Summer semester 2023

344.063 KV Special Topic:

Natural Language Processing with Deep Learning

RNNs: Language and Sequence to Sequence Models

Navid Rekab-saz

navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- Language modeling with RNN
- seq2seq for abstractive summarization
- Text decoding

Agenda

- **Language modeling with RNN**
- seq2seq for abstractive summarization
- Text decoding

Language Modeling – recap

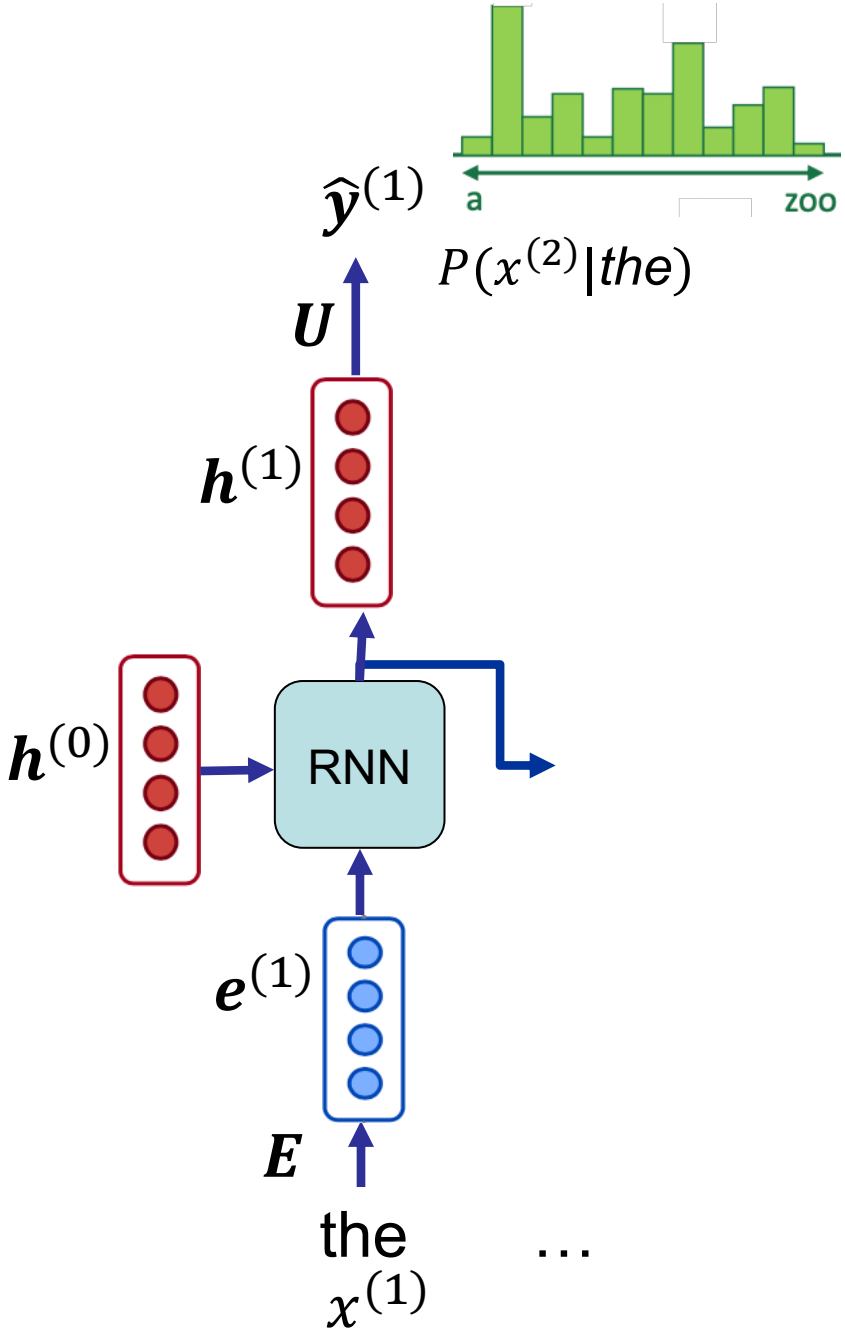
- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, a language model calculates the probability distribution of next word $x^{(t+1)}$ over all words in vocabulary

$$P(x^{(t+1)}|x^{(t)}, x^{(t-1)}, \ldots, x^{(1)})$$

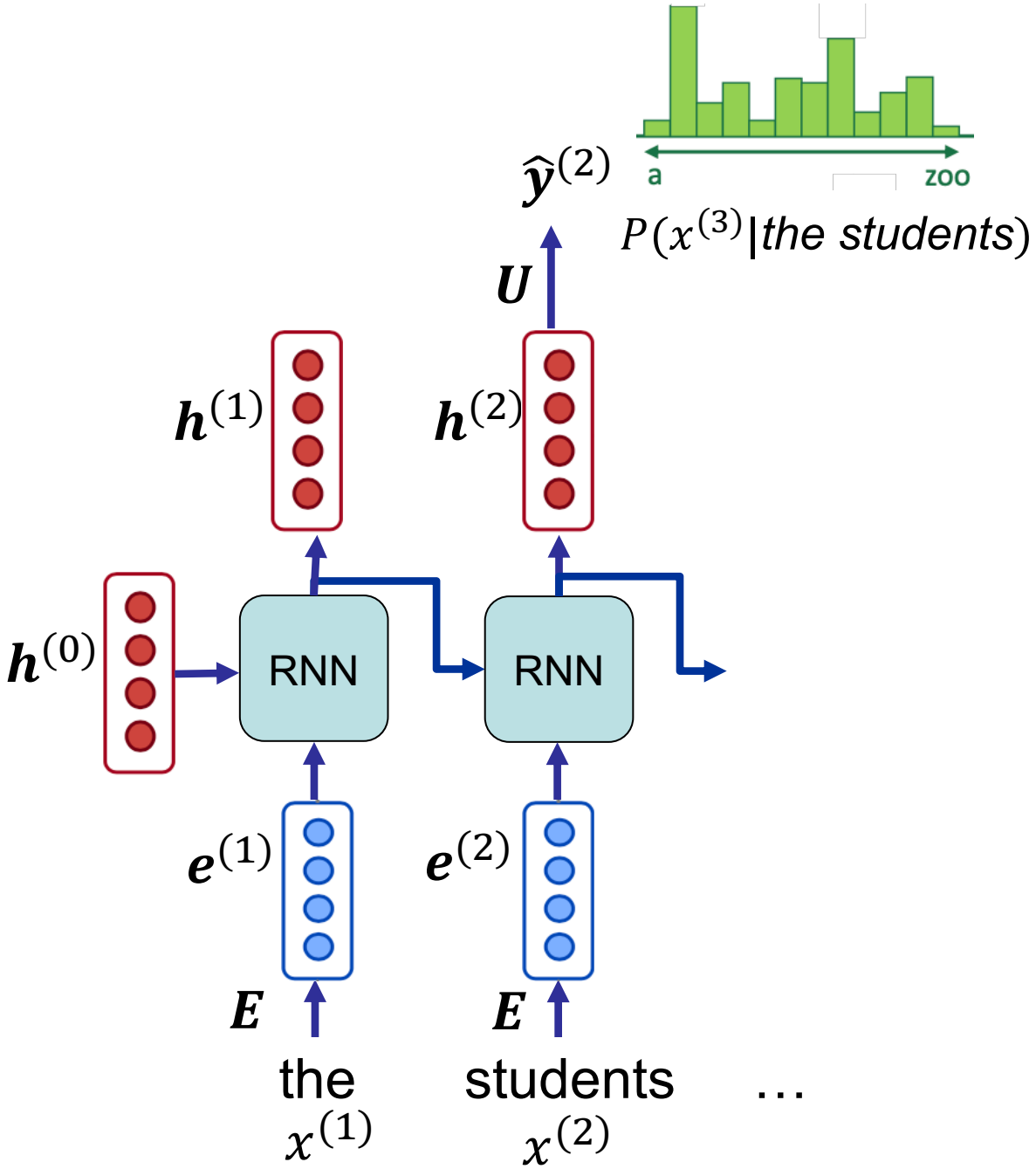
x is any word in the vocabulary $\mathbb{V} = \{v1, v2, \ldots, vN\}$

