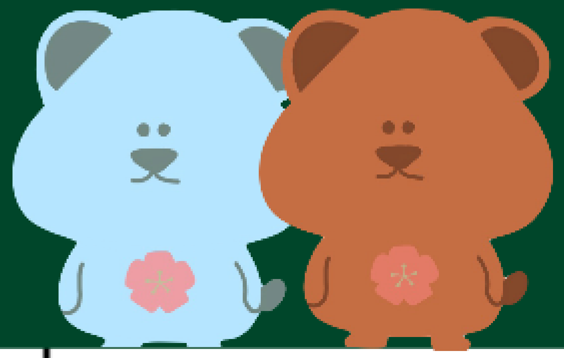


# Named Entity Recognition(NER), Matrix Gradients for Neural Nets and Backpropagation

송민경 송혜준

EWHA EURON

# 목차



EWHA EURON

# 1. 개체명 인식(NER)

# #01 개체명 인식(Named Entity Recognition)

## #1 개체명 인식(Named Entity Recognition)이란?(NER 설명, 예시, 단점)

:문장에서 Location, Person, Organization 등 개체명을 분류하는 방법론

미리 정의해 둔 사람, 회사, 장소, 시간, 단위 등에 해당하는 단어(개체명)를 문서에서 인식하여 추출 분류하는 기법. 추출된 개체명은 인명(person), 지명(location), 기관명(organization), 시간(time) 등으로 분류된다. 개체명 인식(NER)은 정보 추출을 목적으로 시작되어 자연어 처리, 정보 검색 등에 사용된다.

eg.1 *철수[person]는 서울역[location]에서 영희[person]와 10시[time]에 만나기로 약속하였다.*

eg.2 Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

EWHA EURON

# #01 개체명 인식(Named Entity Recognition)

## #2 NER이 어려운 이유

- 개체나 고유명사의 범위를 구하는 것이 쉽지 않음.

 - 텍스트가 '첫 번째 국가 은행이 기부하다' 라고 했을 때, 개체를 '첫 번째 국가 은행(First National Bank)' 으로 할 건지? 아니면 '국가 은행(National Bank)' 이라고 할 것인지?

- 무엇을 '개체' 라고 인식할 건지?

 - '우리 은행' 이라고 했을 때 '나와 너가 일하는 은행' 인지 아니면 상호 '우리 은행' 인지?

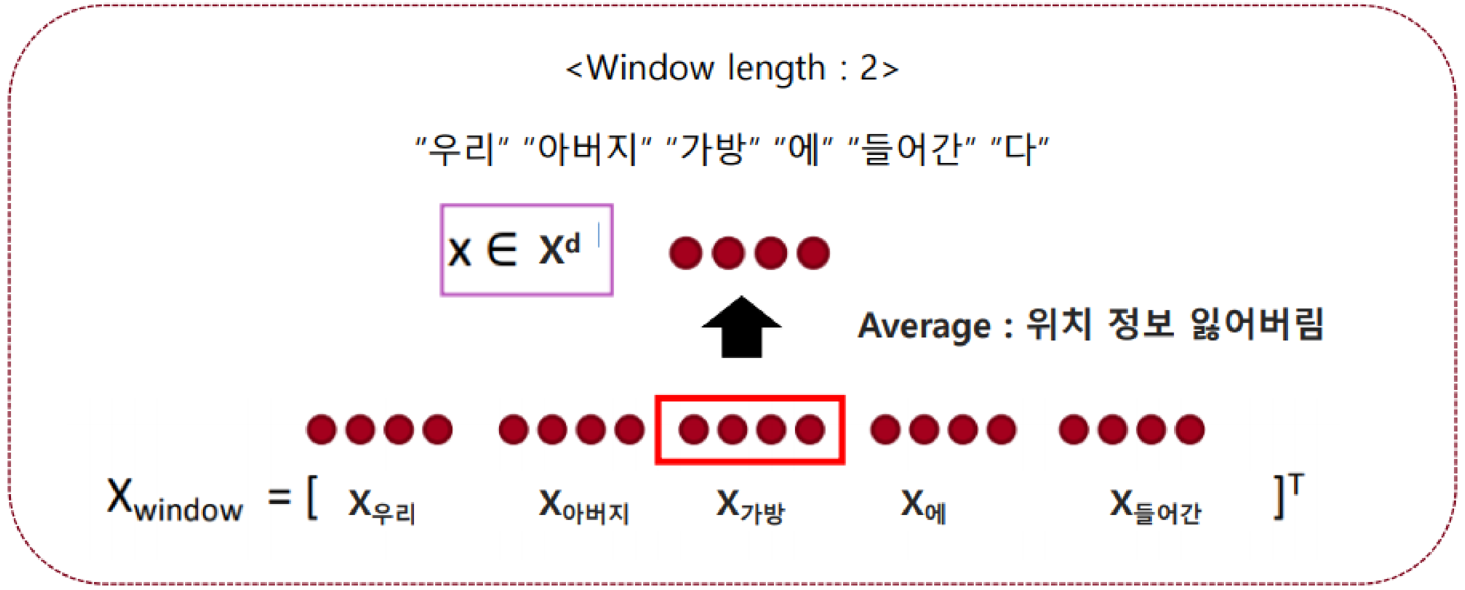
- 알려지지 않았거나 새로운 개체를 분류하기 어렵다
- 개체 분류는 모호하고 맥락에 좌우된다

# #01 개체명 인식(Named Entity Recognition)

## #3 Window Classification

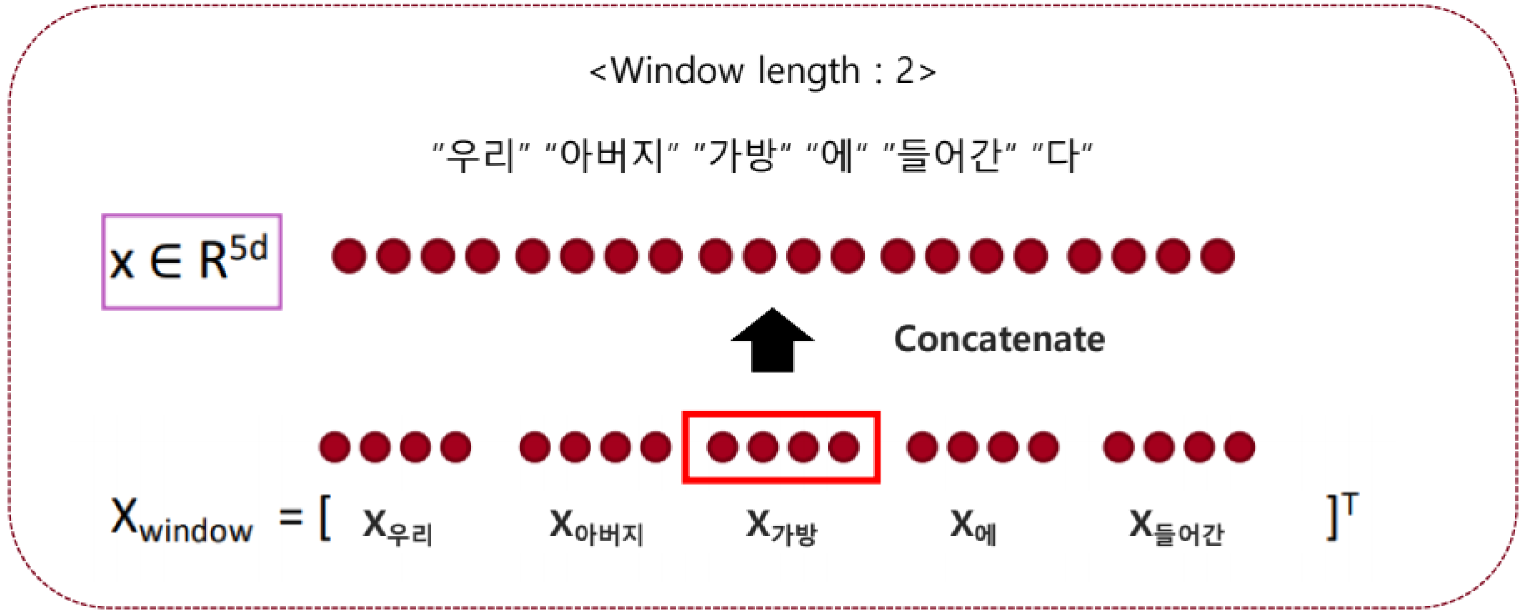
- 모호한 단어들을 문맥까지 고려하는 방법으로, 중심 단어와 주변 단어를 함께 window로 묶어 분류 문제에 활용함

▪ 방법 1 : Average the word vectors in a window




d차원으로 된 각각의 단어들을 더해서
평균을 내주게 되는 average 방법
->위치정보 잃어버린다는 단점

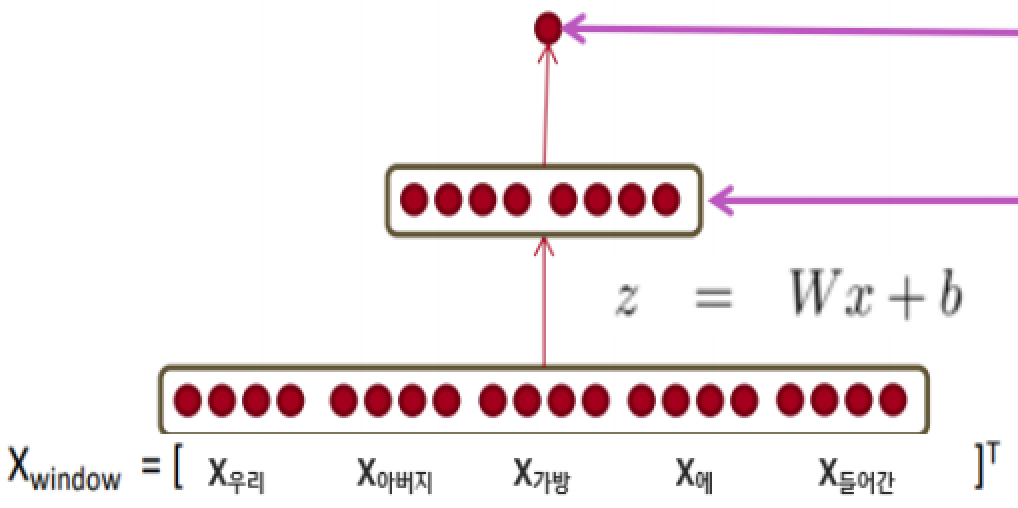
▪ 방법 2 : Concatenate the word vectors in a window




