



Week 09. Lecture 7 – Vanishing Gradients, Fancy RNNs

발표자: 김나현, 조서영

목차

#01 Vanishing gradient problem

#02 LSTM

#03 GRU

#04 More fancy RNN variants



Vanishing gradient problem



Vanishing gradient problem

RNN의 장단점

장점

- 입력의 길이에 제한이 없음
- 길이가 긴 timestep t 에 대해 처리 가능함 (이론적으로는 그러함, 실제로는...)
- 입력에 따라서 모델 크기가 증가하지 않음
- 매 timestep t 에 동일한 가중치를 적용하므로 symmetric

단점

- 다음 단계로 진행하기 위해서는 이전 단계의 계산이 완료되어야 하므로 계산이 병렬적으로 진행되지 않음
→ 느림
- 실제로는 vanishing gradient problem 등의 문제가 있어 context가 반영되지 않는 경우가 있음

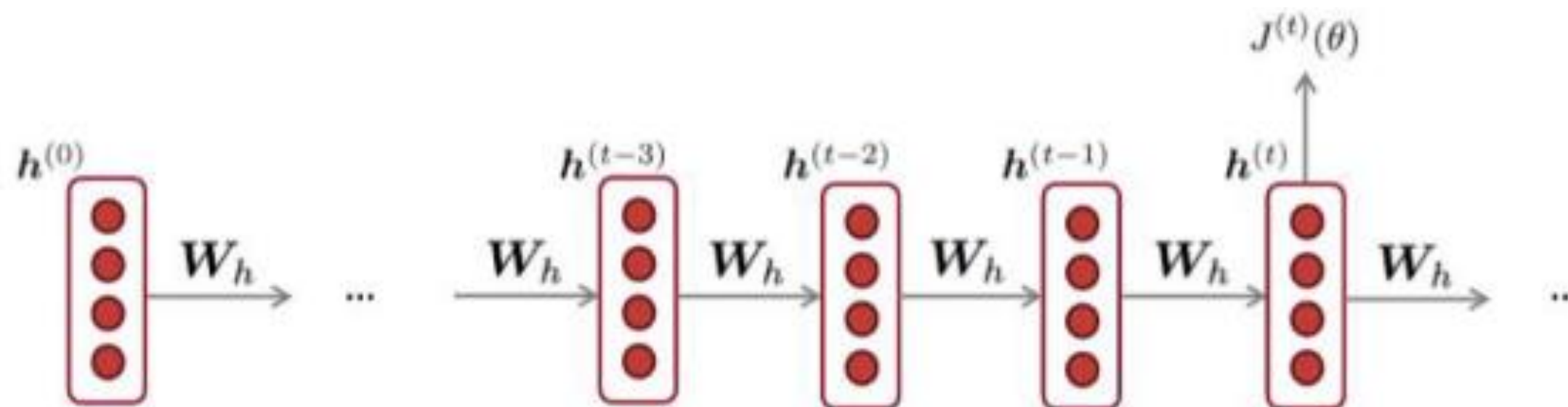
Vanishing gradient problem

What is vanishing gradient problem?

- RNN 역전파 시 gradient가 너무 작아지거나, 반대로 너무 커져서 학습이 제대로 이뤄지지 않는 문제

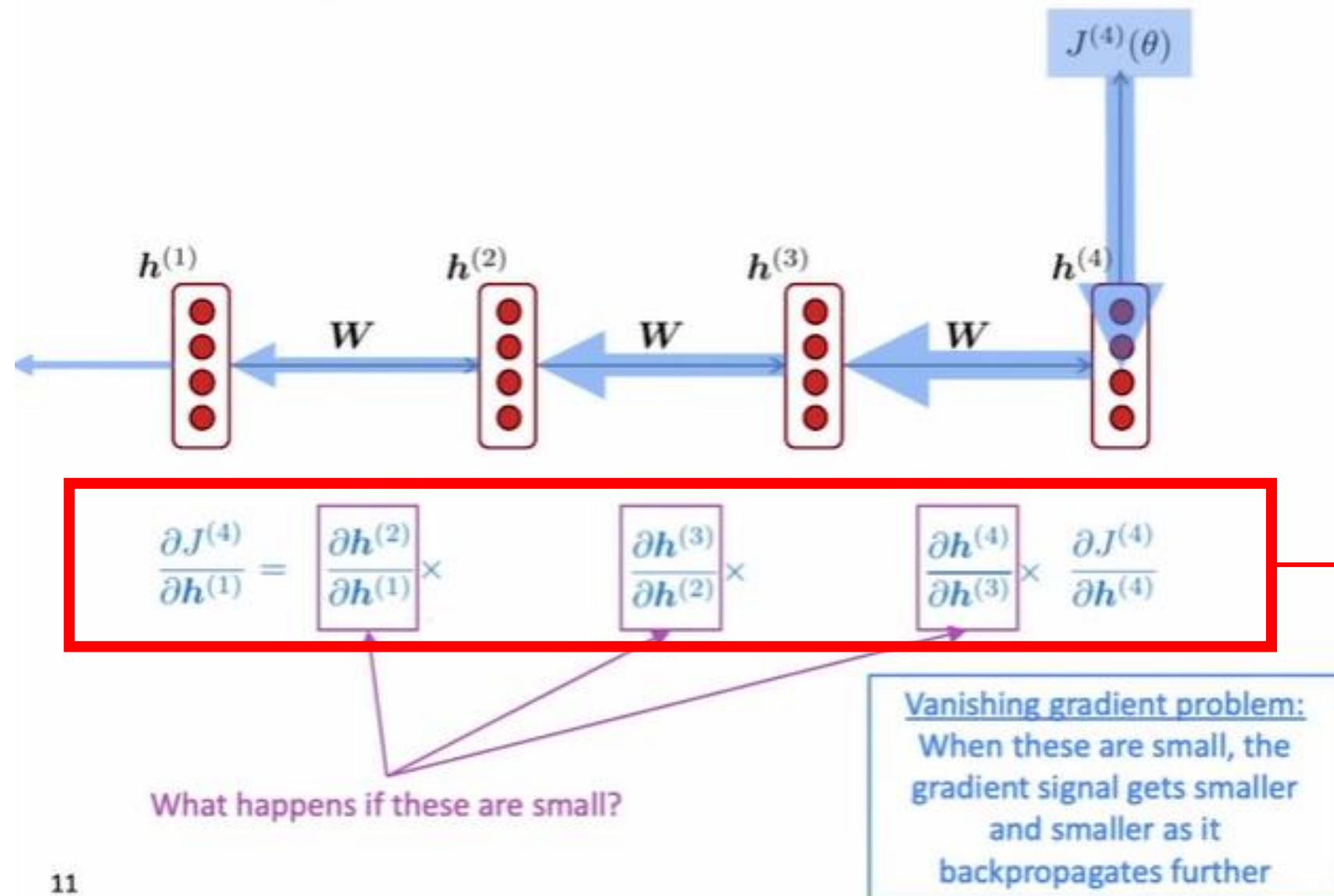
수학적 설명

- 기본적 RNN 셀 t번째 시점의 hidden state:
$$h^{(t)} = \sigma \left(W_h h^{(t-1)} + W_x x^{(t)} + b_1 \right)$$



Vanishing gradient problem

- 기본적 RNN 셀 t번째 시점의 hidden state: $h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(t)} + b_1)$



- By chain rule, 네 번째 시점의 손실 J에 대한 hidden state h의 gradient는 다음과 같이 계산된다.

$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

Vanishing gradient problem

- 앞의 식을 일반화해서 쓰면 다음과 같다.

$$\frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \quad \leftarrow \text{Chain rule에 의해 곱해지는 부분} \quad (\text{chain rule})$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \mathbf{W}_h^{(i-j)} \prod_{j < t \leq i} \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \quad \left(\text{value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \right)$$

- ∴ t번째 hidden state에 대한 t-1번째 hidden state의 gradient:

$$\mathbf{h}^{(t)} = \sigma \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right)$$
$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h$$

Vanishing gradient problem

- 손실 J에 대한 hidden state h의 gradient

$$\begin{aligned}\frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \leftarrow \text{Chain rule에 의해 곱해지는 부분} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \mathbf{W}_h^{(i-j)} \prod_{j < t \leq i} \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + b_1 \right) \right) && \text{(value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \text{)}\end{aligned}$$

- Norm의 성질에 의해 다음 부등식 성립

$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \right\| \|\mathbf{W}_h\|^{(i-j)} \prod_{j < t \leq i} \left\| \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + b_1 \right) \right) \right\|$$

$$\Rightarrow \|\mathbf{W}_h\|_2 = \sqrt{\lambda_{\max}} \quad \leftarrow \mathbf{W}_h \text{의 L2 norm은 } \mathbf{W}_h \text{의 가장 큰 고유값(eigenvalue)이다!}$$

※ L2 norm: L2 Norm 은 행렬 원소의 모든 값들의 제곱 합에 루트를 씌운 것이다.
좌표계에서 두 점 사이의 거리라고도 할 수 있다.

Vanishing gradient problem

- 손실 J에 대한 hidden state h의 gradient

$$\frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \quad \leftarrow \text{Chain rule에 의해 곱해지는 부분 이 값의 L2} \quad (\text{chain rule})$$

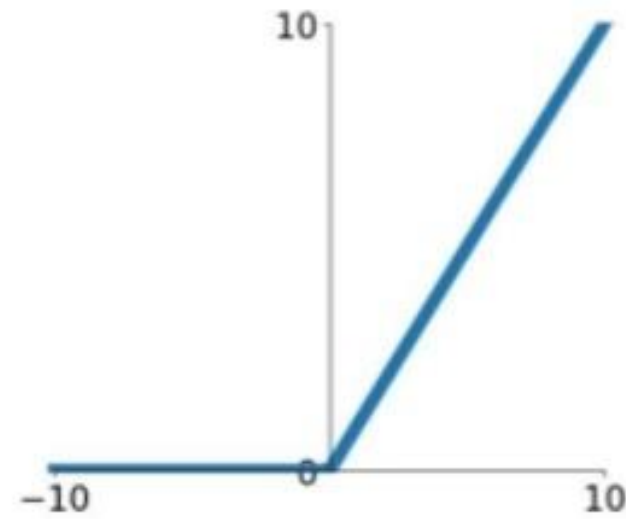
$$= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \boxed{W_h^{(i-j)}} \prod_{j < t \leq i} \text{diag} \left(\sigma' \left(W_h \mathbf{h}^{(t-1)} + W_x \mathbf{x}^{(t)} + b_1 \right) \right) \quad \left(\text{value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \right)$$