$P(v|\textit{the students opened their})$

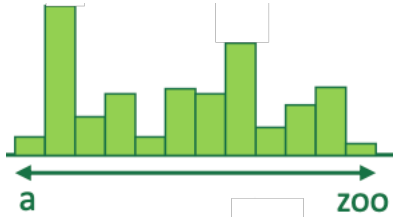
RNN Language Model



RNN Language Model

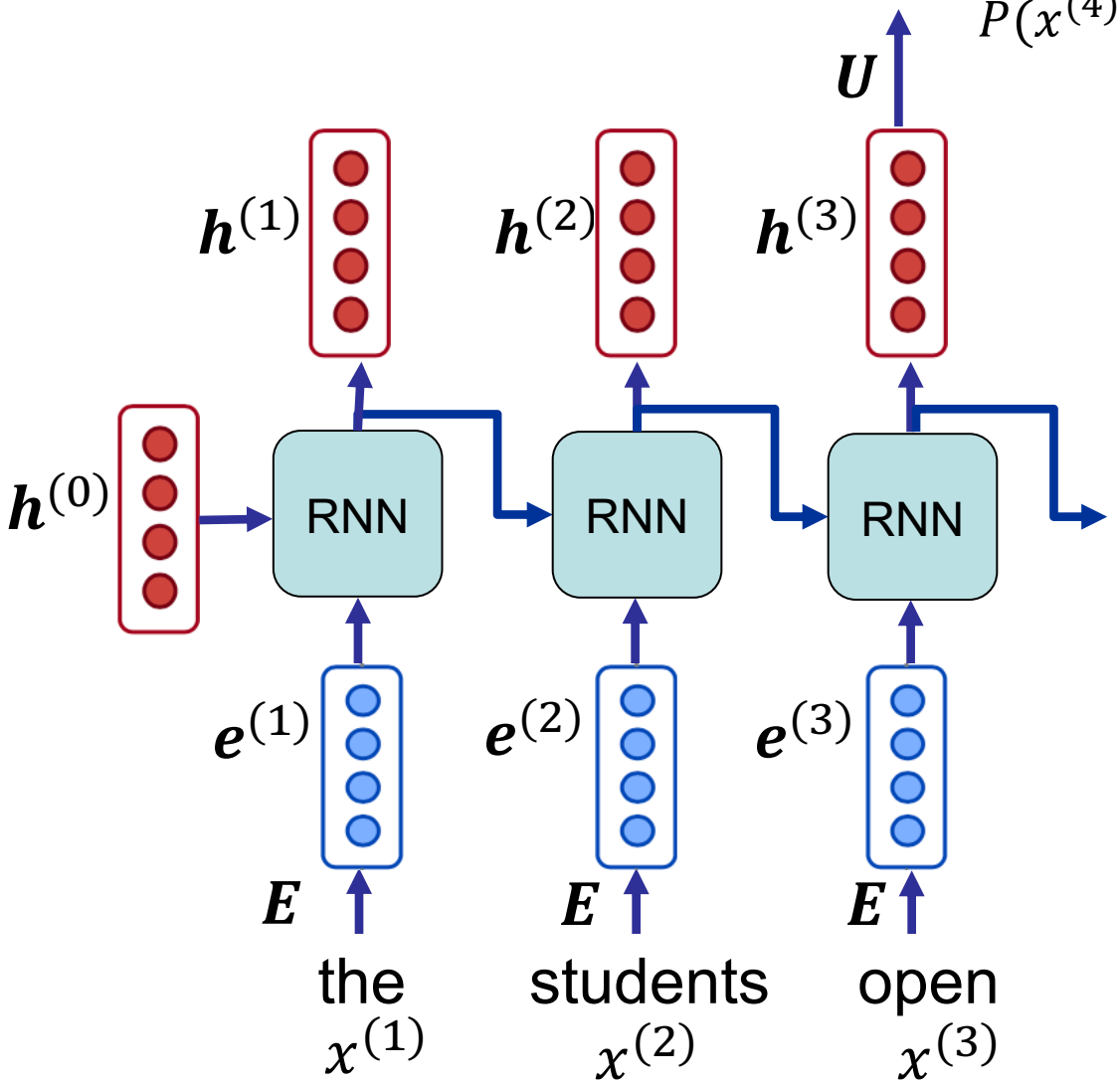


RNN Language Model

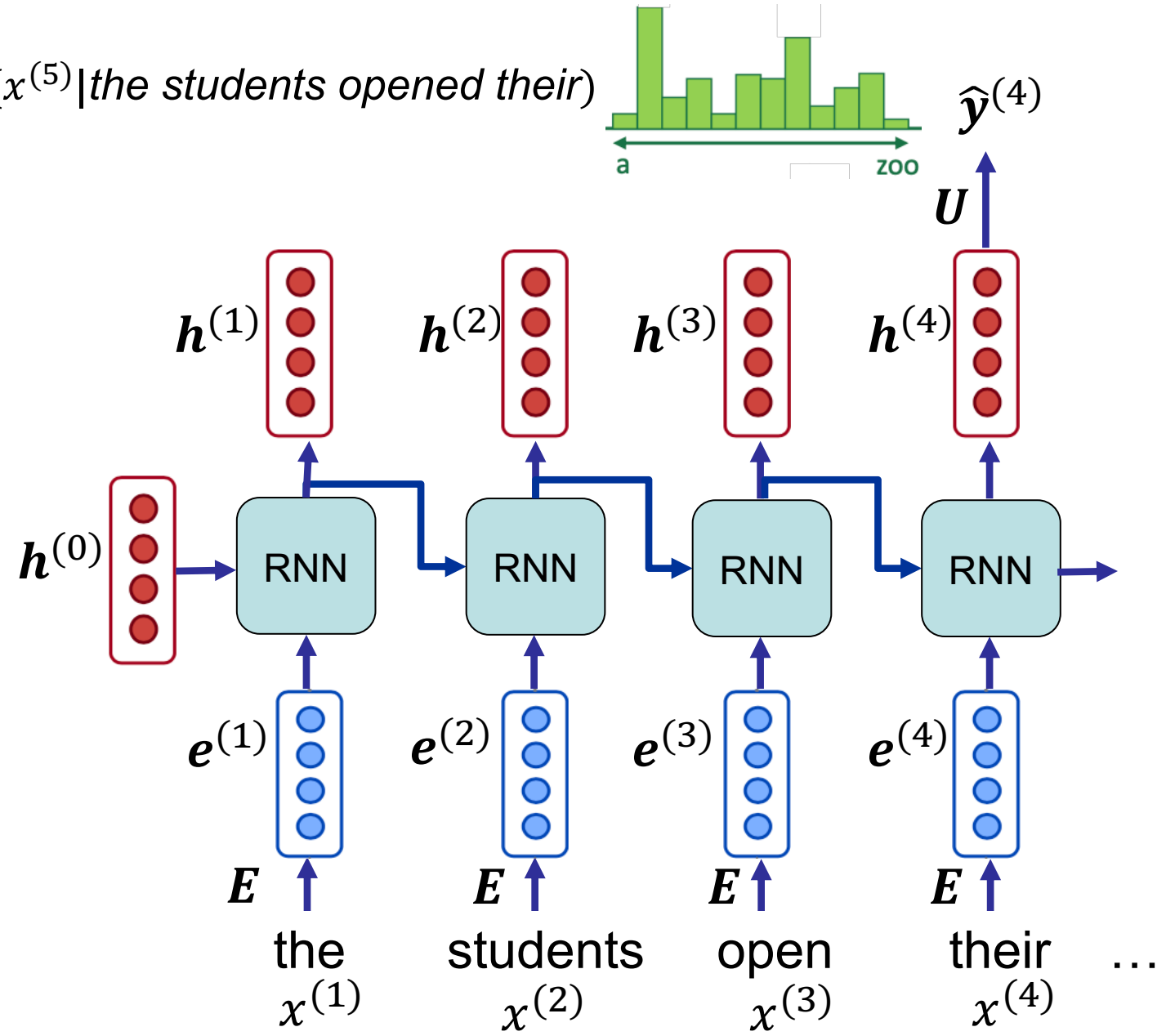


$\hat{y}^{(3)}$

$P(x^{(4)}|\textit{the students opened})$



RNN Language Model



Formulation

Encoder

- From words to word embeddings:
  - One-hot vector of word $x^{(t)} \rightarrow \boldsymbol{x}^{(t)} \in \mathbb{R}^N$
