Attention Is All You Need

Transformer 네트워크 아키텍쳐: 오직 attention mechanisms에만 기반을 둠. RNN, CNN 배제.
-> 더 빠르게 병렬 처리하고, 더 적은 훈련 시간으로도 좋은 성능을 냄

1. Introduction

- 순환 신경망(RNN) 모델: 입력과 출력 시퀀스를 단어의 위치에 따라 순차적으로 계산함(단어 순서가 중요!). 입력의 각 위치를 시간 축에 맞춰 정렬하고, 이전 시점의 은닉 상태 h_{t-1}와 현재 시점의 입력 x_t를 바탕으로 현재 시점의 은닉 상태 h_t를 생성함
  이러한 순차적인 계산 방식 때문에 훈련 과정에서 병렬화가 불가능하며, 특히 긴 시퀀스를 처리할 때 메모리 제약으로 인해 배치(batch) 처리에 어려움이 발생.
- 어텐션 메커니즘: 입력과 출력 시퀀스의 거리와 상관없이 의존성을 모델링 가능(문장이 길어도 앞 뒤 관계 이해 가능). 대부분은 어텐션이 순환 신경망과 함께 사용되지만, 세로운 아키텍쳐인 Transformer는 순환 구조를 완전히 배제하고 어텐션 메커니즘만을 활용해 입력과 출력 간의 전역적인 의존성을 모델링함(시퀀스의 모든 위치 간의 관계 한 번에 파악 가능)
  Transformer는 훨씬 더 높은 병렬 처리 성능을 제공하며, 8개의 P100 GPU에서 단 12시간 훈련만으로 기계 번역 작업에서 최고 성능을 달성가능.

2. Background

- 순차 처리(Sequential Computation) 줄이기: RNN은 앞 단계의 계산 결과가 다음 단계의 입력이 되기 때문에 병렬 처리가 불가능함 -> CNN 기반 모델들(ByteNet, ConvS2S 등)은 병렬 처리가 가능해 RNN보다 빠르게 계산 가능
- CNN 기반 모델의 문제점: 멀리 떨어진 단어들 사이의 관계를 이해하려면 연산이 많아짐
  ConvS2S는 두 단어가 멀리 떨어질수록 연산 수가 선형적으로 증가하고, ByteNet은 연산이 로그 형태로 증가함 -> 멀리 떨어진 위치 간의 의존성을 학습하는 것이 어려워짐
- Transformer의 차별점: Transformer에서는 임의의 두 위치 간 관계를 일정한 연산 수로 계산함(-> 모든 단어를 동시에(병렬로) 처리 가능, 효율적). 즉, 모든 단어 쌍의 관계를 동시에 계산할 수 있기 때문에, 멀리 떨어진 단어 간의 관계도 한 번의 연산으로 파악 가능
  하지만, 어텐션 가중치를 적용한 위치들이 평균화되면서 해상도가 낮아지는(정보가 희석되는) 문제가 발생할 수 있음. 이러한 문제는 Multi-Head Attention을 사용해 다양한 관점에서 정보를 처리하여 이 문제를 해결함(3.2절 참조)
- Self-Attention은 문장 안의 단어들 사이의 관계를 계산해 문장 전체의 의미를 파악하는 방법
  원리) 문장에 있는 모든 단어 쌍의 관계를 동시에 계산 – 중요한 단어 쌍에 높은 가중치를 부여함 - 가중치 정보를 바탕으로 문장의 전체 의미를 담은 표현(embedding; 문장의 전체 의미를 압축한 숫자 벡터)을 만듦

3. Model Architecture

대부분의 경쟁력 있는 neural sequence transduction model은 인코더-디코더 구조를 가짐

- 인코더: 입력 시퀀스(x_1, ..., x_n)를 받아 이를 연속된 표현 벡터 시퀀스(z = (z_1, ..., z_n))로 변환.
- 디코더: 인코더가 생성한 연속된 표현 벡터(z)를 바탕으로 출력 시퀀스(y_1, ..., y_m)를 한 번에 하나씩 생성

모델은 auto-regressive 방식(앞에서 만든 단어를 참고해 다음 단어를 만드는 방식)으로 동작함

Transformer는 이와 같은 인코더-디코더 아키텍처를 따르며, 인코더와 디코더 모두 여러 층으로 쌓여 있음. 각 층은 Self-Attention 층과 point-wise fully connected layers으로 구성됨. 이 구조는 Figure 1의 좌측과 우측에 각각 인코더와 디코더로 표시되어 있음

- Self-Attention: 단어들 사이의 관계를 파악
- point-wise fully connected layers: 각 단어의 의미를 더 정교하게 만듦

3.1 Encoder and Decoder Stacks

인코더(Encoder)

인코더는 6개의 동일한 층으로 구성.