↑ 이 값이 작으면 계속 곱해짐에 따라 값은 더욱 작아짐

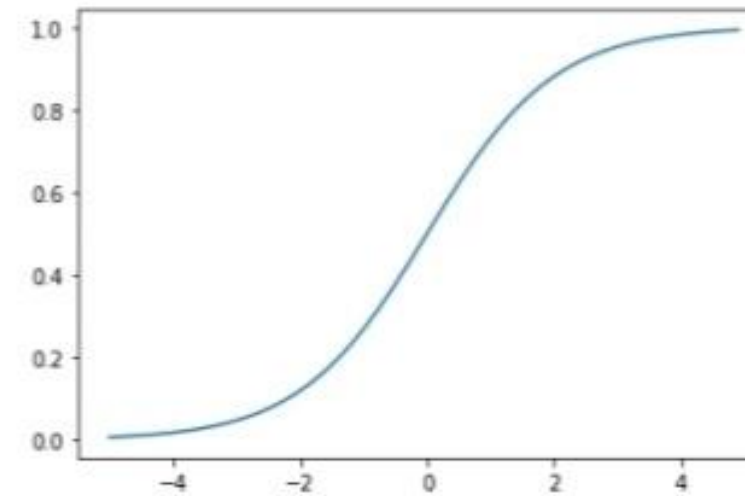
$$\|W_h\|_2 = \sqrt{\lambda_{\max}} \quad \leftarrow \text{이 값이 1보다 작다면 gradient } \left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\| \text{ 가 매우 작아질 것이다.}$$

Vanishing gradient problem

ReLU



Sigmoid



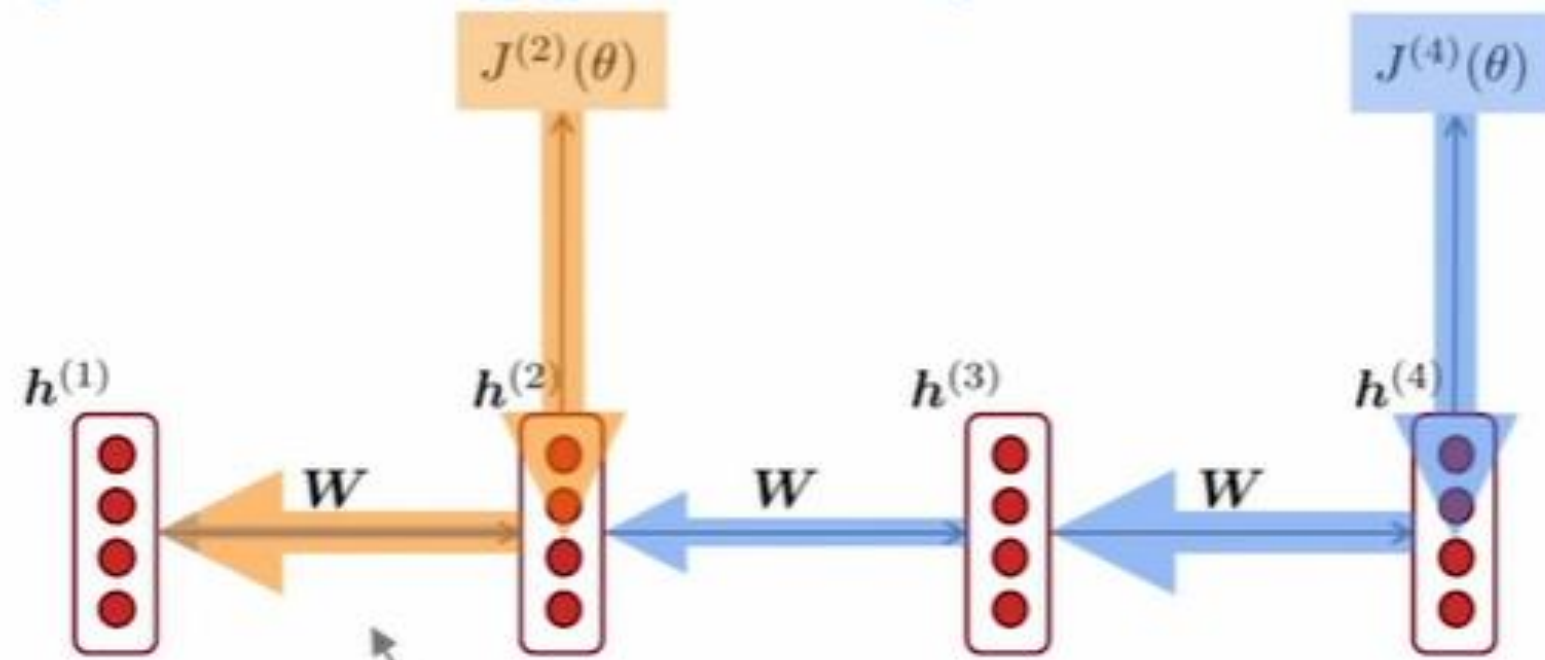
Sigmoid squashes activations between 0 and 1,
accelerating gradient vanishing

Now, you understand why ReLU is widely used

Vanishing gradient problem

- Why is it a problem?

Why is vanishing gradient a problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

- 멀리 떨어진 loss의 영향을 가까운 loss의 영향보다 훨씬 못 받게 된다.

→ 결과적으로 weight update를 할 때 가까운 곳에 대해 update 되고 먼 곳의 영향은 적어질 수 있다.

Vanishing gradient problem

- Why is it a problem?
 - Gradient는 과거가 미래에 얼마나 영향을 미치는지를 나타내 주는 척도이다.
 - 만약 gradient가 너무 작아진다면, 우리는 그것이
 - 1. 실제로 dependency가 존재하지 않아서인지
 - 2. dependency는 존재하지만 parameter를 잘못 설정해서 그런 것인지 알 수 없다.

Vanishing gradient problem

- RNN Language Model에 vanishing gradient가 미치는 영향
 - Language Model task: 문장에서 다음에 올 단어 예측하는 것

Ex1)




LM task: *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her tickets*

- RNN-LM에 gradient vanishing problem이 있을 경우
멀리 있는 단어들 사이의 dependency를 학습하지 못하는 문제 발생
- 위의 예시에서 'tickets' 를 유추하기 어렵다

Vanishing gradient problem

- RNN Language Model에 vanishing gradient가 미치는 영향
 - RNN-LM에 gradient vanishing problem이 있을 경우
멀리 있는 단어들 사이의 dependency를 학습하지 못하는 문제 발생

Ex 2)

- **LM task:** *The writer of the books ____*

- **Correct answer:** *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct)

- **Sequential recency:** *The writer of the books are* (incorrect)


- 가까운 dependency를 더 잘 학습 → 'are' 라는 잘못된 결과
- Syntactic recency보다 sequential recency를 더 빠르게 학습 → 잘못된 결과

Vanishing gradient problem

- Exploding gradient problem
 - 기울기가 너무 커지는 문제
 - gradient가 너무 커지면 SGD update step도 너무 커진다
 - 너무 큰 learning rate를 사용하면 한 번의 업데이트 step의 크기가 너무 커져 잘못된 방향으로 학습 & 발산
- Solution
 - 그냥 learning rate를 작게 해버리면 학습 속도 너무 느려짐

→ Gradient Clipping

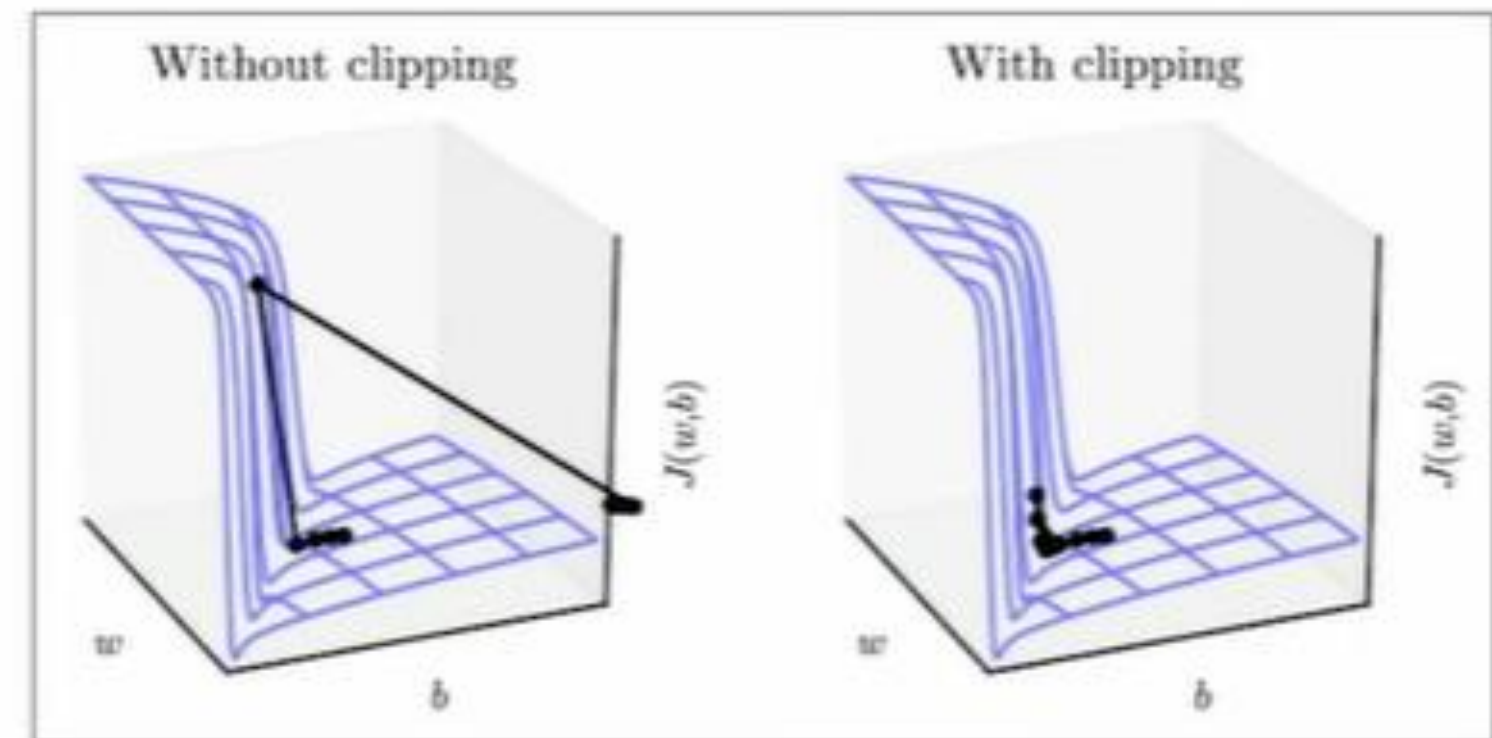
$$\theta^{new} = \theta^{old} - \underbrace{\alpha}_{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

Vanishing gradient problem

- Exploding gradient problem – solution: gradient clipping
 - Gradient가 일정 threshold를 넘어가면 gradient L2 norm으로 나눠주는 방식
 - 손실 함수를 최소화하기 위한 기울기의 방향은 유지한 채로 크기만 조절

