DSAC Module4

Deep Learning (2)

2023년 11월 18일~12월 16일

권오준

(ojkwon@deu.ac.kr)

머신러닝
(Machine Learning)

인공지능과 머신러닝

- 인공지능을 구현하는 방법은 다양
- 머신 러닝 기반의 AI가 2000년대 이후 급속히 발전
- 딥러닝: 인공 신경망을 기반으로 하는 머신 러닝 기술
  - 마치 사람이 많은 정보에 접하면서 학습하듯이 컴퓨터도 데이터를 보고 학습하는 방법
  - 음성인식, 자동차 번호판 인식, 언어 번역, 채팅 대화, 글쓰기, 작곡 등 여러 분야에서 좋은 성과를 낸다

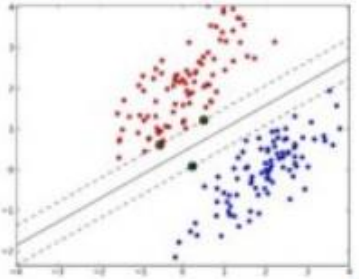
머신 러닝

인공지능(Artificial Intelligence)

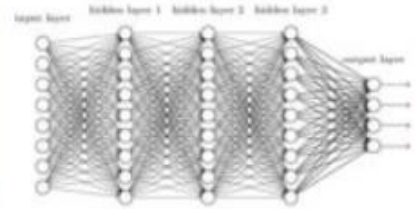
- Deep Blue

머신러닝(Machine Learning)

- Linear Regression
- Logistic Regression
- SVM



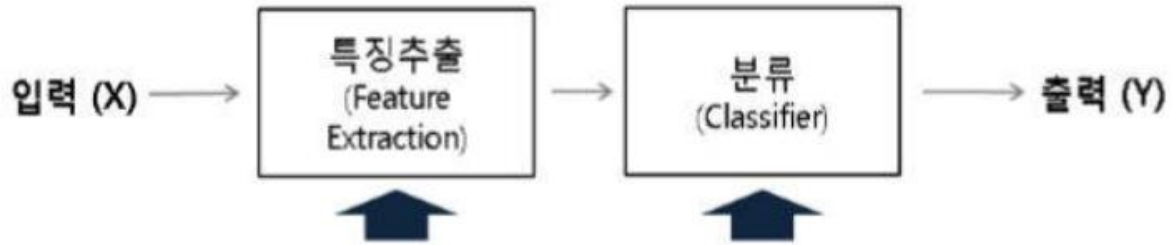
딥러닝(Deep Learning)



- CNN
- RNN
- LSTM

머신 러닝 vs 딥 러닝

< 머신러닝 (Machine Learning >

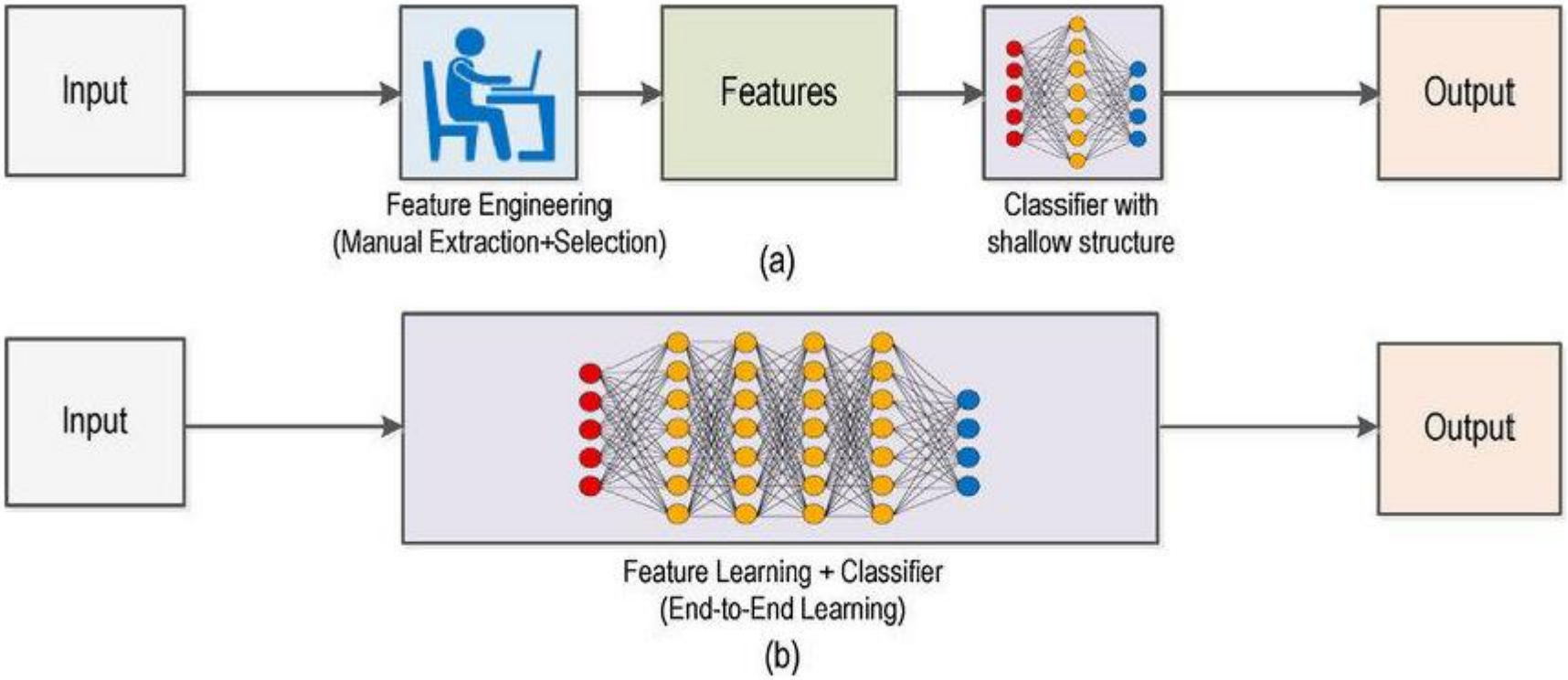


- 특징추출을 위한 알고리즘을 인간이 직접 제공
- 해당분야에 대한 지식 및 직관, 알고리즘 구축을 위한 상당한 노력 필요
- 컴퓨터 '학습'의 영역

< 딥러닝 (Deep Learning >

- 컴퓨터 '학습'의 영역

머신 러닝 vs 딥 러닝


Input
Feature Engineering
(Manual Extraction+Selection)
Features
Classifier with
shallow structure
Output
(a)
Input
Feature Learning + Classifier
(End-to-End Learning)
Output
(b)

머신러닝 특징

- 예전에는 컴퓨터는 프로그래머가 코딩한 대로만 동작
  - 계산을 빨리 하든지,
  - 이미지를 처리하든지,
  - 정해진 알고리즘대로 빠르고 정확하게 동작하는 일
- 머신러닝
  - 컴퓨터가 데이터를 보고, 스스로 기능을 향상시키는 방법을 찾아내어서 점차 성능을 향상시킨다.

머신 러닝?

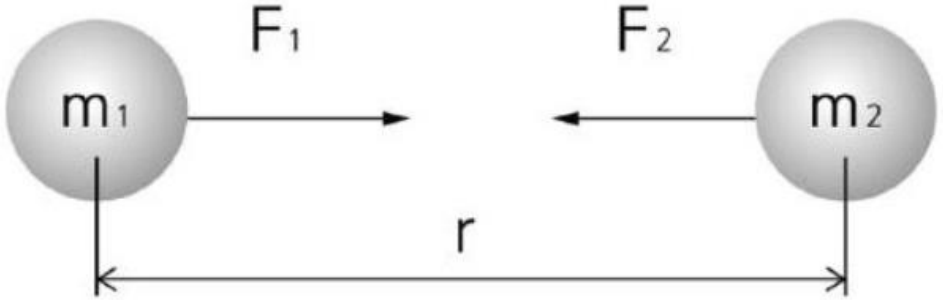
- 현실 세계의 많은 현상
  - 수식으로 간단히 모델링하기 어렵고, 과학적으로 증명할 수도 없다.
  - 하지만 거의 정확히 예측할 수 있는 모델은 만들 수 있다.
    (단, 충분한 데이터 필요)

- 머신 러닝
  - 머신(컴퓨터)이 데이터를 보고 학습을 하여 점차 지능적인 동작 수행
  - 목적 : 학습을 하여 어떤 "모델(model)" 을 만드는 것
    - 모델 : 예측 작업(회귀, 분류)을 수행하는 알고리즘

- 머신 러닝 모델
  - 스팸 메일을 찾아내는 모델
  - 누가 게임에서 이길지 예측하는 모델
  - 내일 날씨를 예측하는 모델

모델

- 수학, 과학에서는 어떤 현상을 설명하는 모델로 수식을 주로 사용
  - 모든 질량을 가진 모든 물체는 서로 끌어당긴다 : 만유인력 법칙

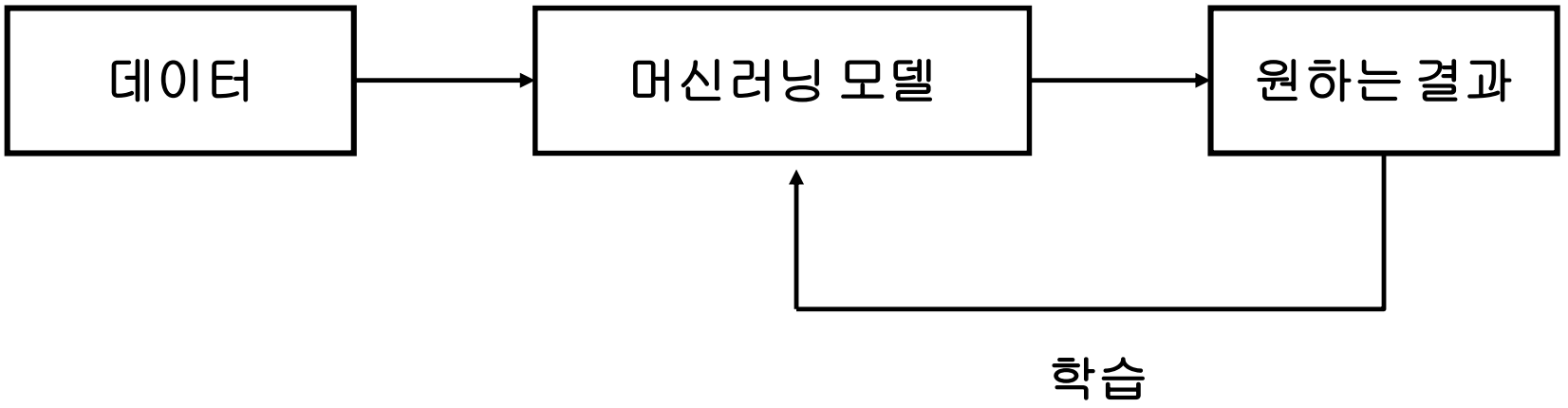


$$F_1 = F_2 = \frac{G \cdot m_1 \cdot m_2}{r^2}$$

- 활용
  - 자유낙하 물체의 속도를 정확하게 예측
  - 많은 관련 기구를 설계

머신 러닝의 기본 동작

- **머신러닝 목표**
  - 주어진 **학습 데이터**를 보고 **원하는 동작을 잘 수행**하는 **모델**을 만드는 것
  - 즉, 회귀 또는 분류 작업을 정확하게 수행
  - **좋은 모델**을 만들기 위해서는 → **좋은 데이터**가 필요



손실 함수

- **손실함수**(loss function)
  - 모델의 **예측값**과 **실제 값**과의 **차이**, 즉 **오차(error)**를 계산
- 이 **오차를 줄이는 방향**으로 모델을 **최적화**(**학습**) 한다
- **회귀분석**에서 **많이 사용**하는 손실함수
  - 오차 자승의 합의 평균치(MSE: mean square error)

$$MSE = \sum_{k=1}^{N} (y - \hat{y})^2$$

  - N: 배치 크기

- 배치 크기 같은 설정 환경 변수를 hyper parameter라고 한다.
  - **hyper parameter** : 사람이 선택하는 변수
  - **parameter** : 기계 학습으로 자동으로 갱신되는 변수

분류의 손실 함수

- 분류에서는 손실함수로 MSE를 사용할 수 없다
- 대신, 분류에서 정확도(accuracy)를 손실함수로 사용할 수 있다
  - 예) 100명에 대해 남녀 분류 문제
    - 96명을 맞추고 4명을 오 분류 : 정확도 0.96
  - 그러나 정확도를 손실함수로 사용하는 데에는 다음과 같은 문제가 있다
- Category 분포 불균형시 문제
  - 예)
    - Group : 남자 95명, 여자 5명
    - 오 분류 케이스 – 남자 1명, 여자 3명
    - 정확도는 여전히 0.96 :
    - 문제 : 여자의 경우, 5명 중 3명을 오 분류 ➔ 결과 심각
  - 데이터 분포가 비대칭인 상황 : 질병 진단의 경우 자주 발생
  - 손실을 제대로 측정하지 못함
  - 이를 보완하기 위해서 크로스 엔트로피(cross entropy)를 사용
    - Category가 둘 이상인 경우에도 동일한 개념으로 적용 가능

크로스 엔트로피(Cross Entropy)

$$CE = \sum_i p_i \log\left(\frac{1}{p_i'}\right)$$

- p_i : 어떤 사건이 일어날 실제 확률, p_i' : 예측한 확률
- 남녀가 50명씩 같은 경우

$$CE = -0.5 \times \log\left(\frac{49}{50}\right) - 0.5 \times \log\left(\frac{47}{50}\right) = 0.02687$$

- 남자가 95명 여자가 5명인 경우

$$CE = -0.95 \times \log\left(\frac{94}{95}\right) - 0.05 \times \log\left(\frac{2}{5}\right) = 0.17609$$

배치와 이포크

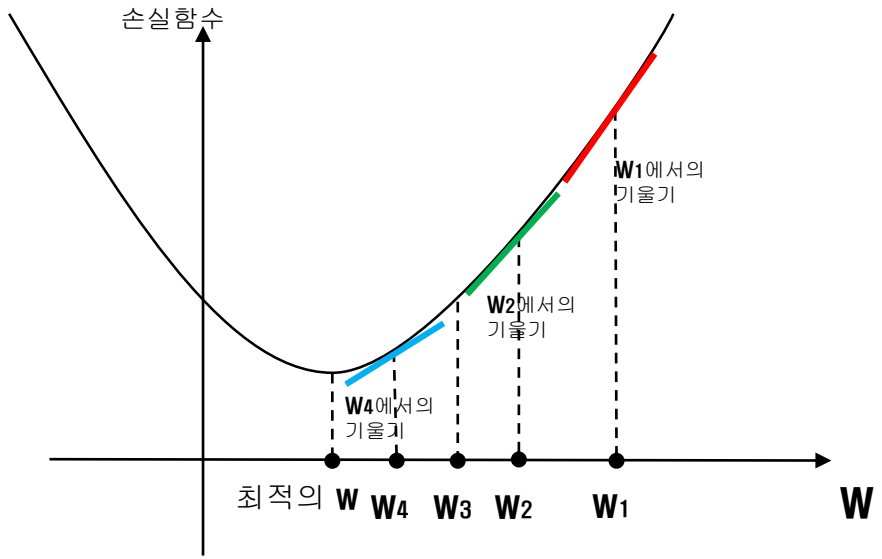
- 배치(Batch) 크기 : 한번 학습에 사용하는 샘플 수
- 예를 들어, 총 1,000개의 데이터로 예측 모델을 만들 때, 한번에 1,000개의 데이터를 모두 입력하여 손실함수의 값을 구하고 학습을 시키는 것은 비효율적
  - 배치 크기가 클수록 학습이 정교하고, 기울기를 정확히 구할 수 있으나 계산량이 많아진다.
  - 한번에 필요한 메모리 사용량이 많아 메모리 오류가 날 가능성이 높다
  - 일반적으로 훈련 데이터를 일정 크기의 배치 단위로 나누어 학습을 시키는 것이 효과적
- 배치 크기가 작을 때에는 기울기가 상대적으로 정확하게 계산되지 못하므로 학습률도 작게 잡는 것이 좋다

배치와 이포크

- 이포크(Epoch) : 주어진 훈련 데이터 전체를 한번 학습에 사용하는 것
- 학습에 주어진 1,000개의 훈련 데이터를 모두 학습에 사용하였어도 아직 최적의 모델 파라미터를 찾지 못했으면 주어진 데이터를 다시 반복하여 사용할 필요 있다
  - 머신러닝에서는 일반적으로 여러 이포크를 수행

훈련 방법 : 최적화 – 경사 하강법

- **경사 하강법(Gradient Descent)**
  - **가장 일반적인 최적화 알고리즘**
  - **손실함수를 계수에 관한 그래프로 그렸을 때 최소값으로 빨리 도달하기 위해서는 현재 위치에서의 기울기(미분값)에 비례하여 반대방향으로 이동**



$$W_i = W_{i-1} - \eta Grad(i)$$

경사 하강법의 원리

$$\underline{W}(n+1) = \underline{W}(n) - \mu \frac{\partial e^2(n)}{\partial \underline{W}(n)}$$

$$= \underline{W}(n) - \mu \frac{\partial e^2(n)}{\partial e(n)} \cdot \frac{\partial e(n)}{\partial \underline{W}(n)}$$

$$= \underline{W}(n) - 2\mu e(n) \cdot \frac{\partial \left[d(n) - \underline{W}^T(n) \cdot \underline{X}(n) \right]}{\partial \underline{W}(n)},$$

$$= \underline{W}(n) + 2\mu e(n) \underline{X}(n)$$

$$\frac{\partial \underline{A}^T \cdot \underline{B}}{\partial \underline{A}} = \underline{B}$$

학습률 (Learning Rate)

