# 장단기 기억 (LSTM)

순환 신경망은 *기울기의 소실* 및 *폭발*과 관련된 문제를 가지고 있다.
이 문제는 *가중치 행렬의 거듭된 곱셈 때문에 불안정성이 생기는* 모든 종류의 신경망에서 흔히 나타나는 문제이다.
**역전파 과정에서 가중치 행렬의 곱셈이 거듭됨에 따라 기울기가 사라지거나 큰 값으로 발산**하게 된다.
순환 신경망에서 이런 종류의 불안정성은 *여러 시각에서의 거듭된 가중치 행렬 곱셈의 직접적인 결과* 이다.

- 이 문제를, 갱신에 곱셈만 쓰이는 순환 신경망은 오직 짧은 순차열의 학습에만 적합하다고 해석할 수 있다.
- 즉, 그런 순환 신경망은 단기 기억 능력은 좋지만 장기 기억 능력은 나쁘다고 할 수 있는 것이다.
    - 이 문제를 해결하는 한 가지 방법은 장기 기억을 활용하는 갱신 공식을 이용해서 은닉 벡터를 갱신하는 것이다.
    - 이런 식으로 만들어진 순환 신경망을 장단기 기억(LSTM) 신경망이라고 부른다.
        - LSTM 신경망의 연산들은 장기 기억에 기록되는 자료를 세밀하게 제어하도록 고안되었다.

# Long-Term Dependency (gradient vanishing/exploding)

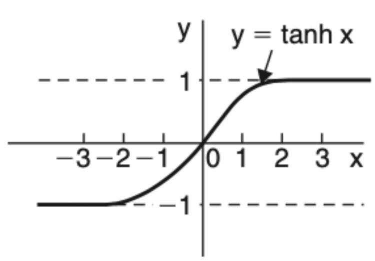
- 참고 *https://brunch.co.kr/@chris-song/9* (장기 의존성 부분만)
- 꼭 다뤄야 하는 내용 : Dependency 문제는 Forward 에서 일어나는 문제인가 Backward 에서 일어나는 문제인가?

RNN 은 신중한 하이퍼파라미터 설정으로 *장기 의존성 (Long-Term Dependency)* 문제를 해결할 능력은 있지만, 실제적으로 RNN은 이러한 문맥을 배울 수 없다는 것이 밝혀짐

## 장기 의존성 이란?

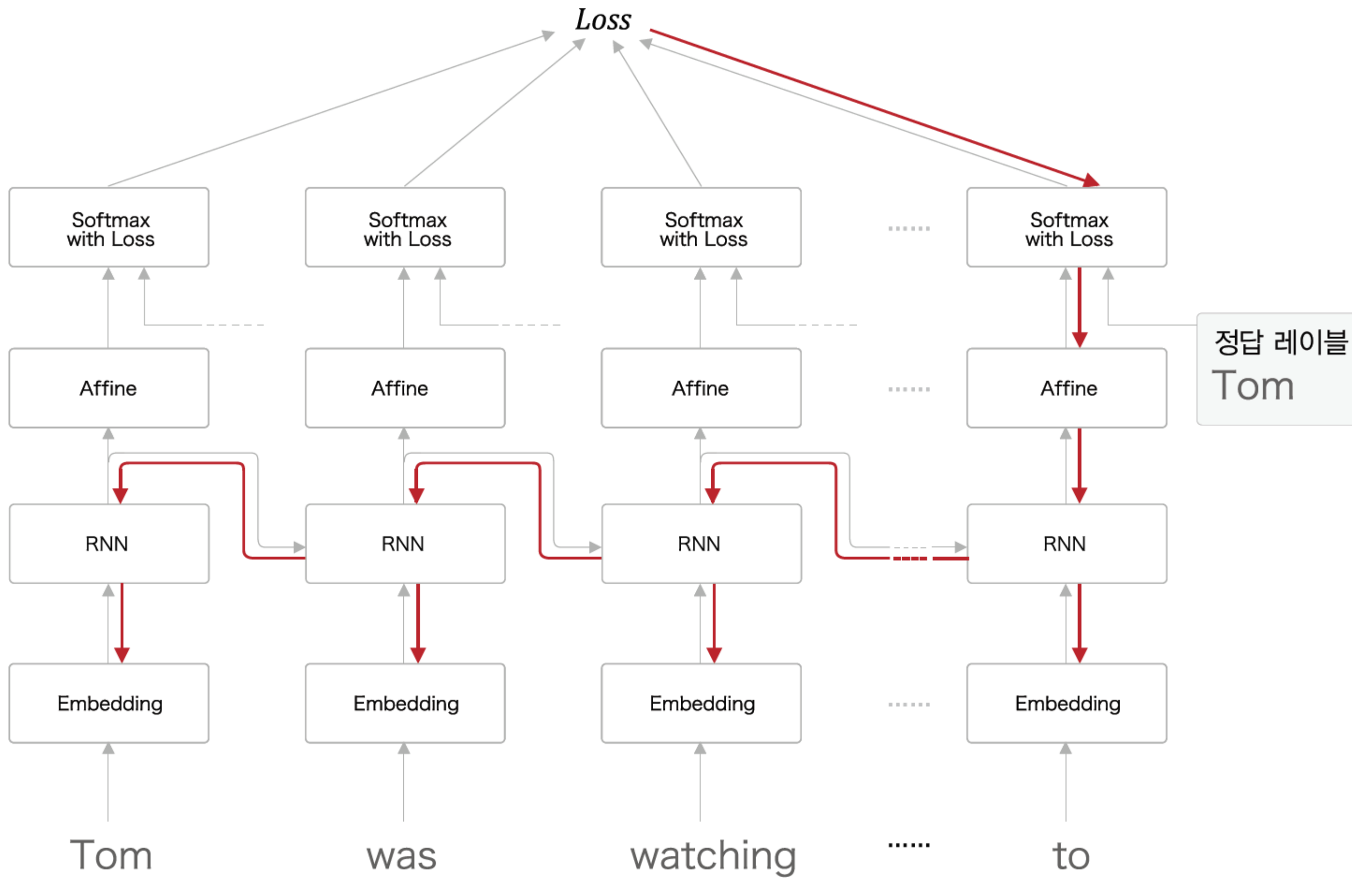
***장기의존성 문제*는 은닉층의 과거의 정보가 마지막까지 전달되지 못하는 현상을 의미한다.**

hyperbolic tangent는 은닉층 값의 계산해서 계~속 곱해진다. 그런데~ 이 tanh 의 값은 *1과-1 사이*!!




*tanh에 대한 자세한 설명*
즉!! 1보다 작은 값이 반복적으로 곱해지기 때문에, feed-forward의 관점에서 뒷단으로 갈 수록 앞의 정보를 충분히 전달할 수 없고, back-propagation의 관점에서는 tanh의 함수의 기울기가 0에 가깝거나 굉장히 큰 값이 발생할 수 있어, 기울기 소실 또는 폭발의 문제를 일으킴

그림 6-4 정답 레이블이 "Tom"임을 학습할 때의 기울기 흐름

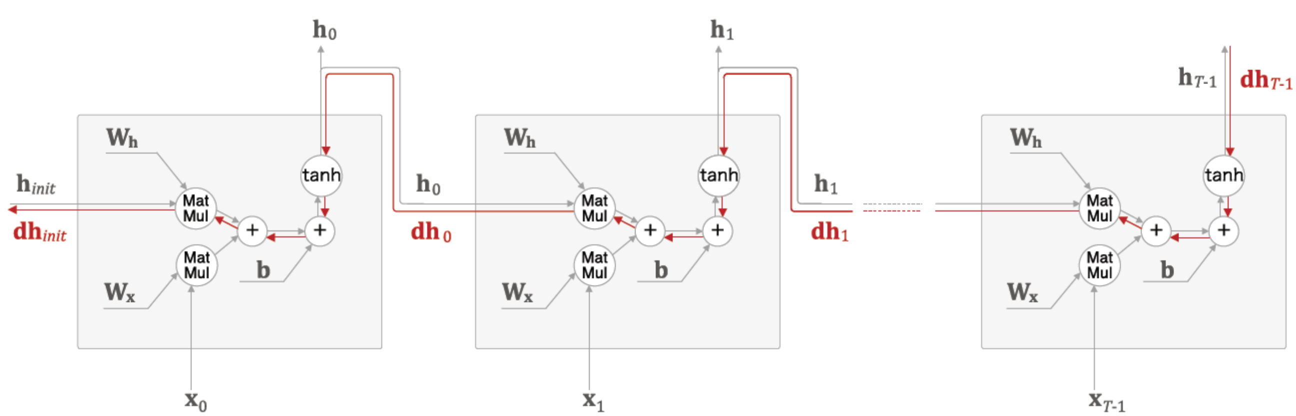



첫번째 그림과 같이 "Tom"을 학습할때 RNN계층이 의미있는 기울기를 전달함으로써 시간 방향의 의존 관계를 학습할 수 있는 것인데, 이때 기울기가 원래대로 라면 학습해야 할 의미가 있는 정보가 들어 있고, 그것을 과거로 전달함으로써 장기 의존 관계를 학습한다.

하지만 중간에 소실되면 가중치 매개변수는 전혀 갱신되지 않게 된다. 즉 장기 의존 관계를 학습할 수 없게 된다.

기울기 소실 혹은 폭발문제를 겪게되는 원인:

그림 6-5 RNN 계층에서 시간 방향으로의 기울기 전파



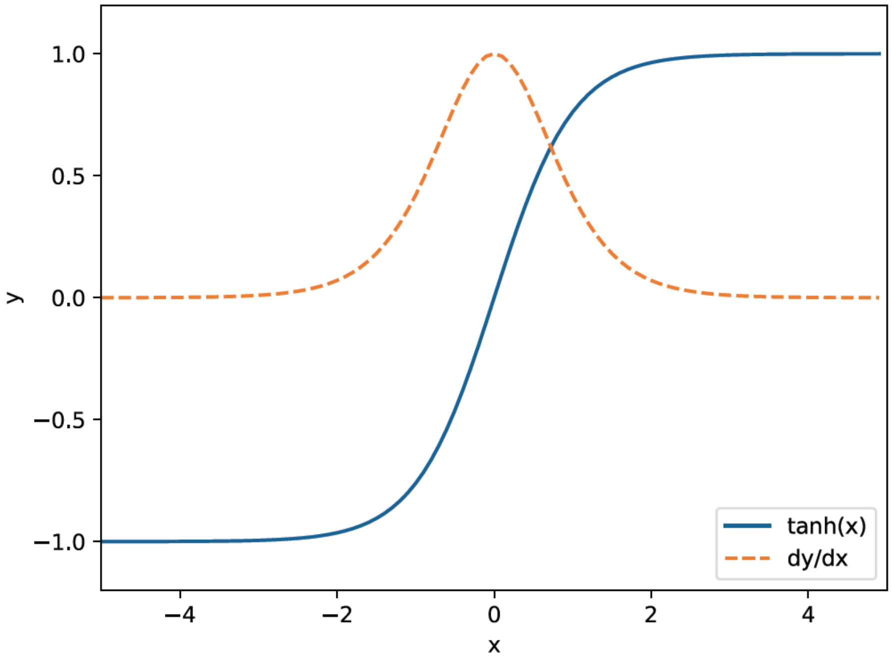

위 그림처럼 길이가 T인 시계열 데이터를 가정하여 T번째 정답 레이블로부터 전해지는 기울기가 어떻게 변하는지 보자!