Algorithm 1 Pseudo-code for norm clipping

```
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$   
if  $\|\hat{\mathbf{g}}\| \geq \text{threshold}$  then  
     $\hat{\mathbf{g}} \leftarrow \frac{\text{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```



Vanishing gradient problem

- OK, now we know what the vanishing gradient problem is & why it is problematic...

But how to fix this?

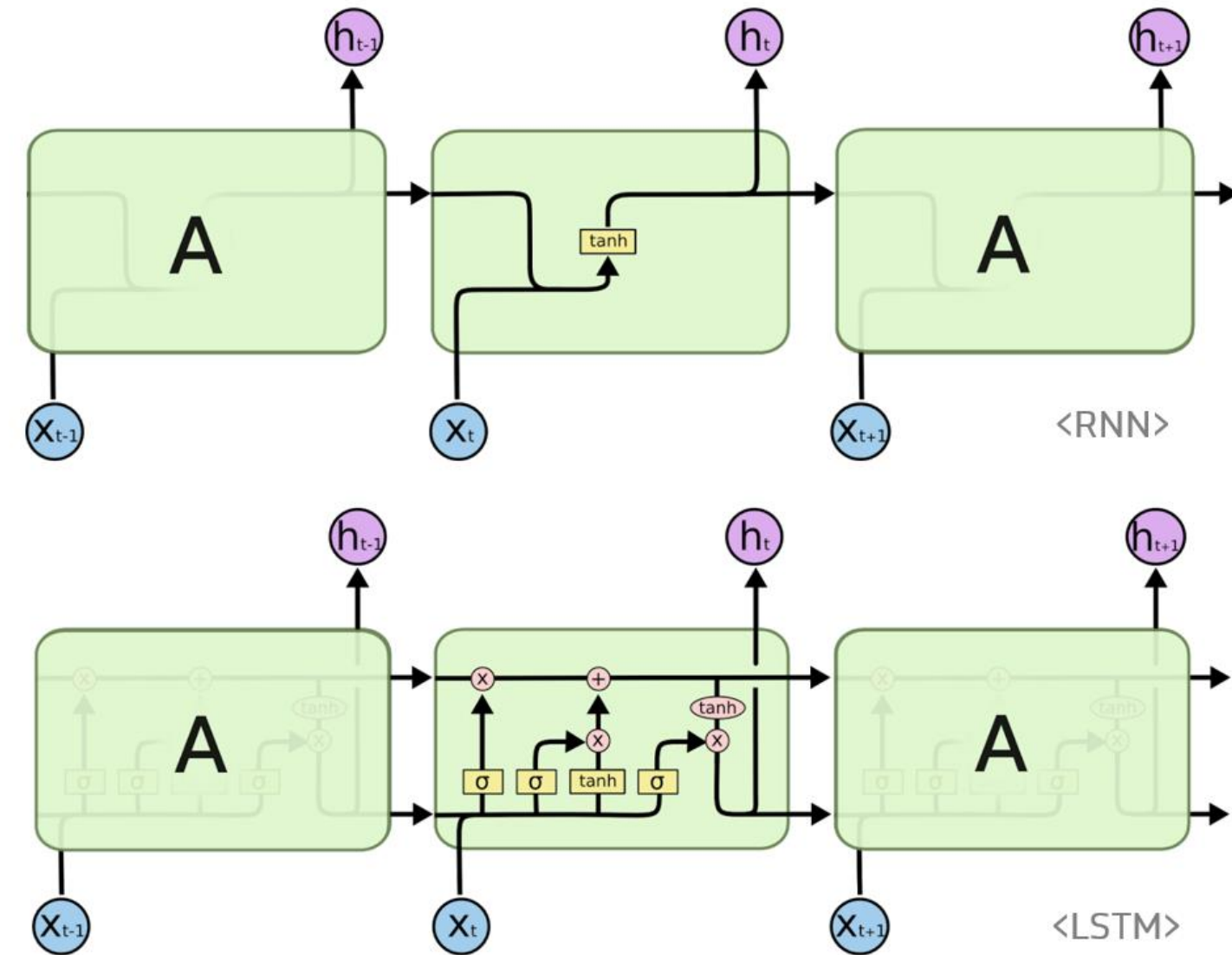
- RNN의 문제는 멀리 있는 정보를 제대로 학습하여 결과에 반영하지 못한다는 것.
- 만약 멀리 있는 정보를 저장할 수 있는 별도의 메모리가 있다면?

LSTM



LSTM

- Long Short-Term Memory (LSTM)
 - RNN에서 메모리를 분리하여 따로 정보를 저장함으로써 한참 전의 데이터도 함께 고려하여 출력을 만들고자 한 모델
 - 이전 단계의 정보를 memory cell에 저장해서 흘려 보낸다
 - 현재 시점의 정보를 바탕으로 과거의 내용을 얼마나 잊을지 곱해주고, 그 결과에 현재 정보를 더해서 다음 시점으로 정보를 전달



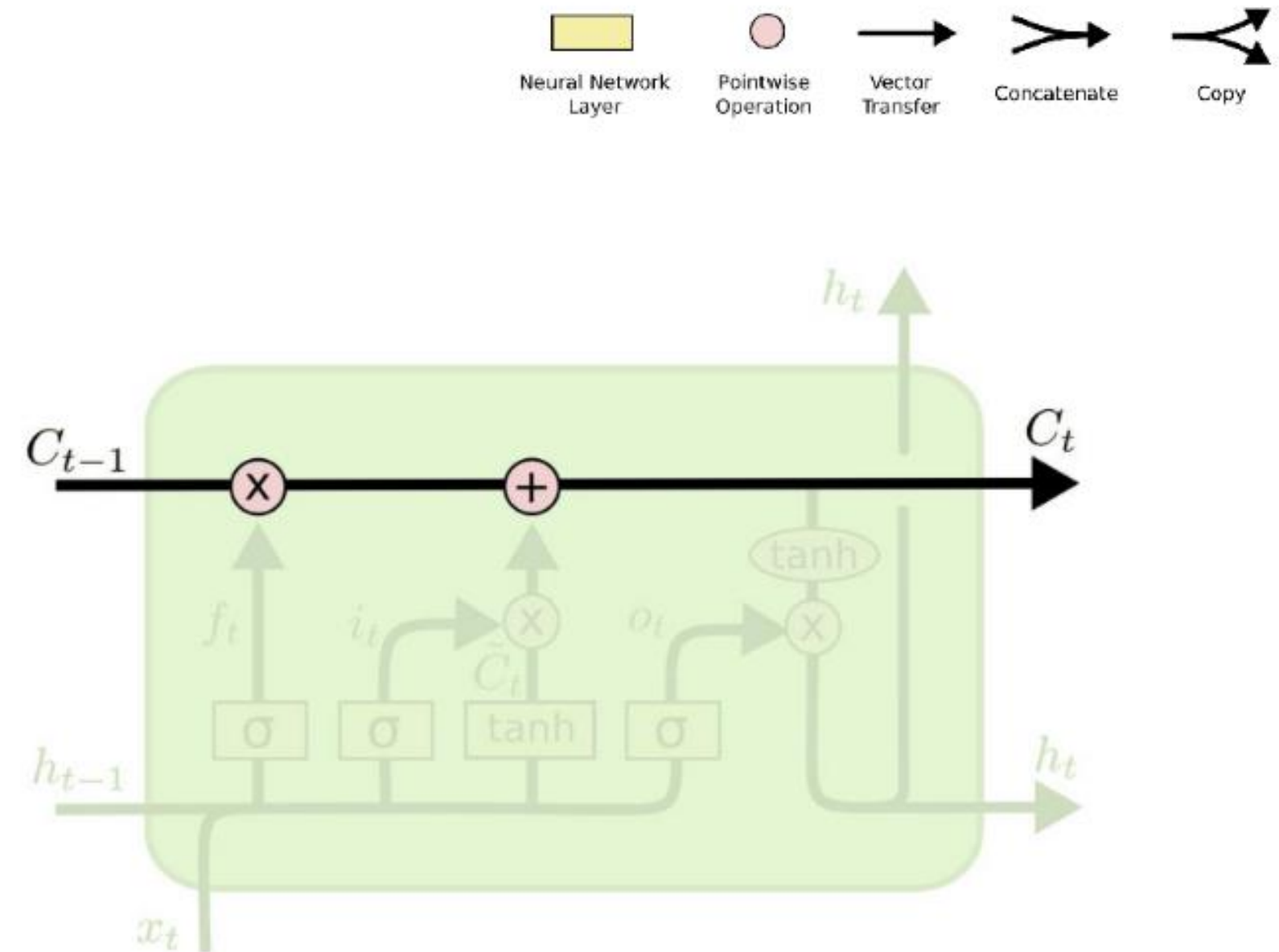
LSTM

- Cell state
 - Step t 일 때 hidden state h 와 cell state c 를 가진다
 - 둘 다 n 길이의 벡터이다
 - Cell은 long-term information을 저장한다
 - LSTM은 cell에 정보를 지우거나, 쓰거나, 읽어올 수 있다
 - 어떤 정보가 지워지고/쓰이고/읽히는지는 각각을 관리하는 세 개의 'gate'에 의해 관리된다
 - Gate 역시 n 길이의 벡터이다
 - 각 timestep에서, gate의 각 요소들은 open(1), closed(0), 혹은 그 중간의 값을 가질 수 있다 (sigmoid 함수를 통과하므로)
 - Gate 값은 현재의 상황에 기반해서 계산된 값이므로 계산될 때마다 달라진다 (dynamic)

LSTM

- Cell state

