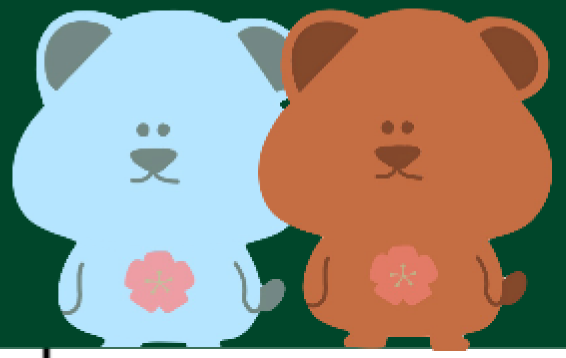
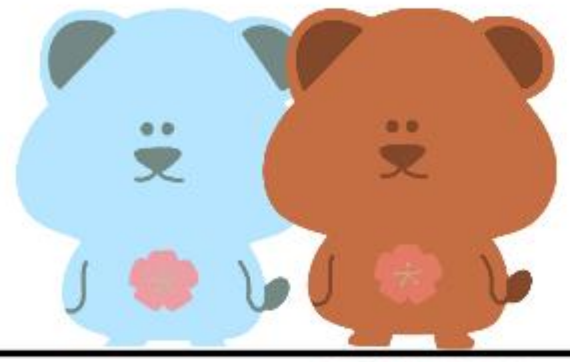


Backpropagation and Computation Graphs

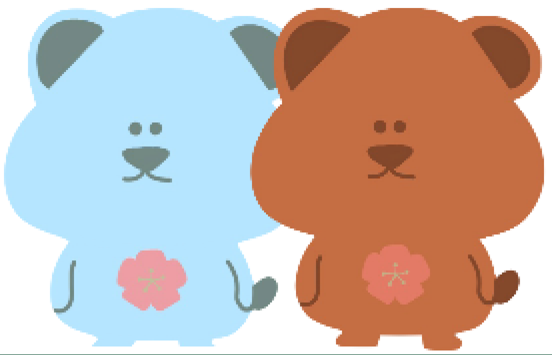
Week4_발표자 : 김나현, 황채원

EWHA EURON

목차



Matrix gradients for Neural Net



#01 Matrix gradients for Neural Net

#1 Chain rule : 함수의 연쇄법칙

$$F = f(g(x))$$

$$\frac{d}{dx}F = f'\big(g(x)\big)g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

#2 NER 모델의 chain rule

$$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial W}$$

$$s = u^T h$$

$$h = f(z)$$

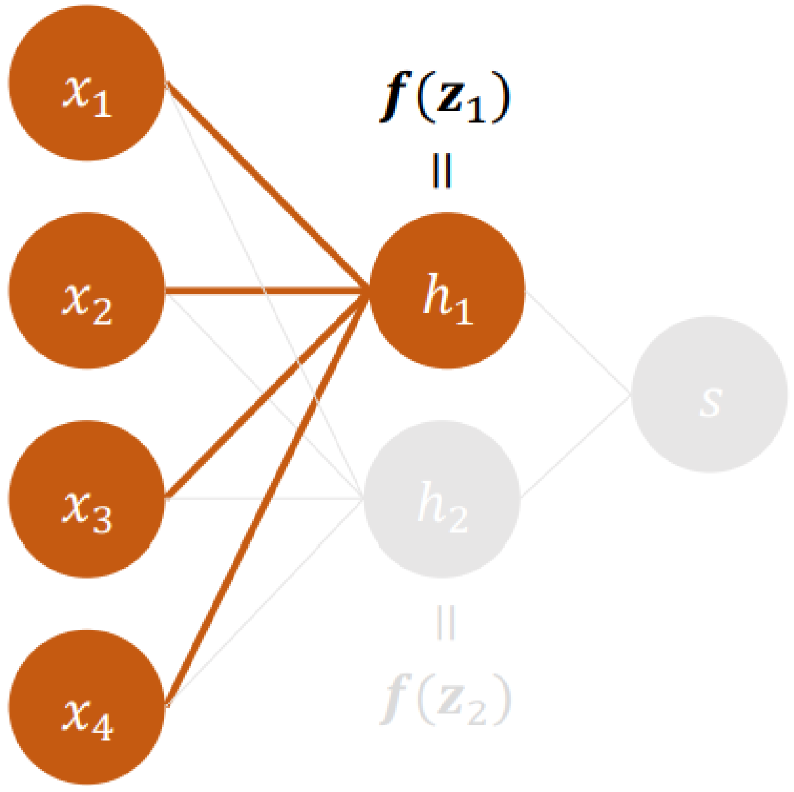
$$z = Wx + b$$

#01 Matrix gradients for Neural Net

Deriving Gradients for Backprop

$$z = Wx$$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$



Weight matrix 의
행의 수 (=2) : 다음 레이어의 뉴런 사이즈
열의 수 (=4) : x의 원소 개수

W의 모든 행에 대해 일반화

$$z_1 = \sum_{k=1}^{4} w_{1k} x_k \qquad \frac{\partial z_1}{\partial w_{1j}} = \frac{\partial \sum_{k=1}^{4} w_{1k} x_k}{\partial w_{1j}} = x_j \qquad \frac{\partial z_i}{\partial w_{ij}} = \frac{\partial \sum_{k=1}^{4} w_{ik} x_k}{\partial w_{ij}} = x_j$$

#01 Matrix gradients for Neural Net

Deriving Gradients for Backprop

$$\frac{\partial s}{\partial W} = \boxed{\frac{\partial s}{\partial h}\frac{\partial h}{\partial z}}\frac{\partial z}{\partial W}$$

