

Cs224n Lecture 8 (Winter 2019)

0. 개괄

- 기계 번역
- seq2seq
- Attention

>> 기계 번역은 seq2seq를 사용하여 수행되는데, 그 이후에 나온 Attention은 seq2seq 보다 더 높은 성능을 보인다.

1. 신경망 기계 번역 이전에

기계 번역: 한 언어(source language)의 한 문장 x 를 다른 언어(target language)의 문장 y 로 번역하는 작업

1.1 1950 [Rule-based]

1950s: Early Machine Translation

Machine Translation research began in the early 1950s.

- Russian → English
(motivated by the Cold War!)



1 minute video showing 1954 MT:
<https://youtu.be/K-HfpsHPmvw>

1.2 1990 - 2010 [Statstical Machine Translation]

>> 자료로부터 확률적 모델을 학습시킴

- We want to find best English sentence y , given French sentence x

$$\operatorname{argmax}_y P(y|x)$$

- Use Bayes Rule to break this down into two components to be learnt separately:

$$= \operatorname{argmax}_y P(x|y)P(y)$$

Translation Model

Models how words and phrases should be translated (*fidelity*).
Learned from parallel data.

Language Model

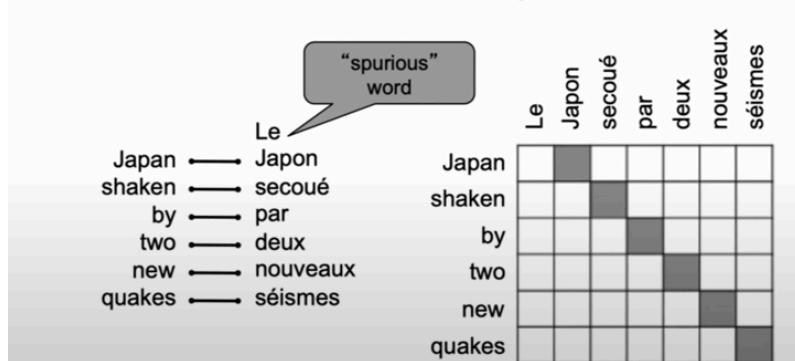
Models how to write good English (*fluency*).
Learned from monolingual data.

1.2.1 어떻게 $P(x|y)$ 번역 모델을 학습시키는가?

- 1) 많은 양의 Parallel data를 필요로 한다
- 2) 더 분해한다. $P(x, a | y)$. 여기서 a는 alignment (가령 프랑스 문장x와 영어 문장 y 사이의 “단어-수준의 상응 관계”)

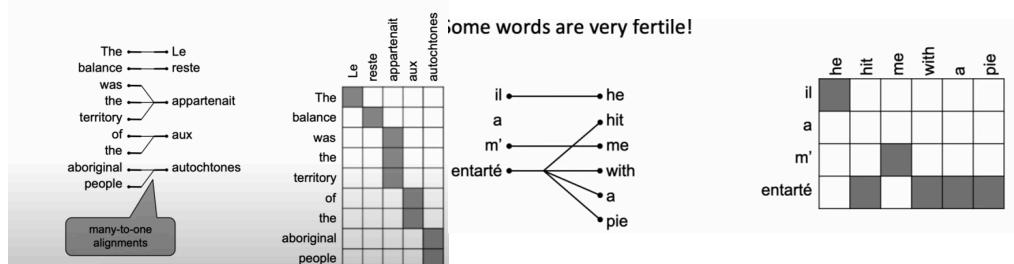
2-1) alignment: 번역된 문장의 쌍 내의 특정 단어들 사이의 상응 관계

Note: Some words have no counterpart



위에서 우리는 단어 간의 1 to 1 관계를 볼 수 있다.

