딥러닝 프레임워크

Transformer

송민아, 정가현

| Date. 2024..06.04 |

TABLE OF CONTENTS

목차 소개

01 Transformer란?

Attention Is All You Need

- 2017년 구글에서 발표한 논문
- 기존의 seq2seq의 인코더와 디코더를 발전시킨 딥러닝 모델
- RNN을 사용하지않는다
- 기존의 RNN보다 성능이 좋고 학습이 빠르다

번역 및 자연어 이해 작업

이전의 순환 신경망(RNN)이나 장기 단기 메모리(LSTM)보다 뛰어난 성능과 병렬 처리 기능을 제공

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

01 Transformer란?

중요성

1. 병렬성

1-1 효율성

Transformer 모델은 순차적으로 데이터를 처리하는 RNN과 달리, 모든 입력 데이터를 동시에 병렬로 처리할 수 있습니다. 이는 학습과 추론 속도를 크게 향상

1-2 확장성

병렬 처리 덕분에 Transformer 모델은 대규모 데이터와 복잡한 작업에 쉽게 적용

01 Transformer란?

중요성

2. Attention 메커니즘

2-1 Self Attention

Self-Attention 메커니즘을 사용하여 입력 시퀀스 내의 모든 단어 간의 관계를 학습
이는 단어의 의미를 더 잘 이해하고, 문맥을 고려한 더 정교한 표현을 생성

2-2 Long-Range Dependencies

긴 문장이나 문맥에서도 중요한 정보를 잃지 않고 학습

01 Transformer란?

중요성

3. 모듈화와 유연성

3-1 Encoder-Decoder 구조

Transformer는 인코더와 디코더로 구성되어 있어 다양한 작업에 쉽게 맞춤
인코더는 입력 데이터를 이해, 디코더는 출력 데이터를 생성

3-2 모듈형 설계

다양한 구성 요소를 추가하거나 수정하여 특정 작업에 맞게 모델을 조정

01 Transformer란?

NLP 및 기타 분야에서의 역할

1. NLP

- 기계 번역
- 텍스트 요약
- 질의 응답 시스템
- 문서 분류 및 감정 분석

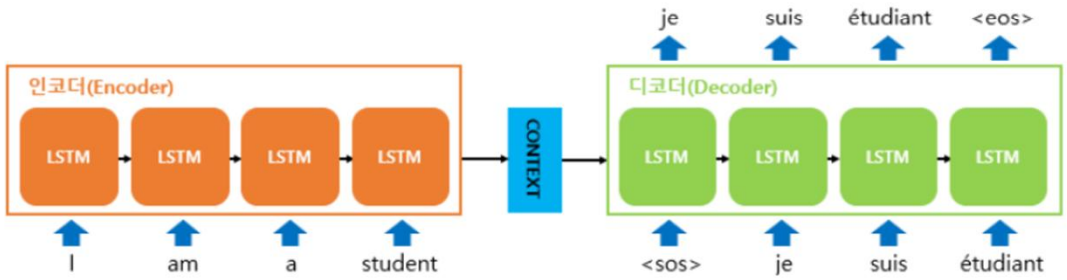
2. 기타분야

- 이미지 처리
- 음성 인식
- 생물정보학

01 Transformer란?

구조: 입력 seq → Encoder → Vector → Decoder → 출력 seq

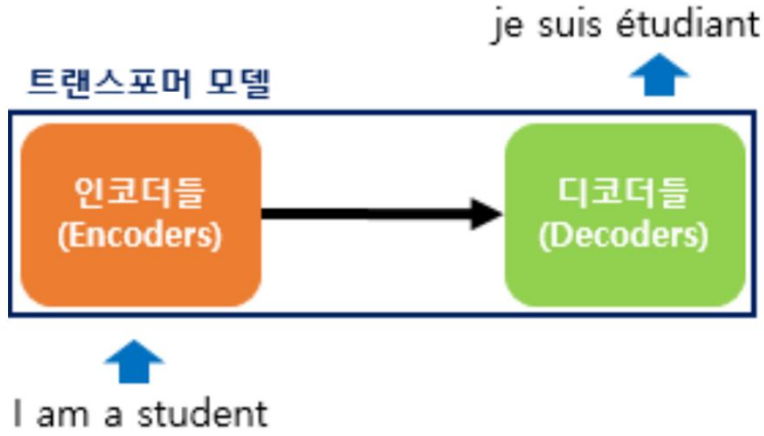
seq2seq



- 첫번째 단어부터 마지막 단어까지 인코딩
- 정확한 성능기대 어려움
- 고정길이벡터 Context Vector

VS

Transformer



- 병렬화구조로 한번에 처리
- Positional Enconding을 통해 자연어 처리
- 중간 표현 Intermediate Representation

