Transformer-Attention is All You Need (2017)

FOM

Focus On Data Mining

INDEX

01 Introduction

Encoder,Decoder의 한계점

- Encoder,Decoder (seq2seq)의 한계점 :
  - 하나의 고정된 크기의 벡터에 모든 정보를 압축하려고 하니까 정보 손실이 발생
  - RNN의 문제인 gradient vanishing 문제가 존재
    -> 입력 문장의 길이가 길어지면 output sequence의 정확도가 떨어짐

- Attention의 main idea
  - RNN의 기능을 오로지 matrix multiple ( )으로 대체
  - 행렬 곱조차 각자 수행하는 게 아닌 병렬적으로 수행해서 한 번에 처리함
  - 병렬 처리를 통해 연산 시간을 효율적으로 줄임
  - RNN은 문장을 왼쪽에서 오른쪽으로 순차적으로 학습함 -> encoder 문장의 첫 번째 단어와 encoder 문장의 마지막 단어 및 decoder 문장의 첫 번째 단어 간의 관계를 RNN은 학습하지 못함 -> 반면, Attention을 한번에 병렬적으로 수행 시 문장의 위치와 관계 없이 모두 영향력을 가지면서 학습 가능

01 Introduction

Key point of paper

- RNN Encoder-Decoder를 완전히 대체
  - 병렬화(paralling)를 통해 RNN의 기능을 한번에 수행
  - RNN의 역할인 각 단어들의 의미를 RNN보다 정확하게 분석

- Attention mechanism
  - RNN을 대체하기 위해 사용된 mechanism
  - RNN기반보다 긴 문장들을 보다 잘 분석
  - Self-Attention, Multi-head Attention을 사용해 적은 연산량으로 더 많은 양과 질의 분석을 수행

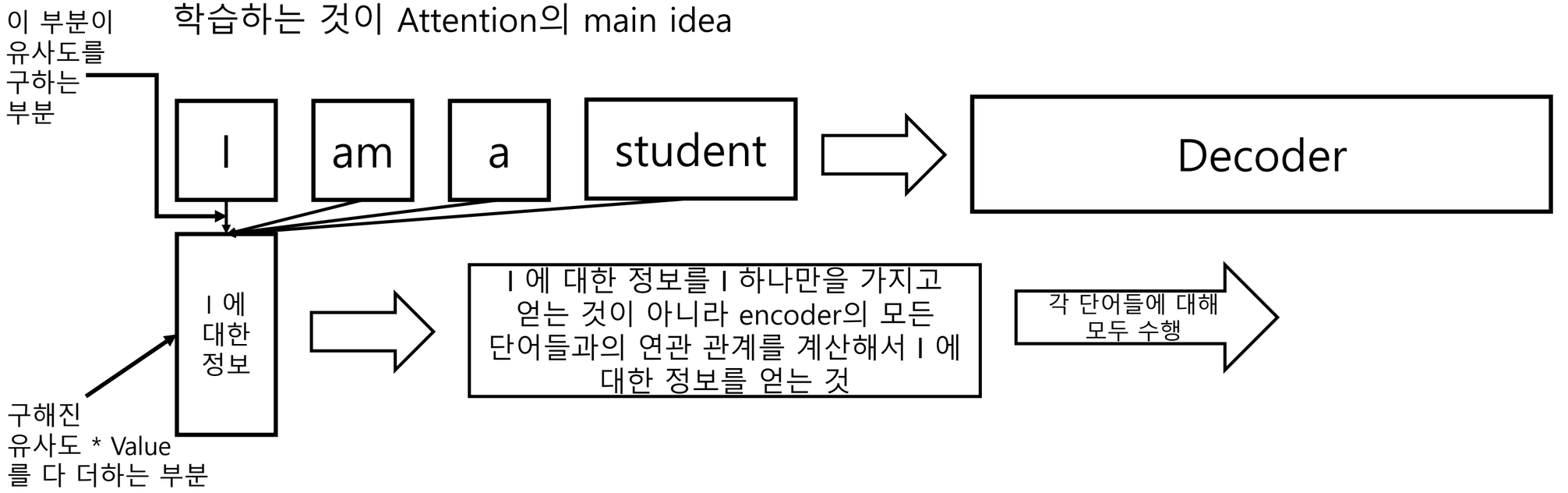
- Poistional encoding
  - RNN의 가장 큰 장점인 문장속 단어의 등장 순서,위치를 분석하기 위한 장치
  - 간단한 연산으로 RNN의 역할을 대체

- Residual connection
  - 위의 positional encoding의 정보를 유지하기 위해 사용

02 Related Works

Attention!

