Team2_25_W_NLP_WEEK4_복습과제(team-paper-review)

상태	완료
담당자	이 이예지[학부재학 / 통계학과] 예일 예일 이 Monator

Paper Review - Attention Is All You Need

작성자: 이예지

1. Introduction

RNN, LSTM, GRU는 언어 모델링과 기계 번역과 같은 태스크에서 SOTA 모델로 자리 잡았고, 이후 순환 언어 모델과 인코더-디코더 아키텍처에 관한 연구가 지속됨.

순환 모델은 일반적으로 입력 및 출력 시퀀스의 각 위치에서 연산을 수행하며, 이전 은닉 상태 h_{t-1}과 해당 위치의 입력 x_t을 바탕으로 새로운 은닉 상태 h_t를 생성. 그러나 이러한 순차적 특성으로 인해 훈련 데이터 내에서 병렬화가 어려우며, 긴 시퀀스의 경우 메모리 제약으로 인해 배치 크기가 제한되는 문제가 발생.

어텐션 매커니즘은 시퀀스 내 거리와 관계 없이 종속성을 모델링 할 수 있으나, 일반적으로는 순환 네트워크와 함께 사용됨.

본 연구에서는 순환 연산을 배제하고 어텐션 매커니즘에 의존하는 Transformer 모델을 제안.

2. Background

순차 연산을 줄이기 위해 Extended Neural GPU, ByteNet, ConvS2S와 같은 모델이 등장. 이들은 CNN을 기반으로 입출력 위치의 은닉 표현을 병렬로 계산하지만, 두 위치 간 신호를 연결하기 위해 필요한 연산 수가 위치 간 거리와 비례하여 증가하기 때문에 먼 위치 간의 종속성을 학습하는 데 어려움을 초래. Transformer에서는 이러한 연산을 일정 수로 줄였으나, 해상도가 감소하는 문제가 있었으며, 이를 해결하기 위해 Multi-Head Attention을 도입.

Self-Attention은 시퀀스 내 위치 관계를 모델링하여 시퀀스 표현을 계산하는 매커니즘이며, 다양한 NLP 태스크에서 성공적으로 활용됨.

End-to-End 메모리 네트워크는 순환 어텐션 매커니즘을 사용하여 간단한 언어 질문 응답 및 언어 모델링 태스크에서 좋은 성능을 보임.

Transformer는 sequence aligned RNN이나 convolution 없이 입출력 표현을 Self-Attention만으로 계산하는 최초의 transduction 모델.

3. Model Architecture

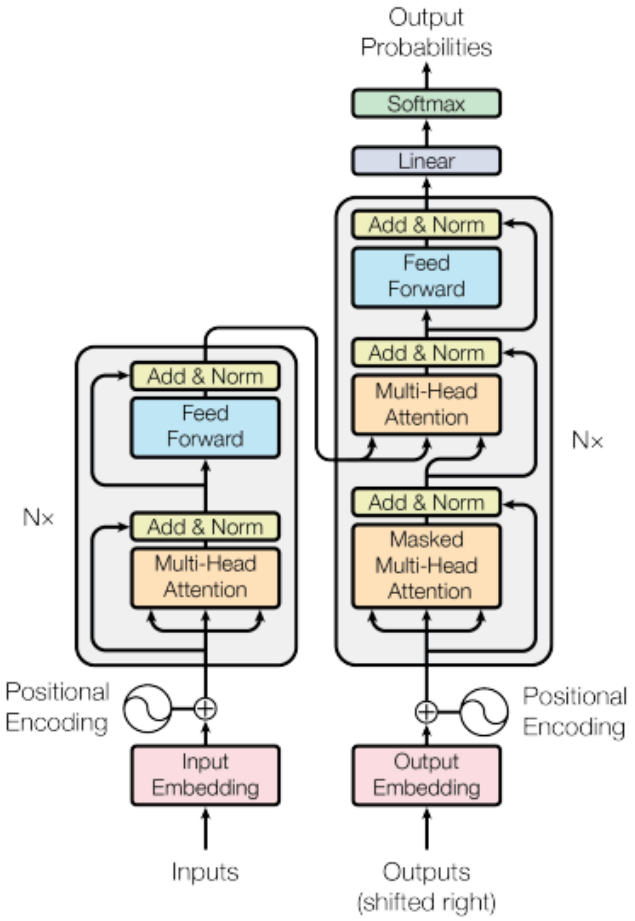

Figure 1: The Transformer - model architecture.

