

Deep Learning

Bok, Jong Soon

javaexpert@nate.com

<https://github.com/swacademy>

1. 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 제한된
이론 구현의 어려움
- 데이터의 한계로 현실
문제 해결하지 못함
- 알고리즘의 한계로 완
성도 부족

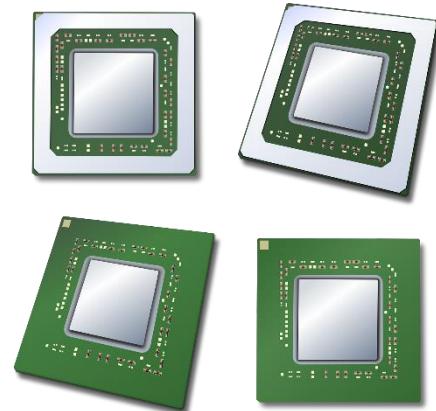
혁신적 알고리즘 제안



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

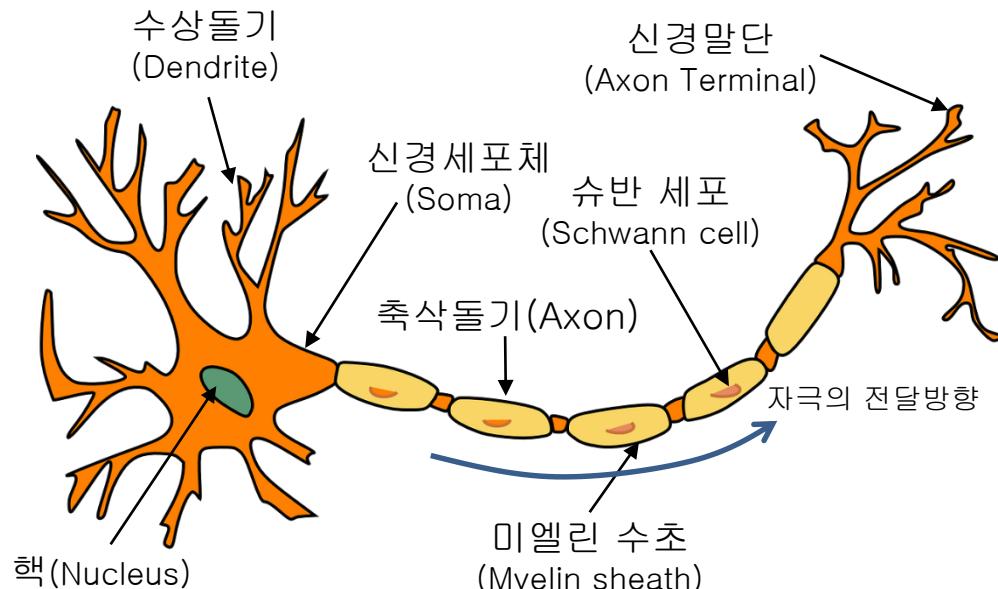
■ 인간의 뉴런(Neuron)

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

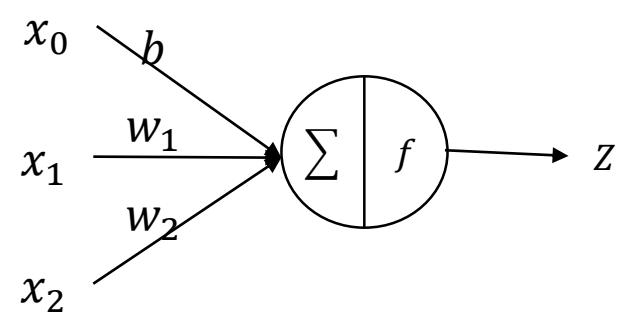
■ 인공 뉴런(노드; Node)

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

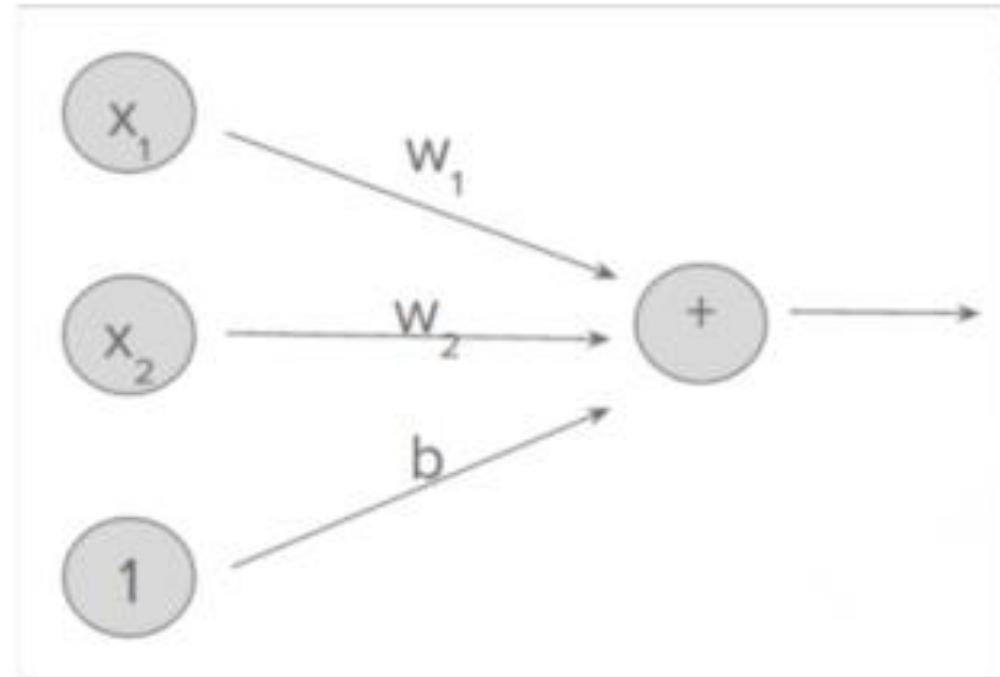
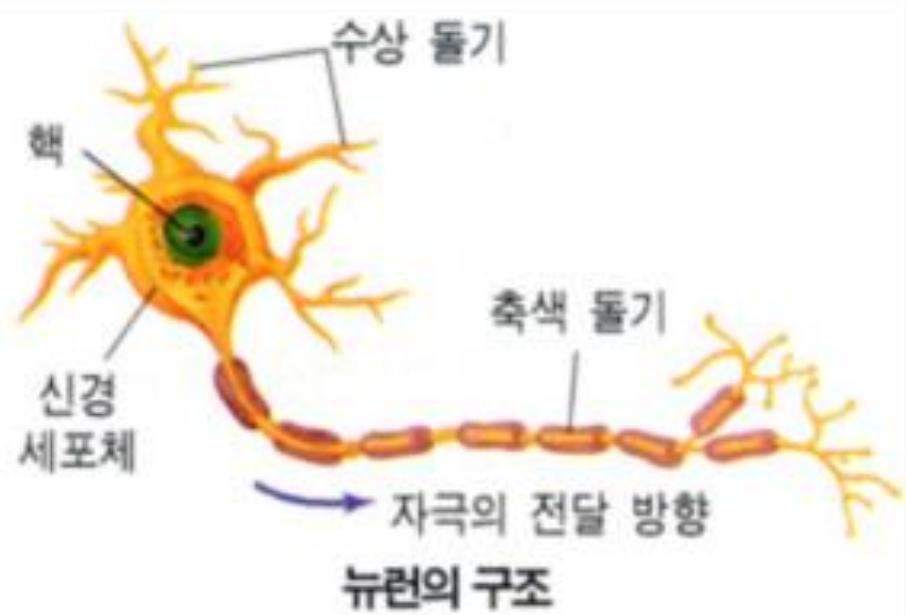
인간의 Neuron 구조



인공 뉴런의 구조

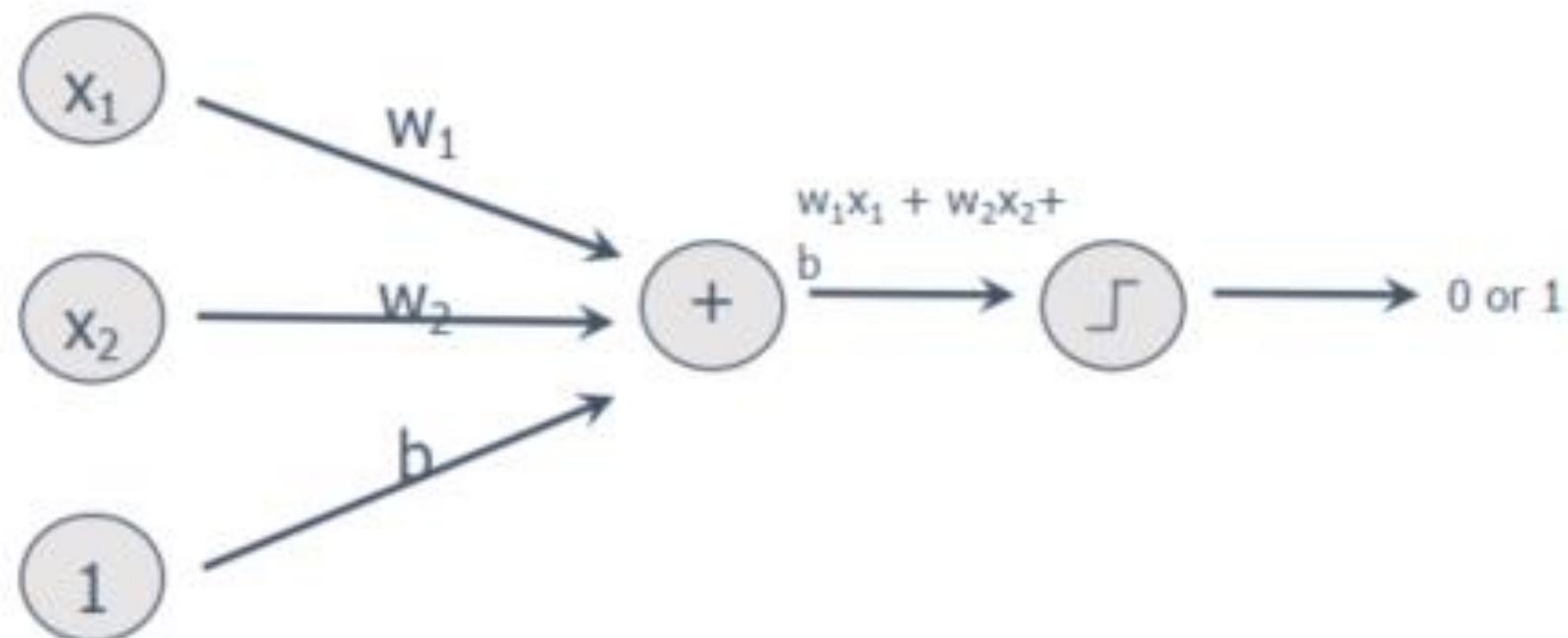


Neuron과 유사

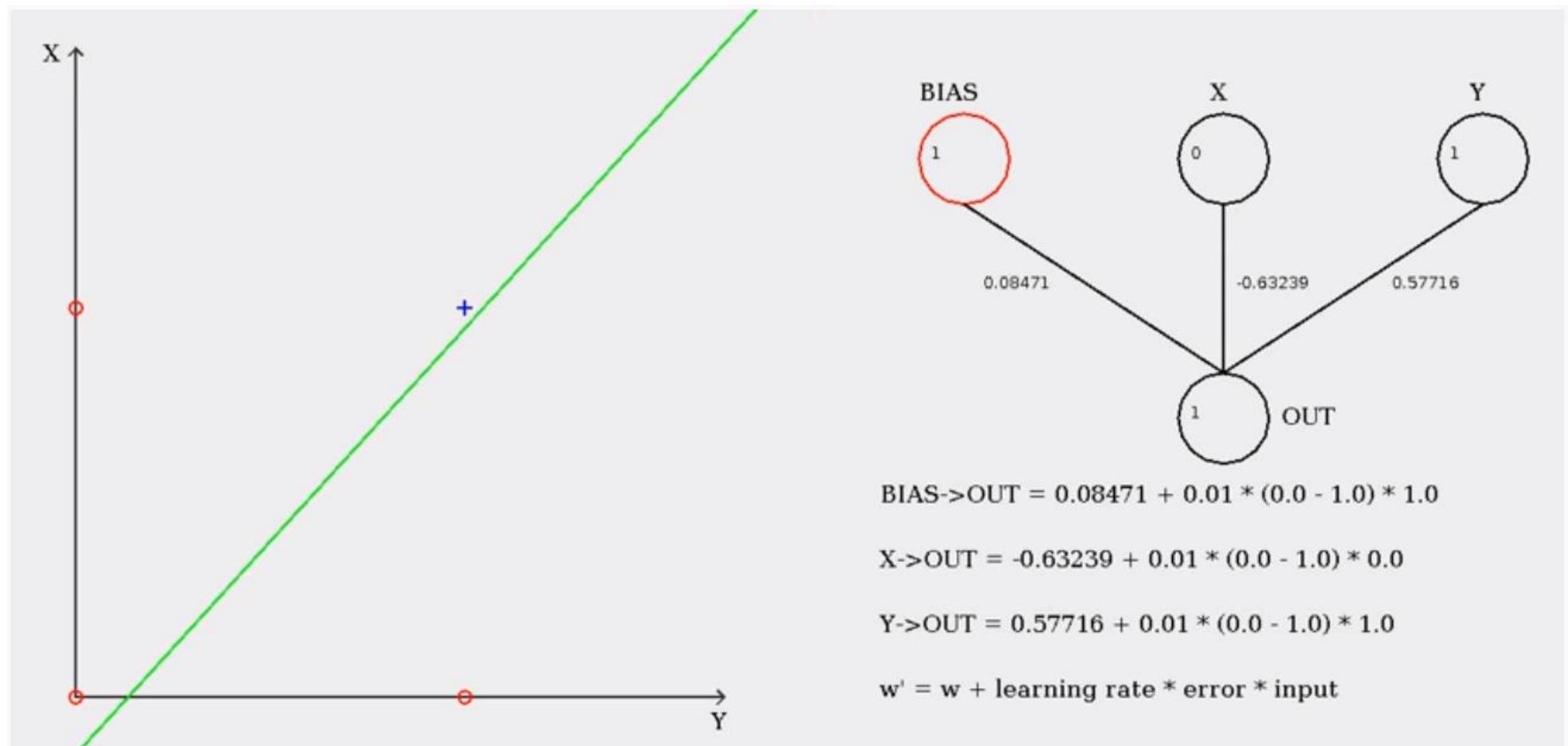


Perceptron

- 선형 분리의 문제를 학습할 수 있음.



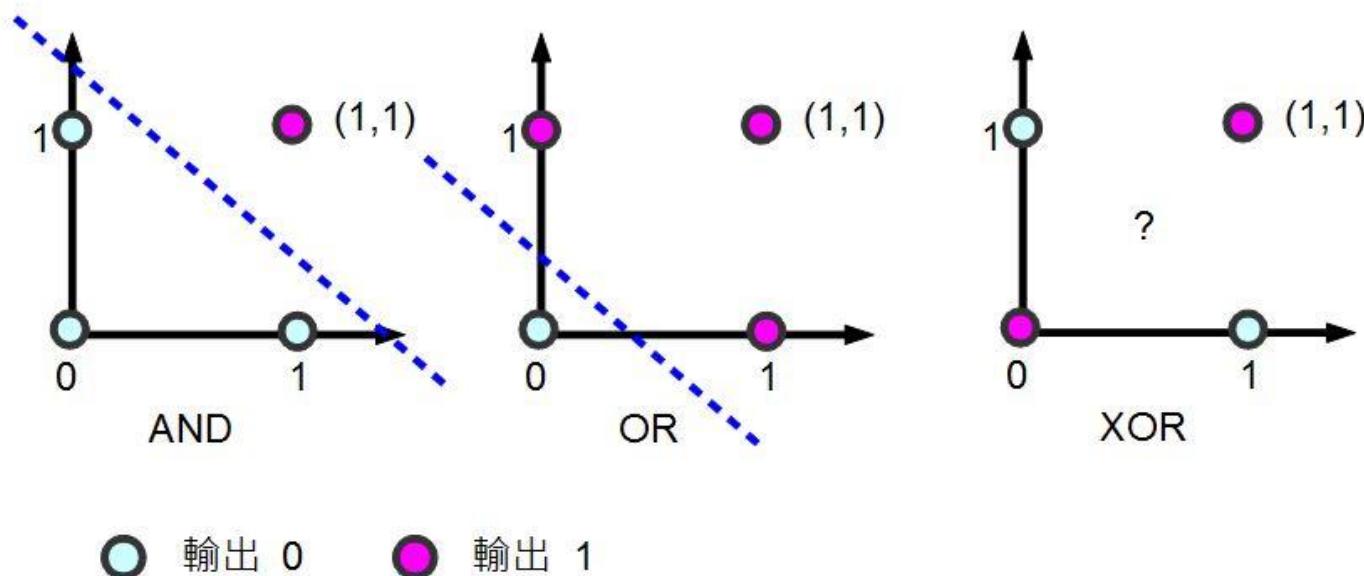
Perceptron Learning Algorithm 1 - AND



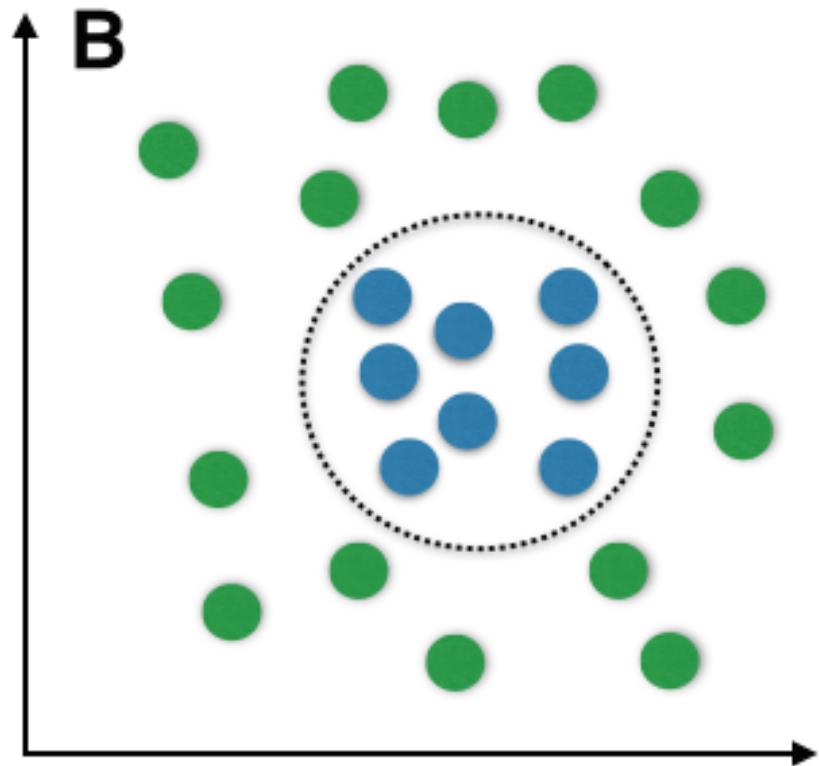
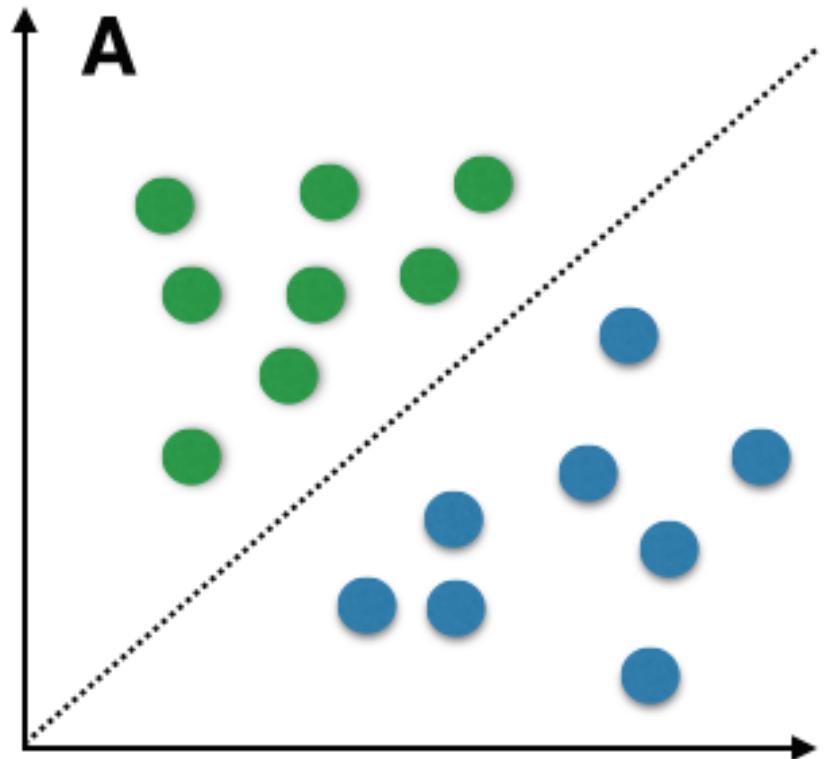
<https://www.youtube.com/watch?v=tYxkIOTdeu8>

Perceptron 의 한계

- 하지만, 간단한 XOR도 못한다.
- 선형분리가 불가능한 것은 풀지 못한다.
- XOR는 선형분리로 풀지 못하는 문제이다.