- Attention이란,
  Encoder 문장의 각 단어들이 decoder 문장의 어떤 단어로 각각 대응이 되는지를 모델이 학습하기 위해 고안된 mechanism
  -> **Decoder에서 해당 단어를 번역할 때 encoder의 어떤 단어에 집중 해야 하는지를** 결정하는 모델
  -> 예로 들어 "I am a student"라는 문장을 번역할 때,
  I 에 대한 hidden state, am 에 대한 hidden state, a 에 대한 hidden state등 각 단어들에 대하 hidden state들을 출력 -> 출력을 할 때에 문장 속 다른 단어들이 현재의 단어와 얼마나 연관이 있는지를 같이 학습하는 것이 Attention의 main idea



Attention은 주어진 Query(Q)에 대해 모든 Key(K)와의 유사도를 구함

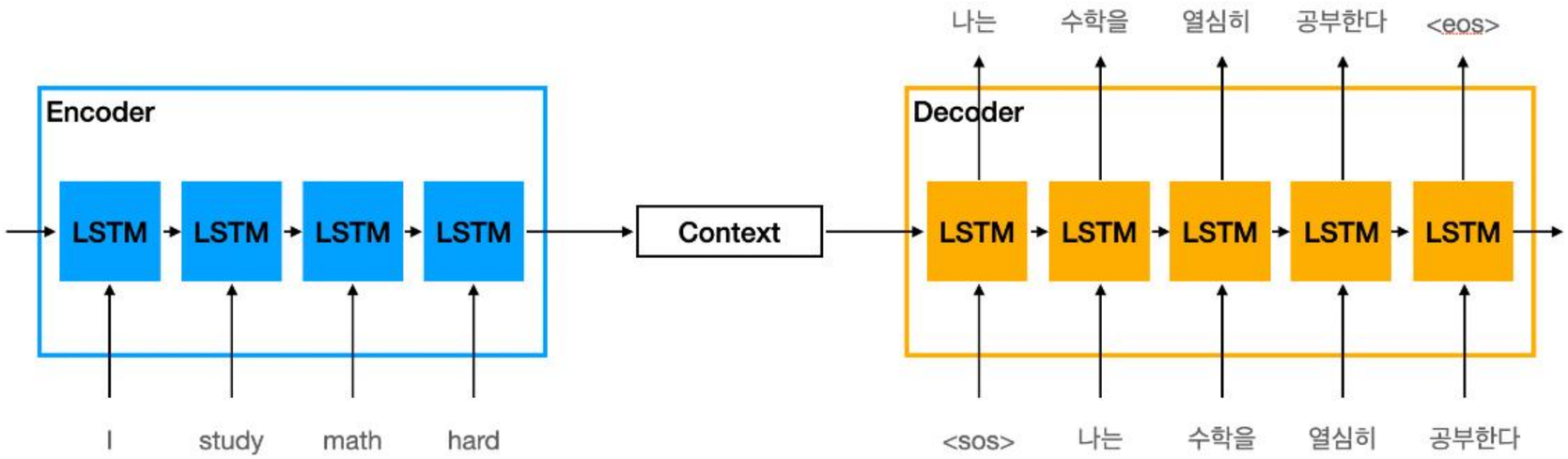
이를 통해 구해진 유사도를 각각의 Key와 매핑된 Value(V)를 곱해 Value에 유사도를 반영시킴
-> 유사도를 반영 시키는 방법은 유사도와 Value를 곱해 softmax를 취함
-> 유사도가 반영된 Value들을 전부 더해서 최종적인 Attention vector를 생성

