9. Self-Attention and Transformers

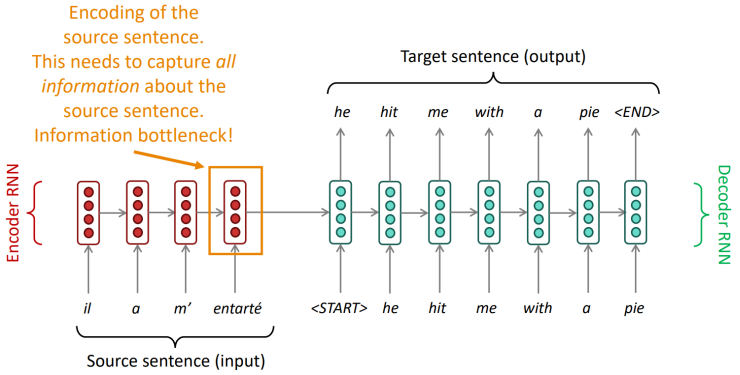
0. Attention

(사실 8강 앞부분 내용임)

Seq2Seq의 한계

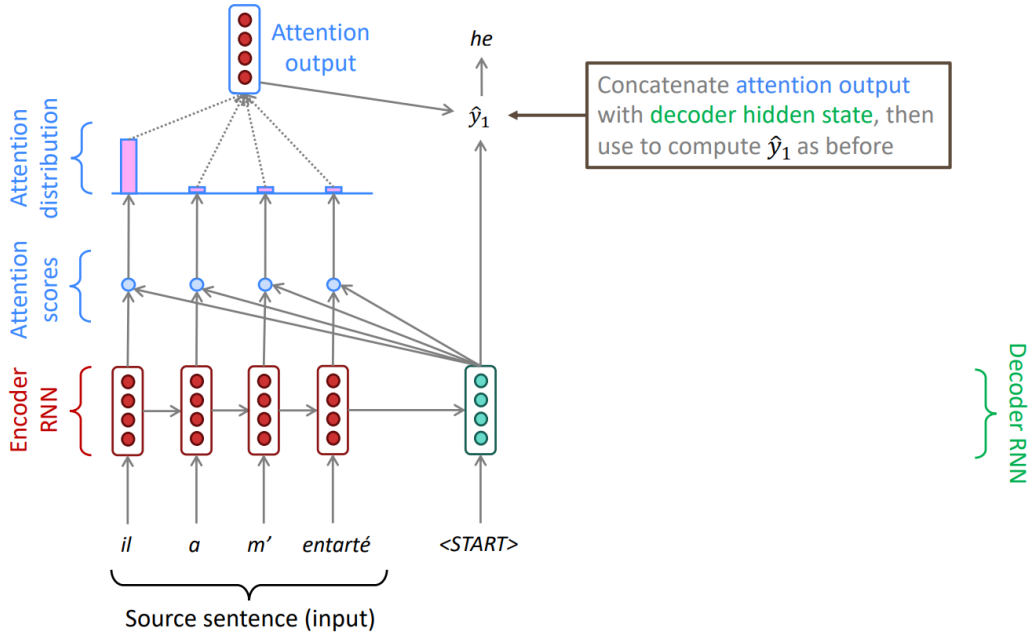
bottleneck problem

- encoder RNN의 마지막 hidden state에 모든 정보가 쏠리는 현상
- Sentiment Analysis task에서는 문장 전체를 보지 않아도 좋은 performance를 낼 수 있었지만 NMT task에서는 아님



⇒ 사람이 번역을 할 때도 source sentence를 확인하며 필요한 부분에 attention을 주는 아이디어가 제안됨

Attention



- decoder의 hidden state와 encoder의 각 단계에서의 hidden state를 내적
  ⇒
  Attention Score 계산
- 각각의 attention score에 `softmax()` 적용
  ⇒
  Attention Distribution 구하기

  → 번역할 때 어떤 단어에 초점을 맞춰야하는지 파악 가능
- attention distribution으로 encoder states를 가중 평균
  ⇒
  Attention Output
- [decoder hidden state; attention output]를 통해 단어 예측
  - source sentence로부터 더 많은 정보를 가져올 수 있고, 그 결과 더 좋은 번역이 가능함

