TRAIN AND TEST

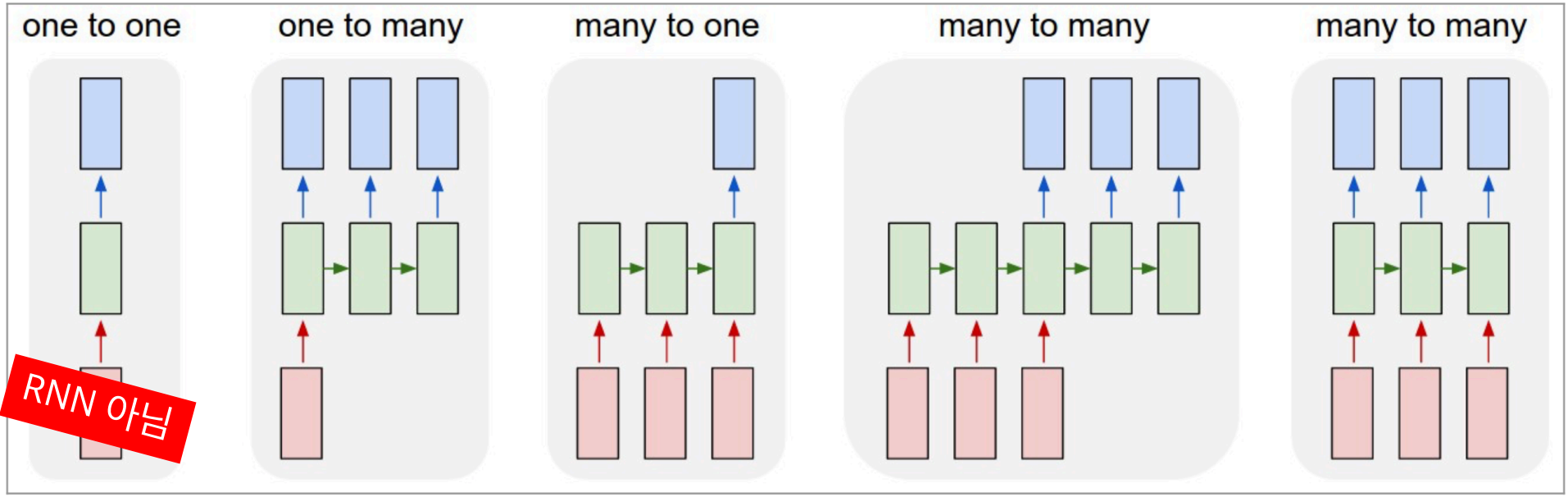
[CS224N] Lecture 7 - Vanishing Gradients, Fancy RNNs

INDEX

0. Review of RNN

1 RNN의 종류

: 입력과 출력에 따라 분류한다



순환 부재 | 이미지에 대한 설명 | 감성 분석 | 영어->한글 번역 | 다음에 나올 단어 (LM)

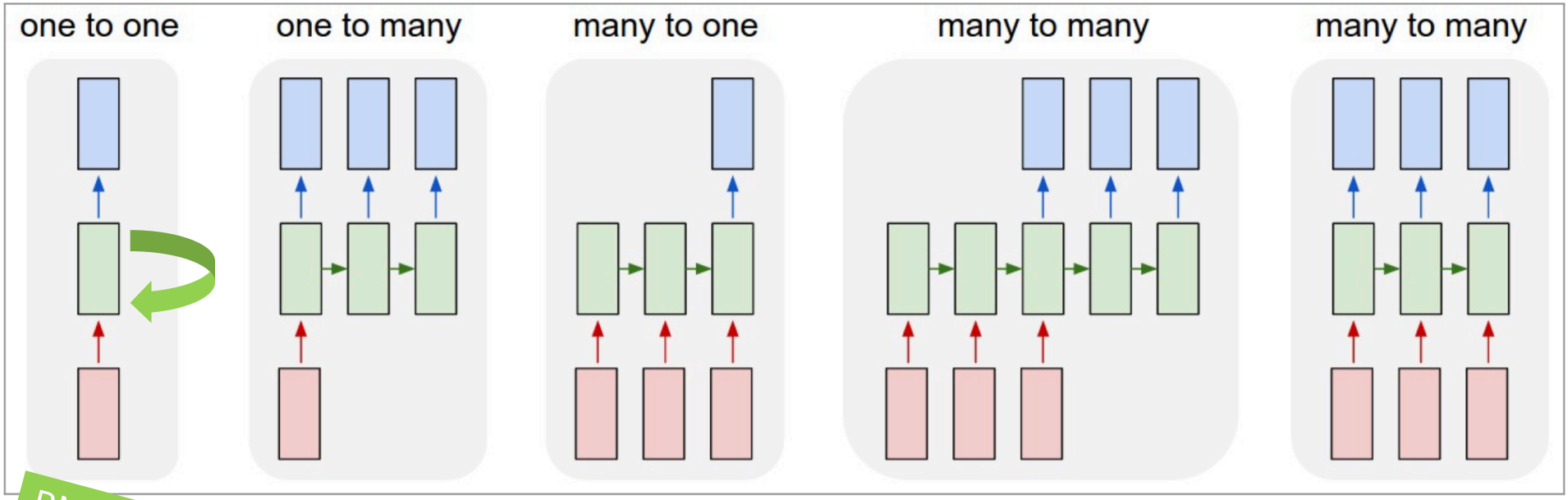
> 히든 노드가 방향을 가지고 순환구조(directed cycle)를 가진 인공 신경망

[h_{t-1}와 h_t의 방향]

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

1 RNN의 종류

: 입력과 출력에 따라 분류한다



RNN 맞음

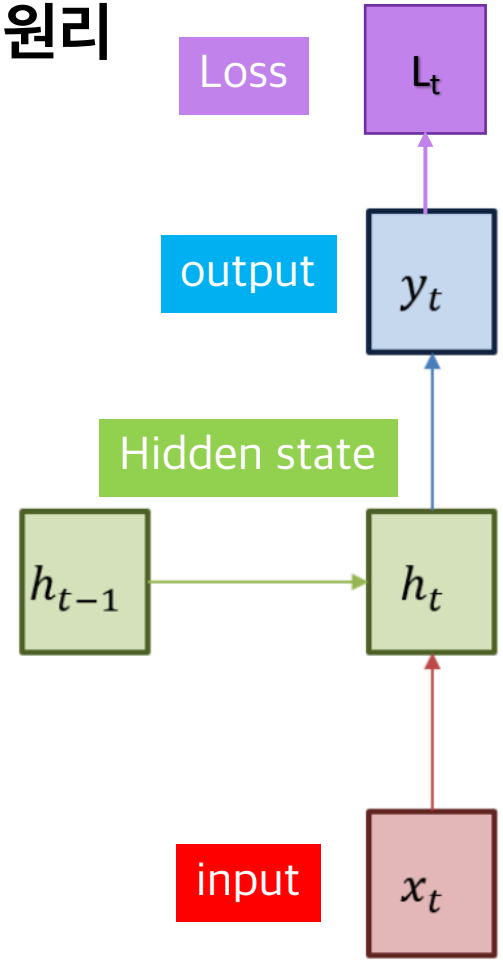
이미지에 대한 설명 | 감성 분석 | 영어->한글 번역 | 다음에 나올 단어 (LM)

> 히든 노드가 방향을 가지고 순환구조(directed cycle)를 가진 인공 신경망

[h_{t-1}와 h_t의 방향]

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(0) 전체적인 원리



$L_t = \text{Loss}(\hat{y}_t, y_t)$

$$\hat{y}_t = softmax(W_{hy} h_t + b_y)$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

이전 hidden state 정보 (h_{t-1})와

현재 input 정보(x_t)를 이용하여

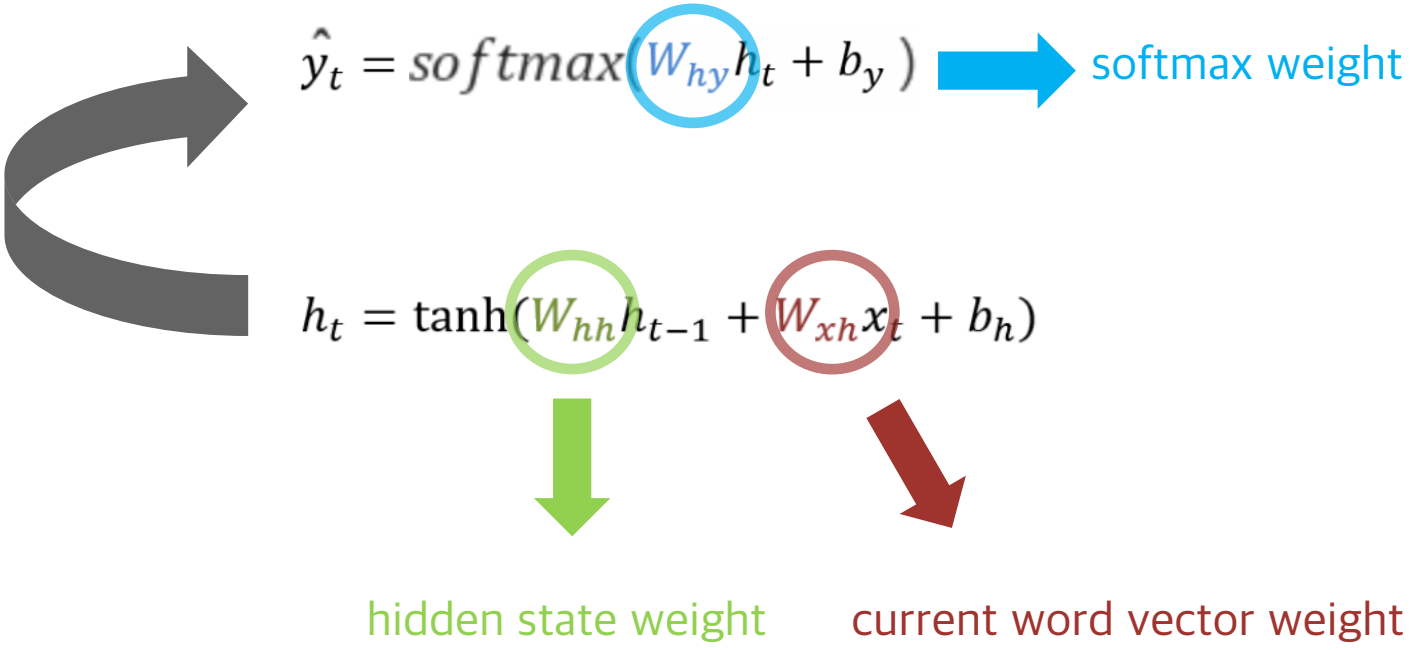
Output(y_t)을 출력

https://gnoej671.tistory.com/6

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

2 RNN의 작동 원리

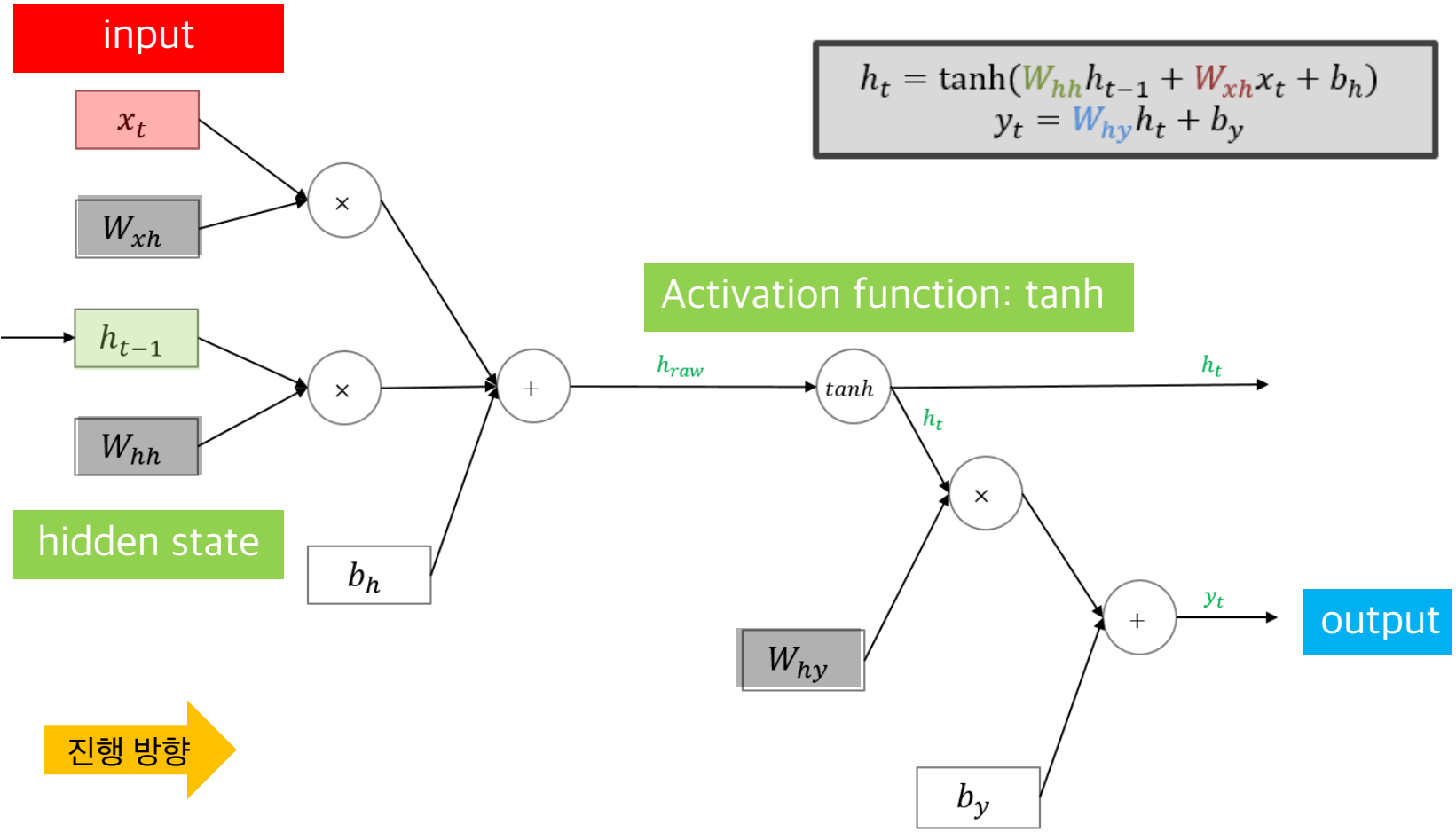
(1) 학습 진행 방식 : 업데이트해야하는 weight는…?



RNN의 특징: 이 weight들은 각 단계마다 공유된다.

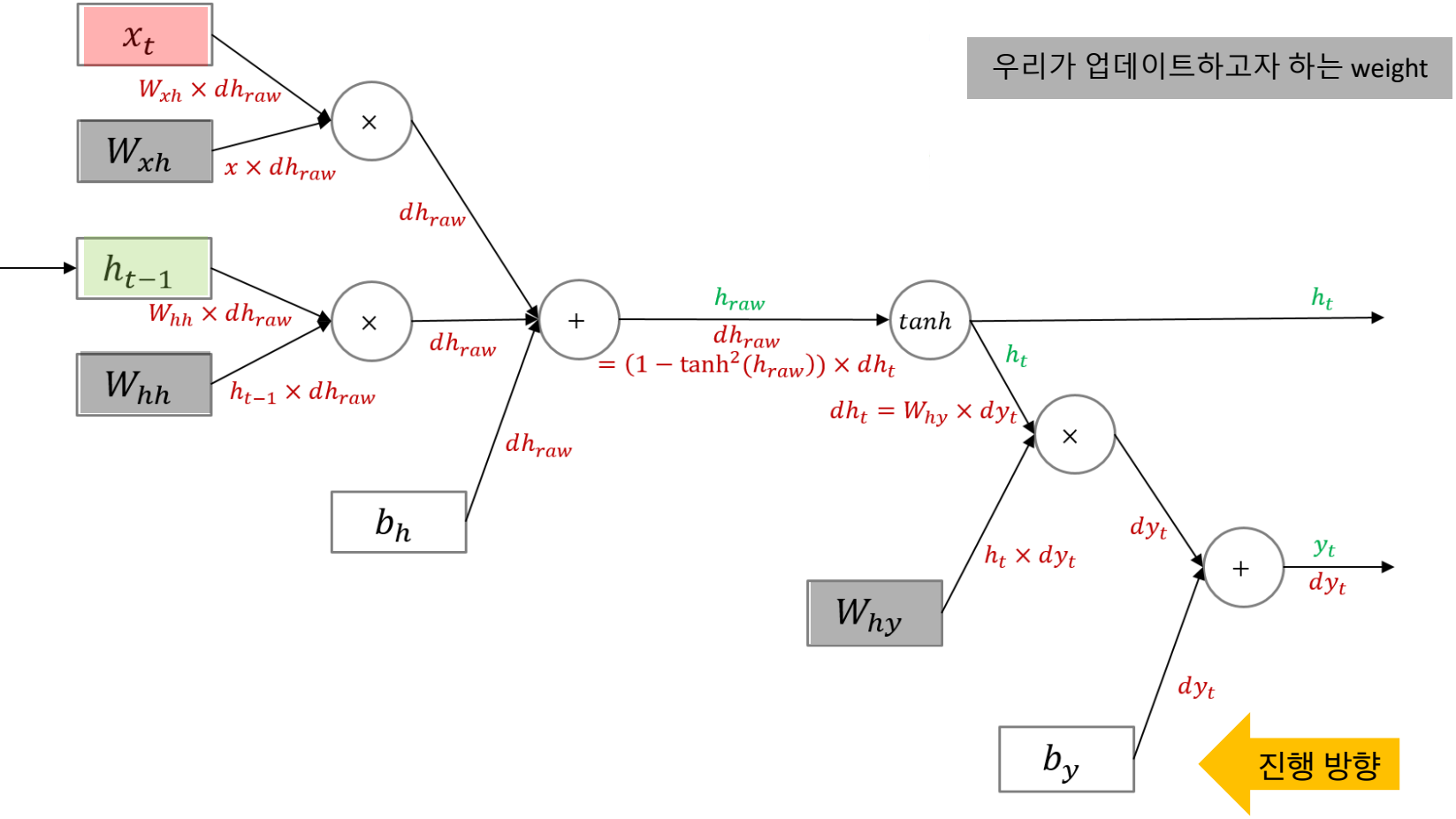
https://gnoej671.tistory.com/6
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Forward Propagation [순전파]



https://gnoej671.tistory.com/6
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

