Attention Is All You Need 논문 정리

arxiv.org

https://arxiv.org/pdf/1706.03762

장건호

1. Abstract & Introduction

Dominant Sequence Transduction models

- RNN or CNN을 기반으로 하여 인코더와 디코더로 구성되는 것이 기존의 모델
- Attention Mechanism 포함
  - attention을 통해 거리가 먼 시퀀스 내 요소들 간의 정보 손실을 방지하고 의존성에 기반한 모델링을 할 수 있도록 해줌.
- LSTM(Long Short-Term Memory) / GRU(Gated Recurrent Unit)
  - language modeling. machine translation에서 우수한 성능

- RNN의 한계
  - 현재의 은닉 상태는 이전의 은닉 상태와 현재 시점의 입력값에 기반하여 계산됨 → 순차적인 계산이 필수적
  - 병렬화 불가
  - 시퀀스가 길어질 경우, 메모리 제한으로 인해 샘플들을 배치 처리하기 어려움.
  - Factorization tricks와 조건부 연산 등의 방법을 통해 계산 효율성을 향상시켰으나 여전히 순차적 연산의 특성을 지닌 RNN의 근본적 문제는 해결 X
  - attention mechanism이 시퀀스로 인한 문제를 어느정도 보완했으나, 독립적인 활용이 이루어지지는 않음.

Transformer

- Attention만을 사용하고 RNN, CNN 구조는 배제함.
- 기존의 모델들보다 더 높은 품질, 병렬화에 용이, 학습속도 역시 빠름.
    - 기계 번역 실험 수행 결과에 기반한 사실

2. Background

- Extended Neural GPU, ByteNet, ConvS2S 등 시퀀셜 연산을 줄이려는 연구는 계속해서 진행되어 옴.
- CNN을 베이스로 모든 입력 및 출력 위치에 대해 병렬 연산을 수행함.
- 하지만 시퀀스가 길어질수록 연산량이 증가한다는 한계가 발생.

⇒ long-range dependency 문제 발생

- Transformer는 연산 수를 constant로 고정하여, 연산 복잡도를 시퀀스 길이와 관계없이 일정하게 유지
- 이러한 과정에서 모델의 effective resolution이 감소하는 문제 발생
- Multi-Head Attention을 통해 임의의 위치 간의 관계성을 더욱 잘 학습할 수 있도록 함.

Self-Attention

- 시퀀스 내에서 다른 두 위치 간의 관계를 학습하여 전체 시퀀스의 표현을 계산하는 attention 메커니즘.
- reading comprehension, abstractive summarization, textual entailment, learning task-independent sentence representations과 같은 task에서 효과적으로 활용됨.
- End-to-End Memory Networks(외부 저장소 활용) → Recurrent Attention Mechanism(RNN X), RNN보다 높은 성능

