

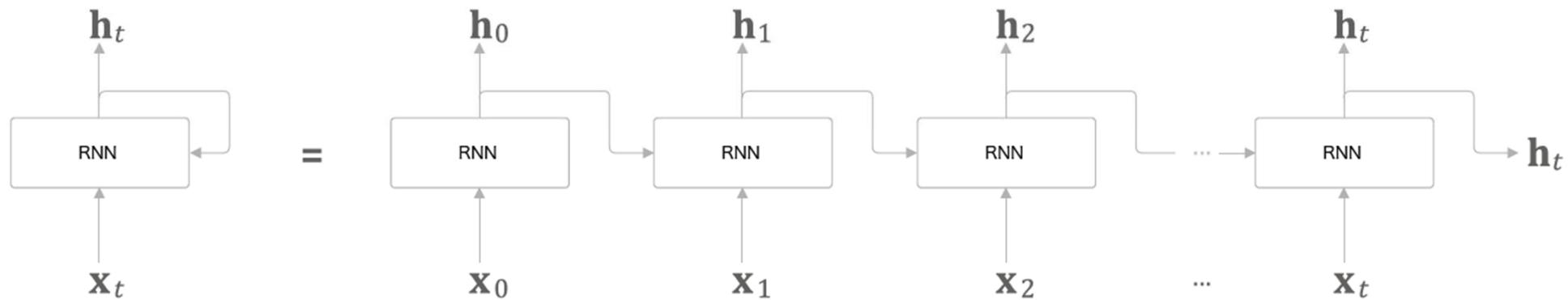
## NLP 개론 (4)

게이트가 추가된 RNN

# RNN 복습

- RNN의 복습
  - 입력 : 시계열 데이터(  $x_t$  )
  - 출력 : 은닉 상태 (  $h_t$  )  
은닉 상태는 과거 정보를 저장한다.
  - 순환 경로를 갖고 있다.

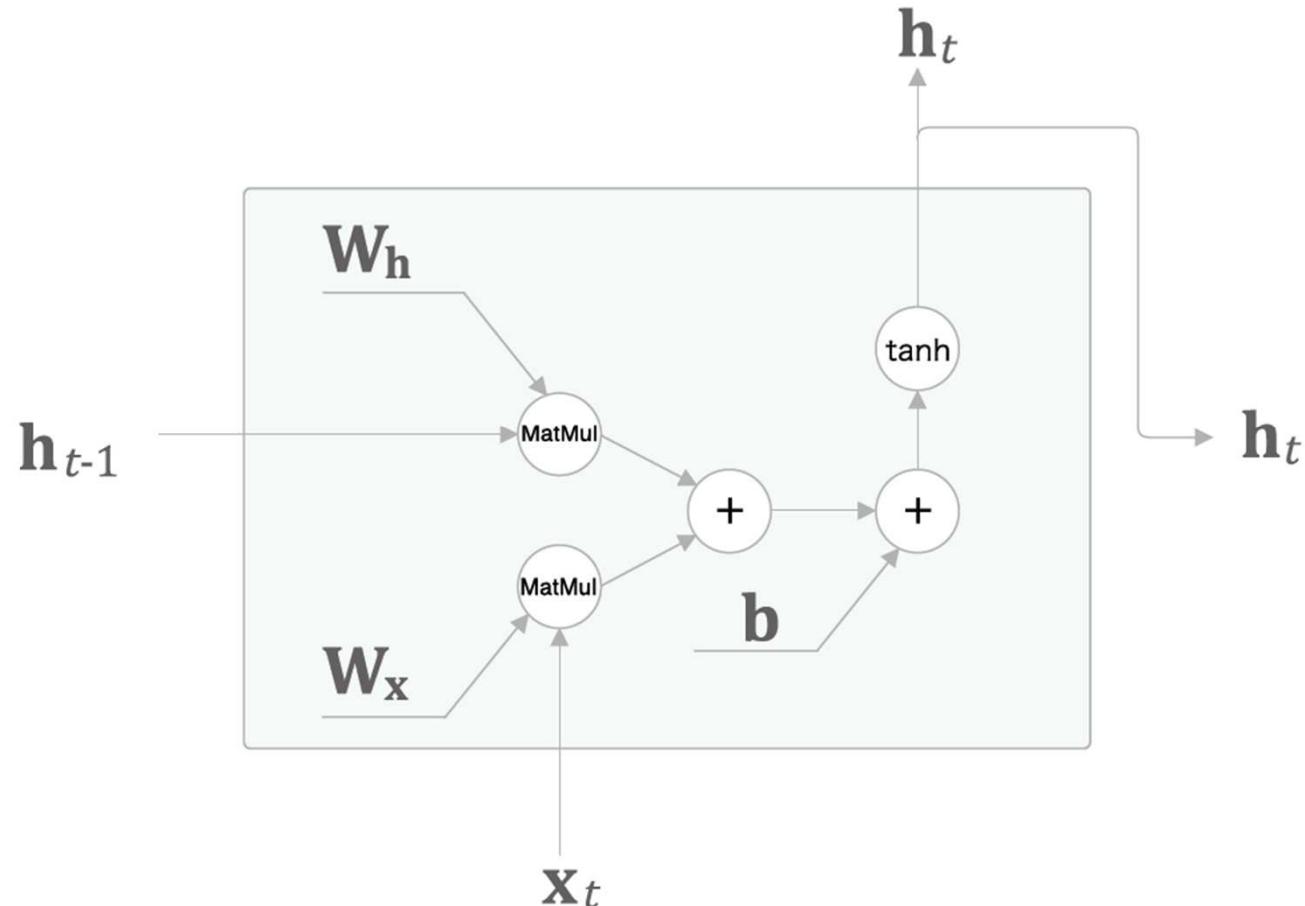
그림 6-1 RNN 계층: 순환을 펼치기 전과 후



# RNN 복습

- RNN 계층이 수행하는 처리\_계산 그래프

그림 6-2 RNN 계층의 계산 그래프(MatMul 노드는 행렬 곱을 나타냄)



# RNN의 문제점

그림 6-3 “?”에 들어갈 단어는?: (어느 정도의) 장기 기억이 필요한 문제의 예

Tom was watching TV in his room. Mary came into the room. Mary said hi to ?

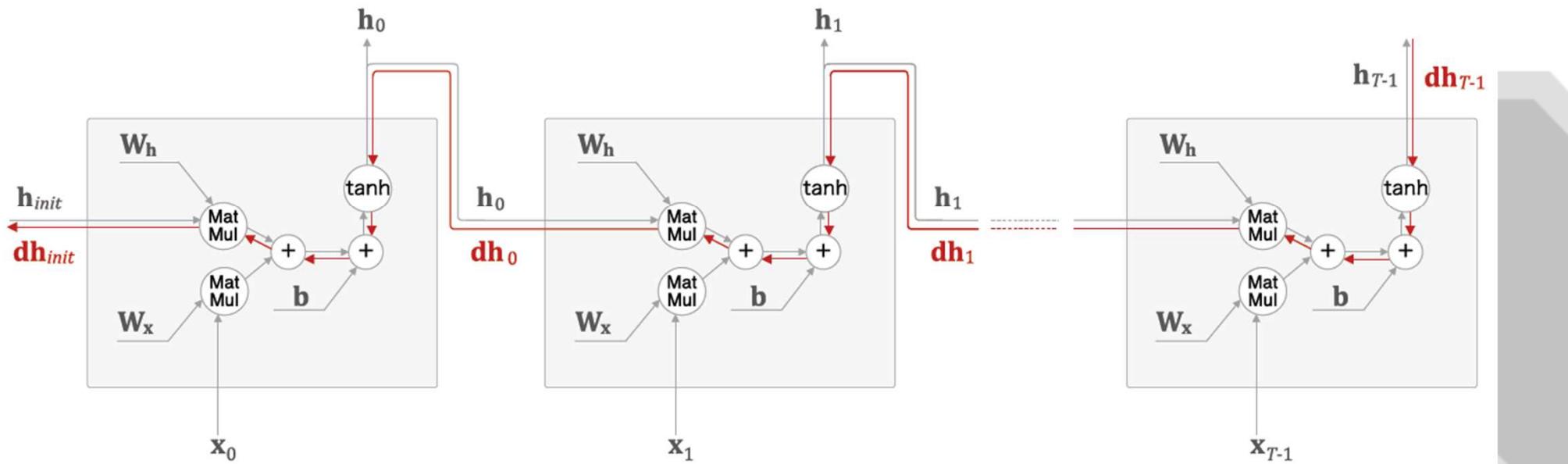
중요한 정보를 은닉 상태에 인코딩해 보관하여, 답을 맞춰야 한다.

- RNN의 학습과정 목표
    - : RNN 계층이 과거 방향으로 의미 있는 기울기를 전달  
-> 시간 방향의 의존성 관계 학습
  - 5장 기본 RNN의 한계
    - : 의미 있는 기울기가 중간에 작아질 경우 -> 가중치 갱신 X  
(너무 커지는 경우 -> 손실 함수 값 발산 -> 불안정한 모델)
- \* 시간을 거슬러 올라갈수록 기울기가 작아진다.

# RNN의 문제점 : 기울기 소실 (폭발)

- 기울기 소실(폭발) 원인  
: RNN 계층의 행렬 곱에 의한 역전파 기울기

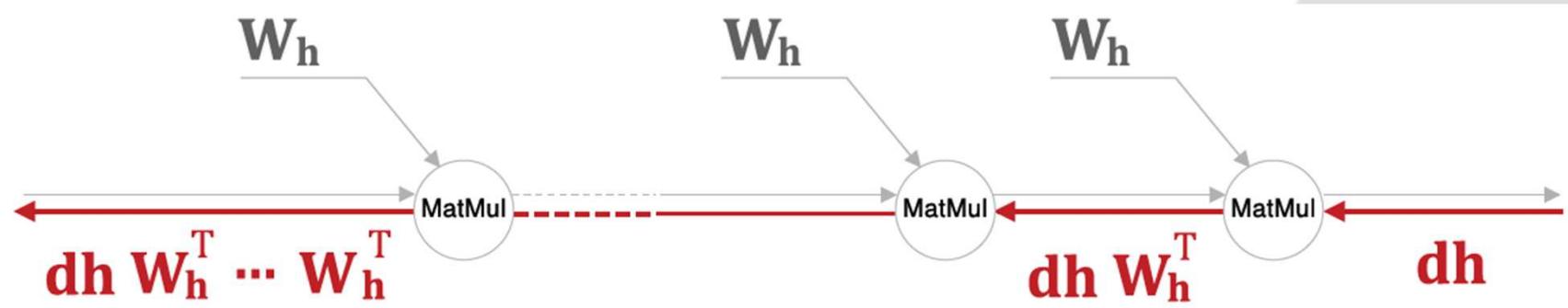
그림 6-5 RNN 계층에서 시간 방향으로의 기울기 전파



RNN 특징 : 가중치 공유  
 -> MathMul 노드를 지날때마다, 지난 기울기에  $W_h$ 를 곱해준다.

