

# NLP using Deep Learning

**Bok, Jong Soon**  
**javaexpert@nate.com**  
<https://github.com/swacademy>



---

# Ann & Deep Learning



# AI의 역사 및 Deep Learning의 혁신

## 인공지능의 역사 및 Deep Learning의 혁신

~1990년 : 이론 정립

~2000년 : 구현 시도

~2010년 : 본격 시도

2010년~ : 혁신의 시작  
(딥러닝 기반의 인공지능)

시대별 한계 : - 컴퓨팅의 한계로 제안된 이론 구현의 어려움 - 데이터의 한계로 현실 문제 해결하지 못함 - 알고리즘의 한계로 완성도 부족

### 혁신적 알고리즘 제안

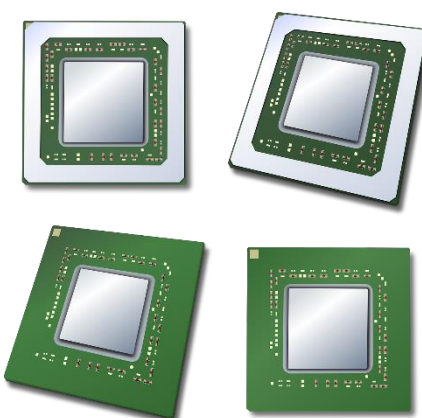


Geoffrey Hinton  
(Univ. of Toronto,  
Google)

- 혁신적 딥러닝 이론 제안(2006년)
- 실제 구현으로 혁신적 성능 증명
- 이미지 인식 대회인 ImageNet Challenge에서 압도적 성능으로 우승(2012년)

+

### 컴퓨팅 파워 발전



- CPU는 고성능화와 함께 저가격화

+

### 데이터 폭증



- 2020년 데이터의 크기는 40ZB (40조 GB) 크기

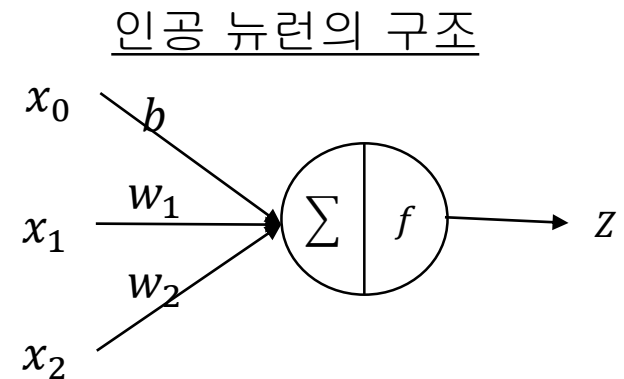
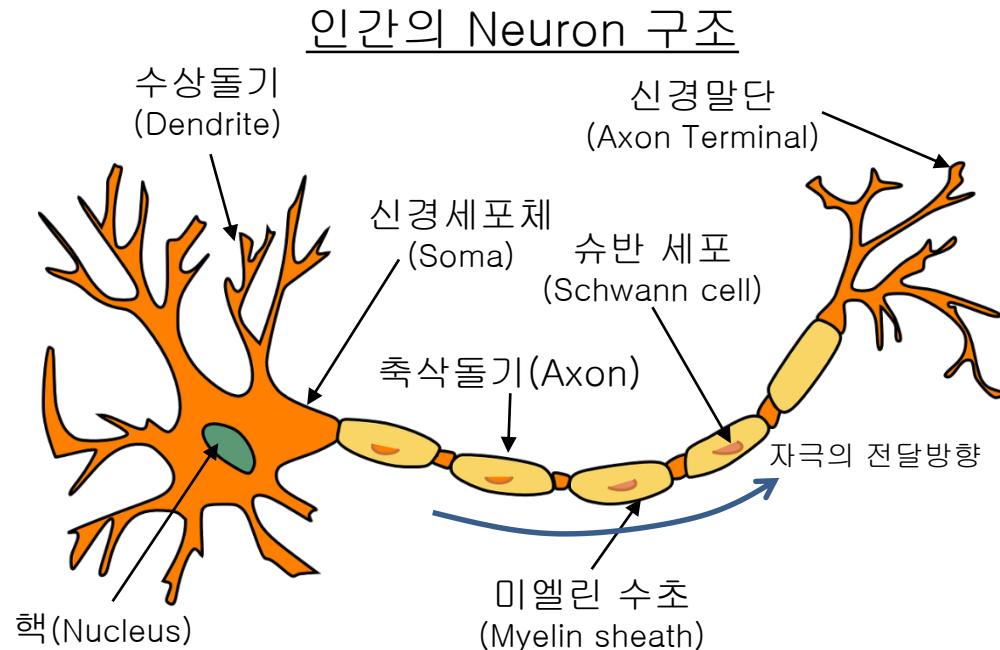
# Neuron

## ■ 인간의 뉴런(Neuron)

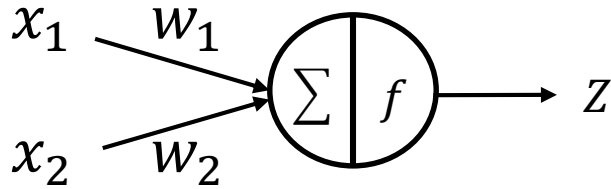
- Synapse를 통해 Neuron間 신호를 전달
- 각 Neuron은 수상돌기(dendrite)를 통해 입력 신호를 받음
- 입력 신호가 특정크기(threshold) 이상인 경우에만 활성화 되어 축삭돌기(axon)을 통해 다음 Neuron으로 전달

## ■ 인공 뉴런(노드; Node)

- 각 Node는 가중치가 있는 입력 신호를 받음
- 입력신호는 모두 더한 후, 활성화 함수(activation function)를 적용함
- 활성화 함수의 값이 특정 값 이상인 경우에만 다음 Node의 입력값으로 전달



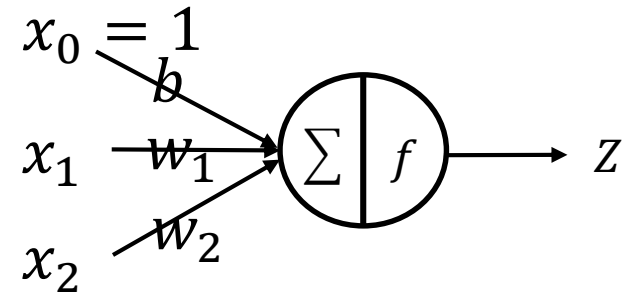
# 인공신경망의 Neuron



$$\Sigma = U = W_1X_1 + W_2X_2$$

$$Z = f(u)$$

if  $Z > b$  then 1 else 0

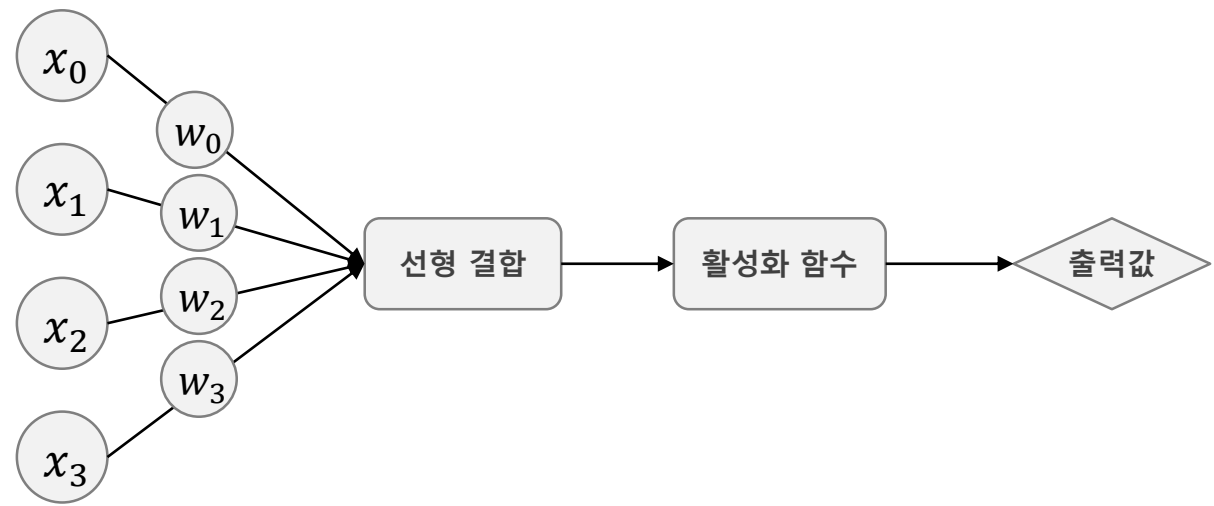
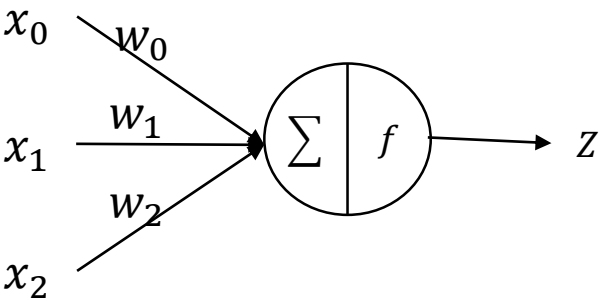
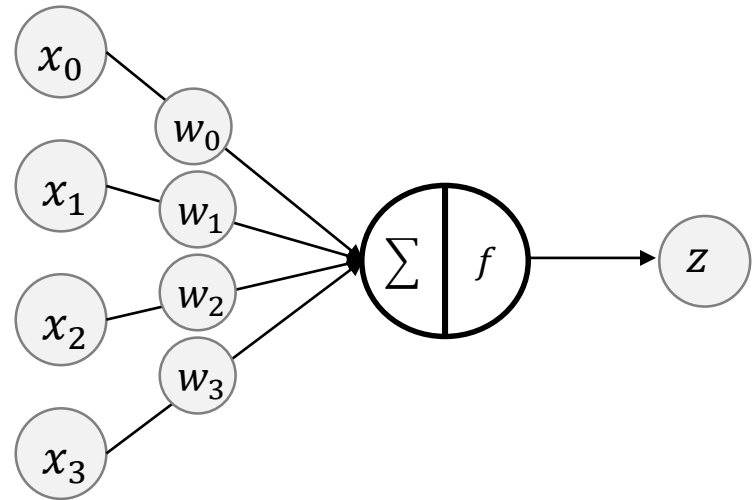


$$\Sigma = U = W_1X_1 + W_2X_2 + b$$

$$Z = f(u)$$

if  $Z > b$  then 1 else 0

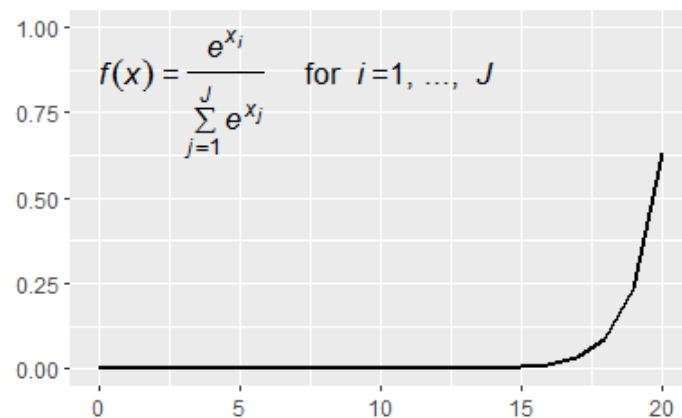
# Neuron의 표현 방법



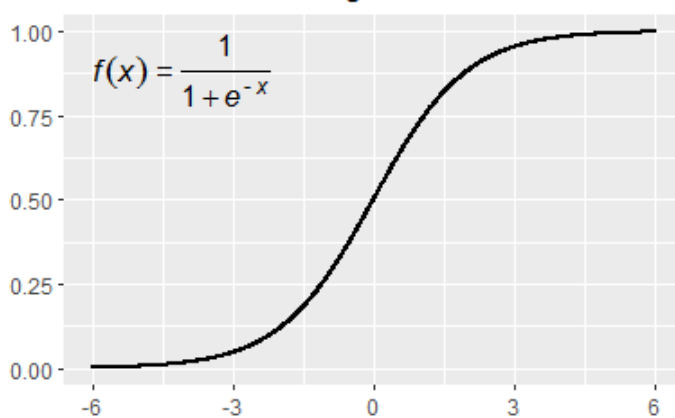
# 활성화 함수(activation function)

