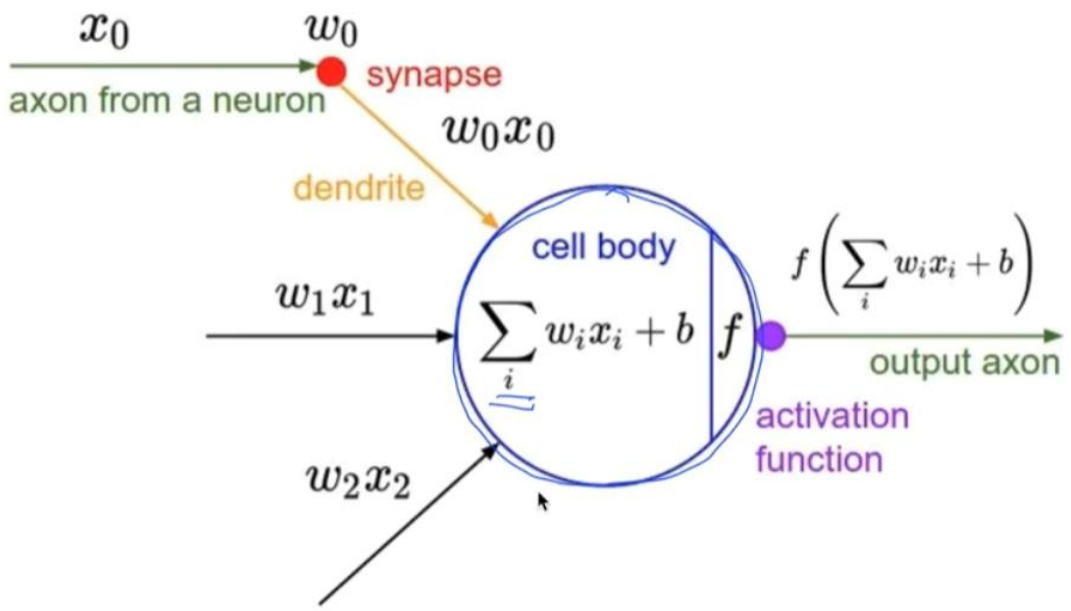
3주차 ML/DL 스터디 발표

GDSC Hanyang

ML/DL core
김남호

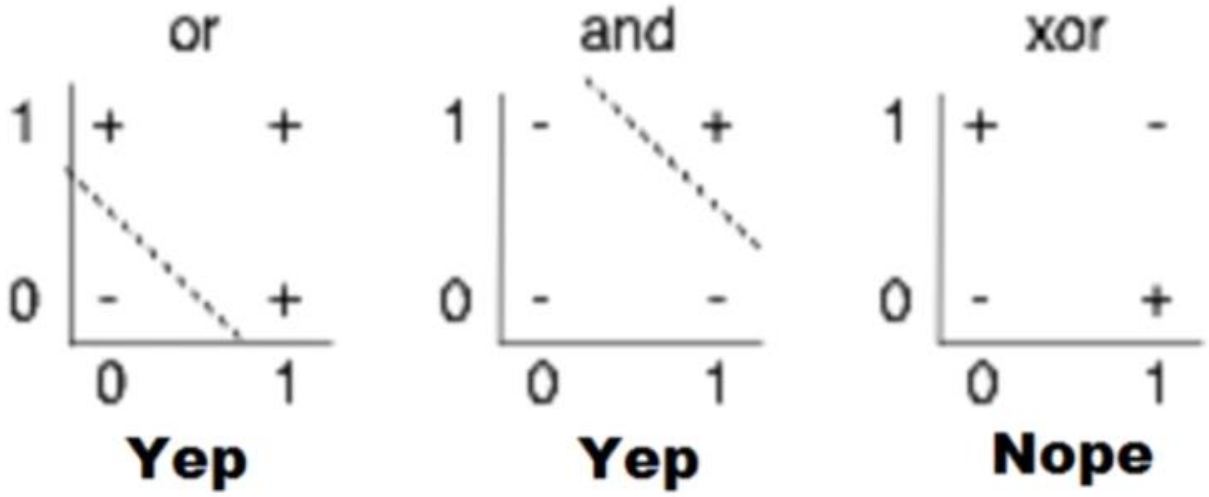
Machine learning의 역사

- Thinking machine을 만들자



Machine learning의 역사

- 한계 봉착
  - Linear 하게 구분짓기 어려움(XOR)

