Lecture 12 - Word embeddings, sequence-to-sequence translation with attention

DD2424

April 29, 2019

Summary of today's lecture

- Learning word embeddings - word2vec.
- Review of Sequence-to-Sequence Translation & Neural Machine Translation.
- Neural Machine Translation with Attention.
- Image captioning with Attention.

Note: Most of this lecture is based on slides taken or adapted from the course CS224n: Natural Language Processing with Deep Learning Lectures 2 and 10.

Word embeddings

Representing words as discrete symbols

- In traditional NLP, words regarded as discrete symbols:

  conference, *hotel*, *motel*, ...

- Words can be represented by **one-hot** vectors:

$$\text{motel} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \text{hotel} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Vector dimension = # of words in vocabulary

Problem with discrete symbols

- No natural notion of **similarity** for one-hot vectors!
- In the example

$$\text{motel} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \text{hotel} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

the vectors are orthogonal even though they refer to similar concepts.

- **Solution**: Learn to encode similarity in the vectors themselves.

Represent words by their context

- Key Idea:

  A word's meaning is given by the words that frequently appear close-by.

  - "*You shall know a word by the company it keeps*" (J. R. Firth 1957)
  - One of the most successful ideas of modern statistical NLP!

- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of w to build up a representation of w:

... government debt problems turning into banking crises as happened in 2009...
... saying that Europe needs unified banking regulation to replace the hodgepodge...
... India has just given its banking system a shot in the arm...

These context words will represent banking.

Represent words by their context

- Build a dense vector for each word:

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

- Vector chosen so that it is similar to vectors of words that appear in similar contexts.
- Word vectors sometimes also called word embeddings or word representations.

Word2vec: Overview

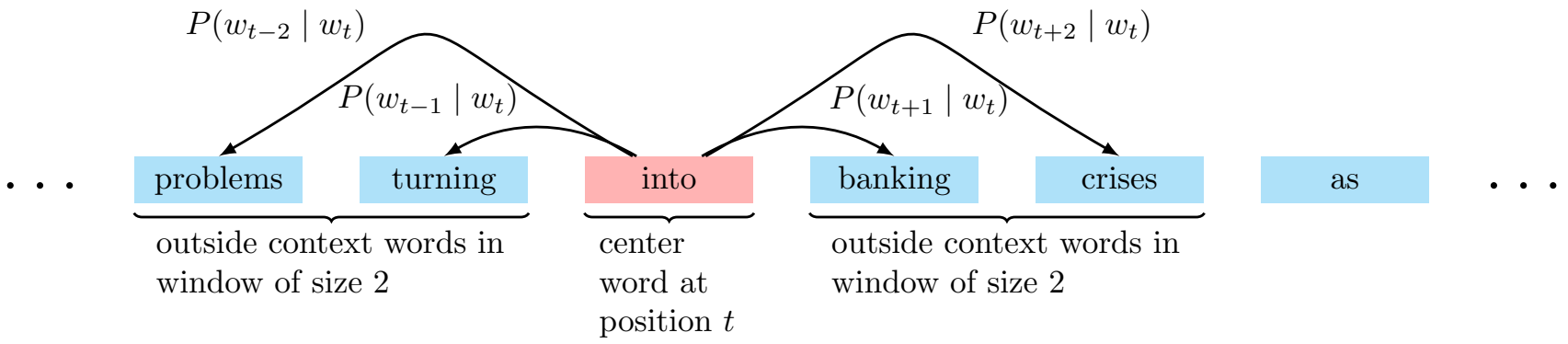
Word2vec(Mikolov et al. 2013): framework for learning word vectors.

Idea:

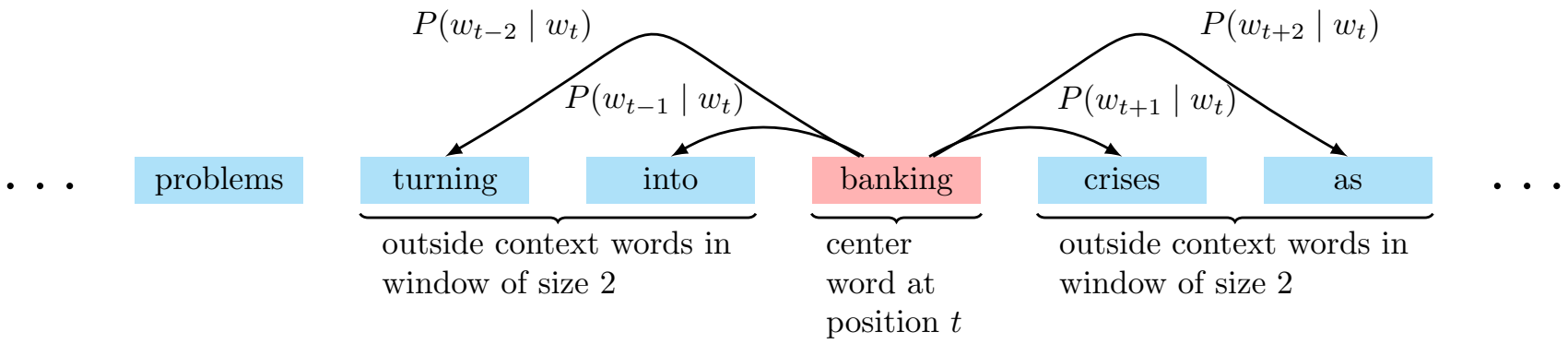
- Have a large corpus of text.
- Every word in a fixed vocabulary is represented by a vector.
- Go through each position t in the text, which has a centre word c and context (outside) words $o_1, \ldots, o_K$.
- Use the similarity of the word vectors for c and o_k's to calculate the probability of each o_i given c (or vice versa).
- Keep adjusting the word vectors to maximize these probabilities.

Word2vec: Overview

Example windows for computing $P_{W_j|W_0,\Theta}(w_{t+j} \mid w_t, \theta)$:



Example windows for computing $P_{W_j|W_0,\Theta}(w_{t+j} \mid w_t, \theta)$:



Word2vec: objective function

For each position $t = 1, \ldots, T$:

- Want to predict context words w_{t+j} within a window of fixed size m given centre word w_t.

How?

- Define likelihood of the data, $\mathcal{D}$, (making lots of independence assumptions) as

$$L(\mathcal{D}, \theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P_{W_j|W_0,\Theta}(w_{t+j} \mid w_t, \theta)$$

  where θ denotes all the parameters to be found.

- The objective function, $J(\mathcal{D}, \theta)$, to be optimized w.r.t. θ as

$$-\frac{1}{T} \log\left(L(\mathcal{D}, \theta)\right) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log\left(P_{W_j|W_0,\Theta}(w_{t+j} \mid w_t, \theta)\right)$$

Word2vec: objective function

- Want to minimize the objective function w.r.t. θ:

$$J(\mathcal{D}, \theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log \left(P_{W_j | W_0, \Theta}(w_{t+j} \mid w_t, \theta) \right)$$

- **Question**: How to calculate $P_{W_j | W_0, \Theta}(w_{t+j} \mid w_t, \theta)$?

- **Solution**:
  - Have two vector representations per word w:
    - $\mathbf{v}_w$ when w is a centre word.
    - $\mathbf{u}_w$ when w is a context word.
  - For centre word c and context word o:

$$P_{W_j | W_0, \Theta}(o \mid c, \theta) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\mathbf{u}_w^T \mathbf{v}_c)}$$

  - It's SoftMax applied to the dot-product between the context word vector $\mathbf{u}_o$ and $\mathbf{v}_c$.

To train the model: Compute all vector gradients!

- Let θ represent all the embedding vectors for the vocab
- With d-dimensional embedding vectors and V-many words then $\theta \in \mathbb{R}^{2dV}$ with

$$\theta = \begin{pmatrix} \mathbf{v}_{\text{aardvark}} \\ \mathbf{v}_{\text{a}} \\ \vdots \\ \mathbf{v}_{\text{zebra}} \\ \mathbf{u}_{\text{aardvark}} \\ \mathbf{u}_{\text{a}} \\ \vdots \\ \mathbf{u}_{\text{zebra}} \end{pmatrix}$$

- Remember! Every word has two vectors.

To train the model

- Use stochastic gradient descent/ mini-batch gradient descent.
- Each window is a training sample.
- In each window compute updates for all context word vectors and the centre word vector used in that window.
- Why two vectors?
  - Easier optimization.
  - Average both at the end.

Word2vec: More Details

- Two model variants
  1. Skip-grams
     Predict context words give centre word. (what has been describe so far)
  2. Continuous Bag of Words (CBOW)
     Predict centre word from (bag of) context words.

- Additional efficiency in training.
  1. Negative sampling

The skip-gram model and negative sampling

- From paper: "*Distributed Representations of Words and Phrases and their Compositionality.*" by Mikolov et al., 2013.

- Before had objective function:

$$J(\mathcal{D}, \theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log \left(P_{W_j | W_0, \Theta}(w_{t+j} \mid w_t, \theta) \right)$$

with

$$P_{W_j | W_0, \Theta}(o \mid c, \theta) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\mathbf{u}_w^T \mathbf{v}_c)}$$

- In the objective function replace $P_{W_j | W_0, \Theta}(o \mid c; \theta)$ with

$$\log \left(\sigma(\mathbf{u}_o^T \mathbf{v}_c) \right) + \sum_{i=1}^{k} E_{l \sim P_W(w)} \left[\log \left(\sigma(-\mathbf{u}_l^T \mathbf{v}_c) \right) \right]$$

The skip-gram model and negative sampling

- Have replaced $P_{W_j|W_0,\Theta}(o \mid c, \theta)$ in objective function with

$$\log\left(\sigma(\mathbf{u}_o^T \mathbf{v}_c)\right) + \sum_{i=1}^{k} E_{l \sim P_W(w)}\left[\log\left(\sigma(-\mathbf{u}_l^T \mathbf{v}_c)\right)\right]$$

- **Intuition**:

  task now is to use logistic regression to distinguish the target word $\mathbf{v}_c$ from vocab words $\mathbf{u}_l$ randomly drawn from $P_W(w)$

- k negative samples are used for each data sample.

The skip-gram model and negative sampling

- Simplify further by replacing $P_{W_j|W_0,\Theta}(o \mid c, \theta)$ in objective function with

$$\log\left(\sigma(\mathbf{u}_o^T \mathbf{v}_c)\right) + \sum_{i=1}^{k} \log\left(\sigma(-\mathbf{u}_l^T \mathbf{v}_c)\right) \qquad \text{where } l \sim P_W(w)$$

- Take k negative samples.

- Maximize probability that real outside word appears and minimize prob. that random words appear around centre word

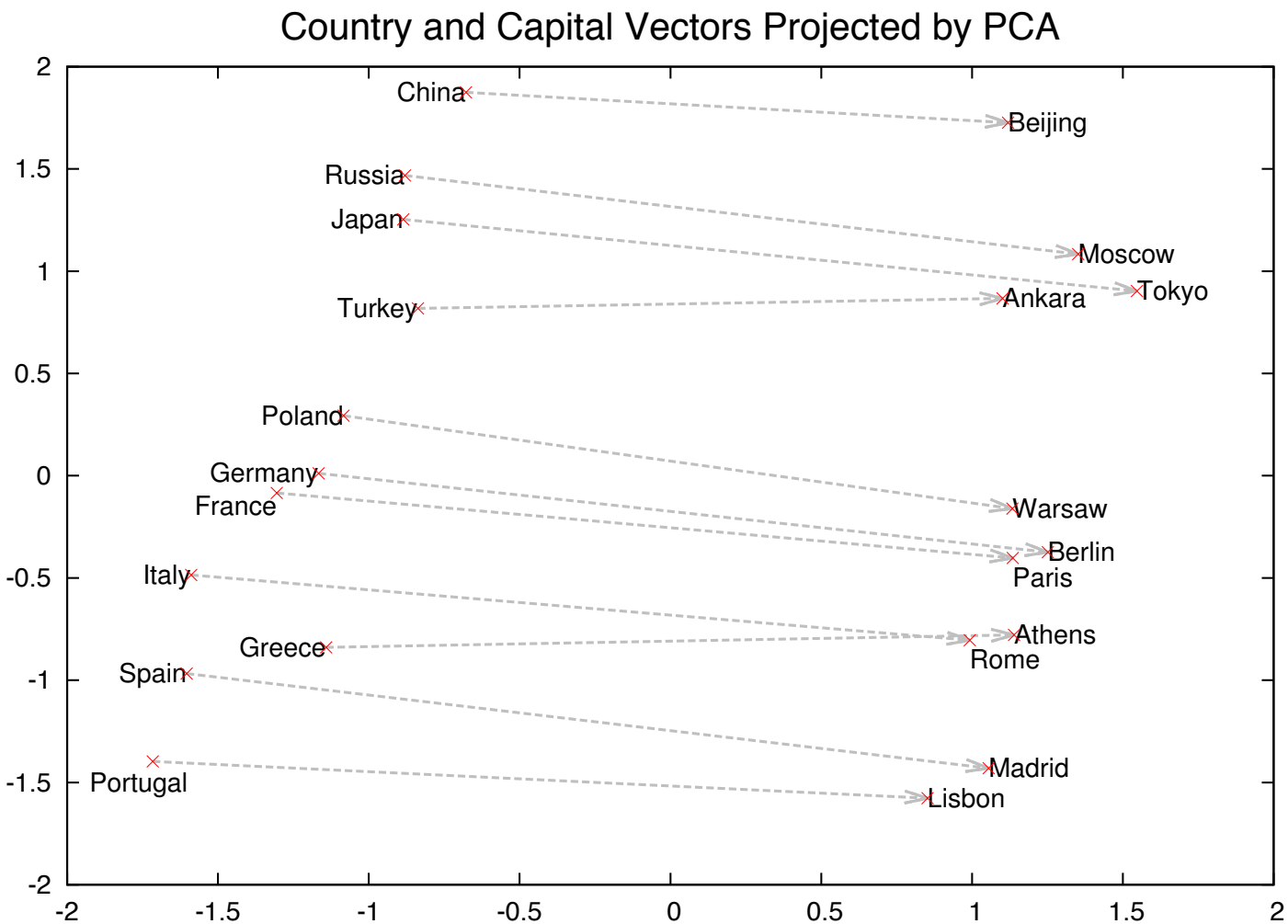
- Draw negative words from

$$P_W(w) = \frac{1}{Z} U_W(w)^{\frac{3}{4}}$$

  where $U_W(w)$ is the histogram of individual words.

- The power proportionally damps the probability more for common than less frequent words $\implies$ increased bias towards sampling less frequent words.

(Partial) Visualization of what word2vec learns



Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities.

(Partial) Visualization of what word2vec learns

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

- Vector compositionality using element-wise addition.
- Four closest tokens to the sum of two vectors are shown using the Skip-gram model.

Another fun word2vec analogies

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Summary of word2vec

- Go through each word of the whole corpus.
- Predict surrounding words of each word.
- Implicitly captures co-occurrence of words.

Alternatives to word2vec

- Explicit word co-occurrence.
  - Build a count matrix X of word co-occurrences in windows.
  - Apply SVD to X
  - This SVD gives a low-dimensional dense embedding vectors of words in the vocab.

  See "*An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence*" by Rohde et al. 2005 for details.

- GloVe by Pennington, Socher, Manning (2014).
  Optimize an objective function combining elements of word2vec and co-occurrence counts. (perform optimization via mini-batch gradient descent.)

Why do we care about word embeddings?

- Can use them for transfer learning!
- You want to build a translation system involving language
  - Initialize the linear transformation of the one-hot encoding of words with their word2vec/GloVe embeddings.
  - Can then fine-tune the embedding dependent on your task.

Back to the sequence-to-sequence deep learning architecture we reviewed in Lecture 9

Remember: Machine Translation

- **Machine Translation** is the task of translating a sentence from a source language to a sentence in the target language.

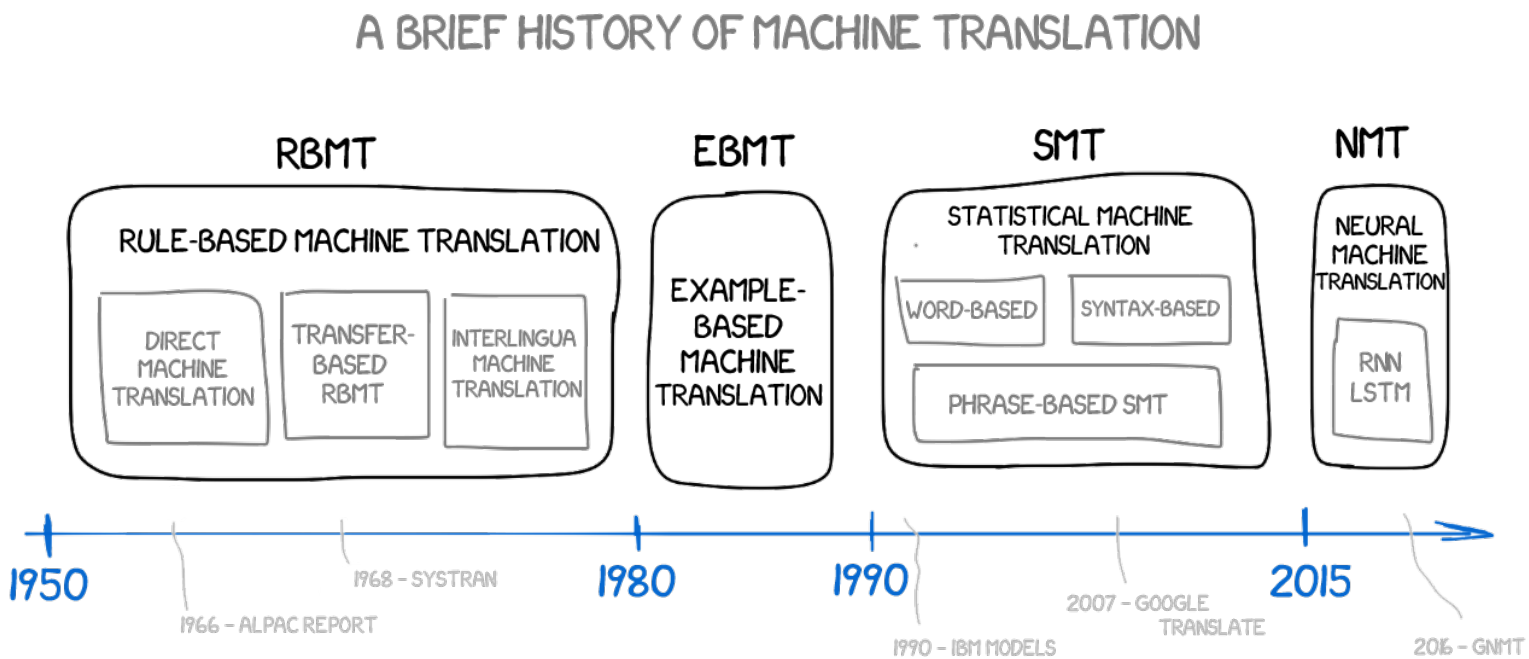
- **Example**:

  L'homme est neé libre, et partout il est dans les fers

  ⇓

  Man is born free, but everywhere he is in chains

Short history of Machine Translation



History of machine translation from the Cold War to deep learning

Picture credit: Blog post - *A history of machine translation from the Cold War to deep learning* by Ilya Pestov.

