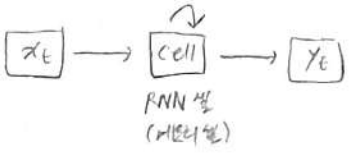
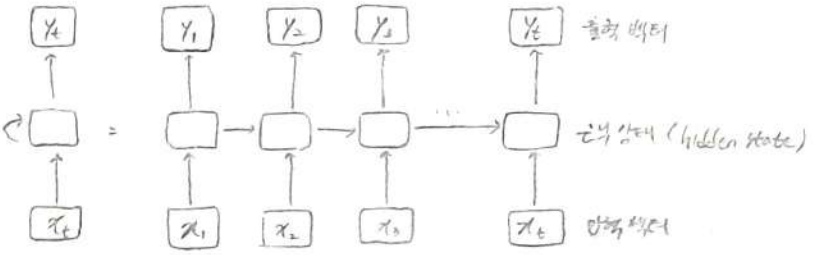
* tokenization : 주어진 코퍼스(corpus)에서 토큰(token)이라 불리는 단위로 나누는 작업

* 순환 신경망 (Recurrent Neural Network, RNN)



: 은닉층이 재귀적임

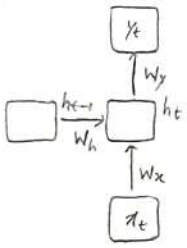


→ 입력과 출력의 길이에 따라서 다르게 설계 가능

one-to-many ex. Image Captioning

many-to-one ex. sentiment classification, spam detection

many-to-many ex. 번역기, 개체명 인식, 품사 태깅



은닉층 : $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 : $y_t = f(W_y h_t + b)$ (f 비선형 활성화 함수 - 시그모이드, 소프트맥스 등)

단어 벡터의 차원이 d, 은닉 상태의 크기가 D_h 일 때

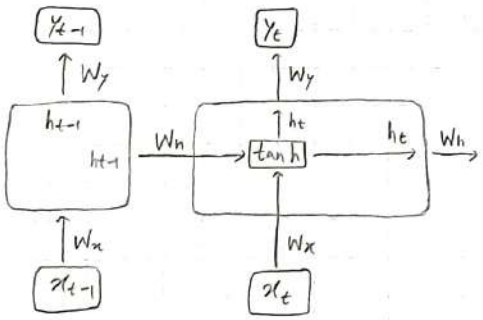
x_t $(d \times 1)$, W_x $(D_h \times d)$, W_h $(D_h \times D_h)$, h_{t-1} $(D_h \times 1)$, b $(D_h \times 1)$

+ Deep Recurrent Neural Network, Bidirectional Recurrent Neural Network

* 장단기 메모리 (Long Short-Term Memory, LSTM)

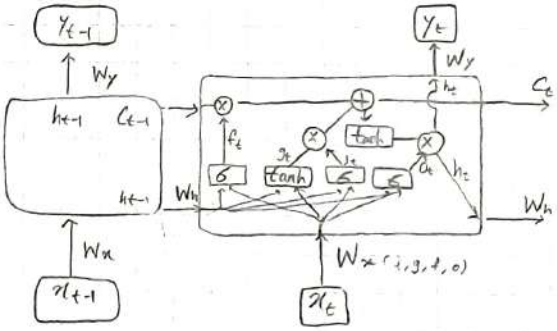
바닐라 RNN의 한계 - 장기의존성 문제 (the problem of Long-Term Dependencies) : 시점(time step)이 지날수록 x_1 정보를 소실

<RNN>



$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

<LSTM>



C_t : 셀 상태 (cell state)

입력게이트 : 현재 정보를 저장하기 위한 게이트 $i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i)$ / $g_t = \tanh(W_{xg} x_t + W_{hg} h_{t-1} + b_g)$

