Attention Is All You Need

AI 키워드	
날짜	@2024년 10월 6일
콘텐츠	논문
태그	유런

Abstract

기존 모델: 복잡한 회귀적 or 합성곱 모델(인코더-디코더, 어텐션 포함) → transformer: 새로운 간단한 모델

어텐션만 사용

두가지 기계번역 태스크로 실험(WMT 2014 영독/영프) → 우수한 성능 달성

병렬처리, 적은 학습 시간

다양한 작업에 적용 - 크거나 제한된 데이터 모두에서 영어 구문 분석에 성공적으로 적용함으로써 볼 수 있음

Introduciton

기존 상황

RNN, LSTM, GRNN: LM, 기계번역에서 좋은 성능 → RNN과 ENC-DEC 구조 계속 연구해옴

RNN: 입출력의 심볼 위치에 따라 요소 별로 계산 → 직렬적인 계산 → 긴 입력 문장 힘듦(메모리 문제)

factorizaiton trick(행렬 분해)과 조건부 계산으로 이를 개선해옴.

근본적인 문제는 남아있음(직렬)

attention mechanism: 다양한 작업에서 시퀀스 모델이나 변환모델의 필수 요소 ← 시퀀스의 거리와 상관없이 의존성 모델링을 허용

: 이또한 거의 대부분 rnn과 결합됨

transformer

입출력에 대한 의존성 탈피(=한번에 입력 받을 수 있음)