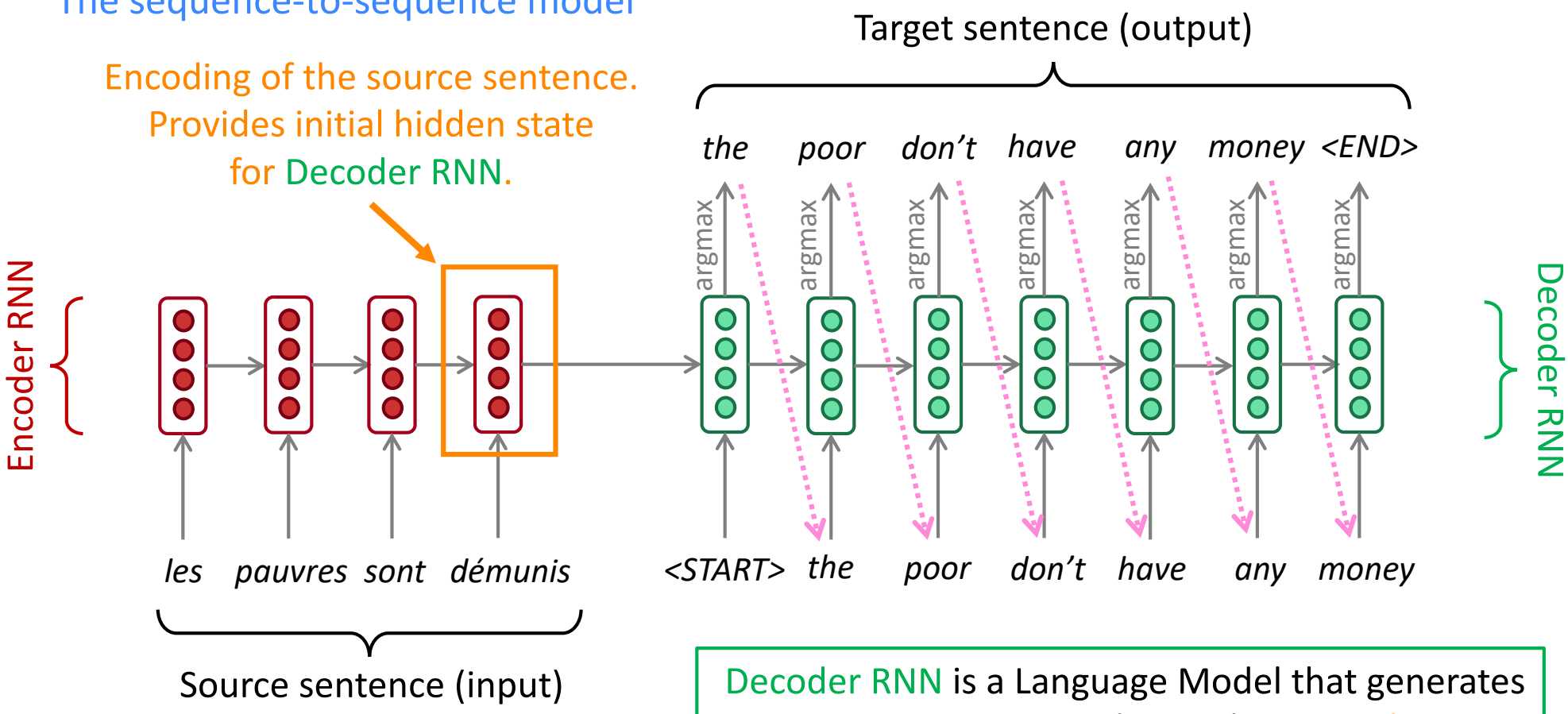
What is Neural Machine Translation?

- **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

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*

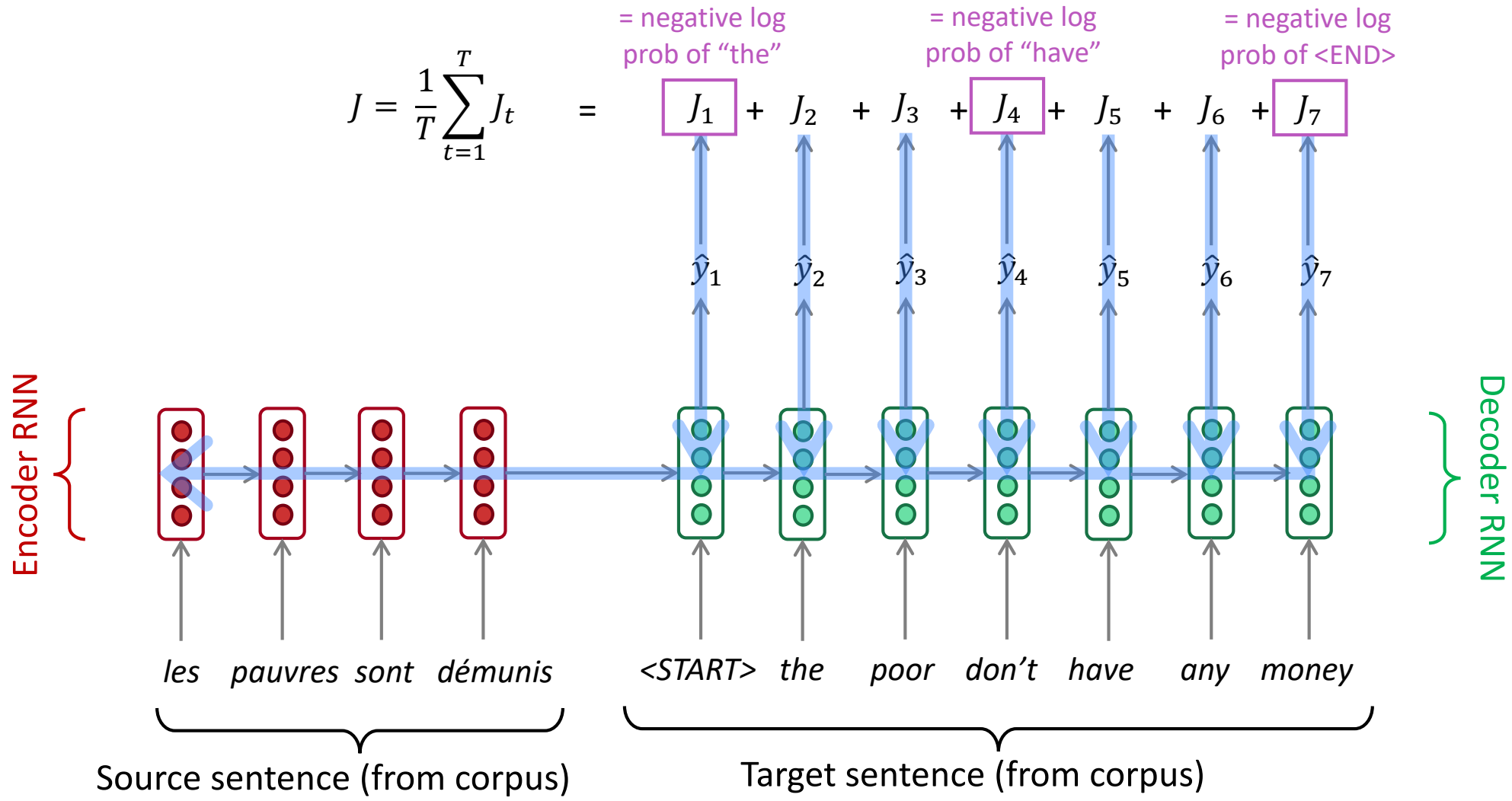
- NMT directly calculates $P(y|x)$:

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

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…

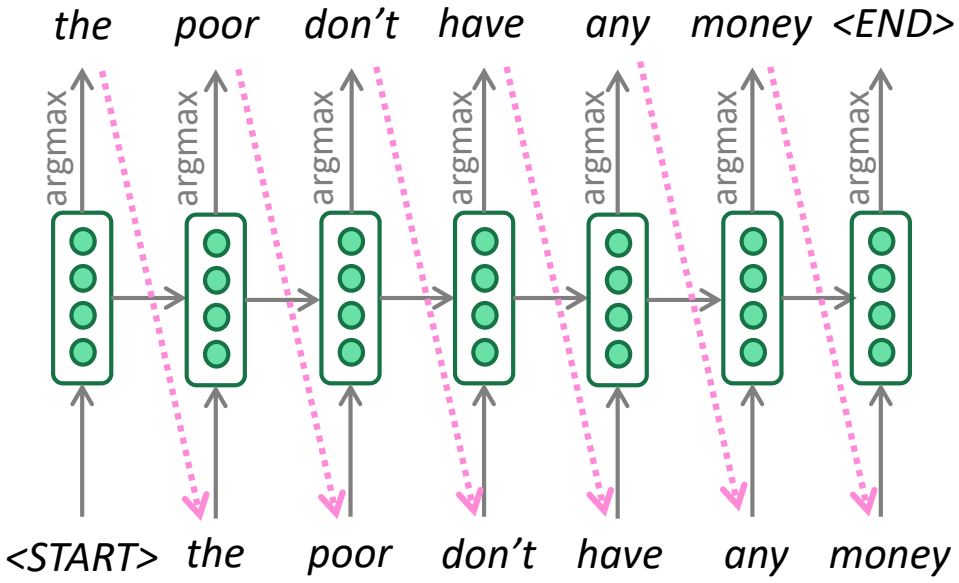
Training a Neural Machine Translation system



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

Better-than-greedy decoding?

- We showed 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?**

Better-than-greedy decoding?

- Greedy decoding has no way to undo decisions!
  - *les pauvres sont démunis (the poor don't have any money)*
  - → *the* ____
  - → *the poor* ____
  - → *the poor are* ____

- Better option: use beam search (a search algorithm) to explore *several* hypotheses and select the best one

Beam search decoding

- Ideally we want to find *y* that maximizes

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

- We could try enumerating all *y* → too expensive!
  - Complexity $O(V^T)$ where *V* is vocab size and *T* is target sequence length

- **Beam search**: On each step of decoder, keep track of the *k* most probable partial translations
  - *k* is the beam size (in practice around 5 to 10)
  - Not guaranteed to find optimal solution
  - But much more efficient!

