NLP with RNNs & Attention, Transformer

HOML – chapter16

TAVE Research DL001

Changdae Oh

2021. 02. 07

Topics

1. Text Generation & Sentimental Analysis
2. Encoder-Decoder Networks for NMT
3. Attention
4. Transformer

Topics

1. Text Generation & Sentimental Analysis
2. Encoder-Decoder Networks for NMT
3. Attention
4. Transformer

Text Generation & Sentimental Analysis

1) Text Generation with Char-RNN

1. Data load

2. Tokenization
   - 텍스트에 사용된 모든 글자(알파벳, 특수문자, 숫자 등)를 찾아 각기 다른 정수 ID에 mapping

3. Data split & slice
   - 시계열 데이터 분할 시에는 보통 정상성(stationary) 가정 후 시간에 따라 train/valid/test를 분할
   - slicing : 훈련셋의 시퀀스가 너무 긴 경우, 훈련이 실현 가능하도록 **텍스트 윈도우들**로 쪼갬 (Truncated BPTT)

4. Encoding
   - 각 ID에 1:1로 대응되는 벡터를 표현을 생성 (one-hot vs embedding)
   - 이 예제에서는 고유한 문자 개수가 39개로 많지 않아 원-핫 인코딩 사용

5. Model build & train

6. Text generate

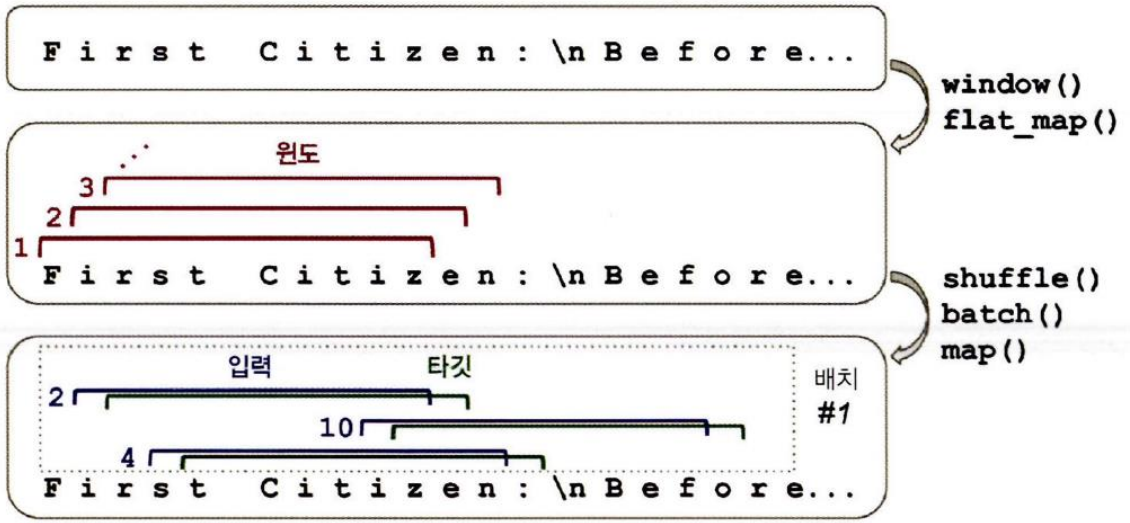
[실습코드]

Text Generation & Sentimental Analysis

1) Text Generation with Char-RNN

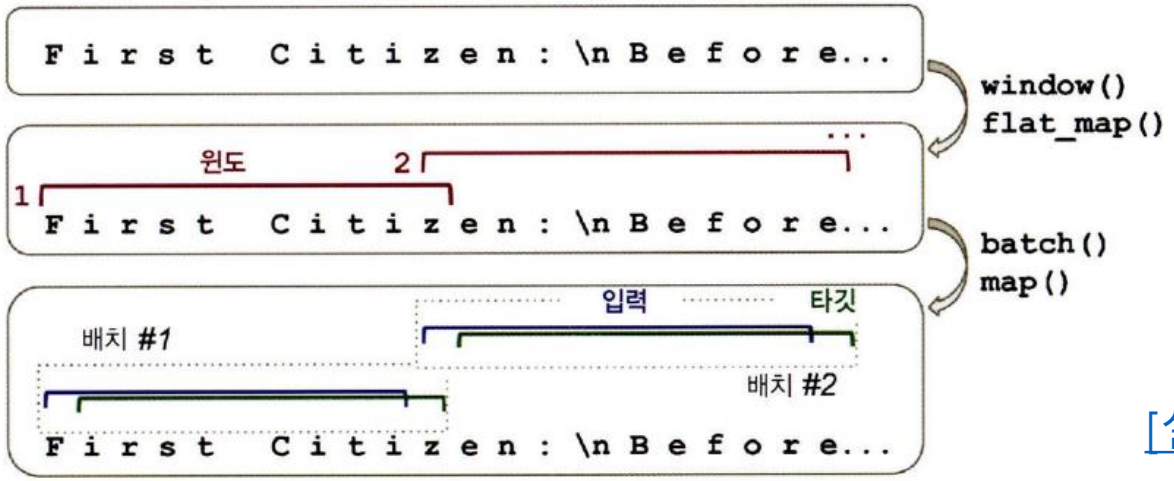
Stateless RNN

: 각 배치의 학습 후 hidden state를 0으로 초기화



Stateful RNN

: 한 배치 학습 후 마지막 state를 다음 훈련 배치의 초기상태로 설정. 1 epoch 종료 시에 state를 0으로 초기화



[실습코드]

Text Generation & Sentimental Analysis

2) Sentimental Analysis

1. Data load

2. Preprocessing
   - 단순히 공백을 기준으로 토큰화 하기보단 subword 수준으로 토큰화 하는 것이 일반적으로 좋음.
   - byte pair encoding, wordpiece 등의 전략도 확인해볼 것.
   - 정규표현식(regular expression)을 이용한 전처리

3. Encoding
   - 단어 인코딩을 위해 우선 모든 단어를 포함할 수 있는 어휘사전을 구축하고 그로부터 Lookup Table을 구축해야 함.
   - 이때 경우에 따라 메모리 효율성, 성능향상을 위해 많이 등장하는 단어순으로 단어사전을 truncate 할 수 있음.
   - OOV bucket 사이즈를 설정하고 룩업 테이블을 만든다. 이제 각 단어들을 고유한 정수 ID로 mapping할 수 있다.

4. Model build & train
   - Padding / Masking

5. Inference

[실습코드]

Text Generation & Sentimental Analysis

2) Sentimental Analysis

Reusing Pre-trained word embeddings

Word embedding을 우리의 task specific dataset에서 직접 훈련시키기보다,
대용량의 general한 corpus에 대해 사전학습된 임베딩을 가져와서 쓰는 것이 훨씬 효율적이며, 성능이 잘나온다.

```
!pip install --upgrade tensorflow_hub
import tensorflow_hub as hub
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)
train_size = info.splits["train"].num_examples
batch_size = 32
train_set = datasets["train"].repeat().batch(batch_size).prefetch(1)


model = keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/tf2-preview/nnlm-en-dim50/1",
                   dtype=tf.string, input_shape=[], output_shape=[50]),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")
])
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train_set, steps_per_epoch=train_size // batch_size, epochs=5)
```

- ✓ 가져와서 임베딩 층으로써 사용하고 그 위에 task specific한 layer들을 추가 *!!*
- ✓ 임베딩을 task에 맞게 Fine-tuning 할 수도 있음. (선택)

[실습코드]

