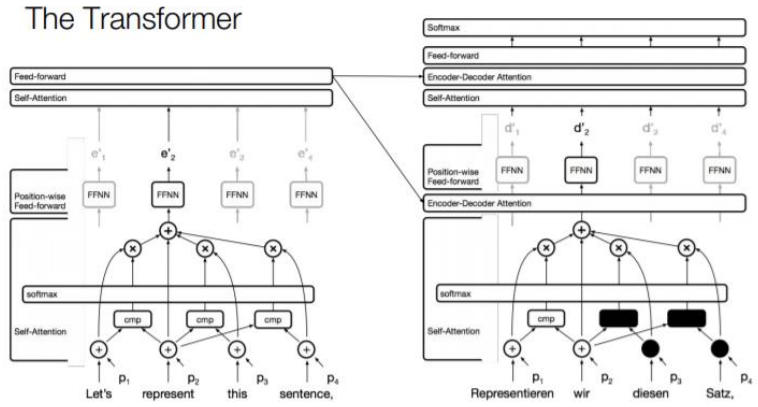
Lecture 14. Transformer and Self-Attention

Ⅰ. RNN, CNN, and Self-Attention

Sequence modeling에서 가변 길이의 데이터를 고정 크기의 데이터로 표현해야 한다. LSTM에서 시퀀스의 길이를 맞춰야 매 단계마다 빠르게 계산할 수 있기 때문이다. 하지만 RNN 계열의 모델은 parallelization이 불가능하고 long-term dependency를 잘 반영하지 못한다. Parallelization이 불가능한 이유는 하나의 단어 이전의 단어가 계산되어 hidden state로 이전의 단어에 대한 값이 넘어가야 다음 단어에 대한 계산이 가능하기 때문이다. 또한 RNN 모델은 vanilla RNN보다 dependency에 강한 편이지만 앞 단어의 영향력이 뒤로 갈수록 옅어진다는 long-term dependency 문제를 여전히 가지고 있다. CNN 모델은 병렬 처리가 가능하지만 long-term dependency를 위해 다수의 convolution layer가 필요하다는 단점이 있다.

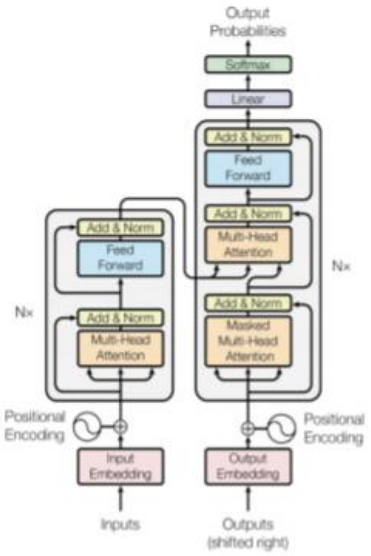
self-attention은 병렬화가 가능하면서 동시에 long-term dependency 문제를 해결해 RNN과 CNN을 대체할 수 있는 learning mechanism으로 등장하게 되었다. 중간에 있는 하나의 단어에 대해 모든 시퀀스의 임베딩 빅터와 dot product, softmax 과정을 거쳐 구한 attention score에 weighted sum을 하여 문맥을 고려한 새로운 임베딩 벡터를 만들 수 있다. 이는 각 token을 sequence 내 모든 token과의 연관성을 기반으로 재표현하는 과정이다.

Ⅱ. Transformer



Self-attention 매커니즘을 극대화한 모델이 transformer이다. Transformer는 self-attention을 사용해 sequential한 토큰을 sequential하게 처리하지 않아 병렬처리를 가능하게 하였다. 기계 번역 task를 처리할 수 있는 transformer는 인코더 블록과 디코더 블록으로 구성되어 있다. 인코더 블록에는 6개의 인코더가 스택처럼 쌓여있다. 6개가 최적의 값은 아니다. 디코더 블록에도 디코더 6개가 스택처럼 쌓여있다.

