5.2. XLNet

논문 : http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding

발표자료 : GPT, GPT2, Transformer-XL, XLNet 발표자료

Transformer-XL = Transformer + 더 넓은 범위의 문맥

XLNet = Transformer-XL + Permutation Language Model

Autoregressive VS Auto Encoding VS Permutation Language

Autoregressive Model

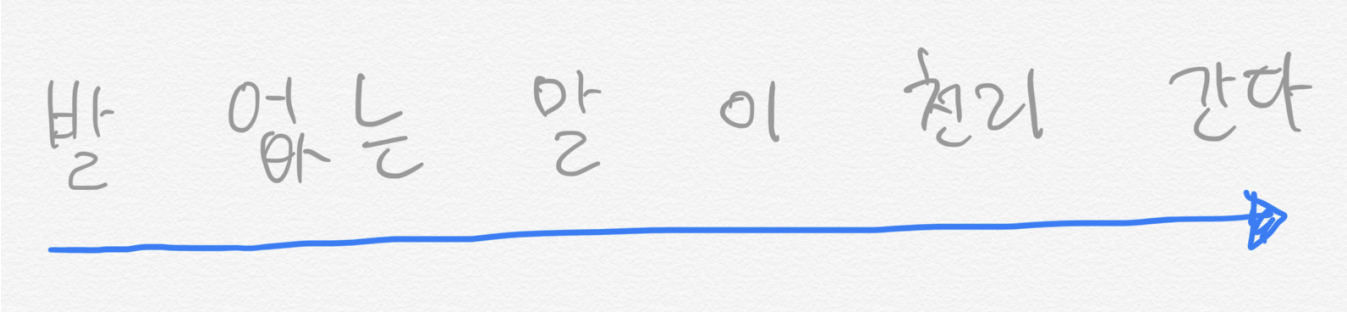

그림 1. AutoRegressive Model의 token 읽는 방향

Autoregressive Model : 이전의 token들을 보고 다음 token을 예측하는 문제를 푸는 모델 (NLP 기준)

대표적인 모델 : GPT, ELMO

단점 : 맞춰야할 단어들을 포함한 문맥정보를 줄 수없기 때문에 문맥을 양방향(bidirectional)으로 볼 수 없는 한계가 있음.

수식으로 표현

Autoregressive model의 Likelihood와 Objective

$$input\ sequence : x = (x_1, x_2, \ldots, x_T)$$

$$forward\ likelihood : p(x) = \Pi_{t=1}^{T} p(x_t \mid x_{<t})$$

$$training\ objective(forward) : \max_{\theta}\ \log p_\theta(x) = \max_{\theta}\ \Sigma_{t=1}^{T} \log p(x_t \mid x_{<t})$$

Likelihood : 단방향(forward, backward)의 conditional probability(조건부 확률)들의 곱

Objective : Likelihood의 conditional distribution(조건부 분포)를 학습 (negative-log likelihood)

