<https://jeongukjae.github.io/posts/cs224n-lecture-8-machine-translation,-seq2seq,-attention/>  
<https://gnoej671.tistory.com/7?category=1034944>

**new task: Machine Translation -> using new architecture: sequence-to-sequence -> is improve by attention**

**Machine Translation**

Source language로부터 targe language로 sentence x를 sentence y로 번역하는 task!

Pre-machine Translation: rule-based = 예외 많고 최적 x (bilingual dictionary, map Russian to English)를 이용 (1950)

Statistical Machine Translation: probabilistic model (from data)를 이용 / given sentence x 일 때 -> best sentence y를 찾는다.  = P(y|x)를 최대로 하는 y를 찾자..!

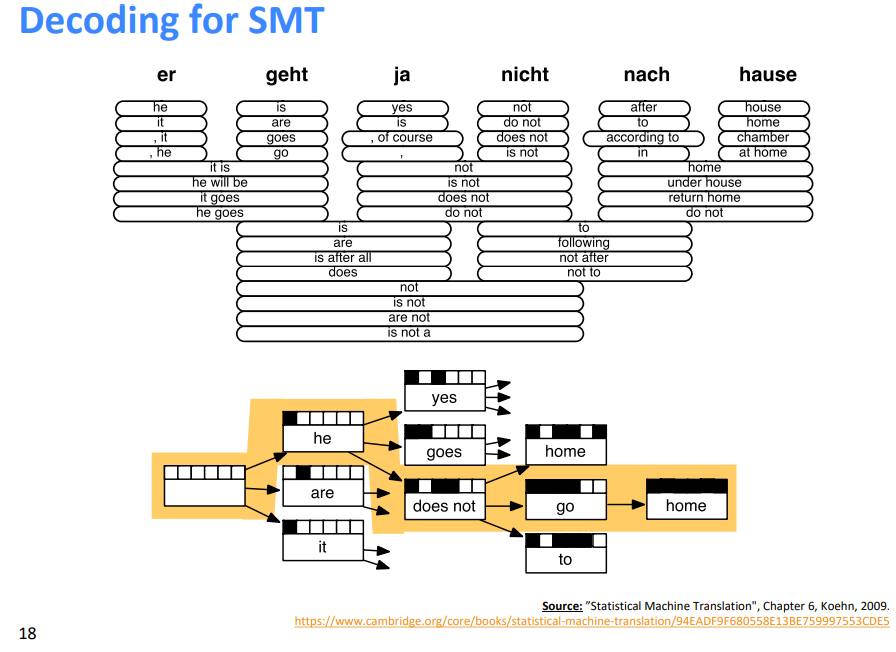
베이지안 공식으로 저 확률을 쪼갠다. => P(x|y)\*P(y)/P(x) (여기서 분모는 사실 생략)  
P(y)는 Language Model(how to write good English, Learnt from monolingual data)  
P(x|y)는 Translation Model(how words and phrases should be translated, Learnt from parallel data)

P(x|y)를 학습하는 것이 목표 -> but 많은 양의 parallel data 필요(pairs of human-translated French/English Sentences)

또한, parallel corpus에서 학습 시키려면, P(x|y)를 P(x, a|y)로 쪼개야함.  
a는 alignment(가지런함, 지지??)로 word-level correspondence between sentences x, y임(French, English)  
하지만 몇몇 단어는 상응되는 counterpart가 없을 수 있다.(spurious)  
또한, 한 단어가 여러 의미를 가질 수 있다.(many to one, one to many, many to many -> complex하다)

즉, SMT를 하기 위해서는 특정 단어에 counterpart가 같은 위치(어순에 따라 위치가 다를 수 있음)에 있어야 하고 / fertility의 경우의 수도 고려해 combination을 계산해야 한다.

* Every possible y and calculate probability = too expensive
* 너무 비싸니까 제일 높은 확률의 sentence 및 결과만 이용 (나머지 경우는 버림) = heuristic search algorithm = 이 과정은 decoding 이라 한다.



많은 경우의 수를 통해 다 확률 따져가며 계산한다. -> system이 점차 complex 해짐  
또한 feature를 매우 세밀히 디자인 해야함.  
또한 많은 compiling and maintaining extra resources  
또한 human effort to maintain 증가

**Neural Machine Translation**

Single neural network로 MT를 한다. / two RNN을 포함한 seq2seq라고 불린다.

