CSE 156, Fall 2024

Lecture 6: Neural Machine Translation & Seq2Seq

Ndapa Nakashole, UCSD

17 October 2024

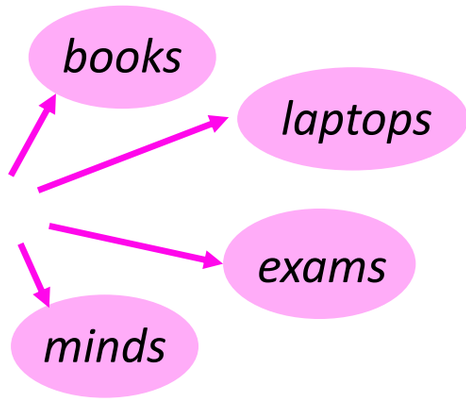
Announcements

- PA1 tomorrow **Friday, October 18th**
- PA2 late tomorrow **Friday, October 18th**
- **Quiz 1 of 3** out sometime next week

Recap

Language Modeling is the task of predicting what word comes next

the students opened their _____

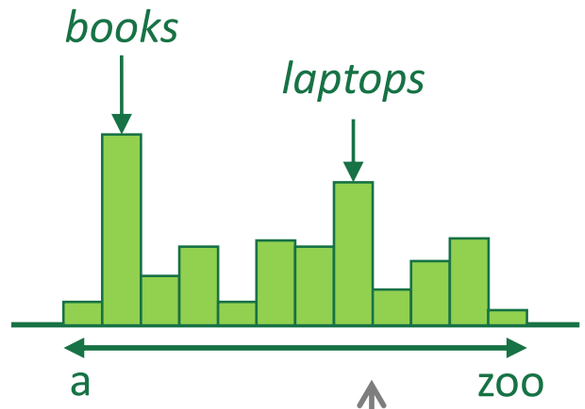


Recurrent Neural Network: A family of neural networks that:

- Take sequential input of any length
- Apply the same weights on each step

Recap: RNN Language Model

$$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

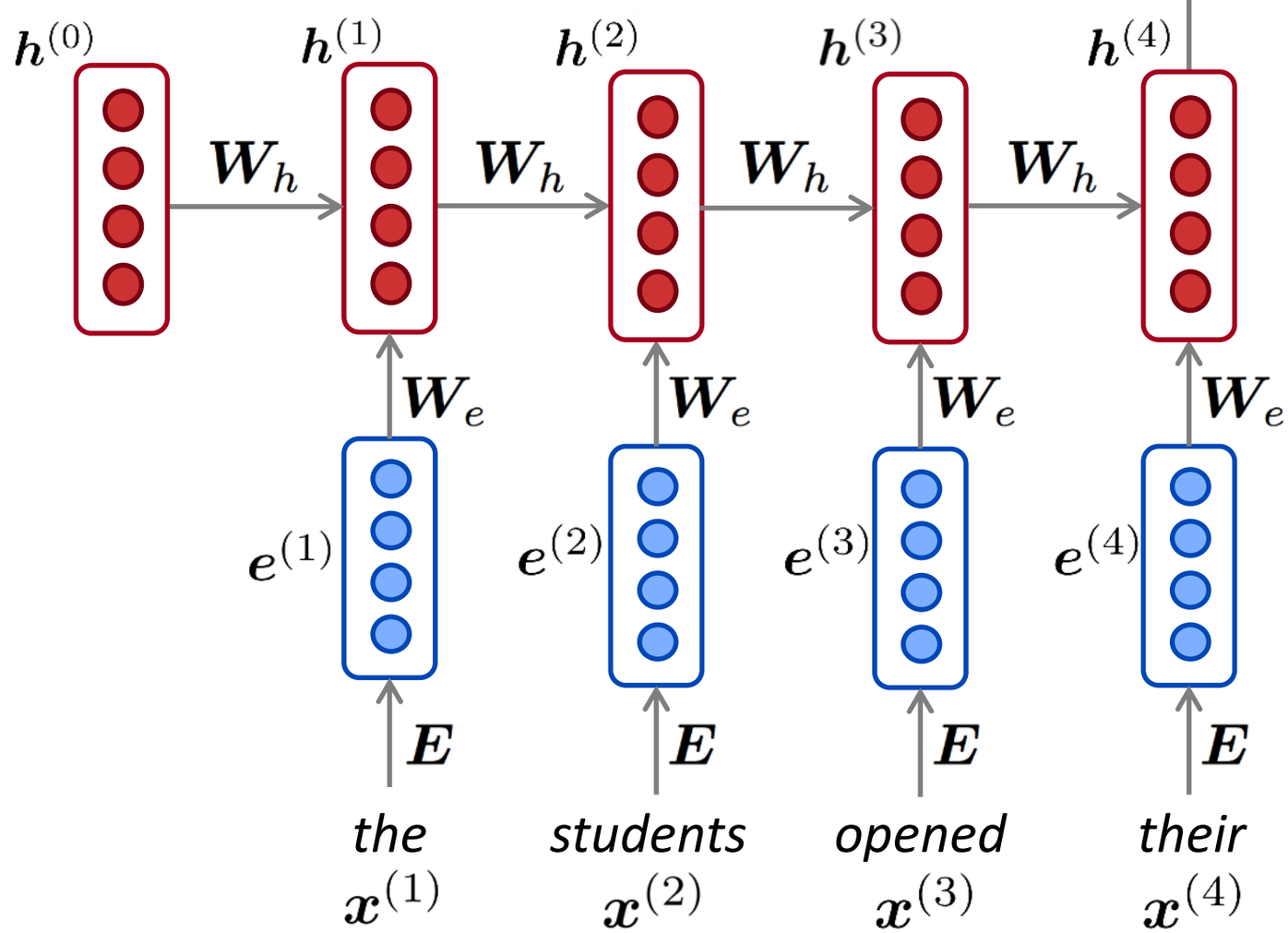
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

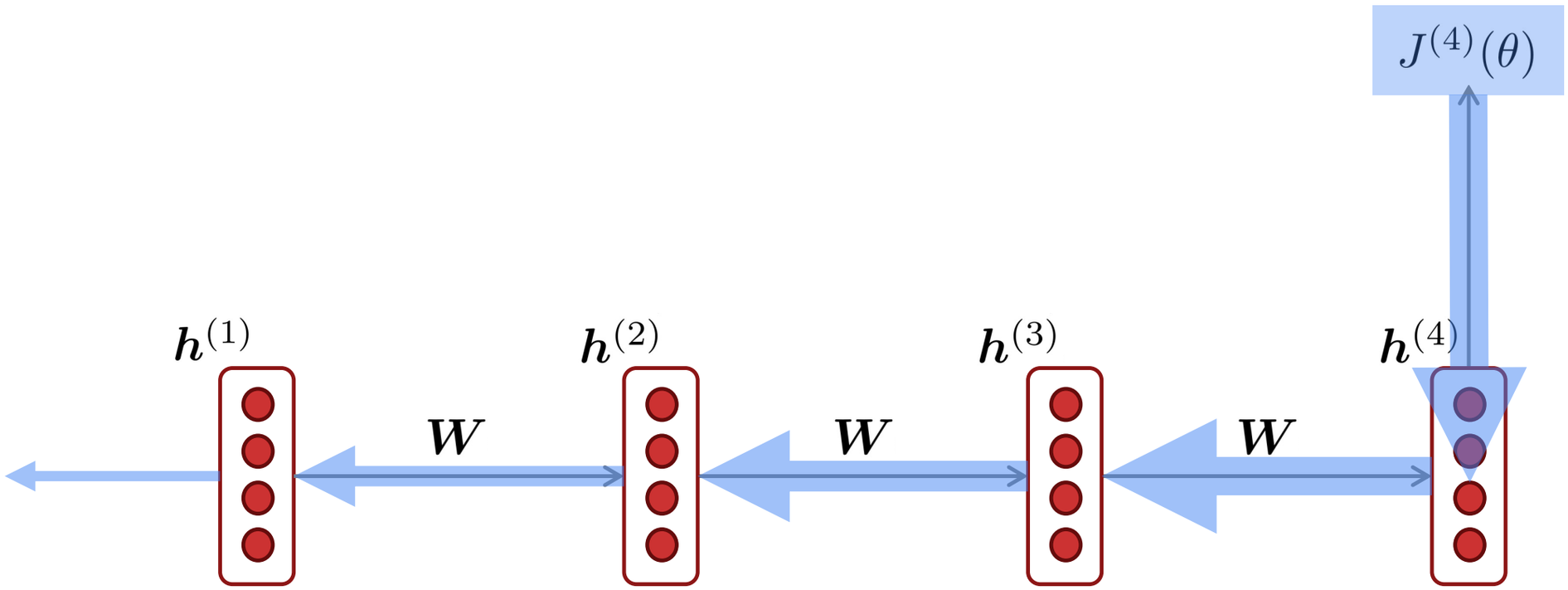
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



Recap: Vanishing Gradients in RNNs



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \boxed{\frac{\partial h^{(2)}}{\partial h^{(1)}}} \times \boxed{\frac{\partial h^{(3)}}{\partial h^{(2)}}} \times \boxed{\frac{\partial h^{(4)}}{\partial h^{(3)}}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

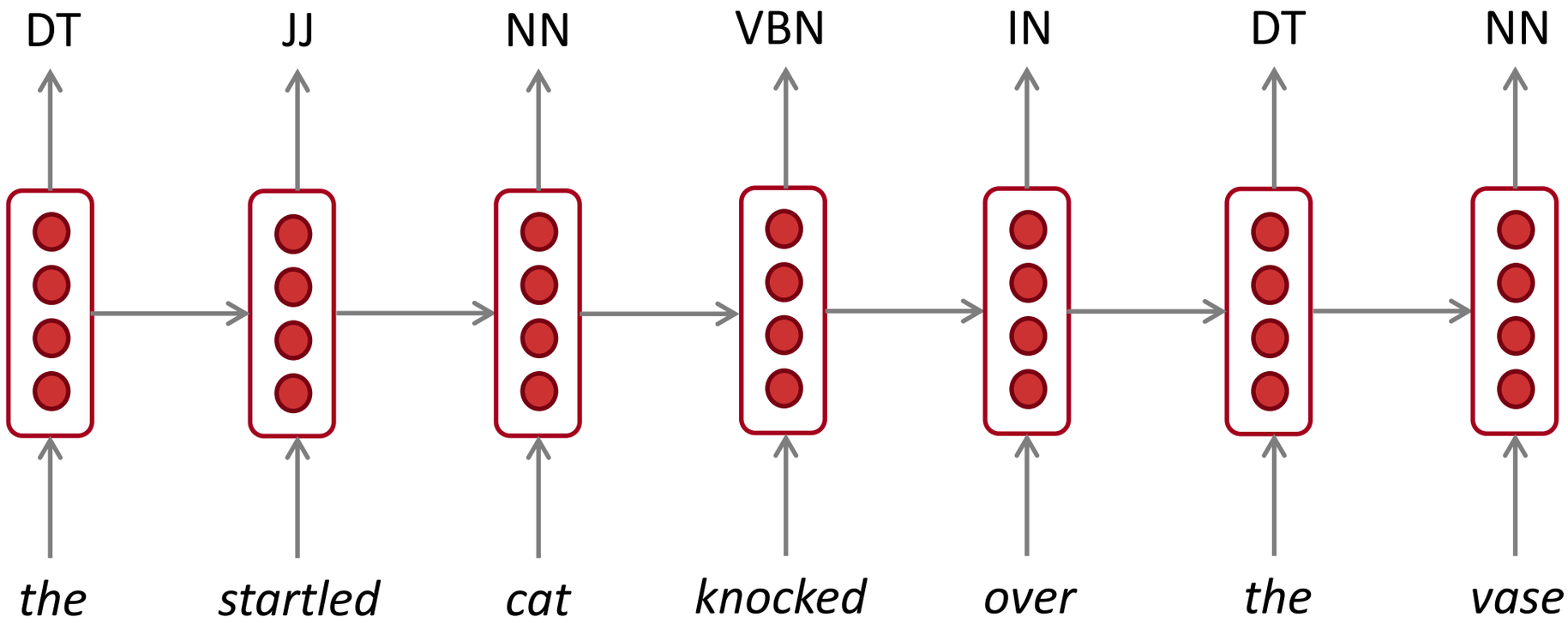
RNN Language Model

- RNN we have described so far = **"vanilla RNN"**

- Fancy **RNN variants**
  - **Bidirectional**
  - **Multi-layer**
  - **LSTM**
  - **GRU**

Bidirectionality & stacked layers

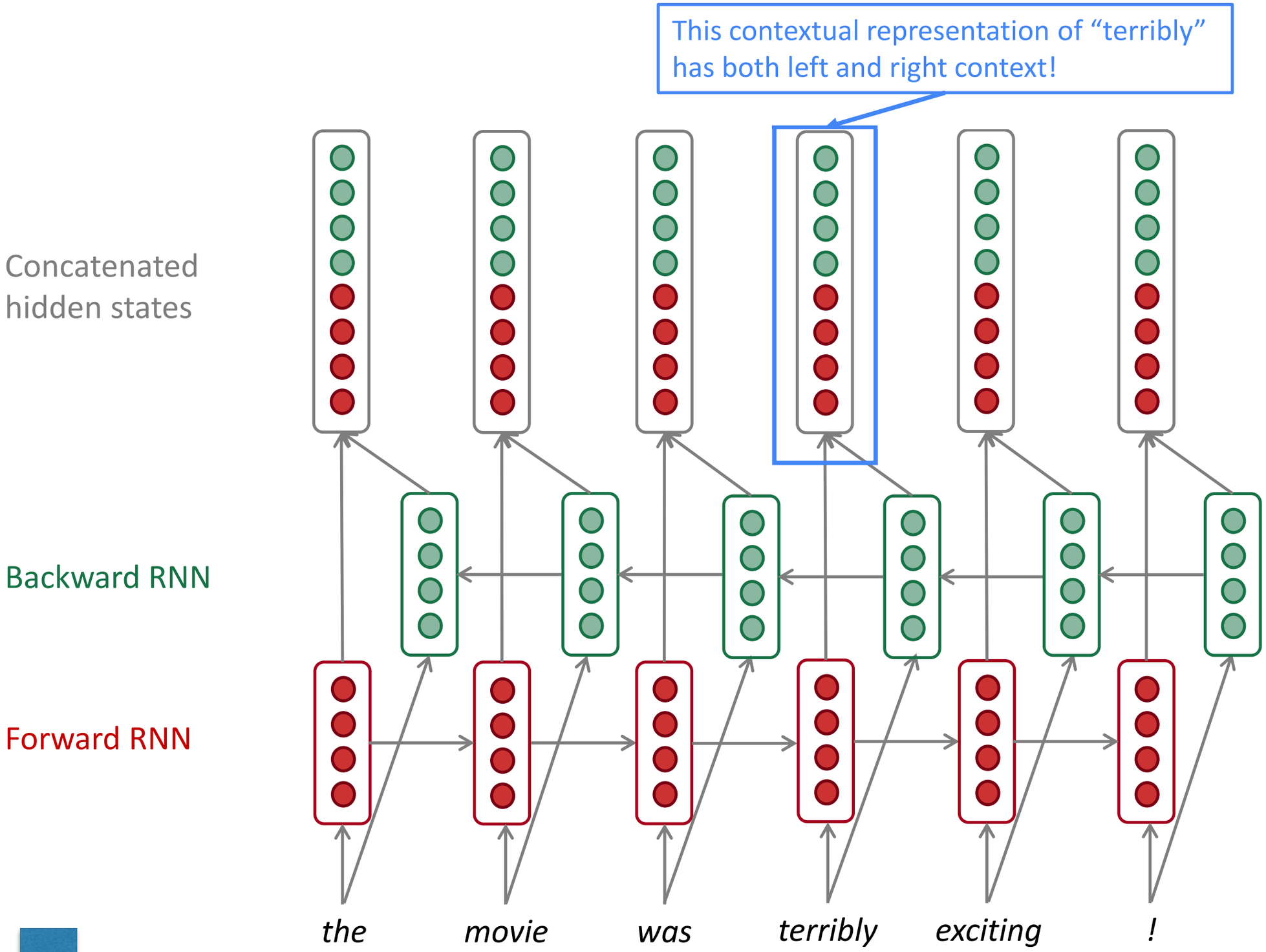
Unidirectional RNNs problem



- Arrows go in one direction, not possible for e.g. output at Y_4 to be influenced by later inputs x5, x6

- Only allows forward computation, also want backward computation

Solution, a different architecture: Bidirectional RNN



Bidirectional RNNs

Note: bidirectional RNNs are only applicable if you have access to the entire input sequence.

- They are **not** applicable to Language Modeling, because in LM you *only* have left context available.

If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).

For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.

- You will learn more about BERT later in the course!

Multi-layer RNNs

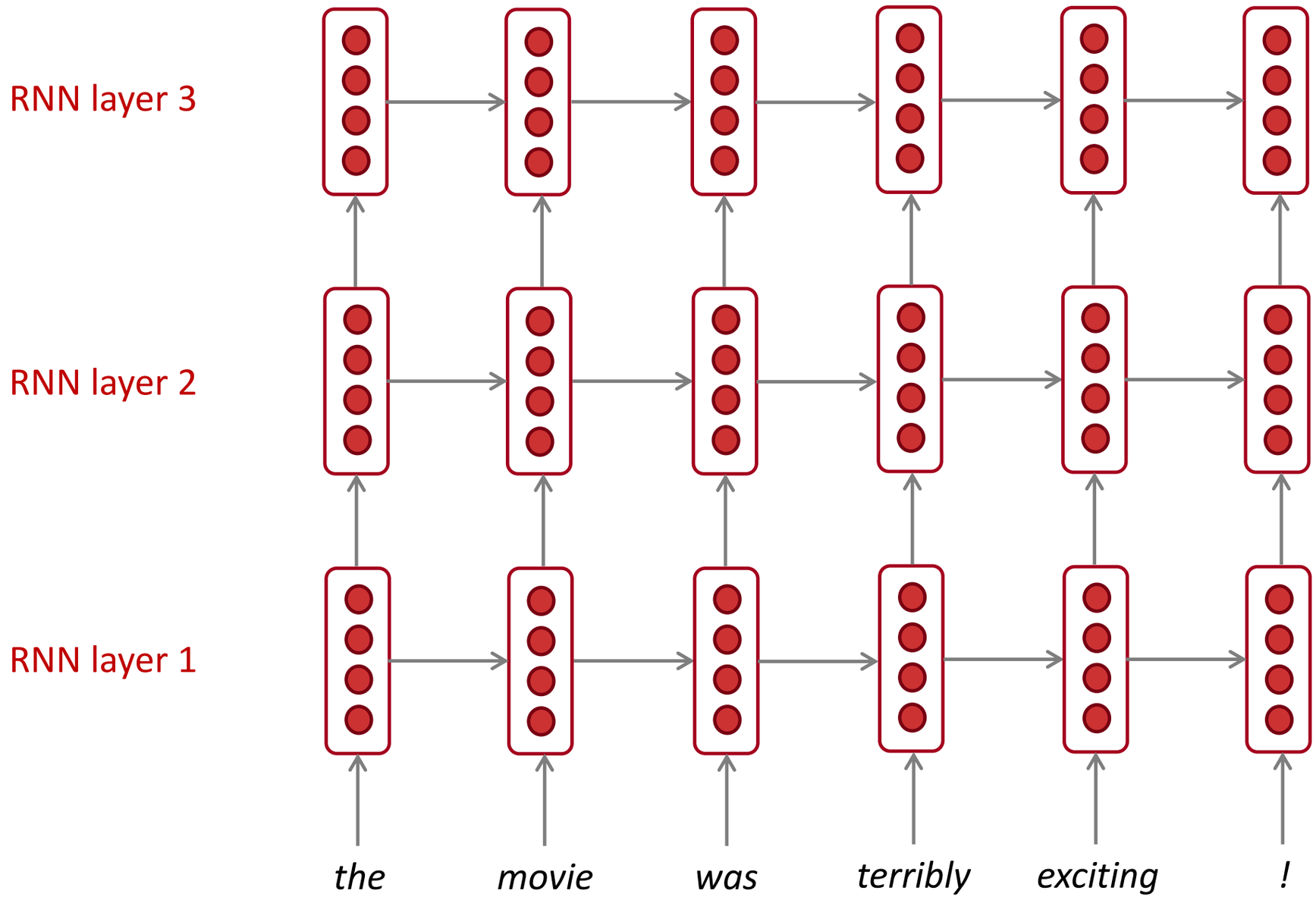
RNNs are already “deep” on one dimension (they unroll over many timesteps)

We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.

