344.063/163 KV Special Topic: Natural Language Processing with Deep Learning

Sequence-to-Sequence Models with LSTM for Text Summarization

Navid Rekab-Saz

navid.rekabsaz@jku.at

Institute of Computational Perception

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- RNNs with Gates: LSTM, GRU
- Document summarization
- Abstractive summarization with seq2seq
- Extractive summarization with RNNs*

* The content of this section will not be a part of the final evaluation

Agenda

- **RNNs with Gates: LSTM, GRU**
- Document summarization
- Abstractive summarization with seq2seq
- Extractive summarization with RNNs

Gate vector

- Gate vector:
  - A vector with values between 0 and 1
  - Gate vector acts as "gate-keeper", such that it controls the content flow of another vector
- Gate vectors are typically defined using sigmoid:

$$\boldsymbol{g} = \sigma(some\ vector)$$

… and are applied to a vector $\boldsymbol{v}$ with element-wise multiplication to control its contents:

$$\boldsymbol{g} \odot \boldsymbol{v}$$

- For each element (feature) i of the vectors:
  - If g_i is 1 $\rightarrow v_i$ remains the same; everything passes; *open* gate!
  - If g_i is 0 $\rightarrow v_i$ becomes 0; nothing passes; *closed* gate!

Long Short-Term Memory (LSTM)

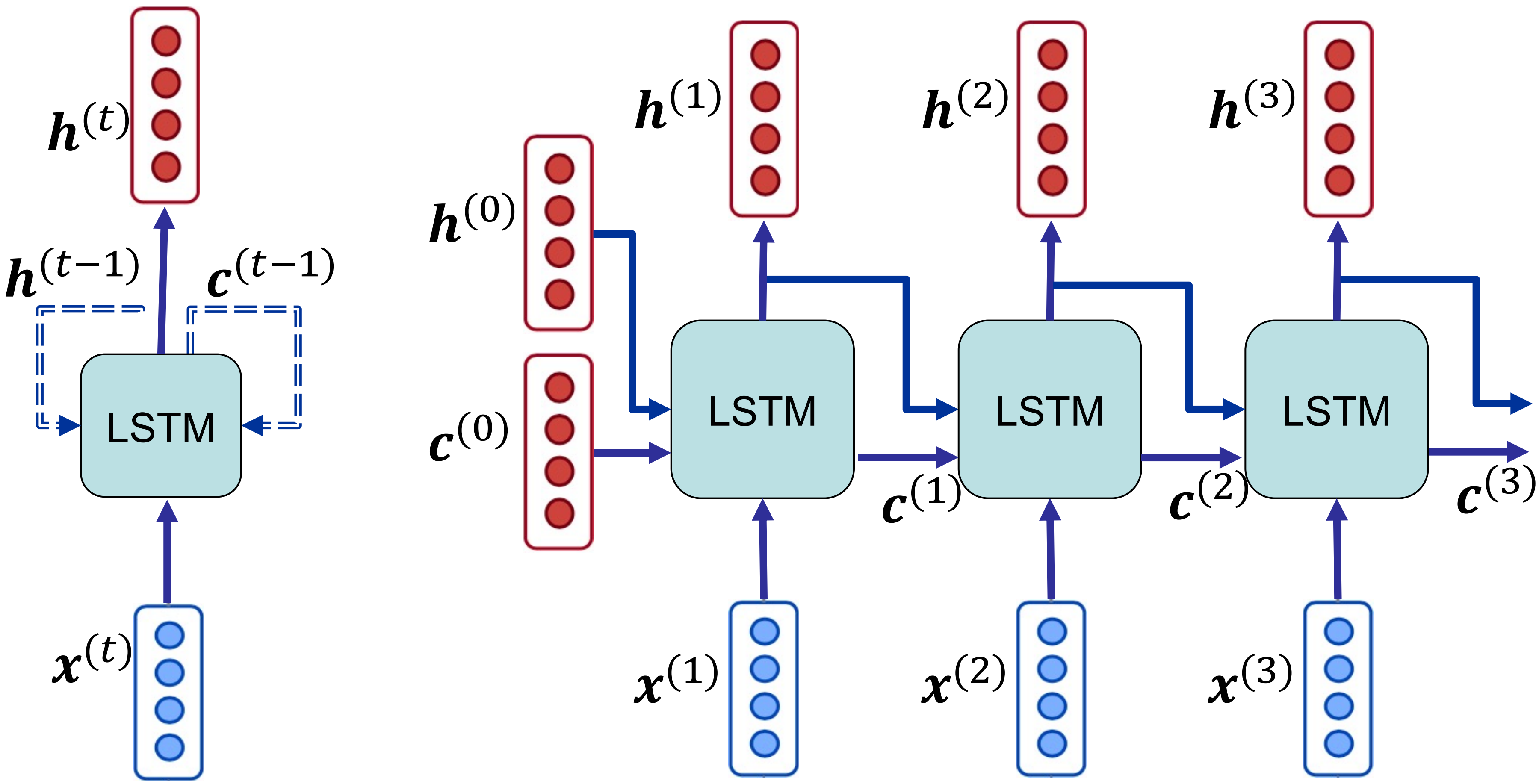
- Proposed by Hochreiter and Schmidhuber in 1997

- LSTM exploits a new vector cell state $\boldsymbol{c}^{(t)}$ to carry the memory of previous states
  - The cell state stores long-term information
  - As in vanilla RNN, hidden states $\boldsymbol{h}^{(t)}$ is used as output vector

- LSTM controls the process of reading, writing, and erasing information in/from memory states
  - These controls are done using gate vectors
  - Gates are dynamic and defined based on the input vector and hidden state

Hochreiter, Sepp, and Jürgen Schmidhuber. "**Long short-term memory**" *Neural computation* (1997)

LSTM – unrolled



LSTM definition – gates

- Gates are functions of input vector $\boldsymbol{x}^{(t)}$ and previous hidden state $\boldsymbol{h}^{(t-1)}$

$$\boldsymbol{i}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$
$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hi} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xi} + \boldsymbol{b}_i)$$

input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{f}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$
$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hf} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xf} + \boldsymbol{b}_f)$$

forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{o}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{ho} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xo} + \boldsymbol{b}_o)$$

output gate: controls what parts of cell are output to hidden state

Model parameters (weights) are shown in red

Inspired from: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTM definition – states

$$\tilde{\boldsymbol{c}}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hc} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xc} + \boldsymbol{b}_c)$$

new cell content: the new content to be used for cell and hidden (output) state

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

cell state: erases ("forgets") some content from last cell state, and writes ("inputs") some new cell content

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

hidden state: reads ("outputs") some content from the current cell state

Model parameters (weights) are shown in red

Inspired from: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTM definition – all together

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hi} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xi} + \boldsymbol{b}_i)$$

input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hf} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xf} + \boldsymbol{b}_f)$$

forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{ho} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xo} + \boldsymbol{b}_o)$$

output gate: controls what parts of cell are output to hidden state

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hc} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xc} + \boldsymbol{b}_c)$$

new cell content: the new content to be used for cell and hidden (output) state

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

cell state: erases ("forgets") some content from last cell state, and writes ("inputs") some new cell content

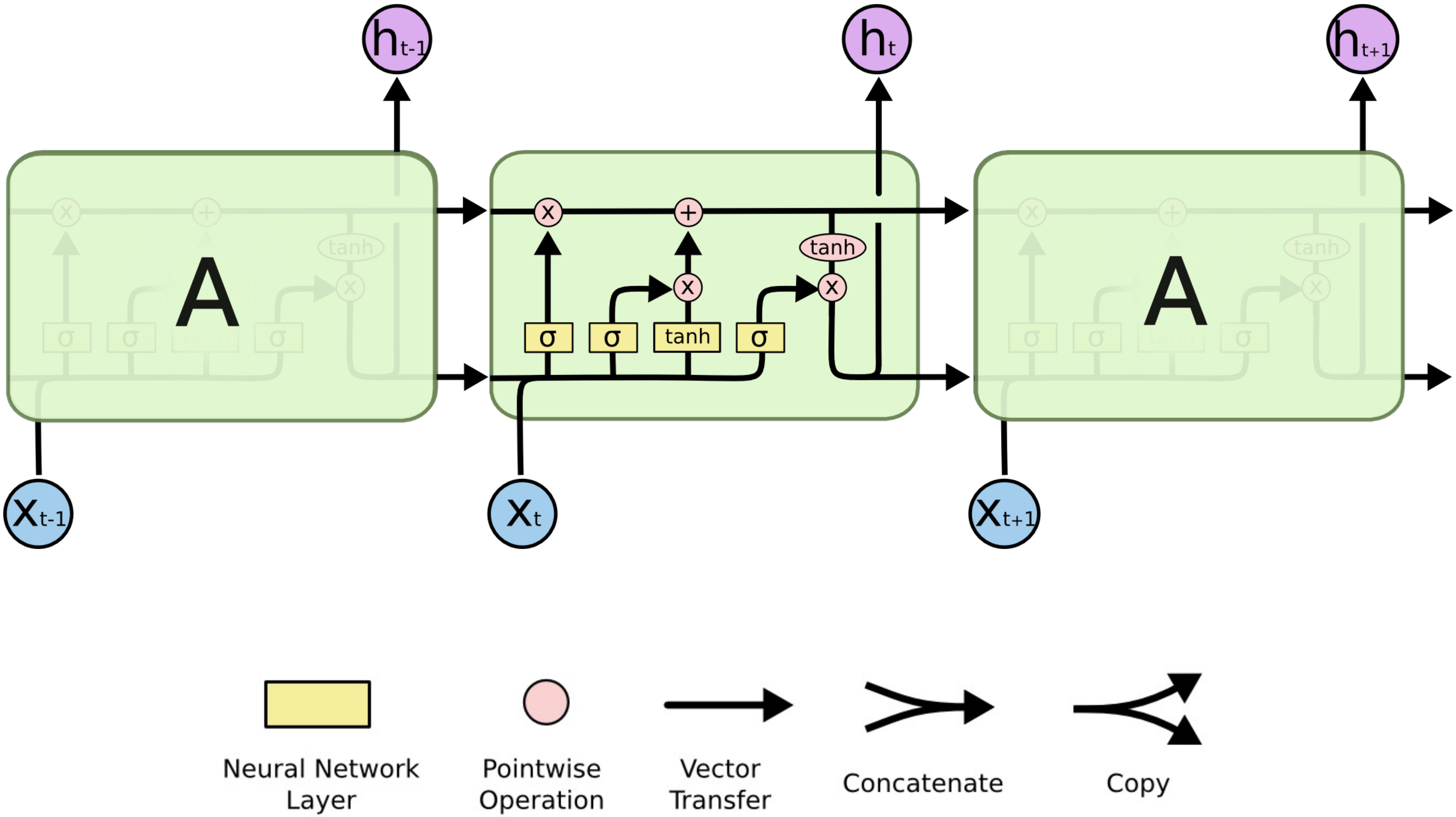
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$$

hidden state: reads ("outputs") some content from the current cell state

Model parameters (weights) are shown in red

Inspired from: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTM definition – visually!



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hu} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xu} + \boldsymbol{b}_u)$$

update gate: controls what parts of hidden state are updated vs preserved

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hr} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xr} + \boldsymbol{b}_r)$$

reset gate: controls what parts of previous hidden state are used to compute new content

new hidden state content: (1) reset gate selects useful parts of previous hidden state. (2) Use this and current input to compute new hidden content.

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh((\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)})\,\boldsymbol{W}_{hh} + \boldsymbol{x}^{(t)}\boldsymbol{W}_{xh} + \boldsymbol{b}_h)$$

$$\boldsymbol{h}^{(t)} = \left(1 - \boldsymbol{u}^{(t)}\right) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)}$$

hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

Model parameters (weights) are shown in red

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). **Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation**. In *Proc. Of EMNLP)*

RNNs with gates – counting parameters

- Parameters in LSTM (bias terms discarded)
  - $\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{ho}, \boldsymbol{W}_{hc} \rightarrow h \times h \ * 4$
  - $\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xo}, \boldsymbol{W}_{xc} \rightarrow d \times h \ * 4$

- Parameters in GRU (bias terms discarded)
  - $\boldsymbol{W}_{hu}, \boldsymbol{W}_{hr}, \boldsymbol{W}_{hh} \rightarrow h \times h \ * 3$
  - $\boldsymbol{W}_{xu}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh} \rightarrow d \times h \ * 3$

- If also considering encoder and decoder embeddings (e.g. in a Language Modeling network)
  - $\boldsymbol{E} \rightarrow |\mathbb{V}| \times d$
  - $\boldsymbol{U} \rightarrow h \times |\mathbb{V}|$

d and h are the number of dimensions of the input embedding and hidden vectors, respectively.

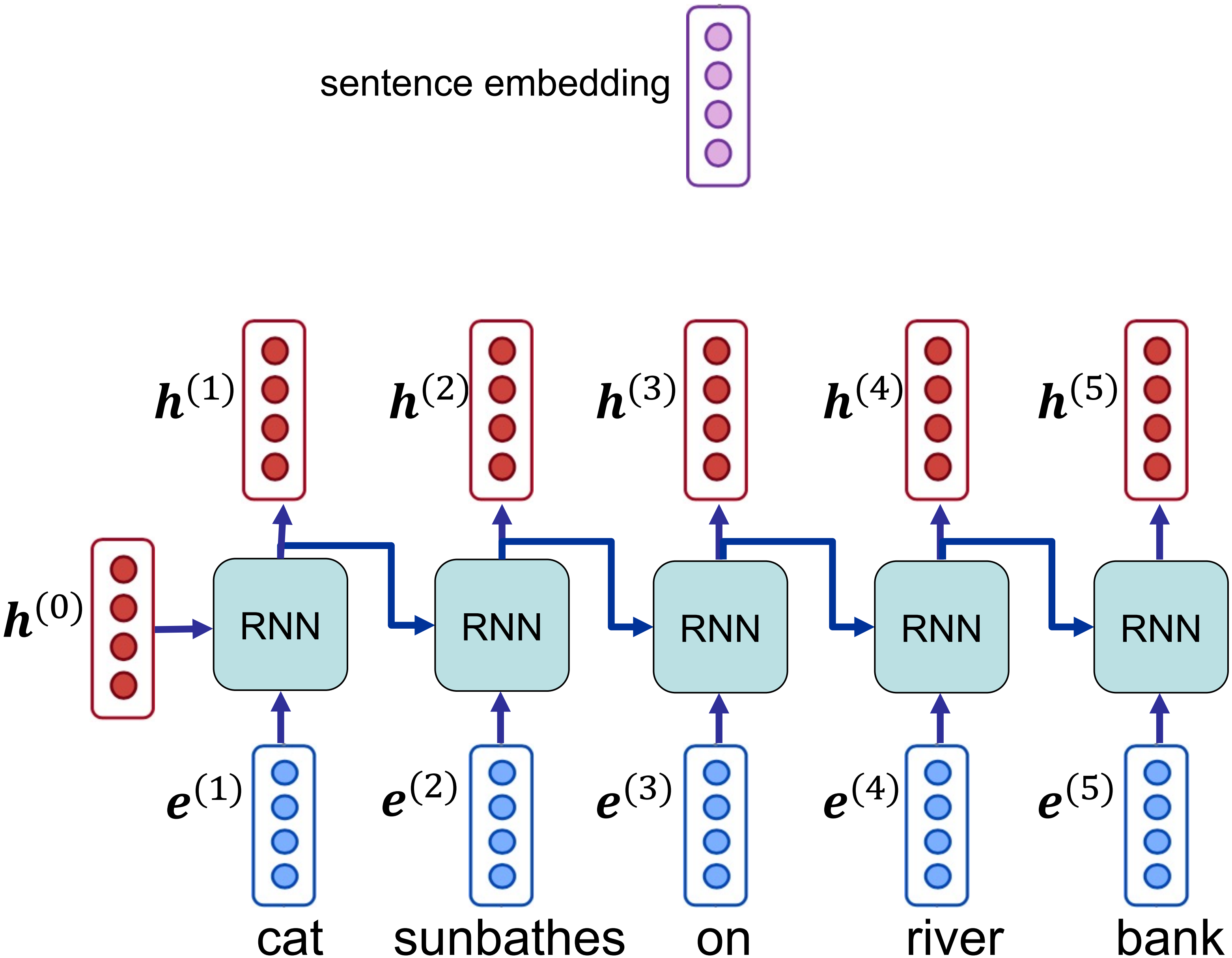
RNNs with gates – summary

- LSTM (and GRU) with dynamic gate mechanisms makes it easier to preserve necessary information over many timesteps

