자연어심화논문구현 강하리 1주차

Created time	@2024년 3월 4일 오후 1:58
활동기간	13기 봄 전반기
작성 주차	1주차
Person	강 강하리
봄 전반기팀	자연어심화논문구현1팀
작성완료	☑
최종 편집 일시	@2024년 3월 4일 오후 1:59
최종 편집자	강 강하리

Transformer : Attention is all you need

Positional Encoding

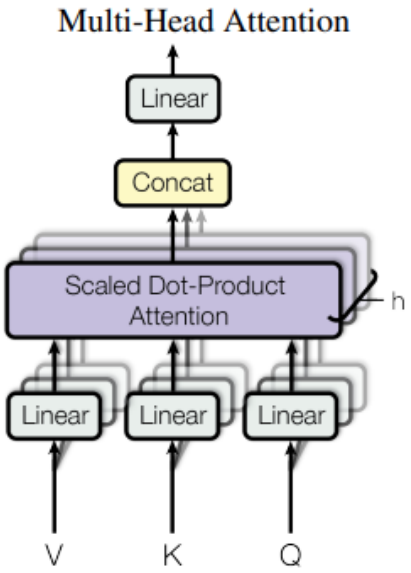
본 논문에서는 Positional encoding을 parameter를 부여하여 학습을 하는 실험과 그냥 시간에 맞게 index를 지정하는 실험을 진행하였는데, 학습 파라미터를 부여하는 것이 크게 다르지 않은 결과를 낸다는 것을 확인했다.

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

Input Embedding

단어 별 dim은 사전 내 단어의 수 + positional encoding의 dim이 더해진 값으로 고정된다.

MHA



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

Q(쿼리)와 K(key)들을 내적한 후, $\sqrt{d_k}$로 나눈 후 softmax를 지나 V에 가중치로 곱한다.

→ 그 이유는 행렬의 곱이 일어난 이후에는 분산도가 커지는데 SOFTMAX를 통과할 때 GRADIENT의 소실을 막기 위해 사용한다.

- Q(Query) : 얼마나 관련있는지를 물어볼 단어
- K(Key) : Q와의 관련성을 비교할 단어 후보들
- V(Value) : K의 의미 벡터들

- Dot-product attention
  - scaling factor를 제외하고는 우리 알고리즘과 동일하다.
- Additive attention

- feed-forward network와 하나의 hidden layer를 사용하여 계산된다.

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

하나의 attention function을 사용하기보다는 h번 다른 learned linear projection을 사용하는 것이 더 효과적인 것을 발견하였다. MHA를 통해 같은 정보에 대해서 서로 다른 위치의 representaion subspaces에서 정보를 얻을 수 있게 된다. (=집단지성)

> 💡 **다시 말해. head별로 512차원에서 64차원으로 변화시킬 때, 그 '변환'을 결과가 좋도록 학습하는 것을 의미한다. 결국 Query가 어떤 key에 주목하면 번역을 잘 할지 그 수치(양,음)를 잘 조절하여 알아낸다.**

Transformer는 MHA를 3가지 다른 방법으로 사용한다.

- Encoder-Decoder Attention
  - Q는 이전 decoder layer으로부터 오고, K와 V는 encoder의 output으로부터 온다. 이것을 통해 decoder가 input sequence의 모든 위치에 접근할 수 있게 해준다.
- Encoder
  - K,V,Q는 같은 place로부터 오며, encoder의 each position 이전 layer의 모든 position에 접근할 수 있다.
- Decoder

  decoder의 each position 이전 layer의 모든 position과 현재 위치까지 접근할 수 있다. 아직 주어지지 않은 정보에 대한 접근을 막고자 masking out을 사용한다.

Position-wise FFN

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

This consists of two linear transformations with a ReLU activation in between.

cf)layer norm은 단어 별 차원을 normalize한다.

예를 들어 [문장 x 단어 x 차원]이라면 차원을 normalize하는 것이다.