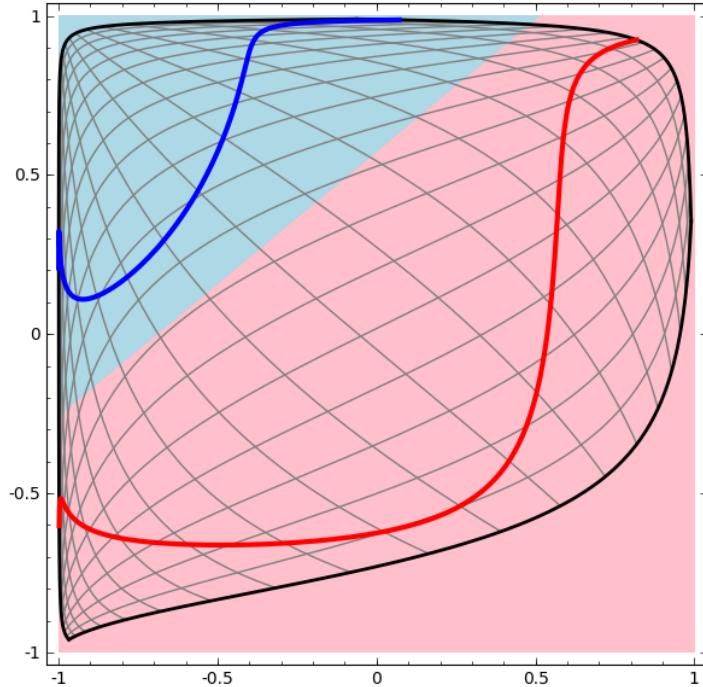
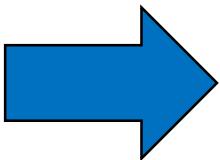
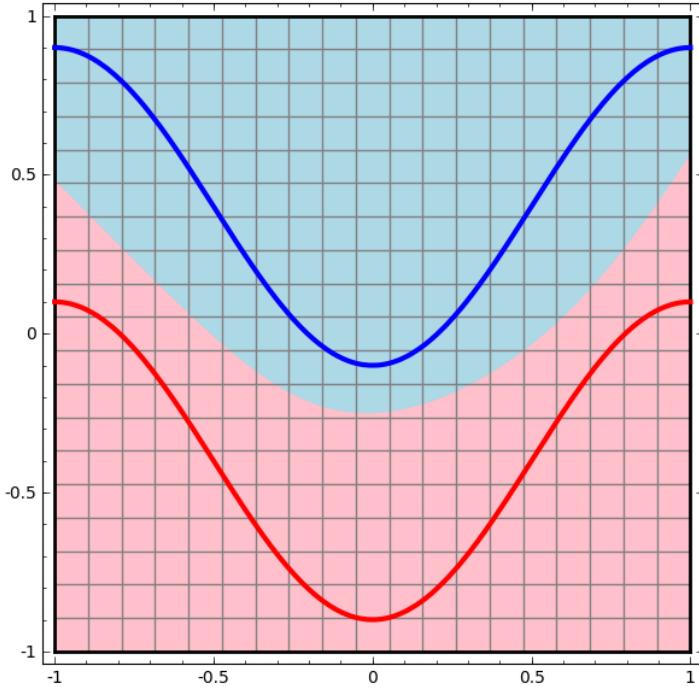


선형 분리 여부



선형 분리 불가 문제의 해결법

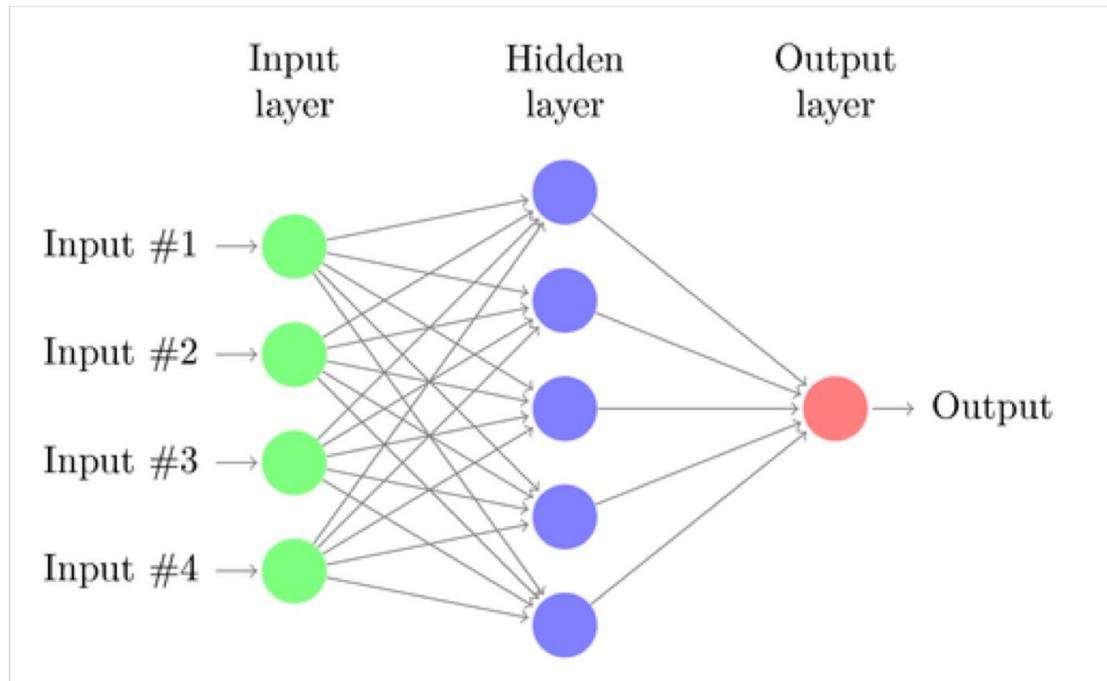
- 입력 차원을 늘린다.
- 입력을 비선형 변환하여 선형분리 가능하도록 한다.



- MLP

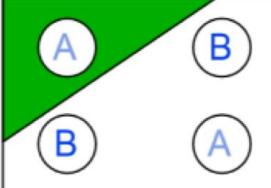
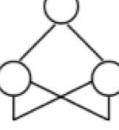
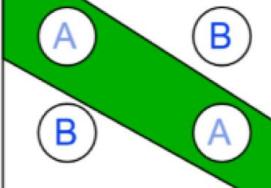
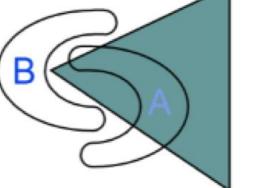
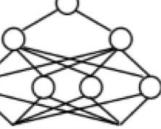
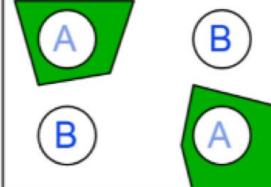
MLP(Multi Layer Perceptron)

- 입력과 출력 사이에 층이 더 있다.
- 개별 Perceptron의 결과를 다음 층의 입력으로 사용하고 결과적으로 선형 분리의 제약을 극복한다.

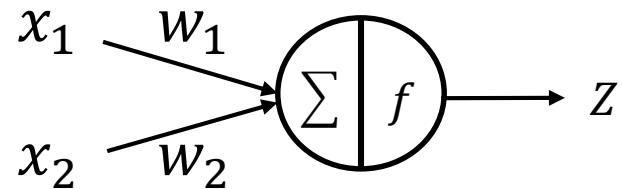


Perceptron의 능력

- 1개의 Perceptron은 1개의 선형 분리를 할 수 있음.
- Perceptron의 결과를 다른 Perceptron의 입력으로 하면 여러 개의 선으로 분리할 수 있음.

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

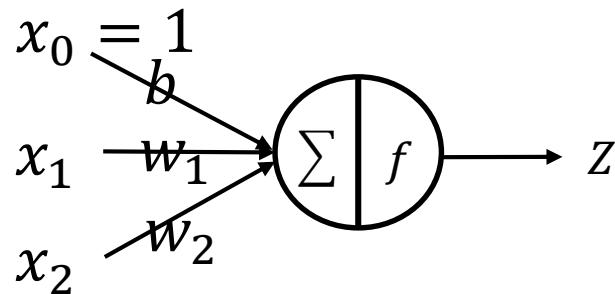
인공신경망의 Neuron



$$\sum = U = W_1X_1 + W_2X_2$$

$$Z = f(u)$$

if $Z > b$ then 1 else 0

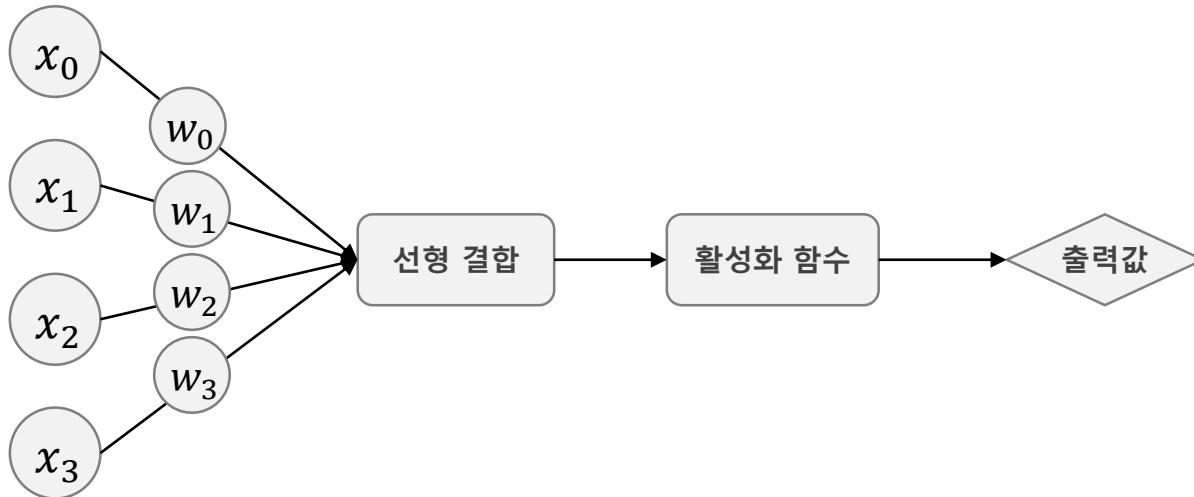
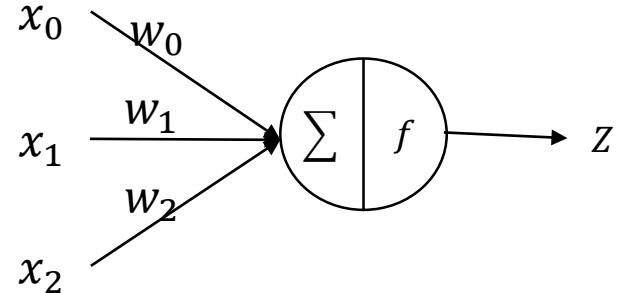
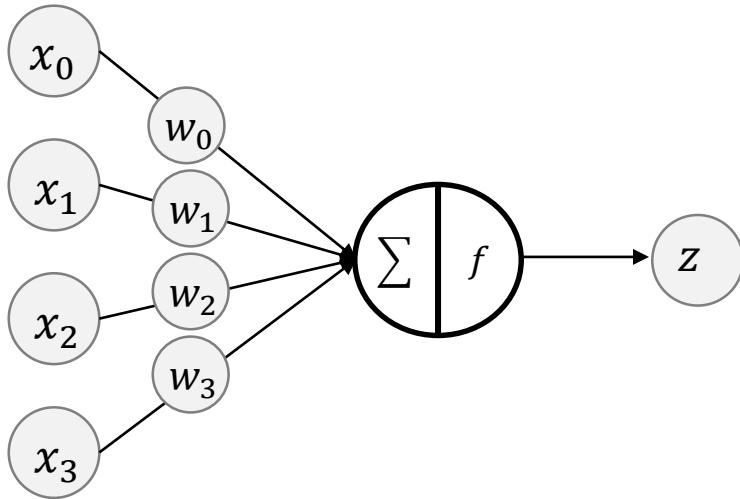


$$\sum = U = W_1X_1 + W_2X_2 + b$$

$$Z = f(u)$$

if $Z > b$ then 1 else 0

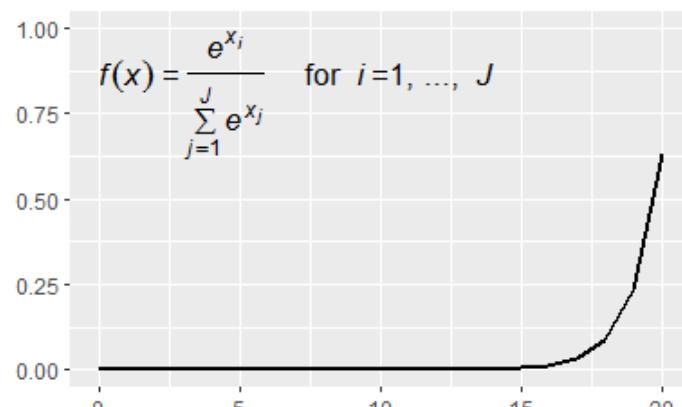
Neuron의 표현 방법



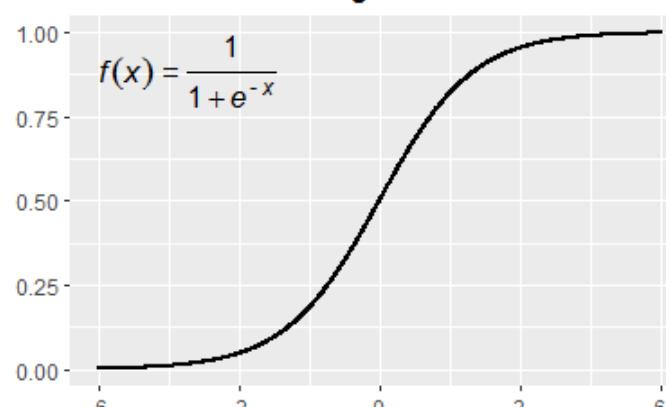
활성화 함수(activation function)

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

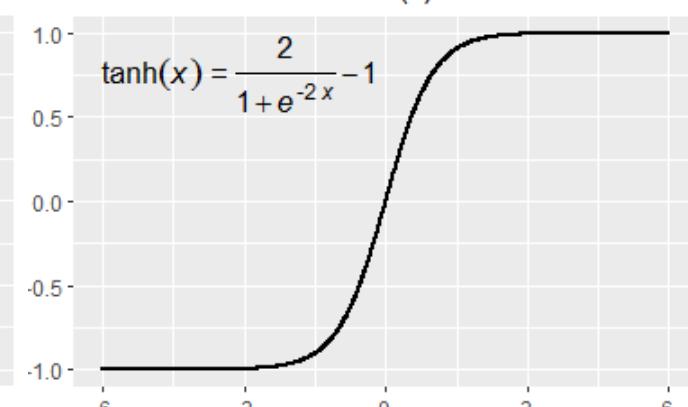
Softmax



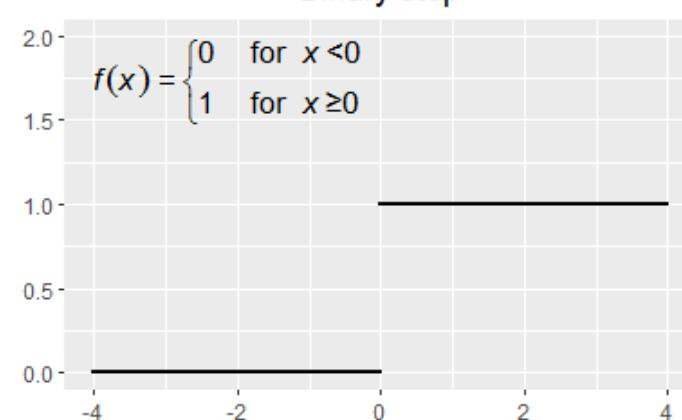
Sigmoid



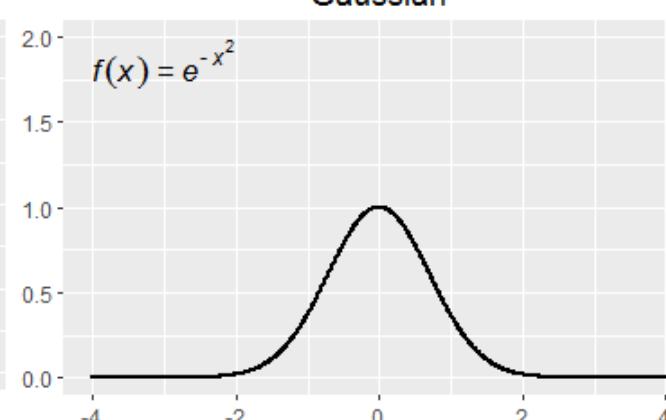
tanh(x)



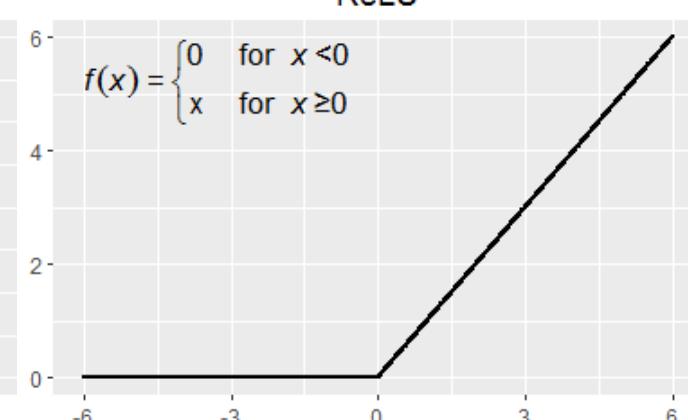
Binary step



Gaussian

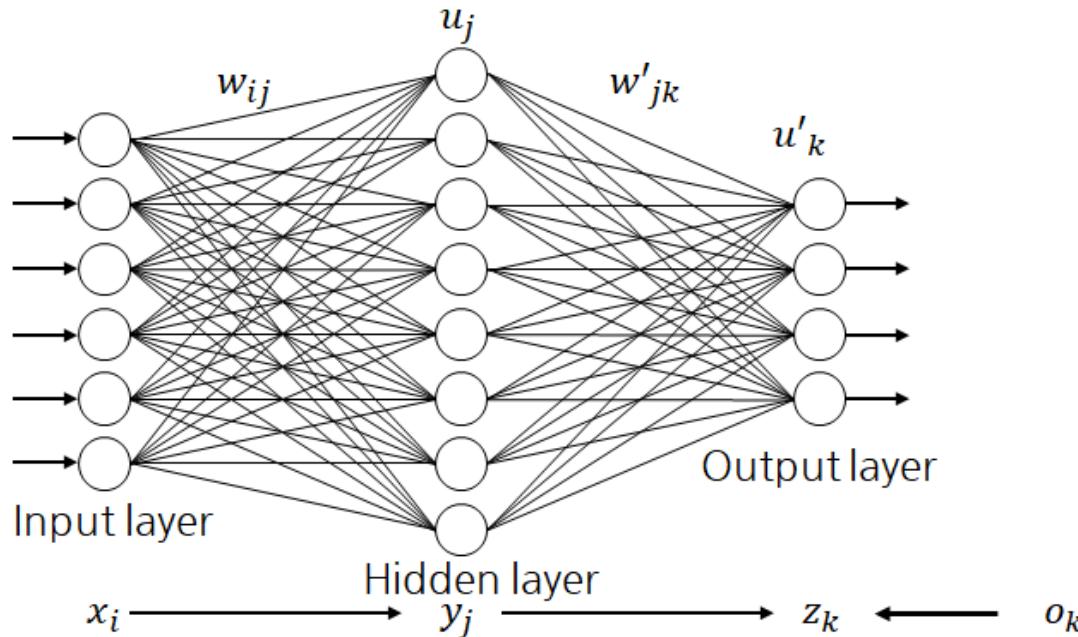


ReLU



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.)

