Attention Is All You Need

Ashish Vaswani et al., NIPS, 2017

translated by. triplet02

0. 요약(Abstract)

지배적인 시퀀스 변환 모델들은 인코더와 디코더를 포함하는 순환신경망(recurrent neural network), 혹은 컨볼루션 신경망(convolutional neural network)에 기반한다. 최고 성능의 모델들 또한 어텐션 매커니즘을 적용한 인코더와 디코더를 연결한 것들이다. 우리는 트랜스포머(Transformer)라는, 오로지 어텐션 매커니즘에만 기반한 새롭고 단순한 신경망 구조를 제안하며, 이는 순환신경망과 컨볼루션 신경망을 완전히 대체할 것이다. 두 기계 번역 과제에서 보인 실험 결과, 이 모델들은 더 용이하게 병렬화가 가능하며 현격히 짧은 학습 시간을 필요로 하므로 기존 모델에 비하여 강점을 가진다. 본 모델은 WMT 2014 영-독 번역 과제에서 28.4 BLEU를 기록하였고, 이는 앙상블을 통해 달성했던 지난 기록을 2 BLEU 더 개선한 것이다. WMT 2014 영-프 번역 과제에서는 8개의 GPU로 3.5일간 학습시킨 결과 단일 모델로서는 최고(state-of-the-art)인 41.0 BLEU를 기록하였고, 이는 타 논문의 최고 모델들과 비교하면 현저하게 적은 학습 비용이다.

1. 도입(Introduction)

순환신경망(recurrent neural network), 장기 단기기억(long short-term memory)[12], 그리고 게이트가 포함된 순환신경망(gated recurrent network)[7]은 언어 모델링과 기계 번역과 같은 시퀀스 모델링 및 변환 과제에서 확고부동한 state-of-the-art 접근법으로써 취급되었다[29, 2, 5]. 순환식 언어 모델(recurrent language model)과 인코더-디코더 구조의 경계를 넓히기 위한 많은 노력들이 현재까지도 계속 이어져 왔다[31, 21, 13].

순환식 모델들은 일반적으로 입력(input)과 출력(output) 시퀀스의 심볼(symbol) 위치에 따라 인자를 계산한다. 계산 시점에 따라 위지를 정렬하며, 모델은 은닉 상태(hidden state) h_t를, 이전의 은닉 상태 h_{t-1}과 위치 t의 입력에 대한 함수로써 생성한다. 이러한 본질적으로 연속적인 성질은 학습 예제들에 대한 병렬화를 배제하며, 이는 메모리 한계가 학습 예제들을 배치로 묶는 것을 제한하기 때문에 보다 긴 시퀀스에서 더욱 치명적이다. 최근의 연구는 인자화(factorization) 기법과 조건부 연산(conditional computation)을 통하여 연산 효율성 측면에서 눈에 띄는 개선을 이루어 냈고, 후자의 경우에는 모델 성능의 개선 또한 가져왔다. 하지만 순차적 연산이라는 본질적인 문제는 여전히 남는다.

어텐션 매커니즘은 입력과 출력 시퀀스 안의 거리에 관계 없이 의존도를 모델링 할 수 있게 함으로써 많은 과제에서 설득력 있는 시퀀스 모델링 및 변환 모델을 구축하는 데 필수적인 부분이 되었다[2, 16]. 그러나 거의 모든 경우[22], 어텐션 매커니즘은 순환신경망 구조와 함께 사용된다.

본 연구에서는, 전역적인 의존성을 도출하기 위하여, 순환성(recurrence)을 피하고 대신 어텐션 매커니즘에만 유일하게 의존하는 모델인 트랜스포머를 제안한다. 트랜스포머는 병렬화가 현저하게 편리하며, P100 GPU 8개를 사용한 20시간 정도의 적은 학습으로도 새로운 state-of-the-art급의 번역 성능에 도달할 수 있다.

2. 배경(Background)

순차적 연산을 줄이려는 목표는 Extended Neural GPU[20], ByteNet[15]과 ConvS2S[8]의 포석이 되었고, 이들은 모두 컨볼루션 신경망을 기본으로 하여 모든 입력과 출력 위치에 대하여 은닉 표현(hidden representation)을 병렬로 연산한다. 이러한 모델들에서는 두 임의의 입력과 출력 위치로부터 신호를 연관시키는 데 필요한 연산의 횟수가 둘 사이의 거리에 따라 ConvS2S의 경우는 선형적으로, ByteNet의 경우에는 대수적으로(logarithmically) 증가한다. 이것은 모델로 하여금 떨어진 두 지점 사이의 의존성을 학습하는 것을 어렵게 한다[11]. 트랜스포머 구조에서 이는 상수 횟수의 연산으로 줄어들며, 비록 어텐션에 따라 가중치를 부여한 위치를 평균화하며 실질적인 분해능(effective resolution)이 감소하는 단점이 있지만 이는 3.2장에서 설명한 멀티-헤드 어텐션을 활용하여 대응할 수 있다.