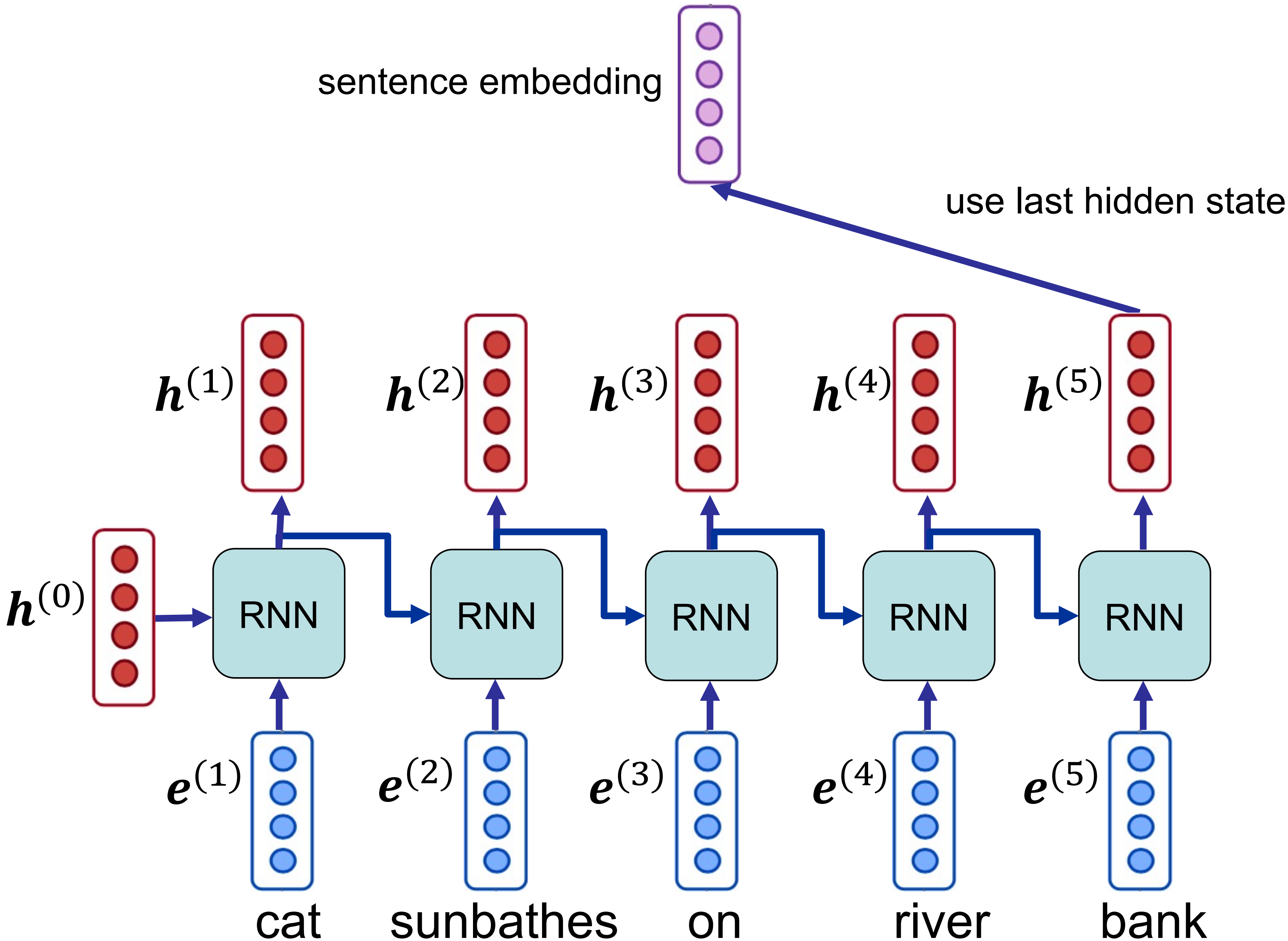
- LSTM does not *guarantee* that there is no vanishing/exploding gradient, but its large success in practice has shown that it can learn long-distance dependencies

- LSTM vs. GRU: LSTM is usually the default choice. Especially, when enough training data is available and capturing longer distances is important. GRU is faster and more suited for settings with low computation resources

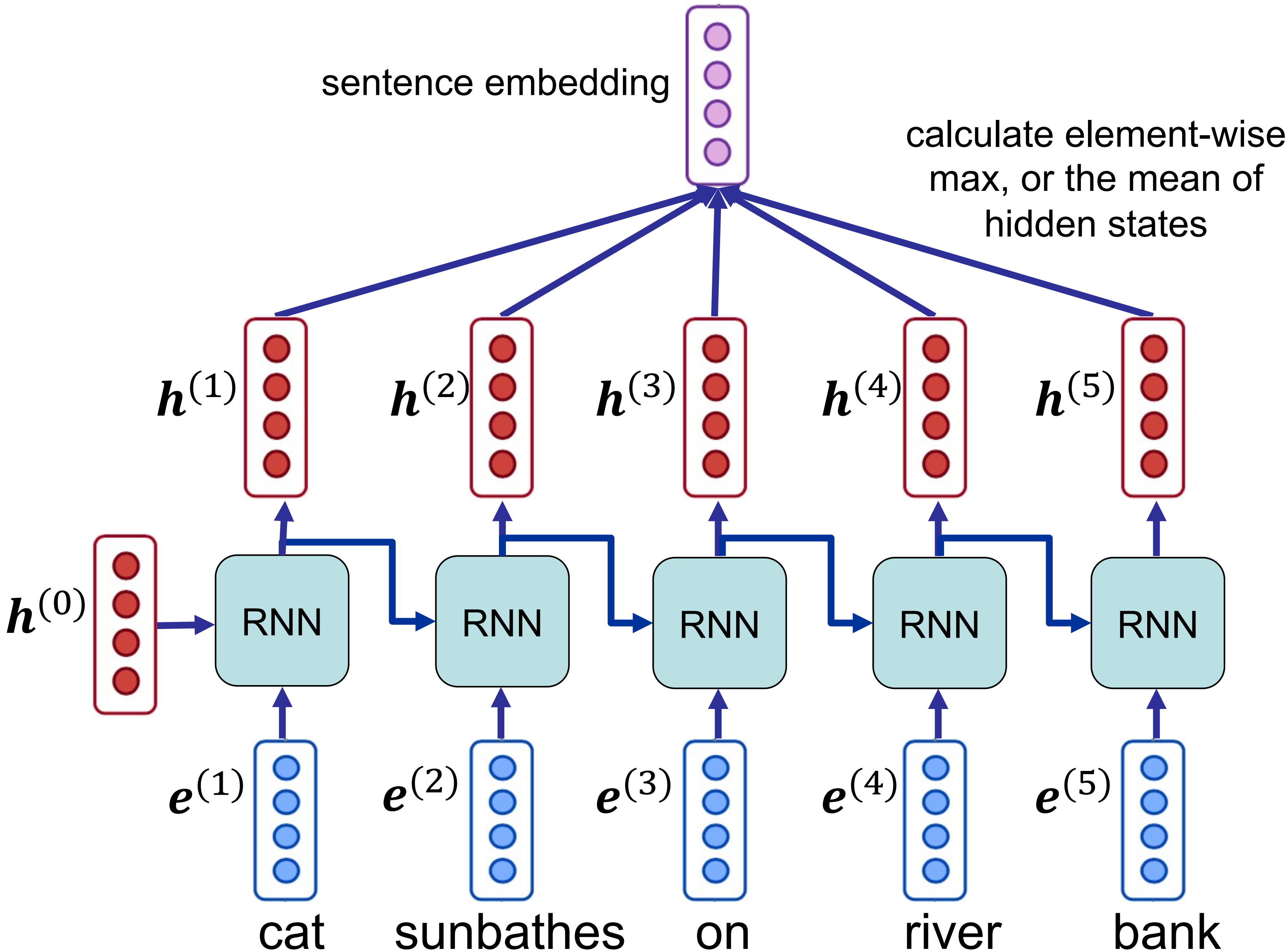
RNN – Compositional embedding



RNN – Compositional embedding

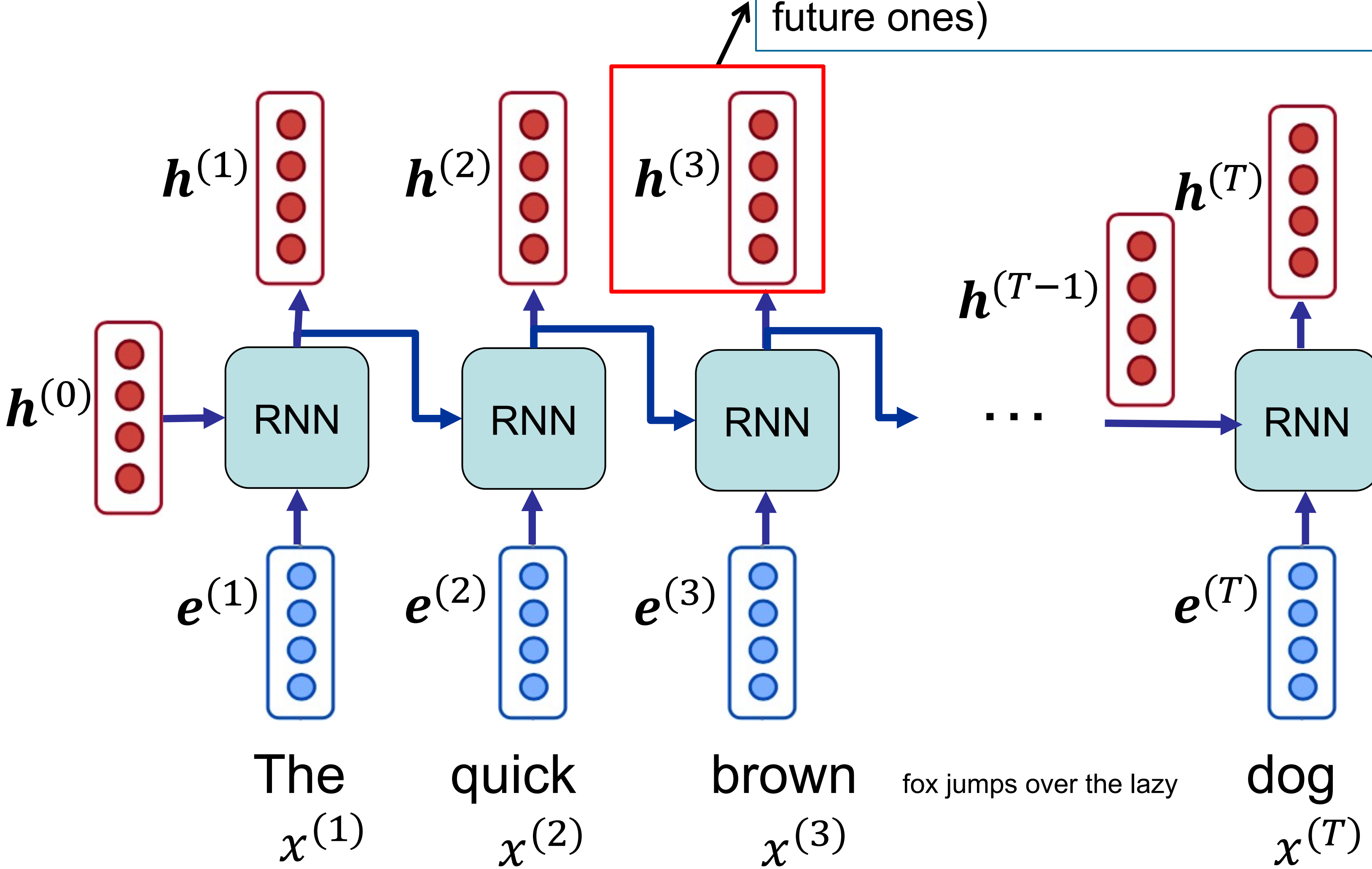


RNN – Compositional embedding



Contextualized Word Embeddings

The contextualized word embedding of "*brown*". However, it only has had access to the previous words (not the future ones)



Bidirectional RNNs

- Bidirectional RNN consists of two RNNs, one reads from the beginning to the end of sequence (forward), and the other reads from the end to the beginning (backward)

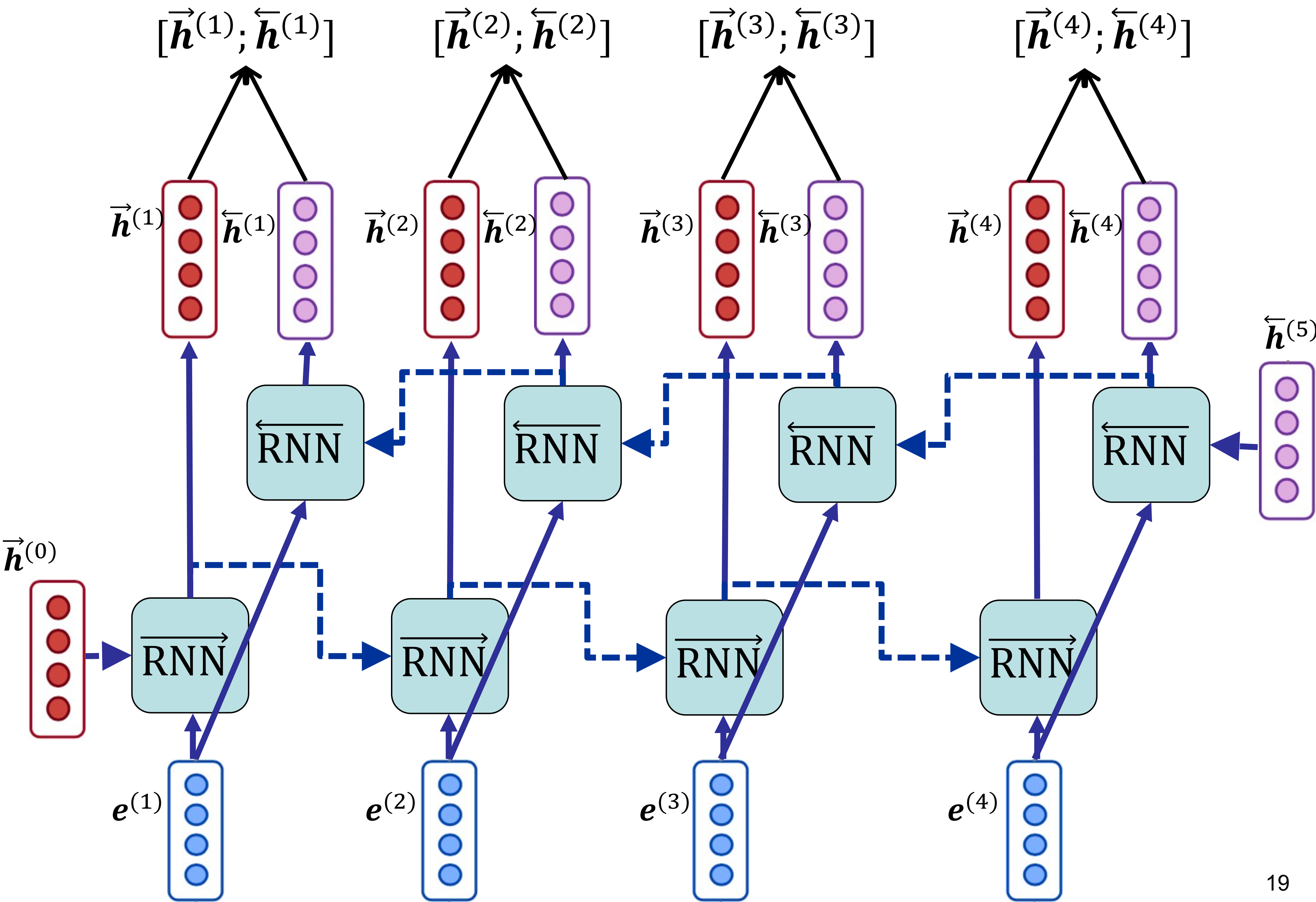
$$\overrightarrow{\boldsymbol{h}}^{(t)} = \overrightarrow{\mathrm{RNN}}\left(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)}\right)$$

$$\overleftarrow{\boldsymbol{h}}^{(t)} = \overleftarrow{\mathrm{RNN}}\left(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)}\right)$$

