# Natural Language Processing with Deep Learning

# CS224N/Ling284

Lecture 8:
Machine Translation,
Sequence-to-sequence and Attention

**Abigail See**

# Announcements

- We are taking attendance today
  - Sign in with the TAs outside the auditorium
  - No need to get up now – there will be plenty of time to sign in after the lecture ends
  - For attendance policy special cases, see Piazza post for clarification

- Assignment 4 content covered today
  - Get started early! The model takes 4 hours to train!

- Mid-quarter feedback survey:
  - Will be sent out sometime in the next few days (watch Piazza).
  - Complete it for 0.5% credit

# Overview

Today we will:

- Introduce a <u>new task</u>: Machine Translation

is a major use-case of

- Introduce a <u>new neural architecture</u>: sequence-to-sequence

is improved by

- Introduce a <u>new neural technique</u>: attention

# Section 1: Pre-Neural Machine Translation

# Machine Translation

**Machine Translation (MT)** is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).

*x:* *L'homme est né libre, et partout il est dans les fers*

*y:* *Man is born free, but everywhere he is in chains*

\- Rousseau

# 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

- Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts

# 1990s-2010s: Statistical Machine Translation

- Core idea: Learn a probabilistic model from data
- Suppose we're translating French → English.
- We want to find best English sentence *y*, given French sentence *x*

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

p(y|x)에서 argmax y를 찾는 것은 매우 어렵다. 이 확률을 좀 더 쉽게 표현하기 위해 bayes rule을 사용한다.

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

같은 조건부 확률이지만, argmax y의 y가 p(x|y)로 된 것을 확인하자. 이는 p(y|x)보다 확률 추정이 훨씬 쉽다.

1개의 확률을 2개의 확률로 나누는 이유?

위의 1개의 확률에는 how to translate, how to write good english text, how to understand sentence structure 등 모든 것을 한 꺼번에 모델링해야 한다.

이는 어렵기 때문에 문제를 쪼개서 생각한다.

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

**Translation Model**

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

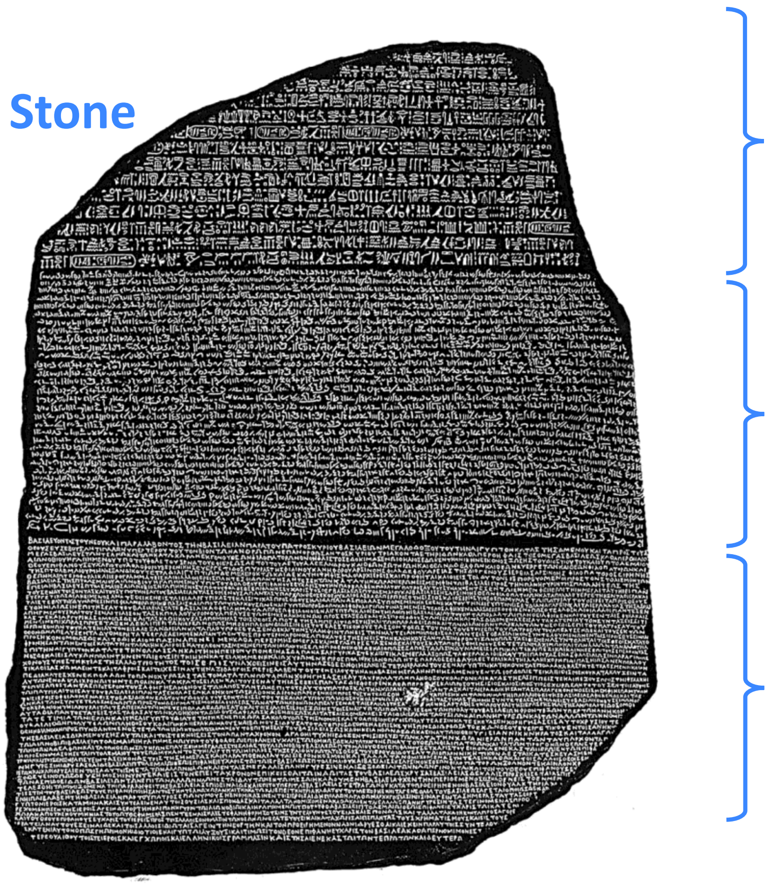
**Language Model**

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

# 1990s-2010s: Statistical Machine Translation

- Question: How to learn translation model $P(x|y)$ ?
- First, need large amount of parallel data
  (e.g. pairs of human-translated French/English sentences)

The Rosetta Stone



# Learning alignment for SMT

- Question: How to learn translation model $P(x|y)$ from the parallel corpus?

- Break it down further: we actually want to consider

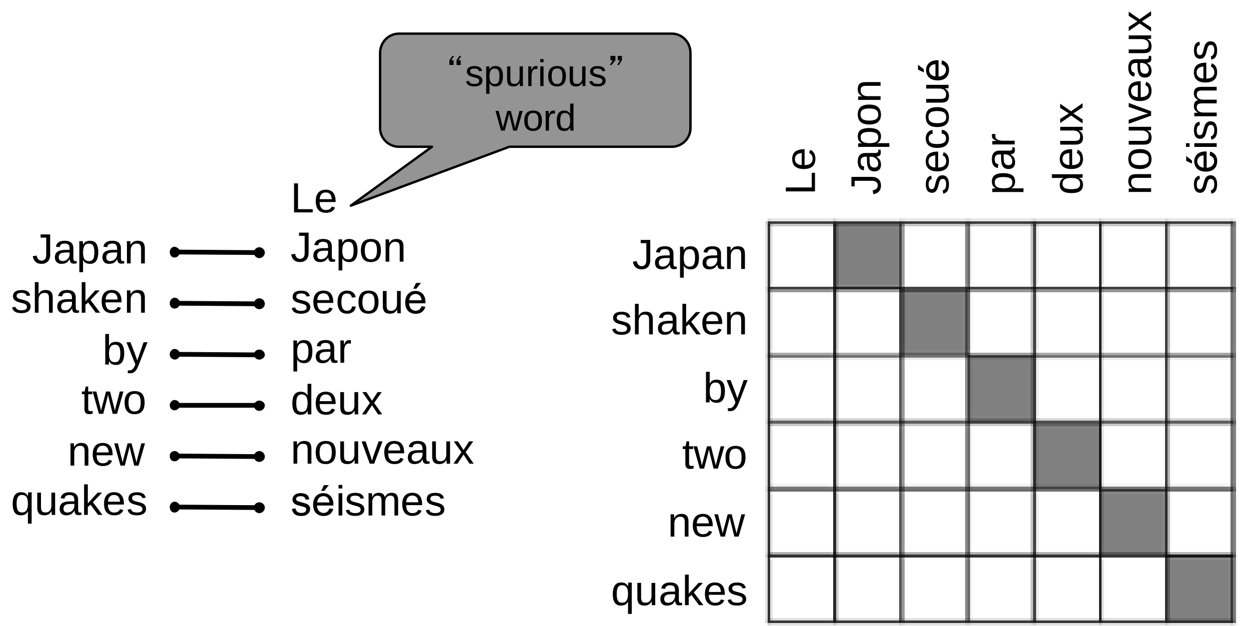
$$P(x, a|y)$$

where *a* is the alignment, i.e. word-level correspondence between French sentence *x* and English sentence *y*

# What is alignment?

Alignment is the correspondence between particular words in the translated sentence pair.

- Note: Some words have no counterpart



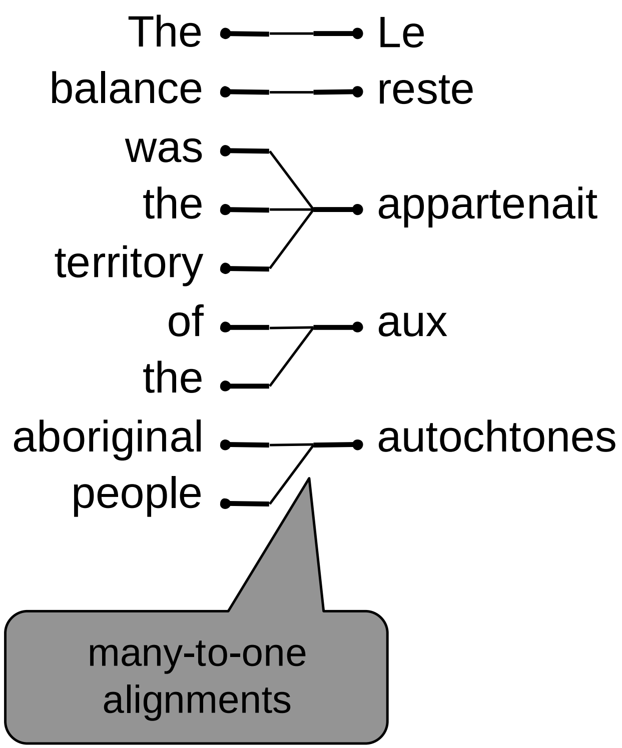

모두 1:1 alignment 이다.

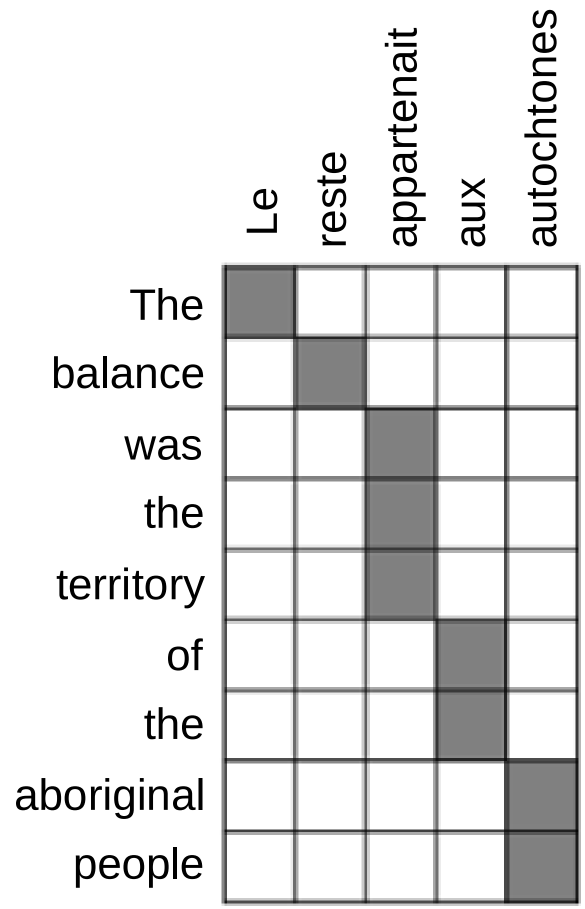


**Examples from:** "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

# Alignment is complex

Alignment can be many-to-one



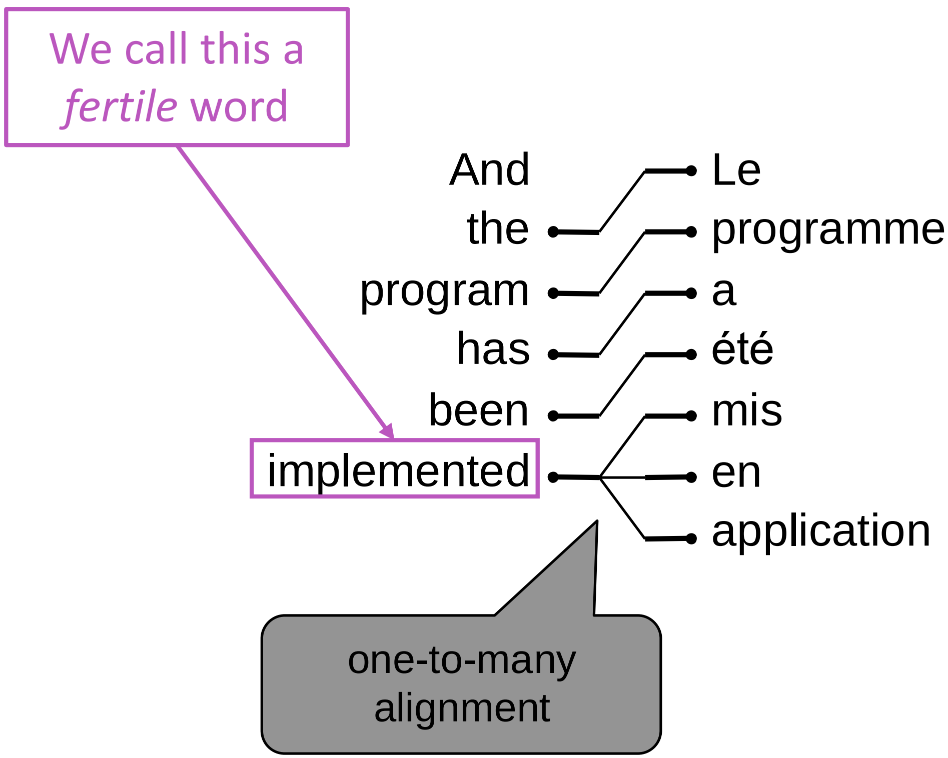



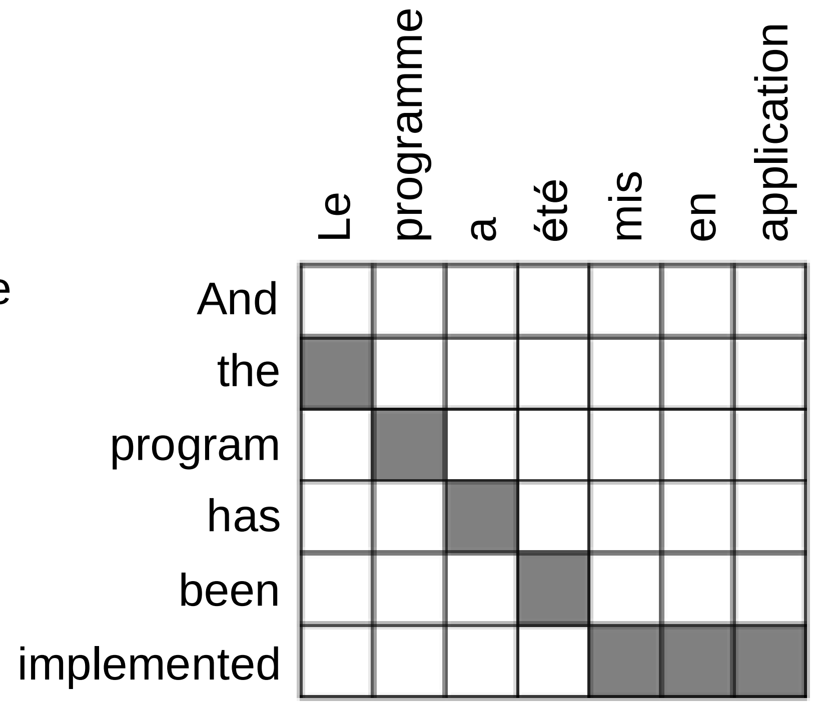



**Examples from:** "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

# Alignment is complex

Alignment can be one-to-many



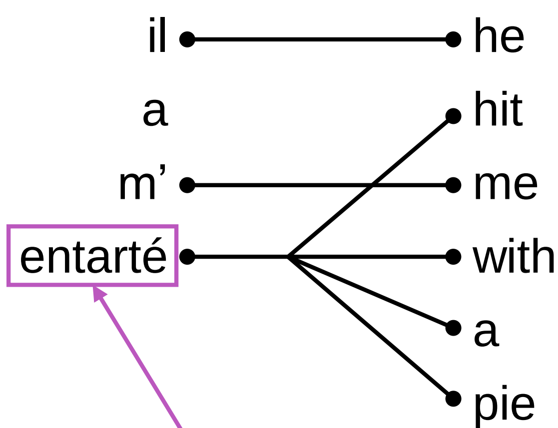



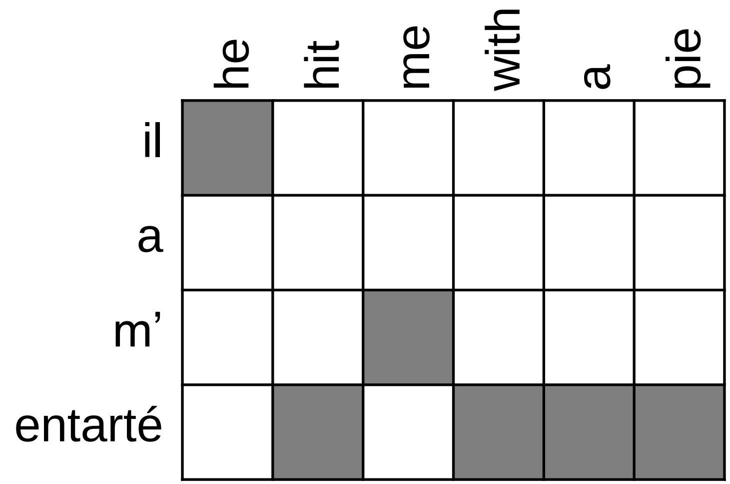



**Examples from:** “The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

# Alignment is complex

Some words are very fertile!

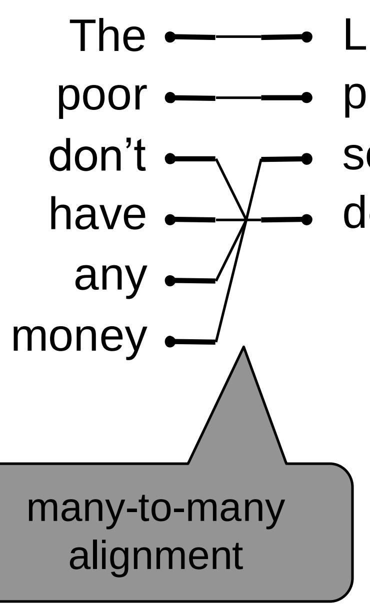


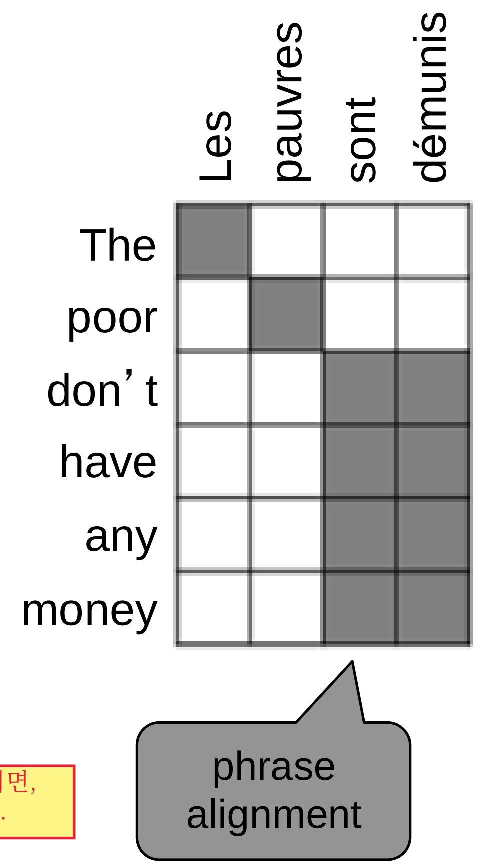

This word has no single-word equivalent in English

# Alignment is complex

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




many - to - many alignment이면,
phrase alignment라고 부른다.






