RNN, Self-Attention

조인영

목차

01

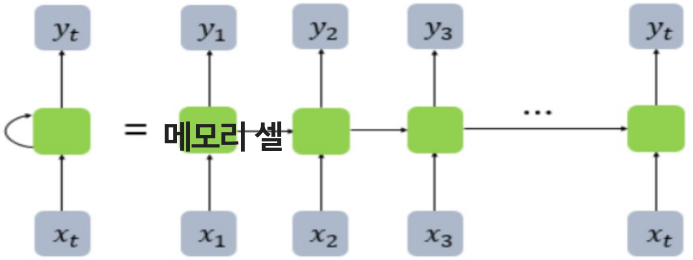
Recurrent Neural Network

RNN이란?

- **Recurrent Neural Network**의 약자, 순환 신경망
- **순차 데이터나 시계열 데이터**를 이용하는 인공 신경망 유형

  →언어변환, 자연어 처리, 음성 인식, 이미지 캡션과 같은 순서, 시간문제에 흔히 사용됨
- 입력과 출력을 시퀀스 단위로 처리(시퀀스: 문장 같은 단어가 나열된 것)
- 입력하기 전 tokenizing 필요(단어→ 숫자)

  ex) 저는[1,0,0] 강사[0,1,0] 입니다[0,0,1]
- RNN의 은닉층에서 활성화 함수를 통해 결과를 내보내는 역할을 하는 노드를 **셀**이라 함
- 이 셀은 이전의 값을 기억하려고 하는 일종의 메모리 역할을 수행 → 메모리 셀
- 은닉층의 메모리 셀에서 나온 값이 다음 은닉층의 메모리 셀의 입력 값



01

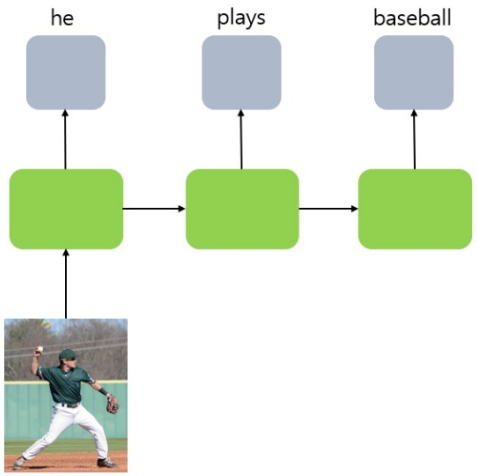
Recurrent Neural Network

RNN의 여러가지 유형

- RNN은 입력과 출력의 길이를 다르게 설계할 수 있어 다양한 용도로 사용할 수 있는 장점이 있음

① 일 대 다(one-to-many)

- 하나의 입력에 여러 개의 출력
- 이미지 캡셔닝

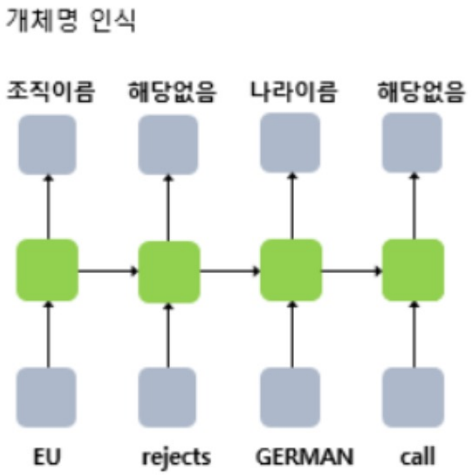


② 다 대 일(many-to-one)

- 시퀀스 입력에 대해 하나의 출력
- 입력문서의 감성분류,스팸 메일 분류

③ 다 대 다(many-to-many)

