week4: Backpropagation and Neural Networks

예습	미완료
복습	미완료
복습과제 날짜	
예습과제 날짜	
내용	

Gradient descent

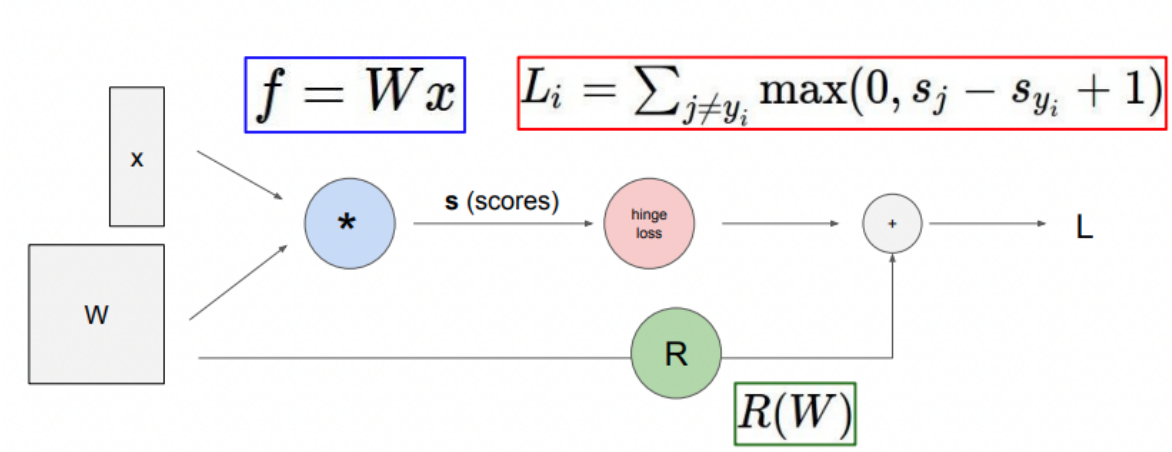
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Neumerical gradient : slow, approximate, easy

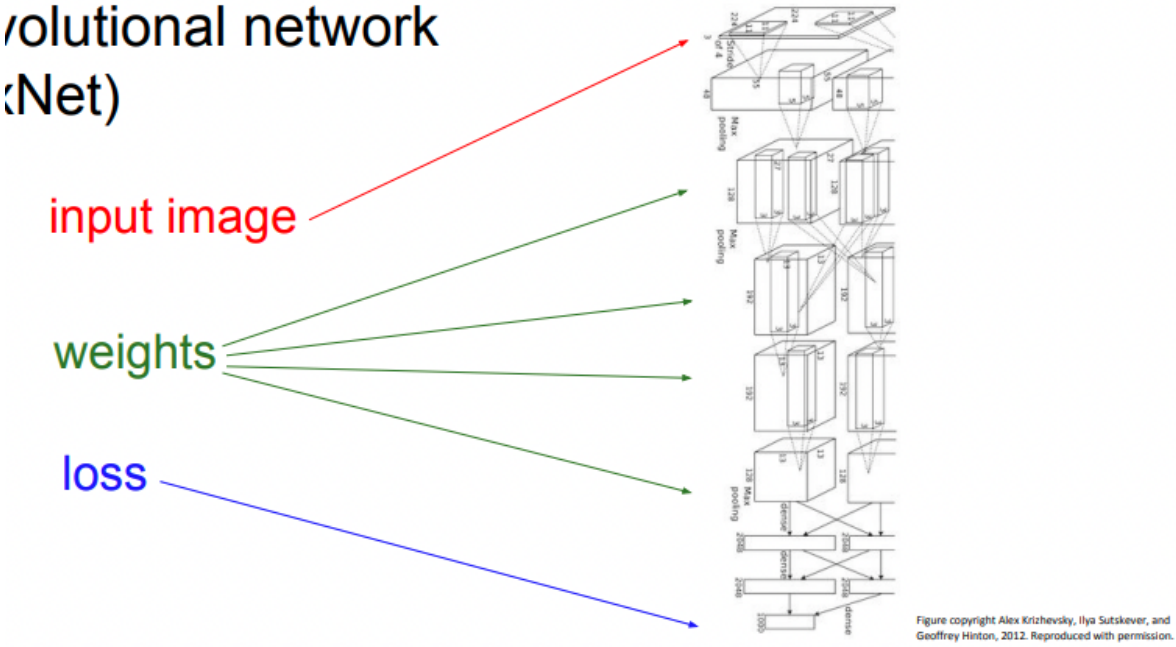
Analytic gradient: fast, exact, error-prone

이 두가지 방법에서 더 많이 사용하는 것은 analytic gradient이다.

