

딥러닝 기초이론 Day 2



Outline

딥러닝 기초이론 Day 1

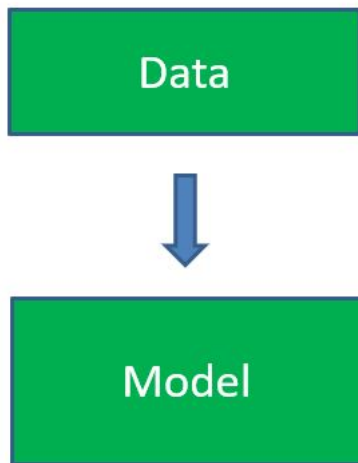


- 1 Intro. To Deep Learning
- 2 Neural Network
- 3 Optimization
- 4 Representation learning
- 5 TensorFlow

Intro. to Deep Learning

머신러닝의 한계

Machine Learning Algorithm은 Data로부터 Model의 parameter를 결정한다.

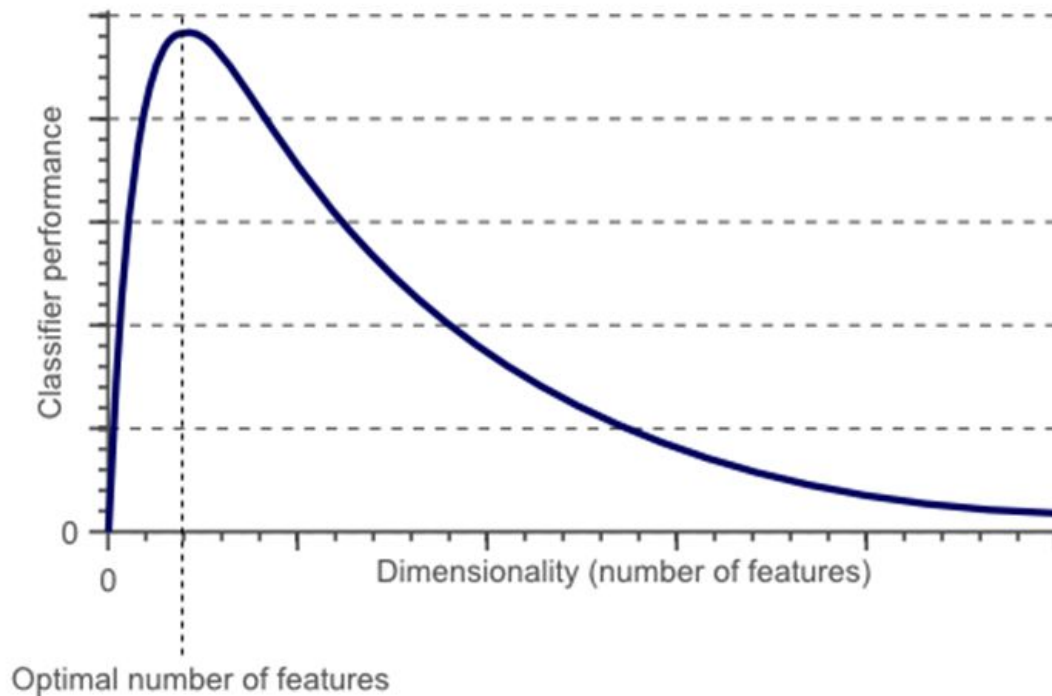


전통적인 머신러닝 알고리즘은 다음과 같은 문제를 갖는다.

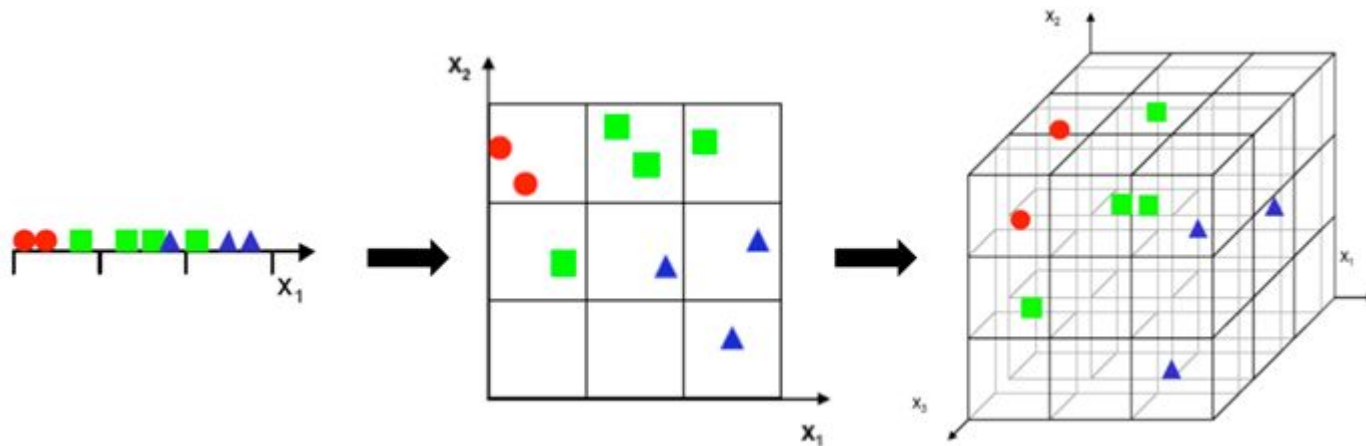
1. Data의 차원이 높을 때 차원의 저주(Curse of dimensionality)가 발생한다.
2. 데이터의 전처리(preprocessing)에 따라 성능이 크게 좌우된다.
3. 데이터가 비선형적으로 분포할 때, 선형 모델로 다루는데 한계를 갖는다.

머신러닝의 한계와 딥러닝

1. 차원의 저주 문제

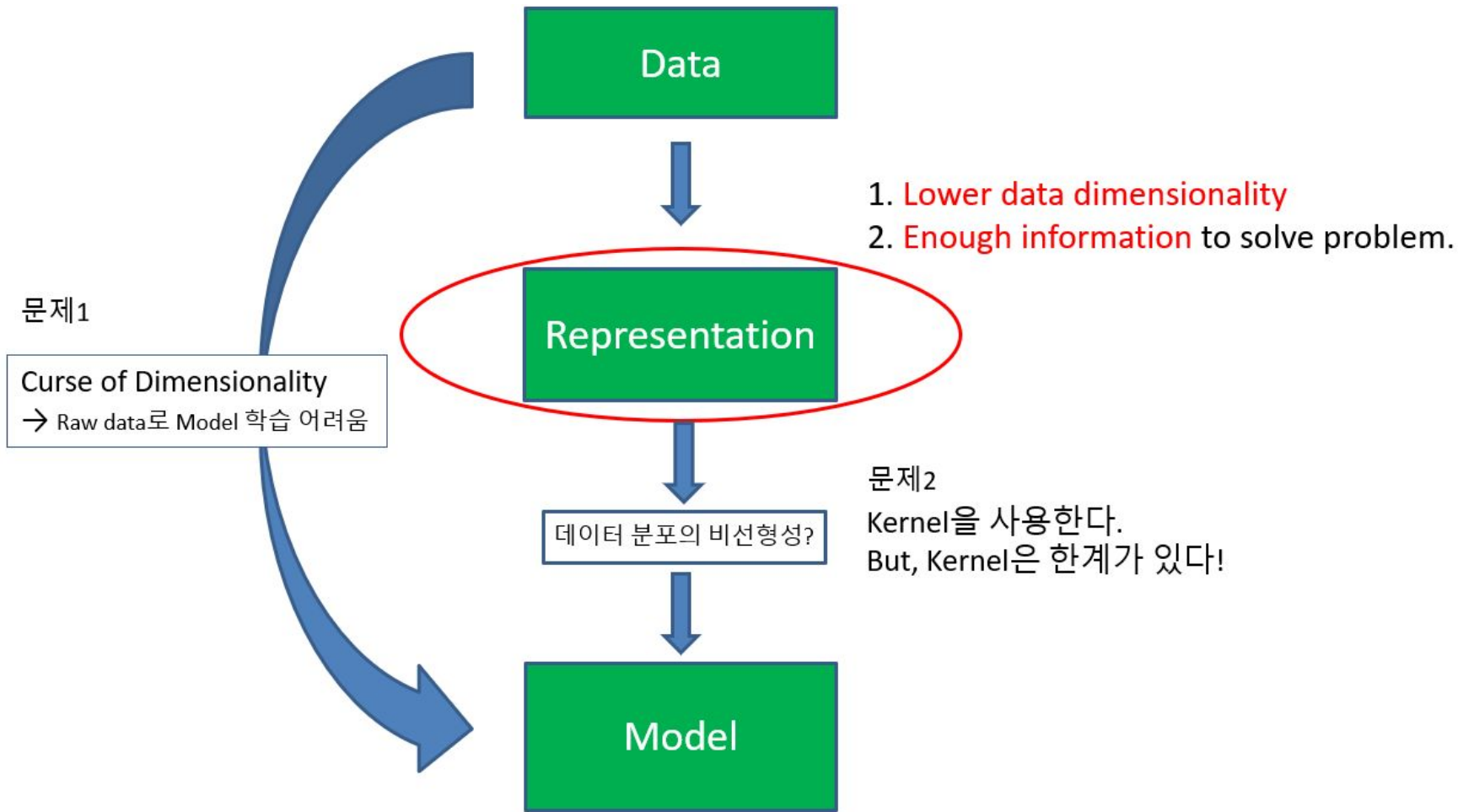


머신러닝의 한계와 딥러닝



데이터의 차원이 증가함에 따라 모델이 예측을 위해 알아야 할 데이터의 수가
지수적으로 증가한다.

현실적으로 모델에 필요한 데이터를 충족할 수 없음.



Data



Representation



Model

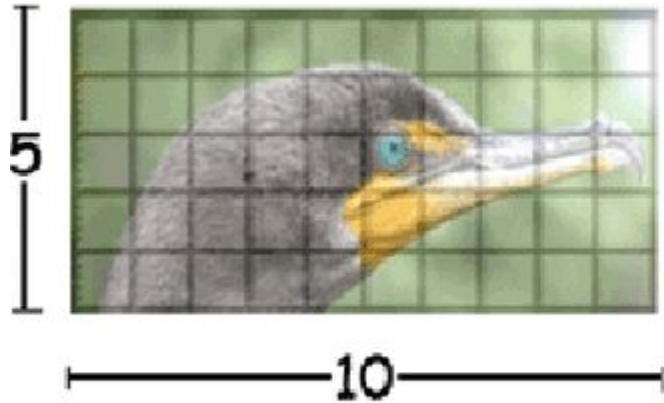


Images : High dimensional data

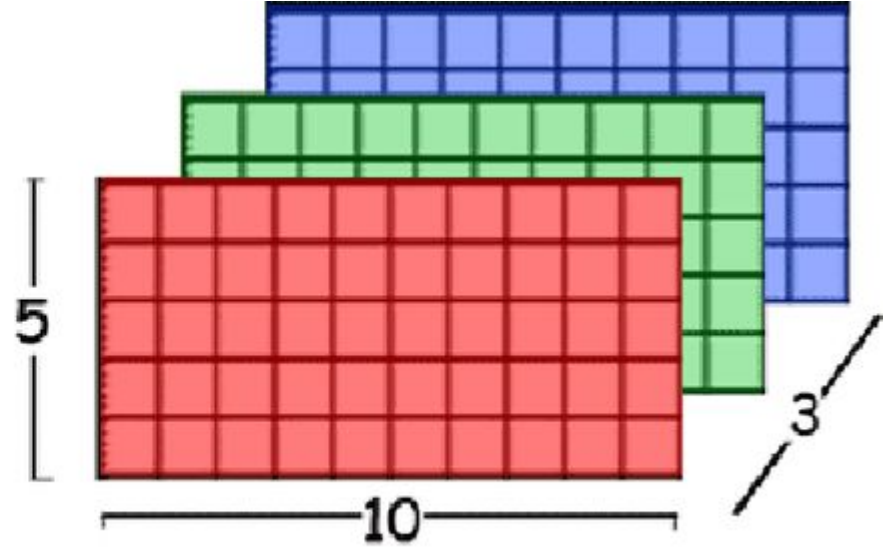
Representation : Number of corners
Feature engineering (Important, but high cost)

Model - Gets an input representation or feature (e.g. no of corners) and applies rules to detect the shape. (like if feature input is 0 then circle).

