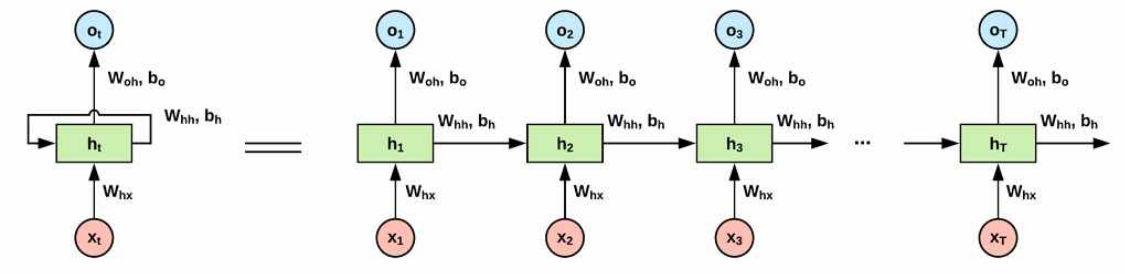
RNN과 Transformer 비교

202008010 정지원

RNN (Recurrent Neural Network)

- 연속형 데이터를 효율적으로 처리하기 위해 고안.
- 내부에 있는 순환 구조때문에 현재 정보에 이전 정보가 쌓이면서 정보 표현이 가능한 알고리즘으로, 데이터가 순환되기 때문에 정보가 끊임없이 갱신될 수 있는 구조.
- 시간에 의존적이거나 순차적인 데이터(Sequential data) 학습에 활용

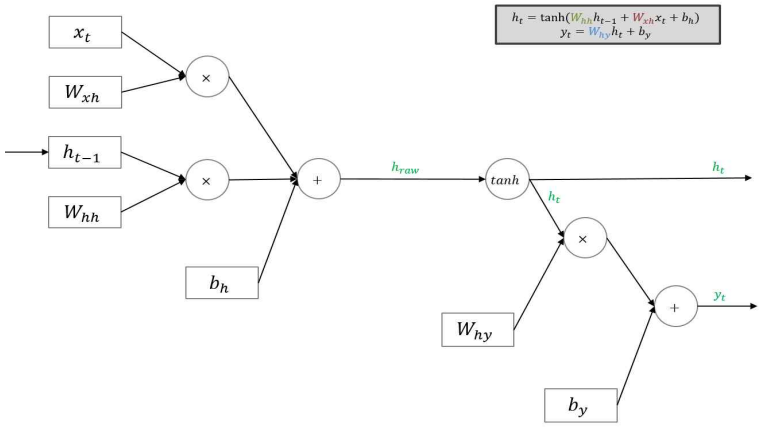


은닉 벡터가 다음 시점의 입력 벡터로 다시 전달되는 특성 때문에 순환(Recurrent) 신경망이라는 이름이 붙었다.

RNN의 단점

RNN의 구조는 벡터가 순차적으로 입력되어 연속적인 데이터를 처리할 수 있게 해주지만 이러한 특성으로 인해 GPU연산의 장점인 **병렬화**(동시에 연산 수행)가 불가능 하다.

또한, 길이가 길어짐에 따라 신경망을 하나 통과할 때마다 역전파 과정에서 활성화 함수 tanh의 미분값을 반복해서 곱해주게 되는데 이 값이 너무 높거나 낮으면 **기울기 폭발(Exploding Gradient)**, **기울기 소실(Vanishing Gradient)**과 같은 문제가 발생한다.
→ 이러한 문제점을 어느 정도 보완한 LSTM, GRU 등등이 존재한다.



Transformer

트랜스포머는 이러한 RNN의 단점들(특히 병렬화의 불가능)을 해결한 모델. 많은 시계열 분석에 위에서 언급한 LSTM을 적용할 때, 잘 작동하는 경우가 많지만, LSTM의 Encoder에서 Decoder로 들어가는 과정에서, 전달되는 정보에서 소실되는 시간 정보들이 생기게 되었다. 예를 들어, output Sequence를 얻기 위해서는 단어의 Mapping만으로 충분한 것이 아니라, Keyword와 중요한 부분들에 대한 정보가 필요하다.

→ 긴 문장이 나왔을 때 앞 쪽에 나온 단어의 영향력을 모델이 잊어버리는 현상 (Vanishing Gradient)이 발생.