앞의 Tom에 대해서 보면 T번째 정답 레이블이 'Tom'인 경우에 해당하며, 시간 방향 기울기에 주목하면 역전파로 전해지는 기울기는 차례로 'tanh' -> '+' -> 'MatMul (행렬곱)' 연산을 통과하는 걸 알 수 있다.

'+' 의 역전파는 상류에서 전해지는 기울기를 그대로 하류로 흘려보내서 기울기는 변하지 않지만, 나머지 두 연산

인 'tanh'와 'MatMul'은 기울기를 변화시키는 요인이다.
다음 그림은 $y = tanh(x)$ 의 값과 그 미분 값을 각각 그래프로 보여준다.

그림 6-6 $y = \tanh(x)$의 그래프(점선은 미분)

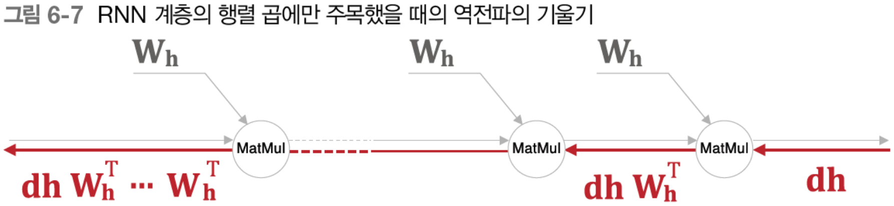



점선이 미분된 값이며 보다시피 그 값은 1 이하이고 x가 0으로부터 멀어질 수록 작아진다. 즉 역전파에서는 기울기가 tanh노드를 지날때 마다 값이 계속해서 작아진다는 말이다.

- 그래서 tanh 함수를 T번 통과하면 기울기도 T번 반복해서 작아지게된다.

또한 역전파동안 'MatMul (행렬의 곱)'노드를 지날 때마다 dh라는 기울기와 같은 가중치 Wh 가 곱해지게 되는데 이때 매번 똑같은 값이 계~속 곱해지게 되면 기울기 소실이 되지 않았을때 오히려 폭발로 가게 되는것이다.

그림 6-7 RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기




# introduction to 3 gates - LSTM

**LSTM은 Long Short-Term Memory 이다 : 즉 단기 기억을 장~시간 지속할 수 있음을 의미한다.**

- 참고 *https://wegonnamakeit.tistory.com/7* ( LSTM 네트워크 구조 와
에 등장하는 그림으로 존재 이유와 해당 게이트가 어떤 값에 영향을 주는지 정도만 다루면 됨)

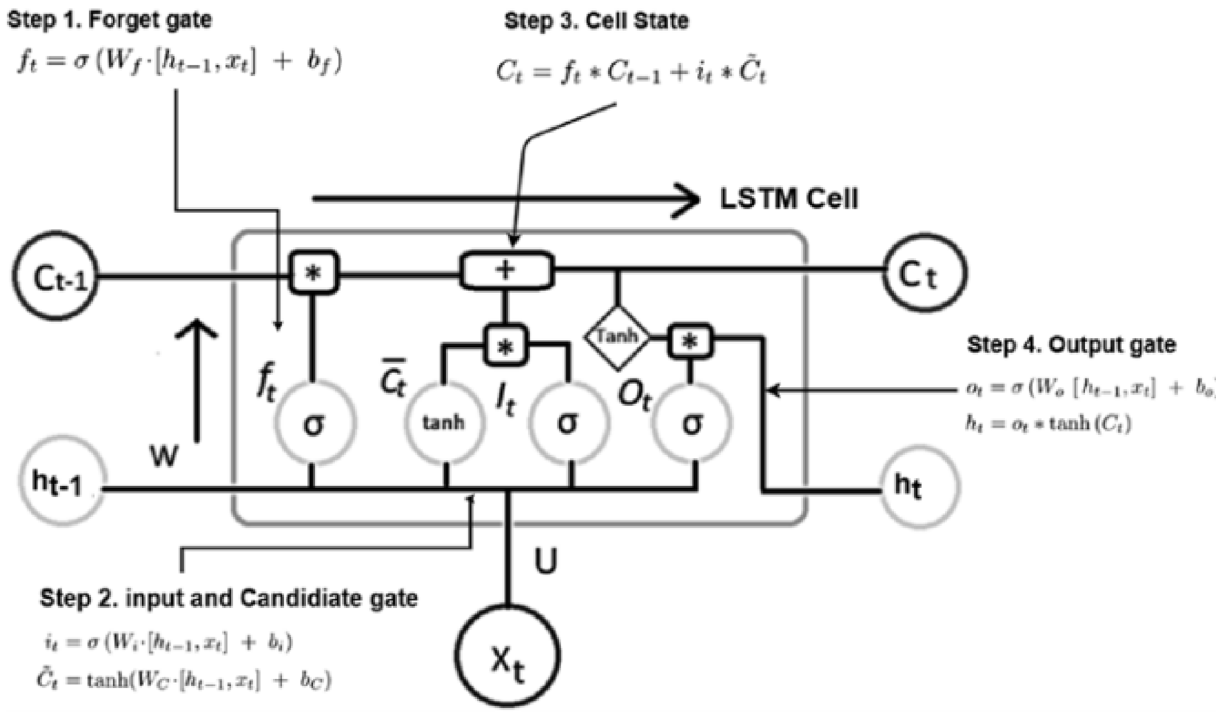
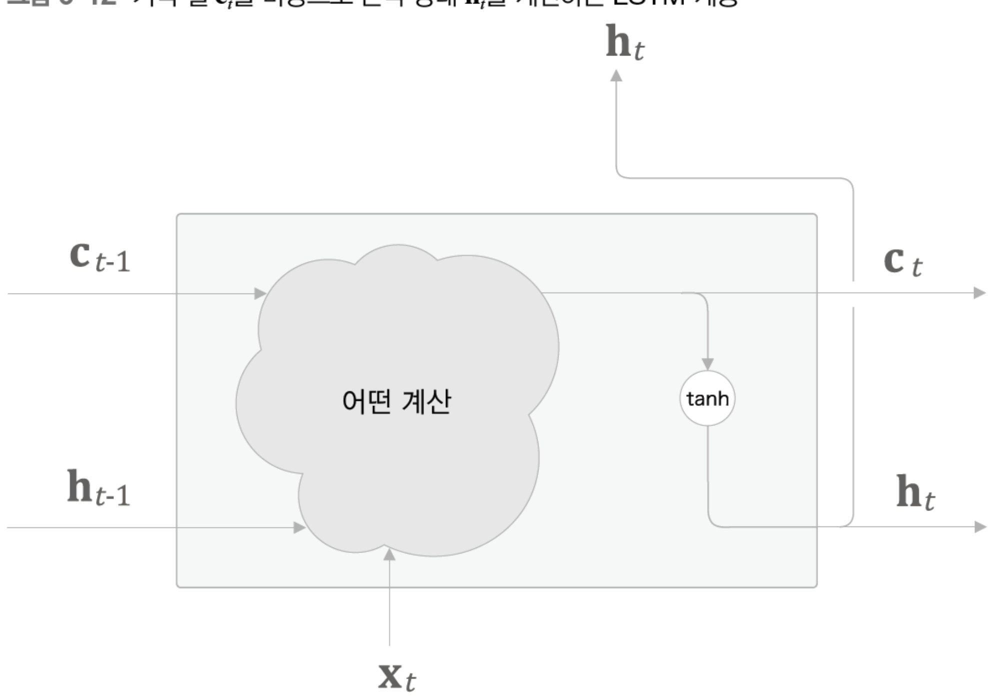



***Figure 5-29.*** *LSTM cell architecture with formulas*

그림 6-12 기억 셀 $\mathbf{c}_t$를 바탕으로 은닉 상태 $\mathbf{h}_t$를 계산하는 LSTM 계층




LSTM에는 기억 셀 $c_t$ ((cell) state) 가 있다. 이 $c_t$ 에는 시각(시간인듯) t에서의 LSTM의 기억이 저장되어 있는데, 과거로부터 시각 t까지에 필요한 모든 정보가 저장되어 있다고 가정해보자
그리고 필요한 정보를 모두 간직한 이 기억을 바탕으로, 외부 계층에 은닉 상태 $h_t$를 출력한다.
이때 출력하는 $h_t$는 기억 셀의 값을 $tanh$ 함수로 변환한 값임.