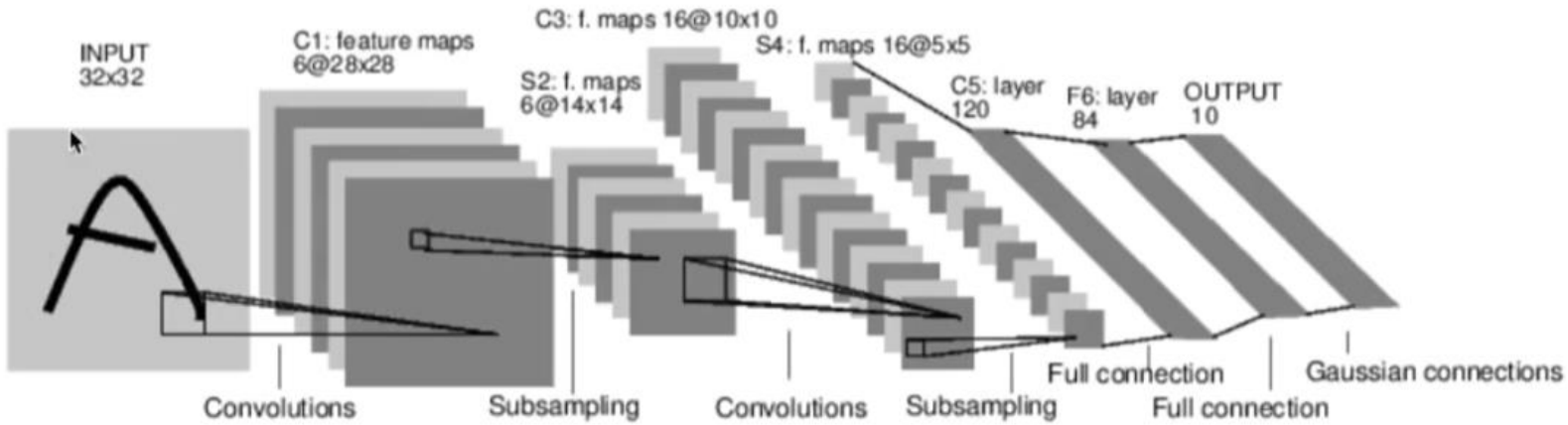


Backpropagation

- 해결책으로 등장
  - 예측값과 결과의 오차 -> 뒤에서 앞으로 반영하는 형식

  - 한계: too many layer에는 앞으로 갈수록, 그 오차의 영향력이 작아지며 반영됨

Convolutional Neural Networks



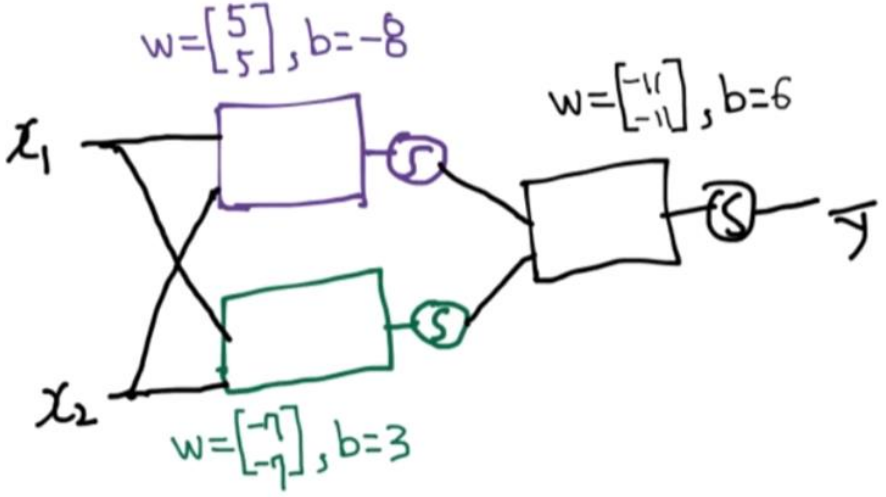
Deep learning의 등장

- NN -> Deep learning(명칭 개선)
- 초기값 잘 주는 것이 중요.
- Deep API learning -> 컴퓨팅에 실용적
- Ex) 추천 알고리즘 생성

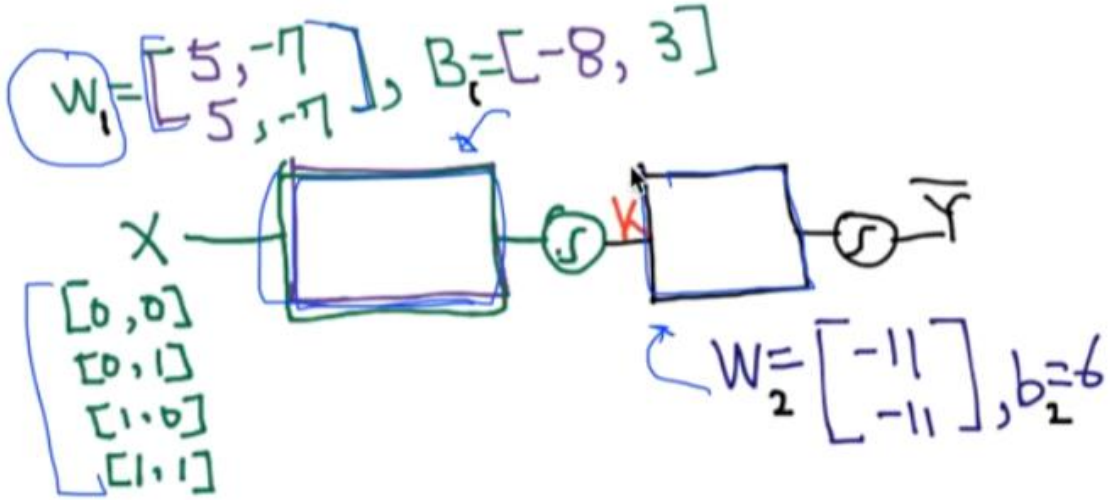
XOR 게이트 문제 해결

- Forward Propagation
- 3개의 게이트 + 시그모이드 함수 적용
- =>Neural network

- 적용되는 W,b값은 어떻게 정하지?