# RNN의 문제점 : 기울기 소실 (폭발)

- 기울기 소실 (폭발) 원인  
: RNN 계층의 행렬 곱에 의한 역전파 기울기



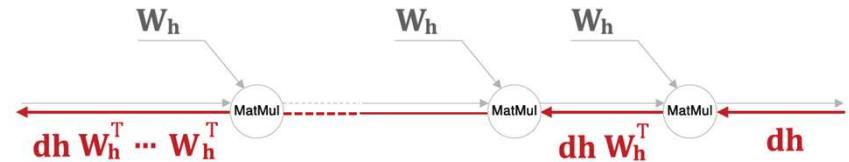
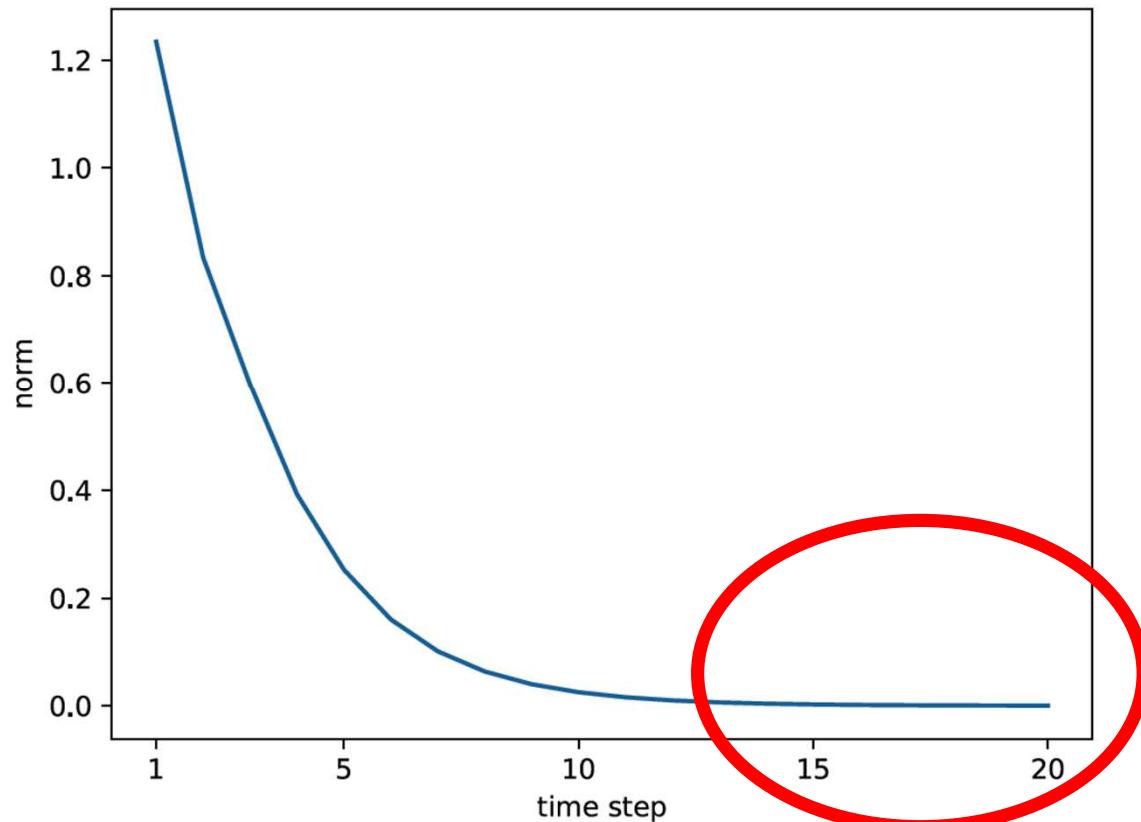
RNN 특징 : 가중치 공유  
 -> MatMul 노드를 지날때마다, 지난 기울기에  $W_h^T$

# RNN의 문제점 : 기울기 소실

- 기울기 소실 (폭발) 원인  
: Matmul 노드를 지날 때마다 기울기( $dh$ )가 **작아진다**(or 커진다).

Case1)  $W_h < 1$  일 때, 기울기  $dh$  그래프

그림 6-9 기울기  $dh$ 는 시간 크기에 비례하여 지수적으로 감소한다.

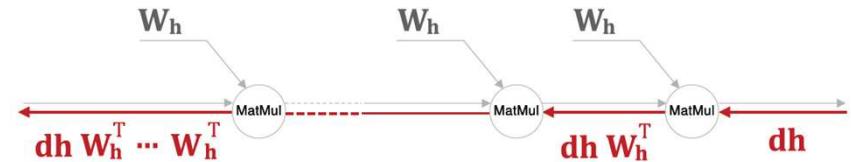
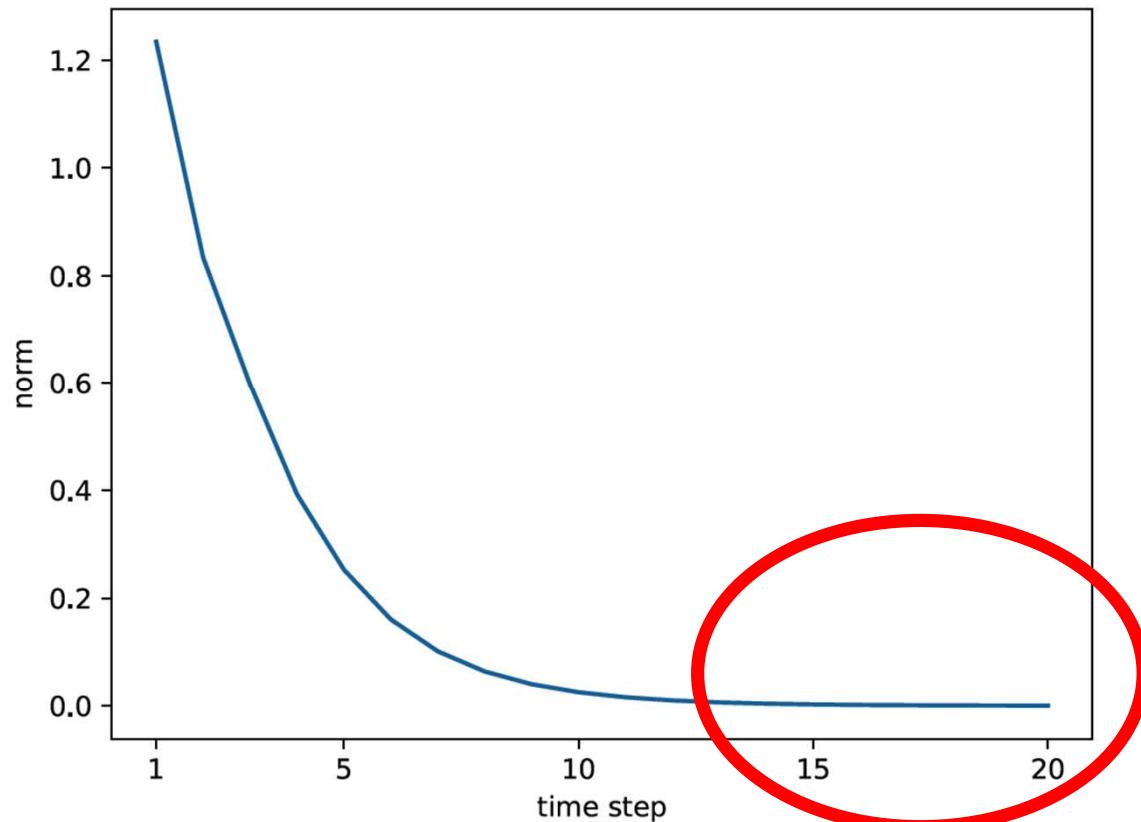


# RNN의 문제점 : 기울기 소실

- 기울기 소실 (폭발) 원인  
: Matmul 노드를 지날 때마다 기울기( $dh$ )가 **작아진다**(or 커진다).

Case1)  $W_h < 1$  일 때, 기울기  $dh$  그래프

그림 6-9 기울기  $dh$ 는 시간 크기에 비례하여 지수적으로 감소한다.



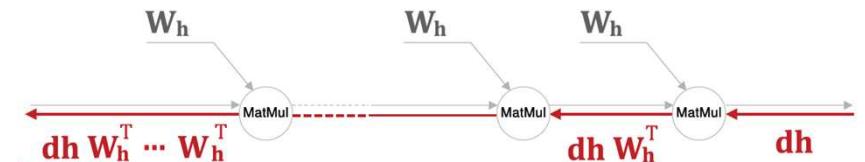
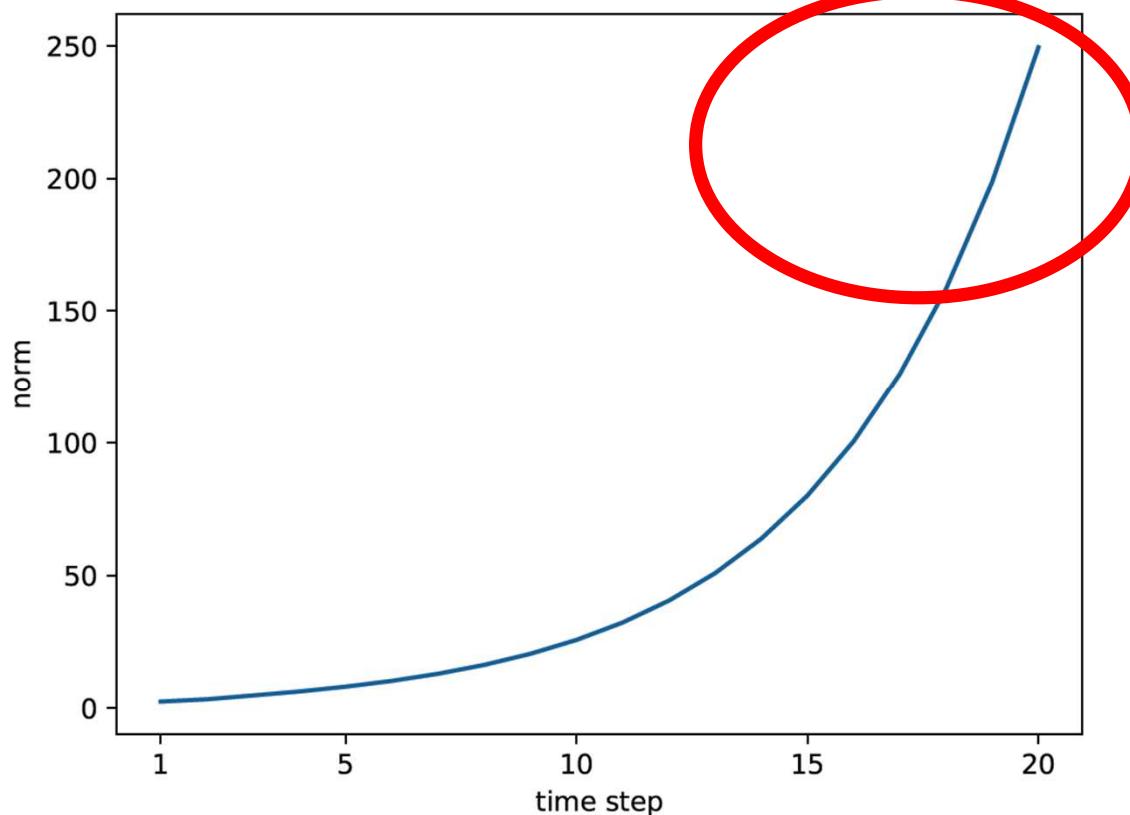
IHS  
매개변수 갱신 X  
→ 장기 의존 관계 학습 X

# RNN의 문제점 : 기울기 폭발

- 기울기 소실(폭발) 원인  
: Matmul 노드를 지날 때마다 기울기( $dh$ )가 작아진다(or 커진다).

Case2)  $W_h > 1$  일 때, 기울기  $dh$  그래프

그림 6-8 기울기  $dh$ 는 시간 크기에 비례하여 지수적으로 증가한다.

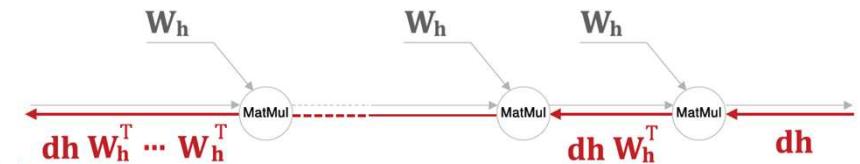
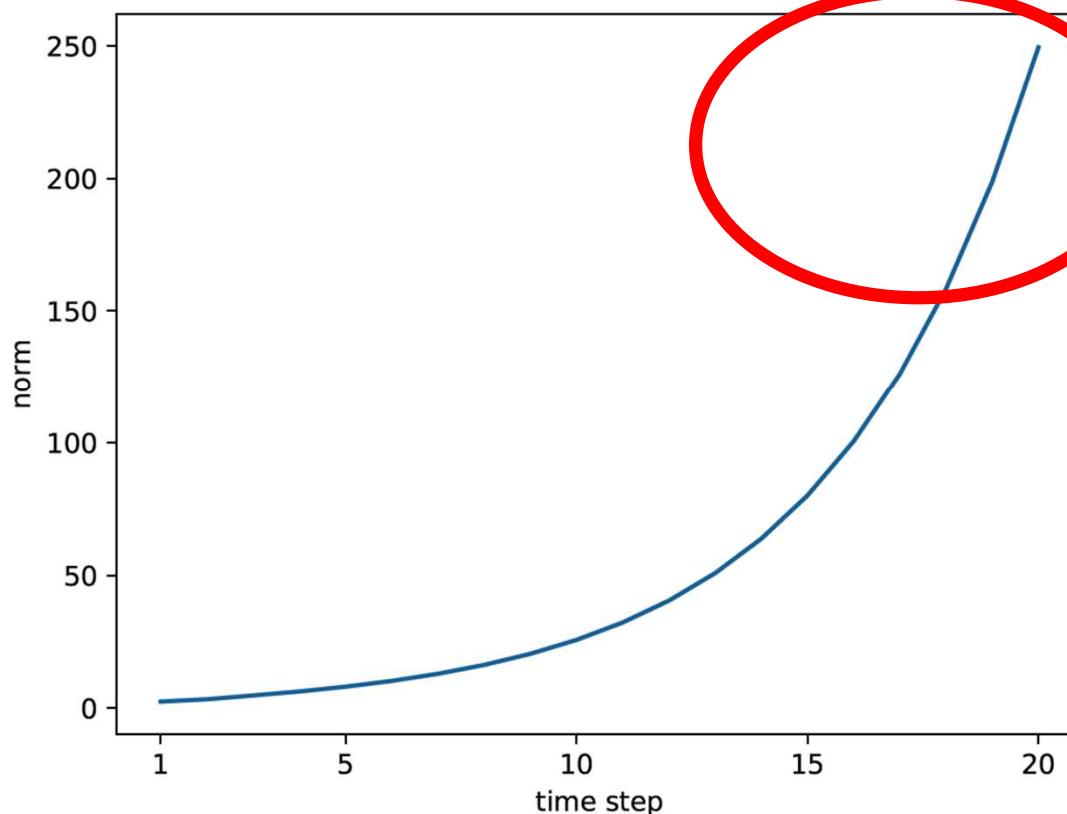


# RNN의 문제점 : 기울기 폭발

- 기울기 소실(폭발) 원인  
: Matmul 노드를 지날 때마다 기울기( $dh$ )가 작아진다(or 커진다).