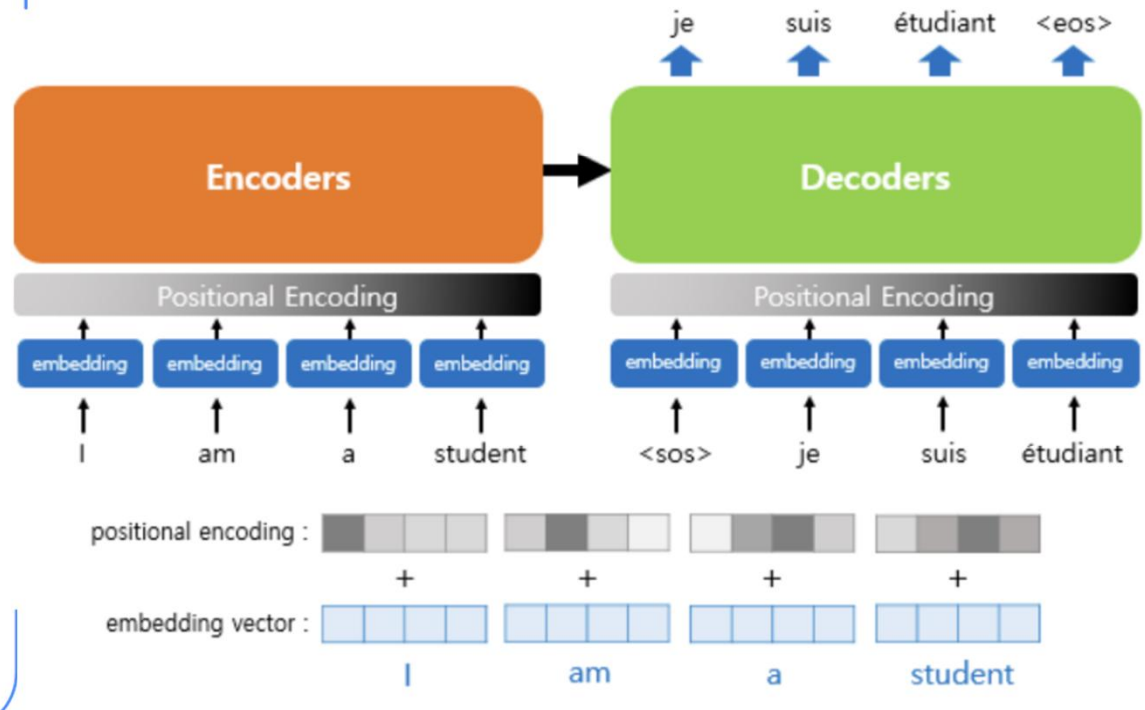
02 Positional Encoding

인코더 및 디코더 입력값마다 상대적인
위치정보를 더해주는 기술

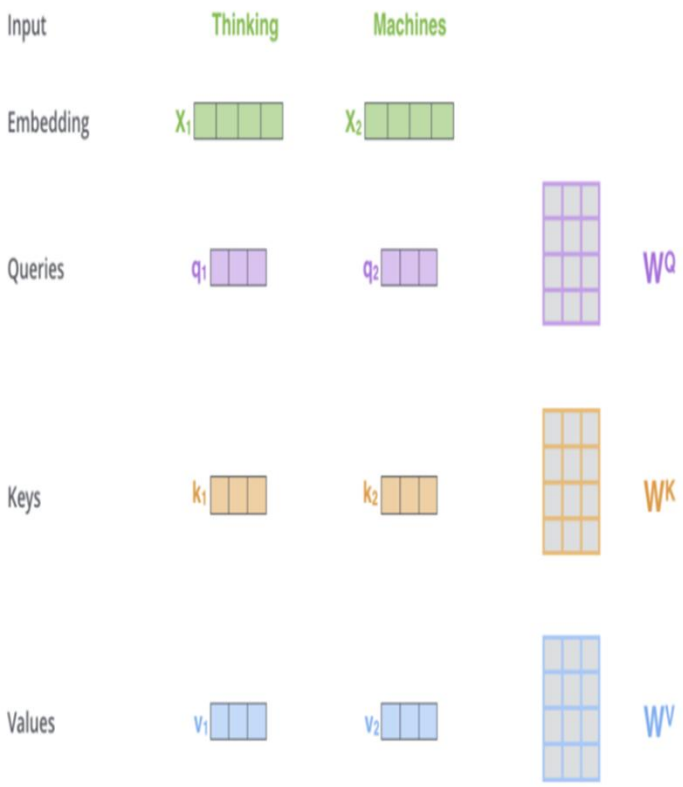
- 단어의 위치 정보를 얻기위해 각 단어의 임베딩 벡터에 위치 정보들을 더해 모델의 입력으로 사용
- 위치값을 만들기 위해서 sin함수와 cos함수를 사용

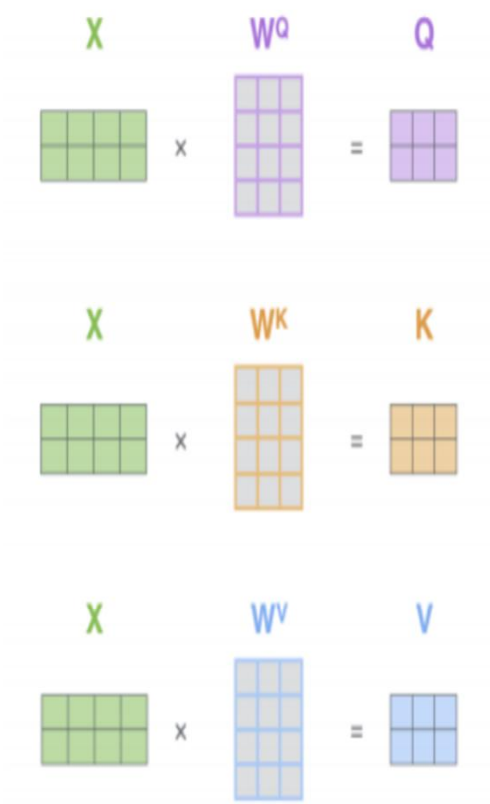
☆sin함수와 cos함수의 장점

- 값의 범위는 -1 ~ 1
- 학습데이터보다 더 긴 문장이 실제 운영중에 들어와도 Positional Encoding이 에러없이 상대적인 인코딩의 값을 줄 수 있음