내부 어텐션이라고도 불리는 셀프 어텐션은 시퀀스의 내재적 의미를 연산하기 위하여 한 시퀀스 상의 서로 다른 두 지점을 연관짓는 어텐션 기법이다. 셀프 어텐션은 독해(reading comprehension), 추상적 요약(abstractive summerization), 텍스트 함의 인식(textual entailment), 과제와 무관하게 문장의 의미를 학습하는 등 다양한 분야에서 성공적으로 사용되어 왔다[4, 22, 23, 19].

엔드투엔드 기억 신경망(end-to-end memory network)는 시퀀스 정렬 반복이 아닌 반복 어텐션 매커니즘에 기반하며, 단순 언어 질문에 대한 응답 및 언어 모델링 과제에서 우수한 성능을 발휘하는 것으로 입증되었다[28].

그러나 우리가 아는 한, 트랜스포머는 시퀀스로 정렬된 RNN이나 컨볼루션 없이 입력과 출력을 연산하기 위하여 오직 셀프 어텐션에만 의존한 첫 번째 변환 모델이다. 다음 장부터 트랜스포머를 설명할 것이며, 셀프 어텐션을 사용한 이유와 그 장점에 대하여 [14,15]와 [8]과 같이 논의할 것이다.

3. 모델 구조(Model Architecture)

가장 경쟁력 있는 신경망 변환 모델들은 인코더-디코더 구조를 가진다. 먼저, 인코더는 연속된 심볼 표현인 입력 x = (x_1 , … , x_n)을 연속적인 표현인 z = (z_1 , … , z_n)으로 변환한다. 주어진 z에 대하여, 디코더는 출력 시퀀스인 y = (y_1, … , y_n)의 심볼을 차례로 생성한다. 매 단계에서 모델은 자기회귀적(auto-regressive)이며, 이전 단계에서 생성한 심볼들을 다음 심볼을 생성하는 데 필요한 추가적인 입력으로써 사용한다.

트랜스포머는 **그림 1**의 왼쪽과 오른쪽 절반에 각각 표현된 바와 같이 인코더와 디코더에서 적층 구조의 셀프 어텐션과 점별 완전연결층(point-wise, fully connected layer)를 각각 사용하여 위에서 설명한 전반적인 구조를 따른다.

3.1 다층 인코더 및 디코더(Encoder and Decoder Stacks)

인코더

인코더는 N = 6개의 동일한 계층의 적층 구조로 구성된다. 각 계층은 두 개의 하위계층을 가진다. 첫 번째는 멀티 헤드 셀프 어텐션 매커니즘이고, 두 번째는, 단순한 위치 기반 완전 연결 피드포워드 신경망(position-wise fully connected feed-forward network)이다. 각각의 하위계층에서 잔류 연결(residual connection)[10]을 사용하며, 이후 계층 정규화(layer normalization)를 수행한다[1]. 즉, 각 하위계층의 출력은 $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$이고, $\mathrm{Sublayer}(x)$는 각각의 하위계층에서 수행하는 연산 그 자체이다. 이 잔류 연결을 촉진하기 위하여, 임베딩 계층(embedding layer)을 포함한 모델의 모든 하위계층은 d_{model} = 512 차원의 출력을 생성한다.

디코더

디코더 또한 N = 6개의 동일한 계층의 적층 구조로 구성된다. 인코더의 두 개 계층에 더하여 디코더에는 세 번째 하위계층을 추가하며, 이는 인코더의 출력에 멀티 헤드 어텐션을 적용하는 계층이다. 인코더와 유사하게, 각각의 하위계층에 잔류 연결을 적용하였으며 이후 계층 정규화를 수행한다. 또한 어떤 한 시점(position)의 정보가 그 다음 시점에 관여하는 것을 막기 위해 디코더 내부의 셀프 어텐션 하위계층을 수정하였다. 이 마스킹(masking) 기법에, 출력 임베딩이 어떤 한 시점에 의하여 상쇄된다는 사실이 더해져, 시점 i의 예측이 그보다 이전 시점에 발생한 출력에만 의존한다는 점이 보장된다.

