6. 게이트가 추가된 RNN

6.1 RNN의 문제점

6.2 기울기 소실과 LSTM

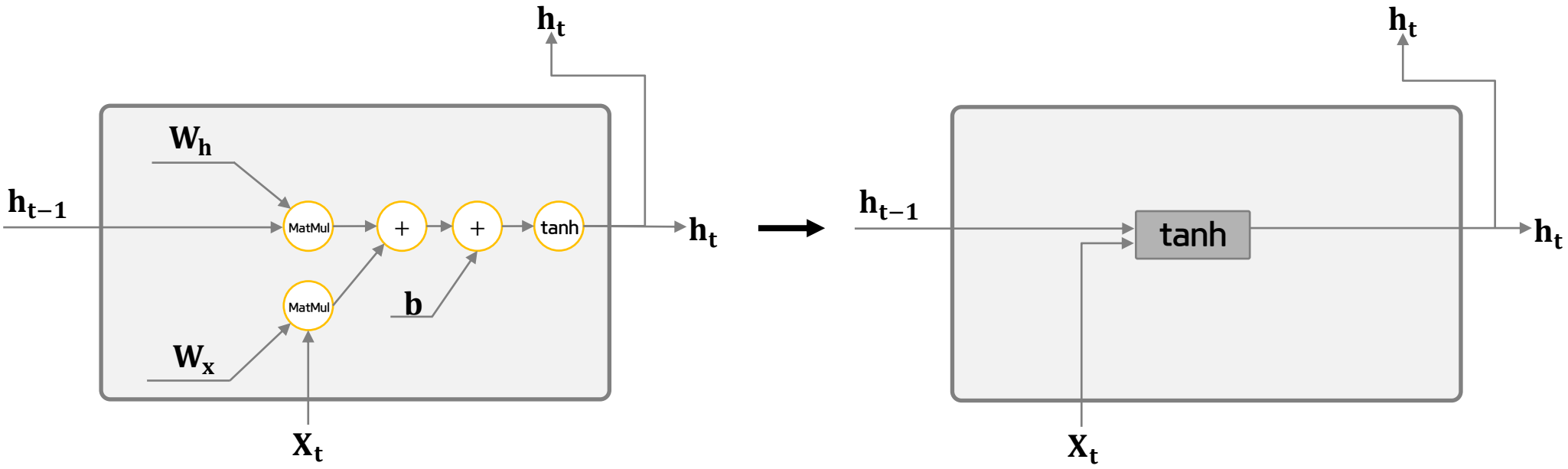
6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

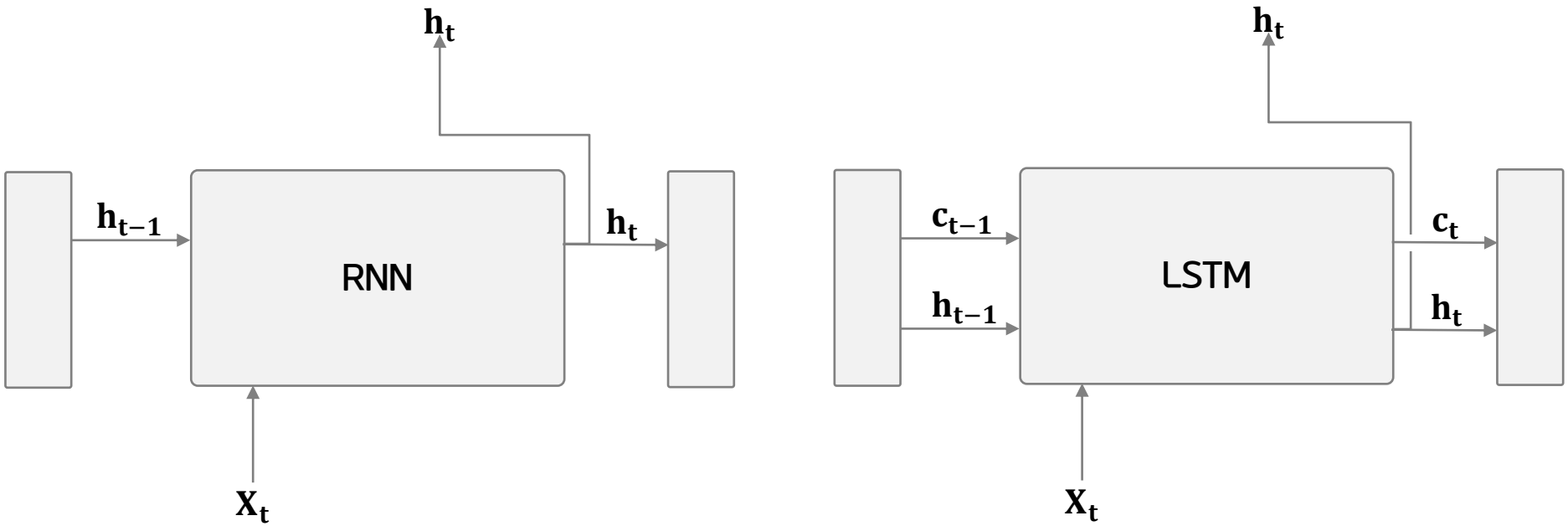
RNN 학습에서 기울기 소실도 큰 문제이다.
이 문제를 해결하려면 RNN 계층의 아키텍처를 근본부터 뜯어고쳐야 한다.

단순화한 도법을 적용한 RNN 계층



여기서는 $\tanh(h_{t-1}W_h + x_tW_x + b)$ 계산을 tanh라는 직사각형 노드 하나로
그리고 직사각형 노드 안에 행렬 곱과 편향의 합, 그리고 tanh 함수에 의한 변환이 모두 포함된 것이다.

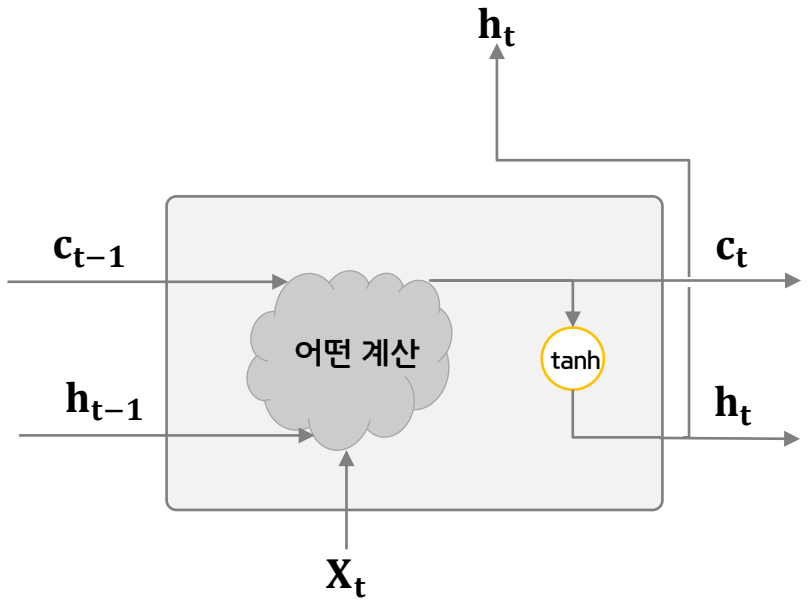
RNN 계층과 LSTM 계층 비교



이 그림에서는 LSTM 계층의 인터페이스에는 c라는 경로가 있다는 차이가 있다.
여기서 c를 기억 셀이라 하며 LSTM 전용의 기억 메커니즘이다.

기억 셀의 특징은 데이터를 LSTM 계층 내에서만 주고받는다는 것이다.
다른 계층으로는 출력하지 않는다는 것이다. 반면, LSTM의 은닉 상태 h는
RNN 계층과 마찬가지로 다른 계층, 위쪽으로 출력된다.

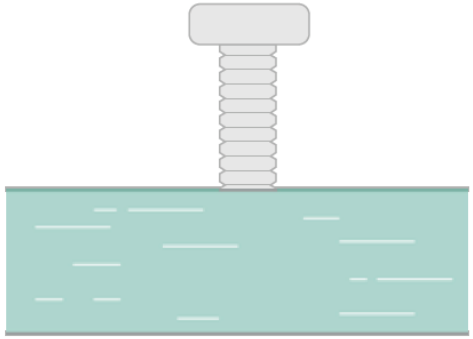
기억 셀 c_t 를 바탕으로 은닉 상태 h_t 를 계산하는 LSTM 계층

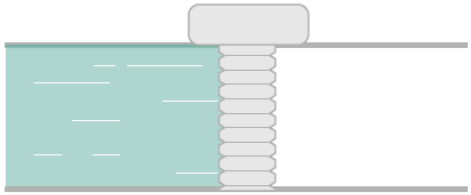


그림에서 현재의 기억 셀 c_t 는 3개의 입력 (c_{t-1}, h_{t-1}, X_t)으로부터
'어떤 계산'을 수행하여 구할 수 있다.
여기서 핵심은 갱신된 c_t 를 사용해 은닉 상태 h_t를 계산한다는 것이다.
또한 이 계산은 h_t =tanh(c_t)인데, 이는 c_t 의 각 요소에 tanh 함수를 적용한다는 뜻이다.

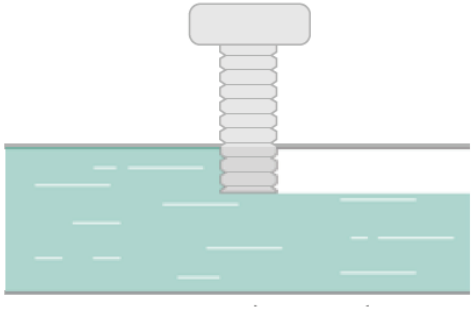
'게이트'란 우리나라로 '문'을 의미하는 단어이다.
문은 열거나 닫을 수 있듯이, 게이트는 데이터의 흐름을 제어한다.

비유하자면 게이트는 물의 흐름을 제어한다.

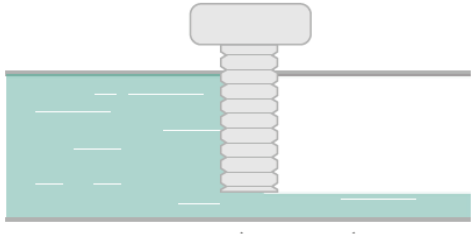




물이 흐르는 양을 0.0~1.0 범위에서 제어한다.



0.7(70%)



0.2(20%)

$\tanh(c_t)$의 각 원소에 대해 '그것이 다음 시각의 은닉 상태에 얼마나 중요한가'를 조정한다.
한편, 이 게이트는 다음 은닉 상태 h_t의 출력을 담당하는 게이트이므로 output 게이트라고 한다.

