# Neural Machine Translation

Dan Klein
UC Berkeley

Slides from Abigail See and John DeNero

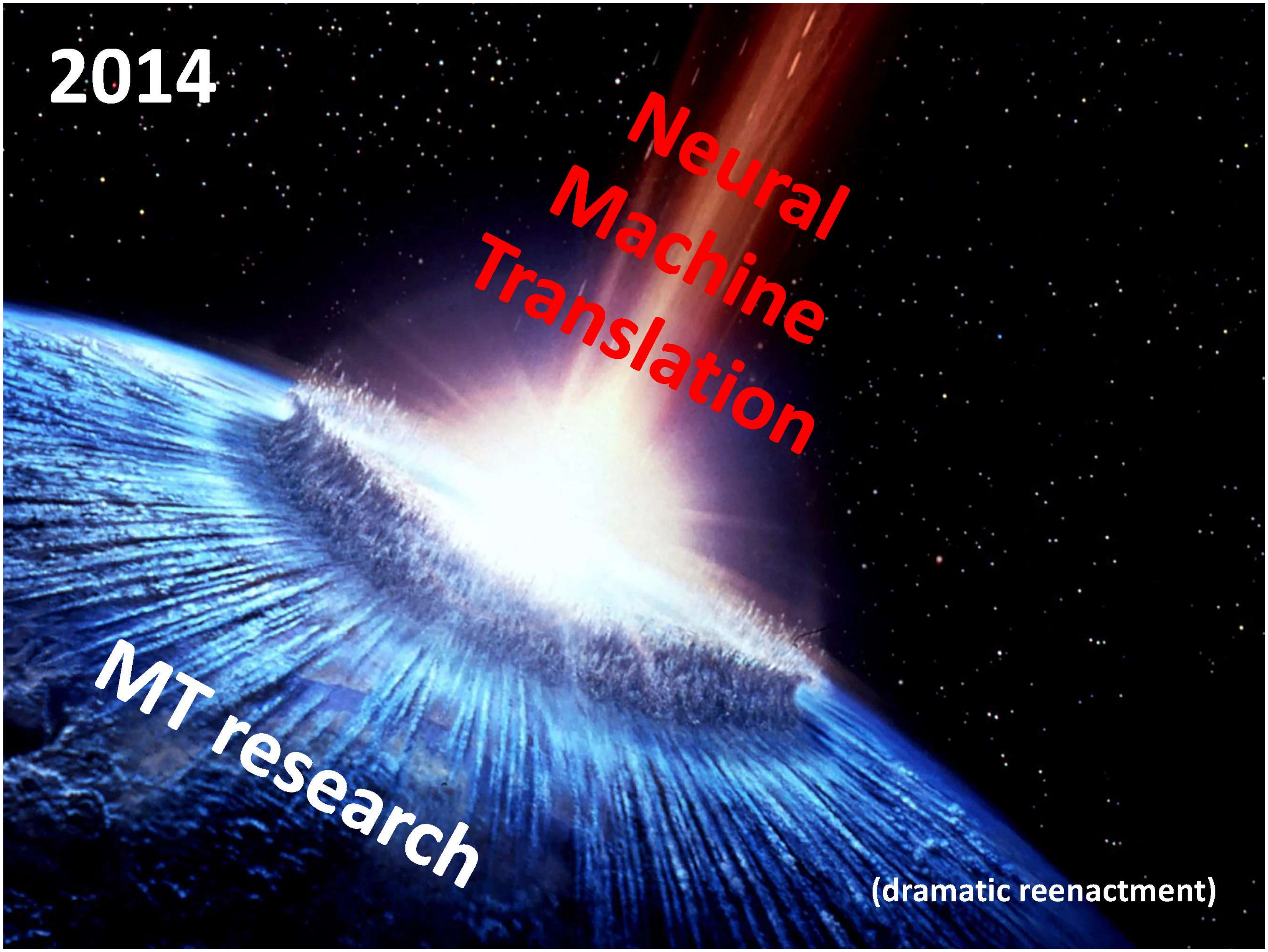
# 1990s-2010s: Statistical Machine Translation

- 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!

# Neural Machine Translation

2014

(dramatic reenactment)


2014
Neural
Machine
Translation
MT research
(dramatic reenactment)

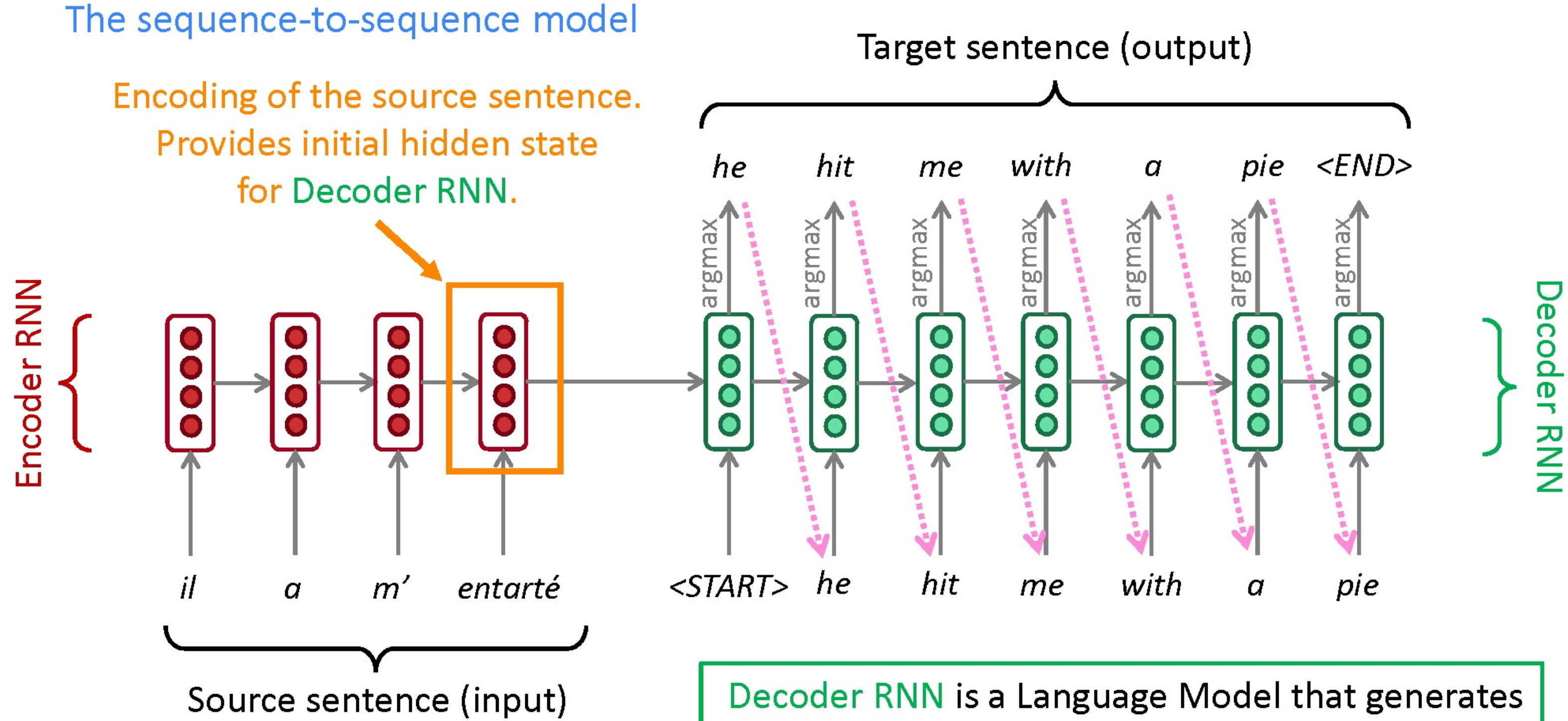
# What is Neural Machine Translation?

- 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




Encoder RNN produces an encoding of the source sentence.

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

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)

# 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*

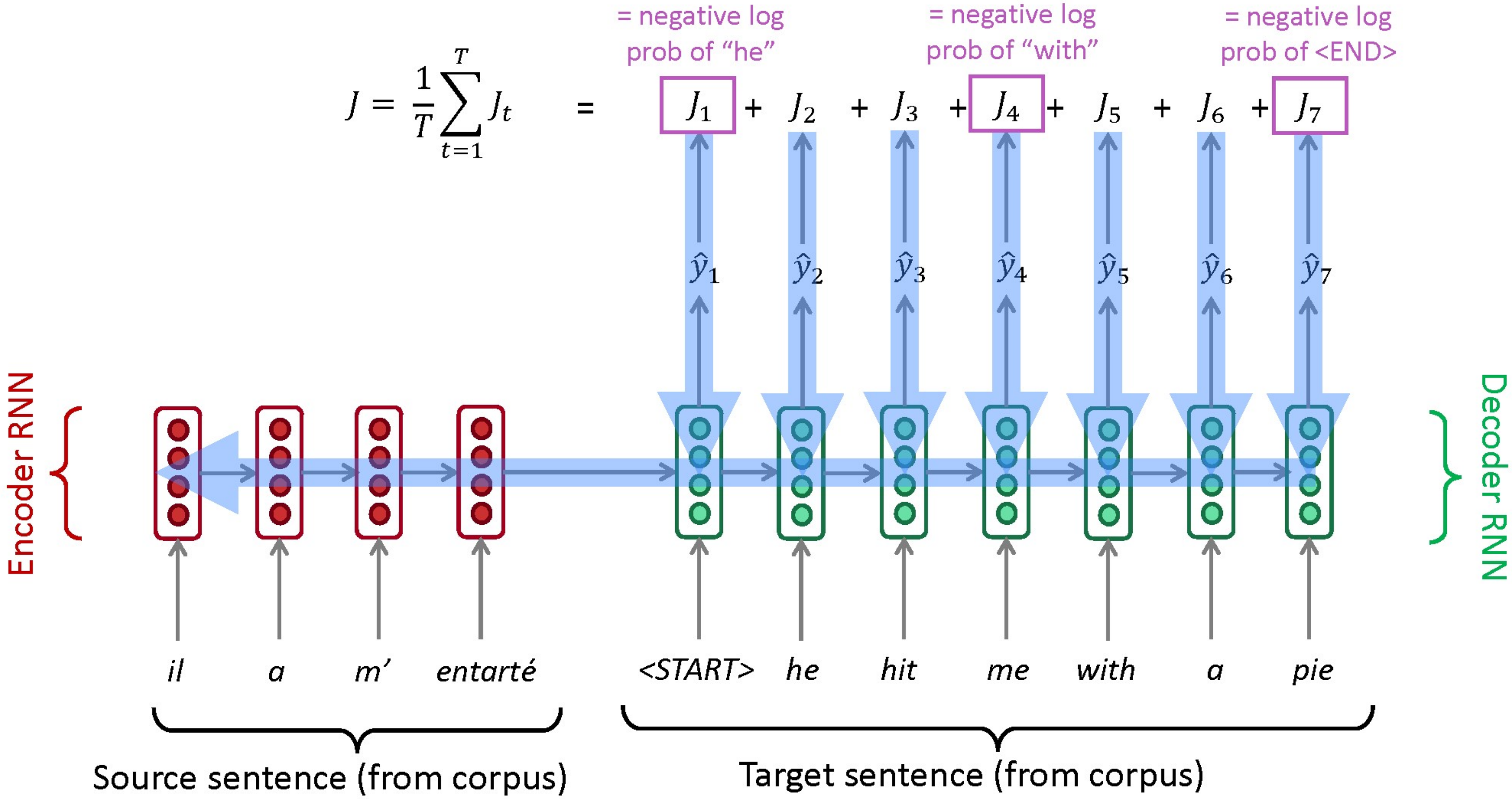
- NMT directly calculates $P(y|x)$ :

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

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

# Training a Neural Machine Translation system

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

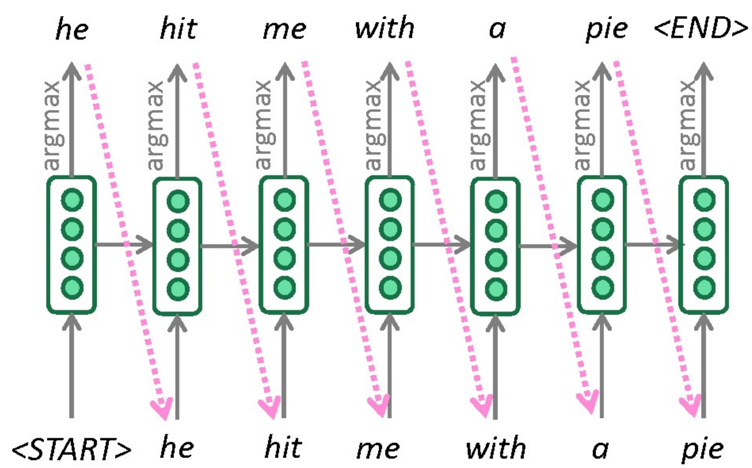



Seq2seq is optimized as a **single system.**
Backpropagation operates *"end-to-end"*.

# NMT Decoding

# 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?**

## 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…)

- How to fix this?

# Exhaustive search decoding

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

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

- 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!

# 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

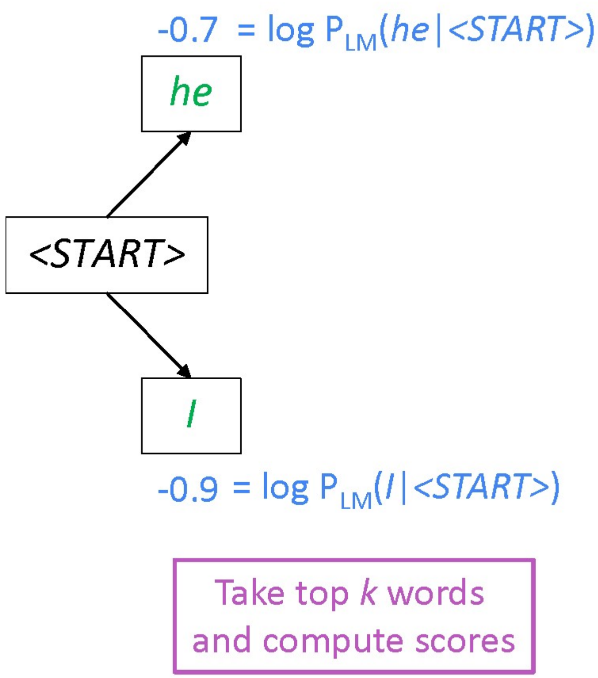
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

*<START>*

Calculate prob dist of next word

# Beam search decoding: example

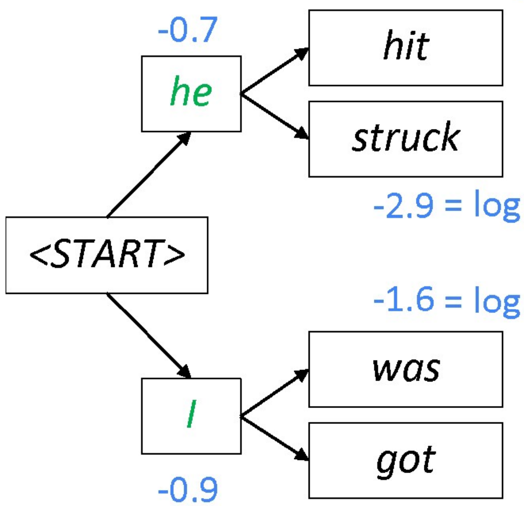
Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$



