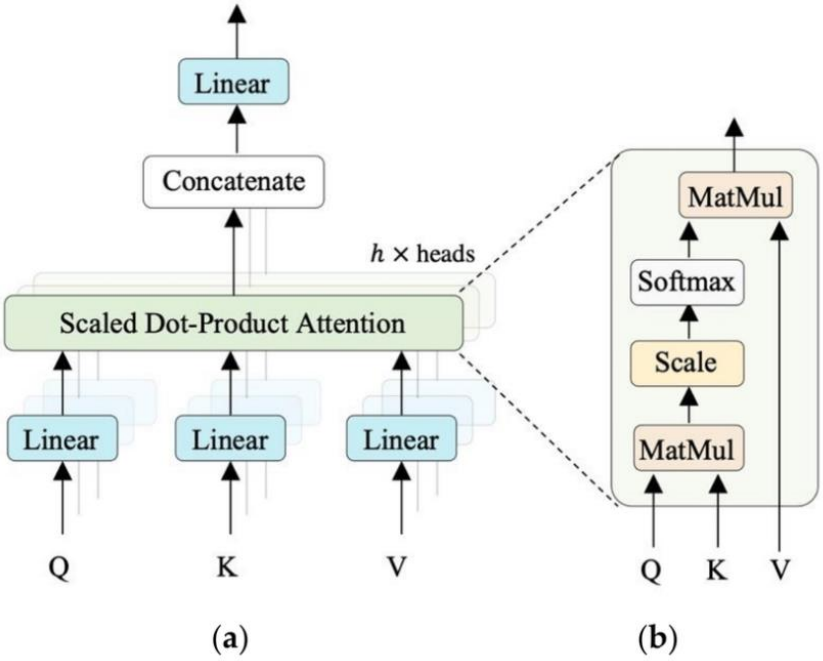
1. Attention Architecture



(a) Multi-head self-attention (b) Scaled dot-product attention

- Q = Query
- K = Keys
- V = Values

(a) Multi-head self-attention:

Multi-head self-attention은 입력 시퀀스의 다양한 부분에 동시에 주의를 기울이기 위해 입력을 여러 개의 헤드로 분할하여 처리하는 메커니즘이다. 각 헤드는 별도의 가중치 행렬을 사용하여 어텐션을 수행하고, 이후에 결과를 결합한다.

<과정>

1. 입력 벡터 생성:

입력 시퀀스로부터 쿼리(Query), 키(Keys), 값(Values) 벡터를 생성한다. 이는 선형 변환을 통해 얻어진다. 각 벡터는 입력 시퀀스의 각 위치에 대한 임베딩이다.

2. 헤드 분할:

입력 벡터를 여러 개의 헤드로 분할한다. 각 헤드는 독립적으로 어텐션 메커니즘을 수행

하며, 서로 다른 관점에서 입력에 주의를 기울인다.

3. 각 헤드에서 어텐션 계산:

각 헤드는 분할된 입력 벡터와 해당하는 가중치 행렬을 사용하여 어텐션을 계산한다. 이는 내적(dot product), 유클리드 거리(Euclidean distance), 코사인 유사도(cosine similarity) 등의 방법을 사용하여 수행된다.

4. 어텐션 값 결합:

각 헤드에서 계산된 어텐션 값을 결합한다. 일반적으로 각 헤드의 어텐션 값은 병렬로 계산되며, 이를 결합하여 하나의 어텐션 값으로 합친다. 이를 위해 각 헤드의 결과를 결합하는 방법 중 하나로 평균, 합 또는 연결(concatenation) 등의 방법이 사용될 수 있다.

5. 최종 출력 생성:

결합된 어텐션 값으로부터 최종 출력을 생성한다. 이는 다시 한번 선형 변환을 통해 적절한 차원으로 변환될 수 있다.