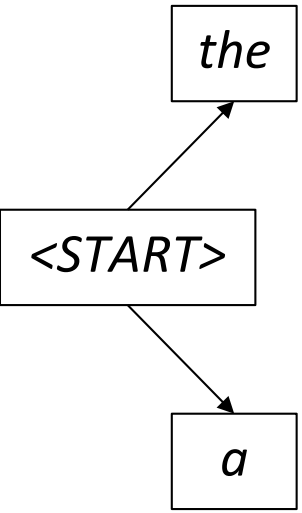
Beam search decoding: example

Beam size = 2

<START>

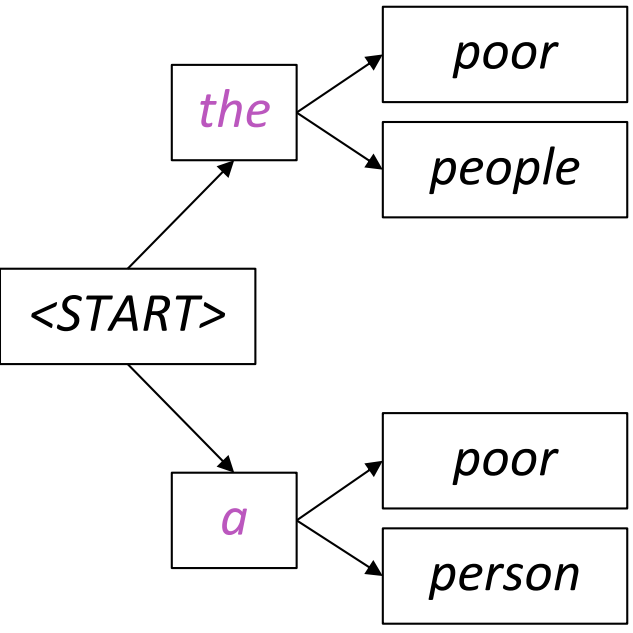
Beam search decoding: example

Beam size = 2



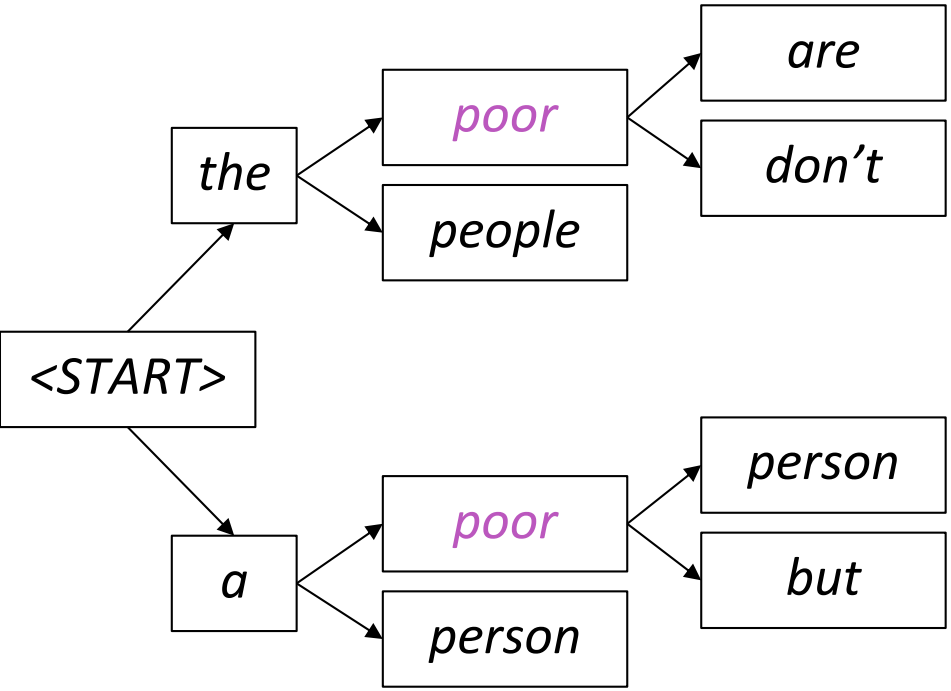
Beam search decoding: example

Beam size = 2



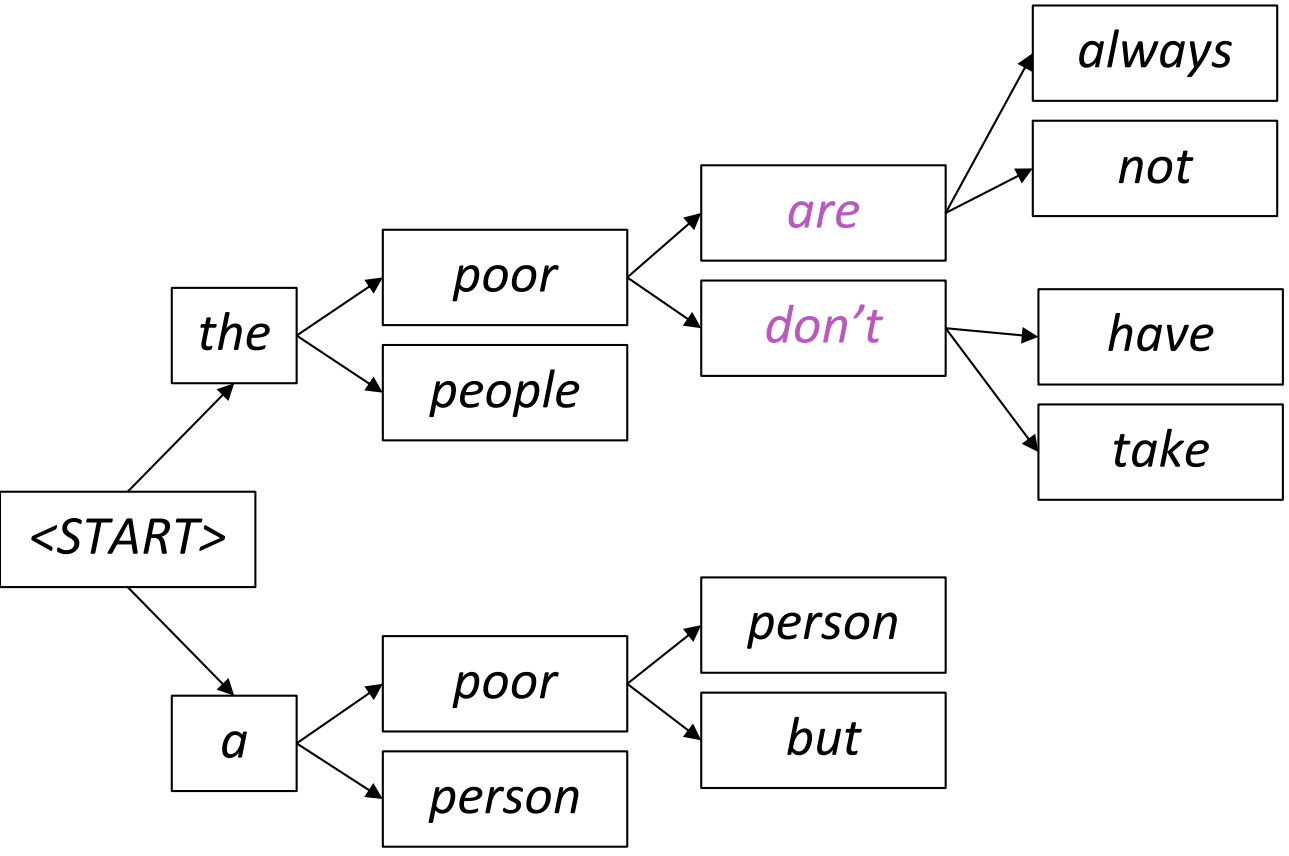
Beam search decoding: example

Beam size = 2



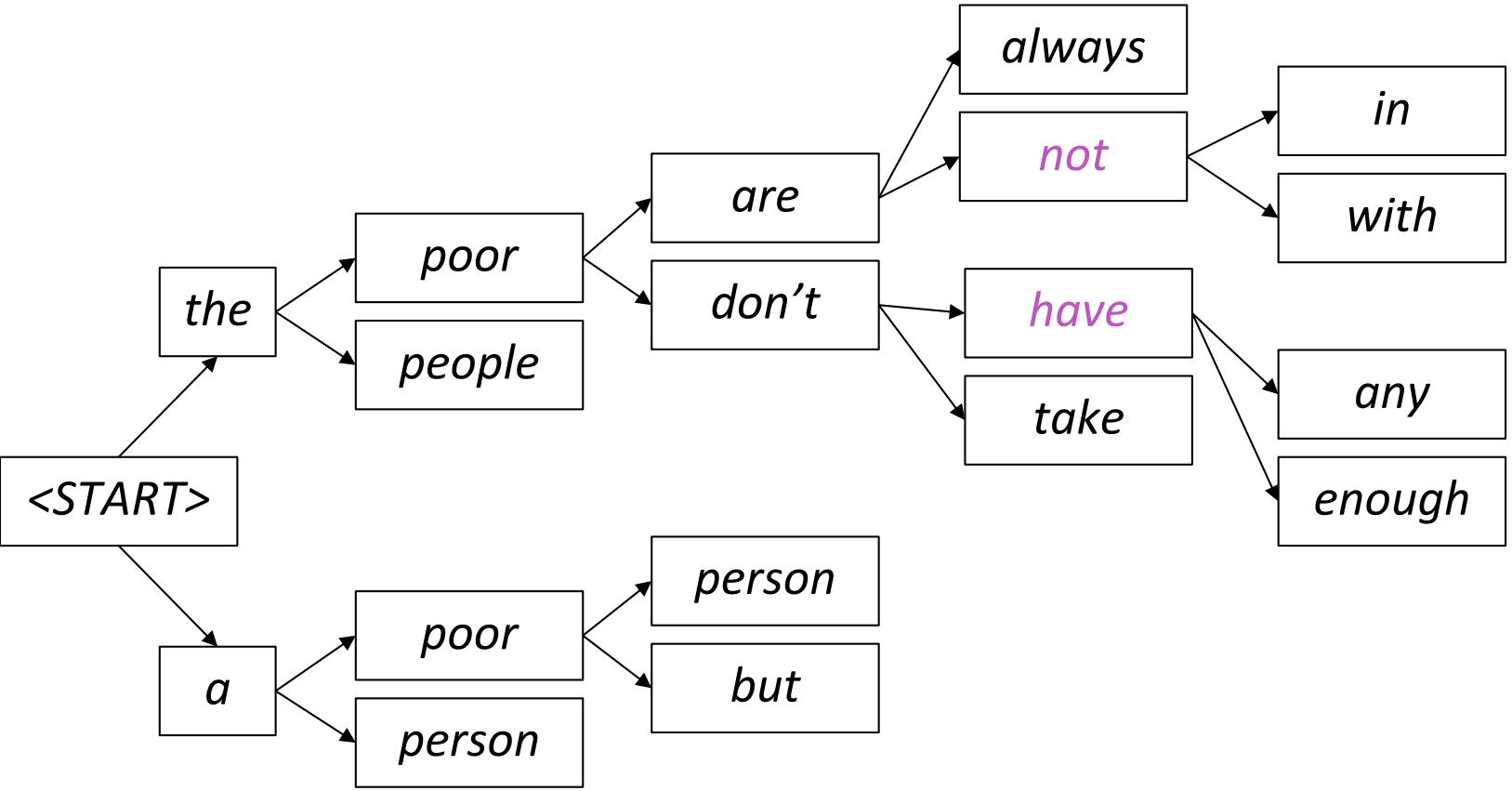
Beam search decoding: example

Beam size = 2



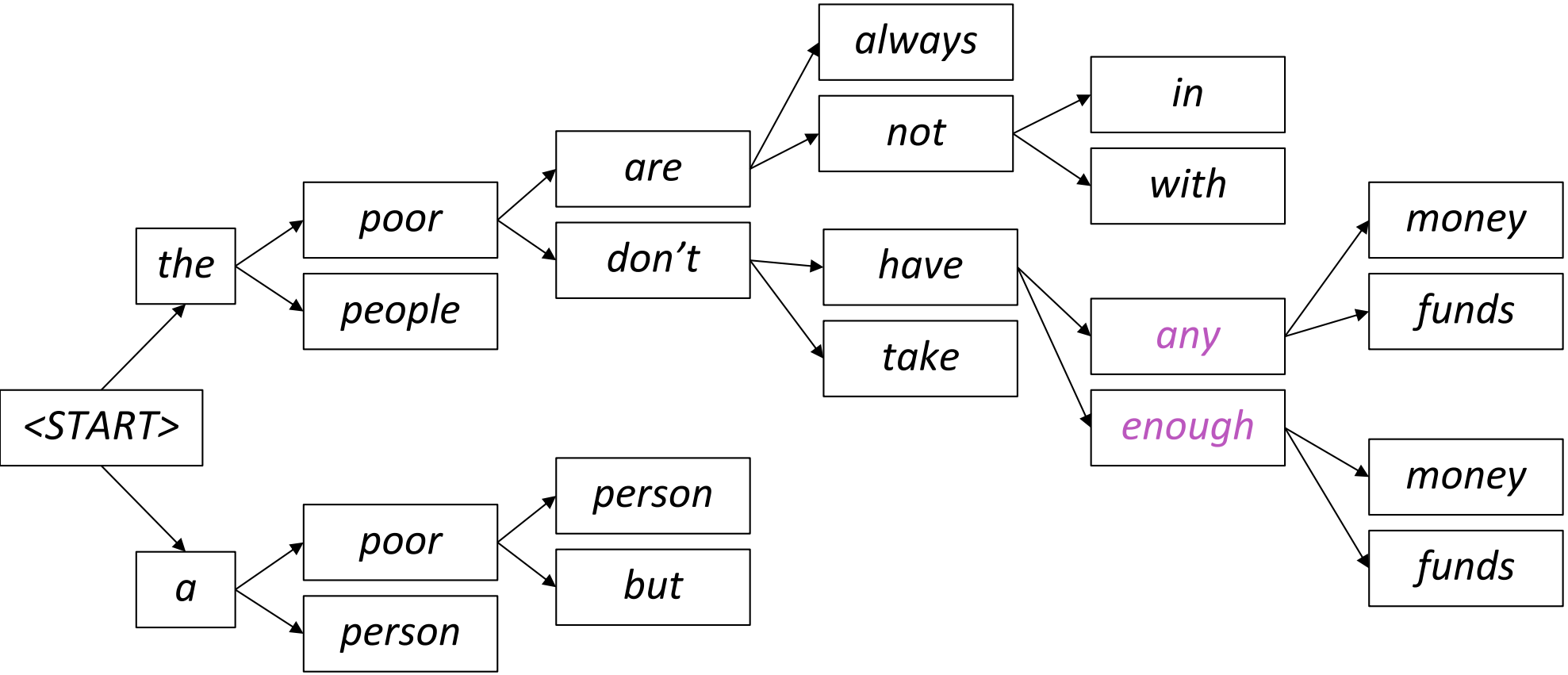
Beam search decoding: example

Beam size = 2



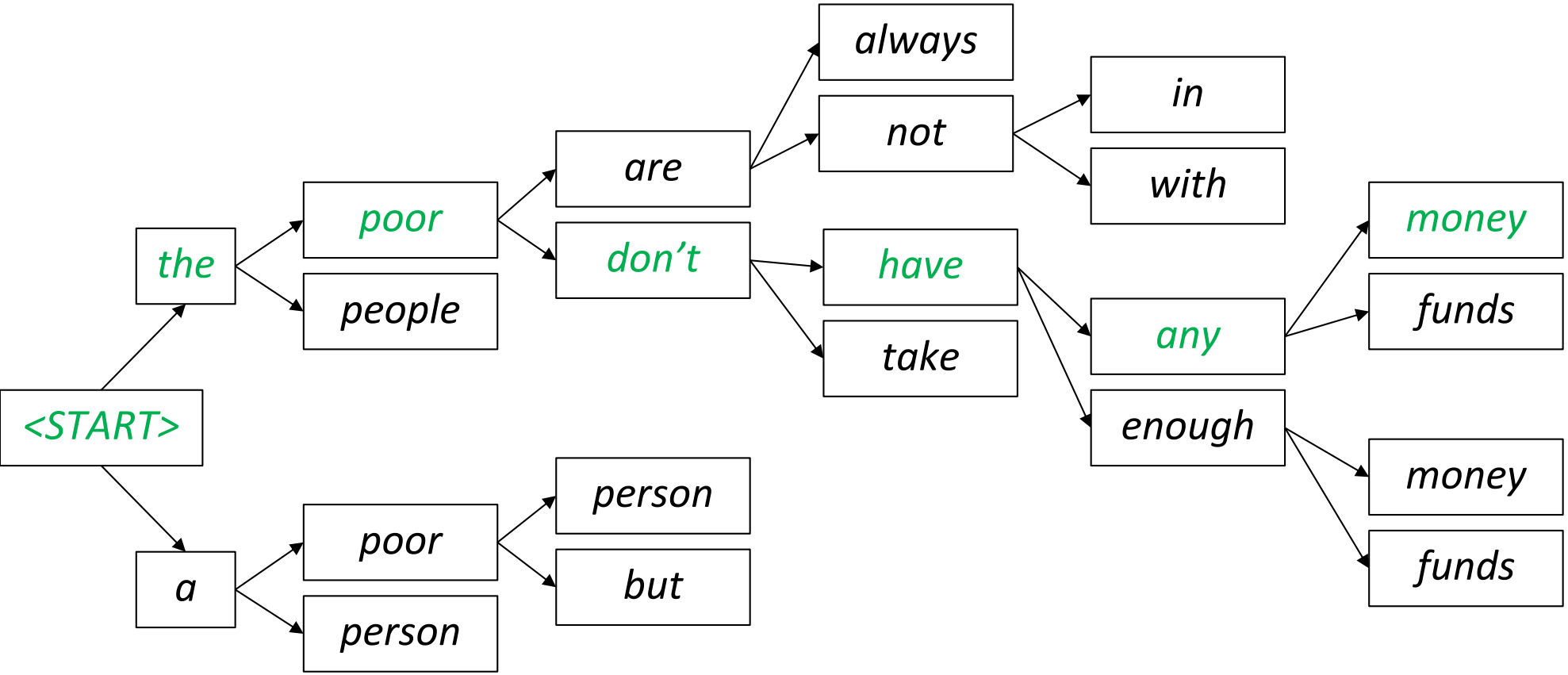
Beam search decoding: example

Beam size = 2



Beam search decoding: example

Beam size = 2



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!

How do we evaluate Machine Translation?

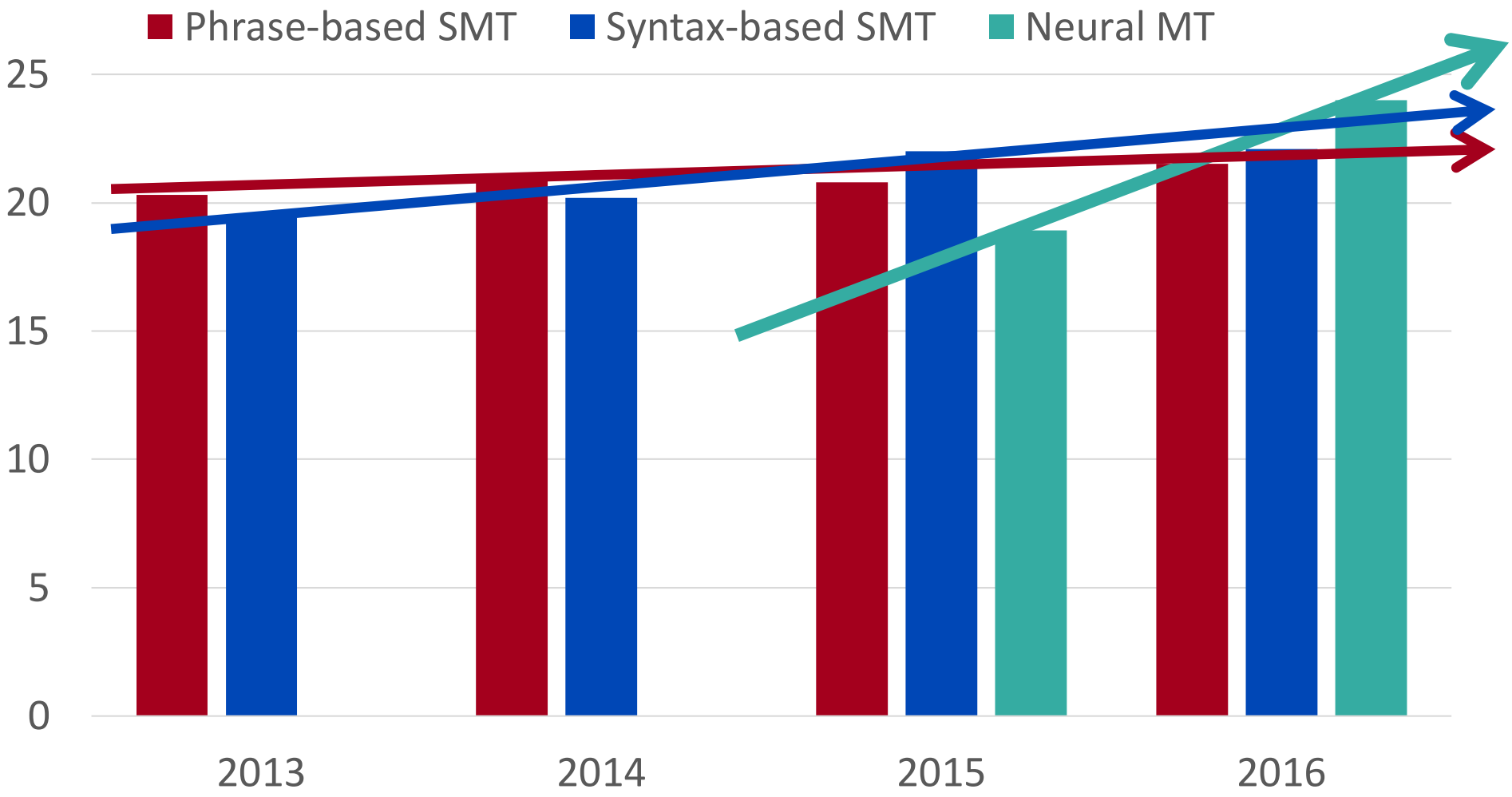
BLEU (**Bi**lingual **E**valuation **U**nderstudy)

- 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 up to 3 or 4-grams)
  - Penalty for too-short system translations

- 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 ☹

MT progress over time

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



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**

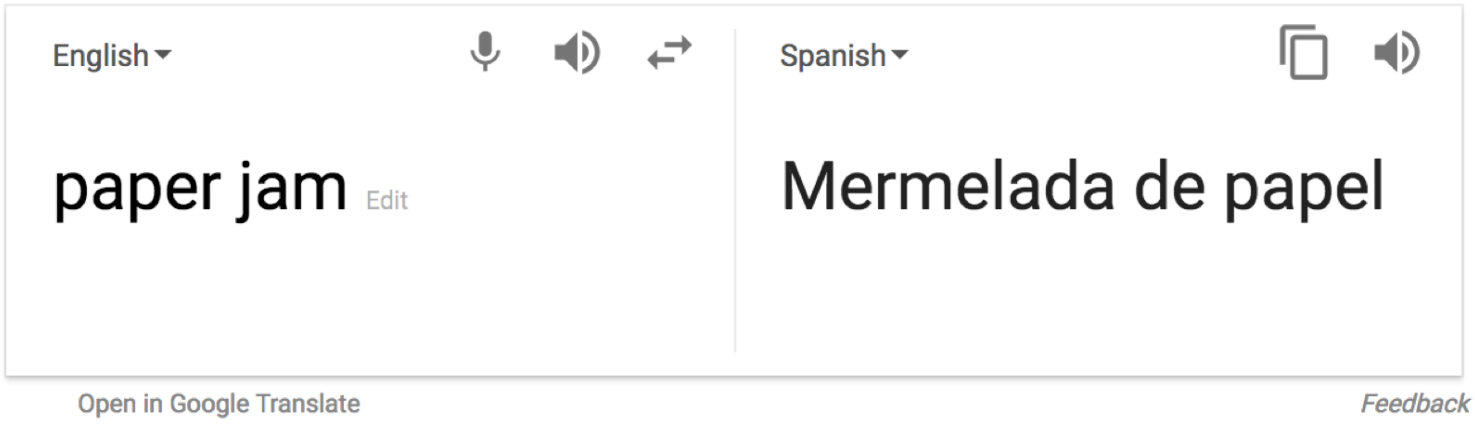
- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

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

So is Machine Translation solved?

