XLNet: Generalized Autoregressive Pretraining for Language Understanding

한나경

목차

1. Abstract

Abstract

기존의 노이즈를 없애는 것은 BERT와 같은 Auto-Encoding 방법이 Auto Regressive한 것보다 더 좋음

-> But 마스킹 된 것들끼리의 dependency를 무시하며, fine-tuning 시에는 마스킹되지 않은 문제들을 푼다는 점에서 불일치가 존재

XLNet의 등장 -> Transformers-XL의 idea를 통합

-> 방법

- 가능한 모든 조합들의 likelihood를 높이는 방식

- Auto Regressive formula로 BERT의 한계점을 보완함

-> 결과

- 20개의 부문에서 SOTA를 기록

2. Introduction

Introduction

표현학습은 NLP Task에서 좋은 성능을 기록하고 있음 -> AR이나 AE 하는 것

AR 기반 언어 모델과 AE 기반 언어 모델의 차이점

AR 기반 언어 모델

- Text-corpus에 대한 확률 분포 추정하는 것을 목표로 함
- Forward와 Backward 양방향을 생각하는 것이 아니라, Forward나 Backward만 인코딩 되는 것으로 학습
- -> Fine-tuning 할 때 양방향 모델의 task가 필요한 경우에 효율적이지 못함

AE 기반 언어 모델

- 확률 분포를 추정하는 방식 X
- input을 재구축하는 것에 초점 -> AR에서의 gap을 줄여주는 역할
- But [MASK] 토큰 자체는 Fine-tuning할 때 사용되는 토큰 X
- -> Pre-training 할 때와 Fine-tuning 할 때의 불일치가 발생할 수 밖에 없음
- masking으로 인해 단어 간의 결합 확률 분포를 추정할 수 X -> 조건부 확률 계산 불가능

Introduction

XLNet 등장

기존 AR의 단점과 AE의 단점을 보완한 모델

기존 AR의 단점 : 양방향 구조에 대한 문제를 학습하지 못함

-> 가능한 모든 permutation을 하여, expected log likelihood를 높이는 것을 목표로 함

-> 각 포지션은 모든 포지션에서 contextual information을 활용할 수 있게 됨

data 훼손에 초점을 맞추지 X -> BERT가 [MASK] 토큰을 이용한 것과 차이점

Introduction

XLNet 모델의 구조

Transformer–XL의 relative scheme를 사용하고, 부분적인 RNN 메커니즘 이용

기존 Transformers의 문제는 고정된 길이가 아니면 의존성을 학습하기 어렵다는 것

-> 512로 임베딩하기 때문에 512가 넘으면 학습 불가

-> Transformer-XL을 통해 512를 각각 묶어서 segment 별로 나눈 다음에 segment 자체도 학습시켜 해결

Transformer-XL 언어 모델을 permutation-based에 적용하는 것은 X

-> permutation은 임의적이고 target은 모호하기 때문

-> 모호성을 지우기 위해 파라미터를 다시 설정

two-stream attention mechanism에 대해 아는 것이 중요

3. Proposed Method

3.1. Background

AR Modeling의 경우 아래의 Likelihood를 최대화하는 방향으로 학습

-> h가 의미하는 것은 t-1 시점까지의 hidden state (context vector)가 있는 것 (RNN이나 Transformers)

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

3.1. Background

BERT의 경우

-> x_hat을 masking된 것이라고 생각하고 x_bar를 예측해야 한다고 생각하면 아래와 같음

-> H_theta는 각 층의 hidden layer를 통과시킬 때의 값 (T개)

-> mt=1이라는 것은 masking이 되었다는 뜻

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}, \qquad (2)$$

3.1. Background

BERT의 Auto-Encoder와 XLNet의 Auto-Regressive를 3가지 측면에서 비교

1. Independence Assumption -> BERT는 독립성 가정, AR은 Product Rule을 적용 (독립성 X)
2. Input Noise : BERT의 경우에는 input의 mask를 씌워서 Noise를 발생시키지만 AR은 바꾸지 X
3. Context Dependency : BERT는 양방향이고, XLNet은 단방향

3.2. Objective : Permutation Language Modeling

BERT와 XLNet 모두 unique한 방법으로 장단점 존재

XLNet의 permutation한 방법을 이용함

-> AR Models의 이점과 동시에 양방향 context를 캡쳐함

-> 각 factorization orders마다 파라미터를 공유하기 때문에, 정보를 공유함 (독립성 X)

-> zt는 가능한 모든 permutation set에 대한 내용이므로 아래의 목적함수 최대화

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T}\log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]. \tag{3}$$

3.2. Objective : Permutation Language Modeling

예를 들어, ['나는', '논문', '을', '읽고', '있다', '.'] 와 같은 집합이 나오면 [1, 2, 3, 4, 5, 6], [2, 3, 4, 5, 6, 1], [3, 4, 5, 6, 1, 2] 와 같이 섞일 수 있음

