[2주차] 논문리뷰

1. Introduction

논문에서 다루고 있는 주제가 무엇인지와 해당 주제의 필요성이 무엇인가

논문에서 제안하는 방법이 기존 방법의 문제점에 대응되도록 제안 되었는가

- 기존의 Sequence Transduction 모델들은 RNN 또는 CNN을 기반으로 하며 보통 Encoder와 Decoder로 구성됨. 최고의 성능을 보이는 모델들은 attention으로 인코더와 디코더를 연결.
- 기존의 구조를 대체할 신경망 구조인 Transformer 제안
  → attention 메커니즘에만 기반함
- 기존 모델들보다 더 높은 번역 품질, 병렬 처리 용이, 훈련 시간 감소

논문이 다루는 분야

- machine translation (기계 번역)

- sequence transduction (시퀀스 변환)

해당 task에서 기존 연구 한계점

- 기존의 RNN, LSTM, GRU 등이 sequence modeling and transduction problems에서 뛰어난 성과를 보이고 있음
- But, recurrent 모델은 이전 결과를 입력으로 받는 순차적 특성으로 인해 병렬 처리에서 문제가 생김 (training example 내에서의 parallelization-병렬화가 불가능)
  → ht(현재 시점)를 계산할 때, 이전 시점의 ht-1을 알아야만 계산할 수 있으므로 모든 시점을 동시에(병렬적으로) 계산할 수 없고, 단계적으로(순차적으로) 계산을 진행해야 한다는 의미
- 최근 연구에서 *factorization tricks*과 *conditional computation*을 통해 연산 효율을 높였지만, 위와 같은 순차적 계산의 근본적 제약은 사라지지 않음

논문의 contributions

- attention mechanism은 인풋 아풋 시퀀스 거리와 상관없이 의존성을 모델링할 수 있으나, 순환 네트워크와 같이 쓰여졌음.
- *In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output.*

  → 순환 구조를 없애고 온전히 attention mechanism만 이용해서 인풋 아웃풋 사이의 global dependencies를 가능하게 함.

2. Related Work

Introduction에서 언급한 기존 연구들에 대해 어떻게 서술하는가

제안 방법의 차별성을 어떻게 표현하고 있는가

- 기존 연구 : Extended Neural GPU, ByteNet, ConvS2S 모두 시퀀스 연산 줄이기를 목표로 했었음 (모두 CNN을 basic building block으로 사용)
  → hidden representation을 병렬로 계산
  → 입력과 출력 위치 간 관계 모델링하는 방식이 거리에 따라 더 많은 연산을 필요로 함. (여러 개의 합성곱 필터를 거쳐야 함.)

→ ConvS2S 모델에서는 선형적으로 연산 증가
→ ByteNet 모델에서는 로그 형태로 연산 증가

- Transformer : 연산이 constant number of operations로 줄어들음
  → 이 과정에서 effective resolution이 감소하는 문제가 발생
  (averaging attention-weighted positions로 인해)
  → Multi-Head Attention으로 보완
- Self-attention : 단일 시퀀스 안에서 서로 다른 위치에 있는 요소들을 연과나지어 해당 시퀀스의 representation을 계산하는 매커니즘
  → 독해, 추상적인 요약, 맥락적 의미, 작업 무관한 문장 표현 학습 등과 같은 과제에 성공적으로 사용됨
  → Transformer 또한 이를 활용하여 모든 단어가 서로 직접 연결될 수 있도록 했음
- End-to-end memory network : sequence-aligned recurrence가 아닌 recurrent attention mechanism를 기반으로 하는 모델
  → 간단한 언어 질의응답 및 언어 모델링을 잘 함
  →
  ??? 이 친구와 Transformer의 차이점은 무엇인가?
  : End-to-End는 명시적 메모리 구조를 사용 / 재귀적 어텐션
  : Transformer는 Multi-Head Self-Attention을 통해 정보를 학습 / 병렬적 어텐션

