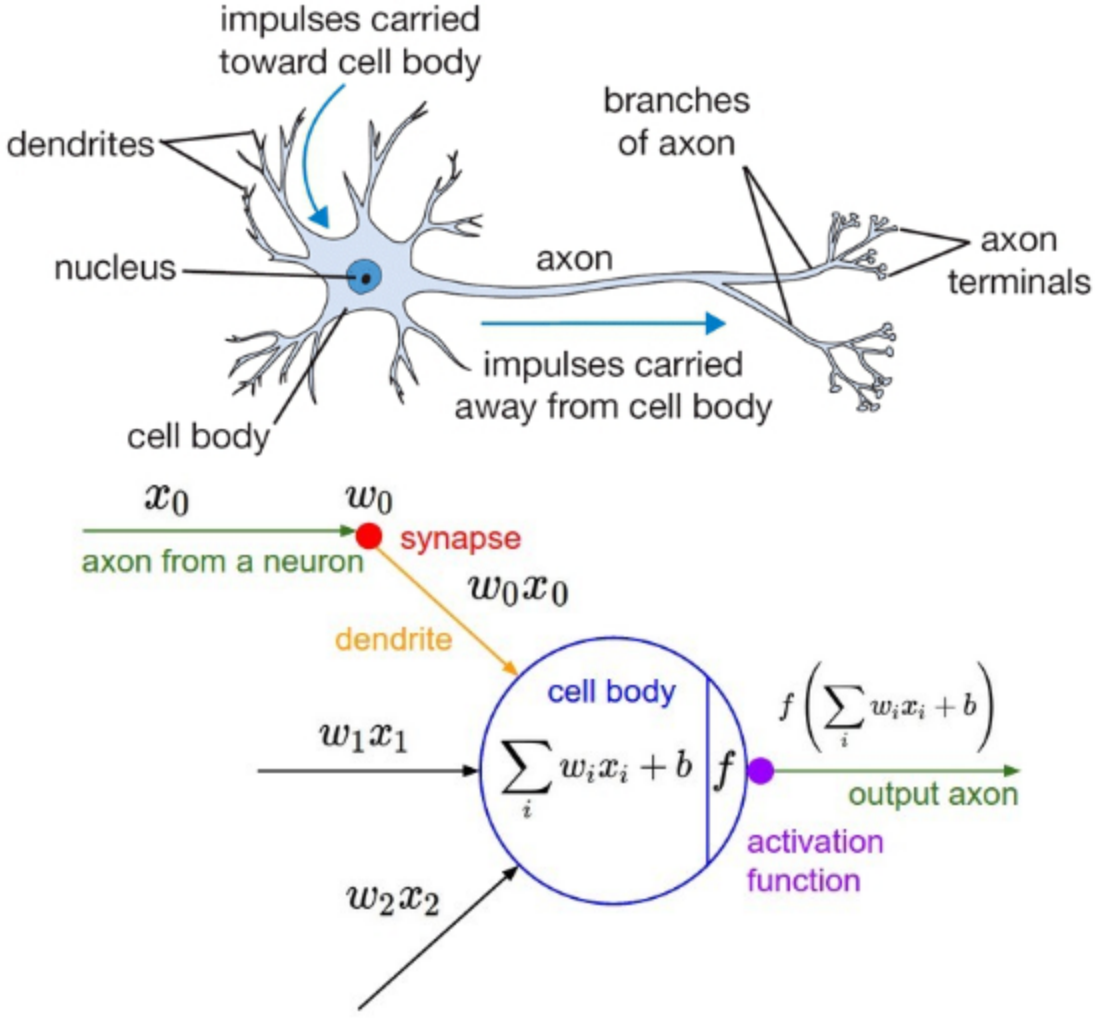
Deep Learning Basic

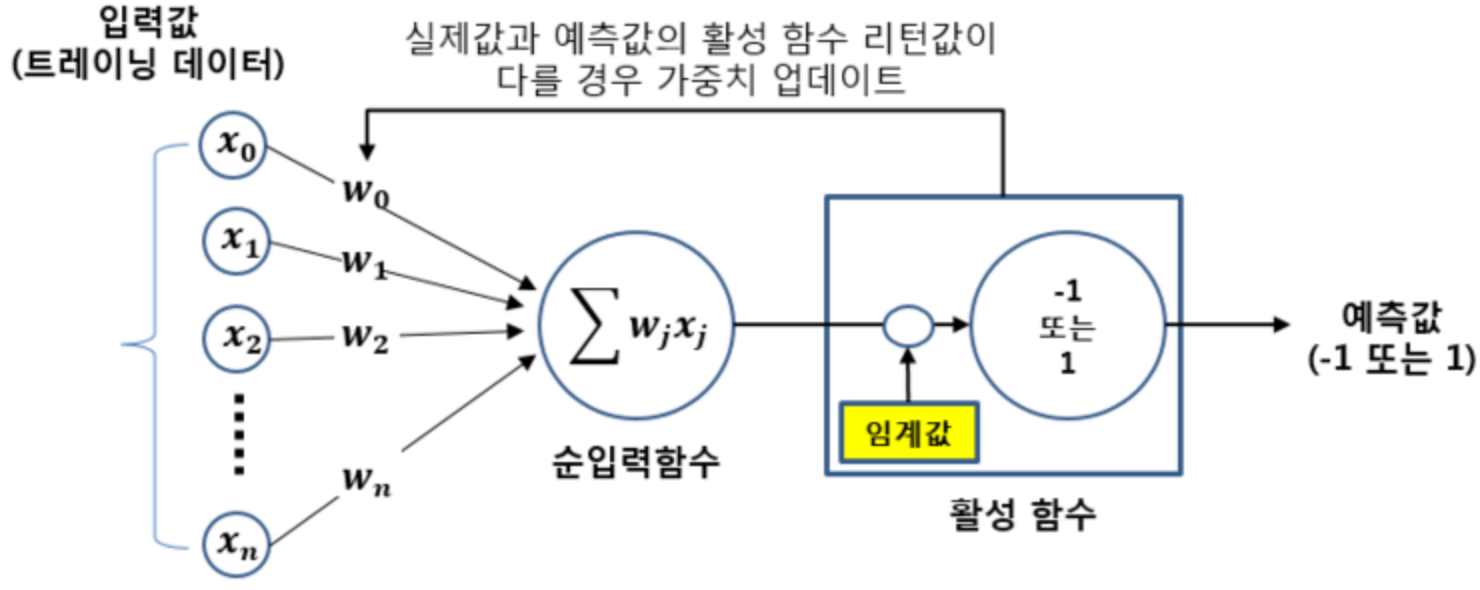
https://github.com/dlcjfgmlnasa/Tongmyong_College_Tensorflow_Tutorial

Created by **Choelhui lee**

Perceptron 이란?



- **입력값을 two class로 구분하기 위한 알고리즘**
- 인간의 신경(뉴런)을 모방 (Neural Network 의 기초 개념)
- activation function은 step function



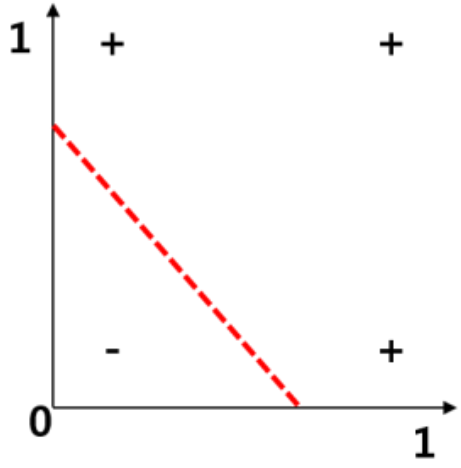
- weight와 bias을 구함으로서 입력 벡터를 분류할 수 있는 직선방정식이 생성