병렬화

적은 시간과 컴퓨팅 자원으로 sota 달성

Model Architecture

기존 상황

좋은 성능의 모델들 다 인코더-디코더 구조임

입출력 모두 순차적이고 이전의 심볼을 하나씩 더해가며 입력으로 쓰는 구조

transformer

전체 구조는 쌓여진 self-attention와 point-wise fully connected layer로 이루어짐

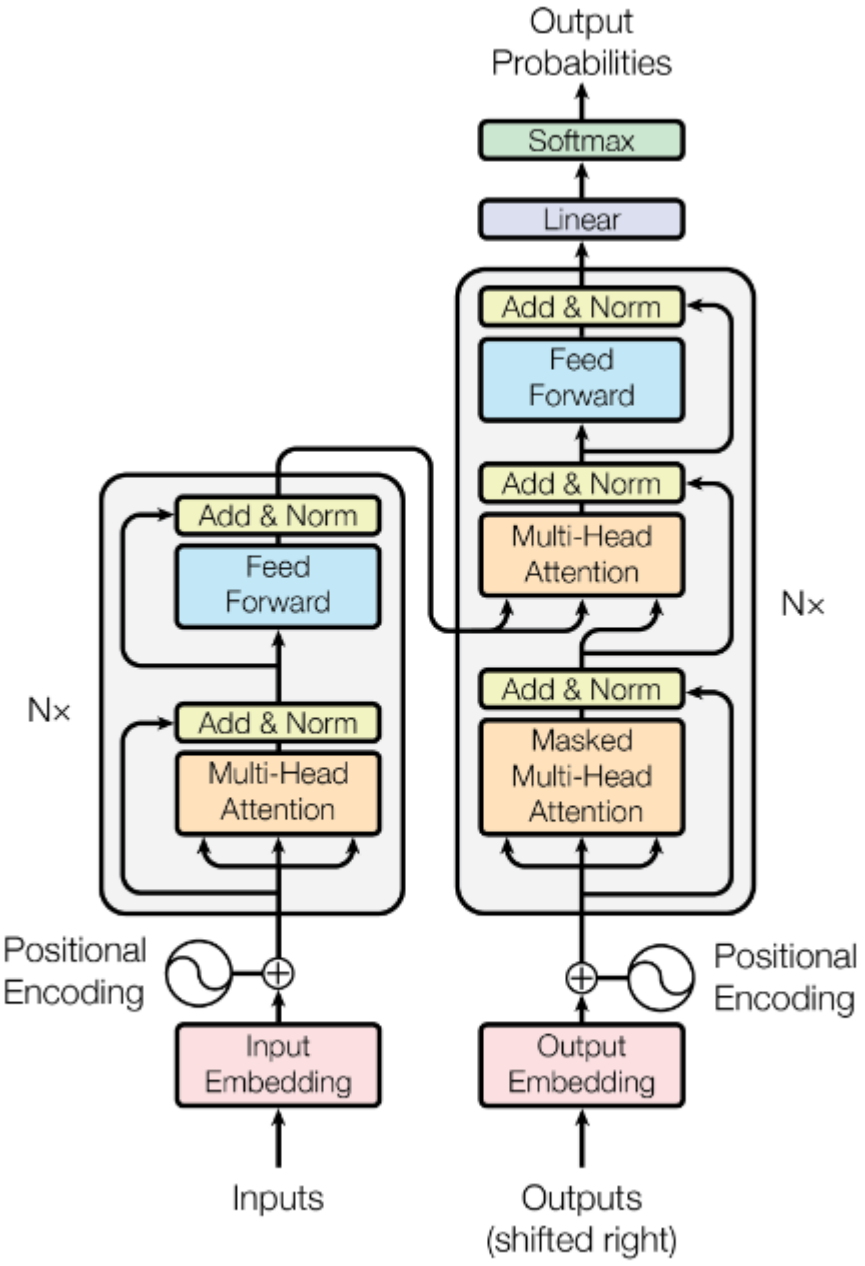


Figure 1: The Transformer - model architecture.

Encoder and Decoder Stacks

Encoder

6개의 레이어

각 레이어는 두 개의 서브 레이어

첫번째 서브 레이어: 멀티헤드 셀프 어텐션

두번째 서브 레이어: fully connected feed-forward network

residual connection과 layer normalization를 두개의 서브레어어에 각각 적용

각 서브 레이어의 output은 LayerNorm(x + Sublayer(x))

residual connection을 편하게 쓰기위해 모든 서브레이어와 임베딩 레이어의 차원은 d_{model}=512

Decoder

6개의 레이어

각 레이어는 세 개의 서브 레이어

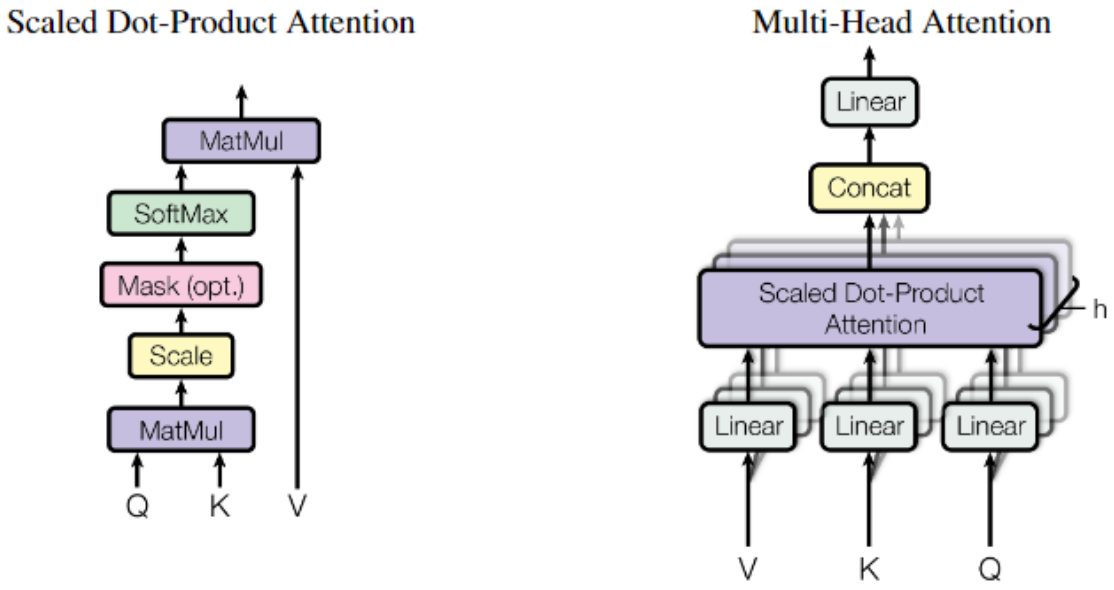
첫번째 서브 레이어: 마스크드 멀티헤드 셀프 어텐션 - 다음 위치를 참조하지 않도록 마스킹 (=i 번째 포지션은 i-1번째의 포지션만 볼 수 있도록 함)