BLEU Metric Paper

[Sequence to Sequence Learning with Neural Networks](https://arxiv.org/pdf/1409.3215.pdf)  paper

[Neural Machine Translation by Jointly Learning to Align and Translate](https://arxiv.org/pdf/1409.0473.pdf)  paper (attention)

<https://blog.naver.com/sooftware/221809101199>

LSTM과 ResNet의 Skip Connection / addition gate의 local gradient 전달에 대해 확인

DNN -> 가변길이 다루기 힘듬 -> RNN -> 가변길이 다룰 수 있음.  
하지만 일반적인 RNN은 input sentence와 output sentence의 길이가 다를 때 다루기 힘듬.

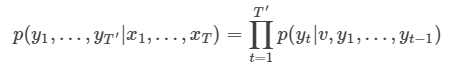
따라서 input sequence를 RNN을 통해 고정된 길이의 vector로 만들고 그 농축된 vector를 통해 target sequence를 구할 수 있다. (따라서 두 RNN이 필요하다.)

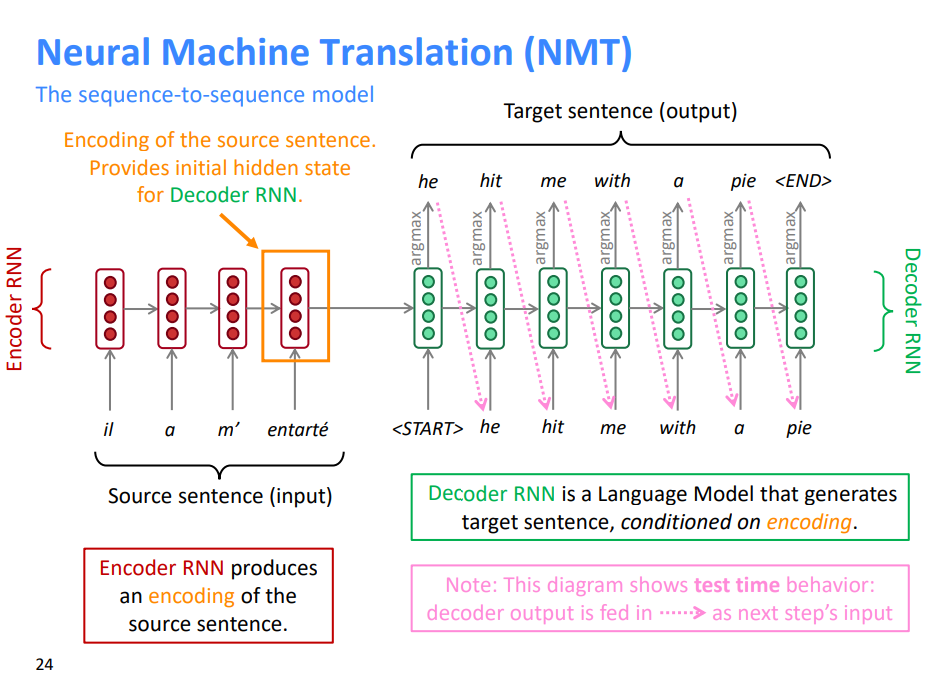
또한 Long-term dependency에 강한 LSTM을 RNN의 업그레이드 버전으로 사용하게 된다.

따라서 우리의 최종 목표는  이며, input sequence가 주어졌을 때 output sequence의 조건부 확률을 구하는 것

가장 처음으로 할 것은 encoder 작업인데 고정된 길이의 vector representation v를 먼저 구하는 것이다. (v = 첫 RNN의 마지막 hidden state 값)

Note-6에서 봤듯이 다음과 같은 식을 찾을 수 있다. 이 확률을 높여야 한다. 또한 가장 높을 때의 문장을 결과로 가져온다.