위 그림처럼 현재의 기억 셀 $c_t$는 3개의 입력 ($c_{t-1}$, $h_{t-1}$, $x_t$)으로부터 '어떤 계산'을 수행하여 구할 수 있다. *핵심은 갱신된 $c_t$를 사용해 은닉 상태 $h_t$를 계산한다는 것이다.*

- 이 계산은 $h_t = tanh(c_t)$ 인데, 이는 $c_t$의 각 요소에 tanh 함수를 적용한다는 뜻
- *$c_t$와 $h_t$의 원소 수는 같다. 즉 하나가 100개의 원소면 다른 하나도 100개임*

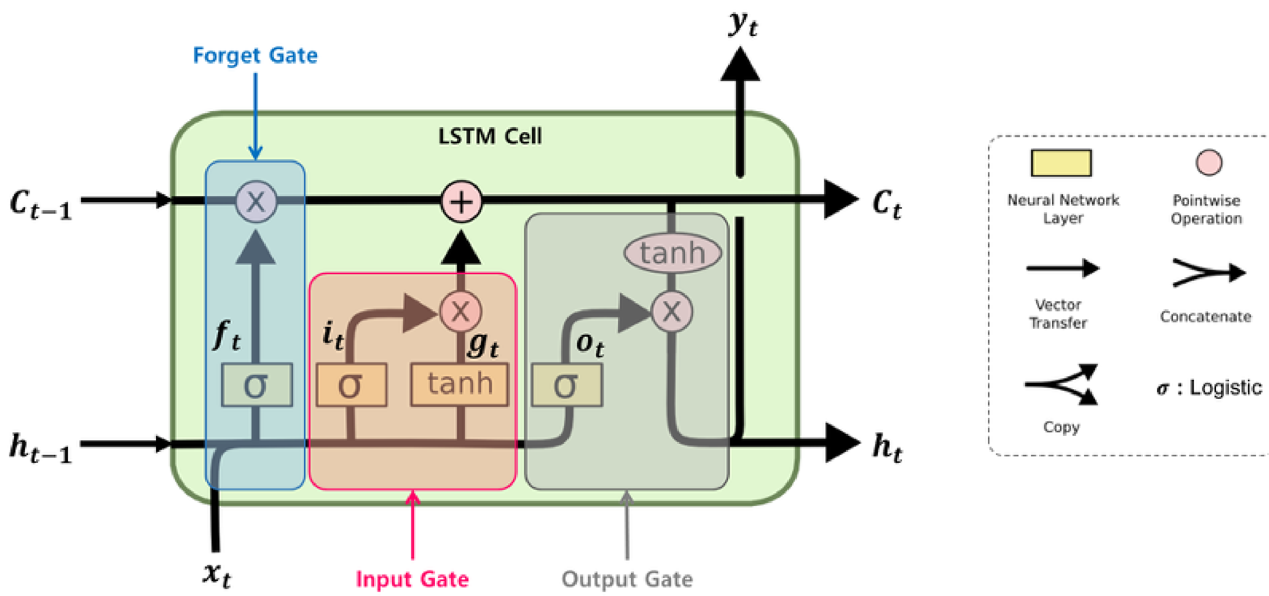
## 게이트의 역할

**그림 6-13** 비유하자면 게이트는 물의 흐름을 제어한다.

**그림 6-14** 물이 흐르는 양을 0.0~1.0 범위에서 제어한다.


게이트는 데이터의 흐름을 제어하는 역할을 한다.

- 댐이 강물을 방류할때 게이트를 얼마만큼 여는것과 비슷한 개념
- 즉, 게이트는 '열기/닫기' 뿐만 아니라 어느정도 열지를 조절할 수 있다.
- 어느 정도 흘려보낼 물의 양을 조절 가능
  - *어느 정도*를 '열림 상태 (openness)'라 부르며 0.7(70%), 0.2(20%) 처럼 제어가 가능함
  - 즉 1또는 0인 정수가 아니고 실수 (*float*)값을 *시그모이드* 함수로 부터 받는다.
- **게이트를 얼마나 열까?** 또한 *데이터로부터 자동으로 학습*한다.



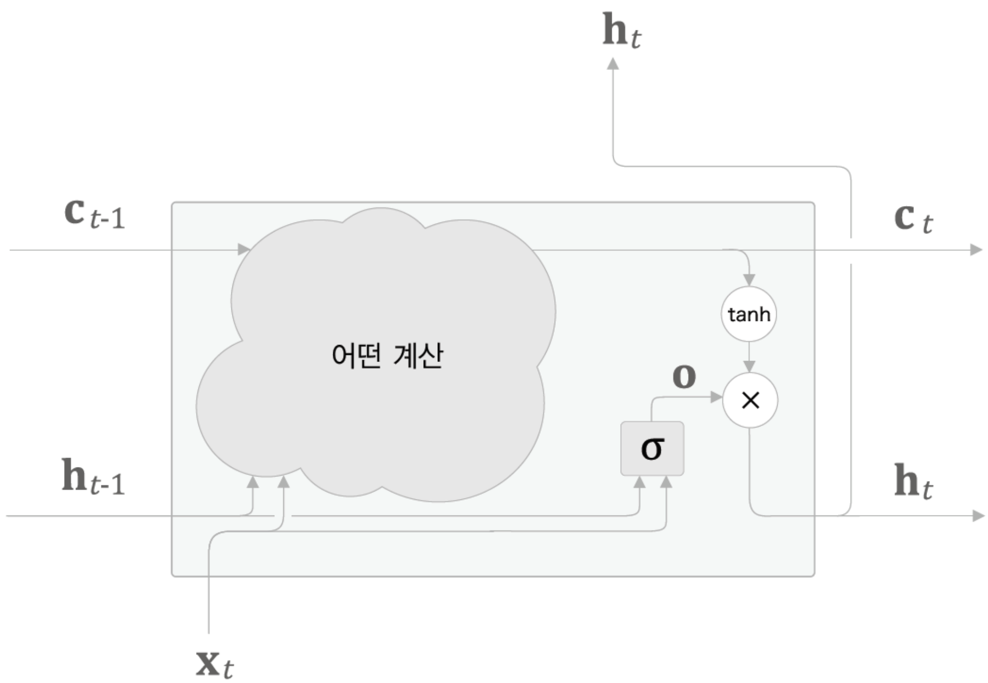

## output gate

- $o_t$는 장기 상태 $c_t$의 어느 부분을 읽어서 $h_t$와 $y_t$로 출력해야 하는지 제어한다.
- 은닉상태 $h_t$는 기억 셀 $c_t$에 단순히 $tanh$를 적용함 -> $tanh(c_t)$
- 즉 출력 gate는 $tanh(c_t)$의 각 원소 또는 요소 에 대해 '*그것이 다음 시각의 은닉 상태에 얼마나 중요한가*'를 조정함
- 출력 게이트의 열림 상태 (다음 몇 %만 흘려보낼까?)는 입력 $x_t$와 이전 상태 $h_{t-1}$로부터 구합니다.

output = $o$

- 출력 계산식:
- $o = \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)})$
  식에서 보다시피 입력 과 은닉 상태에는 각각의 가중치가 붙어 있는것을 확인할 수 있으며, 이 행렬들의 곱과 편향을 모두 더한다음 시그모이드 함수를 거쳐 출력 게이트 $o$를 구한다. 마지막으로 이 $o$와 $tanh(c_t)$의 원소별 곱을 $h_t$로 출력한다.

그림 6-15 output 게이트 추가




그림에서 출력게이트에 해당하는 부분은 $\sigma$로 표기한다. 그리고 $\sigma$의 출력을 $o$라고 하면 $h_t$는 $o$와 $tanh(c_t)$의 곱으로 계산된다.
**여기서 말하는 곱이란 원소별 곱이며, 이것을 아다마르 곱(Hadamard product)이라고 한다.**
아다마르 곱을 기호로는 ⊙ 로 나타낸다.
최종 은닉 상태 출력은 다음과 같다.

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

한가지 짚고 넘어가자~!!

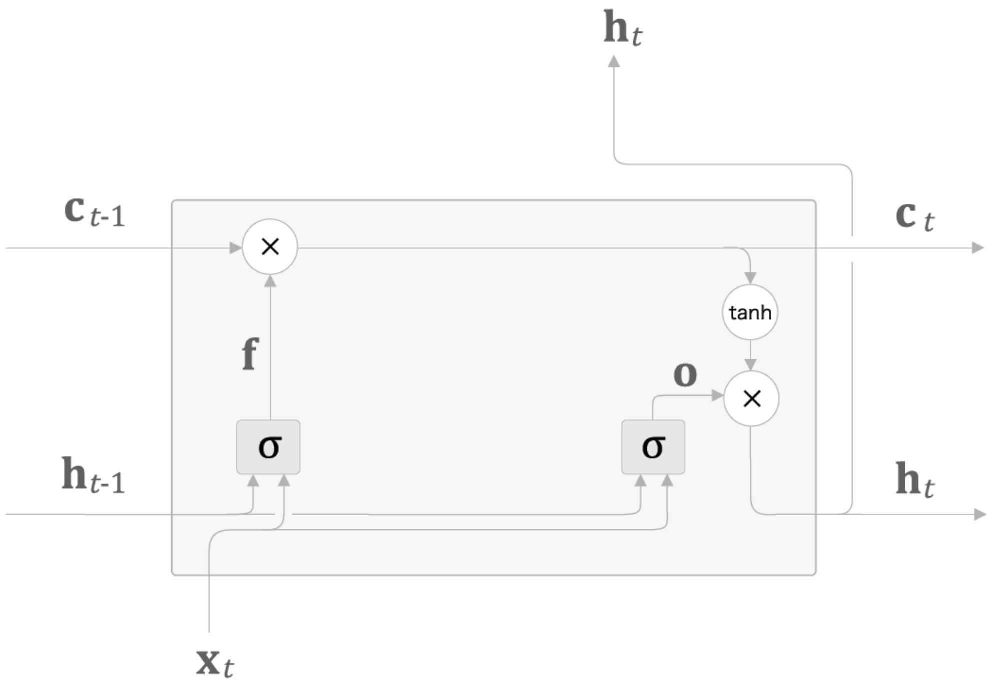
> tanh 의 출력은 -1.0 ~ 1.0 사이의 실수임 이 수치를 인코딩된 '정보'의 강약(정도)를 표시한다고 해석 가능함
> sigmoid 함수는 0.0 ~ 1.0 사이의 실수이며 데이터를 얼마만큼 통과시킬지를 정하는 비율을 뜻함

- 따라서!! 주로 게이트(gate) 에서는 시그모이드 함수가
- 실질적 '정보'를 지니는 데이터에는 tanh 함수가 활성화 함수로 사용됨

# forget gate

- $f_t$ 에 의해 제어되며 장기 상태 $c_t$ 의 어느 부분을 삭제할지 제어한다.
- 망각은 더 나은 전진을 낳는다고 책이 그럽니다.. 가끔은 쓸데없는 정보를 잊어야 우리가 더 앞으로 나아갈 수 있다는 의미인가봐요.
- 우리가 다음에 해야 할 일은 기억 셀(cell state)에 '*무엇을 잊을까?*'를 명확하게 지시하는 것입니다.
  - 이런일도 물론 **게이트를 사용해 해결**합니다.
  - $ct-1$의 기억 중에서 불필요한 기억을 잊게 해주는 게이트(forget gate : 망각게이트)를 추가해 봅시다.

