—

Attention Is All You Need (2017)

Inbox	☐
Due date	@2024년 1월 21일
Project	AITE-R
Resources	ML/DL 이론, Fundamental Papers
Done	☑
Type	

Task

트랜스포머 논문리뷰

Keywords

self-attention

auto-regressive

positional encoding

Contents

Introduction

RNN은 지금까지 여러 노력들을 통해 encoder-decoder 구조에서 sota 성능을 도달해왔다. 하지만 recurrent 한 구조는 parallel하지 않고 시점에 따라 input과 output을 순차적으로 얻어야 한다는 근본적인 제약이 있다.

transformer 는 sequential 데이터와 sequential 모델에 이러한 제약을 극복하기 위해 recurrent한 연결을 끊고 attention mechanism만을 의존하는 모델로, parallelization 하며 dependency를 해결한 모델로 sota 성능을 도달했다.

Background

sequential data를 처리하는데 있어 transformer는 multi-head attention을 사용함에도 operation 수가 줄어들었다. input과 output에 대한 representation을 계산하는데 있어 **오직 self-attention만을 사용**하는 첫번째 모델이자 sequence를 끊어 recurrent나 convolution을 사용하지 않는 첫번째 모델이다.

Model Architecture

가장 경쟁하는 시퀀스 모델은 encoder-decoder 구조의 모델이다. 이는 input sequence $(x_1, \dots, x_n)$를 하나씩 넣어 representations $z = (z_1, \dots, z_n)$ 를 얻고 주어진 z에 대해 output sequence $(y_1, \dots, y_n)$를 하나씩 얻는다. 각 시점에서 모델은 생성하기위해 이전 시점에서 생성한 데이터를 추가적인 입력으로 사용하는 **auto-regressive**하다고 할 수 있다.

auto-regressive

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \dots, x_{i-1}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$

시점 i에서 이전 시점의 정보를 사용하는 모델이 auto-regressive model

- 자기 자신을 입력데이터로 하여 스스로 예측하는 것
- 현재 시점까지 생성한 output을 다음 시점의 input으로 사용해 output을 예측하는 것
- 따라서 특정 시점의 데이터는 이전 시점의 모든 데이터와 dependency를 갖는다.

