Lecture 5: Backpropagation

강의 출처:

https://www.youtube.com/watch?v=isPiE-DBagM&t=2043s&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6&index=5

들어가기 전에

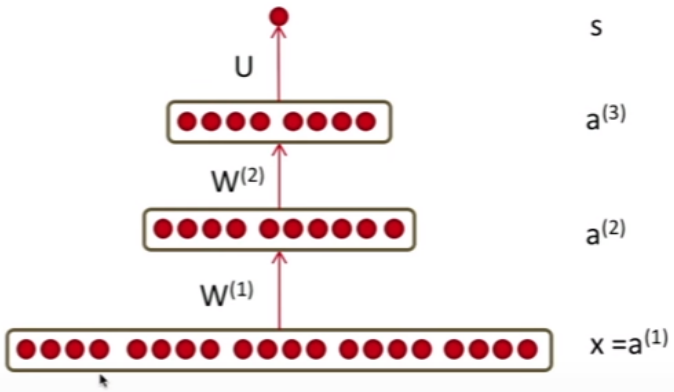
- 이번 강의에서는 backprop에 대해 4가지 방식으로 설명을 해줄 것이다.
- backprop을 통달시키려는 socher 교수님의 친절함을 알 수 있다.

Explanation #1 for backprop

Two layer neural nets and full backprop

- Let's look at a 2 layer neural network
- Same window definition for x
- Same scoring function
- 2 hidden layers (carefully define superscripts now!)

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

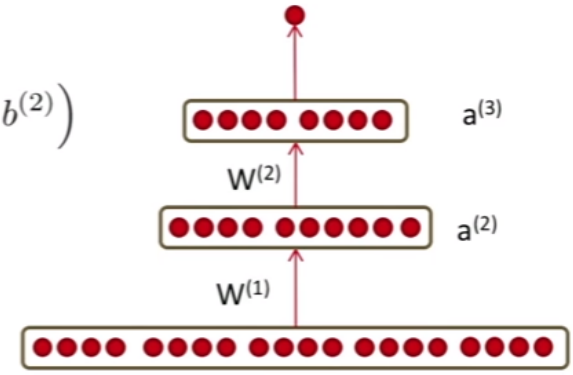


- 전의 강의와 모두 똑같지만, hidden layer하나를 더할 것이다.
- x는 첫번째 activation이자 첫번째 hidden layer이다. 그리고 윈도우, 즉 단어를 concatenate한 것이다.
- $a^{(2)}$에서 sigmoid 함수같은 element-wise nonlinearity를 적용한다.
- $z^{(3)}$는 $z^{(2)}$와 같은 아이디어이지만, $z^{(2)}$와는 다른 차원을 갖는다.
- U는 column vector이다. 모든 x또한 column vector이다.
- 마지막 layer에 꼭 linear layer를 선택해야되는 건 아니다. 원한다면 sigmoid함수를 써도 상관없다.

Two layer neural nets and full backprop

- Fully written out as one function:

$$\begin{aligned} s &= U^T f\left(W^{(2)} f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) \\ &= U^T f\left(W^{(2)} a^{(2)} + b^{(2)}\right) \\ &= U^T a^{(3)} \end{aligned}$$



- Same derivation as before for W(2) (now sitting on a(2))

$$\frac{\partial s}{\partial W_{ij}} = \underbrace{U_i f'(z_i)}_{\delta_i} x_j \qquad \frac{\partial s}{\partial W_{ij}^{(2)}} = \underbrace{U_i f'\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} a_j^{(2)}$$

- 전의 강의에서 했던 것처럼 $w^{(2)}$에 대해 미분을 해준다.

Two layer neural nets and full backprop

- Same derivation as last lecture for top W(2):

$$\frac{\partial s}{\partial W_{ij}^{(2)}} = \underbrace{U_i f'\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} a_j^{(2)}$$

$$\begin{aligned} x &= z^{(1)} = a^{(1)} \\ z^{(2)} &= W^{(1)} x + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\ a^{(3)} &= f\left(z^{(3)}\right) \\ s &= U^T a^{(3)} \end{aligned}$$

- In matrix notation: $\frac{\partial s}{\partial W^{(2)}} = \delta^{(3)} a^{(2)T}$

where $\delta^{(3)} = U \circ f'\left(z^{(3)}\right)$ and $\circ$ is the element-wise product also called Hadamard product ($\otimes, \odot$)

- Last missing piece for understanding general backprop: $\frac{\partial s}{\partial W^{(}}$

- δ는 두 vector U_i와 $f\prime(z_i^3)$의 곱이다. 그리고 해당 layer에서 오는 error 값이다.

- $\delta^{(3)}a^{(2)T}$는 외적이다. 외적을 하는 이유는 모든 i에 대한 쌍(pair)에 대해서 cross product를 주기 때문이다. 즉, 외적을 통해 w_{ij}에 있는 모든 정보를 이용하는 것이다.*

  *잘모르겠으면 해당 링크의 Training with backpropagation의 5번째 슬라이드 참조.

Two layer neural nets and full backprop

- Last missing piece: $\frac{\partial s}{\partial W^{(1)}}$
- What's the bottom layer's error message $\delta^{(2)}$?
- Similar derivation to single layer model
- We already derived $W^{(2)^T}\delta^{(3)}$ as the next lower update in a model where the next lower layer were just the word vectors
- But now, we'll need to apply the chain rule again: $f'(z^{(2)})$

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

- 2번째 layer에서의 update는 $w^{(2)T}\delta^{(3)}$이다. 지난번에 이미 배웠던 거다. 하지만 지난번에는 layer가 1개였기 때문에, 여기서 끝이었지만 지금은 layer를 한번 더 쌓았기 때문에 chain rule을 다시 적용할 필요가 있다.

