Lecture 4 | Introduction to Neural Networks

Backpropagation: a simple example

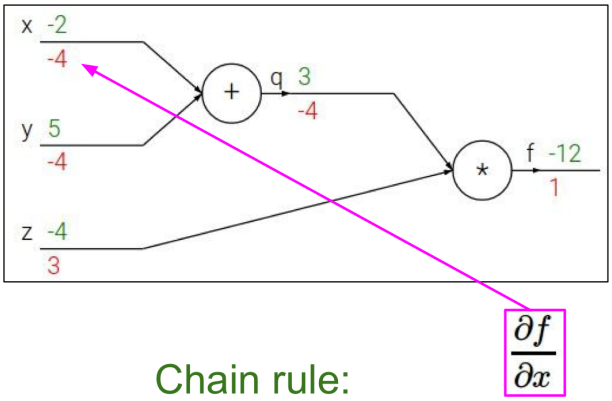
$$f(x,y,z) = (x+y)z$$

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:

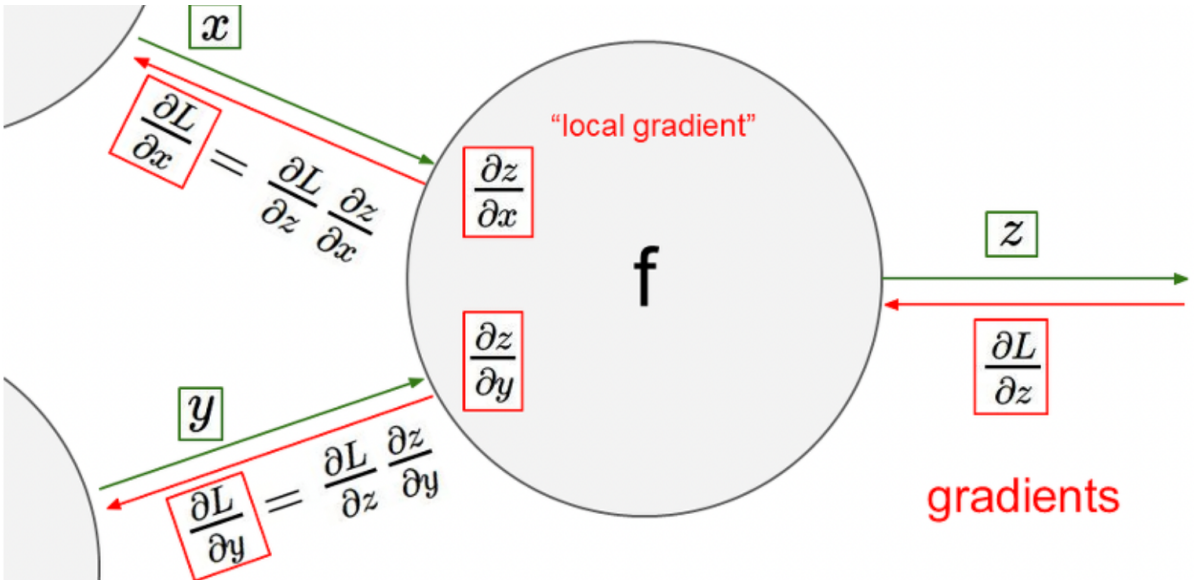
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

지난 강의에서 배운 gradient를 이용하여 가중치를 업데이트 시키는 과정 – backpropagation 역전파에 대한 간단한 예시로 알아봄.

x, y, z가 각각 f에 미치는 영향? 미분 (df/dx, df/dy, df/dz)

=> 바로 나오지 않는 경우 chain rule을 이용

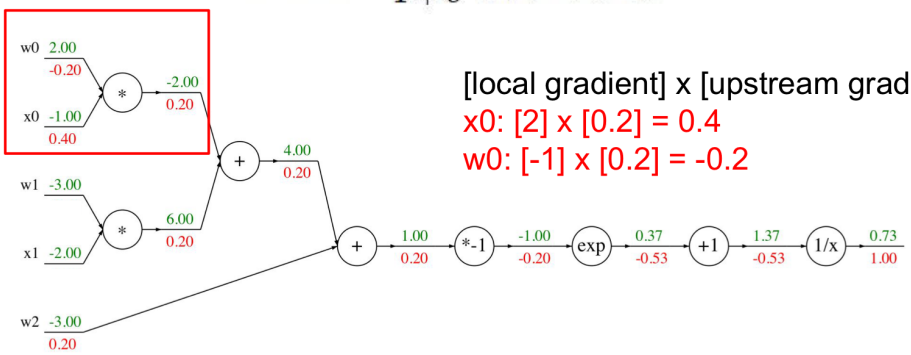
*. 이때, 덧셈은 미분값이 1이 나오고, 곱셈은 서로의 값이 나온다.



- local gradient: forward에서 구하는 그레디언트
- global gradient: backward에서 구하는 그레디언트

=> chain rule을 이용해 둘을 곱하여 gradient를 구한다.