NN(Forward Propagation)



중요한 concept =>

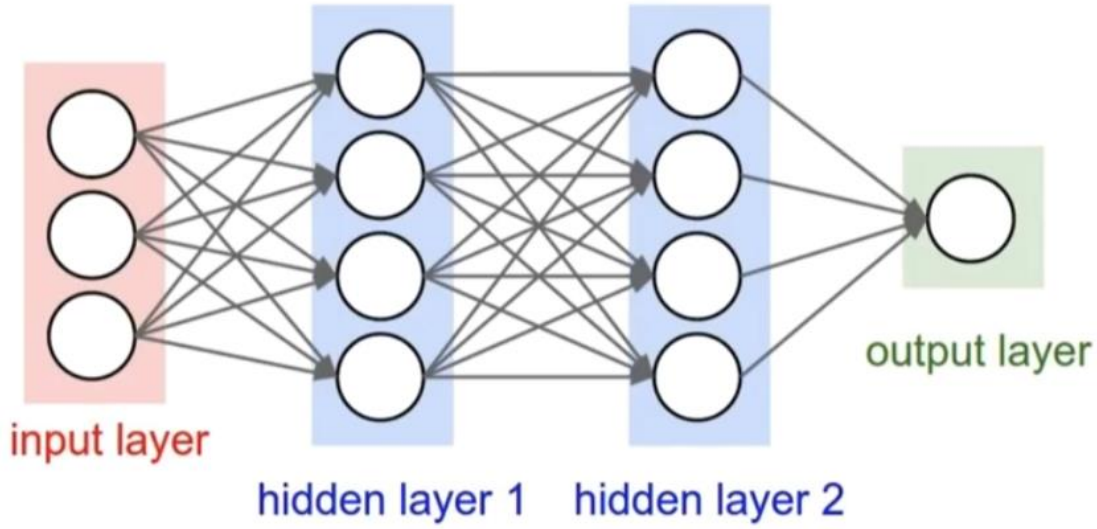
$$K(x) = \text{sigmoid}(XW_1 + B_1)$$

$$\hat{Y} = H(X) = \text{sigmoid}(K(x)W_2 + b_2)$$

어떻게 NN을 학습?

- Cost 함수 만들고, W,b에 대해 경사 하강법 적용 하여 최소의 cost를 찾는 것. (미분)