총 5개의 단어를 concatenate하여(즉, 붙여서)
원래가 d차원이었다면 5*d차원으로 concatenate한
데이터를 가지고 테스트하는 방법

# #01 개체명 인식(Named Entity Recognition)

## #4 Simple Window Classification: softmax




$$s = U^T a$$

Non-linear interactions

$$a = f(z)$$

$$z = Wx + b$$

$$x \in \mathbb{R}^{20\times1}, W \in \mathbb{R}^{8\times20}, U \in \mathbb{R}^{8\times1}$$

$$\boldsymbol{w}_i^k = (w_{i1}^k, w_{i2}^k, \cdots, w_{id}^k)^T$$

$$W^k = (\boldsymbol{w}_1^k \quad \boldsymbol{w}_2^k \quad \cdots \quad \boldsymbol{w}_h^k)^T$$

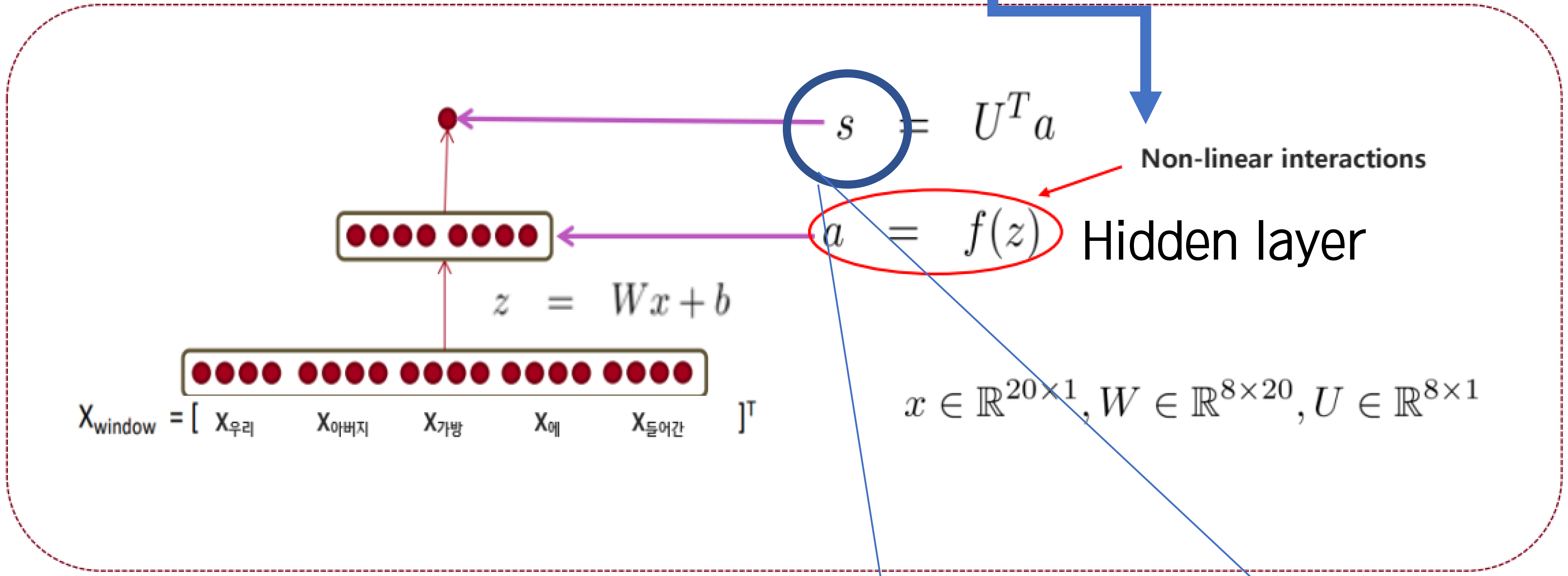
$$W^k = \begin{pmatrix} w_{11}^k & w_{21}^k & \cdots & w_{i1}^k & \cdots & w_{h1}^k \\ w_{12}^k & w_{22}^k & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ w_{1j}^k & & & w_{ij}^k & & w_{hj}^k \\ \vdots & & & & \ddots & \vdots \\ w_{1d}^k & \cdots & \cdots & w_{id}^k & \cdots & w_{hd}^k \end{pmatrix}^T$$

$$W^k \boldsymbol{x} + \boldsymbol{b}^k = \underbrace{\begin{pmatrix} w_{11}^k & \cdots & w_{1d}^k \\ \vdots & \ddots & \vdots \\ w_{h1}^k & \cdots & w_{hd}^k \end{pmatrix}}_{h*d} \underbrace{\boxed{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}}}_{d*1} + \underbrace{\begin{pmatrix} b_1^k \\ b_2^k \\ \vdots \\ b_h^k \end{pmatrix}}_{h*1}$$

Resulting vector $x_{window}$ = $\boxed{x \in R^{5d}}$ , a column vector!

# #01 개체명 인식(Named Entity Recognition)

## #4 Simple Window Classification: softmax




3-layer neural net의 목적!
-> 중심단어(가방)가 장소인지 사람인지 분류하고 싶은 것(NER)
-> 따라서 중심단어(가방)가 장소이면 높은 score를, 사람이면 낮은 score가 도출되어야 함.
-> (True window/Corrupt window)인지 분류하고 싶은 것

# #01 개체명 인식(Named Entity Recognition)

## #4 Simple Window Classification: softmax

-> Output(S)을 확률Probability로 표현

$$p(y|\mathbf{s}) = \frac{\exp(\mathbf{s})}{\sum_{c=1}^{C}\exp(\mathbf{s})} = \mathrm{softmax}(\mathbf{s})$$

$X = X_{window}$

predicted model
output probability

$$\boxed{\hat{y}_y} = p(y|x) = \frac{\exp(\boxed{W_y.x})}{\sum_{c=1}^{C}\exp(W_c.x)}$$

- With cross entropy error as before:

가중치 W 업데이트

same

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

EWHA EURON

# #01 개체명 인식(Named Entity Recognition)

## #5 The Max-margin loss

-> 정답($S$)과 오답($Sc$) 사이의 거리를 최대로 만들어주는 margin을 찾는 것
-> 정답과 오답 사이의 차이가 k 이상일 경우 loss를 0으로 만듦. (= 정답으로 봄)

$$\text{minimize } J = \max(1 + s_c - s, 0)$$

k=1

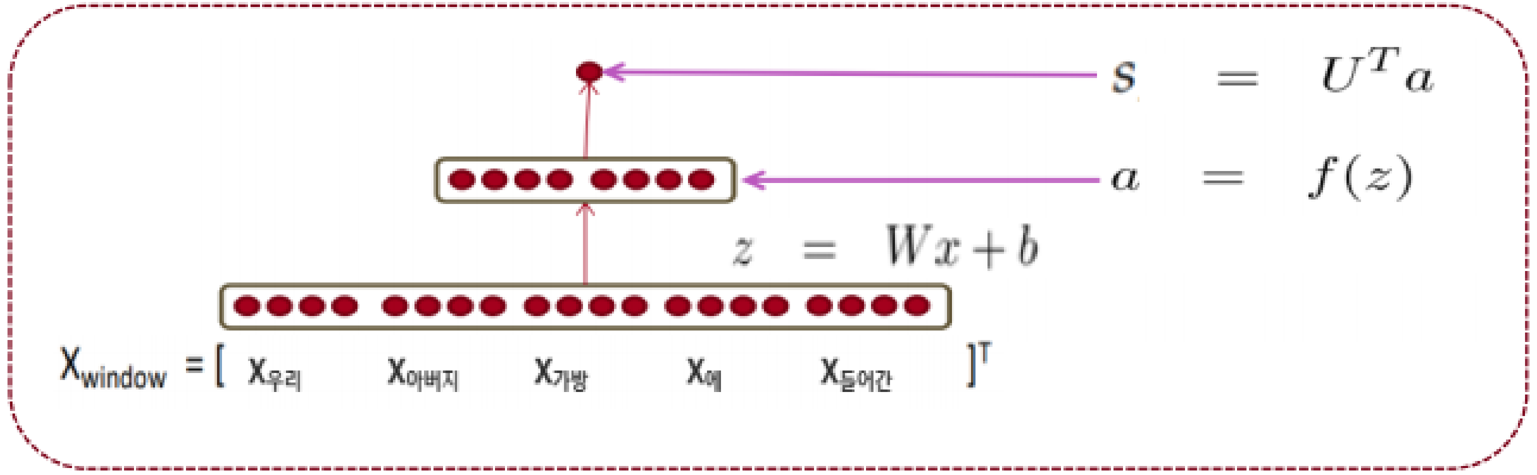
"우리" "아버지" "가방" "에" "들어간" "다"

