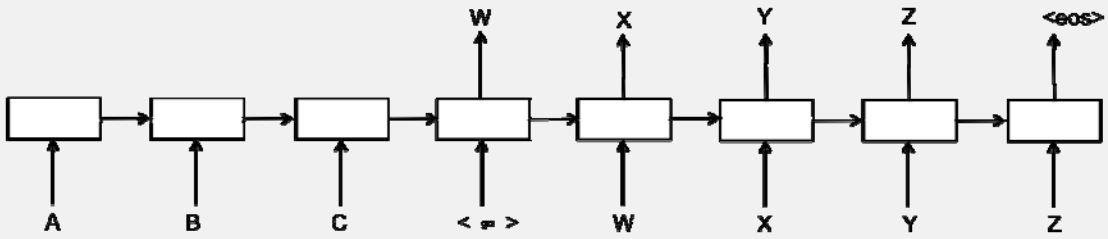
Transformer

Attention is all you need

Outline

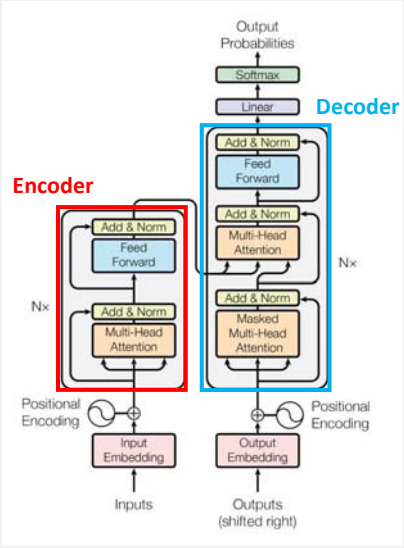
- Limitations of Attention based Seq2Seq
- Transformer
  - Positional Encoding
  - (Masked) Multi-Head Attention
  - Scaled Dot-product Attention
  - Residual Connection

Limitations of Attention based Seq2Seq



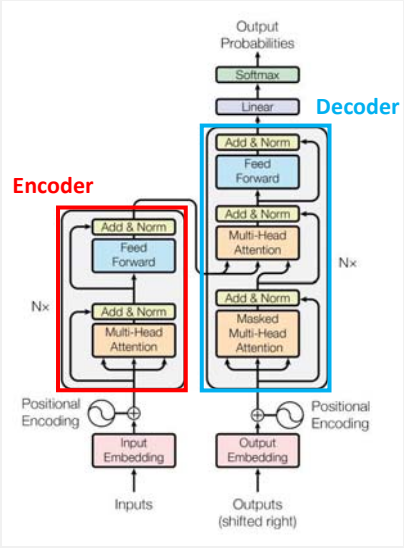
- Seq2Seq 의 두 가지 단점
  1. **병렬처리가 불가능 -> 속도 느림**
  2. Long Term Dependency 문제를 완전히 해결 할 수 없음
- Attention 이 Long Term Dependency 문제를 어느정도 완화시켰으나, 결국 RNN 기반이기에 병렬 처리 불가

Transformer



- Positional Encoding
- (Masked) Multi-Head Attention
- Scaled Dot-Product Attention

Transformer



- Encoder Input

  = Source Sentence

  ex) ich bin fröhlich
- Decoder Input

  = right shifted Target Sentence

  ex) (((((<S>) I) am) happy) <E>) …
- Decoder Output

  = Target Sentence

  ex) I am happy <E> …

Position Encoding

- Position Encoding
  - CNN, RNN은 time series 즉 순서를 알 수 있음
  - Transformer는 순서를 알지 못함

Position Encoding

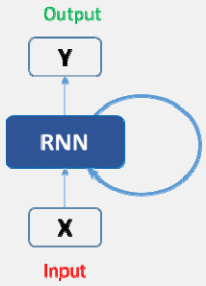
- CNN

Hello my name is Michael nice to meet you

  - N-gram 과 유사함
  - (Hello, my), (my, name) … (meet, you) 이렇게 묶여서 들어가기에 근체에 나타나는 단어를 알 수 있고 이는 순서의 의미를 담음