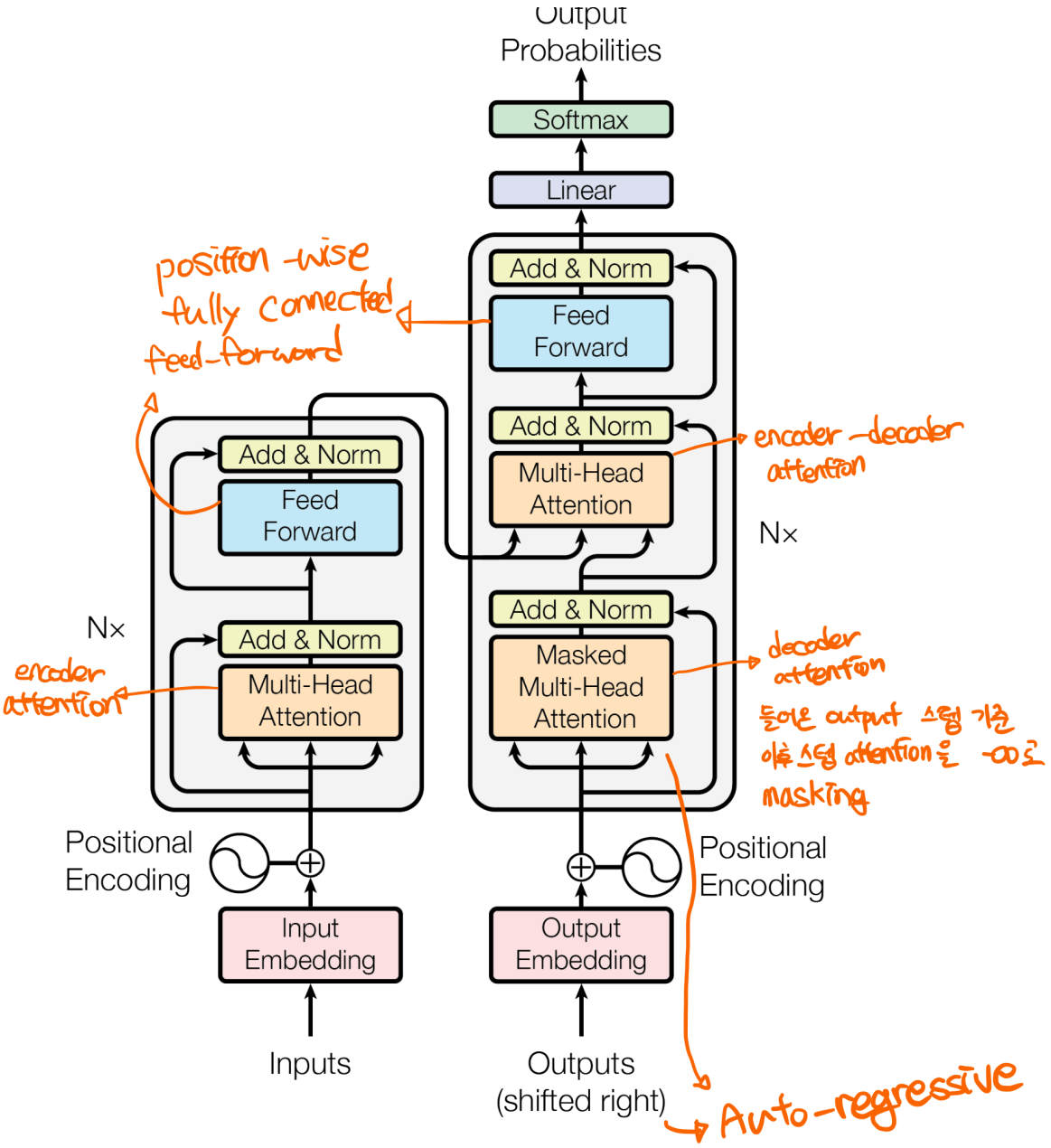


Figure 1: The Transformer - model architecture.

encoder decoder stacks

트랜스포머는 $N(N=6)$개의 encoder와 decoder를 각각 쌓은 multi-head 구조이다.

인코더에서는 각각의 head를 self-attention으로 weighted하는 구조인 **multi-head self-attention mechanism** 을 사용한다. 이 메커니즘, sub-layer 이후에는 position-wise fully connected feed-forward 한 sub-layer가 위치해 있다. 그 다음 residual connection과 layer normalization 를 수행한다. **시점 또는 단어 단위의 output embedding 차원은 512**으로 하였다.

디코더또한 마찬가지로 $N(N=6)$개가 존재하고 encoder stack에서 나온 outputs에 대한 multi-head attention을 수행하는 masked 된 sub-layer가 추가되어있다. decoder

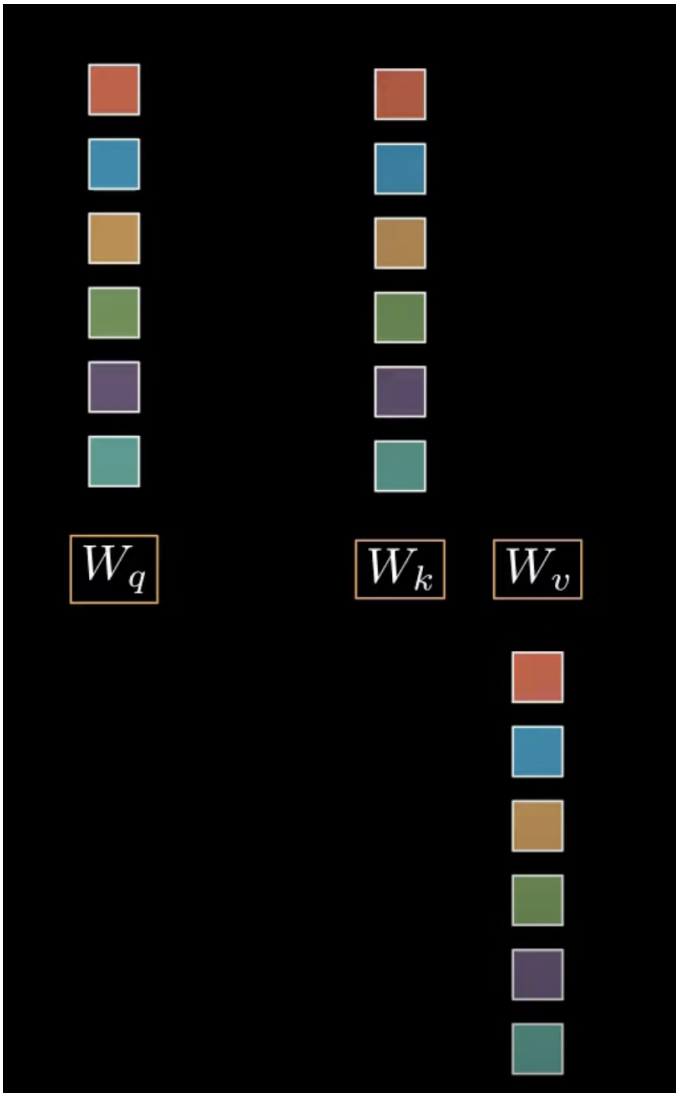
의 attention score를 계산할 때 position 이후 시점의 개입, 즉 예측하고자하는 단어를 사용하면 일종의 치팅이기에 이를 예방하기위해 $-\infty$로 masking을 한다. 이는 추후 softmax에 의해 0.00에 가깝게 가중치가 바뀌게 된다.