-> But 모든 경우의 수를 따지는 것은 불가능하기 때문에, 하나의 텍스트 시퀀스에 대해서 하나의 permutation 순서를 샘플링하고 하나의 순서에 대해 분해하여 모든 순서를 고려

Remark on Permutation

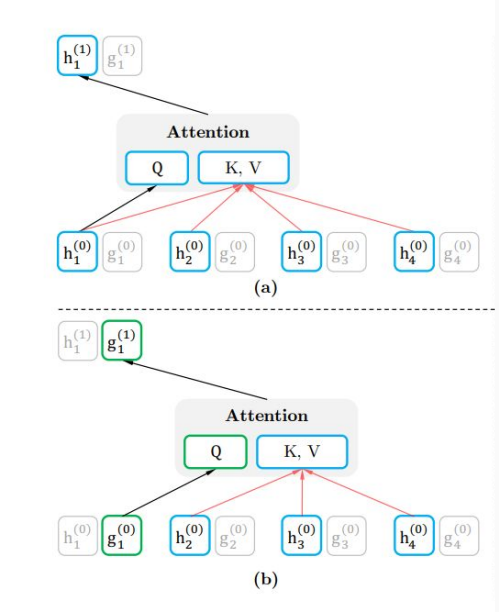
input sequence를 섞는 것이 아니라 요소를 섞는 것

-> 1 2 3 4의 순서가 있으면 3 2 4 1의 형태로 섞음

-> original sequence는 유지한 채로, factorization permutation을 하면 적절한 attention mask를 이용함

-> Two-stream attention 중에서 Content stream attention에 해당됨

3.3. Architecture : Two-Stream Self-attention for target-aware representation



(a)는 Content Stream Attention, (b)는 Query Stream Attention

- Permutation을 하는 데 있어 naive한 방법으로는 해결 X

문제점

- h랑 관련된 식으로 예측을 하면, permutation 때문에 동일한 input이 들어와서 충돌
- 예를 들면, [[3, 2, 4, 1], [3,2, 1, 4]]와 같은 경우, 4를 예측할 때와 1을 예측할 때 모두 3, 2를 참고
  이전 시점에 대한 정보 + 현재 시점에서 예측해야 하는 정보의 위치 정보까지 추가하면 어떨까?
- h에서 g로 바꾸는 방법이 필요 g의 경우에는 t 시점에 대한 정보를 예측할 때, xt(정보)는 이용하면 안 되고 오로지 위치까지만 알고 있어야 함
- t 이후에 예측하는 것을 생각할 때, xt 자체도 인코딩 을 해야함

3.3. Architecture : Two-Stream Self-attention for target-aware representation

파라미터

- 첫 번째 layer의 query stream은 초기화
- content stream은 BERT에서의 word embedding과 동일함
- 각 layer를 통과할 때마다 two-stream은 업데이트가 됨
- 각 layer를 통과할 때마다 기존 transformer에 있던 residual connection이나 layer normalization 진행 X
- Content Stream과 Query Stream은 학습하면서 서로 파라미터 공유하면서 학습
- Fine-tuning 시에는 Query Stream은 제거하고 Context Stream만 이용

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

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

3.3. Architecture : Two-Stream Self-attention for target-aware representation

Partial Prediction

Permutation language modeling objective가 많은 장점이 있지만 섞는 과정에서 학습 양이 많아지고 수렴이 느려짐

-> 다 학습하는 것이 아니라 특정 시점 C 를 기준으로 뒤에 몇개만 예측하도록 하여 시간 절약

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\log p_\theta(\mathbf{x}_{\mathbf{z}_{>c}} \mid \mathbf{x}_{\mathbf{z}_{\le c}})\right] = \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=c+1}^{|\mathbf{z}|} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]. \qquad (5)$$

3.4. Incorporating Ideas from Transformer-XL

기존 **Transformer**와의 차이점

- **기존에는 고정된 길이를 넘을 경우에는 학습에서 토큰들을 제외시킴**

  -> 이를 해결한 것이 Transformers-XL의 Segment Recurrence이며, 더 긴 길이의 sequence를 예측할 수 있음

- **SOTA의 AR 모델을 통합함**

  -> Relative positional encoding scheme + segment recurrence mechanism

  -> 기존의 Positional Encoding은 embedding + absolute positional encoding이었는데, 이것은 하나의 segment 내에서는 여러 의미를 표현하지만, segment 각각에 대해서는 표현하기 어려움

    -> 즉, Transformer-XL에서 메모리와 현재 segment 사이의 relative positional encoding을 사용했던 것을 차용함

    -> segment 자체를 메모리에 저장해놓고, 다음 segment 예측 때 사용할 수 있게 함

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

3.5. Modeling Multiple Segments

BERT와의 차이점은 segment 단위로 예측한다는 것

-> BERT와 유사하게 [CLS, A, SEP, B, SEP] 구조인데, A와 B가 Segment

-> A, B를 이용하여 Permutation을 진행하는 방식인데, XL-Net에서는 NSP(Next Sentence Prediction)에 대한 효과 X, NSP에 대한 objective function 없음

