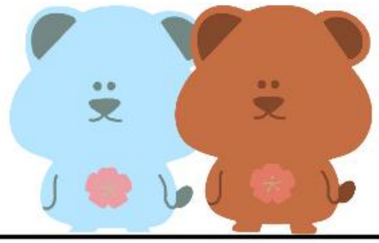
RNNLM, Vanishing Gradient and LSTM(Long-Short Term Memory)

문예지 송혜준

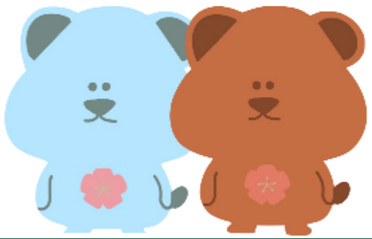
EWHA EURON

목차



EWHA EURON

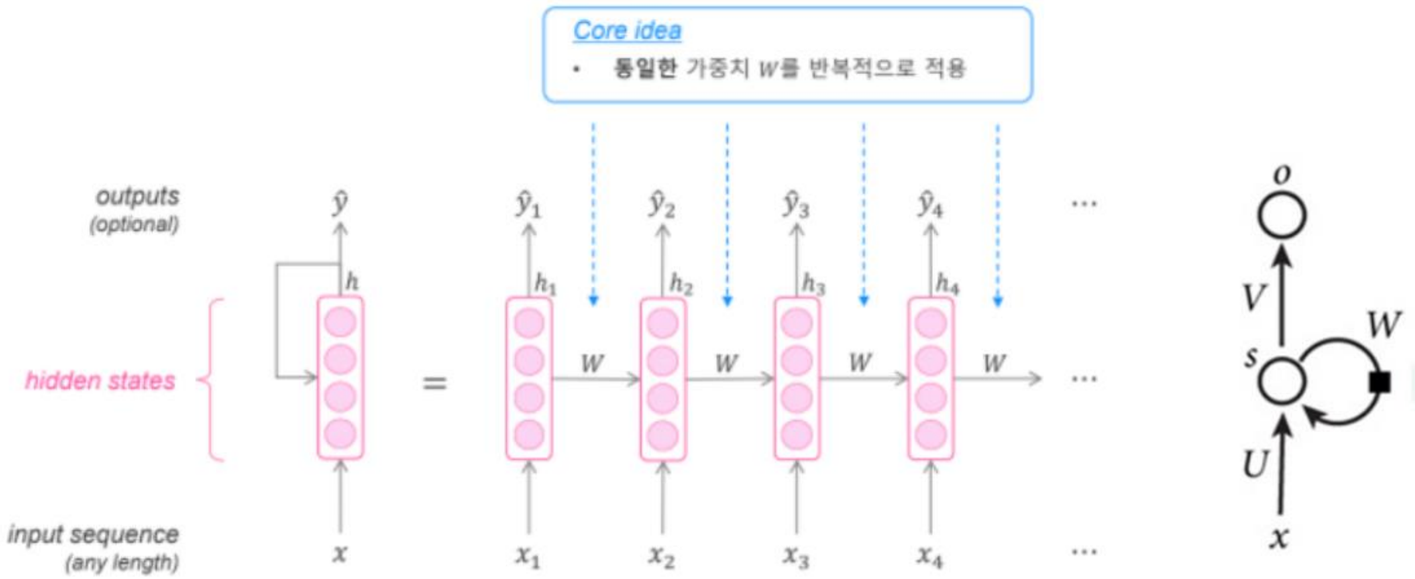
#01 RNN Language Model



EWHA EURON

#01 RNN Language Model

#1 core idea



동일한 가중치를 반복적으로 적용하자!

