Attention is All You Need Summary

지윤 파트

1. Introduction

- RNN, 특히 LSTM과 GRU가 sequence modeling과 transduction 문제에서 SOTA 로 확립됨
  - 예: language modeling, machine translation
- **RNN의 주요 한계:**
  - 계산이 input과 output sequences의 symbol positions를 따라 순차적으로 이루어짐
  - 이로 인해 parallelization이 어려워 긴 sequences에서 효율성 저하
  - 메모리 제약으로 인해 batching across examples도 제한됨
- **최근 연구들:**
  - Factorization tricks과 conditional computation으로 computational efficiency 개선
  - 그러나 sequential computation의 근본적 제약은 여전히 존재
- **Attention mechanisms:**
  - Sequence modeling과 transduction 모델의 중요한 부분이 됨
  - Input/output sequences 내 거리에 관계없이 dependencies 모델링 가능
  - 대부분의 경우 recurrent network와 함께 사용됨

- **Transformer 모델:**
  - Recurrence를 사용하지 않고 전적으로 attention mechanism에 의존
  - Input과 output 간 global dependencies를 모델링
  - 높은 수준의 parallelization 가능
  - 8개의 P100 GPUs로 12시간 훈련 후 translation에서 new SOTA 달성

1줄 요약: **Transformer 모델은 RNN의 한계를 극복하고 병렬 처리를 가능하게 하는 새로운 attention 기반 아키텍처임.**

2. Background

- **sequential computation 감소를 위한 다른 모델들:**
  - Extended Neural GPU, ByteNet, ConvS2S
  - 공통점: convolutional neural networks를 기본 구성 요소로 사용
- **이전 모델들의 한계:**
  - 입출력 위치 간 관계를 계산하는 데 필요한 연산 수가 거리에 따라 증가
    - ConvS2S: 선형적 증가
    - ByteNet: 로그적 증가
  - 먼 위치 간 dependencies 학습이 어려움
- **Transformer의 장점:**
  - 위치 간 관계 계산에 필요한 연산 수가 상수로 고정
  - Multi-Head Attention으로 attention-weighted positions의 평균화(averaging)로 인한 해상도 감소 문제 해결
- **Self-attention (intra-attention):**
  - 단일 sequence 내 다른 위치들을 연관시켜 representation 계산
  - 다양한 태스크에서 성공적으로 사용됨
    - 예: reading comprehension, abstractive summarization, textual entailment, task-independent sentence representations 학습
- **End-to-end memory networks:**
  - Sequence-aligned recurrence 대신 recurrent attention mechanism 사용
  - Simple-language question answering과 language modeling 태스크에서 좋은 성능 보임

- **Transformer의 독창성**:
  - Sequence-aligned RNNs나 convolution 없이 전적으로 self-attention에 의존하는 최초의 transduction 모델
  - Input과 output의 representations를 계산하는 데 self-attention만 사용