This allows the network to compute more complex representations

Deep RNNs (aka Stacked RNNs)

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*



Transformer LLMs: Llama 3 405B has **126 layers**

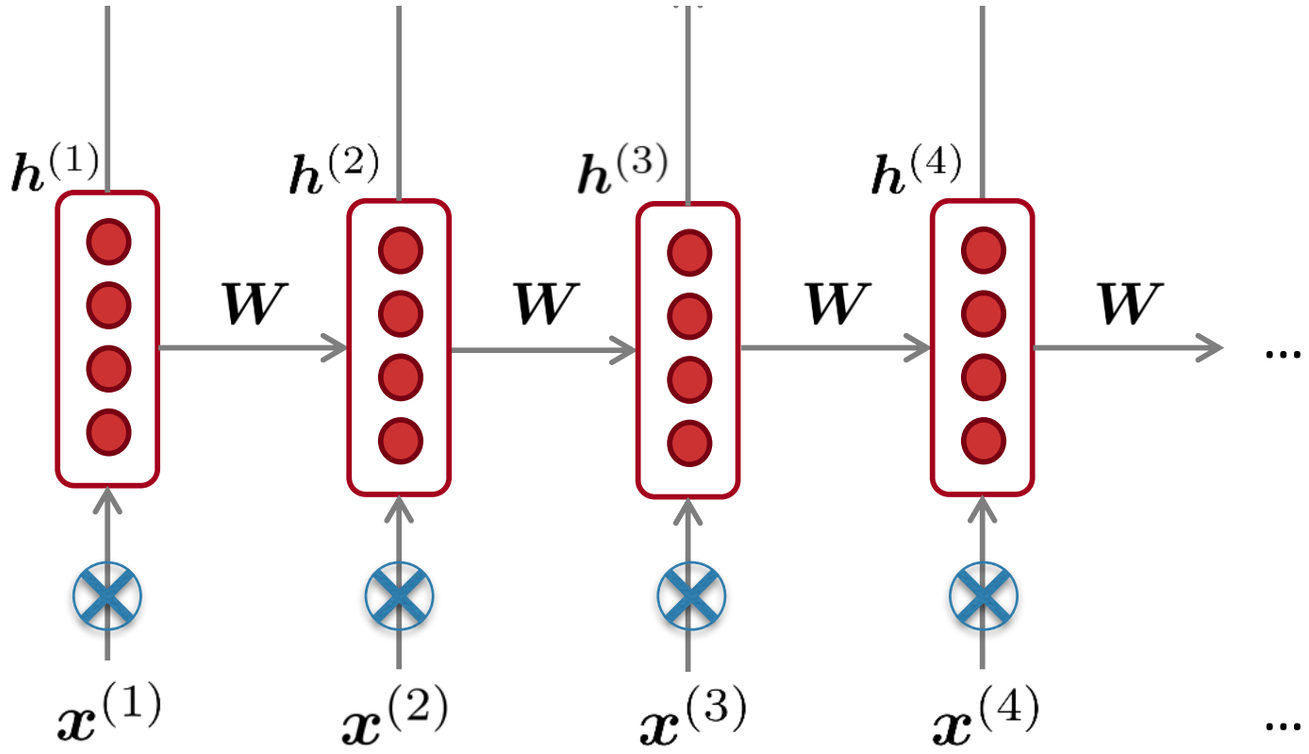
Gated-RNNs

- Simple RNNs have trouble capturing long-range dependencies

Long Short Term Memory (LSTM) [Hochreiter & Schmidhuber, 1997]

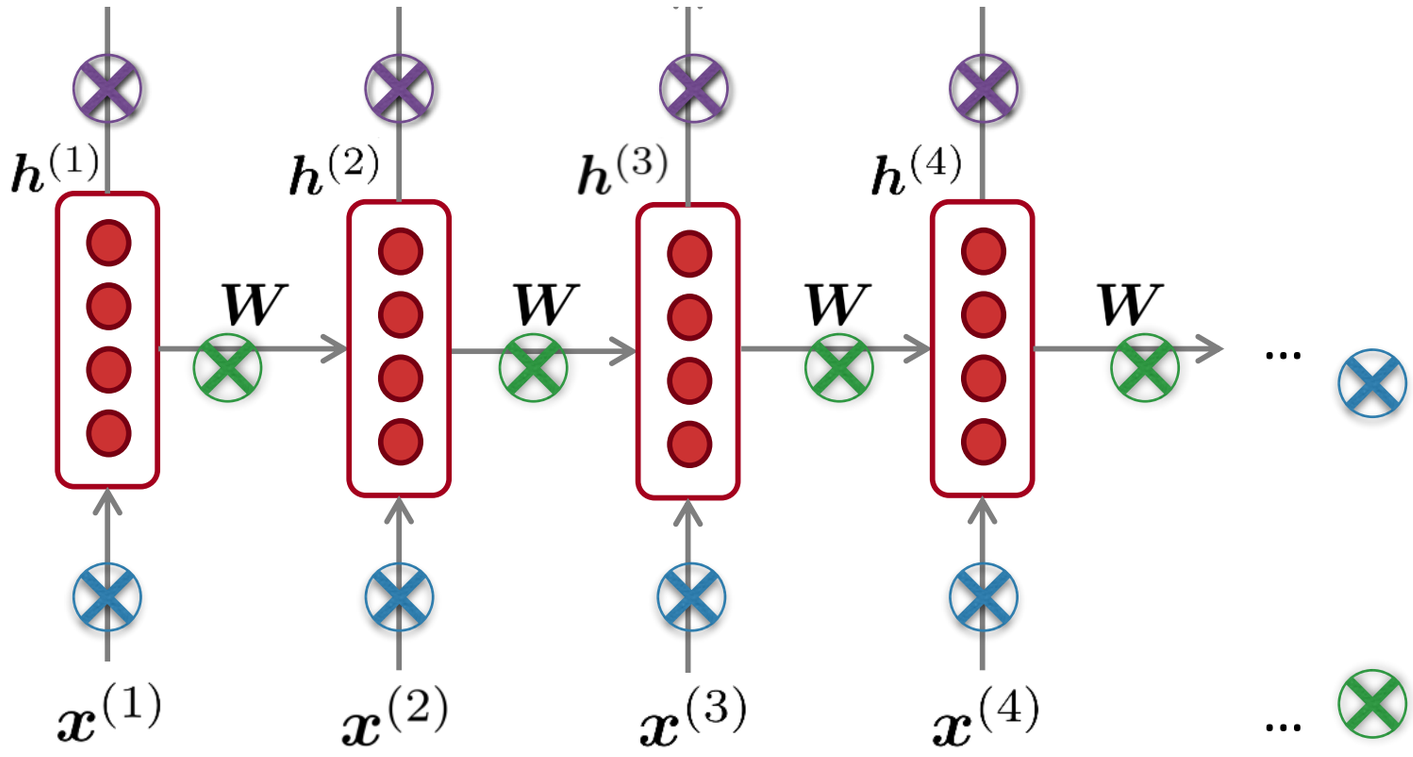
- LSTMs
  - Have special **gated structure to control memorization and forgetting in RNNs**
  - Mitigate gradient vanishing
- Facilitate long term memory
  - They have a memory that persists over longer periods of time

Long Short Term Memory (LSTM) - input gate



⊗ **Input gate:** Controls what parts of the input make it into the hidden state

LSTM - gates



Input gate: regulates the input

... **Forget gate:** regulates the hidden states

Output gate: regulates the output

Gate behavior: hard gates

- Binary vectors can act as a **'gate'** for controlling access to vectors, using **element-wise product**
- Gates are closed (0) or open (1)

$$\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}$$

h_t+1 g x_t 1-g h_t

Computing gate values

- Approximate the hard gating mechanism with a soft—but differentiable—gating mechanism
  - Gates are closed (0) or open (1) or **somewhere in between**
  - How to compute the values of the gate?

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

LSTM: hidden state, cell

On step *t*, there is a hidden state $\boldsymbol{h}^{(t)}$ *and* a cell state $\boldsymbol{c}^{(t)}$

- Both are vectors length *n*
- The cell stores long-term information
- The LSTM can erase, write and read information from the cell

The selection of which information is erased/written/read is controlled by three corresponding gates

- The gates are also vectors length *n*

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

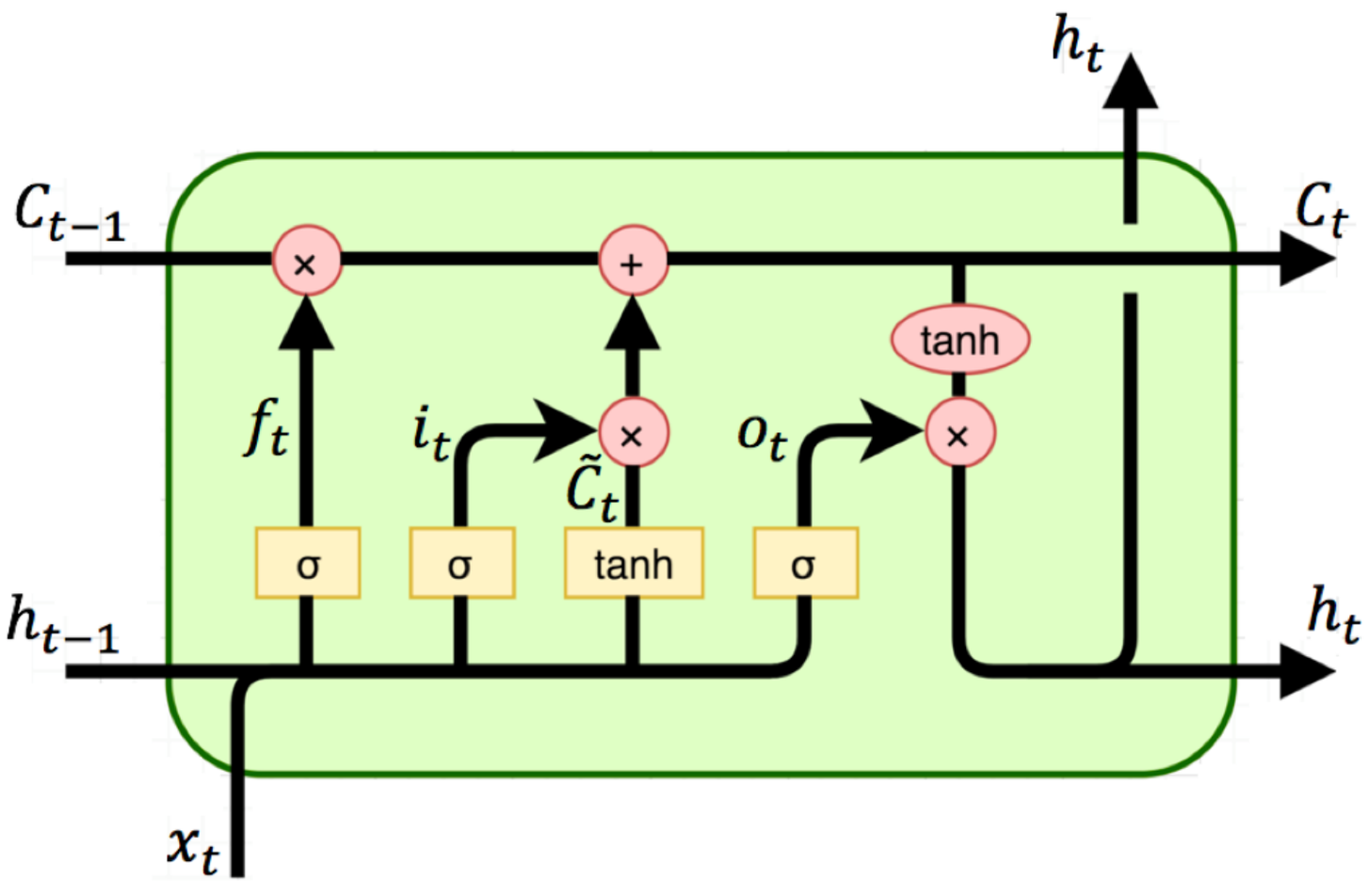
Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product



Gates mitigate gradient vanishing and exploding

- Gates make sure output does not shrink or grow too much
  - **tanh** produces values between -1 and 1
  - This impacts the gradients

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

- Control what goes into the memory and what gets forgotten
  - Facilitates some persistence for long range dependencies

LSTM: real-world success

In 2013–2015, LSTMs started achieving state-of-the-art results

- Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
- LSTMs became the dominant approach for most NLP tasks

2019– Transformers have become dominant for all tasks

Neural Machine Translation

Machine Translation

- MT is a classic test of language understanding
- It requires both language analysis and generation
- Large social/government/military needs

Evaluation of Machine Translation Systems

- What makes a good translation?

- Intuition: good translations are fluent in the target language and faithful to the original meaning

MT Evaluation: BLUE

- BLEU - bilingual evaluation understudy (Papineni et., al 2002)
  - Compare to a human-generated reference translation
  - Or, better: multiple references
  - Weighted average of n-gram precision (across different n)

"A Vinay le gusta Python"

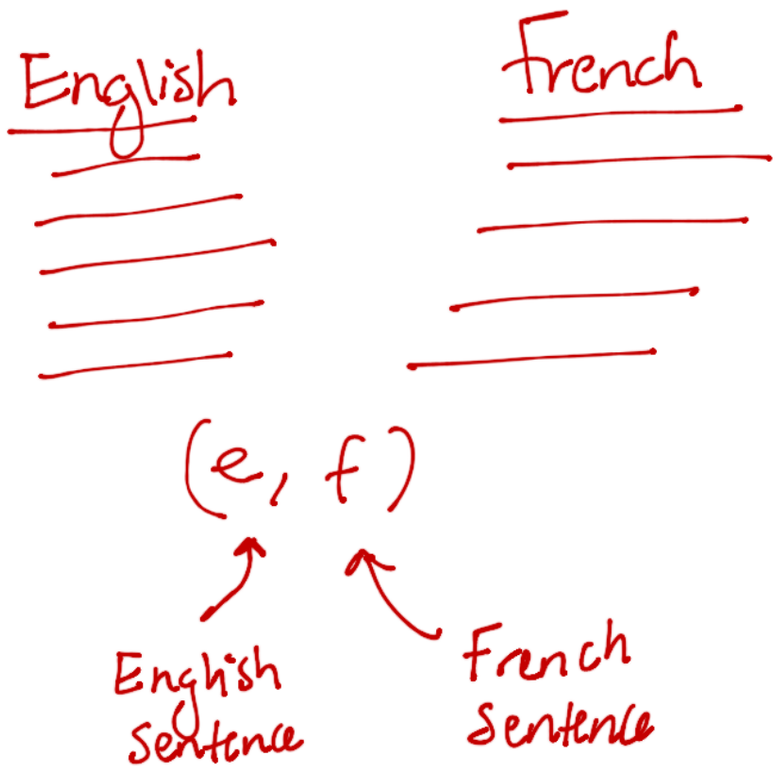
	Adequate?	Fluent?
To Vinay it like Python	yes	no
Vinay debugs memory leaks	no	yes
Vinay likes Python	yes	yes

Machine Translation data

- Basic idea: Parallel corpora are available in several language pairs

– treat translation as a supervised learning problem.

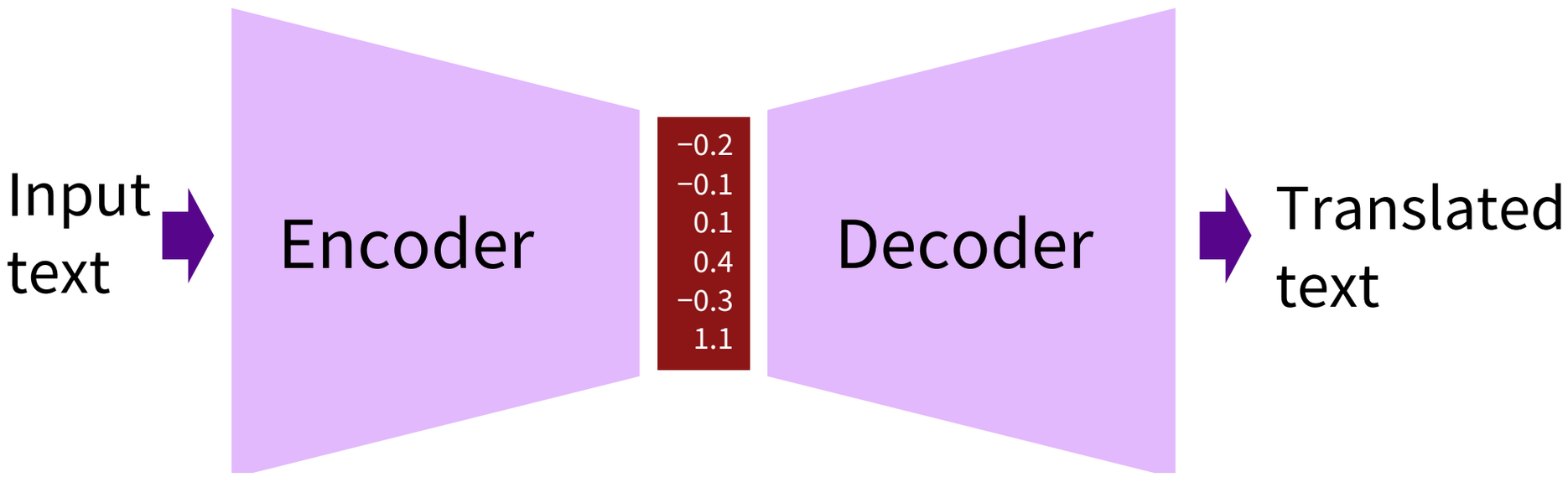
- Where does the data come from?
  - Canadian Parliament Proceedings
  - European Parliament
  - Bilingual Newspapers
  - Book translations
  - Movie Subtitles
  - TED talks



Neural MT

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

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



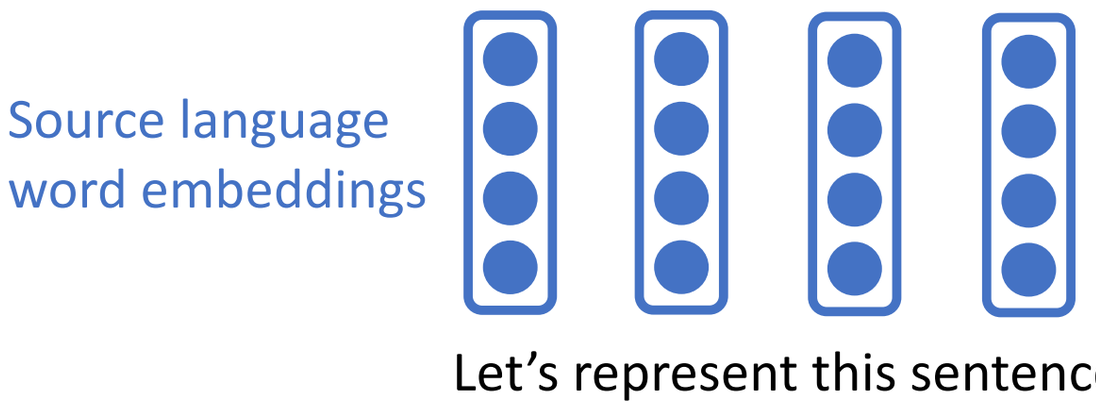
The Sequence-to-sequence model

German: Representieren wir diesen Satz.

English: Let’s represent this sentence.

