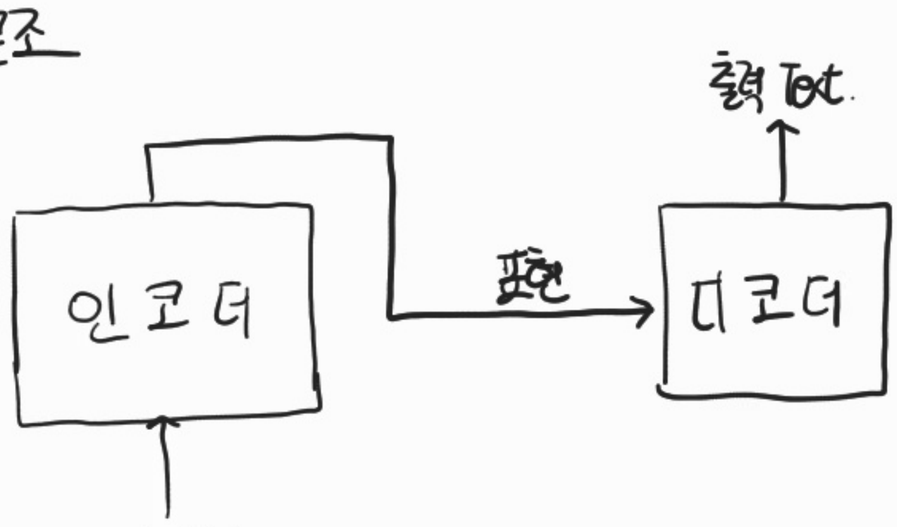
트랜스포머

- RNN과 LSTM은 장기의존성 문제 발생 → Deep 해지면,
- 장기의존성 문제를 극복한 (순수 어텐션 사용?) 트랜스포머 아키텍처 등장
- 기본 골조



트랜스 포머 인코더

- 트랜스포머 인코더 블록은 N개의 인코더가 쌓인다.
- 각각의 인코더에는 멀티 헤드 어텐션 과 피드포워드 layer 가 존재

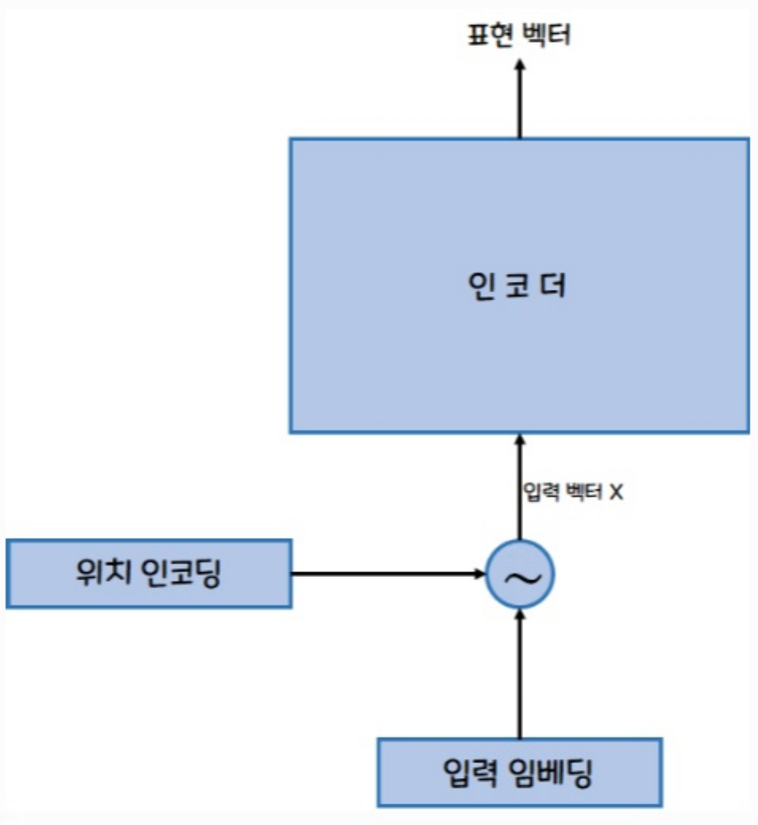
멀티 헤드 어텐션

- 임베딩 (embedding)
  - 입력 문장의 각각의 단어를 표현 하는 벡터 (ex) Shape 7×64 = 단어수 × 표현 차원(N))
  - 임베딩은 학습 과정에서 같이 학습된다. → 가중치가 존재? (word2vec 처럼 (추론 방식))

- 위치(정보) 인코딩 (Positional encoding)
  - RNN은 입력 단어를 순차 적으로 대입한다.
  - 트랜스포머는 RNN 계열의 장기의존성 문제를 해결하기 위해 모든 단어를 병렬 적으로 입력한다. → 하지만 문장이 가지는 순서 정보가 유지 되지 않음
  - 따라서 위치 정보를 담기 위해 "위치-인코딩"을 통해 임베딩한 값에 더해줌.
  - 책에서는 "사인파 함수"를 적용하여 위치 인코딩 행렬 P을 계산 함.
    - $P[pos, 2i] = \sin\left(\frac{pos}{1000^{2i/d}}\right)$ ⇒ 짝수열
    - $P[pos, 2i+1] = \cos\left(\frac{pos}{1000^{2i/d}}\right)$ ⇒ 홀수열

    → P의 Shape은 임베딩 행렬과 동일

  - 위치-인코더 행렬 P는 임베딩 행렬과 "요소별 합"을 통해 계산되어 인코더의 입력 행렬 X로 표현된다.



멀티헤드 어텐션.

- 셀프 어텐션.