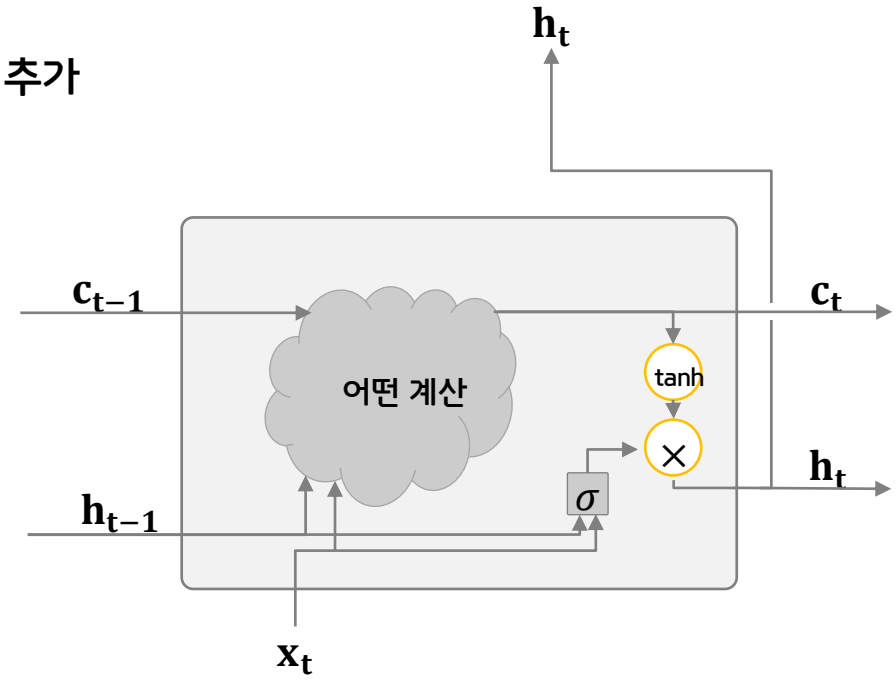
output 게이트의 열림 상태는 입력x_t와 이전 상태 h_{t-1}로부터 구한다.

이때의 식은 밑에 식과 같다.

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

밑에 그림에서 output 게이트에서 수행하는 식의 계산을 sigma로 표기했다.
sigma의 출력을 o라고 하면 $\mathbf{h_t}$ 는 o와 $\mathbf{tanh(c_t)}$의 곱으로 계산된다.

output 게이트 추가

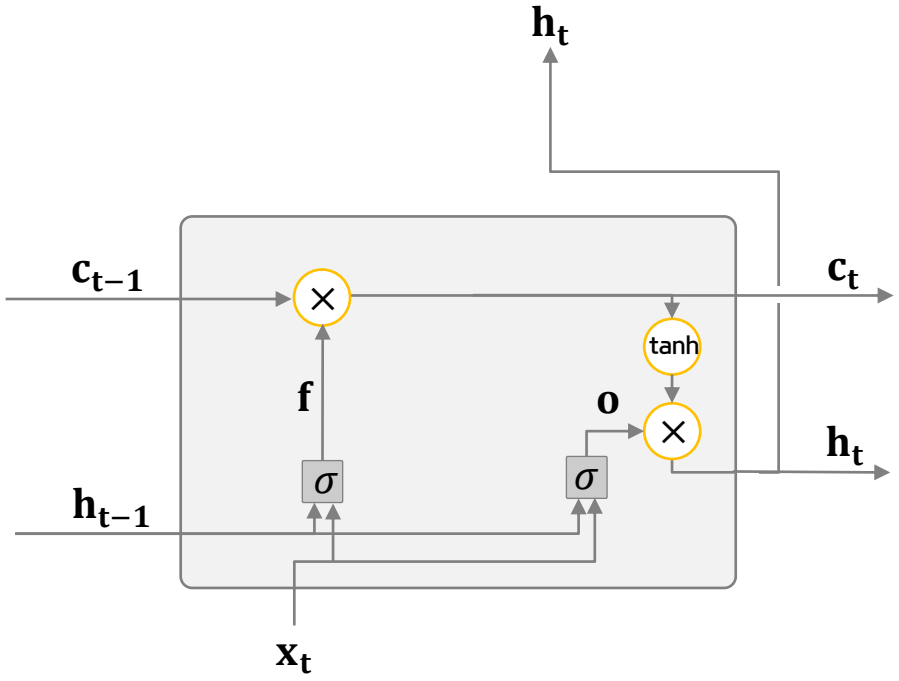


아다마르 곱(Hadamard product)

$$h_t = o \odot \tanh(c_t)$$

다음에 해야 할 일은 기억 셀에 '무엇을 잊을까'를 명확하게 지시하는 것이다.

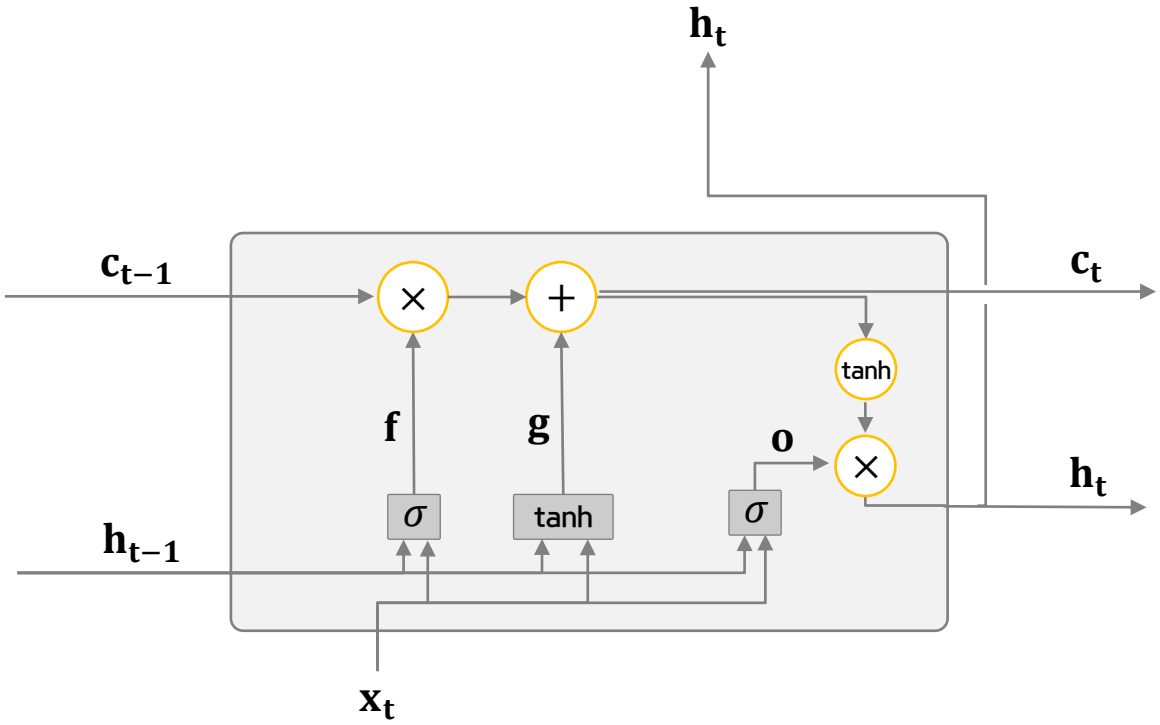
forget 게이트 추가



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

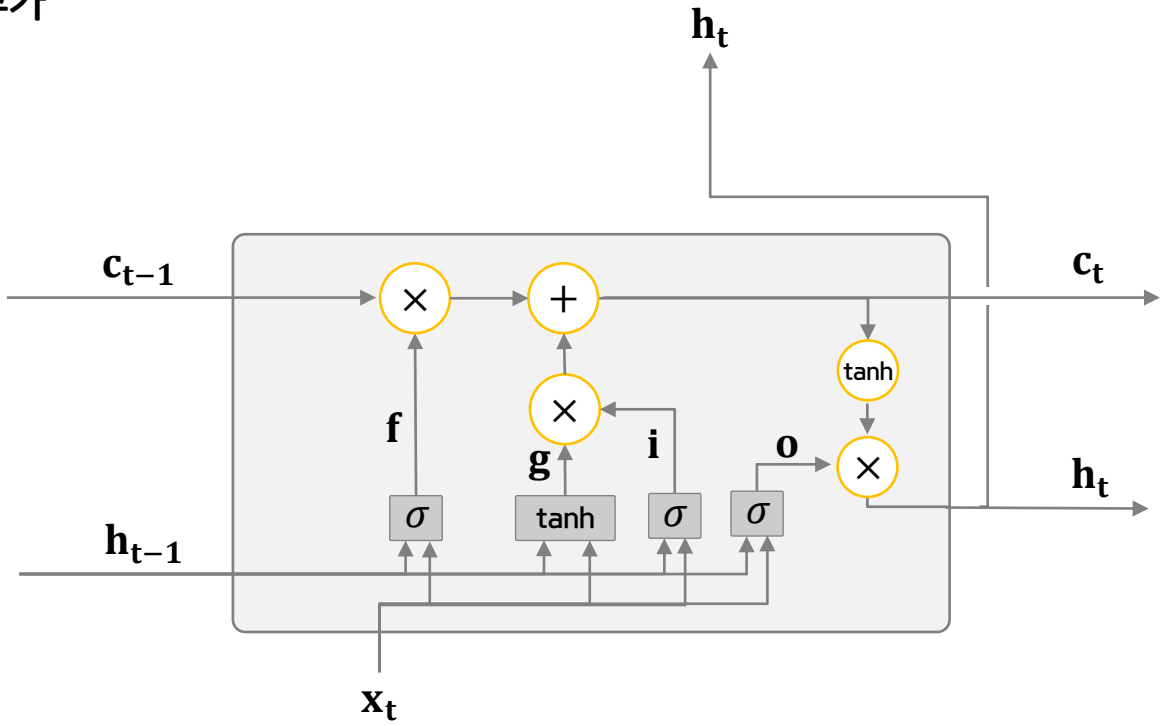
forget 게이트를 거치면서 이전 시각의 기억 셀로부터 잊어야 할 기억이 삭제되었다.