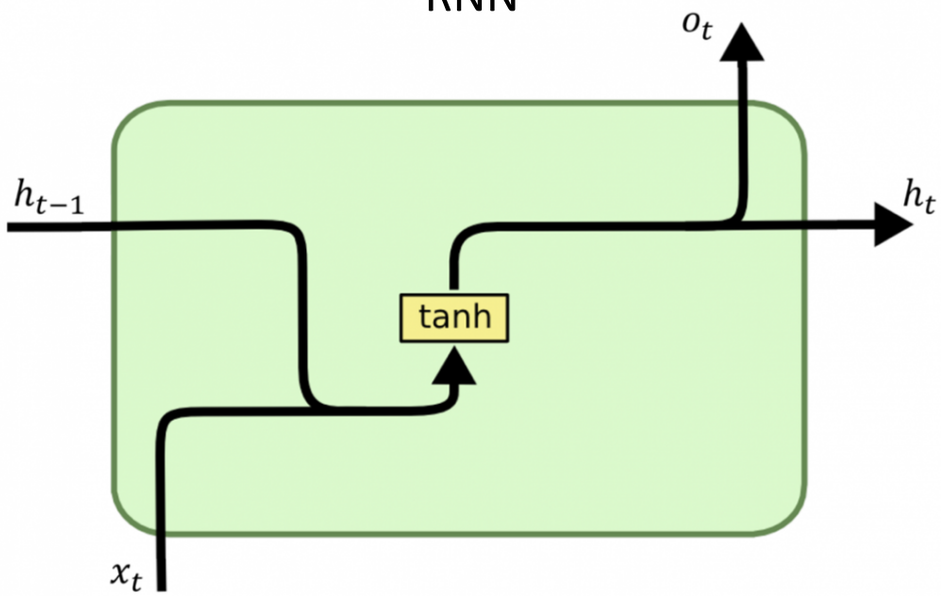
The Sequence-to-sequence model

- Look up the embedding of each source word

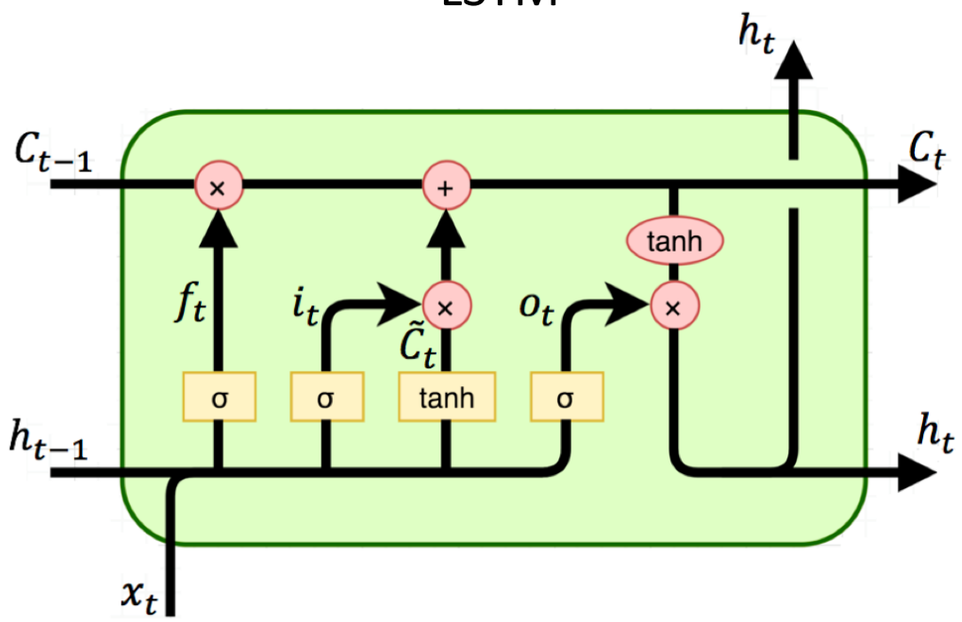


Seq2Seq: first hidden state

RNN



LSTM

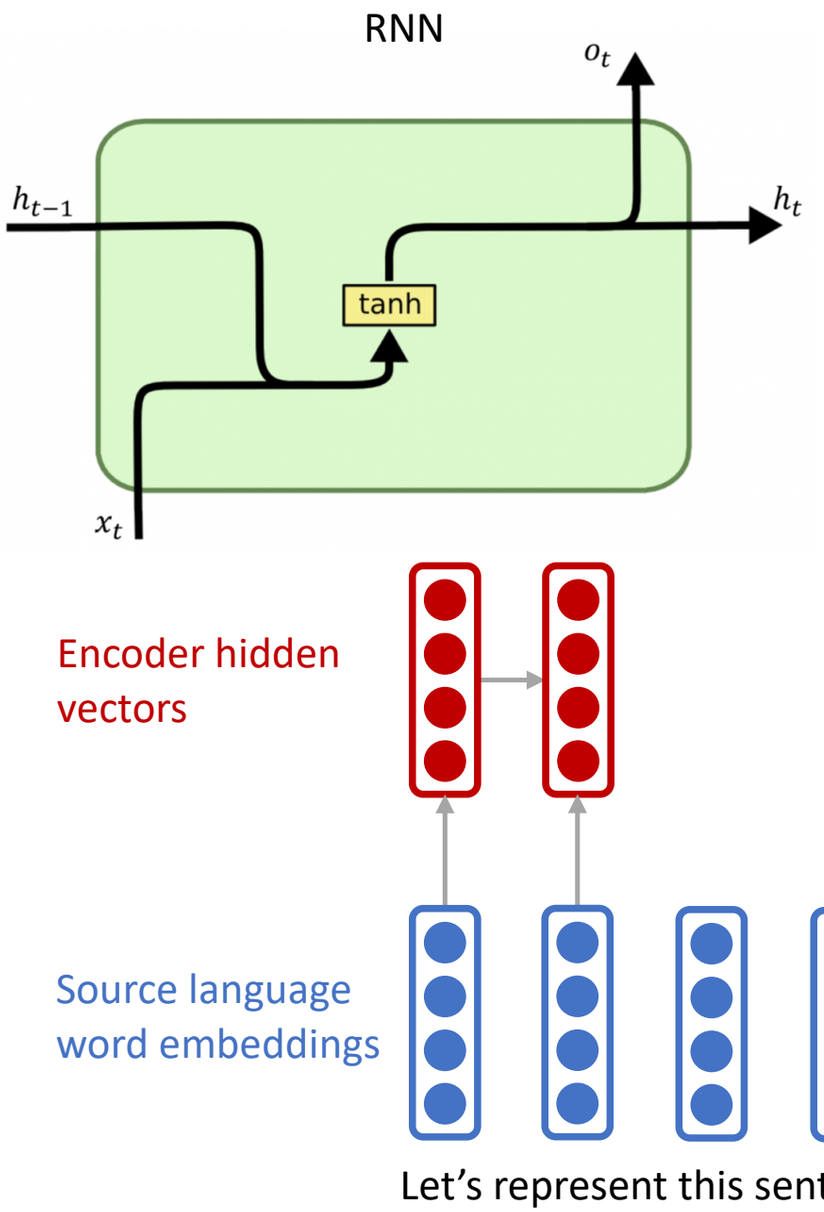


Encoder hidden vectors

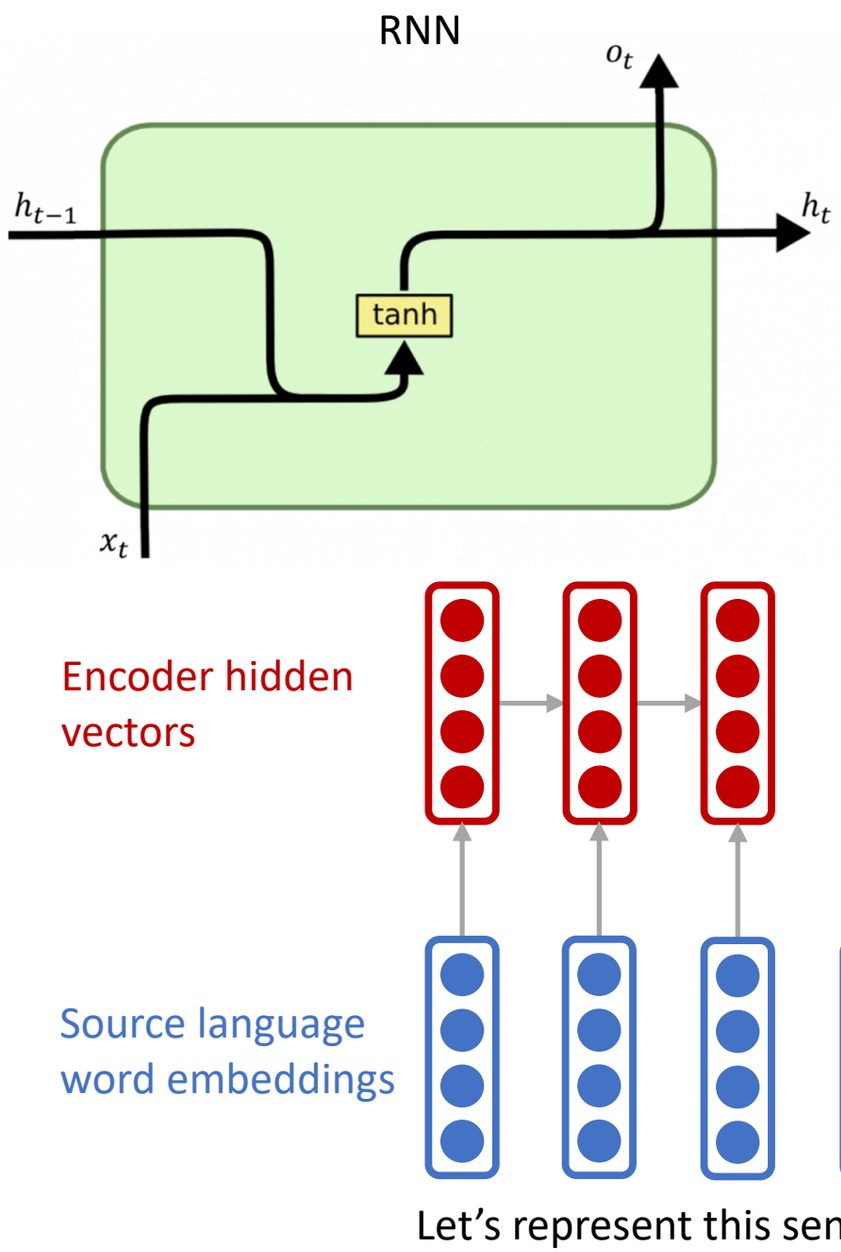
Source language word embeddings

Let's represent this sentence

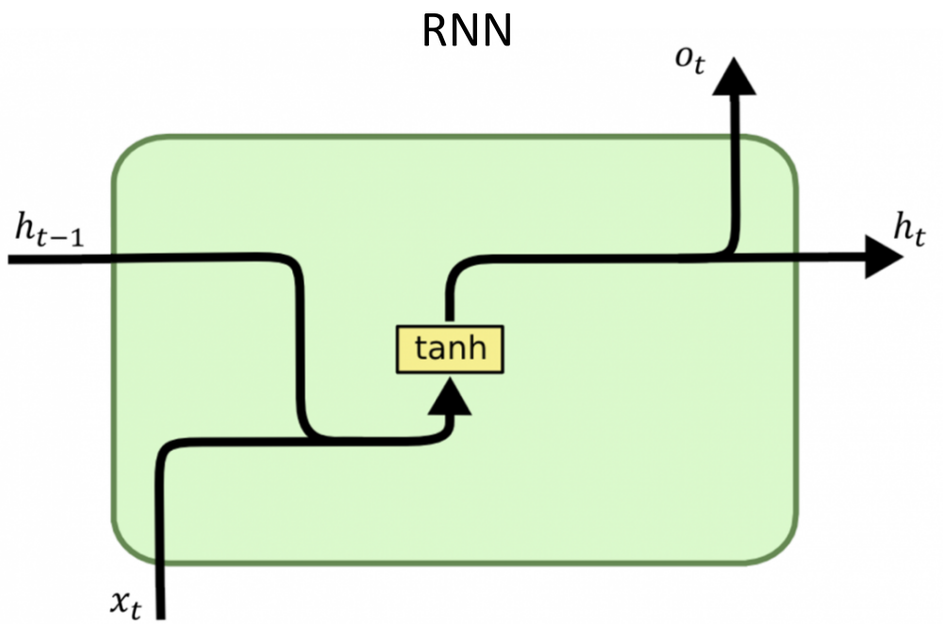
Seq2Seq: second encoder hidden state

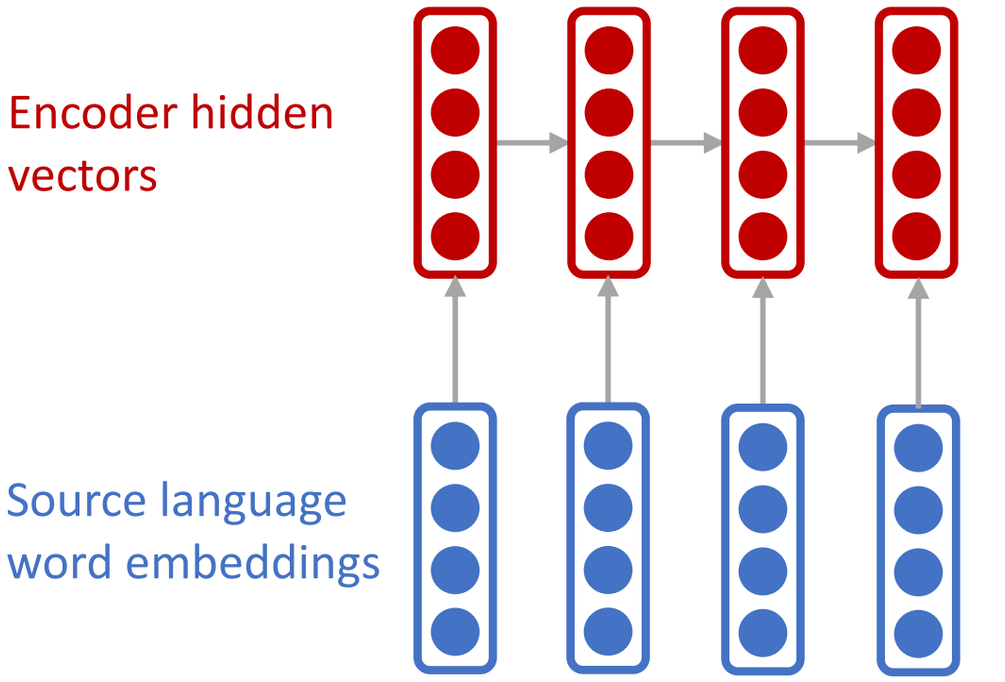


Seq2Seq: third encoder hidden state



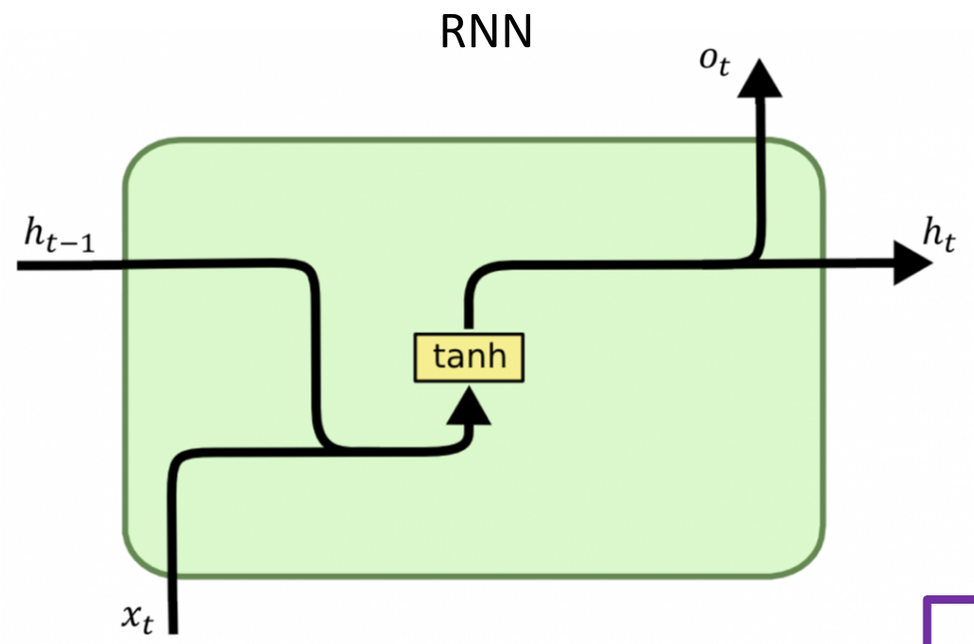
Seq2Seq: encoder hidden states





Let's represent this sentence

Seq2Seq: last hidden summaries the source sentence



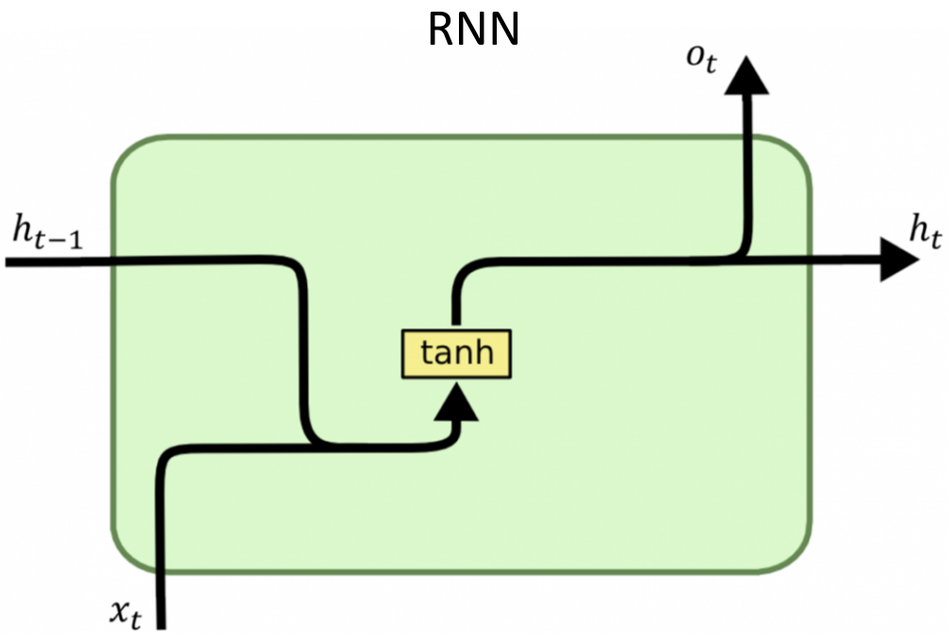
Vector encoding of the source sentence.

Encoder hidden vectors

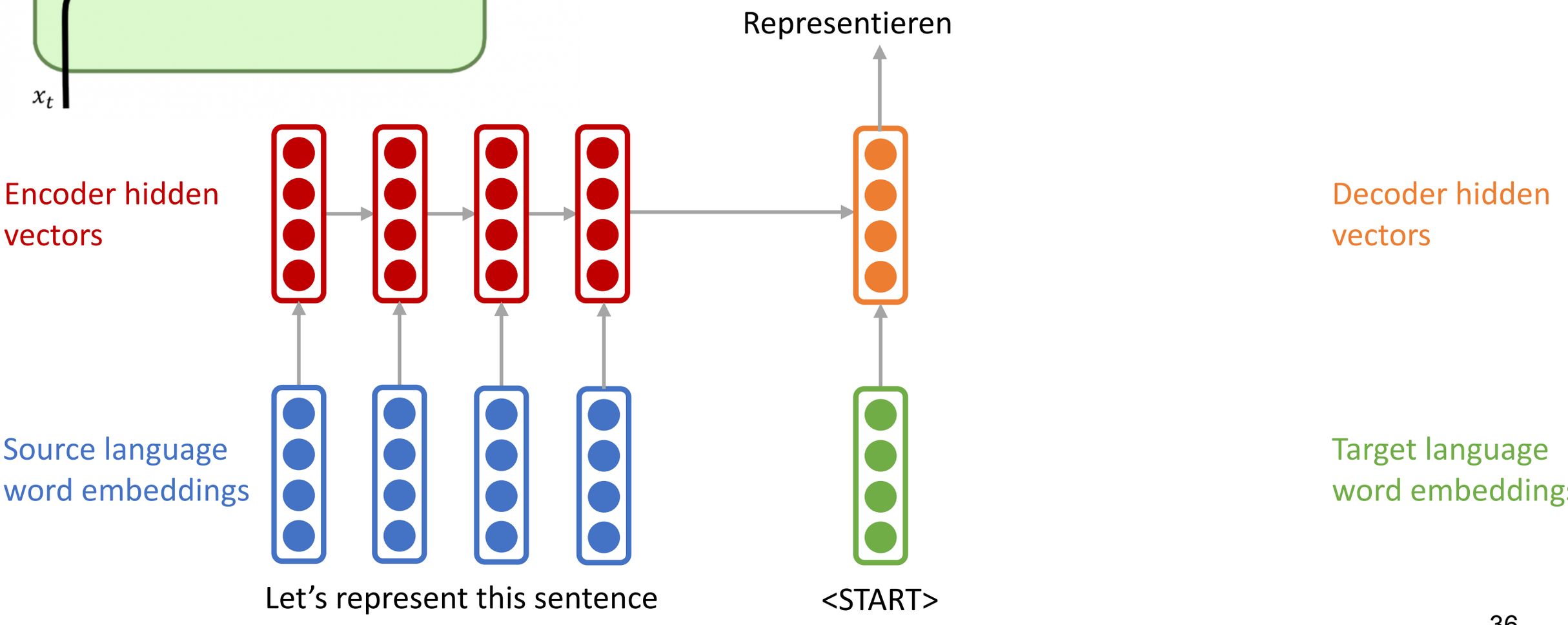
Source language word embeddings

Let's represent this sentence

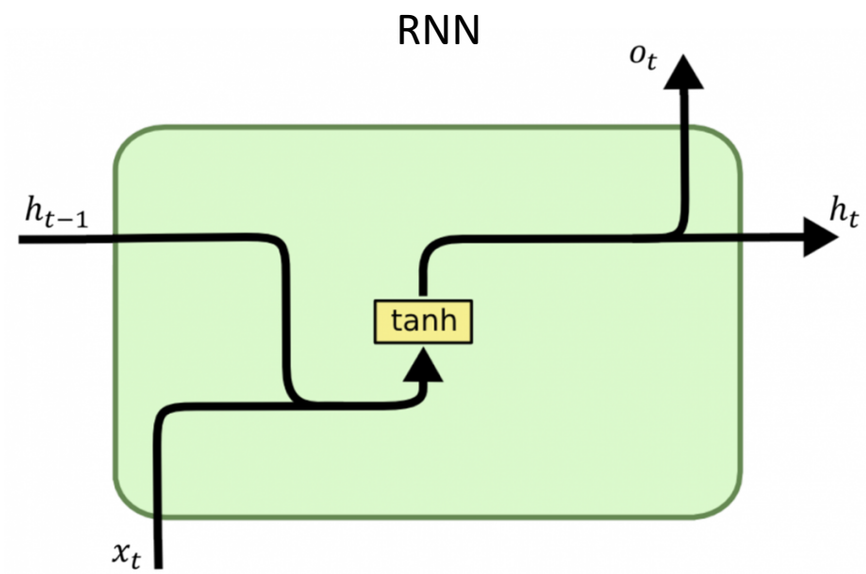
Seq2Seq: Decoder RNN

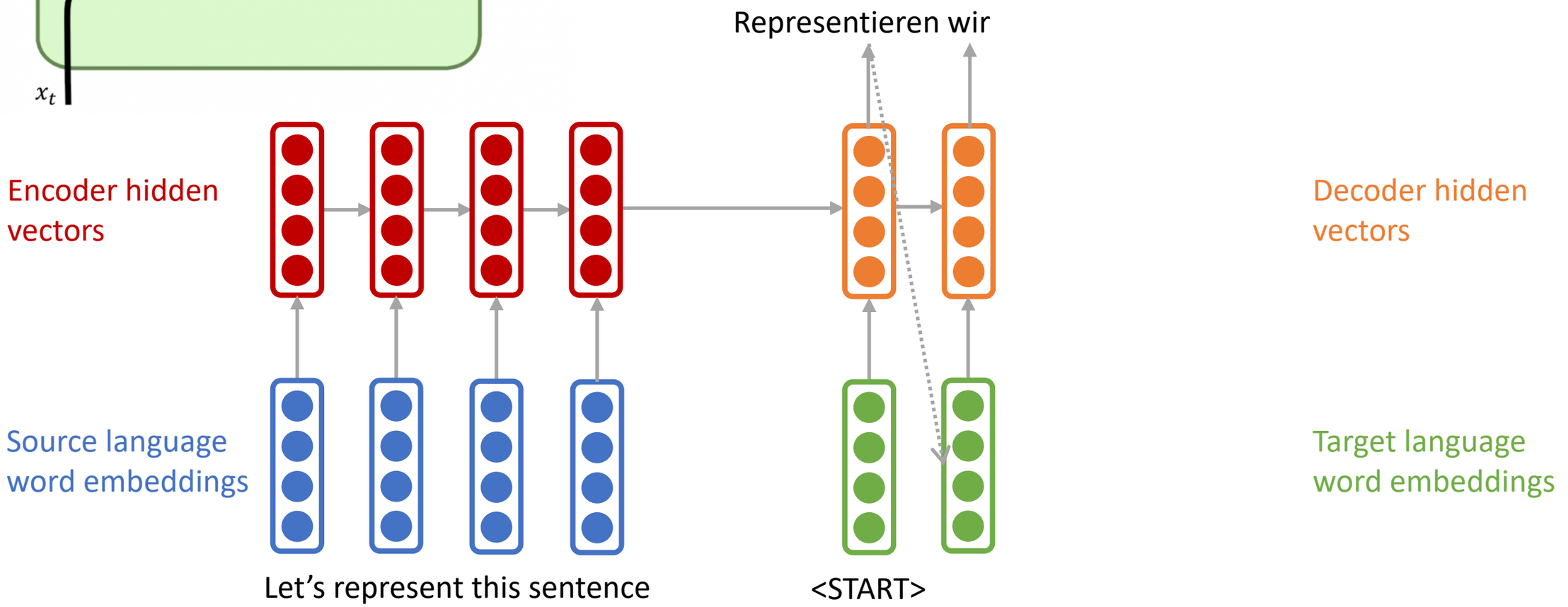


Output function reads the state vector at each time step,
and generates a word (actually generates a distribution over words)