Colored digital image representation



Original Color Image



Matlab RGB Matrix

Data



Representation



Model



Images : High dimensional data

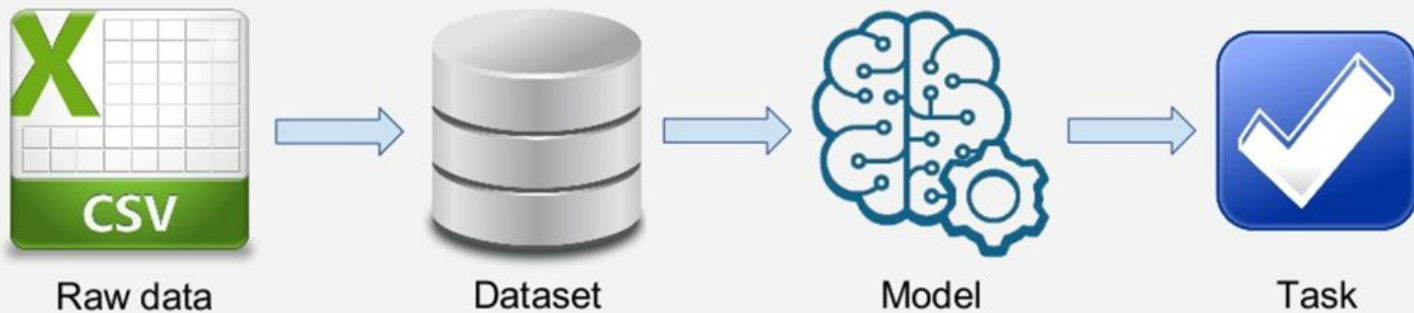
Representation : Number of corners
Feature engineering (Important, but high cost)

Model - Gets an input representation or feature (e.g. no of corners) and applies rules to detect the shape. (like if feature input is 0 then circle).

머신러닝의 한계와 딥러닝

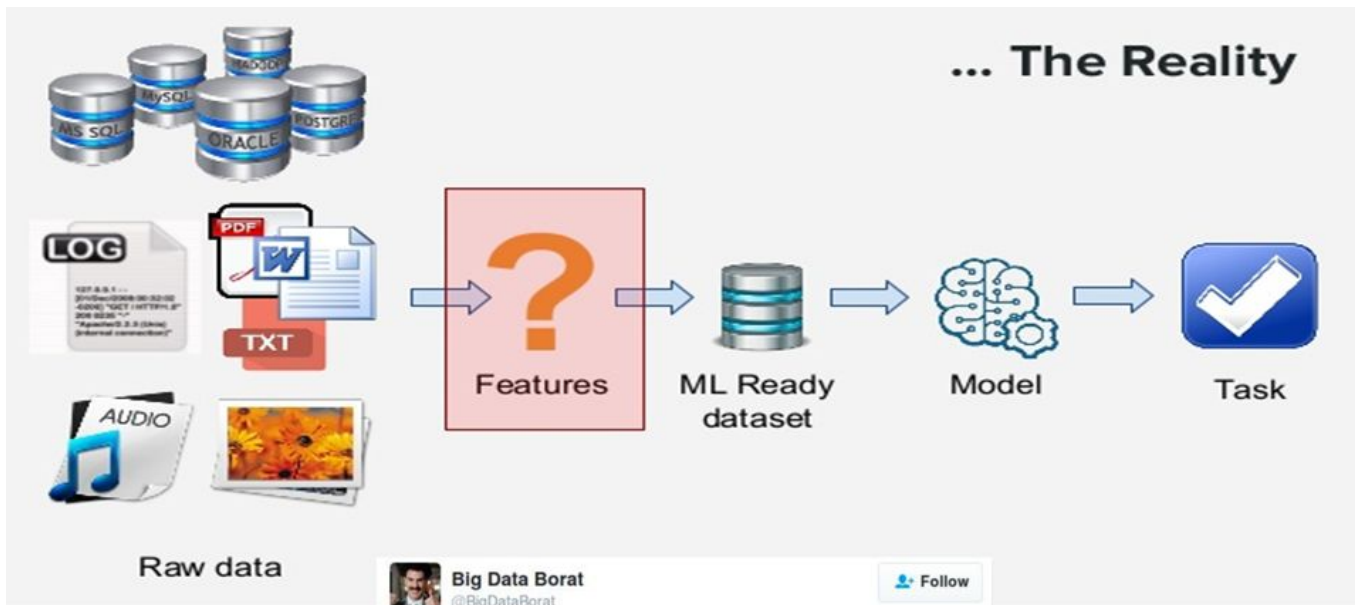
2. 데이터 전처리에 따른 성능 변화 문제

The Dream...



머신러닝의 한계

2. 데이터 전처리에 따른 성능 변화 문제

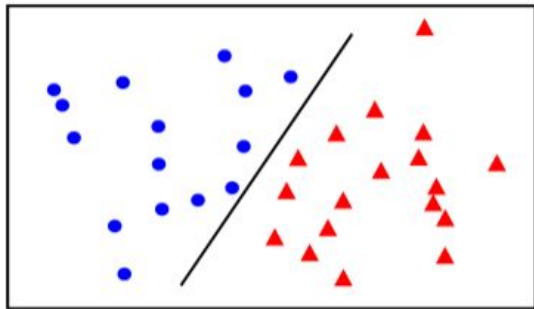
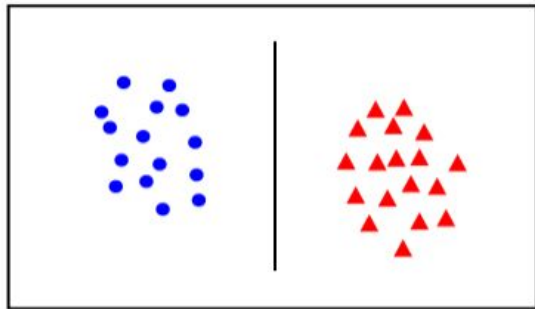


In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.

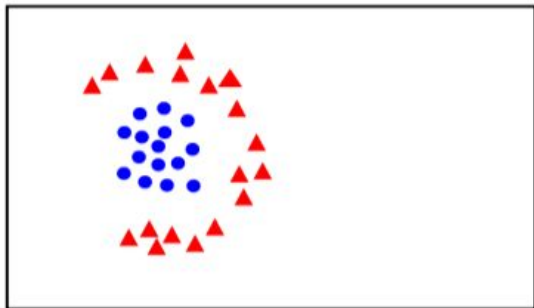
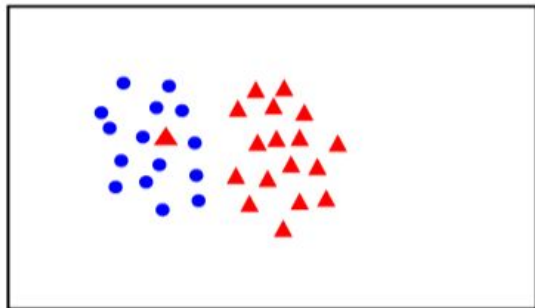
머신러닝의 한계와 딥러닝

3. 데이터 비선형성 문제

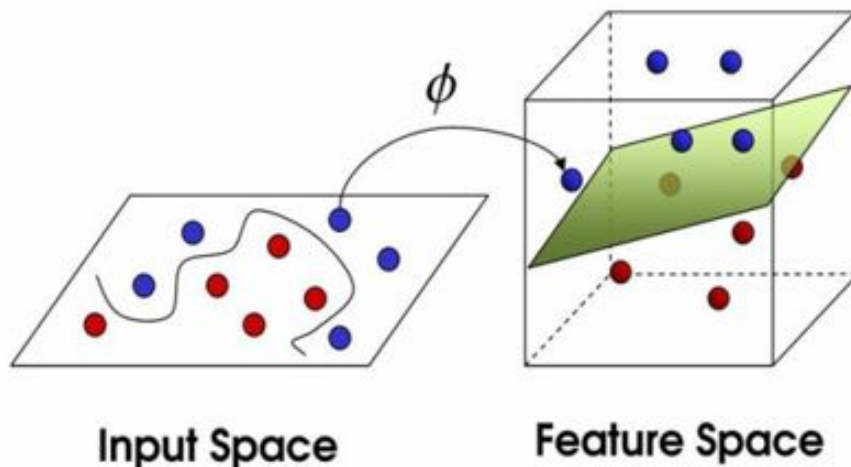
linearly
separable



not
linearly
separable



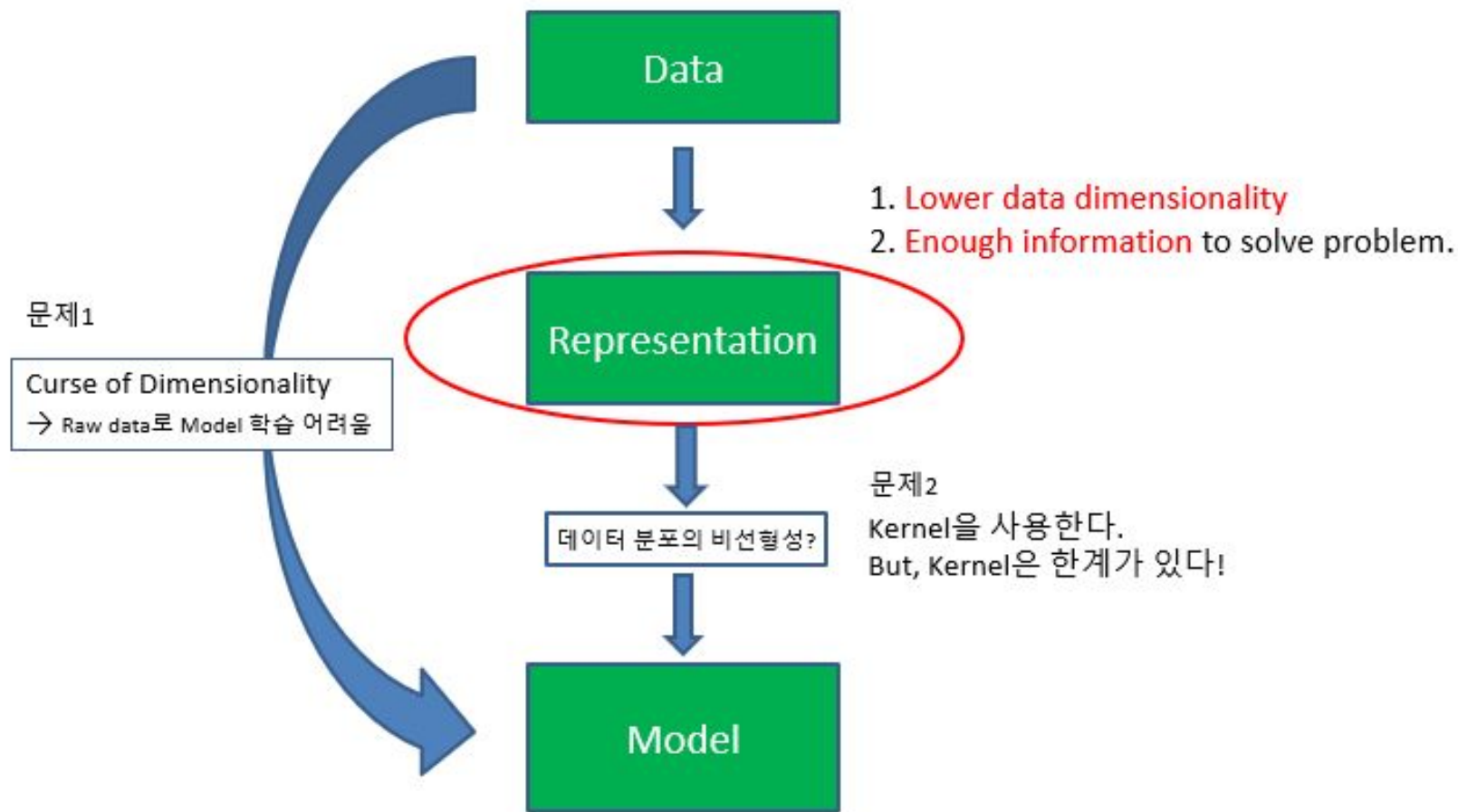
데이터를 선형 분포로 만드는 Kernel 함수를 사용하면 어떨까?