03 Self Attention



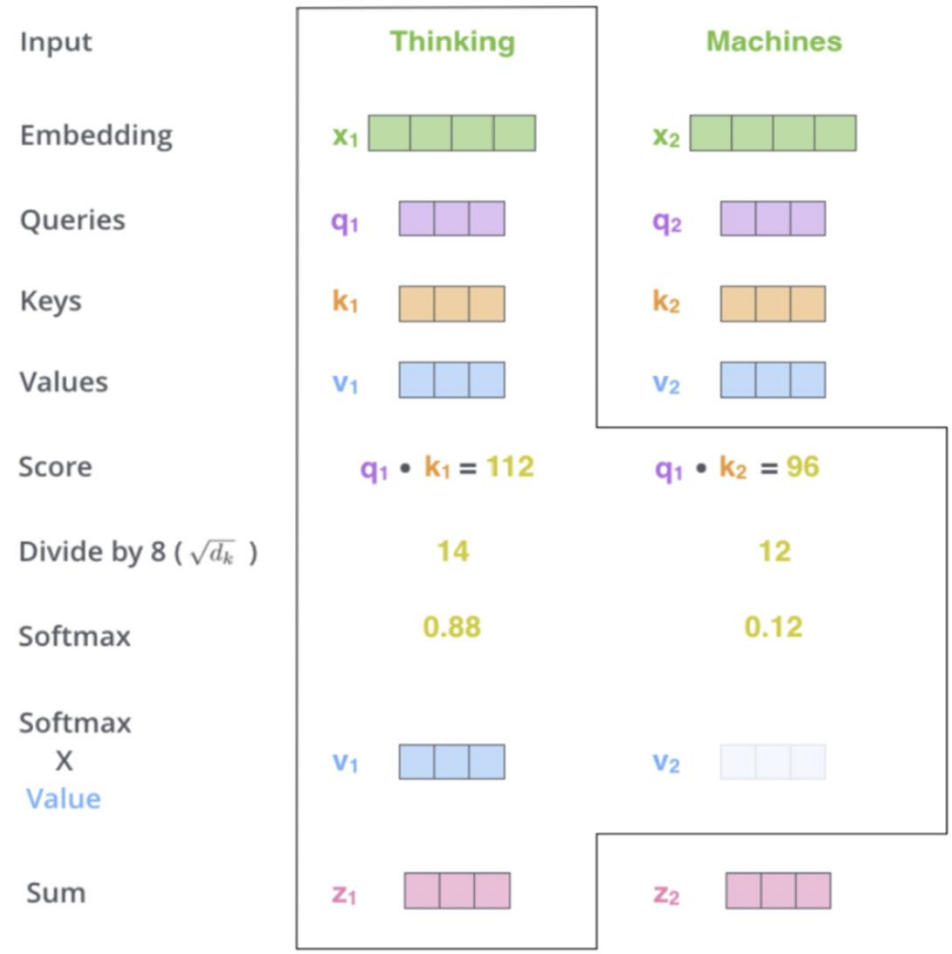


인코딩에서 이뤄지는 어텐션 연산

- Query, Key, Value는
  Wq, Wk, Wv행렬에 의해 각각 생성
- 딥러닝 모델학습 과정을 통해 최적화
- 워드 임베딩은 벡터
- 실제문장은 행렬
- 행렬*행렬이 가능하기에 한번에 Q,K,V