$$\delta = \begin{bmatrix}\delta_1\\ \delta_2\end{bmatrix}$$

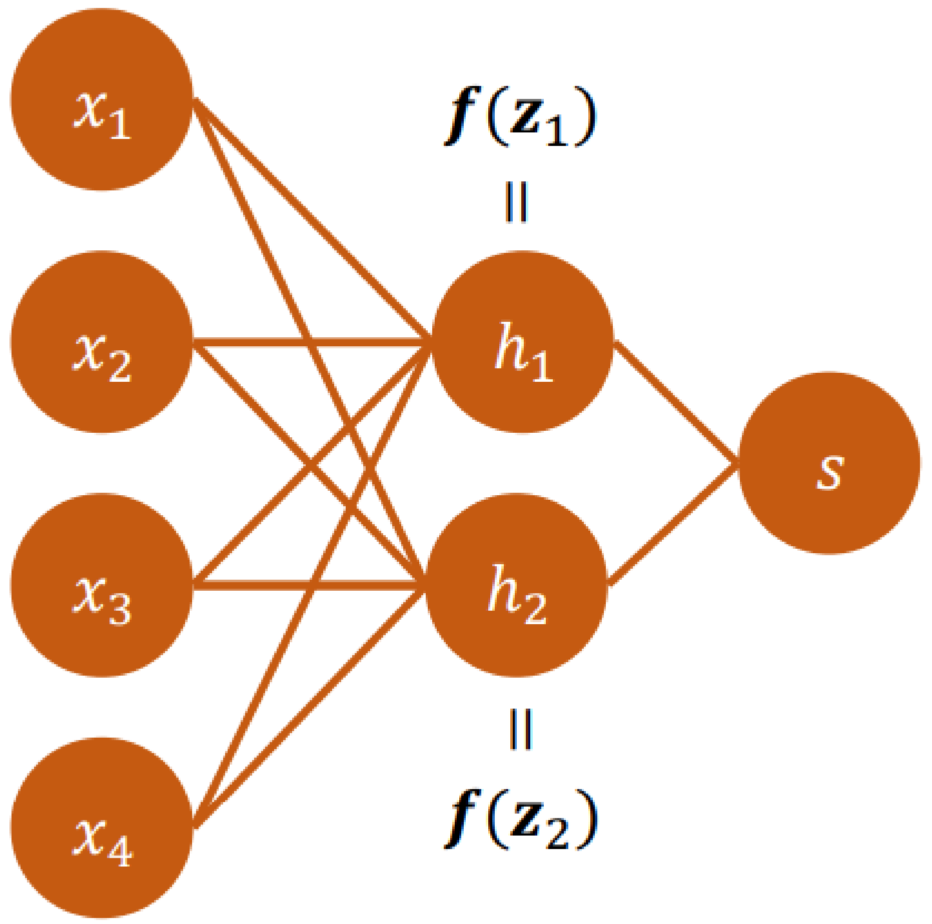
전파된 에러
Z와 같은 차원의 값을 갖는다

$$\frac{\partial s}{\partial W_{ij}} = \delta\frac{\partial z}{\partial W_{ij}} = \sum_{k=1}^{m}\delta_k\frac{\partial z_k}{\partial W_{ij}} = \delta_i x_j$$

$$\frac{\partial s}{\partial W} = \delta x^T$$

행렬의 형태가 δ 와 x 를 외적한 형태

$[n \times m] = [n \times 1][1 \times m]$



#01 Matrix gradients for Neural Net

Deriving Gradients : Tips

#01 변수를 잘 정의하고, 그들의 차원에 주의를 기울여라

#02 Chain rule

#03 softmax 미분 시 correct class 와 incorrect class 를 따로 계산해라

#04 행렬 미분이 헷갈린다면 성분 별 미분부터 시작해라

#05 Shape Convention을 이용해라 : hidden layer의 델타는 hidden layer와 같은 차원을 갖고 있다

#01 Matrix gradients for Neural Net

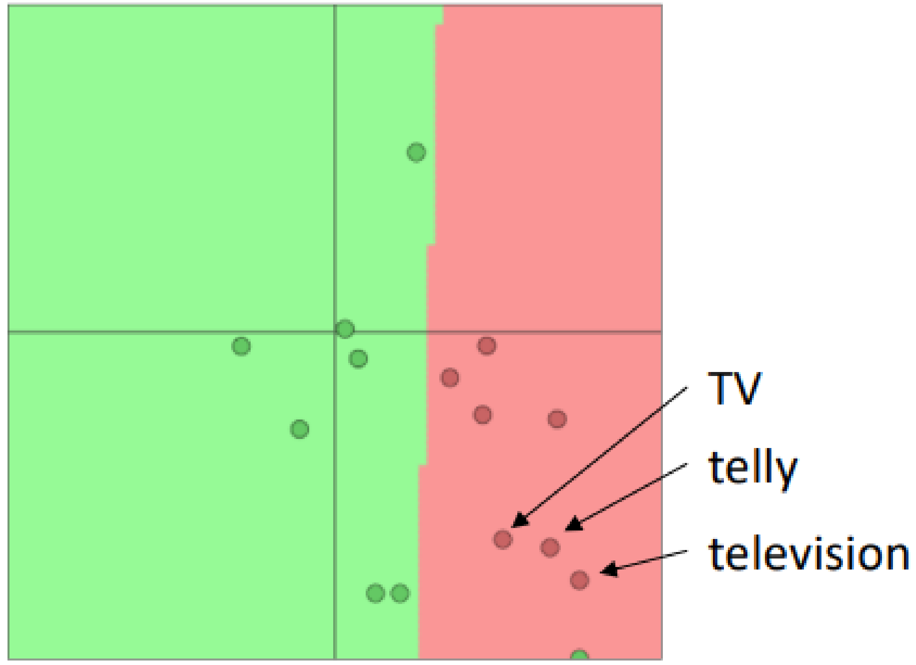
Deriving gradient wrt words

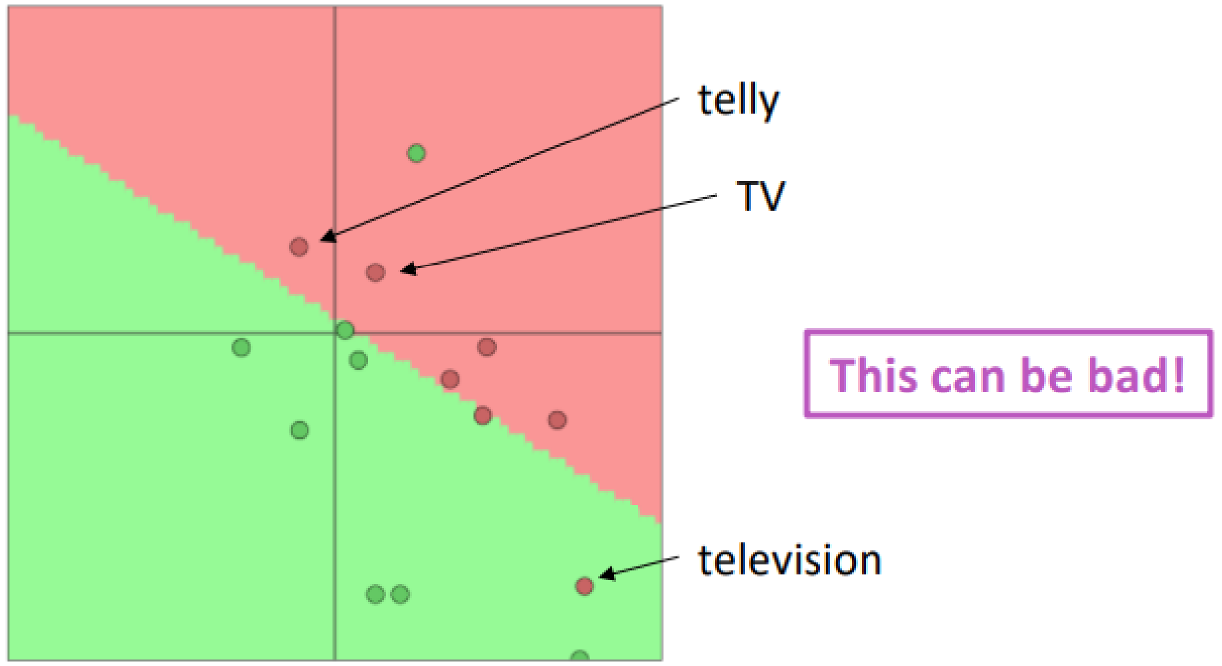
Window의 단어들이 업데이트 -> 단어벡터들이 NER에 적합하도록 변화한다

$$\nabla_x J = W^T \delta = \delta_{window} = \begin{bmatrix} \nabla_{x_{museums}} \\ \nabla_{x_{in}} \\ \nabla_{x_{Paris}} \\ \nabla_{x_{are}} \\ \nabla_{x_{amazing}} \end{bmatrix}$$

#01 Matrix gradients for Neural Net

Pitfall when retraining word vectors





Training data : TV, Telly
Testing data : television
Pre-trained word vectors : TV, Telly, television

Training data에 있는 TV, Telly는 gradient를 받아 업데이트 되지만, Training data에 없는 television은 업데이트 되지 않는다.

변화한 결정경계면에 의해 제대로 분류되지 않은 것을 확인할 수 있다.

