<https://ratsgo.github.io/natural%20language%20processing/2019/09/11/xlnet/>

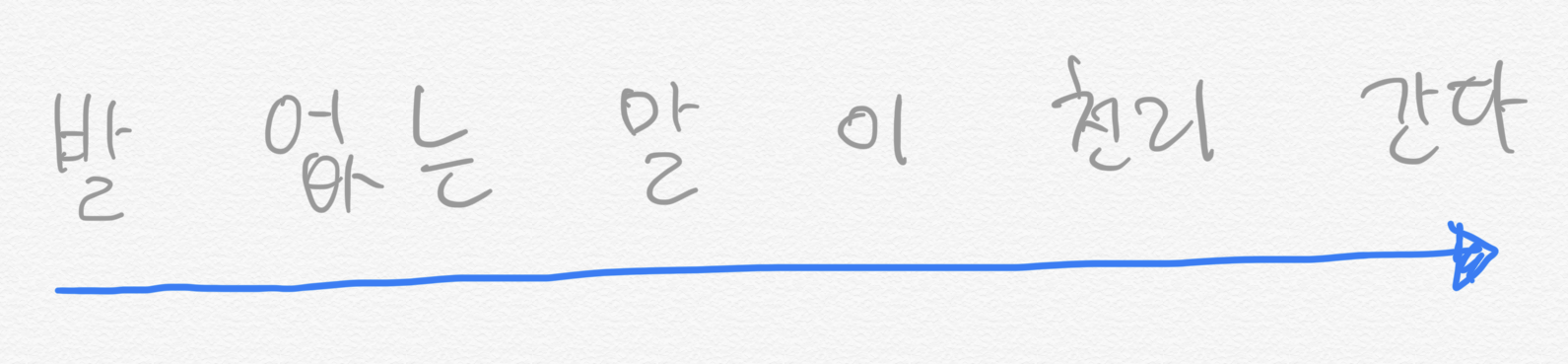
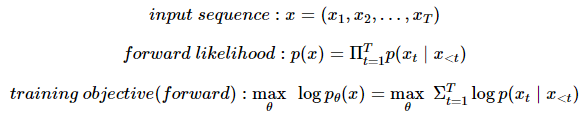
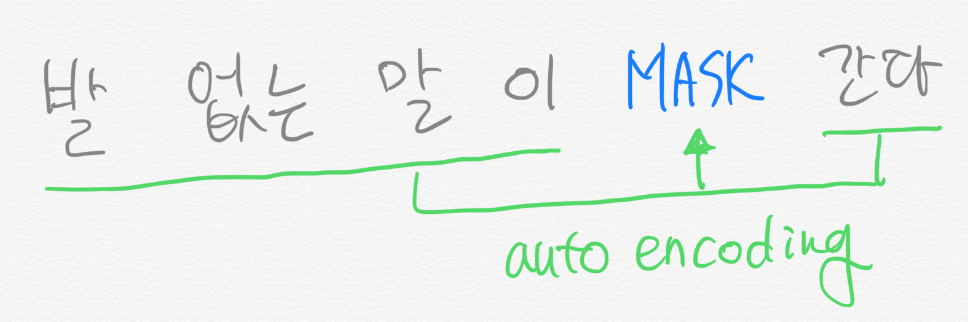
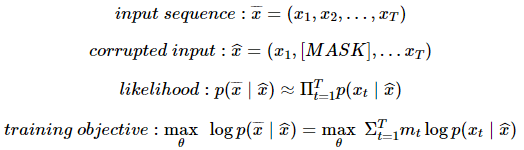
<https://blog.pingpong.us/xlnet-review/>

**XLNet(2019, Yang, google)**

공개 당시 20개 데이터셋에서 최고 성능, 기존 강자인 BERT보다 크게 앞섬.(일부 데이터셋에서는)

트랜스포머를 개선한 트랜스포머-XL의 확장판(eXtra-Long), 기존 트랜스포머보다 좀 더 넓은 범위의 문맥을 볼 수 있다.(트랜스포머-XL은 이 점을 강조)

임베딩 모델의 최근 흐름 크게 두 개!

