# [3차시 수업]

※ 리뷰)

실습;

- 구글콜렙과 구글드라이브 연동(구글드라이브와콜렙연동.ipynb)
- 구글콜렙에서 이미지 올리기(upload_file_and_display_image.ipynb)

1)Binary / Multi-Label Classification(다중 레이블 분류)

- 실습;Flux Experiment/Style Transfer Demo
- 체험; 싸이클 GAN(cyclegan.ipynb)
- 문제:Hypothesis using matrix 괄호 맞추기
- 실습; array(배열_연습).ipynb

2)History of DL / MLP Basic(다중퍼셉트론)

- 영상보기; [카오스 술술과학] 다중우주 (9) 시뮬레이션 다중우주 & 궁극적 다중우주
- 실습; [Keras] MNIST 데이터 셋을 이용한 필기 글씨에 대한 CNN Tutorial

3)Overfitting & Underfitting , Regularization(정규화)

## ■ 1교시

참조 사이트; https://wikidocs.net/35838

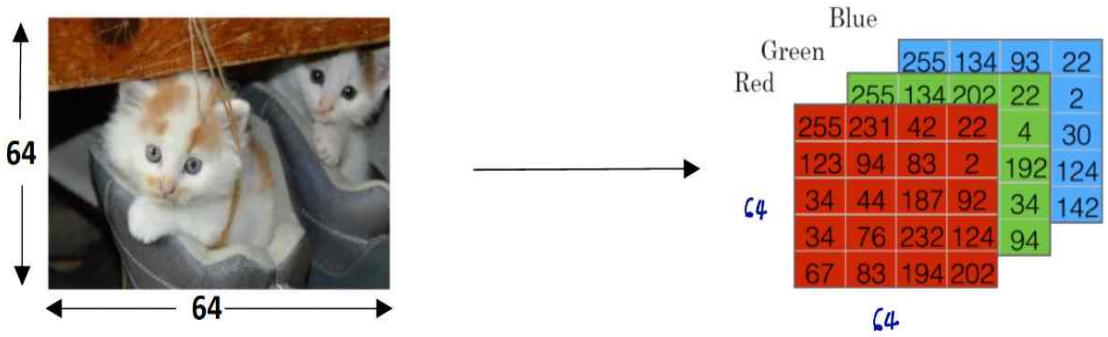
### 1. Binary classification

1) classification은 보통 binary classification을 말함.

2) Binary는 2진법(0과1의 컴퓨터 세계)이다.
, 구분하고자 하는 결과 값이 2가지인 경우 분류하는 방식
예) 고양이와 개의 분류, 불량품과 양품의 분류

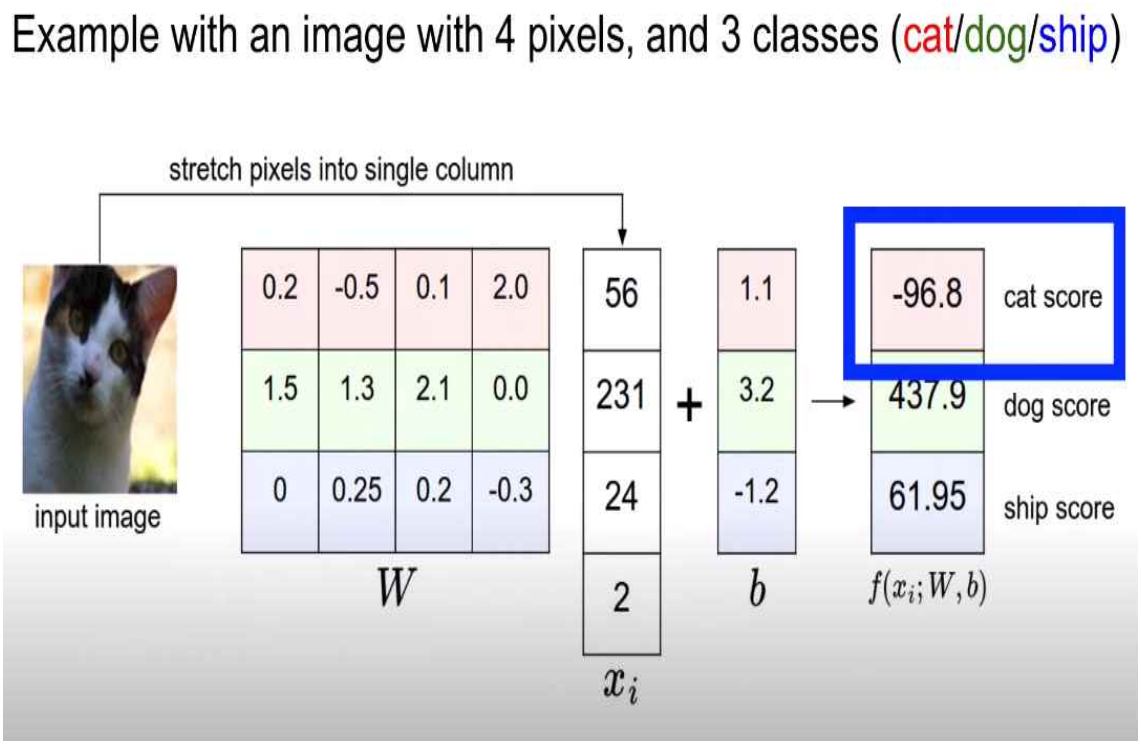
3) 고양이냐 vs 아니냐
- Feature vector x를 입력받아 1 또는 0의 출력

- y를 내는 classifier를 train 한다. 이 경우, 고양이 이미지일 경우 1을, 고양이 이미지가 아닐 경우 0을 출력한다.
- 이미지는 컴퓨터에 각각 red, green, blue 색상 채널에 해당하는 3개 행렬로 저장
- 이 3개 행렬은 이미지와 같은 크기를 갖는다( 고양이 이미지가 64 x 64 pixels 라면 세개의 (RGB) 행렬도 각각 64x64의 크기를 갖는다)




- 각 cell의 값은 pixel의 intensity(의미,정의)를 나타내며, 이는 n 차원 feature vector를 생성하는데 이용된다. 패턴 인식과 기계학습에서 feature vector는 하나의 개체를 대표(고양이인지 아닌지를 나타냄)한다.
- Feature vector x를 생성하려면 각 행렬에 담긴 pixel intensity 값은 "unroll" 또는 "reshape" 된다. 입력 feature vector x의 dimension nx 는 64×64×3=12,288 이다.

※ reshape; 차원을 자동으로 맞춰줌

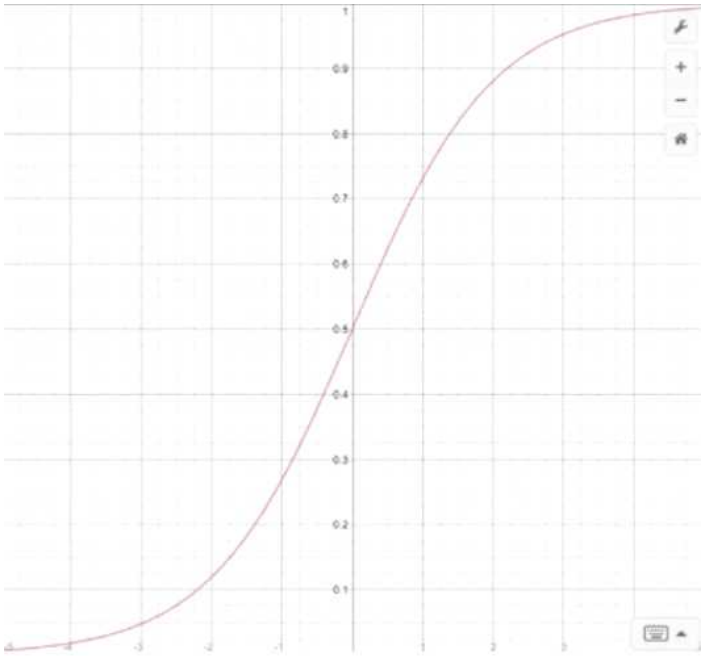



(행렬 계산; 학습이 덜 된 상태, 성능이 안 좋음)

4) Logistic(Binary) Classification?