- Output at each time step is the concatenation of the outputs of both RNNs at that time step:

$$\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$$

- *To remember:* Using bidirectional RNN is only possible when the entire sequence is available


$[\overrightarrow{h}^{(1)}; \overleftarrow{h}^{(1)}]$
$[\overrightarrow{h}^{(2)}; \overleftarrow{h}^{(2)}]$
$[\overrightarrow{h}^{(3)}; \overleftarrow{h}^{(3)}]$
$[\overrightarrow{h}^{(4)}; \overleftarrow{h}^{(4)}]$
$\overrightarrow{h}^{(1)}$
$\overleftarrow{h}^{(1)}$
$\overrightarrow{h}^{(2)}$
$\overleftarrow{h}^{(2)}$
$\overrightarrow{h}^{(3)}$
$\overleftarrow{h}^{(3)}$
$\overrightarrow{h}^{(4)}$
$\overleftarrow{h}^{(4)}$
$\overleftarrow{h}^{(5)}$
$\overleftarrow{\text{RNN}}$
$\overleftarrow{\text{RNN}}$
$\overleftarrow{\text{RNN}}$
$\overleftarrow{\text{RNN}}$
$\overrightarrow{h}^{(0)}$
$\overrightarrow{\text{RNN}}$
$\overrightarrow{\text{RNN}}$
$\overrightarrow{\text{RNN}}$
$\overrightarrow{\text{RNN}}$
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$

Agenda

- RNNs with Gates: LSTM, GRU
- **Document summarization**
- Abstractive summarization with seq2seq
- Extractive summarization with RNNs

Text Summarization

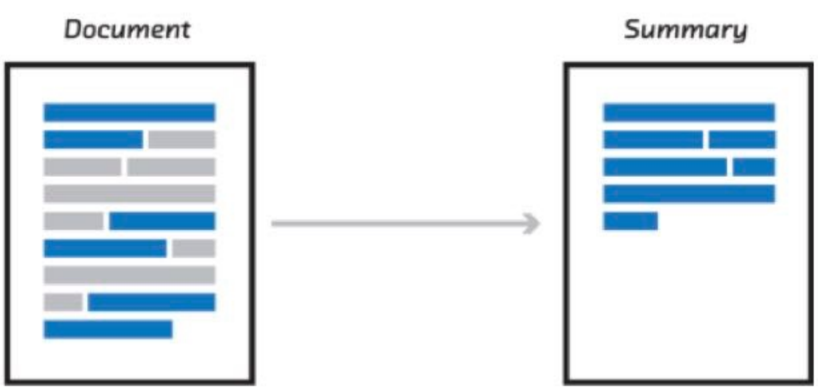
- The task of summarizing the key information content of a text (document) $X = \{x_1, x_2, \dots, x_N\}$ in summary $Y = \{y_1, y_2, \dots, y_M\}$
  - Summary is concise and (much) shorter than document

- Some datasets:
  - *Gigaword*: first one or two sentences of a news article
  - *CNN/DailyMail*: news article
  - *Wikihow*: full how-to article

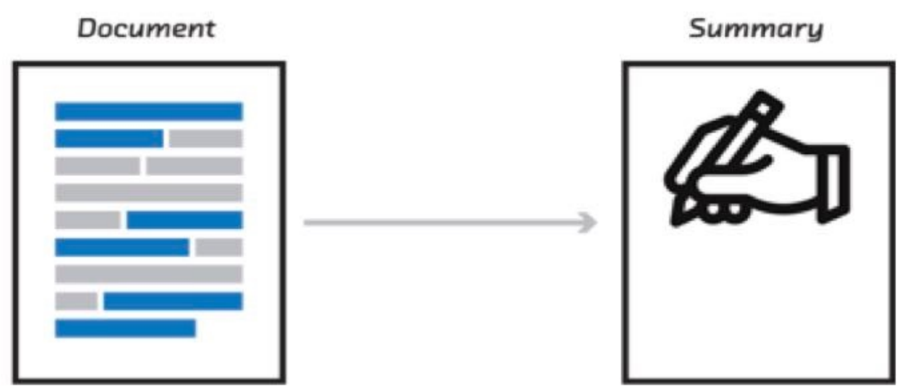
Summarization

- Extractive Summarization
  - Selecting sections (typically sentences) of the document
  - A model decides if a section of document should be selected for the summary
- Abstractive Summarization
  - Writing (generating) new summary text for the document
  - A language generation task conditioned on the document

Extractive Summarization



Abstractive Summarization



Abstractive Summarization - example

Document

SAN FRANCISCO, California (Reuters) -- Sony has cut the price of the PlayStation 3 by $100, or 17 percent, in the United States, a move that should boost the video game console's lackluster sales.
Starting Monday, the current PS3 60 gigabyte model will cost $499 -- a $100 price drop.
The PlayStation 3, which includes a 60-gigabyte hard drive and a Blu-ray high-definition DVD player, will now cost $500, or $20 more than the most expensive version of Microsoft's Xbox 360.
The PS3 still costs twice that of Nintendo's Wii console, whose $250 price and motion-sensing controller have made it a best-seller despite its lack of cutting-edge graphics and hard disk.
"Our initial expectation is that sales should double at a minimum," Jack Tretton, chief executive of Sony Computer Entertainment America, said in an interview.
"We've gotten our production issues behind us on the PlayStation 3, reaching a position to pass on the savings to consumers, and our attitude is the sooner the better."
…

Summary

- Sony drops price of current 60GB PlayStation 3 console by $100 in U.S.
- PS3 still costs twice that of Nintendo's best-selling Wii console, which is $250
- Some expect Microsoft to respond with its first price cuts on the Xbox 360
- Sony to revise PS3 console with bigger 80GB hard drive

Summarization – Evaluation

- **ROUGE-N**: overlap of *n*-grams between output and reference summary
  - **ROUGE-1**: the overlap of ***unigrams***
  - **ROUGE-2**: the overlap of ***bigrams***
  - …

$$\text{ROUGE−N} = \frac{\left| n\text{−grams}\,(\hat{Y}) \cap n\text{−grams}(Y) \right|}{\left| n\text{−grams}(Y) \right|}$$

Y and $\hat{Y}$ are the reference and output summary, respectively. $n\text{−grams}$ retrurns the set of all possible *n*-grams of the given text.

ROUGE: A Package for Automatic Evaluation of Summaries, Lin, 2004 http://www.aclweb.org/anthology/W04-1013

Summarization – Evaluation

- **ROUGE-L** is based on the length of the longest common subsequence between the output and reference summary

$$\text{ROUGE-L} = \frac{LCS\,(\hat{Y}, Y)}{|\hat{Y}|}$$

LCS is the longest common subsequence of the two given texts.

- ROUGE-L
  - does not require consecutive matches but in-sequence matches
    - Example from Wikipedia (LCS=3):
    - Y : this is some text that will be changed
    - $\hat{Y}$: this is the changed text
  - reflects sentence structure
  - don’t need a predefined *n*-gram length

Summarization – Evaluation

- ROUGE (in the discussed definitions) is a recall-based measure
  - ROUGE can also be defined as a precision-based, as well as F-measure

Example

- Y: “police hugged the gunman”
- $\hat{Y}1$: “police hug the gunman”
- $\hat{Y}2$: “the gunman hug police”

- ROUGE-2 of both $\hat{Y}1$ and $\hat{Y}2$ results in the same values!
  - In both $\hat{Y}1$ and $\hat{Y}2$, “the gunman” is the only common bigram with the reference

- $LCS\left(\hat{Y}1, Y\right) =$“police the gunman”$\rightarrow$ ROUGE−L$(\hat{Y}1, Y) = 0.75$
- $LCS\left(\hat{Y}2, Y\right) =$ “the gunman” $\rightarrow$ ROUGE−L$(\hat{Y}2, Y) = 0.5$

Agenda

- RNNs with Gates: LSTM, GRU
- Document summarization
- **Abstractive summarization with seq2seq**
- Extractive summarization with RNNs

Sequence in – sequence out!

- Several NLP tasks are defined as:
  - Given the source sequence $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(L)}\}$
  - Create/Generate the target sequence $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(T)}\}$

X		Y
Was mich nicht umbringt, macht mich stärker. F. Nietzsche	Machine Translation →	What does not kill me makes me stronger.
Then the woman went to the bank to deposit her cash .	POS Tagging →	RB DT NN VBD TO DT NN TO VB PRP$ NN .
How tall is Stephansdom?	Semantic parsing →	[Heightof, ., Stephansdom]

Sequence in – sequence out!

- Tasks such as:
  - Machine Translation (source language → target language)
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Code generation (natural language → SQL/Python code)
  - Named entity recognition
  - Dependency/semantic/ POS Parsing (input text → output parse as sequence)

but also …

  - Image captioning (image → caption)
  - Automatic Speech Recognition (speech → manuscript)

Image captioning →

some elephants standing around a tall tree

Sequence-to-sequence model

- Sequence-to-sequence model (aka seq2seq) is the neural network architecture to approach …
  - given the source sequence $X = \{x^{(1)}, x^{(2)}, \dots, x^{(L)}\}$,
  - generate the target sequence $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(T)}\}$

- A seq2seq model typically creates a model to estimate the conditional probability:

$$P(Y|X)$$

- and then generates a new sequence Y^* by solving:

$$Y^* = \underset{Y}{\operatorname{argmax}} P(Y|X)$$

Seq2seq model

- A seq2seq model in many cases can be as a conditional Language Model
- It calculates the probability of the next word of target sequence, conditioned on the previous words of target sequence and the source sequence:

for $y^{(1)} \rightarrow P(y^{(1)}|X)$

for $y^{(2)} \rightarrow P(y^{(2)}|X, y^{(1)})$

…

for $y^{(i)} \rightarrow P(y^{(i)}|X, y^{(1)}, \ldots, y^{(i-1)})$

… and for whole the target sequence:

$$P(Y|X) = P\left(y^{(1)}\middle|X\right) \times P\left(y^{(2)}\middle|X, y^{(1)}\right) \times \cdots \times P\left(y^{(T)}\middle|X, y^{(1)}, \ldots, y^{(T-1)}\right)$$

$$P(Y|X) = \prod_{t=1}^{T} P(y^{(t)}|X, y^{(1)}, \ldots, y^{(t-1)})$$

Seq2seq – steps

- Like Language Modeling, we …
- … design a model that predicts the probabilities of the next words of the target sequence, one after each other (in auto-regressive fashion): $P(y^{(i)}|X, y^{(1)}, \dots, y^{(i-1)})$
- We train the model by maximizing these probabilities for the correct next words, appearing in training data
- At inference time (or during decoding), we use the model to generate new target sequences, that have high generation probabilities: $P(Y|X)$

Seq2seq with two RNNs

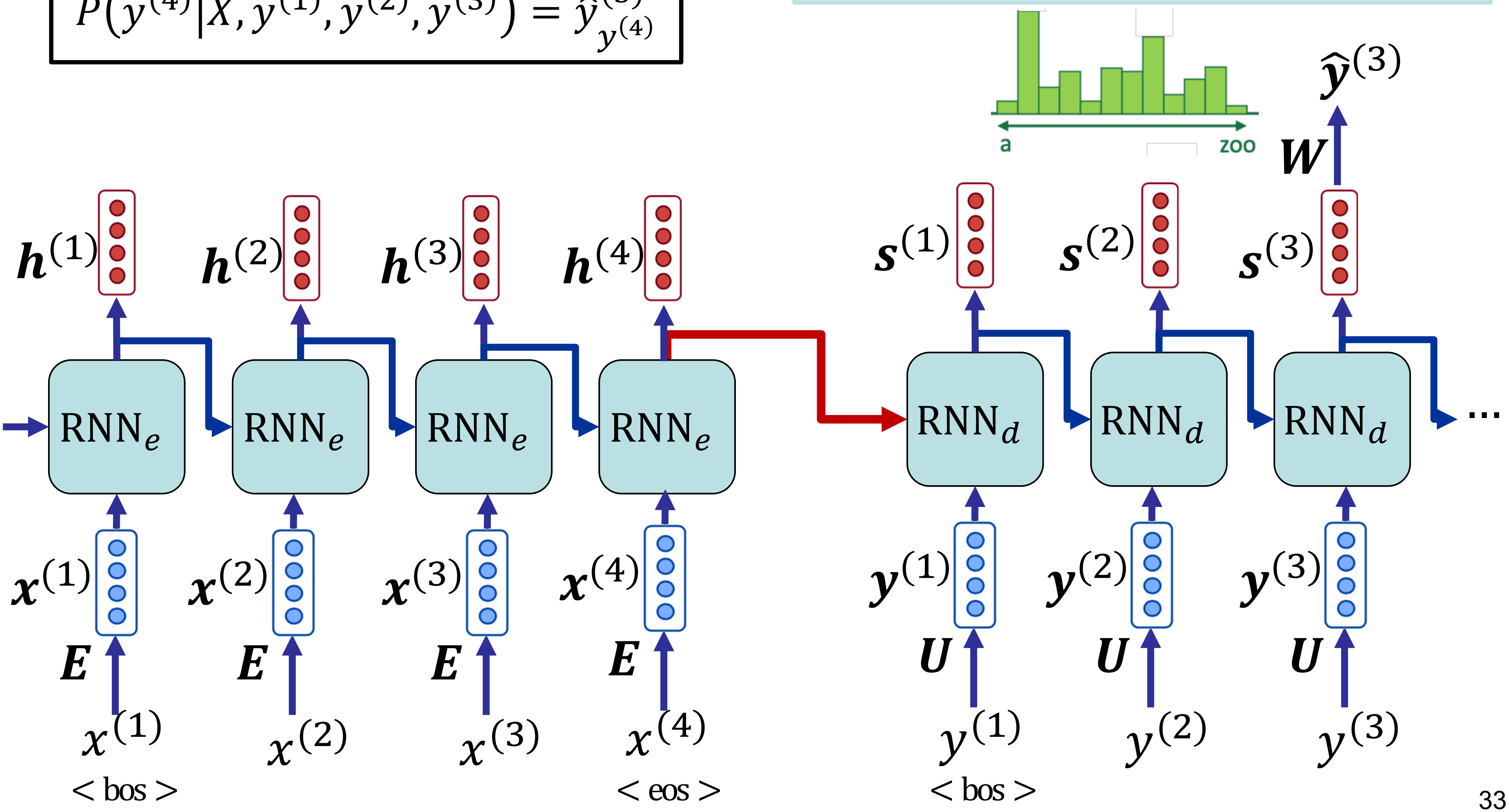
ENCODER

DECODER

Probability of appearance of the next target word:

$$P\left(y^{(4)} \middle| X, y^{(1)}, y^{(2)}, y^{(3)}\right) = \hat{y}^{(3)}_{y^{(4)}}$$

$\hat{\boldsymbol{y}}^{(i)}$: predicted probability distribution of the next target word, given the source sequence and previous target words



Seq2seq with two RNNs – formulation

- There are two sets of vocabularies
  - $\mathbb{V}_e$ is the set of vocabularies for source sequences
  - $\mathbb{V}_d$ is the set of vocabularies for target sequences

ENCODER

- Encoder embedding
  - Encoder embeddings for source words ($\mathbb{V}_e$) $\rightarrow$ $\boldsymbol{E}$
  - Embedding of the source word $x^{(l)}$ (at time step l) $\rightarrow$ $\boldsymbol{x}^{(l)}$

- Encoder RNN:

$$\boldsymbol{h}^{(l)} = \mathrm{RNN}_e\left(\boldsymbol{h}^{(l-1)}, \boldsymbol{x}^{(l)}\right)$$

Parameters are shown in red

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks." *Advances in Neural Information Processing Systems* 27 (2014): 3104-3112.