Encoder and Decoder

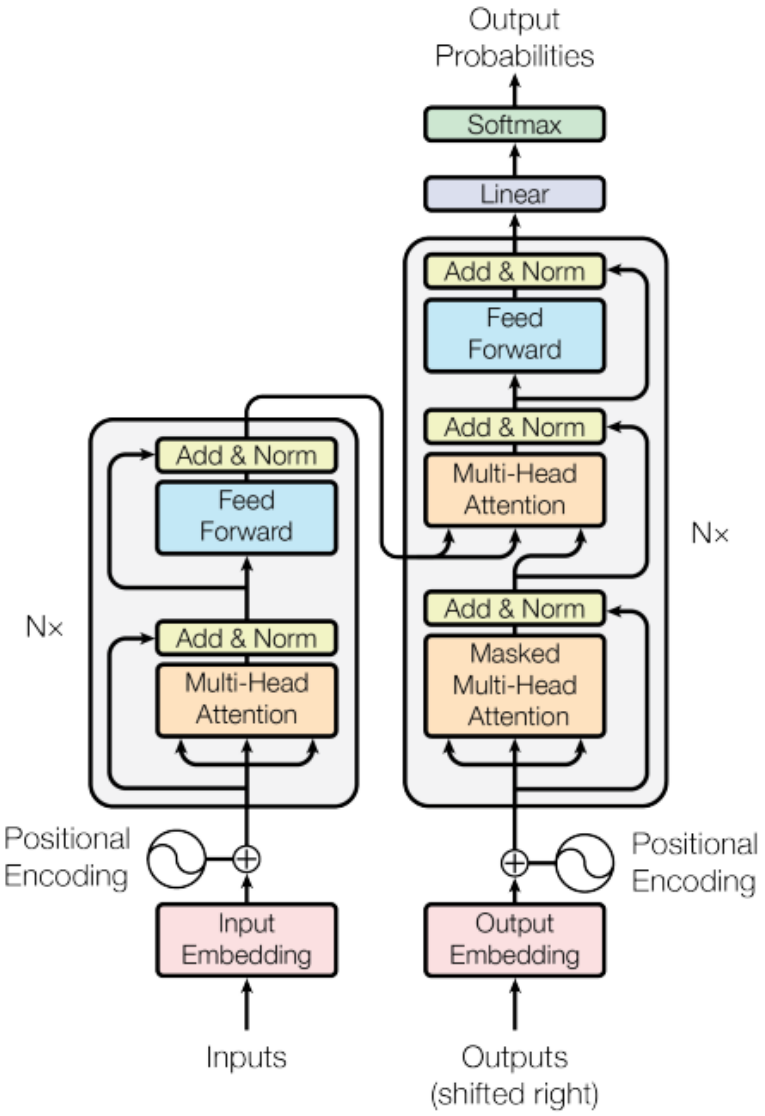


Figure 1: The Transformer - model architecture.

▼ Encoder

The encoder is composed of a stack of N = 6 identical layers.

Each layer has two sub-layers.

The first is a **multi-head self-attention mechanism**, and the second is a simple, **positionwise fully connected feed-forward network.**

We employ a residual connection around each of the two sub-layers, followed by layer normalization.

That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself.

Encoder는 sub-layer별로 skip-connection을 적용하였고, 이는 layer간의 변화가 0이 되는 방향으로 학습하여 깊이와 비선형성을 늘린 것이다.

▼ Decoder

The decoder is also composed of a stack of N = 6 identical layers.

In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs **multi-head attention over the output of the encoder stack**.

Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

뒷 단어를 보지 못하게 Masking이 필요하다. 그 과정이 softmax를 통과하기 직전 엄청 작은 음수로 변화하여 softmax를 지나면 값이 0으로 설정되게 하는 것이다.

This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Encoder의 마지막 output layer에서는 K와 V를 가져오고, maskedMHA를 통과한 값에서 Q imbedding vector를 사용한다.

여기서 Q는 현재 target 단어에 대한 뜻을 잘 담고있는 vector이고, K와 V는 입력문장에 모든 단어에 대한 정보를 담고 있는 vector이다.

즉, 과정을 설명하면 우선 decoder 문장 내 단어를 파악한다. 그리고 입력문장에서 어떤 단어를 주목할지를 판단한다. (=입력문장의 맥락은 봐가면서) 그리고 FFN을 지나면서 다음 단어를 예측하는 것이다.