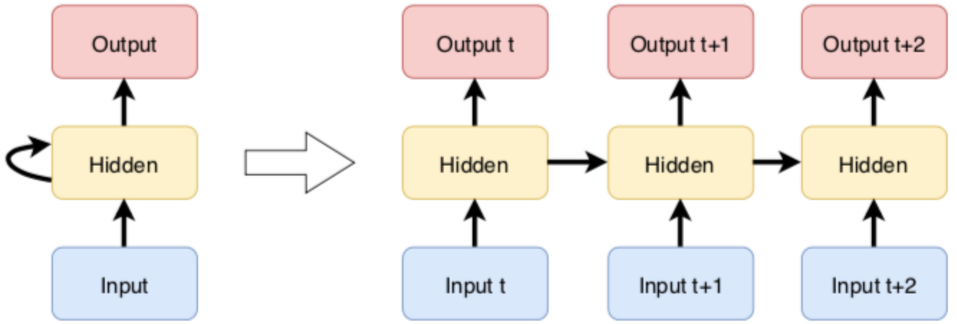
AI기법과 활용

Week-09. Natural Language Processing Part 2

2022-Summer 서중원

Recurrent Neural Networks

일반적인 순환신경망의 한계

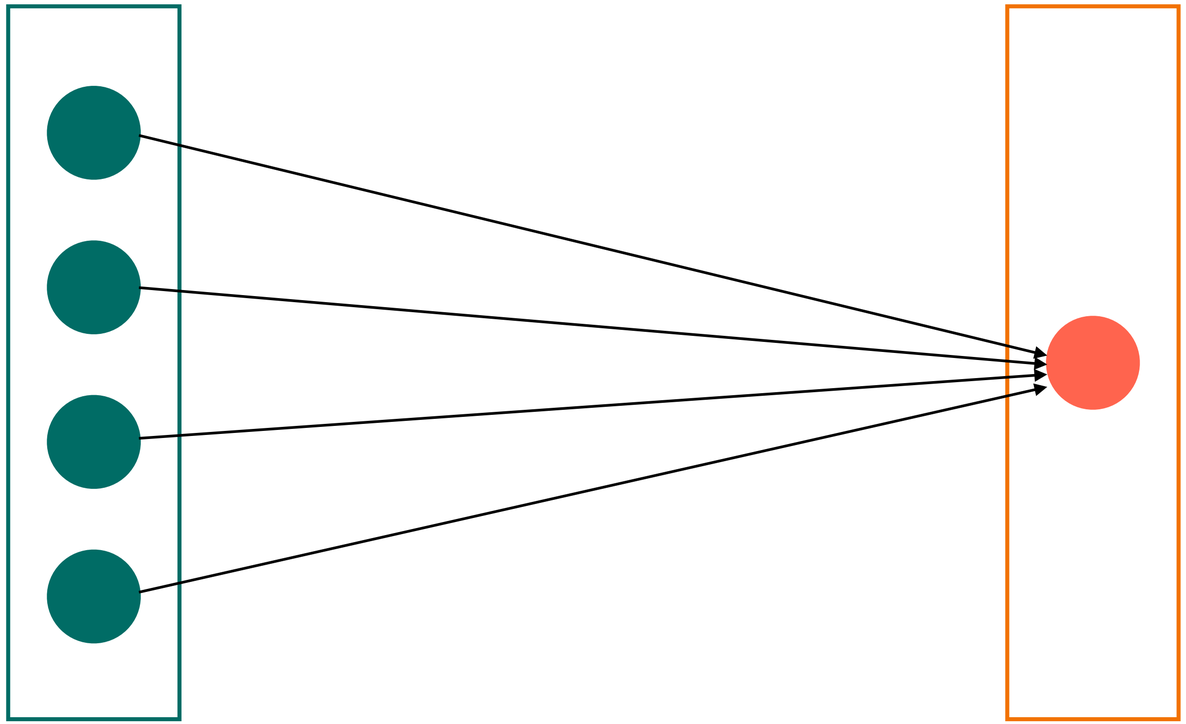


- Sequence Length에 의한 영향
  - Input이 길어지면, 초반에 등장한 단어의 영향이 적어진다
  - Vanishing gradient
- 단어의 순서는 고려하나, 중요한 단어가 무엇인지는..
  - 머신러닝은 앞으로도 그리고 이전에도 매우 많은 사람들에게 흥미로운 주제지만, 딥러닝은 ..
- 연산 속도
  - Sequence가 존재하기 때문에, 이전 input이 처리(병목)되기 전에 다음 input을 처리 할 수 없다.
    - 병렬적으로 학습이 진행되지 않음!

RNN의 한계를 어떻게 극복할까?
다음 시간에..