3. 제안 방법론

💡 Introduction에서 언급된 내용과 동일하게 작성되어 있는가

Introduction에서 언급한 제안 방법이 가지는 장점에 대한 근거가 있는가

제안 방법에 대한 설명이 구현 가능하도록 작성되어 있는가

Main Idea

- 경쟁력 있는 neural sequence transduction 모델은 encoder-decoder 구조 가짐
- Transformer는 전반적인 아키텍처에 대해 stacked self-attention 과 point-wise, fullyconnected layers를 사용함

▼ **[Transformer] 개요 (교재)**

- 기존의 순환 신경망과 같은 순차적 방식이 아닌 병렬로 입력 시퀀스를 처리하는 기능. 긴 시퀀스의 경우 순환 신경망 모델보다 훨씬 더 빠르고 효율적으로 처리

- 입력 토큰 간의 관계를 직접 처리하고 이해할 수 있도록 하는 self-attention을 기반으로 함. → 재귀나 합성곱 없이 입력 토큰 간 관계 직접 모델링
- 대용량 데이터셋에서 매우 효율적, 기계 번역과 같은 작업에 적합
- 자연어 처리 분야에서 널리 사용되고 영향력이 큰 모델

- 어텐션 메커니즘만을 사용하여 시퀀스 임베딩을 표현한다.
- 입력 시퀀스의 중요한 부분에 초점을 맞추어 문맥을 이해하고 적절한 출력을 생성한다.
- 인코더 : 입력 시퀀스를 임베딩하여 고차원 벡터로 변환
- 디코더 : 인코더의 출력을 입력으로 받아 출력 시퀀스를 생성
- 어텐션 메커니즘 : 인코더와 디코더 단어 사이의 상관관계를 계산하여 중요한 정보에 집중
  → 입력 시퀀스 각 단어가 출력 시퀀스의 어떤 단어와 관련이 있는지 파악하여 번역, 요약문 생성과 관련된 작업 등을 수행할 수 있게 함

