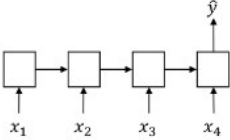
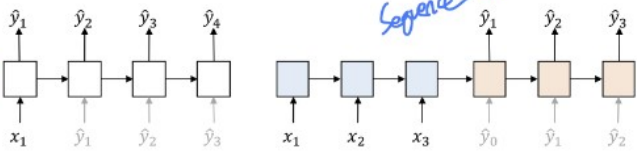
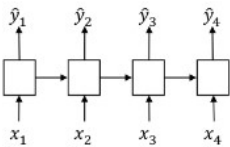


RNN/Lstm/seq2seq/Attention/Transformer 요약

- VIT 공부 이전 배경 공부를 위한 정리
- 쉽게 쓴 RNN, LSTM, Seq2Seq, ... : 네이버블로그 (naver.com) → 여기 참고했음.
- 코드 구현은 mnist dataset에 대해 mini 모델로 학습, 구글 드라이브에 저장

▼ RNN → 구현 O

https://search.pstatic.net/common/?src=http%3A%2F%2Fblogfiles.naver.net%2FMjAyMjEwMzFfMTU4%2FMDAxNjY3MjE1MjQ0MTc2.QhCEYdJF_QsOwdiSkfbl6egAhvXgTXrbC3f6EyJO4g.oy._e3gUQdakTx4kGnqfO7vHYEA-N9F4gs3HNNQlrycog.JPEG.dbwjd516%2Frnn.jpg&type=sc960_832

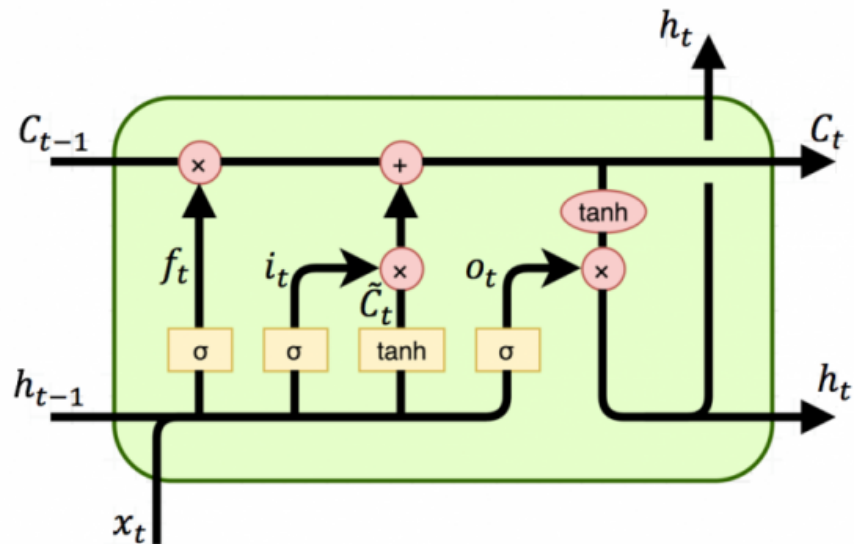
Type	Architecture	Applications
Many to One		Text Classification
One to Many		NLG, Machine Translation Generative Modeling
Many to Many		POS Tagging, MRC 생각은 -

- RNN: 출력 벡터가 다시 입력 벡터로 들어감, 시간적인 흐름 학습
- 단점
 - 병렬화 불가능 (입력 → 출력 → 출력이 다시 입력 | 이 순서기 때문)
 - vanishing gradient