각 층은 두 개의 서브 레이어가 있음

- 1) Multi-Head Self-Attention: 모든 단어들끼리의 관계를 동시에 계산 2) position-wise fully connected feed-forward network: 각 단어의 벡터를 정제하고 조정해 더 좋은 표현으로 만듦

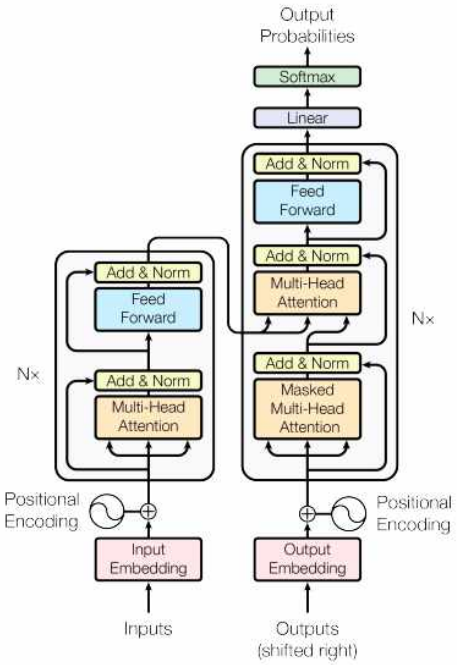


Figure 1: The Transformer - model architecture.

각 서브 레이어의 출력에는 잔차 연결(residual connection)을 적용함 - 이전 입력과 새로운 출력을 더해, 신호 손실 없이 정보가 잘 전달되도록 함. 그 후, layer normalization를 적용해 안정적인 학습을 보장함. 즉, 각 서브 레이어의 출력은 다음과 같이 계산됨:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

* x: 이전 입력. Sublayer(x): 현재 층에서 새롭게 계산된 출력

모든 레이어와 임베딩 층의 출력은 512차원 벡터로 고정 - 같은 차원을 유지하면, 여러 층 사이에서 정보 전달(잔차연결)이 원활(∵ 잔차 연결에서는 이전 입력과 현재 층의 출력을 더하기 때문에, 두 벡터의 차원이 같아야 더할 수 있음)

디코더(Decoder)

디코더도 6개의 동일한 층으로 구성. 인코더 층의 두 서브 레이어에 세 번째 서브 레이어 추가됨.

- 1) Multi-Head Self-Attention: 디코더 내부의 현재까지 생성된 단어들끼리의 관계 계산, 마스킹 적용

  2) Encoder-Decoder Attention: 인코더가 만든 정보(벡터)를 참고해 디코더가 다음 단어를 생성함. 이 과정에서 멀티-헤드 어텐션을 사용해, 인코더 출력의 중요한 부분에 집중함

  * 멀티-헤드 어텐션: 하나의 어텐션 메커니즘을 여러 번 병렬로 수행해 다양한 관점에서 단어들 간의 관계를 파악함

  3) feed-forward network: 각 단어의 벡터를 독립적으로 정제하고 조정해 더 정교한 표현으로
- 마스킹(masking)과 오프셋이 적용됨: 마스킹은 다음 단어를 예측할 때 아직 생성되지 않은 단어를 참고하지 못하게 막고, 앞에서 생성된 단어들만 보고 올바른 다음 단어를 예측할 수 있도록 보장함(오토레그레시브 방식). 오프셋은 출력 임베딩이 한 칸씩 뒤로 밀려 각 위치에서 순서대로 단어가 생성됨 -> 디코더가 정확한 순서로 문장을 생성함
- 인코더와 마찬가지로, 각 서브 레이어에는 잔차 연결과 레이어 정규화가 적용되어 정보 손실 없이 안정적인 학습이 가능함

3.2 Attention (어텐션)

어텐션 함수는 쿼리(query)와 키-값 쌍(key-value pairs)을 출력에 매핑하는 것으로 설명할 수 있음. 여기서 쿼리, 키, 값, 그리고 출력 모두 벡터임.