Two layer neural nets and full backprop

- Chain rule for: $s = U^T f\left(W^{(2)} f\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)$
- Get intuition by deriving $\frac{\partial s}{\partial W^{(1)}}$ **as if it was a scalar**
- Intuitively, we have to sum over all the nodes coming into layer
- Putting it all together: $\delta^{(2)} = \left(W^{(2)^T}\delta^{(3)}\right) \circ f'\left(z^{(2)}\right)$

- vector의 미분이라 다소 복잡할 수 있지만 스칼라(차원이 없고 크기만 가지는 것)라고 생각하면 괜찮다고 한다...
- $\delta^{(2)}$는 이전 layer에서 구한 미분 값에 현재 layer의 미분 값을 element-wise 곱을 해준다.

Two layer neural nets and full backprop

- Final derivative: $\frac{\partial s}{\partial W^{(1)}} = \delta^{(2)} x^T$

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)} x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

- In general for any matrix $W^{(l)}$ at internal layer *l* and any error with regularization E_R all backprop in standard multilayer neural networks boils down to 2 equations:

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)}),$$

$$\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}$$

- Top and bottom layers have simpler δ

- 빨간 네모 안에 있는 두 방정식을 이해하면, 이제 모든 multilayer neural networks에 대한 update의 끝판왕을 안 것이다. update 마스터가 될 수 있다.
- 각각의 W에 대한 마지막 update는 항상 오른쪽 위에 나와있는 외적이 될 것이다. 전의 강의에서 나온 것과 살짝 다른 형태를 띄고 있는데, 단지 regularization을 더해줬다. 해당 layer의 activation에 δ배를 해주고 정규화를 해주는 것이다.

Explanation #2 for backprop: Circuits

- neural network에서 나오는 많은 matrix때문에 골머리가 날 뻔했다. 대신 이번 설명에서는 그냥 간단한 함수로 생각해 볼 것이다. 😀

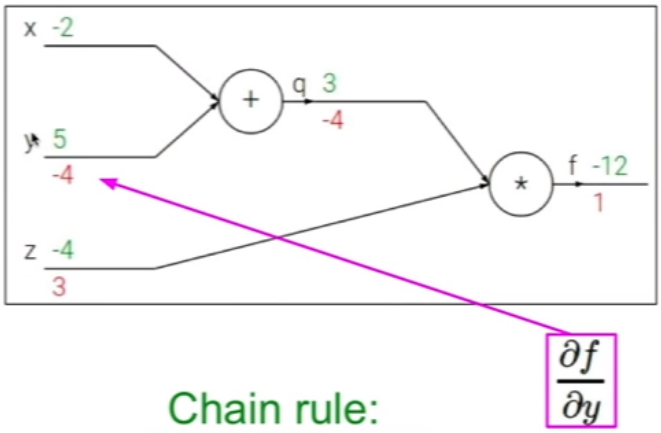
$$f(x,y,z) = (x+y)z$$

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



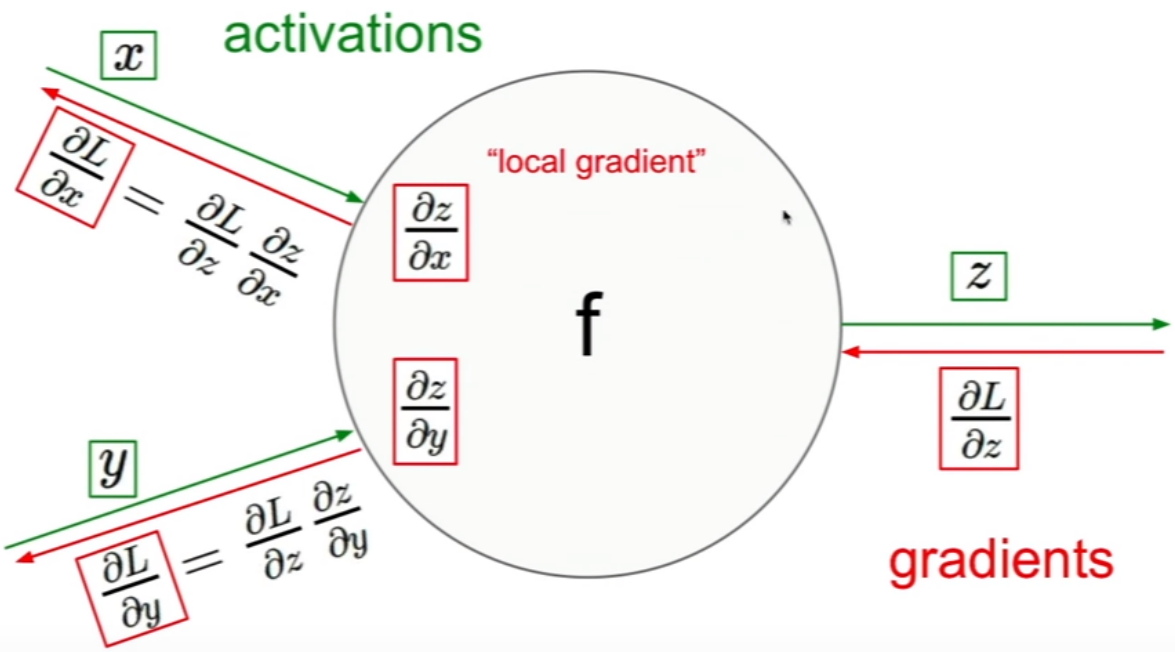
$\frac{\partial f}{\partial y}$

Chain rule:

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

- 즉, network를 이동하면서 반복적으로 error signal혹은 local gradient들을 적용, 계산할 것이다.
- f는 lost function이고 x,y,x는 parameter들이다.
- 예를 들어, x=-2, y=5, z=-4라고 해보자. q와 f에 대해 편미분을 해보면 왼쪽과 같은 결과를 얻을 수 있다.
- 우리는 이 변수들을 update하기 위해서 해당 변수들의 미분 값을 알아야 한다.
- 오른쪽 그림을 봐보자. 빨간색으로 쓰인 숫자가 미분 값이다. 변수들의 미분 값을 알기 위해서는 오른쪽에서부터 시작해야한다. f-12를 f로 미분하면 1이 나온다.
- z벡터를 update하기 위한 미분값은 $\frac{\partial f}{\partial z}$이다. 이 값은 손쉽게 구할 수 있다. 파란 박스에서 정의했듯이 이 값은 q이고, q는 $x+y$의 값인 3이다.
- $\frac{\partial f}{\partial q}$는 위와 비슷하 방식으로 파란 박스에서 나와있듯이 z이다. 하지만 주의할 것은 chain rule을 적용했기 때문에, higher node에서 온 미분 값을 곱해줘야 한다. 첫번째 설명에서 언급했던 것처럼 위에서 온 error signal을 전해받는 거라고 생각하면 된다. 그렇지만 여기서는 이전 미분값이 1이기 때문에 그냥 -4라고해도 상관없다.
- $\frac{\partial f}{\partial y}$는 chain rule을 적용해, 이전 node의 미분값인 -4와 local gradient, 해당 node에서의 미분값인 $\frac{\partial q}{\partial y}$, 1을 곱해주면 된다. 나머지 미분값인 $\frac{\partial f}{\partial x}$도 이와 마찬가지로 구해주면 된다.

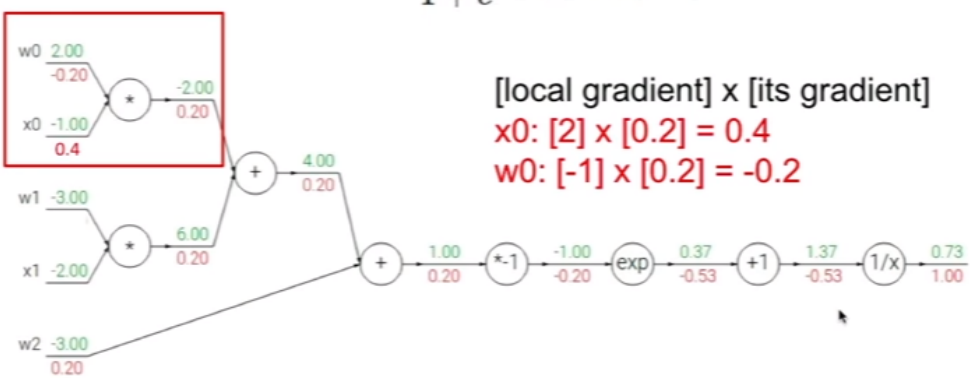
Recursively apply chain rule through each node



- 위의 그림에서 보듯이 각 node에 chain rule을 적용해, 우리가 최종적으로 구하고자 하는 변수의 미분 값, 즉 update값을 알아낼 수 있다.
- 여기서 중요한 것은 local gradient는 forward propagation때 구할 수 있다는 사실이다. 이 때 구한 값을 저장해뒀다가 back prop때 이용하면 된다.

Jumping to the end...

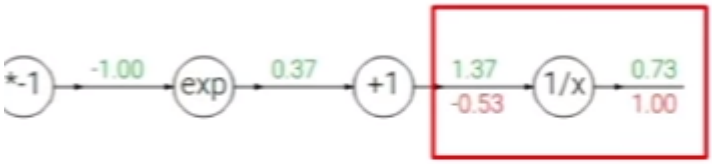
Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



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

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$	$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

$$(\frac{-1}{1.37^2})(1.00) = -0.53$$



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

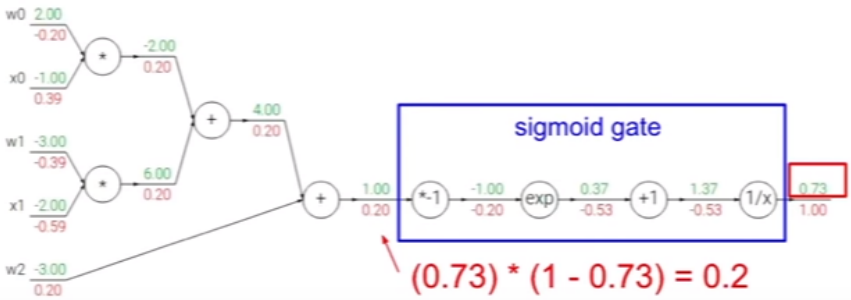
(초록색 숫자는 forward prop이고, 빨간색 숫자는 backprop이다. 앞에 미분과정이 더 나와있는 슬라이드도 있지만 생략하겠다. 위의 예시하나로 어떻게 하는지 감이 올것이다.)

- $f(w,x)$는 sigmoid 함수이다. 여기서 x는 input이고 w는 weights이다. 이때 목표는 모든 elements, w와 x에 대한 편미분을 계산하는 것이다.
- x를 2-dimension으로 w를 3-dimension으로 가정하자. w_2는 bias term이다.
- $f(w,x)$의 값은 parameter가 뭐든지 간에 분자를 1로 취할 것이다.
- 그래프의 밑에 나와있는 식을 이용하면 , 위의 슬라이드에서처럼 미분 값을 구할 수 있다. 제일 첫번째 미분 값이 1인 이유는 $\frac{\partial f}{\partial f}$를 하기 때문이다.
- higher layers에서 온 error signal을 계속해서 곱해나감으로써 미분 값을 다시 쓴다. 나중에는 변수의 미분 값인 local gradient와 위에서 부터 내려온 error signal을 곱함으로써 변수를 update할 수 있게 된다.