- 생물학적 뉴런(neuron)에서 입력 신호가 일정 크기 이상일 때만 신호를 전달하는 메커니즘을 모방한 함수

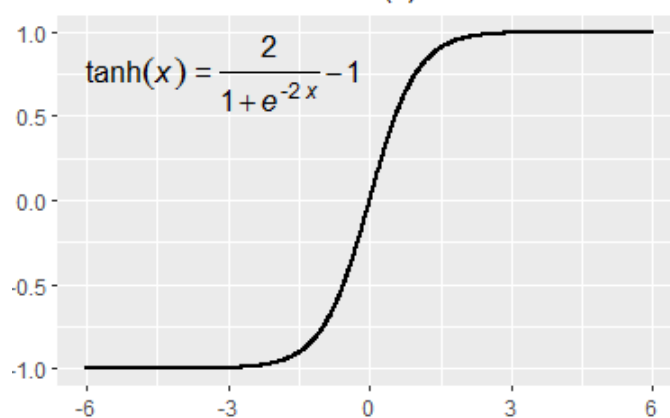
Softmax



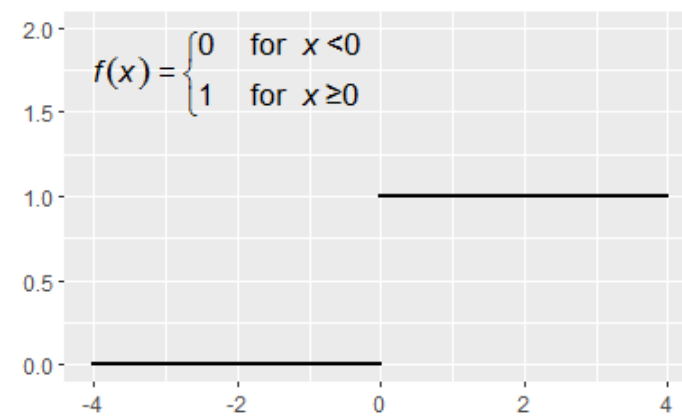
Sigmoid



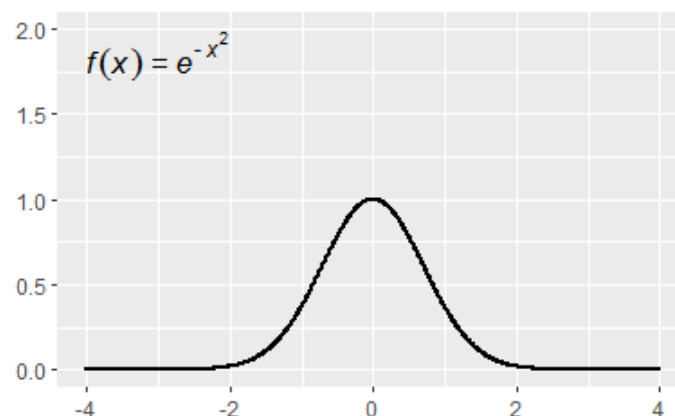
tanh(x)



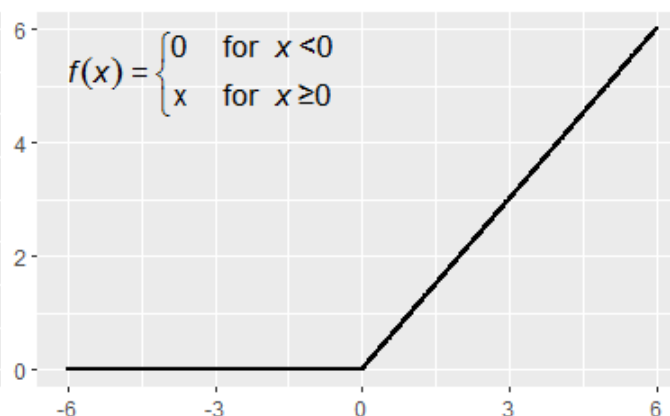
Binary step



Gaussian

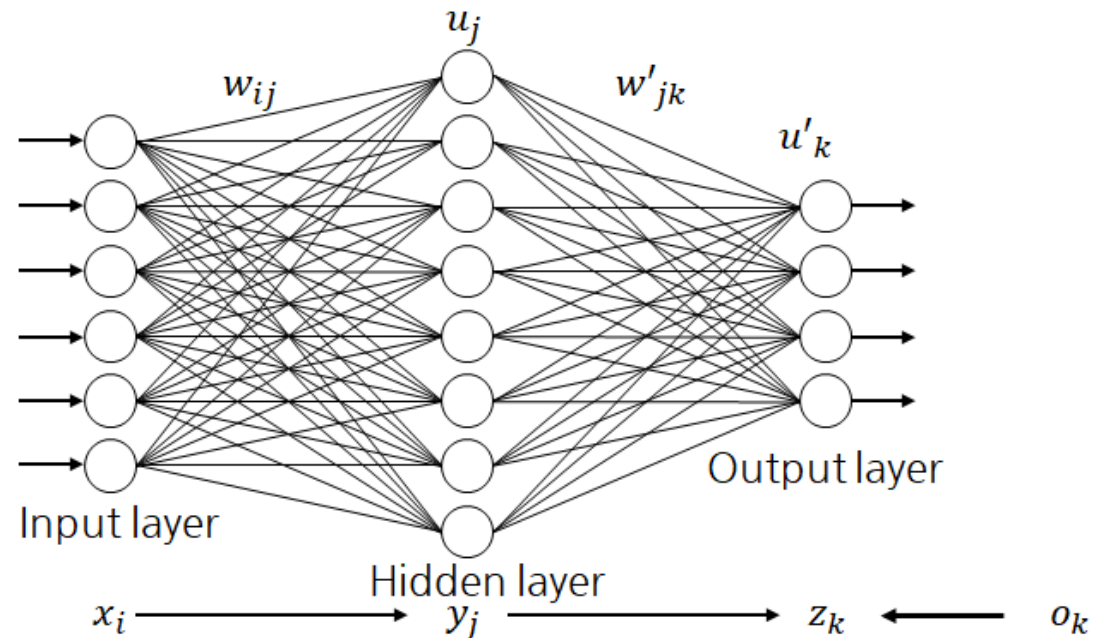


ReLU



# 인공신경망

- 인공신경망은 입력층(Input layer), 은닉층(Hidden layer), 출력층으로 구성
- 각 층의 Neuron들을 퍼셉트론(Perceptron)
  - 퍼셉트론(Perceptron)은 인공 뉴런의 한 종류.
- 입력층의 Neuron의 수는 입력 데이터의 수이며, 출력층은 분류 문제를 해결할 경우에는 분류의 수와 일치





# 인공신경망 구성요소

- 입력층(input layer)
  - 입력 값으로 구성된 Layer
  - 학습 Dataset의 "입력 변수의 개수 Node + 1"만큼의 Node로 구성
    - 0번째 원소는 Bias로 값은 항상 1을 할당
- 출력층(output layer)
  - Model의 출력값을 만들어 내는 Layer
  - 예) IRIS 품종 분류 문제
    - Multi Class 분류 문제의 경우 Softmax 함수를 출력함수로 사용
- 은닉층(hidden layer)
  - 입력층과 출력층 사이의 Layer
  - Neuron과 Synapse로 구성된 인간의 두뇌를 모방하는 Layer
  - 은닉층으로 들어오는 입력값의 합을 계산한 후 활성화 함수를 적용
  - 활성화 함수 출력이 임계치를 넘지 않을 경우 다음 노드로 0을 전달(신호를 전달하지 않음)
  - 은닉층의 개수와 은닉 노드 개수
  - 너무 적으면 입력 데이터를 제대로 표현하지 못해 모델을 제대로 학습하지 못함
  - 너무 많으면 과적합(overfitting)이 발생하며, 학습 시간도 많이 소모

# 인공신경망 종류

- 기계학습과 인지과학에서 생물학의 **신경망**(동물의 중추신경계 중 특히 뇌)에서 **영감을 얻은 통계학적 학습 Algorithm**.
- 인공신경망은 Synapse의 결합으로 Network를 형성한 인공 Neuron(Node)이 학습을 통해 Synapse의 결합 세기를 변화시켜, 문제 해결 능력을 가지는 모델 전반을 가리킨다.
- 신경망 알고리즘 종류
  - 전방 전달 신경망(Feedforward Neural Network)
    - 가장 간단한 방법의 인공신경망
    - 신경망 정보가 입력 노드에서 은닉노드를 거쳐 출력 노드까지 전달되며 순환 경로가 존재하지 않는 Graph를 형성