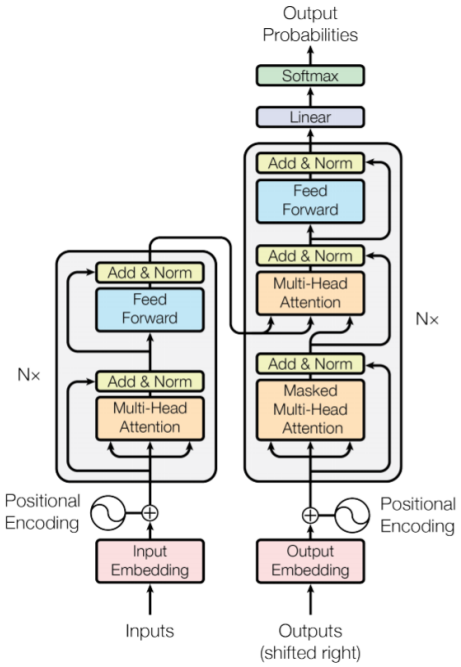

그림 1: 트랜스포머 모델 구조

3.2 어텐션(Attention)

어텐션 함수는 질의(query)와 키-값 쌍(key-value pairs)들의 집합을 출력(output)과 맵핑(mapping)하는 기능을 수행하며, 이때 질의와 키, 값, 그리고 출력은 모두 벡터이다. 출력은 값(value)들의 가중치 합(weighted sum)으로써 계산되며, 이 가중치(weight) 값은 해당하는 키와 질의의 호환성 함수(compatibility function)에 의해 결정된다.

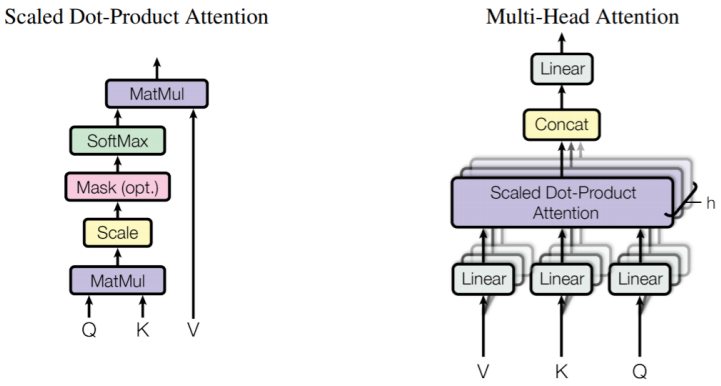

그림 2: (좌) 조정된 내적 어텐션 (우) 병렬로 실행되는 다수의 어텐션 계층으로 구성된 멀티 헤드 어텐션

3.2.1 조정된 내적 어텐션(Scaled Dot-Product Attention)

본 모델에서 수행한 특정한 어텐션을 '조정된 내적 어텐션' 이라고 부른다(**그림 2**). 입력은 d_k차원의 질의와 키, 그리고 d_v차원의 값으로 이루어진다. 모든 키(key)에 대한 질의(query)의 내적을 구하고, 각각을 $\sqrt{d_k}$로 나눈 다음, 값(value)들에 대한 가중치를 얻기 위해 소프트맥스(softmax) 함수를 적용한다.

실제로, 어텐션 함수는 질의를 하나의 집합(set)으로 모은 행렬 Q를 한번에 계산한다. 키와 값들도 역시 하나의 행렬로 모여 K와 V를 형성한다. 이러한 행렬 연산은 다음과 같이 진행된다:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

가장 보편적으로 사용되는 두 가지 어텐션은 덧셈식 어텐션(additive attention)과 내적 어텐션(dot-product attention, 혹은 multi-plicative attention)이다[2]. 내적 어텐션은 $\frac{1}{\sqrt{d_k}}$의 재조정 인자(scaling factor)를 가진다는 점을 제외하면 앞서 설명한 알고리즘과 동일하다. 덧셈식 어텐션은 하나의 은닉 계층(hidden layer)을 가진 피드포워드 신경망을 통해 호환성 함수를 계산한다. 두 어텐션의 이론적 복잡도는 비슷하지만, 내적 어텐션은 고도로 최적화된 행렬곱 연산 코드를 활용하여 구현될 수 있으므로 실제로는 덧셈식 어텐션보다 빠르고 공간 효율적이다.

d_k의 값이 작을 때 두 어텐션은 유사한 성능을 보이지만, d_k의 값이 크면, 재조정하지 않은 내적 어텐션보다 덧셈식 어텐션이 우수한 성능을 보인다[3]. 이는 d_k의 값이 클 경우 내적 자체가 큰 값을 가지게 되고, 이 때문에 소프트맥스 함수의 극도로 작은 그래디언트(gradient)를 보이는 영역으로 떠밀려 가기 때문으로 생각된다.[1] 이 효과에 대처하기 위하여, 내적 값을 $\frac{1}{\sqrt{d_k}}$만큼 재조정 해 준다.