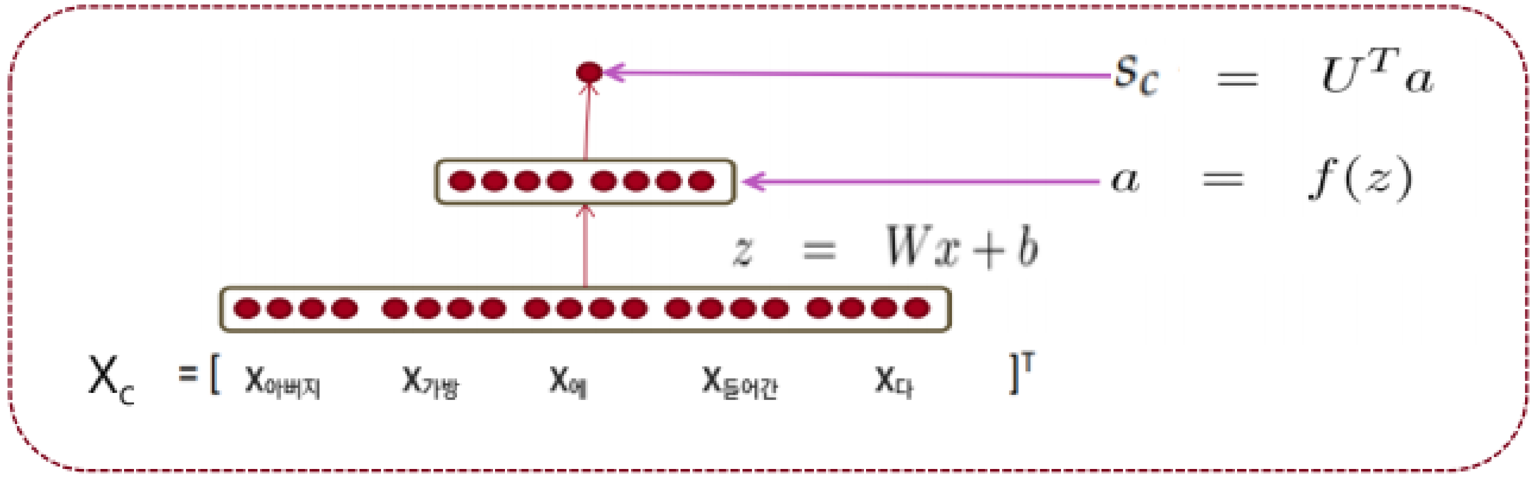
- S : True window's score

  $X_{window}$ = [$X_{우리}$ $X_{아버지}$ $\boxed{X_{가방}}$ $X_{에}$ $X_{들어간}$ ]$^T$

- $S_C$ : Corrupt window's score

  $X_c$ = [ $X_{아버지}$ $X_{가방}$ $\boxed{X_{에}}$ $X_{들어간}$ $X_{다}$ ]$^T$



# #01 개체명 인식(Named Entity Recognition)

#5 The Max-margin loss

-> 정답($S$)과 오답($Sc$) 사이의 거리를 최대로 만들어주는 margin을 찾는 것
-> 정답과 오답 사이의 차이가 k 이상일 경우 loss를 0으로 만듦. (= 정답으로 봄)

$$\text{minimize } J = \max(1 + s_c - s, 0)$$

- $s_c = U^T f(Wx_c + b)$ , Corrupt window's score

  $s = U^T f(Wx + b)$ , True window's score

- $(S_C - S) > 1$ : 학습시킨다

  - Positive margin $\Delta$ = 1

    -> Margin이 없으면 Optimization objective가 risky 함

'Location' 이 있는 window와 아닌 window의 구분이 모호해질 수 있기 때문

# #01 개체명 인식(Named Entity Recognition)

**#5 The Max-margin loss**

-> 정답($S$)과 오답($Sc$) 사이의 거리를 최대로 만들어주는 margin을 찾는 것
-> 정답과 오답 사이의 차이가 k 이상일 경우 loss를 0으로 만듦. (= 정답으로 봄)

**Word2Vec과 유사점**

-> Window가 corpus를 따라 움직이며 모든 위치에 대해 학습시킴
-> Negative sampling: "무관한 단어들에 대해서는 weight를 업데이트하지 않아도 된다 "

# #01 개체명 인식(Named Entity Recognition)

## #6 Gradient descent: backpropagation과 연관

Learning rate란? -> step size: 다음 지점을 결정함

- **Gradient descent algorithm**

가중치 업데이트

$$\theta^{new} = \theta^{old} - \boxed{\alpha} \nabla_\theta J(\theta)$$

**Learning late** Backpropagation 이용해서 loss function 최소화!

- 함수 (J)에 대한 각 Parameter($\theta$)의 미분값 요구 → 각 parameter가 loss에 미치는 영향(기여도)을 구함

$\boxed{\nabla_\theta J(\theta)}$ → **Backpropagation algorithm 으로 구함**

- Neural network에서 Parameter($\theta$) 개수 多

# 2. Matrix Gradients

# #02 Matrix Gradients for Neural Nets

## #1 Chain rule과 Jacobian

- Chain rule : 함수의 연쇄법칙

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

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

- NER 모델의 chain rule

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

$$s = u^T h$$

$$h = f(z)$$

$$z = Wx + b$$

# #02 Matrix Gradients for Neural Nets

## #1.1 Chain rule의 행렬로의 확장 -> Jacobian

- 1Xn 행렬을 편미분한다면

- Given a function with 1 output and *n* inputs

$$f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \boldsymbol{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

- mXn 행렬을 편미분한다면

- Given a function with ***m* outputs** and *n* inputs

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n)]$$

- It's Jacobian is an ***m* x *n* matrix** of partial derivatives

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \qquad \boxed{\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}}$$

EWHA EURON

# #02 Matrix Gradients for Neural Nets

## #1.2 Jacobian 예제

### Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

Stanf

### Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I} \quad \text{(Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

# #02 Matrix Gradients for Neural Nets

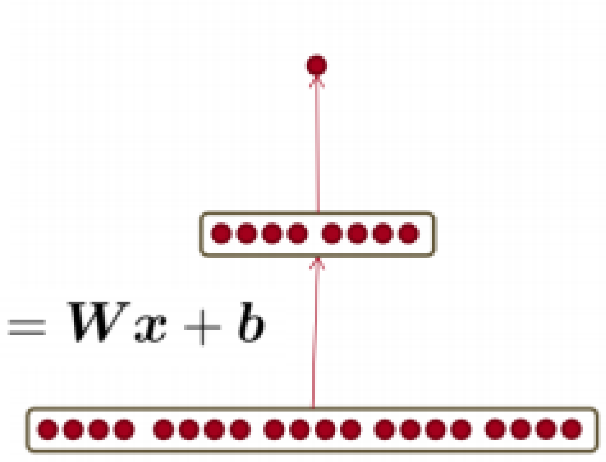
## #1.3 Jacobian 예제_ 3-layer neural net case

- **Jacobian matrix**

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{x}$ (input)




- 입력층

$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \Rightarrow \boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}}$

$\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W} \Rightarrow \boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}^T}$

$(\boldsymbol{W} \in \mathbb{R}^{n \times m})$

- 중간층

$\boldsymbol{h} = f(\boldsymbol{z}) \Rightarrow \boxed{\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))}$

$$\begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix}$$

- 출력층

$s = \boldsymbol{u}^T \boldsymbol{h} \Rightarrow \boxed{\frac{\partial s}{\partial \boldsymbol{h}} = \boldsymbol{u}^T}$

# #02 Matrix Gradients for Neural Nets

## #1.3 Jacobian 예제_ 3-layer neural net case

- **Jacobian matrix**

$$\frac{\partial s}{\partial \boldsymbol{u}} = \boldsymbol{h}^T$$

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

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boxed{\frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

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

전파된 에러
Z와 같은 차원의 값을 갖는다

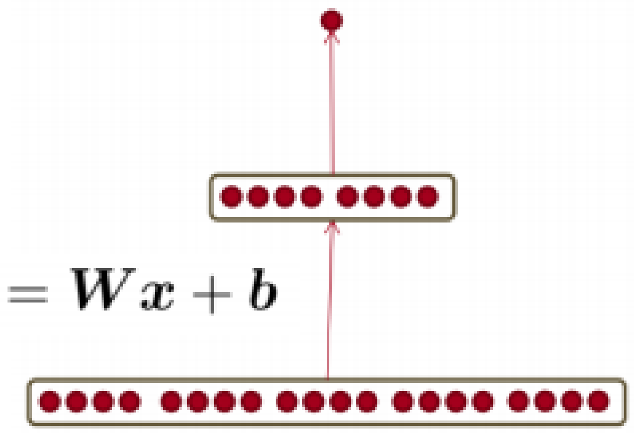
$\boldsymbol{\delta}$ : 중복된 연산 피할 수 있음

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

$\boldsymbol{x}$ (input)



# #02 Matrix Gradients for Neural Nets

## #1.3 Jacobian 예제_ 3-layer neural net case

- **Jacobian matrix**

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

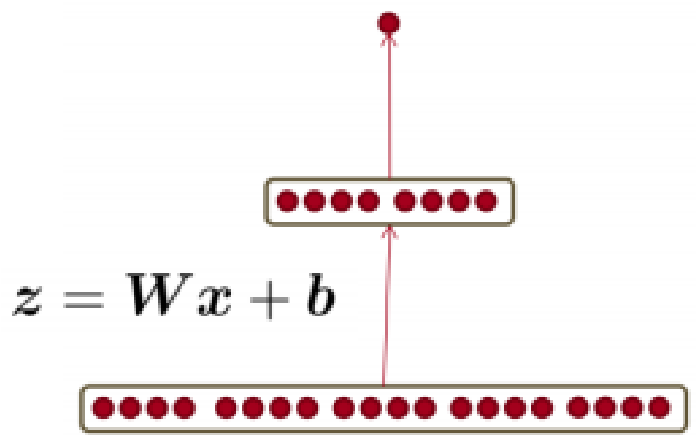
( $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ )

$$\underbrace{\boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))}_{\boldsymbol{\delta}}$$

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{x}$ (input)




$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

Jacobian $\frac{\partial s}{\partial W}$ 가 **1 by nm** 형태??