**Examples from:** "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

# Learning alignment for SMT

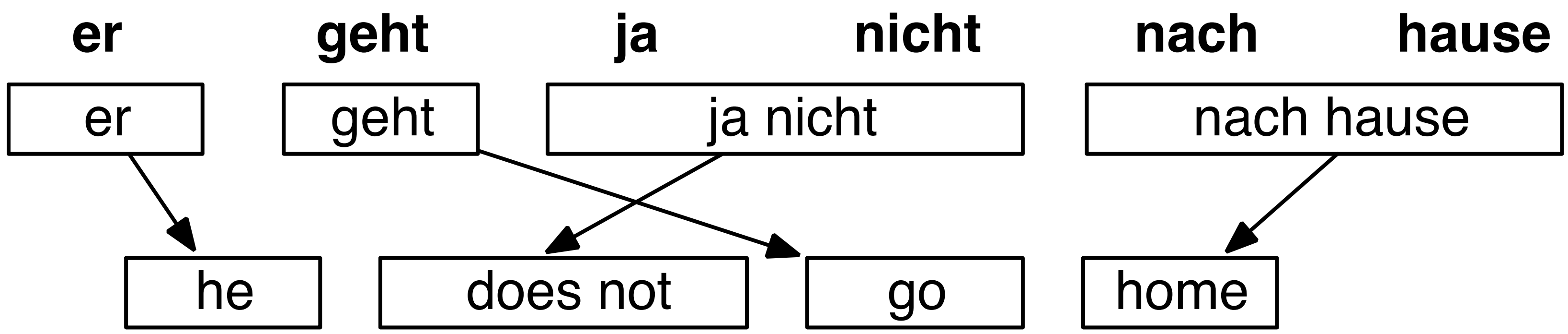
- We learn $P(x, a|y)$ as a combination of many factors, including:
  - Probability of particular words aligning (also depends on position in sent) what's the probability of a particular French word, aligning to another particular word? what's positions of each word in each sentence?
  - Probability of particular words having particular fertility (number of corresponding words) what's the probability of the particular French word, having the particular fertility?
  - etc.

# Decoding for SMT

$$\underbrace{\text{argmax}_y}_{\text{Question: How to compute this argmax?}} \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

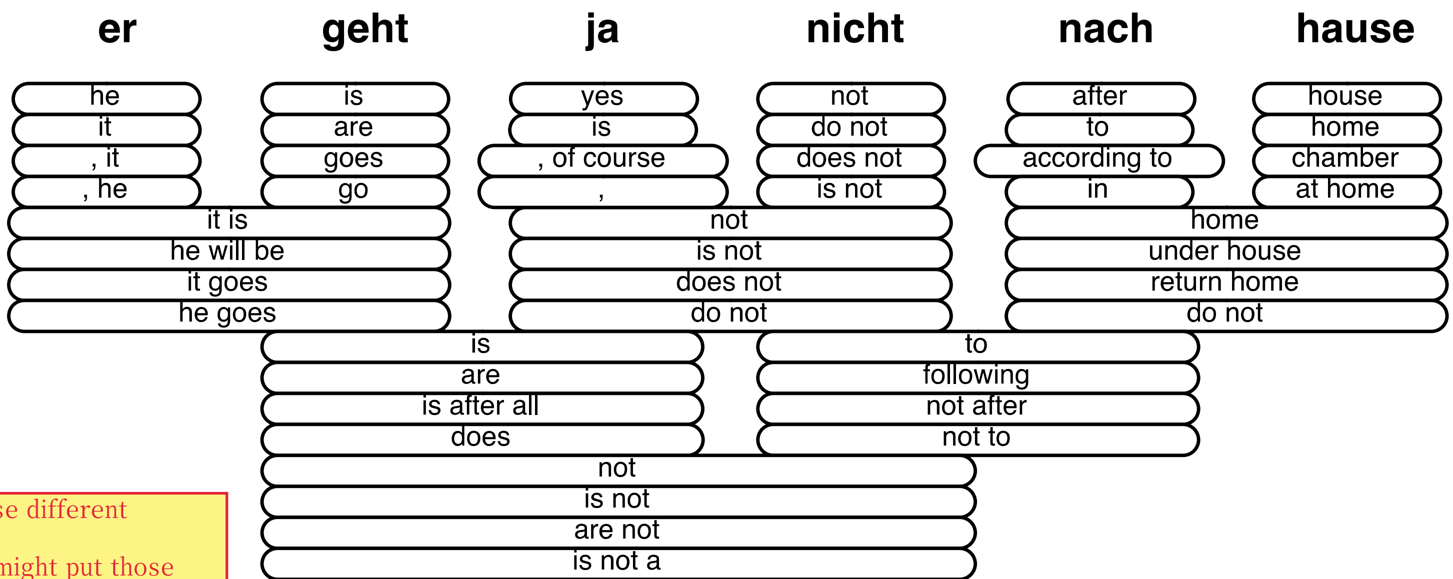
- We could enumerate every possible *y* and calculate the probability? → Too expensive!
- **Answer:** Use a heuristic search algorithm to search for the best translation, discarding hypotheses that are too low-probability
- This process is called *decoding*

# Decoding for SMT

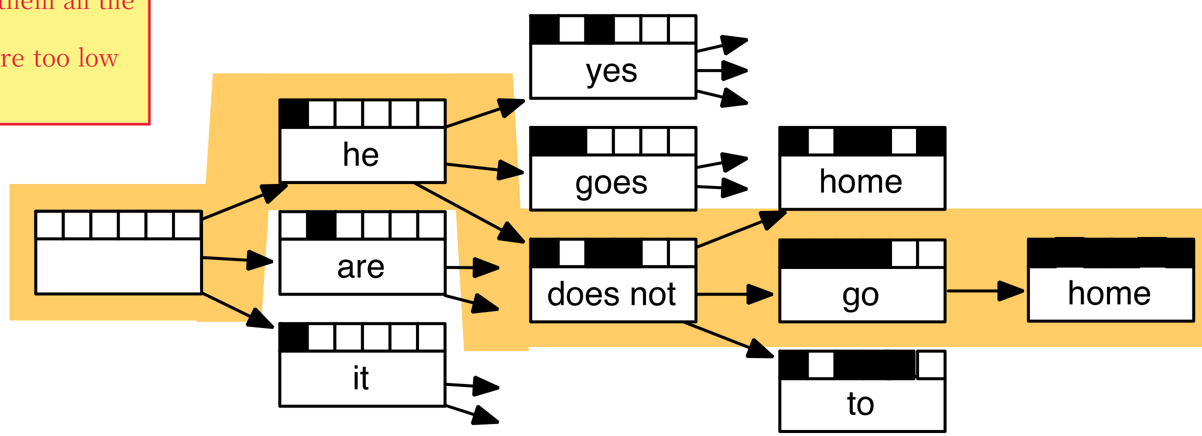



**Source:** "Statistical Machine Translation", Chapter 6, Koehn, 2009.
https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5

# Decoding for SMT




Consider all of these different hypotheses, and
Look into how we might put those together to translate phrases,
but, you don't keep all of them all the time.
Get rid of the ones, that are too low probability.




**Source:** "Statistical Machine Translation", Chapter 6, Koehn, 2009.
https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5

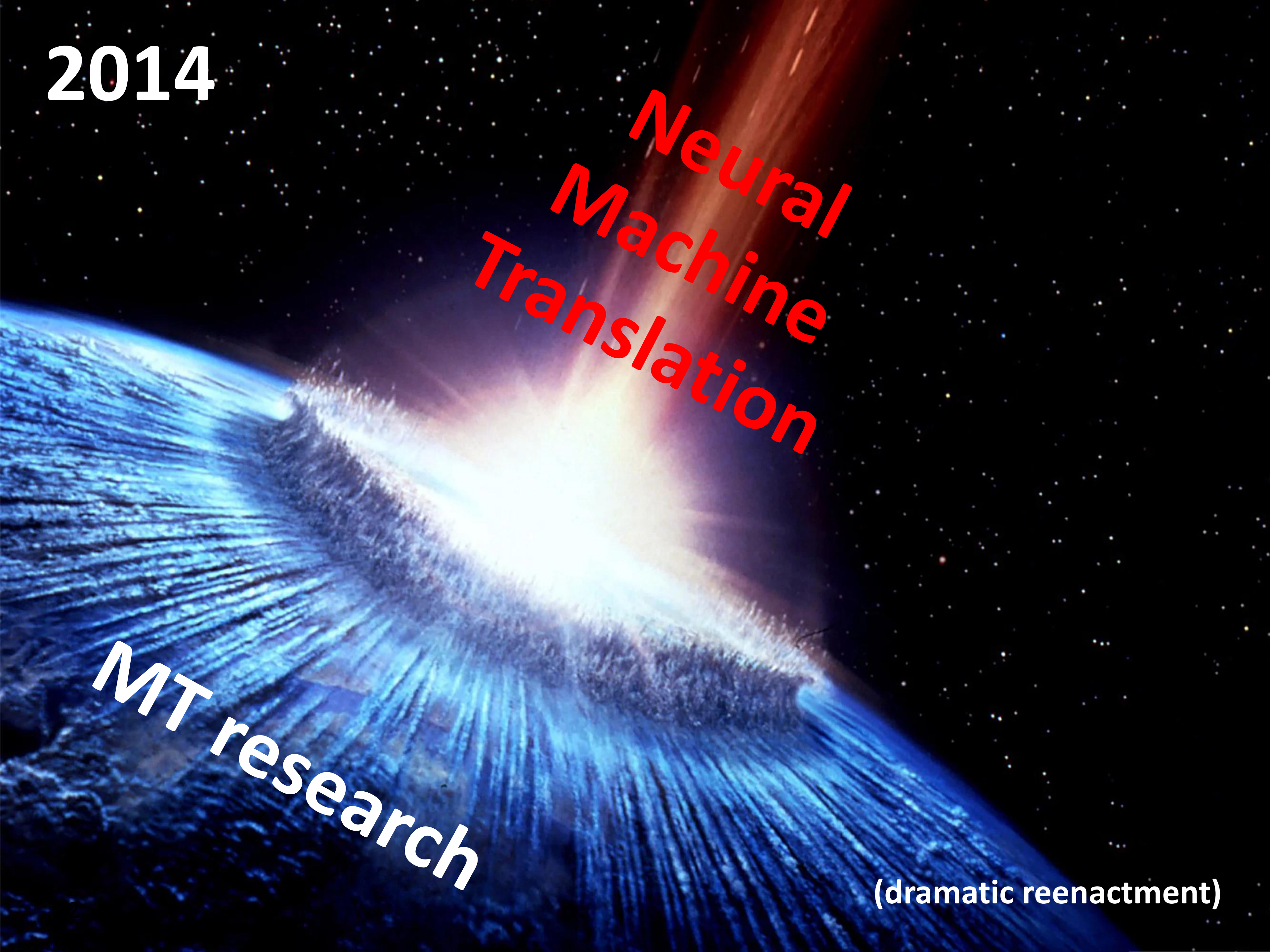
# 1990s-2010s: Statistical Machine Translation

- SMT was a huge research field
- The best systems were extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems had many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

# Section 2: Neural Machine Translation

# 2014

(dramatic reenactment)


2014
Neural
Machine
Translation
MT research
(dramatic reenactment)

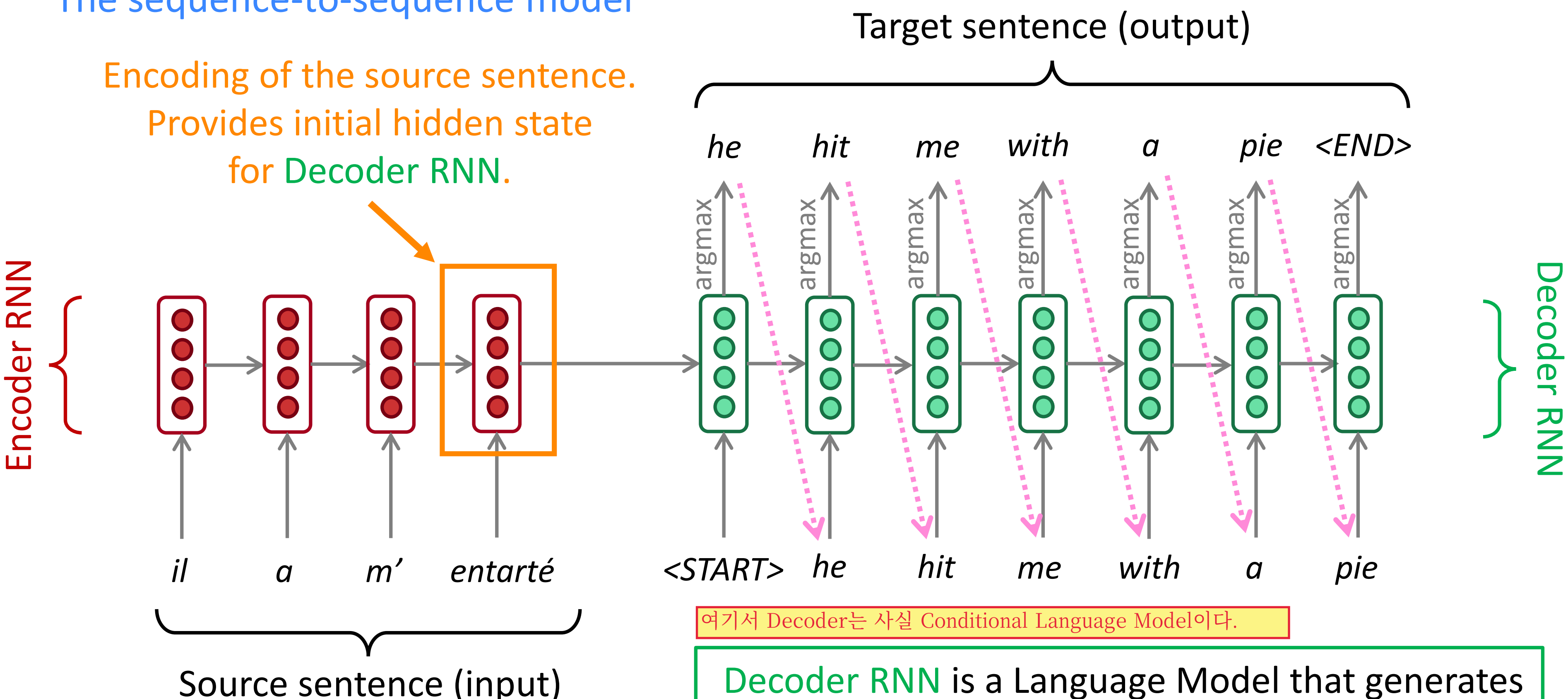
# What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*

- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

# Neural Machine Translation (NMT)

The sequence-to-sequence model




여기서 Decoder는 사실 Conditional Language Model이다.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Encoder RNN produces an encoding of the source sentence.

Note: This diagram shows **test time** behavior: decoder output is fed in ······> as next step's input

# Sequence-to-sequence is versatile!

- Sequence-to-sequence is useful for *more than just MT*

- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

About Parsing.
This might not be the best way to do parsing, but it is a way you can try.

# Neural Machine Translation (NMT)

- 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*

SMT는 directly하게 p(y|x) 확률을 구하진 못했다 (bayes rule을 사용해서 두 개의 smaller component로 쪼갰다)
반면...
NMT는 directly하게 p(y|x) 확률을 학습할 수 있다. 문제는 복잡하지만 풀어야되는 입장에서는 그냥 하나의 시스템만 최적화하면 되므로 더 쉬워졌다고 볼 수 있다.

- NMT directly calculates $P(y|x)$ :

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

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

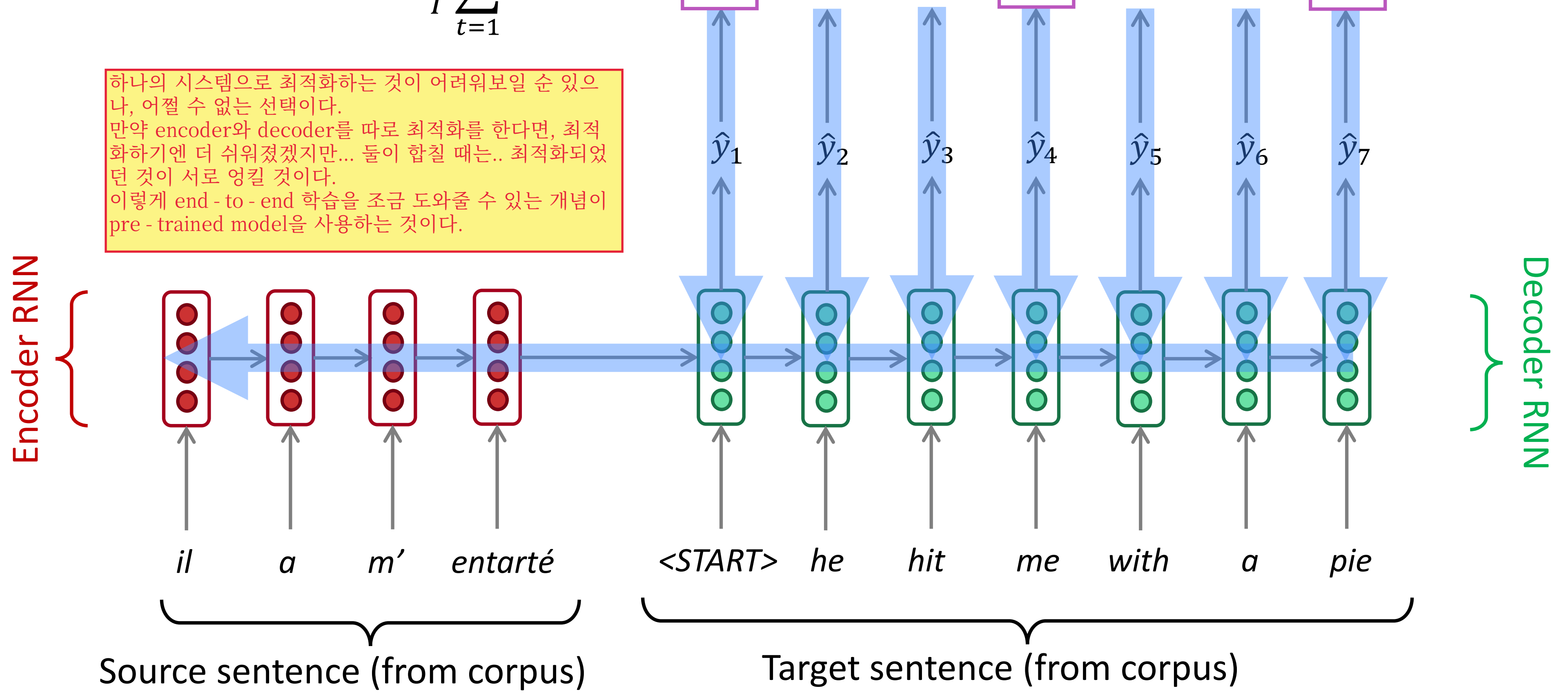
- **Question**: How to train a NMT system?
- **Answer**: Get a big parallel corpus…