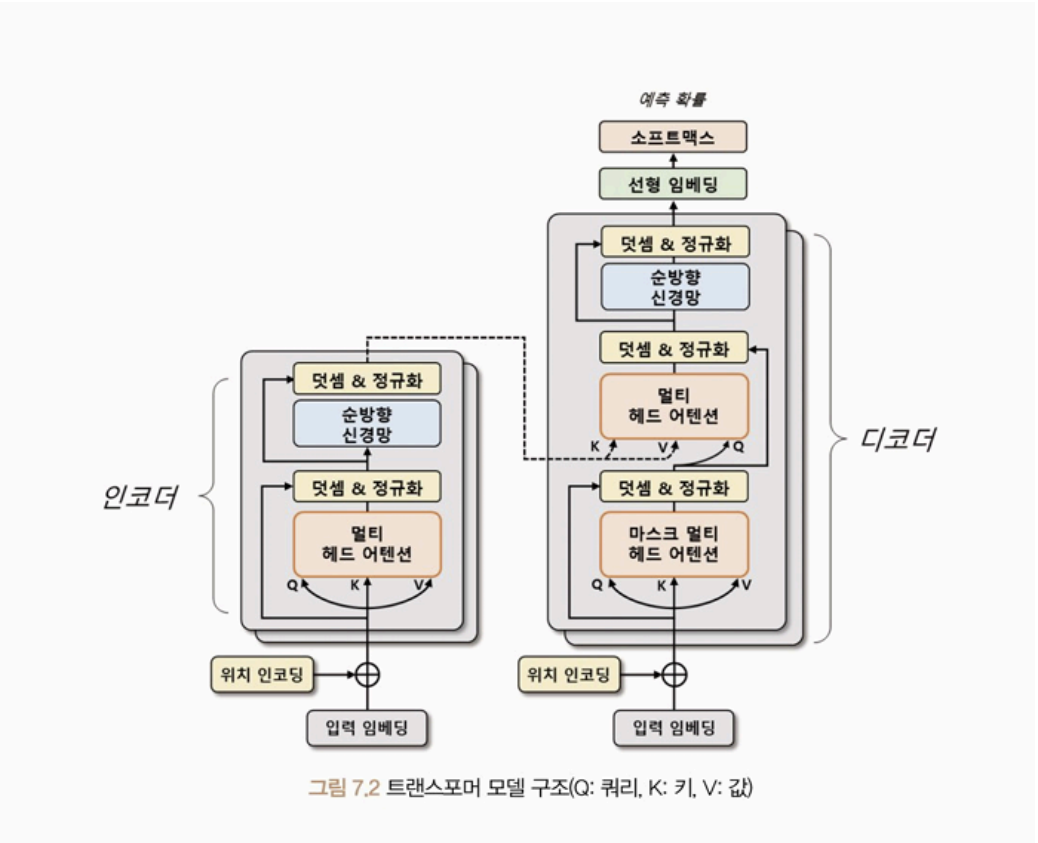


그림 7.2 트랜스포머 모델 구조(Q: 쿼리, K: 키, V: 값)

- N개의 트랜스포머 블록으로 구성됨

- **멀티 헤드 어텐션 & 순방향 신경망**으로 이루어져 있음
- 멀티 헤드 어텐션 : 입력 시퀀스에서 쿼리, 키, 값 벡터를 정의해 입력 시퀀스들의 관계를 셀프 어텐션하는 벡터 표현 방법 - 쿼리와 각 키의 유사도를 계산, 해당 유사도를 가중치로 사용하여 값 벡터를 합산
  → 계산된 어텐션 행렬은 입력 시퀀스 각 단어의 임베딩 벡터를 대체함
- 순방향 신경망 : 산출된 임베딩 벡터를 더욱 고도화. 여러 개의 선형 계층으로 구성돼 있으며, 앞선 순방향 신경망의 구조와 동일하게 입력 벡터에 가중치를 곱하고 편향을 더하며 활성화 함수를 적용함.
  → 학습된 가중치들은 입력 시퀀스의 각 단어의 의미를 잘 파악할 수 있는 방식으로 갱신됨.
- 입력 시퀀스 데이터를 소스와 타깃 데이터로 나눠 처리
- 인코더는 소스 시퀀스 데이터를 위치 인코딩된 입력 임베딩으로 표현
  → 트랜스포머 블록의 출력 벡터를 생성
- 디코더는 마스크 멀티 헤드 어텐션을 사용해서 타깃 시퀀스 데이터를 순차적으로 생성. 인코더 출력 벡터 정보 참조하여 디코더 입력 시퀀스들 간 관계를 고도화
- 최종적으로 생성된 디코더 출력 벡터는 선형 임베딩으로 재표현되어 이미지나 자연어 모델에 활용.
- 위치 인코딩 : 입력 시퀀스의 순서 정보를 모델에 전달하는 방법. 삼각함수 사용해 생성됨 이를 통해 임베딩 벡터와 위치 정보 결합된 최종 입력 벡터를 생성함
  → 단어의 순서 정보 학습
- 특수 토큰 : 단어 토큰 이외의 특수 토큰 활용. 입력 시퀀스의 시작과 끝을 나타내거나 마스킹(일부 입력 무시) 영역으로 사용됨
  → BOS 시작 EOS 끝 UNK 모르는 단어 PAD 공백 채우기