출력은 값들의 가중합(weighted sum)으로 계산됨 - 쿼리와 키 사이의 dot product을 통해 유사도(compatibility)을 계산해, 각 값에 가중치(weight)를 할당하고, 이 가중치에 따라 값들을 가중합하여 최종 출력을 만듦. (compatibility를 통해 중요한 정보에 더 집중)

3.2.1 Scaled Dot-Product Attention

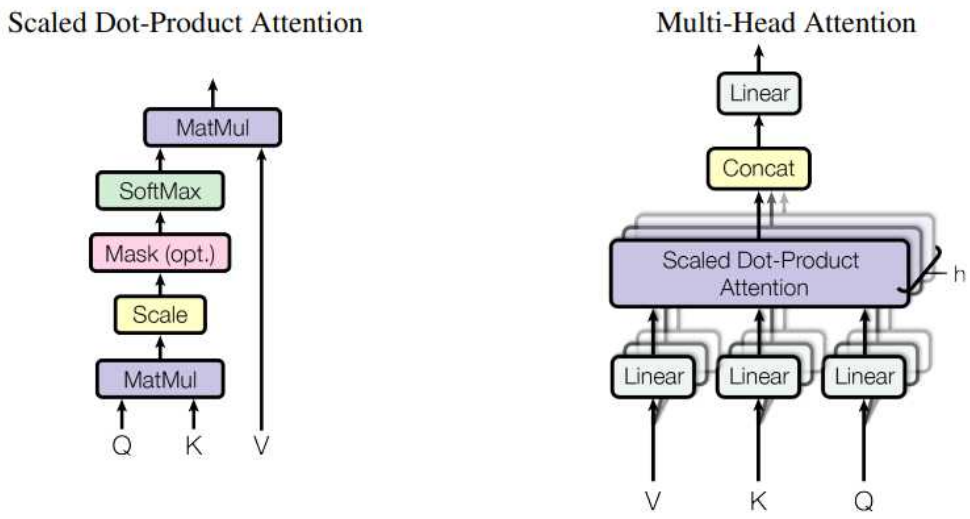
Scaled Dot-Product Attention의 수식

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

- Q: 쿼리 행렬 (차원: d_k); 현재 처리 중인 단어에서 다른 단어들과의 관계를 파악하기 위해 사용되는 벡터 ex) 문장 내 단어 apple이 문맥상 다른 단어들과 어떻게 연결되는지를 파악하고자 함
- K: 키 행렬 (차원: d_k); 각 단어에 부여된 벡터로, 다른 단어가 이 단어와 얼마나 관련이 있는지를 판단할 때 사용됨 ex) 문장 속 다른 단어들("fruit", "is", "red" 등)의 키가 각각 주어짐
- V: 값 행렬 (차원: d_v); 해당 단어가 가진 실제 정보를 담은 벡터. 어텐션 메커니즘은 이 값을 가중합(weighted sum)해 최종 출력을 만듦 ex) "apple"과 관련된 의미를 가지는 벡터
- 쿼리와 키 간의 dot product을 계산한 뒤, 각각의 값을 $\sqrt{d_k}$로 나눔. 이후, 소프트맥스 함수를 적용해 값들에 대한 가중치를 구함. 구해진 가충치를 값(V)에 곱해 최종 출력(V 행렬의 가중합)을 만듦

가장 널리 사용되는 두 가지 어텐션 함수 dot-product attention과 additive attention의 비교

- dot-product attention: 최적화된 행렬 곱셈으로 구현 가능 - 더 빠르고 메모리 효율적임.
- additive attention: hidden layer을 가진 피드포워드 네트워크를 사용해 호환성 계산. d_k가 비슷할 때는 두 메커니즘이 비슷한 성능을 보이지만 d_k가 클 때는 additive attention의 성능이 더 좋을 수 있음(∵ d_k가 클 때 dot-product의 값이 매우 커져 소프트맥스 함수의 기울기가 소실되지만, 이를 $\sqrt{d_k}$로 나눠 스케일링하기 때문) .