Perceptron convergence theorem

- `퍼셉트론`은 직선으로 분리할 수 있는 문제라면 **데이터로부터 자동으로 학습할 가능**
- `비선형`으로 분리되어야 되는 문제의 경우 **학습 불가!!**
- 세상에는 선형적인 문제 보다는 **비선형적인 문제가 훨씬 많음**
    - ex) XOR문제, 자연어 처리, 기타 등등

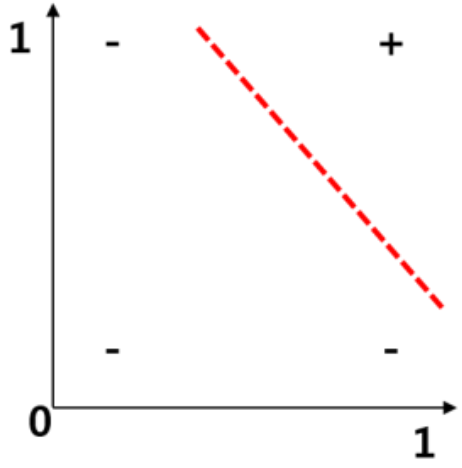
XOR 문제

OR



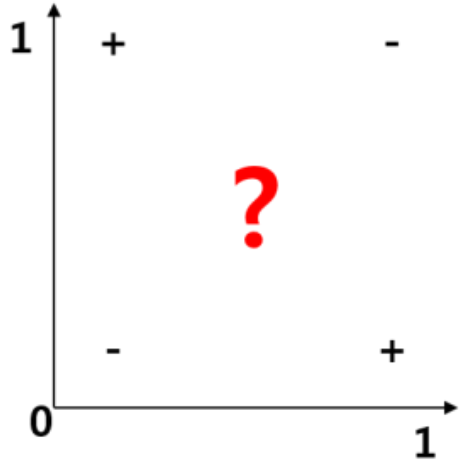
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

XOR



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

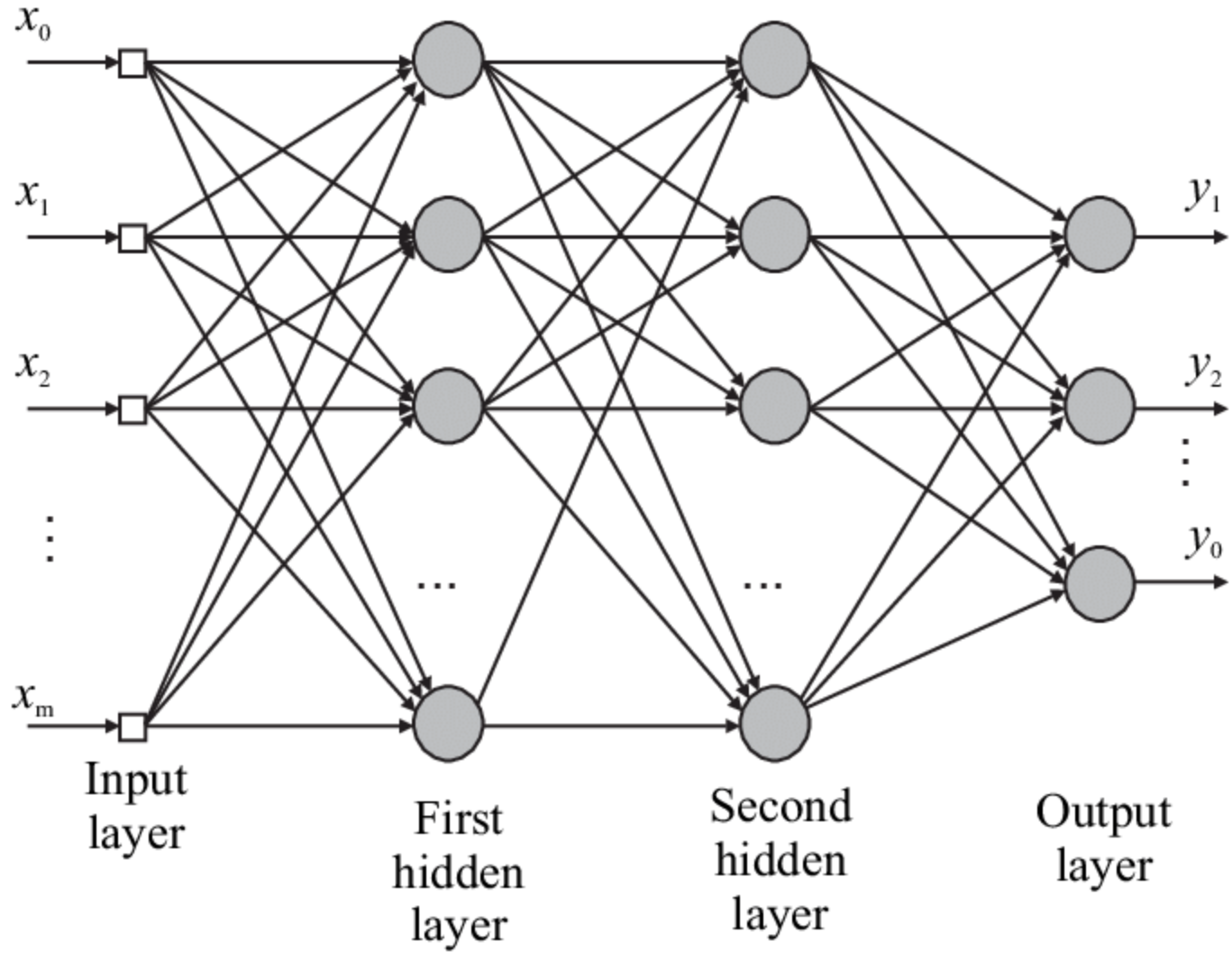
- `직선방정식`으로는 절대 `XOR 문제`를 해결할 수 없다!!

