Review

- 각 class별로 스코어를 알려주는 scores funciton인 Wx.
- Loss function으로 hinge loss, cross-entropy
- Over-fitting 방지를 위한 regularization

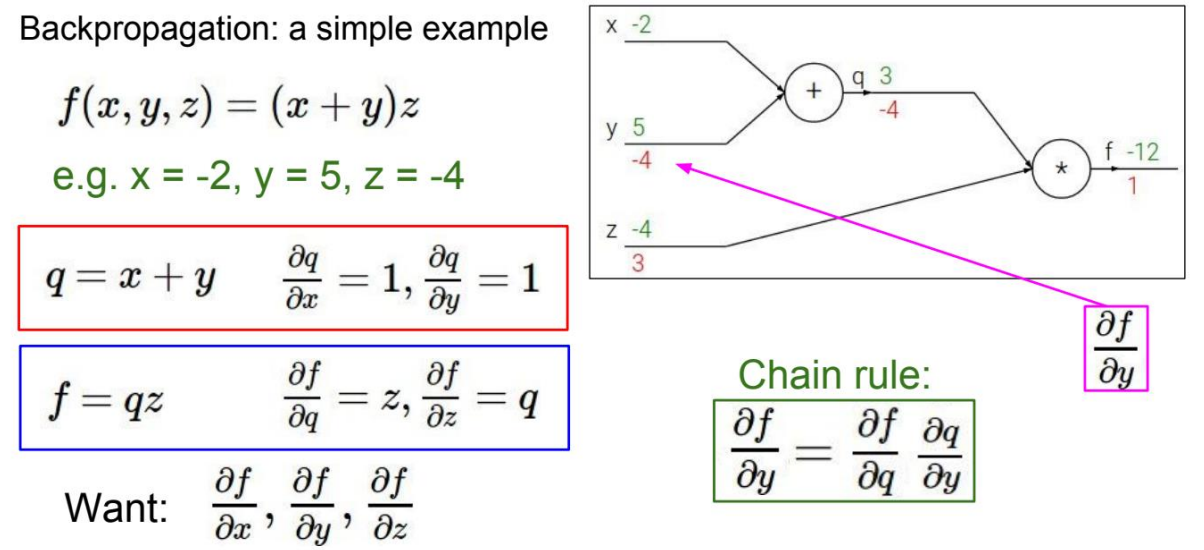
이제 loss func을 최소화하도록 가중치 W를 수정하는 방식에 대해 더 자세히 알아보자.

즉, gradient descent (경사 하강법)을 이용해 loss func(손실함수)가 최소화되도록 가중치 W를 수정해나갈 것. 이때 계산방식의 효율이 좋은 analytic gradient 방식으로 chain rule (합성 함수 미분)을 이용.

Back Propagation (오차 역전파)

결과값에서 거꾸로 거슬러 가중치 W를 계산해주는 것

[첫번째 예시]