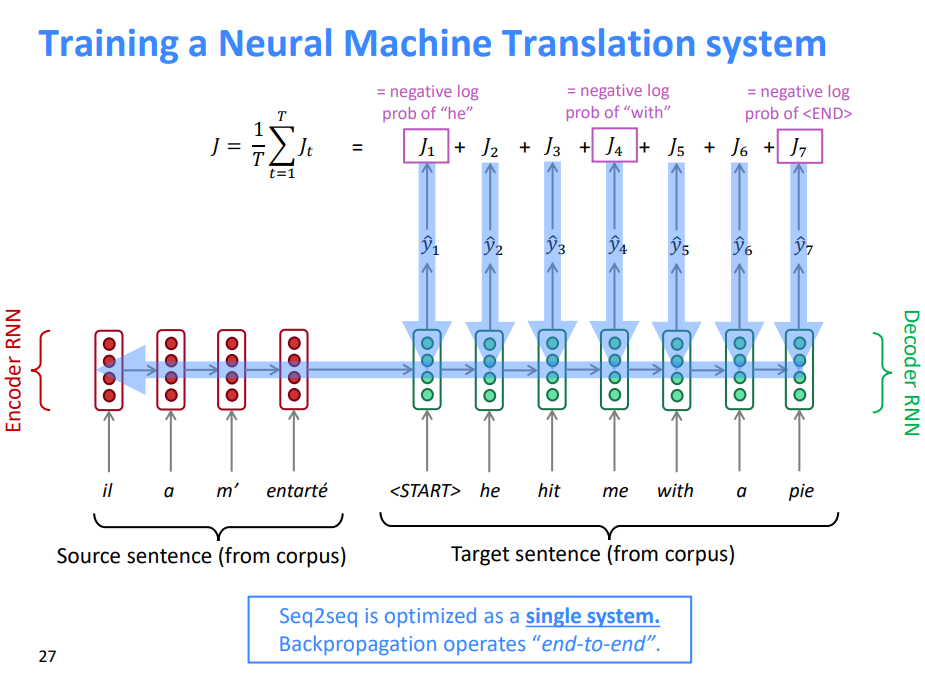




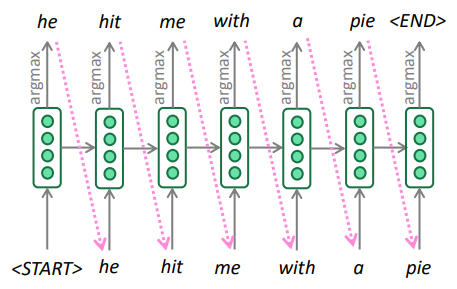
일반적 MT 보다 유용하며 / Summarization(long text->short text), Dialogue(previous utterance -> netx utterance<발언>), Parsing(text-> parse as sequence, 문장들로 쪼개기?), Code Generation(natural language -> python code) 등에 쓰임

Seq2seq는 Conditional Language Model 이다. (input sequence를 조건으로 하여 target sentence y의 다음 단어를 예측 = Language Model 이기 때문(N-gram Model 처럼))

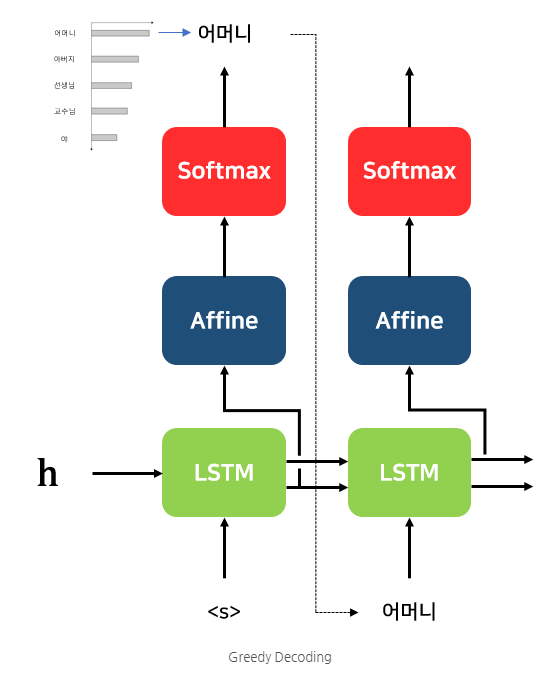
또한 backpropagation 시 다음과 같은 end-to-end system = single system



Greedy Decoding = best word를 예측하여 그것을 이용한다. / can’t undo

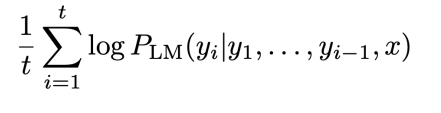


만약 hit -> me 대신 hit -> a로 예측하고 그거를 다시 전해준다면? 이후 연쇄적으로 틀릴 가능성이 높아짐.(시간복잡도는 상수라 보면된다.)

어머니가 정답이 아닐수도 있다. => 2등이 정답인 경우 여기에도 고려할 여지는 필요하다. 그렇다고 모든 경우의 수를 고려하기도 힘들다.(시간복잡도 너무 빡셈)

따라서 선택 시에 Beam Search를 Greedy Decoding & 모든 경우의 수(exhaustive search) 고려 사이에서 이용하게 된다. – 유망한 빔의 개수(K개)에서 골라서 진행하는 방식 (optimal은 아니더라도 exhaustive search 보다는 효율적) (31p~46p 그림)

결론적으로 모델에 k개씩 선별해 timestep마다 넣는 과정이다. => 최종 hypothesis도 k개가 나오고 그중 blue number가 가장 높은 것을 고르게 된다. 하지만 더 긴 문장의 hypothesis는(more timesteps) 더 낮은 score(blue number)가 나오니까

와 같이 normalize 한다.