3.2.2 멀티 헤드 어텐션(Multi-Head Attention)

d_{model}차원의 키, 값, 질의를 통해 단일한 어텐션 함수를 수행하는 것 대신, 우리는 서로 다른, 학습된 선형 투영(linear projection)으로 질의, 키 및 값을 각각 d_k, d_k, d_v차원으로 h번 선형 투영하는 것이 좋은 효과를 불러옴을 알게 되었다. 이후 이렇게 각각 투영된 질의, 키, 값으로부터 d_v차원의 출력을 생성하는 어텐션 함수를 병렬로 수행한다. 이 출력들은 서로 연결(concatenate)되고, 한 번 더 투영되어 **그림 2**와 같이 최종 출력이 된다.

멀티 헤드 어텐션은 모델이 서로 다른 시점(position)의 서로 다른 표현 요소(representation subspace)에 동시에 집중(attend)할 수 있도록 한다. 평균화 과정(averaging)때문에, 단일 어텐션 헤드(single attention head)에서는 억제되는 일이다.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

이 때, 투영(projection)은 $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$의 매개변수 행렬이다.

본 연구에서는 $h = 8$의 병렬 어텐션 계층, 즉 head를 사용하였다. 각각의 계층에서 $d_k = d_v = d_{model}/h = 64$이다. 각 헤드에서 차원 수가 감소하기 때문에, 총 연산량은 완전한 차원을 모두 사용하는 단일 헤드 어텐션과 비슷한 수준이다.

---

[1] 왜 내적 값이 커지는지를 알아보기 위하여, 서로 독립이고 평균이 0, 분산이 1인 임의의 각 요소 q와 k를 가정하자. 그들의 내적 $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$는 평균이 0이고 분산이 d_k이다.

3.2.3 본 모델에서 어텐션의 적용(Applications of Attention in our Model)

트랜스포머는 멀티 헤드 어텐션을 세 가지 다른 방식으로 사용한다:

- "인코더-디코더 어텐션" 계층에서, 질의(query)는 이전의 디코더 계층에서 오고, 기억 키(memory key)들과 값(value)들은 인코더의 출력에서 온다. 이것은 디코더의 각 시점(position)에서 입력 시퀀스의 모든 부분에 집중(attend)할 수 있도록 한다. 이것은 [31, 2, 8]과 같은 시퀀스 투 시퀀스(sequence-to-sequence) 모델의 전형적인 인코더-디코더 어텐션 메커니즘을 따온 것이다.
- 인코더는 셀프 어텐션 계층을 가진다. 셀프 어텐션 계층에서 모든 키, 값, 그리고 질의는 같은 곳에서부터 유래하며, 이 경우에는 인코더의 이전 계층의 출력이다. 인코더의 각 시점(position)에서 이전 시점 인코더 출력의 모든 부분에 집중할 수 있다.
- 이와 유사하게, 디코더에 있는 셀프 어텐션 계층은 디코더의 각 시점(position)으로 하여금 그 자신 시점까지의 디코더 모든 시점에 집중할 수 있도록 한다. 디코더의 자기회귀적(auto-regressive) 성질을 보존하기 위해, 이 왼쪽의 정보 흐름을 통제해야 한다. 본 연구에서는 조정된 내적 어텐션의 내부에서, 잘못된 연결에 해당하는 소프트맥스의 입력 값을 마스킹(-∞로 설정) 함으로써 이를 구현하였다. **그림 2**를 보라.

3.3 위치별 피드포워드 신경망(Position-wise Feed-Forward Networks)

어텐션 하위 계층에 더하여, 인코더와 디코더의 각 계층은 각 시점(position)에 따로, 또 동일하게 적용된 완전 연결형 피드포워드 신경망(fully connected feed-forward network)을 포함한다. 이는 사이에 ReLU 활성화(activation) 함수가 있는 두 개의 선형 변환으로 구성된다.

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

각 선형 변환은 서로 다른 위치에서도 동일하게 적용되지만, 계층마다 다른 매개변수를 사용한다. 이 과정을 설명하는 또 다른 방법은 커널 크기(kernel size)가 1인 두 개의 컨볼루션이다. 입력과 출력의 차원은 d_{model} = 512이고, 내부 계층의 차원 d_{ff} = 2048이다.