- RNN

  - Time step t 입력과 출력으로 sequence data에 적합
  - 이전 time step의 출력이 현재 time step 출력에 영향

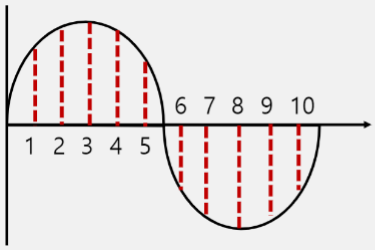


Position Encoding – Transformer

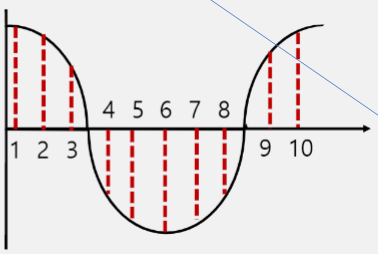
- 문장의 순서를 알기 위해 **Positional Encoding (sinusoidal encoding)** 을 한다

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

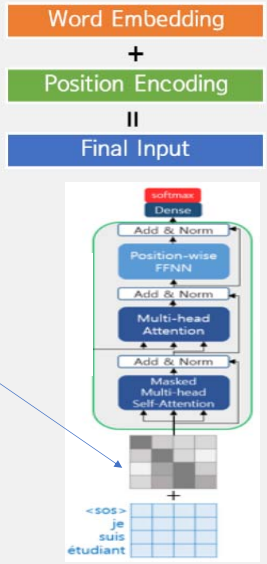
pos : token (word embedding) 위치
i : token vector의 차원 index
d_{model} : token vector의 차원 크기



$\sin x$



$\cos x$

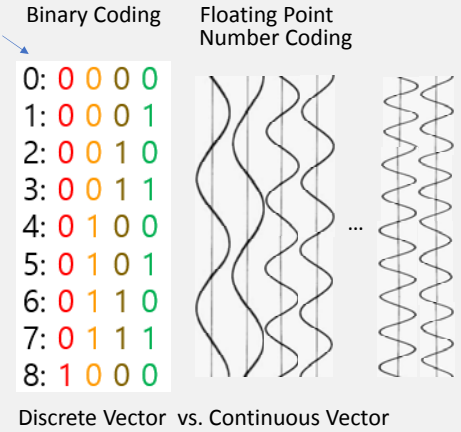


Positional Encoding

- 특징

이진 비트로 표현하였을 때 가장 작은 bit 는 1이 증가 할 때마다 바뀌고 두 번째로 작은 bit 는 2가 증가할 때마다 바뀐다. 세 번째로 작은 bit 는 4가 증가할 때마다 바뀌고 즉, n번째로 작은 bit 는 2^{n-1} 이 증가할 때마다 바뀐다. 즉, 이러한 bit 의 변화 규칙을 통해 위치 벡터를 표현할 수 있다.

이진 비트 벡터 방식은 표현할 수 있는 범위가 제한적이고 이산적이므로, 위치를 표현할 수 있는 범위가 넓고 연속적인 값을 가지는 부동 소수점 벡터 방식으로 표현한다.



Discrete Vector vs. Continuous Vector

(참고) Positional Encoding

- 특징

$$M.\begin{bmatrix}\sin(\omega_k.t)\\ \cos(\omega_k.t)\end{bmatrix}=\begin{bmatrix}\sin(\omega_k.(t+\phi))\\ \cos(\omega_k.(t+\phi))\end{bmatrix}$$

$$\begin{bmatrix}a_1 & a_2\\ a_3 & a_4\end{bmatrix}.\begin{bmatrix}\sin(w_k\cdot t)\\ \cos(w_k\cdot t)\end{bmatrix}=\begin{bmatrix}a_1\sin(w_k\cdot t)+a_2\cos(w_k\cdot t)\\ a_3\sin(w_k\cdot t)+a_4\cos(w_k\cdot t)\end{bmatrix}=\begin{bmatrix}\sin(w_k\cdot(t+\phi))\\ \cos(w_k\cdot(t+\phi))\end{bmatrix}$$