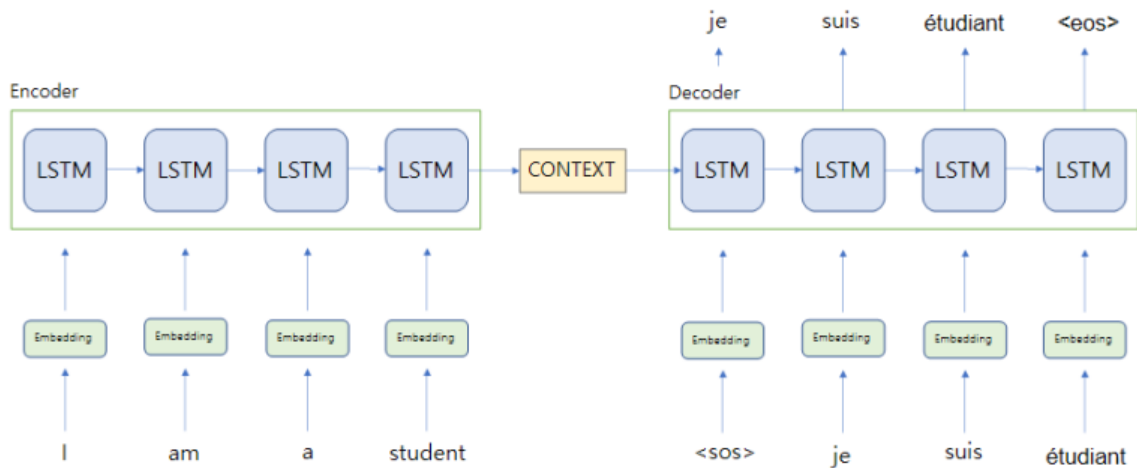
▼ LSTM → 구현 O

기존의 RNN 문제점을 해결해보자!

https://search.pstatic.net/sunny/?src=https%3A%2F%2Fwww.mdpi.com%2Fsustainability%2Fsustainability-13-05400%2Farticle_deploy%2Fhtml%2Fimages%2Fsustainability-13-05400-g004-550.jpg&type=sc960_832



- RNN 구조 기반
 - 특징
 - 전체 정보를 기억할 수 있는 layer 존재
 - forget gate(x) 존재 → 과거의 데이터를 얼마나 기억할 것인가?
 - input gate 존재 → 새로운 데이터를 얼마나 받아들일 것인가?
 - output gate 존재 → 앞선 두 gate 통한 정보를 얼마나 전달할 것인가?
 - 단점
 - 여전히 느린 병렬 처리
 - 여전히 기존 정보가 대부분 덮어 쓰여질 가능성(forget gate의 sigmoid 연산으로 인함.)
 - 변위 → GRU
 - LSTM구조를 간소화 함
- ▼ seq2seq → 구현 X



- Encoder → Decoder 구조
 - Encoder에서 모든 정보를 압축하여 하나의 벡터로 만들.
 - Decoder에서 context벡터를 받아 단어를 순차적으로 출력.
 - CONTEXT_1 + Decoder(<sos>) → je , CONTEXT_2
 - CONTEXT_2 + Decoder(je) → suis, CONTEXT_3
 - CONTEXT_3 + Decoder(suis) → etudiant, CONTEXT_4
 - 의의:
 - 기존의 LSTM은 순차적인 입력으로 문장 구조가 다르면 소용 없음. → 이를 해결
 - 고정된 크기의 context vector사용
 - 특징:
 - output를 거꾸로 출력하게 만들면 효과가 더 좋아진다
 - 교사 강요 방법으로 훈련
- ※ **교사 강요(teacher forcing)**
- 훈련 과정에서 이전 시점의 출력을 현재 시점의 입력으로 넣어주게 될 때
 이전 시점의 출력이 잘못된 경우 현재 시점의 예측도 잘못 될 가능성이 높아지고
 결국 연쇄적으로 Decoder 전체가 잘못될 수 있다.

