NTM에서 Seq2seq 모델이 많이 사용되는데 이는 한 문장(시퀀스)을 다른 문장(시퀀스)으로 변환하는 모델이다.

**RNN 모델**

**입력 문장이 들어오면 Encoder에서 RNN을 이용하여 고정된 크기의 context vector(문맥벡터)로 정보를 압축한다. 이후 context vector로부터 Decoder가 번역 결과를 추론한다.**

* Encoder와 Decoder의 아키텍처의 내부는 RNN 아키텍처로 구성되어 있다. 입력 문장을 받는 RNN 셀을 Encoder라고 하고 출력 문장을 출력하는 RNN 셀을 Decoder라고 한다.
* Encoder에서는 입력 문장이 단어 토큰화를 통해 단어 단위로 쪼개지고 단어 토근 각각은 RNN 셀의 각 시점에 순차적으로 입력이 된다. Encoder는 모든 단어를 입력받은 뒤에 마지막 시점의 은닉 상태를 Decoder로 넘겨주는데 이를 컨텍스트 벡터라고 한다.
* 컨텍스트 벡터는 입력된 모든 단어 정보들을 압축해서 하나 벡터로 만든다.
* 컨텍스트 벡터는 디코더 RNN 셀의 첫번째 은닉 상태로 사용된다. 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력한다.

한계점

Encoder의 마지막 hidden state만을 context vector로 사용해 **context vector가 고정된 크기를 가지기 때문에 많은 정보가 입력되면 정보 손실이 발생하고 성능이 저하된다는 한계점**이 있다. (고정된 크기에 입력 문장의 모든 정보를 압축하려고 하니까 정보 손실이 발생하는 것) 긴 문장이 들어오면 번역이 잘 안되는 문제점이 있으며 학습데이터에 존재하지 않는 단어가 나오면 생략해버리거나, 앞부분에 대해 반복 번역을 진행할 때가 있다.

이 문제의 대안으로 입력 시퀀스가 길어지면 출력 시퀀스의 정확도가 떨어지는 것을 보정해주는 어텐션(Attention) 기법이 등장했다.

**Attention 모델**

**디코더에서 출력 단어를 예측하는 매 시점(time step) 마다 인코더의 입력 문장의 hidden states를 참고한다. 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분에 가중치를 높여 참고한다.**

Attention에서는 Encoder의 매 스텝에 생성되는 벡터가 Attention에 사용되므로 Encoder의 결과를 고정길이 벡터에 담아야 하는 문제가 해소됐다. 따라서 Sequence 길이에 비례하여 더 많은 정보가 활용되어 장문 번역 성능이 높아졌다.

(두 가지 언어를 구사할 수 있는 사람이 번역할 때 단어 별로 원문과 번역문을 대조해가며 번역하는 것과 유사하다고 생각하면 이해가 쉬움)

\*\*참고: Attention 모델 중 Dot-Product Attention 사용시 예측 방법

RNN seq2seq의 구조를 살펴보면, Encoder에서 계산한 여러 Hidden State 중 마지막 Hidden State만을 Decoder에서 사용하게 된다. 즉, Encoder의 마지막 Hidden State를 제외하고는 사용되지 않는다. 어텐션 매커니즘은 바로 이 사용되지 않은 Hidden State를 이용한 아이디어이다.

​*Attention Mechanism (Dot-Product Attention)*

*1. Attention Score*

-> 디코더에서 출력 결과를 예측할 때 어느 인코더의 Hidden State를 얼마나 참고할지를 결정해야 한다. 여기서 현재 예측에 필요한 정도라고 판단되는 점수를 어텐션 스코어 (Attention Score)라고 한다. 어텐션 스코어를 구하기 위해, 현 시점의 디코더의 Hidden State (st)와 인코더의 모든 Hidden State들에 각각 내적을 수행한다.

*2. Attention Distribution*

-> 각 인코더와 디코더의 현재 hidden state를 내적한 값은 스칼라로 나와 여기에 소프트맥스 함수를 적용해서 정규화해 어텐션 분포를 구한다. 이 어텐션 분포는 각 인코더 hidden state의 중요도라고 볼 수 있다.

*3. Attention Distribution X Encoder Hidden State*

-> 소프트맥스를 통해 얻은 어텐션 분포를 각 인코더 Hidden State와 곱해준다.

*4. Weight Sum*

-> 각 어텐션 분포와의 곱을 통해 얻어진 Hidden State들을 전부 더해준다.

(element-wise) 이렇게 얻은 벡터를 인코더의 문맥을 포함하고 있다하여 컨텍스트 벡터(Context Vector)라고도 부른다.  
5. 컨텍스트 벡터와 현 시점의 디코더 셀의 Hidden State와 연결해준다.

(concatenate 방법, 평균 내서 사용하는 방법)