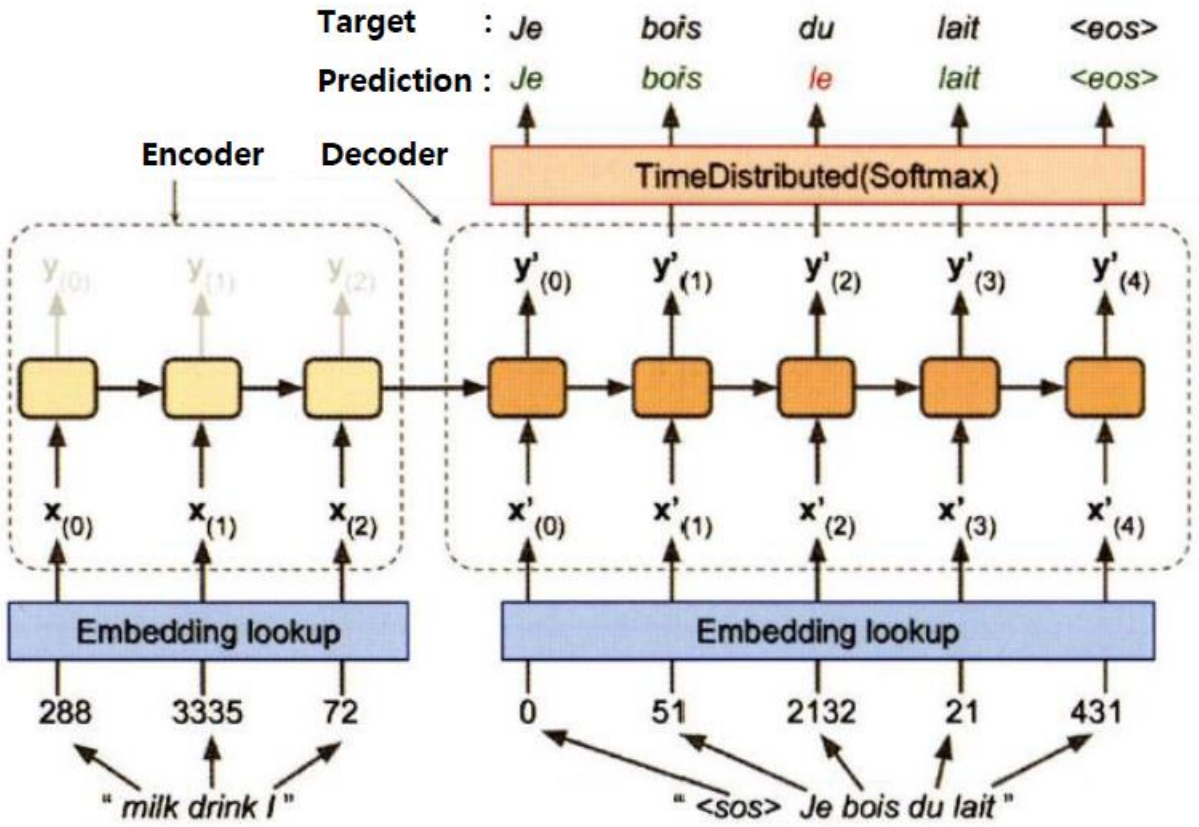
Topics

1. Text Generation & Sentimental Analysis
2. Encoder-Decoder Networks for NMT
3. Attention
4. Transformer

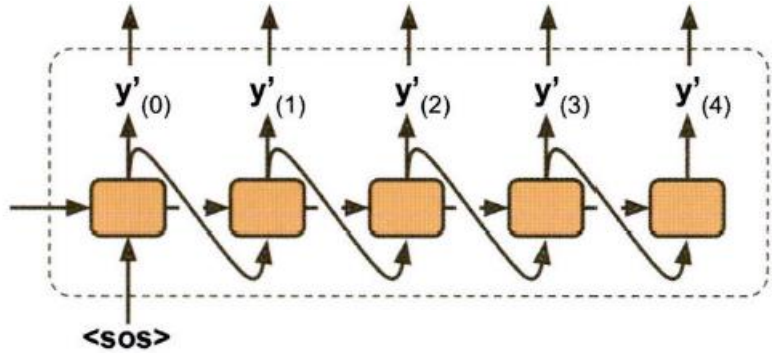
Encoder-Decoder Networks for NMT

Architecture

Training phase



Inference phase

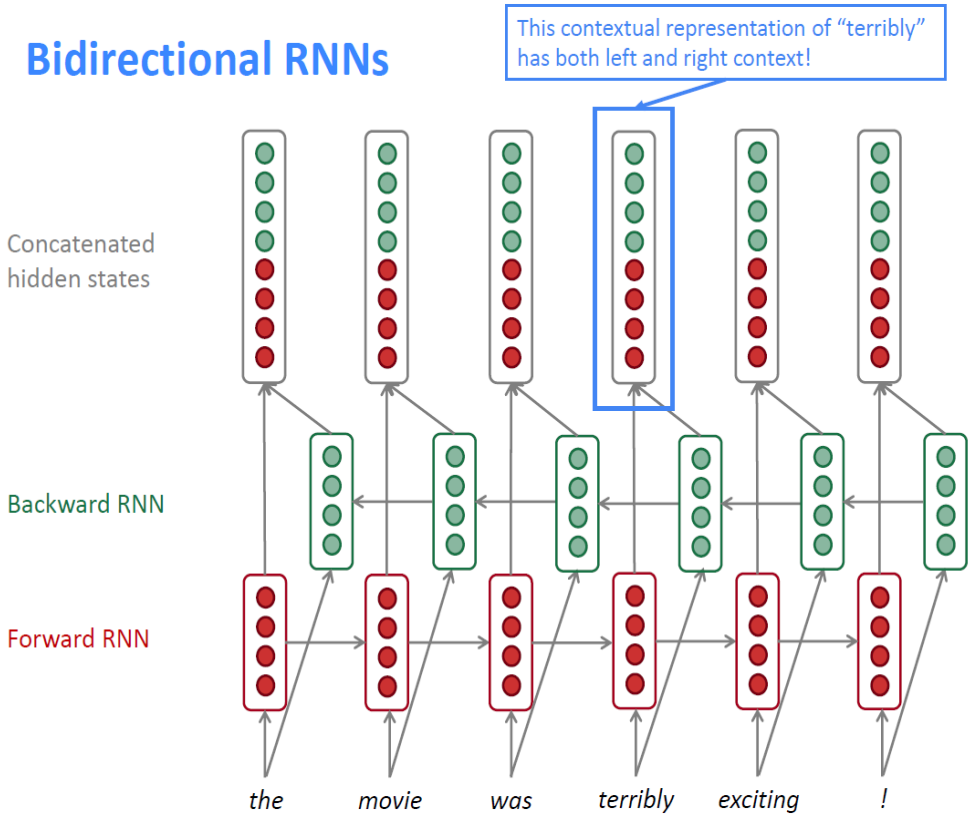


- ✓ Reverse Sequence?
- ✓ bucketing, padding, masking
- ✓ ignore Tokens after <EOS>
- ✓ Sampled softmax

Encoder–Decoder Networks for NMT

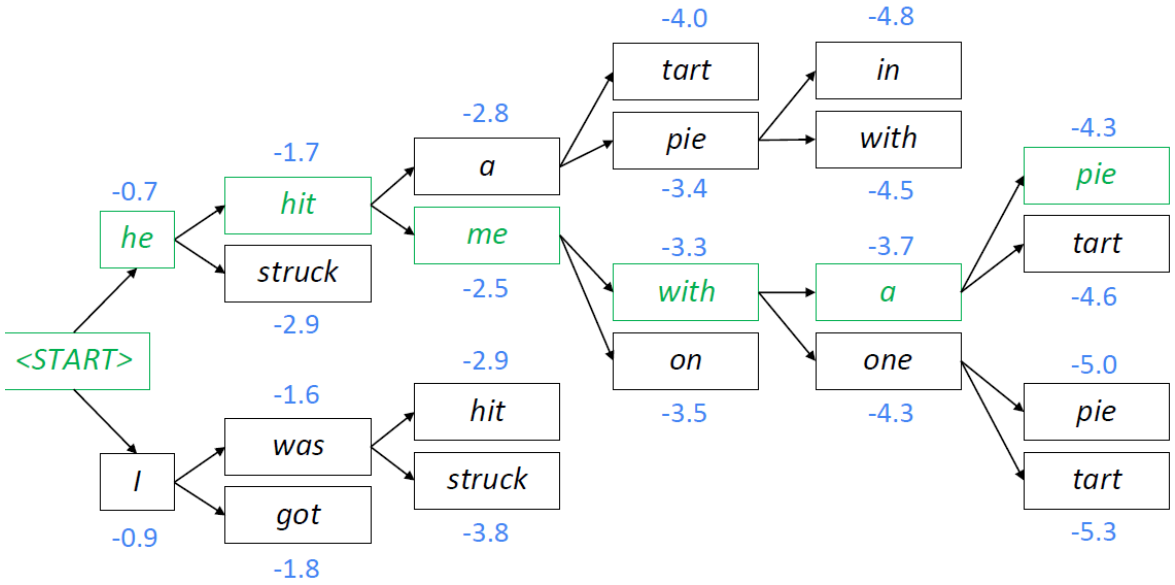
Improvement

Bidirectional RNNs



Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



+ Peeky Decoder

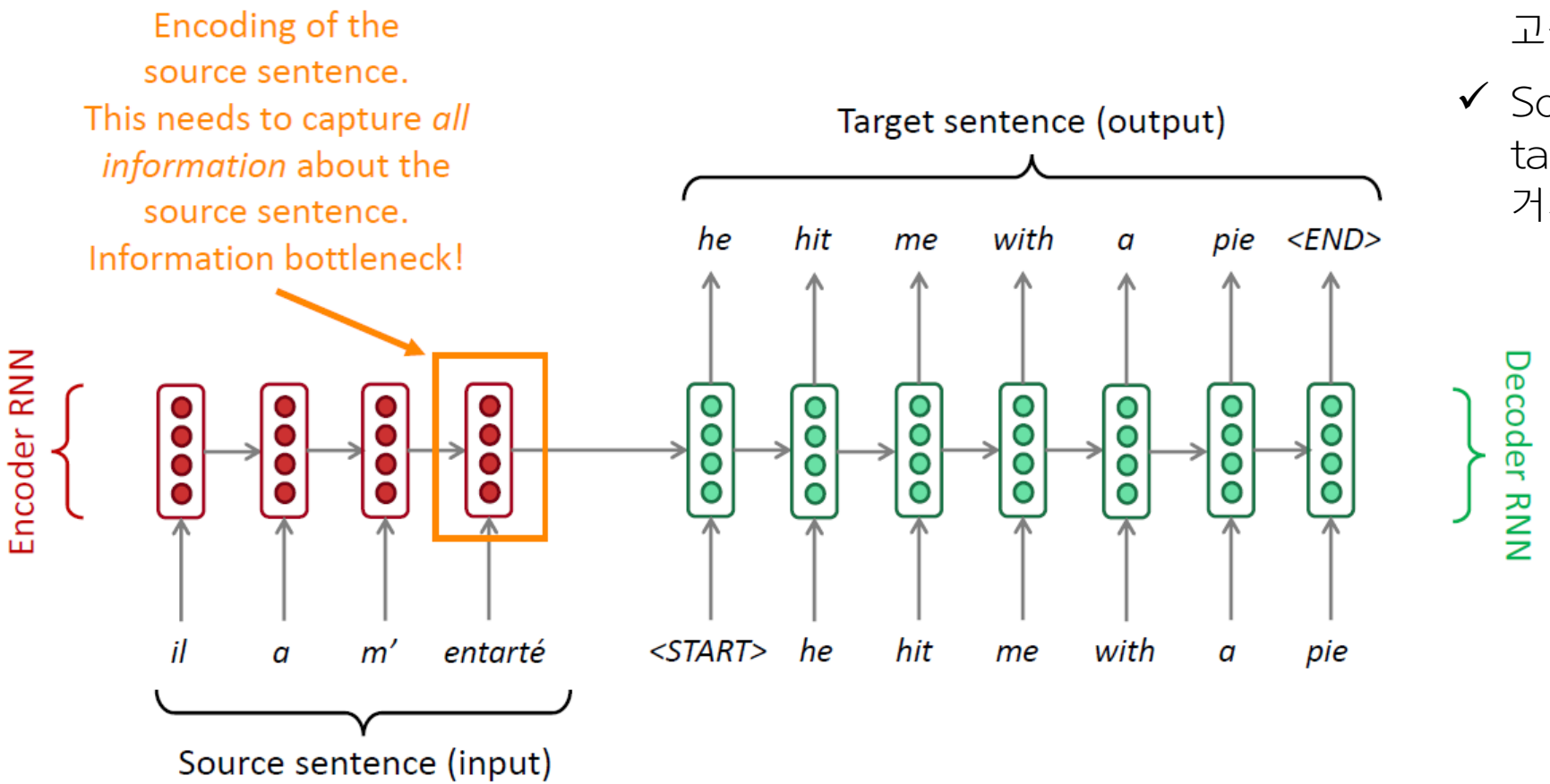
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/