1줄 요약: Sequential computation 감소를 위한 다양한 접근법들이 있었으나 self-attention으로 해결

3. Model Architecture : Encoder-Decoder 구조에 self-attention

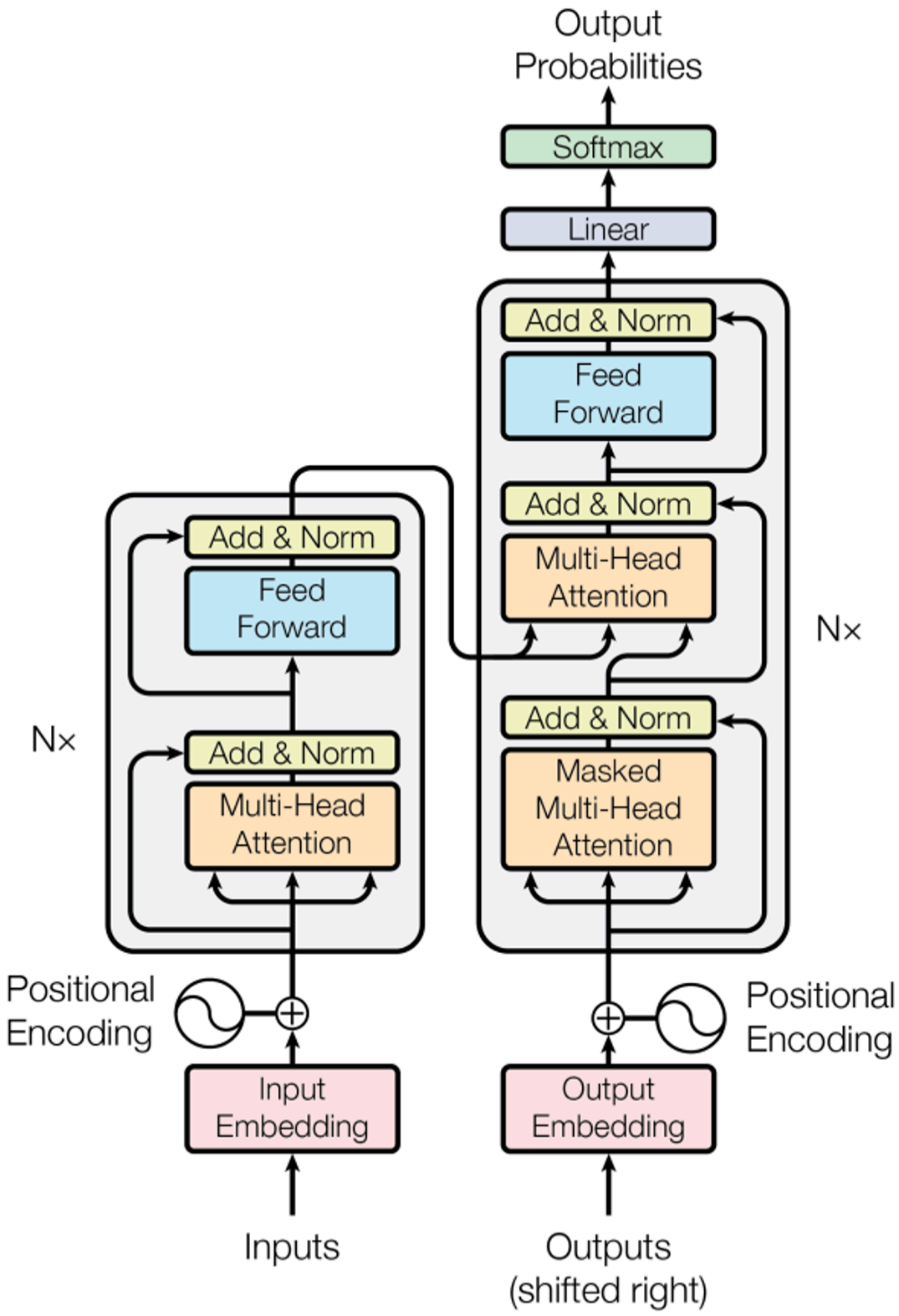


Figure 1: The Transformer - model architecture.

- **기본 구조:**
  - 인코더-디코더 구조 채택 (대부분의 competitive한 neural sequence transduction 모델과 유사)
  - 인코더: symbol 표현((x1, ..., xn)의 입력 시퀀스를 연속적인 표현(z1, ..., zn)으로 매핑
  - 디코더: 인코더의 출력을 바탕으로 한 번에 하나의 element의 symbol 출력 시퀀스 생성
- **Transformer의 특징:**
  - 인코더와 디코더 모두 각각 stacked self-attention과 point-wise, fully connected layers 사용
  - Auto-regressive 특성: 이전에 생성된 심볼들을 다음 심볼 생성 시 추가 입력으로 사용

▶ **아키텍처 구성 (Figure 1):**

3.1 Encoder and Decoder Stacks

- Encoder:
  • 구성: 6개의 동일한 레이어로 구성 (N = 6)
  • 각 레이어의 구조:
    1. Multi-head self-attention 메커니즘
    2. Position-wise fully connected feed-forward network