- **학습률: 학습 속도를 조정하는 변수**
  - **학습률이 너무 작게 잡으면**
    - **수렴하는데 시간이 오래 걸리지만 최저점에 도달했을 때 흔들림 없이 안정적인 값을 얻을 수 있다**
  - **학습률을 너무 크게 정하면**
    - **학습하는 속도는 빠르나 자칫하면 최저점으로 수렴하지 못하고 발산하거나 수렴하더라도 흔들리는 오차가 남아있을 수 있다.**
- **학습 스케줄(**learning schedule**) 기법**
  - **초기에는 학습률을 크게 정하고(학습을 빠르게 하고), 오차가 줄어들면 학습률을 줄여서 안정 상태(**steady state**)의 오차를 줄이는 방법**

경사 하강법 특징

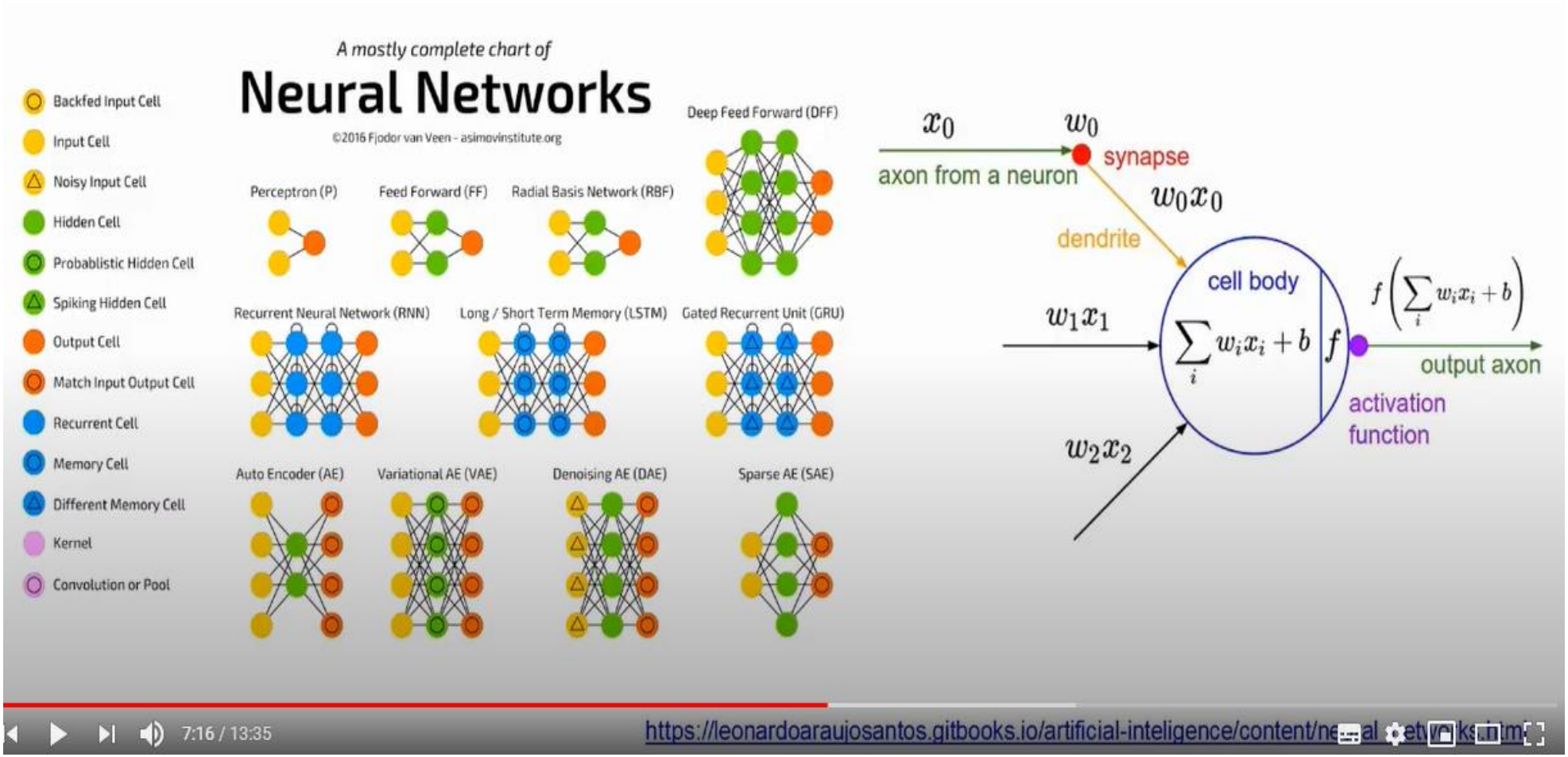
- 경사하강법을 적용하려면 특성 변수들을 모두 동일한 방식으로 스케일링해야 한다.
- 특성 값마다 크기의 편차가 크면
  - 특정 변수에 너무 종속되어 동작할 수 있고 이로 인해 수렴속도가 직선이 되지 않고 오래 걸릴 수가 있다.

경사 하강법의 종류

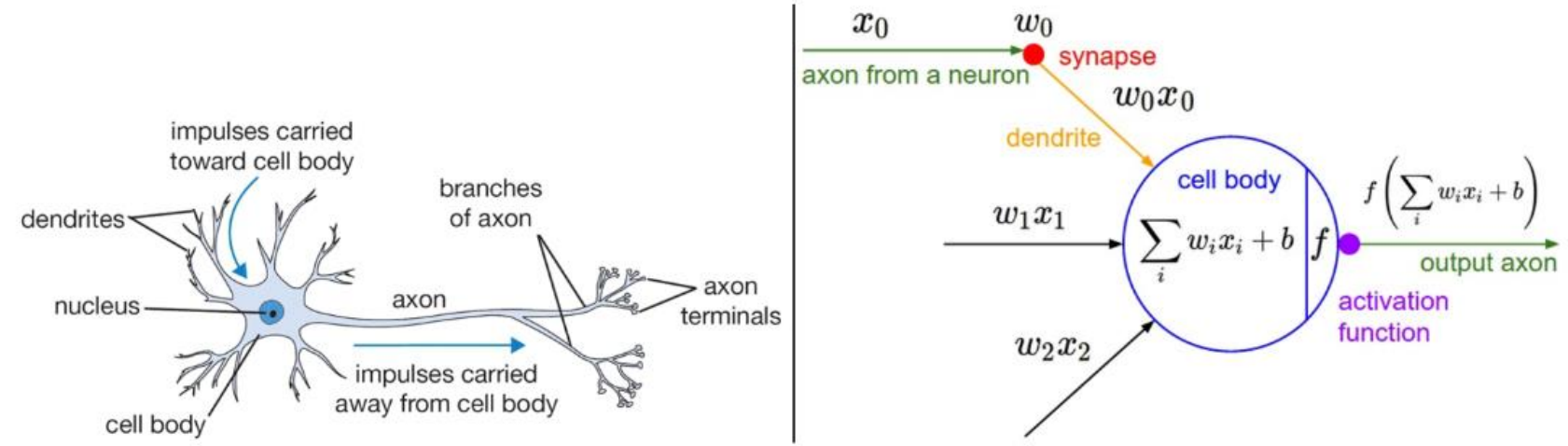
- **배치(Batch) GD**
  - **일반적으로 배치 GD방식을 많이 사용하는데, 적절한 크기의 배치 단위로 입력 신호를 나누어 경사 하강법을 적용하는 방식**
- **SGD (확률적 경사 하강법)**
  - **한 번에 한 샘플씩 랜덤하게 골라서 훈련에 사용하는 방법**
  - **즉 샘플을 하나만 보고 계수를 조정**
  - **계산량이 적어 동작속도가 빠르고, 랜덤한 방향으로 학습을 하므로 전역 최소치를 가능성이 높아진다**
  - **매 샘플이 너무 랜덤하여 방향성을 잃고 수렴하는데 시간이 오래 걸릴 가능성도 있다**

신경망(ANN)

Neural Net – Chart



신경망 개요



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

(출처: http://cs231n.github.io/neural-networks-1/)

- axon (축삭돌기) : 뉴런에서 뻗어나와 다른 뉴런의 수상돌기와 연결
- dendrite (수상돌기) : 다른 뉴런의 축삭 돌기와 연결, 몸체에 나뭇가지 형태로 붙어 있다
- synapse (시냅스) : 축삭돌기와 수상돌기가 연결된 지점, 여기서 한 뉴런이 다른 뉴런으로 신호가 전달

신경망 개요

- 하나의 뉴런은 여러 다른 뉴런의 축삭돌기와 연결
- 연결된 시냅스의 강도가 연결된 뉴런들의 영향력이 결정
- 이러한 영향력의 합이 어떤 값을 초과하면 신호가 발생하여 축삭돌기를 통해서 다른 뉴런에게 신호가 전달

  - x0, x1, x2 : 입력되는 뉴런의 축삭돌기로부터 전달되는 신호의 양
  - w0, w1, w2 : 시냅스의 강도, 즉 입력되는 뉴런의 영향력
  - w0*x0* + *w1*x1 + w2*x2 : 입력되는 신호의 양과 해당 신호의 시냅스 강도가 곱해진 값의 합계
  - f : 최종 합계가 다른 뉴런에게 전달되는 신호의 양을 결정짓는 규칙, 이를 **활성화 함수**라고 부름

신경망과 딥러닝

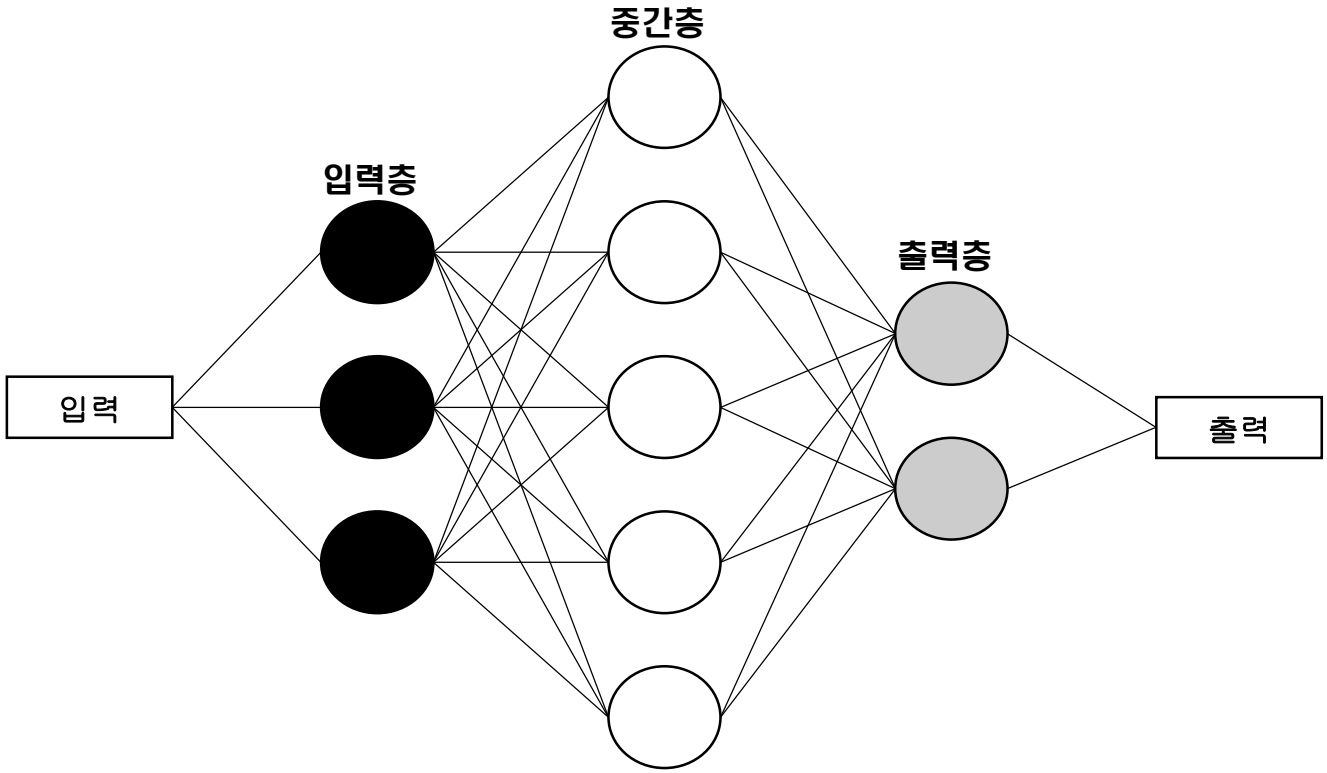
- 신경망 모델
  - 뇌를 구성하는 신경 뉴런(neuron)의 동작을 모방
  - 기본적으로는 입력 신호 벡터에 어떤 가중치를 곱하고 그 결과를 더하거나 비선형 처리를 하여 유용한 정보를 추출하는 구조
  - 이러한 작업을 계층(layer) 단위로 여러 번 수행

- 딥(deep) 러닝 모델
  - 계층이 2개 이상인 신경망 모델

- 성능
  - 이제 거의 사람처럼 듣고, 읽고, 보고 쓰는 능력이 발전
  - 2012년 ILSVRC(ImageNet Scale Visual Recognition Challenge)에서 기존의 알고리즘보다 월등히 우수한 성능을 보이면서 (AlexNet) 딥러닝이 머신러닝 모델로 널리 채택되기 시작

신경망 구성 – 기본 구조

$$Y = WX + b$$



텐서 (tensor)

- 신경망에서는 **계층별 입출력 정보**를 벡터 보다 **텐서**라고 부름

- 텐서는 개념적으로는 벡터와 같은 의미이나 신경망에서는 다차원 벡터를 통칭하여 텐서라고 하고, "벡터"라고 하면 아래와 같이 1차원이면서 항목이 **여러 개**인 신호를 지칭한다.

  - X = [-0.82, 0.94, -1.15, 0.25]

- 텐서에서는 차원 (dimension) 이라는 표현대신 **랭크**(rank)를 사용하는데, 벡터는 랭크가 1인 텐서이다.

텐서 (tensor) – 예

- rank = 2
  - 행(row) : 입력 신호의 sample들
  - 열(column) : 각 sample의 특성(feature) 값

```
X = [[-0.82, -0.25, -1.16, -0.64, 1.13, -0.68],
     [-0.18, -0.42, 1.34, -0.21, -0.57, 1.23],
     [ 2.32, 1.22, -0.43, 0.2 , -0.29, -0.88],
     [-2.02, -0.44, 0.2 , 0.5 , 0.01, -0.88]]
```

- 만일 입력 신호 X가 200 장의 컬러 이미지이고, 각 이미지가 (244, 244) 크기의 픽셀로 구성되며, 각 픽셀이 3원색으로 구성되어 있다면 X의 모양은 다음과 같으며 랭크는 4가 된다.

  - X.shape = (200, 244, 244, 3)

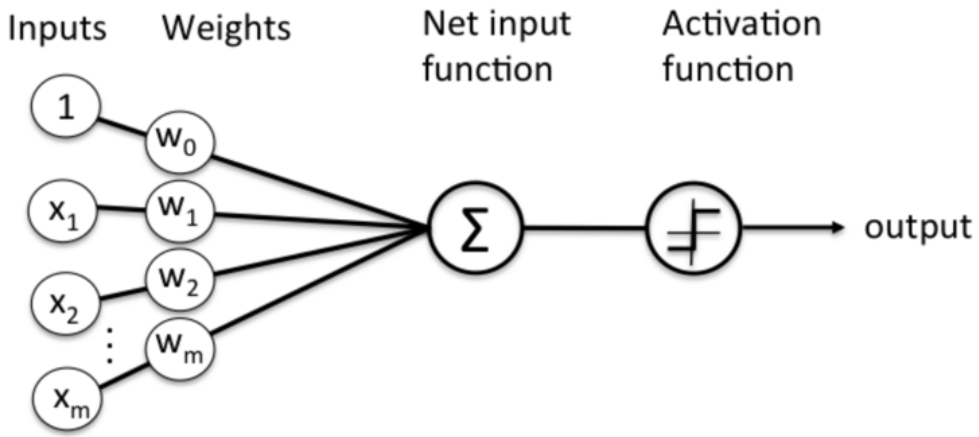
네트워크 (network)

- 신경망 모델은 보통 여러개의 계층으로 구성되며 이를 “네트워크”라 함

퍼셉트론 (Perceptron)

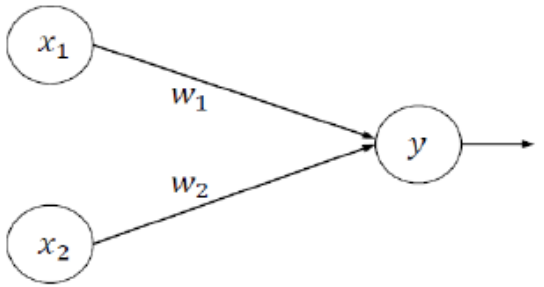
- 인공신경망의 개념
  - 1943년에 최초로 제안
  - 신경망은 인공신경망(Artificial Neural Net, ANN)을 줄여서 부름

- 퍼셉트론(Perceptron)
  - 신경망의 최초 모델



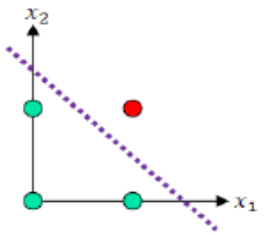
퍼셉트론 (Perceptron)

- 퍼셉트론은 실제로 간단한 XOR 기능도 구현하지 못하는 것으로 판명
  - 실효성에 의문
- 후에 XOR 과 같은 연산은 여러 층으로 구성된 다층 퍼셉트론 (MLP)으로 해결될 수 있다는 것이 증명



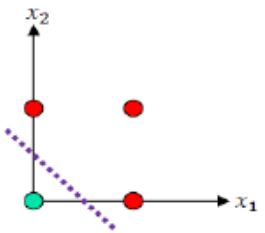
$$y = \begin{cases} 0 & (w_1 \cdot x_1 + w_2 \cdot x_2 \leq \theta) \\ 1 & (w_1 \cdot x_1 + w_2 \cdot x_2 > \theta) \end{cases}$$