# 인공신경망 종류 (Cont.)

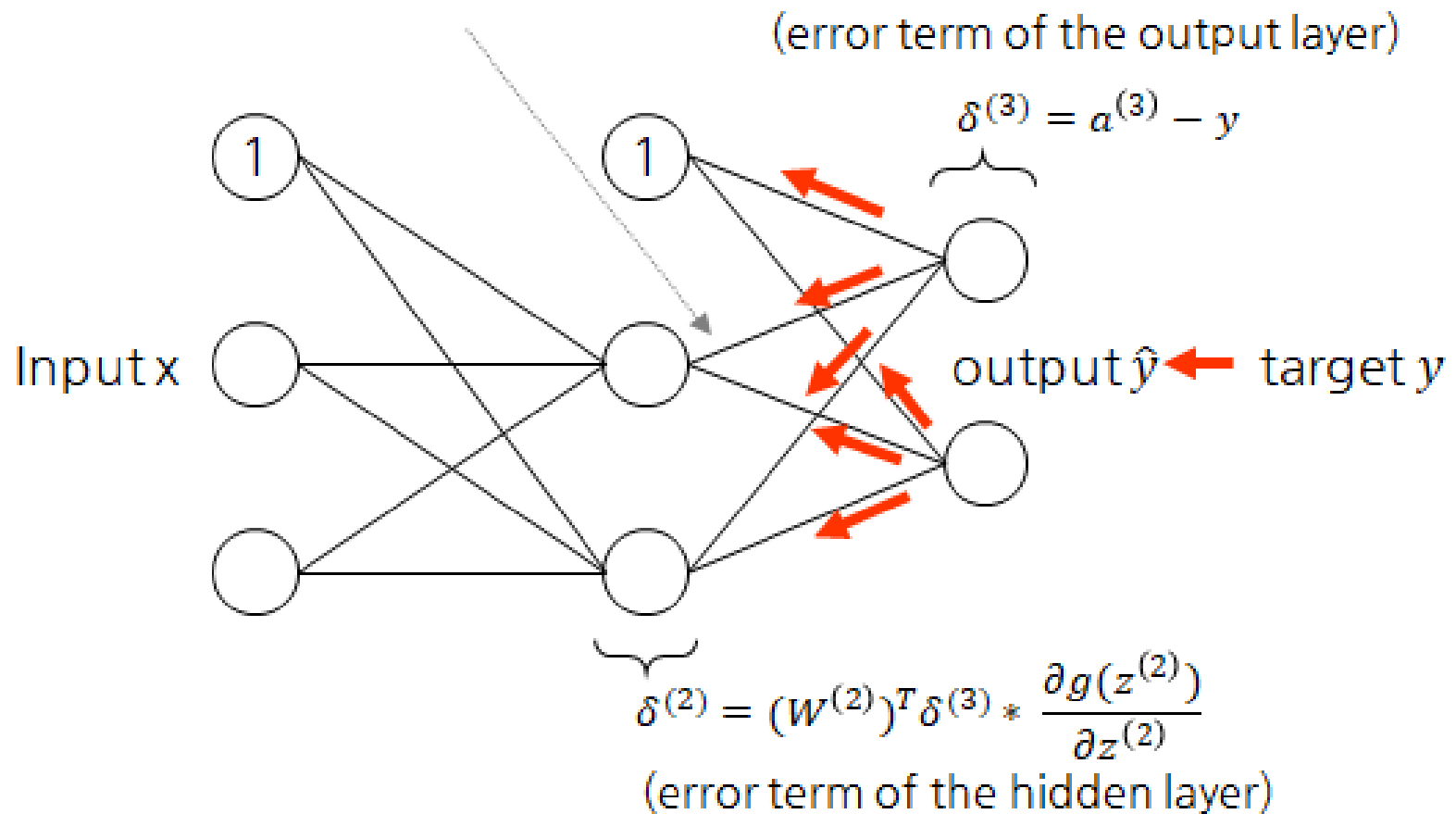
## ■ 신경망 알고리즘 종류

- 방사 신경망(Radial Basis Function Network)
  - 방사상 인공 신경망은 다차원의 공간의 보간법에 매우 강력한 능력을 가지고 있음
  - 방사 함수는 다 계층의 Sigmoid함수를 은닉 Node에서 사용하는 형태를 대체할 수 있음
- 코헨 자기조직 신경망(Kohonen Self-organizing Network)
  - 자율(unsupervised) 학습방법과 경쟁(competitive) 학습방법을 사용
- 순환 인공 신경망(Recurrent Neural Network)
  - 순환 인공 신경망은 전방 신경망과 정 반대의 동작을 함
  - 노드들 간에 양방향으로 데이터가 이동하며 이 데이터는 선형적으로 전달이 됨
  - 데이터가 후방 노드에서 전방노드로 전달하여 연산이 수행됨

# 역전파 Algorithm 수행과정

$$\frac{\partial}{\partial w_{ij}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$$

(compute gradient)



# 오류역전파 Algorithm 수행 과정

1. 전방향 연산을 수행해서 은닉층1, 은닉층2를 통해 최종적으로 출력층까지 Node들의 활성화 함수를 계산
2. 출력층의 각각의 노드  $i$ 에 대해서 에러값을 계산

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. 출력층을 제외한 각각의 Node  $i$ 에 대해서 에러값을 계산

$$\delta_i^{(n_l)} = (\sum_{j=1}^{s_j+1} W_{ji}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(n_l)})$$

4. 신경망의 Parameter  $W$ 와  $b$ 에 대한 미분값을 계산

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_{ij}^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$

# 오류역전파 Algorithm 수행 과정

- $f'(z_i^{(n_l)})$  는 활성화 함수의 미분값, 활성화 함수가 Sigmoid 함수인 경우 다음과 같음

$$f'(z_i^{(n_l)}) = f(z_i^{(n_l)}) \cdot (1 - f(z_i^{(n_l)}))$$

- 오류역전파 Algorithm을 이용해 구현한 W(weight)와 b(bias)의 미분값을 이용해서 W와 b를 업데이트 함

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(j)}} J(W, b; x, y) = W_{ij}^{(l)} - \alpha \cdot a_j^{(l)} \delta_i^{(l+1)}$$

$$b_{ij}^{(l)} = b_{ij}^{(l)} - \alpha \frac{\partial}{\partial b_{ij}^{(j)}} J(W, b; x, y) = b_{ij}^{(l)} - \alpha \cdot \delta_i^{(l+1)}$$

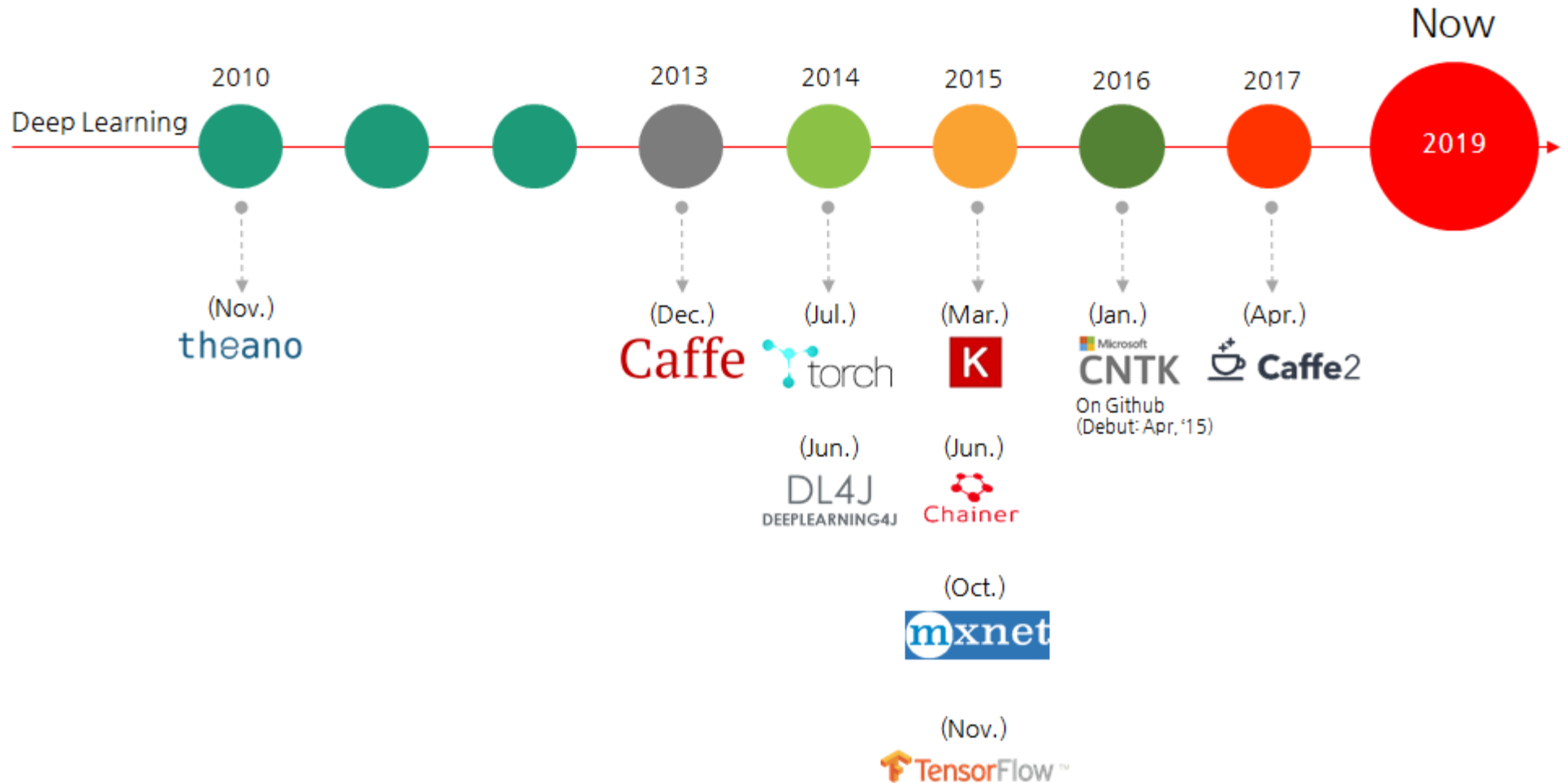


---

# Deep Learning Framework



# Deep Learning Framework Timeline





# Deep Learning Framework 종류

## ■ TensorFlow

- Google Brain Team에서 개발했으며 2015년 OpenSource로 공개
- Python 기반 Library
- 여러 CPU 및 GPU와 모든 Platform, Desktop 및 Mobile에서 사용할 수 있음
- Python, C++, Java, Go 등 언어를 지원하며 Deep Learning을 직접 작성하거나 Keras와 같은 Wrapper Library를 사용하여 직접 작성할 수 있음

## ■ Theano

- 최초의 Deep Learning Library 중 하나
- Python 기반이며 CPU 및 GPU의 수치 계산에 매우 유용
- 저수준 Library로, Deep Learning Model을 직접 만들거나 그 위에 Wrapper Library를 사용하여 Process를 단순화 할 수 있음

# Deep Learning Framework 종류 (Cont.)

## ■ Keras

- 효율적인 인공신경망 구축을 위해 단순화된 Interface로 개발되었음
- TensorFlow 또는 Theano에서 작동하도록 구성 할 수 있음
- Python으로 작성되었으며 가볍고 배우기 쉬움

## ■ Caffe

- 속도 및 모듈성을 염두에 두고 개발되었음
- Berkeley Vision and Learning Center (BVLC)에서 개발한 최초의 Deep Learning Library
- Python Interface를 가지고 있는 C++ Library
- Caffe Model Zoo에서 미리 훈련된 여러 Network를 바로 사용할 수 있음
- 2017년에 Facebook은 최근 고성능 개방형 학습 모델을 구축 할 수 있는 유연성을 제공하는 새로운 가벼운 모듈식 Deep Learning Framework인 Caffe2를 공개했음

# Deep Learning Framework 종류 (Cont.)

## ■ Torch

- Lua 기반의 Deep Learning Framework
- GPU 처리를 위해 C/C++ Library와 CUDA를 사용
- 최대한의 유연성을 달성하고 모델을 제작하는 과정을 매우 간단하게 만드는 것을 목표로 만들어졌음
- PyTorch : Torch의 Python 구현

## ■ Deeplearning4j(DL4J)

- Java로 개발된 Deep Learning Framework
- 상업, 산업 중심의 분산 Deep Learning Platform으로 널리 사용
- Java와 Scala를 위해 작성된 세계 최초의 상용 수준 OpenSource Deep Learning Library
- 상용 서비스를 위해 설계되었고 Hadoop 및 Spark와 통합해 사용할 수 있음

