Attention is all you need - Summary

강서연

1. Introduction

- 기존 sequence modeling
  - RNN, LSTM, GRU 기반 모델링의 확립 → for language modeling, machine translation
  - 기존 매커니즘의 주요 한계
    - 기존 hidden state과 position을 바탕으로 hidden state sequence 계산
    - sequence 길이가 길어지면서 장기기억 손실 문제 발생
    - 계산 효율성 증대를 위해 여러 방식을 시도했지만 한계 존재
  - Attention mechanism의 통합: sequence 내 거리와 관계없이 모델링 가능
- **Transformer의 제안**
  - Recurrence 대신 attention mechanism에만 의존 → input output 간 global dependencies 설명
  - parallelization 증대, 더 나은 translation quality

2. Background

- 목표: sequential computation 감소
  - Extended Neural GPU, ByteNet, ConvS2S - CNN 이용해서 input과 output 병렬로 처리
  - 한계: 랜덤한 input output position 간 시그널을 연관시키기 위해 거리에 따른 연산량의 증가

    → ConvS2S - 선형 증가 / ByteNet - 로그적 증가

    → 먼 위치 간의 dependencies 학습의 어려움
  - Transformer를 통한 한계 개선

    → Multi-Head Attention을 통한 attention-weigthed posirion의 평균화

    → dependencies 학습을 위한 연산량 = constant
- **Self-attention (=intra-attention)**
  - sequence의 representation 계산을 위해 단일 sequence 내 다른 포지션들을 관련시킴
  - reading comprehencion, abstrative summarization, textual entailment, task-independent sentence representation 등에 활용
- End-to-end memory networks
  - sequence-aligened recurrence 대신 recurrent attention mechanism에 기반

- simple-languange question answering, language modeling task에 강함

- Transformer의 제안
  - self-attention에만 의존하는 첫 transduction model

    → input output의 representation을 sequence-aligned RNN이나 convolution없이 계산

3. Model Architecture

💡 전체적인 구조 - Encoder-decoder structure

- Encoder: input sequence → sequence of continuous representations로 변환
- Decoder: time별 output sequence 생성
- 자기 회귀 - 다음 생성 시 이전 생성 심볼 추가 입력
- Encoder & Decoder - stacked self-attention + point-wise FC layers 사용

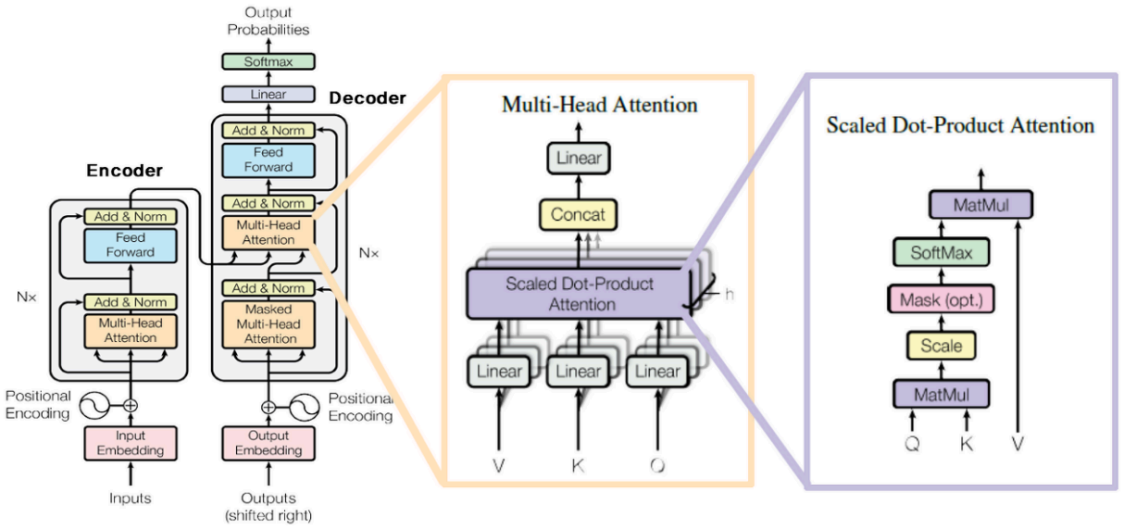

Figure 1: The Transformer - model architecture.