대부분의 신경망 기반 시퀀스 변환 모델은 인코더-디코더 구조를 사용하며, 인코더는 입력 시퀀스를 연속적인 표현으로 매핑하고 디코더는 이를 기반으로 출력 시퀀스를 auto-regressive 방식으로 생성. Transformer는 이러한 구조를 따르면서 인코더와 디코더에 스택 형태의 Self-Attention과 point-wise fully connected layers를 활용.

3.1 Encoder and Decoder Stacks

Encoder: 동일한 6개의 레이어로 구성된 스택. 각 레이어는 두 개의 서브 레이어(Multi-Head Attention + Position-wise Fully Connected Feed-Forward Network)로 구성됨. 각 서브 레이어에는 Residual Connection과 Layer Normalization이 적용됨. 즉, 각 서브 레이어의 출력은 $LayerNorm(x + Sublayer(x))$이며, $Sublayer(x)$는 해당 서브 레이어의 연산을 의미. 모델의 모든 서브 레이어와 임베딩 레이어는 512차원의 출력 생성.

Decoder: 동일한 6개의 레이어로 구성된 스택. 인코더 레이어와 동일한 두 개의 서브 레이어 + 세 번째 서브 레이어(인코더 스택의 출력을 활용한 Multi-Head Attention) 포함. 인코더와 동일하게 Residual Connection과 Layer Normalization 사용. Self-attention 서브

레이어는 미래 위치를 참조하지 않도록 masking 수행. Masking 기법과 디코더의 출력 임베딩을 한 칸씩 밀어 적용하는 방식을 결합하여 i번째 위치의 예측이 i보다 작은 위치의 출력에만 의존하도록 함.

3.2 Attention

Attention 함수는 Query와 Key-Value Pairs를 입력으로 받아 Output을 생성하는 방식으로 작동. Output은 각 Value의 weighted sum으로 계산됨. 각 Value에 할당되는 Weight는 Query와 해당 Key의 유사도를 측정하는 Compatibility Function을 통해 결정됨.

3.2.1 Scaled Dot-Product Attention

Query와 모든 Key 간의 내적 계산 → $\sqrt{d_k}$으로 나누어 정규화 → Softmax 함수를 적용하여 Weight 생성 → Weight를 Value 행렬에 적용하여 최종 Output 생성

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

일반적인 어텐션 방식에는 Additive Attention과 Dot-Product Attention(스케일링 적용 X)이 존재하며, 유사한 복잡도를 가지지만 Dot-Product Attention이 연산 속도가 훨씬 빠르고 메모리 효율성이 높음. d_k가 작은 경우 두 방식의 성능 차이가 크지 않지만, d_k가 커지면 스케일링을 적용하지 않은 Dot-Product Attention은 softmax 함수의 작은 gradient로 인해 성능 저하가 일어남. 이를 방지하기 위해 $\sqrt{d_k}$을 활용하여 정규화 수행.

3.2.2 Multi-Head Attention

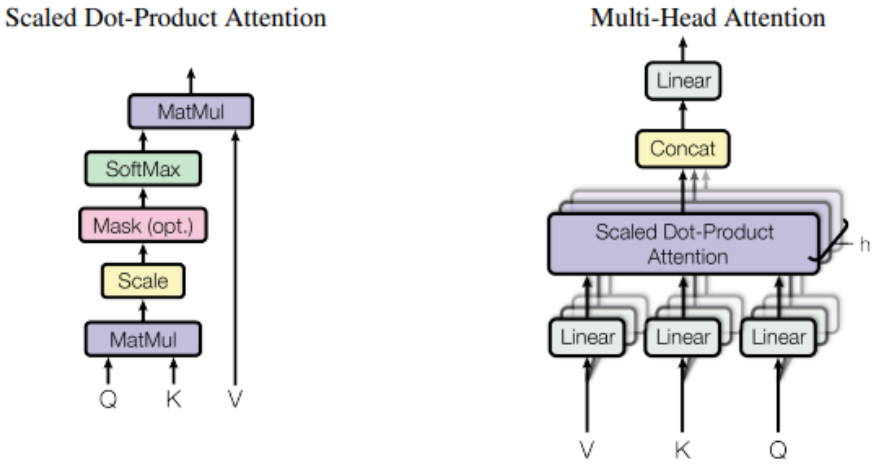

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

기존 Single-Head Attention은 정보 표현력이 부족하다는 문제점 존재. ⇒ Query, Key, Value를 서로 다른 subspaces로 투영한 후, 여러 개의 어텐션을 병렬적으로 수행하는 Multi-Head Attention 적용.