==========

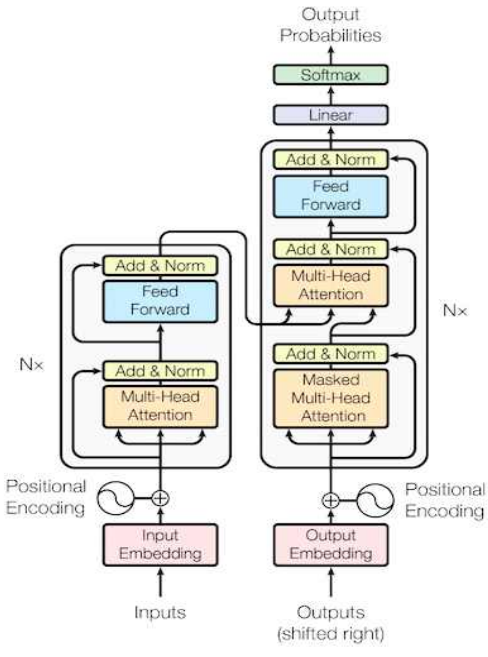
Attention

이를 위해서 나온 개념이 어탠션(Attention)이다. 쉽게 설명하면, 'Input Data의 이 부분(Time Step)이 중요해요! 집중해주세요!'라고 하는 수치들을 같이 output으로 넘겨주는 방식이다. 따라서, output sequence를 생성해내는 매 Time step마다 이전 input time step들을 다시 보고 중요하다는 부분을 참고해서 output을 생성하는 방식.

==========

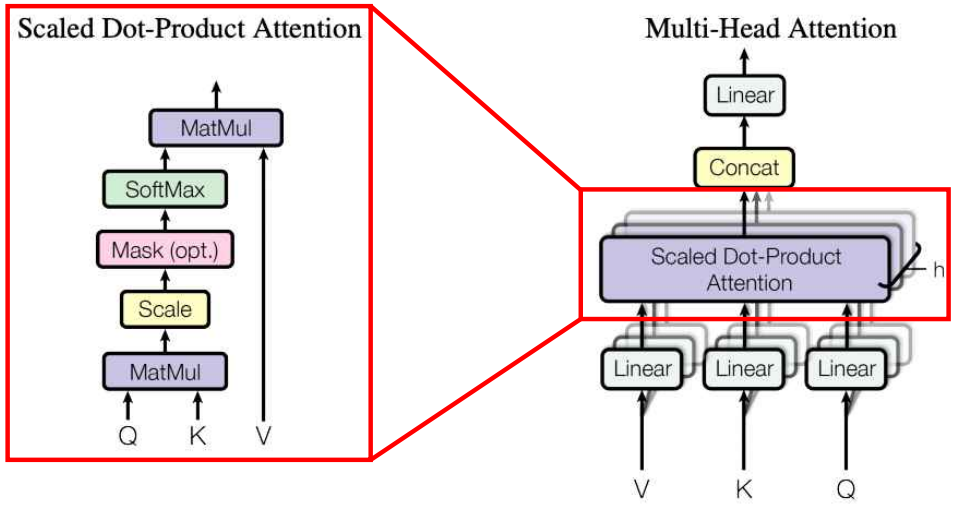
RNN, LSTM의 약점으로 많이 언급되었던 것은, 단어를 순차적으로 입력받아서 병렬처리가 어렵다는 점. 하지만 이 순차적으로 입력받는 것이, 각 input의 위치정보를 반영할 수 있게 해주었다.

Transformer는 순차적으로 Data를 넣는 것이 아니라, Sequence를 한번에 넣음으로써 병렬처리가 가능하면서도, Attention 등의 구조를 통해 어떤 부분이 중요한지를 전달하여, **위치정보를 반영할 수 있게 되었다**.

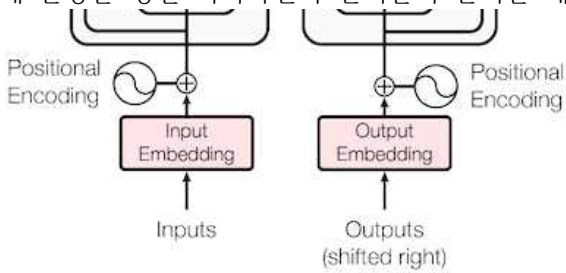


이 구조가 바로 트랜스포머 구조. 기존 구조에서는 문장을 “단어의 연속적인 배열”로 간주했다.(RNN) 그러나 트랜스포머는 문장을 “단어간의 어탠션들의 합”으로 나타냈다. 즉, 문장이라고 하는 구조는 마치 베틀처럼 **어탠션을 촘촘하게 엮은 형태**로 나타낼 수 있는 것이다.

트랜스포머에서 어탠션은 **Scaled-dot Product Attention**을 사용함. 그리고 이를 여러번 반복하는 **Multi-head Attention**을 고안해 냈다.