#01 Matrix gradients for Neural Net

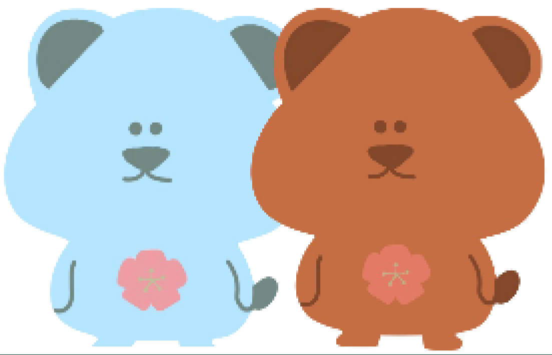
#01 almost always pre-trained word vectors를 이용해라

- Pre-trained data는 이미 많은 학습을 거친 방대한 양의 데이터
- 앞선 예시에서 나타난 훈련 집합 포함 문제 : 훈련 집한 포함 여부에 관계없이 어느 정도 단어 간 관계가 형성되어 있다
- 그러나 데이터가 매우 많다면 (100 millions of words) 랜덤하게 처음부터 학습을 해도 괜찮다

#02 fine-tuning 이 필요한 경우

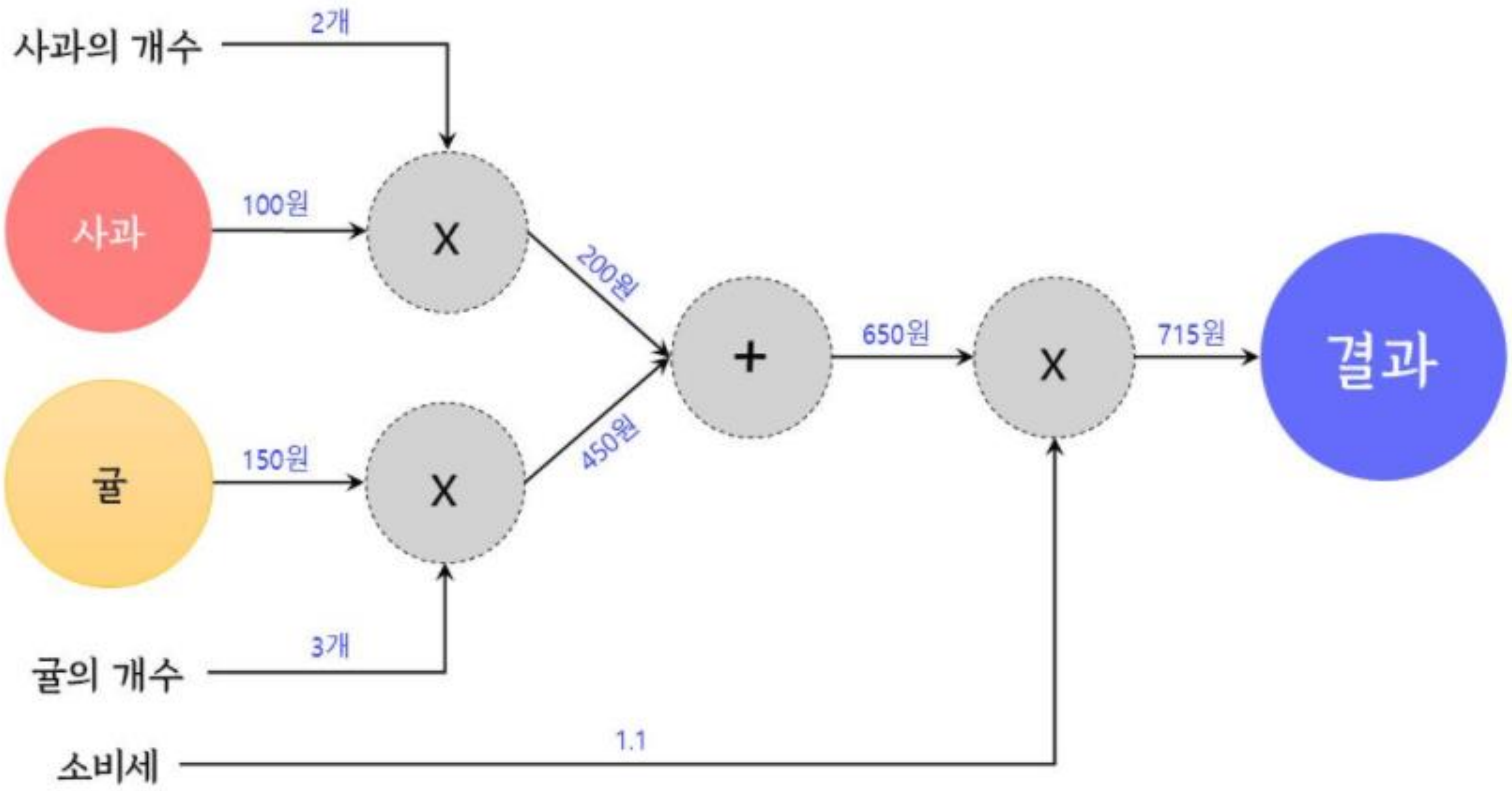
- 훈련 집합이 적으면 fine-tuning하지 말 것. Pre-trained data를 고정시키고 업데이트하지 않는 것이 좋다
- 훈련 집합이 많으면 fine-tuning은 성능 향상에 도움이 된다

Backpropagation and Computation Graphs



#02 Computational Graph

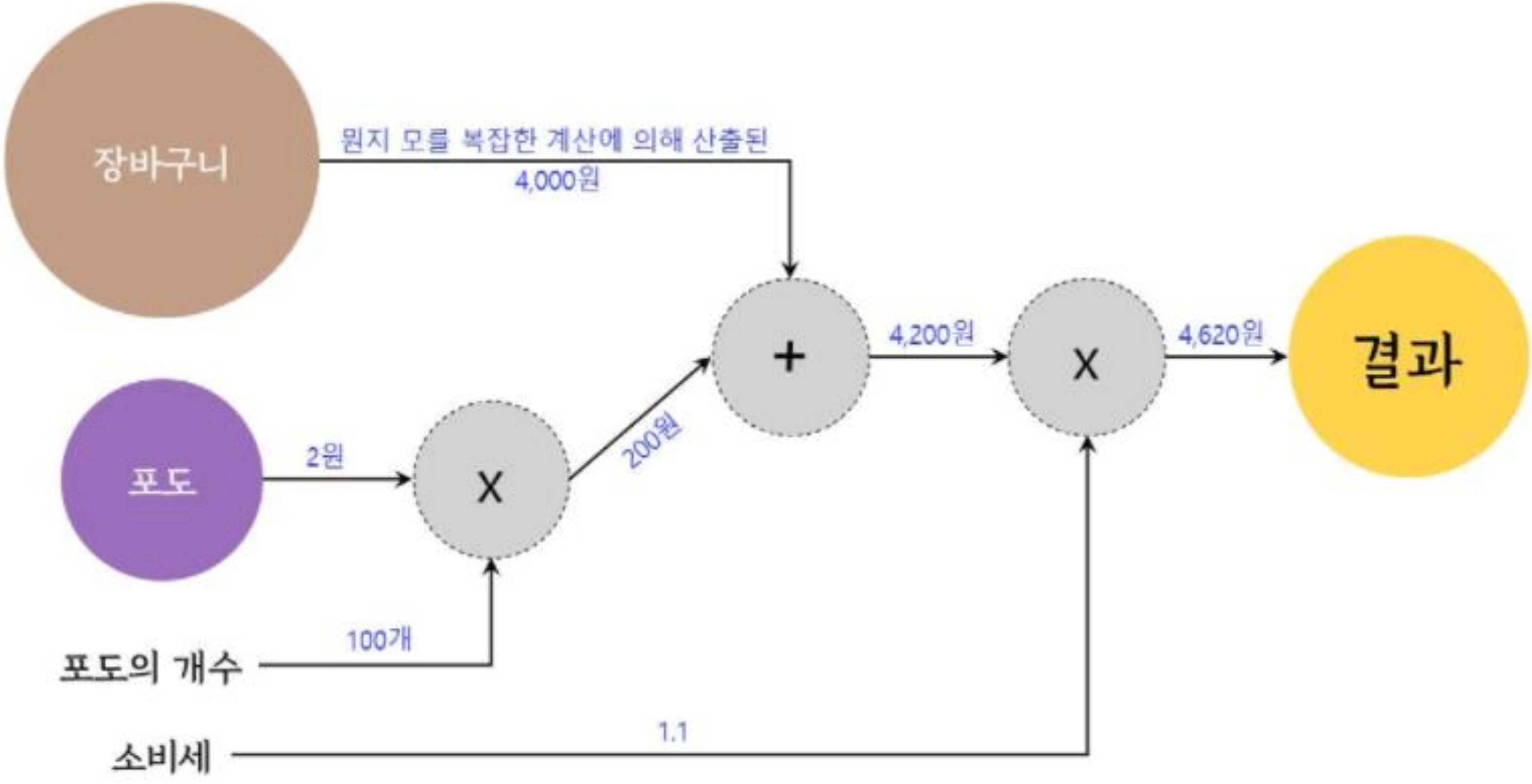
슈퍼에서 사과를 2개, 귤을 3개 샀습니다. 사과는 1개에 100원, 귤은 1개 150원입니다.
소비세가 10%일 때 지불 금액을 구하시오.