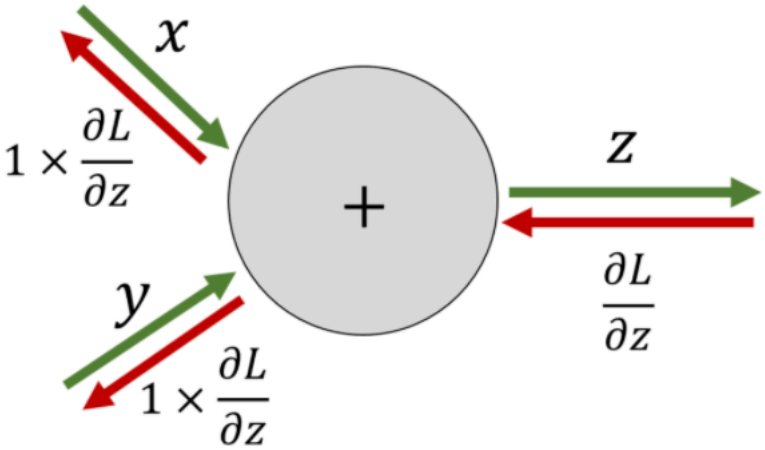
(1) 학습 진행 방식 : Backward Propagation [역전파]



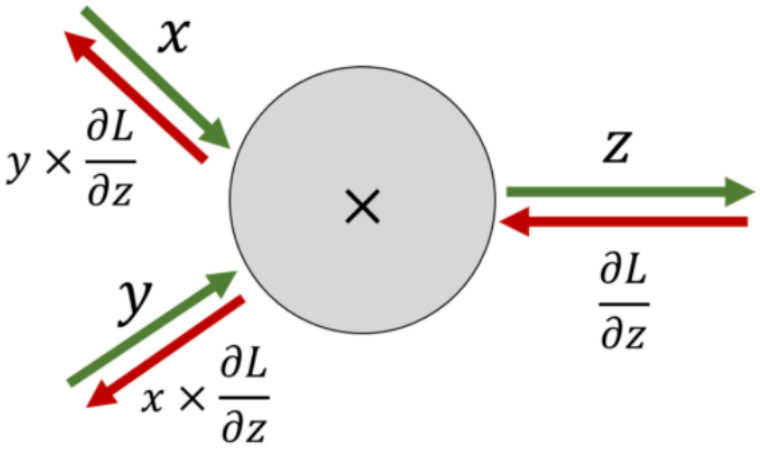
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]

기본적인 원리



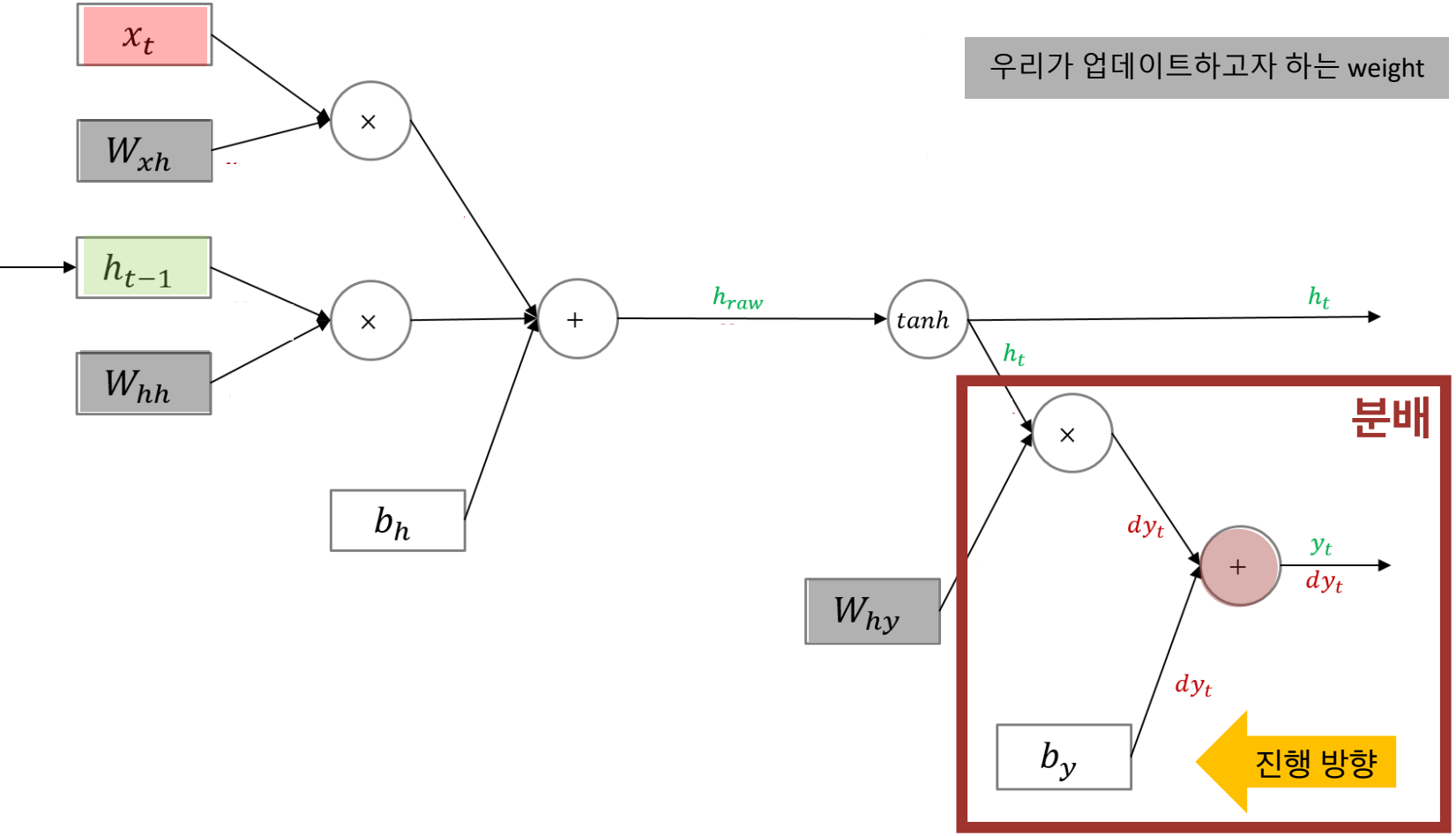
덧셈 노드



곱셈 노드

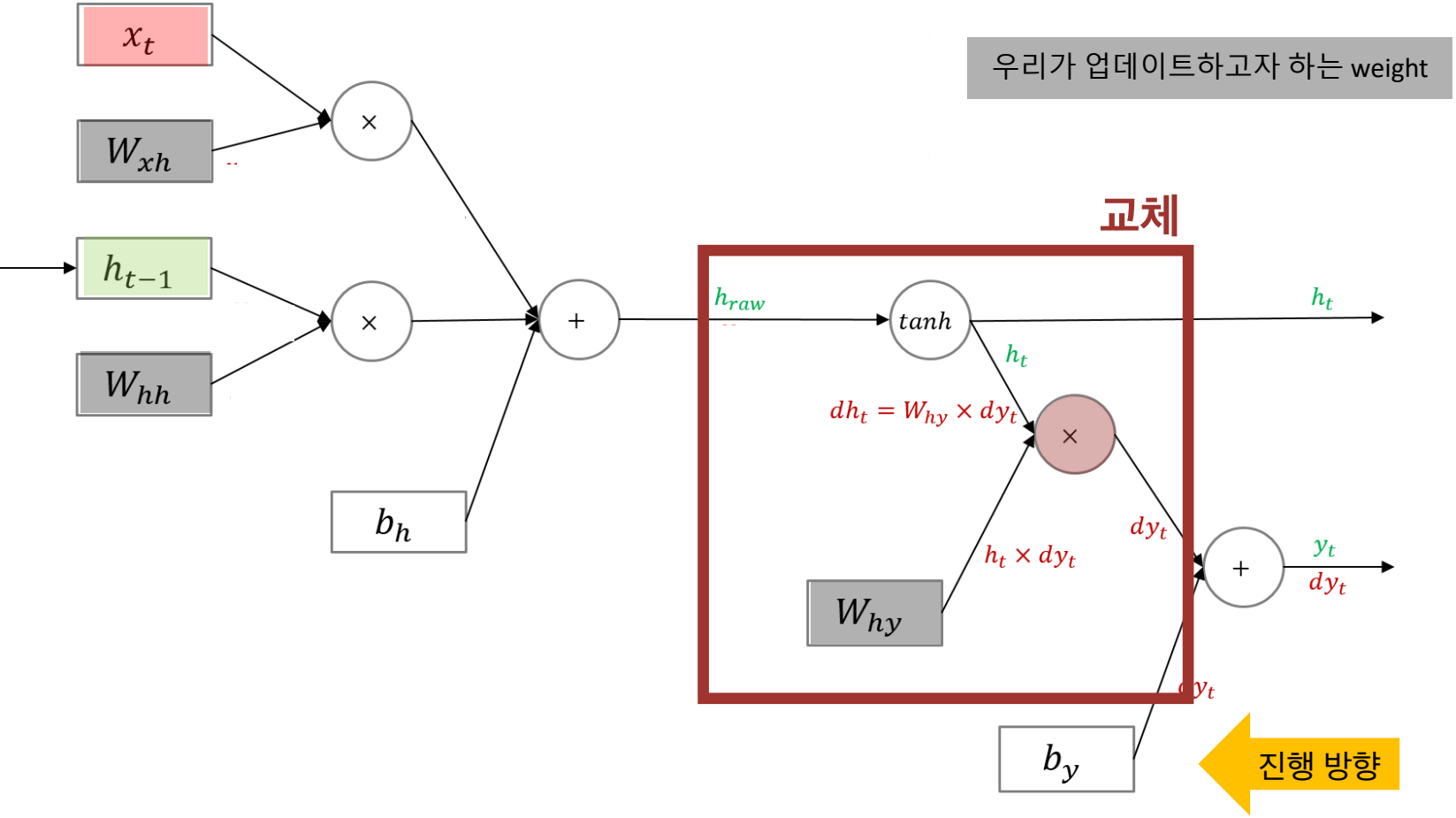
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]



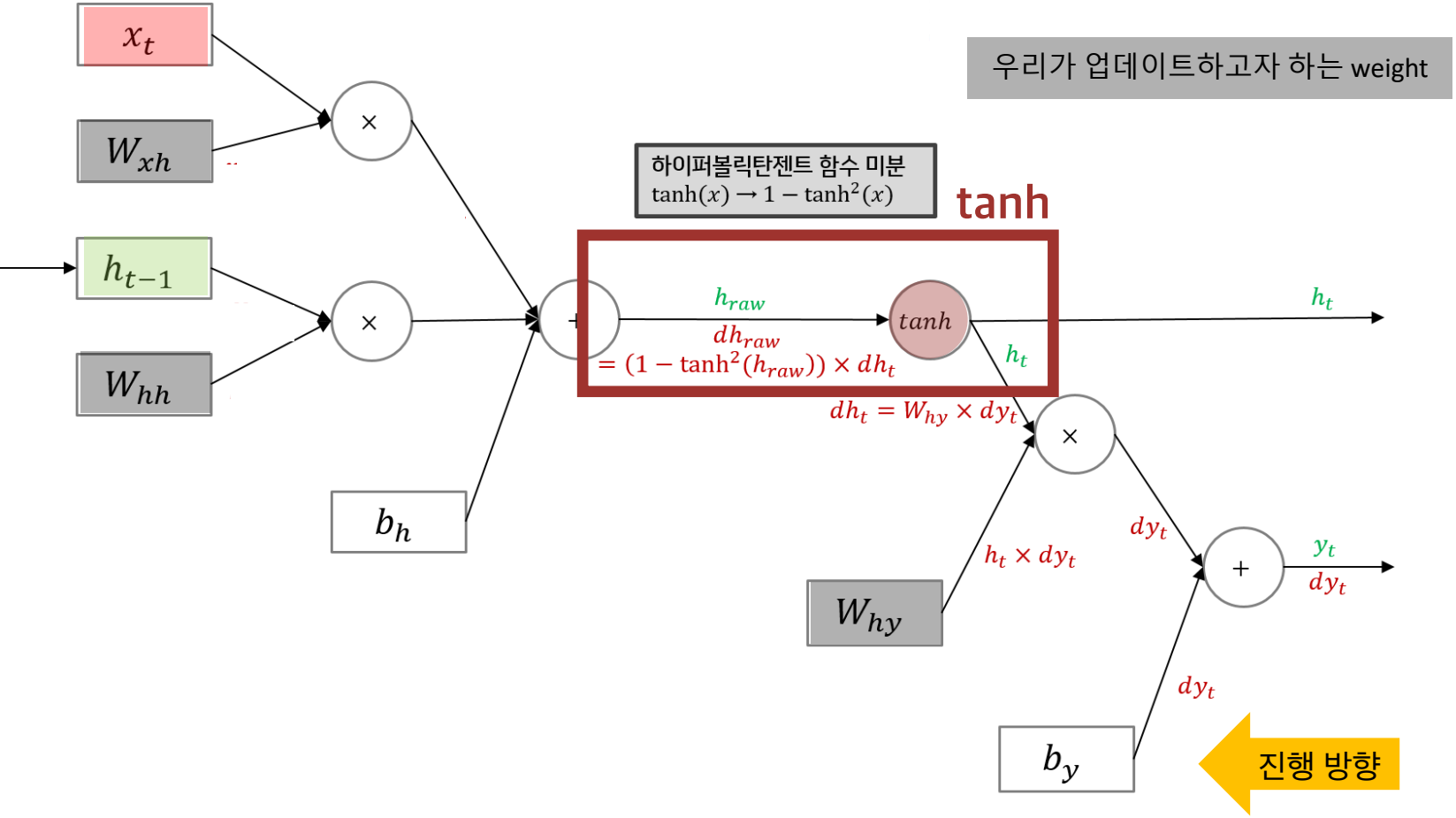
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]



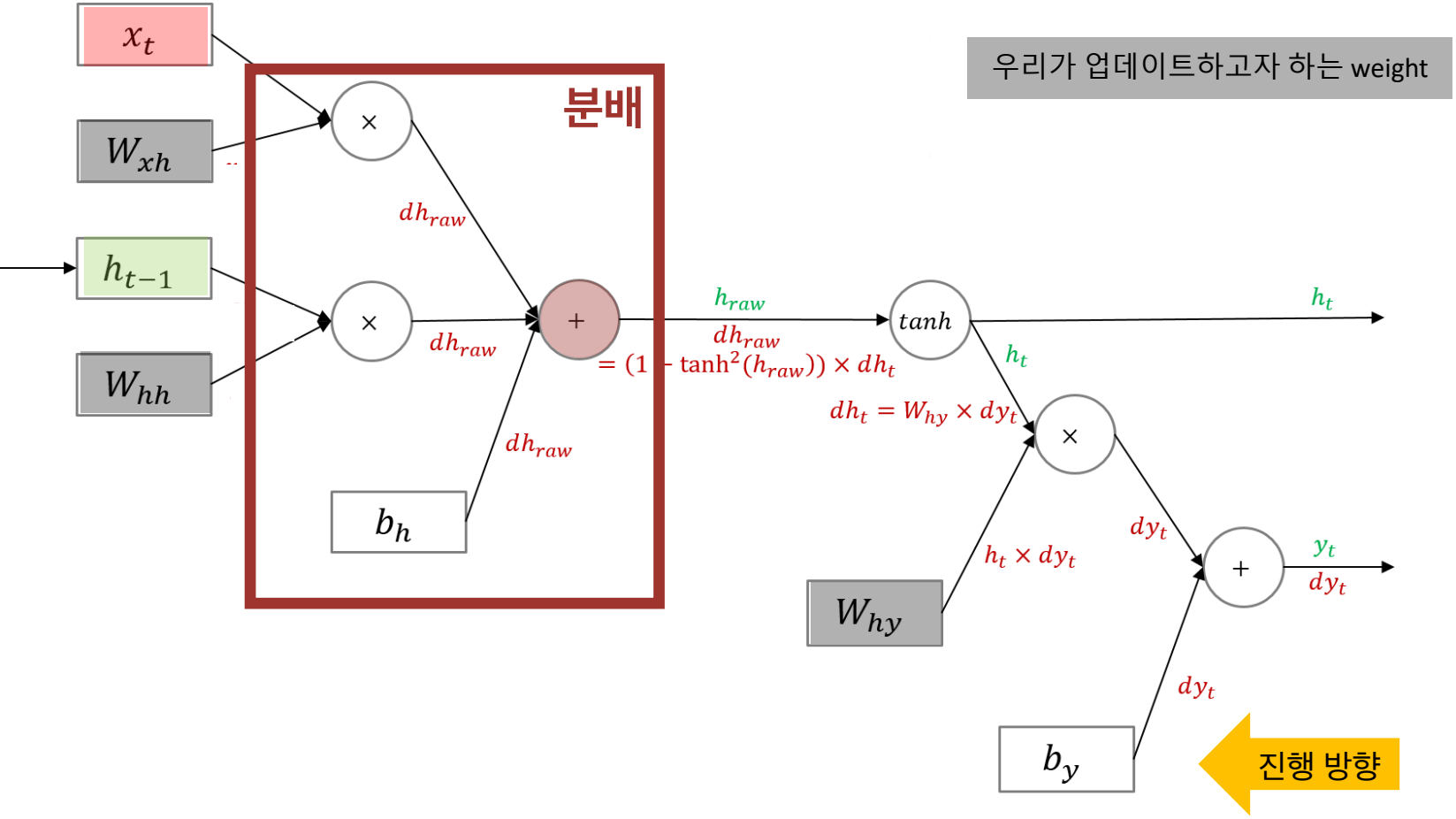
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]



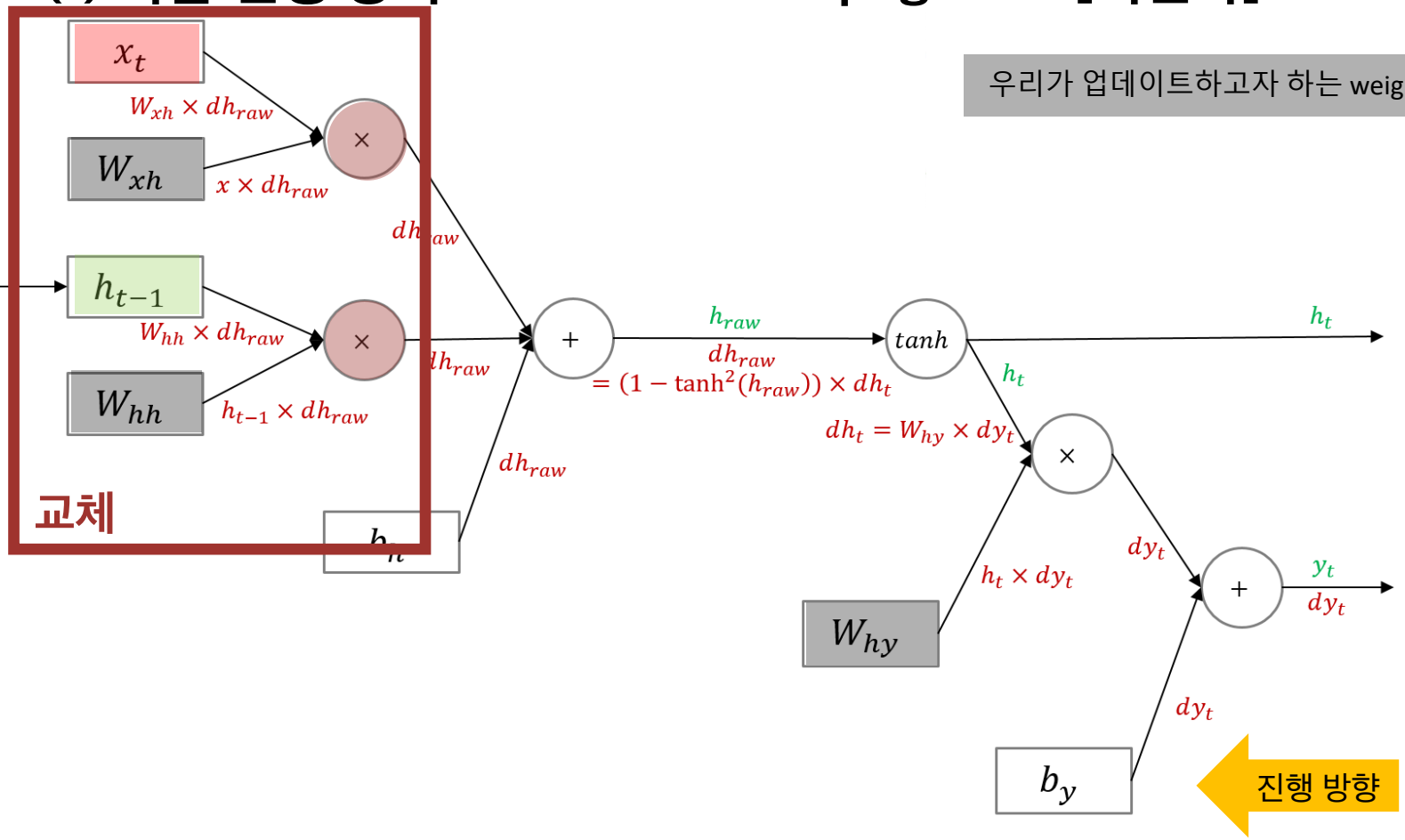
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]