새로운 기억 셀



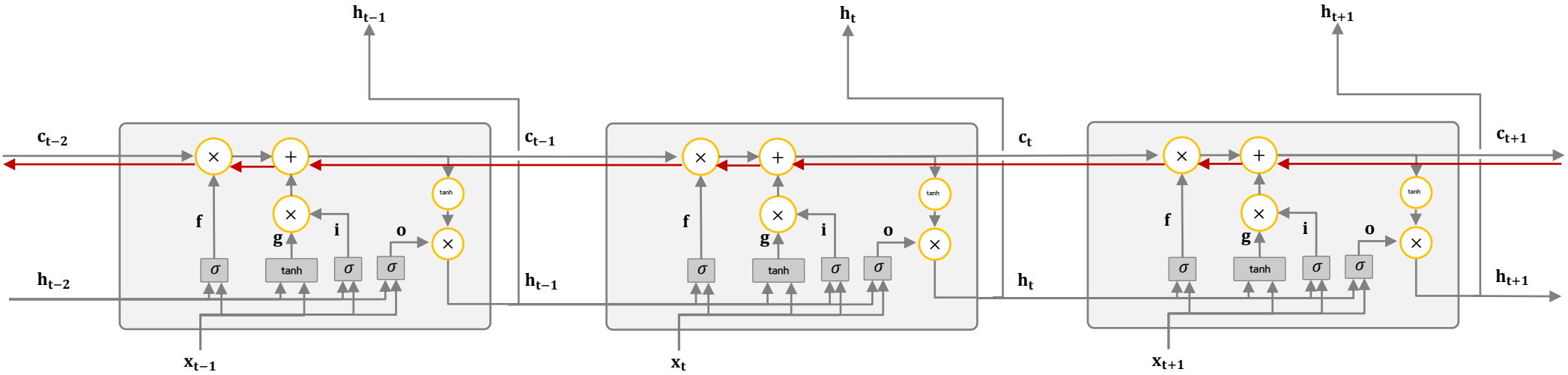
$$g = tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$$

input 게이트 추가



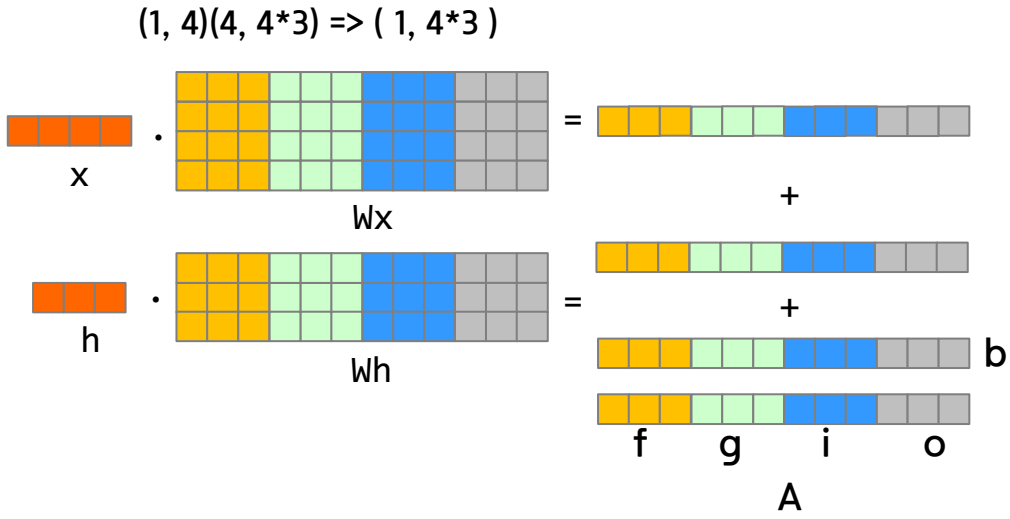
$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$

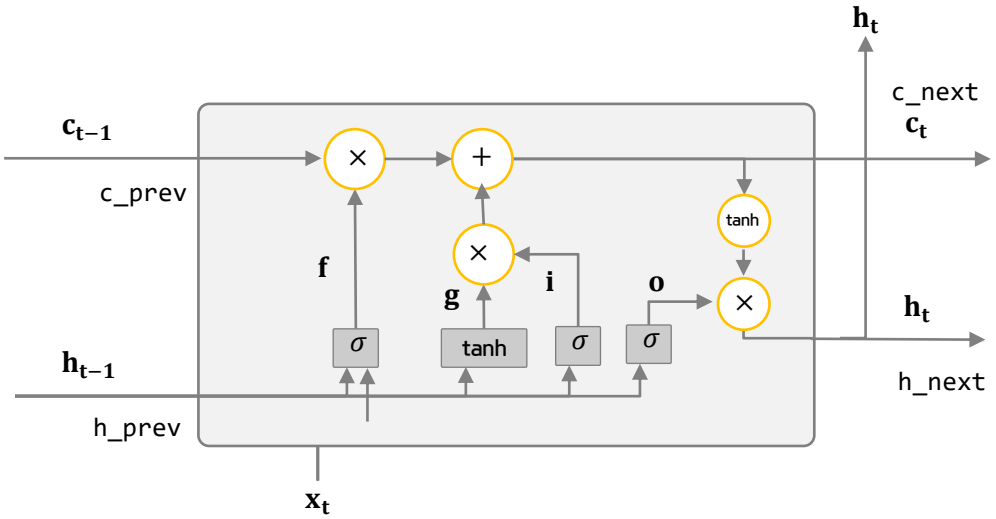
기억 셀의 역전파



forward 분석

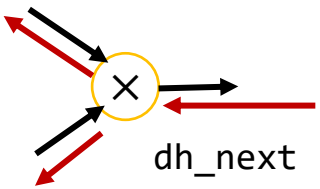
```
def forward(self, x, h_prev, c_prev):
    Wx, Wh, b = self.params
    N, H = h_prev.shape

    A = np.dot(x, Wx) + np.dot(h_prev, Wh) + b

    f = A[:, :H]
    g = A[:, H:2*H]
    i = A[:, 2*H:3*H]
    o = A[:, 3*H:]

    f = sigmoid(f)
    g = np.tanh(g)
    i = sigmoid(i)
    o = sigmoid(o)

    c_next = f * c_prev + g * i
    h_next = o * np.tanh(c_next)

    self.cache = (x, h_prev, c_prev, i, f, g, o, c_next)
    return h_next, c_next
```

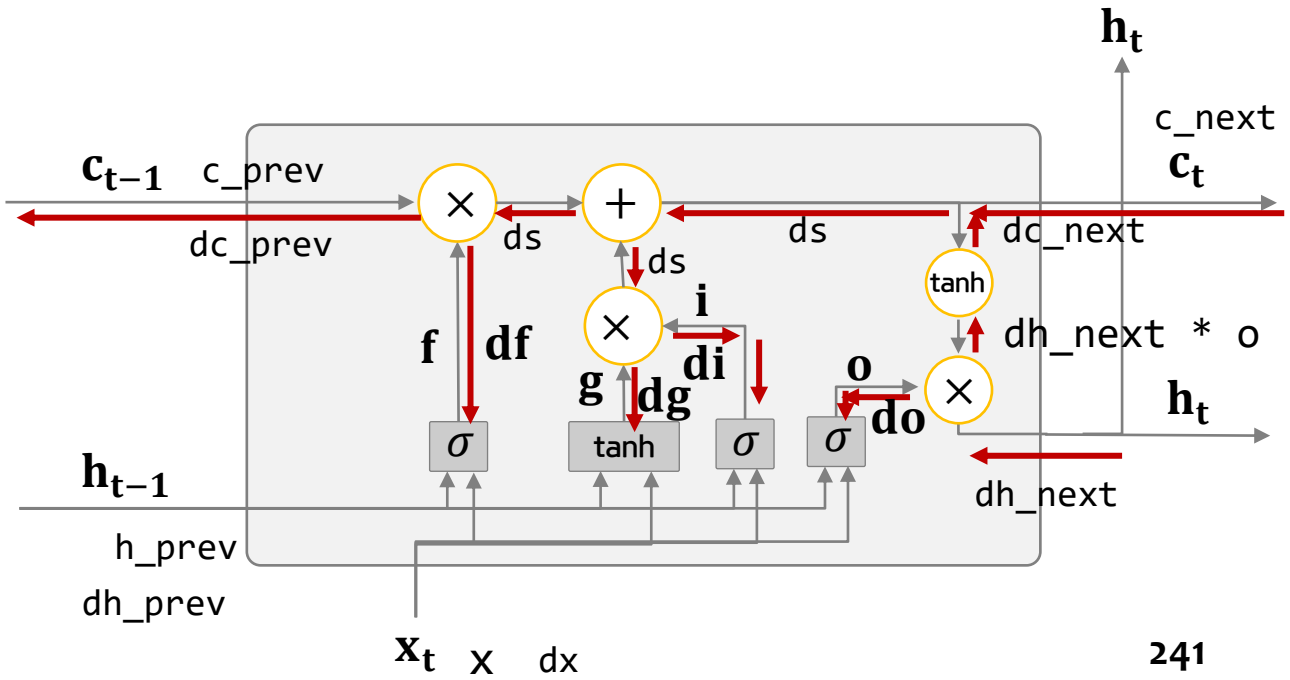




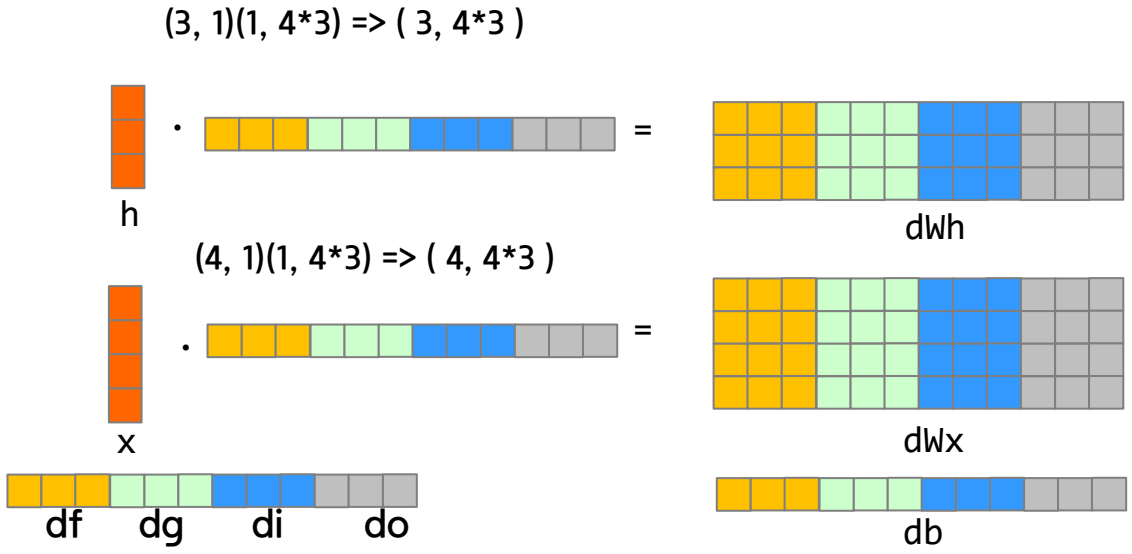
backward 분석

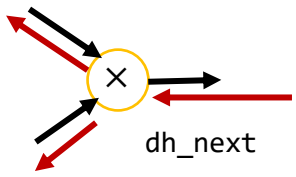
```
def backward(self, dh_next, dc_next):
        Wx, Wh, b = self.params
        x, h_prev, c_prev, i, f, g, o, c_next = self.cache

        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
```