# Deep Learning Framework 종류 (Cont.)

## ■ MXNet

- R, Python, C++ 및 Julia 언어를 지원하는 Deep Learning Framework 중 하나
- Back-end는 C++과 CUDA로 작성되었으며 Theano처럼 자체 Memory를 관리 할 수 있음
- 확장성이 좋고 다중 GPU와 Computer로 작업 할 수 있음
- Amazon은 MXNet을 Deep Learning을 위한 참조 Library로 사용하고 있음

## ■ Microsoft Cognitive Toolkit (CNTK)

- CNTK라는 약어로 알려져 있는 Microsoft Cognitive Toolkit은 Deep Learning Model을 교육하기 위한 OpenSource Deep Learning Tool
- Python과 C++와 언어를 지원
- 높은 확장성과 성능을 발휘하도록 설계되었으며 여러 시스템에서 실행될 때 Theano 및 TensorFlow 같은 다른 Toolkit과 비교할 때 높은 성능을 제공

# Deep Learning Framework 종류 (Cont.)

## ■ Lasagne

- Theano의 최상위에서 실행되는 고급 학습 Library
- Theano의 복잡성을 추상화하고 신경망을 구축하고 훈련시키는 데 보다 친숙한 Interface를 제공하기 위해 개발되었음
- Keras와 많은 공통점이 있음

## ■ BigDL

- Apache Spark에 대한 Deep Learning Library로 배포
- BigDL의 도움을 받아 Hadoop Cluster와 Spark에서 Spark Program으로 직접 작성하여 Deep Learning Application을 직접 실행할 수 있음
- 풍부한 학습 지원을 제공하며 Intel의 수학 커널 라이브러리(MKL)를 사용하여 고성능을 보장
- BigDL을 사용하여 사전 훈련된 Torch 또는 Caffe Model을 Spark에 로드할 수도 있음
- Cluster에 저장된 대규모 Dataset에 Deep Learning 기능을 사용하려는 경우 매우 유용한 Library



---

# TensorFlow



# TensorFlow

- Machine Learning과 Deep Learning을 위해 Google에서 만든 OpenSource Library
- 공식 사이트
  - <https://www.tensorflow.org/>
  - Data Flow Graph를 이용하여 수치 계산을 위한 Open Source Software Library

# TensorFlow Graph

TensorBoard

EVENTS IMAGES GRAPH HISTOGRAMS

Fit to screen

Run `cifar-train`

Upload Choose File

Color Structure

color: same substructure  
gray: unique

Graph (\* = expandible)

Namespace\*

OpNode

Unconnected series\*

Connected series\*

Constant


Summary

Dataflow edge

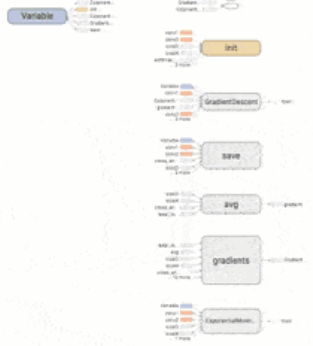
Control dependency edge


Reference edge

Main Graph



Auxiliary nodes





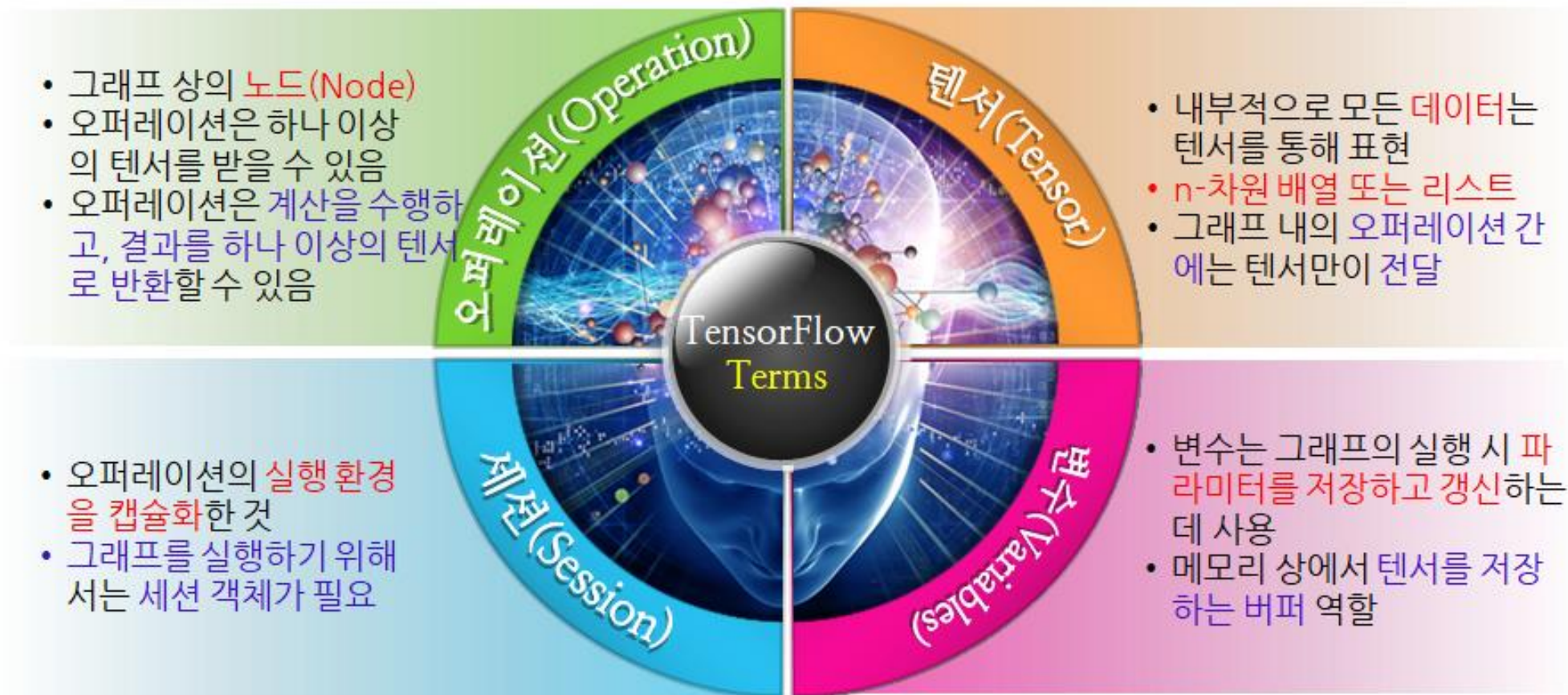
Data Flow Graph : 수학 계산과 데이터의 흐름을 **노드(Node)**와 **엣지(Edge)**를 사용한 **방향 그래프(Directed Graph)**로 표현

**특징**

- Data Flow Graph 방식
- CPU/GPU 모드로 동작
- Idea Test에서 서비스 단계까지 이용 가능
- 계산 구조와 목표 함수만 정의하면 자동으로 미분 계산을 처리
- Python/C++/Java/Go를 지원



# TensorFlow Term



Node에 연산(계산, operation)을 담고, Edge(간선)에 Data를 담고 있다.  
Edge는 Node간에 전달되는 다차원 데이터 배열(Tensor)이다.

# Programming Language and Deep Learning Framework

- Matlab
  - Mathworks사에서 개발한 과학계산용 프로그래밍 언어
- R
  - 뉴질랜드 오클랜드 대학교에서 개발한 통계 및 그래프용 프로그래밍 언어
- Python
  - 범용 인터프리터형 프로그래밍 언어
  - Numpy, Scipy 등 과학계산 및 머신러닝을 위한 패키지가 발전됨
- C/C++
  - 범용 컴파일형 프로그래밍 언어
  - 오랜 역사를 바탕으로 다방면의 라이브러리 포함
- Java
  - 엔터프라이즈 시스템 등 안정성이 중요한 시스템 개발에 사용되며 가장 많은 사용자를 확보
  - 과학계산용으로는 적합하지 않음
- Lua/Go/Scala
  - 최신의 인터프리터형 프로그래밍 언어
  - 쉬운 구문이 장점

✓ numpy(넘파이)는 C 언어에 있는 배열과 같은 형태로 움직이는 다차원 배열을 기반으로 하는 모듈  
✓ 빅데이터, 머신러닝, 과학산술 등의 수치연산이 될 요한 모든 경우에 최적의 성능을 보장해 줌  
✓ 텐서플로우는 내부적으로 numpy를 사용함

# AlexNet을 이용한 Framework 속도 비교

AlexNet (One Weird Trick paper) - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
CuDNN[R4]-fp16 (Torch)	<a href="#">cudnn.SpatialConvolution</a>	71	25	46
Nervana-neon-fp16	<a href="#">ConvLayer</a>	78	25	52
CuDNN[R4]-fp32 (Torch)	<a href="#">cudnn.SpatialConvolution</a>	81	27	53
TensorFlow	<a href="#">conv2d</a>	81	26	55
Nervana-neon-fp32	<a href="#">ConvLayer</a>	87	28	58
fbfft (Torch)	<a href="#">fbnn.SpatialConvolution</a>	104	31	72
Chainer	<a href="#">Convolution2D</a>	177	40	136
cudaconvnet2*	<a href="#">ConvLayer</a>	177	42	135
CuDNN[R2] *	<a href="#">cudnn.SpatialConvolution</a>	231	70	161
Caffe (native)	<a href="#">ConvolutionLayer</a>	324	121	203
Torch-7 (native)	<a href="#">SpatialConvolutionMM</a>	342	132	210
CL-nn (Torch)	<a href="#">SpatialConvolutionMM</a>	963	388	574
Caffe-CLGreenTea	<a href="#">ConvolutionLayer</a>	1442	210	1232

Source : <https://github.com/soumith/convnet-benchmarks>

# Deep Learning Framework 비교

	Core Language	Interface Language	CPU	Single CPU	Multi GPU	Distributed	Comments
Caffe	C++	Python, MatLab	Yes	Yes	Yes	Com.yahoo.ml. CaffeOnSpark	이미지 처리에 특화. 텍스트, 사운드 등 데이터 처리 에는 부적합.
Theano / PyLearn 2	Python	Python	Yes	Yes	In Progress	No	일반적 목적을 위해 사용 Low-level을 제어할 수 있는 API 이지만 복잡하다.
Torch	Lua	Lua	Yes	Yes	Yes	Yes, but Torch '/'	쉽다
TensorFlow	C++	Python, C/C++, Java, Go	Yes	Yes	Yes	Yes	시각화 도구 TensorBoard, 안드로이드, iOS 지원, Low- level/High-level API 모두 제공
DL4J	Java	Java	Yes	Yes	Most likely	Yes	자바
CNTK	C++	Python, C++	Yes	Yes	Yes	Yes	처리 성능의 Linear Scaling
SystemML	Java		Yes	Yes	Not Yet	Yes	

# TensorFlow 설치

```
$ pip install tensorflow
```

## ■ setuptools 오류

- tensorboard 1.14.0 has requirement setuptools>=41.0.0, but you'll have setuptools 40.8.0 which is incompatible.
- setuptools를 Upgrade
  - (base) C:\Windows\system32>python -m pip install --upgrade setuptools

