COS 484: Natural Language Processing

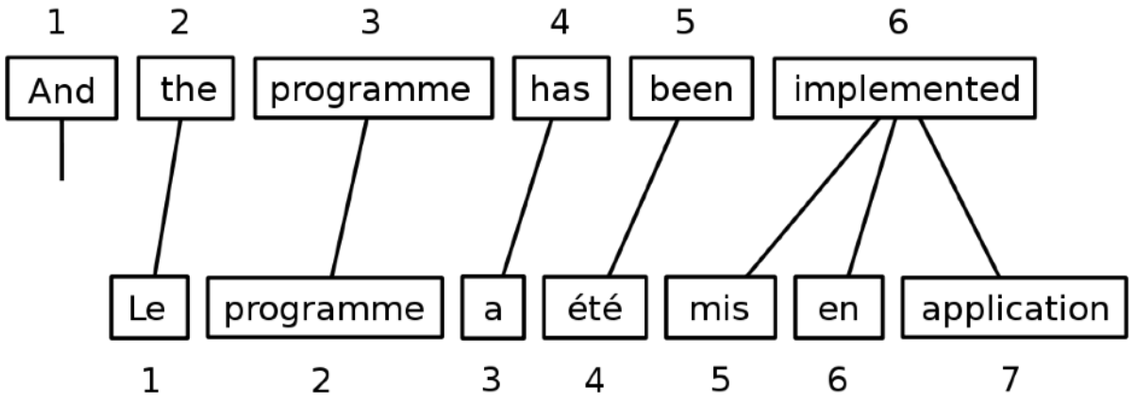
Neural Machine Translation

Fall 2019

Announcements

- Sign up for project meetings ASAP
  - Link will be posted on Piazza by 3pm today
  - Will be held this week

Last time



- Statistical MT
  - Word-based
  - Phrase-based
  - Syntactic

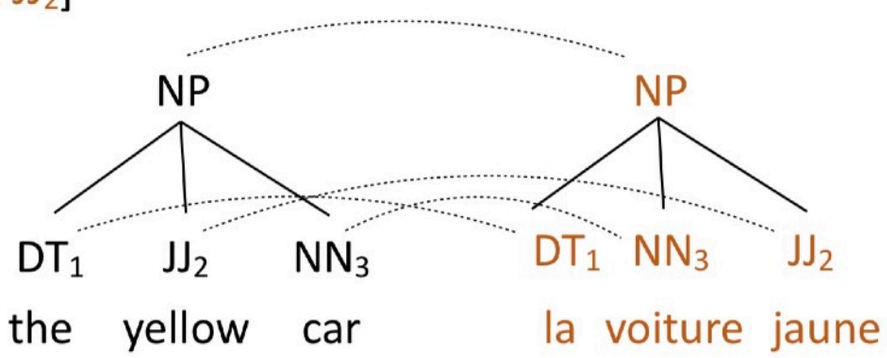
NP → [DT_1 JJ_2 NN_3; DT_1 NN_3 JJ_2]

DT → [the, la]

DT → [the, le]

NN → [car, voiture]

JJ → [yellow, jaune]

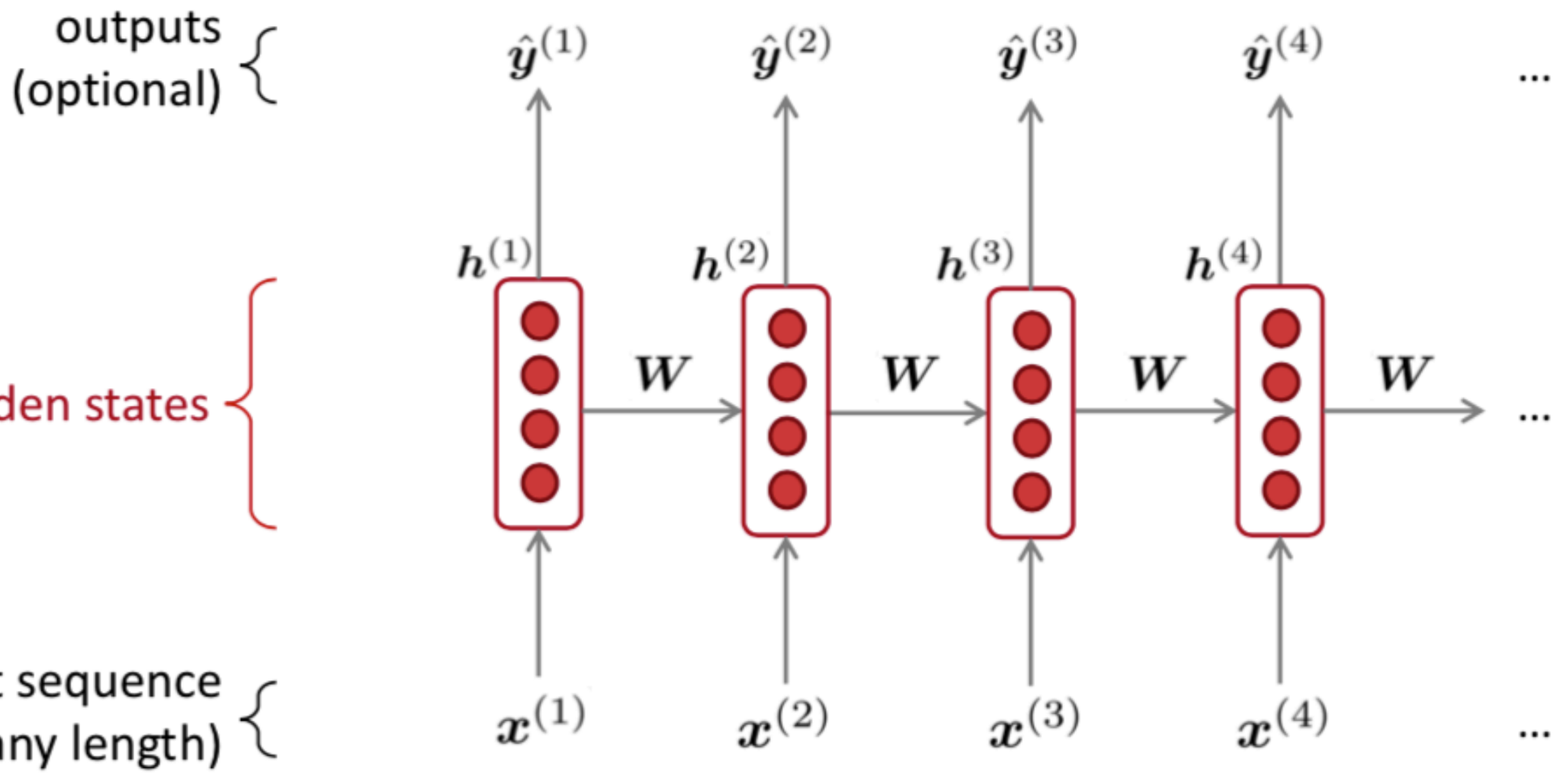


Neural Machine Translation

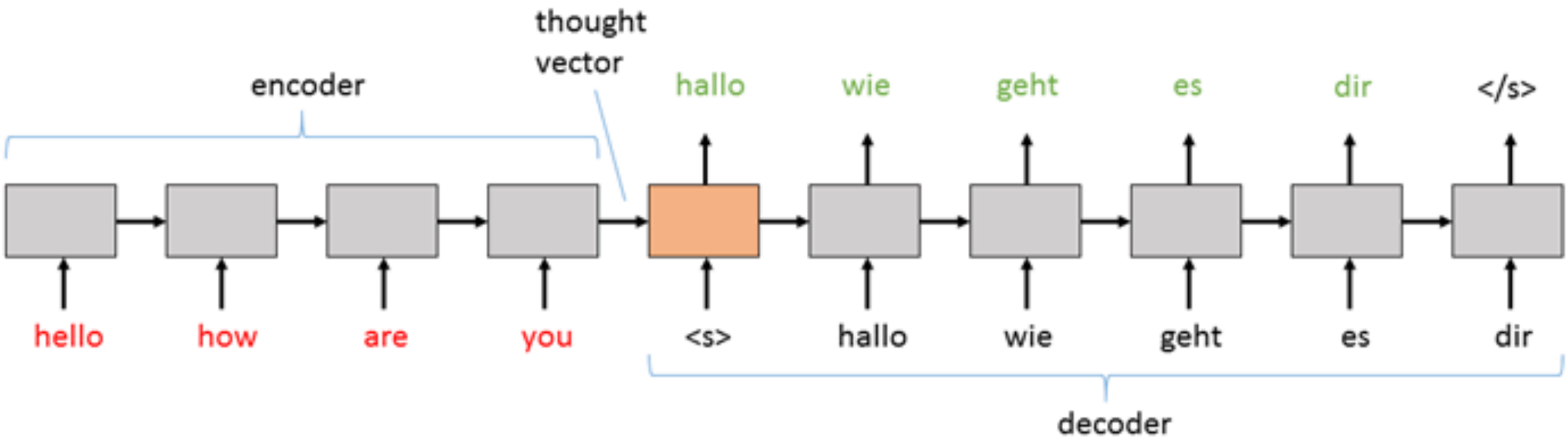
- A **single neural network** is used to translate from source to target
- Architecture: Encoder-Decoder
  - Two main components:
    - Encoder: Convert source sentence (input) into a vector/matrix
    - Decoder: Convert encoding into a sentence in target language (output)

Recall: RNNs

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^d$$



Sequence to Sequence learning (Seq2seq)

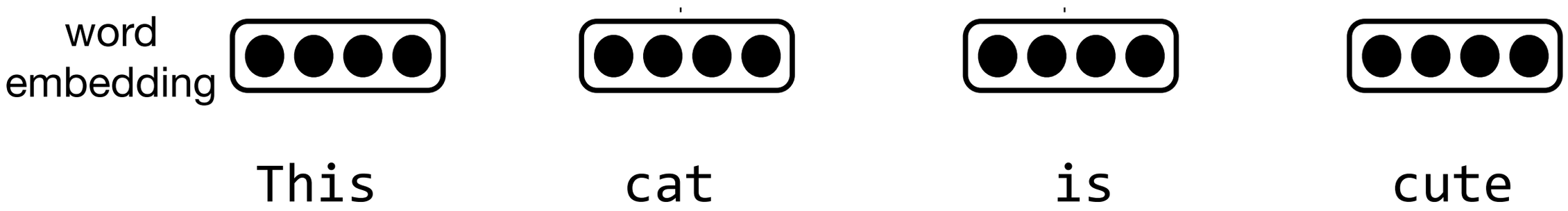


- Encode entire input sequence into a single vector **(using an RNN)**
- Decode one word at a time **(again, using an RNN!)**
- Beam search for better inference
- Learning is not trivial! (vanishing/exploding gradients)

(Sutskever et al., 2014)