(b) Scaled dot-product attention

Scaled dot-product attention은 쿼리와 키 간의 내적을 통해 어텐션 가중치를 계산하는 간단하고 효율적인 메커니즘이다. 내적 결과는 키 벡터의 차원 수의 제곱근으로 스케일링되어 안정성을 높인다.

<과정>

1. 입력 벡터 생성:

입력 시퀀스로부터 쿼리(Query), 키(Keys), 값(Values) 벡터를 생성한다. 이는 선형 변환을 통해 얻어진다.

2. 내적 계산:

쿼리와 키 간의 내적을 계산합니다. 내적은 각 쿼리-키 쌍에 대해 수행되며, 이는 쿼리와 키 간의 유사도를 나타낸다.

3. 스케일링:

내적 결과를 키 벡터의 차원 수의 제곱근으로 나누어 스케일한다. 이렇게 함으로써 내적 결과의 크기를 조절하여 안정성을 높인다.

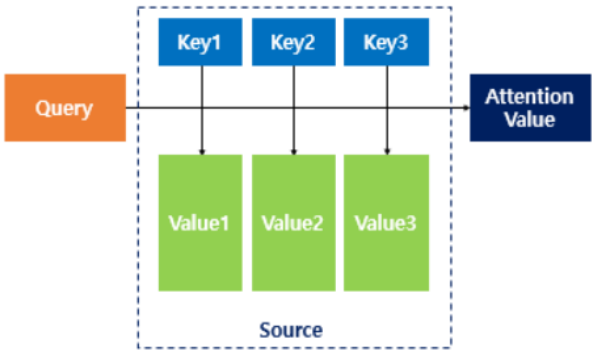
4. 소프트맥스 함수:

스케일링된 내적 결과를 소프트맥스 함수(softmax function)를 통해 어텐션 가중치로 변환한다. 소프트맥스 함수는 각 키에 대한 쿼리의 관심도를 나타내는 확률 분포를 생성한다.

5. 가중 평균:

어텐션 가중치를 사용하여 값 벡터의 가중 평균을 계산한다. 각 값에 대해 해당하는 어텐션 가중치를 곱하여 가중 평균을 계산하고, 이를 통해 출력을 생성한다.

(1) Attention 함수



-> Attention(Q, K, V) = Attention Value

Attention 함수는 주어진 '쿼리(Query)'에 대해 모든 '키(Key)'의 유사도를 각각 구한다. 그리고, 이 유사도를 키(Key)와 매핑 되어 있는 각각의 '값(Value)'에 반영한다. 그리고, 유사도가 반영된 값(Value)을 모두 더해서 리턴하고, Attention 값(Attent value)를 반환한다.

Attention 함수 표현 방식:

$$\text{Attention}(Q,K,V)=\text{Attention Value}$$

함수의 주요 단계:

1. 유사도 계산(Similarity Calculation):

주어진 쿼리 Q와 키 K 간의 유사도를 계산한다. 이는 내적(dot product), 유클리드 거리(Euclidean distance), 코사인 유사도(cosine similarity) 등의 방법을 사용할 수 있다.

2. 어텐션 가중치 계산(Attention Weight Calculation):

계산된 유사도를 소프트맥스 함수(softmax function)를 사용하여 확률 분포로 변환한다. 이를 통해 각 키(Key)에 대한 쿼리(Query)의 관심도를 나타내는 어텐션 가중치를 얻을 수 있다.

3. 가중 평균(Weighted Average):

어텐션 가중치와 값(Value)을 곱하여 가중 평균을 계산한다. 이를 통해 주어진 쿼리에 대한 어텐션 값(Attention Value)을 얻을 수 있습니다. 즉, 각 값에 해당하는 어텐션 가중치를 곱하고 그 결과를 합산하여 어텐션 값을 얻는다.

4. Attention 값 반환(Return Attention Value):

계산된 어텐션 값(Attention Value)을 반환한다. 이 값은 입력에 대한 어템션 메커니즘을 통해 생성된 정보를 나타낸다.