Relative Segment Encodings

-> Transformer-XL에서 아이디어를 착안하여 relative segment encoding을 진행

-> A B가 같은 Segment면 s+, 다르면 s-로 이것또한 학습

-> 단어 자체 뿐만 아니라 segment도 인코딩을 한다는 것

결과

-> Segment Embedding을 통해 2개 이상의 input segments에 대해서도 fine-tuning을 할 수 있는 가능성이 생김

3.6. Discussion

BERT와 XLNet의 공통점과 차이점

공통점

- Partial Prediction을 함 : 효율적인 context를 예측하며 최적화의 어려움을 줄임

차이점

- BERT는 Masking된 단어들 간의 의존성 계산 X

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$
$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city}).$$

- XLNet은 Permutation을 하기 때문에 차이점이 존재
- XLNet이 단어 간의 의존성을 계산하는데 더 유리함

4. Experiments

4.1. Pretraining and Implementation
4.2. Fair Comparison with BERT

Model	SQuAD1.1	SQuAD2.0	RACE	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
BERT-Large (Best of 3)	86.7/92.8	82.8/85.5	75.1	87.3	93.0	91.4	74.0	94.0	88.7	63.7	90.2
XLNet-Large-wikibooks	88.2/94.0	85.1/87.8	77.4	88.4	93.9	91.8	81.2	94.4	90.0	65.2	91.1

Table 1: Fair comparison with BERT. All models are trained using the same data and hyperparameters as in BERT. We use the best of 3 BERT variants for comparison; i.e., the original BERT, BERT with whole word masking, and BERT without next sentence prediction.

- BERT와 최대한 동일하게 환경 설정
- 결론적으로 XLNet이 뛰어남

4.3. Comparison with RoBERTa : Scaling Up
4.4. Ablation Study

3가지 측면에서 Ablation Study

1. permutation language modeling objective의 효과적인 정도를 BERT와 비교
2. Transformer-XL backbone을 Neural architecture의 backbone으로 쓰는 것의 중요성
3. span-based prediction, bidirectional input pipeline, next-sentence prediction의 필요성

이를 위해 6가지의 서로 다른 XLNet 구조를 제안하고, 비교군으로 BERT-base와 DAE objective를 사용

- 실험환경은 BERT-base와 동일하게 트랜스포머를 12층으로 쌓음
- Wikipedia와 BookCorpus를 가지고 사전학습 진행

결과

- 전체적으로 XLNet의 성능이 더 좋음

#	Model	RACE	SQuAD2.0 F1	SQuAD2.0 EM	MNLI m/mm	SST-2
1	BERT-Base	64.3	76.30	73.66	84.34/84.65	92.78
2	DAE + Transformer-XL	65.03	79.56	76.80	84.88/84.45	92.60
3	XLNet-Base ($K = 7$)	66.05	**81.33**	**78.46**	**85.84/85.43**	92.66
4	XLNet-Base ($K = 6$)	66.66	80.98	78.18	85.63/85.12	**93.35**
5	- memory	65.55	80.15	77.27	85.32/85.05	92.78
6	- span-based pred	65.95	80.61	77.91	85.49/85.02	93.12
7	- bidirectional data	66.34	80.65	77.87	85.31/84.99	92.66
8	+ next-sent pred	**66.76**	79.83	76.94	85.32/85.09	92.89

Table 6: The results of BERT on RACE are taken from [38]. We run BERT on the other datasets using the official implementation and the same hyperparameter search space as XLNet. K is a hyperparameter to control the optimization difficulty (see Section 2.3).

Examining rows 1 - 4 of Table 6, we can see both Transformer-XL and the permutation LM clearly contribute the superior performance of XLNet over BERT. Moreover, if we remove the memory caching mechanism (row 5), the performance clearly drops, especially for RACE which involves the longest context among the 4 tasks. In addition, rows 6 - 7 show that both span-based prediction and the bidirectional input pipeline play important roles in XLNet. Finally, we unexpectedly find the the next-sentence prediction objective proposed in the original BERT does not necessarily lead to an improvement in our setting. Hence, we exclude the next-sentence prediction objective from XLNet.

Finally, we also perform a qualitative study of the attention patterns, which is included in Appendix A.6 due to page limit.

5. Conclusion

5. Conclusion

- 기존의 **Pre-training** 모델을 **Autoregressive**와 **Autoencoding**이라는 새로운 관점으로 분석한 것이 의미있음
- **BERT**보다 좋은 **representation**을 학습하면서도 **GPT**처럼 **autoregressive**한 특성을 갖고 있기 때문에 좋은 성능이 기대됨
- 하지만 짧은 문장을 다루는 **task**에서는 **XLNet**이나 **Transformer-XL**이 가지는 긴 문장에 대한 이점을 충분히 활용 **X**여서 큰 성능 향상은 불가능

감사합니다.