Lecture note 6 : Deep Learning Theory

프로젝트 기반 딥러닝 이미지처리
한국인공지능아카데미 x Hub Academy

강사 : 김형욱 (hyounguk1112@gmail.com)

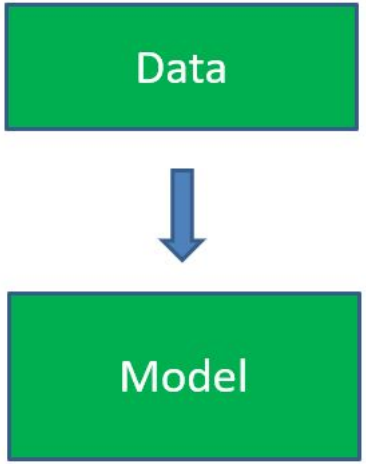
Outline

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.

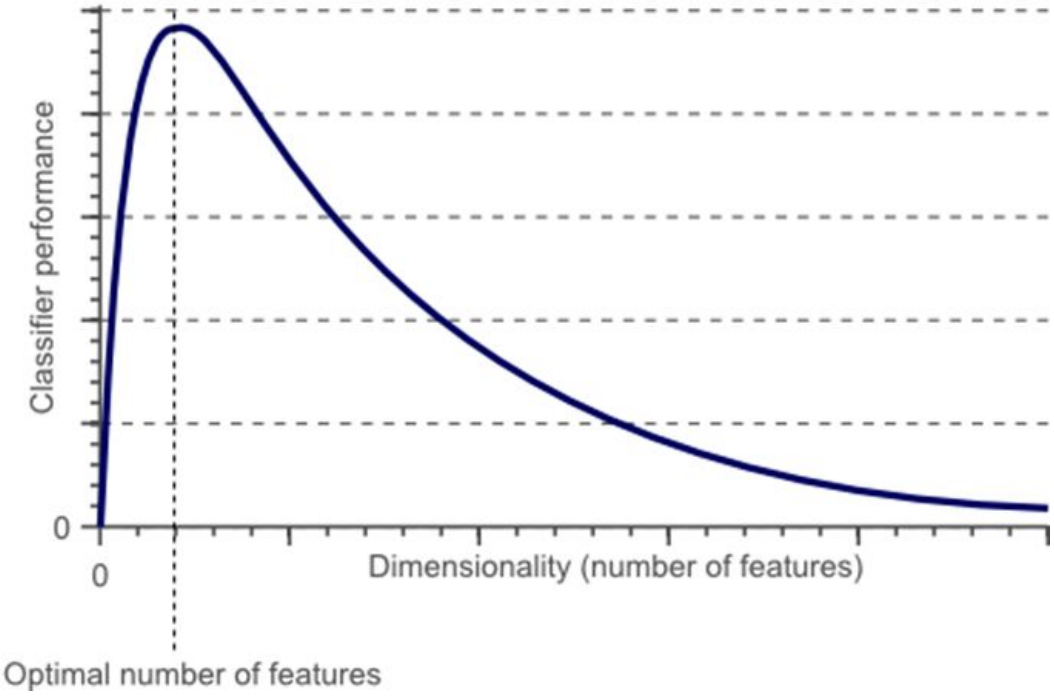


전통적인 머신러닝 알고리즘은 다음과 같은 문제를 갖는다.

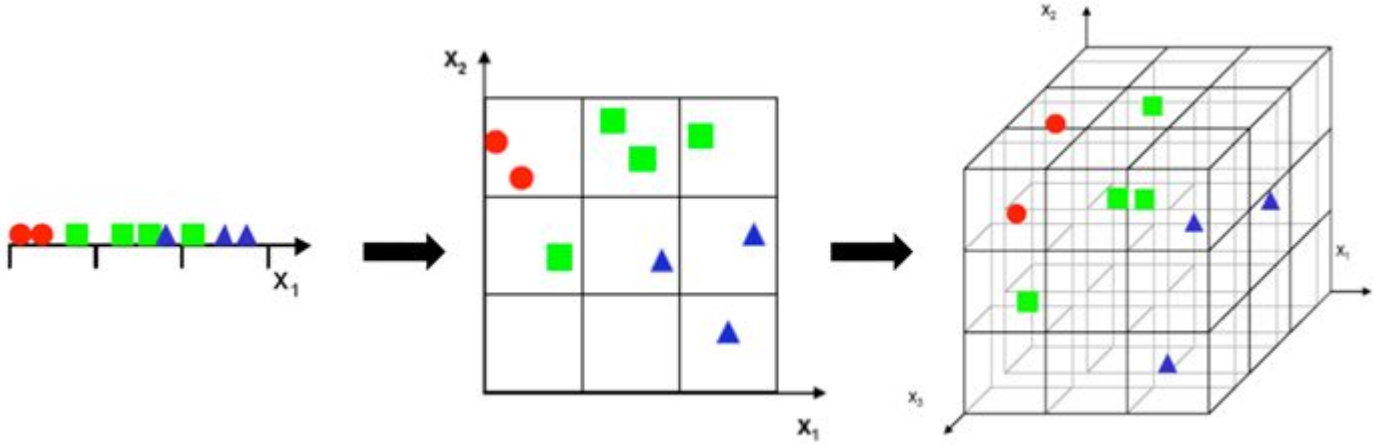
1. Data의 차원이 높을 때 차원의 저주(Curse of dimensionality)가 발생한다.
2. 데이터의 전처리(preprocessing)에 따라 성능이 크게 좌우된다.
3. 데이터가 비선형적으로 분포할 때, 선형 모델로 다루는데 한계를 갖는다.

머신러닝의 한계와 딥러닝

1. 차원의 저주 문제

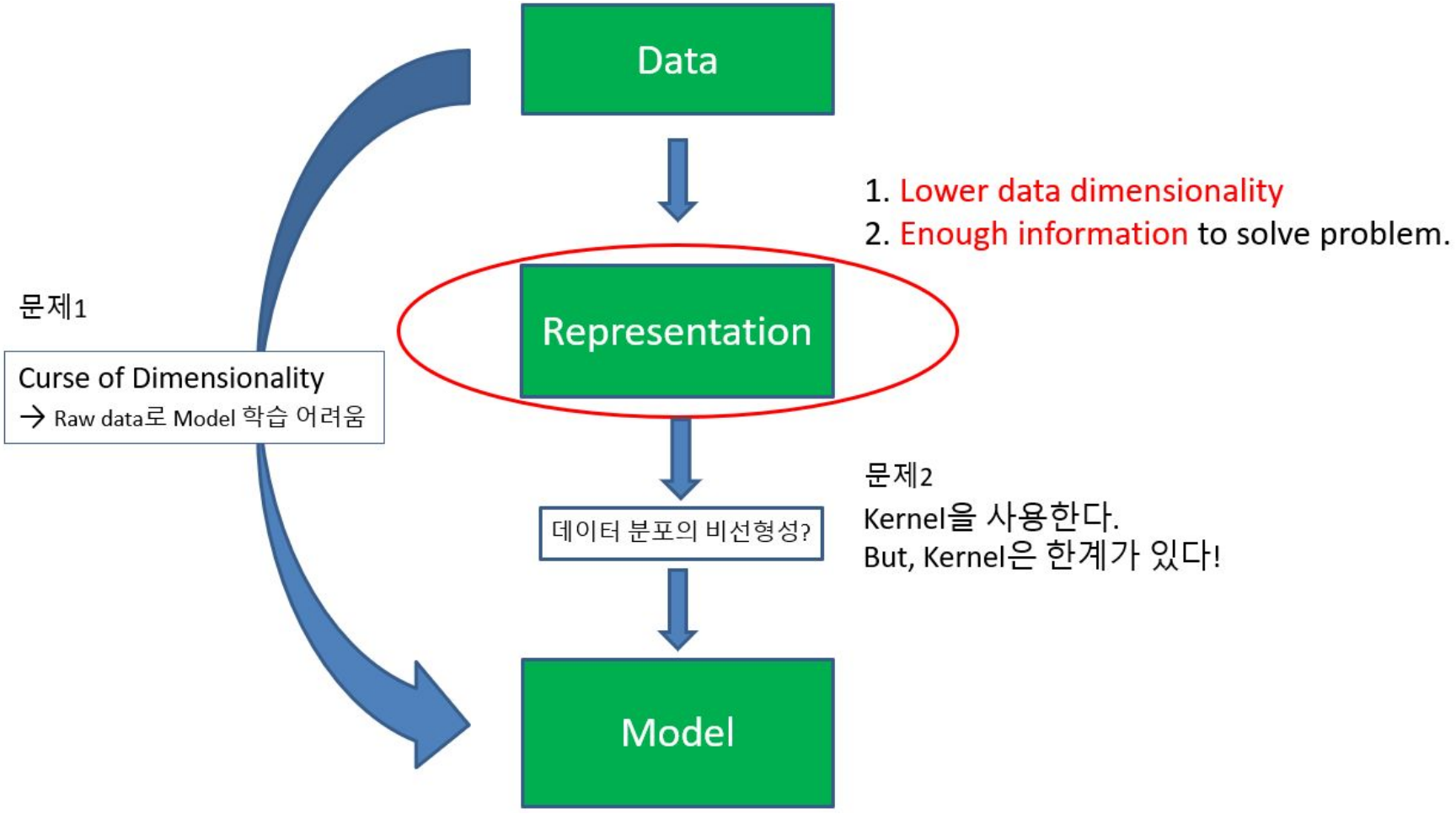


머신러닝의 한계와 딥러닝



데이터의 차원이 증가함에 따라 모델이 예측을 위해 알아야할 데이터의 수가 지수적으로 증가한다.

현실적으로 모델에 필요한 데이터를 충족할 수 없음.