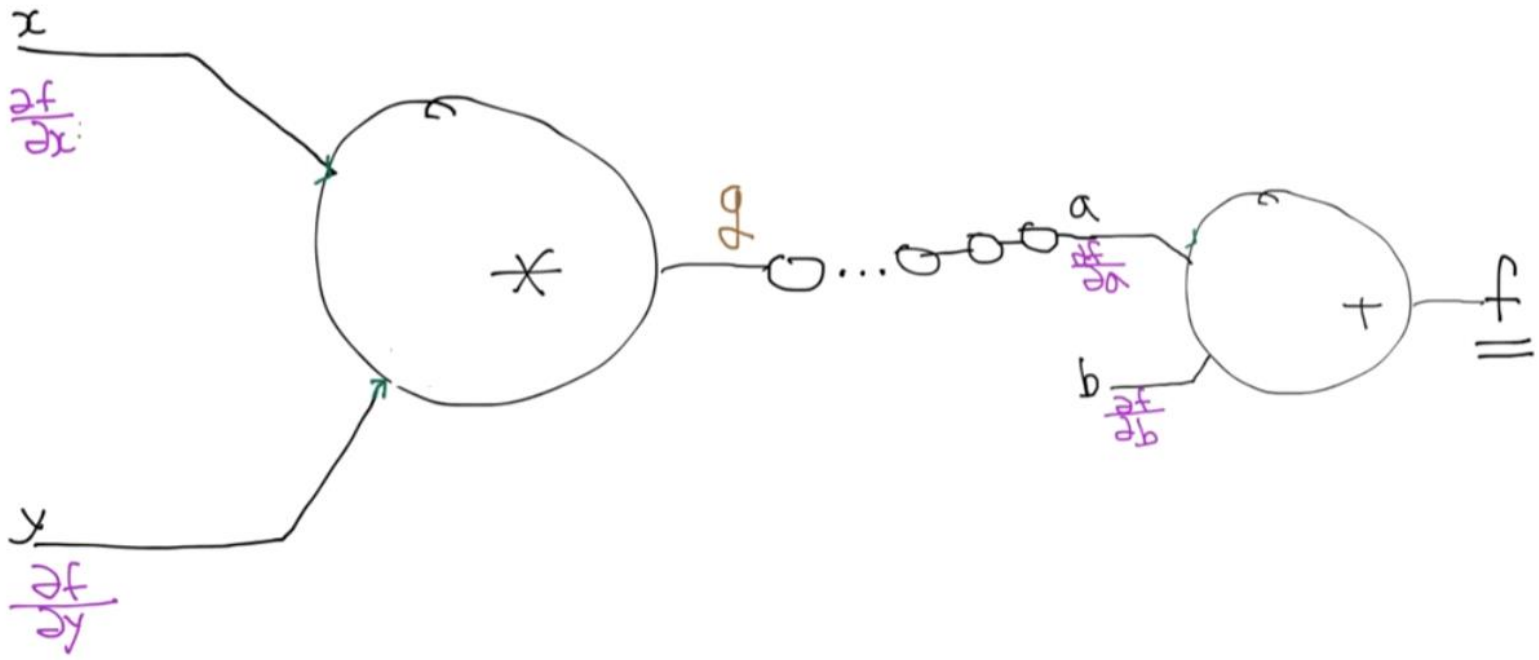
너무 많고 복잡함 =>
어떻게 해결?
: 이전의 concept



Back propagation

- Chain rule

  뒤에서부터 편미분을 적용하여 나누어서 미분값들을 차근차근 구하는것.

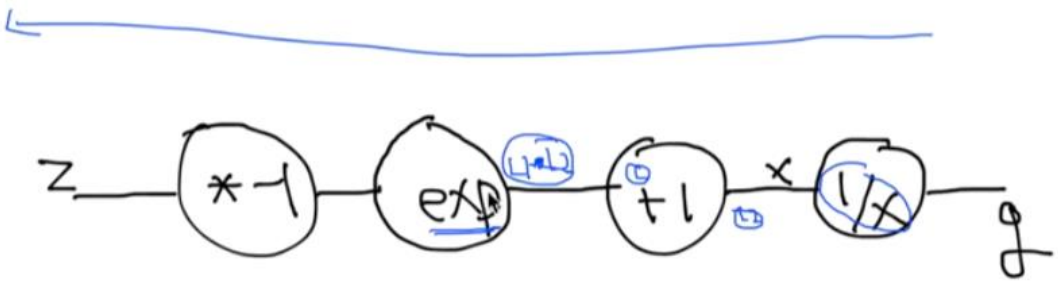


Back propagation

- Chain rule

EX) sigmoid fuction 적용

$$g(z) = \frac{1}{1+e^{-z}}$$

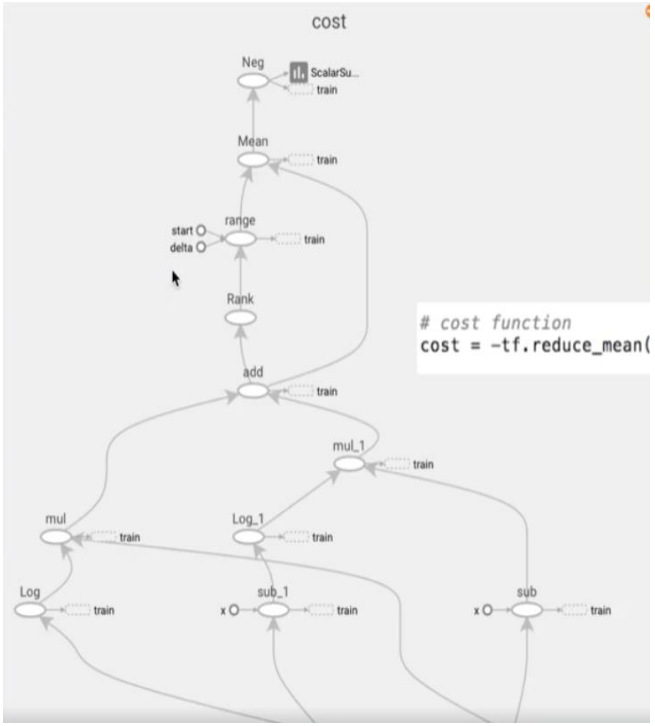


Back propagation

- 그럼 우리가 자동화할 이 미분 프로그램은 어떻게 작동할까?

