텍스트 세미나

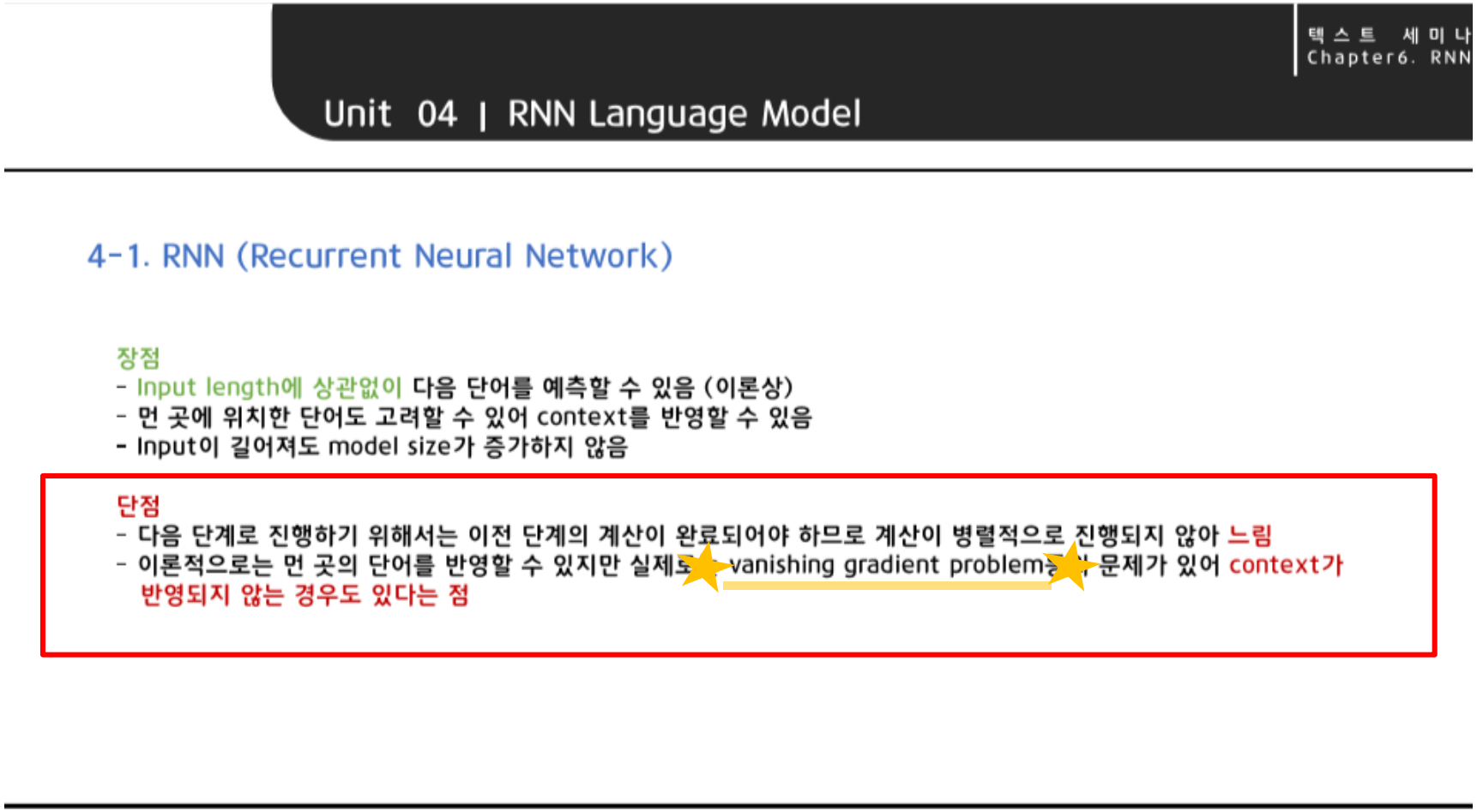
ToBig's 13기 최혜빈

Lecture 7

# Vanishing Gradients And Fancy RNNs

# Unit 00 | Overview

- 지난 강의에서 …



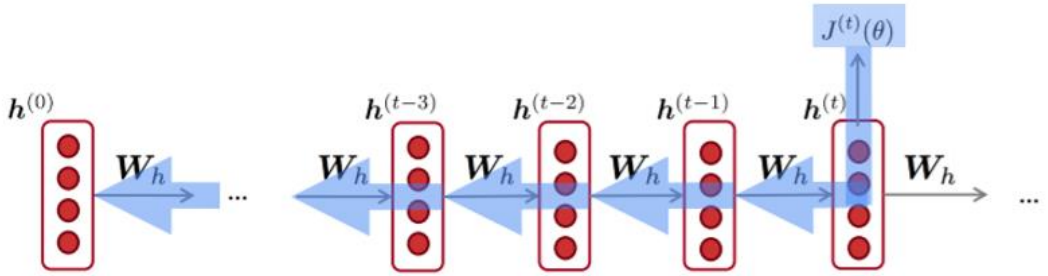
# Contents

# Contents

# Unit 01 | Vanishing gradient problem

- Remark) Backpropagation for RNNs

- RNN 네트워크를 학습하는 것은 기존의 신경망 모델을 학습하는 것과 매우 유사
- 단, 기존의 backpropagation을 그대로 사용하진 못하고 BPTT(Backpropagation Through Time)이라고 하는 약간 변형된 알고리즘을 사용
- 그러나 vanishing/exploding gradient 문제로 단순한 RNN을 BPTT로 학습시키는 것은 어려움




$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}$$

**Question:** How do we calculate this?

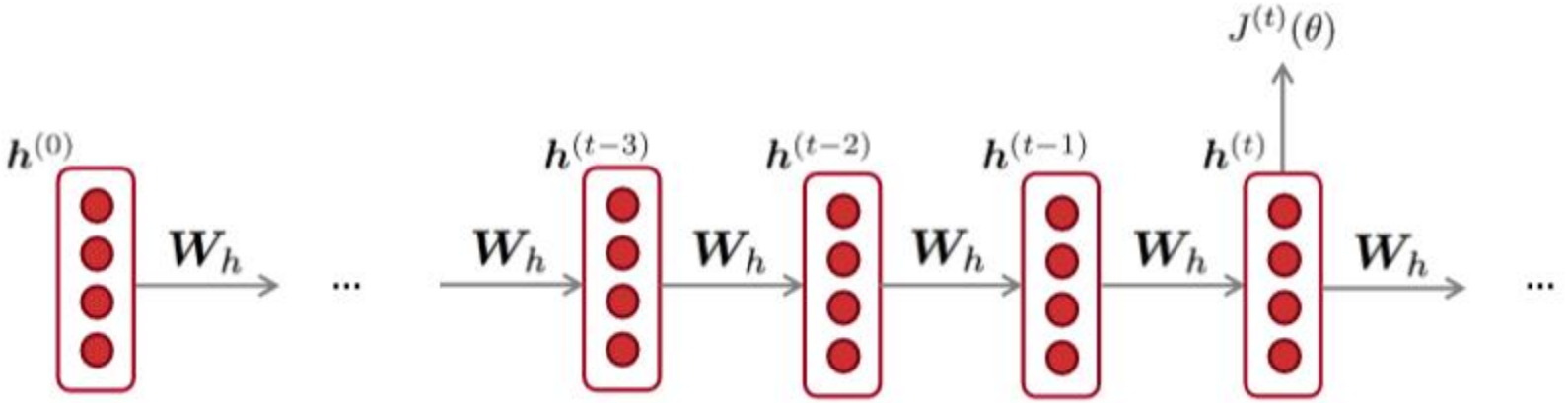
**Answer:** Backpropagate over timesteps *i*=*t*,...,0, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

# Unit 01 | Vanishing gradient problem

## 1. Vanishing gradient intuition

- Gradient 문제
  - RNN 역전파시 그래디언트가 너무 작아지거나, 반대로 너무 커져서 학습이 제대로 이뤄지지 않는 문제

- 수식으로 살펴보면
  - Vanilla RNN 셀 t번째 시점의 hidden state : $\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$



# Unit 01 | Vanishing gradient problem

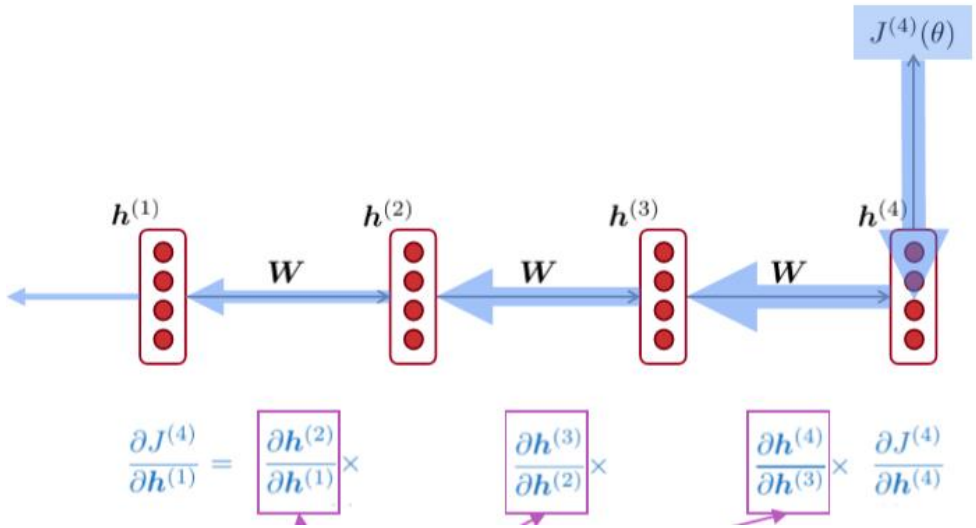
## 1. Vanishing gradient intuition

- **수식으로 살펴보면**
  - Vanilla RNN 셀 t번째 시점의 hidden state :

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

  - 아래의 RNN 구조에서 네번째 시점의 손실 $J^{(4)}$에 대한 $h^{(1)}$의 gradient는 chain rule에 의해 계산!