AND



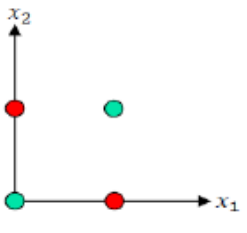
$w_1 = 1.0$
$w_2 = 1.0$
$\theta = 1.5$

OR



$w_1 = 1.0$
$w_2 = 1.0$
$\theta = 0.5$

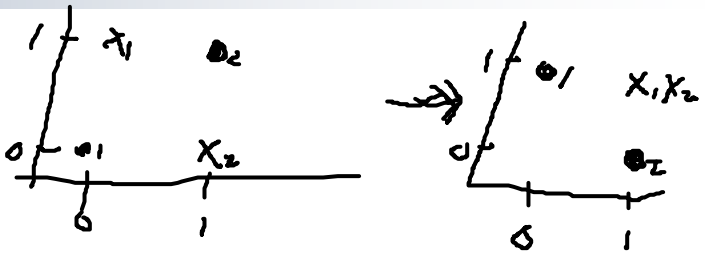
XOR



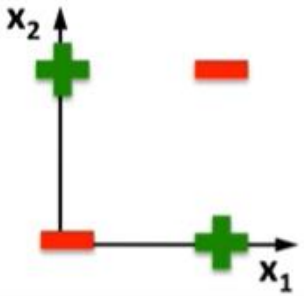
$w_1 = ?$
$w_2 = ?$
$\theta = ?$

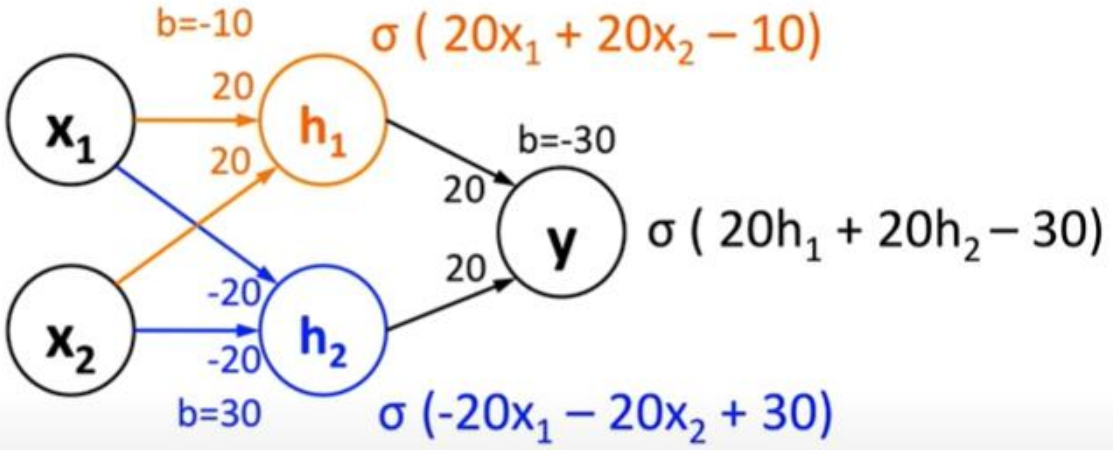
퍼셉트론 (Perceptron)

- Solving XOR Problem using MLP



Linear classifiers
cannot solve this





σ(20*0 + 20*0 – 10) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1
σ(20*0 + 20*1 – 10) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1

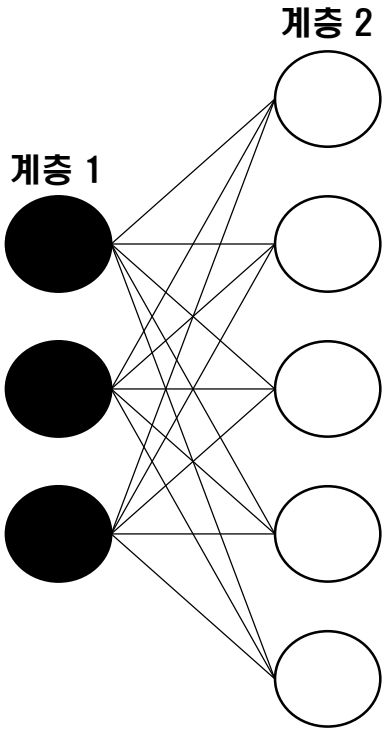
σ (-20*0 – 20*0 + 30) ≈ 1
σ (-20*1 – 20*1 + 30) ≈ 0
σ (-20*0 – 20*1 + 30) ≈ 1
σ (-20*1 – 20*0 + 30) ≈ 1

σ (20*0 + 20*1 – 30) ≈ 0
σ (20*1 + 20*0 – 30) ≈ 0
σ (20*1 + 20*1 – 30) ≈ 1
σ (20*1 + 20*1 – 30) ≈ 1

(Ref: https://www.youtube.com/watch?v=kNPGXgzxoHw)

전 결합망 (Fully Connected Network, FCN)

- 가장 기본적인 신경망 구조
- 각 계층의 **모든 출력**이 다음 단계의 **모든 입력으로 연결**되는 선이 존재하고 여기에 가중치가 곱해지는 구조
  - 가중치를 곱하는 것 외에 스칼라 값인 편이(bias)가 더해진다.



$$Y = \Sigma(W_{ij} \times X_{ij} + b_i)$$

활성화 함수 (Activation Function)

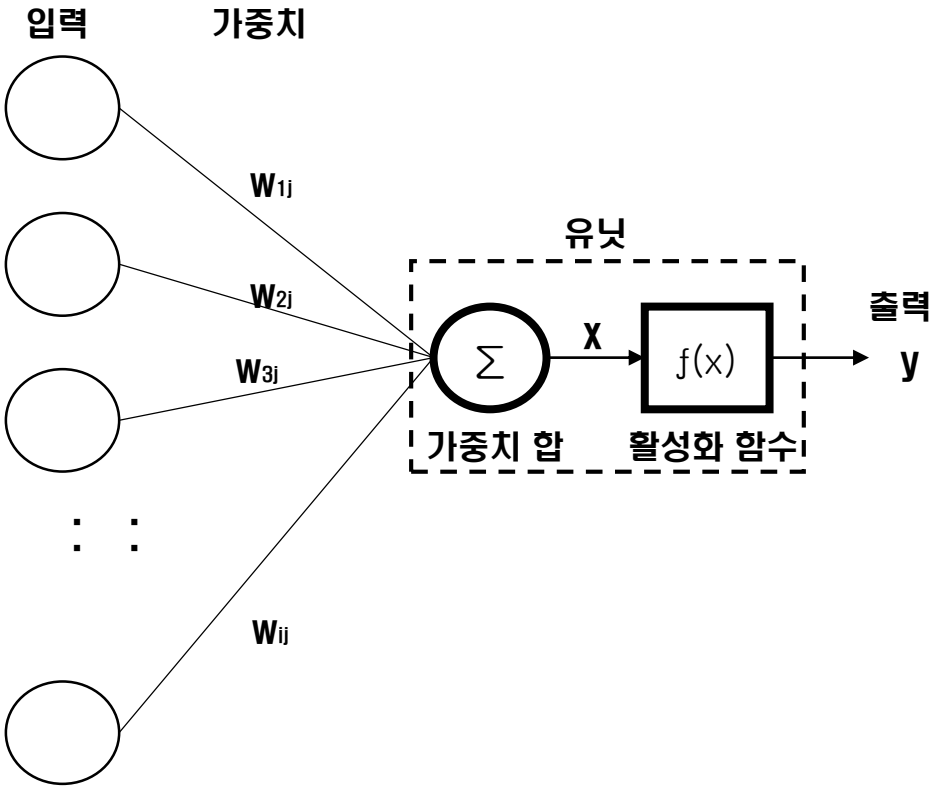
- 신경망은 (선형회귀와 달리) 한 계층의 신호를 다음 계층으로 그대로 전달하지 않고 비선형적인 활성화 함수를 거친 후에 전달

- 활성화 함수 사용 이유
  - 생물학적인 신경망을 모방
  - 약한 신호는 전달하지 않고, 어느 이상의 신호도 전달하지 않는 “S”자 형 곡선과 같이 “비선형적”인 반응을 한다고 생각

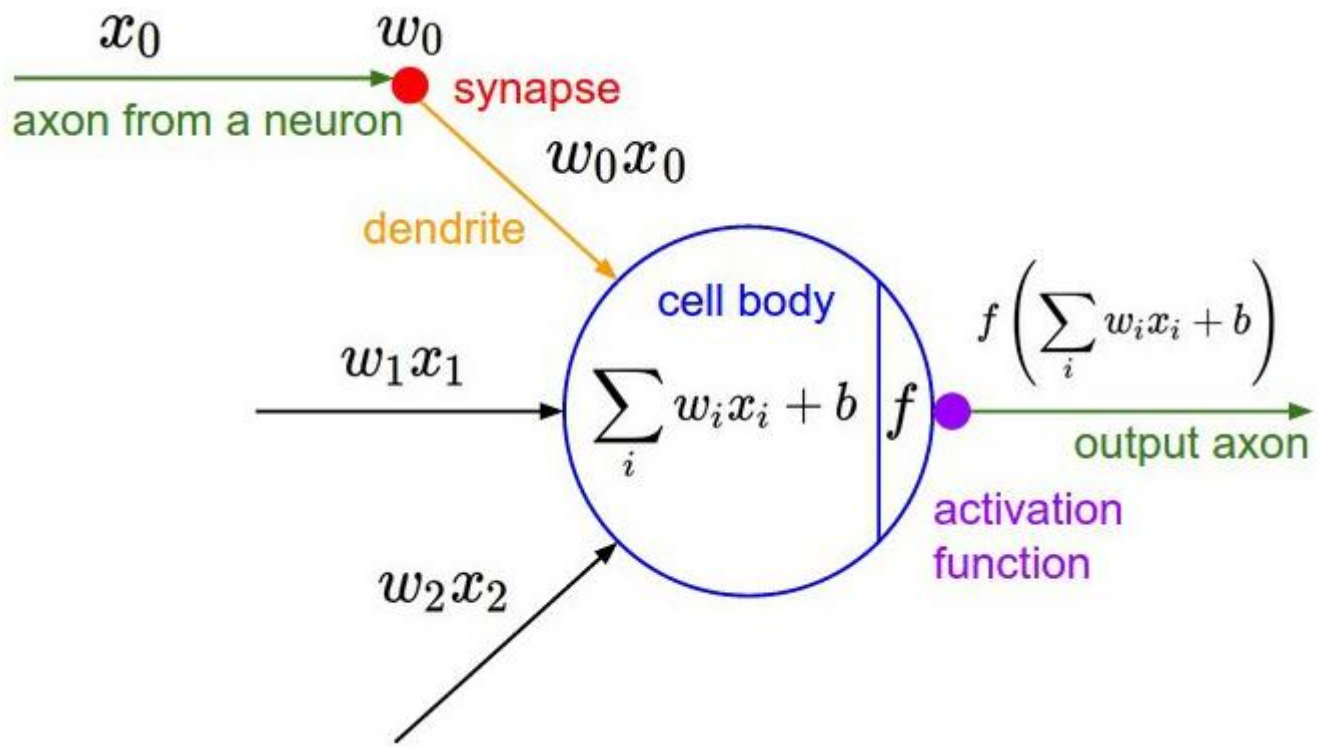
- 실제로 비선형의 활성화 함수를 도입한 신경망이 잘 동작

활성화 함수 (Activation Function)

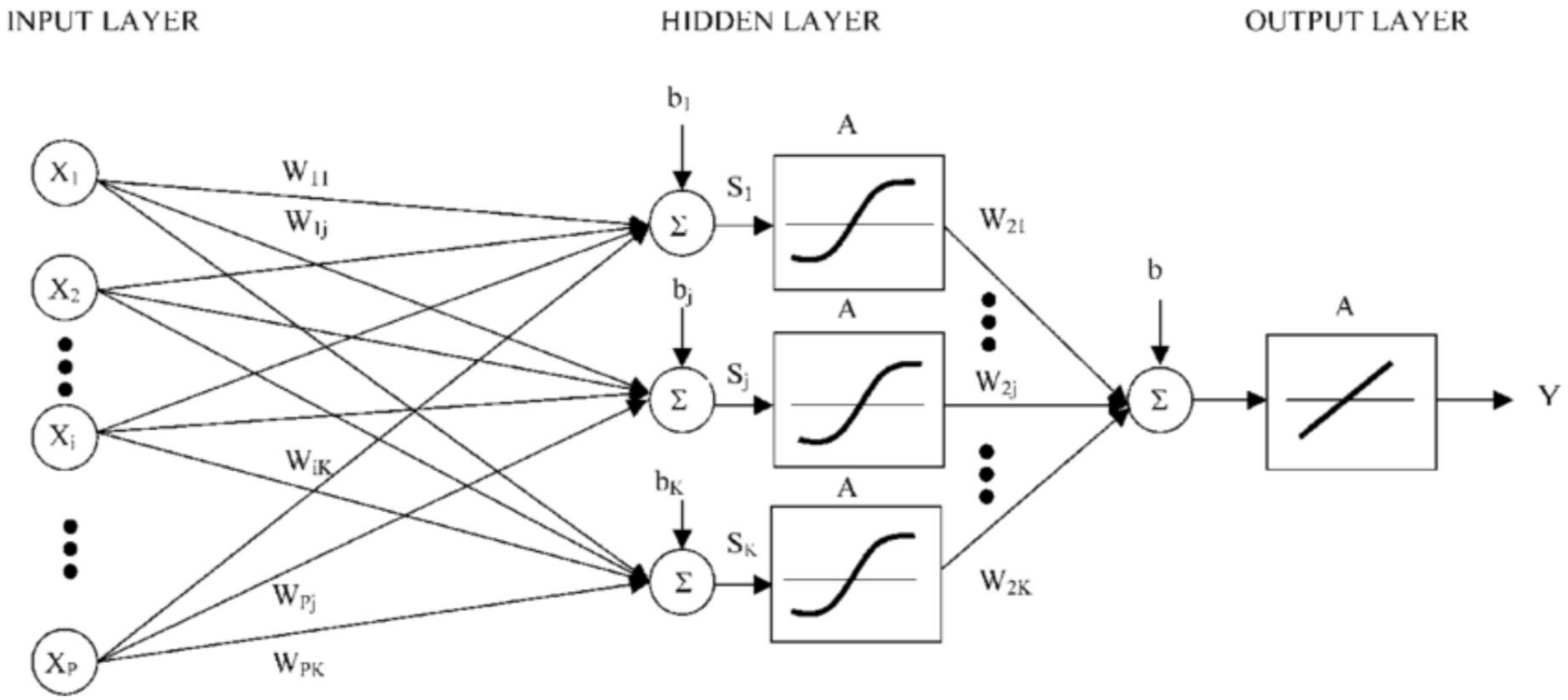
- 다음 계층으로 신호를 전달할 때, 어떤 범위의 신호를 "활성화(activate)" 하여 전달할지를 정하는 함수



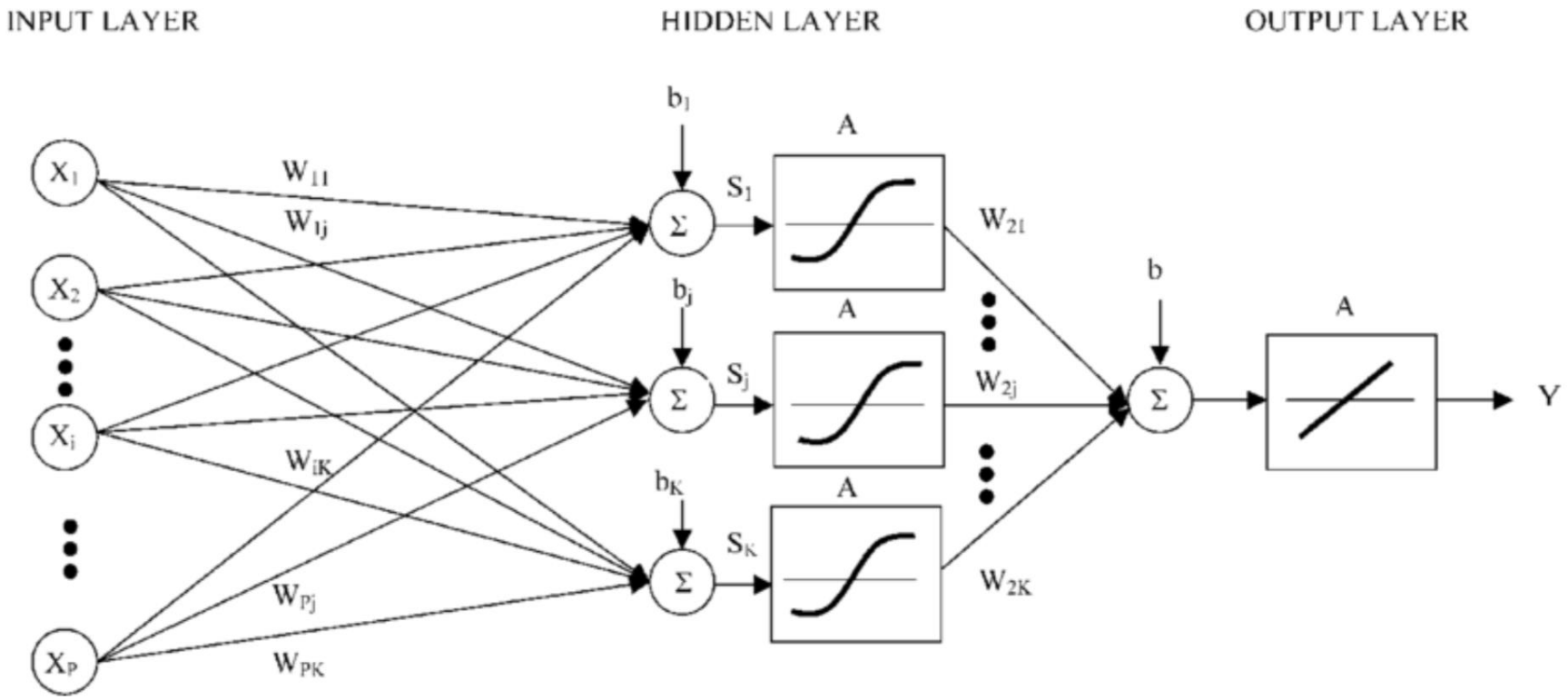
활성화 함수 (Activation Function)



활성화 함수 (Activation Function)