3.1 Encoder and Decoder Stacks

- **Encoder**
  - N=6 의 레이어 개수
  - 각 레이어 당 2개 sub-layer
    - Multi-head self-attention mechanism
    - Position-wise feed-forward network

    → Residual connection + Layer normalization 진행

    → output의 출력 계산: $LayerNorm(x + Sublayer(x))$
  - output dimension d=512

- **Decoder**
  - 동일하게 N=6 의 레이어 개수
  - 각 레이어 당 3개 sub-layer
    - **Masked** multi-head self-attention mechanism
    - Multi-head attention (encoder의 출력에 대해)
    - Position-wise feed-forward network

    → Residual connection + Layer normalization 진행

    → **Masking 진행 (encoder와의 차이점)**

    - Seld-attention sub-layer에 마스킹

      : decoder에서 이후 포지션 정보를 활용하지 못하도록 함

    - output embedding을 한 포지션씩 offset

      : i position의 예측이 i보다 작은 포지션의 출력에만 의존하도록 함

남동연

3.2 Attention

attention 함수는 간단하게 query와 key-value 쌍을 출력으로 내보는 것이라고 할 수 있음. 이때 출력은 query와 key 간의 유사도를 계산하는 `compatibility function`을 이용한 가중치를 이용해 가중합으로 나타남

3.2.1 Scaled Dot-Product Attention

이 논문에서 사용하는 attention을 `Scaled Dot-Product Attention` 이라고 부름

- input:
  - query: d_k 차원으로 모든 query를 하나의 행렬 Q로 표현 + 유사도 계산에 사용
  - keys: d_k 차원으로 모든 key를 하나의 행렬 K로 표현 + 유사도 계산에 사용
  - values: d_v차원으로 모든 value를 하나의 행렬 Q로 표현 + 실제 정보 생성에 사용
- Attention 식: $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- 자수 사용되는 Attention 종류
  - **Additive attention**: `compatibilty function`(qeury, key간의 유사도 평가)을 계산하며 d_{k}로 scaling을 하지 않았을 때 더 나은 성능을 보임
  - **Dot-Product attention**: 위에서 언급한 attention 식과 동일하지만 scaling factor d_{k}를 한 것이 특징.
    - 👍내적 연산을 하기 때문에 더 빠르고 효율적으로 처리 가능
    - 👎d_k차원 증가(고차원 연산일때) ➡️ q,k는 평균이 0이고 분산이 1인 random variable이라고 했을 때 둘의 내적값은 평균이 0 분산이 d_k가 됨➡️내적 값 간의 큰 차이 ➡️softmax 함수의 결과, 하나의 값으로 치우침 ➡️ **매우 작은 기울기 가짐** ➡️ 가중치 업데이트에 문제
    - ⭐ **문제 해결을 위해 $\sqrt{d_k}$로 scaling**

3.2.2 Multihead Attention

하나의 단일한 single attention을 사용하는 것보다 여러 개의 attention을 사용하는 Multihead Attenntion이 더 효과가 좋음

- 방법: 각 버전의 query, key, value들을 병렬적으로 두고 계산한 뒤 concatenate 되어 다시 한번 project되어 사용 ➡️ 이를 통해 서로 다른 representation subspace에 대해 학습하여 일반적인 학습이 가능
- 식: $MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$
- 실제 사용: h=8인 병렬 layer을 사용했고 (8개의 multihead 존재), 각 version 마다 $d_k = d_v = \frac{d_{model}}{h} = 64$의 차원을 동일하게 가짐
    - ⭐ 이때 h의 개수 만큼 각 version의 attention의 차원을 줄임으로써 전체적인 연산비용은 비슷하게 가져감