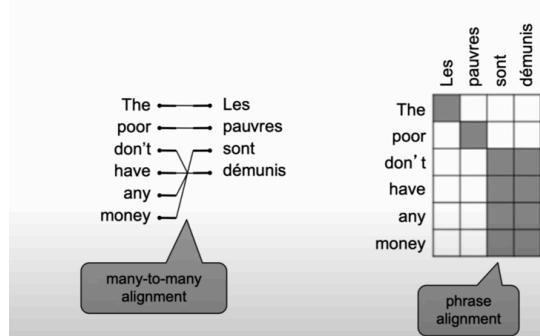
Alignment can be many-to-one



many — 1

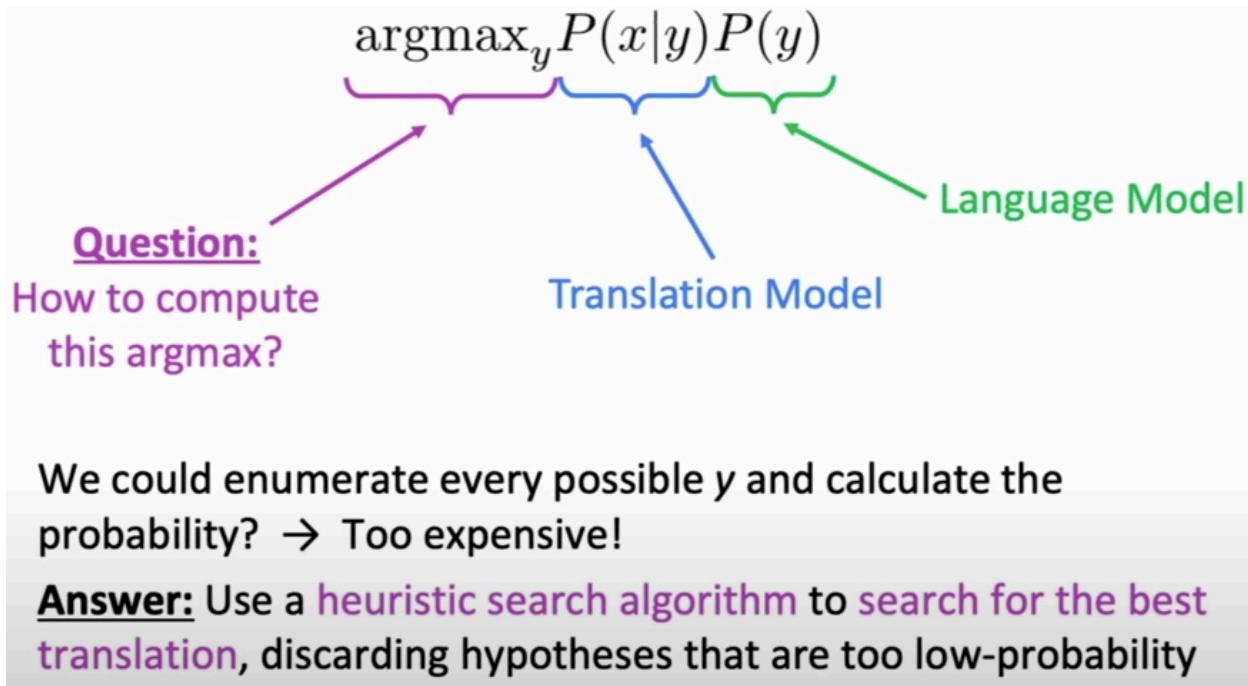
1 — many

Alignment can be many-to-many (phrase-level)

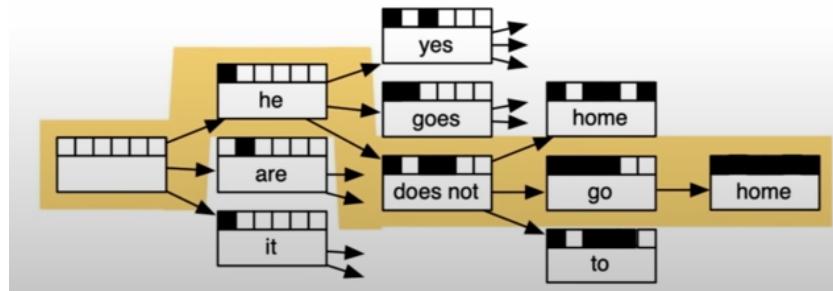


many — many

1.2.2 어떻게 SMT를 학습시키는가?



위 과정을 decoding이라고 부른다.



위 알고리즘은 위 같이 트리로 표현될 수 있다.