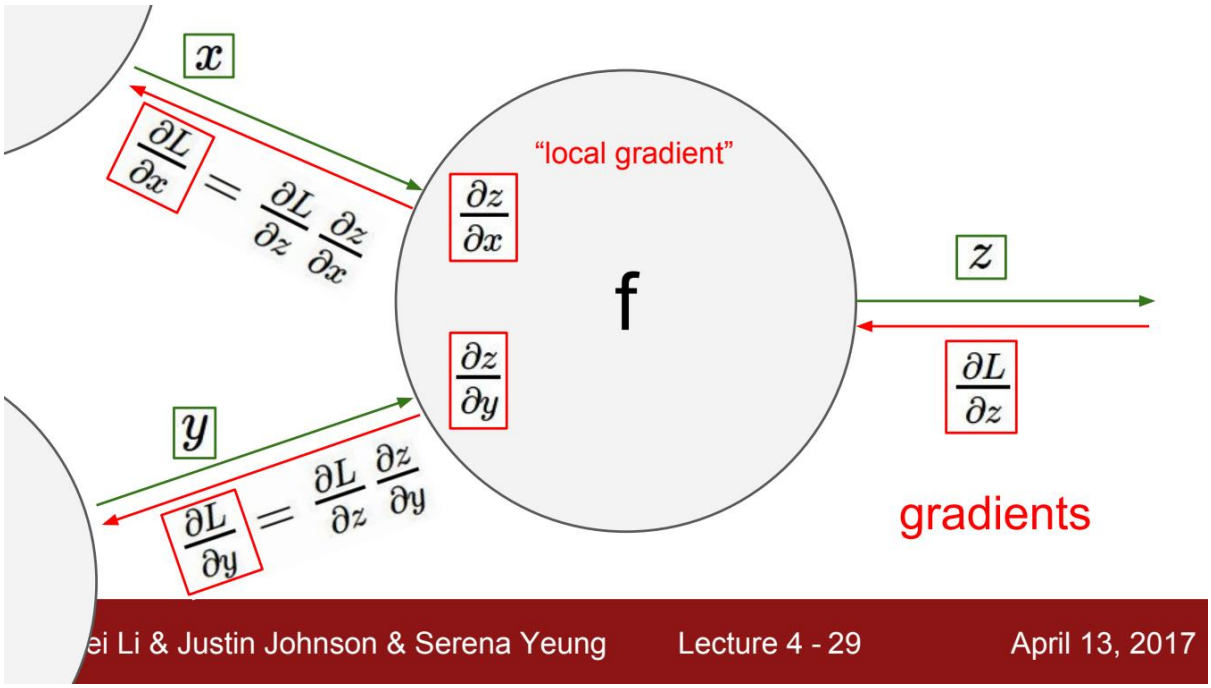
최종 f 값은 df/df 이므로 1.

문제에 의해 f=qz이므로 df/dz를 계산하면 q = 5 + (-2) = 3

마찬가지로, df/dq = z = -4.

dq/dx = 1, dq/dy=1로 구해뒀으므로, 이제 chain rule을 이용해 df/dq * dq/dy = df/dy = -4 * 1 = -4

마찬가지로 df/dx = 3 * 1 = 3



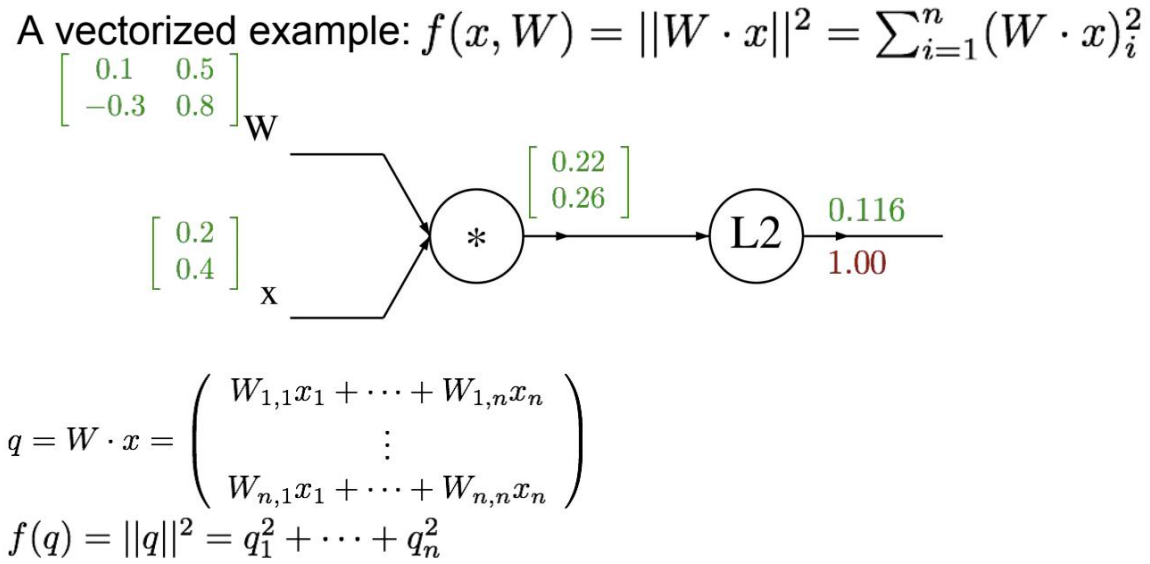
dL/dz 은 global gradient 가 되고, dz/dx 는 local gradient 가 됨.

global gradient 는 앞 노드에서 구해지기 때문에 체인룰을 이용하면 항상 계산이 가능.