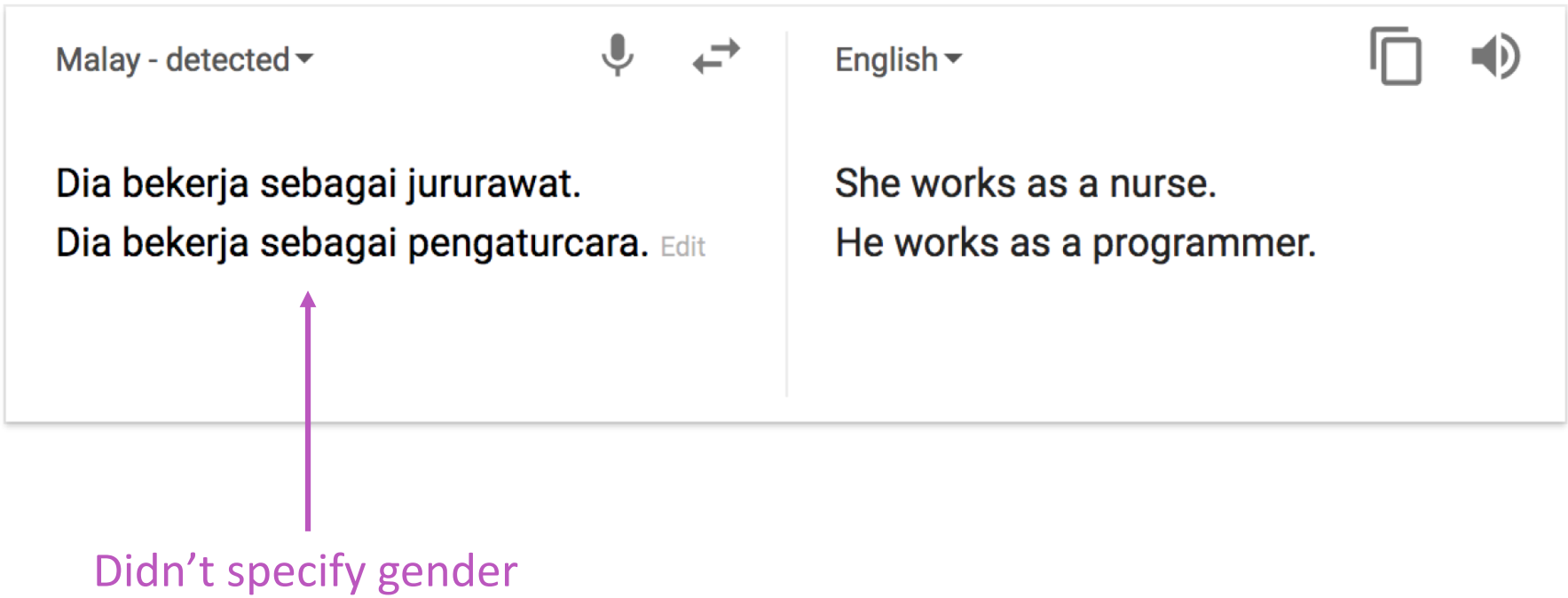
- **Nope!**
- Using common sense is still hard



?

So is Machine Translation solved?

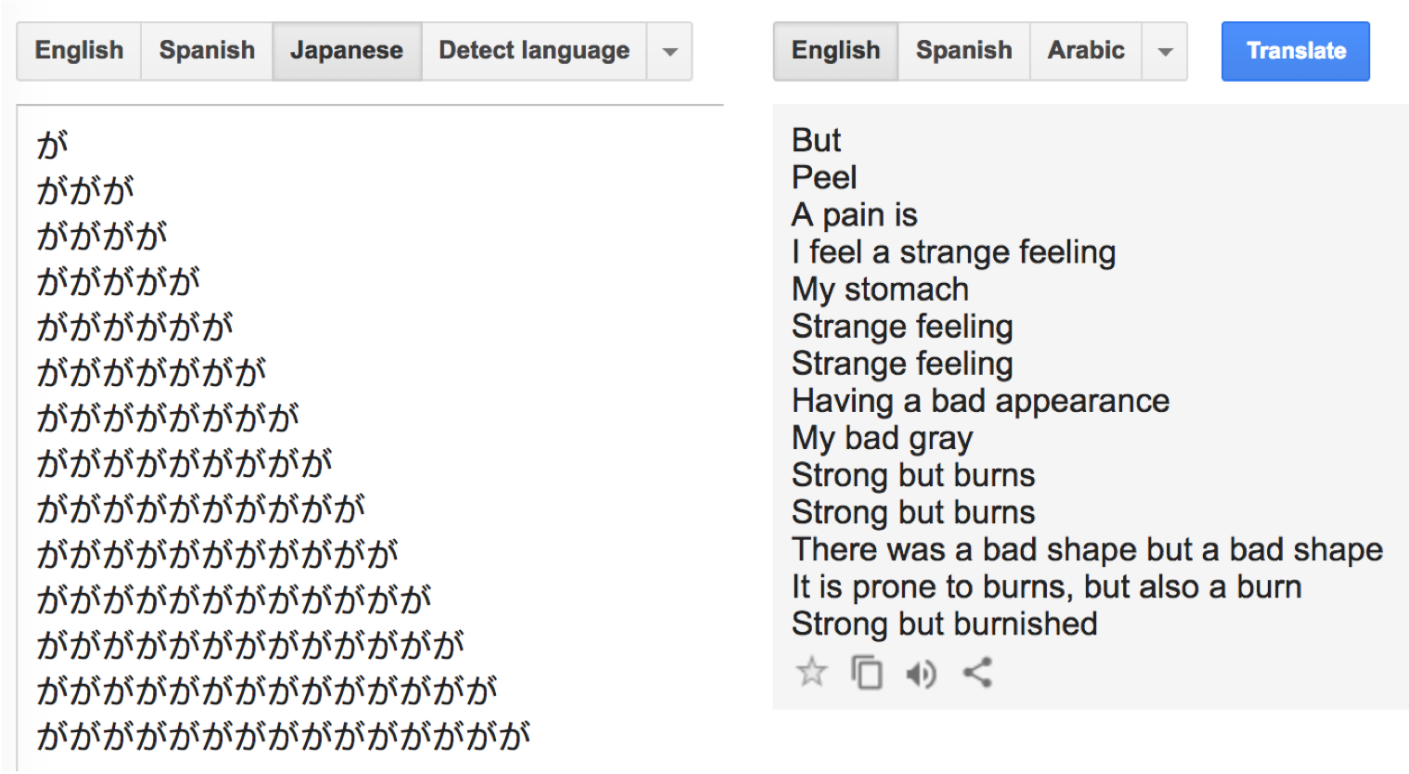
- **Nope!**
- NMT picks up biases in training data



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



Source: http://languagelog.ldc.upenn.edu/nll/?p=35120#more-35120

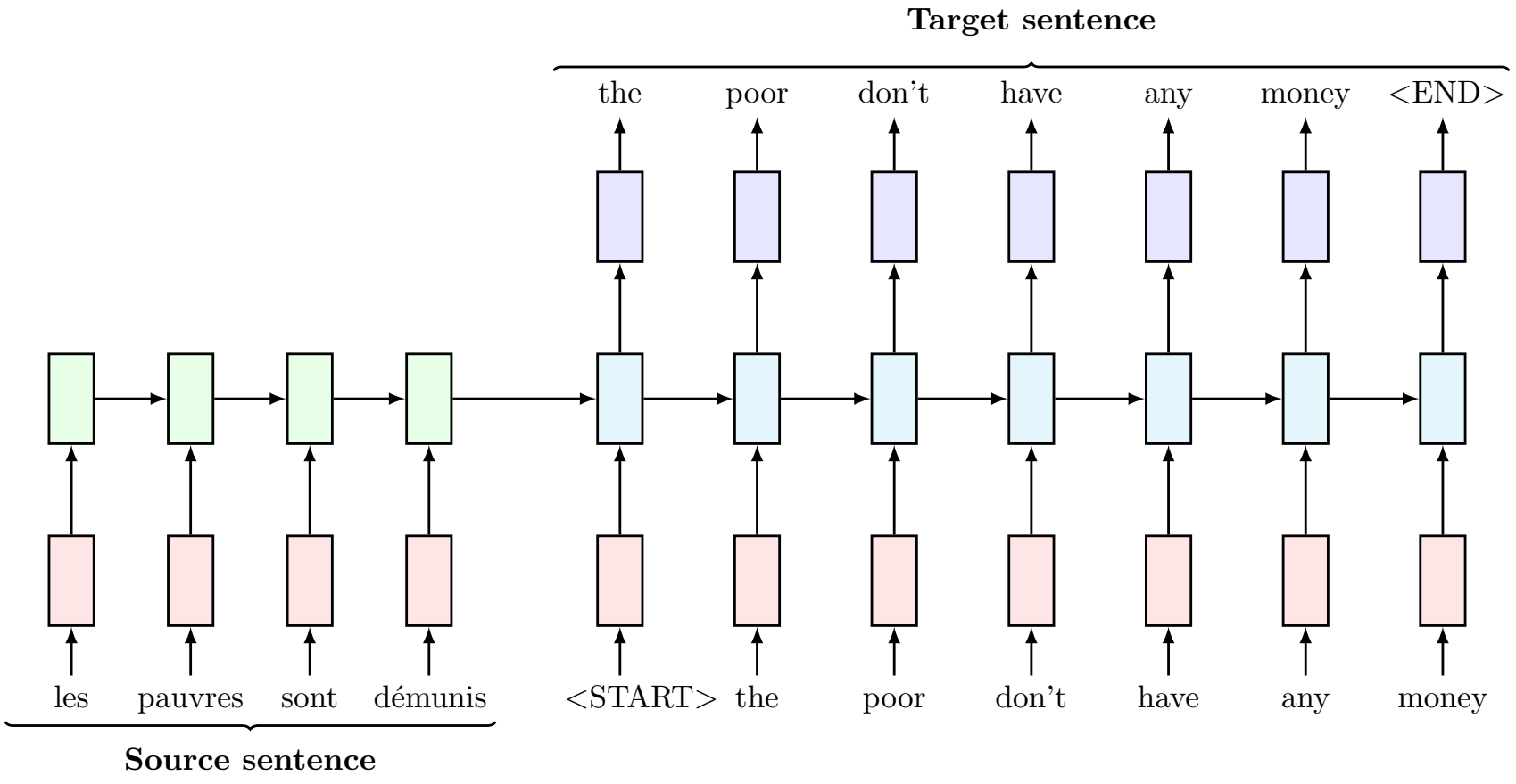
NMT research continues

- NMT is the flagship task for NLP Deep Learning.
- NMT research has pioneered many of the recent innovations of NLP Deep Learning
- In 2018: NMT research continues to thrive
  - Researchers have found many improvements to the *vanilla* seq2seq NMT
  - But one improvement is so integral that it is the new vanilla...

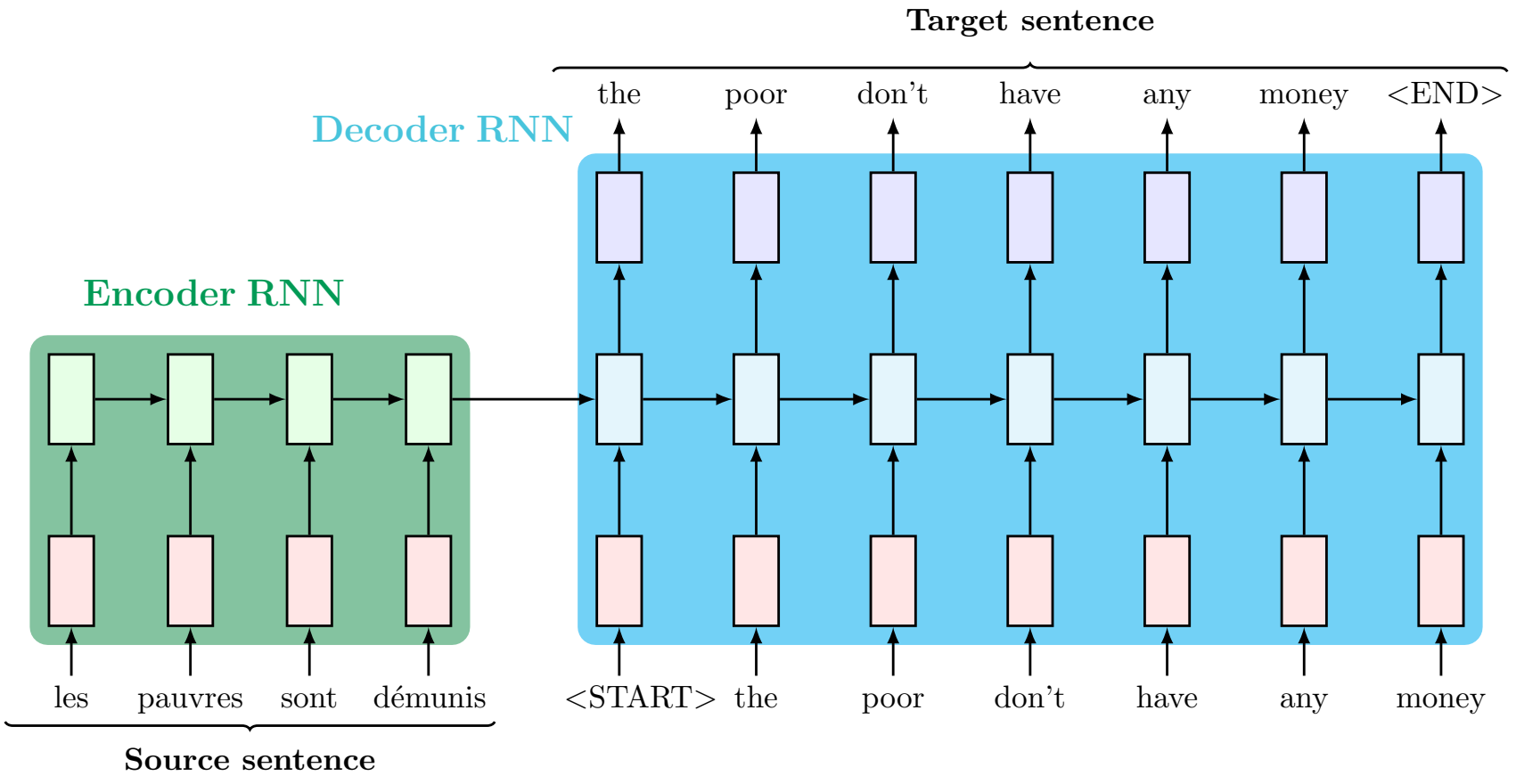
ATTENTION

Sequence-to-sequence: the bottleneck problem

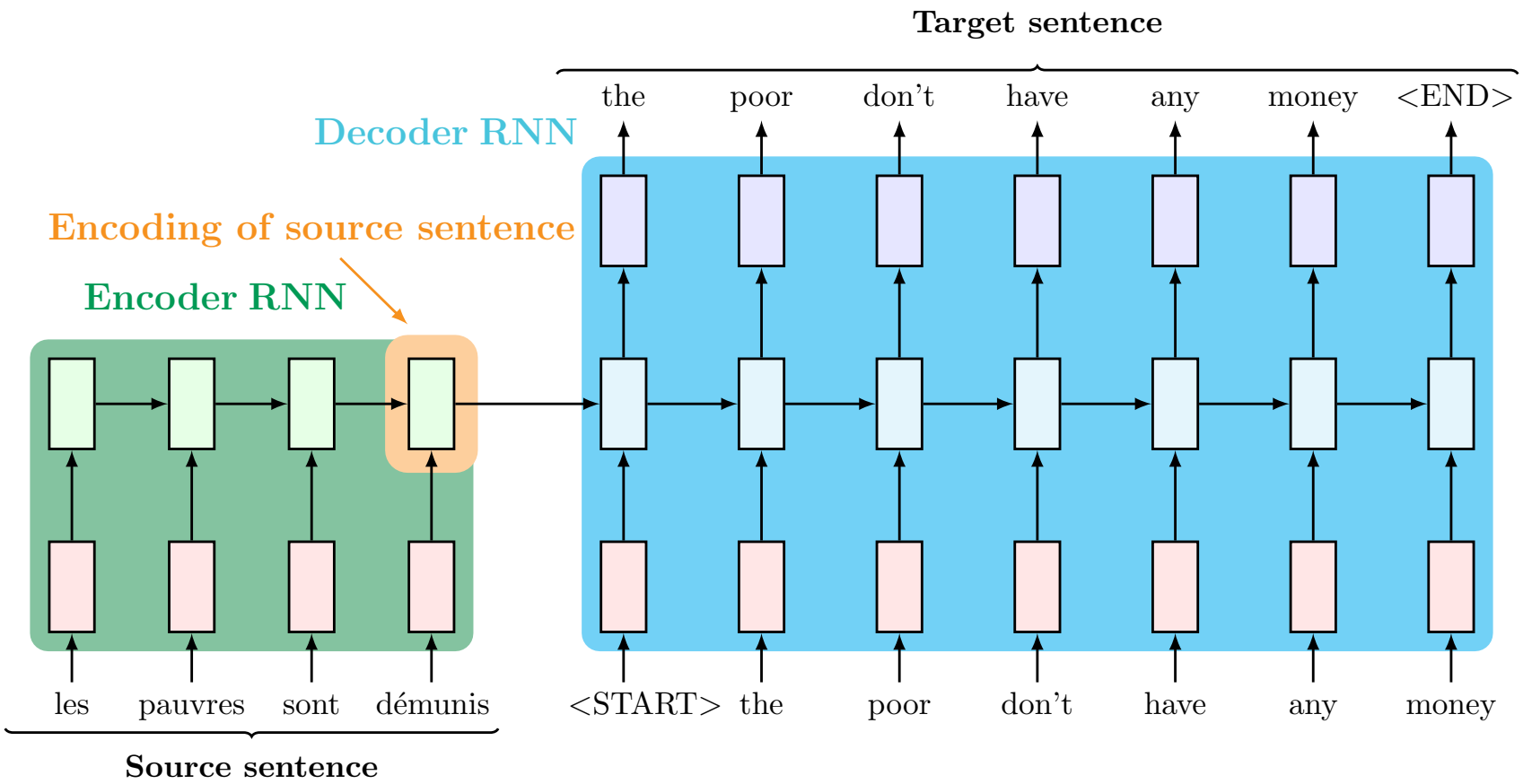
Vanilla sequence-to-sequence for machine translation