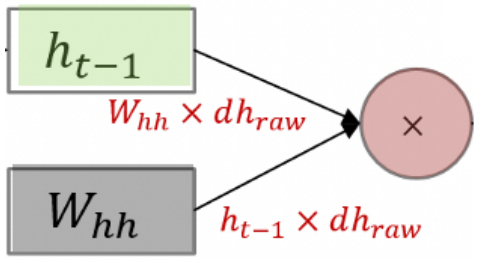
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

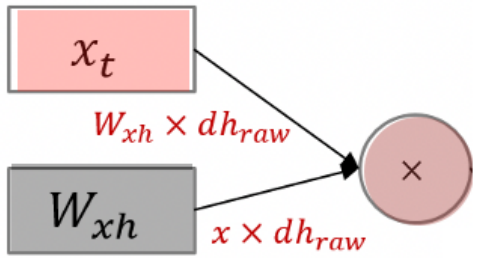
(1) 학습 진행 방식 : Backward Propagation [역전파]

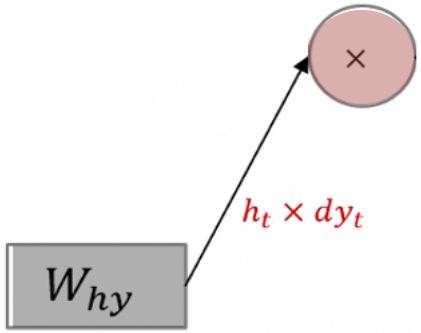


출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(1) 학습 진행 방식 : Backward Propagation [역전파]







$dW_{hh} = h_{t-1} * dh_{raw}$

$dW_{xh} = x * dh_{raw}$

$dW_{hy} = h_t * dy_t$

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

2 RNN의 작동 원리

(1) 학습 진행 방식 : Backward Propagation [역전파]

$$W_{xh}' = W_{xh} - dW_{xh}$$

$$W_{hh}' = W_{hh} - dW_{hh}$$

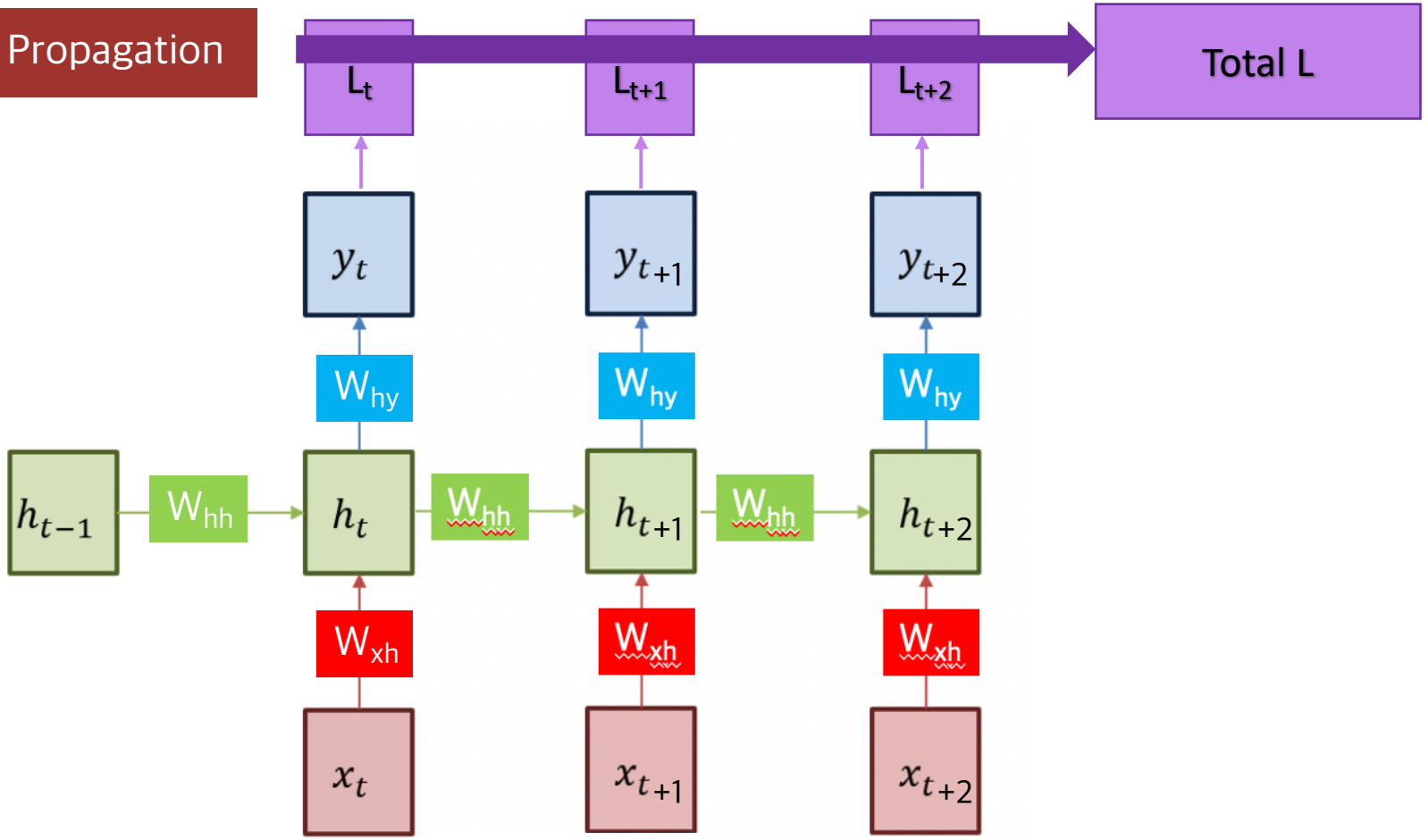
$$W_{hy}' = W_{hy} - dW_{hy}$$

목표: 각 weight의 업데이트를 진행하며 전체 Loss를 줄인다.

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

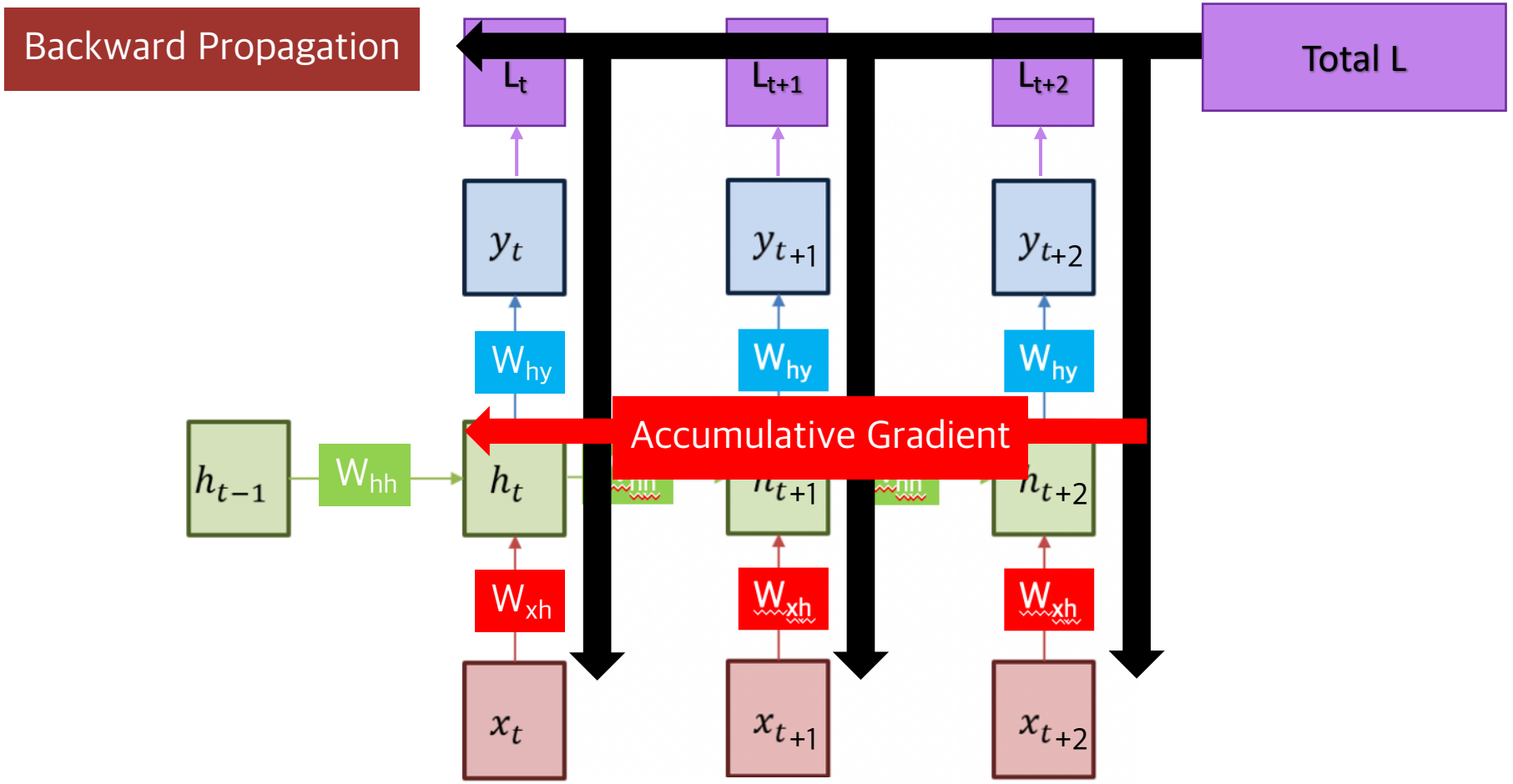
(2) Backward Propagation through TIME

Forward Propagation



출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(2) Backward Propagation through TIME



출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(2) Backward Propagation through TIME

Backward Propagation :공유되는 weight의 동일한 업데이트를 위해 특이한 방식으로 진행
+ 앞선 weight를 고려하여 순서대로 y_t를 구하기 때문

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}$$

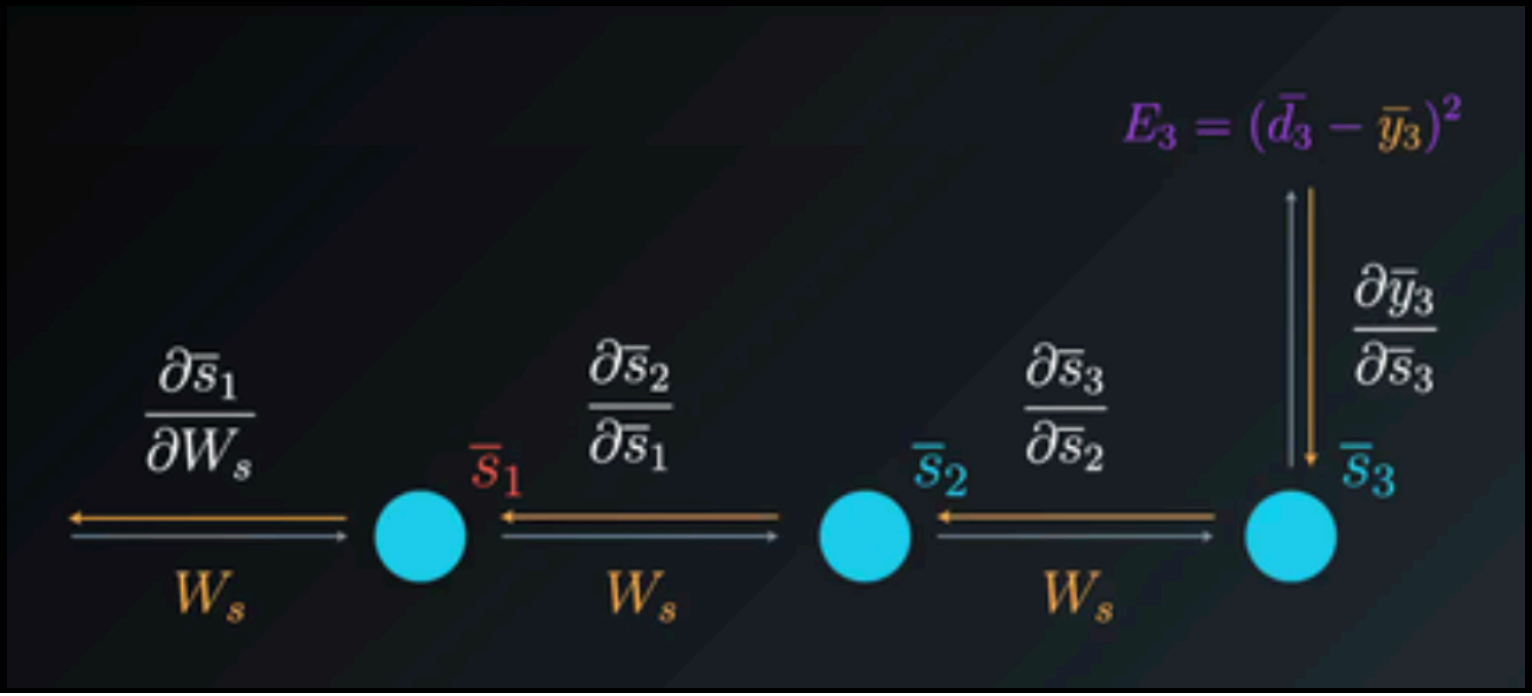
(ex) t=3인 경우, V, W, U를 업데이트하고자 할 때 :

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vs_t)$$

① $$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V}$$

② $$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/



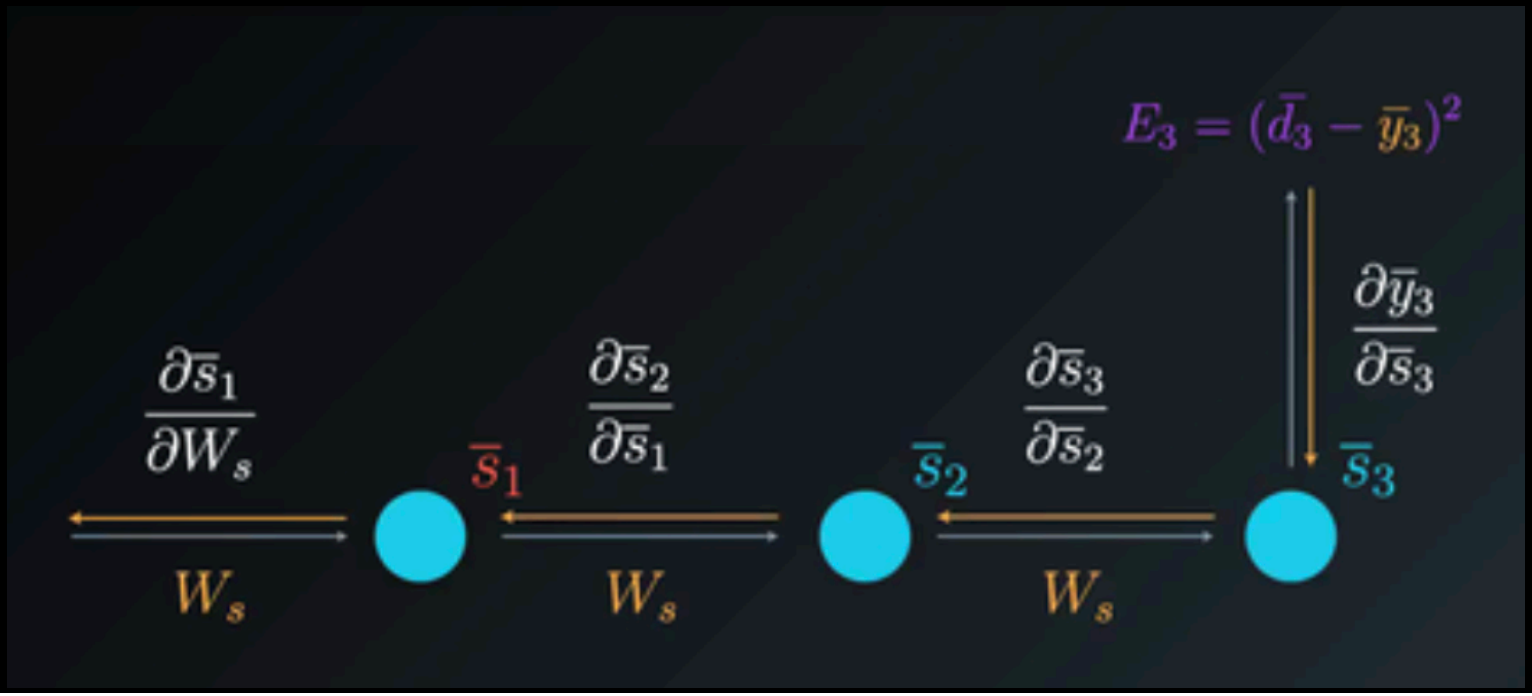
(ex) t=3인 경우, V, W, U를 업데이트하고자 할 때 :

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vs_t)$$

①
$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V}$$

②
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/



(ex) t=3인 경우, V, W, U를 업데이트하고자 할 때 :

