5.1. Transformer-XL (Transformer eXtra-Long)

논문 : https://arxiv.org/abs/1901.02860

스터디자료 : https://docs.google.com/presentation/d/1DxngNiNGG1esu5MZWMFnws3ooRayAvCgIQVzVyb81Zw/edit?usp=sharing

Transformer의 단점

Transformer 모델은 세그먼트 내에서의 Attention만이 가능함

=> 개발자/연구원이 지정한 최대 세그먼트 길이를 초과하는 문장이 입력되는 경우 문장을 나누어 입력으로 사용해야함

Transformer-XL은 Segment recurrence(어텐션의 Q,K,V 입력단 수정)와 Novel positional encoding scheme(dot-product Attention 중 score 연산인 QK^T 수정)를 제안함.

Segment recurrence

해당 기법의 요약

(문제)제안된 세그먼트 길이를 확장하고싶었음.

(제안)recurrent 모델처럼 과거 세그먼트의 히든값을 연산에서 활용하자!

(적용)Attention의 입력 중 K와 V에 과거 세그먼트의 히든 값을 Concatenate하는 것으로 끝!

기존 Transformer : 오토 리그레스브 모델로 최대 길이는 인공지능 개발자가 정해놓음 (문장 생성의 종료지점을 만들기 위함)

기존 Transformer의 단점인 고정된 길이의 문맥 정보만 활용할 수 있음. => Recurrent 모델들 처럼 과거정보를 현재에 전달해주자!

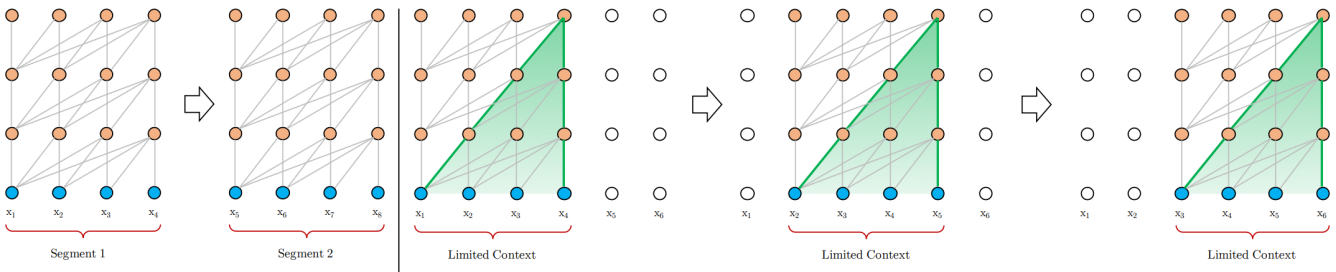

그림 1. 기존의 Transformer의 학습과 연산과정. 세그먼트가 4로 고정인 경우 길이가 6인 문장이 입력됐을 때 문제가 발생함

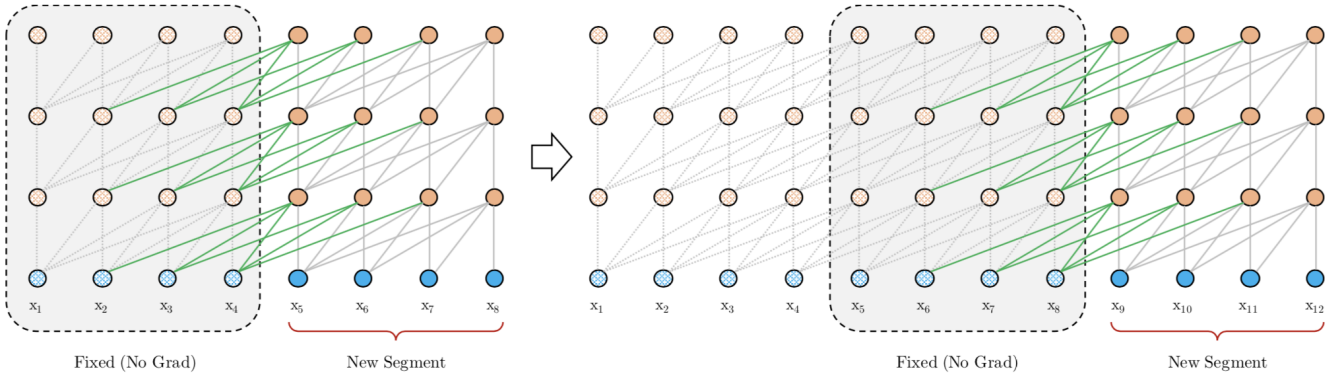

그림 2. Transformer-XL 학습 단계 시각화. 현재의 세그먼트를 계산할 때 과거의 히든값을 같이 활용하는 모습. 학습에 영향을 주면 안되기 때문에 과거의 히든값에 Stop Gradient를 지정한 상태.