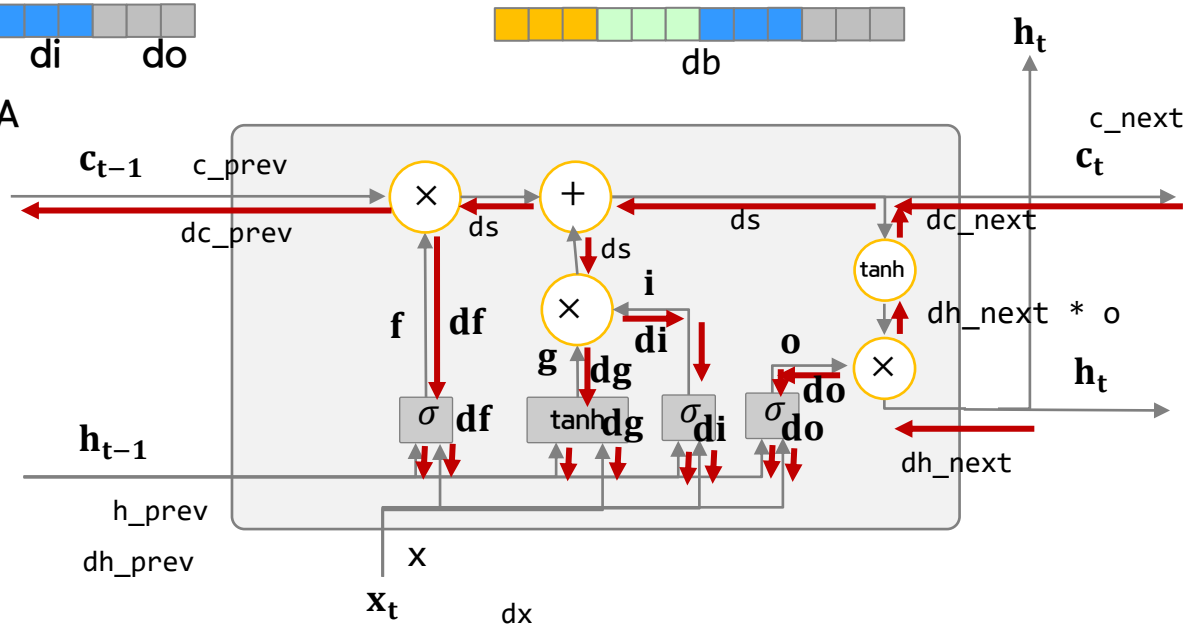


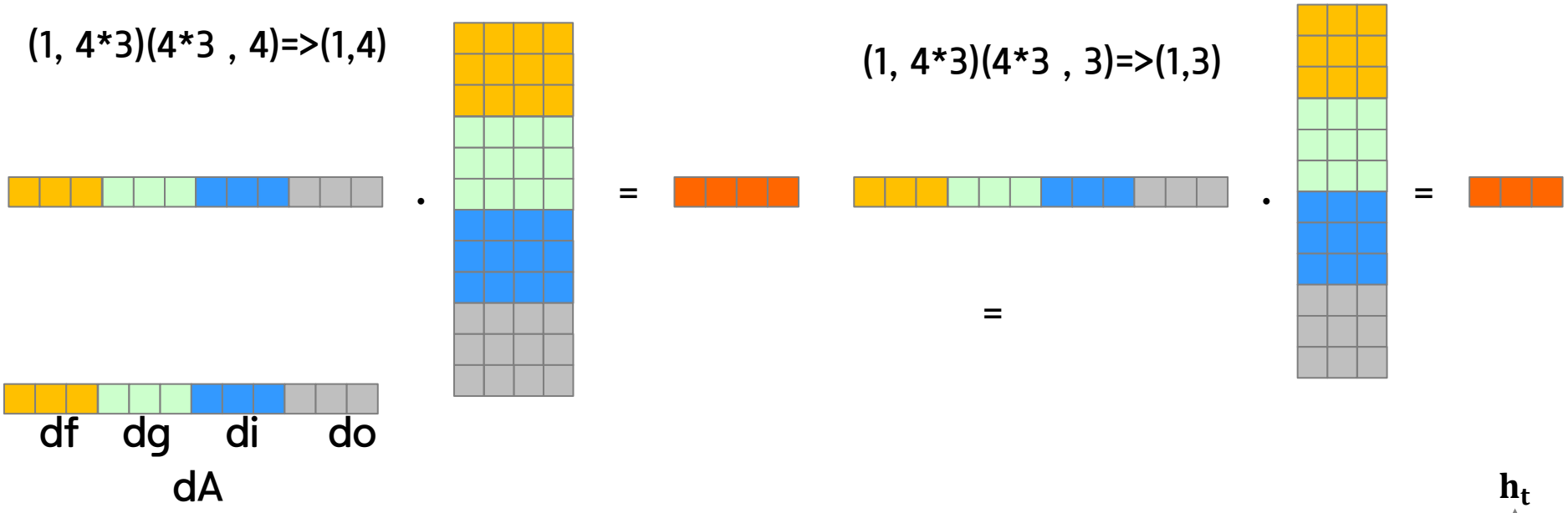


```
di *= i * (1 - i)
df *= f * (1 - f)
do *= o * (1 - o)
dg *= (1 - g ** 2)

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
```

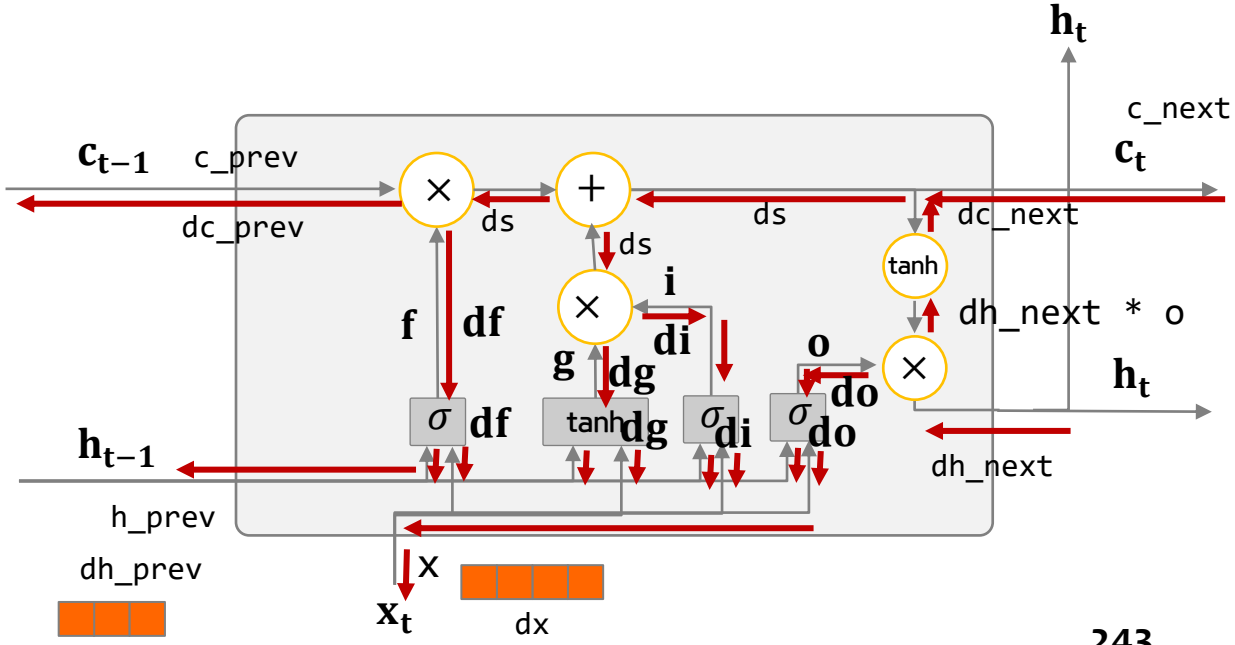









```
dx = np.dot(dA, Wx.T)
dh_prev = np.dot(dA, Wh.T)

return dx, dh_prev, dc_prev
```



6. 게이트가 추가된 RNN

6.1 RNN의 문제점

6.2 기울기 소실과 LSTM

6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

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

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

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

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

$$c_t = f \odot c_{t-1} + g \odot i$$

$$h_t = o \odot \tanh(c_t)$$

각 식의 가중치들을 모아 4개의 식을 단 한 번의 아핀 변환으로 계산

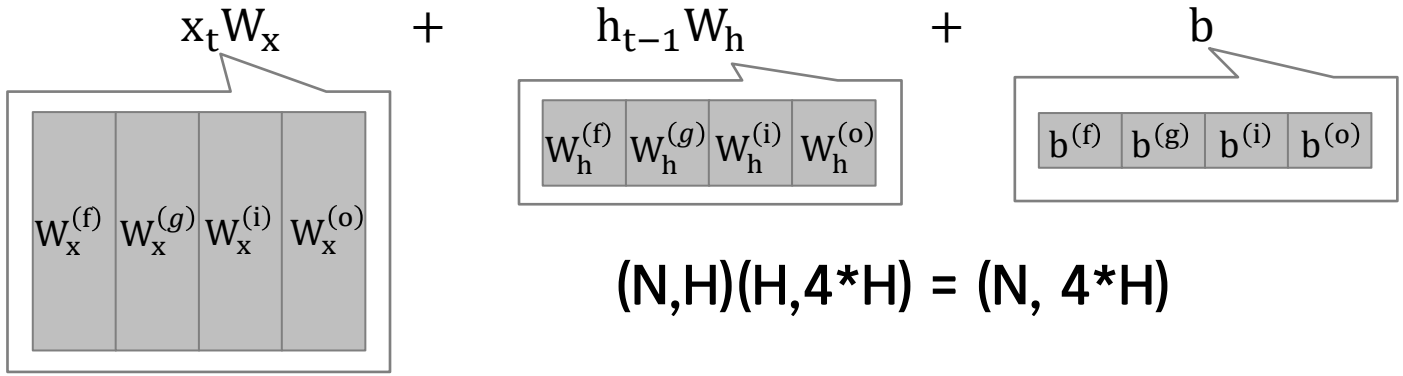
$$x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)}$$
$$x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)}$$
$$x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)}$$
$$x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)}$$

↓

$$x_t \left[W_x^{(f)}\ W_x^{(g)}\ W_x^{(i)}\ W_x^{(o)}\right] + h_{t-1} \left[W_h^{(f)}\ W_h^{(g)}\ W_h^{(i)}\ W_h^{(o)}\right] + \left[b^{(f)}\ b^{(g)}\ b^{(i)}\ b^{(o)}\right]$$

