

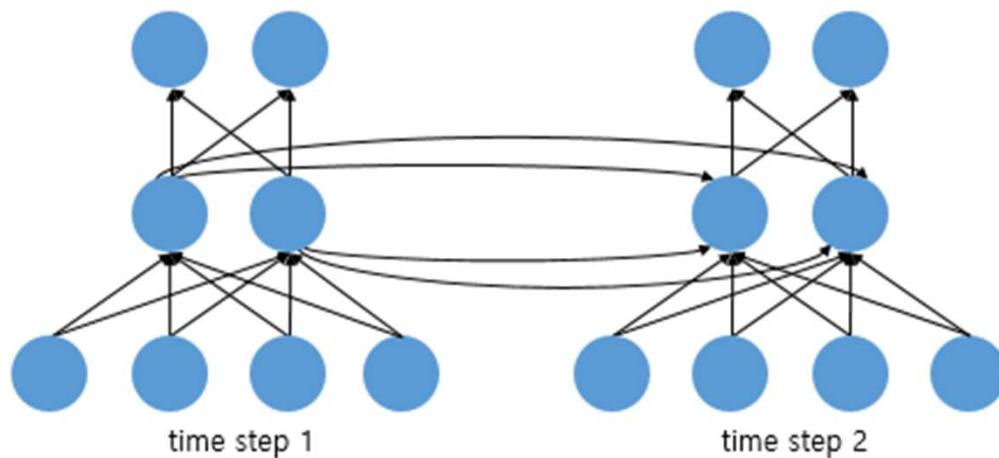
Transformer



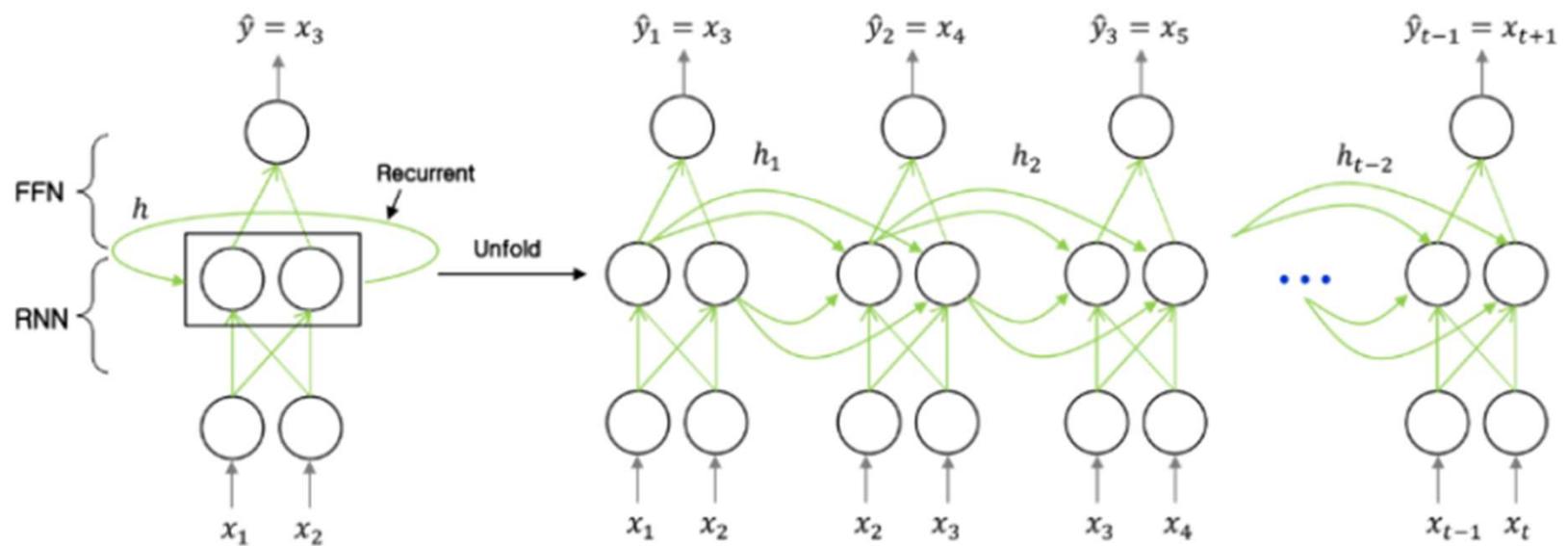
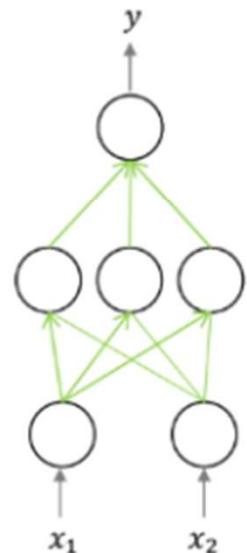
사전 Review : RNN

- RNN : Recurrent Neural Network

- Activation Function : 일반적으로 tanh로 사용 (ReLU는 RNN에서는 좀 계속 곱해지게 되면서, 진동 발산이 잘 되면서 수렴이 좀 어려운 경우가 있어서...최신 AF을 주로 사용하지 않음!!) → sigmoid 보다는 나으며, 최신 것 보다는 덜한 tanh를 주로 사용.
- 순차적인 Sequence !!!!
 - **Recurrent** : 이전의 정보들을 모델에 반영 → h 가 이 역할을 함
 - 입력에 대한 중요한 부분 : **가변적인 입력에 대한 대응이 유리함**!! → 일반적인 DNN의 경우에는 입력하는 차원에 따라서 input layer의 노드의 수가 가변적으로 계속 해야하지만, RNN은 그냥 계속 반복 통과만 시키면 됨!!! → 그래서 시계열, NLP 등에 유리하게 됨!!!
 - **다음이 무엇이 나올까 예측에 주로 사용!!!** → 시계열 상 다음 값, 언어 상에서 연관된 다음 단어, 알파벳 등 예측에 사용!!!

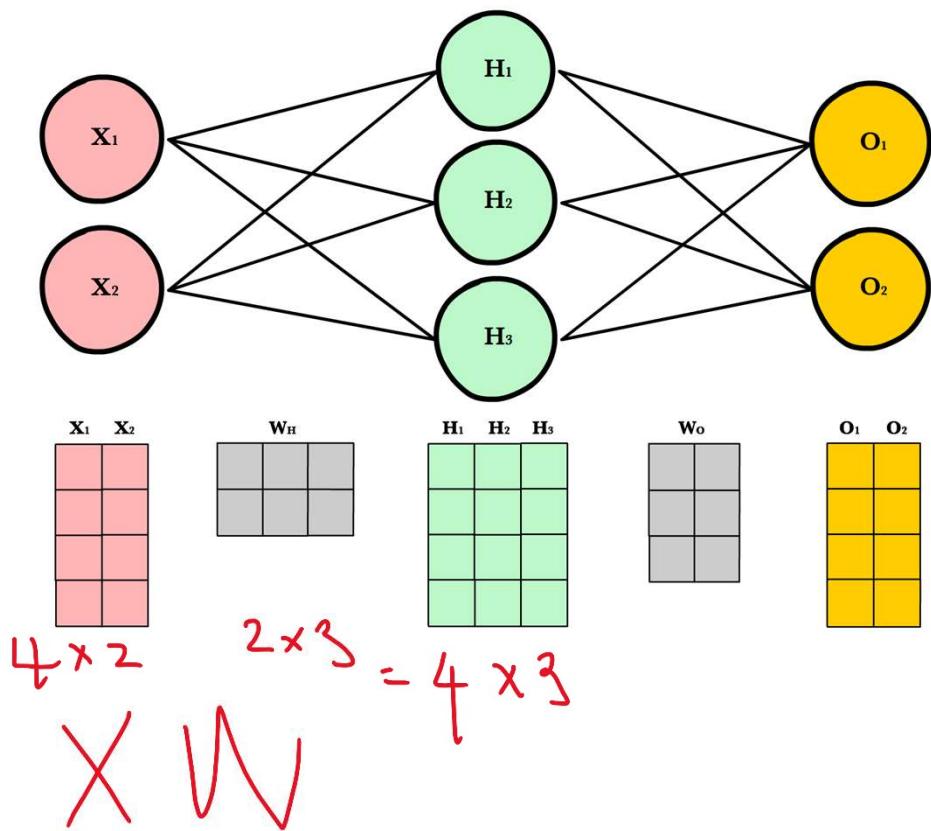


[Feed Forward Network : FNN]

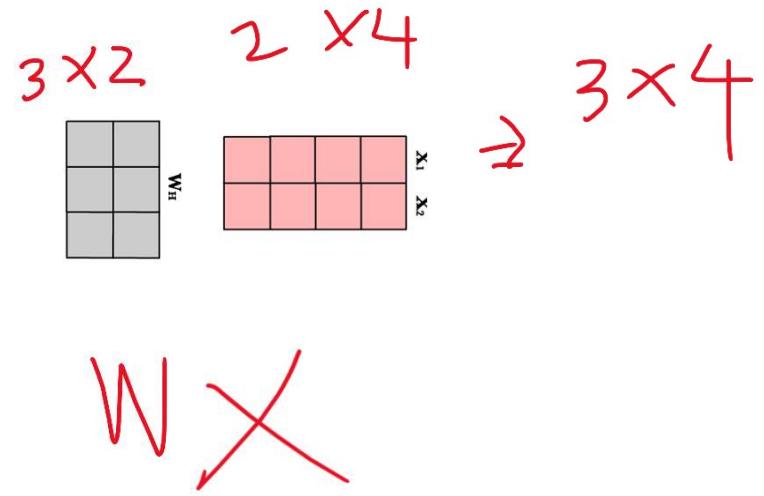


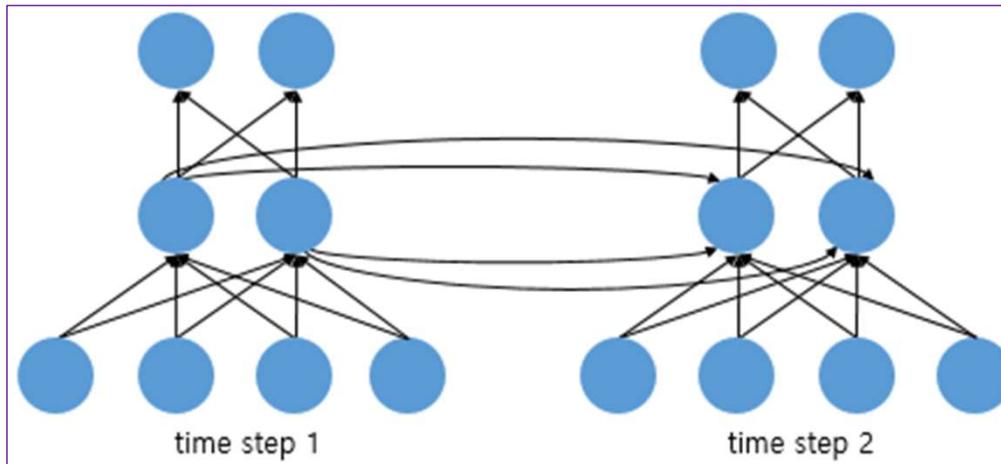
[Recurrent Neural Network : RNN]

[참고] 앞에서 주로 하였을 때에서는 데이터의 순서 중심으로 표기를 하기 위해서 x w 등의 순서대로 표현을 하였지만,
일반적으로는 $W \times$ 등으로 계수를 앞 쪽으로 하는 표현을 함!!
→ 어차피 동일하고, TRANSPOSE 등을 사용하면 됨!!!

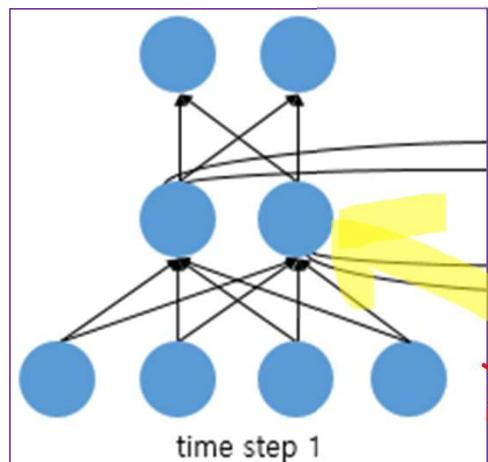


결국에는 동일한 표현들임!!





위의 그림에 대한 수식을 보고 확장을 위해서 일단 왼쪽의 1번 Step에 대해서 보자!

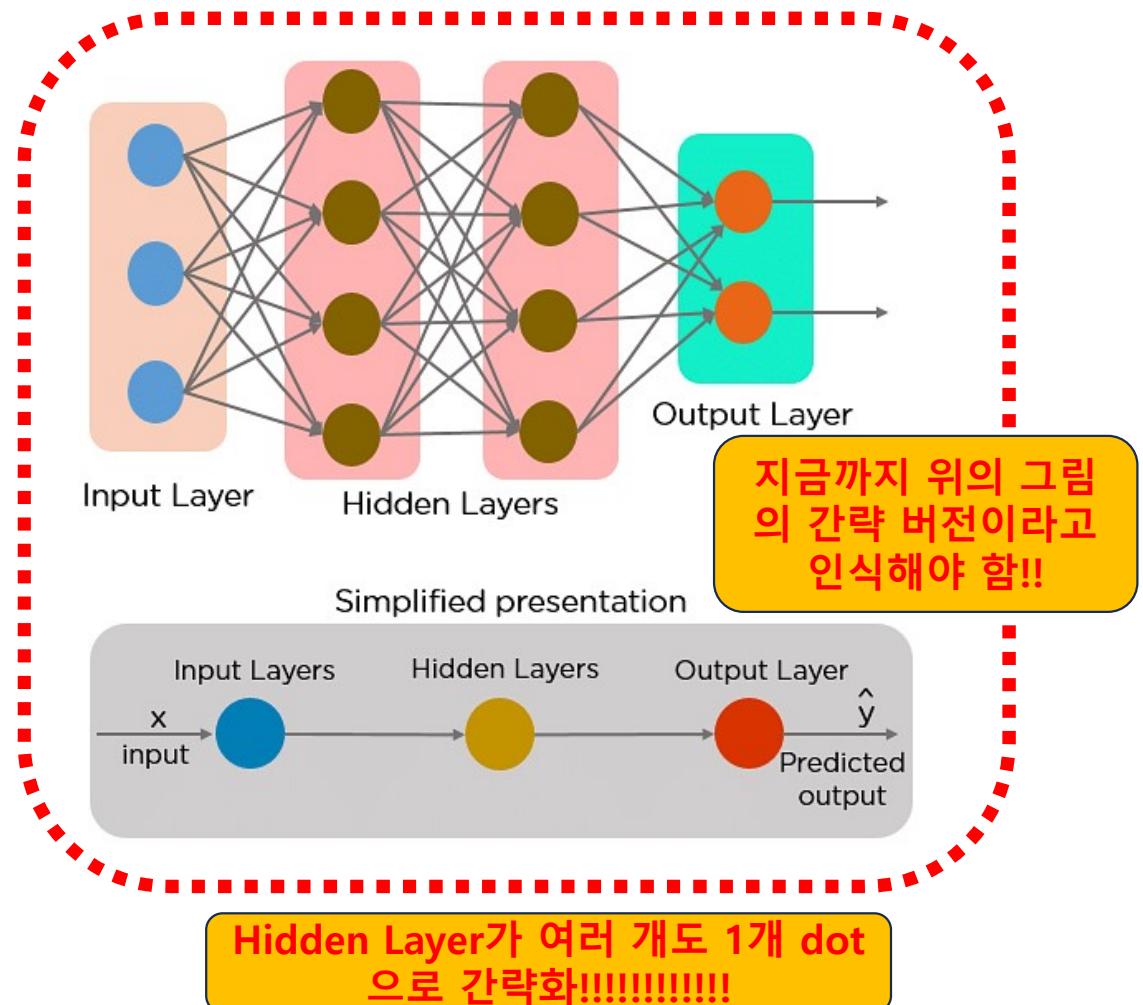
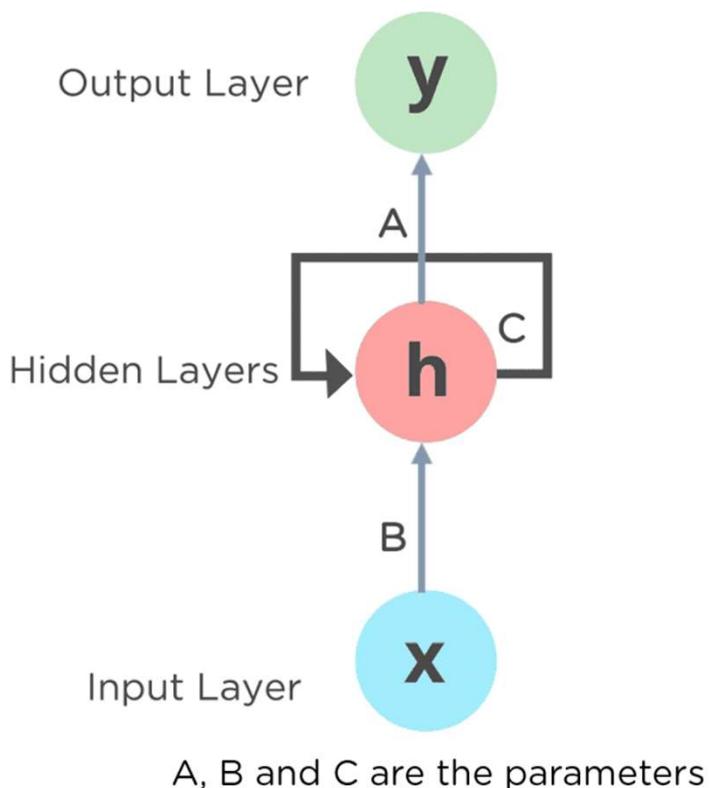


$$h_1 = \tanh \left(\frac{W_x X_1 + b}{\tau} \right)$$

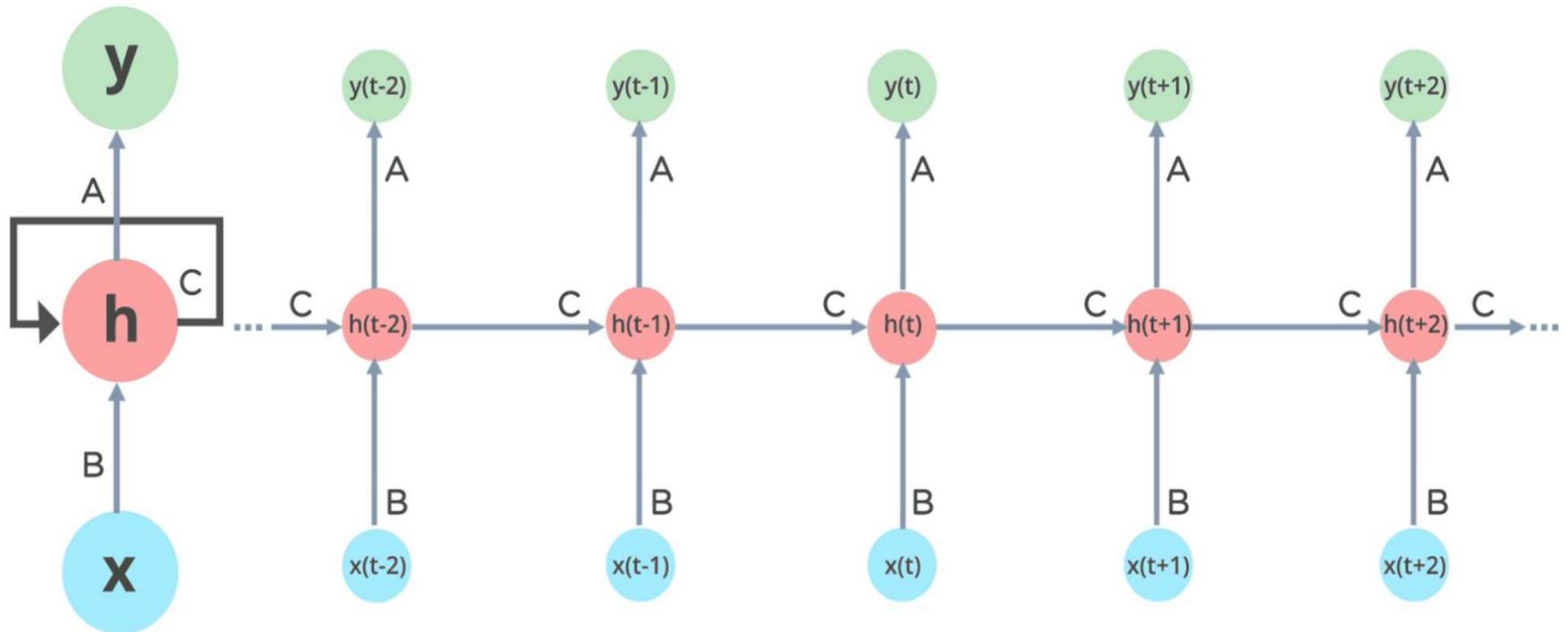
$2 \times 4 \quad 4 \times n$

$n = \text{sample} \Rightarrow 1 : (2 \times 4) \times (4 \times 1) = 2 \times 1$

위의 것들을 옆으로 유연하게 여러개들을 가변적으로 입력을 받을 수 있음!!!

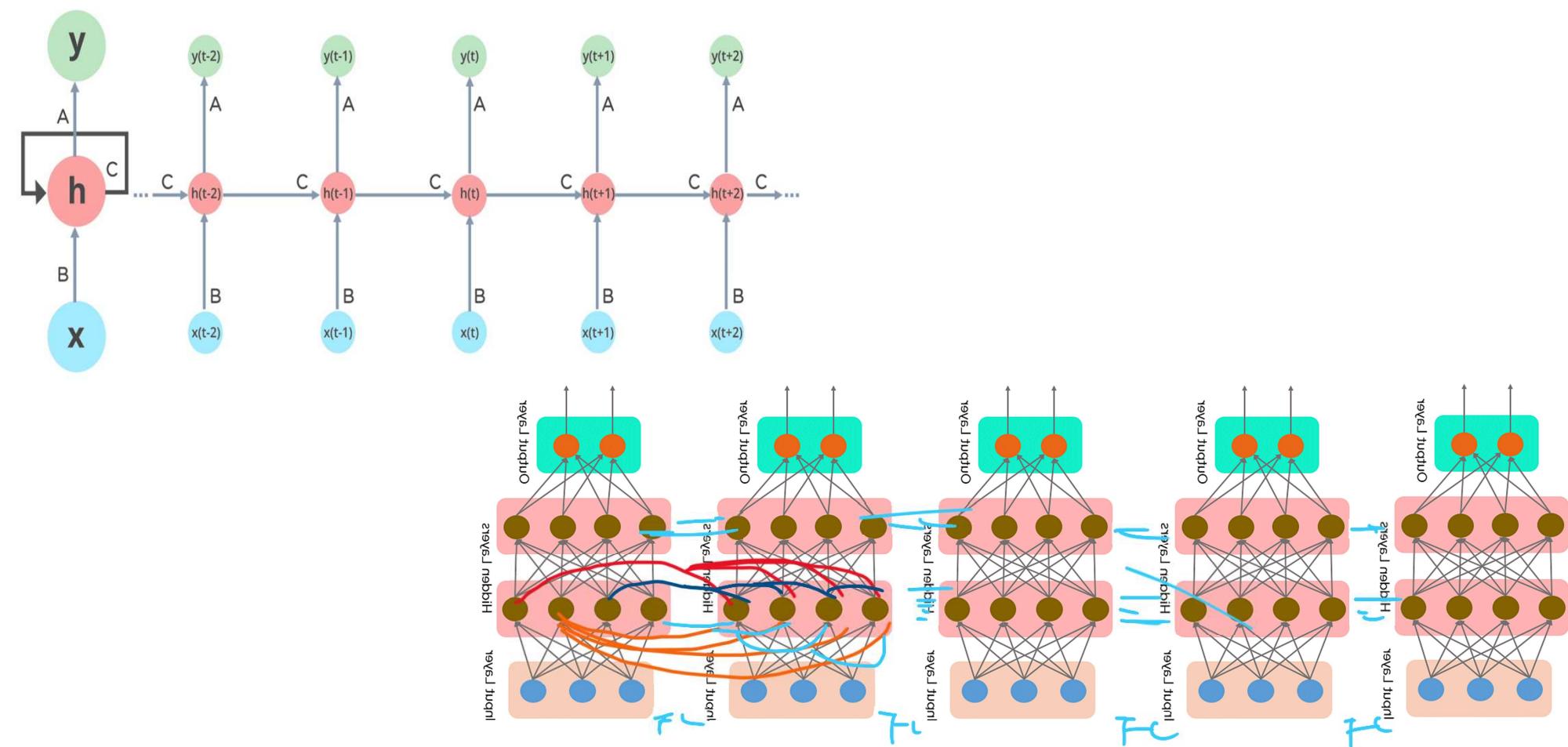


위의 것들을 옆으로 유연하게 여러개들을 가변적으로 입력을 받을 수 있음!!!

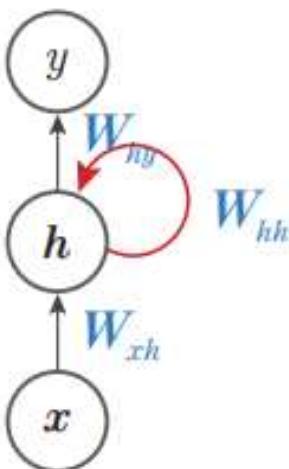


<https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>

위의 것들을 옆으로 유연하게 여러개들을 가변적으로 입력을 받을 수 있음!!!



RNN의 기본 표현 & 수식



$$y_t = W_{hy} h_t$$

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

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

$$= \tanh\left((W_{hh} \quad W_{xh})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right), \quad W = (W_{hh} \quad W_{xh})$$

그림 8-6 기본 순환 신경망 구조

RNN의 핵심 중 하나는 Feed Back 구조라는 점!!!!!!

$$h_1 = f_W(h_0, x_1)$$

$$h_2 = f_W(h_1, x_2)$$

• • •

$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

RNN은 어떻게 Sequence 를 학습할까?

은닉 계층은 '이전 상태, 새로운 입력'을 입력받아서 현재 상태를 매핑하는 함수이다.

$$h_t = f_w(h_{t-1}, x_t)$$

이 식은 h_t 에 대한 점화식 형태이므로 t_0 까지 전개해 보면 다음과 같이 표현된다.

$$h_t = f_w(f_w(\dots f_w(f_w(f_w(h_0, x_1), x_2), x_3), \dots, x_{t-1}), x_t)$$



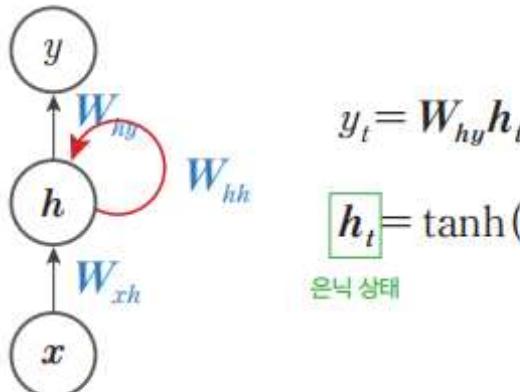
딱 보니 입력의 길이에
따라서 미분이 Back
Propagation에서 문제가
생길 것이 느낌이!!!!