Topics

1. Text Generation & Sentimental Analysis
2. Encoder-Decoder Networks for NMT
3. Attention
4. Transformer

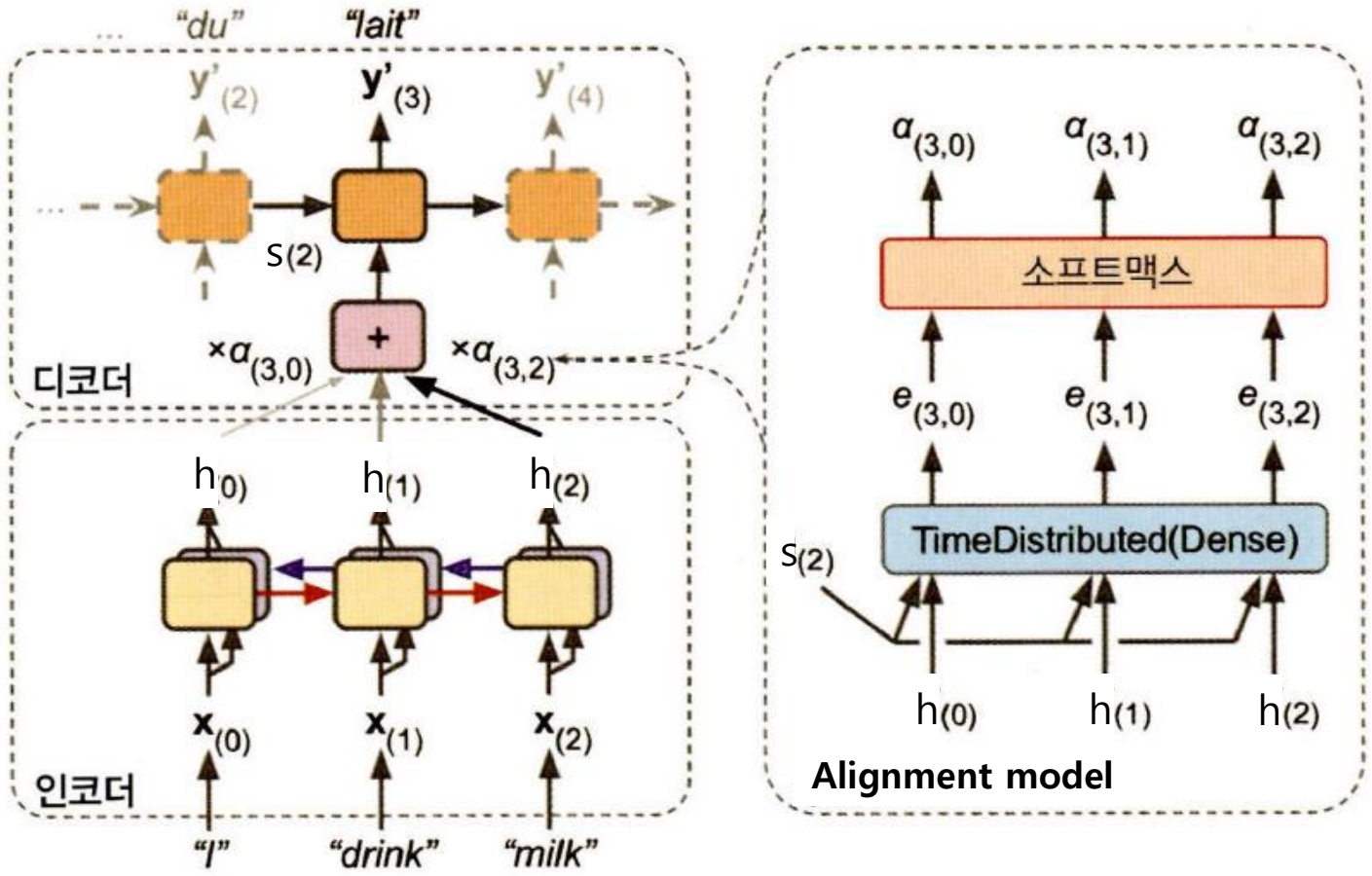
Attention

Problem of Vanilla seq2seq : information bottleneck



- ✓ Source sequence의 모든 문맥을 고정 길이 벡터에 모두 담는 것은 무리
- ✓ Source의 앞부분에 input되는 단어들과 targe의 끝부분에서 output되는 단어간에 거쳐가는 경로가 김

Attention



Notation

1. Attention score $e_{ij} = a(s_{i-1}, h_j)$
2. Attention weight (att. Dist.) $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$
3. Attention output $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

➡ $[c_t; s_t]$

Core Idea

- ✓ Target 문장의 단어와 대응관계에 있는 Source 문장의 단어를 골라내자
- ✓ Decoding 과정에서 매 step마다 encoder의 모든 단어(엄밀히 hidden state)에 접근하여 정보를 이용할 수 있게 하자

Attention

https://wikidocs.net/22893

Various attentions

이름	스코어 함수	Defined by
dot	$score(s_t,\ h_i) = s_t^T h_i$	Luong et al. (2015)
$scaled\ dot$	$score(s_t,\ h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
$general$	$score(s_t,\ h_i) = s_t^T W_a h_i$ // 단, W_a는 학습 가능한 가중치 행렬	Luong et al. (2015)
$concat$	$score(s_t,\ h_i) = W_a^T\ tanh(W_b[s_t; h_i])$ $score(s_t,\ h_i) = W_a^T\ tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
$location - base$	$\alpha_t = softmax(W_a s_t)$ // α_t 산출 시에 s_t만 사용하는 방법.	Luong et al. (2015)

By using attention

- ✓ 말도 안되는 NMT performance 개선
- ✓ Bottleneck 문제 해결
- ✓ Vanishing gradient 문제에 도움
- ✓ 설명 가능성(explain ability) 제공 (어텐션 가중치 시각화)

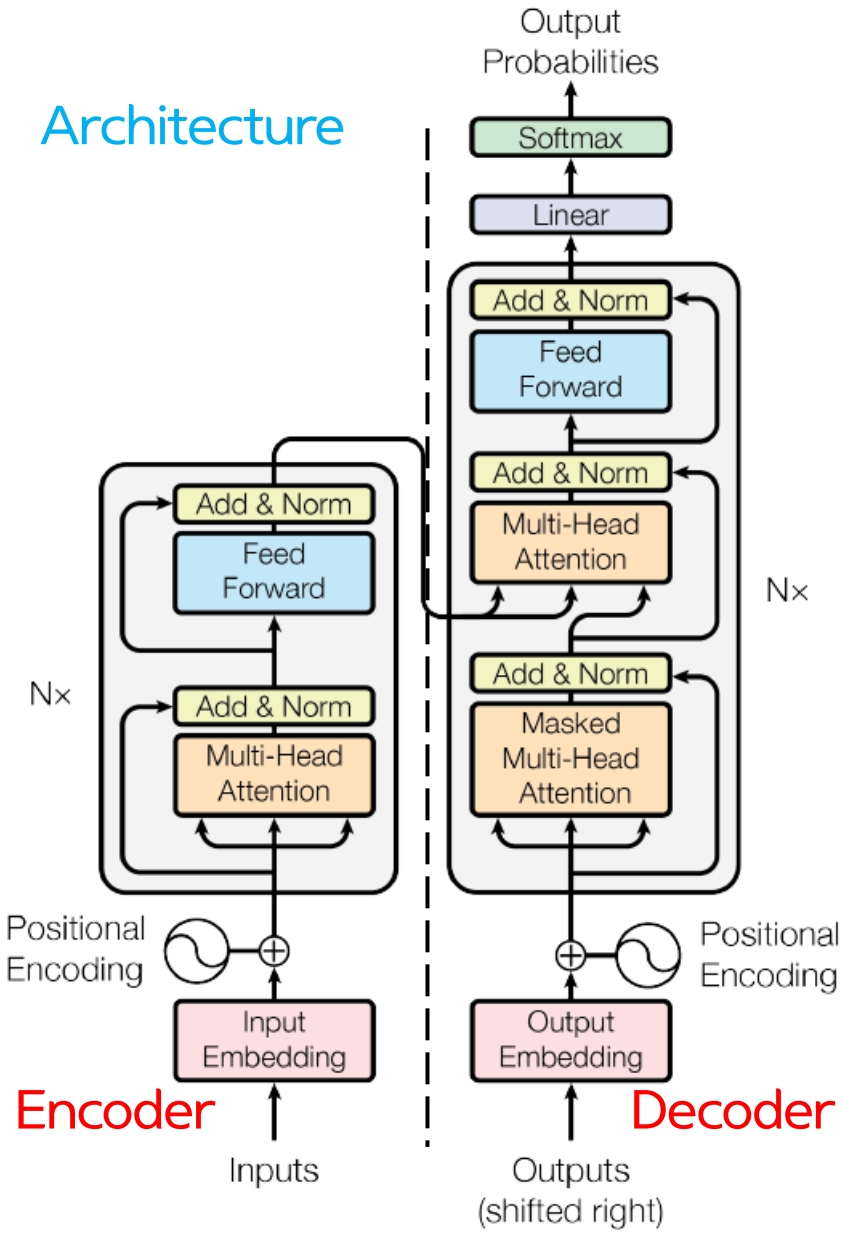
Attention is a general Deep Learning technique !

Topics

1. Text Generation & Sentimental Analysis
2. Encoder-Decoder Networks for NMT
3. Attention
4. Transformer

Transformer

Architecture



Remark

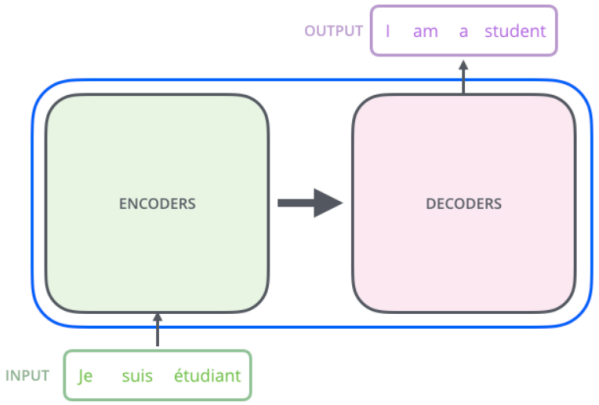
"Recurrent" component 나 Convolution을 사용하지 않고, Attention mechanism을 main component로 채택.