1. AutoRegressive  
     
   ELMo나 GPT를 여기에 분류할 수 있다.  
   두 모델 모두 이전 문맥을 바탕으로 다음 단어 예측하는 과정!  
     
   
2. AutoEncoding(입력과 출력이 같다?!)  
     
   BERT가 대표적, 일부에 마스킹(노이즈)을 주어 문장을 원래대로 복원!  
   = 실제로 어떤 단어일지 맞추는데 중나점을 둔다.  
   이런 맥락에서 BERT를 Denoising Autoencoder라고 표현한다.(노이즈가 포함된 입력을 받아 해당 노이즈를 제거한 원본 입력을 출력으로 한다.)  
     
     
   mt는 이 토큰이 mask라면 1로 두고 나머진 0으로 하여 이 경우에만 objective가 동작하도록 한다. -> 따라서 전체 문장을 복원하는 것처럼 보이지만 사실은 mask 단어에 대해서만 복원하는 것이다.(착시 효과, 하지만 일부 포함 개념이라 보면됨.)

AR -> 문맥을 양방향으로 볼 수 없다.

AE -> BERT는 양방향이다. 그림 2처럼 앞뒤 문맥을 모두 살피기 때문이다. = 덕분에 downstream task에서 상당 기간 절대 강자  
하지만 큰 문제는 마스킹 처리한 토큰들을 서로 독립이라고 가정한다는 점이다.

Likelihood 식을 보면 corrupted input이 주어졌을 때 각 timestep의 곱 = 독립적인 분포 -> 마스크가 여러 개 있을 때 그들 사이의 dependency는 학습 불가

AR 방식은 independent assumption이 없다.

더군다나 fine-tune 시에는 masking을 하지 않는데, 이 달라진 방식의 학습 방법은 일반화를 낮출 수 있다.

마지막으로 BERT는 긴 문맥을 학습하기 어렵다고도 한다.

따라서 이 두 방식에 대한 단점을 극복하기 위해 permutation LM 제시.

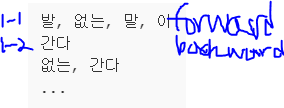
간단히 말하면 토큰을 랜덤으로 셔플한 뒤 그 뒤바뀐 순서가 마치 원래 그랬던 것인 양 언어 모델을 학습하는 기법이다.

Permutation 언어모델?

-> 특정 토큰을 제외한 문장의 부분집합 전체를 학습할 수 있다. “발 없는 말 이 천리 간다.”를 permutation 하면?

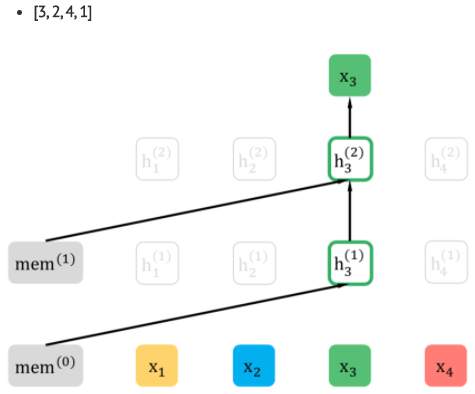
없는 이 말 발 간다 천리(입력 시퀀스: 없는 ~ 간다) / 발 없는 천리 이 말 간다(입력 시퀀스: 발 ~ 없는) ... 등이 나올 수 있다.

또한, 다음과 같이 양방향 AR 학습 방식이 이루어진다.

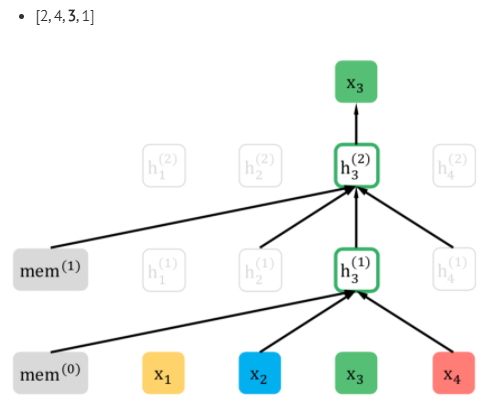
따라서 AR 모델이지만 permutation을 통해 양방향 문맥을 고려할 수 있고 AE 모델이 아니기 때문에 그 단점을 극복할 수 있다.(진짜 backward는 아니고 저런 샘플이 자동적으로 생기기 때문에 가능)

즉, BERT에서 마스킹만 함으로써 생기는 독립적인 부분을 보완하며, pretrain-finetune 사이의 불일치도 해결

XLNet Flow를 해보겠다.