은닉 계층을 나타내는 함수 f_w 는 '추상화된 순차 구조를 한단계 확장된 추상화된 순차 구조로 매핑하는 함수'이다.

$$h_t = f_w(h_{t-1}, x_t)$$

\uparrow \uparrow
 x_1 부터 x_t 까지 순차 구조가 추상화된 상태 x_1 부터 x_{t-1} 까지 순차 구조가 추상화된 상태

RNN의 출력은 무엇?



$$y_t = W_{hy} h_t$$

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

온닉 상태이전 온닉 상태입력 데이터

그림 8-6 기본 순환 신경망 구조

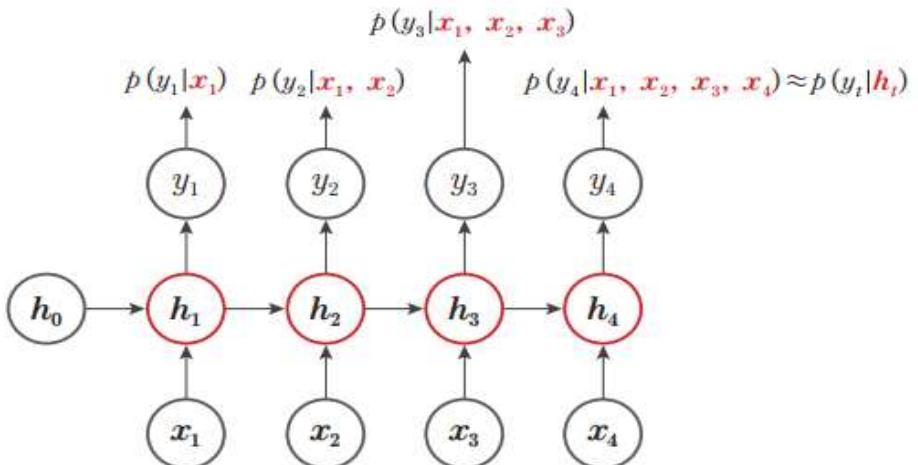


그림 8-7 은닉 계층의 상태에 기억된 입력 데이터

위의 왼쪽은 간략하게 한 것이고, 오른쪽이 실제로 나타나는 것이고, y_t 는 왼쪽의 임의의 t 시전에 대한 것을 의미하고, t 시점에는 앞의 1~ t 시점의 모든 입력이 다 녹아 들어 가게 된다!!! ➔ 오른쪽 그림에서 조건부 확률로 나타나게 됨!!!

Why W_x 에 대한 가중치를 공유를 하는 것일까?

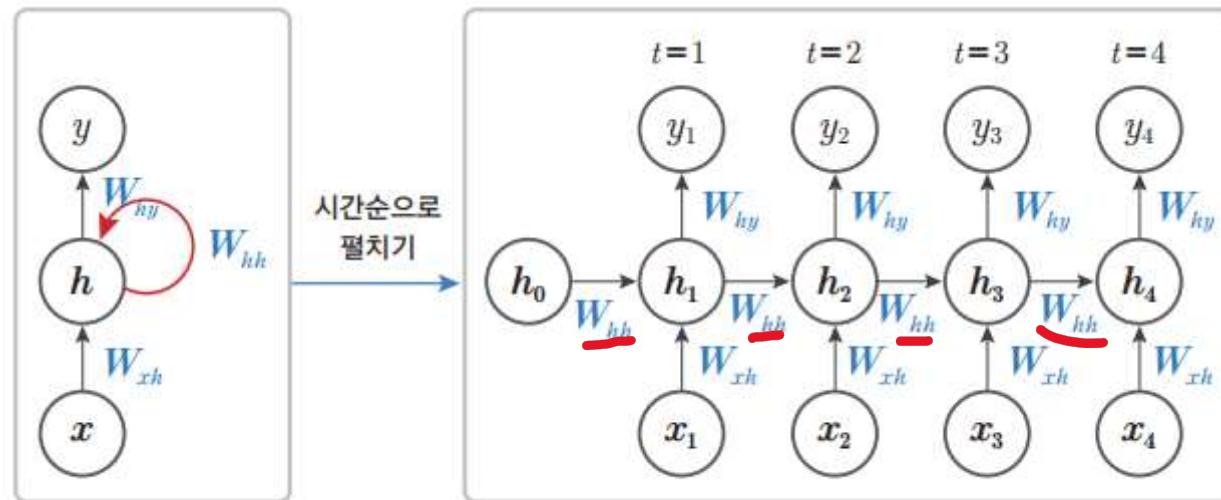
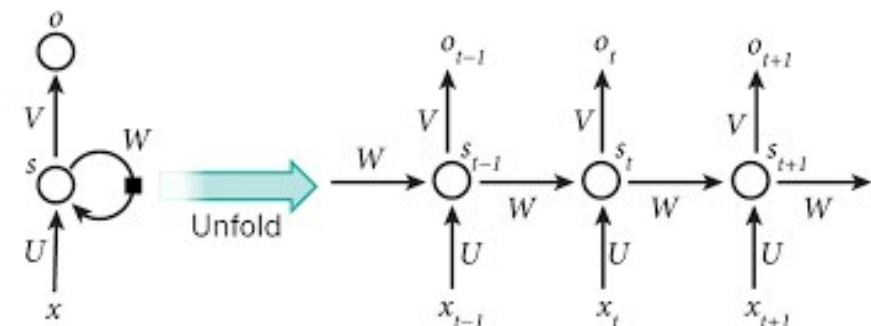
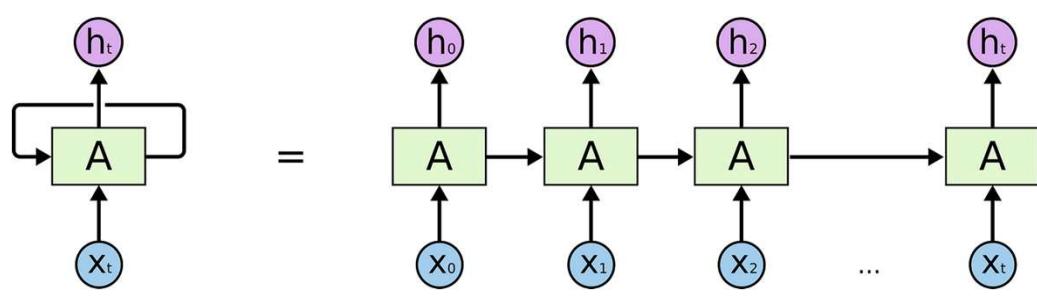
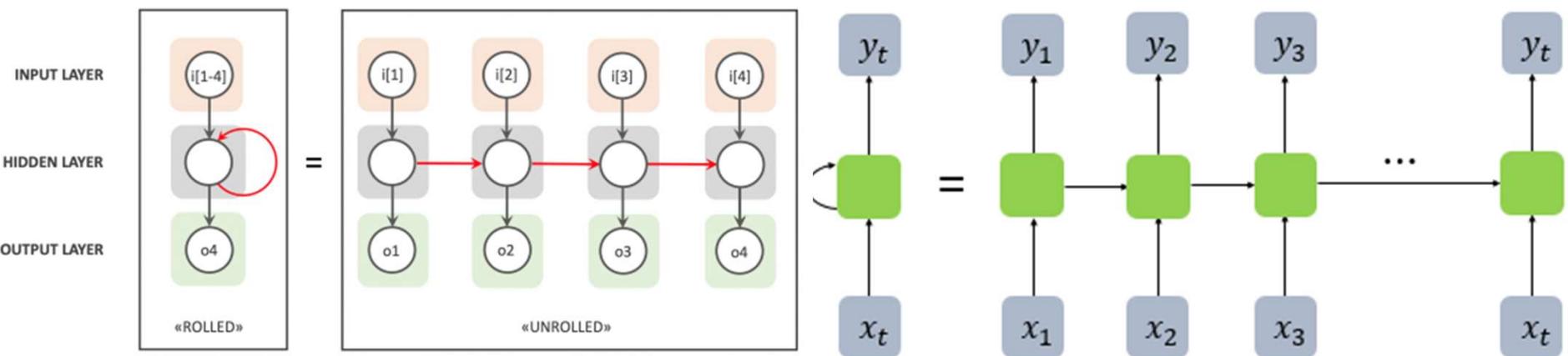


그림 8-8 순환 신경망의 가중치 공유

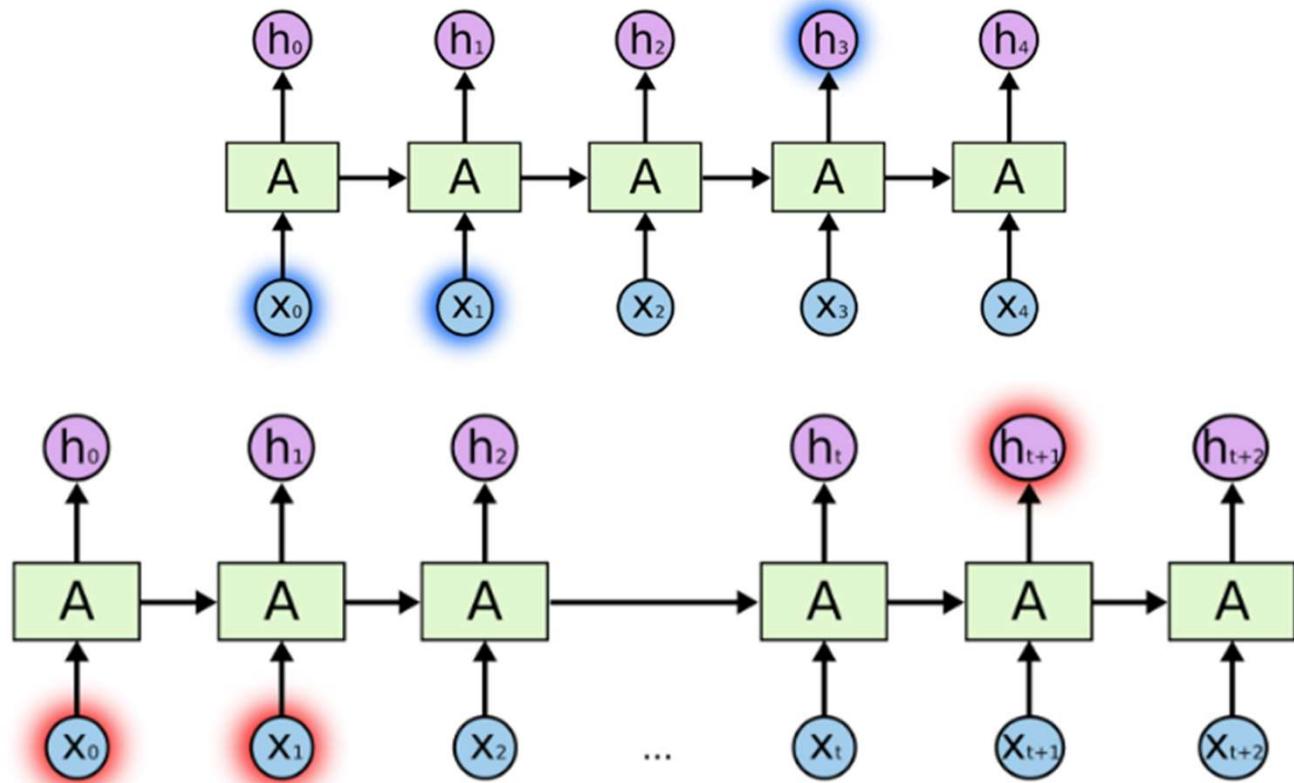
1. 순차 구조를 포착할 수 있음.
2. 가변 길이 데이터 처리가 용이함
3. 파라미터 수가 절약되고, 정규화 효과가 생긴다.

사전 Review : RNN → 다양한 그림 표현 들



사전 Review : RNN의 한계점

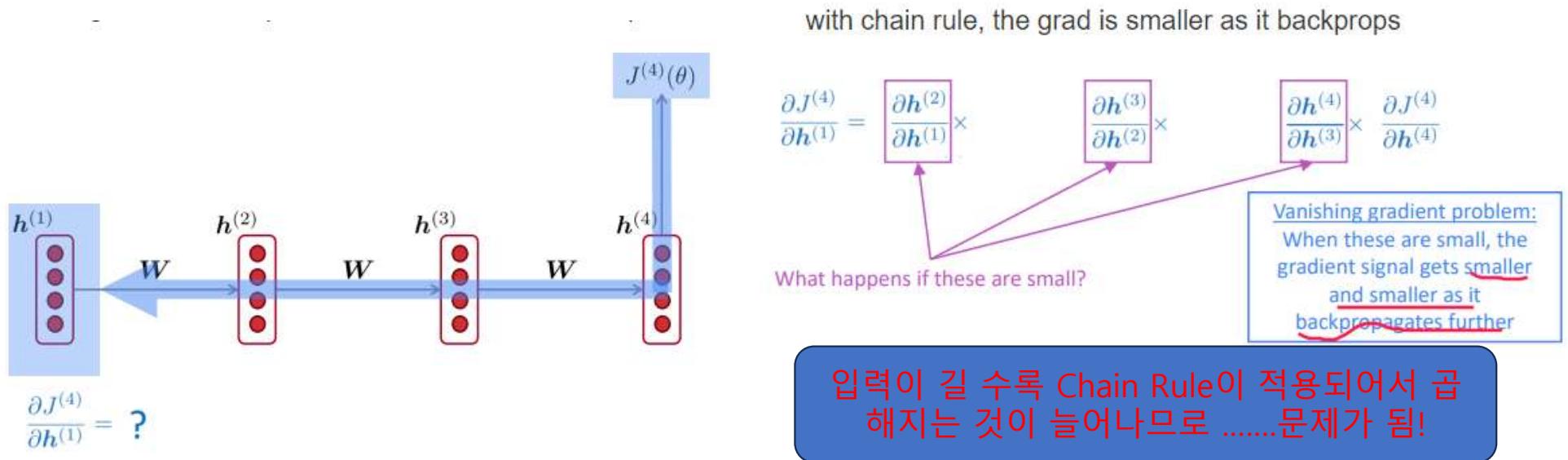
- 장기 의존성 문제 : The Problem of Long-Term Dependencies
 - 일단 Sequence가 일단 길게 되면 Neural Network의 태생적인 문제점인 Vanishing Gradient가 기본적으로 Back Propagation에서 발생을 하고(대안으로 하는 ReLU 계열을 잘 안 쓰기에,,,)
 - 중요한 정보가 맨 앞에 있을 경우에 그 정보가 적게 들어가게 됨.(애플 주식의 가격은 내일은 얼마가 될까? → 중요 정보 '애플'이 제일 먼저 가 되는데, 문제는 수식에서 tanh를 통해서 나온 값들을 계속 처리하게 (Recurrent 구조)되니, 계속 작게 된다, 좀만 길어져도 저 멀리 영향력은 티끌이 되기에....) → LSTM, GRU 등 개선으로 나타나게 됨!!! → LSTM, GRU가 확 주목을 받았지만,,, Transformer로 인기를 넘겨 줌!!!



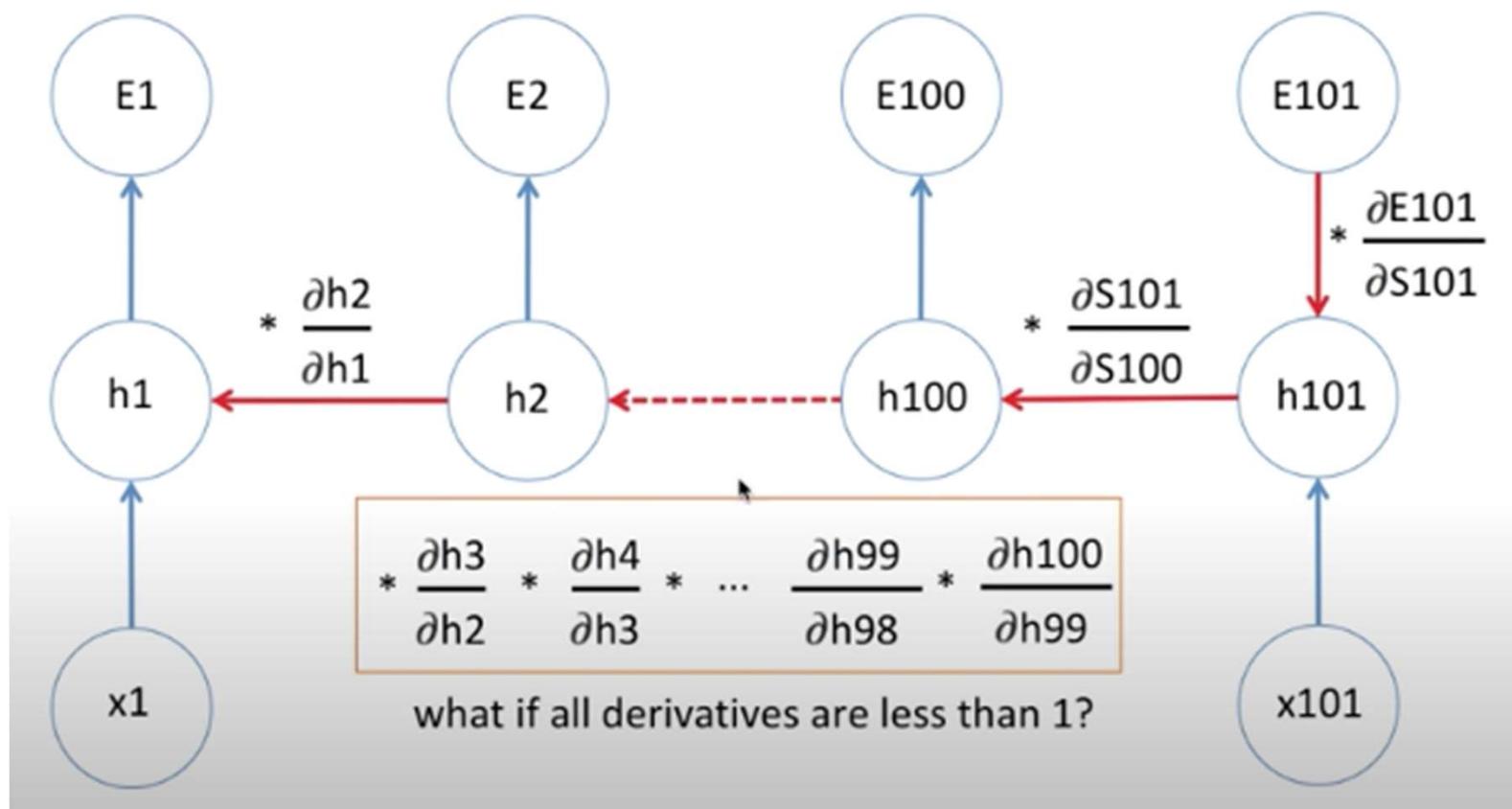
<관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 RNN 학습능력 저하>

RNN에서 Back Propagation은 무엇?

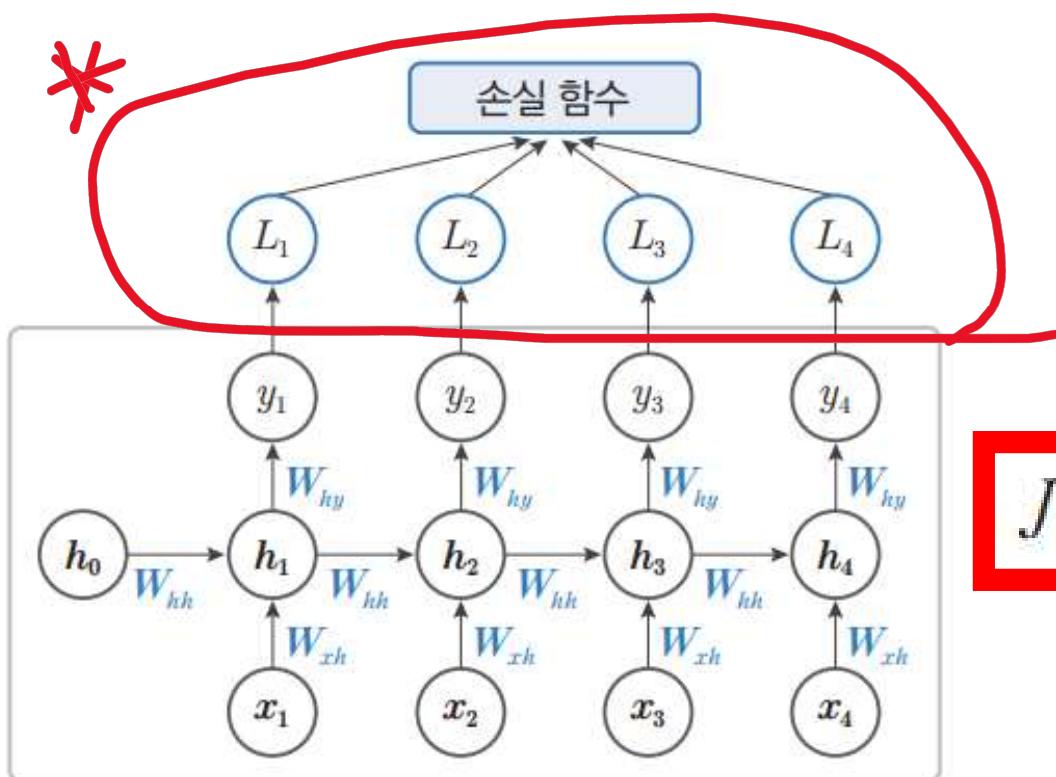
- 일단 출력을 중심으로 입력의 역순으로 진행을 하면 된다!!!! →
시간 순서대로 펼쳐 놓은 상태에서 수행하므로, 시간펼침 역전
파라고 하여서 **BPTT(Back Propagation Through Time : BPTT)**라는 약어를 사용함!!!



극단적으로 입력이 101개 이상인 경우에 대해서의 그림
임!!!!



What is Loss in RNN?



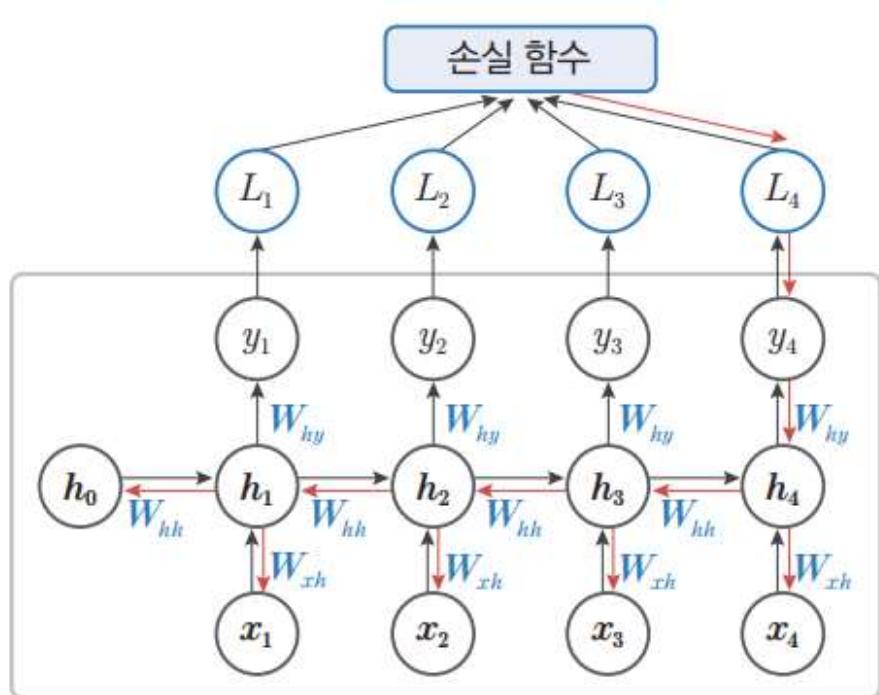
* RNN의 전체적인 Loss에 대해서는 각 시점의 오류들을 싹 합쳐서 하나로 처리를 해야하므로 왼쪽 그림과 같이 각 시점의 Loss들을 하나로 합쳐서 처리를 해야 함!!!!

$$J(f(X; \theta), t) = L_1 + L_2 + L_3 + L_4$$

참고) 교재의 편의상 $t=4$ 로 4개의 순차에 대한 것이지, t 로 일반화 해서 늘리기만 하면 됨!

그림 8-15 순환 신경망의 손실 함수

BPTT에 대해서...



- 왼쪽에서는 가정 : 입력이 4개로 단순화 함
- 마지막 $t=4$ 에서의 역전파를 알아보려고 함!!!
→ RNN 전체는 $t=4$ 1개만의 아니라 $L_1 \sim L_4$ 까지의 전체의 합을 줄여야 하는 것임과 구별 해야 함!!!
- 왼쪽의 붉은색 : 역전파

그림 8-16 마지막 단계 역전파

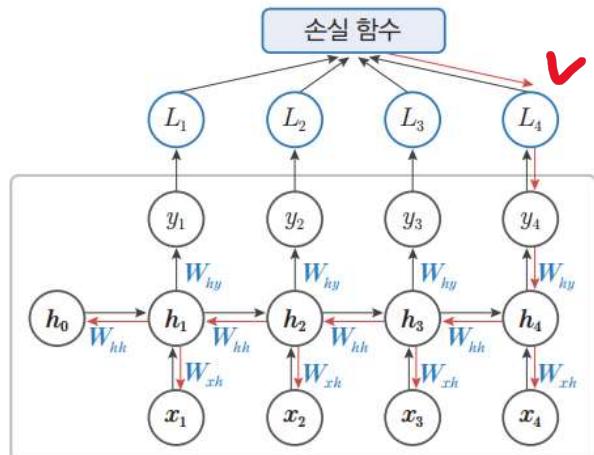


그림 8-16 마지막 단계 역전파

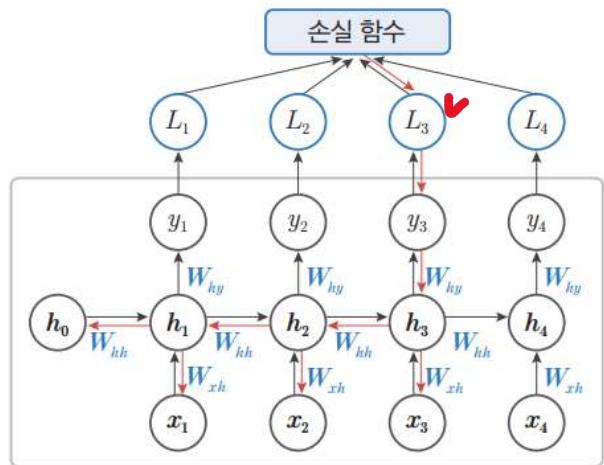


그림 8-17 세 번째 단계 역전파

개별 단계 BPTT

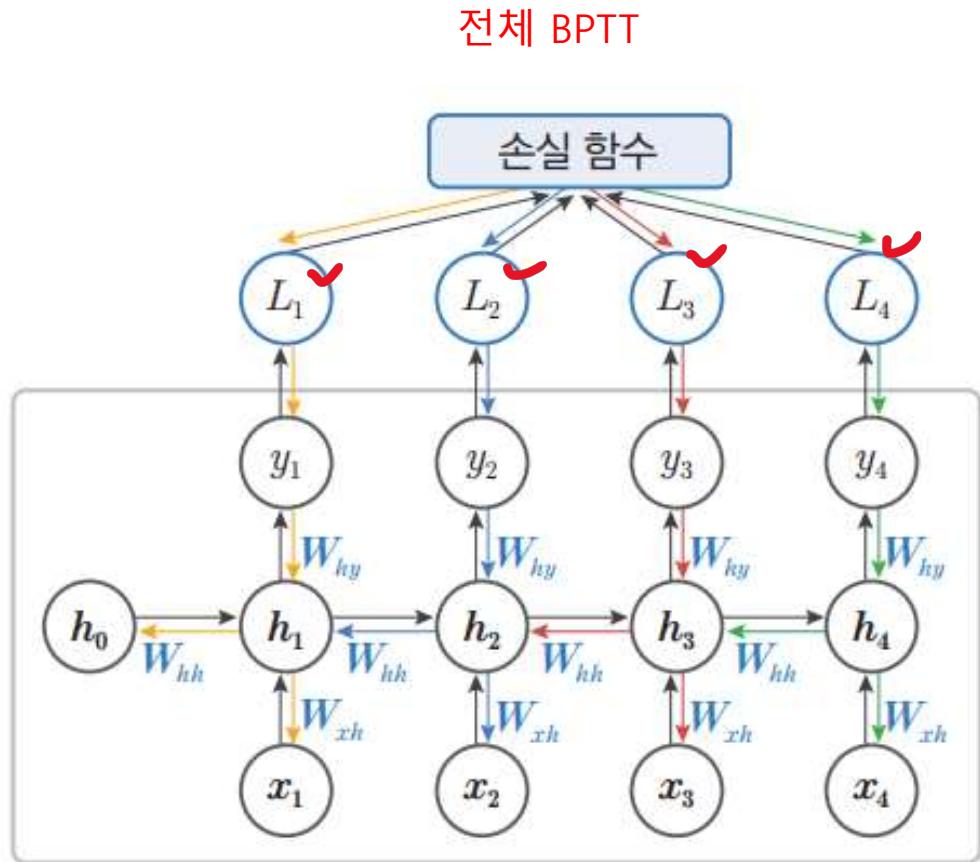
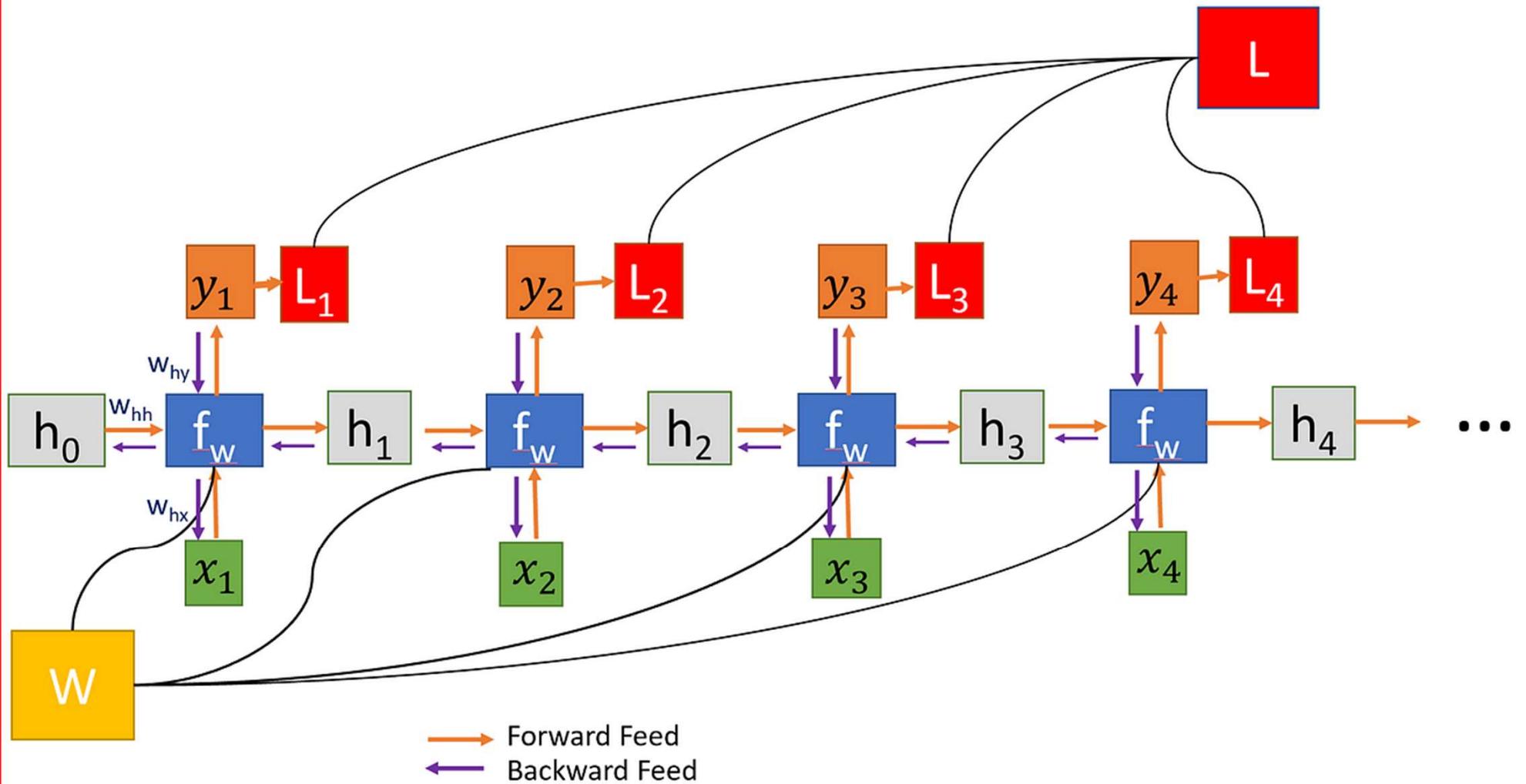