Case2)  $W_h > 1$  일 때, 기울기  $dh$  그래프

그림 6-8 기울기  $dh$ 는 시간 크기에 비례하여 지수적으로 증가한다.



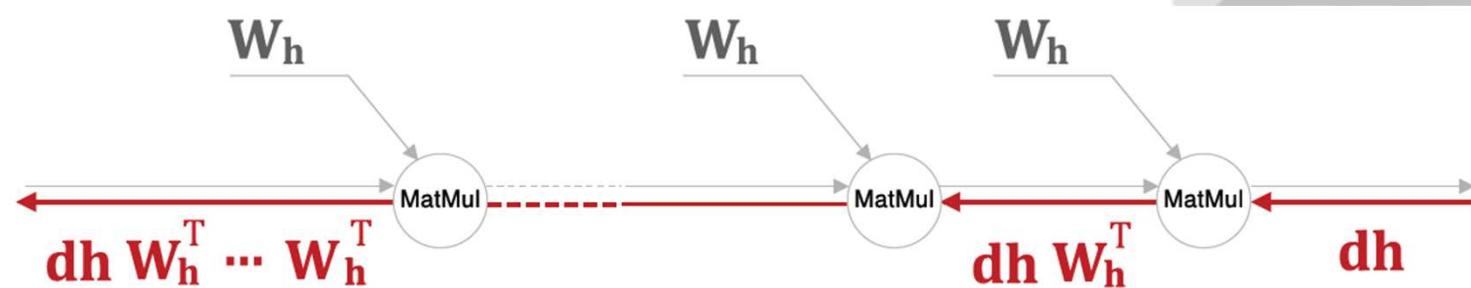
IHS

기울기 폭발  
→ 오버플로 ( $\text{NaN}$  값 발생)  
→ 학습 정상 수행 불가능

# RNN의 문제점 : 기울기 소실

- 기울기 소실 원인\_정리
  - : Matmul 노드를 지날 때마다 기울기( $dh$ )가 지수적으로 변화
  - Case1)  $W_h < 1$  일때, 기울기 소실
  - Case2)  $W_h > 1$  일때, 기울기 폭발

이유 : 행렬  $W_h$ 를 T번 반복해서 곱했기 때문



$W_h$ 가 “스칼라”  $\rightarrow 0$

$W_h$ 는 “행렬”  $\rightarrow ???$

# RNN의 문제점 : 기울기 소실

- 기울기 소실 원인-정리
    - : Matmul 노드를 지날 때마다 기울기( $dh$ )가 지수적으로 변화
    - ~~Case1)  $Wh < 1$  일 때, 기울기 소실~~
    - ~~Case2)  $Wh > 1$  일 때, 기울기 폭발~~
- 이유 : 행렬  $Wh$ 를 T번 반복해서 곱했기 때문

$Wh$ 가 “스칼라”  $\rightarrow 0$

$Wh$ 는 “행렬”  $\rightarrow$  행렬의 특잇값이 척도

- Case1)  $\text{Max}(\sigma_i) < 1$  일 때, 기울기 소실 가능성이 매우 높다.
- Case2)  $\text{Max}(\sigma_i) > 1$  일 때, 기울기 폭발 가능성이 매우 높다.



# RNN의 기울기 문제 대책 (1)

- 기울기 폭발 대책
  - 기울기 클리핑 : 기울기의 L2 norm이 임계값보다 높으면 기울기를 낮추어 전달하는 방법

*if  $\|\hat{g}\| \geq threshold$  :*

$$\hat{g} = \frac{threshold}{\|\hat{g}\|} \hat{g}$$

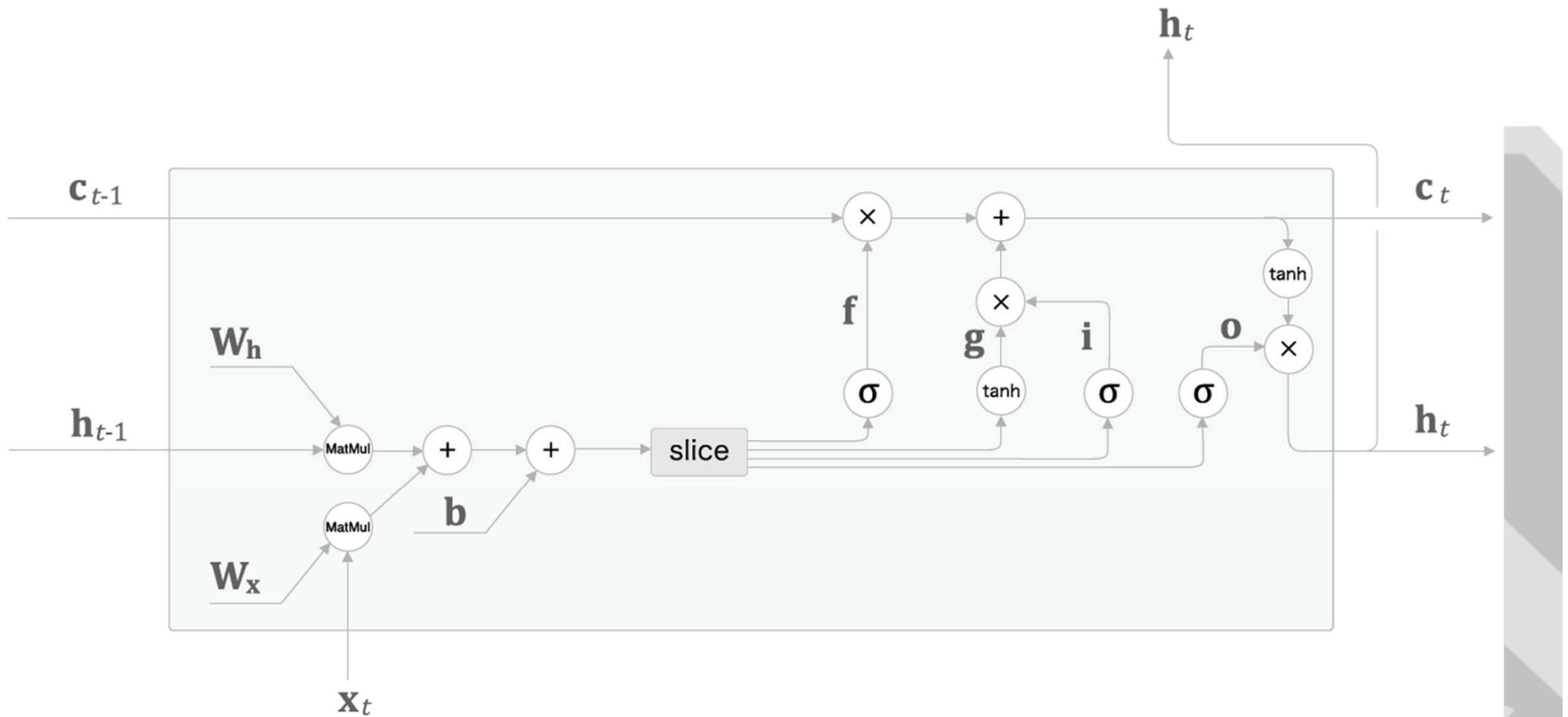
```

1  # coding: utf-8
2  import numpy as np
3
4
5  dw1 = np.random.rand(3, 3) * 10
6  dw2 = np.random.rand(3, 3) * 10
7  grads = [dw1, dw2]
8  max_norm = 5.0
9
10
11 def clip_grads(grads, max_norm):
12     total_norm = 0
13     for grad in grads:
14         total_norm += np.sum(grad ** 2)
15     total_norm = np.sqrt(total_norm)
16
17     rate = max_norm / (total_norm + 1e-6)
18     if rate < 1:
19         for grad in grads:
20             grad *= rate
21
22
23     print('before:', dw1.flatten())
24     clip_grads(grads, max_norm)
25     print('after:', dw1.flatten())

```

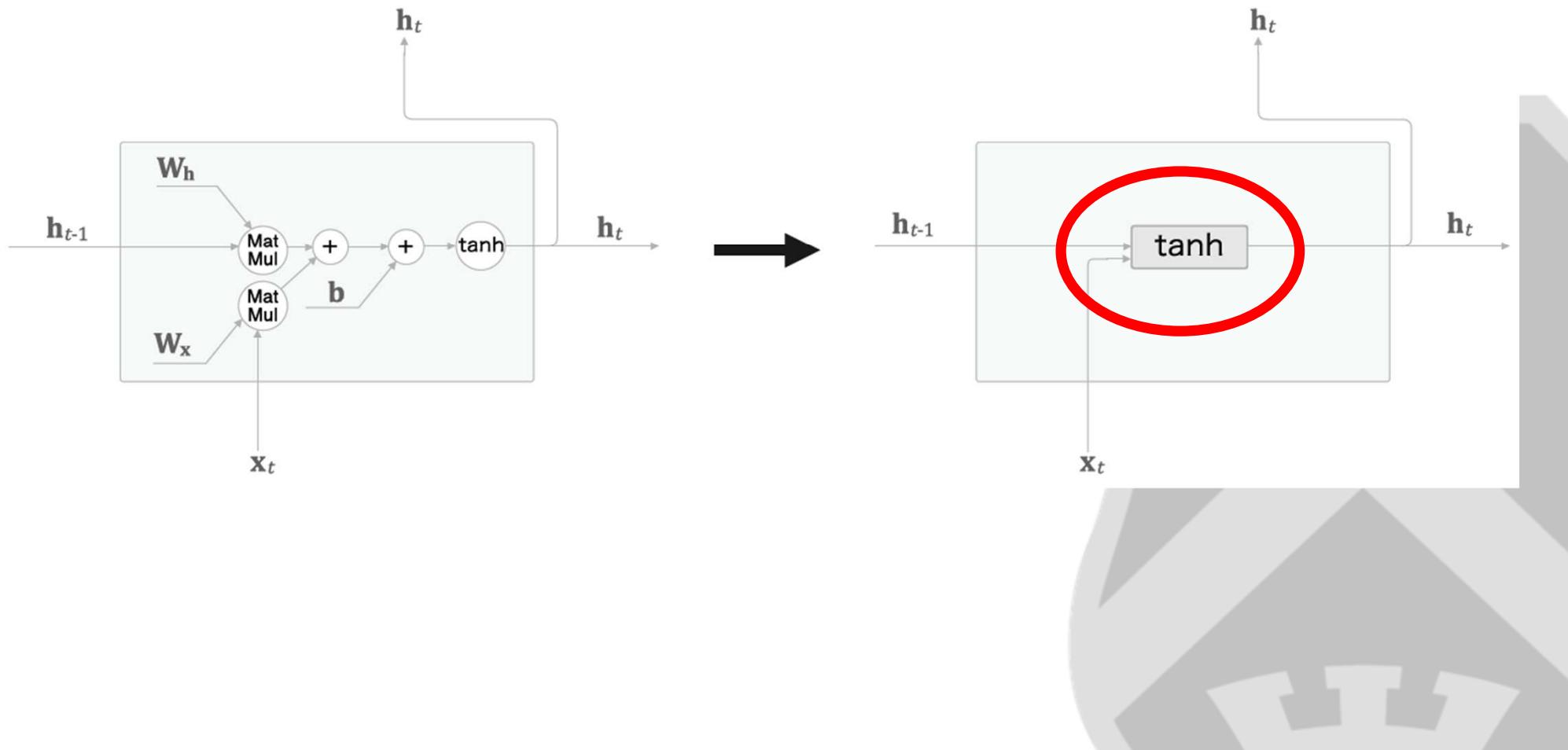
# RNN의 기울기 문제 대책 (2)