3.2.2 Multi-Head Attention

기존에는 하나의 어텐션 함수만으로 쿼리, 키, 값을 사용했음 - 하나의 관점에서만 단어들 간의 관계를 계산하게 되어 여러 가지 의미나 문맥을 다양한 관점에서 동시에 파악하는 게 어려움

-> 여러 개의 어텐션을 병렬로 사용

- 이 병렬 어텐션을 각각 "헤드"라고 부르고, 여러 개의 헤드가 동시에 작동하도록 해 다양한 정보를 동시에 반영 - 각 헤드가 다른 부분에 집중해 더 풍부한 관계 학습 가능
- 각 헤드는 쿼리(Q), 키(K), 값(V)을 서로 다른 차원으로 변환해서 계산함.
- 예) 하나의 헤드는 문장 초반의 단어들에 집중하고, 다른 헤드는 문장 끝에 집중할 수 있음

멀티-헤드 어텐션의 동작 원리

쿼리(Q), 키(K), 값(V)을 여러 번 선형 변환(linear projection)함 - 각 헤드는 서로 다른 방식(관점)으로 쿼리, 키, 값을 분석함 (각 쿼리와 키는 d_k 차원, 값은 d_v 차원으로 변환됨)

이렇게 변환된 쿼리, 키, 값을 사용해 병렬로 어텐션 함수를 수행하고, 각 헤드가 계산한 결과를 연결(concatenate)함. 이렇게 연결된 결과를 다시 한 번 투영해 최종 출력을 만듦.

$$MultiHead(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$

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

매개변수 행렬

- $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$: 쿼리 투영 행렬
- $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$: 키 투영 행렬
- $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$: 값 투영 행렬
- $W^O \in \mathbb{R}^{hd_v \times d_{model}}$: 최종 출력 투영 행렬

멀티-헤드 어텐션의 장점

여러 관점에서 단어들 간의 관계를 동시에 계산함

단일 어텐션에서는 이런 다양한 관점이 평균화돼 표현이 단순해질 수 있지만, 멀티-헤드 어텐션은 그 문제를 해결함

계산 비용

여러 개의 헤드를 사용해도 각 헤드의 차원을 줄이기 때문에, 멀티-헤드 어텐션의 총 계산 비용은 단일-헤드 어텐션과 유사함

이 연구에서는 8개의 헤드를 사용했고, 각 헤드의 차원은 d_k=d_v=64로 설정됨

3.2.3 Applications of Attention in our Model

Transformer는 세 가지 방식으로 멀티-헤드 어텐션을 사용합니다:

1. 인코더-디코더 어텐션(encoder-decoder attention):

쿼리: 이전 디코더 층에서 가져옴. 키와 값: 인코더의 출력에서 가져옴.

기능: 디코더의 각 위치가 입력 시퀀스 전체를 참조할 수 있도록 함.

이는 기존의 시퀀스-투-시퀀스(seq2seq) 모델의 어텐션 메커니즘과 동일한 방식임

2. 인코더의 셀프 어텐션(self-attention):

쿼리, 키, 값이 모두 이전 인코더 층의 출력에서 가져옴

기능: 인코더의 모든 위치가 이전 층의 모든 위치를 참조해 문맥 정보를 충분히 학습할 수 있음
3. 디코더의 셀프 어텐션(self-attention):
쿼리, 키, 값이 모두 디코더의 이전 층에서 가져옴
마스킹 적용: 디코더의 각 위치가 현재와 이전 위치까지만 참조할 수 있도록, 미래 위치에 해당하는 값들을 -∞로 설정해 소프트맥스 계산에서 배제함 - 오토레그레시브 방식으로 단어를 순서대로 생성할 수 있음.

3.3 Position-wise Feed-Forward Networks
어텐션 서브 레이어 외에, 인코더와 디코더의 각 층에는 완전 연결 feed-forward network가 포함됨
1. 구성 요소: 두 개의 선형 변환과 그 사이의 ReLU 활성화 함수로 구성

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

2 동작 원리:
각 위치에 독립적으로 동일한 네트워크를 적용하지만, 각 층마다 서로 다른 파라미터를 사용함.
이 방식은 커널 크기 1인 두 개의 합성곱(convolution)과 비슷함
입력과 출력의 차원 d_{model}=512, 내부 층의 차원 d_{ff}=2048

3.4 Embeddings and Softmax
다른 시퀀스 변환 모델들과 유사하게, 학습된 임베딩을 사용해 입력 토큰과 출력 토큰을 d_{model} -차원 벡터로 변환함.
디코더가 생성한 출력을 선형 변환을 거쳐 소프트맥스(softmax) 함수로 처리해 다음에 나올 단어의 확률로 변환함
Transformer는 입력 임베딩과 출력 임베딩, 그리고 소프트맥스 직전의 선형 변환에 같은 가중치 행렬을 사용함
임베딩 층에서는 가중치 행렬을 $\sqrt{d_{\mathrm{model}}}$로 곱해 스케일을 맞춤(스케일링)

