논문 정리

파트 구분

0. Abstract ~ 2. Background

이승준

3. Model Architecture ~ 4. Why Self-Attention

이소희

5. Training ~ 7. Conclusion

심승현

총정리 심승현

0. Abstract

기존 아키텍처 for sequence

- RNN, CNN 기반 encoder-decoder 구조
    - attention 메커니즘을 사용하여 입력과 출력 시퀀스 간의 의존성을 모델링

새로운 아키텍처 Transformer 제안

- 간단한 구조
- 오로지 attention 메커니즘에 의존
- recurrence, convolution 활용X

모델 성능

- 두 개의 태스크(기계 번역)에서 우수한 BLEU 스코어를 보임

    ▼ BLEU 스코어란

        - 기계번역 시스템 평가를 위한 지표
            - n-gram 추출 후 텍스트 빈도수를 계산, 원본과 생성본을 비교 후 정확도 추출
- 데이터셋의 규모에 관계 없이 좋은 성능을 보임

1. Introduction

기존 language modeling 방법론

- RNN, LSTM, GRU : encoder + decoder 기반
- sequence modeling, language translation SOTA 모델들
    - 해당 모델들은 시퀀스 모델링 및 기계 번역 분야에서 최고의 성능을 보였다.

Recurrence 모델

- 시간 순서에 따라 hidden state (h_t, h_{t-1},...)를 업데이트
- train 시 병렬 연산 제한
    - 시퀀스가 길어지면 메모리 효율 측면, 배치 연산 측면 비효율적

    → 조건부 연산 등으로 개선된 연구, 여전히 sequence 연산의 근본적인 문제 존재

Attention 매커니즘

- 입출력 시퀀스의 길이에 관계없이 모든 위치의 정보를 동시에 처리
    - 병렬 연산 가능

⇒ `Transformer`는 이러한 attention 매커니즘만을 활용하여 병렬 연산의 한계를 극복하고, recurrence 없이도 우수한 성능을 보임.

▼ Recurrence vs. Attention (Dependency)

Recurrence와 의존성 모델링

Recurrent Neural Networks (RNNs):

- **의존성 모델링**: RNN, LSTM, GRU 같은 순환 신경망은 입력 시퀀스의 각 위치에서 이전 상태(hidden state)를 기반으로 다음 상태를 계산합니다. 이 과정에서 시퀀스의 순차적 의존성을 자연스럽게 모델링합니다.
- **문제점**: 순차적 특성 때문에 병렬 연산이 어렵고, 긴 시퀀스에서는 그래디언트 소실 문제(vanishing gradient problem)가 발생할 수 있습니다. 이러한 문제는 학습 시간 증가 및 성능 저하로 이어질 수 있습니다.

Attention 매커니즘과 의존성 모델링

Attention Mechanism:

- **의존성 모델링**: Attention 메커니즘은 시퀀스의 모든 위치 간의 관계를 직접 모델링할 수 있습니다. 이는 각 위치의 정보를 다른 모든 위치와 동시에 비교하고 가중치를 할당하여 중요한 정보를 강조합니다. 예를 들어, 기계 번역에서 특정 단어가 문장 내의 다른 단어와 얼마나 관련이 있는지를 판단할 수 있습니다.

- **장점**: Attention은 모든 시퀀스 위치를 한 번에 처리할 수 있어 병렬 연산이 가능하며, 긴 시퀀스에서도 효율적으로 작동합니다.

Transformer와 의존성 모델링

Transformer:

- **의존성 모델링**: Transformer는 오로지 Attention 메커니즘에 의존하여 시퀀스의 모든 위치 간의 의존성을 동시에 모델링합니다. 이를 통해 입력 시퀀스의 각 위치에서 출력 시퀀스의 모든 위치를 동시에 참조할 수 있습니다.
- **구성**: Transformer는 Self-Attention과 Multi-Head Attention을 활용하여 시퀀스의 모든 부분 간의 관계를 학습합니다. 이는 recurrence가 필요 없이 모든 위치 간의 종속성을 효과적으로 모델링할 수 있습니다.