Computational Graphs



Convolutional Network(AlexNet)



8개의 레이어로 이루어진 신경망. 각 variable에 대해서 미분을 어떻게 할 것인가. → Backpropagation을 진행하여 빠르게 미분

Backpropagation

간단한 예부터 살펴보자.

Backpropagation: a simple example

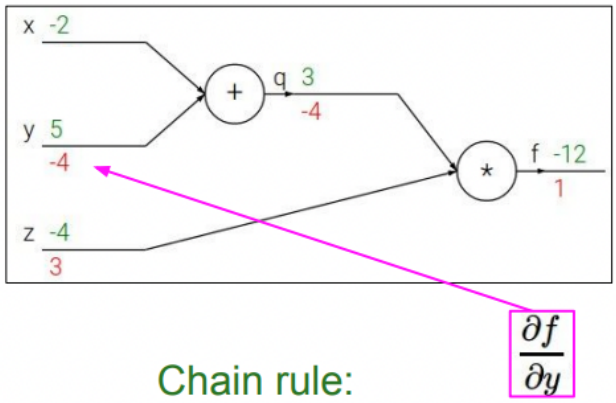
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

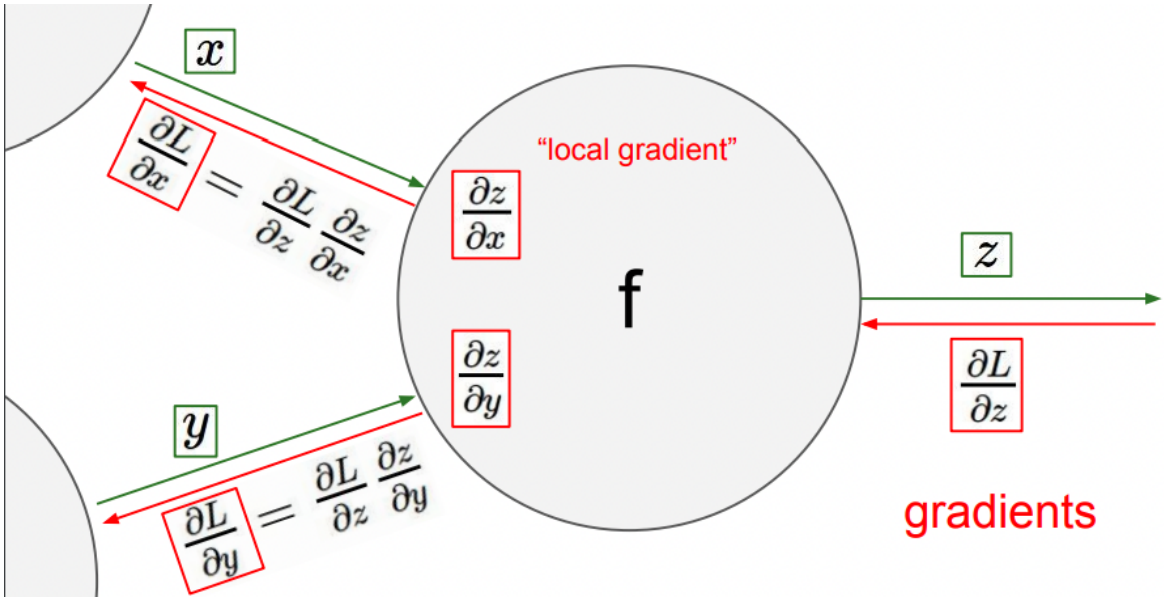
목표는 각 파라미터의 편미분 값을 찾아 gradient descent를 찾는 것이다.