[다른 예시]

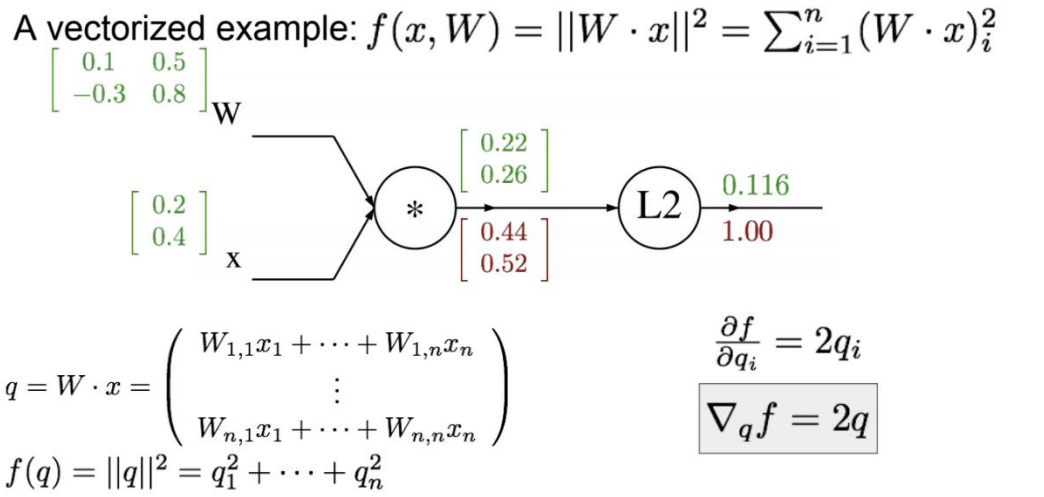
이번에는 L2 정규화를 한 함수 f 를 알아보자.

x 는 n 차원, W 는 n*n 차원.



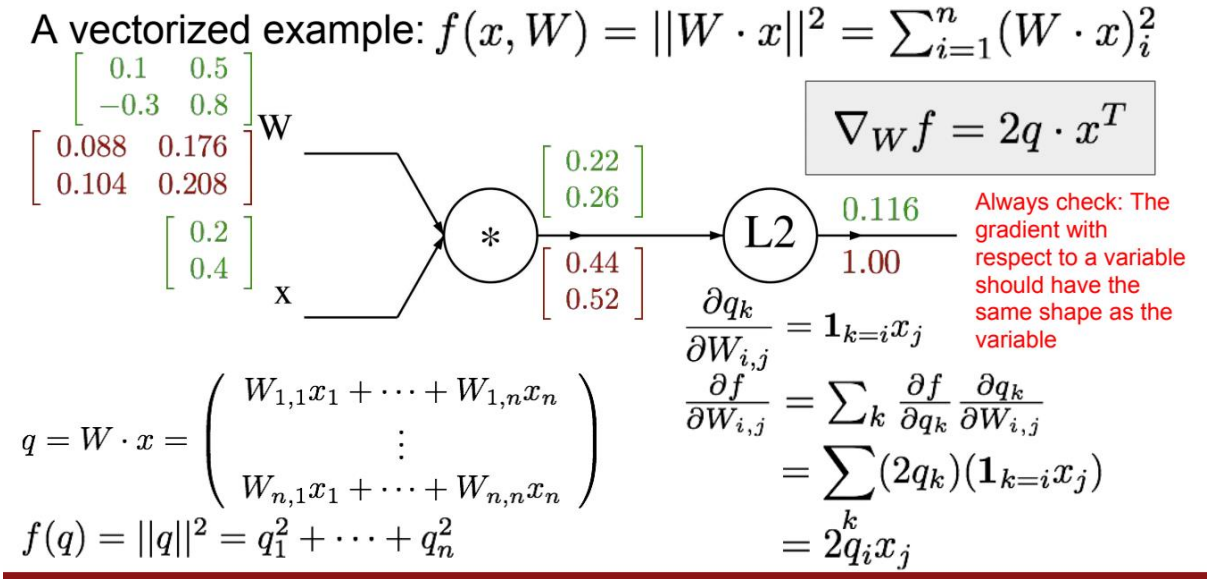
L2 의 최종값은 0.116 이 나왔고, 출력노드의 미분값은 당연히 1.