SMT에 비해

1. 더 좋은 성능
2. Context를 더 잘 이용(input sequence를 인코딩하여 디코딩 시에 이용하니까)
3. Neural net을 사용해서 그런가 처음 phrase를 배우고 그 다음 데이터가 와도 너무 많이 엇나가지 않게 학습한다. (데이터를 점차 일반화하면서 배워, 테스트 시에도 일반성이 올라간다. = overfit에 멀어짐..?)
4. End-to-end => 심플 & 편리 => 추가적인 subcomponents로 processing 할 필요가 없다.
5. 또한 인간의 노력이 덜 든다 => 덜 복잡한 모델 과정, feature engineering 불필요
6. 또한 모든 언어에 대한 똑같은 method, 과정이 사용됨.

하지만

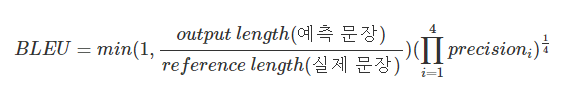
1. Less interpretable (학습 과정, inference 과정을 자세히 볼 수 없음, 모든 모델의 블랙박스 현상) = 디버깅 힘듬.
2. 딥러닝에 대한 어떤 규칙 그런 것을 정할 수가 없다. = 이럴 때의 예외에서는 이렇게 처리해줘 라고 하기에는 딥러닝의 과정에 알맞지 않다.
3. 딥러닝이 나쁜 말을 결과로 내뱉어도 이런 것을 제어할 수가 없다.

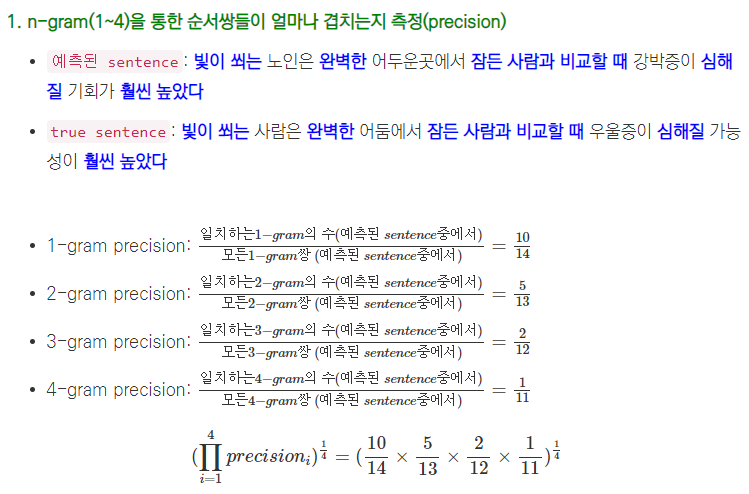
**잠깐 evaluation을 위한 BLEU 메트릭이란? Bilingual Evaluation Understudy score**

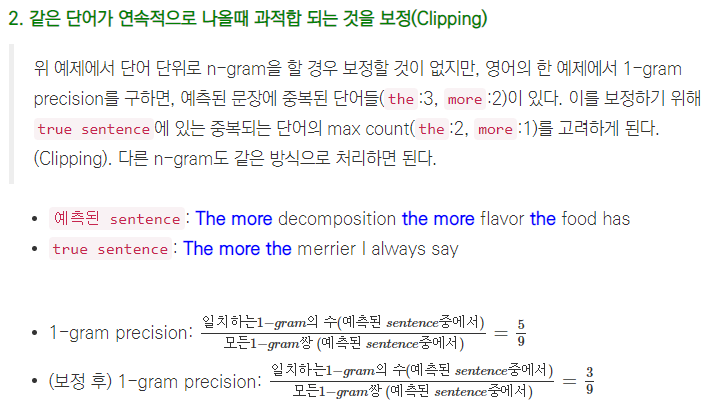
<https://donghwa-kim.github.io/BLEU.html>

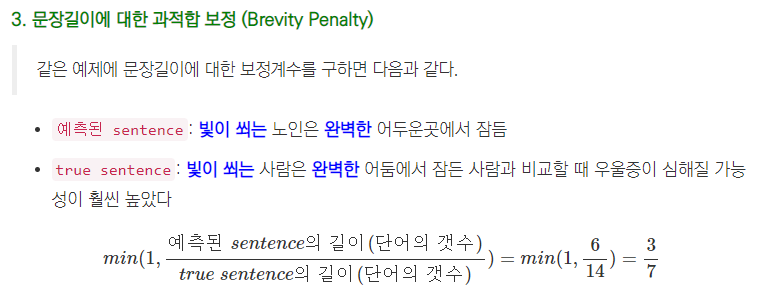
데이터 X가 sequence, y 또한 sequence => 번역 모델에 주로 사용