하지만 우리가 Not linearly separable한 데이터 문제를 해결하기 위한 Kernel을 획득할 수 있는가?

다음과 같은 문제가 있다.

1. Kernel을 문제에 따라 Design한다. → Input space의 분포를 알아야 함 (불가능)
2. Gaussian Kernel 사용한다. → Back to the curse of dimensionality problem!



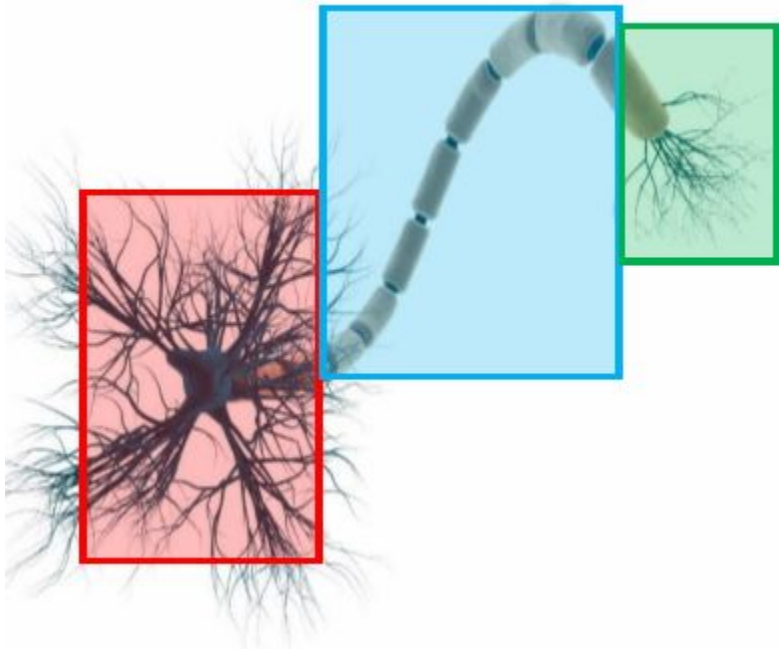
머신러닝의 한계와 딥러닝

딥러닝과 머신러닝의 차이

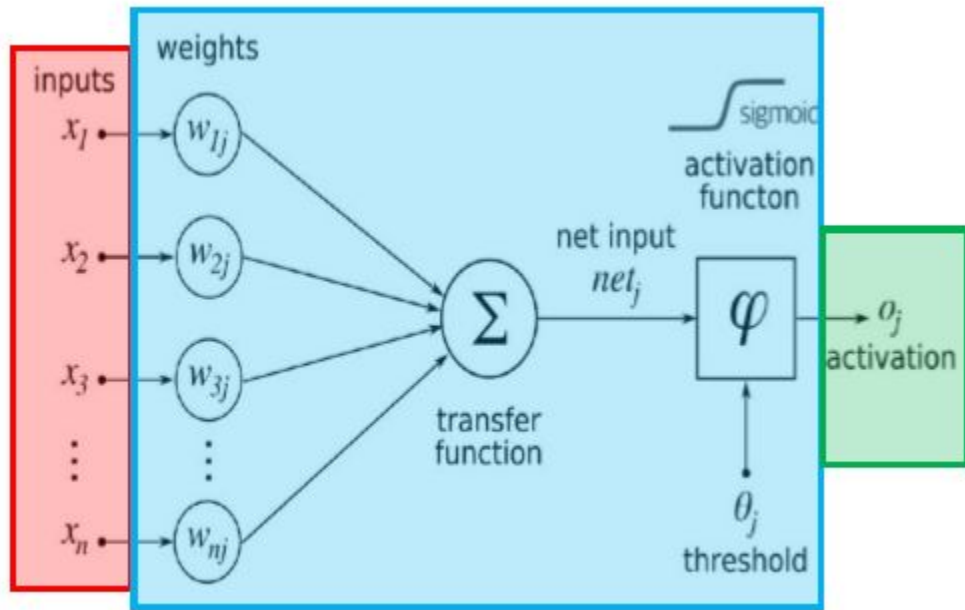
1. 기존의 머신러닝 모델의 한계점을 해결하기 위해 학습 가능한 비선형 모델인 인공신경망을 사용함.
*인공신경망의 은닉층은 데이터로부터 **kernel**을 학습하는 역할을 수행(learnable kernel)*
2. 비선형 함수를 포함하는 신경망 모델을 경사하강법에 의해 학습할 수 있도록 하는 오류역전파(**Back-propagation**) 알고리즘이 고안됨.
3. 데이터가 가지고 있는 특성(영상, 텍스트, 음성, 시계열)을 활용하여 분석할 수 있는 신경망 구조가 고안됨.
4. 딥러닝은 학습데이터셋의 수가 많아도 제약 없이 활용이 가능하며, 높은 복잡도의 모델(인공신경망)을 사용하기 때문에 방대한 데이터셋을 사용함.

Neural Network

인공신경망 모델 사용



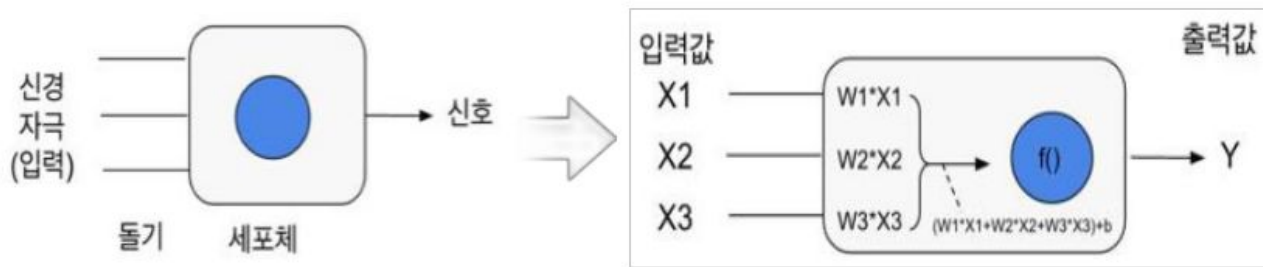
뉴런의 구조



퍼셉트론 모델

인공신경망은 퍼셉트론을 기본 단위로 구성된다. 퍼셉트론은 뇌의 신경망 구조를 구성하는 뉴런을 수학적으로 모델링한 것이다.

퍼셉트론의 모델



뉴런의 입력처리 방식

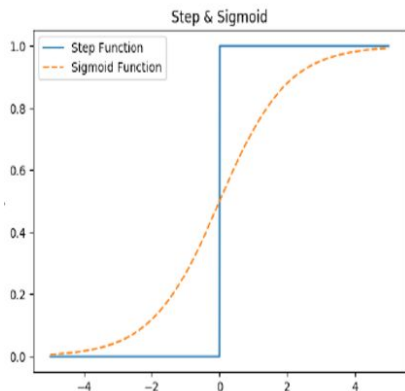
퍼셉트론의 입력처리 방식

뉴런은 연결된 이전 뉴런에서 입력되는 전기신호들을 처리하여 일정 이상의 전기적 자극(역치값)이 넘어설 시 다음 뉴런으로 전기신호를 전달함.

퍼셉트론 역시 다수의 입력을 받아, 입력 별 가중치에 따라 출력값을 계산한다. 또한 뉴런의 역치값을 모사한 활성화함수(Activation function)이라는 일종의 스위치를 사용한다.

활성함수(Activation function)

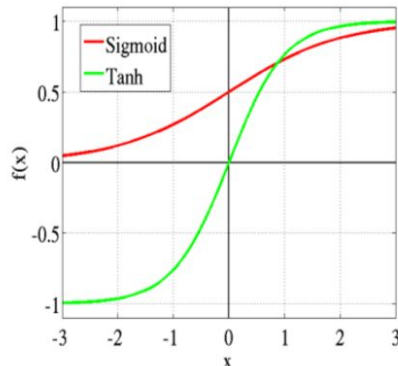
퍼셉트론의 계산결과(가중합)는 아래와 같은 활성함수를 거쳐 최종 출력됨.



<Sigmoid Function>

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

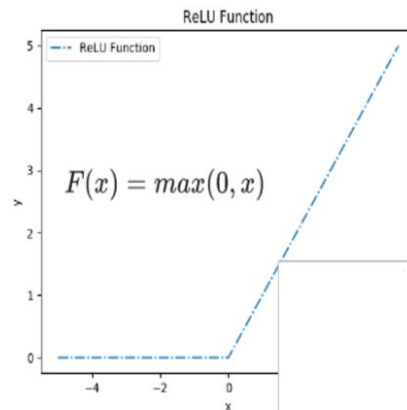
$$f'(x) = f(x)(1-f(x))$$



<tanh Function>

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

$$f'(x) = 1 - f(x)^2$$

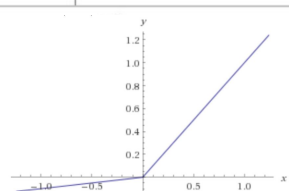


<ReLU Function>

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

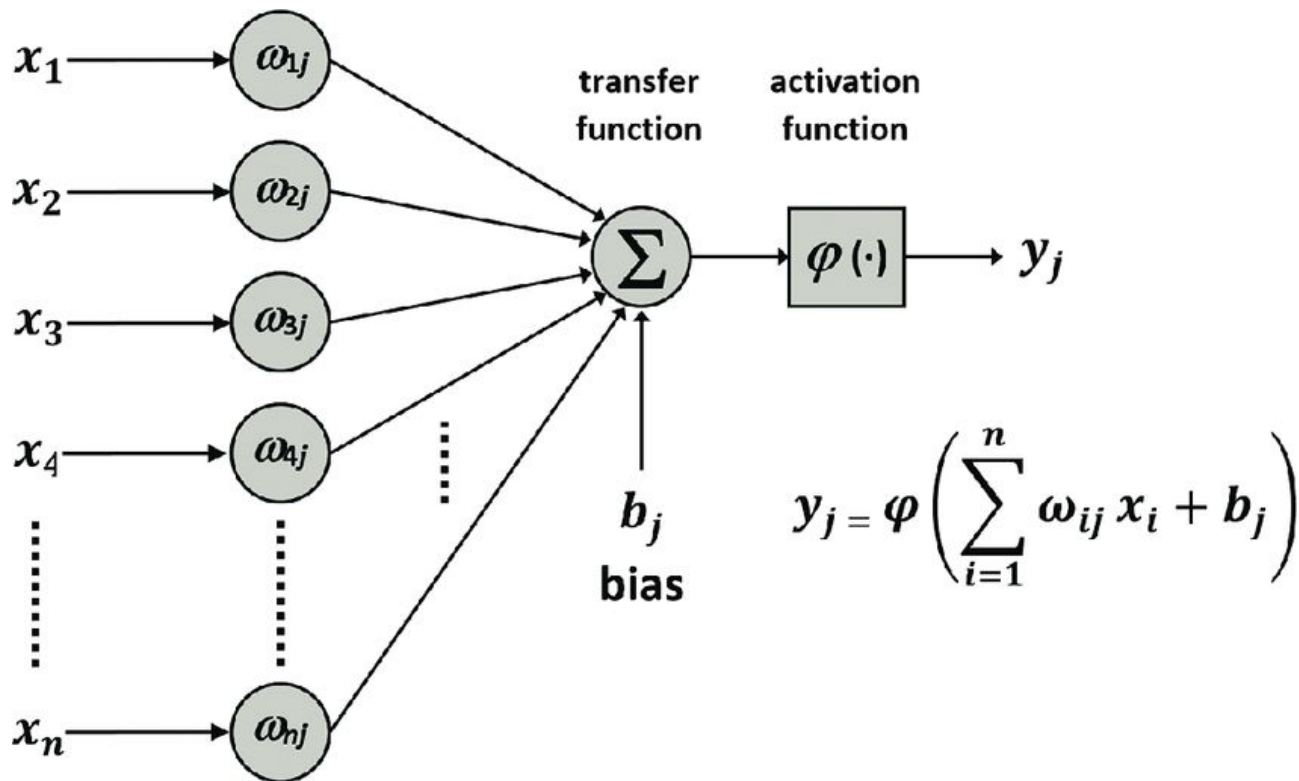
$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Leaky ReLU



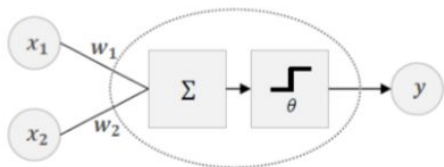
x < 0 구간에도
기울기 존재

최종적으로 퍼셉트론은 1) 입력된 x 의 값들에 대한 가중치합(Linear combination)과 2) 활성화함수로 구성됨.