- 2가지 중 하나를 찾는 모델

예) Logistic(Binary) Classification Hypothesis? -> 0~1사이의 변수를 찾는 것 (sigmoid함수)



(sigmoid 함수)

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

- H(x); Hypothesis(예측)
- e는 자연상수
- W는 weight (상황에 따라 전치를 해야 될 경우와 안해도 될 경우가 있다.)
- X 는 실제 데이터 x_data 값

**※ 전치행렬(transposed matrix)이란?**

- 행과 열을 교환하여 얻는 행렬이다(주대각선을 축으로 하는 반사 대칭을 가하여 얻는 행렬)

$m \times n$ 행렬 $M$의 **전치 행렬** $M^{\mathrm{T}}$은 다음과 같은 $n \times m$ 행렬이다.

$$M_{ij}^{\mathrm{T}} = M_{ji}$$

- 전치행렬의 예:

- $\begin{pmatrix} 1 & 2 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
- $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$
- $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$

# Cost function

$$cost(W) = \frac{1}{m}\sum \quad c\,(H(x), y)$$

$$c\,(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

$$c\,(H(x), y) = -ylog(H(x)) - (1 - y)log(1 - H(x))$$

y데이터가 1이면 뒤에가 사라지게 되며
y데이터가 0이면 앞의 사라지게 된다.

## 2.multi-label classification(다중레이블 분류)

### 1) multi class classification

- True or False가 아니라 여러 output class중에서 어떤 output class에 속하는지, 그리고 반드시 하나의 class에만 속하는 경우를 말한다.
- 과일 사진이 있는 경우 과일은 과일 종에 속한다는 의미(사과이면서 배일 수는 없음)

### 2) multi label classification

- class가 아니라 label임.
- label은 multi-class와 다르게, output class가 배반적이지 않음. 보통 문서를 분류한다고 할 때, 각 문서는 **여러 종류의 특성**을 동시에 가지고 있을 수 있음.

### 3) multi output regression

- y에 속하는 값이 1개가 아닐 때를 의미하는 것
- 예를 들면, "태풍이 어느 정도의 강수량과 풍속으로 온다"를 예측하려고 할 때, 강수량이 y1, 풍속이 y2가 될 수 있다는 것(y 자체가 이처럼 여러 값으로 섞여 있을 때, 이를 multi-output regression이라함)

### 4) multioutput-multiclass classification

- input, ouput이 모두 그 크기의 array로 들어옴으로써 그 값을 처리하고 출력해주는 형식

※ array(배열, 행렬)

(1) 컴퓨터 과학에서 배열(영어: array)은 번호(인덱스)와 번호에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다.

(2) NumPy 배열

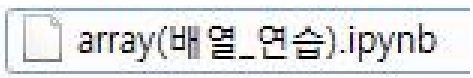
- 많은 숫자 데이터를 하나의 변수에 넣고 관리 할 때 리스트는 속도가 느리고 메모리를 많이 차지하는 단점이 있다. 배열(array)을 사용하면 적은 메모리로 데이터를 빠르게 처리할 수 있다.
- 배열은 리스트와 비슷하지만 다음과 같은 점에서 다르다.

가) 모든 원소가 같은 자료형(정수,문자,숫자등)이어야 한다.

나) 원소의 갯수를 바꿀 수 없다.

**(3) 배열실습**

**array(배열_연습).ipynb**



# [실습 및 체험 ]

1. Style Transfer 체험(Flux Experiment: Style Transfer Demo)

https://fluxml.ai/experiments/styleTransfer/


2. - 체험; 싸이클 GAN(cyclegan.ipynb)

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/cyclegan.ipynb#scrollTo=0KJyB9ENLb2y

3. Hypothesis using matrix

# Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

# Hypothesis using matrix

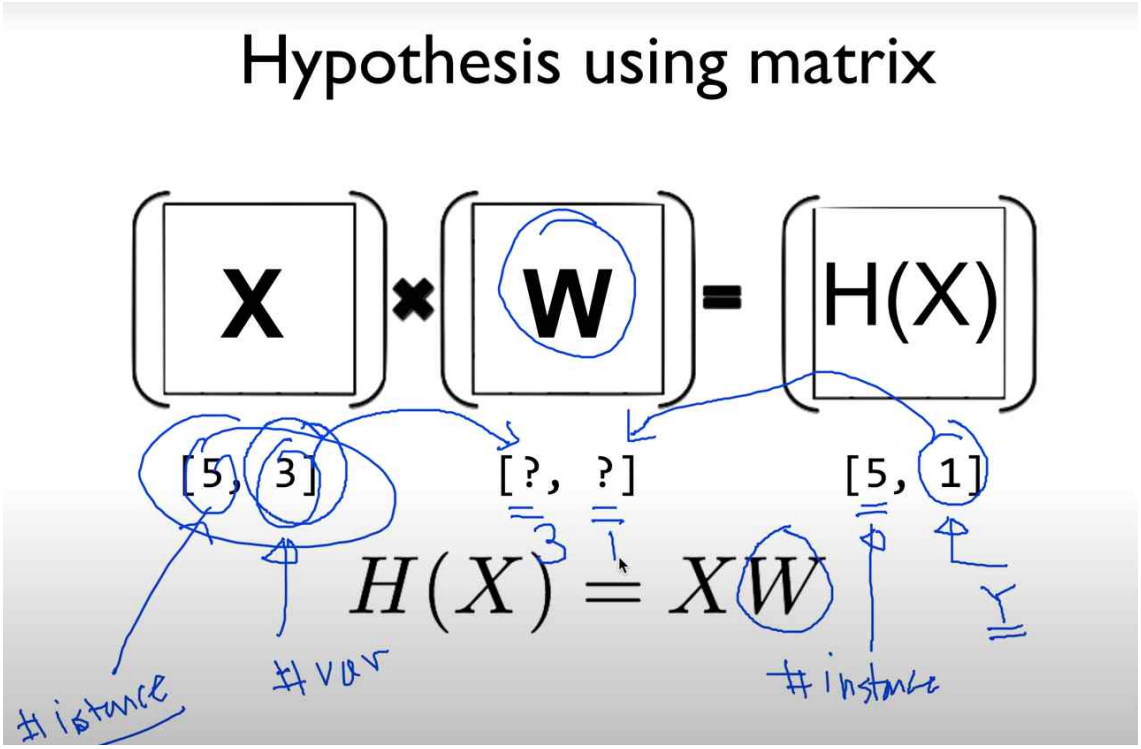
3 — $x_1, x_2, x_3$

5

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[5, 3] [3, 1] = [5, 1]

$$H(X) = XW$$




## ■ 2교시

### 1. 다층 퍼셉트론 (Multilayer Perceptron)

1) 신경망(Neural Network)의 발전

**단층 신경망** : 입력층 + 출력층

**다층 신경망** : 입력층 + 히든층 + 출력층

**심층 신경망** : 입력층 + 2개이상의 히든층 + 출력층

2) 다층 퍼셉트론 (Multi-layer Perceptron)

- 인간이 생각하고 학습하는 방법을 인공지능이 흉내 내기 위해 인공신경망이란 개념을 만들어냈고
- 이를 실현하기 위해 인간의 뉴런을 퍼셉트론으로 흉내를 내서 그 목적을 실현하려 했으나 인간이 생각하기에는 간단한 XOR문제도 해결을 못하는 난관에 봉착해서
- 이러한 XOR문제를 해결하기 위한 시도에서 나온 것이다.

◈ XOR문제란?