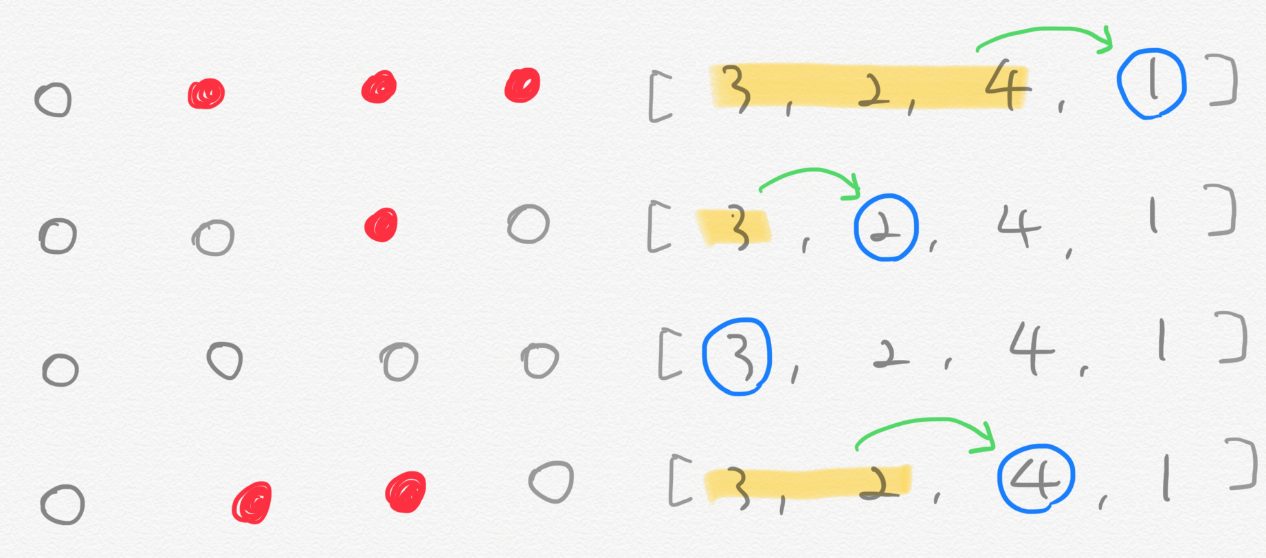
위 그림은 3, 2, 4, 1로 셔플을 하고 우리가 맞출 예측 토큰은 3번 토큰이라 하자. 하지만 이 토큰은 모델이 알면 안되니까 시퀀스로 주면 안된다. 따라서 처음 timestep 때는 시퀀스에 토큰이 들어갈 수가 없다. 따라서 세그먼트의 메모리 정보만 주면 된다.(이 정보는 트랜스포머-XL)과 관련



다음 permutation 에서는 3번토큰이 2,4 뒤에 있다. 따라서 입력 시퀀스에는 메모리, 2, 4 정보가 주어진다.

이런 식으로 모든 permutation을 돌게 된다.

기존에 트랜스포머에서 디코더에 마스킹을 처리해 주었다.



이번엔 그 마스킹을 각 permutation마다 위에 처럼 해줘야 제대로 작동한다. 하지만 아래와 같은 경우 문제가 발생한다.

처럼 두 permutation이 있을 때 똑 같은 3, 2가 주어진 상태에서 다른 4, 1을 예측해야 하는 이상한 상황이 발생한다. => two-stream self attention 제안했다.

