Attention, transformer.

1. 배경

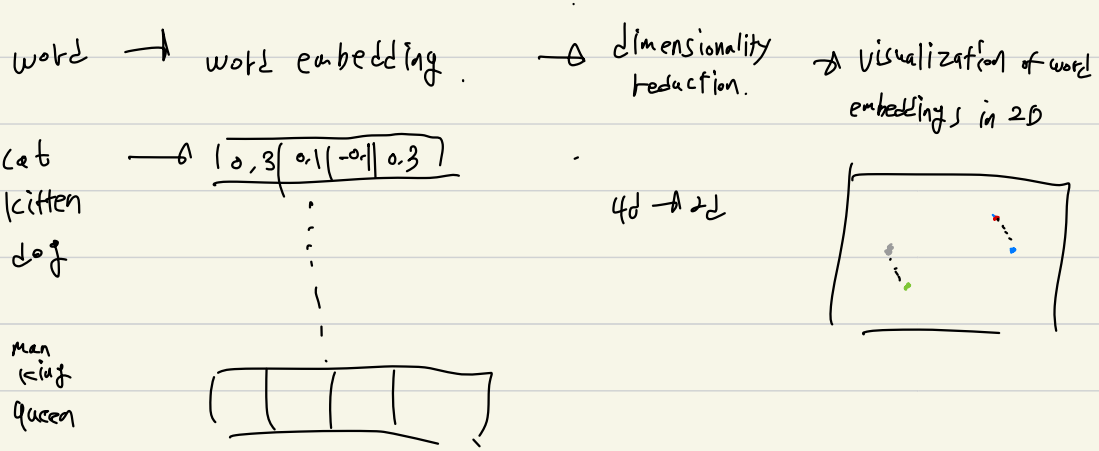
- 기계번역 작업

* 개요

- Input: Sequence of words (번역할 문장들)
- Model : 번역모델
- Output : Sequence of words (번역된 문장들)

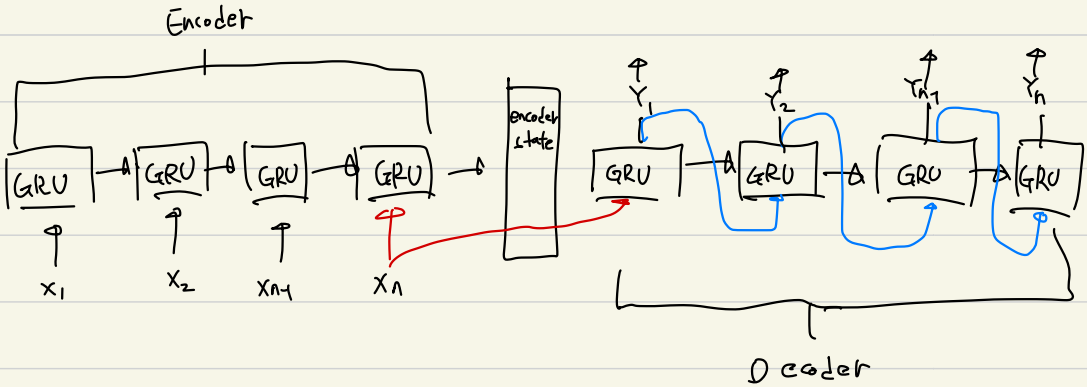
* word Embedding

- 텍스트는 이미지처럼 수치적인 값으로 표현되는 데이터가 아님.
- 단어를 수치적인 자료(벡터)로 표시하기 위한 작업, word embedding이 필요함.
- Word embedding의 방법은 목적이나 모델에 기반하여 매우 다양함
  → 똑같은 단어라도 다른 모델에 따라 임베딩된 벡터가 다름.
- 대용량의 데이터로 잘 학습된 모델의 word embedding 결과를 가져다 사용



*Seq2Seq

- Seq2Seq (sequence-to-sequence)는 LSTM(또는 GRU) cell로 구성된 인코더-디코더 구성의 신경망 모형
- Input: Sequence of text ($R^{T \times d_w}$ 행렬)
- Output: Seaunence of text ($R^{T' \times d_w}$ 행렬)
- 인코더에서 번역할 문장을 차례대로 입력받아 디코더로 넘길 피처 (feature, $h \in R^{1 \times d_h}$)를 학습함
- 디코더에선 인코더에서 넘긴 피처(h)를 입력받아 순차적으로 번역 단어를 생성함