2. Attention Mechanism

- Attention 간단 이해

단순히 이 문장만을 본다고 가정했을 때 우리의 인지 시스템은 '먹었어'라는 동사와 무엇을 먹었는지에 대한 명사 여기서는 '치킨'에 일반적으로 주의를 가질 것으로 쉽게 예상 가능하다. 이와 반대로 '먹었어'라는 동사와 어떤 치킨 인지를 나타내는 '교촌'이라는 단어는 '치킨'에 비해 상대적으로 덜 집중할 것이다.

→ 디코더(decoder)에서 출력 단어를 예측하는 매 시점(time stemp)마다, 인코더(encorder)에서 전체 입력 문장을 다시 한번 참고한다는 점이다. 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 Attention해서 보게 된다.

1. Dot-Product Attention

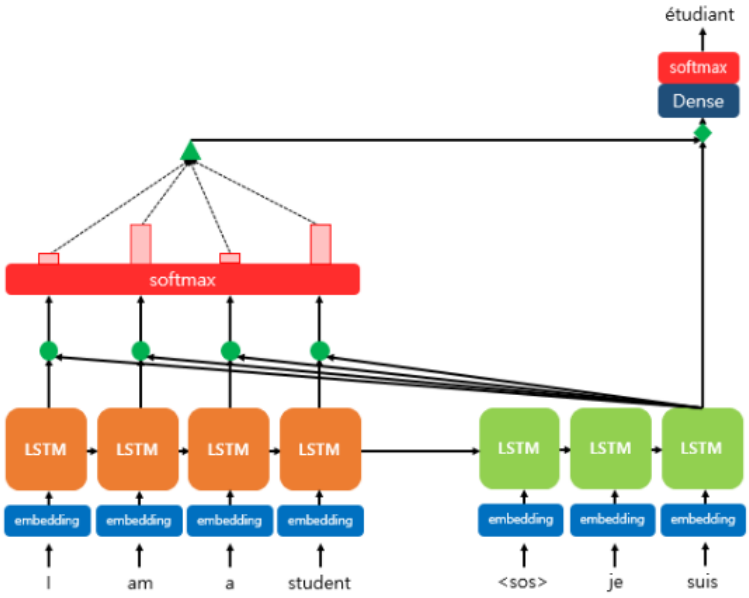
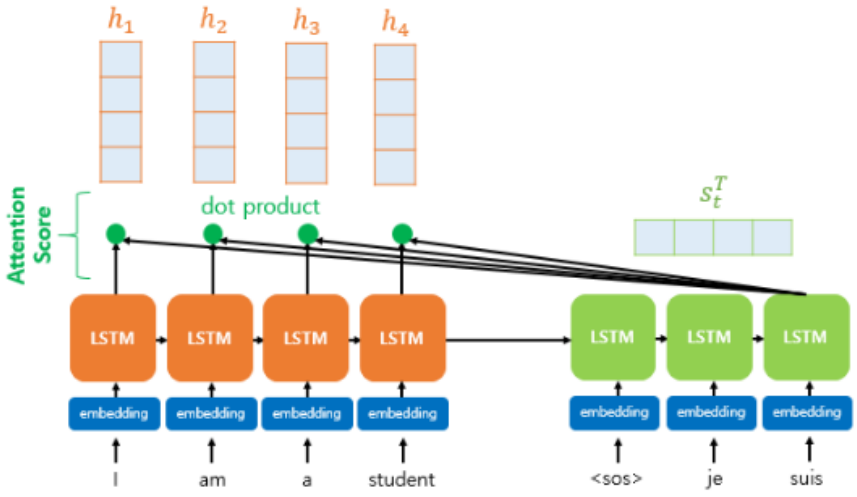