**투-스트림 셀프 어텐션**

쿼리 스트림과 컨텐트 스트림을 혼합했다.

컨텐트 스트림은 기존 트랜스포머 네트워크와 거의 유사하다.

컨텐트 스트림은 t시점 이후에 해당하는 토큰을 예측하기 위해 어텐션이 t 시점의 content를 포함하도록 하는 장치?

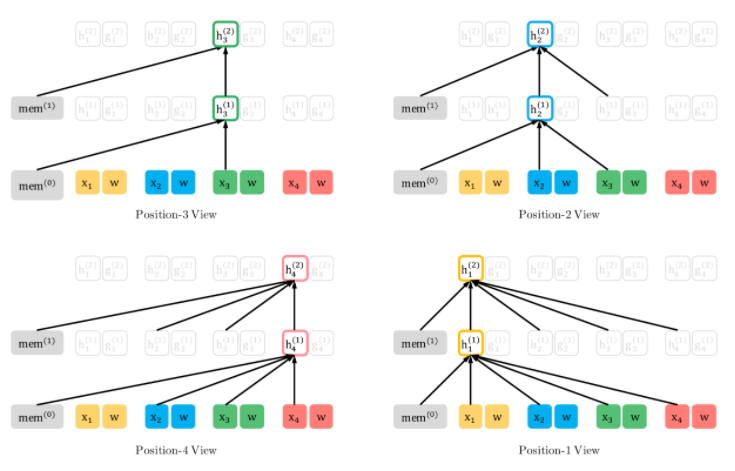
[3,2,4,1]이 셔플 샘플 하나라 하고 = z라 하고, 컨텐트 스트림 벡터 = h라 하자.  
z-t는 이때 샘플의 인덱스 성분이다.



처럼 m 번째 트랜스포머 블록의 컨텐트 스트림 벡터는 m-1 번째 컨텐트 스트림 벡터에서 Q, KV를 찾아 기존 트랜스포머 블록과 동일한 어텐션 계산을 수행

하지만 이전과 다르게(쿼리 스트림과 다르게) 예측 토큰이 3번이라 할 때 자기 자신의 토큰 정보를 빼지 않는다. => 이 방식이 만약 누적되어 블록 layer를 거쳐 적용된다면 x3 이후 등장하는 토큰들을 예측할 때(x3 예측하는거 상관x??) x3 정보를 참고하기 어렵다.

=> 따라서 컨텐트 스트림은 자기 토큰 정보를 추가한다.