# Training a Neural Machine Translation system

= negative log prob of “he” = negative log prob of “with” = negative log prob of <END>

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t \quad = \quad \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$$

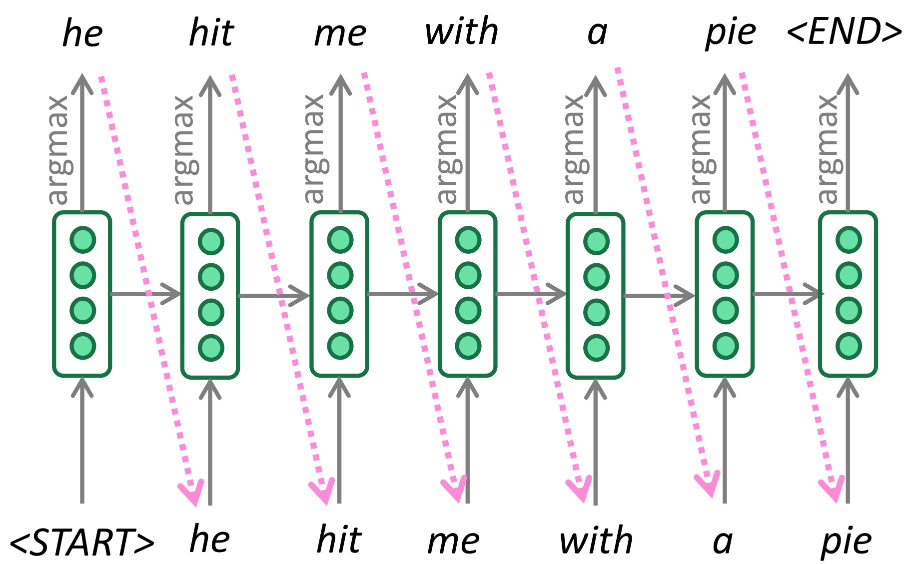
하나의 시스템으로 최적화하는 것이 어려워보일 순 있으나, 어쩔 수 없는 선택이다.
만약 encoder와 decoder를 따로 최적화를 한다면, 최적화하기엔 더 쉬워졌겠지만... 둘이 합칠 때는.. 최적화되었던 것이 서로 엉킬 것이다.
이렇게 end - to - end 학습을 조금 도와줄 수 있는 개념이 pre - trained model을 사용하는 것이다.




Seq2seq is optimized as a **single system.**
Backpropagation operates “*end-to-end*”.

# 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?**

여기서 greediness에 문제가 있다.
each step마다 argmax를 하는 것과.
the entire sentence의 argmax를 하는 것은 다르다.

Greedy decoding은 마치 시퀀스 정보를 없애고 단어마다 모두 독립이다고 간주하는 것과 같다.

# Problems with greedy decoding

- 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…)

greedy decoding의 문제점.
잘못 예측한 경우 다시 되돌아갈 수가 없다.

- How to fix this?

# Exhaustive search decoding

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

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

한 단어 하나하나가 아니라
전체 문장의 확률이 가장 높은 것을 찾아야 한다.

- 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!

그냥 무식하게 다 찾는 것은
경우의 수가 매우 많아서 거의 불가능하다.
똑똑한 search 알고리즘이 필요하다.

시퀀스 조합의 경우의 수가 너무너무 많다.

# Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

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

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

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

# Beam search decoding: example

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

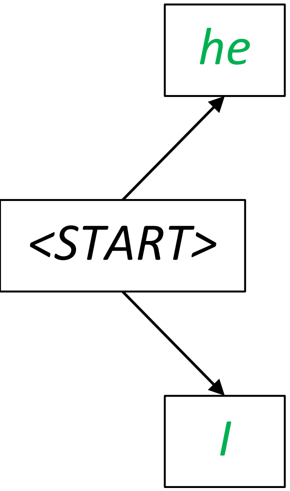
*<START>*

Calculate prob dist of next word

# Beam search decoding: example

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

-0.7 = log $P_{LM}$(*he* | *<START>*)

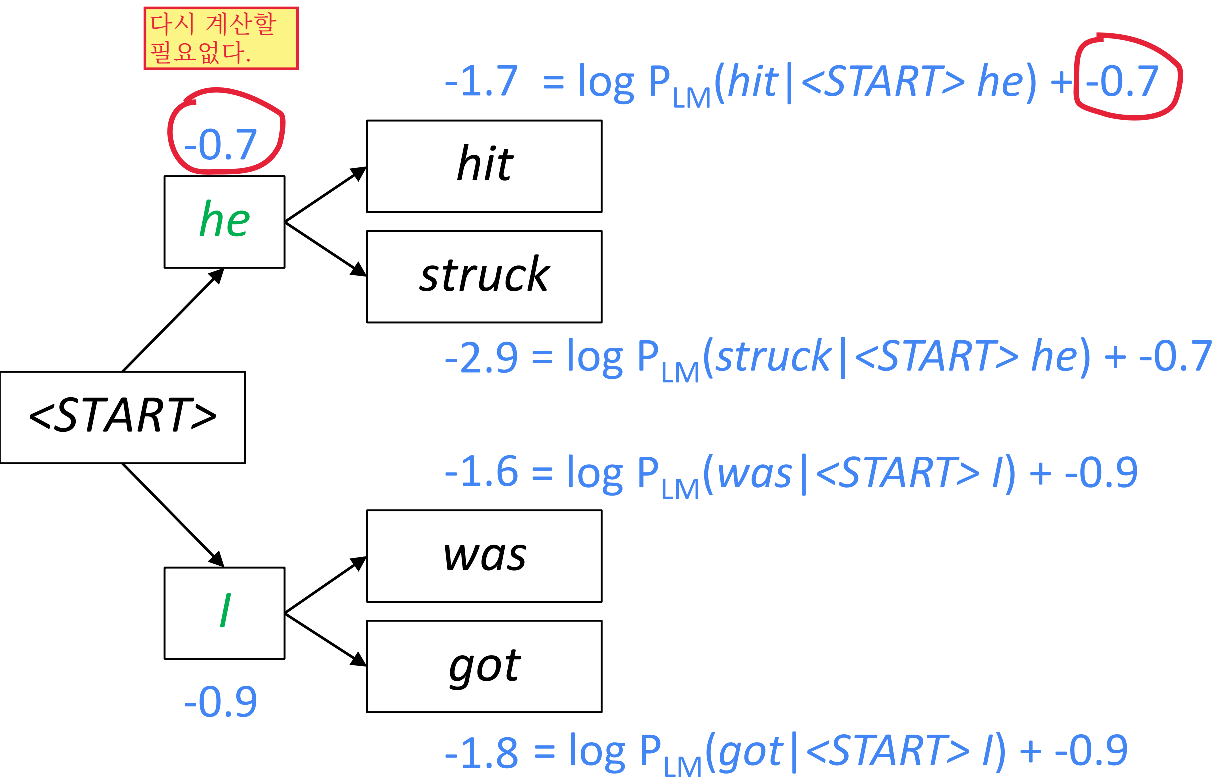



-0.9 = log $P_{LM}$(*I* | *<START>*)

Take top *k* words
and compute scores

# Beam search decoding: example

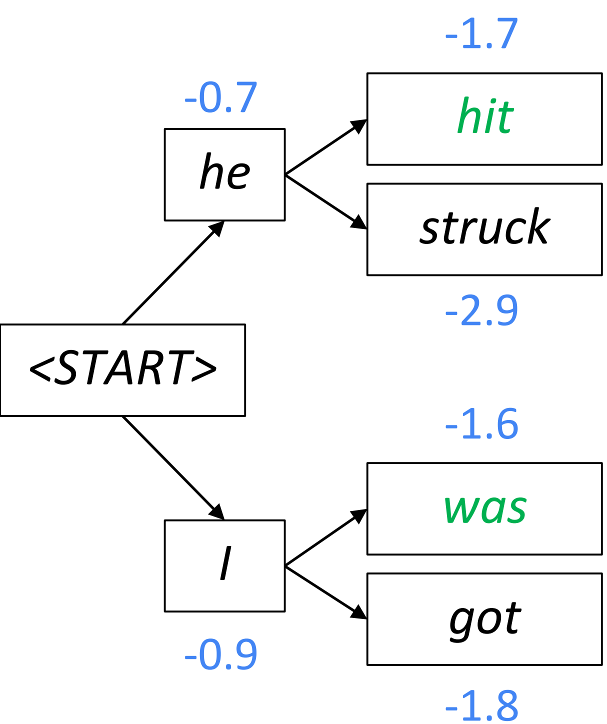
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




For each of the *k* hypotheses, find top *k* next words and calculate scores

# Beam search decoding: example

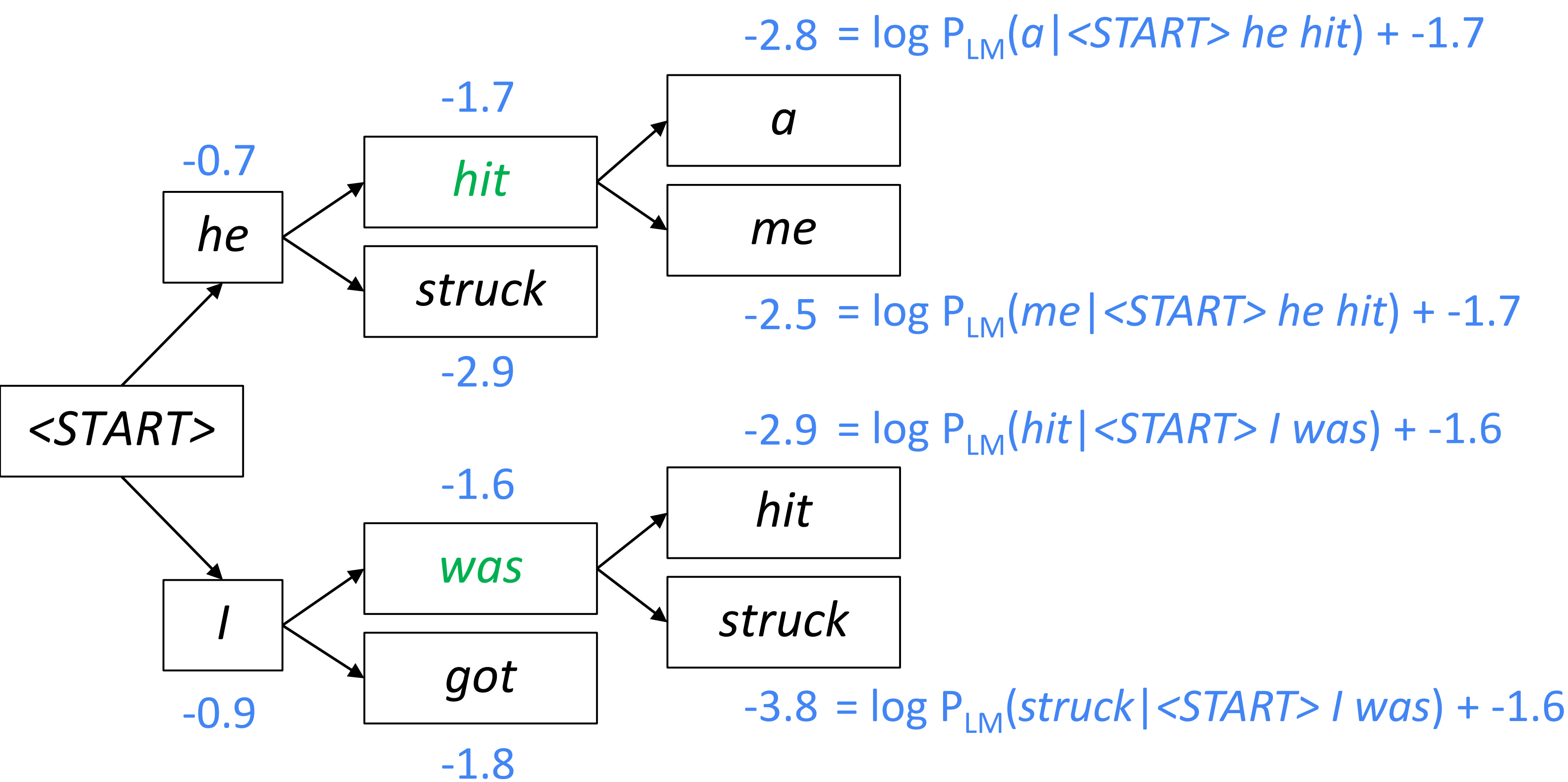
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses, just keep $k$ with highest scores

# Beam search decoding: example

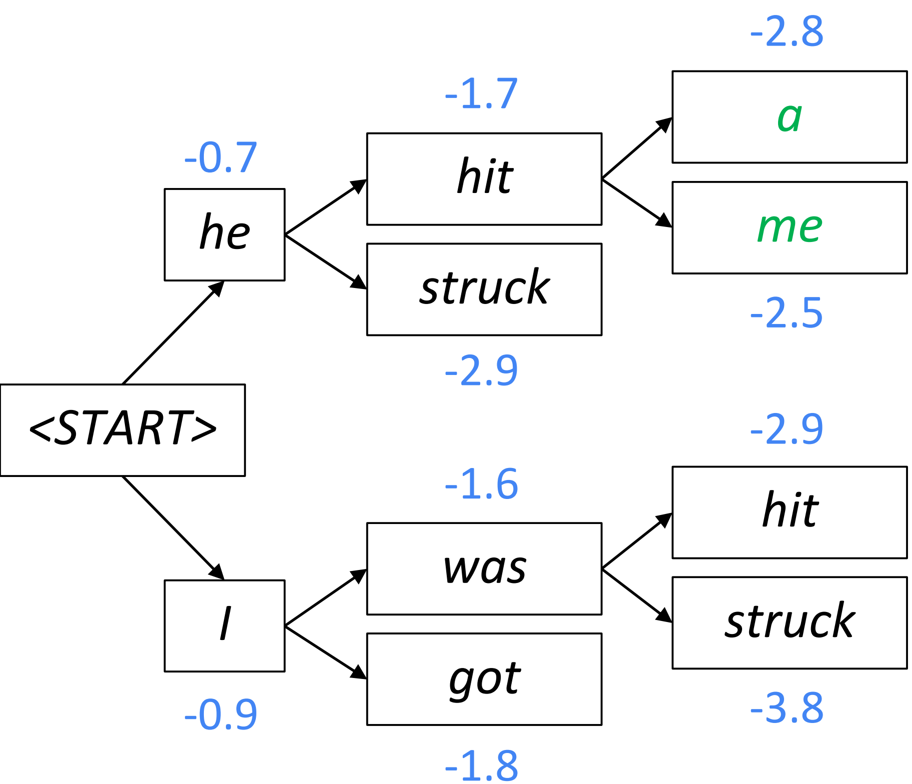
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




For each of the *k* hypotheses, find top *k* next words and calculate scores

# Beam search decoding: example

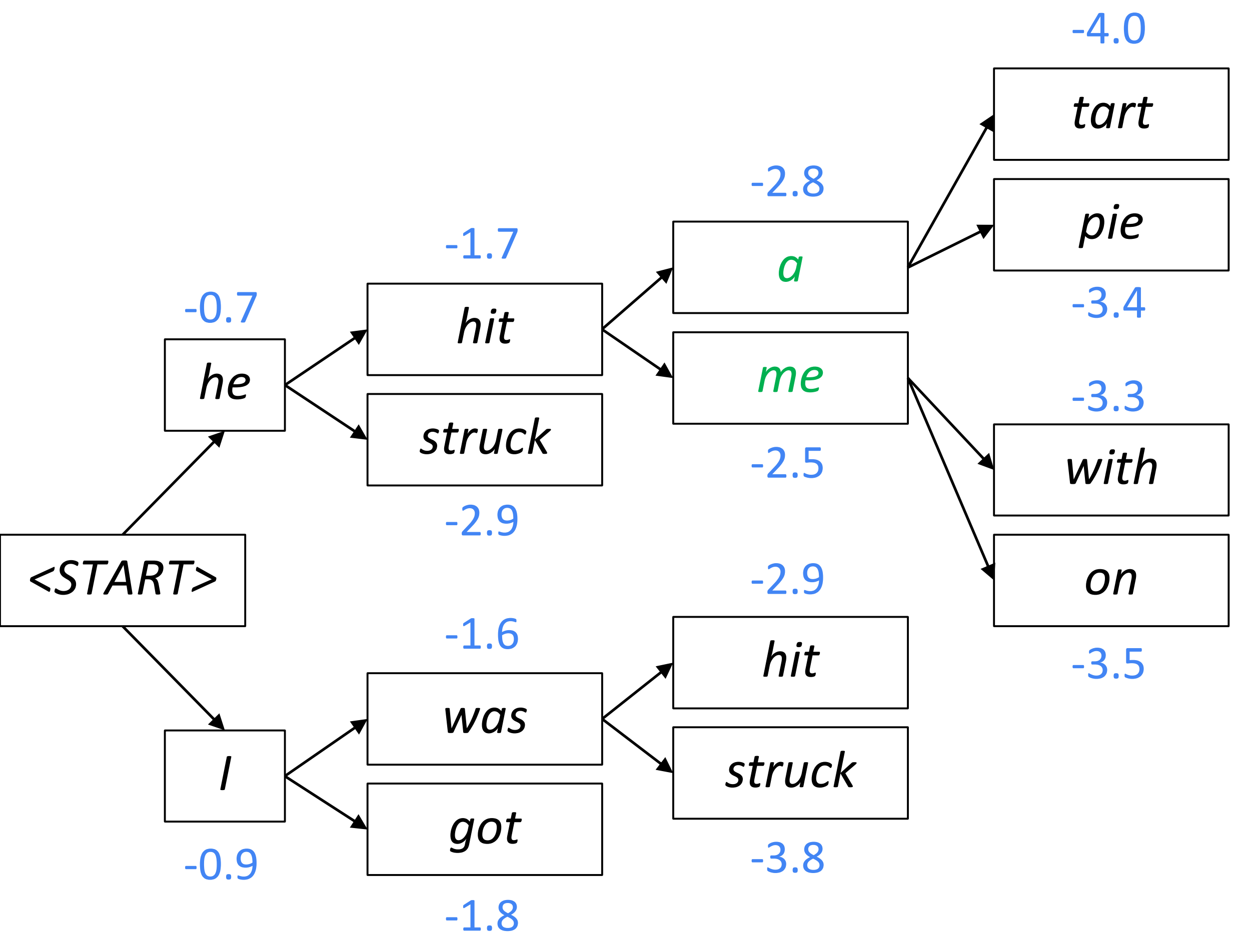
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses,
just keep *k* with highest scores