2. Background

Sequential Computation의 제약점

- 기존 Neural GPU, ByteNet, ConvS2S 등의 모델은 합성곱 신경망을 사용해 계산량이 과도함
    - 해당 방식은 문장 내에서 먼 단어들 간의 관계를 학습하기에 어려움.

Self-Attention

- 입력 시퀀스 길이에 상관없이 모든 위치의 정보를 동시처리 가능
    - 병렬연산이 가능하다.

End to end Memory

- 반복주의 메커니즘을 사용
    - 입력 시퀀스를 한번에 처리하지 않고, 여러번 반복해서 처리함
    - 간단한 언어 질문 응답 및 모델링 작업에서 좋은 성능을 보임

Transformer

- RNN 혹은 합성곱 신경망을 사용하지 않고, self-attention 메커니즘만을 이용해 입출력 계산
    - 기존 모델들과 같이 encoder + decoder 구조로 구성
    - 병렬연산으로 인한 학습시간 단축

- 우수한 성능 (특히 번역작업 등)

3. Model Architecture

📌 **기본 구조 파악**

Encoder-Decoder 구조 → `Transformer` 는 전통적인 seq2seq 모델 구조를 따름

- Encoder: 입력 시퀀스를 받아 고정 길이의 context 벡터로 변환
- Decoder: context 벡터와 디코더 입력을 사용해 출력 시퀀스를 생성
- 레이어 수: base 모델은 6개의 encoder 레이어와 6개의 decoder 레이어로 구성

3.1 Encoder and Decoder Stacks

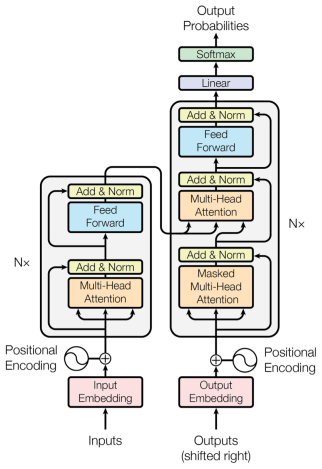


Figure 1: The Transformer - model architecture.

Encoder

- 개별 레이어는 2개의 서브 레이어들로 이루어짐
  1. multi-head self-attention mechanism
     - 입력 시퀀스의 모든 위치 간의 관계를 동시에 모델링

2. position-wise fully connected feed-forward network
    - 각 위치에서 독립적으로 적용되는 두 개의 선형 변환 + ReLU 활성화 함수

→ Add& Norm (Residual Connections와 Layer Normalization 단계) 에서 서브 레이어들 끼리의 잔차 연결을 적용한 후에 레이어 정규화까지 진행

- output
    - 각 서브 레이어의 출력 ⇒ `LayerNorm(x + Sublayer(x))`
    - 모든 서브 레이어 및 임베딩 레이어는 d_{model} = 512 차원으로 출력

Decoder

- Encoder와 같은 구조이나 하나의 서브레이어 추가
    - **Masked** Multi-Head Self-Attention Mechanism

      : decoder의 각 위치가 미래의 위치를 참조하지 않도록 마스킹

나머지 서브 레이어 역할은 모두 동일

▼ 다른 버젼

`Encoder`

- Encoder는 6개의 identical layer로 이루어짐
- 각 layer는 두 개의 sub-layer가 있음
  → 첫 번째 sub-layer는 multi-head self-attention mechanism
  → 두 번째 sub-layer는 간단한 position-wise fully connected feed-forward network
- 각 two sub-layers 마다 layer normalization 후 residual connection
- 즉, 각 sub-layer의 결과는 LayerNorm(x+Sublayer(x)) 임
- residual connection을 구현하기 위해, embedding layer를 포함한 모든 sub-layer들의 output의 dimension은 512