Multi-Layer Perceptron (MLP)

`Nerual Network == Multi-Layer Perceptron`

- `퍼셉트론` 으로는 `XOR문제` 해결 불가!!
- 하지만 **`퍼셉트론`** **이 여러층으로 쌓이게 된다면??**
- **해결 가능!!**

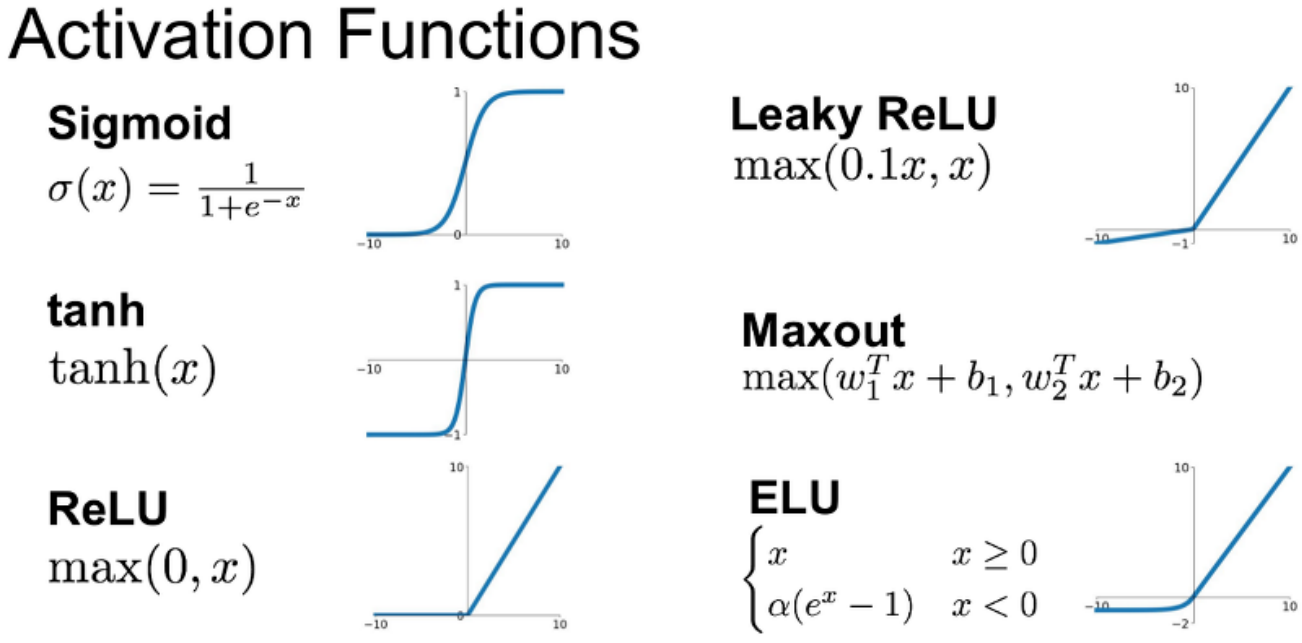
Multi-Layer Perceptron 구조



- `input layer` + `hidden layer` + `output layer` 로 구성