두번째 서브 레이어: 인코더에서 encoder의 출력에 대해 멀티헤드 어텐션

세번째 서브 레이어: fully connected feed-forward network

residual connection과 layer normalization은 인코더와 똑같이 적용됨

Attention



query, key-value pair

output은 values의 weighted sum

weight는 query와 상응하는 key에 의해 계산

Scaled Dot-Product Attention

input: queries, keys(d_k차원), values(d_v차원)

query와 모든 keys를 dot product 하고 $\sqrt{d_k}$ 로 나눔 → 소프트맥스 적용하여 weight 구함

matrix화: queries → Q, keys → K, values → V

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

가장 많이 사용되는 어텐션: additive attention, dot-product attention

transformer에 활용되는 어텐션은 dot-product attention에 scaling 추가한 것

addictive attnetion은 ffn 사용해서 계산

두 어텐션 비교했을 때 이론적인 complexity는 비슷하지만 dot-product가 더 빠르고 공간 효율적임 ← 최적화된 행렬 곱 코드를 사용

하지만 d_k 크고 scaling 없으면 dot-product가 더 안 좋음

d_k 크면 소프트맥스 함수의 그래디언트가 매우 작아질 수 있다고 생각 → 스케일링 적용

Multi-Head Attention

d_{model} 차원의 query, key, value가 아니라 d_k, d_k, d_v으로 선형변환을 h번 하는 것이 더 좋음

이유 1: 병렬적 처리 가능

이후 어텐션 함수를 병렬적으로 적용하여 d_v 차원의 output 생성 → 이는 concat 되고 한 번 더 선형변환

이유 2: 다양한 표현 학습

다른 위치의 다른 표현에 대해 함께 참조할 수 있는 효과. 단일 어텐션에서는 평균화가 이를 억제함

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

선형 변환 파라미터는 W_i^Q, W_i^K, W_i^V

h=8, $d_k = d_v = d_{model}/h$ = 64를 사용

각 head에 대해 차원이 줄어들기 때문에 단일 어텐션과 계산 비용은 비슷

Applications of Attention in our Model

3가지 방식으로 멀티헤드 어텐션 활용

1. encoder-decoder attention

   queries는 이전 디코더 레이어에서, memory keys와 values는 encoder 출력에서 옴 → decoder의 모든 포지션이 input sequence의 모든 포지션을 attend

   seq2seq에서 사용하는 일반적인 어텐션 메커니즘이라고 생각하면 됨

2. self attention in encoder

   keys, values, queries가 모두 같음. 이전 인코더 레이어의 출력 → encoder의 모든 포지션이 이전 encoder 출력의 모든 포지션을 attend

3. self attention in decoder

   decoder의 모든 포지션이 현재까지 decoder 출력의 모든 포지션을 attend

   좌측으로의 정보 흐름(우리는 글을 왼쪽에서 오른쪽으로 읽음)을 방지해야 하므로 마스킹 사용(=auto-regressive property를 보존하기 위함)

   scaled dot-product attention 직후에 마스킹 적용(= $\frac{QK^T}{\sqrt{d_k}}$에 마스킹 적용)

   음의 무한대로 마스킹 → 소프트맥스 거치면 0이 됨

Position-wise Feed-Forward Networks

선형변환 → ReLU 활성화 함수 → 선형 변환로 이루어짐

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

각 레이어에 적용되는 FFN은 다른 파라미터를 가짐

kernel size 1의 convolution을 두번 사용하는 것과 같음

입출력의 차원은 $d_{model} = 512$, 내부 linear의 차원은 $d_{ff} = 2048$

Embeddings and Softmax

다른 시퀀스 변환 모델과 비슷하게 학습된 임베딩 활용

디코더의 output을 예측된 다음 토큰의 확률로 변환하기 위해 일반적인 학습된 선형변환과 소프트맥스 사용

두개의 임베딩 레이어, pre-softmax linear transformation의 가중치를 공유

임베딩 레이어에서 가중치에 $\sqrt{d_{model}}$을 곱함

Positional Encoding

recurrence와 convolution을 안 갖고 있기 때문에 모델이 순서를 사용할 수 있게 하기 위해서 상대 혹은 절대적인 위치 정보를 토큰 시퀀스에 주입해줘야 함

