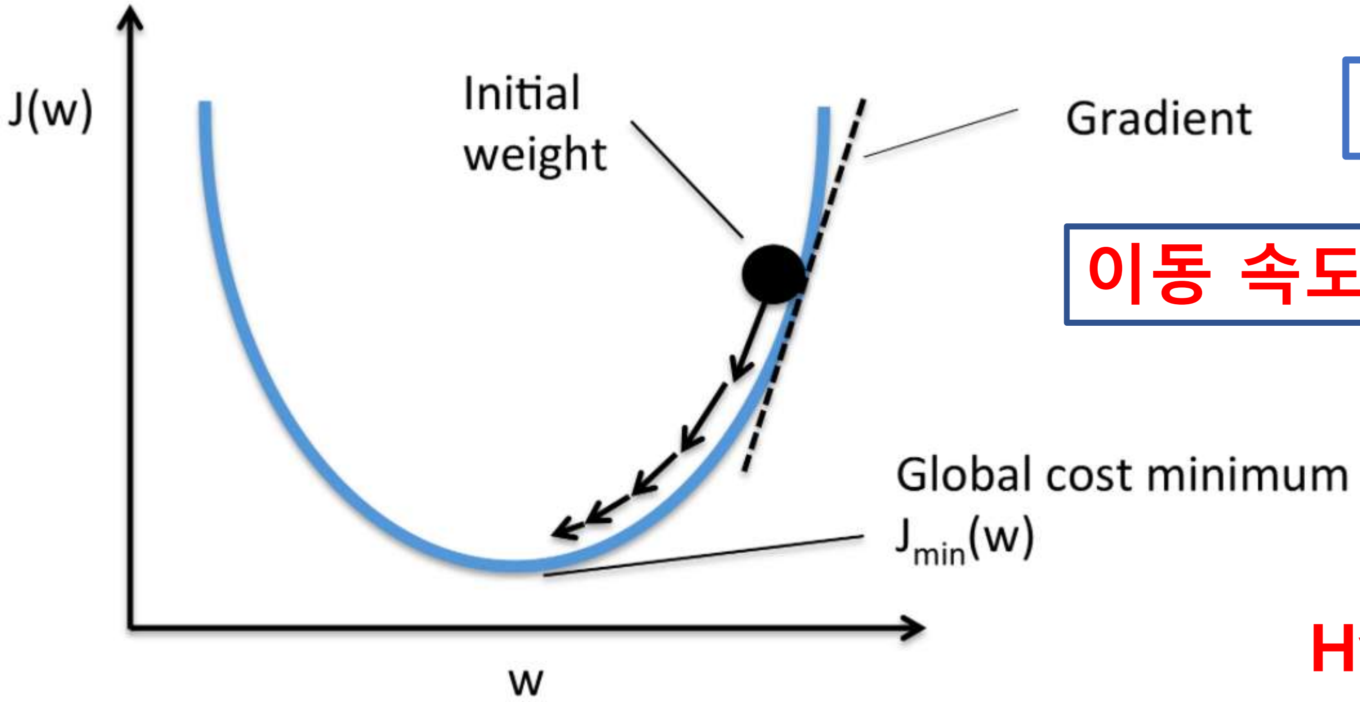
$$J(\theta) = -\frac{1}{m}\sum_{i-1}^{m} y^i \log(\widehat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i)$$

Derivative (도함수, 미분, 접선의 기울기)





Gradient Descent (경사하강법) Optimization

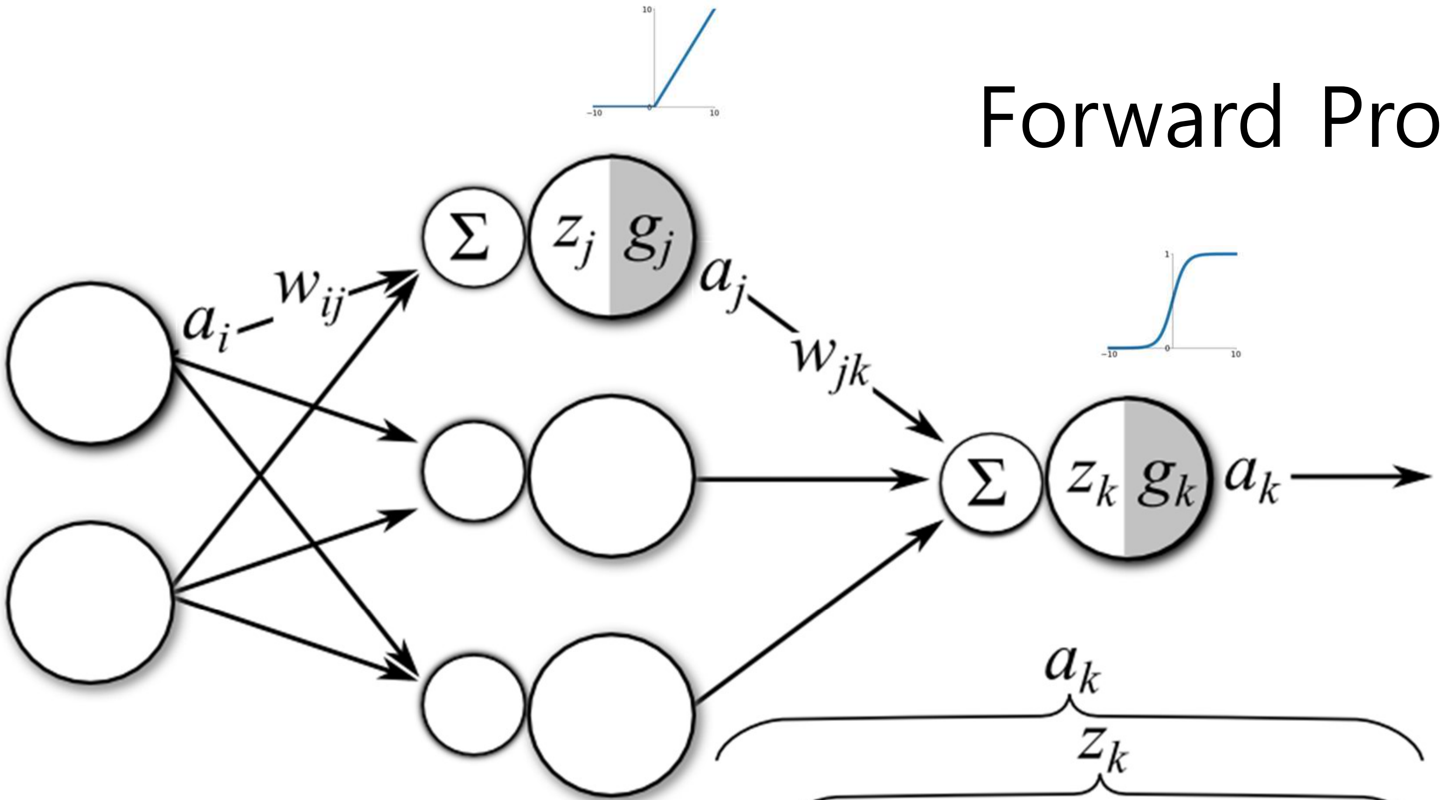


방향 – Gradient

이동 속도 – Learning Rate

Hyper Parameter

Forward Propagation



$$a_k = \overbrace{g_k(\overbrace{b_k + \Sigma_j \underbrace{g_j(\underbrace{b_j + \Sigma_i a_i w_{ij}}_{z_j})}_{a_j} w_{jk}}^{z_k})}^{a_k}$$