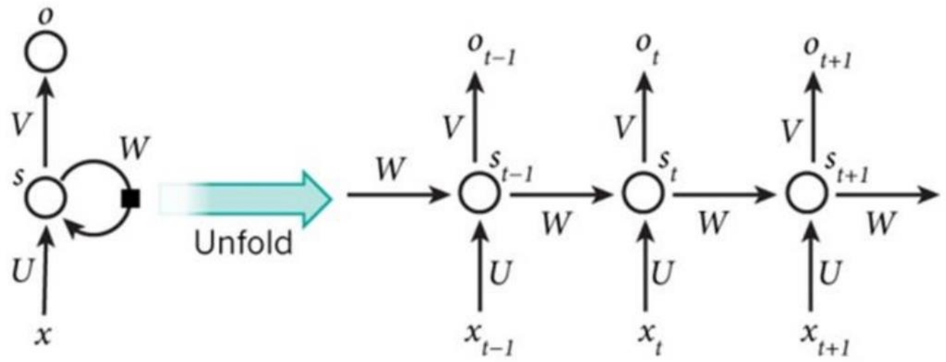
#01 RNN Language Model

#1 기본적인 아이디어 : 순차적인 정보를 처리하는 것

기존의 신경망 구조 -> 모든 입력이 각각 독립적이라고 가정

RNN -> 이전의 계산 결과에 영향을 받음 (현재까지 계산된 결과에 대한 메모리 정보를 갖고 있음

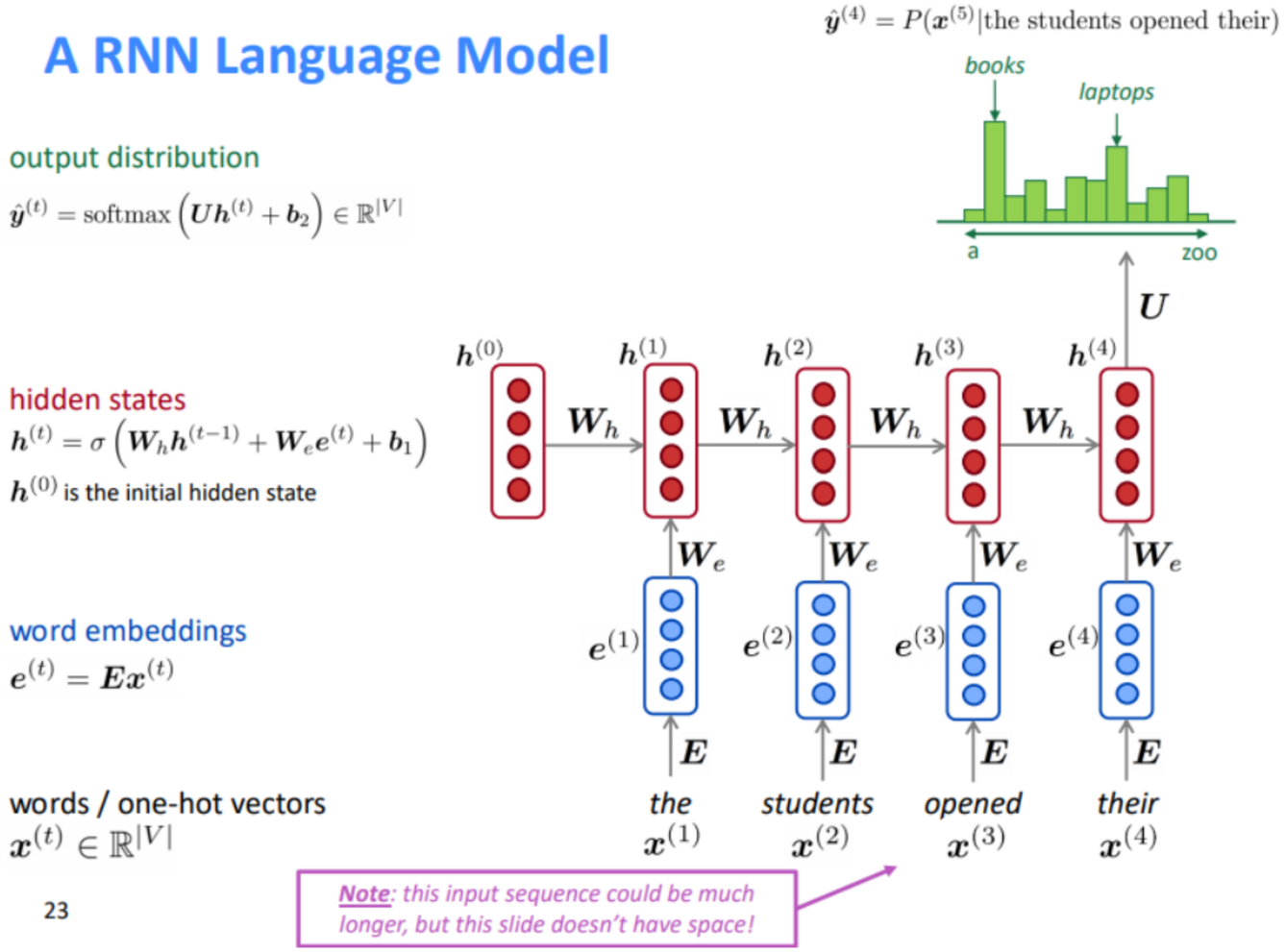
output: 추측된 단어들의 sequence



input: 단어들의 sequence

네트워크를 학습할 때에는 시간 스텝에서의 출력 값이 실제로 다음 입력 단어가 되도록 $o_t = x_{t+1}$로 정해줌

#01 RNN Language Model



#1 단순한 순환 신경망 언어 모델의 아이디어

#2 순환 신경망 방정식을 사용하여
단어의 인코딩 변환

#3 이를 기반으로 다음 시간 단계에 대한
새로운 은닉 표현을 계산

#4 연속적인 시간 단계에 대해 반복

#5 단어에 대한 확률 분포 제공

#01 RNN Language Model

#1 RNN (Recurrent Neural Network)

장점
- Input length에 상관없이 다음 단어를 예측할 수 있음 (이론상)
- 먼 곳에 위치한 단어도 고려할 수 있어 context를 반영할 수 있음
- Input이 길어져도 model size가 증가하지 않음

단점
- 다음 단계로 진행하기 위해서는 이전 단계의 계산이 완료되어야 하므로 계산이 병렬적으로 진행되지 않아 느림
- 이론적으로는 먼 곳의 단어를 반영할 수 있지만 실제로는 vanishing gradient problem 등의 문제가 있어 context가 반영되지 않는 경우도 있다는 점

#01 RNN Language Model

#1 Training RNN

1) 각 단어를 RNN model에 input으로 주고, 모든 단계에서 예상되는 다음 단어를 계산

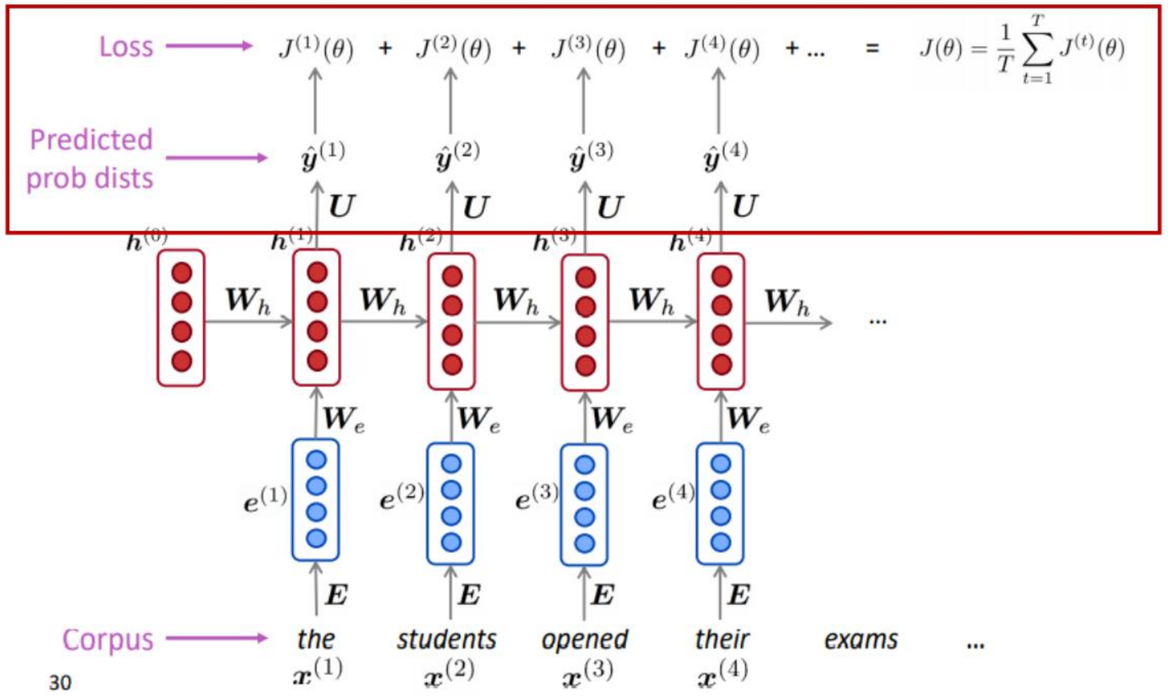
2) 모든 단계에서 예상되는 다음 단어와 실제 다음 단어의 차이의 cross-entropy를 통해 loss를 구함

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

3) 모든 단계에서의 loss의 평균을 통해 전체 loss를 구함

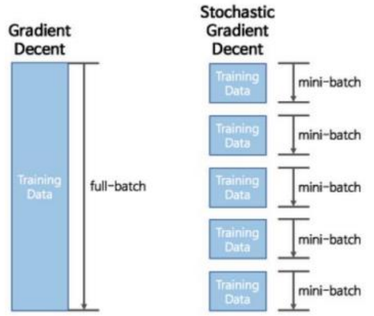
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

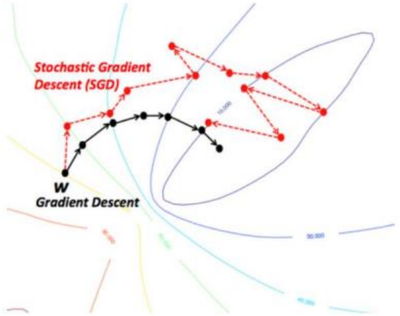
#01 RNN Language Model



30

- 다만, 실제 RNN-model을 학습할 때 이와 같이 학습하면 지나치게 많은 계산이 필요하므로, 실제로는 단어 단위가 아니라 문장 혹은 문서 단위로 입력을 줌
- 또, Stochastic Gradient Descent를 통해 optimize 하는 것도 좋은 방법

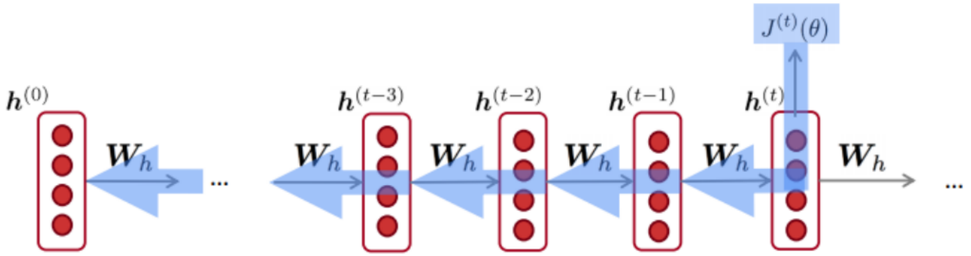




#01 RNN Language Model

#01 Backpropagation for RNNs

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



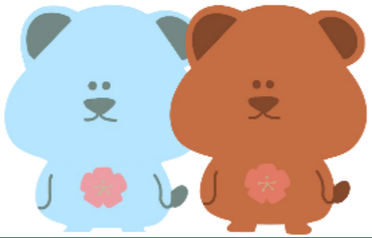
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