input embedding에 positional encoding 더해줌

이를 encoder와 decoder 가장 시작에 넣어줌

embedding 차원과 같은 d_{model}의 차원을 가짐

positional encoding을 구현하는 방법은 다양하지만 이 논문에서는 다른 주기의 sine, cosine 함수를 이용

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

positional encoding의 각 차원은 sinusoid(사인 곡선)에 대응됨

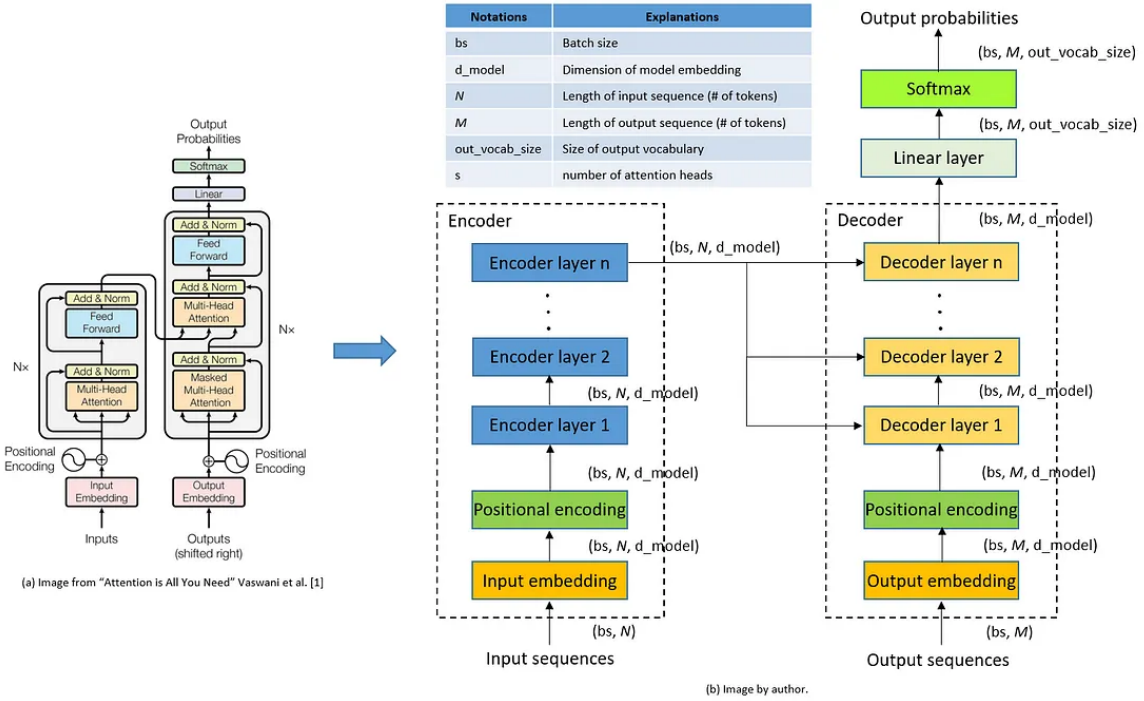
모델이 상대적 위치에 따라 쉽게 attend 할 것이라고 생각 ← PE_{pos}PE_{pos} PE_{pos+k}를 나타낼 수 있음

학습된 positional embedding을 사용해서도 실험해봄

두 방법 거의 같은 결과를 냄

sinusoid한 방법이 학습 시 만난 시퀀스보다 더 긴 시퀀스에 대해서도 대응할 수 있기 때문에 sinusoid한 방법 사용

▼ Matrix Shape



Multi-Head Attention 내부에서 shpae 변화

입력 Q, K, V: (bs, N, d_model)

헤드 나누기: (bs, N, h, d_head) → (bs, h, N, d_head)

$\frac{QK^T}{\sqrt{d_k}}$: (bs, h, N, N)

$Softmax(\frac{QK^T}{\sqrt{d_k}})$: (bs, h, N, N) → attention weight

$Softmax(\frac{QK^T}{\sqrt{d_k}})V$: (bs, h, N, d_head) → result

헤드 합치기: (bs, h, N, d_head) → (bs, N, h, d_head) → (bs, N, d_model)

FFN 내부에서의 shape 변화

입력: (bs, N, d_model)

첫번째 linear: (bs, N, d_ff)

relu: (bs, N, d_ff)

두번째 linear(bs, N, d_model)