  . 각 단어를 기준으로 주어진 문장에 있는 모든 단어를 연결 →

  . 입력 행렬 X 로부터 쿼리(Q), 키(K), 벨류(V) 행렬을 생성한다.

    • Q, K, V는 서로 다른 가중치 행렬을 X와 내적하여 만듦
    • 이때 사용된 가중치는 학습을 통해 최적 값에 수렴.
    • $Q = X \cdot W^Q$
    • $K = X \cdot W^K$
    • $V = X \cdot W^V$

1 단계 : Q와 K^T를 내적

: 벡터간 (Q, K)의 유사도를 구함

⇒ 유사도 행렬.

2 단계 : $Q \cdot K^T$ 행렬의 열차원의 제곱근으로 나눔.

⇒ 정규화 (안정적인 경사값을 위하여)

3 단계 : $\frac{Q \cdot K^T}{\sqrt{d}}$ 에 Softmax 함수를 취함

⇒ 확률값으로 일정한 "비율"로써 생각 가능 (어텐션 개념)

⇒ Score 행렬 = $\text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}}\right)$

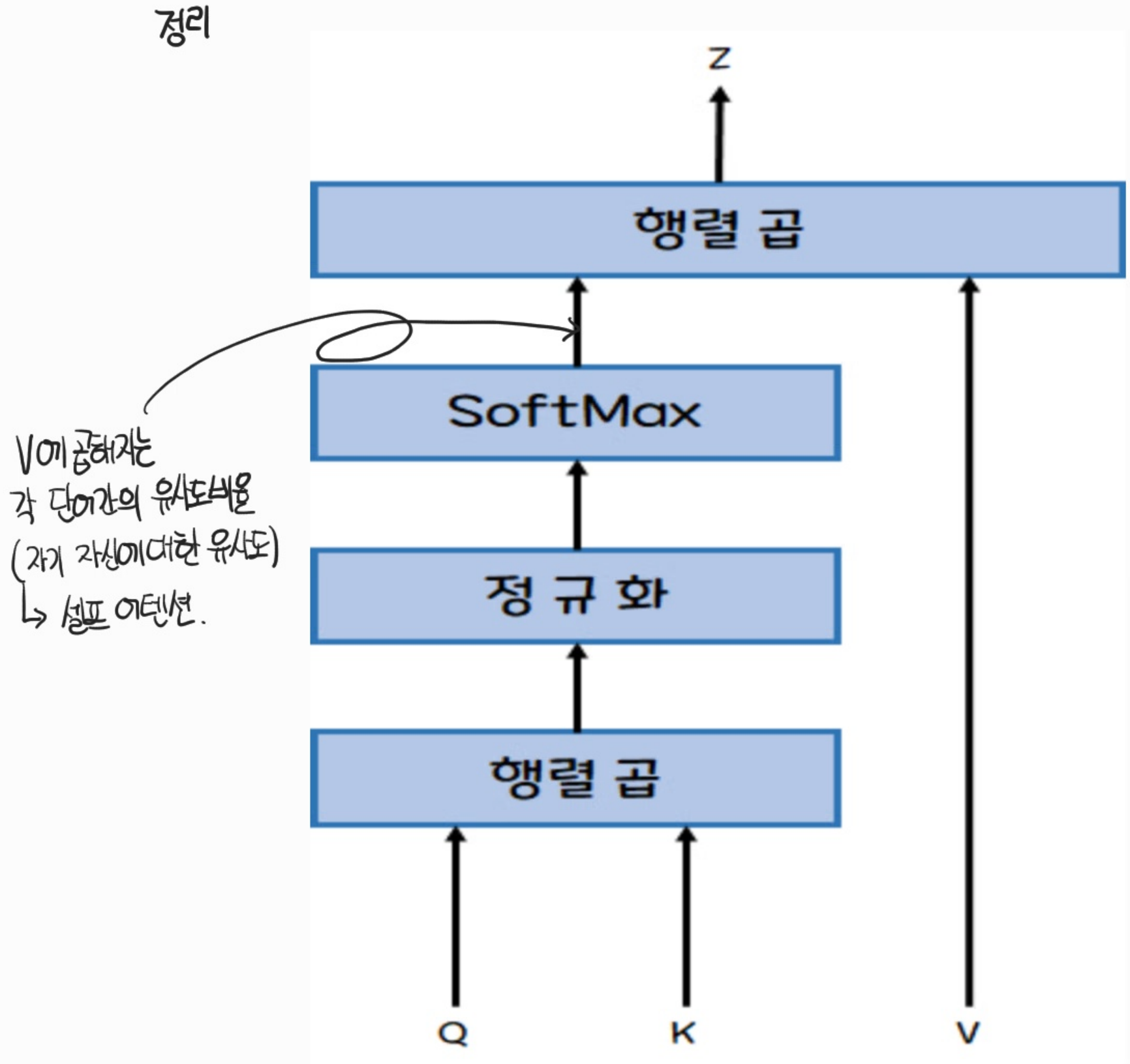
4 단계 : Score와 V를 곱(내적?)하여 어텐션 행렬 Z를 구함.