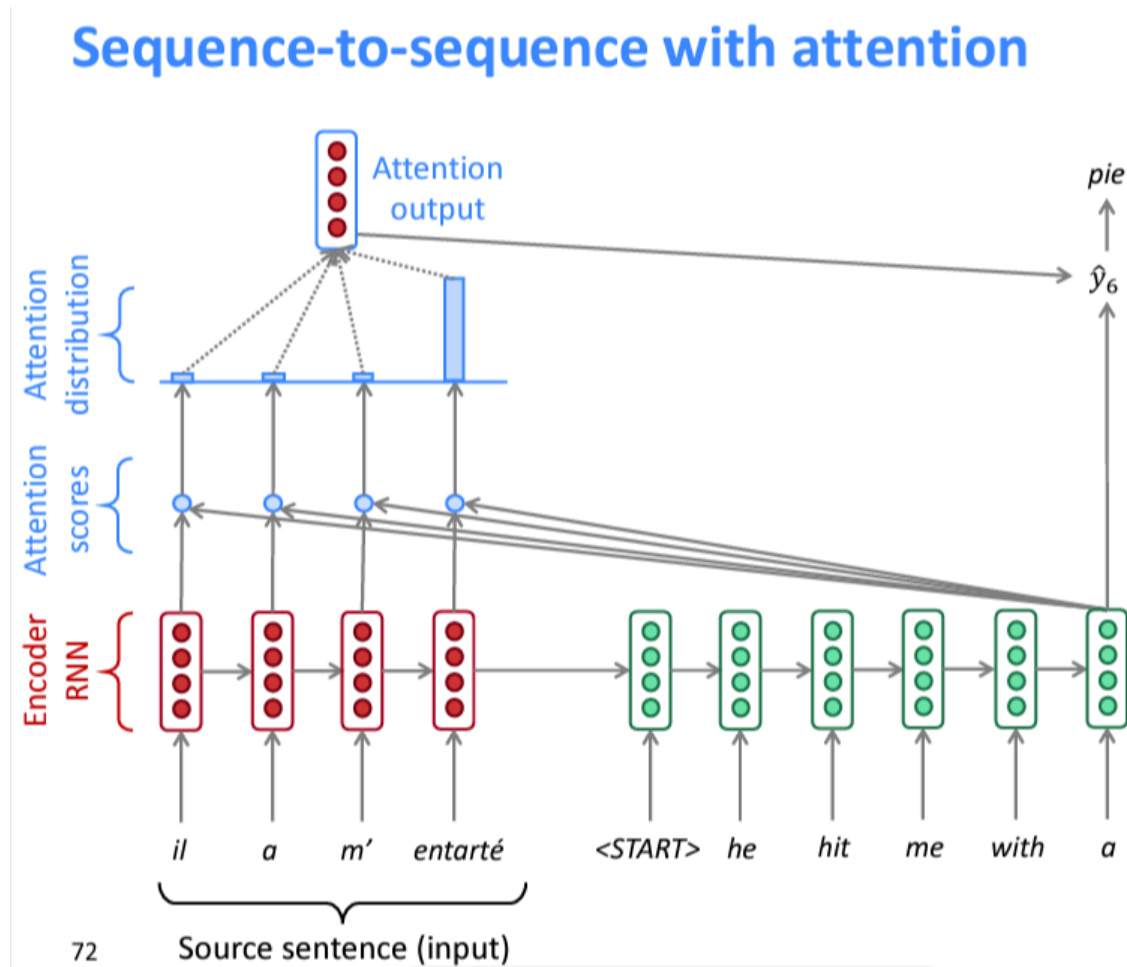
때문에 훈련 과정에서는 이전 시점의 출력(예측값)을 현재 시점의 입력으로 넣어주지 않고
 이전 시점의 실제 값을 현재 시점의 입력으로 넣어주는 방법인 교사 강요(teacher forcing)를 사용한다.

[출처] (딥 러닝) seq2seq|작성자 wooy0ng

- 한계:
 - 여전히 긴 문장에 대해서는 약한 모습을 보임
 - → RNN의 고질적 문제 gradient vanishing

▼ Attention → 구현 X

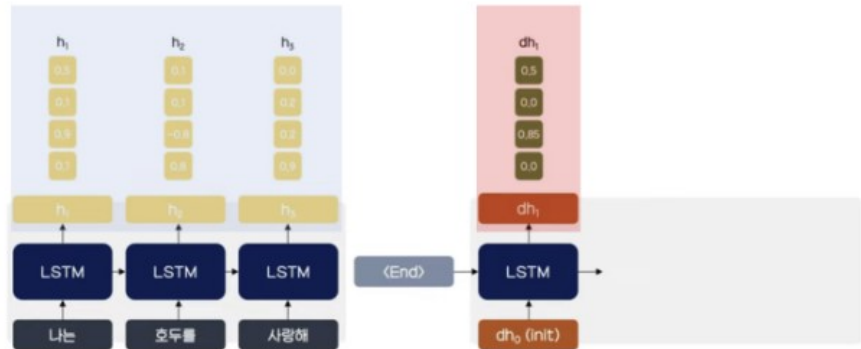
- 기존의 하나의 context 벡터로는 모든 정보를 표현할 수 없다!
- 모든 문장 소스를 입력으로 사용하자! → Attention