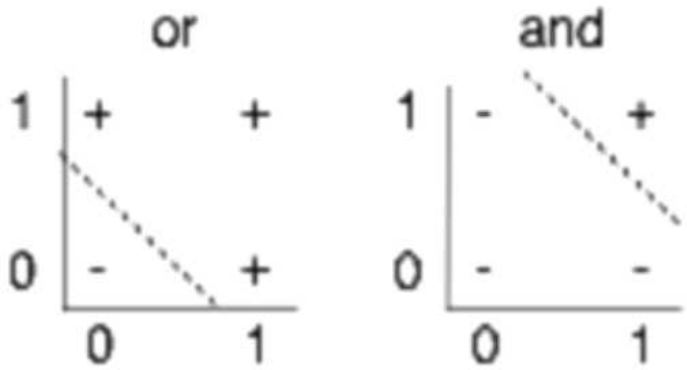
| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

[AND]

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

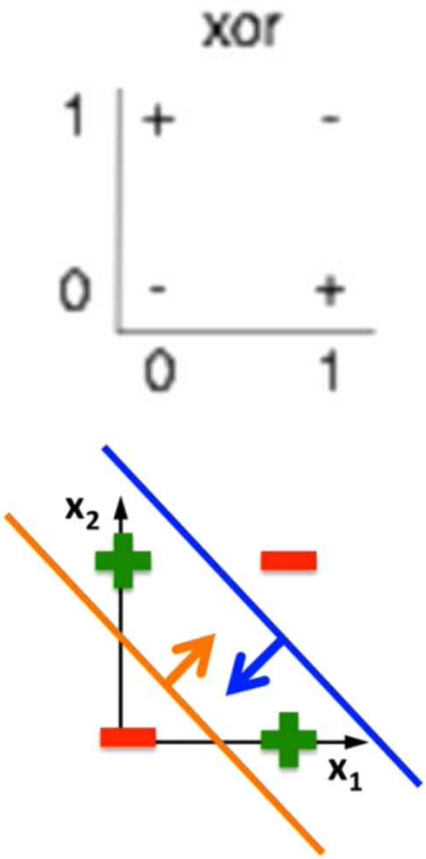
[OR]

AND; 입력 값 중 하나라도 0이 들어가면 결과 값은 0이 나오고
OR; 하나라도 1이 들어가면 결과값이 1이 나온다.




| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

[XOR]




Classification XOR with MLP

XOR문제란 Linear 한 선을 그어서는 XOR을 전부 충족 시킬 수 없고 절반만 충족한다.

- 이러한 문제를 해결하기 위해서는 신경망을 하나가 아닌 여러 개를 가지도록 하면 해결이 되는 문제지만(MLP : MultiLayer Perceptrons) 문제는 당시 1960년대에는 학습시킬 방법이 연구가 되지 않다.

- 이후 딥러닝의 **역전파(Back Propagation) 알고리즘이 고안되면서 학습 문제가 해결**
※ 역전파 알고리즘은  loss 함수를 이용하여 미분을 통해 이전의 Weight 값을 조금씩 수정해 나가는 알고리즘

※ Neural Network는 Layer를 deep하게 쌓은 후 학습한다는 의미에서 Deep Learning이라고 한다.

레이어는 여러 개를 쌓을 수 있으며, 중요한 것은 항상 이전 레이어의 Y 출력값과 다음 W 변수의 X 입력 값이 같아야 한다는 점이고
최종 Y 출력 값에는 제한이 없다.