3.4 임베딩과 소프트맥스(Embeddings and Softmax)

다른 시퀀스 변환 모델들과 유사하게, 입력 토큰과 출력 토큰을 d_{model}차원의 벡터로 변환하는 데 학습된 임베딩을 사용하였다. 또한 일반적인, 학습된 선형 변환(linear transformation)과 소프트맥스 함수를 통해 디코더의 출력을 예상되는 다음 토큰의 확률로 변환한다. 본 모델에서는, [24]와 유사하게, 두 개의 임베딩 계층과 소프트맥스 이전의 선형 변환 계층에서 동일한 가중치 행렬(weight matrix)를 공유한다. 임베딩 계층에서, 이 가중치에 $\sqrt{d_{model}}$을 곱하여 사용한다.

3.5 위치 기반 인코딩(Positional Encoding)

본 모델은 순환성(recurrency)과 컨볼루션을 갖고 있지 않기 때문에, 모델이 시퀀스의 순서에 대한 정보를 사용할 수 있게 하려면 반드시 해당 시퀀스에 포함된 토큰의 상대적인, 혹은 절대적인 위치에 대한 정보를 주입해 주어야 한다. 이를 위해, 우리는 "위치 기반 인코딩" 기법을 인코더와 디코더 적층 구조의 밑바닥에 위치한 입력 임베딩에 적용하였다.

표 1: 서로 다른 계층 종류에 따른 최대 경로 길이, 계층별 복잡도 및 최소 순차적 연산 수. n은 시퀀스의 길이, d는 표현의 차원 수, k는 컨볼루션의 커널 크기이고 r은 제한적인 셀프 어텐션에서 이웃(neighborhood)의 크기이다.

계층 종류	계층별 복잡도	최소 순차적 연산	최대 경로 길이
셀프 어텐션	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
순환(recurrent)	$O(n \cdot d^2)$	$O(n)$	$O(n)$
컨볼루션	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
제한적인 셀프 어텐션	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

위치 기반 인코딩은 임베딩과 같이 d_{model}의 차원 수를 가지므로 둘은 더해질 수 있다. 이외에도 위치 기반 인코딩이 연구되고 수정된 많은 다른 방식들이 있다[8].

본 연구에서는, 서로 다른 주파수를 갖는 사인과 코사인 함수를 사용하였다:

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

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

이 때 pos는 위치(position)이고 i는 차원이다. 즉, 위치 기반 인코딩의 각 차원은 사인 곡선에 해당한다. 그 파장은 2π에서 $10000 \cdot 2\pi$까지의 기하 급수(geometric progression)를 이룬다. 우리는 어떤 고정된 값 k에 대하여 PE_{pos+k}은 PE_{pos}에 대한 일차함수로 나타낼 수 있으므로, 모델이 상대적인 위치에 집중하는 방법을 학습하는 것을 용이하게 해 줄 것이라고 가정하고 이 함수를 선택하였다.

우리는 또한 학습된 위치 기반 임베딩[8]을 대신 적용해 보았고, 두 가지 방식이 거의 유사한 결과를 나타냄을 확인하였다(**표 3**의 (E) 행을 보라). 우리는 모델이 학습 과정에서 접하는 시퀀스 길이보다 더 긴 길이의 시퀀스 역시 추정할 수 있도록 하기 위해서 사인 곡선 방식을 택하였다.

4. 왜 셀프 어텐션인가(Why Self-Attention)

이 장에서 우리는 셀프 어텐션과, 은닉 계층(hidden layer)와 같은 전형적인 시퀀스 변환의 인코더나 디코더와 같이 가변 길이 시퀀스의 심볼 표현(symbol representations) $(x_1, \ldots, x_n)$을 같은 길이의 또 다른 시퀀스 $(z_1, \ldots, z_n)$로 맵핑(mapping)하는 $(x_i, z_i \in \mathbb{R}^d)$ 순환, 컨볼루션 신경망의 다양한 방면을 비교할 것이다. 셀프 어텐션의 사용에 있어서 우리는 세 가지 사항들(desiderata)을 고려한다.

첫 번째는 계층당 총 연산 복잡도이다. 두 번째는 최소로 필요한 순차적 연산의 횟수로써 측정되는 병렬화 가능한 연산의 양이다.

