

2. Seq2Seq

Decoder - Encoder를 이용해 외부한 Seq2Seq 풍부

· 인코더 + 디코더로 구성되어 있음.

- 인코더 : 양방향의 모든 단어를 순차적으로 입력받은 뒤에 RNN이 그 모든 단어 정보들을 인코딩해 하나의 벡터로. = context vector

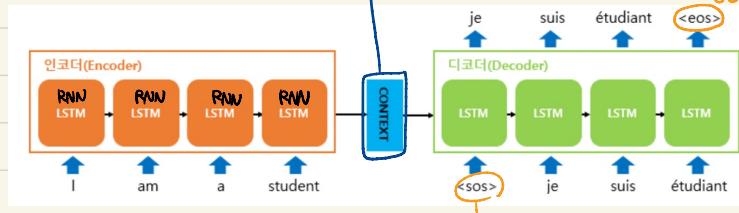
· 디코더 : context vector를 받아서 번역된 단어를 차례대로 순차적으로 출력

· 인코더/디코더가 같은 RNN 아키텍처.

→ [context vector]

: 언어가 입력받았을 때 단어를 순차적으로 입력받은 뒤, 마지막에 모든 단어 정보를
인코딩해 하나의 벡터로 만들었!

문장의 첫 번째 의미하는 symbol



문장의 마지막 의미하는 symbol

· 정리

→ 입력받을 단어 token화 → 각 token은 각 RNN 셀의 time-step의 양자화된 점자

RNN 셀의 양자화된 점자를 다음 RNN 셀의 초기상태로 사용할 수 있다. ⇒ context vector!

→ CASE RNN 셀의 초기상태로 사용할 수 있다.

· 디코더

→ 디코더 <SOS>가 입력되면 디코더는 동일한 확률의 높은 점자를 예측, 예측한 점자를 다음 RNN의 입력으로!

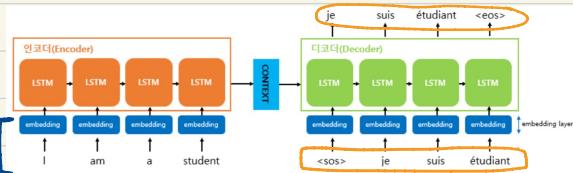
→ 예외적인 경우, <SOS>가 들어온다. "je"를 예측 → 다음 셀의 RNN 양자화하고 ...

→ <EOS>가 예측되었을 때 제보장.

test sample test.

· Seq2Seq는 training과 test 과정의 구조를 다룬다.

· training algorithm은 context vector를 입력받아서 teacher-forcing로 학습을 진행한다.



교사-강요
(teacher-forcing)

→ Seq2Seq의 내용에는 모든 단어들은 워드 임베딩을 통해 임베딩 벡터로 표현된 형태로 있다.

QnA! 워드 임베딩이란?

→ 베이지안 계층을 통해 추출한 결과로 헛정 X

→ 사용자가 설정한 값으로 모든 단어의 벡터 표현의 차원을 맞춘다.

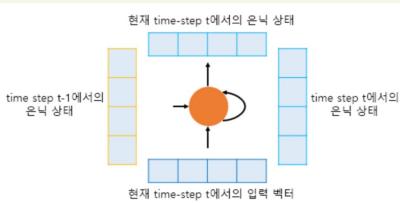
→ 0과 1 X → 부정감!

-	0.157	0.78	0.75	0.88
	-0.25	0.29	-0.81	-0.17
	0.478	-0.96	0.96	0.29
	-0.78	0.52	0.12	0.48

→ 워드 임베딩한 결과

: 사이즈를 4로 가정하였지만, 보통 임베딩 벡터의 차원은 수백

[RNN Model 구조]



현재 time-step t에서의 은닉 상태

→ RNN 셀은 t-1에서의 은닉 상태와 t번의 입력값을 입력으로 ...

이 둘로 은닉상태를 만든다.

→ 이 t번의 은닉상태는 바로 위의 또 다른 은닉층이나 출력층에 있는 경로에는 의의를 끊거나, 필요없으면 값을 무시하는 0

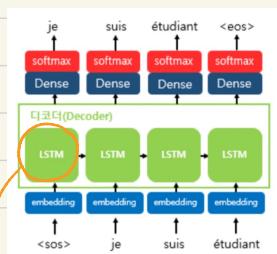
→ RNN 셀은 t-1번의 은닉상태를 t+1번의 RNN 셀의 입력으로 사용해.

↳ 초기시점의 동일한 RNN 셀에서의 모든 은닉상태의 값들이 누적해서 받아온 값.

QnA! context vector ⇒ 인구학자와 마지막 RNN 셀의 은닉상태값 = 임遐용장의 모든 단어 token들의 정보를 모아놓고 있는.

↳ 디코더의 첫번째 은닉상태에 값으로 사용

[다중문]



- 결과 단어를 나올 수 있는 것들을 다양한 단어들...! → 그 중 어떤 단어를 선택하는가? $\rightarrow \text{softmax}$
- 다중문의 각 시장의 RNN hidden state 출력 벡터가 나오면 $\rightarrow \text{softmax}$ 를 통해 **특정 sequence의 각 단어별 확률값 반환** \rightarrow 확률적 확장

주의 RNN 를 context vector 와 함께 $<\text{EOS}>$ 로 다음 단어를 예측함.