[0.22, 0.26]의 값은 W 와 x 의 행렬 연산으로 계산한 값 (forward)



f(q)로 함수를 지정했을때, 맨아래 식처럼 q^2 의 sum 으로 표현 가능합니다.

이들은 전부 2q 로 미분되므로 가중치는 곱하기 2 를 해준 [0.44, 0.52]가 됩니다.

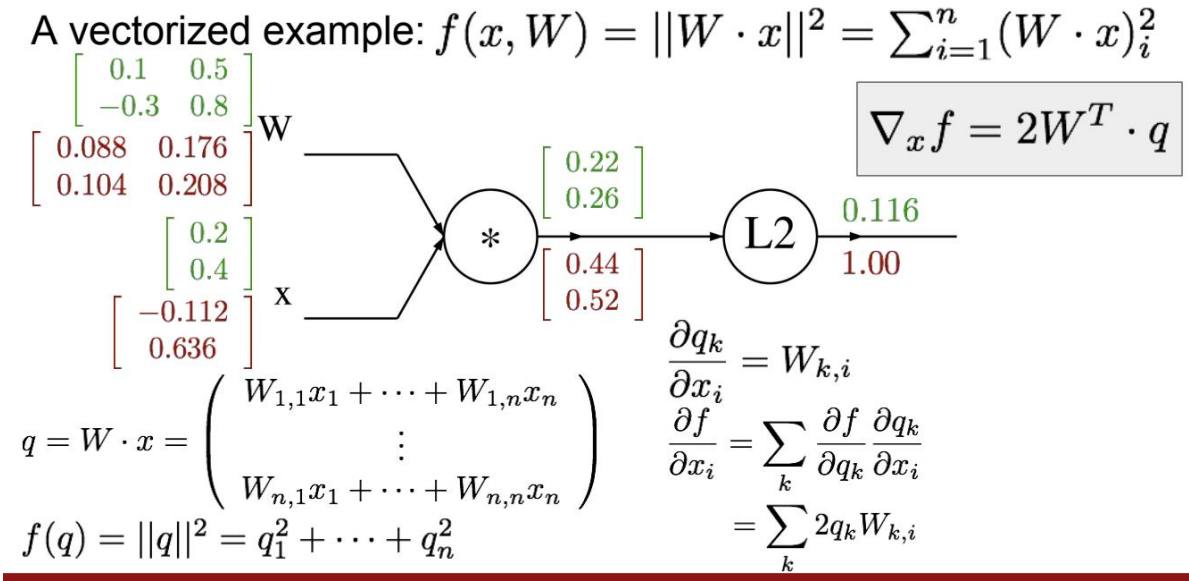


이제 곱하기 게이트.

곱하기 게이트는 서로가 서로의 계수이기 때문에 서로 바꿔주면 됨.

[0.2, 0.4] * [0.44, 0.52] 를 서로 행렬 계산해서 2*2 행렬을 만든다. (W 가 2*2 행렬이기 때문)

그래서 [[0.2*0.44, 0.4*0.44], [0.2*0.52, 0.4*0.52]]의 값이 도출된 것.