세 번째는 신경망의 원거리 의존성(long-range dependencies) 간의 경로 길이(path length)이다. 원거리 의존성을 학습하는 것은 많은 시퀀스 변환 과제에서 주된 도전거리이다. 그러한 의존성을 학습하는 데 영향을 미치는 주요 요인 중의 하나는 신경망 내에서 신호가 앞으로, 혹은 뒤로 얼마나 긴 경로를 이동해야 하는지이다. 입력과 출력 시퀀스의 임의의 위치 조합 사이의 경로가 짧을수록, 원거리 의존성을 학습하는 것이 용이해진다[11]. 때문에 우리는 계층의 종류별로 입력과 출력 임의의 두 위치 사이의 최대 경로 길이 또한 비교할 것이다.

표 1에서 확인할 수 있는 바와 같이 셀프 어텐션 계층은 상수 횟수의 순차적 연산을 통해 모든 다른 위치들과 연결되어 있는 반면, 순환신경망 계층은 $O(n)$번의 순차적 연산이 필요하다. 연산 복잡도의 측면에서 고려하면, 시퀀스 길이인 n이 표현 차원의 수인 d보다 작은 경우에 셀프 어텐션 계층이 순환신경망 계층보다 빠르며, 이

러한 경우는 기계 번역 분야의 state-of-the-art 모델들이 사용하는 워드 피스(word-piece) 표현[31]이나 바이트 페어(byte-pair) 표현[25]에서는 흔하다. 매우 긴 시퀀스를 다루는 작업에서의 연산 성능을 향상시키기 위해, 셀프 어텐션은 출력에서의 위치를 중심으로 크기 r의 이웃, 즉 인접 영역의 입력 시퀀스만을 고려하도록 제한될 수 있다. 이 방식은 최대 경로 길이를 $O(n/r)$로 증가시킨다. 이후 이 접근 방식에 대하여 더 자세히 연구할 계획이다.

커널 폭이 $k < n$인 단일 컨볼루션 계층은 입력과 출력 위치의 모든 쌍을 연결하지 못한다. 모든 쌍을 연결하기 위해서 연속 커널(contiguous kernel)의 경우 $O(n/k)$개의 계층을 쌓아올려야 하고, 확장 커널(dilated kernel)의 경우 $O(log_k(n))$개의 계층이 필요하며 두 경우 모두 신경망 내부 임의의 두 지점을 잇는 최장 경로의 길이는 증가한다. 컨볼루션 계층은 보통 순환식 계층보다 k배로 비용이 많이 든다(expensive). 반면 분리 가능한 컨볼루션(separable convolution)[6]은 복잡도를 $O(k \cdot n \cdot d + n \cdot d^2)$까지 현저하게 낮춘다. 심지어 $k = n$인 상황에서의 복잡도 또한 셀프 어텐션 계층과 점별 피드포워드 계층을 합친 것과 같으며 본 모델에서는 이 접근법을 택하였다.

부가적인 이점으로, 셀프 어텐션으로 보다 해석이 용이한(interpertable) 모델을 만들 수 있다. 우리는 모델의 어텐션 분포(attention distribution)을 들여다보았고, 부록에 그 예와 해석(discuss)를 실었다. 각각의 어텐션 헤드(attention head)는 서로 다른 작업을 수행하는 법을 명확하게 학습할 뿐 아니라, 문장의 통사적, 의미적 구조와 관련된 행동을 나타내는 것으로 보인다.

5. 학습(Training)

이 장에서는 본 모델의 학습 체제에 대하여 설명한다.

5.1 학습 데이터와 배치화(Traning Data and Batching)

우리는 4백 50만 개의 문장 쌍(sentence pair)으로 이루어진 표준 WMT 2014 영-독 데이터셋(dataset)을 사용하여 모델을 학습시켰다. 문장들은 37000개의 토큰으로 구성된, 공유되는 원본-대상 어휘 사전(source-target vocabulary)를 갖는 바이트 페어 인코딩(byte-pair encoding)[3] 방식으로 인코딩하였다. 영-프 데이터셋의 경우, 3천 6백만 개의 문장과 32000개의 구분 토큰(split token)의 워드 피스 어휘 사전(word-piece vocabulary)을 갖는[31], 현저히 큰 WMT 2014 영-프 데이터셋을 사용하였다. 문장 쌍은 근사된(approximate) 시퀀스 길이로 묶여 배치화된다. 각 학습 배치는 근사적으로 25000개의 원본 토큰(source token)과 25000개의 대상 토큰(target token)을 가지는 문장 쌍들의 집합으로 구성된다.