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

행렬의 형태가 $\delta$ 와 $x$ 를 외적한 형태

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

# #02 Matrix Gradients for Neural Nets

## #1.3 Jacobian 예제_ 3-layer neural net case

- **Jacobian matrix**

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

$(\boldsymbol{W} \in \mathbb{R}^{n \times m})$

$\boldsymbol{\delta}$

[ 1 X n ]

[ n X m X n ]

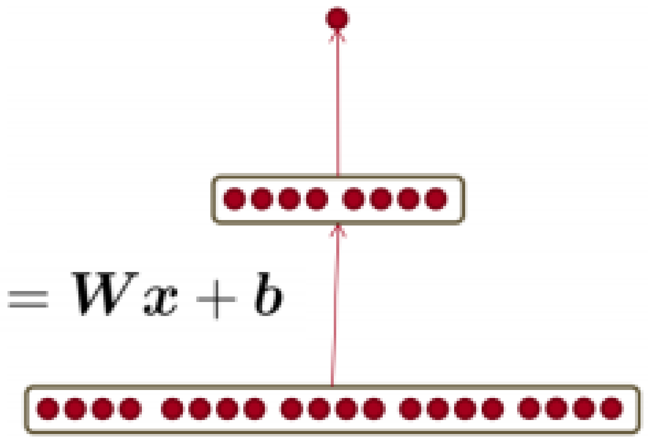
Jacobian $\frac{\partial s}{\partial \boldsymbol{W}}$ 가 **1 by nm** 형태??

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

$\boldsymbol{x}$ (input)



$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{x}^T$$

[ 1 X m ]

# #02 Matrix Gradients for Neural Nets

## #2 Deriving Gradients tip

#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

EWHA EURON

# #02 Matrix Gradients for Neural Nets

## #3 matrix -> word로 넘어간다면?

Window의 단어들이 업데이트 -> 단어벡터들이 NER에 적합하도록 변화

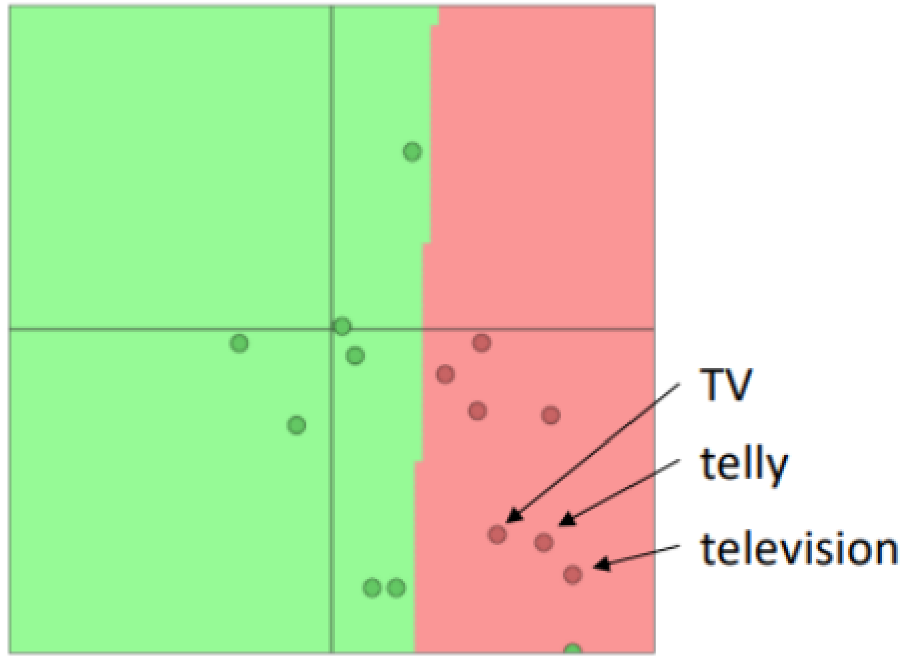
$$\nabla_x J = W^T \delta = \delta_{window} = \begin{bmatrix} \nabla_{x_{museums}} \\ \nabla_{x_{in}} \\ \nabla_{x_{Paris}} \\ \nabla_{x_{are}} \\ \nabla_{x_{amazing}} \end{bmatrix}$$

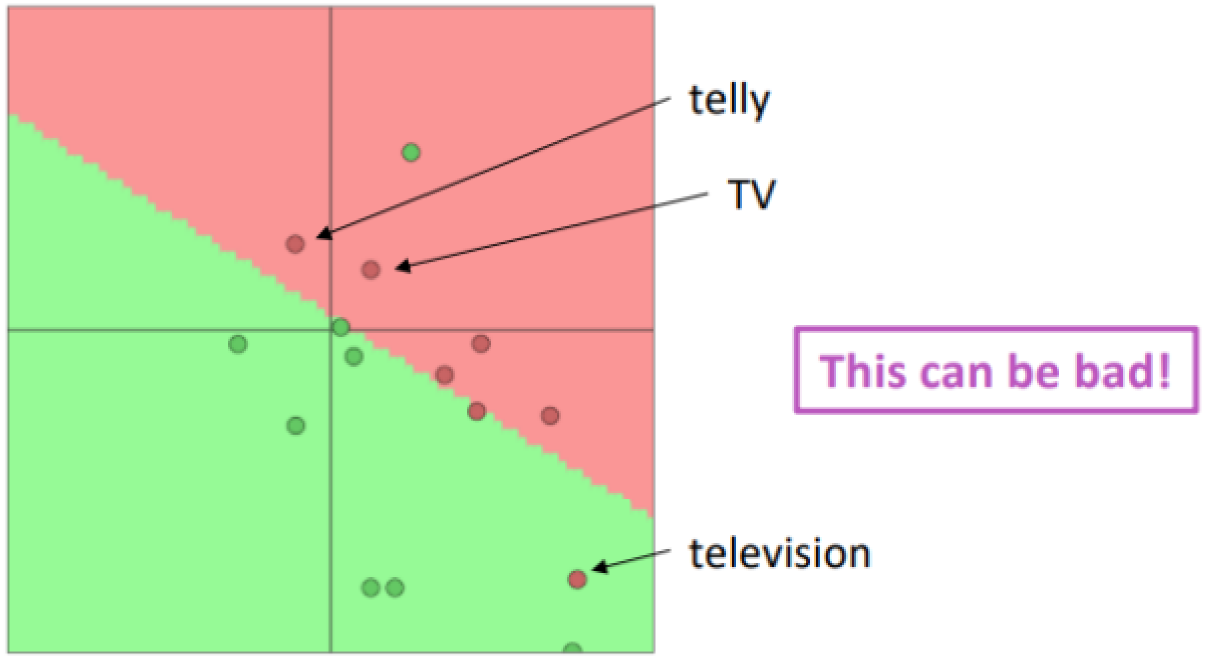
지금까지는 W(matrix)에 대한 미분 확인
-> x 각각의 window 값이 어떻게 gradient 받는지!

즉, Window에 등장한 단어들이 update 되고, 단어벡터들이 task에 더 도움이 되게끔 변화

# #02 Matrix Gradients for Neural Nets

## #4 A 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.

# #02 Matrix Gradients for Neural Nets

## #4 A pitfall when retraining word vectors

- pre-trained vs retraining vs fine-tune
  - pre-trained : 이미 사전에 학습되어 있는 것을 가져다 쓰는 것
  - fine-tuning : pretrained 된 모델을 활용해, task에 맞게 조금 고쳐서 학습시키는 것
  - retraining : data를 새로 가져와서, train 과정부터 다시 학습시키는 것

# #02 Matrix Gradients for Neural Nets

## #5 Fine tune을 해야하는 case

#01 almost always pre-trained word vectors를 이용할 것

- Pre-trained data는 방대한 양의 이미 많이 학습된 데이터
- 앞선 예시 (TV, telly, television) 에서 나타난 바와 같이 훈련 집합 포함 문제 덜함 : 훈련 집합 포함 여부에 관계없이 어느 정도 단어 간 관계가 형성되어 있음
- 그러나 데이터가 매우 많다면 (1억 개 이상) 랜덤하게 처음부터 학습을 해도 무관

#02 fine-tuning 이 필요한 경우

- 훈련 집합이 적으면(10만 개 미만) fine-tuning하지 말 것.
(Pre-trained data를 고정시키고 업데이트하지 않는 것이 좋음)
- 훈련 집합이 많으면(100만 개 이상) fine-tuning은 성능 향상에 도움이 됨