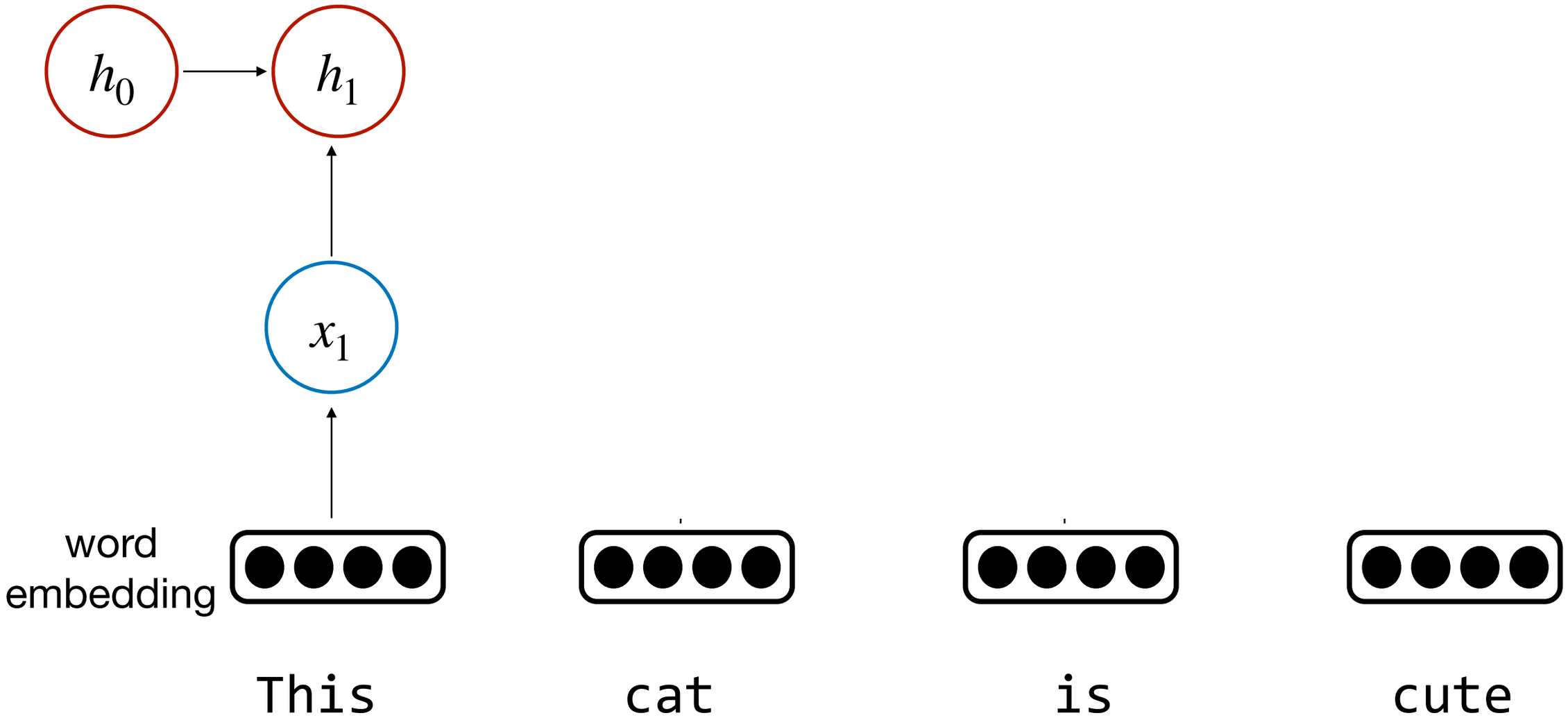
Encoder

Sentence: This cat is cute



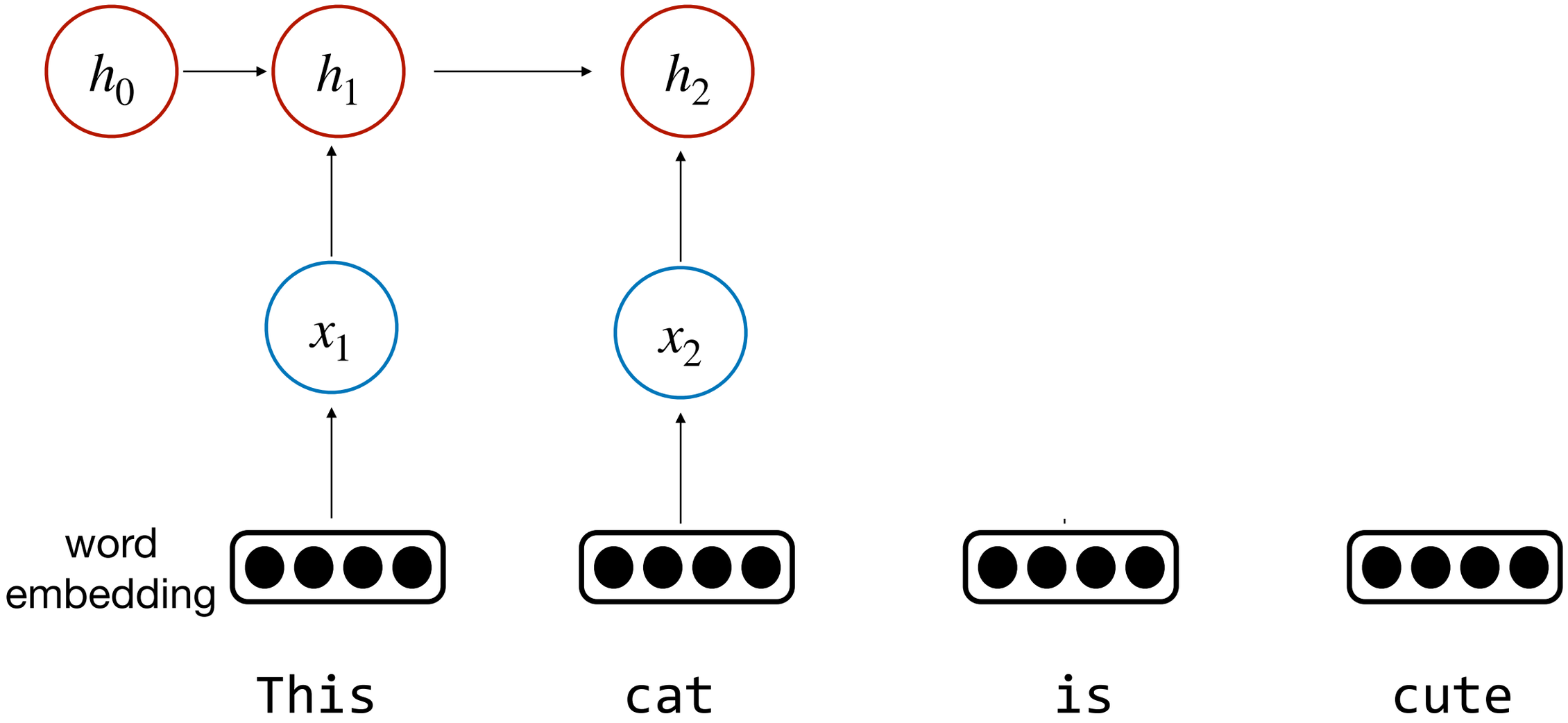
Encoder

Sentence: This cat is cute



Encoder

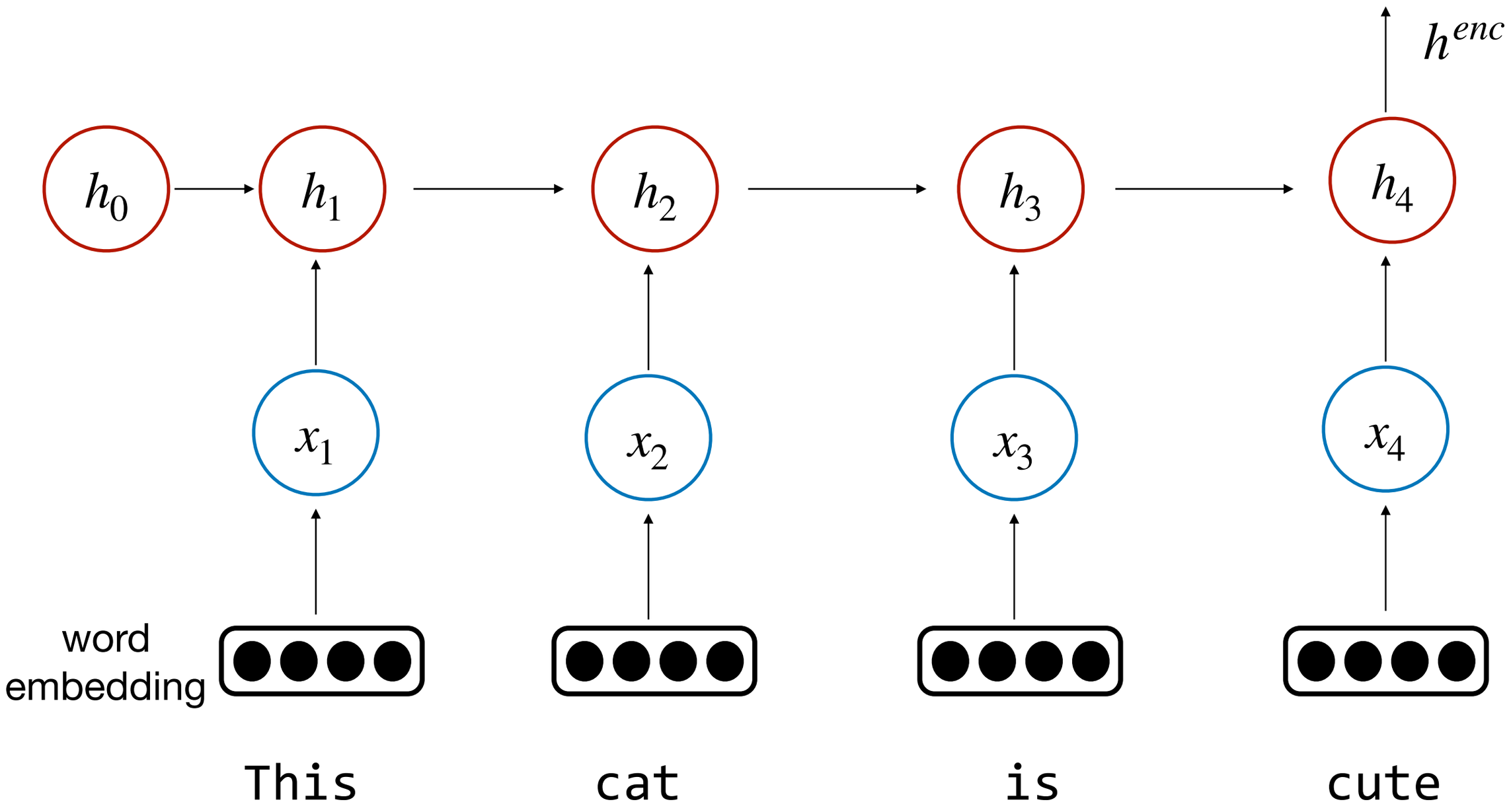
Sentence: This cat is cute



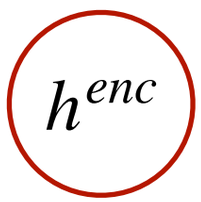
Encoder

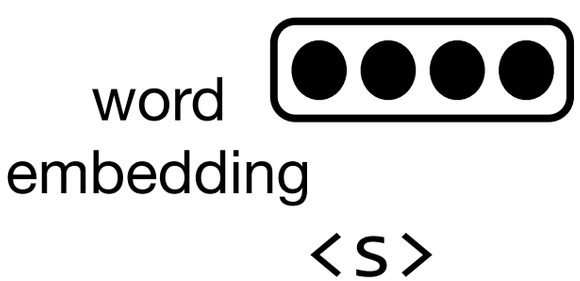
Sentence: This cat is cute

(encoded representation)