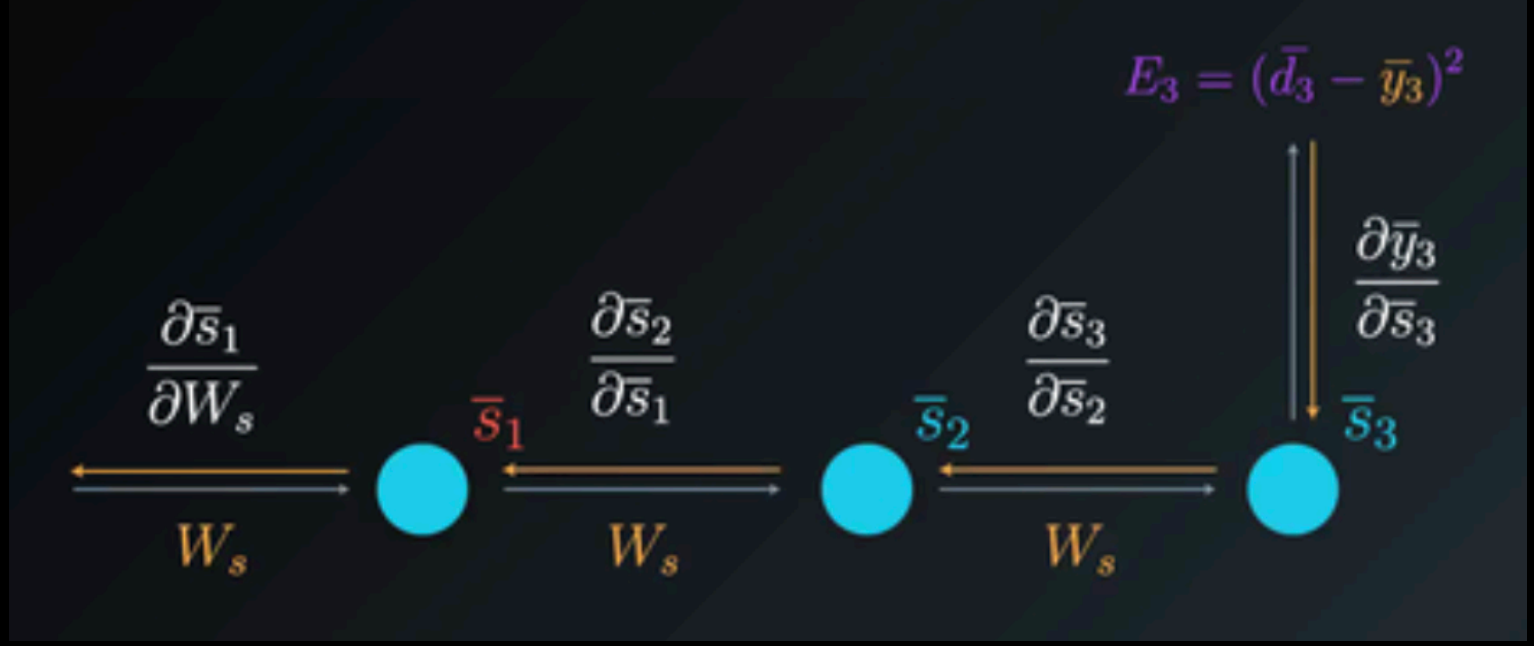
$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vs_t)$$

① $\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V}$

② $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial s_3}\frac{\partial s_3}{\partial W}$

Accumulative Backpropagation 을 진행해야 함

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/



② $$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_k} \frac{\partial s_k}{\partial W} =$$

$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial \bar{s}_1} \cdot \frac{\partial \bar{s}_1}{\partial W_s}$$

Considering $\bar{S}_3$ Considering $\bar{S}_2$ Considering $\bar{S}_1$

③ U 도 ②와 동일한 방식으로 gradient를 구한다.

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vs_t)$$

각 단계의 모든 gradient을 고려하여 update를 진행한다!!

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

1. Problems of RNNs

RNN이 복잡한 이유

(1) Original Backpropagation

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}$$

We will diverge from the classical BPTT equations at this point and re-write the gradients in order to better highlight the exploding gradients problem:

*R. Pascanu, "On the difficulty of training recurrent neural networks"

(2) Backpropagation through TIME

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

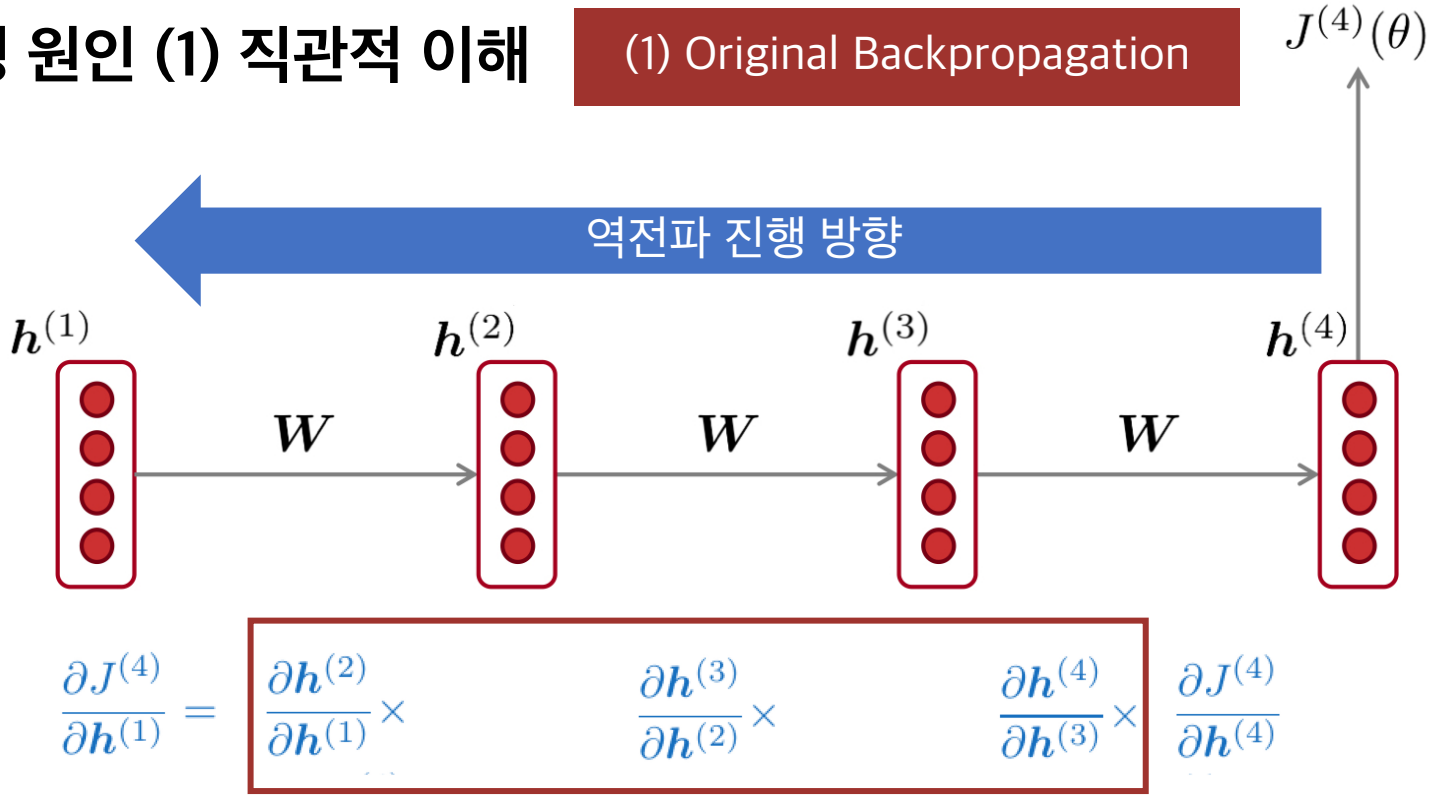
1 RNN의 문제점

(1) Vanishing Gradient Problem

(2) Exploding Gradient Problem

- **발생 원인 (1) 직관적 이해**

(1) Original Backpropagation

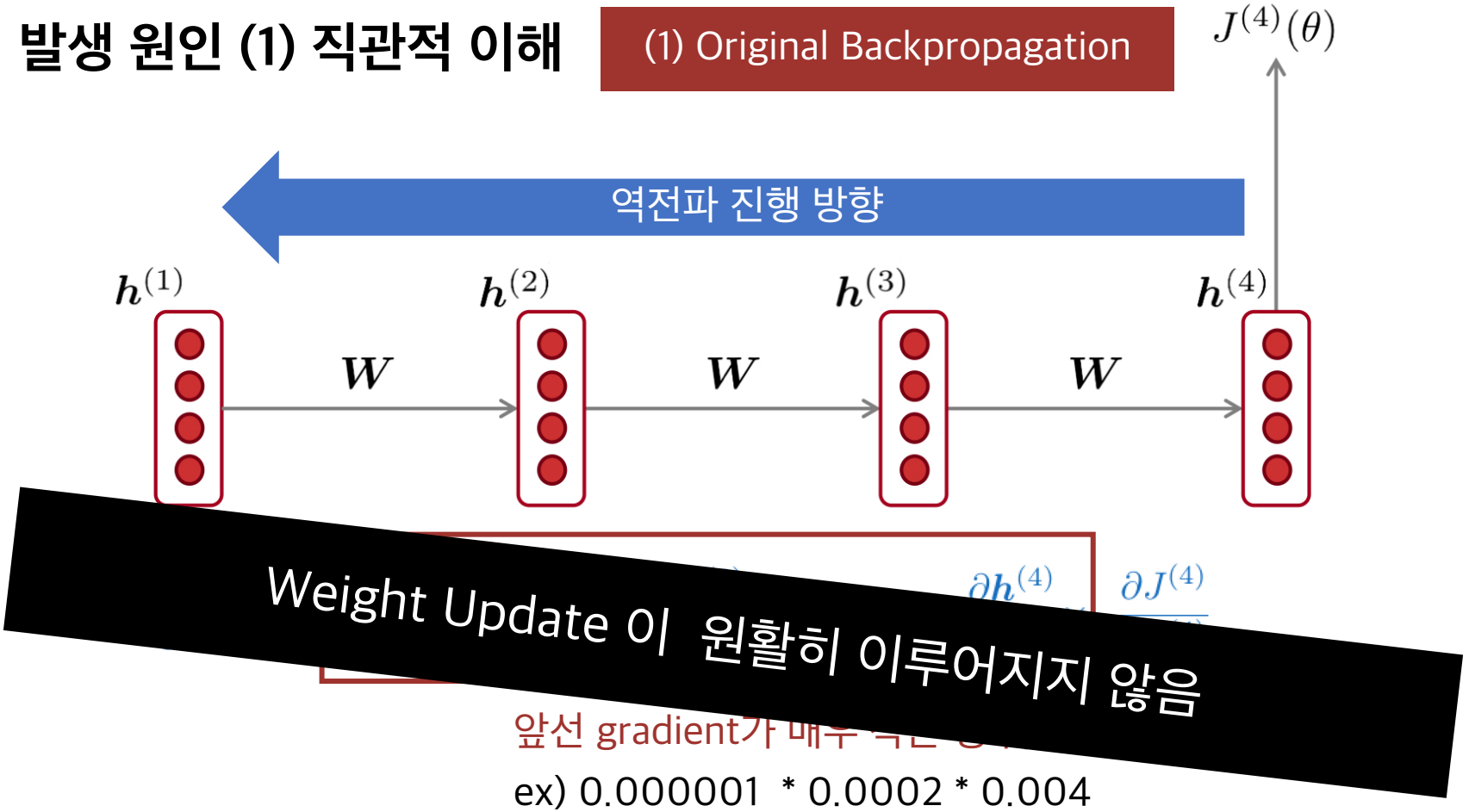


앞선 gradient가 매우 작은 경우, 역전파를 진행할수록 … ?

ex) 0.000001 * 0.0002 * 0.004

- **발생 원인 (1) 직관적 이해**

(1) Original Backpropagation



RNN의 근본적인 원인

- **발생 원인 (2) 수식적 이해**

(1) Original Backpropagation

(1) Forward Propagation

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

(2) Calculating Gradients

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

(3) Gradient of step i with the previous step j

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

- **발생 원인 (2) 수식적 이해**

(1) Original Backpropagation

(3) Gradient of step i with the previous step j

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

(ex) i=3, j=1 일때 : 3 번째 loss와 hidden state의 1번째 step

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

CHAIN RULE

- **발생 원인 (2) 수식적 이해**

(1) Original Backpropagation

i 번째 loss와 hidden state 의 j번째 step

(3) Gradient of step i with the previous step j

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

(4) Rephrase

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

*W_h 값이 작을 때, i,j가 멀어질수록 기하급수적으로 작아진다.

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

2 RNN의 근본적인 원인

- **발생 원인 (2) 수식적 이해**

(1) Original Backpropagation

(4) Rephrase

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

(4+) 수학적 증명 : L2 norm

$$\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\| \leq \left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}}\right\| \|\boldsymbol{W}_h\|^{(i-j)} \prod_{j<t\leq i} \left\|\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\right\|$$

W_h 의 가장 큰 eigenvalue과 1의 비교 (시그모이드 함수를 사용한 경우)

1보다 작은 경우 → Vanishing Gradient

1보다 큰 경우 → Exploding Gradient

RNN의 근본적인 원인

- **발생 원인 (2) 수식적 이해**

(1) Original Backpropagation

(4) Rephrase

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

(4+) 수학적 증명 : L2 norm

$$\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\| \leq \left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}}\right\| \|\boldsymbol{W}_h\|^{(i-j)} \prod_{j<t\leq i} \left\|\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\right\|$$

수알못의 포기 포인트… 😵
왜 갑자기 Norm을 구하고…
Eigenvalue와 비교하는 것일까… ?

Vanishing Gradient

Exploding Gradient

- **발생 원인 (2) 수식적 이해** (1) Original Backpropagation

Eigenvalue 와의 비교

*R. Pascanu, "On the difficulty of training recurrent neural networks"

We generalize this result for nonlinear functions σ where $|\sigma'(x)|$ is bounded, $\|diag(\sigma'(\mathbf{x}_k))\| \leq \gamma \in \mathcal{R}$, by relying on singular values.

We first **prove** that it is *sufficient* for $\lambda_1 < \frac{1}{\gamma}$, where λ_1 is the largest singular value of $\mathbf{W}_{rec}$, for the *vanishing gradient* problem to occur. Note that we assume the parametrization given by eq. (2). The Jacobian matrix $\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k}$ is given by $\mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_k))$. The 2-norm of this Jacobian is bounded by the product of the norms of the two matrices (see eq. (6)). Due to our assumption, this implies that it is smaller than 1.

http://proceedings.mlr.press/v28/pascanu13.pdf

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \quad (2)$$

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

$$\forall k, \left\| \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\| \leq \left\| \mathbf{W}_{rec}^T \right\| \left\| diag(\sigma'(\mathbf{x}_k)) \right\| < \frac{1}{\gamma}\gamma < 1 \quad (6)$$

ing gradient problem to occur. Note that we assume the parametrization given by eq. (2). The Jacobian matrix $\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k}$ is given by $\mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_k))$. The 2-norm of this Jacobian is bounded by the product of the norms of the two matrices (see eq. (6)). Due to our assumption, this implies that it is smaller than 1.

http://proceedings.mlr.press/v28/pascanu13.pdf

- **발생 원인 (2) 수식적 이해** (1) Original Backpropagation

Eigenvalue 와의 비교

*R. Pascanu, "On the difficulty of training recurrent neural networks"

Let $\eta \in \mathrm{R}$ be such that $\forall k, \left\| \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\| \leq \eta < 1$. The existence of η is given by eq. (6). By induction over i, we can show that

$$\left\| \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \left(\prod_{i=k}^{t-1} \frac{\partial \mathbf{x}_{i+1}}{\partial \mathbf{x}_i} \right) \right\| \leq \eta^{t-k} \left\| \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \right\| \qquad (7)$$

As $\eta < 1$, it follows that, according to eq. (7), long term contributions (for which $t - k$ is large) go to 0 exponentially fast with $t - k$. □

http://proceedings.mlr.press/v28/pascanu13.pdf

2 RNN의 근본적인 원인

- **발생 원인 (2) 수식적 이해** (1) Original Backpropagation

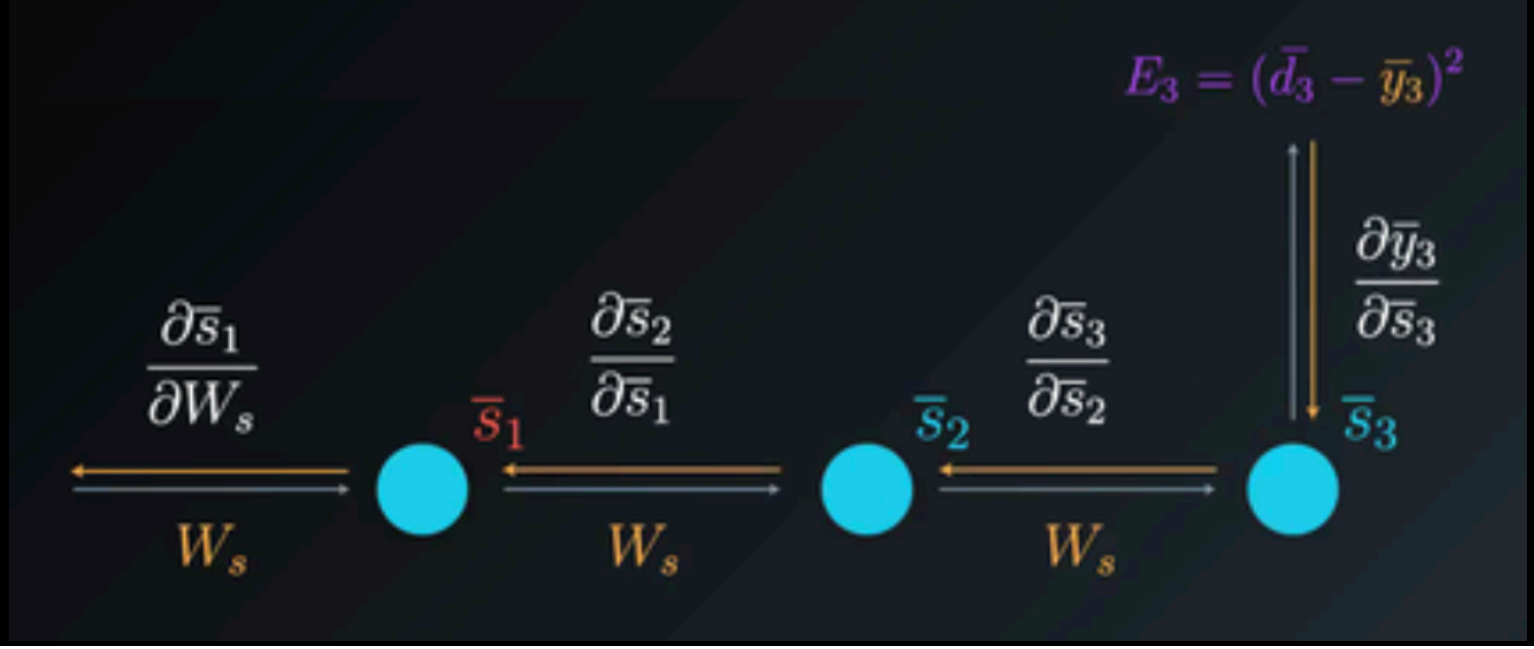
Eigenvalue 와의 비교

*R. Pascanu, "On the difficulty of training recurrent neural networks"

By inverting this proof we get the *necessary* condition for *exploding gradients*, namely that the largest singular value λ_1 is larger than $\frac{1}{\gamma}$ (otherwise the long term components would vanish instead of exploding). For tanh we have $\gamma = 1$ while for sigmoid we have $\gamma = 1/4$.