Query, Key, Value를 각각 h개로 분할. → 각 헤드마다 다른, 학습된 Weight Matrix를 사용하여 Projection 수행. → 각 헤드에서 독립적으로 Attention 수행. → Attention 결과를 Concat 후, Projection 수행.

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

W_i^Q, W_i^K, W_i^V는 각각 Query, Key, Value의 Projection을 위한 Trainable Weight Matrix.

W^O는 모든 Attention 출력을 하나로 합치는 역할을 하는 Weight Matrix.

작성자: 이예일

3.2.3 Applications of attention in our model

transformer의 multi-head attention 세 가지 방식

- **인코더-디코더 어텐션 (Encoder-Decoder Attention)**
  - 쿼리(queries)는 디코더의 이전 레이어에서 가져오고, 키(keys)와 값(values)은 인코더의 출력을 사용합니다.
  - 이를 통해 디코더의 각 위치가 **입력 시퀀스 전체를 참고할 수 있도록** 합니다.
  - 이는 기존의 **시퀀스-투-시퀀스(seq2seq) 모델**에서 사용되던 인코더-디코더 어텐션 방식과 유사합니다.
- **인코더의 셀프 어텐션 (Self-Attention in Encoder)**
  - 인코더의 셀프 어텐션에서는 **쿼리, 키, 값이 모두 동일한 입력을 사용**합니다.
  - 즉, 인코더의 각 위치는 **이전 레이어의 모든 위치를 참고할 수 있습니다.**
- **디코더의 셀프 어텐션 (Self-Attention in Decoder)**
  - 디코더의 셀프 어텐션도 **쿼리, 키, 값이 동일한 입력을 사용**합니다.
  - 하지만, **디코더의 현재 위치까지의 정보만 참고할 수 있도록 설계**되어 있습니다.

- 이를 위해 **미래 정보를 가리는 마스킹(Masking)을 적용**하여, softmax 입력에서 잘못된 연결에 해당하는 값을 -∞로 설정합니다.
- 이렇게 하면 **왼쪽에서 오른쪽으로의 순차적인 생성(auto-regressive property)이 보장**됩니다.

3.3 Position-wise Feed Forward Networks

Feed Forward Network(FFN)의 동작 방식 및 특징

FFN의 동작 방식

- FFN은 **각 위치(Position)별로 독립적이고 동일하게 적용**됩니다.
- 두 개의 선형 변환(Linear Transformation)과 중간의 ReLU 활성화 함수(ReLU Activation)로 구성됩니다.
- 수식은 다음과 같습니다:FFN(x)=max(0,xW1+b1)W2+b2

  FFN(x)=max(0,xW1+b1)W2+b2FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2

FFN의 특징

1. **위치 독립성**
   - FFN은 **각 위치에서 개별적으로 계산**됩니다.
   - 따라서, 특정 위치의 피드포워드 네트워크 계산이 다른 위치의 계산과 독립적입니다.
2. **레이어별 가중치 변화**
   - FFN의 두 개의 선형 변환(가중치 행렬)은 **각 층마다 다른 파라미터를 사용**합니다.
   - 하지만 **같은 층 내에서는 모든 위치에서 동일한 가중치를 공유**합니다.
3. **CNN과의 유사성**
   - 이 연산은 **커널 크기가 1인 CNN(1×1 Convolution)과 유사**합니다.(즉, 위치별로 독립적인 변환을 수행하는 것과 동일한 효과)

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \qquad (2)$$

3.4 Embeddings and Softmax

Transformer의 임베딩과 출력 변환

- **입력 및 출력 토큰 임베딩**

- Transformer는 학습된 임베딩(learned embeddings)을 사용하여 입력 토큰과 출력 토큰을 모델의 차원 벡터로 변환합니다.
- 즉, 각 단어(또는 토큰)가 고정된 차원의 **연속 벡터**로 표현됩니다.