Combine nodes in the circuit when convenient

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

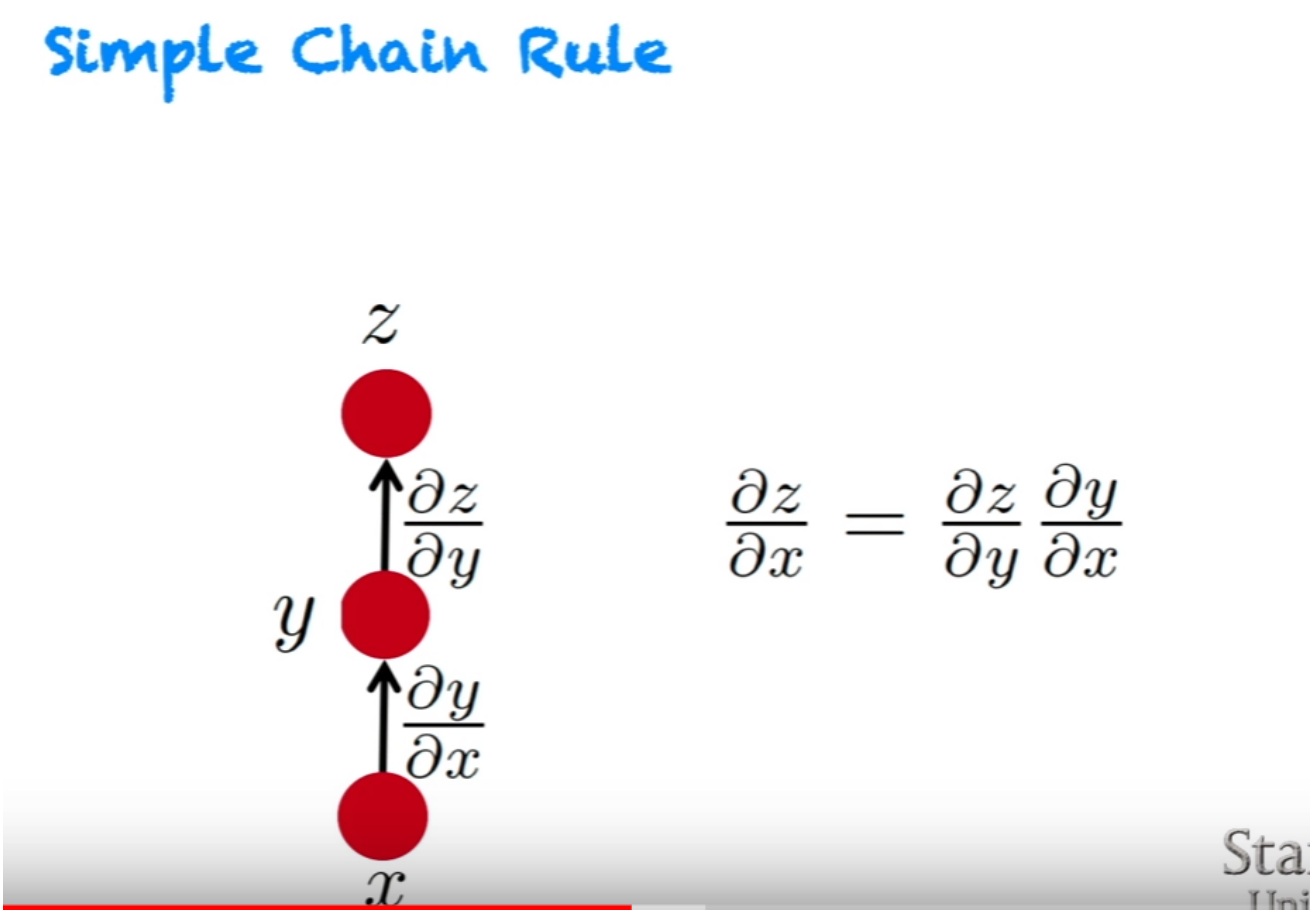
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



- 위의 슬라이드처럼 함수의 부분을 일일이 쓰는 건 매우 번거롭다. 그래서 위처럼 나타낸다. $\sigma(x)$로 sigmoid 함수를 나타낸다.

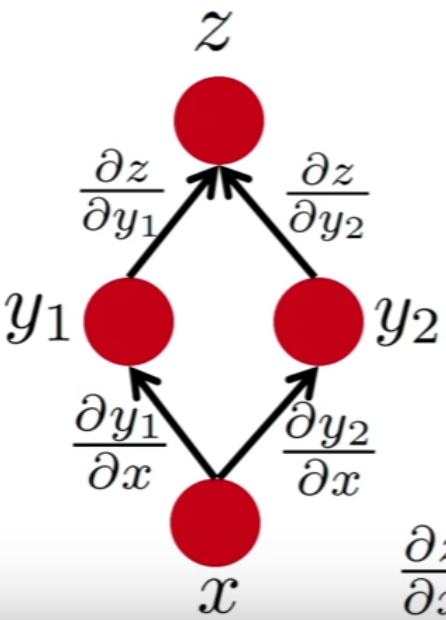
- 위 슬라이처럼 일일이 계산하지 않아도, sigmoid 함수를 x에 대해 미분하면 $(1-\sigma(x))\sigma(x)$ 가 나온다.
- 스탠포드 학생이 여기서 질문을 하는데, 그럼 forward prop은 뭐냐고 한다. forward prop은 그냥 전반적인 함수 값을 계산하는 것이다. 그리고 test time에 이뤄지는 과정이다.