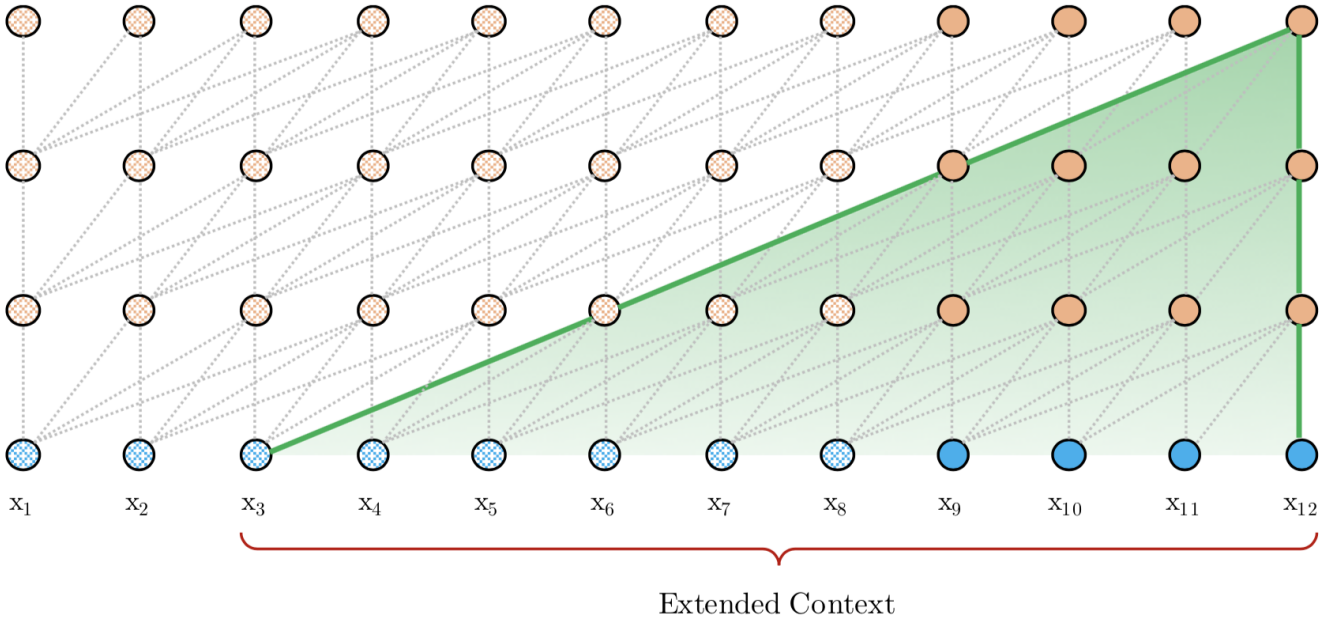

그림 3. Transformer-XL 평가 단계 시각화. 과거의 세그먼트를 연산에 사용함으로써 고정된 세그먼트보다 더 확장된 컨텍스트를 입력으로 사용하는 효과를 가짐. GPU가 넉넉하다면 과거'들'의 세그먼트도 사용할 수 있음.