Seq2seq with two RNNs – formulation

DECODER

- Decoder embedding
  - Decoder embeddings *at input* for target words ($\mathbb{V}_d$) $\longrightarrow$ $\boldsymbol{U}$
  - Embedding of the target word $y^{(t)}$ (at time step t) $\rightarrow \boldsymbol{y}^{(t)}$

- Decoder RNN

$$\boldsymbol{s}^{(t)} = \mathrm{RNN}_d(\boldsymbol{s}^{(t-1)}, \boldsymbol{y}^{(t)})$$

  - The values of the last hidden state of the encoder RNN are passed to the initial hidden state of the decoder RNN:

$$\boldsymbol{s}^{(0)} = \boldsymbol{h}^{(L)}$$

Parameters are shown in red

Seq2seq with two RNNs – formulation

DECODER

- Decoder output prediction
  - Predicted probability distribution of words at the next time step:

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{W}\boldsymbol{s}^{(t)} + \boldsymbol{b}\right) \in \mathbb{R}^{|\mathbb{V}_d|}$$

  - Probability of the next target word (at time step $t+1$):

$$P\left(y^{(t+1)} \middle| X, y^{(1)}, \dots, y^{(t-1)}, y^{(t)}\right) = \hat{y}^{(t)}_{y^{(t+1)}}$$

Parameters are shown in red

Training Seq2seq

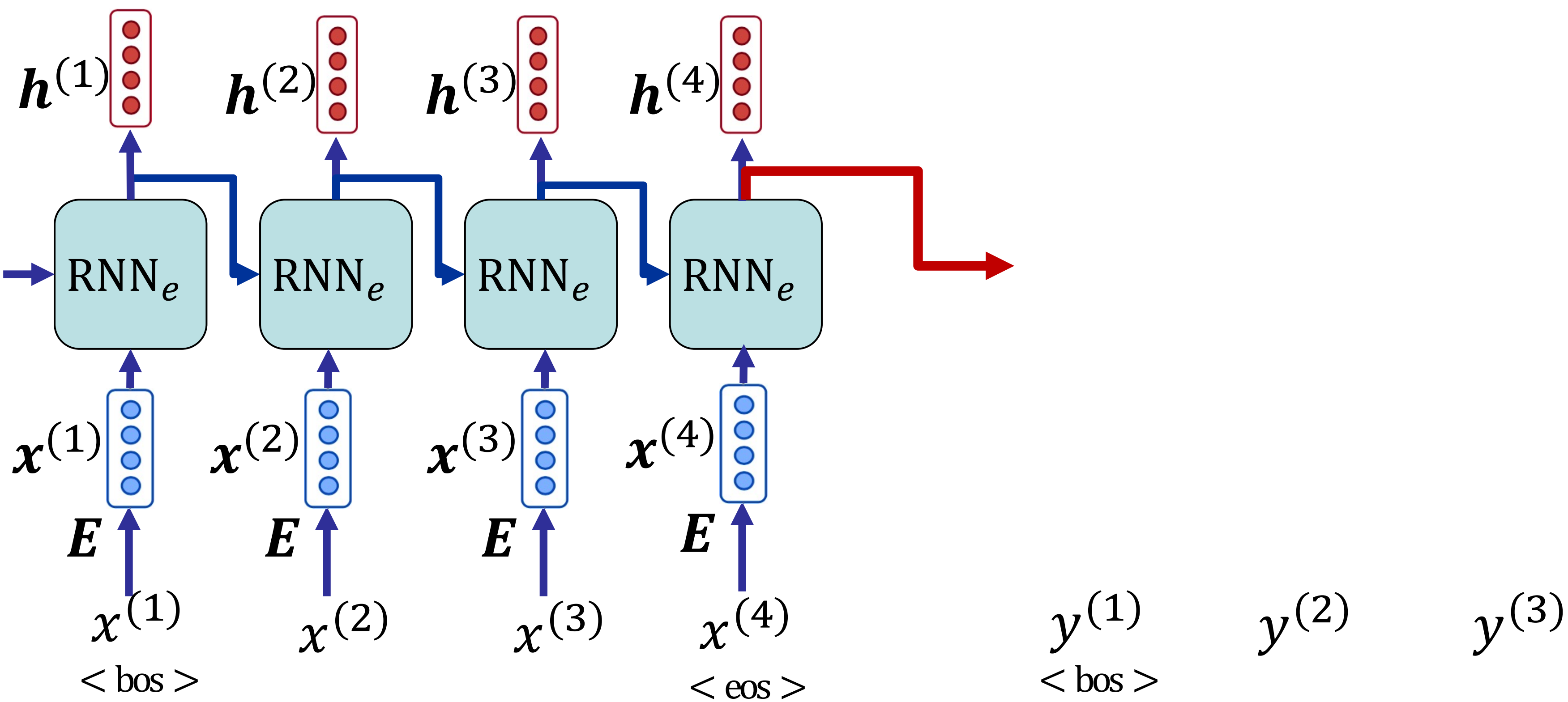
- Training a seq2seq is the same as training a Language Model
  - We predict the next word, calculate loss, backpropagate, and update parameters
  - Since seq2seq is an end-to-end model, gradient flows from loss to all parameters (both RNNs and embeddings)

- Loss function: Negative Log Likelihood of the predicted probability of the correct next target word $y^{(t+1)}$

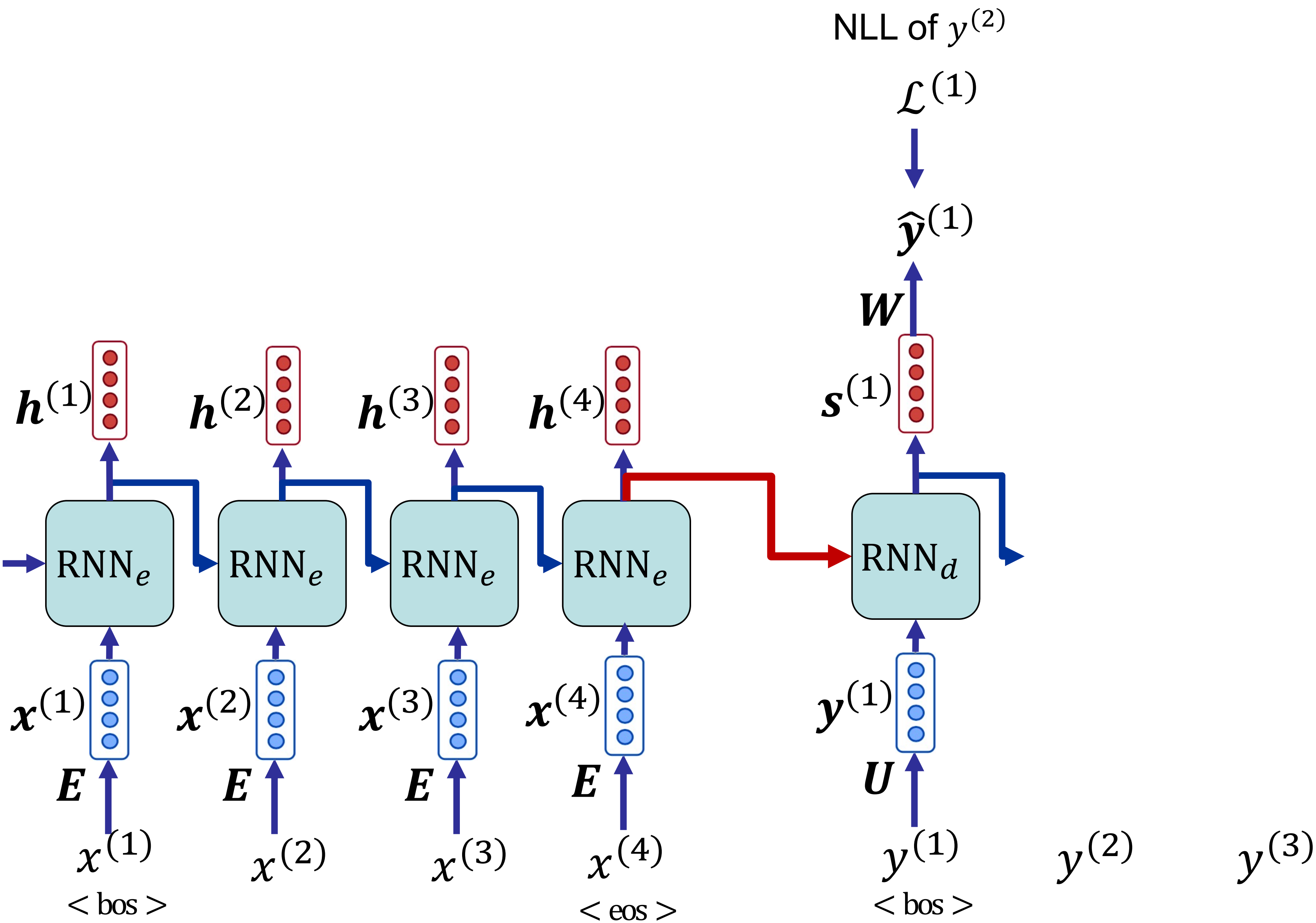
$$\mathcal{L}^{(t)} = -\log \hat{y}^{(t)}_{y^{(t+1)}} = -\log P\left(y^{(t+1)} \middle| X, y^{(1)}, \dots, y^{(t)}\right)$$

- Overall loss: $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{(t)}$

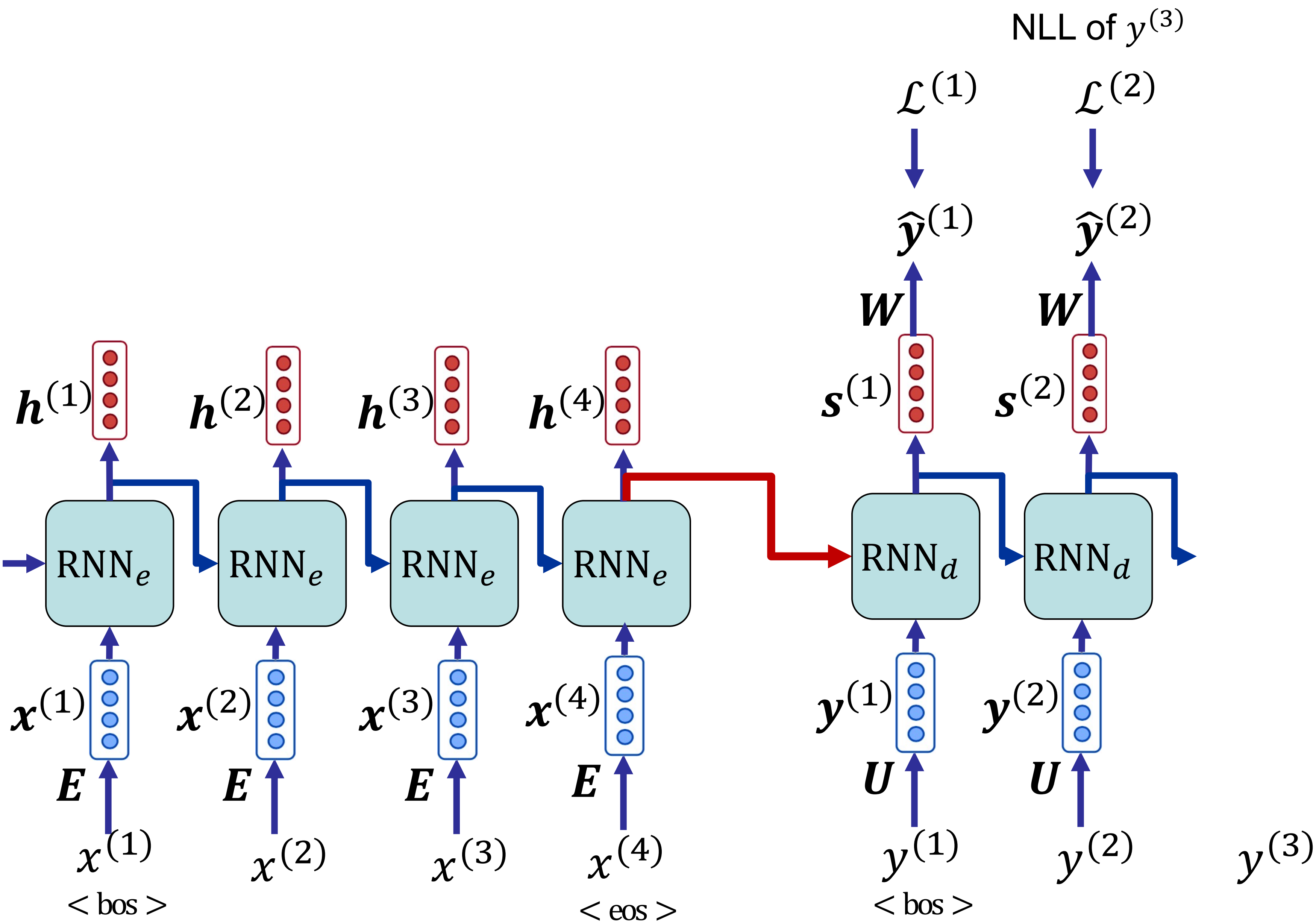
Training Seq2seq



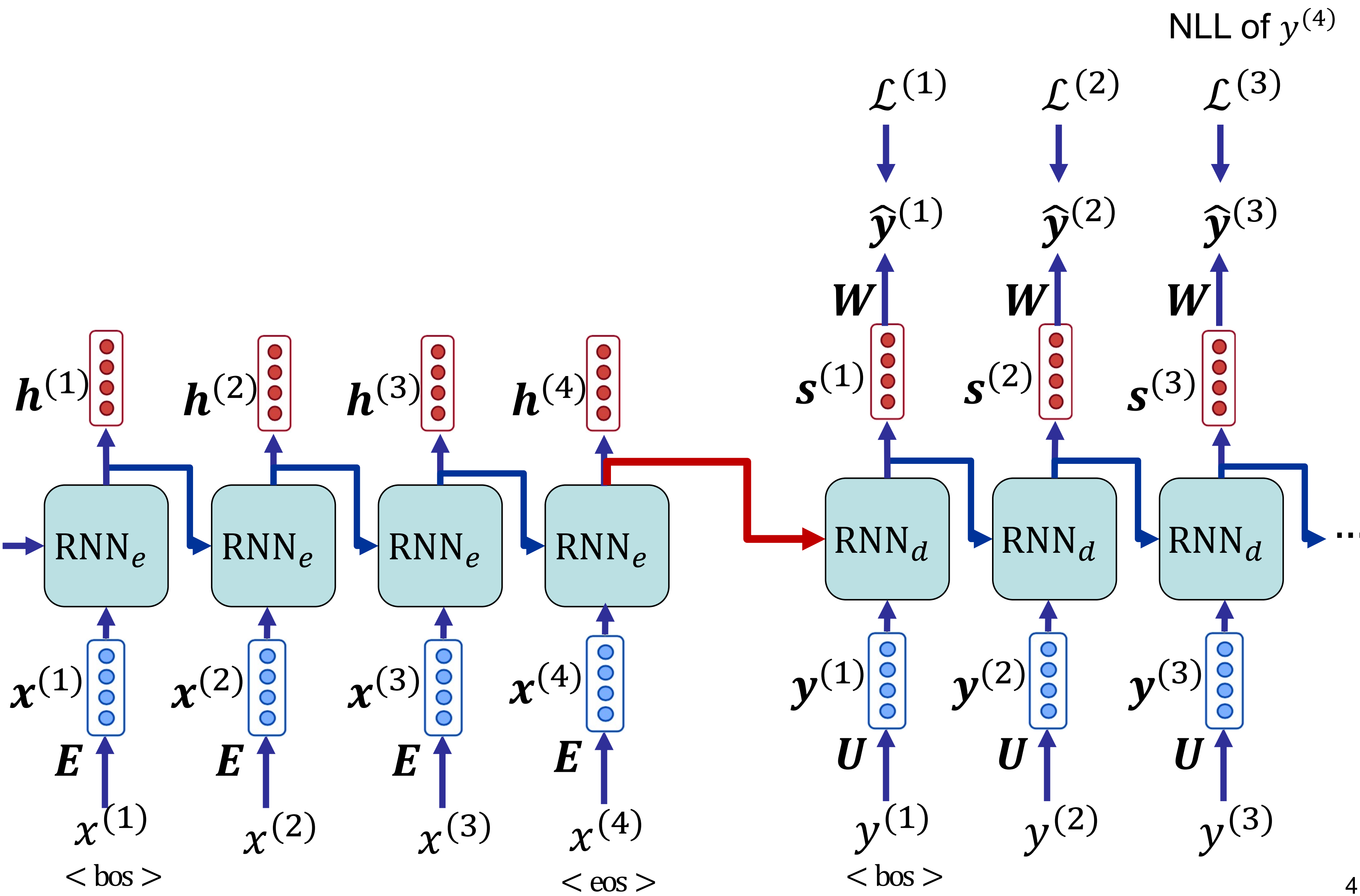
Training Seq2seq



Training Seq2seq



Training Seq2seq



Parameters

- Encoder embeddings $\boldsymbol{E} \rightarrow |\mathbb{V}_e| \times d_e$
- Encoder RNN parameters
- Decoder embeddings $\boldsymbol{U} \rightarrow |\mathbb{V}_d| \times d_u$
- Decoder RNN parameters
- Decoder output projection $\boldsymbol{W} \rightarrow d_w \times |\mathbb{V}_d|$