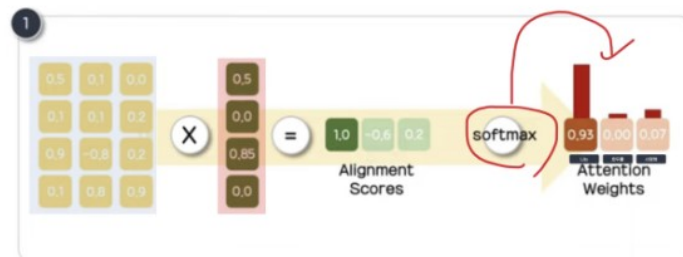
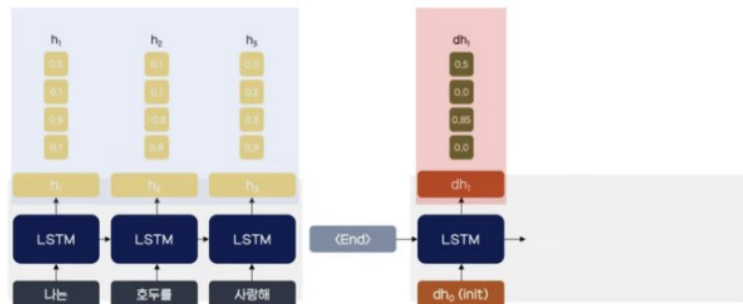
1. 각 input 단어들을 돌면서 각각의 attention score(hidden state(1) 행렬) 구함
2. Decoder part에서 <start>가 input → 새로운 hidden state(2) 행렬 생성
3. hidden state(2)와 Encoder part 각 단어의 hidden state(1) 각각을 dot product 연산함.
4. 그러면 [단어 길이]의 숫자들로 이루어진 alignment scores list가 나옴
5. 이를 softmax 취하여 attention distribution를 구함
6. attention distribution은 [66%,5%,10%,19%] 와 같은 확률 분포임.
7. attention distribution과 Encoder part hidden state(1)들을 행렬 곱 연산하여 새로운 hidden state(3)를 구함
8. 새롭게 구한 hidden state(3)과 hidden state(2)를 concat(행렬 붙이기)하여 이를 통해 <start> 이후 토큰 (he) 예측
9. Decoder part 다음 토큰인 he 역시 2~8 과정 반복
10. 문장 종료 기호가 Decoder part에서 출력 되면 해당 과정 종료

▼ 그림으로 이해하기

Attention
(2015)

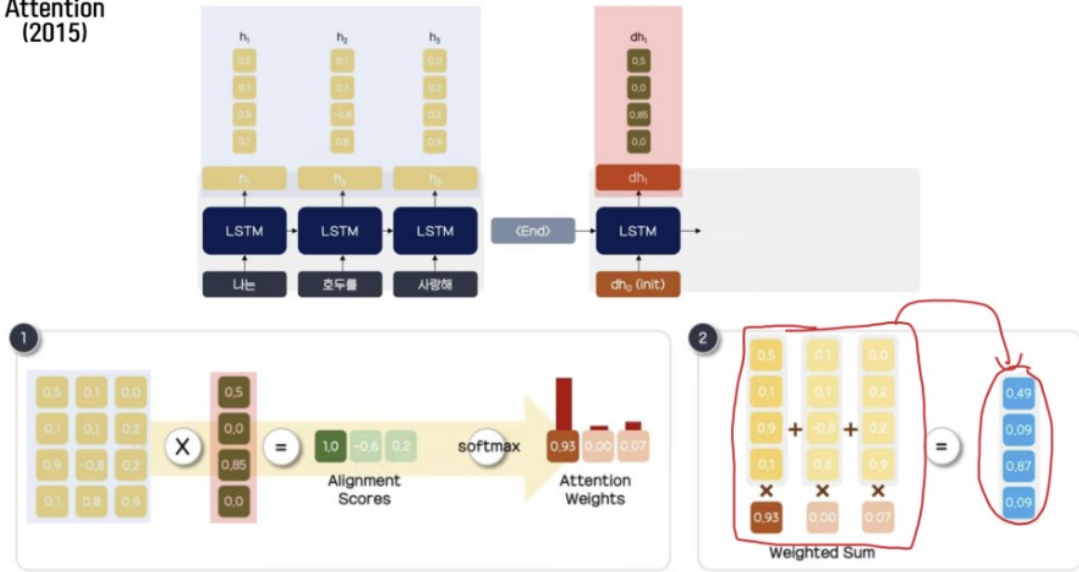


Attention
(2015)