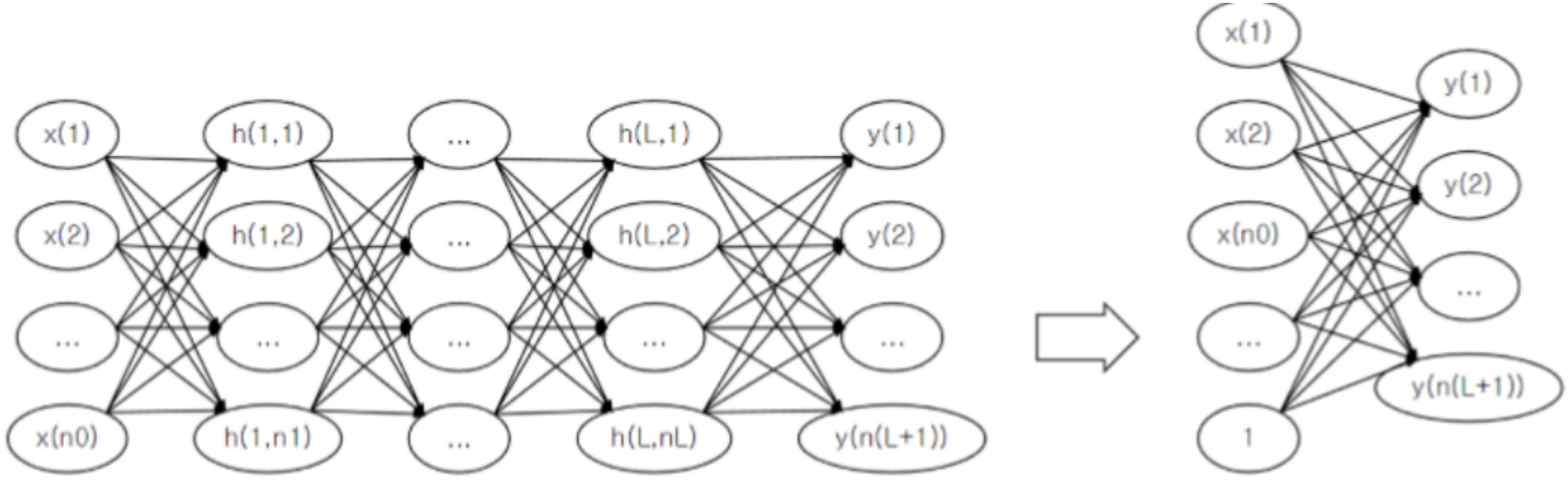
- `hidden layer`가 3개인 **Neural Network**
- `Layer` 마다 **`activation function(활성화 함수)`** 존재
- `activation function`은 **`비선형 함수`** 를 사용

비선형 함수란?



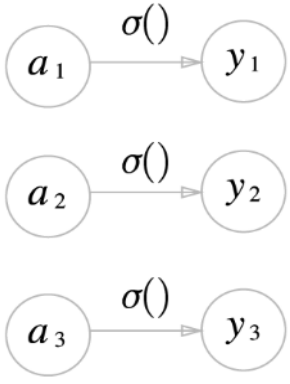
왜 비선형 함수를 사용할까?