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



[local gradient] x [upstream gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

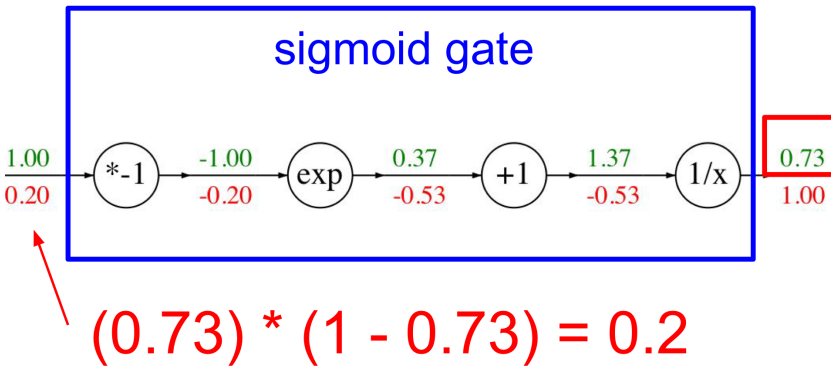
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

: 또 다른 예시 – sigmoid

위에서부터 계산된 upstream gradient(빨강)와 local gradient(초록)를 곱해 gradient를 구함



(0.73) * (1 - 0.73) = 0.2

- sigmoid gate = > 미분을 통해, (1-sigmoid)*sigmoid

이 경우 (0.73)*(1-0.73)으로 그레디언트를 구할 수 있음

- add gate: gradient distributor

local gradient가 1이기때문에, global * local = global

- max gate: gradient router

여러 값들 중 한가지(작은 값은 0이 되므로)만을 전해줌

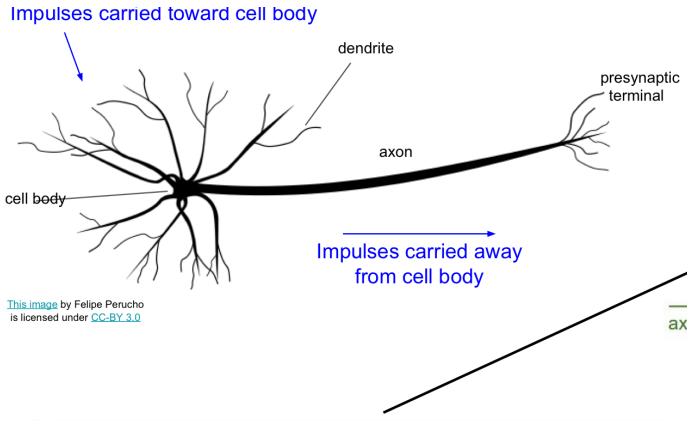
- mul gate: gradient switcher

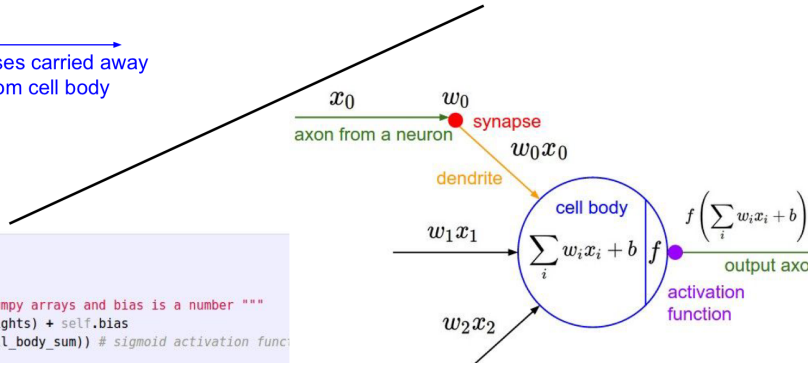
서로 값이 바뀜

*. 만일 노드가 여러개이면? 값을 더하고 벡터일 경우 자코비안(?) 행렬로 쓴다

[Neural Networks]

- 이전 Linear score function: f=Wx
- 2-layer Neural Network: f = W_2max(0, W_1x)
- 3-layer Neural Network: f = W_3max(0, W_2max(0,W_1x))






```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation func
    return firing_rate
```

뇌의 뉴런: 자극이 들어오면 axon을 통해 다른 뉴런으로 들어가며 서로 영향을 줌

-> input에 가중치를 곱해 활성화 함수를 통해 output axc로

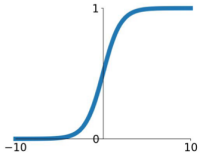
*. 단, 딥러닝의 신경망과 실제 뉴런은 다른 것임

[activation functions]

Activation functions

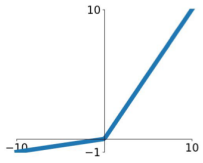
Sigmoid

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



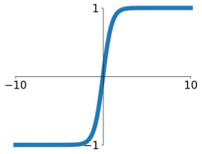
Leaky ReLU

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



tanh

$$\tanh(x)$$

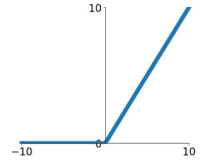


Maxout

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

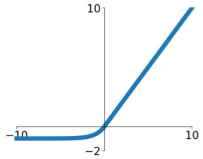
ReLU

$$\max(0, x)$$



ELU

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



- sigmoid를 비롯하여 다양한 활성화 함수가 있음
- ReLU를 가장 많이 사용함