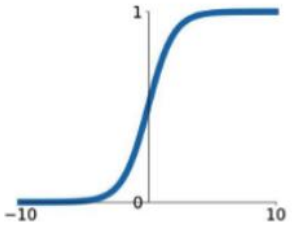
활성화 함수 (Activation Function)



활성화 함수 종류

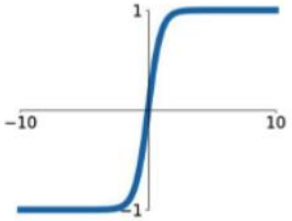
Sigmoid

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



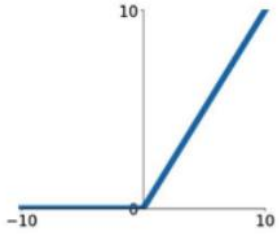
tanh

$$\tanh(x)$$



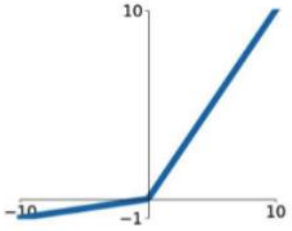
ReLU

$$\max(0, x)$$



Leaky ReLU

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

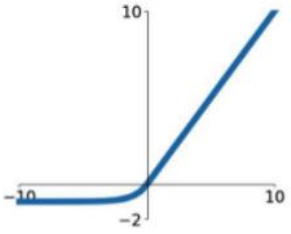


Maxout

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

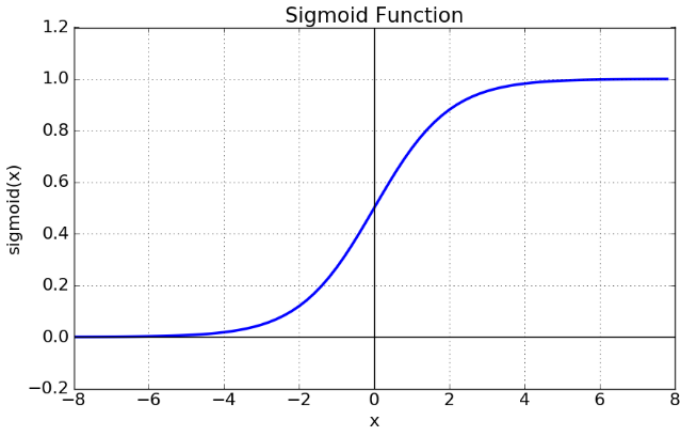
ELU

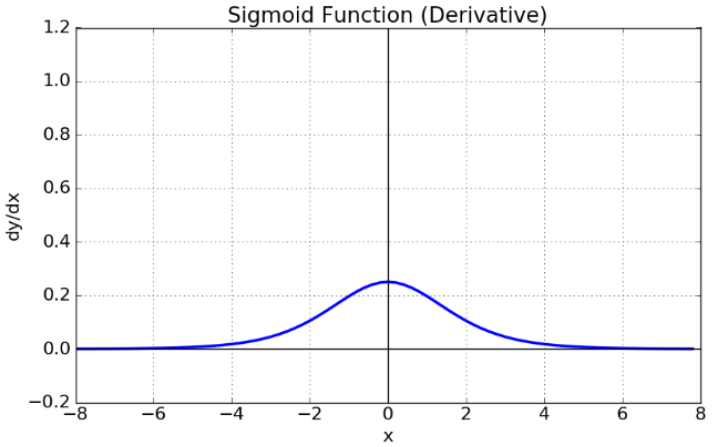
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



활성화 함수 종류

- 시그모이드(Sigmoid) 함수
  - 입력과 출력의 관계가 S 커브를 따른다고 가정한 함수, 생물학적으로 입출력 관계를 설명할 때 많이 사용되는 모델
  - 출력을 0~1 사이로 제한
  - 단점 : 입력값이 일정 범위를 벗어나서 너무 크거나 작으면 모두 1이나 0으로 일정하게 평탄해져서 큰 신호는 무시된다는 문제가 있다. 또한 이 범위에서는 기울기가 모두 0이므로 입력의 변화에 반응하지 못하는 한계





활성화 함수 종류

- Hyperbolic tangent (tanh) 함수
  - 시그모이드와 유사한 모양
  - 출력의 범위가 다르다 : -1 ~ 1 사이
  - tanh함수 역시 크거나 작은 값이 무시되는 단점이 존재

- ReLU 함수
  - 입력이 양수이면 신호를 그대로(선형적으로) 전달
  - 입력이 음수이면 출력을 '0'으로 제한
  - 크기가 **큰 신호**를 출력으로 전달하는 구조를 가지며, 시그모이드를 사용할 때 발생하는 Vanishing Gradient 문제를 해결
  - 현재 많은 신경망에서 ReLU를 채택

활성화 함수 선택

- **활성화 함수의 선택**
  - 신경망의 각 계층마다 다르게 택할 수 있다

- **활성화 함수를 사용하지 않는다는 것**
  - 신호를 선형적으로 그대로 출력으로 전달
  - 이는 신경망을 **회귀분석**에 사용할 때 필요

- **분류 문제나 어떤 사건의 발생 확률을 구할 때**
  - 즉, 0~1 범위의 확률에 해당하는 값을 예측할 때에는 출력단에서 시그모이드 함수를 주로 사용

- **다중 분류를 사용할 때**
  - 소프트맥스(softmax)를 사용

다층 퍼셉트론 (MLP)

다층 퍼셉트론 (MLP, Multi Layer Perceptron)

- 전결합망과 활성화 함수를 합한 기능이 하나의 계층(layer)이며 이러한 계층을 여러 개 쌓은 구조

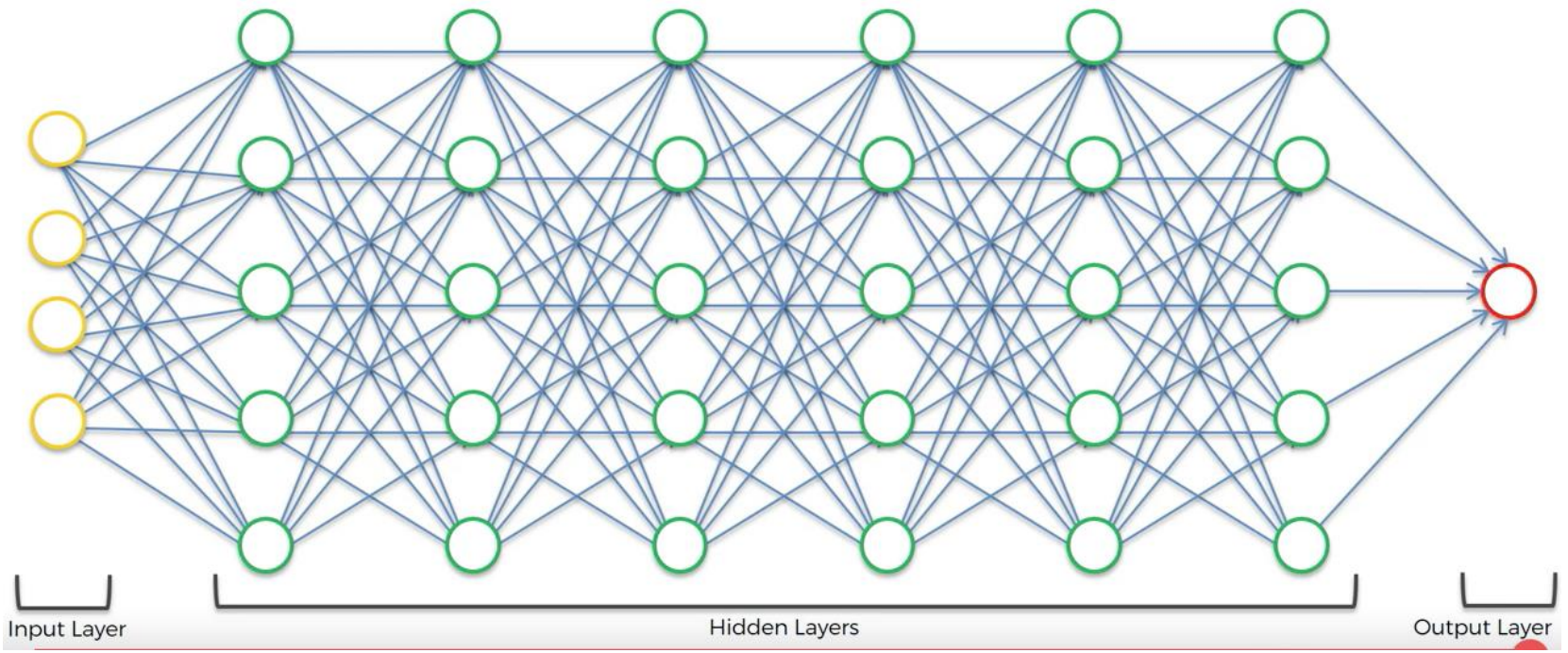
- 은닉(hidden) 계층
  - 입력과 출력 계층을 제외한 중간 계층

- 초기 신경망은 MLP로 구성했는데 필기체 숫자 인식 등에서 상용화되기도 함

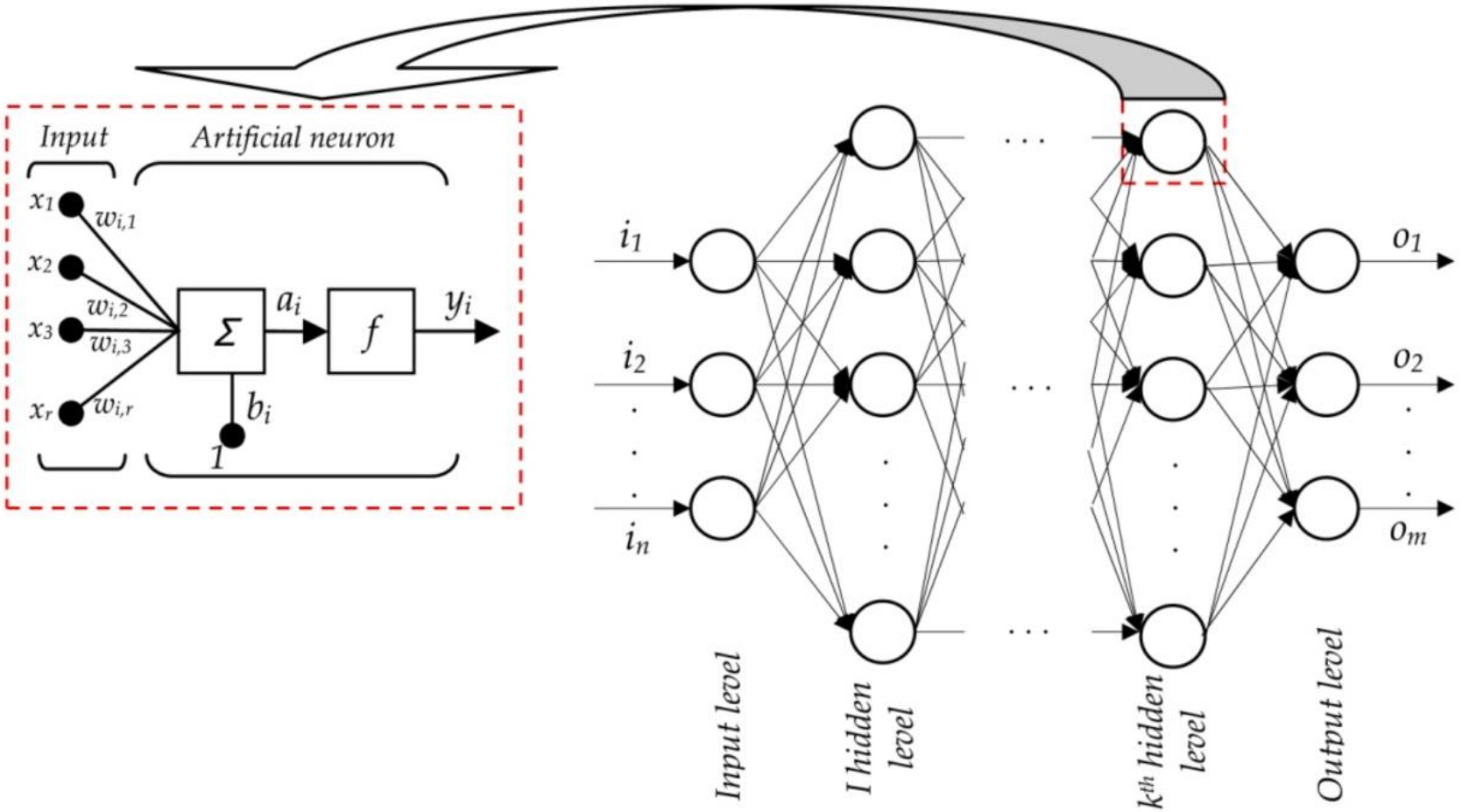
- 지금은 MLP를 개선한 CNN, RNN 등이 신경망으로 사용되며 성능도 급격히 향상됨

다층 퍼셉트론 (MLP)

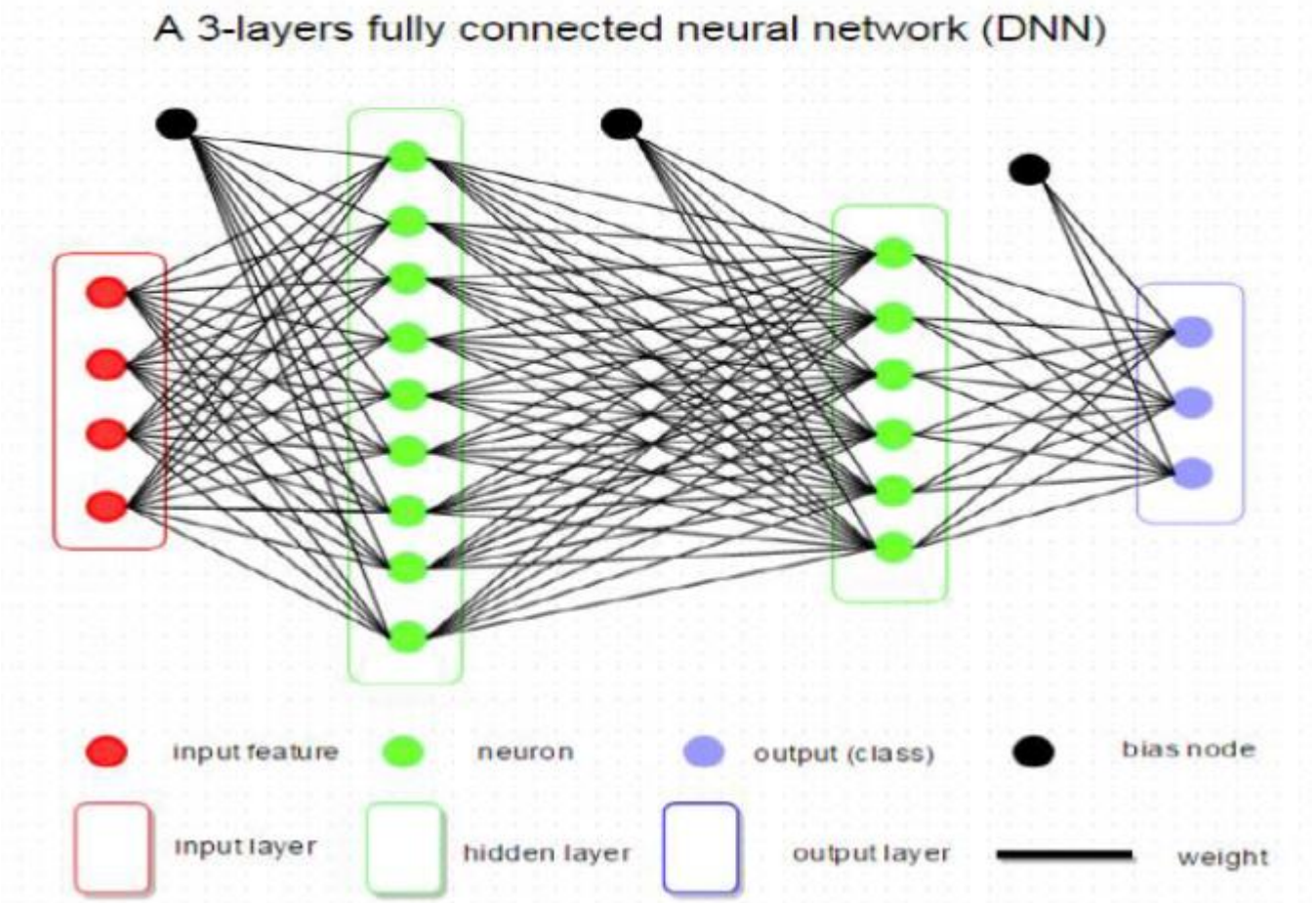
- 은닉(hidden) 계층
  - 입력과 출력 계층을 제외한 중간 계층