논문 figure 에서도 볼 수 있듯 encoder attention, decoder attention, encoder-decoder attention 3 종류의 attention layer를 사용한다

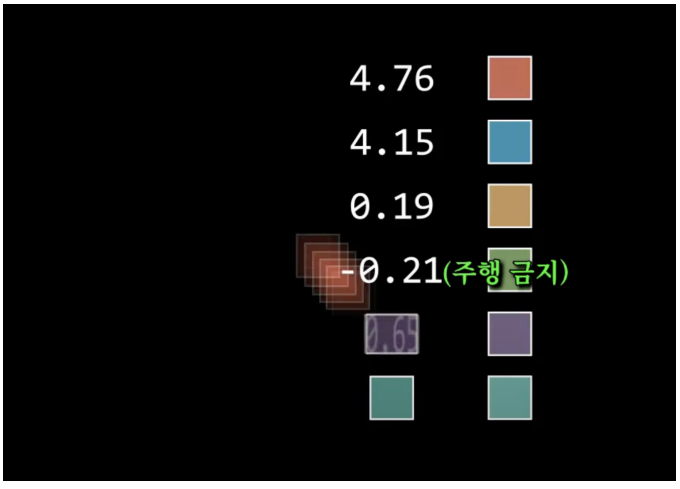
attention (scaled dot product attention)

6개의 워드벡터를 시각화해 self-attention mechanism 를 이해해보자

시퀀스를 6개의 벡터이자 텐서로 생각할 수 있다.



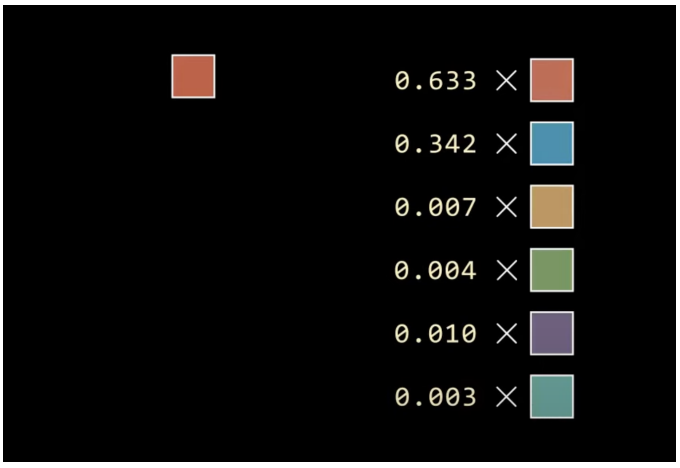
쿼리 키 밸류는 동일한 벡터를 weight matrix인 W_q W_k W_v 를 통해 query key value 로 선형변환된다.



대상이 되는 쿼리 단어 하나를 가지고 여러 개의 키들과 inner product를 통해 그 유사도이자 가중치를 얻을 수 있다.