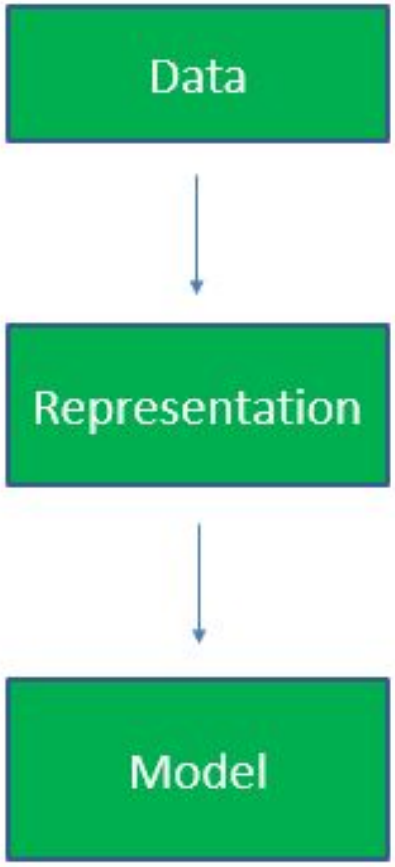
1. Lower data dimensionality
2. Enough information to solve problem.

문제1

Curse of Dimensionality
→ Raw data로 Model 학습 어려움

문제2

Kernel을 사용한다.
But, Kernel은 한계가 있다!

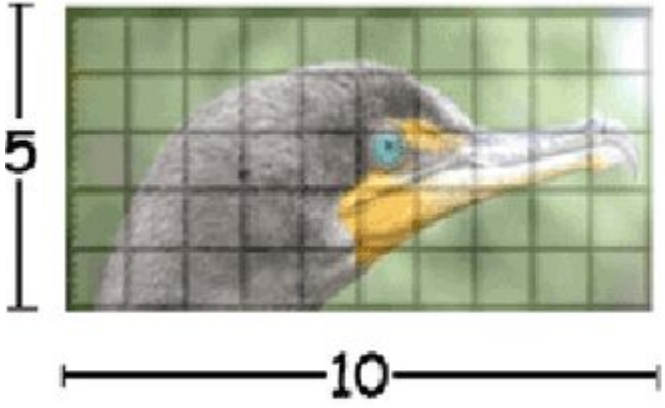


Images : High dimensional data

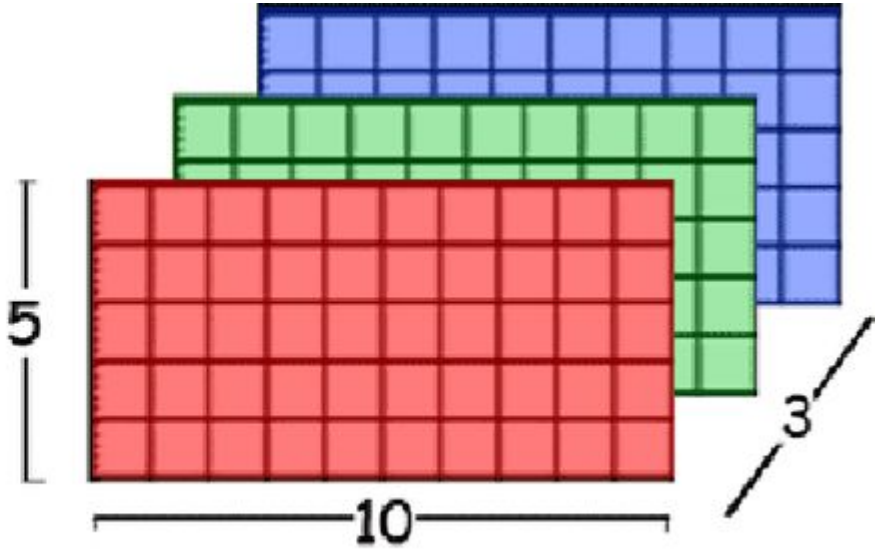
Representation : Number of corners
Feature engineering (Important, but high cost)

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).

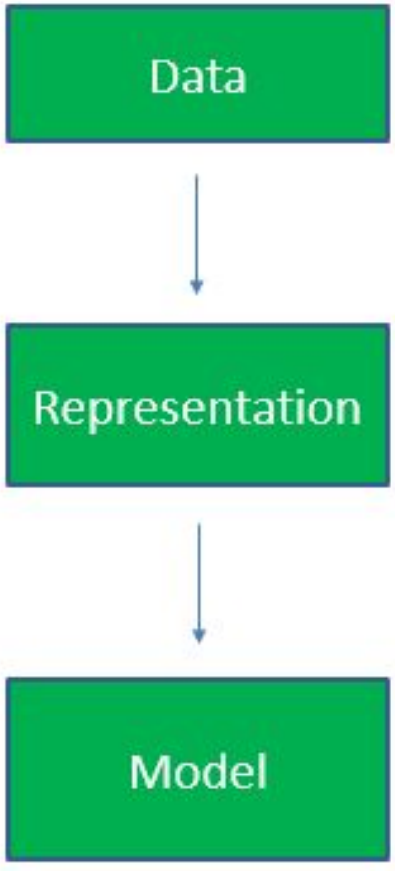
Colored digital image representation



Original Color Image



Matlab RGB Matrix



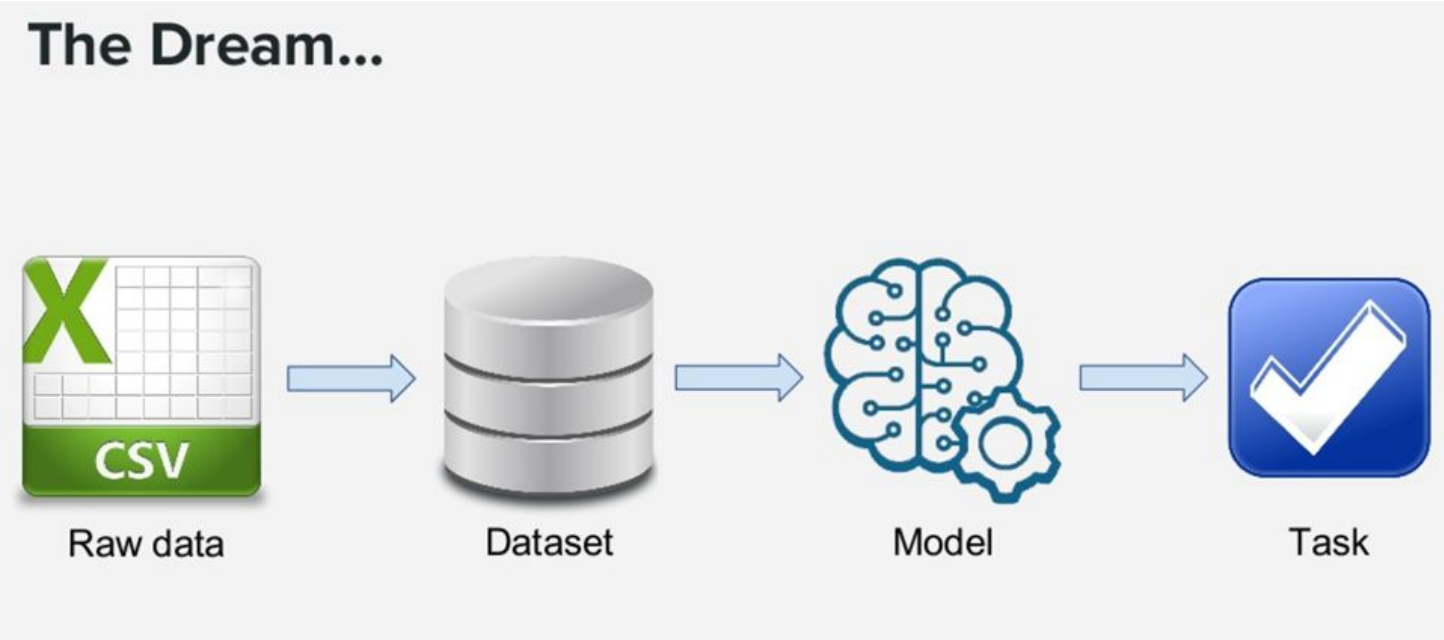
Images : High dimensional data

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).