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




  ⇒ I번째 시점에서의 손실 $J^{(i)}$에 대한 $h^{(1)}$의 gradient

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

# Unit 01 | Vanishing gradient problem

## 1. Vanishing gradient intuition

- 수식으로 살펴보면
  - 1번째 시점에서의 손실 $J^{(i)}$에 대한 $h^{(1)}$의 gradient

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

<- chain rule에 의해 gradient가 곱해지면서 이 값이 커지는가 작아지는가!

  - t번째 hidden state에 대한 t-1번째 hidden state의 gradient (chain rule에 의해) :

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

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

# Unit 01 | Vanishing gradient problem

## 1. Vanishing gradient intuition

- 수식으로 살펴보면
  - 1번째 시점에서의 손실 $J^{(i)}$에 대한 $h^{(1)}$의 gradient

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

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

  - Norm의 성질에 의해, 다음 부등식 성립! => $W_h$의 L2 norm은 $W_h$의 가장 큰 고유값(eigenvalue)이다!

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

$$\Rightarrow \|W_h\|_2 = \sqrt{\lambda_{max}}$$

Norm의 성질: https://imgur.com/zcuKkv2

# Unit 01 | Vanishing gradient problem

## 1. Vanishing gradient intuition

- 수식으로 살펴보면
  - 결국!

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

<- RNN 역전파시 chain rule에 의해 계속 곱해지는데,
이 값의 L2 norm은 $W_h$의 L2 norm 크기에 달려있다!

$\|W_h\|_2 = \sqrt{\lambda_{max}}$ → 이 값이 1보다 작다면 gradient $\left\| \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} \right\|$ 매우 작아질 것이다!

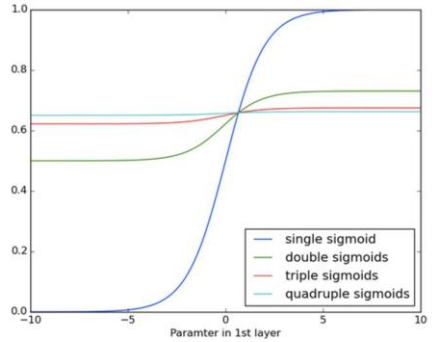
⇒ Weight matrix의 가장 큰 eigen value가 1보다 작으면

1보다 작은 값이 계속해서 곱해지는 것이기 때문에 gradient가 빠르게 줄어든다!

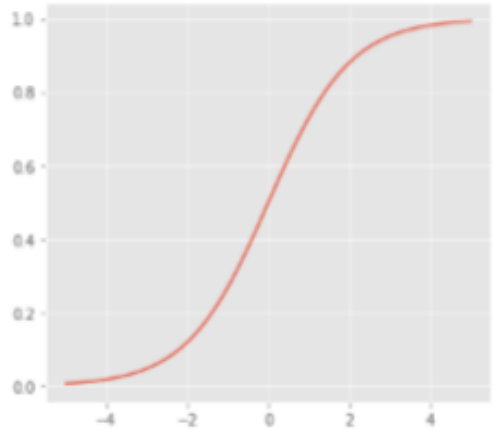
+ 가장 큰 eigenvalue가 1보다 크다면 exploding gradient 문제가 발생하겠죠?

# Unit 01 | Vanishing gradient problem

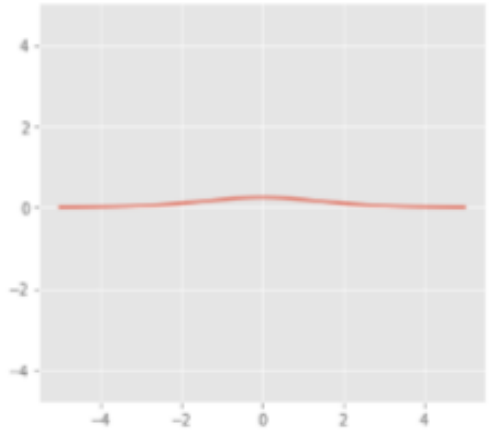
## 1. Vanishing gradient intuition




- Activation function 도 vanishing gradient 문제에 일부 영향을 끼칠 수 있어요!
  - 결국!
  - Sigmoid function



$$\sigma(x) = 1/(1+e^{-x})$$



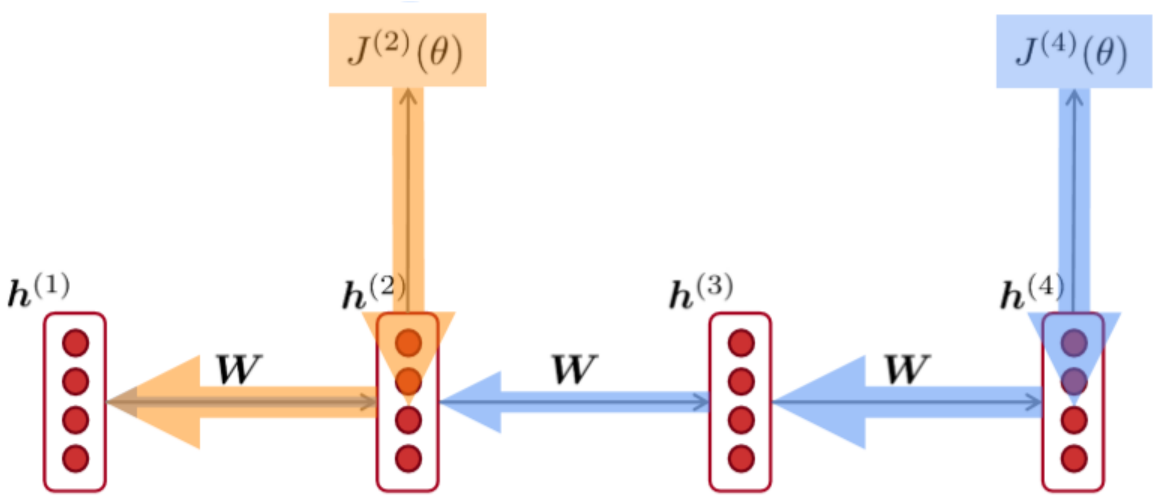
$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

- 입력값의 절대값이 크게 되면 0이나 1로 수렴하게 되어, Gradient 소멸한다
- (RNN의 역전파 과정에서 계속 곱해지는 activation function의 gradien값이 0이 되기 때문에 역전파에서 0이 곱해지기 때문!)

## 2. Why is vanishing gradient a problem?

1) 멀리 떨어진 loss의 영향을 가까운 loss의 영향을 훨씬 못 받는다

=> weight는 long-term effects보다 near effects에 관해 update된다




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

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

$$= \frac{\partial J^{(1)}}{\partial h^{(1)}} + \frac{\partial J^{(2)}}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}}$$

$$+ \frac{\partial J^{(3)}}{\partial h^{(3)}} \cdot \frac{\partial h^{(3)}}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}}$$

$$+ \frac{\partial J^{(4)}}{\partial h^{(4)}} \cdot \frac{\partial h^{(4)}}{\partial h^{(3)}} \cdot \frac{\partial h^{(3)}}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}}$$

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

# Unit 01 | Vanishing gradient problem

## 2. Why is vanishing gradient a problem?

2) Gradient는 미래에 과거가 얼마나 영향을 미치는지에 대한 척도

Gradient 값이 너무 작아져서 0에 가까워지는 경우,

1. 실제로 dependency가 존재하지 않아서 0에 가까운 값으로 나오는 경우
2. Dependency가 존재하지만, 파라미터 값을 잘못 설정해서 0에 가까운 값으로 나오는 경우

⇒ vanishing gradient문제가 발생하면 1번 경우인지, 2번 경우인지 우리는 알 수 X

## 3. Effect of vanishing gradient on RNN-LM

- LM task에 적용하는 경우, 멀리 떨어진 단어들 사이의 dependency를 학습하지 못하는 문제 발생

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*___

- Vanishing gradient 문제로 dependency를 올바르게 밝혀낼 수 X

=> 마지막에 올 단어 ticket 유추하기 쉽지 X

## 3. Effect of vanishing gradient on RNN-LM

- LM task에 적용하는 경우, 멀리 떨어진 단어들 사이의 dependency를 학습하지 못하는 문제 발생

Ex2)

- **LM task:** *The writer of the books* ___ → *is* / *are*
- **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, books와의 dependency 학습 -> 'are'이라는 잘못된 결과
- Syntactic recency보다 sequential recency를 더 빠르게 학습 -> 잘못된 결과

# Unit 01 | Vanishing gradient problem

## + Exploding gradient problem

- 출력의 길이가 길수록, 기울기가 너무 커지는 문제 !