Backward Propagation 기초 공식

미분법의 기본공식

① $f(x)=c$ (단, c는 상수) 이면

$$f'(x)=0$$

② $f(x)=x^n$ (단, n은 자연수) 이면

$$f'(x)=nx^{n-1}$$

③ $\{cf(x)\}'=cf'(x)$ (단, k는 상수)

④ $\{f(x)\pm g(x)\}'=f'(x)\pm g'(x)$

⑤ $\{f(x)g(x)\}'=f'(x)g(x)+f(x)g'(x)$

- Chain Rule :

$$\frac{\partial p}{\partial x_1}=\frac{\partial p}{\partial z_1}\frac{\partial z_1}{\partial x_1}+\frac{\partial p}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$z_1=z_1(x_1,x_2)$$
$$z_2=z_2(x_1,x_2)$$
$$p=p(z_1,z_2)$$

Backpropagation (Reverse-mode differentiation)

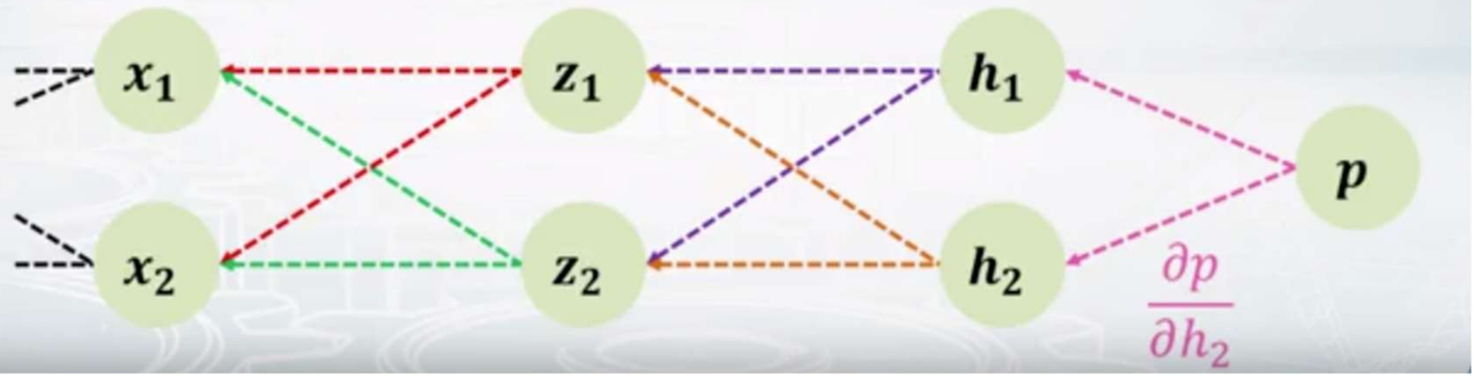
3: $\frac{\partial p}{\partial h_1}$ $\quad$ $\frac{\partial p}{\partial h_2}$ $\quad$ We will need these for GD

2: $\frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}$ $\quad$ $\frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}$

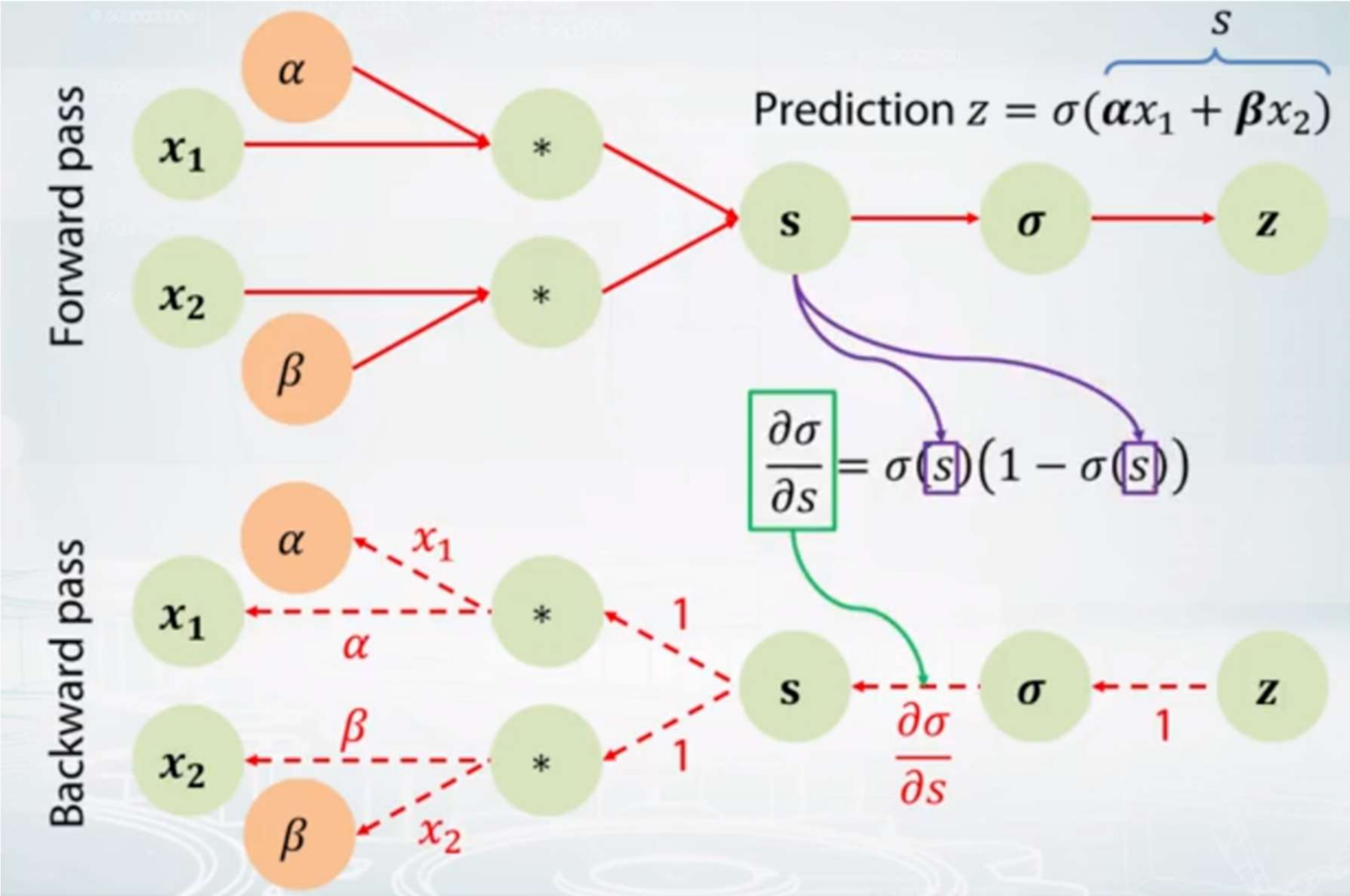
1:

$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_2}$$



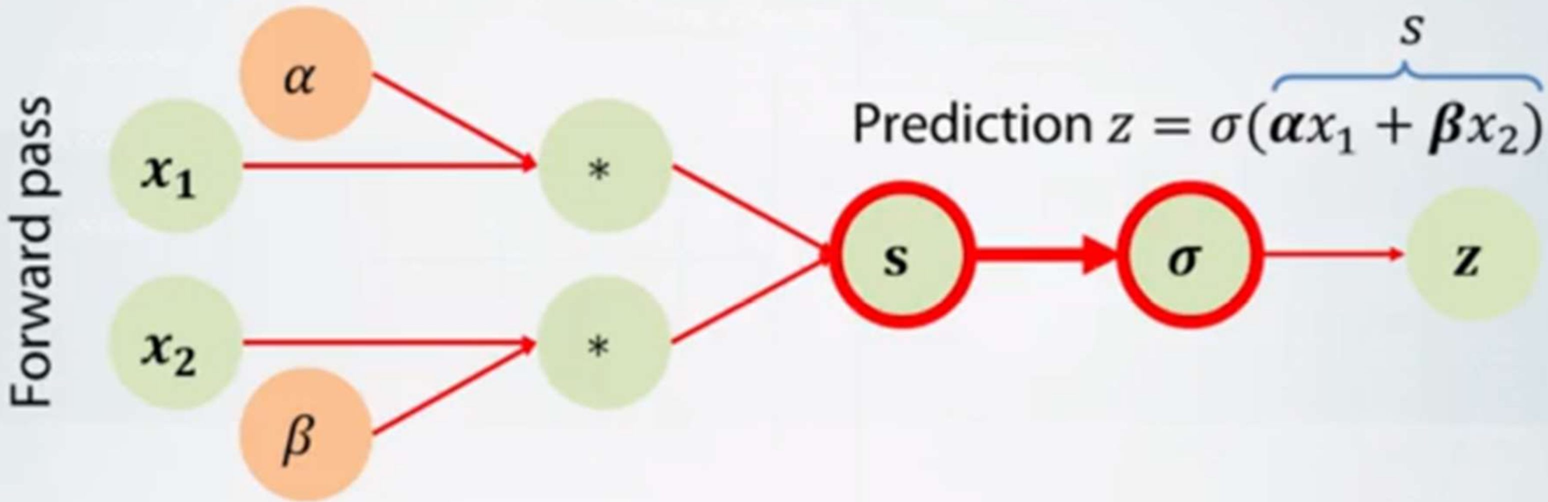
Forward & Backward Propagation



σ : sigmoid

Forward pass interface

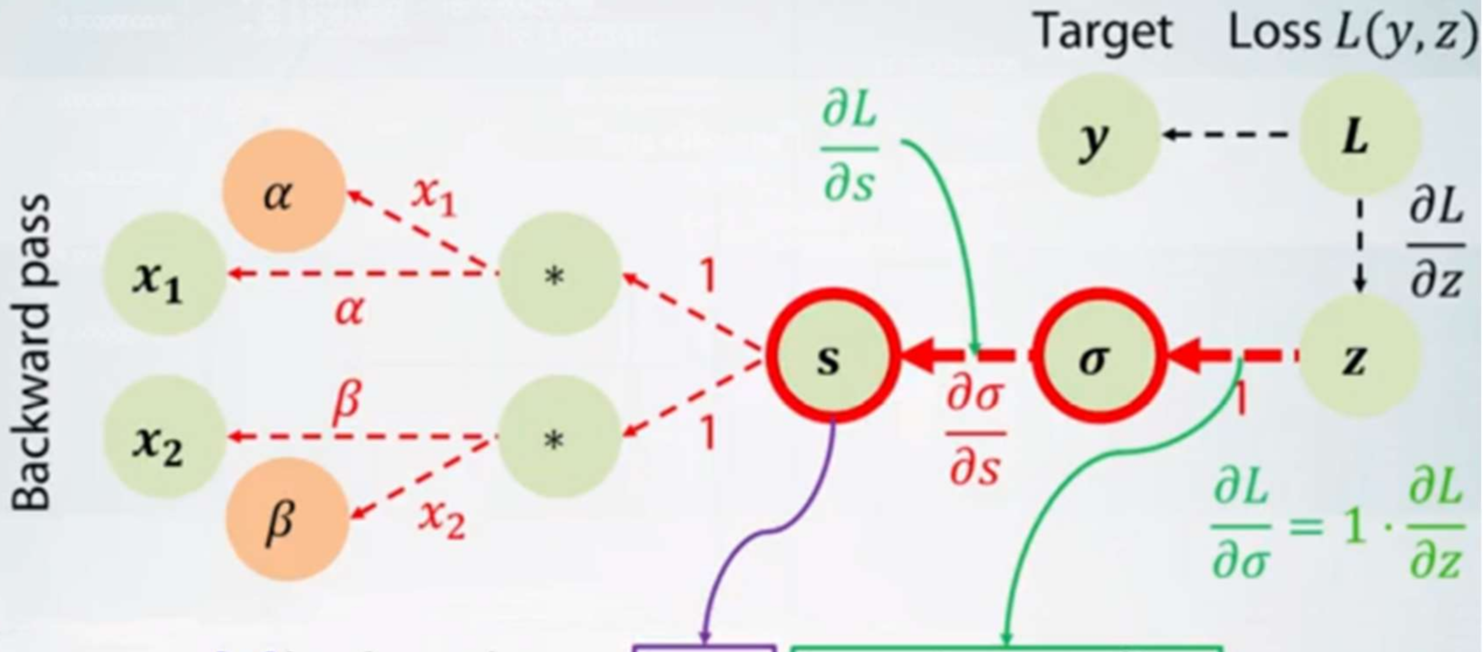
Let's implement a sigmoid activation node!



Prediction $z = \sigma(\underbrace{\boldsymbol{\alpha} x_1 + \boldsymbol{\beta} x_2}_{s})$

```
def forward_pass(inputs):
  return 1. / (1 + np.exp(-inputs))
```

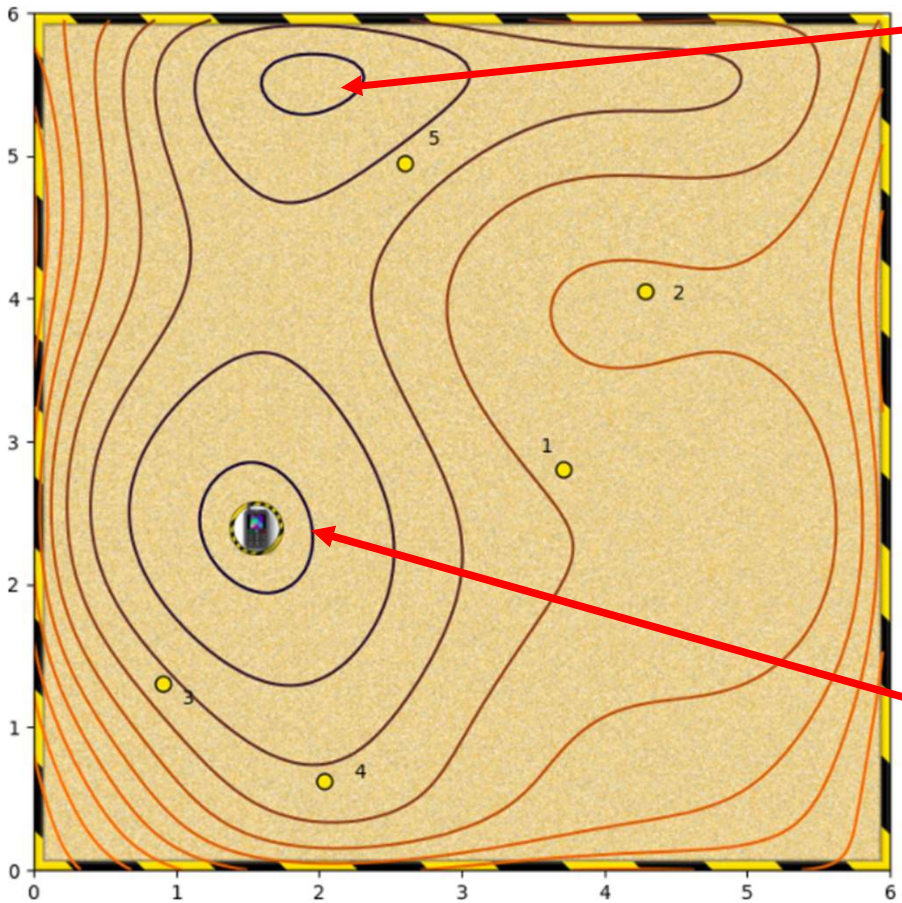
Backward pass interface




```python
def backward_pass(inputs, incoming_gradient):
    sigmoid = 1. / (1 + np.exp(-inputs))
    return sigmoid * (1 - sigmoid) * incoming_gradient
```

$$\frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \cdot \frac{\partial L}{\partial \sigma}$$