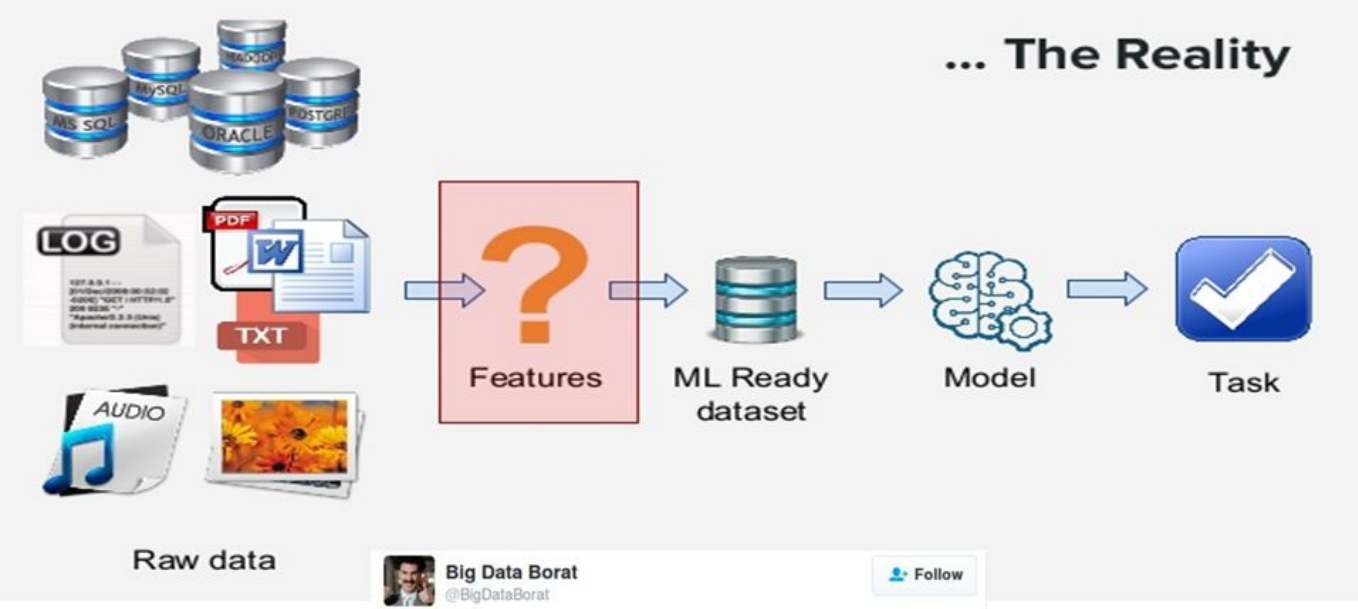
머신러닝의 한계와 딥러닝

2. 데이터 전처리에 따른 성능 변화 문제



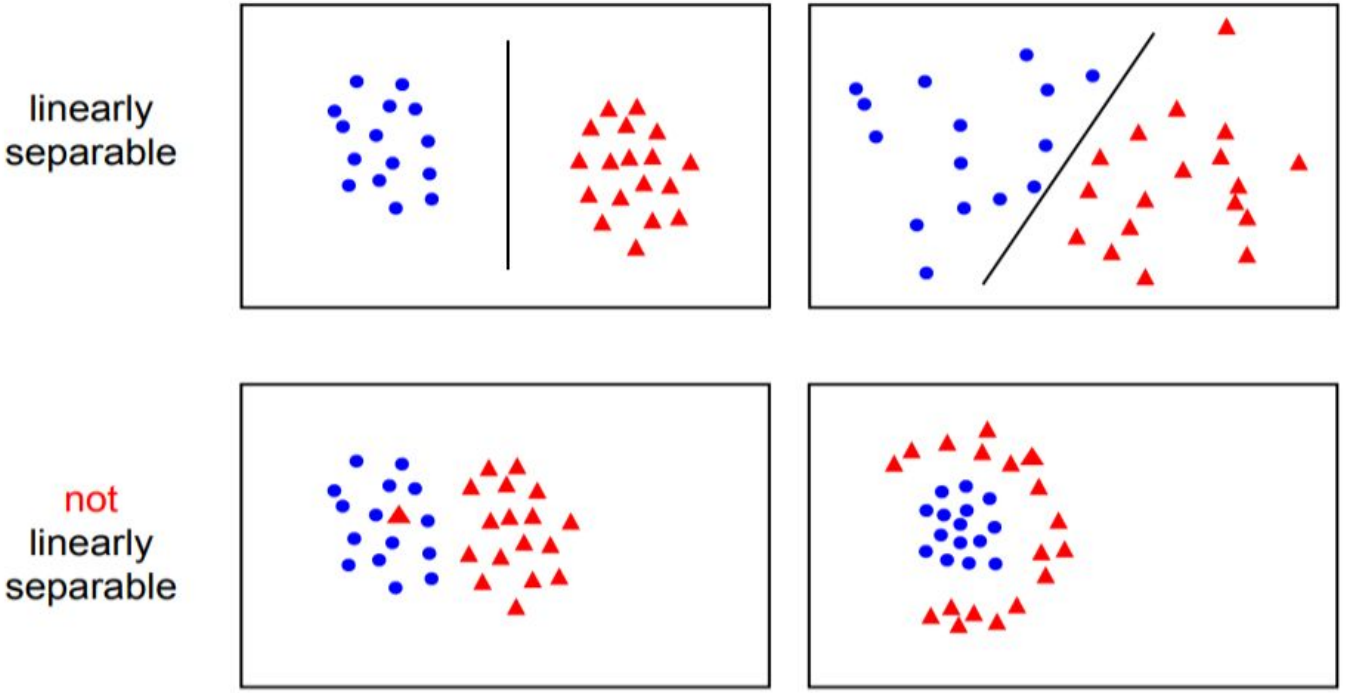
머신러닝의 한계

2. 데이터 전처리에 따른 성능 변화 문제

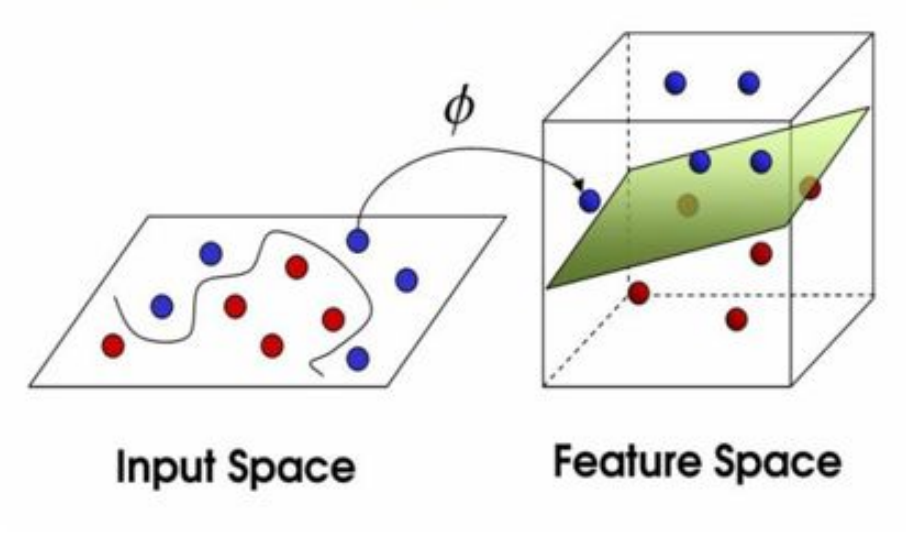


머신러닝의 한계와 딥러닝

3. 데이터 비선형성 문제



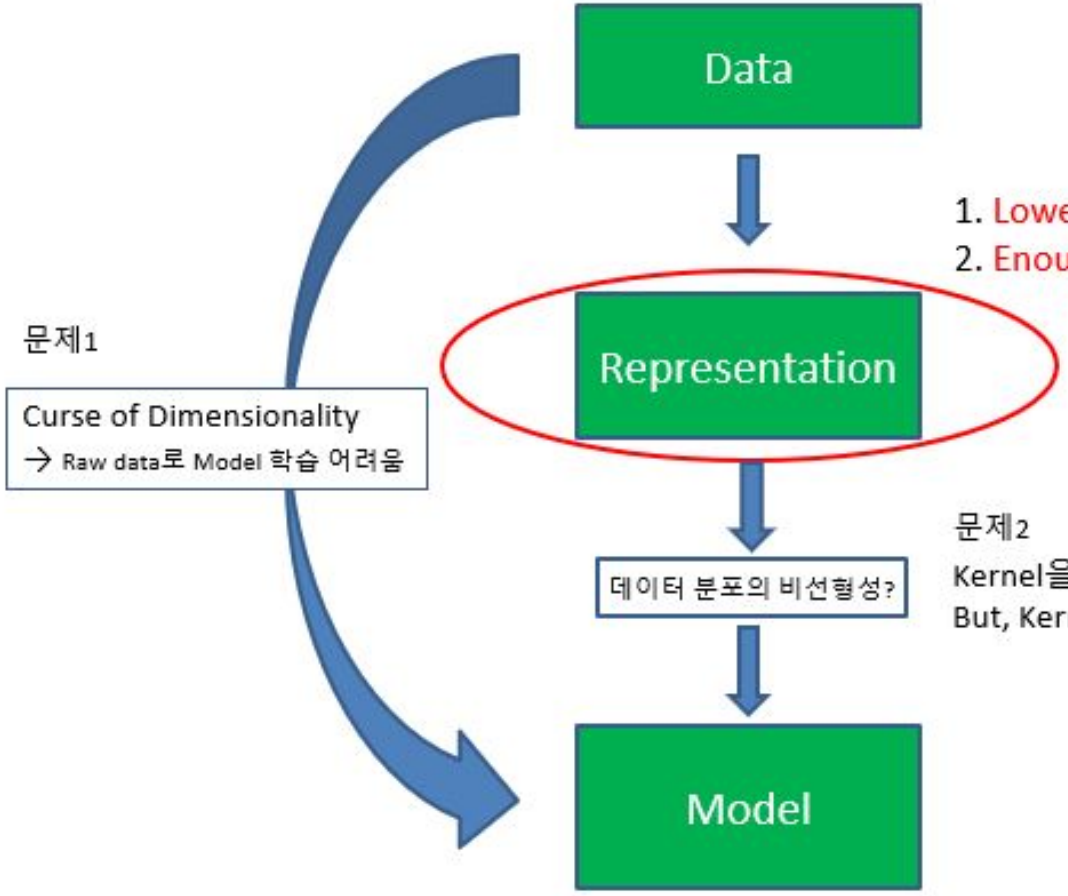
데이터를 선형 분포로 만드는 Kernel 함수를 사용하면 어떨까?



하지만 우리가 Not linearly separable한 데이터 문제를 해결하기 위한 Kernel를 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!



1. Lower data dimensionality
2. Enough information to solve problem.

문제2
Kernel을 사용한다.
But, Kernel은 한계가 있다!

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**)*

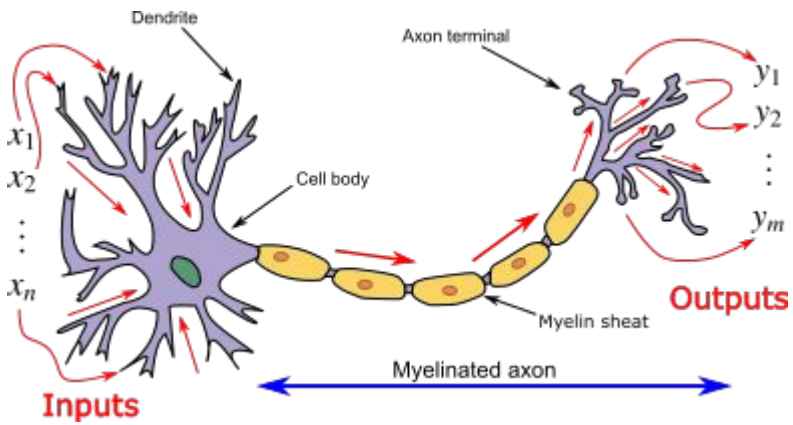
2. 비선형 함수를 포함하는 신경망 모델을 경사하강법에 의해 학습할 수 있도록 하는 오류역전파(Back-propagation) 알고리즘이 고안됨.