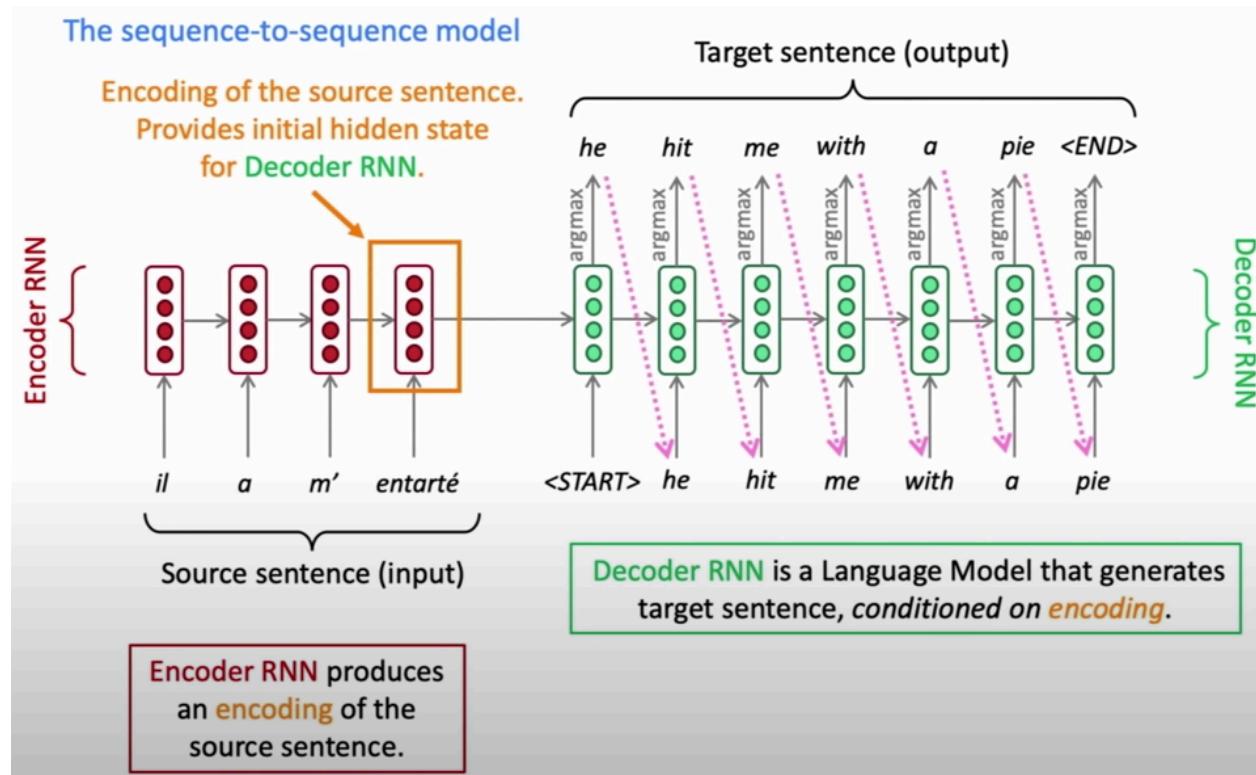
1.2.3 SMT의 단점

- (1) 너무 복잡하다. 이 시스템에는 개별적으로 설계된 하부 요소가 많이 필요했다.
- (2) 추가적인 리소스를 컴파일하고 유지해야 한다. 설상가상으로, 다른 언어에 대해서는 다른 리소스를 가지고 수행해야 한다.
- (3) 사람의 노력이 많이 필요했다.

2. 신경망 기계 번역 NMT 2014 -

- (1) NMT: 하나의 신경망을 사용하는 기계 번역
- (2) seq2seq: 2 개의 RNN을 사용하는 기계 번역

seq2seq:



사용 분야는 여러 군데

- Summarization (long text → short text)
- Dialogue (previous utterances → next utterance)
- Parsing (input text → output parse as sequence)
- Code generation (natural language → Python code)