  -> learning rate를 조절하여 경사하강법의 업데이트 속도를 조절해야 한다

- 너무 큰 learning rate를 사용하면 한 번의 업데이트 step의 크기가 너무 커져 => 잘못된 방향으로 학습 & 발산

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

# Unit 01 | Vanishing gradient problem

## + Exploding gradient problem

- **해결 방법?**
  - 그냥 learning rate로 아주 작은 값을 취해버리면 해결되지 않나?
  - 작은 learning rate를 사용할 경우,

    traning 속도 매우 느려짐…

    길이는 계속 변하니까,,그 때마다 learning rate를 알맞게 조절하는 건 매우 어려운 일!

learning rate

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

gradient

=> **Gradient Clipping**

# Unit 01 | Vanishing gradient problem

## + Exploding gradient problem

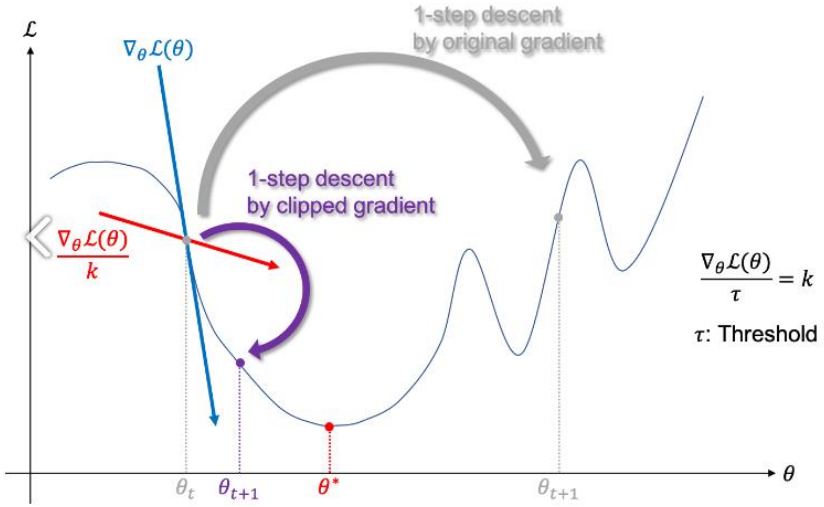
- Gradient Clipping
  - gradient가 일정 thresold를 넘어가면 gradient L2norm으로 나눠주는 방식
  - threshold : gradient가 가질 수 있는 최대 L2norm (hyperparameter)

⇒ 학습의 발산 방지

⇒ 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** $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
**end if**



# Unit 01 | Vanishing gradient problem

## + Is vanishing/exploding gradient just a RNN problem?

- 당연히 RNN만의 문제가 아니다! Feed-Forward, convolutional 포함한 모든 NN의 문제
  - Gradient vanishing 문제 -> backpropagation 때 점점 gradient 작아질 것
  - Lower layer는 업데이트가 잘 되지 X -> train하기 어려워짐

- <Solution> : gradient가 사라지지 않도록 direct connection 추가

  1) Residual connections "ResNet"

     = skip-connections

     - Input x에 convolutional layer를 지나고 나온

     결과를 더해줌으로써 과거의 내용을 기억할 수 있도록 함

     - 기존에 학습한 정보 보존 + 추가적으로 학습하는 정보

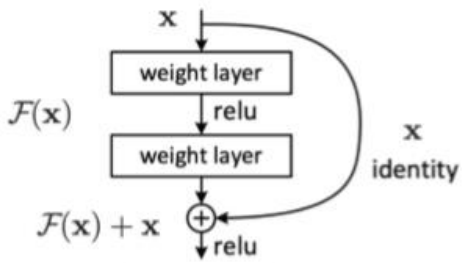


Figure 2. Residual learning: a building block.

# Unit 01 | Vanishing gradient problem

## + Is vanishing/exploding gradient just a RNN problem?

- <Solution> : gradient가 사라지지 않도록 direct connection 추가

  2) Dense connections "DenseNet"

  - 이전 layer들의 feature map을 계속해서 다음 layer의 입력과 연결하는 방식
  - ResNet : feature map끼리 '더하기'를 해주는 방식
  - DenseNet : feature map끼리 Concatenation 시키는 방식

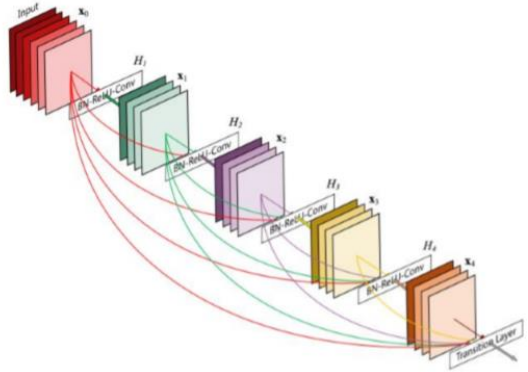


**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

  3) Highway connections "HighwayNet" https://arxiv.org/pdf/1505.00387.pdf

  - Residual connections과 비슷
  - T : transform gate, C : carry gate => output이 input에 대해 얼마나 변환되고 옮겨졌는지 표현

# Unit 01 | Vanishing gradient problem

## + Is vanishing/exploding gradient just a RNN problem?

- <Solution> : gradient가 사라지지 않도록 direct connection 추가

  2) Dense connections "DenseNet"

  - 이전 layer들의 feature map을 계속해서 다음 layer와 입력으로 연결하는 방식
  - ResNet : feature map끼리 '더하기'를 해주는 방식
  - DenseNet : feature map끼리 Concatenation 시키는 방식

  3) Highway connections "HighwayNet" https://arxiv.org/pdf/1505.00387.pdf

  - Residual connections과 비슷
  - T : transform gate, C : carry gate => output이 input에 대해 얼마나 변환되고 옮겨졌는지 표현

**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Vanishing gradient문제는 굉장히 general한 문제!

But! RNN같은 weight matrix를 반복적으로 곱하기 때문에 특히 더 불안정하다!

=> 이를 해결한 모델은~?

# Contents

# Unit 02 | LSTM

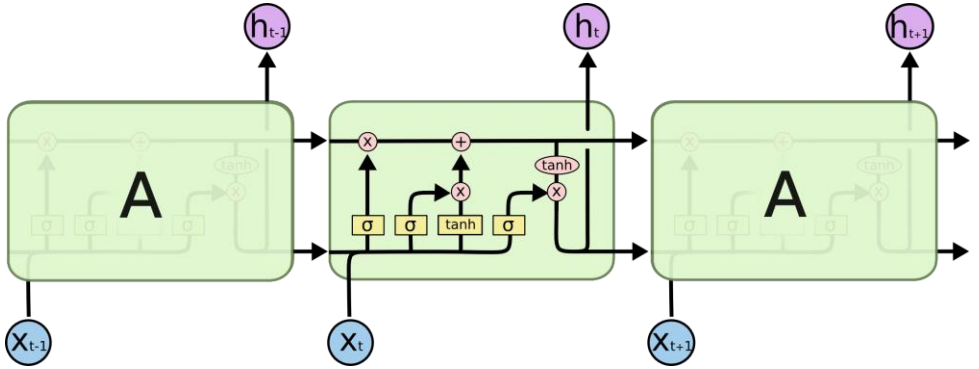
## * LSTM

- Vanilla RNN 의 장기 의존성 문제를 해결하기 위해
- RNN에서 메모리를 분리하여 우리가 나중에 사용할 수 있게 따로 정보를 저장함으로써

  한참 전의 데이터도 함께 고려하여 출력을 만들어보자! => LSTM (Long Short-Term Memory)

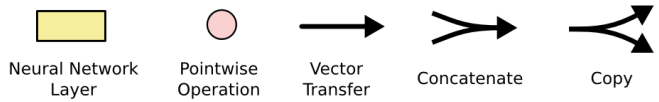
- LSTM의 핵심 아이디어
  - 이전 단계의 정보를 memory cell에 저장해서 흘려 보내는 것!
  - 현재 시점의 정보를 바탕으로 과거의 내용을 얼마나 잊을지 곱해주고,

  그 결과에 현재 정보를 더해서 다음 시점으로 정보를 전달!