⇒ 어텐션 행렬 Z는 각 점수(Score)를 기준으로 가중치가 부여된 행렬

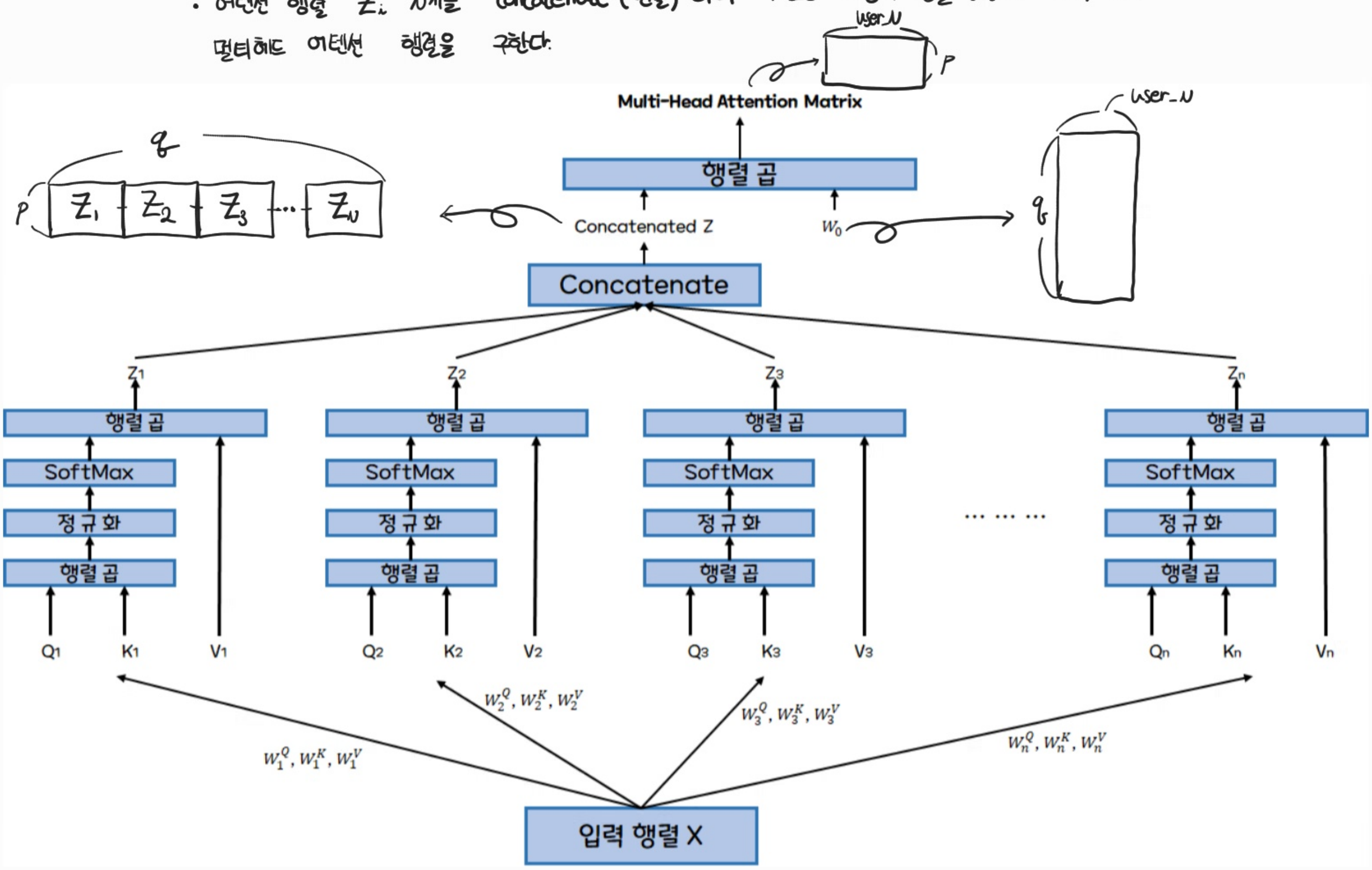
⇒ $Z = \text{Score} \cdot V = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}}\right) \cdot V$

⇒ 책은 내적이 아닌데, 찾아보니 내적으로 설명된 곳도 있음.