Global Minima / Local Minima



Local Minimum

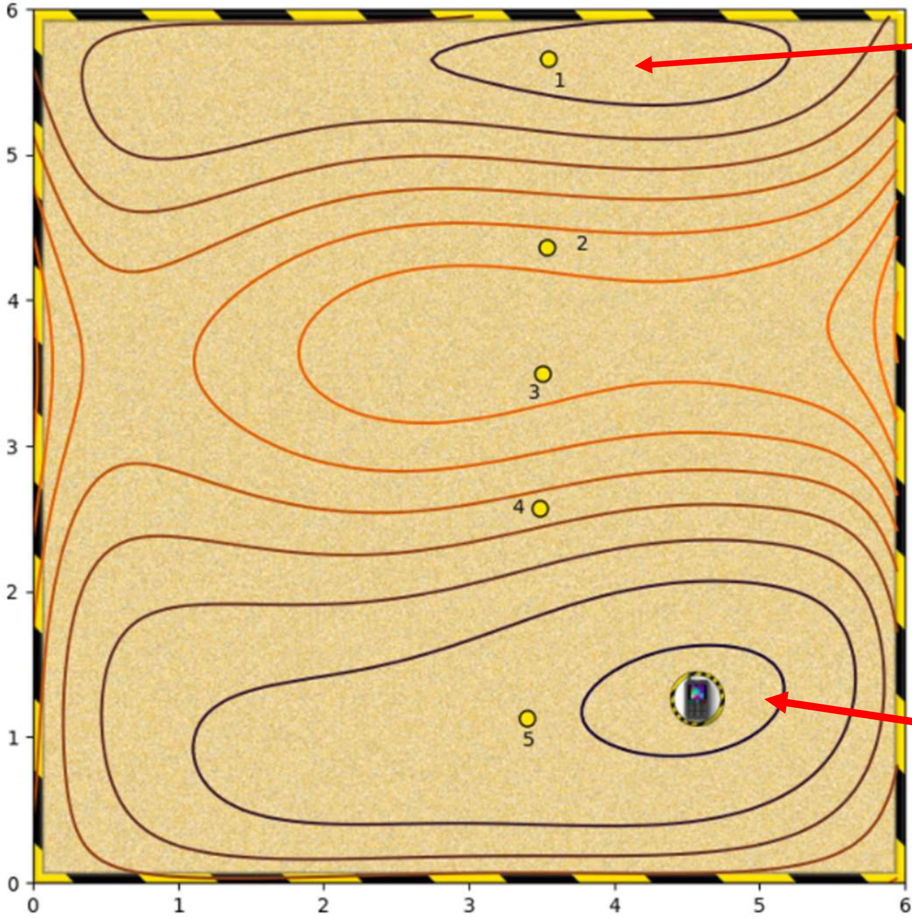
Global Minimum

단순히 가장 가파른 경사만을 따라갈 경우,

1, 3, 4 – Global Minima 도달 가능

2, 5 – Local Minima 도달 가능

Global Minima / Local Minima



Local Minimum

단순히 가장 가파른 경사만을 따라갈 경우,

3, 4, 5 – Global Minima 도달 가능

1, 2 – Local Minima 도달 가능

Global Minimum

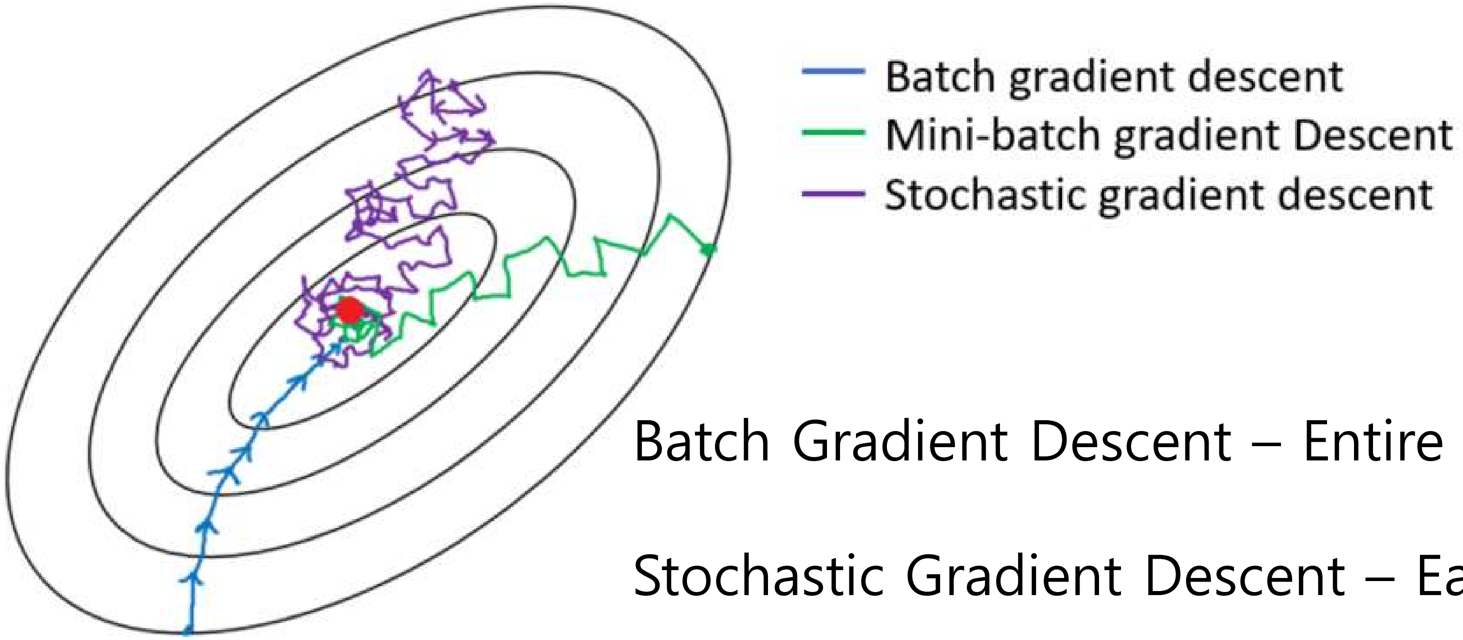
Optimizers

- Stochastic Gradient Descent Optimizer
- RMSProp Optimzer
- Adagrad Optimizer
- Adam Optimizer, etc

http://ruder.io/content/images/2016/09/contours_evaluation_optimizers.gif

http://ruder.io/content/images/2016/09/saddle_point_evaluation_optimizers.gif

Stochastic Gradient Descent (확률적 경사하강법)