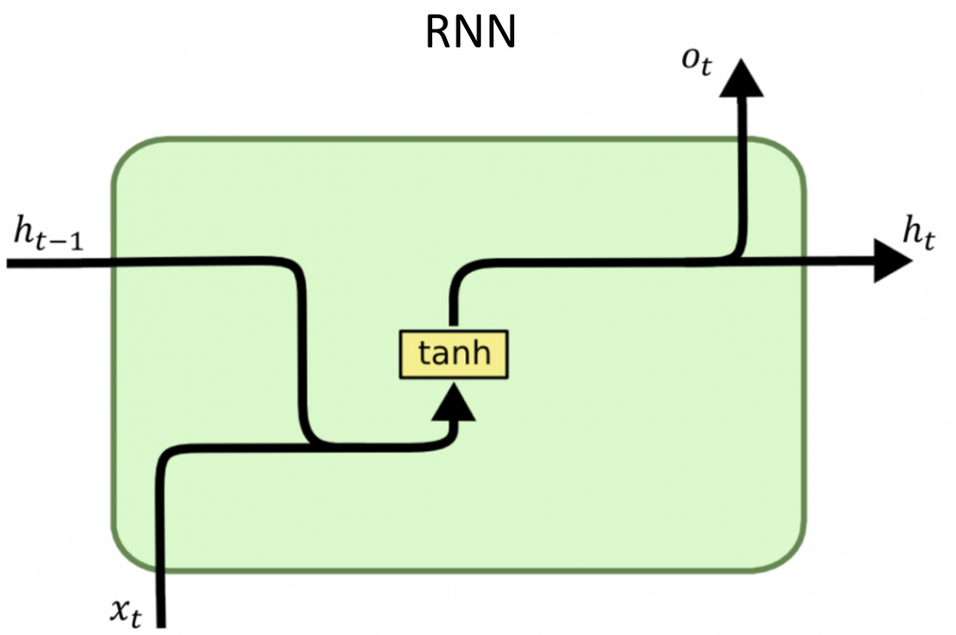


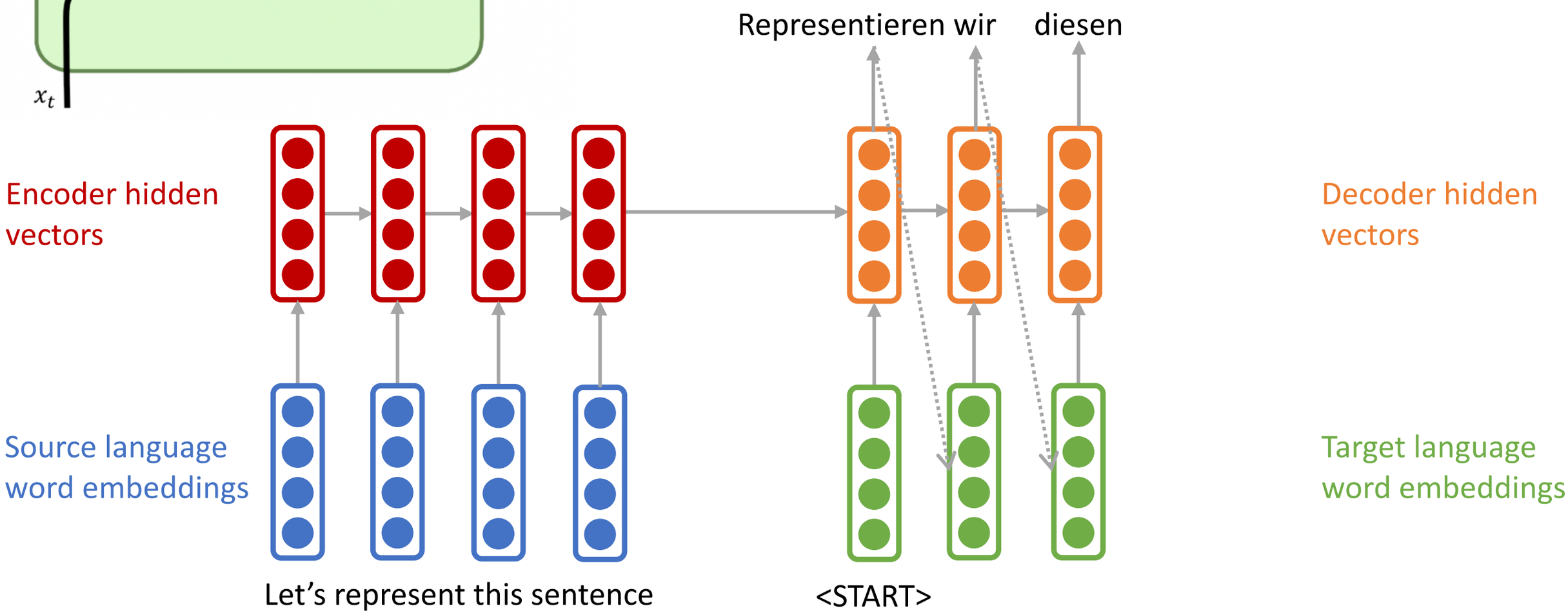
Seq2Seq: Decoder RNN



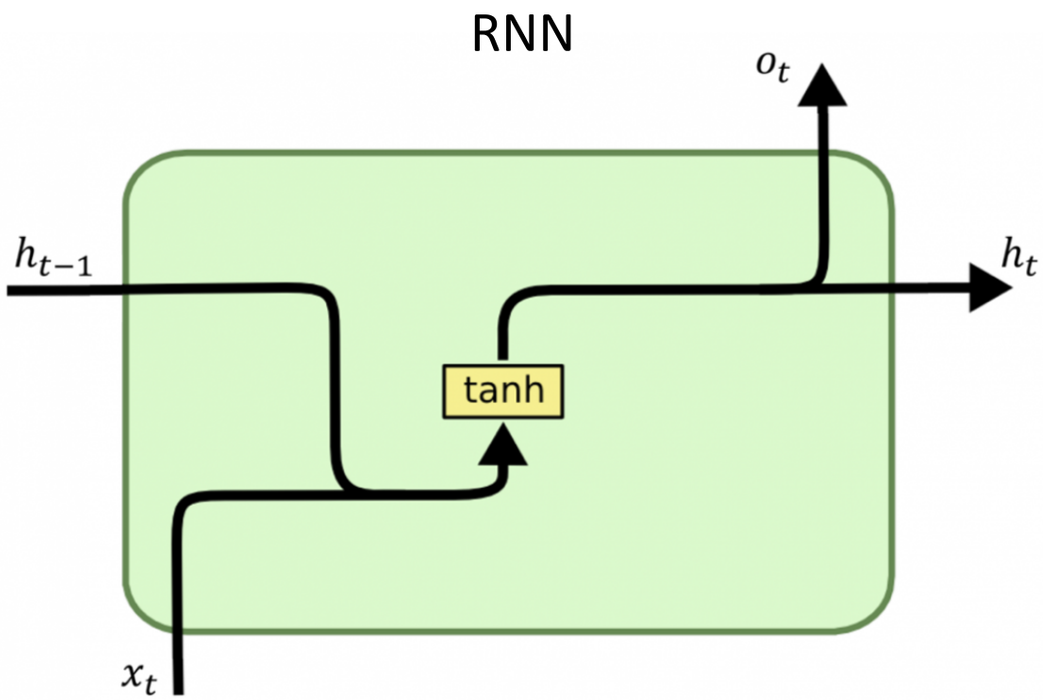


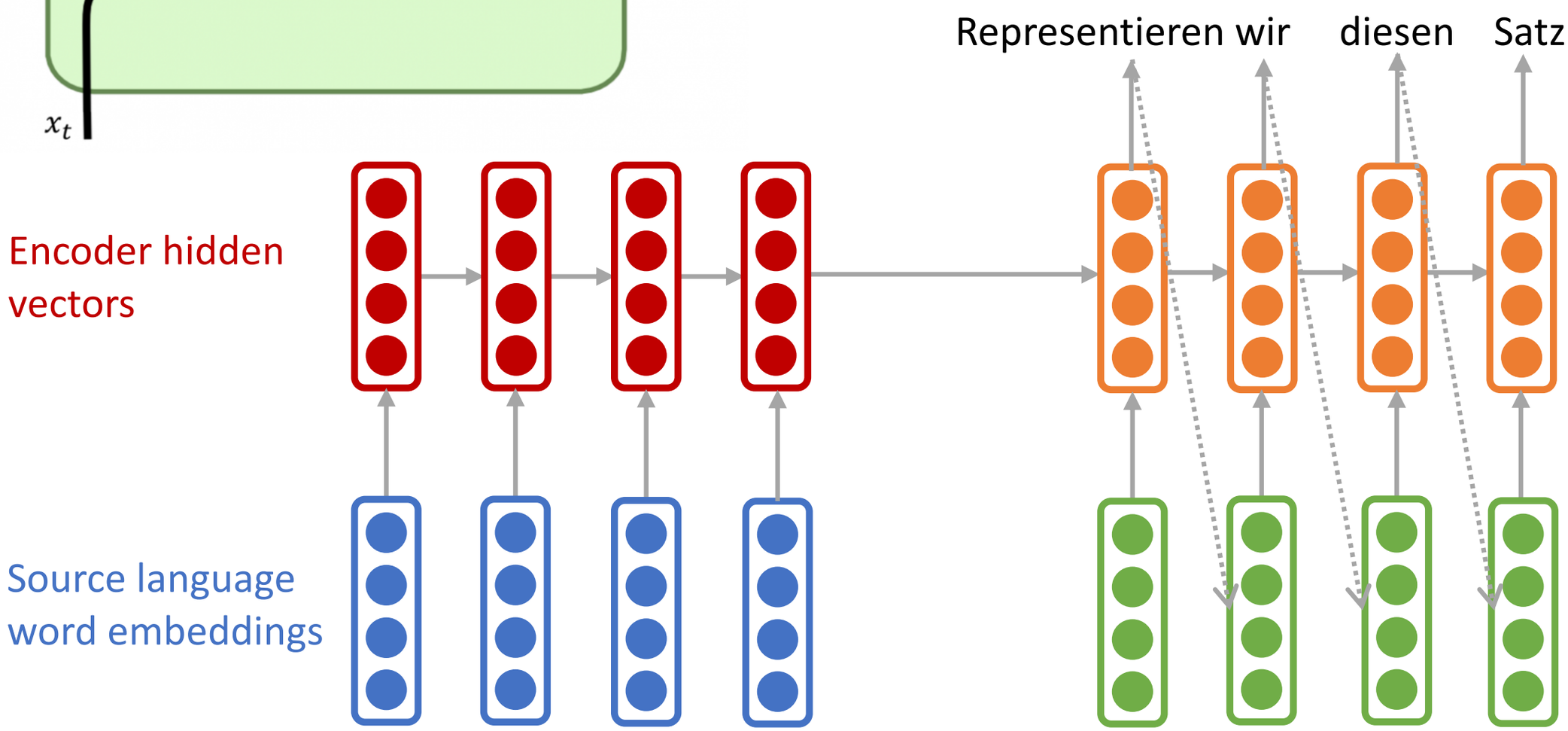
Seq2Seq: Decoder RNN



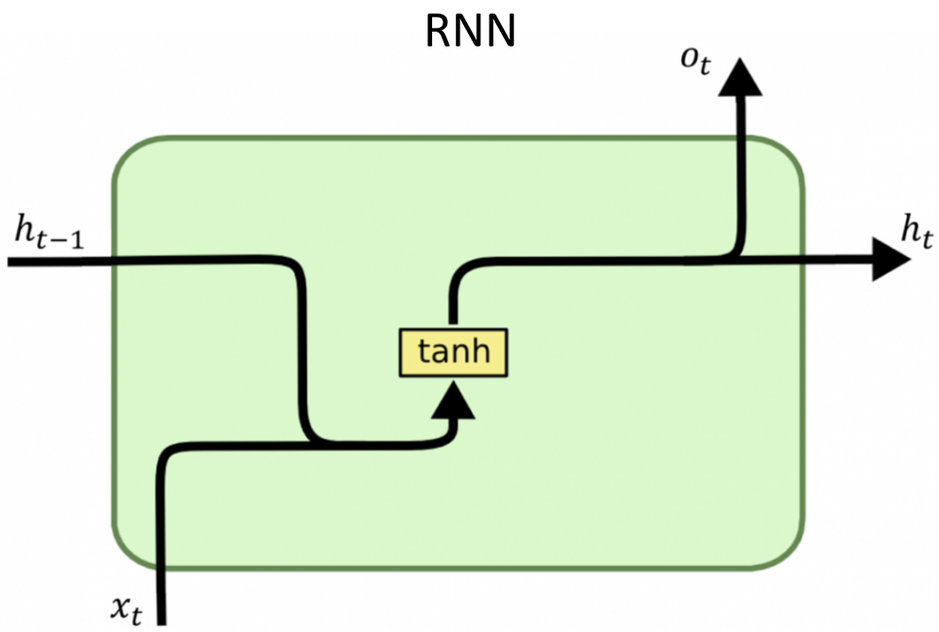


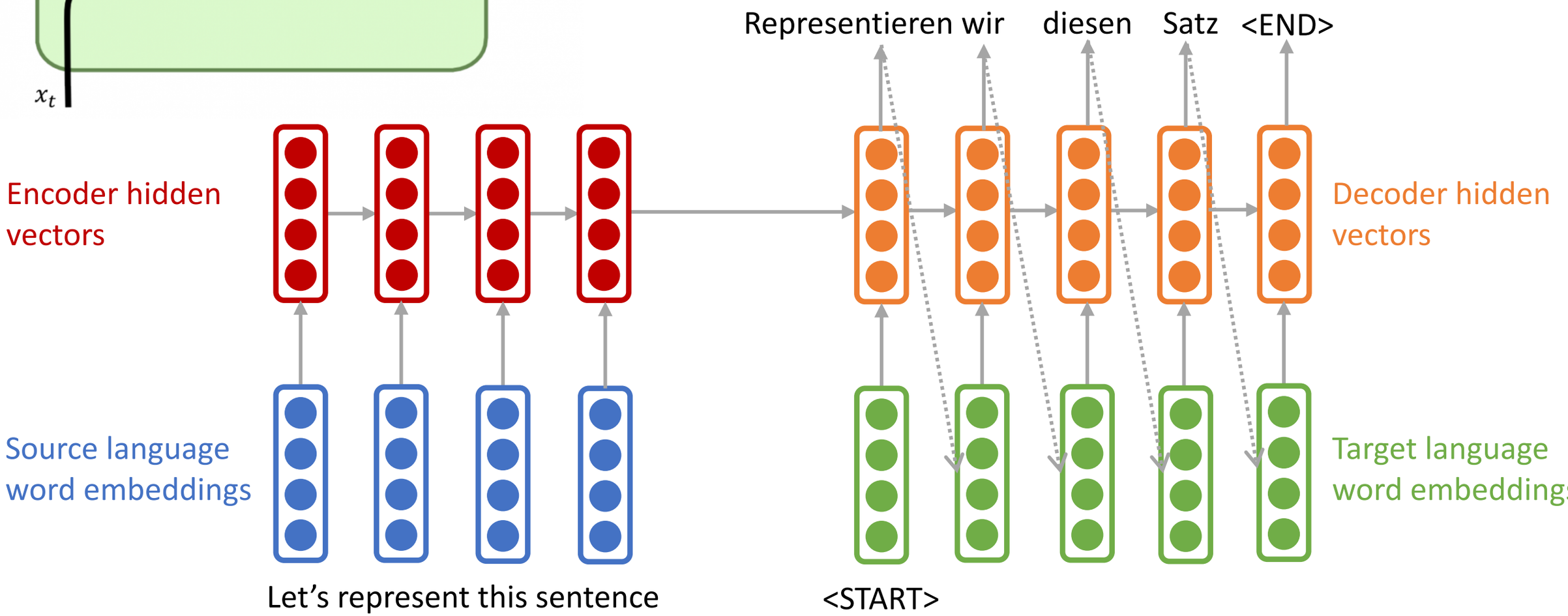
Seq2Seq: Decoder RNN





Seq2Seq: Decoder RNN - emits END token

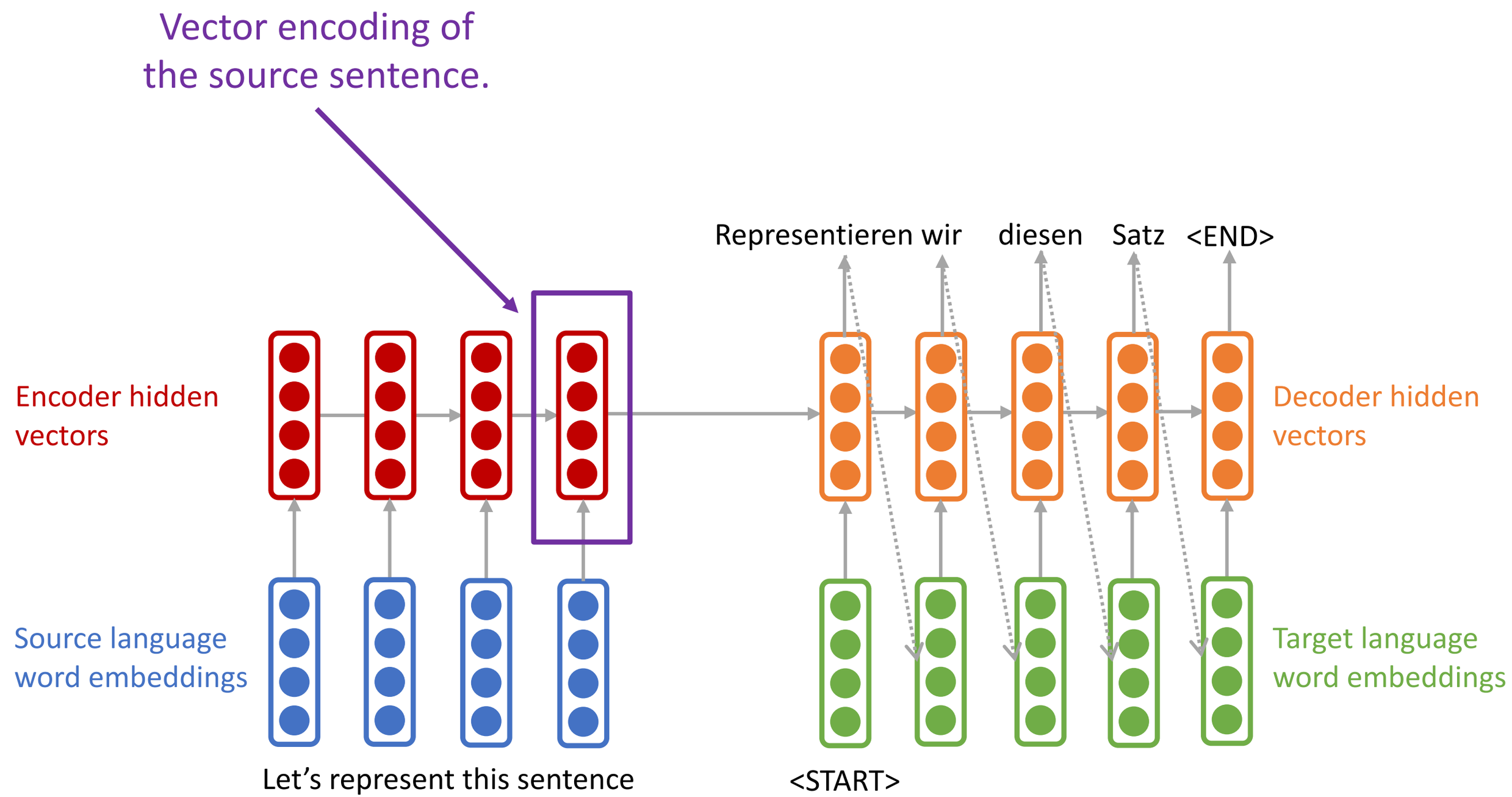




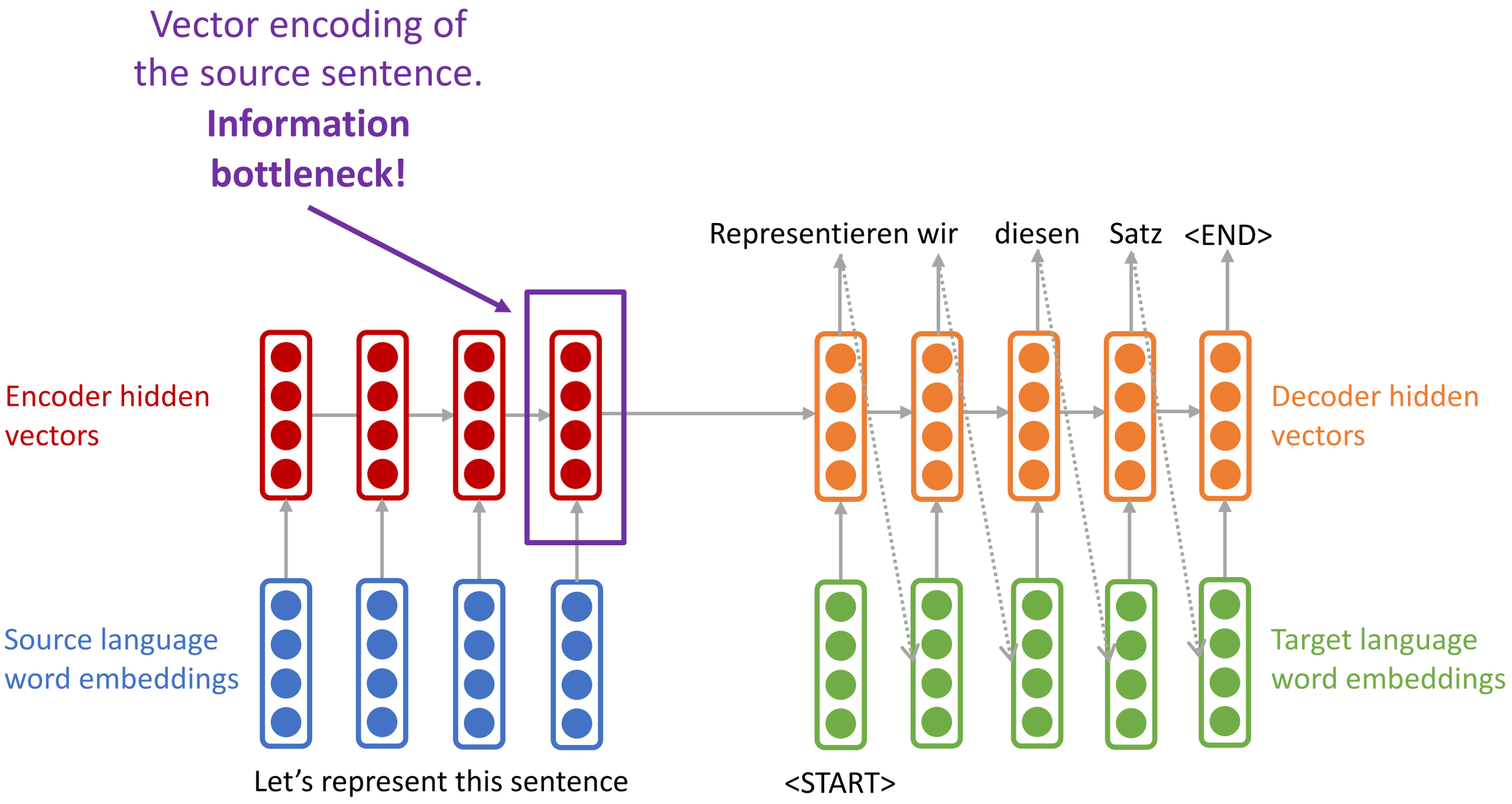
Vanilla Sequence-to-Sequence

- One look at the summary of the source sentence for the duration of translation

Seq2Seq: one summary of the source sentence



Seq2Seq: one summary == information bottleneck



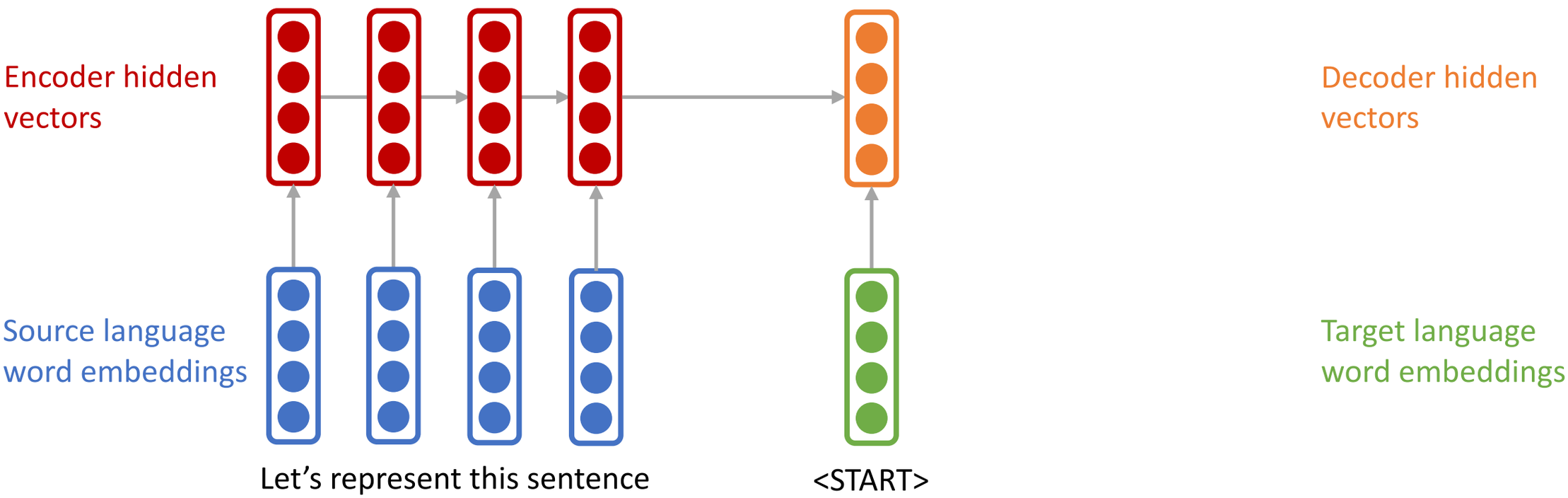
Sequence-to-Sequence with Attention

- May want to **look back at the source sentence from time to time** as we generate the translation

- Attention will allow us to **align** each output word with relevant input words

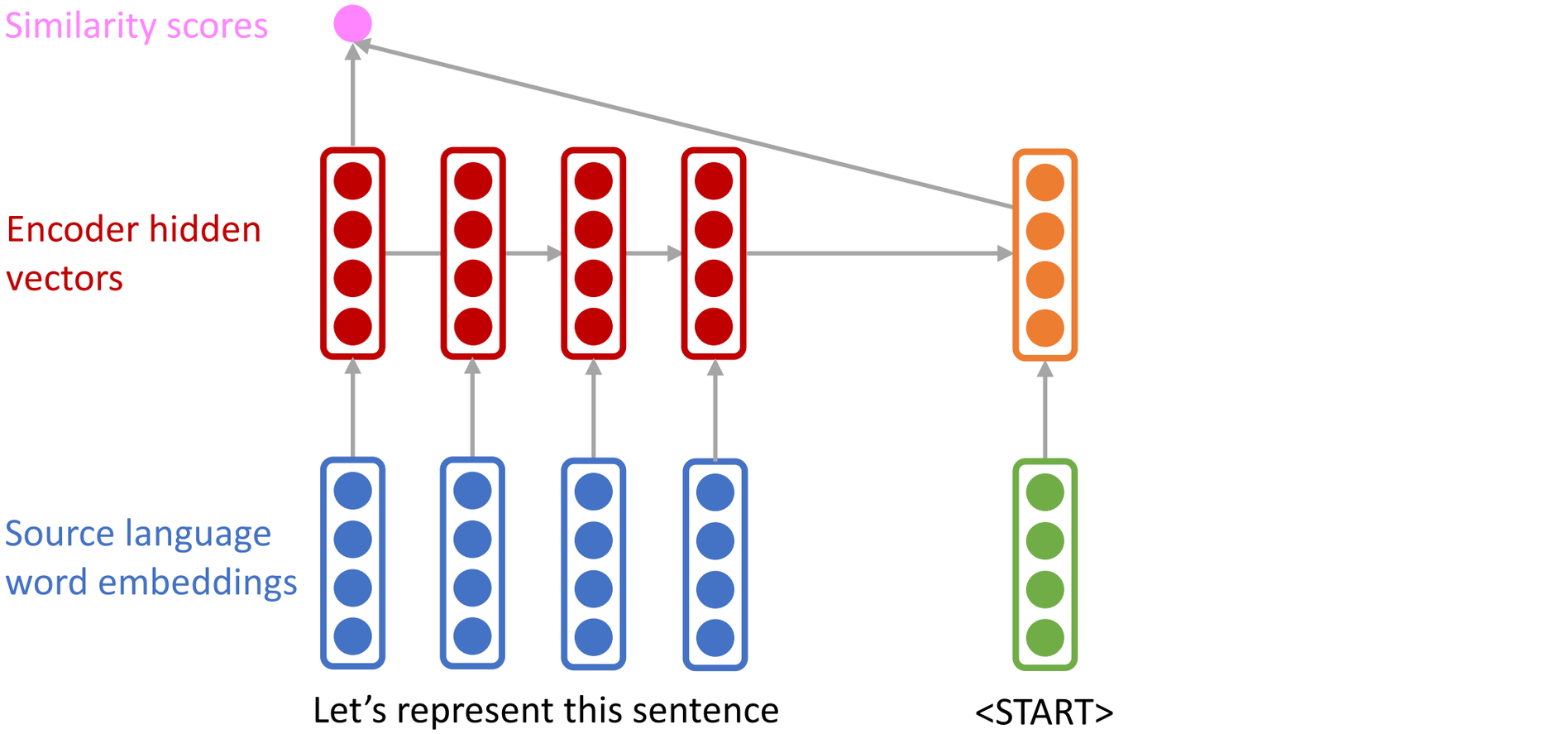
Seq2Seq with Attention

Generate 1st decoder hidden vector as before.