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)},$$

h : Transformer, RNN과 같은 context representation

e : word embedding

Auto Encoding Model

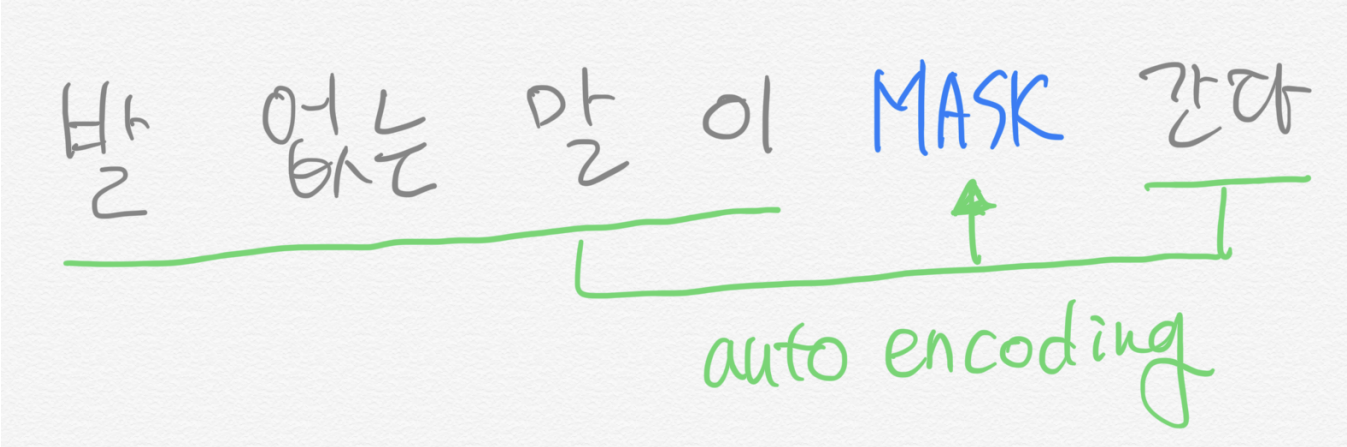

그림 2. AutoEncoding Model의 token 읽는 방향

AutoEncoding Model : 입력값을 복원하는 기법 $y = f(x) = x$

대표적인 모델 : BERT 류 모델

논문에서는 Denoising Autoencoder라고 표현함

단점

1. BERT모델이 사용하는 '<MASK>' 토큰은 서로 독립이라 가정하고 학습하기 때문에 마스킹 토큰들 사이에 의존 관계를 따질 수 없음.

그림 3. AutoEncoding Model로 문장 'New York is a city' 을 학습하는 과정 시각화.

BERT 모델은 두 마스크 간의 선후 관계나 등장 여부와 같은 정보를 따지지 않음.

=> 문장 'is a city'을 보고 'New'와 'York'을 독립적으로 예측함

2. pre-train 과정에는 <MASK> 토큰을 사용하나 fine-tuning(또는 warm-traning) 에서는 <MASK> 토큰을 사용하지 않음

수식으로 표현

Auto Encoding model의 Likelihood와 Objective

$$input\ sequence : \overline{x} = (x_1, x_2, \dots, x_T)$$

$$corrupted\ input : \hat{x} = (x_1, [MASK], \dots x_T)$$

$$likelihood : p(\overline{x} \mid \hat{x}) \approx \Pi_{t=1}^{T} p(x_t \mid \hat{x})$$

$$training\ objective : \max_{\theta}\ \log p(\overline{x} \mid \hat{x}) = \max_{\theta}\ \Sigma_{t=1}^{T} m_t \log p(x_t \mid \hat{x})$$

m_t : <MASK> 토큰이면 1, 아니면 0

Likelihood : Corrupted input(기존 토큰이 <MASK> 토큰으로 바뀐 입력)들로 input sequence의 conditional probability들의 곱

Objective

- Likelihood를 maximize하는 것. 단, <MASK> 토큰에 한정됨
- Independent assumption : 원래는 <MASK> 토큰의 정답 토큰이 등장할 확률은 독립이 아니지만 독립으로 가정해서 확률곱으로 나타냄 ('New York is a city'의 예)

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)},$$

H : Transforemr의 Hidden Vectors

m : t번째 토큰의 마스킹 여부(<MASK> 이면 1, 아니면 0)

e : word embedding

$\bar{\mathbf{x}}$: <MASK> 토큰

$\hat{\mathbf{x}}$: Corrupted input set

Permutation Language Model (순열 언어 모델)

Autoregressive model은 한쪽 방향의 정보만 이용해야되는 문제가 있고,

Auto Encoding model은 <MASK>간의 연관관계 추정도 못하고 fine-tuning 과정에선 <MASK> 토큰을 쓰지도 않는 문제가 있다!

=> 그러면 각 모델의 장점을 짬뽕해서 단점을 없애자!

=> 문맥을 단방향으로 읽는게 문제라면 토큰을 섞어서 예측토큰 뒤에 토큰도 연관되게 하자!