- **디코더의 출력 변환**
  - 디코더에서 최종적으로 나온 벡터는 **학습된 선형 변환(linear transformation)과 softmax 함수**를 거쳐 **다음 토큰의 확률 분포**로 변환됩니다.
  - Softmax를 사용하여 다음에 등장할 단어를 예측하는 확률을 계산합니다.
- **가중치 공유 (Weight Sharing)**
  - **입력 임베딩과 출력 임베딩의 가중치를 공유**합니다.
  - 또한, **출력 변환을 위한 선형 레이어(Pre-Softmax Transformation)도 동일한 가중치를 사용**합니다.
  - 이는 기존 연구에서 사용된 방식과 유사합니다.
- **임베딩 스케일링**
  - 임베딩 레이어에서는 가중치 행렬을 루트(모델의 차원)로 나눠 스케일링합니다.
  - 이는 모델이 초기 단계에서 너무 작은 값에 의해 학습이 어려워지는 문제를 방지하기 위한 기법입니다.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

테이블 1 요약

- **Self-Attention**
  - 모든 단어가 **한 번의 연산**으로 서로 연결 가능
  - **병렬 처리 가능(**빠른 연산 수행 가능)
  - **시퀀스 길이가 길어질수록 계산량 증가**
- **Recurrent (RNN 기반 모델)**

    - **이전 상태를 기반으로 다음 상태를 계산**해야 해서 **순차적 연산 필수**
    - **장기 의존성 문제 존재**(정보가 멀리 갈수록 소실 가능)
    - 병렬화가 어렵고, 시퀀스 길이가 길면 연산이 느려짐
- **Convolutional (CNN 기반 모델)**
    - 지역적인 정보를 효과적으로 학습 (n-gram과 비슷한 구조)
    - **병렬 처리 가능**(빠른 연산 가능)
    - **장기 의존성 학습에는 한계**

  (멀리 떨어진 단어를 연결하기 어려움)
- **Restricted Self-Attention**
    - 일반 Self-Attention보다 **연산량을 줄이기 위해 일부 토큰만 주목**
    - 병렬 처리 가능하지만, **멀리 떨어진 단어 간 관계를 잡기 어려울 수 있음**

3.5 Positional Encoding

- Transformer에는 순환(recurrence)과 합성곱(convolution)이 없으므로, 시퀀스 내에서 토큰의 순서를 활용하기 위해 Positional Encoding(위치 인코딩)을 추가함.
- 위치 인코딩은 **입력 임베딩과 동일한 차원**(dmodel)을 가지며, 이를 임베딩 벡터에 더함.
- 위치 인코딩 방식에는 **학습된(learned) 방식과 고정된(fixed) 방식**이 있음.
- 사인(sin) 및 코사인(cos) 기반 위치 인코딩 공식

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

- 위치 pos와 차원 인덱스 i에 대해 서로 다른 주기의 사인 및 코사인 함수 적용.
- **위치 인코딩의 각 차원은 하나의 사인 곡선에 해당하며**, 주파수는 2π에서 10000·2π까지 기하급수적으로 변화함.
- **이 방식을 선택한 이유**: 상대적 위치 정보를 학습하기 쉬우며, 특정 오프셋 k에 대해 선형 변환으로 표현 가능.

- **학습된 위치 임베딩과 비교**
  - 학습된 위치 임베딩을 실험했으나, 성능 차이가 거의 없음(Table 3, Row E 참고).
  - **사인/코사인 방식이 더 긴 시퀀스에서도 일반화 가능성이 높아 채택됨.**

4. Why Self-Attention

Transformer의 **Self-Attention 레이어**를 기존의 **Recurrent (RNN) 및 Convolutional (CNN) 레이어**와 비교하며, 다음 세 가지 기준을 고려함:

1. **레이어별 전체 계산 복잡도 (Computational Complexity per Layer)**
2. **병렬화 가능한 연산량 (Parallelization, 최소 순차 연산 수)**
3. **장거리 의존성 학습을 위한 최대 경로 길이 (Path Length for Long-Range Dependencies)**

Self-Attention vs. RNN

- Self-Attention은 모든 위치 간 연결을 일정한 수의 연산(O(1) sequential operations)으로 수행할 수 있음.
- 반면, Recurrent Layer는 O(n) 개의 순차 연산이 필요함.
- 계산 복잡도 측면에서 **시퀀스 길이 nnn이 표현 차원 ddd보다 작을 경우**, Self-Attention이 RNN보다 빠름.
- 대부분의 최신 번역 모델(State-of-the-Art MT)에서 사용되는 WordPiece 및 Byte-Pair Encoding(BPE)에서는 이 조건이 충족

Self-Attention vs. CNN

- 단일 CNN 레이어(커널 크기 k<n)는 **입력과 출력의 모든 위치를 직접 연결하지 못함**.
- 이를 해결하려면 **연속적인 커널을 사용하면 O(n/k)O(n/k)O(n/k) 개의 CNN 레이어를 쌓아야 하며**, dilated convolutions(팽창 합성곱)을 사용할 경우 경로 길이는 O(logk(n))이 됨.
- Convolutional Layers는 일반적으로 **Recurrent Layers보다 연산량이 더 많음 (kkk 배 차이)**.
- **Separable Convolutions**를 사용하면 복잡도를 O(k·n·d+n·d2)로 줄일 수 있음.
- 하지만 **k=nk = nk=n 인 경우에도, Self-Attention과 Feed-Forward Layer를 합친 방식과 동일한 연산량이 필요**함.