그림 8-18 단계별 역전파

전체 Loss에서 역전파가 시작되어서 모든 단계로 동시에 진행이 되



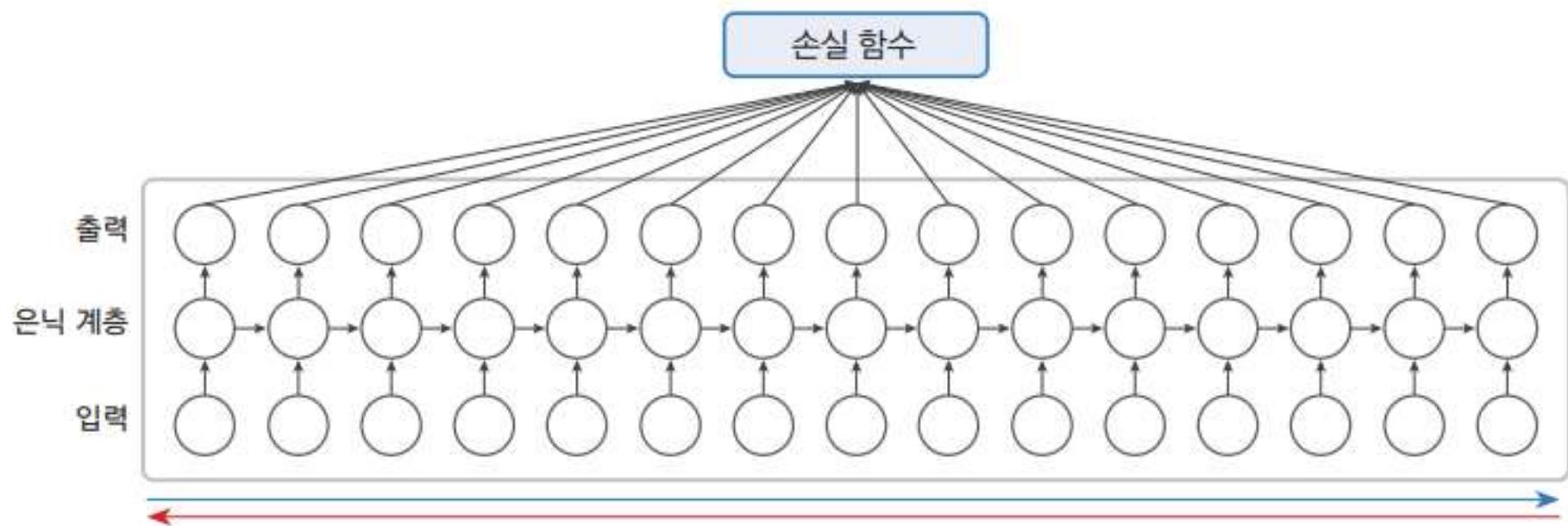


그림 8-19 시간펼침 역전파

위와 같이 길어지게 되므로,
끝이 없는 순서 데이터나
아주 긴 입력 데이터에
적용에는 문제가 발생할 수 있음!!!!

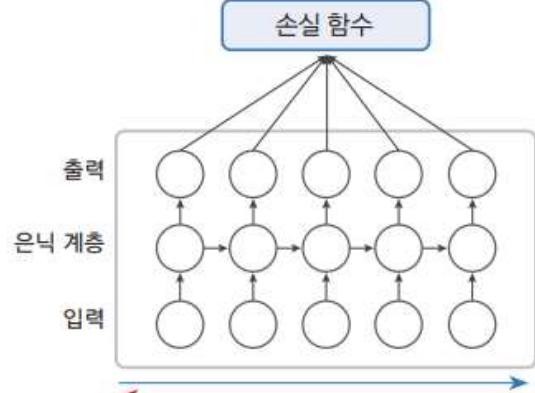


그림 8-20 절단 BPTT 첫 번째 묶음 실행

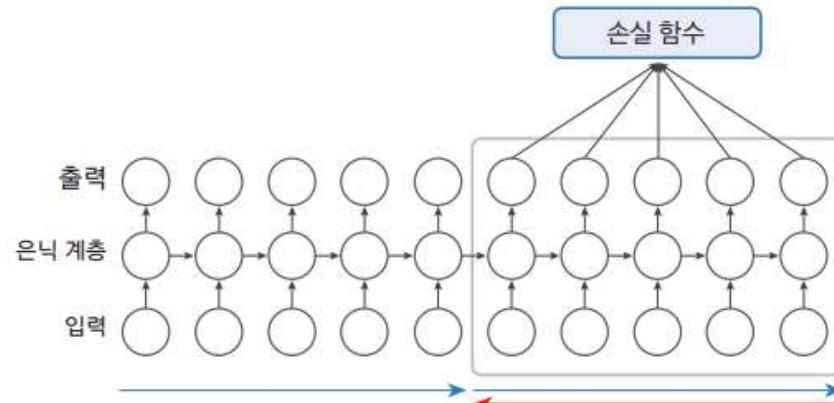


그림 8-21 절단 BPTT 두 번째 묶음 실행

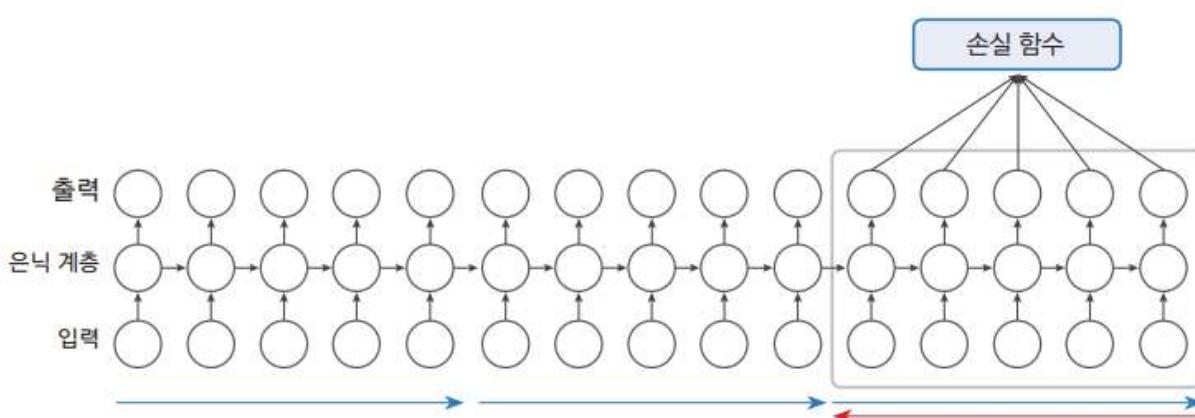


그림 8-22 절단 BPTT 세 번째 묶음 실행

위의 경우와 같이 너무 길어서 일부분씩 짤라서 나눠서 묶어서 진행을 할 수 있음!!
 → 그럼에도 불구하고 최적화가 어렵고, 설득적인 한계가 존재하게 됨!!!

1. 입력 시간이 지나면서 입력 데이터의 영향이 사라지는 “장기의존성 문제”
2. Gradient Vanishing // Eplotion

RNN 의 한계점1

The Problem of Long-Term Dependencies

- The Problem of Long Term Dependencies : RNN의 Hidden Layer에서 과거의 정보가 최종까지 잘 전달이 안 된다는 것을 의미하는 것!!! (결과 → 원인은 Gradient Exploding or Gradient Vanishing)

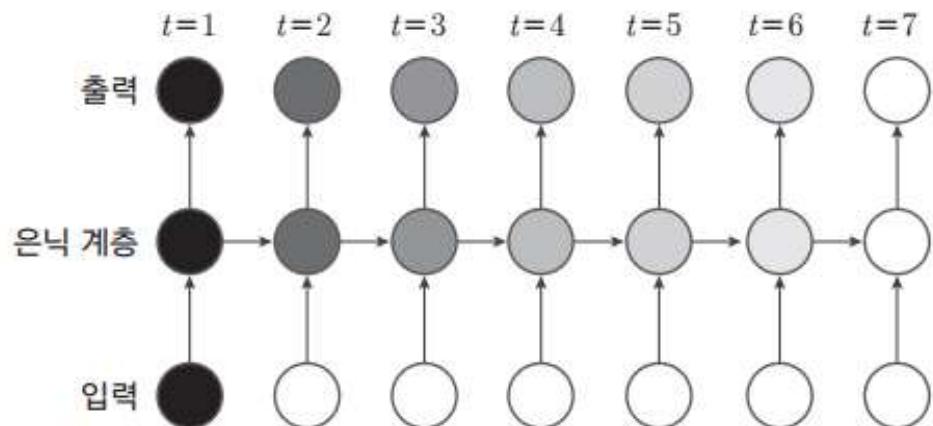


그림 8-23 순환 신경망의 장기의존성 문제

간단하게 예를 든 것인데
“애플 주식 다음 달 증권사 예상 어때?”라는 것
에서 가장 처음에 있는 “애플”이라는 종목명이
상당히 중요한 정보임에도 불구하고, 마지막으로
가면 이 정보의 정보가 제대로 전달이 안 되는 것
임!!!!!!
→ 정보의 중요성이 아니라 정보의 위치가 영향
을 크게 받게 되고, 중요한 정보가 앞 쪽에 있을
수록 잘 반영이 안 될 수 있는 단점!!!

애플 주식 다음 달 증권사 예상 어때?

RNN 의 한계점1

The Problem of Long-Term Dependencies

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right), \quad W = (W_{hh} \quad W_{xh}) \end{aligned}$$

tanh tanh tanh .

$$h_t = f_W(f_W(\dots f_W(f_W(h_0, x_1), x_2), x_3), \dots, x_{t-1}), x_t)$$

$$h_t = f_W(h_{t-1}, x_t)$$

위의 수식에서 보면 계속 tanh가 곱해지는 것을 볼 수 있음!!!!
 → 그래서 Gradient의 Exploding or Vanishing 현상이 나타나고
 → 학습이 잘 안 되는 그래서 장기 의존성 문제가 발생을 함!!!

→ 뒤 페이지에 좀 더 자세히 있음!!!

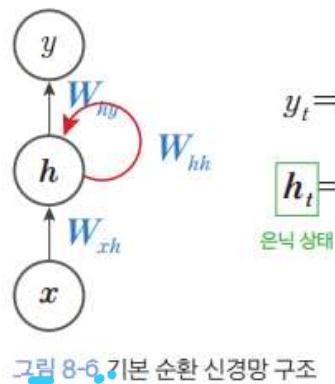
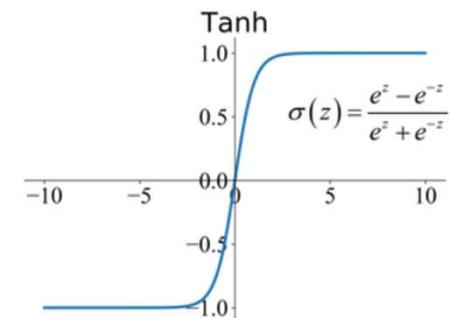


그림 8-6 기본 순환 신경망 구조

$$\begin{aligned} y_t &= W_{hy}h_t \\ h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \end{aligned}$$

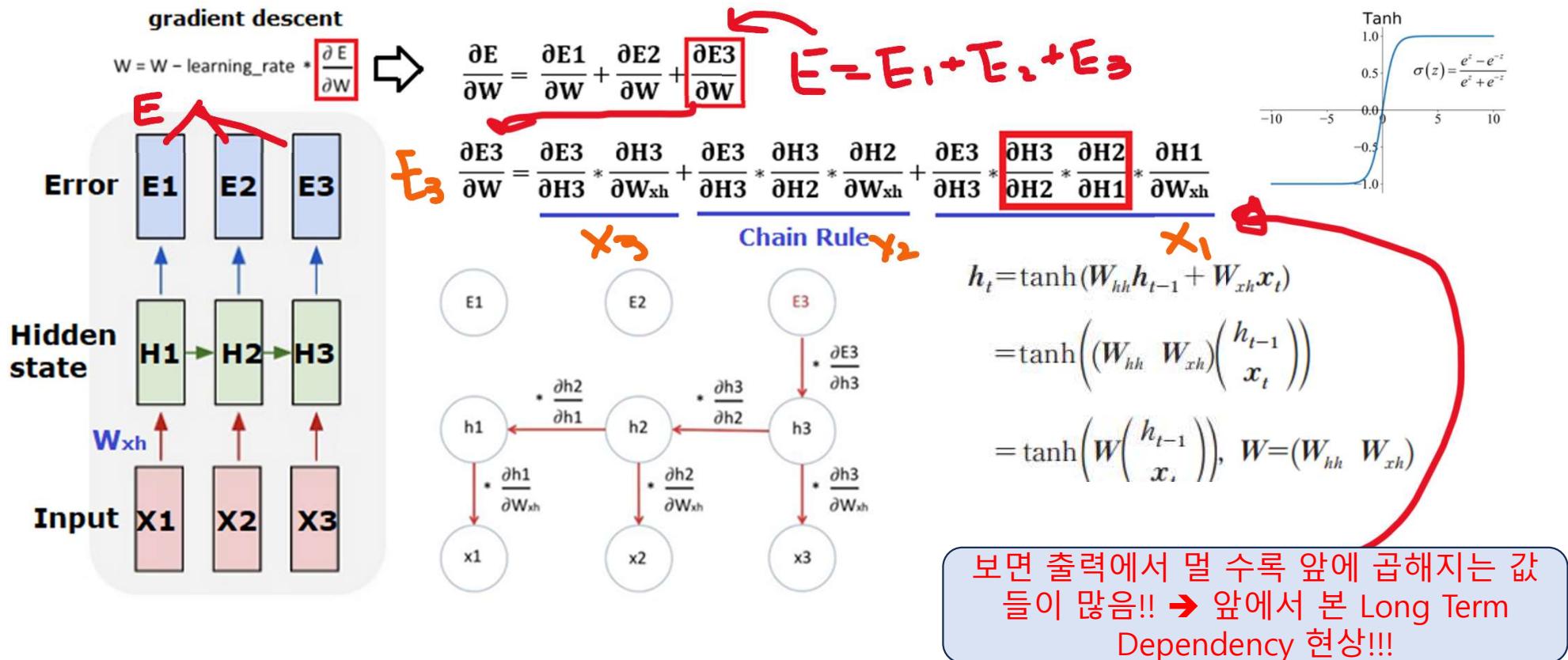
은닉 상태 이전 은닉 상태 입력 데이터

$$\begin{aligned} h_1 &= f_W(h_0, x_1) \\ h_2 &= f_W(h_1, x_2) \\ &\dots \\ h_t &= f_W(h_{t-1}, x_t) \end{aligned}$$



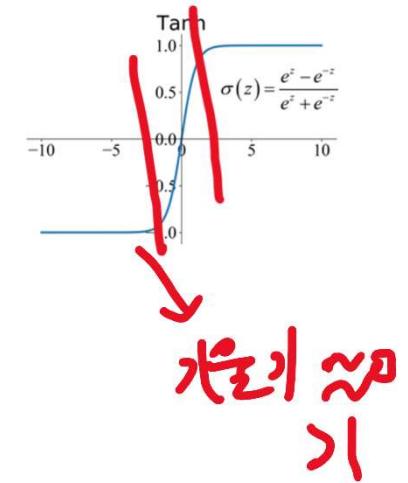
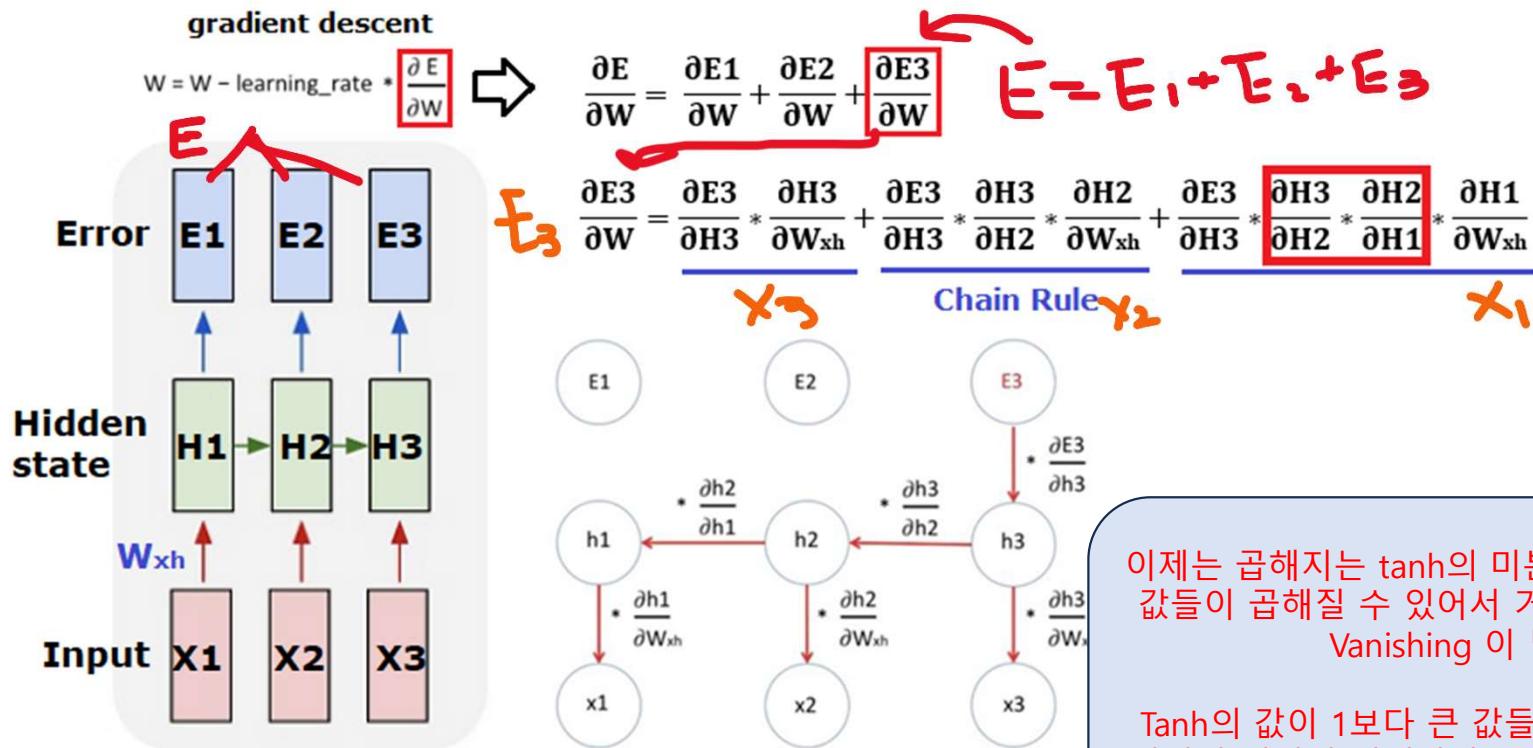
RNN 의 한계점2

Gradient Exploding & Vanishing!!



RNN 의 한계점2

Gradient Exploding & Vanishing!!

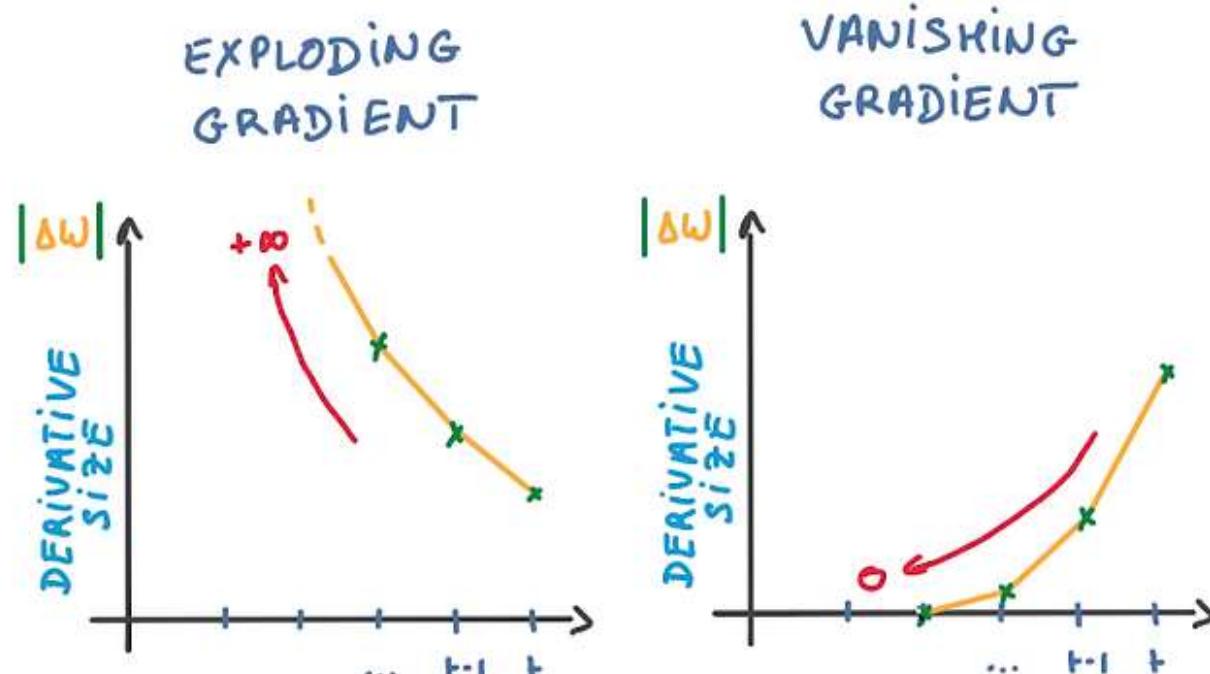


이제는 곱해지는 tanh의 미분값에 대해서 보면 1보다 작은 값들이 곱해질 수 있어서 거의 0으로 가게 되는 Gradient Vanishing 이 벌어질 수 있고!!!

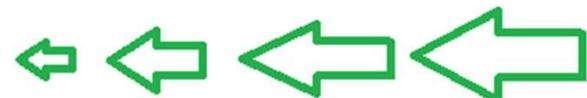
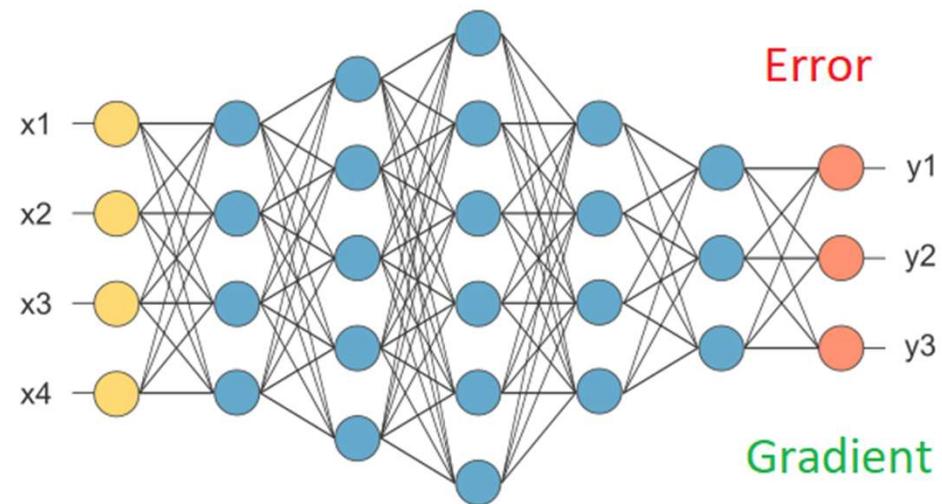
Tanh의 값이 1보다 큰 값들이 곱해지게 되어서 엄청나게 커지게 되어서 터져나갈 수 있는 Gradient Exploding이 벌어질 수 있음!!!!

RNN 의 한계점2

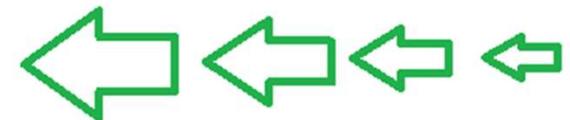
Gradient Exploding & Vanishing!!



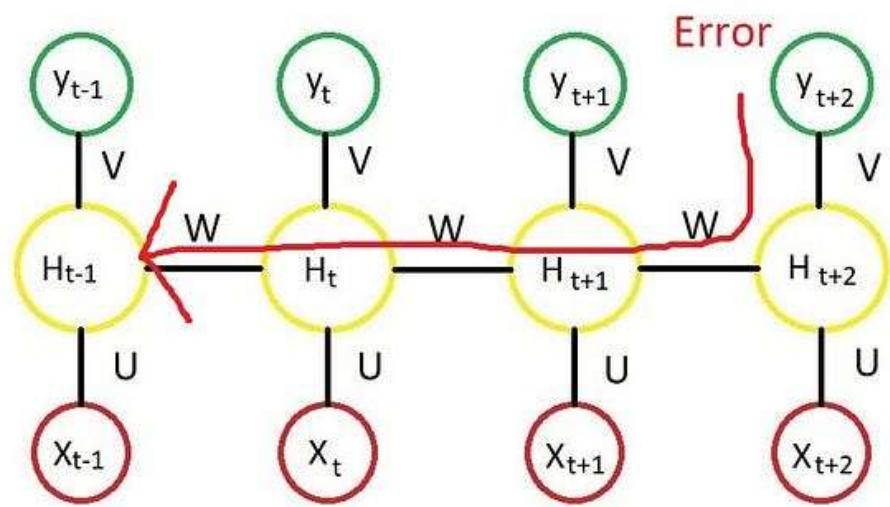
1994년 Bengio 교수의 Learning Long Term Dependencies with Gradient Descent is Difficult 논문에 의하면, 이 문제는 쉽지 않고, 해결하기가 어렵다고 한 부분임!!! → 다른 구조의 모델들을 사람들이 제시를 하고, 발전 시키게 됨!!!