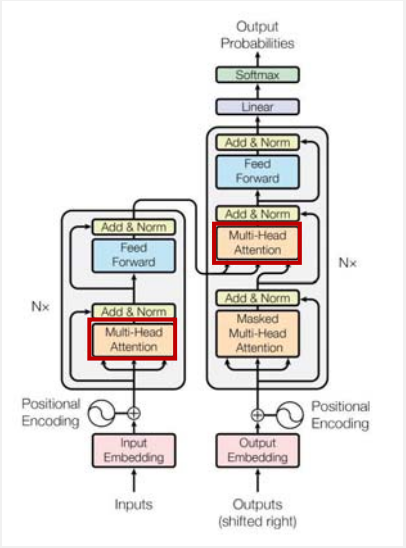
삼각함수 덧셈 정리에 의해

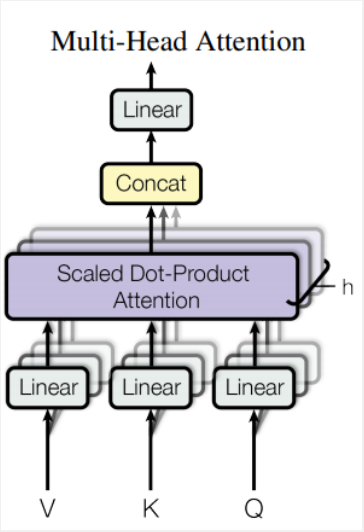
$$\sin(a+b)=(\sin a)(\cos b)+(\sin b)(\cos a)$$
$$\cos(a+b)=(\cos a)(\cos b)-(\sin a)(\sin b)$$

$$\therefore M=\begin{bmatrix}a_1 & a_2\\ a_3 & a_4\end{bmatrix}=\begin{bmatrix}\cos(w_k\cdot\phi) & \sin(w_k\cdot\phi)\\ -\sin(w_k\cdot\phi) & \cos(w_k\cdot\phi)\end{bmatrix}$$

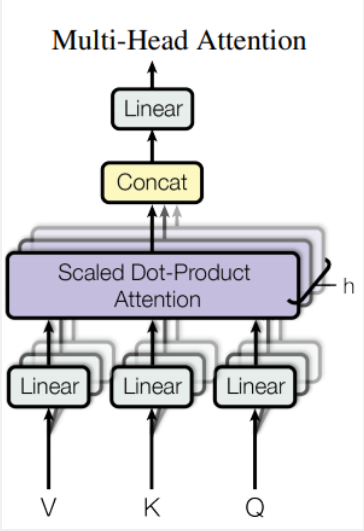
위치변환을 linear transformation으로 가능하게 함으로써, attention head 별로 상대적 위치(∵ residual connection 을 사용하므로...)에 따른 의미 연관성을 파악할 수 있게 함

Multi-Head Attention



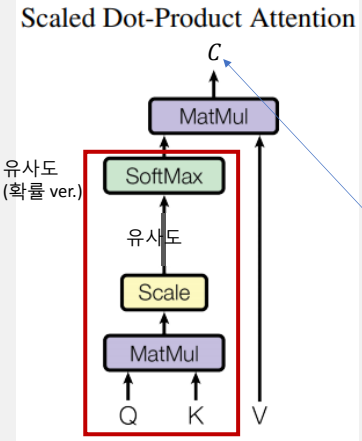


Multi-Head Attention



- Query, Key, Value에 대해 Scaled Dot-Product Attention 여러 번(Multi) 시행
  - 각 Head에서 다른 관점에서 attention을 하기 때문 기존 하나의 관점에서 attention 할 때보다 더 많은 정보 습득
  - 여러 개로 나누어 병렬처리 하므로 계산적인 측면에서도 이득
- 최종적으로 모든 관점에서의 attention 정보를 이어 붙여서(concat) 내보냄