Self-Attention의 장점 및 확장 가능성

- **장거리 의존성 학습(Long-Range Dependency Learning)에 유리**→ Self-Attention의 최대 경로 길이(Max Path Length)는 O(1)로, 장거리 의존성을 쉽게 학습할 수 있음.
- **입력 시퀀스가 매우 길 경우**, Self-Attention을 제한된 영역(r)에서만 적용하는 방법을 고려 가능→ 이 경우 **최대 경로 길이가 O(n/r)O(n/r)O(n/r) 로 증가하지만, 계산량을 줄일 수 있음**.
- **모델 해석 가능성(Interpretability)이 높음**→ **어텐션 가중치(Attention Distribution)를 시각화하면 개별 어텐션 헤드가 문장의 문법적·의미적 구조를 학습하는 방식이 보임**

5. Training

5.1 Training Data and Batching

- **영어-독일어(English-German) 학습 데이터**:
  - **WMT 2014 데이터셋** 사용 (약 **450만 문장 쌍** 포함)
  - **Byte-Pair Encoding (BPE)** 적용하여 토큰화
  - **소스-타겟 공유 어휘(vocabulary) 크기**: 약 **37,000개 토큰**
- **영어-프랑스어(English-French) 학습 데이터**:
  - **WMT 2014 데이터셋** 사용 (약 **3,600만 문장** 포함, 규모가 더 큼)
  - **WordPiece 토큰화** 적용
  - **어휘 크기**: 약 **32,000개 토큰**
- **배치 구성 방식**:
  - 문장 길이를 기준으로 **유사한 길이의 문장 쌍을 그룹화(batch grouping)**
  - **각 배치당 약 25,000개의 소스 토큰과 25,000개의 타겟 토큰 포함**

5.2 Hardware and Schedule

- **GPU 환경**:
  - **8개의 NVIDIA P100 GPU**에서 모델 학습 진행
- **Base 모델 학습**:
  - 논문에서 설명된 **기본 하이퍼파라미터 사용**

    - **각 학습 스텝당 0.4초 소요**
    - **총 100,000 스텝 학습 → 약 12시간 소요**
- **Big 모델 학습**:
    - Table 3의 **하단(Big Model) 설정 사용**
    - **각 학습 스텝당 1.0초 소요**
    - **총 300,000 스텝 학습 → 약 3.5일 소요**

5.3 Optimizer

- **Optimizer**: Adam optimizer 사용
    - **β1 = 0.9, β2 = 0.98, ϵ = 10^{-9}**
- **Learning Rate 스케줄링 공식**:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (3)$$

- 초반 **warmup 단계에서는 선형적으로 증가**
- 이후 **스텝 수의 역제곱근에 따라 감소**
- **Warmup 스텝 수**:
    - **warmup_steps = 4000** 설정

5.4 Regularization

- **Residual Dropout**
    - 각 **서브레이어(sub-layer) 출력에 Dropout 적용** 후, 이를 입력에 더한 후 정규화.
    - 인코더 및 디코더 스택에서 **임베딩과 위치 인코딩 합산 결과에도 Dropout 적용**.
    - **Base 모델에서 Dropout 비율 Pdrop=0.1P_{drop} = 0.1Pdrop=0.1 사용**.
- **Label Smoothing**
    - 학습 시 **라벨 스무딩(Label Smoothing) 값 ϵls=0.1\epsilon_{ls} = 0.1ϵls=0.1 적용**.
    - 이는 모델이 **과도하게 확신(confidence)하지 않도록 유도**.
    - 결과적으로 **Perplexity(혼란도)는 증가하지만, BLEU 점수와 정확도는 향상됨**.