인코딩 블록은 512개의 토큰 중 입력에 해당하지 않아 비어있는 토큰을 패딩으로 두고 시퀀스 단위의 개수의 토큰을 self-attention과 feed forward neural network에 한 번에 전달해 새롭게 표현된 임베딩 벡터를 출력하는 2단 구조로 구성되어있다. 디코딩 블록은 인코딩 블록의 output과 masked self-attention의 벡터를 입력으로 받아 self-attention을 진행하기 때문에 2단 구조가 아닌 3단 구조로 구성되어 있다. 또한 output은 인코딩 블록과 달리 sequential하게 생성되기 때문에 만약 5번째 토큰이 생성되어야 하면 5번째 토큰을 마스킹하는 추가적인 작업이 필요하다.



1. Transformer의 구조

- Input embedding : 단어를 512차원의 벡터로 변환시킨다. 첫번째 인코더 이후의 입력은 아랫단 인코더의 아웃풋이 되어 입력의 size는 512로 유지된다. Sequence의 길이는 가장 긴 문장의 단어 수 또는 상위 95%에 해당하는 토큰의 개수이다.

- Positional encoding : 한 번에 모든 sequence를 적용해서 생기는 단어의 위치 정보가 손실되는 단점을 보완한다. Input embedding과 positional encoding 벡터를 더한 값이 첫 번째 인코딩 블록의 입력으로만 쓰인다.

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

- Multi-head attention, residual connection & normalization : self-attention까지는 NxN 형태로 벡터들이 서로 영향을 끼치지만 feed forward neural network는 각 벡터마다 하나씩 존재해서 하나의 입력을 받아 하나의 아웃풋을 내놓는다. Self-attention은 한 토큰을 처리할 때 크 토큰을 제외한 나머지 다른 토큰을 얼마나 중요하게 볼 것인가를 정하는 레이어를 가리킨다.

2. Self-attention 과정

   [Step 1] self-attention은 인코더의 입력 벡터로부터 query, key, value 이렇게 세 개의 벡터를 만든다. Input embedding에 query matrix를 곱해 query를 만들고 key matrix에 key를 곱해 key를 만들고 value matrix를 곱해 value를 만든다. 각각의 matrix는 미지수로 학습을 통해 도출된다. 인코더의 입출력 벡터가 512차원인데 query, key, value는 64차원인데 이후에 multi-head attention 8개를 사용해 dimension을 concat하여 사이즈를 맞추게 된다.

   [Step 2] query와 문장의 어느 부분이 연관성이 높은지를 수치화한다. Query vector와 key vector를 곱해 각각의 단어에 대해 score를 계산한다.

   [Step 3] 각 score를 dimension의 제곱근으로 나누고 softmax operation을 통과시킨다. 이때 scaling을 하는 이유는 attention score를 다양한 vector에 분산시켜 gradient의 stability에 도움이 되기 때문이다.

   [Step 4] softmax와 value를 곱한다. Weighted value가 크면 단어의 영향력이 큰 것이다.

   [Step 5] 모든 value vector를 더하면 한 단어가 query일 때 self-attention output을 구할 수 있다.

3. Multi-head attention, residual connection & normalization

   한 문장 내에는 누가, 무엇을, 누구에게 등 다양한 정보가 존재하는데 이러한 정보를 한 번의 attention으로 적절히 반영하는 것은 어렵다. 하지만 하나의 attention이 문장 내에서 존재하는 한 가지 정보에 집중하게 된다면 8개의 head를 가진 attention을 사용해 정보를 적절히 attention score에 반영할 수 있다. 더불어 multi-head attention에서 FFNN의 input size(=4)를 맞춰주기 위해 추가적인 작업이 필요하다. 각각의 attention에서 나온 score를 concat한 attention score의 길이와 같은 column의 수를 가진 matrix를 곱해 (row는 input의 개수인 4) attention score를 FFNN의 input dimension과 일치시켜줘야 한다. 이 matrix는 모델의 학습 과정에서 학습되는 matrix이다. Multi-head attention 이후에는 attention output과 vector를 더해 layer normalization 과정을 거친 후 각각의 row를 feed forward neural network에 전달한다.

