Lecture4

경사하강법: 저번 시간에 이야기 했던

수치적 경사 Numerical Gradient(어림값, 작성 쉽다), 분석적 경사 Analytic Gradient(정확함, 실수하기 쉽다).

-> 분석적 경사 (계산 그래프 Computational Graph)에 대한 이야기

Convolution network, AlexNet:

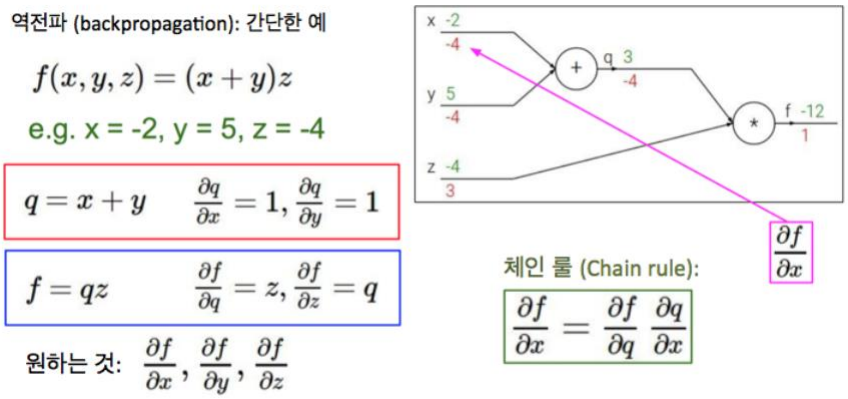
image(input) -> weights(가중치) -> loss(손실)

Neural Turing Machine:

이미지 -> 손실. (규모가 엄청나다) -> 딥러닝 모델

Forward Pass(FP): input이 마지막 단계에 어느 영향을 끼치는 가.

backpropagation(역전파): FP값을 구하기 위해, 역순으로 계산하는 것.




chiain rule: dq/dx는 원래 구하고자 하는 gradient를 위한 값이므로, 이를 local gradient

df/dq는 global gradient라고 함.

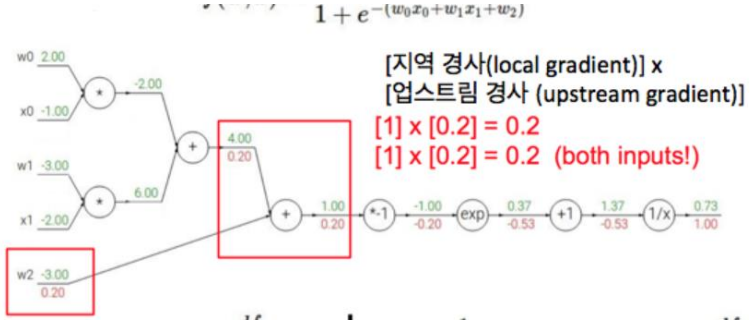
-> Forward Pass(FP) 시에는 local gradient값을 바로 구해서 메모리 저장 가능.

Backpropagation(역전파)시에는 local gradient와 global gradient값을 곱해줘서 gradient를 구할 수 있다.

한마디로 역전파시에는, chain rule이 발생한다.

input이 더 많을 때:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

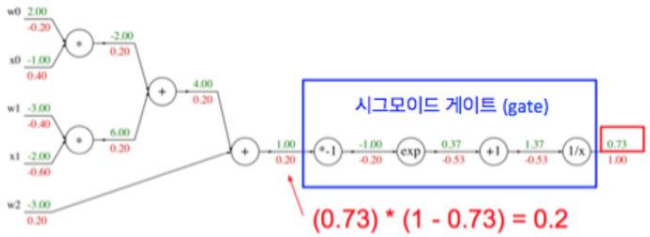



+ 연산은, gradient distributor. 그냥 동일한 연산을 전파해준다.

위에 식은 시그모이드 함수랑 형태가 비슷한데,

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$ 시그모이드 함수 (sigmoid function)

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$




이런식으로 간단하게도 계산이 가능하다.