5.2 하드웨어와 스케줄(Hardware and Schedule)

우리는 8개의 NVIDIA P100 GPU를 보유한 기기에서 학습을 진행하였다. 논문 전반에서 설명된 초매개변수(hyperparameter)를 사용한 기본 모델(base model) 학습 시, 각 학습 단계(training step)는 0.4초 정도의 시간이 걸렸다. 우리는 이 기본 모델을 총 100,000 학습 단계, 시간으로는 12시간 학습시켰다. 보다 규모가 큰 모델들은(표 3의 하단에 표시된 모델들은), 각 학습 단계가 1.0초 정도의 시간을 필요로 했다. 이러한 모델들은 300,000 학습 단계(3.5일) 동안 학습시켰다.

5.3 옵티마이저(Optimizer)

아담 옵티마이저(Adam optimizer)[17]를 $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$의 값으로 사용하였다. 또한 학습률(learning rate)를 학습 기간 동안 변화시켰는데, 사용한 식은 다음과 같다:

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

이것은 학습률을 최초의 $warmup_steps$동안 선형적으로 증가시키고, 이후로는 학습 단계 숫자의 역제곱근의 비율로 감소시킴을 뜻한다. 사용한 $warmup\_steps$ 값은 4000이다.

5.4 정규화(Regularization)

학습 도중 세 가지 종류의 정규화를 적용하였다:

잔류 드랍아웃(Residual Dropout)

우리는 각 하위계층(sub-layer)의 출력이 하위계층의 입력과 더해지고 정규화되기 이전에 드랍아웃[27]을 적용했다. 또한, 인코더와 디코더 적층 구조에서 임베딩이 위치 기반 인코딩과 합쳐진 결과에도 드랍아웃을 적용하였다. 기본 모델(base model)에서, 이 비율 P_{drop} 은 0.1이다.

표 2: 트랜스포머는 영-독, 그리고 영-프 newstest2014 과제에서 이전의 최신(state-of-the-art) 모델보다 적은 학습 비용으로도 더 나은 BLEU 성능을 보였다.

모델	BLEU		학습 비용(FLOPs)	
	영-독	영-프	영-독	영-프
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	29.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
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-Attn + PosUnk 앙상블 [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL 앙상블 [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S 앙상블 [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
트랜스포머 (기본 모델)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
트랜스포머 (대규모)	28.4	**41.0**	$2.3 \cdot 10^{19}$	

라벨 평탄화(Label Smoothing)

학습 과정에서, 우리는 $\epsilon_{ls} = 0.1$의 비율을 갖는 라벨 평탄화를 적용했다[30]. 이것은 모델이 보다 불확실하게 학습하도록 함으로써 퍼플렉시티(perplexity)에 악영향을 주지만, 정확도와 BLEU 성능을 향상시킨다.

6. 결과(Results)

6.1 기계 번역(Machine Translation)

WMT 2014 영-독 번역 과제에서, 대규모 트랜스포머 모델(표 2의 트랜스포머 (대규모) 행)은 새로운 최고(state-of-the-art) 성능인 28.4 BLEU를 기록하며 기존의 (앙상블을 포함한) 최고 성능 모델들을 2.0 BLEU 이상 능가하였다. 모델 설정 값(configuration)은 **표 3**의 아래쪽 행에 나열되어 있고, 학습에는 8개의 P100

GPU를 사용하여 3.5일이 걸렸다. 심지어 트랜스포머 기본 모델조차도 경쟁 모델들이 필요로 하는 학습 비용의 일부만으로도 그들의 성능을 능가하였다.

WMT 2014 영-프 번역 과제에서, 대규모 트랜스포머 모델은 41.0의 BLEU를 기록하여 기존 최고 모델들의 1/4 미만의 학습 비용을 필요로 하고서도 기존의 모든 단일 모델 성능을 능가하였다. 영-프 과제에서 사용된 대규모 트랜스포머 모델은 드랍아웃 비율 P_{drop}을 0.3 대신 0.1로 사용하였다.

기본 모델에서, 우리는 10분 간격으로 기록된 체크포인트(checkpoint) 중 가장 최근의 5개를 평균하여 선택된 단일 모델을 사용하였다. 대규모 모델에서는, 최근의 20개 체크포인트를 평균하였다. 또한 빔 크기(beam size)가 4이고, 길이 페널티(length penalty) $\alpha = 0.6$ [31]인 빔 탐색(beam search)를 적용하였다. 이 초매개변수(hyperparameter)는 개발 세트(development set)에서 진행된 실험의 결과로 결정하였다. 추론(inference) 과정에서의 최대 출력 길이(maximum output length)는 입력 길이 $+50$으로 설정하였으나, 가능한 경우 보다 일찍 종료하도록 하였다[31].