# 3. Computation Graphs and Backpropagation

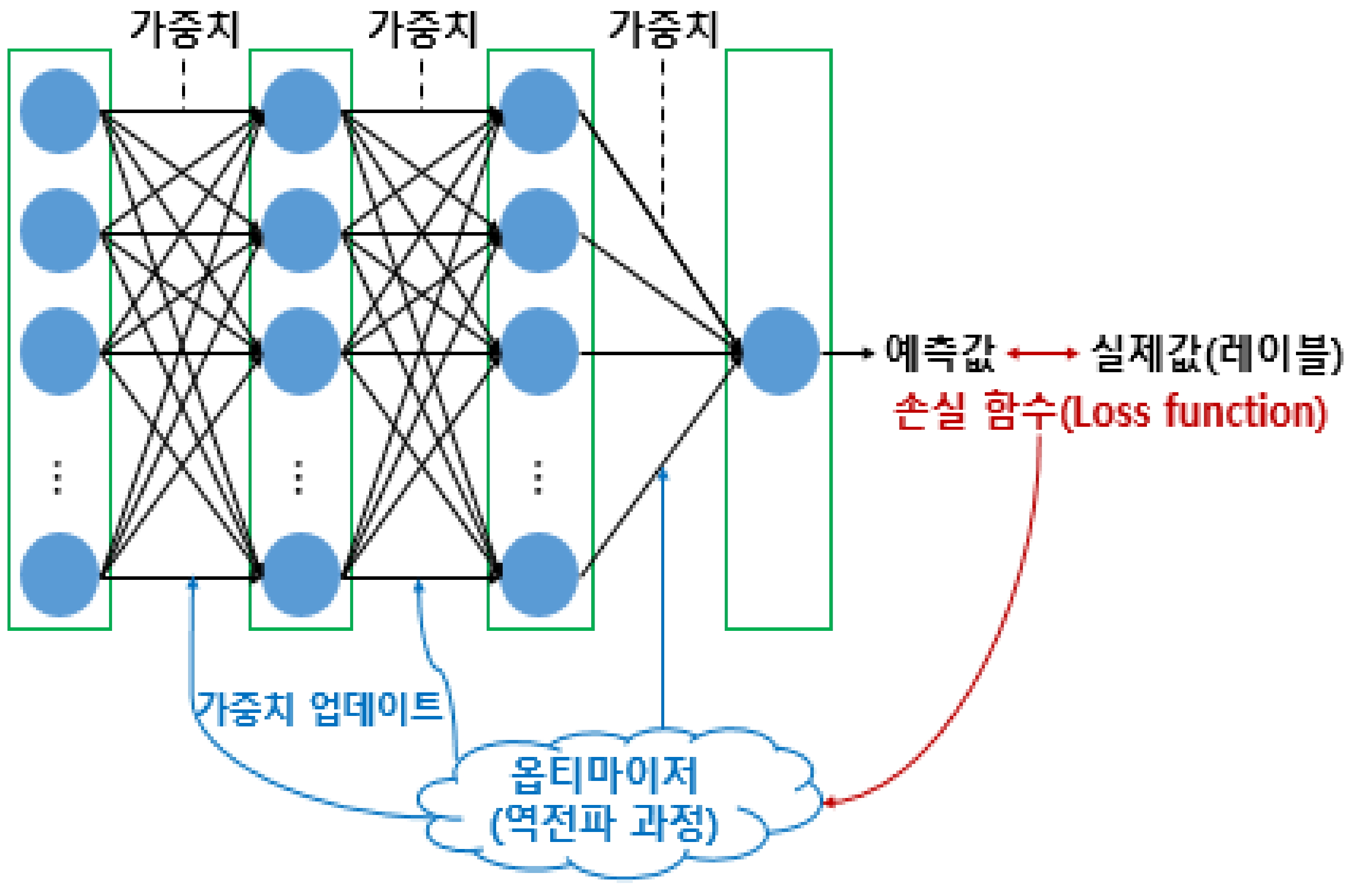
# #3 Computation Graphs and Backpropagation

## Introduction of Back-prop

- Update the weights in the direction of reducing the error

- taking and propagating **derivatives** and using the (matrix) **chain rule**

- efficiently re-use of derivatives by constructing **computational graph**



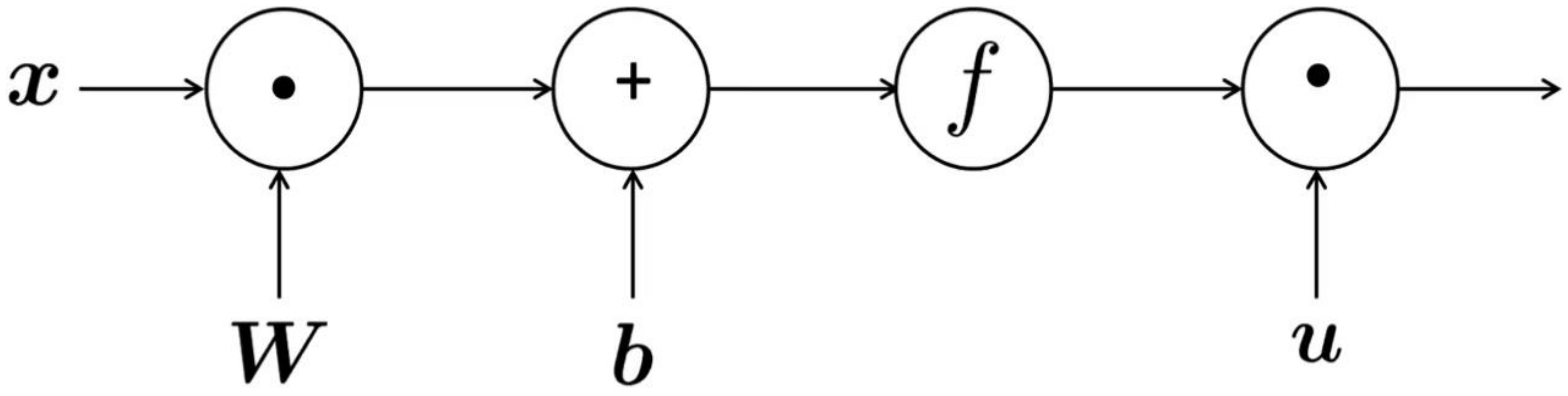
# #3 Computation Graphs and Backpropagation

## Computation Graph

**Computational graph** : a graph of the computational process

- node : operations

- edge : connection between nodes



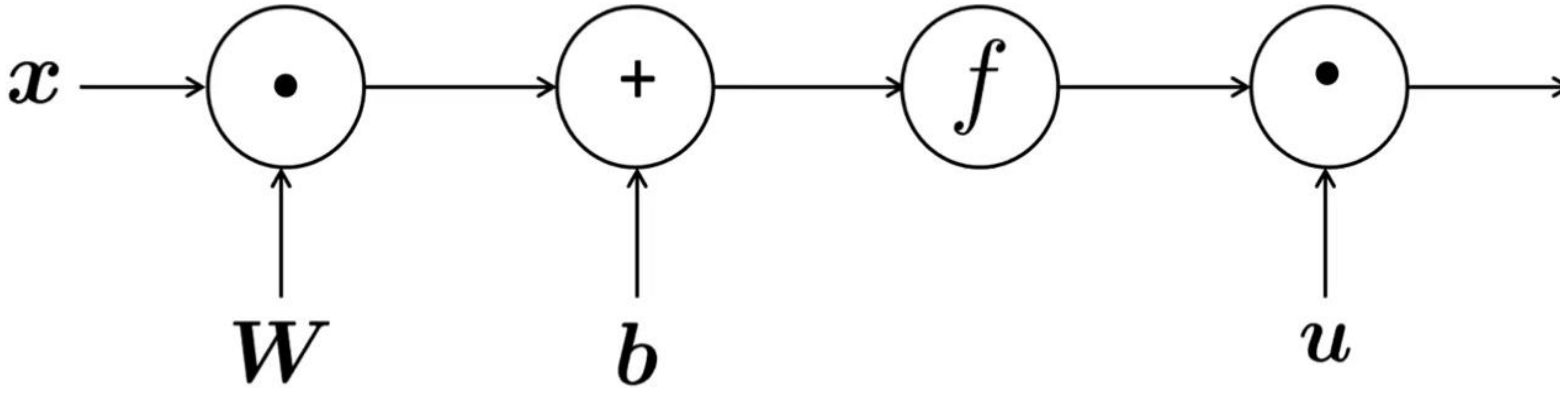

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

## Computation Graph

### Advantages

- **Local calculation** : each node can focus on simple calculations
- **Save intermediate calculations** : differentials can be calculated efficiently using back-prop

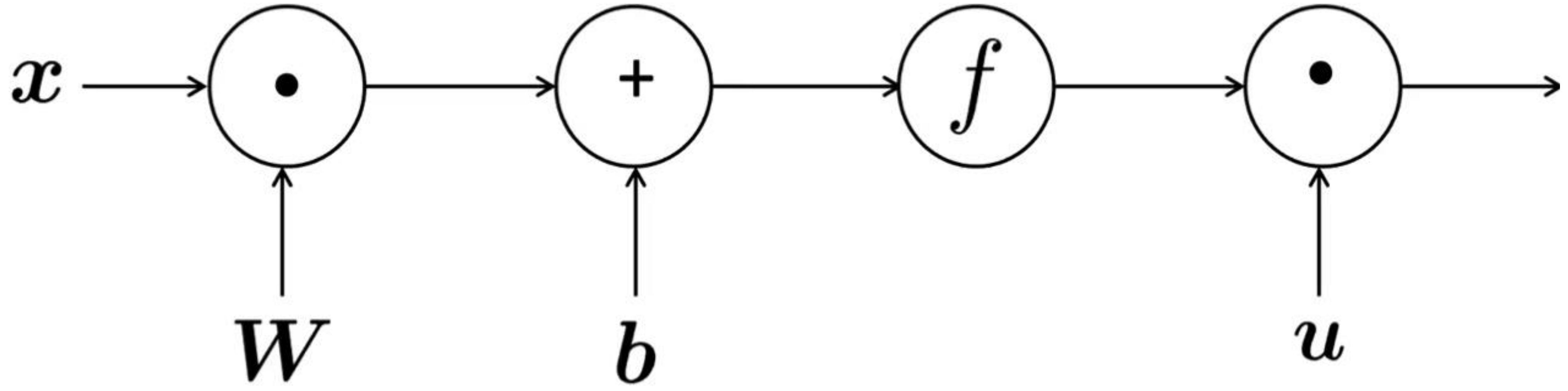



$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
$$x \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