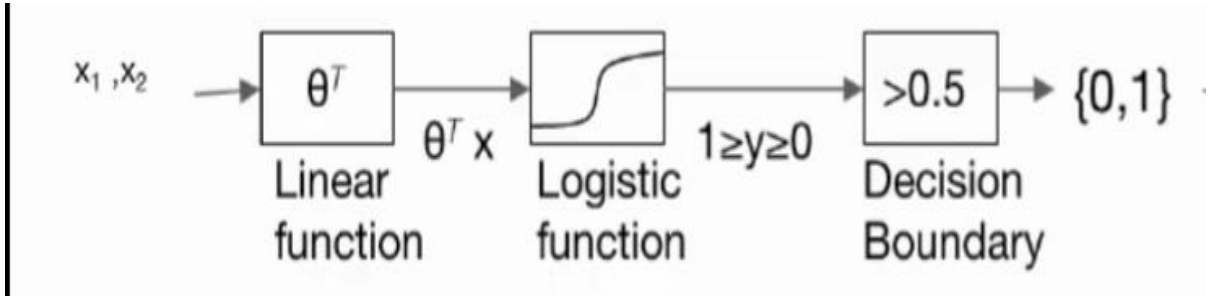
Graph(자료구조)형태로 각 연산들을 저장

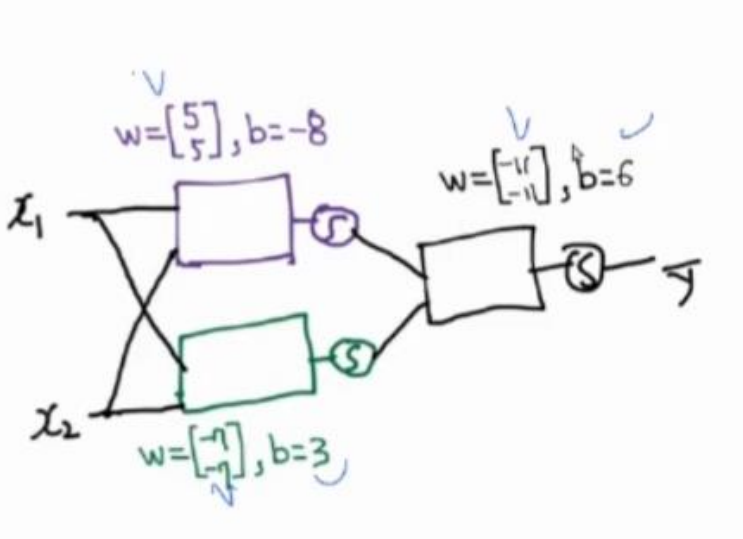
->tensorflow에서 이미 구현됨



NN for XOR

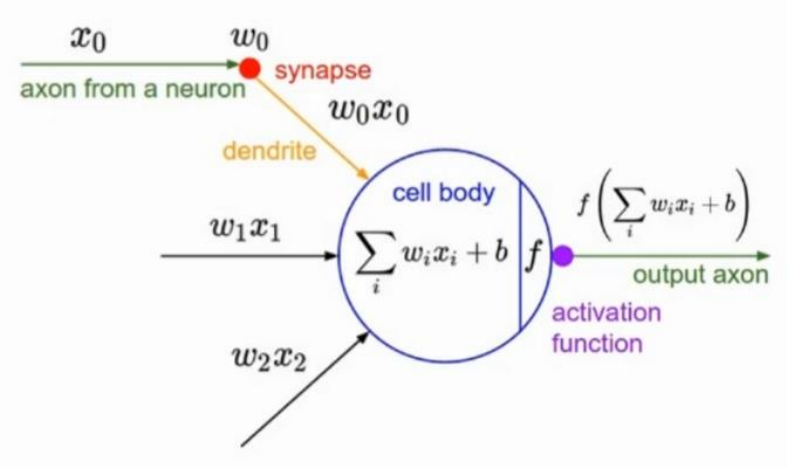
- 각각을 하나의 logistic regression으로 본다.





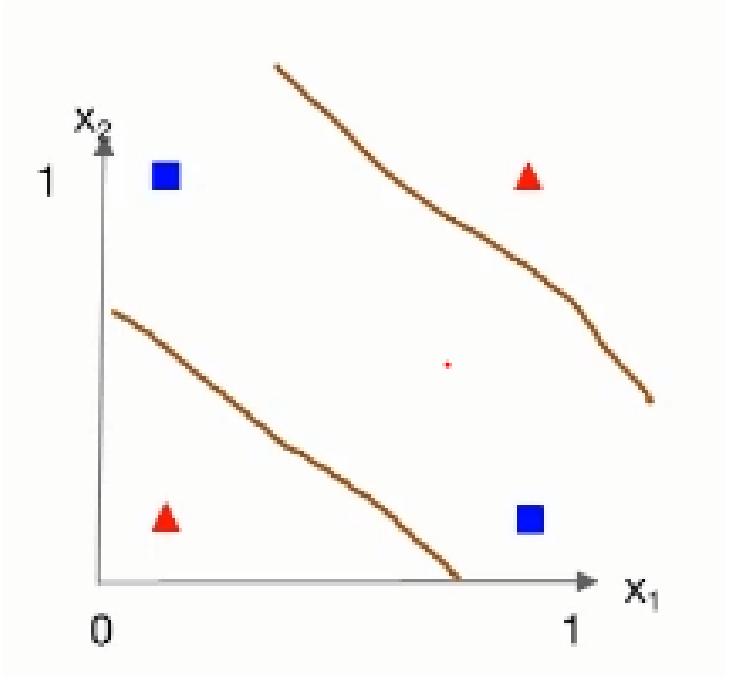
NN for XOR

- 아래 그림이 NN의 기본 구성단위. 다양한 NN모델들의 한 점들도 아래와 같은 구성이다.
- 선형회귀 + activation fuc(like sigmoid)



NN for XOR

- 아래 그림의 두 선이 이전 슬라이드에서 봤던 NN의 2 layer 각각 이라고 봐도 무방할 듯하다.



Tensorboard

- Cost함수, weight, bias 등 시각화도구

- Eager execution이란?