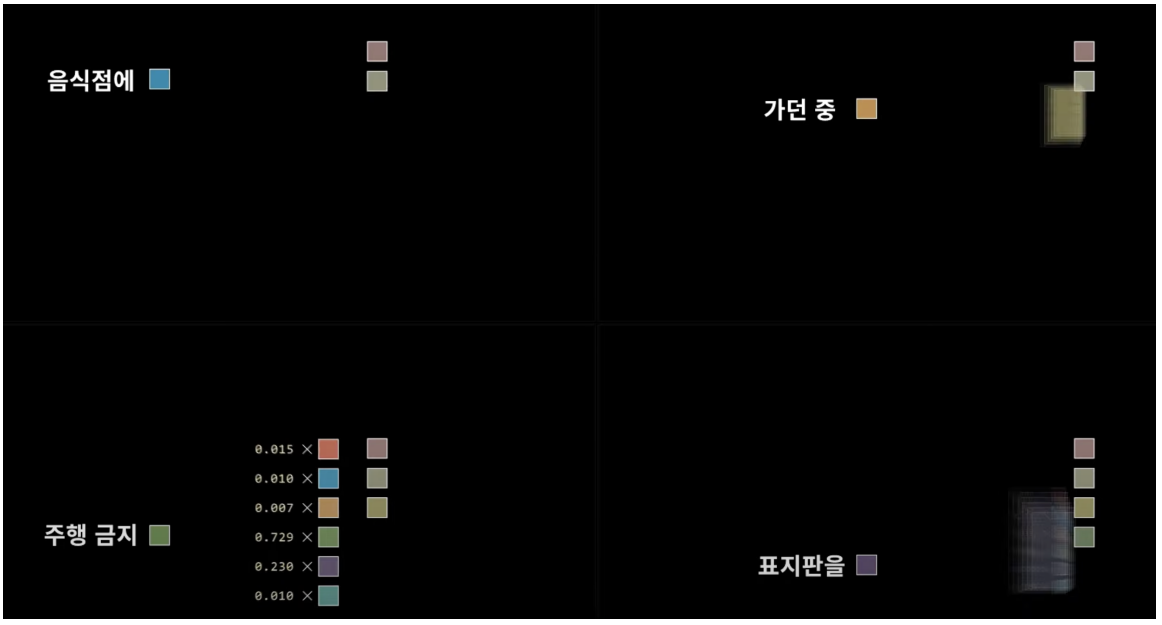
이 가중치 QK^T는 범위가 스케일링되어있지 않다. 가중치 값이 양수일수록 쿼리 단어 벡터와 키 단어 벡터가 상대적으로 유사하고 큰 음수일수록 다르다는 것을 알 수 있다.

이를 key의 차원 d_k에 대해 $\sqrt{d_k}$ 을 scaling factor로 나누는 스케일링을 수행한다.

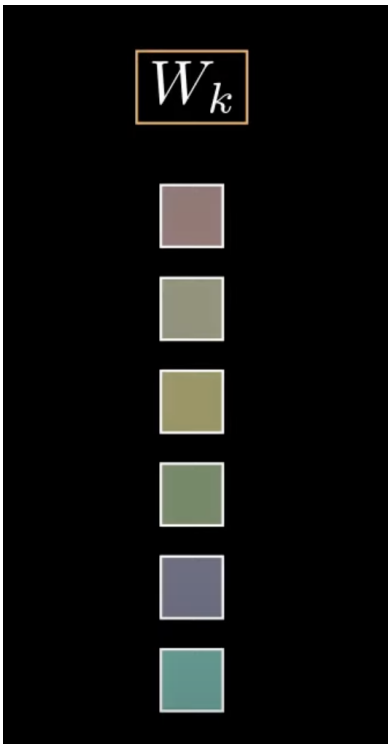


유사도의 성질을 유지하면서 양수로 모든 값을 바꾸고 합이 1이 되도록 만들기 위해 softmax함수를 사용해 총합이 1이고 모두 양수인 가중치로 변환할 수 있다.