Logistic Regression

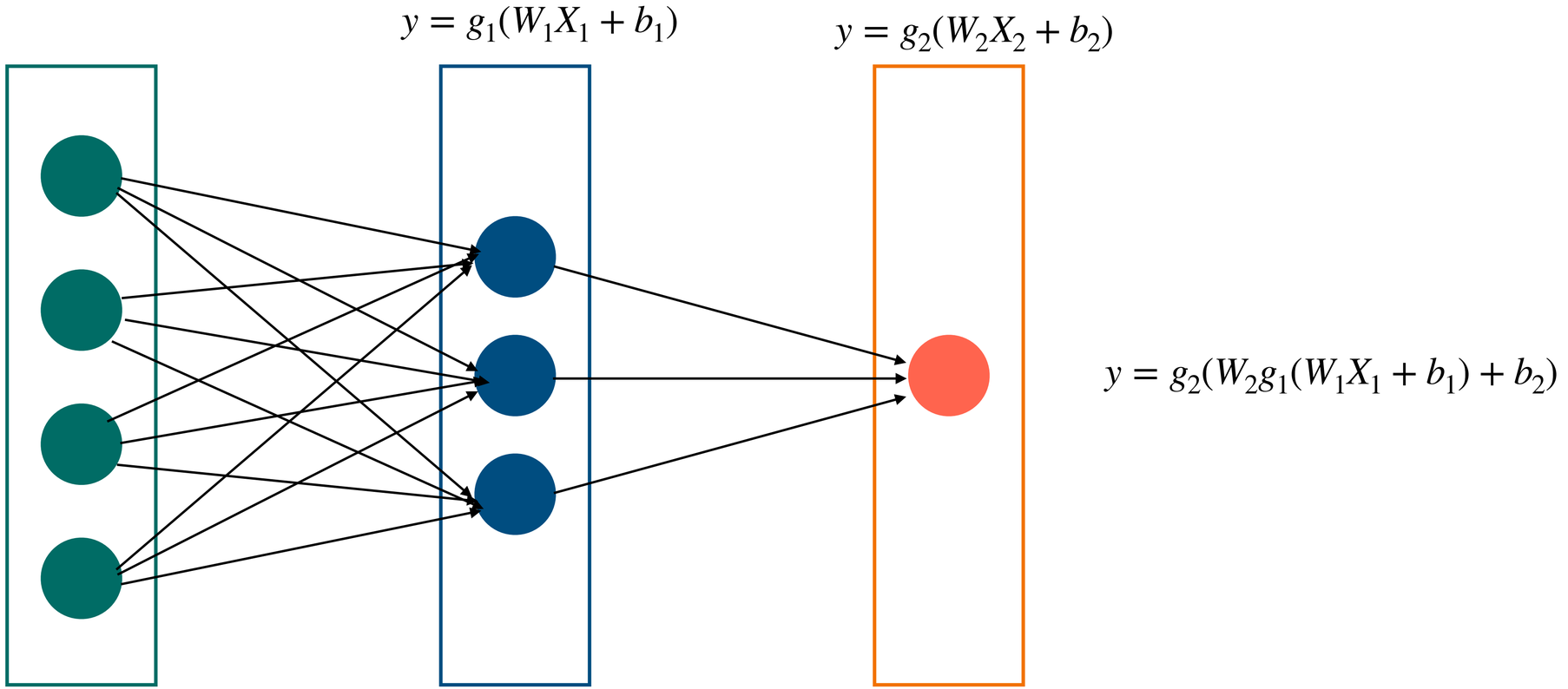
How it works



$$y = g(WX + b)$$

Artificial Neural Networks

How it works

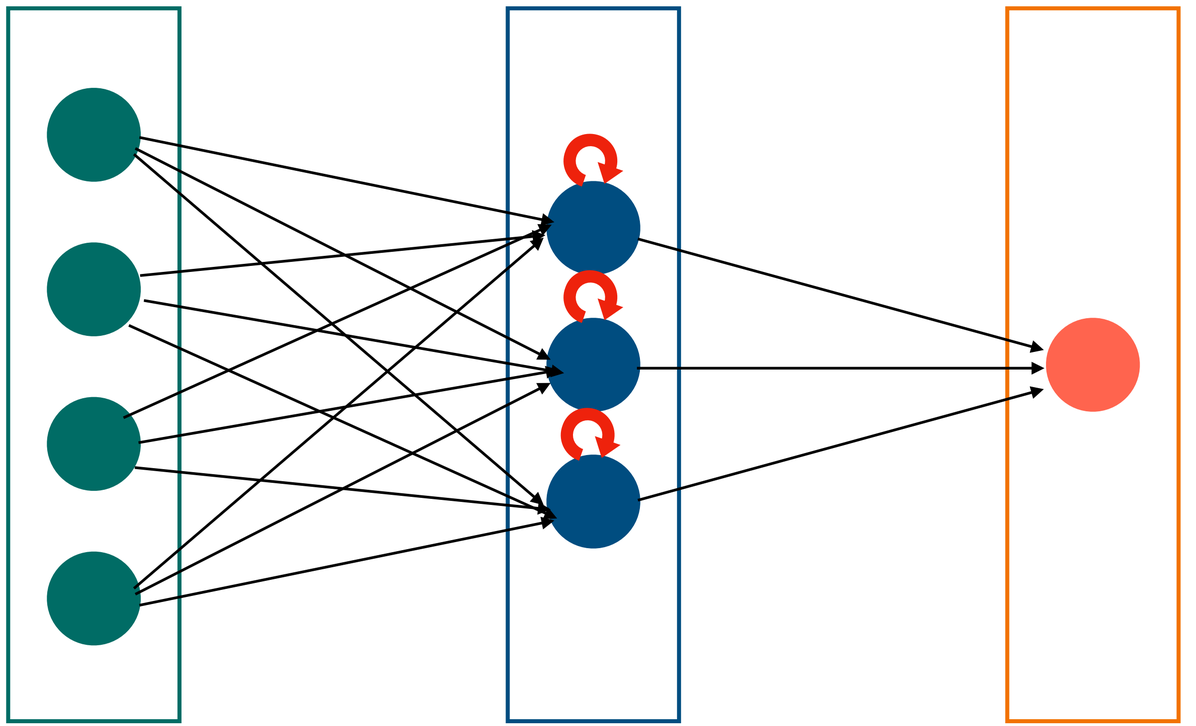


Recurrent Neural Networks

How it works

$$y^t = g_1(y^{t-1} + W_1 X_1{}^t + b_1)$$

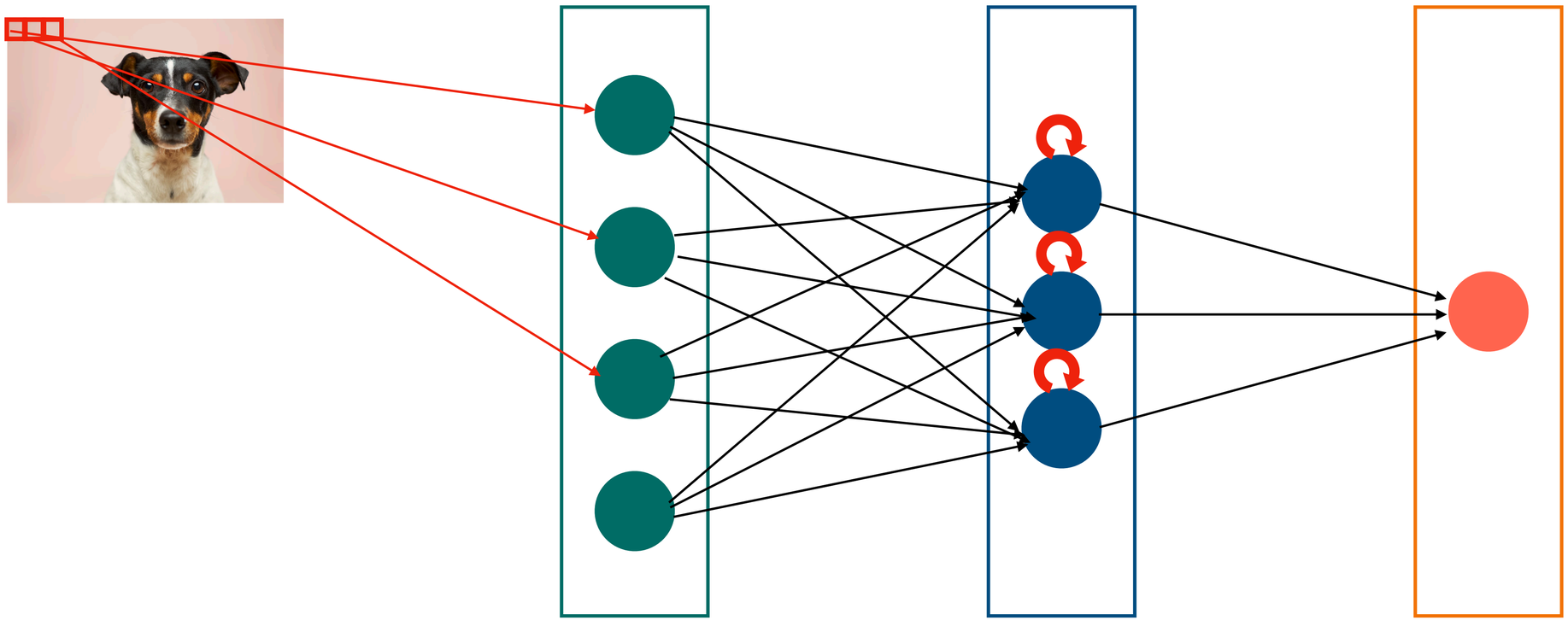
$$y^t = g_2(W_2 X_2{}^t + b_2)$$



$$y = g_2(W_2 g_1(y^{t-1} + W_1 X_1{}^t + b_1) + b_2)$$

RNN을 이미지 분류에 활용 할 수 있을까?

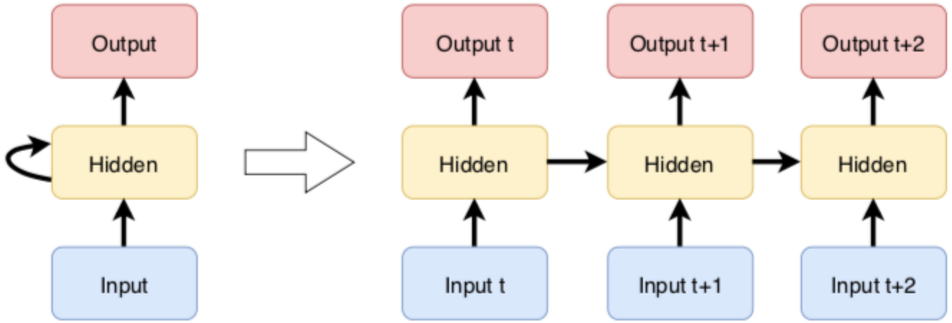
Yes

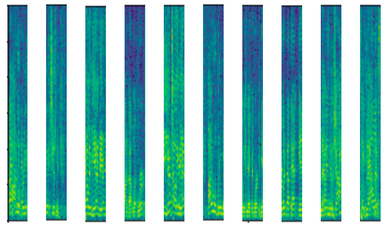


RNN을 이미지 분류에 활용 할 수 있을까?

그런데 의미가 있을까?

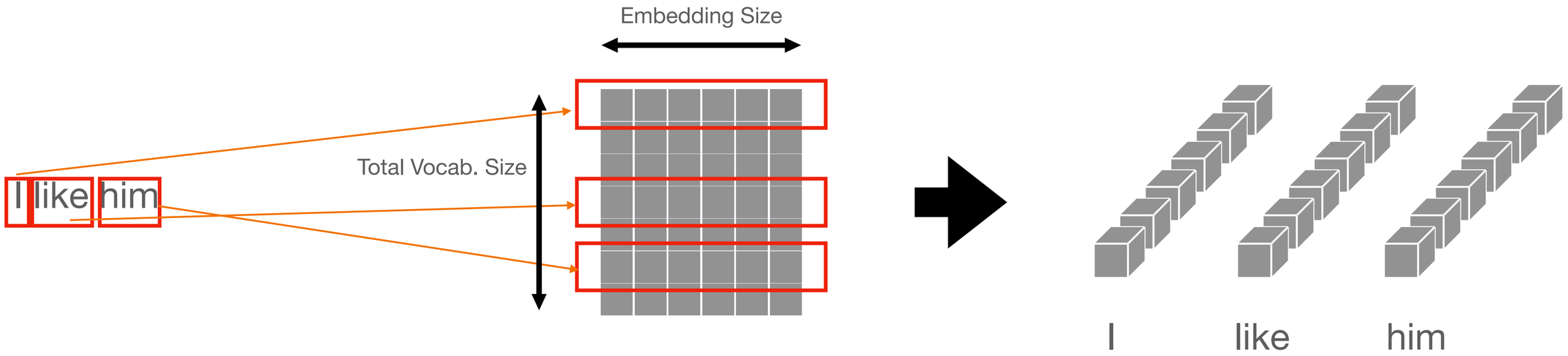
- 한 이미지를 한번에 넣으면서 학습시, 시퀀스 길이가 1인 input
- 그럴 경우 일반 ANN과 차이가 없음
- 그렇다면 시퀀스 길이를 갖게끔 이미지를 재구성 하여 입력 한다면 의미가 있을까?
  - 이미지가 좌-우 또는 상-하에 의한 순서적 의미를 갖고 있는 경우 Yes!