- $\lambda_1 > \frac{1}{\gamma}$ 일 때, exploding gradient이 발생한다.
- $\lambda_1 < \frac{1}{\gamma}$ 일 때, vanishing gradient이 발생한다.

http://proceedings.mlr.press/v28/pascanu13.pdf



② $$\frac{\partial E_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_k} \frac{\partial s_k}{\partial W} =$$

$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial \bar{s}_1} \cdot \frac{\partial \bar{s}_1}{\partial W_s}$$

Considering $\bar{S}_3$ Considering $\bar{S}_2$ Considering $\bar{S}_1$

③ U 도 ②와 동일한 방식으로 gradient를 구한다.

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vs_t)$$

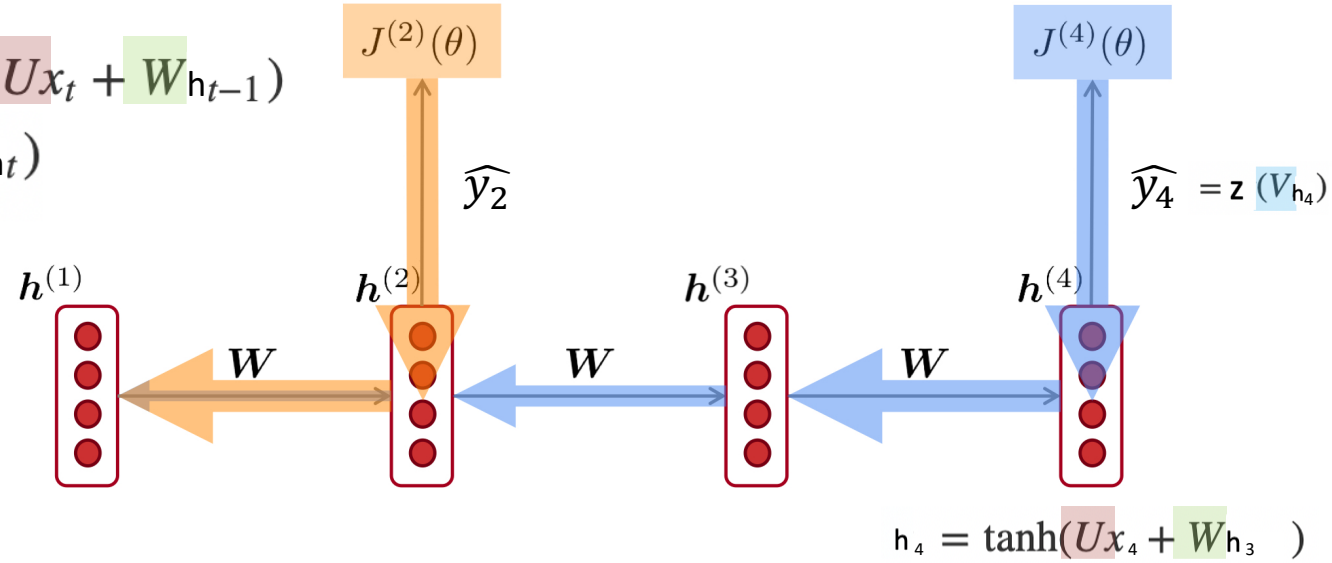
각 단계의 모든 gradient을 고려하여 update를 진행한다!!

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

RNN의 근본적인 원인

- **발생 원인 (1) 직관적 이해** (2) Backpropagation through Time

$$h_t = \tanh(Ux_t + Wh_{t-1})$$
$$\hat{y}_t = \mathrm{z}\,(Vh_t)$$



$$\frac{\partial J^{(4)}}{\partial W} = \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W}$$

$$\frac{\partial J^{(2)}}{\partial W} = \frac{\partial J^{(2)}}{\partial \widehat{y_2}} \cdot \frac{\partial \widehat{y_2}}{\partial h_2} \cdot \frac{\partial h_2}{\partial W} + \frac{\partial J^{(2)}}{\partial \widehat{y_2}} \cdot \frac{\partial \widehat{y_2}}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W}$$

가까운 거리 노드의 영향을 더 많이 받는다!

거리가 먼 gradient의 영향을 적게 받는다.

:updated with near effects, not long-term effects

$$\frac{\partial J^{(4)}}{\partial W} = \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial W} + \frac{\partial J^{(4)}}{\partial \widehat{y_4}} \cdot \frac{\partial \widehat{y_4}}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W}$$

$$\frac{\partial J^{(2)}}{\partial W} = \frac{\partial J^{(2)}}{\partial \widehat{y_2}} \cdot \frac{\partial \widehat{y_2}}{\partial h_2} \cdot \frac{\partial h_2}{\partial W} + \frac{\partial J^{(2)}}{\partial \widehat{y_2}} \cdot \frac{\partial \widehat{y_2}}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W}$$

가까운 거리 노드의 영향을 더 많이 받는다!
각 gradient은 과거의 영향력이라고 생각하면 된다.

3 Vanishing Gradient의 문제점

- **발생 문제 (1) 가까운 거리에 의존**

  [예시] Language Model에서의 문제점

  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 _____*

  RNN-LM은 마지막 단어를 예측할 때에, 7번째에 나온 tickets 의 관계를 파악해야 하지만,

  이 의존성을 파악하지 못한다.

  - RNN-LM은 Long-distance Dependencies에 의한 예측을 할 수 없기 때문이다.

3 Vanishing Gradient의 문제점

- **발생 문제 (1) 가까운 거리에 의존**

  [예시] Language Model에서의 문제점

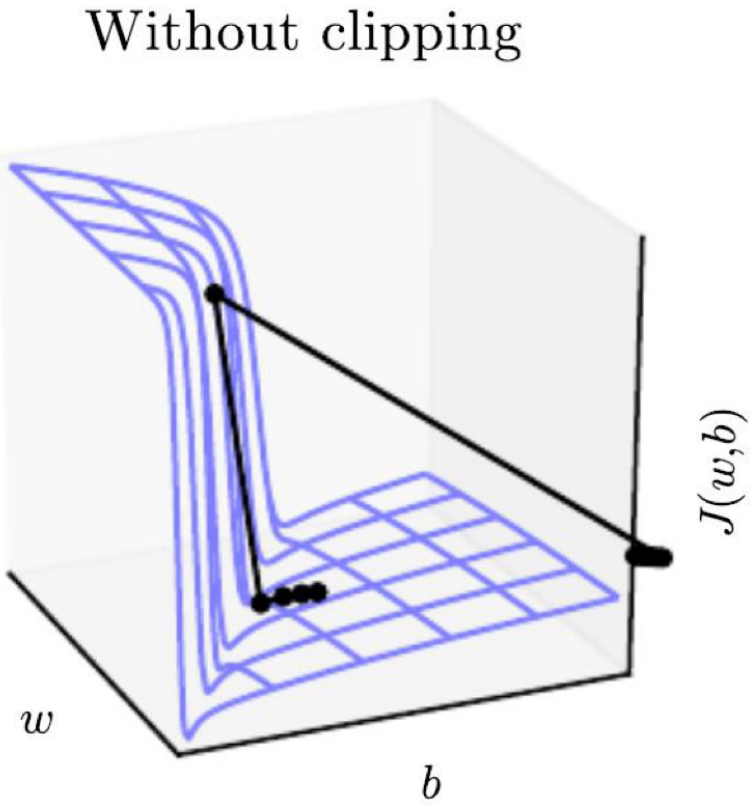
Syntactic recency: *The writer of the books is* (correct)

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

RNN-LM은 Sequential recency를 주로 학습한다.

4 Exploding Gradient의 문제점

- **발생 문제 (1) Update Step이 너무 크다**



learning rate

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

gradient

이 gradient이 너무 크다면, 한 step 또한 크다

이처럼 최적화 과정에서 목표지점을 도달하지 못한다.

5 문제의 해결

- **Vanishing Gradient 문제) 가까운 거리에만 의존**
  - ✓ Exploding Gradient 보다 어려운 문제
  - ✓ LSTM, GRU 의 등장 : 다음 장에서 다룰 예정

- **Exploding Gradient 문제) Update step이 너무 큼**
  - ✓ Gradient Clipping : threshold 로서 조절

문제의 해결

Algorithm 1 Pseudo-code for norm clipping

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

- **Exploding Gradient 문제) Update step이 너무 큼**

✓ Gradient Clipping : threshold 로서 조절

: 스케일링을 통해, 더 작은 step으로 진행

2. Variants of RNNs

1 RNN의 변형

(1) LSTM [Long Short-Term Memory]

(2) GRU [Gated Recurrent Units]

(3) Bidirectional RNNs

(4) Multi-layer RNNS

1 RNN의 변형

(1) LSTM [Long Short-Term Memory]

(2) GRU [Gated Recurrent Units]

(1) Bidirectional RNNs

Gradient Vanishing의 해결 방안: LSTM, GRU

(2) Multi-layer RNNS

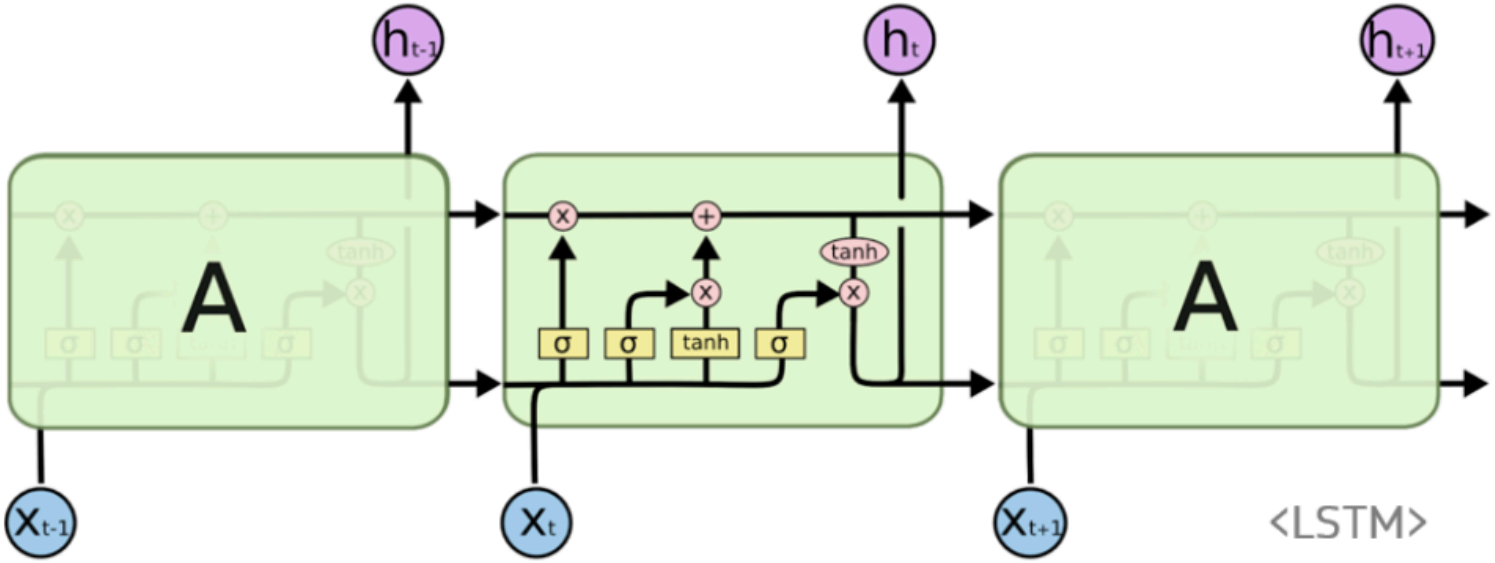
LSTM [Long Short-Term Memory]

- **LSTM 의 등장**

✓ Gate를 통해 정보량을 조절

(각 time step 마다, gate가 얼마나 open 되어 있는 지에 따라 정보량 조절)

✓ 3개의 gate 를 가진다. (Input gate, Forget gate, Output gate)



- **LSTM 이해 (1) 수식적 이해**

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$ → [Forget Gate] 이전 cell 정보 조절

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$ → [Input Gate] 현재 cell 정보 조절

$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$ → [Output Gate] 히든 state 정보 조절

항상 sigmoid로 gate를 조절한다

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$ → [New Cell Content] Cell에 새로 쓰일 정보

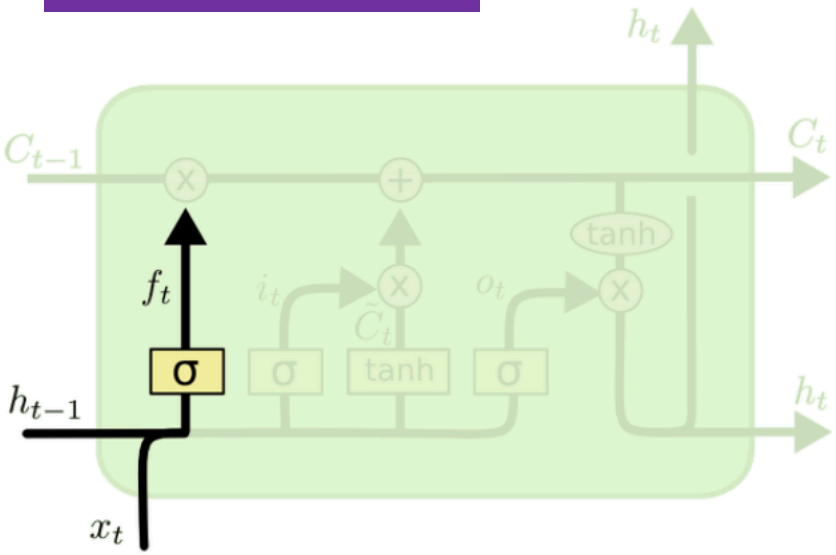
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$ → [Cell State] 이전 cell & 현재 cell 정보 반영

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$ → [Hidden State] Cell에 새로 쓰일 정보

LSTM [Long Short-Term Memory]

- **LSTM 이해 (2) 그림으로 이해**

Forget Gate



$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$f_t = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$

LSTM의 forget gate layer

[Forget Gate]

✓ 시그모이드 함수를 통해, 이전 cell state로부터 어떤 정보를 버릴 것인지를 정함.

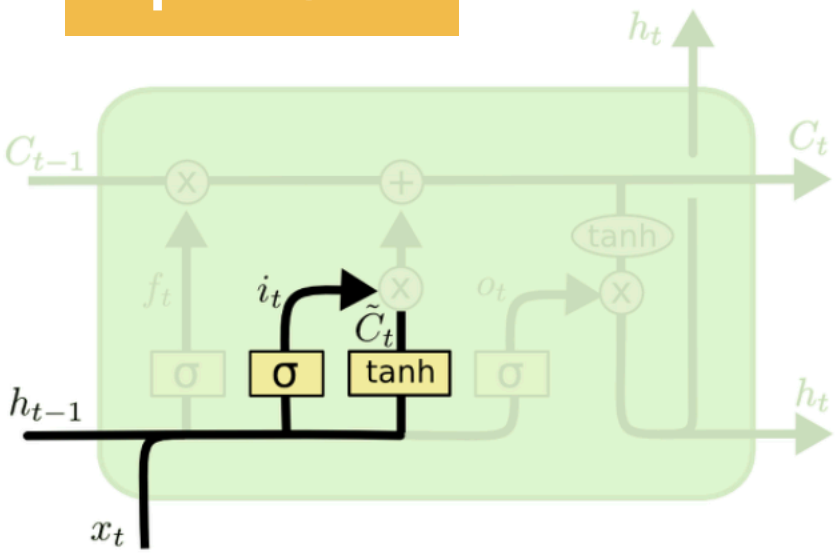
✓ 0과 1 사이의 값을 C_{t-1} 에 보내, 1이면 "모든 정보를 보존해라", 0이면 "죄다 갖다 버려라"

https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

LSTM [Long Short-Term Memory]

- **LSTM 이해 (2) 그림으로 이해**

Input Gate



$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

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

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

LSTM의 input gate layer

[Input Gate]

✓ 새로운 cell state의 정보량을 조절. Sigmoid layer가 어떤 값을 업데이트할 지 정함

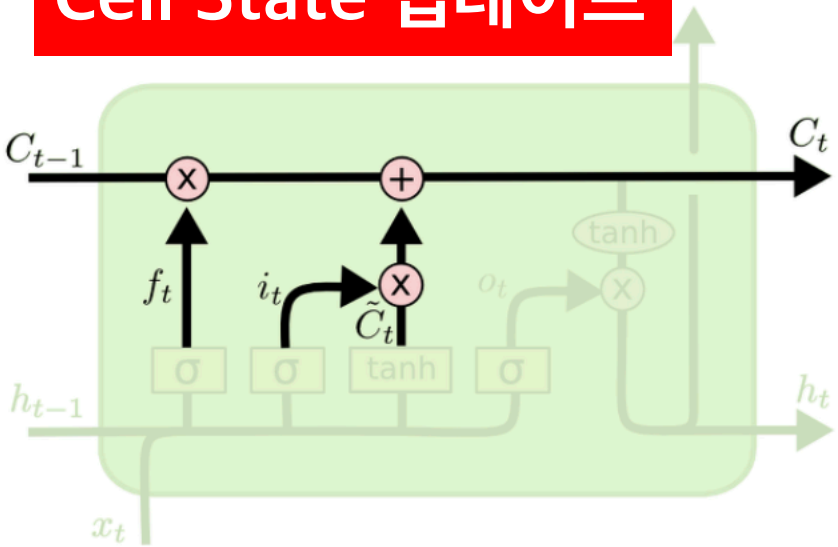
✓ 새로운 Cell state C_t를 생성

https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

LSTM [Long Short-Term Memory]

- **LSTM 이해 (2) 그림으로 이해**

Cell State 업데이트



$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

LSTM의 cell state 업데이트

[Cell Sate Update]

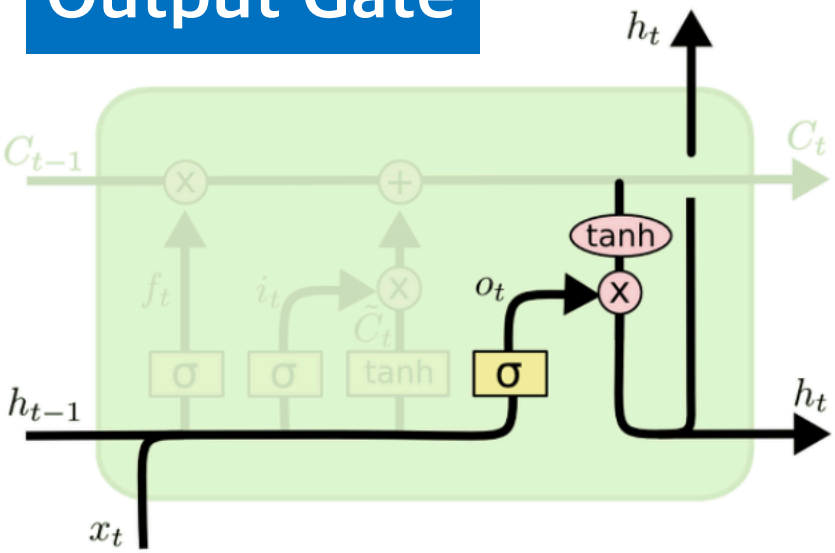
✓ 과거 cell 와 새로운 cell로 현재 Cell 을 생성한다.