- bias terms are discarded
- d_e, d_u, d_w are embedding dimensions
- RNNs can be an LSTM, GRU, or vanilla (Elman) RNN

Practical points: vocabs & embeddings

- In summarization
  - Encoder and decoder vocabularies are typically the same set, as they are in the same language
    - It is different for example in neural machine translation, as there, encoder and decoder vocabularies belong to two different languages
  - Encoder and decoder embeddings ($\boldsymbol{E}$ and $\boldsymbol{U}$) can also share parameters

- Weight tying
  - can be done by sharing the parameters of $\boldsymbol{U}$ and $\boldsymbol{W}$ in decoder

Decoding

Recap

- After training, we use the model to generate a target sequence given the source sequence (decoding). We aim to find the optimal output sequence Y^* that maximizes $P(Y|X)$:

$$Y^* = \operatorname*{argmax}_{Y} P(Y|X)$$

where $P(Y|X)$ for any arbitrary $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(T)}\}$ is:

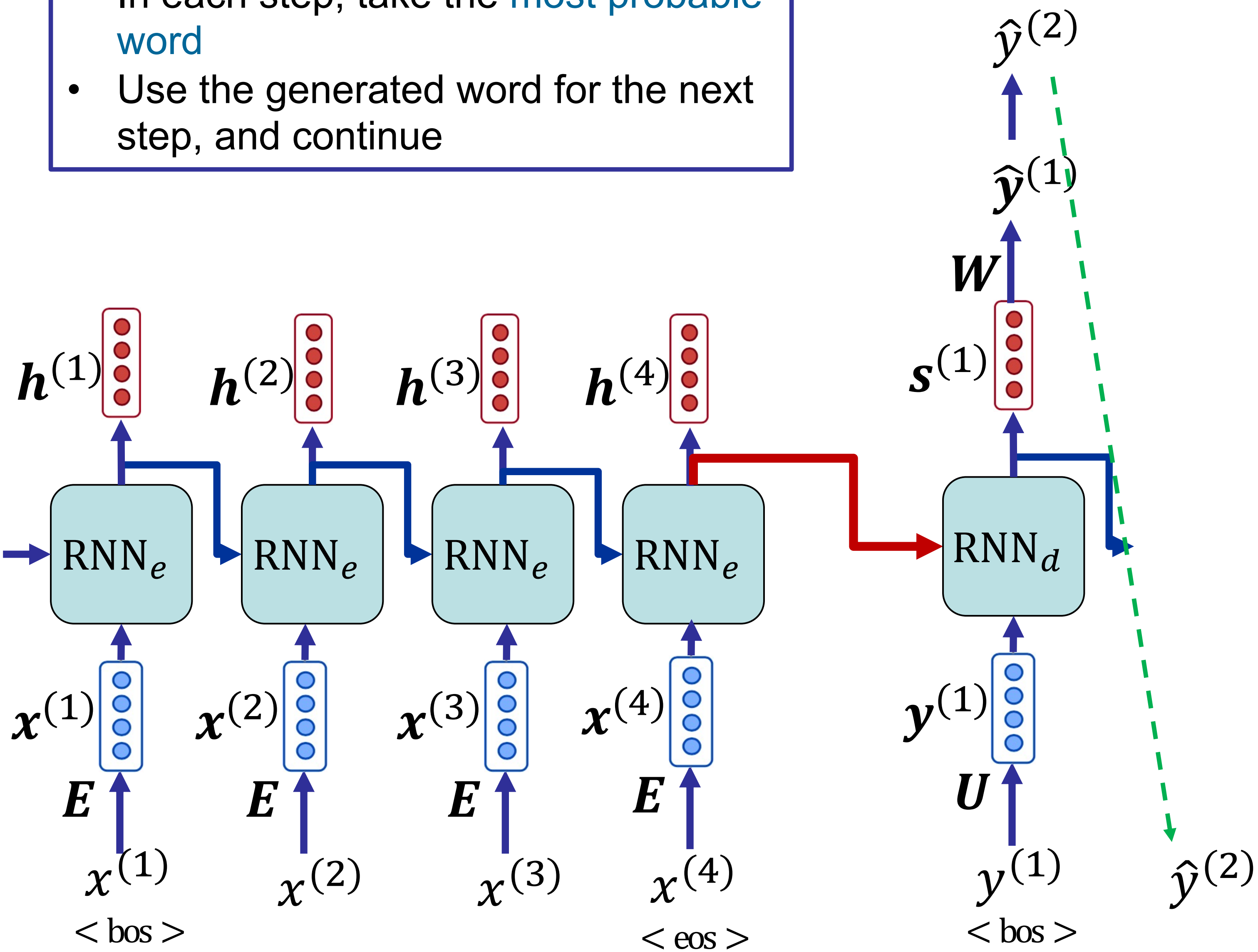
$$P(Y|X) = \prod_{t=1}^{T} P(y^{(t)}|X, y^{(1)}, \dots, y^{(t-1)})$$

- *Question: among all possible Y sequences, how can we find Y^*?*

A first approach: Greedy decoding

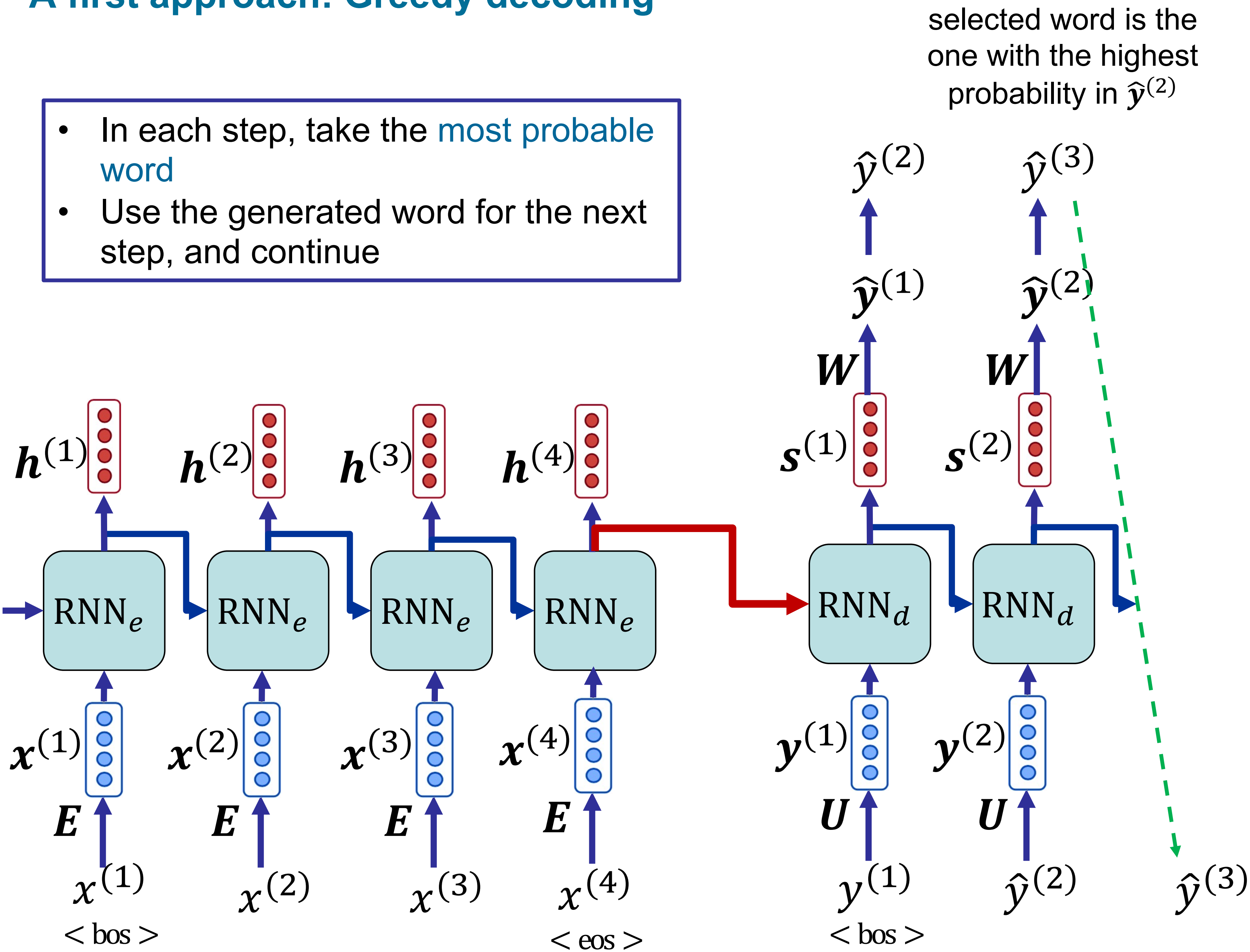
- In each step, take the most probable word
- Use the generated word for the next step, and continue

selected word is the one with the highest probability in $\hat{\boldsymbol{y}}^{(1)}$



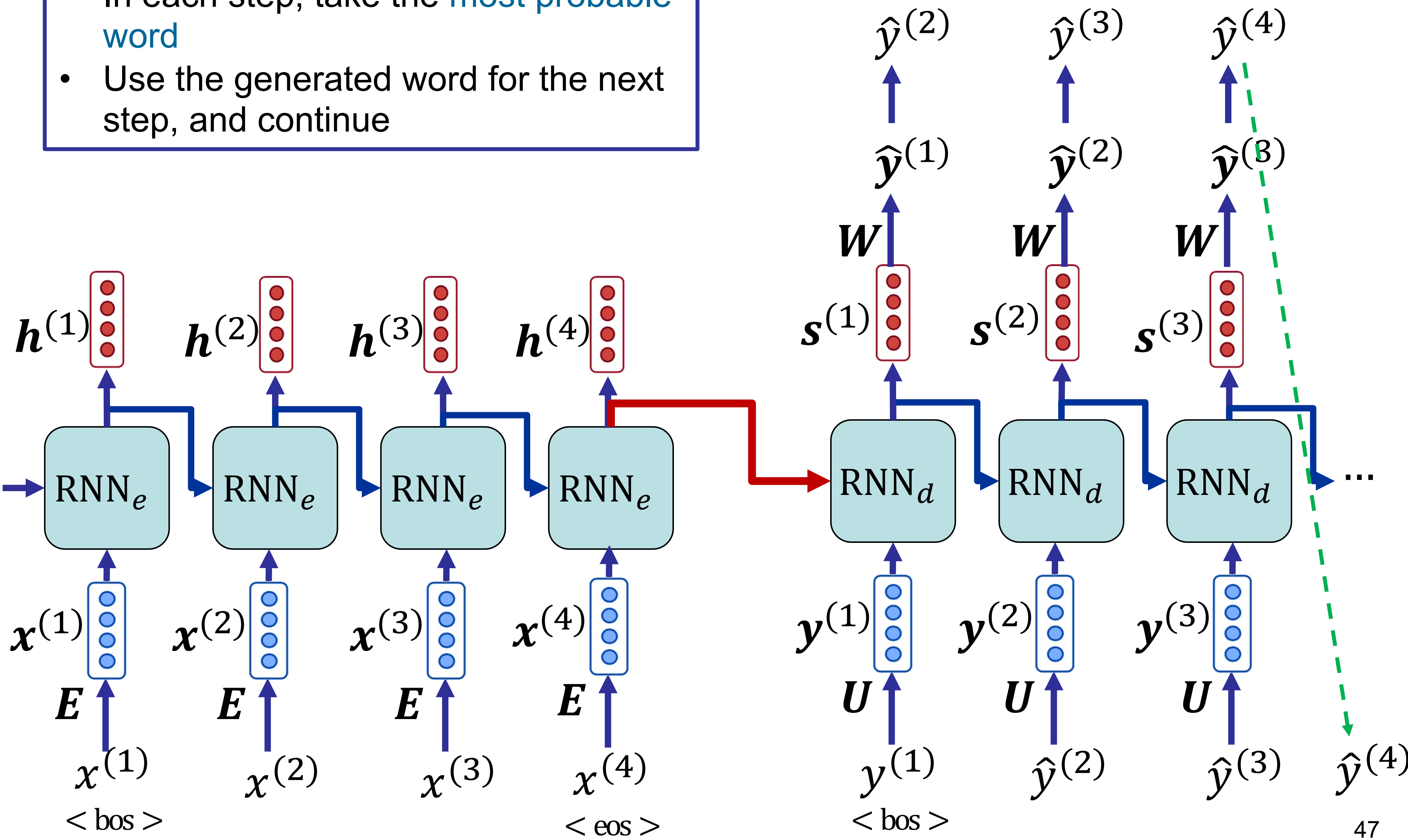
A first approach: Greedy decoding

- In each step, take the most probable word
- Use the generated word for the next step, and continue



A first approach: Greedy decoding

- In each step, take the most probable word
- Use the generated word for the next step, and continue



Decoding

- Greedy decoding
  - Fast but …
  - … decisions are only based on immediate local knowledge
  - A non-optimal local decision can get propagated
  - It does not explore other decoding possibilities

- Exhaustive search decoding
  - We *can* compute all possible decodings
  - It means a decoding tree with $|\mathbb{V}_d| \times T$ leaves!
  - Far too expensive!

- Beam search decoding
  - A compromise between exploration and exploitation!

Beam search decoding

- Core idea: on each time step of decoding, keep only k most probable intermediary sequences (hypotheses)
  - k is the beam size (in practice around 5 to 10)

- To do it, beam search calculates of the following score for each hypothesis till time step l (denoted as $y^{(1\ldots l)}$) :

$$\mathrm{score}\left(y^{(1\ldots l)}\right) = \log P\left(y^{(1\ldots l)} \middle| X\right) = \sum_{i=1}^{l} \log P(y^{(i)} | X, y^{(1)}, \ldots, y^{(i-1)})$$

- In each decoding step, we only keep k hypotheses with the highest scores, and don't continue the rest