- Seq2Seq 모델의 구조

2. Attention

* Seq2Seq

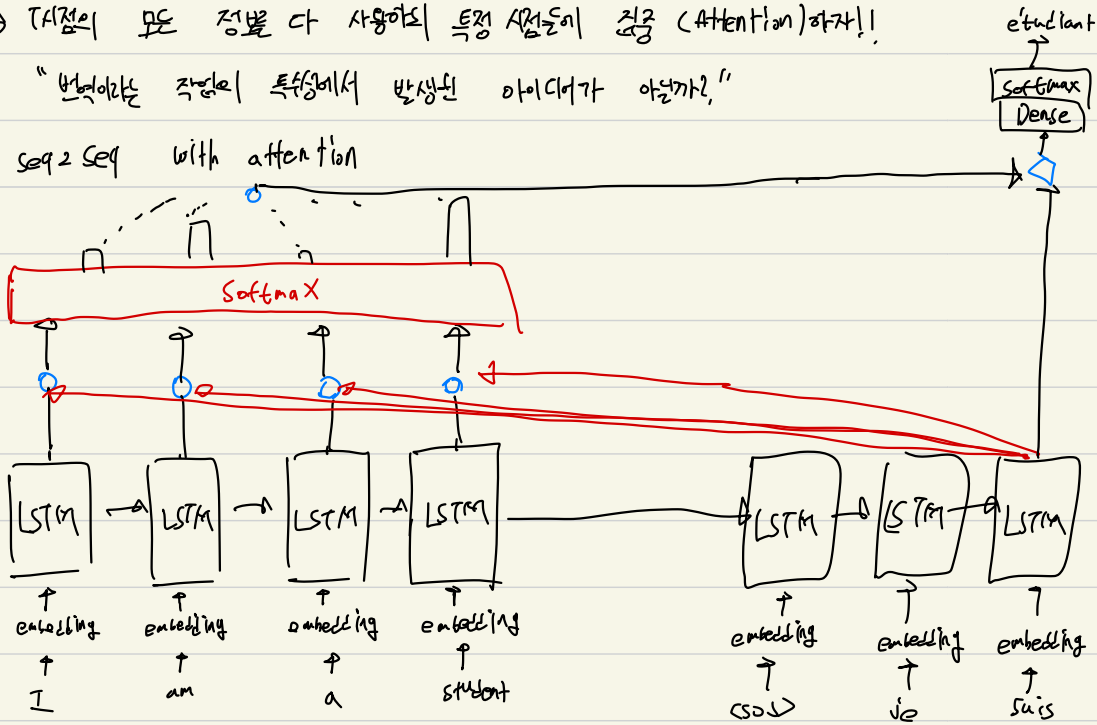
  - T개의 단어로 이루어진 input 시퀀스 ($\in R^{T \times m}$)
  - 하나의 벡터 $h \in R^{1 \times m}$인 인코더의 결과물
  - (T → 1)의 압축이 정보손실을 일으켜서 정확도가 떨어짐

* 해결책

→ T시점의 모든 정보 다 사용하되 특정 시점들에 집중 (Attention)하자!

"번역하는 각각의 특정에서 발생된 아이디어가 아닐까?"

- Seq2Seq with attention



Seq2Seq의 attention 방법 예시

① Attention score

: non-normalized attention weight $\xrightarrow{\text{softmax}}$ attention weight

디코더의 t시점을 기준으로 인코더 시점들의 attention score 계산

- q_t : 디코더 t시점의 LSTM cell의 hidden state 벡터
- $k_i, i \in \{1, \cdots, T\}$: 인코더 i시점의 LSTM cell의 hidden state 벡터
- $q_t^T k_i$: 디코더의 t시점과 인코더의 i시점의 attention score (스칼라)
- $s_t = [q_t^T k_1 \cdots q_t^T k_T]^T$: 디코더의 t시점과 인코더 모든 시점의 attention score 벡터.
- $w_t = \text{Softmax}(s_t) \in R^T$: 디코더의 t시점과 인코더 모든 시점의 attention weight 벡터
  ⇒ 합이 1
- $w_t^T 1 = 1$