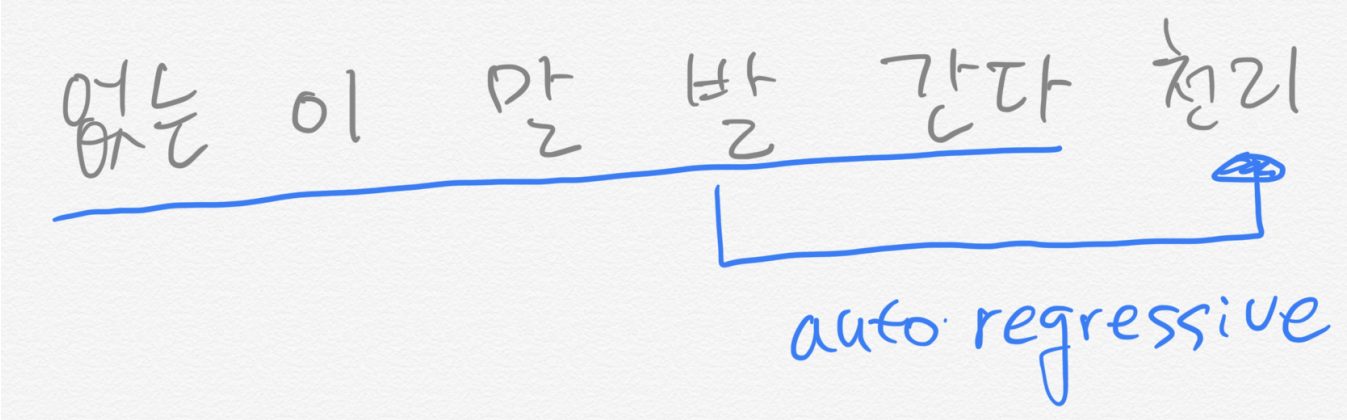

그림 4. 학습과정에서 문장 '발 없는 말 이 천리 간다'의 토큰들을 뒤섞어서 예측하는 과정 시각화.

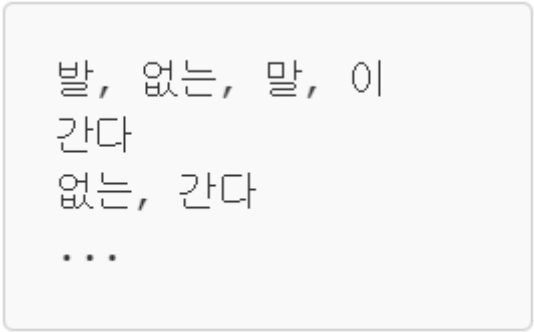

그림 5. '천리' 토큰을 예측하기 위해 사용되는 입력 시퀀스 모음의 예

Permutation Language Model은 BERT(Auto Encoding model 류)가 못했던 마스크 간의 문맥파악이 가능하다!

셔플된 토큰 시퀀스를 순차적으로 읽어서 다음단어를 예측하는 모델이기 때문!

그림 6. 문장 'New York is a city'의 토큰들을 섞은 뒤 학습하는 과정을 시각화.

Permutation Language Model로 학습하는 과정 시각화

실제로 입력 문장의 토큰들을 셔플하는 것이 아닌 Attention Mask로 구현됨. => 현재 단어을 예측하기 위해 Attention에 연산되는 Key, Value의 가짓수가 달라지는 것

상황 1.토큰 4개가 있는 문장을 학습할 때, 문장을 랜덤으로 뒤섞은 결과가 [3, 2, 1, 4] 이고 첫번째 단어(기존 문장에서 3번째 토큰)을 예측해야되는 상황일 경우

첫번째 단어는 자신 앞에 있는 토큰들이 없기때문에 Transformer-XL과 같이 메모리 세그먼트의 히든값으로 연산.