Recurrent Neural Networks

Input Dimension



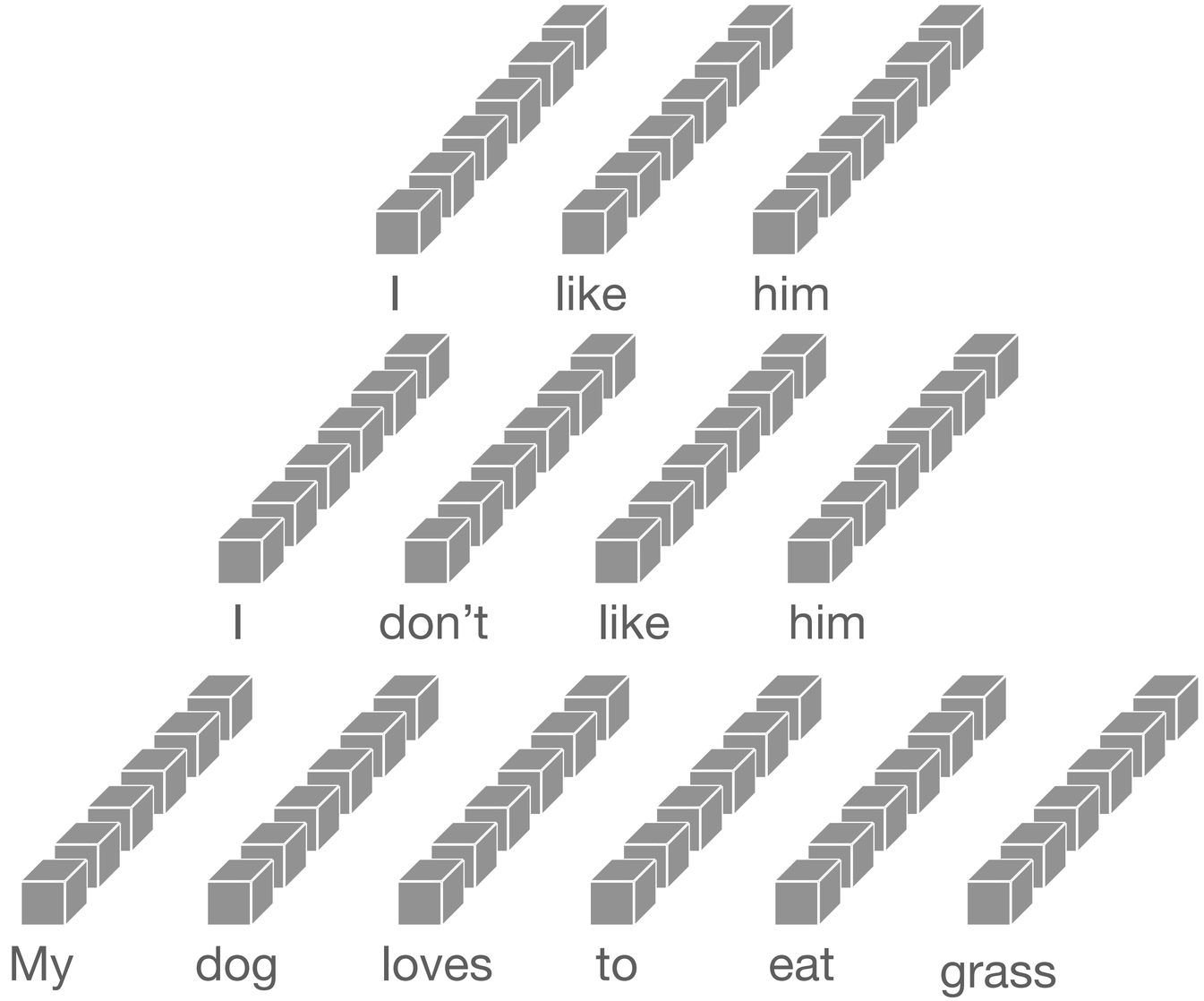
Recurrent Neural Networks

Sequence Length

I like him

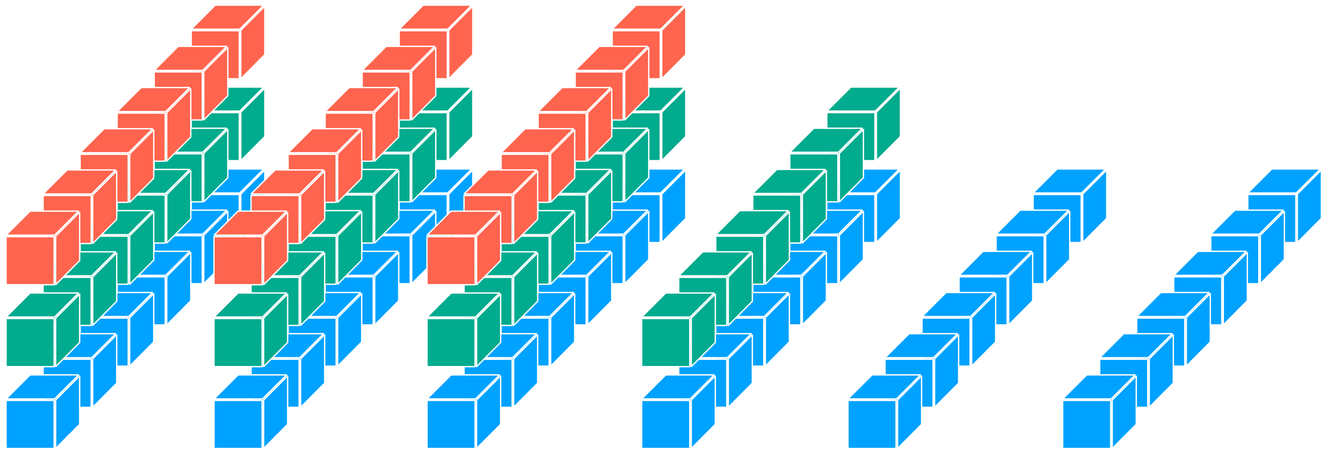
I don’t like him

My dog loves to eat grass



Recurrent Neural Networks

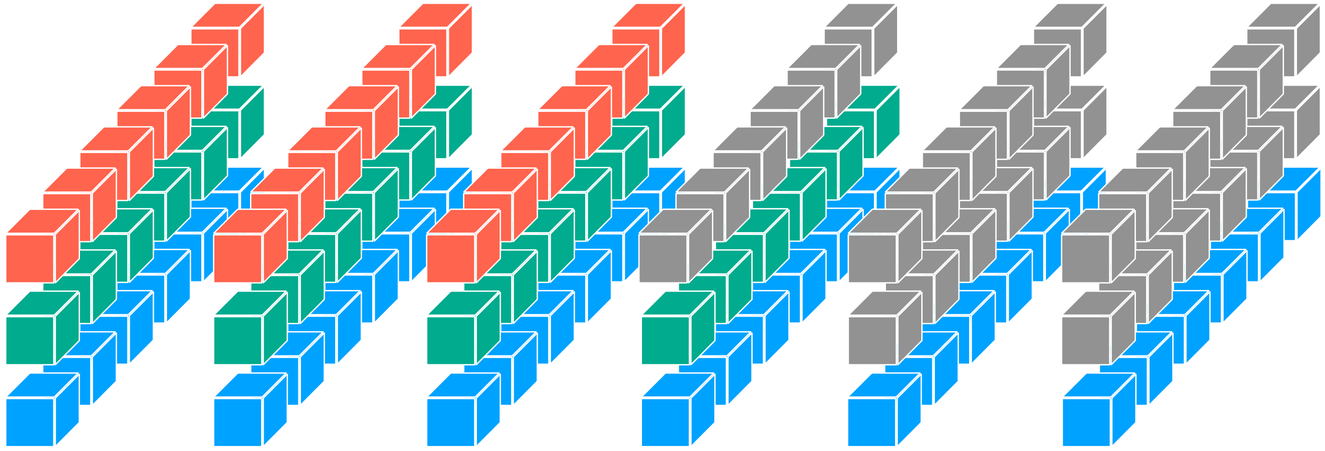
Sequence Length

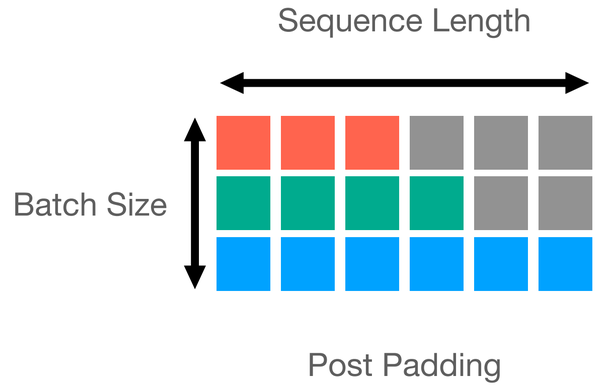


Recurrent Neural Networks

Post Padding

- 효율적인 행렬 연산을 위해 모든 Input의 길이(max sequence length)를 동일하게 가져감
- 길이가 부족한 Input은 0으로 채우고, 길이가 넘는 문장은 잘라냄

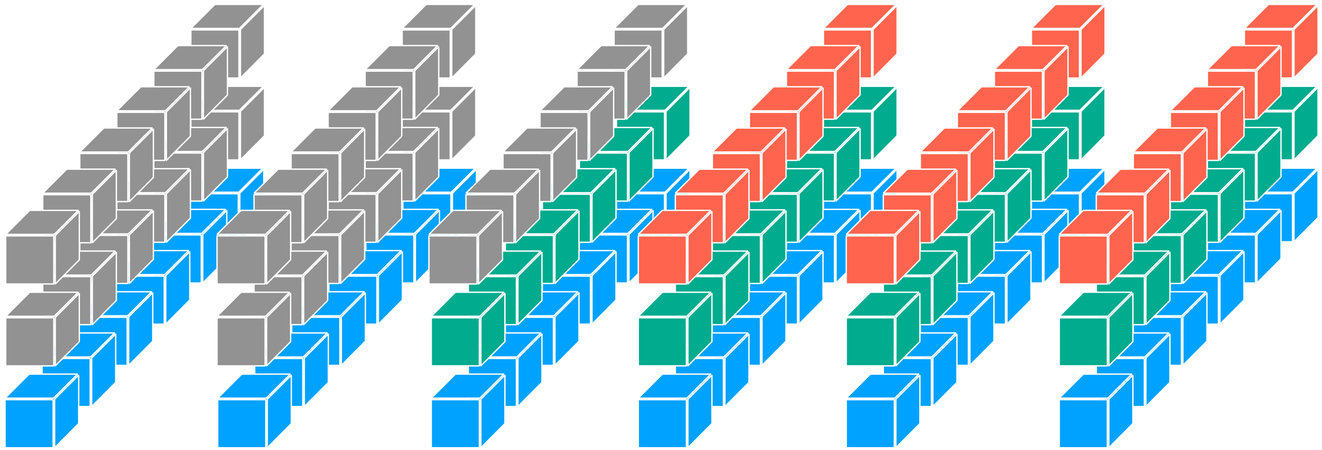


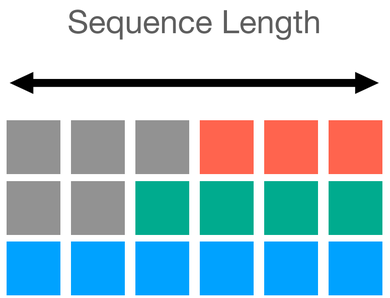


Recurrent Neural Networks

Pre Padding

- NLP의 Post Padding의 경우 Sequence의 마지막 부분이 0이 되므로, 성능 저하를 일으킴, 그러므로 앞 부분을 0으로 채워서 길이를 맞추는 Pre-Padding 방식을 사용
  - 반드시 Pre-Padding이 좋은게 아니라 Task나 모델의 구조에 따라 Post Padding도 사용
- 실제 예측 과정에서는 자유롭게 길이를 설정





Pre Padding

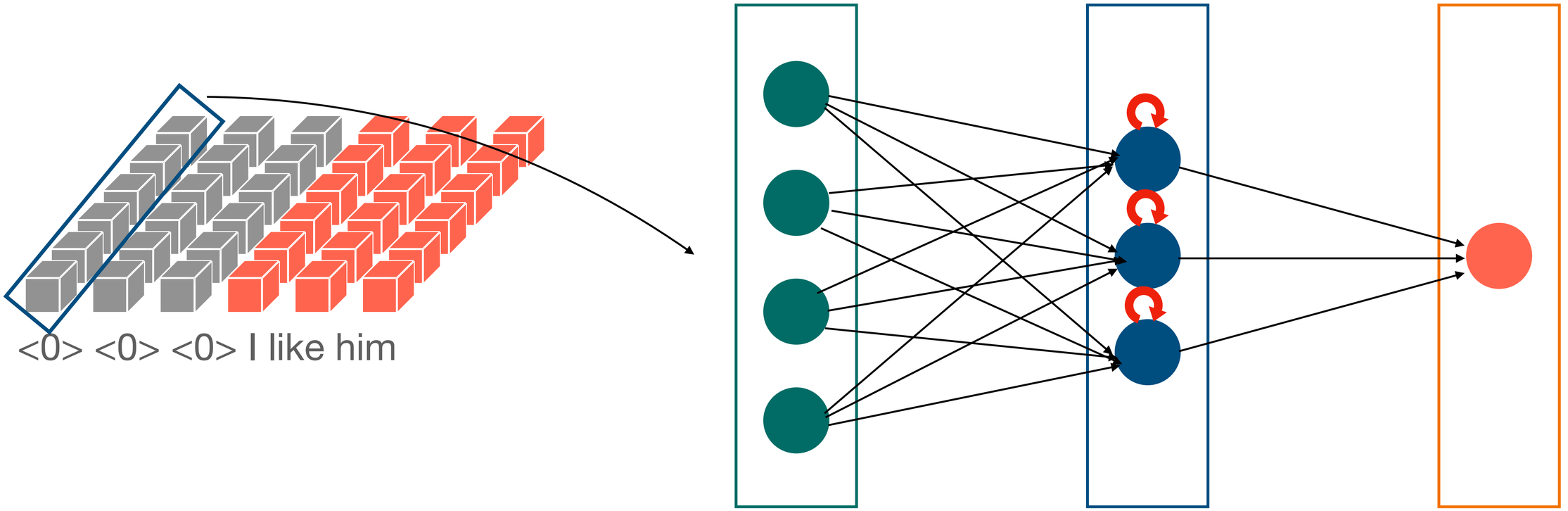
Table 1: LSTM pre-padding vs LSTM post-padding (%)

	LSTM-4 Pre-Padding	LSTM-4 Post-Padding
Train	80.072	49.977
Test	80.321	50.117
Epochs	9	6

Dwarampudi, Mahidhar, and N. V. Reddy. "Effects of padding on LSTMs and CNNs." *arXiv preprint arXiv:1903.07288* (2019).