- 기울기 소실 대책
  - 게이트가 추가된 RNN (LSTM, GRU)



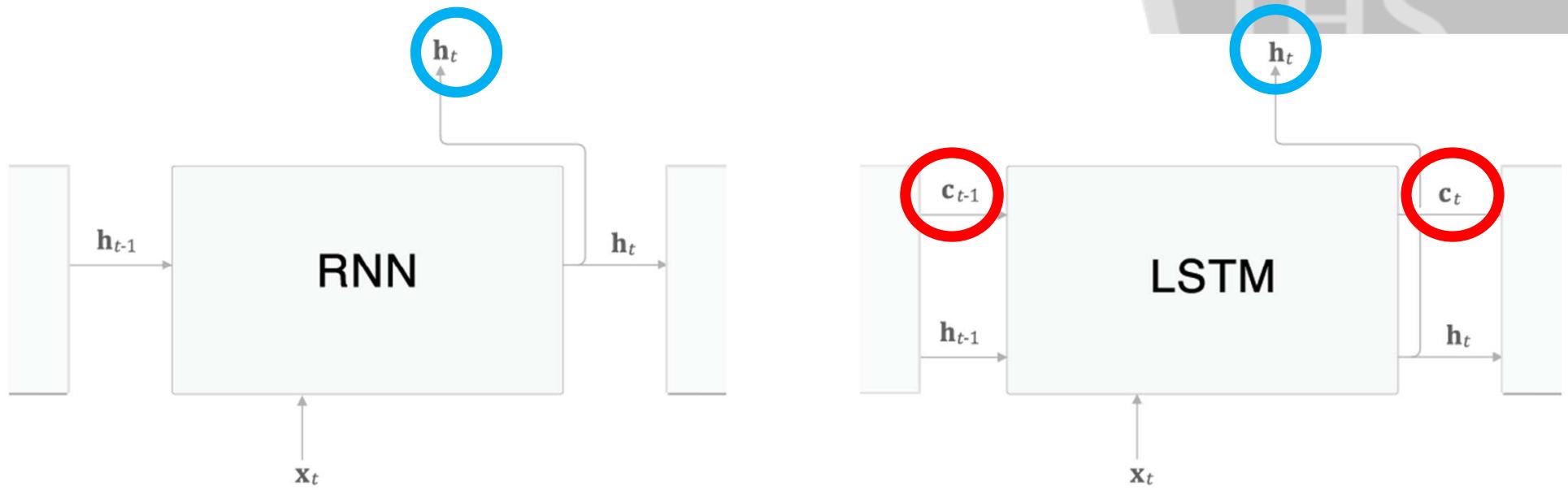
# LSTM 구조

- RNN 인터페이스
  - 단순화한 도법의 RNN 계층 :  $\tanh(h_{t-1}W_h + x_tW_x + b) \rightarrow \tanh$



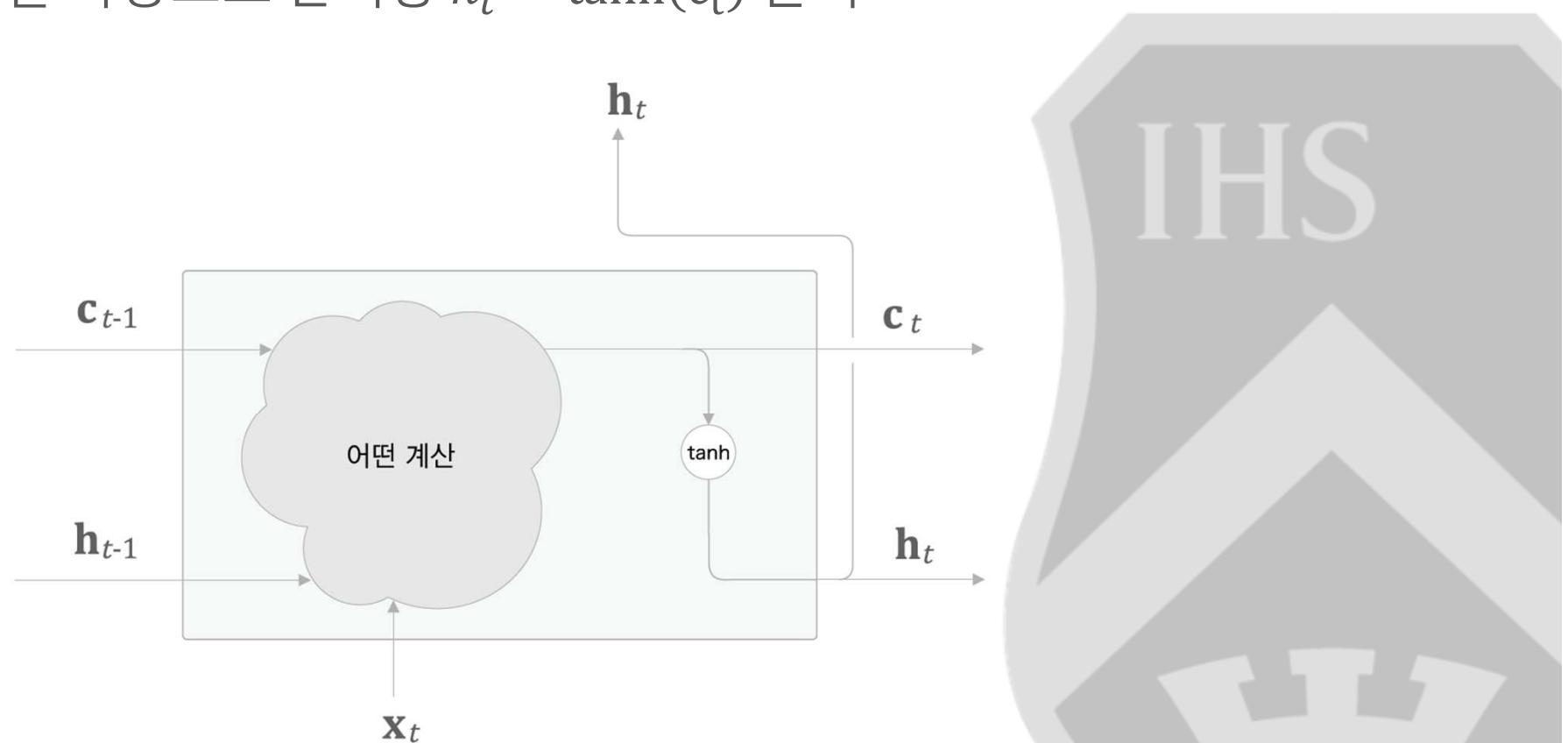
# LSTM 구조

- RNN 계층과 LSTM 계층 비교
  - LSTM 계층 : 경로  $c$  존재
    - \*  $c$  : 기억 셀(Memory Cell)이라 하며, LSTM 전용의 기억 메커니즘
    - \*  $c$  특징 : 데이터를 LSTM 계층 내에서만 주고 받는다.  
다른 계층으로 출력  $x$  (은닉상태  $h$ )는 다른 경로로 주고 받는다.



# LSTM 구조

- LSTM 계층 조립하기
  - LSTM 계층 : 경로  $c$  존재
    - \*  $c$  : 기억 셀(Memory Cell)이라 하며, LSTM 전용의 기억 메커니즘
    - \*  $c_t$  : 시각  $t$ 에서의 LSTM의 기억이 저장
  - $c_t$  를 바탕으로 은닉층  $h_t = \tanh(c_t)$  출력

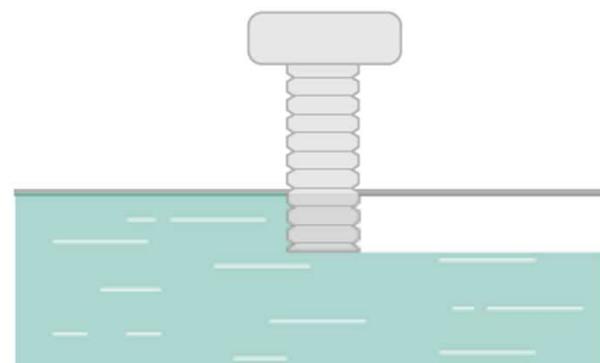


# LSTM 구조

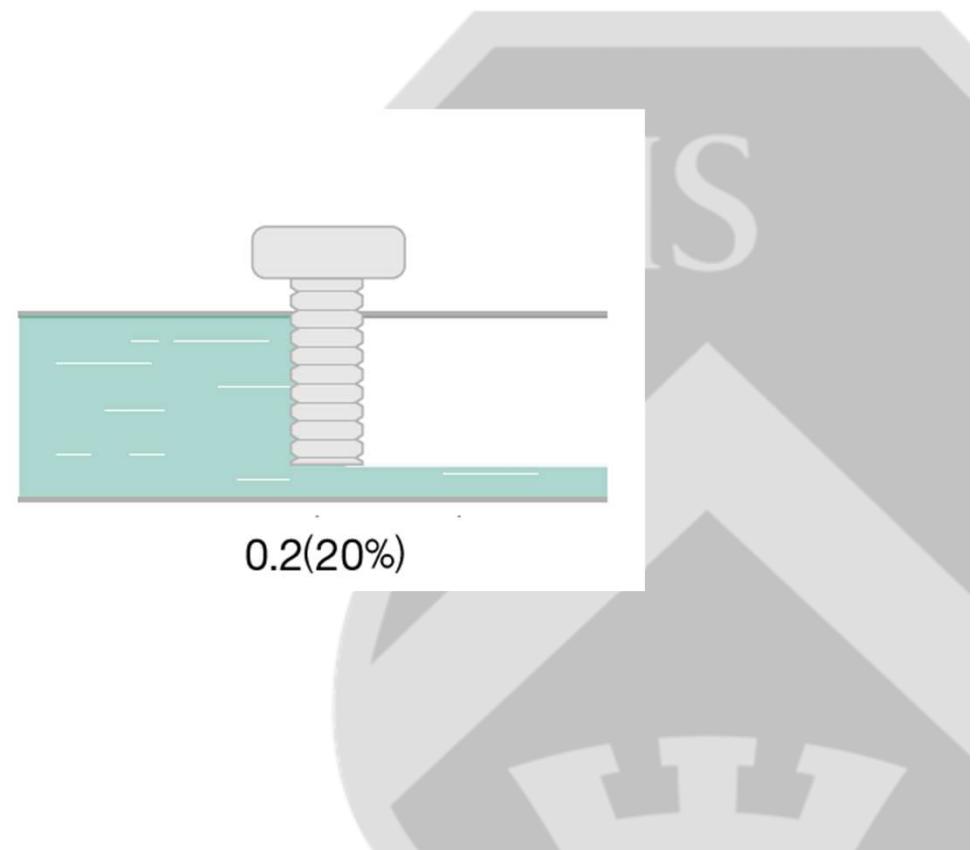
- 게이트

LSTM에서의 게이트 의미 : 데이터의 흐름을 얼마나 내보낼지 제어

- 0.0 ~ 1.0 사이의 실수로 나타낸다. ( 여기서는 시그모이드 사용 )
- 게이트를 얼마나 열지는 학습하면서 조정한다.