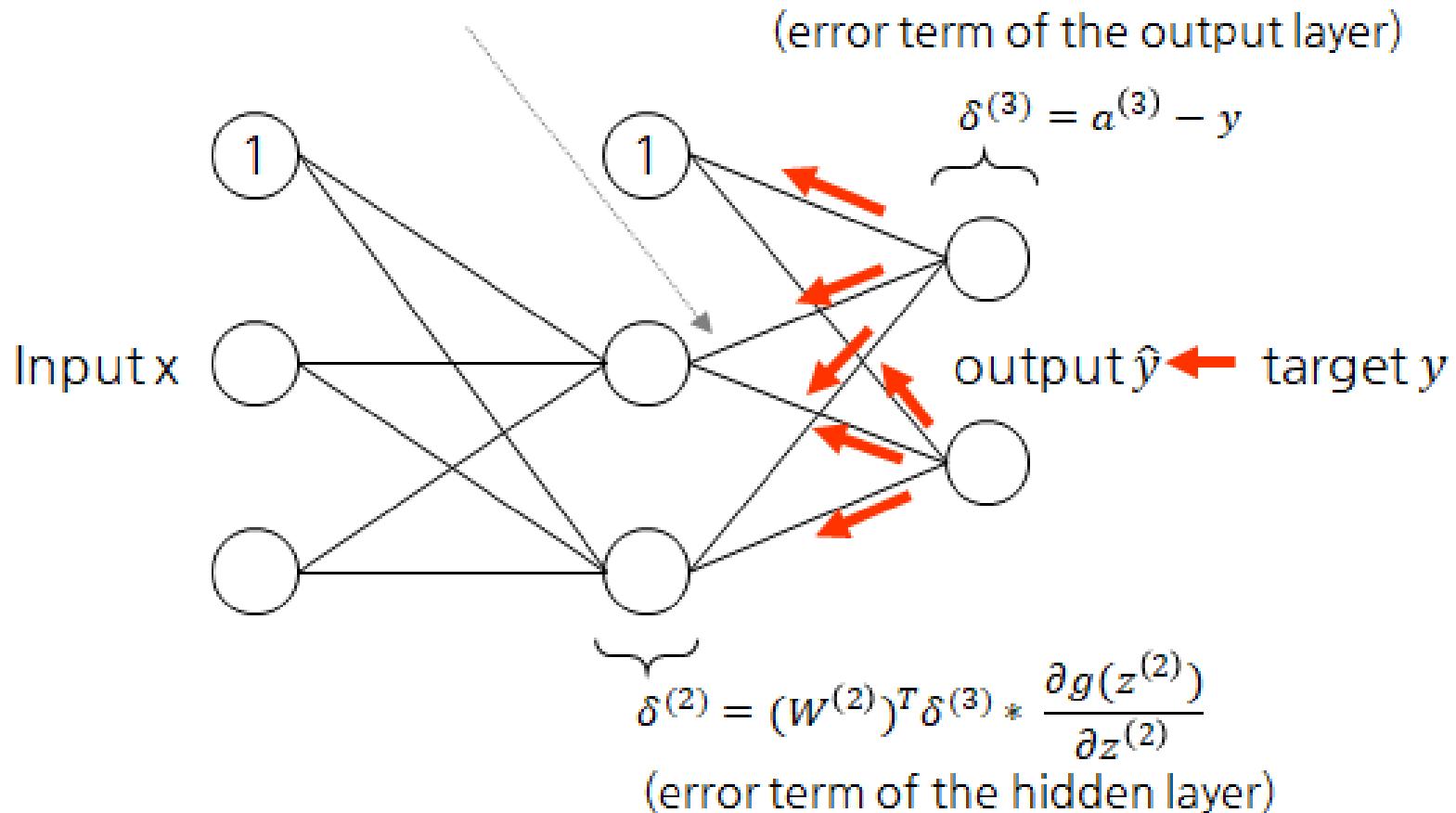
■ 신경망 알고리즘 종류

- 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

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

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 (Cont.)

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

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

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

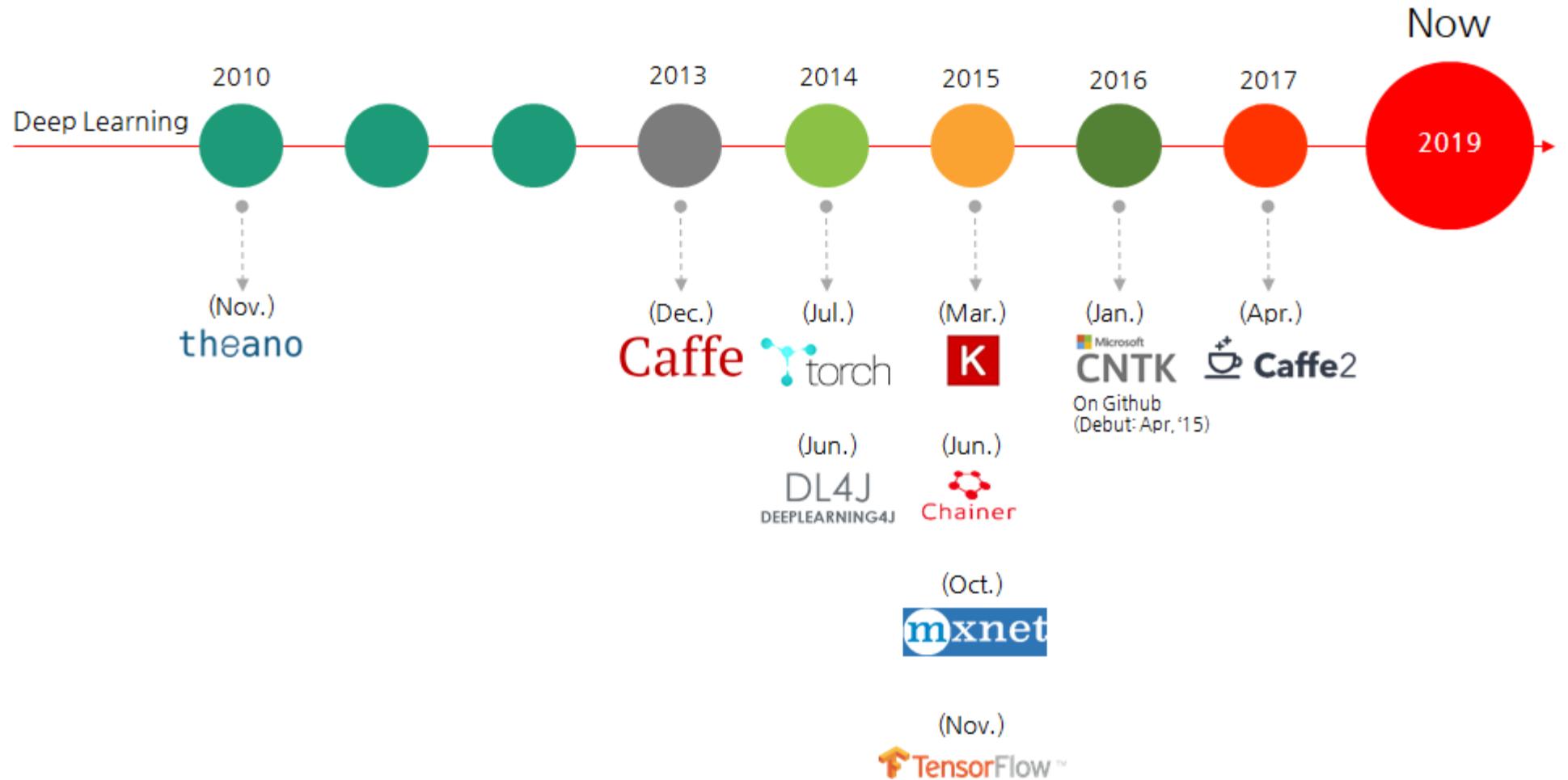
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(j)}} J(W, b; x, y) = W_{ij}^{(l)} - \alpha \cdot a_j^{(l)} \delta_i^{(l+1)}$$

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

2. 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 사용 할 수 있음
- Python, C++, Java, Go 등 언어를 지원하며 Deep Learning을 직접 작성하거나 Keras와 같은 Wrapper Library를 사용하여 직접 작성 할 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 로드 할 수도 있음
- Cluster에 저장된 대규모 Dataset에 Deep Learning 기능을 사용하려는 경우 매우 유용한 Library

3. 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

Run: cifar-train

Upload: Choose File

Color: Structure

색상: 동일한 서브스트럭처는 같은 색상으로 표시
그리고 고유한 서브스트럭처는 회색으로 표시

Main Graph

Auxiliary nodes

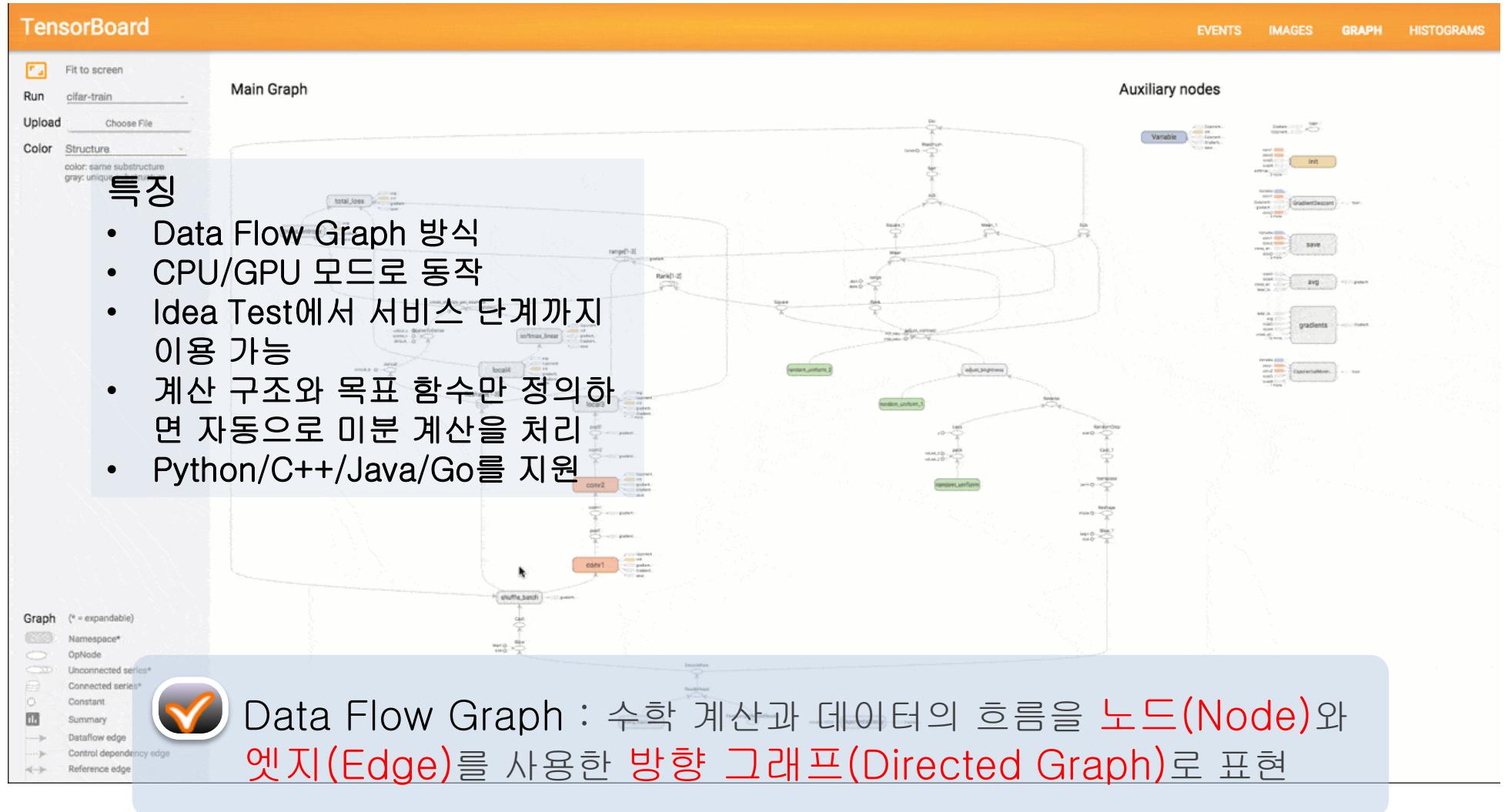
특징

- 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

Graph (* = expandable)

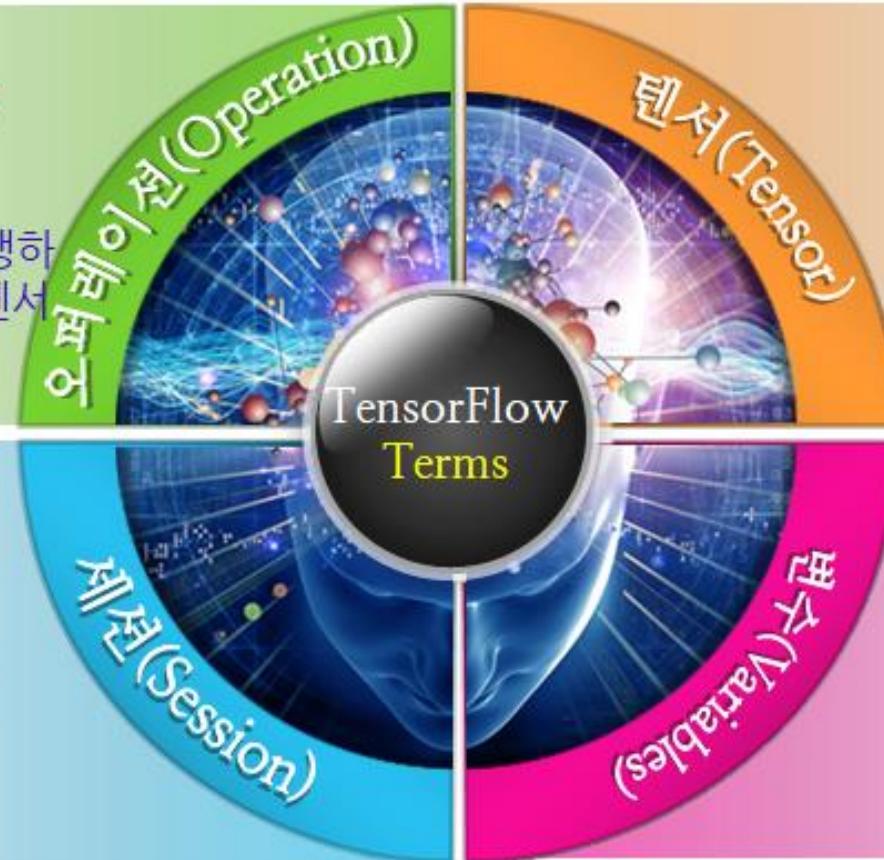
- Namespace*
- OpNode
- Unconnected series*
- Connected series*
- Constant
- Summary
- Dataflow edge
- Control dependency edge
- Reference edge

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



TensorFlow Term

- 그래프 상의 노드(Node)
- 오퍼레이션은 하나 이상의 텐서를 받을 수 있음
- 오퍼레이션은 계산을 수행하고, 결과를 하나 이상의 텐서로 반환할 수 있음



- 오퍼레이션의 실행 환경을 캡슐화한 것
- 그래프를 실행하기 위해서는 세션 객체가 필요

- 내부적으로 모든 데이터는 텐서를 통해 표현
- n-차원 배열 또는 리스트
- 그래프 내의 오퍼레이션 간에는 텐서만이 전달

- 변수는 그래프의 실행 시 파라미터를 저장하고 갱신하는데 사용
- 메모리 상에서 텐서를 저장하는 버퍼 역할

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

AlexNet을 이용한 Framework 속도 비교

AlexNet (One Weird Trick paper) - Input 128x3x224x224

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

4. Deep Learning 이해하기

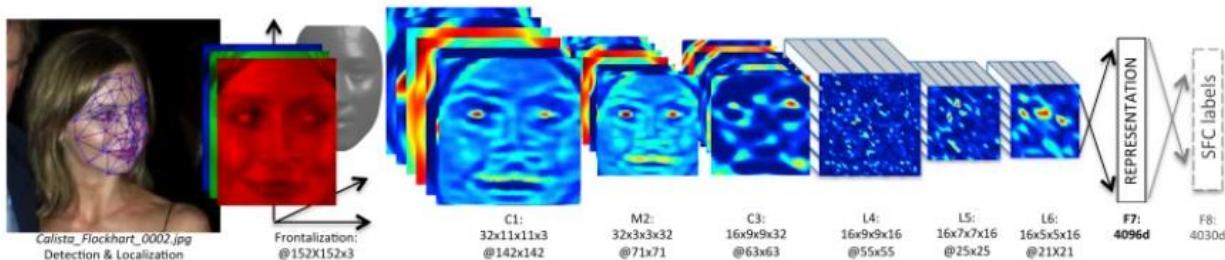


Deep Learning 정의

■ 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)



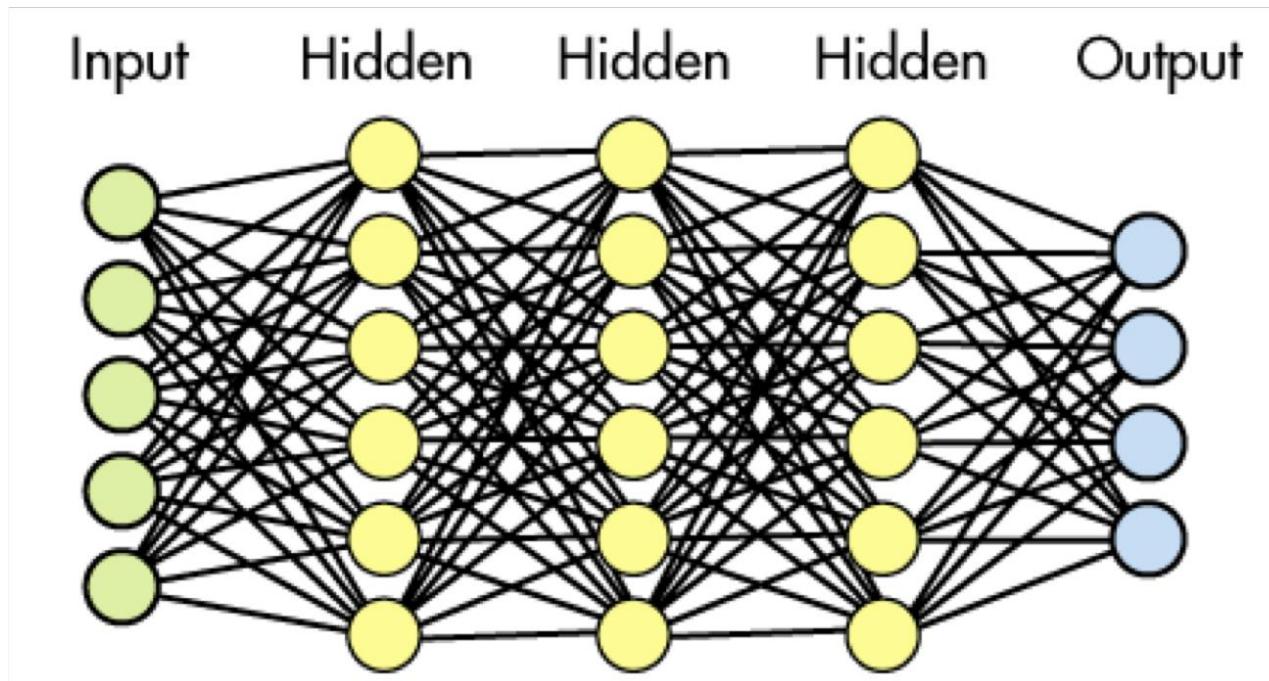
Prediction
(who is it?)