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"**

EWHA EURON

#02 Vanishing Gradient



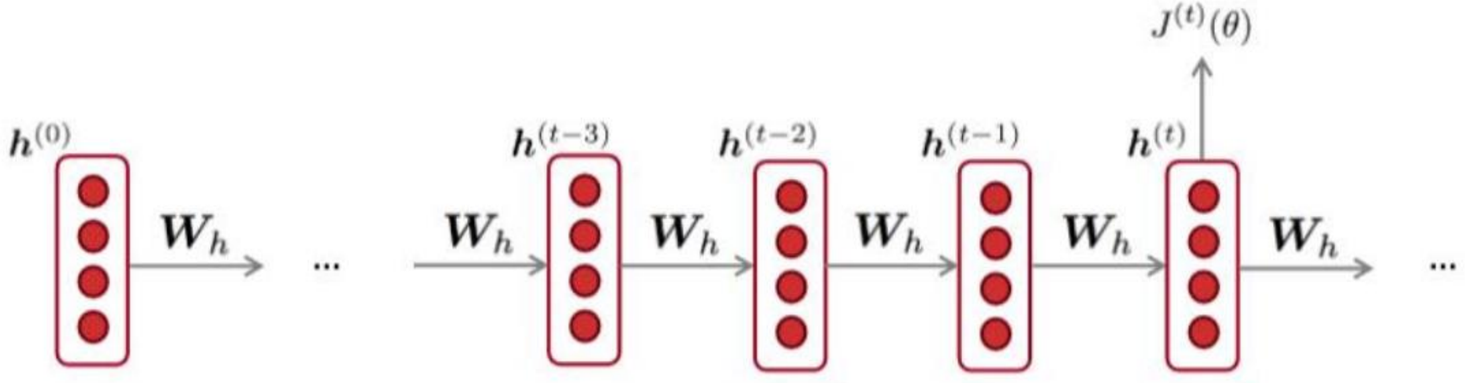
EWHA EURON

#02 Vanishing Gradient

#1 Vanishing gradient intuition

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

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



#02 Vanishing Gradient

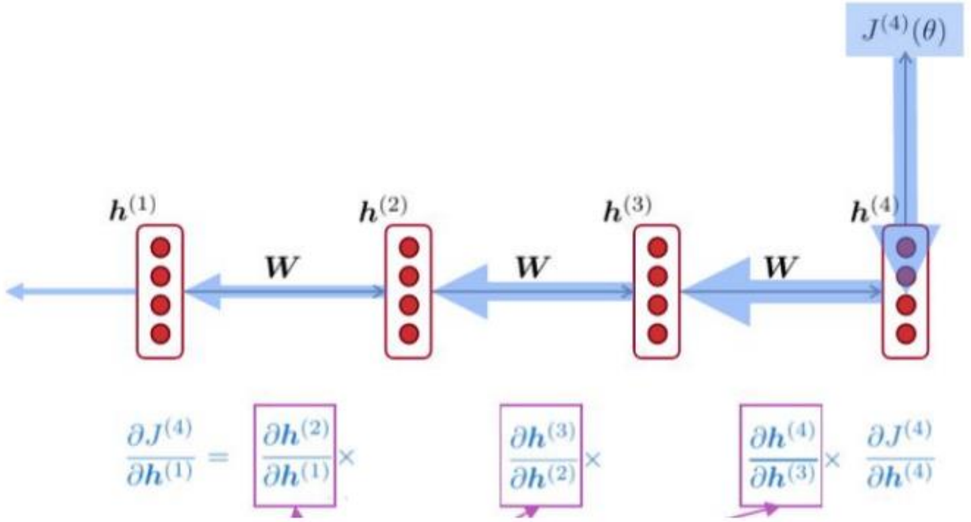
- 수식으로 살펴보면

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

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

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

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



#02 Vanishing Gradient

- 수식으로 살펴보면

  l번째 시점에서의 손실 J(i)에 대한 h(i)의 gradient

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

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

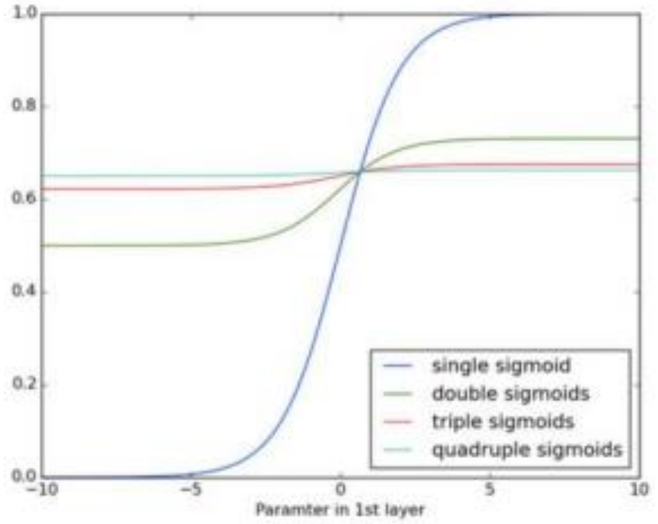
-> Weight matrix의 가장 큰 eigen value가 1보다 작으면
1보다 작은 값이 계속해서 곱해지는 것이기 때문에 gradient가 빠르게 줄어든다

+ 가장 큰 eigenvalue가 1보다 크다면 exploding gradient 문제가 발생

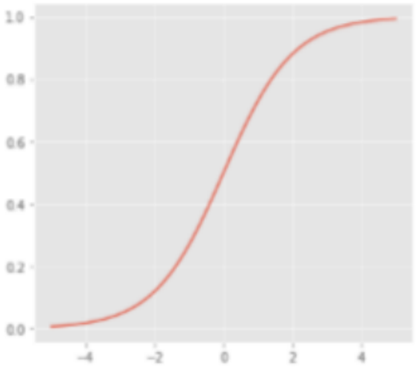
#02 Vanishing Gradient

#01 Activation function 도 vanishing gradient 문제에 일부 영향을 끼칠 수 있음

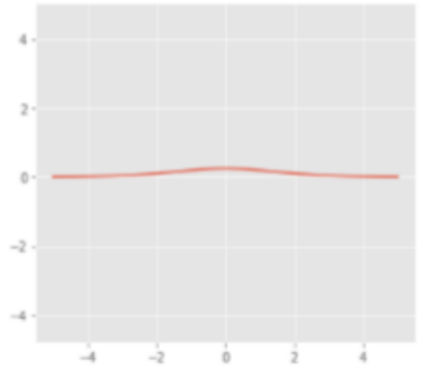
- 입력값의 절대값이 크게 되면 0이나 1로 수렴하게 되어, Gradient 소멸
- (RNN의 역전파 과정에서 계속 곱해지는 activation function의 gradient값이 0이 되기 때문에 역전차에서 0이 곱해지기 때문



- Sigmoid function



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



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

#02 Vanishing Gradient

#01 Vanishing Gradient 문제점

1) 멀리 떨어진 loss의 영향을 가까운 loss의 영향을 훨씬 못 받는다
   -> weigh는 long-term effects보다 near effects에 관해 update된다

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

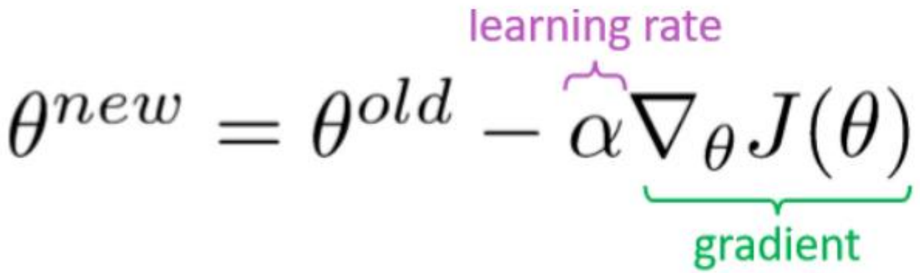
   -> Vanishing Gradient 문제 발생하면 1.2번 경우 중 알 수 없음

#02 Vanishing Gradient

#01 Effect of vanishing gradient on RNN-LM

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

2) 출력의 길이가 길수록, 기울기가 너무 커지는 문제 발생
- learning rate를 조절하여 경사하강법의 업데이트 속도를 조절해야 한다
(너무 큰 learning rate를 사용하면 한 번의 업데이트 step의 크기가 너무 커짐->잘못된 방향 학습)