0.7(70%)

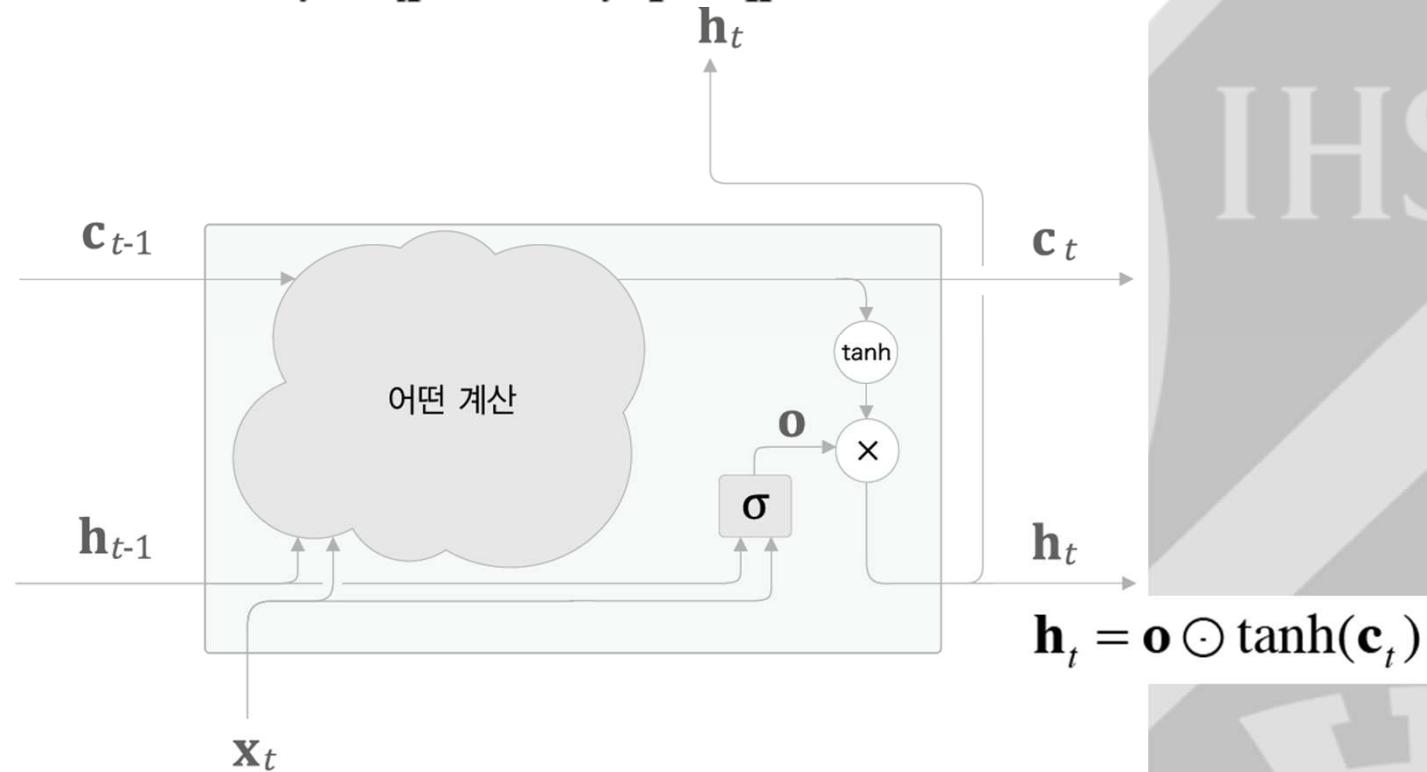


0.2(20%)

# LSTM 구조\_Output Gate

- LSTM 계층 조립하기\_다음 은닉 상태에 무엇이 얼마나 중요할까?
  - Output Gate :  $h_t = \tanh(c_t)$ 에 게이트를 적용하는 것  
즉,  $\tanh(c_t)$ 의 각 원소에 대해 그것이 다음 시각의 은닉 상태에 얼마나 중요한가를 조정한다. ( $h_t$ 의 출력을 담당하는 게이트)

$$\mathbf{o} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(o)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(o)} + \mathbf{b}^{(o)})$$

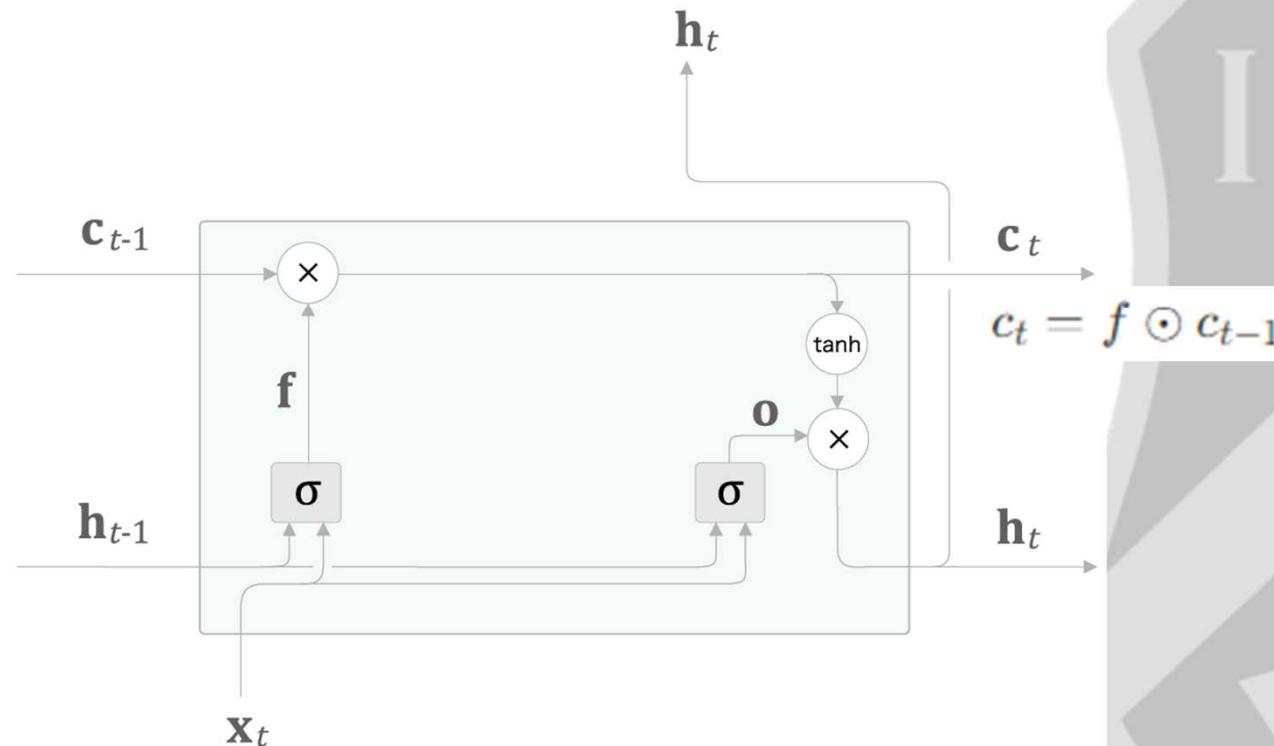


# LSTM 구조\_Forget Gate

- LSTM 계층 조립하기\_무엇을 잊을까? ( Memory Cell 갱신 )
  - Forget Gate :  $c_{t-1}$ 의 기억 중에서 불필요한 기억을 잊게 하는 게이트

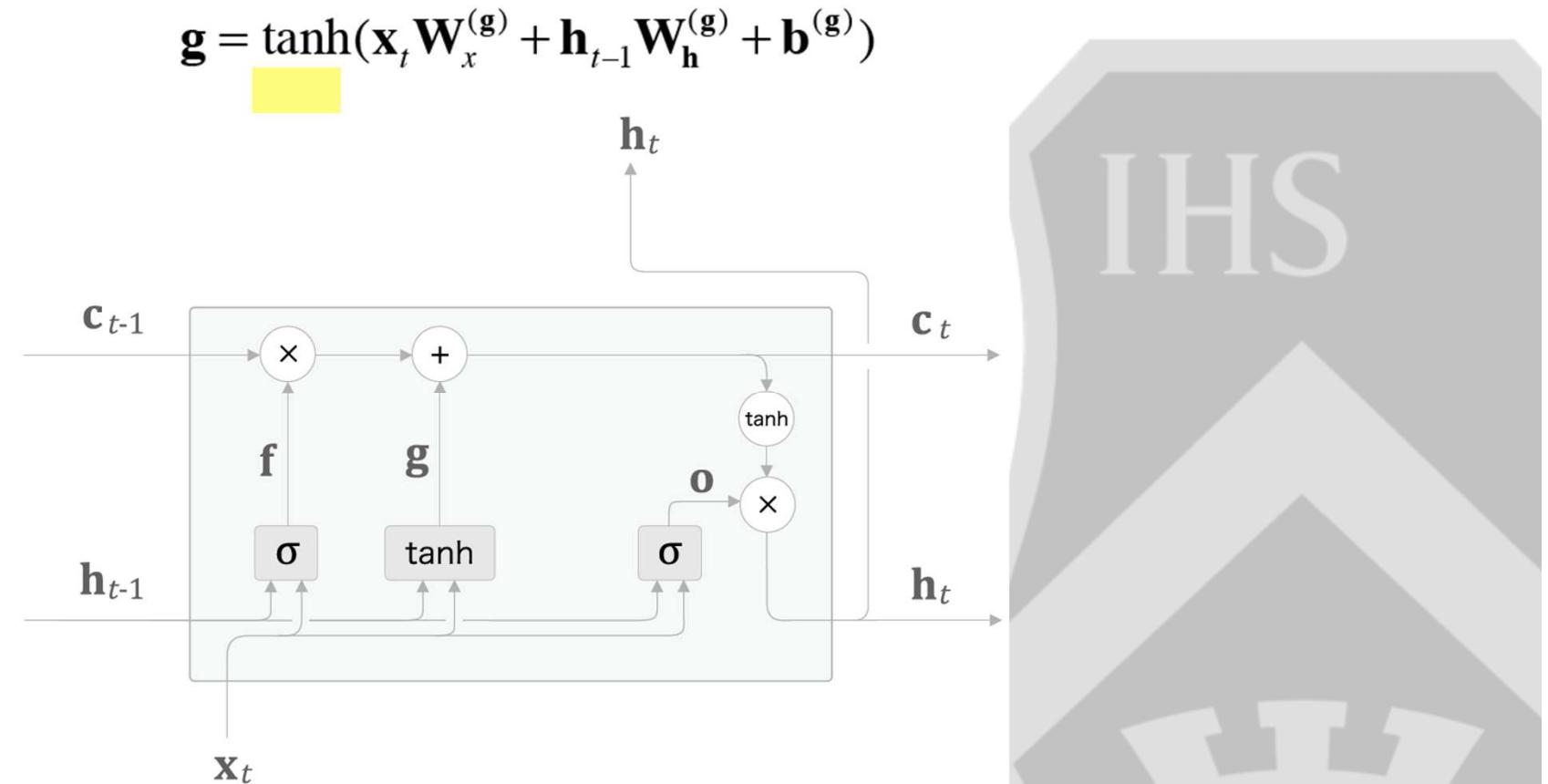
$$\mathbf{f} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(f)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(f)} + \mathbf{b}^{(f)})$$

그림 6-16 forget 게이트 추가



# LSTM 구조\_New Memory (Gate X)

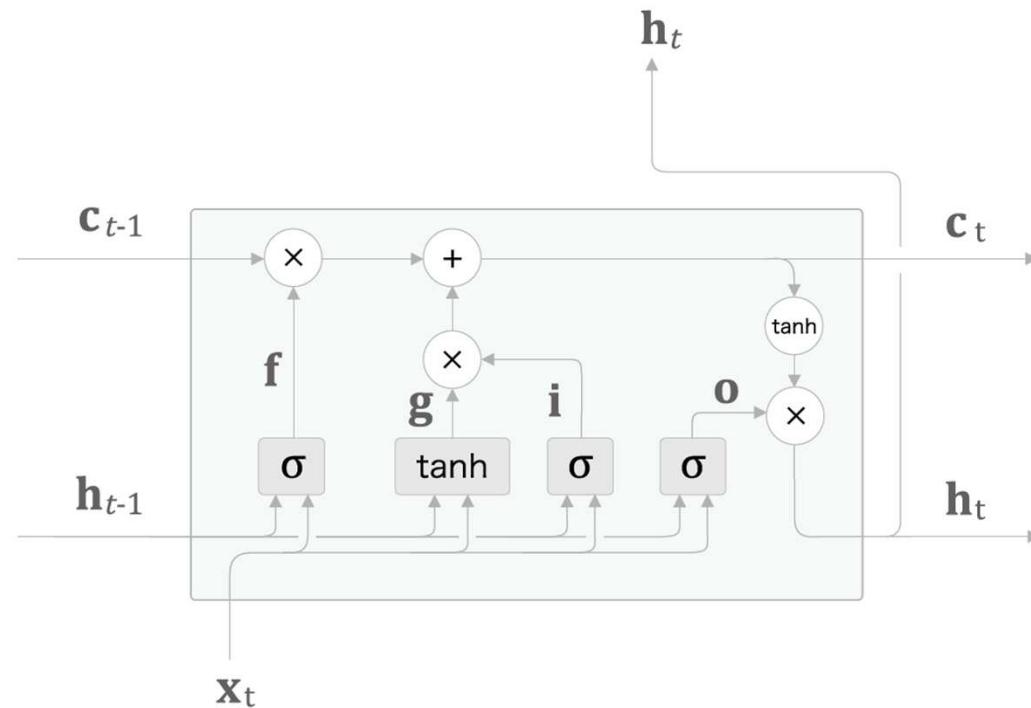
- LSTM 계층 조립하기\_새로운 것을 기억하자( Memory Cell 갱신 )
  - 새로 기억해야 할 정보 : tanh 노드를 추가하여, 이 노드가 계산한 결과가 이전 시각의 셀  $c_{t-1}$ 에 더해진다. 즉, 새로운 정보를 기억 셀에 추가하는 것이 목적이다.