# TensorFlow 설치 확인

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
sess.close()
```

b'Hello, TensorFlow!'



---

# Deep Learning 이해하기

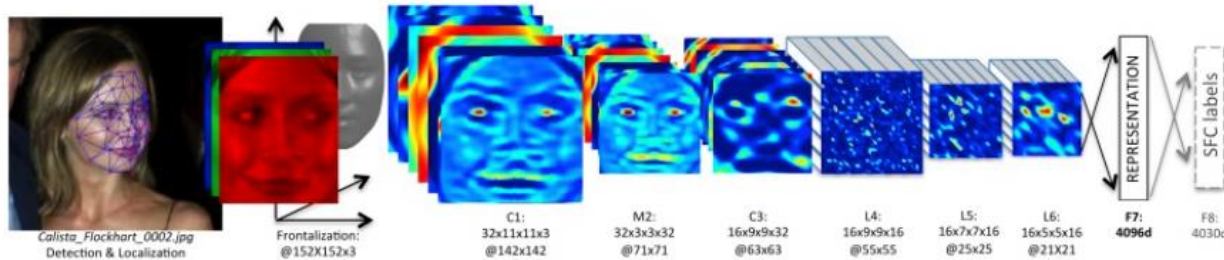


# Deep Learning 정의

## ■ Wikipedia:

- Deep learning is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using architectures composed of multiple non-linear transformations.

Example:  
Input data  
(image)



Prediction  
(who is it?)

*DeepFace rotating a celebrity's face to use for facial verification. Image credit: Facebook*

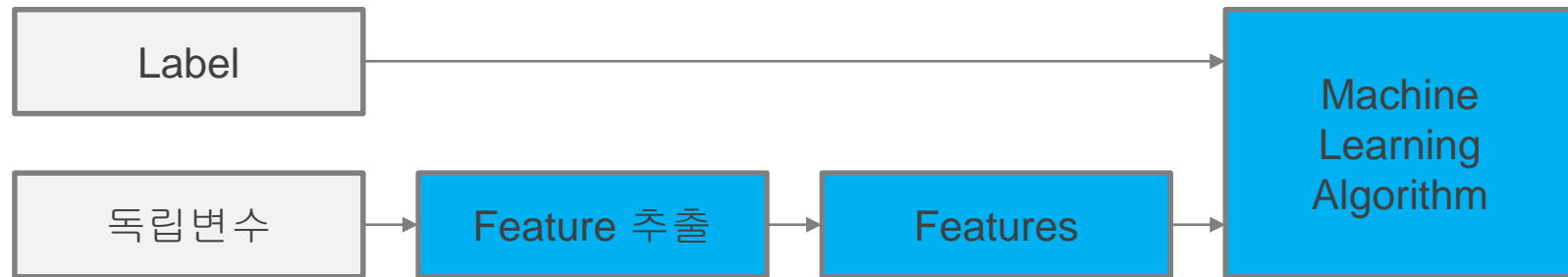
Deep Learning은 Data를 학습시켜 Model을 만드는 Machine Learning Algorithm의 집합



# Machine Learning vs Deep Learning

- Deep Learning은 학습데이터에서 주요 Feature를 추출/선택하는 과정까지도 학습

## Machine Learning 훈련 과정



## Deep Learning 훈련 과정



# 선형방정식으로 Model 구하기

```
In [1]: x = [[32,64,96,118,126,144,152.5,158], [1,1,1,1,1,1,1,1]]  
        y = [18,24,61.5,49,52,105,130.3,125]
```

```
In [2]: import numpy as np  
        A = np.array(x, order='C').T  
        B = np.array(y)
```

```
In [3]: print(A)
```

```
[[ 32.  1.]  
 [ 64.  1.]  
 [ 96.  1.]  
 [118.  1.]  
 [126.  1.]  
 [144.  1.]  
 [152.5  1.]  
 [158.  1.]
```

```
In [4]: print(B)
```

```
[ 18.  24.  61.5  49.  52. 105. 130.3 125.]
```

```
In [5]: #np.dot(x,y) 행렬의 곱  
        #np.linalg.inv(x) 역행렬  
        #np.linalg.solve()  
        A_inv = np.dot(np.linalg.inv(np.dot(A.T, A)), A.T)
```

```
In [6]: ab = np.dot(A_inv, B)
```

```
In [7]: print(ab)
```

```
[ 0.87493126 -26.79078617]
```

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\&\dots \\a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n\end{aligned}$$



$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$



간소화

$$AX = B$$



X를 구하려면?

$$X = A^{-1}B$$



$$\begin{bmatrix} a \\ b \end{bmatrix} = (A^T A)^{-1} A^T B$$

A가 정방행렬이 아니고 행의 수가 열의 수보다 크므로 left pseudo inverse를 이용

# 선형방정식으로 Model 구하기(더 쉬운 방법)

- Numpy의 **polyfit()** 함수를 이용하면 쉽게 결정계수  $w$ ,  $b$  값을 구할 수 있음

```
x = [32, 64, 96, 118, 126, 144, 152.5, 158]  
y = [18, 24, 61.5, 49, 52, 105, 130.3, 125]
```

```
import numpy as np  
w, b = np.polyfit(x, y, 1)
```

```
print(w, b)
```

```
0.8749312625981284 -26.7907861679542
```

# Deep Learning으로 Model 구하기

```
import tensorflow as tf
import datetime as dt
x_data = [32.0, 64.0, 96.0, 118.0, 126.0, 144.0, 152.5, 158.0]
y_data = [18.0, 24.0, 61.5, 49.0, 52.0, 105.0, 130.3, 125.0]

W = tf.Variable(tf.random_normal([1], [-10.0, 10.0]))
b = tf.Variable(tf.random_normal([1], [-100.0, 100.0]))

hypothesis = W * x_data + b

cost = tf.reduce_mean(tf.square(hypothesis - y_data))

rate = tf.Variable(0.00001)
optimizer = tf.train.GradientDescentOptimizer(rate)
train = optimizer.minimize(cost)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)

print(dt.datetime.now())
for step in range(2500001):
    sess.run(train)
    if step % 100000 == 0:
        print('{:7} {:<20.14f} {} {}'.format(step, sess.run(cost),
                                              sess.run(W), sess.run(b)))