즉 다음과 같이 간략화 가능 => 자기 자신 토큰을 포함한 이전 토큰 활용

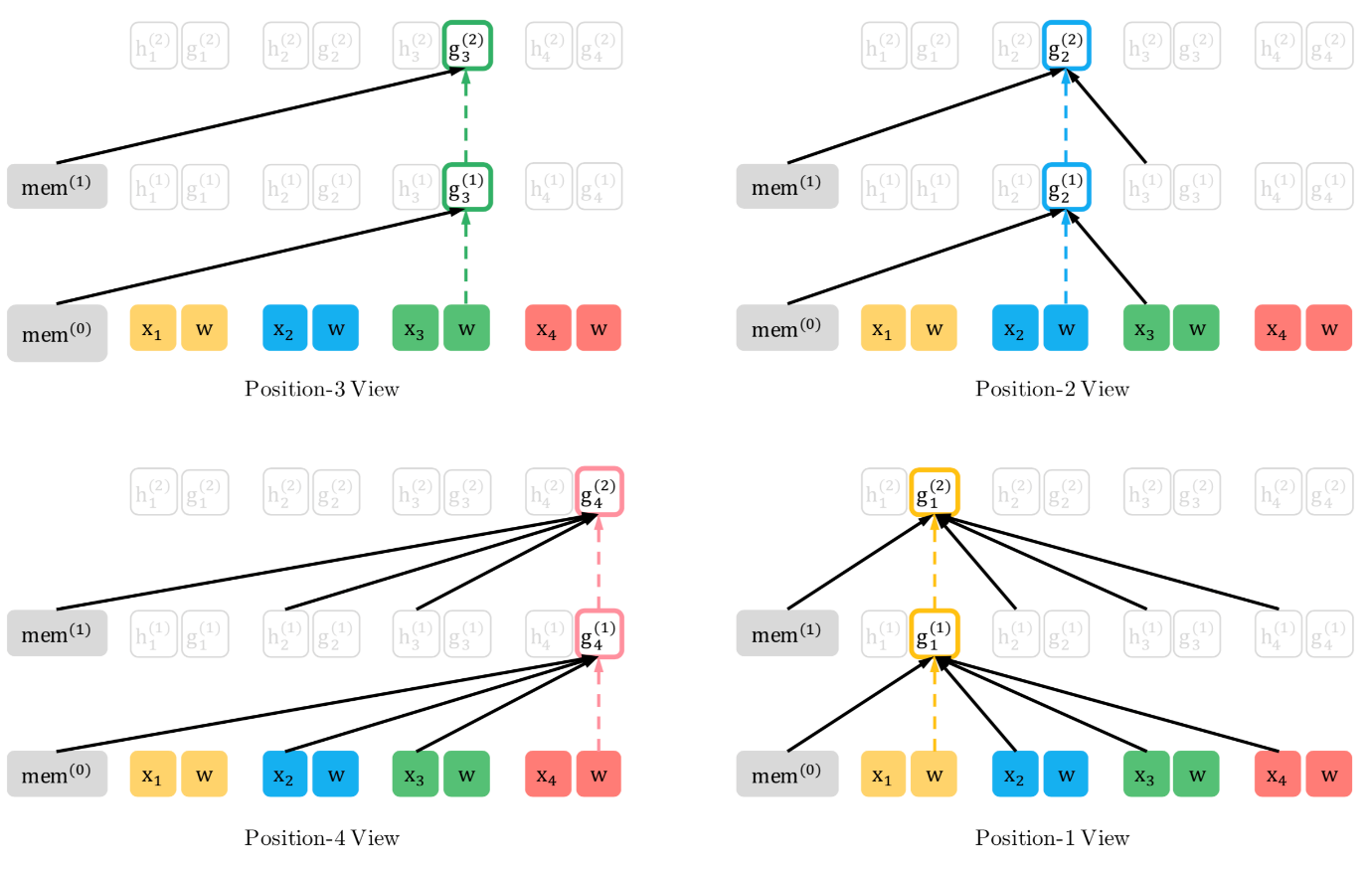
질문..-> 컨텐트 스트림, 쿼리 스트림의 경우의 수 \* permutation \* 예측 토큰 -> 너무 많음?

이렇게 한 종류의 어텐션을 포괄하는 스트림이 끝났다. 또 다른 어텐션을 진행하는 쿼리 스트림을 살펴보자.

뭔가 컨텐트 스트림은 셀프 어텐션을 위한 자료 준비??? KV 준비 느낌이라 자기 토큰 정보를 넣지만 이제 예측을 위해서 쿼리 스트림을 결합할 때는 자기 토큰 정보 중 위치 정보만 넣는 건가 싶다.(답을 몰라야 예측하는 의미가 있으니까)



현 블록의 쿼리 스트림 벡터를 계산할 때는 t번째 미만 컨텐트 스트림이 KV로, 쿼리에는 현 시점 토큰 정보를 빼고 위치 정보만 넣는다. 즉, Xz-t(현 시점 임베딩 정보) 가 애초에 들어가지 않는다.

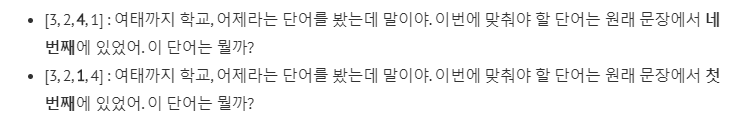


만약 예측 토큰이 [3,2,4,1]에서 1이라 할 때 ‘나’라는 것은 안주고 위치 정보인 1만 계속 전달하게 되는 것이다. 즉 해석을 하면은

“여태까지 학교, 어제, 갔어 라는 단어를 봤고 이번에 맞춰야 할 단어는 원래 문장에서 첫번째에 있었어. 이 단어는 뭘까?”라고 모델이 스스로 생각하는 것이다.