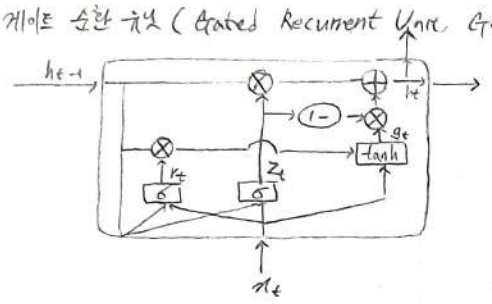
삭제게이트 : 기억을 삭제하기 위한 게이트 $f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$ 0과 1 사이 값. 0에 가까울수록 많이 삭제된 것

셀 상태 : 입력 게이트에서 선택된 기억을 삭제 게이트의 결과만큼 더한 값. $C_t = f_t \circ C_{t-1} + i_t \circ g_t$

삭제 게이트는 이전 시점의 입력을 얼마나 반영할 지 의미하고, 입력게이트는 현재 시점의 입력을 얼마나 반영할지 결정한다

출력게이트 : 현재 시점 t의 은닉 상태를 결정하는 데 쓰인다. $o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o)$ / $h_t = o_t \circ \tanh(C_t)$

* 게이트 순환 유닛 (Gated Recurrent Unit, GRU)



LSTM 보다 단순하다 (매개변수 ↓)

입력 게이트 / 리셋 게이트

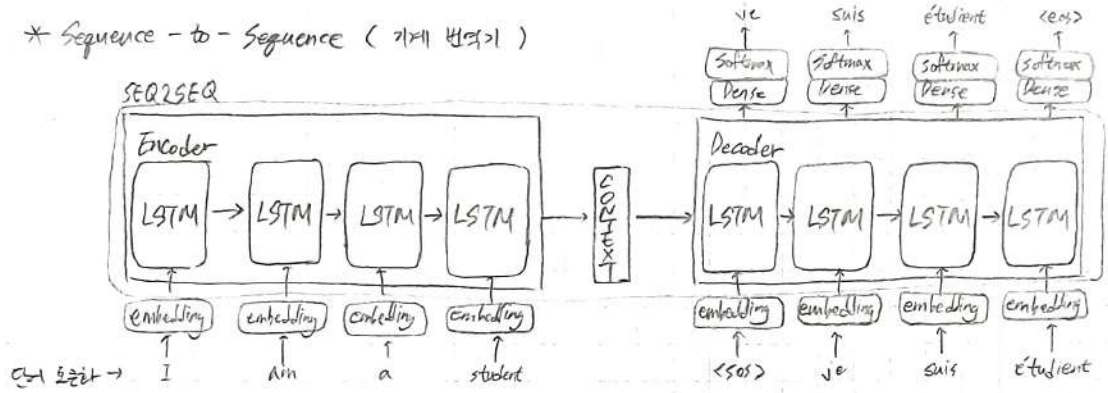
$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$

$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$

$g_t = \tanh(W_{hg}(r_t \circ h_{t-1}) + W_{xg}x_t + b_g)$

$h_t = (1 - z_t) \circ g_t + z_t \circ h_{t-1}$

* Sequence - to - Sequence (기계 번역기)



인코더 : 입력 문장의 정보가 하나의 컨텍스트 벡터로 압축 (Context vector), 디코더에 전송 / 디코더 : 컨텍스트 벡터를 받아 번역된 단어를 한 개씩 순차적으로 출력