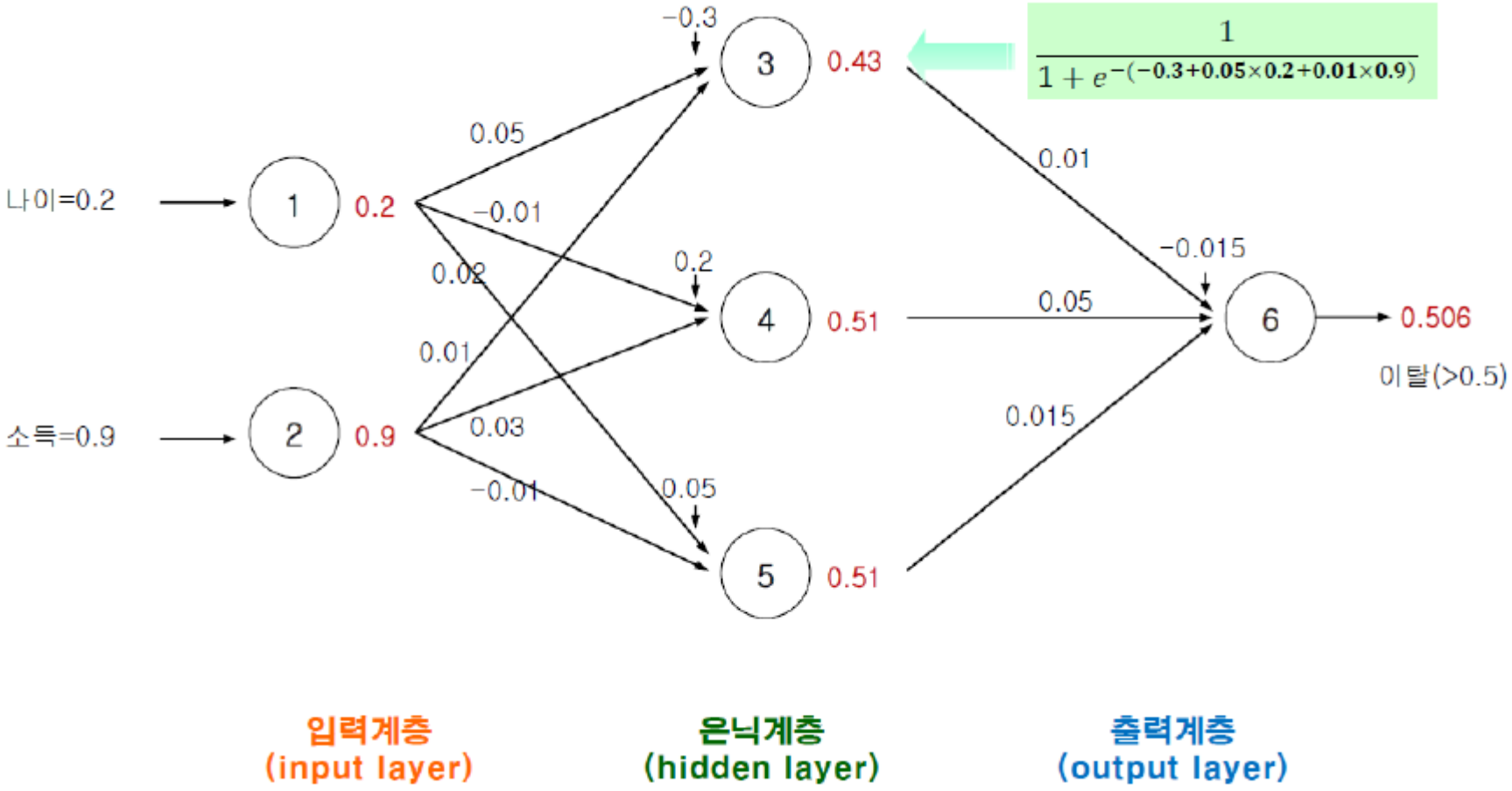
다층 퍼셉트론 (MLP)


Input
Artificial neuron
x_1
x_2
x_3
x_r
$w_{i,1}$
$w_{i,2}$
$w_{i,3}$
$w_{i,r}$
Σ
a_i
f
y_i
b_i
1
i_1
i_2
i_n
o_1
o_2
o_m
Input level
I hidden level
k^{th} hidden level
Output level

다층 퍼셉트론 (MLP)

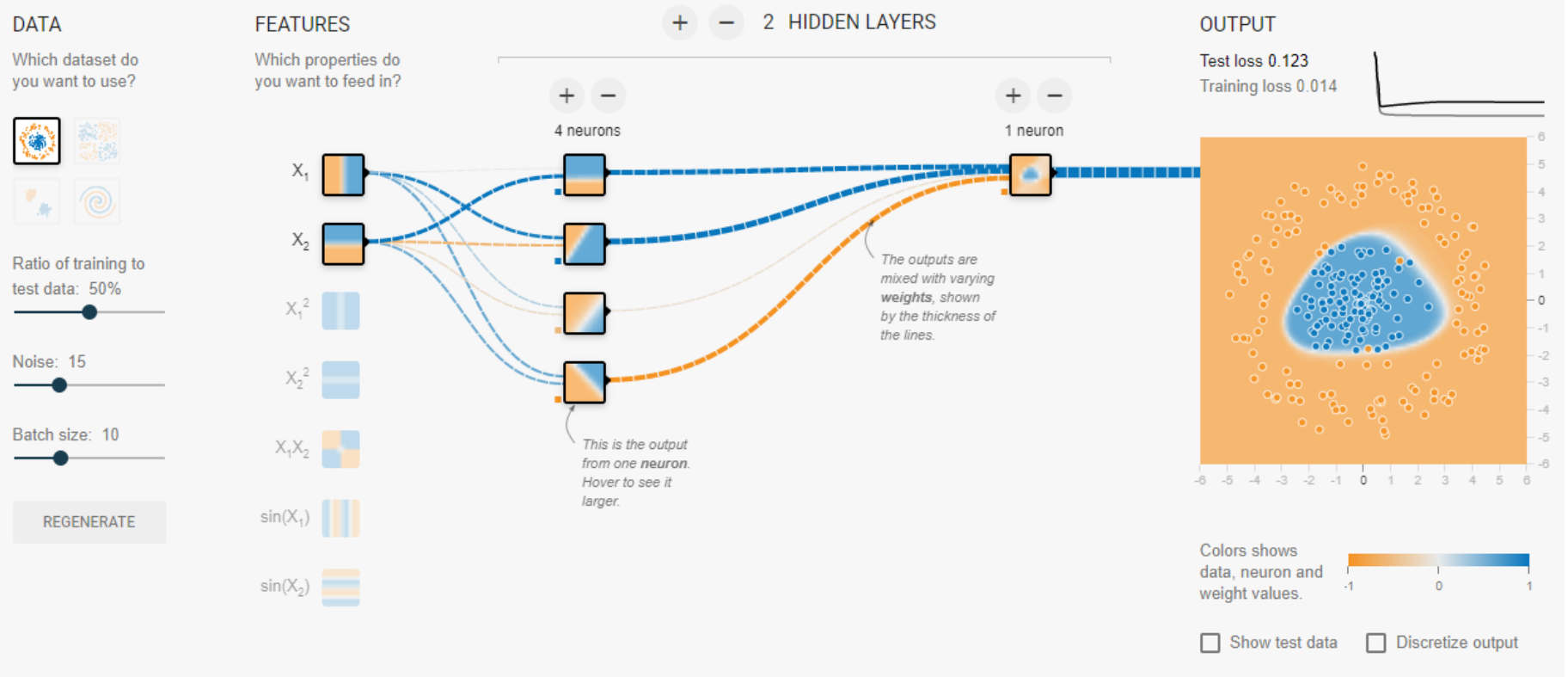

A 3-layers fully connected neural network (DNN)
input feature
neuron
output (class)
bias node
input layer
hidden layer
output layer
weight

다층 퍼셉트론 (MLP)


나이=0.2
소득=0.9
1
2
0.2
0.9
3
4
5
0.43
0.51
0.51
−0.3
0.2
0.05
0.05
−0.01
0.02
0.01
0.03
−0.01
0.05
0.01
−0.015
0.05
0.015
6
0.506
이탈(>0.5)
$\frac{1}{1+e^{-(-0.3+0.05\times0.2+0.01\times0.9)}}$
입력계층
(input layer)
은닉계층
(hidden layer)
출력계층
(output layer)

신경망 동작

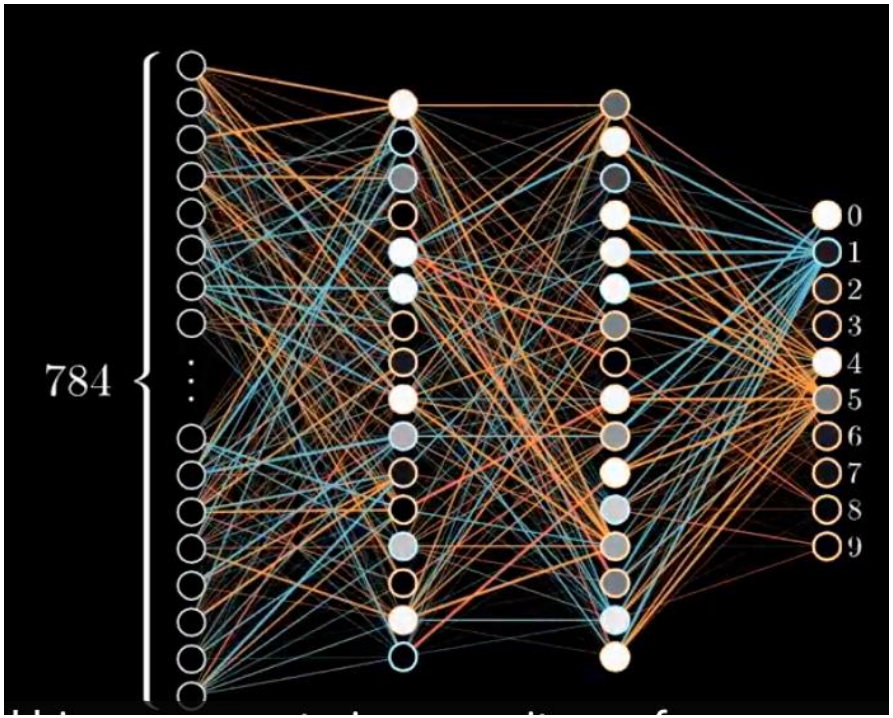
- 신경망 (Playground): https://playground.tensorflow.org/



- 입력 데이터의 분포가 복잡해질수록
  - 더 깊은 신경망 구조와 다수의 신경 유닛이 필요

신경망 동작 상세 설명

- 동영상 (21:00)



역전파 (Back Propagation)

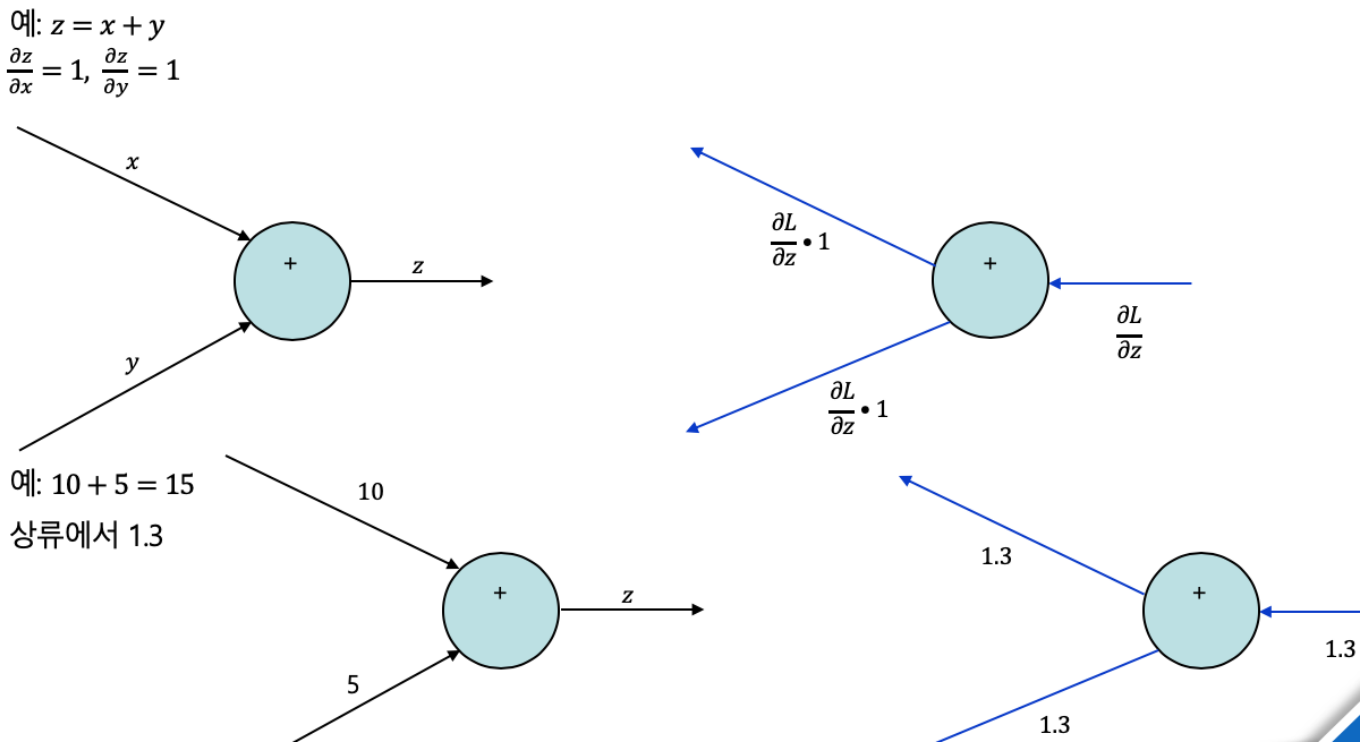
- 신경망 구현에서 초기에 어려웠던 점은 경사하강법을 여러 계층을 통하여 구현하는 방법을 찾지 못했던 것이다.

- 그러나 신경망을 구성하는 내용이 덧셈과 곱셈만으로 구성된다면 이의 경사하강법을 구현하는 것이 간단한 계산으로 가능하다는 것을 알게 되었다.

- 즉 오차를 줄이기 위한 경사하강법이 역전파(back propagation)라는 방법으로 간단하게 구현할 수 있다.

역전파 – 덧셈

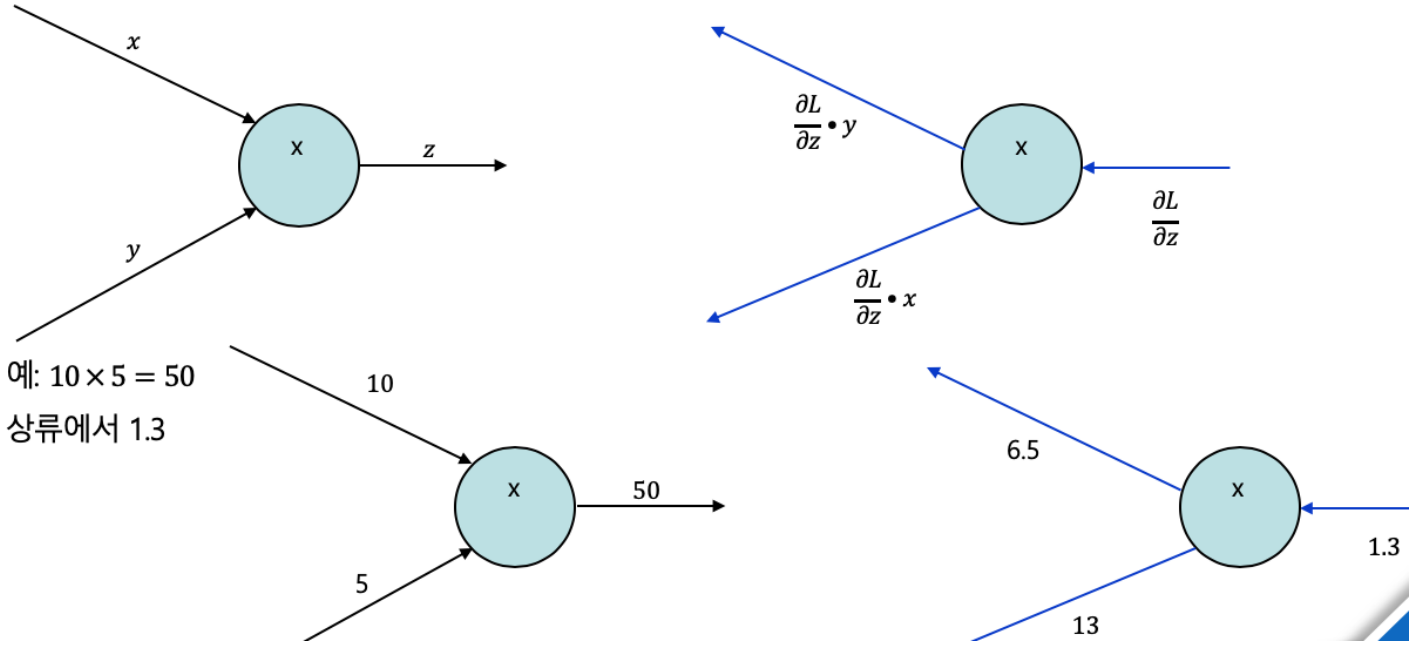
- 아래 그림은 합과 곱으로 구현된 신경망 기본 구조에 대해 역전파가 어떻게 이루어지는 지를 나타냈다.