1. RNN for NLP

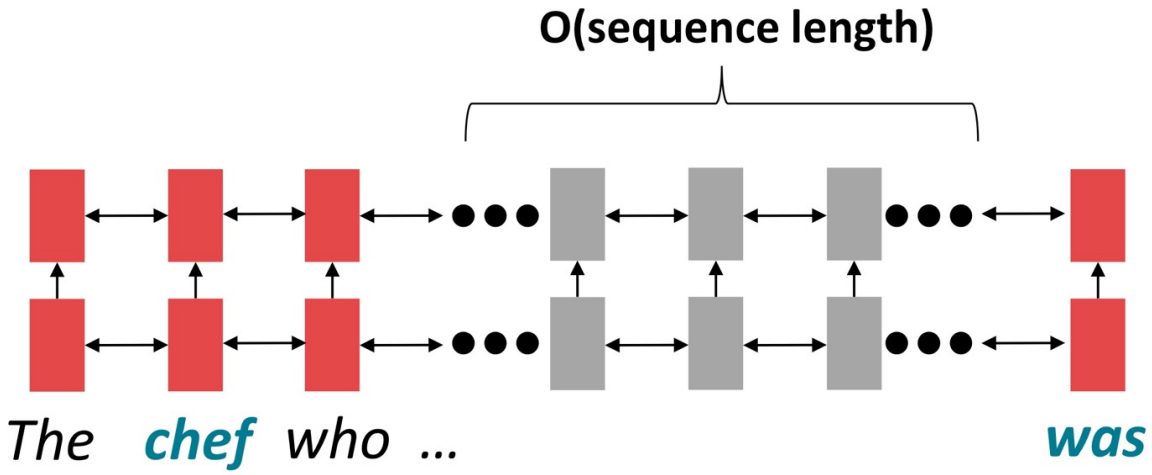
Circa 2016, nlp strategy

- bidirectional LSTM으로 문장을 encoding
- LSTM decoder
- attention을 이용해 메모리에 유연하게 접근하는 방법

⇒ 2021년, 모델에서 최적의 **building block**은 무엇일까?

RNN의 문제점

1. Linear interaction distance

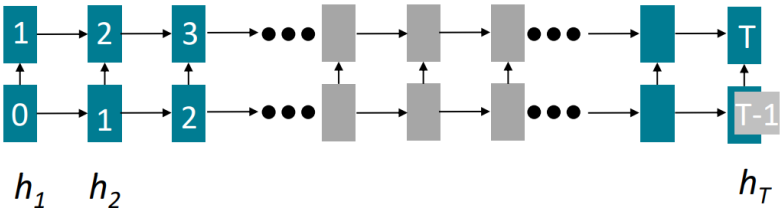


- gradient vanishing 문제

  → long-distance dependencies를 학습하기 어려움

- 선형 인접성을 인코딩하는데 적합하지 않음

2. Lack of parallelizability

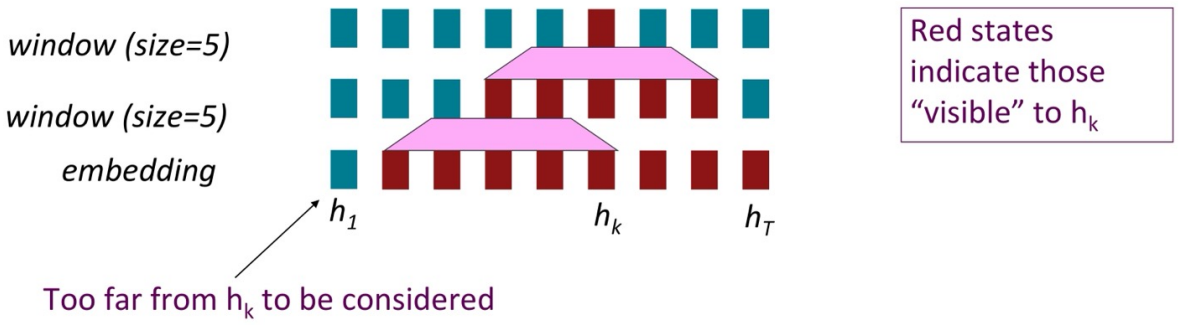


Numbers indicate min # of steps before a state can be computed

- 앞 과정을 계산한 후에 뒤 과정을 계산할 수 있음

  → GPU 병렬 작업을 활용하기 어려움

word window는?