02 Related Works

RNN Encoder-Decoder

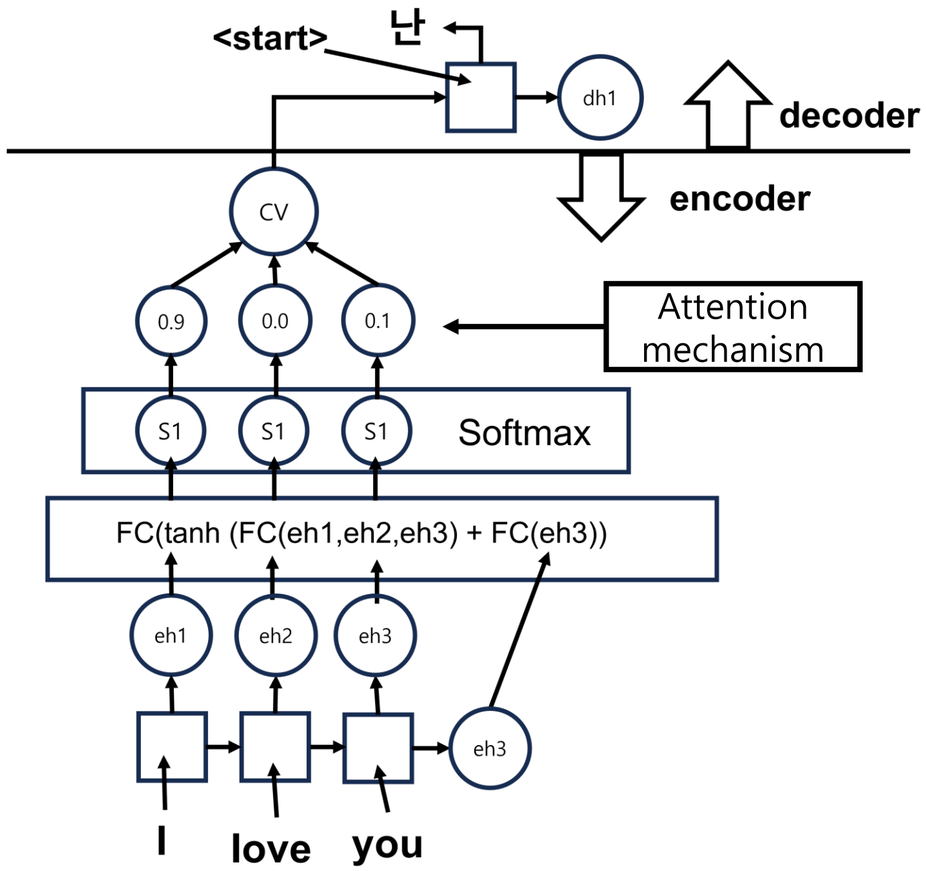
- RNN Encoder-Decoder

RNN Encoder-Decoder 는 encoder cell 내부에 RNN을 넣어서 문장 속 단어들의 단어 등장 위치 및 순서를 고려해서 context vector를 만듦 decoder cell 내부에도 RNN을 넣어서 context vector를 단어들의 등장 순서에 따라서 맞게 번역 시킴



02 Related Works

RNN Encoder-Decoder with Attention



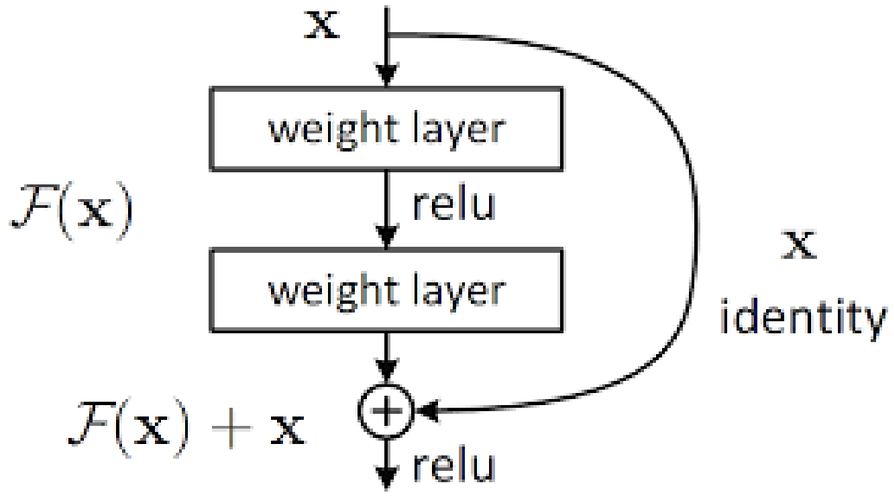
- RNN with Attention

  RNN model에서 context vector를 생성할 때 Attention mechanism을 사용해서 생성
  각 hidden state의 output*softmax(output)의 결과값들을 concate시켜 Attention으로써 사용 ->

02 Related Works

Residual connection

- Neural Network 학습시 발생되는 gradient vanishing,exploding 문제의 해결책



H(x)=F(x)+x 인 상황에서 residual 의 main idea는 H(x)가 x로 수렴하는 즉, F(x)가 0으로 수렴하도록 만드는 것이 residual connection의 목적