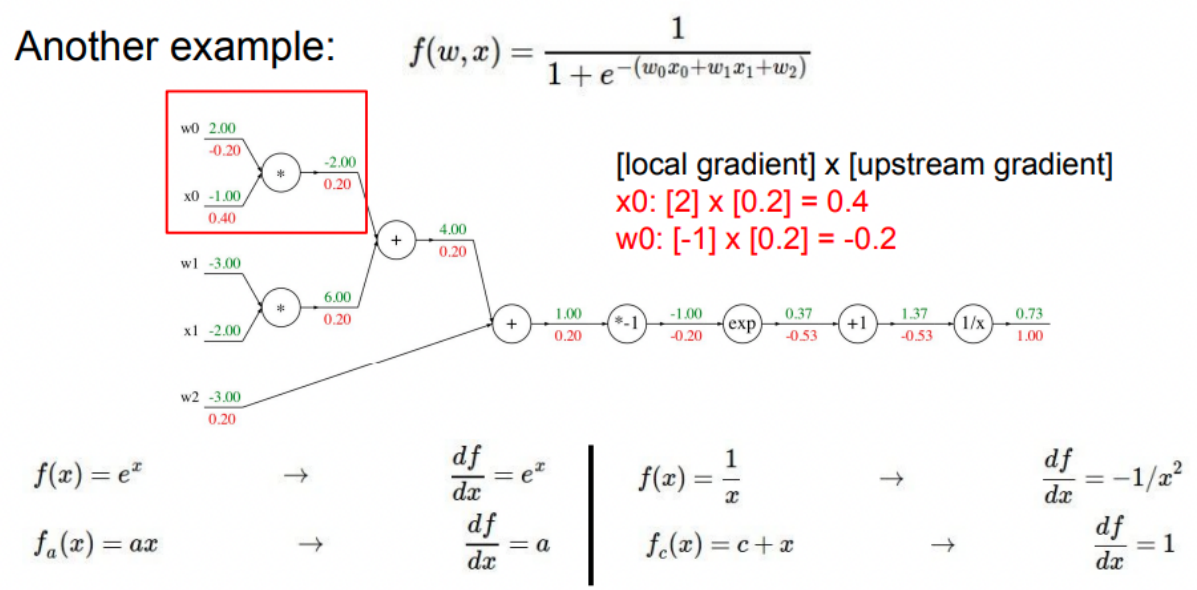
forward pass : 초록색 부분, computational graph에서 차례대로 값을 넣어 함수 값을 구한다.

backward pass(backpropagation): 역방향으로 차례대로 미분해서 gradient 생성

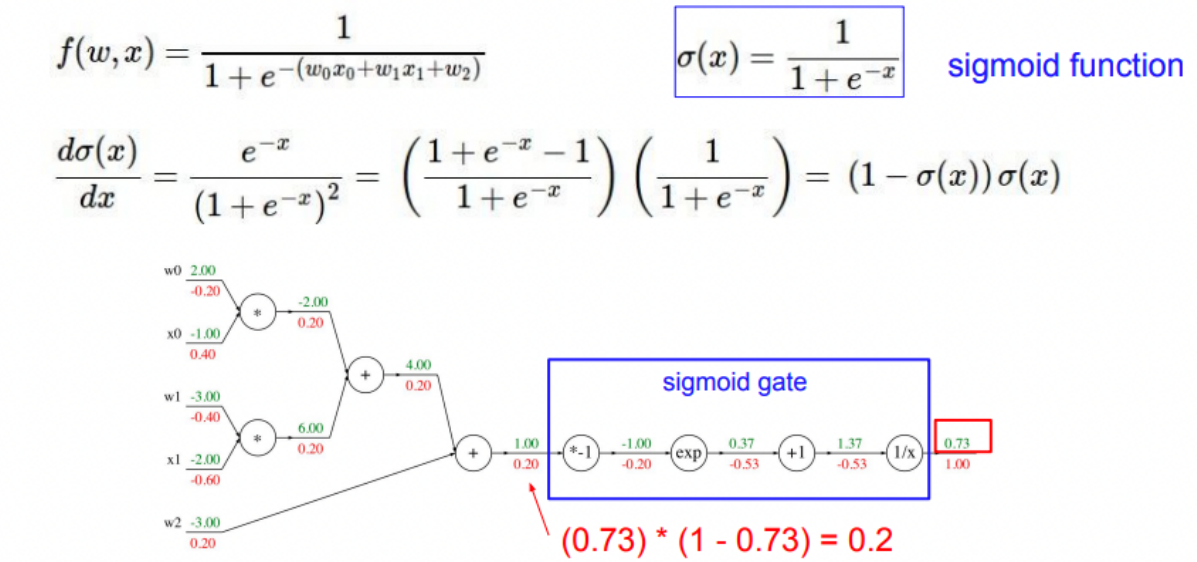
각각의 편미분을 구하는 것은 어렵지 않다. 그러나 우리가 원하는 것은 결국 x에 대한 미분 값인데, f를 y에 대한 미분 값을 한 번에 구하기 위해서는 chain rule을 사용한다.