#02 Vanishing Gradient

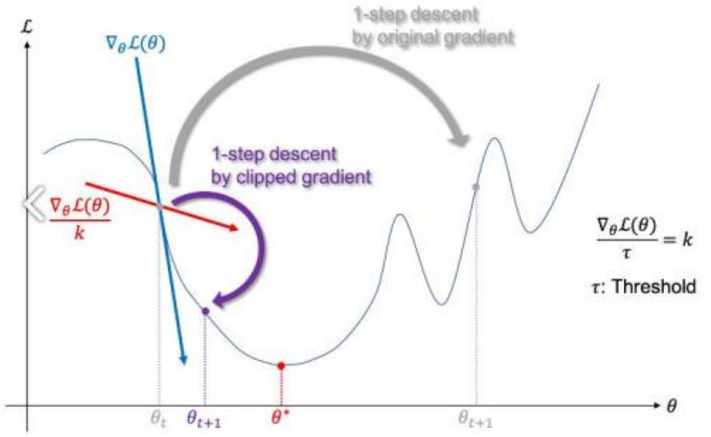
해결방법 Gradient Clipping

- gradient가 일정 threshold를 넘어가면 gradient l2norm으로 나눠주는 방식

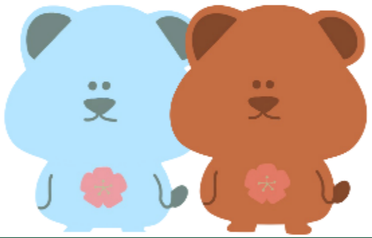
- threshold : gradient가 가질 수 있는 최대 L2norm

-> 학습의 발산 방지

-> 손실 함수를 최소화하기 위한 기울기의 방향은 유지한 채로 크기만 조절



#03 LSTM(Long-Short Term Memory)



EWHA EURON

#03 LSTM(Long-Short Term Memory)

Introduction of LSTM

- A type of RNN that complements **the long-term dependence problem** of RNNs
- **Storing information by separating memory** from the RNN for future use

  : data from long ago can be considered and reflected in the output

Main Idea

- Storing previous steps of information in memory cells and letting it flow
- Based on the information at the present time, **multiply how much you will forget about the past information**

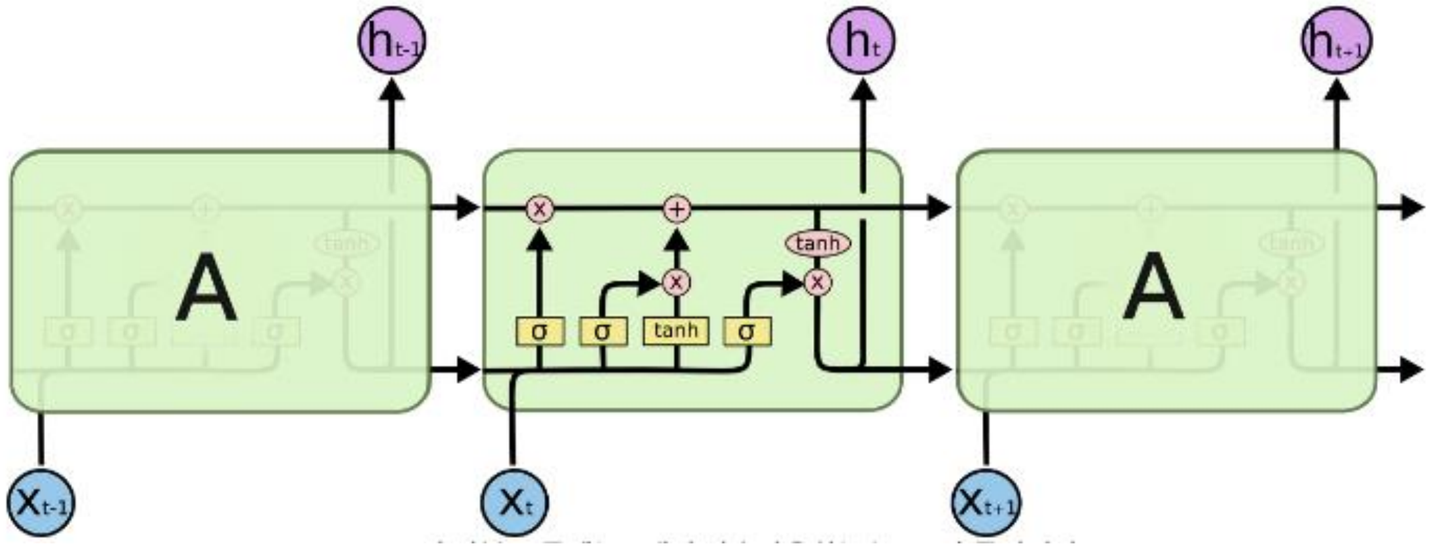
  and add the current information to the result to deliver the information to the next time

#03 LSTM(Long-Short Term Memory)

Introduction of LSTM

- Since **cell-state** is added to the LSTM model, it is more complicated than RNN to calculate the hidden state
- Add **input gates, forget gates, and output gates** to memory cells in the hidden state

  to erase unnecessary memories and decide what to remember.



#03 LSTM(Long-Short Term Memory)

Concept of LSTM

On step t, There is a **Hidden state $\boldsymbol{h}^{(t)}$** and a **cell state $\boldsymbol{C}^{(t)}$**

- both are vectors length *n*
- The cell stores long-term information
- The LSTM can read, erase and write information from the cell

The selection of **which information is erased/written/read is controlled by gates**

- The gates also vectors length n
- On each timestep, each element of the **gates can be opened or closed or somewhere in between**
- The gates are dynamic : their value is computed based on the current context

#03 LSTM(Long-Short Term Memory)

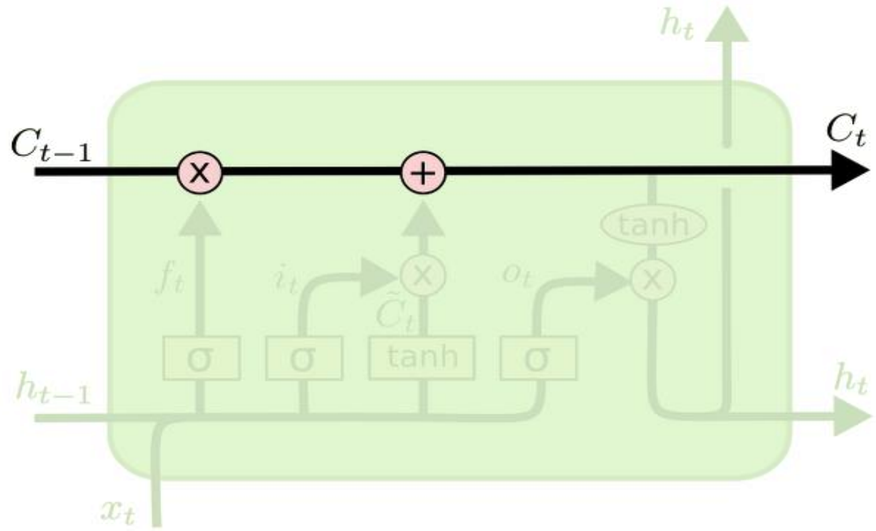
Concept of LSTM

Cell state:

- continues to drive the entire chain while applying only **small linear interaction**, which corresponds to the horizontally drawn upper line

- used as an input for calculating C_{t-1} (previous time point) to obtain C_t (next time point)

- LSTM can add or remove something to the cell state, which is carefully controlled by a structure called **gate**.



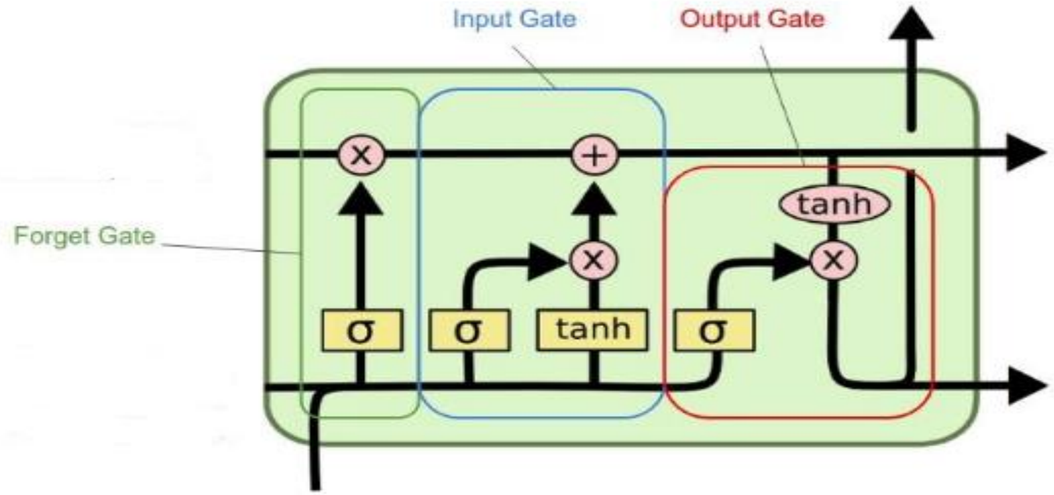
#03 LSTM(Long-Short Term Memory)

Concept of LSTM

Gate:

- an additional way in which information can be communicated, all exist **Sigmoid layers**, which export numbers between 0 and 1, indicating a measure of how much information each component should communicate.