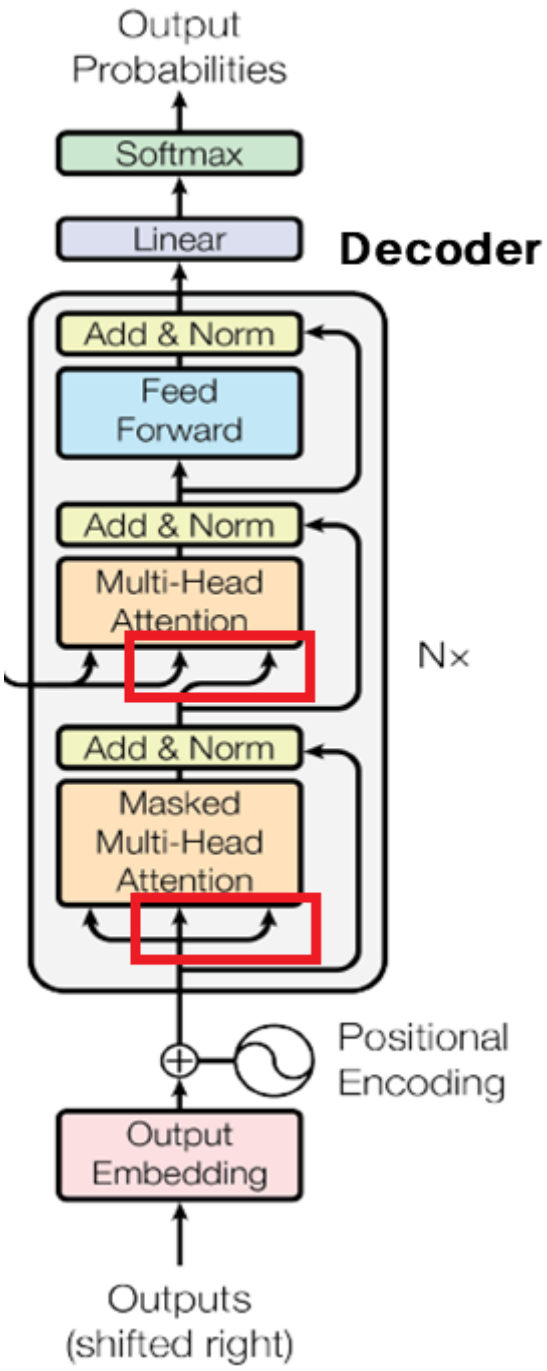

그림 4. Transformer에서 변경되는 부분 (빨간 박스). 해당 기법을 적용하려면 과거 세그먼트의 히든값을 저장하는 것과 그 히든값이 그라디언트에 영향을 주지않도록 해야함.

수식으로 표현

논문이 제안한 self-attention 수식

$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$
$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$
$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

n : Layer의 층수

τ : segment 수
h^T : 길이가 L인 input $[x^1, x^2 .. x^L]$

h^{T+1} : h^T의 뒤에 이어진 길이가 L인 input $[x^{L+1}, x^{L+2} .. x^{L+L}]$
SG : stop-gradient
[h_u ∘ h_v] : 두 hidden state를 concatenate

Relative position embedding

해당 기법의 요약 : (문제) Segment recurrence를 쓸껀데 기존 Transformer의 Position Encoding이 무용지물

(제안) 마지막 토큰을 기준으로 얼만큼 떨어져있는지 상대적인 위치값을 사용하자!

(적용) Attention에서 Score를 연산할 때, 기존의 Key를 컨텍스트에 대한 Key로 지정하고 포지션에 대한 Key를 추가, 마지막 세그먼트의 위치값은 변하지 않으니까 trainable vector 추가, Global Position 대신 Relative Position을 사용.

기존의 Transformer의 position encoding은 하나의 sentence에서 각 token들의 위치를 지정함

원래 문장을 작은 세그먼트들로 쪼개고, 세그먼트별로 트랜스포머 네트워크를 학습하면 각 단어가 문장 내에서 차지하는 절대위치정보가 무의미해지기 때문에 단어 쌍 사이의 거리정보인 상대위치(relative position)을 활용함