# Unit 02 | LSTM

## * LSTM

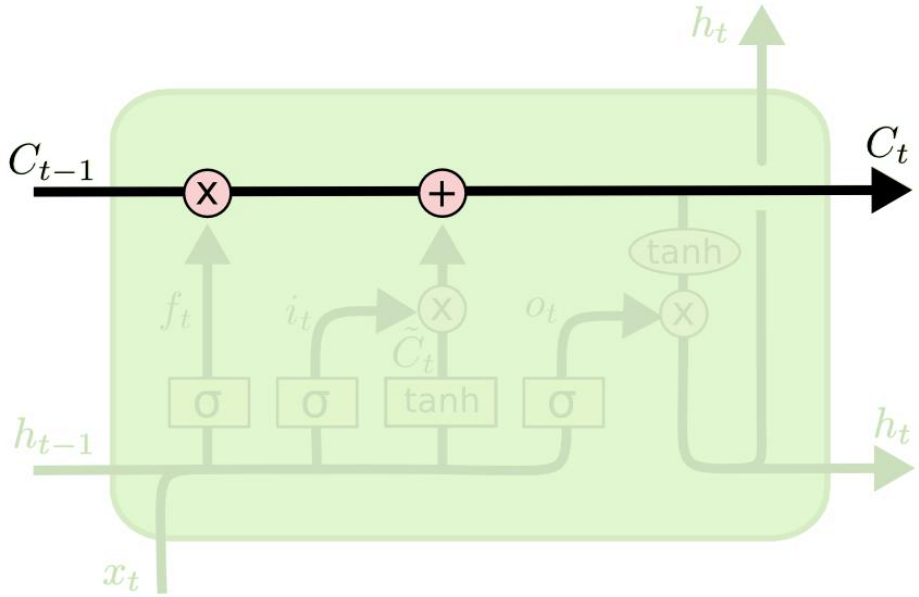


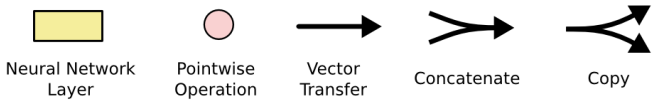

- Cell state : LSTM의 핵심적인 부분

  - 단순한 연산(곱셈, 덧셈)을 거쳐 LSTM Unit 통과
  - Cell state를 통해 이전 정보는 큰 변화 없이 다음 단계로 전달
  - Input gate, forget gate, output gate

    => 세 개의 gate들을 이용하여 정보의 반영 여부를 결정

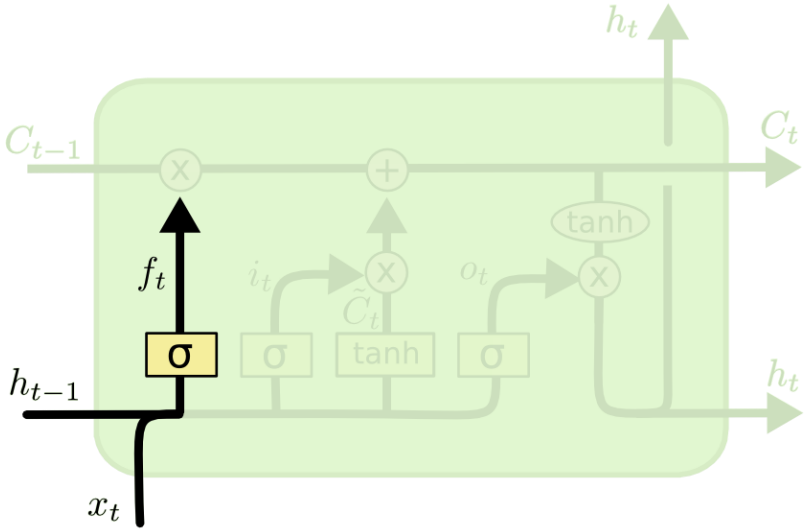
    => cell로부터 정보를 erase, write, read
  - 무엇을 쓰고 -> Input gate
  - 무엇을 읽고 -> output gate
  - 무엇을 잊을 것인가 -> forget gate






## * LSTM 단계

### 1) Forget gate layer

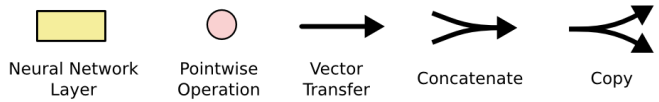



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

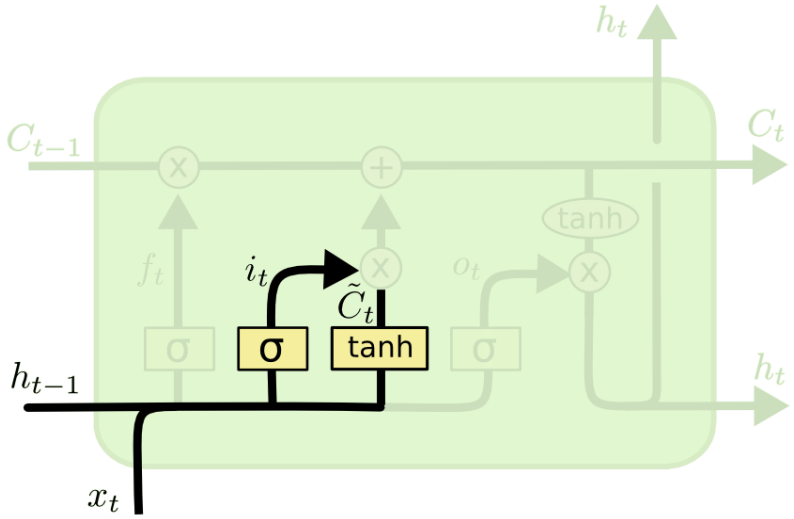
- 어떤 정보를 반영할지 결정하는 게이트
- $C_{t-1}$에서 불필요한 정보 지우는 역할!
- Sigmoid layer를 통해 이루어짐

  => 0~1 사이 값으로 변환
- 0: 이전 과거의 기억 사라짐
- 1: 이전 과거의 기억 보존

## * LSTM 단계



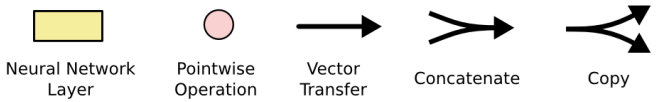

### 2) input gate layer




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

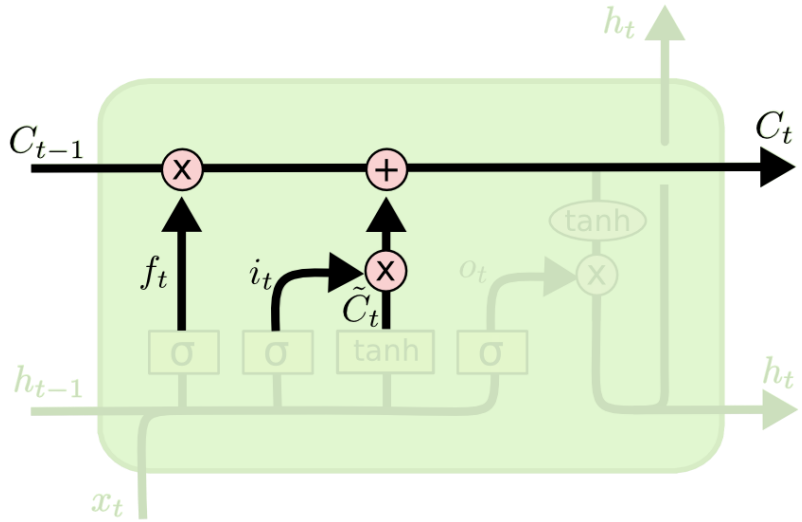
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- 새로운 정보가 Cell state에 저장이 될지 결정하는 게이트
- $C_{t-1}$에 새로운 input $x_t$ , $h_{t-1}$ 보고 중요한 정보를 넣는 역할!
- 2개의 layer
  - Sigmoid layer => 'input gate"
    : 어떤 값을 업데이트할지 결정
  - Tanh layer => 'update gate'
    : cell state에 더해질 $\tilde{C}_t$라는 벡터 만듦
    (vector of new candidate values)




## * LSTM 단계

### 3) Update Cell state

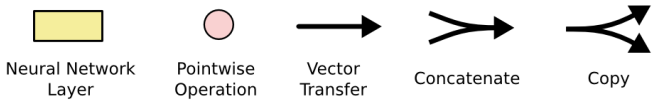



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

과거 현재

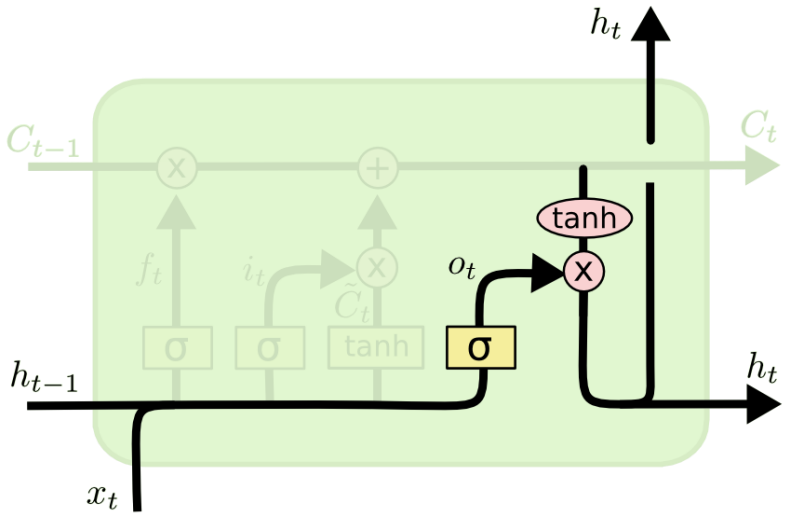
- 과거의 정보는 삭제될 것인지, 유지될 것인지 -> $f_t$
- 현재의 input값이 반영 되는지, 안되는지 -> $i_t$

- 최종적으로 더해져서 $C_{t-1}$가 $C_t$로 업데이트!