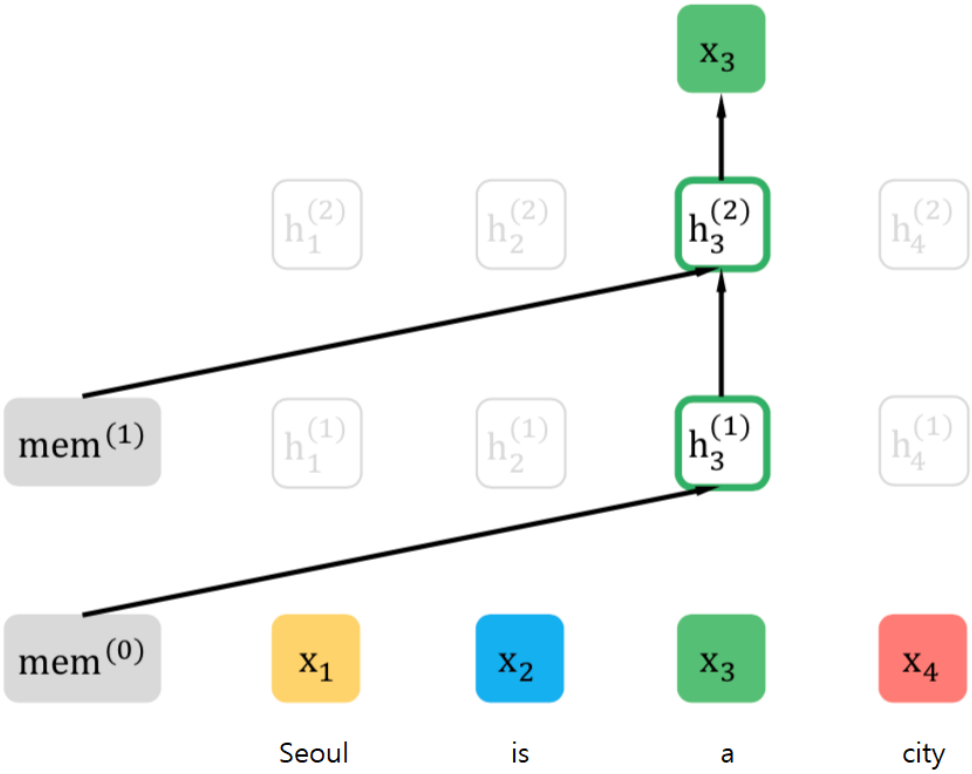

그림 7. 첫번째 토큰인 x_3('a')을 예측하는 과정의 시각화.

상황 2. 토큰 4개가 있는 문장을 학습할 때, 문장을 랜덤으로 뒤섞은 결과가 [2, 4, 3, 1] 이고 세번째 단어(기존 문장에서 3번째 토큰)을 예측해야되는 상황일 경우

3번 토큰 이전의 토큰들(메모리 세그먼트, 2번 토큰, 4번 토큰)을 입력으로 사용.

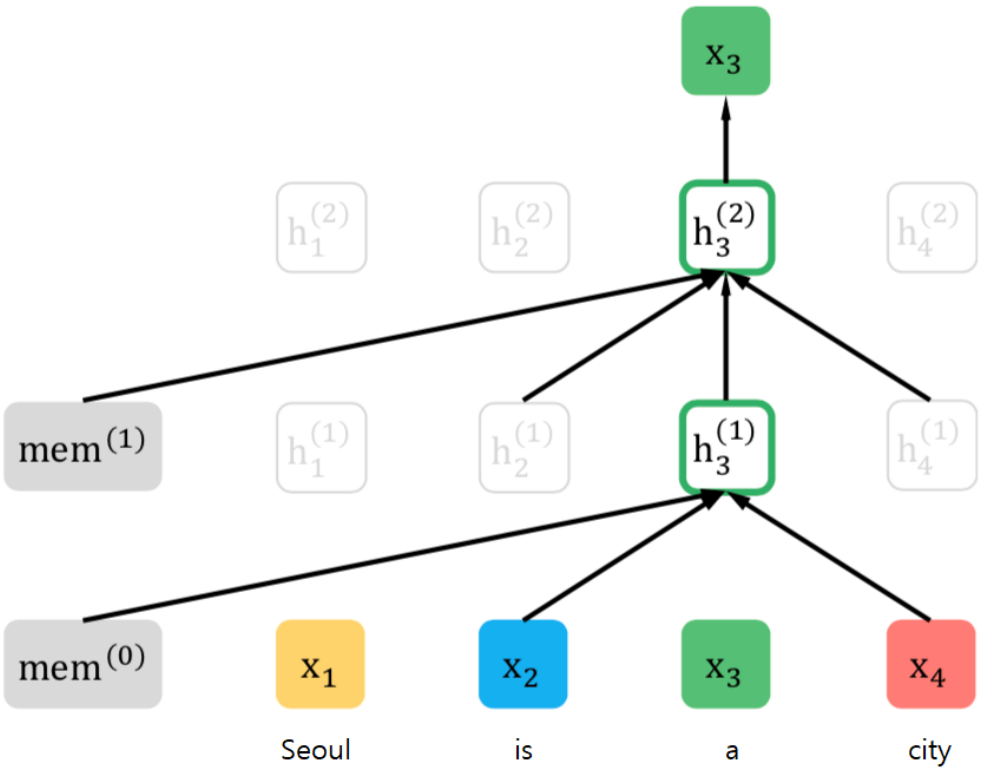

그림 8. 세번째 토큰인 x_3('a')을 예측하는 과정의 시각화.