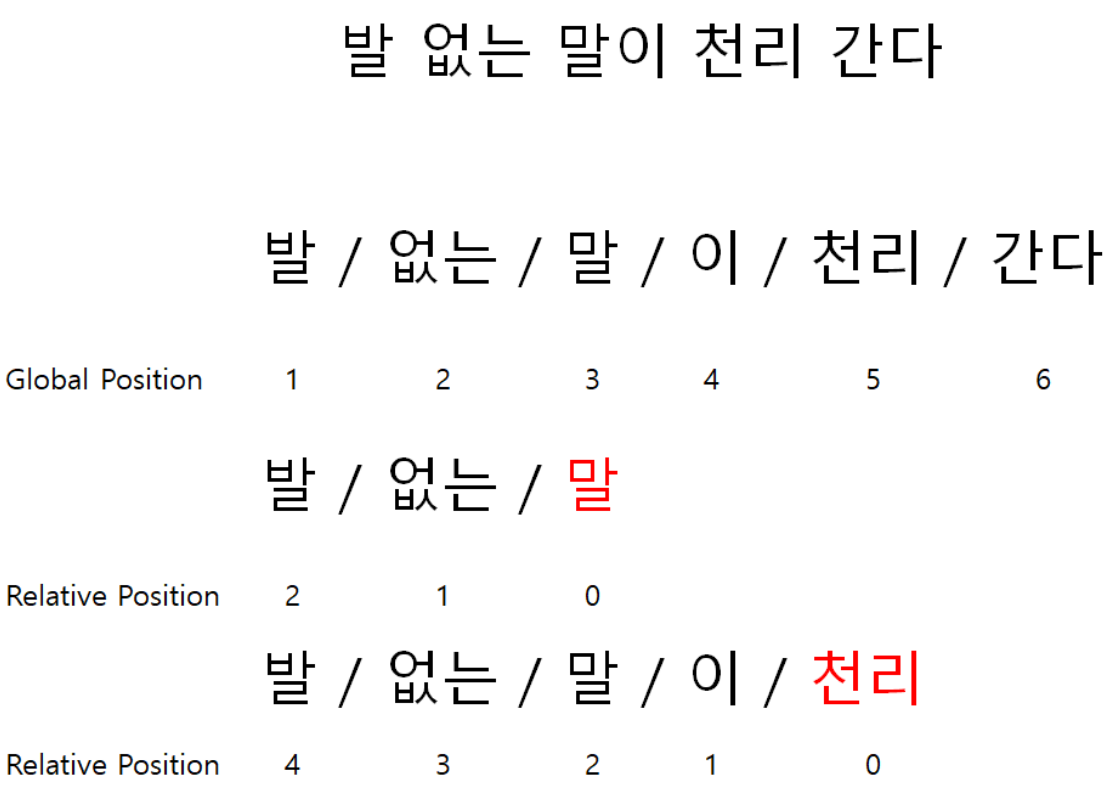

그림 5. Global Postion과 Relative Postion의 차이. Global Position은 세그먼트의 첫번째부터 점차 위치값이 증가함. Relative Position은 세그먼트의 마지막 token을 기준으로 얼만큼 떨어져있는지 상대적인 위치를 다른 토큰에게 부여함

'천리'와 '간다'는 -1? NO!

=> Transformer-XL은 단어를 순차적으로 학습하는 Autoregressive Model이기 때문에 미래에 출현할 단어는 학습에 사용하지 않음