# Beam search decoding: example

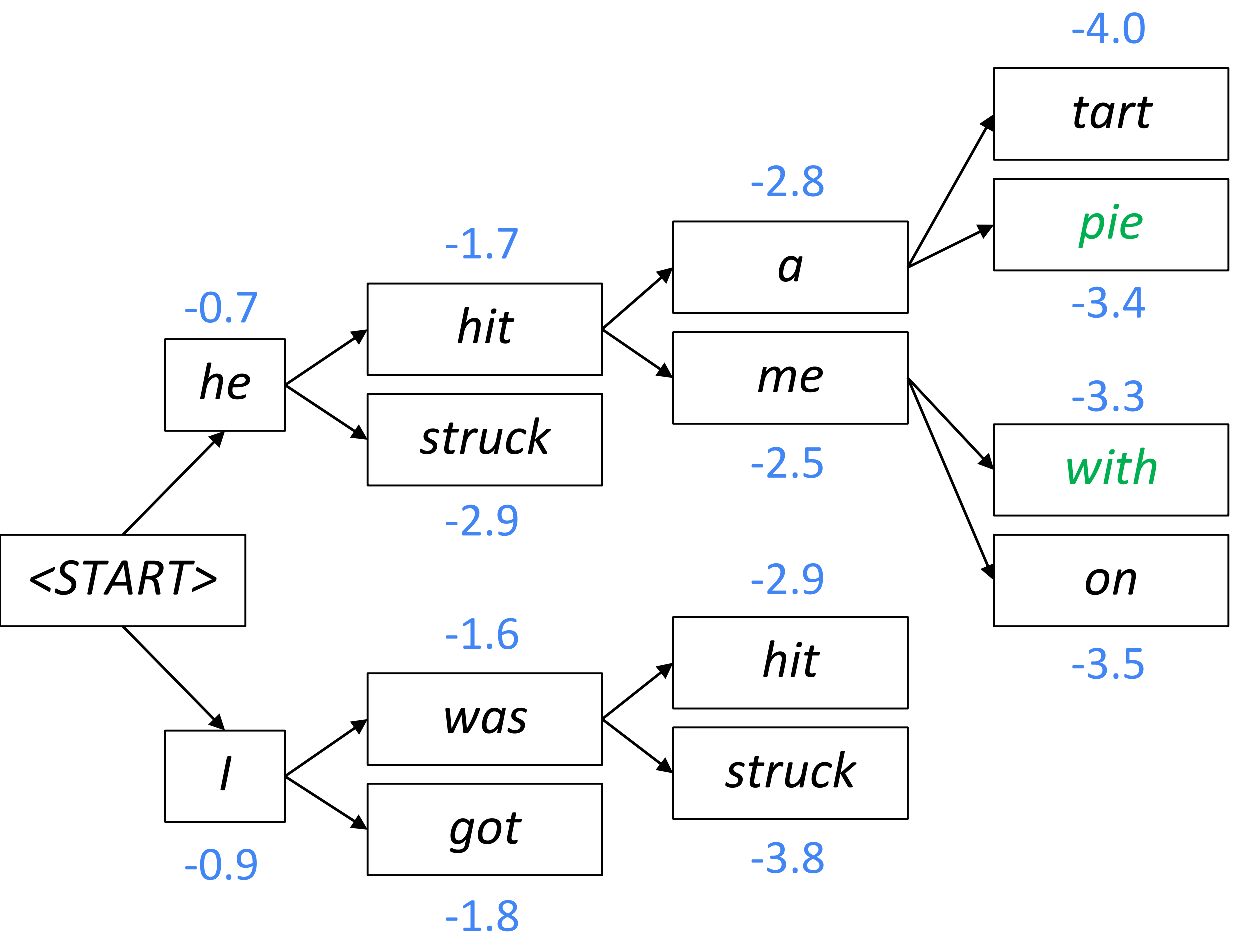
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




For each of the *k* hypotheses, find top *k* next words and calculate scores

# Beam search decoding: example

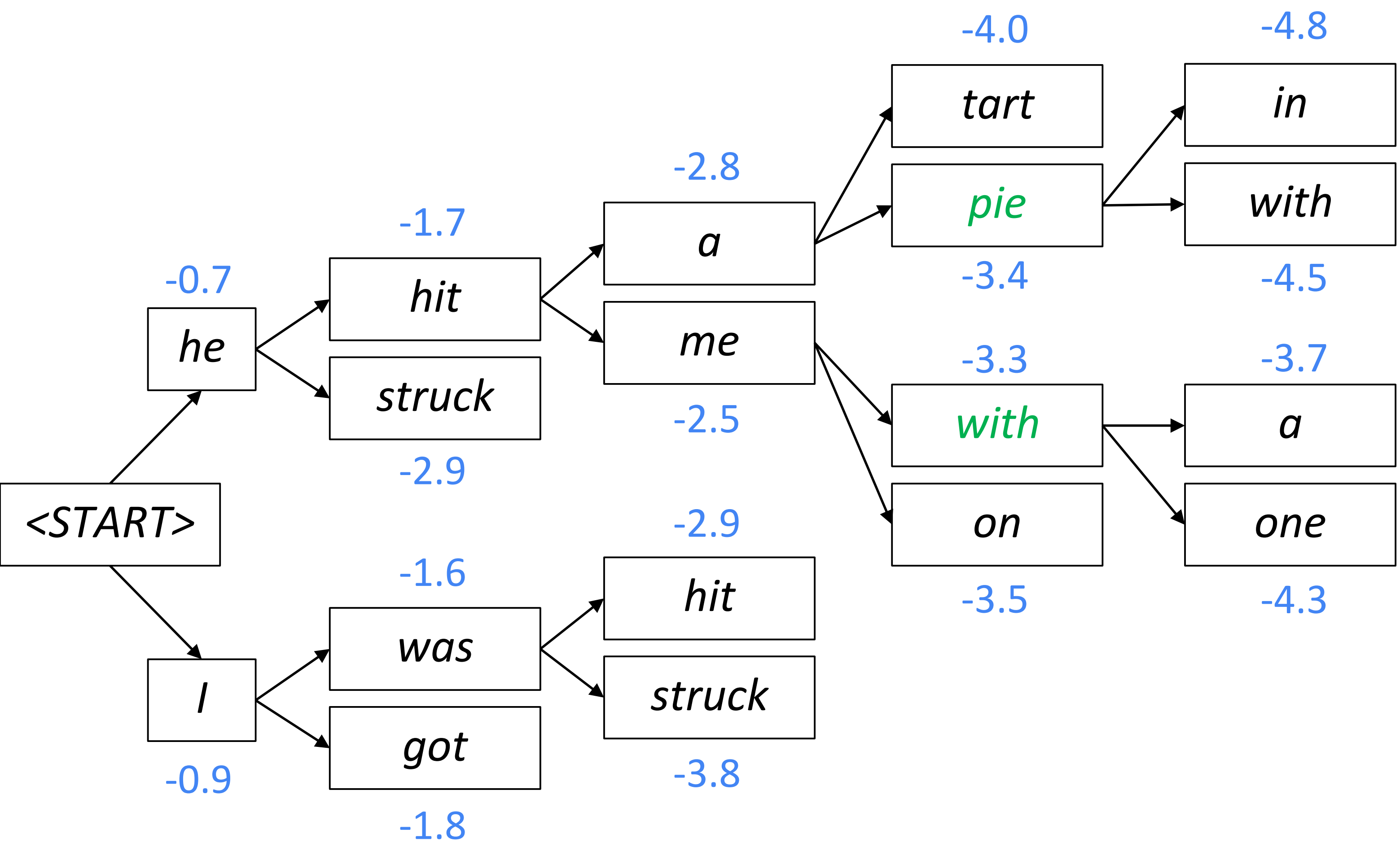
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses, just keep $k$ with highest scores

# Beam search decoding: example

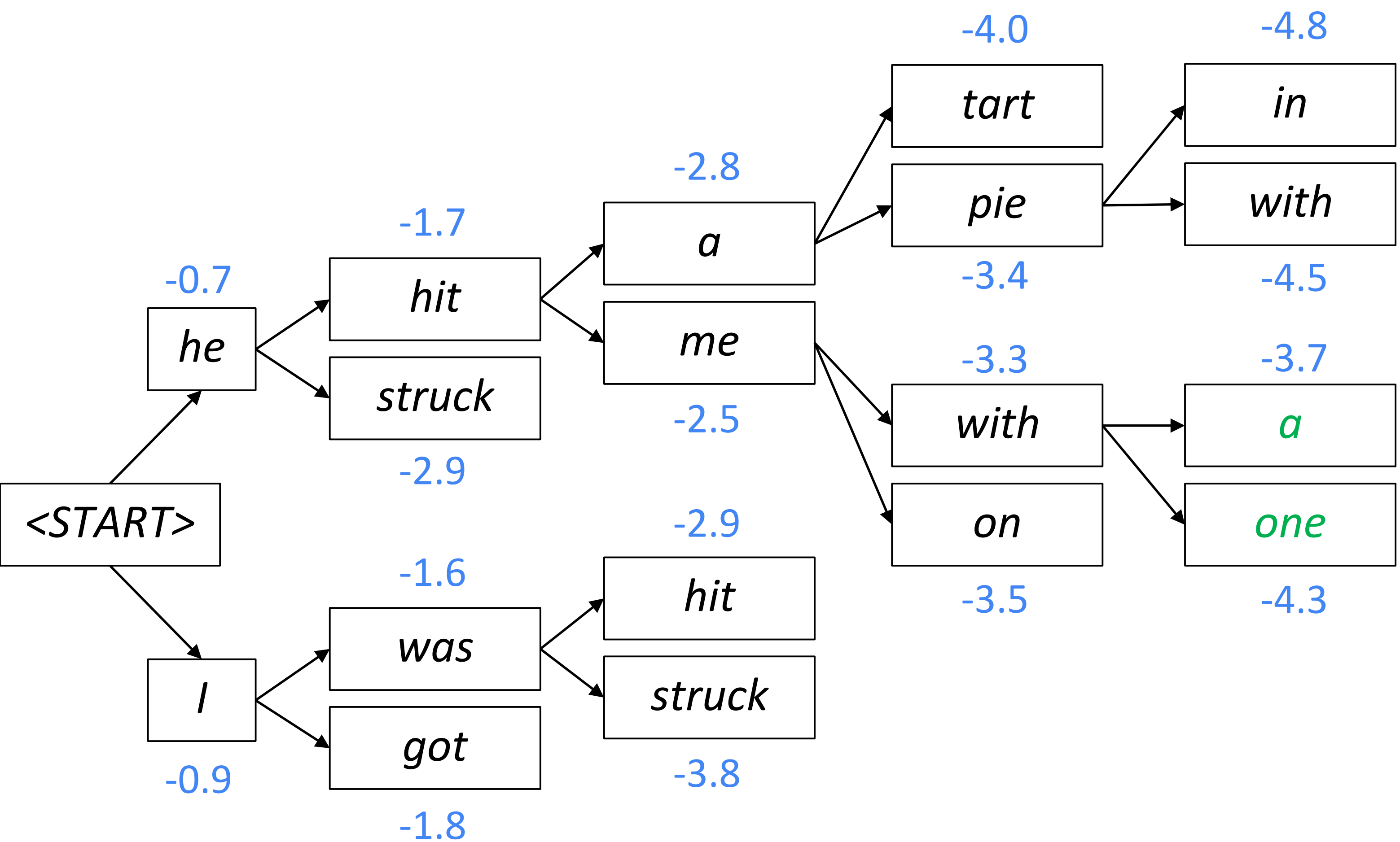
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




For each of the $k$ hypotheses, find top $k$ next words and calculate scores

# Beam search decoding: example

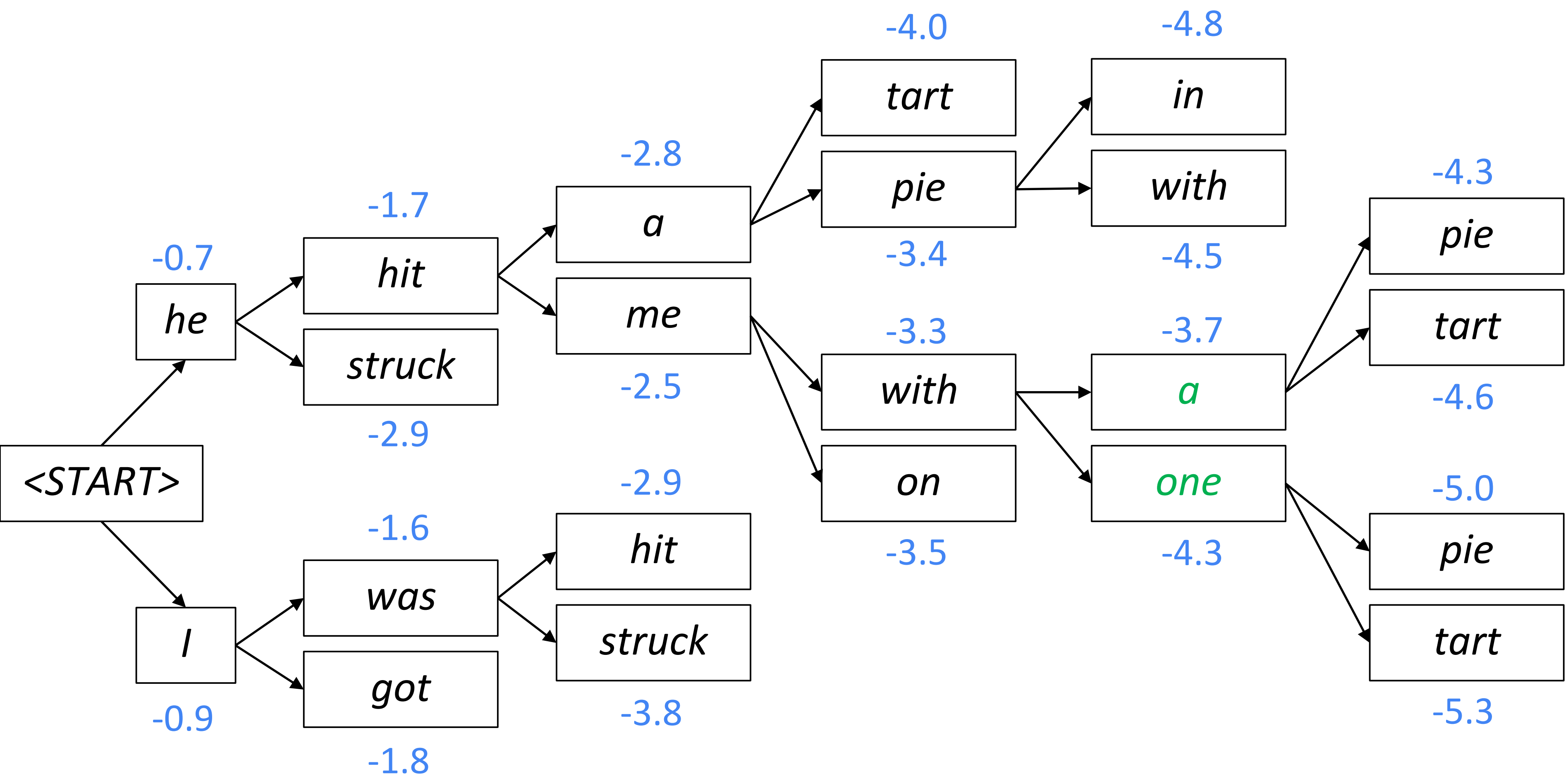
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




Of these $k^2$ hypotheses,
just keep $k$ with highest scores

# Beam search decoding: example

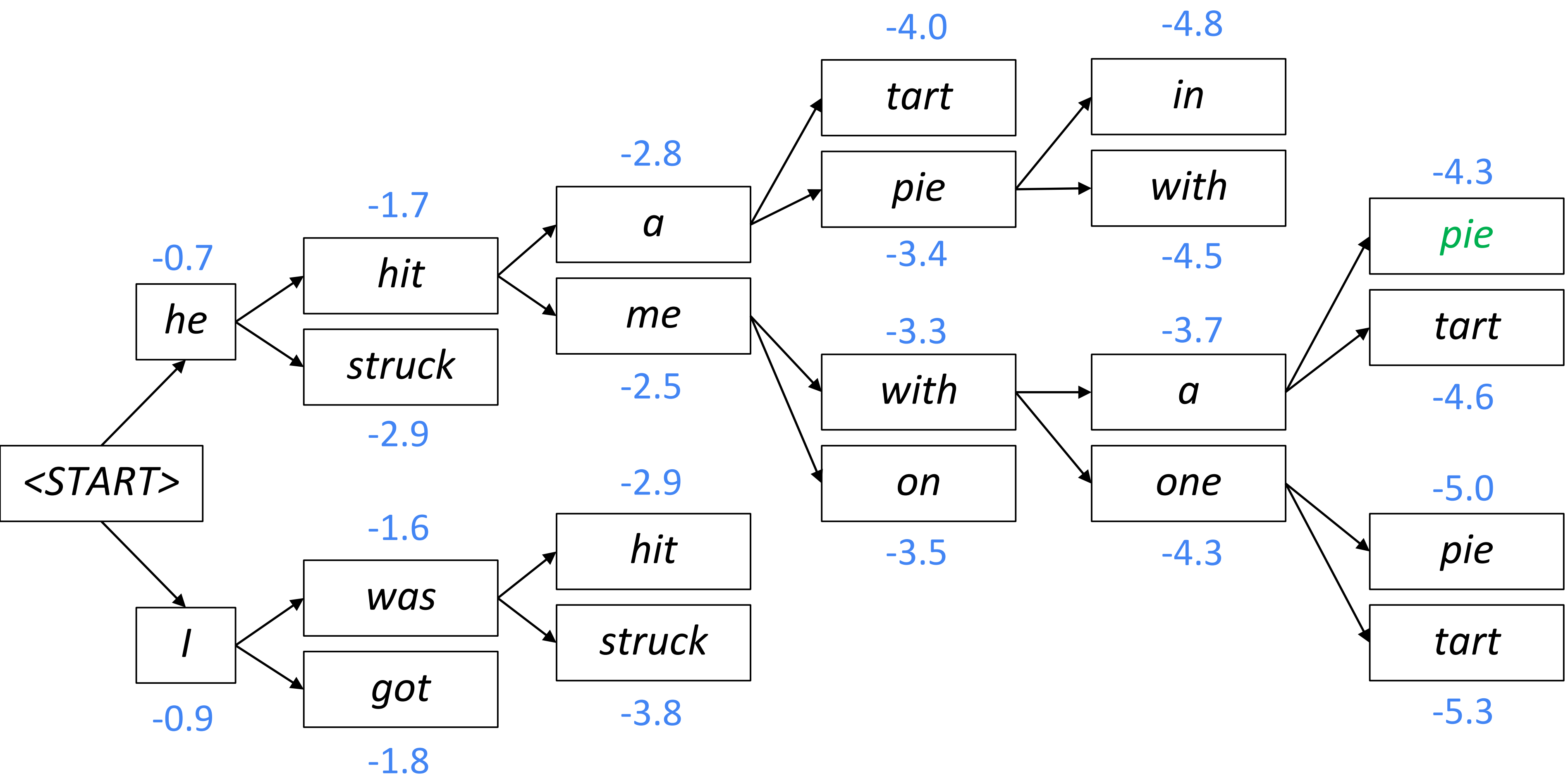
Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$