그림 7.3 특수 토큰을 활용한 문장 토큰 배열

- 문장 토큰 배열을 어휘 사전 위치에 원-핫 인코딩으로 표현 → 입력 임베딩으로 변환 (Word2Vec 방법과 동일)트랜스포머 인코더

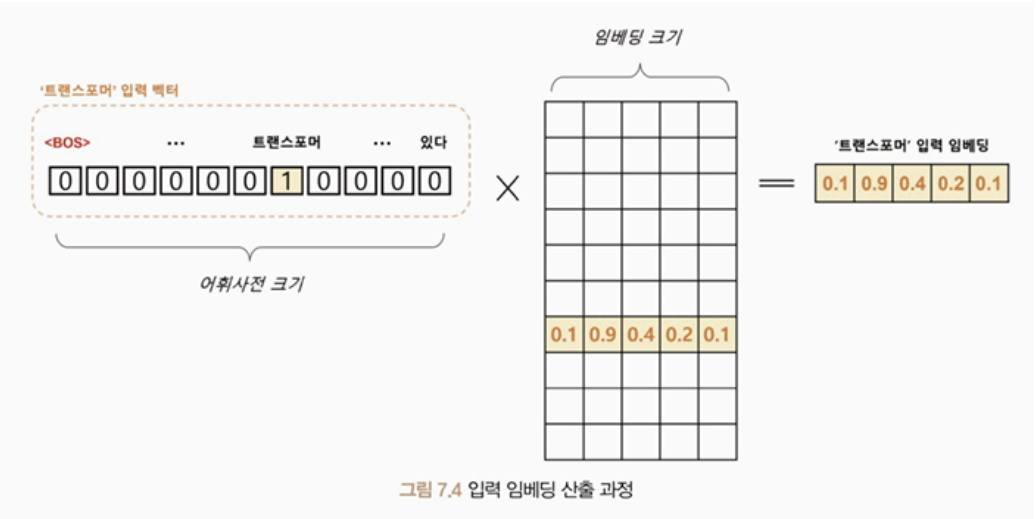


그림 7.4 입력 임베딩 산출 과정

▼ Encoder

- a stack of N = 6 identical layers
- 각 층에는 2개의 서브층이 있음
  → multi-head self-attention
  → feed-forward network
  → 서로 residual connection으로 연결

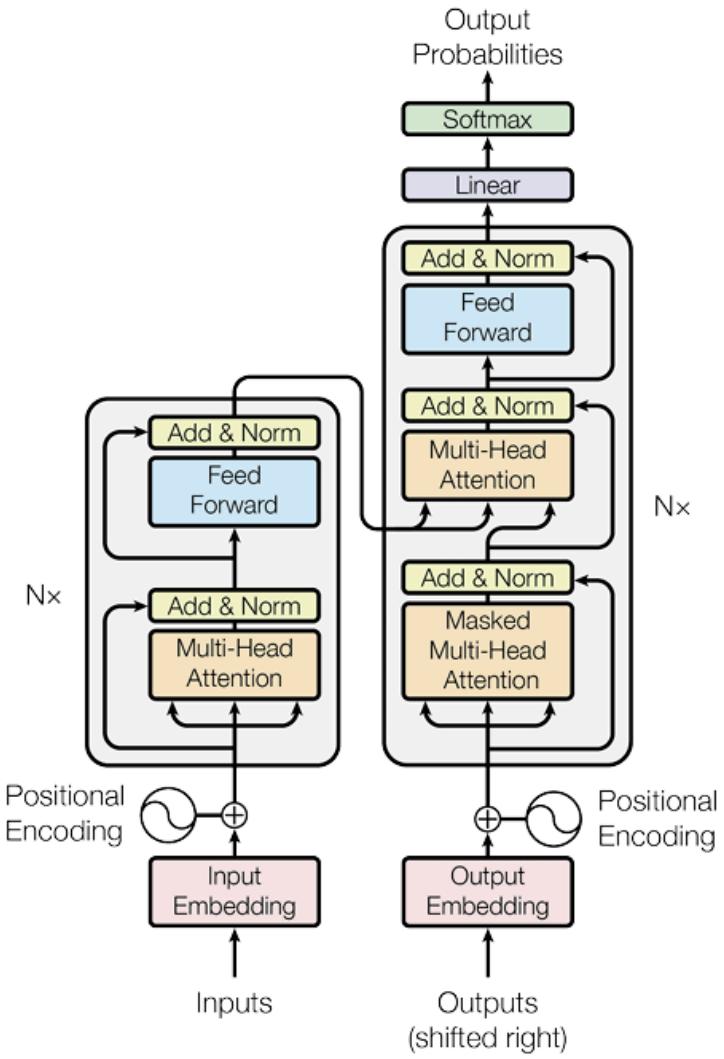


Figure 1: The Transformer - model architecture.