- **forget gate, input gate, and output gate**, which protect and control the cell state.



#03 LSTM(Long-Short Term Memory)

Concept of LSTM

We have a sequence of **input** $\boldsymbol{x}^{(t)}$, and we will compute a sequence **of hidden states** $\boldsymbol{h}^{(t)}$ and a **cell states** $\boldsymbol{c}^{(t)}$

- The gates are dynamic : their value is computed based on the current context

- Based on the information at the present time, multiply how much you will forget about the past information

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

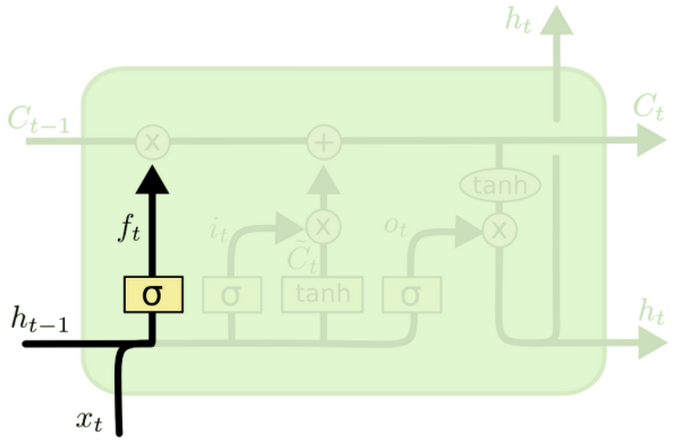
#03 LSTM(Long-Short Term Memory)

LSTM Progress

1) Forget gate layer

- In this step, $\boldsymbol{h_{t-1}}$(state input of RNN Cell) and $\boldsymbol{x_t}$ (input) are received and a value between 0 and 1 is sent to Ct-1.

- After passing the **sigmoid function**, a value between 0 and 1 appears, which is the amount of information that has undergone the deletion process.
    - The closer it is to zero, the more information is deleted, and the closer it is to 1, the more information is fully.



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

#03 LSTM(Long-Short Term Memory)

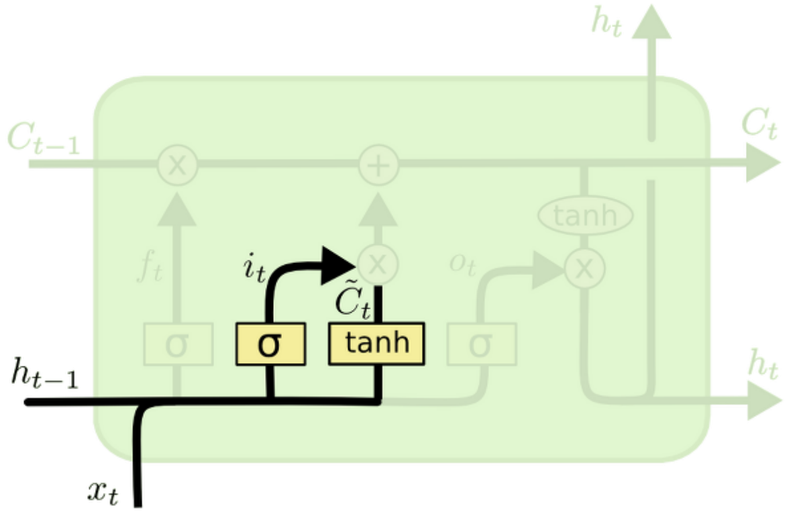
LSTM Progress

2) Input gate layer

- The tanh layer then creates a vector called $\check{C}_t$, which is a new candidate value,

  and prepares to add it to the cell state.

- By combining the information from these two steps, **we make a material to update the state**.



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

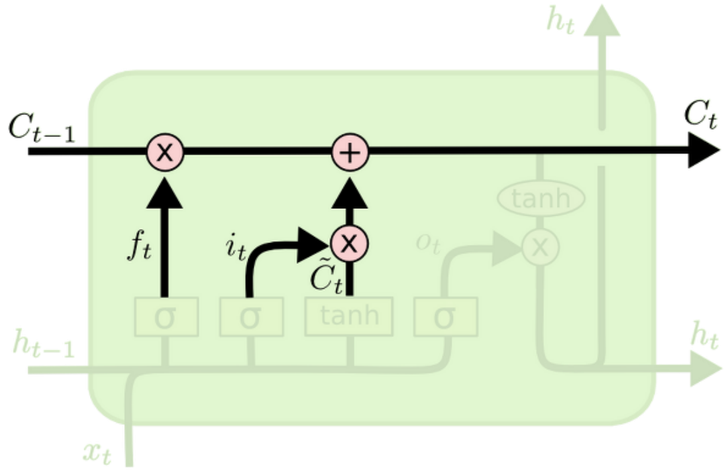
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

#03 LSTM(Long-Short Term Memory)

LSTM Progress

3) Update Cell state

- Past information will be deleted or maintained -> $\boldsymbol{f_t}$
- Whether the current input value is reflected or not -> $\boldsymbol{i_t}$
- C_t : add $f_t * C_{t-1}$ and $i_t * \check{C}_t$



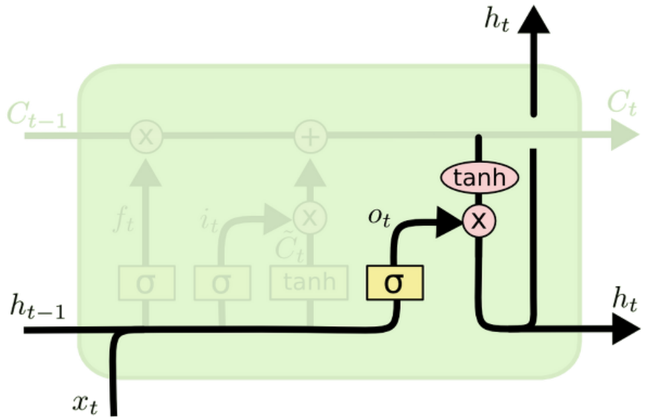
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

#03 LSTM(Long-Short Term Memory)

LSTM Progress

4) Output Gate Layer

- it is determined which part of the cell state is exported as output -> $\boldsymbol{o_t}$
- **hidden state($\boldsymbol{h_t}$)** can be computed multiply tanh($\boldsymbol{C_t}$) by the output gate



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

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

#03 LSTM(Long-Short Term Memory)