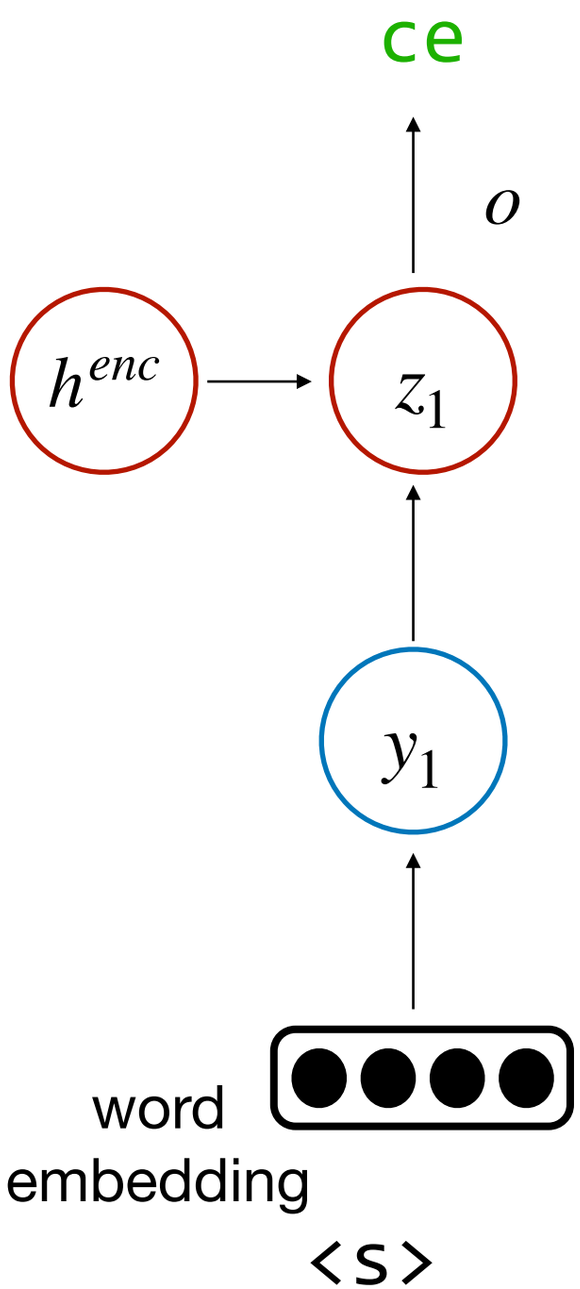


Decoder

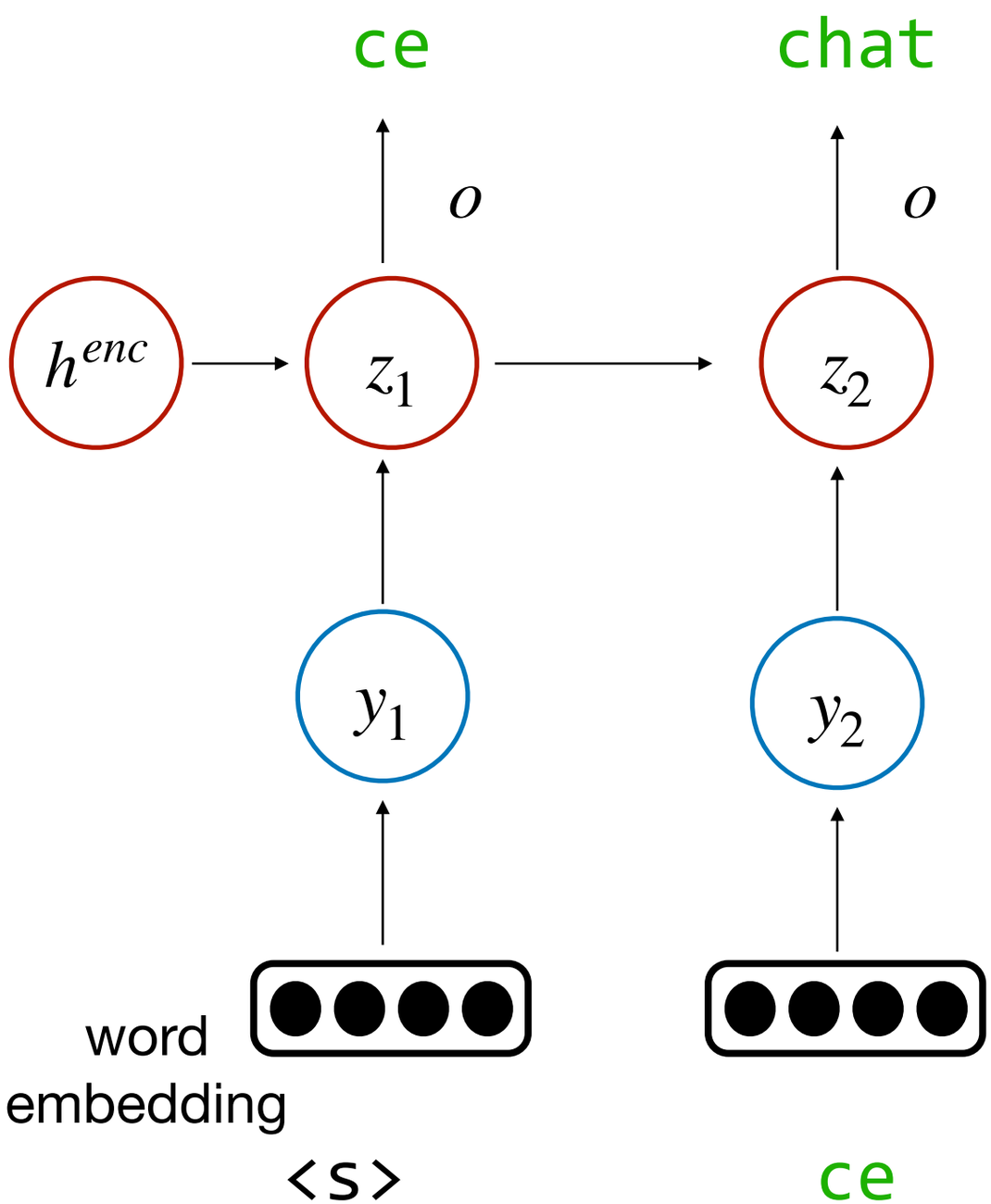




Decoder

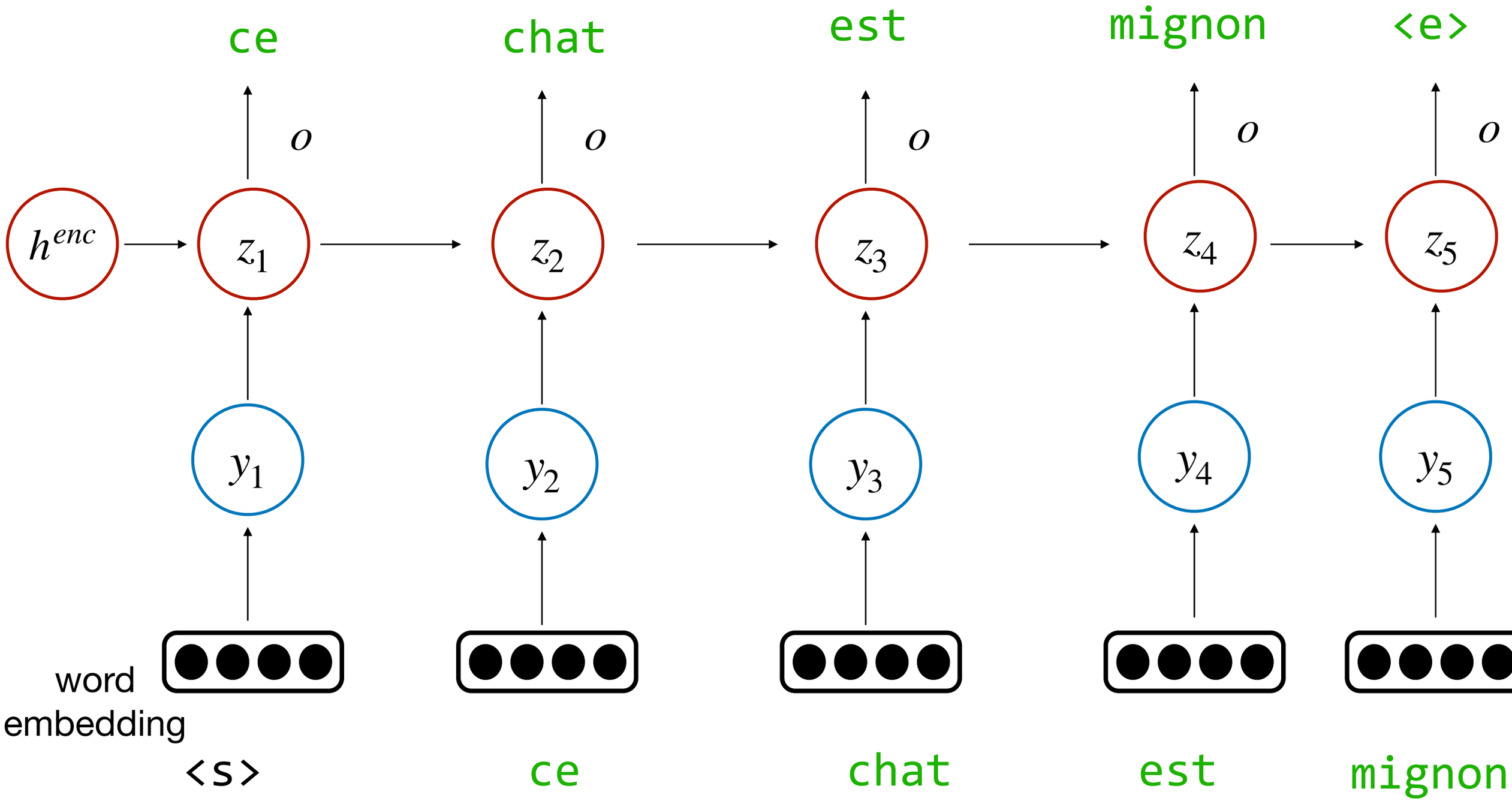


Decoder



Decoder

- A conditioned language model



Seq2seq training

- Similar to training a language model!
- Minimize cross-entropy loss:

$$\sum_{t=1}^{T} -\log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

- Back-propagate gradients through *both decoder and encoder*
- Need a really big corpus

36M sentence pairs

Russian: Машинный перевод - это круто!

English: Machine translation is cool!

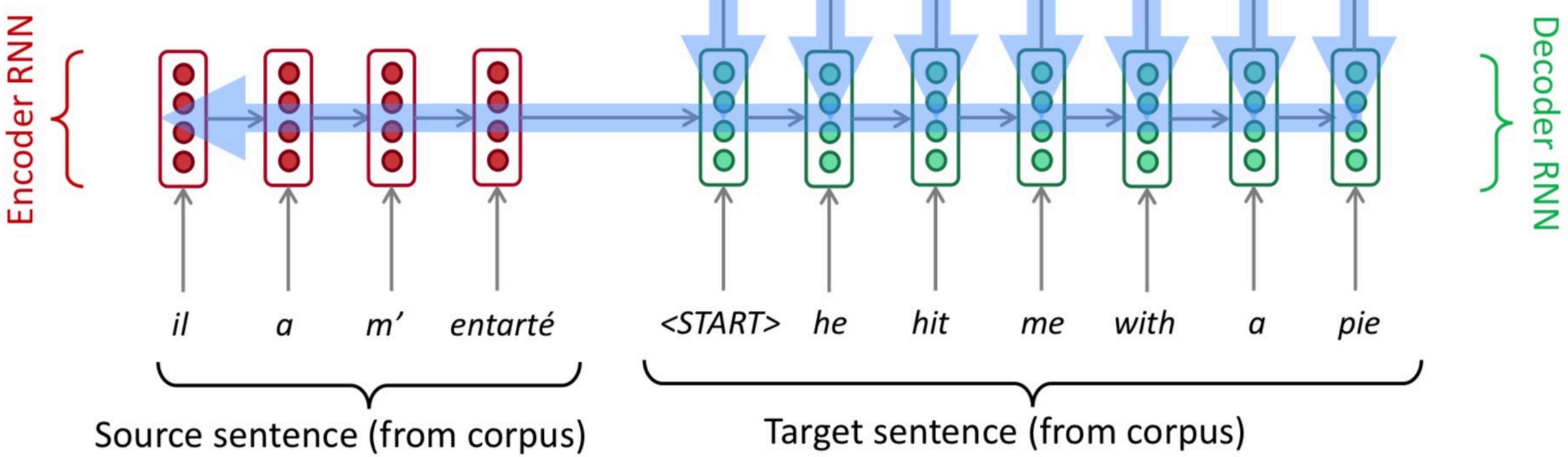
Seq2seq training

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

= negative log prob of "he"

= negative log prob of "with"