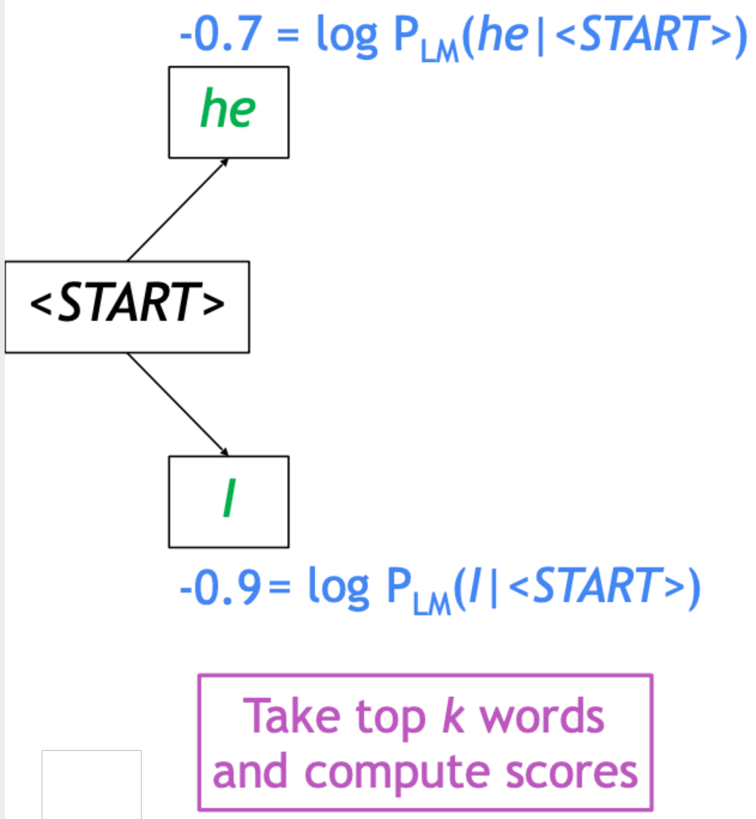
Beam search decoding – example

<START>

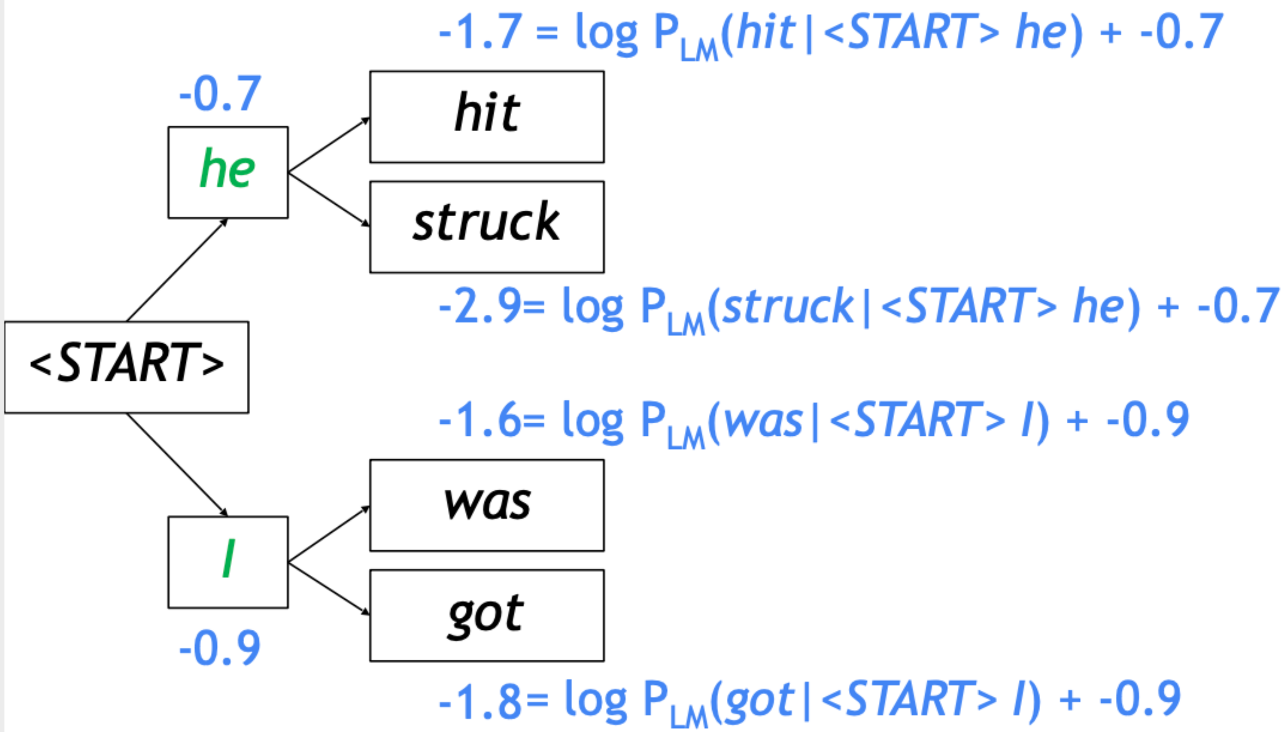
Calculate prob dist of next word

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example

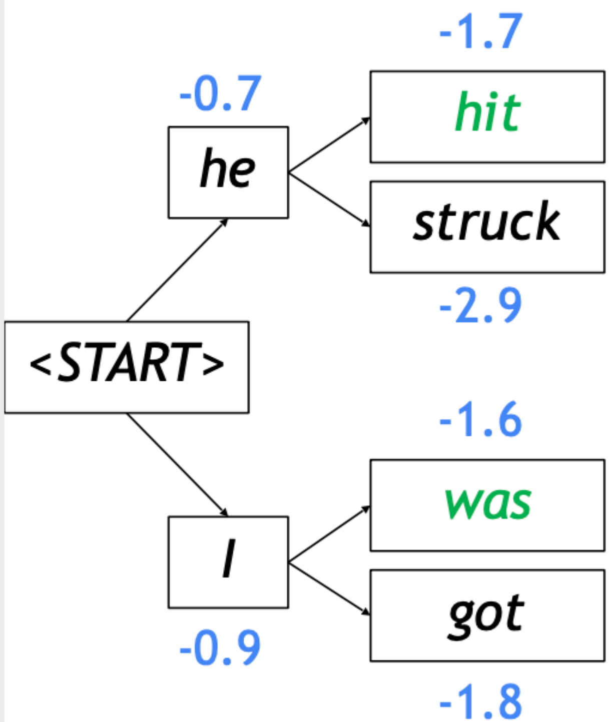


Beam search decoding – example



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

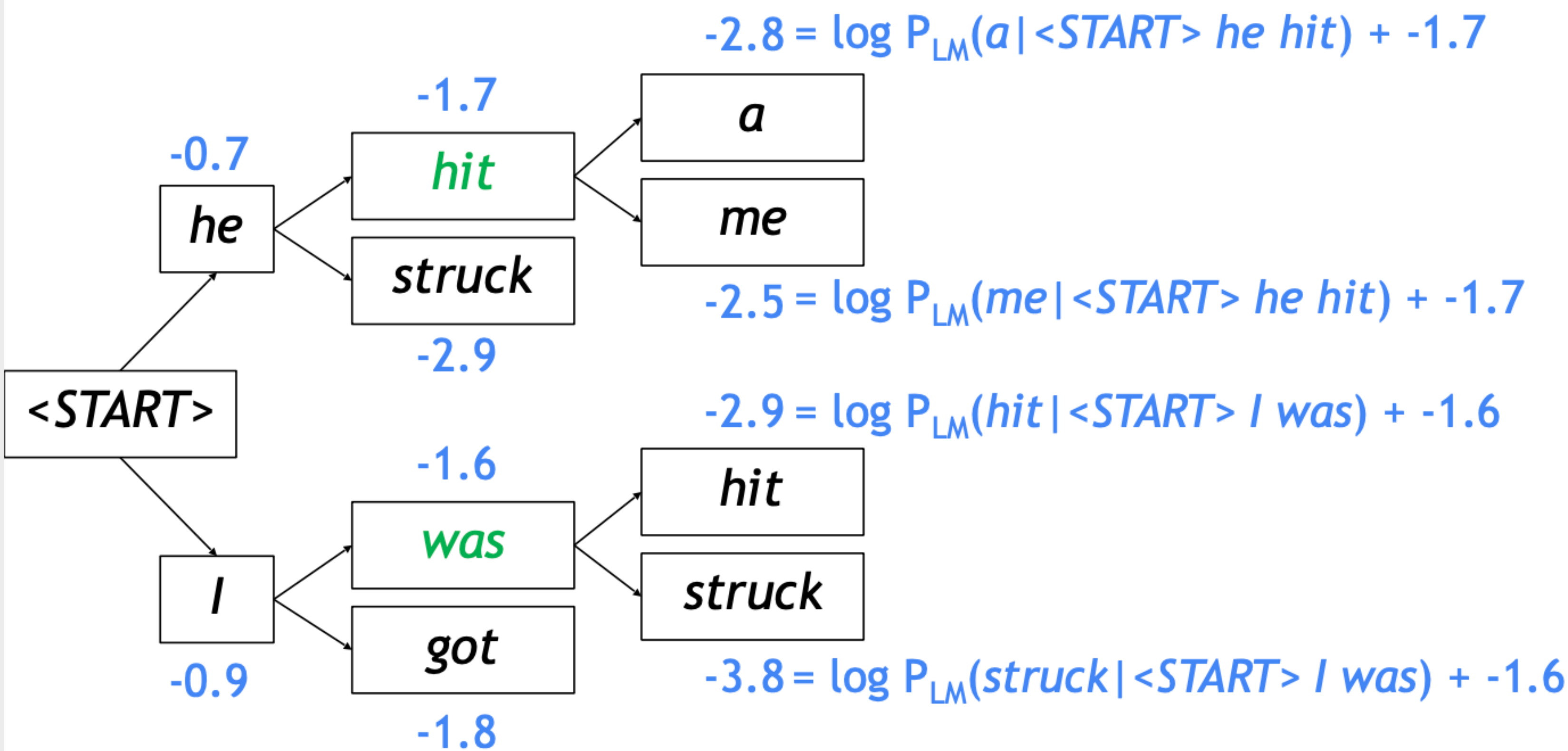
Beam search decoding – example



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

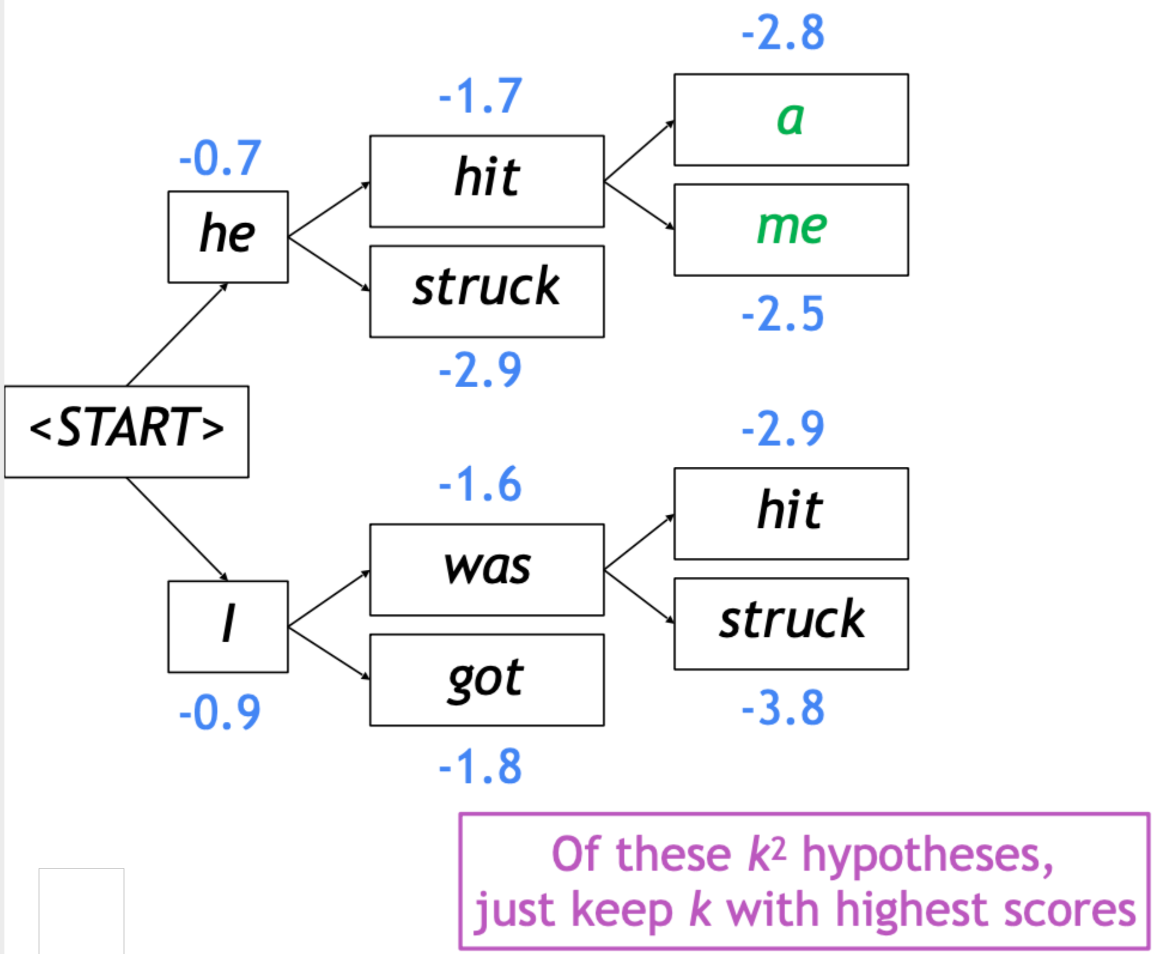
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example