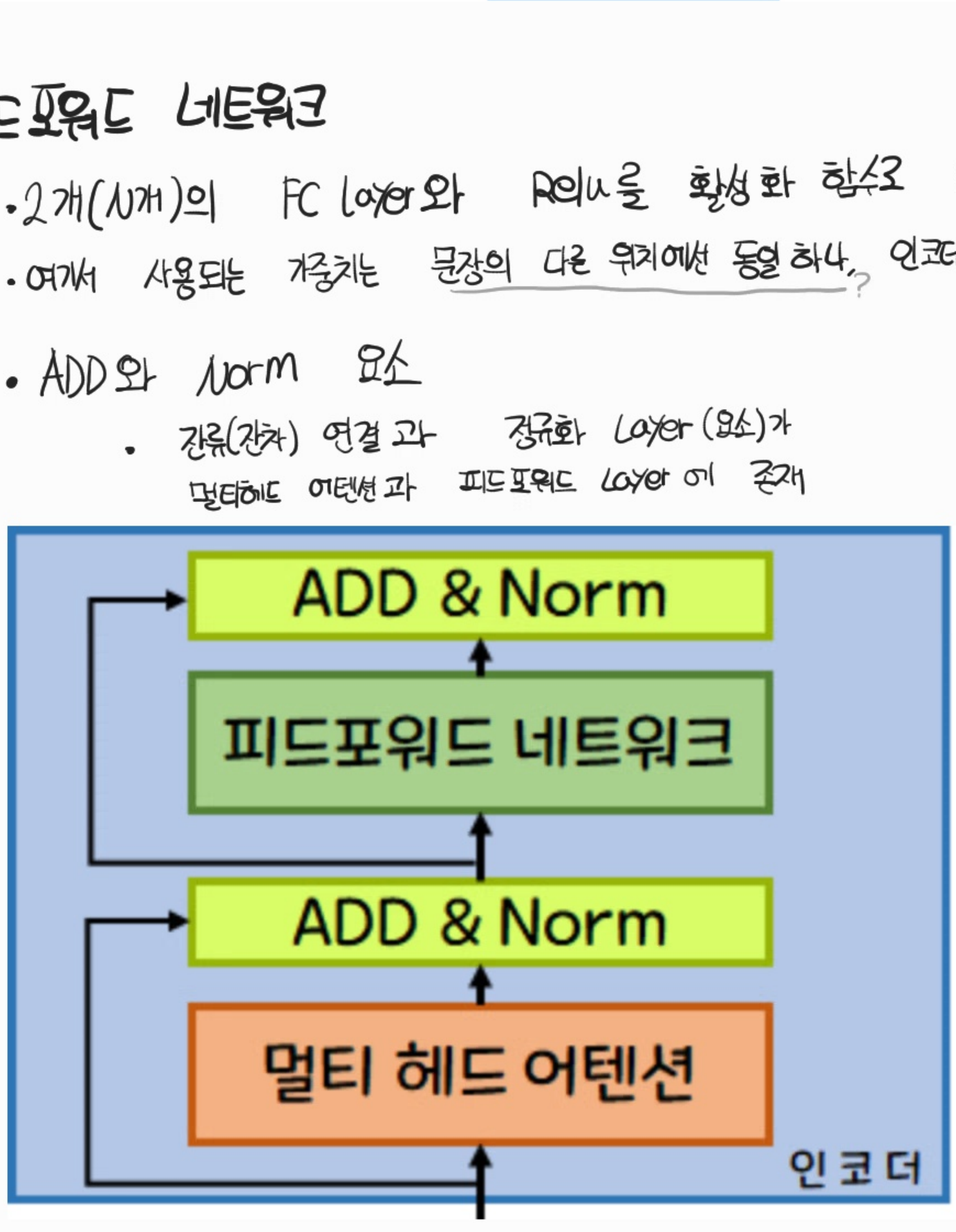
정리

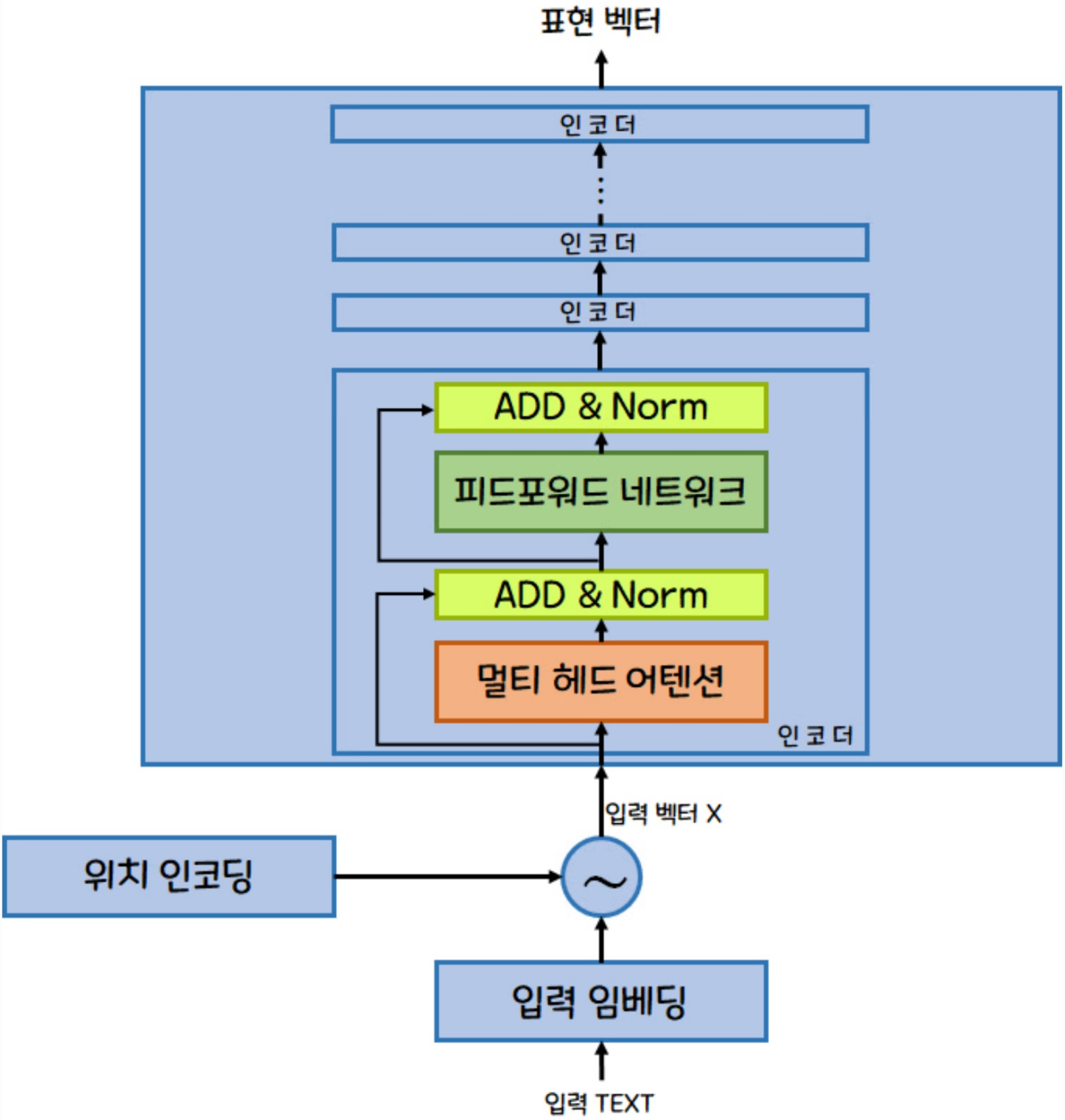


- 멀티 헤드 어텐션
  - 셀프 어텐션은 사실 싱글 헤드 어텐션으로 볼수 있다.
  - 따라서, 멀티 헤드 어텐션은 셀프 어텐션을 여러번 수행 한 것이다.
    ⇒ 정확도를 높이기 위해서 멀티-헤드 어텐션을 사용하면, 더 정확하게 문장을 이해할 수있다는 가정.
    ⇒ 오분류가 일어날 위험을 줄이는 것으로 해석(?)
  - 어텐션 행렬 Z_i N개를 concatenate (연결) 하여 새로운 가중치 행렬 W_0와 내적 하여 멀티헤드 어텐션 행렬을 구한다.