4. Position-wise Feed-Forward Networks

   Fully connected feed-forward network로 layer마다 다른 파라미터를 가진다. 같은 layer의 FFNN은 같은 가중치를 가진다.

5. Masked Multi-Head attention

   디코더는 현 query 앞 token의 attention score만 볼 수 있기 때문에 만약 현재 query보다 매우 작은 attention score값은 아주 작은 -inf 값으로 설정 후 계산하게 된다. 이를 통해 토큰들 사이에 인과관계를 부여할 수 있다.

6. Multi-head attention with encoder outputs
7. The final linear and softmax layer

   -linear layer : FFNN의 일종으로 vector를 펼치는 역할을 한다.

   -softmax layer : attention score를 probability로 바꿔줘 대응하는 단어를 return한다.

Ⅲ. Image Transformer

- Unconditional Image Generation : 대규모의 데이터로 특정한 이미지를 제작하는 태스크. 디코더만 사용.
- Class Conditional Image Generation : 클래스 각각의 임베딩 벡터를 입력으로 받아 이미지를 제작하는 태스크. 디코더만 사용.
- Super Resolution : 저화질의 이미지를 입력받아 고화질의 이미지를 출력하는 태스크. 인코더와 디코더 모두 사용.

Self-attention의 장점은 적은 연산량인데 이미지의 경우 dimension보다 sequence length, 즉 pixel의 길이 n이 커져 self-attention이 비효율적이게 된다.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$

Input	n	$O(n^2 \cdot d)$	
8x8 image	$8 * 8 * 3 = 192$	$192 * 192 * d$	Feasible
32x32 image	$32 * 32 * 3 = 3072$	$3072 * 3072 * d$	Infeasible

그래서 image transformer는 sequence 내 일정 부분 안에서만 self-attention을 적용하는 local self-attention을 사용한다. Input을 겹치지 않은 block으로 구분하면 현재 query 옆에 있는 칸이 target pixel이다. 하나의 블록을 둘러싸는 만큼을 memory block이라고 한다. Memory block 내 pixel을 key와 value로, query pixel을 query로 하는 self-attention은 linear projection, dot-product, softmax 과정을 거쳐 구현할 수 있다. Encoder-decoder의 attention과 FFNN을 거쳐 output이 생성되고 query block과 memory block을 재지정하고 과정을 반복함으로써 pixel by pixel로 image를 생성할 수 있다.

IV. Music Transformer

- Unconditional Music Generation : input sequence 없이 다양한 데이터를 통해 음악을 생성하는 것
- Conditional Music Generation : 음악의 앞 부분을 입력으로 받아 그 뒷부분을 생성하는 것

Music transformer는 Relative Positional Self-Attention을 사용한다. 일반적인 self-attention에 relative positional vector를 더해 query와 key의 sequence 내 거리를 attention weight에 반영한다.

$$Relative\ Attention = Softmax\left(\frac{QK^T + S^{rel}}{\sqrt{D_h}}\right)V$$

[Step 1] Relative positional embedding matrix를 만든다.

[Step 2] query와 matrix르 곱한다. q_2는 0, 1, 2번째 query인 the의 2 token 이전의 positional attention score를 의미한다. 그 다음 각 row 기준 값이 없는 것들을 masking 해준다. 이 방식은 query에 따라 positional attention score가 달라지므로 positional encoding보다 풍부한 표현을 할 수 있다는 장점이 있다.

[Step 3] Skewing : 기존의 attention score와 더할 수 있도록, 즉 마스킹한 모양이 삼각형 모양이 되도록 score를 reshape한다.

[Step 4] 기존 attention score에 relative positional attention score를 더하여 output을 산출한다.

Relative self-attention이 적용된 music transformer는 다양하고 반복적인 곡을 생성함과 동시에 training data보다 2배나 긴 sequence에 대한 generation 또한 가능하다는 것을 확인할 수 있다.