MLP(Multi-Layer Perceptron)

하나의 퍼셉트론으로는 표현할 수 있는 결정 영역이 선형적이거나, 다수의 퍼셉트론을 계층구조로 쌓는 다층 퍼셉트론(MLP)는 더 복잡한 결정 영역을 갖을 수 있다.



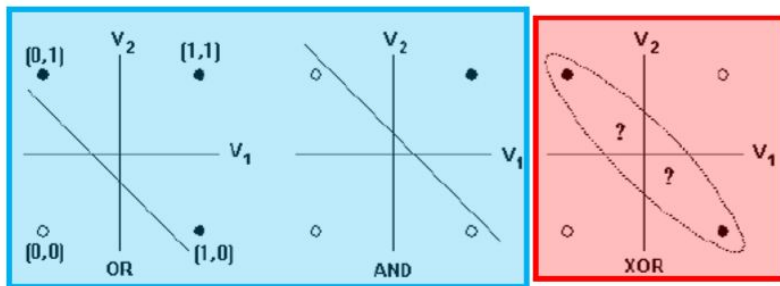
2개의 input 을 받는 Perceptron

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

Activation function
(Step function)


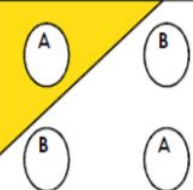


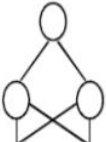
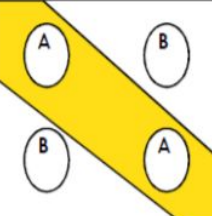
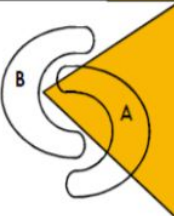

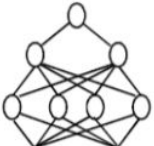
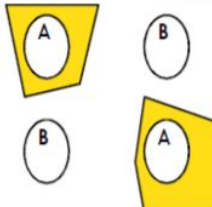
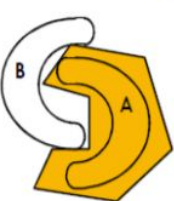
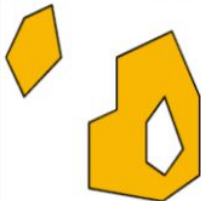
	w_1	w_2	θ
AND	0.5	0.5	0.7
OR	0.5	0.5	0.2

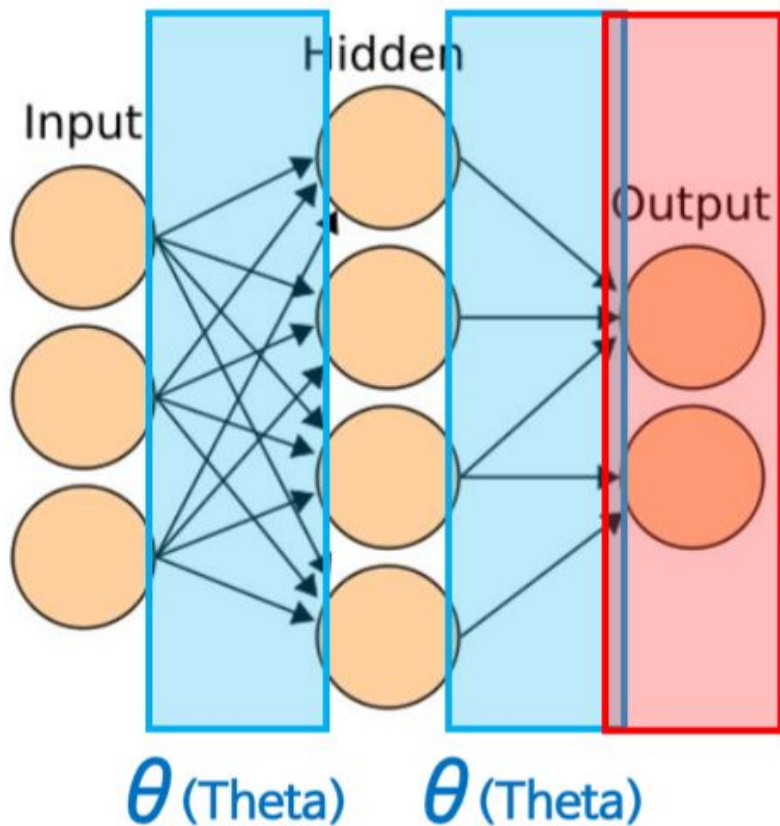
XOR?



단일 Perceptron으로는 풀이가 불가능

Non-linear Activation function & Multi-Layer

구조	결정 영역 형태	Exclusive-OR 문제	불가능 영역을 나타내는 것	이러한 영역을 나타내는 것
단층 	초평면에 의해 나뉘어지는 반평면 (Hyper plane)			
두 개층 	불가능한 모양도 나타낼 수 있음			
세 개층 	임의의 형태 (노드들의 개수에 따라 복잡도가 결정됨)			



- Input Layer
: 초기 입력값을 받는 가장 첫번째 Layer
- Hidden Layer
: 중간 단계의 모든 Layer를 의미함
- Output Layer
: 가장 마지막 단계의 Layer로 모델의 출력값을 계산

여러 간단한 계산을 수행하는 다수의 뉴런들이 층(Layer)를 이루어 복잡한 계산을 수행할 수 있는 망(Network) 구조를 구성하는 것이 바로 **인공신경망**(Neural Network)이다.

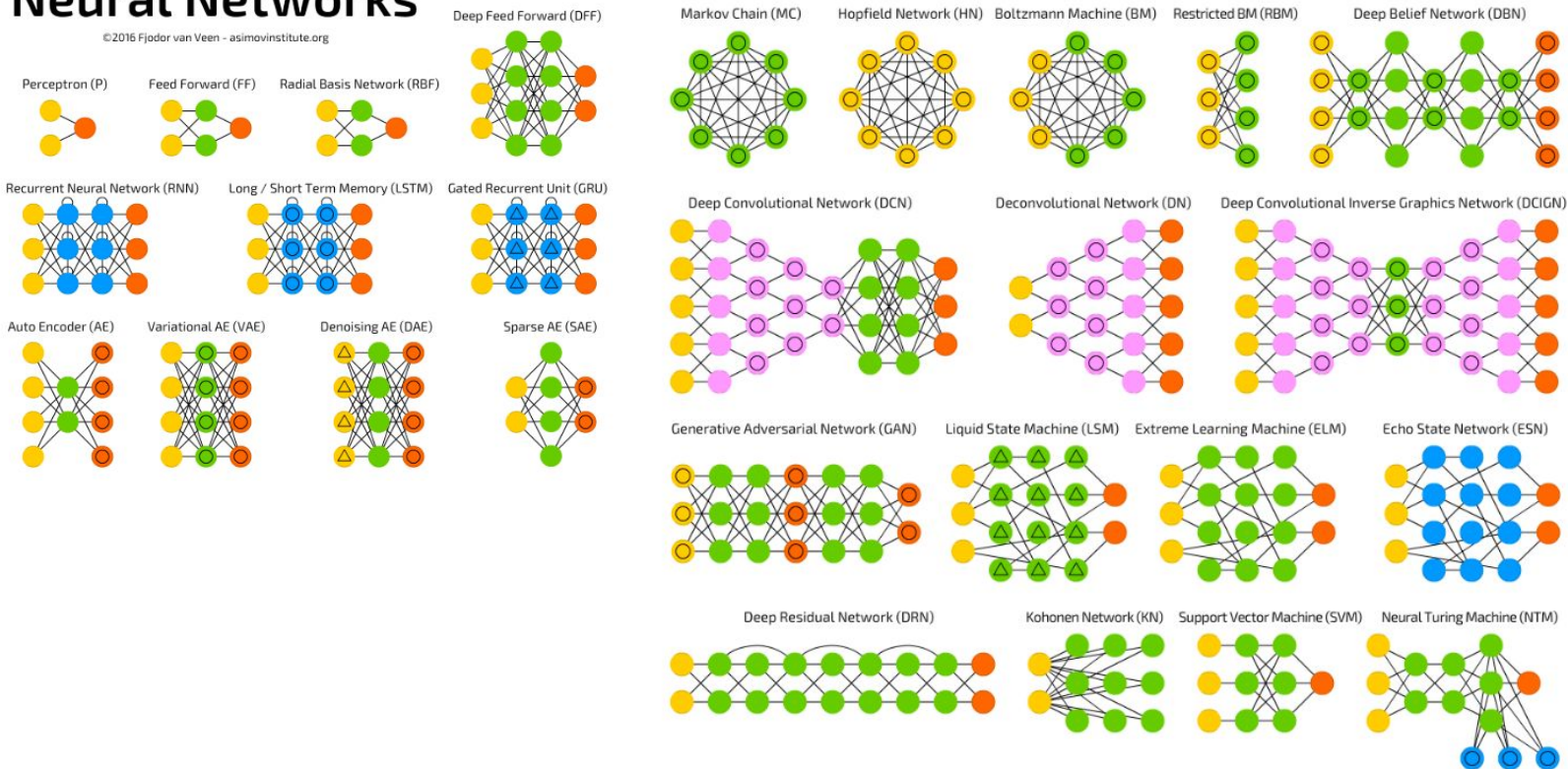
MLP는 가장 기본적인 인공신경망의 종류 중 하나이다.

A mostly complete chart of

Neural Networks

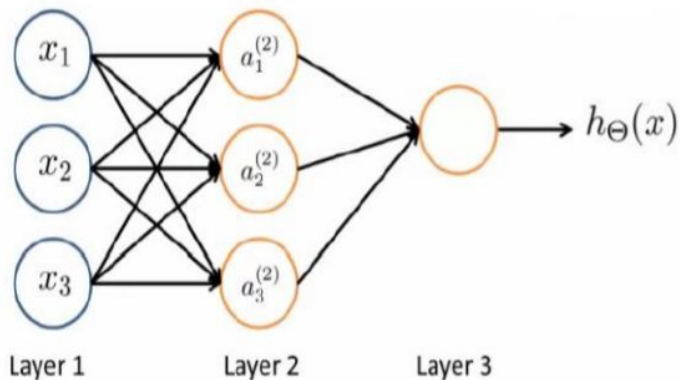
©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool



인공신경망의 계산방식

1. Forward propagation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_{\theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

* # of Layers, # of Neurons, Activation func 등은 **Hyper Parameter** 의 영역

- Hidden Layer 의 뉴런 수와 Hidden Layer 의 개수는 신경망 설계자의 직관과 경험에 의존
- Hidden Layer 의 뉴런 수가 너무 많으면 **Overfitting**, 너무 적으면 데이터를 충분히 표현하지 못함. (**Underfitting**)

Feedforward 신경망의 **학습**

: 원하는 결과를 얻기 위해 **뉴런 사이의 적당한 가중치(θ)**들을 알아내는 것

→ Output 과 실제 Data label 의 차이를 바탕으로 Cost function 을 구성하고, **Gradient Descent** 을 적용하여 최적의 가중치를 찾아감

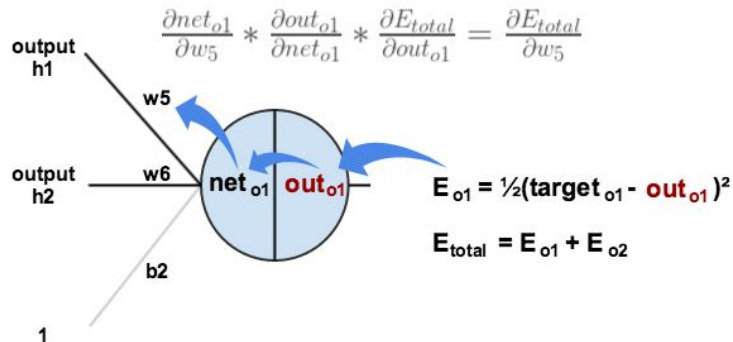
인공신경망의 계산방식

2. Back-propagation (오류역전파)

: 학습된 출력 값과 실제 값과의 차이인 오차를 계산하여
Feedforward 반대인 역방향으로 전파(Propagation)하는 알고리즘

- 이러한 문제를 해결하기 위해 Back propagation 알고리즘이 도입됨
- Forward 방향으로 끊임없이 계산하는 대신,
- Forward 방향으로 한번 연산을 한 다음 그 결과값(error)을 가지고,