▼ **트랜스포머 인코더 (교재)**

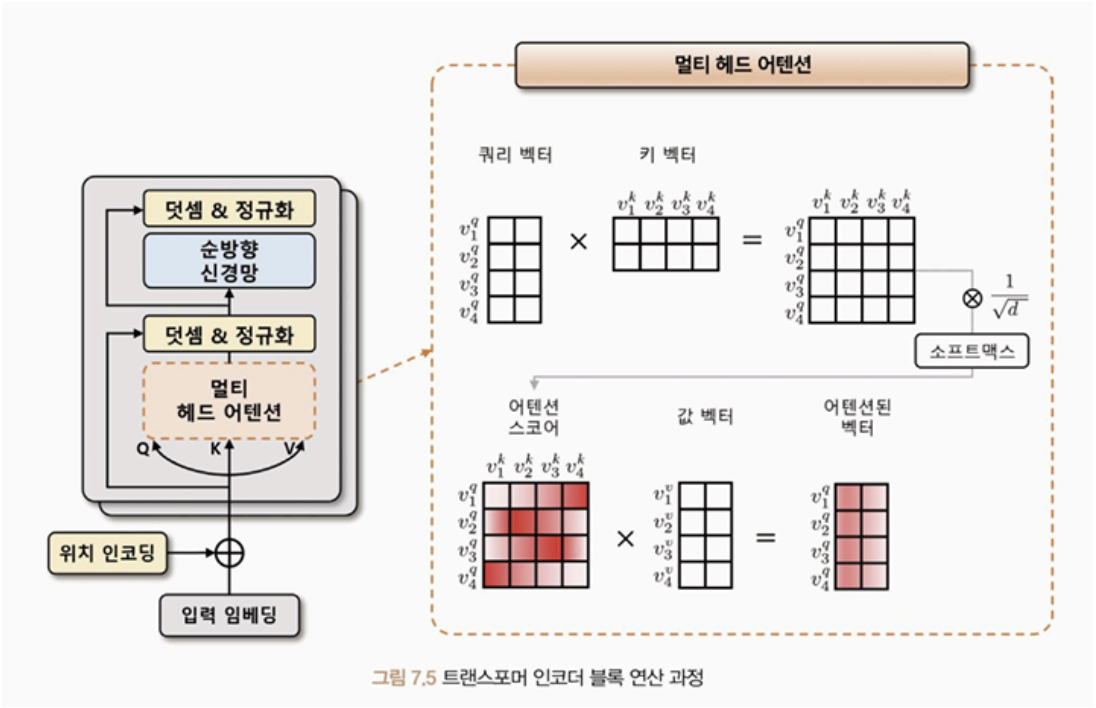


그림 7.5 트랜스포머 인코더 블록 연산 과정

- 입력 시퀀스 받아서 인코더 계층 거쳐서 연산 수행
- 각 계층은 멀티 헤드 어텐션과 순방향 신경망으로 구성
- 위치 정보 반영 위해 임베딩 벡터를 입력 벡터에 더해줌
- 쿼리 벡터 : 현재 시점에서 참조하고자 하는 정보의 위치를 나타내는 벡터, 인코더의 각 시점마다 생성
- 키 벡터 : 쿼리 벡터와 비교되는 대상으로 쿼리 벡터를 제외한 입력 시퀀스에서 탐색되는 벡터, 인코더의 각 시점에서 생성
- 값 벡터 : 쿼리 벡터와 키 벡터로 생성된 어텐션 스코어를 얼마나 반영할지 설정하는 가중치 역할

수식 7.2 어텐션 스코어 계산식

$$score(v^q, v^k) = \text{softmax}\left(\frac{(v^q)^T \cdot v^k}{\sqrt{d}}\right)$$

- root(d)는 보정값. 벡터 차원이 커질 때 스코어값이 같이 커지는 문제 완화 / 보정된 값을 softmax로 확률적으로 표현하고 이걸 벡터랑 내적해서 셀프 어텐션된 벡터를 생성 - 이 과정을 반복해서 셀프 어텐션된 스코어 맵 생성
- 멀티 헤드 : 셀프 어텐션을 여러 번 수행해 여러개의 헤드를 만듬. 각 헤드가 독립적으로 어텐션을 수행하고 그 결과를 합침