## * LSTM 단계

### 4) Output Gate Layer




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

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

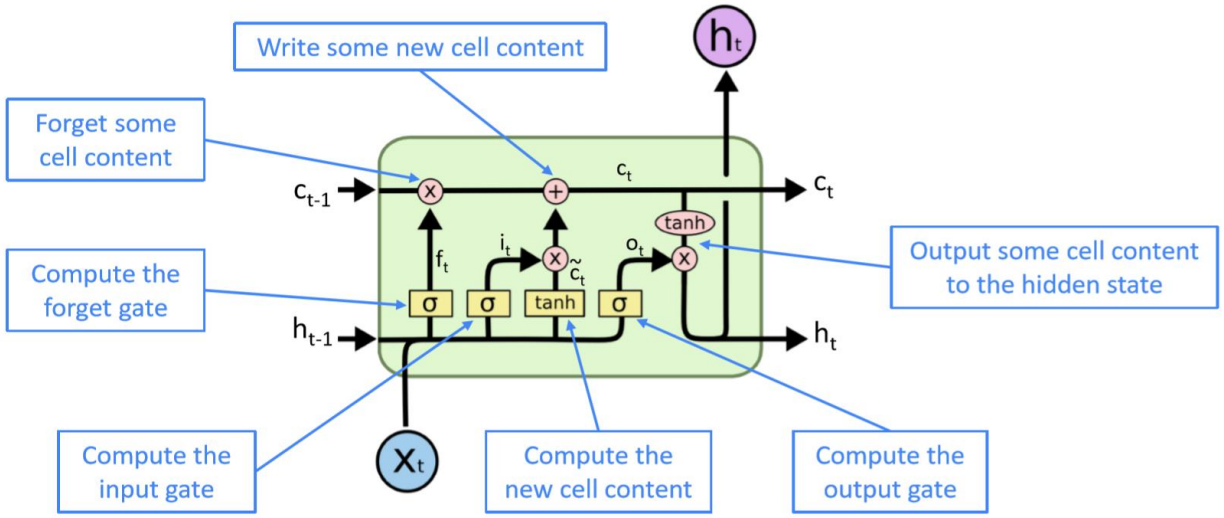
- RNN처럼 각 state마다 출력값을 반환!
- Output : $o_t$(cell state를 바탕으로 가공된 값)

  => sigmoid layer – 출력할 Cell state의 부분 결정
- Tanh(cell state) (-1~1로 출력) x ouput gate에서 나온 값 = ht

  => 출력값으로 나감 + 다음 state의 input값으로 들어감

# Unit 02 | LSTM

## * LSTM




- 모든 state와 gate는 길이가 n인 벡터
- 모든 gate는 sigmoid를 통과해서 0과 1 사이의 숫자
- 전 hidden state와 현재 input context 기반으로 계산되므로 dynamic!

## * LSTM

Cell state & hidden state

- Gate들 : cell로부터 어떤 정보를 erase, write, read 할 지 결정!

- Cell state $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
  - erase "forget" some content from last cell state
  - Write "input" some new cell content

- Hidden state $h_t = o_t * \tanh(C_t)$
  - Read "output" some content from the cell

## * How does LSTM solve vanishing gradients?

Ex) forget gate가 1로 설정되고, input gate가 0으로 설정되면, cell의 정보가 완전하게 보존될 것!

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

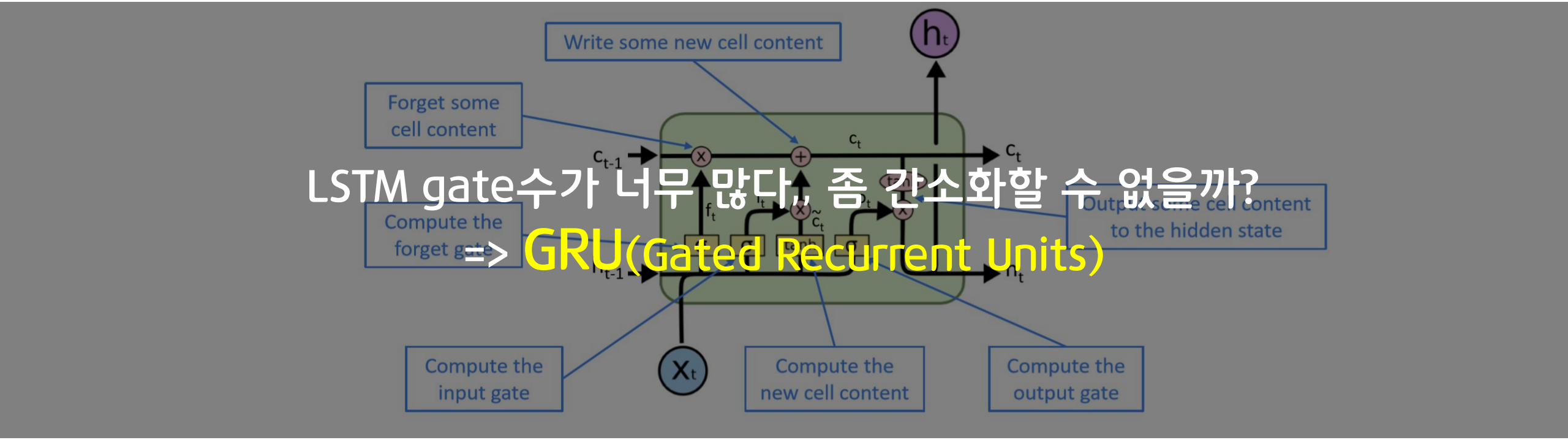
But! Vanilla RNN은 반복적으로 똑같은 matrix $W_h$를 학습하기 때문에 hidden state의 정보를 보존하고 있기 어려움

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

강의에서) LSTM도 vanishing / exploding gradient 문제가 없다고 보장할 수는 없다!
하지만, 확실히 이 문제들을 피하기 쉬운 것은 맞다!

# Contents

# Unit 03 | GRU

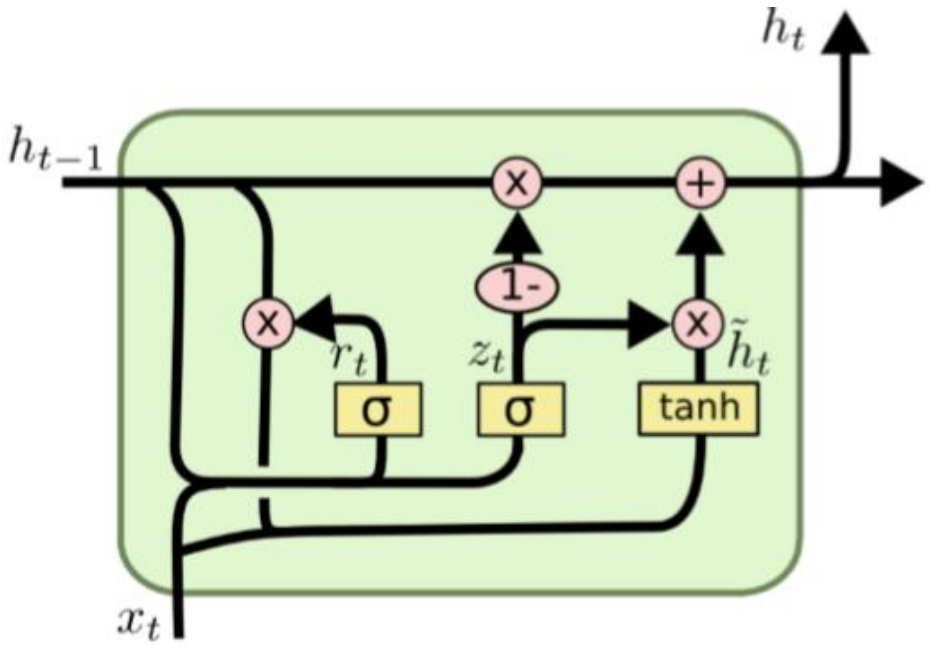


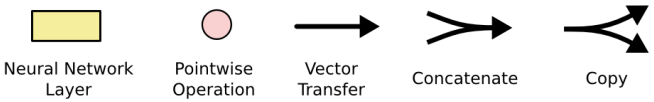

LSTM gate수가 너무 많다,, 좀 간소화할 수 없을까?

=> GRU(Gated Recurrent Units)