  - Fetching word embedding $\rightarrow \boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{E}$
    - In practice, $\boldsymbol{e}^{(t)}$ is achieved by fetching the vector of $x^{(t)}$ from $\boldsymbol{E}$ (no need for $\boldsymbol{x}^{(t)}$)
- RNN: $\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$

Decoder

- Predicted probabilities
  - Predicted probability distribution:

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}\right) \in \mathbb{R}^N$$

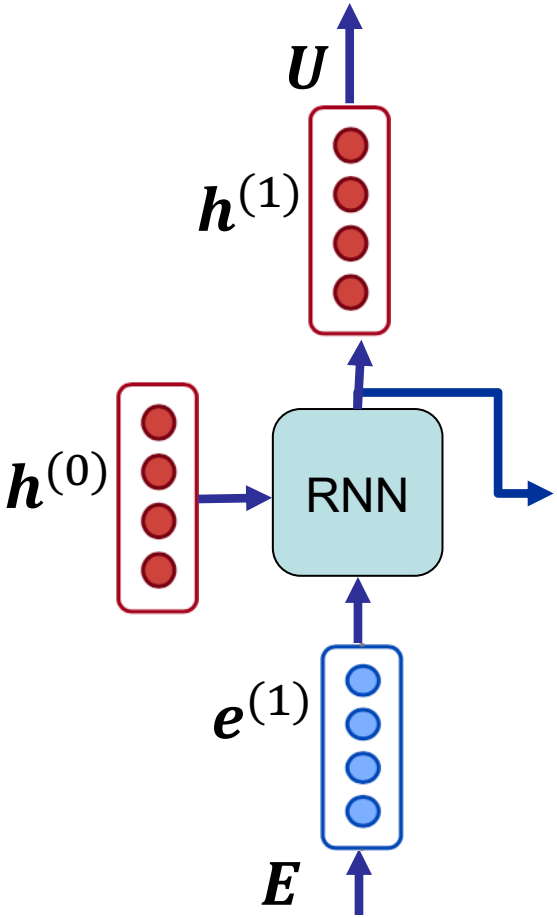
  - Probability of any word v at step t:

$$P\left(v \middle| x^{(t-1)}, \dots, x^{(1)}\right) = \hat{y}_v^{(t)}$$

Training

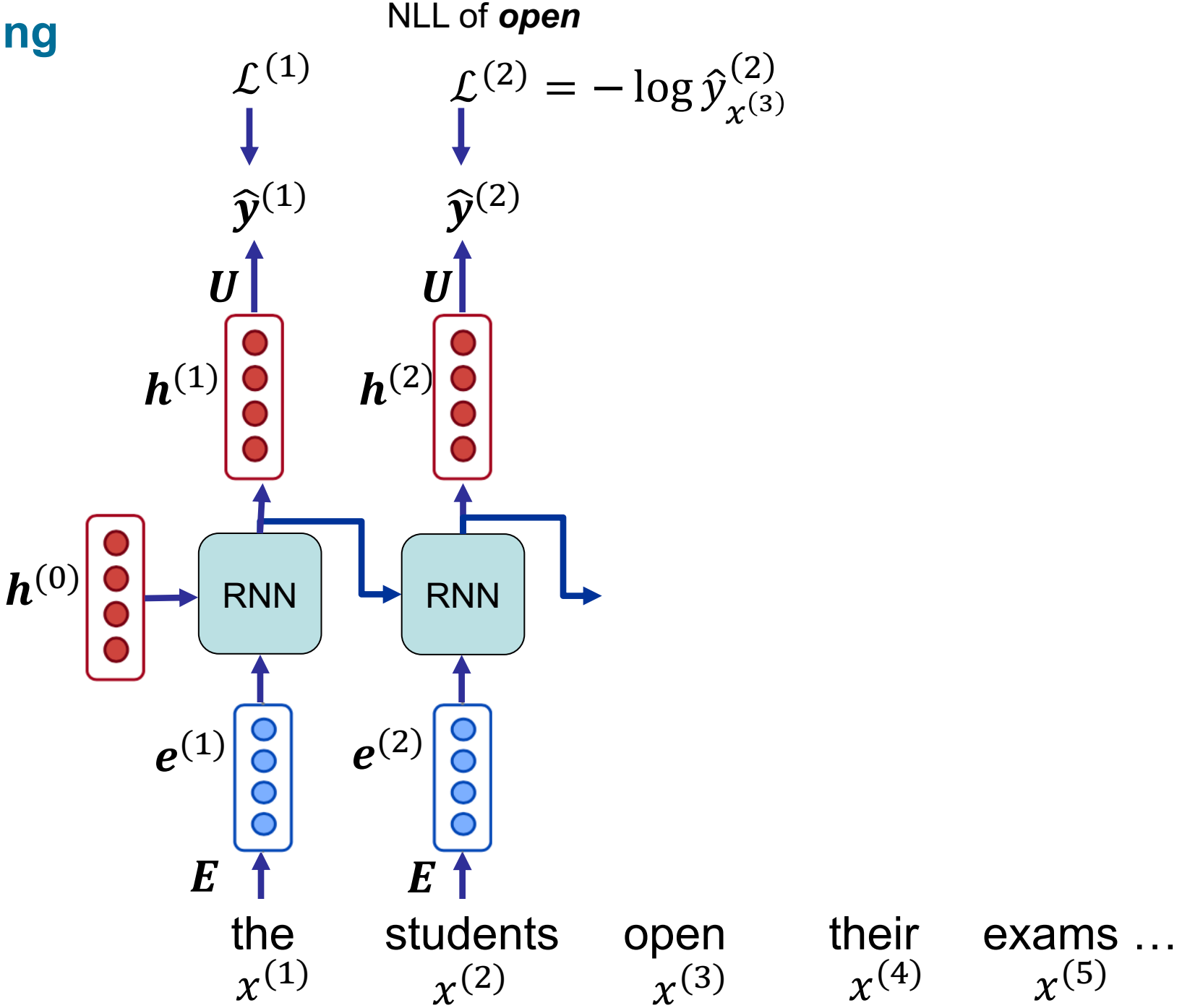
NLL of ***students***

$$\mathcal{L}^{(1)} = -\log \hat{y}^{(1)}_{x^{(2)}}$$

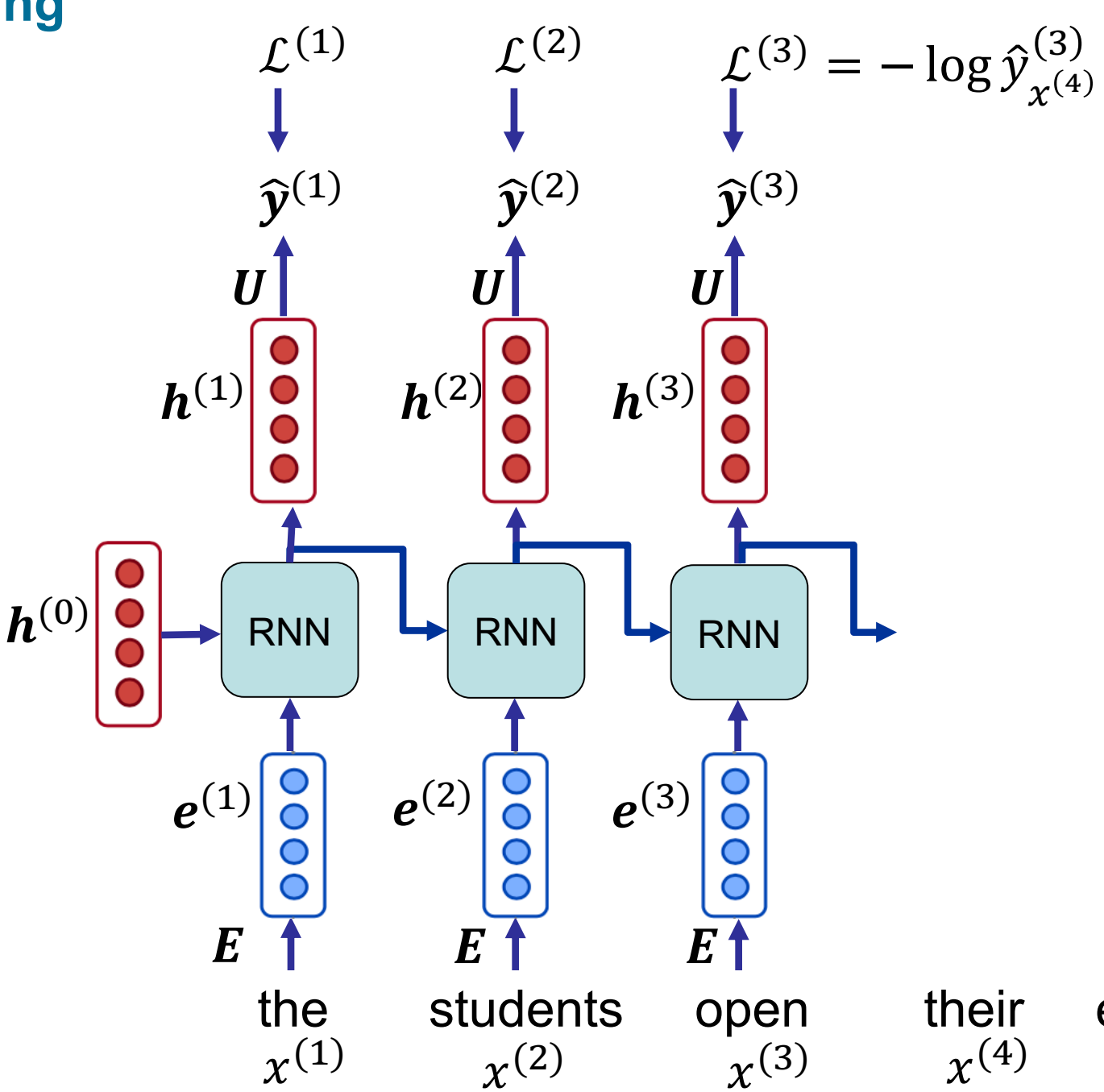


the $x^{(1)}$ students $x^{(2)}$ open $x^{(3)}$ their $x^{(4)}$ exams $x^{(5)}$ …

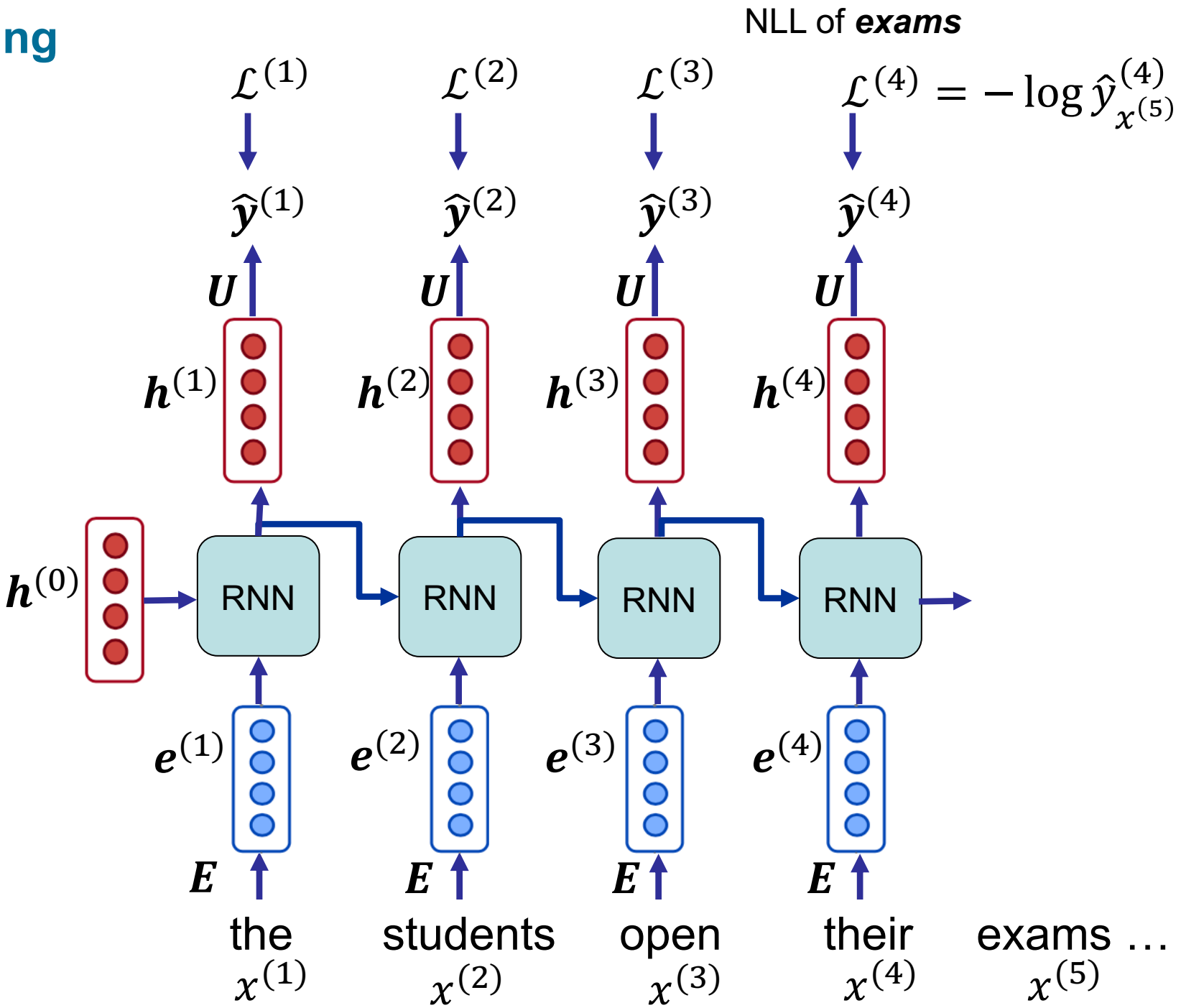
Training


NLL of *open*
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)} = -\log \hat{y}^{(2)}_{x^{(3)}}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(0)}$
RNN
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{e}^{(2)}$
$\boldsymbol{E}$
$\boldsymbol{E}$
the
$x^{(1)}$
students
$x^{(2)}$
open
$x^{(3)}$
their
$x^{(4)}$
exams …
$x^{(5)}$

Training


NLL of *their*
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)} = -\log \hat{y}^{(3)}_{x^{(4)}}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(0)}$
RNN
RNN
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{e}^{(2)}$
$\boldsymbol{e}^{(3)}$
$\boldsymbol{E}$
$\boldsymbol{E}$
$\boldsymbol{E}$
the
$x^{(1)}$
students
$x^{(2)}$
open
$x^{(3)}$
their
$x^{(4)}$
exams …
$x^{(5)}$

Training


NLL of *exams*
$\mathcal{L}^{(4)} = -\log \hat{y}^{(4)}_{x^{(5)}}$
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\hat{\boldsymbol{y}}^{(4)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{e}^{(2)}$
$\boldsymbol{e}^{(3)}$
$\boldsymbol{e}^{(4)}$
$\boldsymbol{E}$
the
$x^{(1)}$
students
$x^{(2)}$
open
$x^{(3)}$
their
$x^{(4)}$
exams …
$x^{(5)}$