## Forward Propagation

**Calculate and store variables** in order from the input layer to the output layer of the neural network model

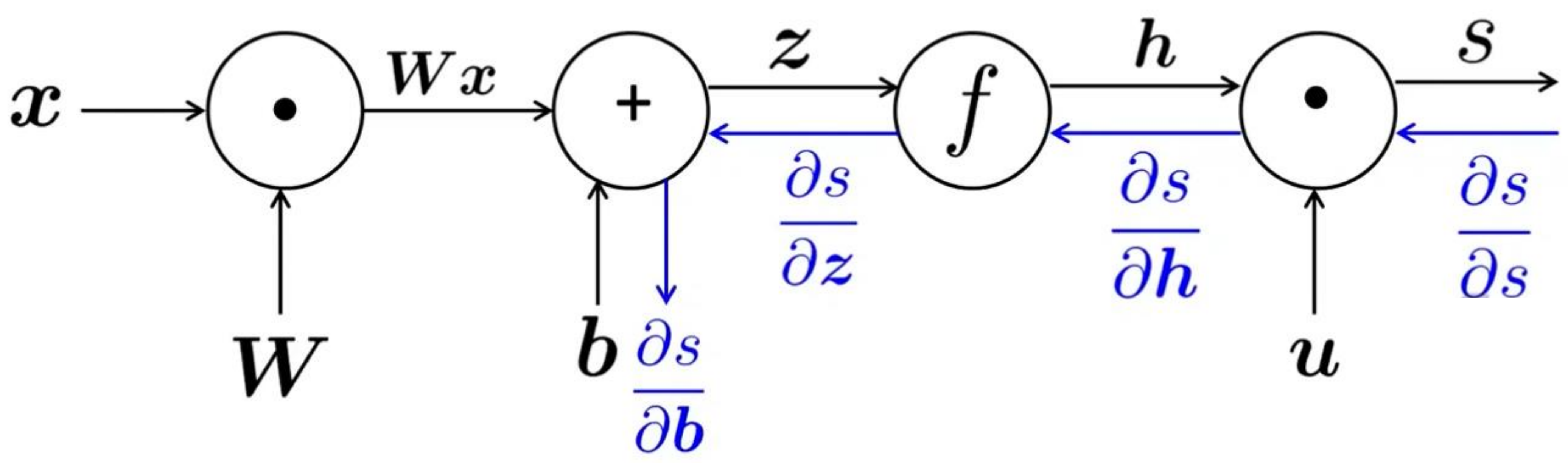



$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
$$x \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

## Backpropagation

- Go backwards along **gradients** (edges)




$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

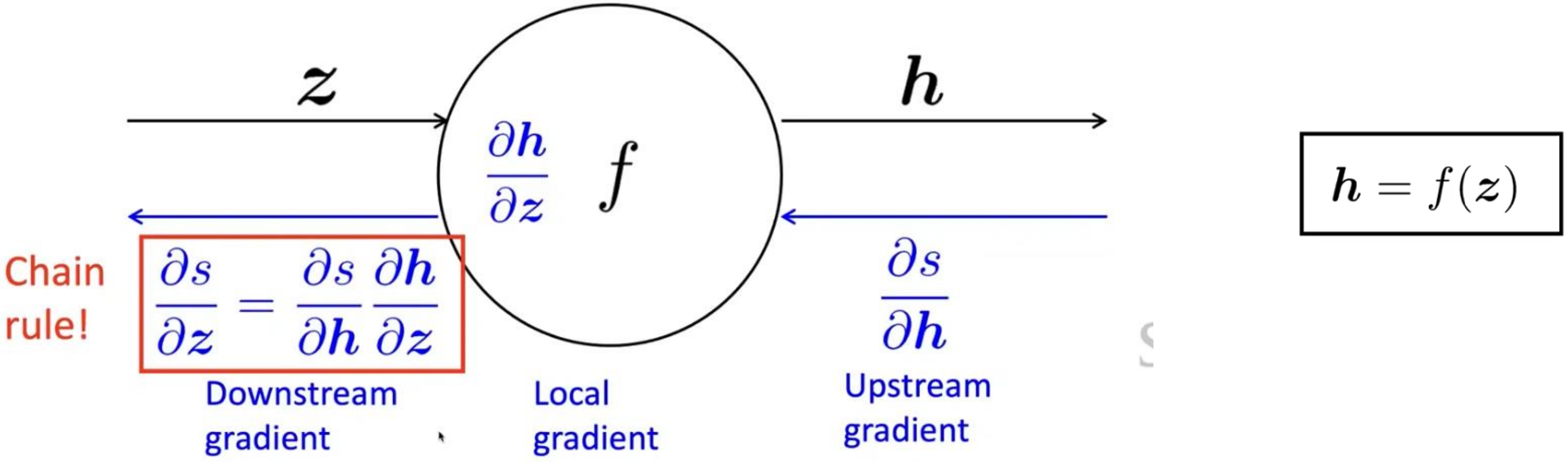
## Backpropagation

- Node receives an **upstream gradient**

- Goal is to pass the correct **downstream gradient**

- **Each node has a local gradient**
  : The gradient of its output with respect to its inputs

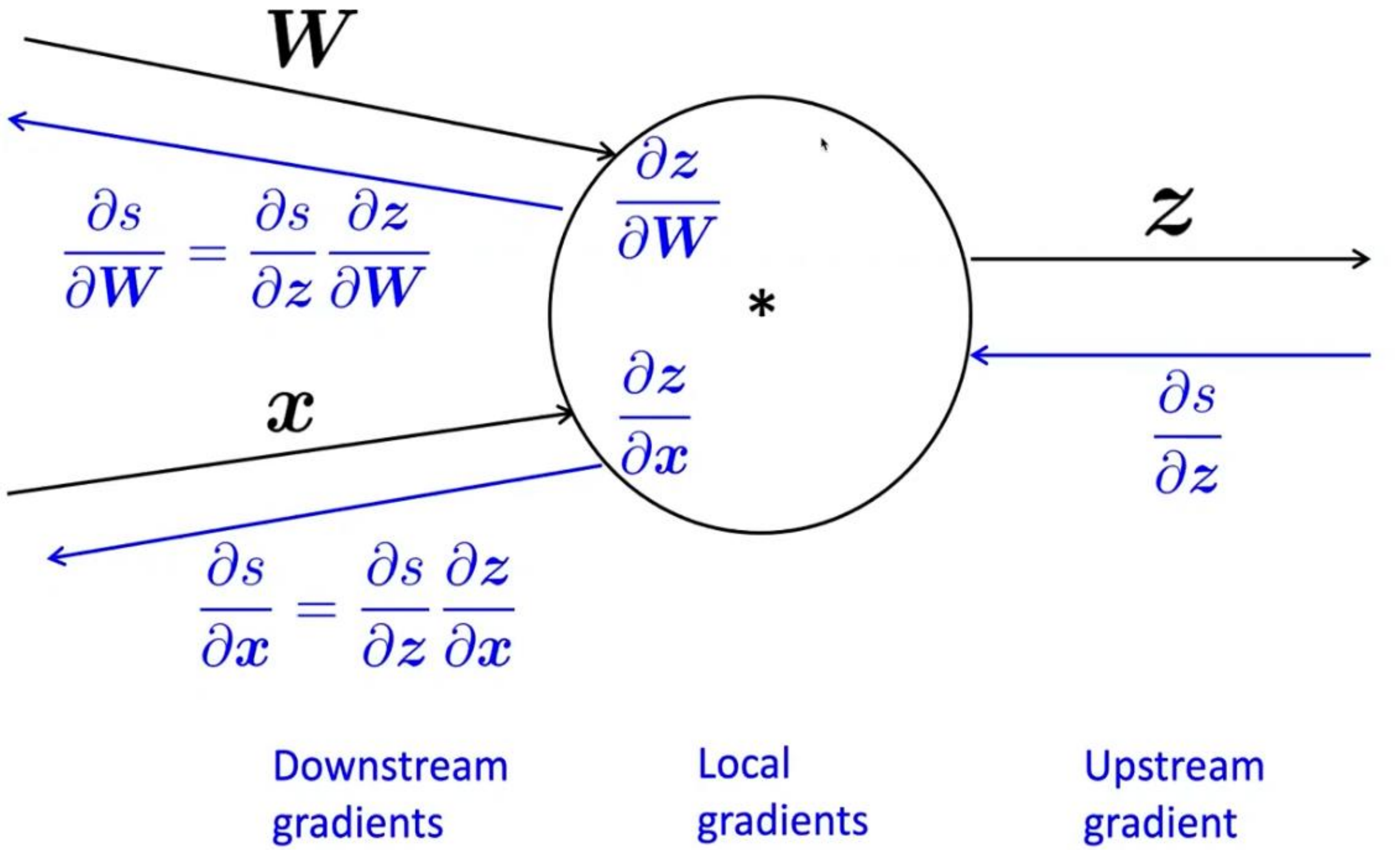
- [downstream gradient] = [upstream gradient] * [local gradient]



# #3 Computation Graphs and Backpropagation

## Backpropagation

- Multiple input -> multiple local gradients
- To calculate **the Local gradient, the output of the node must be differentiated into the input.**
- This local gradient multiplies with the upstream gradient and becomes the downstream gradient and flows down.




$$\boxed{z = Wx}$$

# #3 Computation Graphs and Backpropagation

## An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

1. **Forward-propagation**

**Local gradient** can be obtained during forward propagation,

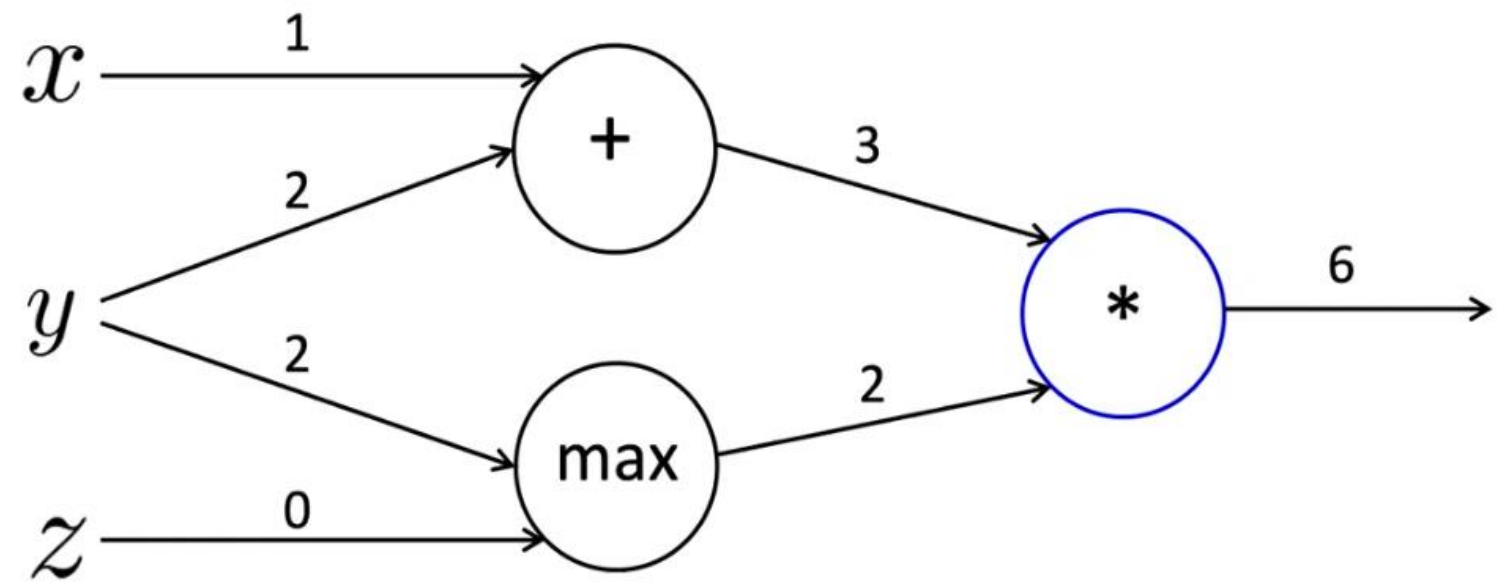
so it is calculated and stored in advance and used during back-propagation.