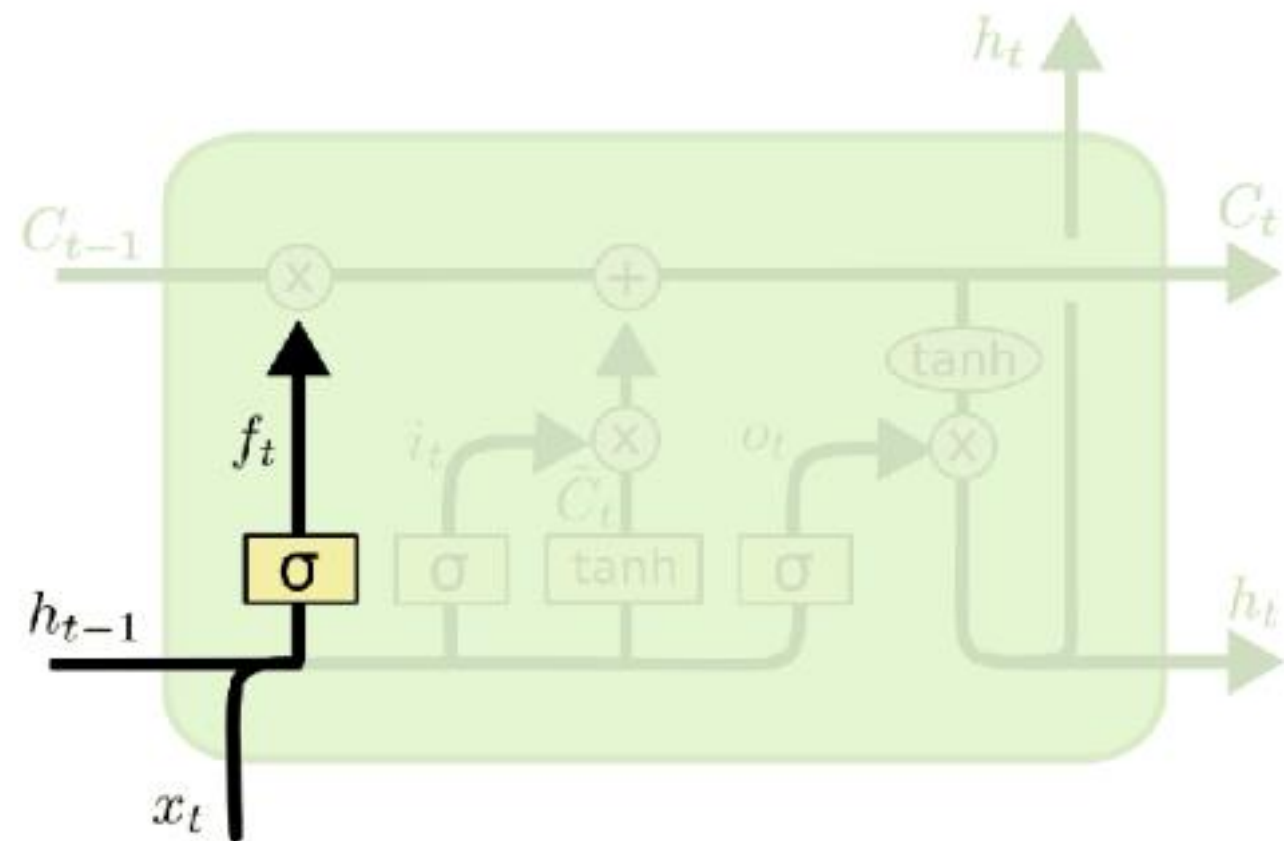
- 단순한 연산 (덧셈, 곱셈)을 거쳐 LSTM unit 통과
- 세 개의 gate들을 이용하여 정보의 반영 여부 결정
 - Input gate: 무엇을 쓸 것인가
 - Output gate: 무엇을 읽을 것인가
 - Forget gate: 무엇을 잊을 것인가



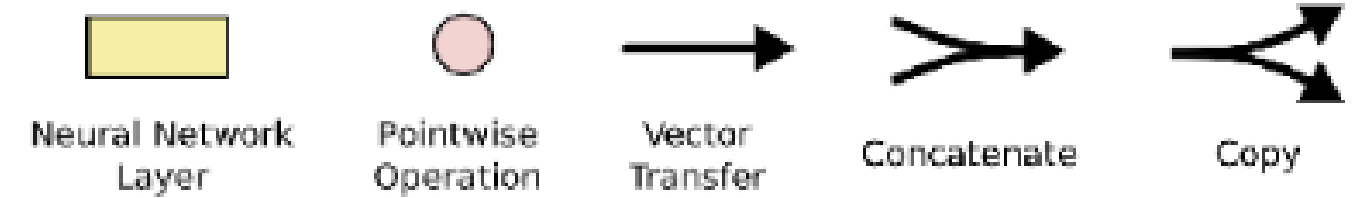
LSTM

- LSTM 단계

- 1) Forget gate layer



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

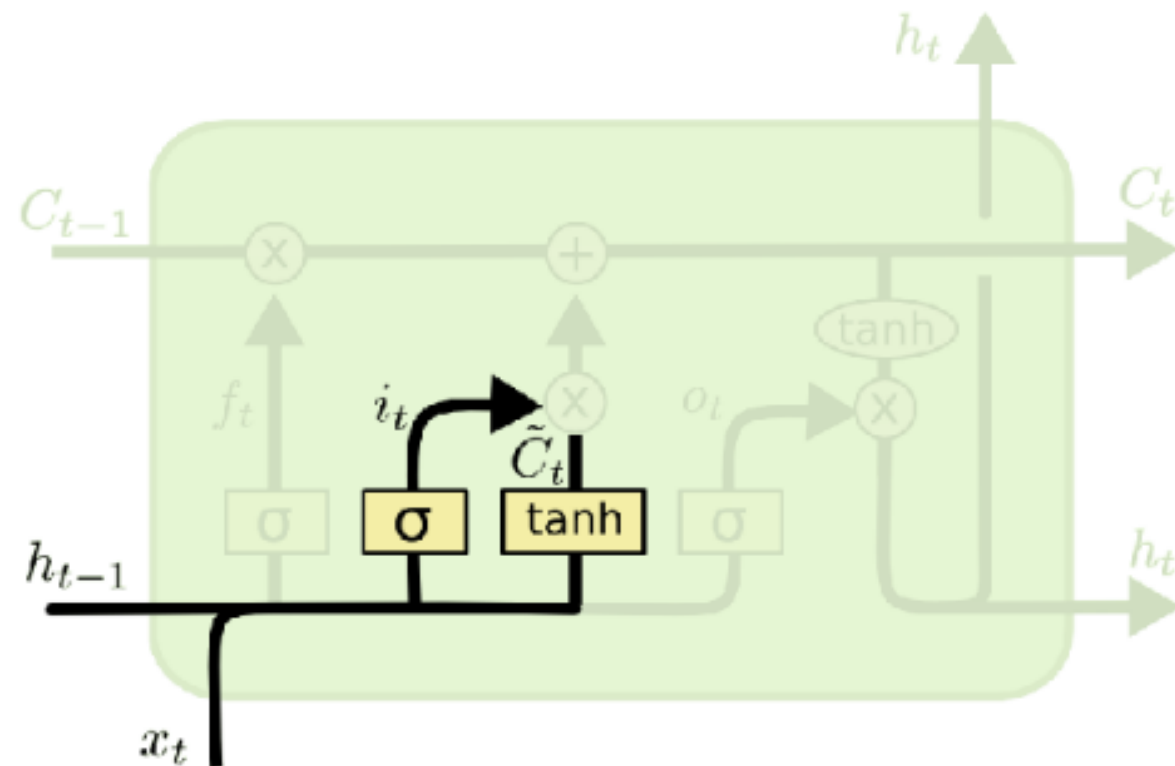


- 어떤 정보를 반영할지 결정하는 gate
- 이전 단계의 cell state에서 불필요한 정보를 지우는 역할
- Sigmoid를 거쳐 0~1사이 값으로 변환
- 0: 저장된 이전 정보 지움
- 1: 저장된 이전 정보 보존

LSTM

- LSTM 단계

- 2) Input gate layer



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

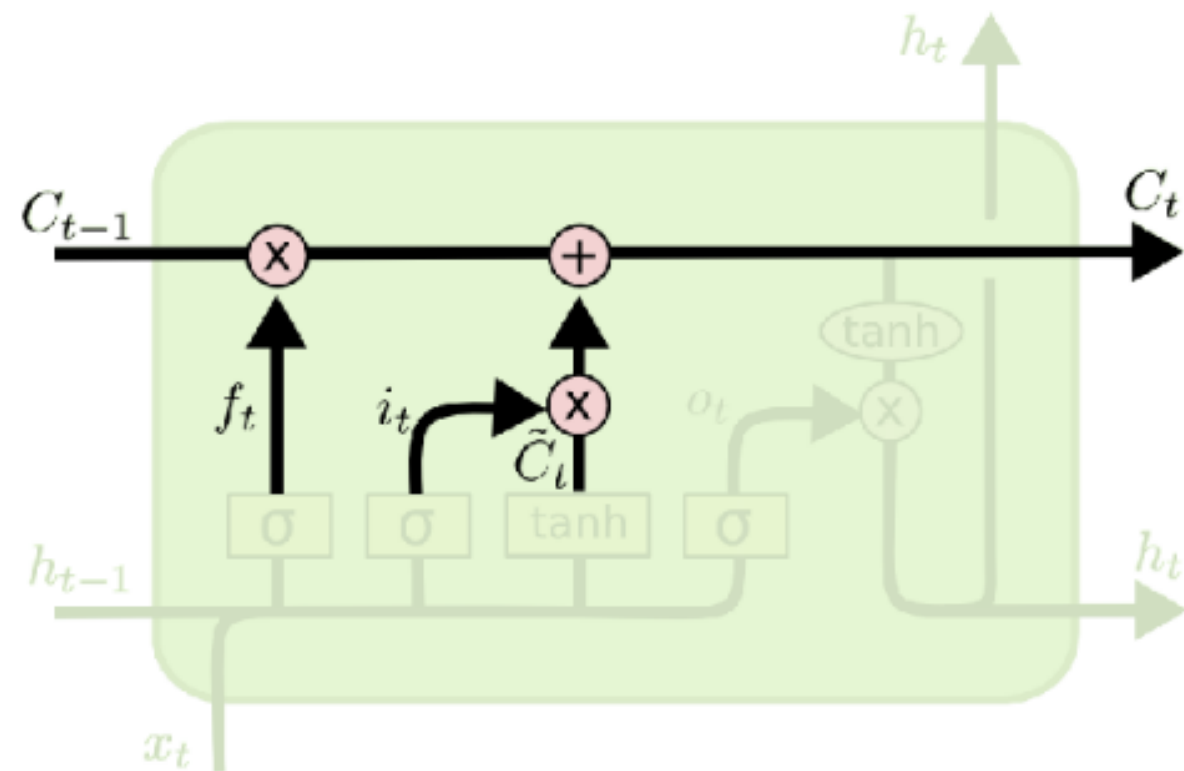


- 새로운 정보가 cell state에 저장될지 결정하는 gate
- input에 중요한 정보가 있다면 이전 단계의 cell state에 넣는 역할
- 2개의 layer
 - Sigmoid layer (input gate) : 어떤 값을 updat할지 결정
 - Tanh layer (update gate) : cell state에 더해질 벡터 만들
(vector of new candidate values)

LSTM

- LSTM 단계

- 3) Update cell state



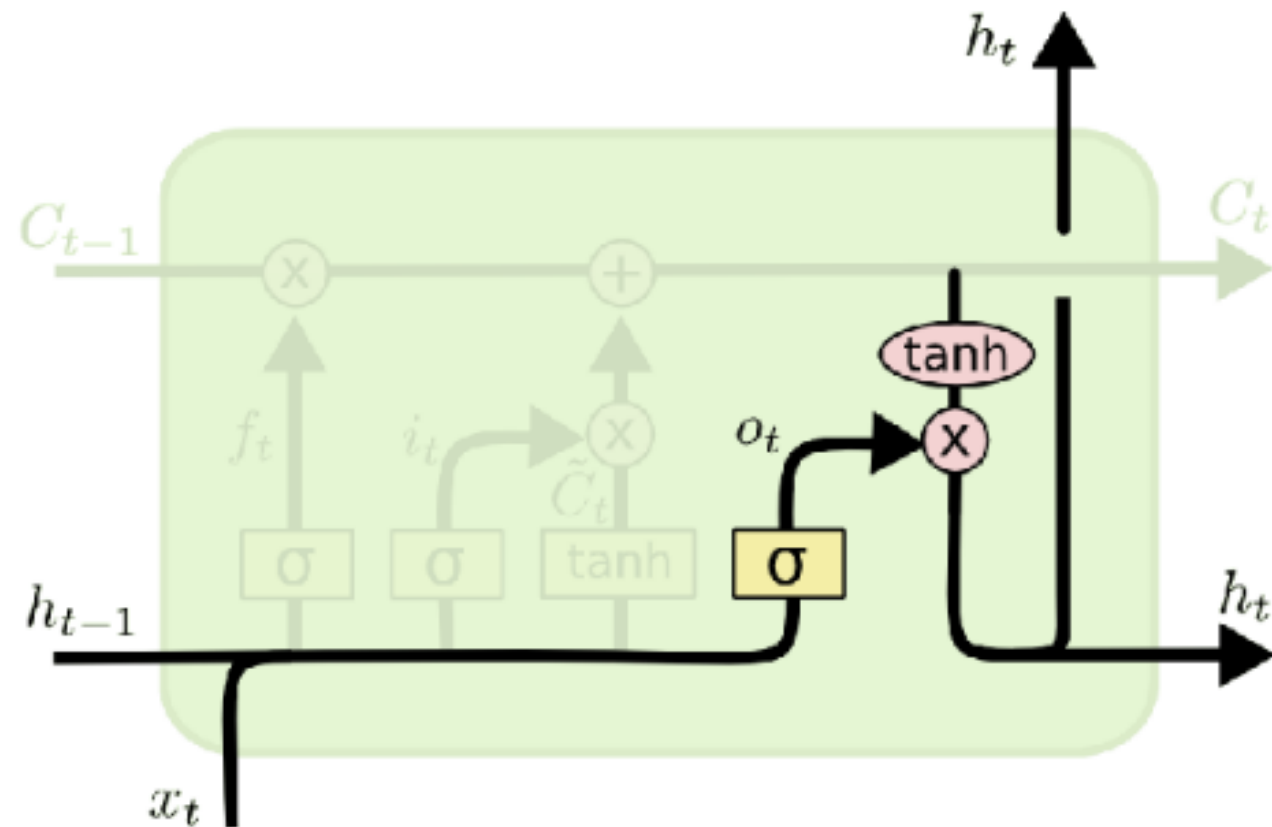
$$C_t = \underbrace{f_t * C_{t-1}}_{\text{과거}} + \underbrace{i_t * \tilde{C}_t}_{\text{현재}}$$

- f_t : 과거 정보 유지 여부
- i_t : 현재의 input값 반영 여부
- 최종적으로 더해져서 cell state update

LSTM

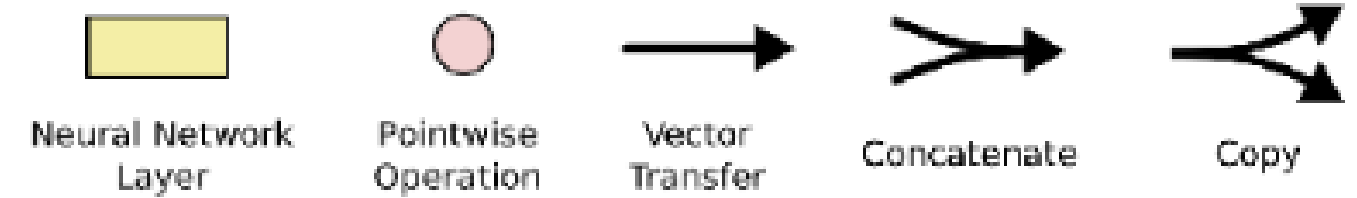
- LSTM 단계

- 4) Output gate layer