- 여전히 문제를 해결해 주지 못함
- 시퀀스가 매우 길어지면 장거리의 문맥이 상실됨

2. Self-Attention

- Attention은 쿼리(queries), 키(keys), 값(values)으로 작동
  - **queries:** $q_1, q_2, ..., q_T, q_i \in R^d$
  - **keys:** $k_1, k_2, ..., k_T, k_i \in R^d$
  - **values:** $v_1, v_2, ..., v_T, v_i \in R^d$
- keys, queries, values는 같은 source에서 생성됨
  - 만약 이전 층의 출력이 $x_1, x_2, \cdot, x_T$(단어 당 벡터 하나)라면, $v_i = k_i = q_i = x_i$로 모두 동일한 벡터임

$$e_{ij} = q_i^\top k_j$$

Compute **key-query** affinities

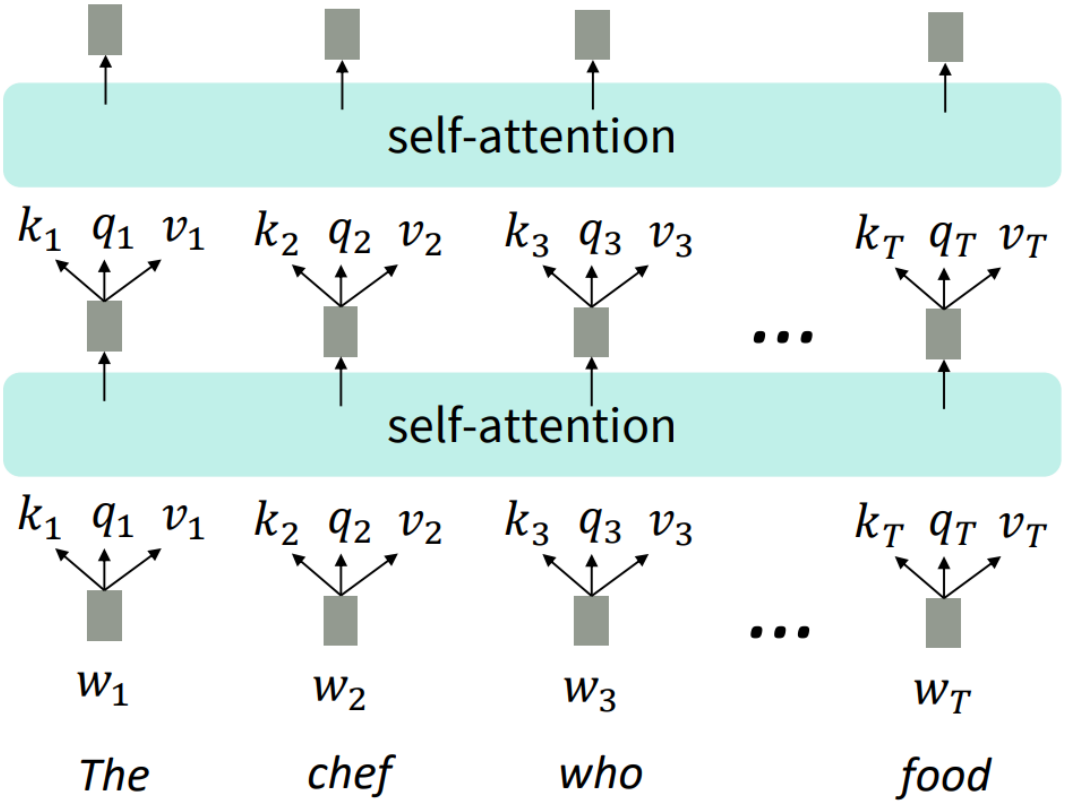
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