Vanishing Gradient



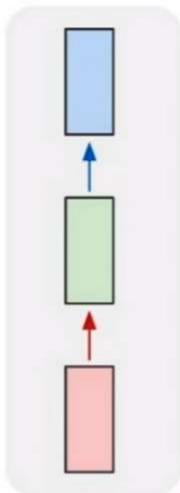
Exploding Gradient



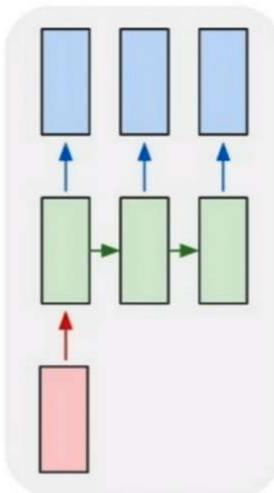
if $|W| < 1$ (Vanishing)
 $|W| > 1$ (Exploding)

사전 Review : RNN 여러 Type

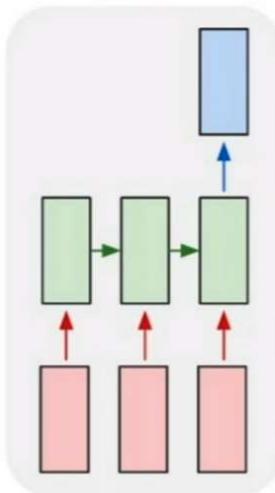
one to one



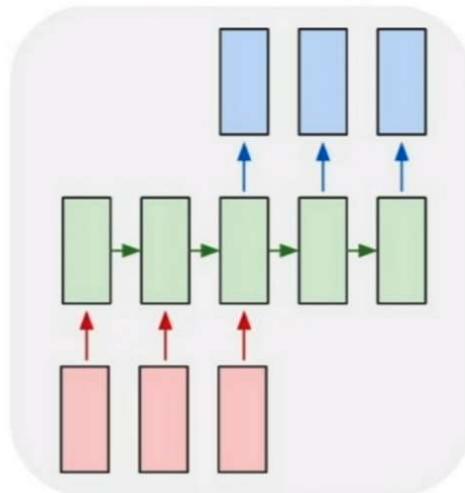
one to many



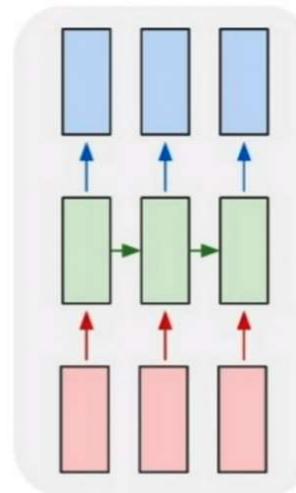
many to one



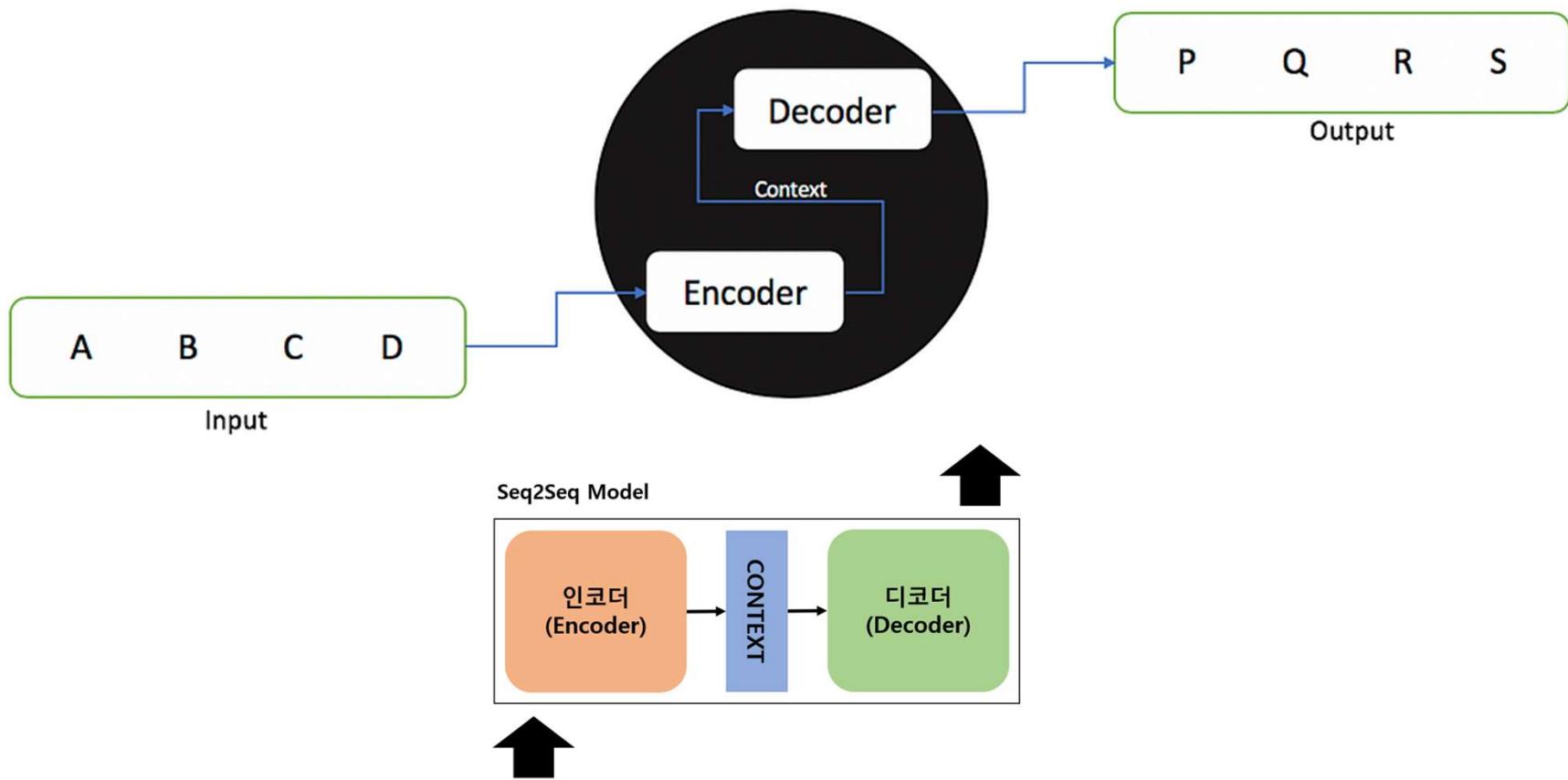
many to many

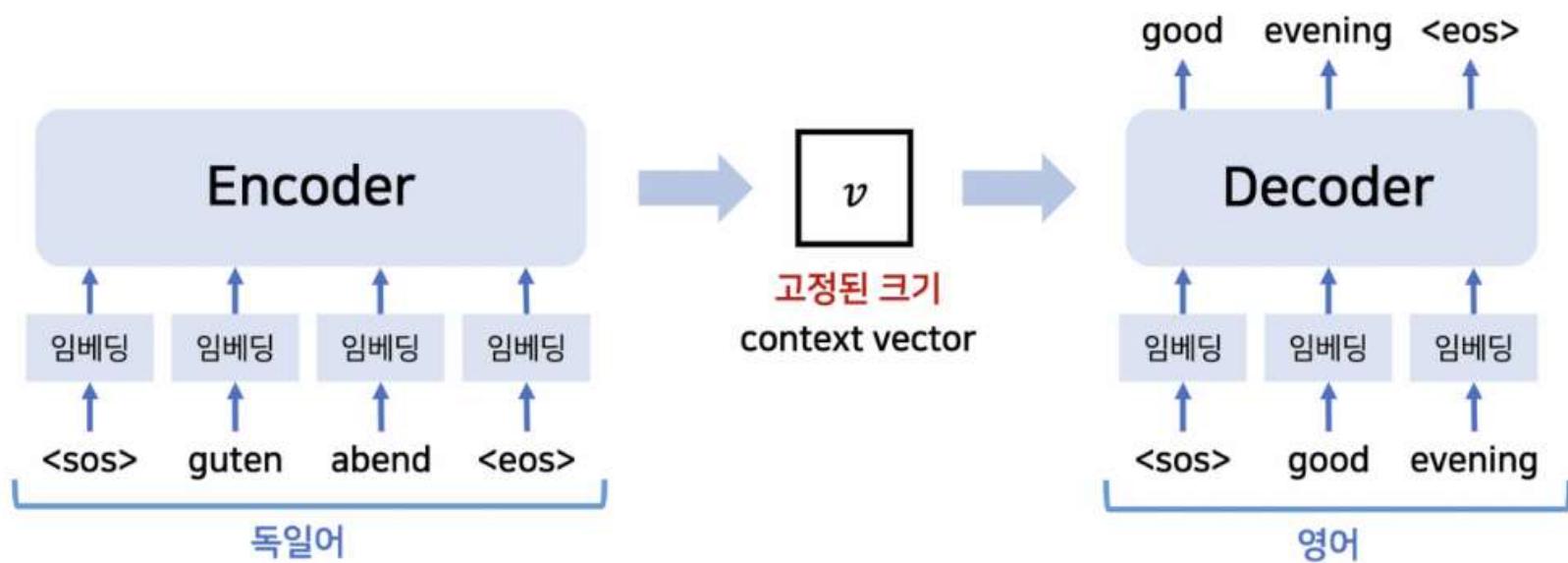


many to many



사전 Review : Seq2Seq 구조의 RNN





(출처) <https://github.com/ndb796/Deep-Learning-Paper-Review-and-Practice>

LSTM & GRU

- 앞의 기본적인 순환 신경망 : Vanilla RNN의 문제점
 - 최적화가 어려움
 - 장기기억 문제 발생
 - Gradient Exploding & Vanishing
- 대안으로 제시가 되어서 인기를 끌어던 것들이 LSTM / GRU가 나왔고, 그 이후에 Transformer로 시대의 흐름이 넘어감!!!!

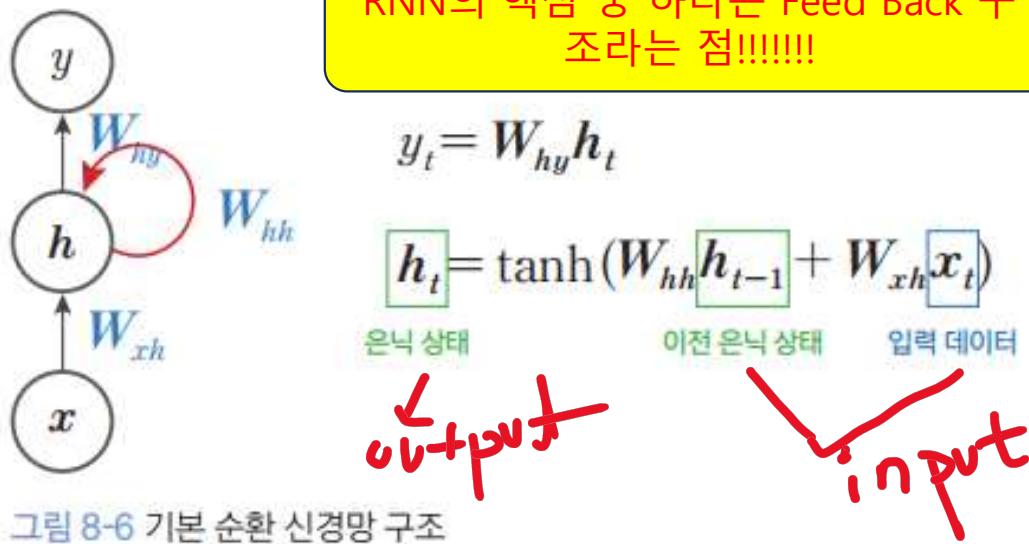
LSTM : Long Short Term Memory

- RNN에서 앞에서 문제가 된 것은 Gradient Vanishing 문제가 발생을 한 것임! → 이전의 정보를 잘 깨먹는다는 문제가 발생을 하는 것임!!!



- Lstm 관련해서 2015년에 작성된 글이 아직도 유명함. 이유는 그 때 잘 작성된 글이 아직도 사람들이 인용하고 하기에 그 그림이 인터넷에 많이 보이게 됨.
- Ref) <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

참고: 다시 RNN의 기본 표현 & 수식 상기!!!



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right), \quad W = (W_{hh} \quad W_{xh}) \end{aligned}$$

$$h_1 = f_W(h_0, x_1)$$

$$h_2 = f_W(h_1, x_2)$$

...

$$h_t = f_W(h_{t-1}, x_t)$$

이제 위의 식에서 input 과 output 관점으로 바라봄!!!
뒤에는 이런 관점으로 lstm으로 넘어가기 위해서 조금
다른 그림으로 볼 것임!!!!

Rnn의 다른 그림

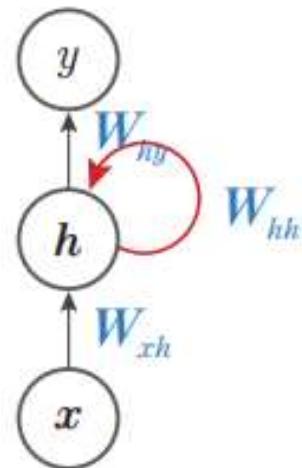


그림 8-6 기본 순환 신경망 구조

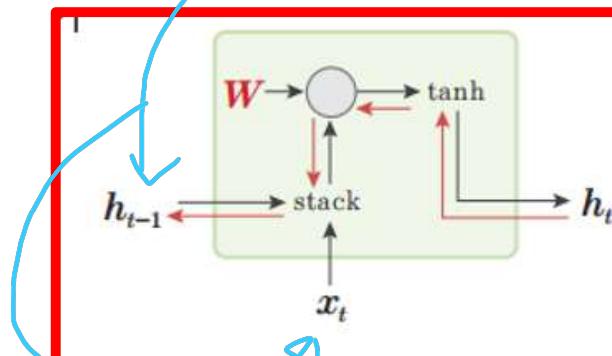
$$y_t = W_{hy} h_t$$

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

은닉 상태

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

이전 은닉 상태 입력 데이터



$$h_t = \tanh((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) = \tanh(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix})$$

그림 8-24 기본 순환 신경망의 그레이디언트 흐름

Lstm 이 아니고 rnn을 lstm 틱하게 그린 그림임!!!

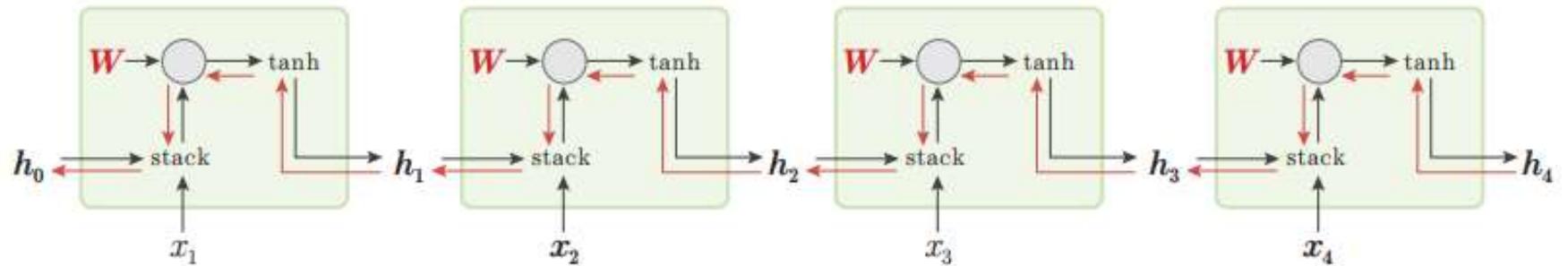


그림 8-25 연속적인 그레이디언트 흐름

핵심 : 각 은닉층 단계마다 W 가 곱해
지는 것을 볼 수 있음!!!

역전파를 할 때에도 미분 W 가 각 단계
마다 곱해진다!!!

→ W 의 크기에 따라서 1보다 크면 발
상, 1보다 작으면 0으로 수렴

→ Gradient Explosion & Vanishing

Gradient Clipping

- 앞에서 일어나는 Gradient Explosion은 비교적 간단하게 막을 수 있음! → Gradient Clipping임!!!
- Gradient를 일정 수준으로 재조정을 해주는 것임!! → 이유는 결국에는 다음에 어디로 가서 최적인지 찾아보는 보폭이니, 그 보폭에 대한 것을 조절을 해주면 되는 것임.. 우리가 learning rate로 조절하는 것과 같이..

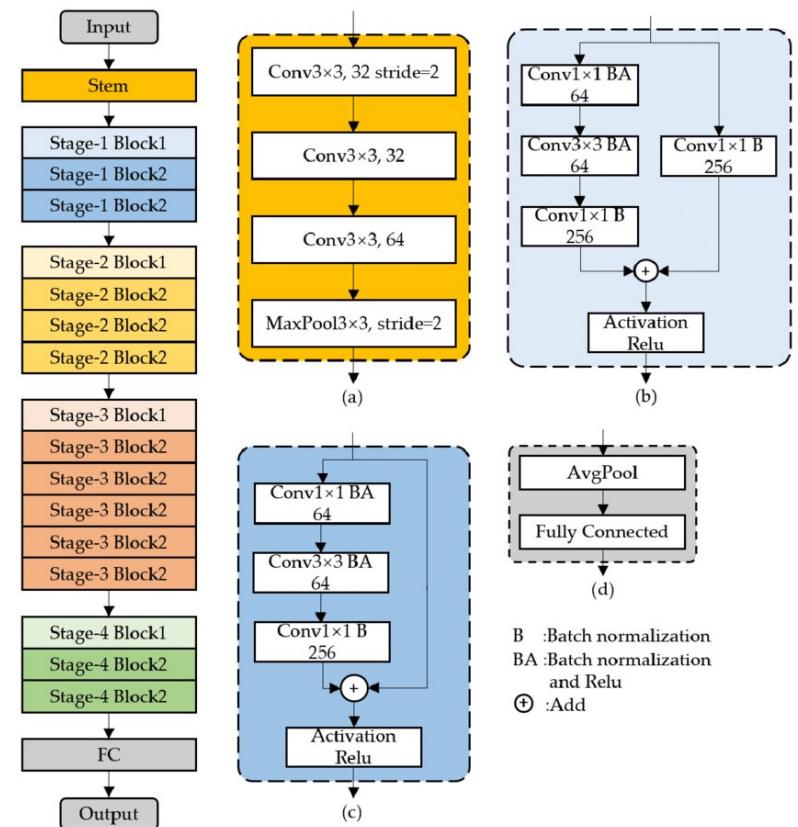


ResNet의 아이디어를 여기서도??



ResNet의 ShortCut??

→ Gradient 보존& 보상처리!!!



→ LSTM은 그런 이유로 Gradient 소실의 원인이 되는 가 중치 W와의 행렬곱 연산이 Gradient 경로에 나타나지 않도록 구조 변경!!!

기존 RNN vs LSTM

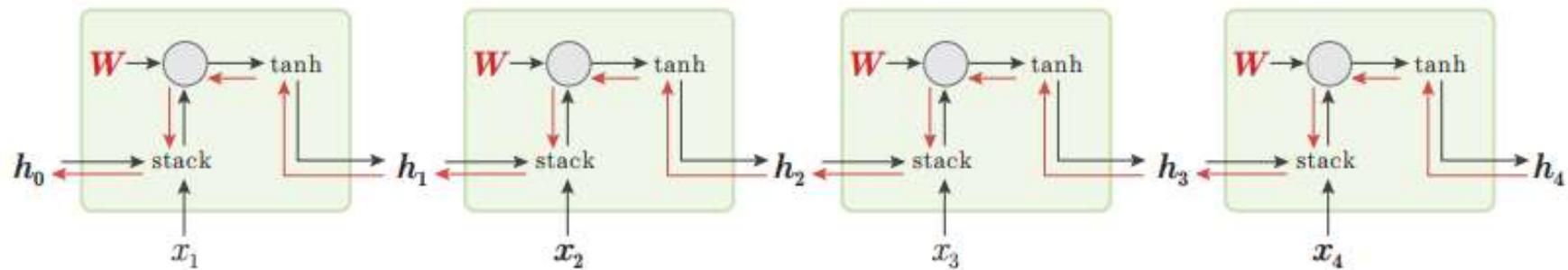


그림 8-25 연속적인 그레이디언트 흐름

셀들의 상태를 연결하는 경로에는 W 와 행렬곱이 없음!!!
 → 순방향과 역방향이 순조롭게 왔다 갔다 할 수 있고, Long Term Dependency & Gradient Vanishing 문제가 어느 정도 완화가 됨!! 100퍼센트 해결이 아님! → 별도 경로를 두는 ResNet과 유사한 아이디어 방식!

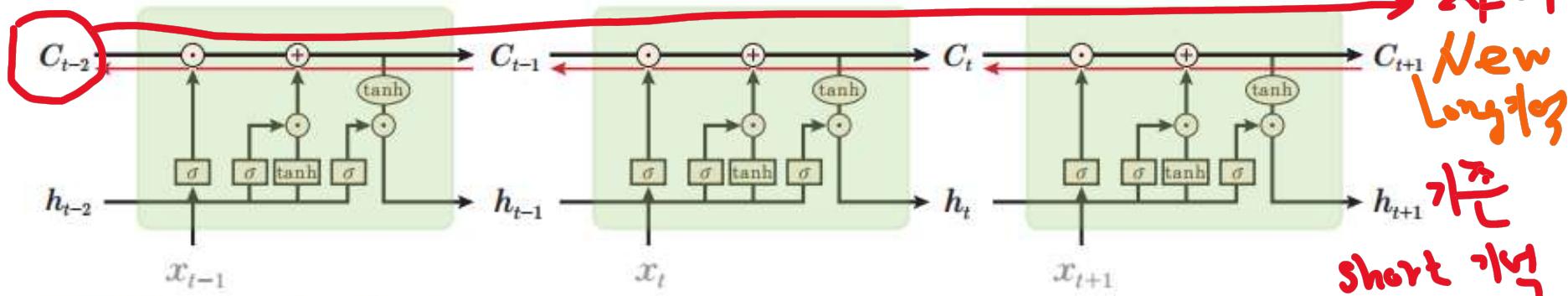


그림 8-27 LSTM의 그레이디언트 흐름

LSTM

Long Term Memory & Short Term Memory

- LSTM은 셀 상태와 은닉 상태로 뇌의 장기 기억(long term memory) 와 단기 기억(short term memory)을 모델링을 함!!
- 그럼 기존 RNN은? → 은닉 상태를 전달하는 방식으로 기억을 모델링을 하였으면,(그냥 흘러가는 식으로만)
- LSTM은 장기 / 단기 구별해서 각기 진행시킴!!!

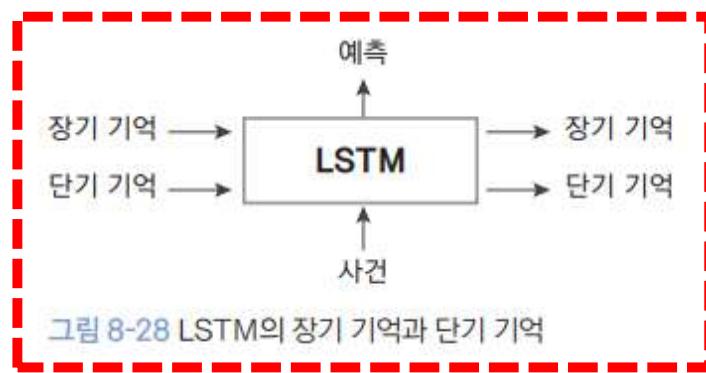


그림 8-28 LSTM의 장기 기억과 단기 기억

LSTM

Long Term Memory & Short Term Memory - 모델링!

- Long Term Memory : 오래 지속이 되지만, 새로운 사건이 발생하는 것에 큰 영향을 받는 것이 아니라 조금씩 강화하거나 약화하면서, 자주 사용하지 않으면 잊게 되는 것!! → 그래서 셀 상태를 W와 연관이 없이 그대로 전달되게 함!!!!
- Short Term Memory : 새로운 사건이 발생하면, 장기 기억 또는 단기 기억화 연합해서 인식하는 과정에서 단기 기억이 형성이 됨!! → 단기 기억은 최근의 것은 빠르게 잘 알지만, 조금만 지나가면 잘 잊게 된다. → LSTM은 이러한 단기 기억의 특징을 모델링을 하기 위해서 “은닉 상태에 최근 사건에 대한 부분이 형성되도록 W 가 곱해지는 구조로 설계”
- 결론 : 기억을 형성하고 지속하고, 망각하는 과정에서 단기 기억과 장기 기억은 상호작용을 하도록 함!

LSTM

Long Term Memory & Short Term Memory - 상호 작용!

- 셀

- 입력1 : 이전 셀의 단기 기억, 이전 셀의 장기 기억,
- 입력2 : 사건
- 출력1 : 이번 셀의 단기 기억, 이번 셀의 장기 기억 (다음 셀이 있다면 넘겨 주는 용도)
- 출력2 : 예측

- 기능의 특징

- 이전 셀에서 전달된 장기 기억 중에 필요 없는 것은 지워 버림
- 단기 기억에는 최근 사건에 대한 정보를 주어서 새로운 기억을 형성 시킴
- 새롭게 형성된 기억 → 일부는 장기로 보내고, 이를 바탕으로 갱신된 정보를 가지는 단기 기억을 만들어 현재 단계 예측에 활용!

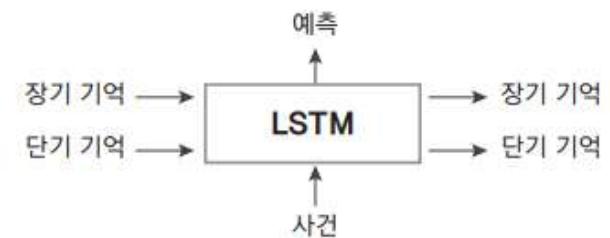


그림 8-28 LSTM의 장기 기억과 단기 기억

LSTM

Long Term Memory & Short Term Memory – 기억 선택 게이트

- LSTM에는 4가지의 기억 형성과 관련된 게이트가 존재!!!

- 역할 : 기억을 지속할지, 지워버릴지 선택
- 종류
 - 망각 게이트 Forget Gate : 장기 기억을 지속할지 잊을지 판단! → 잊어도 되는 장기 기억은 망각 게이트를 통해 시키지 않음
 - 입력 게이트 Input Gate : 새로운 사건으로 형성된 기억 중 장기 기억으로 전환할 기억 선택
 - 기억 게이트 Remember Gate : 장기 기억을 새롭게 갱신하기 위해 망각 게이터를 통과한 장기 기억에 입력 게이트를 통과한 새로운 기억을 더함.
 - 출력 게이트 Output Gate : 갱신된 장기 기억에서 갱신된 단기 기억에 필요한 기억을 선택

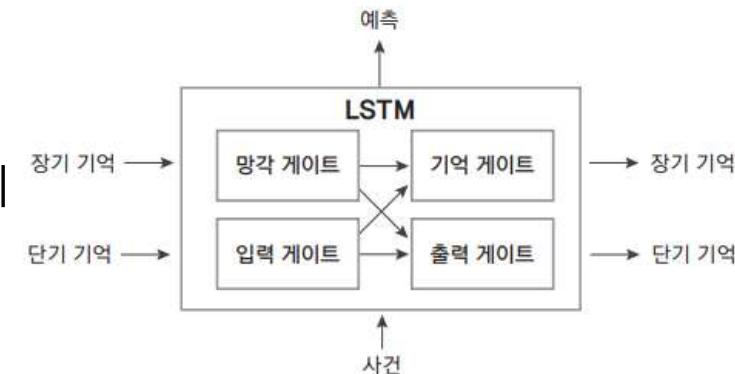


그림 8-29 LSTM 셀의 게이트 구조

LSTM

Long Term Memory & Short Term Memory – 기억 선택 게이트에 따른 전반적인 흐름

- LSTM은 게이트 구조를 통해 새롭게 형성된 기억을 지속하거나 지워버린다.
- 시간의 흐름에 따라 이 과정이 어떻게 진행되는지 살펴보자.
- 이 그림은 첫 번째 사건이 단계 6까지 전달되면서 단계 4와 6의 예측에 활용되는 모습을 보여준다.