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

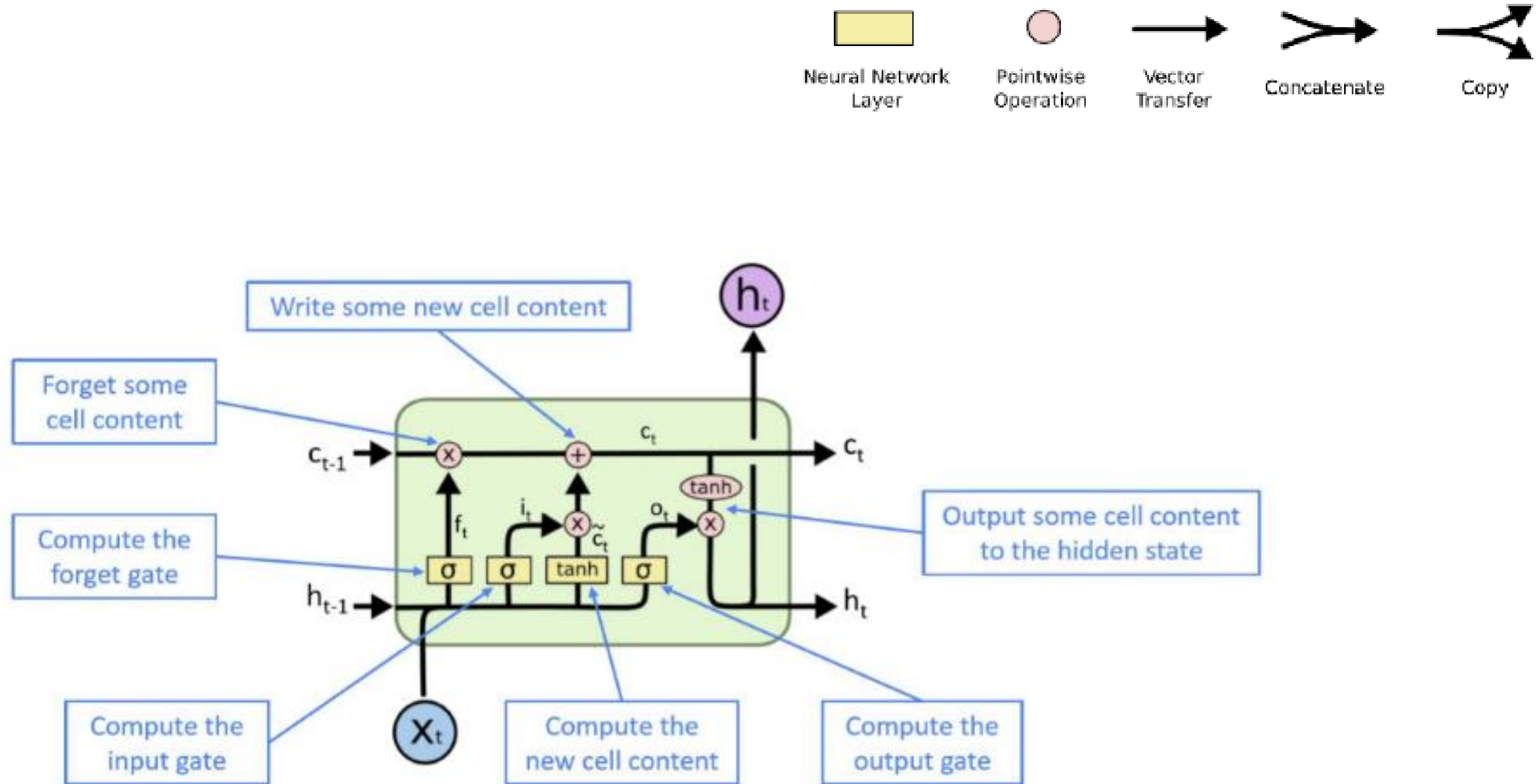
$$h_t = o_t * \tanh (C_t)$$



- RNN처럼 각 state마다 출력값을 반환
- o_t : output
- h_t : 출력값으로 나감 + 다음 state의 input으로 들어감

LSTM

- LSTM 단계



LSTM

- Step t 일 때 hidden state h 와 cell state c 를 가진다
 - 둘 다 n 길이의 벡터이다
 - Cell은 long-term information을 저장한다
 - LSTM은 cell에 정보를 지우거나, 쓰거나, 읽어올 수 있다
- 어떤 정보가 지워지고/쓰이고/읽히는지는 각각을 관리하는 세 개의 'gate'에 의해 관리된다
 - Gate 역시 n 길이의 벡터이다
 - 각 timestep에서, gate의 각 요소들은 $\text{open}(1)$, $\text{closed}(0)$, 혹은 그 중간의 값을 가질 수 있다 (sigmoid 함수를 통과하므로)
 - Gate 값은 현재의 상황에 기반해서 계산된 값이므로 계산될 때마다 달라진다 (dynamic)

LSTM

- How does LSTM solve vanishing gradients?
 - 여러 timestep 이전의 값도 저장하기 쉽게 해준다
 - 만약 forget gate가 모든 timestep값을 저장하도록 설정된다면, cell 내부의 정보는 계속 저장된다
 - 반면에 기본 RNN은 별도의 저장 장치가 없어 저장이 힘들다
 - 물론 LSTM도 vanishing / exploding gradient 문제가 없다고 보장할 수는 없다.
그러나 이러한 문제를 피하기 쉬운 모델인 것은 맞다.

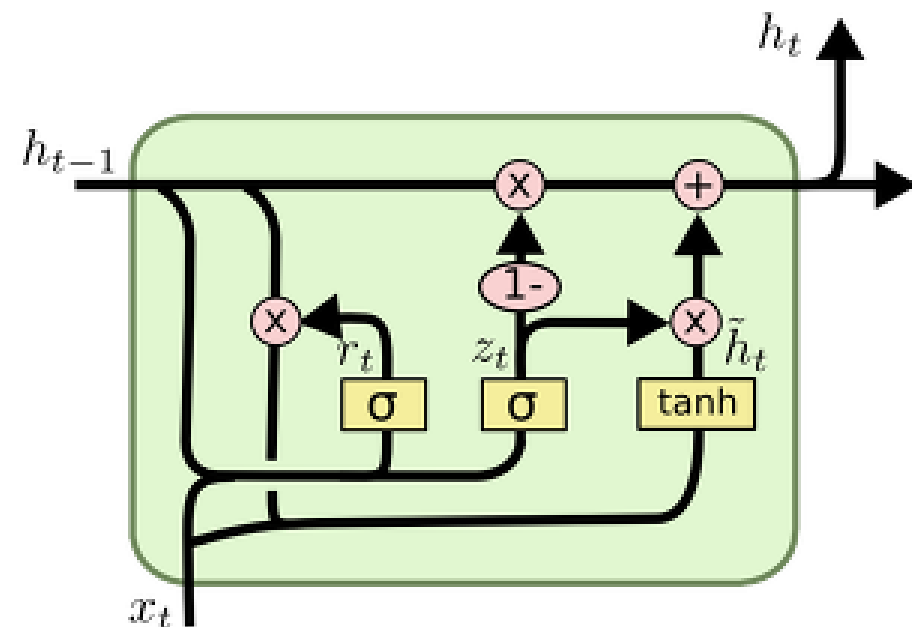
Gated Recurrent Units (GRU)



Gated Recurrent Units (GRU)

LSTM vs GRU

- GRU의 compute 속도가 더 빠르며, parameter의 수가 더 적다.
- GRU는 LSTM의 장기 의존성 문제에 대한 해결책을 유지하면서, 은닉 상태를 업데이트하는 계산을 줄였다.
→ GRU는 성능은 LSTM과 유사하면서, 동시에 LSTM의 복잡한 구조를 간단하게 만들었다.



[GRU 구조]