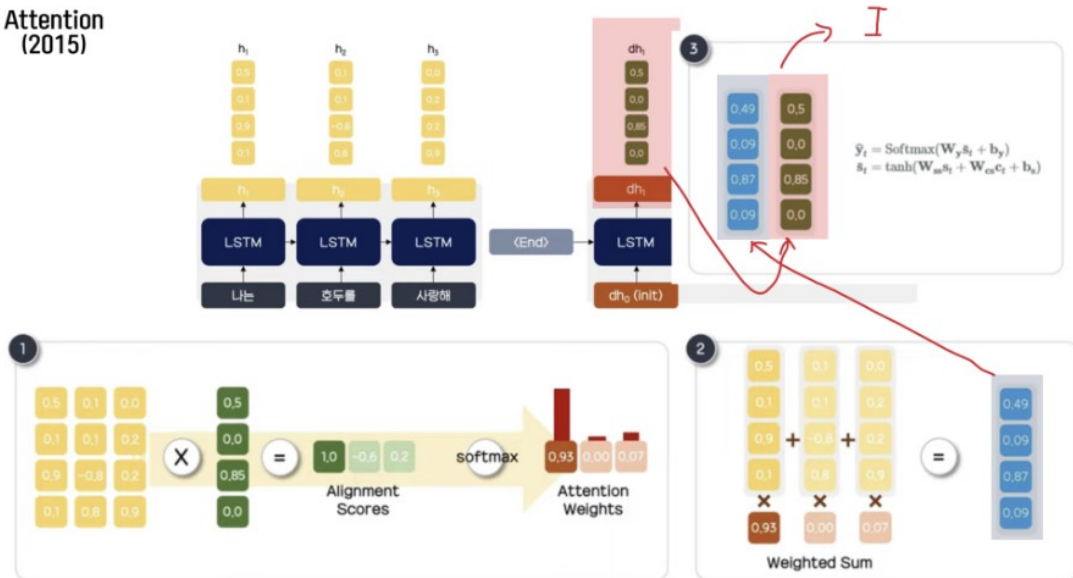


Start of Sequence는 93%로 "나"와 연관이 있다.

Attention (2015)



Attention (2015)



• 의의

- Attention이 이전 모델들보다 뛰어난 성능을 보인 이유는 문장의 길이에 상관없이 강한 모습을 보이기 때문

Attention Visualizations

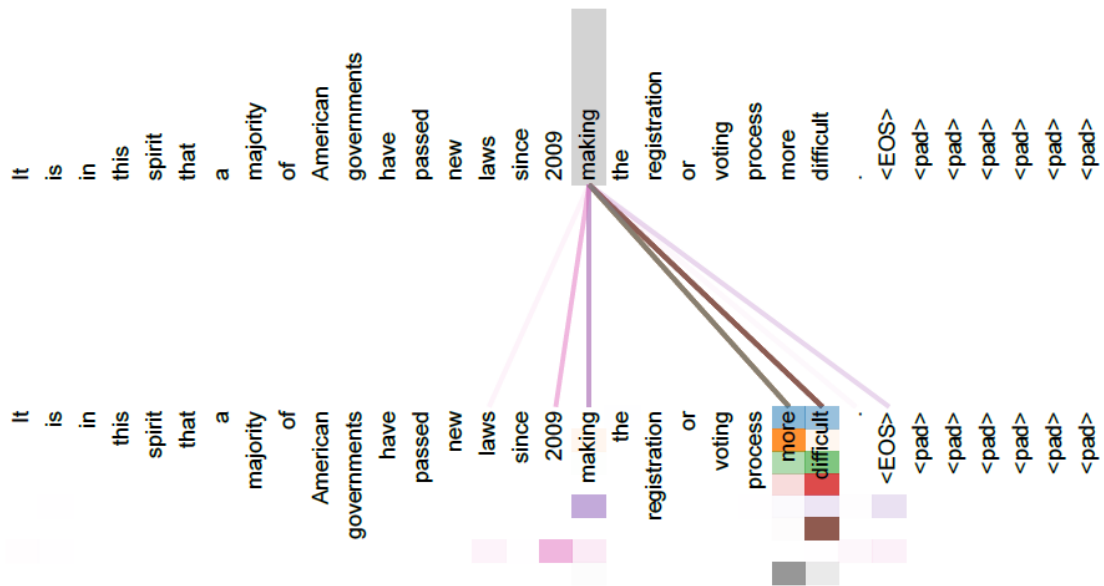


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word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