Compute attention weights from affinities (softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

self-Attention as an NLP building block



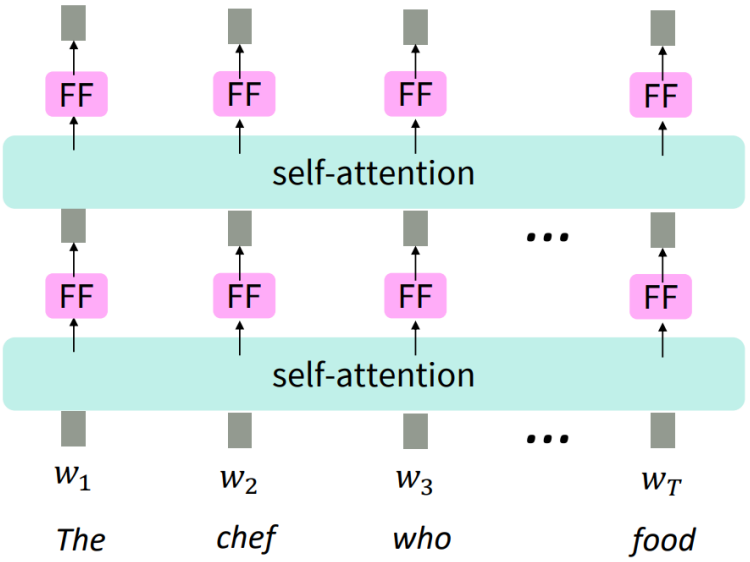
- LSTM을 쌓았던 것과 같이 self-attention block을 쌓을 수 있음
- 여전히 몇 가지 해결되어야 할 문제가 남아 있음

---

1. self-attention은 집합으로 연산되기 때문에 순서에 대한 정보가 누락되어 있음 ⇒ 위치 벡터의 필요성 제시
   - $p_i \in R^d, i \in 1, 2, ..., T$
   - $k_i = \tilde{k_i} + p_i$

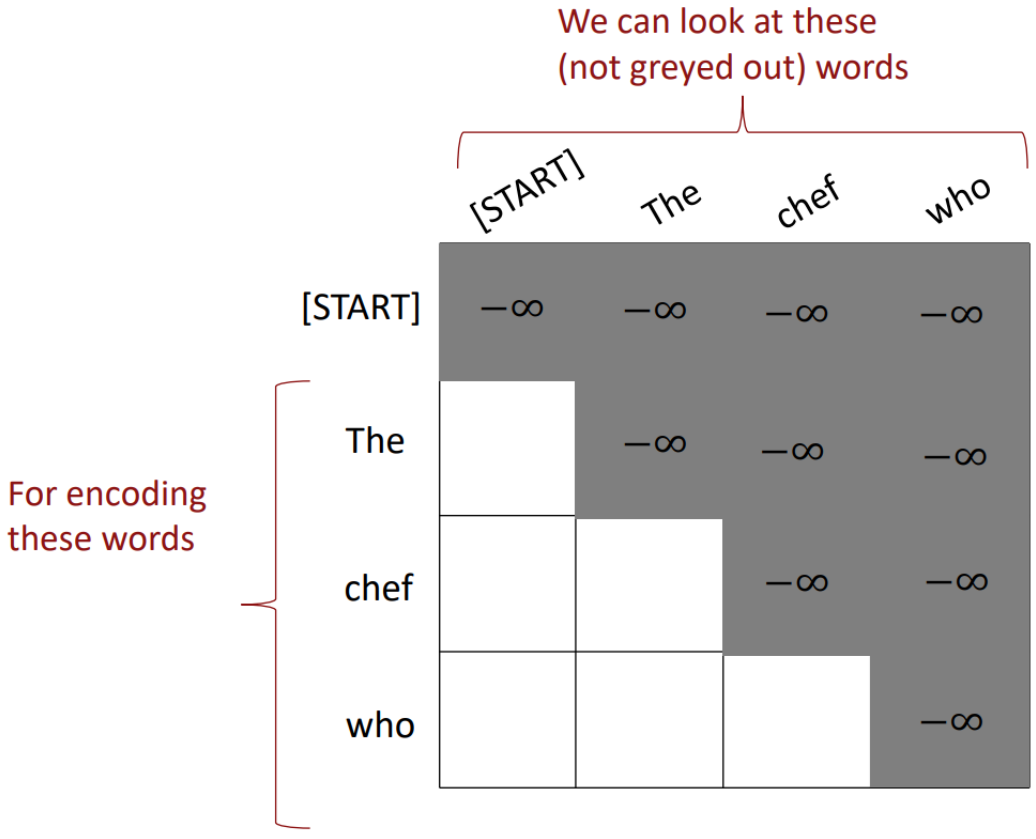
     $q_i = \tilde{q_i} + p_i$

     $v_i = \tilde{v_i} + p_i$
   - sinusoidal position representations: concatenate sinusoidal functions of varying periods
   - learned absolute position representation: 모든 p_i를 학습 가능한 매개변수로 놓기
2. elementwise 비선형성이 존재 x ⇒ 비선형성 추가