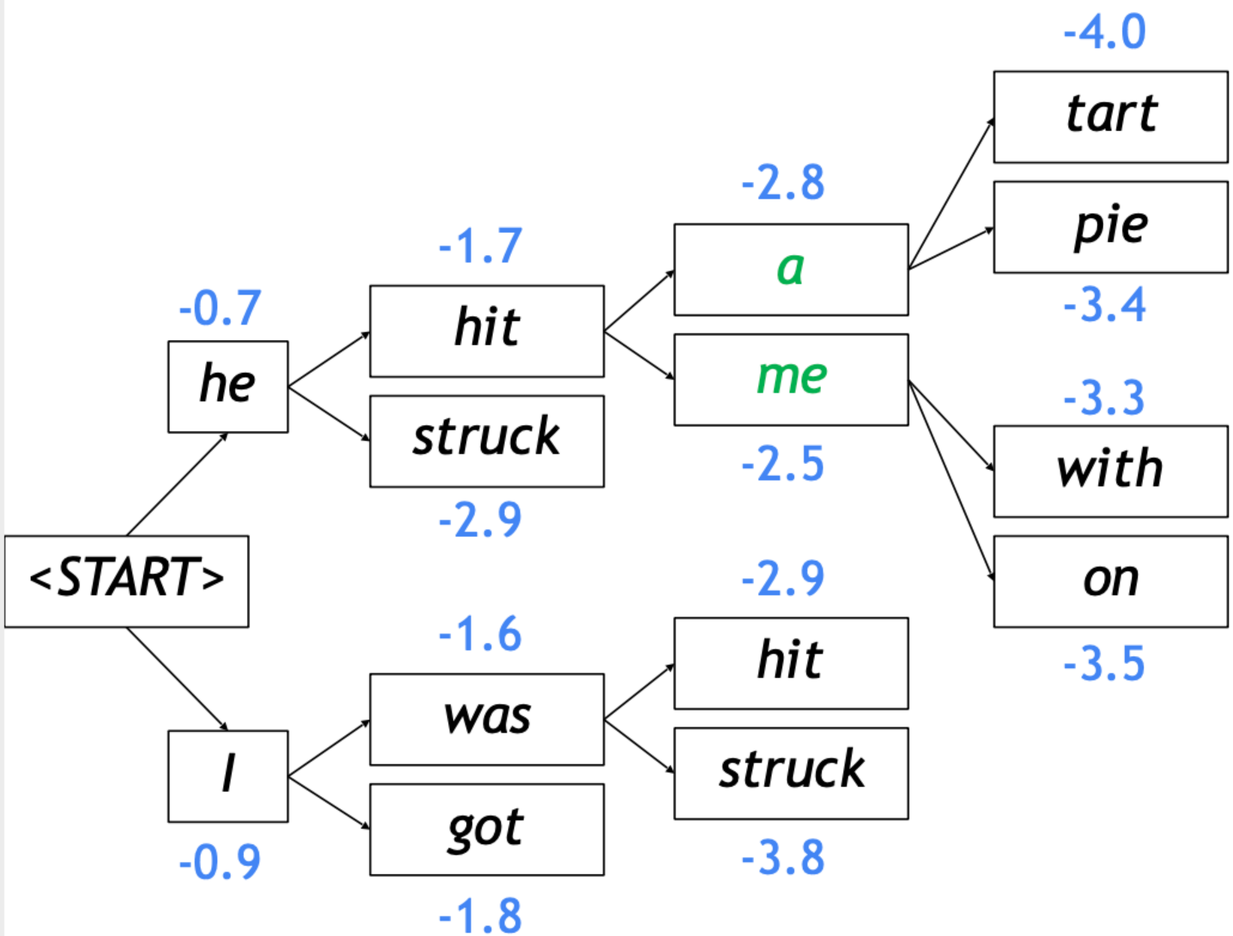


For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



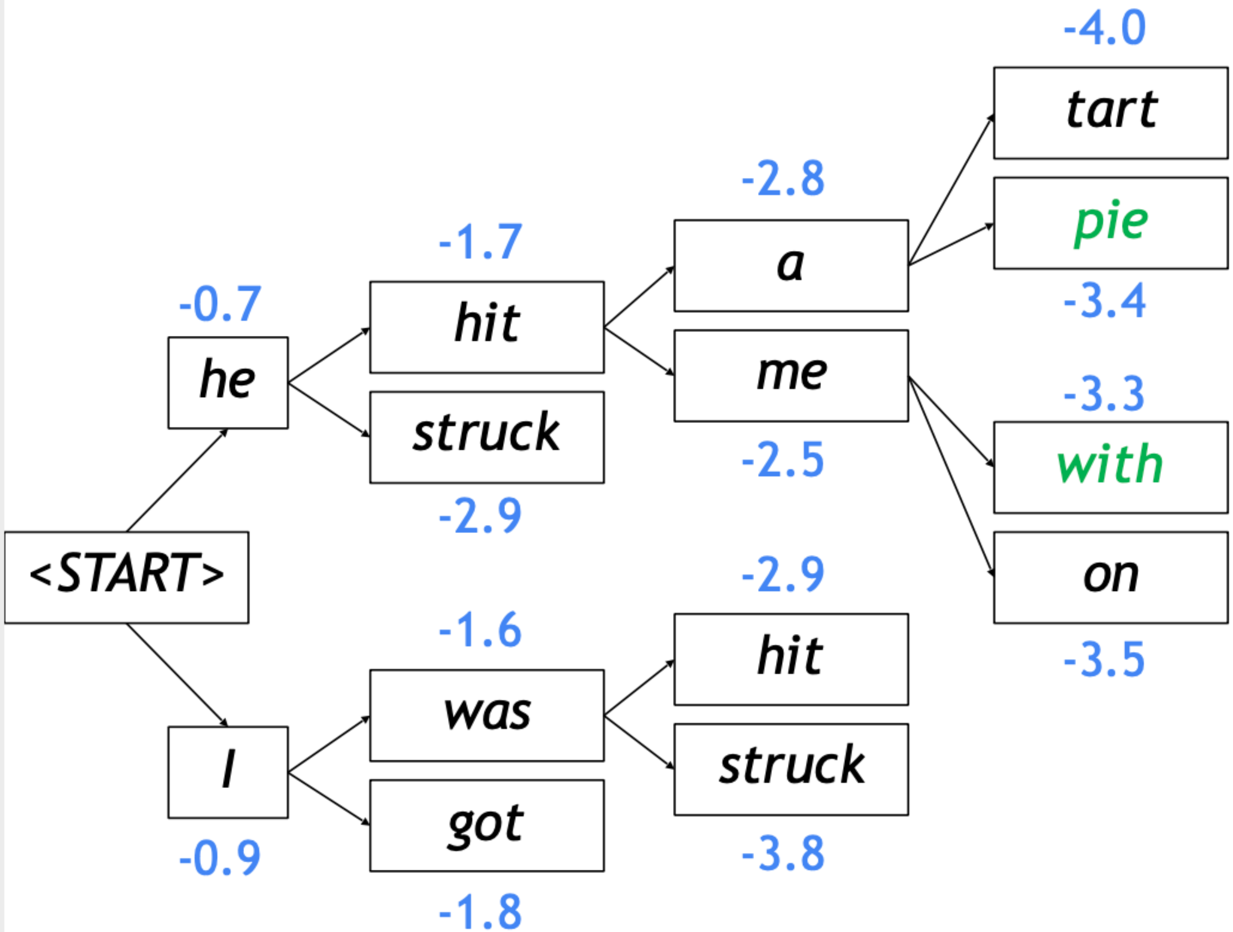
Beam search decoding – example



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

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

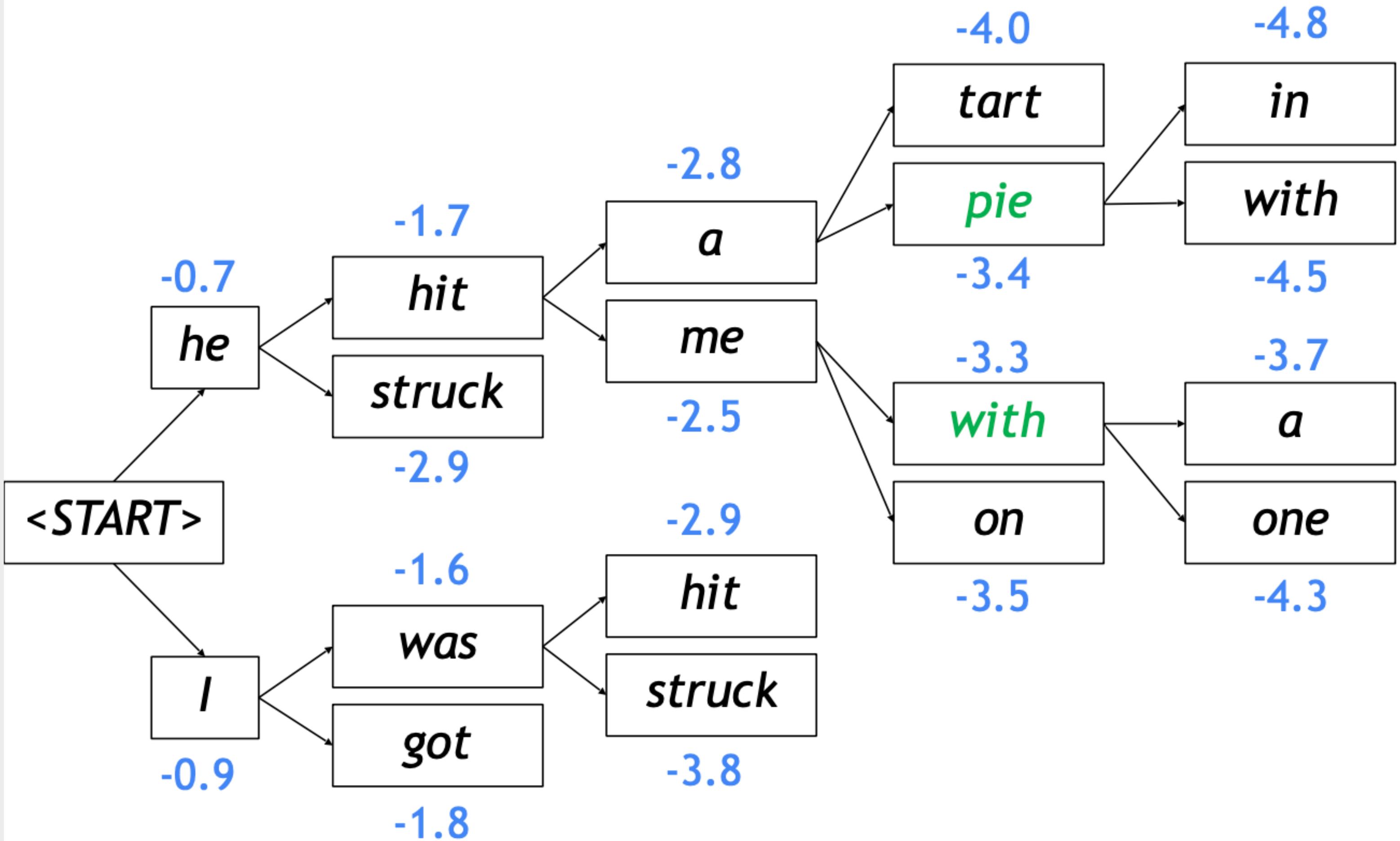
Beam search decoding – example



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

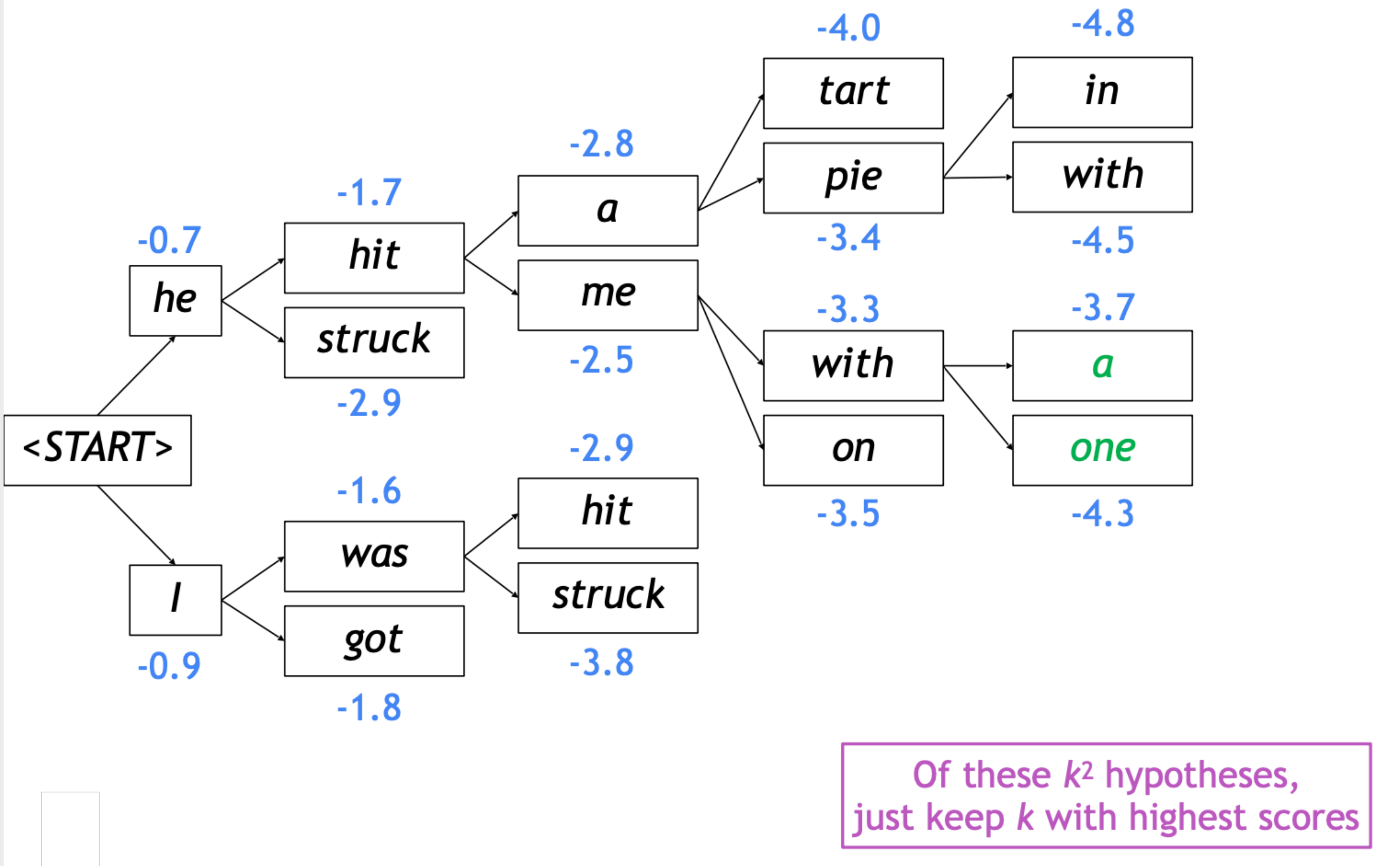
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



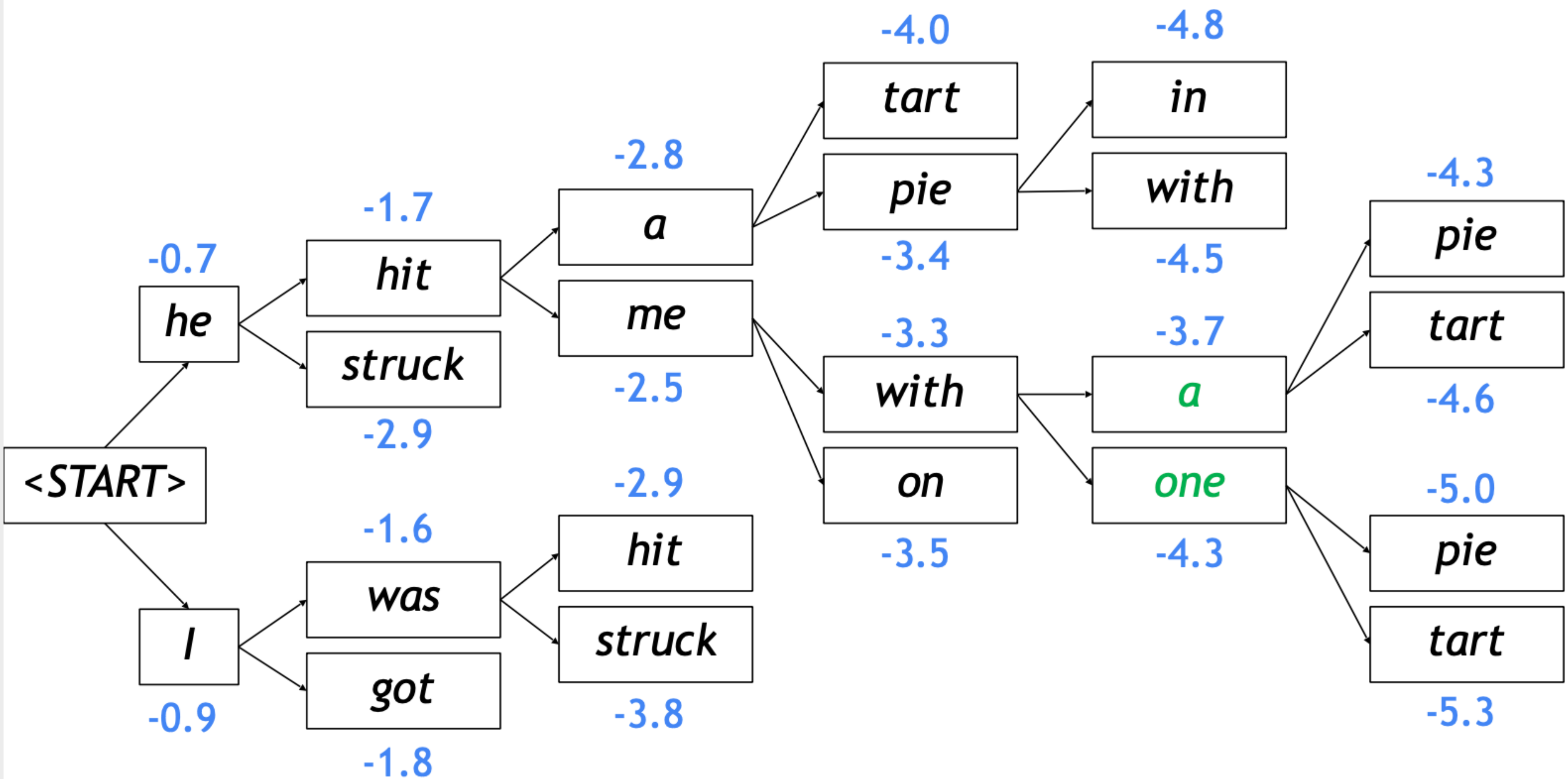
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



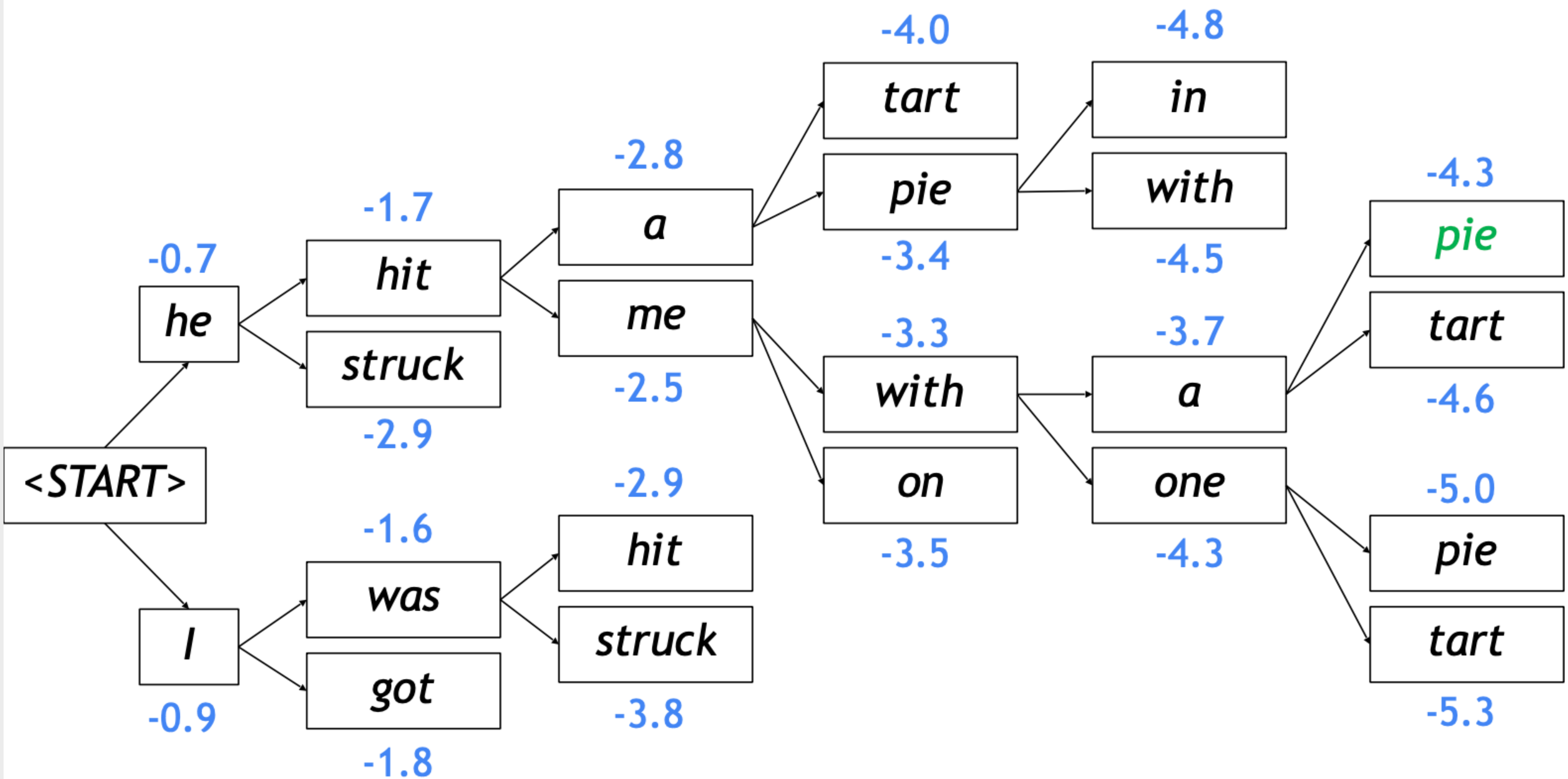
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