LSTM Summary

- **Gate:** their value is computed based on the current context : Dynamic

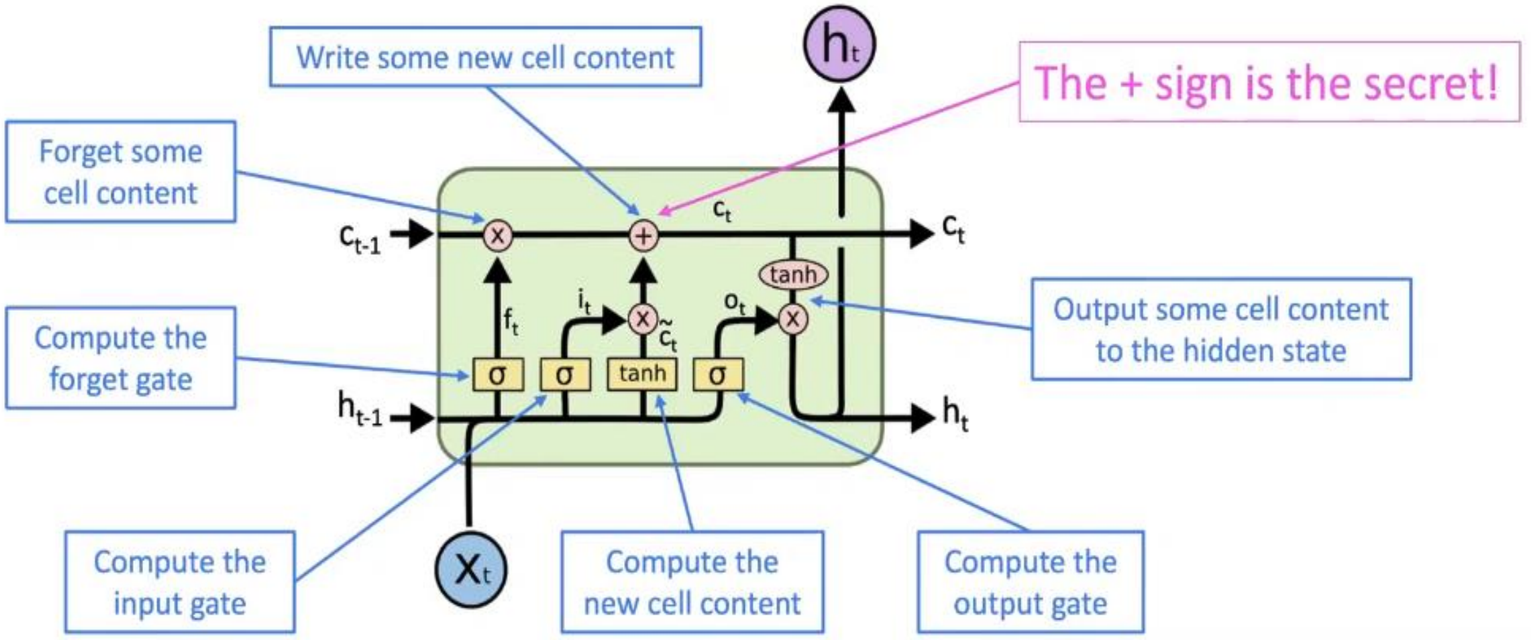
- **Cell state** : $C_t = f_t * C_{t-1} + i_t * \check{C}_t$

  1) Erase forget some content from **last cell state**

  2) Write input some **new cell content**

- **Hidden state** : $h_t = o_t * \tanh(C_t)$

  1) Read output some content from the cell



$$
\begin{aligned}
\boldsymbol{f}^{(t)} &= \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \\
\boldsymbol{i}^{(t)} &= \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \\
\boldsymbol{o}^{(t)} &= \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)
\end{aligned}
\qquad
\begin{aligned}
\tilde{\boldsymbol{c}}^{(t)} &= \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \\
\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)} \\
\boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}
\end{aligned}
$$

#03 LSTM(Long-Short Term Memory)

How does LSTM solve vanishing gradients?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
    - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

#03 LSTM(Long-Short Term Memory)

How does LSTM solve vanishing gradients?

RNN의 h_t에 대한 계산식은 다음과 같다.

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

이때, h_T를 h_t에 대해 미분한다고 하면(T>t), 이는 chain rule에 의해 다음과 같이 표현될 수 있다.

$$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_T}{\partial h_{T-1}} * \frac{\partial h_{T-1}}{\partial h_{T-2}} * \ldots * \frac{\partial h_{t+1}}{\partial h_t}$$

여기서,

$$\frac{\partial h_T}{\partial h_{T-1}} = W_{hh} * tanh'(W_{hh}h_{T-1} + W_{xh}X_T + b_h),$$
$$\frac{\partial h_{T-1}}{\partial h_{T-2}} = W_{hh} * tanh'(W_{hh}h_{T-2} + W_{xh}X_{T-1} + b_h),$$
$$\ldots$$
$$\frac{\partial h_{t+1}}{\partial h_t} = W_{hh} * tanh'(W_{hh}h_t + W_{xh}X_{t+1} + b_h)$$

이므로

$$\frac{\partial h_T}{\partial h_t} = W_{hh}^{T-t} * \prod_{i=t}^{T-1} tanh'(W_{hh}h_i + W_{xh}X_{i+1} + b_h)$$

만약 W_{hh}의 값이 아주 작다면(-1에서 1사이) 미분식이 깊어질수록(T-t가 커질수록) 결과값은 0에 수렴하게 될 것이다(vanished). 반대로 W_{hh}의 값이 아주 크다면, 미분식이 깊어질 수록 결과값은 발산하는 형태를 띌 수 있다(exploded).

LSTM의 Cell State C_t에 대한 계산식은 다음과 같다.

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

이때, C_T를 C_t에 대해 미분한다고 하면(T>t), 이는 chain rule에 의해 다음과 같이 표현될 수 있다.

$$\frac{\partial C_T}{\partial C_t} = \frac{\partial C_T}{\partial C_{T-1}} * \frac{\partial C_{T-1}}{\partial C_{T-2}} * \ldots * \frac{\partial C_{t+1}}{\partial C_t}$$

여기서, $\frac{\partial C_T}{\partial C_{T-1}} = f_T, \frac{\partial C_{T-1}}{\partial C_{T-2}} = f_{T-1}, \ldots, \frac{\partial C_{t+1}}{\partial C_t} = f_{t+1}$ 이므로

$$\frac{\partial C_T}{\partial C_t} = \prod_{i=t+1}^{T} f_i$$

위 식의 f는 sigmoid함수의 output이기 때문에 (0,1)의 값을 갖게 되는데, 이 값이 1에 가까운 값을 갖게되면 미분값(gradient)이 소멸(vanished)되는 것을 최소한으로 줄일 수 있게된다. f값이 1에 가깝다는 것은, Cell State 공식에 의하면 오래된 기억(long term memory)에 대해 큰 비중을 둔다는 것과 같은데, 이로인해 gradient 또한 오래 유지된다는 것은 꽤나 흥미로운 현상이다.

+더불어 f는 1보다 큰 값을 가질 수 없으므로 미분식이 깊어진다고 해서(T-t값이 커진다고 해서) 이로인해 그 값이 넘치게(exploded) 되지는 않는다.

#03 LSTM(Long-Short Term Memory)

Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures, especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can be vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly

Solution: lots of new deep feedforward/convolutional architectures add more direct connections

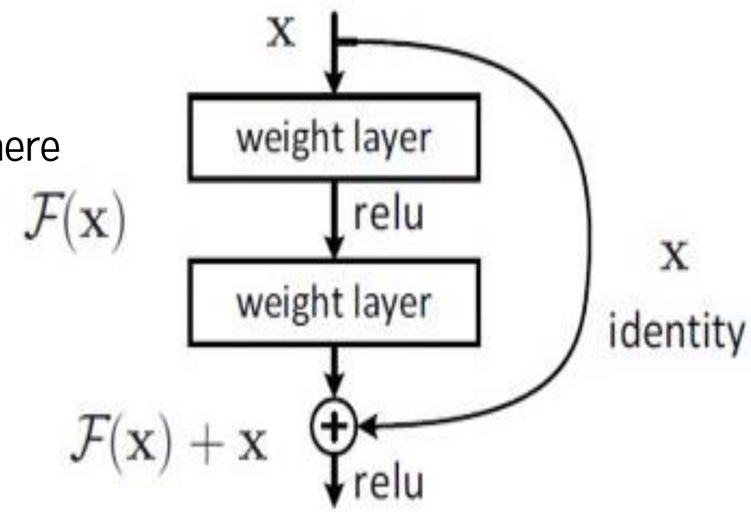
1) Residual connections "ResNet"

= skip-connections

- It only created a shortcut to add the input value to the output value.