- **if** `hidden layer` 의 `activation function` 이 `Linear` 인 경우
- **`hidden layer` 를 얼마를 추가해도, 단층 퍼셉트론과 동등한 효과를 가진다.**
- 가만히 생각해보면 당연한 이야기...

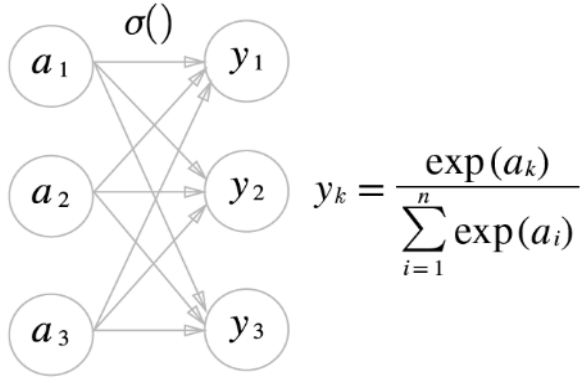


출력층 설계

- 문제냐에 따라 `output layer`에서 추가되는 함수가 달라짐
- **회귀 (Regression)**
  - 주로 `항등함수`를 사용
  - *ex) 사진 속 인물의 몸무게는 얼마일까요??*
- **분류 (Classification)**
  - 주로 `소프트맥스 함수`를 사용
  - *ex) 이 그림의 숫자는 무엇일까요??*



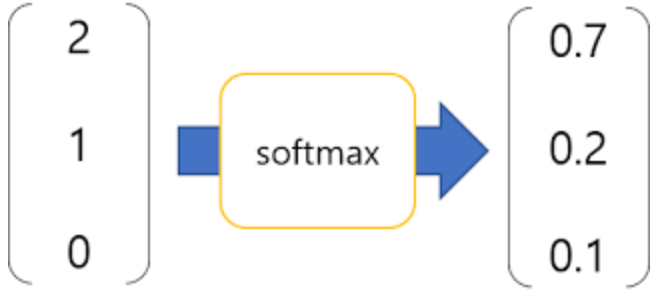
항등 함수



소프트 맥스 함수

Softmax 함수

- 출력 값을 0~1사이의 실수로 강제하며 **출력값의 총합이** 1이 됨
- 즉 출력을 확률로 해석할 수 있어 문제를 `확률적으로 대응할 수 있음`
- softmax 함수를 적용해도 각 원소의 `대소 관계는 변하지 않음` (지수함수처럼)



Loss function(손실 함수)

`손실 함수(loss function) == 비용 함수(cost function) == 목적 함수(objective function) == 오차 함수(error function)`

- **`Neural Network 모델`이 얼마나 정확한지를 수치적으로 이야기 해주는 값**
- **`Neural Network`은 `loss function`을 통해 가장 최적화된 `가중치 매개변수(weight,bias)의 값`을 탐색**
- 임으로 개발자가 만들어 사용하는 경우도 있지만
- `Mean squared error(평균 제곱 오차)`, `cross entropy(교차 엔트로피)`를 가장 많이 사용