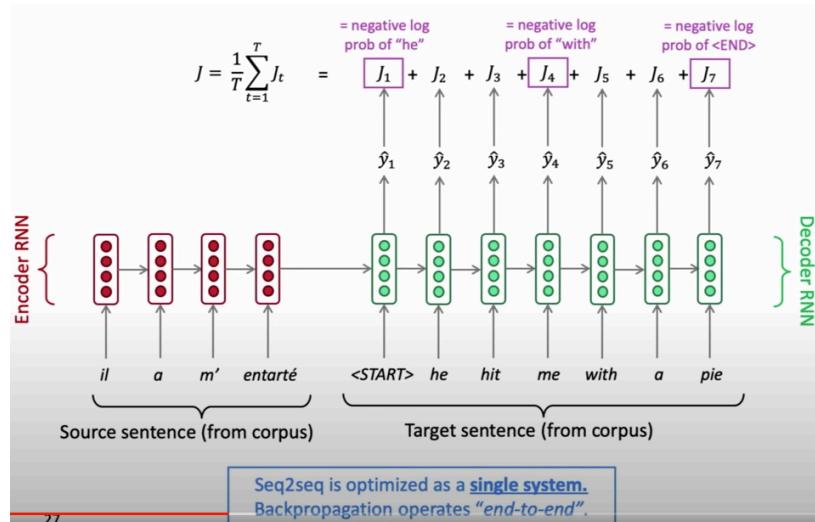
seq2seq는 (1) 언어 모델, (2) 조건적이다.

- The sequence-to-sequence model is an example of a **Conditional Language Model**.
 - **Language Model** because the decoder is predicting the next word of the target sentence y
 - **Conditional** because its predictions are *also* conditioned on the source sentence x
- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots P(y_T|y_1, \dots, y_{T-1}, x)$$

Probability of next target word, given target words so far and source sentence x

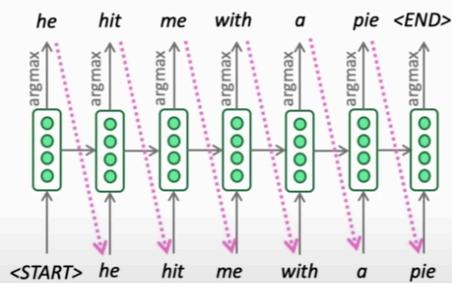
훈련 과정



디코딩 과정

With Greedy Decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder



- This is **greedy decoding** (take most probable word on each step)
- Problems with this method?**

2.1 Greedy 방법의 문제점

- (1) 결정을 취소할 방법이 없다
>> 해결 방안: 서치 알고리즘

Greedy decoding has no way to undo decisions!

- Input: il a m'entarté (he hit me with a pie)
- he ____
- he hit ____
- he hit a ____ (whoops! no going back now...)

2.1.1 나이브 서치 알고리즘

Exhaustive search decoding

- Ideally we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots, P(y_T|y_1, \dots, y_{T-1}, x)$$

$$= \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, x)$$

- We could try computing all possible sequences y

- This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size
- This $O(V^T)$ complexity is far too expensive!

2.1.2 Beam 서치 알고리즘

디코더의 각 단계에서 k 개의 가장 probable 부분 번역 (hypotheses)을 추적하는 것

k : beam size (보통 5 - 10)

A hypothesis y_1, \dots, y_t has a score which is its log probability:

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step

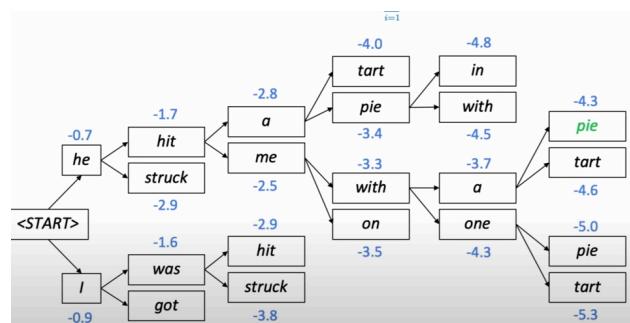
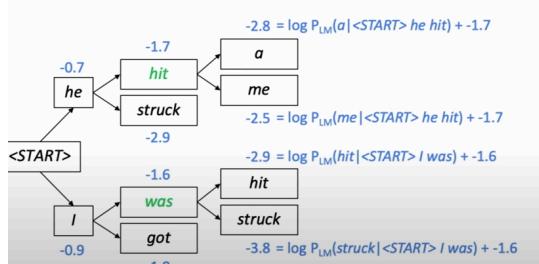
- score는 음수
- 더 높은 score가 더 좋음
- 각 단계에서 가장 높은 k 추적

>> 주의: 빔 서치 알고리즘은 최적 솔루션을 찾게 해준다는 것을 보장하지 않는다.
>> 이점: 더 효율적

2.1.2.1 빔 서치 알고리즘 과정

Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



2.1.2.1.1 빔 서치 알고리즘은 언제 멈추나?