# Beam search decoding: example

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

-1.7 = log $P_{LM}$(*hit*|*<START> he*) + -0.7




-2.9 = log $P_{LM}$(*struck*|*<START> he*) + -0.7

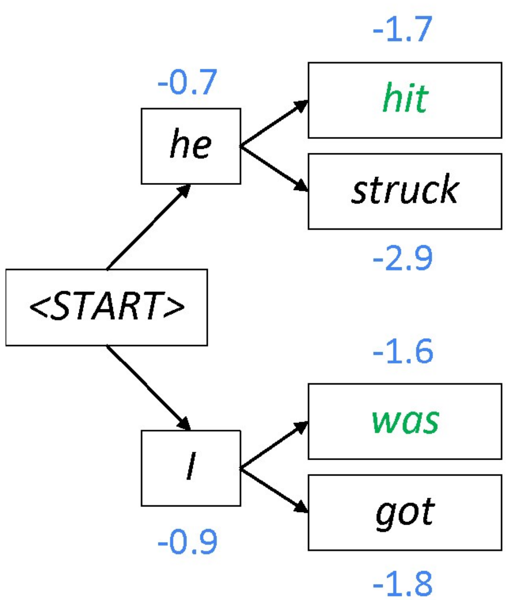
-1.6 = log $P_{LM}$(*was*|*<START> I*) + -0.9

-1.8 = log $P_{LM}$(*got*|*<START> I*) + -0.9

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

# Beam search decoding: example

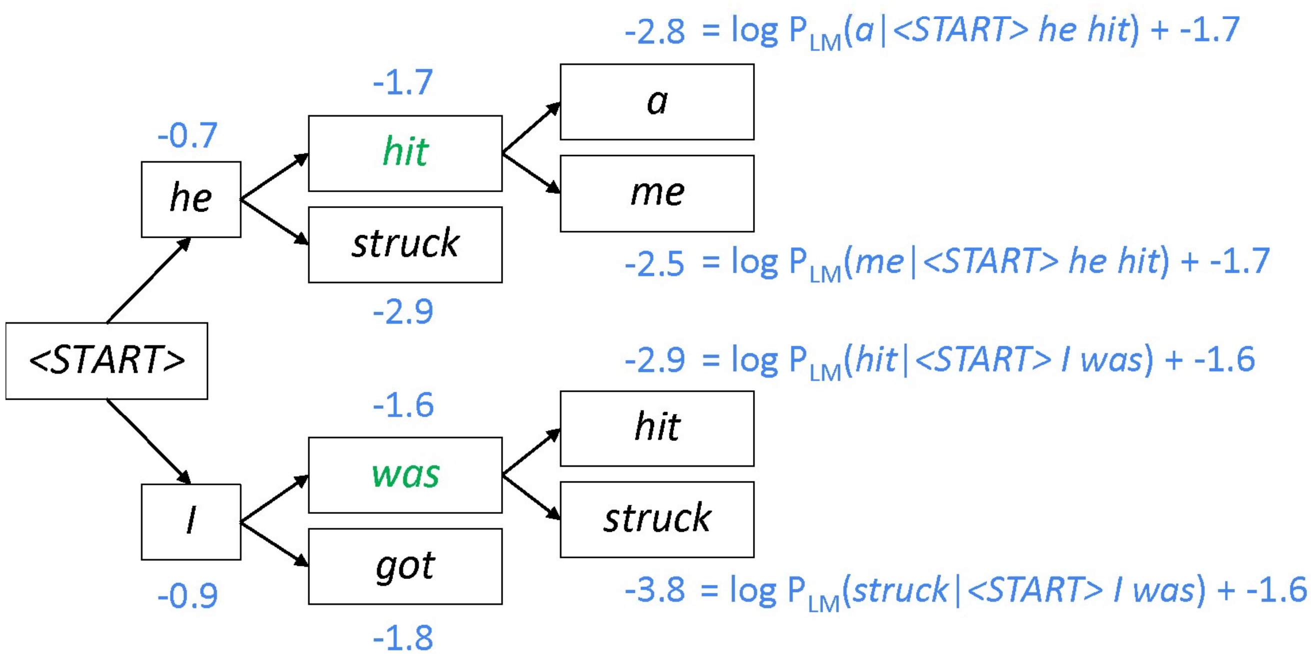
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

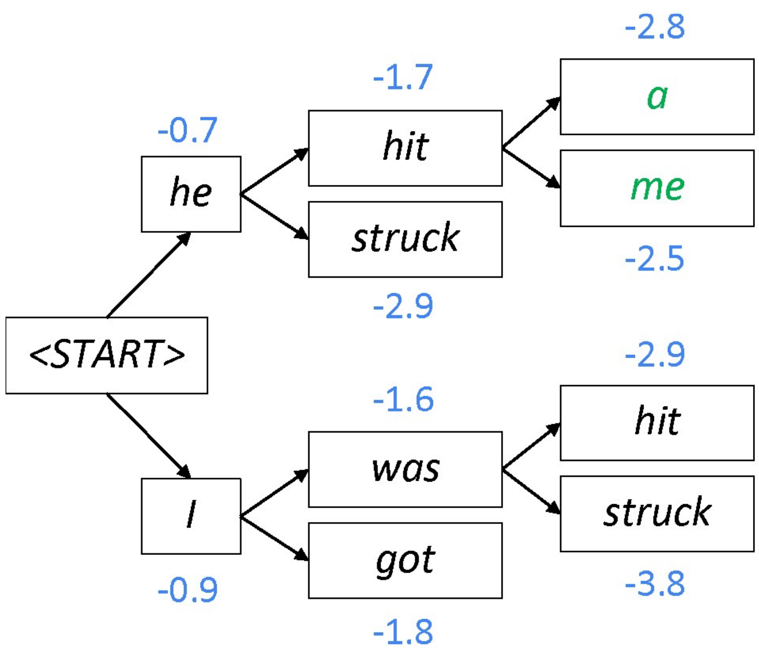
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

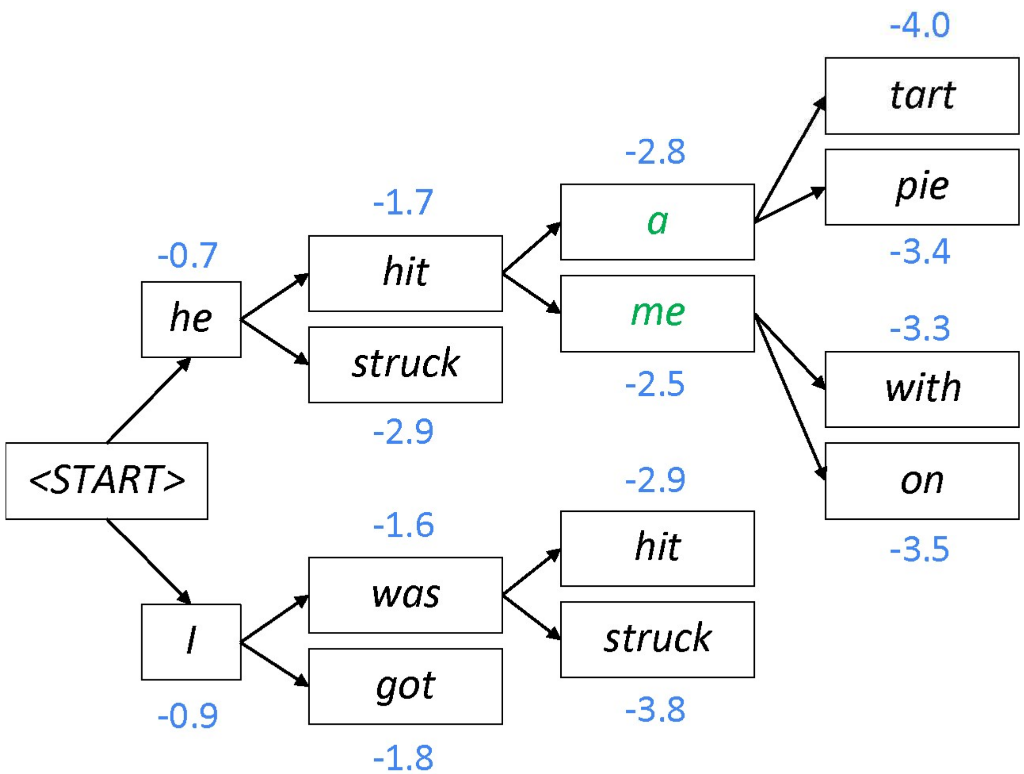
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

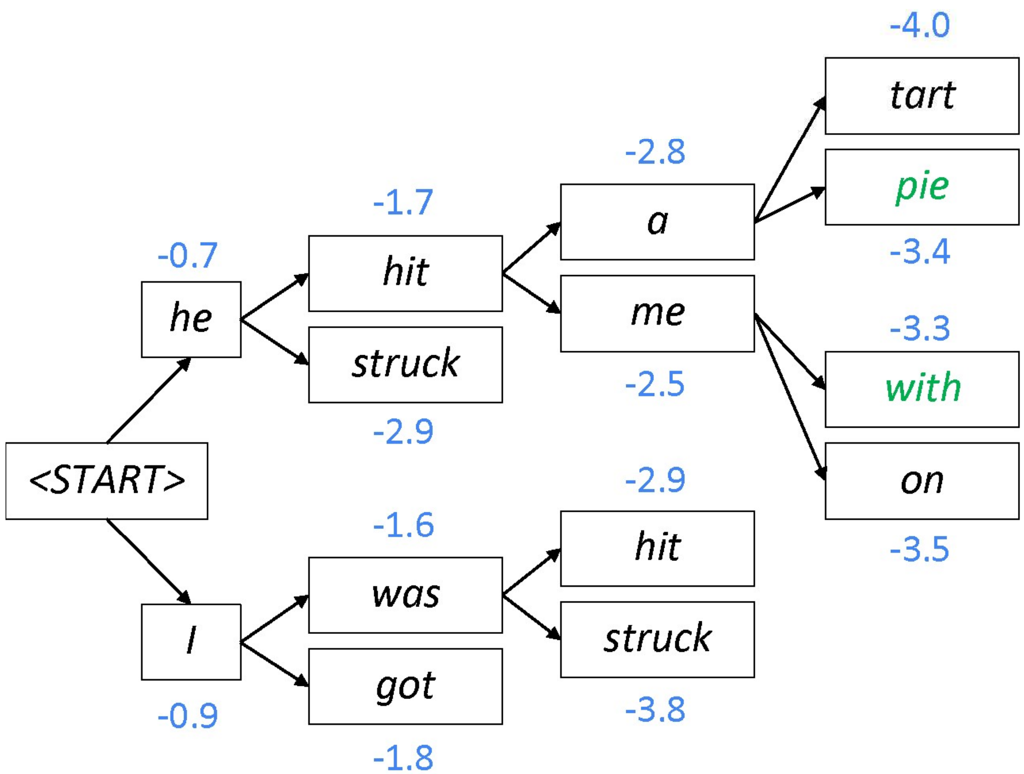
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

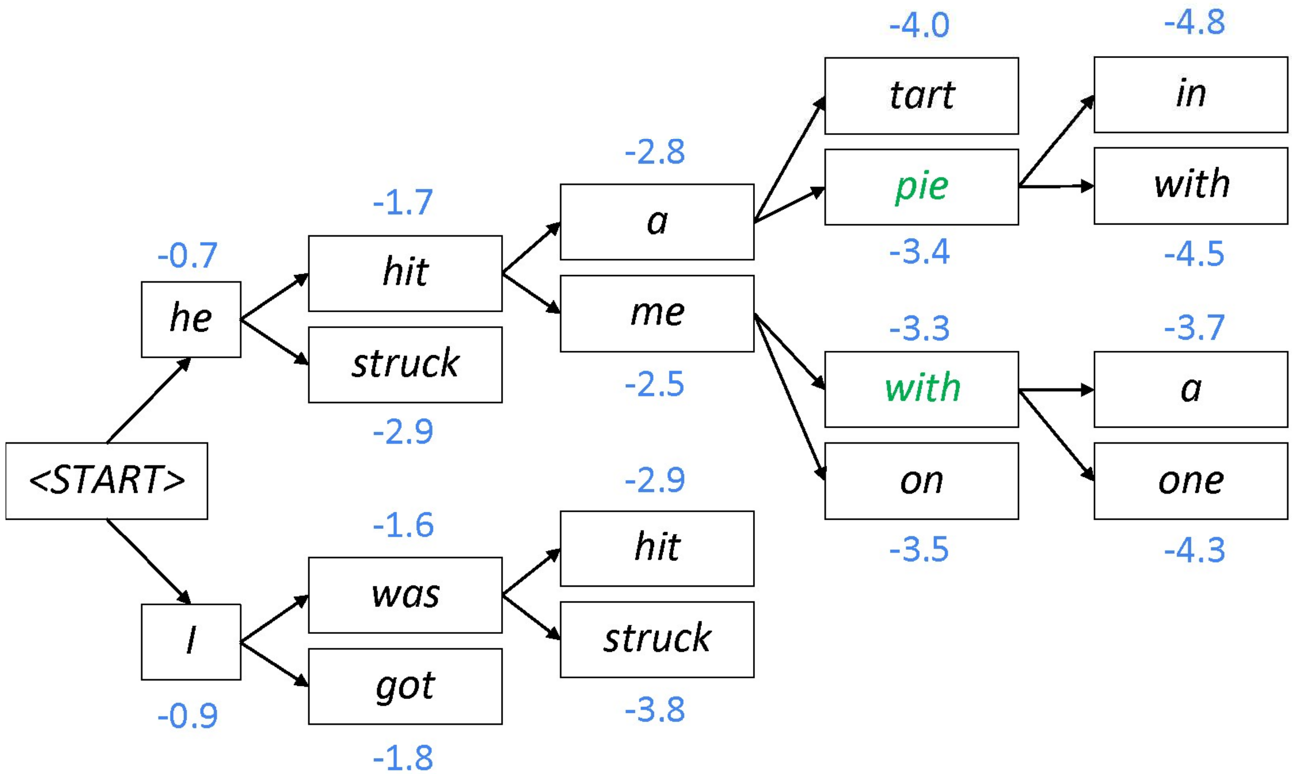
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