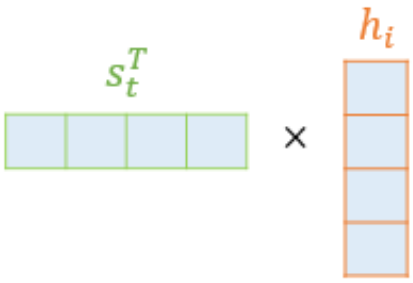
그림. Decoder의 세 번째 RNN 셀에서 출력 단어를 예측할 때, Attention Mechanism을 사용하는 모습



- h_t: t(time step)시점에서 encoder의 은닉 상태(hidden state)
- s_t: t(time step)시점에서 decoder의 은닉 상태(hidden state)
- Encoder와 Decoder의 차원이 같다고 가정한다.
- 시점 t에서 출력 단어를 예측하기 위해서는 decoder의 cell은 두 개의 입력 값을 필요로 하는데, 바로 step의 hidden state인 $s_{t_{-1}}$ 와 이전 step의 output $\hat{y}_{t_{-1}}$에 나온 출력 단어이다.
- Attention mechanism에서는 출력 단어 예측에 또 다른 값을 필요로 하는데 바로 어텐션 값(Attention Value)이라는 새로운 값이다. a_t 이라 정의한다.

1) Attention Score 구하기

- Attention Score란 현재 decoder 시점 t에서 단어를 예측하기 위해, decoder의 hidden state s_t 와 encoder의 hidden state $h_1 \sim h_n$들이 얼마나 유사한지를 판단하는 점수이다.



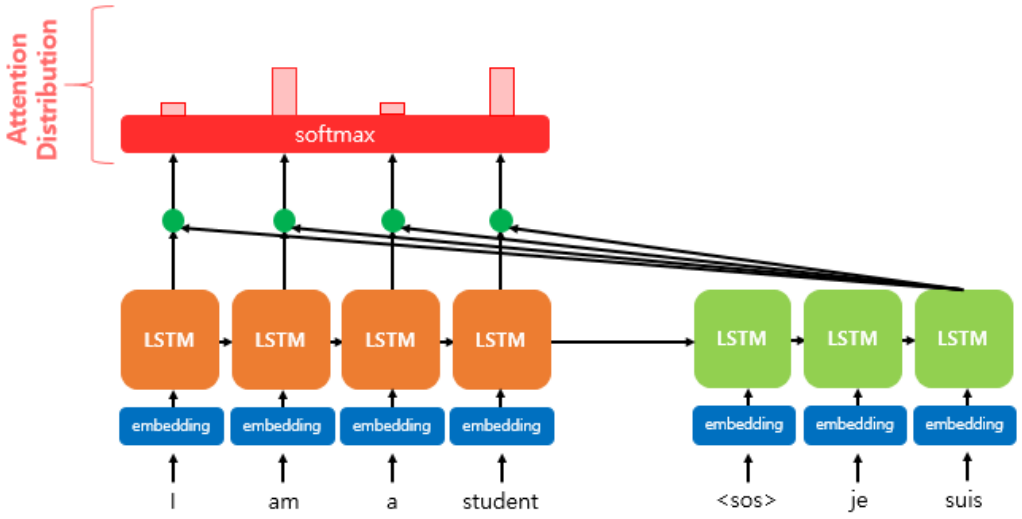
- $score(s_t, h_i) = s_t^T * h_i$, 결과 값은 scalar
- s_t 와 encoder의 모든 hidden state의 attention score의 모음 값을 e^t라 정의한다.

$$e^t = [s_t^T h_1, \ldots, s_t^T h_n]$$

2) Attention Distribution 구하기

- Attention scores e^t에 softmax(소프트맥스)함수를 적용해, 모든 값의 합이 1이 되는 확률 분포 Attention Distribution을 얻는다. 각각의 값은 Attention Weight(어텐션 가중치)이라 한다.
- Ex). 입력 squence인 ['I', 'am', 'a', 'student']에 대응하는 hidden states를 활용해 Attention scores를 구하고, 이로부터 Attention Distribution을 구하게 되면 [0.1, 0.4, 0.1, 0.4]의 형태를 갖는 벡터를 얻게 된다.