▼ Transformer → 구현 X

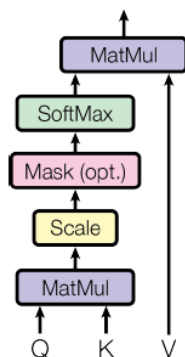
- Attention is all you need(2017)
- [Attention is all you need paper 뽀개기 | 포자랩스의 기술 블로그 \(pozalabs.github.io\)](https://search.pstatic.net/common/?src=http%3A%2F%2Fblogfiles.naver.net%2FMjAyMjAxMjRfMTk2%2FMDAxNjQyOTg4Njk5OTc5.jwZcATXTIy6G120jdqdKdP4b73ZOoYTFk6MRRRCP4Lcg.-nSYGEBIClmKEoS1v2Nr8dun1nVRxifcj_idVHzB7C0g.PNG.hanchaa%2Fimage.png&type=sc960_832)

https://search.pstatic.net/common/?src=http%3A%2F%2Fblogfiles.naver.net%2FMjAyMjAxMjRfMTk2%2FMDAxNjQyOTg4Njk5OTc5.jwZcATXTIy6G120jdqdKdP4b73ZOoYTFk6MRRRCP4Lcg.-nSYGEBIClmKEoS1v2Nr8dun1nVRxifcj_idVHzB7C0g.PNG.hanchaa%2Fimage.png&type=sc960_832

- Encoder / Decoder 구분
- 개요
 - 기존 모델들은 여전히 parrallizing(병렬화)가 불가능하다는 문제
 - attention을 RNN의 보정을 위한 용도가 아니라 아예 attention으로 인코더와 디코더를 만들어보면 어떨까?
- 공통 내용
 - positional encoding
 - 각 블록마다 residual connection 존재
 - encoder, decoder block이 각각 6개씩 쌓여 있음.

- layer normalization
- Encoder layer
 - multi-head self attention
 - position-wise fully connected feed forward network
- Decoder layer
 - Encoder layer와 같은 sub layer 가짐
 - multi-head self attention
 - position-wise fully connected feed forward network
 - Encoder의 output에 multi-head attention 수행
- positional encoding
 - 기존의 RNN/CNN 방법에서 벗어났기에, sequence의 위치에 관한 정보를 주입해야 함.
 - $PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{model}})$
 - $PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{model}})$
- multi-head self attention

Scaled Dot-Product Attention



Multi-Head Attention

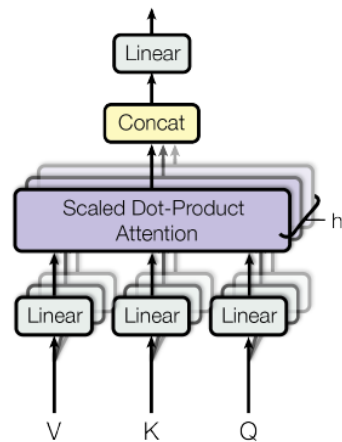
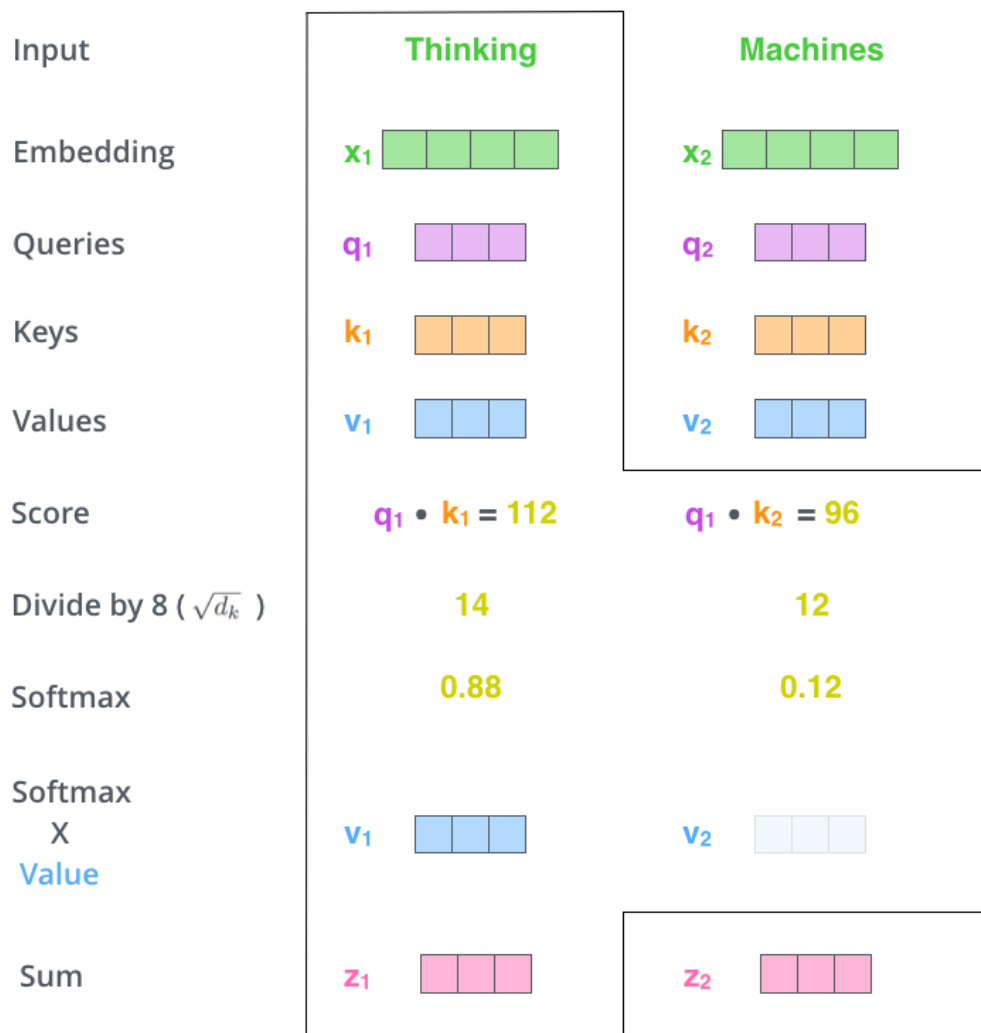