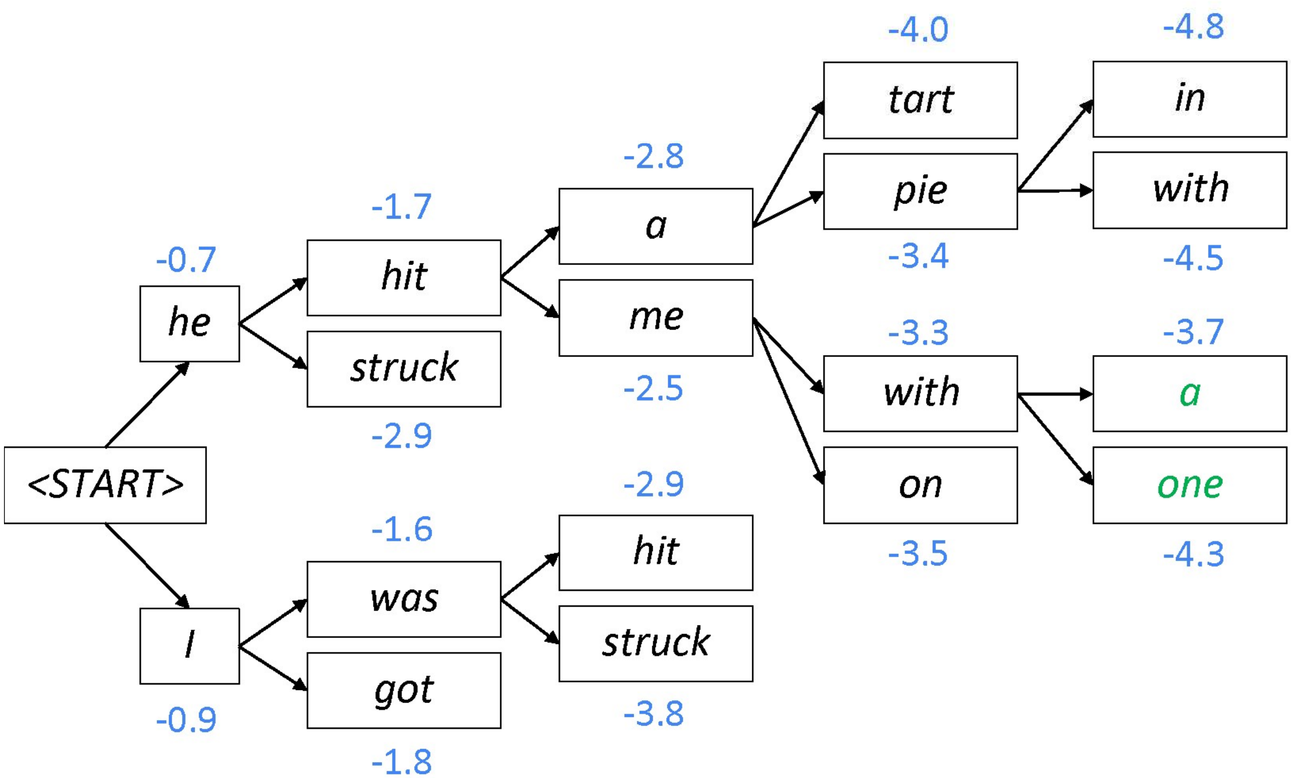
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

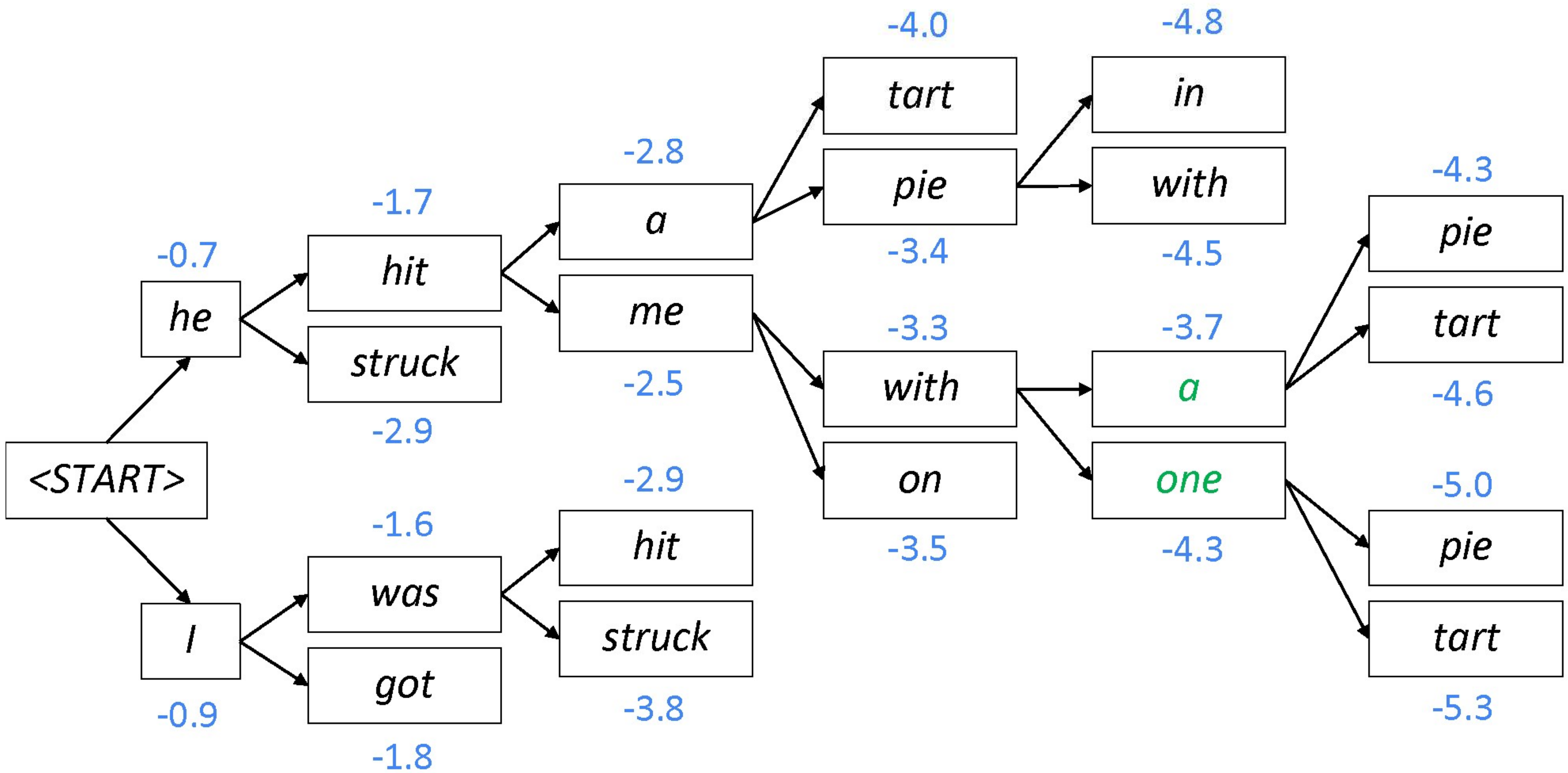
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

# Beam search decoding: example

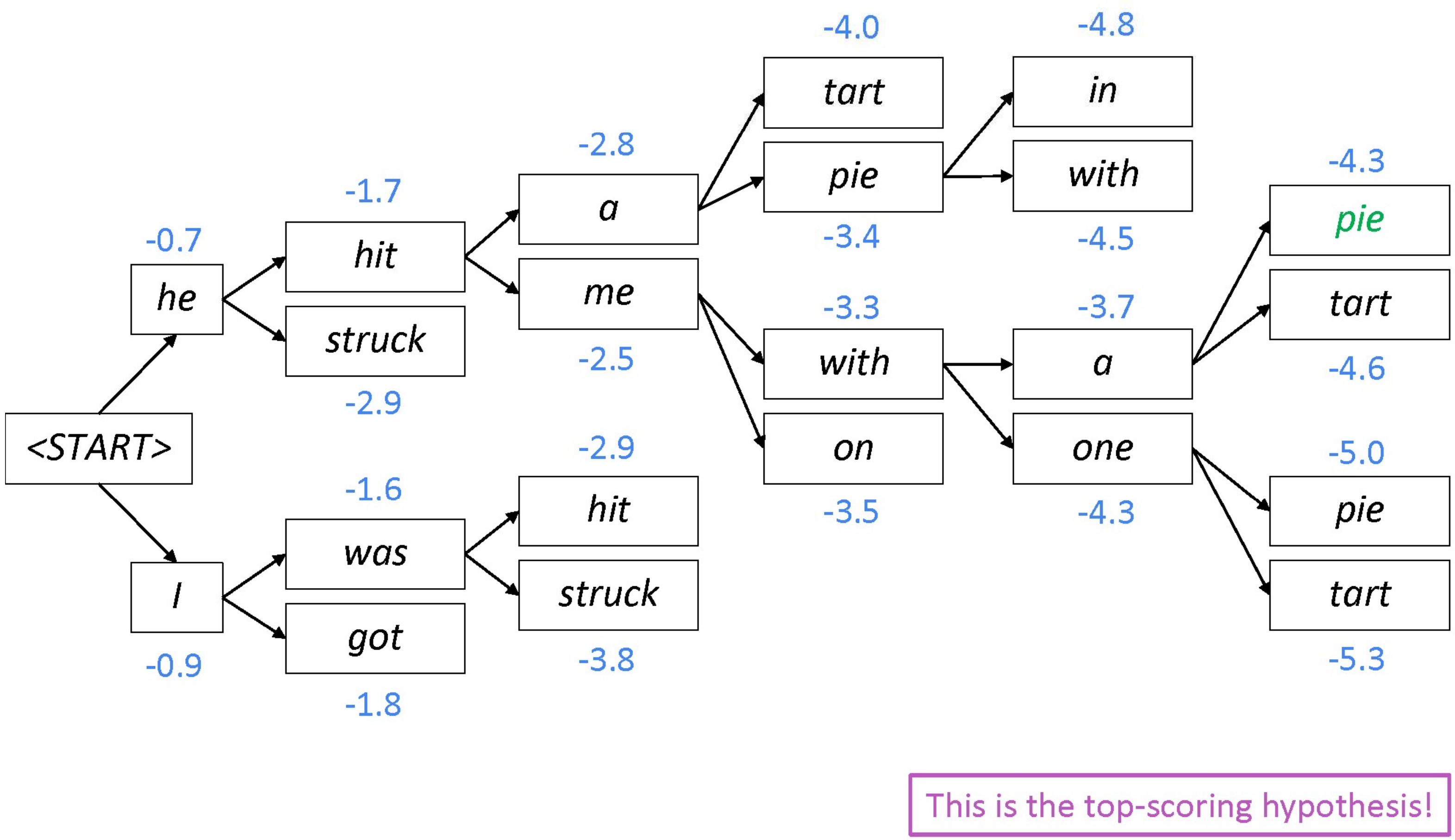
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$




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

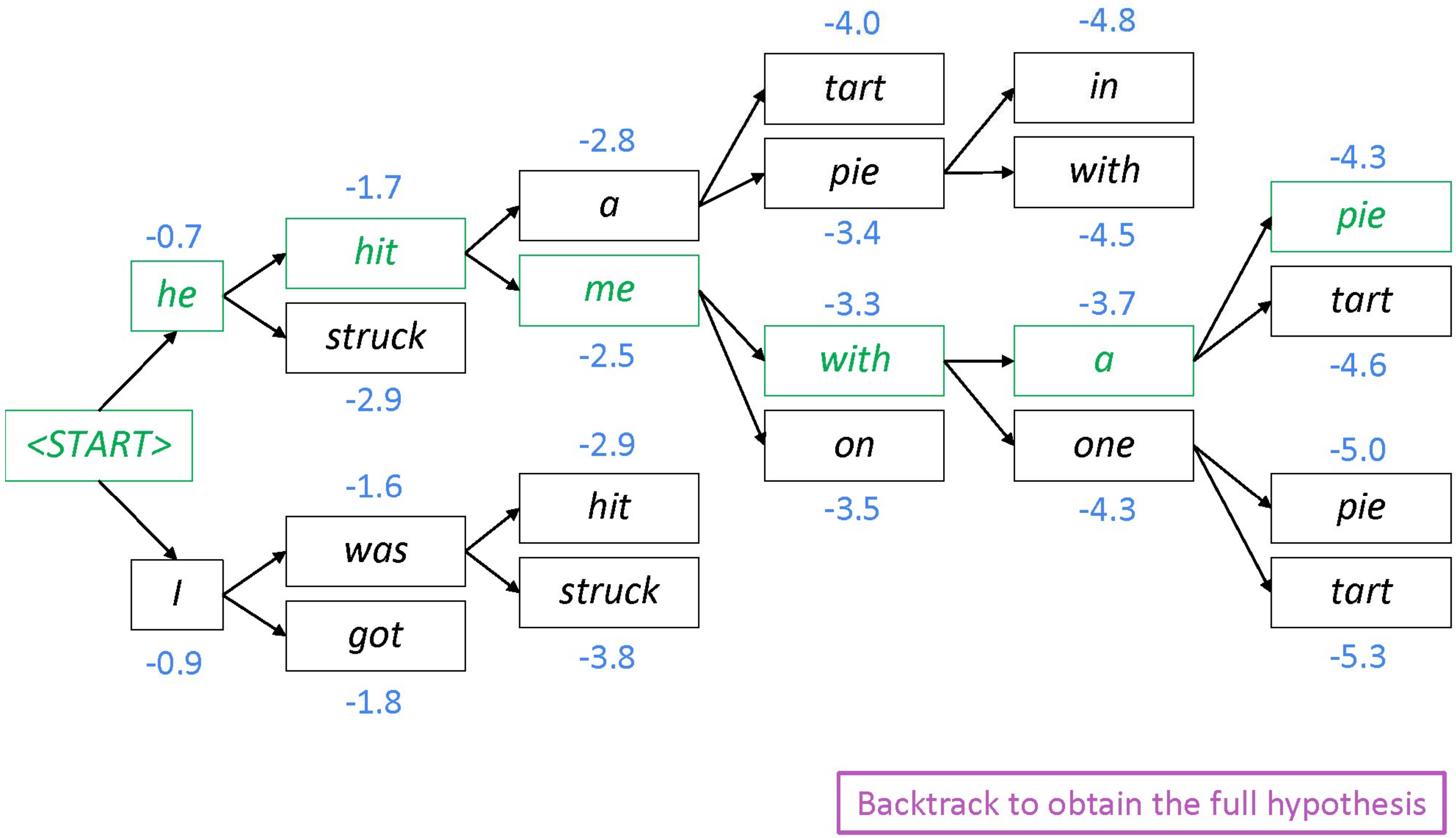
# Beam search decoding: example

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



# Beam search decoding: example

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



# 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

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

- <u>Problem with this:</u> longer hypotheses have lower scores

- <u>Fix:</u> Normalize by length. Use this to select top one instead:

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

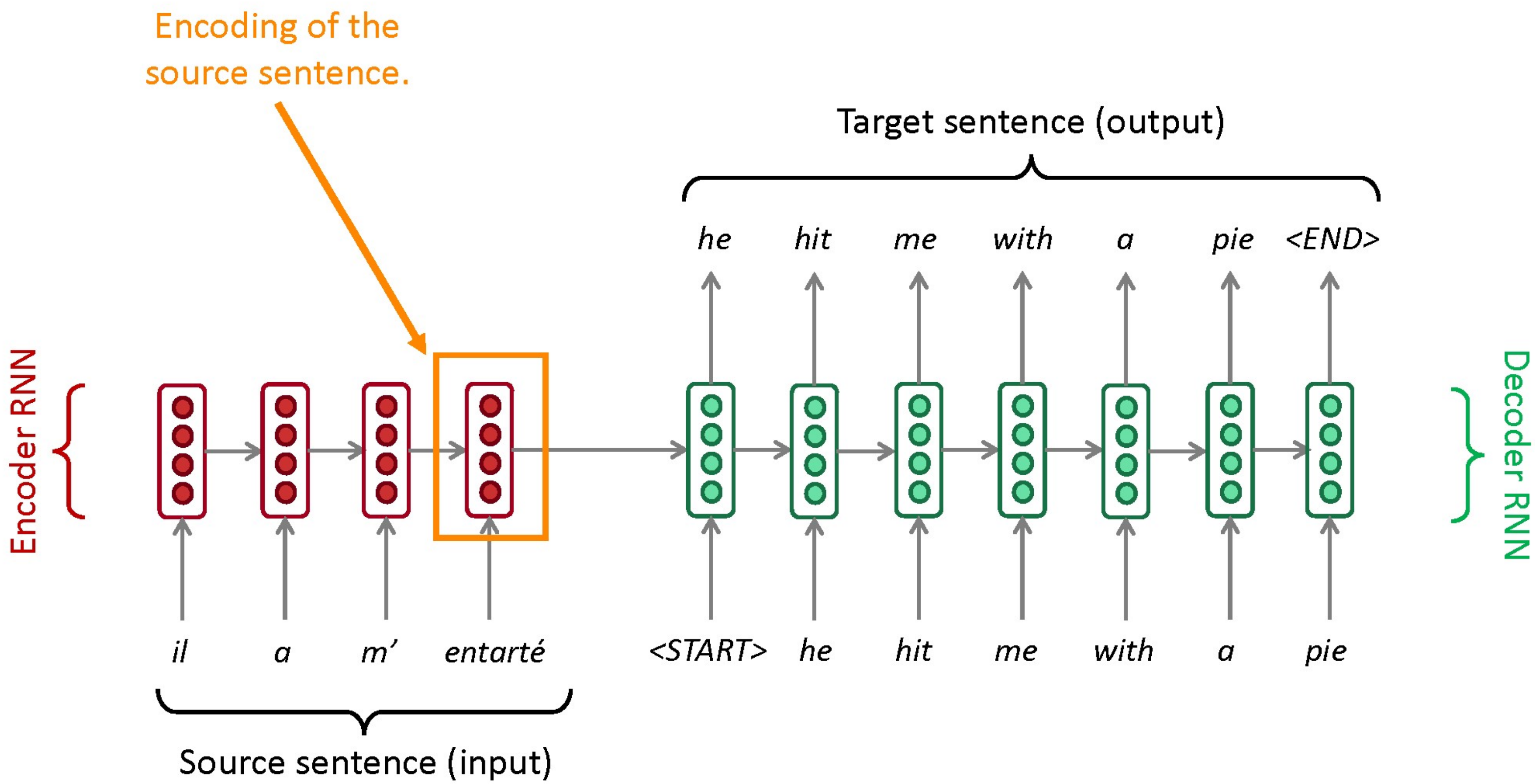
# Neural Machine Translation

Dan Klein
UC Berkeley

Slides from Abigail See and John DeNero

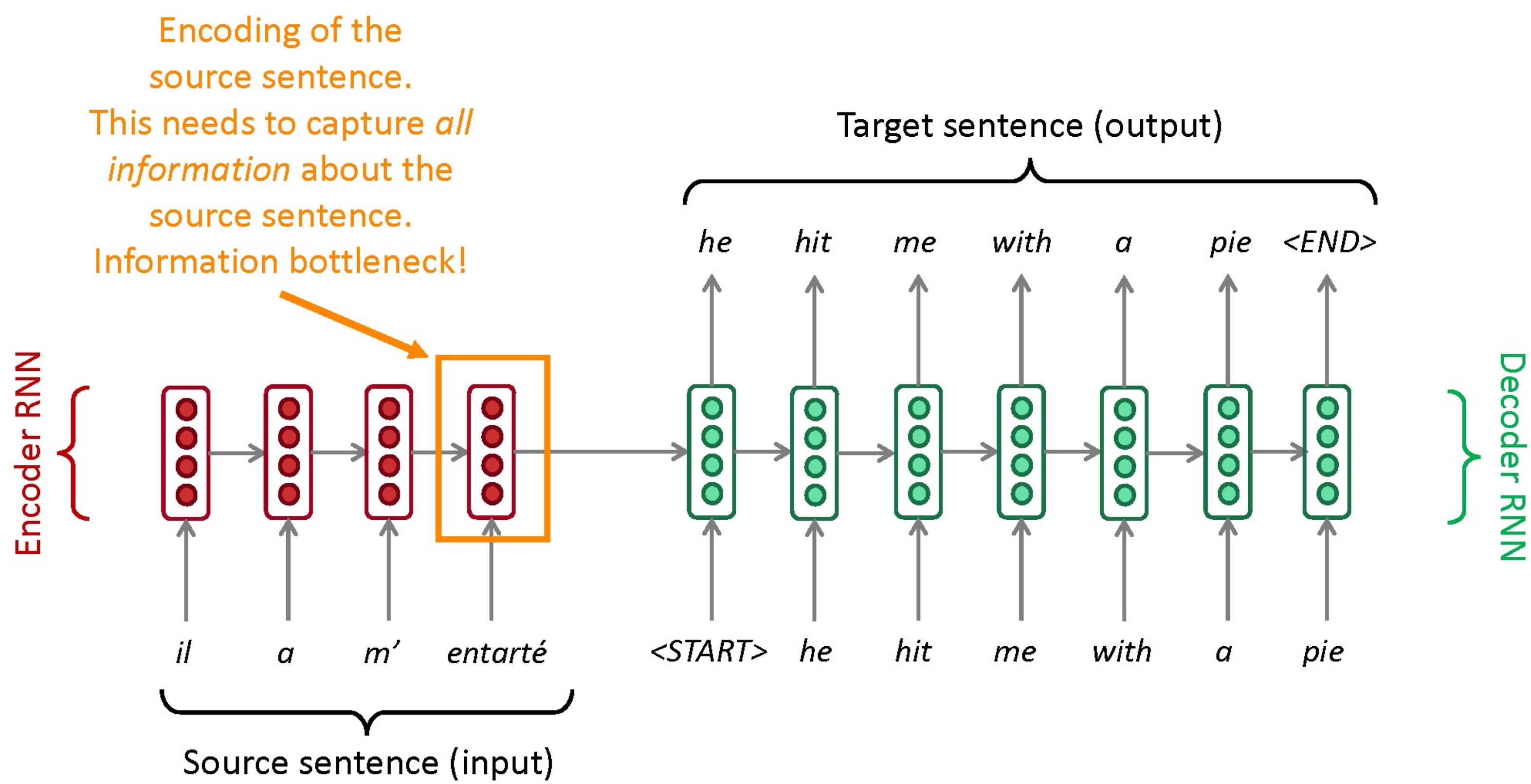
# Attention

# Sequence-to-sequence: the bottleneck problem




**Problems with this architecture?**

# Sequence-to-sequence: the bottleneck problem



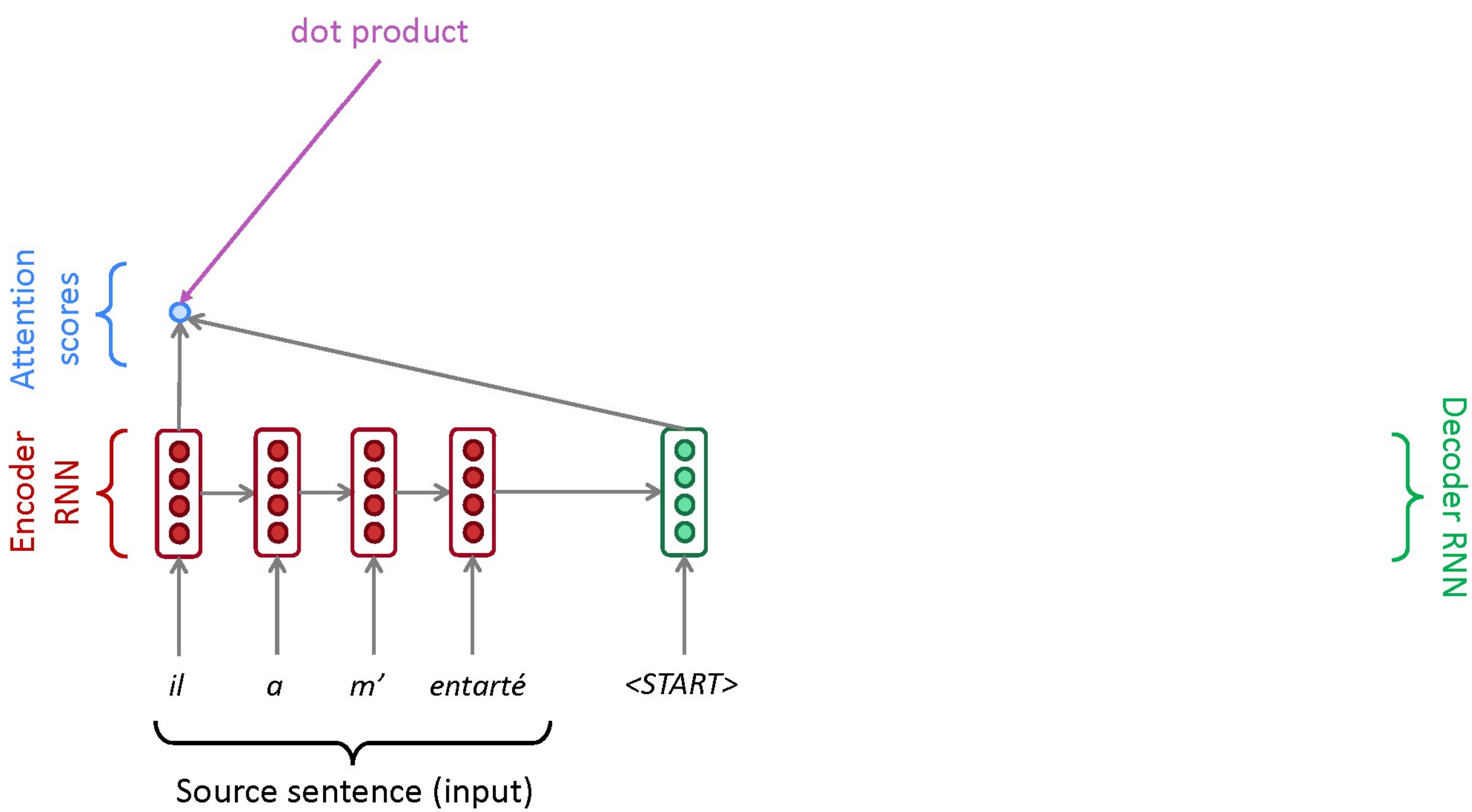
## Attention

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

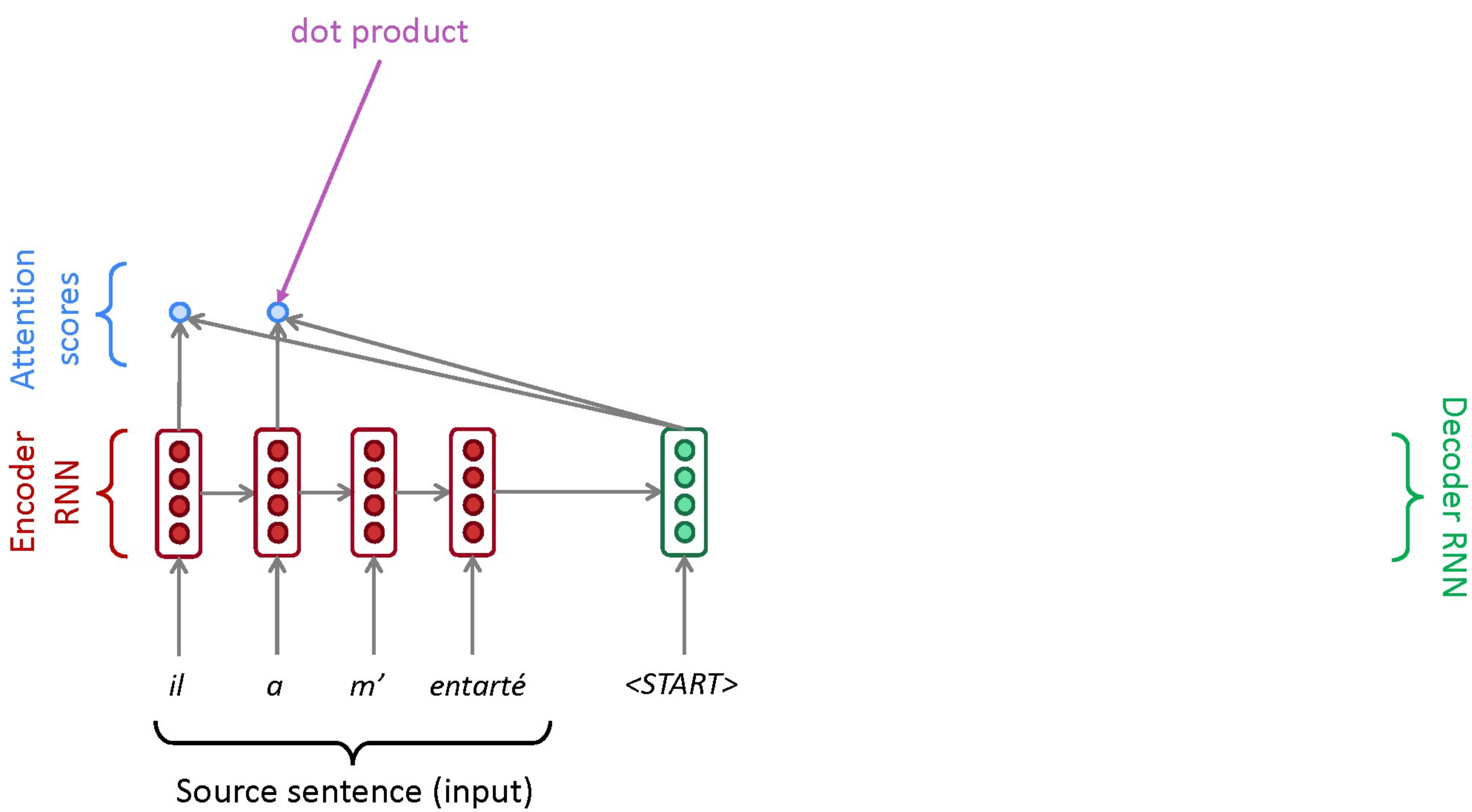
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