Scaled Dot-Product Attention



- Query(Q), Key(K), Value(V) Attention

-> Query 와 Key의 유사도 만큼 attention weight를 줘서 Value를 가중합

유사도(확률 ver.) 유사도

$$c_i = \sum_{j=1}^{n} \alpha_{ij} V_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{N} \exp(e_{ik})} \qquad e_{ij} = similarity\_score(Q_i, K_j)$$

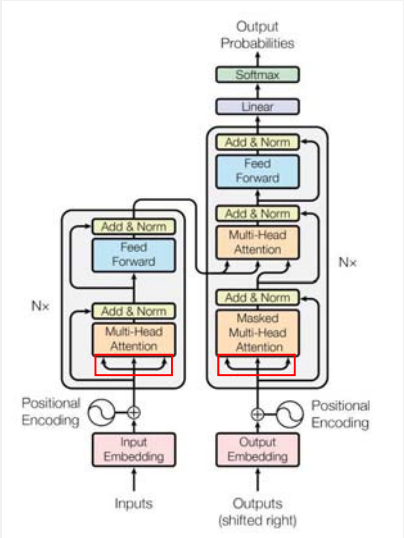
Scaled Dot-Product attention 함수 사용

$$similarity\_score(s_{i-1}, h_j) = \frac{s_{i-1}^T h_j}{\sqrt{n}}$$

why? 그냥 Dot-Product만 하면 곱할 때마다 값이 점점 커짐

따라서, 평균벡터 <1,1,...,1> 크기 만큼 scaling을 적용

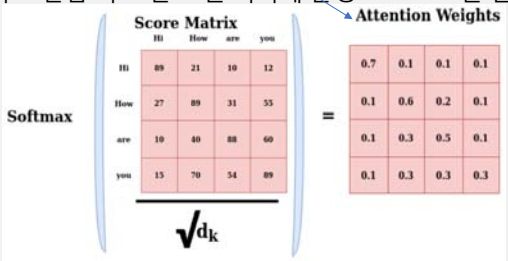
Self Attention



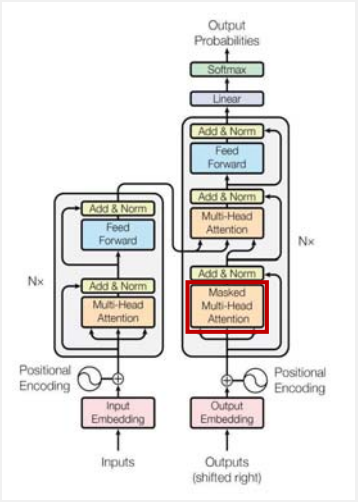
- Query, Key, Value 가 모두 동일
  - -> Q = K = V = T (Token Embedding Vector)

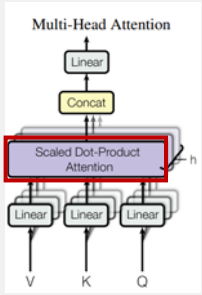
$$c_i = \sum_{j=1}^{n} \alpha_{ij} T_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{Tx} \exp(e_{ik})} \qquad e_{ij} = similarity\_score(T_i, T_j)$$

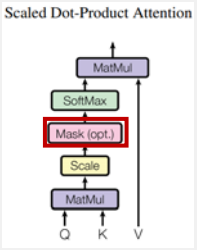
- 즉, 자기 자신(T)의 요소들끼리 attention 하는 것
  - 문장에서 각 토큰들 간의 유사도 측정
  - 즉, 유사도 만큼 각 토큰 표현 벡터에 반영 -> context를 반영한 벡터