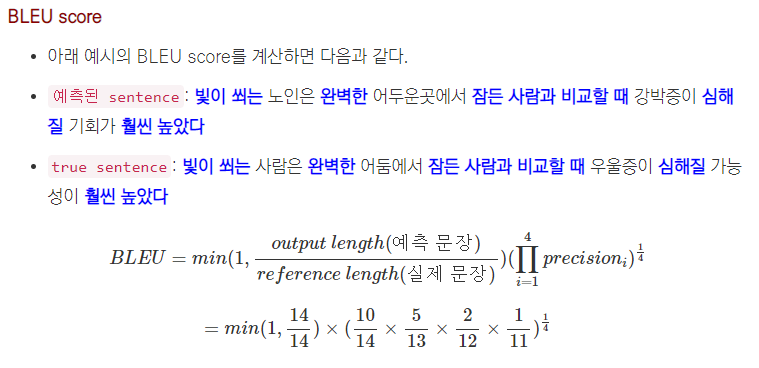
* N-gram을 통한 순서쌍들이 얼마나 겹치는지 측정(precision)
* 문장길이에 대한 과적합 보정(Brevity Penalty)
* 같은 단어가 연속적으로 나올 때 과적합 되는 것을 보정(Clipping)







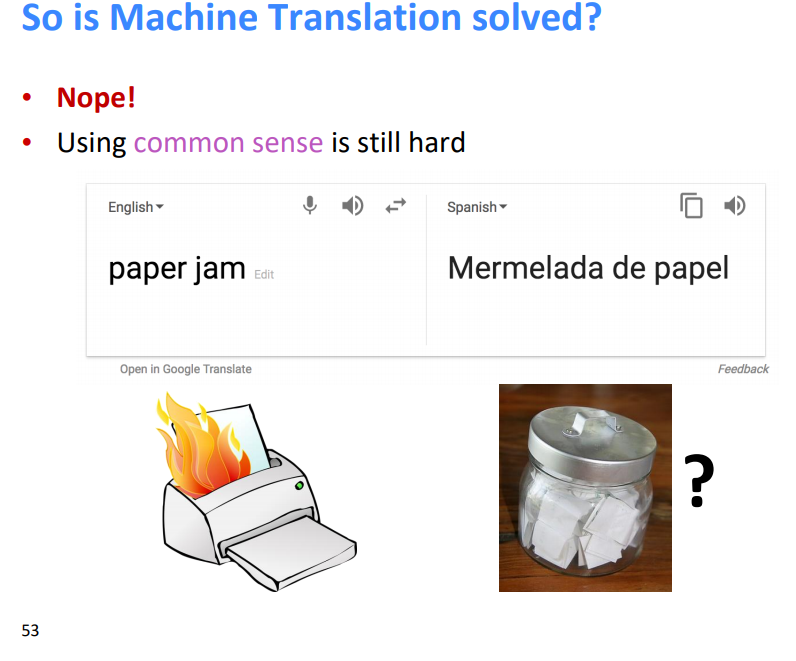




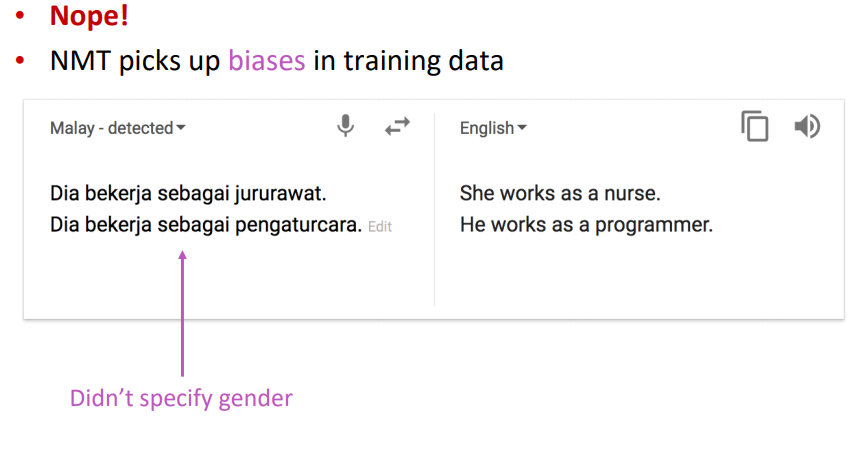
그러면 MT task는 이제 완벽한가??

아니다.