그림 6-16 forget 게이트 추가




그림에서 망각게이트가 수행하는 일련의 계산을 $\sigma$노드로 표기함. (모든 게이트는 저렇게 표시함, 정보를 지니는 것은 tanh로 표시!)

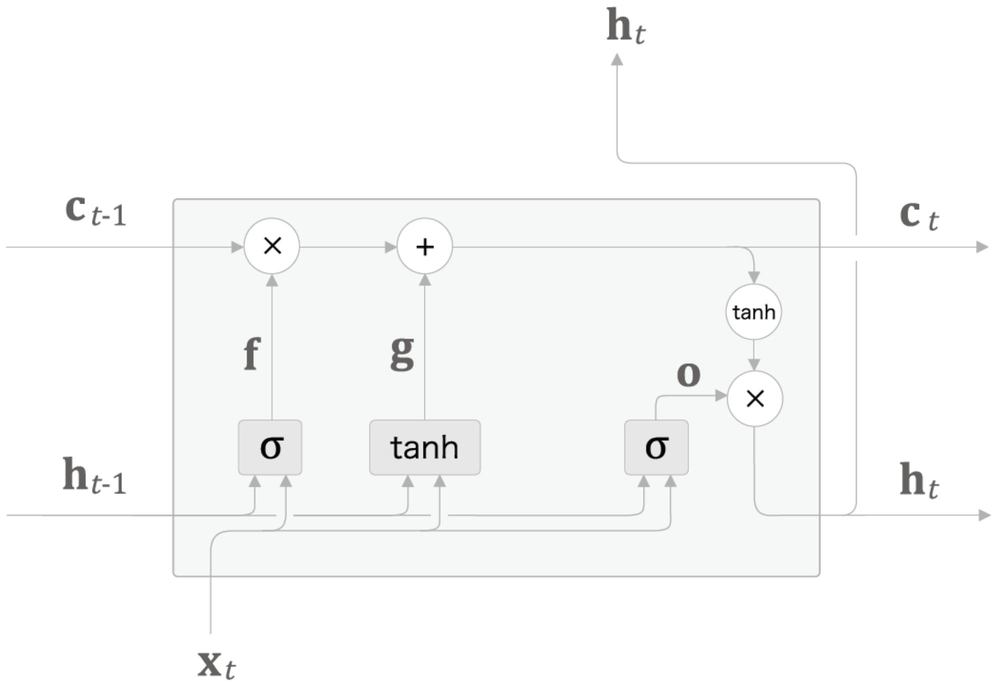
- 이 시그모이드 노드 안에는 forget 게이트 전용의 가중치 매개변수가 있으며, 다음 식의 계산을 수행한다.
- $f = \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)})$
  위 공식을 통해서 망각 게이트의 출력 $f$가 구해짐, 그리고 이 $f$와 이전 기억 셀(cell stste)인 $ct-1$과의 원소별 곱(아다마르 곱)을 구함
- 즉 $c_t = f \odot c_{t-1}$ 을 계산하여 $c_t$를 구함

## 새로운 기억 셀

forget(망각) 게이트를 거치면서 이전 시각의 기억 셀로부터 잊어야 할 기억이 삭제 되었음.
하지만 이 상태로는 기억 셀이 잊는것 밖에 하지 못함 ㅠㅠ (chimae에 걸렸다는 표현인가..)

- 그래서 새로 기억해야 할 정보를 기억 셀(cell state)에 추가해야함!
  - 그러기 위해 tanh 노드를 추가함
  -

그림 6-17 새로운 기억 셀에 필요한 정보를 추가




위 그림과 같이 tanh노드가 계산한 결과가 이전 시각의 기억 셀(cell state) $c_{t-1}$에 더해짐

- 기억셀에 새로운 *정보*가 추가된 것임.
  - 이 노드는 게이트가 아니며 *새로운 정보*를 *기억 셀에 추가* 하는 것이 목적임
  - 그래서 활성화 함수가 시그모이드 함수가 아닌 *tanh*한수가 사용된 것임

  계산식은 다음과 같음:

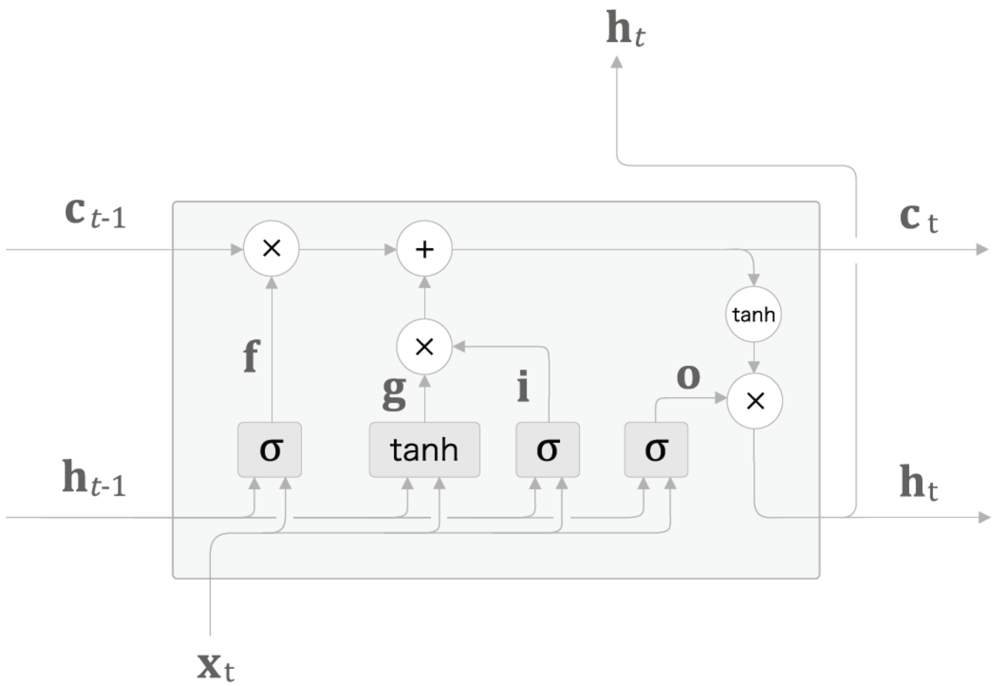
  $g = tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$

  여기서는 새로운 기억을 $g$로 표기함 이 새로운 기억$g$이 이전 시각의 기억 셀인 $c_{t-1}$에 더해짐으로써 새로운 기억이 생겨남.

## input gate

- $i_t$ 에 의해 제어되며 $g_t$의 어느 부분이 장기 상태 $c_t$에 더해져야 하는지 제어한다.
  마지막으로 새로운 기억 $g$에 게이트를 하나 추가할 것임 -> input gate (입력 게이트)

그림 6-18 input 게이트 추가




- 이 입력 게이트는 $g$의 각 원소가 새로 추가되는 정보로써의 가치가 얼마나 큰지를 판단함
- 새 정보를 무비판적으로 수용하는 것이 아니라, 적절히 취사선택 하는것이 이 게이트의 역할임
- 다른 관점에서 보면 input(입력) 게이트에 의해 가중된 정보가 새로 추가되는 것임
- 즉 새로운 정보를 입력게이트가 얼마만큼 흘려보낼지를 결정하는 단계로 이해해도 되겠음
  계산식은 다음과 같음:
  $i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$
  이제 새로운 정보인 $g$와 입력 게이트의 출력 값 $i$의 원소별 곱(아다마르 곱) 결과를 기억 셀(cell state)에 추가함

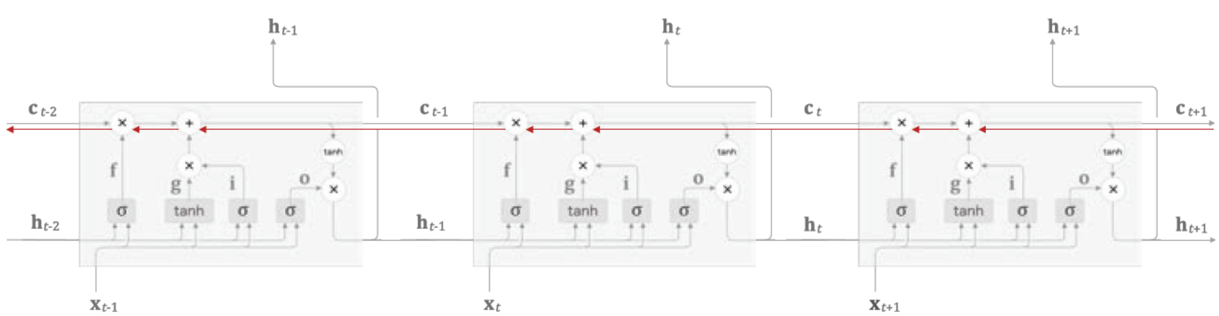
여기까지가 LSTM 안에서 이뤄지는 처리임

## LSTM의 기울기 흐름

그럼 LSTM의 구조 설명이 끝났는데.. 어떤 원리로 기울기 소실을 해결해준다는 말일까????

- 그 원리는 cell state : $c_t$의 역전파에 주목하면 보인다!!!
  해결:
  밑바닥 딥러닝 2 pg 256

그림 6-19 기억 셀의 역전파




위 그림은 기억 셀 (cell state)에만 집중하여, 그 역전파의 흐름을 그린 것이다.

- 이때 기억 셀의 역전파에서는 *'+'* 와 *'x'* 노드만 지나간다
- *'+'* 노드는 상류에서 전해지는 기울기를 그대~로 흘릴 뿐임
    - 즉, 기울기 변화(감소) 는 일어나지 않는다.
- 남은것: *'x'* 노드 **행렬 곱이 아니라 원소별 곱(아마다르 곱) 이다!!!**

RNN의 역전파 에서는 똑같은 가중치 행렬을 사용하여 '행렬 곱'을 반복하는 바람에 기울기 소실 또는 폭발이 일어 났음
반면, LSTM의 역전파에서는 '행렬 곱'이 아닌 '원소별 곱'이 이뤄지고, 매 시각 다른 게이트 값을 이용해 원소별 곱을 계산한다.

이처럼 매번 새로운 게이트 값을 이용하므로 곱셈의 효과가 누적되지 않아서 기울기 소실이 일어나지 않거나 일어나기 어려운 것임

또한 위 그림의 *'원소별 곱'의 계산은 forget gate가 제어*한다.