# LSTM 구조\_Input Gate

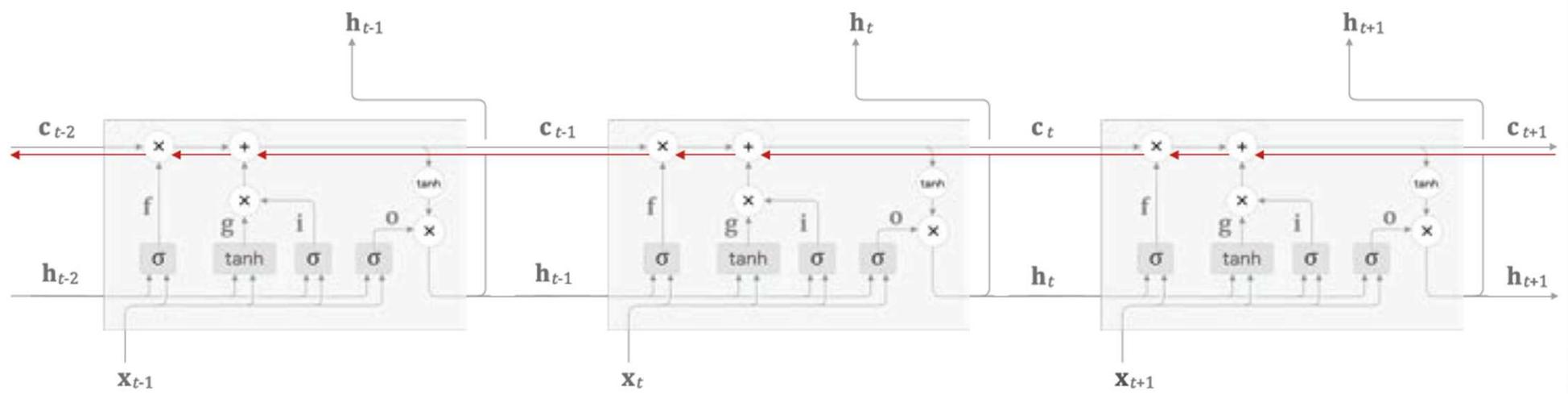
- LSTM 계층 조립하기\_새로운 정보의 가치가 얼마나 되나?  
 - Input Gate : 새로 추가된 정보  $g$ 의 각 원소가 얼마나 가치가 있는지 판단한다. 즉, 무비판적으로 수용하지 않고 적절히 취사선택하는 것이 이 게이트의 역할이다.

$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$



# LSTM의 역할\_기울기 소실 방지

- LSTM의 기울기 흐름\_기억 셀의 역전파
  - + 노드 : 상류 기울기 그대로 전달  $\rightarrow$  변화 x
  - x 노드 : 행렬 곱이 아닌, 아다마르 곱(원소별 곱)을 계산한다. 이 때 매 시각 다른 게이트 값을 이용해 원소별 곱을 계산한다.
  - x 노드의 계산은 forget 게이트가 제어
    - $\rightarrow$  잊어야 한다면, 그 원소에 대해 기울기 약화 o
    - $\rightarrow$  잊지 않아야 한다면, 그 원소에 대해 기울기 약화
    - : 오래 기억해야 할 정보는 소실 없이 전파



# LSTM 구현\_수식 처리

- LSTM 구현 전 수식 처리
  - 각 식의 가중치들을 모아 4개의 식을 단 한번의 Affine 변환, 계산

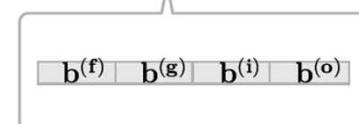
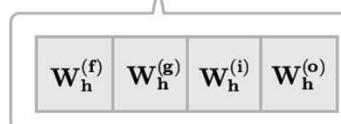
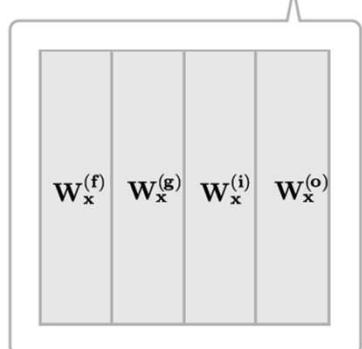
$$\begin{aligned} \mathbf{x}_t \mathbf{W}_x^{(f)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(f)} + \mathbf{b}^{(f)} \\ \mathbf{x}_t \mathbf{W}_x^{(g)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(g)} + \mathbf{b}^{(g)} \\ \mathbf{x}_t \mathbf{W}_x^{(i)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(i)} + \mathbf{b}^{(i)} \\ \mathbf{x}_t \mathbf{W}_x^{(o)} + \mathbf{h}_{t-1} \mathbf{W}_h^{(o)} + \mathbf{b}^{(o)} \end{aligned}$$



$$\mathbf{x}_t \left[ \mathbf{W}_x^{(f)} \ \mathbf{W}_x^{(g)} \ \mathbf{W}_x^{(i)} \ \mathbf{W}_x^{(o)} \right] + \mathbf{h}_{t-1} \left[ \mathbf{W}_h^{(f)} \ \mathbf{W}_h^{(g)} \ \mathbf{W}_h^{(i)} \ \mathbf{W}_h^{(o)} \right] + \left[ \mathbf{b}^{(f)} \ \mathbf{b}^{(g)} \ \mathbf{b}^{(i)} \ \mathbf{b}^{(o)} \right]$$

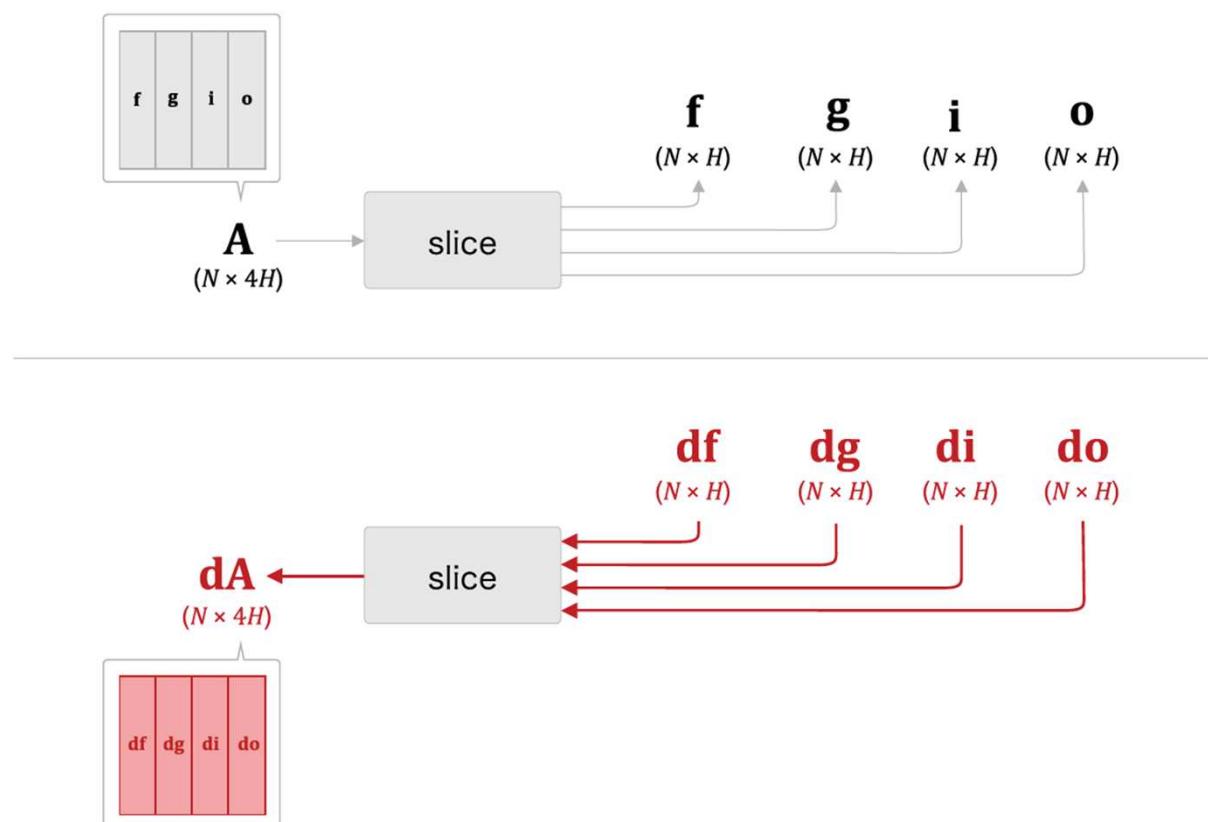


$$\mathbf{x}_t \mathbf{W}_x + \mathbf{h}_{t-1} \mathbf{W}_h + \mathbf{b}$$



# LSTM 구현\_수식 처리

- LSTM 구현 수식 처리
  - 각 식의 가중치들을 모아 4개의 식을 단 한번의 Affine 변환, 계산  
-> slice 노드를 통해 그 4개의 결과를 꺼낸다.
  - slice : 아핀 변환의 결과(행렬)을 균등하게 네 조각으로 나눠서 꺼내주는 단순한 노드



# LSTM 구현

```

96 class LSTM:
97     def __init__(self, Wx, Wh, b):
98         ...
99
100    Parameters
101    -----
102    Wx: 입력 x에 대한 가중치 매개변수(4개분의 가중치가 담겨 있음)
103    Wh: 은닉 상태 h에 대한 가장주 매개변수(4개분의 가중치가 담겨 있음)
104    b: 편향(4개분의 편향이 담겨 있음)
105        ...
106    self.params = [Wx, Wh, b]
107    self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
108    self.cache = None
109
110    def forward(self, x, h_prev, c_prev):
111        Wx, Wh, b = self.params
112        N, H = h_prev.shape
113
114        A = np.dot(x, Wx) + np.dot(h_prev, Wh) + b
115
116        f = A[:, :H]
117        g = A[:, H:2*H]
118        i = A[:, 2*H:3*H]
119        o = A[:, 3*H:]
120
121        f = sigmoid(f)
122        g = np.tanh(g)
123        i = sigmoid(i)
124        o = sigmoid(o)
125
126        c_next = f * c_prev + g * i
127        h_next = o * np.tanh(c_next)
128
129        self.cache = (x, h_prev, c_prev, i, f, g, o, c_next)
130        return h_next, c_next