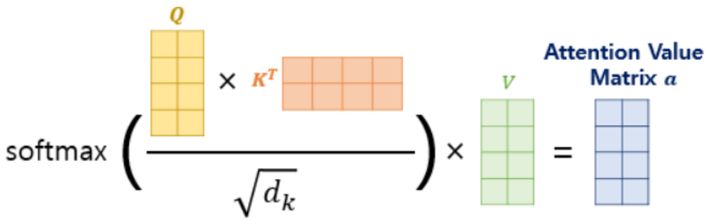
간단히 말하면 딥러닝 학습시 몇 개의 convolutional layer를 지난 후의 output data를 초기의 input data와 connection 하여서 최대한 초기의 input값 x를 보존시키는 것

이번 논문에서는 residual connection을 초기의 값들을 보존하는 장치로도 사용되지만 주요 기능은 positional embedding의 정보를 유지하기 위함

03 | Proposed Method

Self-Attention

- Transformer에서 가장 중요한 method



Scaled Dot-Product Attention

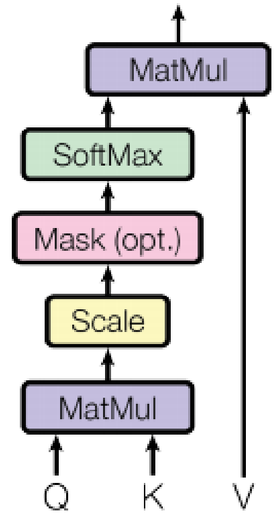
- Attention의 Query,Key,Value를 자기 자신(x_t)에서 추출해서 사용
- input word에 각각의 가중치 행렬(w_Q, w_K, w_V)를 곱해서 각각 Q,K,V를 구함
- 자기 자신(x_t)의 Query와 다른 단어(x_1, x_2..., x_{t-1}, x_{t+1}, …., x_n)의 Key와의 유사도를 구함
- 그 후 다른 단어(x_1, x_2..., x_{t-1})의 Value와 곱해 자기 자신(x_t)의 Attention vector를 생성 해당 Attention vector는 자기 자신(x_t)의 함축적 의미(문장 전체 단어들을 즉, 맥락을 고려한 의미)

- Attention mechanism 수식

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

- $\sqrt{d_k}$로 나눠주는 이유는 문장의 길이가 길어질 수록 Key의 차원이 커져서 행렬곱의 크기가 너무 커지는 것을 방지하기 위해 scaling 진행

- 오른쪽의 Scaled Dot-Product Attention이 해당 논문에서 주장한 Attention



03 | Proposed Method

Positional encoding

- Positional encoding
  - RNN의 기능 중 하나인 해당 단어가 문장 속 어느 위치,순서에 대한 정보를 얻기 위한 method
  - 위치에는 상대적/절대적(relative/absolute) 위치가 있음
  - positional encoding은 상대적 위치(relative position)을 사용
  - 상대적 위치를 표현하기 위해 sinusoid(사인 곡선)을 사용
  - encoding vector의 위치가 sin,cos에 입력됨 -> sin과 cos의 골 값에 따라 단어의 위치 정보를 얻을 수 있음

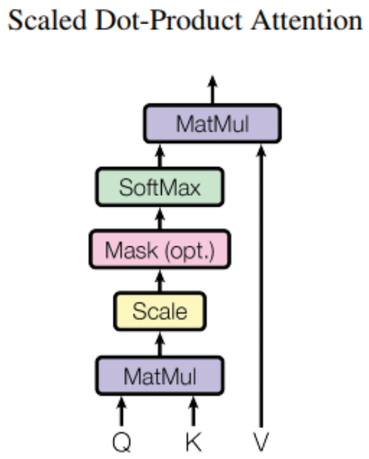
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

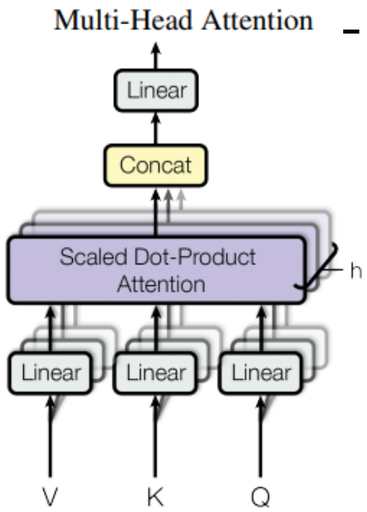
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- sin,cos 함수를 사용하는 이유
  - sin,cos 함수는 위의 수식에 의하면 2π to 10000 · 2π의 길이를 가짐 -> sin,cos함수는 해당 길이 안에서 위치에 따라 -1~1값을 출력하기 때문에 상대적인 위치를 구하기 원할
- 상대적 위치를 사용하는 이유
  - 상대적 위치(sin,cos함수)를 사용하는 이유는 모델이 번역시에 학습 데이터보다 긴 문장이 입력 되었을 때, 원활하게 번역을 진행하기 위함
  - 즉, 훈련 데이터의 길이가 k일 때, 번역 문장이 k+α 일 때에 원활하게 번역을 하기 위함