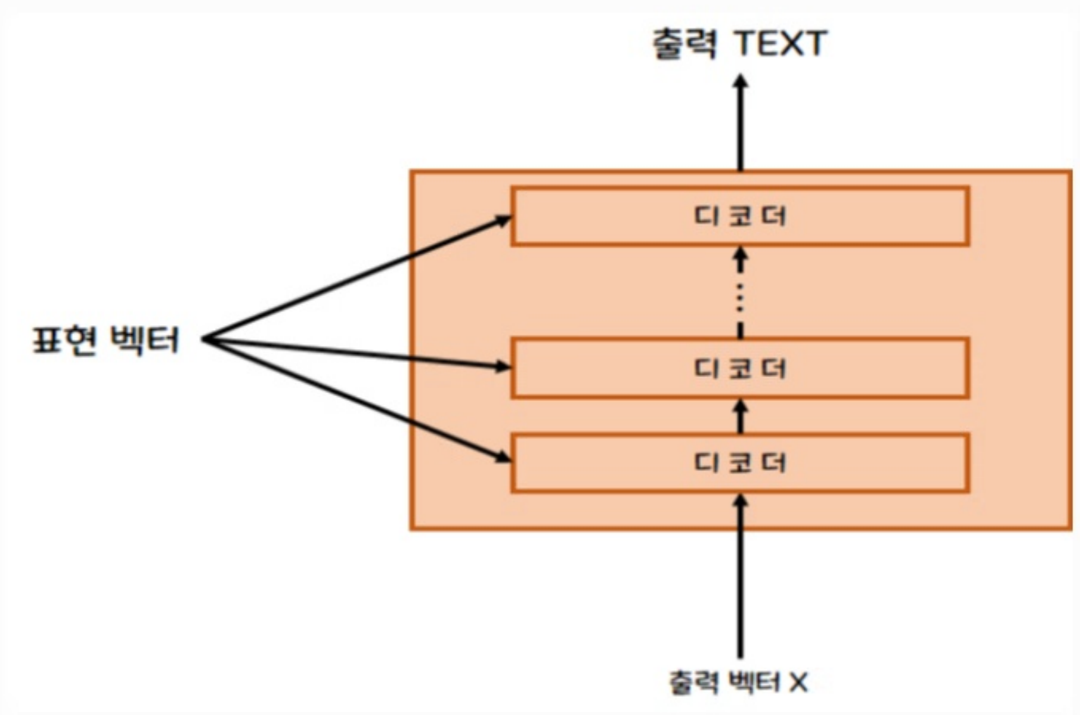
- 피드포워드 네트워크
  - 2개(N개)의 FC layer와 ReLU을 활성화 함수로 가진다.
  - 여기서 사용되는 가중치는 문장의 다른 위치에선 동일하나, 인코더 블록간엔 독립적.
  - ADD와 Norm 요소
    - 잔류(잔차) 연결과 정규화 layer (요소)가 멀티헤드 어텐션과 피드포워드 layer에 존재