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

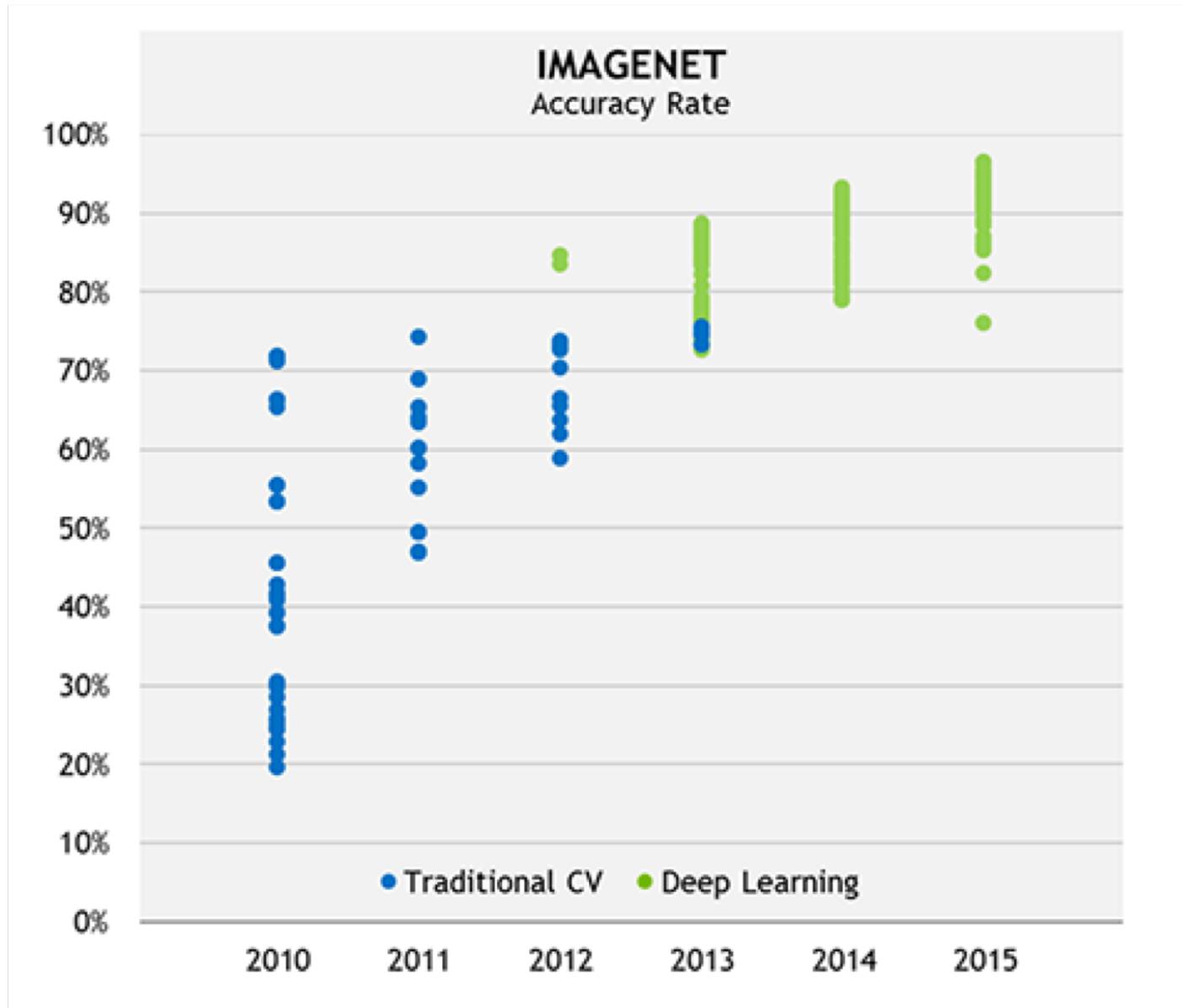
Deep Learning은 Data를 학습시켜 Model을 만드는 Machine Learning Algorithm의 집합

DL(Deep Learning)

- Neural Network을 사용한 Machine Learning의 한 방법
- 신경망의 은닉층이 많아서(Deep) Deep Learning이라고 부른다.

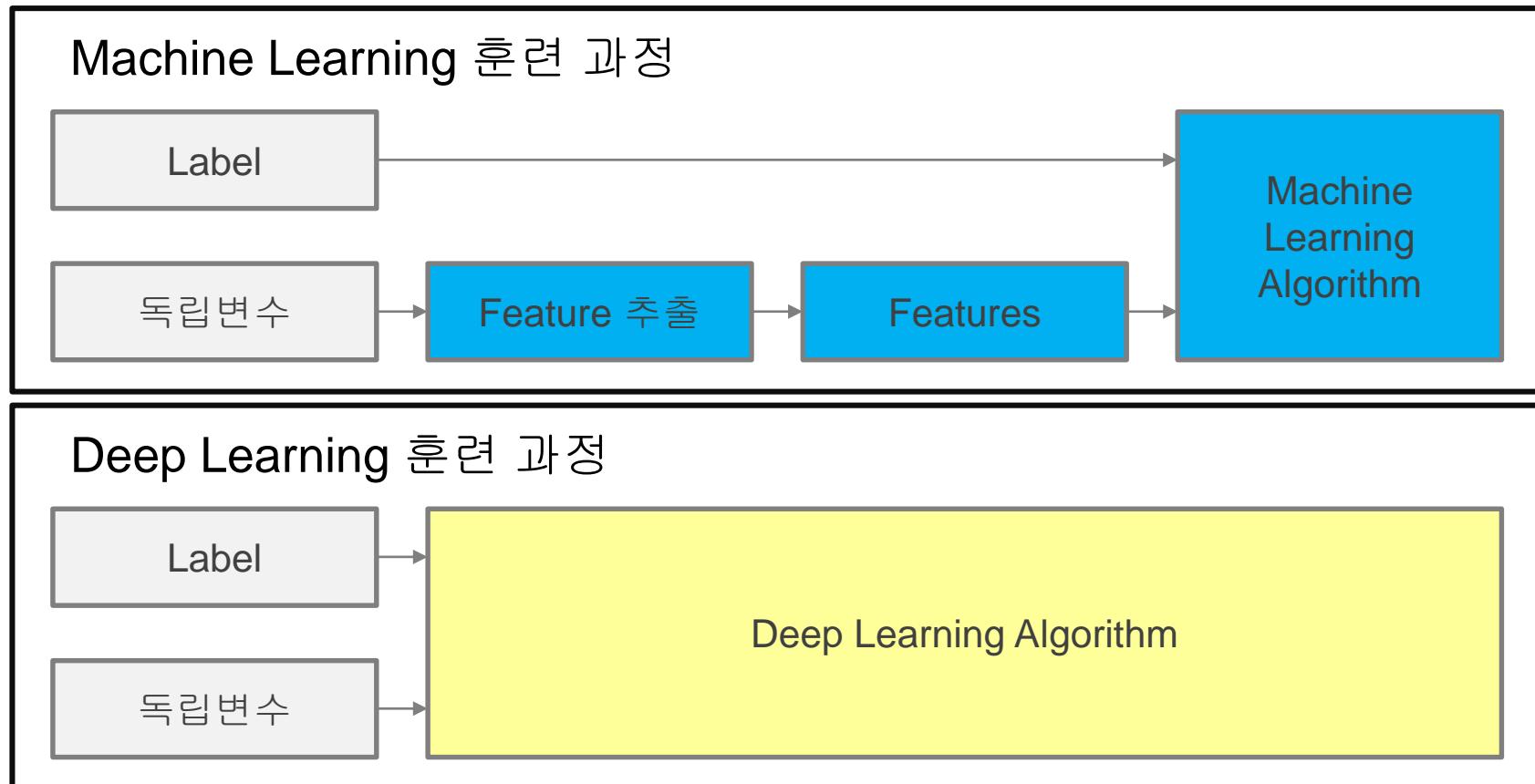


2012년 AI 부활



Machine Learning vs Deep Learning

- Deep Learning은 학습데이터에서 주요 Feature를 추출/선택하는 과정까지도 학습



DL Terminologies

■ Cost Function

- MSE(Mean Squared Error)
 - 각 target값과 계산값과의 차이를 최소로 한다.
- 두 데이터 집합 간의 분포의 차이를 볼 때는
 - CE(Cross Entropy)
 - KL-Divergence
- MLE(Maximum Likelihood Estimation)

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

DL Terminologies (Cont.)

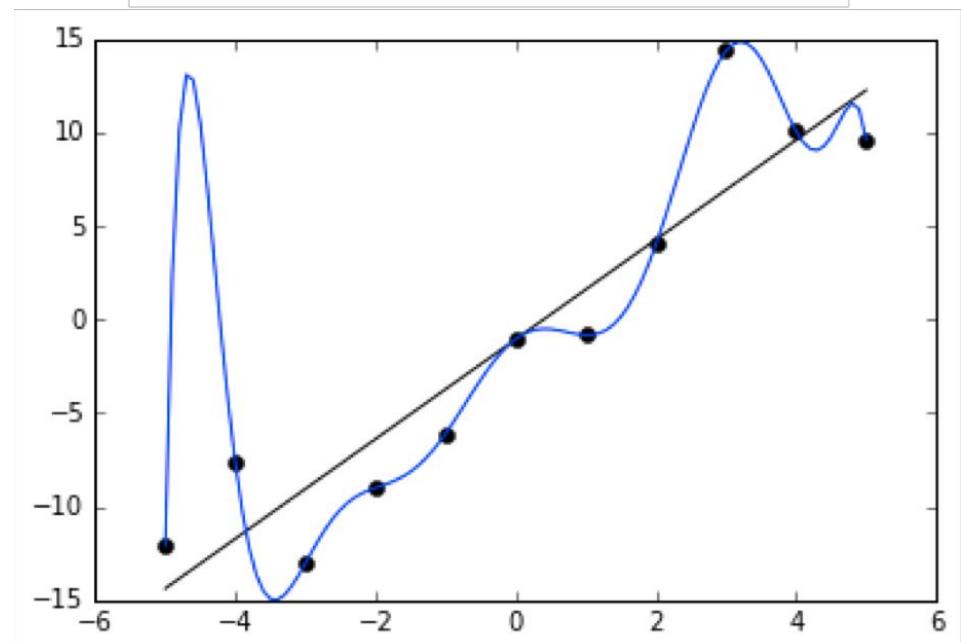
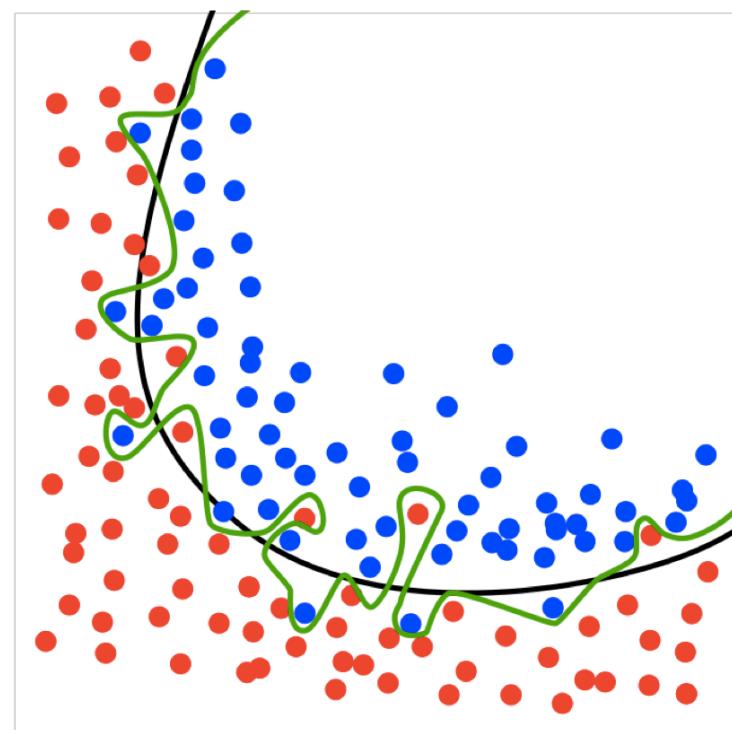
■ Optimizer

- 오차에 대하여 w 를 Update 시키는 Algorithm
- GD(Gradient Descent)
- Batch GD
- Mini-Batch GD
- SGD(Stochastic GD)
- Momentum
- AdaGrad
- Adam
- RMSprop

DL Terminologies (Cont.)

■ Overfitting 방지 법

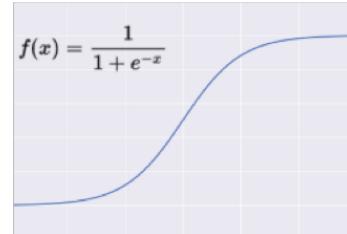
- DropOut
- BN(Batch Normalization)
- Regularization
- Data Augmentation



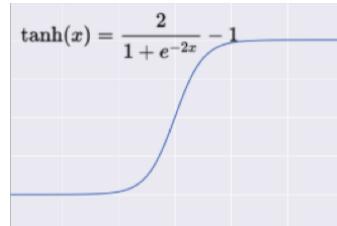
DL Terminologies (Cont.)

■ Activation Function

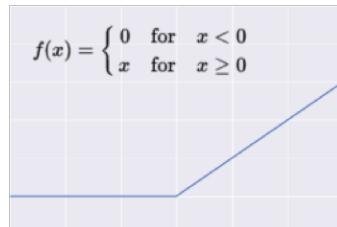
- sigmoid(= logistics)



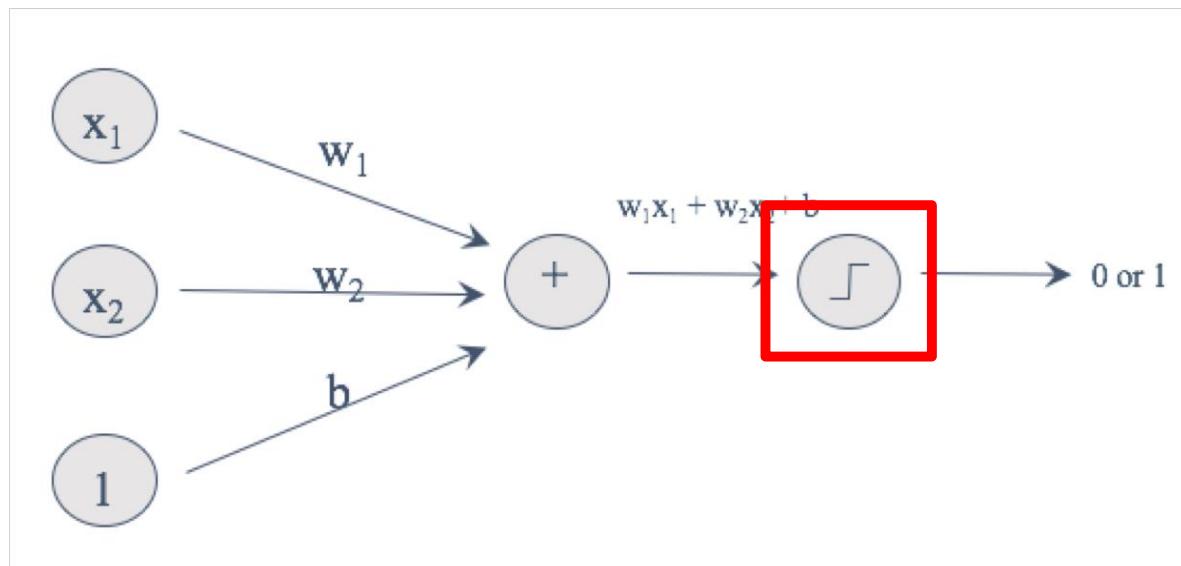
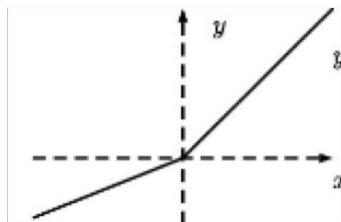
- Tanh



- ReLU



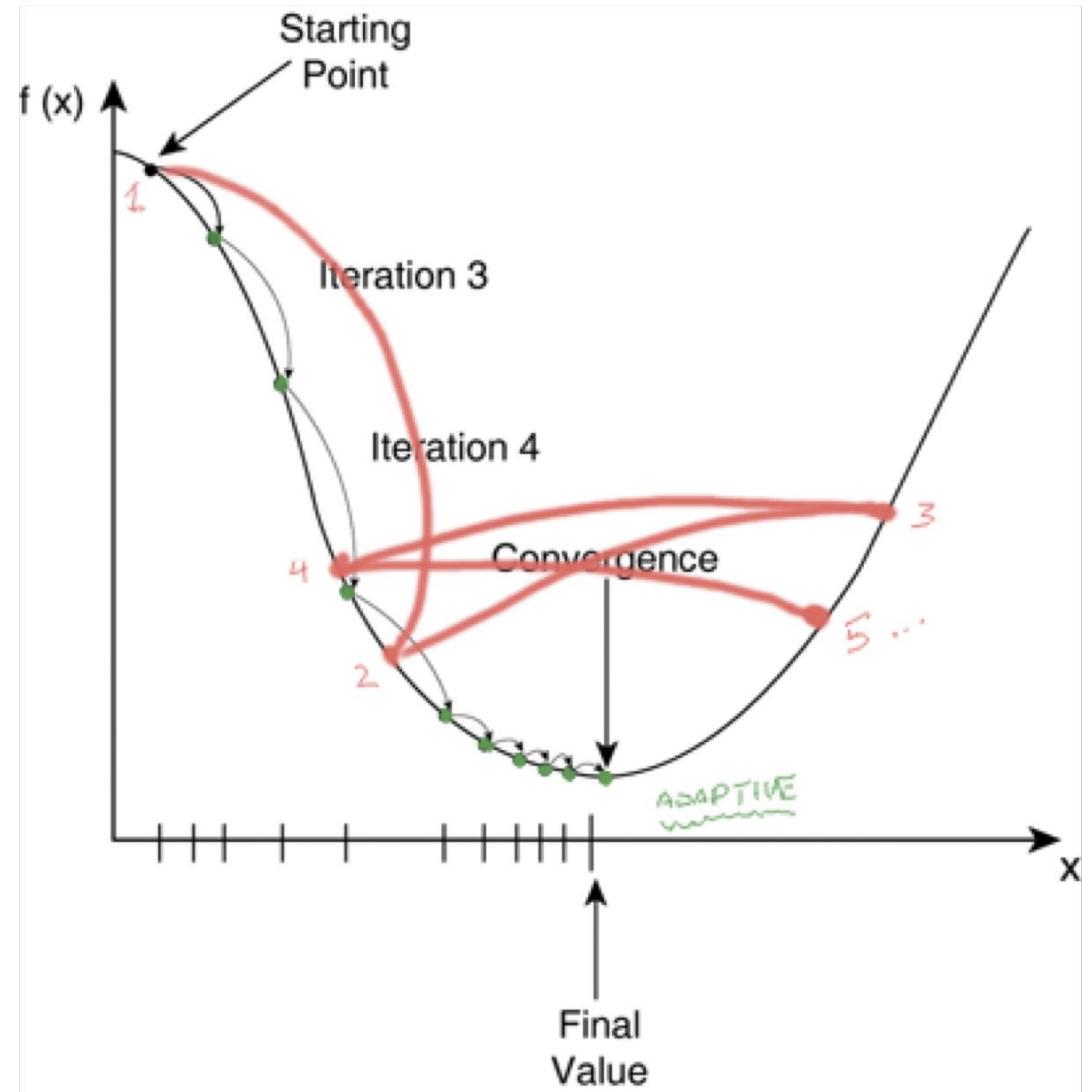
- Leaky ReLU



DL Terminologies (Cont.)