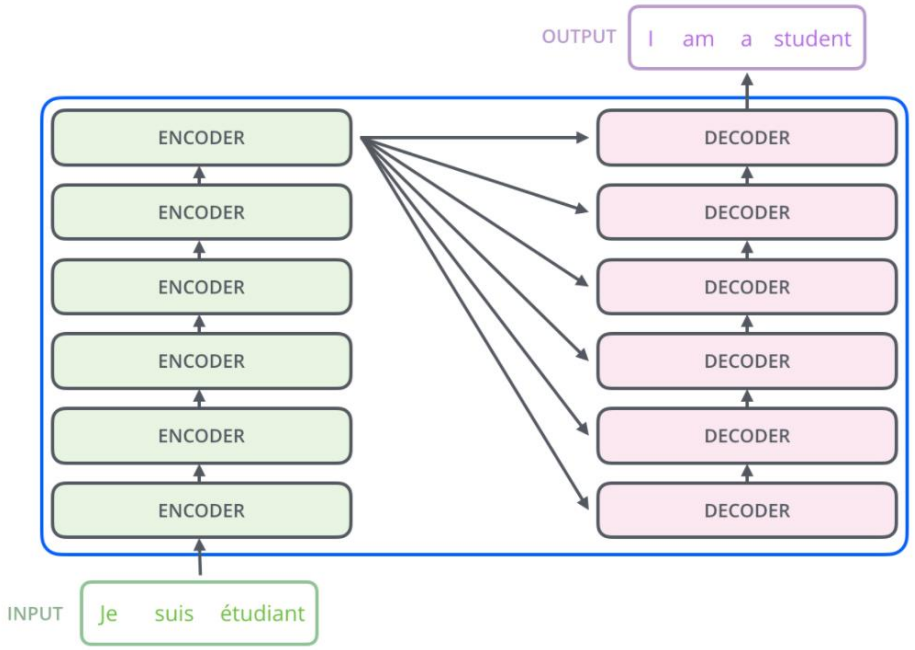
➡ *빠른 훈련, 우월한 성능*

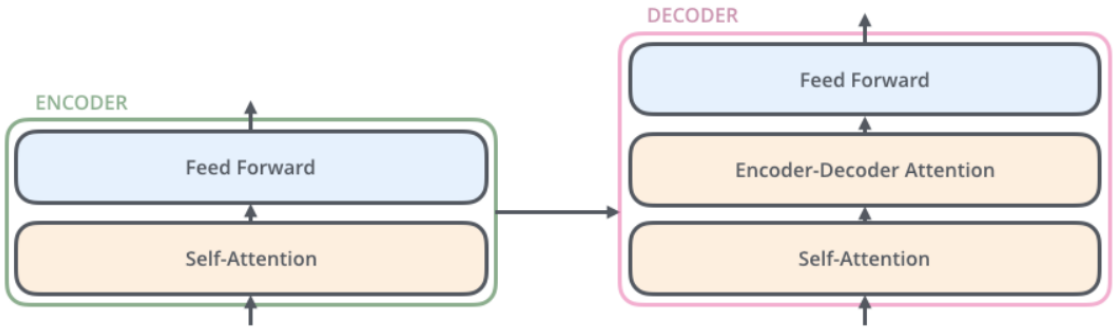
Terms

- Positional Encoding
- Self-Attention
- Scaled dot-product Attention
- Multi-Head Attention
- Masked Multi-Head Attention

Transformer





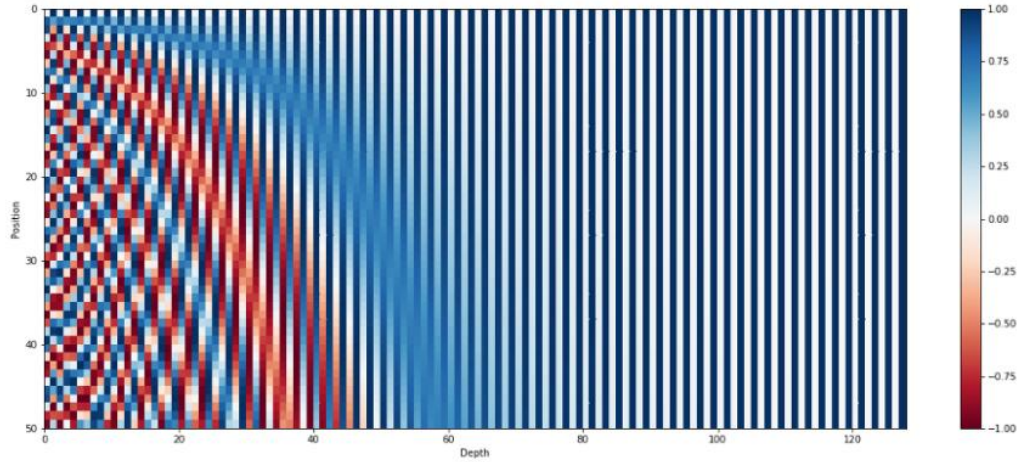


- encoding, decoding 각 과정에서
  여러 개의 인코더와 디코더를 쌓아 올린다.
- **인코더의 마지막 층 출력 벡터가 모든 디코더들에 전달된다.**
- 인코더에 들어온 input은 self-attention층을 거쳐
  Position-wise Feed-Forward Networks로 전달된다.
  (각 위치의 단어마다 독립적으로 동일한 FFN 통과시킴.)
- 디코더에도 인코더에 있는 두 layer가 동일하게 있다.
  두 층 사이에 seq2seq에서의 attention 기능을 수행하는
  Encoder-Decoder Attetion 층 추가. (디코더가 각 step에서
  source 단어들 중 가장 관련있는 부분에 집중할 수 있도록 함.)

https://nlpinkorean.github.io/illustrated-transformer/

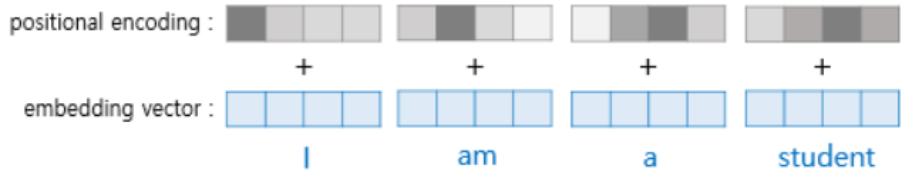
Transformer

1. Positional Encoding

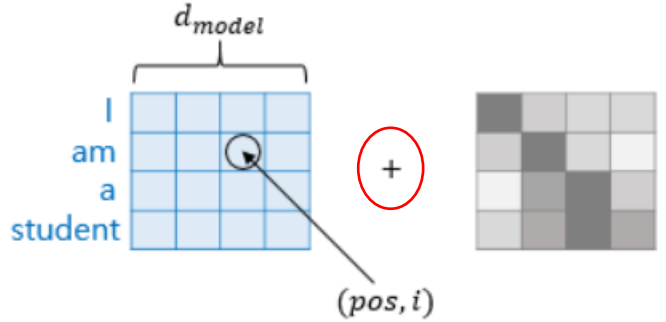


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Concept



Implement



- RNN, CNN의 부재로 모델이 스스로 시퀀스의 순서에 대한 정보를 보존하지 못하기에 별도의 encoding을 추가하여 벡터 표현에 위치정보가 포함되도록 한다.

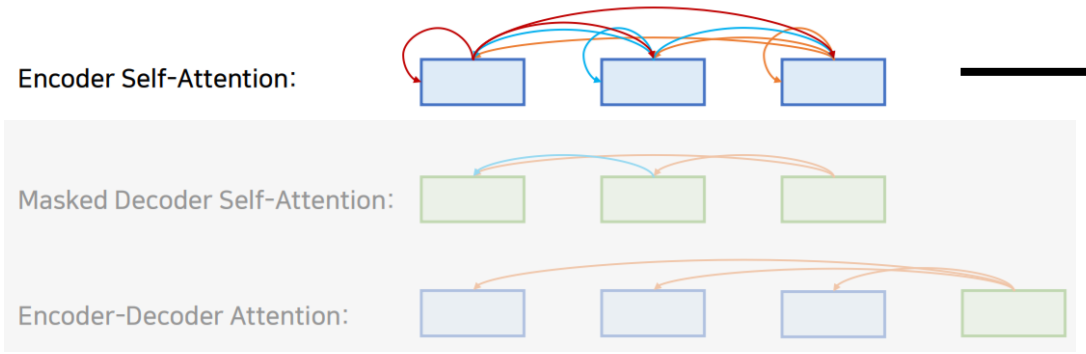
➡ *Embedding with Time signal*

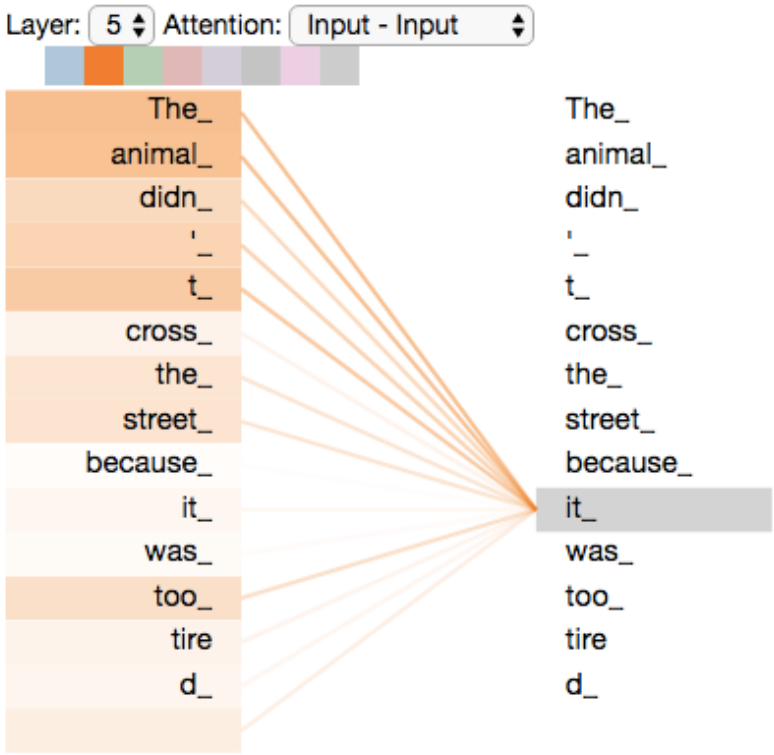
- 각 위치마다 고유한 인코딩이 만들어지며, 주기성이 있는 함수를 이용하여 단어들 간에 상대적인 위치도 학습할 수 있다.
- 임의의 길이의 긴 시퀀스에 대해서도 확장 가능하다.