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

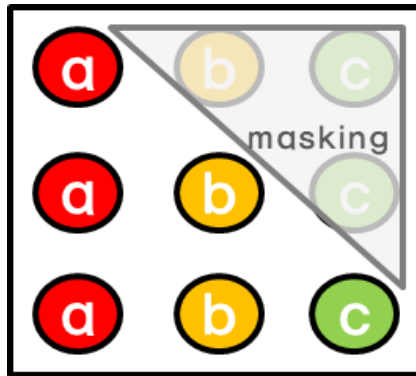


- query / key / value 는 self-attention 과정에서 각각의 vector들을 의미
 - attention 에 대해서 생각하고 계산하려 할 때 도움이 되는 추상적인 개념
 - attention에서 입력 원소 각 단어들의 hidden state ~~ key
 - attention에서 output 원소 단어의 hidden state ~~ query
 - 데이터들의 정보 ~~ value
 - [The Illustrated Transformer – NLP in Korean – Anything about NLP in Korean](#) → 참고
- matmul : 행렬 곱 연산
- attention score을 구할 때 현재 단어의 query vector와 점수를 매기려 하는 다른 위치에 있는 단어의 key vector의 내적으로 계산
- $attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$
- multi-head → attention이 동일 형태로 여러 개 존재
 - 이들 각각 계산 이후, concat(합침) → Linear
 - $MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$

- $head_i = Attention(Q_W Q_i, K_W K_i, V_W V_i)$

- self-attention layer

- self-attention은 한 문장 사이에서 각 단어들 사이의 관계를 표현할 수 있다.
- 모든 key, value, query는 같은 곳(인코더의 이전 layer의 출력)에서 온다. 따라서 인코더의 각 원소는 이전 layer의 모든 원소를 고려할 수 있다.
- 디코더에서는 다르다. auto-regressive 속성을 보존하기 위해 디코더는 출력을 생성할 시 다음 출력을 고려해서는 안 된다. 즉, **masking**을 통해 이전 원소는 참조할 수 없도록 한다.
 - masking 방법: dot-product를 수행할 때 원하는 부분을 $-\infty$ 로 설정, softmax를 통과할 때 0이 되는 효과



- position-wise feed forward network

- 재귀적인 구조 없이, 앞으로만 전파되는 network
- FC(fully conned layer)는 모두 여기에 해당
- $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$

seq2seq ~ Transformer 코드 구현은 생략, VIT 코드 구현으로 대체

→ seq2seq는 구현 가능 할 수도? 시간만 된다면..