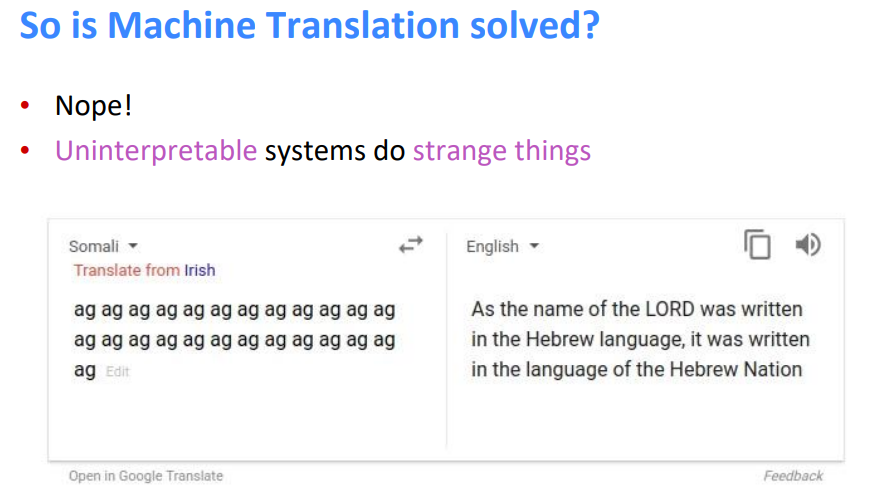
1. Vocab에 없는 단어가 input or output sequence에 필요할 때 어려움
2. Train은 엄청 formal한 데이터인 wikipedia로 하고 test는 twitter 대화 내용으로 한다면 도메인이 어느정도는 차이가 있어 성능이 떨어질 수도 있다.
3. 매우 긴 문장에서(book, news article)는 약할 수 있다.
4. Languages pairs 데이터가 부족하다.(구하기 어렵다.)



또한 common sense(상식?) 부족, 그냥 literal 하게 글자 그대로 해석.

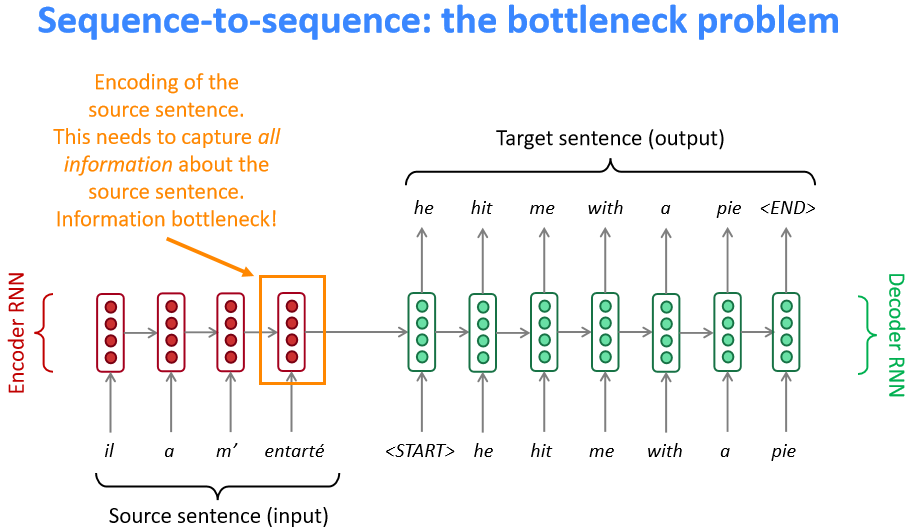


이 경우 사실 남자인지 여자인지 모르는데 train data에서 남자인 경우가 많기 때문에(=bias) 남자로 확정짓는 모습.



이런 non-sense 상황은 model이 예상하지 못하고 데이터셋에서 경험한 적이 없어서 잘 대처를 못하고 있다.