print(dt.datetime.now())
sess.close()
```

```
2019-07-11 15:45:33.584657
0 7757.28222656250000 [ 1.26818371] [-155.01182556]
100000 1594.54614257812500 [ 1.66345227] [-126.93096924]
200000 1111.77551269531250 [ 1.49090207] [-105.01753998]
300000 817.21948242187500 [ 1.35613441] [-87.90235138]
400000 637.27111816406250 [ 1.25072932] [-74.51621246]
500000 527.81628417968750 [ 1.16866457] [-64.09416199]
600000 460.83105468750000 [ 1.10439444] [-55.93204117]
700000 419.94287109375000 [ 1.05417418] [-49.55418396]
800000 394.95355224609375 [ 1.01487303] [-44.56302261]
900000 379.80795288085938 [ 0.9843877] [-40.69148254]
1000000 370.52301025390625 [ 0.96048731] [-37.65619278]
1100000 364.79058837890625 [ 0.94158077] [-35.25510788]
1200000 361.37966918945312 [ 0.92718023] [-33.42626953]
1300000 359.25460815429688 [ 0.91571325] [-31.96999741]
1400000 357.96847534179688 [ 0.90683025] [-30.84187508]
1500000 357.18487548828125 [ 0.89991522] [-29.96367836]
1600000 356.70297241210938 [ 0.89448369] [-29.27389526]
1700000 356.39553833007812 [ 0.89002919] [-28.7081852]
1800000 356.23291015625000 [ 0.88702548] [-28.32671547]
1900000 356.10662841796875 [ 0.8840366] [-27.94713974]
2000000 356.05664062500000 [ 0.88253474] [-27.75640488]
2100000 356.01562500000000 [ 0.88103288] [-27.56567001]
2200000 355.98364257812500 [ 0.87953103] [-27.37493515]
2300000 355.96069335937500 [ 0.87802917] [-27.18420029]
2400000 355.95977783203125 [ 0.87797189] [-27.17692947]
2500000 355.95977783203125 [ 0.87797189] [-27.17692947]
2019-07-11 15:52:31.466055
```



---

# DNN

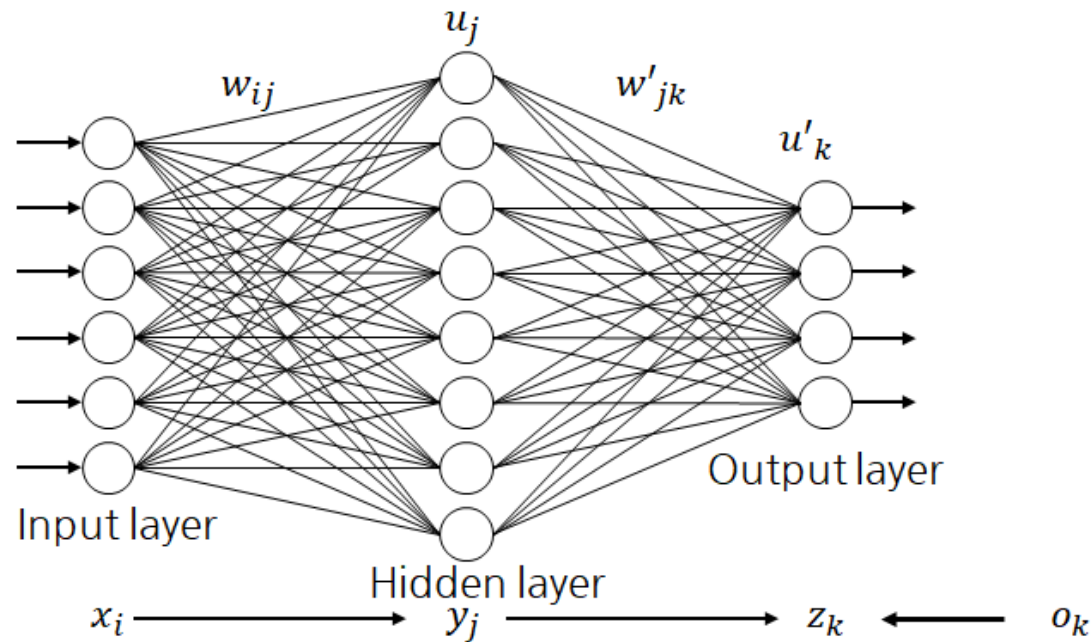


# DNN 이해



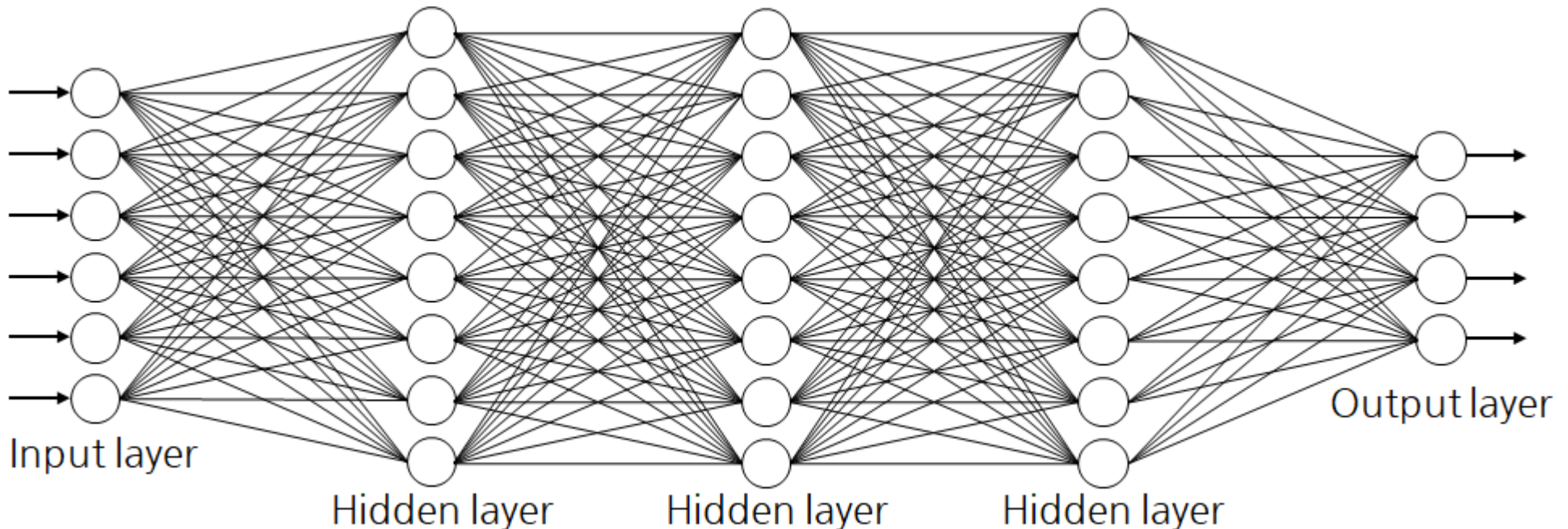
# DNN 구조

- DNN(Deep Neural Network)은 Deep Learning을 구현하기 위한 Algorithm들 중에서 가장 기본이 되는 심층 신경망
- DNN은 [입력 계층(Input Layer)] - [은닉 계층(Hidden Layer)] - [출력 계층(Output Layer)]으로 나뉘져 있음



# 은닉 계층이 3개인 인공신경망

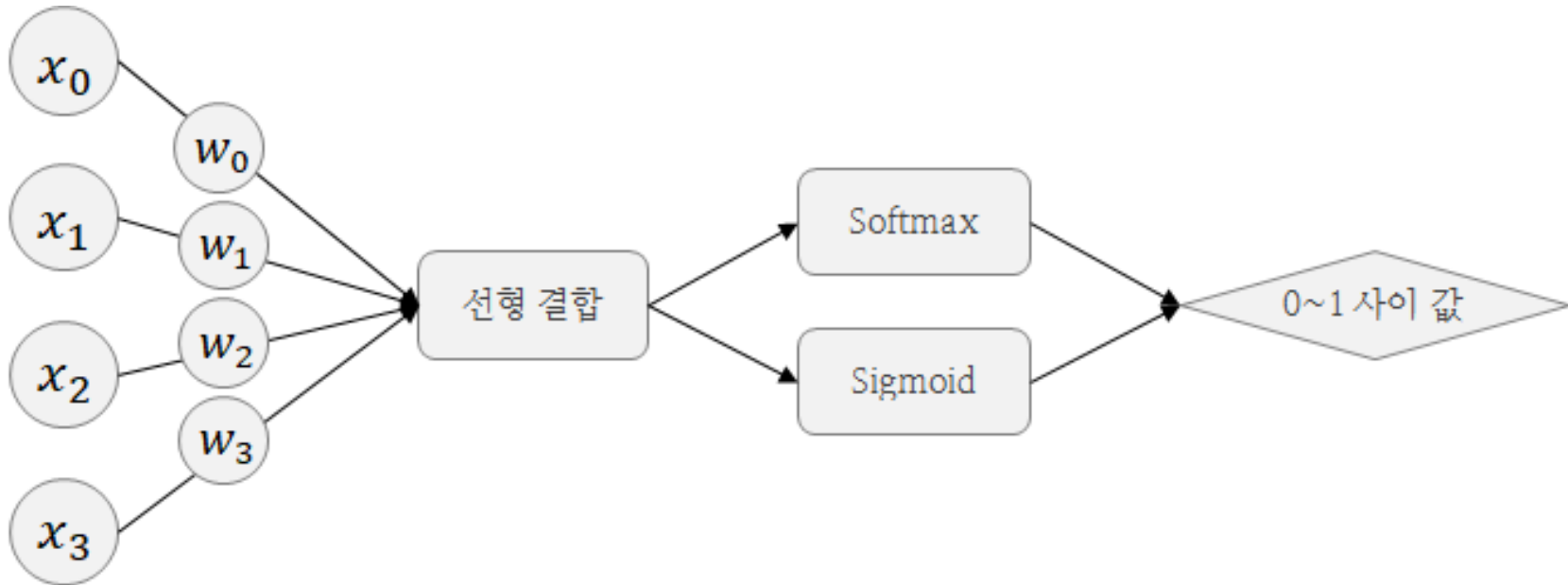
- 일반적으로 은닉 계층이 3개 이상 들어간 인공 신경망을 DNN
- 실제로 구현된 DNN의 경우에는 이렇게 간단하지 않음
- 은닉 계층이 60~90개 정도로 이뤄진 신경망도 많이 있음





# 활성화 함수 (Activation Function)

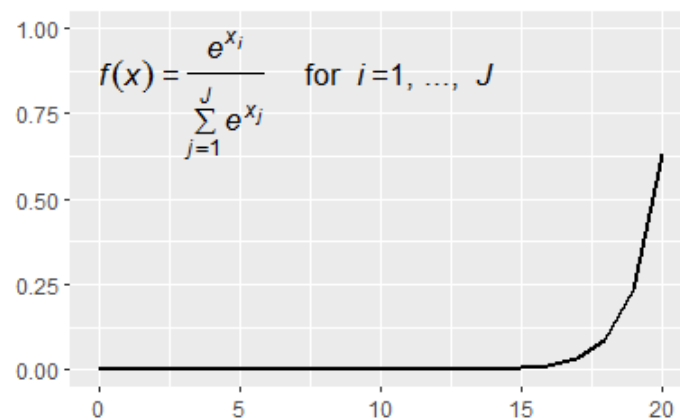
- 입력 값들은 가중치(Weight)와의 선형 결합과 활성화 함수 (Activation Function)를 통해 출력 값으로 변환



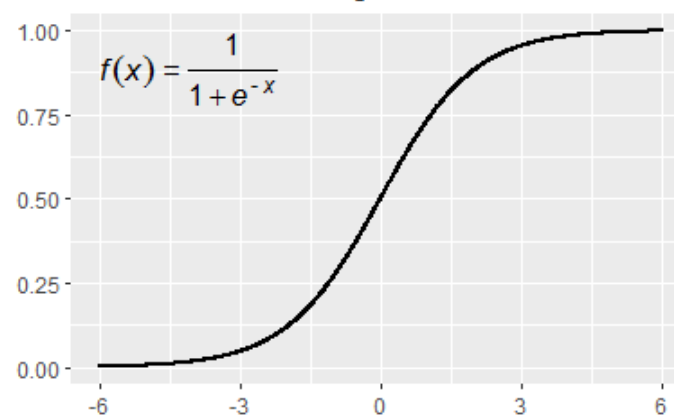
# 활성화 함수(Activation Function) (Cont.)

- 생물학적 뉴런(neuron)에서 입력 신호가 일정 크기 이상일 때만 신호를 전달하는 Mechanism을 모방한 함수

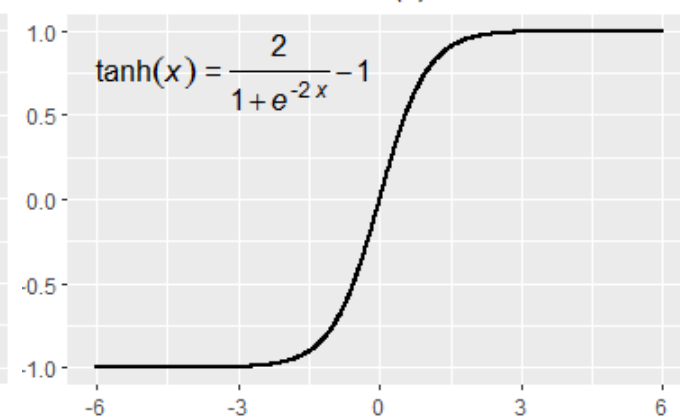
Softmax



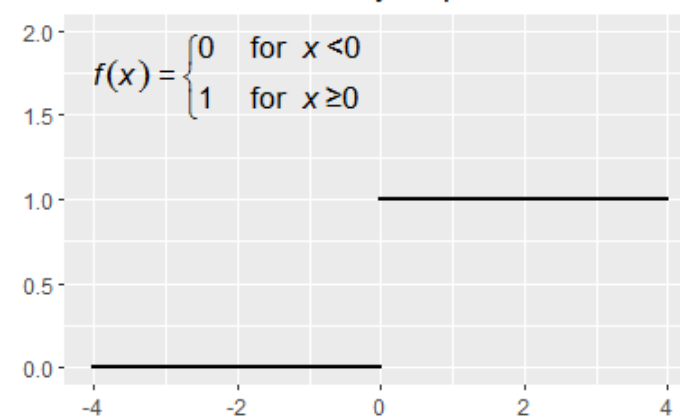
Sigmoid



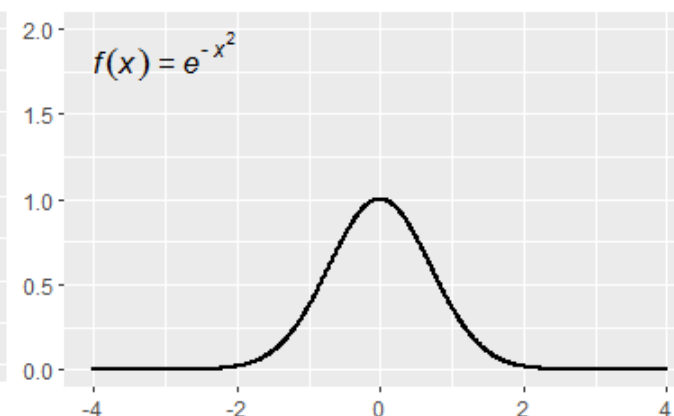
tanh(x)



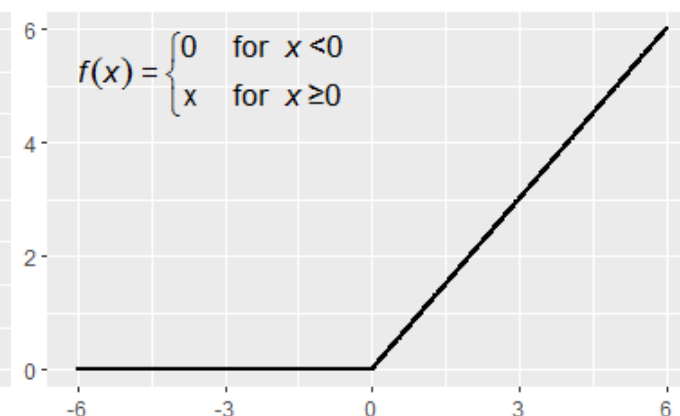
Binary step



Gaussian

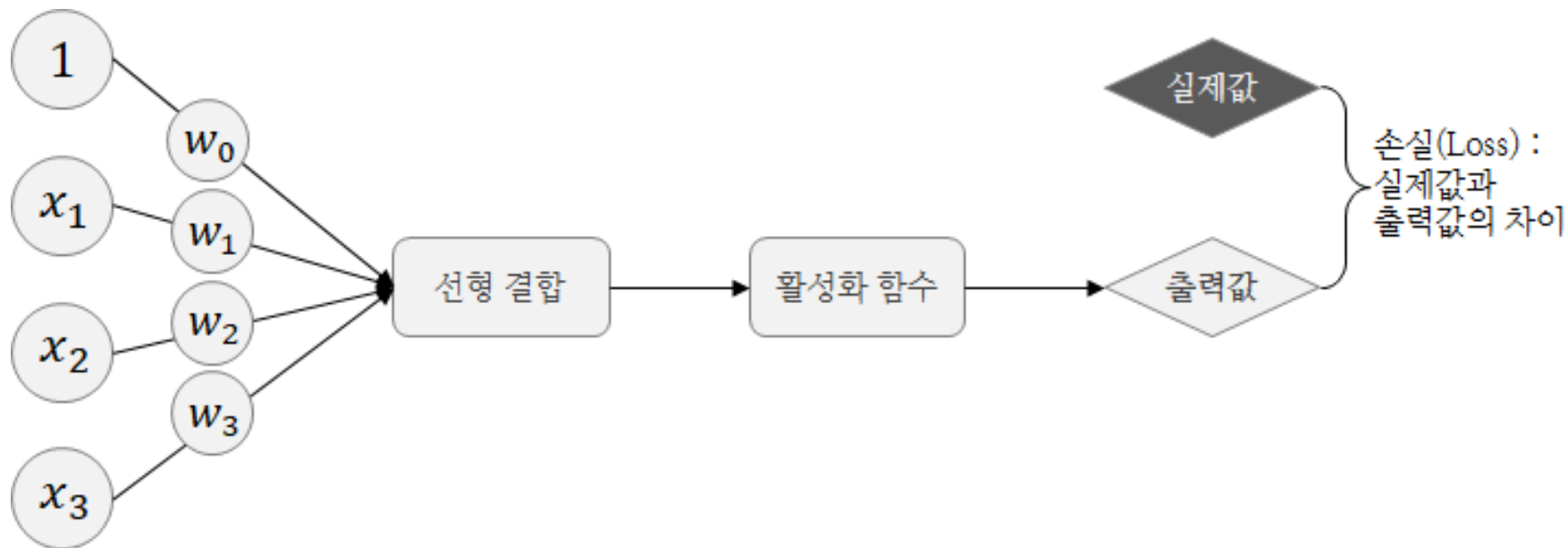


ReLU



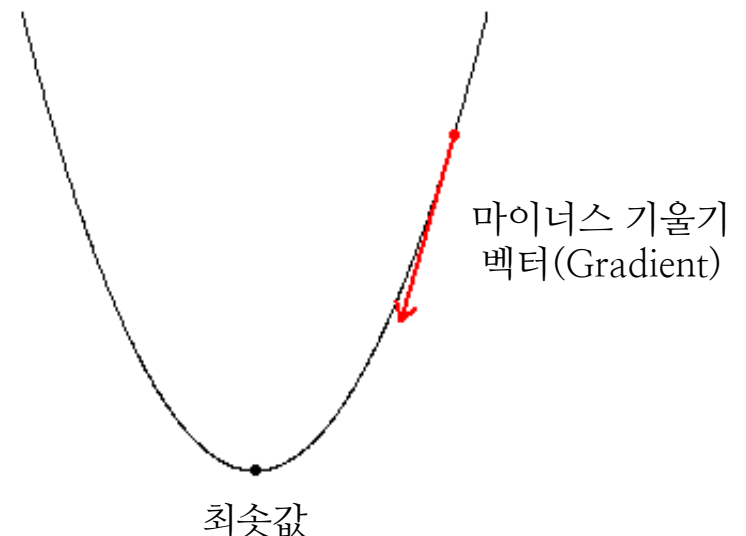
# 손실함수

- 인공신경망에서 가중치(Weight)는 학습을 통해 손실(Loss)을 최소화 하는 방향으로 추정