▼ Decoder

- a stack of N = 6 identical layers
- 각 층에는 3개의 서브층이 있음
  → Self-Attention layer
  → Feed Forward Neural Network
  → 하나 더 있음 (Multi head Attention 수행)
  → 하위 계층끼지 residual connectino, 계층 정규화 진행

▼ **트랜스포머 디코더 (교재)**

- 위치 인코딩이 적용된 타깃 데이터의 입력 임베딩을 입력받음
  → 디코더가 입력 시퀀스의 순서 정보를 학습할 수 있게 됨
- 인과성을 반영한 마스크 멀티 헤드 어텐션 모듈이 사용됨
  → 어텐션 스코어 맵을 계산할 때 첫 번째 쿼리 벡터가 첫 번째 키 벡터만을 바라볼 수 있게 마스크를 씌움. 두 번째는 첫번째와 두 번째만 볼 수 있게 → 인과성이 보장되는 방식 (셀프 어텐션 방지)
  → 마스크 영역에는 수치적으로 굉장히 적은 값을 더해주고 softmax 계산 시 어텐션 가중치가 0이되어 정보참조 X

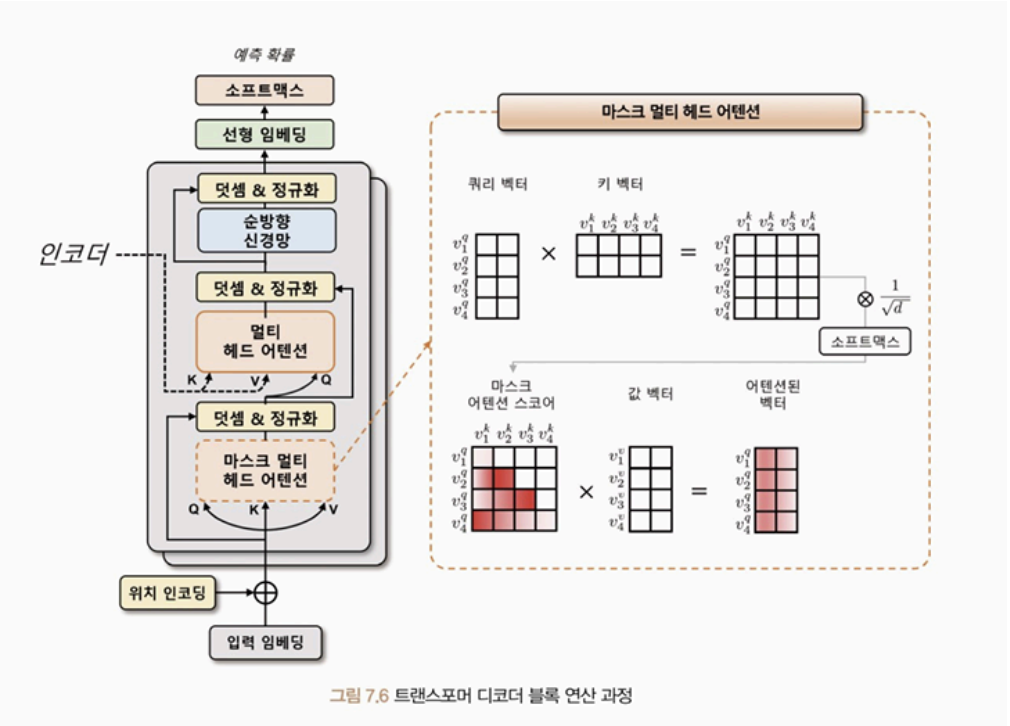


그림 7.6 트랜스포머 디코더 블록 연산 과정

- 타깃 데이터가 쿼리 벡터로 사용됨
  인코더의 소스 데이터가 키와 값 벡터로 사용됨.
  → 쿼리벡터는 타깃데이터 위치정보 포함한 입력 임베딩과 위치 인코딩을 더한 벡터가 됨