역전파 – 곱셈

예: $z = xy$

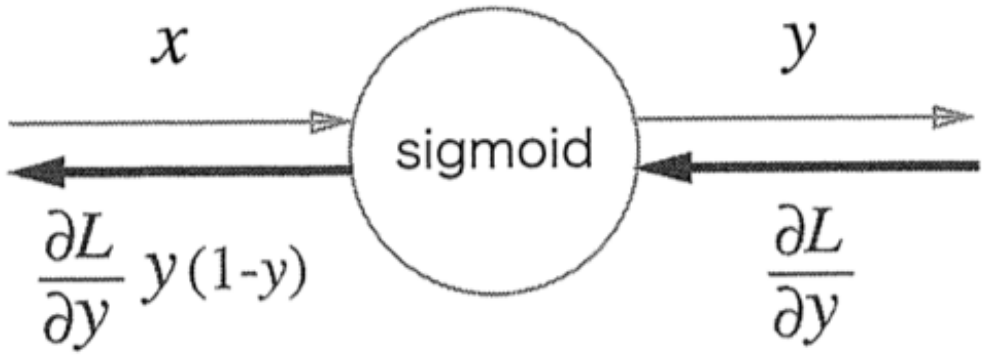
$\frac{\partial z}{\partial x} = y, \frac{\partial z}{\partial y} = x$



예: $10 \times 5 = 50$

상류에서 1.3

역전파 – 활성화 함수(sigmoid) 통과



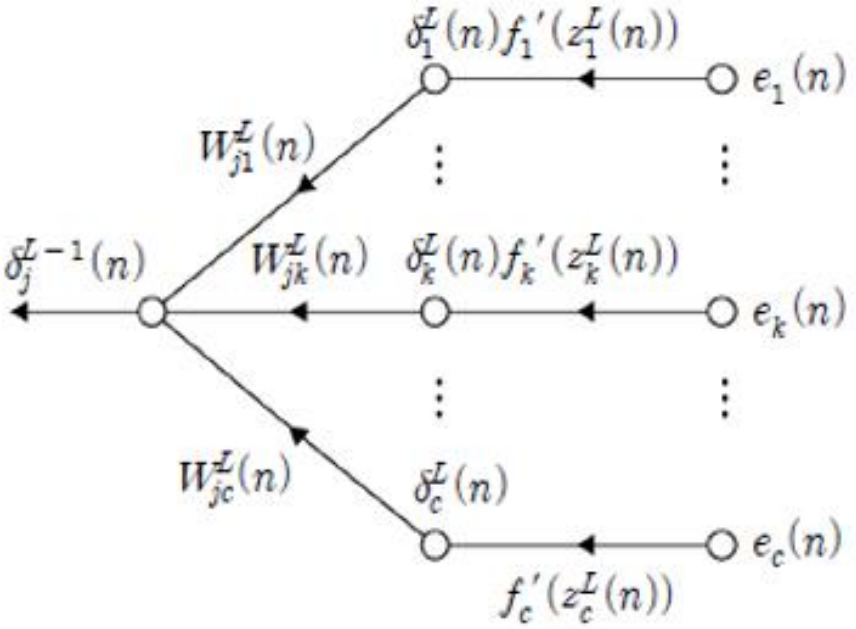
오류 역전파 알고리즘 (EBP)

1. 신경망 모형의 구조, 학습률 및 중지규칙 등을 결정하고 가중치를 초기화한다.
2. 훈련 데이터 $\mathfrak{I}$의 N개 관측값을 어떤 형식으로 정렬한 후 신경망에 제시한다.
3. 전방향 계산forward computation
   - $z_j^l(n) = \sum_{i=1}^{m_l} W_{ij}^l(n) h_i^{l-1}(n)$, $l = 1, \cdots, L$, L: 출력층
   - $h_j^l(n) = f_j(z_j^l(n))$, $1 = 2, \cdots, L$, $h_j^1(n) = x_j(n)$, $h_j^L(n) = o_j(n)$
   - $e_j(n) = y_j(n) - o_j(n)$
4. 역방향 계산backward computation
   - $\delta_j^L(n) = -e_j^L(n) f_j'(z_j^L(n))$, 노드 j는 출력층 L의 노드이다.
   - $\delta_j^l(n) = -f_j'(z_j^l(n)) \sum_k \delta_k^{l+1}(n) W_{jk}^{l+1}(n)$, 노드 j는 은닉층 l의 노드이다.
5. 각 관측값에 대해 다음 델타규칙에 따라 가중치를 갱신한다.
   - $W_{ij}^l(n+1) = W_{ij}^l(n) + \eta \delta_j^l(n) h_i^{l-1}(n)$
6. 중지규칙이 충족될 때까지 단계 2부터 단계 5까지 반복한다.

역전파 – 오류 역전파

$$\Delta W_{ij}^{l}(n) = \eta \times \delta_{j}^{l}(n) \times h_{i}^{l-1}(n)$$

가중치 변화량		학습률 매개변수		국부적 기울기		노드 j의 입력



가중치 초기화 (추천)

활성함수	균등분포 $U(-r, r)$	정규분포 $N(0, \sigma^2)$
시그모이드	$r = \sqrt{\frac{6}{m_l + m_{l+1}}}$	$\sigma = \sqrt{\frac{2}{m_l + m_{l+1}}}$
쌍곡탄젠트	$r = 4\sqrt{\frac{6}{m_l + m_{l+1}}}$	$\sigma = 4\sqrt{\frac{2}{m_l + m_{l+1}}}$
ReLU	$r = \sqrt{2}\sqrt{\frac{6}{m_l + m_{l+1}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{m_l + m_{l+1}}}$

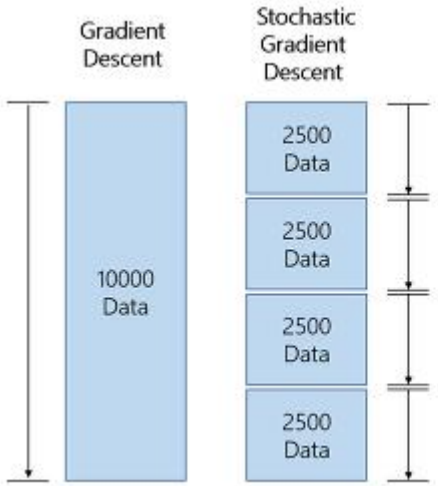
m_l : l 번째 층의 노드 수

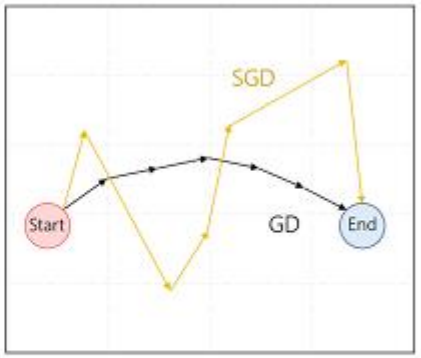
Optimizer (최적화기)

$$w^{+} = w - \eta * \frac{\partial E}{\partial w}$$

learning rate :
한번에 얼마나 학습할지

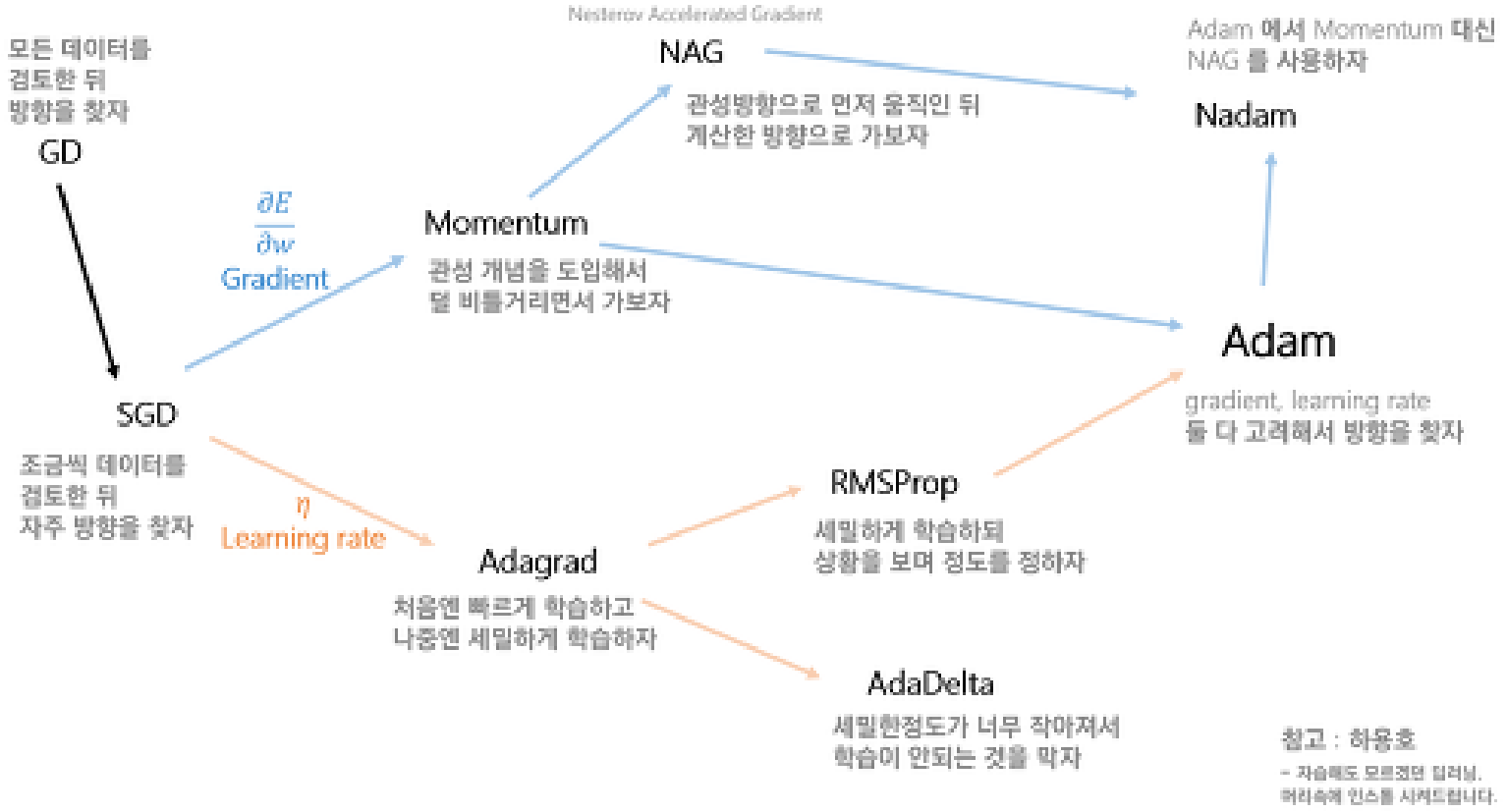
gradient :
어떤 방향으로 학습할지





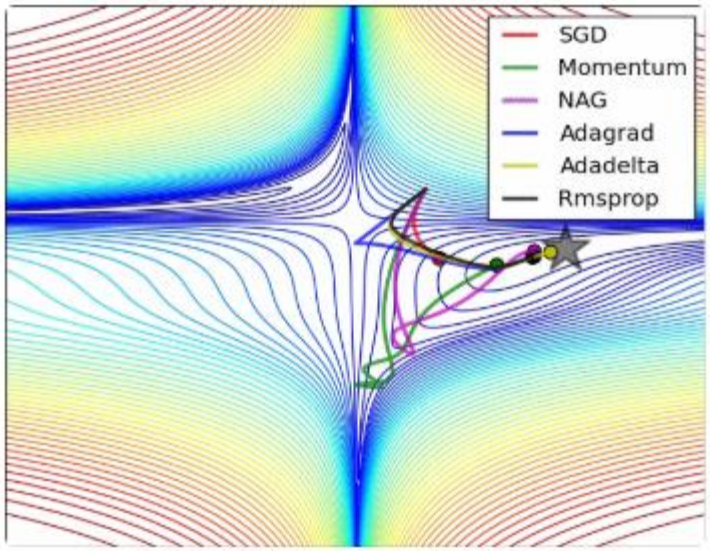
더 비틀비틀 가는 것 같지만
기존 보다 더 빠르게 학습합니다

Optimizer (최적화기)

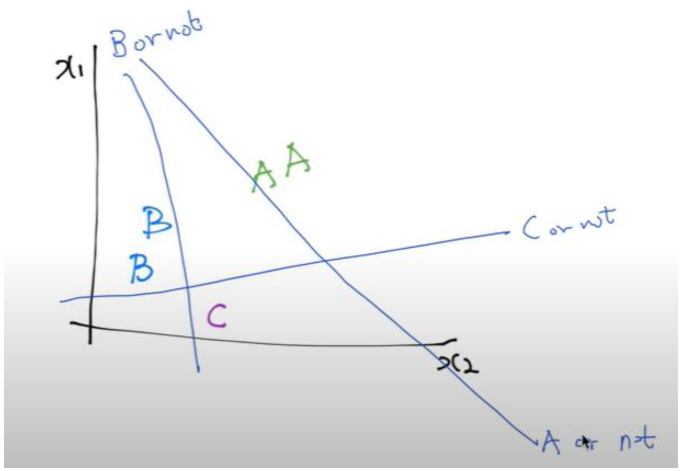


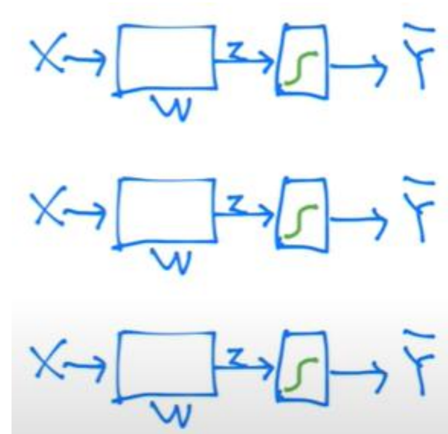
Optimizer (최적화기)

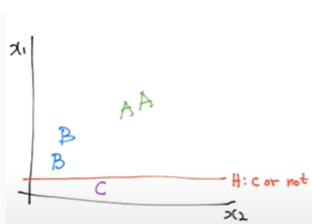
Optimizer 성능 비교

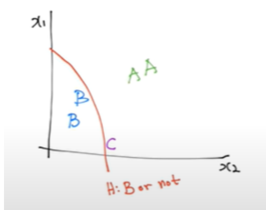


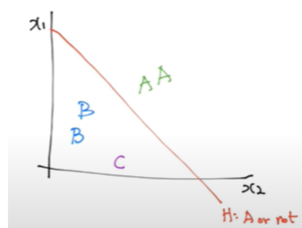
Multinomial Classification

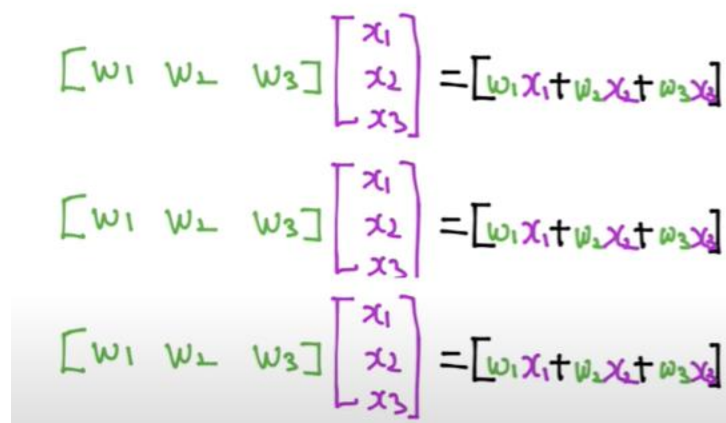








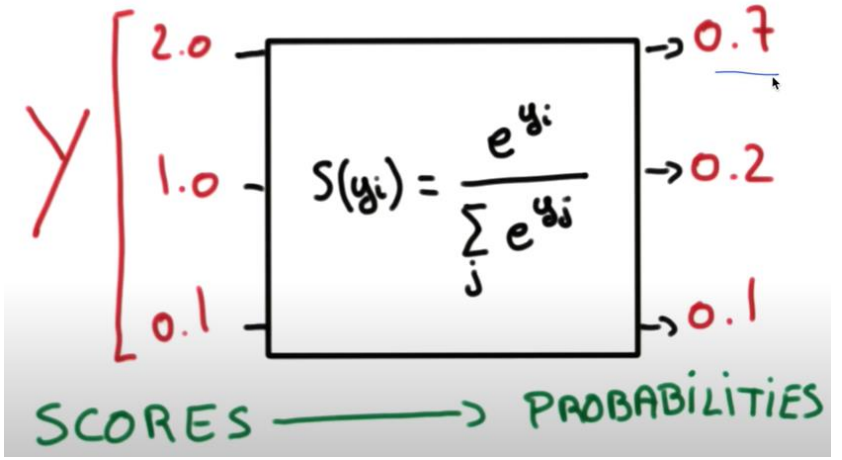




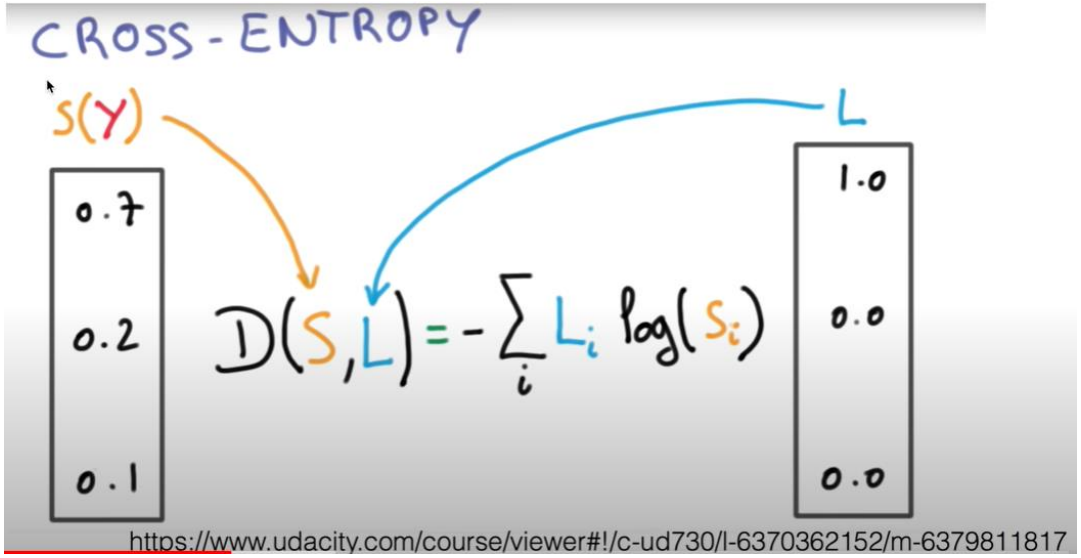
From: Prof. Kim Sunghun's lecture youtube

Multinomial Classification

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \overline{y}_A \\ \overline{y}_B \\ \overline{y}_C \end{bmatrix} \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$



Cross Entropy



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

$$D(S, L) = -\sum_i L_i \log(S_i)$$

Binary cross entropy

cost (binary cross-entropy)