■ Learning Rate

- w 의 변경되는 정도



DL Terminologies (Cont.)

■ Softmax

- Activation Function 중의 하나
- 최종 출력층에 사용되며, 여러 개의 출력 Node의 합 중에 비중의 값으로 나타낸다.
- 확률처럼 표현됨.

DL Terminologies (Cont.)

■ BackPropagation

- 출력된 값과 원하는 값과의 차이를 가지고 그 전의 w값들을 변경하는 Algorithm
- 뒤에서부터 그 오차의 값이 전파된다는 이름.
- 실제 변경되는 값의 크기는 GD로 결정됨.

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()
```

날짜	시간	cost	W	b
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

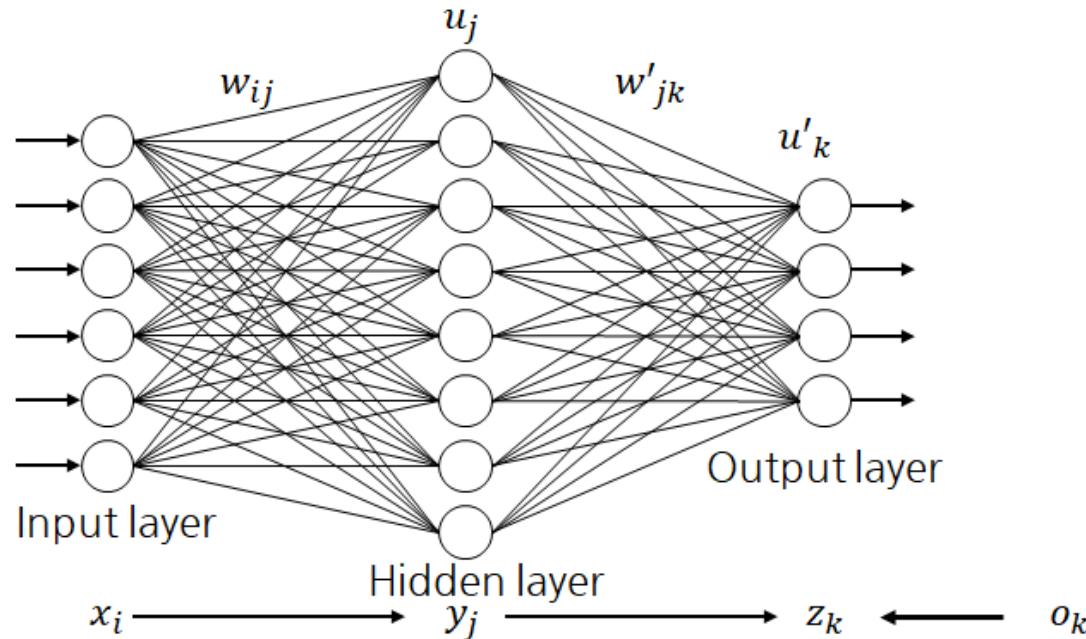


1. DNN 이해



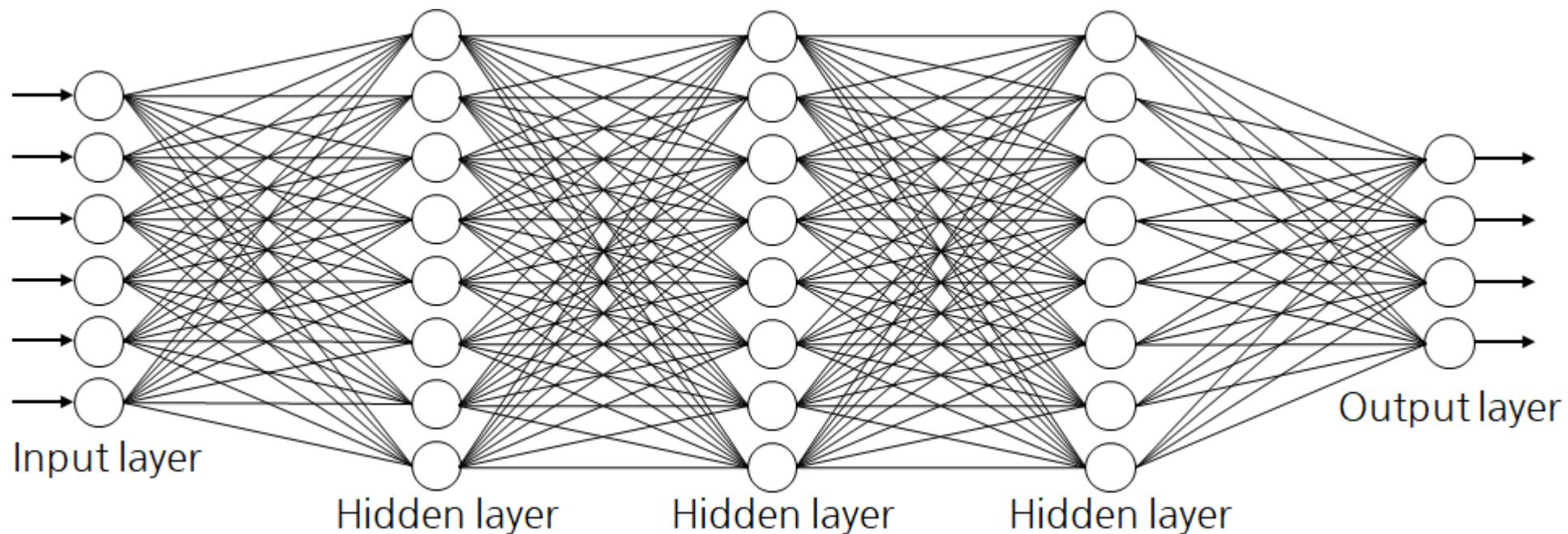
DNN 구조

- 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 나눠져 있음



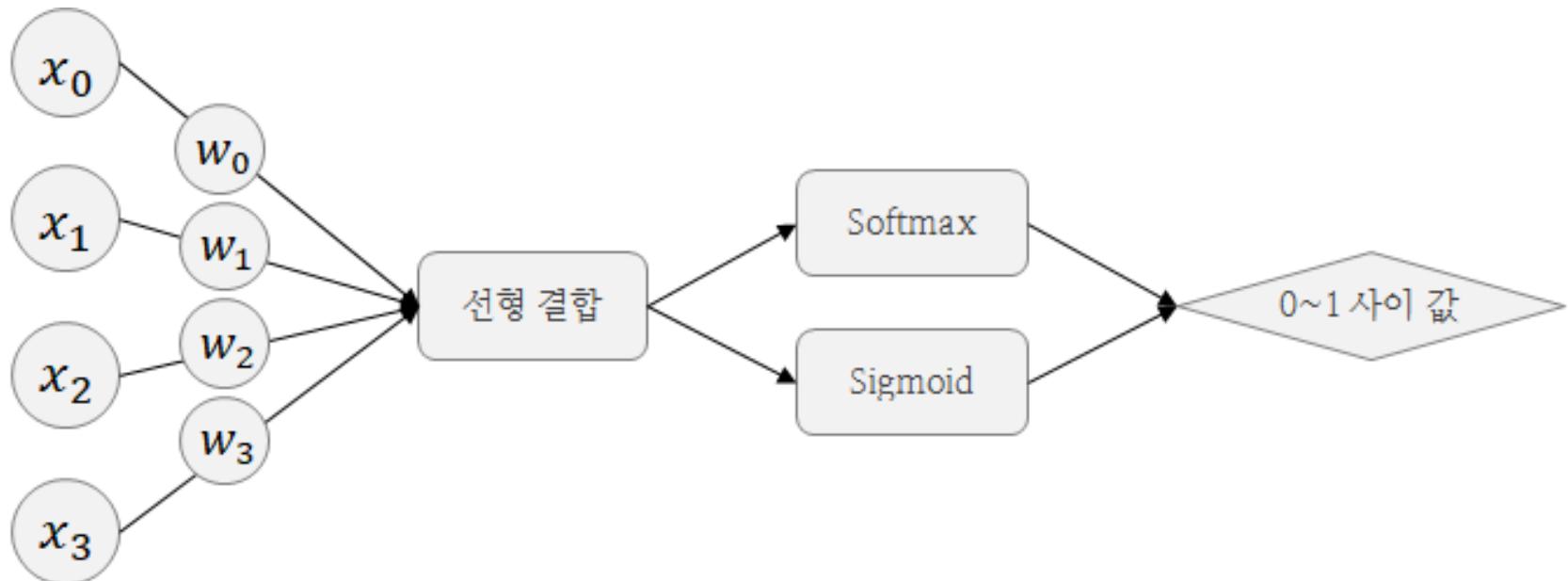
은닉 계층이 3개인 인공신경망

- 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)

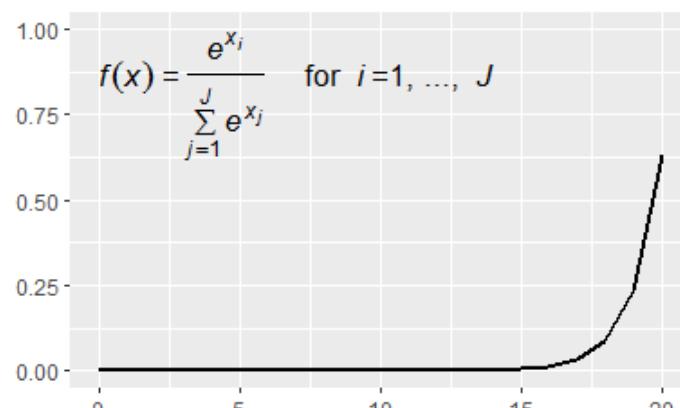
- 입력 값들은 가중치(Weight)와의 선형 결합과 활성화 함수(Activation Function)를 통해 출력 값으로 변환



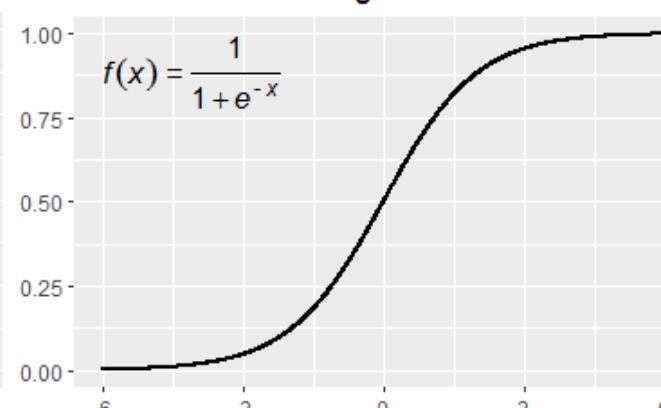
활성화 함수(Activation Function) (Cont.)

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

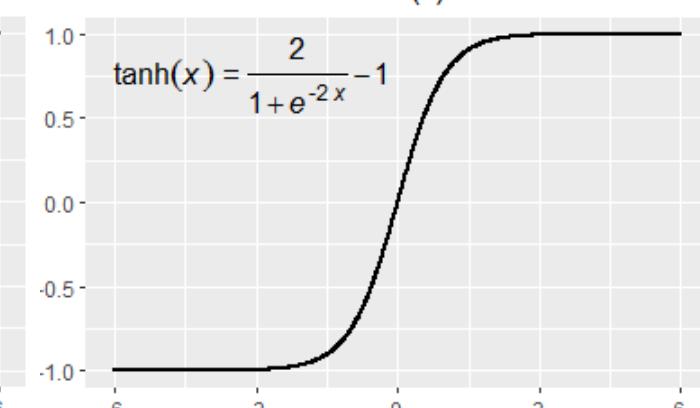
Softmax



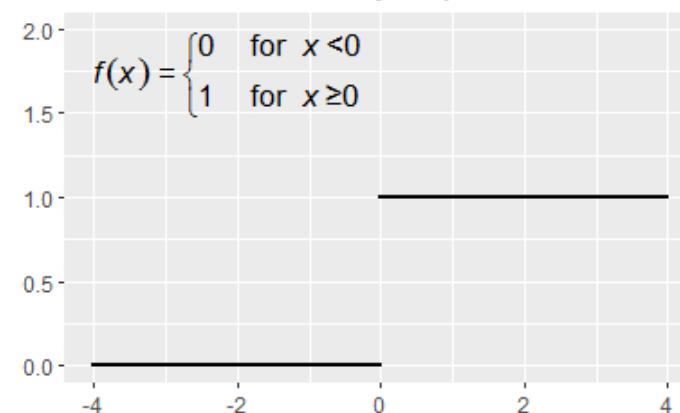
Sigmoid



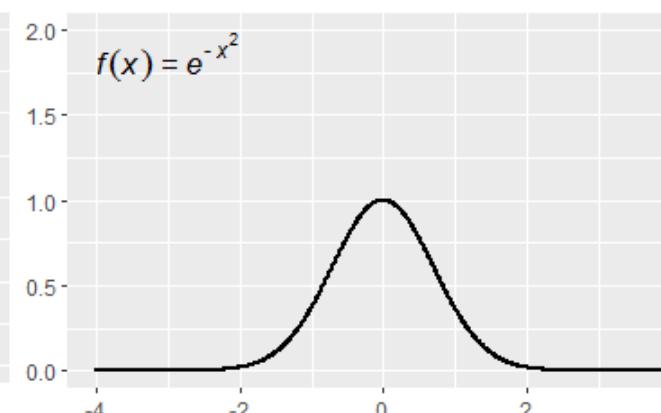
tanh(x)



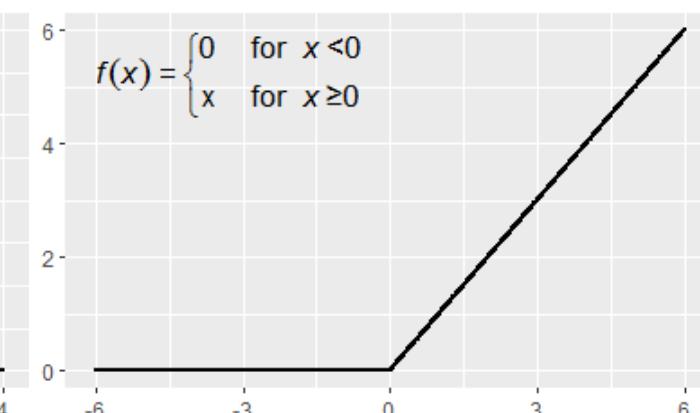
Binary step



Gaussian

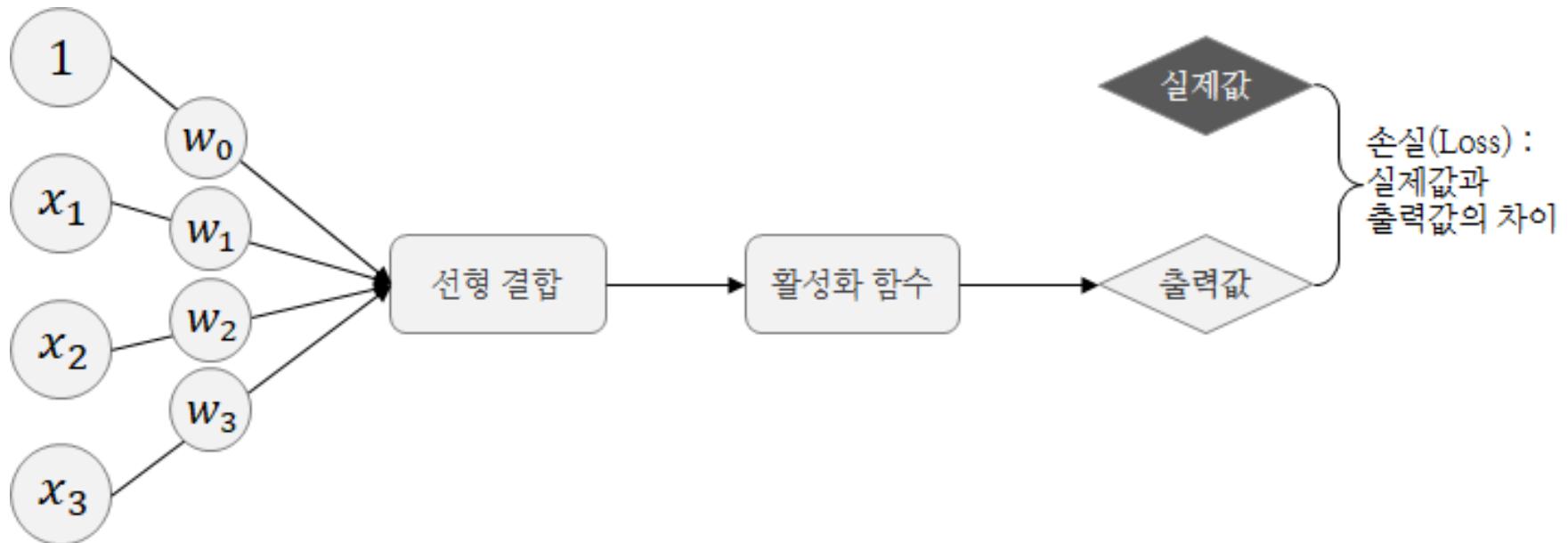


ReLU



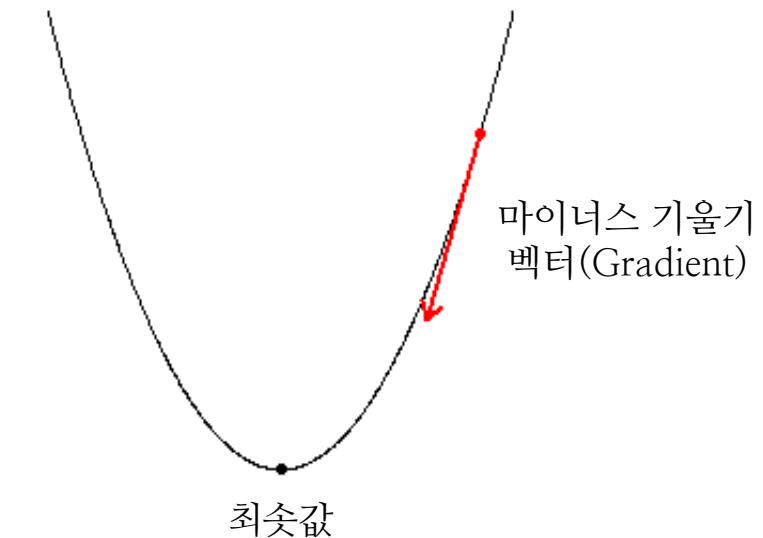
손실함수

- 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



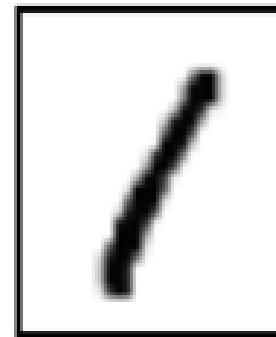
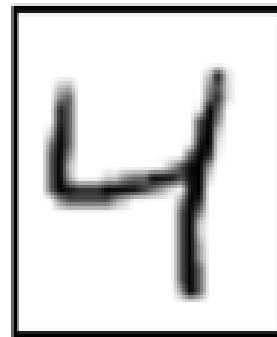
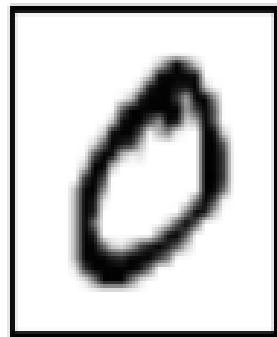
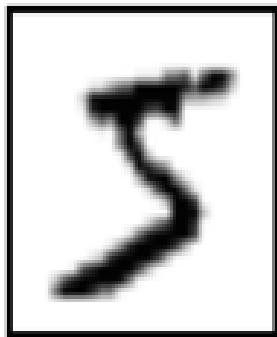


2. MNIST Data로 우편번호 손글씨 자동분류기 만들기



우편번호 자동 분류기

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

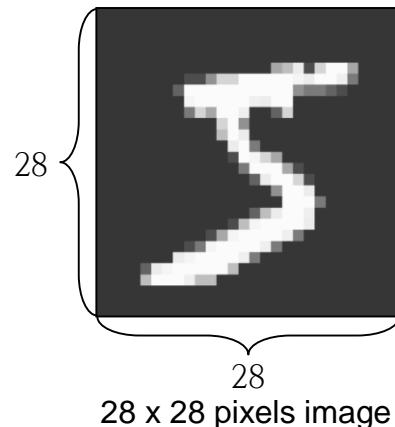


우편번호 자동 분류기 (Cont.)

- 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 읽어 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 기울이도록 다른 무언가의 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

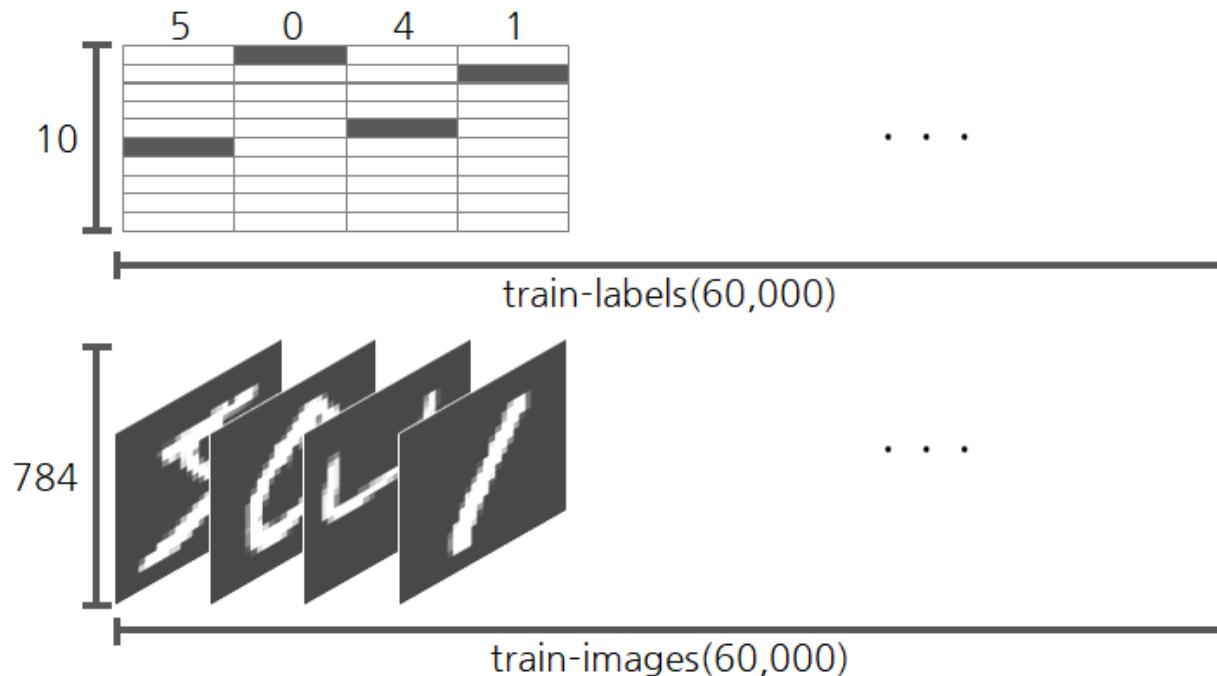


레이블과 화소의 값들

이미지를 784개 화소(Pixel)의 숫자로 표현

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

Extracting MNIST_data/train-images-idx3-ubyte.gz

Successfully downloaded train-labels-idx1-ubyte.gz

Extracting MNIST_data/train-labels-idx1-ubyte.gz

Successfully downloaded t10k-images-idx3-ubyte.gz

실행 시 출력되는 경고는 무시할 것.

Successfully downloaded t10k-labels-idx1-ubyte.gz

Extracting MNIST_data/t10k-labels-idx1-ubyte.gz

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)
```

