Transformer : Attention Is All You Need 논문 리뷰

Twobigs14 이혜린

논문에서는 Recurrent layers를 multi-headed의 self-attention로 대체한 첫번째 sequence transduction model인 **Transformer**를 소개하고 있다.

- 위 모델은 recurrent 나 convolutional layers를 기반으로 하고 있는 구조보다 훨씬 더 빠르다.
- 그 전의 recurrent model들은 메모리의 제한으로 인해 sequence가 길어질 때 문제가 발생했는데, 이 모델의 경우 계산 복잡성과 계산의 양이 적은 self attention 만을 기반으로 만들어진 모델이기 때문에 과거 연구들의 이러한 문제점을 개선한 모델이라고 할 수 있다.
- Encoder : multi-headed self-attention layer, forward layer (two sublayer로 구성됨)
- Decoder : multi-headed masked self attention layer – multi headed layer(encoder의 결과값을 input중 하나로 받는) – forward layer (3 sublayer로 구성됨)

[Positional encoding]

문장에서의 위치를 반영하는 임베딩. 같은 단어여도 문장에서 쓰인 위치에 따라서 임베딩 값이 달라진다. 이는 transformer는 cnn이나 rnn을 사용하지 않기 때문에 sequence의 order에 대한 정보가 누락될 수 있는데 이를 방지하기 위해서 사용하는 것이다. Positional encoding에 사용되는 함수는 여러가지가 있는데 이 모델에서는 sine과 cosine함수를 사용한다. (그 결과 encoding값이 [-1,1] 범위 내에 있다.) Encoder와 Decoder에 모두 사용된다.

[Multi head attention]

한 문장을 여러 head가 파트를 나눠 각 head의 관점에서 처리하고 결과를 합치는 방식. (= 한 문장을 여러 head에서 self-attention 시킨다.)

[Self attention]

Attention : softmax(query와 key의 내적값 / sqrt(key matrix의 dimension)) * value (→ dot product attention에서 key matrix의 dimension으로 스케일링을 추가한 결과).

Attention의 값이 클수록 단어와 해당 단어가 관련성이 높다.

In encoder,

- Query와 key와 value가 모두 인코더의 전 레이어의 결과에서 비롯된다.
- 현 인코더의 각각의 포지션은 인코더의 이전 레이어의 모든 포지션을 향할 수 있게 된다.

In decoder,

- Encoder에서와 마찬가지로 현 디코더의 각각의 포지션들이 그 포지션까지 모두 향할 수 있다. 그러나 자기회귀 성질(현재 값이 이전 값의 영향을 받는 것)을 보존하기 위해서 decoder에서는 masked self attention을 사용한다.