3.2.3 Applications of Attention in our Model

다음과 같은 방법으로 multi-head attention을 사용한다

1. encoder-decoder attention에서 query는 이전의 decoder layer에서, key와 value는 encoder의 출력을 가져옴 ➡️ decoder가 입력 시퀀스의 모든 위치를 이용(attend, 주의를 기울임) = seq-to-seq와 유사
2. encoder가 self-attention layer을 가짐 ➡️ encoder가 이전 layer의 encoder의 모든 위치를 이용(attend)
3. decoder 역시 self-attention layer을 가짐 ➡️ ⭐미래의 정보를 decode 하는데 사용하면 안됨 ➡️ masking을 통해 자기회귀 특징을 지키면서, 이전 정보의 모든 위치를 이용(attend)

김재훈

3.3. Position-wise Feed Forward Network

- Attention layer에 더해 인코더와 디코더는 FFN을 가지는데 각각의 단어를 개별적, 독립적으로 움직인다.
- $FFN(x) = max(0, xW1 + b1)W2 + b2$

  → F1과 F2, F3으로 구성되어있는데 F2는 ReLU 함수, F1과 F3은 가중치와 편향을 가진 선형함수.

3.4. Embeddings and Softmax

- Linear transformation가 softmax는 decoder의 output(벡터로 표현)을 다음 토큰의 확률로 변환해주는 역할을 한다.

3.5. Positional Encoding

- Transformer에는 recurrence나 convolution이 쓰이지 않아 단어의 순서를 고려하기 어렵다.

  ⇒ 즉, 순서가 조금만 변하더라도 **의미가 아예 달라진다**
- 해결 방법: **Positional encoding**
    - 순서를 고려하기 위해 각 토큰들의 상대적인 위치 혹은 절대적인 위치를 정보로 사용한다.

- 임베딩과 같은 차원인 d_model차원을 가진다. 그래야 서로 합이 가능하기 때문.
- $PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$
- $PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$

  : Pos - 현재 토큰의 위치 / i - 인코딩 벡터의 차원
- 왜 sin, cos인가?

  → 삼각함수는 주기를 가지고 있다. 따라서 초기값 하나를 알면 나머지 다른 값의 상대적 위치는 주기성을 통해 쉽게 알 수 있다. 따라서 이상한 값이 들어오더라도 다 주기성을 통해 알 수 있기 때문이다.

4. Why Self Attention?

- 보통 특정 시퀀스를 다른 시퀀스로 바꿀 때 Convolution 혹은 Recurrent를 사용.
- 그 중 왜 Self Attention이었는지 세 가지 기준에서 비교할 것임.
  1. 얼마나 각 레이어에 대한 계산량이 적다는 것(계산 시간 단축 가능)
  2. 얼마나 병렬계산을 통해 처리 가능한 계산량이 많아진다는 것
  3. 얼마나 장기기억이 가능한가?(얼마나 path length가 짧은가?)

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- Recurrent
  - Sequential Operations 관점에서 볼 때 Recurrent의 성능이 가장 떨어지는 것을 알 수 있다.

    : 나머지는 데이터 크기나 그런 것에 상관없이 **처리 시간이 일정**한 반면 Recurrent는 데이터가 커질수록 처리 시간이 늘어난다.
  - Complexity per Layer의 관점에서 볼 때 Self Attention은 n이 d보다 작을 때 Recurrent보다 더 빠르며 convolution보다도 더 간단하다. 이렇게 n이 d보다 작은 경우는 최신 기계 번역 모델들이 자주 사용하는 문장 표현에서 자주 나타난다.

  ⇒ 따라서 기존의 모델들보다 Self Attention이 기존의 방법들보다 더 좋은 성능을 보인다.
- Convolution
  - Convolution은 커널을 이용해 데이터를 잘게 분해하여 데이터를 처리한다. 커널의 크기가 작을수록 데이터를 더 잘게 분해하므로 더 많은 레이어를 쌓아야하는데 이 때 한 레이어를 쌓을 때 마다 O(n/k)의 시간이 걸린다.(k는 커널의 크기, n은 데이터 크기)
  - Dilated convolution은 커널을 간격을 두고 배치시킴으로써 더 넓은 간격을 커버하여 계산 속도는 convolution보다 더 빨라지지만 path length가 증가한다.