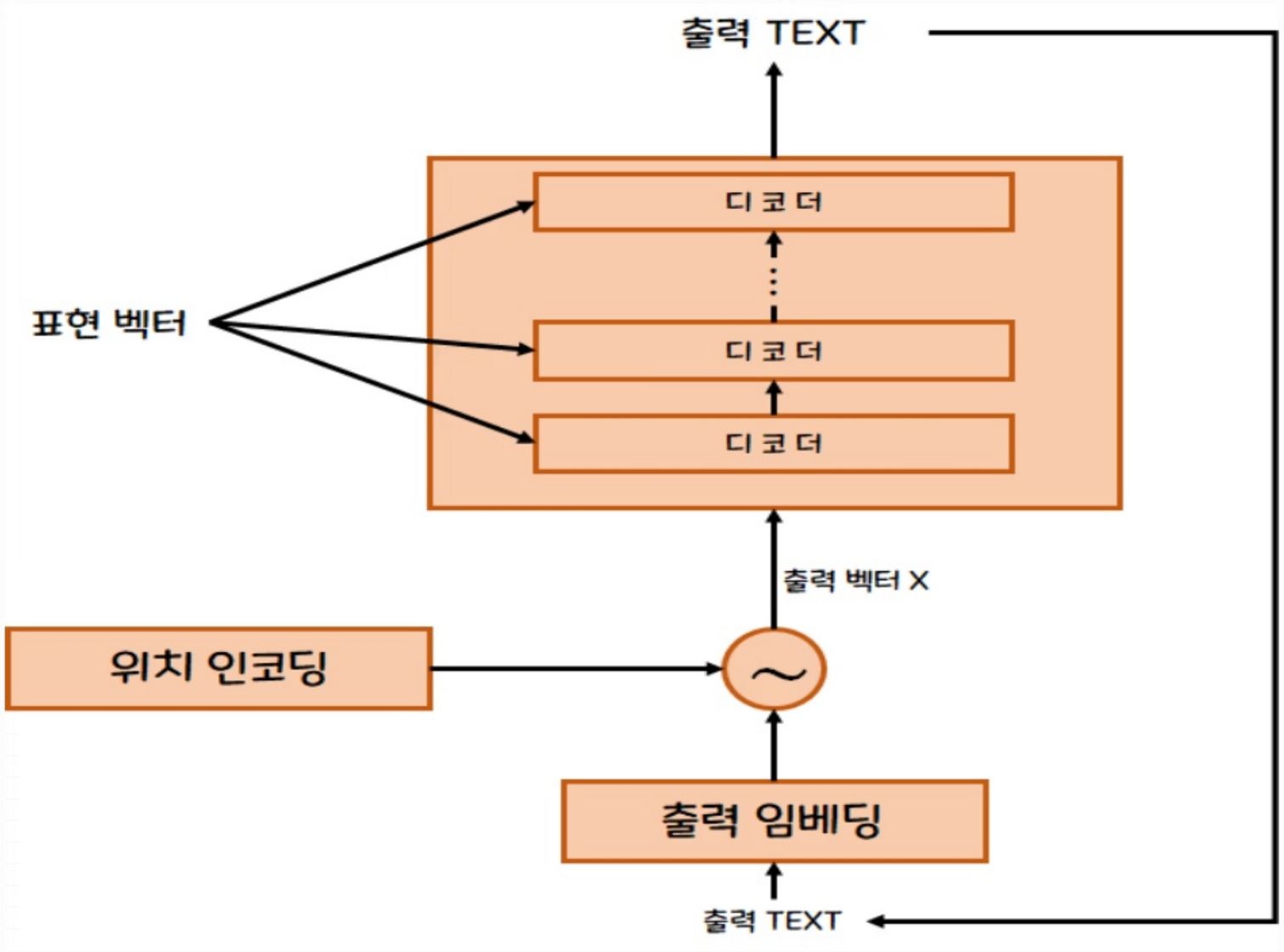


트랜스포머 디코더

- 인코더의 출력 값인 "표현" 벡터를 통해 원하는 출력을 만드는 과정
- 디코더 또한 N개를 쌓을 수 있다.
- 디코더의 입력은 하위 디코더의 출력 값 뿐만 아니라 인코더의 "표현" 벡터도 또한 모든 디코더의 입력으로 들어간다.



- 디코더 또한 최초 <SOS> 토큰을 시작으로 디코더가 출력한 단어를 바로바로 다음 출력의 임베딩 벡터로 사용한다.
- 그리고 인코더와 똑같이 위치 인코딩을 통하여 출력 Text에 대한 순서 정보를 가진다.



- 디코더 내부 동작
  - 마스크된 멀티 헤드 어텐션
  - 멀티 헤드 어텐션
  - 피드포워드 네트워크
- Dense + Soft max Layer

마스크된 멀티헤드 어텐션.

- 동작 자체는 멀티-헤드 어텐션과 동일
- 디코더의 입력 데이터는 이전 Time의 단어들이다.
  예를들어 최초에는 <SOS> 이고 하나씩 추가 될 것
- 즉, 디코더는 출력했던 단어 까지만 봐야함으로 해당 단어 이후의 단어를 masking 한다.
  가령, 인코더에서 "나는 어제 카페에 갔다"의 정보가 디코더에 들어와 디코더가 "I" 까지 예측한 상황에서 다음 단어인 "went"를 마스킹 하지 않는다면, 학습의 의미가 없다.
  ∴ went 이후의 어휘를 masking 한다.
- Q, K, V 행렬을 생성하고 $Q \cdot K^T/\sqrt{d}$ 에서 Softmax를 취하기 전 masking할 값을 $-\infty$로 변경한다. → 이후 동작은 동일
- 이후 마스킹된 행렬 Z를 연결해서 W_0와 내적 한 후 결과를 도출한다.

$$Q \cdot K^T/\sqrt{d} = \begin{array}{c} \\ \langle SOS \rangle \\ I \\ went \\ to \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} \langle SOS \rangle & I & went & to & Cafe & \cdots \end{array} \\ \begin{bmatrix} 9.21 & -\infty & -\infty & -\infty & -\infty & \cdots \\ 5.1 & 12.7 & -\infty & -\infty & -\infty & \cdots \\ 2.7 & 4.6 & 11.5 & -\infty & -\infty & \\ \vdots & & & & & \end{bmatrix} \end{array}$$

멀티-헤드 어텐션

- 인코더와 달리 디코더의 멀티-헤드 어텐션은 입력이 2개다.
  하나는 이전 layer 출력 값 M 벡터이고,
  다른 하나는 인코더의 출력인 표현 벡터 R이다.
  → 여기서 인코더와 디코더의 상호 작용이 발생한다. (인코더-디코더 어텐션 layer)

- Q는 M벡터를 통해, K, V는 R벡터를 통해 생성한다.
  why? ⇒ Q행렬은 타깃문장의 표현을 포함 함으로 타깃문장의 정보를 담고 있는 M에서
  K, V는 입력 문장의 표현을 가지기 때문에 R을 참조한다.