즉 permutation 모델은 예측할 때에 쿼리 스트림을 결과적으로 이용하는데 자기 자신 토큰 정보를 빼야 한다.

앞서 말했던 동일한 입력을 받아 다른 출력을 내는 모순을 위치 정보를 통해 해결했다.



즉 쿼리 스트림 과정을 간략화 하면 다음과 같다. 

하지만 여기서 아까 질문 했듯이 많은 경우의수 -> 느린 수렴 -> optimization difficulty 극복 위해서 특정 순서에서 마지막 몇 개의 예측만 사용한다.

사실 뭔소린지 이해 잘 안감.

이 아래 부분은 여기서 Transformer-XL을 어떻게 이용했나? => 아직 안 읽어 봤지만 맛보기로

XLNet은 긴 문장 처리를 위하여 Transformer-XL에서 사용된 2가지 테크닉 차용.

1. **Relative Positional Encoding**
2. **Segment Recurrence Mechanism**

**Relative Positional Encoding**

기존 Transformer -> CNN or RNN 처럼 단어들의 상대적 or 절대적 위치 정보를 직접적으로 모델링하지 않는다. 대신에 input의 절대적 위치에 대한 representation(absolute positional encoding)을 추가하는 방식으로 순서에 대한 모델링!

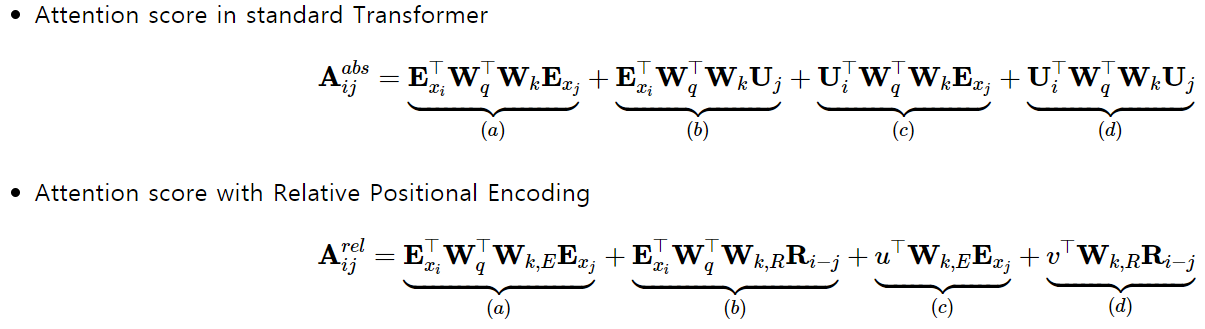
-> but 하나의 segment(문장?) 내에서는 위치 표현 가능하지만, 여러 segment(여러 문장?)에 대해서는 (Transformer-XL 처럼) recurrent 모델링을 하는 경우 문제가 있다.

Segment-level Recurrence 식 

는 segment 순서, E는 인풋 문장 s의 word embedding, f는 transformation function.

즉 이전 세그먼트 문장을 통해 hidden state를 구함. 하지만 다른 세그먼트인데 U는 뭐길래 같은가? U는 일단 위치를 나타내는 값이다.

예를 들어 과거 segment가 [0,1,2,3]이라는 position 정보, 현재 segment가 과거 정보를 포함해 [0,1,2,3,4,5,6,7] 이라는 position 정보라 할 때 위의 식을 적용하면 [0,1,2,3,0,1,2,3]을 이용 => multiple segments의 absolute positional encoding 문제 해결 => input-level이 아닌 self-attention mechanism에서 relative positional encoding이라는 단어 간의 상대적 위치 정보를 모델링!



B,d 에서의 U를 R로 대체 -> R은 learnable params 아닌 sinusoid encoding mat이다.