3. 데이터가 가지고 있는 특성(영상, 텍스트, 음성, 시계열)을 활용하여 분석할 수 있는 신경망 구조가 고안됨.

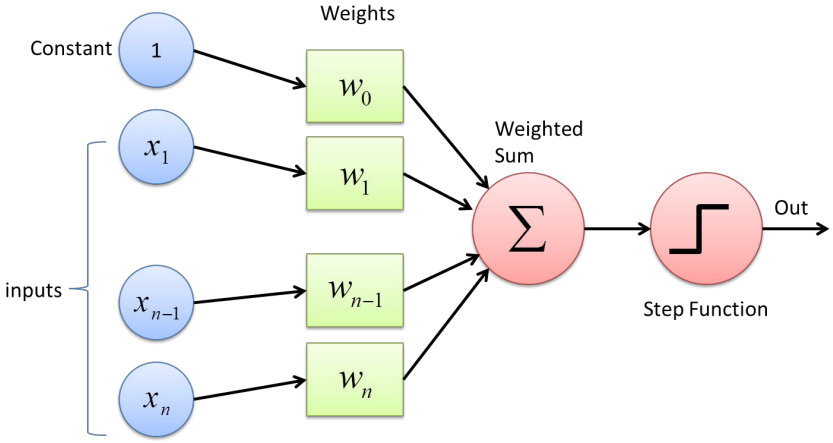
4. 딥러닝은 학습데이터셋의 수가 많아도 제약 없이 활용이 가능하며, 높은 복잡도의 모델(인공신경망)을 사용하기 때문에 방대한 데이터셋을 사용함.

Neural Network

인공신경망 모델 사용



생물학적 뉴런의 구조



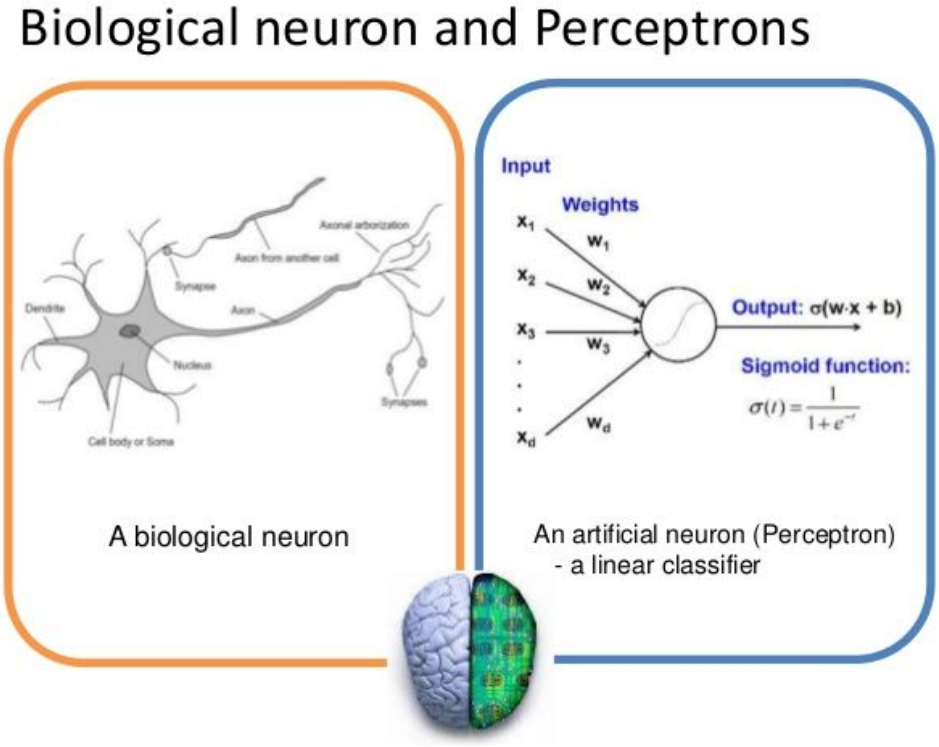
퍼셉트론 모델

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

[1] 좌측그림 : https://en.wikipedia.org/wiki/Biological_neuron_model
[2] 우측그림 : ttps://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

퍼셉트론의 모델

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 활성함수(Activation function)이라는 일종의 스위치를 사용한다.

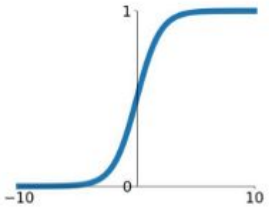
Image ref : https://www.slideshare.net/jbhuang/lecture-29-convolutional-neural-networks-computer-vision-spring2015

활성함수(Activation function)

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

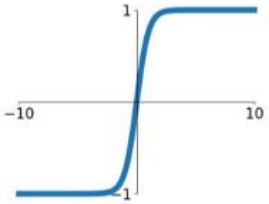
Sigmoid

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



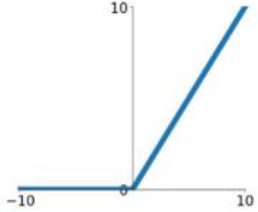
tanh

$$\tanh(x)$$



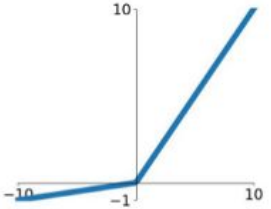
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

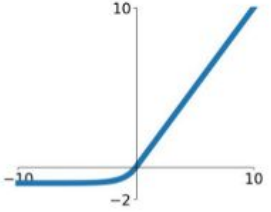
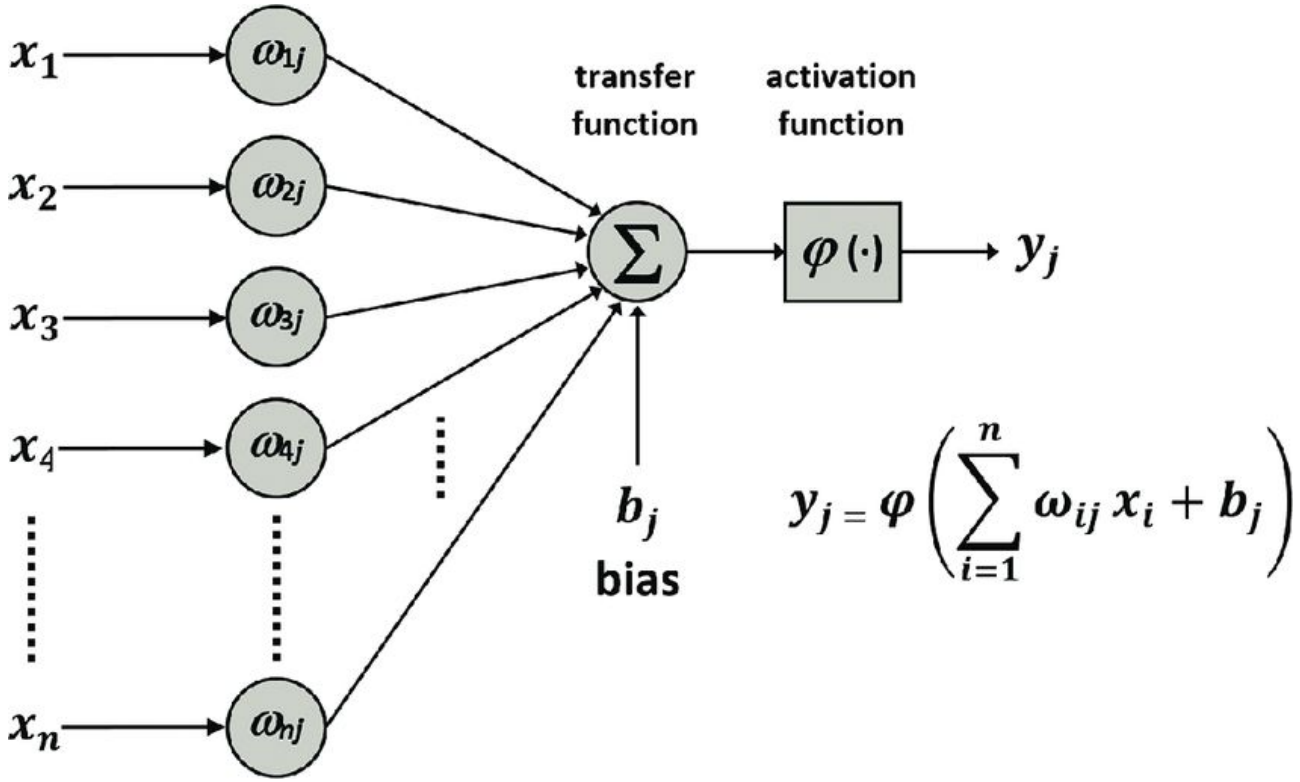


Image ref : https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

최종적으로 퍼셉트론은 1) 입력된 x의 값들에 대한 가중치합(Linear combination)과 2) 활성함수로 구성됨.