**Attention**



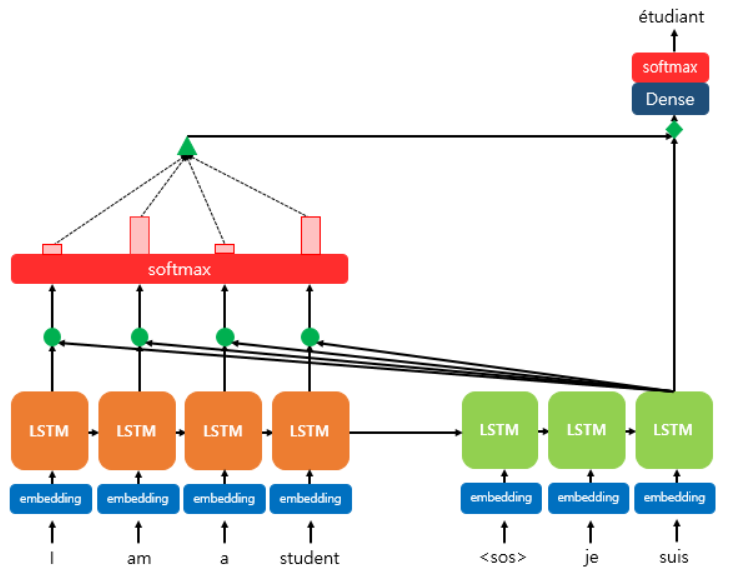
보는 것과 같이 마지막 유닛에서 sequence의 모든 정보를 encode 하게 된다. => bottleneck이 생길 수 밖에 없다.(정보 손실) => attention은 bottleneck problem을 해결해 준다.

또한 추가적으로 LSTM으로 보완했더라도 문장이 길어지면 long-term dependency(Vanishing Gradient)가 발생한다. Attention은 그걸 또한 막아준다.

따라서 decoder의 각 step에서 encoder에 direct connection! (source sequence의 특정 part에 집중!) 이게 핵심이자 컨셉

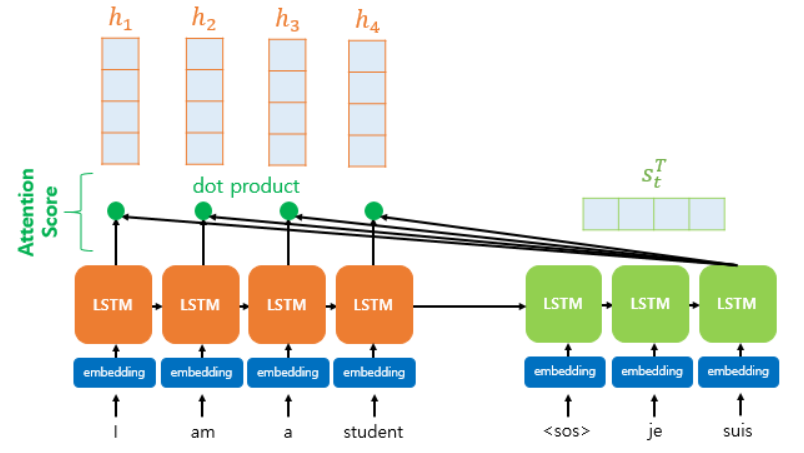
Flow를 설명하겠다.

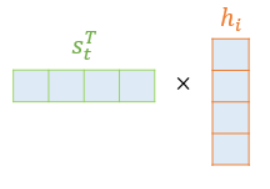
요약: decoder의 출력 단어 예측 직전에 인코더의 모든 입력 단어들의 정보를 다시 한번 참고