For each of the *k* hypotheses, find top *k* next words and calculate scores

# Beam search decoding: example

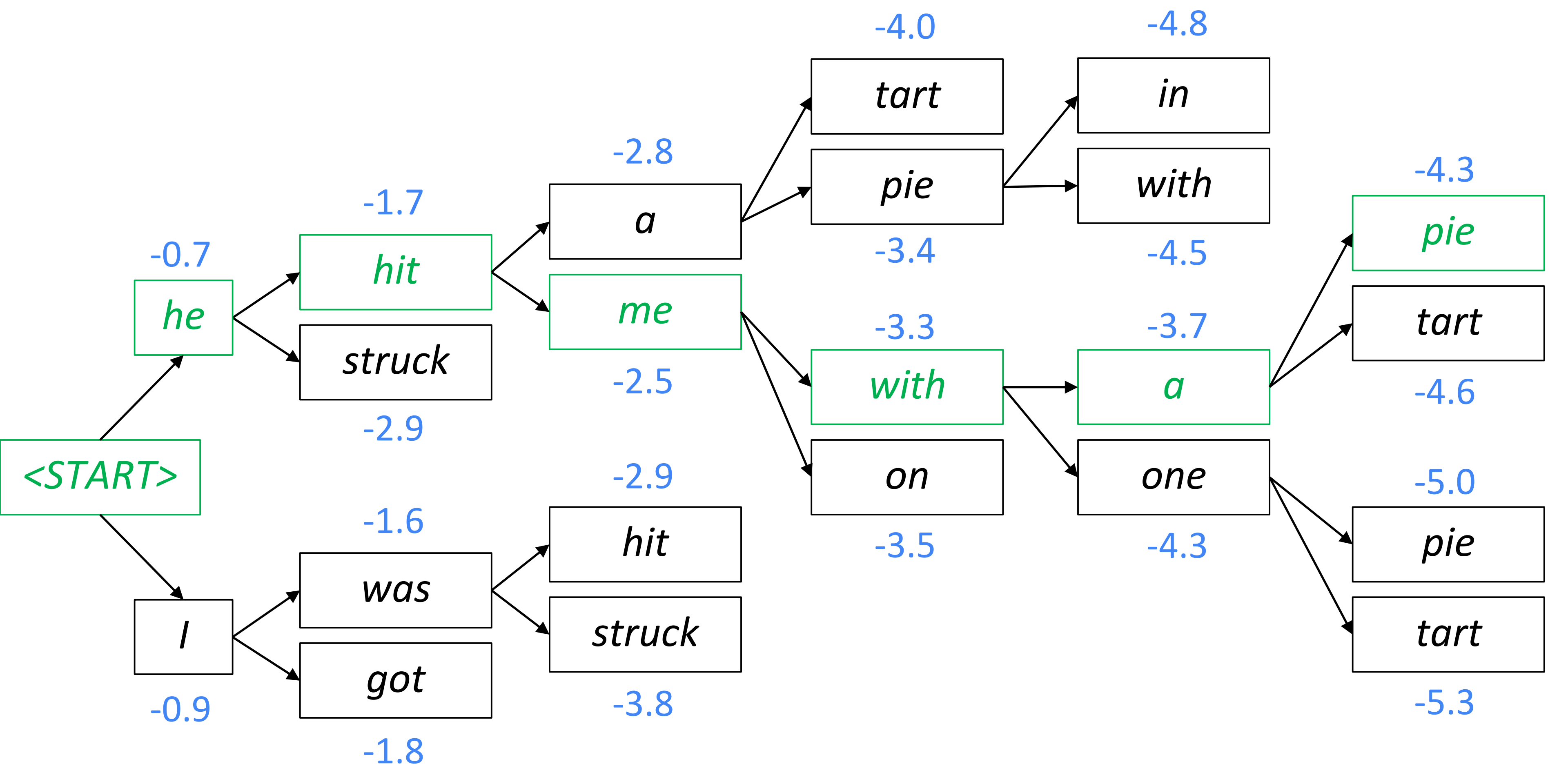
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




This is the top-scoring hypothesis!

# Beam search decoding: example

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




Backtrack to obtain the full hypothesis

# Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: *<START> he hit me with a pie <END>*

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep $T$ (where $T$ is some pre-defined cutoff), or
  - We have at least $n$ completed hypotheses (where $n$ is pre-defined cutoff)

# Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

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

- Problem with this: longer hypotheses have lower scores

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

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

# Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities

More fluent
: RNNs are particularly good at learning Language Model

Better use of context
: better at conditioning on the source sentence

Better use of phrase similarities
: more able to generalize what they learn about phrases and how to translate them

- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

# Disadvantages of NMT?

Compared to SMT:

- NMT is less interpretable
  - Hard to debug

예측된 target sentence가 오류가 있을 때,
오류를 디버깅하기 힘들다.
뉴런 단위로 이해하고 디버깅해야 되는데 쉽지 않다.

반면에, SMT는 subcomponent들이 각자 맡은 역할이 있기 때문에
NMT보다는 디버깅이 쉽고 more interpretable하다.

- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

실무에서..
중요한 특정 케이스에 대해서 100% 성능을 가지고 싶다.
이런 경우 보통 규칙을 시스템에 적용하면 되지만,
(You can't easily control what it's doing on a step - by - step basis.)
NMT는 이러한 규칙을 적용하기가 어렵다.

SMT의 좋은 점을 NMT에 녹여내는 연구도 있다.

# How do we evaluate Machine Translation?

**BLEU** (**Bi**lingual **E**valuation **U**nderstudy)

You'll see BLEU in detail in Assignment 4!

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

n - gram precision만 사용하면, 짧은 길이의 문장일수록 높은 점수를 준다. 밸런스를 맞추기 위해 짧은 길이에 패널티를 준다.

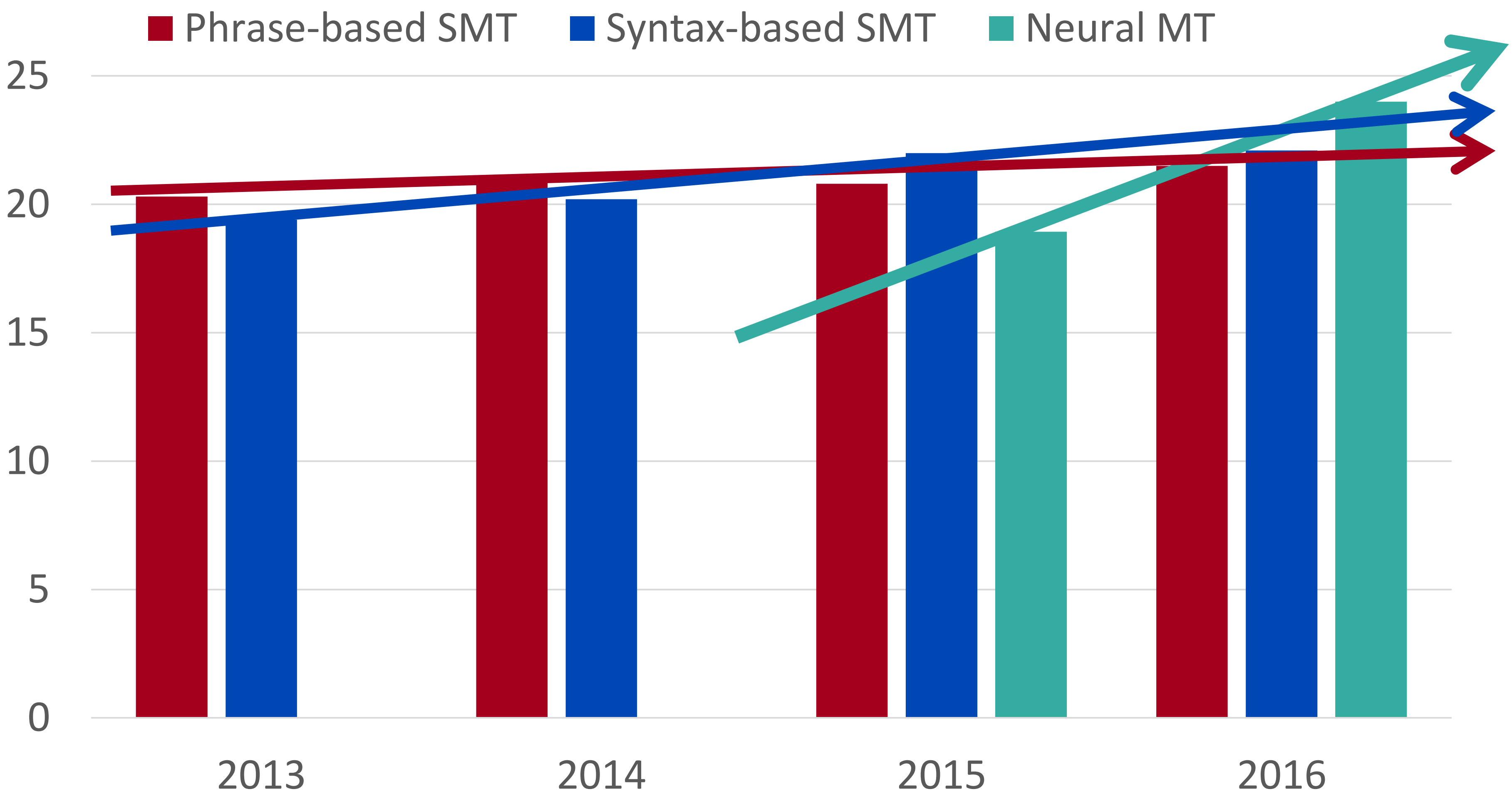
- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹



**Source:** " BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002.

# MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]




SMT는 매년 조금씩밖에 좋아지지 못했다.
반면, NMT는 가파르게 성장했다.



**Source**: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf

# NMT: the biggest success story of NLP Deep Learning

Neural Machine Translation went from a fringe research activity in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published

2014 ~ 2016년이
자연어처리 딥러닝 기술에 있어서
인정받게 된 기간이었다고 볼 수 있다.

- **2016**: Google Translate switches from SMT to NMT

- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

# So is Machine Translation solved?

- **Nope!**
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs

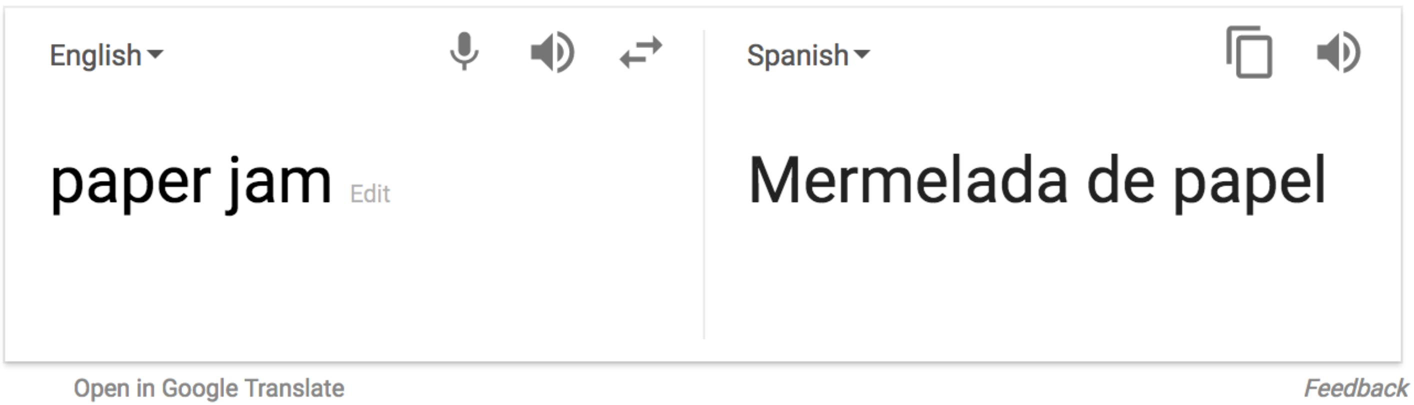
OOV 문제는 자연어처리에서 고질적인 문제이다.

학습한 데이터가 위키피디아 같은 formal text인 반면, 테스트할 데이터가 트윗트와 같은 informal text이면 잘 안 될 것이다.

**Further reading:** *"Has AI surpassed humans at translation? Not even close!"*
https://www.skynettoday.com/editorials/state_of_nmt

# So is Machine Translation solved?

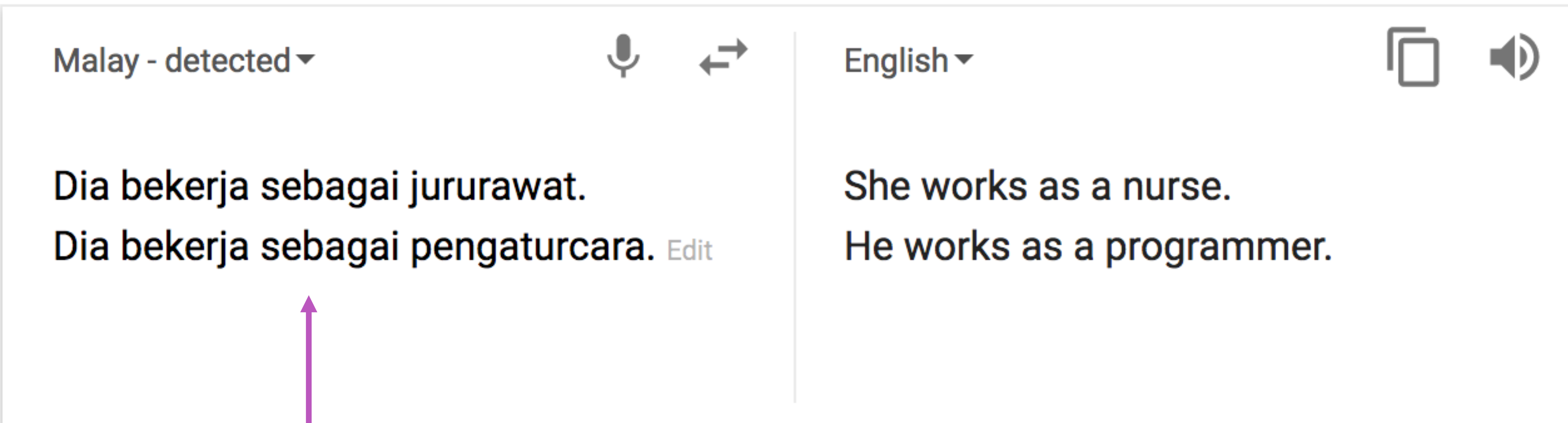
- **Nope!**
- Using common sense is still hard




?

# So is Machine Translation solved?

- **Nope!**
- NMT picks up biases in training data




Didn’t specify gender

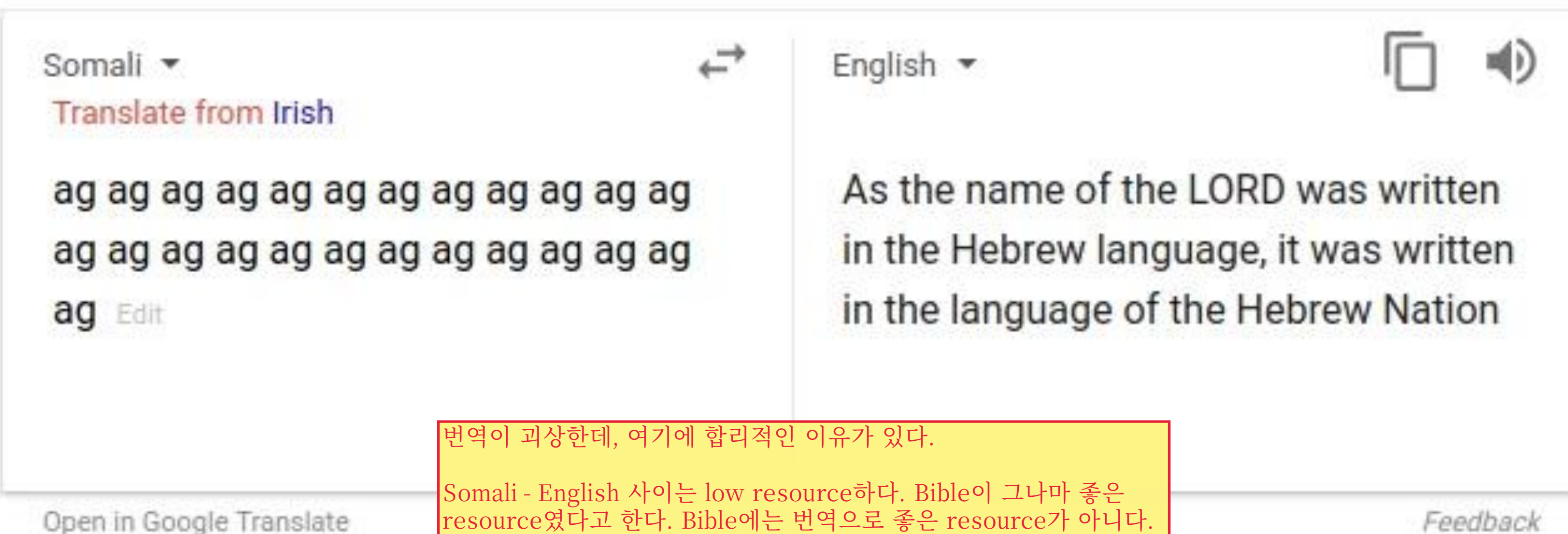
문장 레벨에서 문제가 생길 수 있다.
source sentence에서 gender에 대한 아무런 정보가 없는데 (ex. they), target sentence에서 특정 gender로 말하고 있다. 이유는 학습 데이터에는 they 대명사보다는 she, he가 많기 때문이다. 또는 female nurse, male programmer 예시가 많다.

이는 전형적인 data - driven approach에서 만연히 발생하는 문제점이다.

**Source:** https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c

# So is Machine Translation solved?

- Nope!
- Uninterpretable systems do strange things




번역이 괴상한데, 여기에 합리적인 이유가 있다.

Somali - English 사이는 low resource하다. Bible이 그나마 좋은 resource였다고 한다. Bible에는 번역으로 좋은 resource가 아니다.