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\ \tilde{h}_t &= \tanh(W \cdot [r_t * h_{t-1}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned}$$

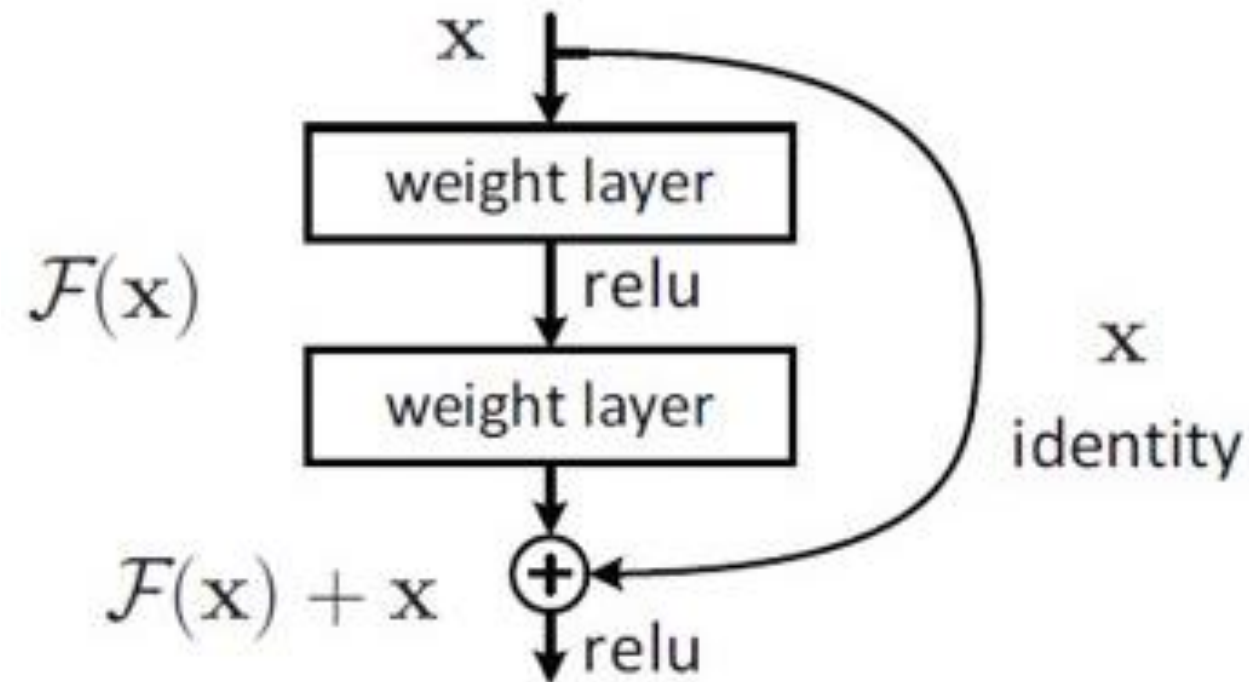
- GRU는 Reset Gate(r_t)와 Update Gate(z_t) 2개의 게이트로 이루어져있다.
- LSTM에서 forget gate와 input gate를 GRU에서는 update gate(z_t)로 합쳐 주었다.
- LSTM에서 forget gate역할이 r_t 와 z_t 둘 다에 나눠졌다고 볼 수 있다.
- 출력값(h_t)를 계산할 때 추가적인 비선형(nonlinearity) 함수를 적용하지 않는다.

Is vanishing/exploding gradient just a RNN problem?

<https://warm-uk.tistory.com/46>

RNN의 문제만으로 볼 수는 없다!

- neural architectures 전반에 걸친 문제로 보아야 한다.
- **솔루션** : 새로운 deep feedforward/convolutional 구조를 통해 direct connection을 추가하는 방법
- **Ex)** “ResNet”(Residual connections), “DenseNet”(Dense connections), “HighwayNet”(Highway connections)

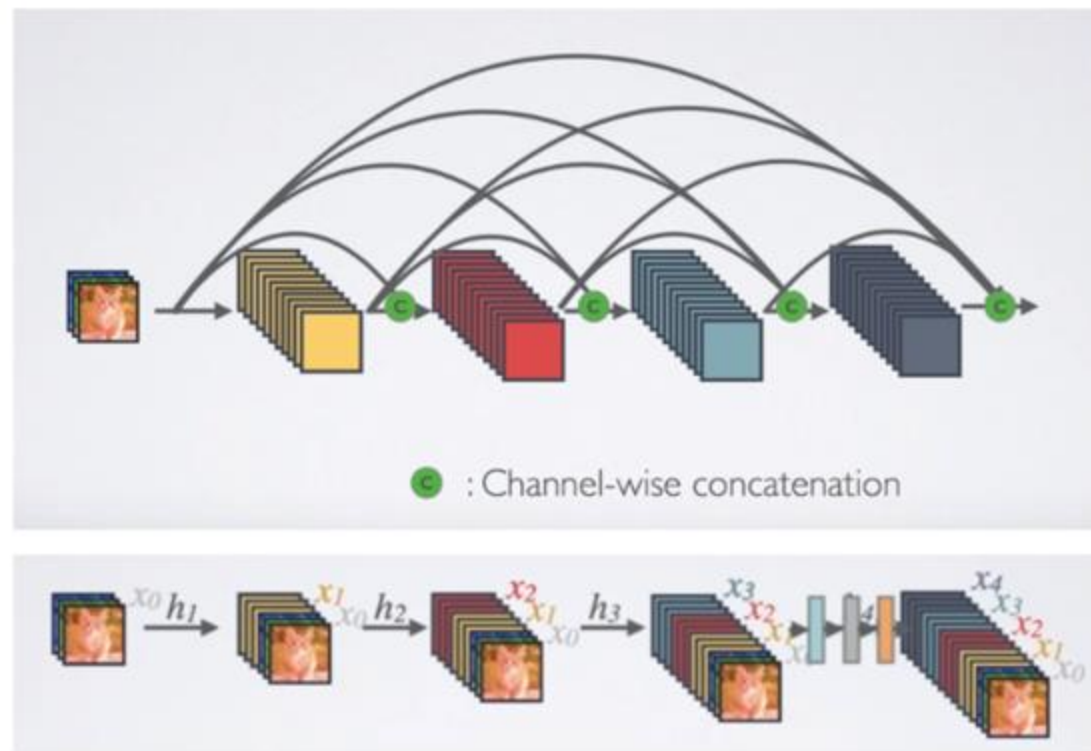


ResNet (이전 layer의 결과를 다시 이용)

- Skip connection이라는 아이디어 적용
- Skip connection은 입력에서 바로 출력으로 연결되는 것
- 기존 네트워크 : $H(x)$ 를 얻기 위해 학습. (출력 : $H(x)$)
- Resnet 네트워크 : 기존의 출력값($H(x)$)에서 입력값(x)을 뺀 차이를 얻기 위해 학습.
(출력 : $H(x) = F(x) + x$)
- 기존 네트워크 : 역전파로 가중치를 업데이트 할 때, 앞 층으로 갈수록 기울기가 0으로 소실
- Resnet 네트워크 : $f(X)$ 에 추가된 $+X$ 덕분에, 역전파 시 앞 층에 기울기 1 이 추가

Is vanishing/exploding gradient just a RNN problem?