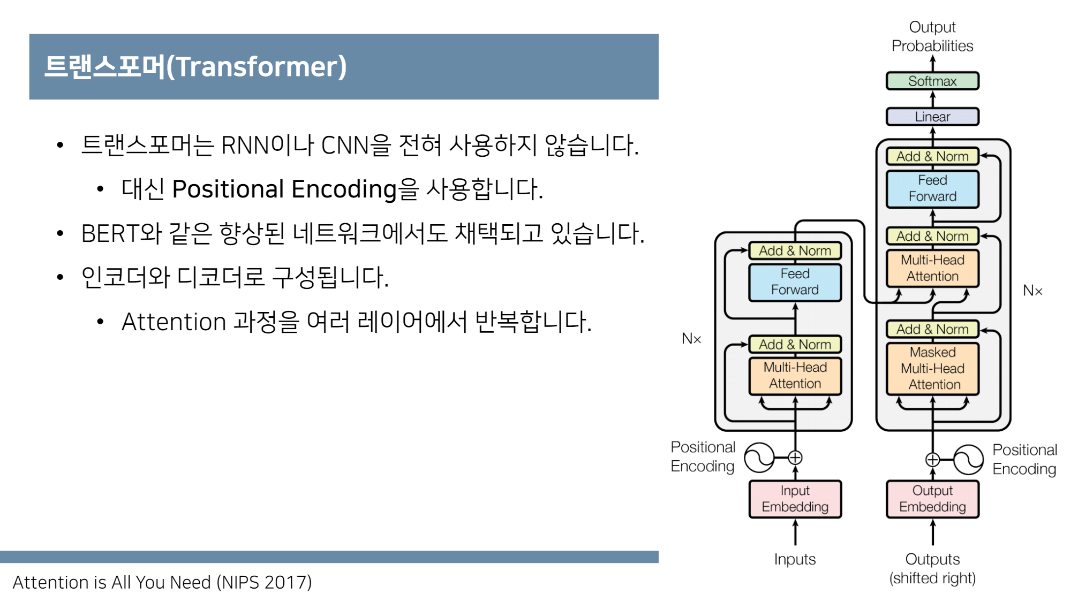
6. 구한 벡터를 이용해서 최종 예측 값을 구하게 된다.

**Transformer 모델**

**기존의 seq2seq의 구조인 Encoder-Decoder를 따르면서도, RNN, CNN을 사용하지 않고 Attention만으로 구현한 모델이다**

**Attention Net이** 기존에는 Decoder의 Sequence와 Encoder의 Sequence 간 Align을 맞춰주는 용도로 사용되었다면, **여기서는 추가로 Encoder/Decoder 각 Layer의 입력 정보를 함축하는 데 사용**되는 방식으로 확장되었다.

밑은 Transformer 모델의 전체적인 과정이다. **현재 가장 많이 사용되는 모델**이기에 논문 구현 코드를 분석했다.

****

Q(uerky): 현재 시점의 token을 의미

K(ey): attention을 구하고자 하는 대상 token을 의미

V(alue): attention을 구하고자 하는 대상 token을 의미

**1. self-attention**

Self-attention층에서 입력 내의 모든 다른 단어들과의 관계를 살펴본다.

우선, Query 벡터, Key 벡터, 그리고 Value 벡터를 파라미터로 받는다.

그 다음 Query 벡터와 Key 벡터의 행렬곱을 한 후, gradient vanishing 발생을 방지하기 위해 이를 key의 제곱근으로 나눠준다.

이 값으로부터 softmax 확률을 계산한 후 이 확률값과 value 벡터를 가중합해서 Self-attention 계산을 마무리한다.

**2. multi-head attention**

Multi-headed attention은 Self-Attention을 8 번 진행하여 각각 다르게 표현된 Attention Head를 얻게 되는 것이다. 8 개의 행렬을 하나의 행렬로 합쳐 feed-forward layer으로 보낸다.

**3. encoder**

Residual connection: 성능 향상을 위한 잔여학습

\*\*참고

 Residual Connection이라는 것은 정말 단순하다. y=f(x)y=f(x)를 y=f(x)+xy=f(x)+x로 변경하는 것이다. 즉, output을 그대로 사용하지 않고, output에 input을 추가적으로 더한 값을 사용하게 된다. 이로 인해 얻을 수 있는 이점은 명확하다. Back Propagation 도중 발생할 수 있는 Gradient Vanishing을 방지할 수 있다

encoder layer는 크게 multi-head attention layer와 feed forward layer로 구성된다.

입력이 들어오면 multi-head attention layer를 거친 출력이 feed-forward layer 과정을 거쳐 가중치영향을 받은 다음 Encoder로 전달된다.

**4. decoder**

Decode는 encoder와 거의 유사하다. Encoder의 output을 입력으로 받는데 여기서 attention 방식의 차이를 보인다. “masked\_multi\_head\_attention” layer는 output sequence내에서 현재 위치의 이전 위치들에 대해서만 Attention을 진행할 수 있다. 또한 “Encoder-Decoder Attention” layer은 Query행렬들을 그 밑의 layer에서 가져오고 Key와 Value행렬들을 encoder의 출력에서 가져온다는 점이다.

**5. positional encoding**

Transformer은 RNN을 사용하지 않기에 각 단어의 위치와 시퀀스 내의 다른 단어 간의 위치 차이에 대한 정보를 포함하고 있는 positional encoding을 사용해 정규화를 해야 한다.

이 때, 주기함수를 활용해 각 단어의 상대적인 위치 정보를 네트워크에게 입력한다.

\*\*참고

Positional Encoding의 목적은 positional정보(대표적으로 token의 순서, 즉 index number)를 정규화시키기 위한 것이다. 단순하게 index number를 positionalEncoding으로 사용하게 될 경우, 만약 training data에서는 최대 문장의 길이가 30이었는데 test data에서 길이 50인 문장이 나오게 된다면 30~49의 index는 model이 학습한 적이 없는 정보가 된다. 이는 제대로 된 성능을 기대하기 어려우므로, positonal 정보를 일정한 범위 안의 실수로 제약해두는 것이다

**6. transformer**

최종적인 전체 트랜스포머(Transformer) 모델을 정의한다.

입력이 들어왔을 때 앞서 정의한 인코더와 디코더를 거쳐 출력 문장을 생선한다.

<참고 문헌>

1. RNN 모델

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

2. Attention 모델

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

3. Transformer 모델

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.