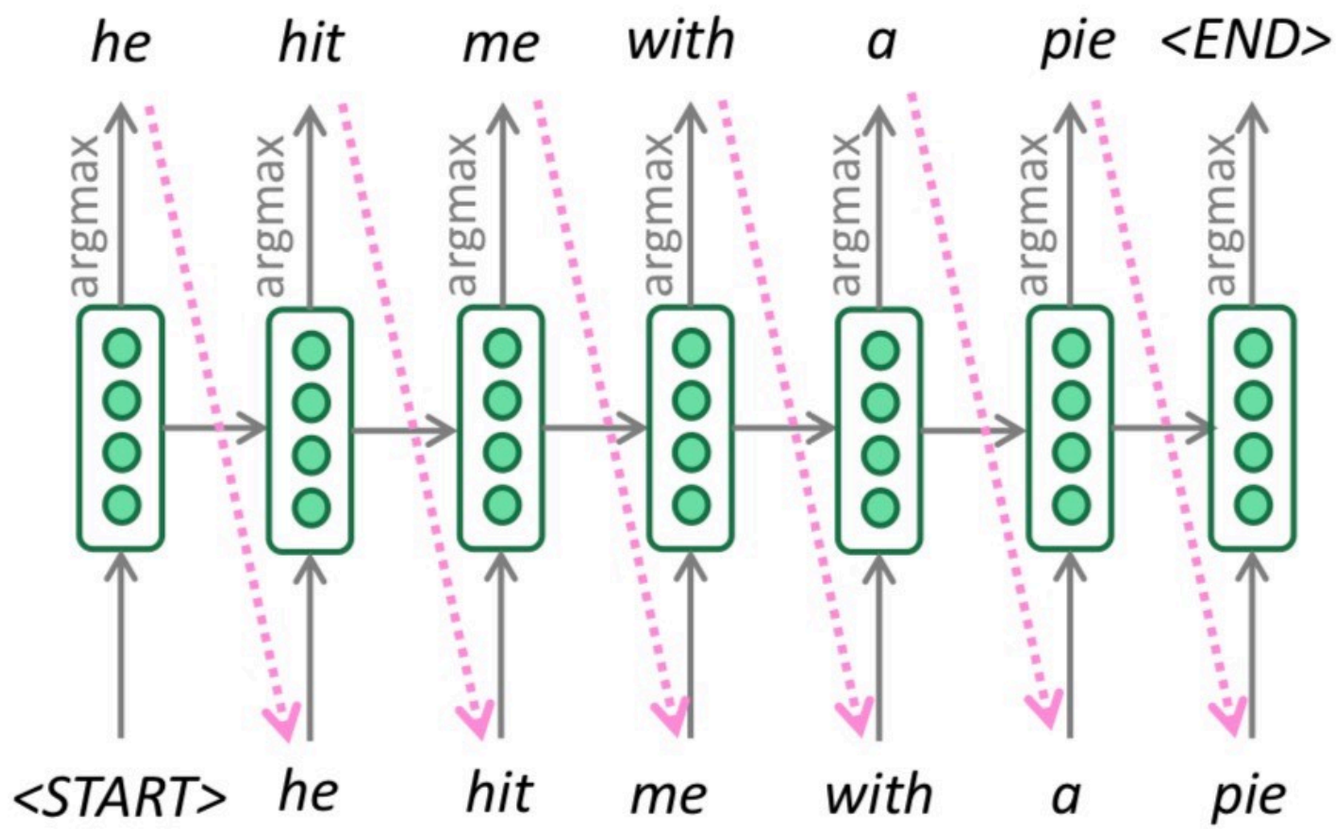
= negative log prob of <END>



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

(slide credit: Abigail See)

Greedy decoding



- Compute argmax at every step of decoder to generate word
- What's wrong?

Exhaustive search?

- Find $\arg\max_{y_1,\ldots,y_T} P(y_1,\ldots,y_T|x_1,\ldots,x_n)$
- Requires computing all possible sequences
  - $O(V^T)$ complexity!
  - Too expensive

A middle ground: Beam search

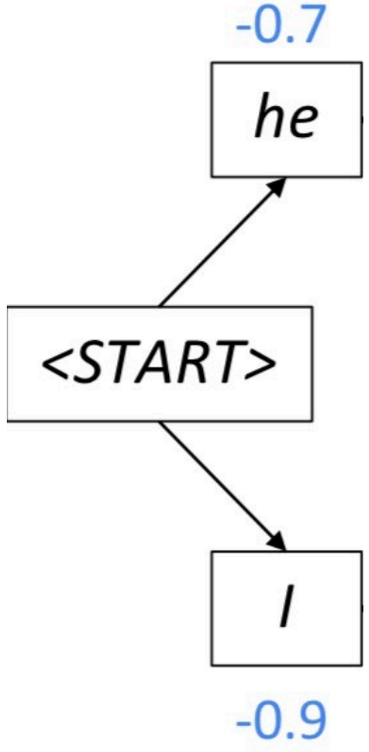
- **Key idea:** At every step, keep track of the k most probable partial translations (hypotheses)
- Score of each hypothesis = log probability

$$\sum_{t=1}^{j} \log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

- Not guaranteed to be optimal
- More efficient than exhaustive search

Beam decoding

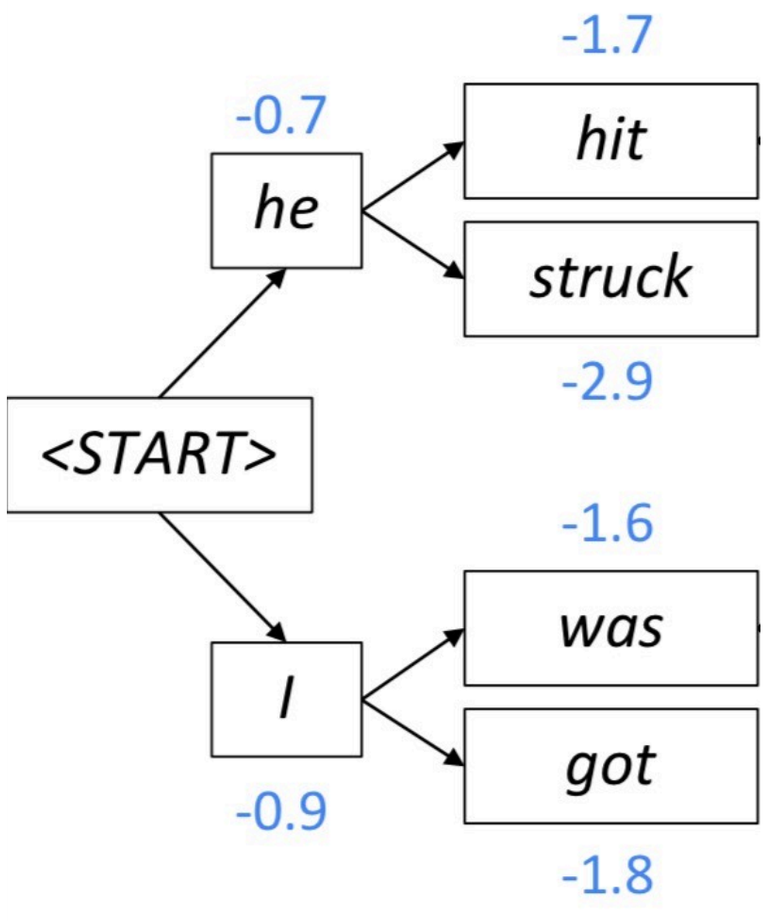
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



(slide credit: Abigail See)

Beam decoding

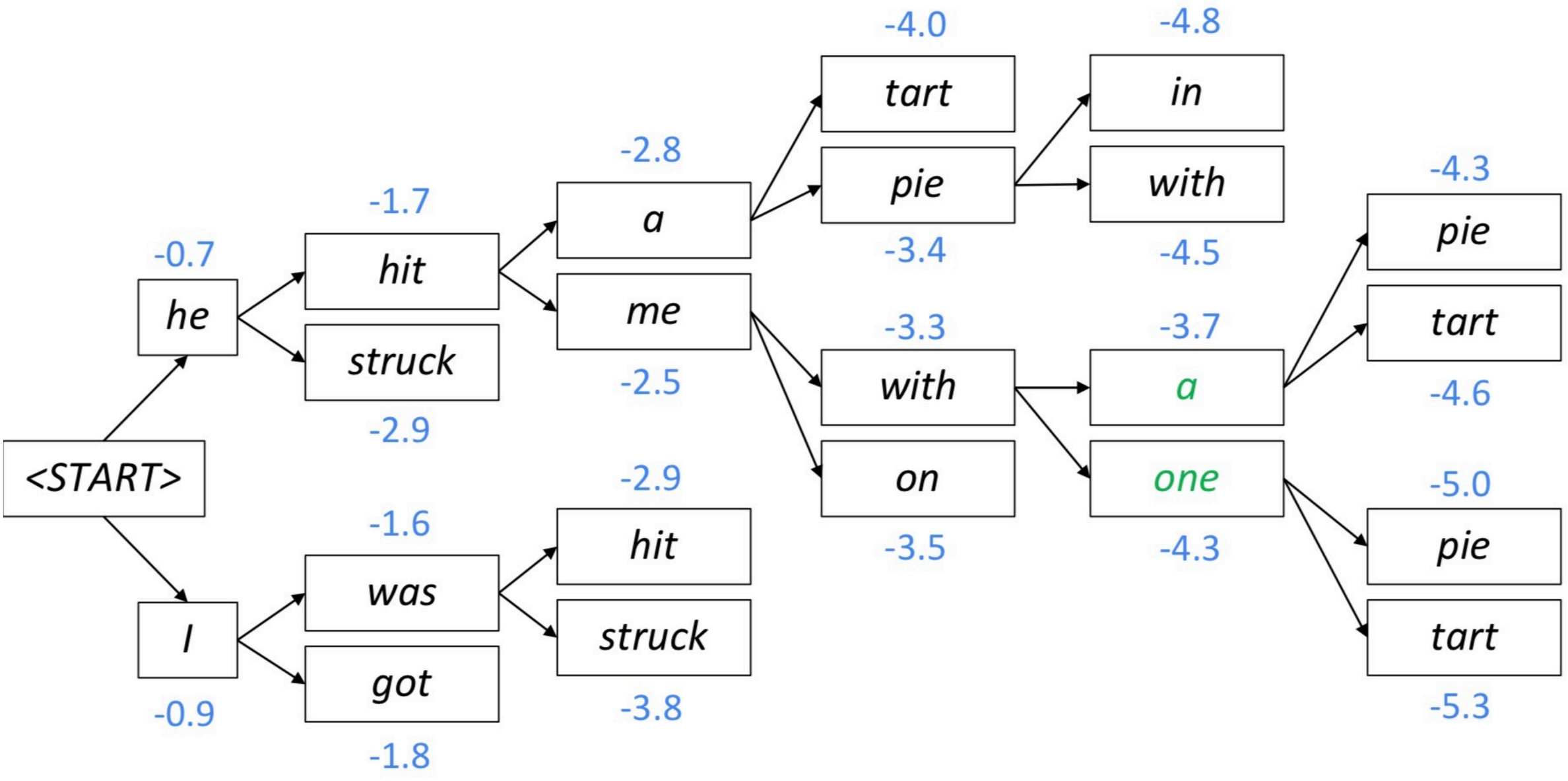
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



(slide credit: Abigail See)

Beam decoding

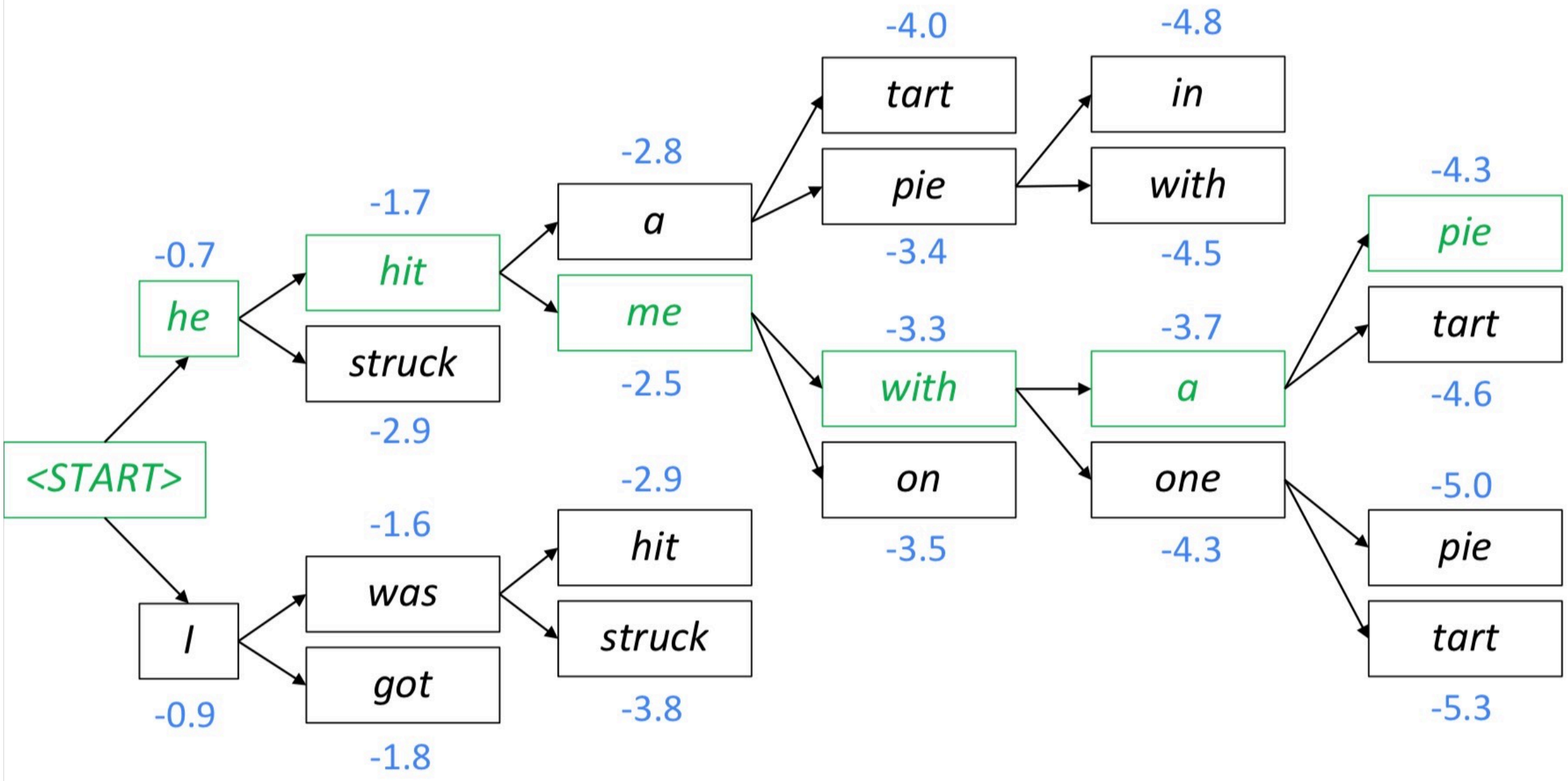
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