전체 함수의 f의 미분값은 global gradient라고 부르지만, f가 아닌 중간 노드의 미분 값은 local gradient라고 표현. 즉, 한 노드의 input과 output만 고려해서 구한 gradient를 local gradient라고 한다.





노드 별 local gradient를 구해서 곱한다. 각 gate의 local gradient가 1이라서 gradient가 그대로 나오기 때문에 gradient distributer라고 할 수 있다.



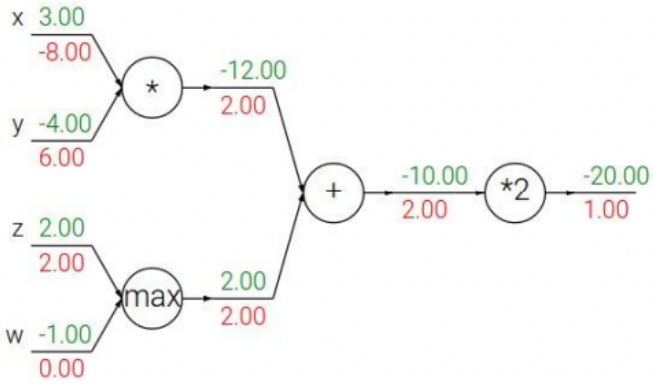
위와 같이 sigmoid gate로 한 묶음으로 노드를 구성해도 된다. sigmoid 를 사용하면 직접 analytic gradient를 할 수 있기 때문에 모든 게이트마다 backpropagation하는 것 보다 빠르게 값을 구할 수 있다.

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher



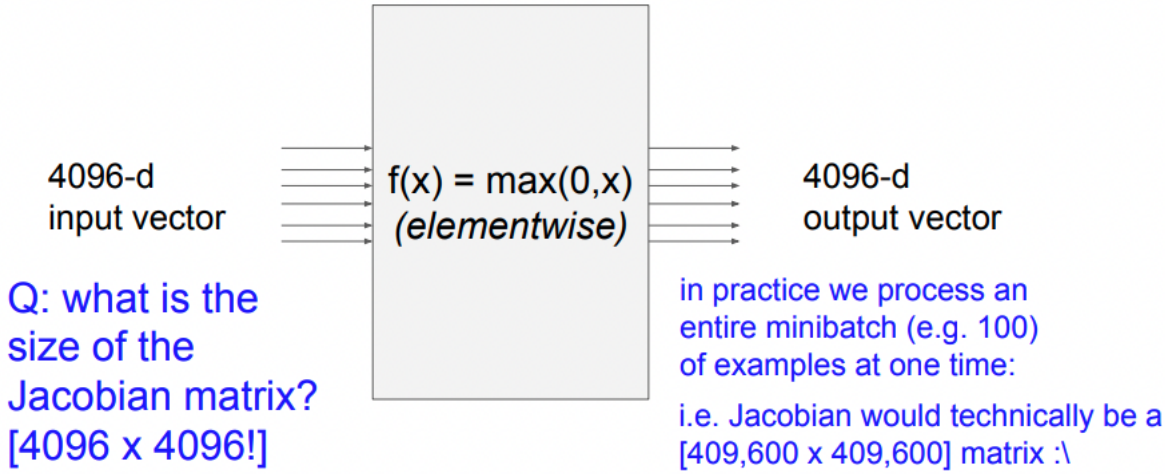
add gate: gradient를 그대로 전달

max gate: 큰 값인 gradint만 전하고, 다른 하나는 0으로 전달 된다.