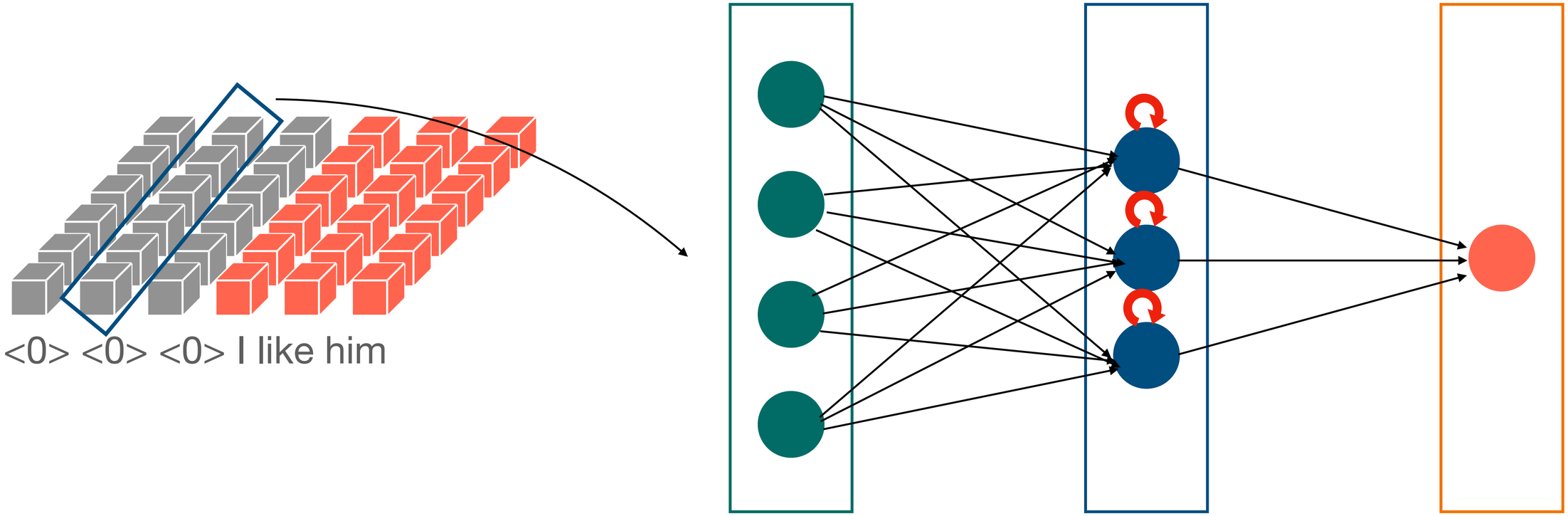
Recurrent Neural Networks

Training



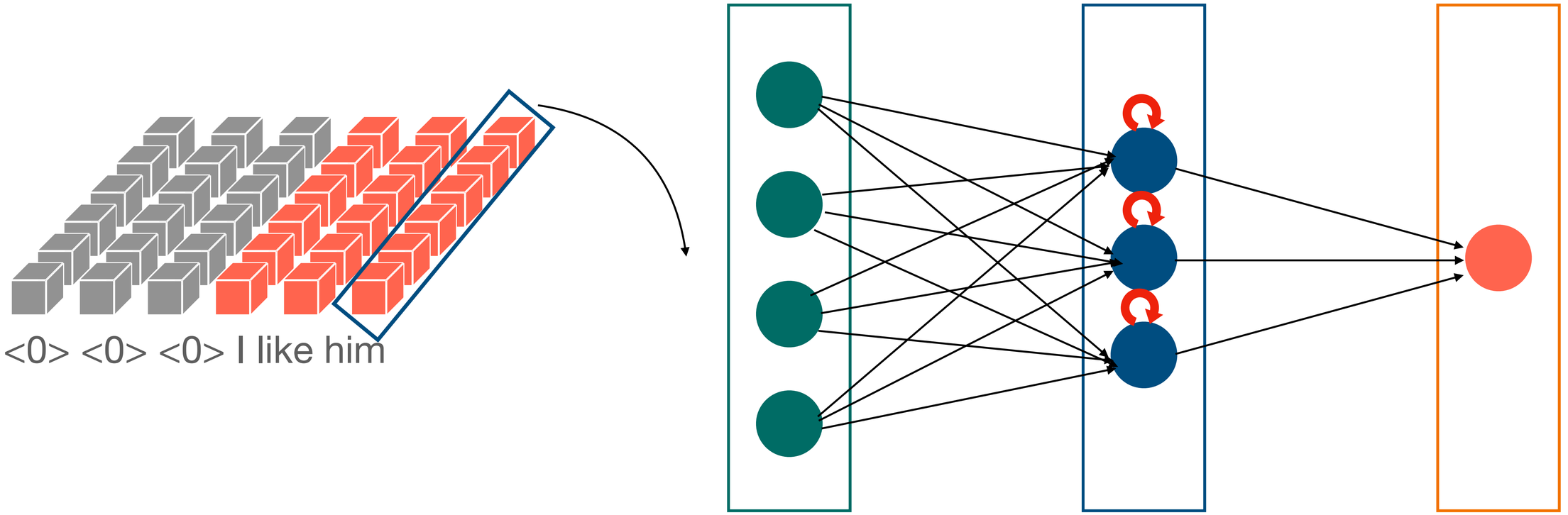
Recurrent Neural Networks

Training



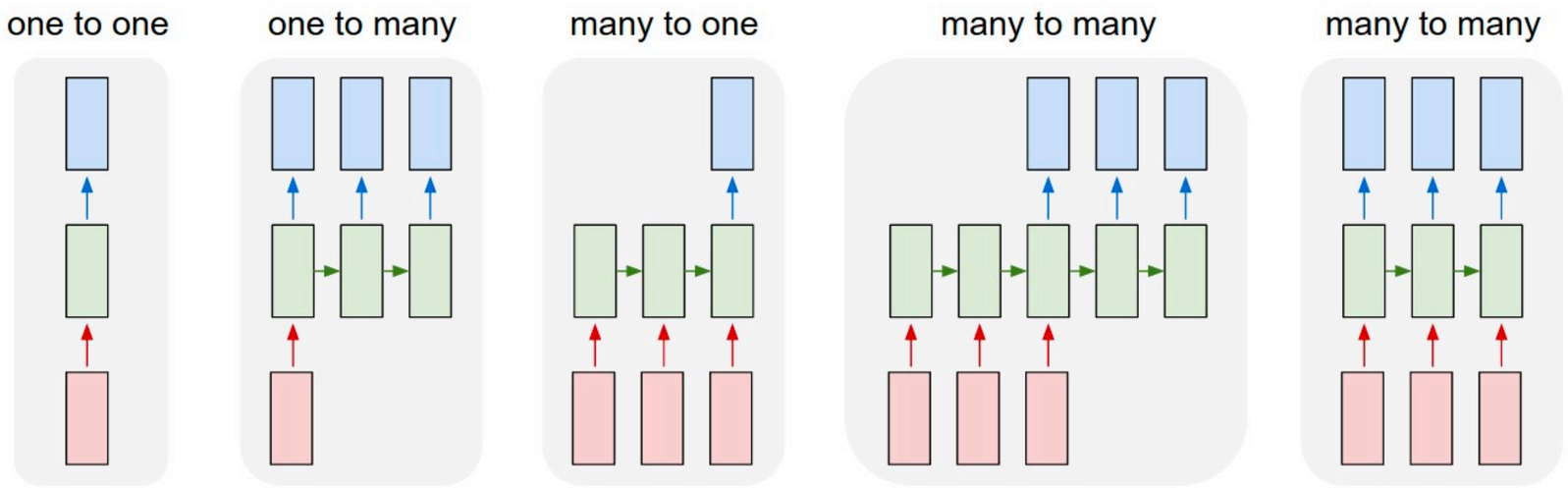
Recurrent Neural Networks

Training



Recurrent Neural Networks

Outputs

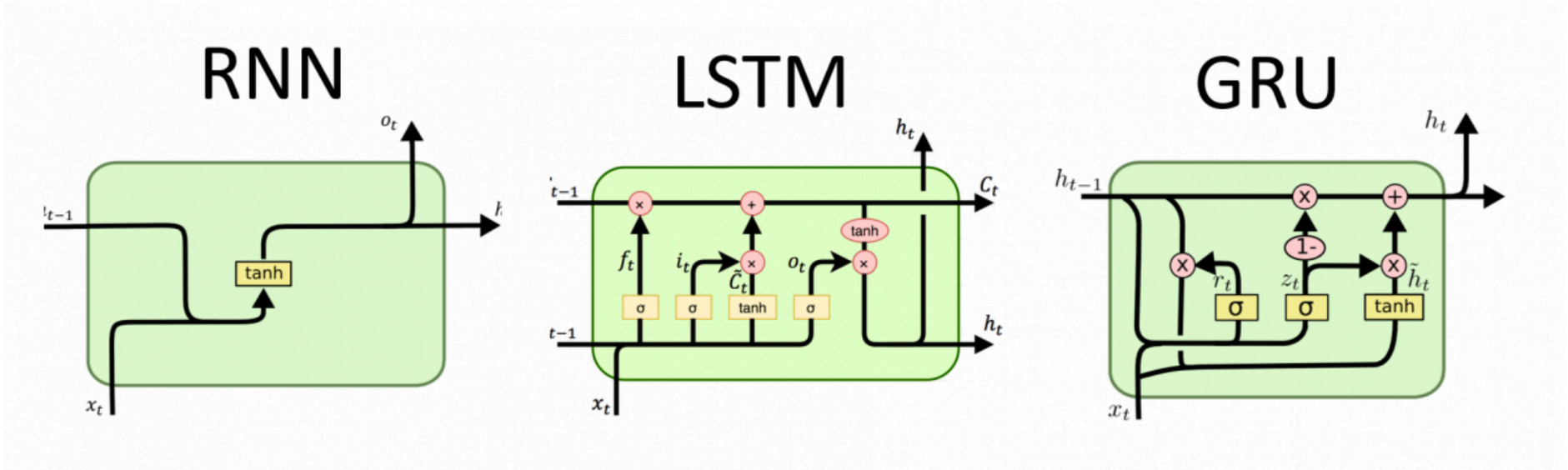



```python
num_epochs = 2
print_interval = 3000

for epoch in range(num_epochs):
    random.shuffle(train_dataset)
    for i, (name, label) in enumerate(train_dataset):
        hidden_state = model.init_hidden()
        for char in name:
            output, hidden_state = model(char, hidden_state)
        loss = criterion(output, label)

        optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), 1)
        optimizer.step()

        if (i + 1) % print_interval == 0:
            print(
                f"Epoch [{epoch + 1}/{num_epochs}], "
                f"Step [{i + 1}/{len(train_dataset)}], "
                f"Loss: {loss.item():.4f}"
            )
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
https://jaketae.github.io/study/pytorch-rnn/

Recurrent Neural Networks

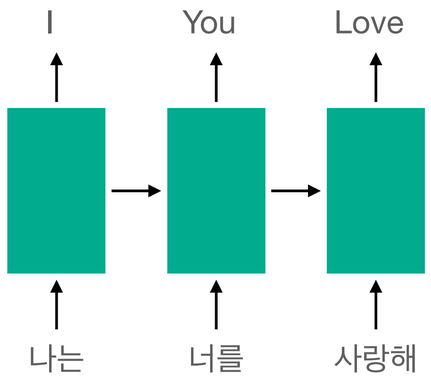
Variants



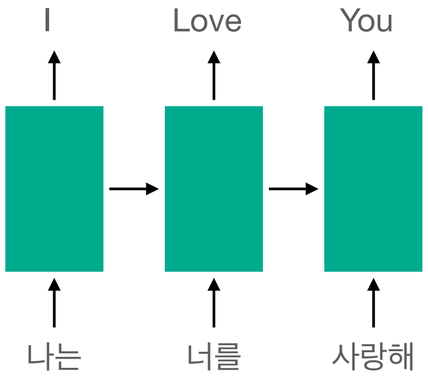
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Machine Translation

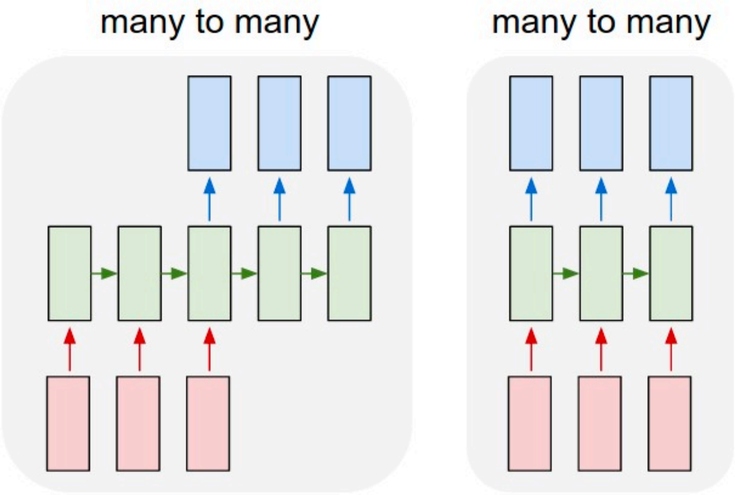
Sequence to Sequence



✕ 이전에 나온 Output을 고려하지 못함

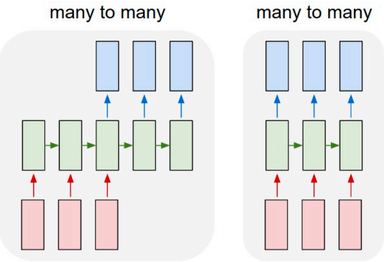


? 갑자기 ‘너를’이 ‘Love’가 된다고?

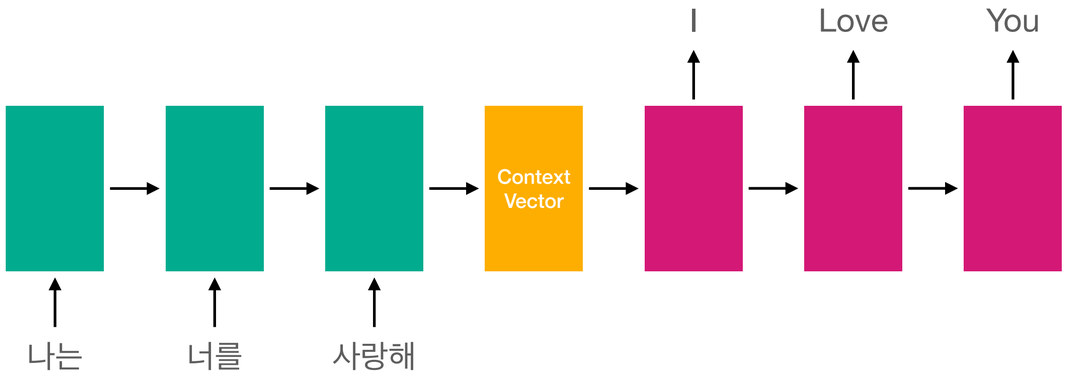


Machine Translation

Sequence to Sequence