C,d 에서 UW가 u,v로 각각 대체 -> Query vector가 모든 query position에 대해 같기 때문이다.  
다른 단어들에 대한 attention bias가 query position에 상관없이 동일하게 유지

W-k를 W-k,E와 W-k,R로 분리, content 기반의 key vector와 location 기반의 key vector를 각각 만들기 위해서

A -> content 기반의 처리 / b -> content에 의존한 positional bias 잡음 / c-> global content bias를 잡음 / d -> global positional bias 인코딩

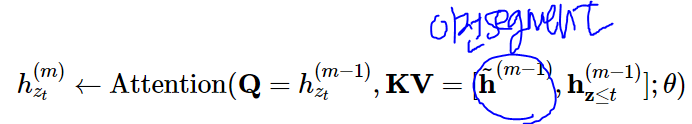
**Segment recurrence Mechanism**

Segment로 분리하고 recurrent하게 모델링하는 Transformer-XL 이 기술을 XLNet에 적용하기 위해 1) permutation setting에 어떻게 적용? 2) 이전 segment의 hidden state를 어떻게 재사용? 이 논점.

: 두 segments

: 두 permutations

일단 앞의 segment를 처리하여 content representation 얻고(caching) 그 다음 두 번째 segment 처리



저 부분이 caching 되었기 때문에 reusing 가능. 쿼리 스트림도 마찬가지

이번엔 어떤식으로 학습 했는지(multiple segment에 대해서 autoregeressive 하게)

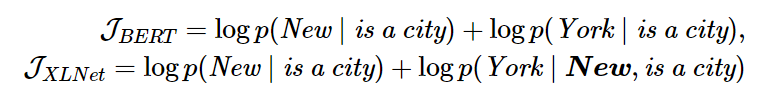
BERT와 유사하게 [A, sep, b, sep, cls]-> 같이 들어가게 될 문장은 랜덤 샘플링 -> 두 문장 concat 후 permutation

XLNet-Large는 next sentence prediction 사용 x / masking 사용 x

BERT는 absolute segment embedding, XLNet은 relative position encoding, position i, j 가 같은 segment라면 sij = s+, 아니면 sij=s\_를 사용, s+,s\_는 각 attention head에 존재하는 learnable params

=> relative encoding의 inductive bias가 일반화 향상, 둘 이상의 segment에 대한 fine-tuning task 가능성 열어줌.

**Discussion**



BERT와의 차이점이다. NewYork이 토큰화 되어 New, York 전부 masking 됐다(토큰 단위로 masking 해서 new만 마스킹 되는 논문도 있다.) => 따라서 위와 같은 식이 나오고, New와 York은 마스킹 되어 그 사이의 dependency는 잡을 수 없다.

반면에 XLNet은 masking이 아닌 AR 방식이라 가능하다. (단 현재 factorization order는 is a city new York 이라 가정)

질문 -> factorization 단위가 BPE 단위(subword)인가 아니면 word 단위인가?

BERT와 XLNet은 미래 context인 is a city 사용 but GPT(2018)은 단방향 AR이라 불가.

ELMo는 양방향 같지만 찐 양방향이 아님(shallow, deep interaction 학습 어렵)

**Experiments**

Google의 SentencePiece tokenizer를 이용해 BooksCorpus, English Wikipedia, Giga5, Clue Web2012-B, Common Crawl dataset 사용 -> token은 각각 2.78B, 1.09B, 4.75B, 4.3B, 19.97B 개 있고, 총 32.89B token으로 pre-training!

질문: 임베딩 할 때 BPE의 토큰? 여기서의 이 토큰? 뭐가 다른 거지?

+ multitask로 학습하는 방법?  
GLUE dataset은 9가지 natural language understanding에 관련된 테스크 집합입니다. Single-task learning을 통한 결과 뿐 아니라, multi-task learning에 대한 결과도 공개하였습니다. MNLI, SST-2, QNLI, QQP, 총 4개의 가장 큰 dataset으로 multi-task learning을 진행하였고, 나머지 dataset에 대해서 fine-tuning을 하였습니다. 전체 9개의 테스크 중 7개에서 state-of-the-art 성능을 얻을 수 있었습니다.

결과는 블로그 보기