훈련 과정 : Teacher forcing (교사 강요) - RNN의 모든 시점에서 이전 시점의 예측값 대신 실제값을 입력으로 주는 방법

테스트 과정 : 인코더 RNN 셀 마지막 시점의 은닉 상태 (컨텍스트 벡터)를 디코더 RNN 셀로 넘긴다

→ 디코더는 문장의 시작을 의미하는 심볼 <sos>가 입력되면 다음에 등장하는 단어를 예측하고, <eos>를 예측할 때까지 반복한다

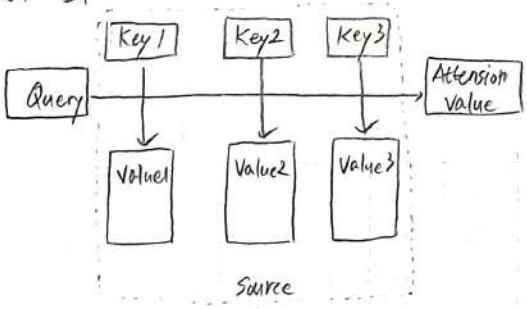
→ 디코더에서 각 시점의 RNN 셀에서 출력벡터가 나오면 소프트맥스 함수를 통해 출력 시퀀스의 각 단어별 확률값을 반환하고, 디코더는 출력 단어를 결정한다.

* Attension Mechanism

seq2seq 모델의 한계 - 1) 컨텍스트 벡터 정보 손실 2) 기울기 소실 문제

→ attension : 디코더에서 출력 단어를 예측하는 매 시점마다, 전체 입력 문장을 참고하되, 연관 부분을 더 집중해서 본다.

- Attension 함수



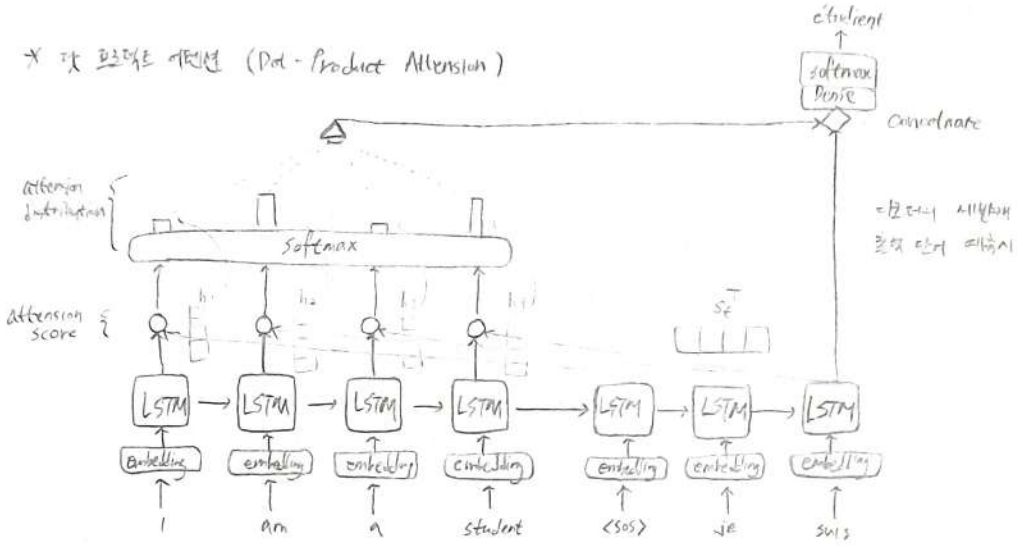
Attension (Q, K, V) = Attension Value

Q = Query : t 시점의 디코더 셀에서의 은닉상태

K = Keys : 모든 시점의 인코더 셀의 은닉 상태들

V = Value : 모든 시점의 인코더 셀의 은닉 상태들

* 닷 프로덕트 어텐션 (Dot-Product Attension)



1) Attension score : 인코더의 모든 은닉 상태 각각이 디코더의 현 시점의 은닉 상태 s_t와 얼마나 유사한지 판단하는 스코어값을 구하기 위해 s_t를 전치하고 각 은닉 상태와 내적한다 (내적하는 것도 유사도를 판단하는 것)

$$\text{Score}(s_t, h_i) = s_t^T h_i$$

어텐션 스코어의 모음값이 e^t 라면 $e^t = [s_t^T h_1, \cdots, s_t^T h_N]$

2) Attension Distribution : e^t에 소프트맥스 함수를 적용하여, 모든 값을 합하면 1이 되는 분포를 만든다.
각각의 값은 어텐션 가중치 (attension weight) 라고 한다.
어텐션 가중치의 모음값인 어텐션 분포를 α^t라 하면 $\alpha^t = \text{softmax}(e^t)$

3) Attension Value : 각 인코더의 어텐션 가중치와 은닉 상태를 가중합 (Weighted Sum) 하여 어텐션 값을 얻는다.