- **Mean squared error(평균 제곱 오차)**
  - `Regression(회귀)` 문제를 해결하기 위해 주로 사용

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

- **cross entropy(교차 엔트로피)**
  - `Classification(분류)` 문제를 해결하기 위해 주로 사용

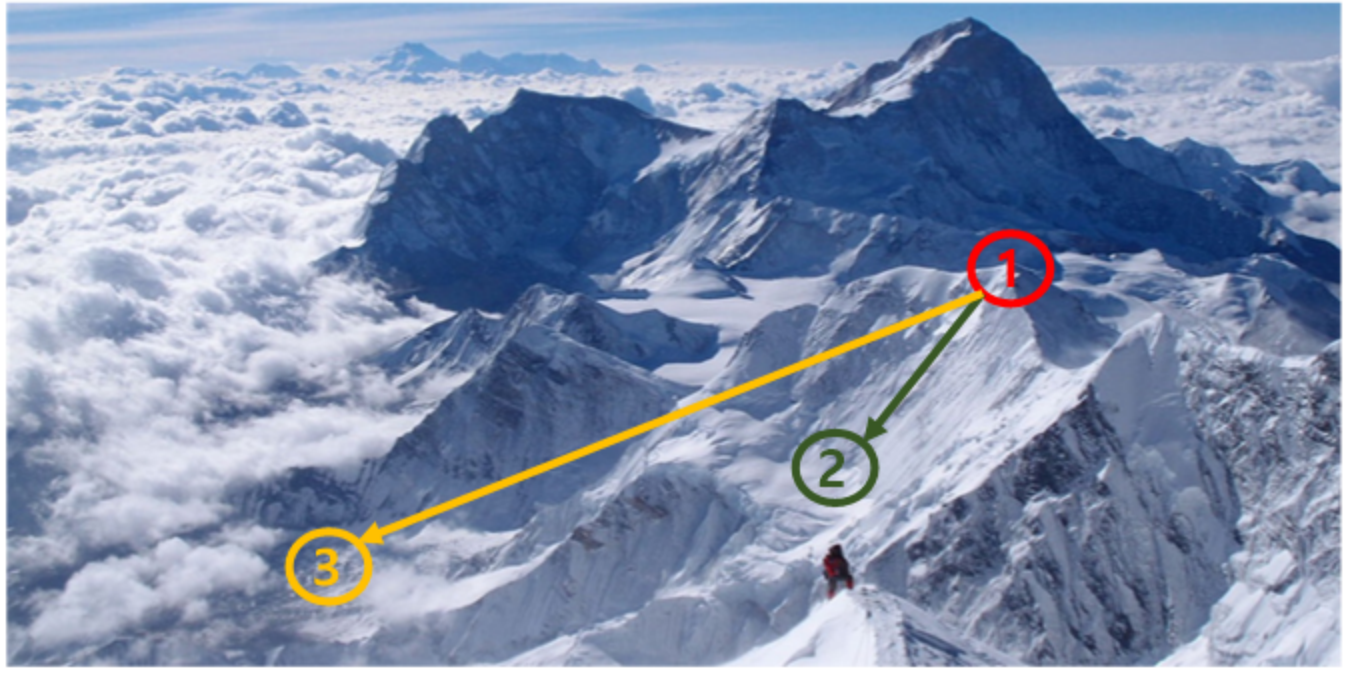
$$E = -\sum_{k} t_k \log y_k$$

Gradient Descent

- `ML` 은 결국 문제를 해결하기 위한 `최적의 매개변수` 를 찾는 방법을 말함
- `loss function` 의 값이 가장 낮을 때 최적이라고 할 수 있음
- `loss function` 을 가장 작게 만들어주는 방법 중 하나가 **경사 하강법(Gradient Descent)**
- *직관적으로 산에서 가장 낮은 곳을 찾아 내려가는 문제로 생각하면 됨*

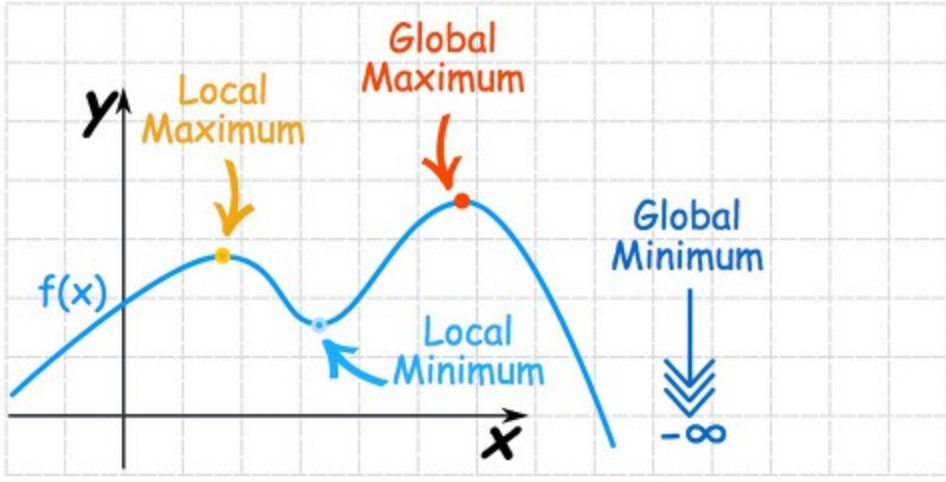
- 낮은 쪽의 방향을 찾기 위해 `loss function` **을 현재 위치에서 미분**
- 최소점을 찾을 때까지 계속 미분