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

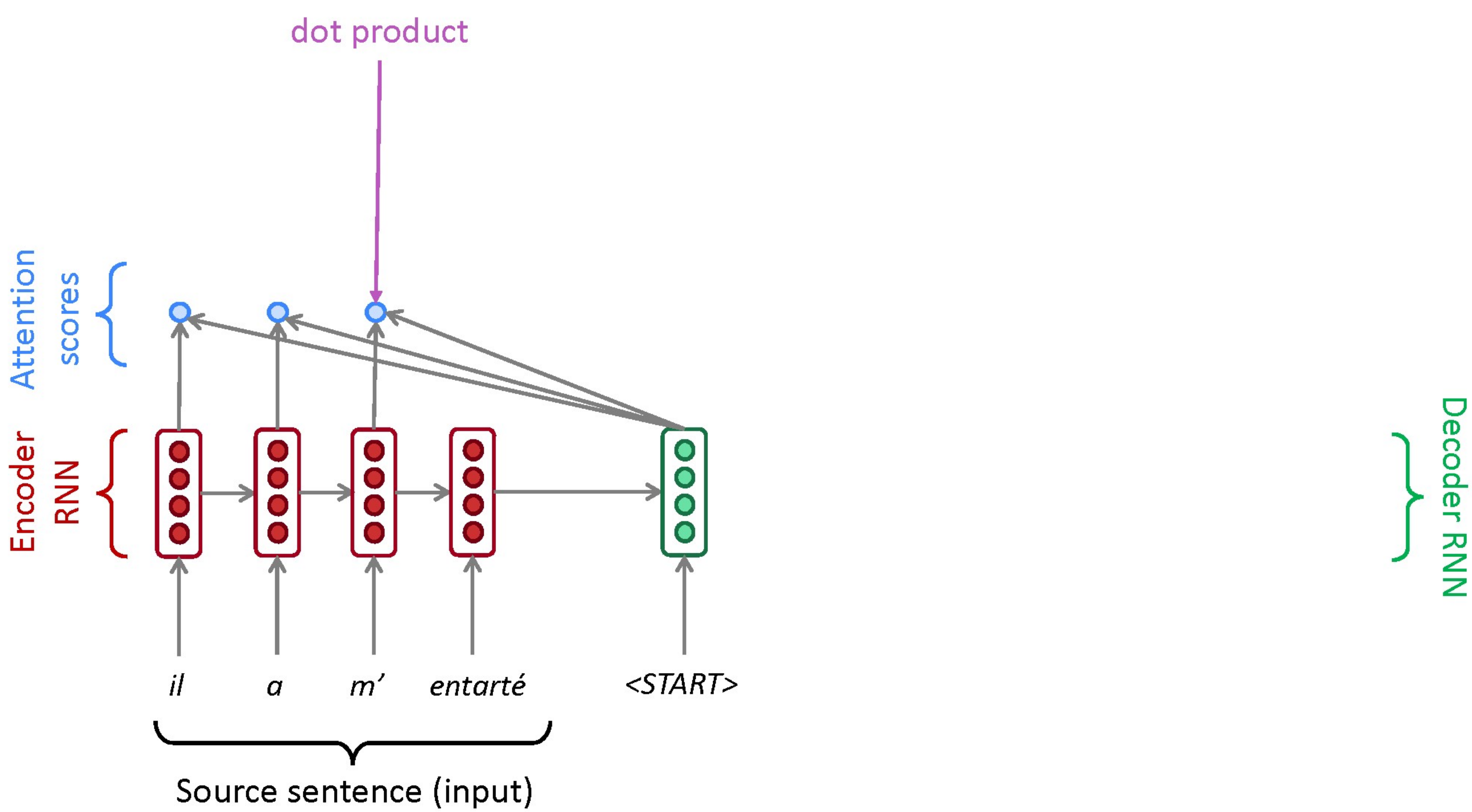
# Sequence-to-sequence with attention



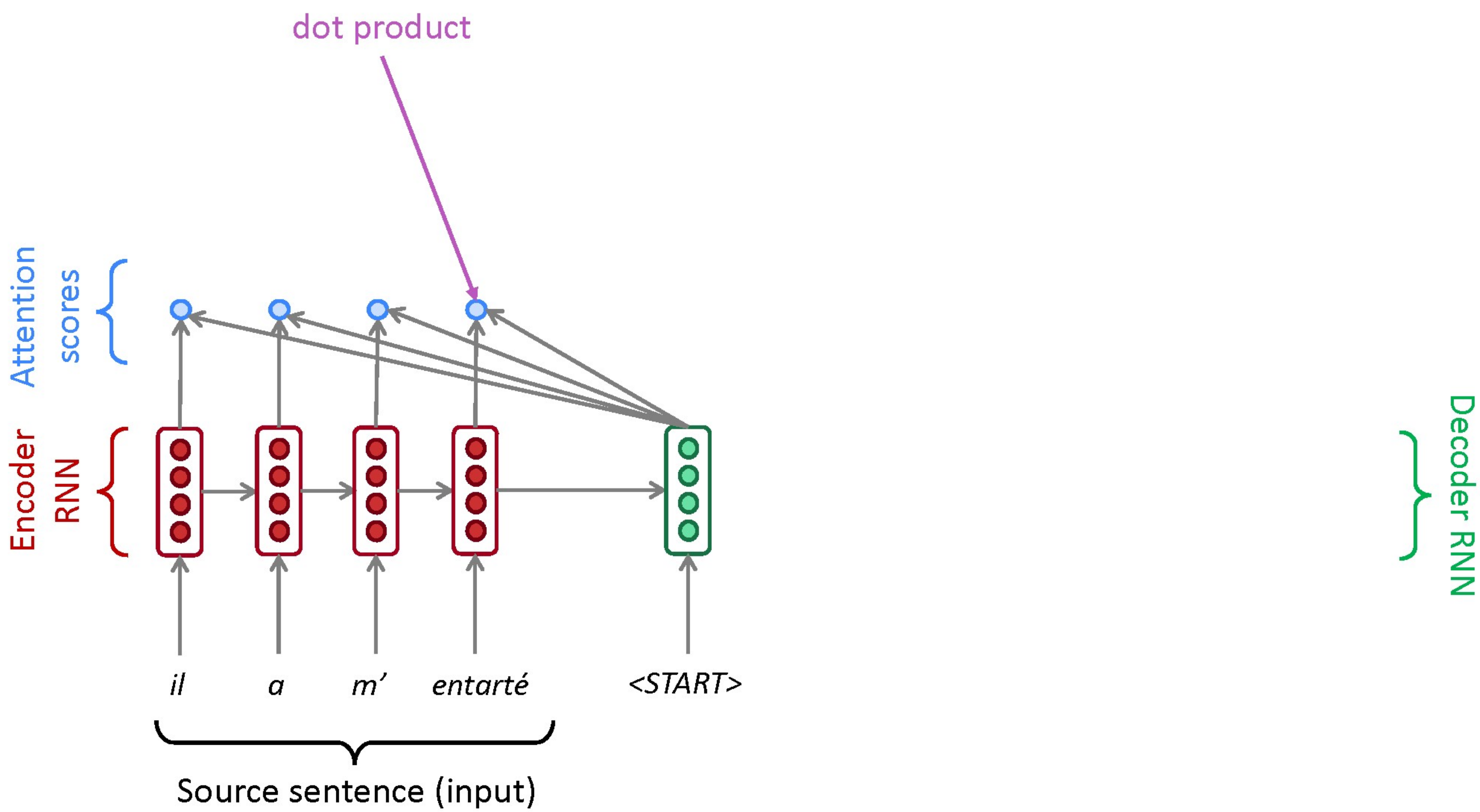
# Sequence-to-sequence with attention



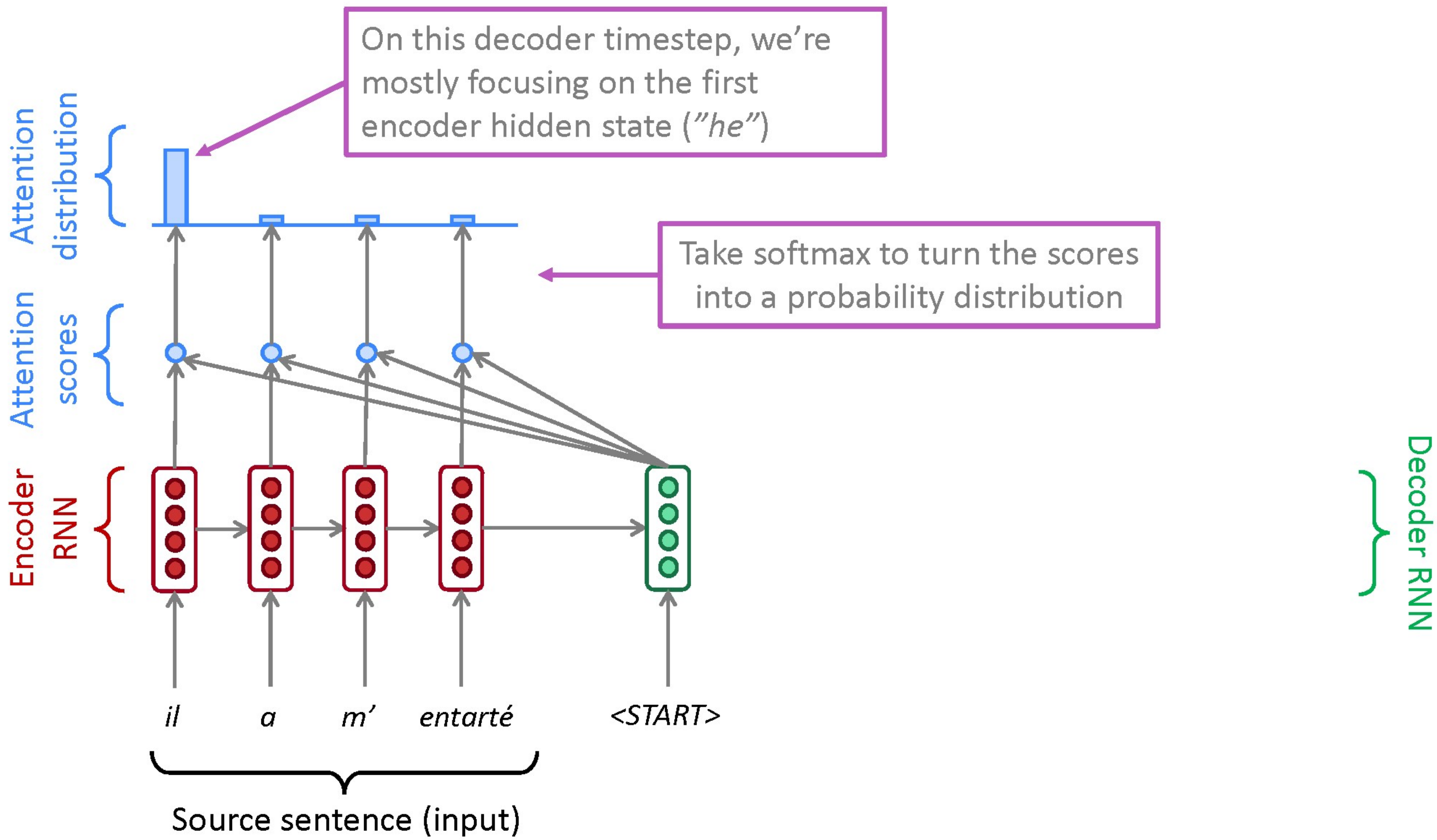
# Sequence-to-sequence with attention



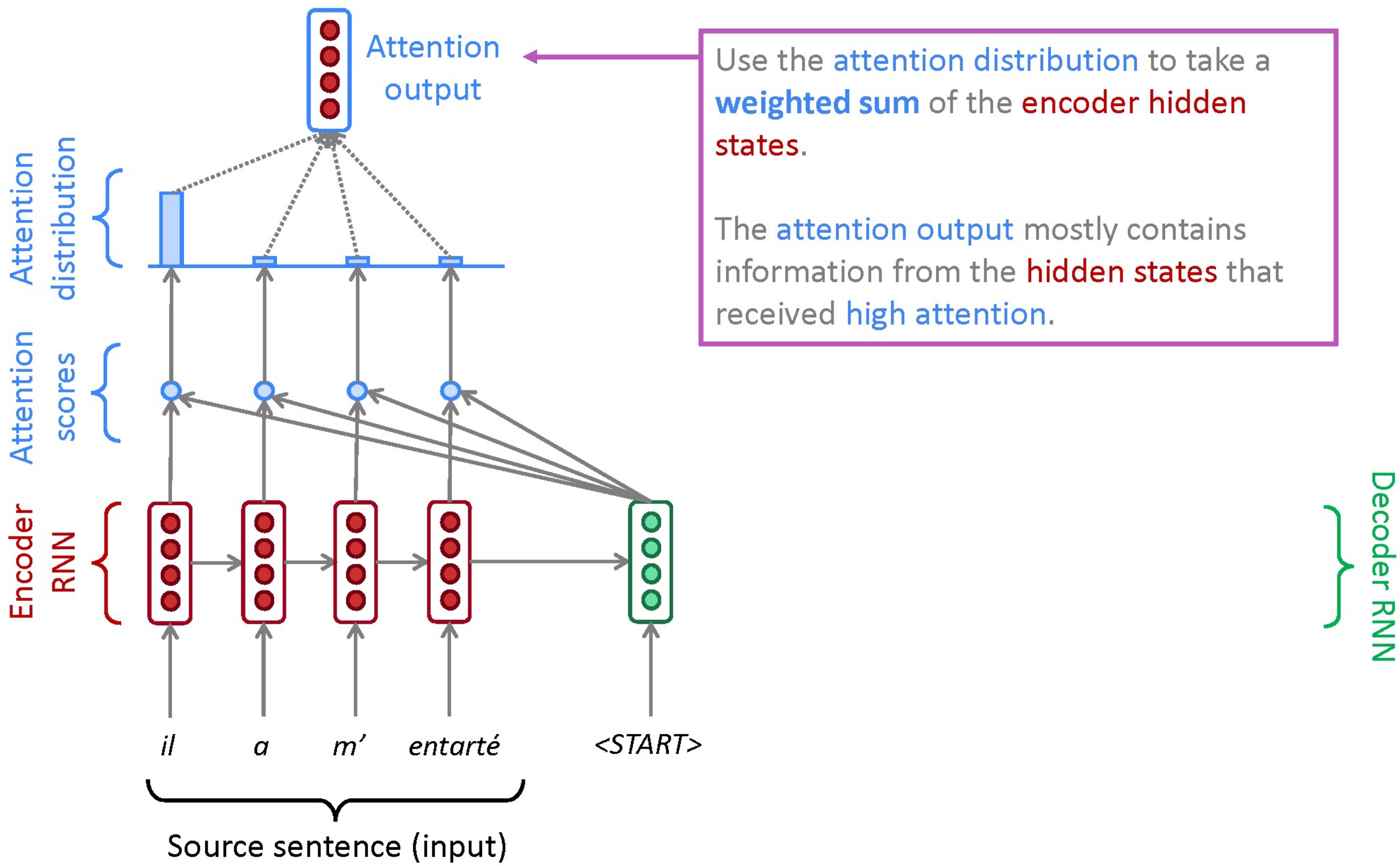
# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