https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

LSTM [Long Short-Term Memory]

- **LSTM 이해 (2) 그림으로 이해**

Output Gate



$$h_t = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

$$o_t = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

LSTM의 output gate layer

[Output Gate]

- ✓ 무엇을 output으로 내보낼 지 정함
- ✓ sigmoid layer에 input 데이터를 넣어 cell state의 어느 부분을 output으로 내보낼 지를 정함
- ✓ 현재 cell state를 tanh layer에 넣어 -1과 1 사이의 값을 받고 시그모이드 값과 곱해줌

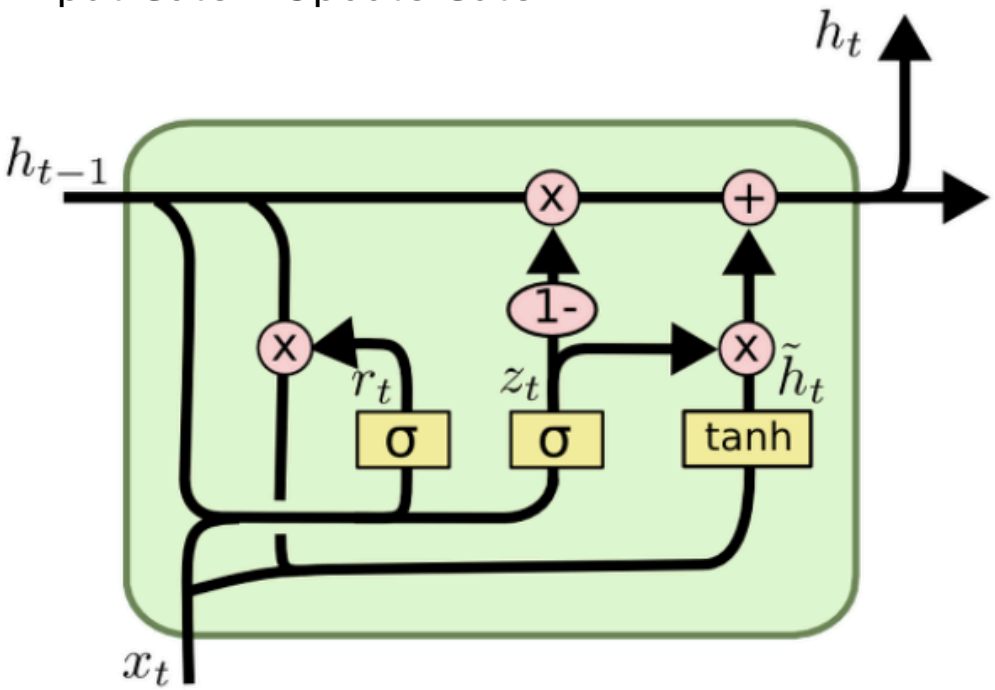
https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

GRU [Gated Recurrent Units]

- **GRU 의 등장**

  ✓ LSTM의 변형 버전으로 LSTM 보다 단순한 구조를 지닌다.

  ✓ Forget Gate + Input Gate = Update Gate



GRU [Gated Recurrent Units]

- **GRU 의 등장 (1) 수식적 이해**

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

➡ [Update Gate] hidden state 조절

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

➡ [Reset Gate] 이전 Hidden state 조절

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

➡ [New Hidden State Content] reset gate을 통해, 이전 hidden state 조절

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

➡ [Hidden State] update gate을 통해 과거, 현재 hidden state 조절

GRU [Gated Recurrent Units]

- **GRU 의 등장 (2) 그림으로 이해**

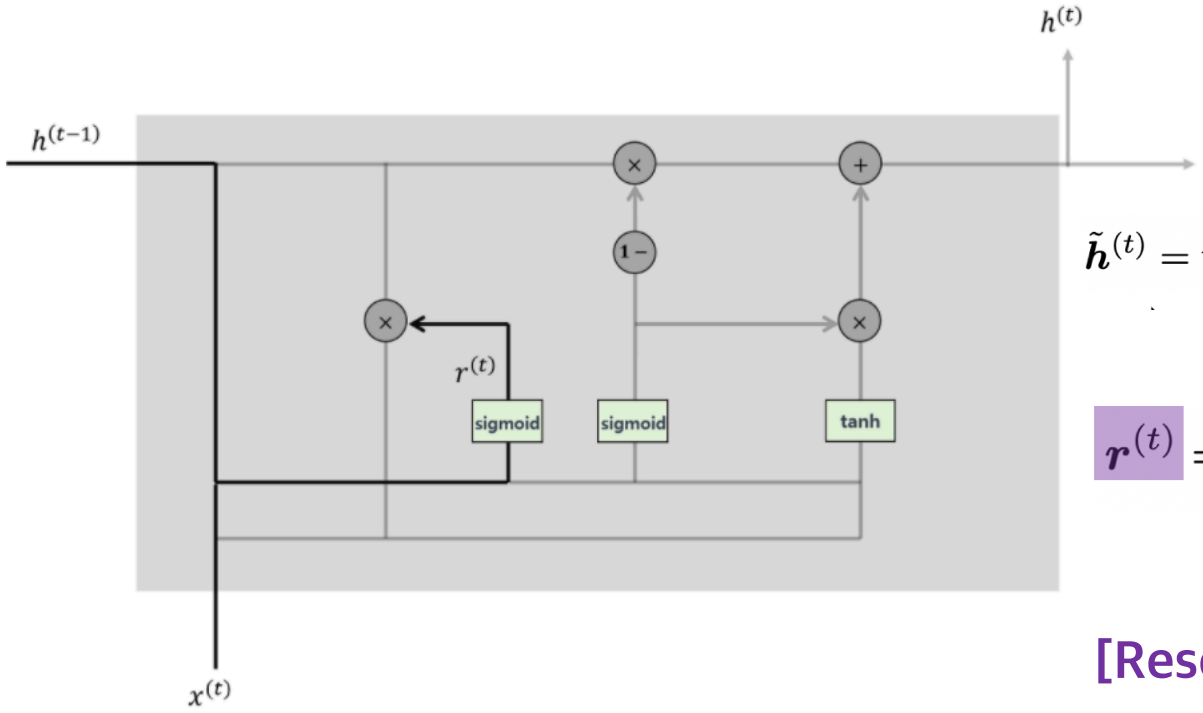


Figure 2: GRU, Reset Gate

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h\left(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}\right) + \boldsymbol{U}_h\boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r\boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

[Reset Gate]

✓ 과거 히든 노드의 정보를 적당히 잊음

출처: https://yjjo.tistory.com/18

GRU [Gated Recurrent Units]

- **GRU 의 등장 (2) 그림으로 이해**

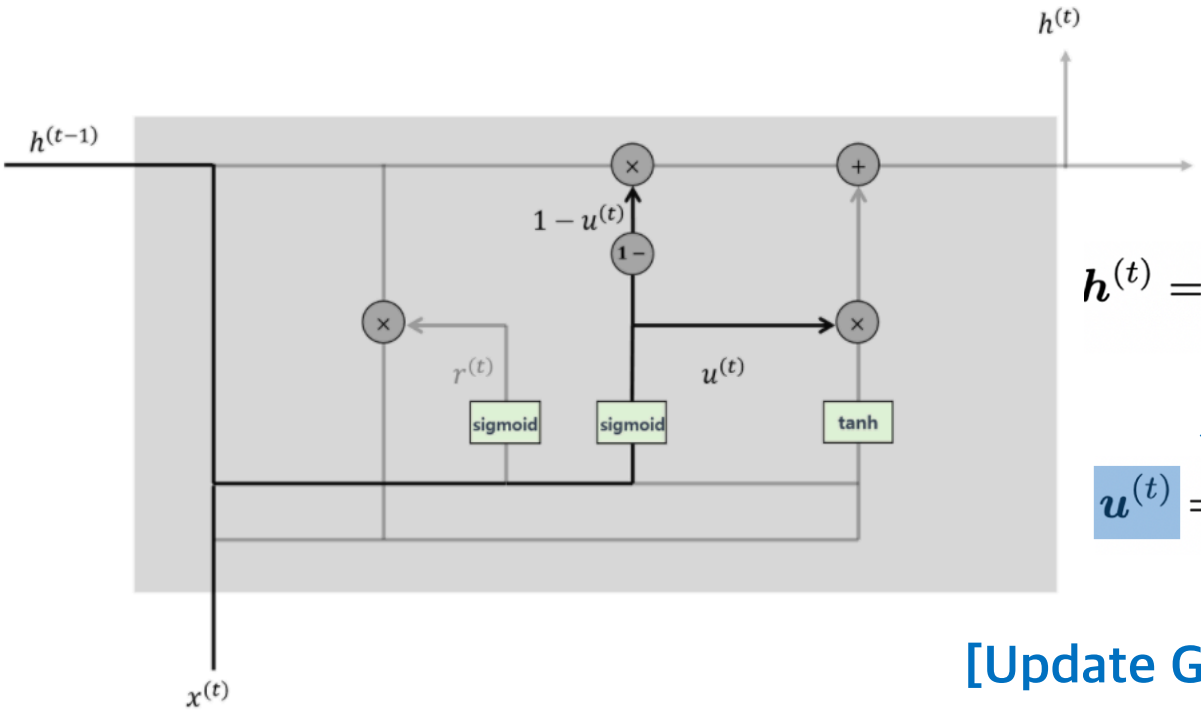


Figure 3: GRU, Update Gate

$$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$$

$$u^{(t)} = \sigma\left(W_u h^{(t-1)} + U_u x^{(t)} + b_u\right)$$

[Update Gate]

✓ 과거와 현재의 히든노드를 반영할 비율을 정함

✓ LSTM의 foget+input gate

출처: https://yjjo.tistory.com/18

- **LSTM/GRU 이해 : Vanishing Gradient 문제 해결?**
  - ✓ GRU가 더 시간효율적이다. (학습할 가중치가 적기 때문에)
  - ✓ Vanilla Rnn 보다는 정보를 잘 보존한다.

[예] Forget Gate가 모든 정보를 기억하도록 설정한다 : 확실한 정보 보존

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

$$\Rightarrow \quad f_t = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

* 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에, 1이라면 이전 상태의 정보를 온전히 기억하게 된다.

1 RNN의 변형

(1) LSTM [Long Short-Term Memory]

추가적인 RNN 변형 모델

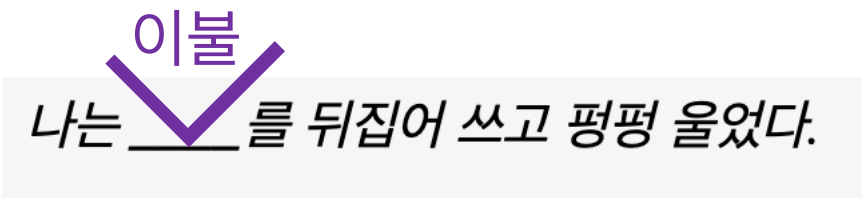
(2) GRU [Gated Recurrent Unit]

(1) Bidirectional RNNs

(2) Multi-layer RNNS

5 Bidirectional RNN

- **Motivation**



✓ 인간에게는 간단한 과제지만, LM에서는 어렵다.

✓ ' 나는' 뒤에는 올 수 있는 단어 w 는 많지만,

'를 뒤집어 쓰고 펑펑 울었다' 앞에 올 확률이 높은 단어는 적다.

✓ 이렇듯, 정방향 추론만이 항상 중요한 것이 아니다.

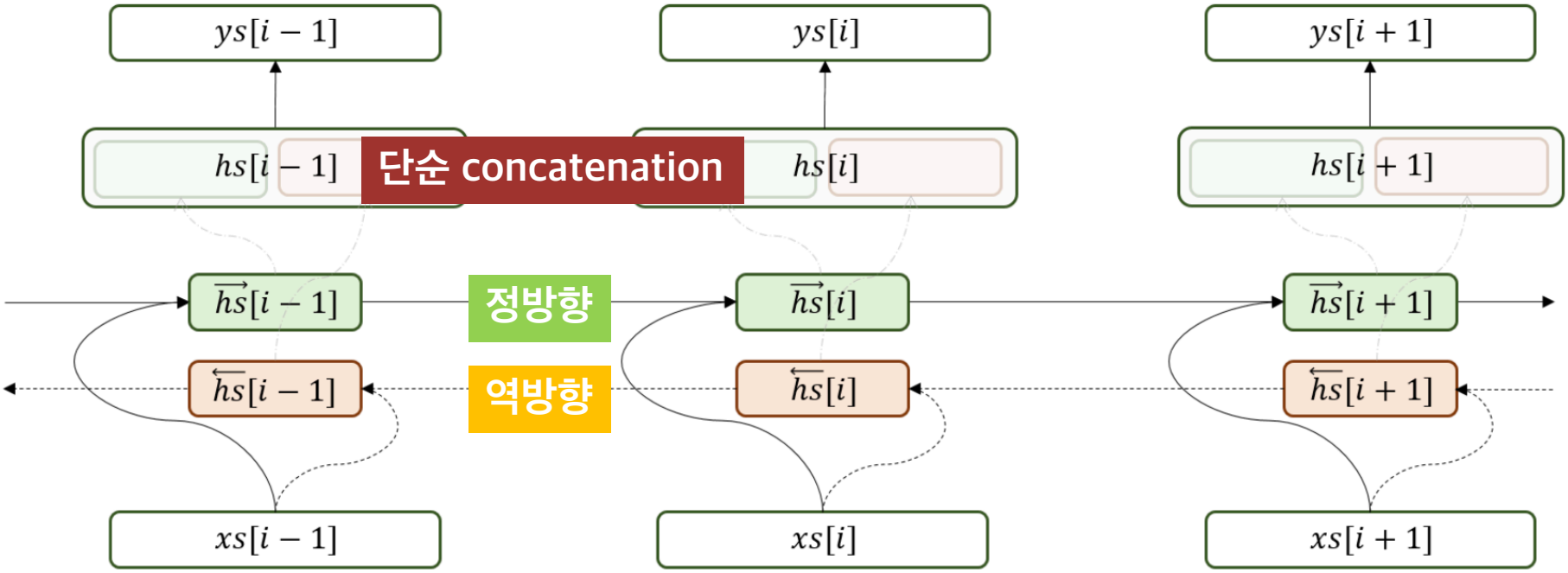
✓ 하지만, 우리가 지금까지 공부한 RNN은 정방향만을 고려한다.

Bidirectional RNN 의 등장

http://intelligence.korea.ac.kr/members/wschoi/nlp/deeplearning/Bidirectional-RNN-and-LSTM/

Bidirectional RNN

- **Bidirectional RNN의 구조**



http://intelligence.korea.ac.kr/members/wschoi/nlp/deeplearning/Bidirectional-RNN-and-LSTM/

Bidirectional RNN

- **Bidirectional RNN의 구조**

Any RNN is possible

정방향 Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

역방향 Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

단순 concatenation

✓ 전체 input sequence가 있을 때에만 사용할 수 있다. (Language Model에선 사용불가-정방향 존재)

✓ BERT(Bidirectional Encoder Representations from Transformers)의 기원

Multi-layer RNN

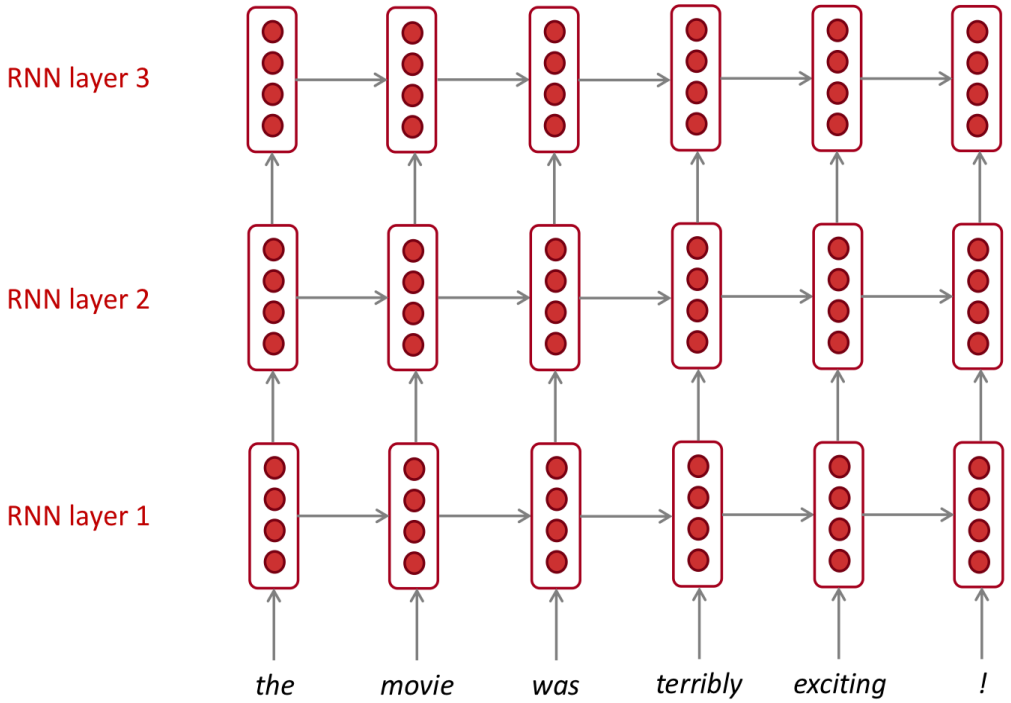
- **특징**

✓ Stacked RNNs 로 불린다.

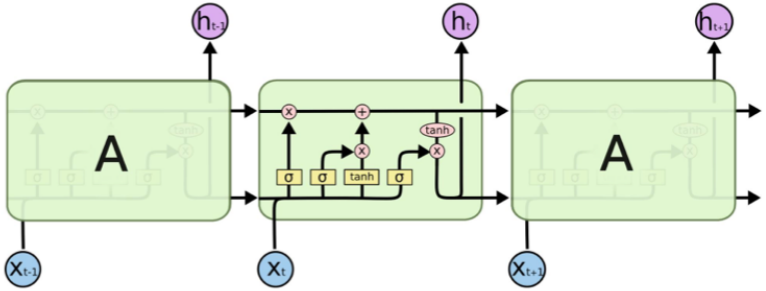
✓ Deep 하도록 차원을 증가시킨다.

✓ 더 복잡한 feature을 나타낼 수 있다.