# Unit 03 | GRU

## * Gated Recurrent Units (GRU)



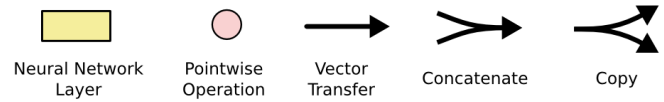



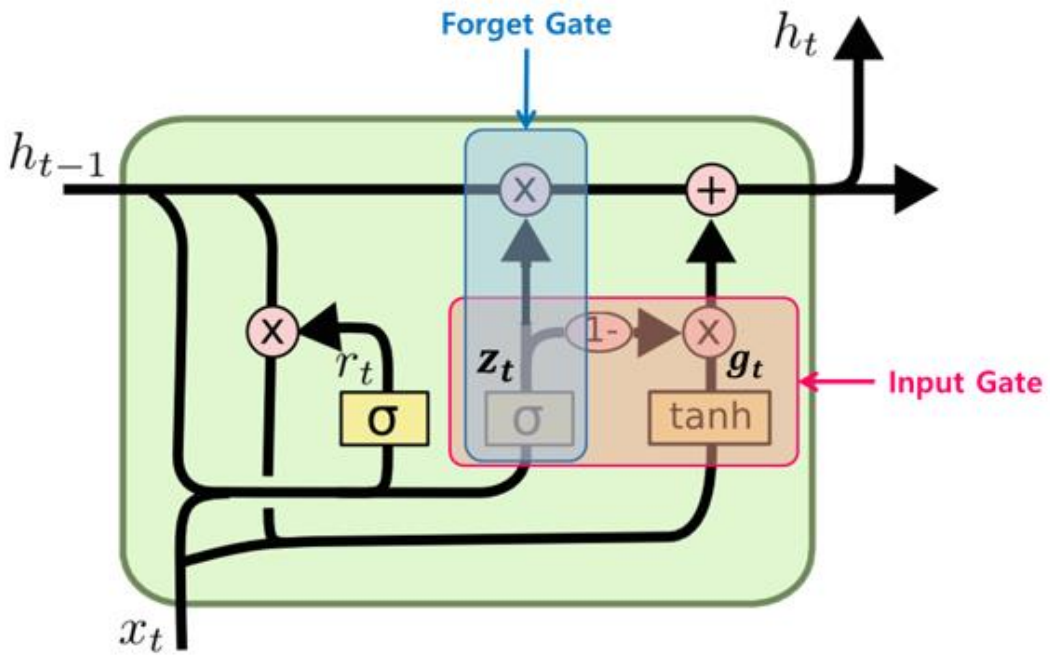

- LSTM의 강점을 가져오되,
  불필요한 복잡성을 제거한 모델

- 특징
  - 매 timestep t 마다 input $x_t$와
  - hidden state $h_t$는 있지만,
  - Cell state는 존재하지 X
  - Gate들을 통해서 정보의 흐름 통제
  - 두 개의 gate : update gate, reset gate

# Unit 03 | GRU

## * Gated Recurrent Units (GRU)



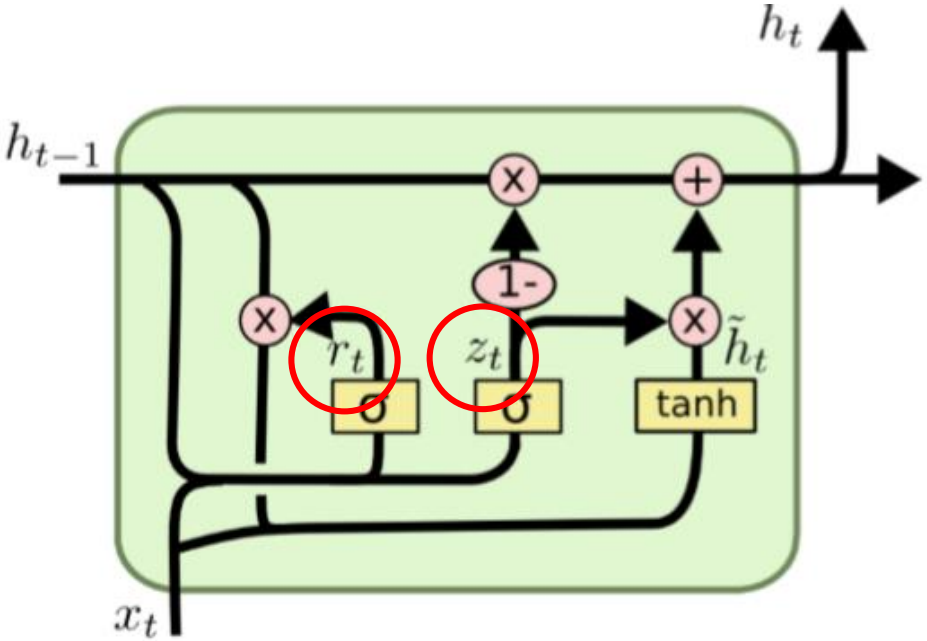

두 개의 gate : update gate, reset gate

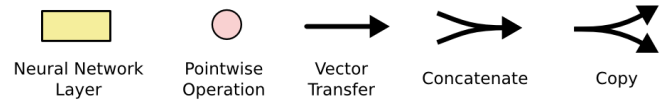
- Update gate
  - Forget gate + input gate
- Reset gate 추가

- Cell state + hidden state => hidden state

# Unit 03 | GRU

## * Gated Recurrent Units (GRU)




두 개의 gate : update gate(z), reset gate(r)

- Reset gate :

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

과거의 정보를 적당히 리셋시키는 것이 목적

- Update gate:

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

Forget gate + input gate

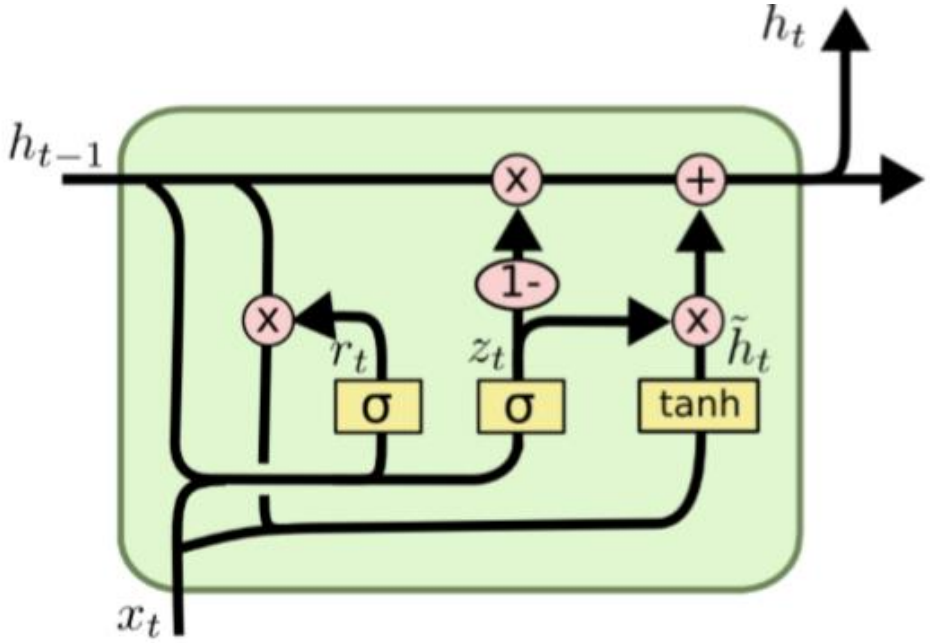
과거와 현재의 정보의 최신화 비율 결정

$z_t$ : 현재상태를 얼마나 반영할지(input gate)

$1 - z_t$: 이전 상태를 얼마나 잊을지 (forget gate)

# Unit 03 | GRU

## * Gated Recurrent Units (GRU) 단계




$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

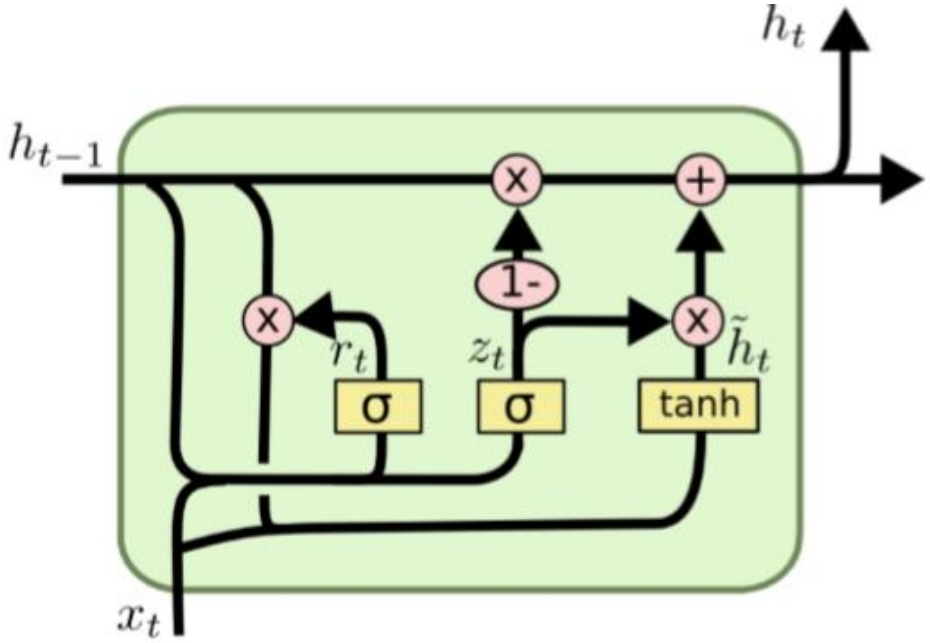
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

1. Reset gate($r_t$)를 계산해서 임시 $h_t$를 만듦
2. Update gate ($z_t$)를 통해 $h_{t-1}$와 $\tilde{h}_t$간의 비율 결정
3. $z_t$ 를 통해 최종 $h_t$ 계산

# Unit 03 | GRU

## * Gated Recurrent Units (GRU)




$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

- Reset gate($r_t$) -> 이전 hidden state에서 유용한 정보 선택 + 현재 input과 함께 새로운 hidden content로 계산
- Update gate($z_t$) -> 이전 hidden state& new hidden content balance 조절
- Ex) $z_t$ = 0 -> hidden state 계속 보존

  ⇒ gradient vanishing 문제 해결!!