- **shortcut** : to send the block to the next block without learning to proceed here

Through this shortcut method, even if the gradient disappears during the learning process of the block, the problem of vanishing gradient can be reduced by adding x before learning the block



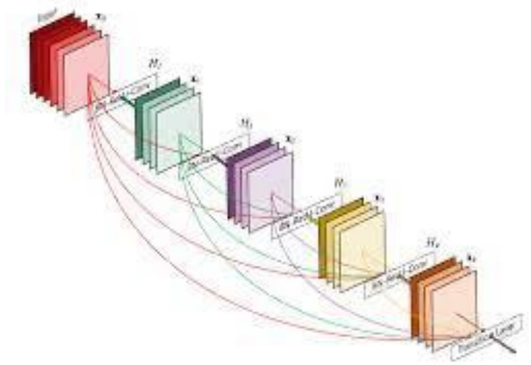
#03 LSTM(Long-Short Term Memory)

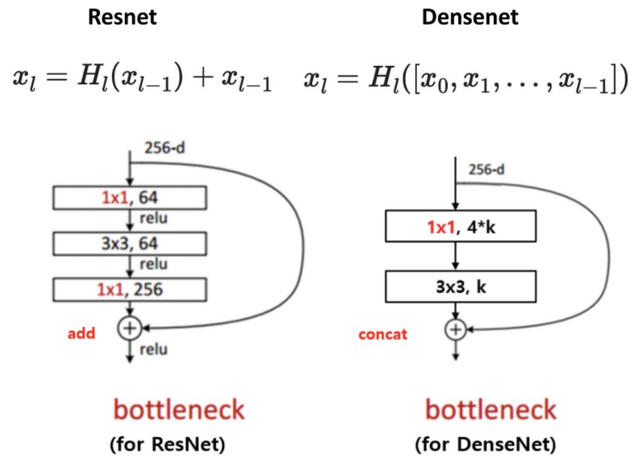
Is vanishing/exploding gradient just a RNN problem?

Solution: lots of new deep feedforward/convolutional architectures add more direct connections

2) Deep connections **"DenseNet"**

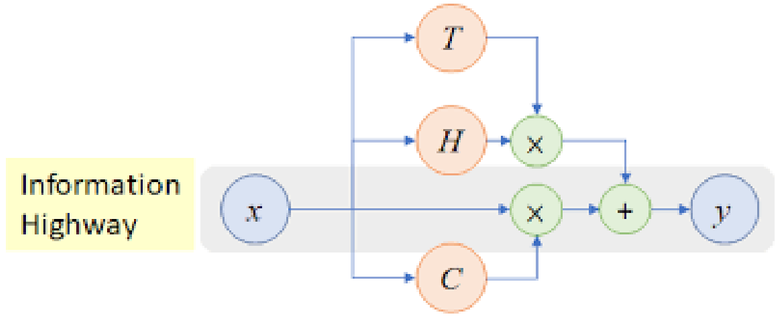
- Directly connect each layer to full layers
- Skip-connection with concatenation
- cf) ResNet: Skip-connection with addition





3) Highway connections **"HighwayNet"**

- like residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feed-forward/ convolutional network

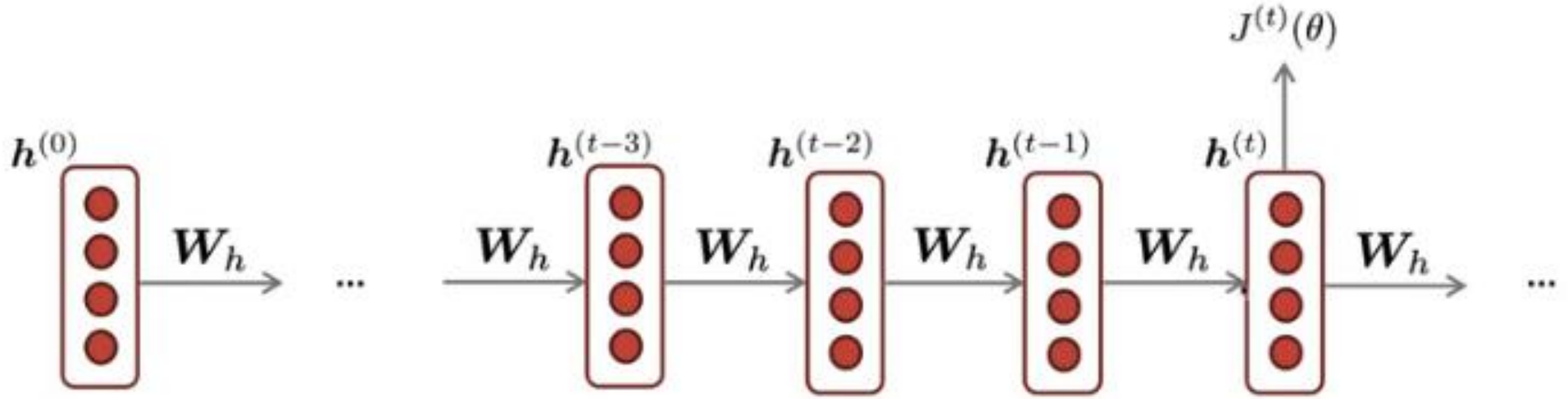


#03 LSTM(Long-Short Term Memory)

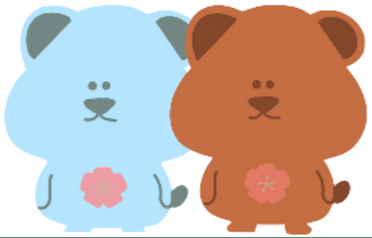
Is vanishing/exploding gradient just a RNN problem?

Conclusion:

Though vanishing/exploding gradients are a general problem,
RNNs are particularly unstable due to the repeated multiplication by the same weight matrix



