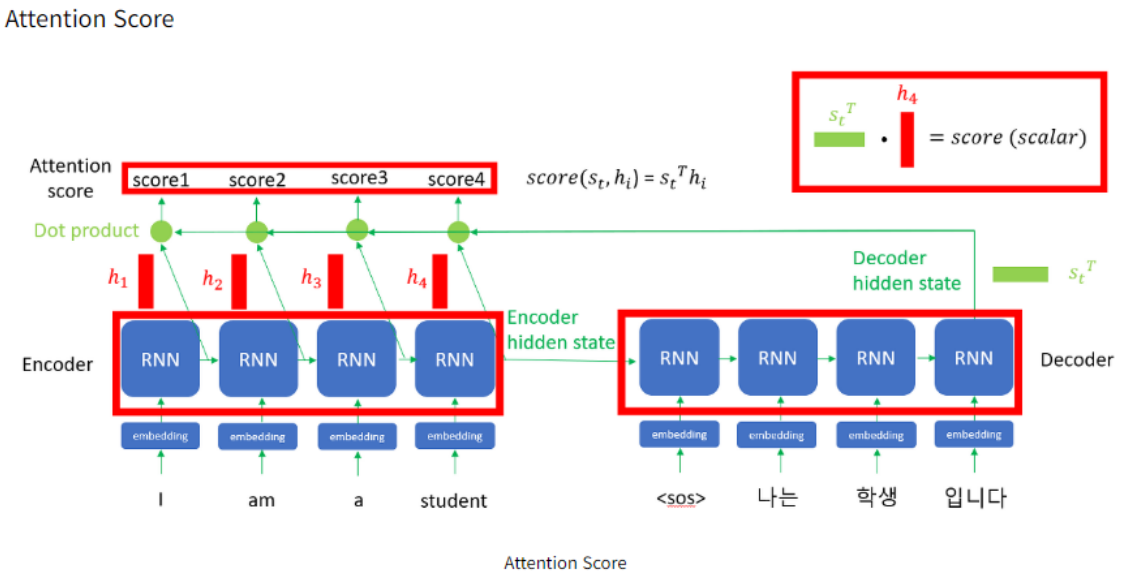
Attention, Self-attention

기존 Attention에서는 Encoder의 각 임베딩 토큰에 대해서 Decoder와 내적(Dot Product)하여 Attention Score를 구해야 했음



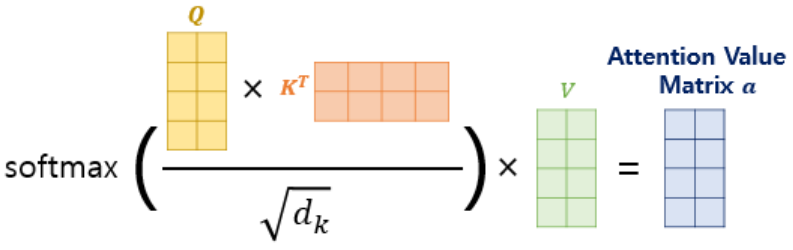
Attention Score

Decoder의 각 토큰에 대한 hidden state를 Encoder와 내적하여 스코어를 구하는 모습은 위와 같음. 즉, 그냥 Attention을 하기 위해서는 Encoder와 Decoder 간의 내적 연산이 필요함. (여러 RNN 모델에서 사용)

하지만, Transformer에서는 Self-Attention이라는 개념을 사용, 이를 위해서는 Query, Key, Value라는 3가지 변수가 필요함.

각 값은 같은 값을 통해 초기화 됨(그래서 Self), 하지만 각 Q,K,V에 대해 최적화를 위한 Weight를 갖는 W_q, W_k, W_v 값을 학습함.

세 값을 통해 Atttention Value(Score) a가 계산되고, 이를 구하는 공식은 다음과 같음. 각 차원이 너무 커질 수도 있기 때문에, key의 차원의 루트만큼 나눠지는 스케일링 작업이 진행되기도 함. → **Scaled dot-product Attention 연산이라고 부른다.**



Attention 구하는 공식

Query, Key, Value의 의미

하나의 문장 “I am a student”가 있다고 가정,

Attention 메커니즘을 사용하기 위해서는 각 Encoder가 단어 별로 임베딩 해주게 됨

만약 I의 임베딩이 [1,1,1,1]이라고 한다면,

I에 대한 Query, Key, Value의 original embedding이 모두 [1,1,1,1] 이라고 하면, 이는 각 weight인 W_q, W_k, W_b와 내적하여 I에 대한 Q,K,V를 생성한다.

$$Q_{I,origial} = [1, 1, 1, 1] \cdot W^Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = Q_I = [1, 1, 1, 1]$$

$$K_{I,origial} = [1, 1, 1, 1] \cdot W^K = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = K_I = [1, 0, 1, 1]$$

$$V_{I,origial} = [1, 1, 1, 1] \cdot W^V = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = V_I = [1, 1, 1, 0]$$

Q_I와 (K_I)^T를 내적하면 [1,1,1,1] * [1,0,1,1] = 1+0+1+1 = 3, Key의 차원은 3이므로

3/sqrt(4) = 1.5 즉, I에 대한 self-attetnion은 1.5인 것

당연히 자기 자신에 대한 Attention값이 가장 높다.

Q_I * K_am , Q_I * K_a , Q_I * Q_student가 추가적으로 계산하여

4*4 attention 행렬이 만들어진다.

(Encoder 단에서 스스로 Attention score를 생성하기 때문에 Self-attention)

⇒ 하나의 Q,K,V을 여러 개로 쪼개서 sclaed dot-product attention을 병렬적으로 처리하는 것이 **Multi-head attention** 이다. attention은 한번에 처리하는 것이 아니라 여러 개로 분할하여 처리한다. 마치 CNN에서 필터를 여러 개로 쓰는 것과 비슷한 효과를 만든다.

https://codingopera.tistory.com/43