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



# #3 Computation Graphs and Backpropagation

## An example

$$f(x,y,z) = (x+y)\max(y,z)$$
$$x = 1, y = 2, z = 0$$

### 2. Backpropagation

1-step (product node)

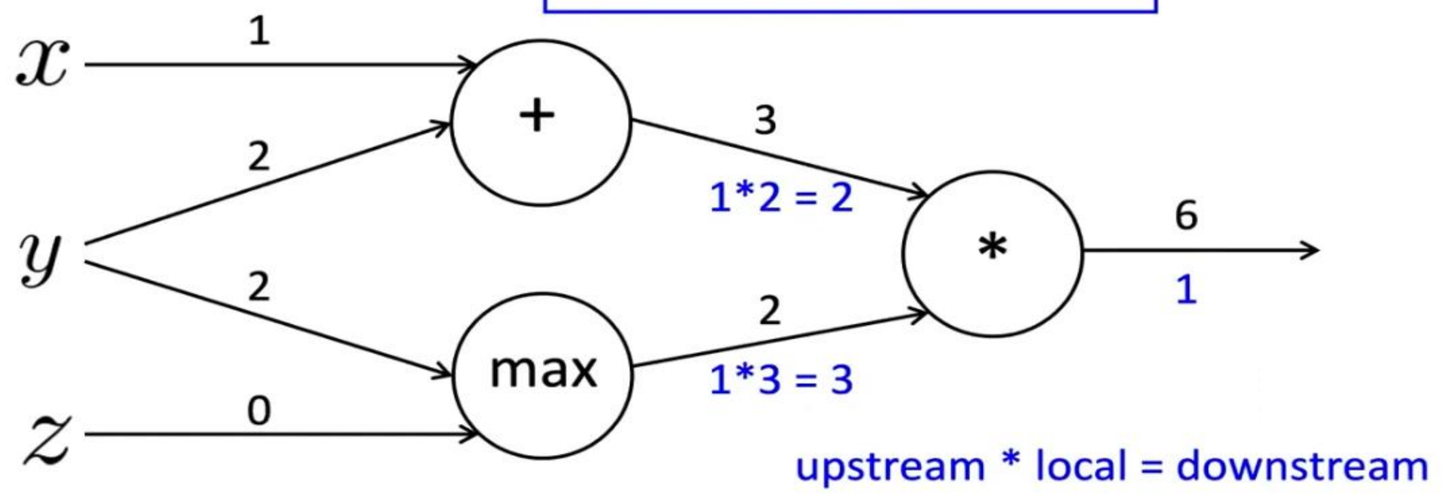
- upstream gradient : $\frac{df}{df}$
- local gradient : $\frac{df}{da}$ and $\frac{df}{db}$
- Compute downstream gradient $\frac{df}{da}$ and $\frac{df}{db}$

Forward prop steps

$$a = x + y$$
$$b = \max(y,z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\boxed{\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3}$$



# #3 Computation Graphs and Backpropagation

## An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

### 2. Backpropagation

2-step (plus node)

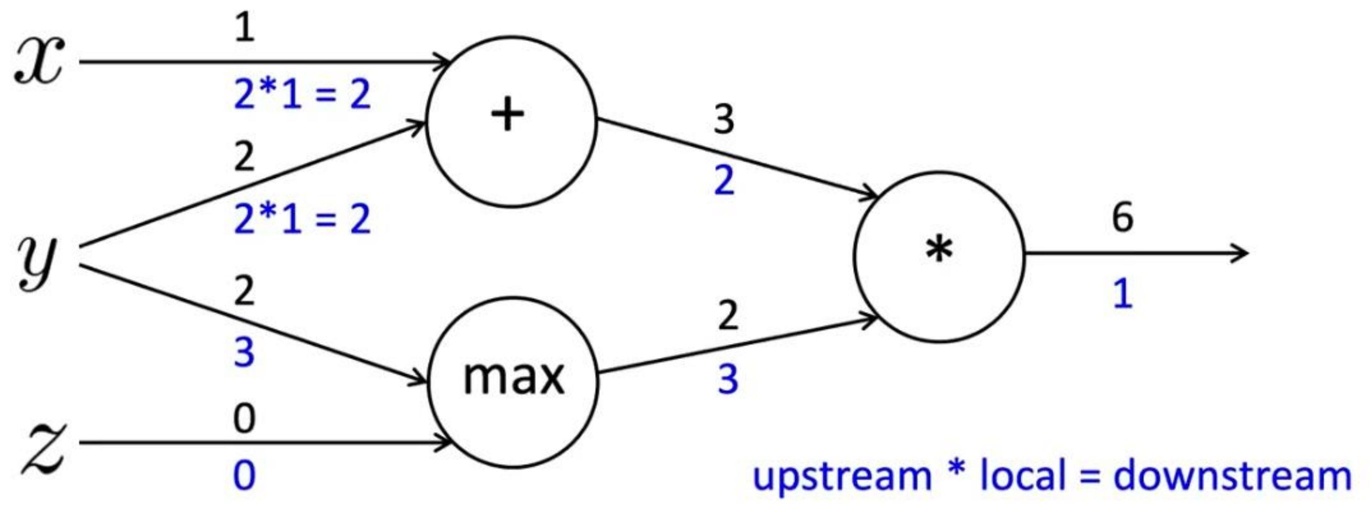
- upstream gradient : $\frac{df}{da}$
- local gradient : $\frac{da}{dx}$ and $\frac{da}{dy1}$
- Compute downstream gradient $\frac{df}{dx}$ and $\frac{df}{dy1}$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\boxed{\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1}$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



# #3 Computation Graphs and Backpropagation

## An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

### 2. Backpropagation

3-step (max node)

- upstream gradient : $\frac{df}{db}$
- local gradient : $\frac{db}{dy2}$ and $\frac{db}{dz}$
- Compute downstream gradient $\frac{df}{dy2}$ and $\frac{df}{dz}$

4-step

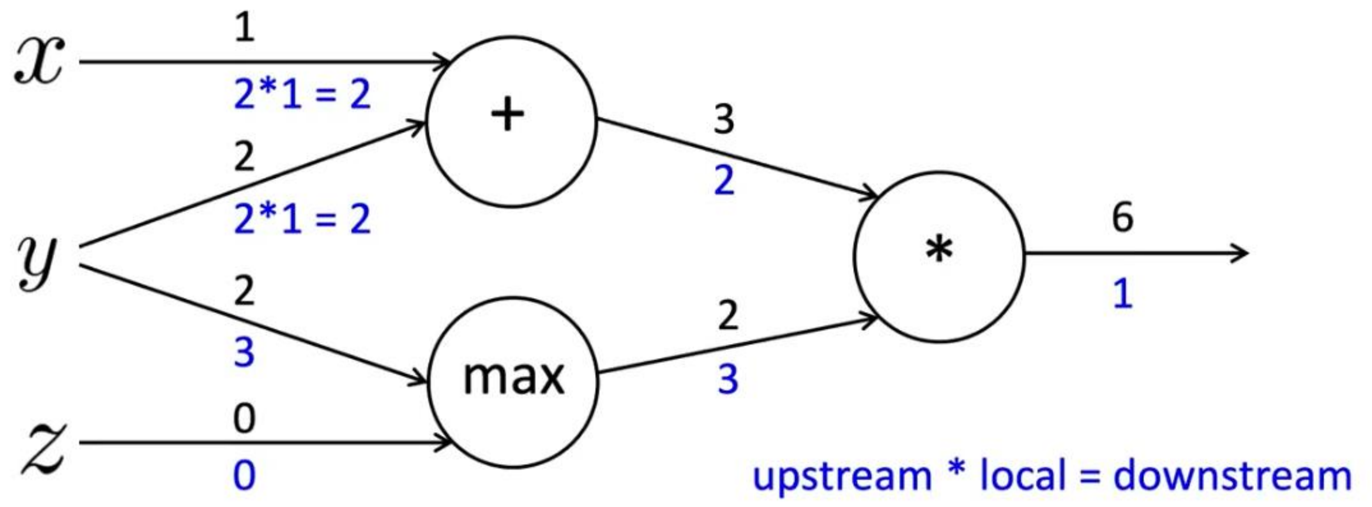
- compute downstream gradient $\frac{df}{dy} = \frac{df}{dy1} + \frac{df}{dy2}$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\boxed{\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1}$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