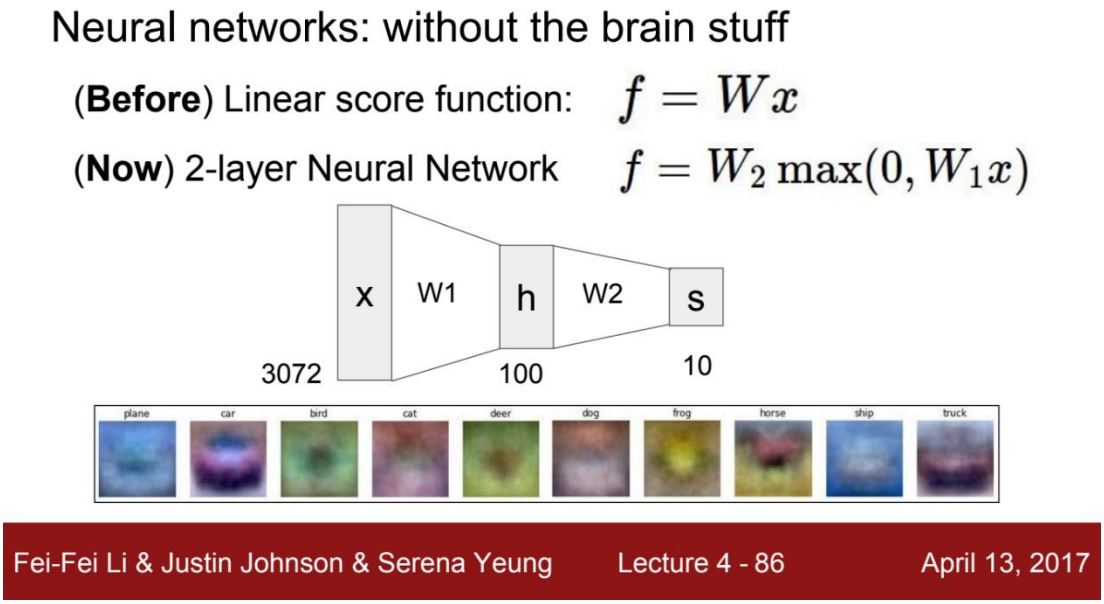


공식으로 x 에는 local gradient 가 w 값들이 됩니다!

Transpose(전치)를 조심해야함. 트랜스포즈를 한 이유는 Wx 에서 x1 에 들어가는 w 가 0.1 과 0.3 이기 때문.

따라서 (0.1*0.44)+(-0.3*0.52)=-0.112, (0.5*.044) + (0.8 * 0.52) = 0.636 의 값이 도출된 것.

Neural Network



이제 단순한 Linear regression 이 아니라 히든레이어를 추가.

히든 레이어를 input 노드와 ouput 노드 사이에 100 개를 추가했고, 이를 102 - layer 라고 하는게 아니라, **2 - layer NN 이라고 함.**

이는 예로 들었을때, NN(Nerest Neighbor)와 같은 모델은 각각 하나의 클래스에 하나의 분류기로만 이용해서 그 특징에만 부합한 클래스만 부여했던 것인데 히든레이어를 포함하므로써 **한개의 클래스에도 여러개의 분류기가 생겨 다양한 특징을 잡아 분류할 수 있게 된 것.**

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

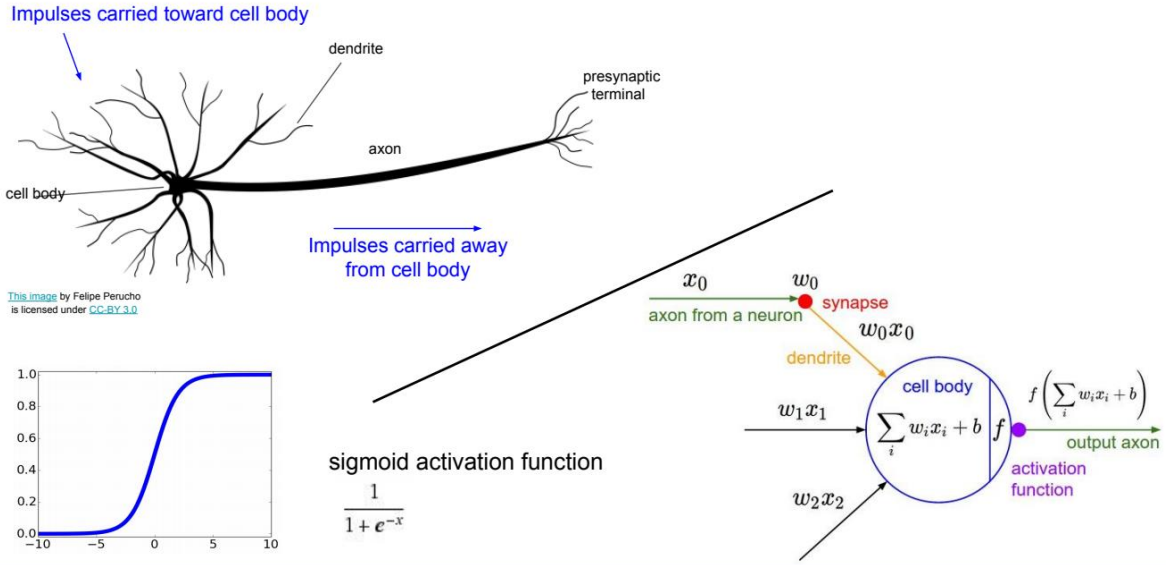
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