#02 Computational Graph

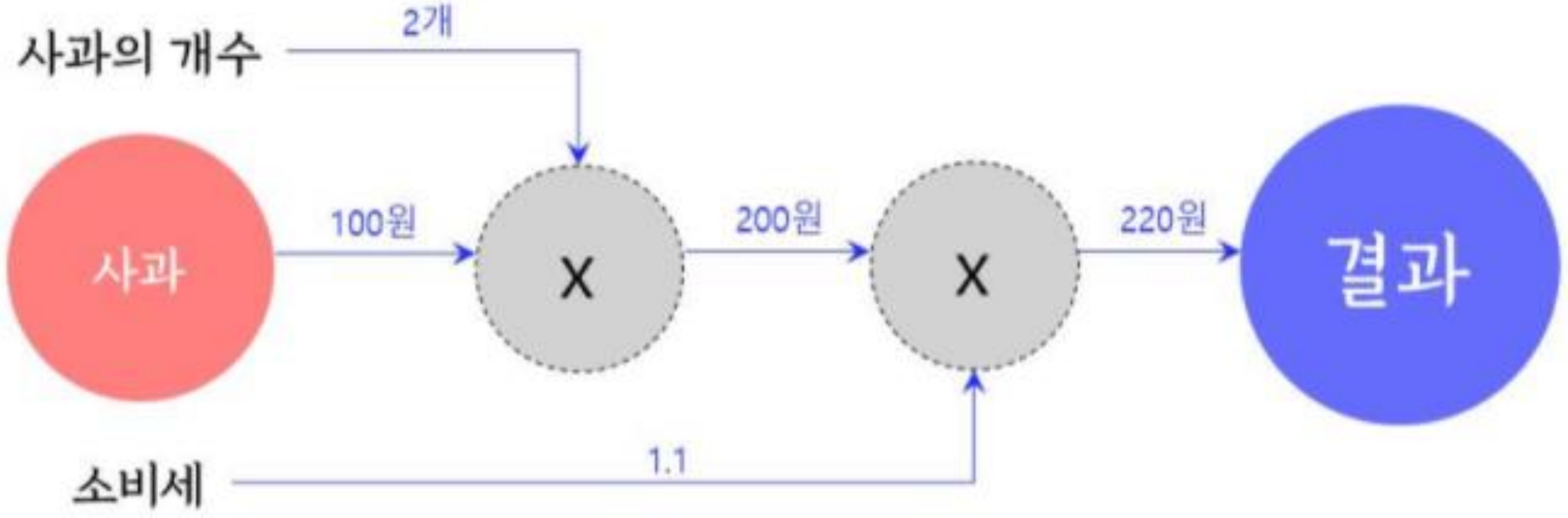
Computational Graph의 장점
: 국소적으로 계산을 따로 할 수 있다.
(전체에서 어떤 일이 벌어지든 상관없이 자신과 관계된 정보만으로 결과를 출력할 수 있다)

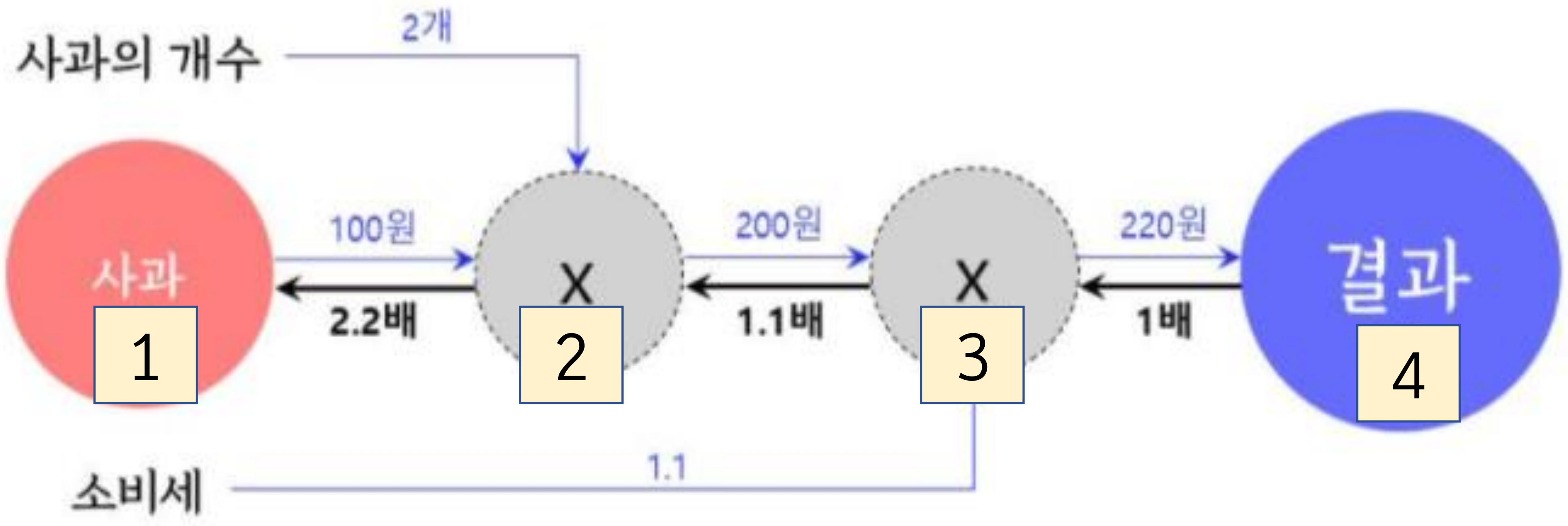
: 역전파(BackPropagation)를 통해 미분을 효율적으로 계산할 수 있다.