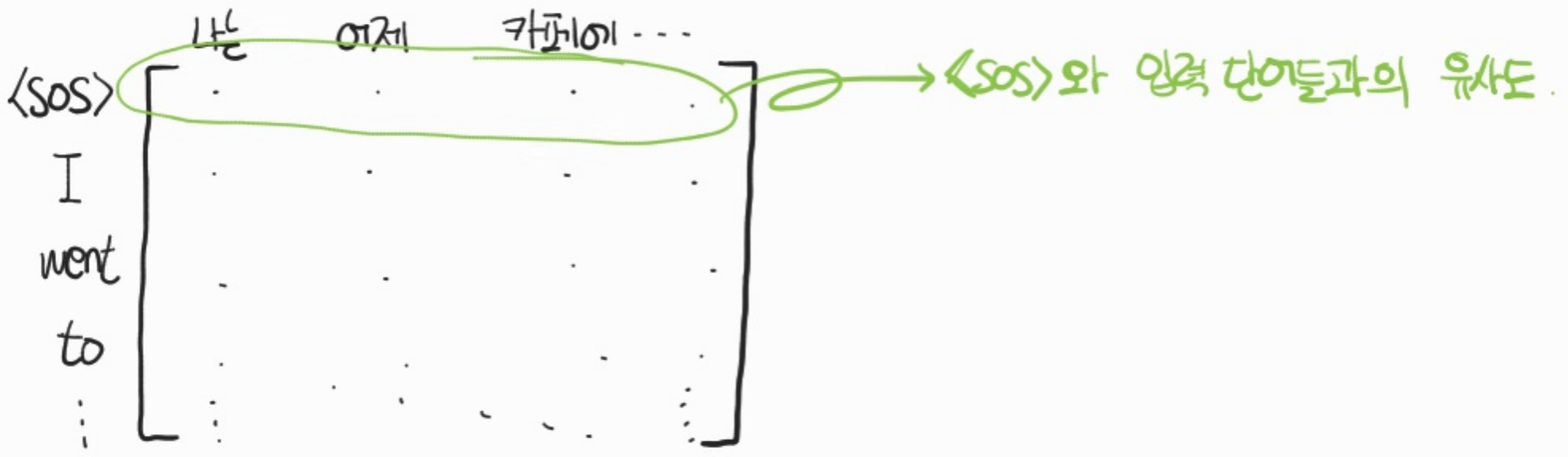
  $Q_i = M \cdot W_i^Q$

  $K_i = R \cdot W_i^K$

  $V_i = R \cdot W_i^V$

- $Q \cdot K^T$ 을 자세히 보면,
  대충, M(타겟 문장의 정보)와 R(표현 벡터) 간의 유사도 행렬을 계산하는 것으로 해석 가능.

$\therefore Q \cdot K^T =$



- 이후에는 Z_i를 구하기 위한 과정.

  $score = softmax(Q \cdot K^T / \sqrt{d})$

  $Z_i = score \cdot V$

- 피드포워드 네트워크

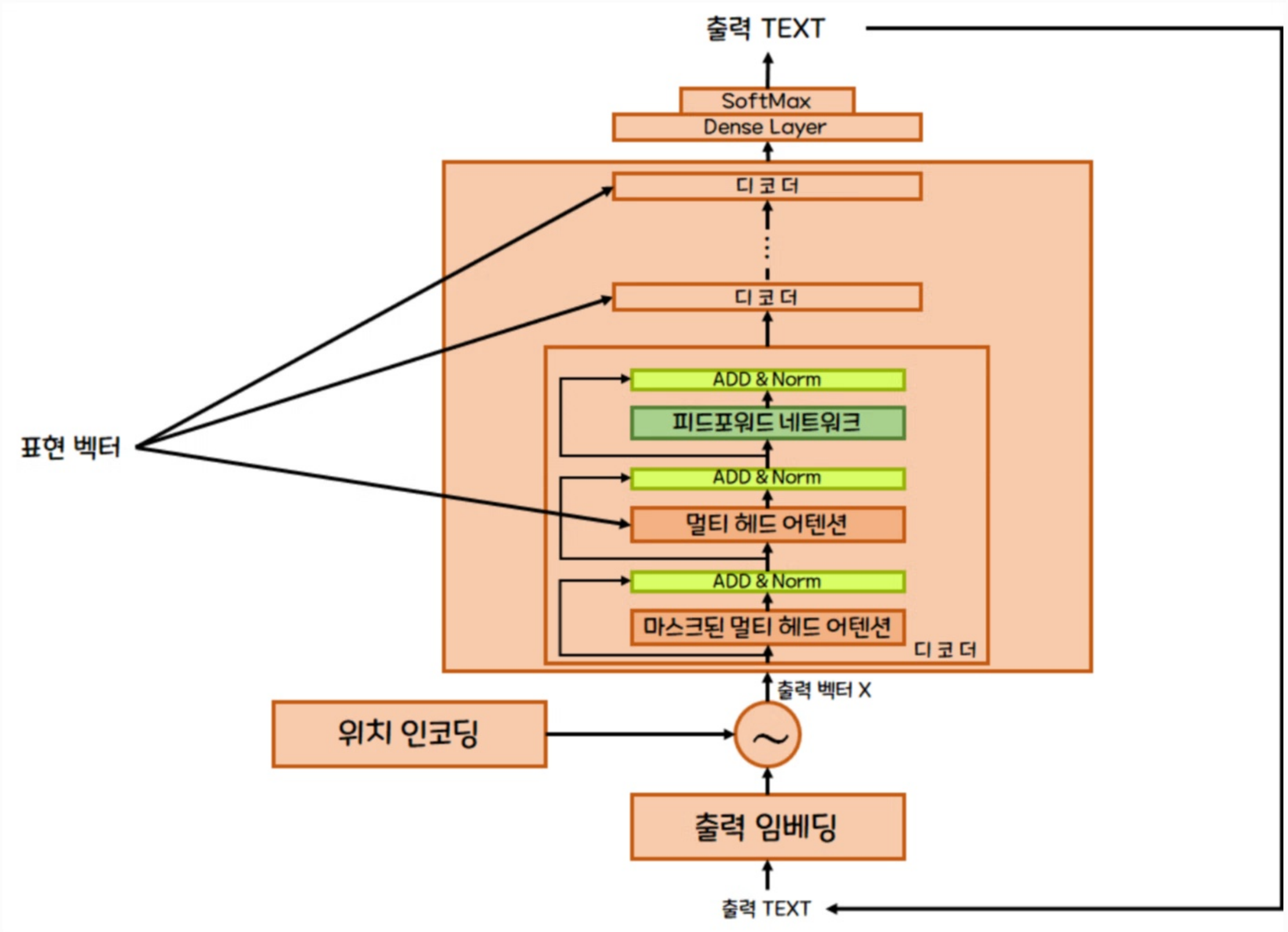
- 인코더의 그것과 동일하게 작동
- 멀티-헤드 어텐션 layer에서 나온 결과값을 통해 계산.