☆Q,K,V는 벡터 형태

03 Self Attention

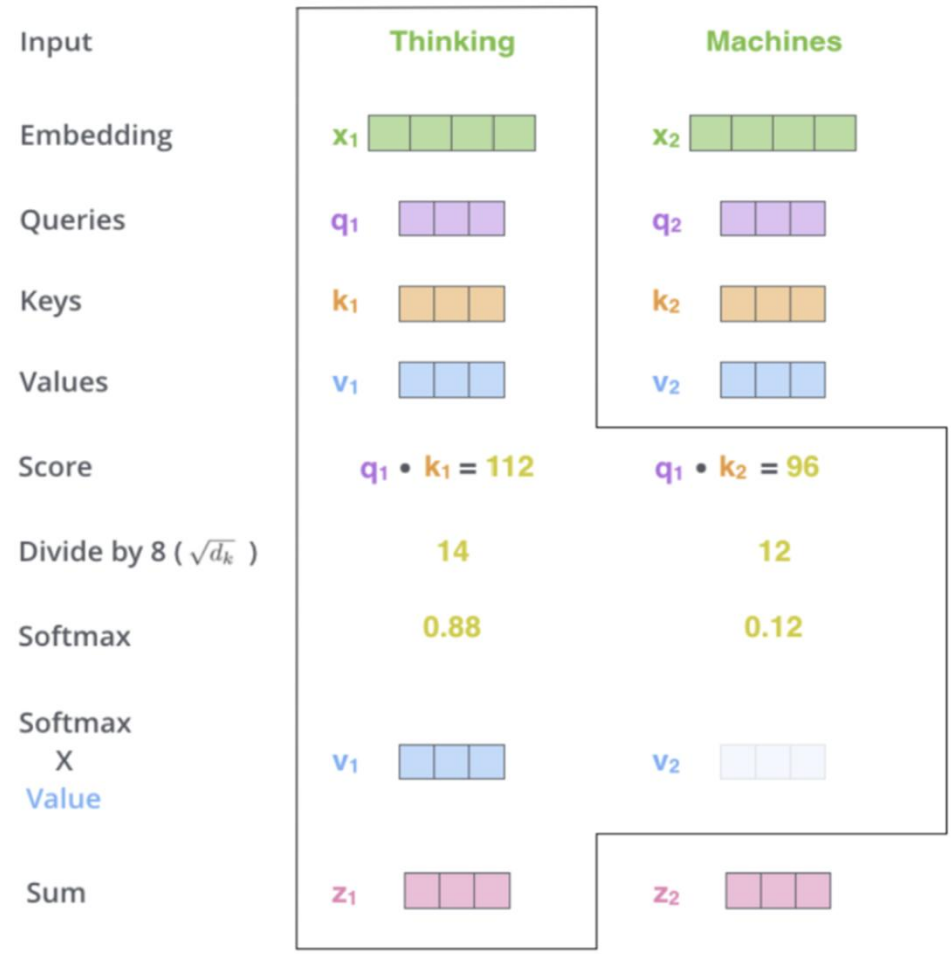


인코딩에서 이뤄지는 어텐션 연산

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

- 현재 단어가 Query일때 어떤 단어와 상관관계를 구할때 어떤 단어의 key값을 곱해줌
- Attention Score가 높을 수록 단어의 연관성이 높다.

03 Self Attention



인코딩에서 이뤄지는 어텐션 연산

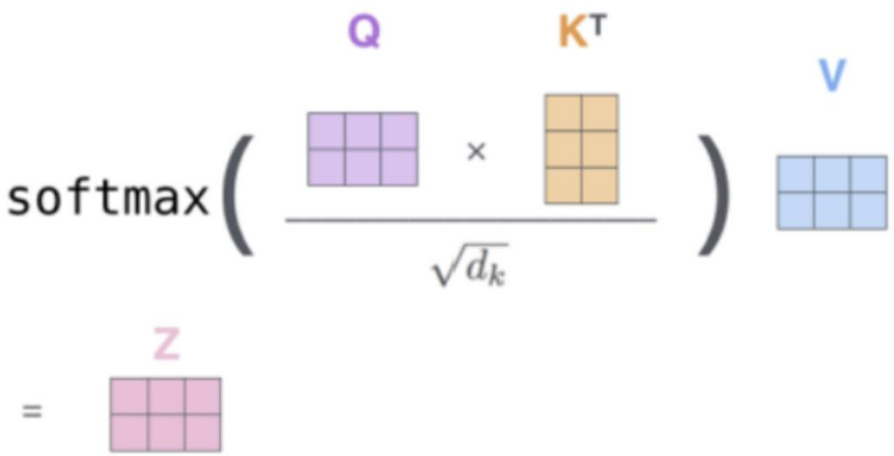
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- Attention Score를 0과 1사이의 확률 개념으로 바꾸기 위해 Softmax함수를 적용
- 함수에 적용하기전 Attention Score를 Key벡터의 차원의 루트값으로 나눠줌

☆루트로 나누는 이유

- Key벡터의 차원이 늘어날수록 벡터를 내적할때 값이 증대되는 문제를 해소

03 Self Attention



인코딩에서 이뤄지는 어텐션 연산

- Softmax의 결과값은 Key값에 해당하는 단어와 현재단어와의 연관성 지표
- 각 퍼센트지에 따른값은 각 Key의 Value에 곱해줌
- 최종벡터는 단순히 단어가 아닌 문장 속 단어가 지닌 전체적인 의미를 지닌 벡터
- 모든 단어에 대한 Attention연산은 행렬곱을 한번에 처리할 수 있다.

>Attention을 사용한 병렬처리의 가장 큰 장점.

03 Self Attention

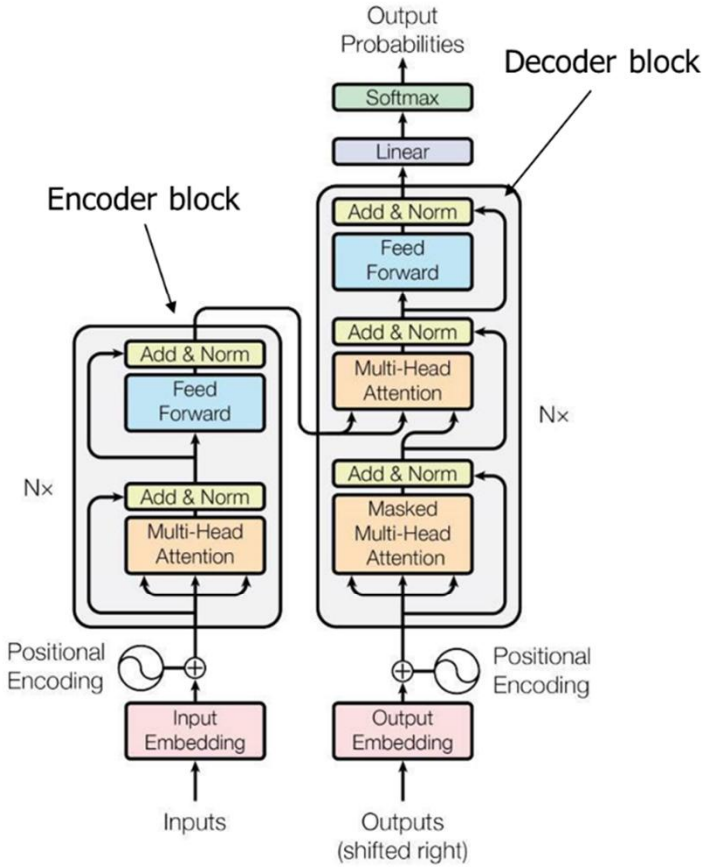
인코딩에서 이뤄지는 어텐션 연산

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

1. 각 encoder의 input vector에 대한 Query, Key, Value 벡터를 생성
2. Query 벡터와 Key 벡터를 곱함
3. Key 매트릭스 차원의 제곱근으로 나눔
4. softmax 함수를 사용해 정규화 진행
5. 가지고 있던 단어들의 Value들과 softmax값을 곱함
6. softmax에 의해서 가중합이 된 value값들을 첫번째 토큰의 최종적인 아웃풋