역방향(backward)으로 가면서 Parameter 를 Update! (각 weight 마다 에러에 미치는 영향을 계산하여 weight를 update)



Optimization

인공신경망의 학습

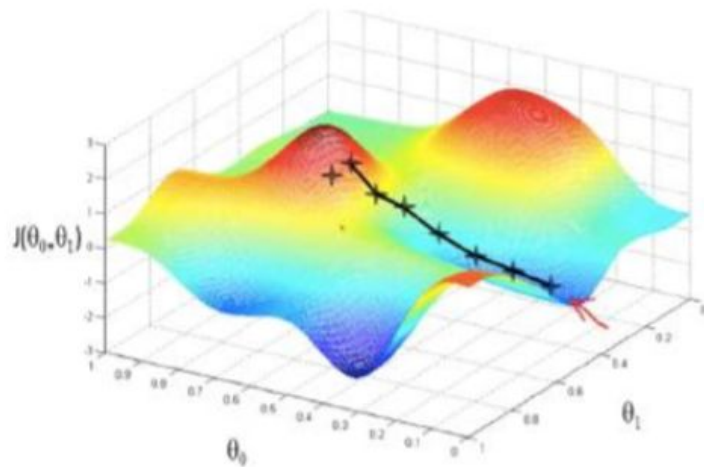
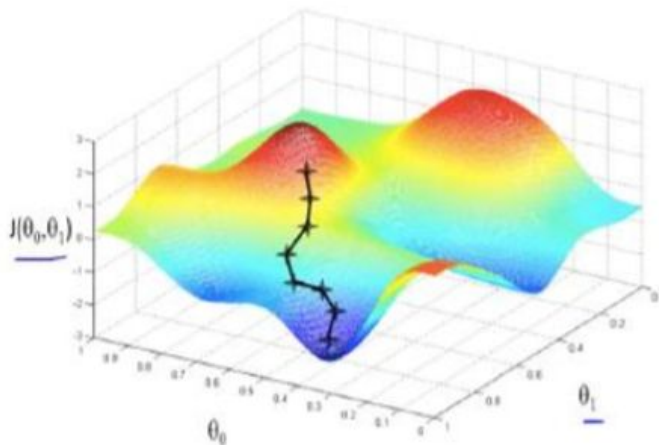


- 1) 임의의 Parameter(θ) 초기값을 시작으로
- 2) Cost Function을 최소화하는 방향으로
- 3) Optimizer를 통해 Parameter(θ)를 반복적으로 Update

Neural Network learning process

초기값 설정(Initialization)

- 학습을 진행하기 전, 가중치(θ = weight)의 초기 값을 설정해야함
- 초기값 설정은 1) **Training Speed**, 2) 실제 **Learning 가능 여부**를 결정하는 중요한 역할을 함



Use **Xavier Initialization** (with Sigmoid or tanh)

- Xavier Glorot와 Yoshua Bengio 논문에서 권장
- 활성화 함수로 **Sigmoid** 함수나 **tanh** 함수를 사용할 때 적용
- 일반적인 딥러닝 프레임워크가 표준적으로 이용

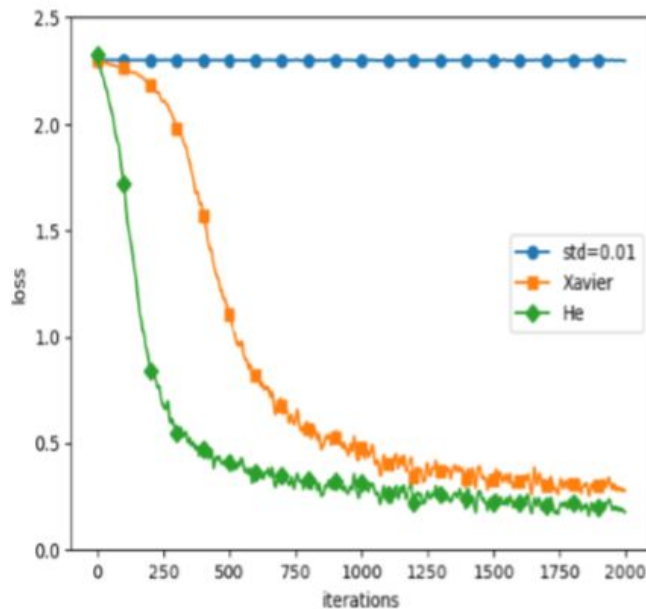
- 표준편차가 $\sqrt{\frac{1}{n}}$ 인 정규분포를 따르도록 가중치 초기화

* n = # of nodes of previous layer

or **He Initialization** (with ReLU)

- Kaiming He 가 발견
- 활성화 함수가 **ReLU** 함수일 때 적용

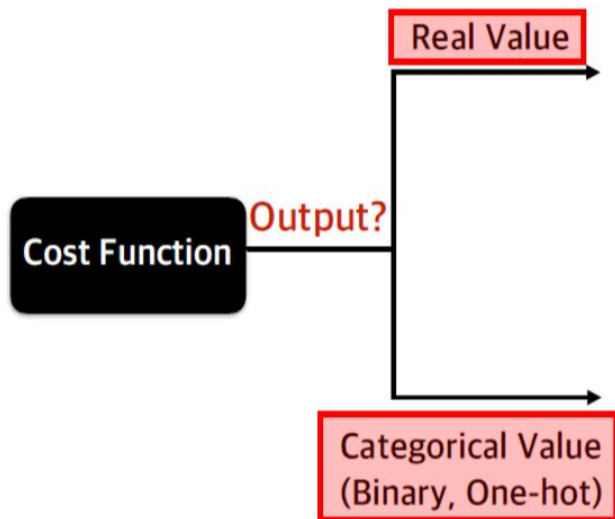
- 표준편차가 $\sqrt{\frac{2}{n}}$ 인 정규분포를 따르도록 가중치 초기화



ReLU 활성화 함수를 이용한 5층
신경망에서 MNIST 테스트

Neural Network learning process

비용 함수(Cost function)



<Least Square Method>

- 실제값과 모델 출력값 **차이의 제곱들의 합**
- Linear Regression 모델에서 주로 사용

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

<Cross-Entropy>

- **오분류에 대한 확률적인 비용을 의미**
- Classifier (특히, softmax) 모델과 주로 사용
- y 값은 0 또는 1을 나타냄

$$J(\theta) = -\frac{1}{m} \sum_i y^{(i)} \log h_{\theta}(x^{(i)})$$

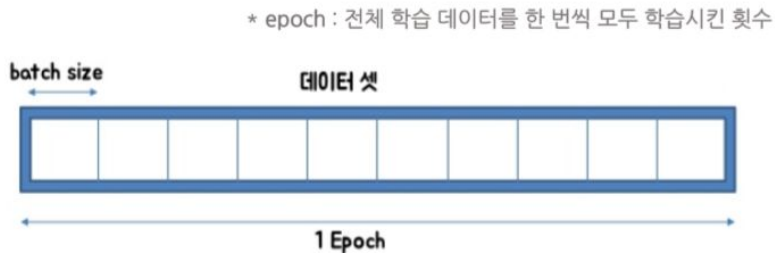
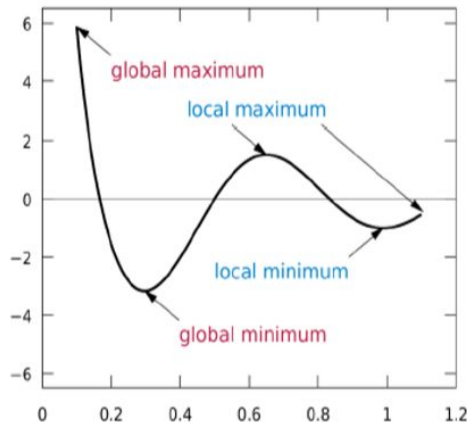
Neural Network learning process

경사하강법(Gradient descent method)

(Full-Batch) Gradient Descent

모든 데이터를 학습하고 오차를 이용하여 가중치 갱신 값을 계산한 다음 갱신

- (Weak) 학습 데이터가 많으면 가중치 갱신 값 평균 계산 시 계산량 많음. SGD 대비 **학습이 오래 걸림.**
- (Weak) 가중치 x 의 초기 설정에 따라 Global minimum 값이 아닌 **local minimum 값으로 수렴**할 수 있다.



ex) 총 데이터가 100개, batch size가 10이면,
1 iteration = 10개 데이터에 대해서 학습
1 Epoch = 100/batch size = 10 iteration

Neural Network learning process

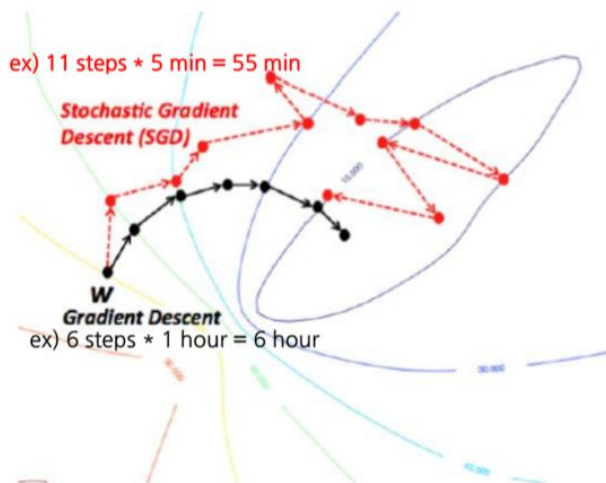
경사하강법(Gradient descent method)

: 추계학(推計學)의, 확률(론)적인

Stochastic Gradient Descent (SGD, 확률적 경사 하강법)

하나의 학습 데이터(Batch size = 1)마다 오차를 계산해서 신경망의 가중치를 바로 조절, 속도가 무척 빠르다.

- (Weak) 한 개의 학습 데이터마다 매번 가중치를 갱신함으로 **신경망의 성능이 들쭉날쭉 변함**.
- (Weak) 최적의 학습률을 구하기 위해 **일일이 튜닝하고 수렴조건(Stop)을 조정해야 함**