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 [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**	$2.3 \cdot 10^{19}$	

표 요약

- **Transformer 모델이 기존 모델 대비 BLEU 점수에서 우수한 성능을 보임**
  - **Transformer (Big)** 모델은 **EN-DE: 28.4, EN-FR: 41.8**로 가장 높은 BLEU 점수를 기록.
  - 특히 **EN-FR**에서는 **ConvS2S Ensemble(41.29)보다도 높은 성능**을 보임.
- **Transformer는 기존 모델 대비 낮은 연산 비용으로 높은 성능을 달성**
  - **Transformer (Base)의 FLOPs**:
    - **EN-DE: 3.3×10183.3 \times 10^{18}3.3×1018, EN-FR: 2.3×10192.3 \times 10^{19}2.3×1019**
    - 기존 **GNMT + RL(2.3×10192.3 \times 10^{19}2.3×1019, 1.4×10201.4 \times 10^{20}1.4×1020)보다 훨씬 적은 연산량**으로 더 높은 성능을 보임.
  - **Transformer (Big)도 비교적 적은 연산량으로 높은 성능을 달성**
- **Ensemble 모델과 비교 시, 단일 Transformer (Big) 모델도 경쟁력 있음**
  - 기존 **GNMT + RL Ensemble(41.16)과 ConvS2S Ensemble(41.29)** 보다 **Transformer (Big)이 유사하거나 더 높은 성능을 보임.**
  - 하지만 **Ensemble 모델은 연산 비용이 훨씬 큼** (예: ConvS2S Ensemble은 1.2×1021 FLOPs).
  - 즉, **Transformer는 단일 모델로도 Ensemble 수준의 성능을 달성하면서 연산 비용은 훨씬 낮음.**

작성자: 이영서

6. Results

6.1 Machine Translation

- Task 1: WMT 2014 English-to-German
  - Transformer(big)이 기존 모델들 능가
  - 28.4 BLEU (타 모델 대비 2.0 이상의 성능)
  - 학습 디테일: 3.5 days on 8 P100 GPUs.
  - Transformer(base model) 또한 기존 모델들보다 성능이 높게 나타남
- Task 2: WMT 2014 English-to-French
  - Transformer(big)이 기존 모델들 능가
  - 41.0 BLEU (기존 best model 대비 1/4 비용)
  - 학습 디테일: droupout rate = 0.1
- Transformer 학습 디테일
  - base model은 마지막 5개의 체크포인트 평균, big model은 마지막 20개의 체크포인트 평균
  - 빔 서치(beam search) 사용: beam size = 4, length penalty α= 0.6
  - Inference 최대 출력 길이: 입력 길이 + 50, 가능 시 조기 종료

6.2 Model Variations

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

Transformer 구성 요소의 중요성 평가하기 위해, base model을 여러 방식으로 변형하여 EN-DE 태스크 성능 측정.

- Table 3 실험을 통해 알아낸 점
  - 단일 Attention head는 최상의 설정보다 0.9 BLEU 성능 저하, But head가 너무 많아도 성능 저하
  - Attention key의 크기를 줄이면 모델 품질 저하 → 호환성 계산 쉽지 않음
  - 모델 크기가 클수록 성능 좋아짐
  - Dropout이 과적합 방지에 매우 도움됨
  - 사인 함수 기반의 위치 인코딩 대신 학습된 위치 Embedding 사용해도 결과는 거의 동일

6.3 English Constituency Parsing

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

Transformer 일반화 성능 관찰 위해, 영어 구문 분석 실험 수행. 해당 태스크는 단순 번역보다 출력의 구조가 복잡해서 수행하기 더 어려움.

- 실험 디테일
  - 데이터셋: WSJ 데이터셋(4만) + BerkleyParser 코퍼스(1,700만 개, for 반지도 학습)
  - d_model=1024인 4-Layer Transformer를 사용
  - 영어-독일어 번역 모델의 기본 설정을 그대로 유지하되
    - 최대 출력 길이: 입력 길이 + 300
    - Beam Size는 21, 길이 페널티 α = 0.3
- 실험 결과
  - Recurrent Neural Network Grammar를 제외한 모든 이전 모델보다 더 나은 결과
  - RNN 시퀀스-투-시퀀스 모델과는 달리, Transformer는 WSJ 훈련 세트 40,000 문장만으로도 BerkeleyParser보다 더 좋은 성능 발휘

7. Conclusion

- Transformer의 의의
  - Attention 매커니즘만을 사용하여 시퀀스 변환을 수행하는 최초의 모델
  - Recurrent Layer를 Multi-Head Self-Attention로 대체
  - 번역 작업에서 Recurrent, Convolutional Layor 기반 아키텍처보다 훨씬 빠르게 훈련
    - EN-DE 태스크에서는 최상의 모델이 모든 앙상블 모델도 능가