- Separable convoluition은 기존의 convolution보다 복잡도를 크게 줄였고 k=n일경우 self attention 레이어와 feed forward레이어의 합과 같다.

⇒ 기본적으로 convolution은 커널을 고려해주어야 하므로 Recurrent보다 계산량이 더 많이 필요하다.

김민재

5. Training

Transformer 모델은 WMT 2014 영-독 및 영-불 번역 데이터셋을 사용하여 학습되었다.

- **데이터 및 배칭**:
  - 영-독: 450만 개 문장, BPE(Byte-Pair Encoding) 사용, 약 37,000개의 공유된 단어 사전.
  - 영-불: 3,600만 개 문장, 32,000개 단어 사전.
  - 각 배치는 약 25,000개의 소스 및 타겟 토큰 포함.
- **하드웨어 및 학습 일정**:
  - 8개의 NVIDIA P100 GPU 사용.
  - 기본 모델: 12시간(10만 스텝), 빅 모델: 3.5일(30만 스텝).
- **최적화 기법**:
  - Adam 옵티마이저 사용 ($\beta1 = 0.9$, $\beta2 = 0.98$, $\epsilon = 10^{-9}$).
  - 학습률 조정: `lrate = dmodel^(-0.5) * min(step^(-0.5), step * warmup_steps^(-1.5))`, warmup_steps = 4000.
- **정규화 기법**:
  - **드롭아웃**: 각 서브레이어 출력 및 임베딩에 0.1 확률 적용.
  - **레이블 스무딩**: 값 0.1 적용하여 모델이 더 일반화되도록 함.

6. Results

Transformer 모델은 기계 번역 및 문법 분석 과제에서 기존 모델 대비 우수한 성능을 보였다.

- **기계 번역 (Machine Translation)**
  - WMT 2014 영-독 번역:
    - Transformer (Big) 모델이 BLEU 28.4로 기존 최고 성능을 2.0 BLEU 이상 초과.
    - 기본 모델도 이전 최고 모델을 뛰어넘음.
  - WMT 2014 영-불 번역:
    - BLEU 41.0을 기록하며, 기존 최고 모델보다 4배 적은 학습 비용으로 뛰어난 성능 달성.
- **모델 변화에 따른 성능 분석**
  - 어텐션 헤드 수, 임베딩 차원, 드롭아웃 비율 등 다양한 하이퍼파라미터를 변경하며 성능을 비교.
  - 최적 세팅: 8개의 어텐션 헤드, 64차원 키-값 벡터, 1024차원 피드포워드 네트워크.
- **영어 문법 분석 (Constituency Parsing)**
  - WSJ Penn Treebank 데이터셋에서 학습.

- RNN 기반 모델 대비 높은 정확도 (F1-score 92.7) 기록.

7. Conclusion

이 논문에서는 Transformer 모델을 제안하며, 기존 RNN 기반 번역 모델을 완전히 대체하는 **어텐션 기반 시퀀스 변환 모델**을 소개했다.

1. **완전한 어텐션 기반 모델**
   - RNN 없이 **Multi-Head Self-Attention**을 활용하여 전역적인 의존성을 효과적으로 모델링.
   - 병렬 학습이 가능해 연산 속도를 대폭 향상.
2. **기계 번역에서 SOTA (State-of-the-Art) 달성**
   - WMT 2014 영-독 번역에서 **BLEU 28.4**로 기존 최고 성능을 초과.
   - WMT 2014 영-불 번역에서도 **BLEU 41.8**을 기록하며, 기존보다 짧은 시간 안에 더 나은 결과 도출.
3. **다양한 NLP 태스크로 확장 가능**
   - 영어 문법 분석(Constituency Parsing)에서도 우수한 성능을 보여 **Transformer의 일반화 가능성**을 입증.

향후 연구 방향

- **다양한 입력 및 출력 형태 (이미지, 오디오, 비디오) 적용**
- **로컬 어텐션(Local Attention) 기법 연구**를 통해 긴 시퀀스를 효과적으로 처리하는 방법 탐색
- **비순차적(less sequential) 생성 방식 연구**를 통해 더 빠르고 효율적인 모델 개발