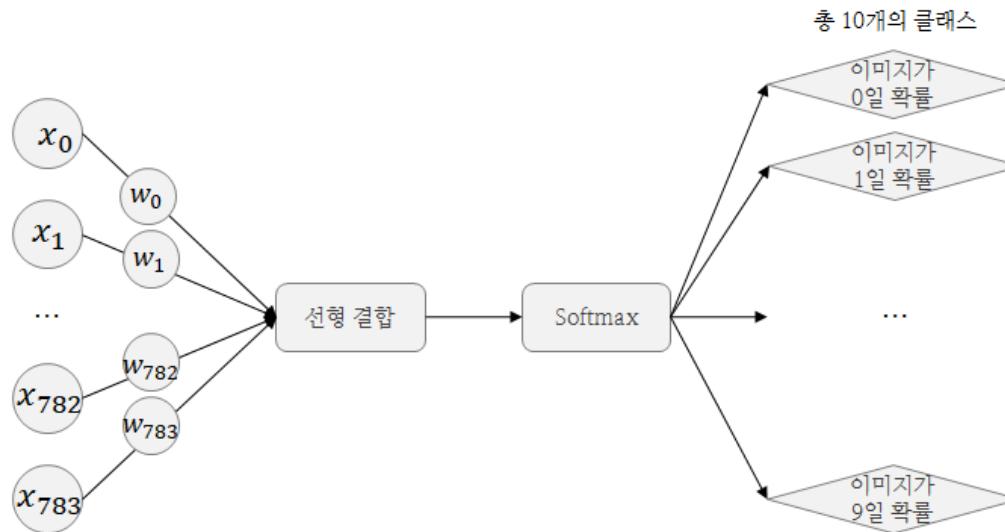


3. 단일 계층을 이용한 Modeling



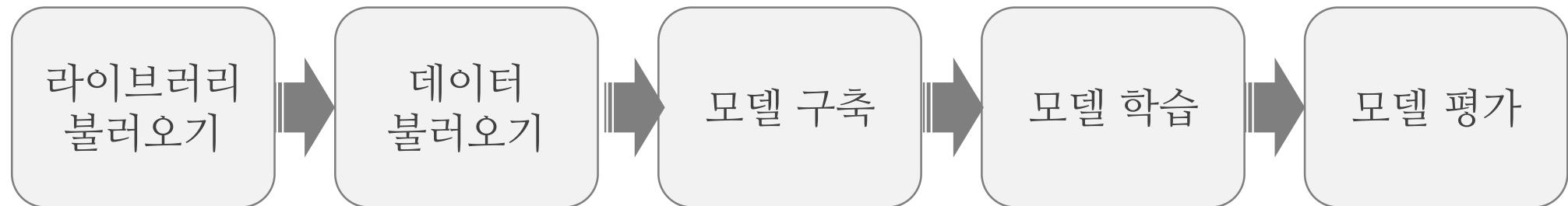
단일 계층 모델링 구성도

- 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 10 개의 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,
 - None이라고 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 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 함
- 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 최소화하도록 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.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  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]



4. 단일 계층을 이용한 보험 사기꾼 분류



실습

- 보험 사기꾼 찾기
 - https://github.com/swacademy/NLP-Lab/tree/master/Chapter8/data_cust.zip
- 학습용 DataSet : data_cust_2-3_train.csv
- 테스트용 DataSet : data_cust_2-3_test.csv
- 파일의 첫 번째 열(CUST_ID)은 사원의 번호이며, 두 번째 열(SIU_CUST_YN)은 사기자(1) 또는 정상인(0) 여부를 저장한 열
- 그 외의 열은 고객의 정보들을 포함하고 있음

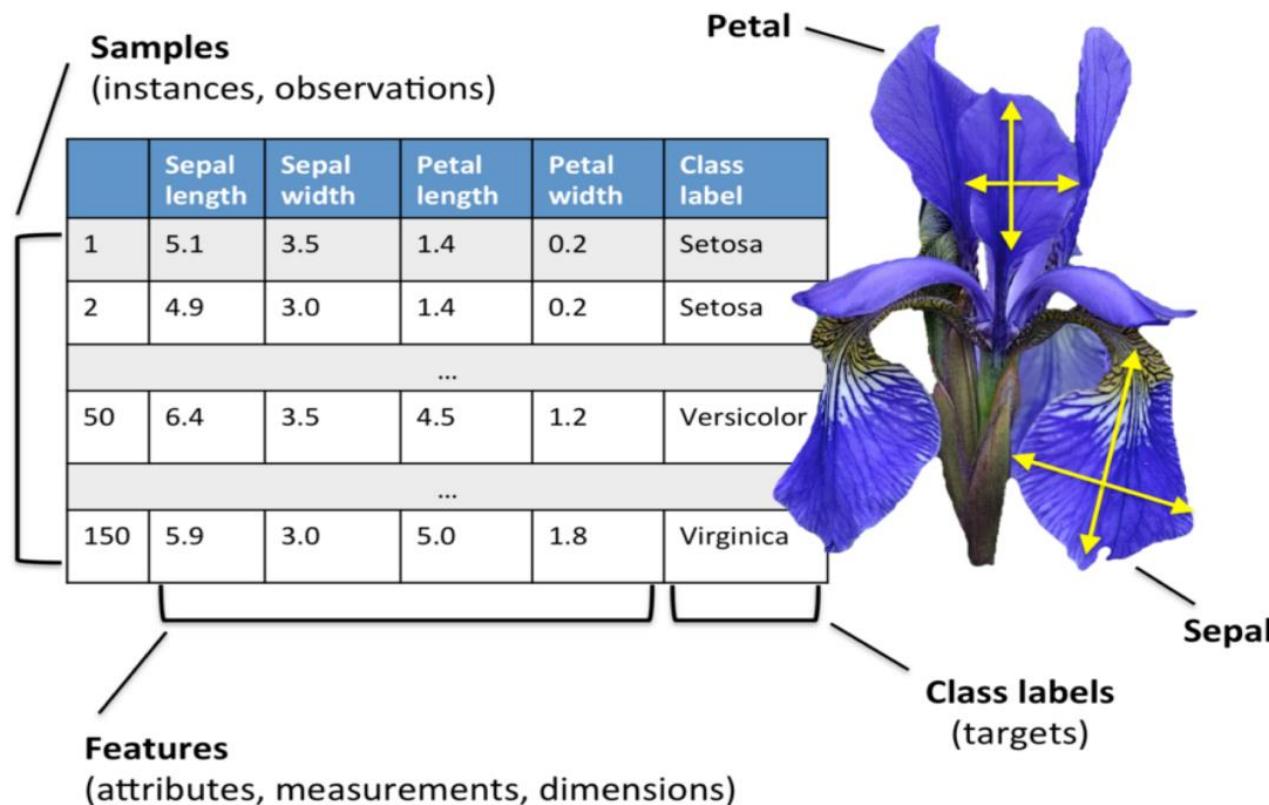


5. 단일 계층을 이용한 iris종 분류



Iris Data

- 3가지 Species
- 꽃받침과 꽃잎의 폭과 길이로 구분
- 입력 Vector는 4차원



Iris Data 가져오기

```
1 # define library & load data set
2 import tensorflow as tf
3 import numpy as np
4 import pandas as pd
5 from sklearn import datasets
6
7 def get_one_hot(targets, nb_classes):
8     res = np.eye(nb_classes)[np.array(targets).reshape(-1)]
9     return res.reshape(list(targets.shape)+[nb_classes])
10
11 iris = datasets.load_iris()
12 train_x = iris.data
13 train_y = get_one_hot(iris.target, 3)
14
15 print(train_y[0:10])
```

Model 만들기

```
1 # define data & label
2 x = tf.placeholder(tf.float32, [None, 4])
3 y = tf.placeholder(tf.float32, [None, 3])
4
5 # define Weight & Bias
6 W = tf.Variable(tf.zeros([4, 3]))
7 b = tf.Variable(tf.zeros([3]))
8
9 # define output
10 # activation function is softmax
11 h = tf.nn.softmax(tf.matmul(x, W)+b)
12
13 # define Cross Entropy function
14 cross_entropy = -tf.reduce_sum(y * tf.log(h), reduction_indices=[1])
15
16 # define loss function
17 loss = tf.reduce_mean(cross_entropy)
18
19 # define train step, use gradient descent, set running rate ....
20 train = tf.train.GradientDescentOptimizer(0.001).minimize(loss)
```

Model 학습하기

```
1 # initialize variable
2 init = tf.global_variables_initializer()
3
4 # start session
5 sess = tf.Session()
6 sess.run(init)
7
8 # Training
9 for i in range(100001):
10     _, loss_value = sess.run([train, loss], feed_dict={x: train_x, y: train_y})
11     if i % 10000 == 0:
12         print('Step %d, %.5f' % (i, loss_value))
```

Step 0, 1.09861
Step 10000, 0.35752
Step 20000, 0.27034
Step 30000, 0.22396
Step 40000, 0.19488
Step 50000, 0.17486

Model 평가하기

```
1 # model assess with Accuracy  
2 correct_prediction = tf.equal(tf.argmax(h, 1), tf.argmax(y, 1))  
3 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))  
4 print(sess.run(accuracy, feed_dict={x:train_x, y:train_y}))
```

0.9866667

```
1 predict_y = sess.run(tf.argmax(h, 1), feed_dict={x:train_x, y:train_y})  
2 cross_tab = pd.crosstab(train_y.argmax(axis=1), predict_y, margins=True)  
3 print(cross_tab)
```

col_0	0	1	2	All
row_0				
0	50	0	0	50
1	0	48	2	50
2	0	0	50	50
All	50	48	52	150

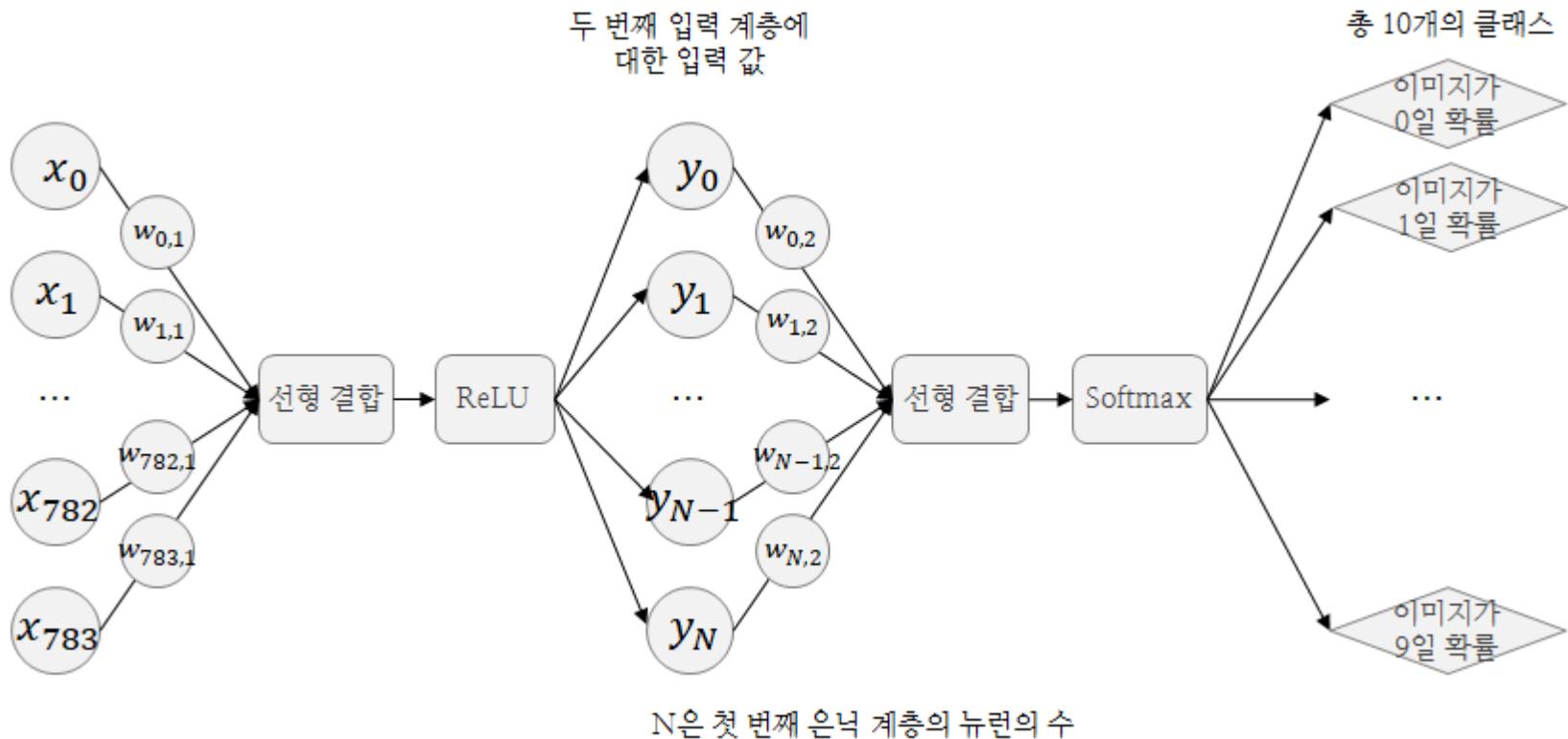


6. 복합 계층 DNN을 이용한 Modeling



복합 계층 모델링 구성도

- 복합 계층(Multi-Layer)은 두 개 이상의 은닉 계층이 들어간 것
- 복합 계층과 단일 계층의 차이점은 은닉 계층이 더 들어간 것



복합 계층 DNN 코드와 실행

```
1 W_1 = tf.Variable(tf.truncated_normal([784, 360], stddev=0.1))  
2 b_1 = tf.Variable(tf.constant(0.1, shape=[360]))
```

```
1 h_1 = tf.nn.relu(tf.matmul(x, W_1) + b_1)
```

```
1 W_2 = tf.Variable(tf.truncated_normal([360, 10], stddev=0.1))  
2 b_2 = tf.Variable(tf.constant(0.1, shape=[10]))
```

```
1 h = tf.nn.softmax(tf.matmul(h_1, W_2) + b_2)
```