- 매 시각 다른 게이트 값을 출력한다.
- forget gate가 '잊어야됭~' 하고 판단한 기억 셀 (cell state) 일 경우 기울기가 작아지게 되는 원리임
- 반대의 경우 기울기가 약화 되지 않은 채로 과거 방향으로 전해짐
    - 즉, 기억 셀 (cell state)의 기울기가 오래 기억해야 할 정보일 경우 소실 없이 전파되리라 기대함

순환 신경망은 시간이 지남과 동시에 역전파가 이뤄지는데 같은 시점에서 오래 기억해야 하는 정보는 입력 게이트가 더해주니까 단기 메모리가 오~래 가는 느낌

## what is calculated at each gate - LSTM

- 수식은 *https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/* 에 가장 정확히 작성되어 있으나 어디가 ft를 의미하는지, 어디가 ot 인지 그림에 나타있지 않 아 해석하기 어려움.

- 따라서 *https://brunch.co.kr/@chris-song/9* 의 부분을 통해 step- by-step 에 등장하는 notation을 먼저 확인하고 ratsgo 수식을 다시 보는것을 권장 (브런치 글 은 수식이 두루뭉실 적혀 있음. 수식은 ratsgo 꺼 기준으로)

- 현재 계산되고 있는게 scalar 인지, vector 인지 따라가다보면 헷갈릴 수 있는데, 각 입출력이 어떻게 생긴 데이터일지 예시를 하나 샘플로 만들어서 따라가 보면 좋음.

- 위 scalar, vector 내용 관련해서 *http://blog.naver.com/PostView.nhn*?blogId=apr407&logNo=221237917815&parentCategoryNo=&categoryNo=58&viewDate=&isShowPopularPosts=true&from=search 에 적힌 몇가지 문구가 도움이 될듯하여 추가.

> Cell state : 기존 RNN과 달리 덧셈 연산으로 구성되어 기울기 소실 문제를 해결할 수 있 다.
> forget gate에서는 과거의 cell state 에서 어떤 정보를 제거할지 결정한다.

## LSTM 구현

아래 식들이 LSTM에서 수행하는 계산임

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

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

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

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

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

$h_t = o \odot tanh(c_t)$

여기서 주목할 점은 상위 4개의 수식에 포함된 아핀 변환 임 (affine transformation)

- *어파인 변환이란 행렬 변환과 평행 이동 (편향)을 결합한 형태*, 즉 $xW_x + hW_h + b$ 형태의 식을 가리킴
  위 4개의 수식은 아핀 변환을 개별적으로 수행하지만, 이를 하나의 식으로 정리해 계산 가능함
- **신경망의 순전파 때 수행하는 행렬의 곱은 기하학에서는 어파인 변환 이라고 한다.**
- 위와같이 신경망의 순전파에서는 가중치 신호의 총합을 계산하기 때문에 행렬의 곱을 사용하게 되고, 행렬의 곱계산은 대응하는 차원의 원소 수를 일치시키는게 핵심이기 때문

아래는 시각적으로 설명:

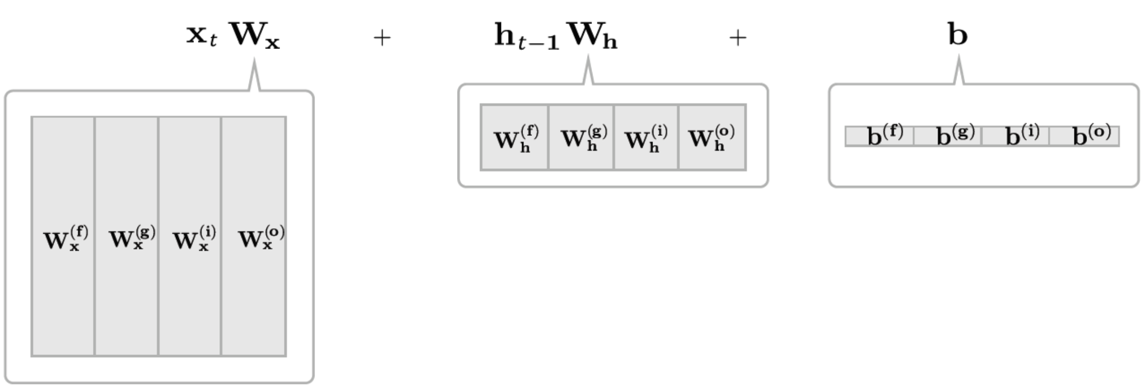
**그림 6-20** 각 식의 가중치들을 모아 4개의 식을 단 한 번의 아핀 변환으로 계산

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

$$\downarrow$$

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

$$\downarrow$$



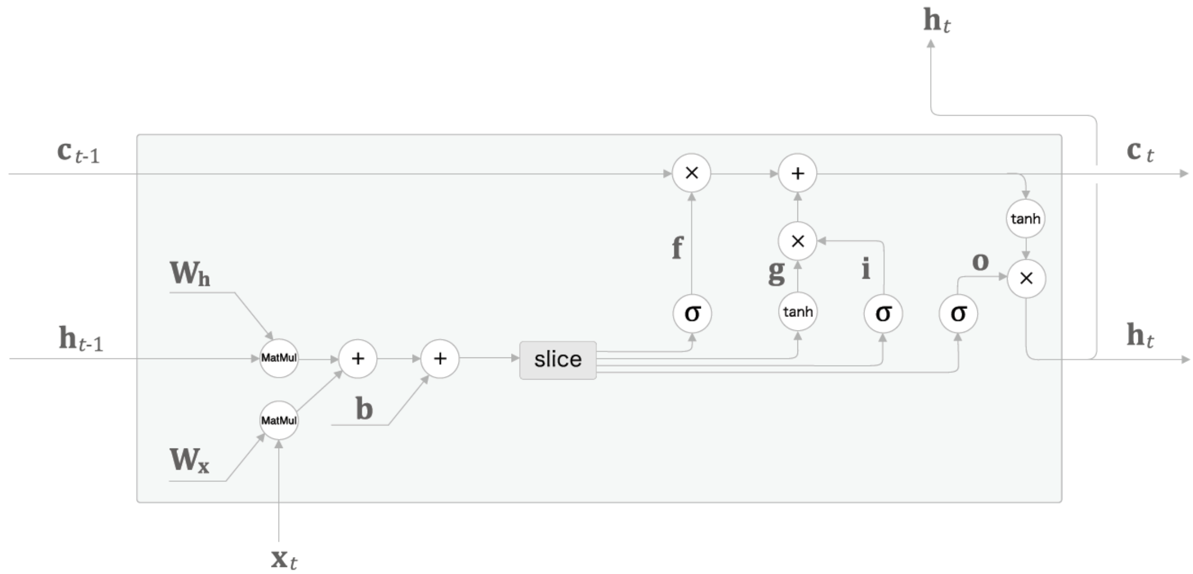

그림에서 보듯 4개의 가중치 (또는 편향)를 하나로 모을 수 있고, 그렇게 하면 원래 개별적으로 총 4번을 수행하던 아핀 변환을 단 1회의 계산으로 끝마칠 수 있음!

장점: 계산속도 빨라짐

- 일반적으로 행렬 라이브러리는 '큰 행렬'을 한꺼번에 계산할 때각 각각을 따로 계산할 때보다 빠르기 때문임
- 가중치를 한 데로 모아 관리하게 되어 소스 코드도 간결해짐

그리하여~ $W_x, W_h, b$ 각각에 4개분의 가중치 (혹은 편향)가 포함되어 있다고 가정하고, 이때의 LSTM을 그래프로 그려보면 다음과 같음

그림 6-21 4개분의 가중치를 모아 아핀 변환을 수행하는 LSTM의 계산 그래프




그림과 같이 처음 4개분의 아핀 변환을 한꺼번에 수행하고 slice 노드를 통해 4개의 결과를 다시 나눠줌.

- slice는 아핀 변환의 결과(행렬)를 균등하게 네 조각으로 나눠서 꺼내주는 단순한 노드임.
- slice 노드 다음에는 활성화 함수만 적용하면 간단하게 적용됨

## 어파인 변환 형상 추이

그림 6-22 아핀 변환 시의 형상 추이(편향은 생략)