Self attention의 장점

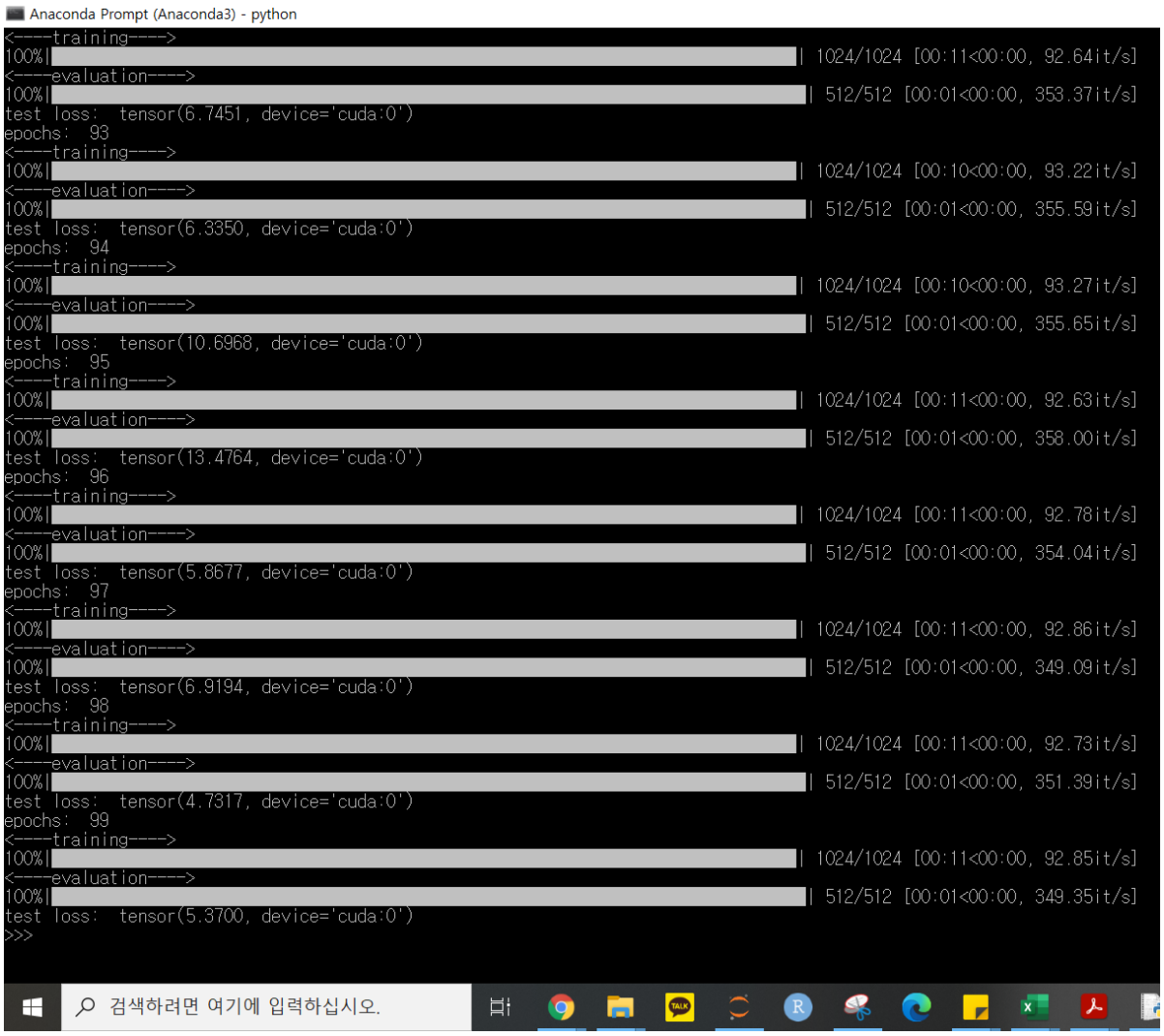
- 상수값의 operation으로 모든 position을 연결하기 때문에 layer당 총 계산 복잡성이 recurrent layer에 비해 적다.
- sequential operations (병렬화된 계산의 양)이 recurrent layer에 비해 적다.
- 좀 더 해석력 있는 모델을 생산할 수 있다.

[Masked attention]

디코딩에서의 self-attention에서만 적용되는 방식이다.

Transformer의 경우 attention을 이용하여 앞의 값과 뒤의 값을 모두 볼 수 있는데, 이 때 뒤의 값을 보면 디코딩에 영향을 미치기 때문에 뒤의 값을 보지 못하게 하기 위해서 masked attention을 실행한다.

이로 인해 i 위치에서의 예측값은 i 전의 위치들에 있는 이미 알려진 결과값에 대해서만 의존하게 된다.


Anaconda Prompt (Anaconda3) - python
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.64it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 353.37it/s]
test loss: tensor(6.7451, device='cuda:0')
epochs: 93
<----training---->
100%| | 1024/1024 [00:10<00:00, 93.22it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 355.59it/s]
test loss: tensor(6.3350, device='cuda:0')
epochs: 94
<----training---->
100%| | 1024/1024 [00:10<00:00, 93.27it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 355.65it/s]
test loss: tensor(10.6968, device='cuda:0')
epochs: 95
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.63it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 358.00it/s]
test loss: tensor(13.4764, device='cuda:0')
epochs: 96
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.78it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 354.04it/s]
test loss: tensor(5.8677, device='cuda:0')
epochs: 97
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.86it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 349.09it/s]
test loss: tensor(6.9194, device='cuda:0')
epochs: 98
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.73it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 351.39it/s]
test loss: tensor(4.7317, device='cuda:0')
epochs: 99
<----training---->
100%| | 1024/1024 [00:11<00:00, 92.85it/s]
<----evaluation---->
100%| | 512/512 [00:01<00:00, 349.35it/s]
test loss: tensor(5.3700, device='cuda:0')
>>>
검색하려면 여기에 입력하십시오.