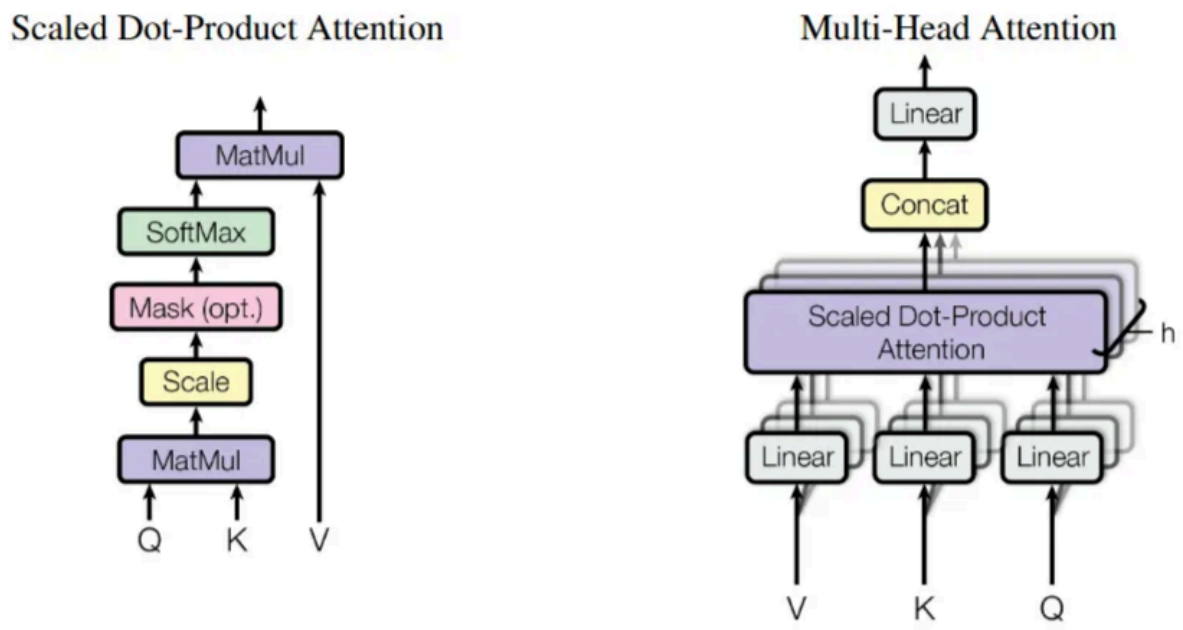
3. Model Architecture

3.1 Encoder and Decoder Stacks

- **Encoder** : 입력 시퀀스(x1, x2, ..., xn)을 입력으로 받아 z = (z1, ..., zn)으로 변환
  - 6개의 identical layers로 구성
  - 각 layer는 두 개의 sub-layer로 구성
    - Multi-head Self Attention mechanism
    - Position-wise fully connected feed forward network
  - **Residual Connection & Layer Normalization** 적용
  - output → LayerNorm(x + Sublayer(x))
  - 모든 sub-layer와 embedding layer는 512차원의 출력 생성
- **Decoder** : 인코더에서 생성한 z를 기반으로 출력 시퀀스 (y1, ..., yn) 출력
  - 동일하게 6개의 identical layers로 구성
  - 3개의 sub-layer로 구성
    - Multi-head Self Attention mechanism
    - Position-wise fully connected feed forward network
    - Multi-head Attention over Encoder Outputs(인코더의 출력을 디코딩에 활용)
  - **Residual Connection & Layer Normalization** 적용
  - **Masking** → 미래의 정보는 차단
    - 현재 시점 이후의 정보에 대해서는 어텐션 적용 X
    - 출력 Embedding을 한 위치씩 오른쪽으로 이동하면서 구현 가능.

기광민

3.2 Attention



Attention 함수 →

- Query와 Key & Value쌍을 Output으로 변환하는 매핑과정
- Output은 Value들의 가중합으로 계산되며, 각 값의 가중치는 Query와 Key의 유사도를 기반으로 결정

3.2.1 Scaled Dot-Product Attention

Transformer에서 사용하는 Attention 방식 = Scaled Dot-Product Attention

Input →Query (Q), Key (K), Value (V)

Attention 계산 과정:

1. Query와 Key의 내적을 계산하여 유사도 측정
2. Key 차원(dk)의 제곱근으로 나누어 정규화 (스케일링)
3. Softmax를 적용하여 가중치 생성
4. 해당 가중치를 Value에 적용하여 최종 출력 계산

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

가장 흔하게 사용되는 Attention으로는 Addictive Attention과 Dot-Product (Multiplicative) Attention이 존재 ⇒ Dot-Product Attention은 연산 속도가 빠르고 메모

리 효율적이지만, 차원이 커질수록 softmax의 결과가 치우치는 문제가 발생 가능 ⇒ 방지하기 위해 2번처럼 스케일링 수행

3.2.2 Multi-Head Attention

Single Attention만 사용할 경우 정보 표현력이 제한될 수 있기 때문에, **Multi-Head Attention**을 적용하여 여러 개의 Attention을 병렬로 수행

- Query, Key, Value를 h개의 개별 subspace로 투영하여 연산을 수행
- 각 Attention head는 독립적으로 연산을 수행한 후, 최종적으로 concatenate 및 linear transformation 적용

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

- W^Q, W^K, W^V는 Query, Key, Value를 변환하는 학습 가능한 가중치 행렬
- W^O는 최종 출력을 위한 가중치 행렬

3.2.3 Applications of Attention in our Model

Transformer에서는 Multi-Head Attention을 다음 세 가지 방식으로 적용

1. **Encoder-Decoder Attention**:

- Query는 Decoder의 이전 레이어에서 가져오고, Key와 Value는 Encoder의 출력을 사용
- 이를 통해 Decoder가 입력 시퀀스 전체를 참고할 수 있도록 설계됨 (기존 Seq2Seq Attention과 유사)

2. **Self-Attention in Encoder**:

- Query, Key, Value가 동일한 입력을 사용
- Encoder의 각 위치가 이전 레이어(참고. 첫 번째 Encoder 레이어는 입력 임베딩을 사용하지만, 이후 레이어들은 이전 레이어의 출력을 입력으로 받음)의 모든 위치를 참고할 수 있도록 설계됨

3. **Self-Attention in Decoder**:

- Query, Key, Value가 동일한 입력을 사용하되, 미래의 정보는 참조하지 않도록 Masking 적용

- 이를 통해 Auto-Regressive 특성을 유지하며, 현재 위치까지의 정보만을 기반으로 학습 가능

3.3 Position-wise Feed-Forward Networks

Transformer의 Encoder 및 Decoder의 각 레이어에는 Self-Attention Layer 외에도 Feed Forward Network (FFN)이 존재

- Feed Forward Network은 **각 단어(토큰)에 독립적으로 적용**되며, 동일한 구조를 가짐

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

- W_1, W_2 : 학습 가능한 가중치 행렬
- b_1, b_2 : 편향
- $max(0,)$: ReLU 활성화 함수

3.4 Embeddings and Softmax

Embeddings (임베딩)

- Transformer는 입력 토큰과 출력 토큰을 d_{model} 차원의 벡터로 변환하기 위해 learned embeddings을 사용
- 각 단어(토큰)은 고정된 차원의 연속 벡터로 변환 (d_{model}=512)
- 디코더 출력은 선형 변환과 Softmax를 통해 다음 토큰의 확률 분포로 변환
- 입력 및 출력 임베딩은 가중치를 공유하여 메모리 사용량을 줄이고, 학습 효율성을 높임
- **가중치 공유 (Weight Sharing):**
  - Input 임베딩과 Output 임베딩의 가중치 행렬을 공유
  - 또한, Pre-Softmax Transformation에서도 동일한 가중치를 사용
- 임베딩 레이어에서는 가중치에 $\sqrt{d_{model}}$을 곱하여 스케일

3.5 Positional Encoding

Transformer은 RNN이나 CNN을 사용하지 않기 때문에, input data의 sequence order 인식 불가능

⇒ 해결책: Positional Encoding을 도입하여 각 단어의 상대적 또는 절대적 위치 정보를 모델에 추가

- 위치 인코딩은 임베딩과 동일한 차원을 가지며, 입력 벡터에 더하는 방식으로 적용
- 다양한 위치 인코딩 방법이 있지만, Transformer는 sine 및 cosine 함수를 사용한 fixed positional encoding 채택
    - 상대적 위치 학습 용이성: 특정 위치 간의 상대적 차이를 모델이 쉽게 학습 가능
    - 일반화 가능성: 학습 중 관찰되지 않은 더 긴 시퀀스에 대해서도 잘 작동 가능

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

- pos: 단어의 위치
- i: 임베딩 벡터의 차원에서의 인덱스
- d_{model} : 임베딩 차원 수

김정찬

4. Why Attention?

(1) Computational Complexity

한 layer을 처리하는 데 필요한 총 계산량이 RNN과 달리 self-Attention은 모든 입력 위치에서 모든 다른 위치로의 연결이 한 번에 이루어지기 때문에 병렬 처리가 가능하여 계산 시간에서 유리하다. 예를 들어,

$$Score(Q, K) = QK^T$$

에서 입력 길이에 대해 모든 쌍을 한 번에 계산할 수 있어 GPU와 같은 병렬 하드웨어에서 효율적으로 처리할 수 있다. 이에 반해, RNN의 경우 시간 스텝마다 이전 출력이 필요하기에 이전 시간 단계의 출력이 다음 단계 입력에 종속되기 때문에 병렬화가 어렵다. 또한, CNN의 경우 커널의 크기 k가 입력 시퀀스의 길이 n보다 작은 경우, 모든 입력 위치 간의 연결을 형성하려면 여러 층의 합성곱이 필요하다.