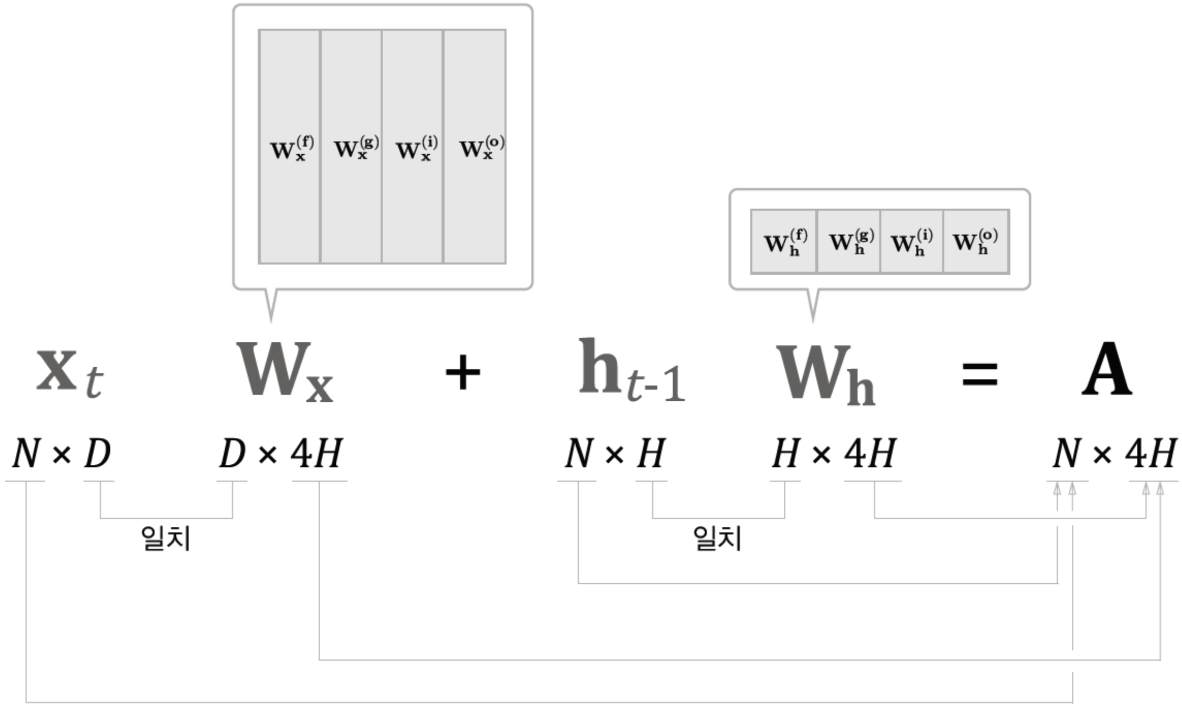
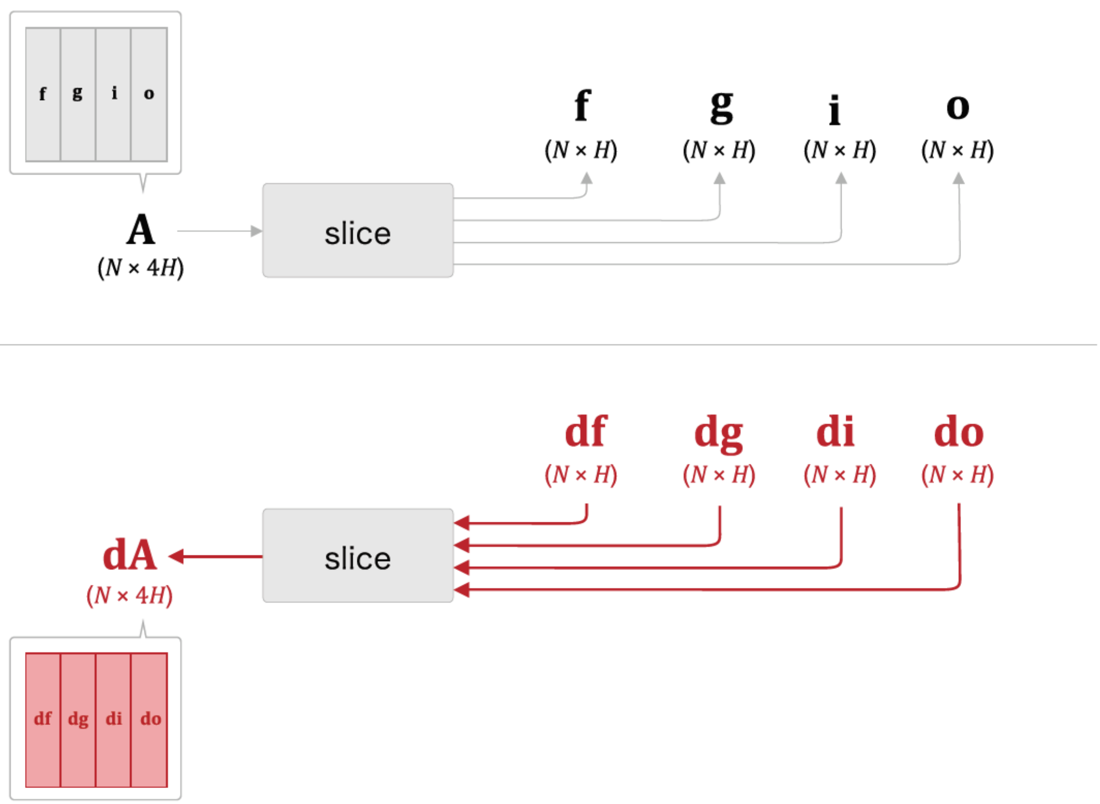
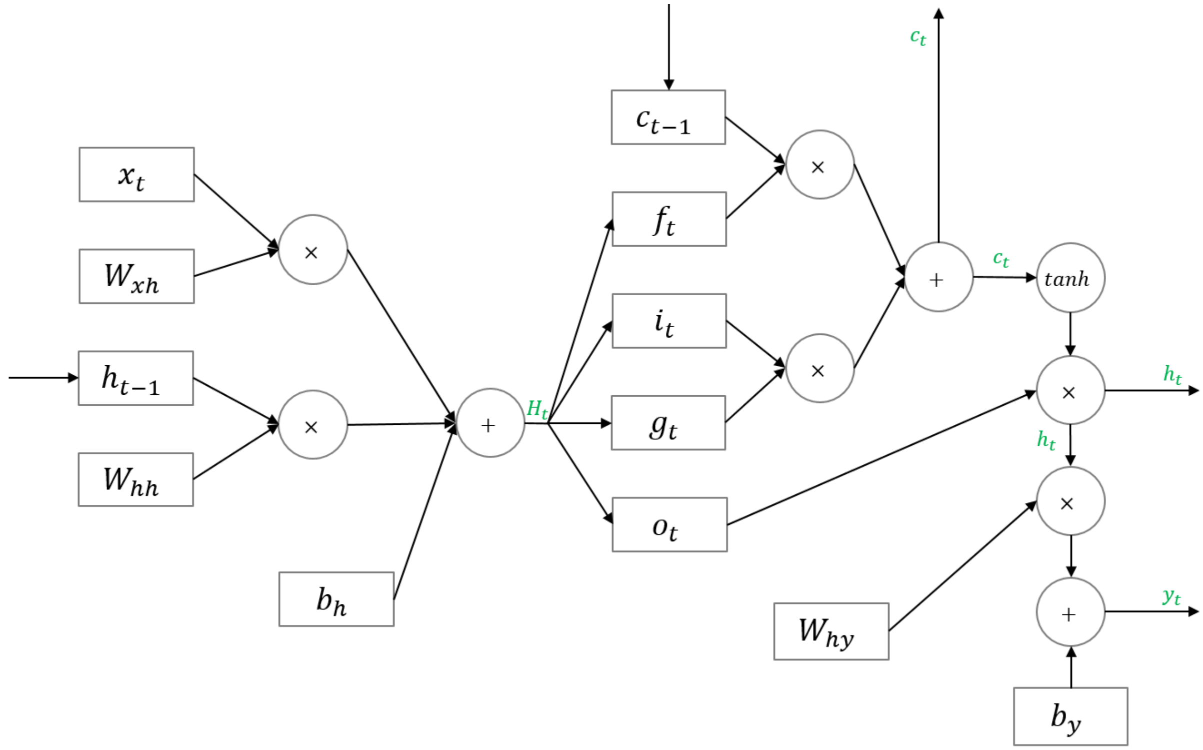


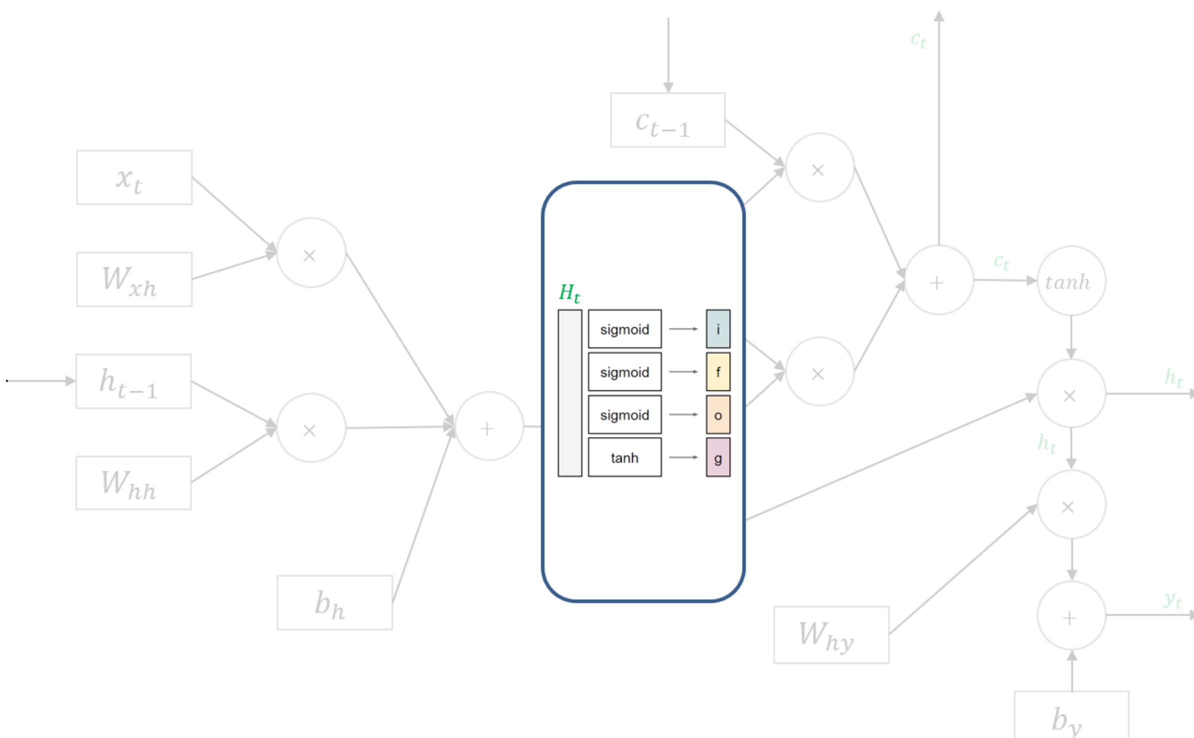

그림 6-23 slice 노드의 순전파(위)와 역전파(아래)