Cross-entropy cost

Cross entropy – Softmax

- **교차 엔트로피 (Cross entropy)**

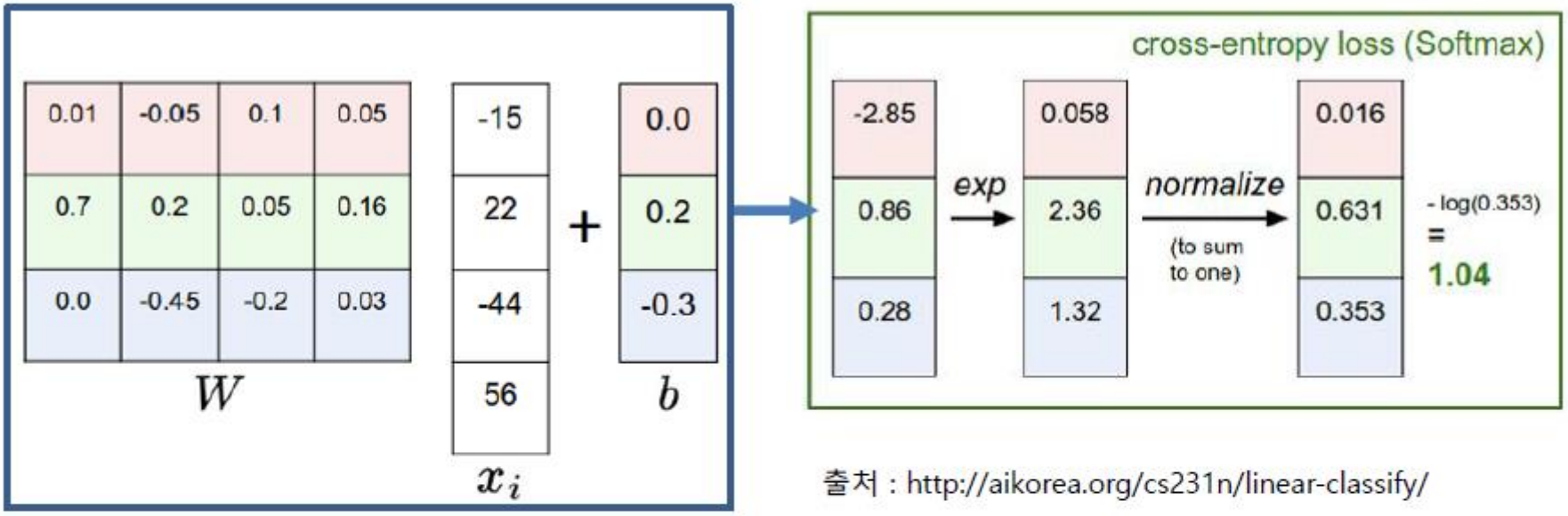
$$E = -\sum_k t_k \log y_k$$

(y: predict, t: target, k: dimension, log: natural log)

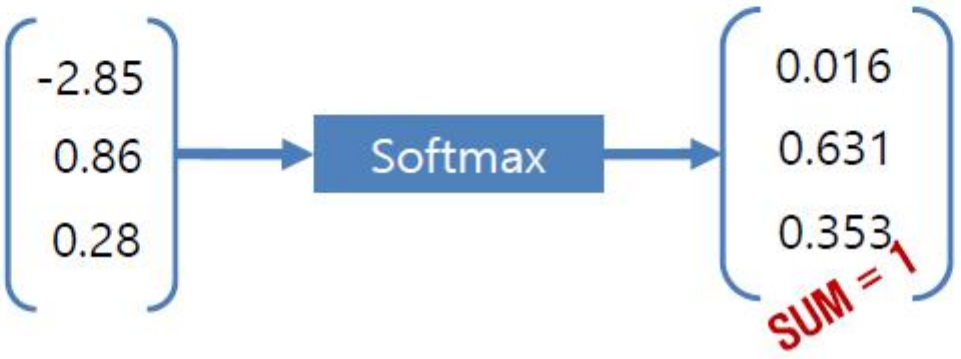
- **소프트맥스(softmax) 함수를 사용**

$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

Cross entropy – Softmax

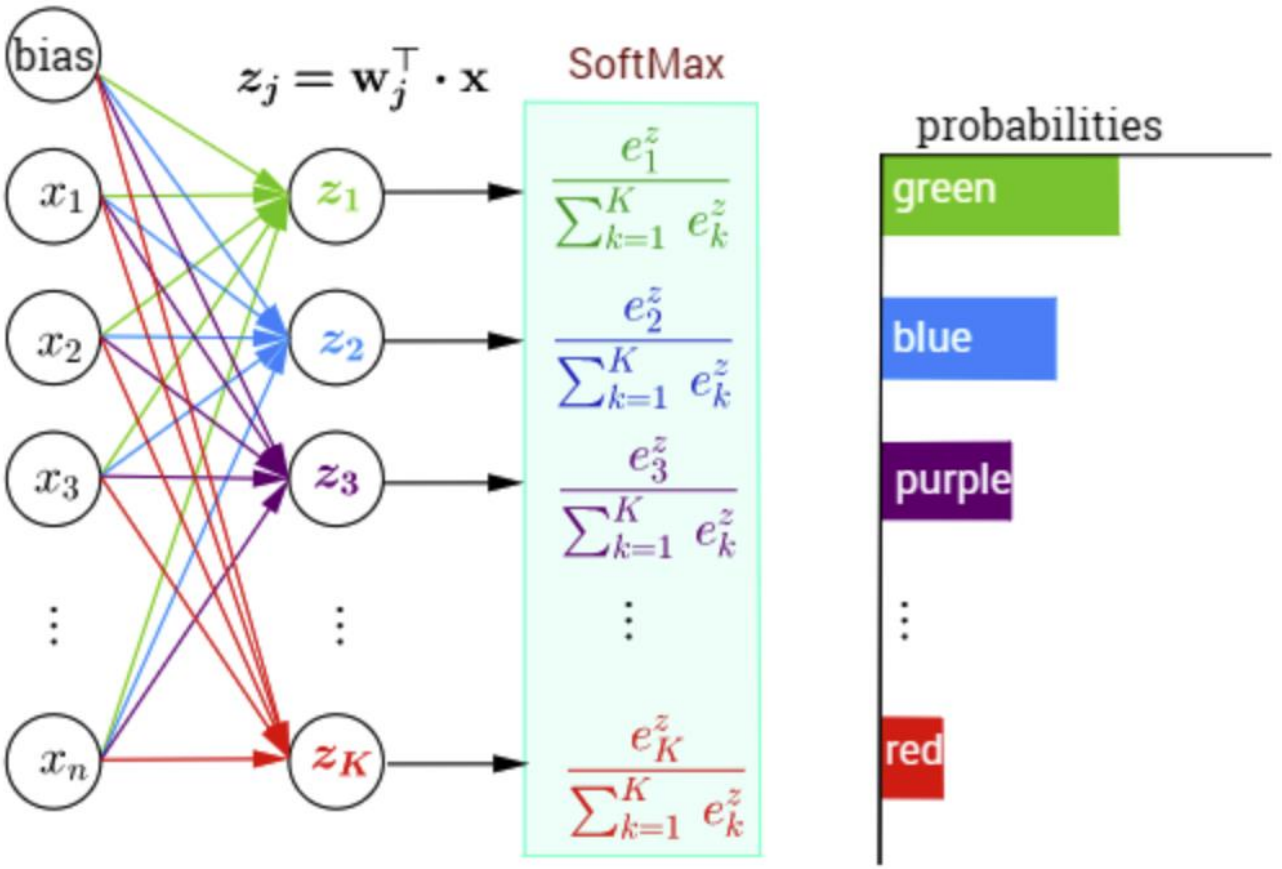


출처 : http://aikorea.org/cs231n/linear-classify/



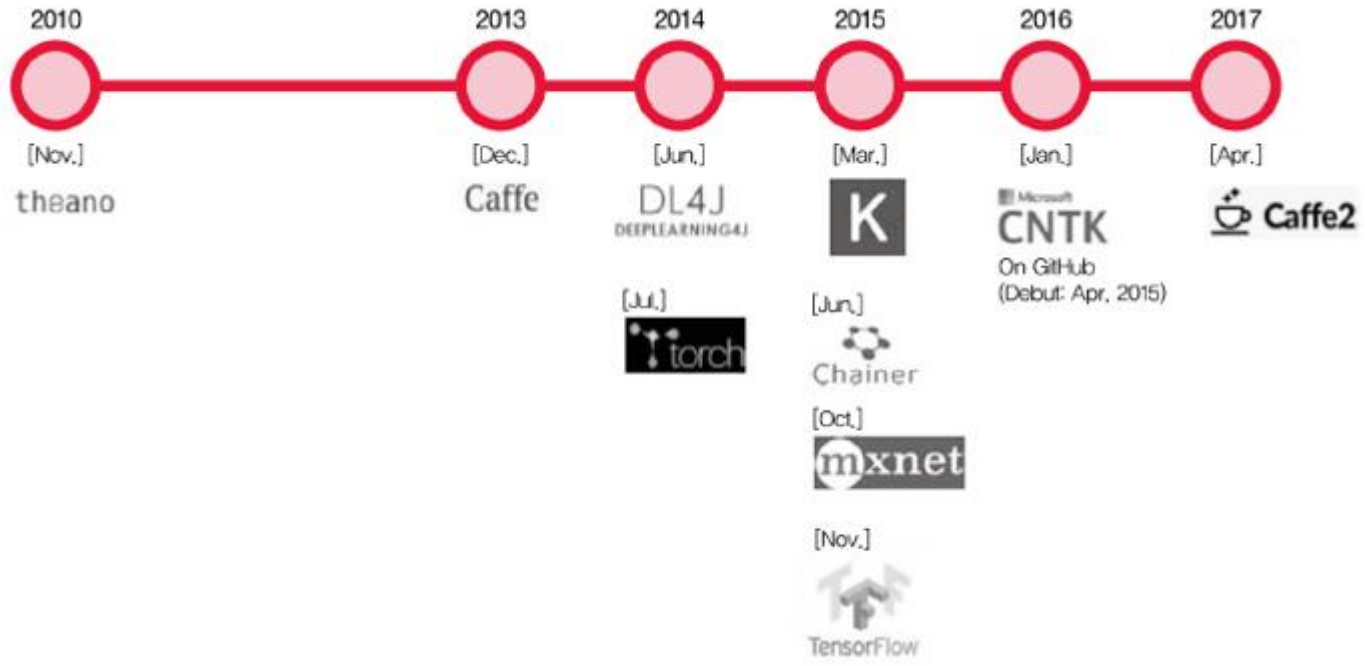
Cross entropy – Softmax

- 상대적인 점수 비교 : 확률처럼 0~1 사이 값으로 매핑



텐서플로우

딥러닝 소프트웨어


2010
2013
2014
2015
2016
2017
[Nov.]
theano
[Dec.]
Caffe
[Jun.]
DL4J
DEEPLEARNING4J
[Jul.]
torch
[Mar.]
K
[Jun.]
Chainer
[Oct.]
mxnet
[Nov.]
TensorFlow
[Jan.]
CNTK
On GitHub
(Debut: Apr, 2015)
[Apr.]
Caffe2

텐서플로우 (Tensorflow)

- 데이터 처리 및 머신러닝에 특화된 소프트웨어로, 신경망 구축을 지원하는 라이브러리
  - 2015년 구글에서 공개

- Python의 머신러닝 라이브러리인 sklearn와 호환되는 tf.learn 라이브러리를 제공

- 시각화 도구로 텐서보드(TensorBoard)를 제공

- 텐서플로우의 가장 큰 특징
  - 다수의 GPU 칩을 병렬로 사용하거나 여러 대의 컴퓨터에서 분산 실행하여 대용량의 신경망 알고리즘을 쉽게 구축할 수 있다

텐서플로우 초보자를 위한 가이드
- https://www.tensorflow.org/tutorials/

계산 그래프 (Computational Graph)

- 텐서플로우의 동작을 그래프로 나타낸 것으로 노드들을 링크로 연결한 구조를 갖는다.
  - 노드 : 어떤 작업이 실행
  - 링크를 따라서 텐서 (즉 데이터)가 전송
  - 노드에는 상수를 담을 수 있다

```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0)                    # also tf.float32 implicitly
print(node1, node2)
```

세션 (Session)

- 계산 그래프를 실행하려면 먼저 세션을 하나 만들어야 한다.

- 세션
  - 하나의 프로그램 실행 환경을 담는 것
  - 텐서플로우 내부는 C언어로 구현되어 있는데 이 실행 프로그램과의 연결 창구

- 아래는 세션 객체 sess를 만들고 run() 프로그램을 실행하면서 내용으로 실행 그래프(computational graph)의 일부인 node1, node2를 보여주라는 명령

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

```
> [3.0, 4.0]
```

세션 (Session)

- 아래는 위의 두 상수 노드의 값을 더하는 작업을 하는 새로운 노드 nose3을 만들었다.

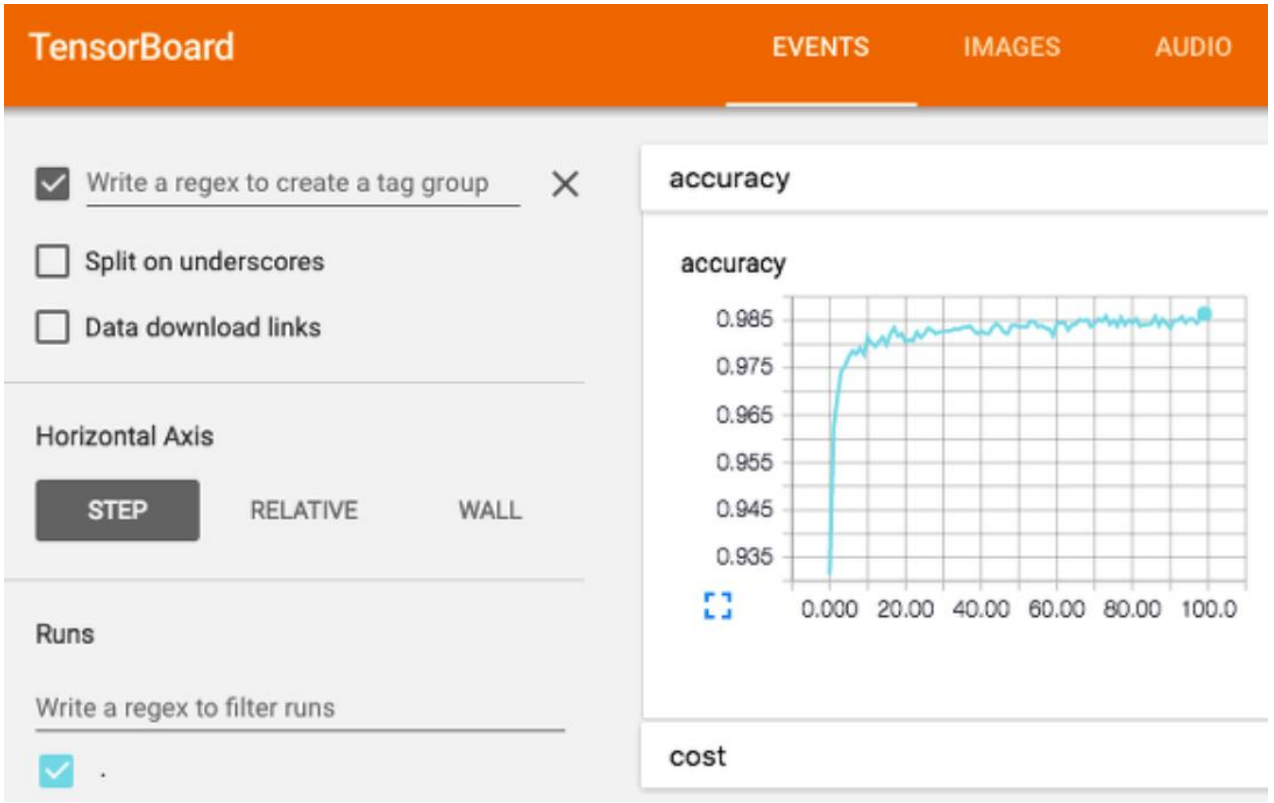
```
node3 = tf.add(node1, node2)
print("node3: ", node3)
print("sess.run(node3): ", sess.run(node3))
```

> node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0

텐서 보드 (tensorBoard)

- 계산 그래프의 내용을 그래픽하게 보여주는 기능을 제공

Tensorboard tutorials



텐서플로우 모듈, 클래스, 함수

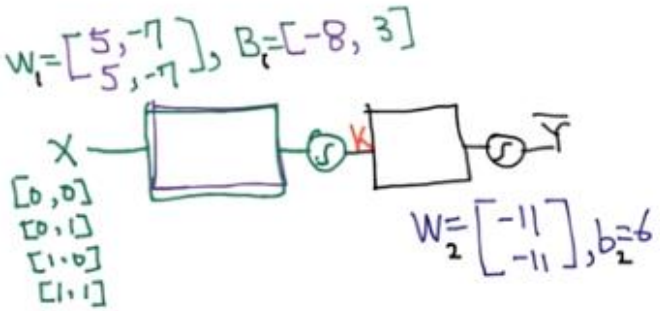
텐서플로우 모듈, 클래스, 함수

텐서플로우 텐서 연산

텐서플로우 2.0 변환

Tensorflow 표현 (2-stage network)