$$RNN : O(n \cdot d^2)$$
$$Self - Attention : O(n^2 \cdot d)$$

(2) Path Length for Long-Range Dependencies

Long-range dependency란 멀리 떨어진 입력 간의 의존성을 학습하는 것을 의미한다. 또, 네트워크가 이런 의존성을 잘 학습하기 위해서는 입력에서 출력으로 정보가 전달되는 경로의 길이가 짧을수록 좋습니다. RNN는 경로의 길이가 O(n)로 길어질 수 있지만, Self-Attention은 모든 위치가 직접 연결되어 있기 때문에 경로 길이가 상수 O(1)로 짧다.

(3) Self-Attention vs Convolution Layers

CNN의 경우 커널의 크기 k가 입력 시퀀스의 길이 n보다 작은 경우, 모든 입력 위치 간의 연결을 형성하려면 여러 층의 합성곱이 필요하다. 또, 이를 처리하기 위해 CNN은 O(n/k)의 층이 필요하기에 일반적으로 계산복잡도가 계산 복잡도가 O(k·n·d)로 병렬화가 가능한 self-attention에 비해 계산량이 많다. 더 나아가, dilated convolution을 사용할 경우 O(log_k(n))의 경로를, Separable Convolution을 사용할 경우 계산 복잡도가 O(k · n · d + n · d^2)로 줄긴 하지만, 여전히 Self-Attention과 동등한 수준으로 효율적이지는 않다.

(4) Interpretability

Self-Attention은 각 입력이 어느 부분에 주목하고 있는지를 직접적으로 시각화할 수 있어 모델의 해석 가능성을 높이며
개별 Attention Head가 문장의 구문 구조에 따라 서로 다른 역할을 하는 것을 관찰할 수 있다.

5. Training Regime

(1) Training Data and Batching

- Dataset: WMT 2014 English-German Dataset (450만 개의 문장 쌍), WMT 2014 English-French Dataset (3,600만 개의 문장 쌍)
- Byte-Pair Encoding (BPE): 각 Dataset 별로 약 37,000개, 32,000개의 단어 조각 (word-piece) 토큰화가 이루어졌음
- Batching: 각 배치는 약 25,000개의 source tokens와 target tokens를 포함하여 효율적인 GPU 메모리 사용을 보장

(2) Hardware and Schedule

- 8개의 **NVIDIA P100 GPU**
- Base model: 100,000 steps for 12hrs, Big model: 300,000 steps for 3.5days

- Optimizer: Adam, Learning Rate: 학습 초기에는 학습률을 선형적으로 증가시키고, 이후에는 스텝 수의 제곱근에 반비례하여 감소

$$lrate = d^{0.5} \cdot min(stepnum^{-0.5}, stepnum \cdot warmupsteps^{-1.5})$$

(d: 모델의 임베딩 차원, warmupsteps = 4000)

(3) Residual Dropout

- Dropout 확률 = 0.1

(4) Label Smoothing

- 라벨 스무딩 기법을 도입하여 모델이 학습 시 불확실성을 더 잘 다루도록 함.
- 스무딩 값 ϵ = 0.1
- 그 결과, 모델의 perplexity를 약간 증가시키지만, BLEU 점수를 향상시킴.
  - Perplexity (혼란도)

    낮을수록 모델이 예측을 잘하고 있다는 의미이며
    Perplexity가 10이라는 것은 모델이 평균적으로 10개의 단어 중에서 하나를 선택하는 수준으로 혼란스럽다는 뜻
  - BLEU (Bilingual Evaluation Understudy Score)

    모델이 번역한 결과물이 reference translation과 얼마나 유사한지를 평가하는 지표이며 0에서 1 사이의 값을 가지며, 1에 가까울수록 좋은 번역을 의미