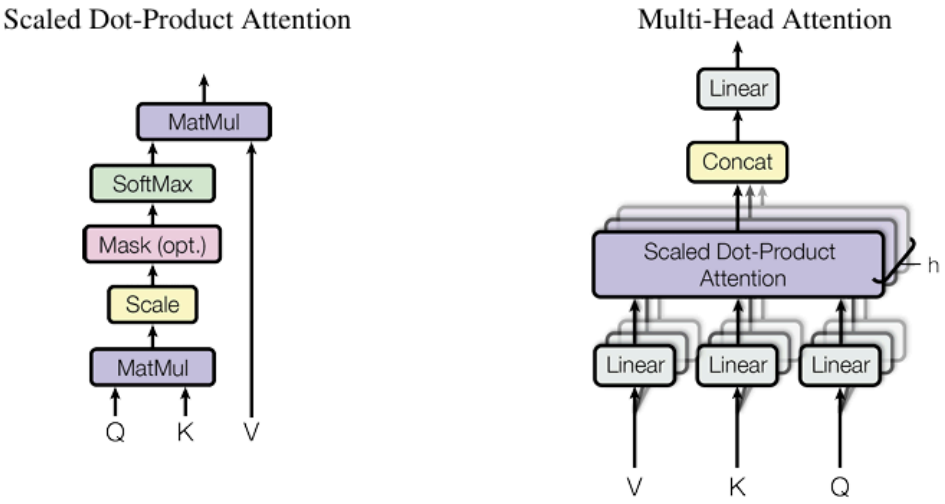
    - 스코어 맵 계산 → softmax 적용 가중치 구함
      → 어텐션 가중치와 값 벡터 가중합하여 멀티 헤드 어텐션의 출력 벡터를 얻음
    - 마지막 디코더 블록의 출력 텐서에 선형 변환 및 소프트맥스 함수를 적용해 각 타깃 시퀀스 위치마다 예측 확률을 계산할 수 있게 됨.
    - 타깃 데이터를 추론할 때 토큰 또는 단어를 순차적으로 생성시킴

- **Attention**

  → 쿼리와 키밸류 페어를 아웃풋에 매핑시키는 것

- **Scaled Cot-Product Attention**

  → 특정 어텐션을 이렇게 지칭함.



- 쿼리 세트에 대한 어텐션을 동시에 계산 (수식 설명은 encoder > 교재 정리 참고) → ??? 수식에 대한 더 명확한 이해 필요

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

    - 워드 임베딩에 가중치를 곱해서 쿼리, 키, 밸류 계산
    - 쿼리 키 곱한 게 어텐션 스코어 : 값이 높을수록 연관성 높음
    - 키 차원수로 나누고(규모 측면) softmax 적용

- Multi-Head Attention

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

  - 다양한 공간으로 projection 시키는 방식 사용하는 듯 ???(맞나)
  - 모델이 다양한 관점에서 문장을 해석할 수 있도록 하는 역할
  - Head(8개) 만큼 Scaled dot product Attention 연산 수행
    → 다양한 관점의 어텐션맵 만듦
- **Applications of Attention in our Model**
  1. 이전 decoder 레이어에서 오는 쿼리들과 encoder의 출력으로 나오는 memory key, value들과의 attention
  2. encoder의 각 위치들은 이전 레이어의 모든 위치들을 참조
  3. decoder의 각 위치들은 decoder 내의 본인 이전까지의 위치들을 참조
- **Position-wise Feed-Forward Networks**

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

  - 어텐션 층과 함께 fully connected feed-forward network 사용됨
  - 사이에 ReLU 활성화가 있는 두 개의 선형 변환으로 구성
- **Embeddings and Softmax**
  - input, ouput token embedding layer를 거쳐서 사용
  - input embedding 과 output embedding의 가중치 매트릭스 쉐어함
- **Positional Encoding**

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

  - 토큰의 상대적인 위치에 대한 정보를 제공하기 위한 역할

- sin cos 함수 이용 (???구체적인 수식 이해가 안됨)