https://wikidocs.net/31379

Transformer

2. Self-Attention (in Encoder)



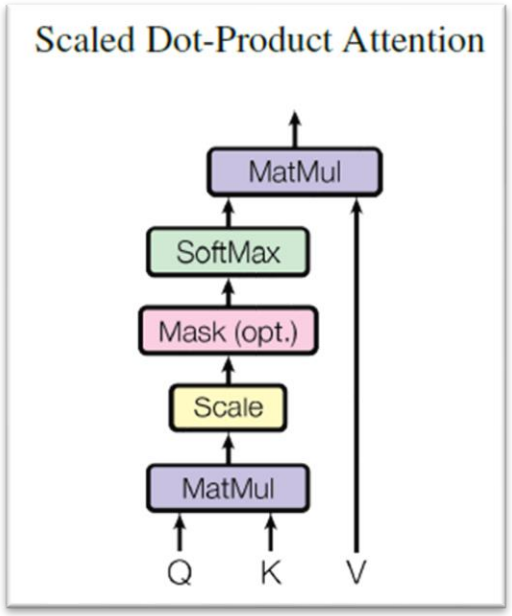


- '자신에 대한 주목' –> 하나의 시퀀스 내에서 각 원소가 다른 원소들과 어떻게 관련되는지 살펴보자.
- 그 관련성들을 확인함으로써 각 단어들이 문장 전체의 context를 더 잘 반영하도록 encoding 할 수 있다.

https://github.com/ndb796/Deep-Learning-Paper-Review-and-Practice
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

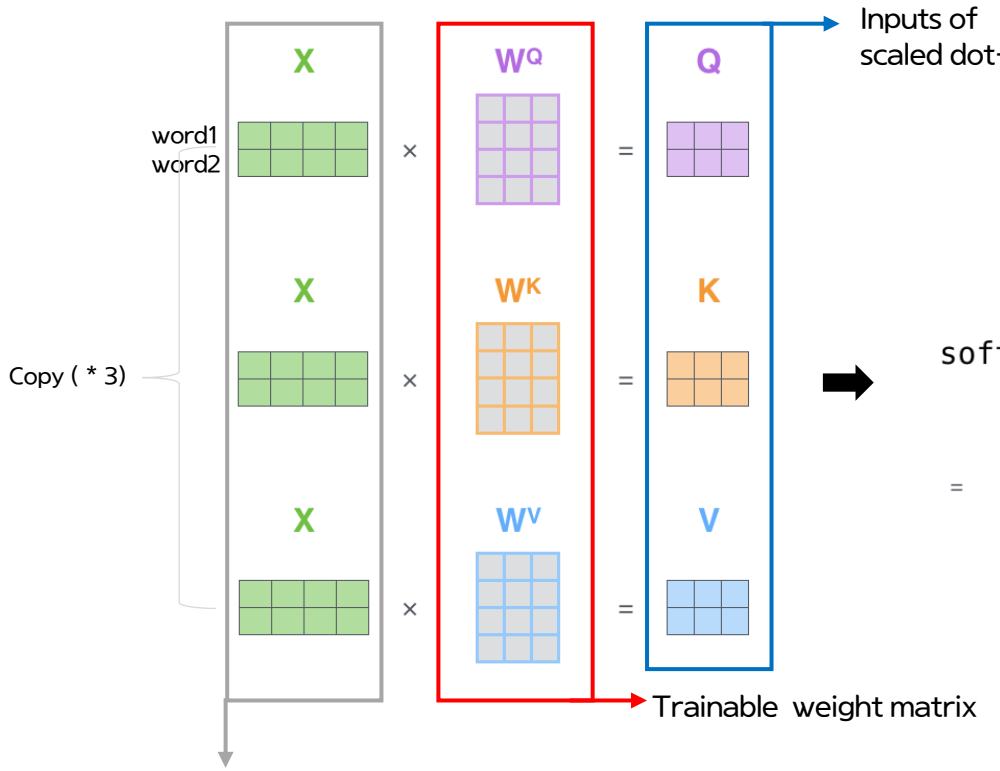
Transformer

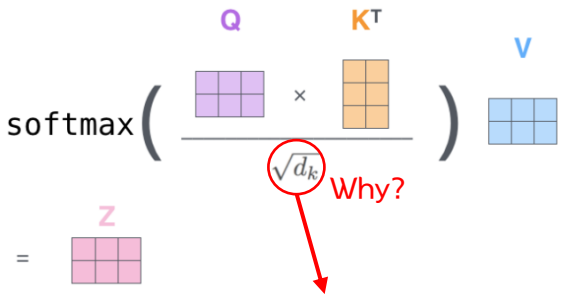
2. Self-Attention (in Encoder)

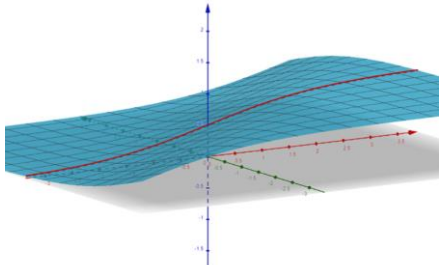


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Procedure





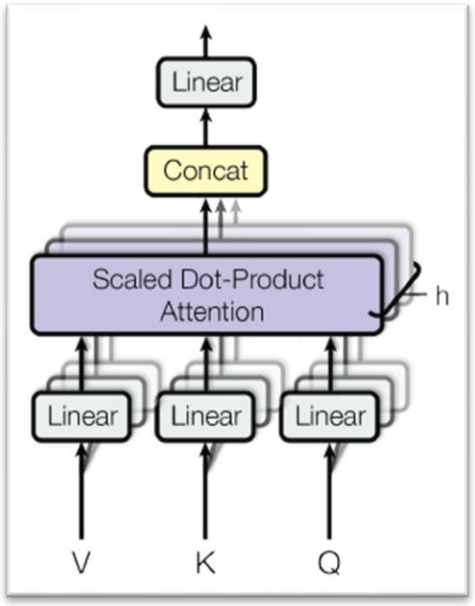


Softmax 는 input이
극단으로 갈수록saturated !!

- 각 단어는 3갈래로 복사되어 3개의 서로 다른 선형변환 W를 거쳐 query, key, value 벡터로 변환된다.
- 각 쿼리에서 모든 key 벡터들과의 내적을 통해 유사도를 구한다. (이후 key벡터의 차원의 제곱근으로 나눠 down-sizing)
- 구한 유사도점수를 Softmax를 취해 어텐션 가중치로써 만들어주고, 각 단어의 value 벡터들과 weighted sum 한다.