- 시퀀스 입력에 대해 시퀀스 출력을 함
- 챗봇, 번역기에 주로 쓰임
- 개체명 인식,품사 태깅과 같은 작업도 포함



01

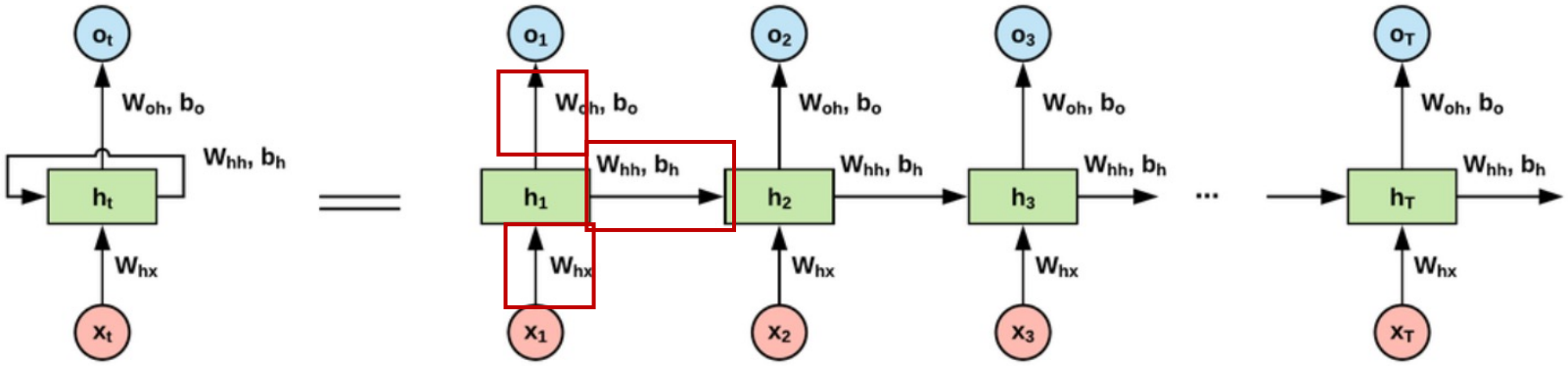
Recurrent Neural Network

RNN구조

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

→ 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정

→ **순환 신경망(Recurrent Neural Network)**

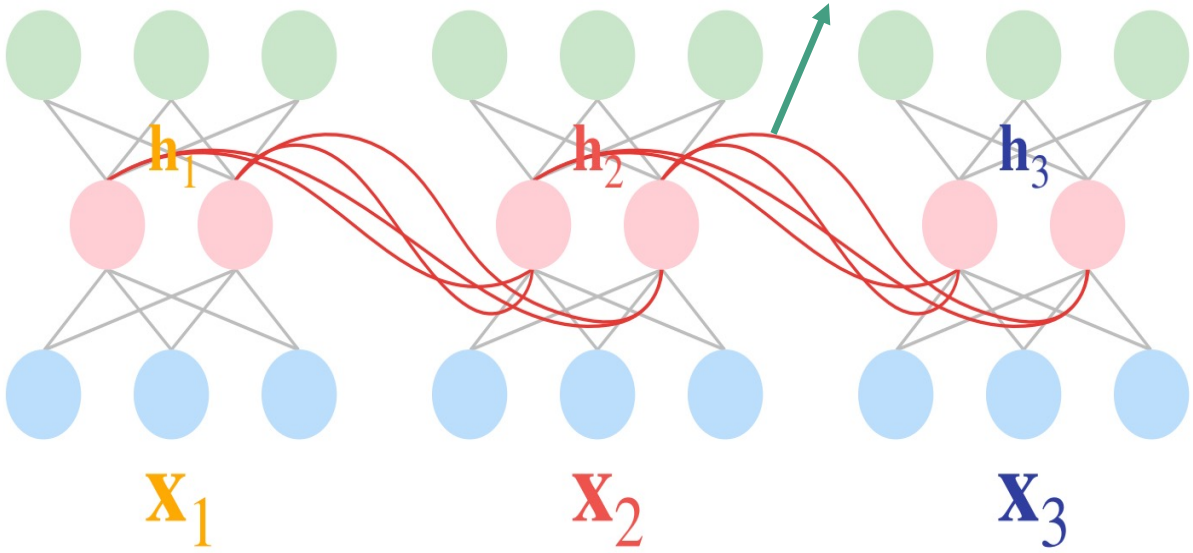


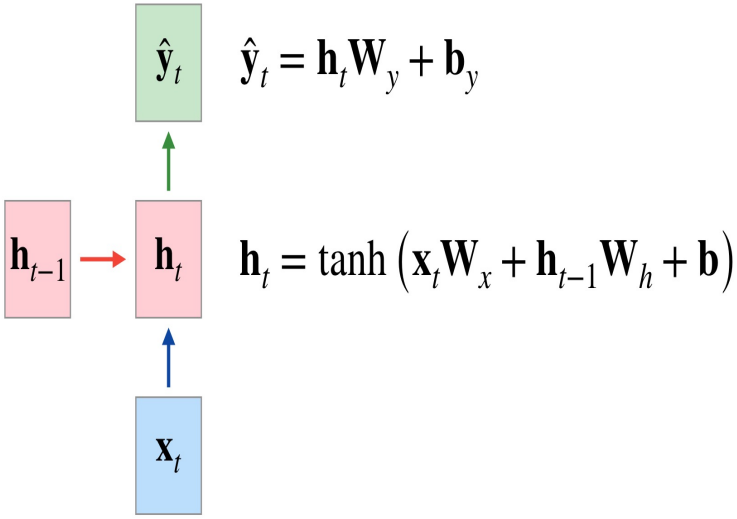
01

Recurrent Neural Network

RNN학습 방식

H를 통해서 이전 정보를 넘길 수 있는 것임





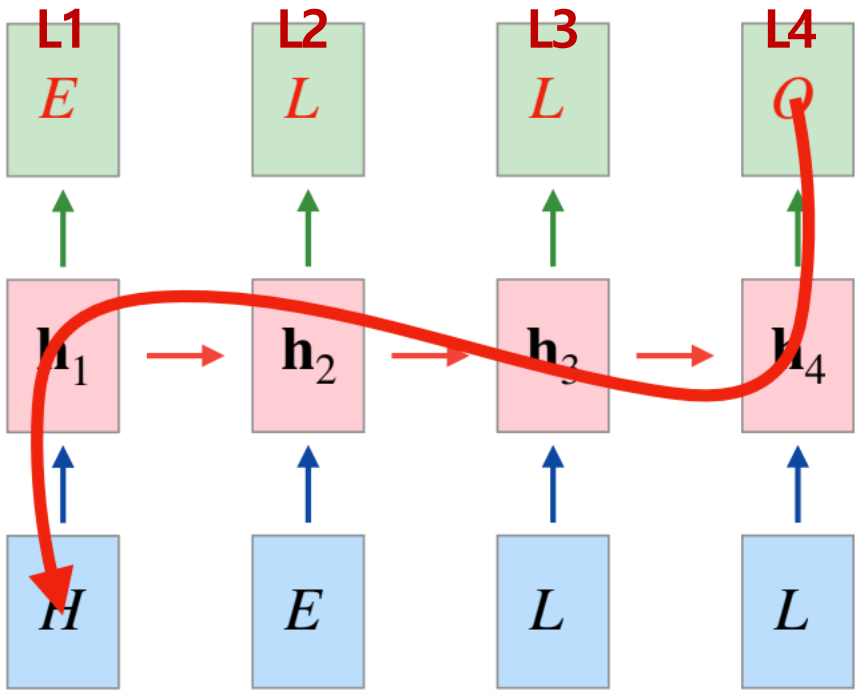
- $h1 = \tanh(x1 * Wx + b)$
- $h2 = \tanh(x2 * Wx + h1Wh + b)$
- $h3 = \tanh(x3 * Wx + h2Wh + b)$
- y(hat)_3=$h3 * Wy + by$

$* Wx, Wh, Wy, b, by$는 시간에 따라 다르지 않음!