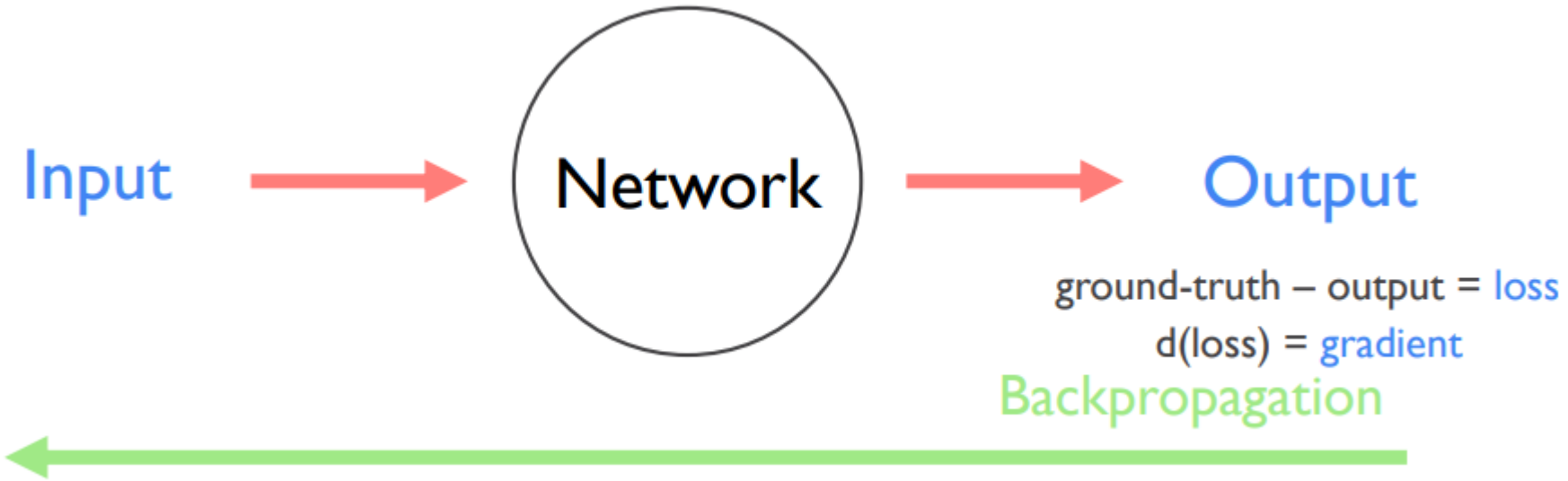
- 텐서 객체와 numpy사이의 연산 호환이 된다
- 세션에 변수 저장이 아니라 변수의 참조가 끝날때까지 변수의 생명주기가 늘어난다
- 모델을 학습없이?(이미 있는 mnist같은거?)바로 사용 가능

Tensorboard

- Eager execution
  - Contrib아래에 summary에 저장된다.(loss, cost)
  - Scala값으로 넣었을 때 그려줌

- Keras
  - Callbacks하단에 텐서보드를 저장
  - 알아서 loss같은 함수를 학습한다.