[모델1] : DEEP

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
```

## << XOR 문제를 해결하기 위해 나온 것 들>>

**(1) 컨볼 루션 신경망 (Convolutional Neural Network)**모델을 통해 제시된 역 전파 (Backpropagation)
하지만 역 전파 알고리즘의 효과가 현실의 문제를 다루기위한 계층의 수를 늘려 갈수록 희미하게 문제(Vanishing gradient problem)가 생겼다.

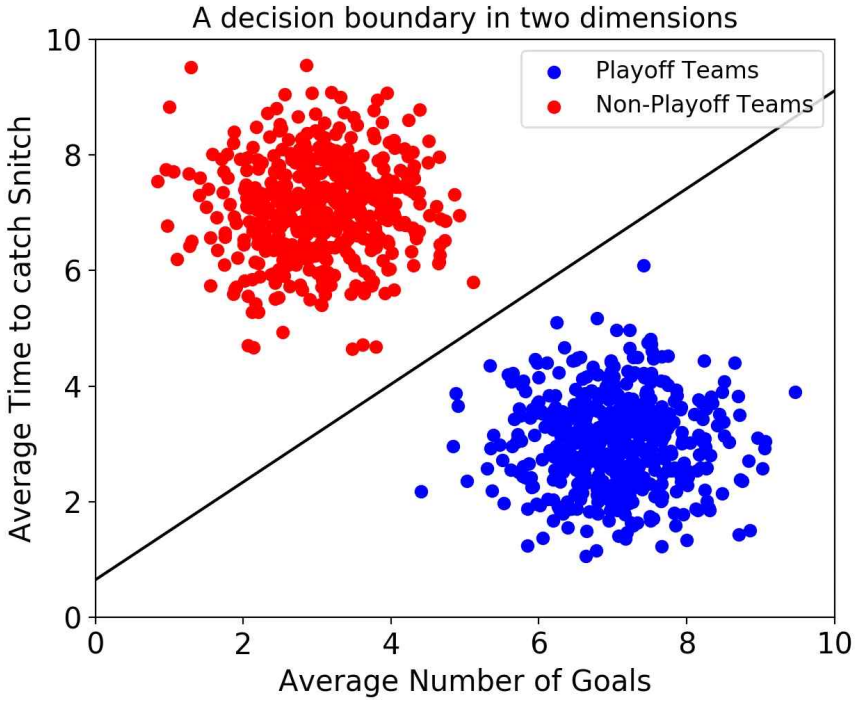
(2) SVM (Support Vector Machine), 의사 결정 나무와 랜덤 포레스트 등의 학습 알고리즘 신경망의 대두

### ※ SVM 소개

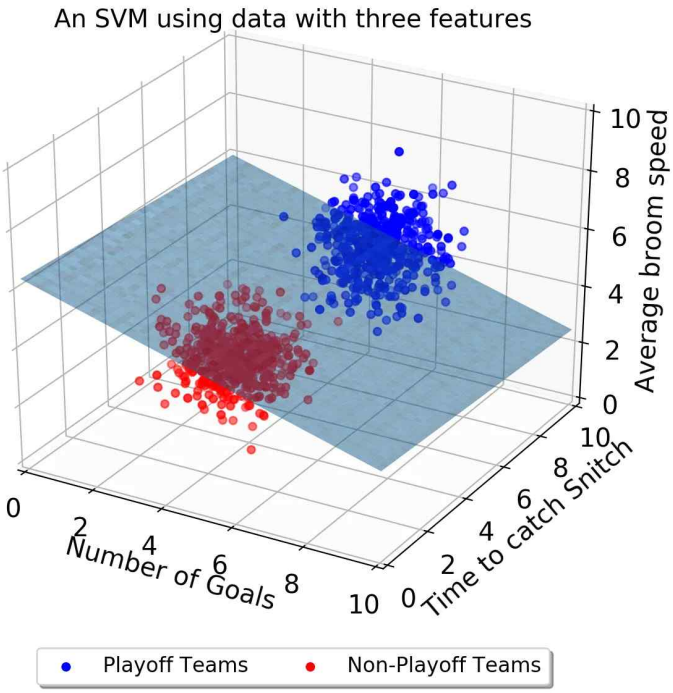
- 결정 경계(Decision Boundary), 즉 분류를 위한 기준 선을 정의하는 모델이다.
- 분류되지 않은 새로운 점이 나타나면 경계의 어느 쪽에 속하는지 확인해서 분류 과제를 수행할 수 있게 된다.
- 결국 이 결정 경계라는 걸 어떻게 정의하고 계산하는지 이해하는 게

중요하다는 뜻이다.

- 만약 데이터에 2개 속성(feature)만 있다면 결정 경계는



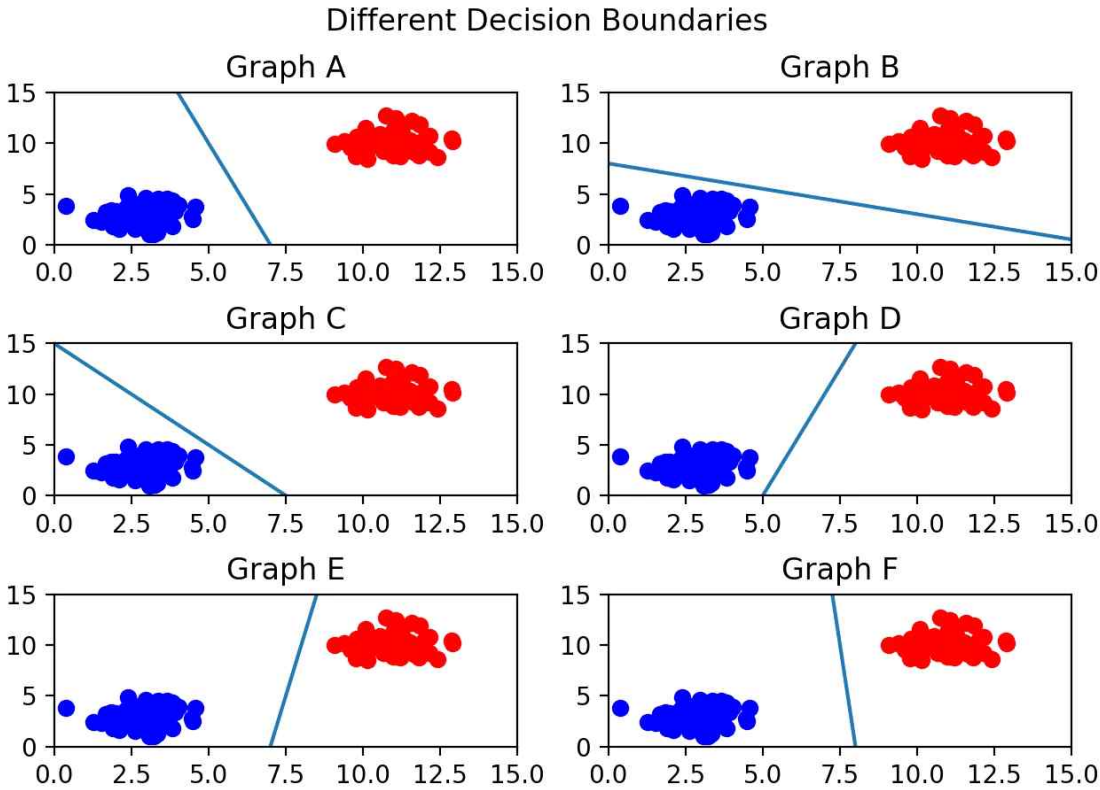

- 속성이 3개로 늘어난다면 3차원으로 그려야 한다.



이 때의 결정 경계는 '선'이 아닌 **'평면'**이 된다.
- 우리가 이렇게 시각적으로 인지할 수 있는 범위는 딱 3차원까지다.
- 차원이 늘어 날수록 결정경계도 복잡해짐. 이러한 고차원은 단순한 평면이 아닌, "초평면(hyperplane)"이라고 부른다.

◈ **최적의 결정 경계(Decision Boundary)**

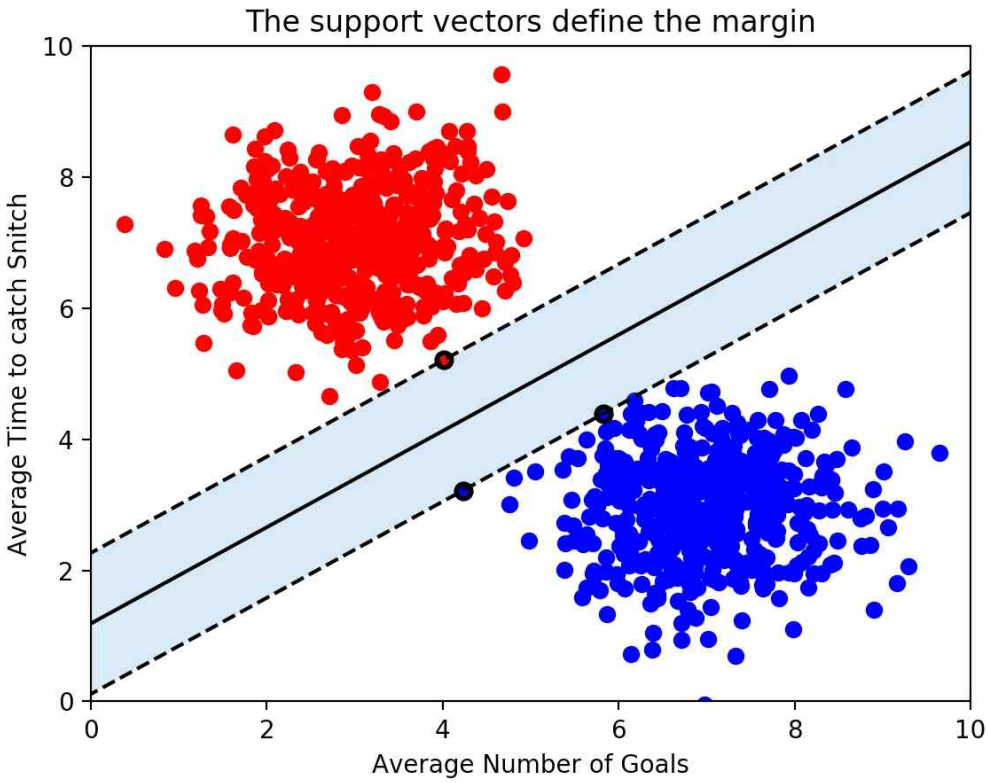
결정 경계는 무수히 많이 그을 수 있는데 어떤 경계가 좋은 경계일까?




- 정답: F(두 클래스(분류) 사이에서 거리가 가장 멀기 때문)
- 결정 경계는 데이터 군으로부터 최대한 멀리 떨어지는 게 좋은 것

◈ **마진(Margin)**

- 마진(Margin)은 결정 경계와 서포트 벡터(Support Vectors)사이의 거리를 의미




- 가운데 실선이 하나 그어져있는데, 이게 바로 '결정 경계'를 뜻함
- 그 실선으로부터 검은 테두리가 있는 빨간점 1개, 파란점 2개까지 영역을 두고 점선을 그어놓았다. 점선으로부터 결정 경계까지의 거리가 바로 '마진(margin)'이다.

**- 최적의 결정 경계는 마진을 최대화한 것**

- x축과 y축 2개의 속성을 가진 데이터로 결정 경계를 그었는데, 총 3개의 데이터 포인트(서포트 벡터)가 필요한 경우 즉, n개의 속성을 가진 데이터에는 최소 n+1개의 서포트 벡터가 존재한다는 걸 알 수 있다.

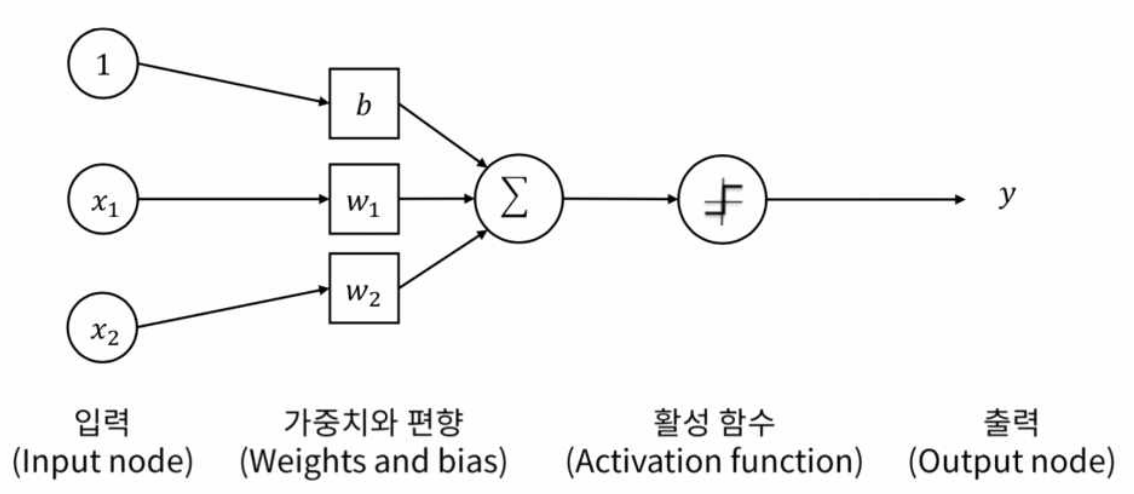