Recurrent Neural Network

RNN의 구조적 한계

- **Next token prediction**
- 다음에 어떤 알파벳이 나올지를 예측(다중분류)
- ELLO 네개 글자에 대해 cross-entropy를 더해 loss 정의



Loss=L1+L2+L3+L4

Recurrent Neural Network

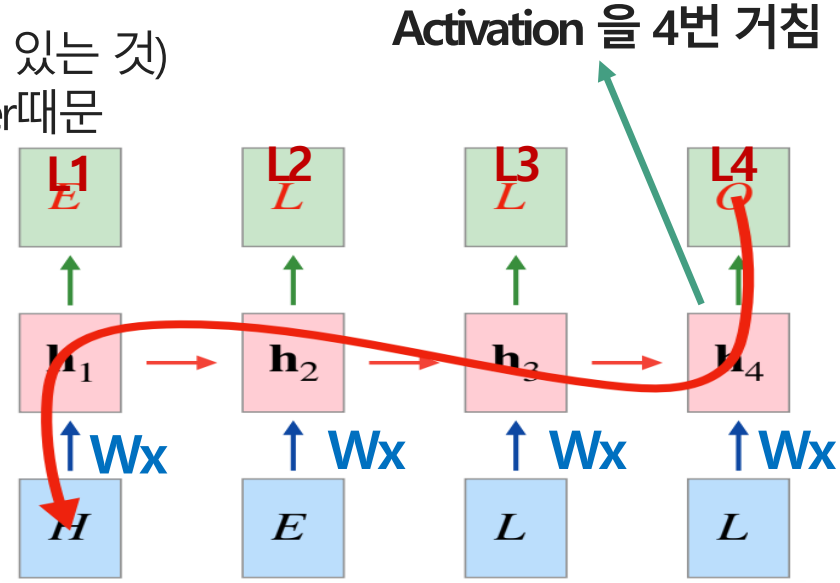
RNN의 구조적 한계

- O가 나오게 하기 위해 H가 gradient에 미치는 영향력?

1) 멀수록 잊혀진다(Back propagation)
→ vanishing Gradient 발생
→ 계속해서 activation function이 적용됨(tanh의 미분 값은 최대1)
→ ∂L4/∂Wx = H + E + L + L (L에서 H로 갈수록 영향력이 작아짐) *Loss=L1+L2+L3+L4

2) 갈수록 흐려진다(forward propagation)
→단어의 정보가 점점 흐려짐(정보를 잃으면서 진행하고 있는 것)
→ tanh의 (-1,1)사이의 값에 갇히고 weight, hyperparameter때문

거리에 영향을 받는 것이 문제!
→ 가까운 단어에 영향을 크게 받음
→ 전체를 봐서 다음을 예측하는 것이 좋지 않을까?