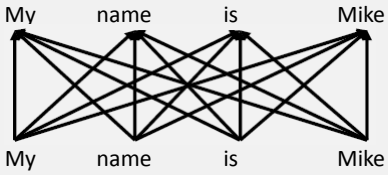
Masked Multi-Head Attention



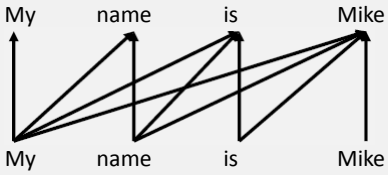




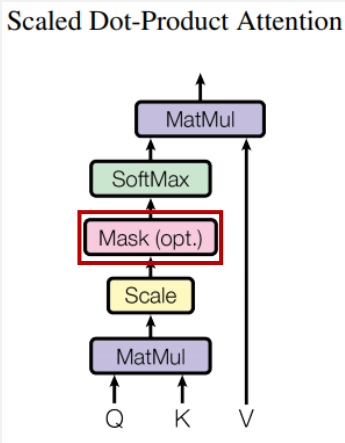
- 일반 Self Attention



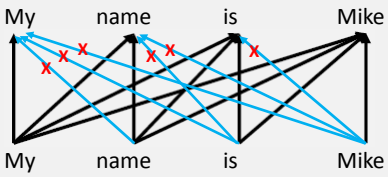
- Masked Self Attention

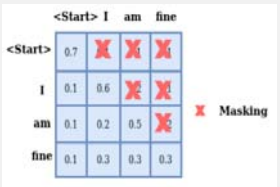


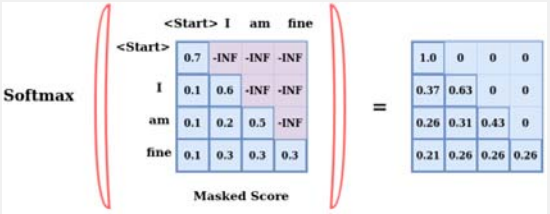
Masked Multi-Head Attention



- Why?
  - Decoder 부분은 아직 예측하지 않은 부분을 attention 하지 못하기 때문에 앞의 단어들에 대해서만 attention 하기 위해 Masked Self Attention 기법을 사용

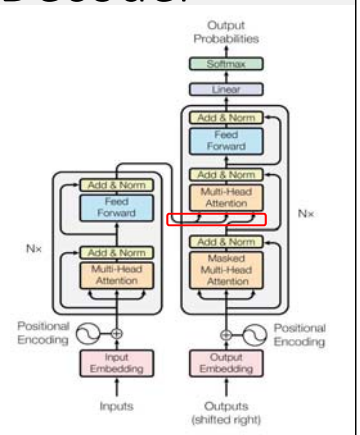






Attention between Encoder and Decoder

- Transformer



source Hello my **name** is **Michael**, nice to meet you

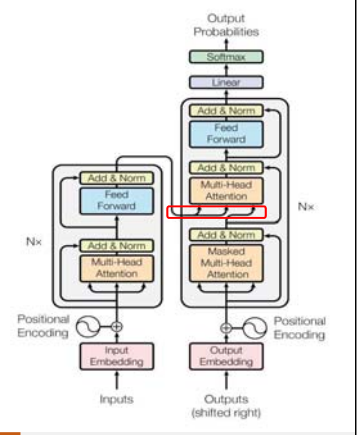
attention 0.01 0.04 0.1 0.01 0.8 0.01 0.01 0.01 0.01

translate

target 안녕 내 이름은 마이클이야 만나서 반가워

Attention between Encoder and Decoder

- Transformer



source Hello **Michael** is my **name**, nice to meet you

attention 0.01 0.8 0.01 0.04 0.1 0.01 0.01 0.01 0.01

translate

target 안녕 내 이름은 마이클이야 만나서 반가워

Summary

- Attention 만을 사용한 새로운 모델 Transformer 제안
- 기존 모델들의 병렬 처리 한계 개선
- Long term dependency 문제 해결
- Neural Machine Translation 분야에서 높은 성능을 보임

Wrap-Up Question & Answer ?