04 Transformer 구조

Encoder



Multi-Head Attention

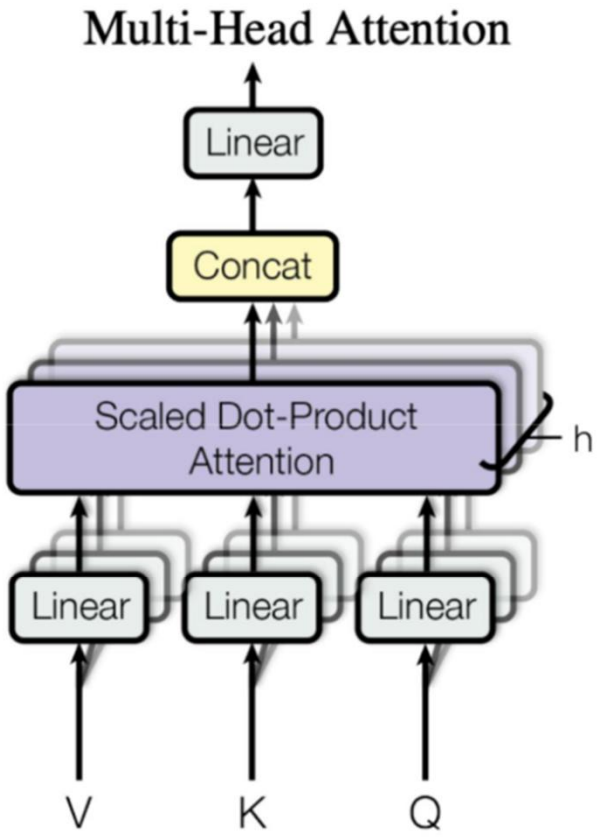
병렬처리가 된 Attention Layer

Transformer는 Attention의 여러개의 Layer를 병렬로 동시에 수행한다.

- 기계번역에 큰 도움을 준다.
- 사람의 문장을 한개의 Attention으로 인코딩하기가 어려움
- Multi-Head Attention을 사용하여 되도록 연관된 정보를 다른 관점에서 수집하여 모호한 점을 해결

04 Transformer 구조

Encoder



Multi-Head Attention

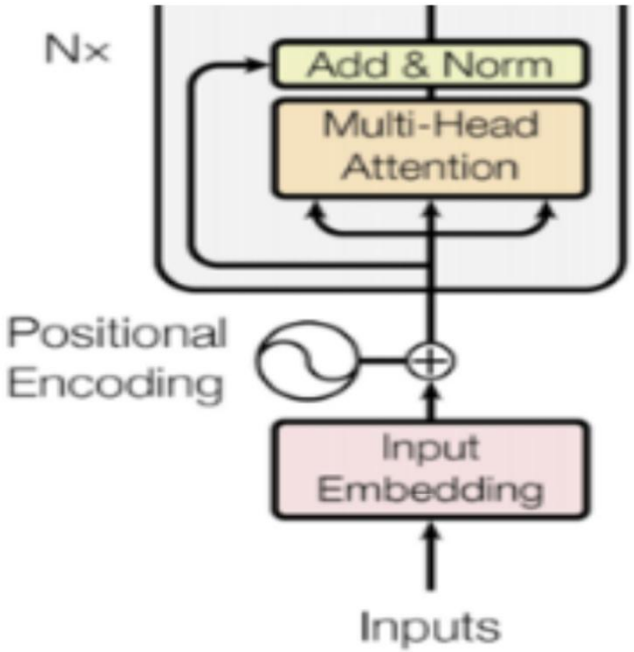
1. head 개의 Self Attention을 병렬적으로 수행
2. head 개의 Attention value 를 결합.(concat)
3. 가중치 행렬 W^O를 곱한다 → 출력 벡터 구성

   차원 조절 및 정보 통합 목적이며, 학습 과정에서

   경사하강법과 같은 최적화 알고리즘을 통해 최적의 W^O도출한다.