입력 자체가 매우 노이즈하다. NMT가 전혀 좋은 condition을 받지 못했다. 이러면 NMT는 단지 English Language Model로써의 의무만 다한다. (generating random text)

이러한 오류가 발생하기 전까지는 모른다. NMT는 uninterpretable 하기 때문에..

**Picture source**: https://www.vice.com/en_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies

**Explanation**: https://www.skynettoday.com/briefs/google-nmt-prophecies

# NMT research continues

NMT is the **flagship task** for NLP Deep Learning

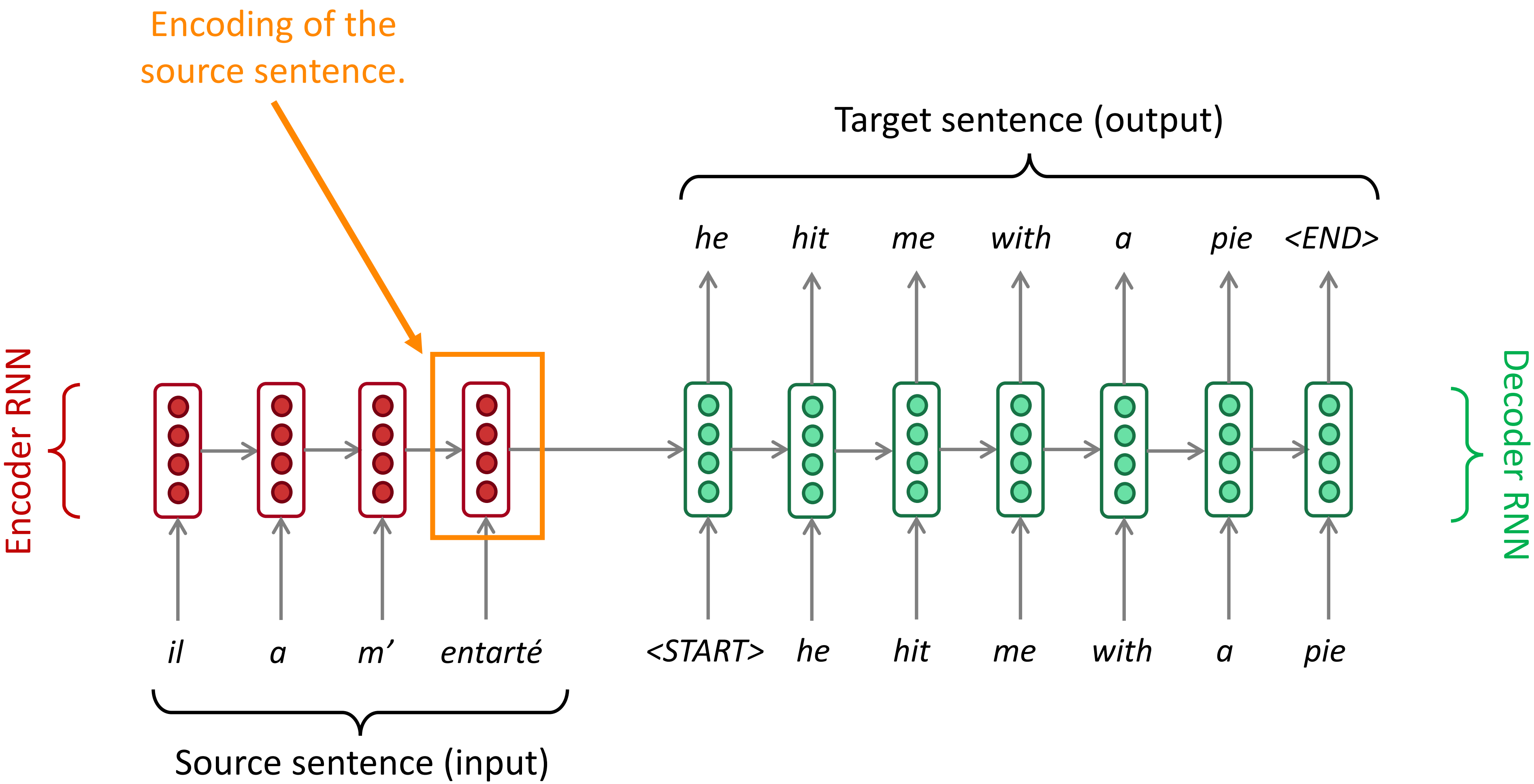
- NMT research has pioneered many of the recent innovations of NLP Deep Learning

- In **2019**: NMT research continues to thrive
  - Researchers have found *many, many* improvements to the "vanilla" seq2seq NMT system we've presented today
  - But one improvement is so integral that it is the new vanilla…

**ATTENTION**

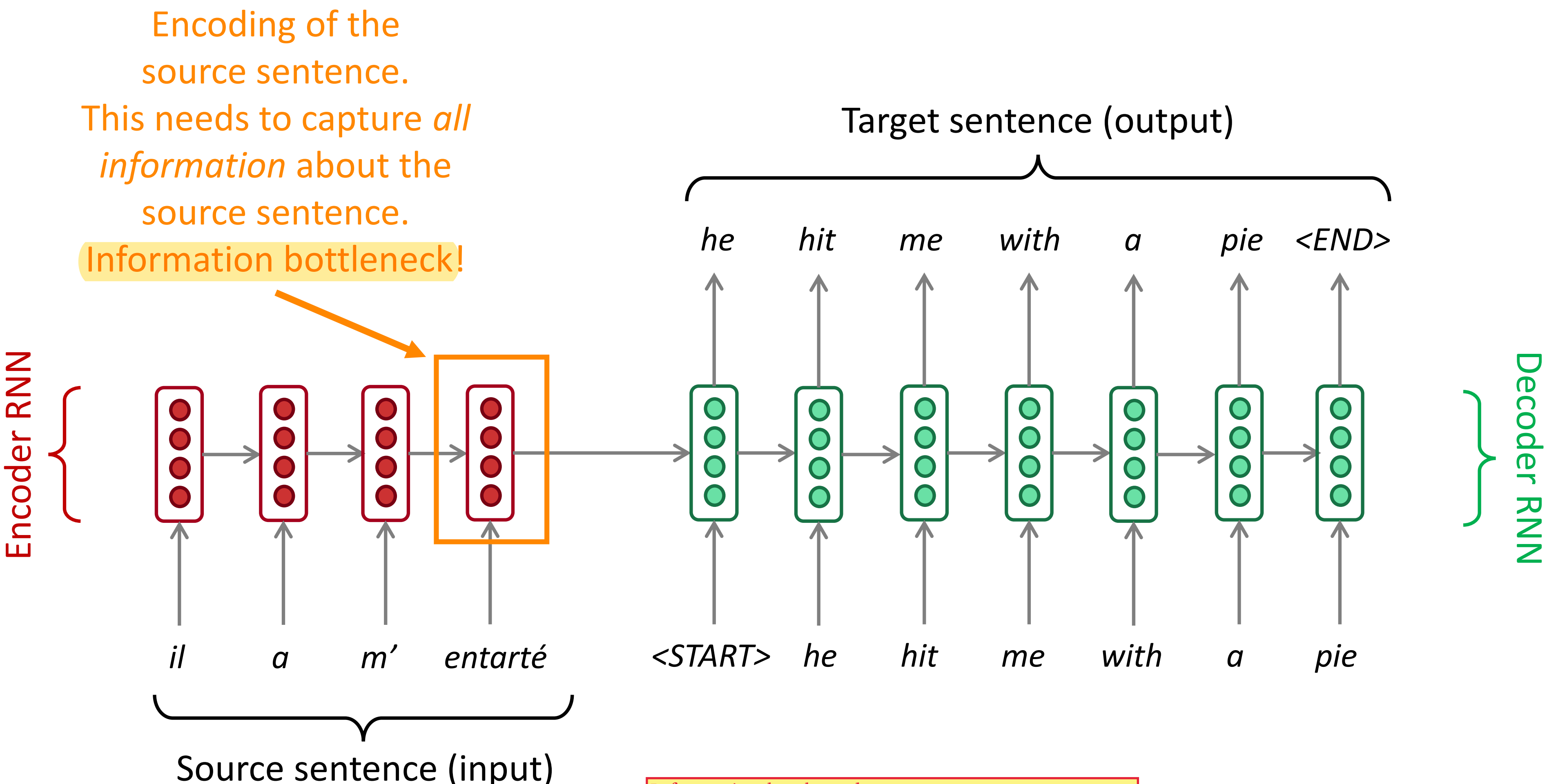
# Section 3: Attention

# Sequence-to-sequence: the bottleneck problem




**Problems with this architecture?**

# Sequence-to-sequence: the bottleneck problem




Information bottleneck
It's putting kind of too much pressure on a single vector to be a good representation of the encoder.

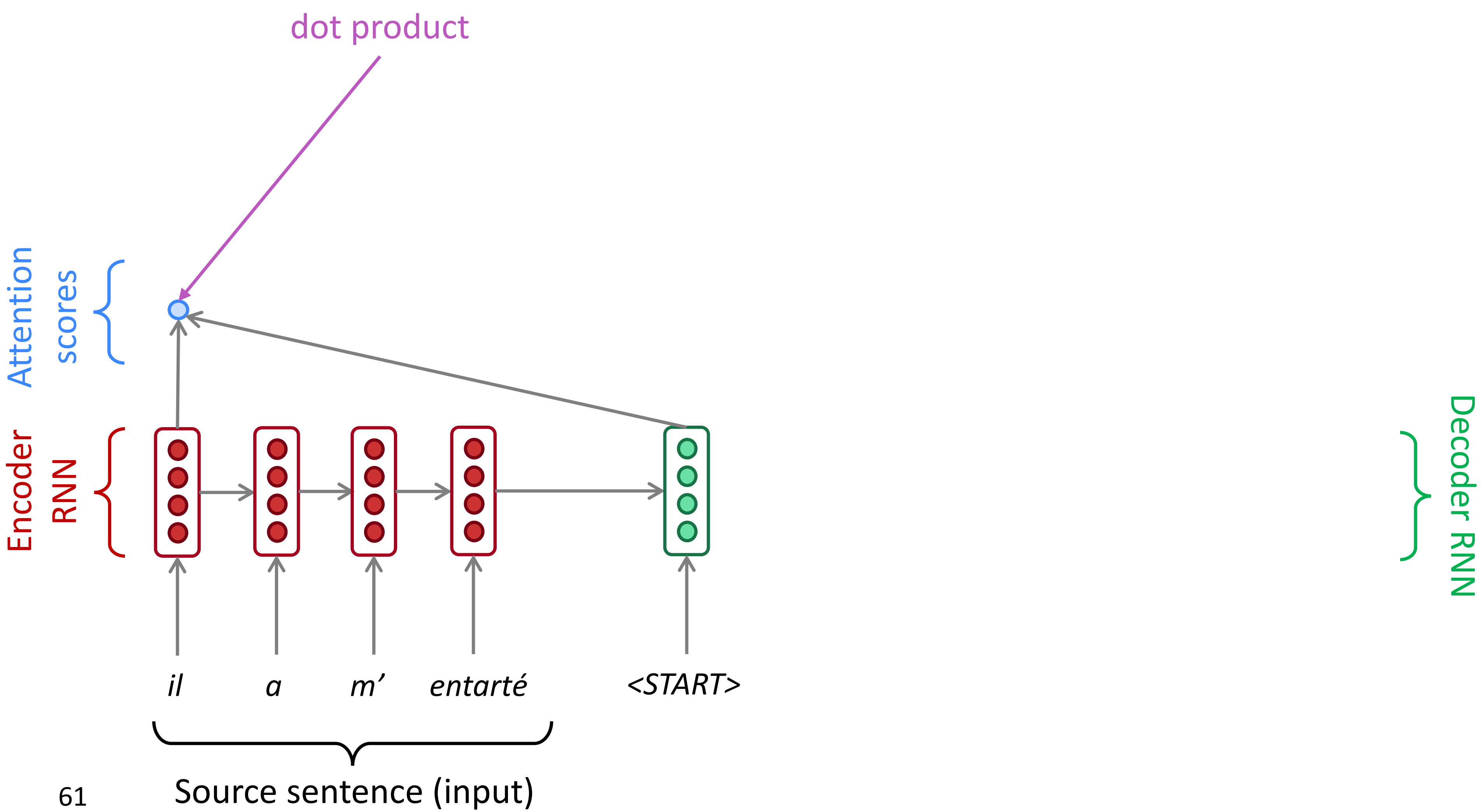
# Attention

- **Attention** provides a solution to the bottleneck problem.

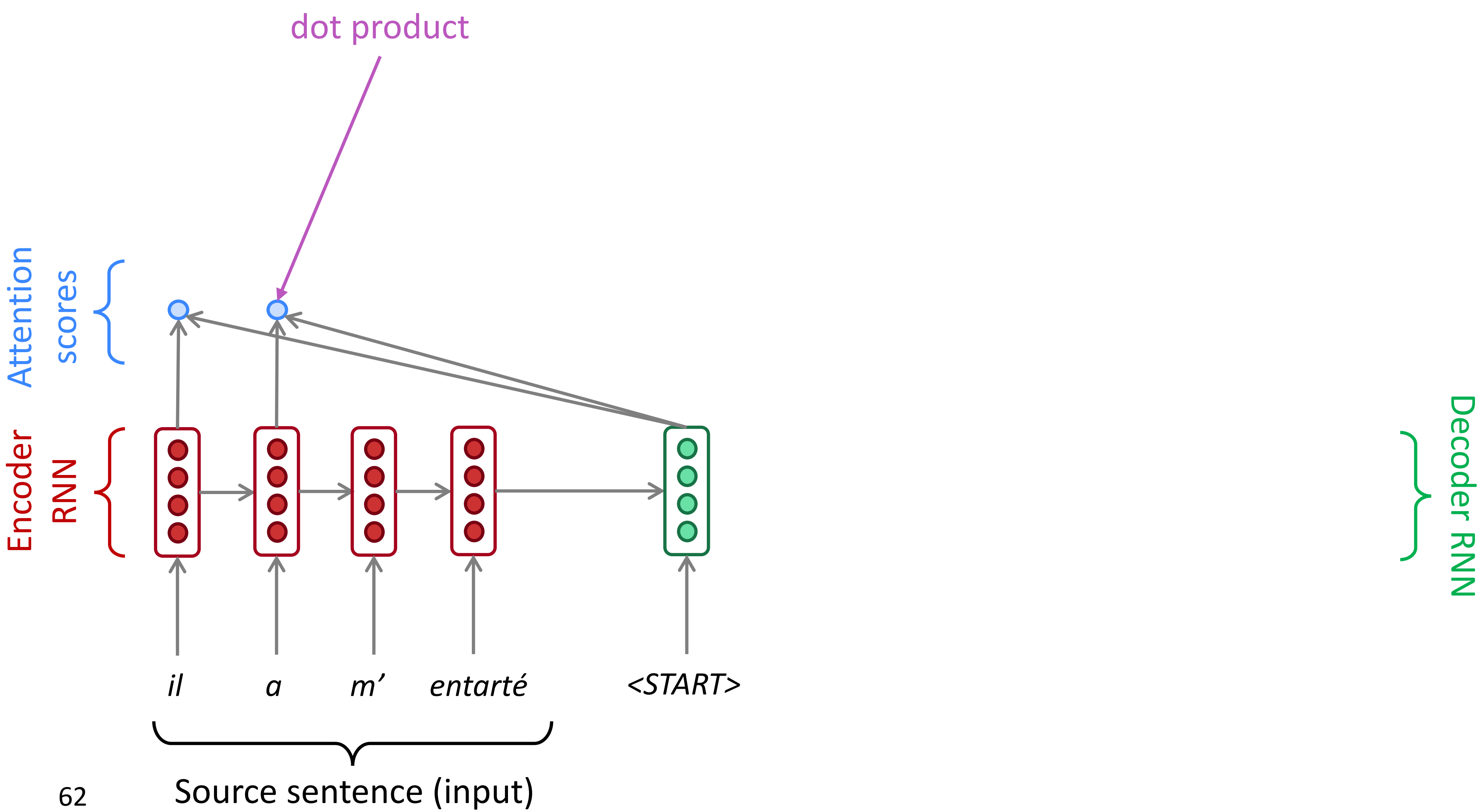
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First we will show via diagram (no equations), then we will show with equations