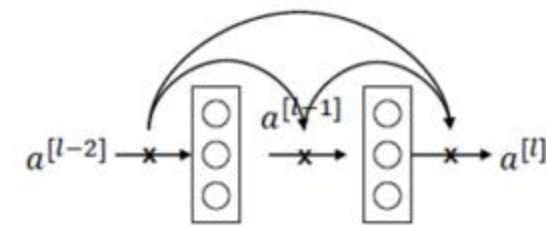
DenseNet



Dense Connectivity(DenseNet)

$$a^{[l]} = H_l([a^{[0]}, a^{[1]}, \dots, a^{[l-1]}])$$

$$L(L + 1) / 2$$



Resnet의 Skip-connection에서 발전된 Dense-connection

각 layer는 loss function과 original input signal로부터의 gradient에 직접 접근할 수 있으므로, 유사 deep supervision이 이루어진다.

이는 deeper network architecture의 학습에 도움이 된다.

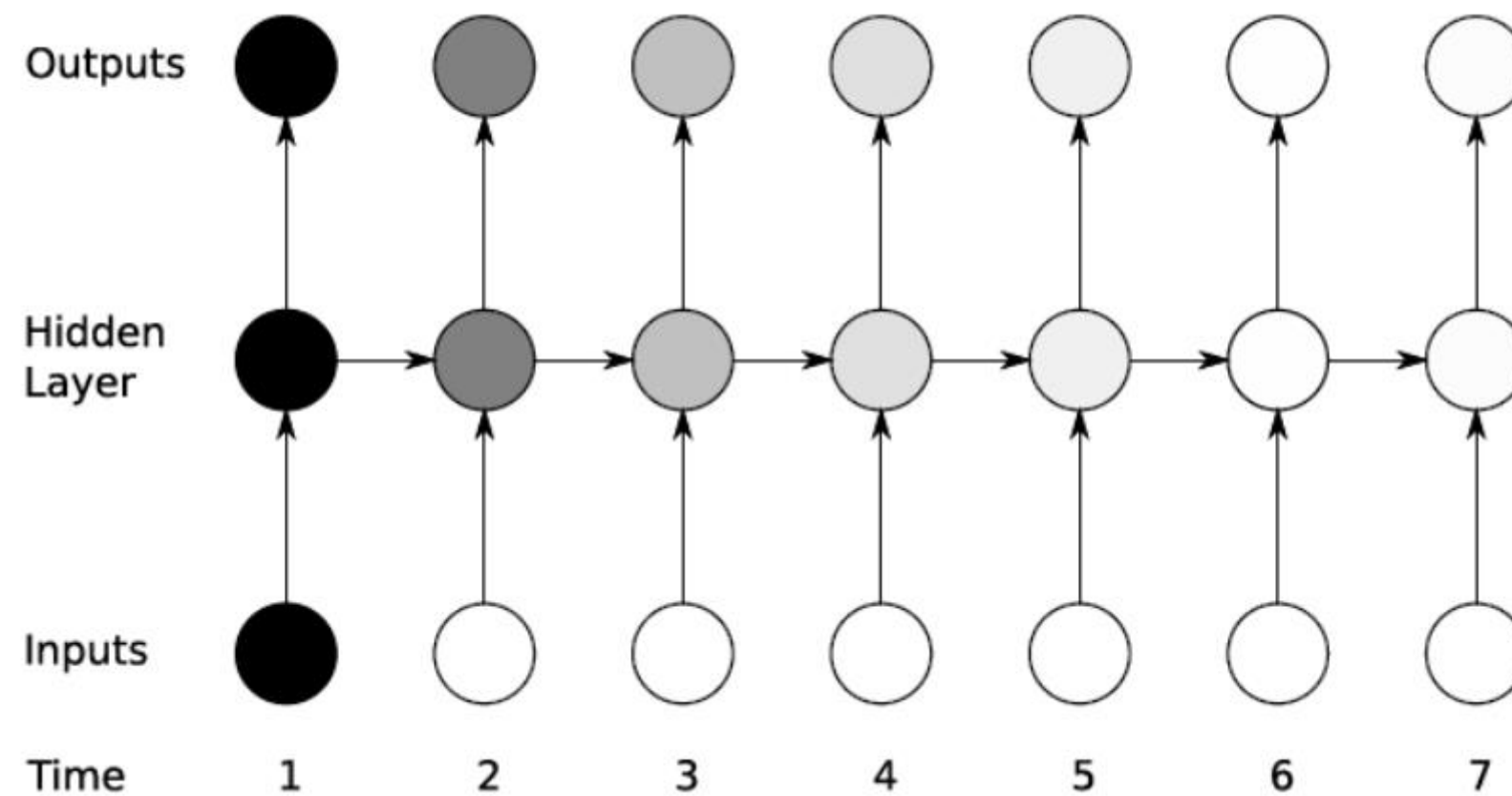
DenseNet은 extremely deep 하거나 wide한 구조로부터 representational power를 끌어내는 대신, feature의 재사용을 통해 네트워크의 잠재력을 활용함으로써, 학습하기 쉬우면서도 효율적인 parameter를 가진 압축 모델을 만든다.

→ Substantially reduce the number of parameters, Direct connection을 가능케 한다.

Is vanishing/exploding gradient just a RNN problem?

<https://it-ist.tistory.com/27>

Vanishing/exploding gradient는 복합적인 이유로 발생하지만,
같은 weight matrix로 반복적으로 연산하는 RNN이 특히나 이 문제에 대해서 안정적이지 못하다.



Fancy RNN

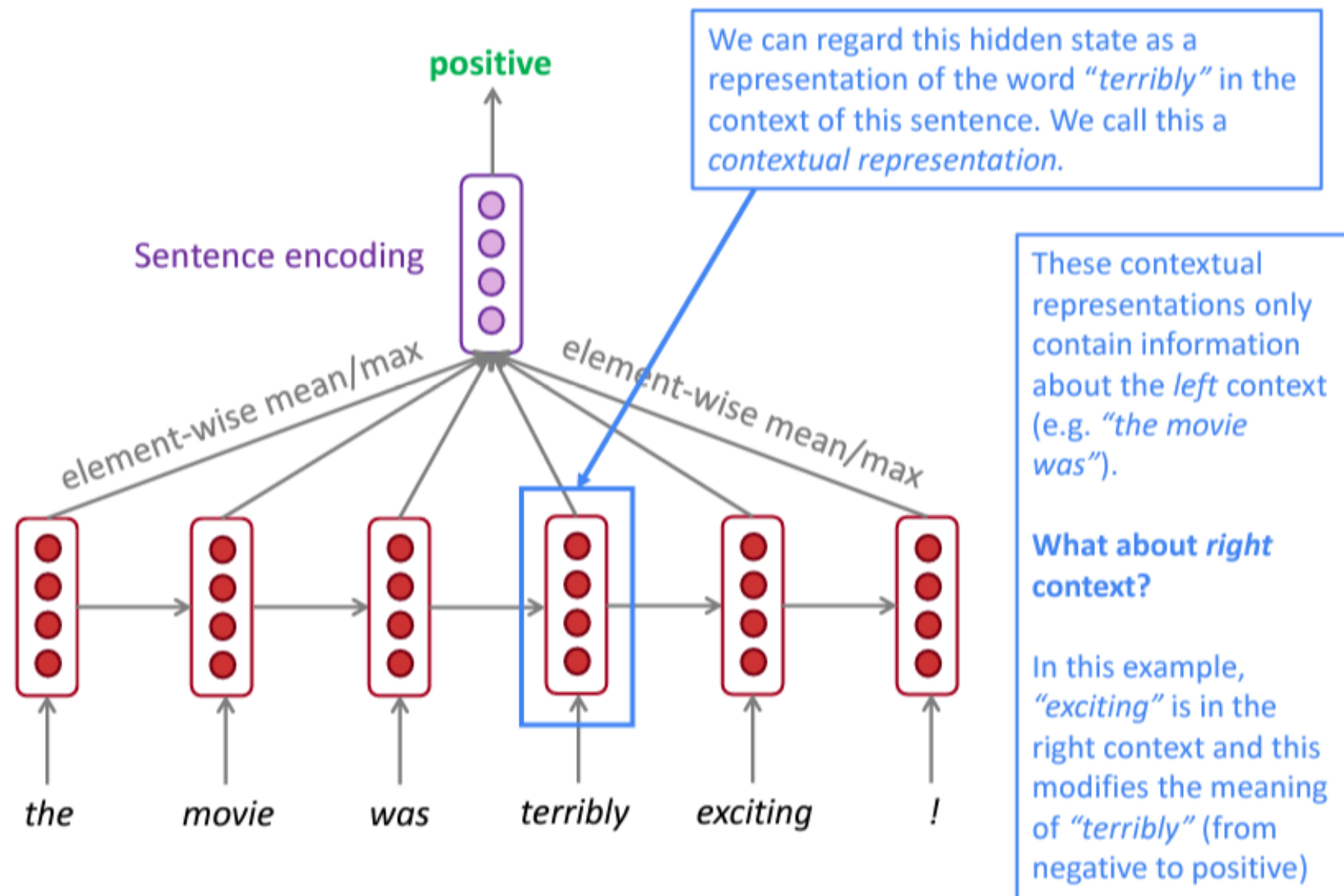


Bidirectional RNNs

Bidirectional RNNs:

과거의 상태뿐만 아니라, 미래의 상태까지 고려하는 확장된 RNN 형태

기존의 Recurrent Neural Networks(RNNs)는 일반 Neural Networks에 비교해서, 데이터의 이전 상태 정보를 “메모리(Memory)” 형태로 저장할 수 있다는 장점이 있었다.

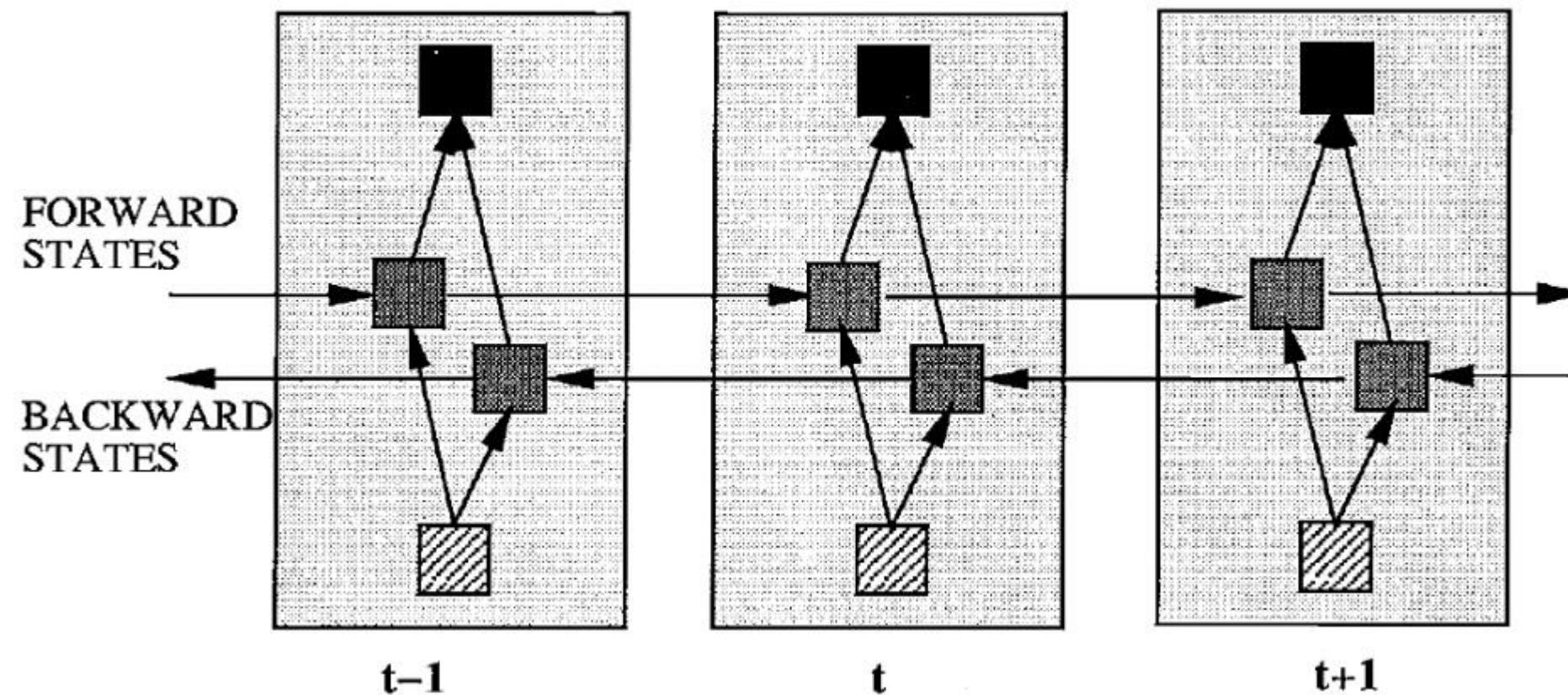


* 양방향 RNN은 하나의 출력 값을 예측하는 데 메모리 셀 두 개를 사용한다. 첫 번째 메모리 셀은 이전 시점의 은닉 상태(forward states)를 전달받아 현재의 은닉 상태를 계산한다.