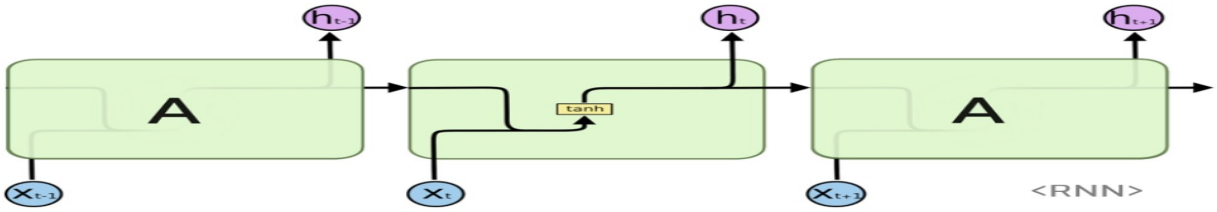
02

Long Short Term Memory

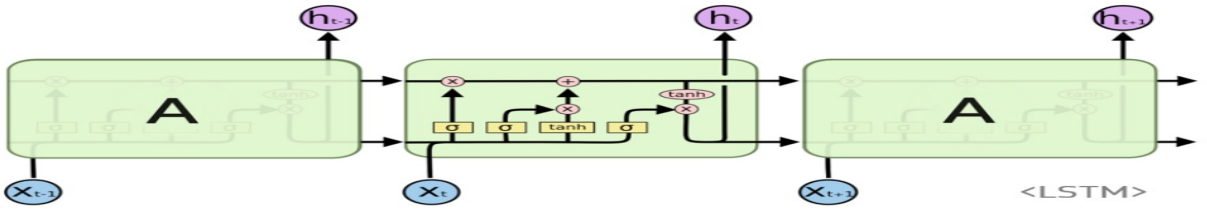
LSTM

- RNN은 관련정보와 그 정보를 사용하는 지점 사이의 거리가 멀 경우 학습능력이 크게 저하되는 문제점이 있음(vanishing gradient problem)
- 이 문제를 해결하기 위해 고안된 것이 LSTM
- 단순한 neural network layer한 층 대신에, 4개의 layer가 특별한 방식으로 서로 정보를 주고받음
- LSTM의 핵심은 cell state(정보를 선택적으로 활용할 수 있도록 함)
- Cell state와 forget gate, input gate, output gate를 통해 계산이 이루어진다.
- 어떤 정보를 잊을지 유지할지를 선택하여 long term과 short term에 대한 정보를 고려 가능

RNN 구조



LSTM 구조

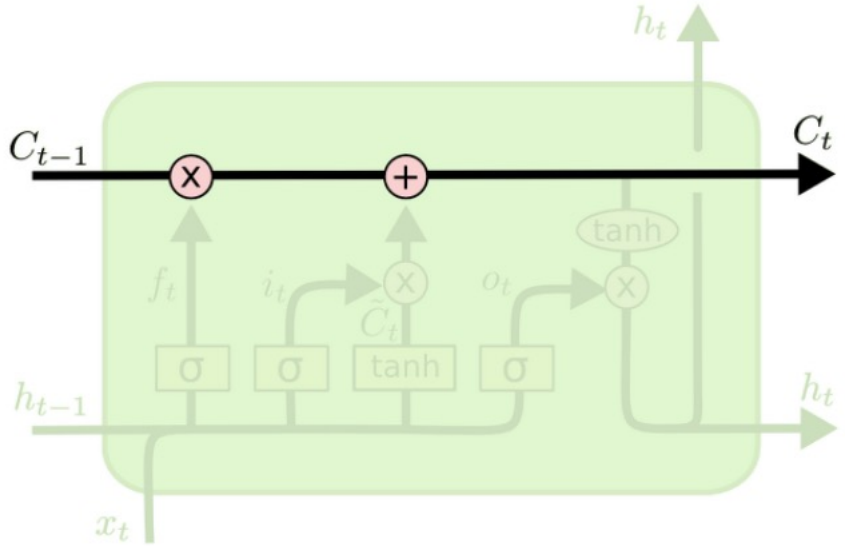


02

Long short Term Memory

LSTM

- **LSTM의 cell state**
  → 벨브역할: 얼만큼 이 정보를 받아들일지, 오랫동안 기억할 것은 무엇인지, 선택적으로 h를 담음
  → cell state는 hidden state와 마찬가지로 이전 시점의 cell state를 다음 시점으로 넘겨줌
  → cell state의 주역할은 gate들과 함께 작용하여 정보를 선택적으로 활용할 수 있도록 함
  → cell state의 update는 각 gate의 결과를 더함으로써 진행하는데, 이는 시퀀스가 길더라도 gradient, 오차를 상대적으로 잘 전파할 수 있도록 함



LSTM의 cell state

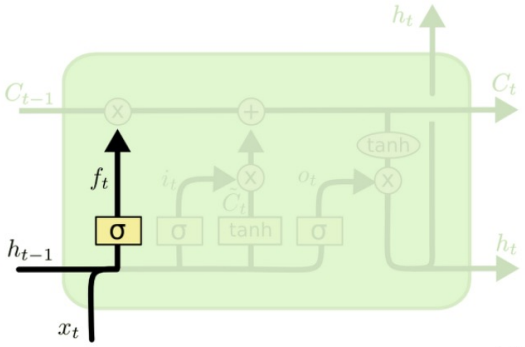
02

Long short Term Memory

LSTM

LSTM의 Gate: Forget gate, input gate, output gate
→ gate는 cell state와 함께 정보를 선택적으로 활용할 수 있도록 함
→3개의 gate 모두 활성화함수로 시그모이드를 적용