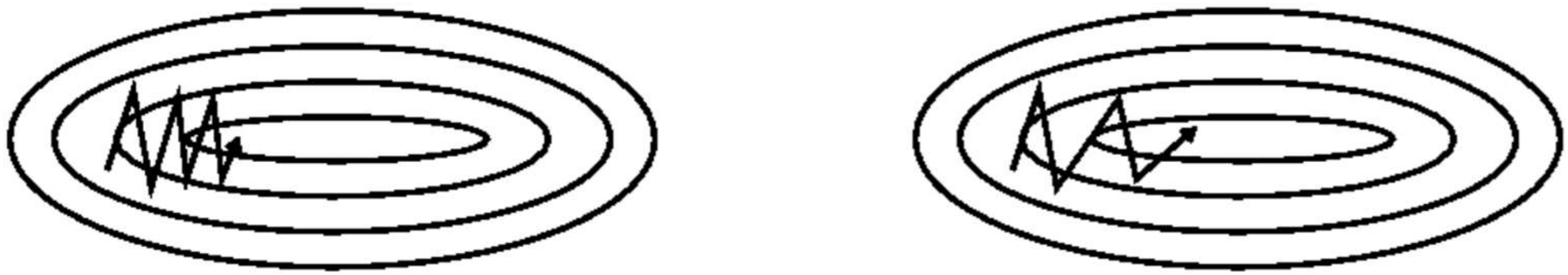


Batch Gradient Descent – Entire samples

Stochastic Gradient Descent – Each sample (size 1)

Mini-batch Gradient Descent – small size of samples

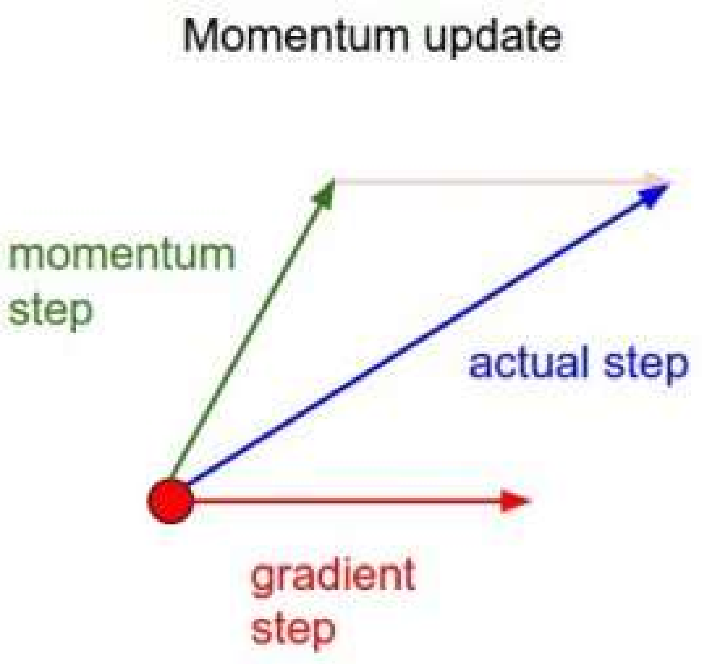
Momentum : 방향성을 유지하며 가속



(a) SGD without momentum (b) SGD with momentum

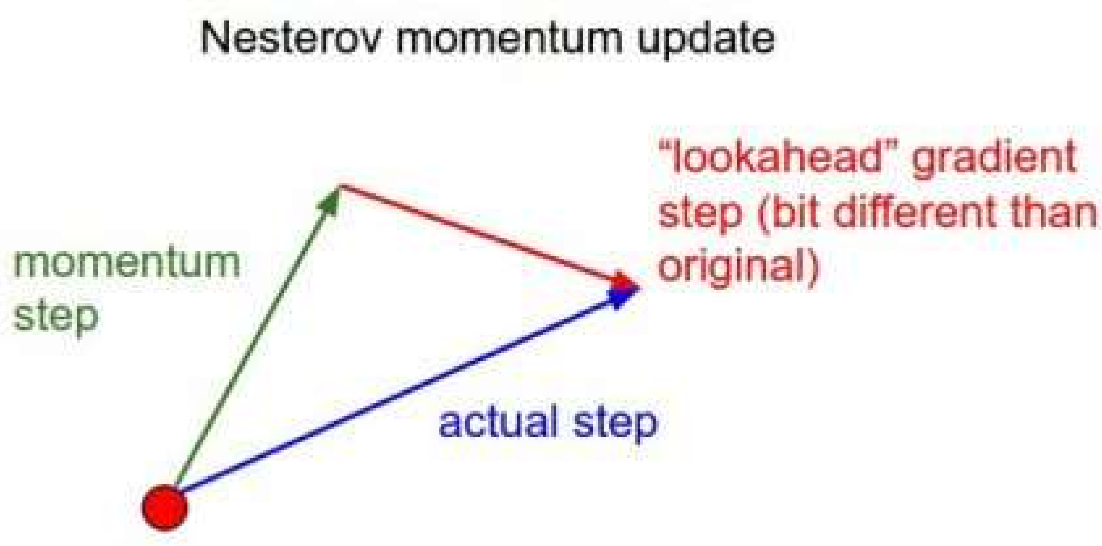
Global minimum 에 빨리 도달하기 위해 vertically 는 변화가 적고 horizontally 는 변화가 크도록 parameter 조절

Momentum



$$v_{t+1} = \mu v_t - \epsilon g(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Nesterov Accelerated Gradient (NAG)

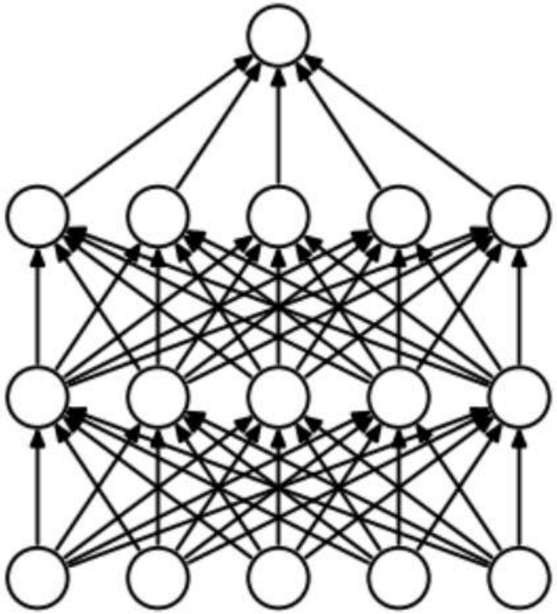


$$v_{t+1} = \mu v_t - \epsilon g(\theta_t + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

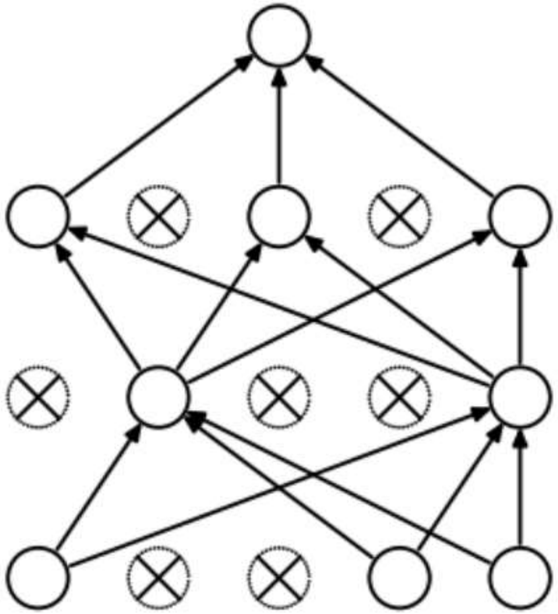
v_t : 누적모멘트
θ_t : 현재위치
ϵ : 학습속도
μ : 가중치, g : gradient

Dropout regularization

Random 한 drop out 을 통한 과적합 방지 (특정 feature 의존 방지)



(a) Standard Neural Net



(b) After applying dropout.

epoch

- 정의 – 전체 dataset 이 neural network 을 통해 한번 처리된 것
- Epoch 은 model 의 training 시에 hyperparameter 로 횟수 지정
- 하나의 epoch 은 한번에 처리하기 어려운 size 이므로 여러 개의 batch 로 나누어 처리
- Parameter training 을 위해서는 여러 번 epoch 을 반복해야 한다.
- One epoch 내에서의 iteration 횟수는 total sample size / batch size
- Ex) 1 epoch = 4 iterations = 2000 training example / 500 batches

Hyper-parameters

- α - Learning Rate
- β - momentum term
- # of layers
- Dropout rate
- # of epochs
- Batch size

Network Layer 와 Neuron 의 개수는 어떻게 결정하는가 ?