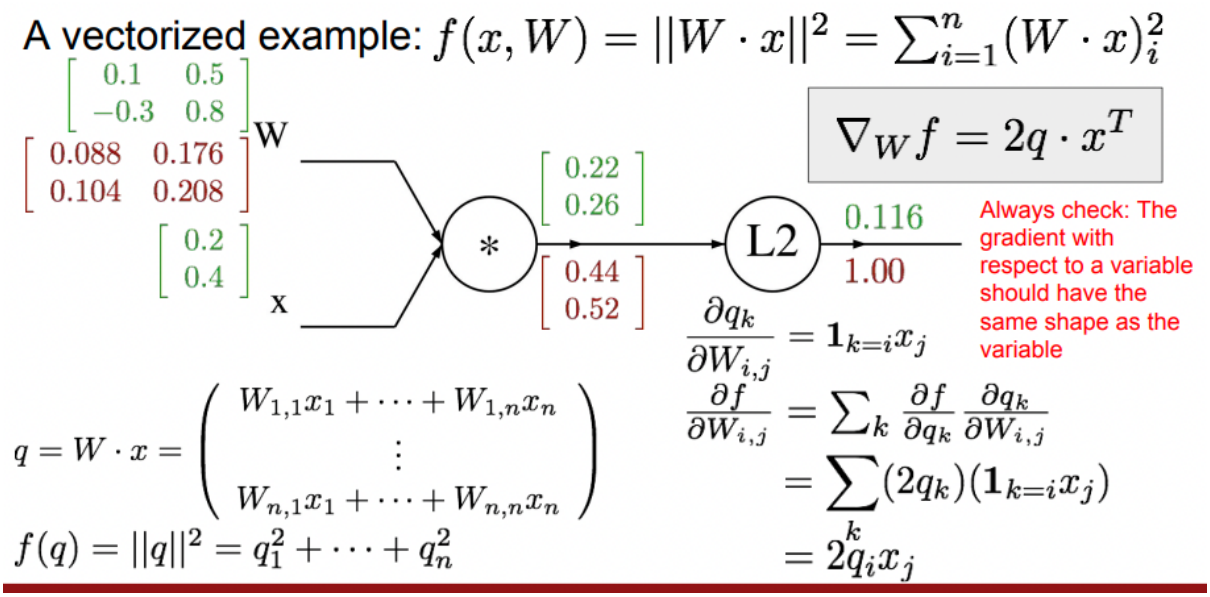
mul gate: 현재의 gradient를 각각 숫자에 곱해서 바꾼다.

input이 벡터 형태라면, jocobian matrix 형태로 구현된다.

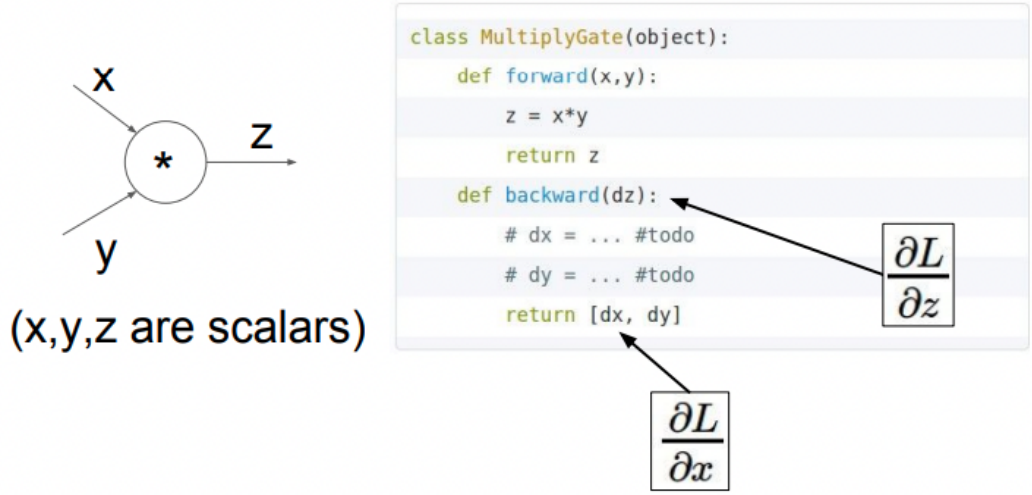
Vectorized operations



elementwise max 계산에서 input 벡터의 element가 ouput의 모든 element에 영향을 끼치는 것이 아니라 연관되는 하나의 element에만 영향을 끼치기 때문에 jacobian matrix는 diagonal하다. 때문에, 전체를 구하지 않아도 된다.