- **LSTM의 Forget gate: 과거 정보를 잊기'위한 게이트**
  → 시그모이드 함수의 출력 범위는 0에서 1사이이기 때문에 그 값이 0 이라면 이전 상태의 정보는 잊고 1이라면 이전 상태의 정보를 온전히 기억하는 것



LSTM의 forget gate layer

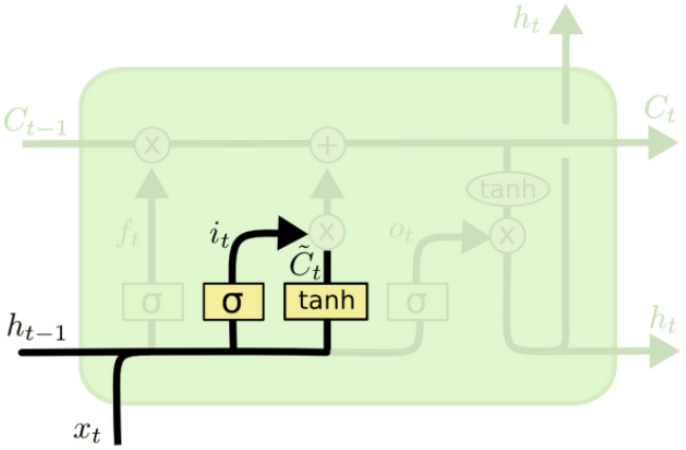
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

02

Long short Term Memory

LSTM

- **Input gate: ' 현재 정보를 기억하기 ' 위한 게이트**
  → 앞으로 들어오는 새로운 정보 중 어떤 것을 cell state에 저장할 것인지
- *it*의 범위는 0~1, 강도를 의미, 0에 가까울수록 많은 정보를 삭제한다는 것을 의미
- C~*t*의 범위는 −1~1, 나중에 현재정보를 *cell state*에 얼마나 더할지를 결정하는 역할



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

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

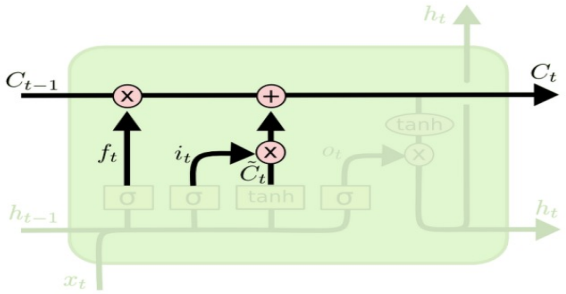
LSTM의 input gate layer

02

Long short Term Memory

LSTM

- **LSTM의 cell state업데이트**
  → 과거 state인 Ct-1을 업데이트해 새로운 cell state를 만듦, 이전 단계 값들을 사용하기만 하면 됨
  → 과거에서 유지할 정보+현재에서 유지할 정보 = 현재 시점의 cell state update



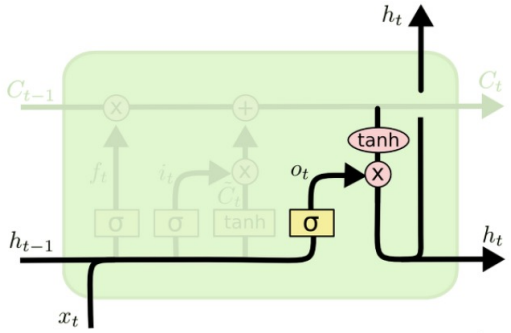
이전 시점의 cell정보를
얼마나 유지할지

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

현재 기억할 정보

LSTM의 cell state 업데이트

- **LSTM의 output gate layer**
  →최종적으로 우리가 출력할 값



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

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

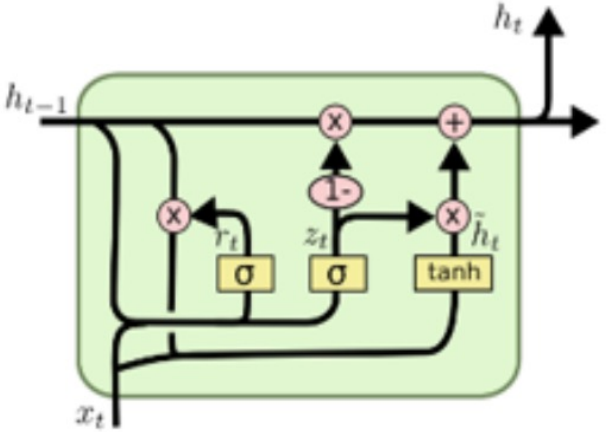
LSTM의 output gate layer

03

Gated Recurrent Unit

GRU

- 기존 LSTM의 구조를 조금 더 간단하게 개선한 모델
- GRU에는 reset gate, update gate 2개의 gate만을 사용
- Cell state, hidden state가 합쳐져 하나의 hidden state로 표현
- Reset gate: 이전 hidden state의 값을 얼마나 활용할 것인지에 대한 정보
- Update gate:과거와 현재의 정보를 각각 얼마나 반영할지에 대한 비율을 구함