Explanation #3 for backprop: The high-level flow graph



- x에서 시작해 어떤 값을 계산하기 위해, 중간 변수 y를 거쳐 forward prop을 할 것이다. 그리고 backprop에서는 forward prop과는 반대 방향으로 gradient를 계산할 것이다.

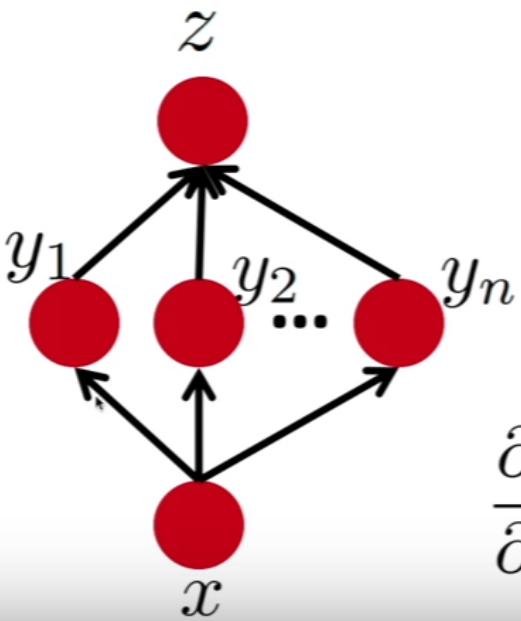
Multiple Paths Chain Rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

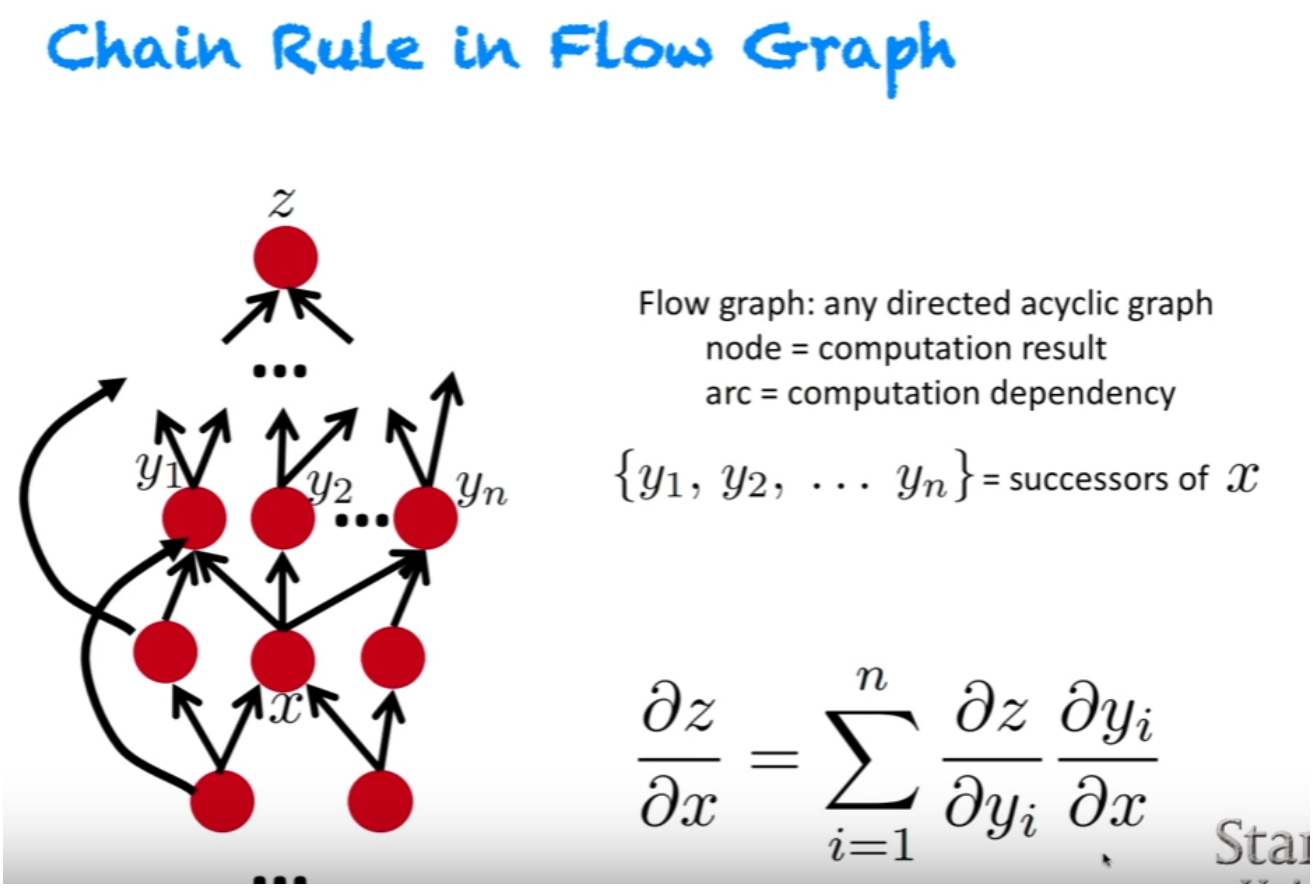
- 위의 슬라이스에서와 달리, $y1$과 $y2$에서 온 error signal들을 더해야한다.