- 타임스텝 T에 도달할 때
- 적어도 n 개의 완료된 hypotheses를 가질 때 (A hypothesis가 <END> 만들 시, 그 hypothesis는 완료됨. 그런 뒤에, 그 가설을 제쳐두고 다른 가설을 빔 탐색)

2.1.2.1.2 빔 서치 알고리즘은 어떻게 highest score로 가장 높은 점수를 선택하는가?

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

문제: 가설이 더 길수록, scores는 낮아진다.

>> 해결 방안: Normalization

Fix: Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

2.2 NMT vs SMT

장점:

1. NMT는 SMT 보다 더 좋은 성능을 가진다.
 - More fluent
 - Better use of context (source sentence를 사용하여 출력한다는 점에서)
 - Better use of phrase similarities
2. NMT에서 사용하는 하나의 신경망은 end to end에 최적화되어 있다.
 - 별도의 하부 요소가 필요 없다.
3. 인간의 노력이 덜 요구된다.
 - 모든 언어에 대해 동일한 아키텍처, 방법을 사용할 수 있다.

단점:

1. NMT는 어떻게 이것이 작동되었는지 잘 알기 어렵다 (less interpretable)

- 하나의 소스 문장을 신경망에 넣은 다음 일부 타깃 문장이 출력되었을 때 이것이 어떻게 일어났는지 알 수 없다. 그래서 debug하기 어렵다.

2. NMT는 컨트롤하기 어렵다.

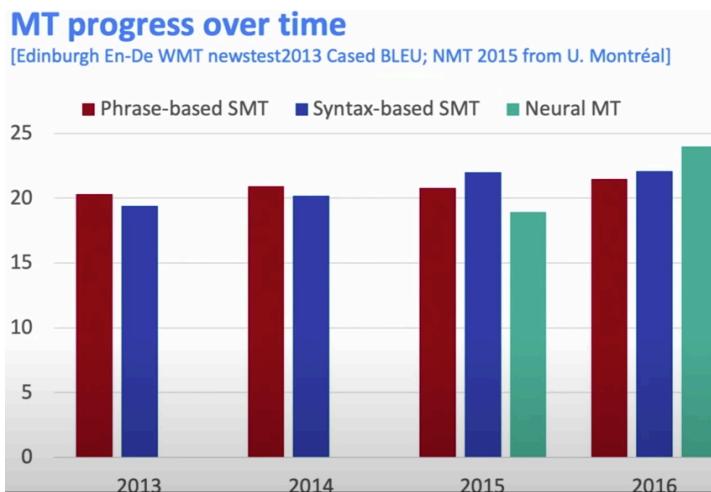
- NMT에게 특정 규칙이나 번역의 가이드라인을 유저가 부여하는 건 어렵다.

3. Machine Translation의 평가 지표

BLEU (Bilingual Evaluation Understudy)

- 기계가 번역한 것과 인간이 번역한 하나 또는 몇 개의 것을 비교하고, (1) n-gram precision, (2) too-short system translations에 대한 패널티 — 를 기초로 하여 유사도 점수를 계산.
- BLEU는 유용하지만 불완전하다