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$

어텐션 값 a_t는 인코더의 문맥을 포함하고 있다고 하여 context vector 라고 불린다.

4) Concatenate : 어텐션 값 (a_t)과 디코더 t 시점의 은닉 상태 (s_t)를 연결한다
v_t 라는 하나의 벡터로 만들고, 이후 $\hat{y}$ 예측 연산의 입력으로 사용한다

5) 출력층 연산의 입력이 되는 $\tilde{s}_t$를 계산한다 : $\tilde{s} = \tanh(W_c[a_t; s_t] + b_c)$ (W_c: 학습 가능한 가중치 행렬)

6) $\tilde{s}_t$를 출력층의 입력으로 사용한다 : $\hat{y}_t = \text{softmax}(W_y \tilde{s}_t + b_y)$

* 바다나우 어텐션 (Bahdanau Attension)

Attention (Q, K, V) = Attention value. Q = Query : t-1 시점의 디코더 셀에서의 은닉 상태

1) Attention Score : $\text{Score}(s_{t-1}, h_i) = W_a^T \tanh(W_b s_{t-1} + W_c h_i)$ (W_a, W_b, W_c: 학습 가능한 가중치 행렬)

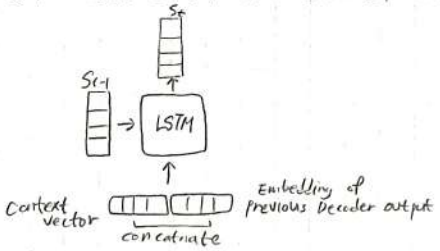
$$\text{Score}(s_{t-1}, H) = W_a^T \tanh(W_b s_{t-1} + W_c H)$$

$$e^t = W_a^T \tanh(W_b s_{t-1} + W_c H)$$

2) Attention Distribution : $\alpha^t = \text{softmax}(e^t)$

3) Attention value : $a_t = \sum_i \alpha_i^t H$. context vector.

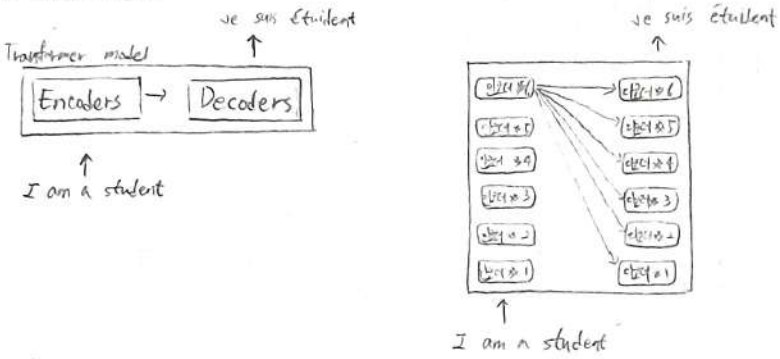
4) Concatenate : 컨텍스트 벡터와 현재 시점의 입력인 단어 임베딩 벡터를 concatnate, 현재 시점의 새로운 입력으로 사용
→ 이전 시점의 셀로부터 전달받는 은닉 상태 s_{t-1}와 현재 시점의 새로 입력으로부터 s_t를 구한다



5) s_t는 출력층으로 전달되어 현재 시점의 예측값을 구하게 된다

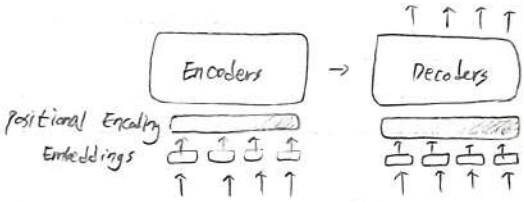
No.