Why Self-Attention

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

self-attention과 rnn/cnn을 세 가지 관점에서 비교

1. 레이어 당 전체 계산 복잡도

   보통 n이 d보다 작기 때문에 self-attention이 complexity가 더 작다고 말할 수 있음

2. 병렬 계산 가능 정도

   필수적인 sequential operation의 최소값을 통해 구함

3. path length between long-range dependencies in the network

   입력과 출력 사이의 path가 짧을수록 long-range dependencies(=긴 시퀀스 입력)를 학습하기 쉬움

   (두 단어 사이의 관계를 알 수 있도록 하는 과정의 개수)

추가적으로 self-attention은 해석이 더 쉬운 모델을 만들어냄(=어텐션 시각화 가능)

▼ 결과 상세 분석

1. Self-Attention
   - Complexity per Layer: Self-Attention은 시퀀스 내의 모든 단어가 다른 모든 단어와 상호작용하는 방식. n개의 단어가 있는 문장에서 각 단어는 n개의 다른 단어와 상호작용. 각 상호작용은 d차원의 벡터(단어 임베딩)를 이용해서 계산되기 때문에, 복잡도는 n^2d
   - Sequential Operations: 순차적인 계산이 필요하지 않음. 한 번의 연산으로 모든 상호작용 계산 가능
   - Maximum Path Length: 어떤 두 단어 사이의 관계도 한 단계에서 직접 연결되기 때문에, 정보가 전달되는 데 최대 한 단계
2. Recurrent (순환 신경망)

- Complexity per Layer: 시퀀스의 각 단어를 차례로 처리. 시퀀스의 길이가 n이라면, n번의 계산이 필요. 각 계산에서, 이전 상태와 현재 단어의 정보를 결합하는데, 이때 d차원의 벡터를 사용해 복잡도가 d^2. 따라서 전체 복잡도는 nd^2
- Sequential Operations: 앞 단어가 처리되어야 그 다음 단어를 처리할 수 있으므로 시퀀스 길이만큼의 순차적인 계산이 필요함
- Maximum Path Length: 정보가 시퀀스의 처음부터 끝까지 전달되려면, 모든 단어를 거쳐야 함. 그래서 최대 n단계가 필요

3. Convolutional (컨볼루션 신경망)
    - Complexity per Layer: 컨볼루션 신경망(CNN)은 커널 크기 k만큼의 단어 그룹을 한 번에 처리. 시퀀스의 길이가 n이라면, n개의 위치에 대해 커널이 적용. 각 커널 적용에서 d차원의 벡터 연산이 필요하기 때문에, 이 연산의 복잡도는 d^2. 전체 복잡도는 knd^2
    - Sequential Operations: 모든 위치에서 커널이 동시에 적용될 수 있으므로 병렬 처리가 가능
    - Maximum Path Length: 커널 크기에 따라 한 번에 여러 단어를 처리할 수 있어 정보가 로그 스케일로 전달
4. Restricted Self-Attention
    - Complexity per Layer: Restricted Self-Attention에서는 각 단어가 가까운 r개의 이웃과만 상호작용. n개의 단어가 있고, 각 단어는 r개의 이웃과 상호작용하므로 n×r의 상호작용이 필요. 이 연산은 d차원의 벡터를 이용하므로, 전체 복잡도는 rnd
    - Sequential Operations: 모든 단어가 동시에 병렬로 처리될 수 있음
    - Maximum Path Length: 각 단어가 r개의 이웃과만 상호작용할 수 있기 때문에, 전체 시퀀스에 정보를 전달하려면 여러 단계가 필요

Training

Training Data and Batching

1. standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs

    BPE 인코딩

    단어 37000개

2. larger WMT 2014 English-French dataset consisting of 36M sentences

   32000 word-piece vocabulary

시퀀스 길이에 따라 배치 생성

각 배치는 25000개의 소스/타겟 토큰으로 이루어짐

Hardware and Schedule

h 8 NVIDIA P100 GPUs

시작 모델: 각 스텝 0.4초, 총 100,000 스텝 훈련 → 12시간

최종모델: 각 스텝 1초, 총 300,000 스텝 훈련 → 3.5일

Optimizer

Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$

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

warmup_stemps = 4000

Regularization

1. **residual dropout**

   각 서브레이어의 출력에 드롭아웃 적용

   임베딩과 positional encoding의 합에도 드롭아웃 적용

   기본 모델 $P_{drop} = 0.1$

2. **label smoothing**

   $\epsilon_{ls} = 0.1$

   perplexity는 저하되고 모델이 더 불확실해 지지만 정확도와 BLEU score가 향상