https://stellarway.tistory.com/21

#02 Back propagation

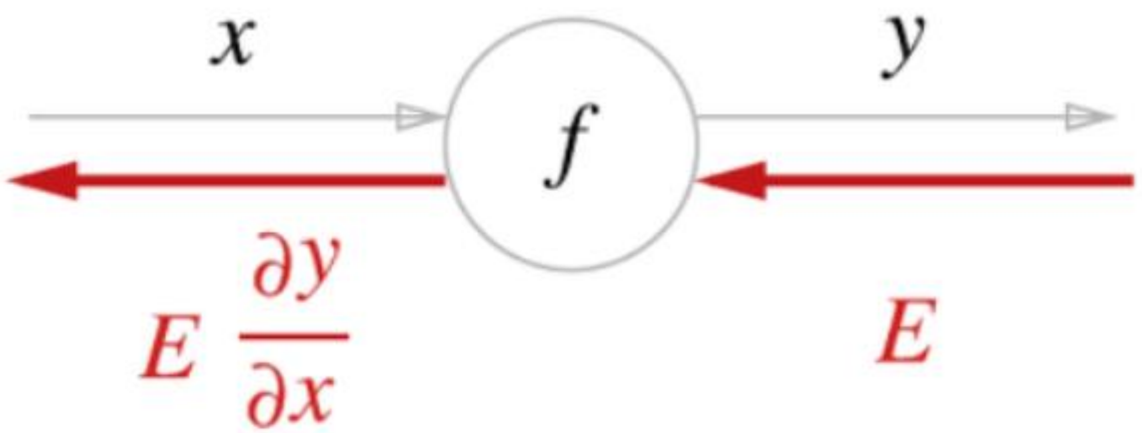


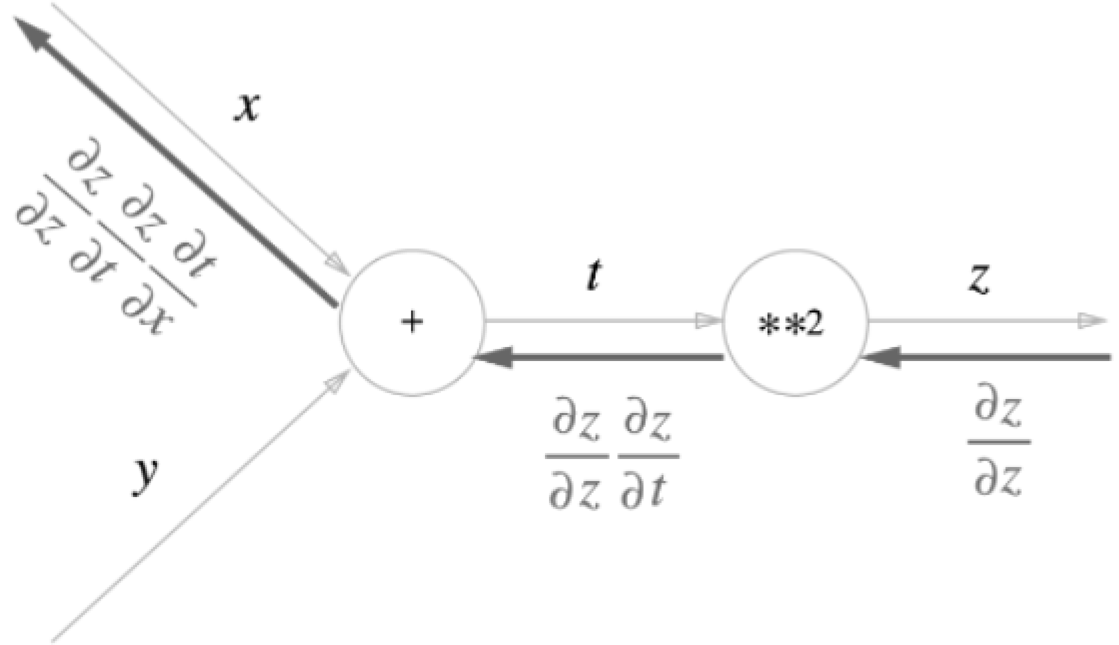


인상된 가격 1원이라는 정보만으로 최종금액이 얼마 올라야 하는지 예측할 수 있다.

#02 Back propagation

y=f(x)의 역전파



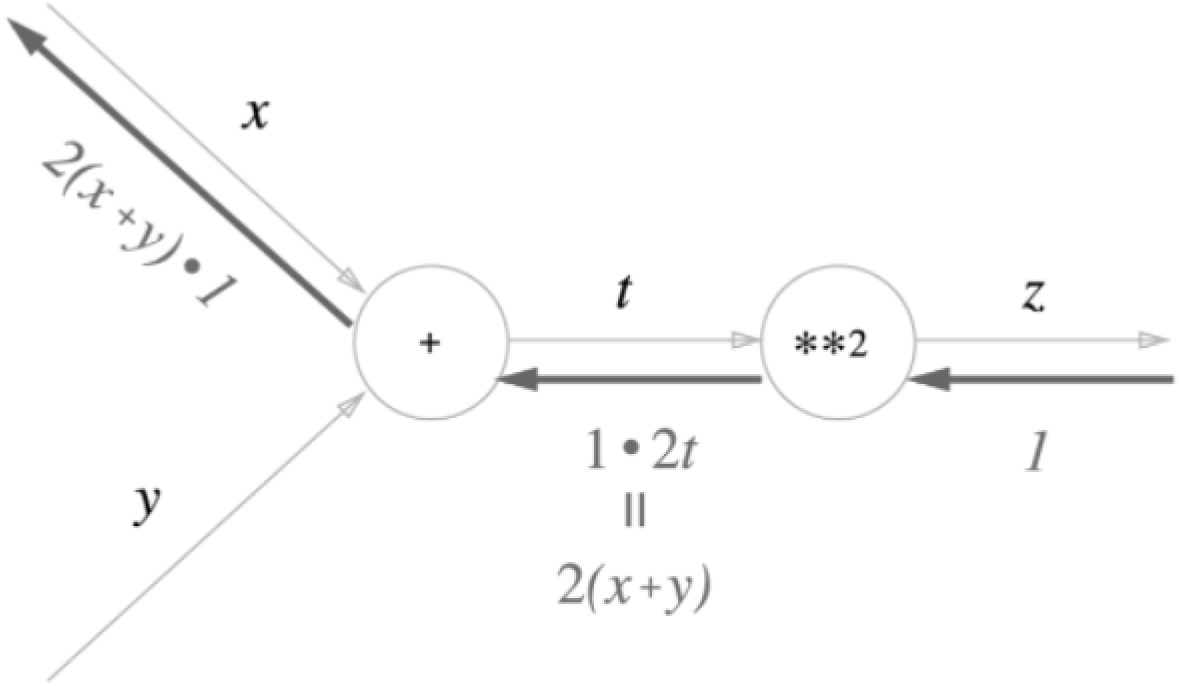


노드로 들어온 입력 신호에 그 노드의 국소적 미분(편미분)을 곱한 후 다음 노드로 전달합니다.
위 그림에서 주목할 것은 맨 왼쪽 역전파입니다. ∂z와 ∂t는 전부 소거되어 'x에 대한 z의 미분'이 됩니다.
즉, 역전파가 하는 일은 연쇄법칙의 원리와 같습니다.