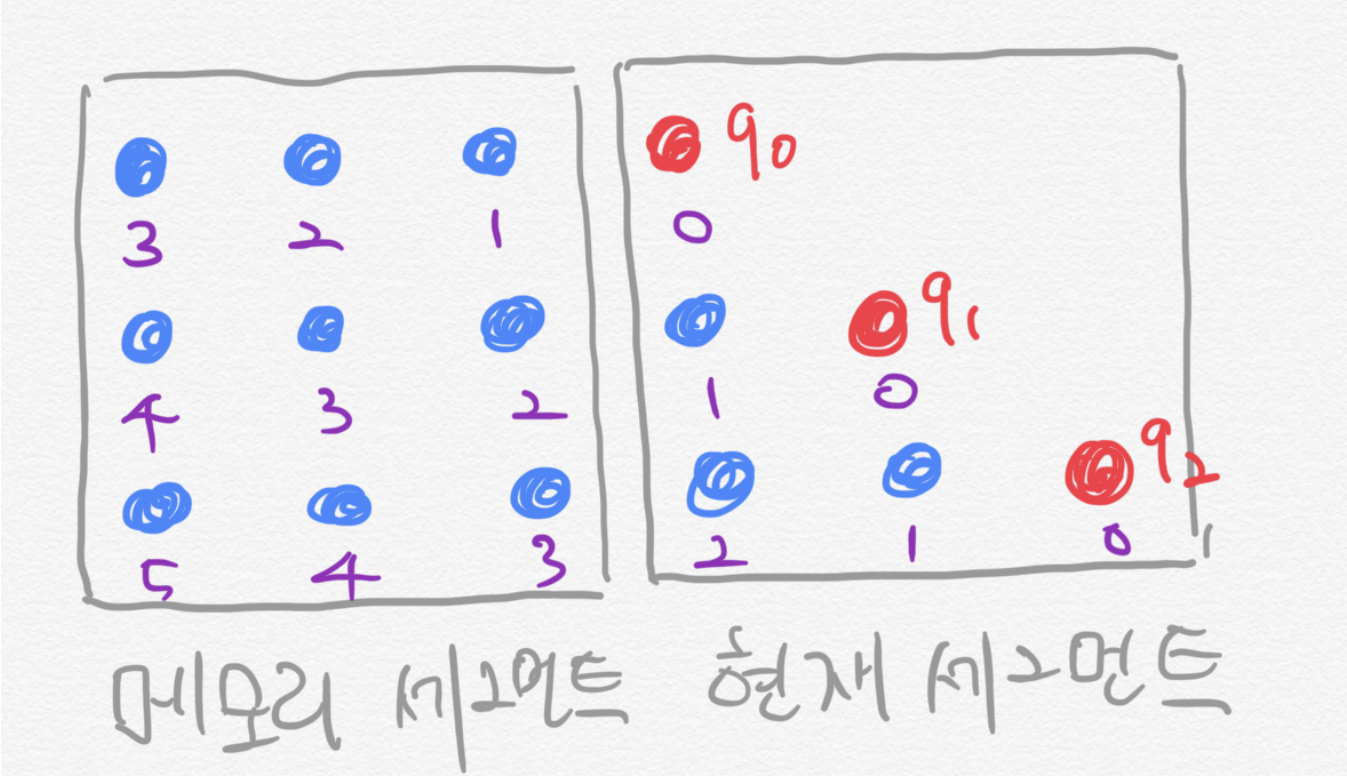

그림 6. 메모리세그먼트(과거의 세그먼트)와 현재 세그먼트의 Relative Postion 설정

논문이 사용한 Relative Position 계산 방법은 기존의 Transformer 모델의 Position Encoding과 동일한 함수 사용 (마찬가지로 자유롭게 변경가능)

수식으로 표현

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

기존의 QK^T를 상세히 표현

$Q = W_Q (E_{x,i} + U_i)$

$K = W_K (E_{x,j} + U_j)$

E : 워드 임베딩

U : 포지션 인코딩

Transformer의 Q^TK 를 수식으로 표현

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

Transformer-XL의 제안한 방법

$$\mathbf{A}_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

1. 상대 위치를 쓰기 때문에 U 대신 R로 대체한다
   R : 기존의 Transformer와 동일하게 사인과 코사인 함수를 활용한 non-trainable parameter
2. 상대 위치를 쓰기 때문에 Query의 위치는 항상 0 이므로 trainable parameter인 u와 v로 대체한다
3. Wk 대신 Wk,E와 Wk,R로 대체한다
   Wk,E : context에 대한 Key vector 연산을 위한 가중치 행렬
   Wk,R : position에 대한 key vector 연산을 위한 가중치 행렬

논문이 제안한 Attention 연산을 수식화

$$\begin{aligned}
\widetilde{\mathbf{h}}_\tau^{n-1} &= \left[\text{SG}(\mathbf{m}_\tau^{n-1}) \circ \mathbf{h}_\tau^{n-1}\right] \\
\mathbf{q}_\tau^n, \mathbf{k}_\tau^n, \mathbf{v}_\tau^n &= \mathbf{h}_\tau^{n-1}\mathbf{W}_q^{n\top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_{k,E}^{n\top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_v^{n\top} \\
\mathbf{A}_{\tau,i,j}^n &= \mathbf{q}_{\tau,i}^{n\top}\mathbf{k}_{\tau,j}^n + \mathbf{q}_{\tau,i}^{n\top}\mathbf{W}_{k,R}^n\mathbf{R}_{i-j} \\
&\quad + u^\top \mathbf{k}_{\tau,j} + v^\top \mathbf{W}_{k,R}^n \mathbf{R}_{i-j} \\
\mathbf{a}_\tau^n &= \text{Masked-Softmax}(\mathbf{A}_\tau^n)\mathbf{v}_\tau^n \\
\mathbf{o}_\tau^n &= \text{LayerNorm}(\text{Linear}(\mathbf{a}_\tau^n) + \mathbf{h}_\tau^{n-1}) \\
\mathbf{h}_\tau^n &= \text{Positionwise-Feed-Forward}(\mathbf{o}_\tau^n)
\end{aligned}$$