Problem of Sigmoid



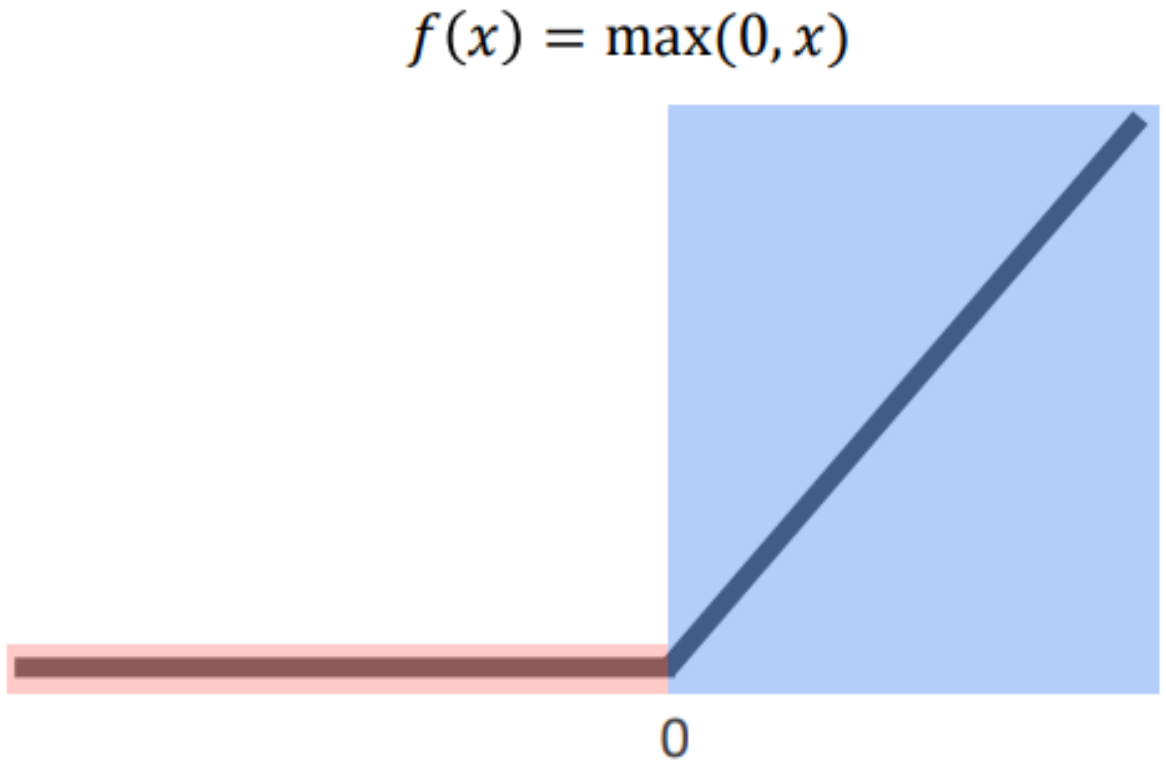
Problem of Sigmoid

... Vanishing Gradient

Relu 함수

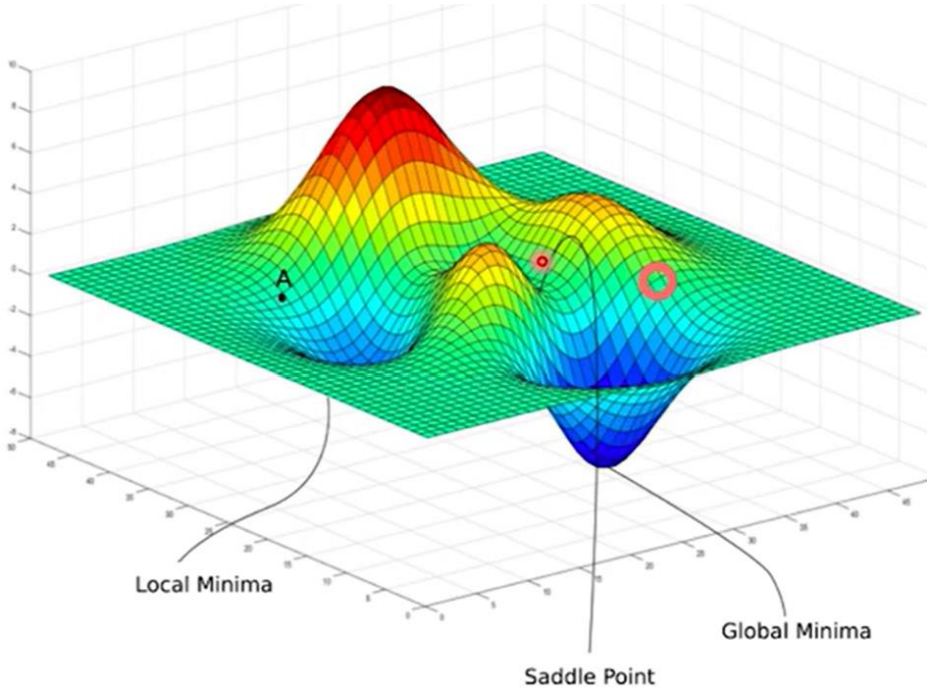
- 문제 발생(x<0에서)
  => leaky relu로 해결

그 外에도 다양한 activation 함수 존재



Weight initialization

- 그림과 같이 global minima를 찾아야한다.
  - How? -> 초기 weight를 잘 설정하자.
  - 좋은 Weight초기화 -> 성능향상