`Decoder`

- Decoder도 마찬가지로 6개의 identical layer로 이루어짐

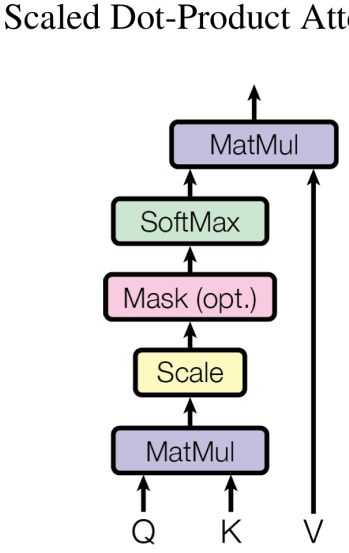
- 각 Encoder layer의 두 sub-layer에, decoder는 세번째 sub-layer를 추가함
  → encoder stack의 결과에 해당 layer가 multi-head attention을 수행함
  전체 time step을 모두 입력으로 받기 때문에 self attention score 을 계산할 때 미래의 값 무시하기 위해 masking
- 마찬가지로 residual connection 적용

3.2 Attention ⭐

> Query와 Key-Value 쌍의 집합을 출력으로 매핑하는 함수 (Q, K, V는 모두 벡터)

> output은 weighted sum, Q ↔ K 에 호환되는 값으로 가중치 할당

3.2.1 Scaled Dot-Product Attention (multiplicative)



- input: Q, K, V 벡터의 집합
- 쿼리와 모든 키의 내적을 계산하고, 각 값을 d_k로 나누어 스케일링
  → 소프트맥스 함수를 적용하여 가중치를 구함

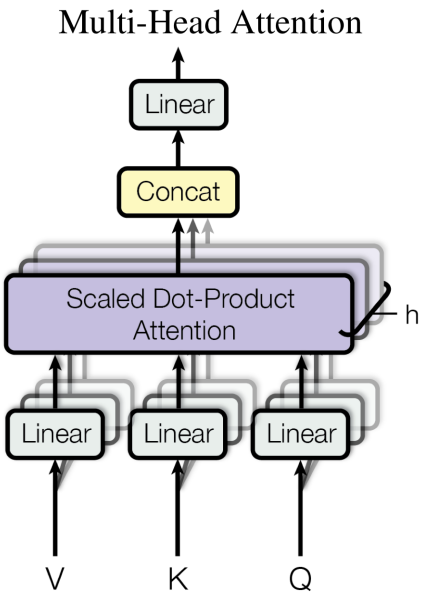
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- (matmul) 쿼리, 키, 값 벡터를 행렬 Q, K, V에 패킹하여 동시에 연산 수행

⇒ additive Attention에 비해 계산 속도가 빠르고 메모리 효율성이 좋음

3.2.2 Multi-Head Attention ⭐⭐⭐

어려움..



> Q, K 값 벡터를 각각 h번 선형 변환하여 여러 개의 독립적인 attention 메커니즘을 병렬로 수행

- 각 헤드는 서로 다른 매트릭스 집합 W_Q, W_K, W_V를 사용하여 입력을 변환하고, 최종 출력을 concat해 선형 변환까지 적용

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

- 헤드 별 계산

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

- 모두 병렬 연산 작용 (논문에서는 8개 layer)

⇒ 정보를 동시에 참조할 수 있음

3.2.3 Applications of Attention in our Model

- Encoder-Decoder Attention
  - 디코더의 쿼리는 이전 디코더 레이어에서 나오고, 키와 값은 인코더 출력에서 옴
  - 디코더의 모든 위치에서 입력 시퀀스의 모든 위치를 참조 가능
- Encoder Self-Attention
  - 키, 값, 쿼리가 모두 이전 인코더 레이어 출력에서 옴
  - 인코더의 각 위치가 이전 레이어의 모든 위치 참조
- Decoder Self-Attention
  - 디코더의 각 위치가 디코더 내의 모든 위치를 참조하되, 미래의 위치는 제외
  - 이를 위해 마스킹 적용

▼ 다른 버젼

3.2.1 Scaled Dot-Product Attention

(1) input : query, key의 dimension d_k, value의 dimension d_v
(2) 모든 q와 k에 대해 아래 식으로 Attention score 계산

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

Attention function은 2가지가 있는데, single hidden layer로 feed-forward layer network를 사용해 compatibility function을 계산하는 `Additive attention`, scaling factor (1 / √d_k) 를 제외하면 이 연구에서의 attention 방식과 동일한 `Dot-product attention` 가 있다.