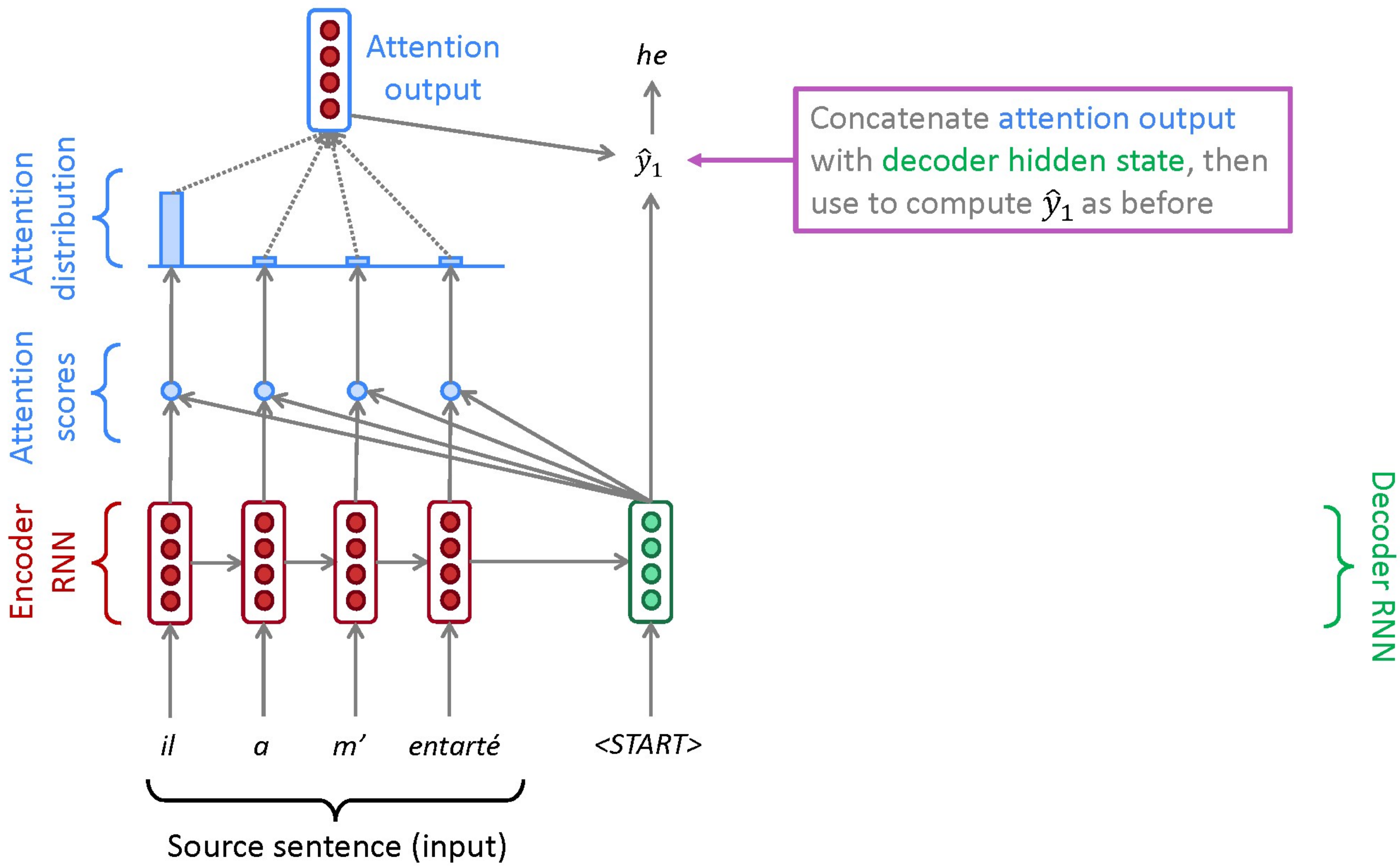
# Sequence-to-sequence with attention



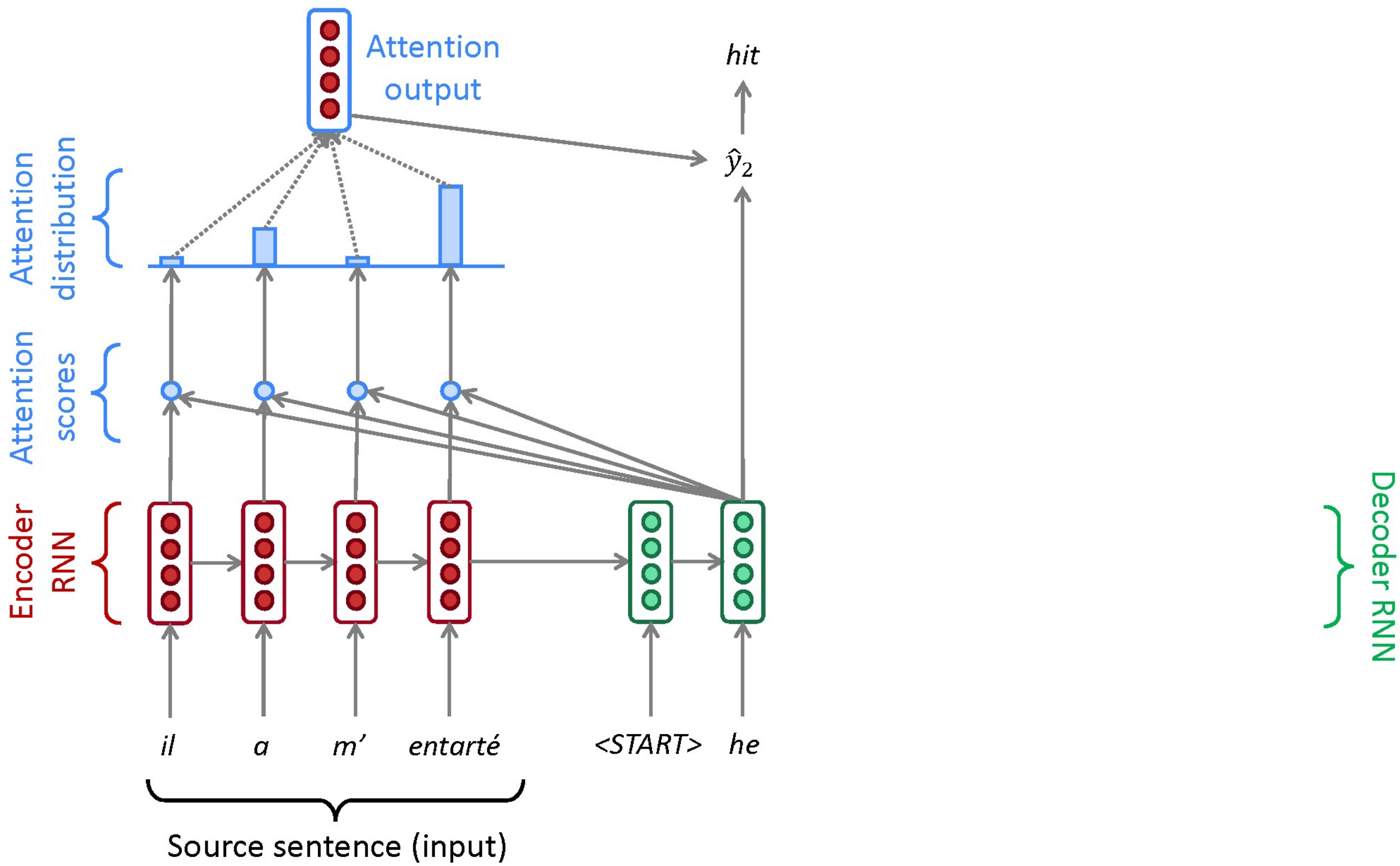

Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

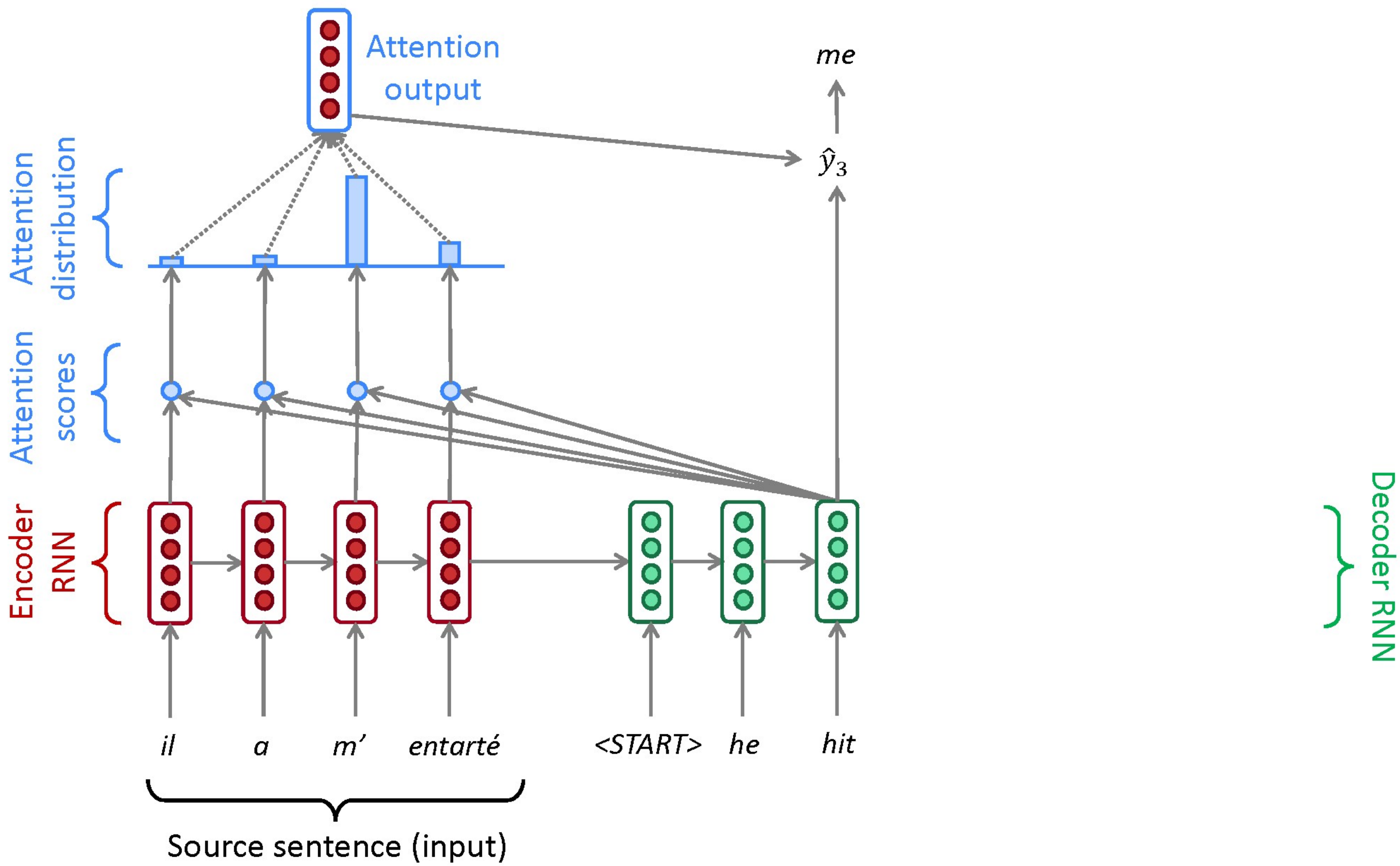
# Sequence-to-sequence with attention



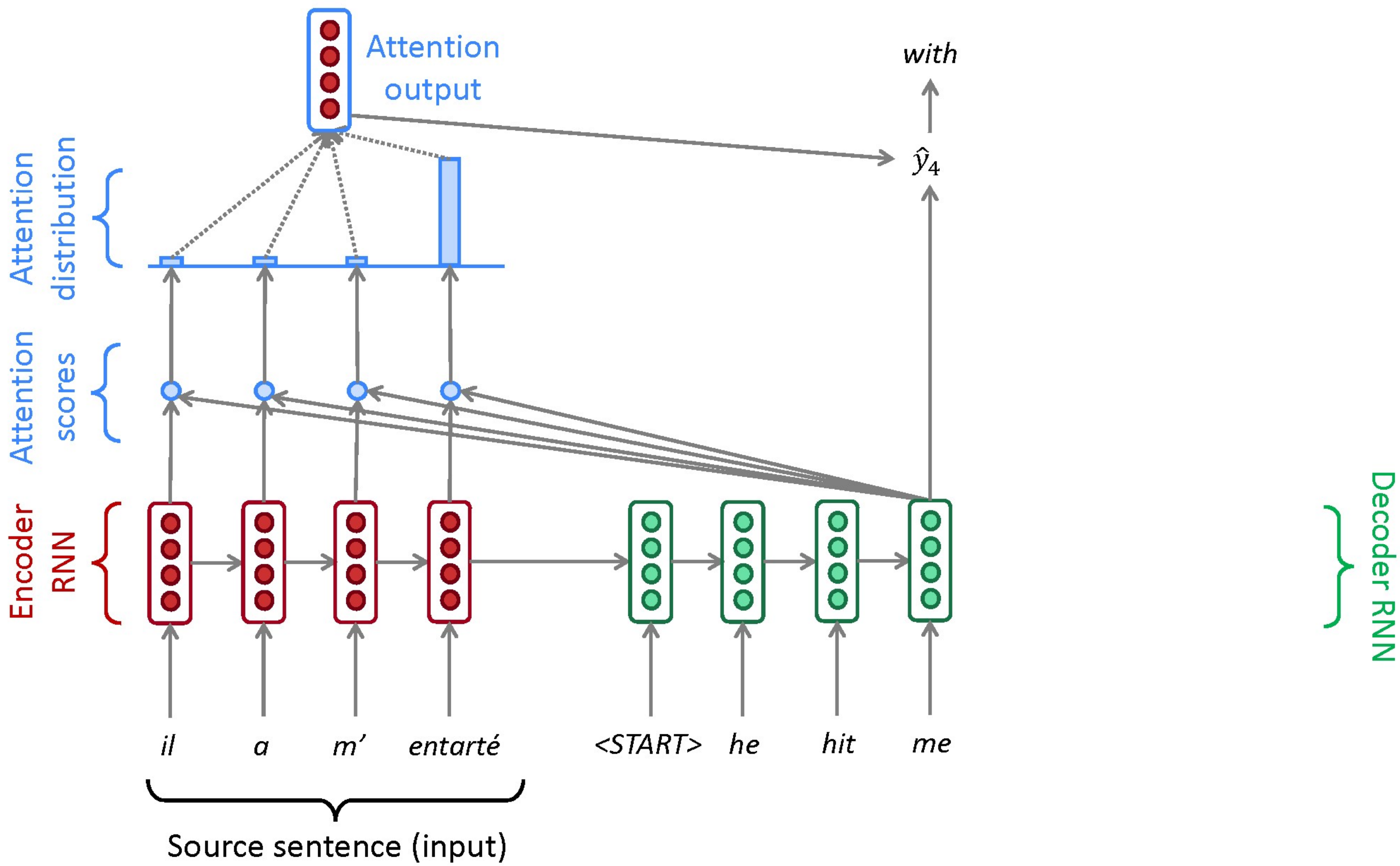
# Sequence-to-sequence with attention



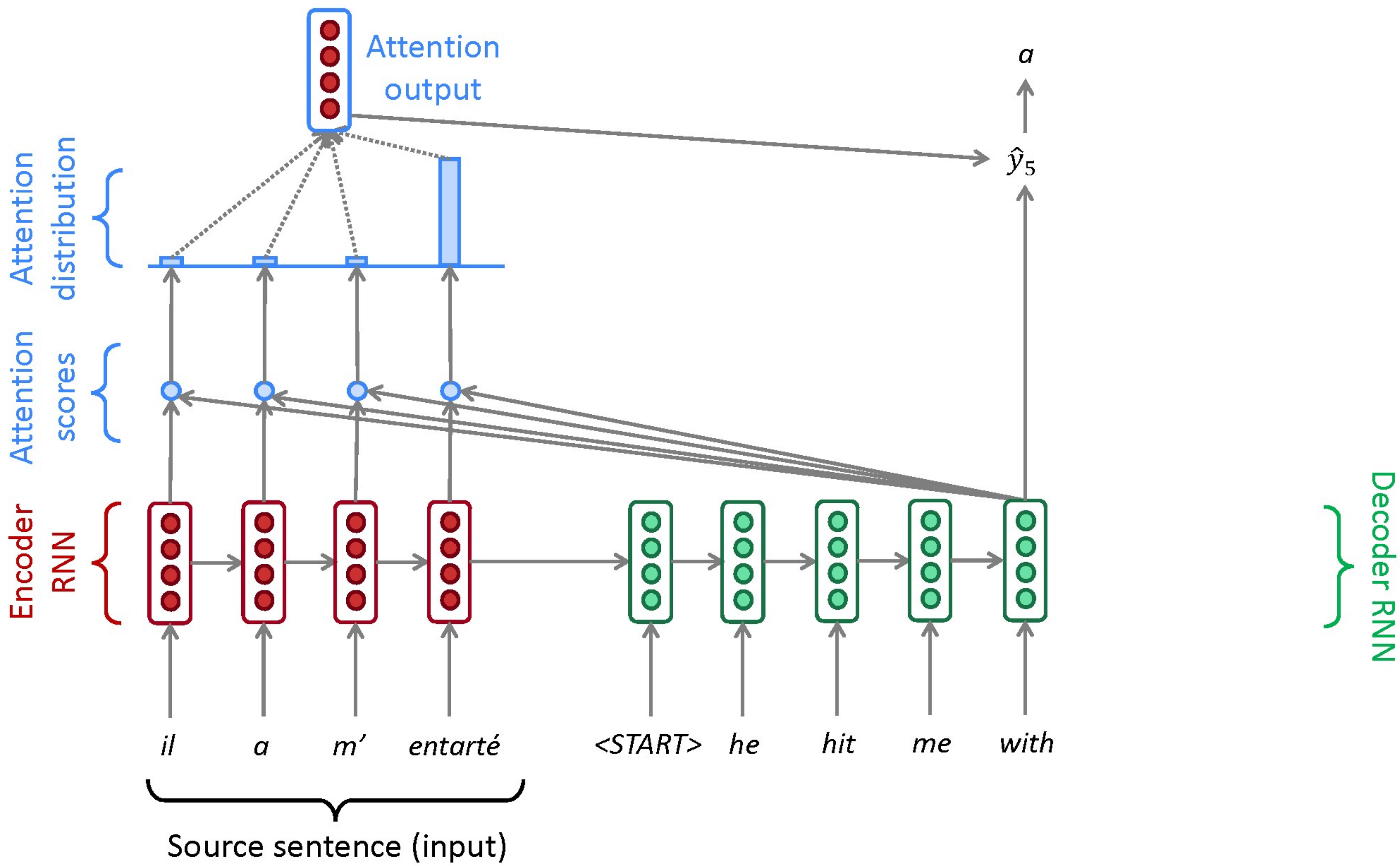
# Sequence-to-sequence with attention



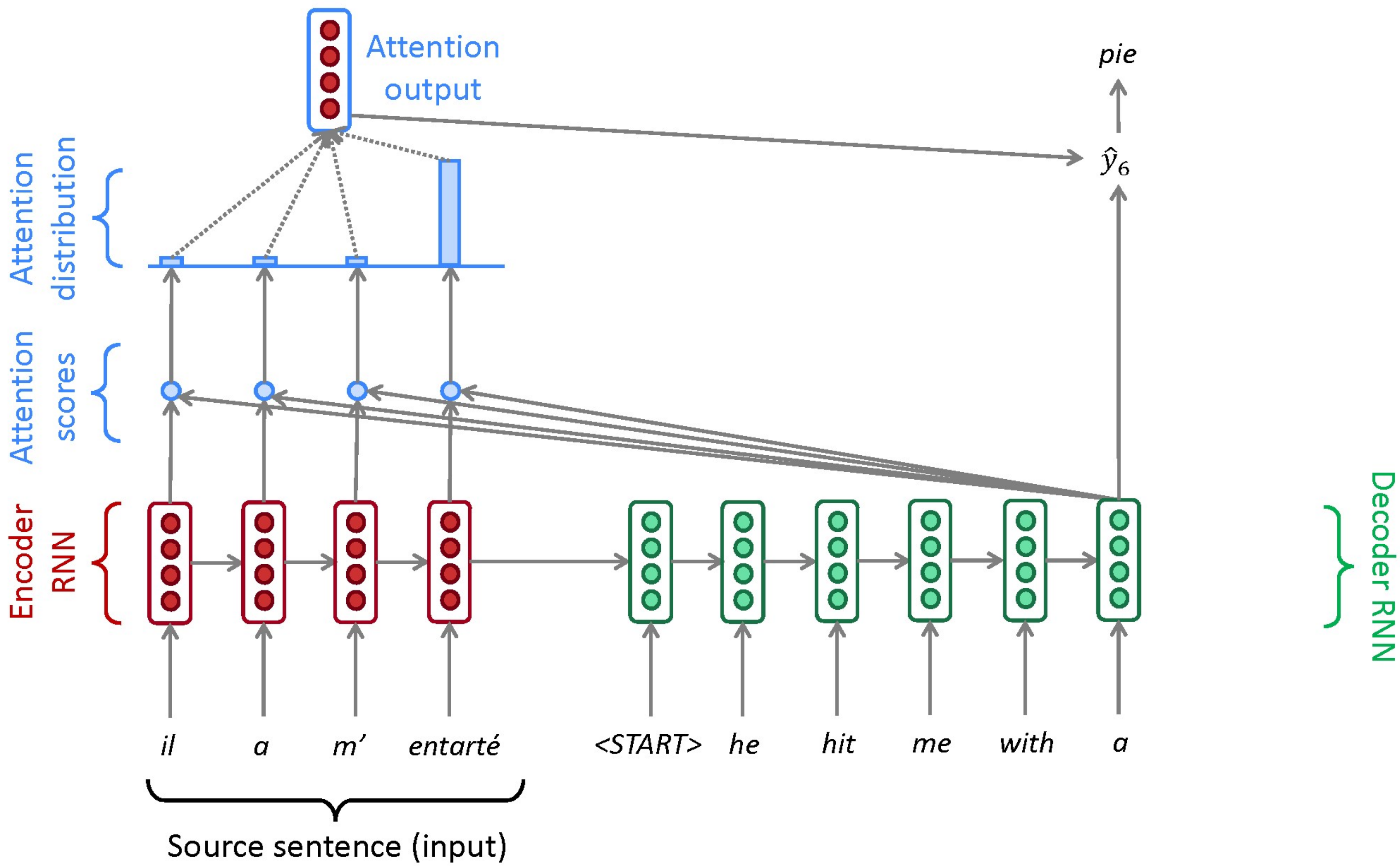
# Sequence-to-sequence with attention



# 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)

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

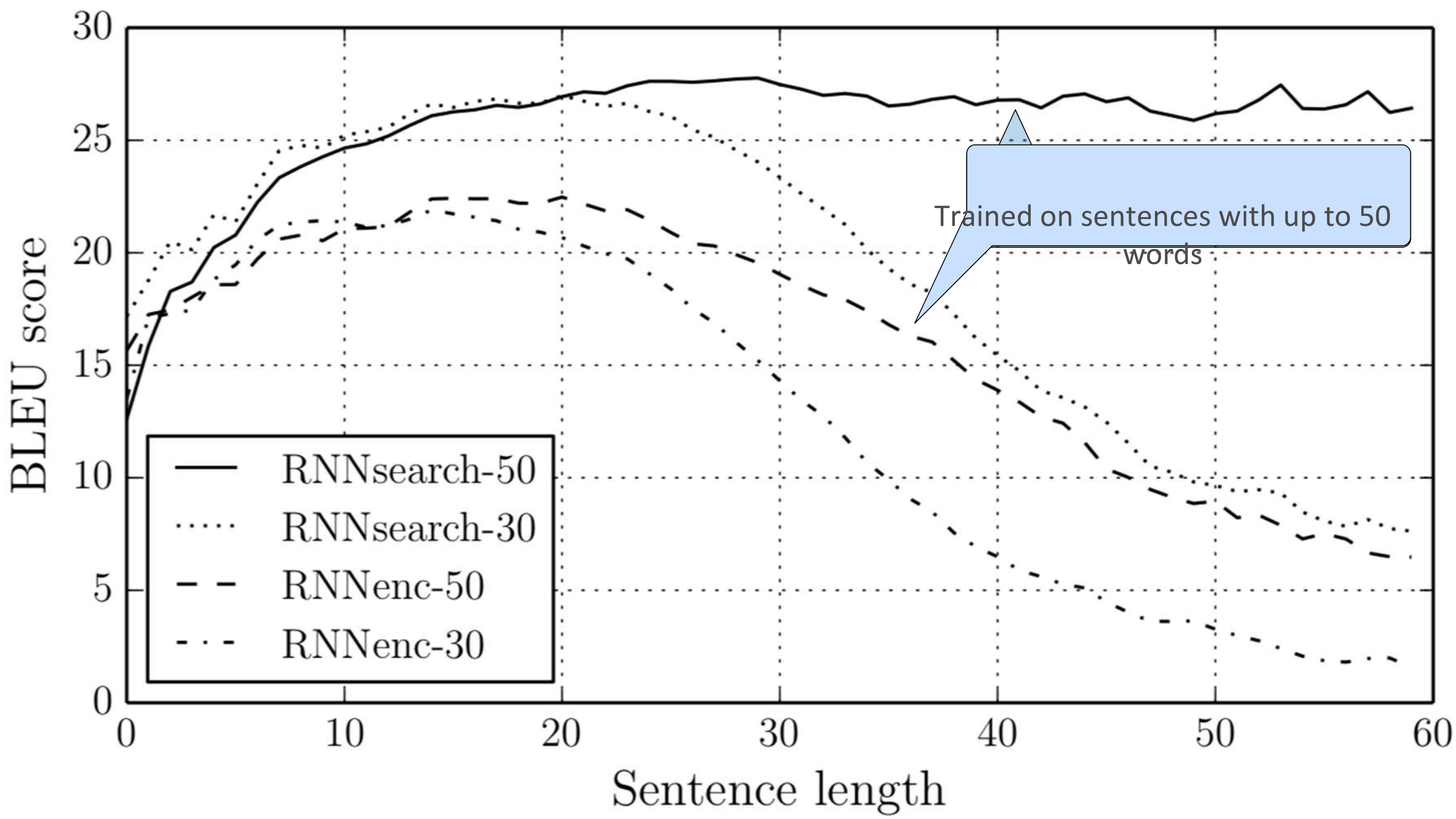
- We use $\alpha^t$ to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

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

- 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