- 각 output vector에 feedforward network 추가

3. sequence 예측 시 미래는 알지 못한다고 가정
   ⇒ attention을 음의 무한대로 설정하여 미래 단어에 대한 attention 차단



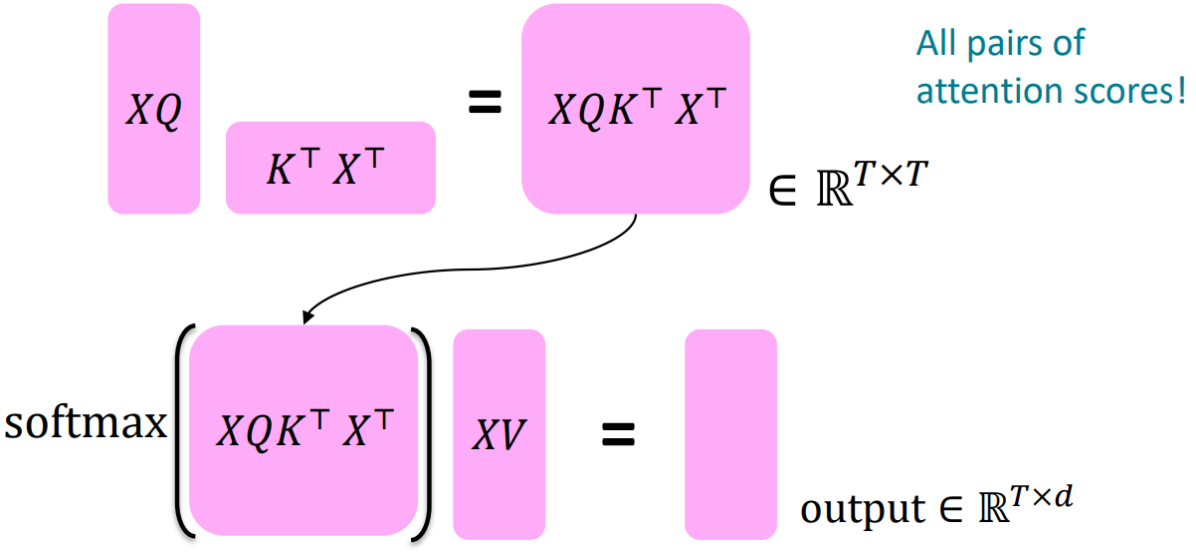
Self-attention Building Block의 구성 요소

- self attention
  - 가장 기본적인 요소
- 위치 표현
  - 자기 주의는 순서가 없는 함수 ⇒ 입력의 순서를 지정해 주는 역할
- 비선형성
  - 셀프 어텐션 블록의 출력에서 활용
  - 간단한 feed-forward 네트워크로 구현되는 경우가 많음
- 마스킹
  - 미래를 내다보지 않고 작업을 병렬화하기 위해서
  - 미래에 대한 정보가 과거로 "유출"되지 않도록 함

2. Transformer Model

Transformer Encoder: Key-Query-Value Attention

- concatenation of input vectors $X = [x_1, ..., x_T] \in R^{T \times d}$
  - **key matrix:** $k_i = Kx_i, K \in R^{d \times d}$
  - **query matrix:** $q_i = Qx_i, Q \in R^{d \times d}$
  - **value matrix:** $v_i = Vx_i, V \in R^{d \times d}$