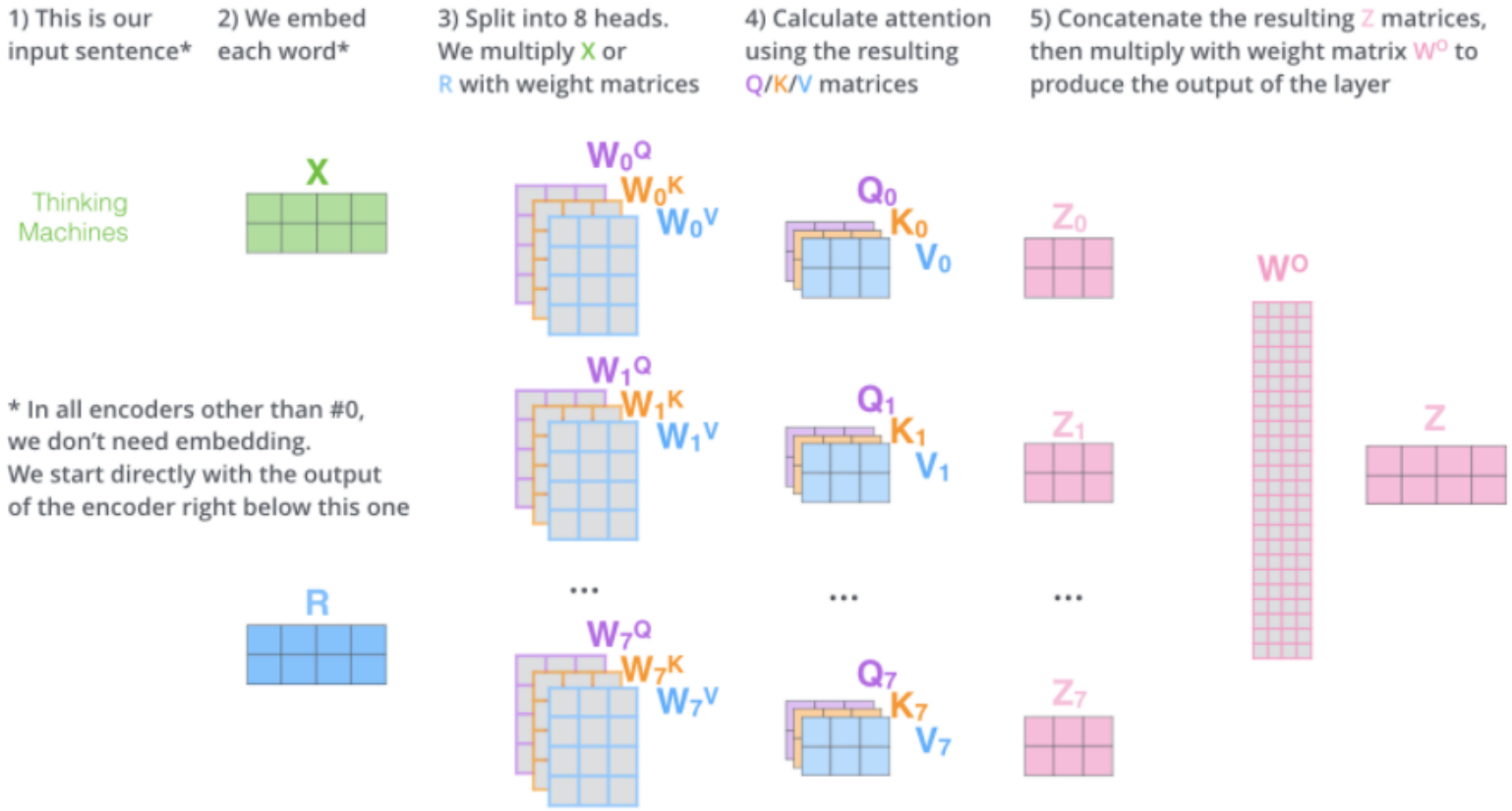
Transformer

3. Multi-headed attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

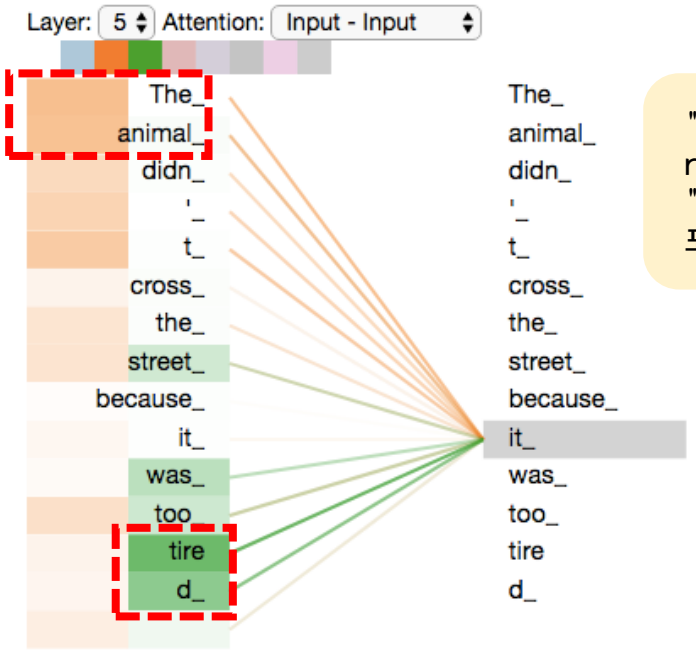
Procedure



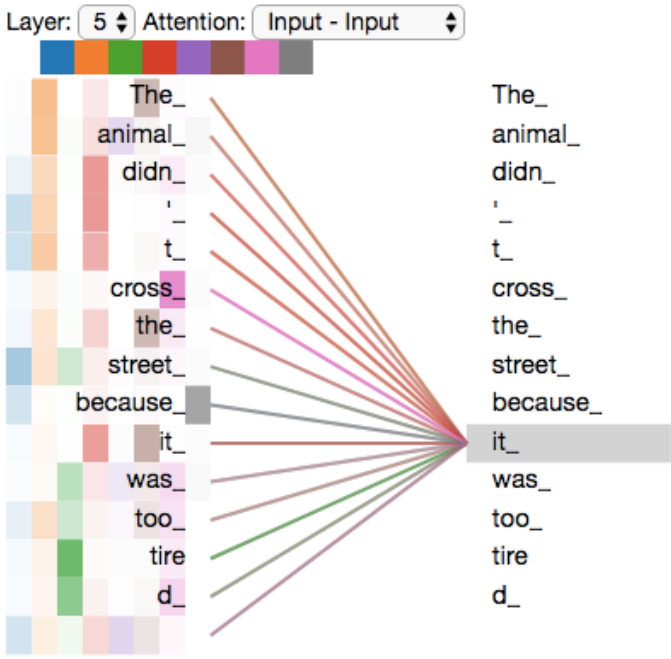
- scaled dot-product attention을 설정한 어텐션 헤드 수만큼 반복한다.
- 각 헤드에서 얻은 어텐션 결과 행렬을 concat하고 Wo를 곱하여 최종 선형변환을 수행한다.
- 이렇게 얻어져 다음 encoder로 전달될 결과 행렬은 초기 입력 행렬과 / 이전 층의 입력 행렬과 동일한 shape를 갖는다.

Transformer

Benefits of multi-head



"그 동물"과 "피곤"에 대한 representation 모두를 "그것(It)"의 representatio에 포함시킬 수 있다!!

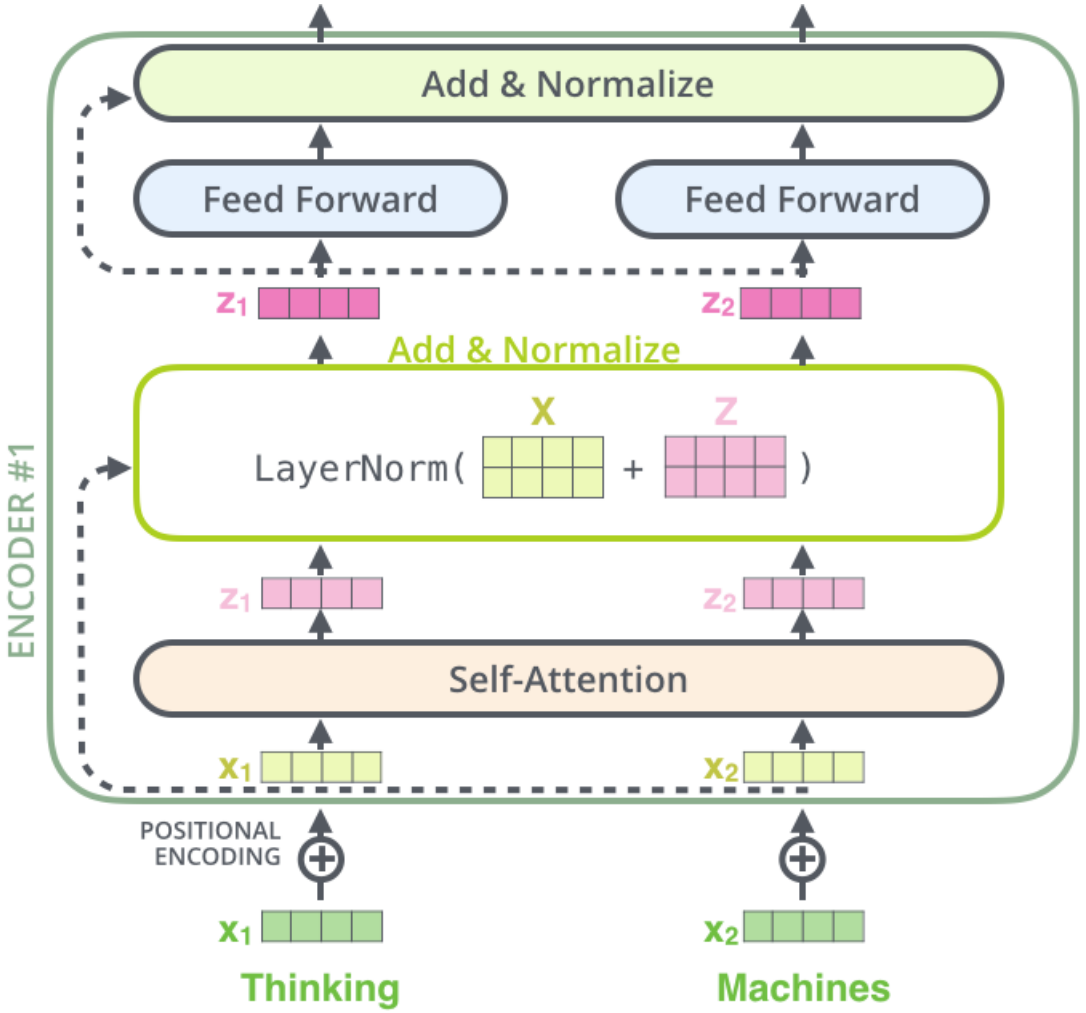


- 가중치행렬 Wq, Wk, Wv 는 각 헤드에서 모두 다름.
  –> 서로 다른 시각에서 단어들 간의 관계를 바라보겠다는 것.

- 랜덤 초기화되는 Wq, Wk, Wv는 각각 다른 representation의 subspace를 생성하도록 학습되며, 학습된 후 입력 시퀀스의 벡터들을 각기 다른 representation 공간으로 투영하게된다.

Transformer

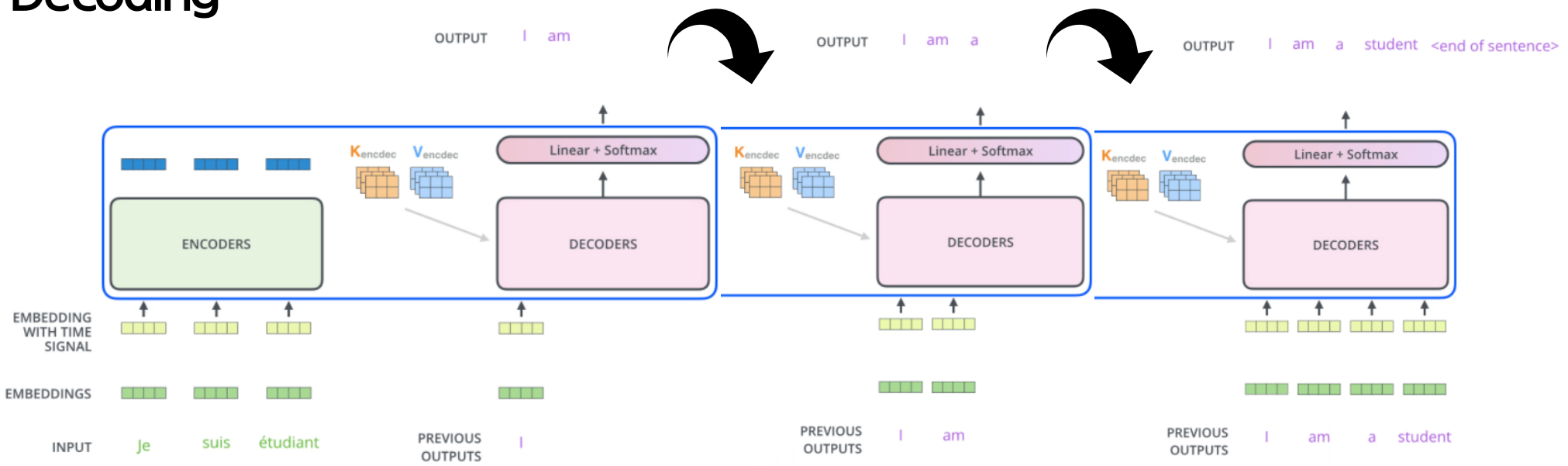
Residual connection & Layer normalization



- 각 sub layer 마다 connection을 추가하였고, 뒤이어 layer-normalization을 적용하였다.