> 장님이 산을 내려가고 있다... == **Gradient Descent**



> `1번` 에서 출발한 `눈먼 장님` 은 산의 가장 아래가 어디 있는지 모르기 때문에 `3번` 이 아니라 `2번` 이 목적지로 착각할 수 있음

Local Minimum & Global Minimun



- `Gobal Minimum` 을 찾아야되지만 `어디서 출발하느냐` 혹은 `어느 방향으로` 출발하는가에 따라서 결과가 `local minimum` 으로 빠질수도 있음
- 16장 그림을 기준으로 보면
  - `1 : starting point` `2 : local Minima` `3 : Global Minima` 으로 볼 수 있음

Gradient Descent Problem

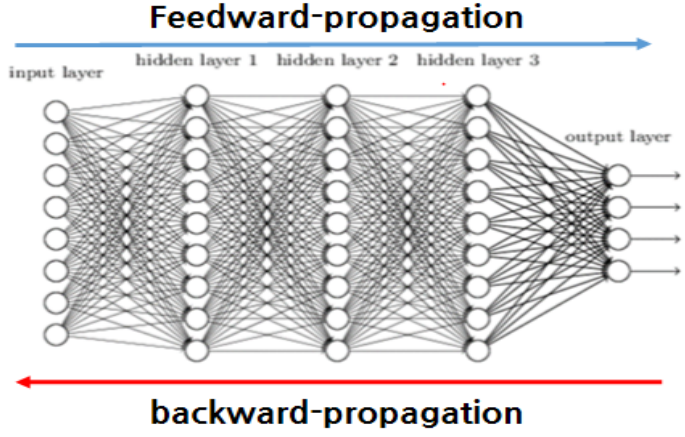
- `Dataset` 전체를 **`배치(batch)`** 라고 부름
- 모든 Data 대해 `Gradient` 를 계산 평균 낸 뒤 `파라미터 업데이트`
- 이러한 특징 때문에 **`Full-batch GD`** 라고도 불림
- **`학습 데이터가 매우 크기 때문에 한 번에 모든 데이터를 보고 계산하게되면 문제가 발생`**
    - Memory 부족
    - 오래걸림
    - **해결법 : Stochastic gradient descent**

Stochastic Gradient Descent

- **학습 데이터가 매우 크기 때문에 한 번에 모든 데이터를 보고 계산하는 과정은 매우 비효율적**
  - **데이터 몇 개를 random하게 뽑아서 그라디언트를 계산하자!**
  - **데이터 몇 개의 Gradient의 평균을 통해 근사된 값을 도출**
- **Stochastic GD** VS **mini-batch GD**
  - Stochastic GD는 배치 크기가 1인 경우
  - 배치 크기가 1보다 크다면 mini-batch GD
  - 일반적으로 두 경우 모두 SGD 라고 부름

Neural Network 학습방법

1. `Feedward-propagation`
   - 입력값이 주어지면 NN을 통과시켜 생성되는 최종 결과를 출력
2. `Backward-propagation`
   - 도출된 결과를 이용해 레이어를 거꾸로 순회하면서 NN을 학습
3. **네트워크 학습은 함수 최적화 방법에 따라 진행**
   - `backward-propagation` 시 `gradient descent` 알고리즘을 사용하여 학습



수치 미분