  • 특징:
    - 각 sub-layer 주변에 residual connection 사용
    - Layer normalization 적용
    - 출력 계산: LayerNorm(x + Sublayer(x))

  • 차원: 모든 sub-layer와 임베딩 레이어의 출력 차원은 d_{model} = 512

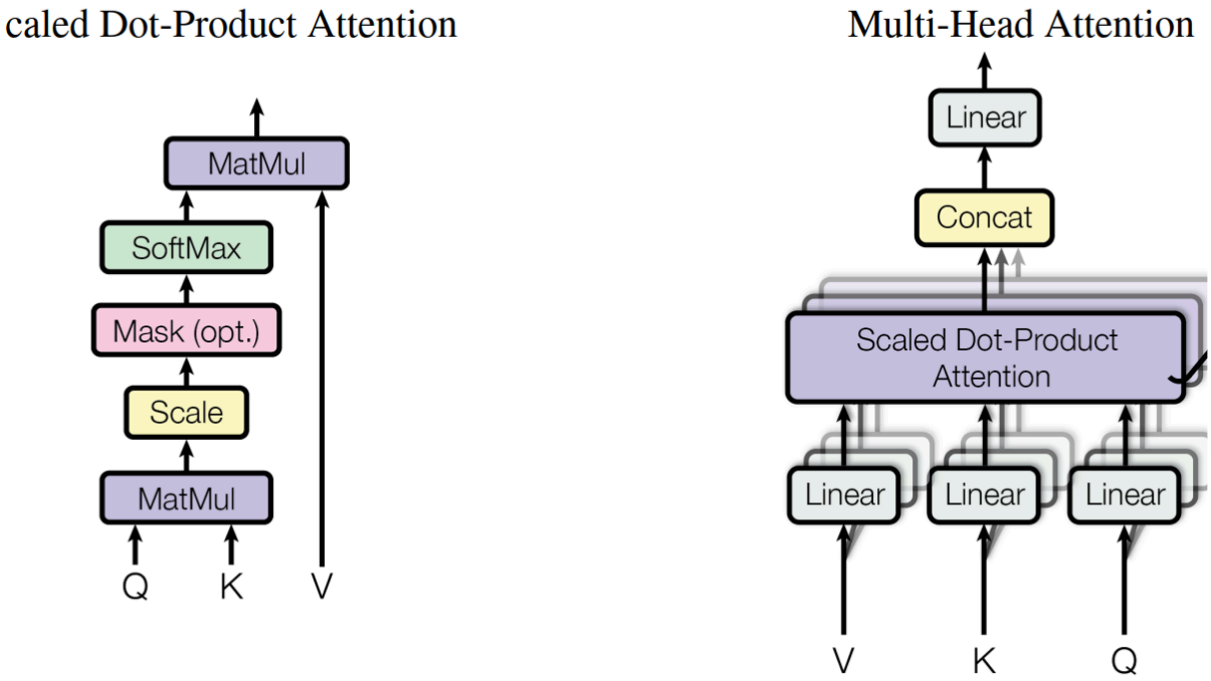
- Decoder:
  • 구성: Encoder와 마찬가지로 6개의 동일한 레이어 (N = 6)
  • 각 레이어의 구조:
    1. Masked multi-head self-attention
    2. Multi-head attention (인코더 출력에 대해)
    3. Position-wise fully connected feed-forward network
  • 특징:
    - 인코더와 같이 residual connection과 layer normalization 사용
    - Self-attention sub-layer에 마스킹 적용
      - 목적: 현재 위치가 이후 위치의 정보를 참조하지 못하게 함
    - 출력 임베딩을 한 위치씩 오프셋
      - 효과: 위치 i의 예측이 i보다 작은 위치의 알려진 출력에만 의존하도록 보장

Encoder와 Decoder의 주요 차이점:

1. Decoder는 Encoder 출력에 대한 추가적인 multi-head attention layer를 포함하고 있음.
2. Decoder의 self-attention에는 마스킹 적용 (미래 정보 참조 방지)

- 즉, Transformer가 병렬 처리를 효과적으로 수행하면서도 시퀀스의 순서 정보를 유지하고, 장거리 의존성을 학습할 수 있게 함.