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

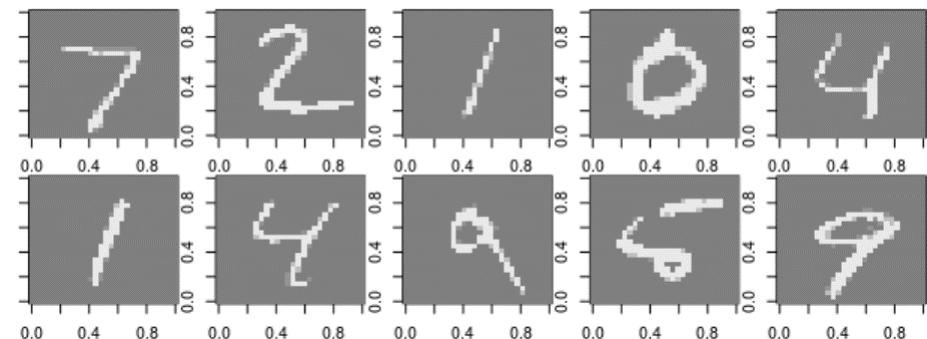
예측 값 출력

```
1 print(sess.run(tf.argmax(y, 1),  
2                 feed_dict={x: mnist.test.images[0:10],  
3                               y: mnist.test.labels[0:10]}))
```

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

```
1 print(mnist.test.labels[0:10])
```

```
[[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.]]
```



RNN

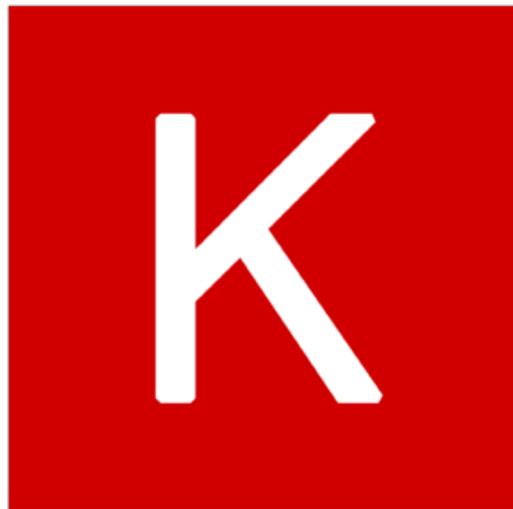


1. Keras



Keras

- User가 손쉽게 Deep Learning을 구현할 수 있도록 도와주는 상위 Level의 Interface
- <https://Keara.io>
- Keras is compatible with Python **2.7 ~ 3.6**



Keras

Sequential을 이용한 Keras Model

■ Sequential Model

```
from keras.models import Sequential  
model = Sequential()
```

■ add()를 이용한 Layer 추가

```
from keras.layers import Dense  
model.add(Dense(units=64, activation='relu', \  
                input_dim = 100))  
model.add(Dense(units=10, activation='softmax' ))
```

■ compile()로 학습 process 설정

```
model.compile(loss='categorical_crossentropy', \  
              optimizer='sgd', metrics=['accuracy'])
```

Sequential을 이용한 Keras Model (Cont.)

- 앞의 코드는 다음과 같음

```
model.compile(loss=keras.losses.categorical_crossentropy,  
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9,\n              nesterov=True))
```

- Model 학습

```
# x_train and y_train are Numpy arrays -just like in the  
Scikit-learn API.  
  
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

- 평가

```
loss_and_metrics = model.evaluate(x_test, y_test, \n                                  batch_size=128)
```

- 예측

```
classes = model.predict(x_test, batch_size=128)
```

Sequential API로 만든 Model

■ 예)

```
from keras.models import Sequential  
from keras.layers import Dense  
model = Sequential()  
model.add(Dense(3, input_dim=4, \  
                activation='softmax'))
```

■ 장점

- sequential API로 Model을 설계하는 것은 직관적이고 편리함

■ 단점

- 단순히 층을 쌓는 것만으로는 구현할 수 없는 인공 신경망은 구현 할 수 없다.
- 즉, 복잡한 인공 신경망을 구현할 수 없다.

Fully-Connected FFNN

- `input()`에 입력의 크기 기재

```
from keras.layers import Input  
inputs = Input(shape=(10, )) # tensor return
```

- 은닉 층과 출력 층 추가(이전 층을 다음 층 함수의 입력으로 사용하고, 변수에 할당)

```
from keras.layers import Input, Dense  
inputs = Input(shape=(10, ))  
hidden1 = Dense(64, activation='relu')(inputs)  
hidden2 = Dense(64, activation='relu')(hidden1)  
output = Dense(1, activation='sigmoid')(hidden2)
```

Fully-Connected FFNN (Cont.)

- 코드를 하나의 모델로 구성, Model에 입력 Tensor와 출력 Tensor를 정의하여 완성

```
from keras.layers import Input, Dense  
from keras.models import Model  
inputs = Input(shape=(10, ))  
hidden1 = Dense(64, activation='relu')(inputs)  
hidden2 = Dense(64, activation='relu')(hidden1)  
output = Dense(1, activation='sigmoid')(hidden2)  
model = Model(inputs = inputs, outputs= output)
```

Fully-Connected FFNN (Cont.)

- `model.compile`, `model.fit` 등을 사용

```
model.compile(optimizer='rmsprop', \
    loss='categorical_crossentropy', \
    metrics=['accuracy'])

model.fit(data, labels)
```

- 은닉층과 출력층 변수를 x로 통일

```
inputs = Input(shape=(10,))

x = Dense(8, activation='relu')(inputs)
x = Dense(4, activation='relu')(x)
x = Dense(1, activation='linear')(x)

model = Model(inputs, x)
```

Linear Regression

```
from keras.layers import Input, Dense
from keras.models import Model
inputs = Input(shape=(3,))
output = Dense(1, activation='linear')(inputs)
linear_model = Model(inputs, output)
linear_model.compile(optimizer='sgd', \
                      loss='mse')
linear_model.fit(x=dat_test, y=y_cts_test, \
                  epochs=50, verbose=0)
linear_model.fit(x=dat_test, y=y_cts_test, \
                  epochs=1, verbose=1)
```

Logistic Regression

```
from keras.layers import Input, Dense  
from keras.models import Model  
inputs = Input(shape=(3,))  
output = Dense(1, activation='sigmoid')(inputs)  
logistic_model = Model(inputs, output)  
logistic_model.compile(optimizer='sgd', \  
                      loss='binary_crossentropy' \  
                      metrics=['accuracy'])  
logistic_model.optimizer.lr = 0.001  
linear_model.fit(x=dat_train, \  
                  y=y_classifier_train, epochs=5, \  
                  validation_data=(dat_test, y_classifier_test))
```

다중 입력을 받는 모델(Model that accepts multiple inputs)

```
from keras.layers import Input, Dense, concatenate  
from keras.models import Model
```

#2개의 입력층을 정의

```
inputA = Input(shape=(64, ))  
inputB = Input(shape=(128, ))
```

#첫 번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의

```
x = Dense(16, activation='relu')(inputA)  
x = Dense(8, activation='relu')(x)  
x = Model(inputs=inputA, outputs=x)
```

다중 입력을 받는 모델(Model that accepts multiple inputs) (Cont.)

#두 번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의

```
y = Dense(64, activation='relu')(inputB)
```

```
y = Dense(32, activation='relu')(y)
```

```
y = Dense(8, activation='relu')(y)
```

```
y = Model(inputs=inputB, outputs=y)
```

#2개의 인공 신경망의 출력을 연결(concatenate)

```
result = concatenate([x.output, y.output])
```

다중 입력을 받는 모델(Model that accepts multiple inputs) (Cont.)

#연결된 값을 입력으로 받는 밀집층을 추가(Dense layer)

```
z = Dense(2, activation='rulu')(result)
```

#선형 회귀를 위해 activation=linear를 설정

```
z = Dense(1, activation = 'linear')(z)
```

#결과적으로 이 모델은 2개의 입력층으로부터 분기되어 진행된 후 마지막에는 하나의 출력을 예측하는 모델이 됨.

```
model = Model(inputs=[x.input, y.input], outputs=z)
```

RNN(Recurrence Neural Network) 은닉층 사용하기

```
from keras.layers import Input, Dense  
from keras.layers.recurrent import LSTM  
from keras.models import Model  
  
inputs = Input(shape=(50,1))  
lstm_layer = LSTM(10)(inputs)  
#RNN의 일종인 LSTM사용  
x = Dense(10, activation='relu')(lstm_layer)  
output = Dense(1, activation='sigmoid')(x)  
model = Model(inputs=inputs, outputs=output)
```

동일한 표기법

`encoder = Dense(128)(input)`

- 위의 코드는 아래의 코드와 같음.

`encoder = Dense(128)`

`encoder(input)`

2. RNN

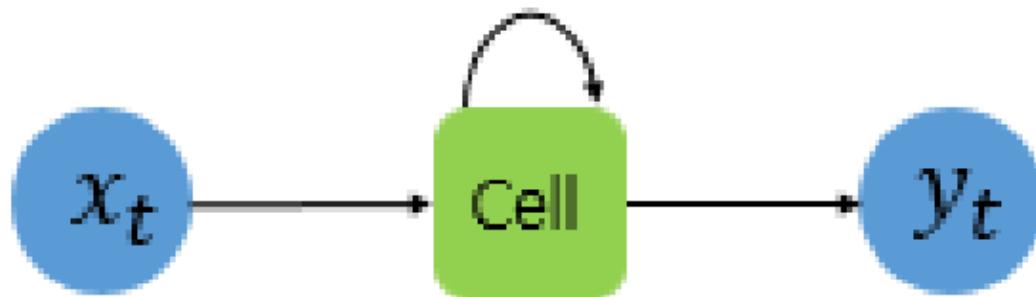


순환신경망(RNN)

- RNN(Recurrent Neural Network)은 Sequence Model 임.
 - 입력과 출력을 Sequence로 처리하는 Model
 - 번역기를 생각해보면 입력은 번역하고자 하는 문장. 즉 Sequence이다.
 - 출력에 해당되는 번역된 문장 또한 Sequence이다.
 - Sequence Model들을 처리하기 위해 고안된 Model들을 Sequence Model이라고 함.
 - RNN은 Sequence Model의 가장 대표적이고 기본적인 Model

순환신경망(RNN) (Cont.)

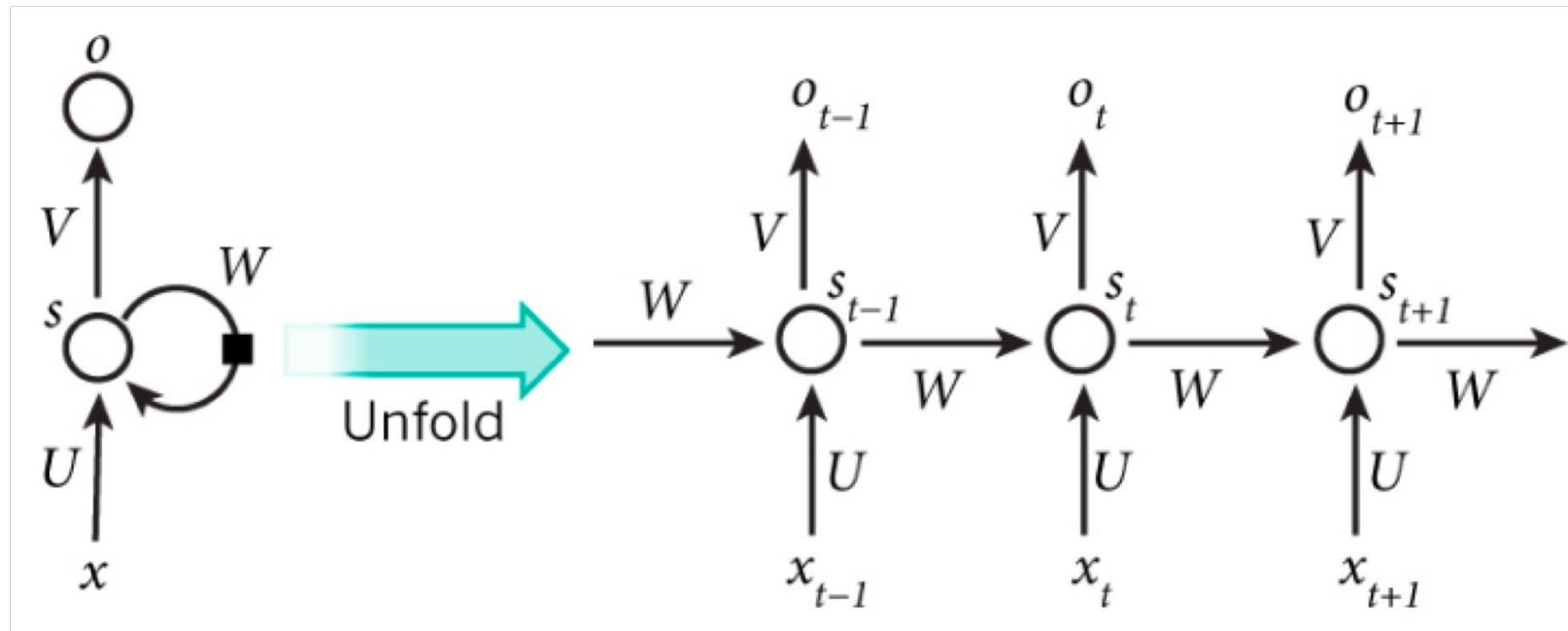
- RNN은 은닉층에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내면서, 다시 은닉층의 다음 계산의 입력으로 보내는 특징을 갖고 있음.



편향 b 도 은닉층과 출력층의 입력으로 존재 하지만 앞으로의 신경망의 그림에서 편향 b 는 생략함.

순환신경망(RNN) (Cont.)

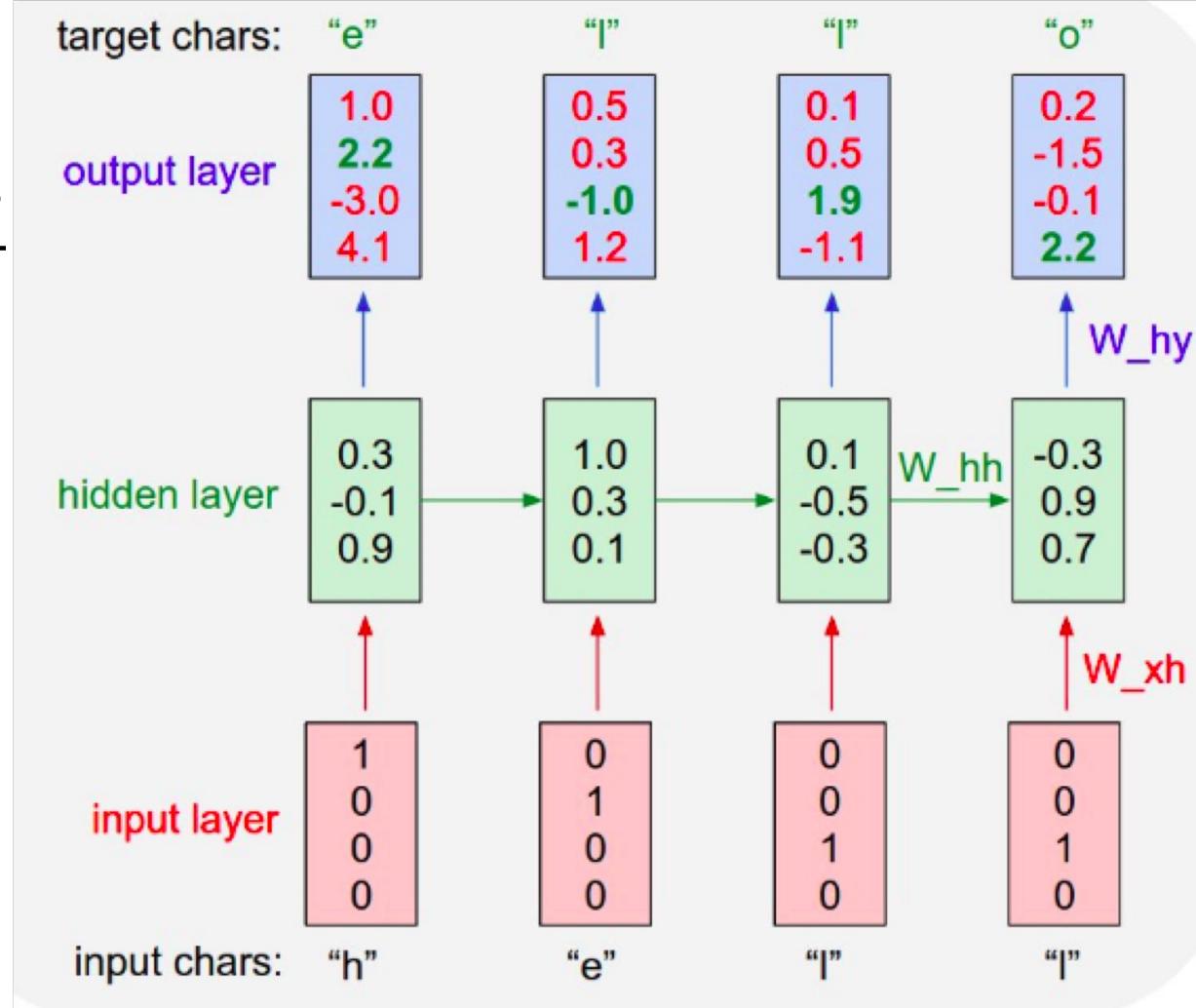
- 상태를 가지고 있고, 다음 연산 때 그 상태가 입력으로 사용된다.



Refer to <http://aikorea.org/blog/rnn-tutorial-1/>

입출력 예

- [h,e,l,l,o]의 학습 예
- 첫 번째 l일 때와 두 번째 l일 때의 출력이 다르다.

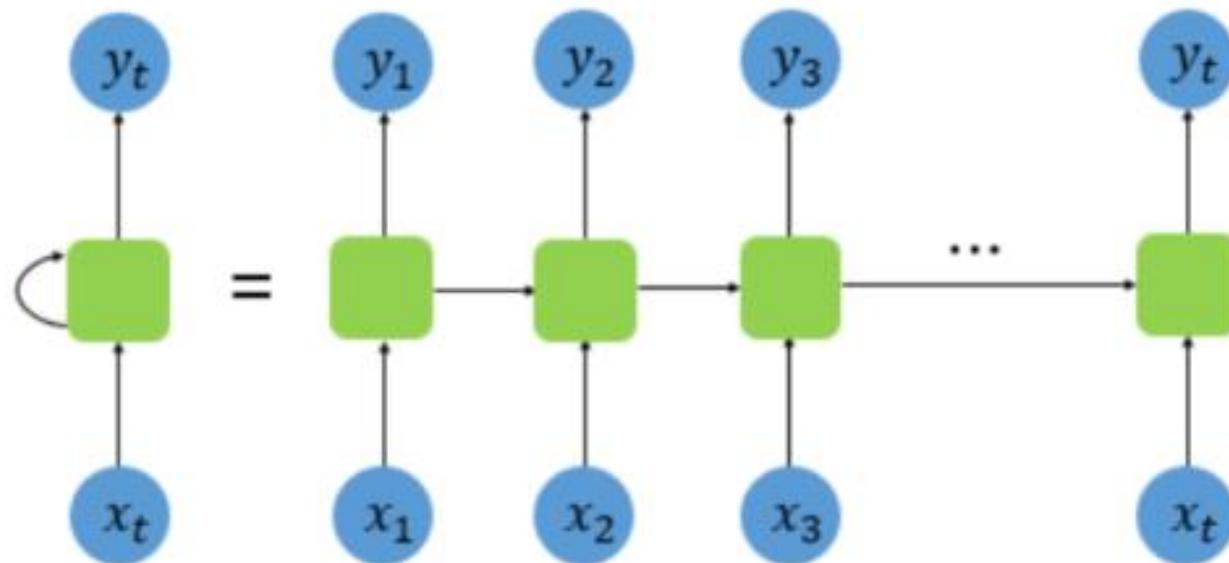


은닉 상태

- 은닉층의 Memory Cell은 각각의 시점(time-step)에서 바로 이전 시점에서의 은닉층의 Memory Cell에서 나온 값을 계속해서 자신의 입력으로 보내는 재귀적 활동을 하고 있음.
 - 현재 시점을 t 로 표현하고, 이전 시점을 $t-1$, 다음 시점을 $t+1$ 과 같은 형식으로 표현함.
 - 이는 현재 시점 t 에서의 Memory Cell이 갖고 있는 값은 과거의 Memory Cell들의 값에 영향을 받은 것임을 의미함.

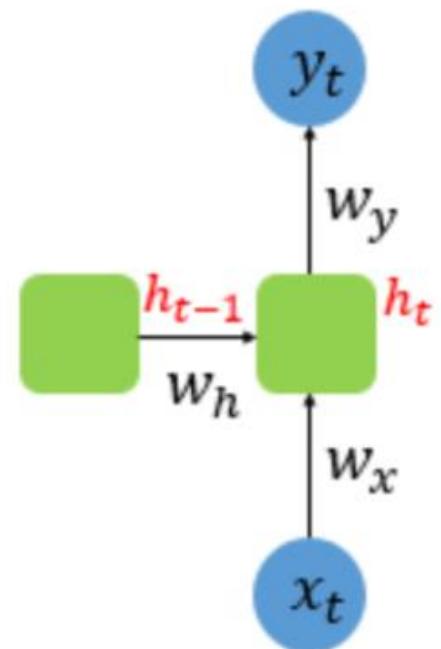
은닉 상태 (Cont.)

- Memory Cell이 다음 시점 $t+1$ 에 다시 자신에게 보내는 이 값을 은닉 상태(hidden state)라고 함.
 - 다시 말해서 현재 시점 t 의 Memory Cell은 이전 시점 $t-1$ 에서의 Memory Cell이 보낸 은닉 상태값을 다시 계산을 위한 입력값으로 사용



RNN의 은닉층, 출력층에 대한 수식

- 현재 시점 t에서의 은닉 상태값을 h_t 라고 정의
- 은닉층의 Memory Cell은 h_t 를 계산하기 위해서 총 2개의 가중치를 갖게 됨.
 - 입력층에서 입력값을 위한 가중치 W_x 이고,
 - 이전 시점 $t-1$ 의 은닉 상태값인 h_{t-1} 을 위한 가중치 W_h
- 식
 - 은닉층 : $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$
 - 출력층 : $y_t = f(W_y h_t + b)$
 - 단, f 는 비선형 활성화 함수 중 하나.



RNN의 은닉층, 출력층에 대한 수식(Cont.)

- 배치 크기가 1이고, d 와 D_h 두 값 모두를 4로 가정하였을 때, RNN의 은닉층 연산을 그림으로 표현

$$\tanh \left(W_h \begin{pmatrix} h_{t-1} \\ \vdots \end{pmatrix} + W_x \begin{pmatrix} x_t \\ \vdots \end{pmatrix} + b \right) = \begin{pmatrix} h_t \\ \vdots \end{pmatrix}$$

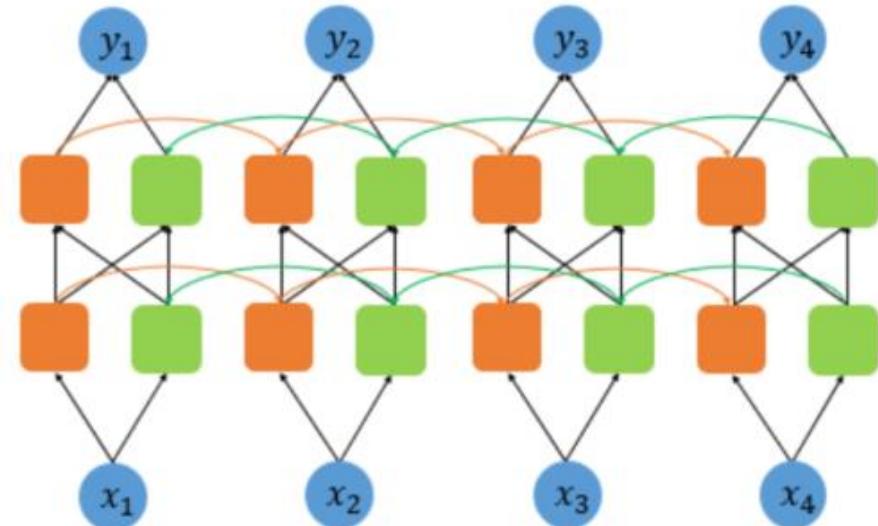
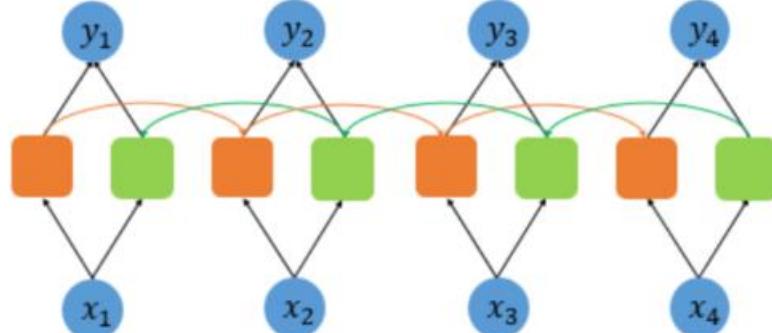
Diagram illustrating the computation of the hidden state h_t in an RNN. The input x_t is a column vector of size $d \times 1$. The previous hidden state h_{t-1} is a column vector of size $D_h \times 1$. The weight matrix W_h is a $D_h \times D_h$ matrix. The weight matrix W_x is a $D_h \times d$ matrix. The bias vector b is a column vector of size $D_h \times 1$. The output h_t is a column vector of size $D_h \times 1$.

양방향 순환 신경망(Bidirectional Recurrent Neural Network)

- 시점 t에서의 출력값을 예측할 때 이전 시점의 데이터뿐만 아니라, 이후 데이터로도 예측할 수 있다는 Idea에 기반
- 영어 빈칸 채우기 문제

Q) Exercise is very effective at [] belly fat.

- ① reducing
- ② increasing
- ③ multiplying



문맥을 예측해서 다음 단어 예측해보기