**※ SVM 알고리즘의 장점**

대부분의 머신러닝 지도 학습 알고리즘은 학습 데이터 모두를 사용하여 모델을 학습한다. 그런데 SVM에서는 결정 경계를 정의하는 게 결국 서포트 벡터이기 때문에 데이터 포인트 중에서 **서포트 벡터만 잘 골라내면 나머지 쓸데 없는 수많은 데이터 포인트들을 무시**할 수 있다. 그래서 **매우 빠르다.**

## (3) 심층 신경망 (Deep neural network)

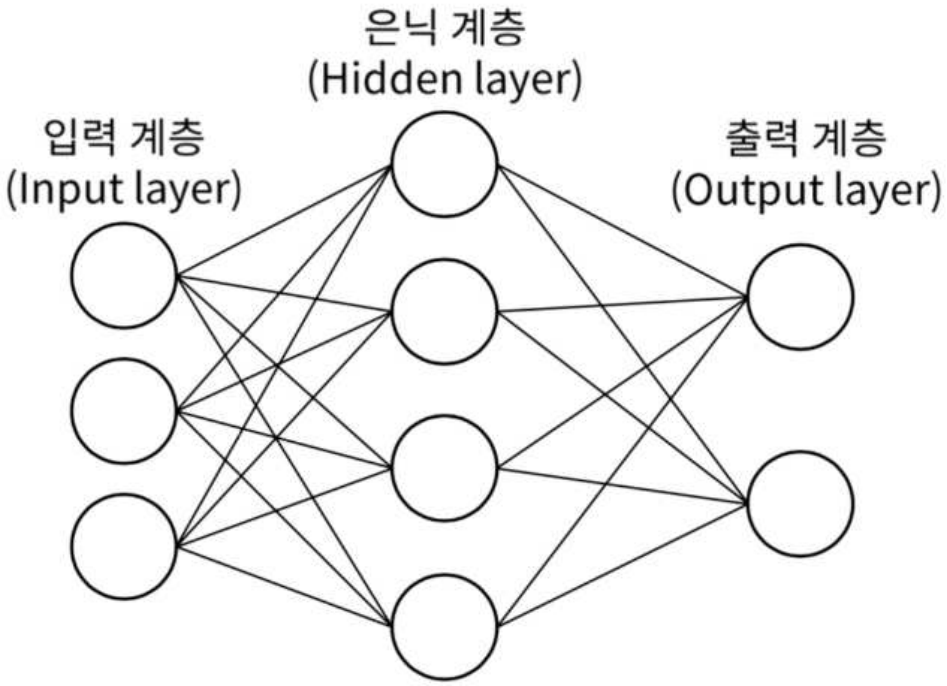
가) 뉴런(Neuron)

신경망은 뉴런을 기본단위로 하며, 이를 조합하여 복잡한 구조를 이룬다.

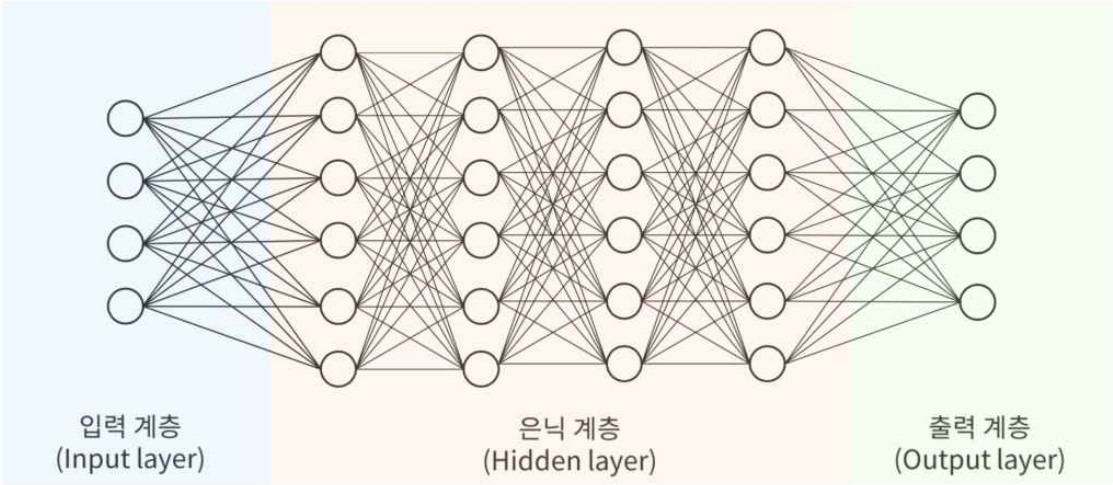



나) 얕은 신경망(Shallow Neural Network)

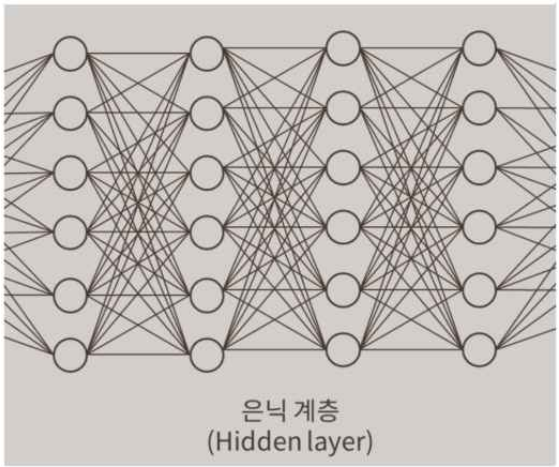
가장 단순한 신경망 구조(은닉계층 1개)



다) 심층 신경망(Deep Neural Network(DNN))
- 얕은 신경망 보다 은닉계층이 많다.



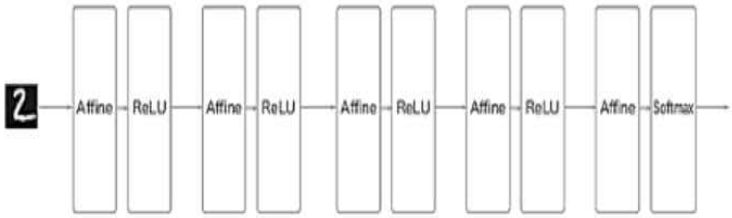

## 심층 신경망은 무엇이 다를까?

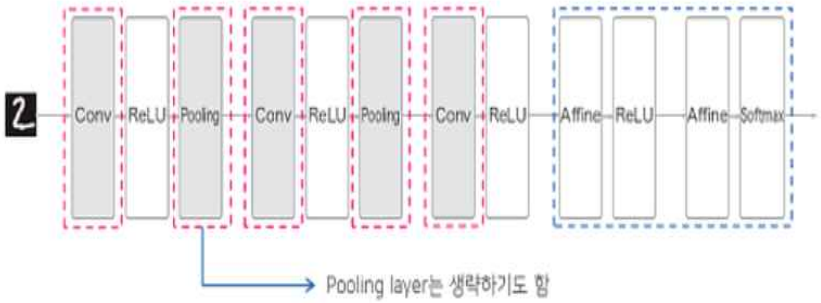



- 은닉 계층 추가 = 특징의 비선형 변환 추가
- 학습 매개변수의 수가 계층 크기의 제곱에 비례
- Sigmoid 활성 함수 동작이 원활하지 않음
  → ReLU (Rectified Linear Unit) 도입 필요
  Gradient Vanishing 문제로 인해

### (4) 합성 곱 신경망 (Convolutional Neural Network)

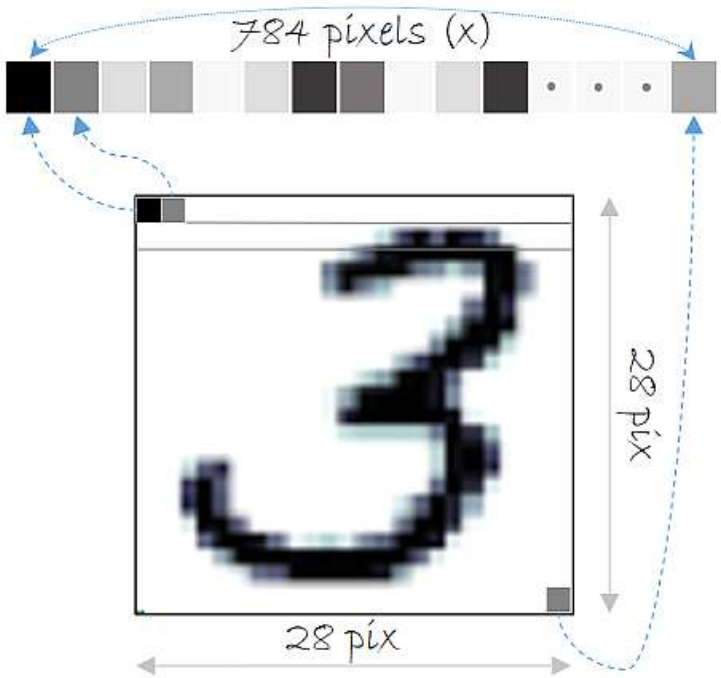
가) CNN의 구조는 완전연결(fully connected)계층과는 달리 **CNN**은 **합성곱층(covolutional layer)**과 **풀링층(pooling layer)**으로 구성되어 있다.



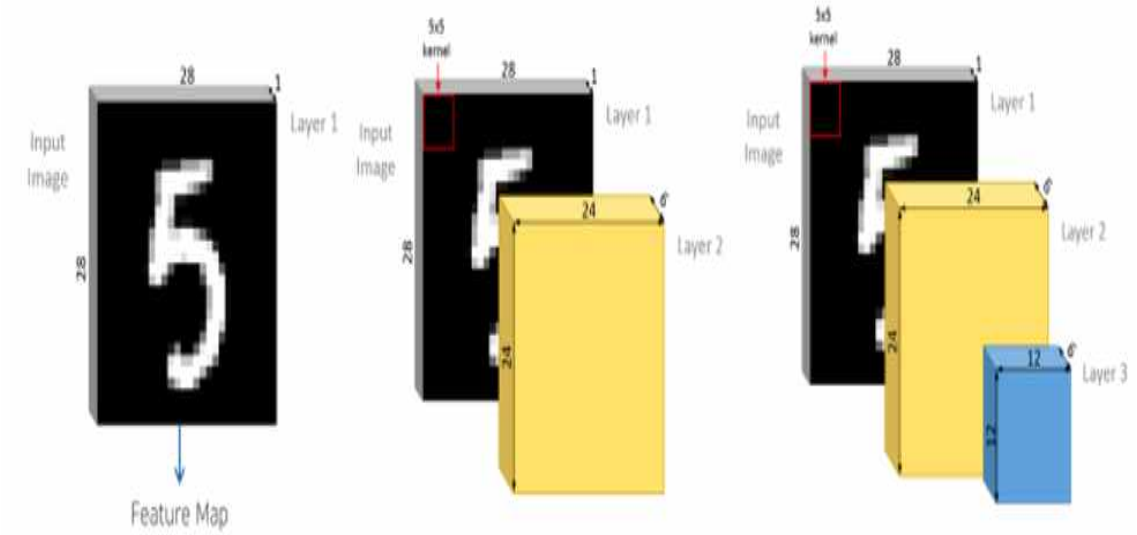

### a) 합성곱층 (Convolutional layer)

- 완전연결 계층(fully connected layer)의 문제점은 3차원데이터를 1차원으로 펼쳤을 때 공간감이 사라진다.
- 완전연결 계층(fully connected layer)을 이용해 MNIST 데이터셋을 분류하는 모델을 만들 때, 3차원(세로, 가로, 채널)인 MNIST 데이터(28, 28, 1)를 입력층(input layer)에 넣어주기 위해서 아래 그림처럼 3차원 → 1차원의 평평한(flat) 데이터로 펼쳐줘야 한다. 즉, (28, 28, 1)의 3차원 데이터를 의 1차원 데이터로 바꾼 다음 입력층에 넣는다.

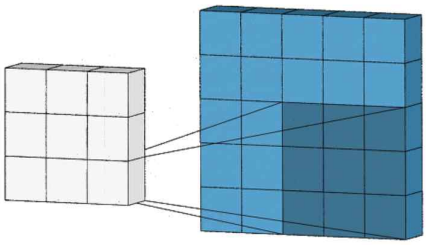


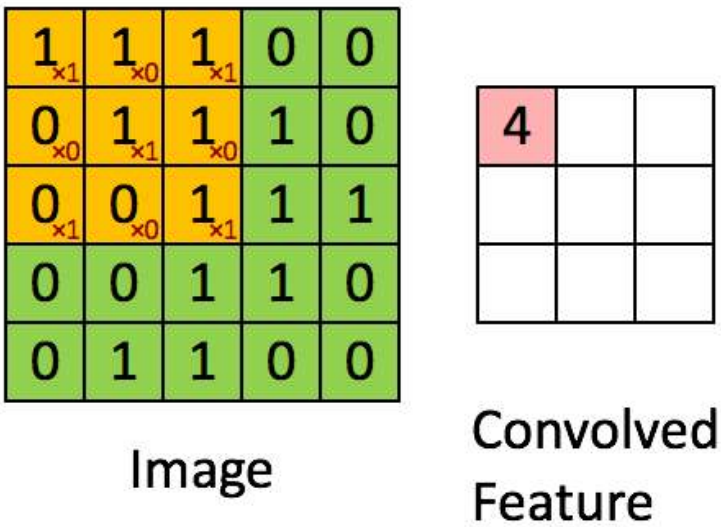