04 | Transformer 구조

Encoder



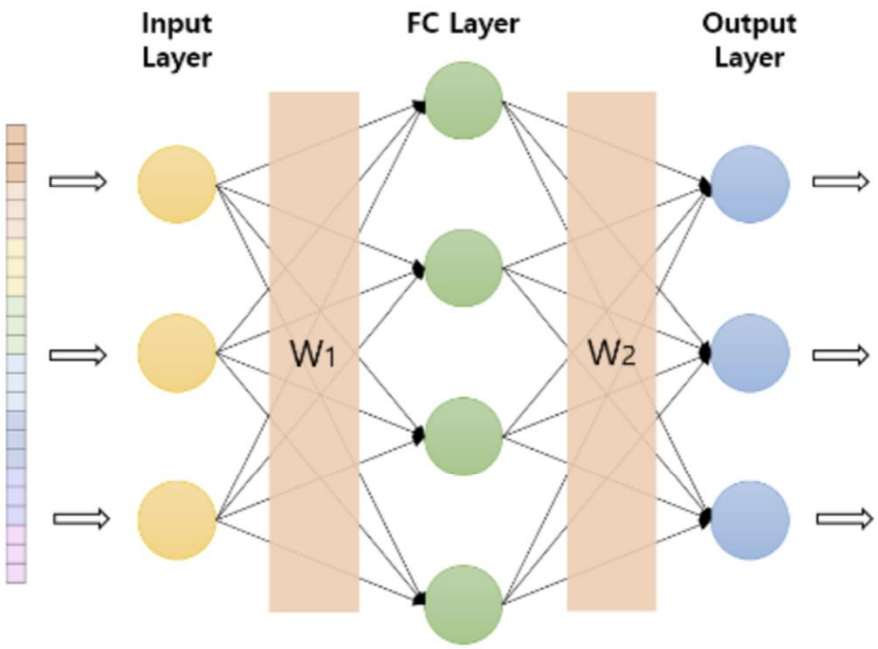
Multi-Head Attention

병렬처리가 된 Attention Layer

1. 단어를 워드 임베딩으로 전환한 후에 Positional Encoding을 적용
2. Multi-Head Attention에 입력
3. Multi-Head Attention을 통해 출력된 여러 결과 값들을 이어붙임
4. 또다른 행렬과 곱해 최초의 워드 임베딩과 동일한 차원을 갖는 벡터로 출력

04 | Transformer 구조

Encoder



각각의 벡터는 Fully Connected Layer에 들어가서 입력과 동일한 사이즈의 벡터로 또 다시 출력

딥러닝모델을 학습하다 보면 역전파에 의해 Positional Encoding이 많이 손실

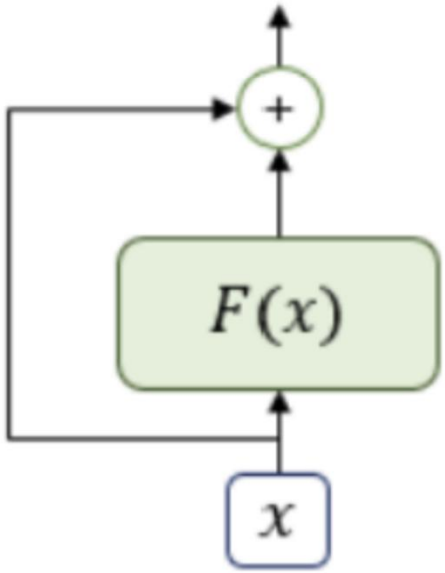
이를 보완하기 위해 Residual Connection으로 입력된 값을 다시 한번 더해줌.

★출력벡터의 차원의 크기가 입력벡터와 동일

04 Transformer 구조

Encoder

$$H(x) = x + F(x)$$



Residual Connections

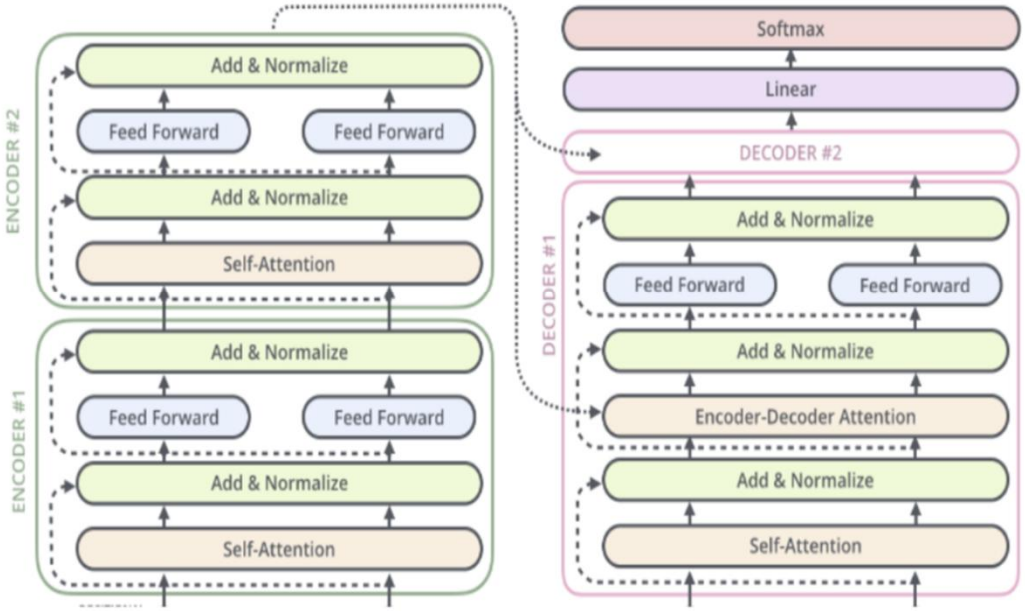
서브층의 입력과 출력을 더하는것

>>F(x)만 학습하면 되는 형태!

전체를 학습하는 것보다
학습이 오히려 쉬워짐.