(slide credit: Abigail See)

Backtrack

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



(slide credit: Abigail See)

Beam decoding

- Different hypotheses may produce $\langle e \rangle$ (end) token at different time steps
  - When a hypothesis produces $\langle e \rangle$, stop expanding it and place it aside
- Continue beam search until:
  - All k hypotheses produce $\langle e \rangle$ OR
  - Hit max decoding limit T
- Select top hypotheses using the *normalized* likelihood score

$$\frac{1}{T} \sum_{t=1}^{T} \log P(y_t \,|\, y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$

  - Otherwise shorter hypotheses have higher scores

NMT vs SMT

Pros

-

Cons

-

NMT vs SMT

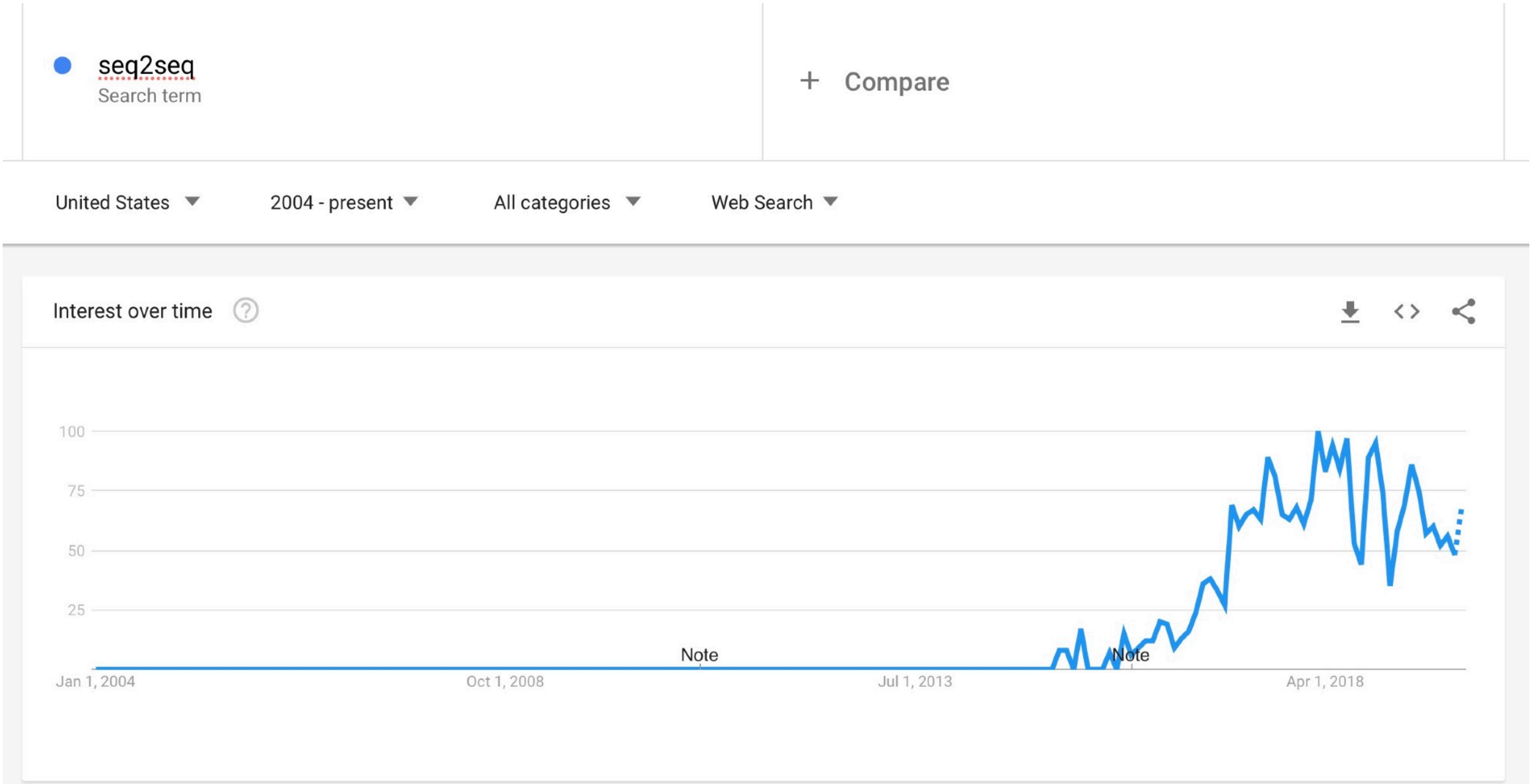
Pros

- Better performance
  - Fluency
  - Longer context
- Single NN optimized end-to-end
- Less engineering
- Works out of the box for many language pairs

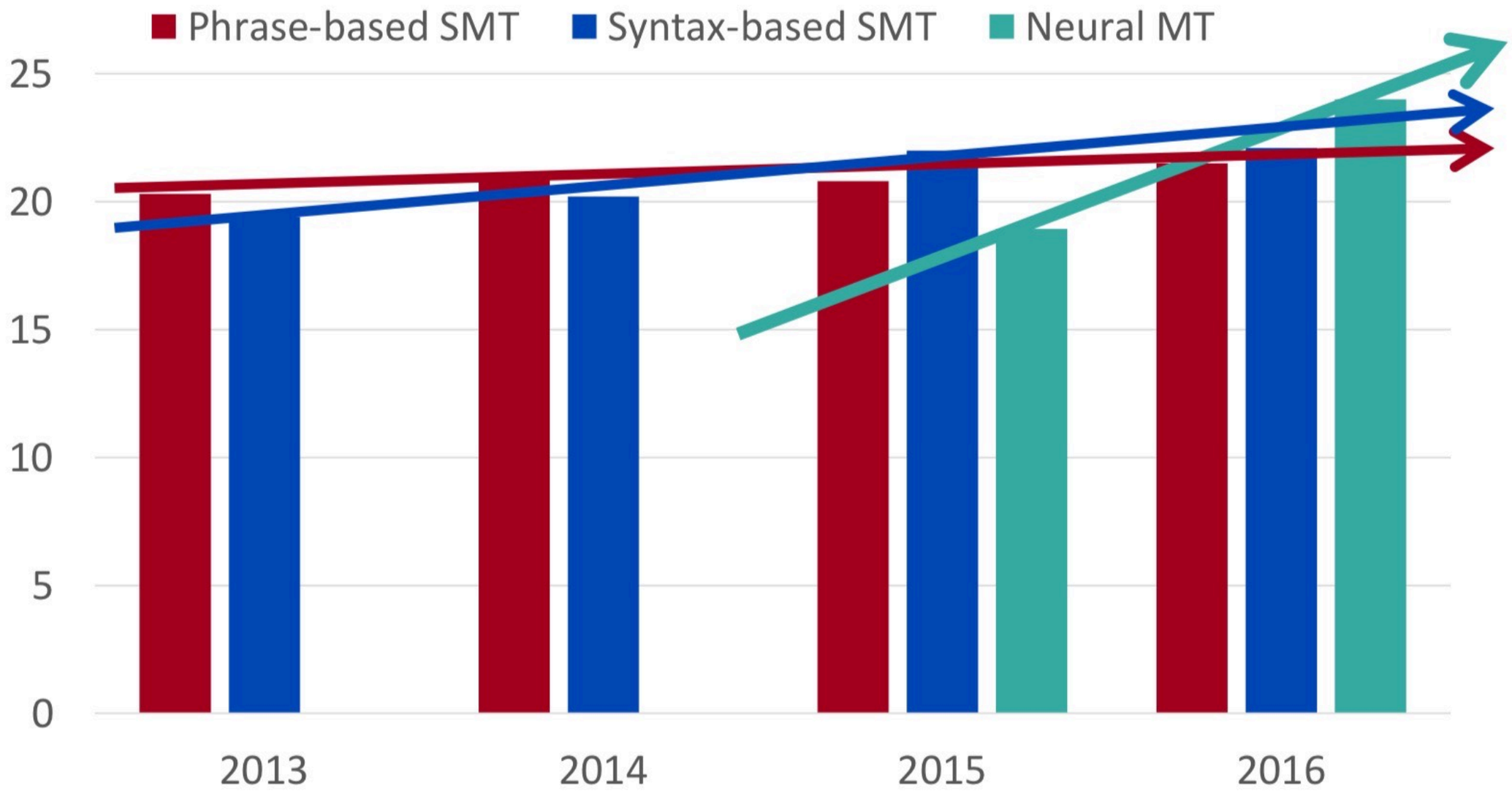
Cons

- Requires more data and compute
- Less interpretable
  - Hard to debug
- Uncontrollable
  - Heavily dependent on data - could lead to unwanted biases
- More parameters

How seq2seq changed the MT landscape



MT Progress



(source: Rico Sennrich)

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Table 10: Mean of side-by-side scores on production data

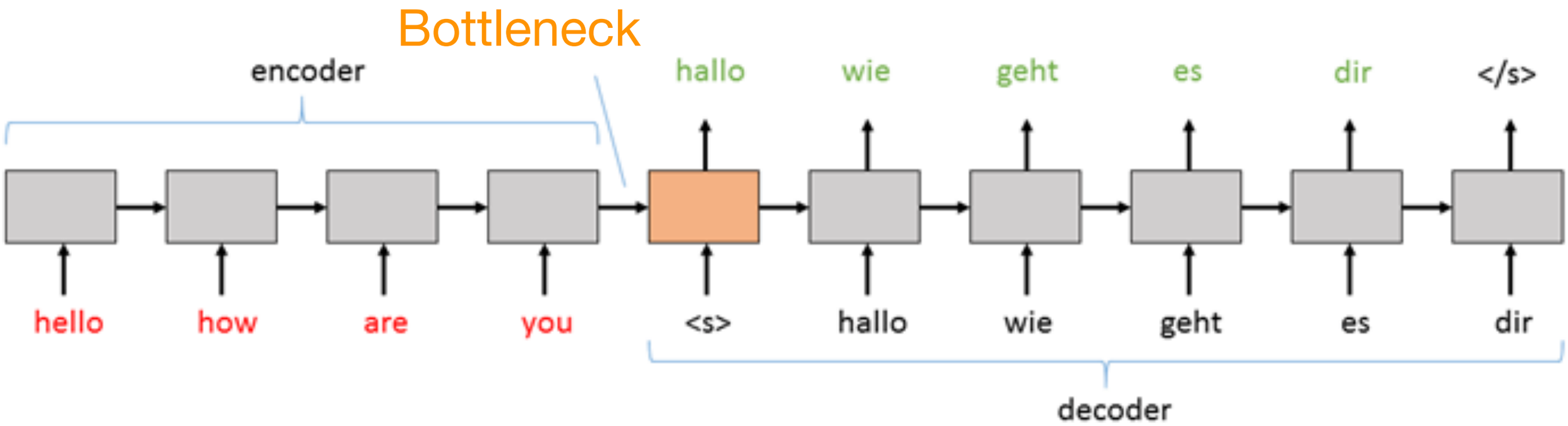
	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