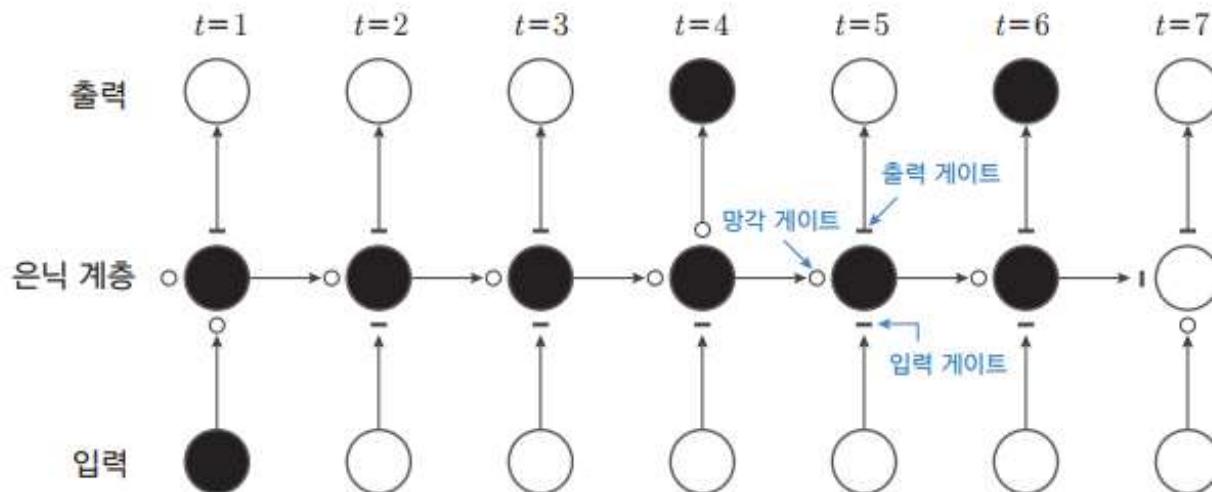


그림 8-30 장기 기억의 전달

이제는 뒤에 자세히
구조별로 보자

Into To the LSTM

- Cell Structure

七八점

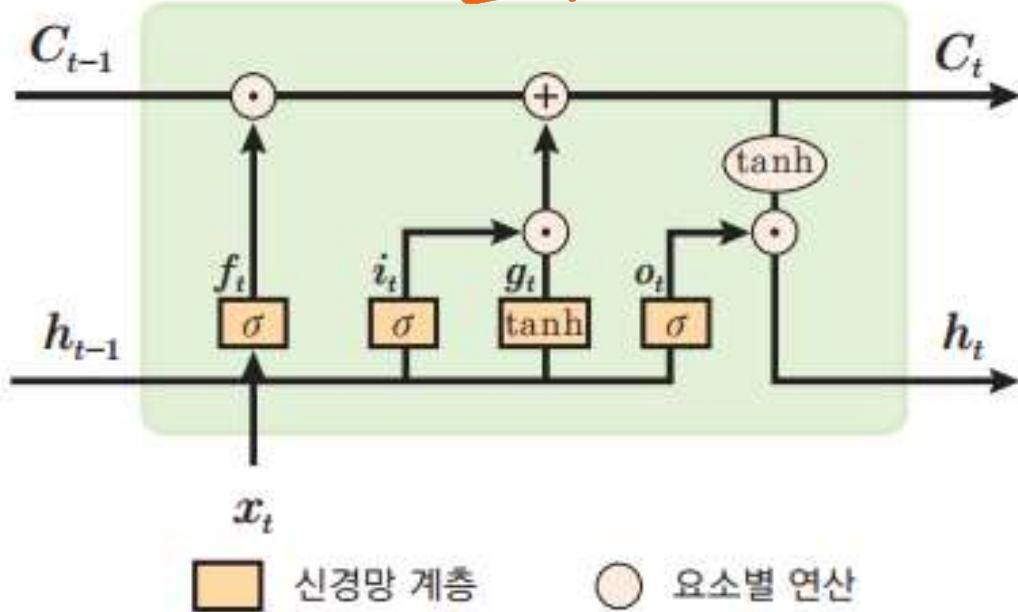


그림 8-31 LSTM 셀 구조

장기 기억 : 셀 상태 C_t
단기 기억 : 은닉 상태 h_t

사건 : 입력 x_t

망각 게이트 : f_t

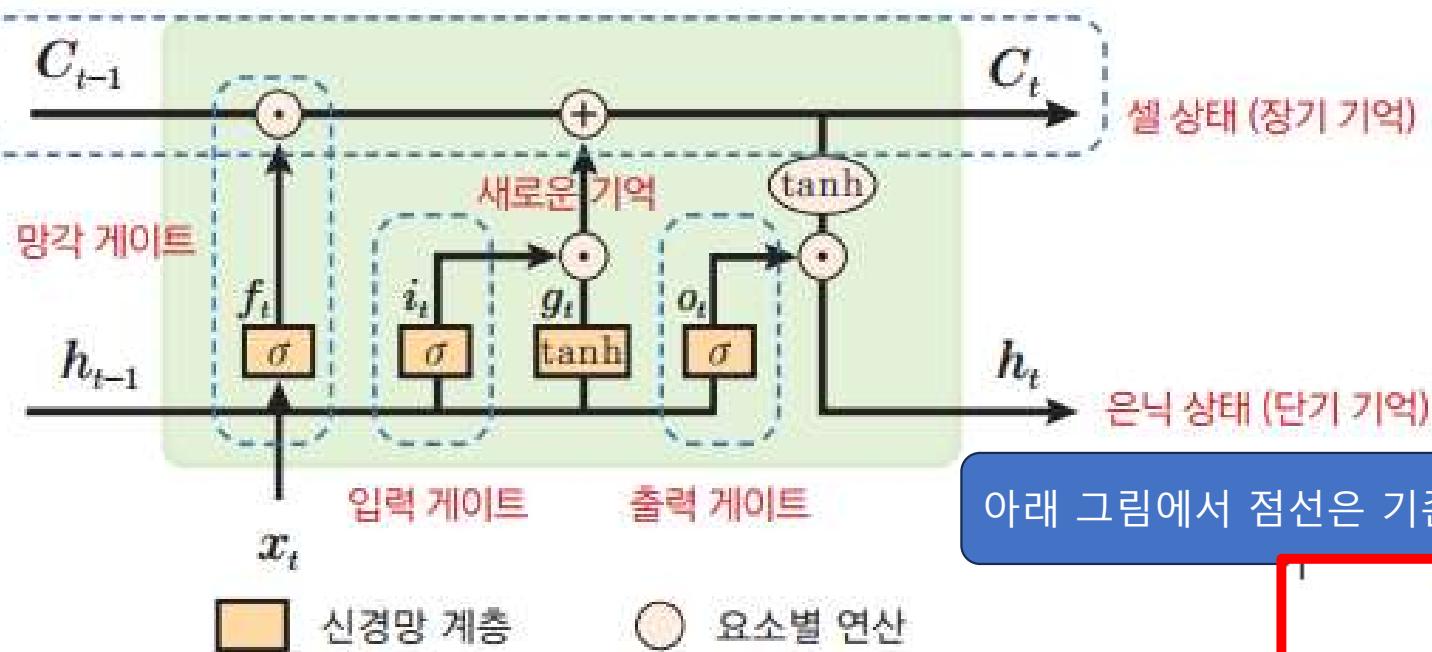
입력 게이트 : i_t

출력 게이트 : o_t

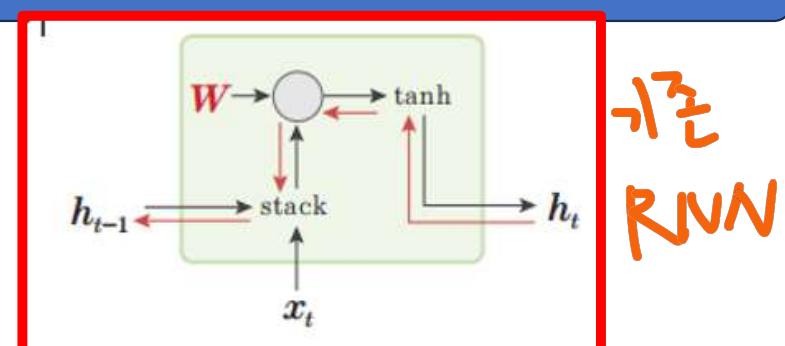
새롭게 형성된 기억 : g_t

Into To the LSTM

- Cell Structure



아래 그림에서 점선은 기존 RNN에 없는 새로운 구조들임!!!



Into To the LSTM

- Cell Structure

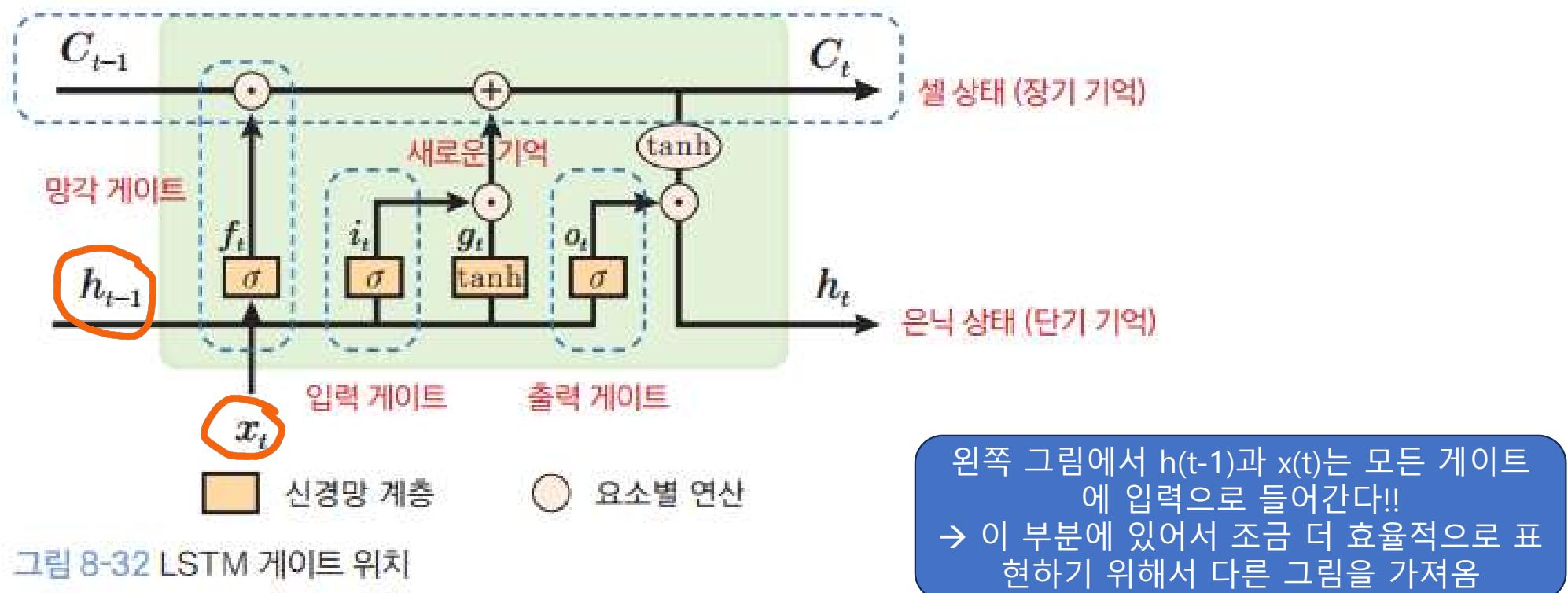
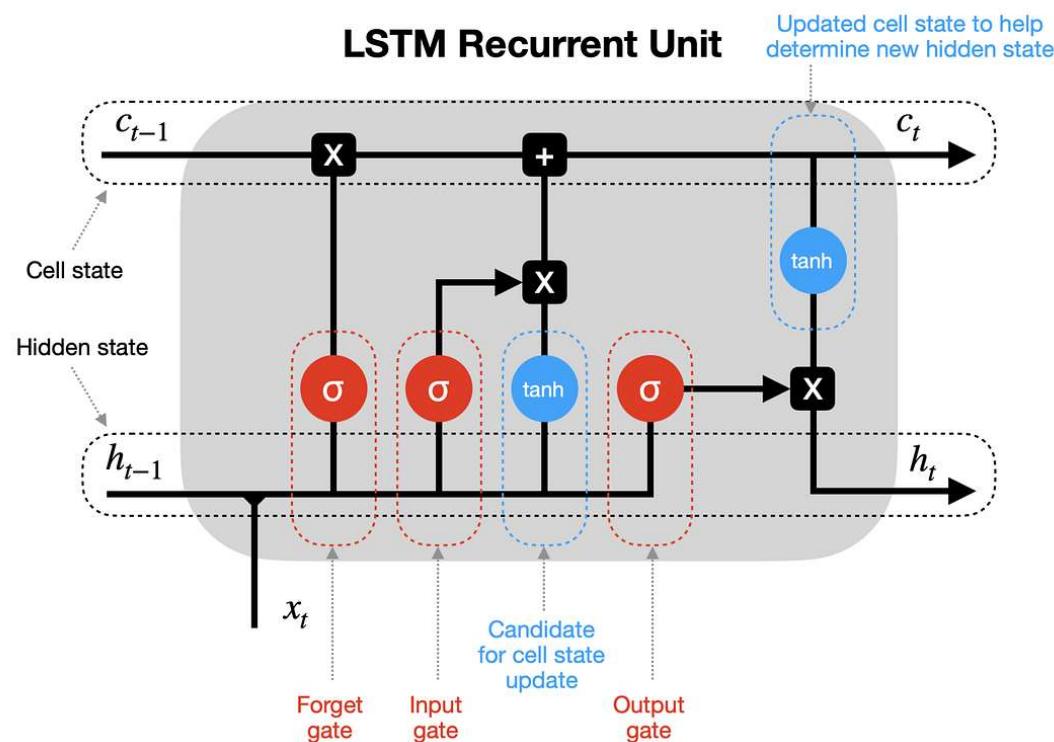
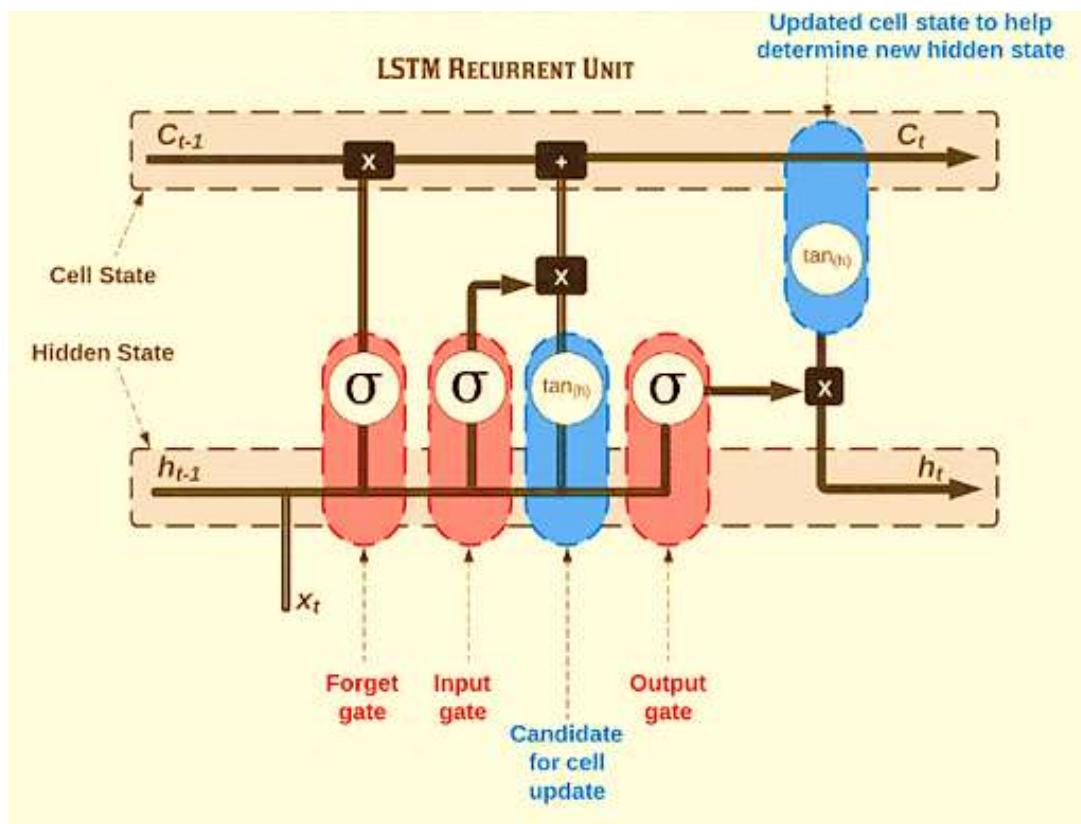


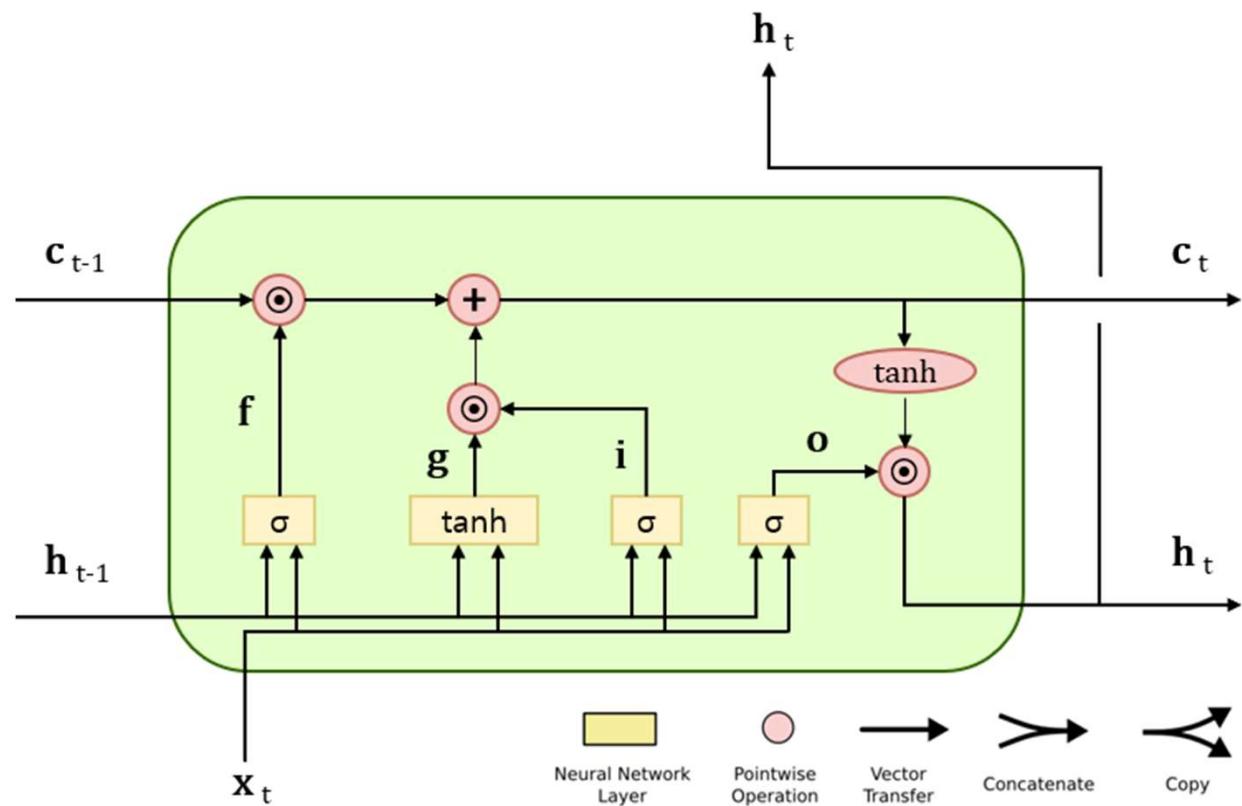
그림 8-32 LSTM 게이트 위치

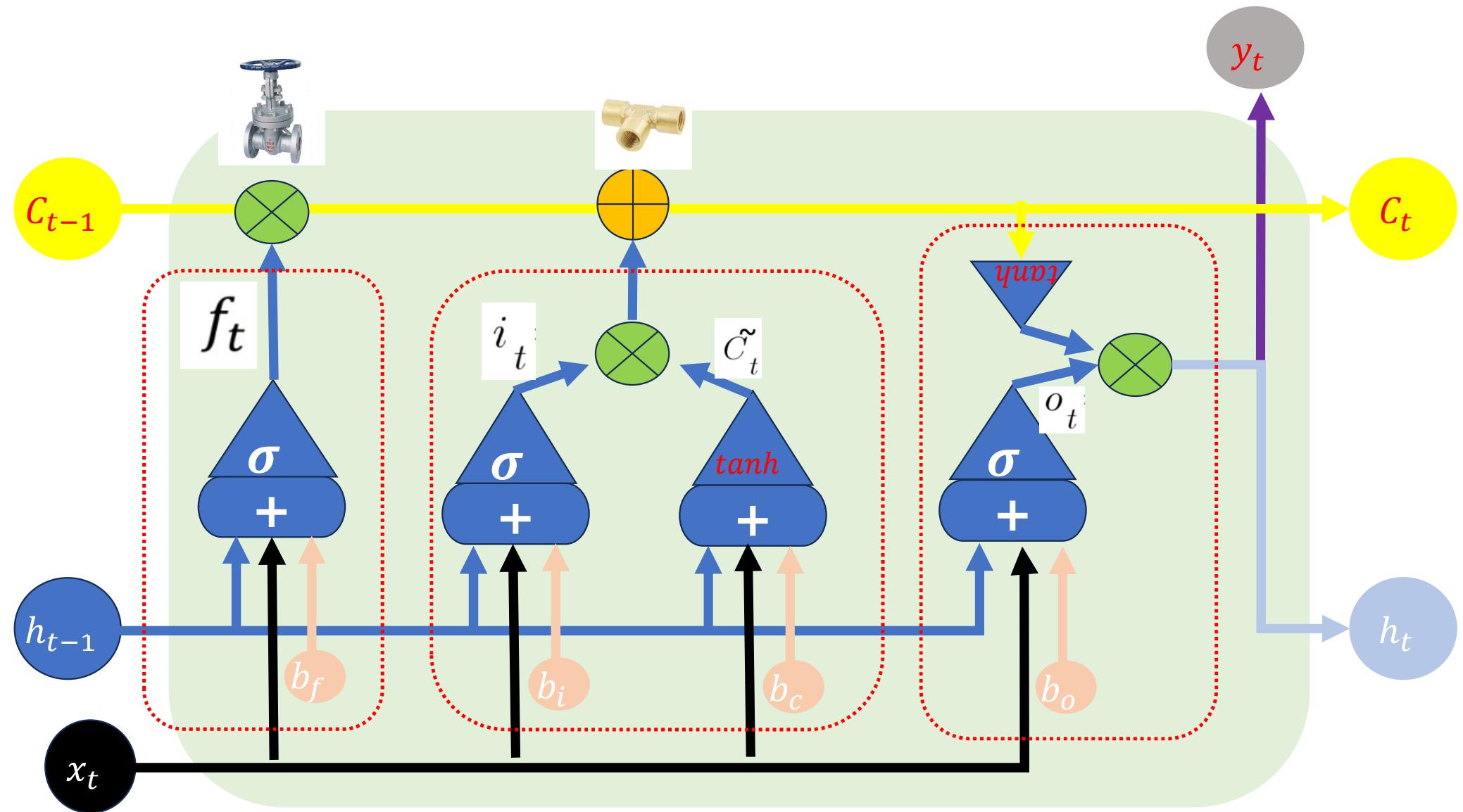
<https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e>

LONG SHORT-TERM MEMORY NEURAL NETWORKS









LSTM

입력 게이트 $\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix}$ = $\begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix}$

망각 게이트 이전 단기 기억 $W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$, $W = \begin{bmatrix} W_{hi} & W_{xi} \\ W_{hf} & W_{xf} \\ W_{ho} & W_{xo} \\ W_{hg} & W_{xg} \end{bmatrix}$

출력 게이트

새로운 기억

장기 기억 이전 장기 기억 새로운 기억

망각 게이트 입력 게이트

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

LSTM

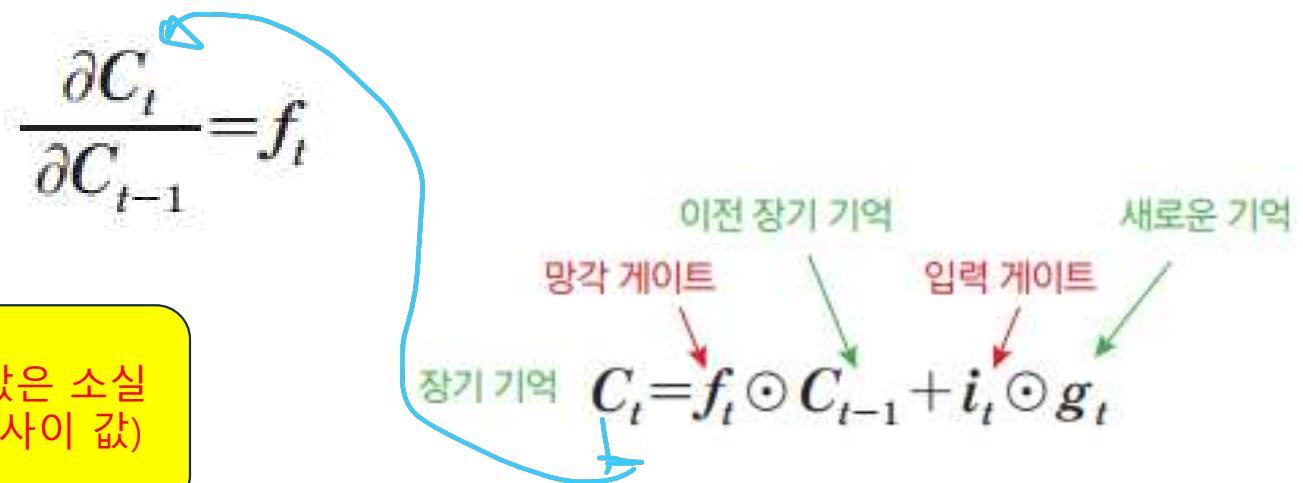
$$h_t = o_t \odot \tanh(C_t)$$

LSTM

- Gradient Vanishing이 안 생기는 이유

- 그레이디언트 소실을 유발하는 요인이었던 W 의 반복적인 곱 연산이 사라졌음!!!