⇒ $q_t^T k_j$: 디코더의 t시점과 인코더의 i시점의 attention score

- 두 벡터의 내적은 비슷할 수록 큰 값을 가짐
- 디코더의 t시점의 q_t와 비슷할수록 인코더의 i시점은 큰 가중치를 가짐

② Attention value

: 인코더의 hidden states와 attention weights의 가중 평균

디코더의 t시점을 기준으로 인코더 시점과의 attention value 계산

- $K = [k_1, \cdots, k_T]^T \in R^{T \times d_m}$: 인코더 hidden states들을 쌓은 행렬
- $C_t^T = w_t^T K \in R^{1 \times d_m}$: attention value 벡터

→ Attention value는 디코더의 t시점과의 연관성을 기반으로 입력정보를 사용하여 새롭게 생성한 피처

→ C_t와 k_t를 결합한 벡터를 가지고 기존의 LSTM과 동일한 과정 수행

Attention은 input과 output 시점 간의 연관성을 고려한 피처 추출 방법 (feature learning method)

3. Transformer

- Transformer 요약

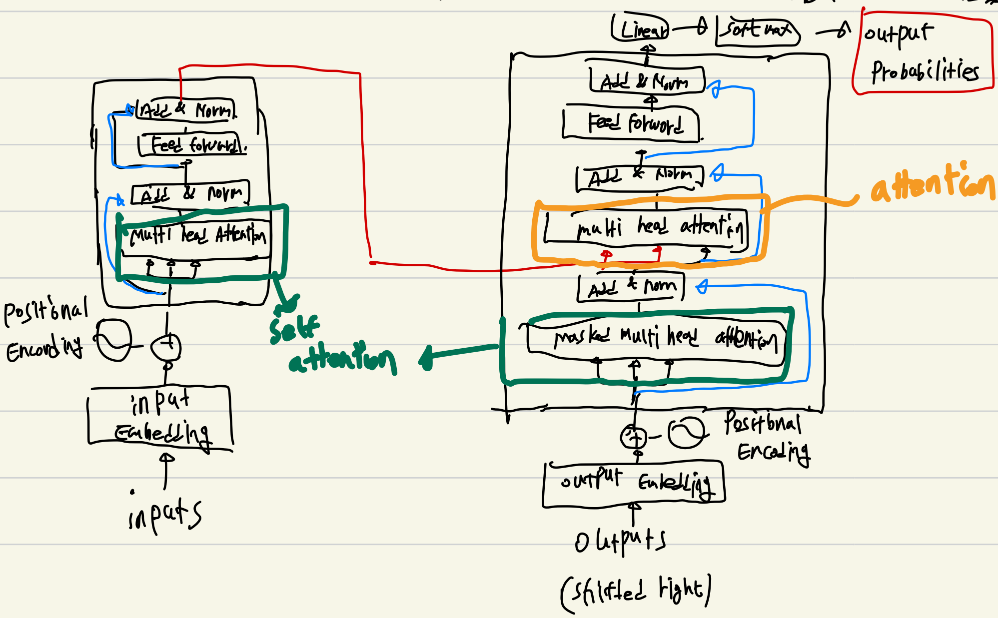
a new simple network architecture, the Transformer, based soley on attention mechanisms, dispensing with recurrence and convolutions entirely.

Superior in quality, being more parelleizable.

→ 모델은 반복과 합성곱으로 인한 기존의 문제점을 해결하고자 과감히 생략하고 attention만 base로 함.
높은 성능과 더 많은 병렬화가 가능하게 됨.

- 배경

- LSTM과 Convolution 기반의 모델에서 순차적인 계산량은 너무 많고 비효율적임.
- LSTM과 Convolution층을 버리고 병렬화가 가능한 연산구조를 가진 모델을 구성
- Self-attention은 병렬화가 가능하고 이미 많은 자연어처리 분야에서 성공적으로 쓰이고 있음.



① Positional encoding

Transformer는 기존의 모델들이 순차적인 연산을 비판하면서 그 구조를 과감하게 없앰.