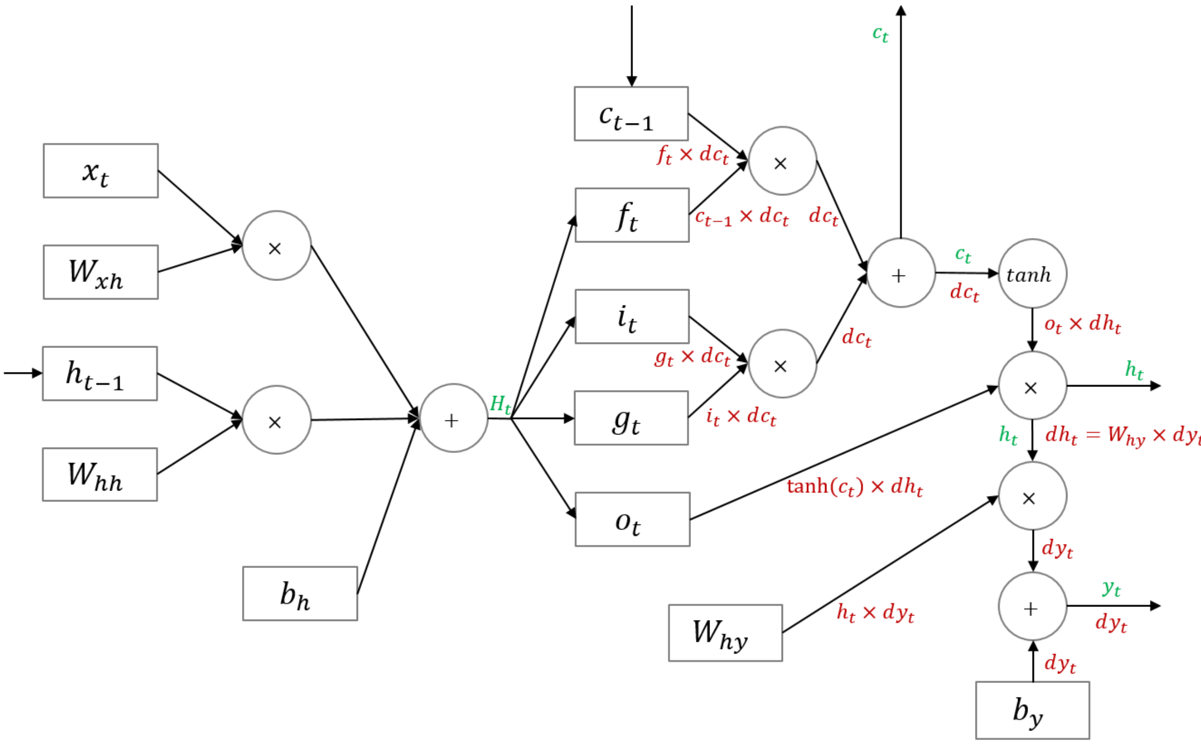

*자세한 순전파와 역전파에 대한 설명 콰츠고~!*

## LSTM의 순전파

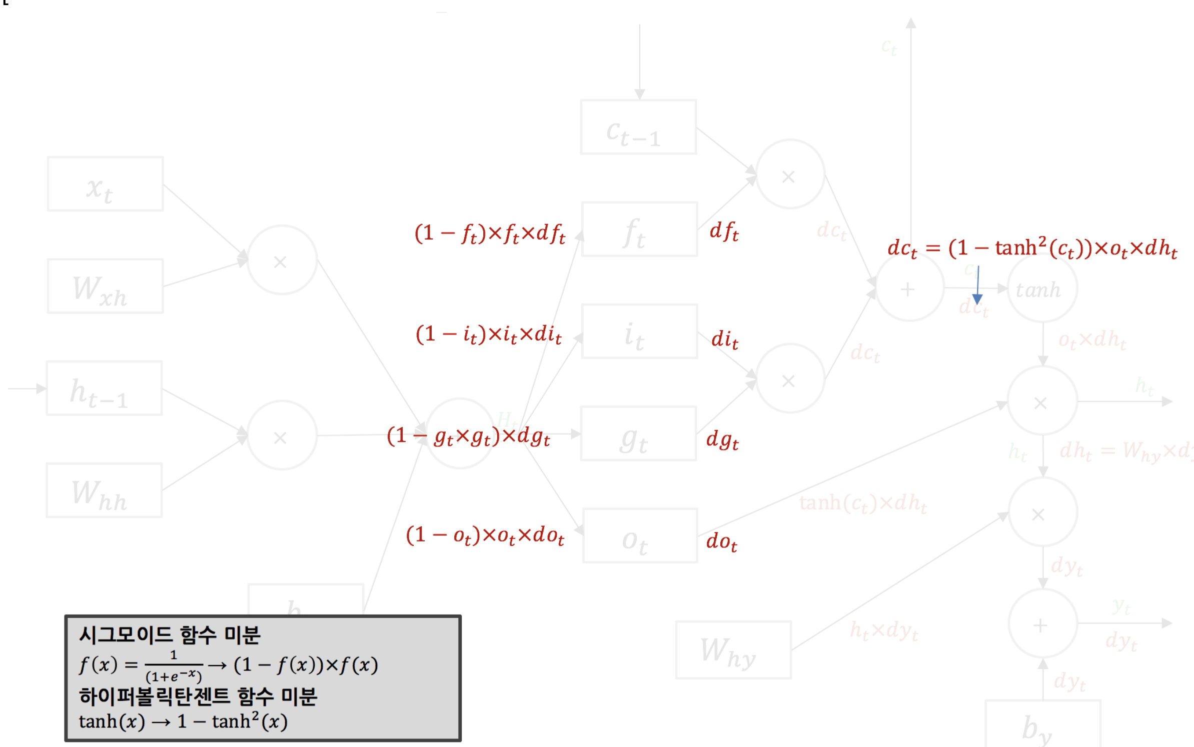

$x_t$
$W_{xh}$
$h_{t-1}$
$W_{hh}$
$b_h$
$H_t$
$c_{t-1}$
$f_t$
$i_t$
$g_t$
$o_t$
$c_t$
tanh
$h_t$
$W_{hy}$
$y_t$
$b_y$

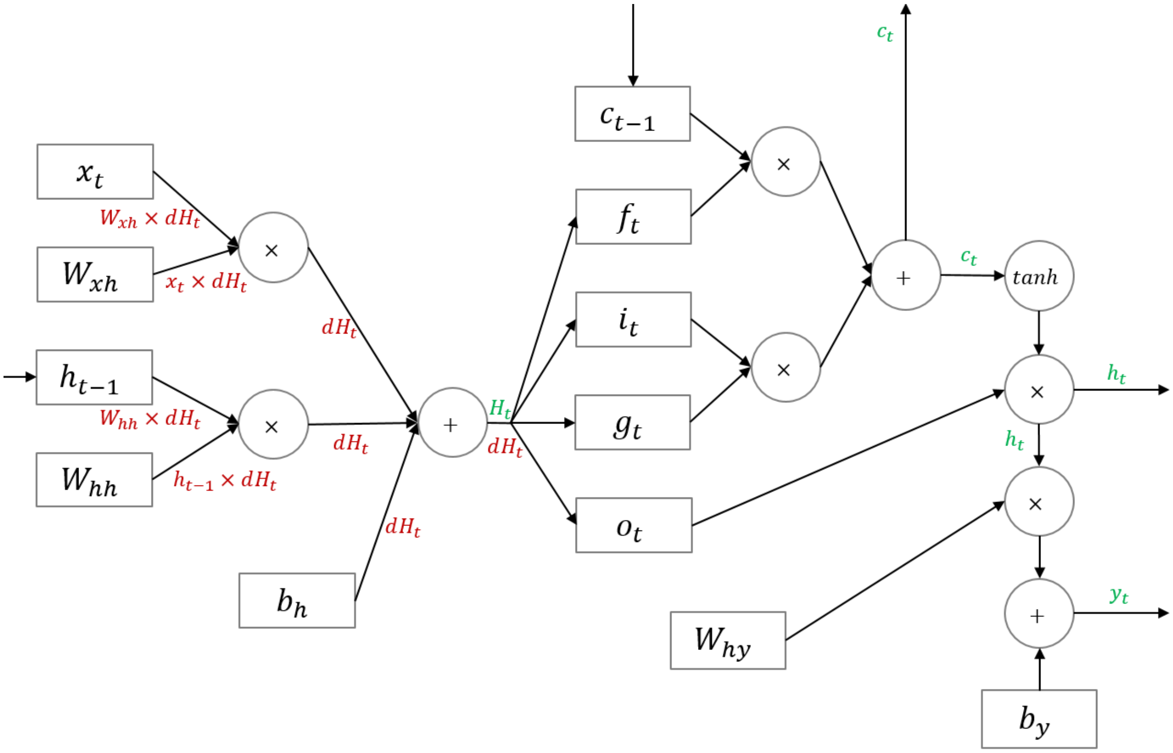

$H_t$
sigmoid
sigmoid
sigmoid
tanh
i
f
o
g

## LSTM의 역전파


$x_t$
$W_{xh}$
$h_{t-1}$
$W_{hh}$
$b_h$
$H_t$
$c_{t-1}$
$f_t$
$i_t$
$g_t$
$o_t$
$f_t \times dc_t$
$c_{t-1} \times dc_t$
$dc_t$
$g_t \times dc_t$
$i_t \times dc_t$
$c_t$
$tanh$
$o_t \times dh_t$
$h_t$
$dh_t = W_{hy} \times dy_t$
$\tanh(c_t) \times dh_t$
$W_{hy}$
$h_t \times dy_t$
$dy_t$
$y_t$
$b_y$

[


$(1 - f_t)\times f_t \times df_t$
$df_t$
$(1 - i_t)\times i_t \times di_t$
$di_t$
$(1 - g_t \times g_t)\times dg_t$
$dg_t$
$(1 - o_t)\times o_t \times do_t$
$do_t$
$dc_t = (1 - \tanh^2(c_t))\times o_t \times dh_t$
시그모이드 함수 미분
$f(x) = \frac{1}{(1+e^{-x})} \rightarrow (1 - f(x))\times f(x)$
하이퍼볼릭탄젠트 함수 미분
$\tanh(x) \rightarrow 1 - \tanh^2(x)$




# LSTM의 변칙

여기까지가 보통의 LSTM 설명이었음
모든 LSTM이 설명과 같은 동일한 구조를 갖고 있지는 않는다. 그리고 LSTM을 구현한 모든 논문들은 서로서로 약간 다른 구현체 버전을 갖고 있는다.
그중 좀 유명한 LSTM 변칙 패턴을 갖고 있는 버전이 'peephole connections'임.