  ➔ 이 때 각각의 값을 **Attention Weight(어텐션 가중치)**이라 한다.

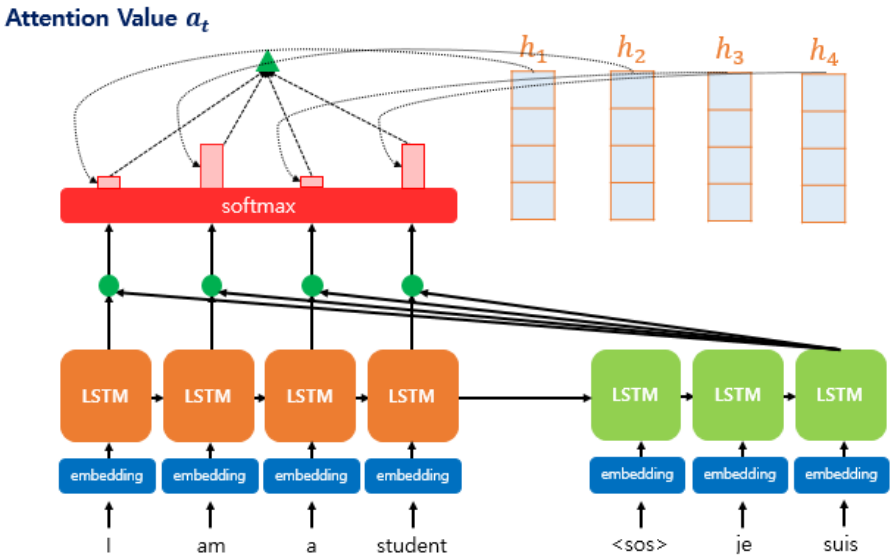


$a^t = softmax(e^t)$ - $a_t(attention\ \ value) \quad \neq \quad a^t$

➔ Decoder의 시점 t에서의 attention weight의 모음 값인 어텐션 분포를 a^t 이라 한다.

3) 인코더와 각 Attention Weight와 그에 대응하는 hidden state를 가중합하여 Attention Values를 구한다.

- Attention Weight와 각 hidden state를 통해 최종적인 Attention value a_t 를 얻는다.

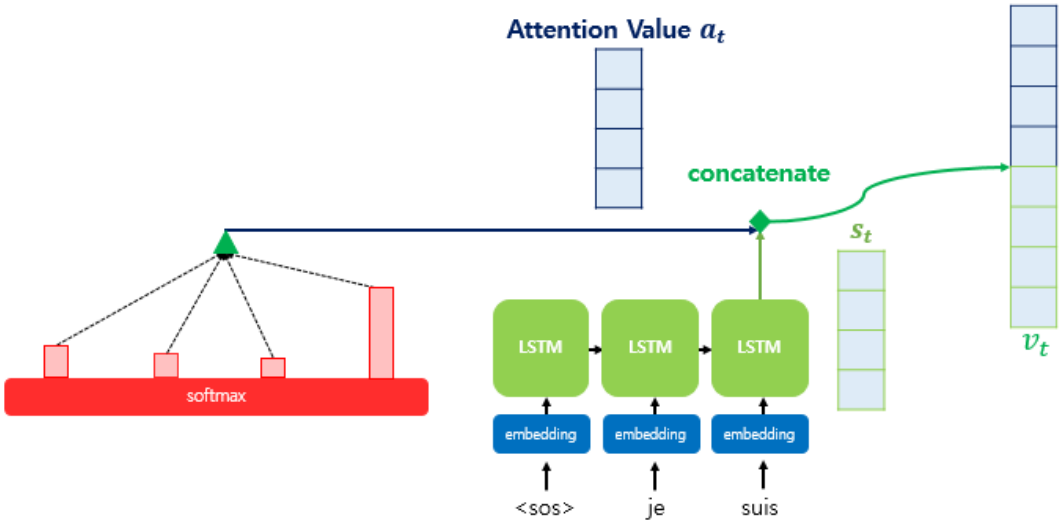


$$a_t = \sum_{i=1}^{N} a_i^t h_i$$

- 이러한 Attention value a_t는 인코더의 맥락을 포함하고 있기 때문에 **Context Vector**(맥락 벡터)라고 불린다.