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

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \quad \text{---(2)}$$ reset gate

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \quad \text{---(3)}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad \text{---(4)}$$ 현 시점의 출력값
→ 1-z: 과거정보를 얼마나 사용할지

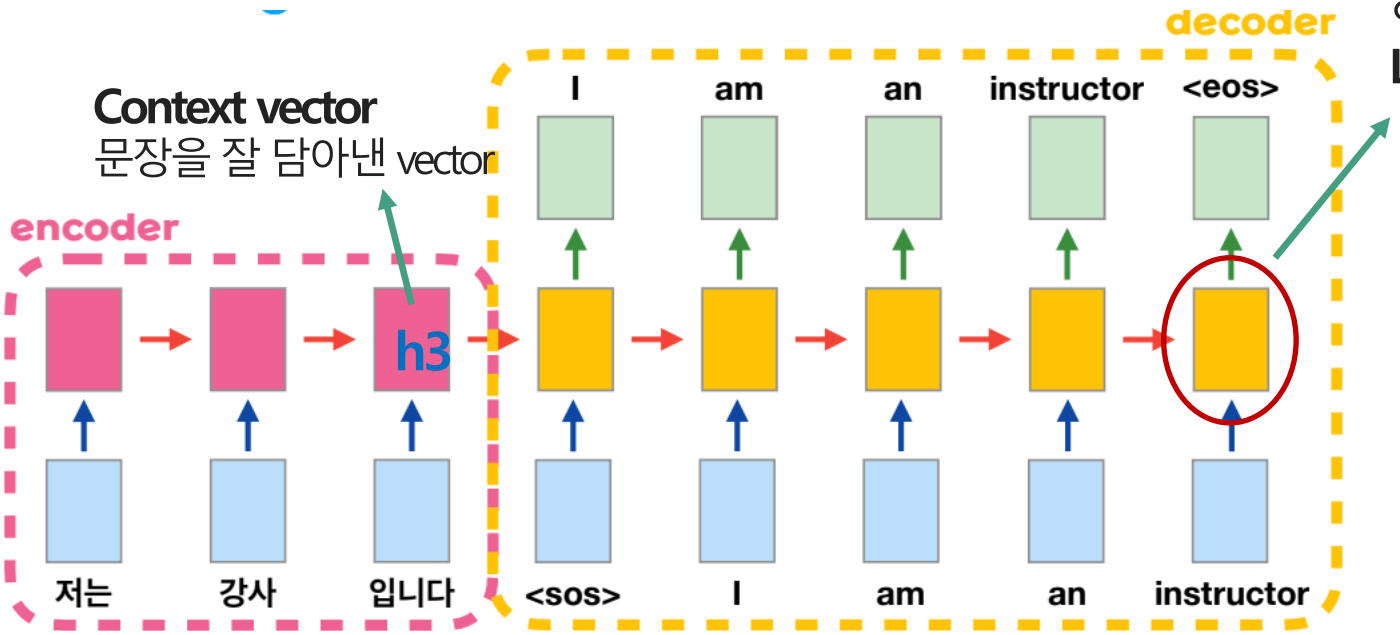
Problem : 여전히 많은 activation function을 통과해야함, 각종 Hyperparameter, weight들이 잘 못 학습되면 기능을 제대로 하지 못함

04

Sequence to sequence

Seq2seq

- 2개의 RNN을 이용, 한 시퀀스를 다른 시퀀스로 변환하는 작업을 수행 (기계번역 분야)
- 전체 input을 살펴본 후, 임의의 context vector를 출력하여 전체적인 맥락 파악 가능
- 2개의 모듈 encoder, decoder로 구성
- Encoder는 어떤 시계열 데이터를 압축해서 표현해줌
- Decoder는 압축된 데이터를 다른 시계열 데이터로 변환
- Encoder의 마지막 h를 decoder의 처음 h로 사용



하나하나의 cell들은
LSTM, GRU를 주로 사용

04

Sequence to sequence

Seq2seq 문제점, RNN의 해결책

- 멀수록 잊혀짐(encoder, decoder 모두)
- 갈수록 정보가 흐려짐
  → 하나의 context vector에 마지막 단어의 정보가 가장 뚜렷하게 담기기 때문
  → 이러한 context vector를 입력 값을 받은 decoder가 번역하기 때문에 마지막 단어의 영향이 큼