MLP(Multi-Layer Perceptron)

하나의 퍼셉트론으로는 표현할 수 있는 결정 영역은 선형적이나, 다수의 퍼셉트론을 계층구조로 쌓는 다층 퍼셉트론(MLP)는 더 복잡한 결정 영역을 갖을 수 있다.
(Neural Network playground)

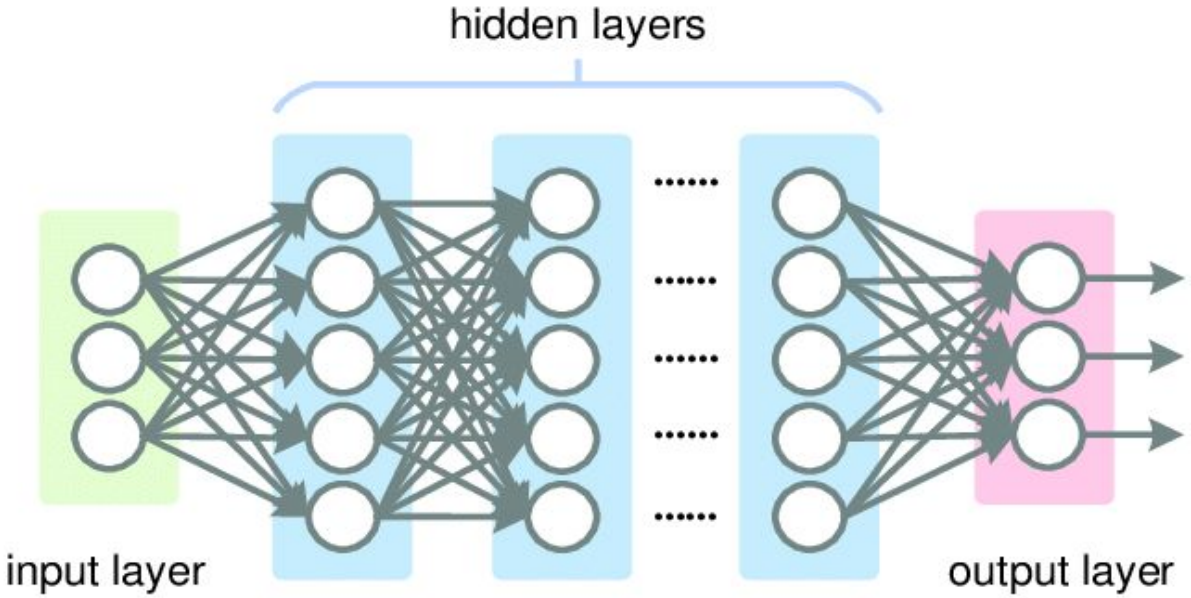


Image ref : https://community.intersystems.com/post/run-deep-learning-demo-python3-binding-healthshare-part-ii

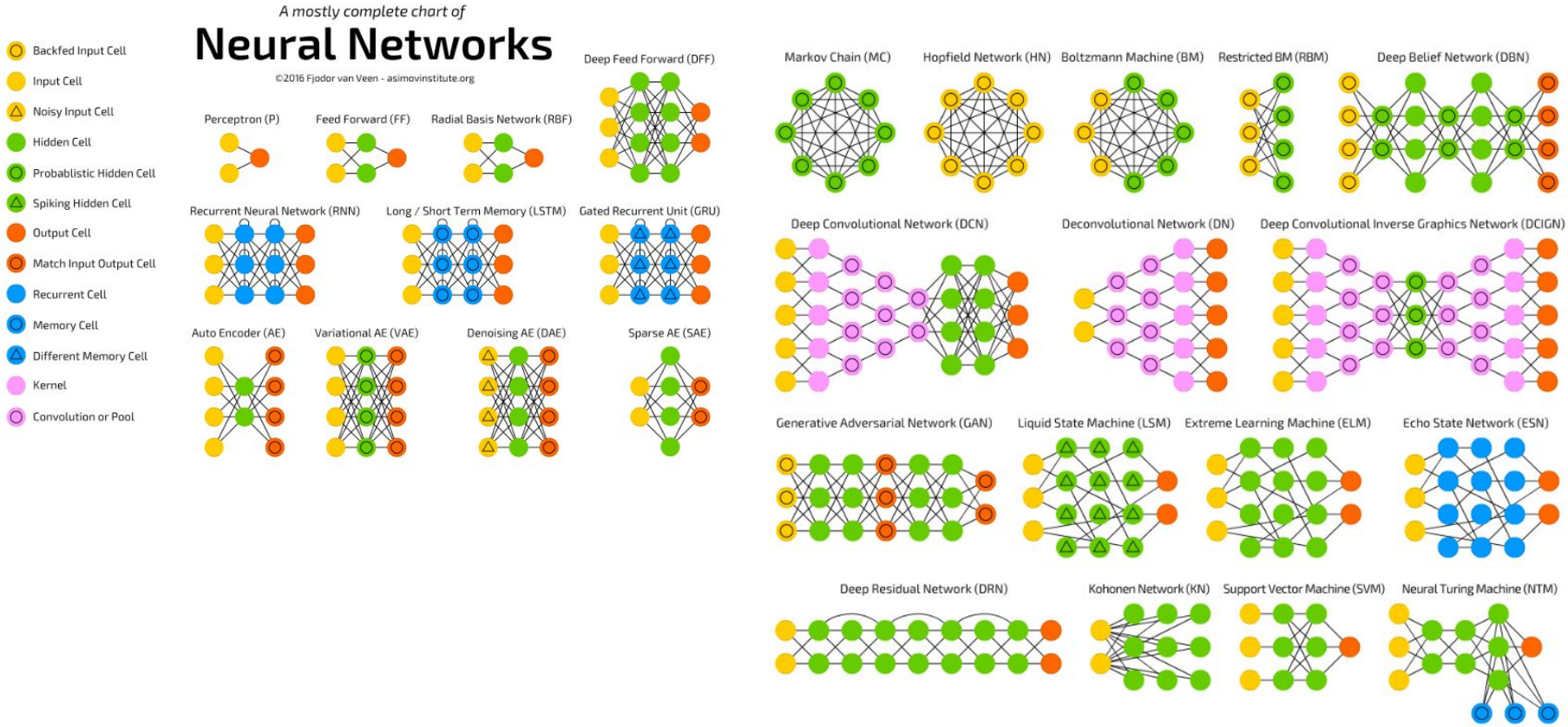
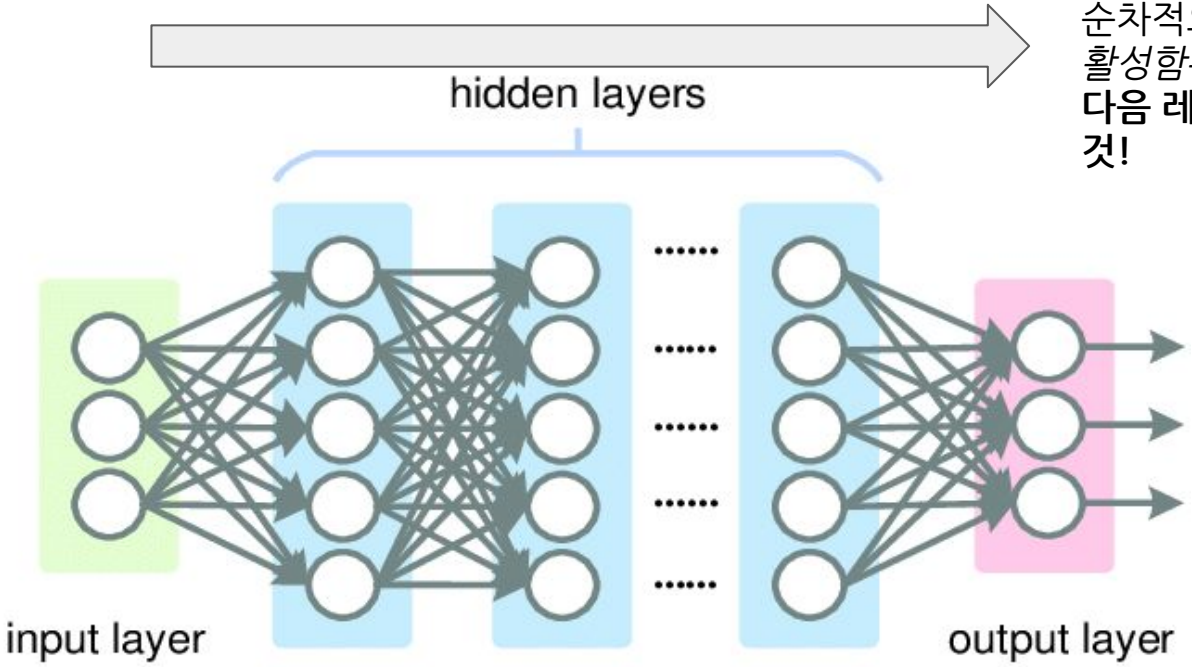


Image ref : https://www.asimovinstitute.org/neural-network-zoo/

인공신경망의 계산방식

1. Forward propagation



Input layer의 입력 데이터에 대해 각 레이어가 순차적으로 모든 퍼셉트론에 대해 *가중치합*과 *활성함수 계산*한 결과인 **새로운 표상정보를 다음 레이어로 보내 output layer까지 보내는 것!**

Image ref : https://community.intersystems.com/post/run-deep-learning-demo-python3-binding-healthshare-part-ii

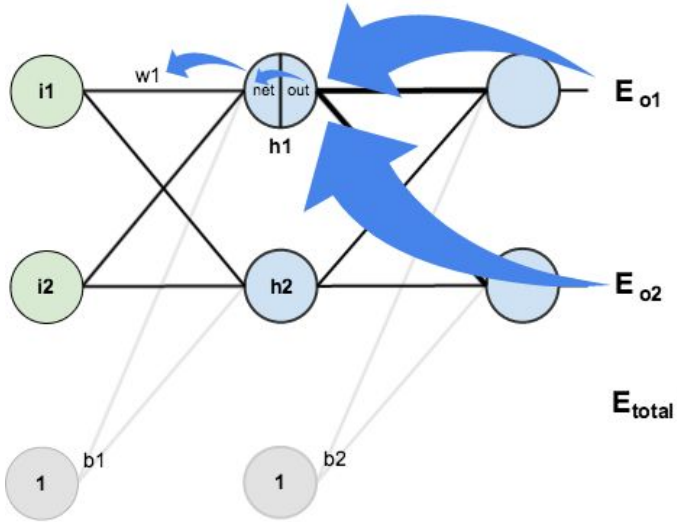
인공신경망의 계산방식

2. Back-propagation (오류역전파)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

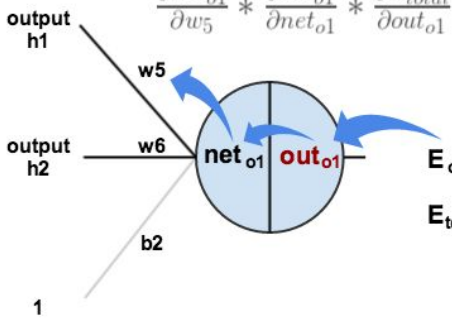
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$

Chain rule

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



Back-propagation : Forward propagation과 반대로 Loss function의 에러부터 첫 번째 hidden layer의 각 퍼셉트론의 파라미터까지 gradient(미분값)을 전달하는 것. 이때 미분의 Chain rule이 사용되므로, 네트워크의 모든 레이어는 입력 범위 내에서 모두 미분가능한 함수여야 한다.