→ Positional encoding 이라는 대안을 사용함

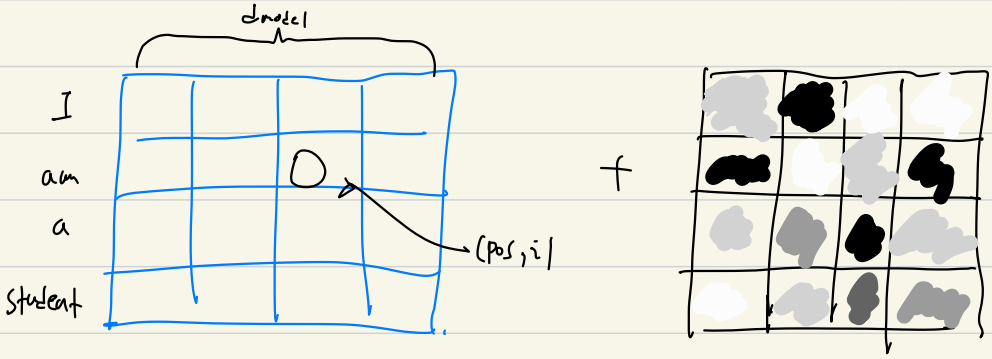
$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

여기서 pos는 단어의 시점, 즉 단어 임베딩 벡터에서의 인덱스

positional encoding 값으로 만든 행렬을 임베딩된 행렬에 더함.

→ 같은 단어라도 위치에 따라 다른 임베딩 벡터를 가짐.



Positional encoding과 임베딩 행렬의 합.

② Self-attention

서로 다른 input과 output 간의 연관성이 아닌 동일한 문장 안에서 단어들 간의 연관성을 통해 새로운 피처 생성

- Self-attention을 하기 위해서는 하나의 입력 행렬(X)에 각각 다른 FF시을 거쳐 query(Q), key(K), value(V)를 생성
- $Q = XW_Q$
- $K = XW_K$
- $V = XW_V$
- Q와 K를 가지고 attention score을 계산하고 attention weight을 계산
- V의 attention weight만큼의 가중 평균으로 새롭게 피처 생성

③ Seq2Seq의 attention과 Transformer의 Self-attention의 차이점.

1. Seq2Seq에서는 디코더 t시점의 attention 연산을 위해서는 t-1 시점까지의 attention 연산이 선행되어야함.

→ 디코더의 모든 시점에서 동시에 진행될 수 없어서 t시점의 hidden state(벡터)만 계산함.

2. Transformer는 순차적인 연산이 없어서 모든 시점에 대해 동시에 attention을 진행할 수 있음.

→ t시점의 벡터만이 아닌 모든 시점 (1, ..., T)의 행렬로 가지고 계산

3. Query

- Attention : 디코더의 t시점에서의 hidden state (벡터)
- Self-Attention : 모든 시점의 단어 행렬 X와 W_Q의 곱 (행렬)

4. Key

- Attention : 인코더의 모든 시점의 hidden state들 모든 행렬 (행렬)
- Self-Attention : 모든 시점의 단어 행렬 X와 W_K의 곱 (행렬)

5. Value

- Attention : 인코더의 모든 시점의 hidden state들 모든 행렬 (행렬)
- Self-Attention : 모든 시점의 단어 행렬 X와 W_V의 곱 (행렬)

(4) Self-attention 연산과정

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\underbrace{\frac{QK^T}{\sqrt{d_m}}}_{\text{attention score}}\right)V$$

attention weight

여기서 d_m로 행렬곱의 연산값이 너무 커지는 것을 방지하기 위한 조정 상수

- $QK^T/\sqrt{d_m} \in R^{T\times T} = [S_1, \cdots, S_T]^T$: 모든 시점의 attention score을 담은 행렬
- $W = \text{Softmax}(QK^T/\sqrt{d_m}) = [w_1, \cdots, w_T]^T \in R^{T\times T}$
- Attention(Q, K, V)의 첫번째 행 :

$$C_1 = \sum_{t=1}^{T} w_{1,t} v_t \qquad \sum_{t=1}^{T} w_{1,t} = 1$$

Self-attention의 의미

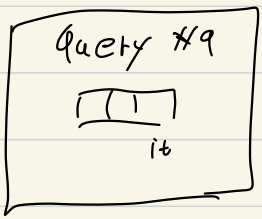
: 내부 단어들 간의 연관성(문맥)을 고려한 피처 생성

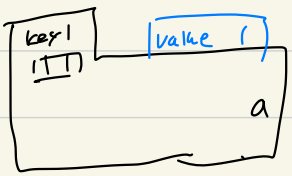
ex) Tom likes a dog. He also likes a cat.