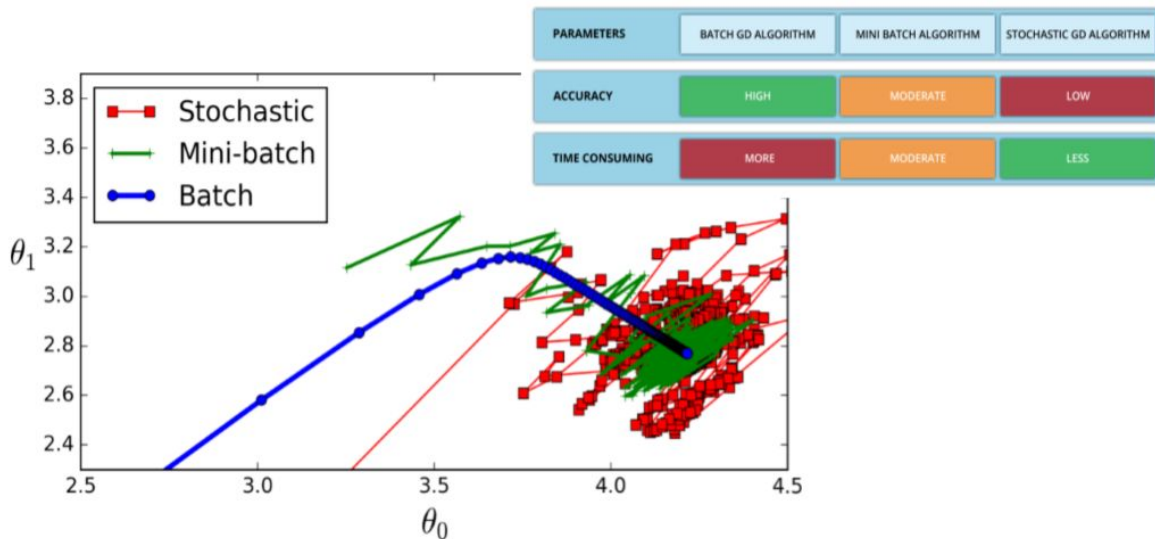


Neural Network learning process

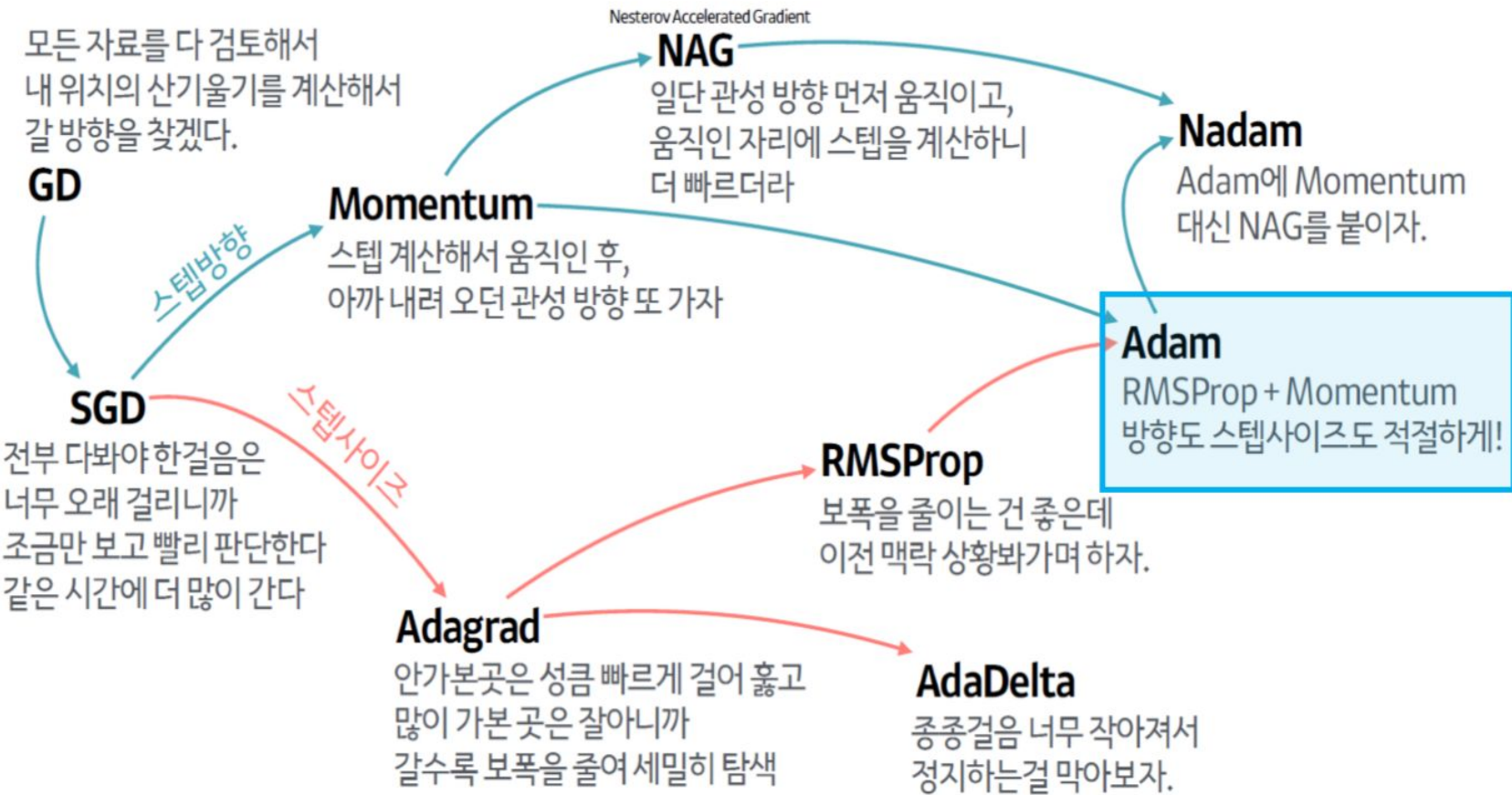
경사하강법(Gradient descent method)

Mini-Batch Stochastic Gradient Descent

트레이닝 데이터에서 소량의 데이터를 적당히 선택하여 학습 후 갱신 처리를 수행
위 두 가지 경사 하강법의 단점을 보완하고 장점을 취함.
GPU 기반의 효율적인 병렬 컴퓨팅이 가능해 진다.

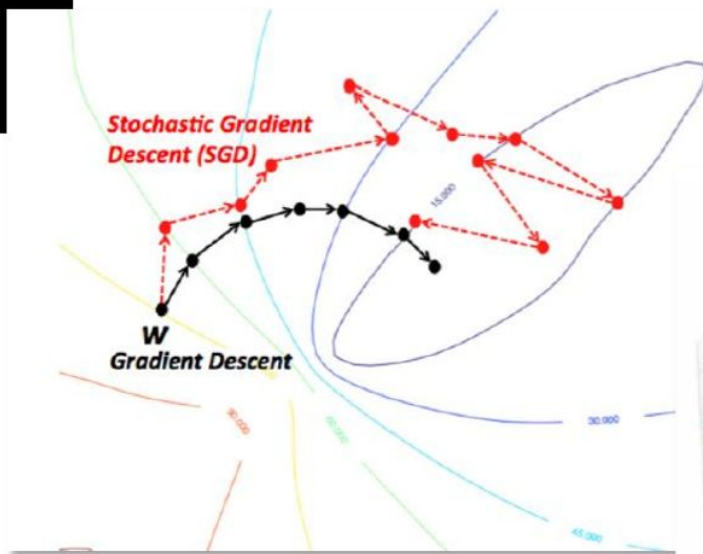


이름	특징
SGD (Stochastic Gradient Descent)	<ul style="list-style-type: none"> • 기울어진 방향(기울기)으로 일정 거리(학습률)만 이동 • 단순하지만 문제에 따라 비효율적이다.
Momentum ('운동량')	<ul style="list-style-type: none"> • 기울기 방향으로 물체가 가속되는 관성의 원리 적용. 속도 (Velocity)라는 변수가 추가 • 한 스텝전의 기울기를 일정한 비율로 반영하여 새로 계산된 기울기와 합해서 사용
Nesterov accelerated gradient (NAG)	<ul style="list-style-type: none"> • Momentum에서 발전된 방법으로 momentum보다 더 공격적으로 속도(Velocity)를 활용 • 현재 가중치에서 기울기를 계산하기 전에, 먼저 학습했던 만큼 한 번 더 가본 상태로 계산
AdaGrad	<ul style="list-style-type: none"> • 처음에는 학습률을 크게 하다가 조금씩 작게 학습하는 방법 • 전체 학습률을 조정하는 것이 아니라, 각각의 매개변수에 맞는 학습률을 조정. • 과거의 기울기를 제공해서 계속 더해감으로 학습이 진행될수록 갱신 강도가 약해짐
Adadelta	<ul style="list-style-type: none"> • AdaGrad의 extension. 그러나 생각보다 잘 작동 안 함.
RMSprop	<ul style="list-style-type: none"> • AdaGrad의 단점은 무한히 학습하다 보면 순간 갱신량이 0이 되어 학습이 안됨. • 이를 개선하여 RMSprop은 먼 과거의 기울기는 잊고 새로운 기울기 정보를 크게 반영
Adam (Adaptive Moment Estimation)	<ul style="list-style-type: none"> • Momentum과 AdaGrad의 이점을 조합. • 학습률, 1차 momentum용 계수, 2차 momentum용 계수의 3개의 hyper paramter 설정

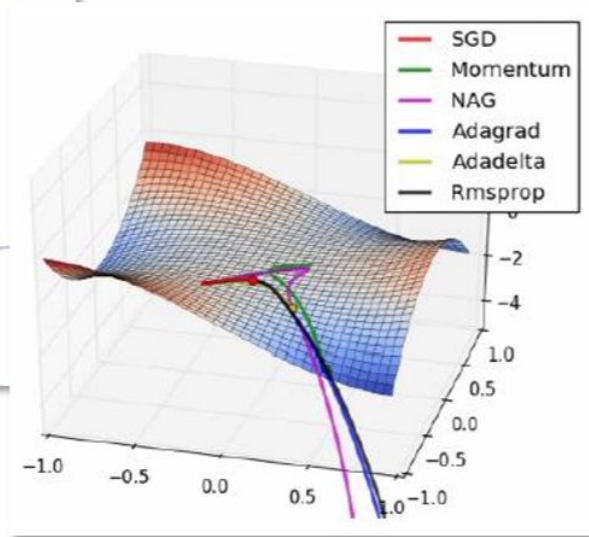


[백날 자습해도 이해 안 가던 딥러닝, 머리 속에 인스톨 시켜드립니다.] by 하용호 님 (49p)

Gradient-based Learning



<http://imgur.com/NKsFHJb>



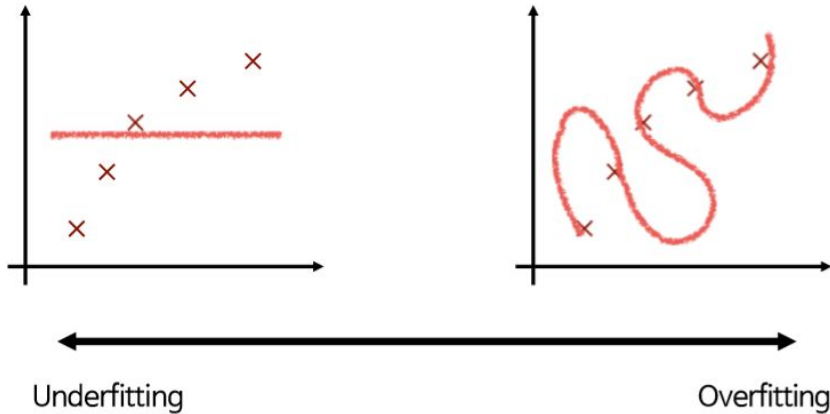
Use **Adam (Adaptive Moment Estimation) Optimizer**

(Momentum과 AdaGrad/RMSProp의 이점을 조합)

Neural Network learning process

Avoiding Overfitting : Regularization term

〈Overfitting Example〉



How to avoid Overfitting?

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



$$J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$$

Regularization Term

Regularization Term

$$J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$$

L1 regularization: $\lambda |\Theta|$

: L1 정규화

- 가중치의 절대값의 합에 비례하여 가중치에 페널티를 준다.
- 관련성이 없거나 매우 낮은 특성의 가중치를 정확히 0으로 유도하여, 모델에서 해당 특성을 배제하는 데 도움이 됨.