- **정해진 rule 이 없음** : Empirical Try and See

  → Too few : 과소적합, Too many : 과대적합

- Input 및 output node 고려
- Training data 의 volume 고려
- Function 의 복잡도 고려
- Training algorithm 고려

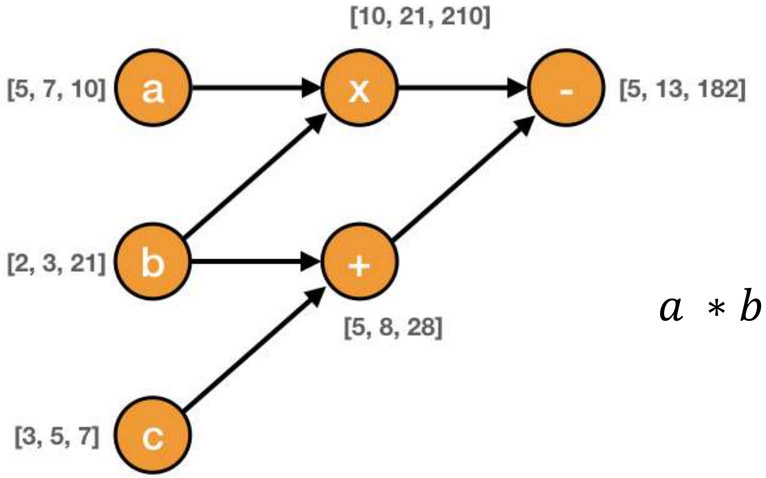
Open Source Libraries for Deep Learning

- Scikit-Learn – 2007, Python Library based on Matplotlib, NumPy, SciPy
- Theano - 2007, Open Source Python Library
- Tensorflow – 2015, Google. Open Source Machine Learning Framework
- Keras – 2015, Open Source Python Library
  (working on top of Tensorflow, Theano, CNTK)
- Microsoft Cognitive Tool – 2016, CNTK
- Caffe – 2017, Berkeley AI Research
- Pytorch – 2016, Facebook
- H2O – 2011, Open Source Big Data platform on Apache Hadoop

What is Tensorflow ?

- 기계 학습과 딥러닝을 위해 구글에서 만든 오픈소스 라이브러리

- 데이터 플로우 그래프(Data Flow Graph) 방식을 사용
  → 장점 : 계산순서의 최적화 (ex. 병렬처리)



$$a * b - (b + c)$$

What is Tensorflow ?

- Tensorflow 의 기본 recipe
  Operation – 연산
  tensor – node 간의 data 흐름
  session – graph 를 수행시키기 위한 환경
  variable – training 시킬 parameters

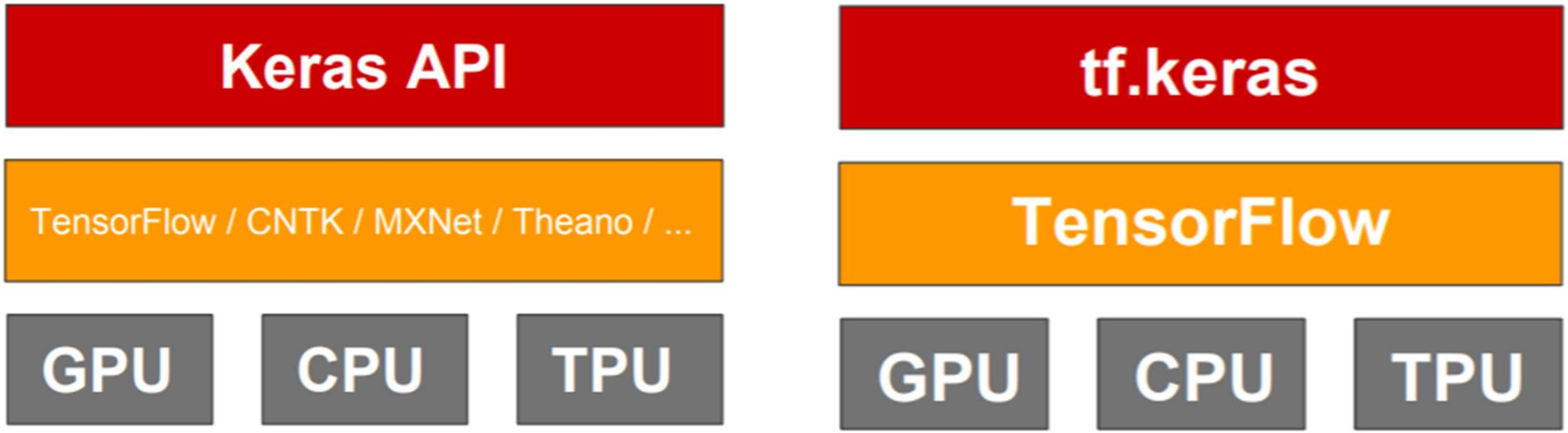
- Tensorflow is a low-level computational library → not easy

- Parallel Training of Large Scale System
- GPU support

What is Keras ?

- User friendly API
- Module 조립 방식의 model 구성
- 유연한 확장성
- 별도의 configuration 불필요 및 쉬운 debugging
- Tensorflow, Theano, CNTK backend
- 별도 설치 혹은 Tensorflow 내의 통합 module 사용

Keras 소개


Keras API
TensorFlow / CNTK / MXNet / Theano / ...
GPU
CPU
TPU
tf.keras
TensorFlow
GPU
CPU
TPU

Keras Sequential Model 의 구성

- layers – Dense, Activation, Dropout, Flatten
- compile – optimizer, loss, metrics
- fit – batch_size, epoch
- evaluate
- predict