```

```

def backward(self, dh_next, dc_next):
    Wx, Wh, b = self.params
    x, h_prev, c_prev, i, f, g, o, c_next = self.cache

    tanh_c_next = np.tanh(c_next)

    ds = dc_next + (dh_next * o) * (1 - tanh_c_next ** 2)

    dc_prev = ds * f

    di = ds * g
    df = ds * c_prev
    do = dh_next * tanh_c_next
    dg = ds * i

    di *= i * (1 - i)
    df *= f * (1 - f)
    do *= o * (1 - o)
    dg *= (1 - g ** 2)

    dA = np.hstack((df, dg, di, do))

    dWh = np.dot(h_prev.T, dA)
    dWx = np.dot(x.T, dA)
    db = dA.sum(axis=0)

    self.grads[0][...] = dWx
    self.grads[1][...] = dWh
    self.grads[2][...] = db

    dx = np.dot(dA, Wx.T)
    dh_prev = np.dot(dA, Wh.T)

    return dx, dh_prev, dc_prev

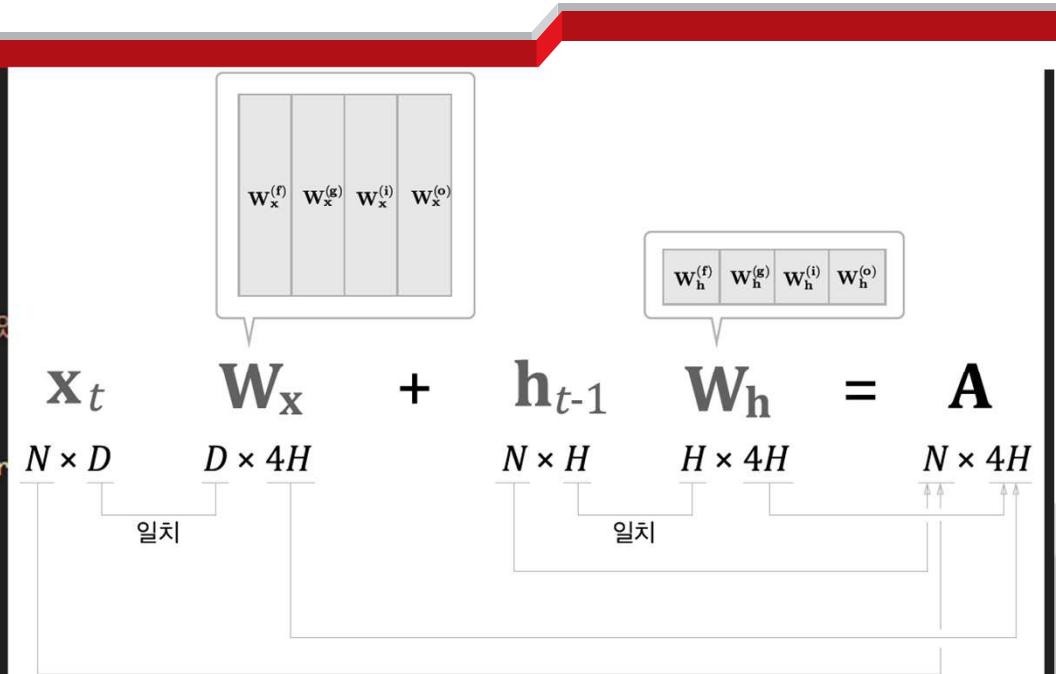
```

# LSTM 구현

```

96  class LSTM:
97      def __init__(self, Wx, Wh, b):
98          ...
99
100     Parameters
101     -----
102     Wx: 입력 x에 대한 가중치 매개변수(4개분의 가중치가 담겨 있음)
103     Wh: 은닉 상태 h에 대한 가장주 매개변수(4개분의 가중치가 담겨 있음)
104     b: 편향(4개분의 편향이 담겨 있음)
105     ...
106     self.params = [Wx, Wh, b]
107     self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
108     self.cache = None
109
110    def forward(self, x, h_prev, c_prev):
111        Wx, Wh, b = self.params
112        N, H = h_prev.shape
113
114        A = np.dot(x, Wx) + np.dot(h_prev, Wh) + b
115
116        f = A[:, :H] 
117        g = A[:, H:2*H]
118        i = A[:, 2*H:3*H]
119        o = A[:, 3*H:]
120
121        f = sigmoid(f)
122        g = np.tanh(g)
123        i = sigmoid(i)
124        o = sigmoid(o)
125
126        c_next = f * c_prev + g * i
127        h_next = o * np.tanh(c_next)
128
129        self.cache = (x, h_prev, c_prev, i, f, g, o, c_next)
130        return h_next, c_next

```



```

do *= o * (1 - o)
dg *= (1 - g ** 2)

dA = np.hstack((df, dg, di, do))

dWh = np.dot(h_prev.T, dA)
dWx = np.dot(x.T, dA)
db = dA.sum(axis=0)

self.grads[0][...] = dWx
self.grads[1][...] = dWh
self.grads[2][...] = db

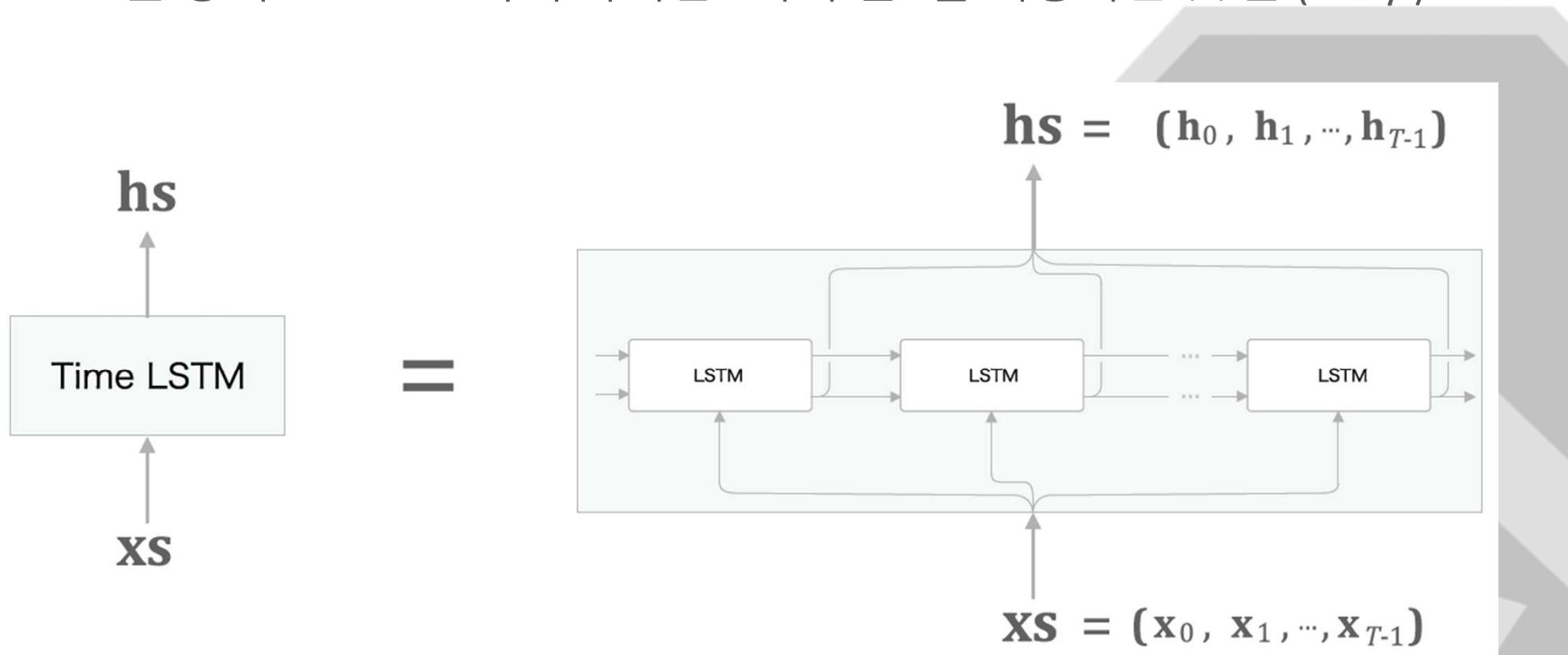
dx = np.dot(dA, Wx.T)
dh_prev = np.dot(dA, Wh.T)

return dx, dh_prev, dc_prev

```

# Time LSTM 구현

- Time LSTM 구현
  - Time LSTM : T개 time step의 시계열 데이터를 한번에 처리하는 계층  
-> T개의 LSTM 계층으로 구성 (이어 붙인 것)
  - 앞 장의 Time RNN과의 차이점 : 기억 셀  $c$ 를 이용하는 것 뿐 ( Easy )



```

class TimeRNN:
    def __init__(self, Wx, Wh, b, stateful=False):
        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
        self.layers = None

        self.h, self.dh = None, None
        self.stateful = stateful

    def forward(self, xs):
        Wx, Wh, b = self.params
        N, T, D = xs.shape
        D, H = Wx.shape

        self.layers = []
        hs = np.empty((N, T, H), dtype='f')

        if not self.stateful or self.h is None:
            self.h = np.zeros((N, H), dtype='f')

        for t in range(T):
            layer = RNN(*self.params)
            self.h = layer.forward(xs[:, t, :], self.h)
            hs[:, t, :] = self.h
            self.layers.append(layer)

        return hs

    def backward(self, dhs):
        Wx, Wh, b = self.params
        N, T, H = dhs.shape
        D, H = Wx.shape

        dxs = np.empty((N, T, D), dtype='f')
        dh = 0
        grads = [0, 0, 0]
        for t in reversed(range(T)):
            layer = self.layers[t]
            dx, dh = layer.backward(dhs[:, t, :] + dh)
            dxs[:, t, :] = dx

            for i, grad in enumerate(layer.grads):
                grads[i] += grad

        for i, grad in enumerate(grads):
            self.grads[i][...] = grad
        self.dh = dh

        return dxs

    def set_state(self, h):
        self.h = h

    def reset_state(self):
        self.h = None

```

```

class TimeLSTM:
    def __init__(self, Wx, Wh, b, stateful=False):
        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
        self.layers = None

        self.h, self.c = None, None
        self.dh = None
        self.stateful = stateful

    def forward(self, xs):
        Wx, Wh, b = self.params
        N, T, D = xs.shape
        H = Wh.shape[0]

        self.layers = []
        hs = np.empty((N, T, H), dtype='f')

        if not self.stateful or self.h is None:
            self.h = np.zeros((N, H), dtype='f')
        if not self.stateful or self.c is None:
            self.c = np.zeros((N, H), dtype='f')

        for t in range(T):
            layer = LSTM(*self.params)
            self.h, self.c = layer.forward(xs[:, t, :], self.h, self.c)
            hs[:, t, :] = self.h
            self.layers.append(layer)

        return hs

    def backward(self, dhs):
        Wx, Wh, b = self.params
        N, T, H = dhs.shape
        D = Wx.shape[0]

        dxs = np.empty((N, T, D), dtype='f')
        dh, dc = 0, 0
        grads = [0, 0, 0]
        for t in reversed(range(T)):
            layer = self.layers[t]
            dx, dh, dc = layer.backward(dhs[:, t, :] + dh, dc)
            dxs[:, t, :] = dx
            for i, grad in enumerate(layer.grads):
                grads[i] += grad

        for i, grad in enumerate(grads):
            self.grads[i][...] = grad
        self.dh = dh

        return dxs

    def set_state(self, h, c=None):
        self.h, self.c = h, c

    def reset_state(self):
        self.h, self.c = None, None

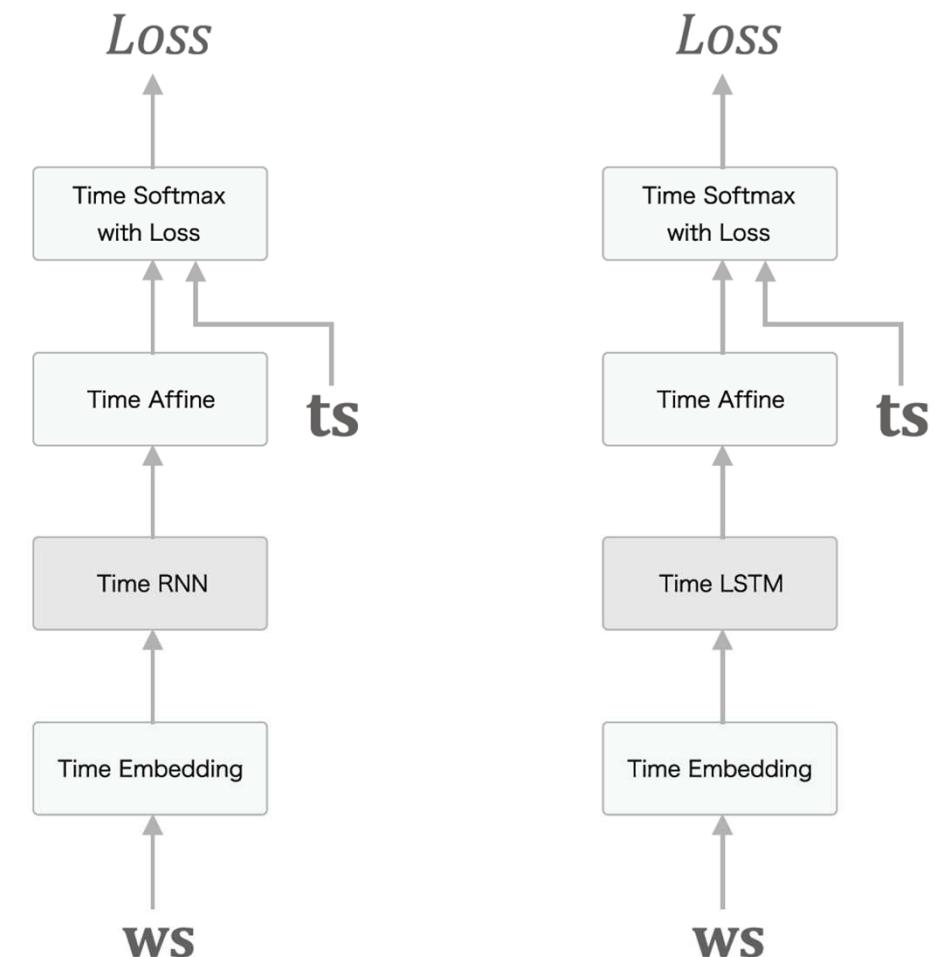
```