Training an RNN Language Model

- Start with a large text corpus: $x^{(1)}, \ldots, x^{(T)}$
- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$
- Calculate the loss function: Negative Log Likelihood of the predicted probability of the true next word $x^{(t+1)}$

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

- Overall loss is the average of loss values over the entire training set:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{(t)}$$

Training RNN Language Model – Mini batches

- In practice, the overall loss is calculated not over whole the corpus, but over (mini) batches of length L :

$$\mathcal{L} = \frac{1}{L}\sum_{t=1}^{L} \mathcal{L}^{(t)}$$

- After calculating the $\mathcal{L}$ for one batch, gradients are computed, and weights are updated (e.g. using SGD)

Training RNN Language Model – Data preparation

- In practice, every forward pass contains k batches. These batches are all trained in parallel
  - with batch size k, tensor of each forward pass has the shape: $[k, L]$

- To prepare the data in this form, the corpus is splitted into sub-corpora. Each sub-corpus contains the text for each row of forward tensors

- For example, for batch size $k = 2$, the corpus is splitted in middle, and the first forward-pass tensor looks like:

$$\text{batch size } k \rightarrow \begin{bmatrix} x^{(1)} & \dots & x^{(L)} \\ x^{(\tau+1)} & \dots & x^{(\tau+L)} \end{bmatrix}$$

$\tau = {}^{T}/_{2}$ is the overall length of the first sub-corpus

Training RNN Language Model – $h^{(0)}$

- At the beginning, the initial hidden state $\boldsymbol{h}^{(0)}$ is set to vectors of zeros (no memory)

- To carry the memory of previous forward passes, at each forward pass, the initial hidden states are initialized with the values of the last hidden states of the previous forward pass

Example

- First forward pass: $\boldsymbol{h}^{(0)}$ is set to zero values

$$\begin{bmatrix} x^{(1)} & \dots & x^{(L)} \\ x^{(\tau+1)} & \dots & x^{(\tau+L)} \end{bmatrix}$$

- Second forward pass: $\boldsymbol{h}^{(0)}$ is set to the $\boldsymbol{h}^{(L)}$ values in the first pass

$$\begin{bmatrix} x^{(L+1)} & \dots & x^{(2L)} \\ x^{(L+\tau+1)} & \dots & x^{(\tau+2L)} \end{bmatrix}$$

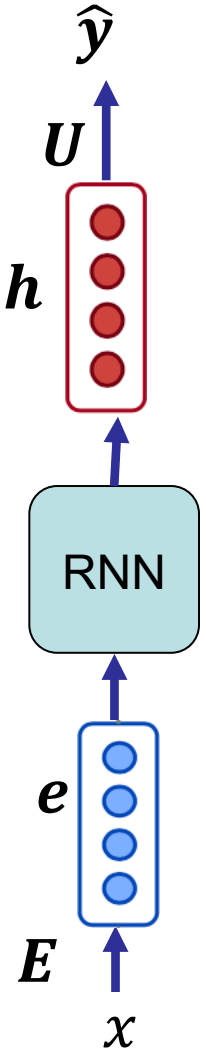
Training RNN Language Model – Parameters

- Parameters in a vanilla RNN (bias terms discarded)
  - $\boldsymbol{E} \rightarrow N \times d$
  - $\boldsymbol{U} \rightarrow h \times N$
  - $\boldsymbol{W}_i \rightarrow d \times h$
  - $\boldsymbol{W}_h \rightarrow h \times h$

d: dimension of input embedding

h: dimension of hidden vectors

- Encoder and decoder embeddings have most of the parameters



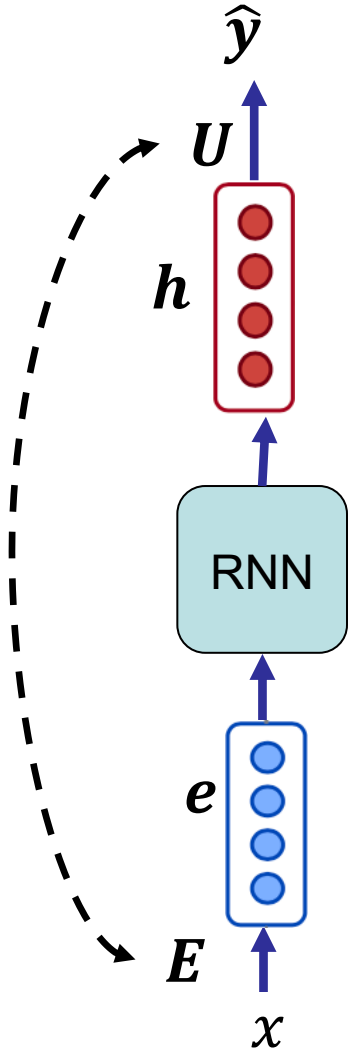
Training RNN Language Model – Weight Tying

- Parameters in a vanilla RNN (bias terms discarded)
  - $\boldsymbol{E} \rightarrow N \times d$
  - $\boldsymbol{U} \rightarrow h \times N$
  - $\boldsymbol{W}_i \rightarrow d \times h$
  - $\boldsymbol{W}_h \rightarrow h \times h$