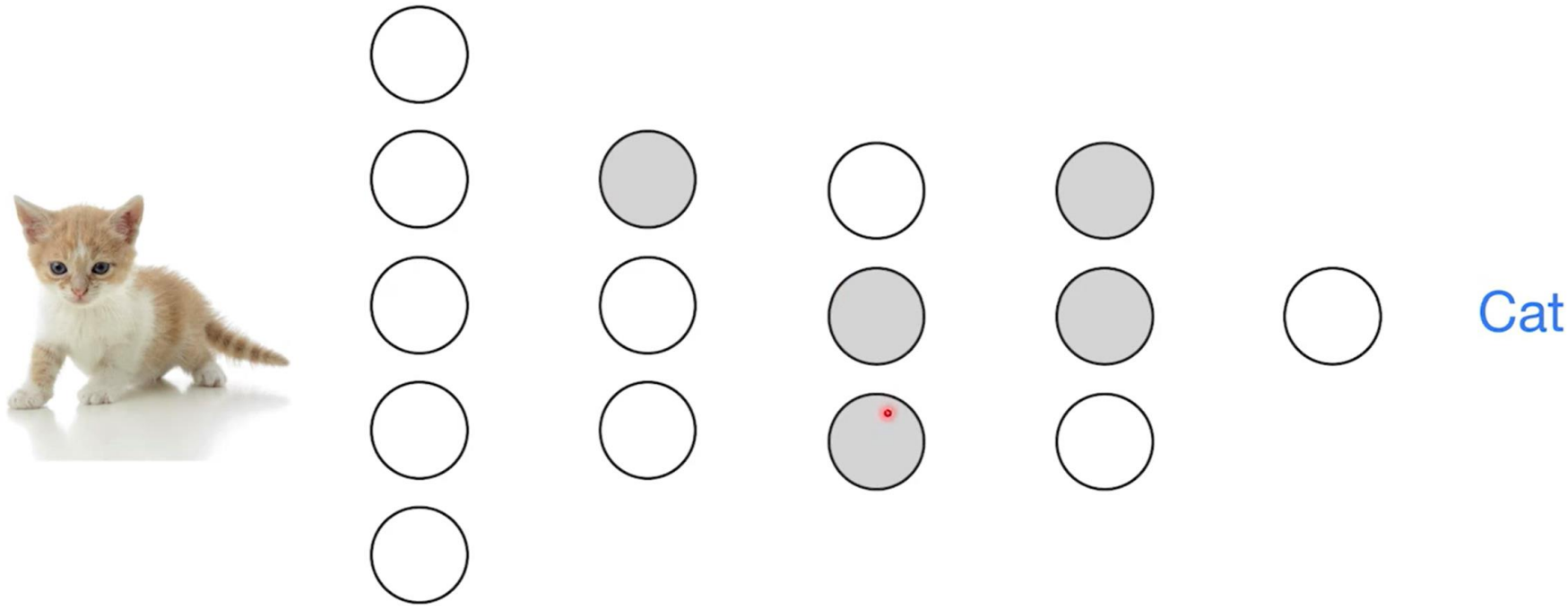


Xavier initialization

- 원래는 random.normalization이용
- 자비어 => 분산 = 2/(input채널+output채널)활용
- He initialization => Relu함수에 특화
  - 분산 => 자비어 2배

Dropout

- Underfitting피하고 overfitting을 방지
- 뉴런을 몇 개(random으로 정함)를 끄고 training data set을 학습

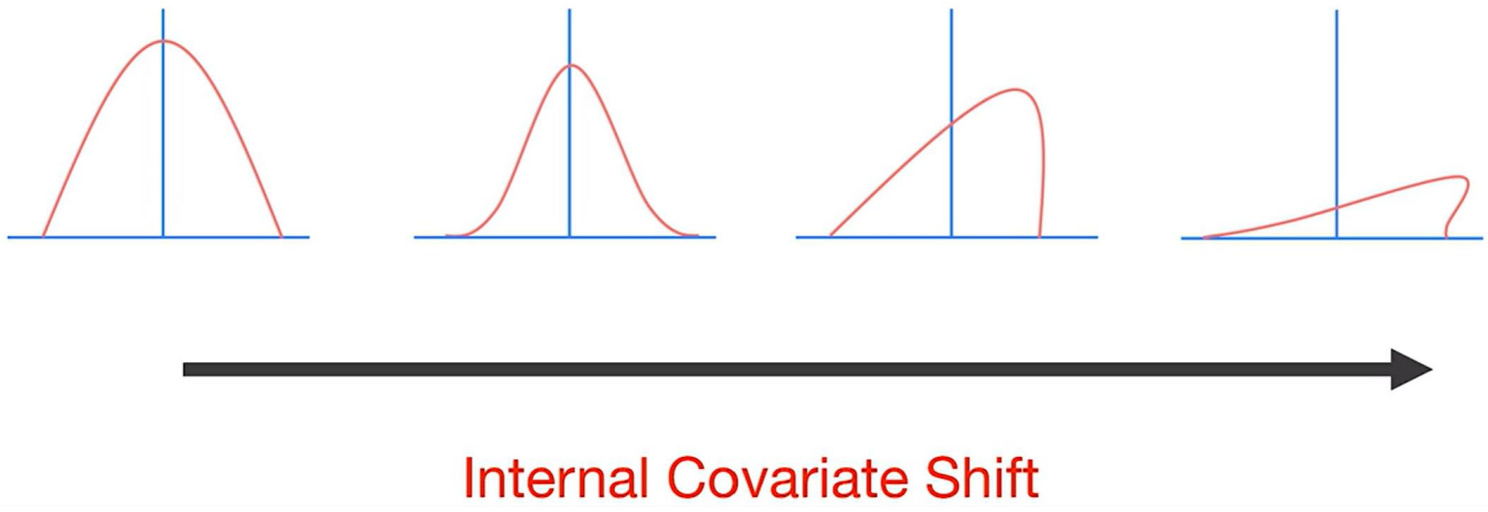


Dropout

- 효과
  - 고양이의 발톱, 꼬리, 눈 만으로 학습을 하므로, 나중에 test에서 검은고양이, 다른 색 고양이가 나와도 과적합되지않고, 고양이인지 판단이 가능하다

Batch Normalization

- 네트워크 layer를 지날때마다, data set의 분포가 이상하게 바뀌는 현상. => 방지해야함.



Batch Normalization

- 분포를 지속적으로 normalization을 시켜서 다음 layer에 넣어 줌.

$$\bar{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \hat{x} = \gamma \bar{x} + \beta$$

Batch Normalization

```
for i in range(2):
    # [N, 784] -> [N, 256] -> [N, 256]
    self.model.add(dense(256, weight_init))
    self.model.add(batch_norm())
    self.model.add(relu())
```

layer
norm
activation

발표를 들어주셔서 감사합니다 ☺

- Forward propagation과 back propagation 의미 이해하고, 구분.
- 직접 코드를 통해 구현(다양한 activation func, normalization…)