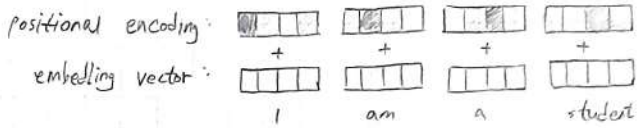
* 트랜스포머 (Transformer)



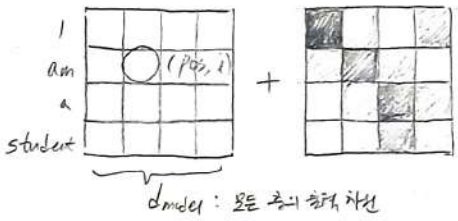
- 포지셔널 인코딩 (Positional Encoding)

포지셔널 인코딩 : 트랜스포머의 입력에서, 단어의 위치 정보를 알기 위해 각 단어의 임베딩 벡터에 위치 정보들을 더하여 모델의 입력으로 사용.





임베딩 벡터가 모여 만들어진 문장 행렬과 포지셔널 인코딩 행렬의 덧셈 연산을 통해 이루어진다



pos : 입력 문장에서의 임베딩 벡터의 위치 / i : 임베딩 벡터 내의 자신의 인덱스

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

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

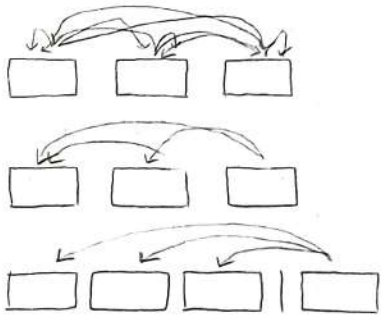
⇒ 주기적인 값을 생성함으로서 서로 다른 위치에 대해 고유한 값을 생성
pos에 대한 스케일링 역할
홀수/짝수 차원을 구분하여 차원 간 다양성을 확보해, 모델이 단어의 상대적 위치를 효과적으로 학습

- Attension

Encoder — Encoder self-Attention

Decoder — Masked Decoder self-Attention

Decoder — Encoder-Decoder Attention



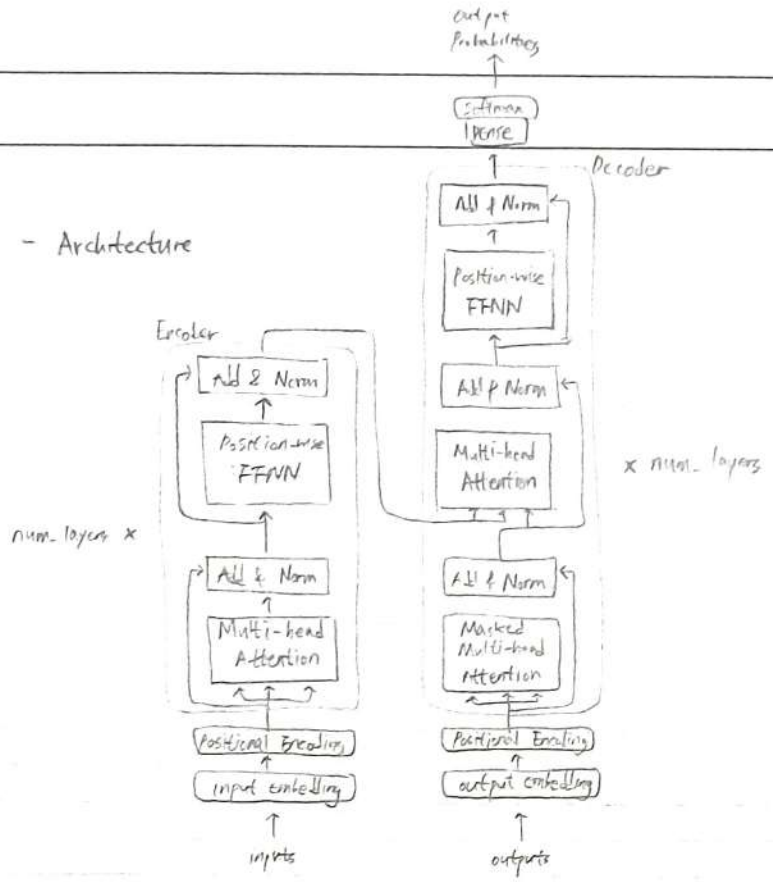
(벡터의 출처)