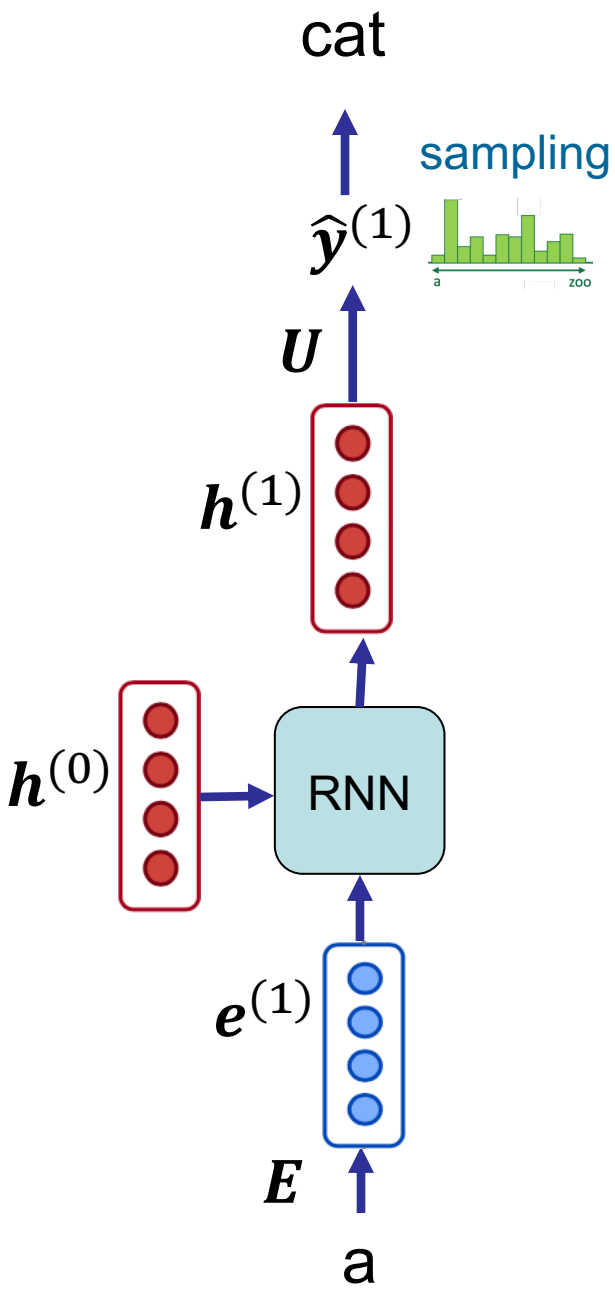
d: dimension of input embedding

h: dimension of hidden vectors

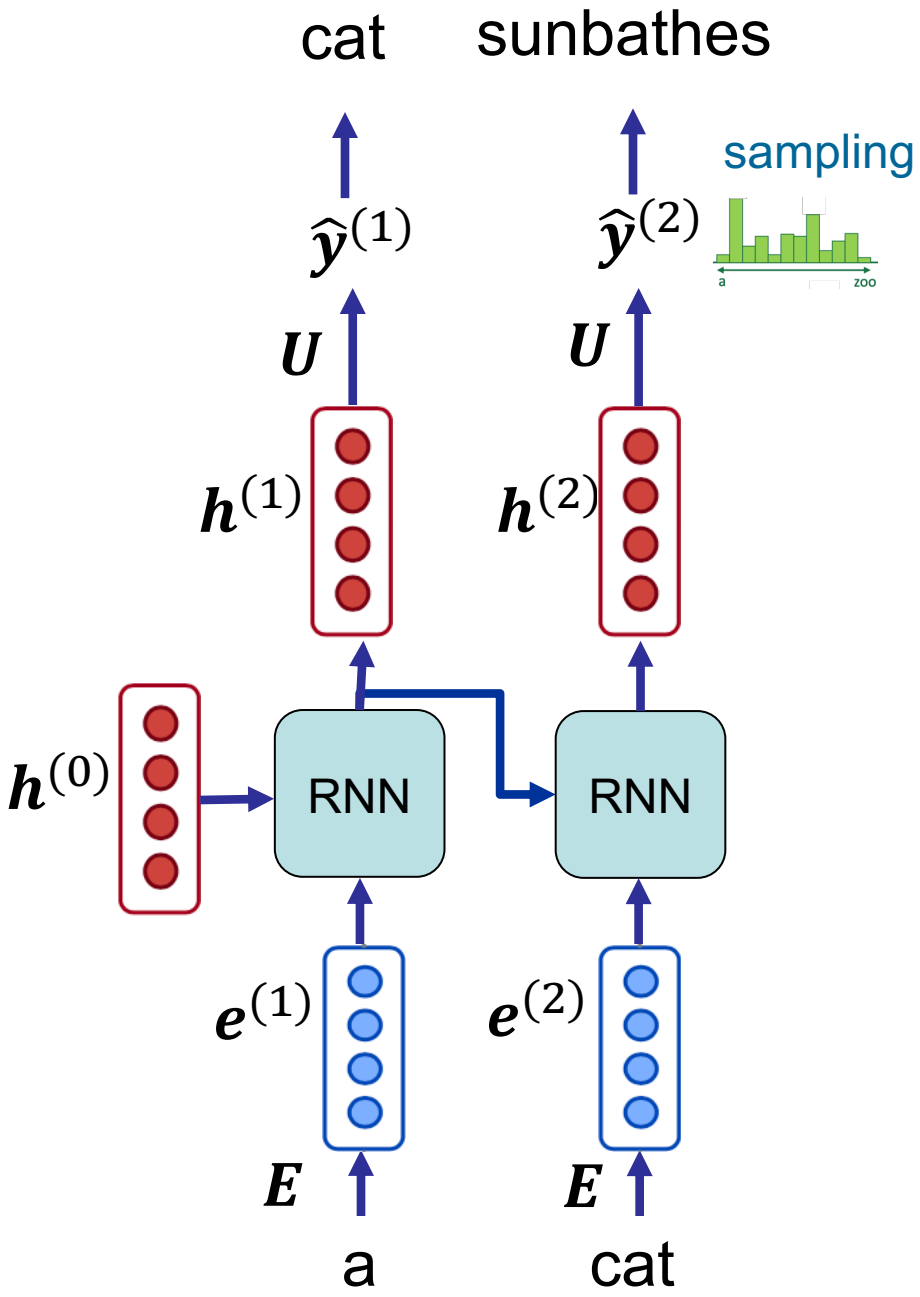
- Weight tying: set the decoder parameters the same as encoder parameters (saving $N \times h$ decoding parameters)
  - In this case d must be equal to h
  - If $d \neq h$, usually a linear projects the output vector from h to d dimensions