# Unit 03 | GRU

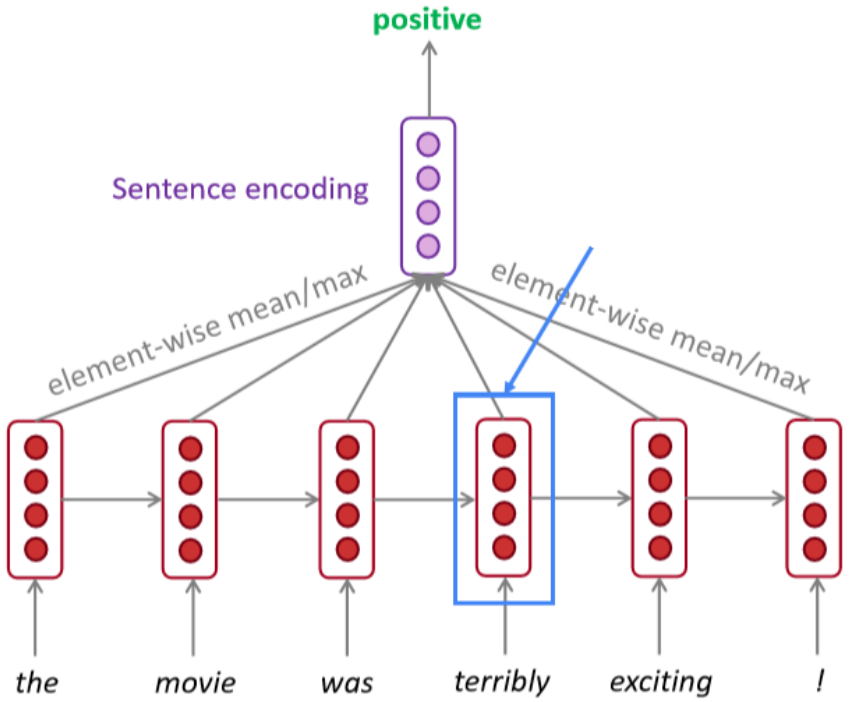
## + LSTM vs. GRU

- 둘 다 오래 기억하기 좋음
- 가장 큰 차이점 : GRU 계산 더 빠르고, 파라미터 수 더 적음
- GRU는 파라미터 좀 더 적게 해서 학습하기 좋고, LSTM은 그냥 기본적으로 선택하기 좋다
  - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- LSTM으로 시작해본 후 efficient함을 원하면 GRU를 시도 해보길 권한다

# Contents

# Unit 04 | More fancy RNN variants

## * Bidirectional RNNs



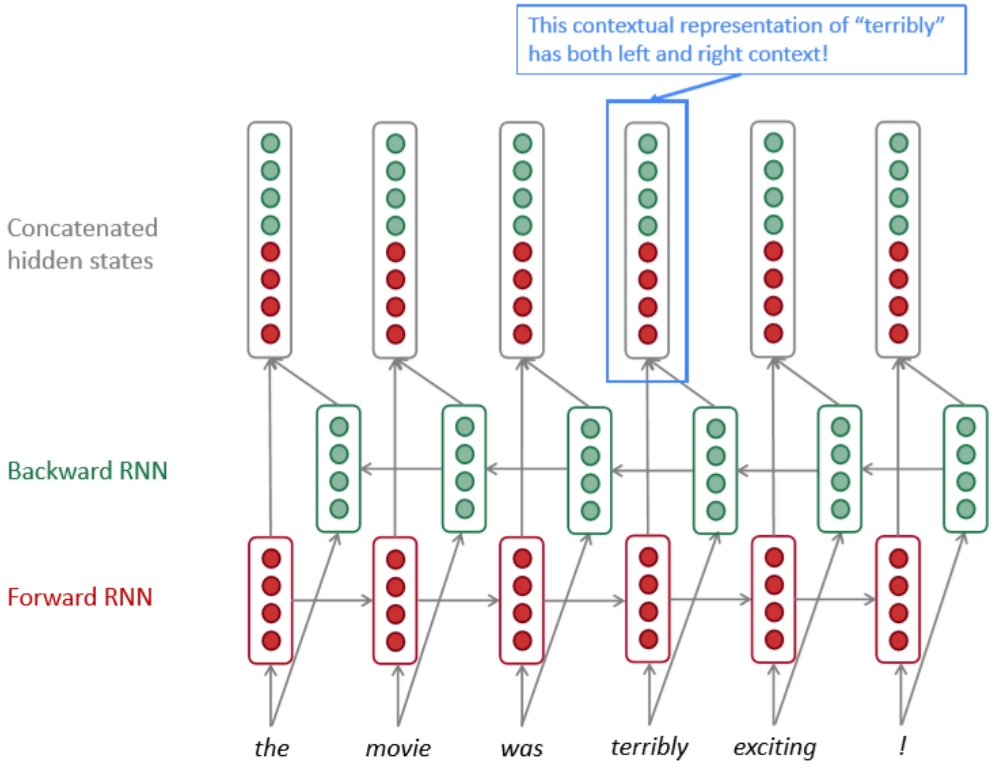

- Task : Sentiment Classification

Ex) RNN의 한 방향으로 흐르는 성질때문에 terribly를 표현할 때, this movie was만 고려한다는 것이 문제!
사실상 중요한 정보는 exciting,,

⇒ left, right 두 방향으로 모두 정보를 이용해서 representation 하기 위해 ! bidirectional RNN

# Unit 04 | More fancy RNN variants

## * Bidirectional RNNs – 구조



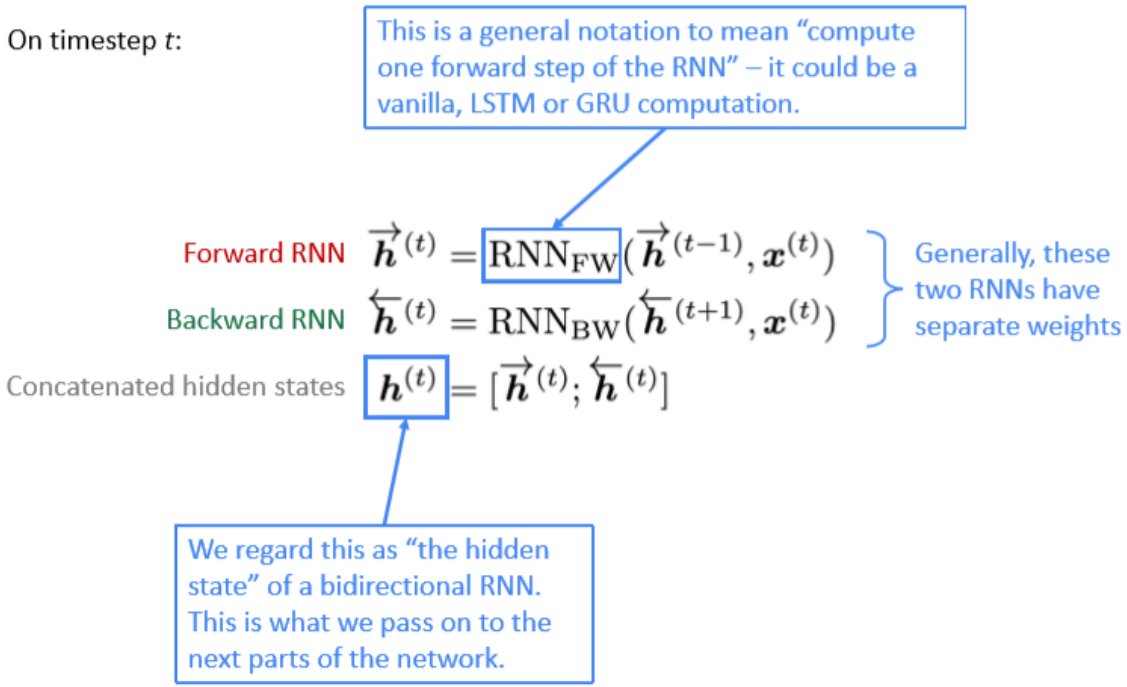

- Forward RNN : 정방향
- Backward RNN: 역방향

⇒ 두 RNN이 생성한 hidden state concatenate

⇒ 전체 모델의 hidden state로 사용!

# Unit 04 | More fancy RNN variants

## * Bidirectional RNNs – 구조




- Forward RNN : 정방향
- Backward RNN: 역방향

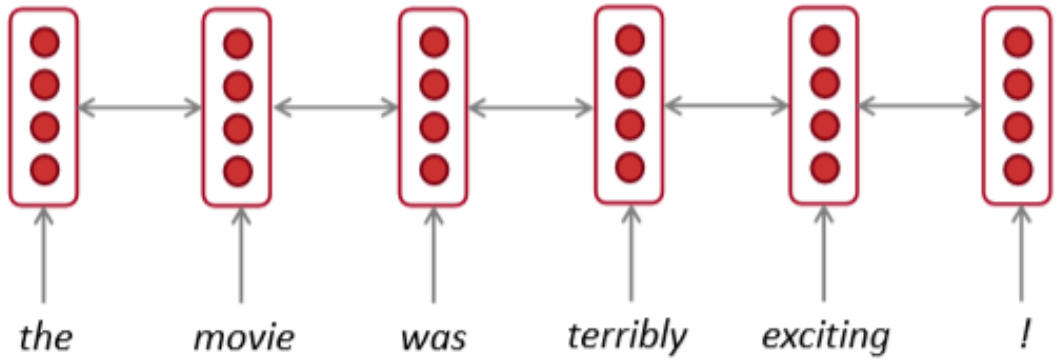
⇒ 두 RNN이 생성한 hidden state concatenate

⇒ 전체 모델의 hidden state로 사용!