```
1 text="경마장에 있는 말이 뛰고 있다\n"
2 그의 말이 법이다\n
3 가는 말이 고와야 오는 말이 곱다"
```

```
1 from keras.preprocessing.text import Tokenizer
2 t = Tokenizer()
3 t.fit_on_texts([text])
4 encoded = t.texts_to_sequences([text])[0]
```

토큰화와 정수 인코딩

```
1 vocab_size = len(t.word_index) + 1
2 # 케라스 토크나이저의 정수 인코딩은 인덱스가 1부터 시작하지만,
3 # 케라스 원-핫 인코딩에서 배열의 인덱스가 0부터 시작하기 때문에
4 # 배열의 크기를 실제 단어 집합의 크기보다 +1로 생성해야하므로 미리 +1 선언
5 print('단어 집합의 크기 : %d' % vocab_size)
```

단어 집합의 크기 : 12

```
1 print(t.word_index)
```

```
{'말이': 1, '경마장에': 2, '있는': 3, '뛰고': 4, '있다': 5, '그의': 6, '법이다': 7,
'가는': 8, '고와야': 9, '오는': 10, '곱다': 11}
```

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 print(t.word_index)
```

```
{'말이': 1, '경마장에': 2, '있는': 3, '뛰고': 4, '있다': 5, '그의': 6, '법이다': 7,  
'가는': 8, '고와야': 9, '오는': 10, '꼽다': 11}
```

```
1 sequences = list()  
2 for line in text.split('\n'): # \n을 기준으로 문장 토큰화  
3     encoded = t.texts_to_sequences([line])[0]  
4     for i in range(1, len(encoded)):  
5         sequence = encoded[:i+1]  
6         sequences.append(sequence)  
7  
8 print('훈련 데이터의 개수: %d' % len(sequences))
```

훈련 데이터의 개수: 11

[2, 3]은 [경마장에, 있는]에 해당되며 [2, 3, 1]은 [경마장에, 있는, 말이]에 해당됨 모든 훈련 데이터에 대해서 맨 우측에 있는 단어에 대해서만 y로 분리하면 X와 y의 쌍(pair)이 됨

```
1 print(sequences)
```

```
[[2, 3], [2, 3, 1], [2, 3, 1, 4], [2, 3, 1, 4, 5], [6, 1], [6, 1, 7], [8, 1], [8, 1,  
9], [8, 1, 9, 10], [8, 1, 9, 10, 1], [8, 1, 9, 10, 1, 11]]
```

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 from keras.preprocessing.sequence import pad_sequences  
2 sequences = pad_sequences(sequences, maxlen=6, padding='pre')
```

Using TensorFlow backend.

```
1 print(sequences)
```

```
[[ 0  0  0  0  2  3]  
 [ 0  0  0  2  3  1]  
 [ 0  0  2  3  1  4]  
 [ 0  2  3  1  4  5]  
 [ 0  0  0  0  6  1]  
 [ 0  0  0  6  1  7]  
 [ 0  0  0  0  8  1]  
 [ 0  0  0  8  1  9]  
 [ 0  0  8  1  9 10]  
 [ 0  8  1  9 10  1]  
 [ 8  1  9 10  1 11]]
```

- `pad_sequences()`는 모든 데이터에 대해서 0을 추가하여 길이를 맞춰줌
- `maxlen`의 값으로 6을 주면 모든 데이터의 길이를 6으로 맞춰주며, `padding`의 인자로 '`pre`'를 주면 길이가 6보다 짧은 데이터의 앞을 0으로 채움

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 import numpy as np  
2 sequences = np.array(sequences)  
3 X = sequences[:, :-1]  
4 y = sequences[:, -1]  
5 # 리스트의 마지막 열을 제외하고 저장한 것은 X  
6 # 리스트의 마지막 열만 저장한 것은 y
```

```
1 print(X)
```

```
[[ 0  0  0  0  2]  
 [ 0  0  0  2  3]  
 [ 0  0  2  3  1]  
 [ 0  2  3  1  4]  
 [ 0  0  0  0  6]  
 [ 0  0  0  6  1]  
 [ 0  0  0  0  8]  
 [ 0  0  0  8  1]  
 [ 0  0  8  1  9]  
 [ 0  8  1  9  10]  
 [ 8  1  9  10  1]]
```

X와 y의 분리

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 print(y)
```

```
[ 3  1  4  5  1  7  1  9 10  1 11]
```

```
1 from keras.utils import to_categorical  
2 y = to_categorical(y, num_classes=vocab_size)
```

```
1 print(y)
```

```
[[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. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

훈련 데이터를 훈련
시키기 전에 y에 대해서
원-핫 인코딩을 수행

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 from keras.layers import Embedding, Dense, SimpleRNN  
2 from keras.models import Sequential  
3  
4 model = Sequential()  
5 # y를 제거하였으므로 이제 X의 길이는 5  
6 model.add(Embedding(vocab_size, 10, input_length=5))  
7 model.add(SimpleRNN(32))  
8 model.add(Dense(vocab_size, activation='softmax'))  
9 model.compile(loss='categorical_crossentropy', optimizer='adam',  
10                 metrics=['accuracy'])  
11 model.fit(X, y, epochs=200, verbose=2)
```

Epoch 180/200

- 0s - loss: 0.1980 - acc: 1.0000

Epoch 181/200

- 0s - loss: 0.1948 - acc: 1.0000

Epoch 182/200

- 0s - loss: 0.1917 - acc: 1.0000

Epoch 183/200

- 0s - loss: 0.1886 - acc: 1.0000

RNN 모델에 데이터를
훈련시킴

중간부터 정확도가
100%가 나옴

문맥을 예측해서 다음 단어 예측해보기 (Cont.)

```
1 def sentence_generation(model, t, current_word, n): # 모델, 토크나이저, 현재 단어, 반복할 횟수
2     init_word = current_word # 처음 들어온 단어도 마지막에 같이 출력하기 위해 저장
3     sentence = ''
4     for _ in range(n): # n번 반복
5         encoded = t.texts_to_sequences([current_word])[0] # 현재 단어에 대한 정수 인코딩
6         encoded = pad_sequences([encoded], maxlen=5, padding='pre') # 데이터에 대한 패딩
7         result = model.predict_classes(encoded, verbose=0)
8         # 입력한 X(현재 단어)에 대해서 Y를 예측하고 Y(예측한 단어)를 result에 저장.
9         for word, index in t.word_index.items():
10             if index == result: # 만약 예측한 단어와 인덱스와 동일한 단어가 있다면
11                 break # 해당 단어가 예측 단어이므로 break
12             current_word = current_word + ' ' + word # 현재 단어 + ' ' + 예측 단어를 현재 단어로 변경
13             sentence = sentence + ' ' + word # 예측 단어를 문장에 저장
14     # for문이므로 이 행동을 다시 반복
15     sentence = init_word + sentence
16     return sentence
```

```
1 print(sentence_generation(model, t, '경마장에', 4))
2 # '경마장에'라는 단어 뒤에는 총 4개의 단어가 있으므로 4번 예측
```

경마장에 있는 말이 뛰고 있다

모델이 정확하게 예측하고 있는지 문장을 생성하는 함수를 만들어서 실제로 출력

```
1 print(sentence_generation(model, t, '그의', 2)) # 2번 예측
```

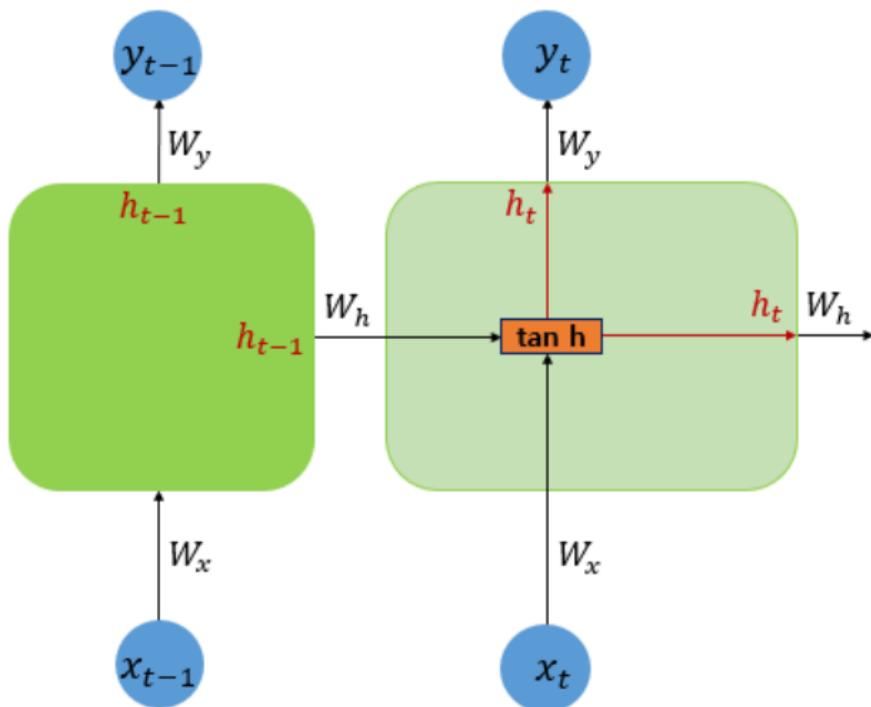
그의 말이 법이다

3. LSTM

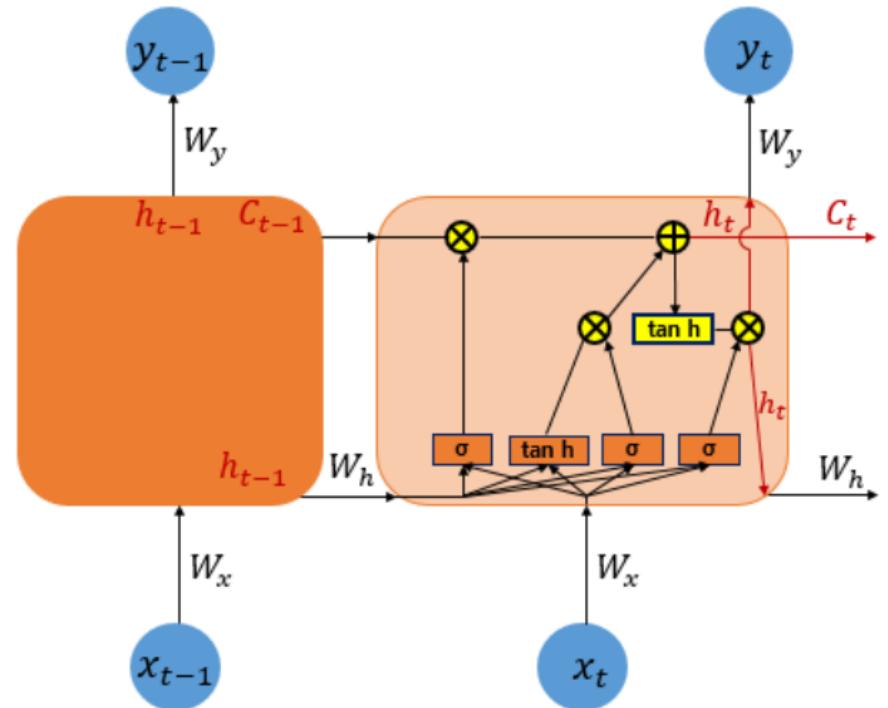


RNN vs. LSTM

Vanilla RNN



LSTM(Long Short-Term Memory)



LSTM

- 기존 코드와 크게 다른 점이 없고 모형을 만드는 클래스가 바뀌었음.

```
1 from keras.layers import Embedding, Dense, LSTM  
2 from keras.models import Sequential  
3  
4 model = Sequential()  
5 model.add(Embedding(vocab_size, 10, input_length=max_len-1))  
6 # y를 제거하였으므로 이제 X의 길이는 기존 데이터의 길이 - 1  
7 model.add(LSTM(128))  
8 model.add(Dense(vocab_size, activation='softmax'))  
9 model.compile(loss='categorical_crossentropy', optimizer='adam',  
10                 metrics=['accuracy'])  
11 model.fit(X, y, epochs=200, verbose=2)
```