이렇게 2-layer, 3-layer NN 을 만드는데, 함수는 max 함수를 이용함.

이는 활성화 함수를 말하는데 예전에는 시그모이드를 활성화 함수로 많이 이용했지만, 최근에는 relu 함수를 많이 이용함.



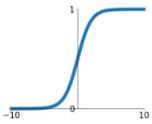
이런 NN 이 사람의 신경망과 유사함. (하지만 실제 생물학과는 차이가 많음)

데이터를 input 해서 w 를 부여하고 활성화 함수를 통해 출력값을 출력하는 것.

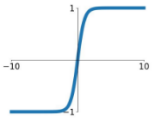
위에서 말한 시그모이드 함수가 바로 대표적인 예.

Activation functions

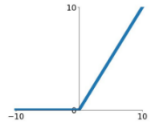
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



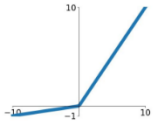
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

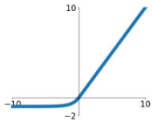


Leaky ReLU
$\max(0.1x, x)$

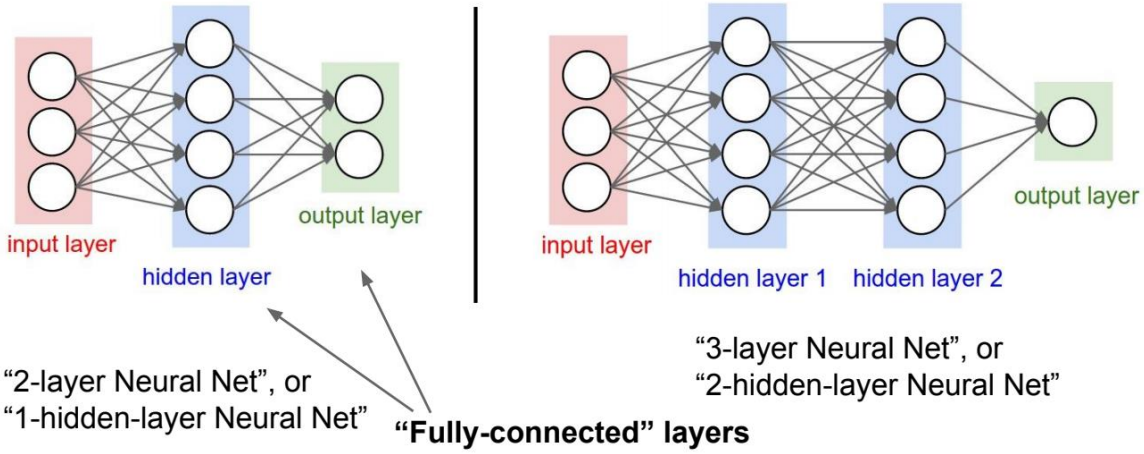


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

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



Neural networks: Architectures



이렇게 인풋, 여러 개의 히든, 아웃풋 레이어는 전부 서로의 모든 노드에 관여하여 값을 도출.

그래서 이를 FC(Fully Conneted)라고 부름.