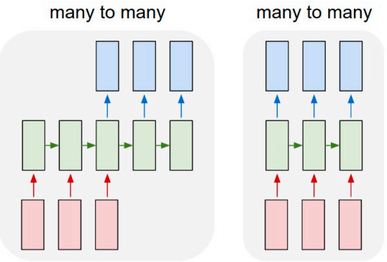


- 입력값을 다 받고, 전체 맥락을 고려해서, 출력값을 만들자!

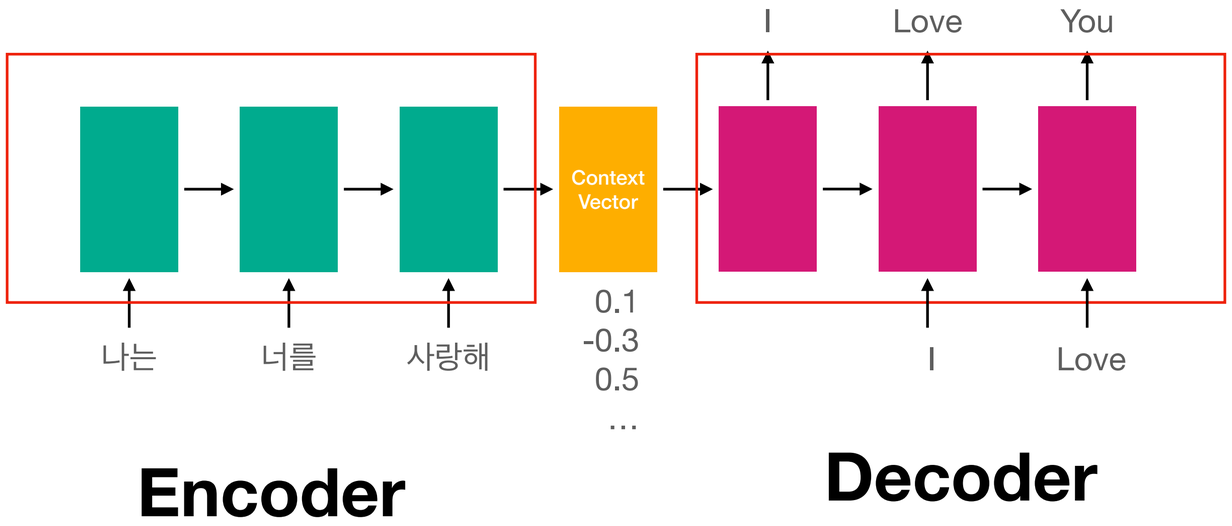


Machine Translation

Sequence to Sequence



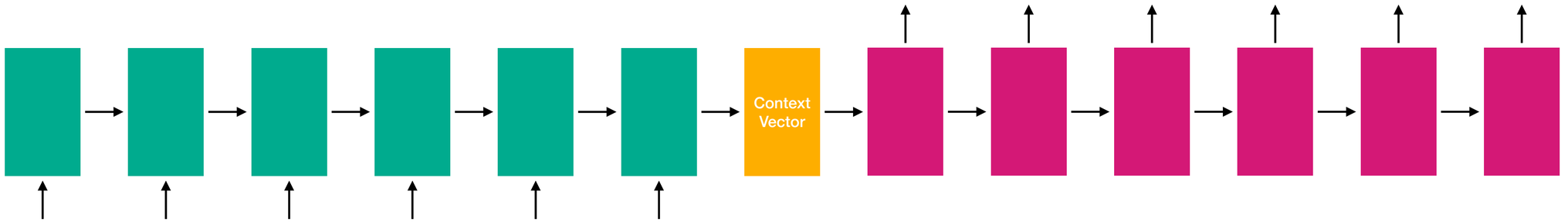
- Encoder: Context Vector를 만드는 부분
- Decoder: Context Vector를 풀어서 출력을 만드는 부분



Machine Translation

단순 RNN 기반의 Seq2Seq의 한계

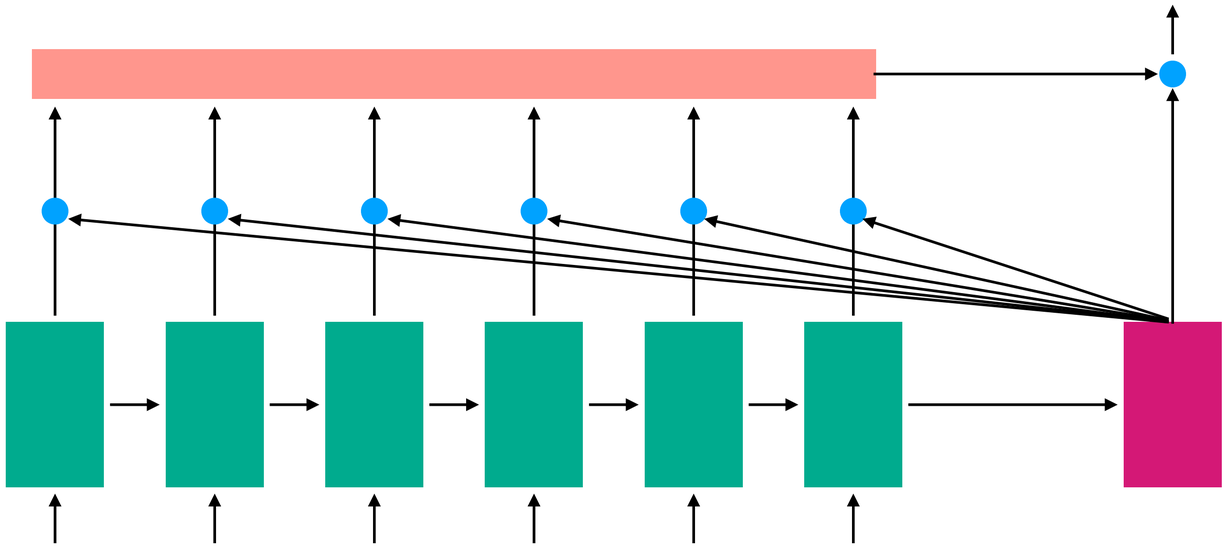
- 문장이 길어지면 성능이 떨어짐
  - 정보 손실: 문장 전체를 하나의 고정된 Vector로 담기 때문에
  - Gradient Vanishing: RNN의 고질적인 문제



Machine Translation

Seq2Seq + Attention

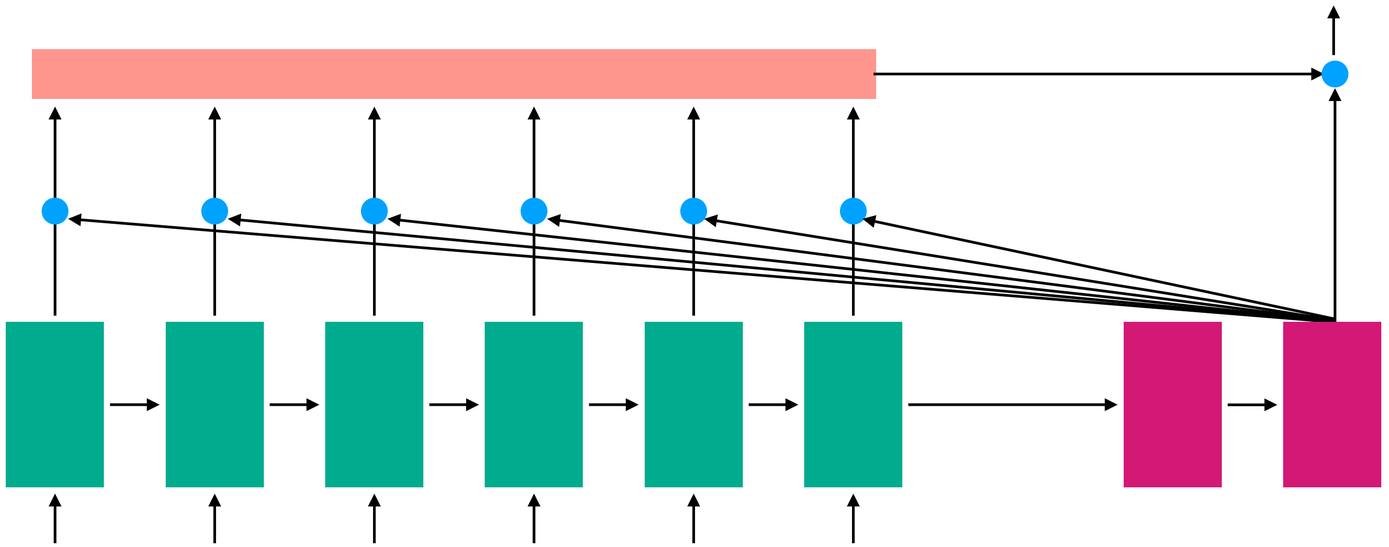
- 다음 단어를 출력할때, 이전 단계만 보지말고, 입력 전체를 보자



Machine Translation

Seq2Seq + Attention

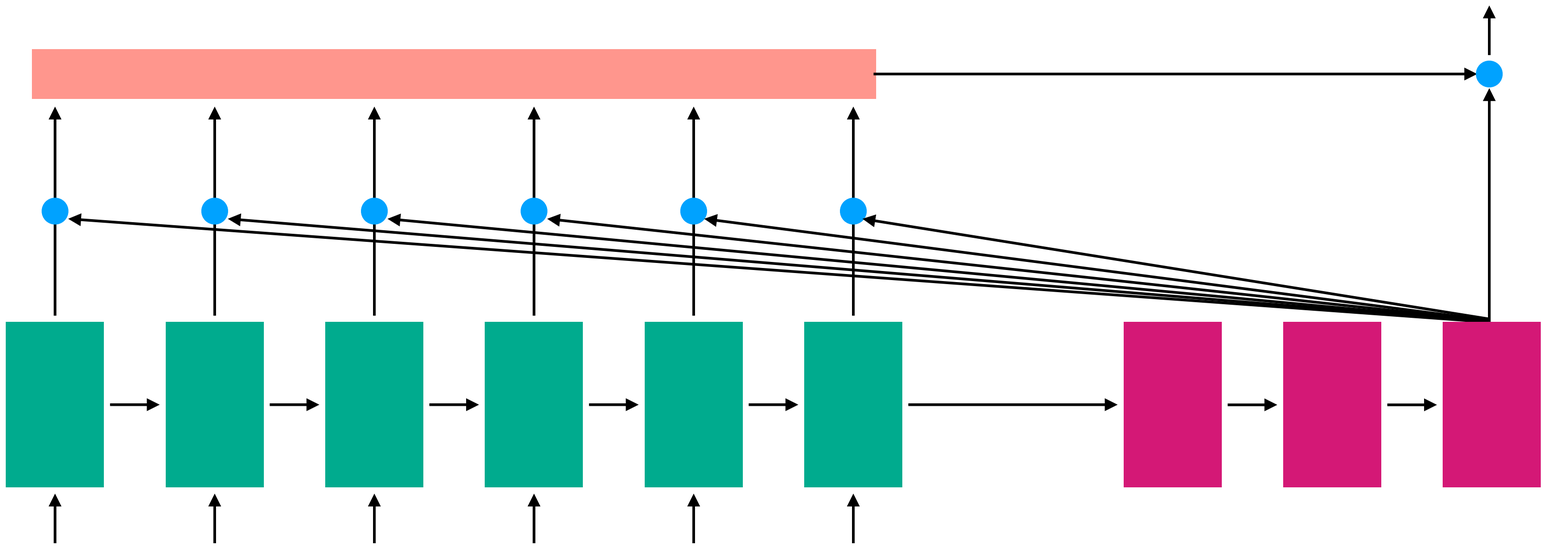
- 다음 단어를 출력할때, 이전 단계만 보지말고, 입력 전체를 보자



Machine Translation

Seq2Seq + Attention

- 다음 단어를 출력할때, 이전 단계만 보지말고, 입력 전체를 보자

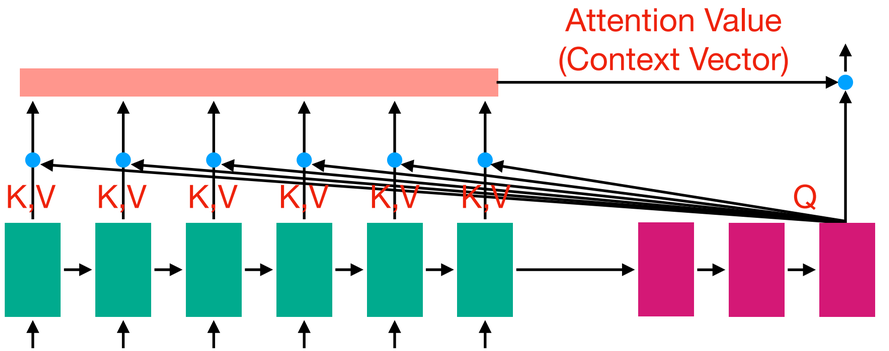


Machine Translation

Seq2Seq + Attention

- 다음 단어를 출력할때, 전체 입력 단어 중 가장 중요한 또는 관련있는 단어를 **집중**해서 고려하자

- Attention(Q,K,V) = Attention Value
  - Query: t 시점의 디코더 셀에서의 은닉(hidden) 상태
  - Keys: 모든 시점의 인코더 셀의 은닉 상태
  - Values: 모든 시점의 인코더 셀의 은닉 상태



Machine Translation

Seq2Seq + Attention

Attention output

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$$

↑ "source context for decoder step t"

↑ (weighted sum)

Attention weights

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \text{k} = 1..\text{m}$$

↑ "attention weight for source token k at decoder step t"

↑ (softmax)

Attention scores

$$\text{score}(h_t, s_k), \text{k} = 1..\text{m}$$