03 Proposed Method

Multi – Head Attention



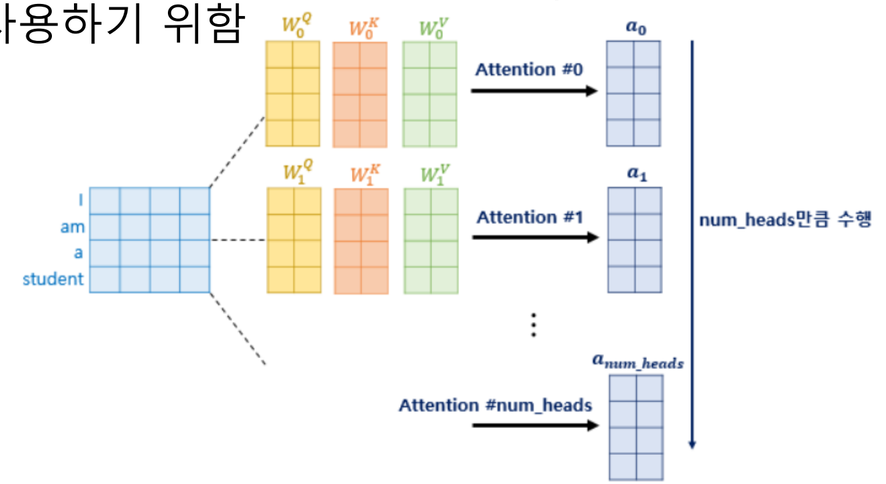


- 앞의 Scaled Dot-Product Attention(Self-Attention)의 반복

- 즉, Transformer의 최대 장점인 병렬적 수행을 진행
- Attention layer들을 병렬적으로 여러 개 연결해서 수행
- Encoder,Decoder들은 각각 8개의 Attention layer들을 병렬적으로 붙여서 한번에 수행
- 각각들의 Attention output들을 concatenate시킴
- concatenate된 matrix들을 projection 시켜 처음의 input dimension과 같도록 만들어 줌
- input dim과 같도록 만드는 이유는 Attention의 output을 다시 Attention의 input으로 사용하기 위함

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

- MultiHead Attention의 수식
Transformer에서는 h=8로 설정해서 사용



03 Proposed Method

Applications of Attention in our Model

-The Transformer uses multi-head attention in three different ways:

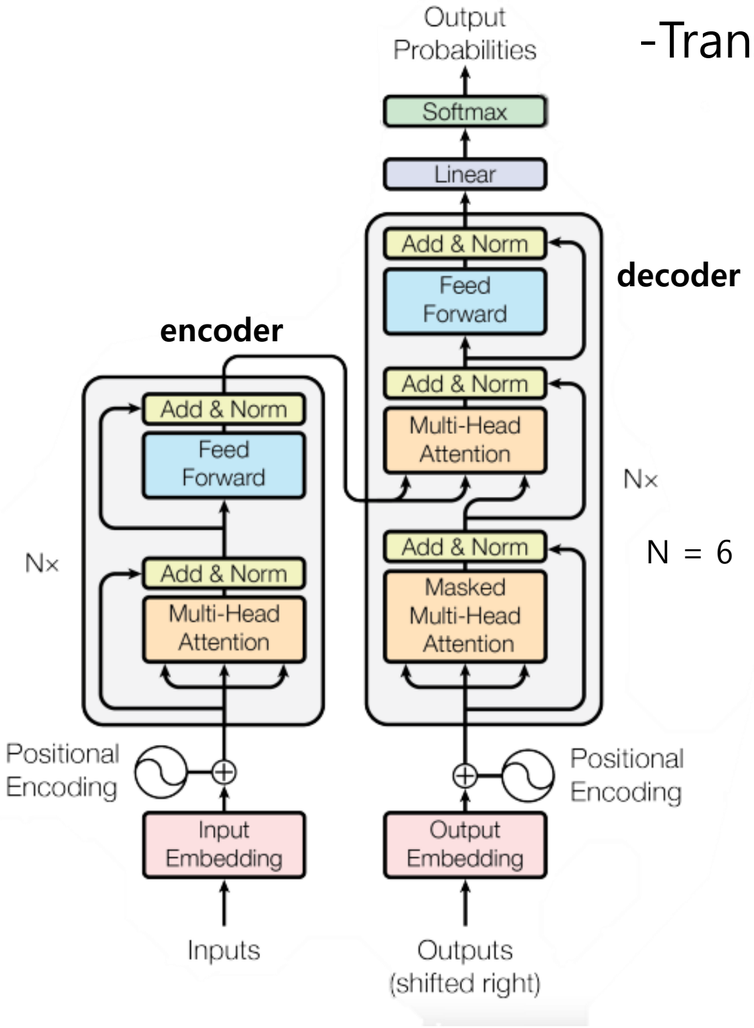
1. Encoder-decoder attention layer들은 이전의 decoder layer의 query, memory key,value들을 encoder의 output을 사용해서 진행 -> seq2seq with attention model과 비슷한 방식