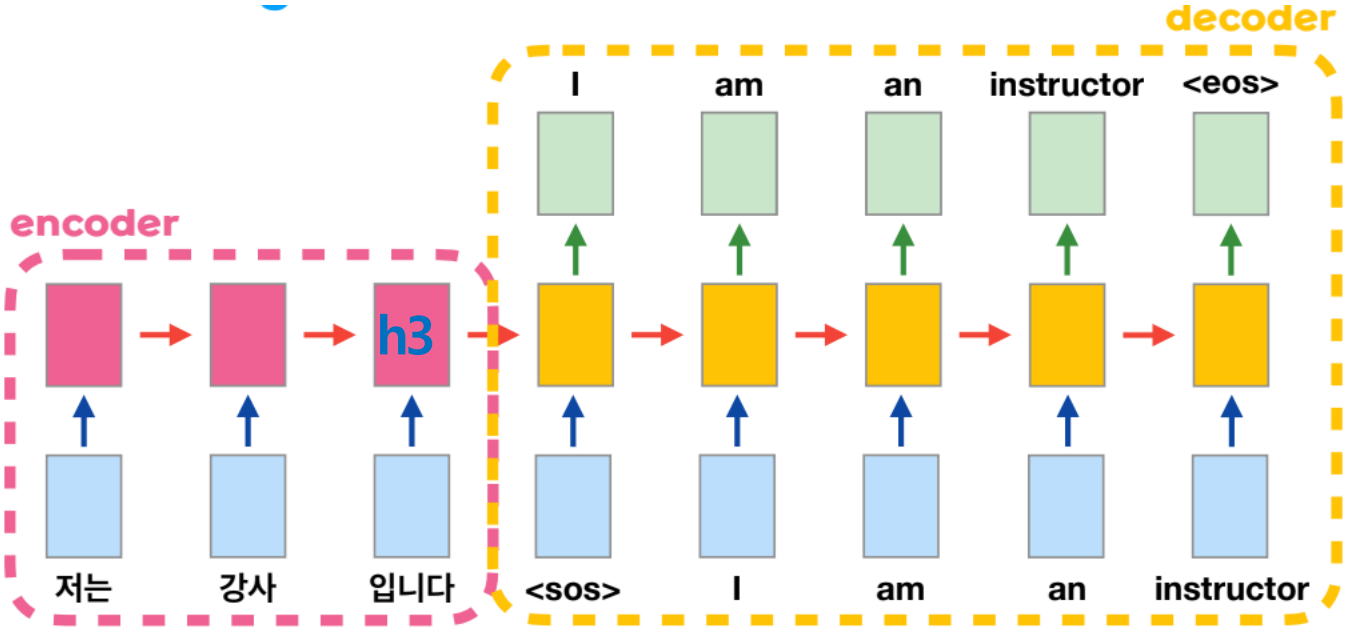
어떤 단어를 주목할지 학습하는 transformer가 필요

→ transformer의 중요 개념: attention

05

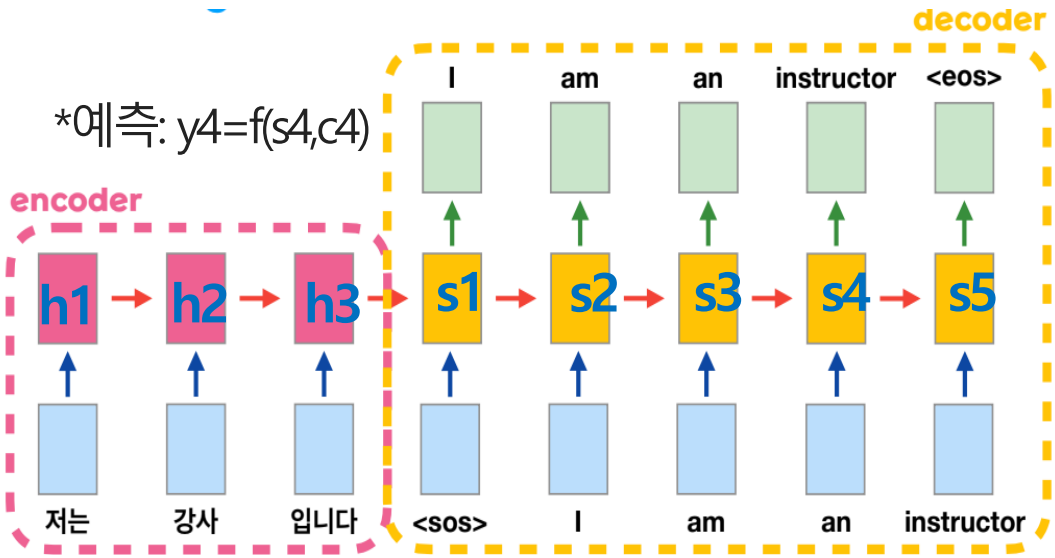
RNN+attention

- 입력문장이 긴 상황에서 번역 품질이 떨어지는 현상을 보정하기 위해, 중요한 단어에 집중하여 Decoder에 바로 전달하는 **attention 기법**이 등장
- 기존에는 context vector로 h3만을 사용
- 시점마다 다른 context vector를 사용