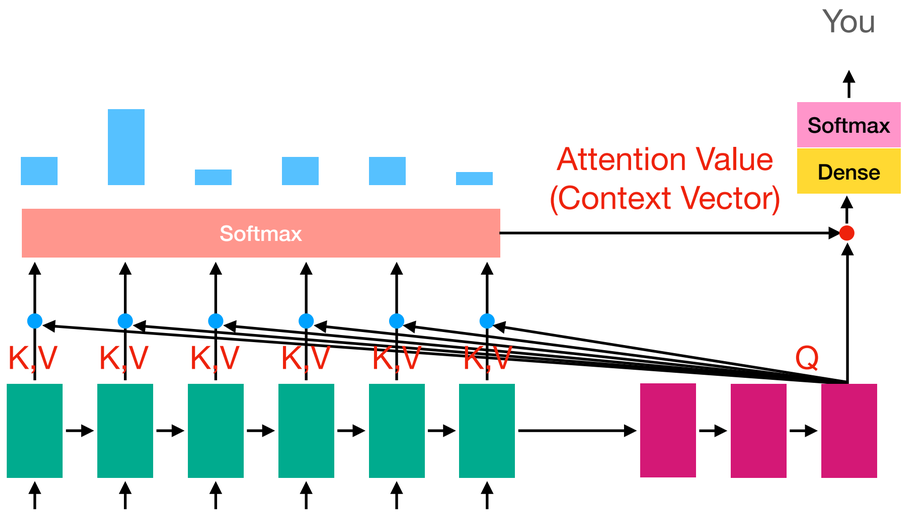
↑ "How relevant is source token k for target step t?"

↑

Attention input

$s_1, s_2, \ldots, s_m$ all encoder states

h_t one decoder state



https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Machine Translation
현재까지

- 단순 RNN으로 1:1 Mapping되는 Many-to-Many방식은 문맥을 고려하지 못한다
  - I love you => 나는 사랑해 너를
  - Context Vector 형태로 전체 입력을 저장해서 번역하는 식으로해결
- Context Vector 기반의 Many-to-Many는 문장의 길이와 상관없이 고정된 벡터 공간으로 표현하기 때문에 정보 손실이 크다.
  - Attention Mechanism으로 입력 전체를 바라보며 상관관계가 높은 단어들 위주로 집중(Attention) 한다.
  - 기존의 Context Vector를 Attention Value로 대체한다.
- 그런데 RNN이 꼭 필요한가..?

Machine Translation

Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

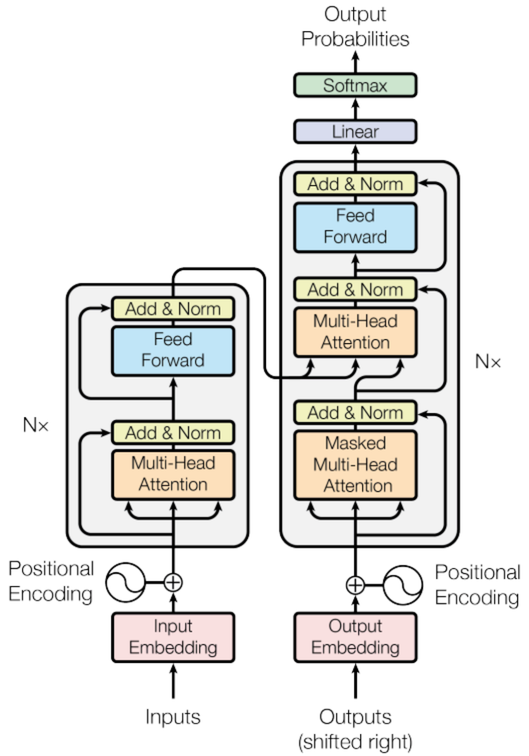
1 Introduction

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Machine Translation

Transformer

- Seq2Seq와 마찬가지로 Encoder-Decoder구조
- No RNN
  - 그러면 단어의 위치와 순서는 어떻게 알 수 있나?
  - 이것 때문에 RNN을 썼던 건데?
- Positional Encoding!



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

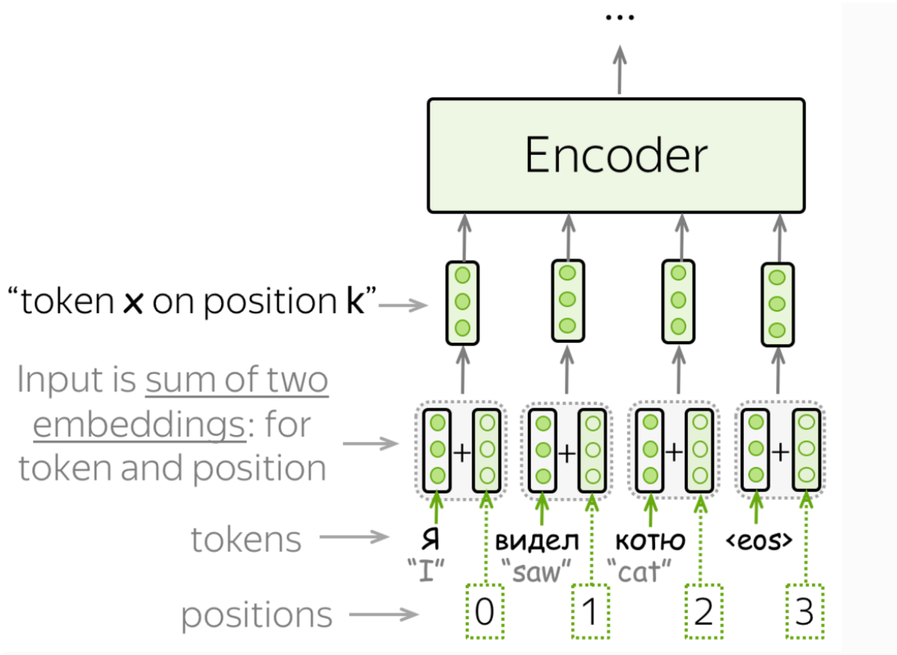
Transformer

Positional Encoding

- 각각의 단어의 위치를 알려주는 역할
- 단어에 대한 Embedding과 Position에 대한 Embedding을 결합

$$\mathrm{PE}_{pos,2i} = \sin(pos/10000^{2i/d_{model}}),$$

$$\mathrm{PE}_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}}),$$



Transformer
Positional Encoding

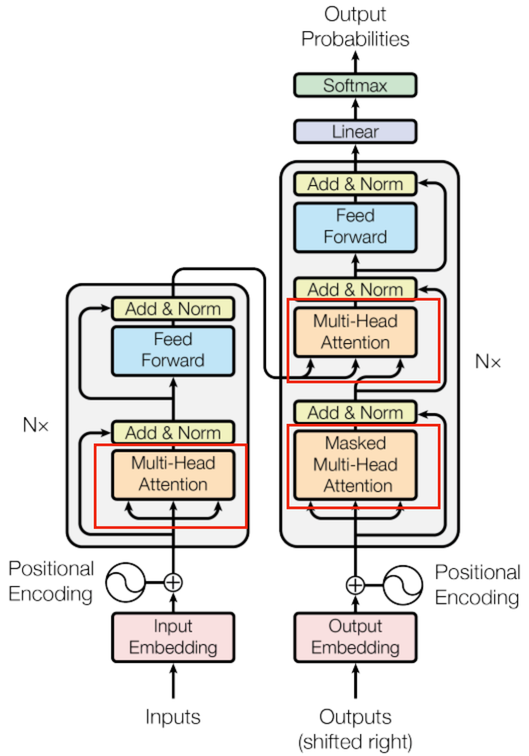
- RNN이 없다?
  - 이전 단어가 네트워크를 통과하기까지 기다릴 필요가 X
  - 병렬 연산 가능

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformer

Attention

- Attention in Encoder