L1 & L2 Regularization

=

Weight Decaying

(가중치 감소)

L2 regularization: $\lambda |\Theta|^2$ $\|w\|_2^2 = w_1^2 + w_2^2 + \dots + w_n^2$

: L2 정규화

- 가중치의 제곱의 합에 비례하여 가중치에 페널티를 준다.
- 가중치를 작은 값으로 유도하지만 정확히 0.0 으로 만들지는 못함

Regularization Rate

$$J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$$

최소화(손실 함수 + λ (정규화 함수))

Regularization Rate: λ

: 정규화율(Lambda)

- 스칼라 값
- 정규화 함수의 상대적 중요도를 지정
- 정규화율을 높이면 과적합이 감소하지만 모델의 정확성이 떨어질 수 있음 (Underfitting)

Neural Network learning process

Avoiding Overfitting : Early stopping

<Data Split with No Validation>

Overall Dat

: Training Data에 과적합된 학습 가능성 높음

<Data Split with Validation>

Training Data

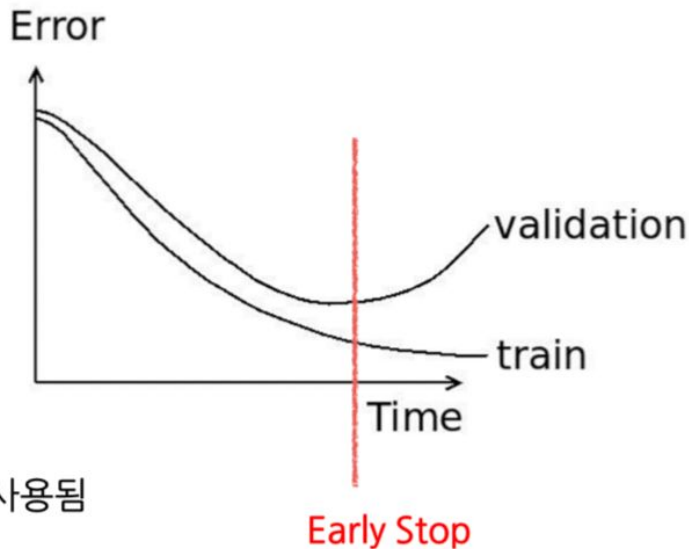
Validation

Test

: 학습 반복마다 일반화(Generalization)에 대한 검증 진행

* Test 데이터는 학습 완료 후 최종 성능을 확인하기 위해서만 사용됨

<Learning Curve>

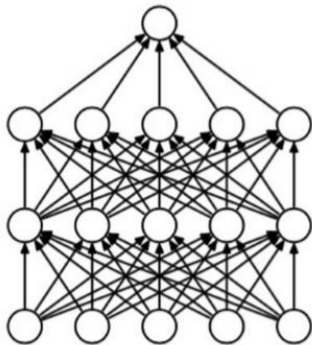


Neural Network learning process

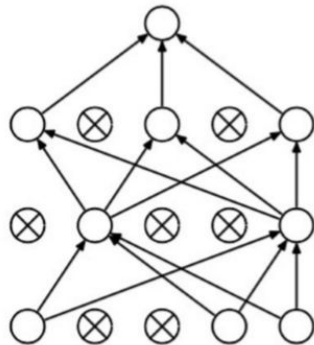
Avoiding Overfitting : Dropout

- Overfitting을 예방하기 위해 사용되는 가장 대표적인 방법 중 하나
- 매 학습 과정마다 전체 뉴런 중 $(1-p)$ 의 비율 만큼의 뉴런을 생략 하고 학습 진행
(일반적으로 입력층에서 20~25%, 은닉층에서 50% 정도 생략)
- Test 시에는 dropout을 사용하지 않고, 기존과 같은 방법으로 모든 뉴런에 대하여 결과를 계산함
- 일종의 Model ensemble

<Dropout Architecture>

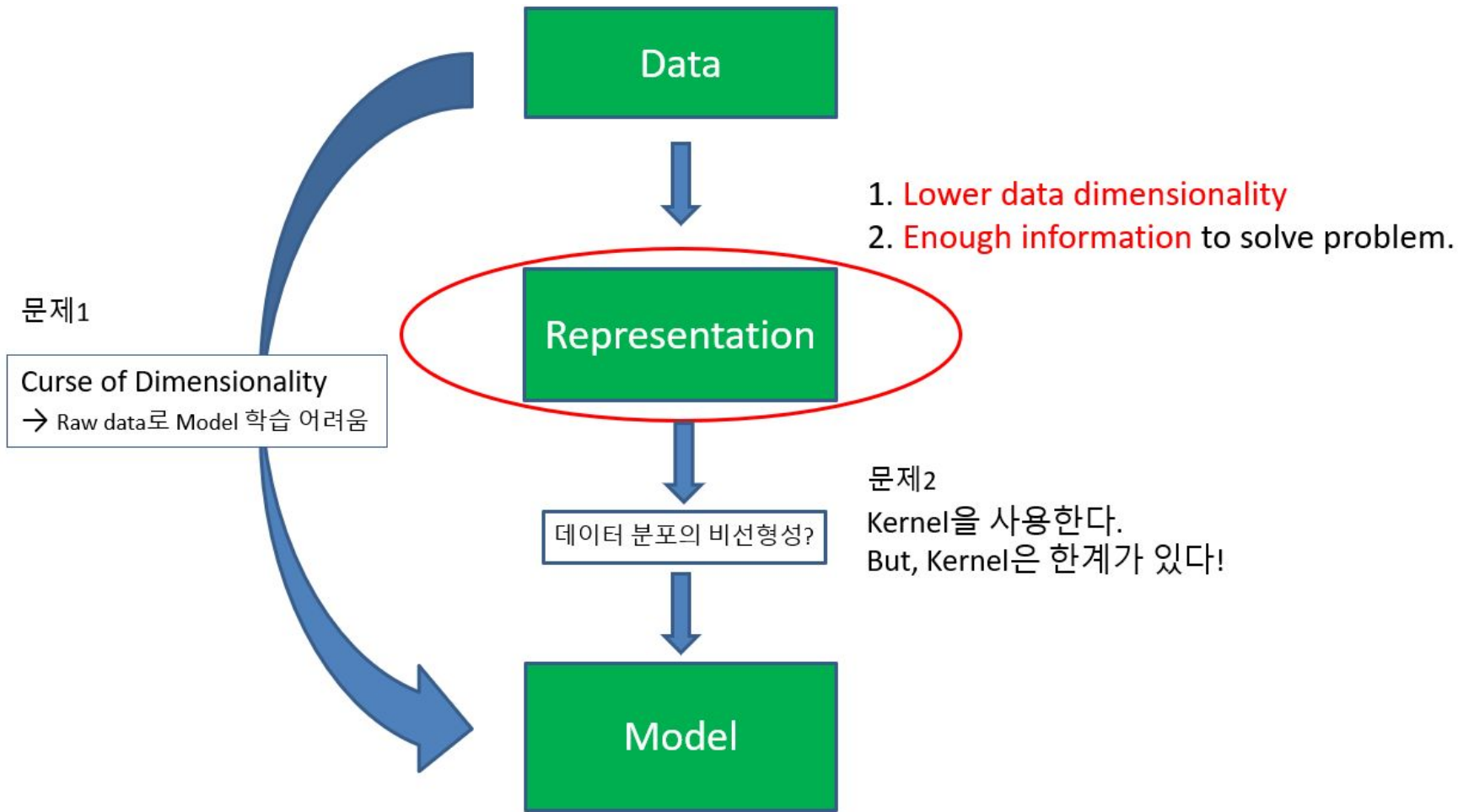


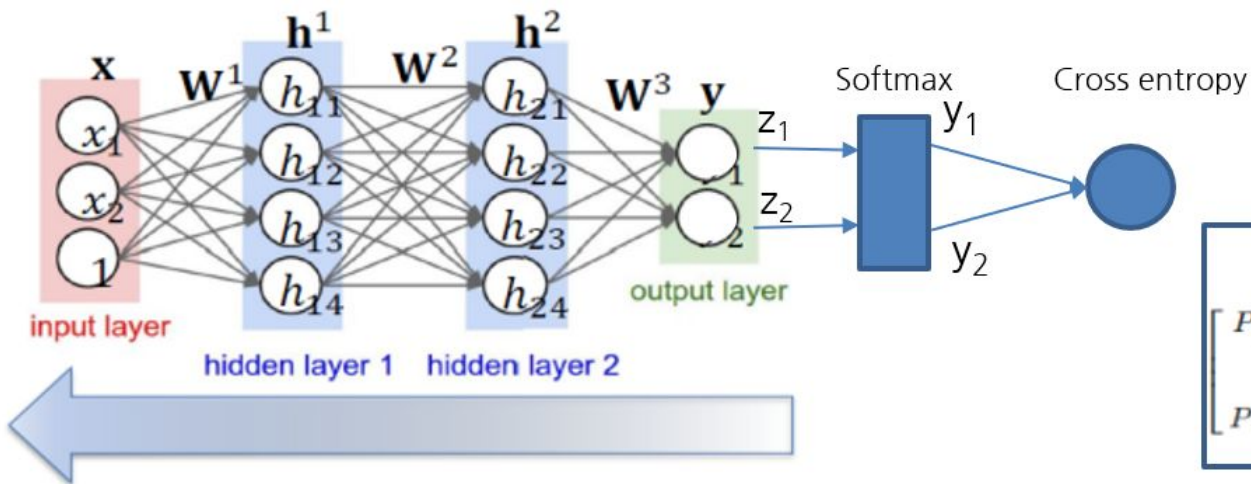
(a) Standard Neural Net



(b) After applying dropout.

Representation learning

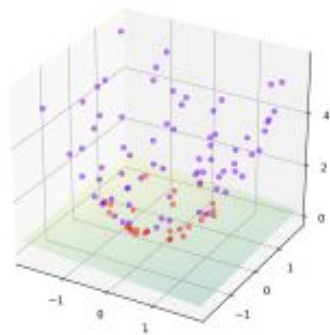
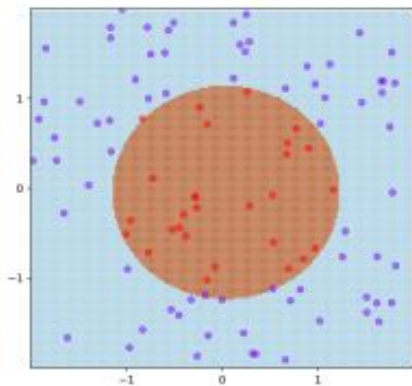




Softmax

$$\begin{bmatrix} P(t = 1|\mathbf{z}) \\ \vdots \\ P(t = C|\mathbf{z}) \end{bmatrix} = \begin{bmatrix} \varsigma(\mathbf{z})_1 \\ \vdots \\ \varsigma(\mathbf{z})_C \end{bmatrix} = \frac{1}{\sum_{d=1}^C e^{z_d}} \begin{bmatrix} e^{z_1} \\ \vdots \\ e^{z_C} \end{bmatrix}$$

Cross entropy : $-\sum t_i * \log y_i$



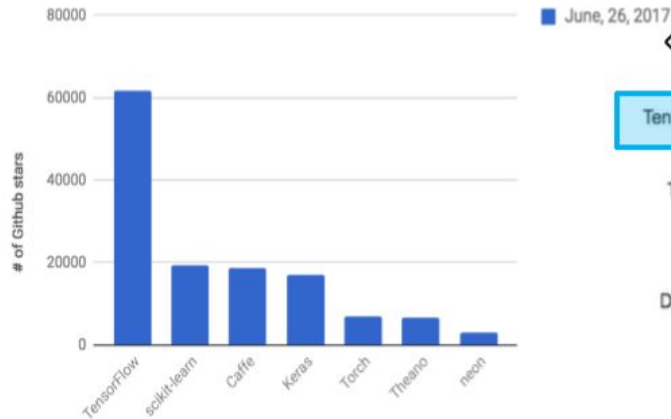
TensorFlow

Why TensorFlow?

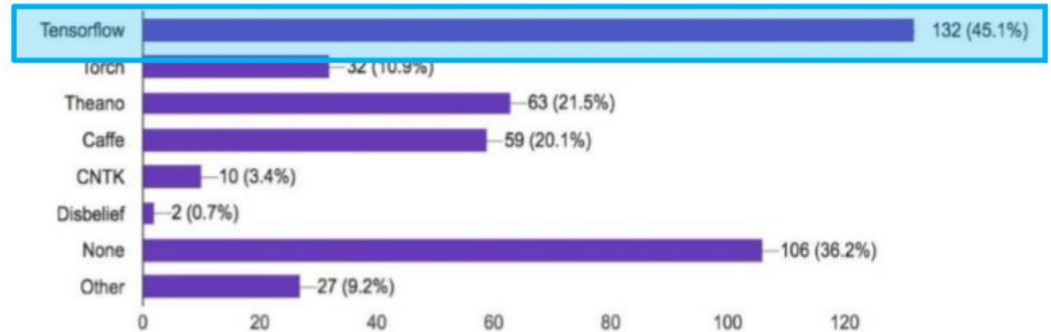
〈Various Software and Library for Deep Learning〉



Github Stars per Deep Learning Github Repository (as of June 2017)



〈Stanford Students' Deep Learning Library Usage Experience〉

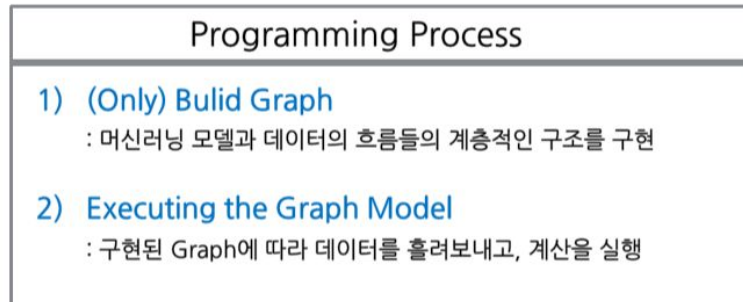


Why TensorFlow?