3.5 Positional Encoding
- Transformer에는 순환(RNN)이나 합성곱(CNN)처럼 순서 정보를 자연스럽게 학습할 수 있는 구조가 없기 때문에, 시퀀스의 순서 정보를 활용하려면 각 토큰의 상대적 또는 절대적 위치에 대한 정보를 추가해야 함.
- 이를 위해, 인코더와 디코더 스택의 입력 임베딩에 위치 인코딩(positional encoding)을 더함
- 위치 인코딩과 임베딩의 차원은 동일하게 d_{model}이므로, 두 값을 더할 수 있음
- 위치 인코딩은 학습 가능한 방식과 고정된 방식 중에서 선택할 수 있음

사인(sine)과 코사인(cosine) 함수 기반의 위치 인코딩(; fixed positional encoding)
각 차원마다 서로 다른 주파수의 사인(sine)과 코사인(cosine) 함수를 사용해 위치를 인코딩함:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

pos: 토큰의 위치
i: 차원의 인덱스

- 각 차원의 위치 인코딩은 사인/코사인 함수를 사용해 표현됨.
- 이때 파장은 2π에서 $10000 \cdot 2\pi$까지 기하급수적 증가를 따름
- 사인/코사인 인코딩은 고정된 오프셋이 있는 경우에도 상대적인 위치를 쉽게 학습할 수 있게 함
  - 예를 들어, "I have a pen"에서 "pen"은 항상 "a" 다음에 나옴. 이때, "a"와 "pen"의 상대적 거리(offset)는 항상 +1. 사인/코사인 위치 인코딩에서는 "a"와 "pen"의 위치 인코딩 간의 관계가 선형으로 표현됨
- 사인/코사인 함수는 주기적인 특성 덕분에, 훈련되지 않은 더 긴 시퀀스에도 일관된 상대적 위치를 반영할 수 있음

4 Why Self-Attention

1. 병렬화: Self-Attention은 입력 시퀀스의 모든 단어를 동시에 처리하므로 병렬 처리가 가능함
2. 장기 의존성(Long-range dependency) 해결: 문장 내 먼 단어 간 관계도 쉽게 학습함
3. 효율적인 계산: RNN에 비해 계산이 단순하며 더 빠름. 모델의 각 층은 셀프 어텐션과 피드포워드 네트워크로 구성되며, 간단한 연산과 행렬 곱셈만으로 수행됨

5 Training

Transformer 모델의 훈련에서는 효율적이고 안정적인 학습을 위해 여러 가지 기법을 사용함

5.1 학습률 스케줄링

Transformer는 일정한 학습률이 아닌 학습률 스케줄링을 사용함. 초반에는 학습률이 점진적으로 증가하다가, 일정 단계 이후 감소. 이 방식을 통해 학습 초기 불안정한 가중치를 빠르게 조정하고, 후반부에는 정밀한 조정을 도와줌.

5.2 옵티마이저

Adam 옵티마이저를 사용해 가중치 업데이트를 수행. 학습률 스케줄링과 결합하여 빠르고 안정적인 학습을 가능하게 함

5.3 드롭아웃(Dropout)

각 층에 드롭아웃(dropout)을 적용해 오버피팅을 방지함

드롭아웃 비율은 0.1로 설정됨

5.4 배치 크기와 GPU 사용

Transformer는 대규모 데이터셋을 8개의 GPU에서 병렬 처리해 빠르게 학습됨-> 학습 시간 단축

6 Results

Transformer는 WMT 2014 영어-독일어 번역에서 BLEU 점수 28.4를 기록하며, 기존 모델보다 우수한 성능. 훈련 시간과 사용 자원도 크게 단축되었음(3.5일 동안 8개의 GPU 사용).

7 Conclusion

Transformer는 RNN과 CNN 없이도 최고의 성능을 내는 모델로 자리 잡았다.

특히, 멀티-헤드 어텐션과 병렬 처리 가능성 덕분에, 기존 시퀀스 모델에 비해 훨씬 빠르고 효율적으로 학습됨

다양한 NLP 작업에 응용될 가능성이 높으며, 텍스트 외에도 이미지, 오디오, 비디오 처리 등으로 확장 가능함