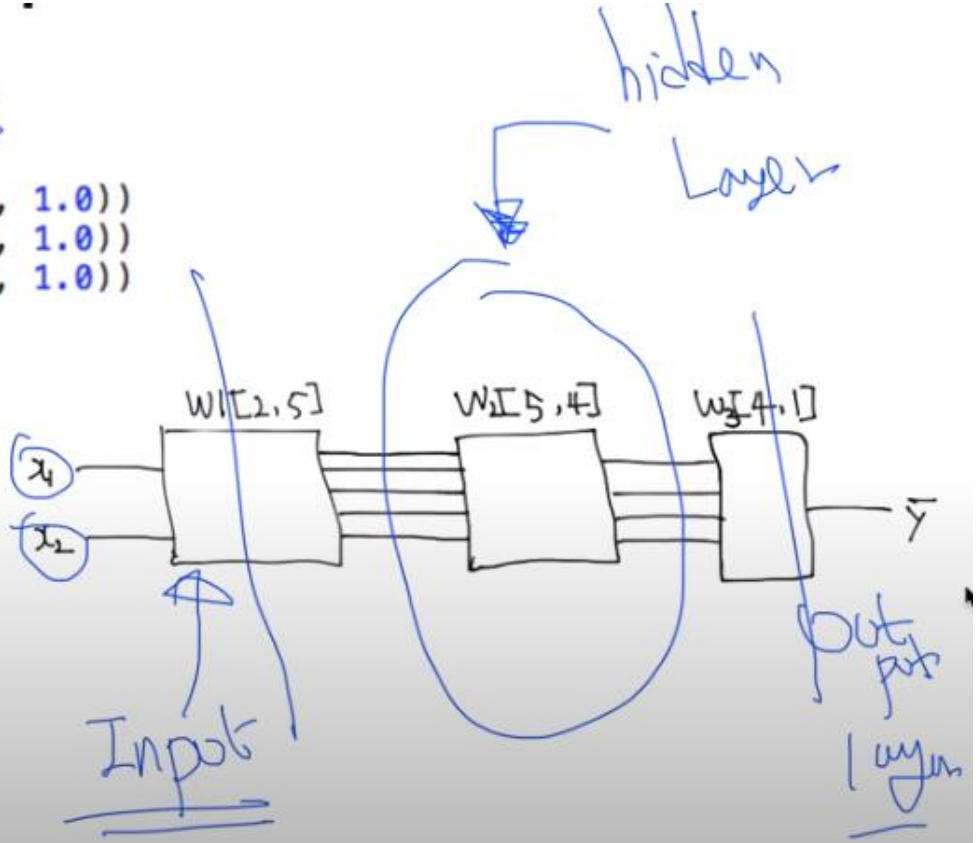
NN for XOR




```
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

Tensorflow 표현 (3-stage network)

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```



고급 API (Keras)

고급 API의 필요성

- 심층 신경망(Deep neural networks)이 유행이기는 하나 텐서플로우 등 신경망의 주요 프레임워크들이 원시적인 코딩이 가능하고 세세하게 기능을 구현할 수 있는 라이브러리이지만 사용의 복잡성으로 인해 초보자가 사용법이 다소 복잡

- 초보자도 쉽게 신경망을 구현할 수 있는 고급 API 형태의 라이브러리의 제공이 필요함

https://keras.io/applications/

TensorFlow APIs

TensorFlow provides multiple APIs (Application Programming Interfaces).

These can be classified into 2 major categories:

1. Low level API:
    - complete programming control
    - recommended for machine learning researchers
    - provides fine levels of control over the models
    - **TensorFlow Core** is the low level API of TensorFlow.
2. High level API:
    - built on top of **TensorFlow Core**
    - easier to learn and use than **TensorFlow Core**
    - make repetitive tasks easier and more consistent between different users
    - **tf.contrib.learn** is an example of a high level API.

텐서플로우 Core API : 저 수준 API

- https://www.tensorflow.org/guide/eager
- 텐서플로우 계산 그래프
- 텐서
- 연산
- 세션
- 텐서플로우를 처음 접하는 사람에게는 이해하기가 어려운 부분이 있다
- 사용할 경우 얻는 이점도 있으므로(대부분 디버깅 관련) 고수준과 저수준 텐서플로우 API를 필요에 따라 섞어 사용하면 된다.

텐서플로우 Core API : 저 수준 API

- 신경층(neural layer)
- 비용 함수(cost function)
- 옵티마이저(optimizer)
- 초기화 방식(initialization scheme)
- 활성화 함수(activation function)
- 정규화 방식(regularization scheme)

- 모두 독립적인 모듈이며 결합을 통해 새로운 모델을 만들 수 있다.
- 새로운 모듈을 새 클래스와 함수로 간단히 추가할 수 있다.
- 모델은 별도의 모델 구성 파일이 아닌 파이썬 코드로 정의된다.

텐서플로우 사용 딥러닝 모델 개발 단계

1. 데이터 준비 : 수집 및 전처리
2. 모델 구성
   - 입력층, 은닉층, 출력층 등 계층 구성 포함
3. 손실 함수 정의
4. 최적화 알고리즘 선택
5. 모델 훈련
6. 모델 평가 및 예측
7. 모델 저장 및 재사용
   - 모델 구조 & 가중치 저장 및 로드

케라스 (Keras)

- 딥러닝 모델을 python으로 쉽게 구축해주는 패키지 (고급 API)
  - Python으로 작성
- 다양한 Backend 신경망 엔진 지원
  - Tensorflow, Theano, CNTK(Microsoft), MXNet, Plai의 등과 같은 플랫폼으로 기반으로 동작
- 고수준 Keras API
  - tf.keras
  - https://www.tensorflow.org/guide/keras
- 케라스 설치 (colab – 미리 설치되어 있음)
  - conda, pip 명령으로 설치
  - Pip3 install keras

https://keras.io/applications/

케라스 (Keras)

- 텐서 곱(tensor products), 합성곱(convolutions)과 같은 저수준 작업을 자체적으로 수행하지 않고 백엔드에 의존
- 여러 백엔드 엔진을 지원하지만 주 백엔드이자 기본 백엔드는 텐서플로우
- 케라스의 가장 큰 지지 기업도 구글

케라스 (Keras) – model

- 핵심 데이터 구조
- 두 가지 주 모델 유형
  - Sequential Model: linear stack of layers
  - Functional API: way to define complex models, such as multi-output models, directed acyclic models, or shared layers

케라스 (Keras) – Sequential model

- 케라스 시퀀셜(Sequential) 모델
  - 계층의 선형적인 스택
    - 입력과 출력이 각각 하나
  - 계층은 아주 단순하게 기술이 가능 : *model.add()* 사용
  - 각 계층 정의에 한 줄의 코드가 필요
  - 컴파일(학습 프로세스 정의)에 한 줄의 코드가 필요
  - 피팅(학습), 평가(손실 및 메트릭 계산), 학습된 모델에서의 예측 출력에서 각각 한 줄의 코드를 사용

```
1 model = Sequential([
2             Dense(32, input_shape=(784,)),
3             Activation('relu'),
4             Dense(10),
5             Activation('softmax'),
6 ])
```

```
10 model = Sequential()
11 model.add(Dense(32, input_dim=784))
12 model.add(Activation('relu'))
```

케라스 (Keras) – Sequential model

- Specifying Input shape
  - *input_shape* (or *input_dim* and *input_length*)
  - *batch_size*
- Compilation
  - Optimizer
  - Loss function
  - List of metrics
- Training
  - Train on Numpy arrays of features and label

```
model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

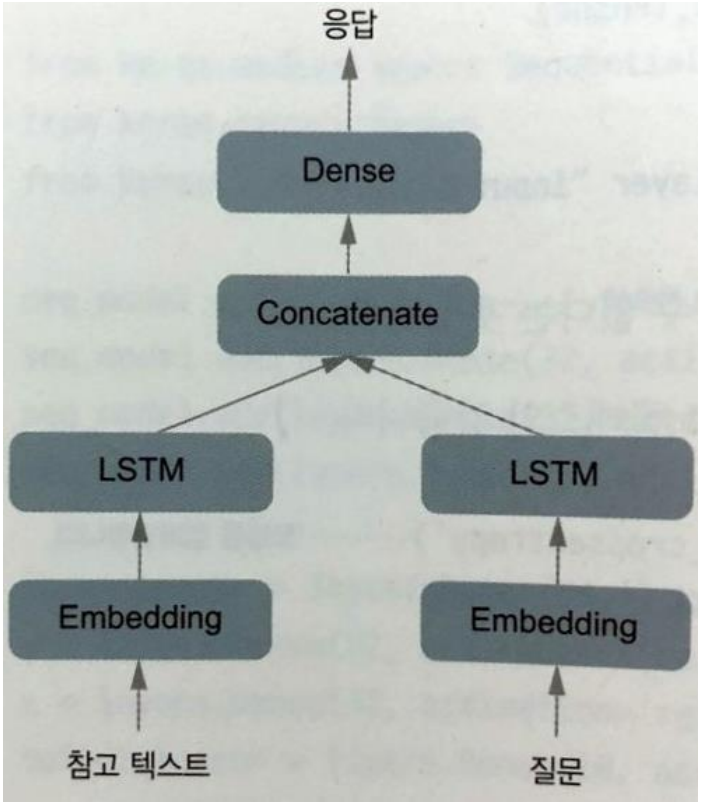
케라스 (Keras) – 함수 API model

- 케라스 함수 API
  - 케라스 시퀀셜 모델은 간소하지만 모델 토폴로지는 제한적
  - 다중 입력/다중 출력 모델
  - 방향성 비순환 그래프(DAG)
  - 공유된 계층이 있는 모델과 같은 복잡한 모델을 만드는 데 유용
  - 시퀀셜 모델과 같은 계층을 사용하지만 조합 측면에서 더 높은 유연성 제공
  - 먼저 계층을 정의한 다음 모델을 생성하고 컴파일하고 피팅(학습)한다.
  - 평가와 예측은 기본적으로 시퀀셜 모델과 동일

```
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

# 입력과 출력 텐서를 지정하여 Model 클래스의 객체를 만듭니다.
model = Model(input_tensor, output_tensor)
```

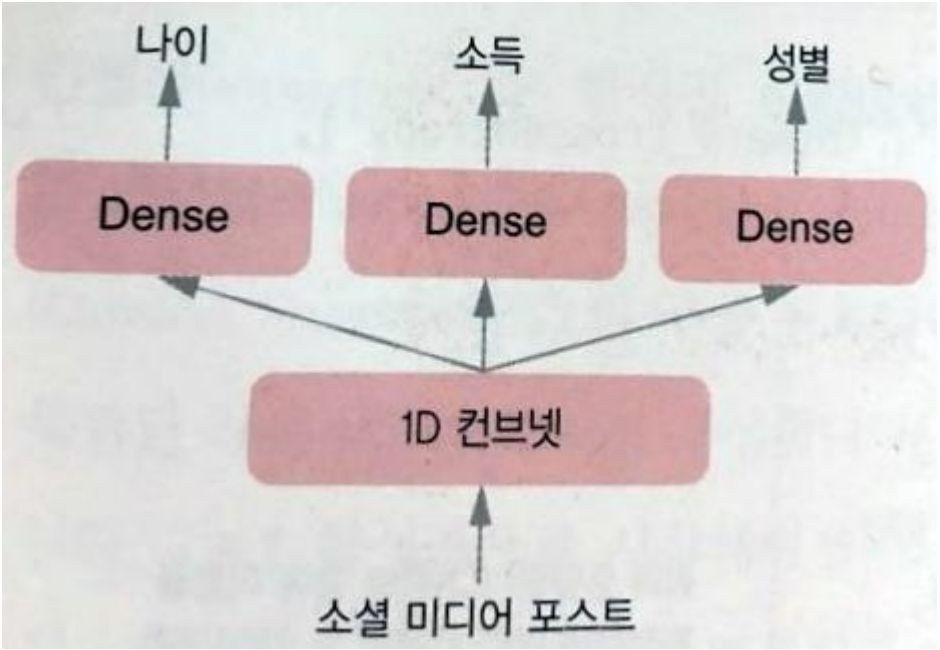
케라스 (Keras) – 함수 API model

- 다중 입력 모델
  - ➢ 질문-응답(Question-Answering) 모델



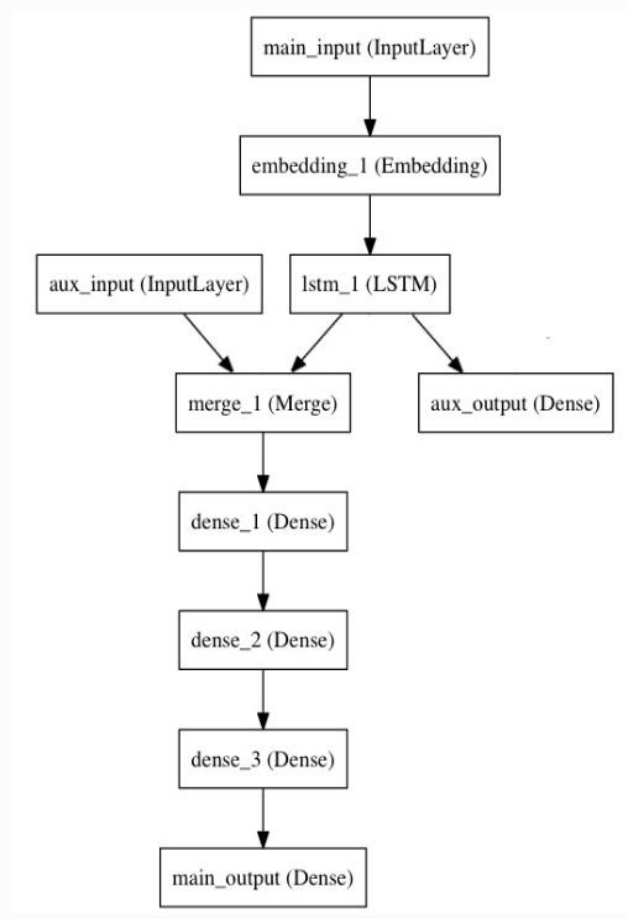
케라스 (Keras) – 함수 API model

- 다중 출력 모델
  - ➢ 익명 사용자의 포스트로부터 그 사람의 나이, 성별, 소득 수준 예측



케라스 (Keras) – 함수 API model

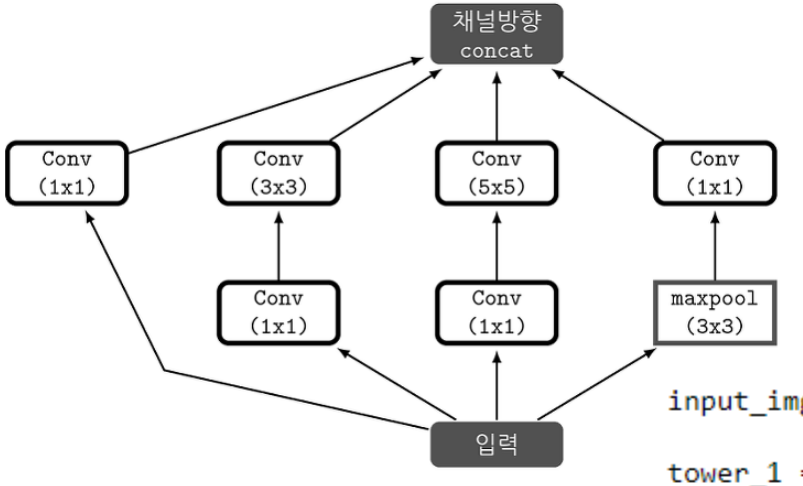
- multi-input multi-output models example (다중 입력 다중 출력 모델)




```
# Headline input: meant to receive sequences of 100 integers, between 1 and 10000
# Note that we can name any layer by passing it a "name" argument.
main_input = Input(shape=(100,), dtype='int32', name='main_input')

# This embedding layer will encode the input sequence
# into a sequence of dense 512-dimensional vectors.
x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)

# A LSTM will transform the vector sequence into a single vector,
# containing information about the entire sequence
lstm_out = LSTM(32)(x)

auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)

auxiliary_input = Input(shape=(5,), name='aux_input')
x = keras.layers.concatenate([lstm_out, auxiliary_input])

# We stack a deep densely-connected network on top
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)

# And finally we add the main logistic regression layer
main_output = Dense(1, activation='sigmoid', name='main_output')(x)

model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output, auxiliary_output])
```

케라스 (Keras) – 함수 API model

- 내부 토폴로지가 복잡한 모델
  - Inception 모듈 (GoogleNet)




```
input_img = Input(shape=(256, 256, 3))

tower_1 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)
tower_1 = Conv2D(64, (3, 3), padding='same', activation='relu')(tower_1)

tower_2 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)
tower_2 = Conv2D(64, (5, 5), padding='same', activation='relu')(tower_2)

tower_3 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(input_img)
tower_3 = Conv2D(64, (1, 1), padding='same', activation='relu')(tower_3)

output = keras.layers.concatenate([tower_1, tower_2, tower_3], axis=1)
```

GPU (Graphics Processing Unit)

- 3D 그래픽 연산 전용의 processor
  - 엔비디아(NVIDIA)사 처음 개발, 가장 널리 사용
- 신경망 모델을 학습하려면 GPU를 사용해야 속도가 빠르다
  - driver인 CUDA와 cuDNN(cuda Deep Neural Network)을 설치
  - 텐서플로우도 gpu 버전(Tensorflow-gpu) 을 설치
- GPU를 동시에 사용 – 케라스 명령
  - Data parallelism: replicating the target model once on each device, and each device processes a different fraction of the input data
    - ✓ Refer to keras.utils.multi_gpu_model
  - Device parallelism: running different parts of a same model on different devices
- Colab 사용하면 GPU를 무료로 사용

실습

회귀 분석 예제 (Tensorflow, Keras)

- 고객 1,319명에 대한 신용카드 데이터
- 출력 변수 : share
- 입력 변수 : 신용카드 데이터 중 수치형 데이터로 구성

구분	변수명	변수 설명
출력변수	share	소득에 대한 신용카드 지출의 비율
입력변수	reports	주요 경멸적 보고서의 수
	age	나이(연수 더하기 십이분의 일 년)
	income	연소득(USD 10,000)
	expenditure	월 평균 신용카드 지출
	dependents	부양가족 수
	months	현 주소지에 살고 있는 개월수
	majorcards	보유한 주요 신용카드 수
	active	활성 신용계정 수

분류 예제 (Tensorflow, Keras)

- infert 데이터
- 자연 유산 및 유도 유산 후의 불임 여부를 판정
- 은닉층 2개

열 번호	변수명	설명
1	Education 교육받은 연수	0 = 0~5 years 1 = 6~11 years 2 = 12+ years
2	age	age in years of case
3	parity	count
4	number of prior induced abortions 유도유산(낙태) 횟수	0 = 0 1 = 1 2 = 2 or more
5	case status	1 = case 0 = control
6	number of prior spontaneous abortions 자연유산 횟수	0 = 0 1 = 1 2 = 2 or more
7	matched set number	1~83
8	stratum number	1~63

Q & A

데이터 download

- goo.gl 을 주로 이용
- zip 파일을 풀 때
  - 한글파일명이 깨지지 않게 하려면 알집의 “보기-언어변환” 옵션을 유니코드(MAC) 등으로 조절해야 된다 (맥에서 만든 파일의 경우)
- 작업 폴더 아래에 data 폴더를 만들고 여기에 데이터를 저장
- 압축된 파일은 알집 등으로 푼다
- train.csv 등 이름이 겹치면 새로운 이름으로 변경한다 -> image-train.csv 등
- 파일을 찾을 때도 ‘탭’ 키를 사용하면 대상 파일명을 보여준다

터미널 주요 명령어 (windows)

- dir – 파일들을 보여준다 (맥에서는 ls)
- pwd – 현재 폴더 명 (present working directory)
- cd – 폴더 이동 (change directory)
  - cd .. – 위 폴더로 이동
- cp – 복사 (copy)
- 탭 키를 누르면 자동완성을 한다 (폴더명, 파일명, 함수명 등)