- 여기서 RNN은 vanilla RNN, LSTM, GRU 다양한 종류 사용 가능 !

# Unit 04 | More fancy RNN variants

## * Bidirectional RNNs




- 전체 input data를 가지고 있는 경우 강력한 성능!
- But, 전체 data가 없다면 사용할 수 X!
  => 빈칸에 오는 단어를 예측해야 하는 Language Modeling에서 사용할 수 X
- Ex) BERT ( Bidirectional Encoder Representations from Transfers) => 나중에 배움☺

# Unit 04 | More fancy RNN variants

## + Bidirectional RNNs – training

https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf

- forward pass 계산 : 일반적인 RNNs과 같다
- Forward hidden layer와 Backward hidden layer에 input값을 반대 방향으로 집어넣고,
  output layer 값은 두 방향의 hidden layer에 모든 input이 적용한 후 계산
- Backward pass의 가중치 업데이트 할 때, 기본적인 RNNs와 같이 BPTT 이용.
- 단지, output layer에서 모든 시간에 대해 에러값을 먼저 계산하고,
  이를 forward hidden layer와 backward hidden layer에 반대바향으로 전달한다는 점이 차이점!

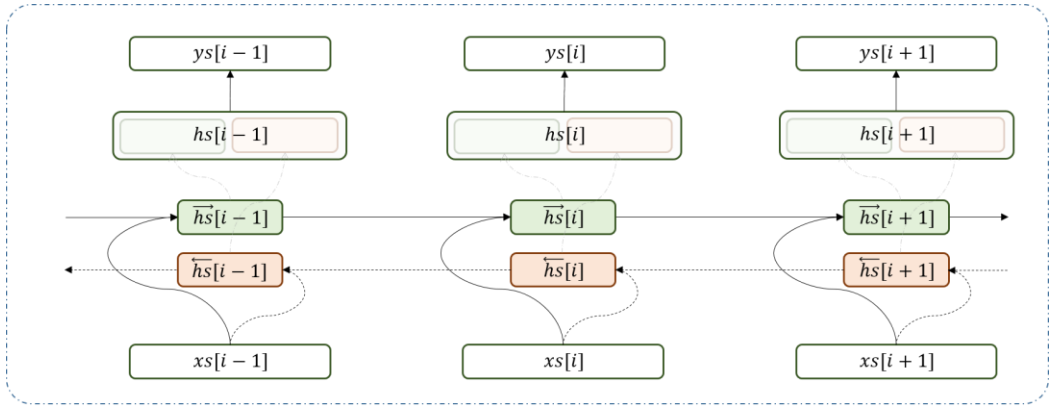
**Algorithm 1** BRNNs Forward Pass

1: **for** t=1 to T **do** Do forward pass for the forward hidden layer, storing activations at each timestep
2: **end for**
3: **for** t=T to 1 **do**
Do forward pass for the backward hidden layer, storing activations at each timestep
4: **end for**
5: **for** t=1 to T **do**
Do forward pass for the output layer, using the stored activations from both hidden layers
6: **end for**

**Algorithm 1** BRNNs Backward Pass

1: **for** t=T to 1 **do**
Do BPTT backward pass for the forward hidden layer, using the stored $\delta$ terms from the output layer
2: **end for**
3: **for** t=T to 1 **do**
Do BPTT backward pass for the forward hidden layer, using the stored $\delta$ terms from the output layer
4: **end for**
5: **for** t=1 to T **do**
Do BPTT backward pass for the backward hidden layer, using the stored $\delta$ terms from the output layer
6: **end for**

# + Bidirectional RNNs - backpropagation




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

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

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

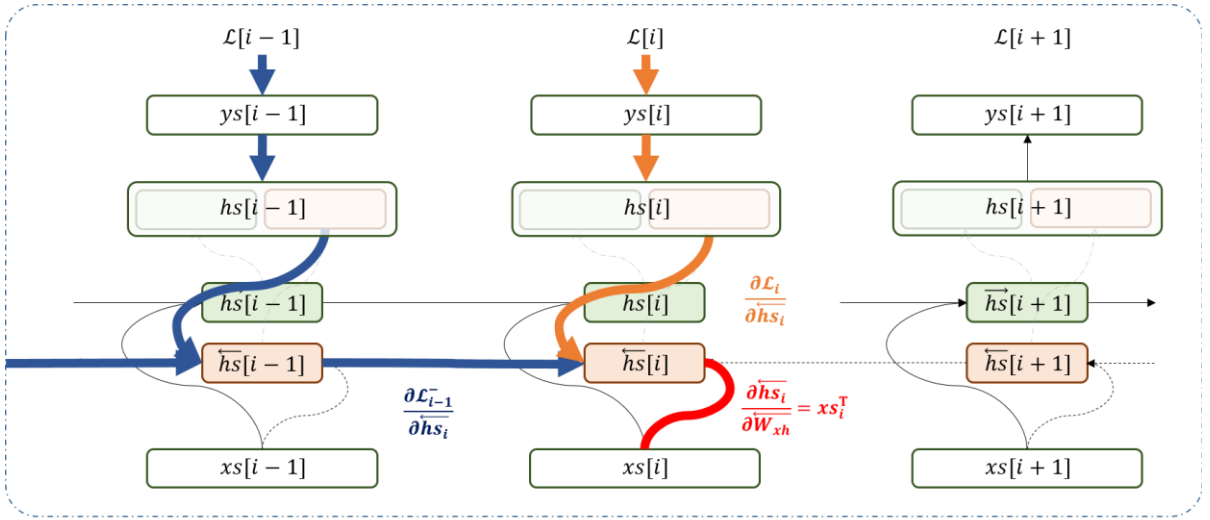
*stacked!*

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

*Output Size = Y*

*Hidden Size = H*

*Vocabulary size = V*



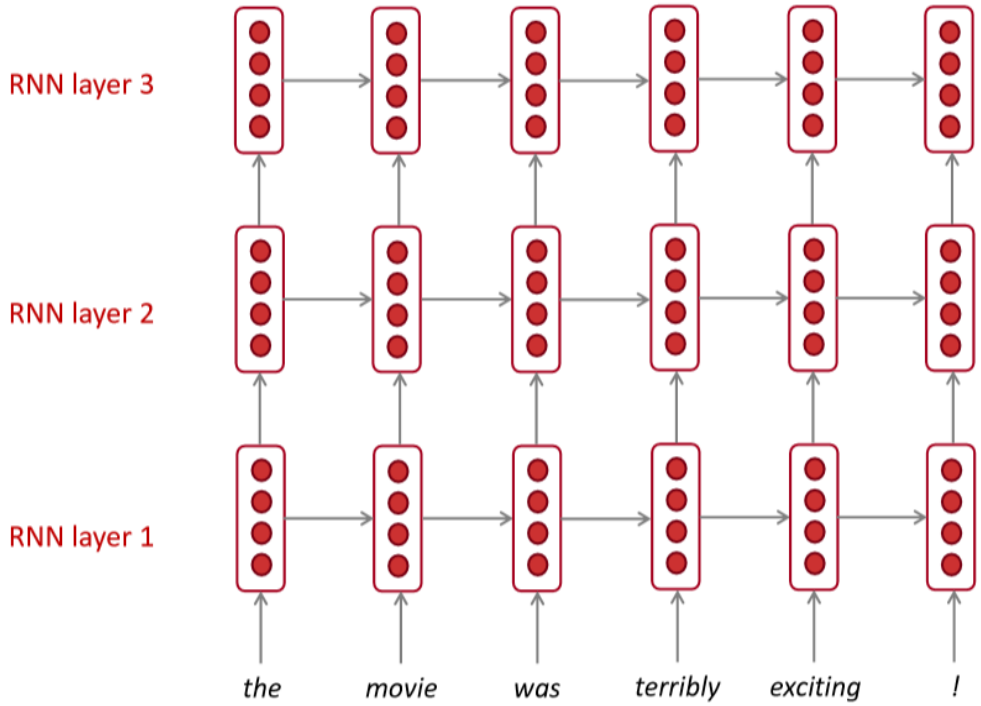

$$let:\ \mathcal{L} = \Sigma_{i=0}^{T}\mathcal{L}_i \qquad let:\ \mathcal{L}_k^{-} = \Sigma_{i=0}^{k}\mathcal{L}_i$$

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

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

# Unit 04 | More fancy RNN variants

## * Multi-layer RNNs (= stacked RNNs)




- 여러 개의 층으로 사용
- 더 복잡한 특성을 학습할 수 0
- lower RNNs - lower-level features
- higher RNNs - higher-level features
- 보통 2개에서 4개 정도의 layer 쌓음

cf) transformer based network: 24개의 layer ( + skip connection 트릭,,)


들어주셔서 감사합니다.

# Reference

- Stanford CS224n Lecture 7 강의 & 강의자료
- 13-14기 정규세션 13기 이예지님 모델심화2 강의자료
- 기타 블로그
- https://blog.naver.com/dmsquf3015/222067819691
- https://excelsior-cjh.tistory.com/89
- https://ratsgo.github.io/deep%20learning/2017/10/10/RNNsty/
- https://jeongukjae.github.io/posts/cs224n-lecture-7-vanishing-gradients-fancy-rnns/
- https://yjjo.tistory.com/18?category=881892