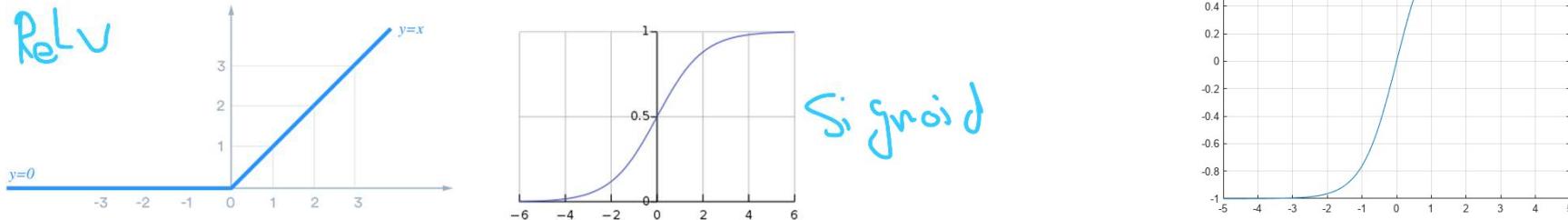
F는 셀마다 값이 다르고, .f의 값은 소실될 가능성이 적어져서임!(0~1사이 값)



$$\frac{\partial C_t}{\partial C_0} = \frac{\partial C_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial C_{t-2}} \cdot \dots \cdot \frac{\partial C_1}{\partial C_0} = f_t \cdot f_{t-1} \cdot \dots \cdot f_1 = \prod_{i=1}^t f_i$$

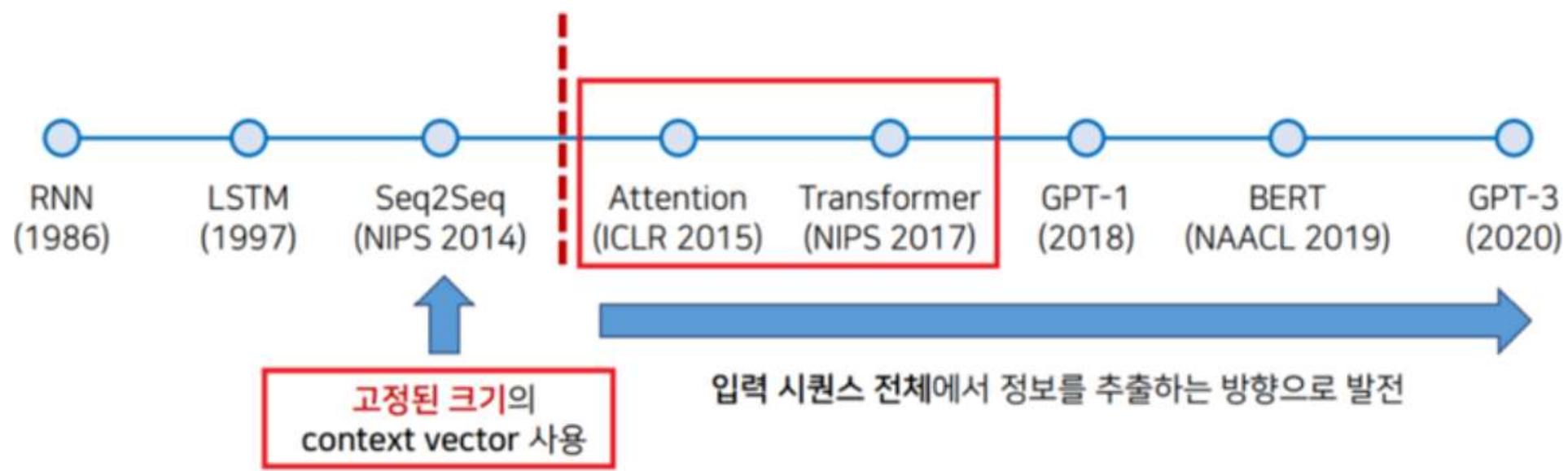
Why tanh/sigmoid가 여기서 사용되나? ReLU보다도.....Why?

- 기존에는 영상쪽이나 일반적인 부분에서 tanh/ sigmoid는 안 좋다고 이야기를 하는데, 왜 여기서는 과거의 gismoid, tanh를 사용하는가?
- 아래 그림처럼 ReLU는 0보다 작으면0, 크면 자기 값 그대로 출력을 내보낸다. RNN 계열에서는 연속적으로 입력이 들어가게 되는데, 0으로 생략되어서는 안 되어서,,relu는 잘 사용 안 함!!
- 활성화로 sigmoid는 안 쓰고 tanh를 쓰는 이유? (sigmoid로는 어느 정도 사용할지만 하지, AF으로는 안 씀) ➔ 계속 재귀하게 되면 평균인 0.5정도로 편향이 발생을 하고, bias가 있어도 계속 보상하기가 쉽지가 않음, 그래서 tanh는 0으로 되어서,,,



Attention please





출처 : 유튜브 나동빈 채널

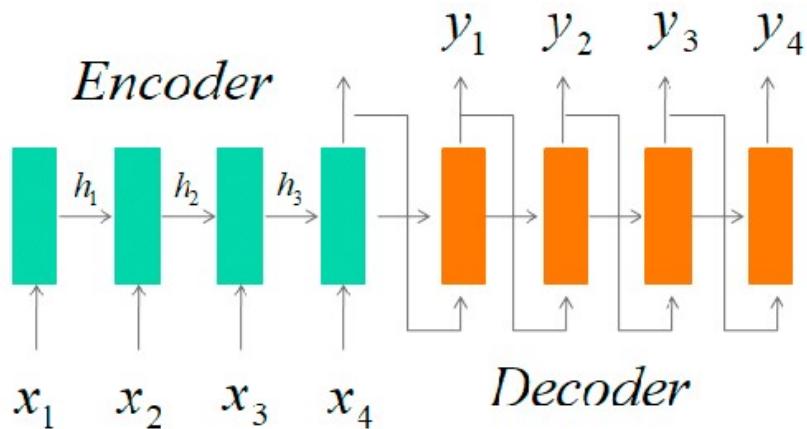
Transformer : Attention is all you need

- 소개

- NLP 분야에서 확 전환을 이끈 모델임
- 번역의 경우에 있어서 단어들에 대한 숫자를 잘 처리를 해야하며
- Attention
 - 일반 attention : 번역 시 어떤 단어를 집중해야할지 선택
 - Self – Attention : 각 단어에 대한 숫자화를 잘 하고, 이는 내적을 사용을 하여, 단어간의 관계를 잘 파악하게 하는 특징이 있음.
- 핵심 : Inner Product!!!! (이미지 데이터의 Convolution이라는 연산이 큰 역할을 했다고 하면, NLP에서는 그냥 그 쉬운 Inner Product가 핵심적인 역할을 하게 됨!!!)

Your cat is a lovely cat

너의 고양이 는 사랑스러운 고양이 입니다



$$x_n = \{Feature\ 1, Feature\ 2, \dots\} \quad y_n = \{Feature\ 1, Feature\ 2, \dots\}$$

[기존의 Seq2Seq 의 모델을 수정을 함]

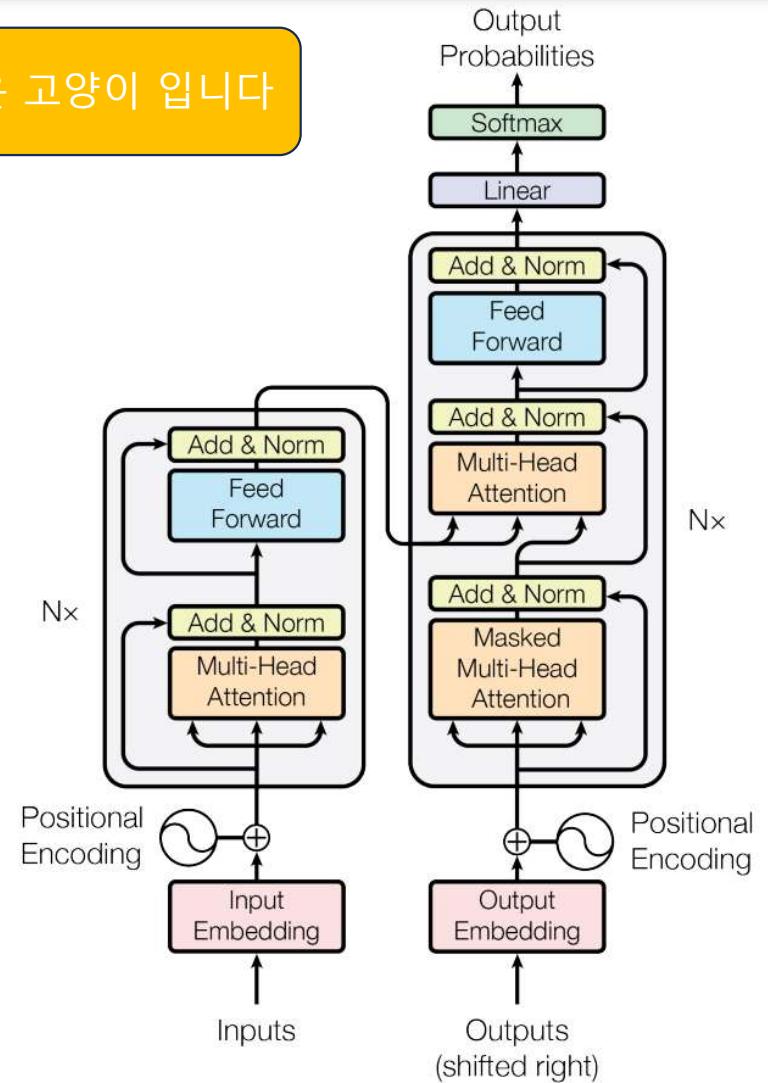


Figure 1: The Transformer - model architecture.

Transformer : Attention is all you need

- **Attention : 집중!!** → 기존의 RNN의 Sequence의 구조에다가 문제점으로 거론이 되는 주의 결핍에 대한 것을 Attention 을 접목하여서 어디에 집중하고 강조할지를 구조화 한 모델!!!!!!
- RNN(Sequence) + Attention (Where!!) → Transformer!!!

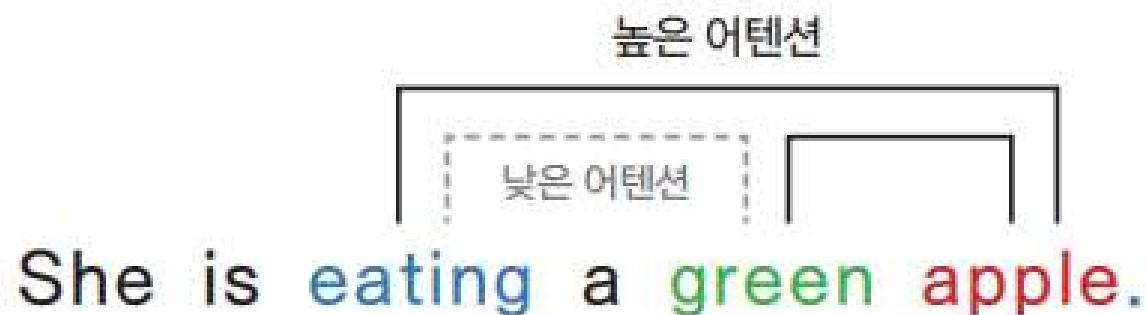


그림 8-34 단어 간 어텐션

Transformer의 핵심

- 1. 한 덩어리 입력 데이터 받기 Input Embedding

참고 : 단어 수는 긴 문장 기준

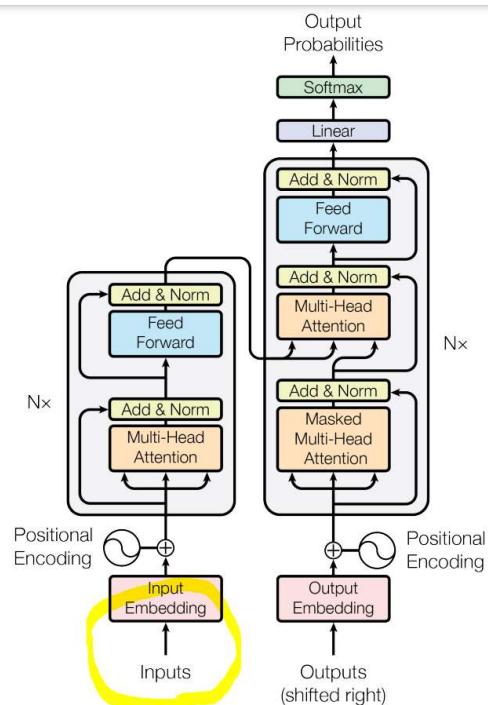


Figure 1: The Transformer - model architecture.

핵심 중 하나는 일반적으로 One Hot Encoding된 단어에 대해서 512 등으로 뭔가 제한적인 벡터 값으로 변환!!!!

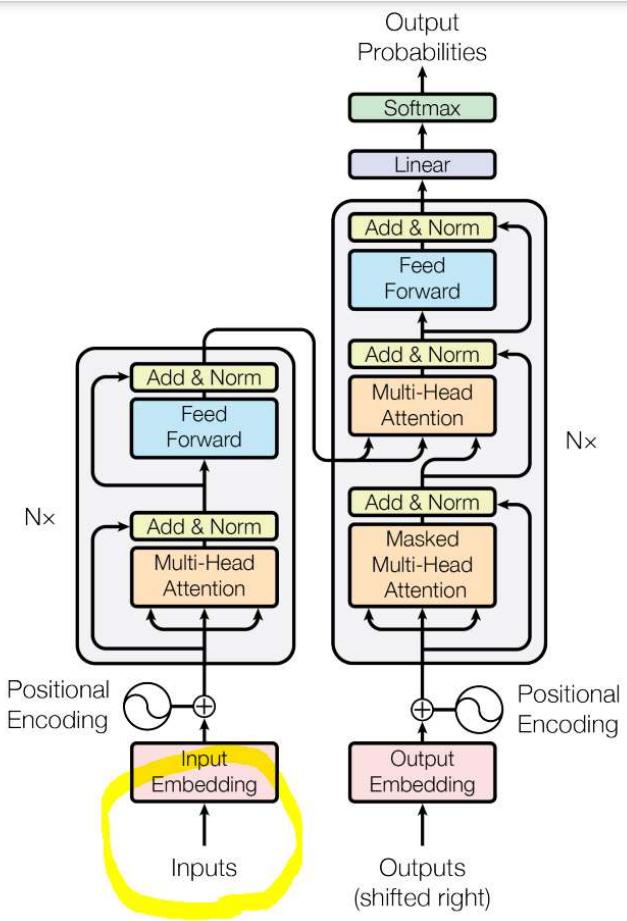


Figure 1: The Transformer - model architecture.

Transformer의 핵심

- 2. 위치 잡아주자 Positional Encoding

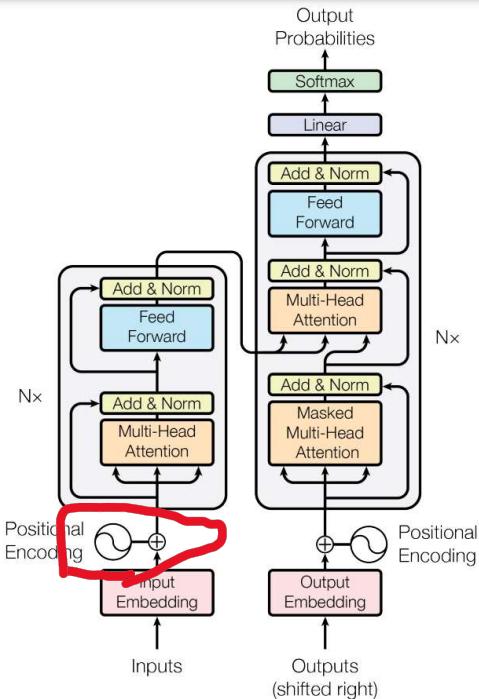


Figure 1: The Transformer - model architecture.

- 앞에서와 같이 단순히 단어에 대한 인코딩만으로는 RNN에서 하는 것처럼 단어에 대한 순서에 대한 정보를 표현할 수 없음!!!
- 제안 사항 : 그럼 단어의 위치에 대한 것도 One – Hot 인코딩으로 표현하면 되지 뭐!!!!
 - 실제로 그러면 내가 가진/ 학습하려는 데이터 중에서 제일 긴 문장의 길이를 체크를 한다 max_len
 - 아니면, 특정 최대 길이로 제한을 하거나 max_len (현실은 적당하게 제한을 함. 이유는 연산과 너무 불필요한 부분 때문에 gpt등으로 제한이 있는 이유임)
 - 모양을 앞의 덩어리 기준으로 하면 32개 문장단위로 처리하고, 50개의 단어에 대해서, 각 단어의 수치정보를 이번에는 position의 max len의 길이로 주자는 것임!!!!
 - 앞의 예 같은 경우에는 (32, 50, max_len)
- 1번 위치 [1, 0, 0, 0, ..., 0]
 - 2번 위치 [0, 1, 0, 0, ..., 0]
 - 이런 것들이 max_len 길이가 되는데, 여기서는 예로 200이라면 앞에서 넘어온 것과 모양이 안 맞음(32, 50 , 512)인데, 여기서는 (32, 50, 200)

왜 Position에 대한 정보를 주는 것일까?

[내일 엔비디아의 주식의 가격이 이천비디아를 향해서 갈까?]

[코로나 초기 때 엔비디아 역시 다른 종목들과 마찬가지로 주식이 엄청난 하락을 하였다]

- ➔ 위의 문장에서 보면, 엔비디아, 주식이라는 단어가 있는데, 의미상 주식이 하나는 내일 관련 주식을 의미하고, 하나는 전의 코로나때를 의미하게 된다!!!!
- ➔ 단순 word embedding만 하면 이러한 부분이 반영이 안 되고, 엔비디아/ 주식이라는 것들의 word embedding의 값이 동일하기에, 두 문장에서 관련 미묘한 위치에 대한 부분을 반영할 수 없게 되어서 이를 개선하고자 함!!!!

- 이러한 부분에서 학습을 통해서, 데이터가 알아서 위치에 따라서 어떻게 값을 주는게 좋을지 학습을 하도록 하게 함!! → 주어진 데이터에 대해서 위치에 따라서 어떠한 벡터를 더해줄지에 대한 것이 결정이 됨!!!!
- 그리고 이러한 것들은 규칙이 아니라 네트워크 님이 학습해서 해주라!!!!!
- 참고1) 논문에서 보면 위치 정보들에 대한 것들이 너무 크게 지표가 되는 것을 방지하고자 d_{model} (논문에서 512)의 root를 처리를 함!!

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model} . We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. **In the embedding layers, we multiply those weights by $\sqrt{d_{\text{model}}}$.**

- 참고2) 위치에 대한 인코딩 관련 부분

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned}$$

일단 여기서 대상은 Position Encoding임

- 문장에서 단어의 위치가 POS
- 각 단어를 우리가 512 차원의 벡터로 하였으므로 그 단어의 벡터값을 i로 의미
- d_{model} 은 논문상 512로 하였고,,
- 예 : 나는 학교에 갔습니다. → 이 문장의 경우에 대해서 단어 위치임.
- 나(pos =0,), 학교(pos=1), 갔습니다(pos=2)

나(pos=0)를 나타내는 값들이
[0.001, 0.002, 0.014,.....] 이렇게
512차원의 벡터임.

0.001은
이 문장에서 0번 단어의 0번 값
 $PE(0, 0)$

0.002은
이 문장에서 0번 단어의 1번 값
 $PE(0, 1)$

각기 단어의 벡터값의 훌쩍에서
sin/cos으로 값을 변환해서 처리를
함...

- 참고2) 위치에 대한 인코딩 관련 부분

3.5 Positional Encoding

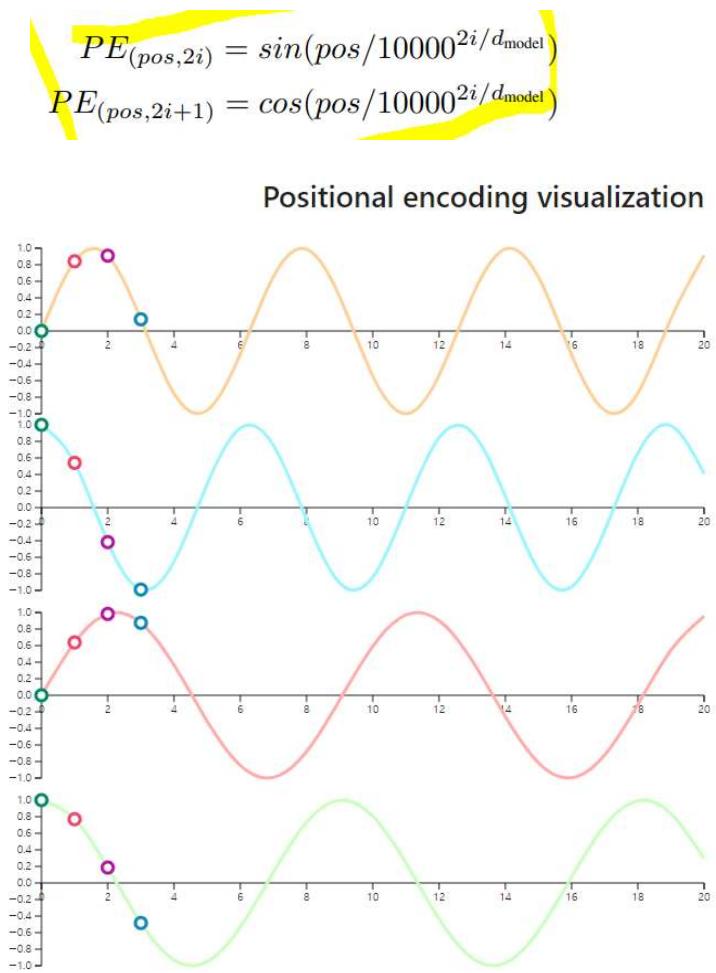
Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned}$$

일단 이렇게 하면, 학습의 과정이나 추론의 과정에서 여러 번 재사용
이 될 것으로, 처음에 인코딩 과정에서 한 번만 계산을 해두면 여
러 번 사용이 가능함!! 정확하게 단어에 대해서 문장상의 위치정보
 pos 와 그 단어의 표현값에 대한 i 를 같이 해서, 이 부분이 동일하면
다른 문장에서도 동일한 값을 가지게 됨.

- 참고2) 위치에 대한 인코딩 관련 부분



Why Sin /Cosine?

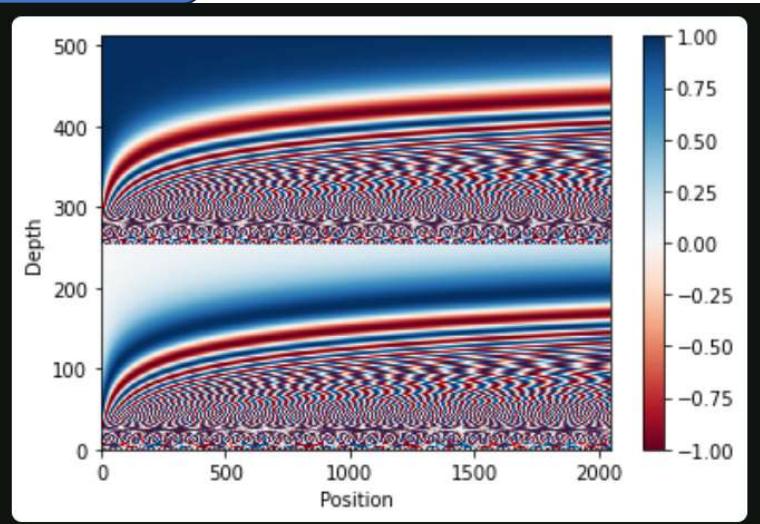
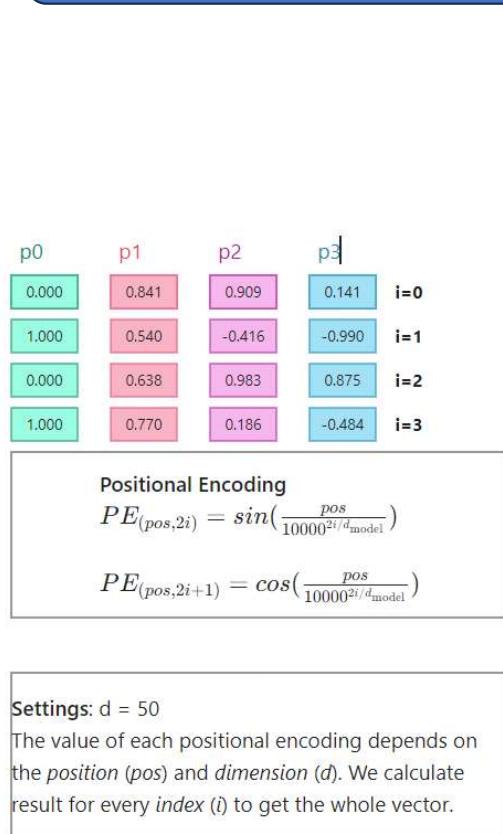


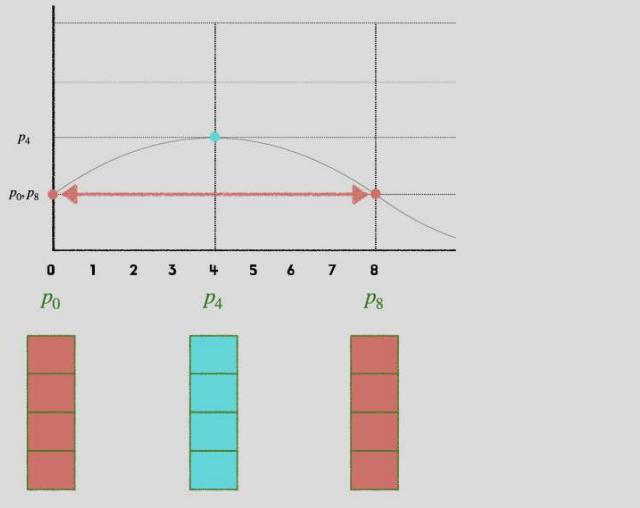
Figure 1. Visualization of sinusoidal positional encoding.

Source: TensorFlow tutorial: Transformer model for language understanding

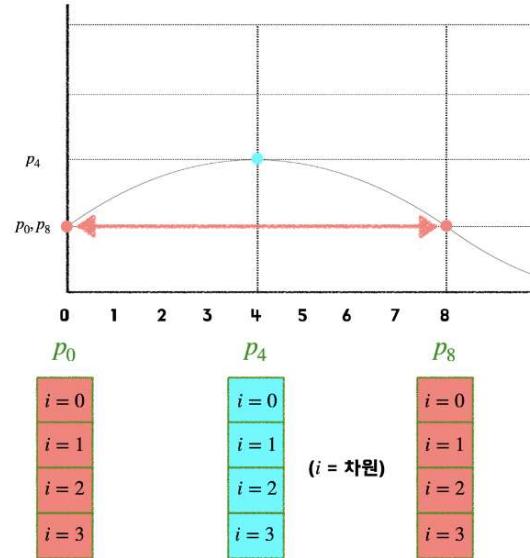
* Sin/cos은 $-1 \sim +1$ 사이를 반복하는 주기함수 → 값이 바운드가 되면서 & 주기성(sigmoid는 주기성이 없음)

*

예를 들어 아래 그림과 같이 Sine 함수가 주어진다면 1 번째 토큰(position 0)과 9 번째 토큰(position 9)의 경우, 위치 벡터값이 같아지는 문제가 발생한다.



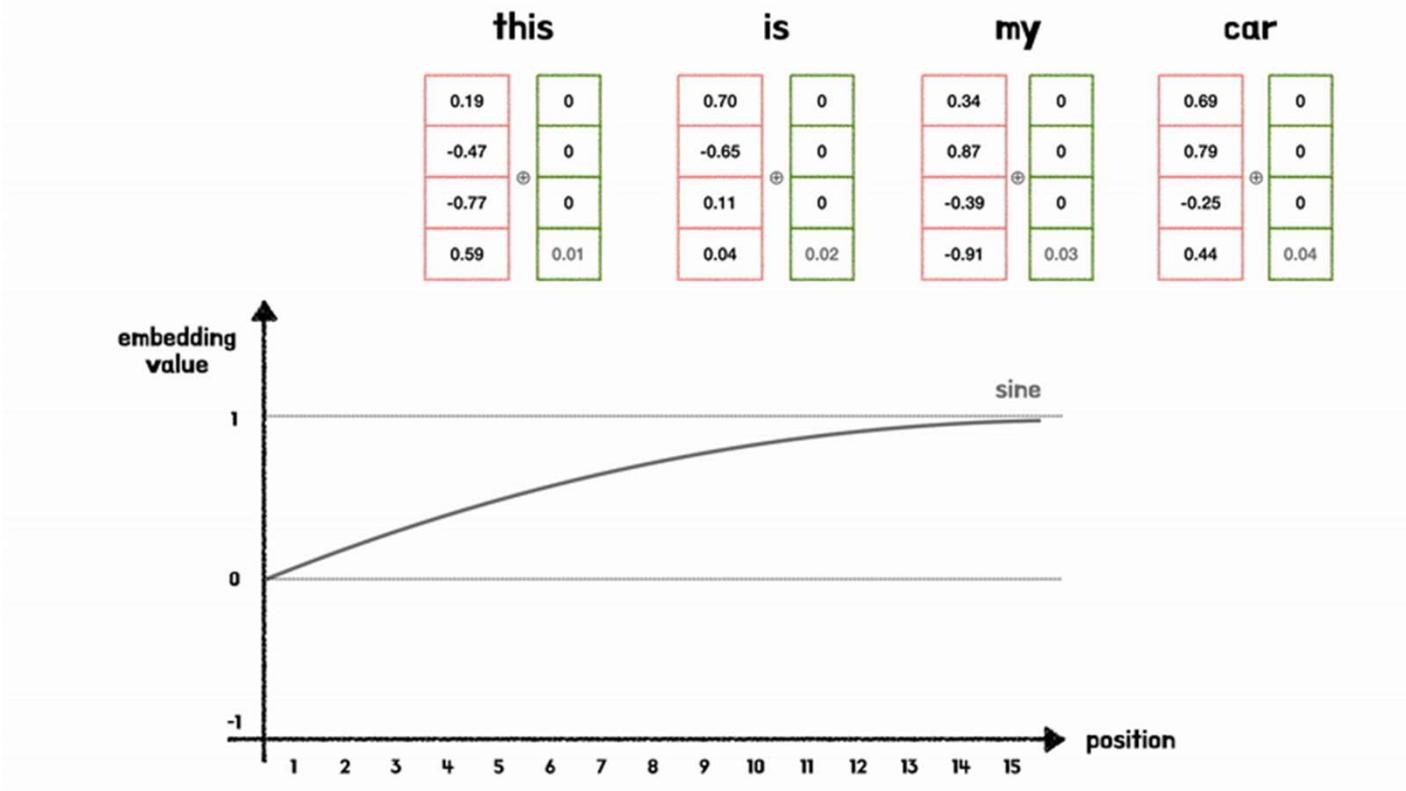
- positional encoding은 스칼라값이 아닌 벡터값으로 단어 벡터와 같은 차원을 지닌 벡터값이다.