Keras API - 1

Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Keras API - 2

Functional API

복잡한 model
ex) multiple output, shared layers, etc

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Keras API - 3

Model sub-classing

Customized model

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

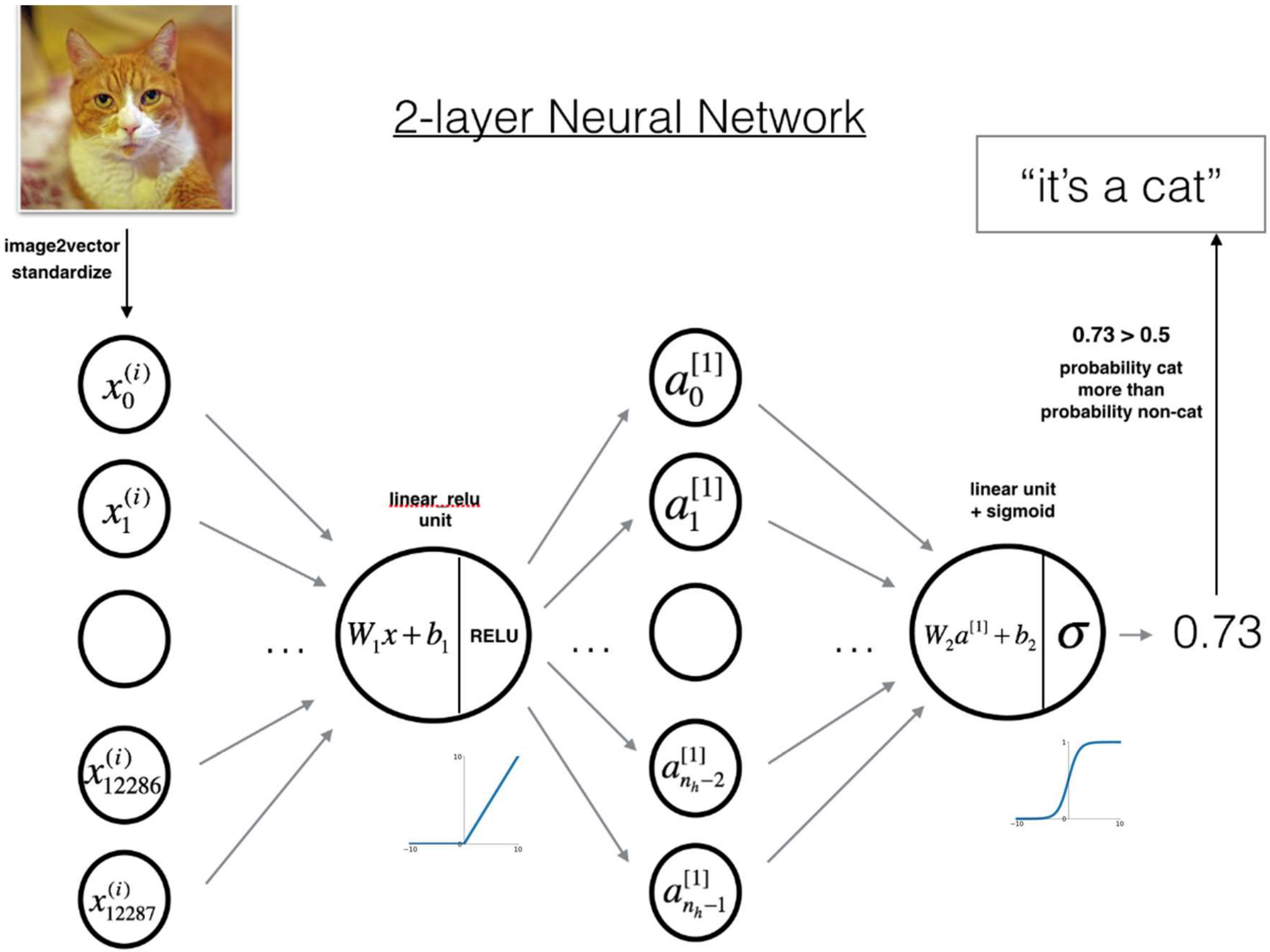
Deep Learning

Models

Basic Neural Network

실습 : Simple Logistic Regression

1. Hidden Layer 없는 Simple Logistic Regression Neural Network ➔ Shallow Neural Network
2. 입력된 image 가 고양이인지 여부 판별
3. Image 자료 시각화
4. Neural Network Model 구성 및 Compile
5. Model Train



2-layer Neural Network
"it's a cat"
image2vector
standardize
0.73 > 0.5
probability cat
more than
probability non-cat
$x_0^{(i)}$
$x_1^{(i)}$
$x_{12286}^{(i)}$
$x_{12287}^{(i)}$
linear_relu
unit
W_1x+b_1
RELU
$a_0^{[1]}$
$a_1^{[1]}$
$a_{n_h-2}^{[1]}$
$a_{n_h-1}^{[1]}$
linear unit
+ sigmoid
$W_2a^{[1]}+b_2$
σ
0.73

실습 : Boston 주택가격 Regreesion

1. Boston House Price Dataset
   - sklearn.datasets.load_boston 이용

2. 보스턴 시의 주택 가격에 대한 데이터
   - 주택의 여러가진 요건들과 주택의 가격 정보가 포함.
   - 주택의 가격에 영향을 미치는 요소를 이용하여 회귀분석

3. 13 개의 종속변수와 1 개의 독립변수 (주택가격 중앙값) 으로 구성

- Feature 설명

CRIM 자치시(town) 별 1인당 범죄율,

ZN 25,000 평방피트를 초과하는 거주지역의 비율

INDUS 비소매상업지역이 점유하고 있는 토지의 비율

CHAS 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)

NOX 10ppm 당 농축 일산화질소

RM 주택 1가구당 평균 방의 개수

AGE 1940년 이전에 건축된 소유주택의 비율

DIS 5개의 보스턴 직업센터까지의 접근성 지수

RAD 방사형 도로까지의 접근성 지수

TAX 10,000 달러 당 재산세율

PTRATIO 자치시(town)별 학생/교사 비율

B 1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함

LSTAT 모집단의 하위계층의 비율(%)

MEDV 본인 소유의 주택가격(중앙값) (단위: $1,000)

Deep Neural Network

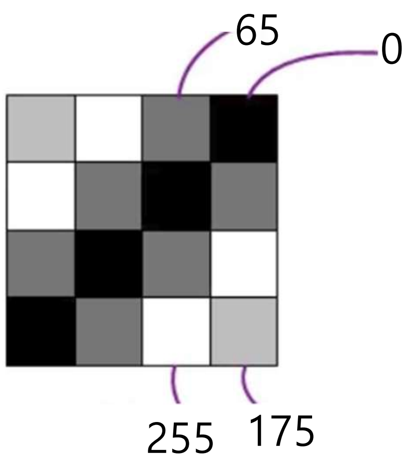
Mnist Dataset 소개

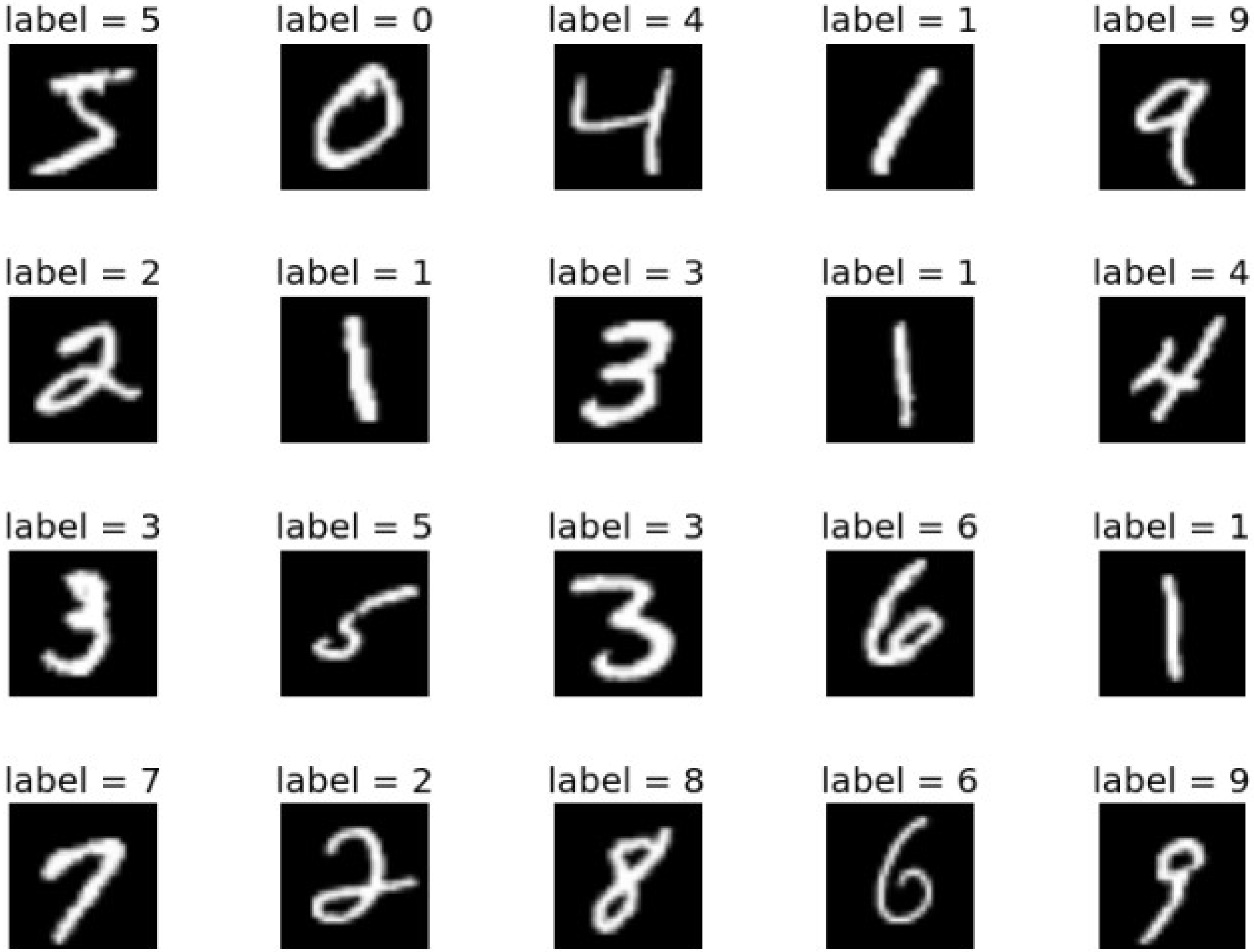
- National Institute of Standards and Technology