연쇄법칙과 역전파
예시) z = (x+y)^2

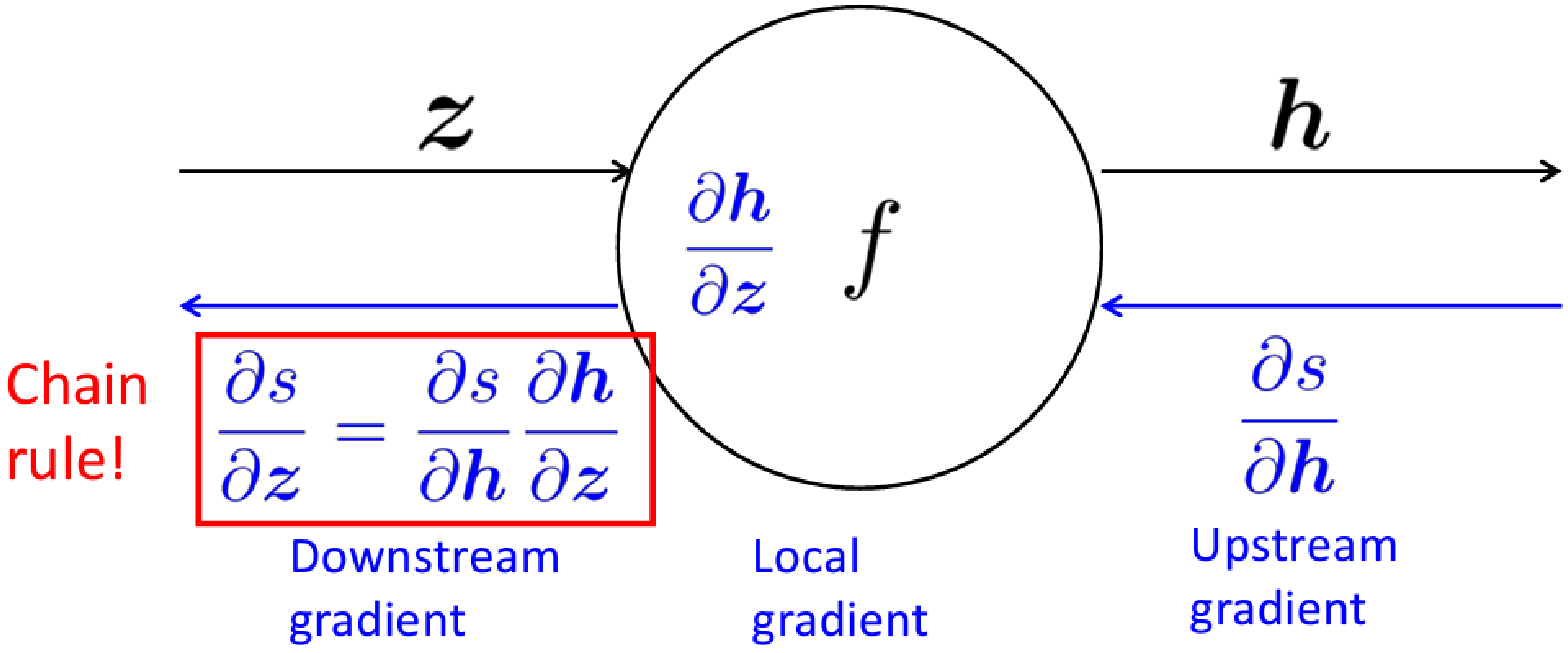
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x+y)$$



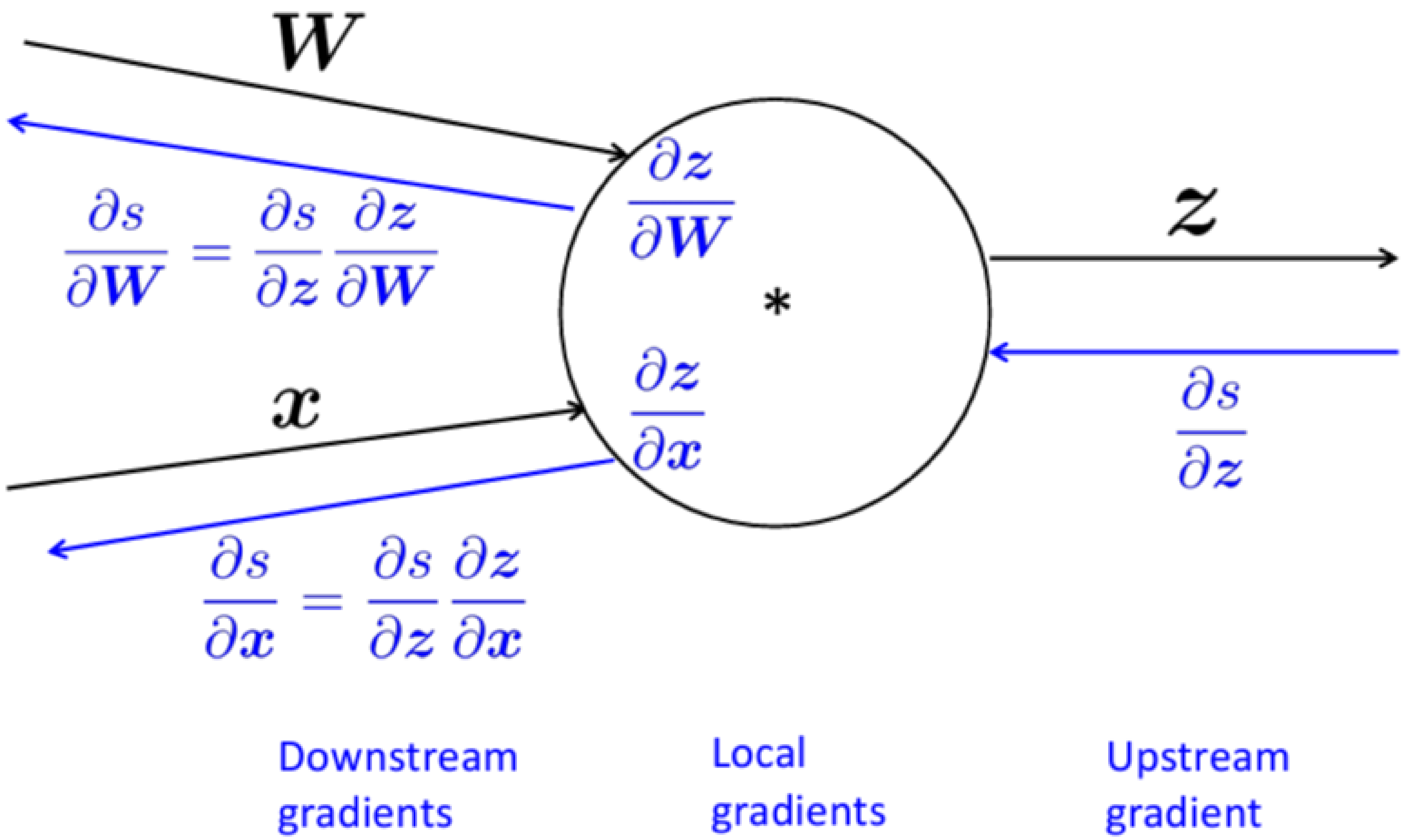
#02 Back propagation

Upstream gradient: 노드의 output에 대한 gradient.
Local gradient : 해당 노드 내에서만 계산되는 gradient.
Downstream gradient : 노드의 input에 있는 변수에 대한 gradient.