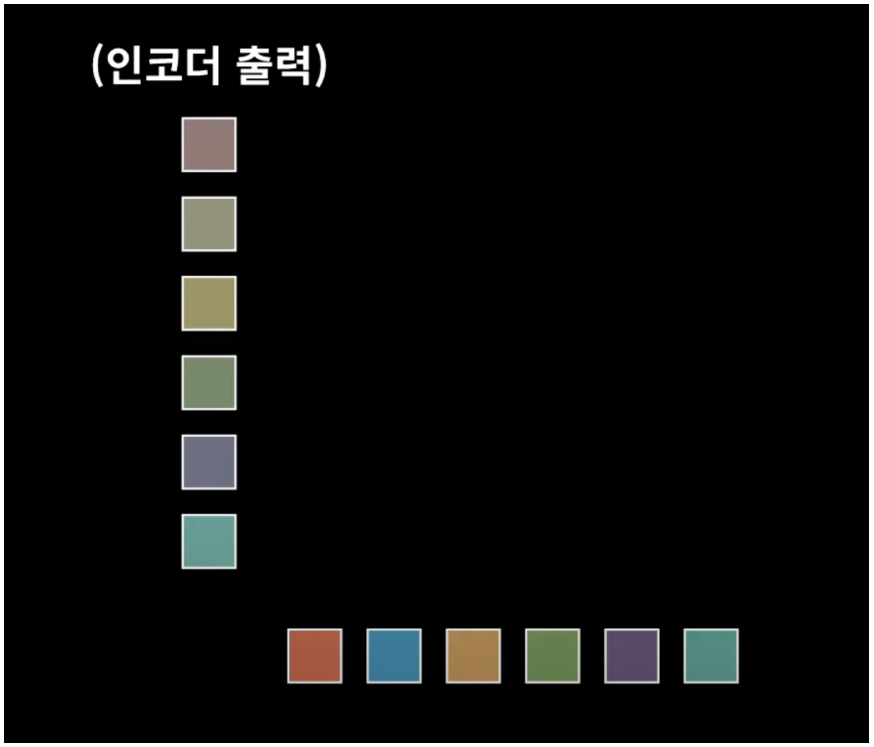
하나의 쿼리 벡터에 대해 키 벡터들의 weighted sum으로 밸류 벡터 , 빨간색 단어 정보에 대한 가중치합을 얻는다.



모든 쿼리 벡터에 대해 동일한 방법으로 구할 수 있다. 얻게된 output을 다시 다음 layer의 input으로 사용한다.



첫번째 layer을 지나고나서부터는 벡터하나가 단어 하나만의 의미를 갖고 있는 것이 아니고 특정단어 'a'를 분석하는데 있어 여섯 개의 단어와의 관련도가 반영된 'a'의 정보를 얻게 된다.



여러 레이어를 거쳐 decoder에서 next token prediction을 수행할 때, encoder 출력인 키 들 중에서 낱말 'a'를 본다는 것은 'a' 정보만 보는 것이 아닌 낱말 'b', 'c' 'e' 'f' 'g'와 함께 맥락 정보가 담긴 단어들로 attention score를 구하게 된다.

단순히 weighted sum으로 계산한는 방법으로는 위치정보도 없고 근처 키 맥락 정보도 없는 값이다. 따라서 트랜스포머는 위치정보를 주입하기위해 positional encoding을 추가하고, 맥락 정보를 주입하기 위해 self-attention 이라는 방법을 사용했다.

position-wise feed-forward network

attention sub-layer 이후 각 position, 단어 단위마다 독립적인 feed forward layer를 2개로 구성된 sub-layer를 사용한다.

. Another way of describing this is as two convolutions with kernel size 1.

☐ position-wise한 네트워크가 conv1d 에서 kernel size가 1인 convolution과 동일하다고 생각할 수 있나?

→ 내 생각은 'yes'이다. 각 단어, 시점마다 문맥에 해당하는 feed forward network를 가지고 있는 것으로 이해할 수 있다.

positional encoding

- 무엇이고 왜 쓰는가

단어의 임베딩값에 벡터인 positional encoding 값을 추가한다. 이는 seq2seq 또는 recurrent하게 연결되어있지 않은 transformer에게 단어 간의 순서 정보를 전달하기 위함이다. 순서 정보가 중요한 이유는 단어가 위치한 자리에 따라 그 의미가 완전히 달라질 수 있기 때문이다. 'it'이라는 단어가 문장 안에 위치한 자리에 따라 문맥에 따라 해석이 달라질 수 있는 것을 생각해보면 된다.

- 어떻게 위치정보를 전달하는가

input 값에 따라 시퀀스의 길이가 달라진다. 길이가 달라지더라도 position embedding 값은 항상 동일하게 유지한다. 또한 position embedding 값이 원래의 representation embedding 값에 비해 크게 된다면 그 의미가 사라질 것이다.