04 | Transformer 구조

Encoder



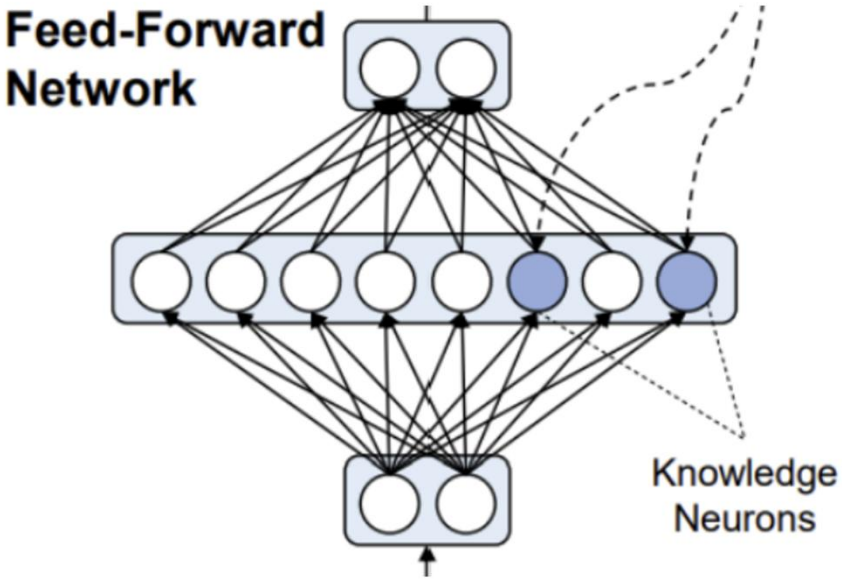
Layer Normalization

Residual Connection의 결과는
Layer Normalization을 통해서 학습의 효율을 증진

각 레이어 값이 크게 변화하는 것을 방지해
모델을 더 빠르게 학습

04 | Transformer 구조

Encoder



Feed-Forward Networks

은닉층을 통해 가중치를 학습하는 공간

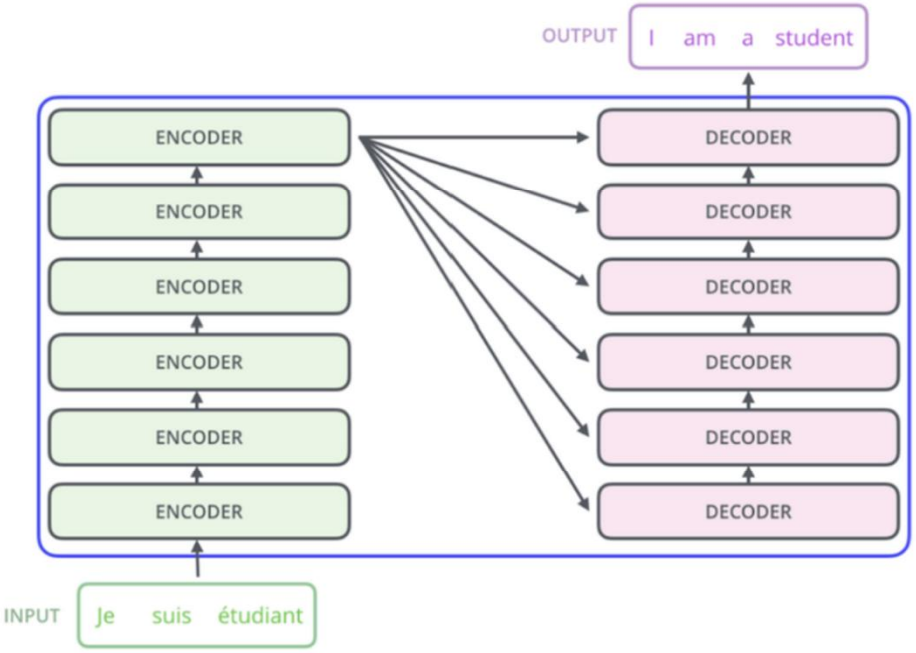
각 위치에서 독립적으로 작동하며, 다층 퍼셉트론(MLP)과 같은 구조

일반적으로 두 개의 선형 변환과 활성화 함수로 구성하여
네트워크가 더 복잡한 관계를 학습가능

주로 각 위치에서 단어의 표현을 더 깊게 변환하고
복잡한 패턴을 학습하는 데 사용

04 | Transformer 구조

Encoder



Encoder Layer

Encoder Layer의 입력벡터와 출력벡터의
차원의 크기가 같다는것은

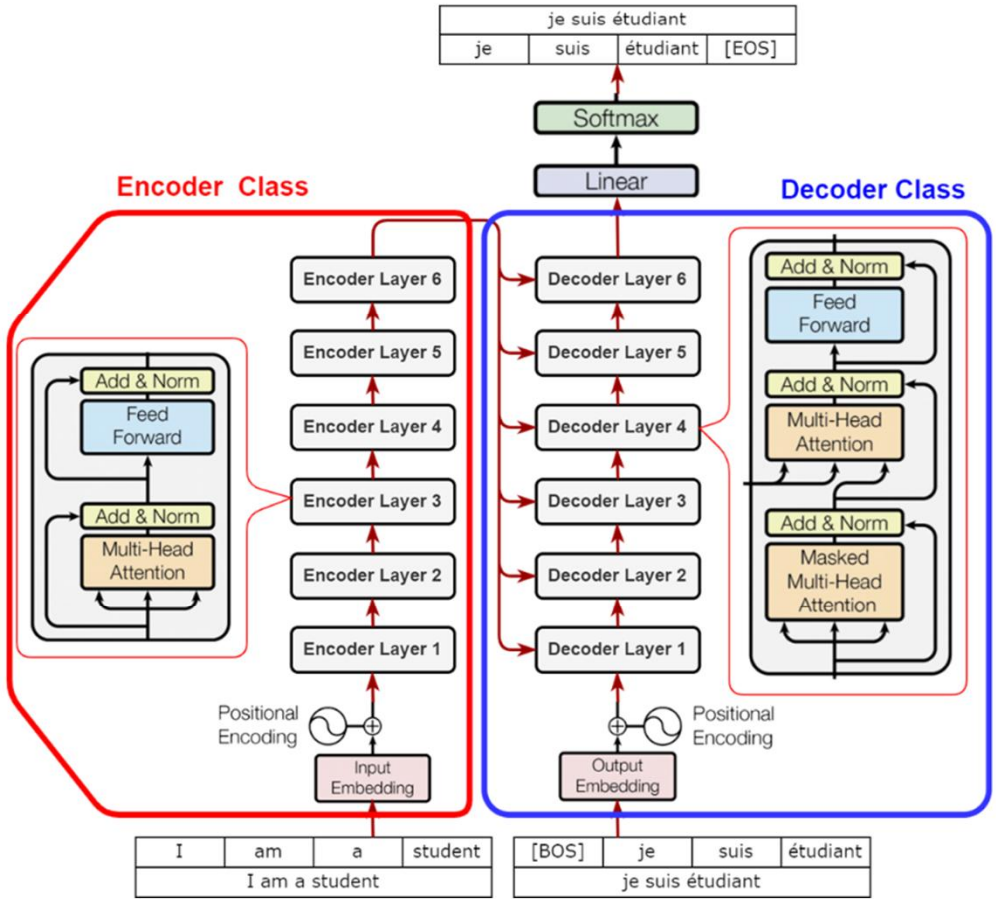
Encoder Layer를 여러개 붙여서
사용할 수 있다는것.