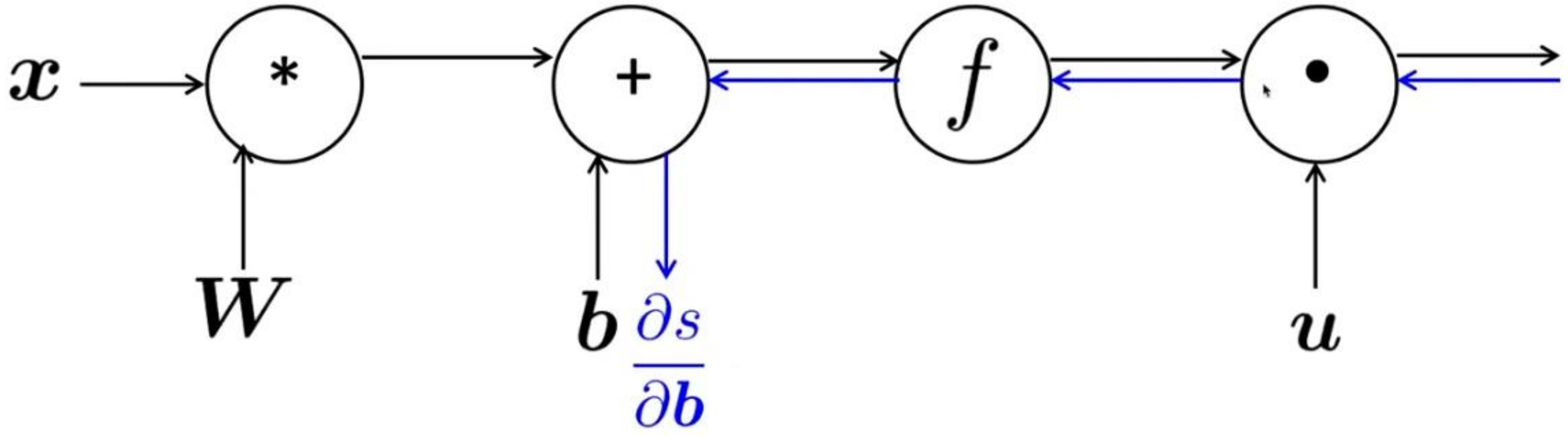
# #3 Computation Graphs and Backpropagation

## Efficiency: compute all gradients at once

**Inefficient way**

- Compute $\frac{ds}{db}$ first

- Compute again $\frac{ds}{db}$ to get $\frac{ds}{dW}$



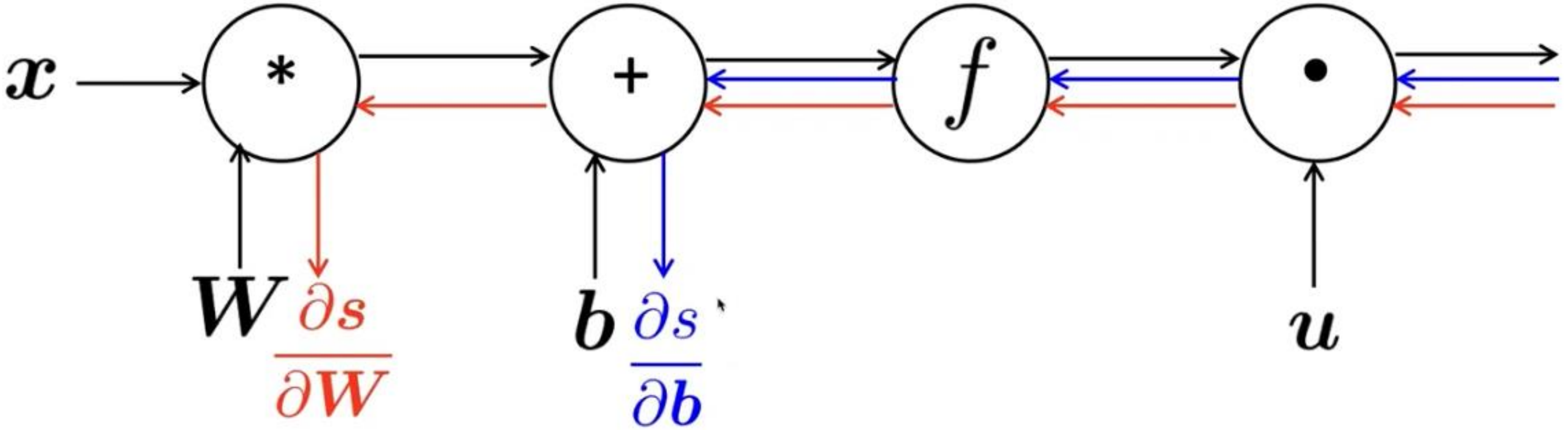

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

## Efficiency: compute all gradients at once

### Efficient way

- Compute all gradient at once
- Re-use previously used upstream gradient and local gradient for $\frac{ds}{db}$ to compute $\frac{ds}{dW}$




$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
$$x \quad \text{(input)}$$

# #3 Computation Graphs and Backpropagation

## Back-prop in General Computation Graph

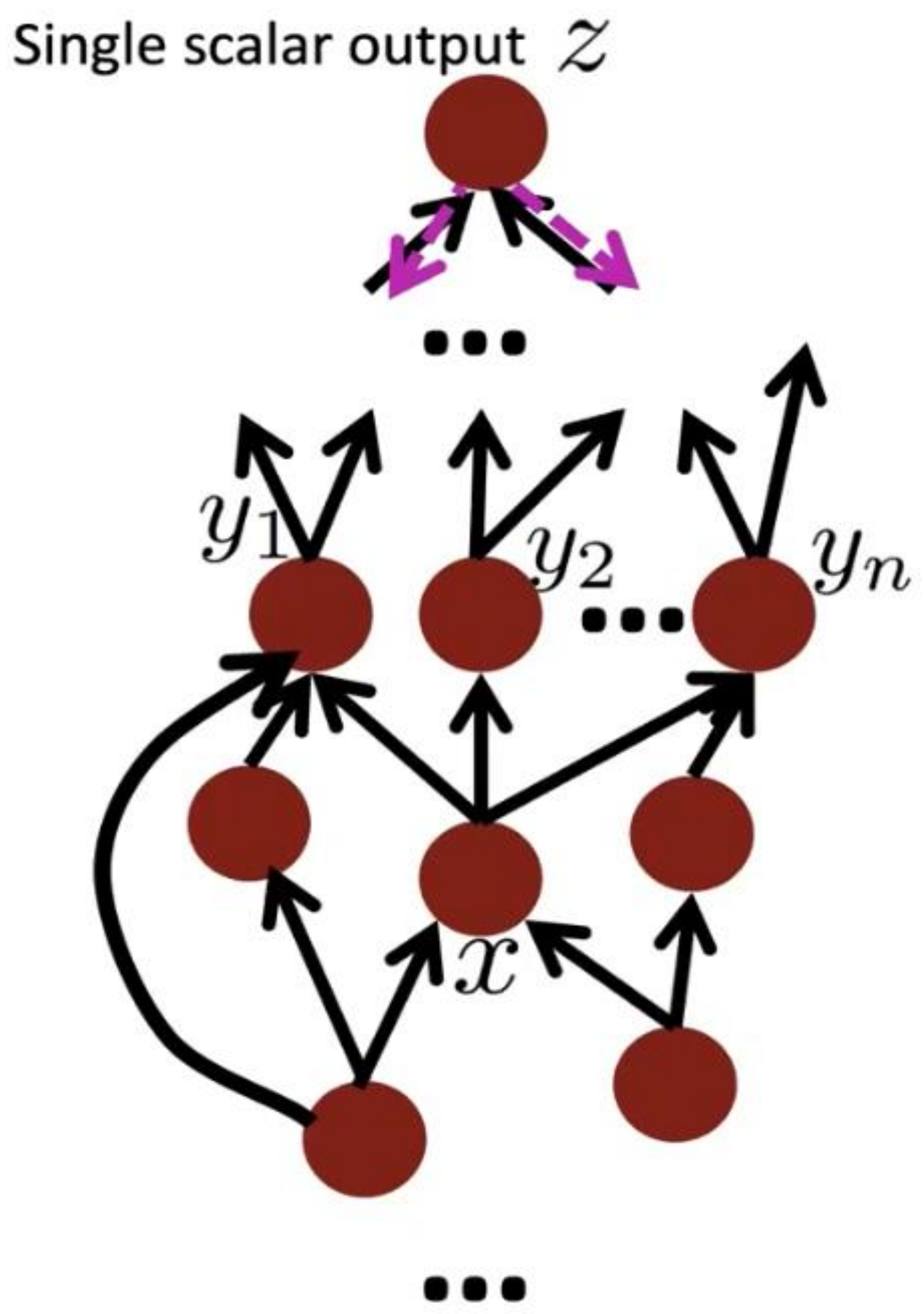
1. **F-prop** : visit nodes in topological sort order

2. **B-prop**:

   - initialize output gradient =1

   - visit nodes in reverse order
   : compute gradient wrt each node using
   gradient wrt successors {y1,⋯yn} = successors of x

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



# #3 Computation Graphs and Backpropagation

## Manual Gradient checking: Numeric Gradient

- Easy to implement correctly $$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- But approximate and very slow : recompute f for every parameter of model

- **Useful for checking implementation** : not much less needed, just check layers are correctly implemented

# #3 Computation Graphs and Backpropagation

## Summary

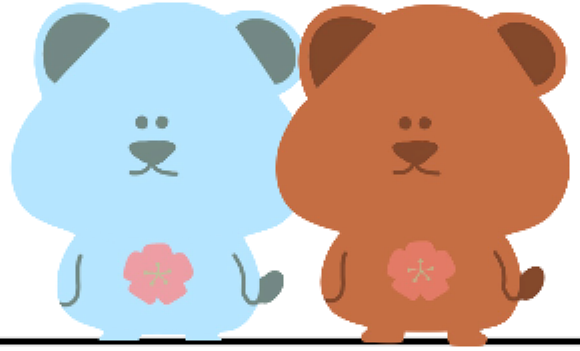
- **Backpropagation** : recursively (and hence efficiently) apply the chain rule along computation graph

  > [downstream gradient] = [upstream gradient] * [local gradient]

- **Forward pass** : compute results of operations and save intermediate values

- **Backward pass** : apply chain rule to compute gradients

# THANK YOU



EWHA
EURON