Multiple Paths Chain Rule - General

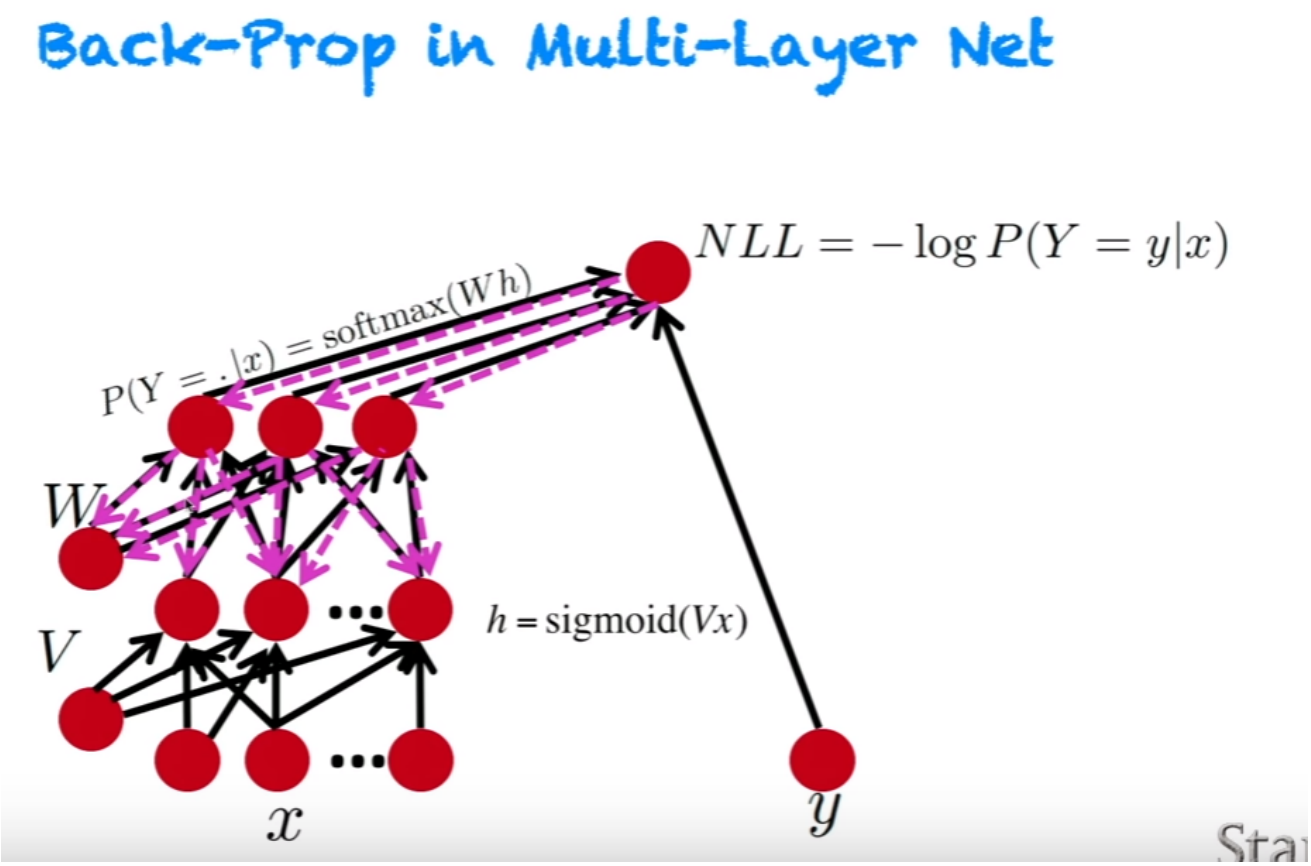


$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

- 일반적으로 x가 flow graph에서 multiple한 element들을 거친다면, 위같이 sigma를 써서 편미분 값을 더해주면 된다.



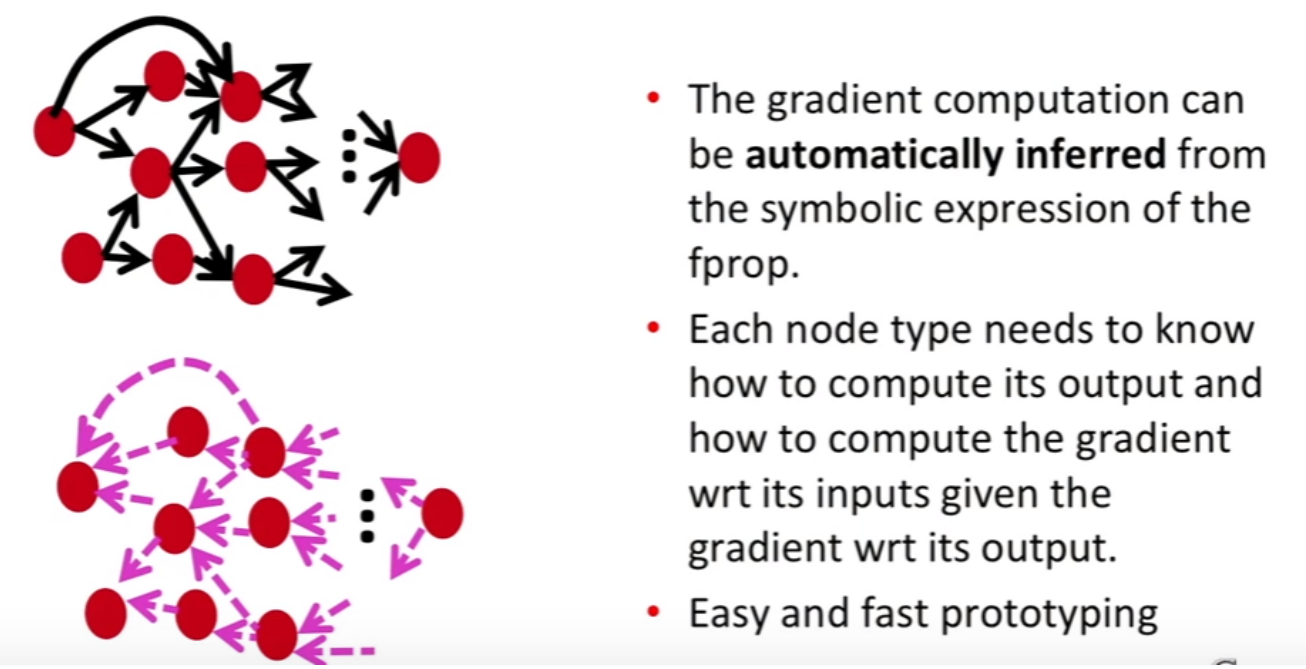
- 각각의 node는 계산 결과이고, 각각의 화살표는 계산이 어떻게 이루어지는지 보여준다. 화살표로 이어진 노드는 미분 값을 계산하기 위해 서로의 미분 값을 필요로 한다.
- 좀 더 복잡한 것도 정의할 수 있는데, 그림의 왼쪽을 보면 한 layer를 뛰어넘는 화살표가 보일 것이다. 이런 방식을 short circuit connections라고 부른다.