- 이러한 완전연결 계층의 문제점은 바로 '**데이터의 형상이 무시**'된다는 것이다. 이미지 데이터의 경우 3차원(세로, 가로, 채널)의 형상을 가지며, 이 형상에는 공간적 구조(spatial structure)를 가진다. 예를 들어 공간적으로 가까운 픽셀은 값이 비슷하거나, RGB의 각 채널은 서로 밀접하게 관련되어 있거나, 거리가 먼 픽셀끼리는 관련이 없는 등, 이미지 데이터는 3차원 공간에서 이러한 정보들이 내포 되어있다. 하지만, 완전연결 계층(fully connected layer)에서 1차원의 데이터로 펼치게 되면 이러한 정보들이 사라지게 된다.

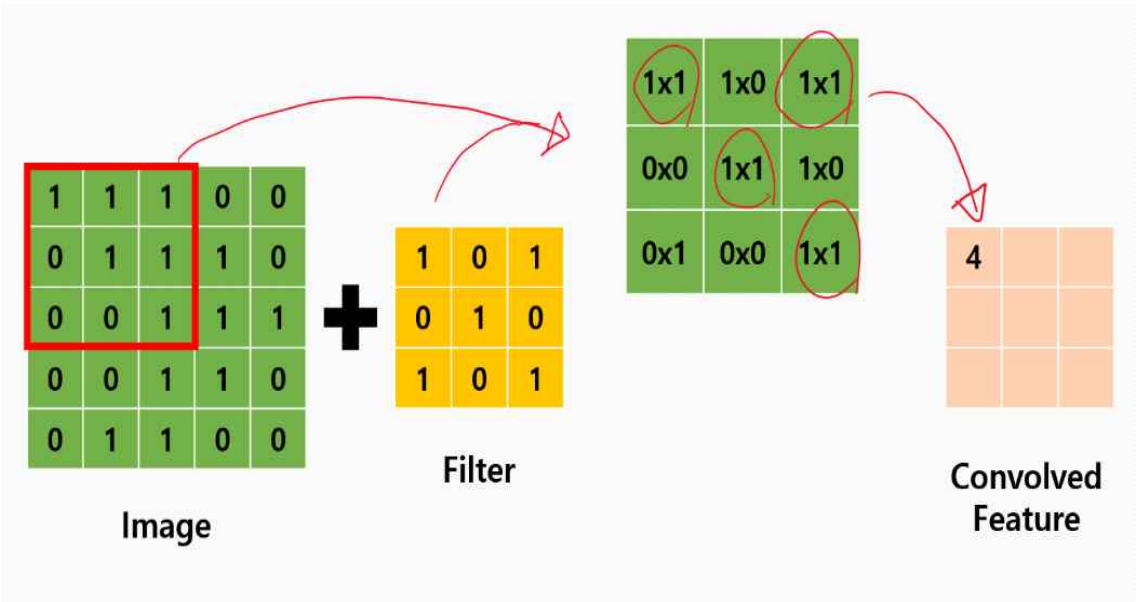
- **합성곱층(Convolutional layer)은 입력 데이터의 형상을 유지한다.** 3차원의 이미지 그대로 입력층에 입력받으며, 출력 또한 3차원 데이터로 출력하여 다음 계층(layer)으로 전달
- 때문에 CNN에서는 이미지 데이터처럼 형상을 가지는 데이터를 제대로 학습할 가능성이 높다.



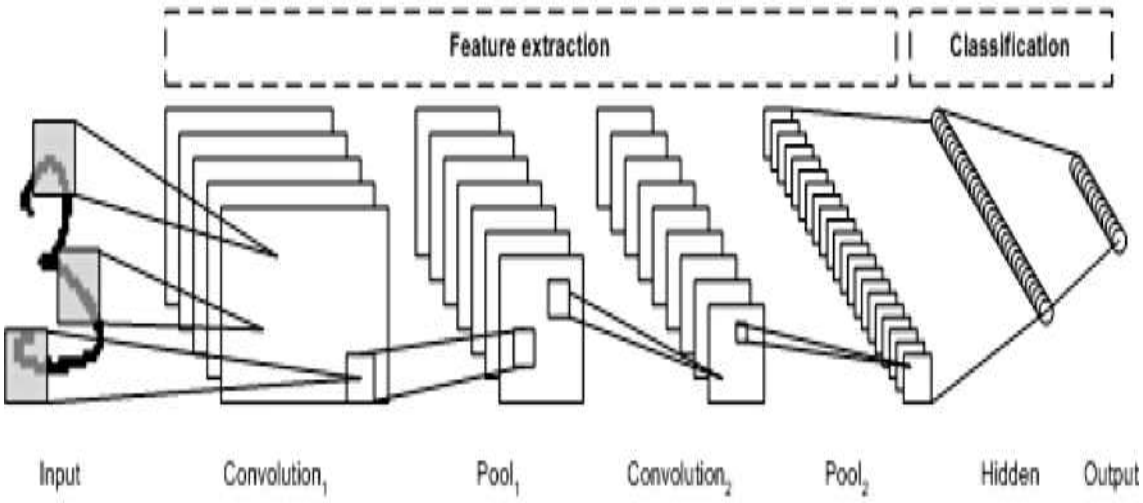

- Convolutional layer의 뉴런은 입력 이미지의 모든 픽셀에 연결되는 것이 아니라 합성곱층 뉴런의 수용영역(receptive field)안에 있는 픽셀에만 연결이 되기 때문에, 앞의 합성곱층에서는 저수준 특성에 집중하고, 그 다음 합성곱층에서는 고수준 특성으로 조합해 나가도록 해준다.






빨간 박스는 필터가 적용될 영역이고 한 칸씩 이동시키면서 적용 시킬건지, 두 칸씩 이동할건지 정한다.(stride 개수 정하기)
이를 통해 이미지의 feature map을 만들고, filter(또는 kernel)의 구성에 따라 이미지의 특징을 뽑을 수 있다.




**• 특징 추출 단계(Feature Extraction)**

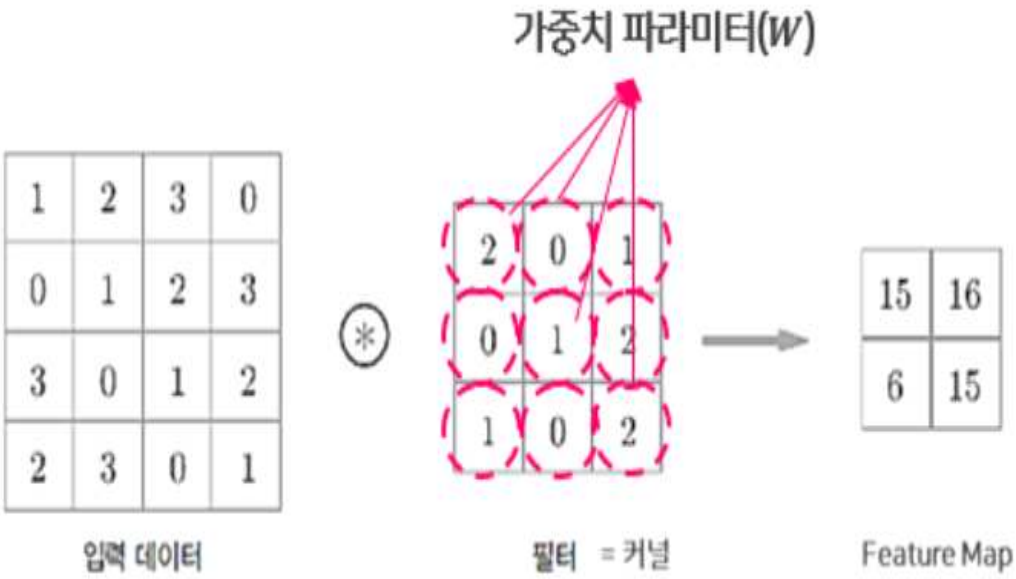
Convolution Layer : 필터를 통해 이미지의 특징을 추출.
Pooling Layer : 특징을 강화시키고 이미지의 크기를 줄임.
(Convolution과 Pooling을 반복하면서 이미지의 feature를 추출)

**• 이미지 분류 단계(Classification)**

Flatten Layer : 데이터 타입을 FC네트워크 형태로 변경. 입력데이터의 shape 변경만 수행.
Softmax Layer : Classification수행.
Output : 인식결과

### ◑ 필터 (Filter)

- 수용영역(receptive field)을 합성곱층(Convolutional layer)에서 필터(filter) 또는 커널(kernel)이라고 한다. 아래의 그림처럼, 이 필터가 바로 합성곱층에서의 가중치 파라미터(w)에 해당하며, 학습단계에서 적절한 필터를 찾도록 학습되며, 합성곱 층에서 입력데이터에 필터를 적용하여 필터와 유사한 이미지의 영역을 강조하는 특성맵(feature map)을 출력하여 다음 층(layer)으로 전달한다.