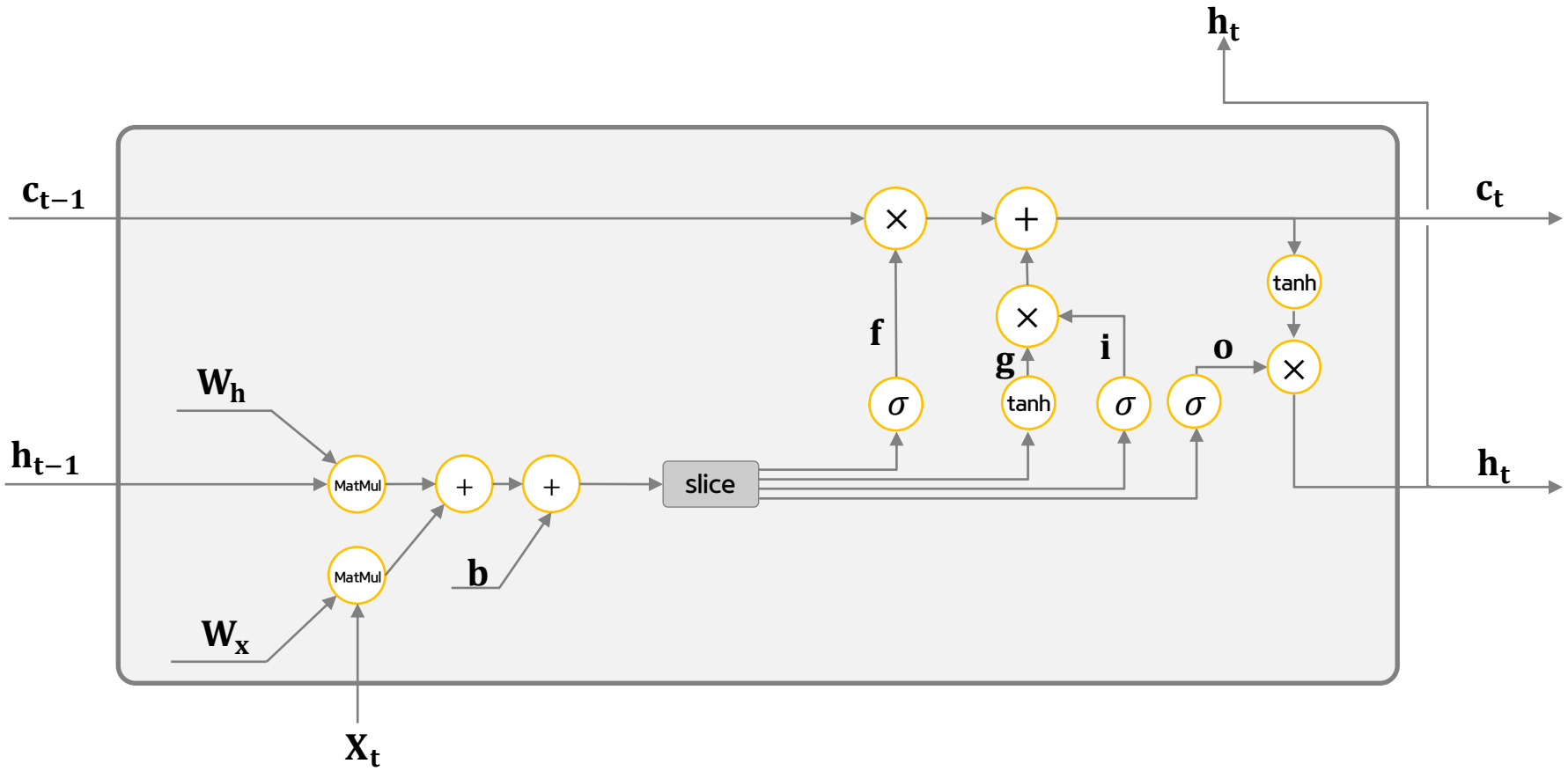
↓

(N,D)(D,4*H) = (N, 4*H)



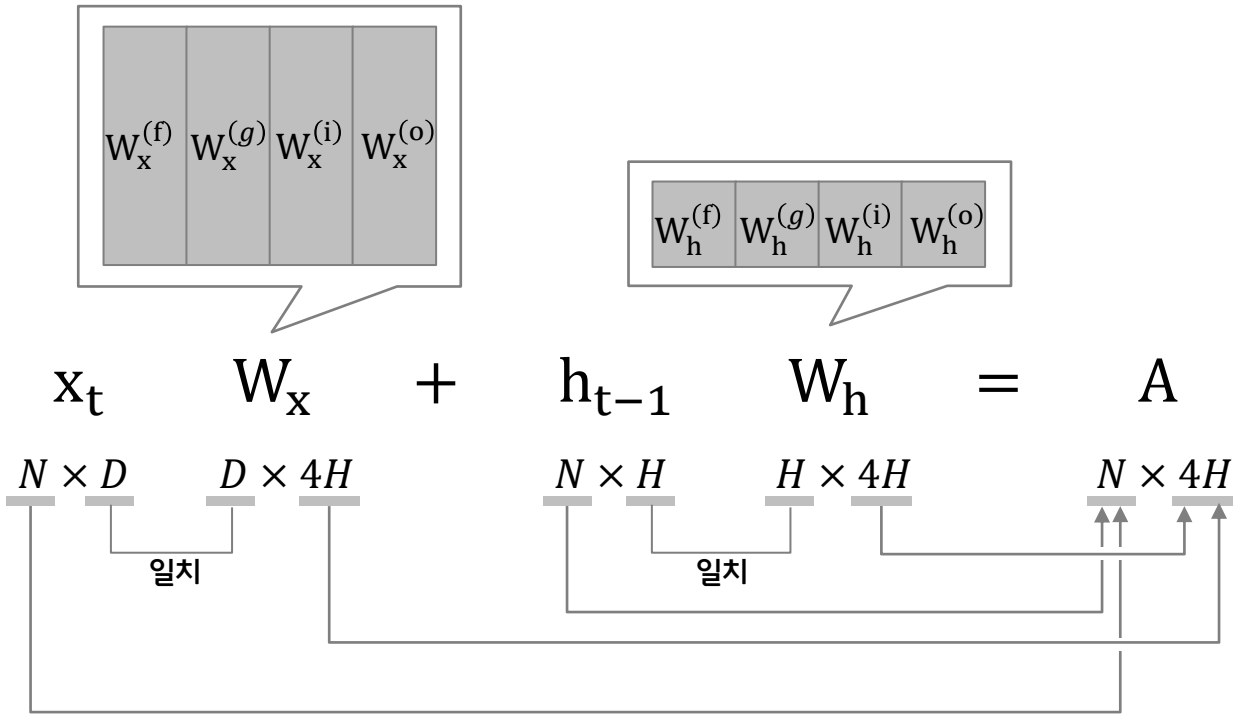
(N,H)(H,4*H) = (N, 4*H)

4개분의 가중치를 모아 아핀 변환을 수행하는 LSTM의 계산 그래프

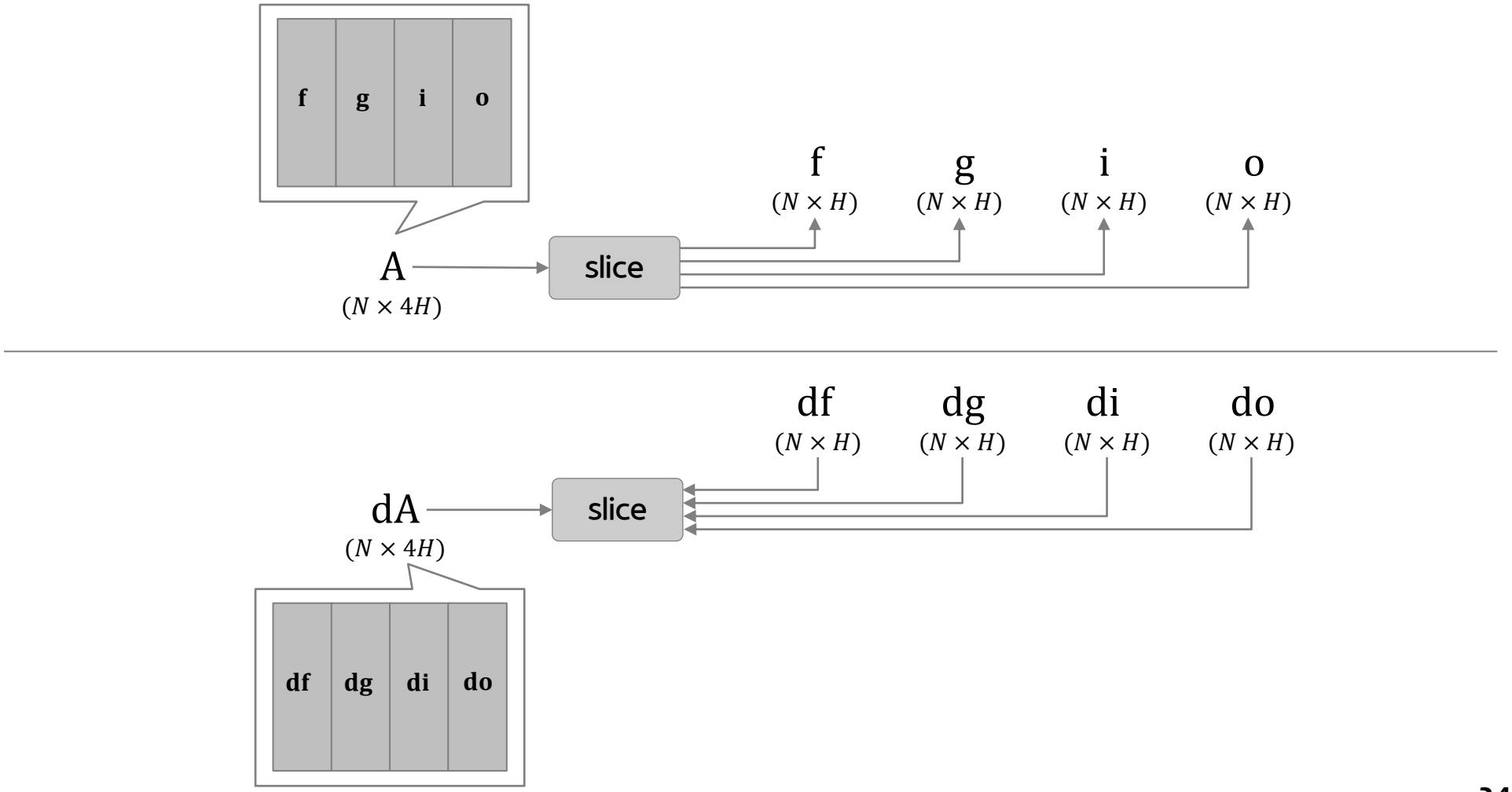


만약 W_x, W_h, b 각각에 4개분의 가중치가 포함되어 있다고 가정하면 위의 그림처럼 그래프가 그려진다.