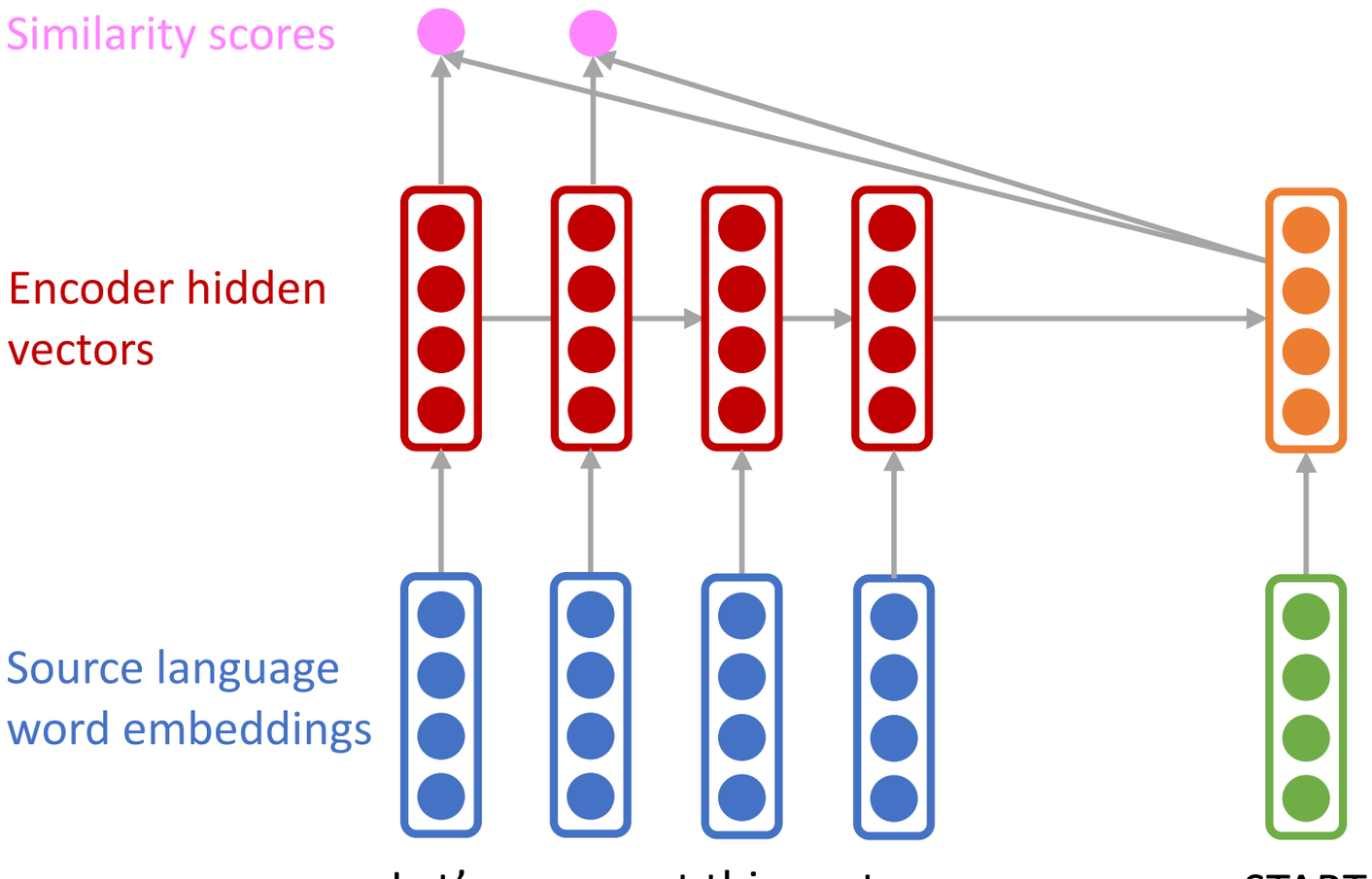
Seq2Seq with Attention: similarity scores

Calculate a **similarity score** between the decoder hidden vector and all the encoder hidden vectors.



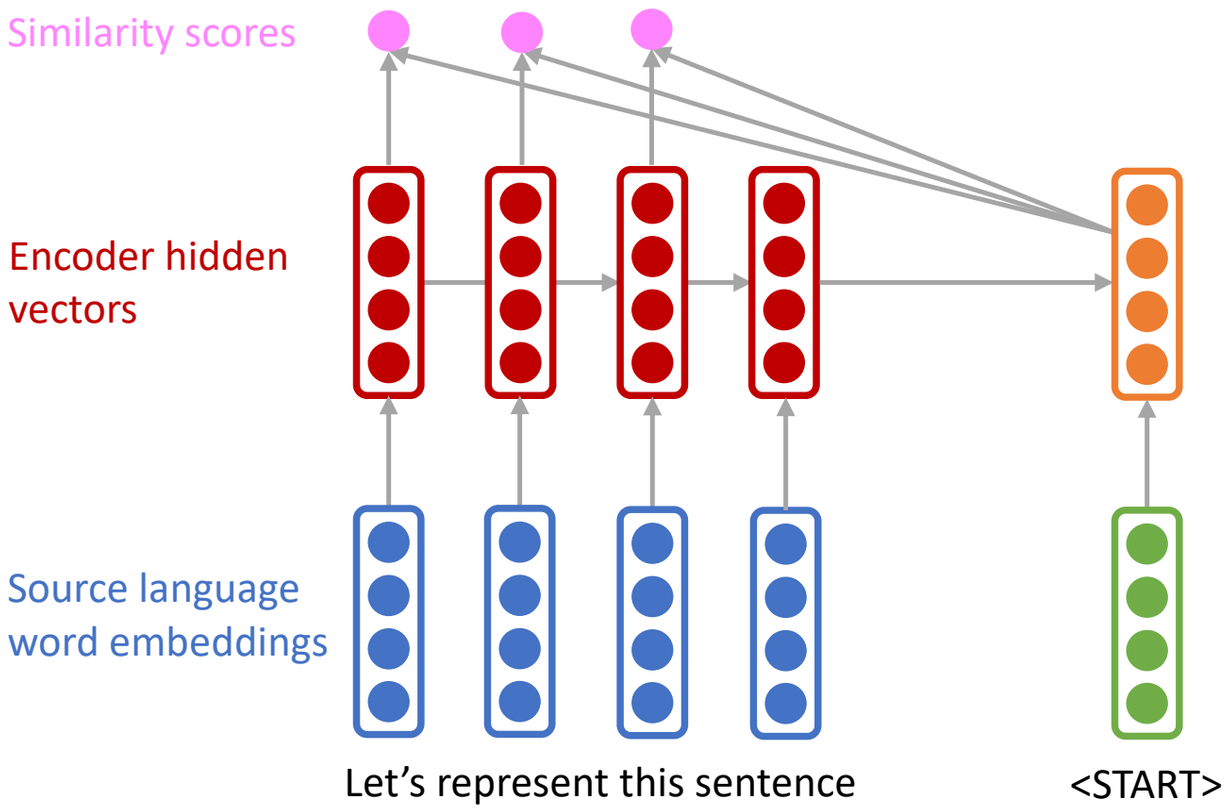
Seq2Seq with Attention: similarity scores

Calculate a **similarity score** between the decoder hidden vector and all the encoder hidden vectors.



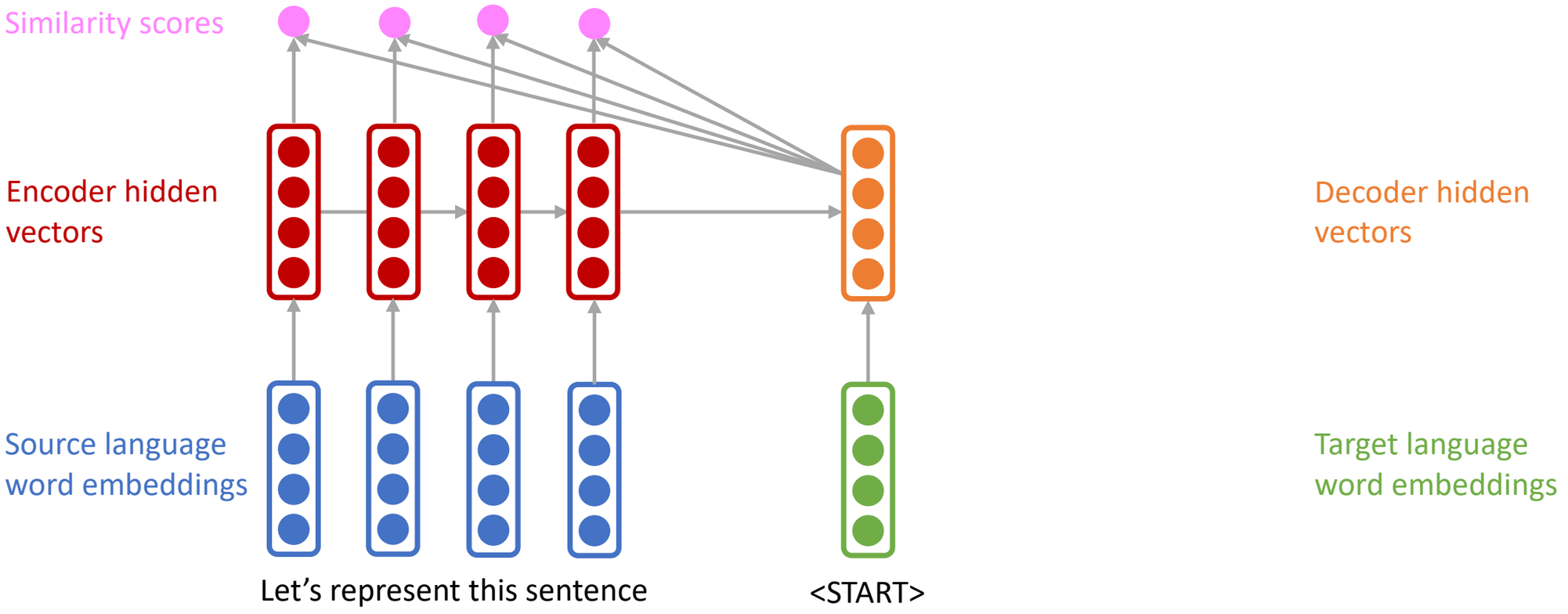
Seq2Seq with Attention: similarity scores

Calculate a **similarity score** between the decoder hidden vector and all the encoder hidden vectors.



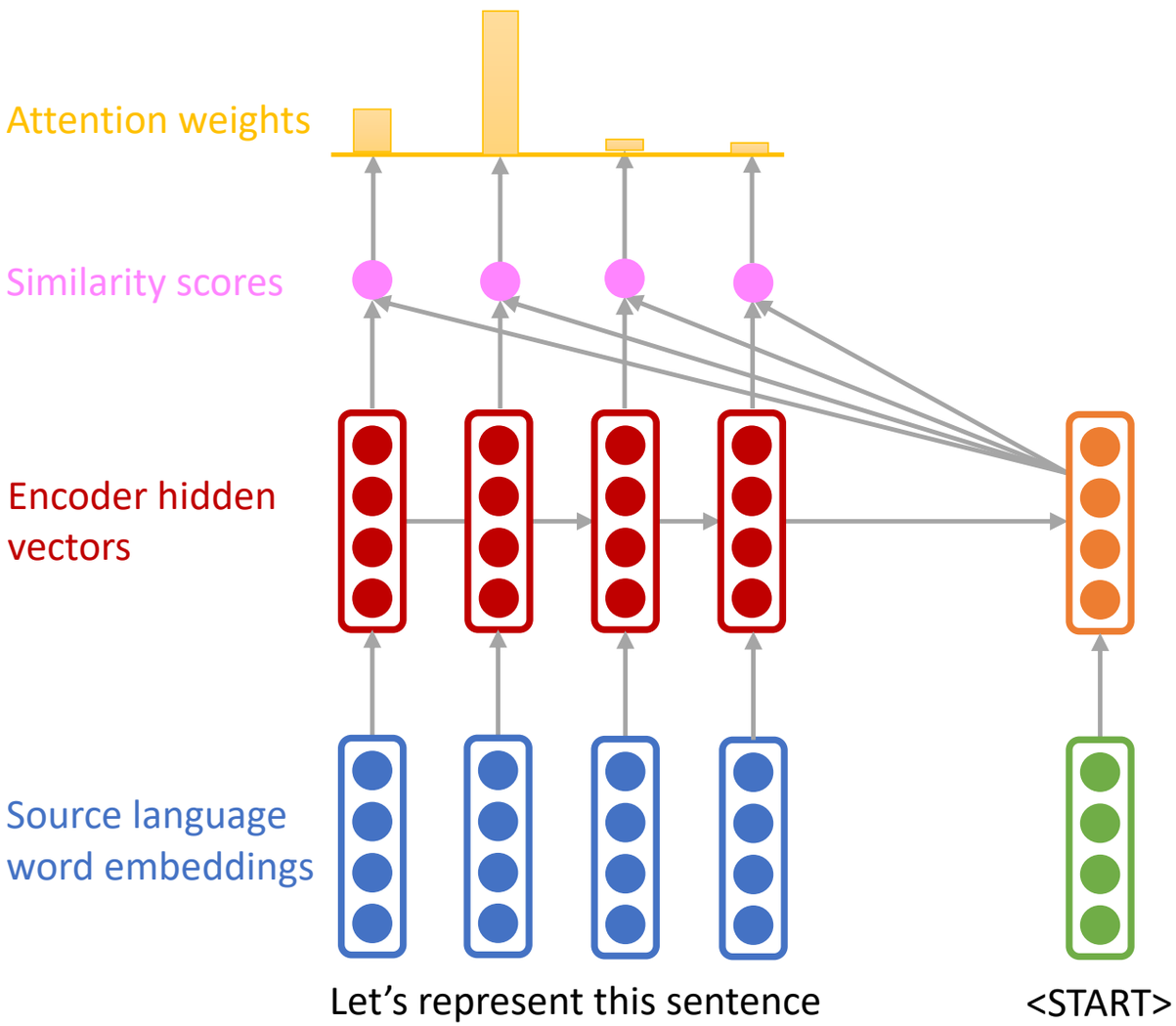
Seq2Seq with Attention: similarity scores

Calculate a **similarity score** between the decoder hidden vector and all the encoder hidden vectors.

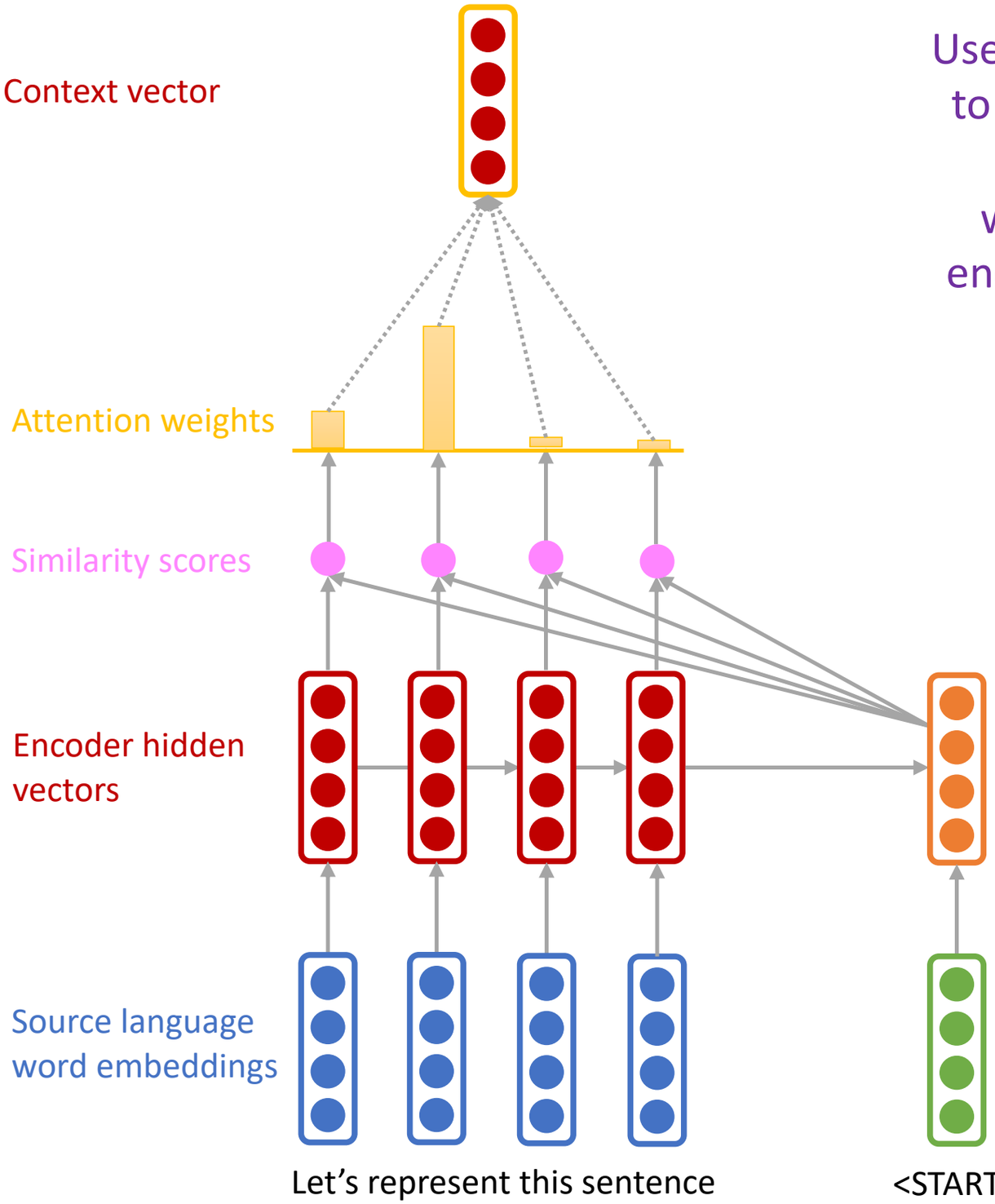


Seq2Seq with Attention: softmax similarity scores to get attention weight

Pass similarity score vector through a softmax to generate **attention weights**.