6. Result

(1) 번역 Task 결과

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Base 모델의 경우, 마지막 5개의 체크포인트를 평균 내어 최종 모델을 생성
- Big 모델의 경우, 마지막 20개의 체크포인트를 평균 내어 최종 모델을 만듦
- Beam Search Size = 4 (기계 번역, 텍스트 생성 등에서 가장 가능성이 높은 출력 시퀀스를 찾기 위한 탐색 알고리즘으로 매 단계에서 가장 높은 확률의 후보 4개를 선택하여 다음 단어 예측에 사용하겠다는 의미!)
- Length Penalty α = 0.6 (Beam Search는 보통 시퀀스 확률의 곱을 기준으로 최적의 후보를 선택하기 때문에 짧은 문장이 곱셈의 특성상 더 높은 확률을 가지는 경향이 있어 이를 방지하기 위해 번역된 문장의 길이에 대해 패널티를 적용하여 보다 적절한 번역이 선택되도록 함)

$$LP(Y) = (\frac{5 + |Y|}{5 + 1})^{\alpha}$$

- Maximum Output Length: +50 (기계 번역에서는 입력 문장과 비슷한 길이로 번역되는 것이 일반적이지만, 일부 경우 **비정상적으로 긴 번역**이 나올 수 있어 이를 방지하기 위해 **최대 출력 길이**를 설정하여 번역이 무한히 확장되는 것을 막음)

(2) Model Variations

	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

- 변수

N	Transformer의 인코더와 디코더에 사용된 레이어의 개수
d_model	입력 임베딩 및 각 레이어의 출력 벡터의 차원
d_ff	각 레이어의 내부 FFN의 차원
h	Multihead Attention에서 사용되는 병렬적인 attention head의 개수
d_k	Attention에서 Query와 Key의 벡터 크기
d_v	Attention에서 Value 벡터의 크기
P_drop	각 레이어에서 드롭아웃이 적용되는 확률
ϵ_ls	정답 확률 분포에 스무딩을 적용하여 모델이 확률적으로 덜 확신하도록 하여 일반화 성능을 향상
Train steps	각 모델이 학습에 사용된 총 훈련 스텝
PPL (dev)	개발 데이터셋에 대해 계산된 언어 모델의 혼란도
BLEU (dev)	개발 데이터셋에 대해 번역 품질을 나타내는 지표
Params (×10^6\times)	모델이 학습하는 총 파라미터의 수를 나타내며, 파라미터가 많을수록 모델이 더 크고 복잡

- 실험 세부 내용

(A) h, d_k, d_v 의 크기를 조정하여 멀티-헤드 어텐션의 영향을 테스트

(B) Feed-forward 네트워크d_ff의 차원을 변경하여 FFN의 모델 성능에 미치는 영향을 테스트

(C) 레이어 수 N와 모델 차원 d_model을 조정하여 모델 크기와 성능의 관계를 분석

(D) 드롭아웃 확률 P_drop을 변경하여 과적합 방지의 효과를 테스트

(E) Positional encoding을 사인 함수 대신 다른 방법으로 설정하여 임베딩 방식의 변화를 평가

(3) English Constituency Parsing

Constituency Parsing은 출력 구조가 고정되지 않고 문장의 구문적 구조에 따라 강력한 제약이 존재해 출력 길이(구문 트리)가 입력보다 훨씬 길어질 수 있기 때문에 일반적인 시퀀스-시퀀스 모델이 잘 작동하기 어려움.

→ 4개의 레이어(layer)로 구성된 Transformer와 d_model = 1024로 설정하고,

Wall Street Journal (WSJ) Penn Treebank 데이터셋에서 약 40,000개의 훈련 문장과 BerkeleyParser로 구성된 약 1700만 개의 문장을 추가로 사용(Semi-supervised setting)하여 WSJ Section 23의 F1 점수를 기준으로 Transformer 모델의 성능을 비교

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

⇒ Transformer는 특정 작업에 대한 튜닝 없이도 English Constituency Parsing에서 높은 성능을 보여 일반화 능력이 뛰어나다는 점을 입증, semi-supervised 설정에서 특히 뛰어남!

장건호

7. Conclusion

- 완전히 Attention Mechanism에 기반한 시퀀스 변환 모델
- RNN → Multi head self-attention
- RNN, CNN보다 훨씬 빠른 학습 속도

미래 연구 방향

- 텍스트 외 입력 및 출력(이미지, 오디오, 비디오)로 확장
- local 혹은 restricted attention mechanism 연구 → 효율적 데이터 처리
- auto-regressive → less-sequential 생성 방식 연구, 속도 개선