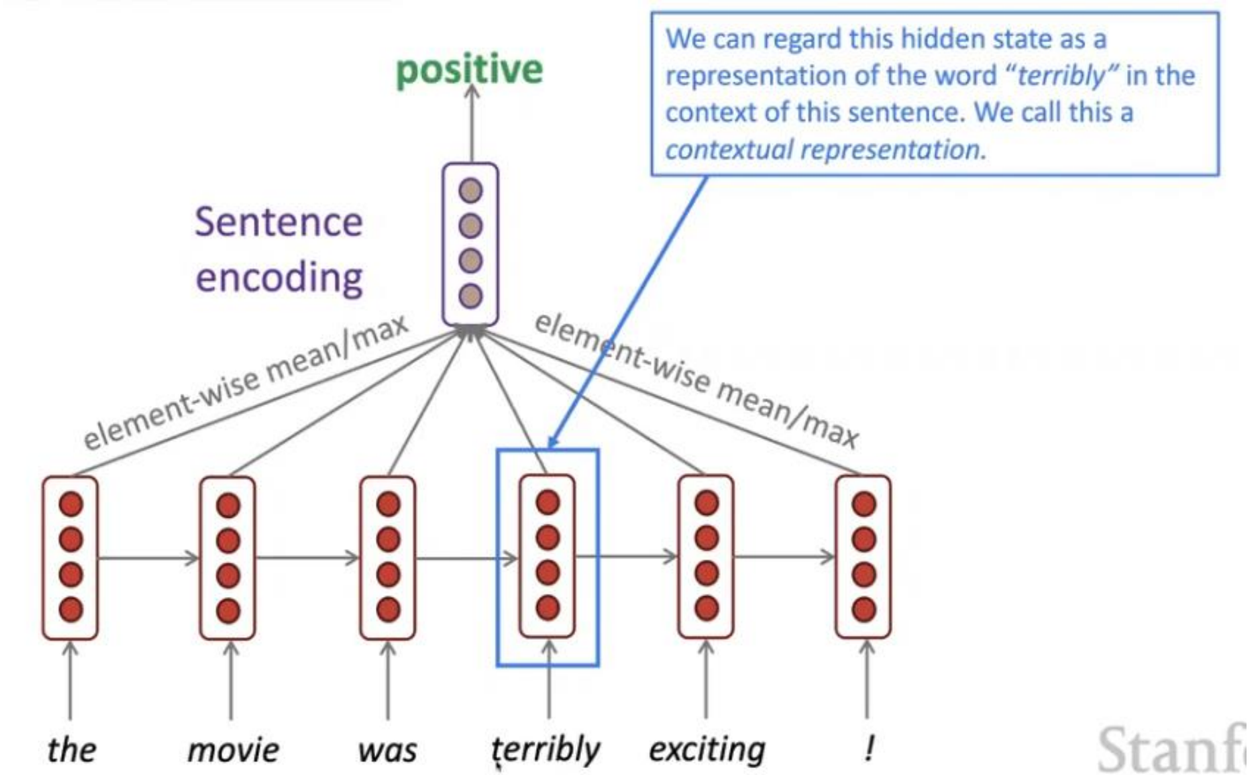
#04 Bidirectional and Multi-layer RNNs



EWHA EURON

#04 Bidirectional and Multi-layer RNNs

Bidirectional and Multi-layer RNNs : Motivation



Task : Sentiment Classification

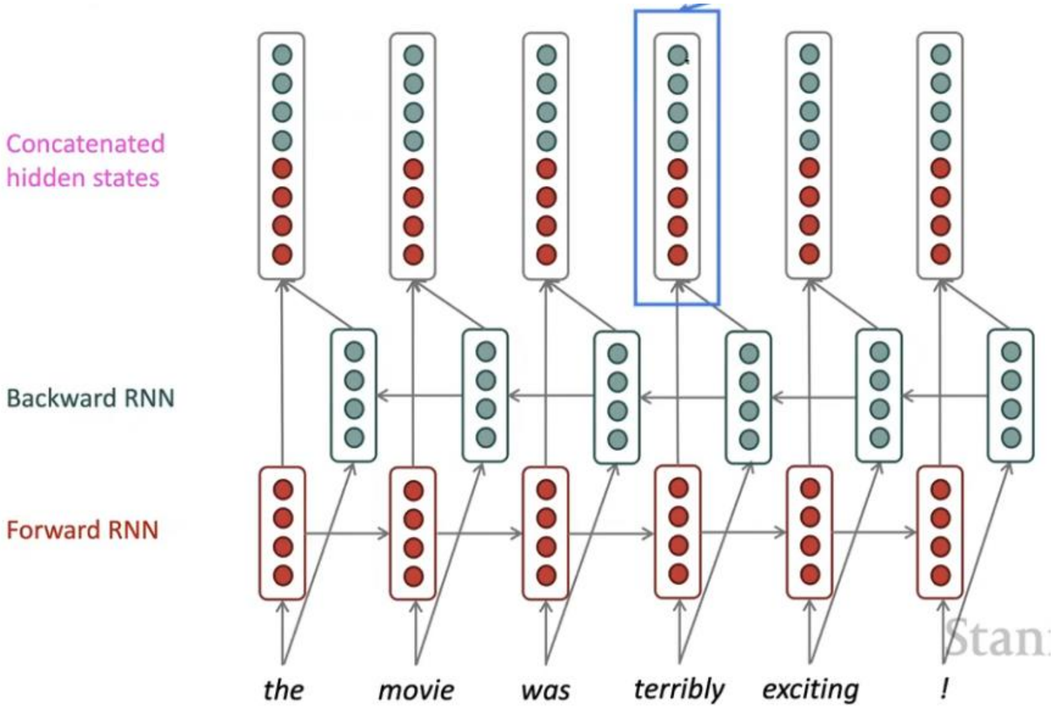
The problem is that only **"this movie was"** considered when expressing the meaning of **"terribly"** due to the nature of RNN flowing in one direction

But in fact, the important information is **exciting**.

=> to represent by using information in both left and right directions.

#04 Bidirectional and Multi-layer RNNs

Bidirectional RNNs



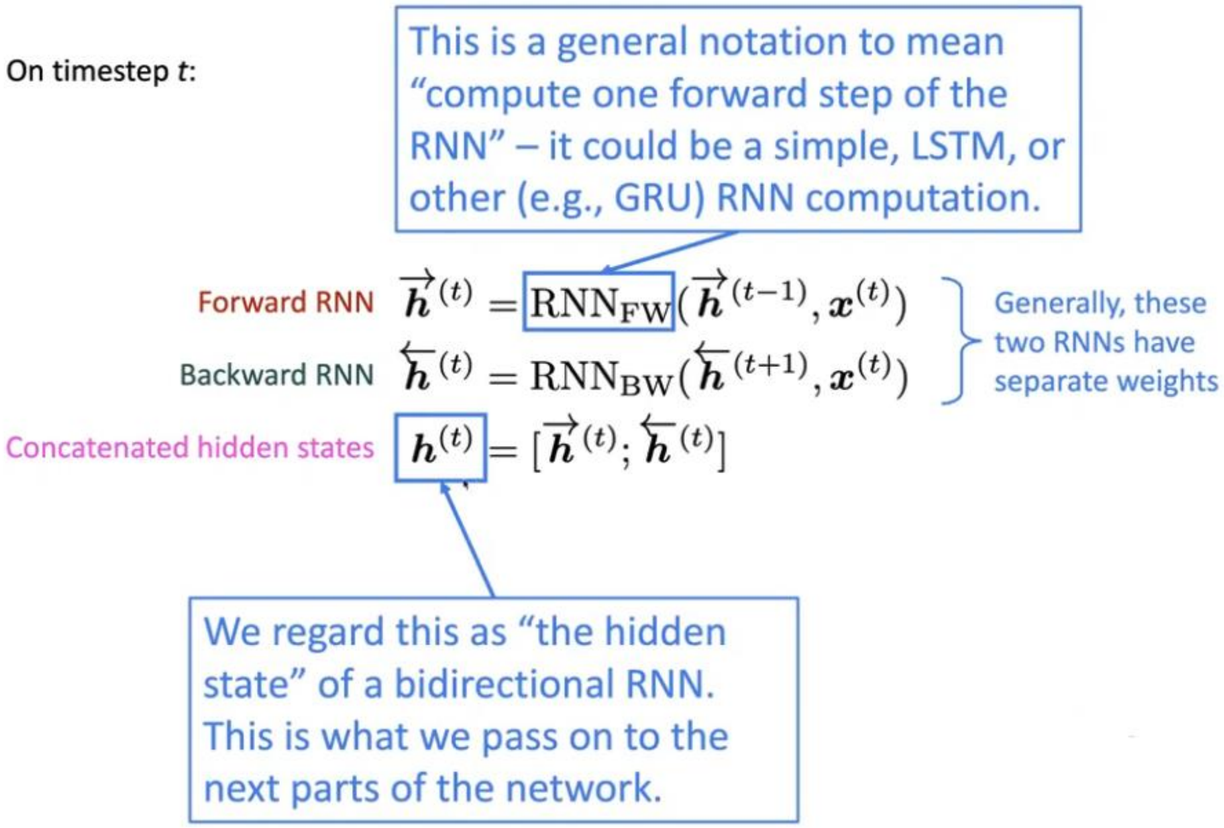
forward RNN, Backward RNN

The hidden state created by concatenating the two RNNs

It is used as the hidden state of the entire model.

#04 Bidirectional and Multi-layer RNNs

Bidirectional RNNs



- Rnn can be used in various types of vanila RNN, LSTM, and GRU.

#04 Bidirectional and Multi-layer RNNs

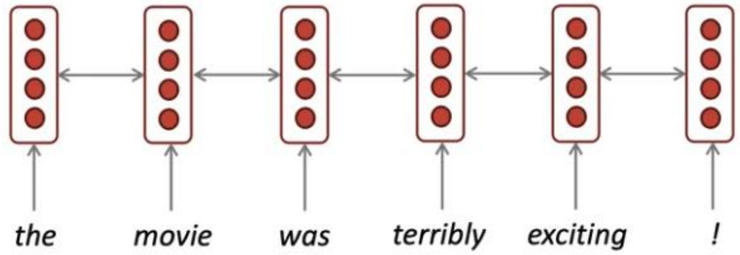
Bidirectional RNNs

Note : bidirectional RNNs are only applicable if you **have access to the entire input sequence**

- They are not applicable to Language Model, because in LM you only have left context available

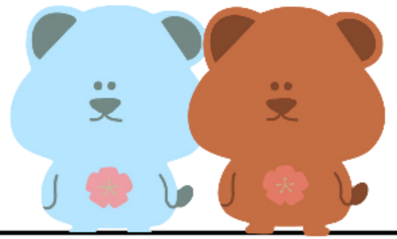
If you do have entire input sequence, bidirectionality is powerful.

For example, BERT(Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

THANK YOU



EWHA EURON