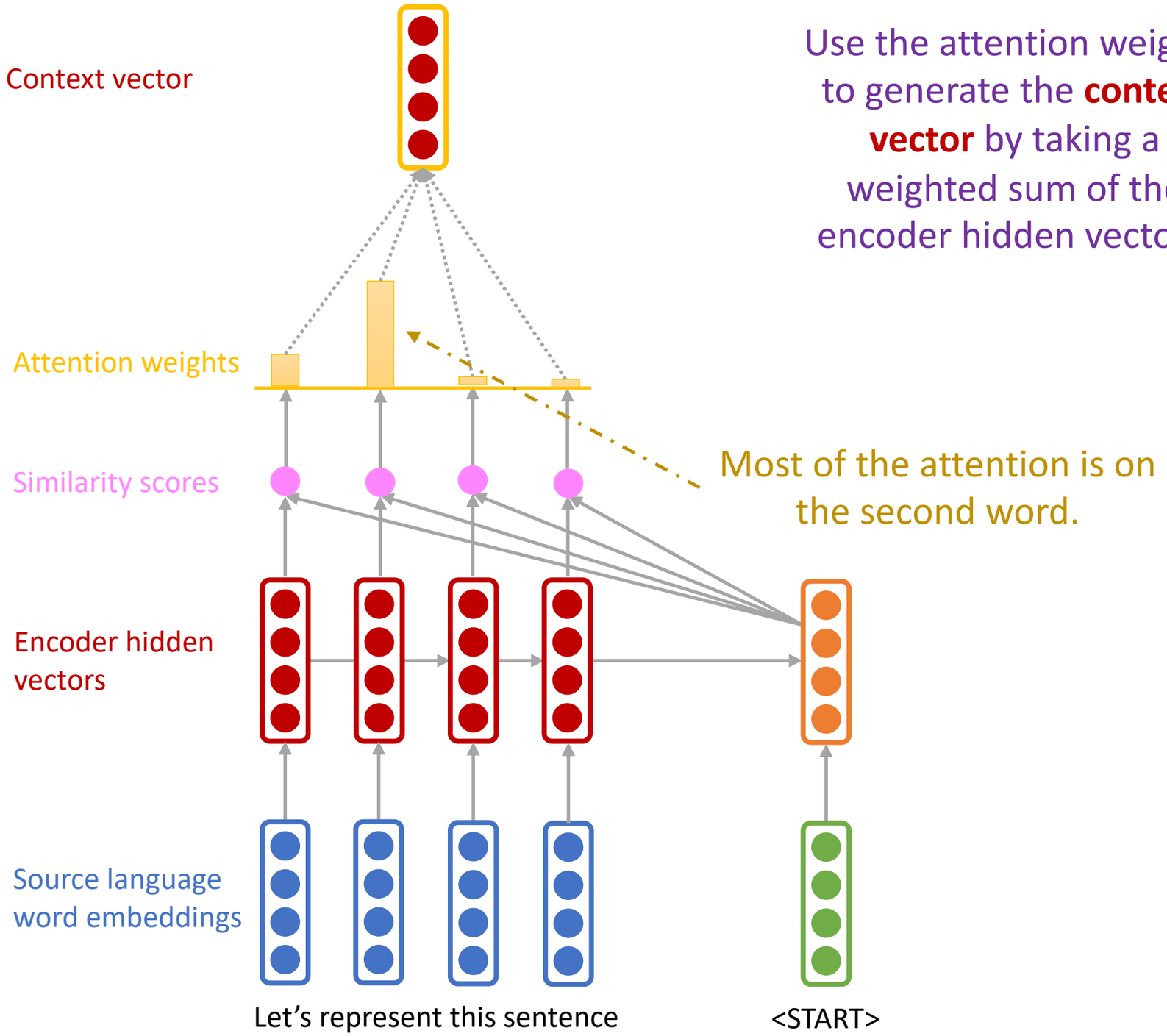


Selective summary of relevant parts of the source sentence



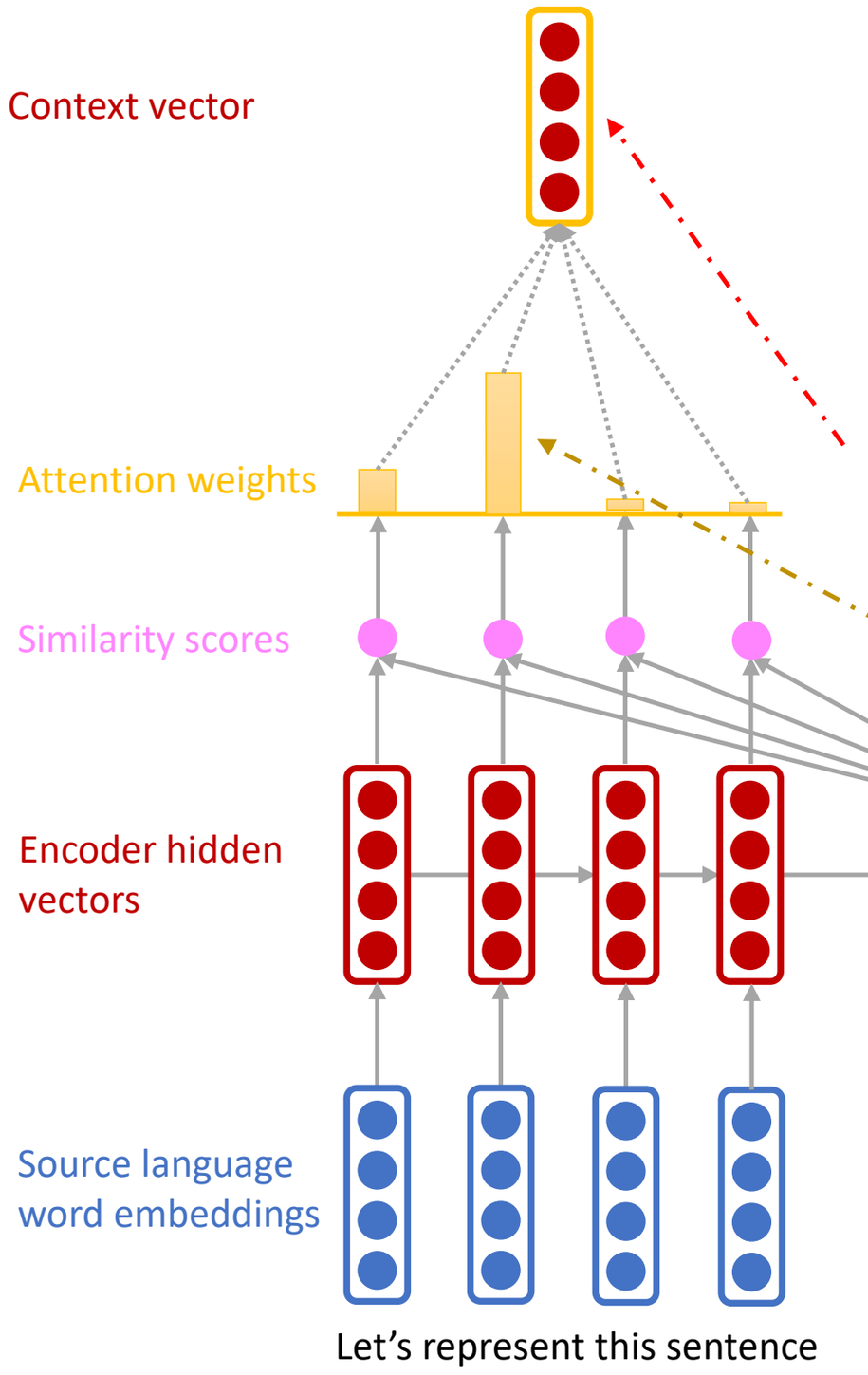
Use the attention weights to generate the **context vector** by taking a weighted sum of the encoder hidden vectors.

Softmax will provide a higher probability whenever there is an alignment



Use the attention weights to generate the **context vector** by taking a weighted sum of the encoder hidden vectors.

Selective summary



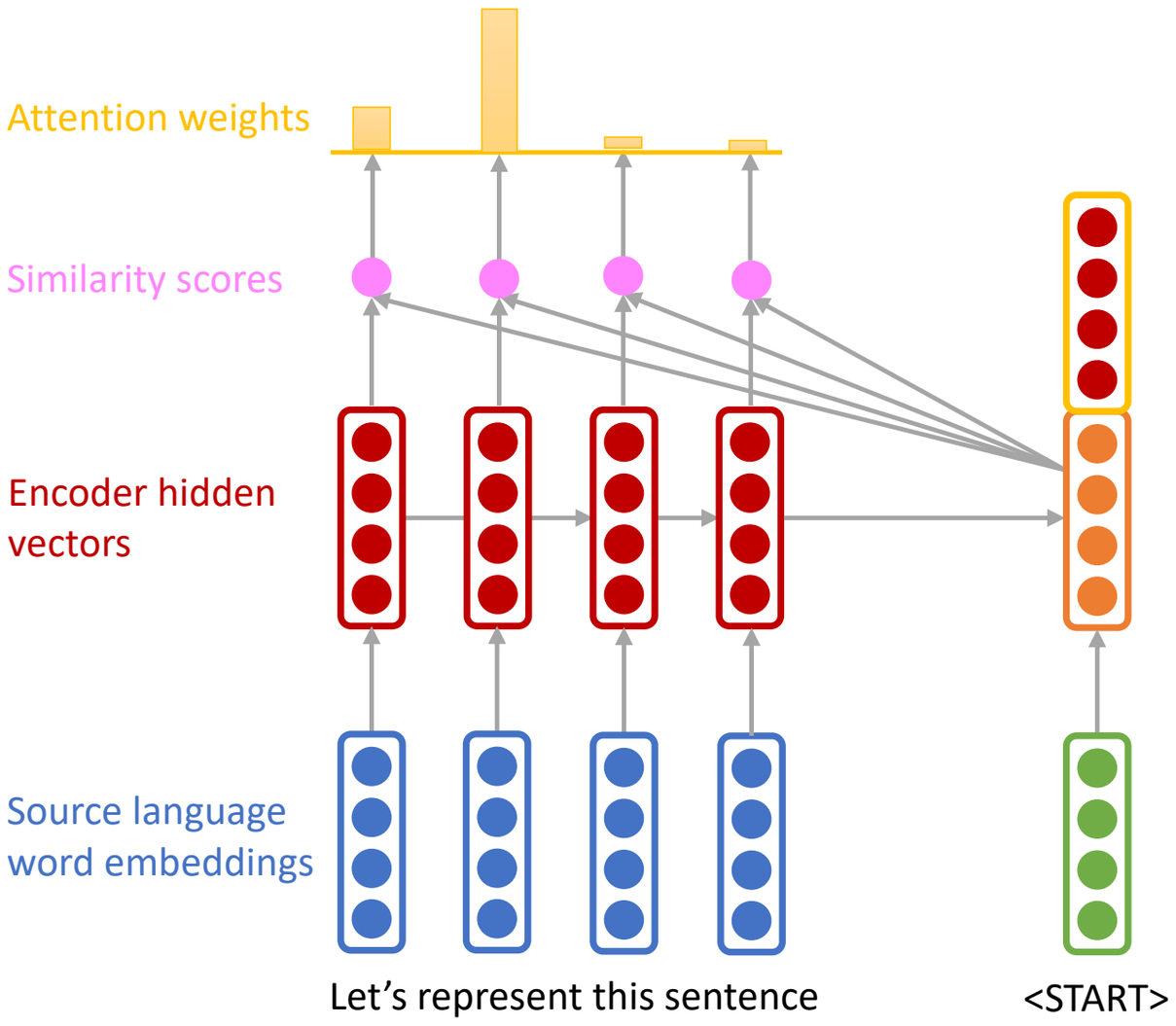
Use the attention weights to generate the **context vector** by taking a weighted sum of the encoder hidden vectors.

Mostly comprised of the second encoder hidden vector.

Most of the attention is on the second word.

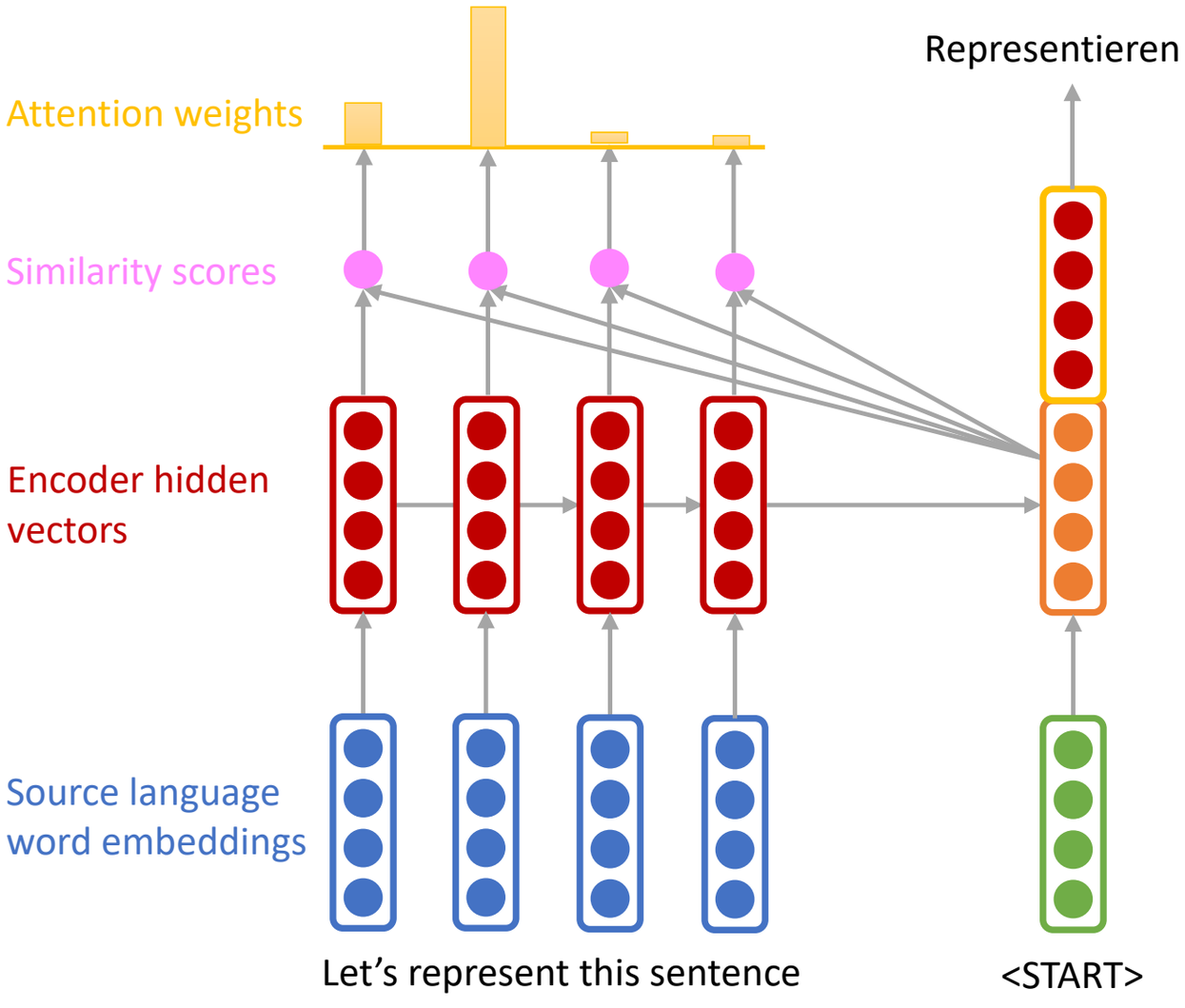
Add context vector to decoder hidden state

Concatenate the context vector to the decoder hidden vector

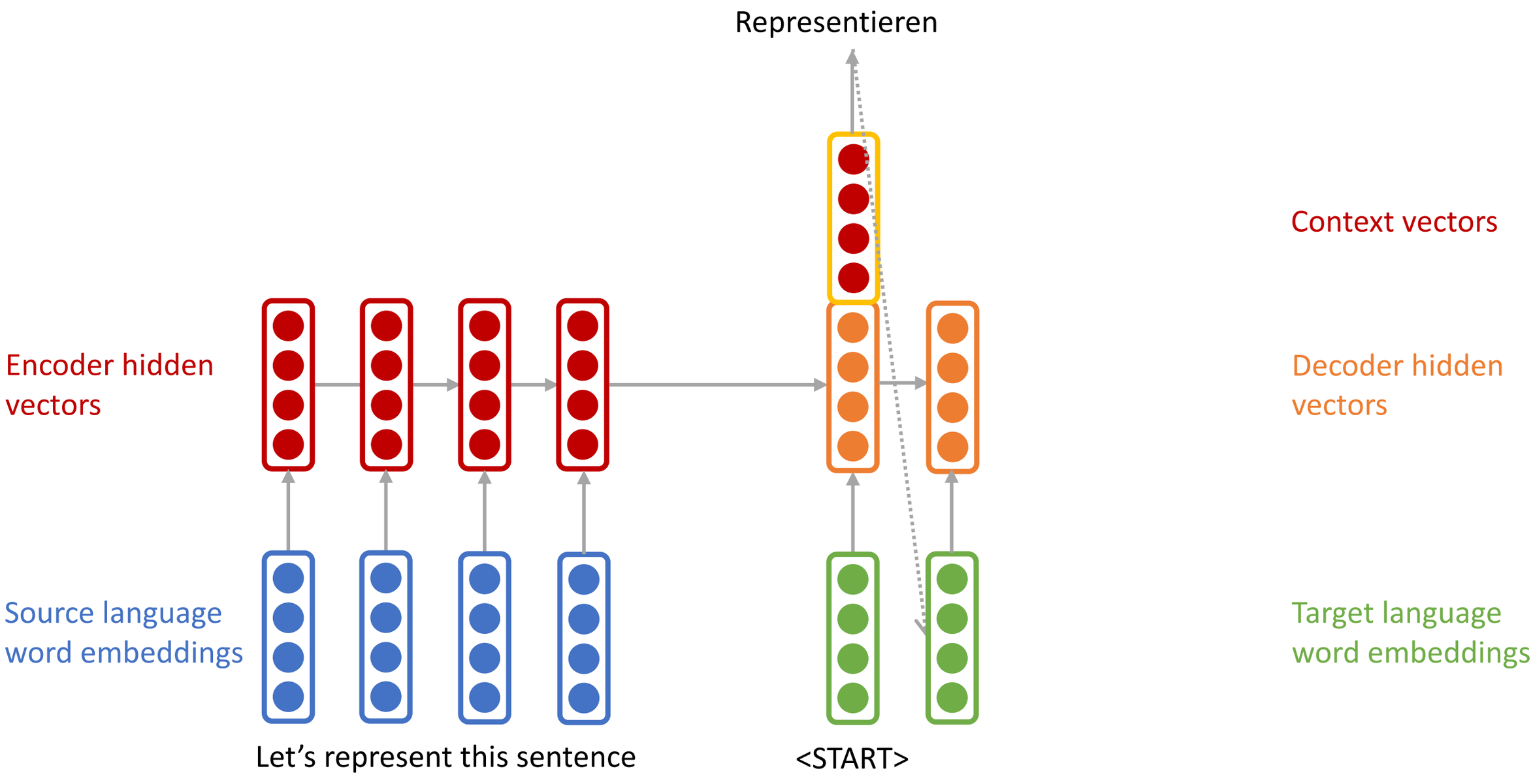


Generate next word

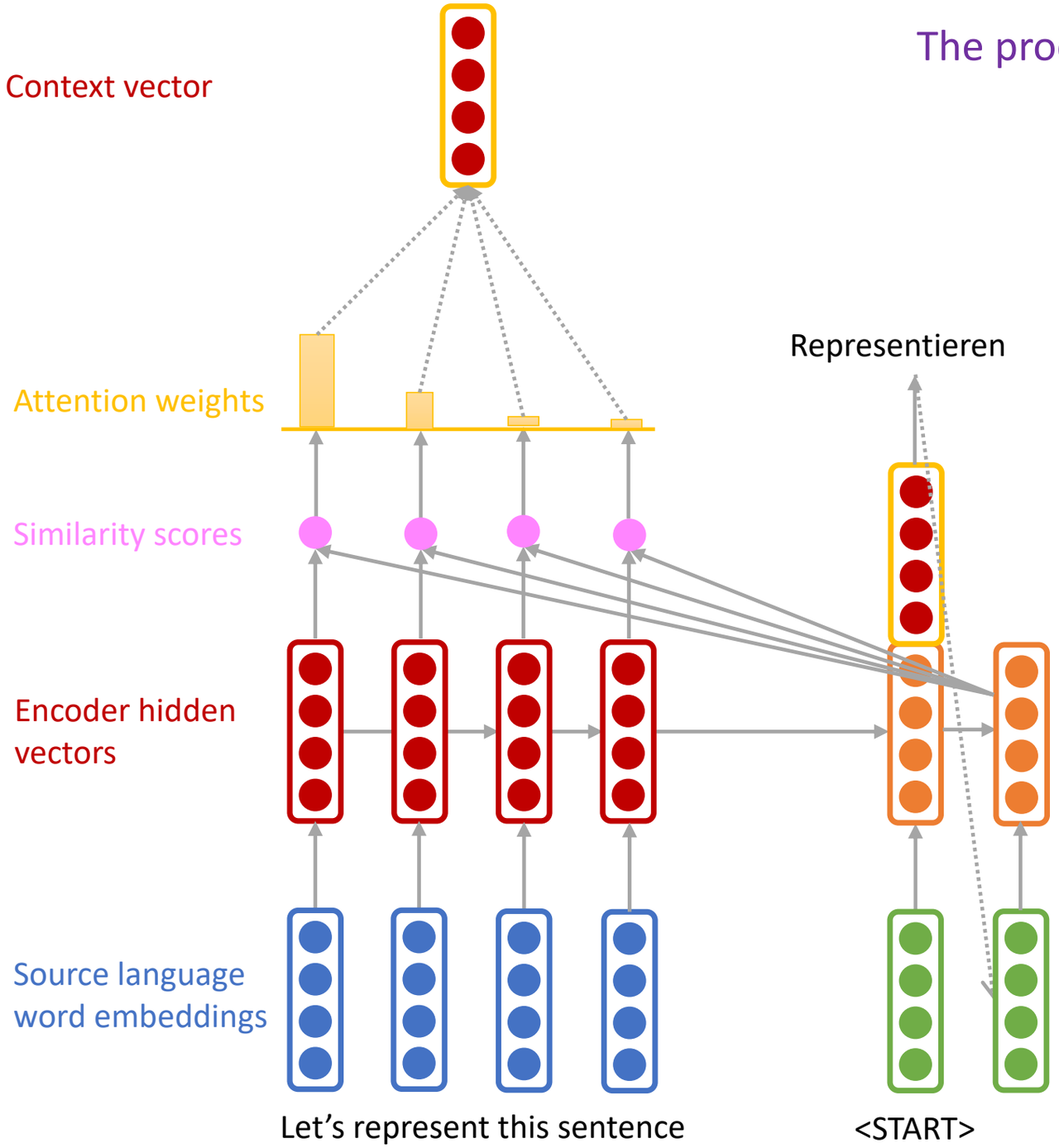
Concatenate the context vector to the decoder hidden vector and generate the first word.



Proceed as before



Selective context vector for each time step



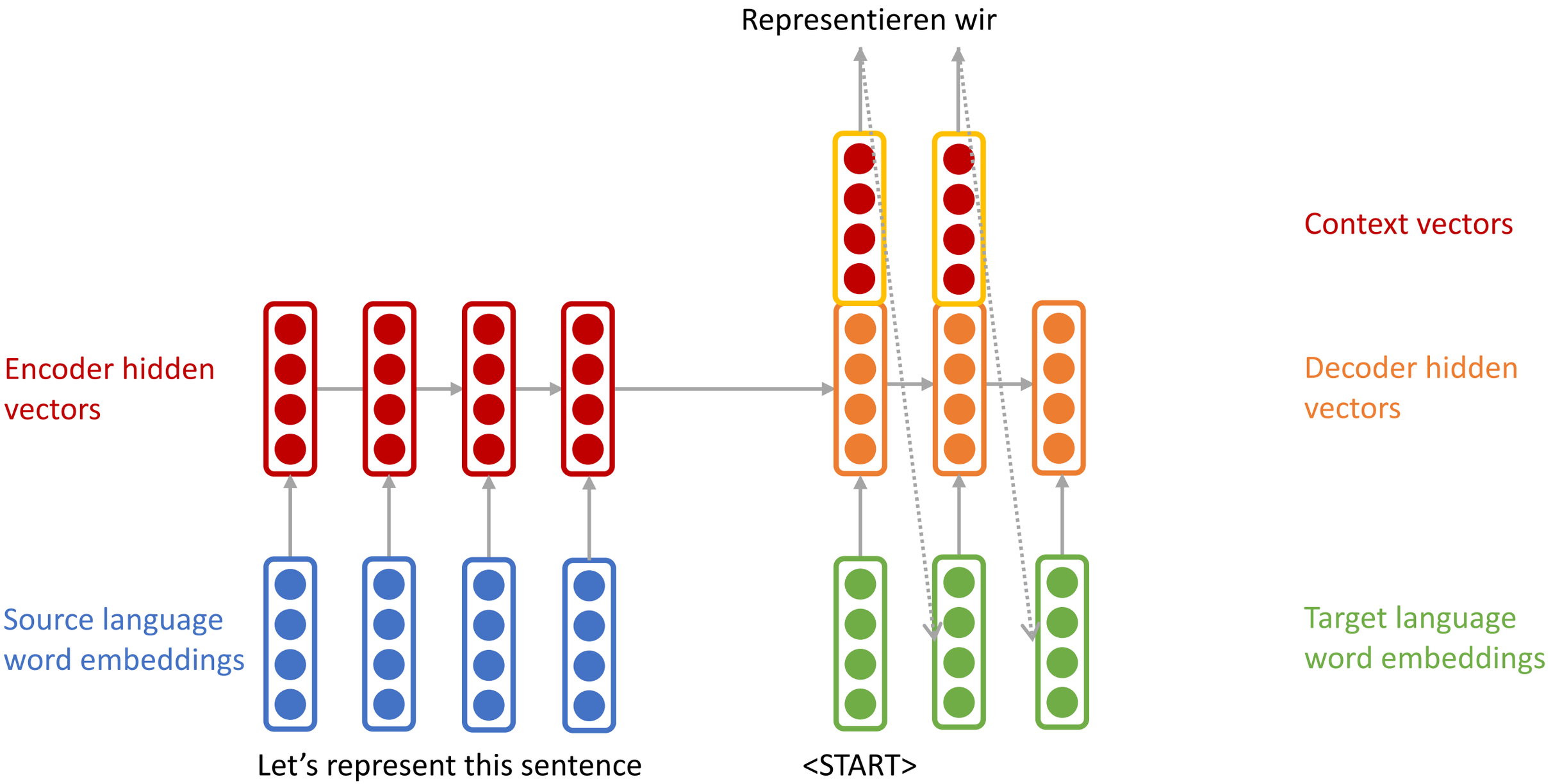
The proceed as before.