A Step by Step Backpropagation Example | Matt Mazur

Optimization

인공신경망의 학습

인공신경망의 학습은 4가지의 주요 파트가 존재한다.

1) **Initialization** : 네트워크의 학습하려는 파라미터의 초기값을 선정한다.
2) **Cost function** : 파라미터를 문제(데이터)에 대해 정량적으로 평가하기 위한 함수를 정의한다. 일반적으로 비용이 최소화되도록 파라미터를 업데이트하는 방식으로 학습한다.
3) **Optimizer** : 경사하강법을 통해 파라미터를 업데이트할 때 좀 더 안정적이고, 일반화할 수 있는 파라미터를 학습하기 위한 파라미터 업데이트 알고리즘을 다룬다.
4) **Regularization** : 더 단순한 모델을 사용하는 대신 모델의 과적합 문제를 줄여줄 수 있는 테크닉들을 다룬다.

Neural Network learning process

초기값 설정(Initialization)

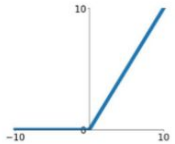
인공신경망의 경우 대표적인 Non-convex function으로, 초기값 설정에 따라 학습 속도와 학습 가능 여부가 달라질 수 있다는 사실이 딥러닝 초기 연구에서 Geoffrey Everest Hinton에 의해 발견되었다.

(1) 모든 파라미터를 0으로 초기화 : 가장 나쁜 경우이다. 이 경우 모든 레이어는 입력값과 상관 없이 결과로 값이 0인 벡터를 다음 레이어로 전달할 것이다. Gradient는 모든 파라미터에 대해 0이 되므로 학습이 되지 않는다.

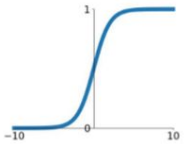
(2) 모든 파라미터를 랜덤하게 초기화 : 파라미터가 정규 분포로부터 샘플링한 아주 랜덤한 값을 갖도록 초기화한다. 이 경우 활성함수를 Sigmoid나, Tanh의 경우 파라미터가 작아지거나, 커질수록 backpropagation 알고리즘의 gradient(미분값)이 전달하는 과정에서 점차 작아져 소멸하는 현상(Vanishing gradients)이 일어난다. 반대로 경우에 따라 점차 gradient가 계속해서 증가하는 현상(Exploding gradients)의 위험도 있다.

모든 파라미터를 랜덤하게 샘플링하여 초기화하되 가급적 활성함수는 RELU를 사용
-> 비선형 함수지만 선형적인 특성을 가지고 있어 Vanishing gradient, Exploding gradient 문제에 강건한 특성을 가짐.

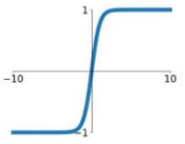
ReLU
$\max(0, x)$



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



tanh
$\tanh(x)$



(1) 활성함수를 Sigmoid함수나 Tanh를 사용하는 경우

Xavier initialization 사용 : 많은 연구를 통해 이전 레이어와 다음 레이어의 퍼셉트론 개수로 파라미터를 샘플링하기 위한 분포를 찾아내었다.

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

n_{in} : 이전 layer(input)의 노드 수

n_{out} : 다음 layer의 노드 수

(2) 활성함수가 Relu를 사용하는 경우

He initialization 사용 : Xavier initialization은 비선형함수 (Sigmoid, Tanh)에서는 효과적인 결과를 보여주나 ReLU에서는 출력값이 0으로 점차 수렴하는 현상을 보였다. 이에 따라 ReLu에 적합한 분포를 찾아내었다.

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

n_{in} : 이전 layer(input)의 노드 수

Neural Network learning process

비용 함수(Cost function)

Regression

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

makes result quadratic

y_i: true y

$\hat{y}_i$: estimate of y

$\frac{1}{N}$: average over all results

Classification

$$H(x) = \sum_{i=1}^{N} p(x) \log q(x)$$

Cross-entropy

$p(x)$: true label

$q(x)$: estimate

Neural Network learning process

경사하강법(Gradient descent method)

(1) Batch gradient descent

파라미터를 업데이트하기 위해 cost function의 gradient를 구할 때, 모든 데이터에 대해 계산한다. 이 방식은 데이터가 방대한 경우 Memory의 부족을 발생시킬 수 있음. 또한 동일한 cost function에 대해 미분하기 때문에 local minima에 가능성이 다른 방법에 비해 높다.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

(2) Stochastic gradient descent

Batch gradient descent와는 대조적으로 학습데이터 각 한 개씩에 대해서 파라미터를 업데이트하는 방식이다. 가장 빠른 방법이지만, 각 데이터에 대해 모델이 지나치게 fitting되는 단점이 존재한다.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

(3) Mini-batch gradient descent

두 방법을 보완한 것으로 가장 보편적이며 좋은 학습 성능을 보인다. 전체 데이터 분포를 반영할 수 있을 정도의 데이터 n개 만큼씩 학습시키는 방식이다.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

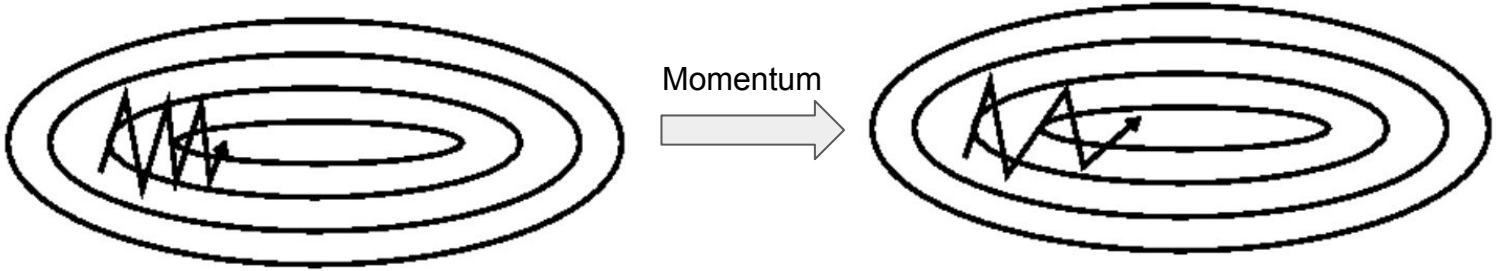
Neural Network learning process

경사하강법(Gradient descent method)

Mini-batch gradient descent 방식의 학습도 항상 global optimum에 가까운 좋은 수렴 결과를 보장하지 않는다. 따라서 알고리즘적으로 개선한 Mini-batch gradient descent을 사용한다면 여러 측면에서 학습 과정을 개선할 수 있다.

1. Momentum

경사하강법은 한 파라미터의 차원이 다른 파라미터에 비해 훨씬 가파르게 감소하는 지역에서 경사를 탐색하는데 어려움을 갖는다. Momentum은 파라미터 업데이트에 일종의 관성을 부여하여 학습 속도를 개선한다.



Neural Network learning process

경사하강법(Gradient descent method)

2. Adagrad

초기에 학습률을 크게하다가 점차 작게 변화시켜 안정적인 수렴이 될 수 있도록 스케쥴링한다. 또한 모든 파라미터에 동일한 학습률을 적용하는 것이 아닌 각각의 매개변수에 맞는 학습률을 조정한다. 구체적으로 학습이 많이 된 파라미터일수록 기울기를 누적하여, 학습률을 감쇄함으로써 파라미터의 갱신 크기가 약해진다.

3. RMSprop

Adagrad의 단점은 학습률이 누적되는 방식으로 감쇄하기 때문에 복잡한 non-convex function의 지형에서는 급진적 수렴해버린다는 것이다. RMSprop은 앞선 몇 개의 학습 iteration에서 구했던 기울기들에 대한 이동평균(moving average)를 업데이트를 위해 사용함으로써 Adagrad처럼 단조 감소하지 않도록 개선하였다.

3. ADAM

ADAM은 비교적 최근에 제안된 방식으로 RMSprop과 Momentum 방식을 혼합하였다. 가장 보편적으로 사용되는 최적화 알고리즘이다.

Neural Network learning process

Avoiding Overfitting : Regularization term

Cost function에 Parameter에 대한 penalty term을 추가시킨다. 여기서 parameter는 weight을 의미.

다음은 Regularization term을 더해준 새로운 Cost function이다.