- 추론한 값과 실제 값의 차이를 손실(Loss)로 정의하고 이러한 차이를 표현한 함수를 손실 함수(Loss Function)이라고 함



# 경사하강법

- 손실을 최소화 하는 방향은 기울기 벡터(Gradient)를 통해 정해짐
  - 마이너스 기울기 벡터(Gradient)는 현재 위치에서 함수의 최솟값으로 가는 가장 빠른 방향을 정해 줌
  - 이와 같은 과정을 반복하여 마이너스 기울기 벡터를 따라가게 되면 결국은 해당 함수의 최솟값으로 수렴할 수 있음
  - 이러한 방법을 경사하강법(Gradient Descent)이라고 함
- 인공신경망의 관점에서 설명하며, 학습을 통해 현재 가중치 값에 해당하는 손실함수의 마이너스 기울기 벡터를 구하게 되고 이런 과정을 반복해 손실을 최소화 하는 가중치 값을 추정하게 됨
- 이렇게 추정된 값을 기반으로 모델이 결정 됨





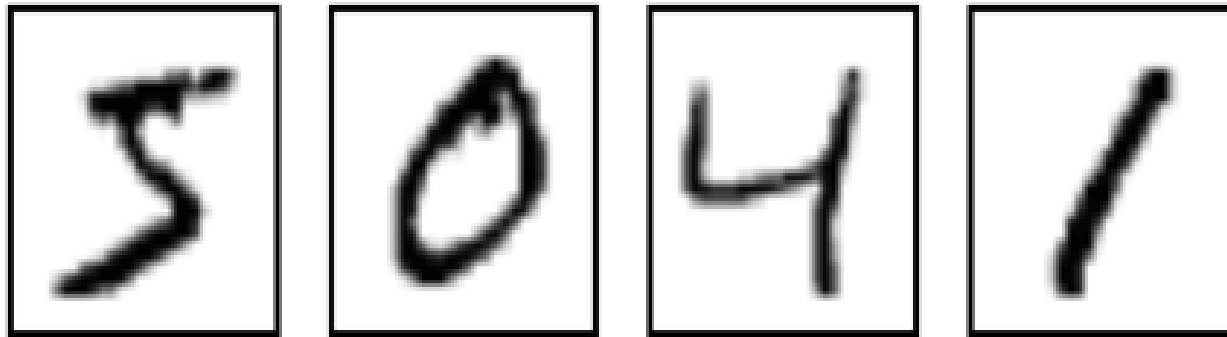
---

# MNIST Data로 우편번호 손글씨 자동분류기 만들기



# 우편번호 자동 분류기

- 우편번호 자동 분류기처럼 숫자를 분류할 수 있도록 만들기 위해 필요한 것
  - Deep Learning 구현 Algorithm
  - 손으로 쓴 숫자 Data

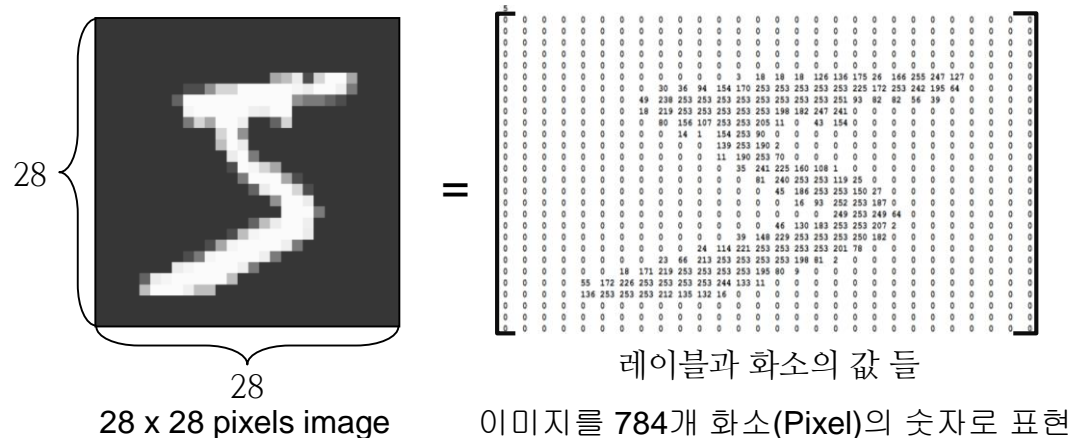


## 우편번호 자동 분류기 (Cont.)

- 이 문제를 해결하기 위해 앞에서 학습한 DNN을 사용할 수 있다.
- 그런데, 손으로 쓴 숫자 Data는 어디서 구할 수 있는가?
- Deep Learning으로 학습시키기 위해서는 많은 학습 DataSet이 필요하다. 그런데, 우리가 손으로 일일이 숫자를 쓰는 것은 너무 불편한 일이다. 0 ~ 9까지 숫자를 각 숫자마다 1000개 이상 써야 한다면...그리고 그 숫자들을 일일이 파일로 저장하고 전처리하고...이러한 일들은 너무 불편한 일이다. 우리가 지금 하고 있는 것은 Deep Learning을 배우는 것이다. 그러므로 전처리 및 형식 지정 등은 최소한의 노력을 기울이고, 다른 누군가의 도움을 받아 해결하는 것이 더 바람직하다.

# MNIST Data

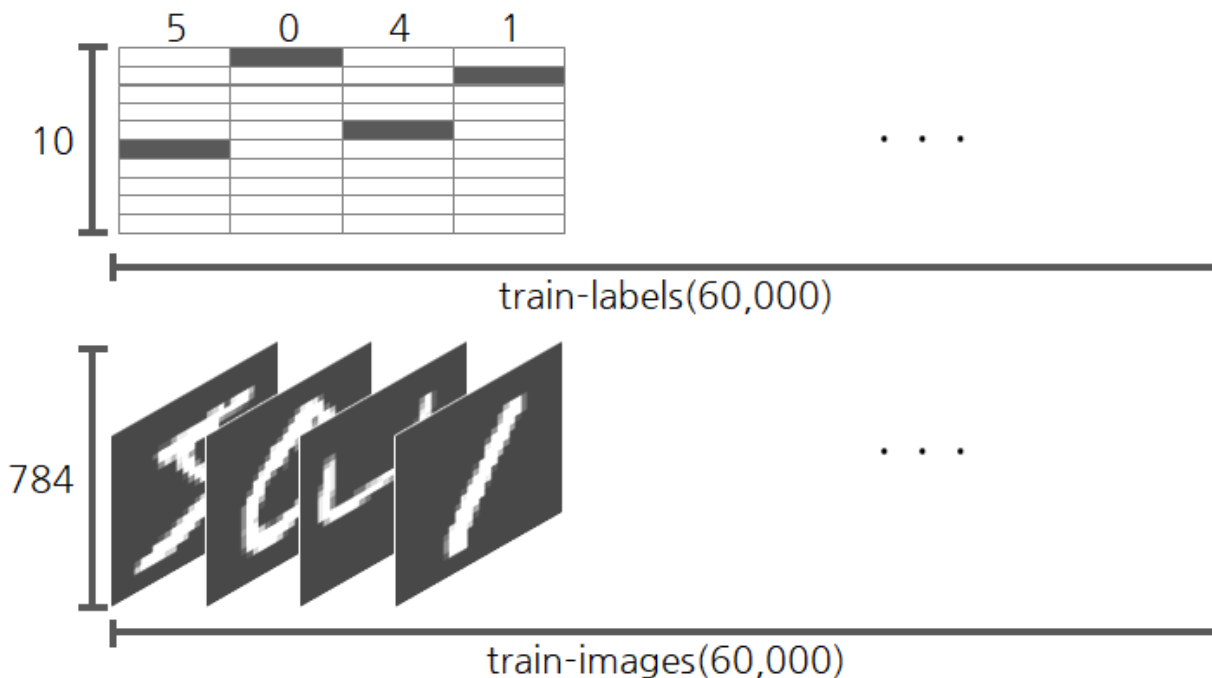
- 손으로 쓴 숫자의 Image Database
- National Institute of Standards and Technology가 제작
- 다운로드 : <http://yann.lecun.com/exdb/mnist>
- 55,000개의 학습용 이미지(mnist.train) + 10,000개의 테스트 이미지(mnist.test) + 5,000개의 검증 이미지(mnist.validation)
- 각 이미지는 28x28 크기, 이것을 펼치면 784 차원의 벡터





# One-Hot Vector

- MNIST의 레이블은 연속된 숫자가 아닌 카테고리 값을 가져야 하므로 원-핫 인코딩(One-Hot Encoding)이 되어있어야 함
- 원-핫 벡터(One-Hot Vector)
  - 레이블이 이미지가 숫자 5임을 알려주기 위해  $[0,0,0,0,0,1,0,0,0,0]$  형식처럼 구성되어 있는 1차원 배열 형식으로 저장되어 있어야 함



# MNIST Data 불러오기

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes  
Extracting MNIST\_data/train-images-idx3-ubyte.gz  
Successfully downloaded train-labels-idx1-ubyte.gz 51456 bytes  
Extracting MNIST\_data/train-labels-idx1-ubyte.gz  
Successfully downloaded t10k-images-idx3-ubyte.gz 9912422 bytes  
Extracting MNIST\_data/t10k-images-idx3-ubyte.gz  
Successfully downloaded t10k-labels-idx1-ubyte.gz 51456 bytes  
Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

실행 시 출력되는 경고는 무시할 것.

현재 TensorFlow 공식  
사이트에서는 Keras를  
이용해 설명하고 있다.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```



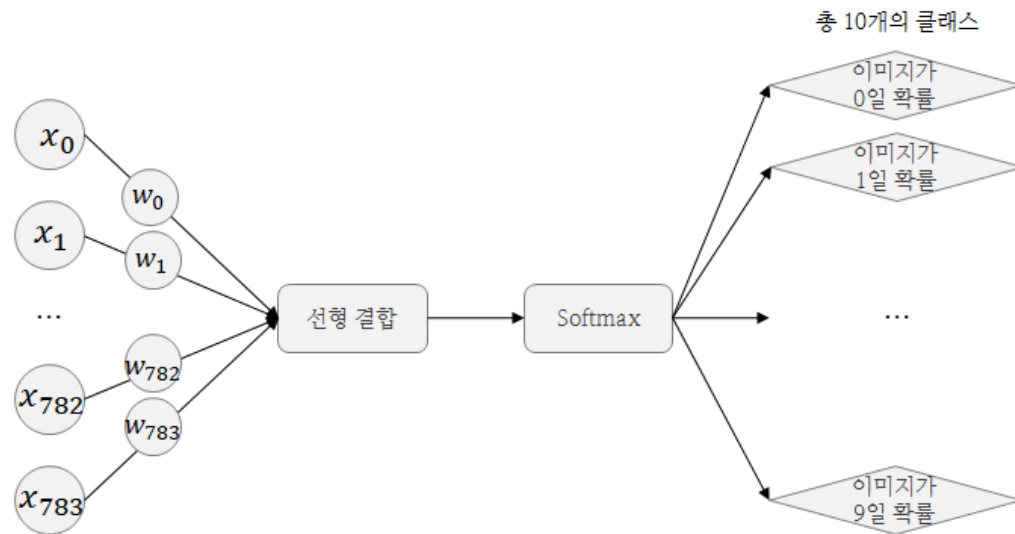
---

# 단일 계층을 이용한 Modeling



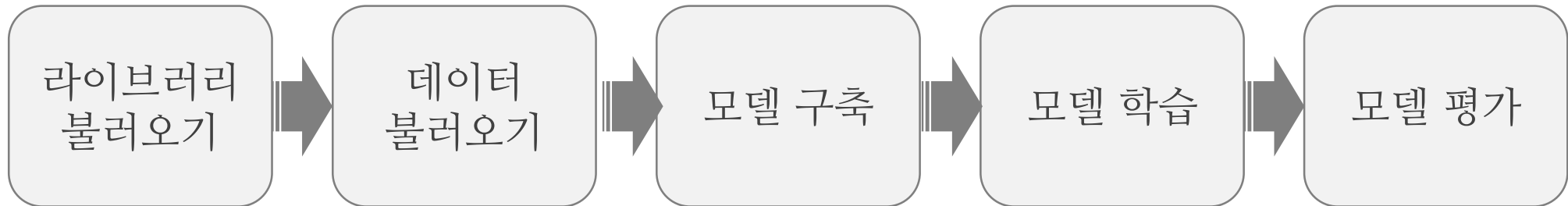
# 단일 계층 모델링 구성도

- 입력 값은 각 숫자 이미지의 화소 값( $28 \times 28 = 784$  화소)
- 출력은 입력한 이미지가 0부터 9까지 숫자 중에서 어떤 것 일지에 대한 확률
- 0부터 9까지 중에서 하나를 분류해야 하므로 출력은 총 10개의 클래스
- 추정된 총 10개의 확률 중에서 가장 높은 값을 갖는 클래스를 해당 숫자 이미지로 추론



# 분석 과정

- 분석 과정은 Library와 Data를 불러온 후 Model을 구축하고 학습 시킴
- 학습이 완료되면 TestSet을 이용하여 Model을 평가함



Tensorflow  
Numpy