Q = K = V

Q = K = V

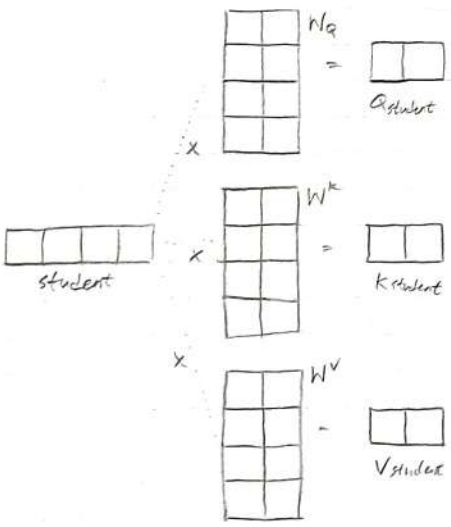
Q 디코더 벡터
K = V : 인코더 벡터

- Architecture



- Self Attention

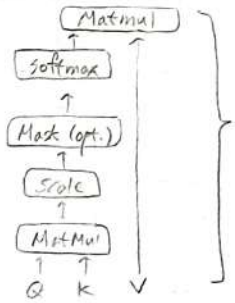
└ Q = K = V : 입력 문장의 모든 단어 벡터들



Q, K, V 벡터들은 d_{model}의 차원을 가지는 단어 벡터들보다 더 작은 차원을 가진다. 하이퍼 파라미터 num_head로 결정된다

가중치 행렬은 $d_{model} \times (d_{model} / num_head)$의 크기를 가진다

모든 단어 벡터 각각에 Q, K, V 벡터를 갖는다



Scaled dot-product Attention

(mask : 문장에 <PAD> 토큰이 있을 경우 어텐션에서 제외)

└ Scaled dot-product Attention : 닷-프로덕트 어텐션에서 스케일링 과정을 추가한 것

$$Score(q,k) = \frac{q \cdot k}{\sqrt{n}}, \quad (\text{논문에서 } n = d_{model} / num\_head = d_k = d_v)$$

└ 행렬 연산으로 일괄 처리 - 문장 행렬에 가중치 행렬을 곱하고, 각 단어의 어텐션 값을 가지는 어텐션 값 행렬로 구한다

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

문장의 길이를 (seq_len, d_{model})이고 d_k는 Q와 K벡터의 차원, d_v는 V벡터의 차원 일때

Q와 K 행렬 크기는 (seq_len, d_k), V 행렬의 크기는 (seq_len, d_v)