이렇게 동일하게 되는 것들에 대해서
주기함수를 이용해서 위치에 대한 것을 바탕으로
주기를 위치별로 다르게 설정할 수 있도록 함!!

특별한 이유는 없음... 그냥 다른 값 부여해보자임!!!

<https://www.blossominkyung.com/deeplearning/transfomer-positional-encoding#37c2b888-b613-477e-9053-31a4cf2d3e67>



Transformer의 핵심

- 3. Multi-Head Attention

역할 : 입력 문장 내에서 단어간의 관계를 학습시키기 위해서 내적을 사용을 함!!!!

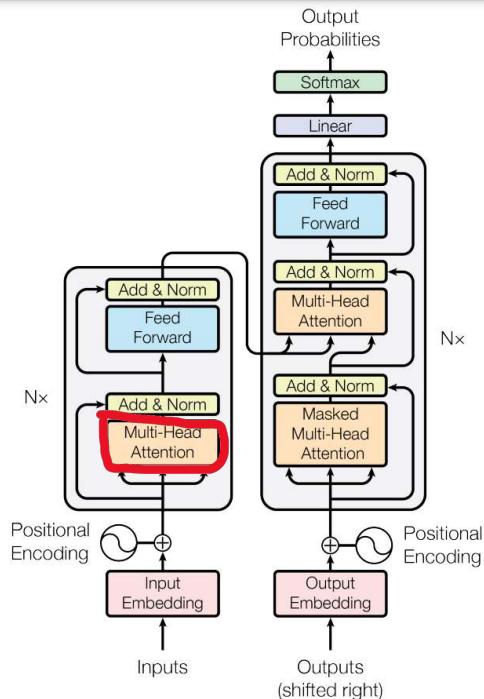
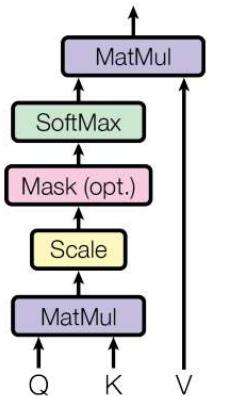


Figure 1: The Transformer - model architecture.

Scaled Dot-Product Attention



Multi-Head Attention

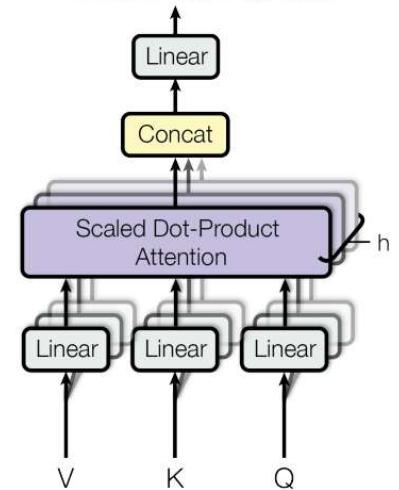
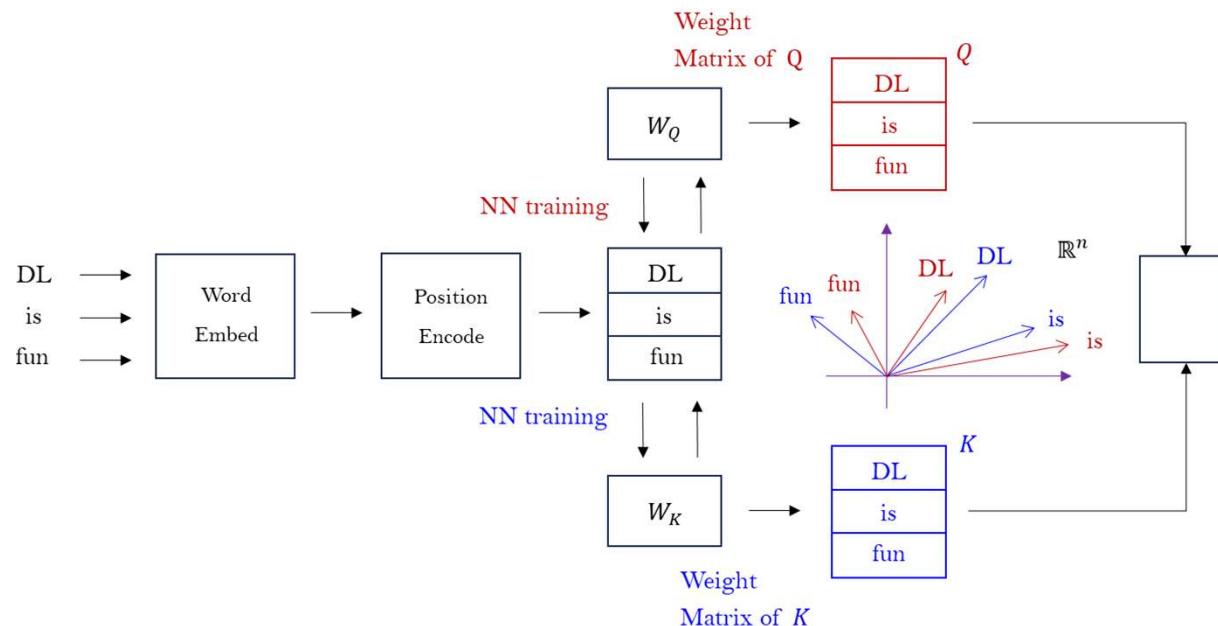


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

어떤 단어들이 서로 연관성이 깊고, 어떤 단어에 Attention 해야할지 학습!!!!

→ loss를 줄이는데 기여하는 단어들은 가깝게하고, loss를 크게 하는 단어들은 멀리하자!!!

→ 그럼 이를 어떻게 표현하나????

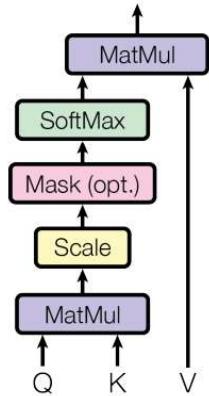


인공 신경망의 레이어는 Weight 를 계산을 하는 것!!

→ 이것은 입력에 대한 WeightMatrix를 곱한 결과

→ 선형 메트릭스의 경우에는 선형 변환을 의미를 하여, 공간을 재조정을 할 수 있다는 것!!! 이것을 통해서 잘 변형을 하자는 것이고, 학습의 대상임!!!!

Scaled Dot-Product Attention



Multi-Head Attention

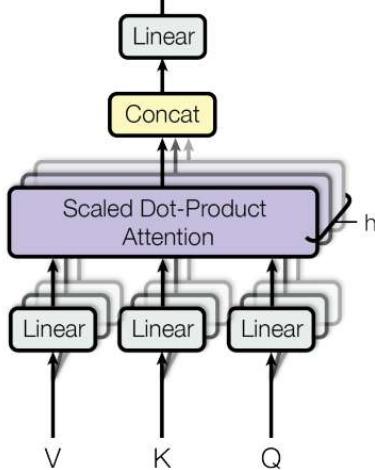


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

영어: I love her.

한국어: 나는 그녀를 사랑한다.

영어를 한국어로 번역하는 것으로 예시를 들면, **나는이라는** 주체가 **Query**, 연관성을 찾는 대상인 (**I, love, her**)이 **Key**, Q와 K의 유사성을 계산하여 유사한 만큼의 **Value**값을 가져올 수 있습니다. 일반 딕셔너리는 일치하는 K값에 V값을 가져오는 것인데, Attention Mechanism은 Q, K의 유사한 만큼의 V를 가져온다고 보시면 될 것 같습니다. Q, K, V는 단순한 스칼라가 아니라 Vector이고, 출력도 마찬가지로 Vector입니다.

What is Q / K / V ?

- Q : Query
- K : Key
- V : Value

→ 논문 상에서는 특별하게 의미를 부여를 하지를 않고, 그냥 Query / Key / Value라고 하고 있음.

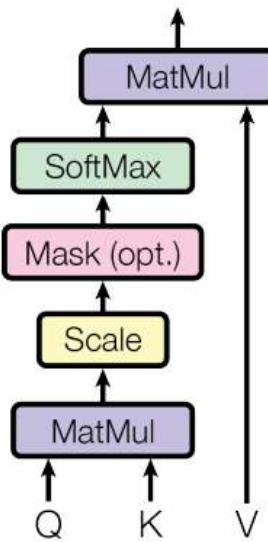
→ 그냥 단순히 봐서 Q / K의 행렬이여서 행렬의 곱 dot-Product를 가지고 V의 Weight를 한다는 의미!!!

→ [개인 의견] 왼쪽과 같은 예를 들지만, 개인적인 의견으로는 seq 2 seq 이므로 이 seq과 seq을 연결하는 뭔가의 정보들을 담당하는 것으로 파악을 함. 각기 정보들을 담당하게 하여, 이에 대한 곱을 통한 종합적인 사항을 반영하게 하는 듯

→ 주의!! Q/K/V 모든 동일한 대상을 다른 각도로 보는 것임에 유의!!!(내부 숫자 같마 다른!!)

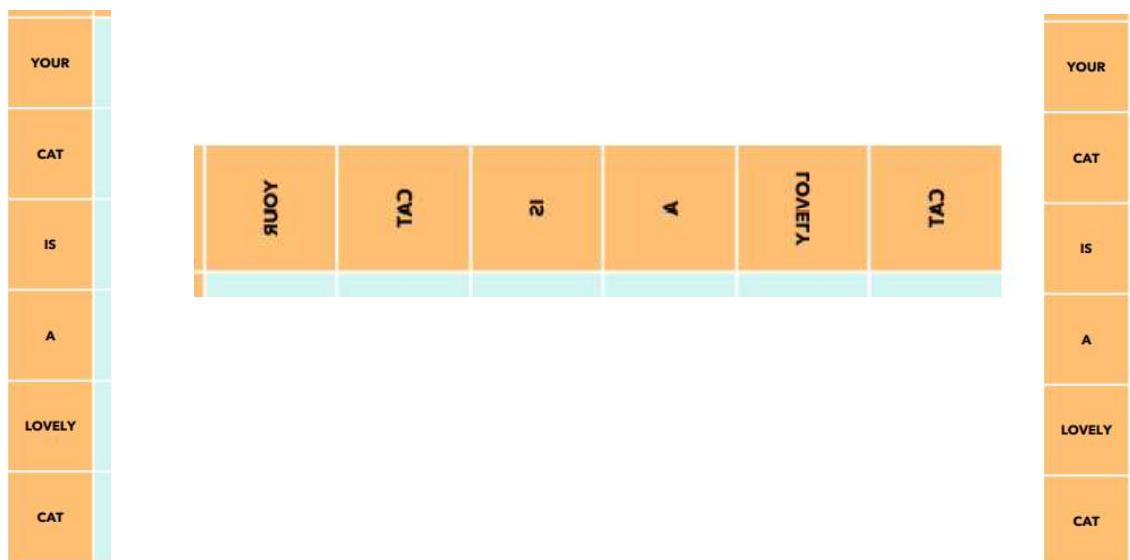
<https://lcyking.tistory.com/entry/%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%87%BC-Attention-is-All-you-need%EC%9D%98-%EC%9D%B4%ED%95%BA>

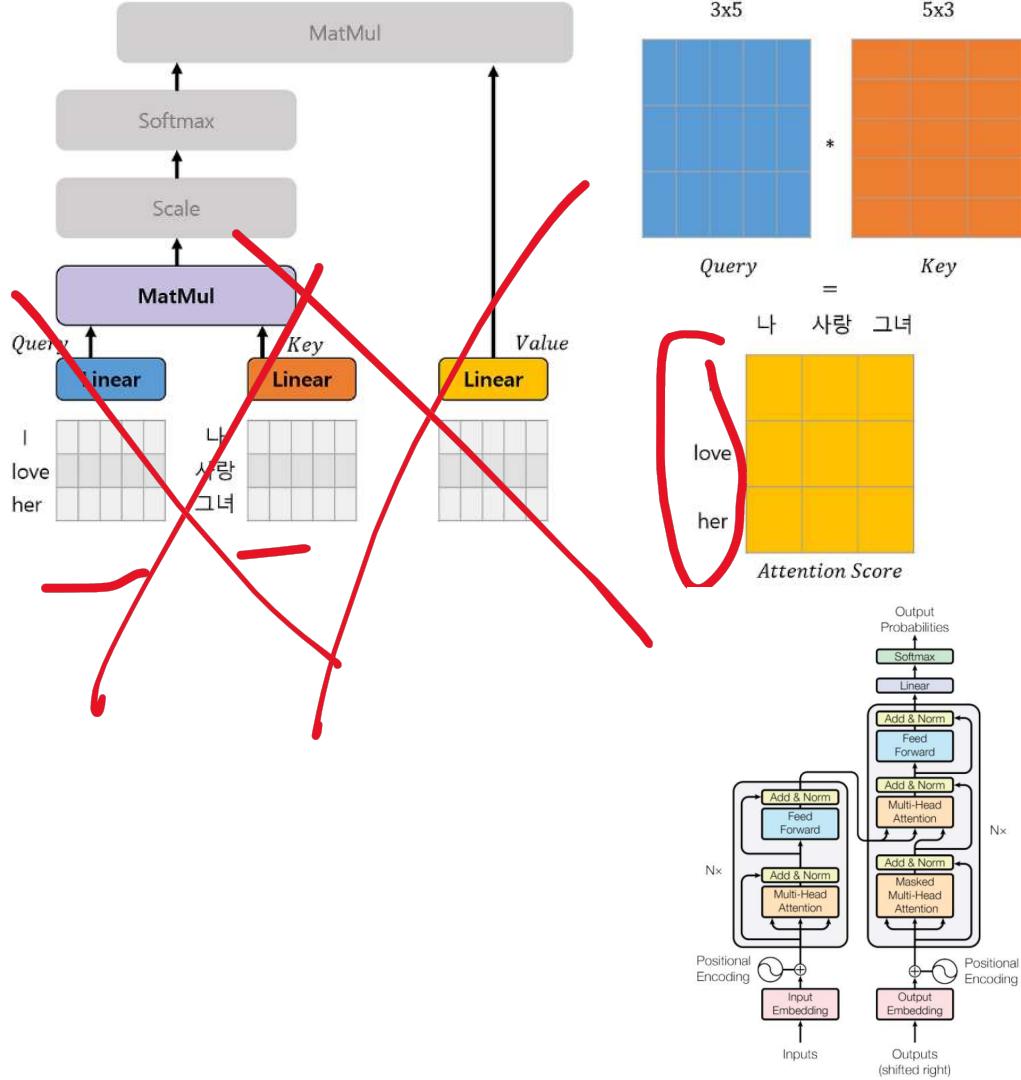
Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q / K / V 모두 같은 대
상이고, 다만 안의 값만
다를 뿐이다!!!





주의!!!

인터넷들을 찾아보고 하면
대략적인 그림들을 이쁘게 그리는 것들이 있는
데.....

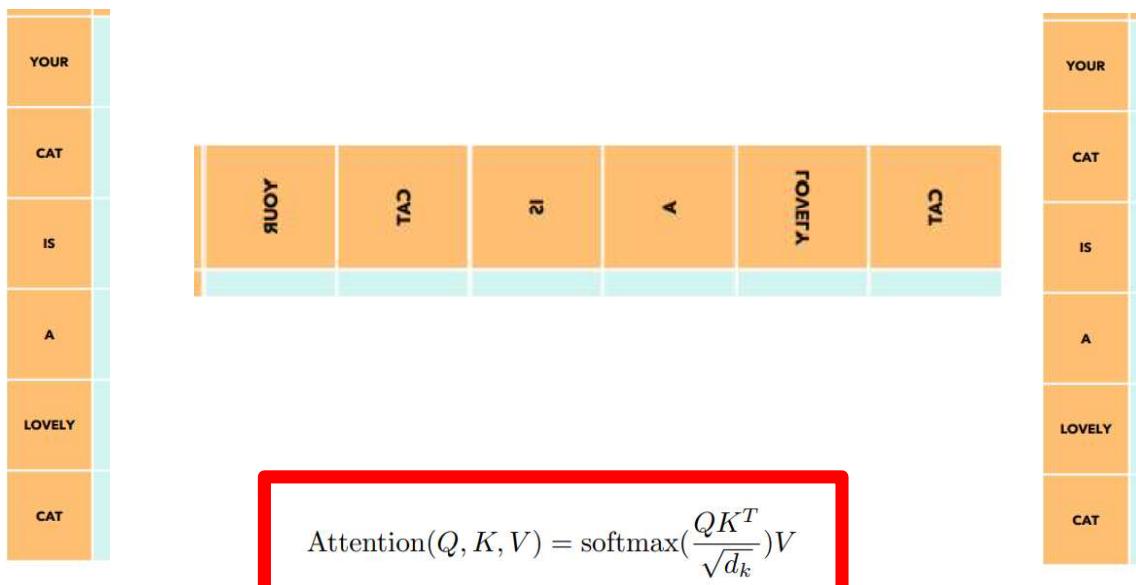
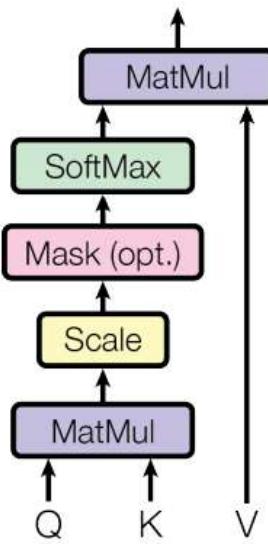
솔직히 다들 제대로 알고 그린 그림을 잘 찾아
서 이해를 해야한다. 다음과 같은 것을 보면, 오
히려 오해와 혼동을 불러일으킨다.

Q / K 가 번역을 위해서 서로 다른 것을 나타내
는 것은 이 부분에서 오해의 소지가 있다....의도
와 유치를 고려하면 다음과 같이 표현할 수 있
지만, 인코더쪽의 설명을 이리 하면.....

Q / K / V의 대상은 같고, 다만 안의 숫자값만
다름에 유의할 것!!! (그림을 인코더 쪽에서 보
면 디코더쪽 정보가 넘어올 수 없다!!!!)

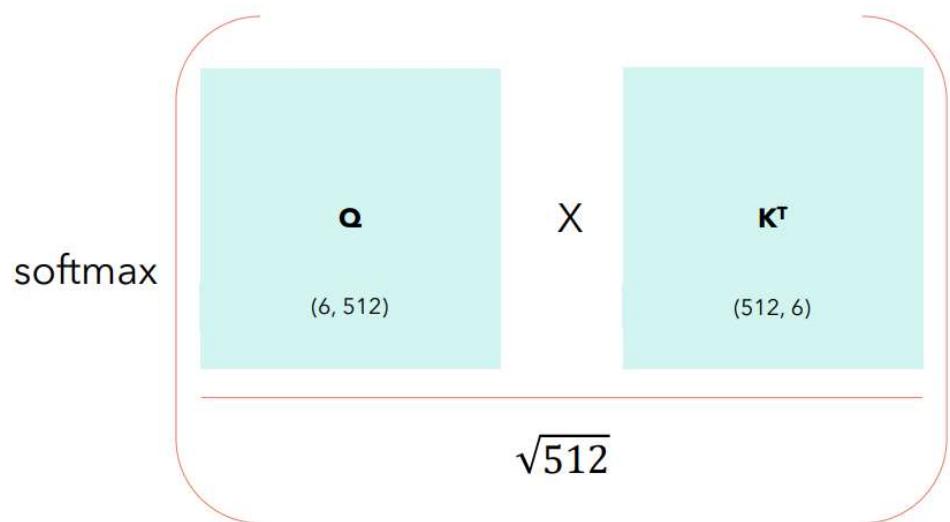
디코더쪽이라고 해도.....인코더쪽의 Q/K가 넘어
가는 것이라서...그림이 틀림!!!!

Scaled Dot-Product Attention



1은 편의상 표
시를 생략함!!!

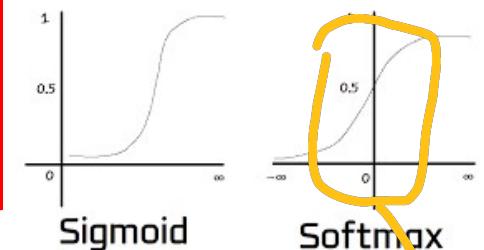
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$d_{\text{model}} = 512$ 의 root의 값으로 분모로 나누는 이유!!!
 → d_{model} 의 값이 클 수록, 1개의 값을 형성하는 것들이
 d_{model} 의 갯수들의 내적들로 형성이 되므로, 1개 단어의
 표현이 길게 할 수록 (d_{model} 이 클 수록) 분사이 크게 되
 어서, 값들이 0 주변에서 기울기가 있을 수 있도록 작은 값
 을 하기 위해서 BatchNormal 유사하게 함.. Softmax도
 sigmoid 처럼 0 주변에서만 기울기 값이 0이 아닌게 있
 으니..

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

(6, 6)



Softmax



YOUR							
CAT							
IS	YOUR	CAT	A	LOVELY	CAT		
A							
LOVELY							
CAT							

가로로 SoftMax

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

(6, 6)

마치 gram-matri처럼 채널들의 정보들이 하나의 값들로 오는 것 처럼, 여기서는 단어들의 조합들이 값들로 온다!!!

$$\begin{array}{c}
 n_H \times n_W \\
 \downarrow \quad \downarrow \\
 \text{Gram Matrix} \\
 \text{Correlation between filters}
 \end{array}
 =
 \begin{array}{c}
 n_C \\
 \downarrow \quad \downarrow \\
 \times \\
 \downarrow \quad \downarrow \\
 \text{Gram Matrix} \\
 \text{Correlation between filters}
 \end{array}$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

(6, 6)

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

(6, 6)

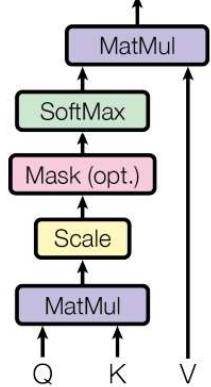
X

V
(6, 512)

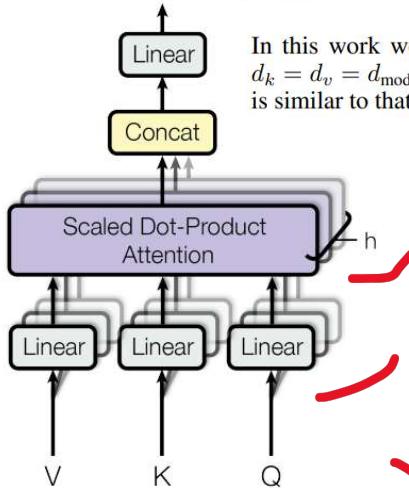
=

Attention
(6, 512)

Scaled Dot-Product Attention



Multi-Head Attention



In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

마치 우리가 Conv에서 모양은 동일한
데, kerne의 수를 여러 개를 하는 것과
같은 역할을 하게 됨!!

여기 논문에서는 8개를 했다고 함!!!
 $Q / K / V$ 를 앞에서 한 것들을 8번...

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

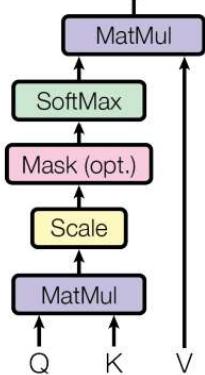
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

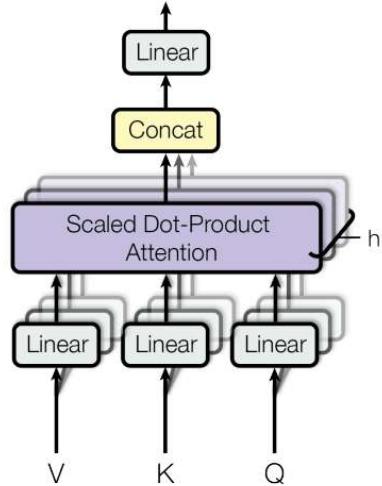
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Scaled Dot-Product Attention

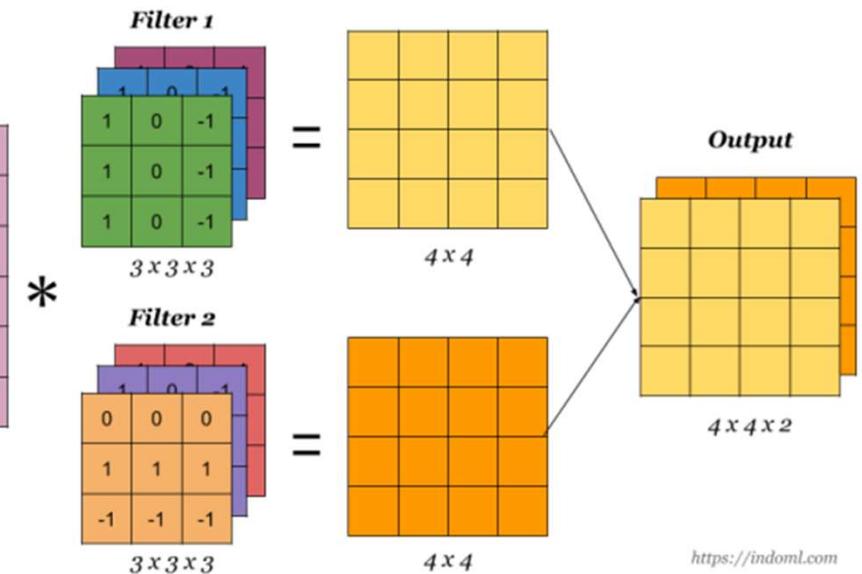
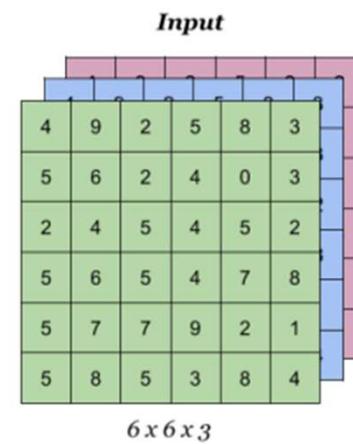


Multi-Head Attention

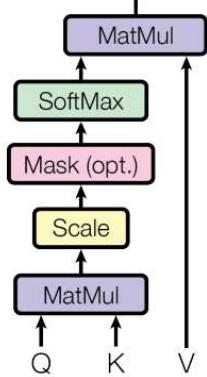


CNN 구조에서 여러 개의 Filter를 사용하는 것들과 유사한 역할!!! → Filter의 수가 Transformer의 Multi Head Attention에서 h의 수와 같은 역할을 함!!