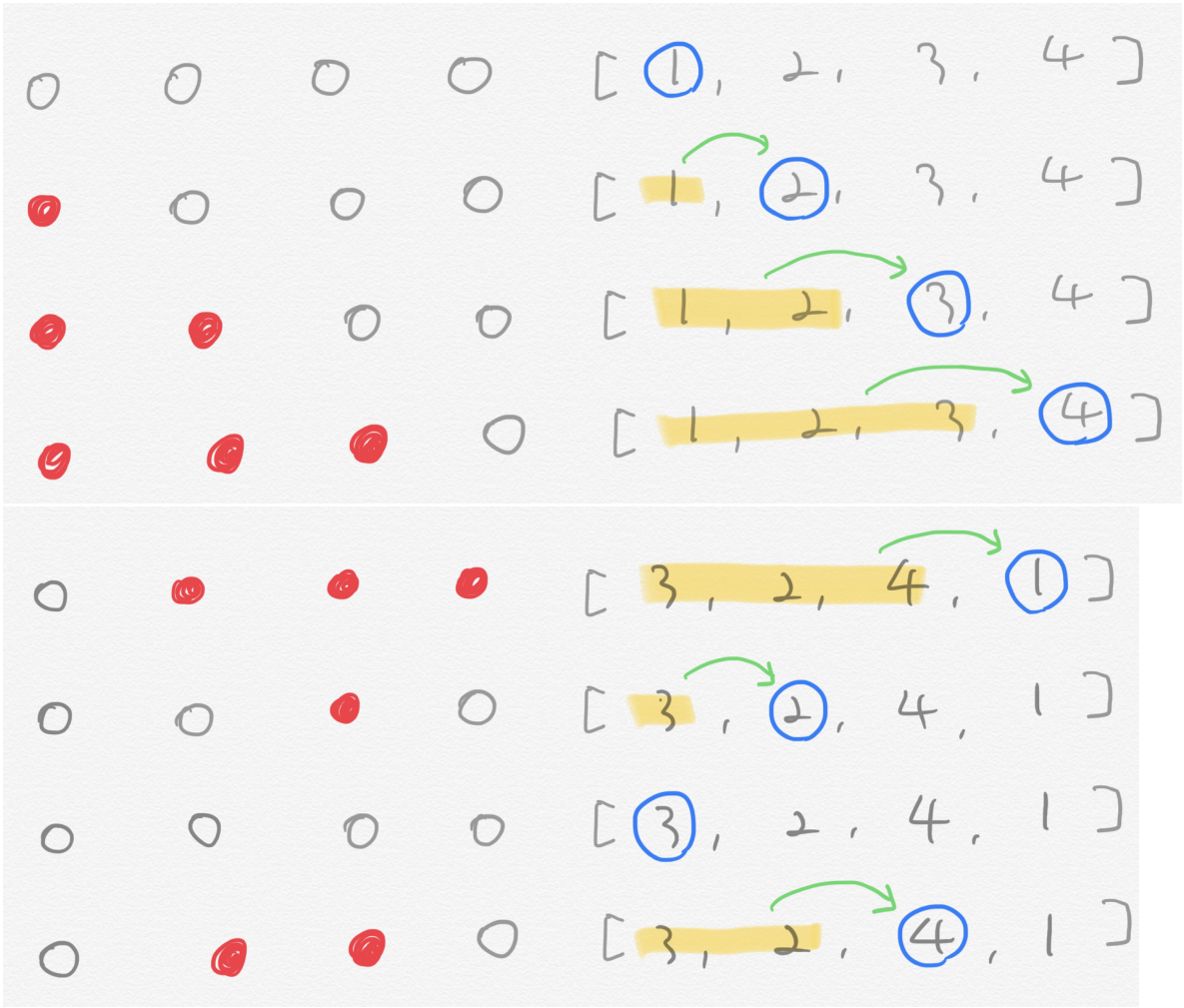

그림 9. (왼) Transformer의 Masked Self-Attention의 Attention Mask. (오) XLNet이 제안하는 Attention Mask.

수식으로 표현

$$input\ sequence : x = (x_1, x_2, \ldots, x_T)$$

$$likelihood : \mathbb{E}_{z \backsim Z_T}[\Pi_{t=1}^{T} p(x_{z_t} \mid x_{z<t})]$$

$$training\ objective : \max_{\theta}\ \mathbb{E}_{z \backsim Z_T}[\Sigma_{t=1}^{T} \log\ p_\theta(x_{z_t} \mid x_{z<t})]$$

Likelihood & objective

- input sequence index의 모든 permutation을 고려한 AR방식을 이용함.
- 예를들어 4개의 토큰으로 이뤄진 문장의 permutation의 집합은 4! = 24. Z_T = [[1,2,3,4],[1,2,4,3],[1,3,2,4]…[4,3,2,1]]

Target-Aware Representations

Permutation Language Model을 사용하면 뒤죽박죽 섞인 문장에 대해서 학습을 진행함.