(검은 화살표가 forward prop, 분홍 화살표가 back prop)

- $\boldsymbol{x}$는 input이고 $\boldsymbol{y}$는 class이다.
- forward prop에서 sigmoid neural layer를 거칠 것이다. $\boldsymbol{h}$는 $\boldsymbol{Vx}$의 sigma이다.
- 다음 layer로 이동할 것이고, 마지막쯤에 softmax layer를 만날 것이다.
- 그 다음에 negative log likelihood를 거쳐 x와 y의 pair에 대한 cost function을 계산할 것이다.
- 그리고 forward를 했으니, parameter들을 update하기위해 back prop을 한다.

> 하지만 굿뉴스는 일일이 계산할 필요가 없다 ㅎㅎㅎ 이미 좋은 패키지들이 나와서 알아서 다해주기 때문이다. socher 교수님이 박사 시작하실때만 해도 이런게 없었다고 한다...

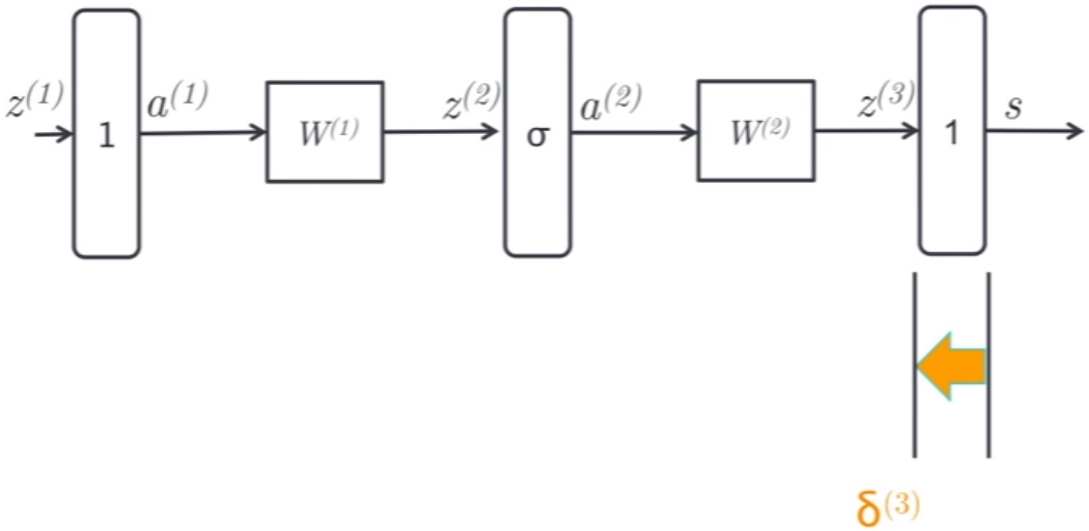


- gradient의 계산은 알아서 도출된다.

Explanation #4 for backprop: The delta error signals in real neural nets

Visualization of intuition

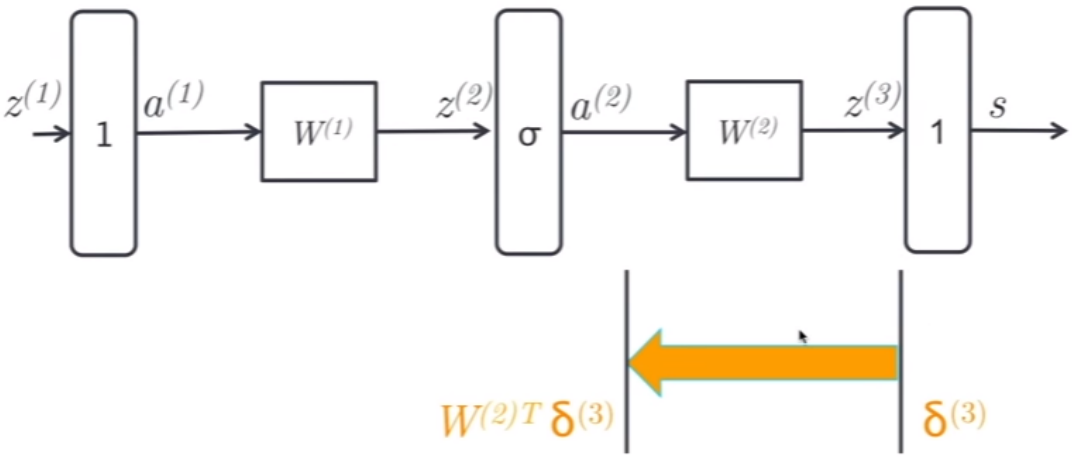
- Let's say we want $\frac{\partial s}{\partial W^{(1)}} = \delta^{(2)} a^{(1)^T}$ with previous layer and f = σ



Gradient w.r.t $W^{(2)} = \delta^{(3)} a^{(2)\mathrm{T}}$

- 첫번째 설명에서 나오던 복잡한 함수 구성을 좀 더 간편하게 표현했다.
- $\delta(3)$는 score에서 오는 error signal이다.
- 이때, $W(2), W(1)$모두 update하고 싶다.
- linear score을 지나갈 때 delta는 변하지 않는다.
- $W(2)$에 대한 update는 그냥 $\delta^{(3)} a^{(2)T}$의 외적값이다.