Vanilla sequence-to-sequence for machine translation

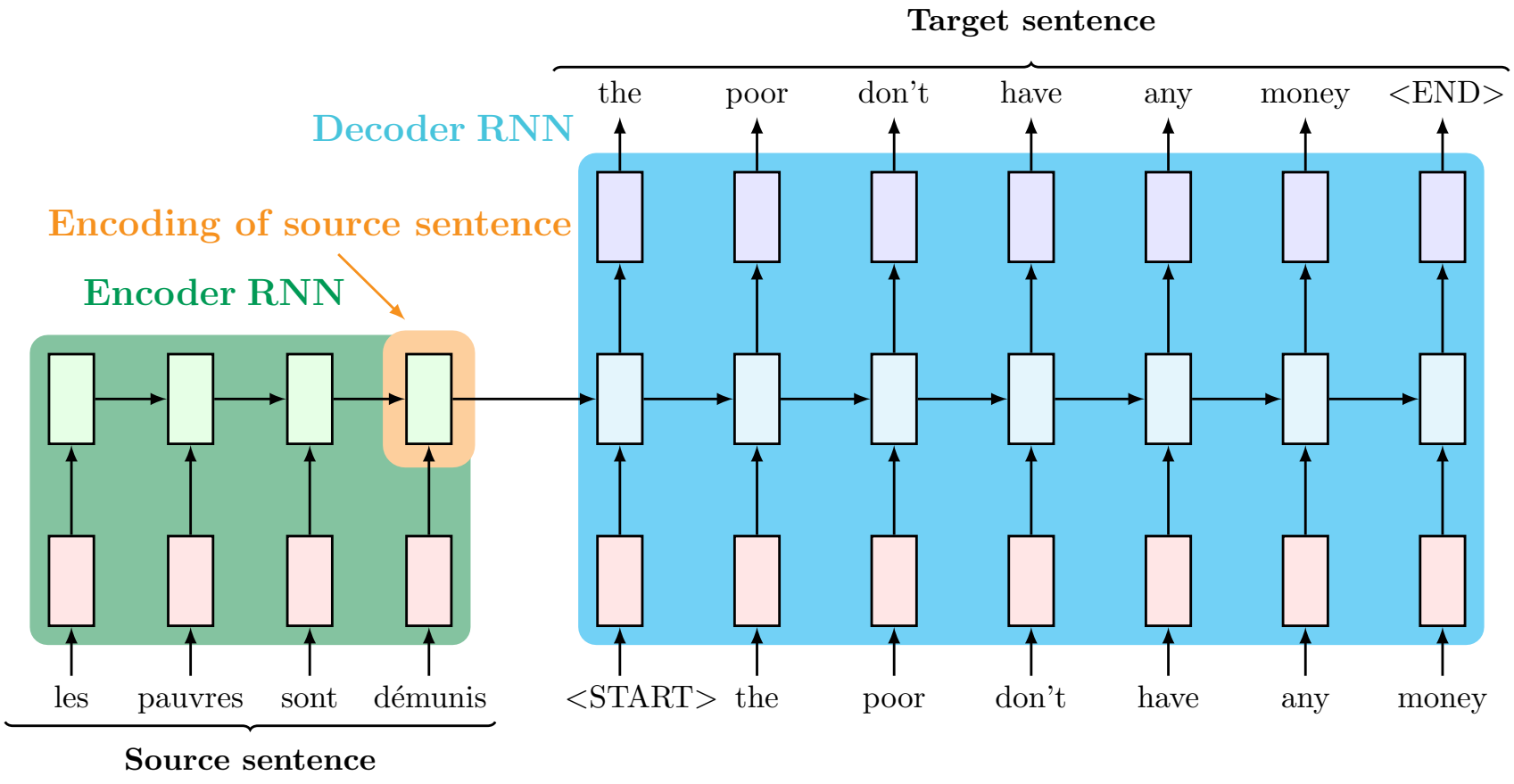


Sequence-to-sequence: the bottleneck problem



Problem with this architecture?

Sequence-to-sequence: the bottleneck problem



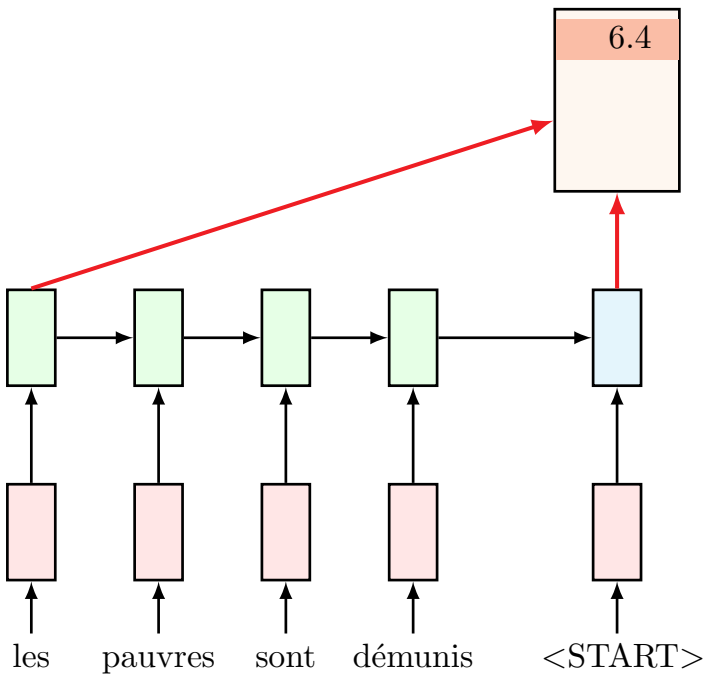
Encoding of the source sentence needs to capture **all information** about the source sentence. Information bottleneck!

Attention

- Attention provides a solution to the bottleneck problem.
- **Core idea** At each step of the decoder, focus on a particular part of the source sequence.
- Let's look at a pictorial description.

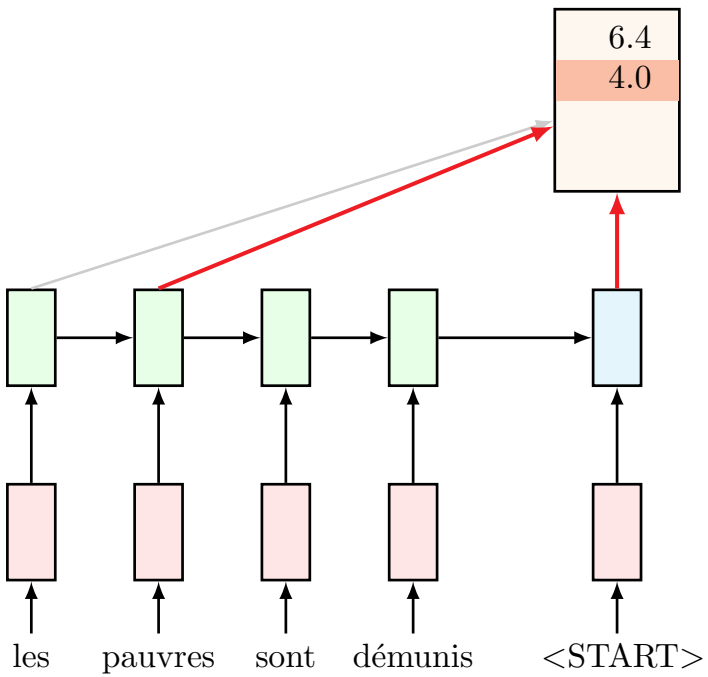
Sequence-to-sequence with attention

Compute *attention score* between decoder RNN's **current** hidden state and encoder RNN's **first** hidden state via dot-product.



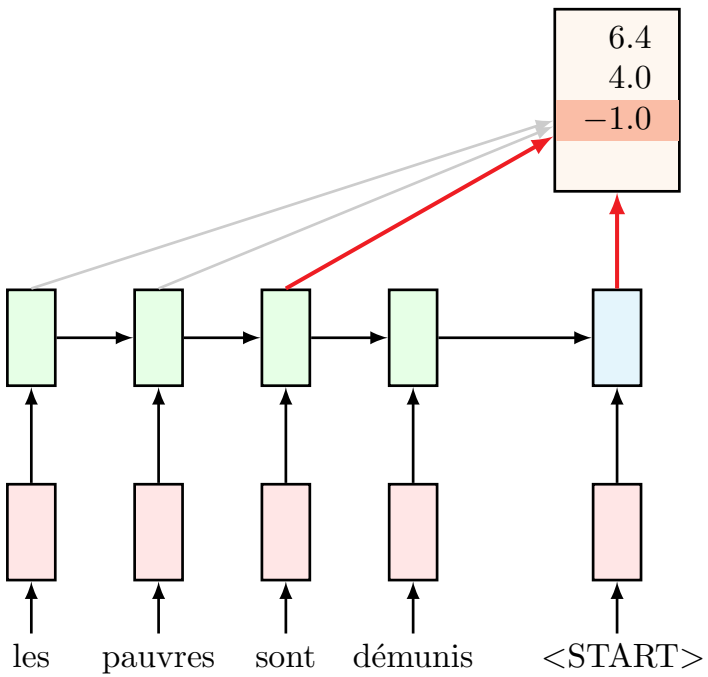
Sequence-to-sequence with attention

Compute *attention score* between decoder RNN's **current** hidden state and encoder RNN's **second** hidden state via dot-product.



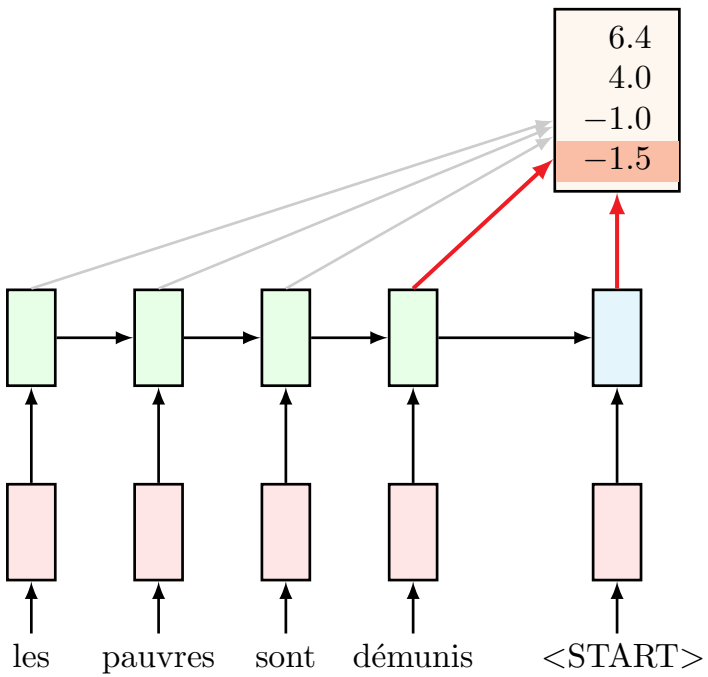
Sequence-to-sequence with attention

Compute *attention score* between decoder RNN's **current** hidden state and encoder RNN's **third** hidden state via dot-product.



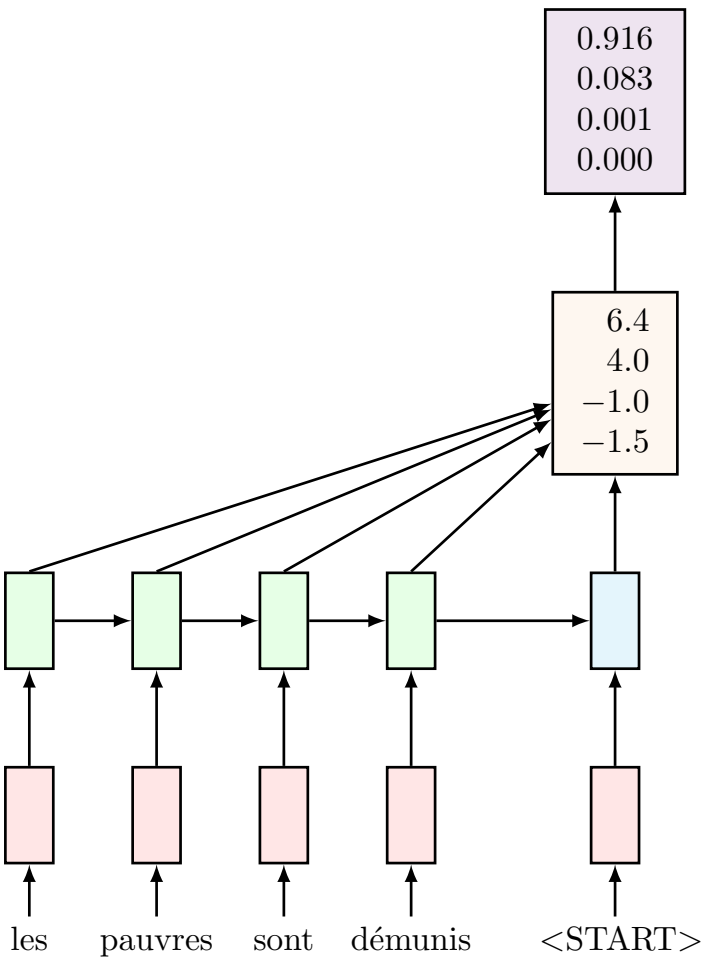
Sequence-to-sequence with attention

Compute *attention score* between decoder RNN's **current** hidden state and encoder RNN's **fourth** hidden state via dot-product.

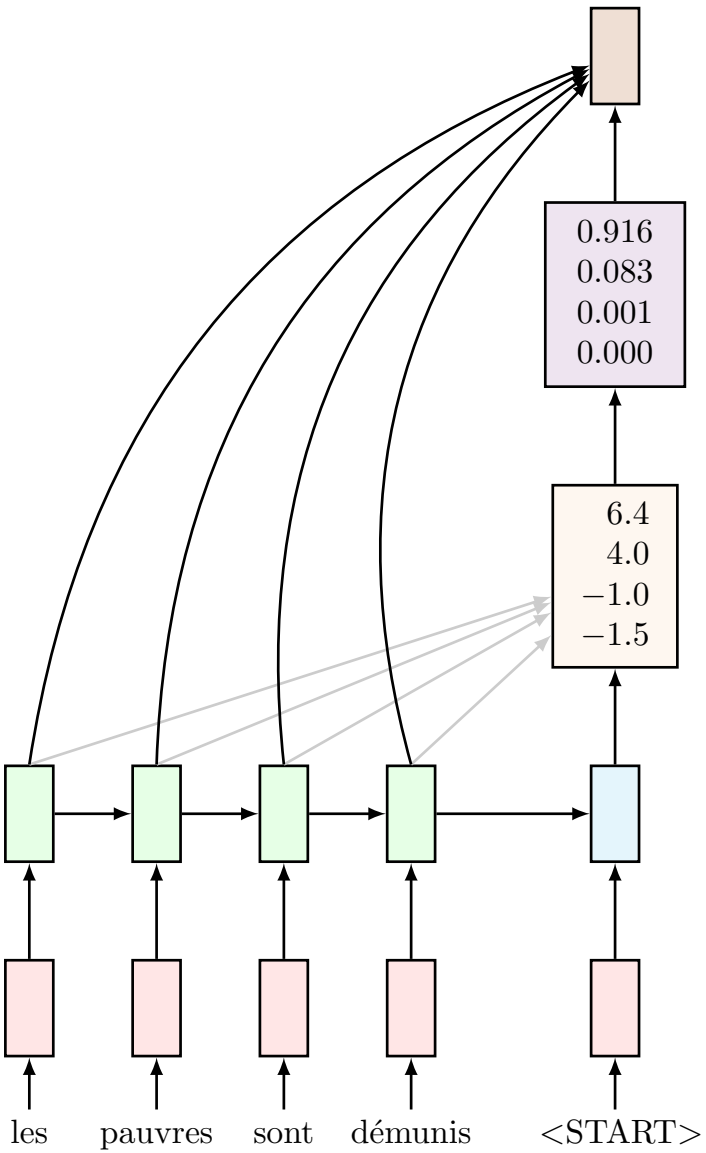


Sequence-to-sequence with attention

- Apply SoftMax to attention scores and get probability distribution.
- At this timestep, mostly focus on first encoder hidden state ("les")