- Attention with Encoder+Decoder
- Masked Attention in Decoder

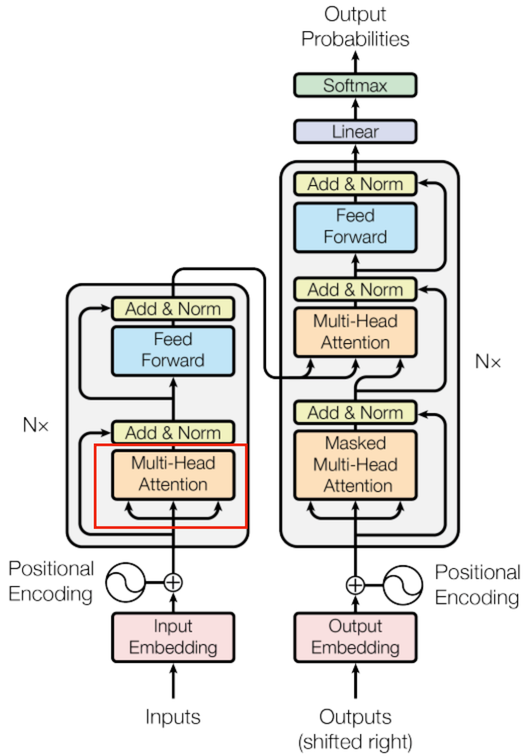


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformer

Attention in Encoder

- 기존 Seq2Seq의 Q,K,V
  - Query: t 시점의 디코더 셀에서의 은닉(hidden) 상태
  - Keys: 모든 시점의 인코더 셀의 은닉 상태
  - Values: 모든 시점의 인코더 셀의 은닉 상태
- Self Attention
  - Query: 모든 입력 문장의 단어 벡터
  - Key: 모든 입력 문장의 단어 벡터
  - Value: 모든 입력 문장의 단어 벡터

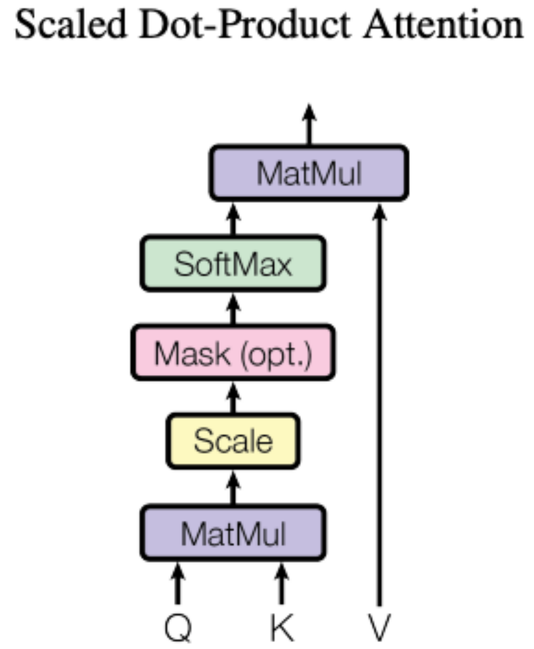


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformer

Self-Attention

- Self Attention
  - Query: 모든 입력 문장의 단어 벡터
  - Key: 모든 입력 문장의 단어 벡터
  - Value: 모든 입력 문장의 단어 벡터

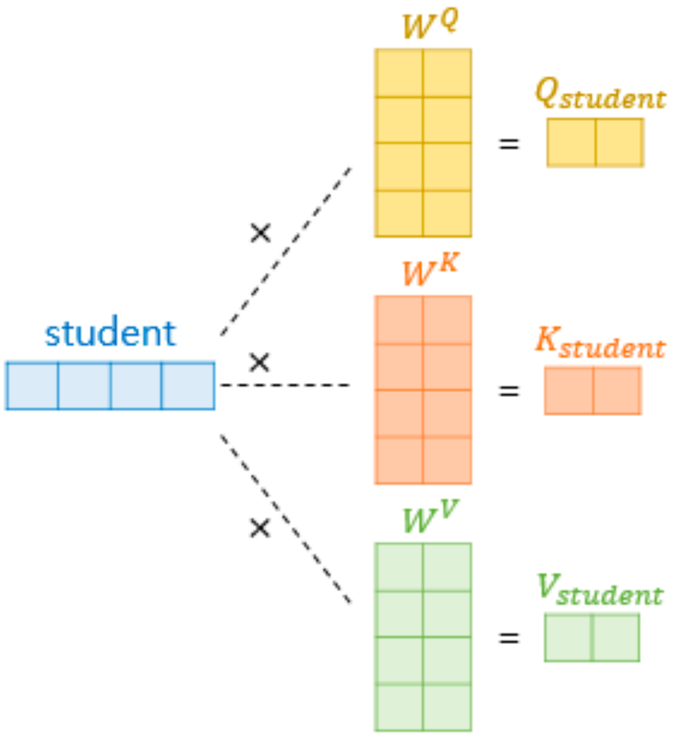


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

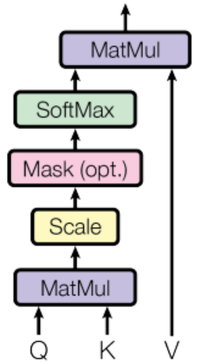
Transformer

Self-Attention

- 단어 벡터를 Q,K,V로 추출



Scaled Dot-Product Attention

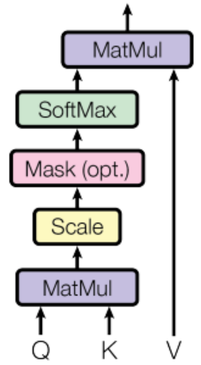


https://wikidocs.net/31379

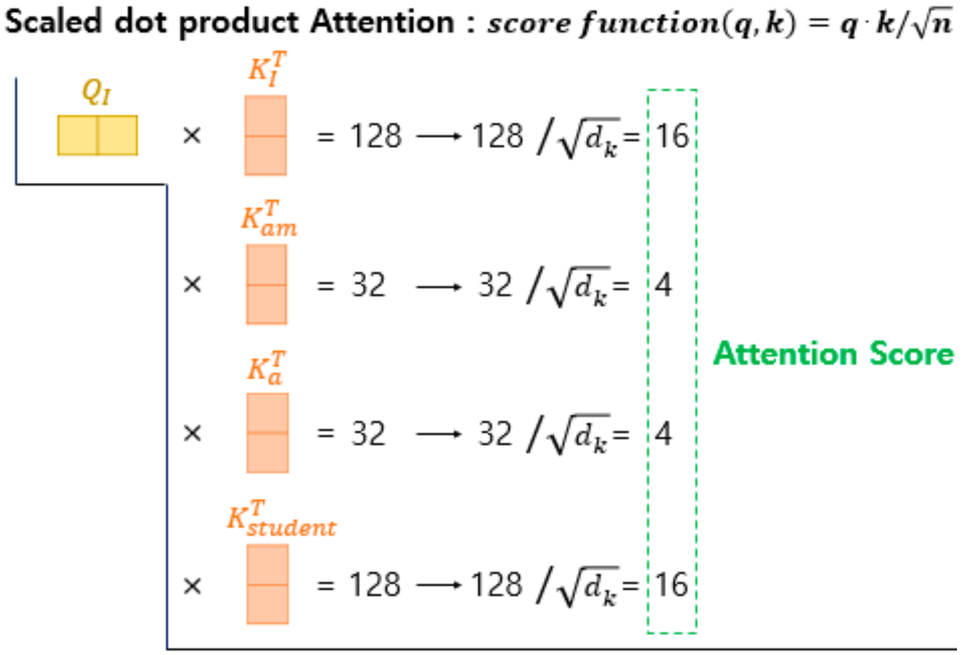
Transformer

Self-Attention

Scaled Dot-Product Attention



- 자기 자신을 포함한 나머지 단어 벡터들과 모두 관련성 계산

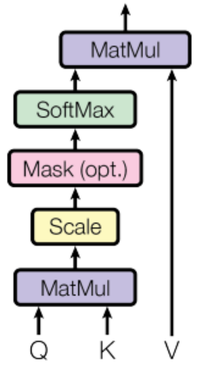


https://wikidocs.net/31379

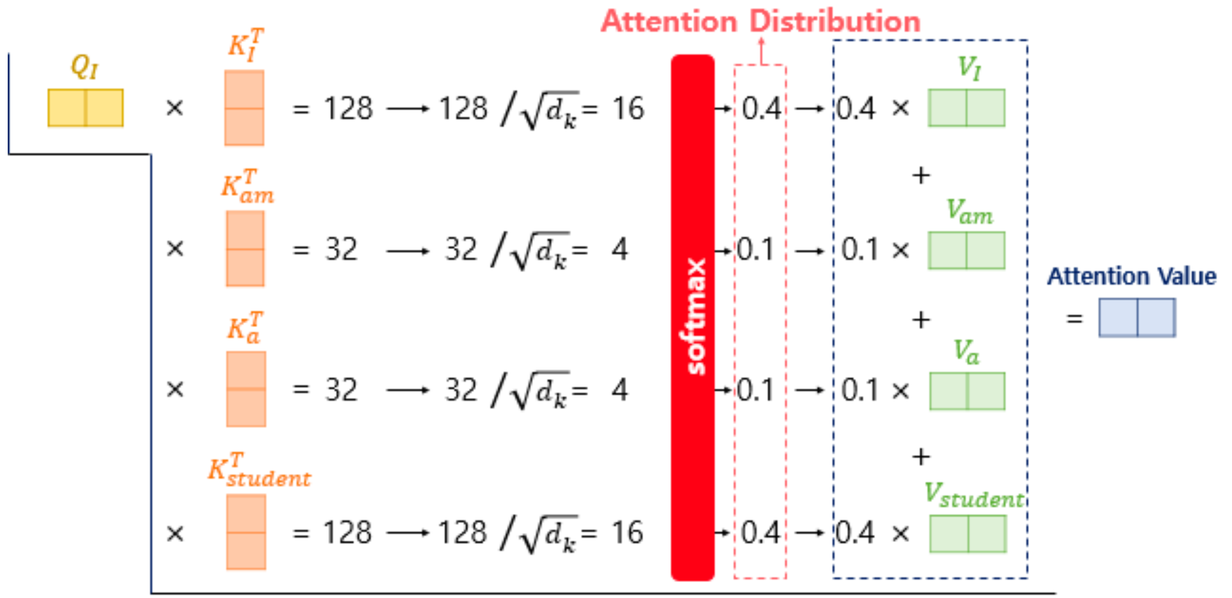
Transformer

Self-Attention

Scaled Dot-Product Attention



- 관련성 점수를 Softmax를 통하여 분포로 변환 (합 = 1)
- 가중치 합을 통하여 최종 Attention Value생성 (단어 I에 대한 Context Vector)

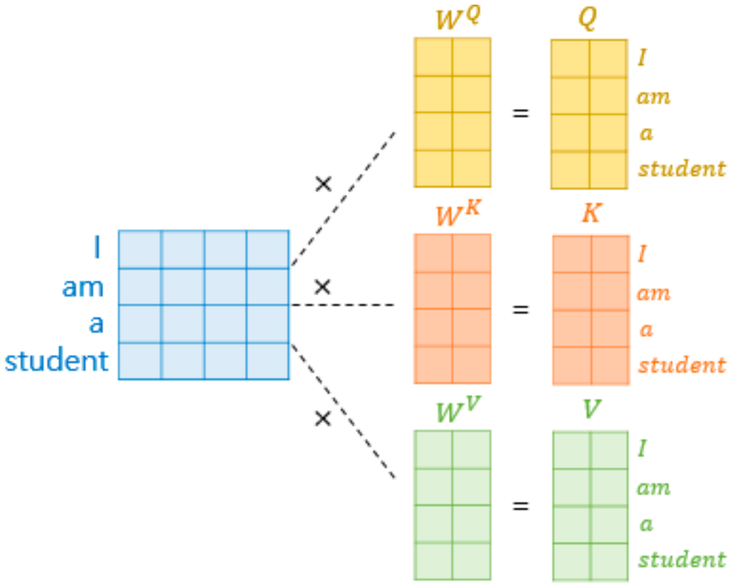


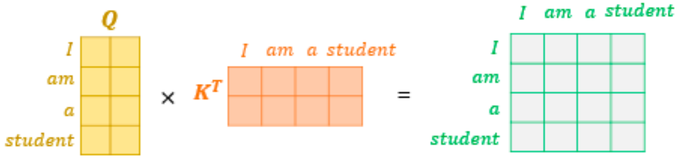
https://wikidocs.net/31379

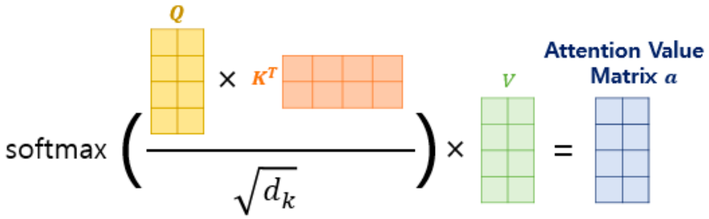
Transformer

Self-Attention