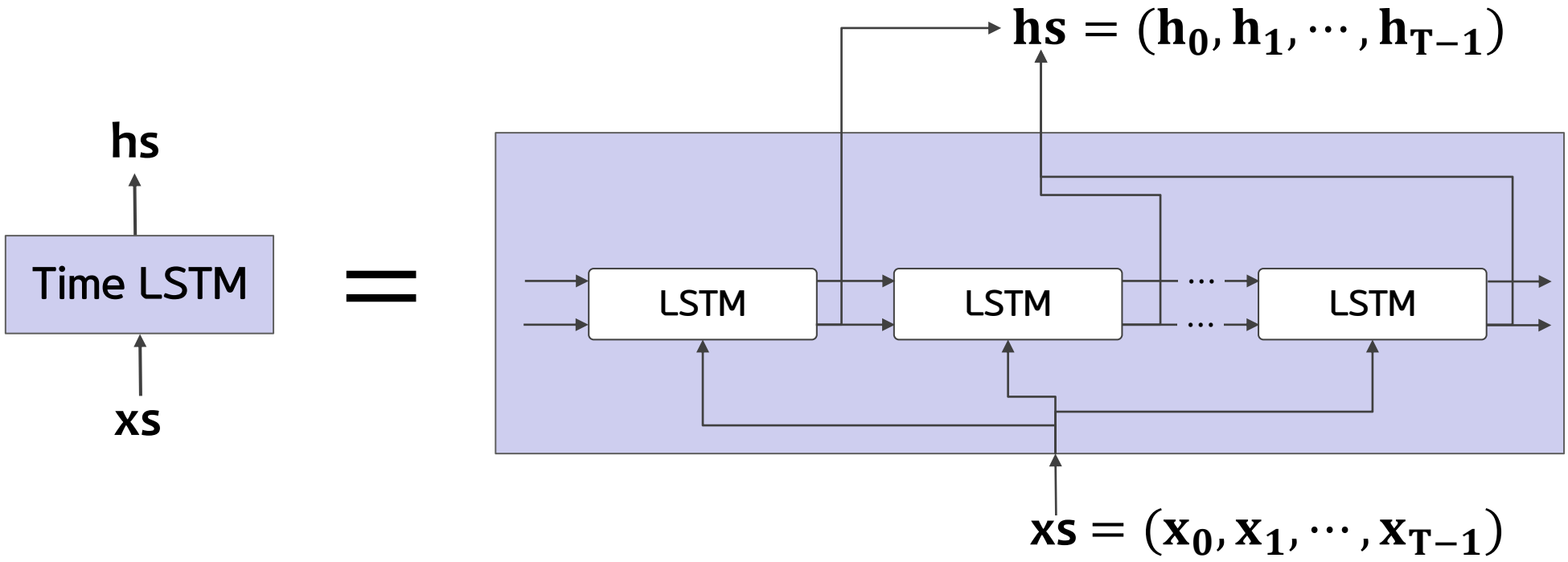
아핀 변환 시의 형상 추이


$W_x^{(f)}$ $W_x^{(g)}$ $W_x^{(i)}$ $W_x^{(o)}$
$W_h^{(f)}$ $W_h^{(g)}$ $W_h^{(i)}$ $W_h^{(o)}$
x_t W_x + h_{t-1} W_h = A
$N \times D$ $D \times 4H$ $N \times H$ $H \times 4H$ $N \times 4H$
일치
일치

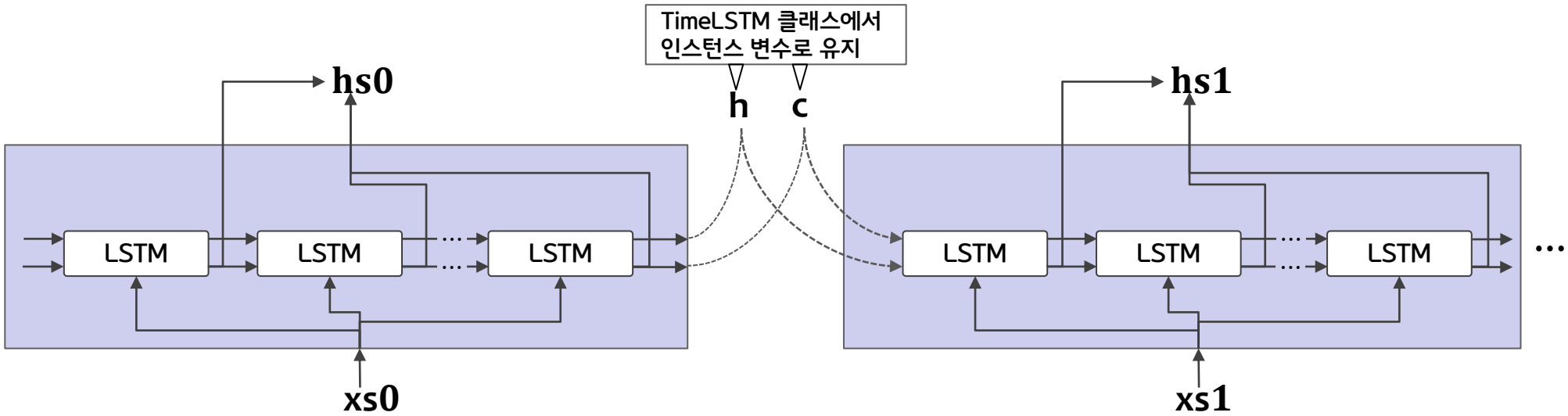
slice 노드의 순전파(위)와 역전파(아래)



Time LSTM의 입출력



Time LSTM 역전파의 입출력



6. 게이트가 추가된 RNN

6.1 RNN의 문제점

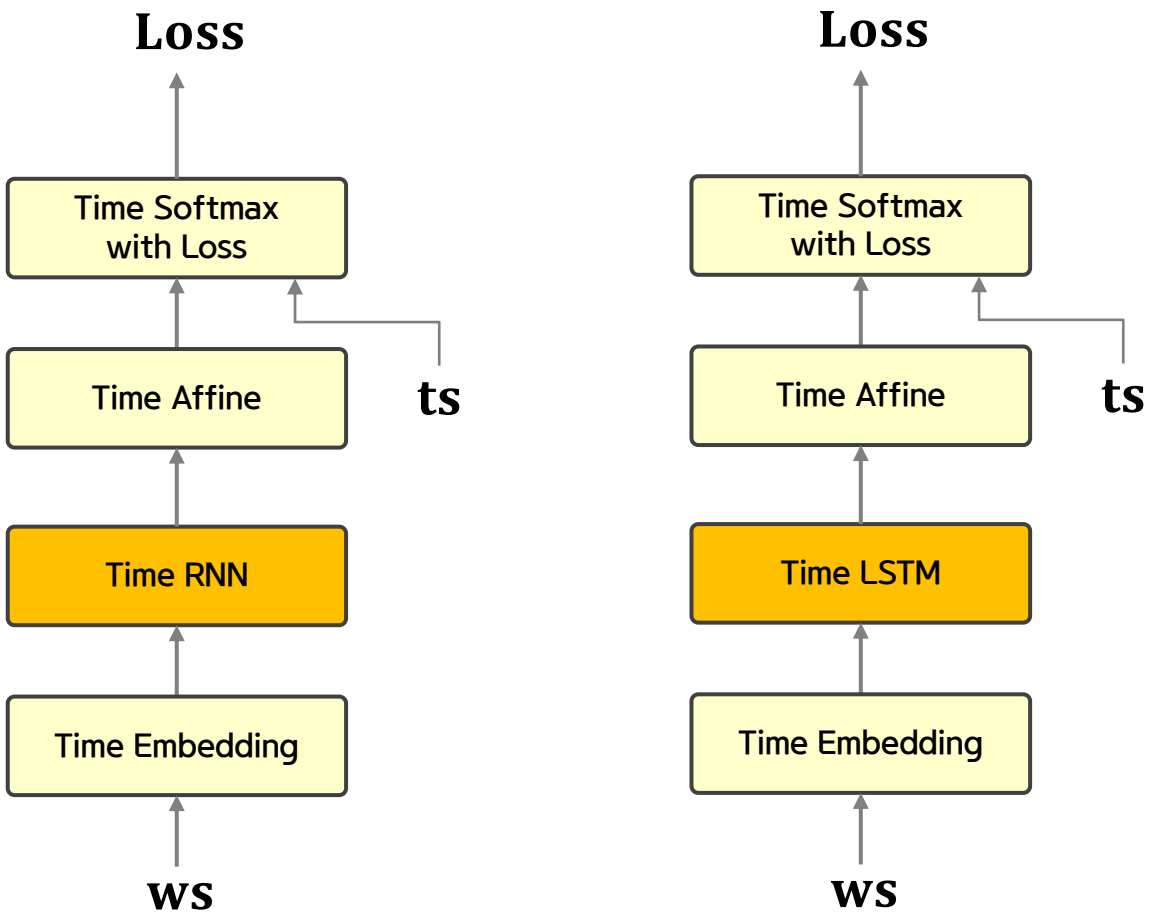
6.2 기울기 소실과 LSTM

6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

언어 모델의 신경망 구성


Loss
Time Softmax with Loss
Time Affine
ts
Time RNN
Time Embedding
ws
Loss
Time Softmax with Loss
Time Affine
ts
Time LSTM
Time Embedding
ws