- Python API
- Portability : deploy computation to one or more CPUs or GPUs in a desktop, server, or **mobile device** with a single API
- Flexibility : from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- **Visualization** (TensorBoard)
- **Checkpoints** (for managing experiments)
- Auto-differentiation autodiff (no more taking derivatives by hand. Yay)
- **Large community** (> 10,000 commits and > 3000 TF-related repos in 1 year)

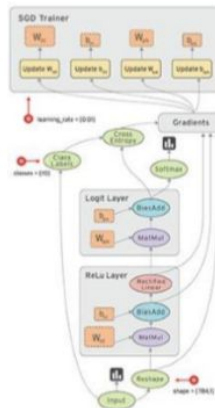
TensorFlow의 특징

- 2015년 10월 Google Brain Team이 개발하고 공개한 머신러닝/딥러닝 오픈소스 라이브러리 (Apach 2.0)
- “TensorFlow is an open source software library for numerical computation using data flow graphs.”
- 1개 이상의 CPU 또는 GPU를 다양한 디바이스에서 사용가능하도록 구조화되어 있음
- 1) **Building Dataflow Graph**, 2) **Executing the Graph Model** 두 단계를 통해 프로그램을 구현하고 실행함



Data flow Graphs

- TensorFlow = **Tensor(Data)** + **Flow**
 - Node: Computational Operation
 - Edge: Tensors



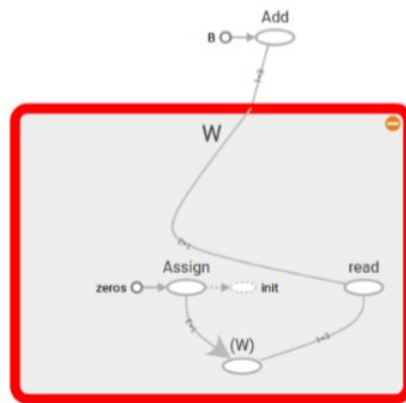
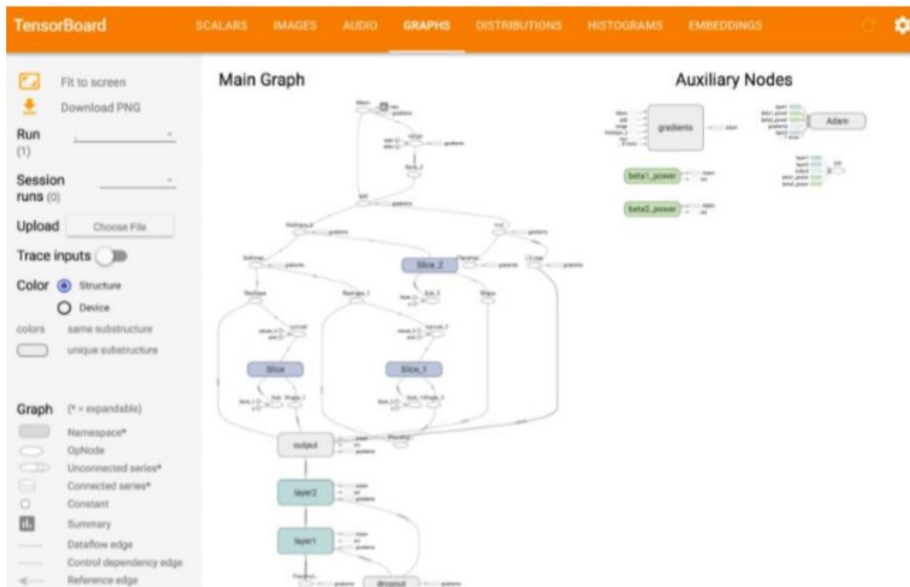
TensorFlow의 특징

1) (Only) Build Graph

: 머신러닝 모델과 데이터의 흐름들의 계층적인 구조를 구현

2) Executing the Graph Model

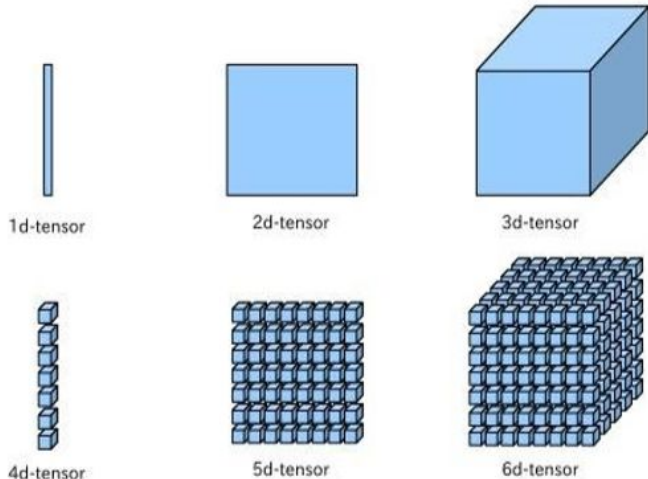
: 구현된 Graph에 따라 데이터를 흘려보내고, 계산을 실행



Tensor 개념

- TensorFlow 내에서 모든 데이터를 표현하는 기본 자료 구조 (본디 “**벡터의 확장 개념**”)
- 동적 크기를 갖는 다차원 데이터 배열으로써 진리형(boolean), 문자열, 정수형, 실수형 숫자 등 다양한 자료형의 데이터를 다룸
- **N-dimensional Matrix**

<Tensor Examples>

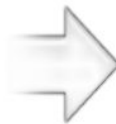
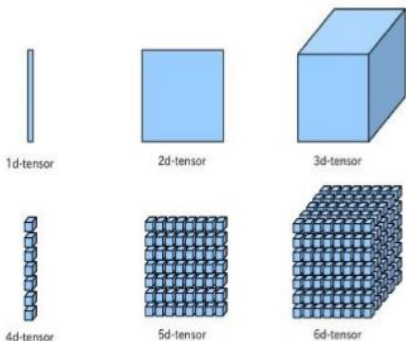


- 1D Matrix: Vector
- 2D Matrix: Matrix
- Above 3D Matrix: **Tensor!**

Tensor 개념

- 구조(Shape), 랭크(Rank), 차원번호(Dimension number) 세 종류의 명칭을 통해 텐서의 차원을 표기

<Tensor Examples>



Python Code	Rank	Shape	Math Entity
483	0	[]	Scalar
[1.1, 2.2, 3.3]	1	[3]	Vector
[[1, 2, 3], [4, 5, 6]]	2	[2, 3]	Matrix
[[[2], [4], [6]], [[8], [10], [12]]]	3	[2, 3, 1]	3-Tensor
...	n	[D0, D1, ..., Dn-1]	n-Tensor

Dimensi
on
number
0-D
1-D
2-D
3-D
n-D

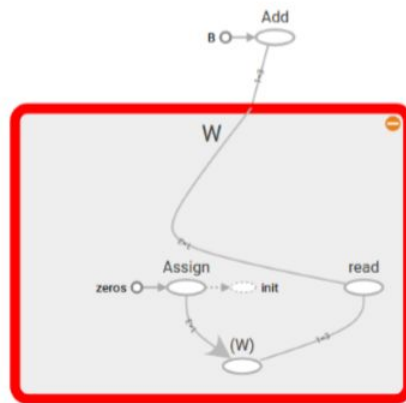
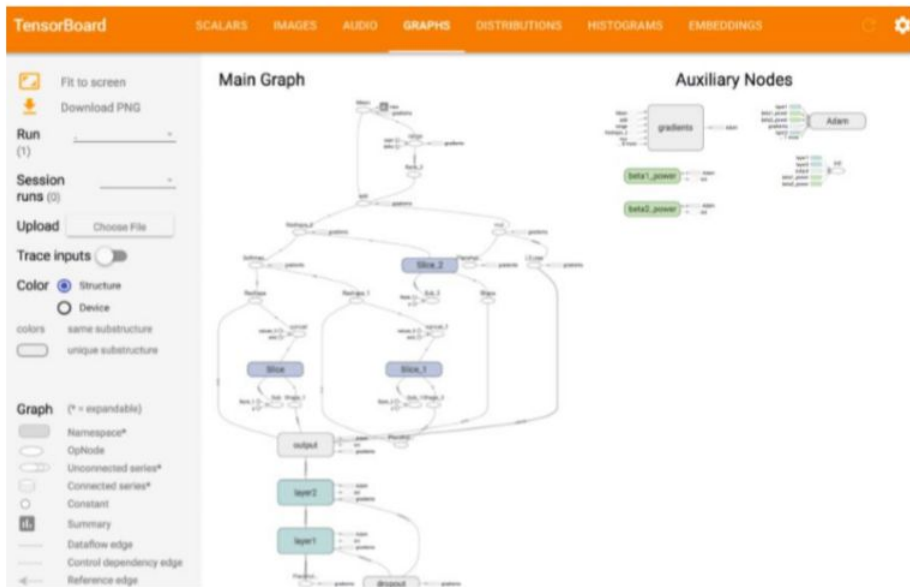
TensorFlow의 특징

1) (Only) Build Graph

: 머신러닝 모델과 데이터의 흐름들의 계층적인 구조를 구현

2) Executing the Graph Model

: 구현된 Graph에 따라 데이터를 흘려보내고, 계산을 실행



TensorFlow의 배우기

TensorFlow 자체적으로 다양한 기본 연산들을 제공

Operation	Description
<code>tf.add</code>	sum
<code>tf.sub</code>	subtraction
<code>tf.mul</code>	multiplication
<code>tf.div</code>	division
<code>tf.mod</code>	module
<code>tf.abs</code>	return the absolute value
<code>tf.neg</code>	return negative value

<code>tf.sign</code>	return the sign
<code>tf.inv</code>	returns the inverse
<code>tf.square</code>	calculates the square
<code>tf.round</code>	returns the nearest integer
<code>tf.sqrt</code>	calculates the square root
<code>tf.pow</code>	calculates the power
<code>tf.exp</code>	calculates the exponential
<code>tf.log</code>	calculates the logarithm
<code>tf.maximum</code>	returns the maximum
<code>tf.minimum</code>	returns the minimum
<code>tf.cos</code>	calculates the cosine
<code>tf.sin</code>	calculates the sine

TensorFlow의 배우기

Operation	Description
tf.add	sum
tf.sub	subtraction
tf.mul	multiplication
tf.div	division
tf.mod	module
tf.abs	return the absolute value
tf.neg	return negative value
tf.sign	return the sign
tf.inv	returns the inverse
tf.square	calculates the square
tf.round	returns the nearest integer
tf.sqrt	calculates the square root
tf.pow	calculates the power
tf.exp	calculates the exponential
tf.log	calculates the logarithm
tf.maximum	returns the maximum
tf.minimum	returns the minimum
tf.cos	calculates the cosine
tf.sin	calculates the sine

모든 함수의 종류와 사용법을 외우고
자유자재로 사용하는 것은
매우 숙련되지 않으면 사실상 불가능



구현시 필요한 결과에 따라
알맞은 함수를 검색 + 사용법 검색

ex) "python tensorflow linear regression"

TensorFlow의 배우기

tf.add

tf.add

```
add(  
    x,  
    y,  
    name=None  
)
```

Tensorflow 홈페이지에 모든 정보가 존재
→ 공식 문서와 친해지는 것이 중요

<https://www.tensorflow.org/>

Defined in tensorflow/python/ops/gen_math_ops.py .

See the guide: [Math > Arithmetic Operators](#)

Returns $x + y$ element-wise.

NOTE: Add supports broadcasting. AddN does not. More about broadcasting [here](#)

Args:

- `x`: A Tensor . Must be one of the following types: half , float32 , float64 , uint8 , int8 , int16 , int32 , int64 , complex64 , complex128 , string .
- `y`: A Tensor . Must have the same type as `x` .
- `name`: A name for the operation (optional).

Returns:

A Tensor . Has the same type as `x` .

Thank you!