3.2. Attention (Scaled Dot-Product Attention과 Multi-Head Attention)



1. Scaled Dot-Product Attention (왼쪽 그림):
   - 입력: Query (Q), Key (K), Value (V) 벡터
   - 과정:
     a. Q와 K의 내적(dot product) 계산: MatMul(Q, K)
     b. 스케일링: Scaling facotr, (일반적으로 $\frac{1}{\sqrt{d_k}}$ d_k는 key의 차원)
     c. 마스킹 (선택적): Mask (optional)
     d. Softmax 적용
     e. V와의 행렬 곱: MatMul(Softmax 결과, V)
        - 수식: $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$

2. Multi-Head Attention (오른쪽 그림):
   • 여러 개의 attention 레이어를 병렬로 실행
   • 과정:
   a. Q, K, V를 각각 linear 변환
   b. 각 변환된 Q, K, V에 대해 Scaled Dot-Product Attention 수행
   c. 여러 attention의 결과를 연결(Concat)
   d. 최종 linear 변환 적용
   • 특징:
   - h개의 병렬 attention 레이어 (그림에서는 h개의 화살표로 표시)
   - 각 attention이 다른 표현 부분공간에 집중할 수 있게 함
3. Attention의 일반적 개념:
   • Query와 Key-Value 쌍들을 입력으로 받아 출력 생성
   • 출력: Value들의 가중합
   • 가중치: Query와 해당 Key의 호환성(compatibility)에 따라 결정

- 결론: 이 구조를 통해 Transformer는 입력 시퀀스의 다양한 부분에 동시에 주목할 수 있으며, 이는 모델의 성능을 크게 향상시킴 → 즉, Multi-Head Attention은 여러 관점에서 정보를 추출할 수 있게 하여 모델의 표현력을 높임.

3.2.1 Scaled Dot-Product Attention

Scaled Dot-Product Attention:

- 입력:
  - Queries (Q), Keys (K): 차원 d_k
  - Values (V): 차원 d_v
  - 계산 과정:
    1. Q와 K의 내적 계산
    2. 결과를 $\sqrt{d_k}$로 나누어 스케일링
    3. Softmax 함수 적용
    4. 결과를 V와 곱함
- 실제 구현:
  • Q, K, V를 행렬로 패킹하여 동시에 여러 쿼리에 대해 계산
  • 최적화된 행렬 곱셈 코드로 구현 가능 → 빠르고 공간 효율적

다른 Attention mechanism과의 비교:

1. Additive Attention:
    - 호환성 함수로 단일 은닉층을 가진 FeedForward 네트워크 사용
    - 이론적 복잡도는 비슷하지만 실제로는 더 느리고 공간 비효율적
2. Dot-Product Attention:
    - Scaled Dot-Product Attention과 거의 동일
    - 차이점: scaling factor $\frac{1}{\sqrt{d_k}}$ 없음

성능 차이:
• d_k가 작을 때: 두 메커니즘의 성능 유사
• d_k가 클 때: 스케일링 없는 dot-product attention의 성능 저하

- 원인: 큰 d_k값에서 내적 결과의 크기가 커져 softmax 함수가 매우 작은 기울기를 가지는 영역으로 pushed 됨
- 해결책: $\frac{1}{\sqrt{d_k}}$로 스케일링하여 이 효과를 상쇄

Scaled Dot-Product Attention의 장점:

1. 구현의 간단함
2. 계산 효율성
3. 큰 차원에서도 안정적인 성능

3.2.2 Multi-Head Attention

Multi-Head Attention 구조:
• 기본 개념: single attention 함수 대신 여러 개의 병렬 attention 함수 사용