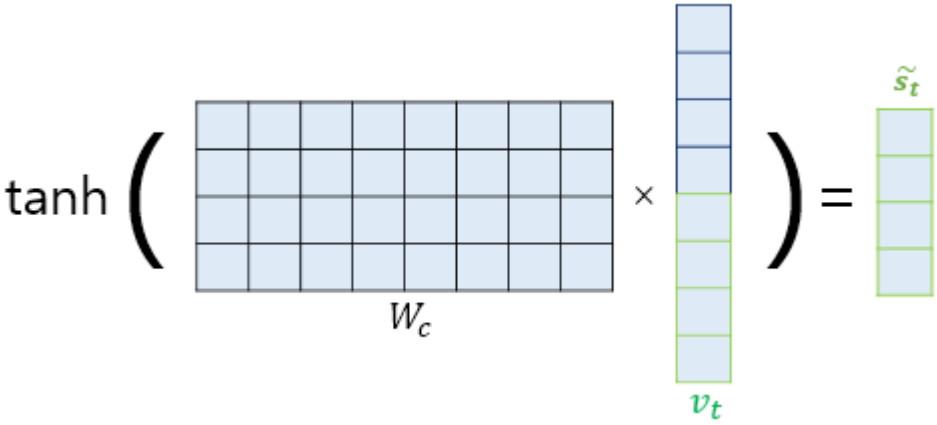
➔ 정확히는, decoder 내 time step t의 context vector

4) Attention value a_t와 decoder의 t 시점의 hidden state를 연결(concatenate)한다.



- a_t를 s_t와 결합(concatenate)하여 하나의 벡터로 만드는 작업을 수행한다. 이를 v_t라고 정의한다. v_t를 $\hat{y}$ 예측 연산의 입력으로 사용함으로써 encoder로부터 얻은 정보를 활용하여 $\hat{y}$를 좀 더 잘 예측할 수 있게 된다. 이것이 Attention Mechanism의 핵심이다.

5) 출력층 연산의 input이 되는 $\tilde{s}_t$를 계산한다.



$$\tilde{s}_t = \tanh(W_c[a_t; s_t] + b_c)$$

- W_c = 학습 가능한 가중치 행렬
- b_c = 편향
- $v_t = [a_t \ ; s_t]$
- ; = concat( )
- v_t를 바로 출력층으로 보내기 전에 신경망 연산을 한 번 더 추가하였다. 가중치 행렬과 곱한 후에 하이퍼볼릭탄젠트(Hyperbolic Tangent, Tanh) 함수를 지나도록 하여 출력층 연산을 위한 새로운 벡터인 $\tilde{s}_t$를 얻는다.

6) 최종적인 예측 $\hat{y}_t$를 얻는다.

$$\hat{y}_t = Softmax(W_y \tilde{s}_t + b_y)$$

+ 다른 Attention 방법

이름	스코어 함수	Defined by
dot	$score(s_t,\ h_i) = s_t^T h_i$	Luong et al. (2015)
$scaled\ dot$	$score(s_t,\ h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
$general$	$score(s_t,\ h_i) = s_t^T W_a h_i$ // 단, W_a는 학습 가능한 가중치 행렬	Luong et al. (2015)
$concat$	$score(s_t,\ h_i) = W_a^T\ tanh(W_b[s_t; h_i]) score(s_t,\ h_i) = W_a^T\ tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
$location-base$	$\alpha_t = softmax(W_a s_t)$ // α_t 산출 시에 s_t만 사용하는 방법.	Luong et al. (2015)