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

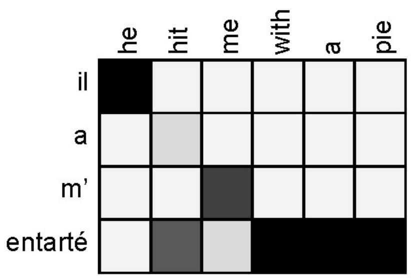
# Impact of Attention on Long Sequence Generation




(Badhanau et al., 2015) Neural Machine Translation by Jointly Learning to Align and Translate

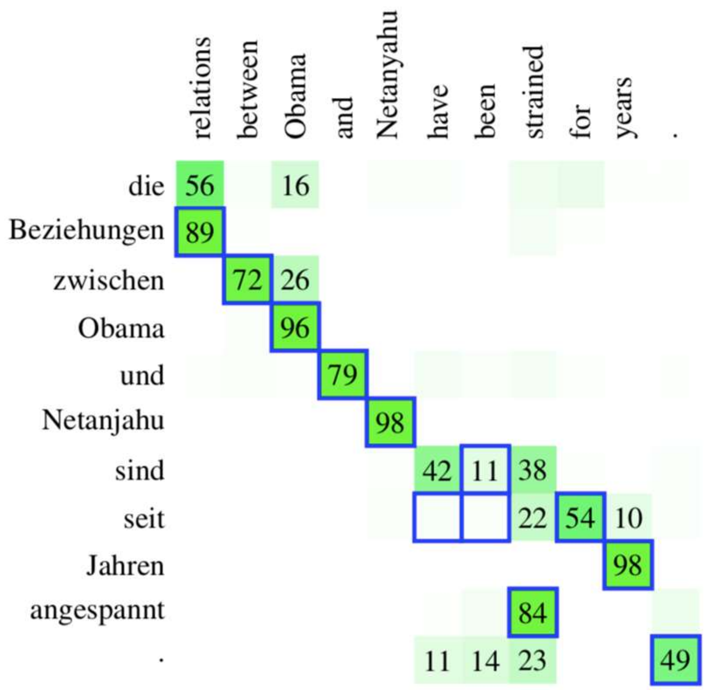
# Attention is great

- 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

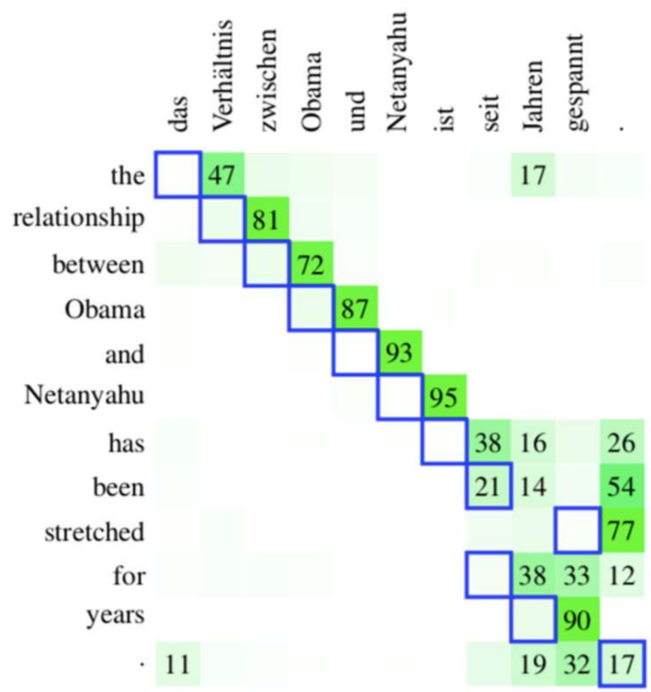


# Attention vs Alignment

Attention activations above 0.1




English-German




German-English

(Koehn & Knowles 2017) Six Challenges for Neural Machine Translation

# 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.
- 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

# Transformers

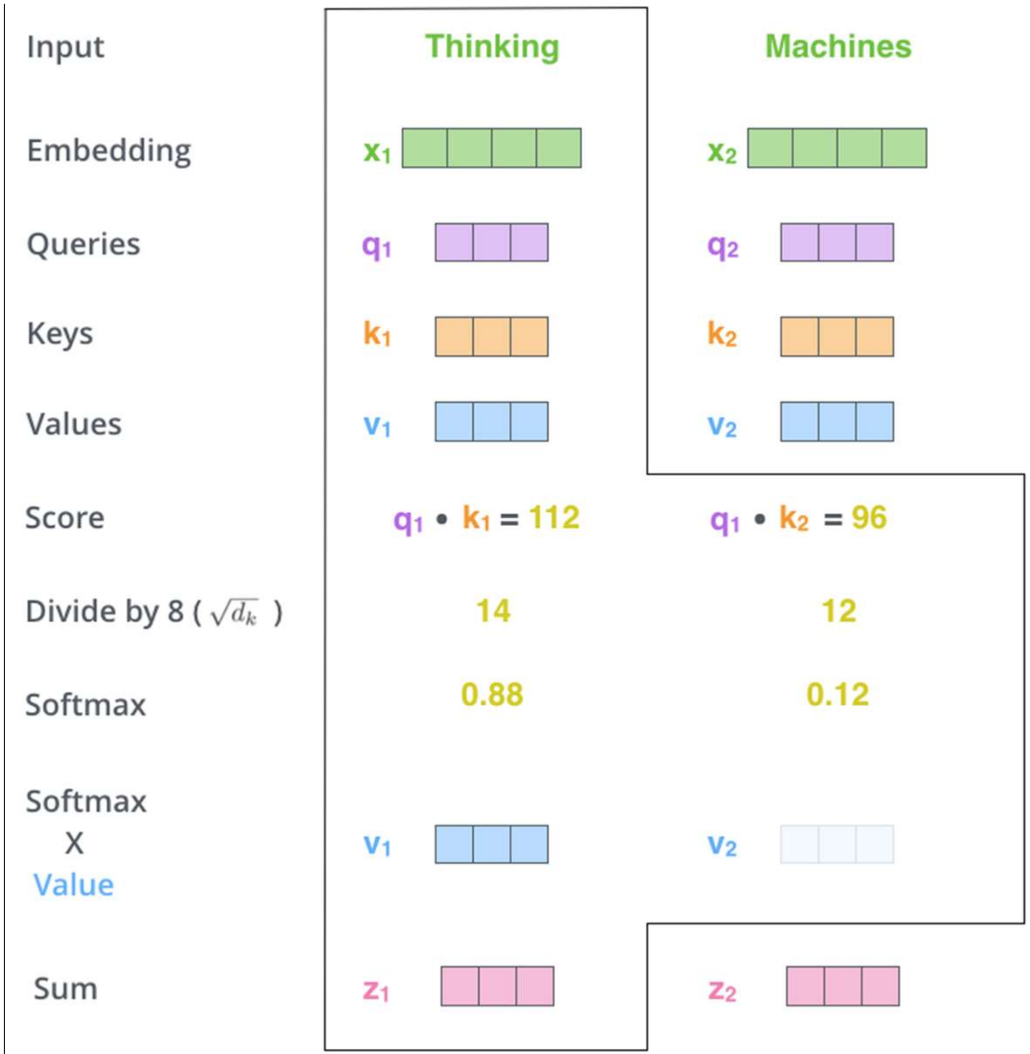
# Transformer

In lieu of an RNN, use ONLY attention!