```
read_data_set()  
x = tf.placeholder()  
y = tf.placeholder()
```

```
W = tf.Variable()  
b = tf.Variable()  
h = f(x*W + b)  
loss = reduce(error)  
Optimizer().minimize(loss)
```

```
global_variables_initializer()  
Session()  
run(operation, feed_dict={})
```

# 데이터 불러오기

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.  
Extracting MNIST\_data/train-images-idx3-ubyte.gz  
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.  
Extracting MNIST\_data/train-labels-idx1-ubyte.gz  
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.  
Extracting MNIST\_data/t10k-images-idx3-ubyte.gz  
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.  
Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

# 입출력 정의

- placeholder는 Machine Learning에 전달되는 데이터를 변경하기 위한 수단
  - placeholder를 만들 때는 Data의 자료형에 대해 알려줘야 함
- MNIST DataSet의 각 이미지들은  $28 \times 28 = 784$  화소 정보들로 구성되어 있음
- x 변수를 float32 타입, [None, 784] 크기로 정함,
  - None 이라고 정한 것은 '어떠한 크기도 가능하다'는 것을 의미함
- 타겟 변수 y는 출력 클래스가 10개 이므로 [None, 10]으로 크기를 정함

```
1 x = tf.placeholder(tf.float32, [None, 784])
2 y = tf.placeholder(tf.float32, [None, 10])
```

# 가중치와 편향을 저장할 변수 선언

- 두 변수는 반복된 훈련으로 최적의 값을 찾는 것이 목표
- `tf.Variable()` 함수를 이용하여 변수를 생성
  - 모두 0으로 초기화
- 가중치가 [784, 10]이고, 편향이 [10]인 것은 하나의 이미지가 갖는 입력 값이 784(28x28)이고, 출력 값이 10개의 클래스를 가져야 함

```
1 W = tf.Variable(tf.zeros([784, 10]))
2 b = tf.Variable(tf.zeros([10]))
```



## 출력 값 정의

- $H = wx + b$  수식을 계산해야 한다면  $x$ 의 크기는  $[None, 784]$ 이며,  $W$ 의 크기는  $[784, 10]$ 이므로 이를 행렬의 곱을 이용해 계산하면 출력은  $[10]$ 이 된다.
- 이를 계산하기 위한 코드가  $tf.matmul(x, W) + b$
- 활성화 함수는 소프트맥스(SoftMax)
  - Softmax 함수는 주로 DNN의 마지막 출력 노드에서 분류(Classification)를 사용하고자 할 때 사용
  - 여러 개의 Softmax 값들(0에서 1사이의 값들)을 합치면 1이 되는 확률로 해석하기 위해 사용한 것

```
1 h = tf.nn.softmax(tf.matmul(x, W) + b)
```

# 손실함수 정의

- 손실함수란 모델을 통해 얻은 출력 값과 실제 값 사이의 오차를 나타내는 함수
- 오차를 측정하는 방법
  - 선형회귀모델에서 사용하는 평균제곱오차, 유클리드제곱거리 등이 있음
  - 예제에서는 '크로스 엔트로피 함수(Cross Entropy Function)'라는 손실함수를 사용

$$H_{y'}(y) = - \sum_i y_i' \log(y_i)$$

```
1 cross_entropy = -tf.reduce_sum(y * tf.log(h),  
2                               reduction_indices=[1])
```

# 학습 방법 정의

- 손실을 최소화 하도록 학습을 정의함
- 경사하강법(Gradient descent)은 함수의 한 점이 기울기 벡터(Gradient)의 반대 방향으로 이동할 때 그 함수 값을 가장 빠르게 감소시킨다는 특성을 이용하여 최솟값을 찾아나가는 방법

```
1 loss = tf.reduce_mean(cross_entropy)
```

```
1 optimizer = tf.train.GradientDescentOptimizer(0.5)  
2 train = optimizer.minimize(loss)
```

# 변수 초기화

- 모든 변수를 초기화 한 후 세션을 시작

```
1  init = tf.global_variables_initializer()  
2  
3  sess = tf.Session()  
4  sess.run(init)
```

# 학습

```
1 for i in range(1001):
2     batch_xs, batch_ys = mnist.train.next_batch(100)
3     _, loss_value = sess.run([train, loss],
4                               feed_dict={x: batch_xs, y: batch_ys})
5     if i % 100 == 0:
6         print('Step %d, %.5f' % (i, loss_value))
```

Step 0, 2.30259  
Step 100, 0.43021  
Step 200, 0.60392  
Step 300, 0.24596  
Step 400, 0.38157  
Step 500, 0.39824  
Step 600, 0.37457  
Step 700, 0.15268  
Step 800, 0.30896  
Step 900, 0.49296  
Step 1000, 0.41663

확률적 훈련(Stochastic Training)을  
이용하여 데이터를 학습

# 평가

- accuracy를 정의하고 Test Dataset을 입력으로 Operation을 실행시킴

```
1 correct_prediction = tf.equal(tf.argmax(h, 1), tf.argmax(y, 1))
2 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
3 print(sess.run(accuracy, feed_dict={x: mnist.test.images,
4                                     y: mnist.test.labels}))
```

0.9205

# 예측 값과 실제 값 비교

```
mnist.test.labels[0:10]
```

```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]])
```

```
pred = sess.run(y, feed_dict={x: mnist.test.images[0:10]})
pred
```

...

```
import numpy as np
print(np.argmax(mnist.test.labels[0:10], axis=1))
```

```
[7 2 1 0 4 1 4 9 5 9]
```

```
print(np.argmax(pred, axis=1))
```

```
[7 2 1 0 4 1 4 9 6 9]
```