(Single Attention 보다 각 query, key, value에 대해 서로 다르게 학습해 linear projection 하는 것이 더 나음. 이유: 서로 다른 위치에서 서로 다르게 나타나는 representation 정보를 반영 가능하기 때문)

- Multi-Head Attention의 핵심 아이디어: 모델이 입력의 다양한 측면에 동시에 집중할 수 있게 하는 것 → 모델의 표현력을 향상 (특히 복잡한 언어 이해 Task에서 효과적)
- 각 head가 서로 다른 특성에 집중함으로써, 모델은 입력의 다양한 linguistic 및 semantic 특성을 포착할 수 있게 됨

• 과정:

1. Q, K, V를 h번 서로 다른 학습된 선형 투영으로 변환 (*project된 query, key, value에 대해 나눠 주는 개념: 여러 개의 가중치로 선형 결합 계산을 해 쪼개지는 것)
   - Q → d_k 차원
   - K → d_k 차원
   - V → d_v 차원
2. 각 투영된 버전에 대해 병렬로 Scaled Dot-Product Attention 수행
3. 결과 (d_v 차원) 연결 (concatenate)
4. 최종 선형 투영 적용 (project(가중치와 선형 결합)를 통해 최종 출력 값 산출)

파라미터 행렬:
$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$
$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$
$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$
$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

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

이번 연구의 구현 세부 사항:
• 병렬 attention 레이어 (heads) 수: h = 8
• 각 head의 차원: $d_k = d_v = \frac{d_{\text{model}}}{h} = 64$
• 총 계산 비용: 전체 차원의 single head attention과 유사

→ 각 head의 차원이 감소해 multi로 계산됨에도 불구하고 single attention과 계산 비용은 비슷

장점:

1. 다양한 표현 부분공간의 정보를 동시에 처리
2. 서로 다른 위치의 정보를 결합
3. 단일 head의 평균화 효과 방지 (multi-head attention에서는 모델이 서로 다른 위치에서 서로 다른 표현 하위 공간의 정보를 공동으로 attention할 수 있게 해주는데, single head에서는 averaging이 방해)

1줄요약: Multi-head attention은 다양한 표현 공간에서의 정보 추출을 single head와 비슷한 계산 비용으로 효율적으로 하게 했다.

준희 파트

3.2.3 Applications of Attention in our Model

Transformer에서는 multi-head attention을 세 방법으로 사용

- "Encoder-Decoder Attention"
  - Decoder의 결과 query 와 Encoder의 결과 key, value를 매핑
- "Self-attention layers of the Encoder"
- "Self-attention layers of the Decoder"
  - Masked된 layer 존재

3.3 Position-wise Feed-Forward Networks

Encoder, Decoder에 포함된 모든 layer들에는 Feed-Forward Network을 적용

- Linear Transformation - ReLU - Linear Transformation으로 구성
- 각 단계에서는 모두 다른 parameter를 적용

3.4 Embeddings and Softmax

Encoder와 Decoder에 문장 입력 과정에서 Embedding 진행

- Encoder, Decoder에서 같은 weight matrix 사용

Decoder에서 output 문장을 다음 단어 확률(예측 확률)로 convert하는 과정에서 Liner Transformation & Softmax 활용

- Linear Transformation에서도 같은 weight matrix 사용

3.5 Positional Encoding

문장을 행렬로 Embedding하여 입력하기 때문에 맥락과 순서가 리셋되는 문제 발생

→ Embedding 결과에 Positional Encoding을 더하여 문장 순서 명시

Positional Encoding은 sine, cosine 함수를 이용해 산출

장점 1) 쉬우며 모든 input을 표현할 수 있음

장점 2) training 과정에서 만나지 못한 문장 길이에도 대응 가능함!

4 Why Self-Attention

Recurrent and Convolutional Layer에 비해 Self-Attention Layer는 네 가지 기준에서 장점을 가짐

1. Total Complextity