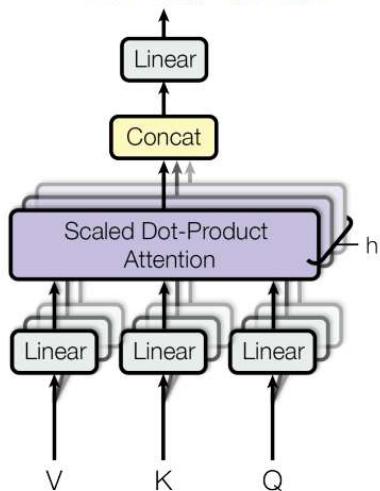
Transformer의 Multi-Head Attention



Scaled Dot-Product Attention



Multi-Head Attention



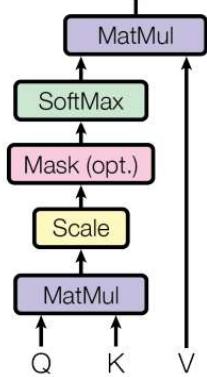
h개를 수행을 하는 것이므로, h장의 $(6 * 512)$ 가 쌓여있고, 여기서는 h=8로 주로 한다고 논문에서 되어 있어서,,

$(8, 6, 512)$

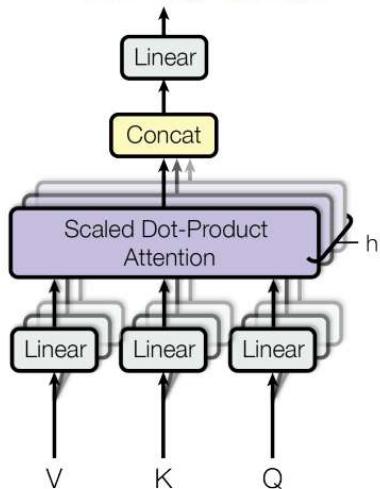
→ 이것이 **Multi Head Attention**!!

→ CNN에서 동일 크기의 필터에도, 여러 개를 사용하면서, 다양한 특징을 뽑아내는 것과 동일한 개념으로, 주목하는 단어들을 다양하게 본다는 비유적인 개념임!!!!

Scaled Dot-Product Attention



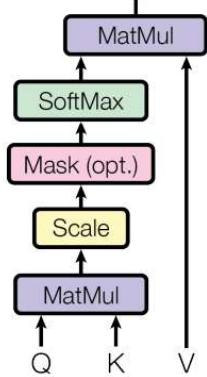
Multi-Head Attention



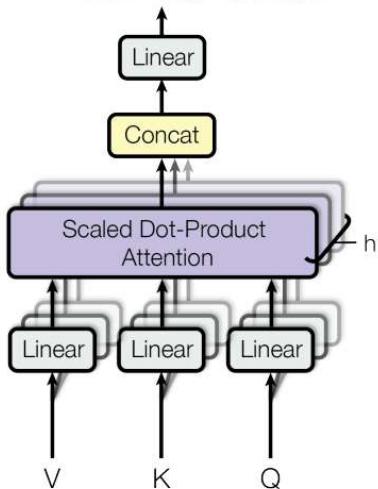
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



Multi-Head Attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

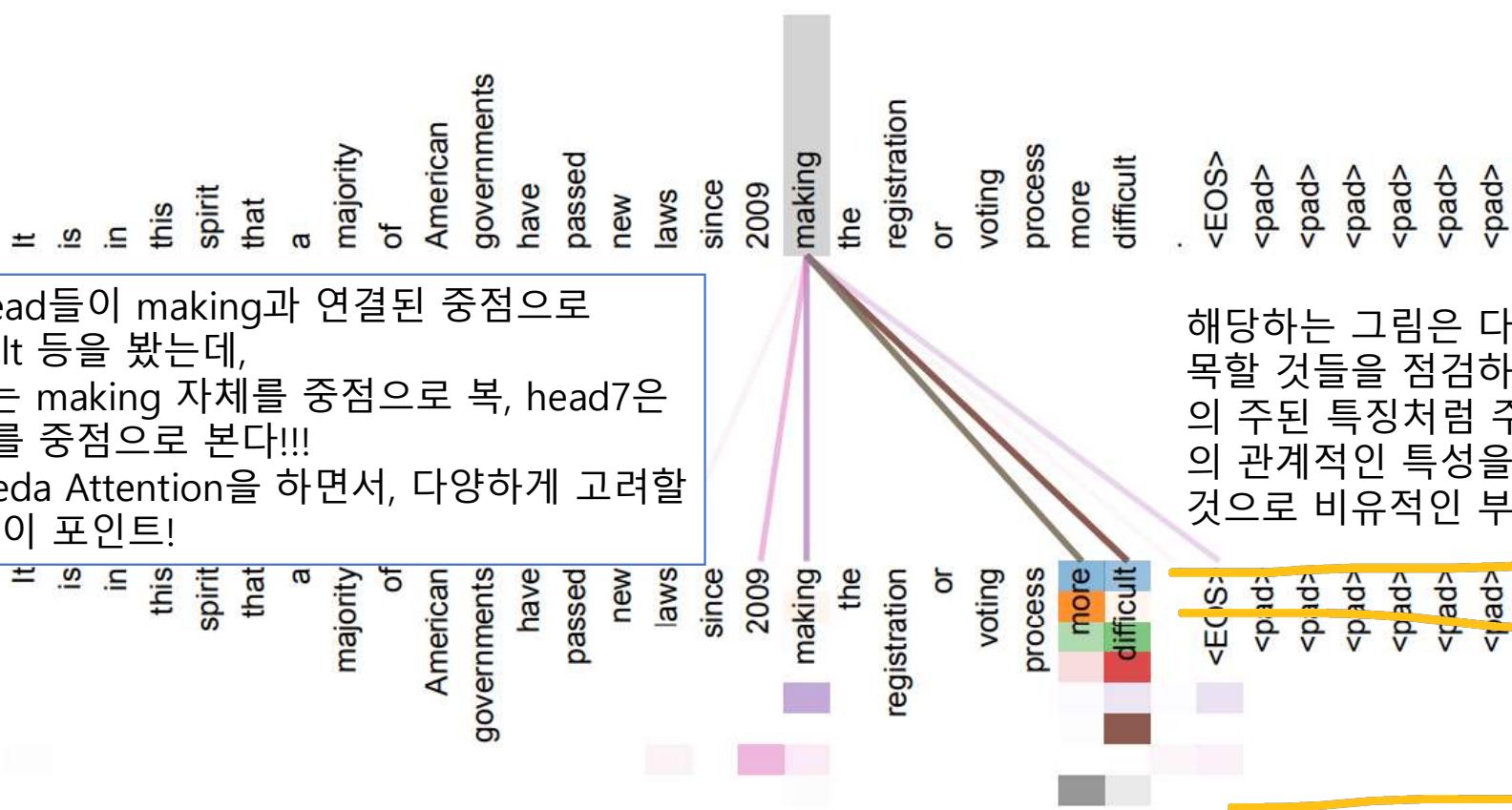
$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Attention Visualizations

한문의 참고부



보면 여러 head들이 making과 연결된 중점으로
more, difficult 등을 봤는데,
Head5 정도는 making 자체를 중점으로 봐, head7은
대상과 시기를 중점으로 본다!!!
→ Multi-Headed Attention을 하면서, 다양하게 고려할
수 있다는 점이 포인트!

Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

해당하는 그림은 다양하게 주
목할 것들을 점검하면서, CNN
의 주된 특징처럼 주된 단어들
의 관계적인 특성을 생성하는
것으로 비유적인 표현임!!!

Transformer의 핵심

- 4. Encoder 전체

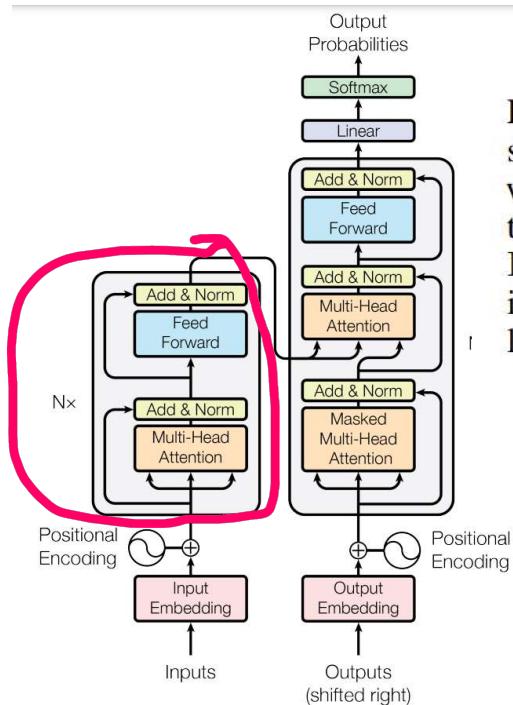
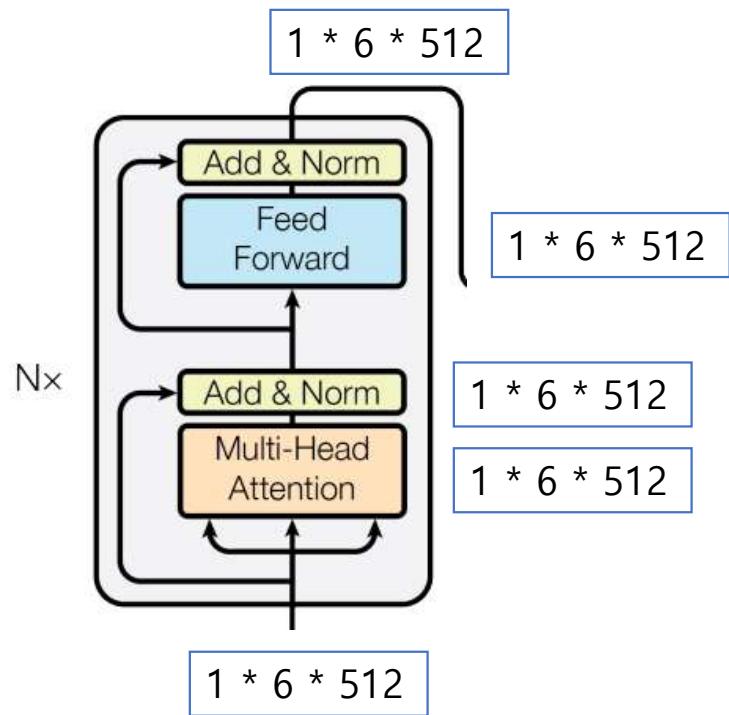


Figure 1: The Transformer - model architecture.

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

- 1) ResNet에서 보던 그 Skip – Connection 사용 : 이유는 조금씩 변화를 통해서 학습하자!!
- 2) Layer Normalization을 사용을 함!! → 바로 뒤에 자세히!
- 3) 그리고 이러한 작업을 계속적으로 반복 N 번이라지만, 논문에서는 6번 반복을 함!!! → 이유는 계속해서 동일한 크기의 모양이 나와서 계속 계속 적층을 N 번을 해도 가능하게 된다!!!! (마치 Conv에서 여러 층 반복할 때 padding 사용해서 크기 유지하면서 하는 것과 동일함!!)



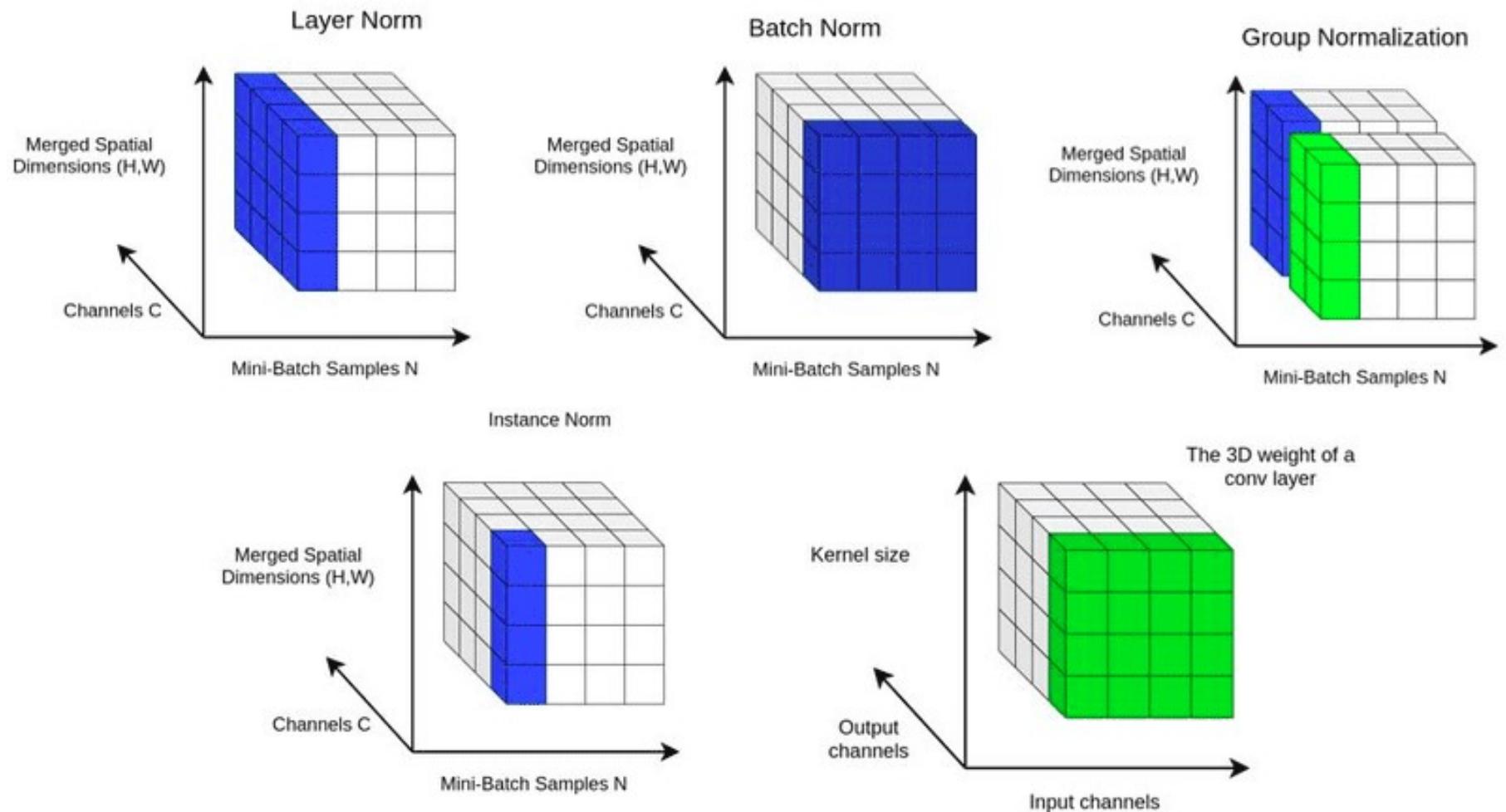
계속 통과할 수록 계속 같은 모양이 되므로, 여러 번 해도 가능하고, Skip-Connection을 해도 가능하고!!!

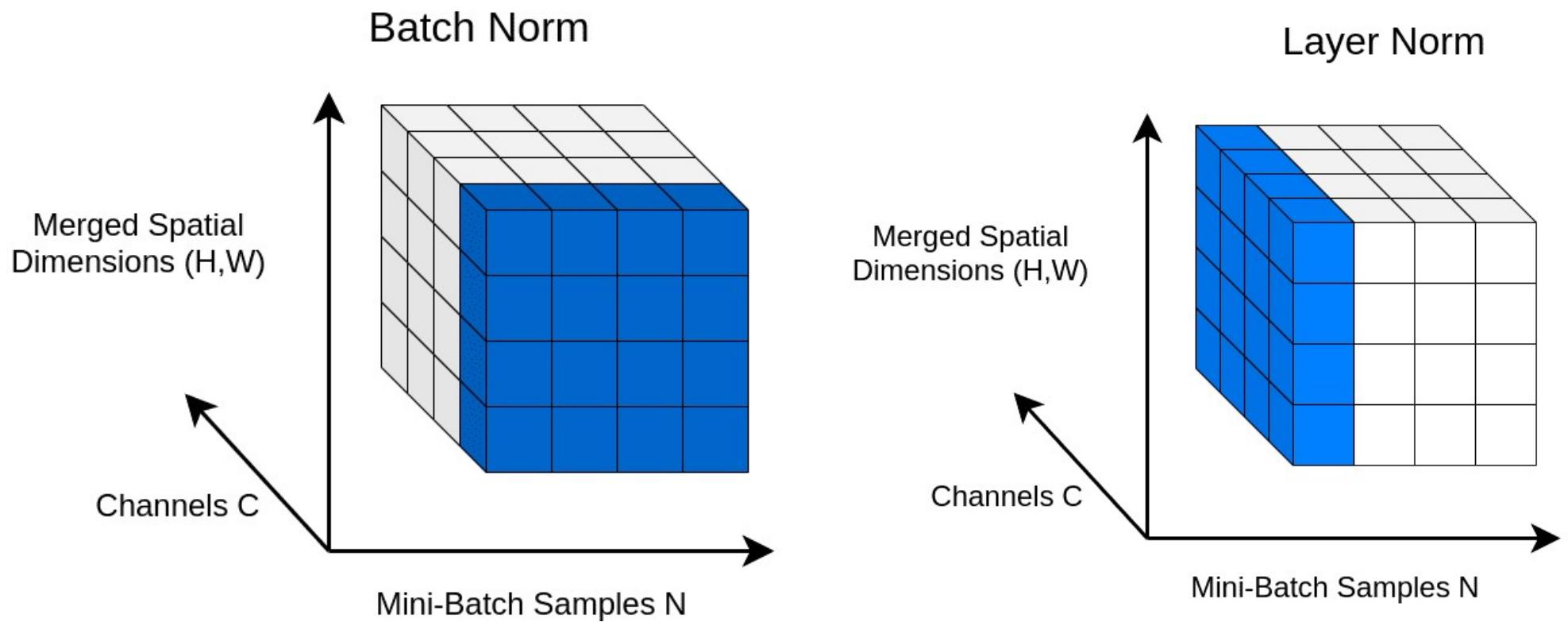
N을 겁나게 키우는 것들이 기본적인 LLM의 기본적이 부분들이 됨!!!

잠시) normalization

- 일반적으로 normalization을 사용하는 이유
 - 입력값들에 대한 위치를 조정을 하는 부분임!!!!!! → position Shifit!!!
- Batch Normalization
 - Batch 별로 normalization을 수행을 함
 - 그 결과 batch_size 값에 따라서 변동성이 있음!!
 - Rnn/ Lstm과 같은 sequential 데이터에서는 제약이 있음.
 - 이유는 이런 시계열 데이터의 경우에는 특정한 길이를 만족시켜주기 위해서 pad값을 사용한다. 그러다 보니 짧은 데이터의 경우에는 pad의 값들이 들어가서 batch_normalization의 효과가 없게 된다.
- Layer Normalization
 - 이것은 뒤의 그림에서 보듯이 batch 단위에서 일어나는 것들이 아니라, 개별 단어에서 이루어지게 된다. 즉, Feature 단위 별로 수행이 되는 것임!!!!!(**입력 데이터의 sequence가 아니라 변형이 된 특징 Feature 단위에서 수행을 함!!!**)
 - 그래서 Transformer 에서는 일반적인 Batch Normalization을 사용하지 않고, Feature 중심의 Layer Normalization 으로 함!!!

여러 방식의 Normalization의 방식들





<https://theaisummer.com/normalization/>

Encoder 끝!!!!

Encoder Transformer의 핵심

- 5-1. Decoder (M-MHA)

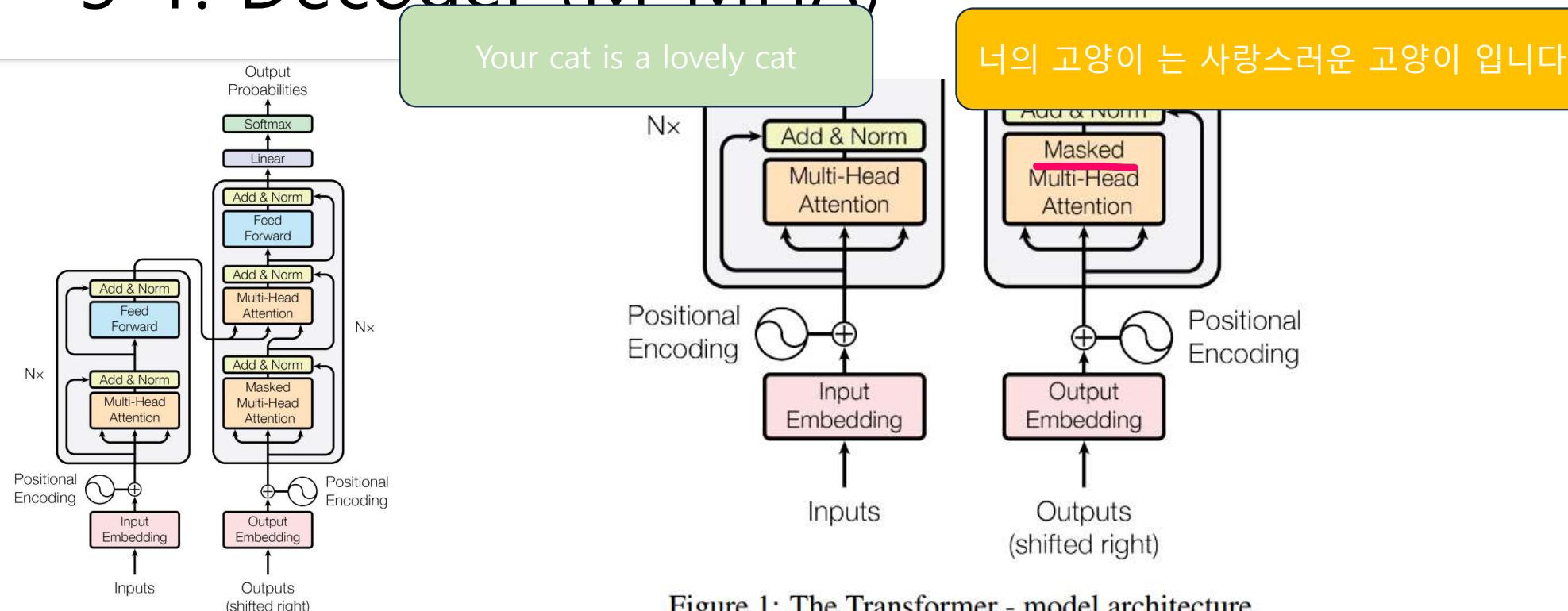


Figure 1: The Transformer - model architecture.

Figure 1: The Transformer - model architecture.

오른쪽 Decoder 쪽으로 생각을 하면, 거의 구조가 유사한데, **Masked Multi Head Attention**이라고 있음!!

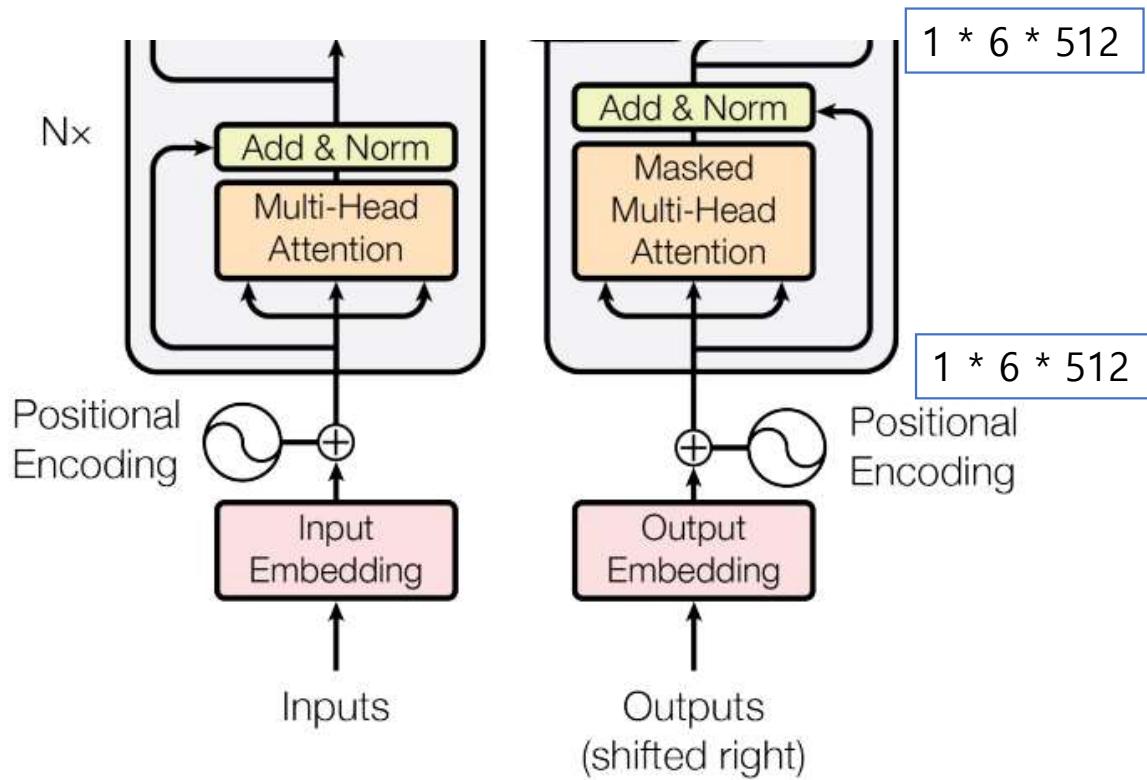
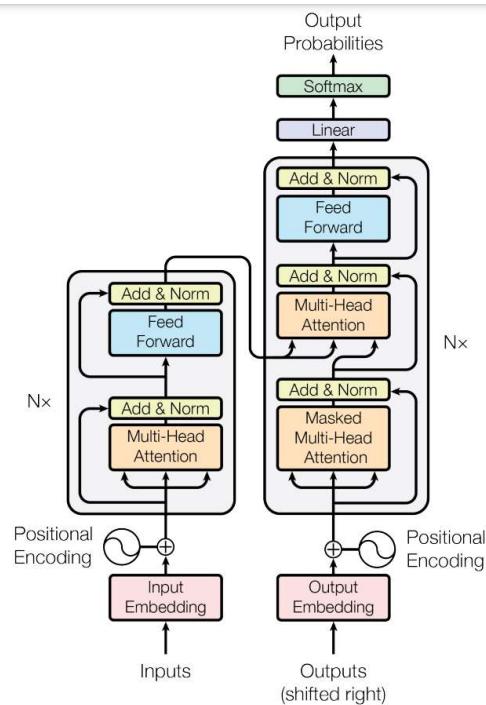


Figure 1: The Transformer - model architecture.

Encoder Transformer의 핵심

- 5-2. Decoder(MHA)



디코더의 MHA에서는
V/K는 인코더의 최종 결과물을 V
/K로 사용을 하고,
Q는 디코더쪽을 사용을 함!!!

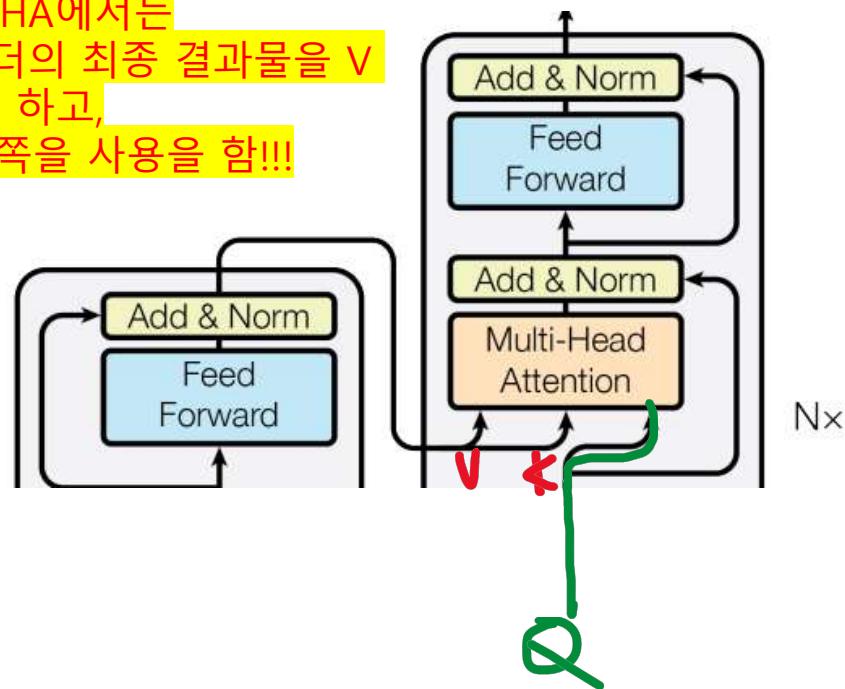


Figure 1: The Transformer - model architecture.

<https://zeuskwon-ds.tistory.com/88>