Transformer의 Encoder는
Encoder Layer 6개를 연속적으로 붙인 구조

각각의 Encoder Layer는 가중치를 공유하지않고 따로 학습

Encoder의 최종 출력값은 6번째 Encoder Layer의 출력값

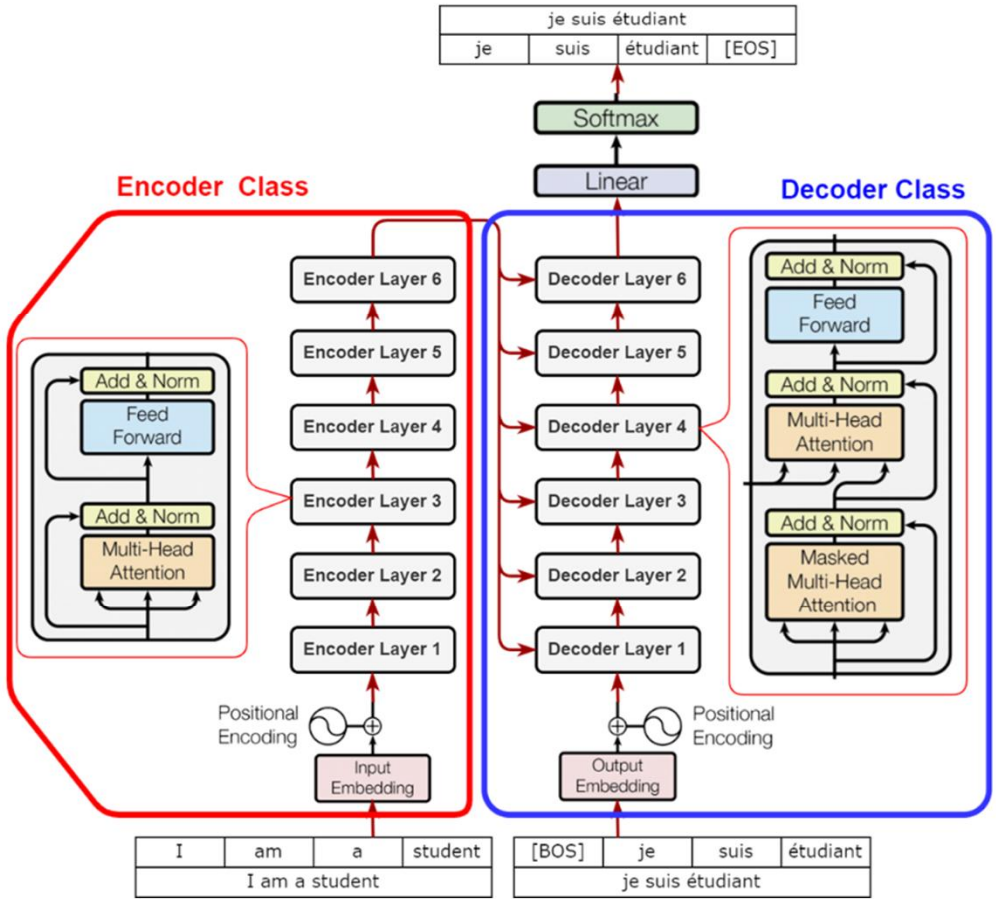
04 Transformer 구조



Decoder

- Decoder의 첫번째 Multi-Head-Attention Layer는 Masked Multi-Head-Attention Layer
- 현재 Decoder의 입력값을 Query로 사용하고 Encoder에 최종 출력값을 Key와 Value로 사용함.
- Feed Forward Layer를 통해 최종값을 벡터로 출력

04 Transformer 구조



Decoder

벡터를 단어로 출력하기위해
Linear Layer와 Softmax Layer가 존재

Linear Layer: Softmax 입력값으로 들어가 로짓을 생성
Softmax Layer: 모델이 알고있는 모든 단어들의 확률값을 출력

Label Smoothing 기술을 적용
Label Smoothing: 0과 1이 아닌 정답은 1에 가까운값

모델 학습시에 모델이 너무 학습데이터에 치중지못하도록하는 기술

감사합니다!