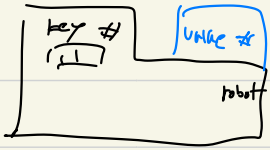
"He" 단어의 문맥적 의미를 주변의 단어들을 통해 생성

- 그림으로 설명

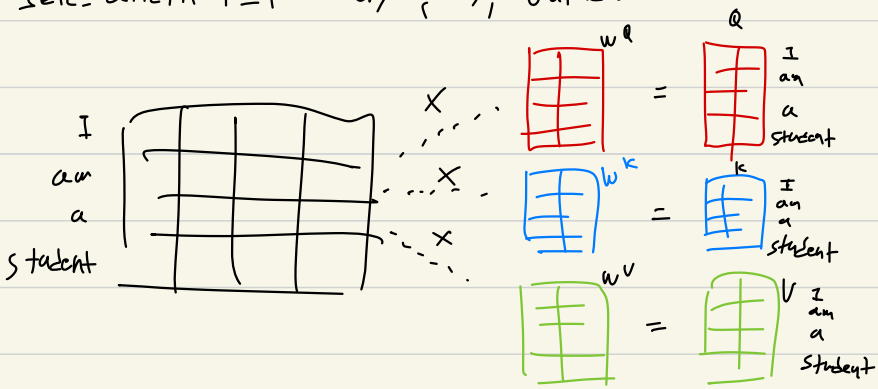
① Database에서의 query, key, value.



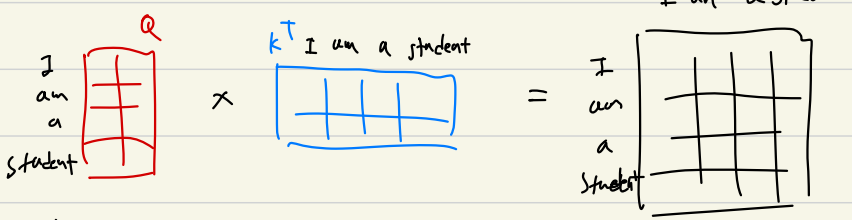




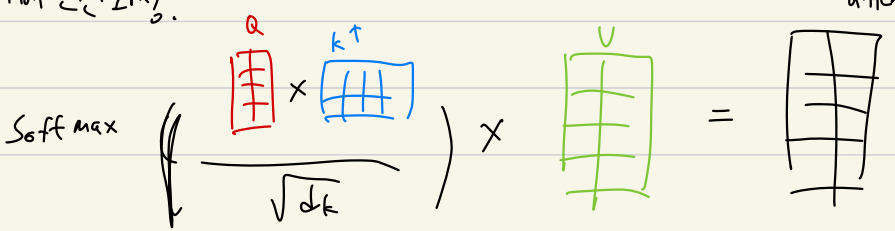
② self-attention 의 query, key, value.



③ self-attention에서 attention score 얻기

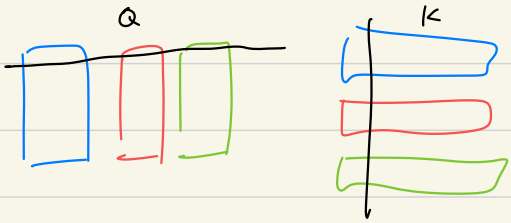


④ attention 연산 과정.



5. Multihead attention : 어텐션 병렬화

- Attention 연산은 행렬을 임베딩 차원의 방향으로 나눠서 연산을 수행할 수 있음.
- 나눠서 연산을 수행한 다음 합쳐서 다음 층에 수행



우리가 진행하는 시계열 예측을 위한 Transformer 모델 → MQ Transformer (Arxiv)