※ 코드 참조

6. 게이트가 추가된 RNN

6.1 RNN의 문제점

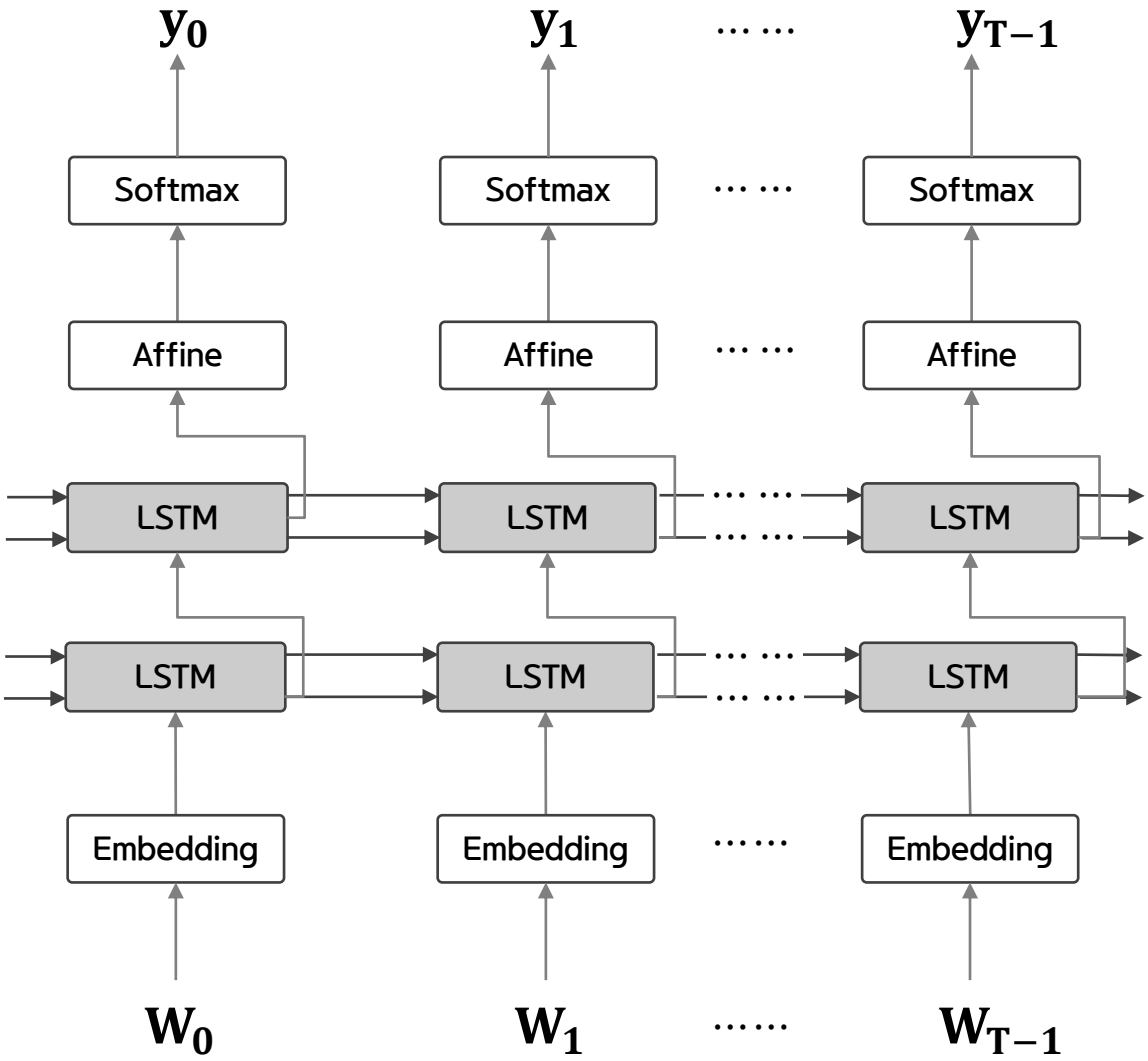
6.2 기울기 소실과 LSTM

6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

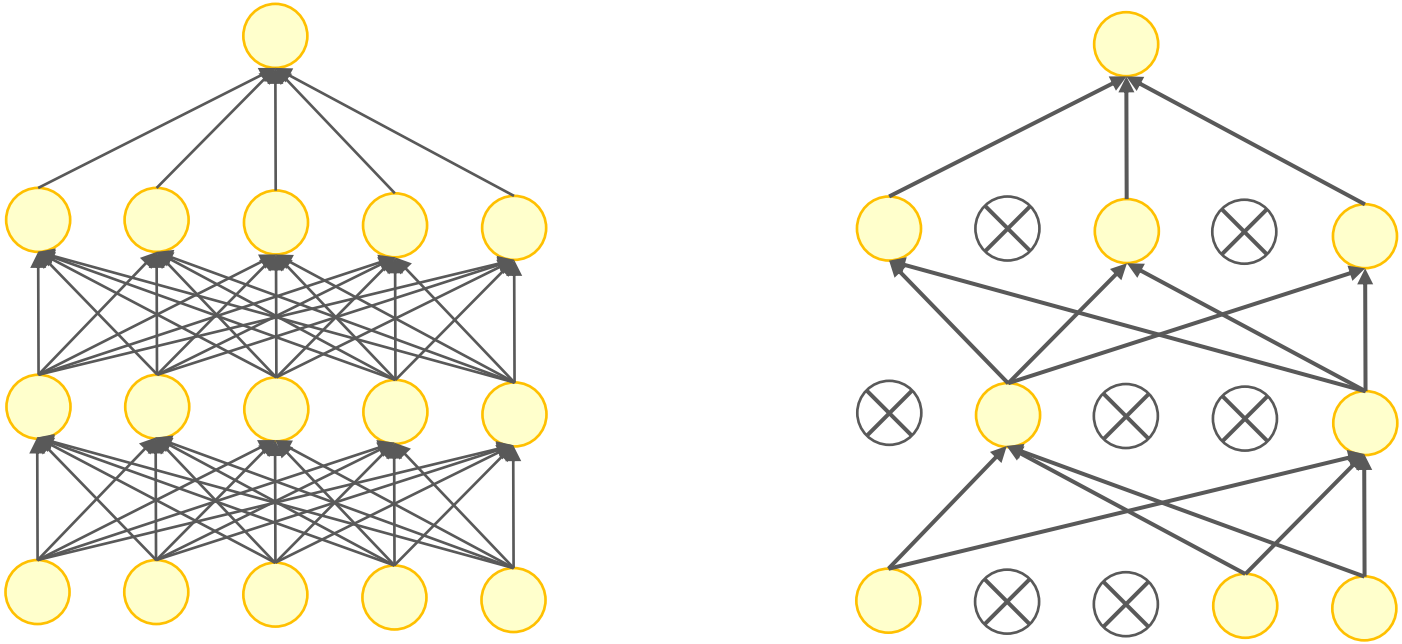
6.5 RNNLM 추가 개선

LSTM 계층을 2층으로 쌓은 RNNLM



드롭아웃에 의한 과적합 억제

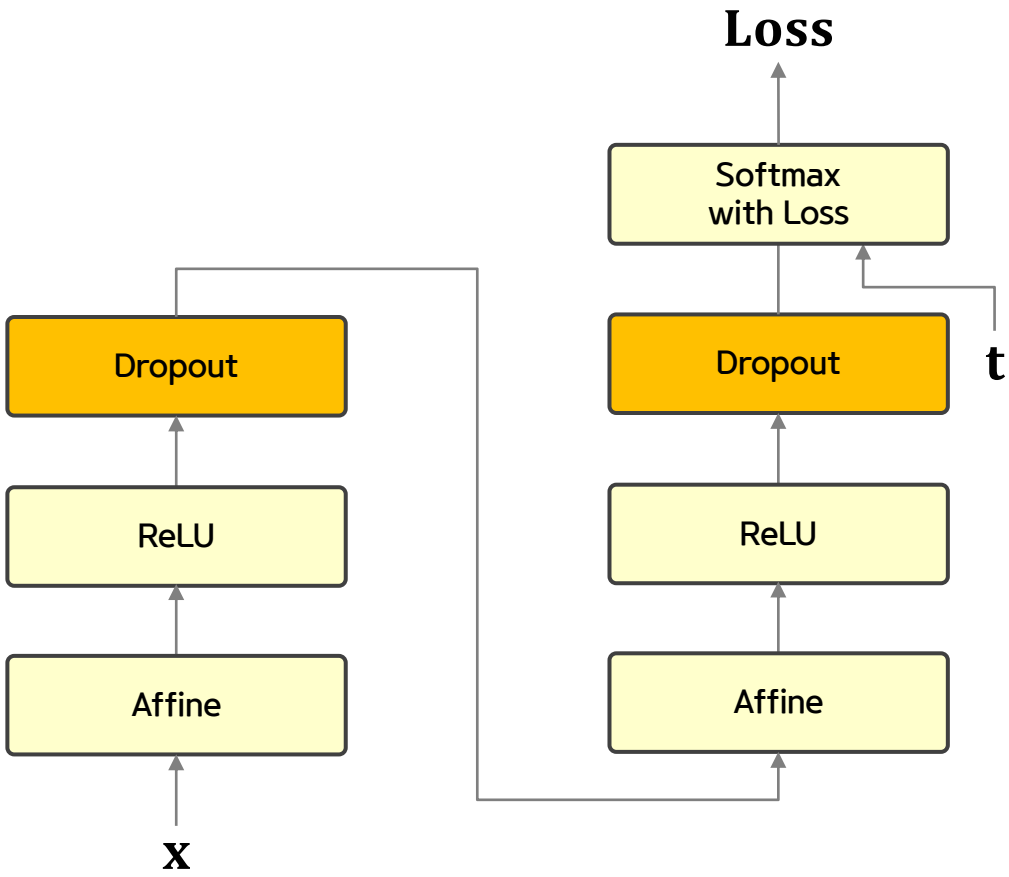
드롭아웃 개념도 : 왼쪽이 일반적인 신경망, 오른쪽이 드롭아웃을 적용한 신경망



드롭아웃은 무작위로 뉴런을 선택하여 선택한 뉴런을 무시한다.
무시한다는 말은 그 앞 계층으로부터의 신호 전달을 막는다는 뜻이다.
이 '무작위한 무시'가 제약이 되어 신경망의 일반화 성능을 개선하는 것이다.

드롭아웃에 의한 과적합 억제

피드포워드 신경망에 드롭아웃 계층을 적용하는 예

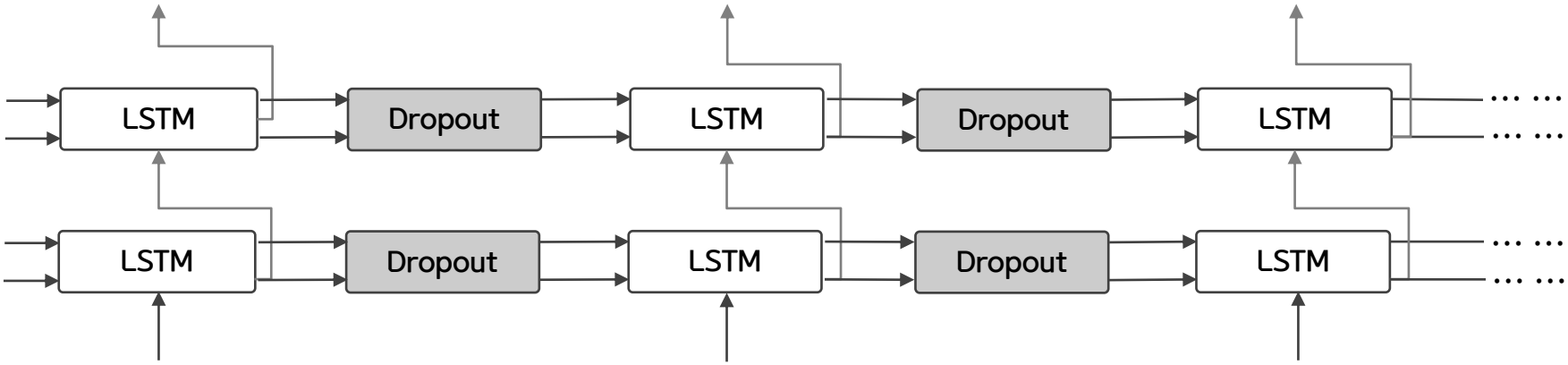


이 그림은 드롭아웃 계층을 활성화 함수 뒤에 삽입하는 방법으로 과적합 억제에 기여하는 모습이다.

RNN을 사용한 모델에서 드롭아웃 계층을 LSTM 계층의 시계열 방향으로 삽입하면 좋은 방법이 아니다.