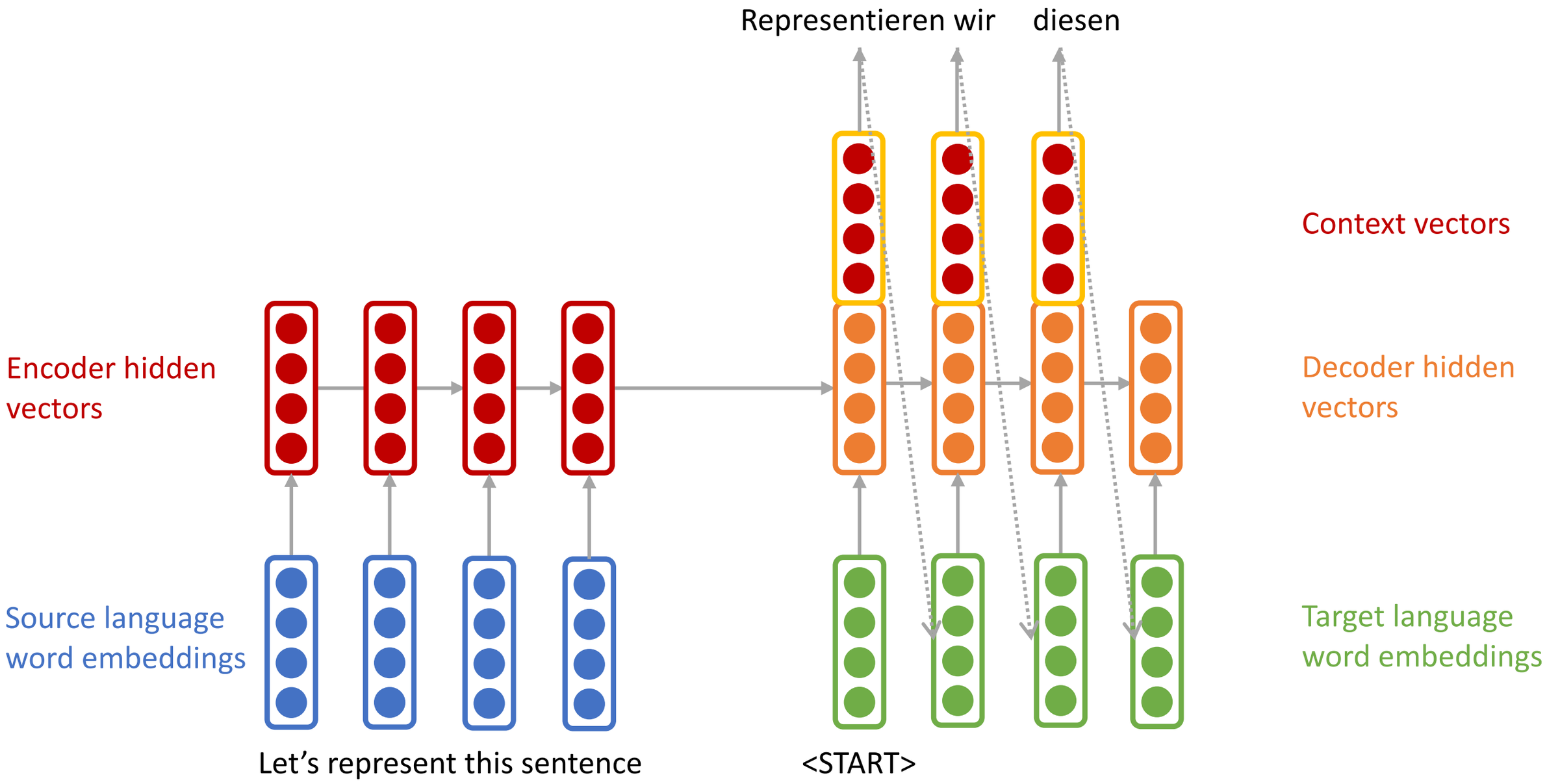
Context vectors

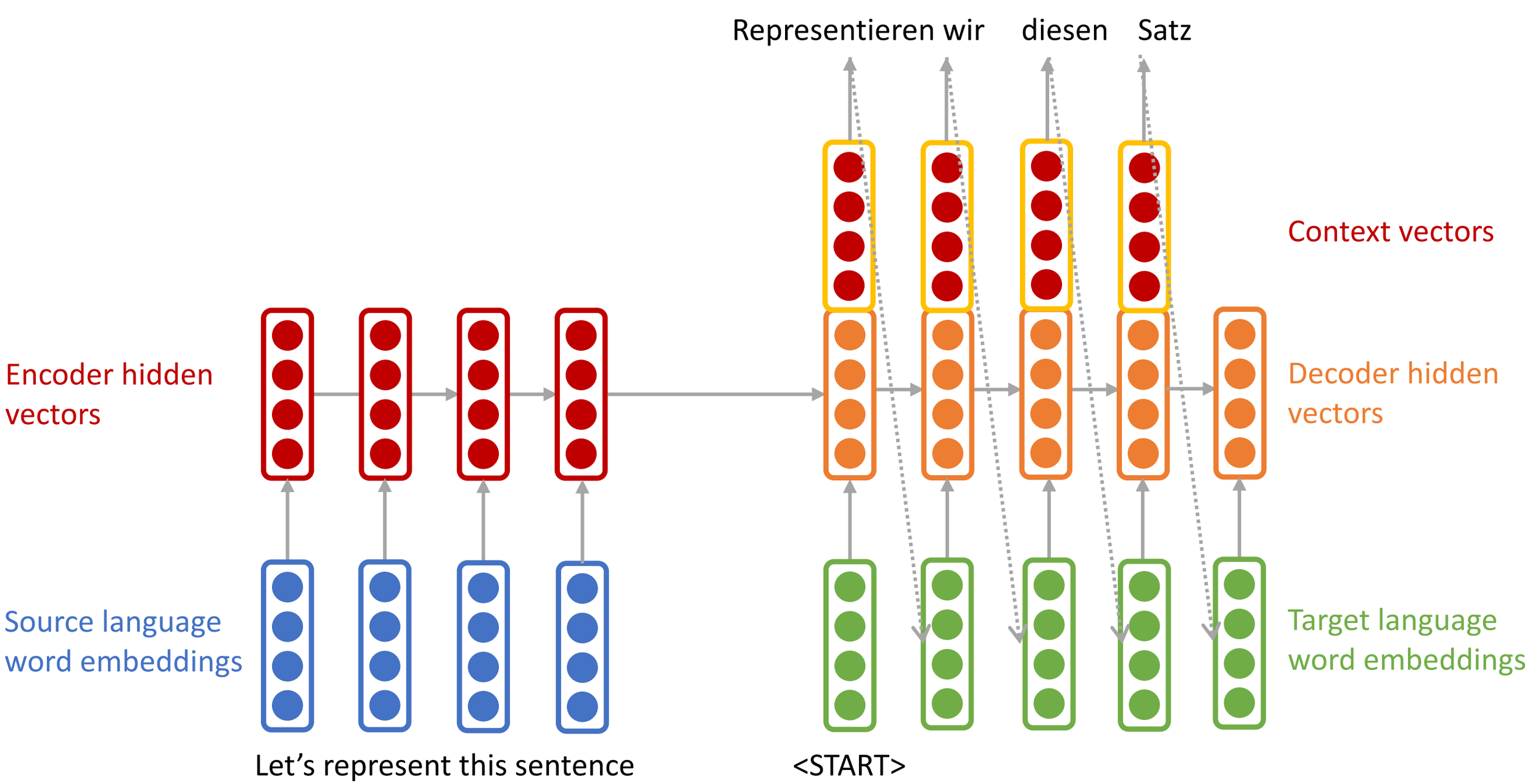
Decoder hidden vectors

Target language word embeddings

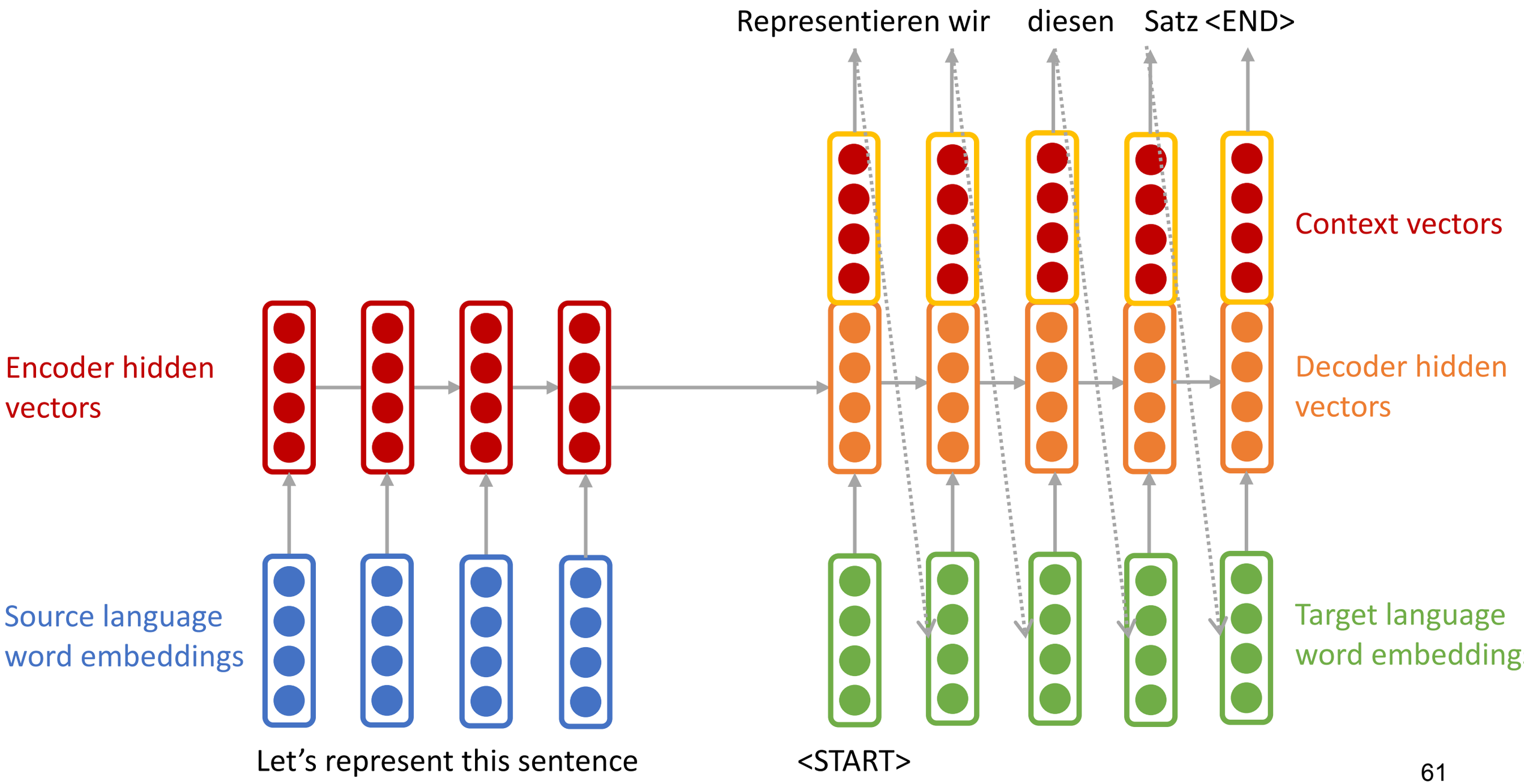
Generate next word




Representieren wir diesen
Context vectors
Encoder hidden vectors
Decoder hidden vectors
Source language word embeddings
Target language word embeddings
Let’s represent this sentence
<START>

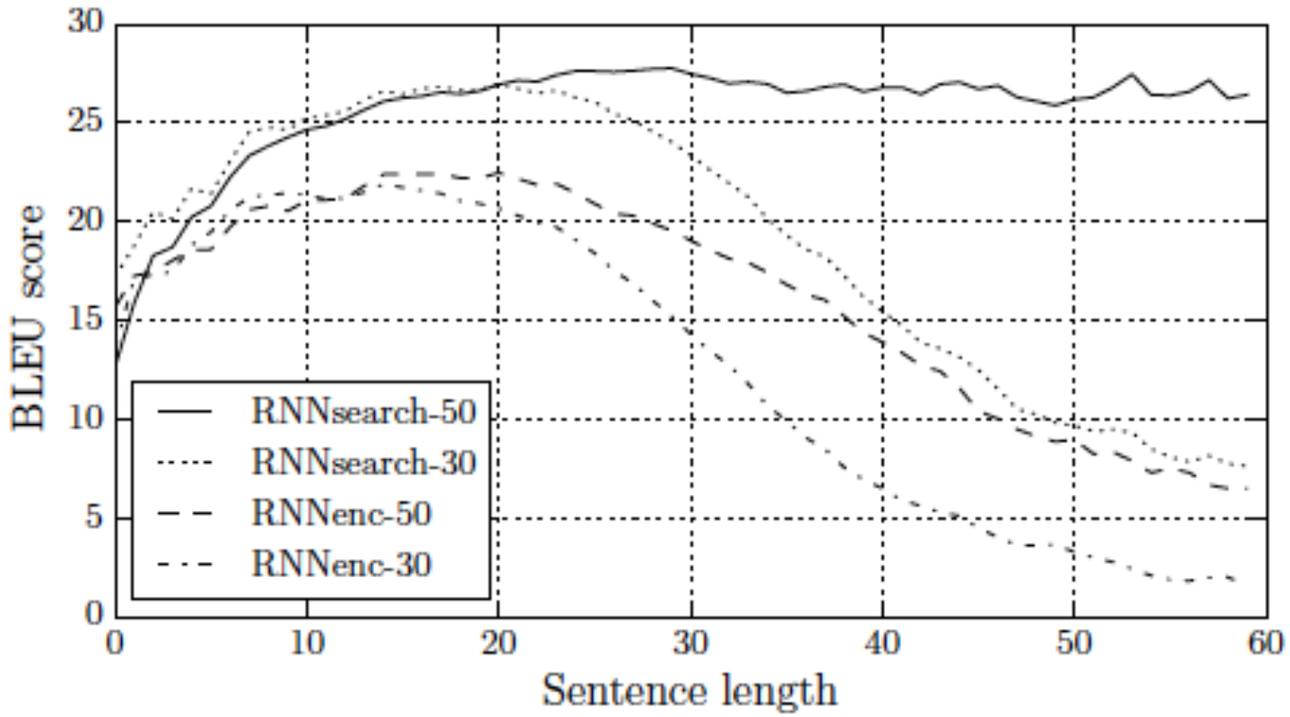

Representieren wir
diesen
Satz
Context vectors
Encoder hidden vectors
Decoder hidden vectors
Source language word embeddings
Target language word embeddings
Let’s represent this sentence
<START>

Translation is complete when the <END> token is generated.



Attention Machine Translation performance

Bahdanau, Cho, Bengio (ICLR-2015)



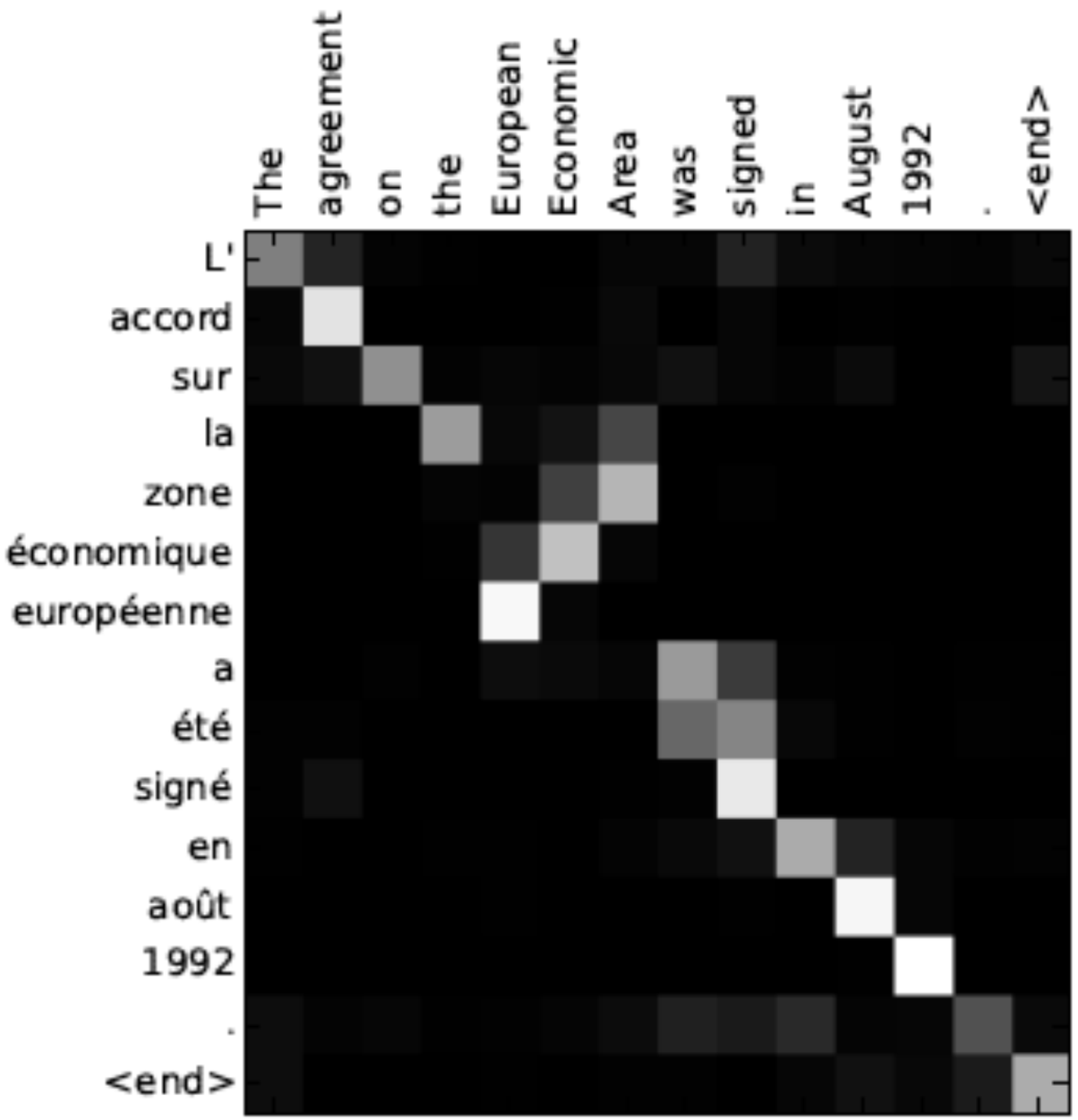
RNNsearch: with attention
RNNenc: no attention

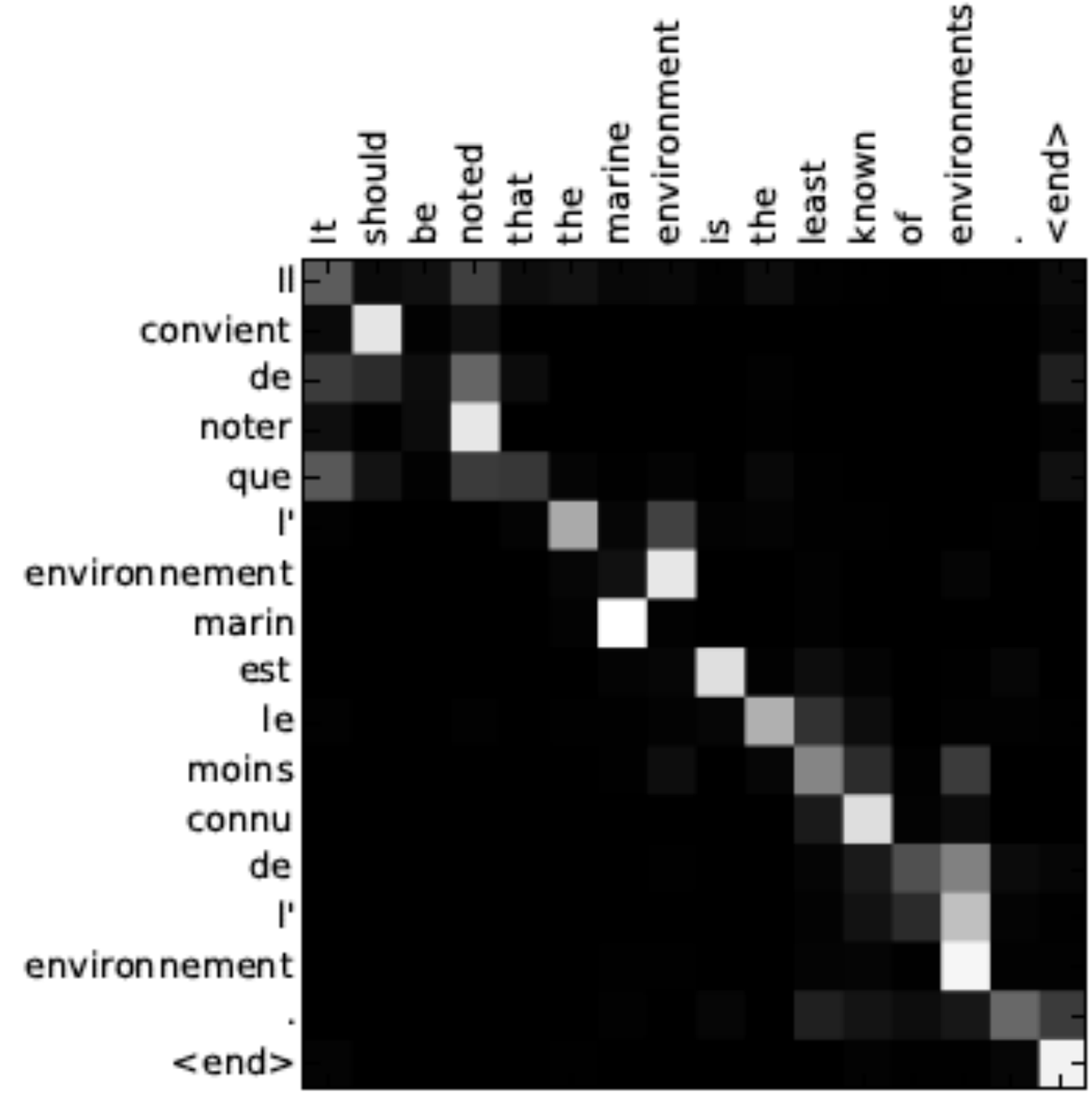
Bleu: BiLingual Evaluation Understudy

- Percentage of translated words that appear in ground truth

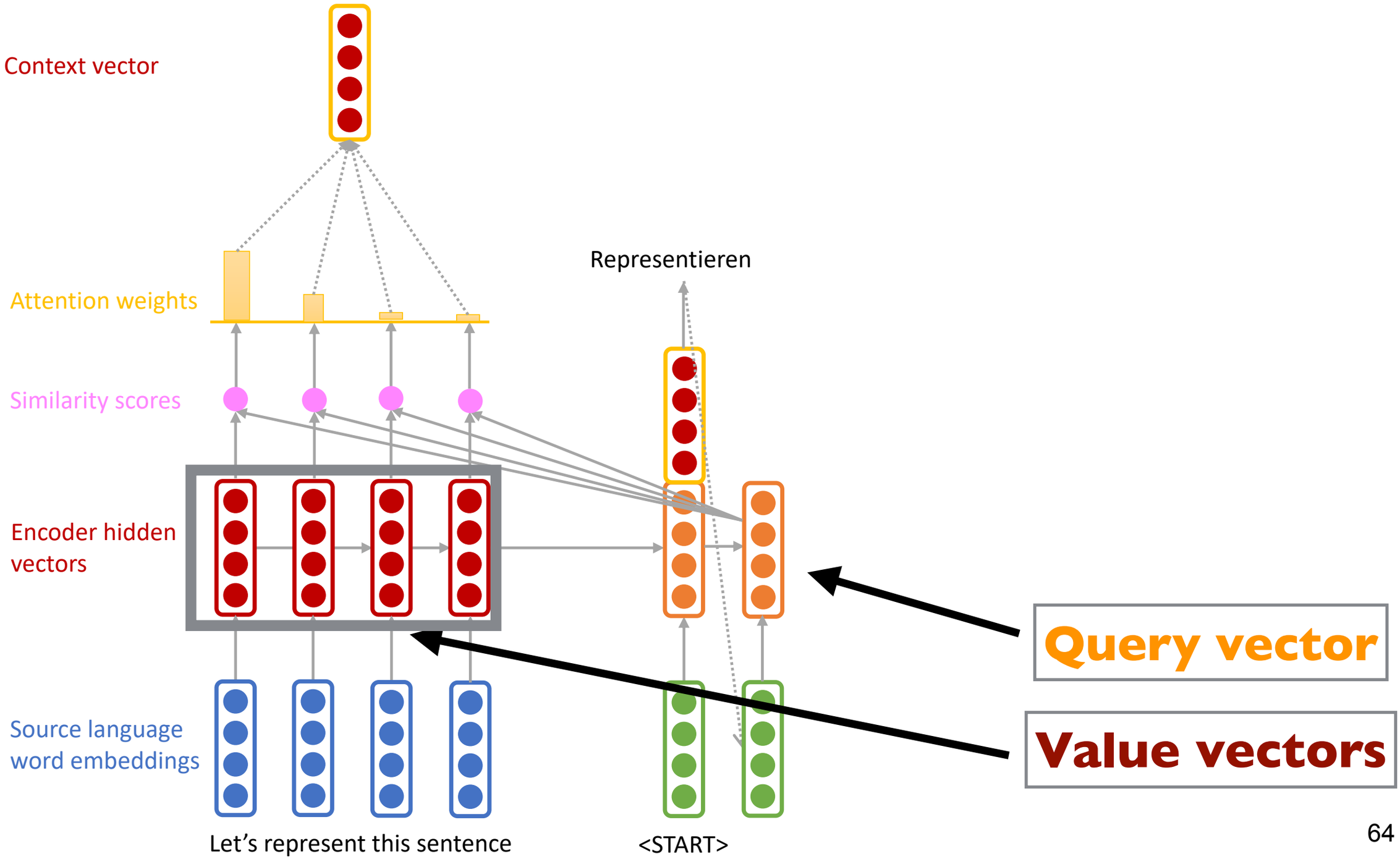
Alignment Example

Bahdanau, Cho, Bengio (ICLR-2015)





Important terminology: query vector, and value vectors



Attention is a general Deep Learning technique

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* $\boldsymbol{e} \in \mathbb{R}^N$

There are multiple ways to do this

2. Taking softmax to get *attention distribution* α:

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

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* $\boldsymbol{a}$ (sometimes called the *context vector*)

Attention: the many forms

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 $d_1 = d_2$.

Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$ [Luong, Pham, and Manning 2015]

- Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix. Perhaps better called "bilinear attention"

Reduced-rank multiplicative attention: $e_i = s^T(\boldsymbol{U}^T\boldsymbol{V})h_i = (\boldsymbol{U}s)^T(\boldsymbol{V}h_i)$

- For low rank matrices $\boldsymbol{U} \in \mathbb{R}^{k \times d_2}, \boldsymbol{V} \in \mathbb{R}^{k \times d_1}, k \ll d_1, d_2$

Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$ [Bahdanau, Cho, and Bengio 2014]

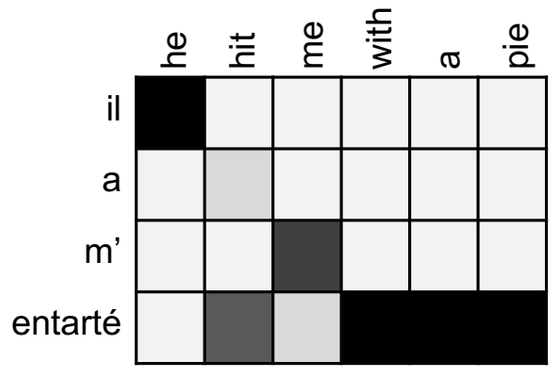
- Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.
- d_3 (the attention dimensionality) is a hyperparameter
- "Additive" is a weird/bad name. It's really using a feed-forward neural net layer.

More information: "Deep Learning for NLP Best Practices", Ruder, 2017.
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017

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 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



Advantages of NMT: from 3 yrs of team effort, to a few weeks of one person's effort

Compared to SMT, NMT has many advantages:

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

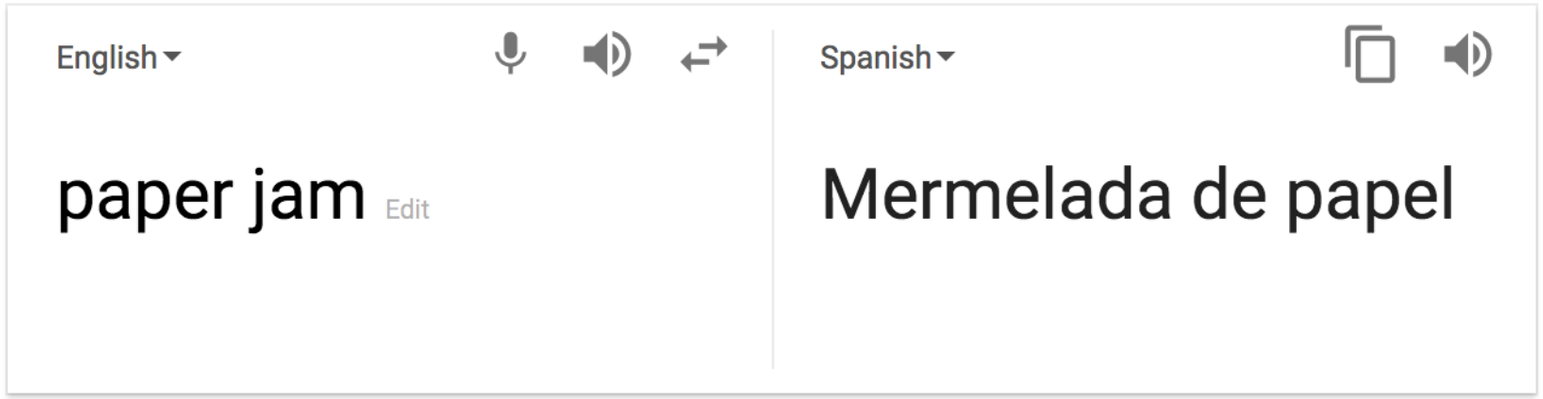
Disadvantages of NMT?

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

So is Machine Translation solved?

- Using common sense is still hard



?

Frontiers in MT: Is ChatGPT a good translator?

Table 3: Comparison of different prompts for ChatGPT to perform Chinese-to-English (Zh⇒En) translation.

System	BLEU↑	ChrF++↑	TER↓
Google	31.66	57.09	56.21
DeepL	31.22	56.74	57.84
Tencent	29.69	56.24	57.16
ChatGPT w/ TP1	23.25	53.07	66.03
ChatGPT w/ TP2	24.54	53.05	63.79
ChatGPT w/ TP3	**24.73**	**53.71**	**62.84**

- Works okay for Chinese-English, but less good at generating into low-resource languages (English -> Romanian doesn't work well)

"Is ChatGPT A Good Translator? Yes With GPT-4 As The Engine" Jia et al. (2023)

MT for low-resource languages

Burmese, Indonesian, Turkish

Transfer	BLEU My→En	Id→En	Tr→En
baseline (no transfer)	4.0	20.6	19.0
transfer, train	17.8	27.4	20.3
transfer, train, reset emb, train	13.3	25.0	20.0
transfer, train, reset inner, train	3.6	18.0	19.1

Table 3: Investigating the model's capability to restore its quality if we reset the parameters. We use En→De as the parent.

Aji et al. (2020)

- Interest in deploying MT systems for languages with little or no
- BPE allows us to transfer models even without training on a specific language

How do we evaluate MT

- We can ask a human to do evaluation, but slow, expensive, and sometimes inconsistent
- Or use automated metrics like BLEU

BLEU limitations

- BLEU - bilingual evaluation understudy (Papineni et., al 2002)

  - BLEU is **useful** but **imperfect**
    - Easy to use, good for measuring system improvement
    - But, there are many valid ways to translate a sentence
    - A good translation can get a poor BLEU score because of low n-gram overlap with the human translation

Embedding-based Metrics

- Recent: metrics based on neural models
  - **BertScore:** Find similarity between BERT embeddings (unsupervised) (Zhang et al. 2020)
  - **BLEURT:** Train BERT to predict human evaluation scores (Sellam et al.
  - …

Frontiers: Evaluation with LLMs

```
Score the following translation from {source_lang} to {target_lang} with respect
to the human reference on a continuous scale from 0 to 100, where score of zero mea
"no meaning preserved" and score of one hundred means "perfect meaning and grammar"

{source_lang} source: "{source_seg}"
{target_lang} human reference: {reference_seg}
{target_lang} translation: "{target_seg}"
Score:
```

Figure 1: The best-performing prompt based on Direct Assessment expecting a score between 0–100. Template **portions in bold face** are used only when a human reference translation is available.

- Outperforms many learned MT metrics (Transformers trained over (source, target, reference) triples to reproduce human judgments of quality)

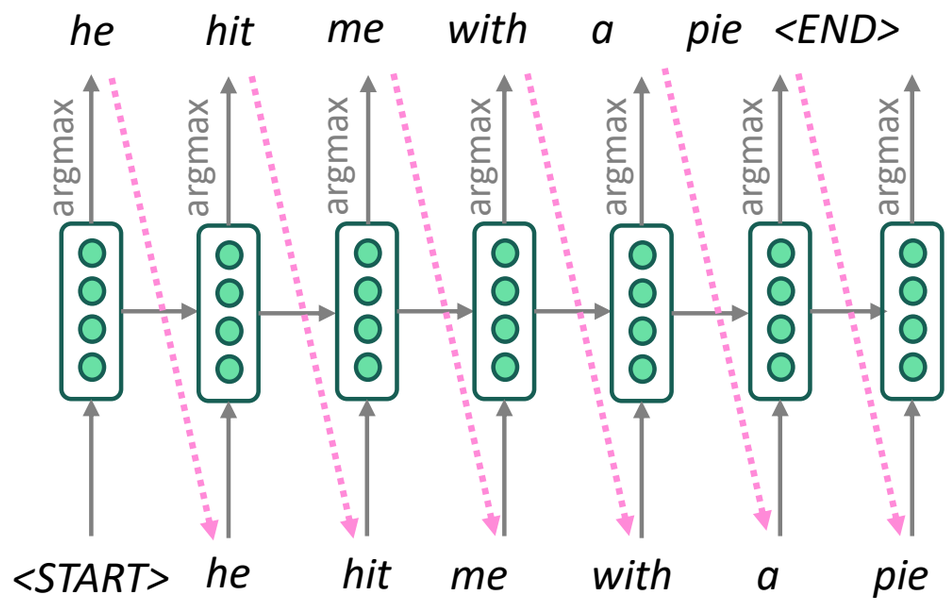
"Large Language Models Are State-of-the-Art Evaluators of Translation Quality" Kocmi et al. (2023)

Decoding

- Greedy decoding is not great

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

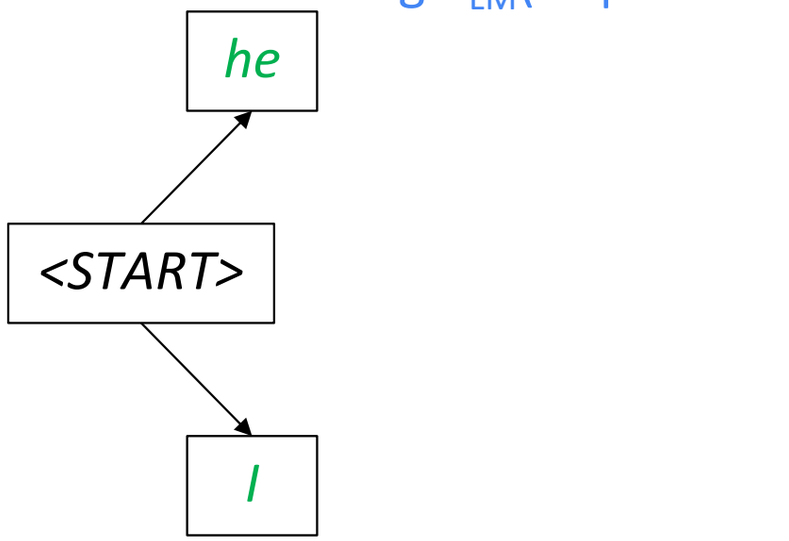
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam search decoding: example

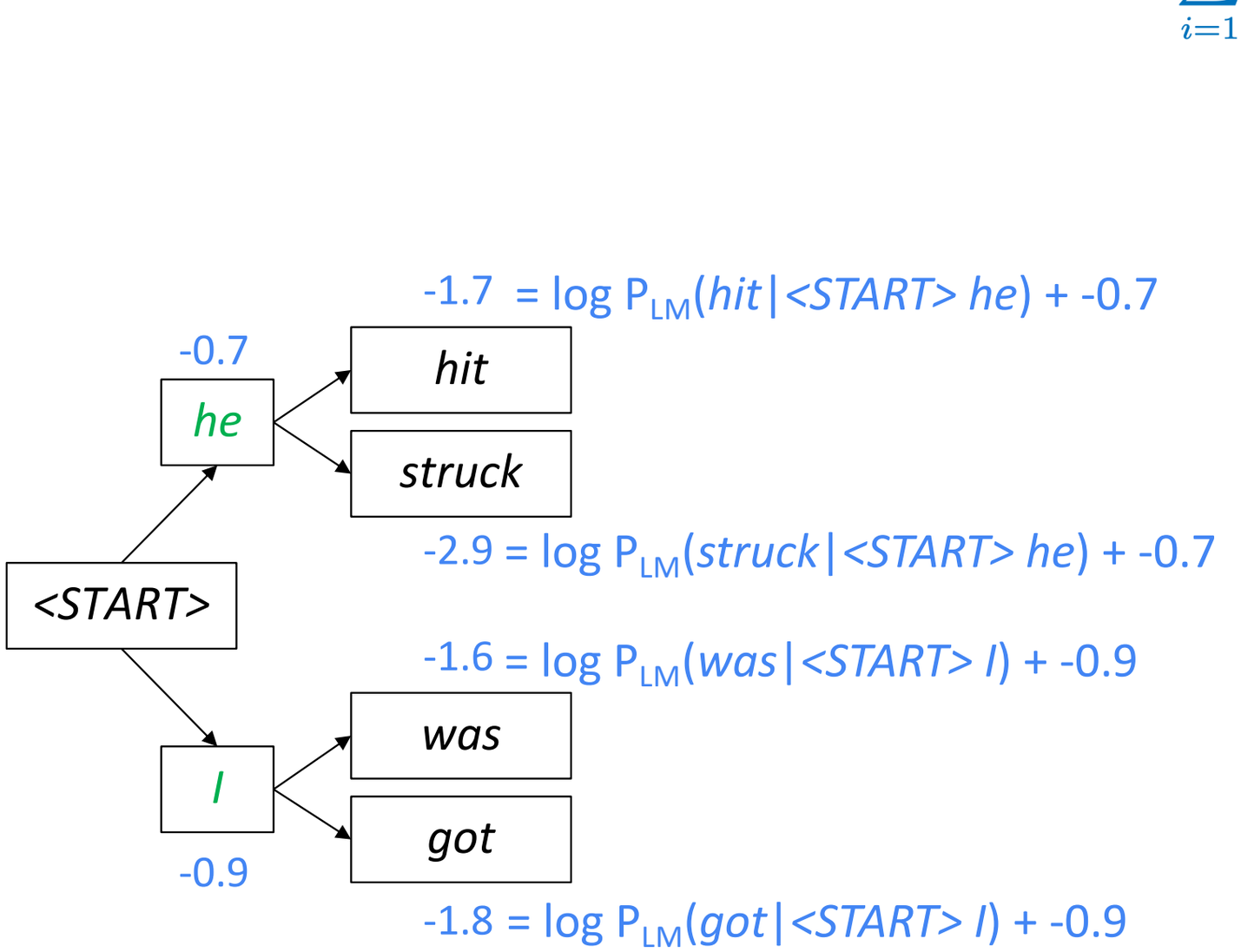
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

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

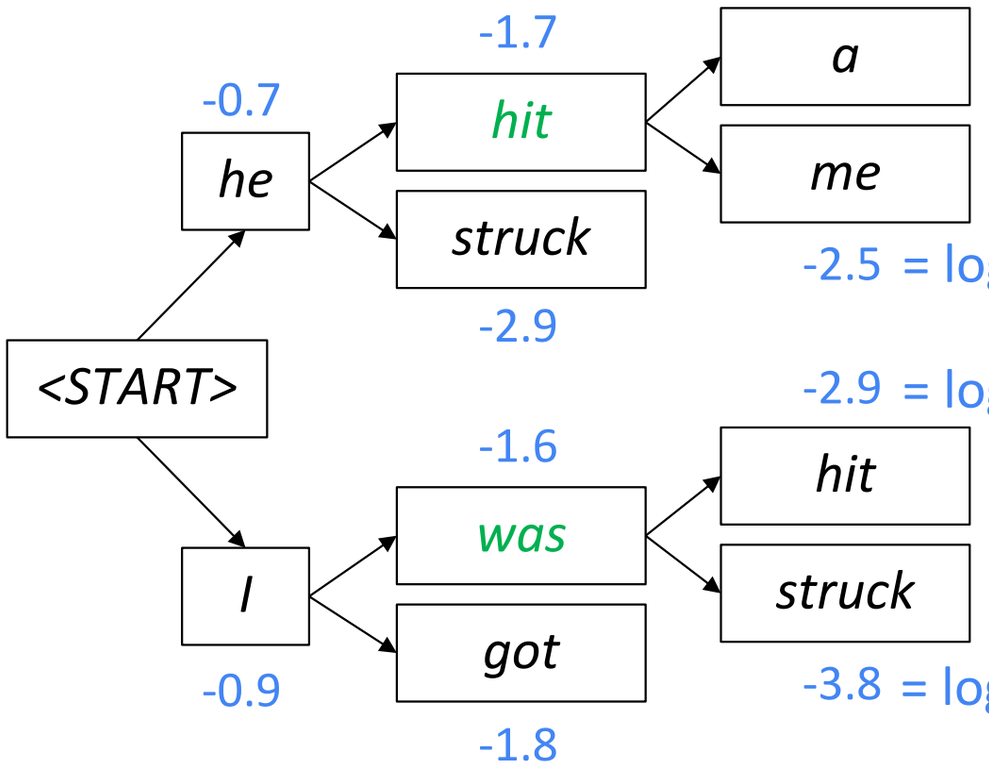


For each of the *k* hypotheses, find top *k* next words and calculate scores

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

Beam search decoding: example

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



-2.8 = log P_{LM}(*a* | *<START> he hit*) + -1.7

-2.5 = log P_{LM}(*me* | *<START> he hit*) + -1.7

-2.9 = log P_{LM}(*hit* | *<START> I was*) + -1.6

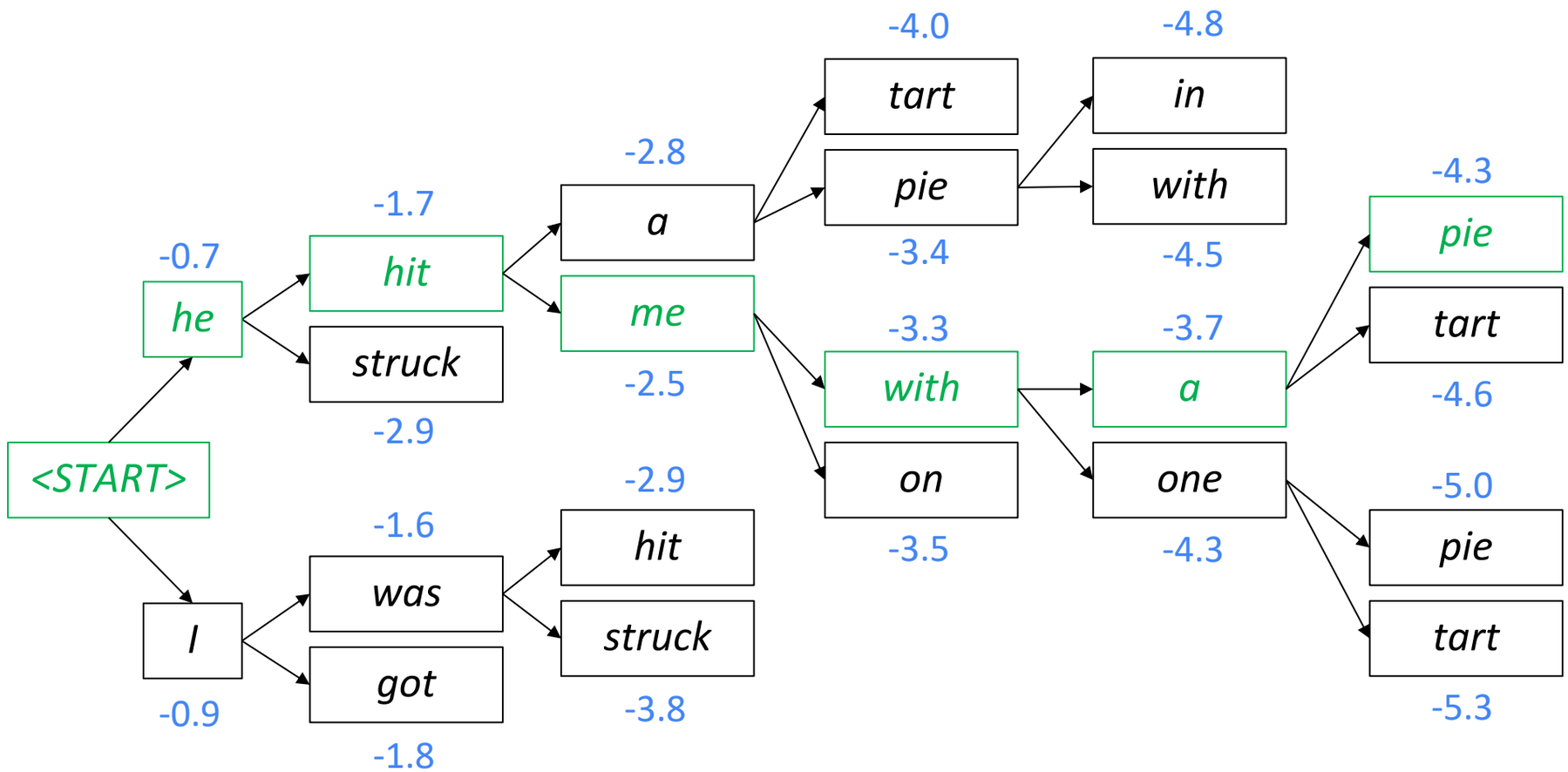
-3.8 = log P_{LM}(*struck* | *<START> I was*) + -1.6

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

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

Beam search decoding: example

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



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

In greedy decoding, usually we decode until the model produces an <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:

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

Summary

- Since 2014, Neural MT rapidly replaced intricate Statistical MT

- Sequence-to-sequence is the architecture for NMT (uses 2 models: encoder and decoder)

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

NMT: the first big success story of NLP Deep Learning?

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

- **2014**: First seq2seq paper published

- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone has

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

Acknowledgements

- Includes slides from
  - Abigail See
  - Chris Manning
  - Many other directly or indirectly