## LSTM의 변칙 1 : 핍홀 연결 (peephole connections)

게이트가 cell state 자체를 입력값으로 받는 방식

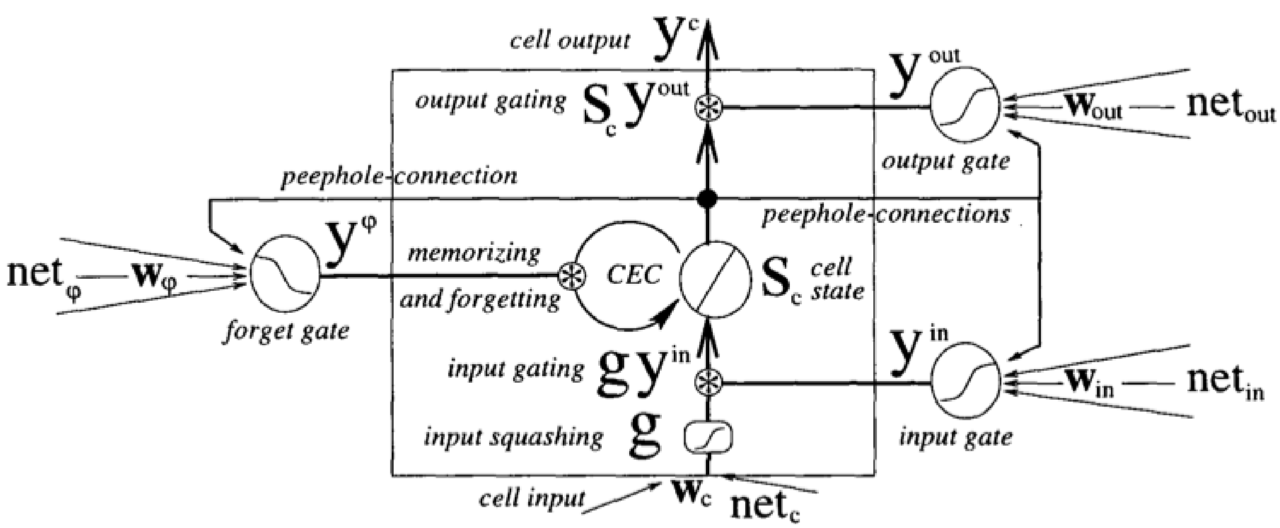
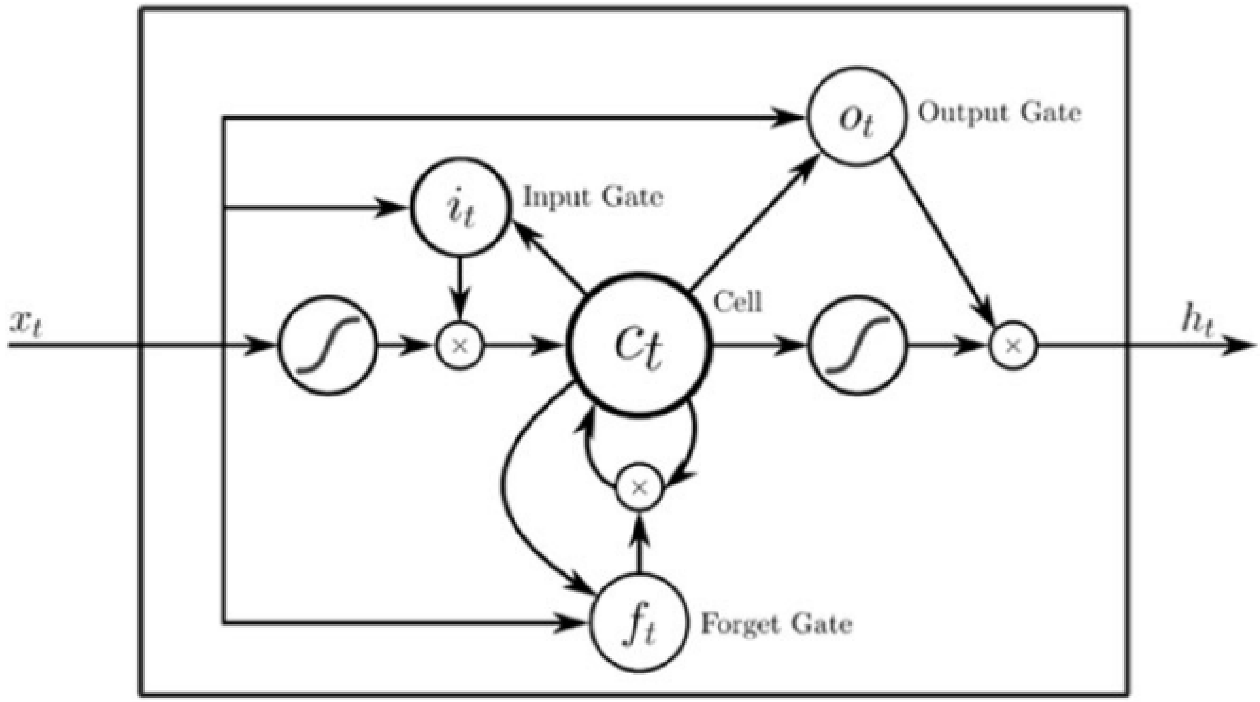
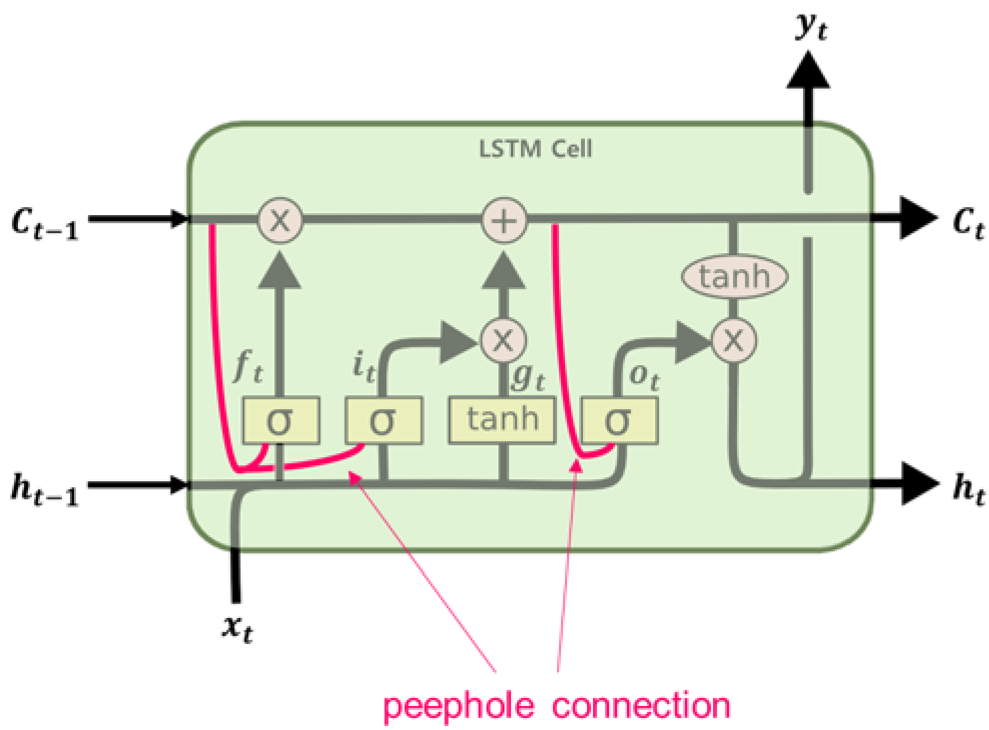



Figure 1: LSTM memory block with one cell, peephole connections connect $s_c$ to the gates.




$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c) \\
h_t &= \sigma_h(o_t \circ c_t)
\end{aligned}
$$

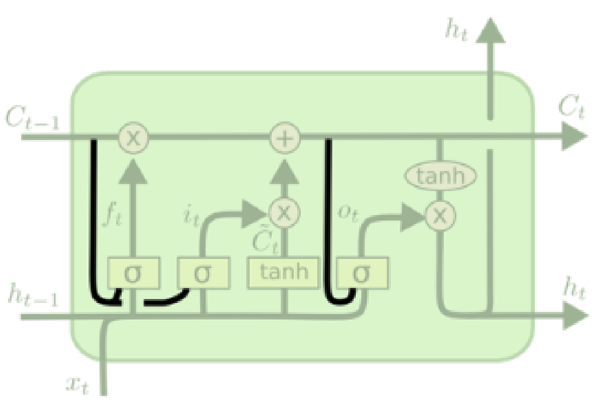
***Figure 5-30.*** *Representation of peephole LSTM*




Output gate가 Ct를 전달하기 때문에, LSTM 블록별 cell state는 output게이트에 따라 달라진다.
(input, forget 게이트는 C(t-1)을 전달함

Output 게이트가 계속 닫혀있는 경우 (아마 활성화함수 시그모이드가 0인 경우) cell state에 접근할 수 없다는 문제가 발생함. 이 문제를 해결하기 위해 도입된 것이 '핍홀 연결' 이다.

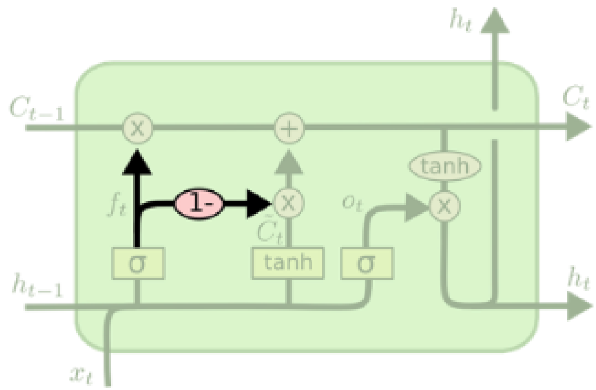
위 그림을 보면 cell state에 각 게이트를 연결하여, cell state를 각 게이트에 전달하는 것을 확인할 수 있다.




$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

## LSTM의 변칙 2 :

*망각 게이트와 입력 게이트를 합친* 버전.
어떤 값을 잊어버리고 어떤 값을 더할지 과정을 따로따로 수행하는 것이 아니라, 이를 동시에 결정하는 방식.
이때 새로운 값이 제공될 때만 이전 값을 잊어버리게됨.




$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

## LSTM의 변칙 3 : 그건 바로 GRU (the Gated Recurrent Unit)

2014년 뉴욕대학교 조경현 교수님 집필
*망각 게이트와 입력 게이트를 하나의 업데이트 게이트로 통일* 시킨 버전
추가적으로 셀 스테이트와 히든 스테이트를 합쳤으며 결과적으로 보통의 LSTM 모델보다는 단순해짐과 동시에 인기를 얻고 있다고 알려짐

추가적으로 Depth Gated RNNs 및 완전 다른 방식으로 장기 의존성 문제를 해결한 Clockwork RNNs 등 도 있다.

LSTM은 RNN으로 성취할 수 있는 하나의 빅 스텝인데, 그럼 다른 빅 스텝은 없는걸까요..? 라는 말이 *brunch*에 소개되어 있다. 그리고 그 다른 빅 스텝은 바로 **Attention** 임!

### 참고:

*사이트1*
*https://wegonnamakeit.tistory.com/7*
*# Recurrent nets that time and count*