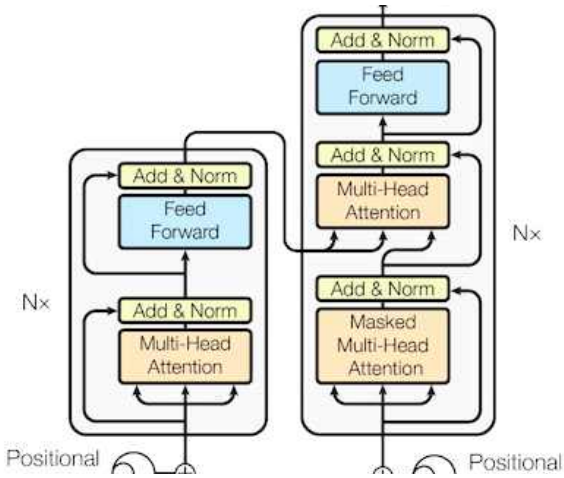
그럼 트랜스포머는 어떻게 문장을 병렬 처리하면서 단어들의 순서를 재배열하는 것인가.



먼저 임배딩(Embedding)이라는 단계가 있다. 임배딩은 다른 알고리즘에서도 많이 쓰는 방식으로, data를 임의의 N-dimension data로 만들어주는 단계다.

본격적인 트랜스포머는 Positional Encoding으로 시작한다. 순서를 알아내기 위해서는 데이터의 위치정보를 전달해주는 방법이 필요한데, Positional Encoding이 이 역할을 담당한다.

Positional Encoding은 'Sequence 내에서 해당 정보의 위치 정보'와 '임배딩된 데이터'를 사인함수와 코사인함수 형태로 만들어서 다음 Layer의 Input으로 전달하게 된다. 쉽게 설명하면, 임배딩을 할 때 위치정보까지 함께 넣어주자라는 내용이다. 이런 방식으로, 트랜스포머는 해당 Input의 위치를 반영한 pre-processing을 할 수 있게 된다.



이렇게 정리된 Input Data들은 본격적으로 Encoder로 들어간다. Encoder는 Multi-head Self Attention / Add & Normalize / Position-wise FFNN라는 모듈들로 구성되어 있다.

Encoder는 문장을 잠재 표현으로 변환하는 부분이다. 논문에서 저자들은 6개의 동일한 구조를 활용하여 이를 구현했다. 각 구조는 위에서 설명한 Multi-head attention 층과 Fully-connected 층으로 구성했다. 논문에서는 각 구조들의 출력값을 합하여 한 단어당 총 512차원의 잠재 표현으로 변환했다.

Decoder도 Encoder와 유사하게 형성되었다. 단, Encoder의 출력이 잠재 표현인 것과 다르게, Decoder의 출력은 단어의 출현 확률이 된다. 아울러 Decoder는 실제 단어를 출력하는 부분이기 때문에 해당 단어를 기준으로 뒷 단어들에 의한 Attention을 Masking 처리를 하고 참조하지 않았다.

Transformer의 단점

모델이 훈련하기에 너무 크다. 때문에 어느정도 이상의 성능을 내기 위해서는 보다 막대한 양의 훈련 데이터와 시간이 필요하다. 훈련도 훈련이지만 실행 시에도 시간이 꽤 걸린다. 실제로 상용화(제품화) 되기에는 모델이 많이 느리다.

실제로 Colab 실습 시에도 RNN은 수 초 ~ 수 분 안에 끝났지만 Transformer는 학습하는데 수십 분이 걸렸다...

정리

RNN의 장단점

- 장점: 순환 구조때문에 현재 정보에 이전 정보가 쌓이면서 정보가 끊임없이 갱신 된다.
- 단점: 벡터가 순차적으로 입력되어 GPU연산의 장점인 병렬화가 불가능

Transformer의 장단점

- 장점: RNN이 가진 병렬화의 불가능을 해결.
- 단점: 모델을 훈련하거나 실행시키기가 너무 무거움.

	RNN	Transformer
데이터 처리 방법	순차 처리	병렬 처리
Attention	없음	Scaled-dot Product Attention 사용
성능	가벼움	무거움

참고한 자료

- https://www.goldenplanet.co.kr/our_contents/blog?number=857
- https://velog.io/@rsj9987/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EC%9A%A9%EC%96%B4%EC%A0%95%EB%A6%AC
- https://engineering-ladder.tistory.com/73
- https://tech.scatterlab.co.kr/transformer-review/
- https://www.youtube.com/watch?v=EFkbT-1VGTQ&ab_channel=MachineLearningTV