3.1 BLEU를 이용한 평가



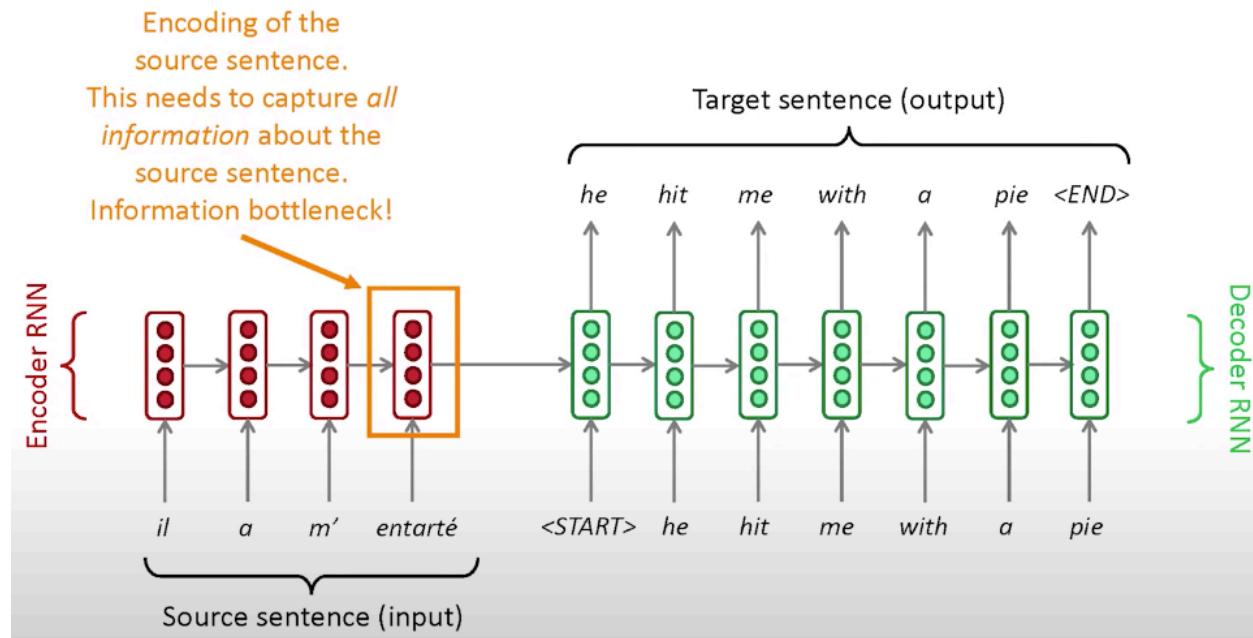
4. 여전히 남아 있는 기계 번역이 갖는 곤경들

- Source data, Target data(voca)에 없는 단어가 포함된 단어 처리 어려움
- 훈련 데이터와 테스트 데이터 간의 domain mismatch
- 긴 글에 대한 context를 유지하기 (가령 뉴스 기사 전체)
- common sense 는 여전히 다루기 어렵다. (e.g. paper jam)
- NMT는 트레이닝 셋에서 biases를 고른다. (nurse → she, programmers → he)

5. Attention

5.1 Motivation

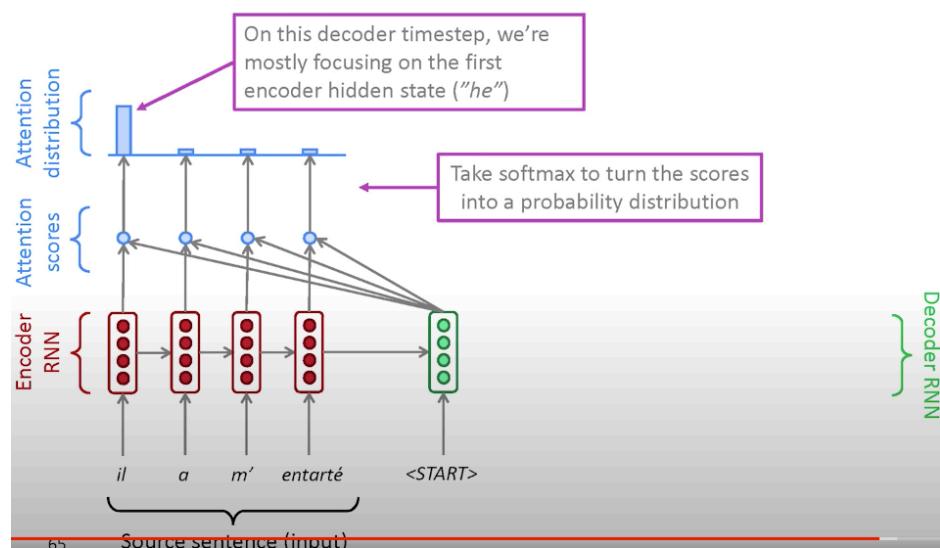
Sequence-to-sequence: the bottleneck problem



- seq2seq 에서는 소스 문장에 대한 모든 정보를 포착해야 한다. —> 정보 병목 현상
- 어텐션은 위 병목 현상을 해결해준다.

5.2 Attention

>> 디코더의 각 단계에서, 소스 시퀀스의 특정 부분에 집중하기 위해 인코더에 직접 연결을 사용.