+ 디코더 또한 ADD와 Norm 요소가 각 layer에 배치됨.

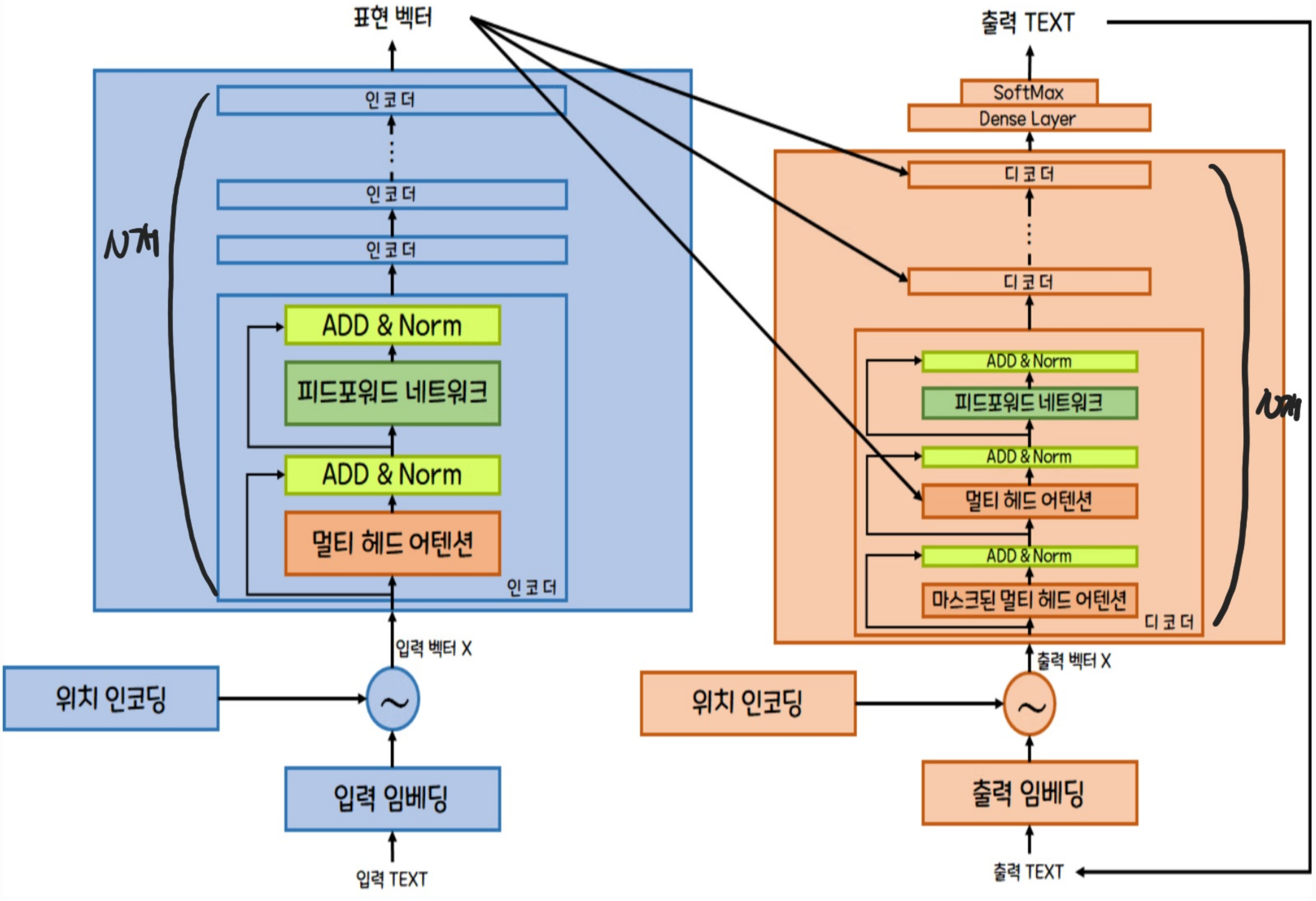
Dense / Softmax layer

- 최상위 디코더에서 온은 출력 값을 Dense / Softmax layer에 전달.
- Dense layer의 경우 출력 차원이 어휘(vocab)의 크기이다.
  ↳ Task에서 알고 있는 단어가 (vocab이) 10개라면 Dense layer의 크기도 10×10일 것
- 이 Dense layer의 결과를 Softmax 함수를 취하여 확률로써 표현하고
- 최종적으로 Decoder 단에서는 확률이 높은 단어를 선택해 결과로 내놓는다.

정리 (그림)



Transformer



트랜스포머 학습

- loss_fn ?

  이후에 대한 확률 분포를 예측하고 확률이 가장큰 단어를 선택 함.
  실제 확률 분포와 예측 확률 분포의 차이가 최소가 되도록
  ⇒ Cross_Entropy

. 또한, 과적합 방지로 각 Layer에 Drop_out을 적용하고
  임베딩, 위치 인코딩 합을 구할 때에도 Dropout을 적용한다 함.