#02 Back propagation

- Multiple inputs → multiple local gradients $\boxed{z = \boldsymbol{W}\boldsymbol{x}}$



#02 Numerical Gradient

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

numerical gradient : 수치적 방법 (수식으로 하나하나 계산해서 사용)

analytical gradient : 해석적 방법 (미분 활용)

numerical gradient는 정확하지 않고 느리지만 간편함
analytical gradient는 정확하고 빠르지만 오류가 많이 나올 수 있음

보통은 analytic gradient를 많이 사용하나, 디버깅 및 점검을 할 때는
numerical gradient를 사용하기도 함. (gradient check)

current W:	W + h (third dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	**0**,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

(1.25347 - 1.25347)/0.0001
= 0

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 65 April 11, 2017

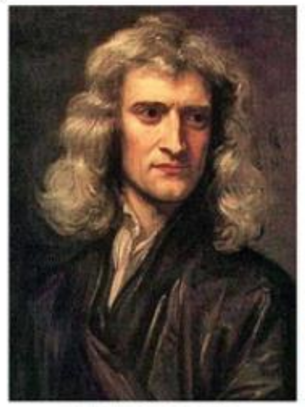
This is silly. The loss is just a function of W:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j\neq y_i} \max(0, s_j - s_{y_i} + 1)$$

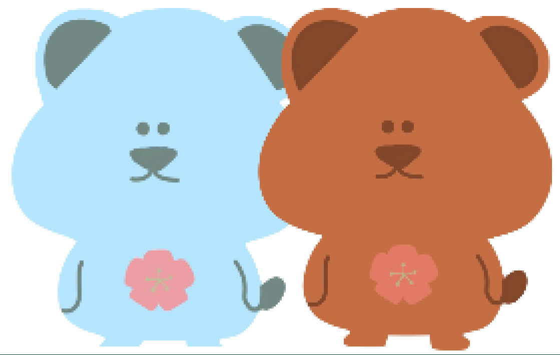
$$s = f(x;W) = Wx$$

want $\nabla_W L$



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 67 April 11, 2017

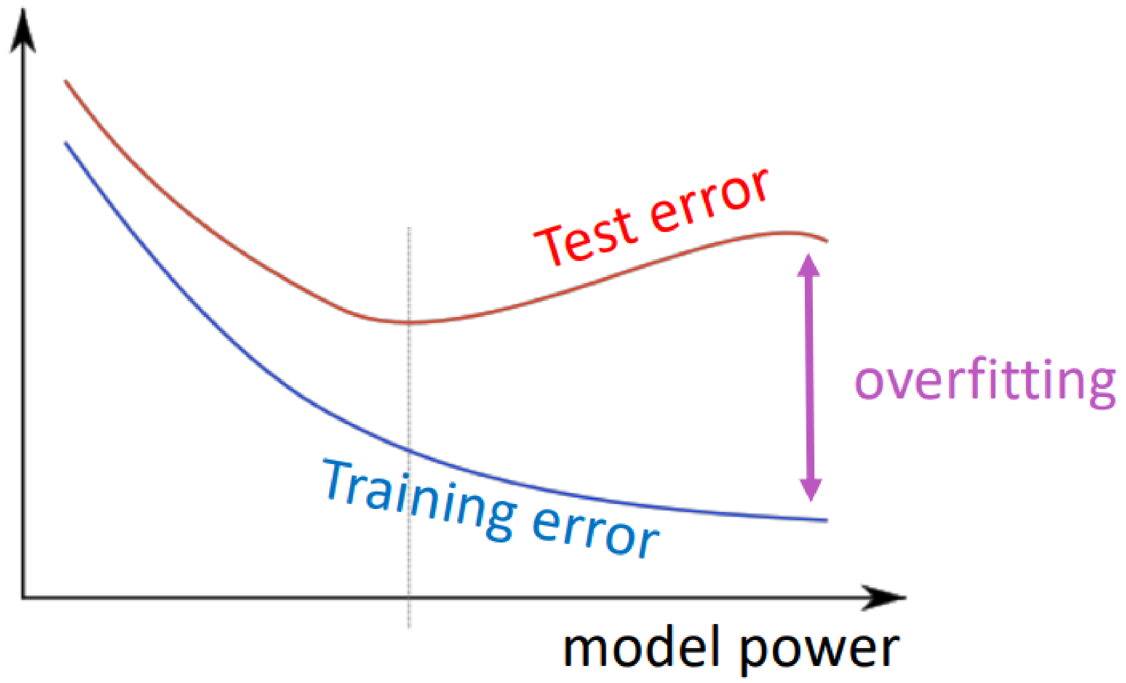
Tips and Tricks for Neural Networks



#03 Tips and Tricks for Neural Networks

#01 Regularization

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) + \lambda \sum_{k} \theta_k^2$$



- 손실함수를 그냥 사용할 경우 훈련 집합에 과적합이 발생할 수 있음
- 훈련집합과 검증집합은 다른 집합이므로, 훈련 집합에 과적합되면 어느 시점부터는 테스트 에러가 증가하는 현상이 발생한다.
- 이를 방지하기 위한 Regularization

#03 Tips and Tricks for Neural Networks

#02 Vectorization

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

For loop을 이용해
word vector를 하나씩 W와 내적

Word vector 집합을 하나의
행렬로 만들어 W와 내적

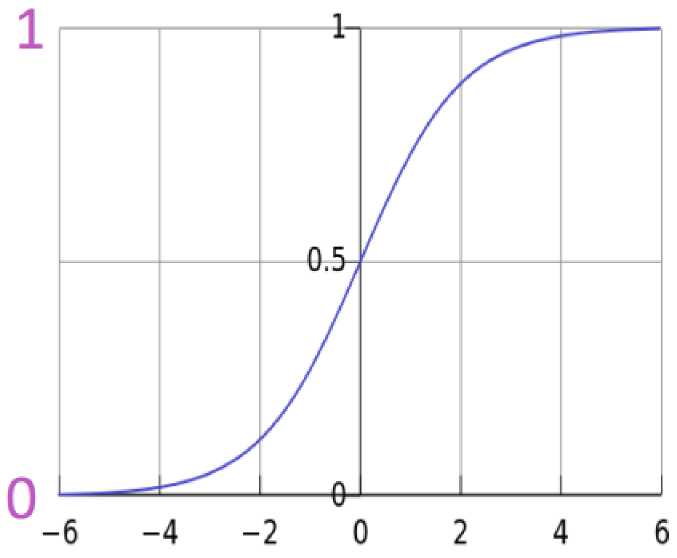
벡터와 행렬을 이용하는 방식이 10배 가량 빠르다!