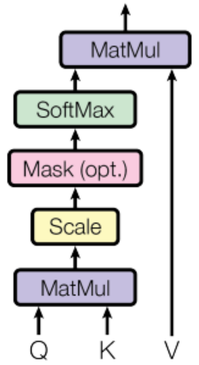
- 모든 입력 단어를 한번에 처리하기







Scaled Dot-Product Attention

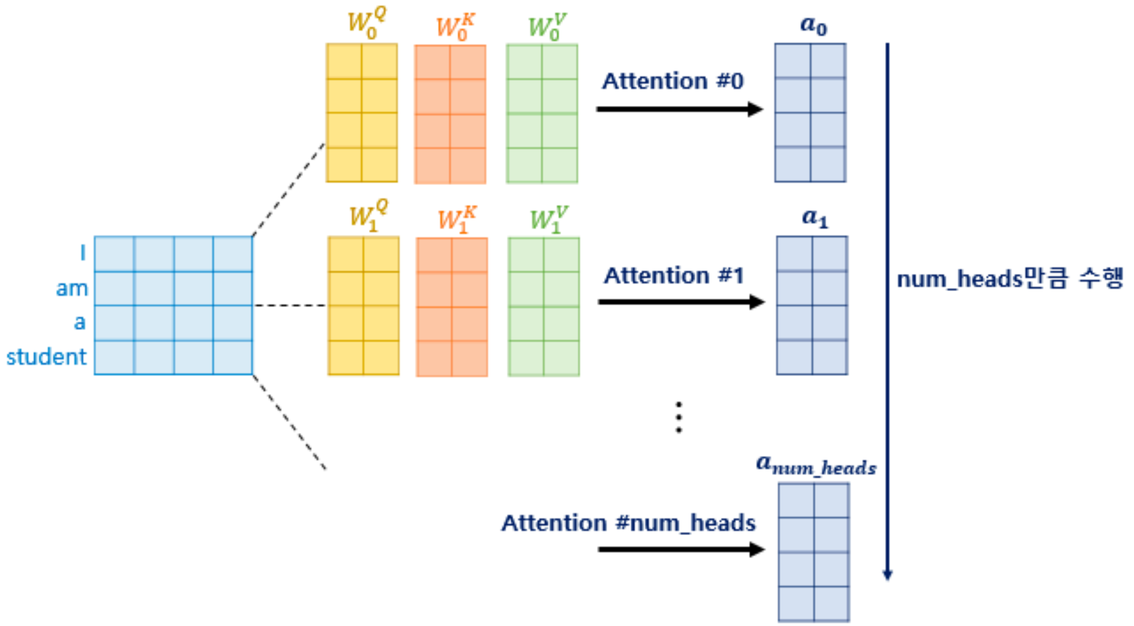


https://wikidocs.net/31379

Transformer

Multi-Head Attention

- 다양한 관점으로 집중해보자
- 나는 오늘 강의자료를 만들어서 피곤했어
  - Attention 0: 나는 ——피곤
  - Attention 1: 강의자료 —- 피곤
  - Attention 2: 오늘 —- 피곤
  - ..
  - ..

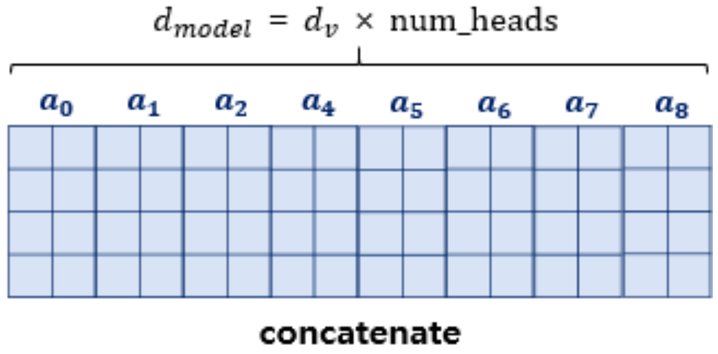


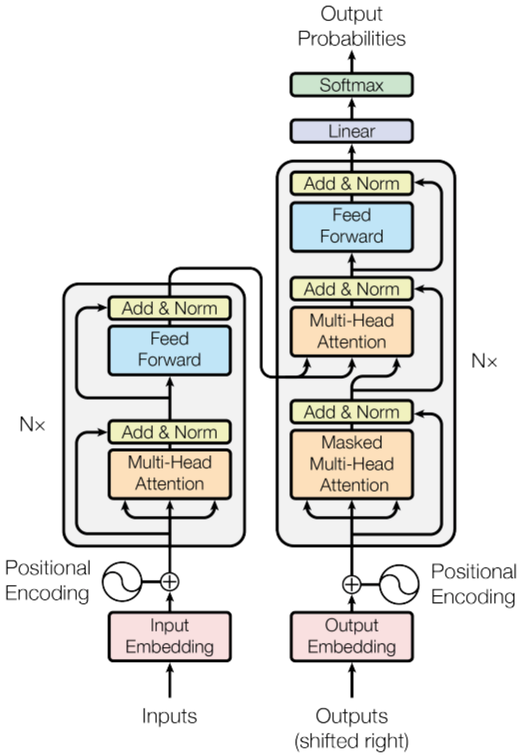
https://wikidocs.net/31379

Transformer

Multi-Head Attention

- 이렇게 최종적으로 나온 Attention의 결과물을 FFN을 통해 학습
  - Attention: 주변의 단어들을 보면서, 정보들을 모으자
  - FFN: 이렇게 정보가 모였으면, 그걸 종합적으로 고려해서 **생각**을 해보자



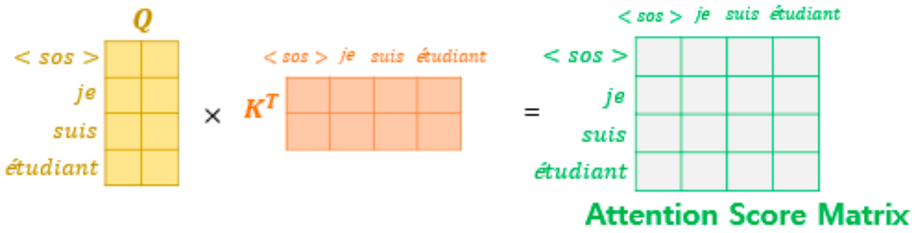


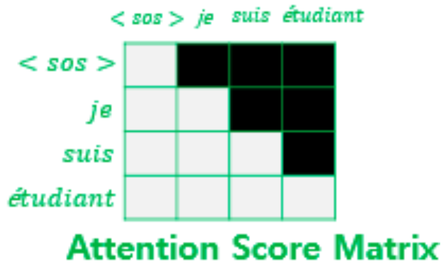
https://wikidocs.net/31379

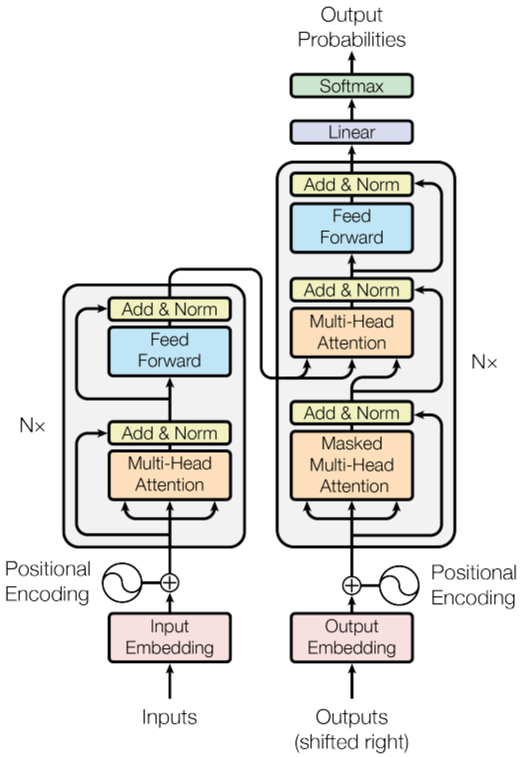
Transformer

Masked Multi-Head Attention

- Decoder에서 이제 정답을 맞춰야 할때는, 미래의 단어는 보지말자





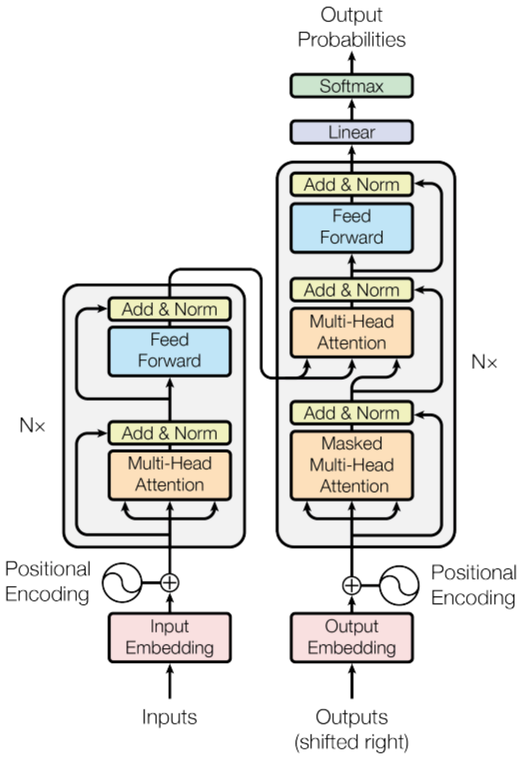


https://wikidocs.net/31379

Transformer

Multi-Head Attention with Encoder and Decoder

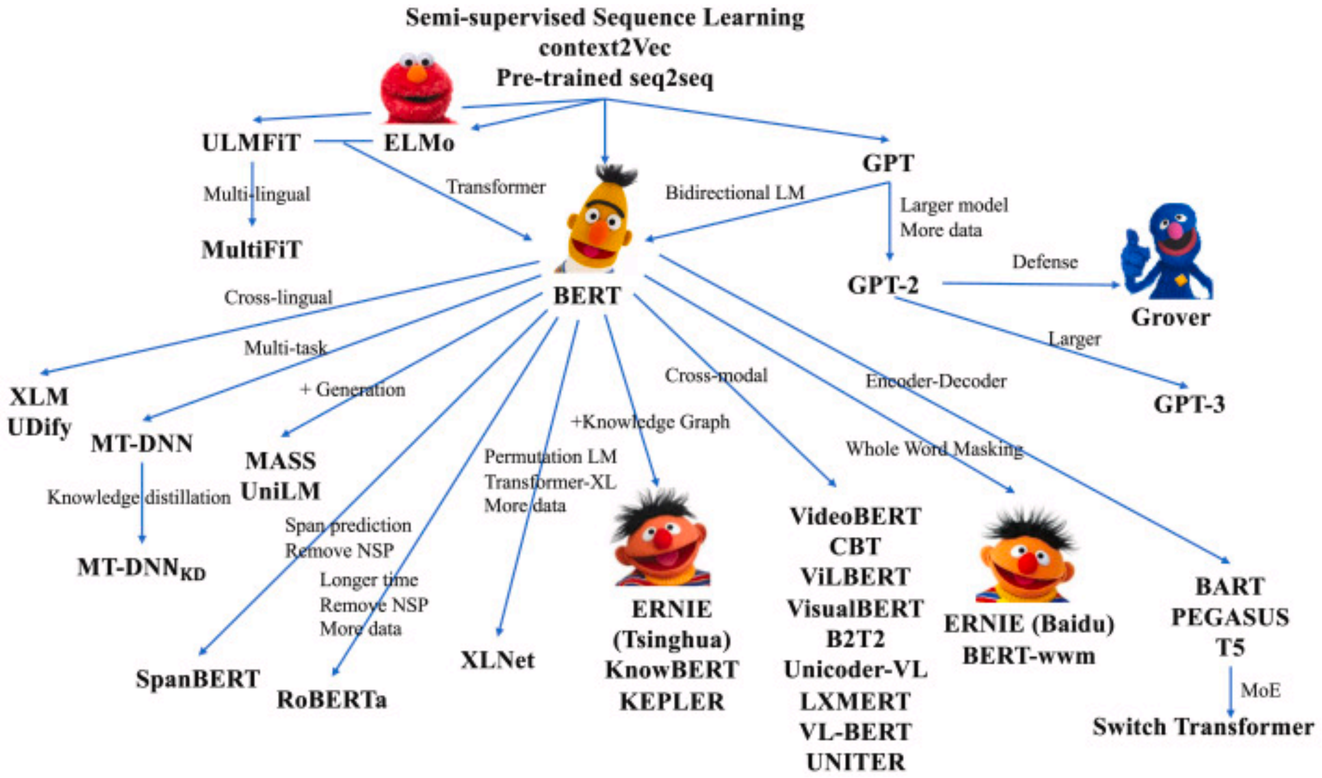
- Encoder에서 최종적으로 나온 정보들을 K와 V로 보고 Decoder의 Masked MHA에서 나온 정보를 Q로 보자
  - K와 V는 단어간의 관계를 잘 이해하고, FFN을 통해 종합적으로 고려가 된 정보
  - Q는 이전에 나온 단어와 지금 단어간의 관계를 알고있는 정보
- 그리고 마지막으로 FFN을 한번 더 거쳐서 종합적으로 본것을 또 종합적으로 보자!



https://wikidocs.net/31379

Transformer

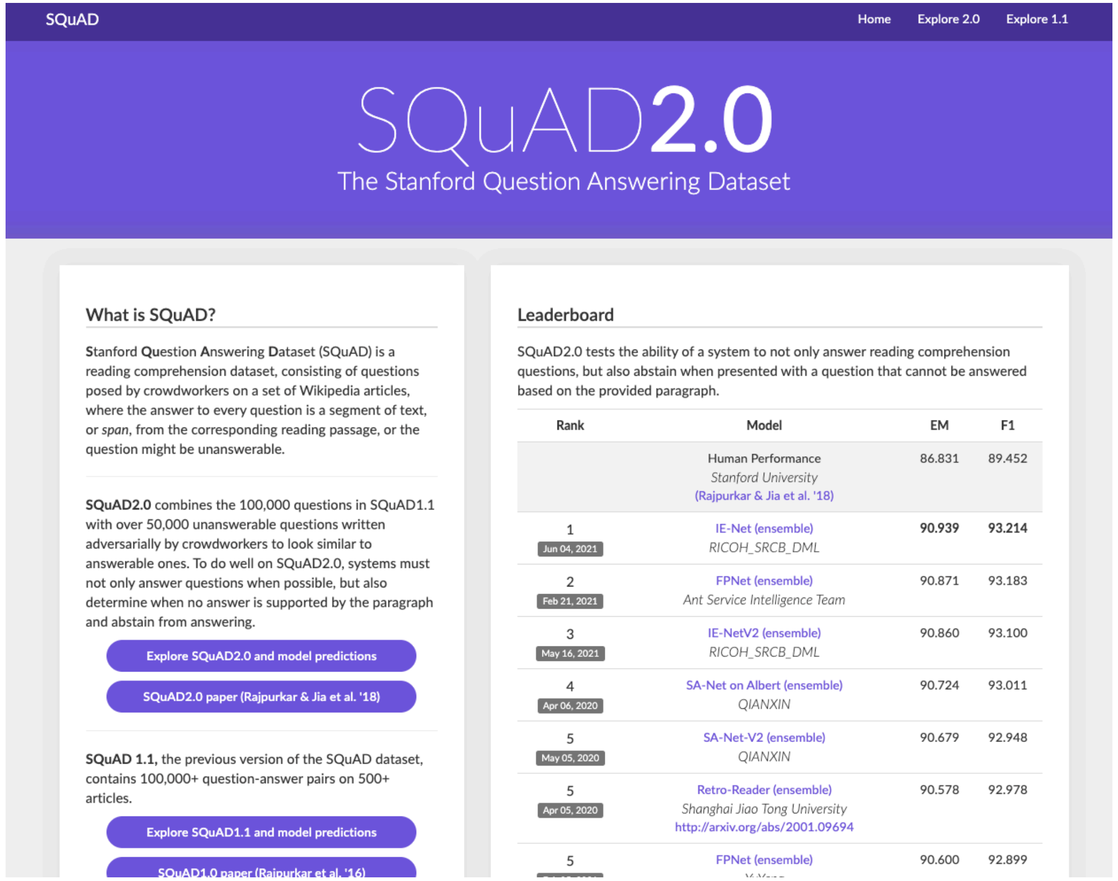
Transformer가 쏘아올린 작은(?) 공



Han, Xu, et al. "Pre-trained models: Past, present and future." *AI Open* 2 (2021): 225-250.

Transformer

Transformer가 쏘아올린 작은(?) 공



https://rajpurkar.github.io/SQuAD-explorer/

E.O.D