## <실습>

**- [Keras] MNIST 데이터 셋을 이용한 필기 글씨에 대한 CNN Tutorial**

참조사이트; https://pinkwink.kr/1121

# ■ 3교시

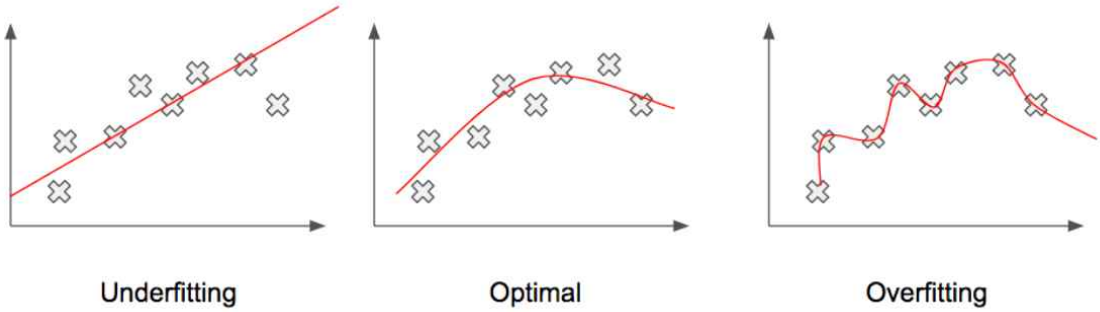
## 1. Overfitting & Underfitting

- 모델이 실제 분포보다 학습 샘플들 분포에 더 근접하게 학습되는 현상을 오버피팅(overfitting) 이라고 하며, 오버피팅(overfitting)을 피하는 방법을 정규화(regularization)라고 한다.

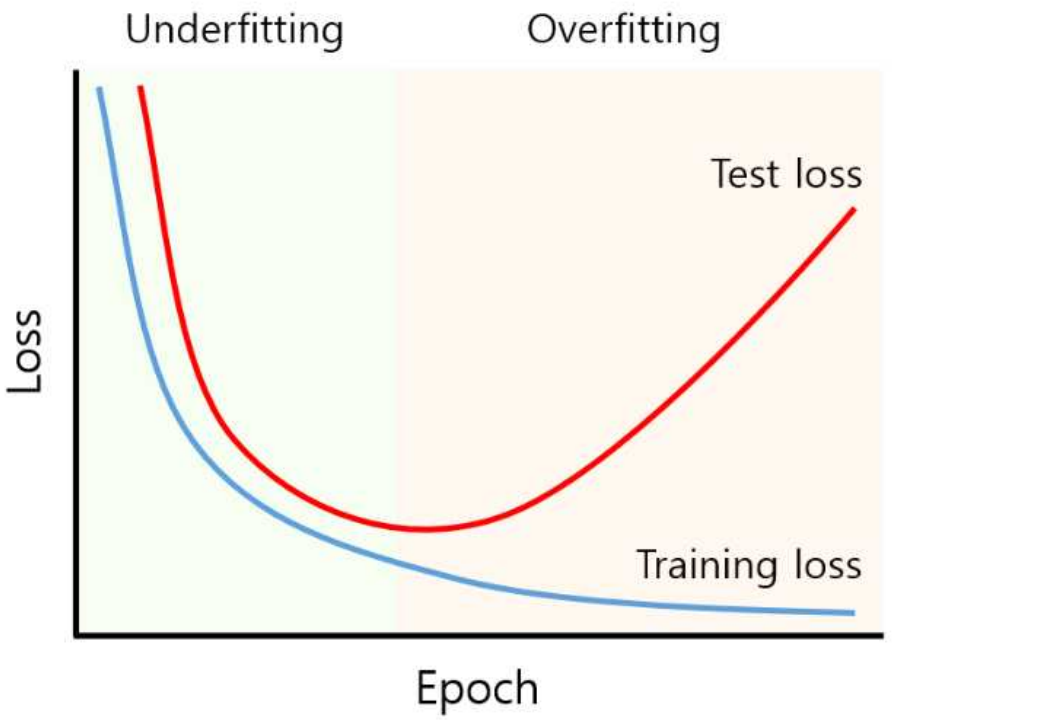
### 1) Overfitting(과대적합)

- 오버피팅(overfitting)은 학습 데이터를 지나치게 학습하여, 학습 데이터는 잘 맞추지만 실제로 모델이 맞추어야 할 검증 데이터는 에러(error)가 많은 경우를 의미한다.
- 모델에 에러가 많을 때(=데이터에 노이즈가 있을 때) 일반적으로 맞추고 싶었던 목적 함수(object function)가 복잡할수록 overfitting의 가능성이 있다.

- 모델을 학습시킬 때 오차함수 E값을 최소화하다보면 오버피팅될 수 있으므로, 학습셋 결과에 비해 검증셋 결과가 좋지 않다면 오버피팅이 발생하여 일반화가 성능이 떨어진 것일 수 있다.




※ Stochastic Noise (비결정적 노이즈)
오버피팅이 일어나는 대표적인 이유는 노이즈 때문이며, 노이즈에는 관측 에러가 있다.
※ Deterministic Noise (결정적 노이즈)
노이즈가 모델(objective function)의 복잡도(차원)가 커질수록 증가하며, 이에 오버피팅이 발생할 확률이 높아진다.
이러한 노이즈는 모델링을 할 수 없다.



**가) Overfitting 방지책; 오버피팅은 항상 발생한다! 어떤 문제를 풀던 발생하기 때문에 정규화를 항상 고려해야 한다.**

(1) Early Stopping (조기 종료)
Epoch 수(학습 횟수)가 클수록 학습 셋에 대한 오차는 줄어들지만 에폭 수가 늘어날수록 오차가 증가하며 오버피팅이 발생할 수 있다. 따라서, 이전 에폭과 비교해서 오차가 증가하면 오버피팅이 발생하기 전에 학습을 멈추는 것을 'early stopping'이라고 한다.

(2) 데이터를 조금씩 늘린다.
- 샘플을 조금씩 늘려가면서 레이어를 늘린다 (예: 64 -> 128.. 512-> 1024...)

### 3) underfitting(과소적합)

가) 학습셋이 적거나 학습이 제대로 되지 않아 n=1처럼 목적 함수와 학습 데이터 간의 오류가 많은 경우를 의미한다.
나) underfitting이 방지책:
- 데이터가 부족할 경우, data agumentation(증가)을 이용해서라도 늘린다(예: 좌우 반전, 이미지 자르기, 밝기 조절)
- Data agumentation: 데이터를 늘려 모델의 성능을 좋게 하는 과정 (CNN의 대표 모델 중 하나인 VGG에서 많이 사용함)
- 적절한 능력(capacity)을 갖는 모델을 활용한다.
- 서로 다른 앙상블(ensemble)을 활용한다.