Visualization of intuition

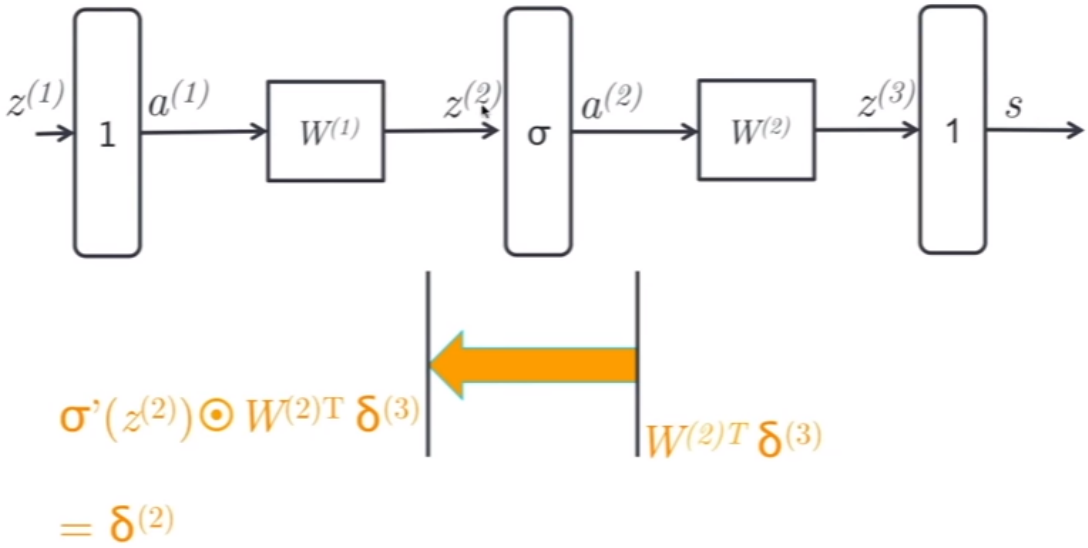


--Reusing the $\delta^{(3)}$ for downstream updates.
--Moving error vector across affine transformation simply requires multiplication with the transpose of forward matrix
--Notice that the dimensions will line up perfectly too!

- matrix vector product 대한 간단한 affine 변환*을 거칠때, 그냥 forward prop matrix를 transpose한 것만 있으면 된다.

  *affine transformation 설명
- output에서의 dimension은 n행 1열이다. 이 vector에 δ를 곱한다. 그러면 output의 dimension과 같아진다.
- $W^{(2)T}\delta^{(3)}$는 W와 같은 dimension을 갖는다.

Visualization of intuition



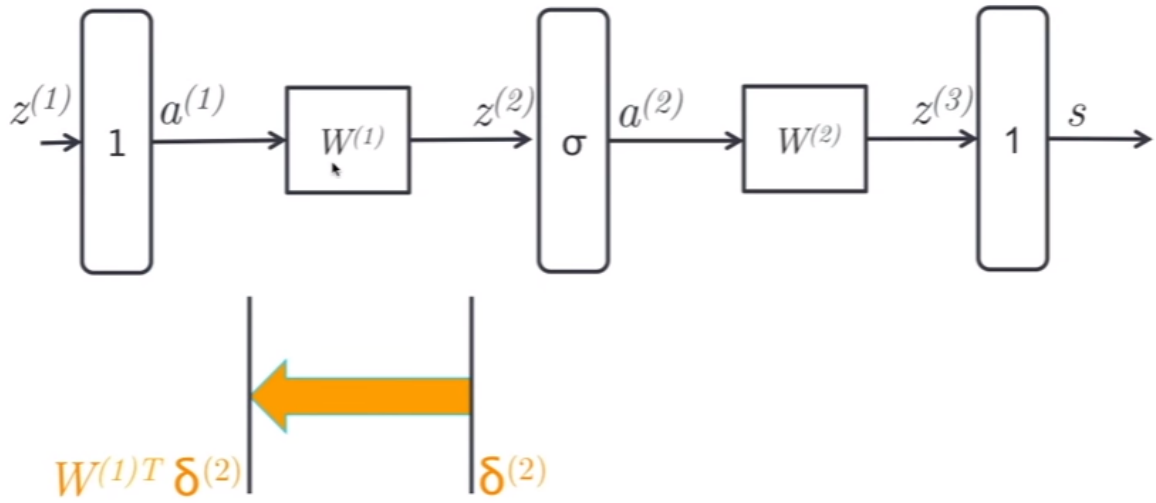
--Moving error vector across point-wise non-linearity requires point-wise multiplication with local gradient of the non-linearity

Sta

- σ는 element wise nonlinearity를 특성으로 갖는다. (여기서는 sigmoid)
- 그래서 다음 δ를 update할 때, 즉, error vector(error signal)을 point-wise* nonlinearity을 통과할 때, non-linearity의 local gradient와 point-wise 곱셈을 적용해야 한다.

  *element wise
- 이 과정을 거쳐 $W(1)$에 도달하는 $\delta(2)$를 얻었다.

Visualization of intuition



Gradient w.r.t $W^{(1)} = \delta^{(2)} a^{(1)\mathrm{T}}$

Sta

- 이제 $W(1)$에 대한 마지막 gradient를 계산할 수 있게 되었다.