# LSTM을 사용한 언어 모델

- LSTM 계층을 사용한 언어 모델
  - 앞 장의 SimpleRnnlm 클래스( RNN 계층을 사용한 언어모델 ) 와의 유일한 차이점 :

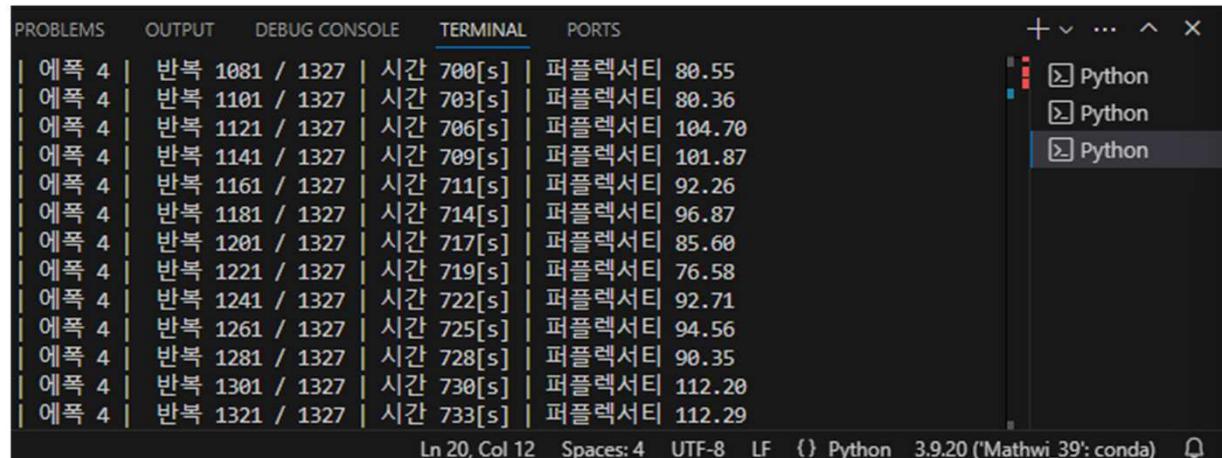
Time RNN 대신, Time LSTM 사용

왼쪽 : SimpleRnnlm  
 오른쪽 : Rnnlm



# LSTM을 사용한 언어 모델\_Rnnlm

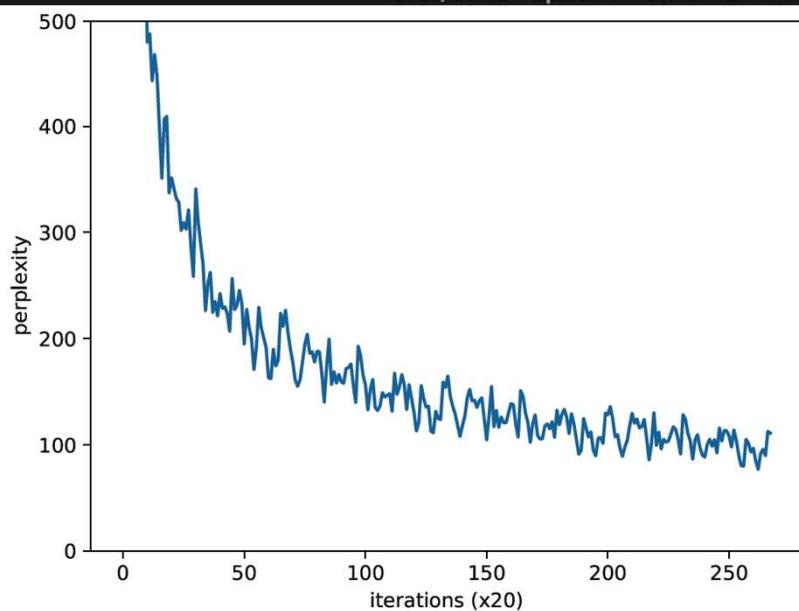
- LSTM 계층을 사용한 언어 모델
  - Rnnlm을 이용한 PTB 데이터셋 (단어 10,000개) 학습



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

	에폭	반복	시간	퍼플렉서티
4	4	1081 / 1327	700[s]	80.55
4	4	1101 / 1327	703[s]	80.36
4	4	1121 / 1327	706[s]	104.70
4	4	1141 / 1327	709[s]	101.87
4	4	1161 / 1327	711[s]	92.26
4	4	1181 / 1327	714[s]	96.87
4	4	1201 / 1327	717[s]	85.60
4	4	1221 / 1327	719[s]	76.58
4	4	1241 / 1327	722[s]	92.71
4	4	1261 / 1327	725[s]	94.56
4	4	1281 / 1327	728[s]	90.35
4	4	1301 / 1327	730[s]	112.20
4	4	1321 / 1327	733[s]	112.29

Ln 20, Col 12 Spaces: 4 UTF-8 {} Python 3.9.20 ('Mathwi\_39': conda)



```

1 # coding: utf-8
2 import sys
3 sys.path.append('C:/Users/1408_Mathwi/Desktop/석사 과정 자기개발/deep'
4 from common.optimizer import SGD
5 from common.trainer import RnnlmTrainer
6 from common.util import eval_perplexity
7 from dataset import ptb
8 from rnnlm import Rnnlm

# 하이퍼파라미터 설정
batch_size = 20
wordvec_size = 100
hidden_size = 100 # RNN의 은닉 상태 벡터의 원소 수
time_size = 35 # RNN을 펼치는 크기
lr = 20.0
max_epoch = 4
max_grad = 0.25

# 학습 데이터 읽기
corpus, word_to_id, id_to_word = ptb.load_data('train')
corpus_test, _, _ = ptb.load_data('test')
vocab_size = len(word_to_id)
xs = corpus[:-1]
ts = corpus[1:]

# 모델 생성
model = Rnnlm(vocab_size, wordvec_size, hidden_size)
optimizer = SGD(lr)
trainer = RnnlmTrainer(model, optimizer)

# 기울기 클리핑을 적용하여 학습
trainer.fit(xs, ts, max_epoch, batch_size, time_size, max_grad,
            eval_interval=20)
trainer.plot(ylim=(0, 500))

# 테스트 데이터로 평가
model.reset_state()
ppl_test = eval_perplexity(model, corpus_test)
print('테스트 퍼플렉서티: ', ppl_test)

# 매개변수 저장
model.save_params()

```

# LSTM을 사용한 언어 모델\_Rnnlm

- Rnnlm 결과 분석
  - 1 epoch 때 10,000에 가까운 숫자로, 학습이 안된 상태에서 시작하여 마지막에는 100에 가까운 숫자로 떨어졌다.

하지만 실제 발전된 모델들은 60까지도 감소하는게 일반적이다.  
성능을 높일 수 있는 개선법에 대해 살펴보자.



# Rnnlm 추가 개선

- Rnnlm 추가 개선법

## 1. LSTM 계층 다층화

: LSTM 계층을 깊게 쌓아 복잡한 패턴을 학습 할 수 있게 한다.  
( 몇 층을 쌓을지는 하이퍼파라미터로, 엔지니어링의 영역 )  
-> 과적합 발생 가능성이 높아진다.

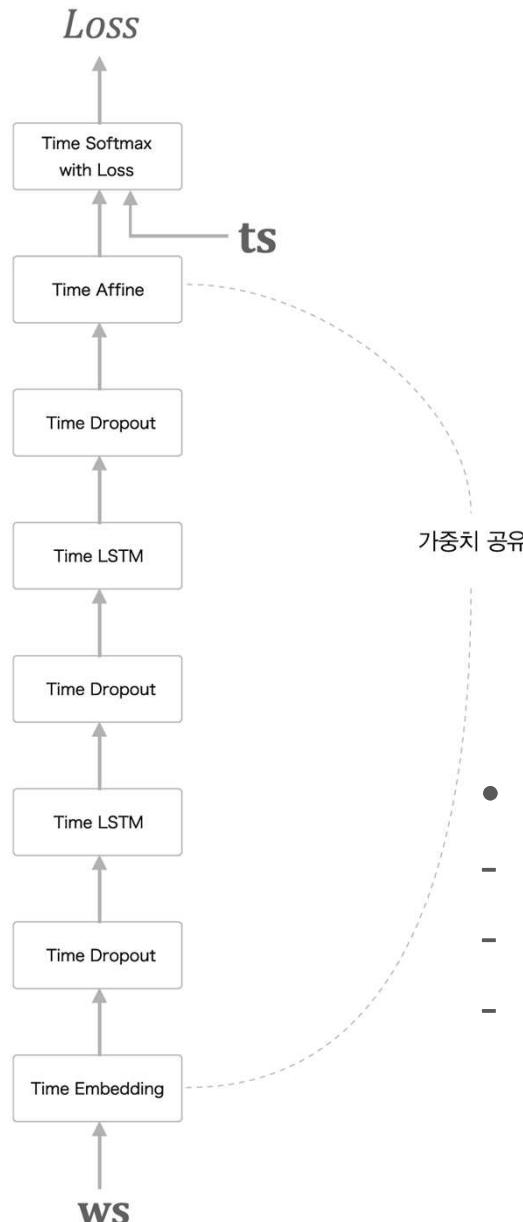
## 2. 드롭 아웃

: 훈련 시 계층 내의 뉴런 몇 개를 무작위로 무시하고  
계층으로부터의 신호 전달을 막는다.  
-> 과적합을 막는 방법론으로 사용

## 3. 가중치 공유

: Embedding 계층과 Affine 계층의 가중치를 연결시  
매개변수 수를 줄여 속도와 과적합 이슈를 억제할

# Rnnlm 추가 개선\_구현



```
# 세가지 개선점
self.layers = [
    TimeEmbedding(embed_W),
    TimeDropout(dropout_ratio),
    TimeLSTM(lstm_Wx1, lstm_Wh1, lstm_b1, stateful=True),
    TimeDropout(dropout_ratio),
    TimeLSTM(lstm_Wx2, lstm_Wh2, lstm_b2, stateful=True),
    TimeDropout(dropout_ratio),
    TimeAffine(embed_W.T, affine_b) # weight tying!!
]
```

- 세가지 개선점
  - Time LSTM을 2개 겹친다.
  - 사이 사이에 드롭아웃 계층을 사용한다
  - Embedding 계층과 Affine 계층에서 가중치를 공유한다

# 게이트가 추가된 RNN 정리

- 정리
  - 단순한 RNN 학습에서는 기울기 소실과 기울기 폭발이 문제가 된다
  - 기울기 폭발에는 기울기 클리핑, 소실에는 게이트가 추가된 RNN이 효과적이다.
  - LSTM에는 input, forget, output 게이트 가 있다.
  - 게이트에는 전용 가중치가 있고, 시그모이드 함수를 사용하여 0~1 사이의 실수를 출력한다
  - 언어모델 개선에는 LSTM 계층 다층화, 드롭아웃, 가중치 공유 등의 기법이 효과적이다.