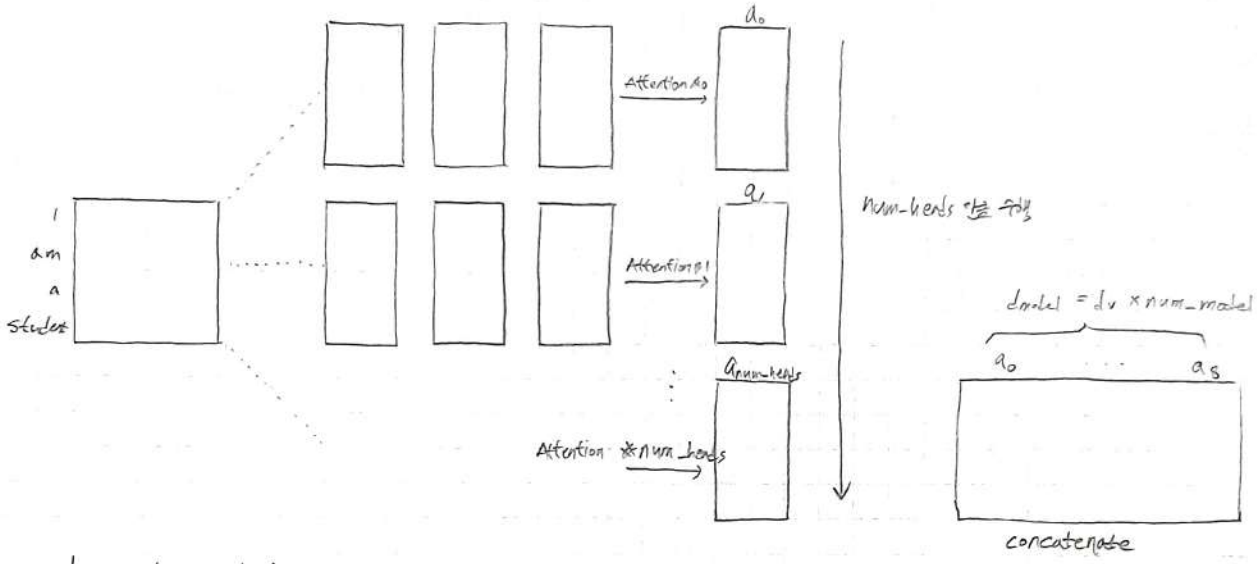
→ 가중치 행렬 W_Q, W_K의 크기는 (d_{model}, d_k), W_V의 크기는 (d_{model}, d_v)

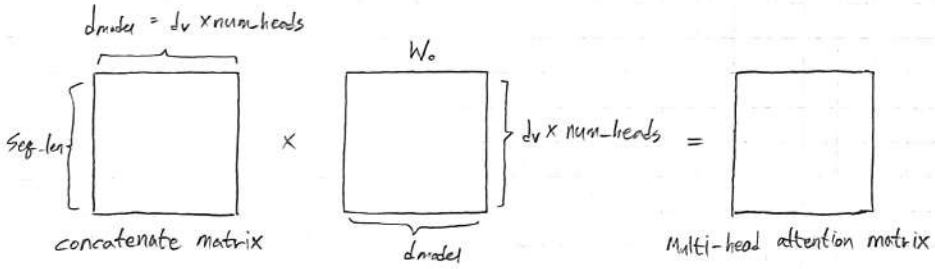
어텐션 값 행렬 A의 크기는 (seq_len, d_v)

— Encoder

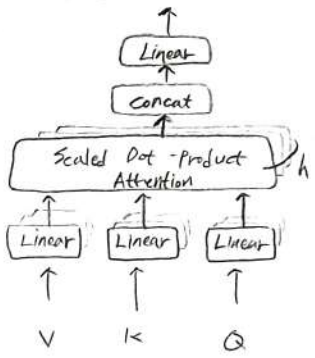
num_layers (하이퍼 파라미터) 개수의 인코딩 층을 쌓는다.
인코딩 층은 크게 서브층(sublayer) 멀티 헤드 어텐션 (self attention 배웠까지 사용)과 피드 포워드 신경망으로 이루어져 있다.

└ Multi-Head Attention
병렬 어텐션 : 다른 시각으로 정보들을 수집하겠다는 것





병렬 어텐션 수행 → 모든 어텐션 헤드 연결 (concatenate) : (seq_len, dmodel)
→ 연결되는 행렬을 가중치 W0와 곱함 → 멀티-헤드 어텐션 행렬 : (seq_len, dmodel)

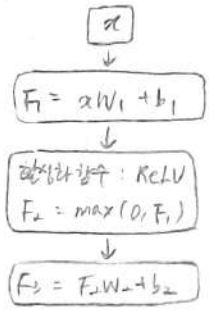


1) WQ, WK, WV에 해당하는 d_model 크기의 밀집층 (Dense layer)을 지나게 한다
2) 지정된 헤드 수 (num_heads) 만큼 나눈다 (split)
3) Scaled Dot-Product Attention.
4) 나뉘었던 헤드들을 연결 (concatenate) 한다
5) WO에 해당하는 밀집층을 지나게 한다.