Transformer

Decoding

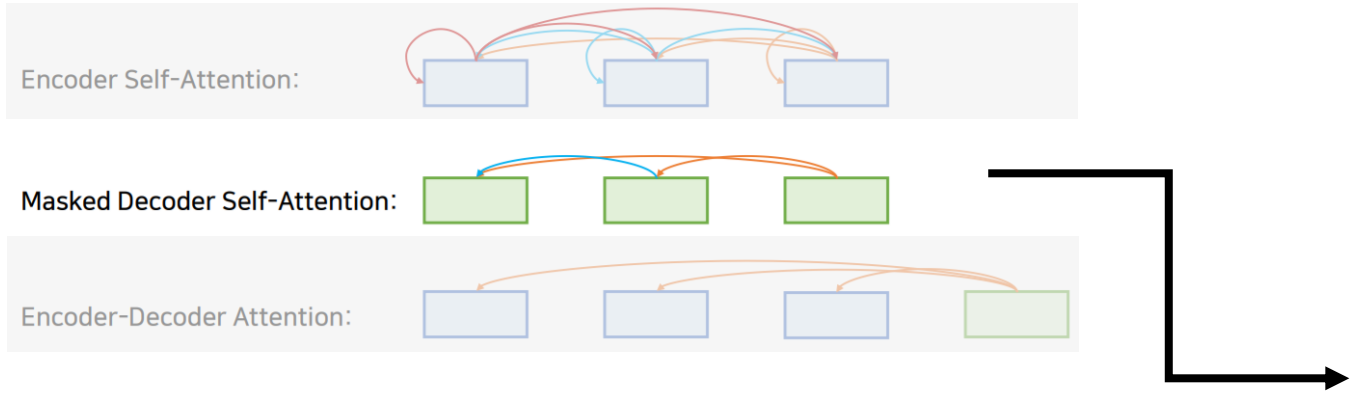


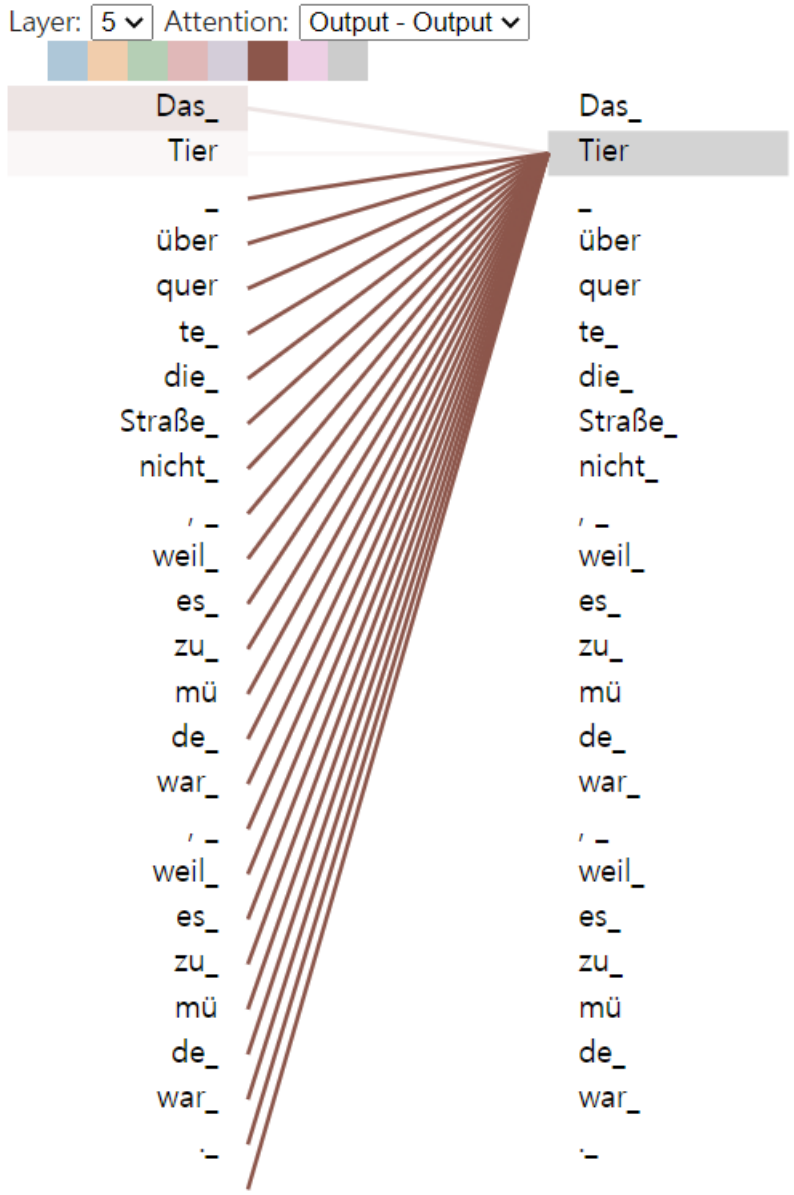
- Inference 단계에서, decoder는 <sos>토큰으로 시작, 이전 step의 출력 단어를 현재 step의 input으로 취하여 순차적으로 prediction 시퀀스를 만들어낸다.
- Training 시 Teacher Forcing에 의해 target 시퀀스를 한꺼번에 입력받아 각 step에서의 단어를 예측하도록 훈련된다.

문제점 ➡ RNN 계열의 신경망은 입력 단어를 매 시점 순차적으로 받기 때문에
다음 단어 예측에 현재 시점 이전에 입력된 단어들만 참고할 수 있으나, 트랜스포머에서는 그렇지 않음

Transformer

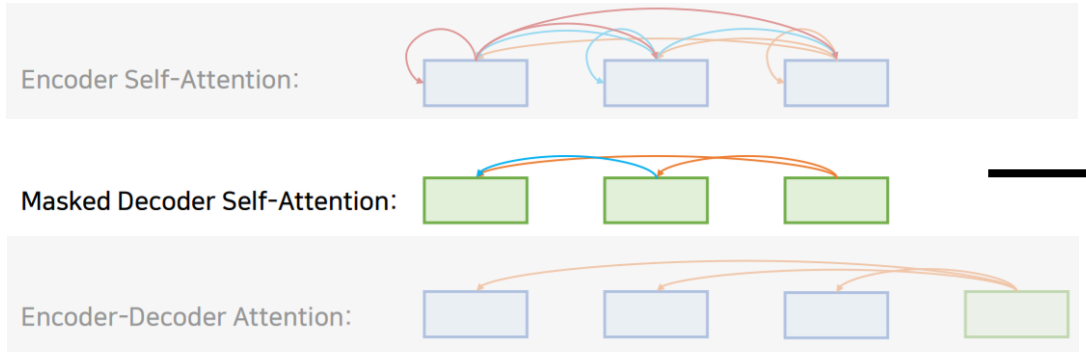
4. Masked Self-Attention (in Decoder)

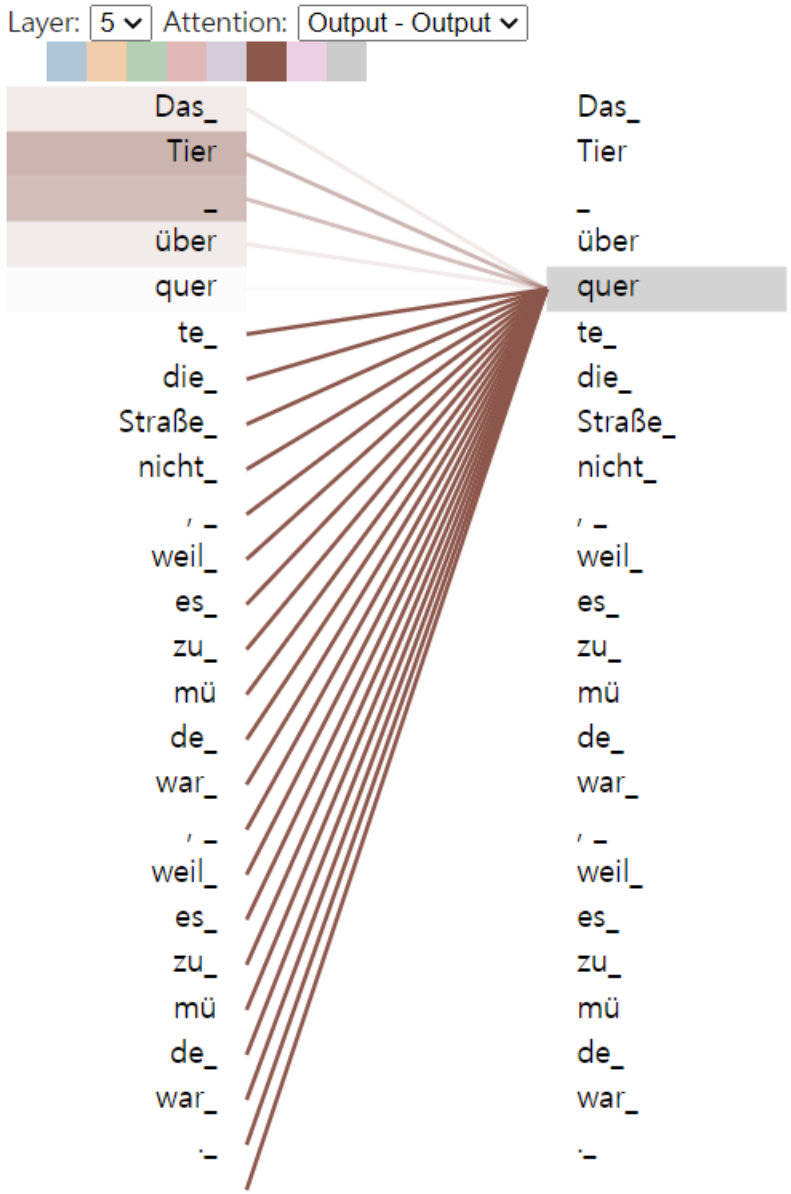




Transformer

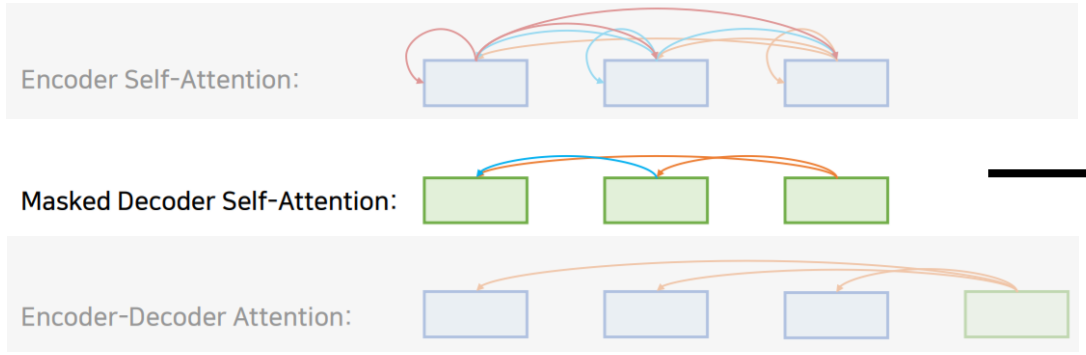
4. Masked Self-Attention (in Decoder)