Bidirectional RNNs

<http://solarisailab.com/archives/1515>

Bidirectional Recurrent Neural Networks(BRNNs)을 이용하면, 이렇게 이전 정보와 이후 정보를 모두 저장할 수 있다.



2개의 Hidden Layer

전방향 상태(Forward States) 정보를 가지고 있는 Hidden Layer와 후방향 상태(Backward States) 정보를 가지고 있는 Hidden layer가 있고, 이 둘은 서로 연결되어 있지 않다.

하지만, 입력값은 이 2가지 Hidden Layer에 모두 전달되고, Output Layer도 이 2가지 Hidden Layer로 모두 값을 받아서 최종 Output을 계산한다.

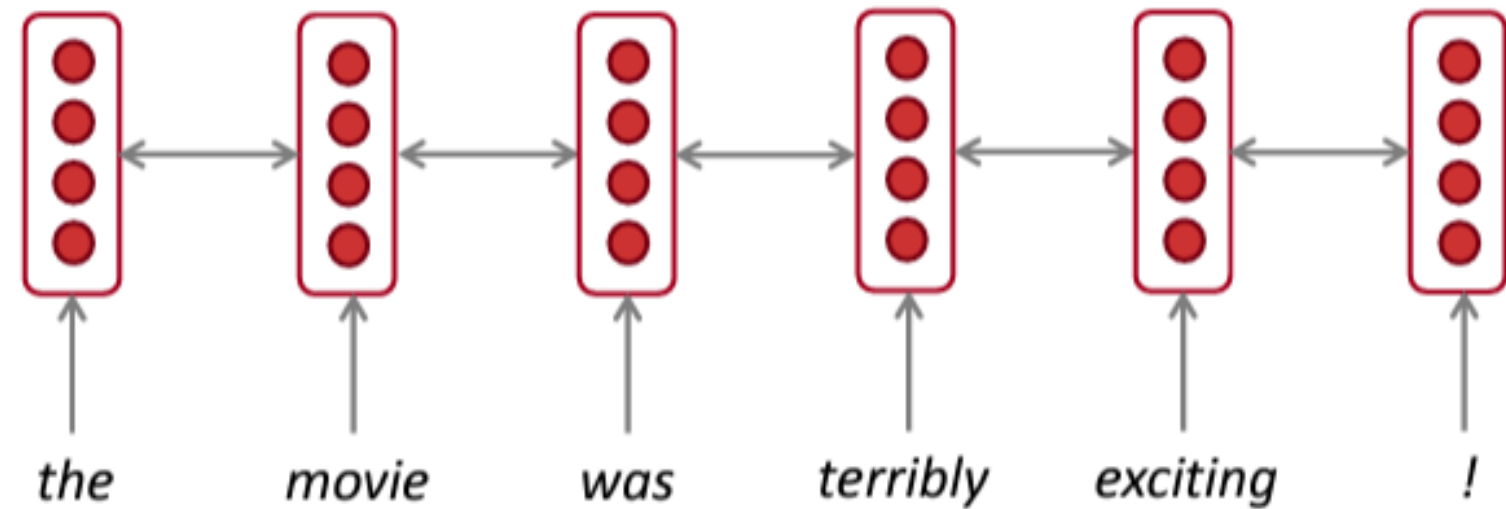
Bidirectional RNNs

최종 output 계산

$$\vec{h}_t = \sigma(W_{x\vec{h}}x_t + W_{\vec{h}\vec{h}}\vec{h}_{t-1} + b_{\vec{h}})$$

$$\overleftarrow{h}_t = \sigma(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}})$$

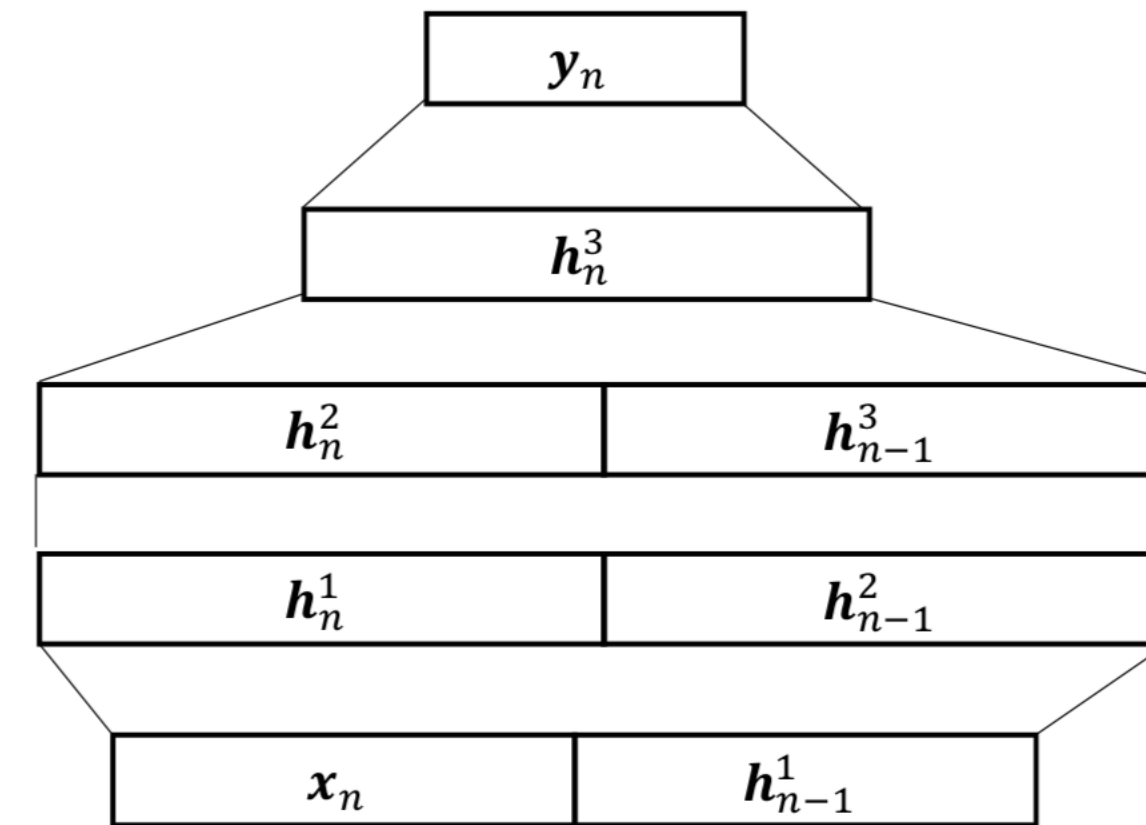
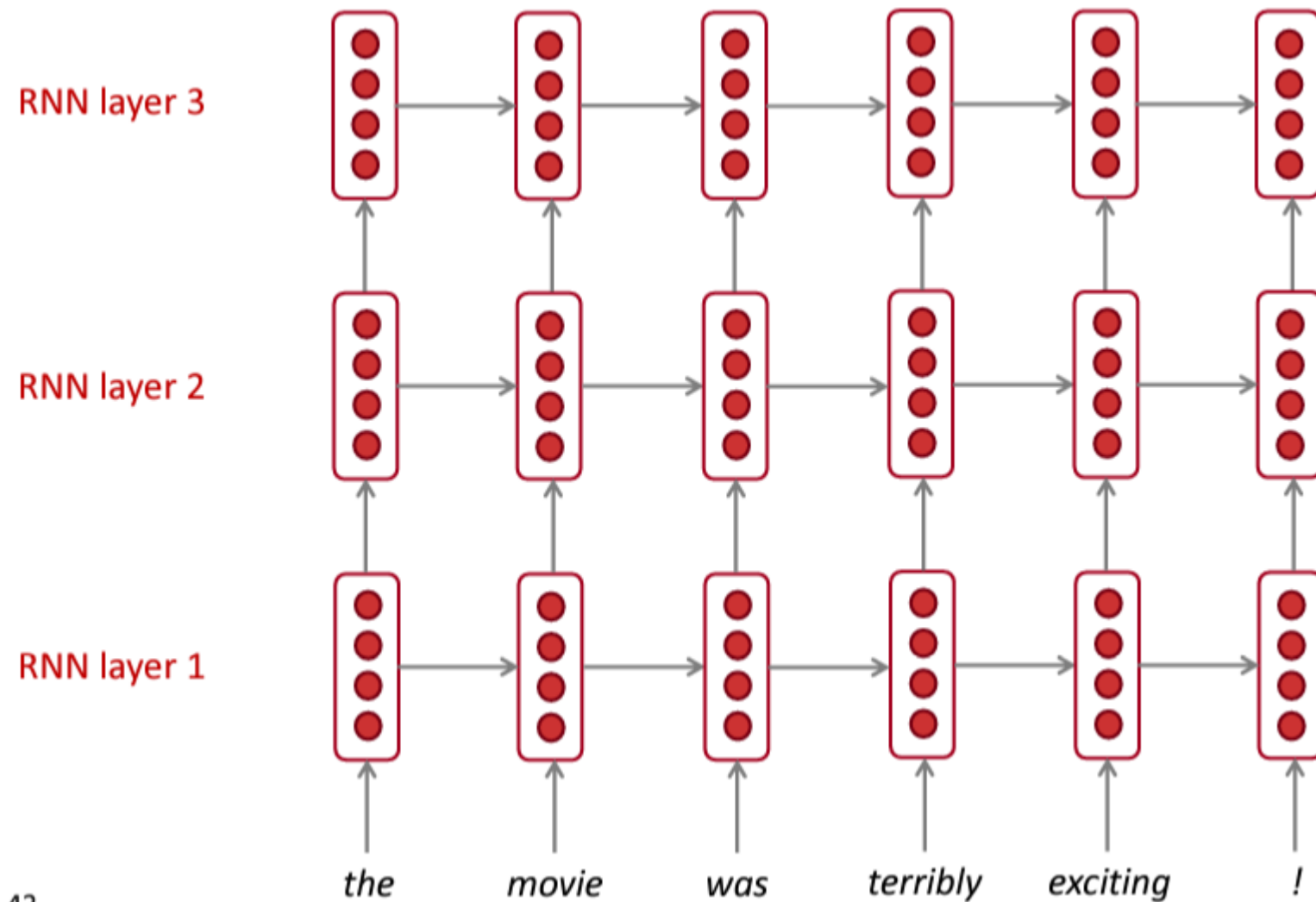
$$y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y$$



시간 t 에서 전방향(forward) hidden layer의 활성화값(activation) output \vec{h}_t ,
전방향(backward) hidden layer의 활성화값(activation) output \overleftarrow{h}_t , output layer의 output y_t

* 위 수식에서는 시그모이드 함수를 사용했지만 vanilla, LSTM, GRU computation도 쓰이기도 한다.

Multilayer RNNs



n 번째 Time-Step에서의 결과

RNN도 심층 신경망처럼 쌓아 올릴 수 있지만,
신경망의 구조가 매우 복잡해지고 학습이 잘 되지 않을 수도 있다.
따라서 skip-connection이나 dense-connection가 필요한 경우가 있을 수 있다.

THANK YOU