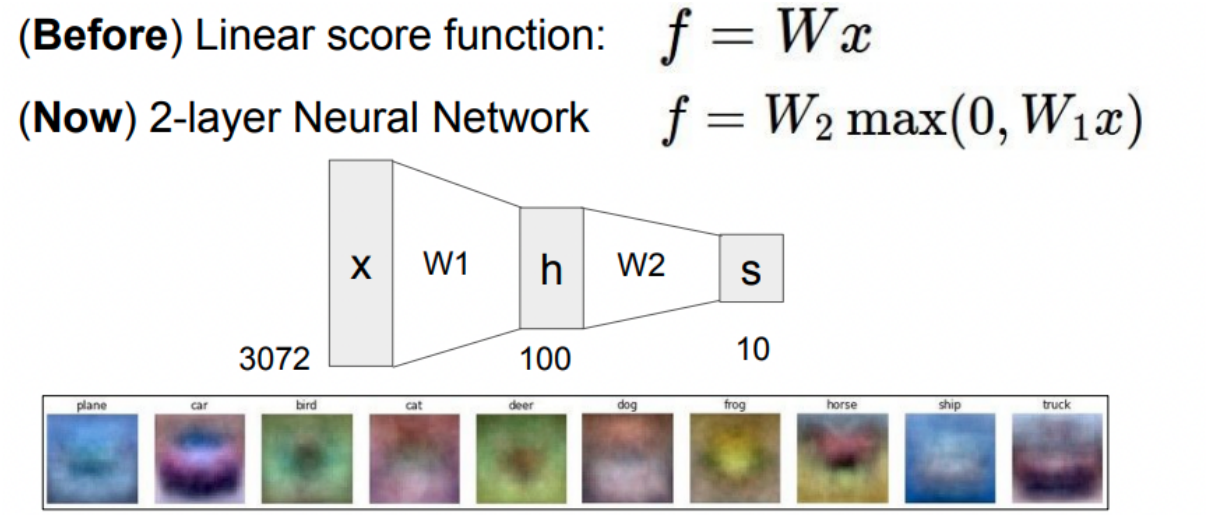


Modularizaed implementation- forward/backward API



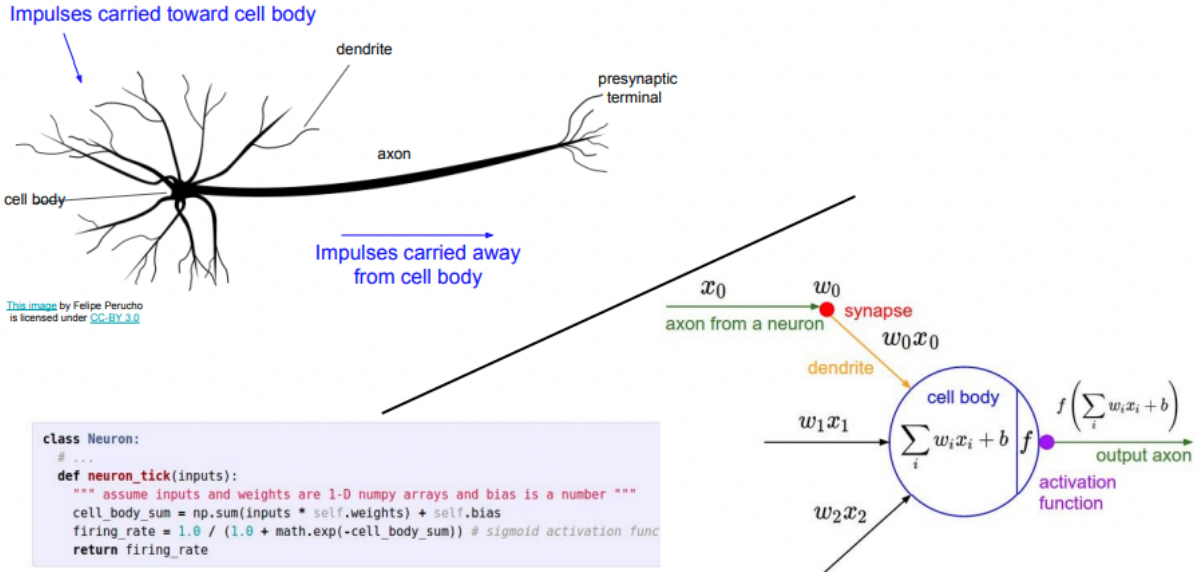
구현이 이렇게 될 것이지만… 실제로는 라이브러리 사용

Neural Networks



이전과 다른 점이 있다면 중간에 hidden layer을 끼워 넣은 것이다. 이전에는 W는 해당 클래스를 닮은 모양 자체를 띄고 있었다. hidden layer을 통해 이러한 문제를 해결 할 수 있다. 이전에는 차에 대해 하나의 탬플릿만이 존재한다면 hidden layer에서 차에 대한 탬플릿을 여러개로 저장하고, 마지막 레이어에서 모두 차에 대한 스코어로 연결할 수 있다.

layer의 아웃풋을 다음 레이어에 넣기 전에 비선형 함수에 통과 시킨다. 선형함수를 사용하면 레이어를 쌓는 의미가 없다

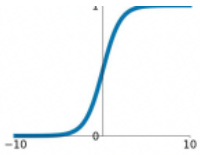


뉴런을 보면, dendrite를 타고 흐르는 신호가 cell body에서 합쳐진 뒤 axon를 통해 나간다. 뉴런과 가장 유사한 activation function을 Relu라고 한다.