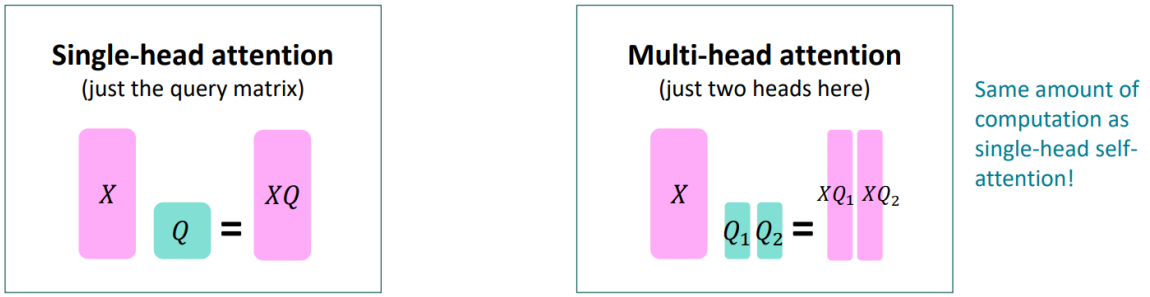


- $XK \in R^{T \times d}, XQ \in R^{T \times d}, XV \in R^{T \times d}$
- output = `softmax(` $XQ(XK)^T$ `)` $\times XV$

  ⇒ key, query로 softmax 구해서 value랑 weigthed sum

Transformer Encoder: Multi-headed Attention

- 문장 내 여러 곳을 동시에 보려면?
  - $output_l = softmax(XQ_l K_l^T X^T) * XV_l, output_l \in R^{d/h}$
  - $output = Y[output_1, .., output_h], Y \in R^{d \times d}$
  - 각 head는 각기 다른 것에 집중하며 서로 다른 벡터 생성



Transformer Encoder: Residual connections

- $X^{(i)} = X^{(i-1)} + \texttt{Layer}(X^{(i-1)})$
  - layer *i*가 layer *i*−1과 어떻게 달라야 하는지만 학습

⇒ gradient vanishing problem 완화

Transformer Encoder: Layer normalization

- hidden vector의 불필요한 정보 변동을 표준화를 통해 제거

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

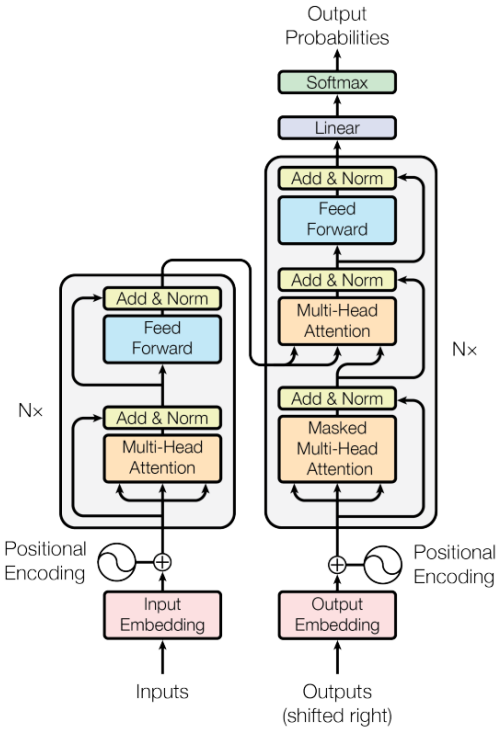
Transformer Encoder: Scaled Dot Product

- 차원 *d*가 증가하면 내적인 attention score 역시 커짐
- score가 커지면 softmax의 일부 값이 굉장히 커지게 되고, 낮은 확률을 지니고 있던 값들도 너무 작아져 기울기가 0으로 수렴하는 문제 발생

⇒ attention score을 $\sqrt{d/h}$로 나누어 줌

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

3. Transformer Encoder-Decoder



- 매 층마다 residual과 layer normalization 수행
- feed-forward는 attention 연산이 완료된 후에 적용
- encoder는 단순 multi-head attention
- decoder는 미래를 추론해야 하기 때문에 masked multi-head attention
- **cross-attention**: encoder의 출력값과 decoder의 출력값을 합쳐서 attention 수행

Transformer Decoder: Cross-attention

- **encoder vectors:** $H = [h_i, ..., h_T] \in R^{T \times d}$
- **decoder vectors:** $Z = [z_i, ..., z_T] \in R^{T \times d}$
- keys와 values는 encoder로부터 $k_i = Kh_i, v_i = Vh_i$
- quries는 decoder로부터 $q_i = Qz_i$

- output = `softmax(` $ZQ(HK)^T$ `)` × *HV*