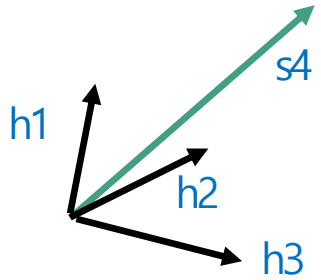
RNN+attention 작동방식

1. Encoder의 hidden state(h1,h2,h3)들을 step 별로 구한다.
2. 각 hidden state(h1,h2,h3)에 decoder의 hidden state를 각각 dot product한다. → Attention score
3. 점수를 softmax한다.(점수의 합이 1)
4. Softmax 된 점수에 해당하는 각각의 hidden state를 곱한다.
5. 점수에 곱해진 vector들을 더한다 → Context vector



$$C4 = W1 * h1 + W2 * h2 + W3 * h3$$
$$=< S4, h1 >* h1 +< S4, h2 >* h2 +< S4, h3 >* h3$$

<S4,h1>: h1을 얼마나 많이 보는 것이 좋을지를 내적을 통해 판단
→ 즉, 어떤 단어를 중요하게 봐야할지를 고려 가능
→ Ex) S4는 h2와 가깝게 되도록 학습할 것이고
이는 강사에 **attention** 하는 것임



05

RNN+attention 문제

- Decoder에서는 여전히 멀수록 잊혀진다는 문제가 발생+ 갈수록 흐려지는 정보에 attention함

  → "쓰다"라는 단어의 뜻을 이해하기 위해 "돈을", "모자를" 등과 같은 x1 앞단어를 봐야 알 수 있음

  → h7에는 x1이 이미 흐려진 채로 들어가 있음(x7의 참의미를 담지 못하고 있는 것)

- 영어라면 뒤의 단어들도 고려해주어야 함

거리에 영향을 받는 것이 여전함

→ self attention

06

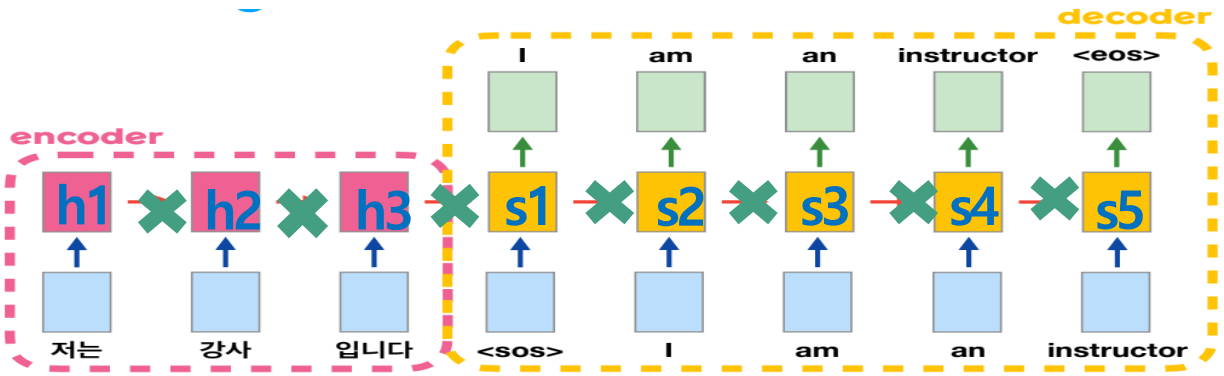
Self-attention

- Attention을 자기 자신한테 취함, attention 중에서 자기 자신에게 attention 메커니즘을 행하는 방식
- 어떤 단어를 주목할지 학습, 거리라는 개념을 버림
- 왜 self attention? → 단어들의 연관성을 알기 위해

$$C4^{new} = < S4^{new}, h1^{new} > * h1^{new} + < S4^{new}, h2^{new} > * h2^{new} + < S4^{new}, h3^{new} > * h3^{new}$$

$$h2^{new} = < h2, h1 > * h1 + < h2, h2 > * h2 + < h2, h3 > * h3$$ *<h2,h1>: 얼머나 (저는,강사)사이의 관계가 깊은지

$$S4^{new} = < s4, s1 > * s1 + < s4, s2 > * s2 + < s4, s3 > * s3 + < s4, s4 > * s4 + < s4, s5 > * s5$$



Next token 문제이기 때문에
다음 시점의 정보를 알 수 있음
→ future는 전부 masking
→Masked self attention

06

Self-attention 장점

- 각각의 단어가 다른 단어들과 얼마나 밀접한 관련이 있는지를 고려
- 연산량이 적고 속도가 빠르다는 장점이 있음
  (RNN경우 순차적인 연산이기에 병렬화 불가능, 연산속도 저하)
- RNN의 vanishing gradient문제 해결
- 거리에 따른 정보가 흐려진다는 문제를 해결 가능
  (decoder가 마지막 단어만 열심히 보는 문제를 해결 가능)