#03 Tips and Tricks for Neural Networks

#03 non-linearities : the starting points

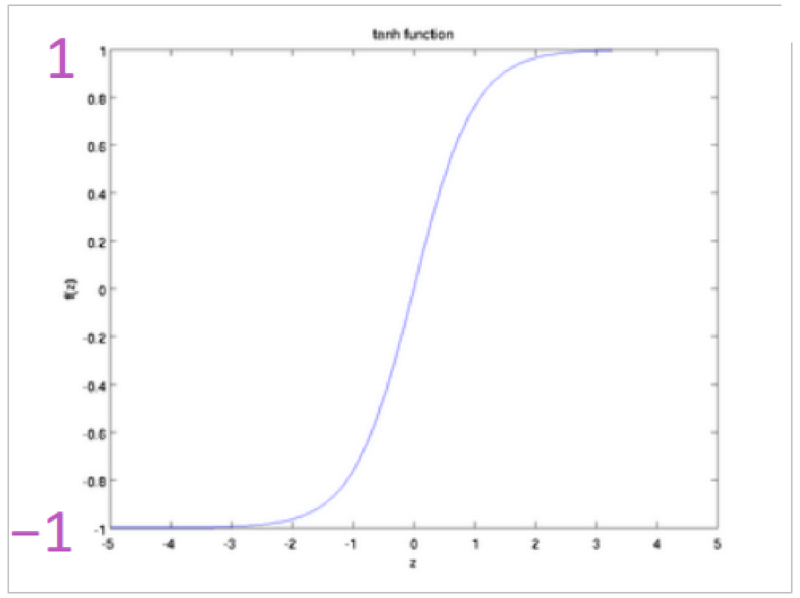
logistic ("sigmoid")

$$f(z) = \frac{1}{1+\exp(-z)}.$$



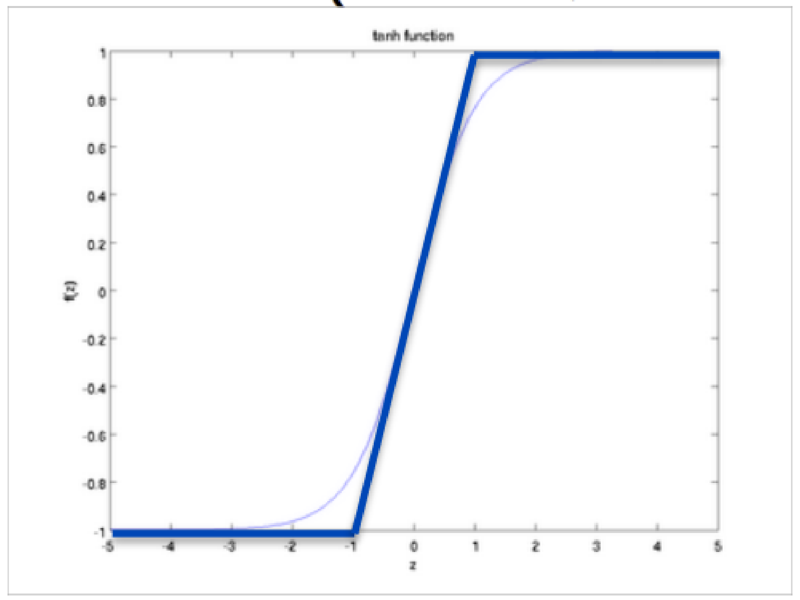
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



- tanh = rescaled logistic
- Logistic과 tanh는 exponential 때문에 계산량이 많아 deep networks에서는 쓰이지 않는다
- Hard tanh : exponential이 제거되어 cheap to compute
- Hard tanh를 더 심플하게 만든 것이 ReLU

#03 Tips and Tricks for Neural Networks

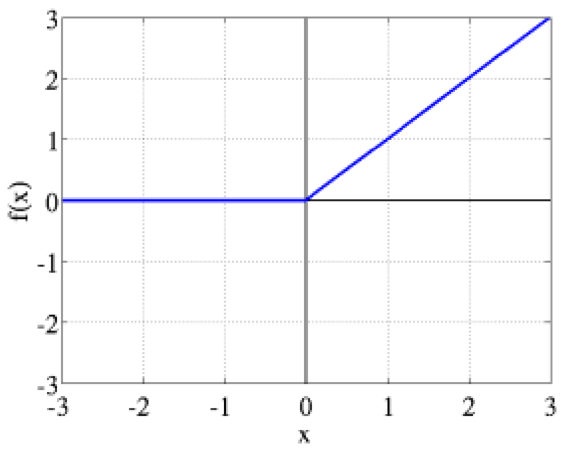
#03 non-linearities : the starting points

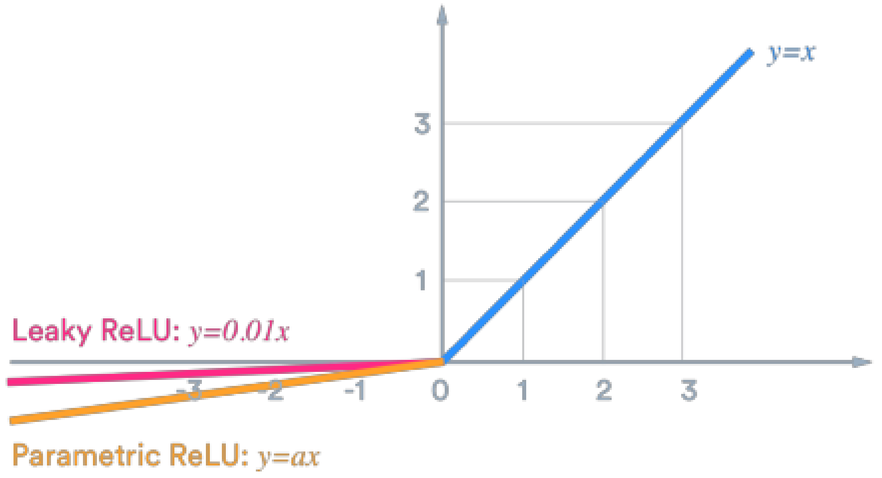
ReLU (rectified linear unit) hard tanh

Leaky ReLU

Parametric ReLU

$$\mathrm{rect}(z) = \max(z, 0)$$





- Deep networks에서의 default choice
- Train quickle]y, perform very well
- Leaky ReLU, Parametric ReLU에는 extra parameter 적용

#03 Tips and Tricks for Neural Networks

#04 Parameter Initialization

- Small random value로 가중치를 초기화 한다
- Hidden layer bias는 0으로 초기화
- 다른 모든 가중치들은 Uniform(-r, r)에서 초기화 (r은 작지도 크지도 않은 수)
- Xavier initialization :
  input layer size, output layer size 고려하여 weight varianc를 조절한다

#03 Tips and Tricks for Neural Networks

#05 Optimizer

- SGD 를 이용해도 최적화가 잘되지만, 더 좋은 결과를 얻기 위해서는 learning rate의 hand-tuning이 필요하다

- 복잡한 신경망을 학습할 때에는 “adaptive” optimizer들의 성능이 좋다
  - Adagrad, RMSprop, Adam, SparseAdam…
  - 이 모델들은 parameter의 민감도에 따라 조절하는, per-parameter learning rates를 이용한다

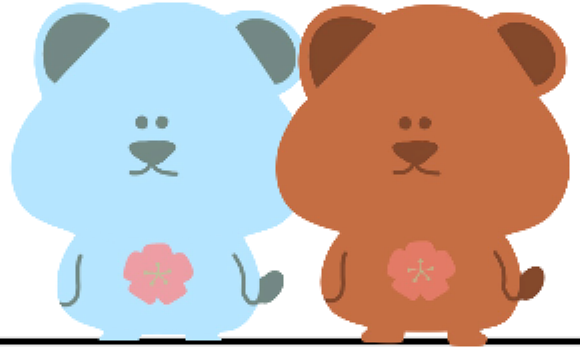
#03 Tips and Tricks for Neural Networks

#06 Learning Rates

- 0.001 정도의 일정한 leaning rate가 일반적임
  - 10의 제곱수를 많이 사용한다
  - 너무 클 경우 모델이 발산한다
  - 너무 작을 경우 모델의 학습 속도가 느리다

- 학습이 진행될 수록 learning rate를 감소하는 방법이 성능이 뛰어나다
  - K epoch마다 learning rate를 반으로 줄인다

THANK YOU



EWHA EURON