(Wu et al., 2016)

Versatile seq2seq

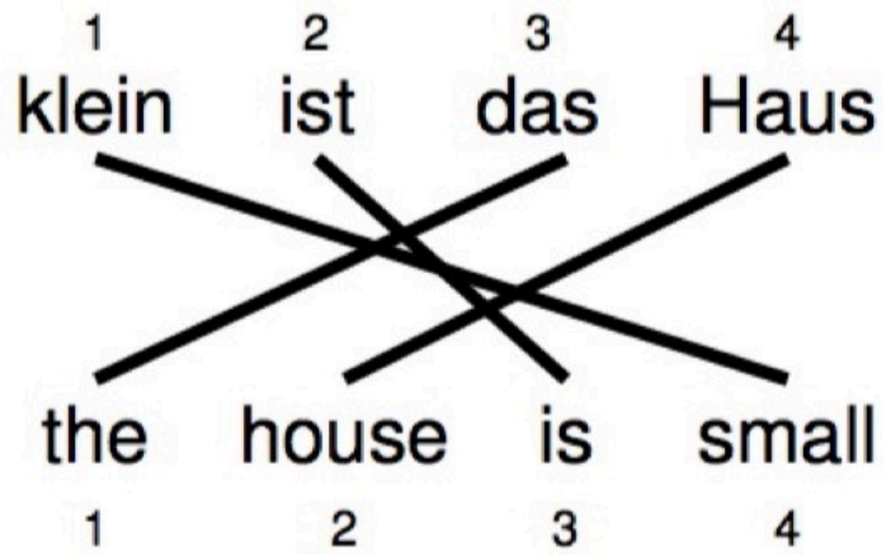
- Seq2seq finds applications in many other tasks!
- Any task where inputs and outputs are sequences of words/ characters
  - Summarization (input text → summary)
  - Dialogue (previous utterance → reply)
  - Parsing (sentence → parse tree in sequence form)
  - Question answering (context+question → answer)

Issues with vanilla seq2seq

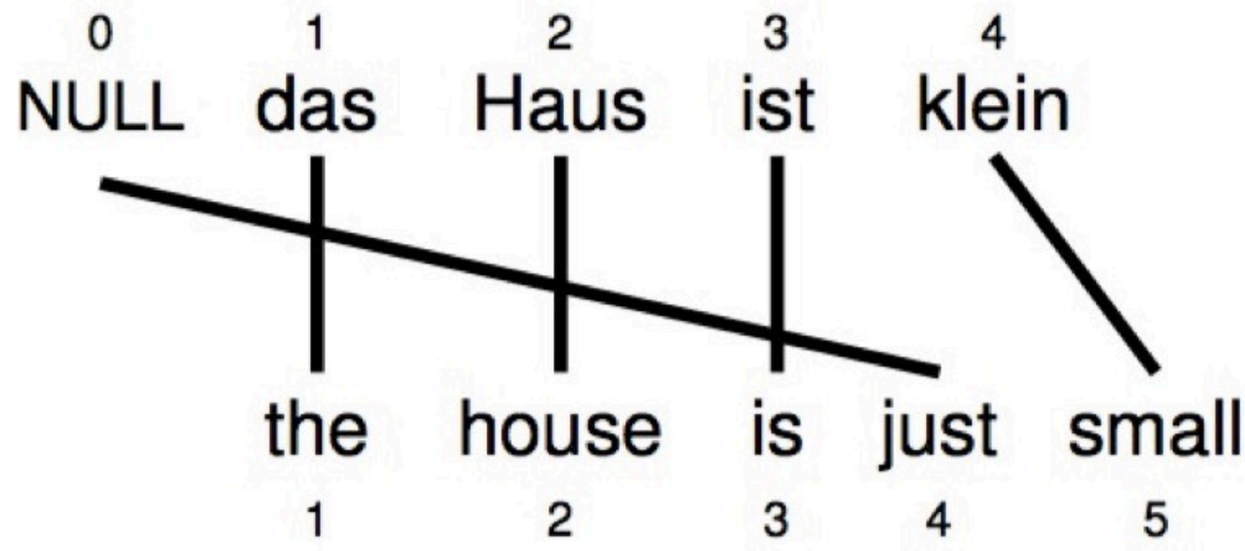


- A single encoding vector, h^{enc}, needs to capture all the information about source sentence
- Longer sequences can lead to vanishing gradients
- Overfitting

Remember alignments?



$$\mathbf{a} = (3, 4, 2, 1)^\top$$

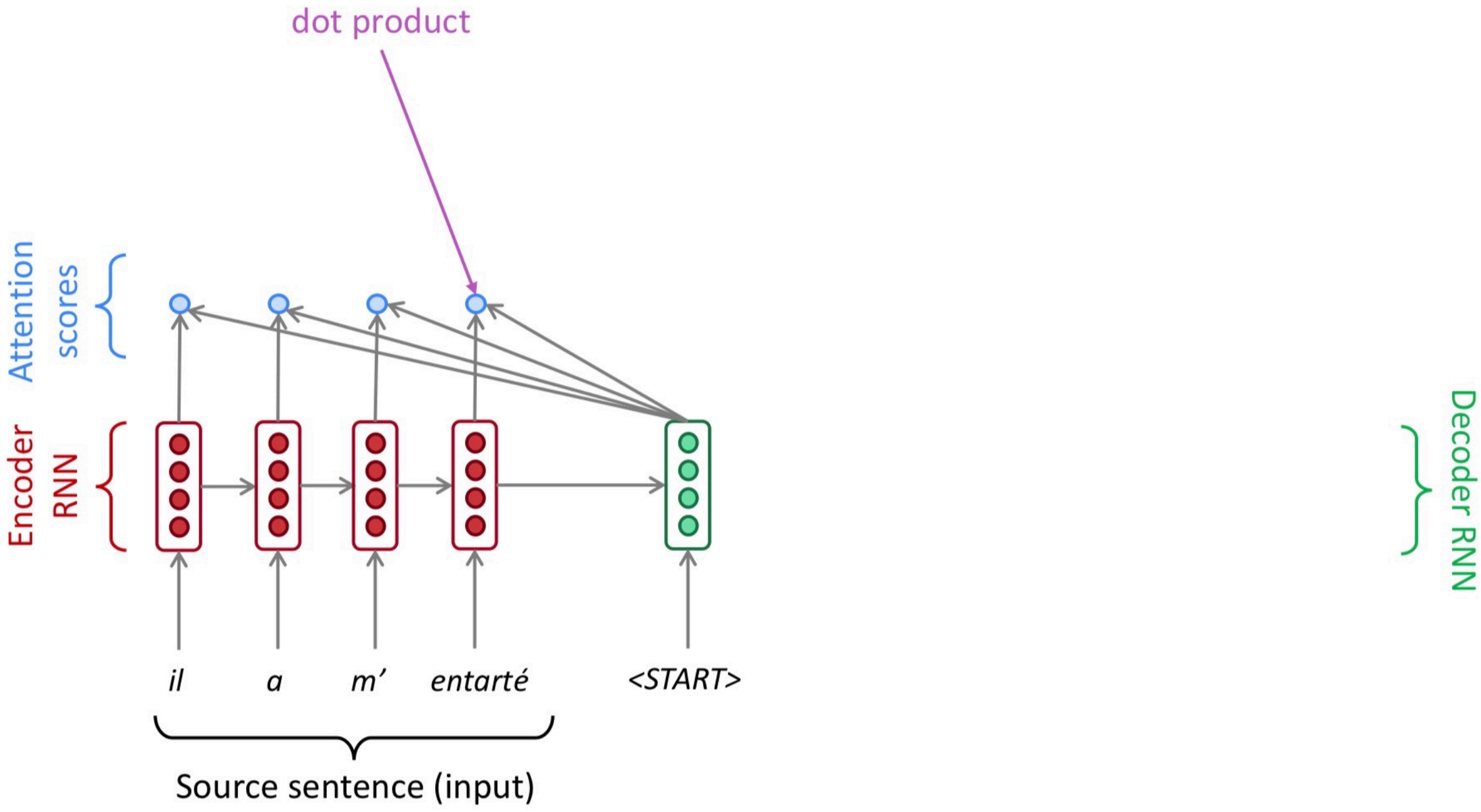


$$\mathbf{a} = (1, 2, 3, 0, 4)^\top$$

Attention

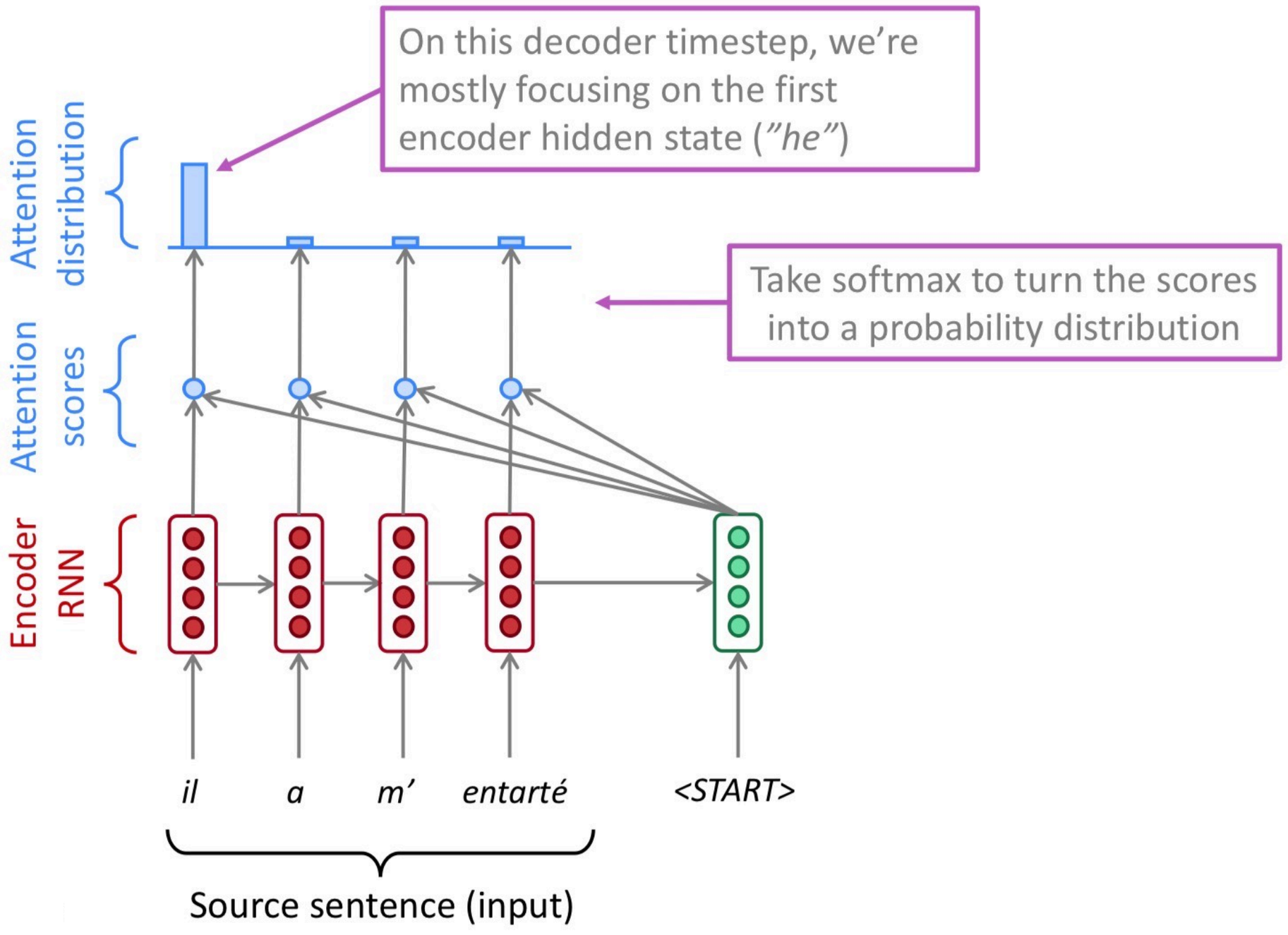
- The neural MT equivalent of alignment models
- Key idea: At each time step during decoding, **focus on a particular part** of source sentence
  - This depends on the decoder's current hidden state (i.e. notion of what you are trying to decode)
  - Usually implemented as a probability distribution over the hidden states of the encoder (h_i^{enc})