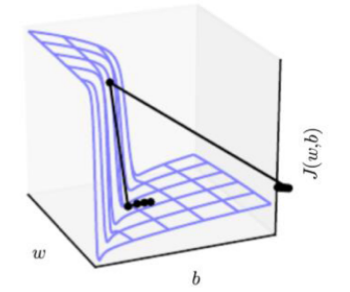
✓ Example : BERT는 24 층을 지닌다.



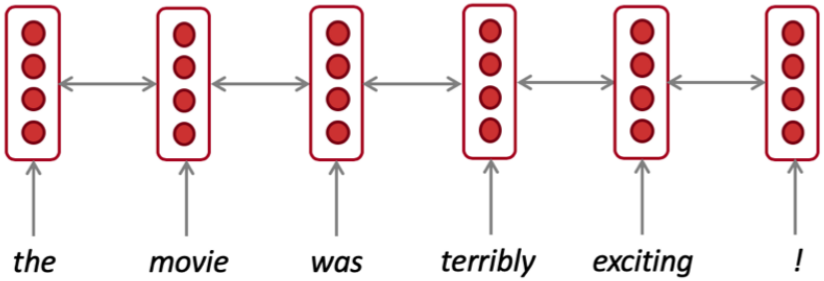
7 Conclusion



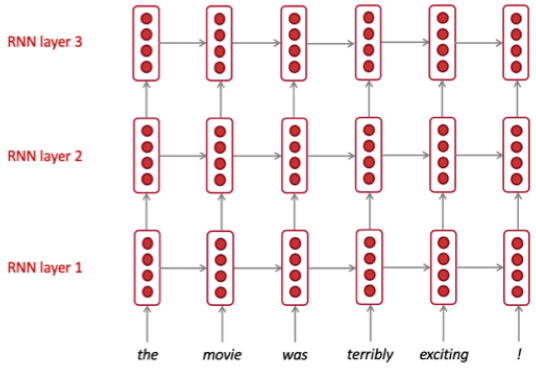
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients



3. Use bidirectionality when possible



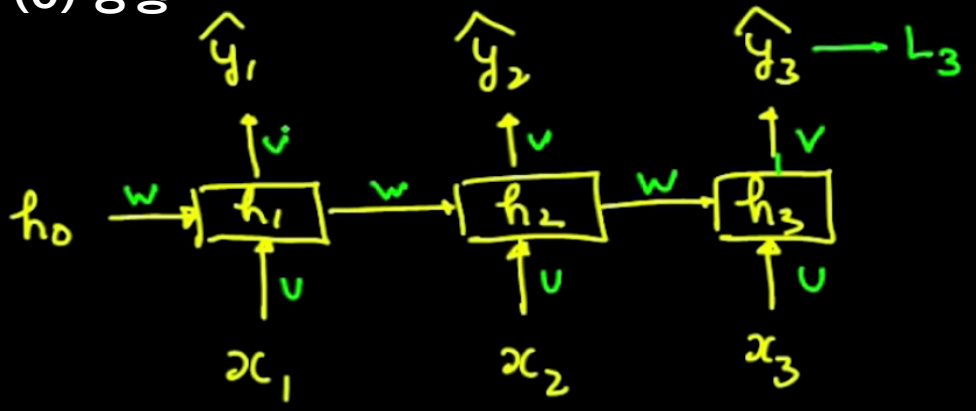
4. Multi-layer RNNs are powerful, but you might need skip/dense-connections if it's deep

+ 부록

2 RNN의 작동 원리

(2) Backward Propagation through TIME

(0) 상황



(1) 구하고자 하는 목표

$$L = \sum_{t=1}^{T} L_t$$

Consider $\frac{\partial L_3}{\partial W}, \frac{\partial L_3}{\partial V}, \frac{\partial L_3}{\partial U}$

(2) V 구하기

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V}$$

: 이전 시간을 고려하지 않아도 됨

(3) W 구하기 : 아래 사이트에서 자세히 확인 가능

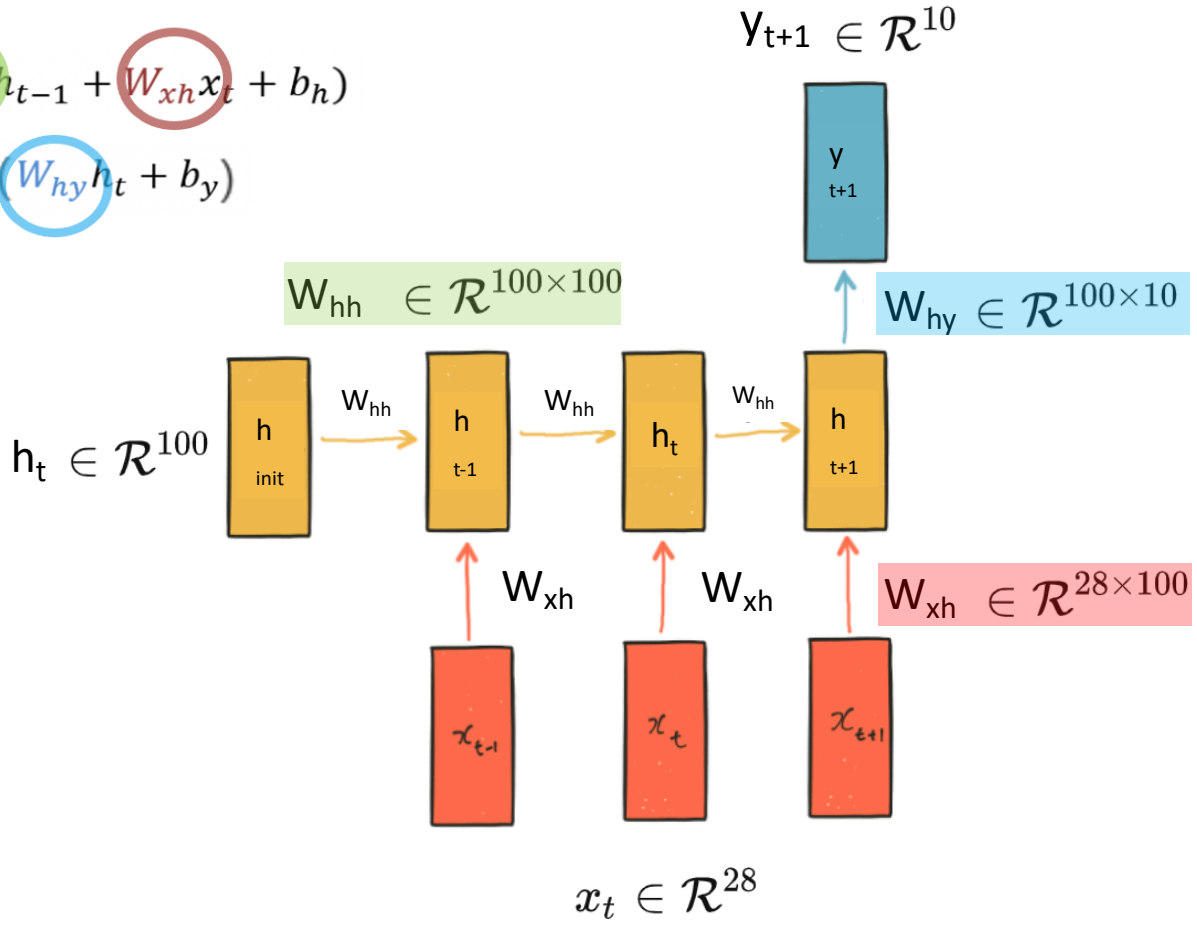
https://m.blog.naver.com/infoefficien/22121006151

https://www.youtube.com/watch?v=RrB605Mbpic

(3) 차원 수 확인

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

$$\hat{y}_t = softmax(W_{hy} h_t + b_y)$$

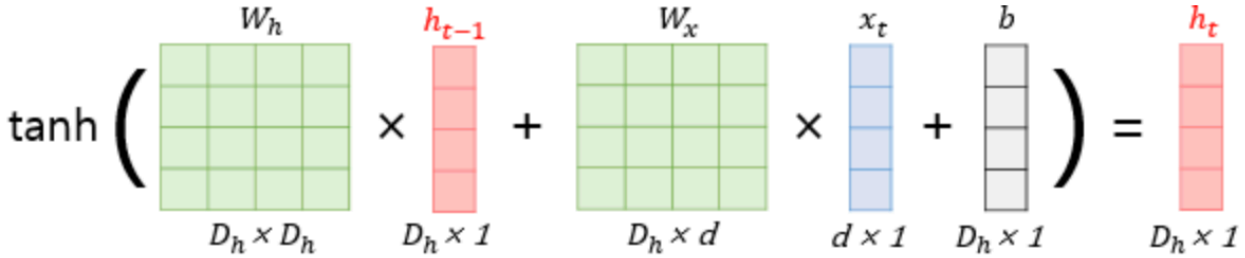


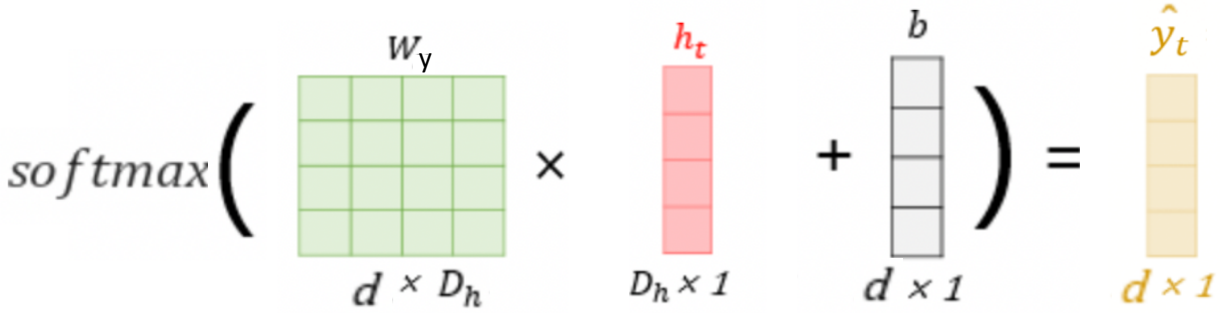
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(3) 차원 수 확인

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$\hat{y}_t = softmax(W_{hy}h_t + b_y)$$

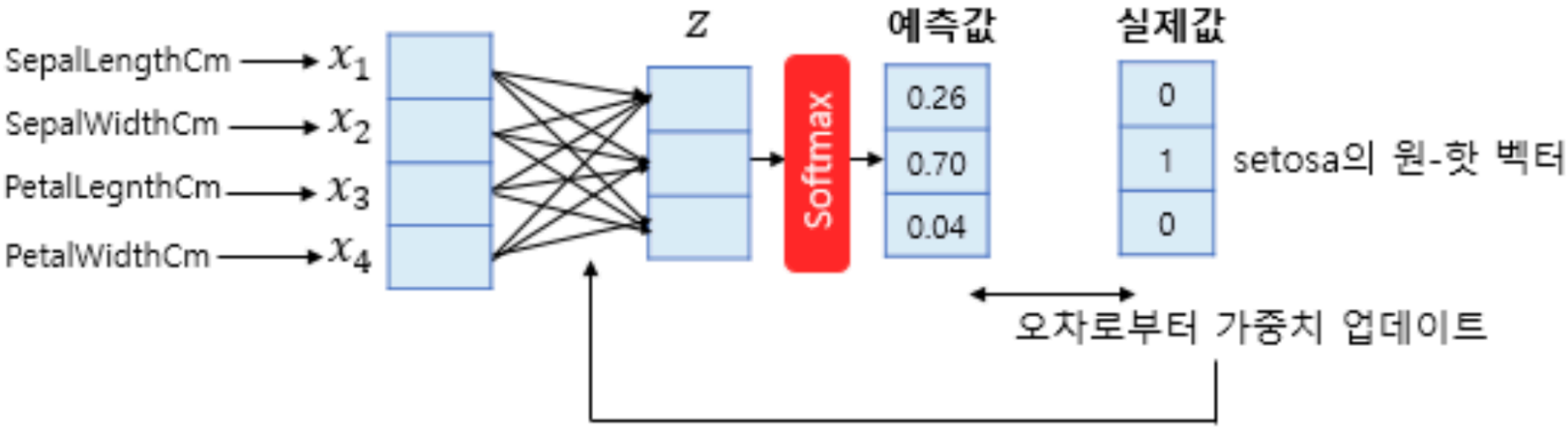




각 클래스에 대한 확률

출처: https://wikidocs.net/22886
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

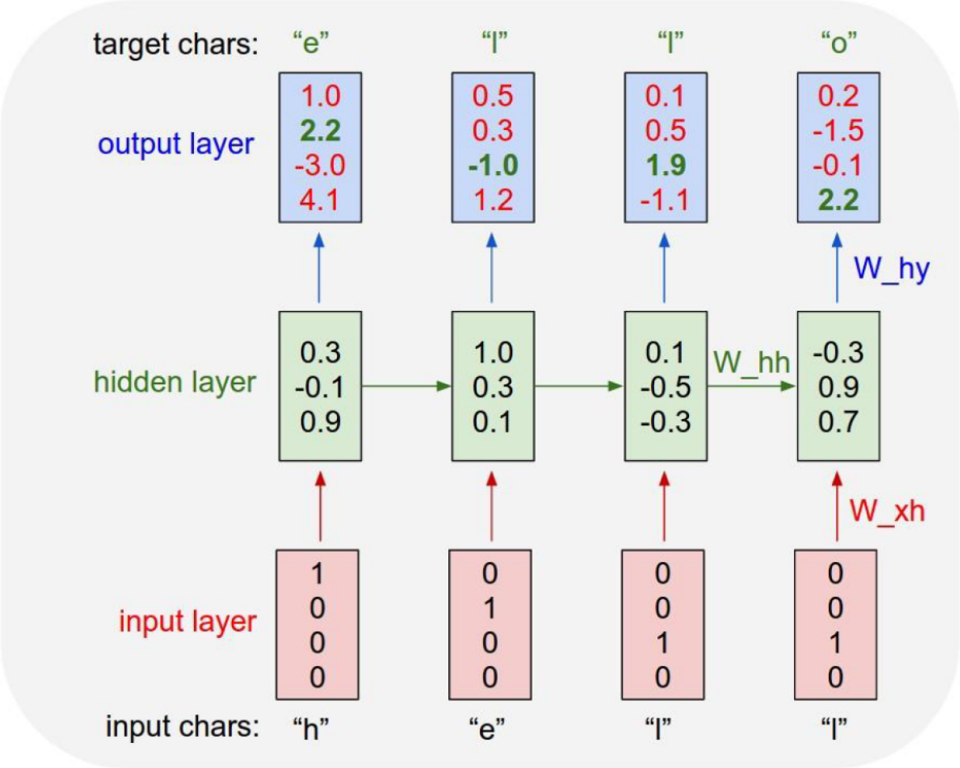
(2) 차원 수 확인



간단한 softmax 이해 그래프

출처: https://wikidocs.net/22886
출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(4) 간단한 예시 [Language Model]



*d = 4, D_h = 3

Output 값 반환

순전파

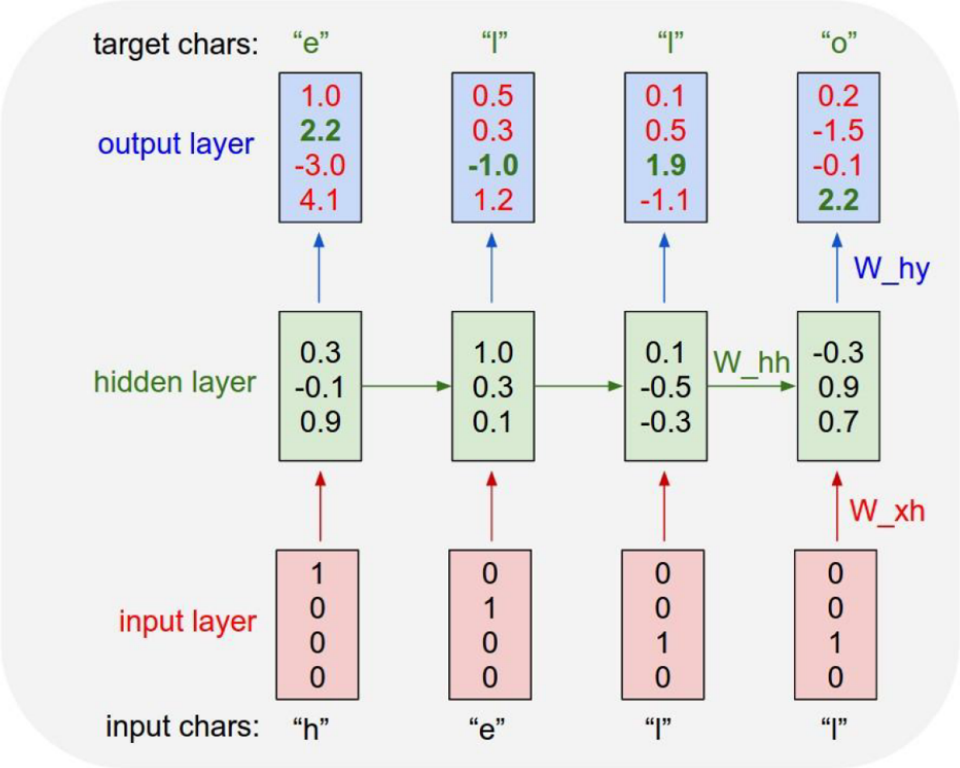
W_xh , W_hh, W_hy 의 **shared weights**

순전파

Input: one hot vector의 단어

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

(4) 간단한 예시 [Language Model]



*d = 4, D_h = 3

Loss 구하기: (using softmax)
Output 값과 실제 target 값의 오차

역전파

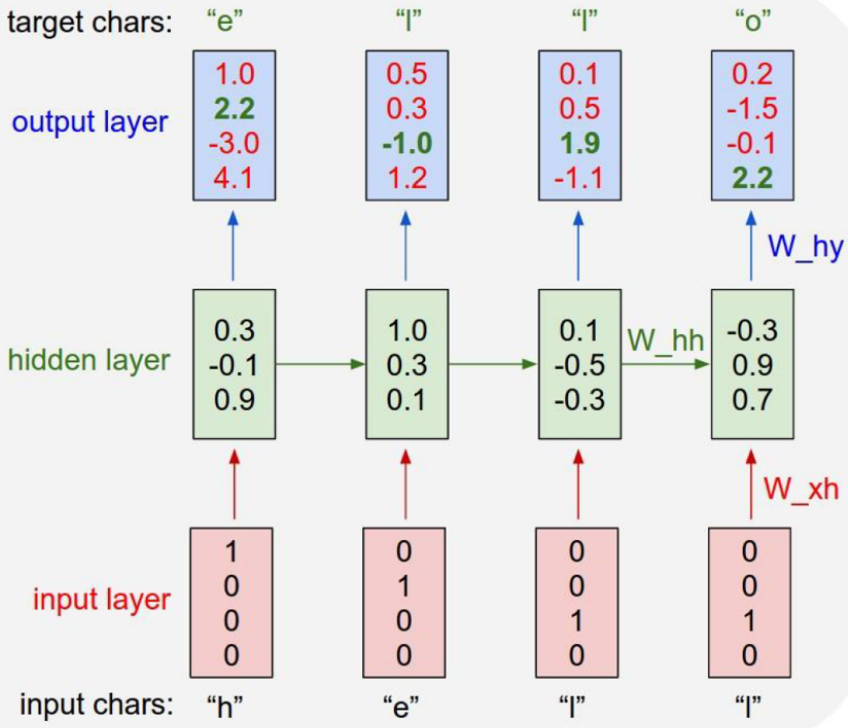
W_xh , W_hh, W_hy 의 **shared weights**
동일한 업데이트 진행