단순히 첫 번째 토큰을 0로 마지막 토큰을 1로 normalize한 인코딩 값을 위치정보로 쓰게 되면 시퀀스 길이가 얼마가 되는지 알 수 없고 시퀀스 길이에 따라 토큰 간 위치의 변위를 알 수 없다. 이를 막고 벡터를 부과하기 위해서 삼각함수를 사용한다.

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

sine cosine 함수 안 분모가 커질수록 주기가 길어진다 → 벡터 임베딩 차원 $2i$ 가 d_{model} 에 가까워질수록 주기가 길어진다.

- 왜 concatenate 이 아닌 summation 을 하는가

concat과 sum 둘다 장단점이 존재하는 방법이다.

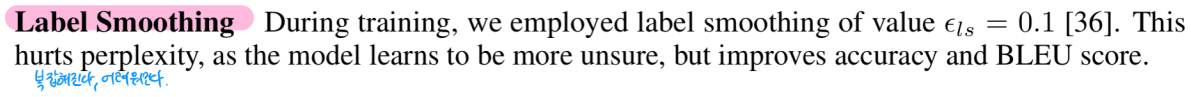
concat 하게 되면 의미 정보와 위치 정보가 각각 고유한 차원을 갖게 된다는 장점이 있으나 차원이 늘어나 계산 코스트가 늘어난다는 단점을 갖는다. 반대로, sum 하게 되면 두 정보가 더해지기에 섞이지만 차원이 늘어나지 않는다.

왜 summation을 했는지 정확한 이유는 모르지만 2017년의 GPU 연산문제로 선택하지 않았을까 생각이 든다.

Training

base / big model 두 개를 나누어 학습을 진행하였다.

label smoothing



논문에서는 라벨스무딩을 통해 모델이 좀 더 불확실성을 학습하며 더 어려워지지만 accuracy와 BLEU가 향상되었다고 말한다.

라벨 스무딩 은 `Szegedy et al., Re-thinking the inception architecture for computer vision` 에서 처음 나온 개념이다.

classification task에서 Hard label 을 soft label 로 바꾼다.

Hard labeling : 정답은 1 오답은 0으로 원핫 인코딩, soft labeling : {0, 1} 이 아닌 [0,1] 사이 실수 값으로 인코딩한다.

K 개의 클래스와 smoothing parameter α 에 대해 다음 수식으로 계산한다.

$$
$$

클래스가 5개, 스무딩 파라미터가 0.2, 클래스 1이 정답이라면 아래처럼 인코딩된다.

$y_1^{LS} = 1 \cdot (1 - 0.2) + 0.2/5 = 0.84$

$y_2^{LS} = 0 + 0.2/5 = 0.4$

Results

hyperparameter 를 튜닝하는데 dev 셋을 사용해 beam size = 4 인 beam search를 사용

beam search

greedy encoding 을 보완하기 위한 방법이다.

greedy encoding은 해당 시점에서 가장 확률이 높은 하나의 정답만을 고르는 것이다. 이는 틀린 답이 한번 나오게 되면 다시 output이 다음 시점에서의 input으로 들어가기에 성능에 문제가 될 수 있다.

beam search 는 이번 시점에서 확률이 높은 k개를 선택하고 이들 다음 시점에서 k개마다 k개를 골라 k^2 개의 경우가 생긴다. 이중에서 다시 k개만을 선별한다. <EOS> 토큰에 도달했을 때도 k개의 문장 후보를 가지고서 가장 확률이 높은 문장을 선택하게 된다.

빔서치 의 적용 여부가 NLP 모델의 성능에 크게 기여한다는 것은 잘 알려진 사실이다.

Conclusion

transformer는 multi-head self-attention mechanism을 전적으로 기반한 시퀀스 모델이자 encoder-decoder 구조의 모델을 대체하는 모델임을 논문의 저자들이 보였다.

translation task에서 transformer는 rnn, cnn에 비해 상당히 빠른 속도로 학습하였고 english-to-french translation task에서 sota 성능을 달성하였으며, 기존 모델들에 ensemble 기법을 적용하였음에도 성능이 더 좋았다.

> Reference : https://youtu.be/6s69XY025MU?si=ccFxnrsT5d3oB7pS

Comment & Feedback