High throughput & expressivity: compute queries, keys and values as (different) linear transformations of the input.

Attention weights are queries • keys; outputs are sums of weighted values.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$




(Vaswani et al., 2017) Attention is All You Need

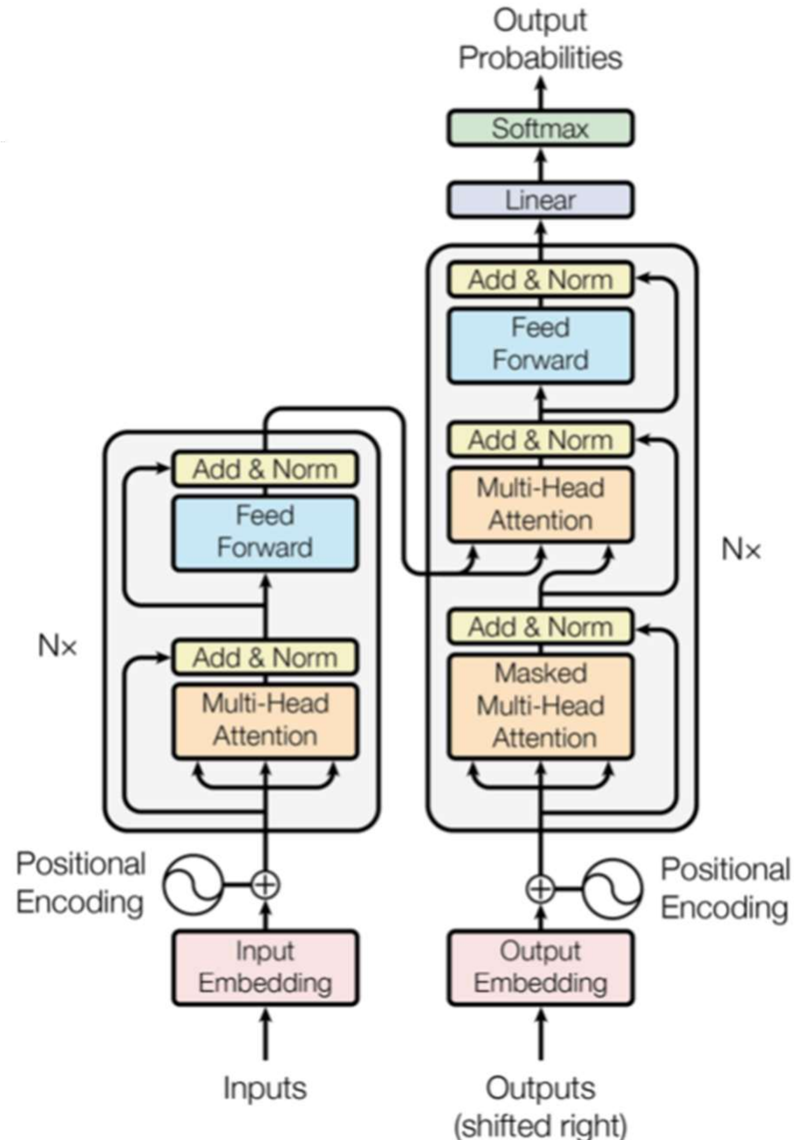
Figure: http://jalammar.github.io/illustrated-transformer/

# Transformer Architecture

- Layer normalization
  ("Add & Norm" cells)
  helps with RNN+attention architectures as well.

- Positional encodings can be learned or based on a formula that makes it easy to represent distance.

| | EN-DE |
|---|---|
| ByteNet [18] | 23.75 |
| Deep-Att + PosUnk [39] | |
| GNMT + RL [38] | 24.6 |
| ConvS2S [9] | 25.16 |
| MoE [32] | 26.03 |
| Deep-Att + PosUnk Ensemble [39] | |
| GNMT + RL Ensemble [38] | 26.30 |
| ConvS2S Ensemble [9] | 26.36 |
| Transformer (base model) | 27.3 |
| Transformer (big) | **28.4** |




(Vaswani et al., 2017) Attention is All You Need

## Some Transformer Concerns

**Problem**: Bag-of-words representation of the input.
**Remedy**: Position embeddings are added to the word embeddings.

**Problem**: During generation, can't attend to future words.
**Remedy**: Masked training that zeroes attention to future words.

**Problem**: Deep networks needed to integrated lots of context.
**Remedies**: Residual connections and multi-head attention.

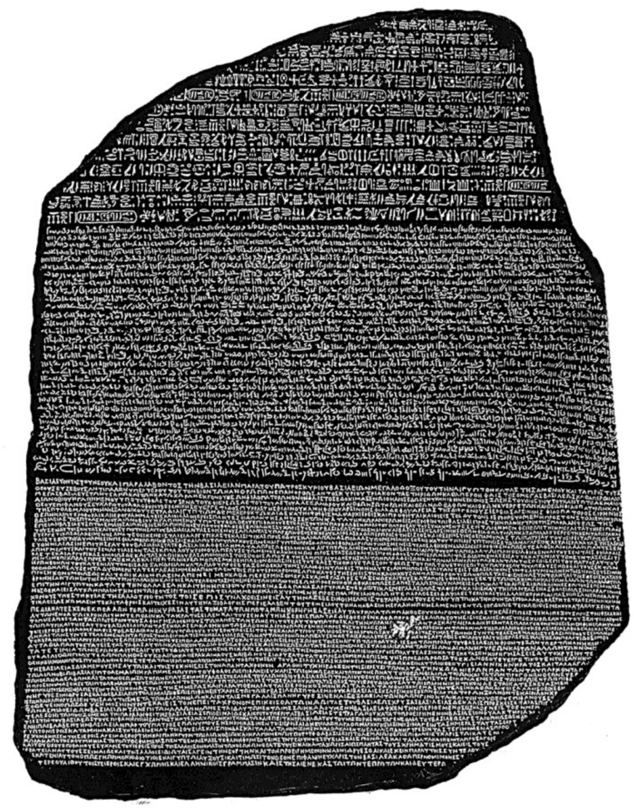
**Problem**: Optimization is hard.
**Remedies**: Large mini-batch sizes and layer normalization.

# Training Data

# Bitexts

Where do bitexts come from?

- Careful, low level / literal translations: organizational translation processes (eg parliamentary proceedings), multilingual newsfeeds, etc
- Discovered translations (ad hoc translations on webpages, etc)
- Loose translations (multilingual Wikipedia, etc)
- Synthetic data (distillation, backtranslation, etc)



# Back Translations

Synthesize an en-de parallel corpus by using a de-en system to translate monolingual de sentences.

- Better generating systems don't seem to matter much.
- Can help even if the de sentences are already in an existing en-de parallel corpus!

| system | EN→DE dev | EN→DE test | DE→EN dev | DE→EN test |
|---|---|---|---|---|
| baseline | 22.4 | 26.8 | 26.4 | 28.5 |
| +synthetic | 25.8 | 31.6 | 29.9 | 36.2 |
| +ensemble | 27.5 | 33.1 | 31.5 | 37.5 |
| +r2l reranking | **28.1** | **34.2** | **32.1** | **38.6** |

Table 2: English↔German translation results (BLEU) on dev (newstest2015) and test (newstest2016). Submitted system in bold.

(Sennrich et al., 2015) Improving Neural Machine Translation Models with Monolingual Data
(Sennrich et al., 2016) Edinburgh Neural Machine Translation Systems for WMT 16

# Subwords

The sequence of symbols that are embedded should be common enough that an embedding can be estimated robustly for each, and all symbols have been observed during training.

**Solution 1**: Symbols are words with rare words replaced by UNK.

- Replacing UNK in the output is a new problem (like alignment).
- UNK in the input loses all information that might have been relevant from the rare input word (e.g., tense, length, POS).

**Solution 2**: Symbols are subwords.

- Byte-Pair Encoding is the most common approach.
- Other techniques that find common subwords aren't reliably better (but are somewhat more complicated).
- Training on many sampled subword decompositions improves out-of-domain translations.

(Sennrich et al., 2016) Neural Machine Translation of Rare Words with Subword Units
(Kudo, 2018) Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates

```
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
        'n e w e s t </w>':6, 'w i d e s t </w>':3}
def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out
```

```
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
```

(Sennrich et al., 2016) Neural Machine Translation of Rare Words with Subword Units

# BPE Example

| system | sentence |
|---|---|
| source | health research institutes |
| reference | Gesundheitsforschungsinstitute |
| word-level (with back-off) | Forschungsinstitute |
| character bigrams | Fo\|rs\|ch\|un\|gs\|in\|st\|it\|ut\|io\|ne\|n |
| BPE | Gesundheits\|forsch\|ungsin\|stitute |

Example from Rico Sennrich

# Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- 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
- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

## 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
- **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

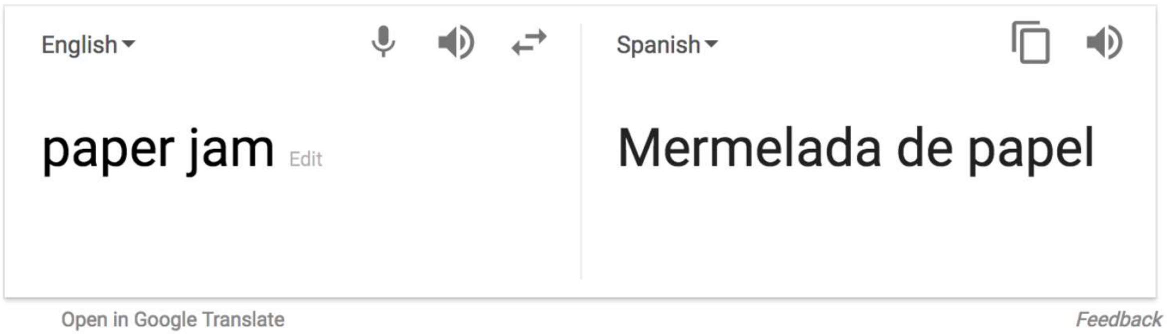
## So is Machine Translation solved?

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

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

# So is Machine Translation solved?

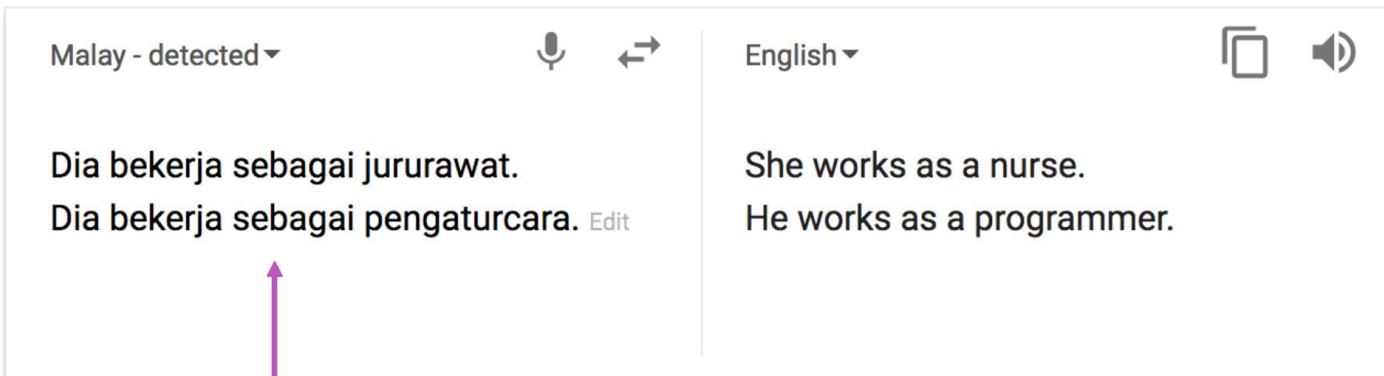
- **Nope!**
- Using common sense is still hard




?

# So is Machine Translation solved?

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

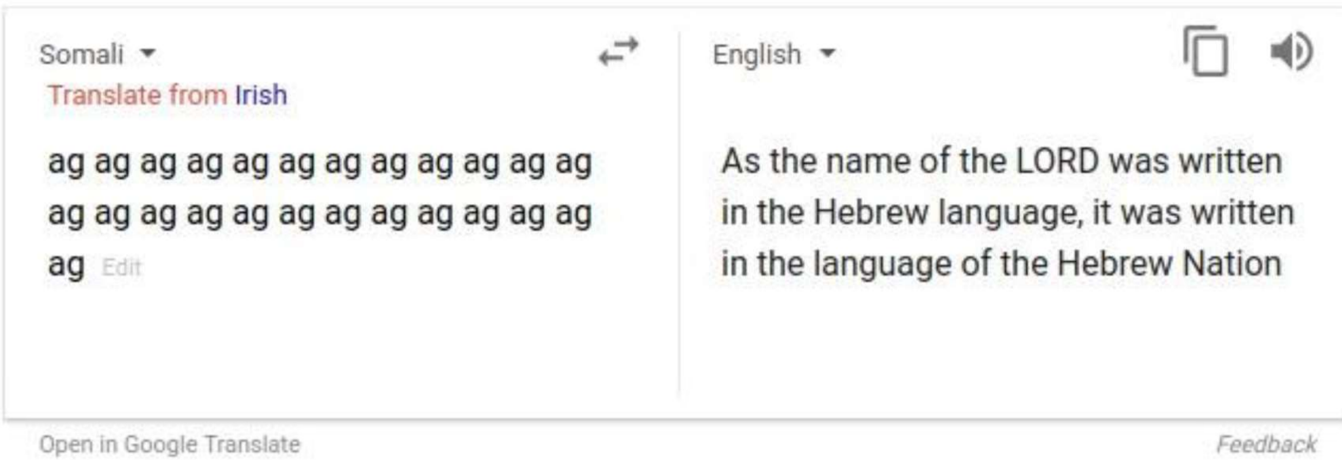



Didn’t specify gender

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

# So is Machine Translation solved?

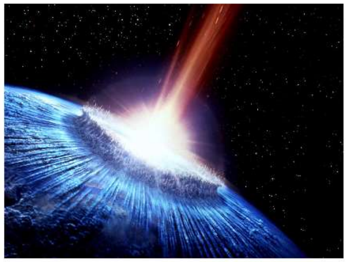
- Nope!
- Uninterpretable systems do strange things




**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

## Summary

- 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!