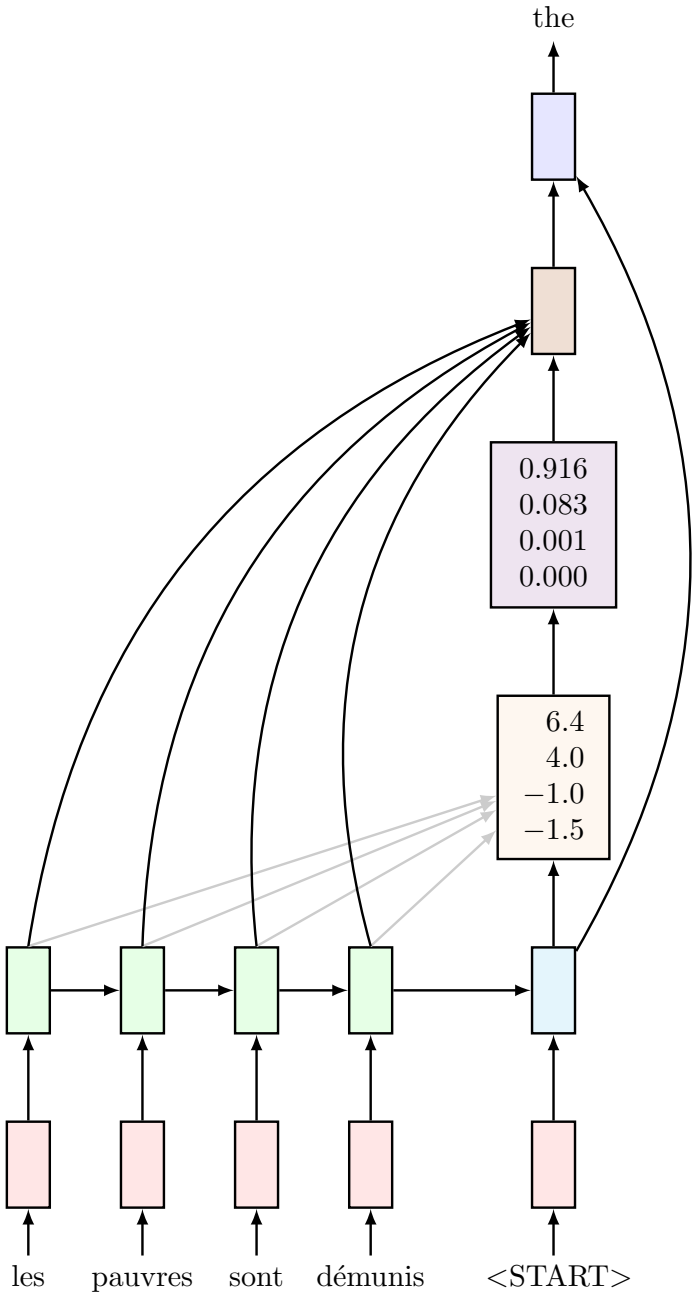


Sequence-to-sequence with attention



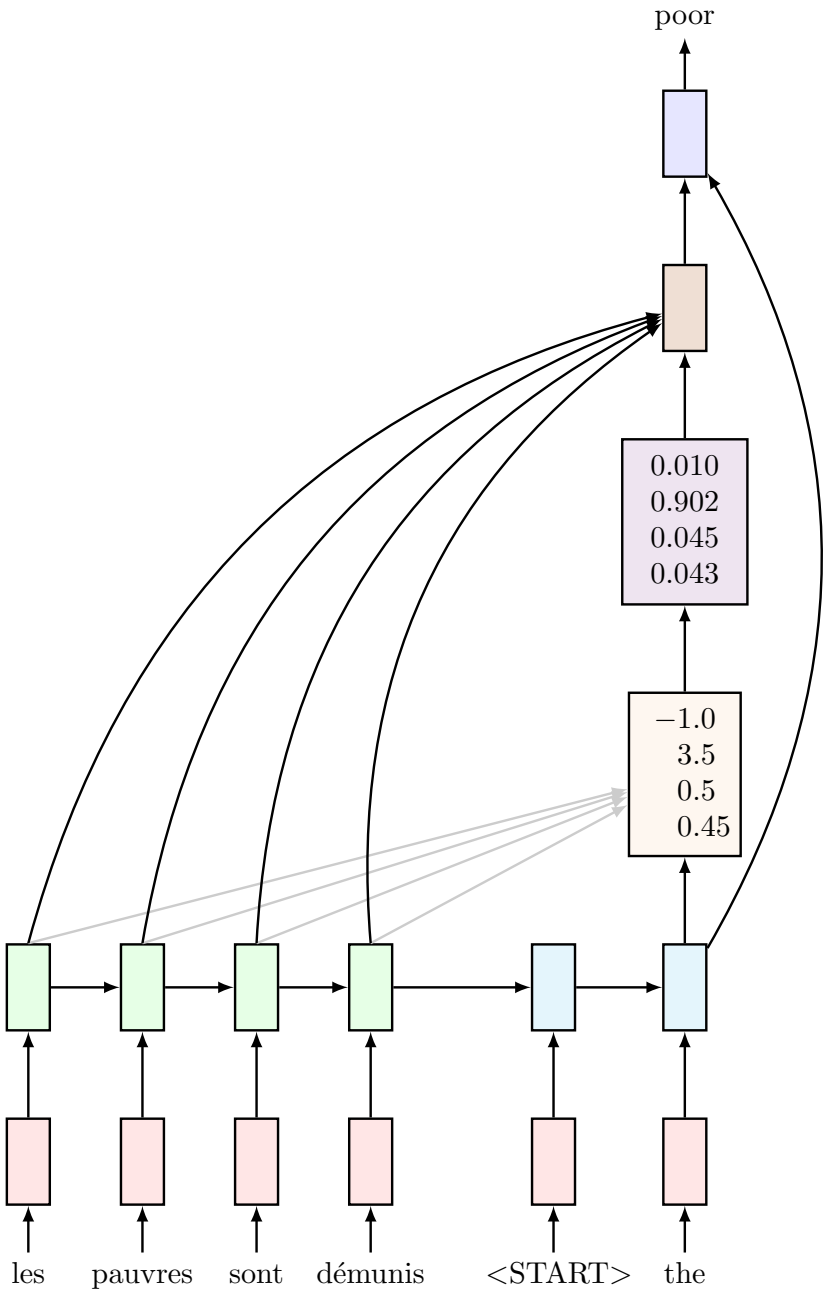
- Use the attention distribution to take a weighted sum of the encoder hidden states to get the context vector.
- Context vector mostly contains information from the encoder hidden states that received high attention probabilities.

Sequence-to-sequence with attention

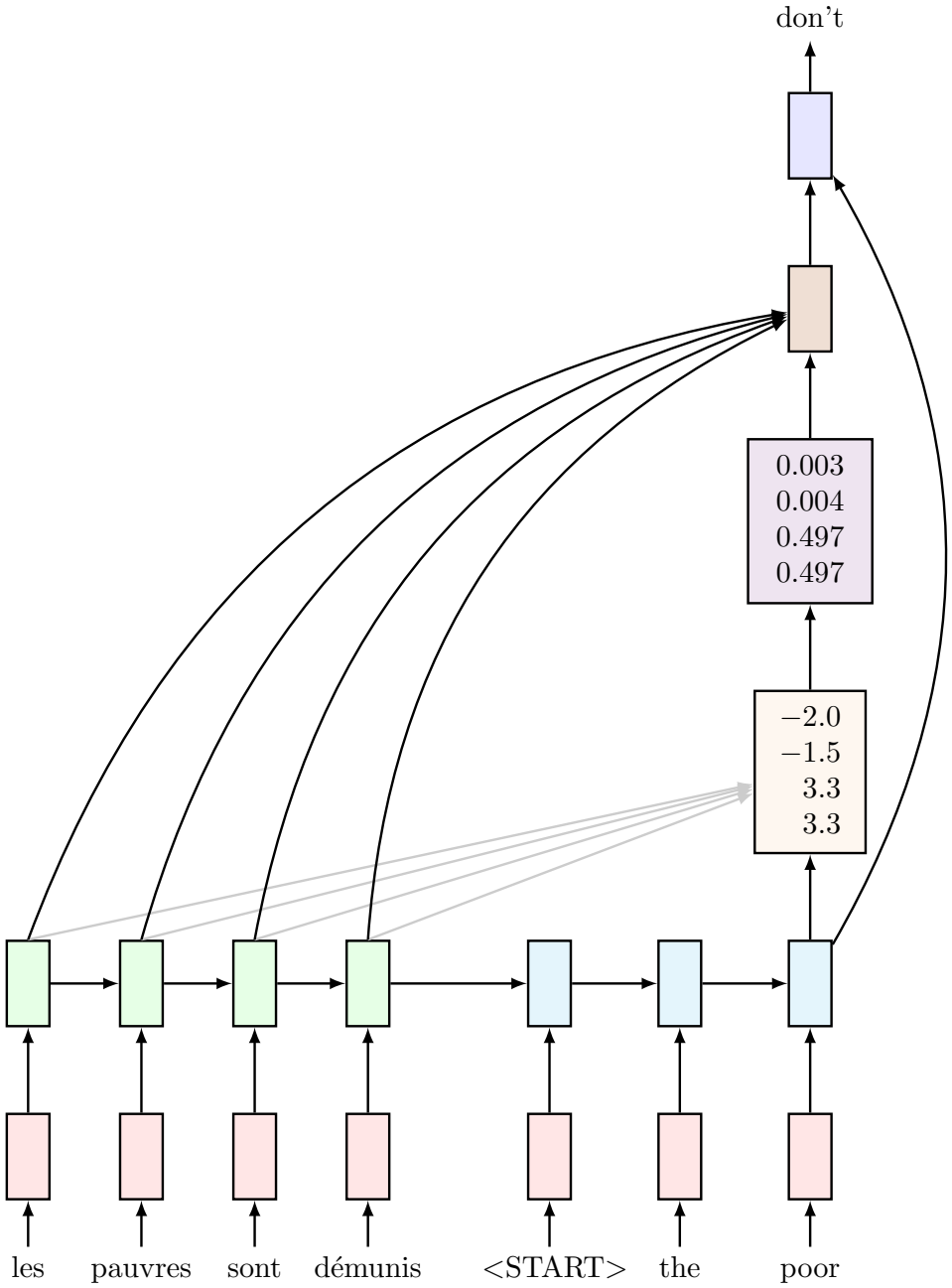


- Use context vector and decoder hidden state to predict the output scores (probabilities) for each word.
- Can sample the probabilities to predict the next word.