드롭아웃에 의한 과적합 억제

나쁜 예: 드롭아웃 계층을 시계열 방향으로 삽입

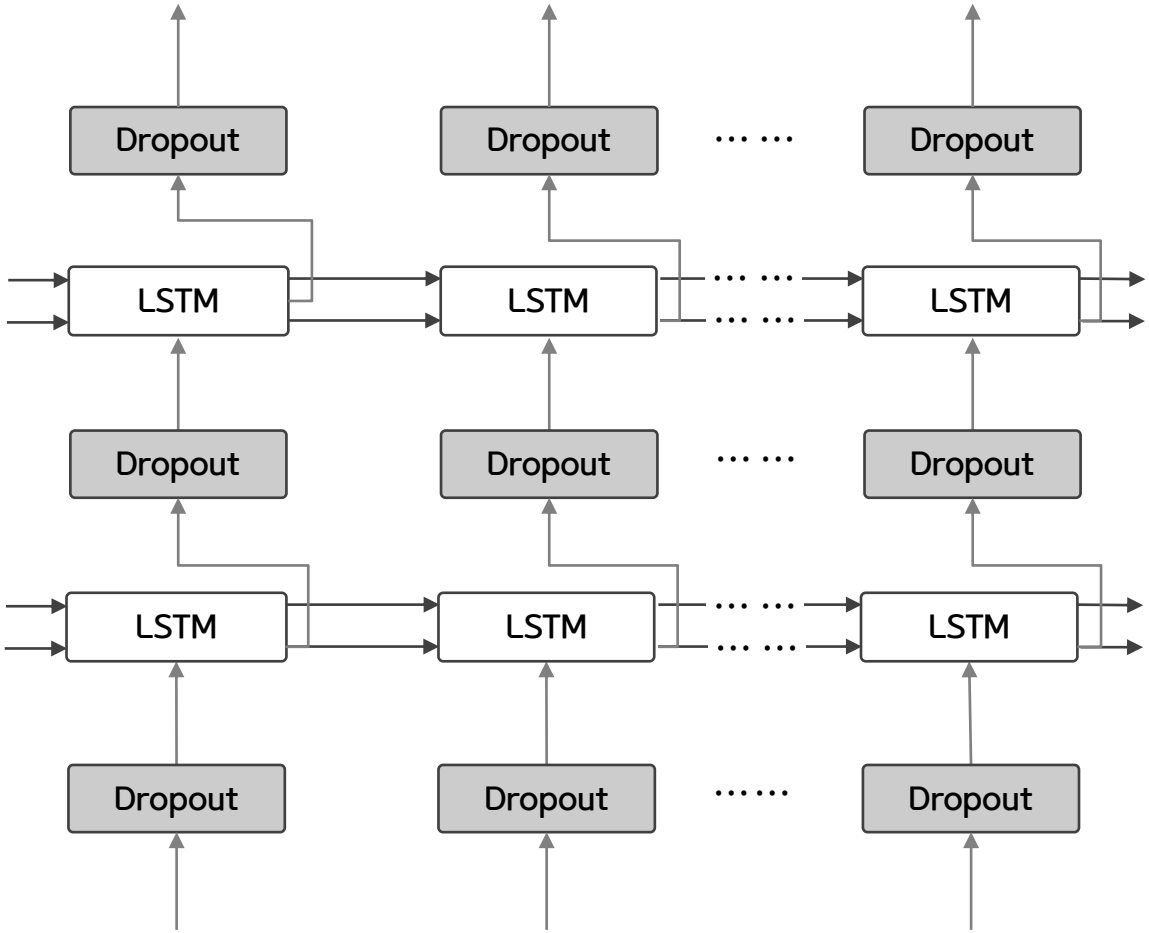


RNN에서 시계열 방향으로 드롭아웃을 학습 시 넣어버리면 시간이 흐름에 따라 정보가 사라질 수 있다.
즉, 흐르는 시간에 비례해 드롭아웃에 의한 노이즈가 축적된다.

드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입하는 방안을 생각해보자

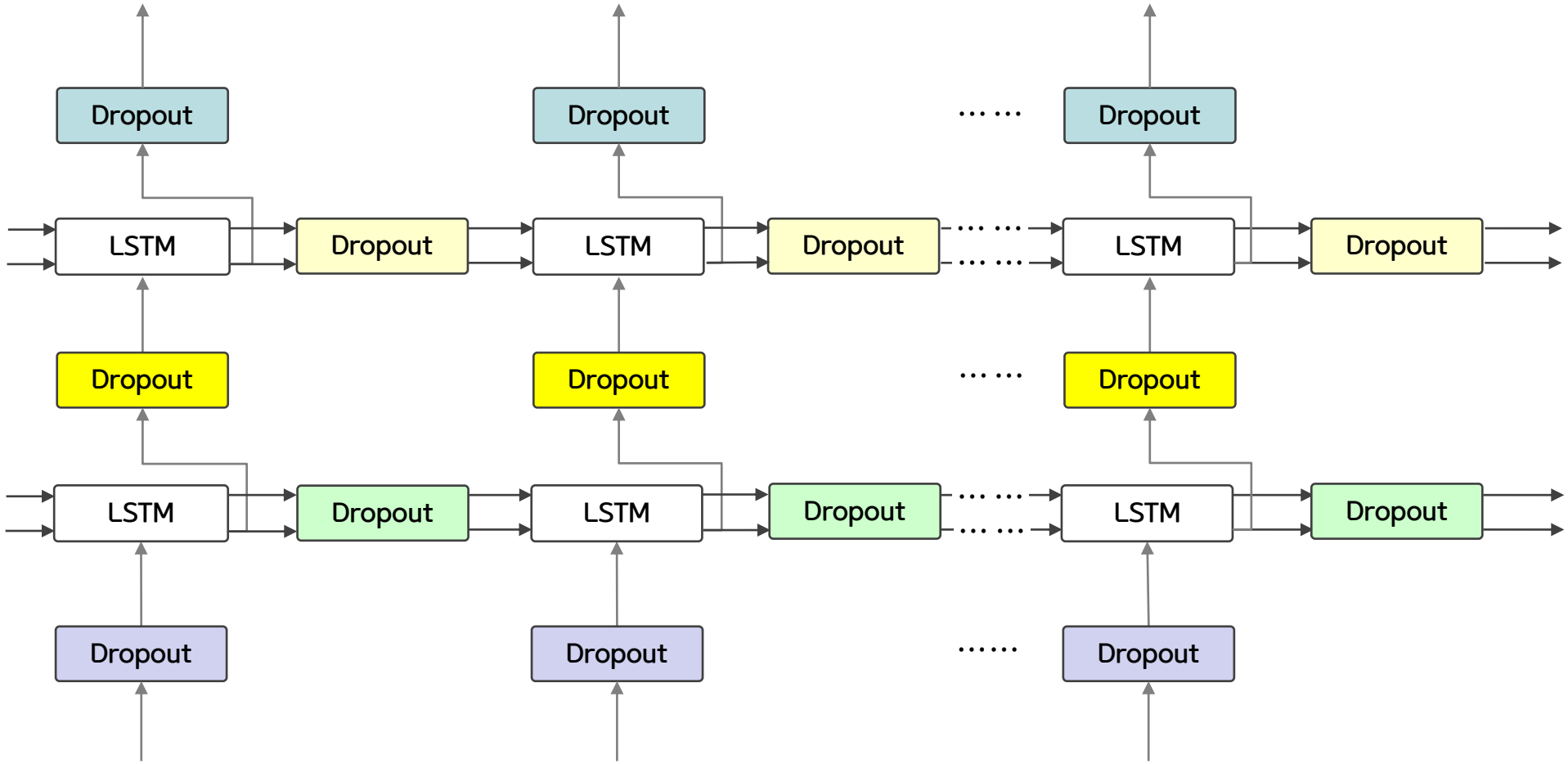
드롭아웃에 의한 과적합 억제

좋은 예: 드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입



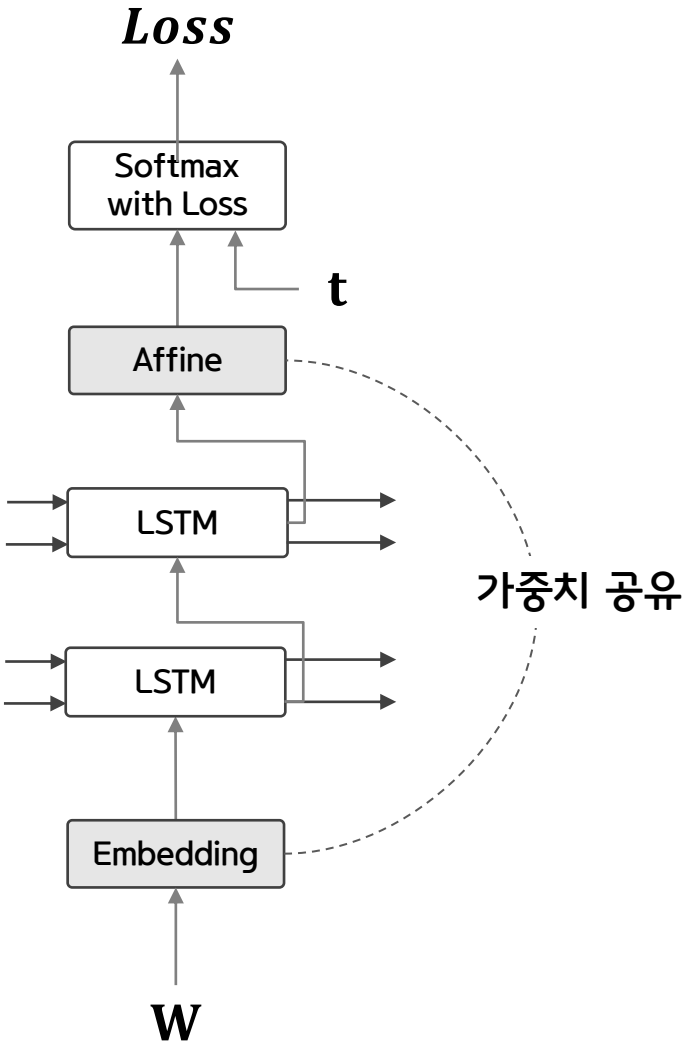
드롭아웃에 의한 과적합 억제

변형 드롭아웃의 예: 색이 같은 드롭아웃 끼리는 같은 마스크를 이용한다. 이처럼 같은 계층에 적용되는 드롭아웃 끼리는 공통의 마스크를 이용함으로써 시간 방향 드롭아웃도 효과적으로 작동할 수 있다.

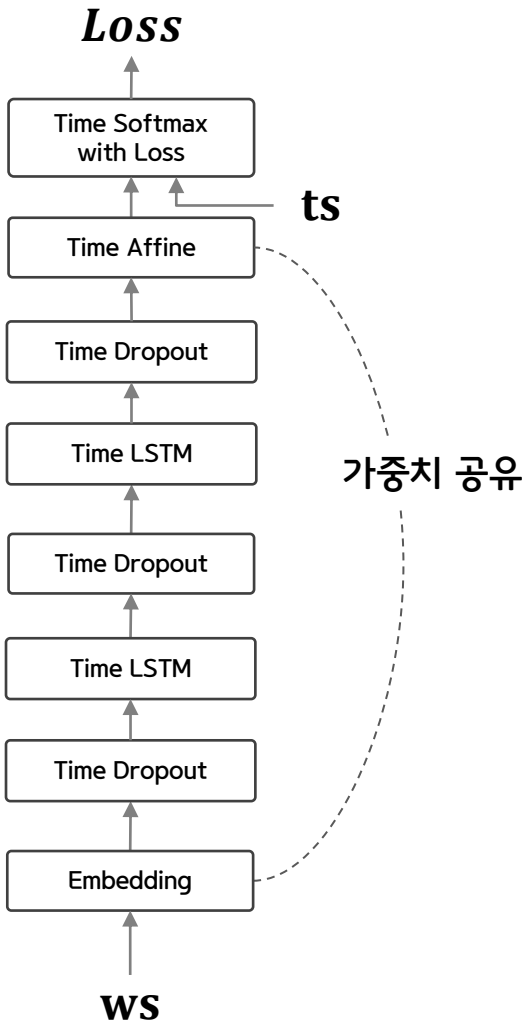


가중치 공유

언어 모델에서의 가중치 공유 예: Embedding 계층과 Sotmax 앞단의 Affine 계층이 가중치를 공유한다.



BetterRnnlm 클래스의 신경망 구성



- LSTM 계층의 다층화(여기서는 2층)
- 드롭아웃 사용(깊이 방향으로만 적용)
- 가중치 공유(Embedding 계층과 Affine 계층에서 가중치 공유)

※ 코드 참조