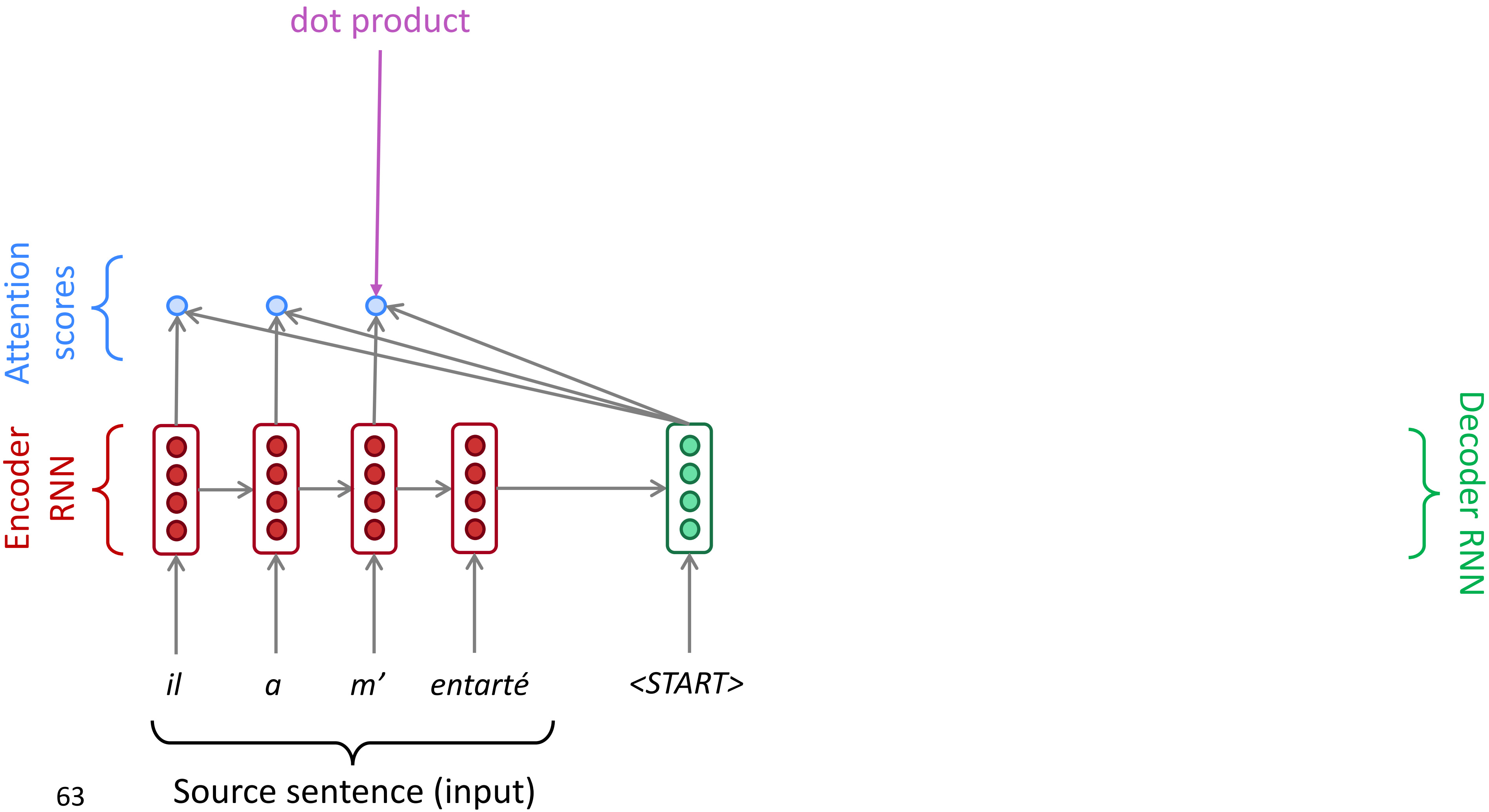
# Sequence-to-sequence with attention



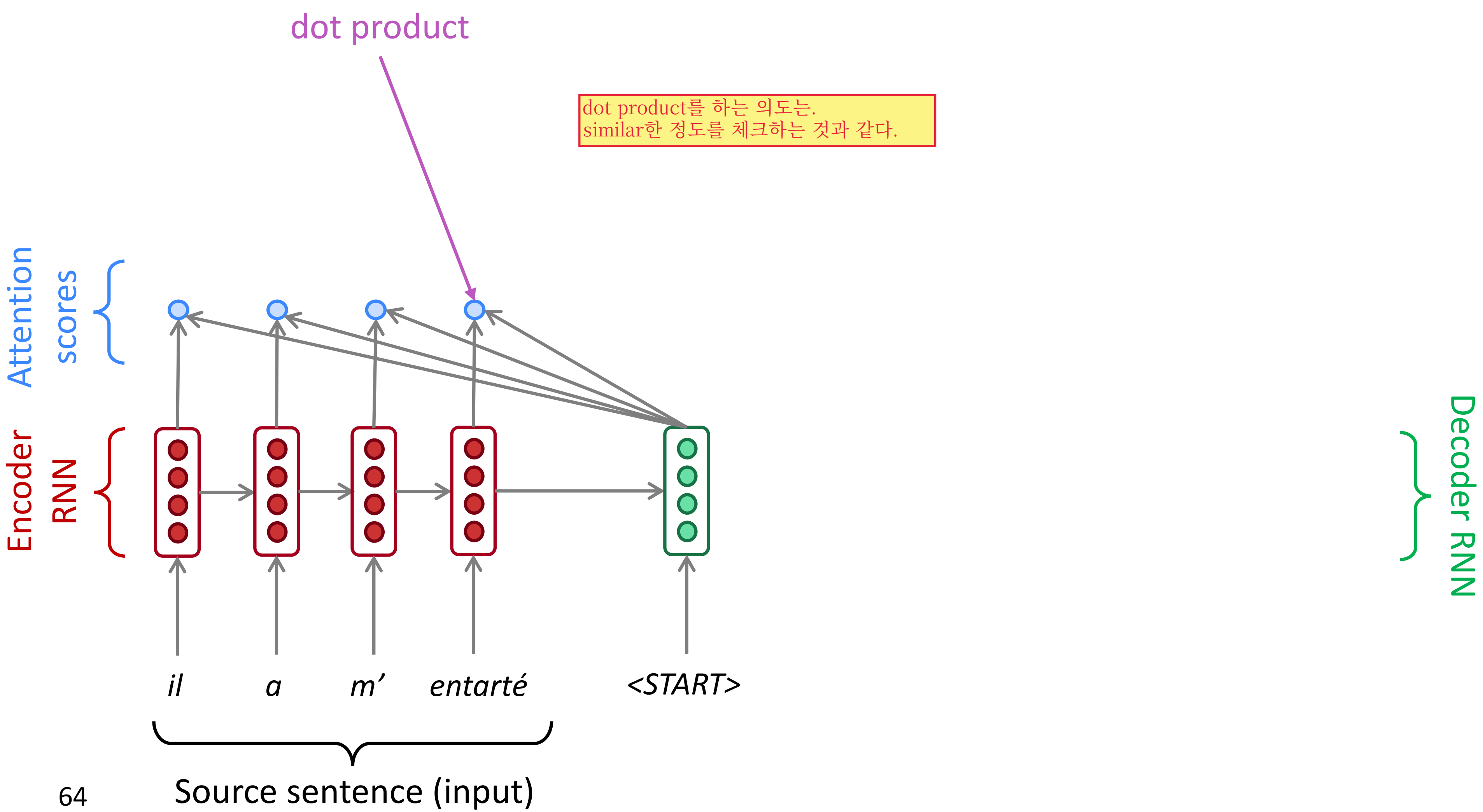
# Sequence-to-sequence with attention



# Sequence-to-sequence with attention

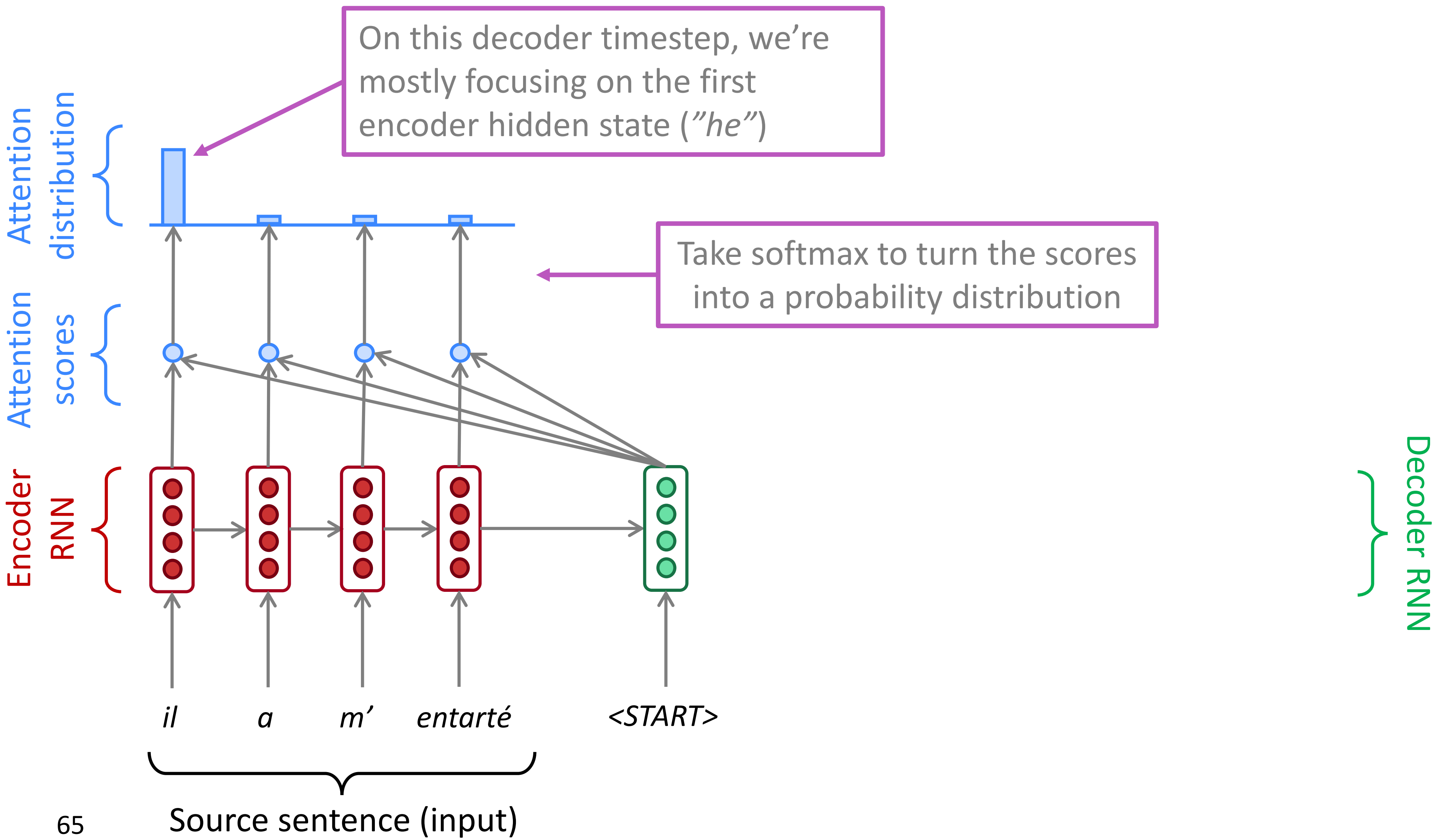


# Sequence-to-sequence with attention

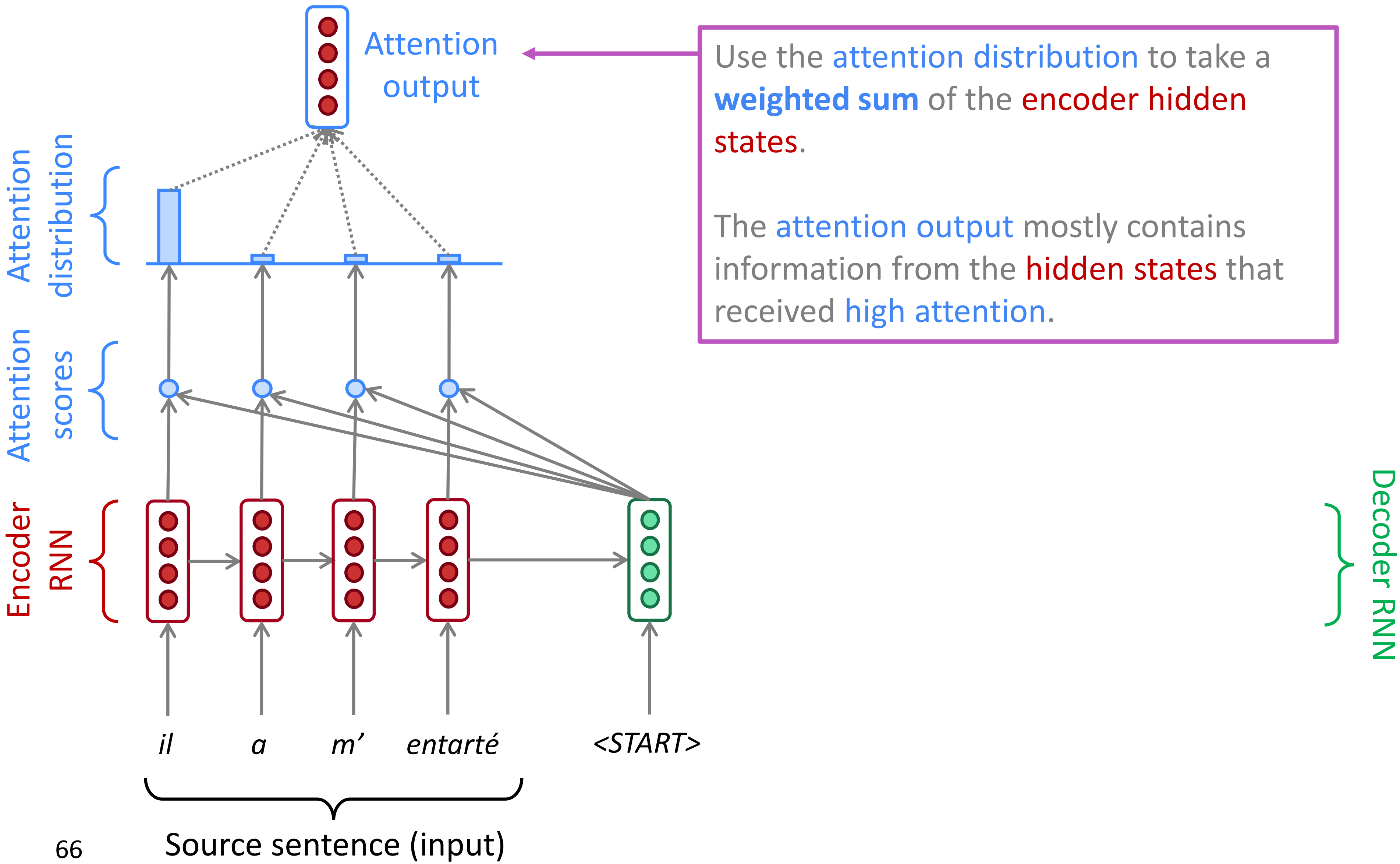



dot product를 하는 의도는.
similar한 정도를 체크하는 것과 같다.

# Sequence-to-sequence with attention



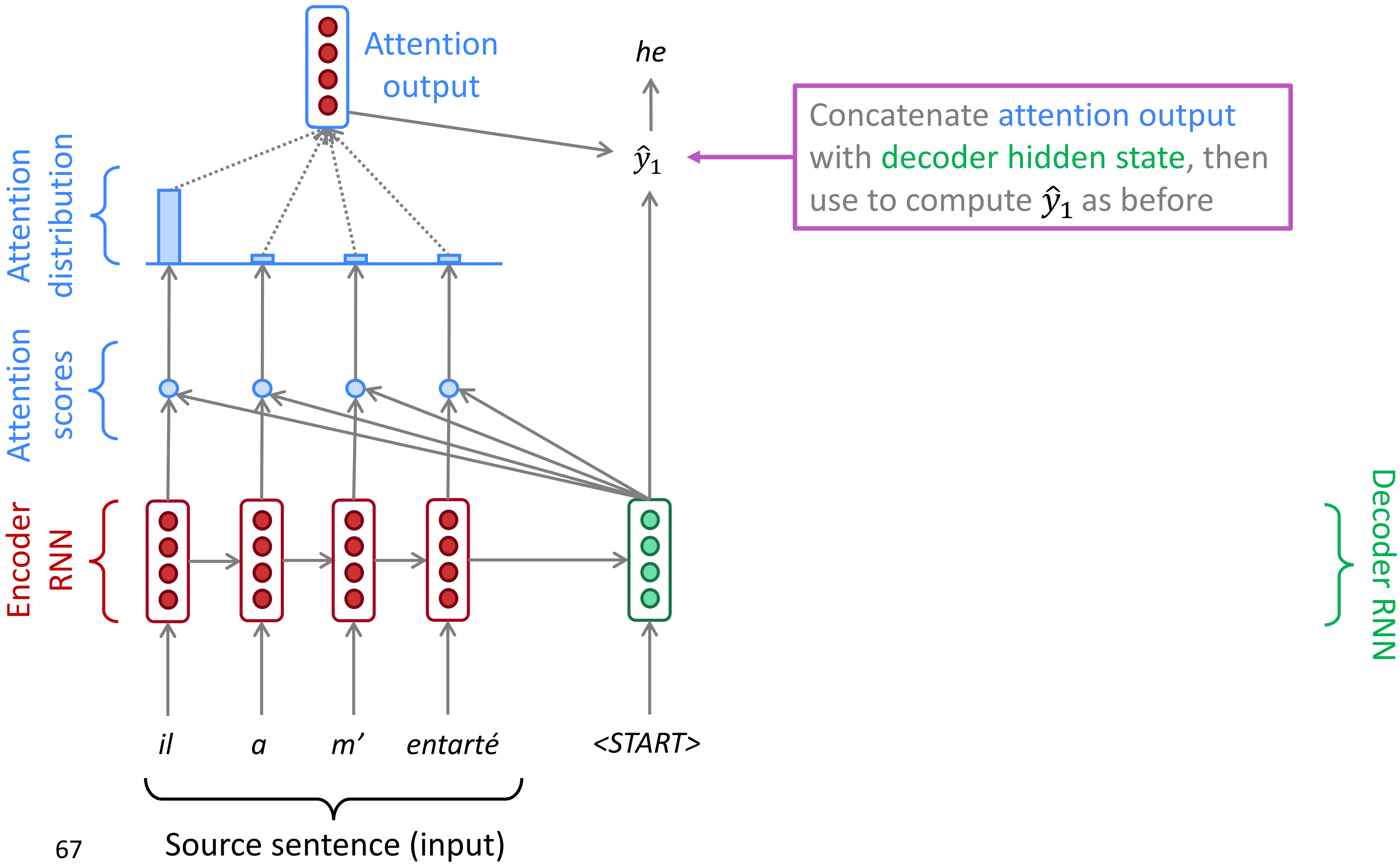
# Sequence-to-sequence with attention




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

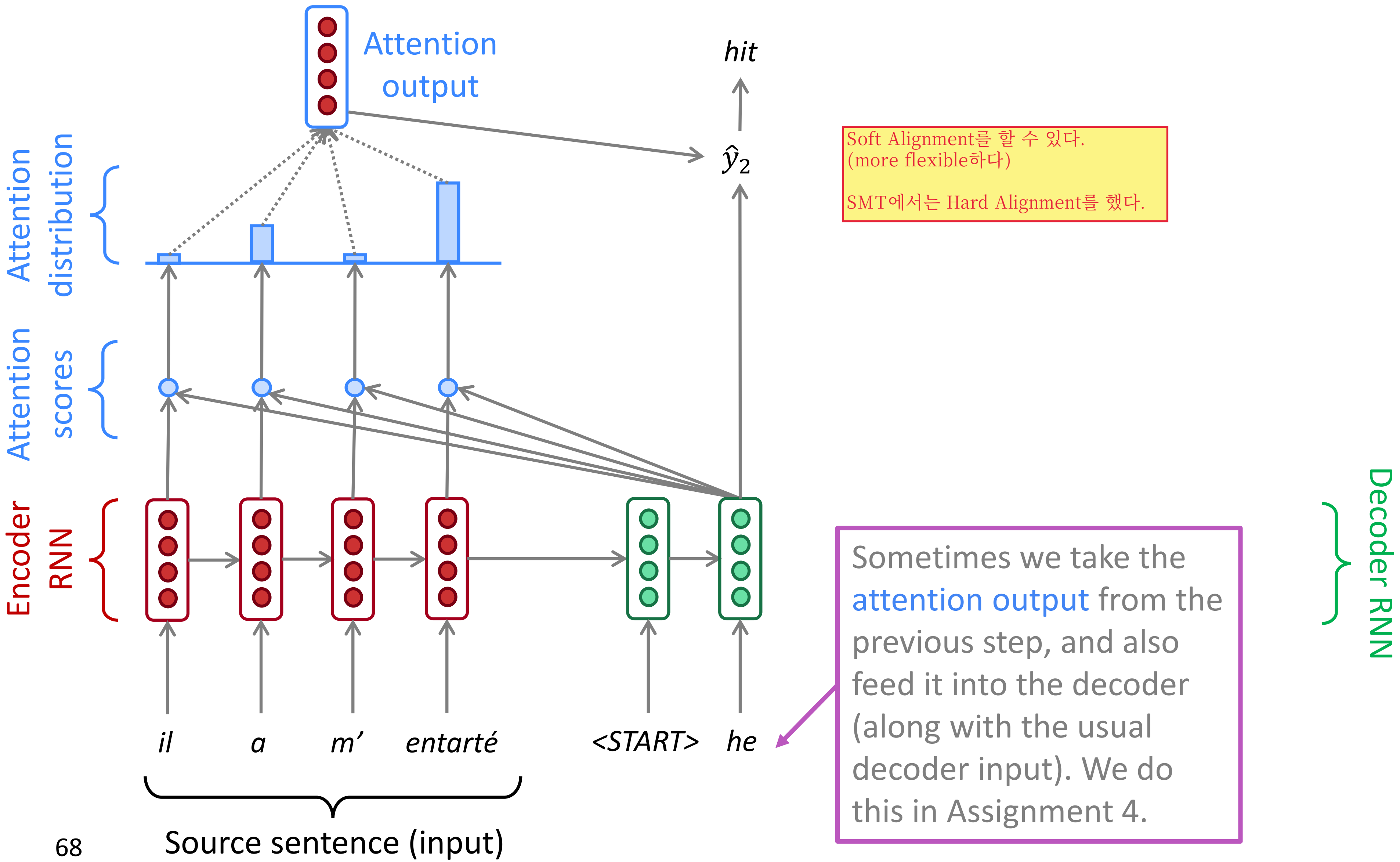
The attention output mostly contains information from the hidden states that received high attention.