=> Standard Softmax Formulation을 사용하면 타겟 토큰의 위치에 관계없이 토큰이 등장할 동일한 분포가 형성되는 문제가 발생 ! ex) 'New York is a city'와 'York is a new city' 라는 문장이 주어졌을 때 모델이 문맥을 고려하지 않고 학습.

=> 그러면 기존의 Attention representation를 사용하지말고 위치정보를 부여한 Vector를 사용해야되겠다!

$$\frac{\exp\left(e(x)^\top h_\theta(\mathbf{x}_{\mathbf{z}<t})\right)}{\sum_{x'}\exp\left(e(x')^\top h_\theta(\mathbf{x}_{\mathbf{z}<t})\right)}$$

$$p_\theta(X_{z_t}=x\mid \mathbf{x}_{z<t})=\frac{\exp\left(e(x)^\top g_\theta(\mathbf{x}_{\mathbf{z}<t},z_t)\right)}{\sum_{x'}\exp\left(e(x')^\top g_\theta(\mathbf{x}_{\mathbf{z}<t},z_t)\right)},$$

g_θ : 맞춰야할 토큰 정보 + 맞춰야할 타겟 토큰의 위치정보를 고려한 hidden representation

Two-Stream Self Attention

위 수식을 만들고 보니 training objective 중 g_θ를 어떻게 만들어야할지 난감... (아직도 고려 대상)

g_θ의 조건

1. 특정 시점 t에서 target position z_t 의 token x_{z_t}을 예측하기 위해, hidden representation $g(x_{z<t}, z_t)$는 t 시점 이전의 context 정보 $x_{z<t}$ 와 target position 정보 z_t 만을 이용해야 합니다.
2. 특정 시점 t 이후인 $j\ (>t)$ 에 해당하는 x_{z_j} 를 예측하기 위해, hidden representation $g(x_{z<t}, z_t)$ 가 t 시점의 content인 x_{z_t} 를 인코딩 해야 합니다.

위에 말을 간단히 풀이

1. 셔플된 토큰들에서 어떤 토큰을 예측하기 위해서는 해당 토큰 이전의 정보와 해당토큰의 위치정보로 Attention을 사용해야한다. (Autoregressive Model에서 Masked Self-Attention과 동일)
2. 셔플된 토큰들에서 어떤 토큰(시점 t) 뒤에 있는 다른 토큰을 예측하기 위해서는 어떤 토큰(시점 t)를 인코딩해야한다.

=> XLNet에서는 1번 조건을 충족하기 위해 'Query Stream Attention'을 사용, 2번 조건을 충족하기 위해 'Content Stream Attention'을 사용해서 'Two-Stream Self Attention'을 제안함!

Two-Stream Self Attention = Query Stream Attention+ Content Stream Attention + (Transformer-XL의 Memory Segment도 포함)