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

Weight 값에 penalty를 주지만 Bias에는 주지 않는다.
(Underfitting 유도, penalty 줘도 영향력 거의 X)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

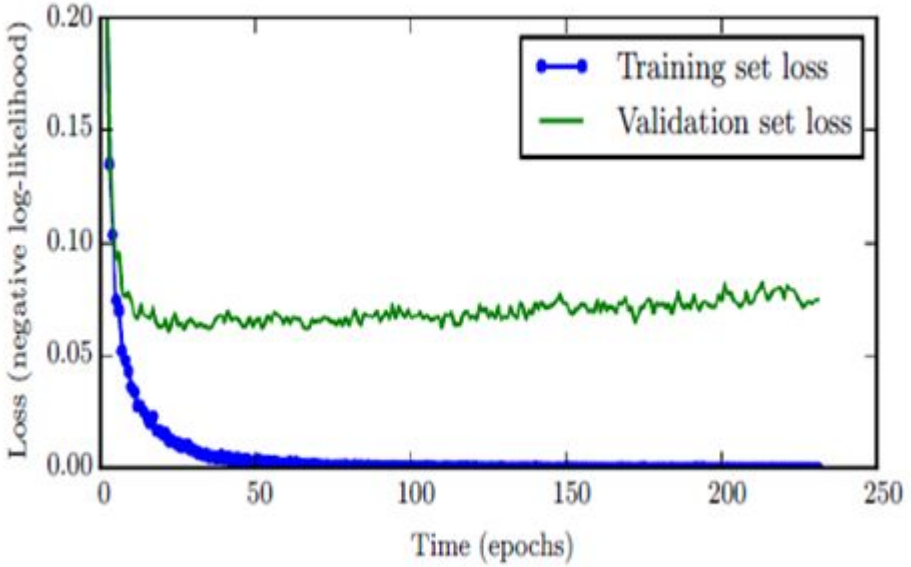
Neural Network learning process

Avoiding Overfitting : Early stopping

Capacity가 큰 모델은 시간이 가면 갈 수록 training error는 감소하지만 Validation error는 감소하다가 다시 증가함을 확인할 수 있다.

Validation error가 가장 낮은 시간의 모델 parameter를 얻으면 좋음.

이 시간(Learning step)이 Hyper parameter로 볼 수 있다.



Neural Network learning process

Avoiding Overfitting : Dropout

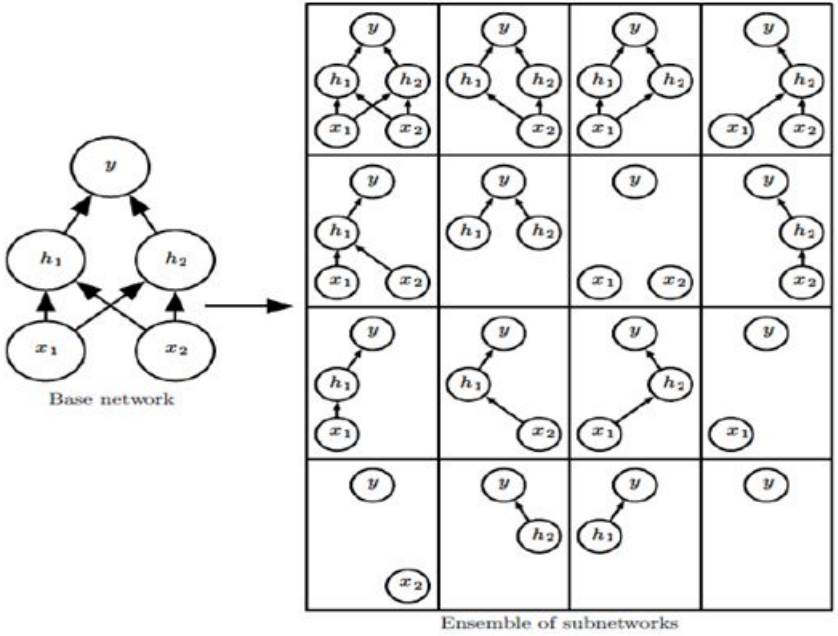
Dropout은 연산이 많지 않지만 성능이 좋은 방법이다.

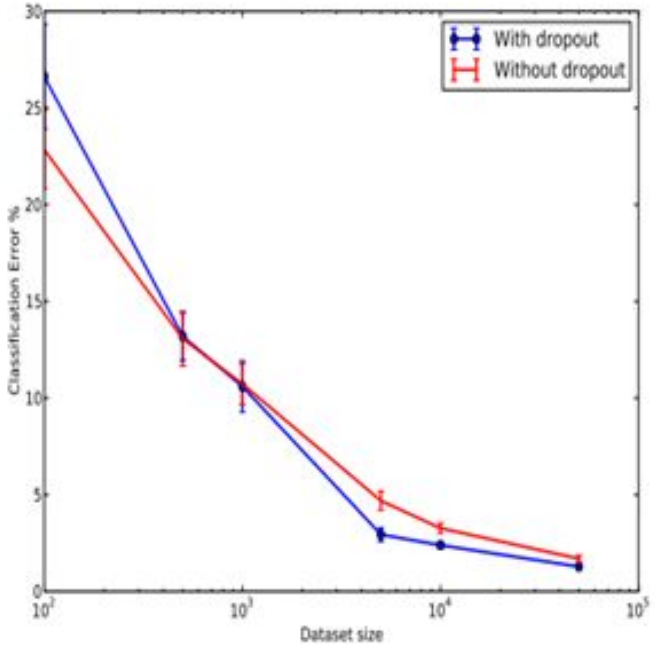
전체 네트워크의 파라미터들에 Hyper-parameter로 지정한 비율만큼 랜덤으로 0을 곱하여 네트워크를 끊어서 학습시킨다. 0이 곱해진 Unit은 원래 basic network에서 일시적으로 제거되었다고 볼 수 있다.

매번 다른 방식으로 랜덤하게 네트워크의 일부 파라미터를 제거하므로 무수한 Sub-network를 만들어서 학습시키는 효과를 갖는다. 이는 일종의 앙상블 효과로 Bagging은 k개의 앙상블이 생기는 반면, Dropout은 파라미터의 수(n개)에 따라 exponential하게 sub-network(2^n개)가 생긴다.