└ Position-wise FFNN (Fully-connected FFNN)

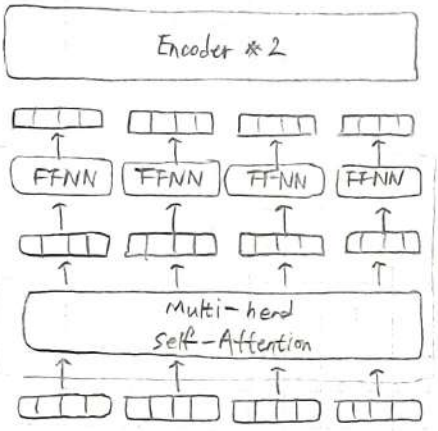
$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$

매개변수 W_1, b_1, W_2, b_2는
하나의 인코더 층 내에서는 다른 문장, 다른 단어에 동일하게 적용, 인코더 층마다는 다른 값.



멀티 헤드 어텐션의 결과로 나온 (seq_len, d_{model})의 크기를 가지는 행렬
가중치 행렬 W_1의 크기는 (d_{model}, d_{ff})

가중치 행렬 W_2의 크기는 (d_{ff}, d_{model})



인코더 #1의 출력
인코더 #2의 입력
(seq_len, d_{model})

인코더 #1의 입력
(seq_len, d_{model})

└ 잔차 연결 (Residual connection)과 층 정규화 (Layer Normalization)

Add & Norm 개념

1) 잔차 연결

서브층의 입력과 출력을 더한다. : $x + sublayer(x)$

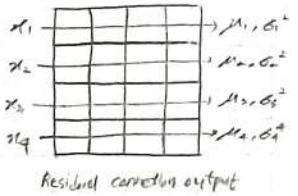
ex. $H(x) = x + Multi\text{-}head\ attention(x)$

Multi-head Attention input + Multi-head attention output = Residual Connection output

⇒ 층 정규화

$LN = LayerNorm(x + Sublayer(x))$

텐서의 마지막 차원에 대해서 평균과 분산을 구하고, 이를 가지고 어떤 수식을 통해 값을 정규화하여 학습을 돕는다



Residual connection output

1) 평균과 분산을 통해 정규화

벡터 x_i의 각 k차원의 값이 정규화 $\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$

2) 감마와 베타 도입

γ | 1 | 1 | 1 | 1 |

β | 0 | 0 | 0 | 0 |

⇒ $ln_i = \gamma\hat{x}_i + \beta = LayerNorm(x_i)$

x_i는 ln_i나 벡터로 계산된다.

No.

- Decoder

  └ Masked Multi-head Self-Attention : 첫 번째 서브층

  look-ahead mask : 트랜스포머는 문장 행렬로 입력을 한 번에 받기에
  현재 시점의 예측에서 현재 시점보다 미래에 있는 단어들을 참조하지 못하도록 하는 것.

  어텐션 스코어 행렬에서 마스킹을 수행한다

  └ Encoder - Decoder Attention (Multi-Head Attention) : 두 번째 서브층

  Query : 디코더 행렬 / key = value : 인코더 행렬 (Self Attention X)

- 학습률

  학습률 스케줄러 (Learning rate Schedular) : 미리 학습 일정을 정해두고 그 일정에 따라 학습률이 조정되는 방법

  step_num (단계) : 옵티마이저가 매개 변수를 업데이트 하는 한 번의 단계 횟수
  step_num이 warmup_steps 보다 작을 경우 학습률을 선형적으로 증가시키다가, 초과하면 step_num의 제곱근에 따라 감소시킨다.

  $lrate = d_{model}^{-0.5} \times \min(step\_num^{-0.5}, step\_num \times warmup\_steps^{-1.5})$