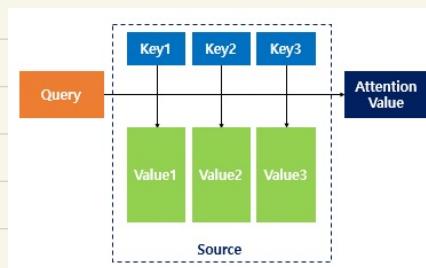
→ 예측한 단어는 다음 RNN의 입력이 됨.

3. attention

- 디코더 OH 사용자 인터랙션의 전개 방식을 다양한 종류로 한다는 거!

Key : Value \rightarrow 디버깅 같은

운영체계 창고 X \rightarrow 해당 사용자 이용해야 할 단어와 연관이 있는 부분을 좀 더 강조해라!



Attention 계산

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

Q: Query = 디코더 모델의 원래 상태

K: keys = 모든 사용자 인터랙션 단어의 원래 상태들

V: values = 모든 사용자 인터랙션 단어 원래 상태들

→ Query를 통해 모든 key에对自己的 유사도를 구한다

→ 구한 유사도를 key에 mapping된 해당 value에 반영해줌

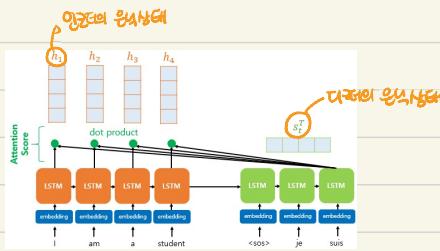
→ 그 value를 **return** = attention value.

[Attention 이해하기]

Dot-Product Attention을 통해 이해해보자!!



1) Attention 스코어 구현하기.



원래 인코더-디코더와 동일한 단어별 셀프-인코딩: 키의 원상 상태 + 키의 출력 단어
- 어떤 단어에 대해서도!
=> 같은 단어에 Attention Value 값도 필요없는
=> 키와 키를 예측하기 위한 Attention = 0.5 를 정의.

Q. Att를 어떻게 구현하나...

A. ① Attention score를 구현하기.

→ 같은 단어의 키와 키의 값을 예측하기 위해, 인코더의 모든 원상태 각각이 디코더의 첫 번째의 원상태인 S_t 와 얼마나 유사한지 판별하는 드립.

→ dot-product attention이라는 이름을 부르기 위함

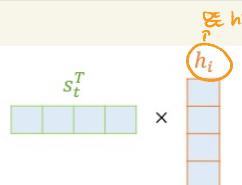
: S_t 를 차례대로 각각 h_i 와 비교 수행!

⇒ 각 attention score 값을 드립자!

→ score(S_t, h_i) = $S_t^T h_i$

→ e_t : S_t 와 인코더의 원상태의 어떤 단어의 스코어의 합계

$e_t = [S_t^T h_1, \dots, S_t^T h_n]$



2) Softmax 흐름을 통해 Attention 분포를 구한다

: e^t_i 는 softmax 계산식에서 학습률을 입력하는 것

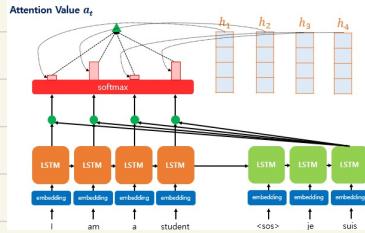
Attention distribution
(Attention 분포)

: 각각의 값 = attention weight (attention 가중치)

: 모델은 기존의 모델과의 차이점은 α_t^t

$$\alpha_t^t = \text{softmax}(e^t)$$

3) 각각의 Attention weight에 의해 상태를 가중평균하여 Attention Value를 구한다.



=> 각각의 기울임을 높여주고 합하는 단계

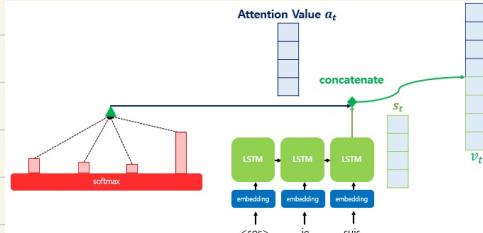
: 기울임을 높여주는 원인은 각각 공유하고 있는 단계 == 기울임

: attention value α_t

$$\alpha_t = \sum_{i=1}^n \alpha_i^t h_i$$

= context vector \rightarrow 결과적으로는 인코더와 디코더 원래는 동일한 단계
= summarizing α_t context vector

4) 차원별 값과 context + hidden 을 통해 상태를 업데이트한다 (concatenate)



: 차원별 값과 결합된다,

α_t 를 Seek concatenate 단계 하나의 단계로 만드는 v_t

$\rightarrow v_t$ 를 \hat{y} 예측의 영상의 양쪽으로 사용한다.