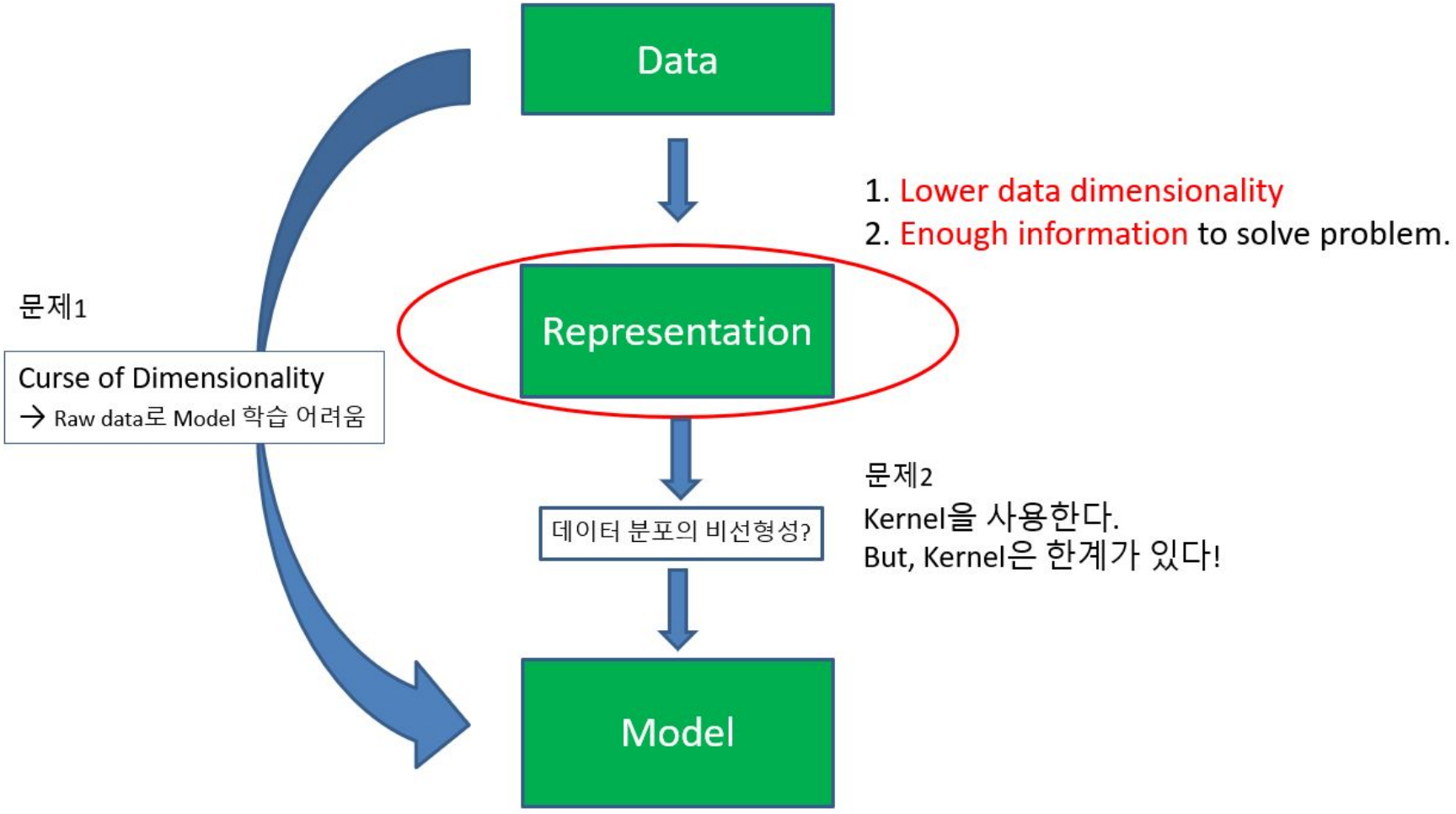
Neural Network learning process

Avoiding Overfitting : Dropout





Representation learning



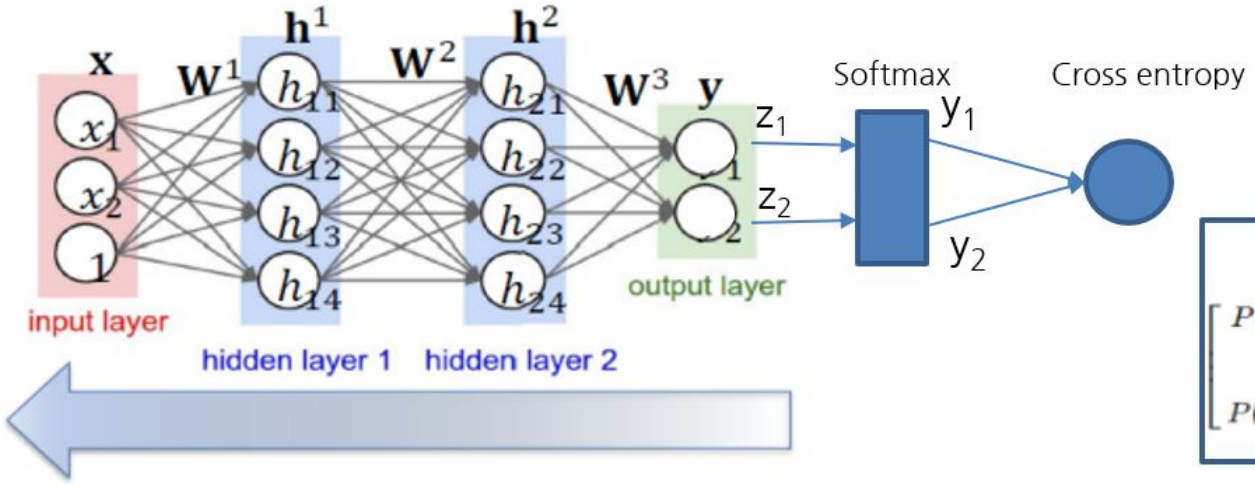
1. Lower data dimensionality
2. Enough information to solve problem.

문제1

Curse of Dimensionality
→ Raw data로 Model 학습 어려움

문제2

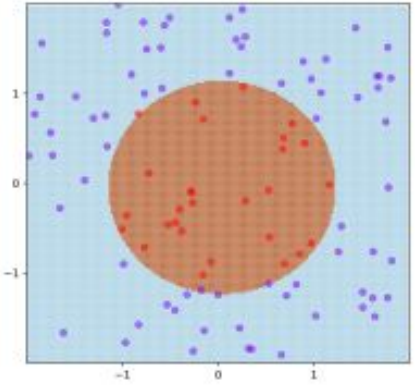
Kernel을 사용한다.
But, Kernel은 한계가 있다!

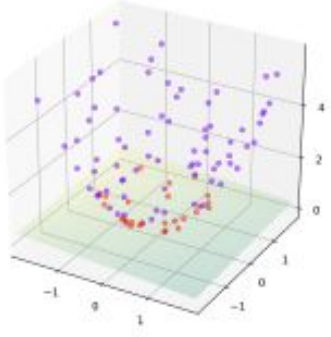


Softmax

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

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





TensorFlow

Why TensorFlow?

Why TensorFlow?

There are a lot of alternatives:

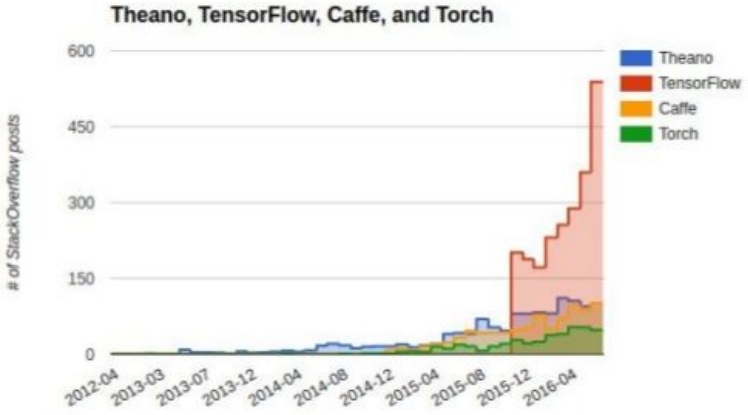
- Torch
- Caffe
- Theano (Keras, Lasagne)
- CuDNN
- Mxnet
- DSSTNE
- DL4J
- DIANNE
- Etc.

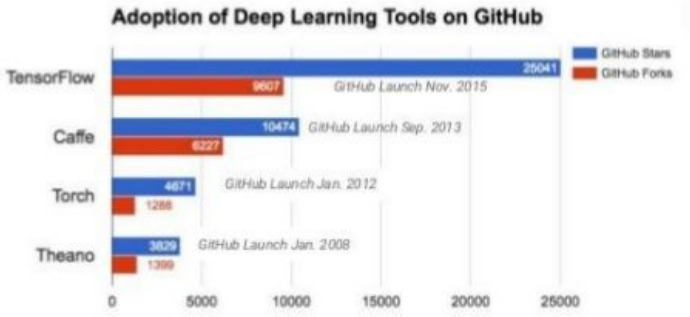
theano

19

Why TensorFlow?

TensorFlow has the largest community





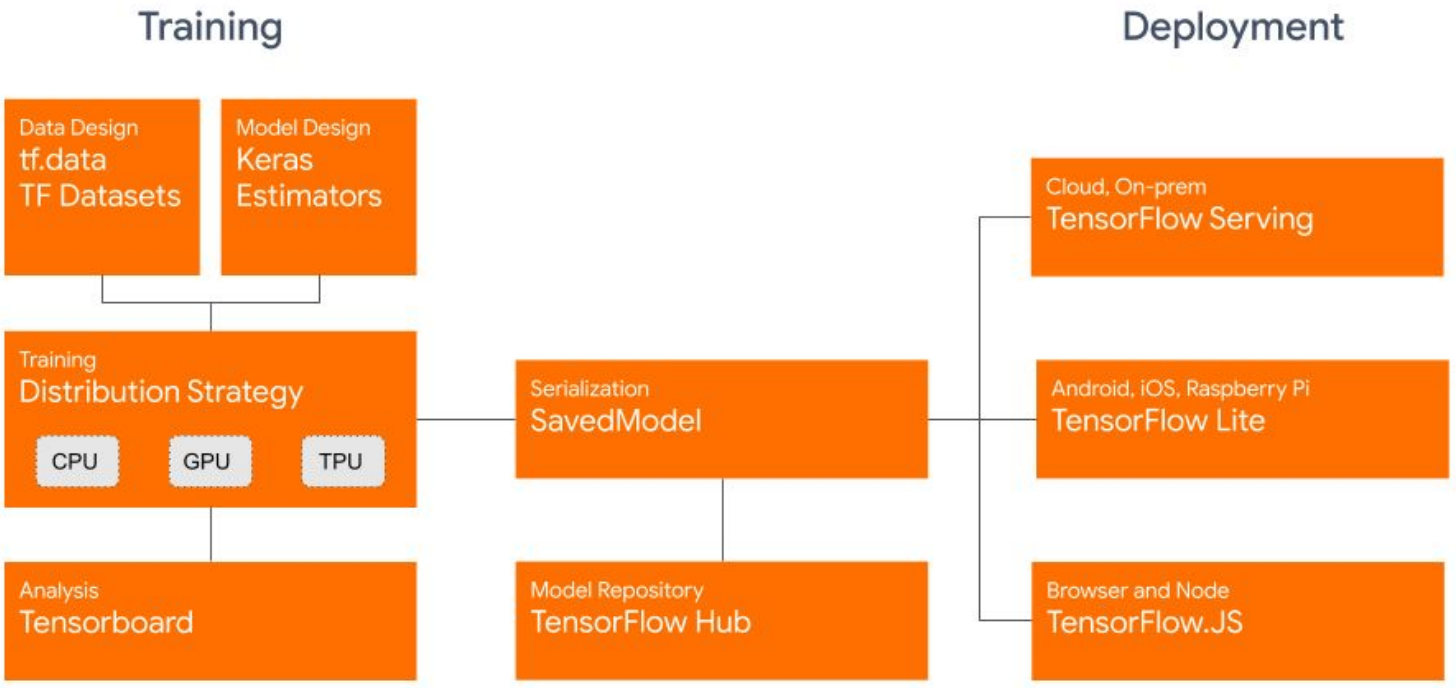
Sources: http://deliprao.com/archives/168
http://www.slideshare.net/JenAman/large-scale-deep-learning-with-tensorflow

Why TensorFlow?

TensorFlow is very portable/scalable

Runs on CPUs, GPUs, TPUs over one or more machines, but also on phones(android+iOS) and raspberry pi's...

Why TensorFlow?

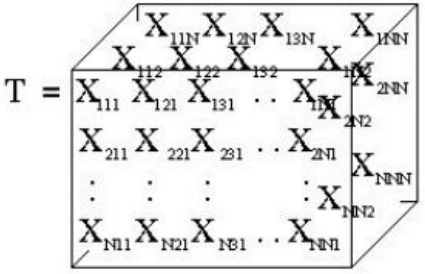


TensorFlow의 특징

Tensor?

Simply put: Tensors can be viewed as a **multidimensional array of numbers**.
This means that:

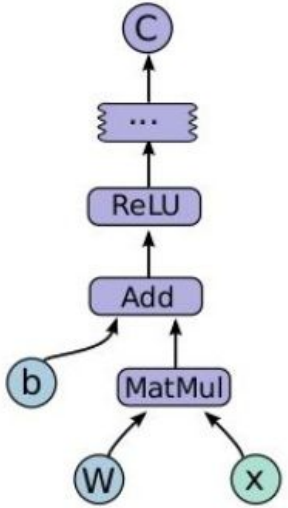
- A scalar is a tensor,
- A vector is a tensor,
- A matrix is a tensor
- ...



TensorFlow의 특징

Data Flow Graph?

- Computations are represented as **graphs**:
  - Nodes are the operations (*ops*)
  - Edges are the *Tensors* (multidimensional arrays)
- Typical program consists of 2 phases:
  - **construction phase**: assembling a graph (model)
  - **execution phase**: pushing data through the graph



Just go TensorFlow.org and learn it!

Thank you!