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

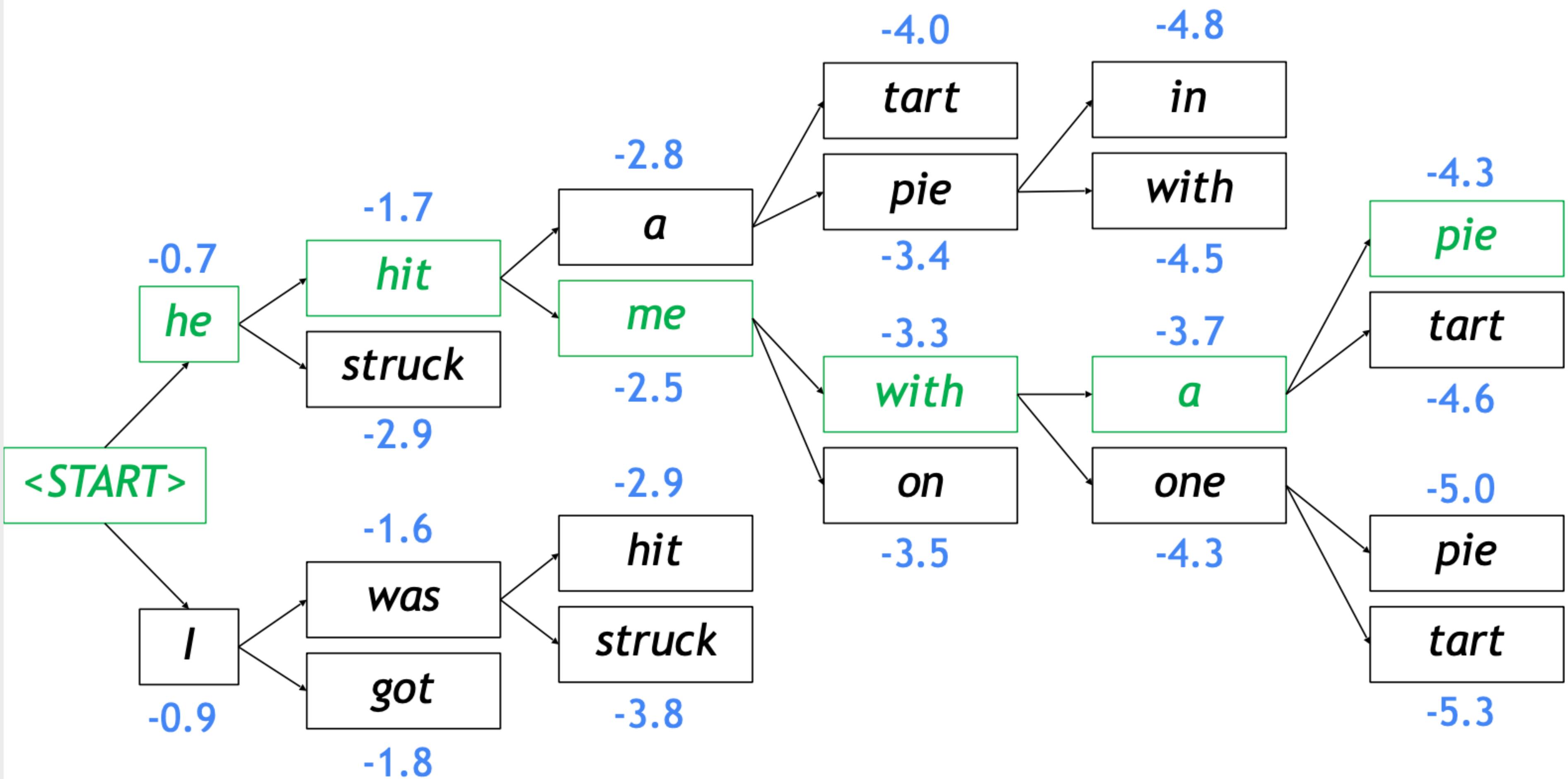
Beam search decoding – example



This is the top-scoring hypothesis!

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



Backtrack to obtain the full hypothesis

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – last words!

- Achieving the optimal solution is not guaranteed …
  - ... but it is much more efficient than exhaustive search decoding

- Stop criteria:
  - Each hypothesis continues till reaching the <eos> token
  - Usually beam search decoding continues until:
    - We reach a cutoff timestep T (a hyperparameter), or
    - We have at least n completed hypotheses (another hyperparameter)

Agenda

- RNNs with Gates: LSTM, GRU
- Document summarization
- Abstractive summarization with seq2seq
- **Extractive summarization with RNNs**

The content of this section will not be a part of the final evaluation

Extractive Summarization – paper walkthrough

<u>Problem definition</u>

- Document D contains L sentences: $D = \{s_1, s_2, \dots, s_L\}$
- Each sentence s_i contains a set words, each annotated with $x^{(i)}$
- Extractive summarization objective: select l sentences of the document such that they provide the "*best*" summary (concise and comprehensive)

Zhou, Q., Yang, N., Wei, F., Huang, S., Zhou, M., & Zhao, T.. **Neural Document Summarization by Jointly Learning to Score and Select Sentences**. In *Proceedings of ACL* 2018

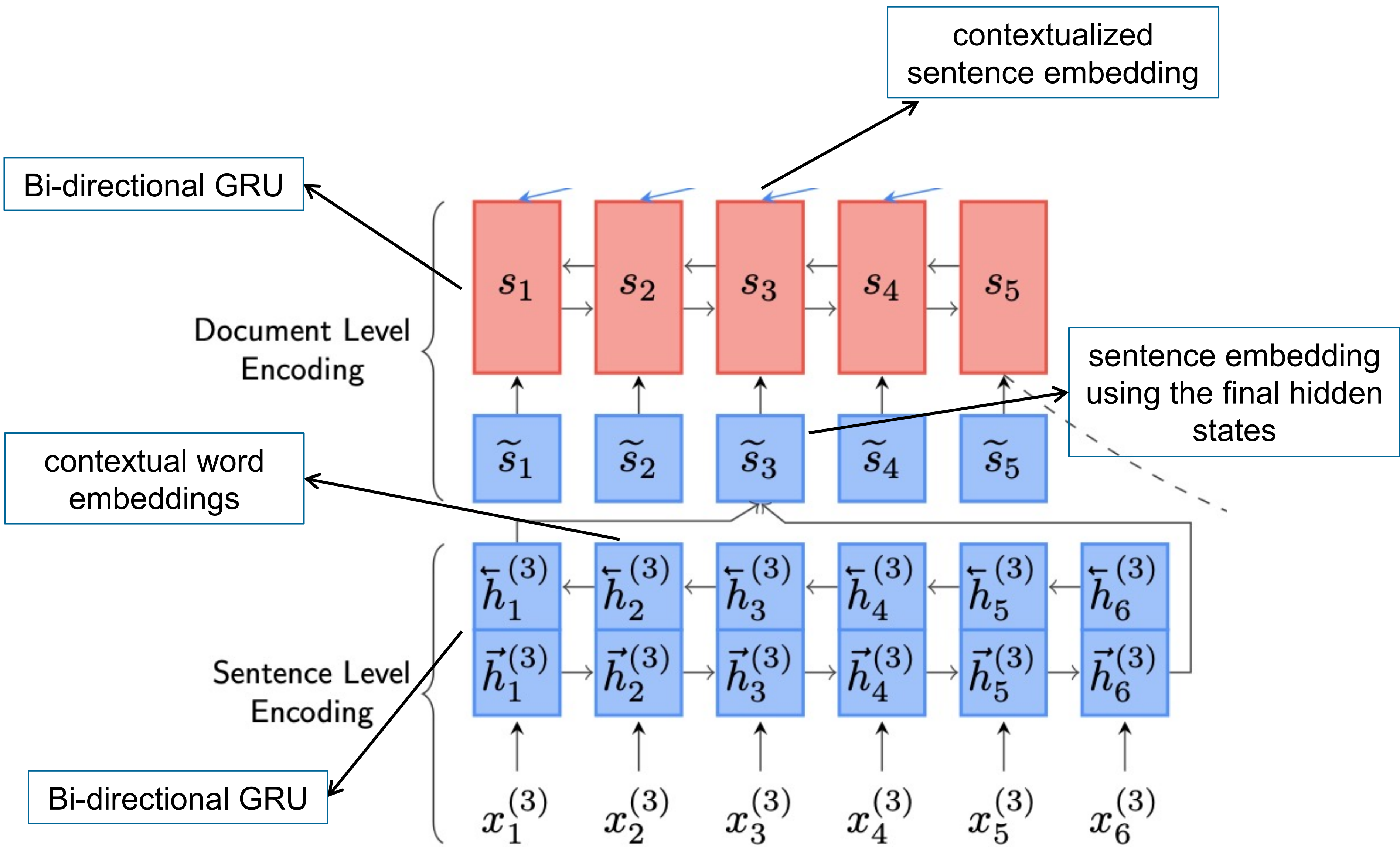
Core Ideas – NeuSum model

- At each time step, the model wants to decide which sentence to include in the summary (**sentence selection**)

- To do sentence selection, at each time step, the model assigns scores to sentences that are not included in summary (**sentence scoring**), and selects the one with the highest score

- The sentence scoring is based on the representation of each sentence, but also the content of the previously selected sentences
  - *Why on previously selected sentences?* Intuitively, if some contents are already included in the summary, the model should avoid selecting the sentences with similar contents

Sentence encoding



Sentence scoring and selection

- Sentence scoring learns a function δ that assigns a score to each sentence s_i at time step t. It uses:
  1. sentence embedding $\boldsymbol{s}_i$
  2. information of previously selected sentences, embedded in the vector $\boldsymbol{h}_t$ → **current state of summary**

$$\delta(s_i) = \text{function}(\boldsymbol{h}_t, \boldsymbol{s}_i)$$

$$\delta(s_i) = \boldsymbol{w}_s \tanh(\boldsymbol{h}_t \boldsymbol{W}_q + \boldsymbol{s}_i \boldsymbol{W}_d + \boldsymbol{b}_i)$$

- $\delta(s_i)$ is calculated for the sentences that are not yet included in the summary
- The sentence with highest δ is added to summary

Sentence scoring

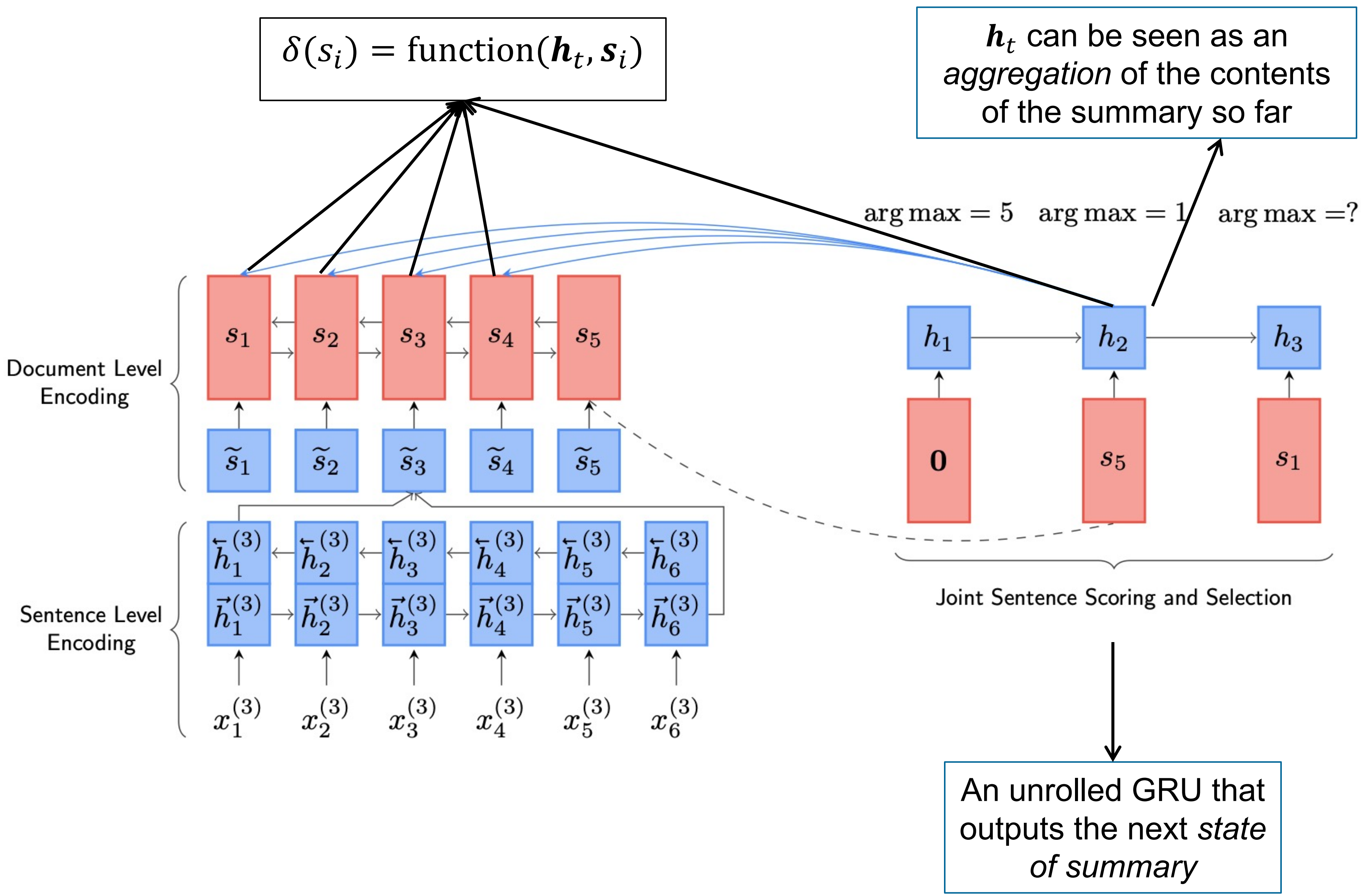
- How is current state of summary $\boldsymbol{h}_t$ calculated?
  - Using another GRU

- A GRU model outputs the new state of the summary using the previous state of summary and the last selected sentence

$$\boldsymbol{h}_t = \mathrm{GRU}(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1})$$

<u>You can find such more details by looking into the paper:</u>

- $\boldsymbol{h}_0$ is created based on the document embedding
- The model is optimized using the Kullback-Leibler divergence between the distribution of output scores and the distribution of ROUGE-2 F1 gains of sentences


$\delta(s_i) = \text{function}(\boldsymbol{h}_t, \boldsymbol{s}_i)$
$\boldsymbol{h}_t$ can be seen as an *aggregation* of the contents of the summary so far
$\arg\max = 5$
$\arg\max = 1$
$\arg\max = ?$
Document Level Encoding
Sentence Level Encoding
Joint Sentence Scoring and Selection
An unrolled GRU that outputs the next *state of summary*