표 2는 본 연구의 결과를 요약하고 본 모델의 번역 성능과 학습 비용을 타 논문들의 모델 구조와 서로 비교한 것이다. 우리는 학습 시간, 사용된 GPU의 수, 그리고 각 GPU의 초당 부동소수점 연산의 추정치를 곱하여 모델을 학습시키는데 사용되는 부동소수점 연산(floating point operation)의 추정 횟수를 계산하였다[2].

6.2 모델 변형(Model Variations)

트랜스포머 모델 각 부분의 중요성을 평가하기 위하여, 우리는 기본 모델을 다양한 방식으로 변형하여 개발 세트(development set)인 newstest2013의 영-독 번역 성능을 측정하였다. 빔 탐색은 전 장에 설명한 바와 같이 적용하였으나, 체크포인트를 평균하는 과정은 거치지 않았다. 이 결과를 **표 3**에 정리하였다.

표 3의 (A) 열에서, 우리는 3.2.2장에서 설명한 바와 같이 연산량은 유지하면서 어텐션 헤드, 그리고 어텐션 키와 값 차원 수를 변경하였다. 단일 헤드 어텐션은 최고 성능의 설정보다 0.9 낮은 BLEU를 보였지만, 헤드 수가 너무 많아도 성능이 저하되었다.

표 3의 (B) 열에서, 우리는 어텐션 키 크기인 d_k를 줄이는 것이 모델에 악영향을 미치는 것을 확인하였다. 이것은 호환성을 결정하는 것이 쉽지 않은 작업이며, 내적(dot product)보다 조금 더 복잡한 호환성 함수가 효과적일 수 있다는 점을 보여 준다. 이후 (C)열과 (D)열에서 우리는 규모가 더 큰 모델들이 성능이 더 좋았으며, 드랍아웃(dropout)이 과적합(over-fitting)을 방지하는 데 매우 효과적임을 확인할 수 있었다. (E) 열에서 사인 곡선형(sinusoidal) 위치 기반 인코딩(positional encoding)을 학습된 위치 기반 임베딩(learned positional embedding)으로 대체하였고[8], 기본 모델(base model)과 거의 동일한 결과를 얻었다.

---

[2] 우리는 K80, K40, M40, 그리고 P100에 대하여 각각 2.8, 3.7, 6.0, 그리고 9.5의 TFLOPs 값을 사용하였다.

표 3: 트랜스포머 구조의 다양한 변형. 비어 있는 값들은 기본 모델의 값과 동일한 것이다. 모든 단위는 영-독 번역 개발 세트인 newstest2013에 따른 것이다. 표기된 퍼플렉시티(perplexity)는 바이트 페어(byte-pair) 인코딩에 따라 단위 워드 피스 당(per-wordpiece)으로 계산된 것이며 단위 단어 당(per-word) 퍼플렉시티와 비교되어서는 안 된다.

	N	d_{model}	d_{ff}	h	d_k	d_{model}	d_{model}	d_{model}	학습 단계	PPL (개발)	BLEU (개발)	매개변수 $\times 10^6$
기본	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)	사인 곡선 이외의 위치 기반 임베딩 적용									4.92	25.7	
대규모	6	1024	4096	16			0.3		300K	4.33	26.4	213

7. 결론(Conclusion)

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprintarXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. CoRR, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. arXiv preprint arXiv:1601.06733, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statisticalmachine translation. CoRR, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint

arXiv:1610.02357, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555, 2014.

[8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017.

[9] Alex Graves. Generating sequences with recurrent neural networks. arXiv preprintarXiv:1308.0850, 2013.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and PatternRecognition, pages 770–778, 2016.

[11] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow inrecurrent nets: the difficulty of learning long-term dependencies, 2001.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation,9(8):1735–1780, 1997.

[13] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploringthe limits of language modeling. arXiv preprint arXiv:1602.02410, 2016.

[14] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In International Conferenceon Learning Representations (ICLR), 2016.

[15] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. arXiv preprint arXiv:1610.10099v2,2017.

[16] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In International Conference on Learning Representations, 2017.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[18] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. arXiv preprintarXiv:1703.10722, 2017.

[19] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, BowenZhou, and Yoshua Bengio. A structured self-attentive sentence embedding. arXiv preprint arXiv:1703.03130, 2017.

[20] Samy Bengio Łukasz Kaiser. Can active memory replace attention? In Advances in Neural Information Processing Systems, (NIPS), 2016.