Contribution

- 각 레이어의 연산 복잡도 감소
- recurrence 없앰으로써 더 많은 양을 병렬 처리함
- Long-term dependency에 대한 효율적인 처리

4. 실험 및 결과

💡 Introduction에서 언급한 제안 방법의 장점을 검증하기 위한 실험이 있는가

Dataset

- WMT 2014 English-German dataset
  - consisting of about 4.5 million sentence pairs
  - byte-pair encoding / source-target vocabulary of about 37000 tokens
- WMT 2014 English-French dataset
  - consisting of 36M sentences
  - split tokens into a 32000 word-piece vocabulary
- **Hardware and Schedule**
  - 8 NVIDIA P100 GPUs
  - base model(step took about 0.4 seconds) - 100,000 steps or 12 hours
  - big models(step time was 1.0 seconds) - trained for 300,000 steps (3.5 days)
- **Optimizer**
  - Adam optimizer
  - learning rate은 아래의 식에 따라 바꾸었음 (warmup step = 4000)

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

- **Regularization**

- Residual Dropout : 각 서브층 결과에 대해 dropout 수행한 후 residual connection 수행 (Pdrop = 01)

Baseline

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

결과

▼ Machine Translation

- On the WMT2014English-to-German translation task
  - (big) SOTA 달성함 → 이전의 성능 좋았던 모델보다 BLEU가 2.0 더 높음 (28.4)
  - base 모델조차 이전의 모델들을 능가함. training cost도 경쟁력 있음
- On the WMT2014English-to-French translation task
  - BELU 41.0 달성함. 이전 SOTA 모델보다 성능은 뛰어난데 계산량은 1/4임
- averaging the checkpoints의 의미
  → 훈련 과정에서 저장된 여러 개의 모델 가중치를 평균 내어 최종 모델을 만드는 기법
  → 훈련 후반부의 변동이 적은 상태에서 안정적인 모델 만들기 위함.
  → overfitting 방지의 효과도 있음
  → (base) 마지막 5개 체크포인트 평균내
  → (big) 마지막 20개 체크포인트 평균냄
- Beam size = 4 & Length penalty = 0.6 (실험을 통해 최적값 찾음)
  - Beam Search

    - 가장 높은 확율을 가진 문장을 찾기 위해 여러 개의 후보를 동시에 탐색하는 방법 (순차적 문제에서 가장 가능성이 높은 출력을 찾기 위한 탐색 알고리즘)
    - Beam Size : 탐색할 후보 개수 (후보 유지하면서 단어 생성)
- maximum output length : input length + 50

▼ Model Variations (English-to-German에 대해)

	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) head의 개수, dk.dv
- (B) dk - 어텐션 키 사이즈 줄이면 모델 성능 안좋아짐
- (C) 모델의 크기 키움 - 성능 좋아짐
- (D) dropout - 오버피팅 피하기에 좋음
- (E) sinusoidal positional encoding → learned positional embeddings → 거의 동일한 결과를 보임

5. 결론 (배운점)

💡 연구의 의의 및 한계점, 본인이 생각하는 좋았던/아쉬웠던 점 (배운점)

- 연구의 의의

    - Attention 메커니즘만을 사용하는 최초의 시퀀스 변환 모델
    - 순환 신경망을 완전히 대체하여 병렬 연산이 가능하게 함
    - 속도도 빠르고 성능도 훨씬 좋아짐
- 본인이 생각하는 좋았던/아쉬웠던 점 (배운점)
    - 어텐션 메커니즘 하나에만 집중해서 새로운 네트워크를 형성했다는 것이 인상적
    - 사람이 언어를 번역하는 메커니즘을 고려하고 기술에 적용할 수 있도록 생각해보는 방향도 좋을 것 같음

6. 기타

공부 참고

- https://www.youtube.com/watch?v=p216tTVxues
- https://www.youtube.com/watch?v=AA621UofTUA&t=13s
- https://velog.io/@qtly_u/Attention-is-All-You-Need-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0
- https://velog.io/@tobigs-nlp/Attention-is-All-You-Need-Transformer