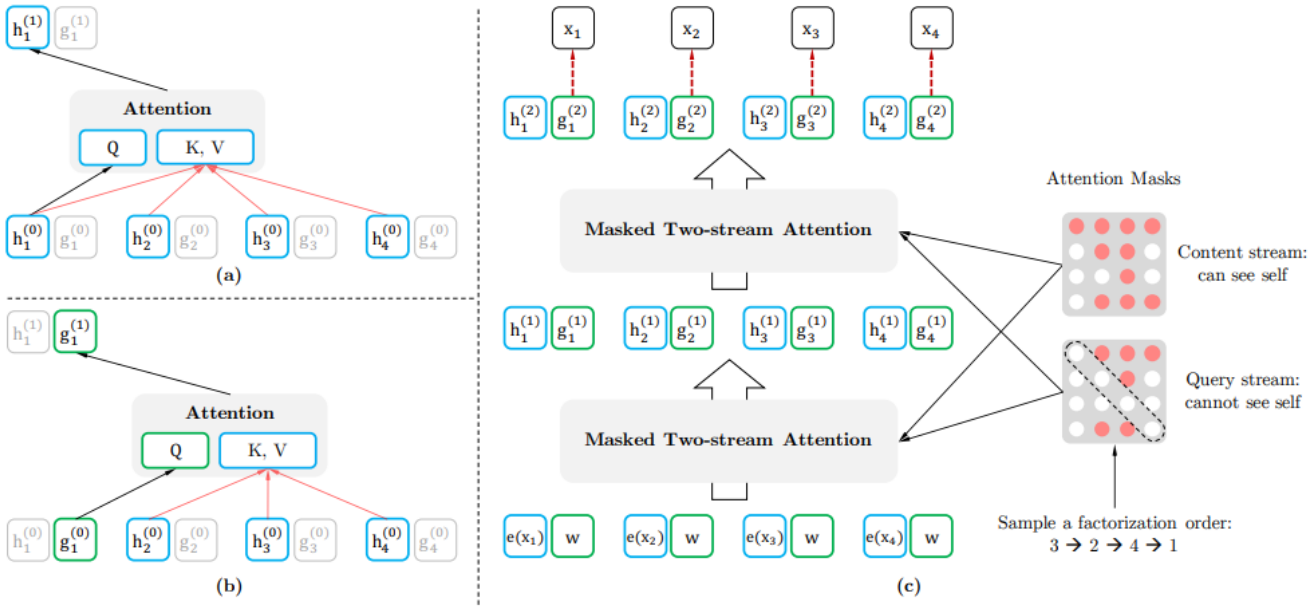

그림 10. 논문에서 제공하는 사용한 Attention 기법 그림. (a) Content Stream Attention. (b) Query Stream Attention. (c) Masked Two-stream Attention 적용.

Content Stream

기존의 Masked-Self-Attention 기법과 동일!

Q : 예측하고자하는 토큰의 히든 값, K, V : 문장 내에 모든 토큰의 히든 값

해당 논문에서는 h로 Content representation를 표현

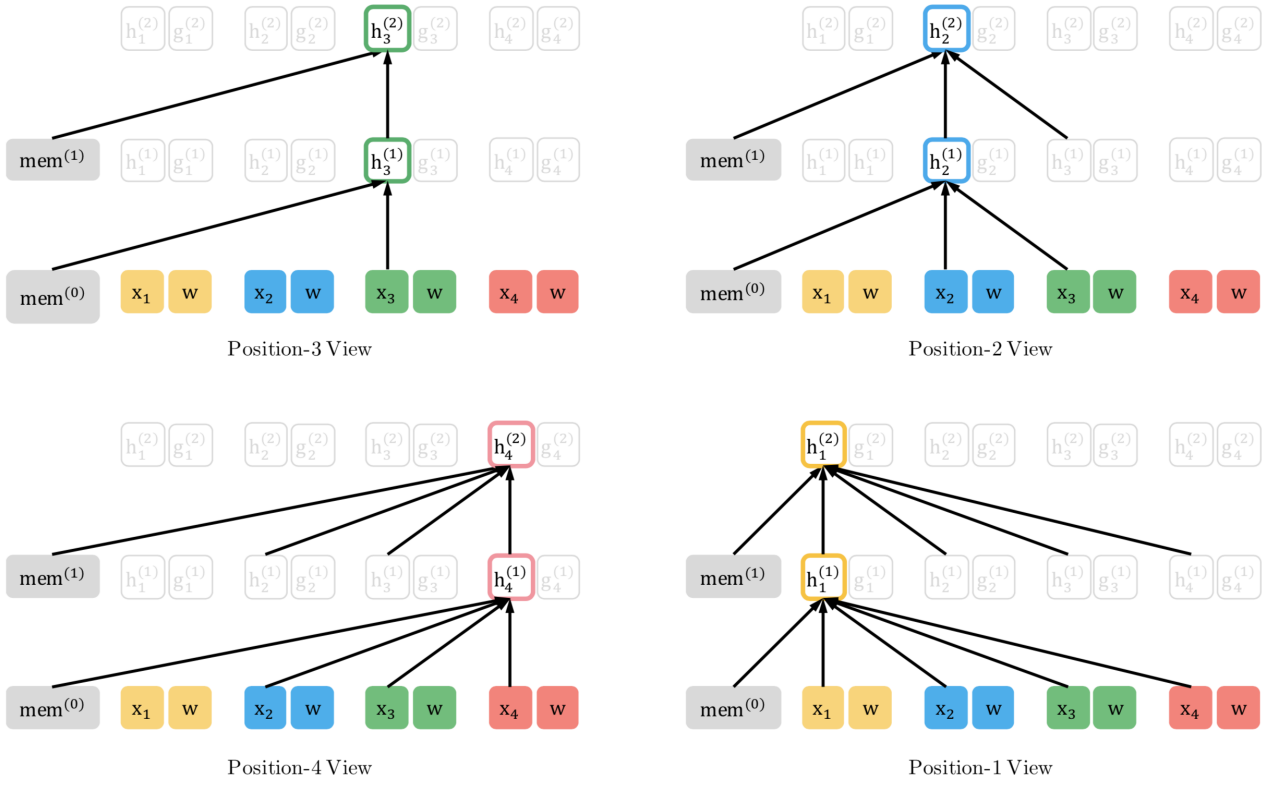

그림 11. Content Stream Attention 시각화

Query Stream

토큰과 위치정보를 활용한 셀트어텐션 기법.