2. Encoder attention layer들에서는 self-attention mechanism을 진행 -> self-attention layer들은 key,value,query를 모두 previous encoder layer의 output에서 추출

3. Decoder도 Encoder와 같은 구조를 가지고 있지만 decoder는 지금까지 학습해 온 결과값만 가지고 다음을 학습함
   즉, 현재 시점 이후의 내용들은 학습하지 않음 -> Masking method(setting to -∞)를 사용하여
   leftward information이 학습되지 않도록 조절함

03 Proposed Method

Final Encoder-Decoder



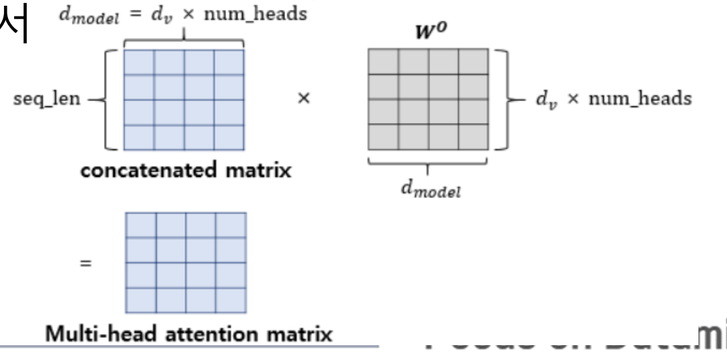
-Transformer Encoder-Decoder

- Encoder-Decoder of transformer에선 각각 6개의 encoder,decoder layer들이 존재

- encoder-decoder들의 layer 속 sub layer들 사이에는 residual connection이 있음, 해당 residual connection은 positional encoding information을 보존함

- encoder와 다르게 decoder의 layer들의 첫 attention layer은 leftward information의 유입을 막기 위해 masked attention을 수행

- Transformer에서 가장 중요한 점은 encoder,decoder layer들은 모두 input,output의 dim이 같은 것 -> dim이 같아야 과거의 output값을 현재의 새로운 input값으로 사용할 수 있어서



03 Proposed Method

Label smoothing

- 간단한 작업으로 transformer의 BLEU score을 상승 시키기 위해 적용

Softmax의 출력값을 통해 output값을 출력할 때 one-hot encoding 처럼 [0,1,0,0,0,0] 형태로 출력하는 것이 아닌 [0.01, 0.95, 0.01, 0.01, 0.01, 0.01] 로 출력해 나중의 정보 손실들을 줄임
-> 기존 출력값이 1 인 부분에는 1에 가까운 0.95를 출력, 기존 출력값이 0 인 부분에는 0.01등 0에 가까운 수를 출력하여서 기존의 출력값과는 같은 의미를 가지지만 정보의 손실을 줄 일수 있는 labeling method

04 | Experiment

05 Conclusion

완전한 RNN의 제거 및 병렬처리의 등장

- RNN의 기능을 Attention mechanism으로 완전히 대체
- Attention을 병렬적으로 반복 수행시킴으로써 성능을 상승

- 가장 의미가 있는 부분은 연산량이 많고 복잡했던 RNN기반의 모델들을 단순한 행렬곱으로 해결하여 혁신적으로 연산량을 줄이고 성능을 높임

Q & A

감사합니다