Seq2seq with attention



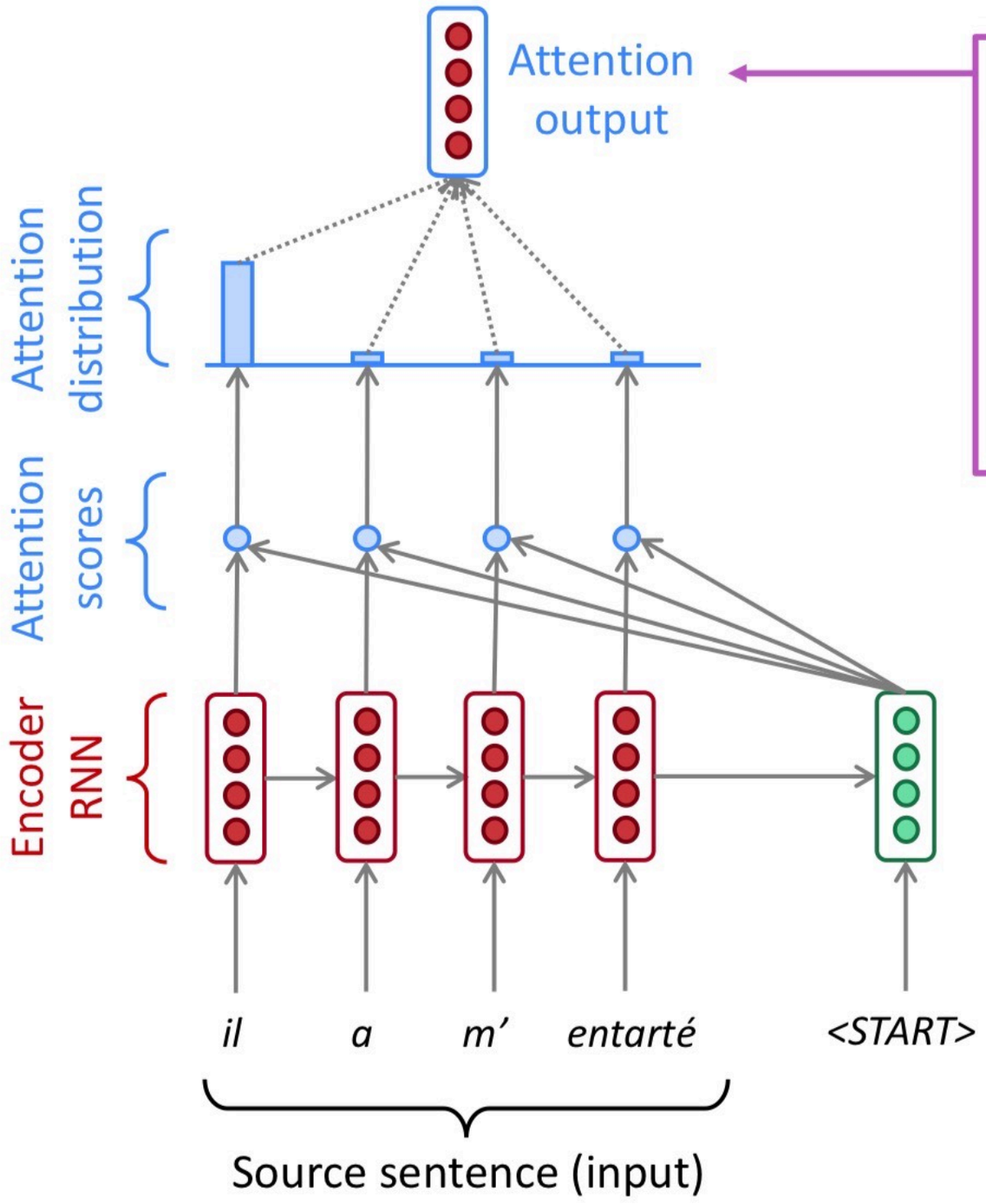
(slide credit: Abigail See)

Seq2seq with attention



(slide credit: Abigail See)

Seq2seq with attention

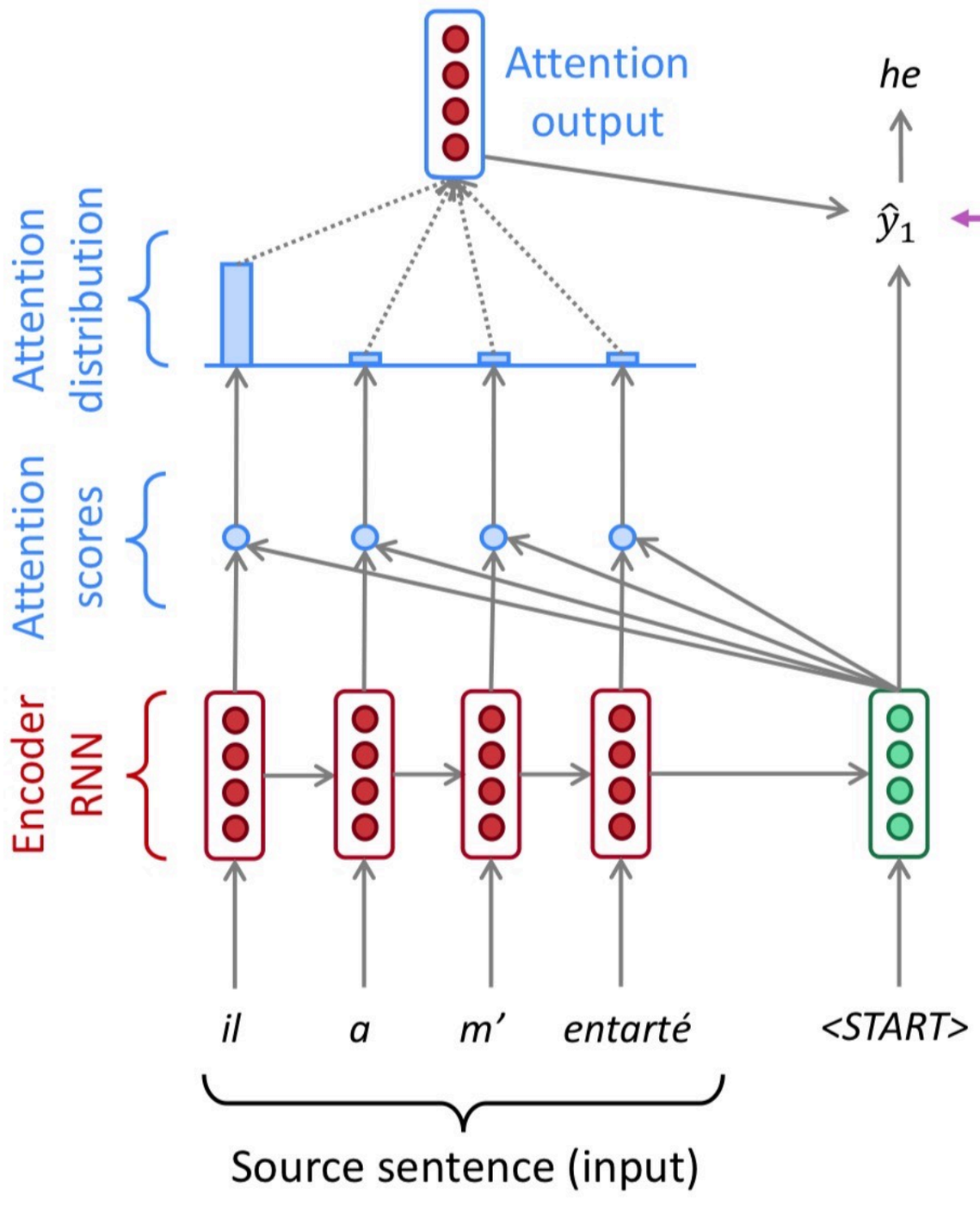


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

(slide credit: Abigail See)

Seq2seq with attention

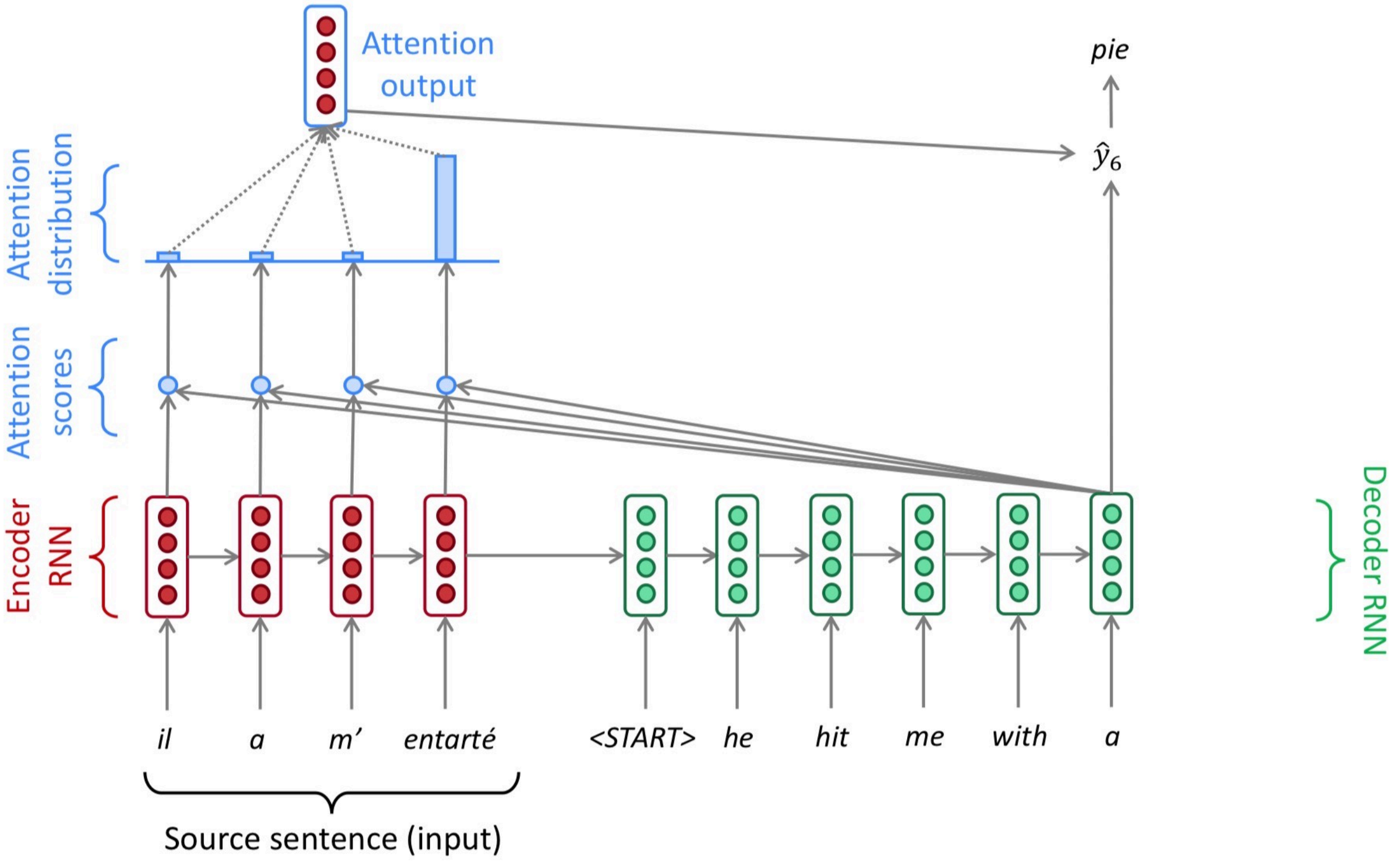


Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Can also use $\hat{y}_1$ as input for next time step