※ 앙상블(ensemble)
데이터를 나누어 여러 모델을 만들어서 나온 결과 중 가장 많이 나온 결과를 선택함 (=bagging)
예를 들어 10,000개 데이터 중 8,000개씩 다섯 개의 모델을 만들고 가장 많이 나온 결과를 선택하는 것을 뜻한다.

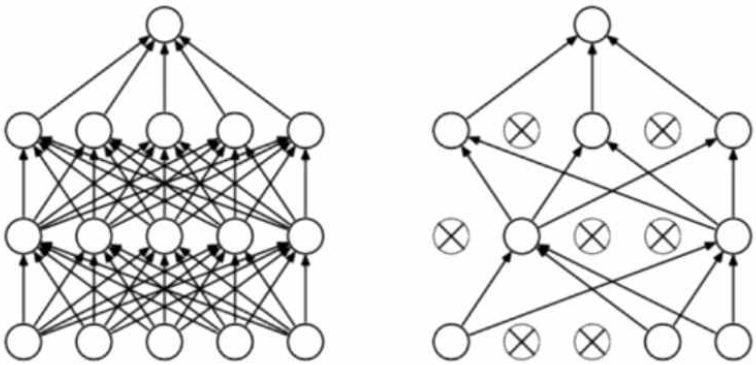
### 4) Batch Normalization(배치 정규화)

(1) 다양한 정규화 방법 중 하나는 모델의 복잡도가 높아질수록 불이익(패널티)을 주는 것이다.
(2) Batch Normalization은 각 미니배치별로 사전 정규화 처리를 수행하는 기법이다.

(3) 데이터 셋 전체를 정규화하는 것보다 학습에 사용하는 미니배치별로 정규화하기 때문에 학습하는 과정 전체에서 효과가 나타난다.
(4) 미니배치에 있는 값에서 평균을 빼고 variance로 나눠준다.
(5) 데이터 전처리보다 배치 정규화가 학습 과정 전체에 효과적인 이유는 데이터의 전처리로 데이터 셋을 정규화(백색화)하고 웨이트의 초깃값을 정리함으로써 학습이 잘 진행되기는 하지만, 학습시킬 때는 네트워크 내부에서 분산이 편중돼버리므로 그 정리한 효과가 한정적이다.
(6) 배치 정규화는 학습에 사용하는 각 미니배치별로 정규화하므로 학습 과정 전체에서 효과가 발휘된다.
(7) Parameter Norm Penalties
- 비용함수에 제곱을 더하거나(L2) 절댓값을 더해서(L1) 웨이트의 크기에 제한을 준다.
- L2 weight decay (제곱값)
  L2 파라미터는 ridge 회귀분석 또는 tikhonov 정규화로도 알려져 있음
- L1 weight decay (절댓값)
  LASSO는 선형모델의 L1 패널티와 최소제곱법(LSM)을 합친 모델
(8) Dataset Augmentation
- 정규화 방법 중 학습셋의 크기를 늘리는 것
- 가지고 있는 데이터의 수는 제한적이기 때문에, 학습셋에 가짜 데이터(fake data)를 포함할 수 있다.
(9) Dropout
- Dropout은 오버피팅 문제를 해결하고자, 일반화(generalization) 성능을 향상시키기 위한 방법이다.
학습할 때 뉴런의 일부를 네트워크 상에 존재하지 않는 것처럼 랜덤으로 '0'으로 만든다(= 비활성화). 드롭아웃의 확률은 일반적으로 p=0.5를 사용하며, 학습할 때만 드롭아웃을 하며 테스트할 때는 모든 뉴런을 사용한다.
- 하나의 모델로 학습하면 오버피팅이 발생할 수 있지만, 드롭아웃을 통해 앙상블 학습(ensemble learning)처럼 마치 여러 모델을 학습시킨 것과 같은 효과를 주어 오버피팅 문제를 해결할 수 있다.

## Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net (b) After applying dropout. *[Srivastava et al., 2014]*

전체 weight을 계산에 참여시키는 것이 아니라 layer에 포함된 weight 중에서 일부만 참여시키는 것이다.