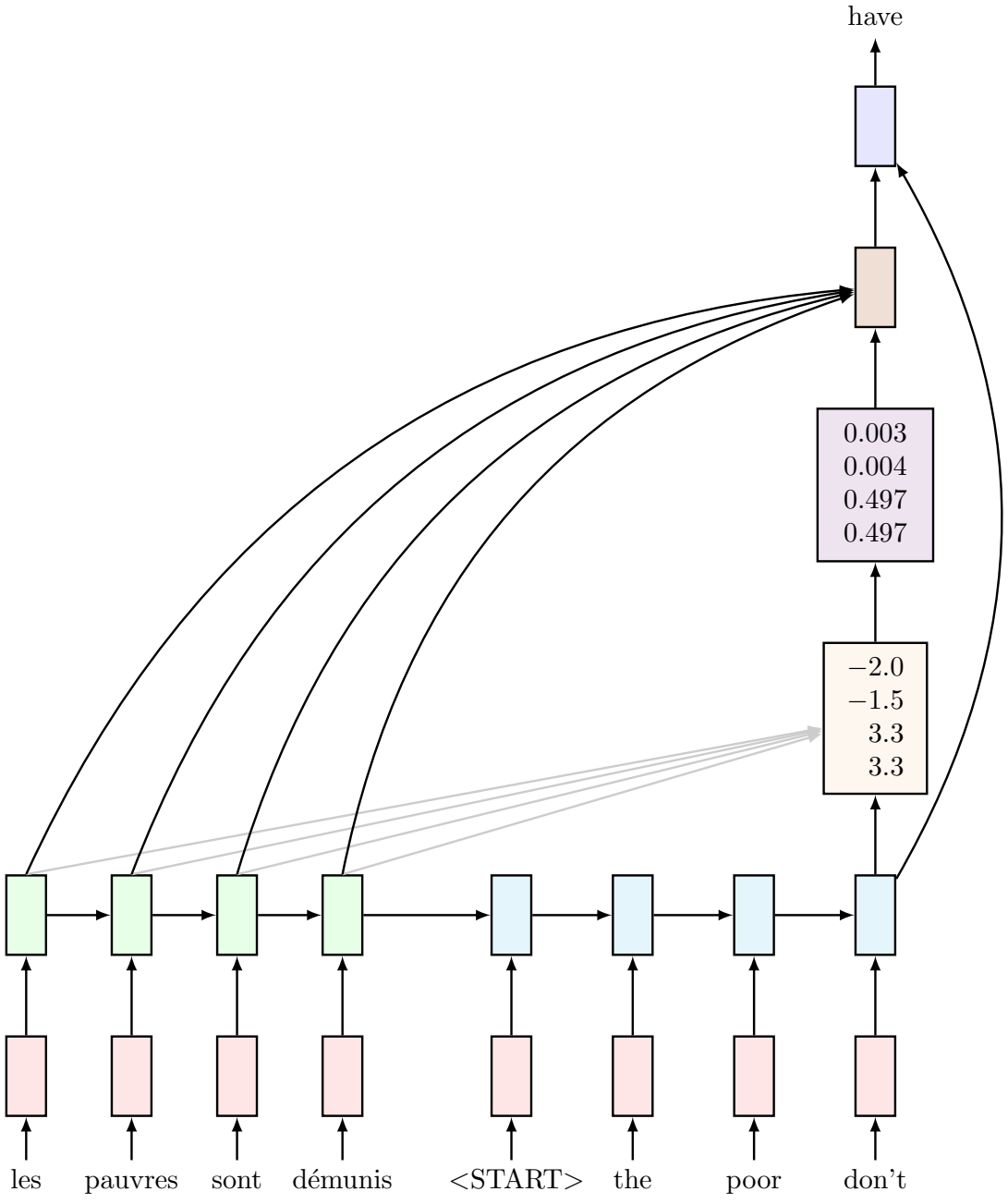
Sequence-to-sequence with attention



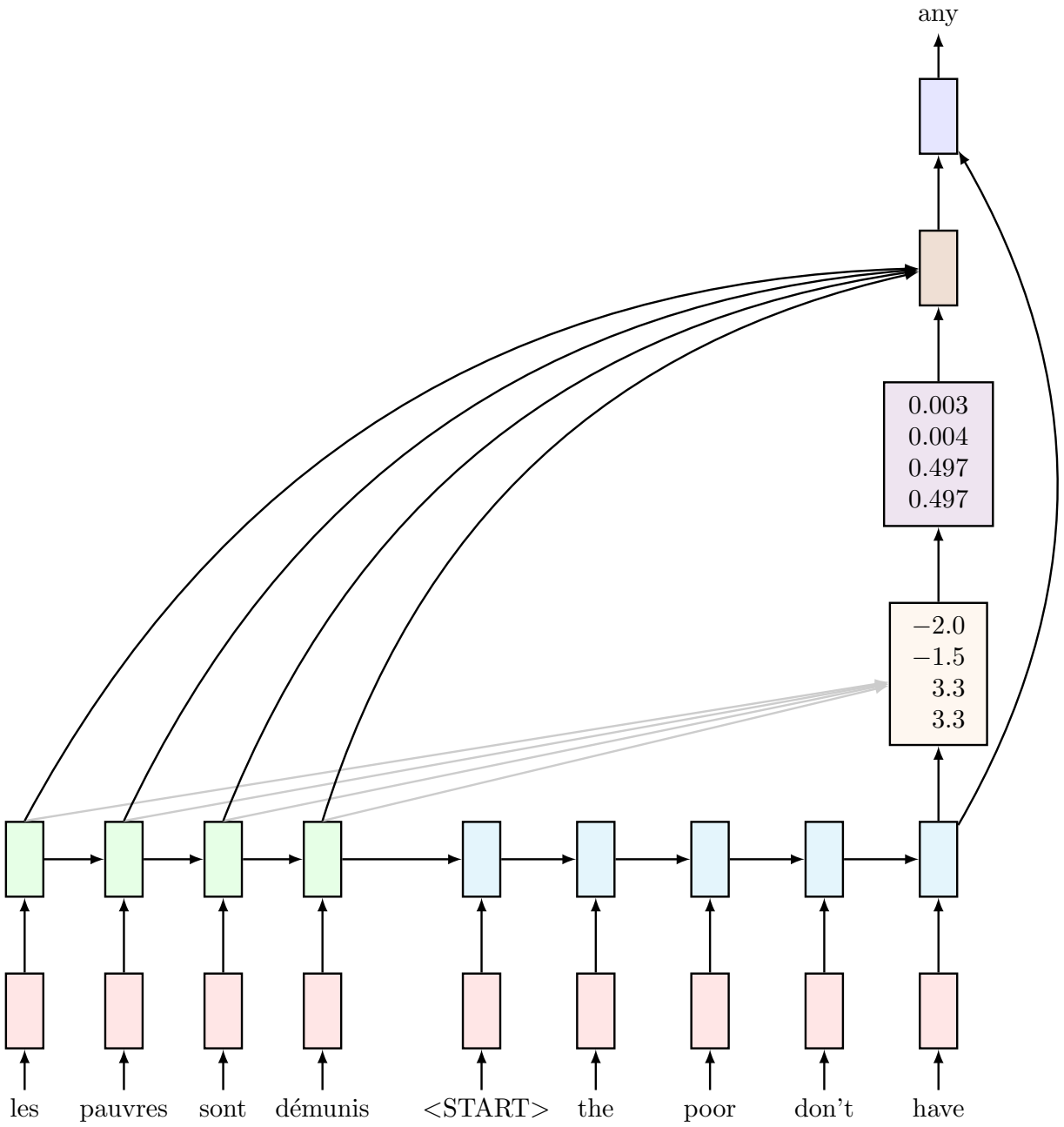
Sequence-to-sequence with attention



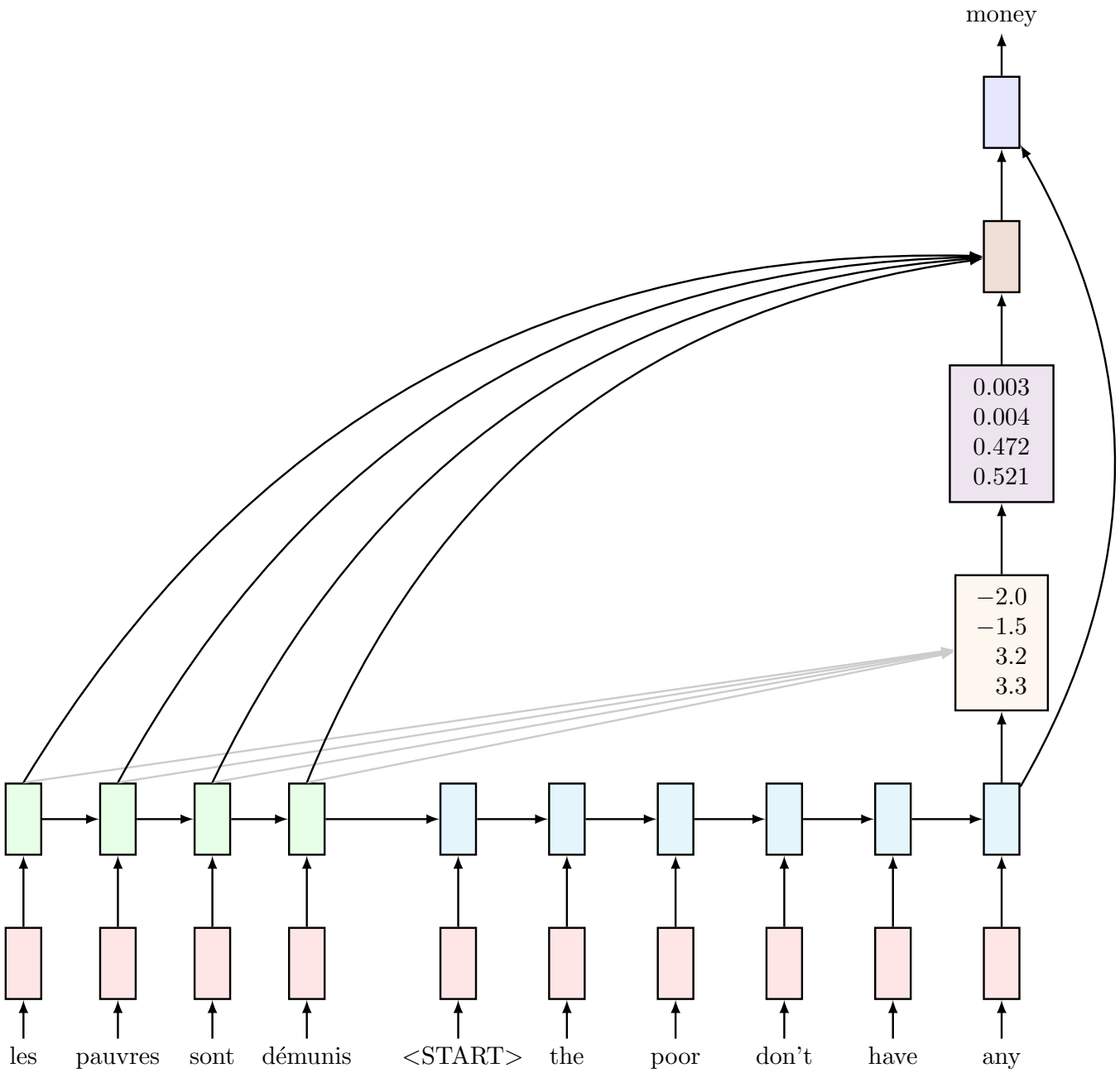
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention Variant 1: in equations

- Have encoder hidden states: $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ with each $\mathbf{h}_i \in \mathbb{R}^m$.
- At time-step t have decoder hidden state: $\mathbf{s}_t \in \mathbb{R}^m$
- Get attention scores: $\mathbf{e}_t = (e_{t,1}, \ldots, e_{t,N})^T$ where

$$e_{t,j} = \mathbf{s}_t^T \mathbf{h}_j$$

- Apply SoftMax to $\mathbf{e}_t$ and get attention probabilities:

$$\boldsymbol{\alpha}_t = \text{SoftMax}(\mathbf{e}_t)$$

- Create the context vector:

$$\mathbf{c}_t = \sum_{j=1}^{N} \alpha_{t,j} \, \mathbf{h}_j$$

- Then calculate the output scores with:

$$\mathbf{o}_t = V_s \mathbf{s}_t + V_c \mathbf{c}_t$$

Attention is GREAT!

- Attention significantly improves NMT performance
  - Very helpful to allow the decoder focus on certain parts of source.
- Attention solves the bottleneck problem.
  - Attention allows decoder to look directly at source.
  - Can by-pass the bottleneck.
- Attention helps with vanishing gradient problem.
  - Provides shortcut to faraway states.

Attention is GREAT!

- Attention provides some interpretability.

	les	pauvres	sont	démunis
The	0.916	0.083	0.001	0.000
poor	0.010	0.902	0.045	0.043
don't	0.003	0.004	0.497	0.497
have	0.003	0.004	0.497	0.497
any	0.003	0.004	0.497	0.497
money	0.003	0.004	0.472	0.521

  - By inspecting attention distribution, can see what the decoder was focusing on.
  - Get alignment for free.
  - Fantastic because we did not explicitly train for alignment.
  - The network just learned alignment by itself.

Attention more sophisticated variant: in equations

- Have encoder hidden states: $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ with each $\mathbf{h}_i \in \mathbb{R}^{m_1}$.
- At time-step t have decoder hidden state: $\mathbf{s}_t \in \mathbb{R}^{m_2}$
- Get attention scores: $\mathbf{e}_{t+1} = (e_{t+1,1}, \ldots, e_{t+1,N})^T$ where

$$e_{t+1,j} = f_{\text{att}}(\mathbf{s}_t, \mathbf{h}_j)$$

- Apply SoftMax to $\mathbf{e}_{t+1}$ and get attention probabilities:

$$\boldsymbol{\alpha}_{t+1} = \text{SoftMax}(\mathbf{e}_{t+1})$$

- Create the context vector:

$$\mathbf{c}_{t+1} = \sum_{j=1}^{N} \alpha_{t+1,j}\, \mathbf{h}_j$$

- Then calculate the next decoder hidden state:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{c}_{t+1}, \mathbf{y}_{t+1})$$

There are several common ways f_{att} in

$$e_{t+1,j} = f_{\text{att}}(\mathbf{s}_t, \mathbf{h}_j)$$

is defined:

- **Basic dot-product attention** (when $m_1 = m_2$)

$$e_{t+1,j} = \mathbf{s}_t^T \mathbf{h}_j$$

- **Multiplicative attention**

$$e_{t+1,j} = \mathbf{s}_t^T W \mathbf{h}_j$$

where $W \in \mathbb{R}^{m_2 \times m_1}$

- **Additive attention**

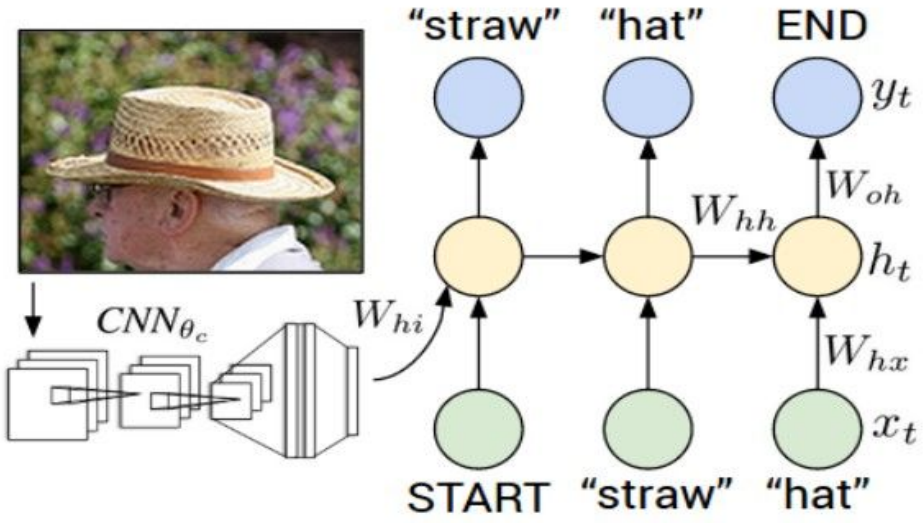
$$e_{t+1,j} = \mathbf{v}^T \tanh(W_1 \mathbf{s}_t + W_2 \mathbf{h}_j)$$

Attention is a general Deep Learning technique

- Have a set of vectors $\mathbf{h}_1, \ldots, \mathbf{h}_N$, called the values and a query vector $\mathbf{s}$.
- Attention is a technique to compute a weighted sum of the $\mathbf{h}_j$'s dependent on the query $\mathbf{s}$.
- Say the query attends to the values.
- 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).

Image Captioning with Attention

Before: Image Captioning with ConvNets + RNNs



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Image Captioning with Attention

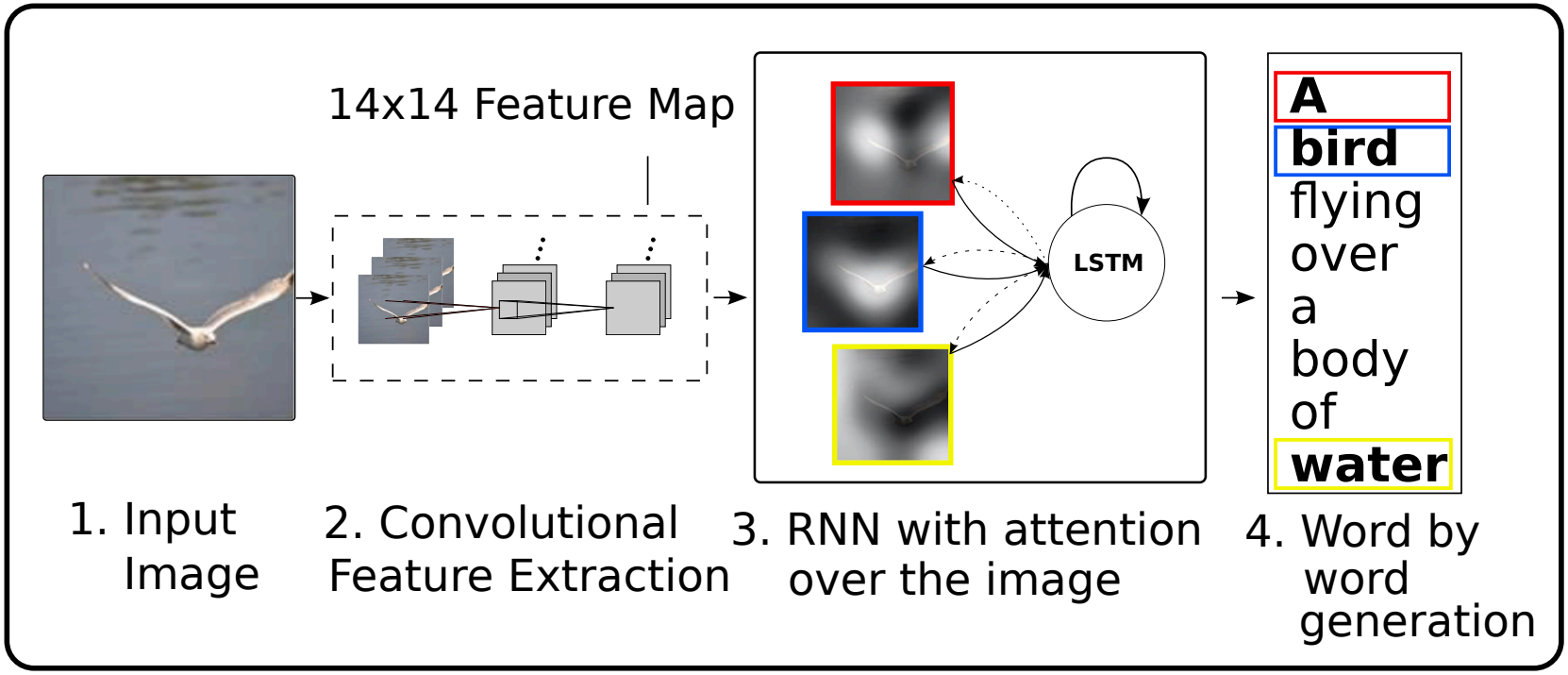


Image Feature Extraction

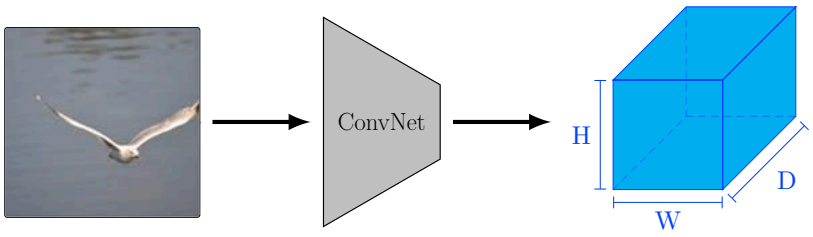
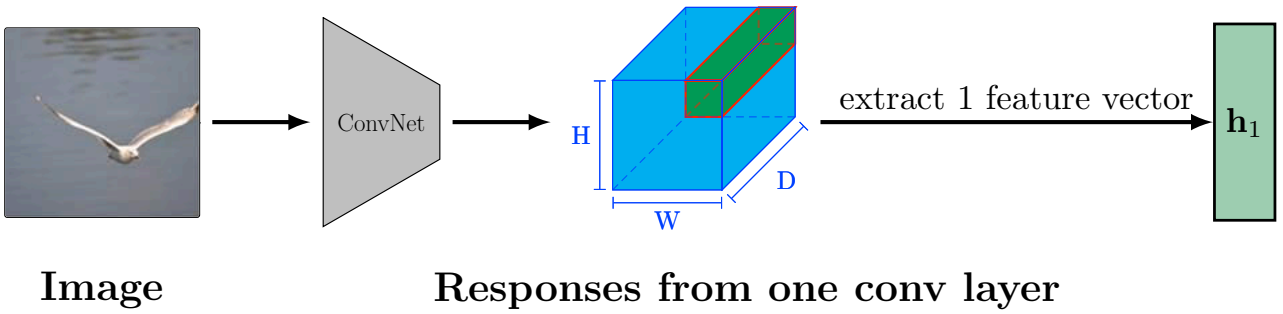


Image **Responses from one conv layer**

Process to extract the **value** vectors $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ from image.

- Apply ConvNet. Keep responses from one convolutional layer.

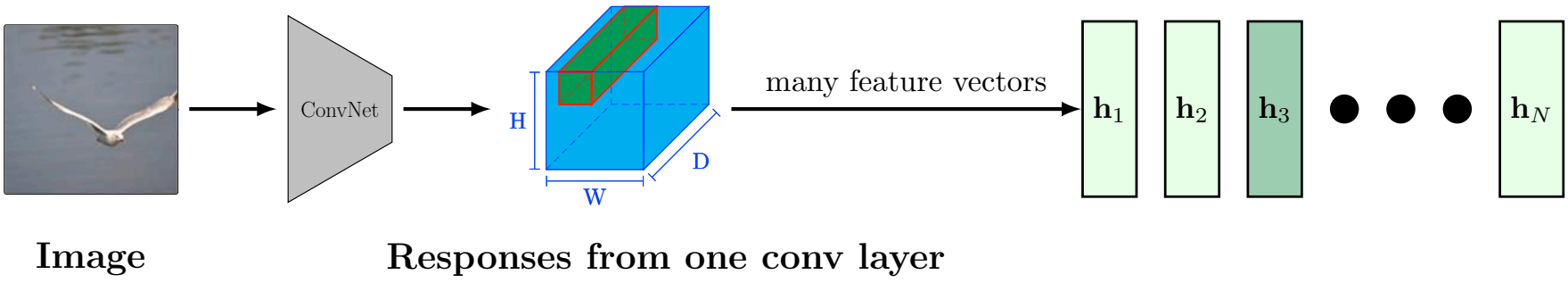
Image Feature Extraction



Process to extract the **value** vectors $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ from image.

- Apply ConvNet. Keep responses from one convolutional layer.
- Extract a sub-volume of size $w \times h \times D$ and vectorize $\implies$ have feature description of a sub-region of input image.

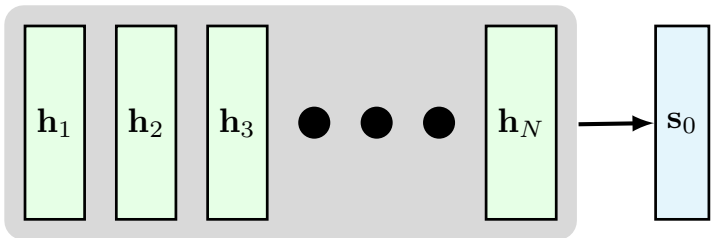
Image Feature Extraction



Process to extract the **value** vectors $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ from image.

- Apply ConvNet. Keep responses from one convolutional layer.
- Extract a sub-volume of size $w \times h \times D$ and vectorize $\implies$ have feature description of a sub-region of input image.
- Extract different sub-volumes of size $w \times h \times D$ and vectorize each one $\implies$ have feature description of different sub-regions of input image.