Patterns in Backpropagation:

add gate: gradient distributor(경사 분배기)

max gate: gradient router(경사 라우터)

mul gate: gradient switcher(경사 스위처)

Vectorized code(벡터화된 코드)경사

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

-> 가운데 파란색이 Jacobian matrix.

만약 4096차원의 입력벡터로 넣고, 4096이 출력된다면, 이 중간 matrix값은 [4096 x 4096]크기가 됨.

신경망:

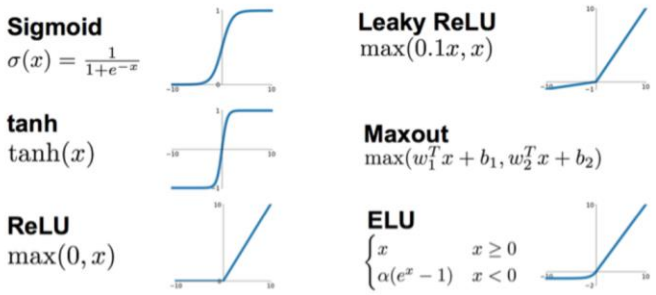
기존에는, f = Wx라고 간단히,

이제는 2-layer nn을 다를 것이기 때문에, f = $W_2$max(0, $W_1$x)로

3-layer nn은 f = W3max(0, w2max(0,W1x))

activation function중에서, sigmoid function을 가장 많이 사용한 이유가, 0~1 사이의 값을 갖기 때문.

활성화 함수:



Lecture5

neural network의 역사:

1957년 퍼셉트론 (역전파 불가) -> 아다린,마다린(퍼셉트론 쌓은 것, 동일하게 역전파 불가)

-> 루멀하트 논문(역전파, 문제가 많아서 암흑기) -> 2006년(역전파 제대로 동작하는) -> 연구 활성화(딥러닝이라는 말 나옴) -> 2010년대 초반, 딥러닝 폭발적 발전.

오늘날, CNN은 많은 곳에 사용된다 ex) 영상 분리, 자율주행(GPU에 의해 가능), 얼굴인식, 자세인식, 이미지 캡셔닝(이미지에 대한 문장 기술), 딥드림(요상한 그림)

Fully Connected Layer(완전 연결 계층):

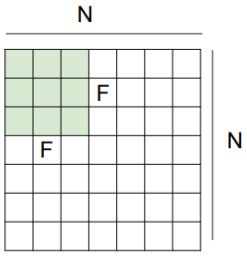
input -> W(가중치) -> activation

ex) 32 x 32 x 3 의 이미지 -> 3027 x 1 배열.

Convolution Layer(합성곱 계층):

-> 위와 다르게, 일렬로 늘리는 대신, 3차원 구조 유지(중간에 필터 껴넣는 형태).

CNN은 Convolution Layer(합성곱 계층)의 결과, activation function(활성함수. ex ReLU)가 중간중간 껴있다.



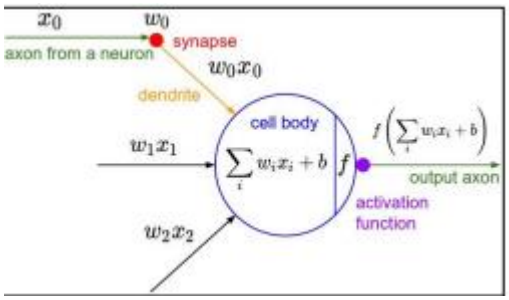

Output size:
**(N - F) / stride + 1**

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

-> spartial dimension(공간 차원)에 맞춰서, stride크기를 고려해야 한다.

실제에서는 0으로 패딩(zero pad).

CONV계층(커널 넓이, 패딩)의 예시: 딥러닝 프레임워크(Torch, Caffe




여기서 파란 동그라미는 local connectivity(지역연결, 전체적인 연결 말고 단순 공간적 연결)

그리고 5x5의 필터를 receptive field(수용장)라고도 부름.

Pooling layer(풀링 계층):

다운샘플링을 위해서(더 작고, 관리하기 쉽게 하기 위함)

흔히 사용하는 Max Pooling.

-> 풀링계층에서는 다운샘플링을 위해 0패딩을 사용하지 않는다.