# Sequence-to-sequence with attention




Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

# Sequence-to-sequence with attention

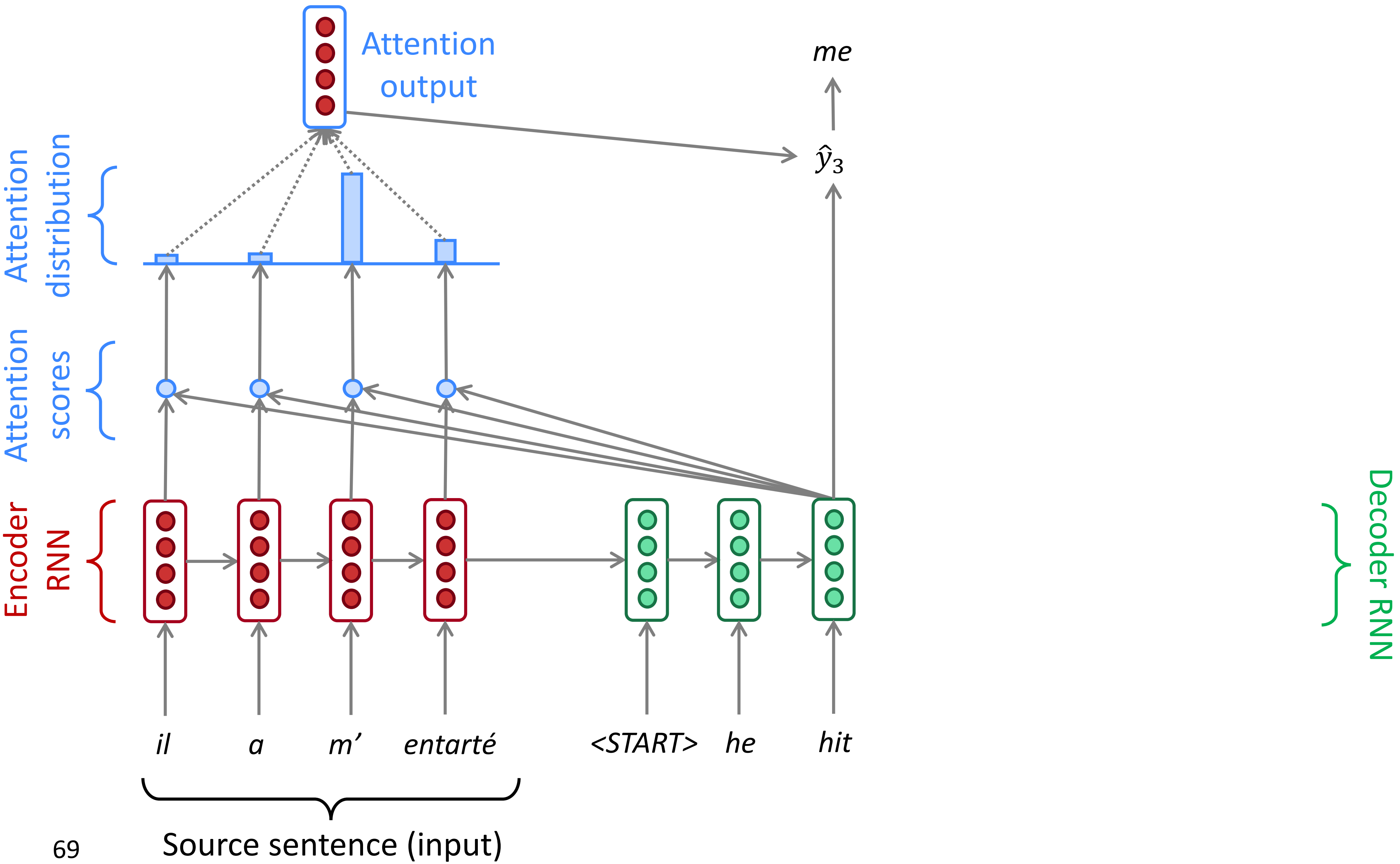



Soft Alignment를 할 수 있다.
(more flexible하다)

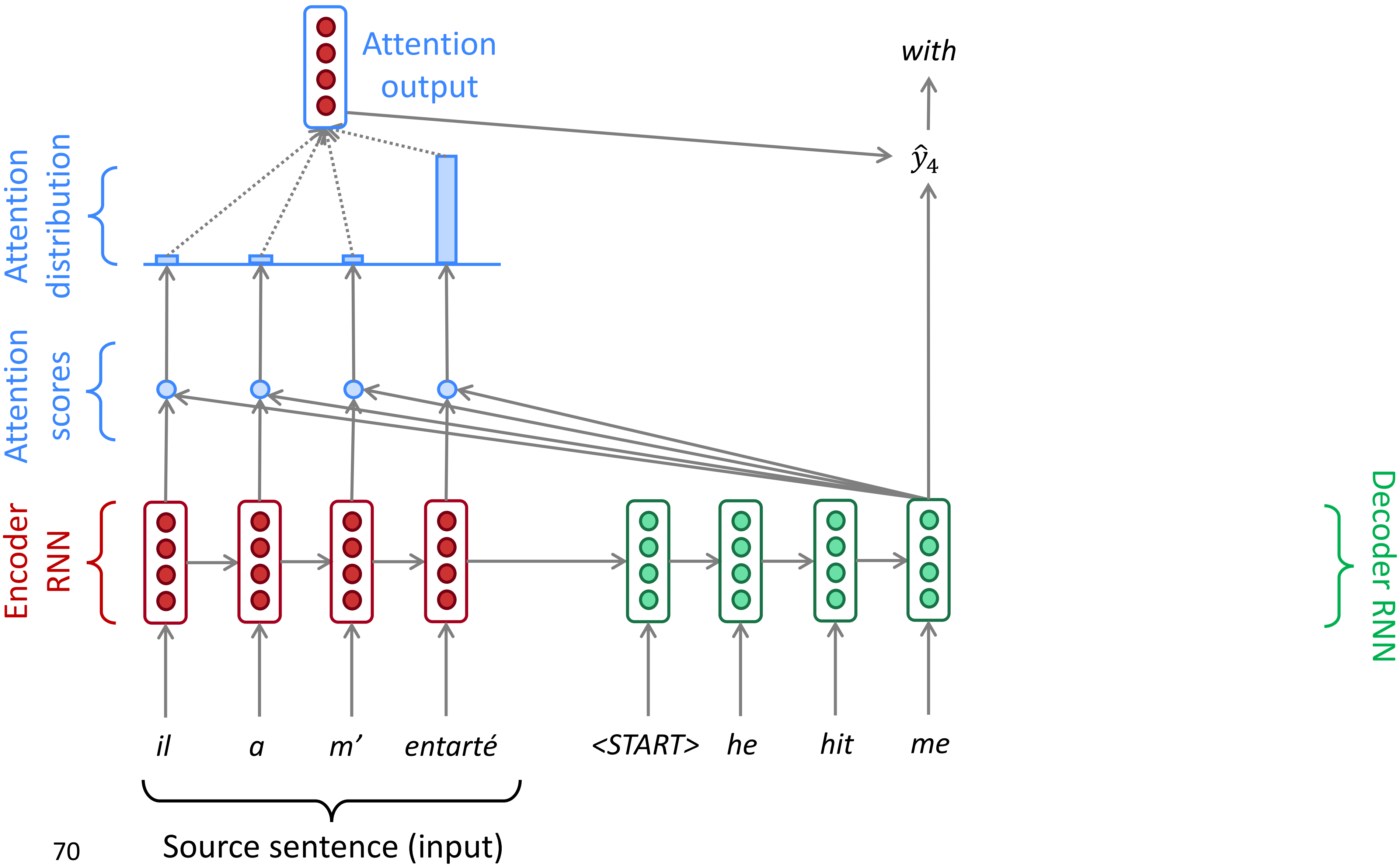
SMT에서는 Hard Alignment를 했다.

Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input). We do this in Assignment 4.

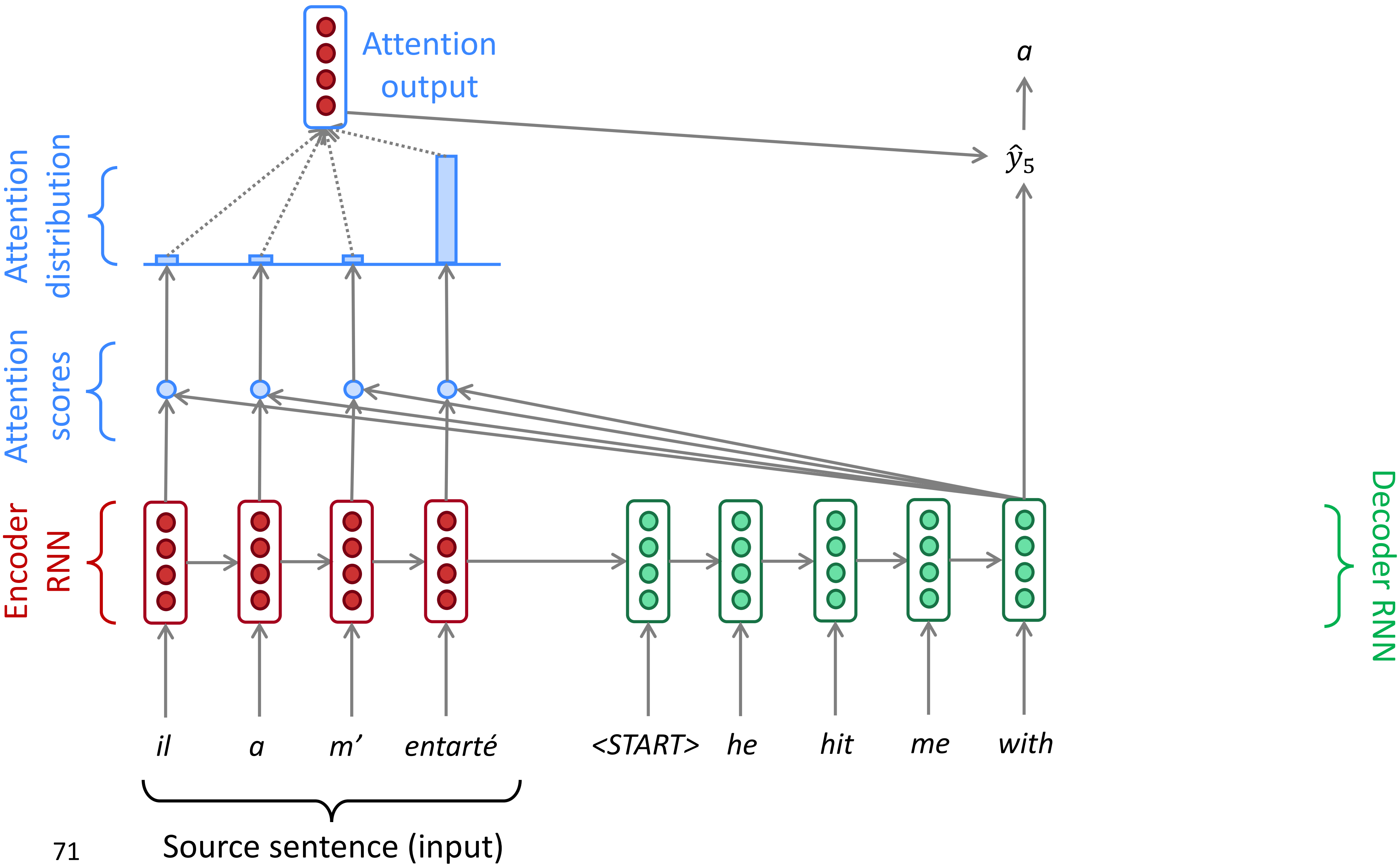
# Sequence-to-sequence with attention



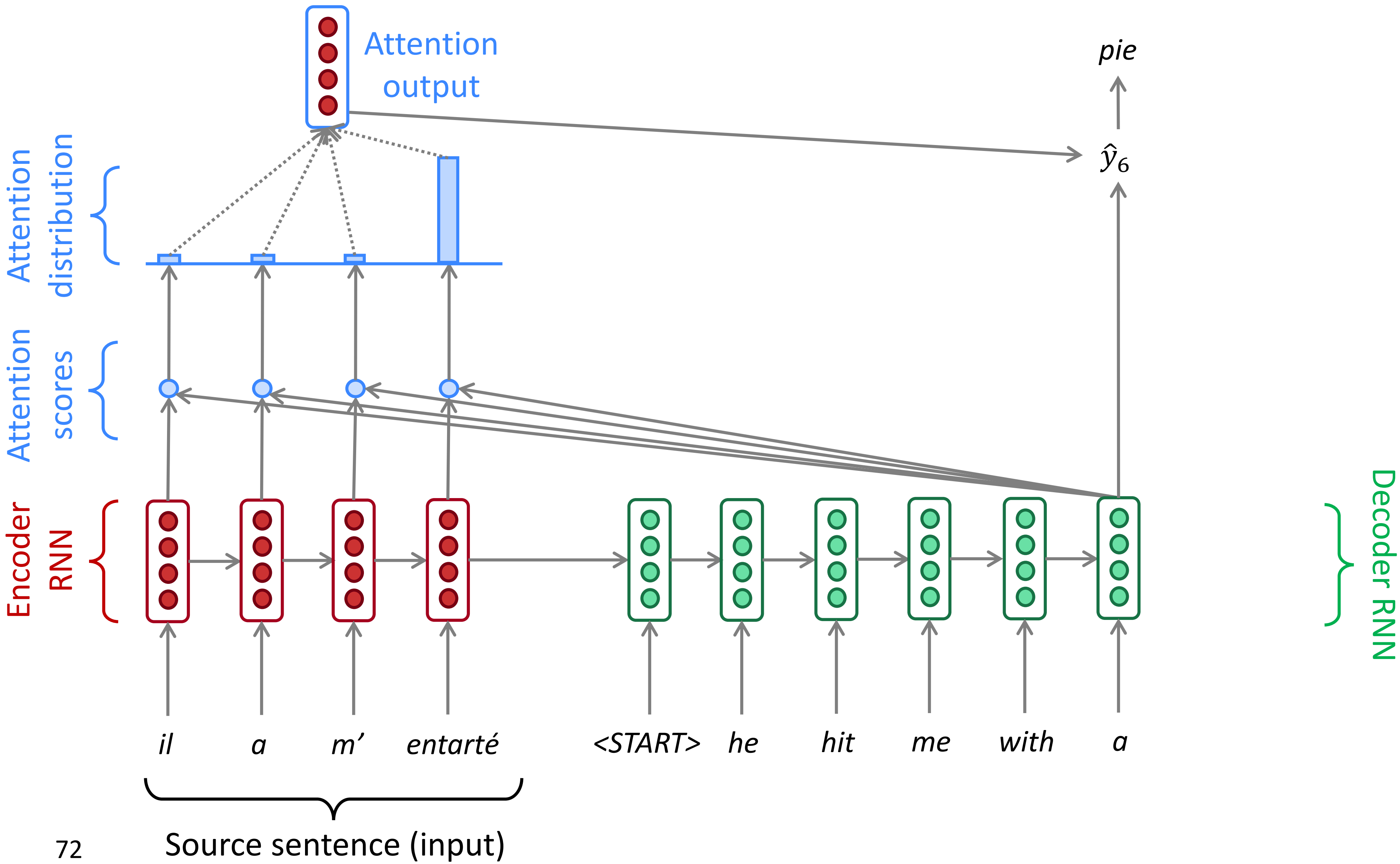
# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



# Attention: in equations

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

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

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

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

- We use $\alpha^t$ to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

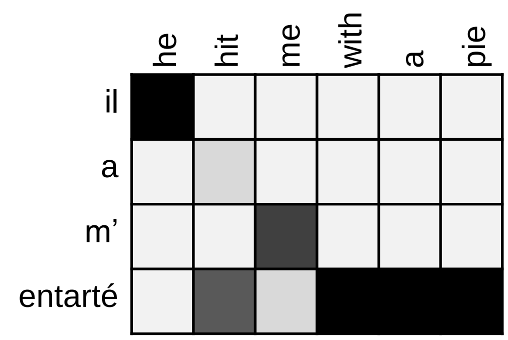
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state $s_t$ and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

# 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




우리는 NMT 시스템에 directly하게 alignment를 학습 해라고 지시한 적이 없다.
(we never had a loss function that tells you how good your alignment was)

비지도 학습 방식으로 이러한 structure를 NMT 시스템이 발견한다.

# Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - 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.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

# Attention is a *general* Deep Learning technique

**More general definition of attention**:

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.

**Intuition**:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on. LSTM의 gate mechanism도 이와 비슷하다.
- 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).

# There are *several* attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution* α:

$$\alpha = \mathrm{softmax}(\boldsymbol{e}) \in \mathbb{R}^N$$

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output* ***a*** (sometimes called the *context vector*)

There are multiple ways to do this

# Attention variants

You'll think about the relative advantages/disadvantages of these in Assignment 4!

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

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

**More information:**

"Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

# Summary of today's lecture

- We learned some history of Machine Translation (MT)

- Since 2014, Neural MT rapidly replaced intricate Statistical MT

- Sequence-to-sequence is the architecture for NMT (uses 2 RNNs)

- Attention is a way to *focus on particular parts* of the input
  - Improves sequence-to-sequence a lot!