- 0 ~ 9 의 10 개 숫자 손글씨 image dataset
- 28 x 28 pixel 의 gray scale image
- 각 image 마다 0 to 9 의 label 로 쌍을 이루고 있음
- Train set 60,000 / Test set 10,000
- Machine Learning 의 Hello World 에 해당

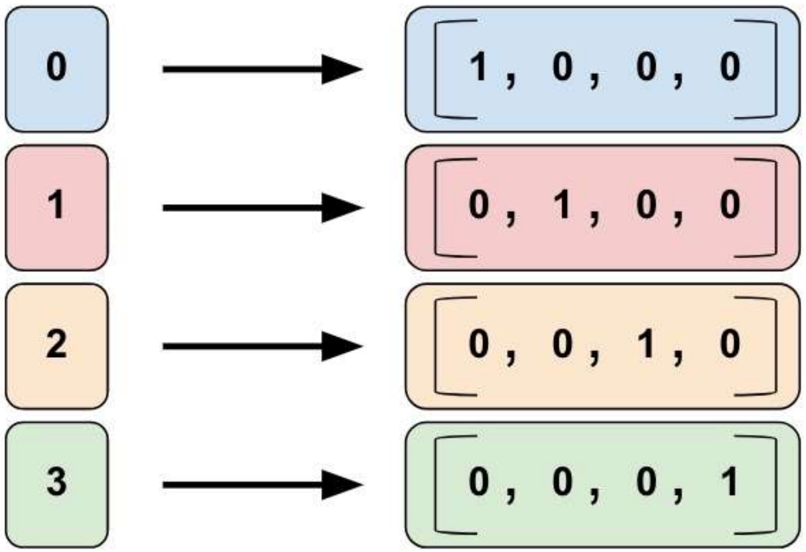
Pixel 의 구성

0 – black
255 - white





One-Hot encoding



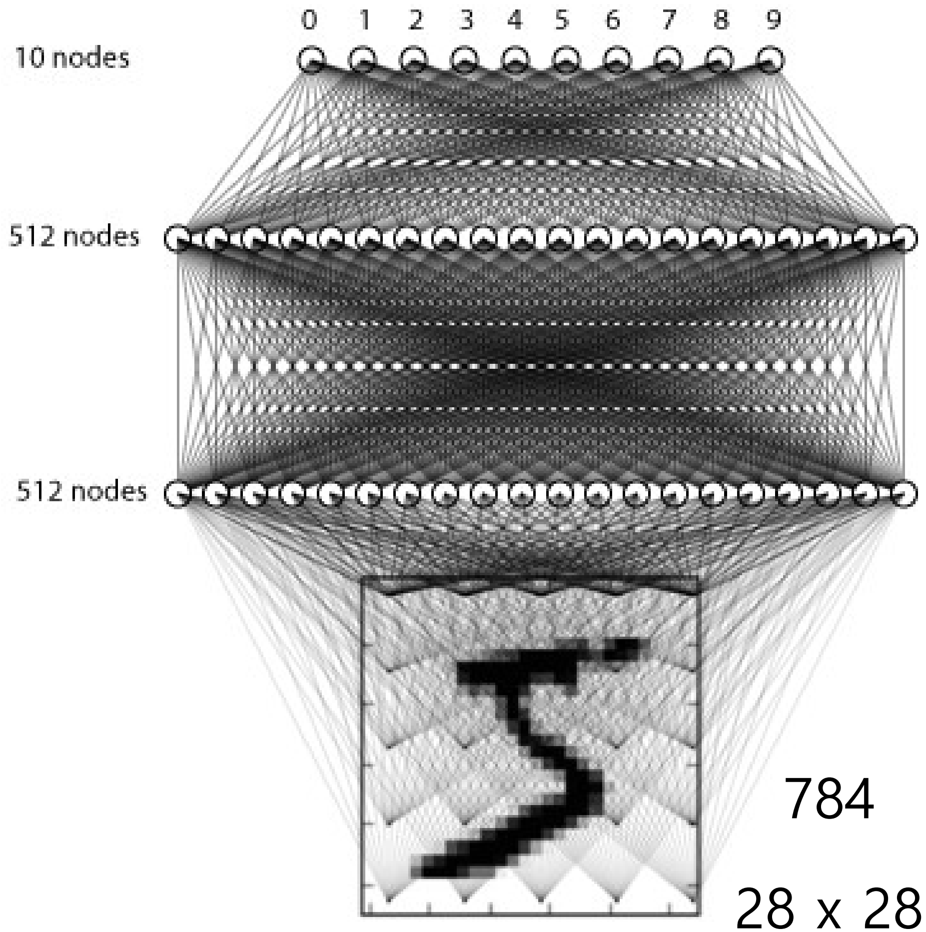
color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0

Output Layer - softmax

Output Layer

2^{nd} Hidden Layer

1^{st} Hidden Layer



784

28 x 28

실습 : Mnist set 을 이용한 손글씨 인식

1. 2-Layer 이상의 Fully Connected(Dense) Neural Network
2. Input reshaping and scaling
3. One-hot encoding
4. Neural Network Model 구성 및 Compile
5. Model Train
6. Performance Evaluation

실습 : Hyper-parameter Tunning 을 이용한 손글씨 인식 성능 개선

	Model 1	Model 2	Model 3	Model 4	Model 5
# of Hidden Layers	0	2	2	2	3
# of Hidden neurons	128	128	128	512	?+?+?
# of epochs	10	10	10	10	10/15/20
Dropout	0	0	0.2	0.2	0.2/0.3
Batch size	128	128	512	512	256/512
accuracy	92.6	97.4	98.01	98.06	98.40

➡ Hyperparameter tuning 을 통해 98.4 % 의 정확도 달성

➡ **Enough ? → No ! → CNN**