기존의 Transformer에서 Masked Self-Attention에서 예측하려는 토큰의 위치정보가 추가된 것!

예측하려는 토큰의 위치보다 문맥상 뒤에있는 토큰들은 현재의 예측에 영향을 주지않음! (시점 t에서는 t보다 작은 토큰들의 정보만을 사용)

해당 논문에서는 g로 Query representation를 표현

w : random trainable variable

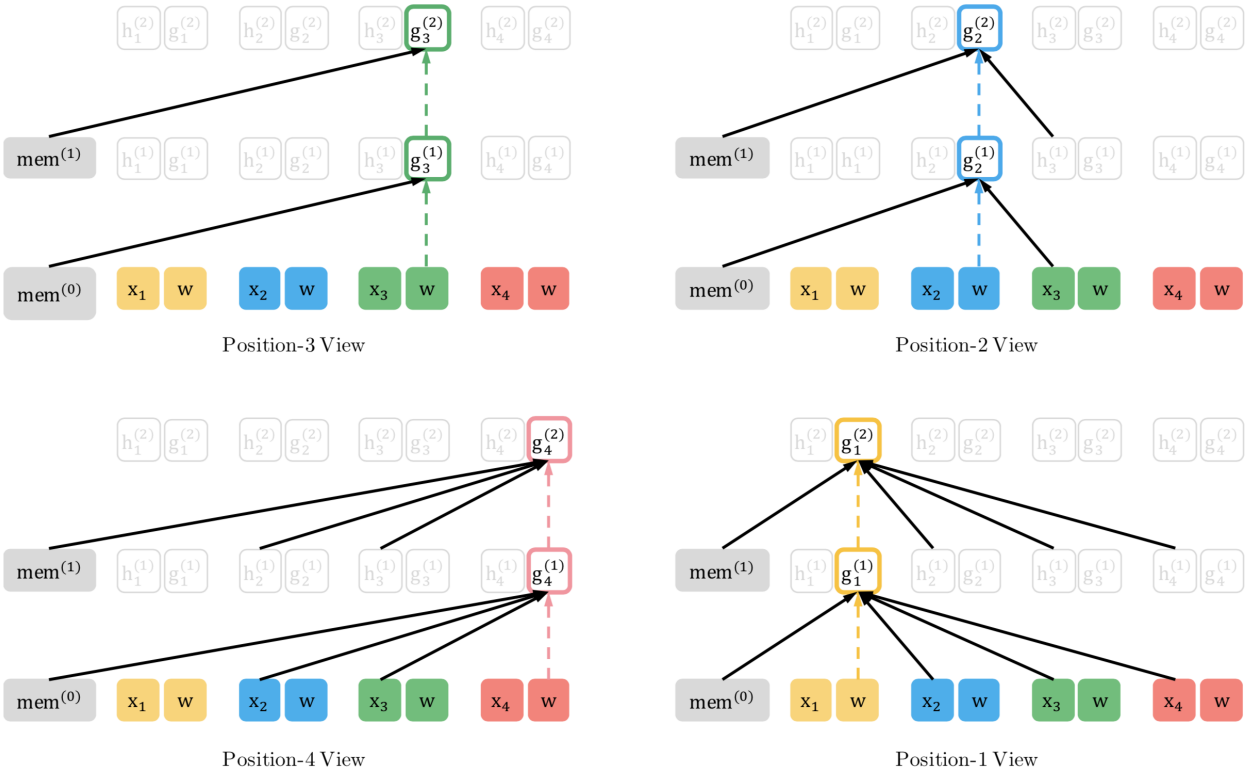

그림 12. Query Stream Attention 시각화

수식으로 표현

Query Stream과 Content Stream 수식

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = g_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad \text{(query stream: use } z_t \text{ but cannot see } x_{z_t})$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta), \quad \text{(content stream: use both } z_t \text{ and } x_{z_t}).$$

Content Stream과 Transformer-XL의 Memory Segment 결합 수식

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \left[\tilde{\mathbf{h}}^{(m-1)}, \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}\right]; \theta)$$