3.2.2 Multi-Head Attention

Single attention을 d_model-dimensional keys, values, queries에 적용하는 것보다, queries, keys, values를 h번 서로 다른, 학습된 linear projection으로 d_k, d_k와d_v 차원에 linear하게 project하는 게 더 효과적이라는 사실을 알아내게 되었다.

project된 각 값들은 병렬적으로 attention function을 거쳐 d_v-dimensional output value를 만들고, 이 결과들은 다시 합쳐진 다음, 다시 한번 project 되어 최종 결과값을 만들게 된다.

$$MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

이 연구에선 h=8 이고, 각각 d_k = d_v = d_model / h =64
각 head 마다 차원을 줄이기 때문에 total computation은 full dimensionality의 single-head attention와 유사하다.

3.2.3 Applications of Attention in our Model

Transformer는 세 가지 방법으로 multi-head attention을 사용한다.

1. `self-attention in encoder` : queries, keys, values 모두 encoder로부터 가져오고, 어떤 한 단어가 모든 단어들 중 어떤 단어들과 correlation이 높고, 또 어떤 단어와는 낮은지를 계산한다.
2. `self-attention in decoder` : encoder의 self-attention과 같다. 하지만 decoder의 경우, sequence model의 auto-regressive property를 보존해야하기 때문에 masking vector를 사용하여 해당 position 이전의 벡터들만을 참조한다.
3. `encoder decoder attention` : queries는 이전 decoder layer에서 가져오고, keys와 values는 encoder의 output에서 가져온다. decoder의 sequence vector들이 encoder의 sequence vector들과 어떠한 correlation을 가지는지를 계산한다.

3.3 Position-wise Feed-Forward Networks ⭐

> 각 레이어는 **두 개의 선형 변환 + ReLU 활성화 함수 = fully connected feed forward net** 포함

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

- 모든 위치에서 동일하게 적용
- 입출력 차원: d_{model} = 512, 내부 레이어 차원: d_{ff} = 2048

▼ 다른 버전

3.3 Position-wise Feed-Forward Networks

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

attention layer과 함께 fully connected feed-forward network가 사용된다. 각 position에 따로따로, 동일하게 적용되며 • ReLu 활성화 함수를 포함한 두 개의 선형 변환이 포함된다.

input과 output의 차원은 512, inner-layer의 차원은 2048이다.

3.4 Embeddings and Softmax

- 입력 토큰, 출력 토큰 → d_{model} 차원의 벡터로 변환하는 학습된 임베딩 사용
- 디코더 출력을 예측된 다음 토큰 확률로 변환하기 위해 학습된 **선형 변환 및 소프트맥스 함수** 사용
- 두 임베딩 레이어와 pre-softmax 선형 변환 사이에 동일한 가중치 행렬 사용
- 임베딩 레이어에서 가중치를 $\sqrt{d_{model}}$로 스케일링.

▼ 다른 버전

3.4 Embeddings and Softmax

다른 sequence transduction models 처럼, learned embedding을 사용하며, input 토큰과 output 토큰을 d_model 의 벡터로 변환한다. decoder output을 예측된 다음 토큰의 확률로 변환하기 위해 선형 변환과 softmax를 사용한다. input embedding 과 output embedding에서 weight matrix를 서로 share하여 사용한다.

3.5 Positional Encoding ⭐

> 모델이 시퀀스의 순서 정보를 활용할 수 있도록 위치 정보를 입력

- 사인 코사인 함수 주기 이용

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}}))$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}}))$$

▼ 다른 버젼

3.5 Positional Encoding

어떤 recurrence, convolution도 사용하지 않기 때문에, sequence의 순서를 유지하기 위해 sequence의 상대적, 절대적 position에 대한 정보를 입력한다.

인코더와 디코더 stack 아래의 input 임베딩에 다른 주기의 sine, cosine function을 사용하여 positional encoding 해준다.