Activation Functions

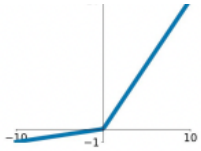
Sigmoid

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



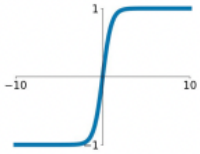
Leaky ReLU

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



tanh

$$\tanh(x)$$

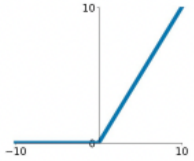


Maxout

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

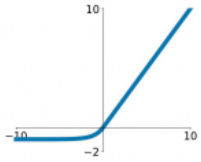
ReLU

$$\max(0, x)$$

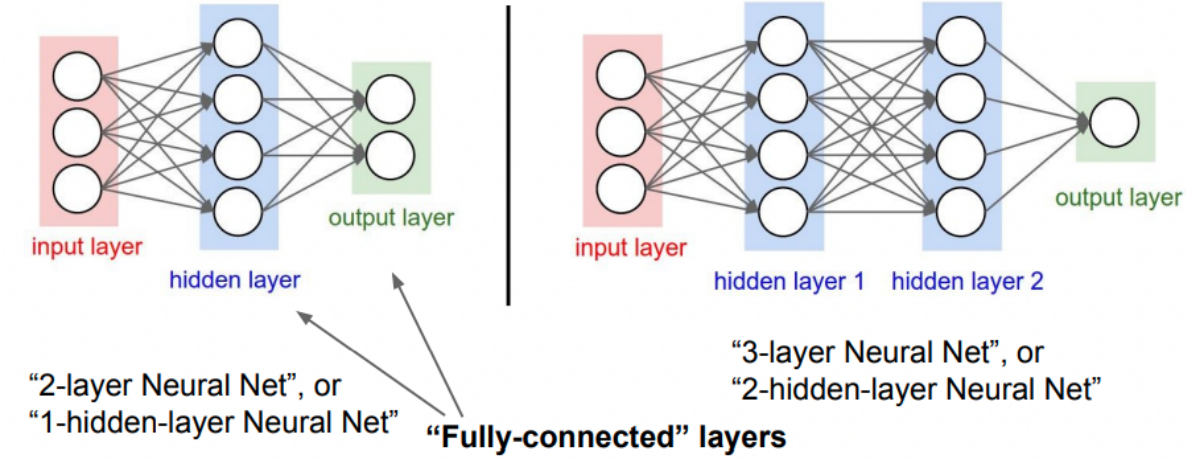


ELU

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



NN: architectures



모두가 연결되어있는 구조: FC