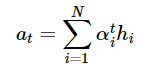
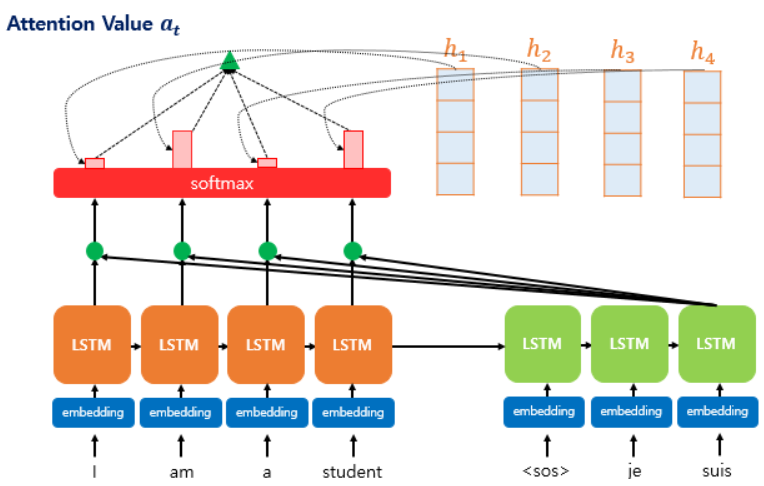
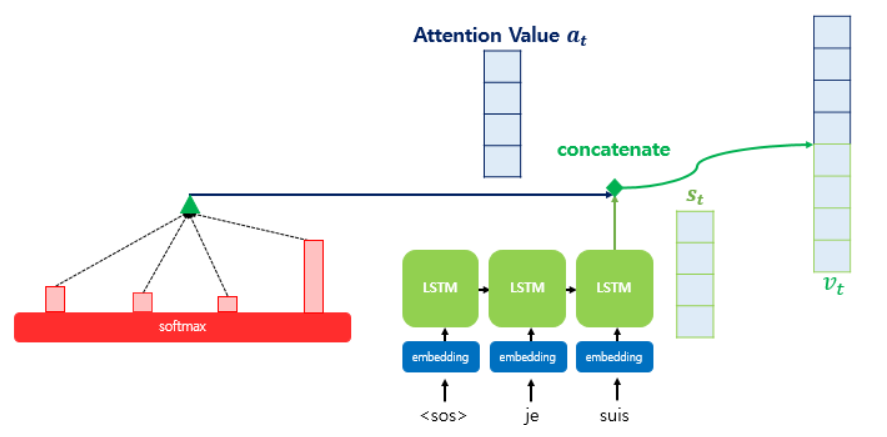
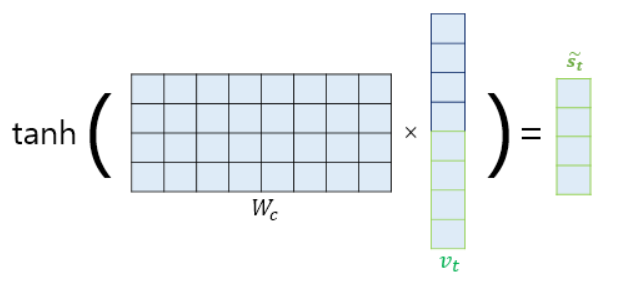
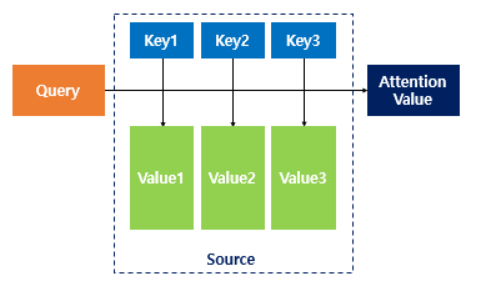
Softmax 결과 = 각 입력 단어가 출력 단어를 예측할 때 얼마나 도움이 되는지의 정도를 수치화한 값, 이 정보를 하나의 정보로 담아 디코더에 전송(초록색 삼각형)

1. Attention score 구하기  
   아래 그림을 참조하면 된다.





(et는 스코어 값들의 모음)

1. Softmax로 attention distribution을 구함.  
   (at는 어텐션 분포, 어텐션 가중치들의 모음)
2. 어텐션 가중치 \* 은닉 상태의 weigthed sum으로 어텐션 값 구함.  
   (at가 어텐션 값(어텐션 밸류)으로 바뀜), 기존 seq2seq는 encode된 vector가 context vector 였지만 여기서는 이 값이 context vector라 불림.  
     
     
   가 되어 st 대신에 st~가 되어 출력이 되고 최종적으로 가 되어 다음 timestep의 입력이 된다.  
   여기서 새로 추가된 Wc 웨이트 행렬은 다른 timestep과 연관이 크게 없으므로 backpropagation 시 그 timestep만 고려하면 되므로 복잡하지 않다.
3. 이때 attention score를 구하는 연산은 dot product를 사용했는데 다른 여러가지 방법이 존재 하기는 한다.
4. 또한 이 컨셉은 query, key, value 컨셉과 비슷한데  
   를 생각하면 된다. Query = st / key = hi들 / value = score 값들 이라고 보면 되고 최종적으로 attention value를 구하게 된다.

**Attention의 장점**

1. **Source를 timestep 예측 시 다시 한번 참조 하므로 소스의 특정 부분을 더 잘 파악하게 된다. => 성능 향상**
2. **Bottleneck(기존 seq2seq의 context vector 뿐만 아니라) 추가적인 새로운 context vector를 통해 bottleneck을 해결한다.**
3. **또한 source를 다시 한번 참조하니 long-term dependency 해결에 도움을 주며 자연스럽게 먼 거리의 소스 단어도 참조 가능하며 vanishing gradient의 해결에 도움을 준다.**
4. **또한 해석적인 의미를 가져올 수 있는데 디코더 출력하기 전에 우리는 소스 단어들의 어텐션 분포를 가져온다. 따라서 그때 마다의 timestep에서 어떤 source의 단어에 집중하고 있는지 해석할 수 있다.**
5. **즉, 소스, 타겟 시퀀스(랭귀지)에 대해서 어떠한 관계, 정렬(alignment)를 자연스럽게 알 수 있고, Wc의 학습을 통해 그것을 최적화한다.**

**attention이란 기법은 어디에든 강화 용도로 쓸 수 있다. Not just seq2seq also MT!  
즉, given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query!**