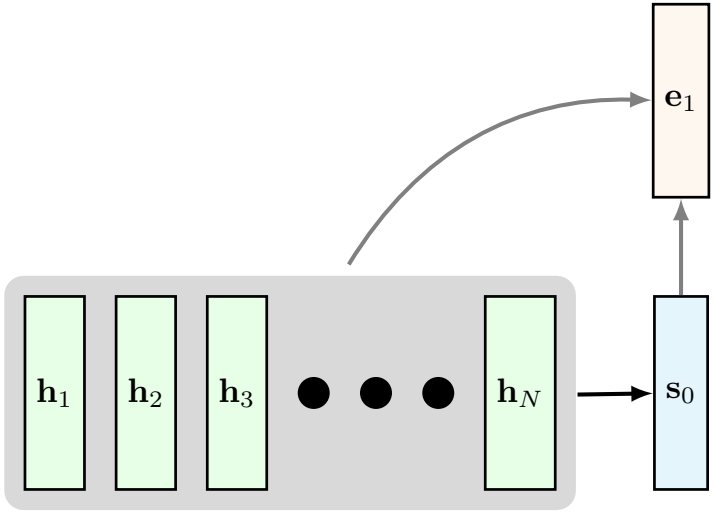
Image Captioning with Attention



- Generate the initial hidden state of the decoder RNN with

$$\mathbf{s}_0 = f_{\text{init}}\left(\frac{1}{N}\sum_{j=1}^{N}\mathbf{h}_j\right)$$

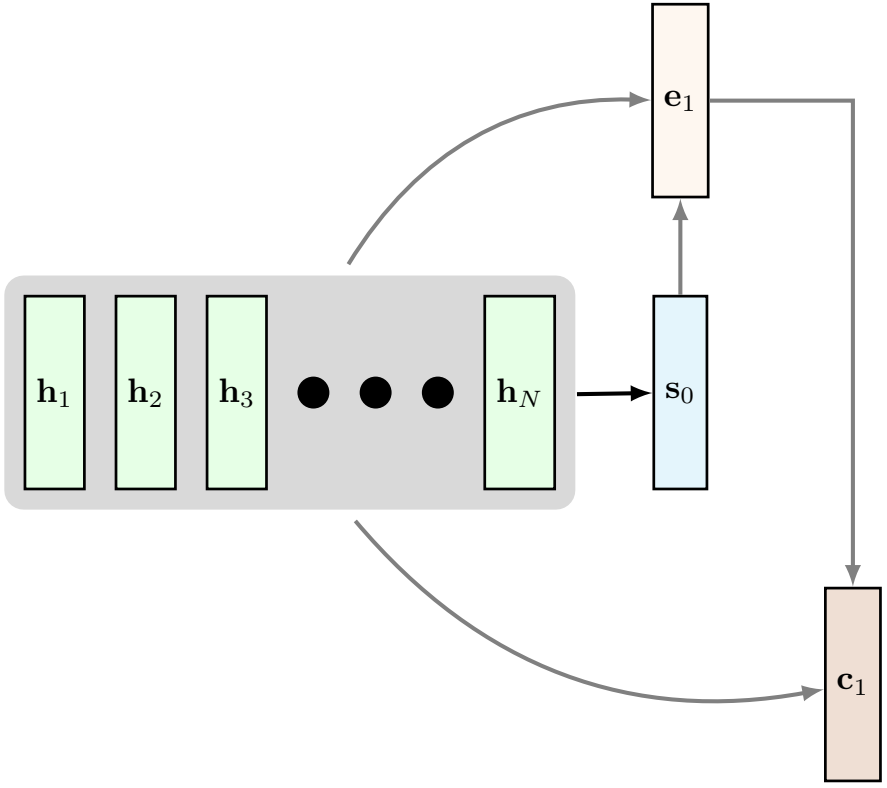
Image Captioning with Attention



- Compute the attention scores for the initial hidden decoder state

$$\mathbf{e}_{1,j} = f_{\text{att}}\left(\mathbf{s}_0, \mathbf{h}_j\right)$$

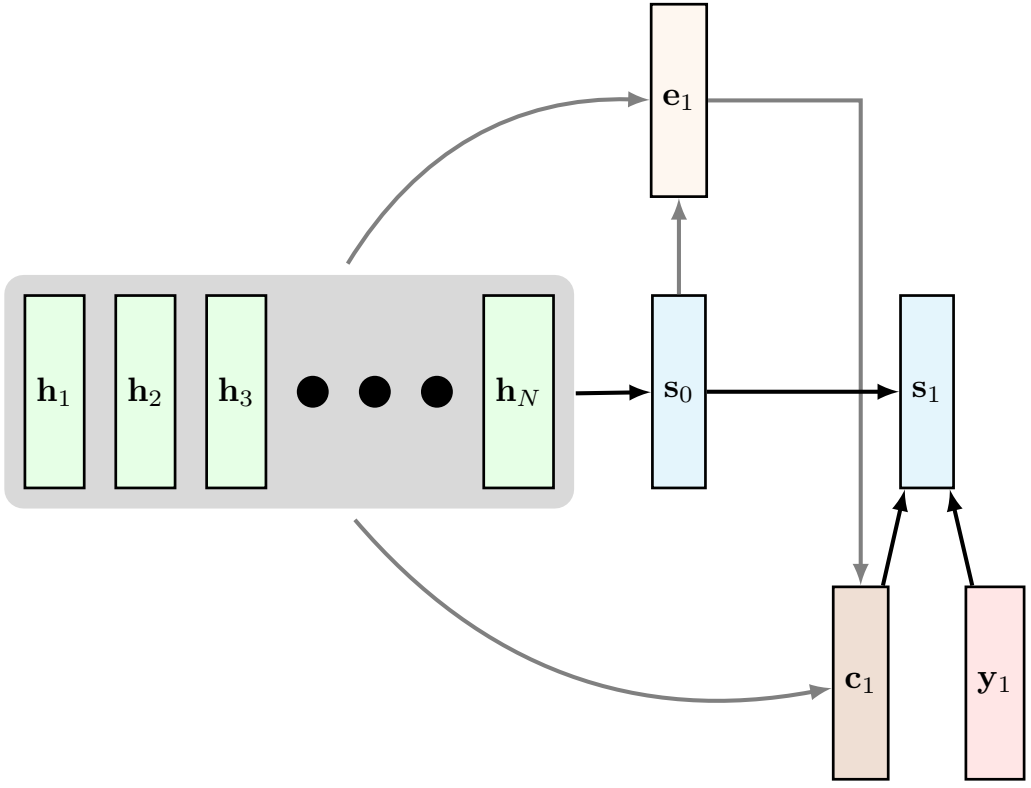
Image Captioning with Attention



- Apply SoftMax to $\mathbf{e}_1$ to get the attention distribution $\boldsymbol{\alpha}_1$.
- Compute the first context vector:

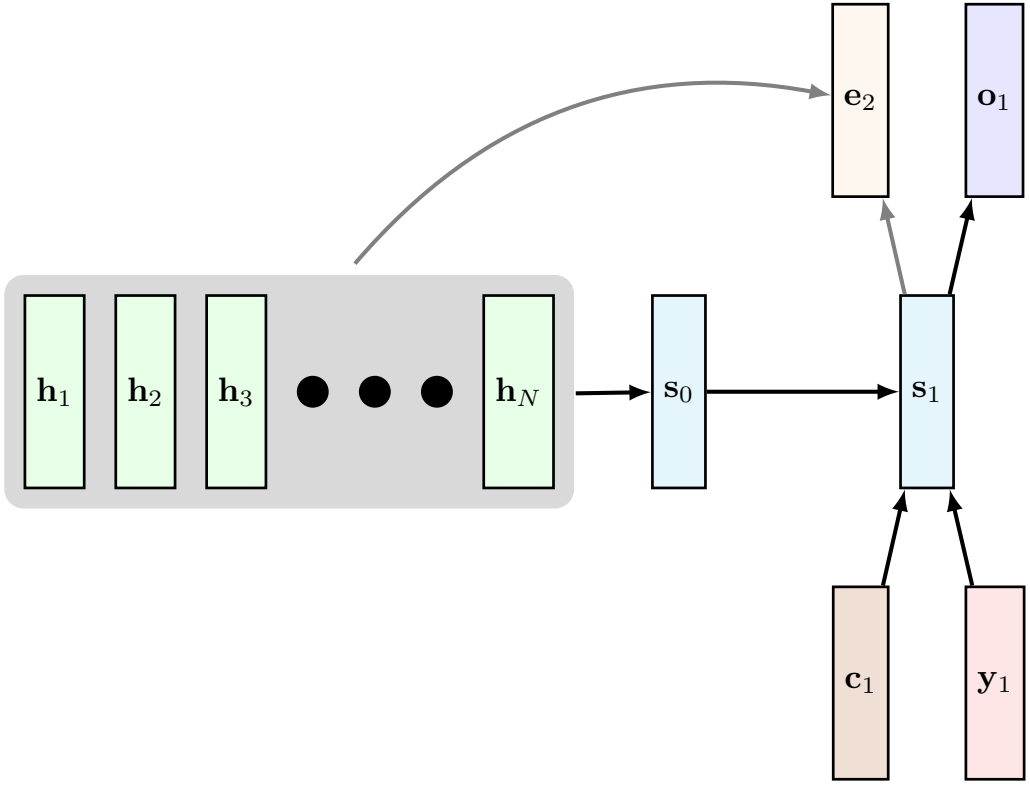
$$\mathbf{c}_1 = \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{\alpha}_{1,j} \mathbf{h}_j$$

Image Captioning with Attention



- Assume the first input vector $\mathbf{y}_1$ is the <START> token.
- Compute hidden state $\mathbf{s}_1$ from $\mathbf{s}_0$, $\mathbf{c}_1$ and $\mathbf{y}_1$.

Image Captioning with Attention

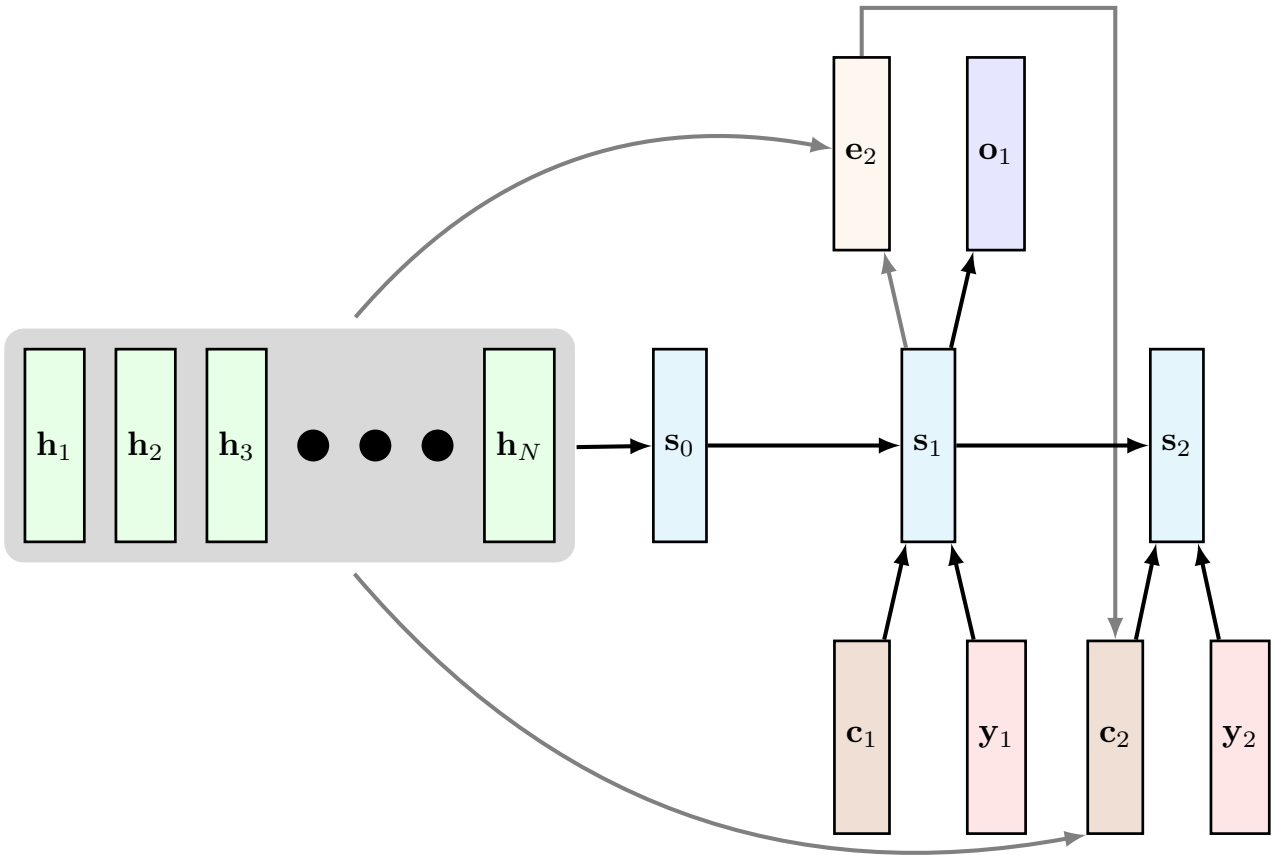


- Compute the attention scores for $\mathbf{s}_1$

$$\mathbf{e}_{2,j} = f_{\text{att}}\left(\mathbf{s}_1, \mathbf{h}_j\right)$$

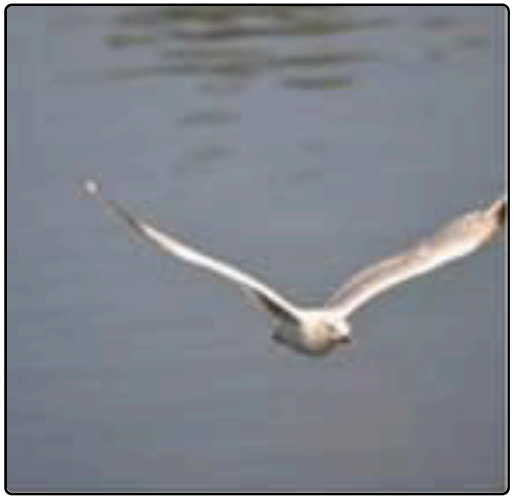
- Compute the output scores (for each word in the vocab) $\mathbf{o}_1$ from $\mathbf{s}_1$.

Image Captioning with Attention



- Apply SoftMax to $\mathbf{e}_2$ to get the attention distribution $\boldsymbol{\alpha}_2$.
- Compute the context vector $\mathbf{c}_2$ from $\boldsymbol{\alpha}_2$ and the $\mathbf{h}_j$'s.
- **During testing**: Sample based on $\mathbf{o}_1$ to generate the second word $\mathbf{y}_2$
- Compute hidden state $\mathbf{s}_2$ from $\mathbf{s}_1$, $\mathbf{c}_2$ and $\mathbf{y}_2$

Image captioning with attention - Some Results



A bird is flying over a body of water.

Attention over time

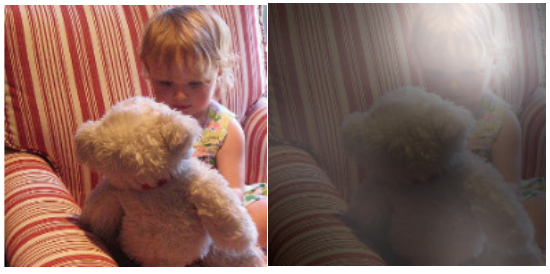
- White indicates the attended region.

Image captioning with attention - Some Results

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

- White indicates the attended region.
- The underline indicates the corresponding word.

Image captioning with attention - Some Results

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

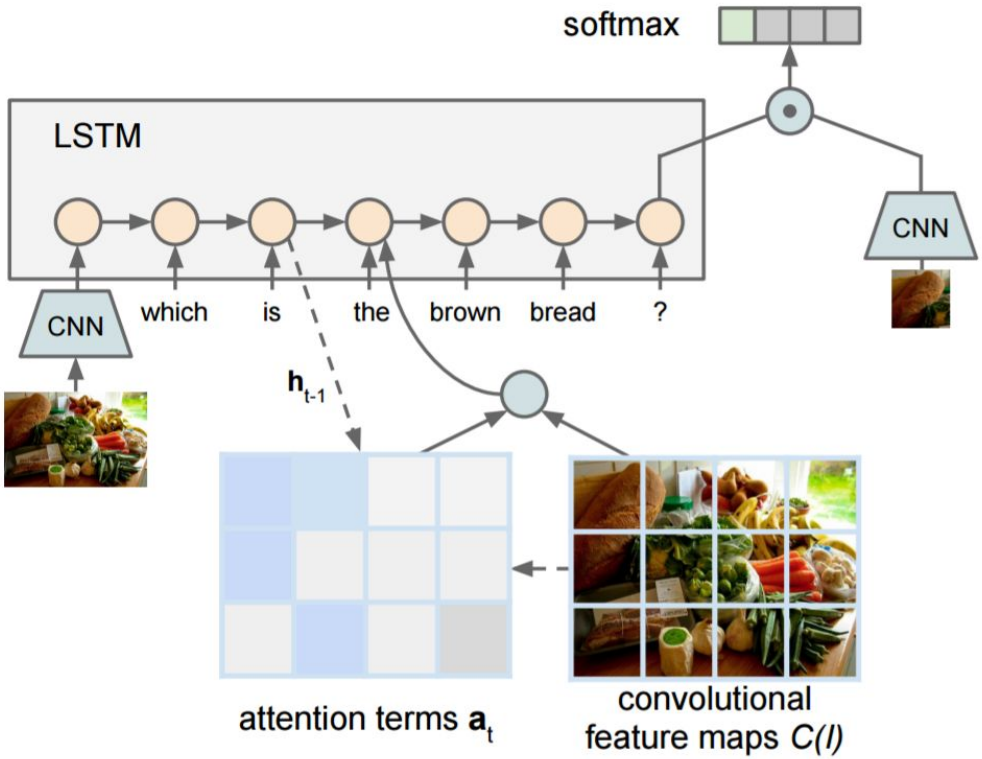
A man is talking on his cell phone while another man watches.

- White indicates the attended region.
- The underline indicates the corresponding word.

Attention can help illuminate why the captioning process went wrong.

Visual Question Answering with Attention

Visual Question Answering; RNNs with Attention



Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

What kind of animal is in the photo?

A **cat**.

Why is the person holding a knife?

To cut the **cake** with.

Sequence-to-sequence is versatile!

- Sequence-to-sequence is useful for more than just Machine Translation.
- Many NLP tasks can be phrased as sequence-to-sequence:
  - **Summarization** (long text $\rightarrow$ short text)
  - **Dialogue** (previous utterance $\rightarrow$ next utterance)
  - **Parsing** (input text $\rightarrow$ output parse as sequence)
  - **Code generation** (natural language $\rightarrow$ Python code)