4. Why Self-Attention

연산 효율성

- Self-Attention은 시퀀스의 모든 위치를 동시에 처리할 수 있어 병렬 처리에 유리
- 모든 위치 간의 직접적인 관계를 모델링하므로, 긴 시퀀스에서도 효과적
    - RNN의 경우, 긴 시퀀스에서는 정보가 손실되기 쉽지만 Self-Attention은 이 문제를 완화함

해석 측면

- Attention 가중치를 통해 모델이 어떤 입력 위치에 집중하는지 파악 가능
- 특정 위치가 예측에 미치는 영향 확인 가능

▼ 다른 버젼

- recurrent, convolution layer와 비교했을 때 self-attention이 complexity가 더 작다. (주로 n이 d보다 작음)
- computation이 parallelized가 가능하므로 input의 모든 position 값들을 연결하여 한번에 처리할 수 있다. (Sequential operations이 O(1))
- network에서 long-range dependency 사이의 path 길이가 짧다. (input과 output sequence에서 position의 조합 간의 path가 짧을수록, long-range

dependecy를 학습하기가 쉬움)

또한, 더 interpretable한 모델을 만들 수 있다.

5. Training

5.1 Training Data and Batching

- Dataset
  - WMT 2014 English-German 및 English-French 데이터셋
- Batching
  - 문장 길이에 따라 비슷한 길이의 문장을 그룹으로 묶어 배치 처리

    → padding 양 줄여 연산 효율화

5.2 Hardware and Schedule

- Hardware
  - NVIDIA P100 GPU
- Schedule
  - learning rate 점진적 감소

5.3 Optimizer

- Optimizer
  - Adam optimizer w/ $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$

5.4 Regularizer

- Regularization
  - dropout으로 과적합 방지
  - $P_{drop} = 0.1$ (dropout_rate; attention layer, ff layer)
- Label smoothing
  - 과적합 방지
  - $\epsilon_{ls} = 0.1$

6. Results

6.1 Machine Translation

- BLEU 스코어
  - (WMT 2014) English-German, English-French 기계 번역 태스크
    - 28.4, 41.0의 BLEU 점수 → SOTA 달성
  - 기존 RNN 기반 모델보다 더 높은 성능을 보임

6.2 Model Variations

- Base 아키텍처의 변형 실험 (English-German 번역 태스크 기준 성능 확인)

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

(A) 단일 헤드 attention은 최적 점수에 비해 BLEU 점수가 0.9 낮고, 너무 많은 헤드를 사용해도 성능 저하.

(B) attention key 크기를 줄이면 성능 저하.

(C), (D) 모델 크기가 커질수록 성능 향상.

dropout 과적합 방지 효과적.

(E) 사인 함수 기반 위치 인코딩을 학습된 위치 임베딩으로 대체 시 효과 미미

6.3 English Constituency Parsing

Transformer 모델이 다른 태스크에도 일반화될 수 있는지 평가하는 English constituency parsing 실험

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

- Train model 설정
  - Wall Street Journal (WSJ) 부분의 Penn Treebank 에서 약 40K개의 훈련 문장을 사용

    (16K 토큰)
    - d_model = 1024인 4-layer Transformer를 훈련
  - high-confidence, BerkeleyParser 코퍼스(더 큰 데이터셋)로 반지도 학습에서도 훈련

    (32K 토큰)

- 실험 조건
  - 모든 파라미터는 English-to-German 기본 번역 모델과 동일하게 유지하되
  - dropout, attention, residual, learning rate, beam size 최적 선정에 대한 소수의 실험 진행

- 실험 결과
  - 태스크 특화의 튜닝이 부족했음에도 매우 좋은 성능을 보임
    - Recurrent Neural Network Grammar 를 제외한 모든 기존 모델보다 더 나은 결과
  - WSJ 데이터만 train 시킨 경우에서도 RNN seq2seq 모델들보다 더 좋은 성능을 보임

7. Conclusion

의의

Transformer는 recurrence 없이 등장한 최초의 sequence transduction model로 multi-head self attention 기법을 활용함

- RNN, CNN 기반 아키텍처보다 빠른 훈련 속도 → 병렬 연산이 가능해 효율성 극대화
- ENG-GER, ENG-FR 기계번역 태스크(WMT2014)에서 SOTA 달성