새로운 weight 값으로 학습 재진행

출처: https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

2 RNN의 작동 원리

(4) 간단한 예시 [Language Model]

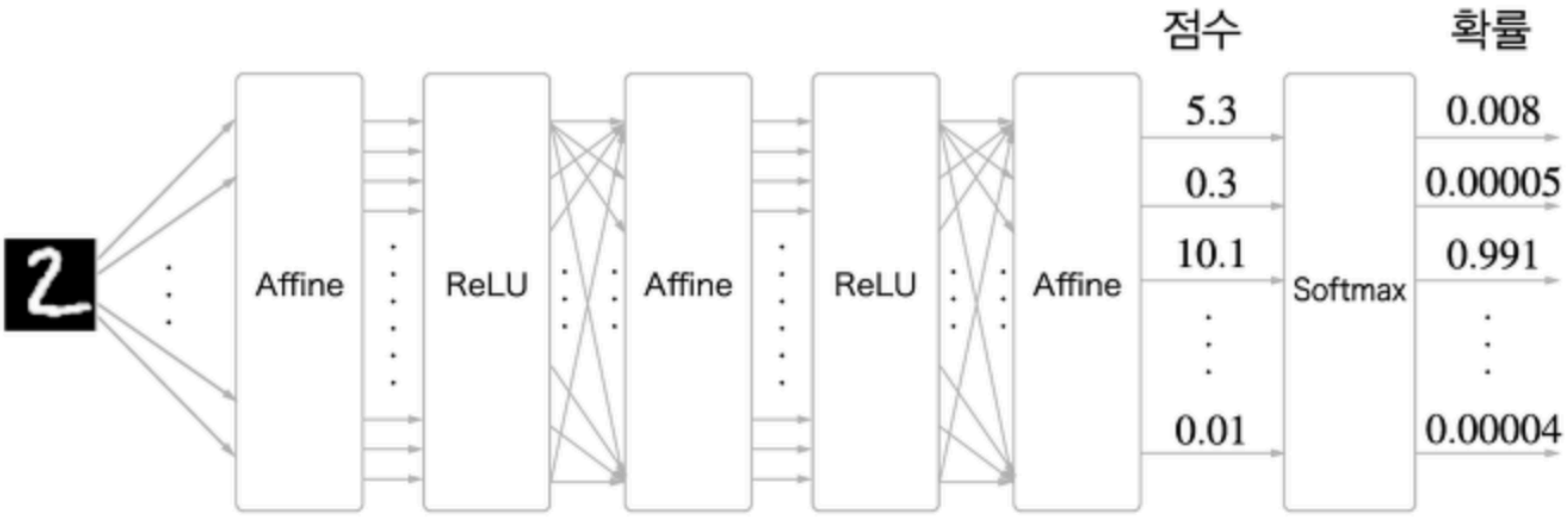


왜 softmax 를

이용하지 않을까…?

(4) 간단한 예시 [Language Model]

Test일 때이기 때문 …



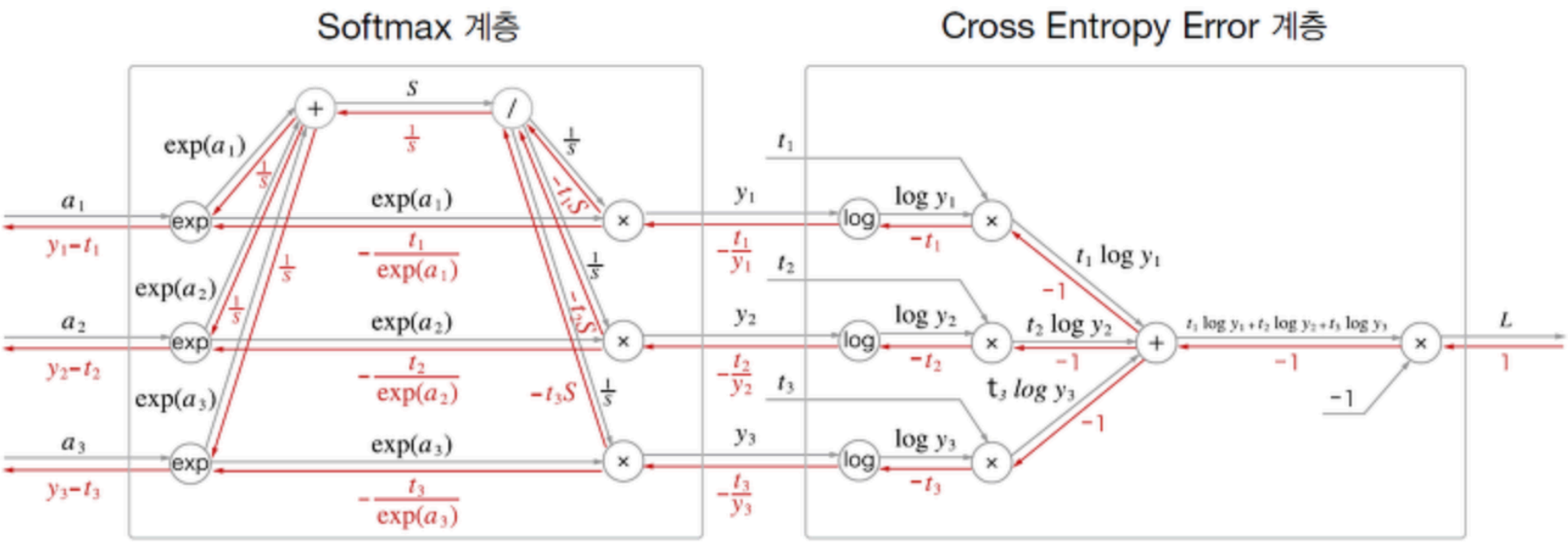
학습을 진행할때에는 softmax 계층이 필요하다.

검증을 진행할때에는 softmax 계층이 필요하지 않다.

출처: https://excelsior-cjh.tistory.com/171

(4) 간단한 예시 [Language Model]

(+α) Softmax-with-loss 계층을 이용하여 역전파를 진행한다



출처: https://excelsior-cjh.tistory.com/171

RNN의 근본적인 원인

- **발생 원인 (2) 수식적 이해**

(3) Gradient of step i with the previous step j

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

(4) Rephrase

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

*W_h 값이 작을 때, i,j가 멀어질수록 기하급수적으로 작아진다.

https://stats.stackexchange.com/questions/193352/in-rnn-back-propagation-through-time-why-is-the-dh-t-dh-t-1-diagonal

원인

p i with the previous step j

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

) Rephrase

$$\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$

왜 diagonal matrix

로 만드는 것일까…?

https://stats.stackexchange.com/questions/193352/in-rnn-back-propagation-through-time-why-is-the-dh-t-dh-t-1-diagonal

- **발생 원인 (2) 수식적 이해** (Bias term을 제외하여 생각)

$$\mathbf{h_{t+1}} = tanh(\mathbf{U x_t} + \mathbf{W h_t})$$ → Element-wise로 계산된다.
:각 행에 tanh이 씌워지는 것 →

h_t 와 h_{t+1} 는 vector 형태이다. → 도함수 또한 vector의 형태로 유지된다

우리가 구하고자하는 것 ; $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}$ → 기존 weight의 형태와 동일 하지 않음.

→ diagonal matrix를 만들어, 형태를 유지한다

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

이들의 도함수는 Jacobian 형태이다.

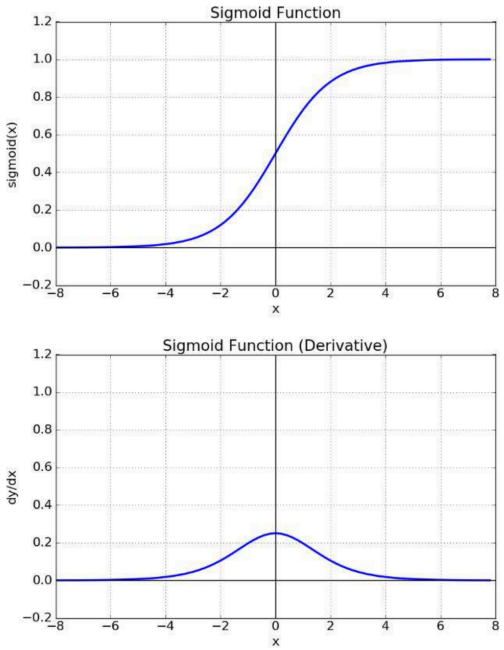
2 RNN의 근본적인 원인

- **개인적으로 궁금했던 점**

(1) Sigmoid 함수를 사용해서 발생하는 문제가 아닐까?

시그모이드
함수 결과 범위 : 0~1
도함수 결과 범위 : 0~0.25

미분 값



시그모이드 함수의 경우,
도함수 범위가 ¼로 줄어들기 때문에,
Vanishing gradient이 발생하는 것이 아닌가?

**즉, 다른 활성 함수를 사용하면,
이런 문제가 없지 않을까?
(예를 들면 ReLu 나 Tanh)**

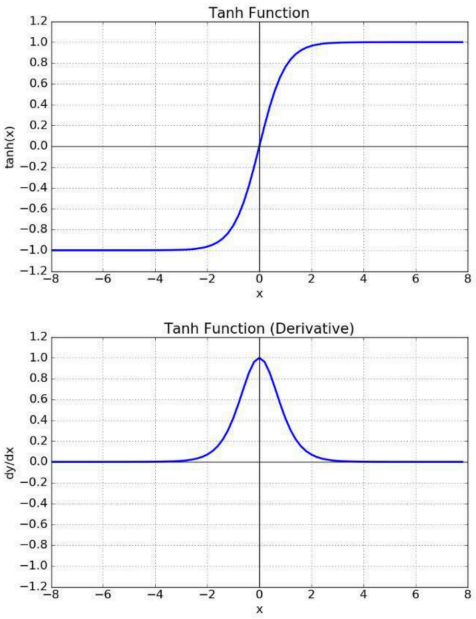
참고: https://muzukphysics.tistory.com/193

2 RNN의 근본적인 원인

- **개인적으로 궁금했던 점**

(1-A) Tanh 함수를 사용했을 경우

하이퍼볼릭 탄젠트
함수 결과 범위 : -1~1
도함수 결과 범위 : 0~1



미분 값

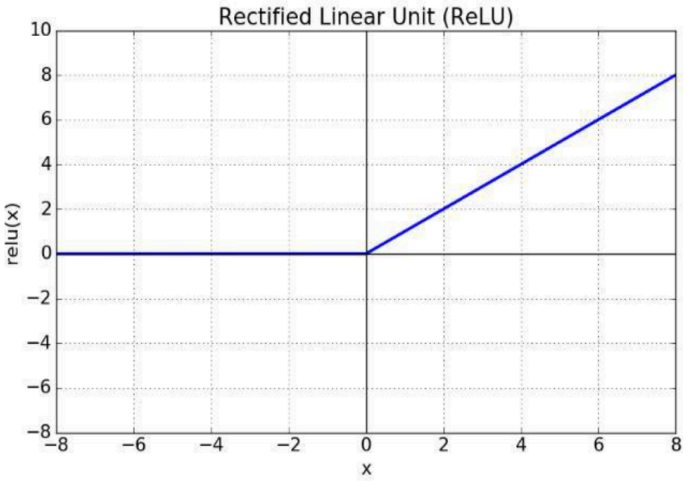
문제 해결

참고: https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b

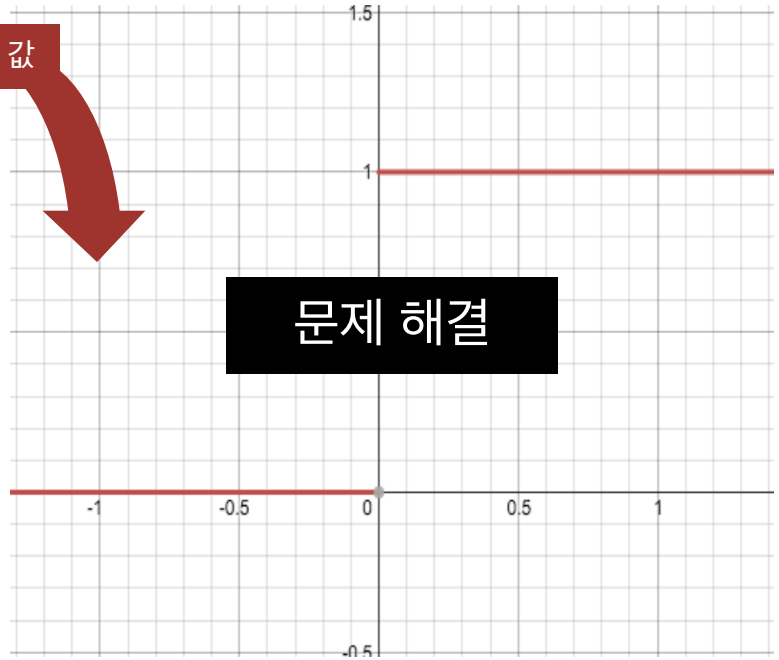
- **개인적으로 궁금했던 점**

(1-A) ReLu 함수를 사용했을 경우

ReLU
함수 결과 범위 : 0~x
도함수 결과 범위 : 0 or 1



미분 값



문제 해결

*그러나 weight이 음수일 때 업데이트가 이루어지지 않기에, 주의해야한다.

참고: https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b

Bidirectional RNN

- **Bidirectional RNN의 forward propagation**

$$ys[i] = W_{hy}\, hs[i] + by$$
$$Y \times 1 \quad Y \times H \quad H \times 1 \quad Y \times 1$$

$$\overleftarrow{hs}[i] = \tanh(\overleftarrow{W_{xh}} xs[i] + \overleftarrow{W_{hh}}\, \overleftarrow{hs}[i+1] + \overleftarrow{bh})$$
$$\frac{H}{2} \times 1 \quad \frac{H}{2} \times V \quad V \times 1 \quad \frac{H}{2} \times \frac{H}{2} \quad \frac{H}{2} \times 1 \quad \frac{H}{2} \times 1$$

$$\overrightarrow{hs}[i] = \tanh(\overrightarrow{W_{xh}}\, xs[i] + \overrightarrow{W_{hh}}\, \overrightarrow{hs}[i-1] + \overrightarrow{bh})$$
$$\frac{H}{2} \times 1 \quad \frac{H}{2} \times V \quad V \times 1 \quad \frac{H}{2} \times \frac{H}{2} \quad \frac{H}{2} \times 1 \quad \frac{H}{2} \times 1$$

$$hs[i] = \begin{bmatrix} \overrightarrow{hs}[i] \\ \overleftarrow{hs}[i] \end{bmatrix} \quad stacked!$$

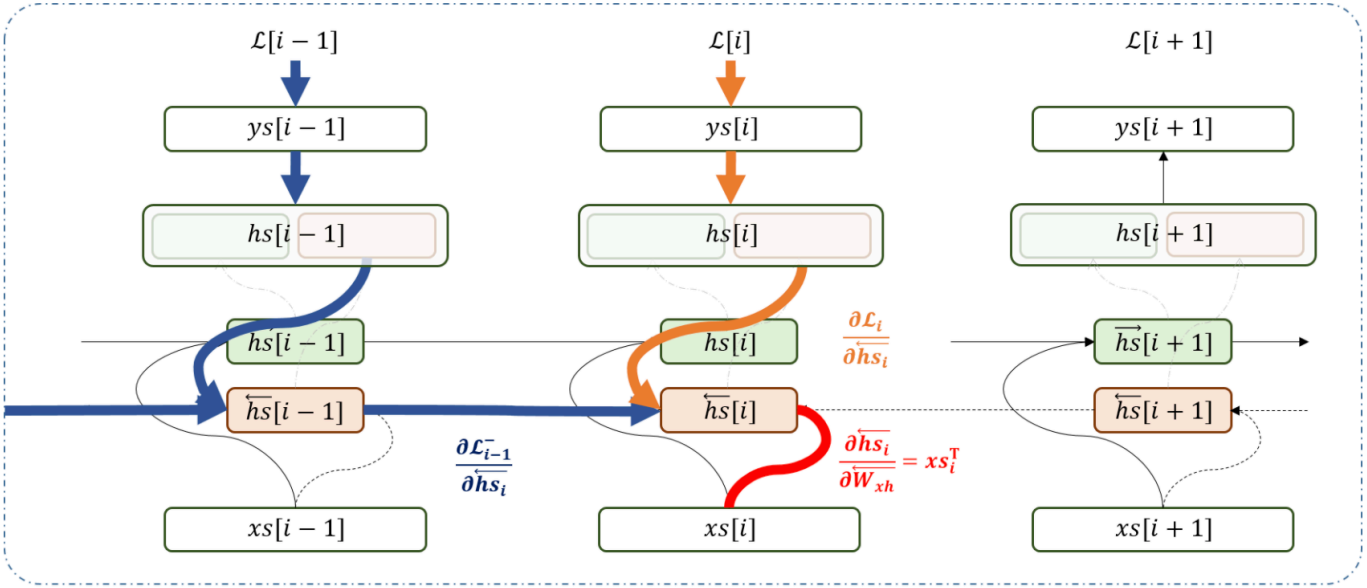
$Output\ Size = Y$

$Hidden\ Size = H$

$Vocabulary\ size = V$

Bidirectional RNN

- **Bidirectional RNN의 backward propagation**



$$let:\ \mathcal{L} = \Sigma_{i=0}^{T}\mathcal{L}_i \qquad let:\ \mathcal{L}^{-}_{k} = \Sigma_{i=0}^{k}\mathcal{L}_i$$

$$\frac{\partial \mathcal{L}}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\frac{\partial L}{\partial \overleftarrow{hs}_i}\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\sum_{j}^{T}\frac{\partial L_j}{\partial \overleftarrow{hs}_i}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\frac{\partial \mathcal{L}^{-}_{i-1}}{\partial \overleftarrow{hs}_i} + \frac{\partial \mathcal{L}_i}{\partial \overleftarrow{hs}_i}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\left[\frac{\partial \mathcal{L}^{-}_{i-1}}{\partial hs_i}\right]_{\frac{H}{2}:H} + \left[\frac{\partial \mathcal{L}_i}{\partial hs_i}\right]_{\frac{H}{2}:0}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}}$$

$$= \sum_{i=0}^{T}\left(\left[\frac{\partial \mathcal{L}^{-}_{i-1}}{\partial hs_i}\right]_{\frac{H}{2}:H} + \left[\frac{\partial \mathcal{L}_i}{\partial hs_i}\right]_{\frac{H}{2}:0}\right)\odot\left(1-\overleftarrow{hs}_i^{\,2}\right)xs_i^{\mathrm{T}}$$

TRAIN AND TEST