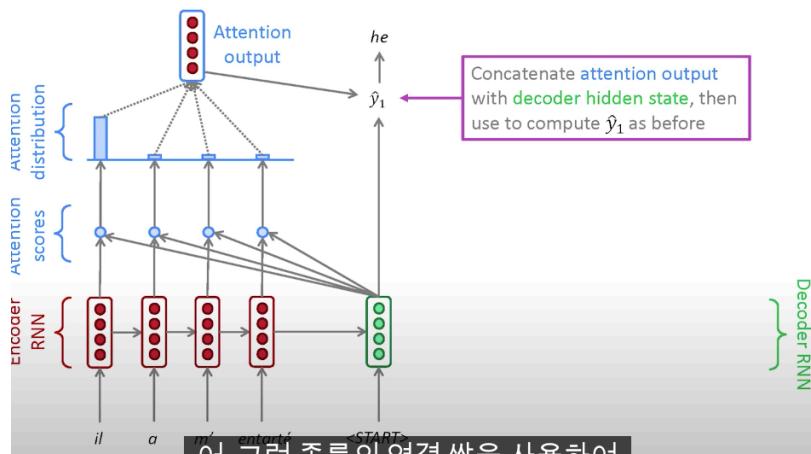
여기서는 ‘he’가 가장 높은 분포를 갖는다. → 가장 목표 문장 내에서 먼저 출력

5.2.1 어텐션 사용

Use the **attention distribution** to take a **weighted sum** of the **encoder hidden states**.

The **attention output** mostly contains information from the **hidden states** that received **high attention**.

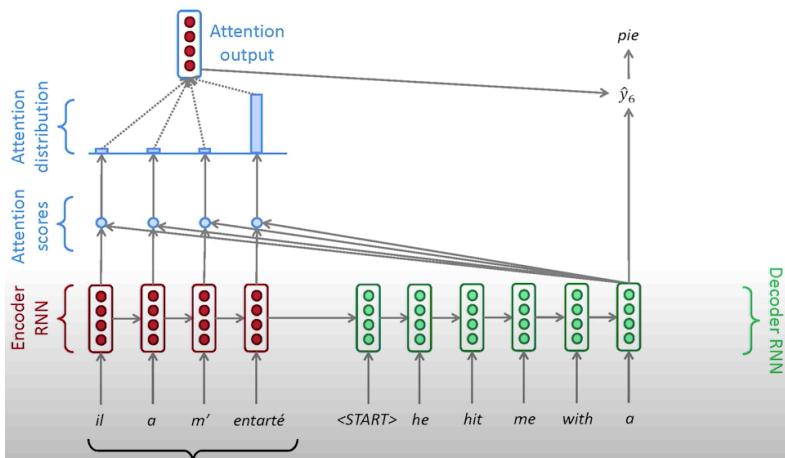
점선으로 표현된 어텐션 출력에 대한 분포 아이디어는 인코더 RNN의 요약을 함의한다.



위 같이 분포를 구하면, <어텐션 출력값>과 <디코더 은닉 상태>를 연결한 뒤, y 예측값을 계산하는데 사용한다.

위 과정을 반복

Sequence-to-sequence with attention



5.3 Equations

Attention: in equations

- We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$
- On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

73

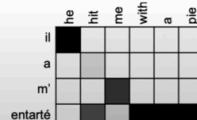
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

e^t : 어텐션 스코어 = 디코더의 은닉 상태와 인코더의 은닉 상태 간 내적

5.4 advantages

Attention is great

- Attention significantly improves NMT performance
 - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
 - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
 - Provides shortcut to faraway states
- Attention provides some interpretability
 - By inspecting attention distribution, we can see what the decoder was focusing on
 - We get (soft) alignment for free!
 - This is cool because we never explicitly trained an alignment system
 - The network just learned alignment by itself



직접 연결 -> vanishing grads 줄여줘
alignment 정의 안해줘도 돼.

5.5 어텐션에 대한 직관

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

5.6 여러 변칙들

Attention variants

There are *several ways* you can compute $e \in \mathbb{R}^N$ from $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $s \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $e_i = s^T \mathbf{h}_i \in \mathbb{R}$
 - Note: this assumes $d_1 = d_2$
 - This is the version we saw earlier
- Multiplicative attention: $e_i = s^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$
 - Where $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $e_i = v^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 s) \in \mathbb{R}$
 - Where $\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}$, $\mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector.
 - d_3 (the attention dimensionality) is a hyperparameter

References

[1] Stanford CS224N: NLP with Deep Learning | Winter 2019 | Lecture 8 – Translation, Seq2Seq, Attention