(slide credit: Abigail See)

Seq2seq with attention



(slide credit: Abigail See)

Computing attention

- Encoder hidden states: $h_1^{enc}, \ldots, h_n^{enc}$
- Decoder hidden state at time t: h_t^{dec}
- First, get attention scores for this time step (we will see what g is soon!):

$$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})]$$

- Obtain the attention distribution using softmax:

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

- Compute weighted sum of encoder hidden states:

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

- Finally, concatenate with decoder state and pass on to output layer: $[a_t; h_t^{dec}] \in \mathbb{R}^{2h}$

Types of attention

- Assume encoder hidden states $h_1, h_2, \ldots, h_n$ and decoder hidden state z

1. **Dot-product attention** (assumes equal dimensions for a and b:

$$e_i = g(h_i, z) = z^T h_i \in \mathbb{R}$$

2. **Multiplicative attention:**

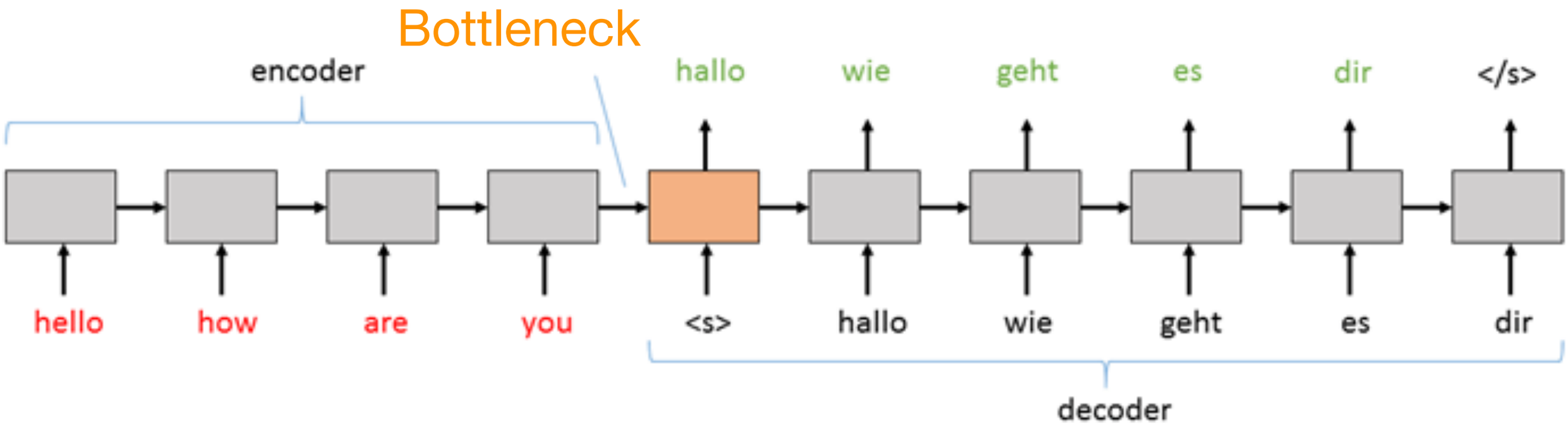
$g(h_i, z) = z^T W h_i \in \mathbb{R}$, where W is a weight matrix

3. **Additive attention:**

$$g(h_i, z) = v^T \tanh\left(W_1 h_i + W_2 z\right) \in \mathbb{R}$$

where W_1, W_2 are weight matrices and v is a weight vector

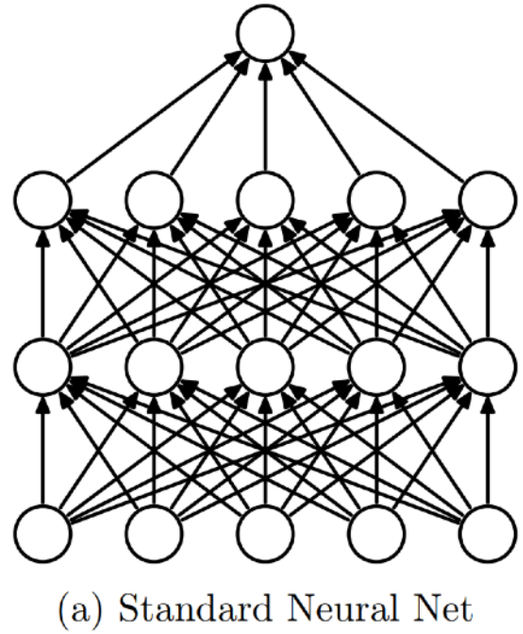
Issues with vanilla seq2seq



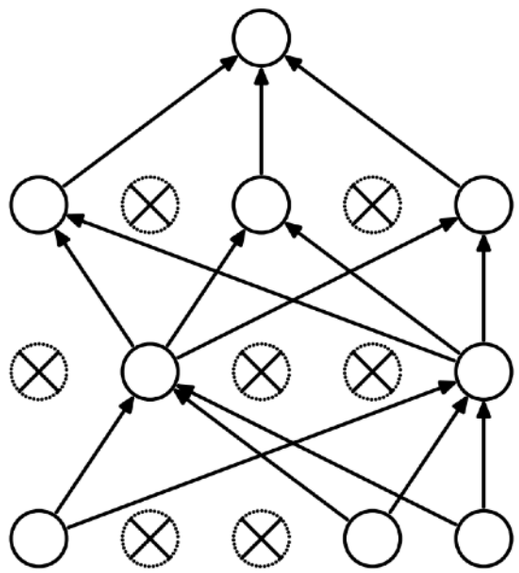
- A single encoding vector, h^{enc}, needs to capture all the information about source sentence
- Longer sequences can lead to vanishing gradients
- **Overfitting**

Dropout

- Form of regularization for RNNs (and any NN in general)
- Idea: “Handicap” NN by removing hidden units **stochastically**
  - set each hidden unit in a layer to 0 with probability p during training ($p = 0.5$ usually works well)
  - scale outputs by $1/(1-p)$
  - hidden units forced to learn more general patterns
- Test time: Simply compute identity



(a) Standard Neural Net

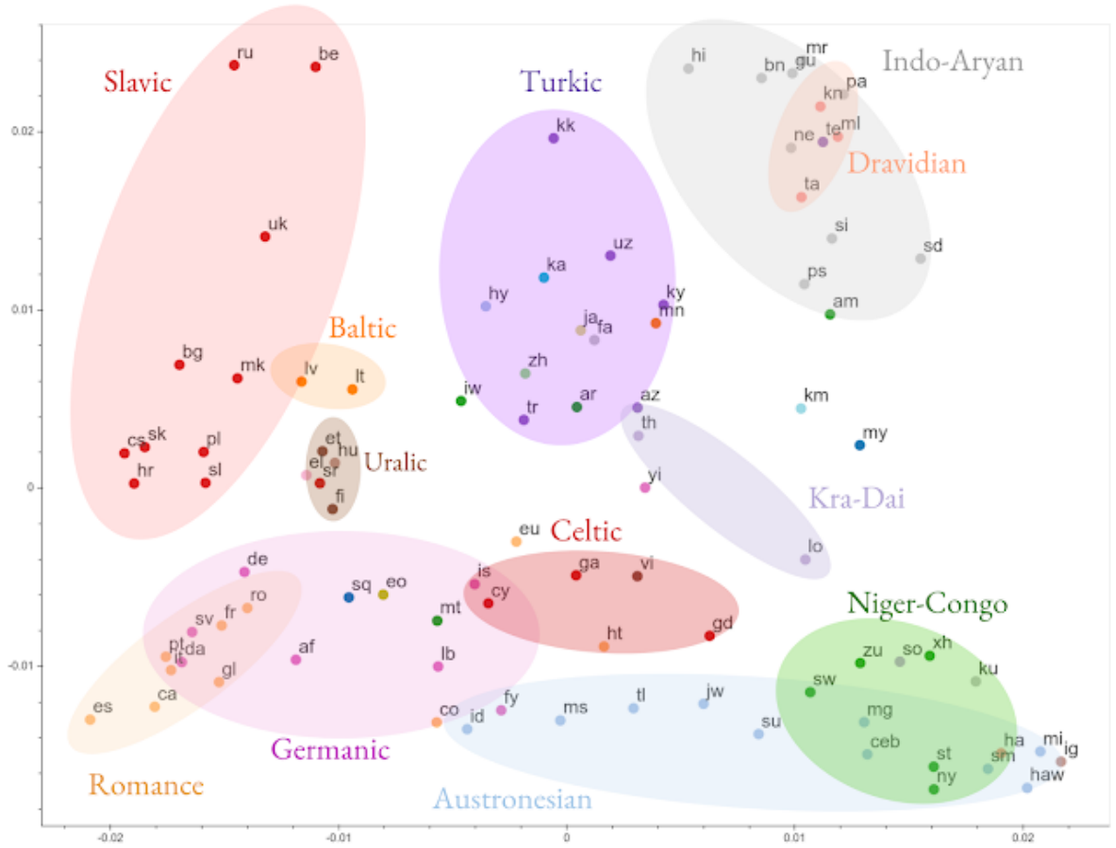


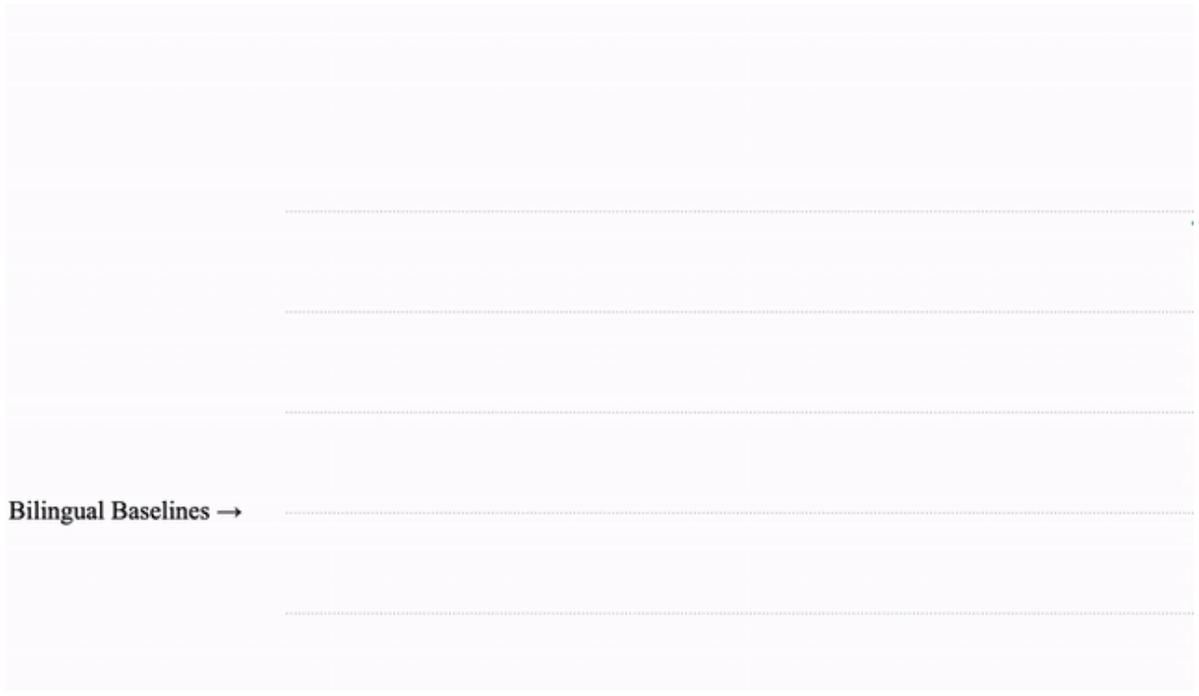
(b) After applying dropout.

Existing challenges with NMT

- Out-of-vocabulary words
- Low-resource languages
- Long-term context
- Common sense knowledge (e.g. hot dog, paper jam)
- Fairness and bias
- Uninterpretable

Massively multilingual MT





- Train a *single* neural network on 103 languages paired with English (remember Interlingua?)
- Massive improvements on low-resource languages

(Arivazhagan et al., 2019)