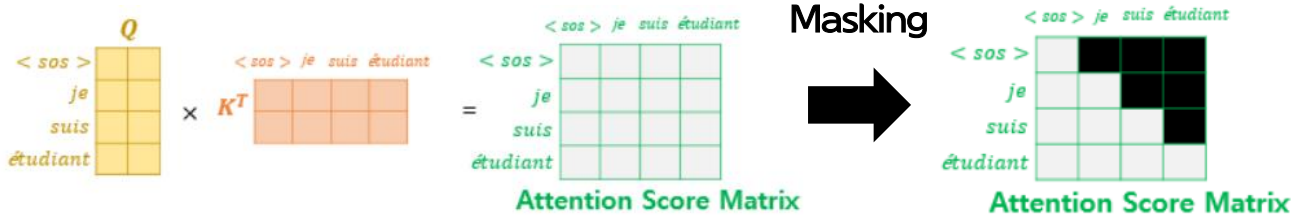


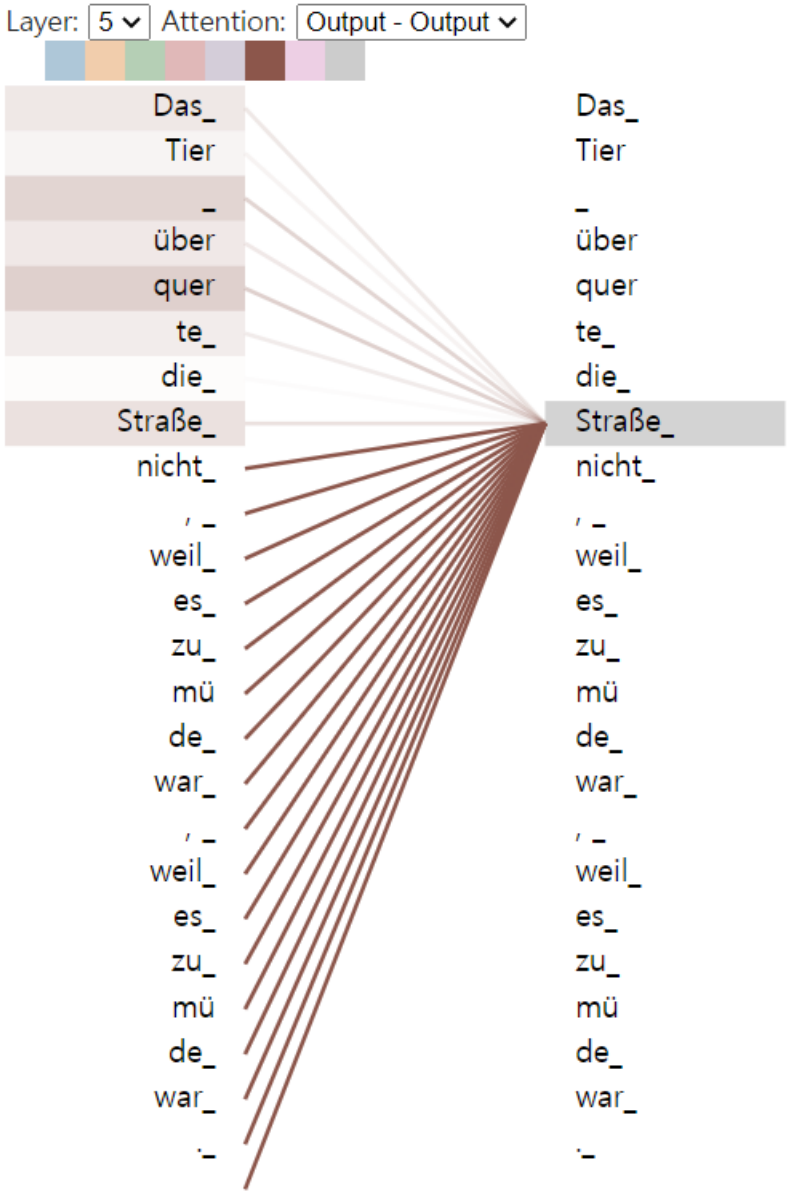


Transformer

4. Masked Self-Attention (in Decoder)



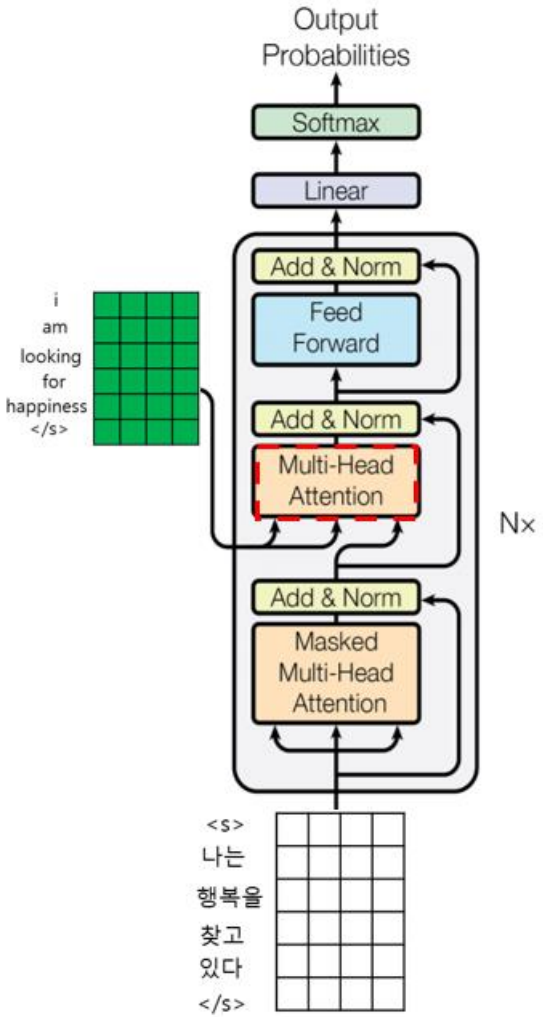


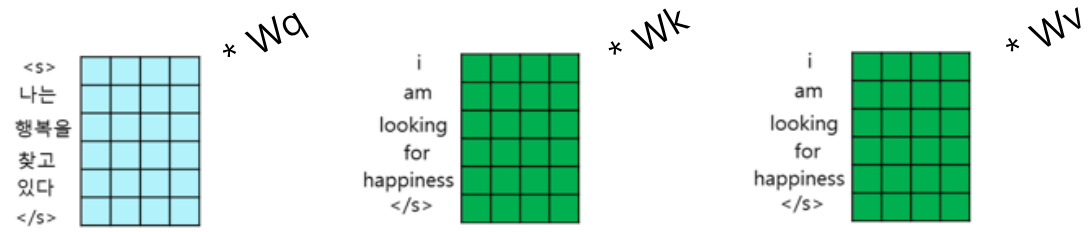


- 현재 시점의 예측에서 현재 시점보다 미래에 있는 단어들을 참고하지 못하도록 look-ahead mask 도입.

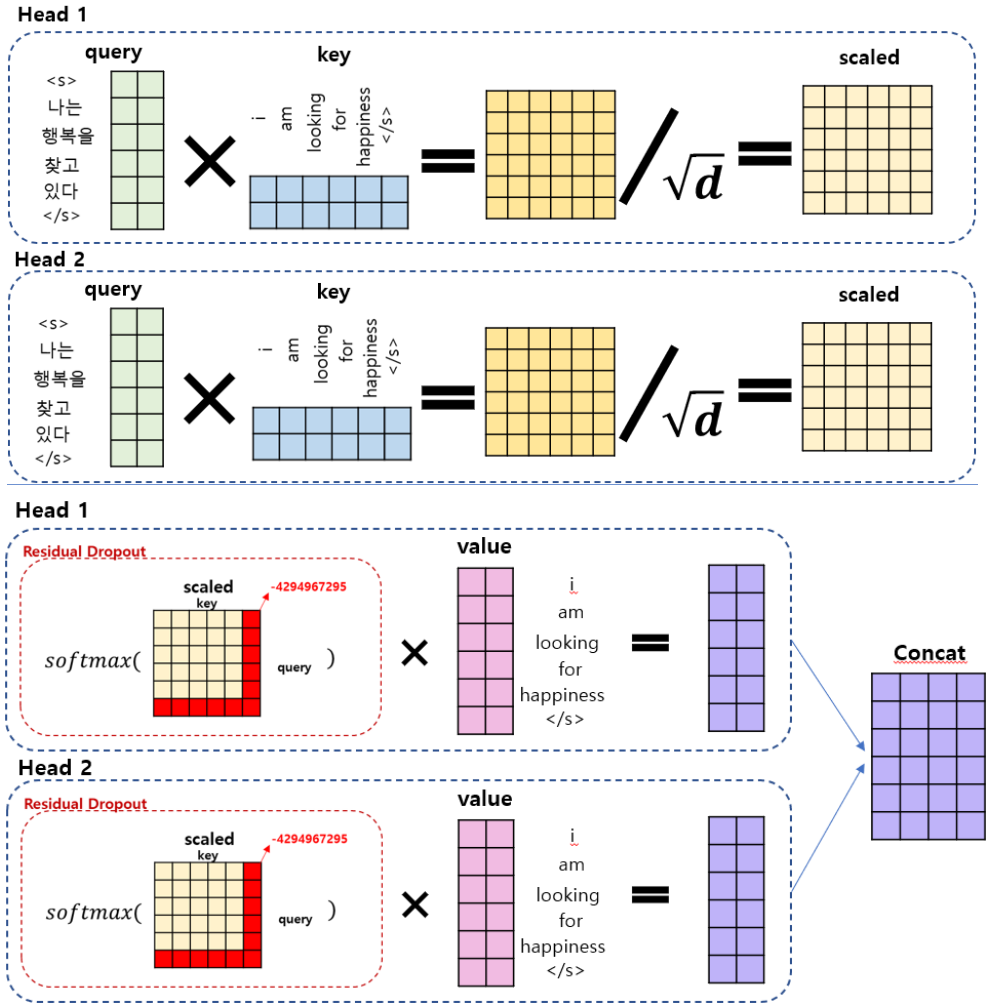
Transformer

5. Encoder-Decoder Attention





In Decoder from Encoder



https://www.youtube.com/watch?v=xhY7m8QVKjo

Transformer

Output

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (`argmax`)

5