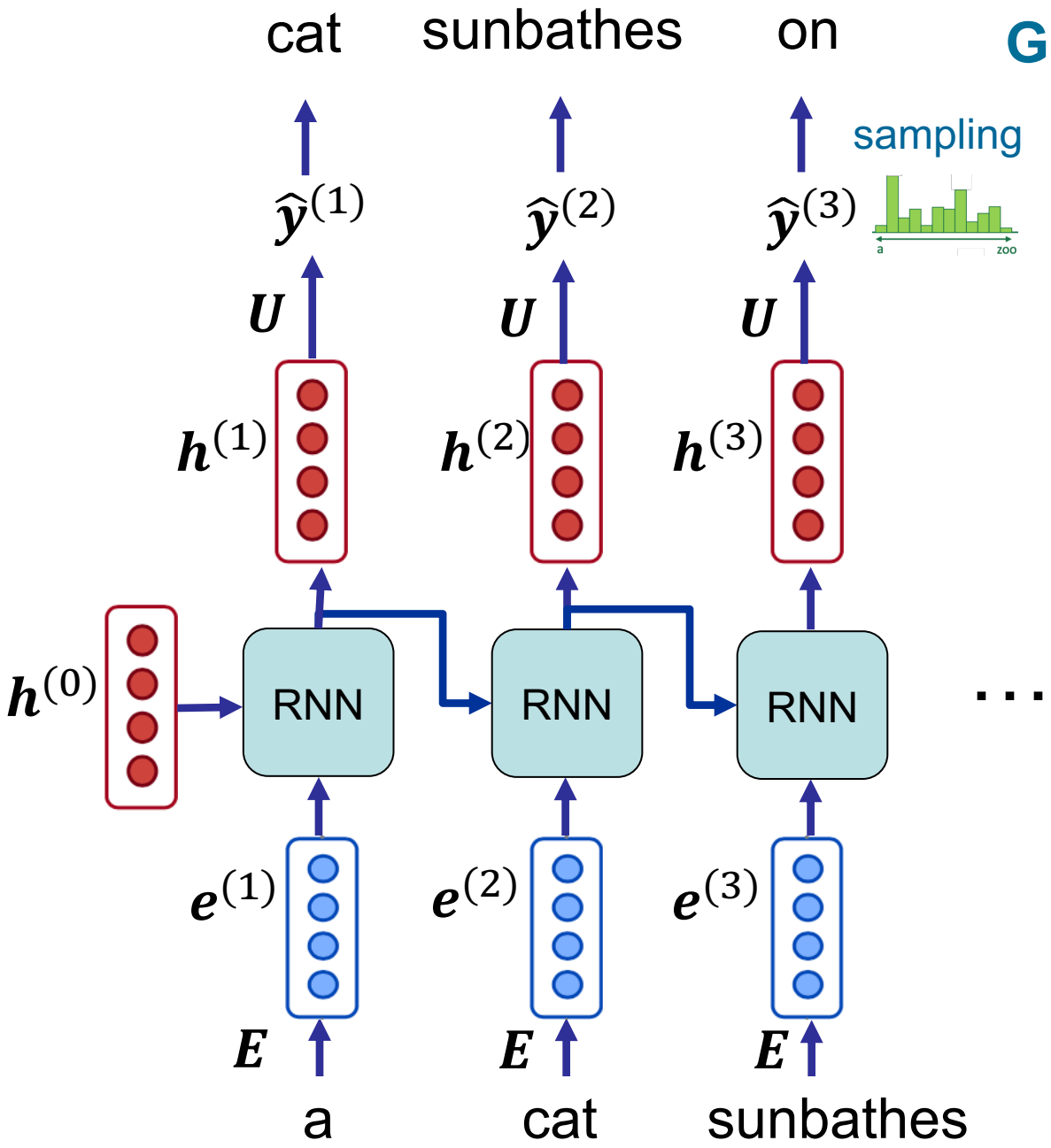
Generating text



Generating text



Generating text



Generating text with RNN Language Model

- Trained on Obama speeches

Jobs

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people.

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with RNN Language Model

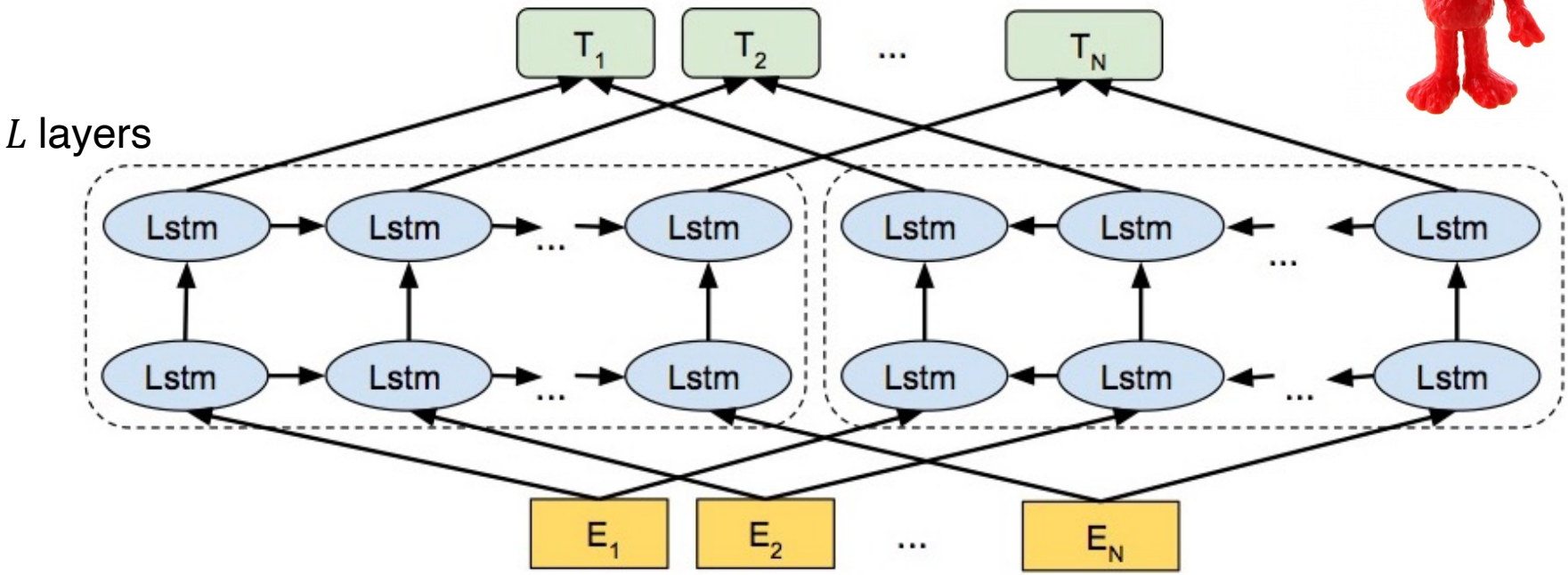
- Trained on Trump speeches

make the country
rich. it was terrible. but saudi arabia, they make a billion dollars a day. i was the king. i was the king. i was the smartest person in yemen, we have to get to business. i have to say, but he was an early starter. and we have to get to business. i have to say, donald, i can't believe it. it's so important. but this is what they're saying, dad, you're going to be really pro, growth, blah, blah. it's disgusting what's disgusting., and it was a 19 set washer and to go to japan. did you hear that character. we are going to have to think about it. but you know, i've been nice to me.

https://github.com/ppramesi/RoboTrumpDNN

ELMo (Embeddings from Language Models)

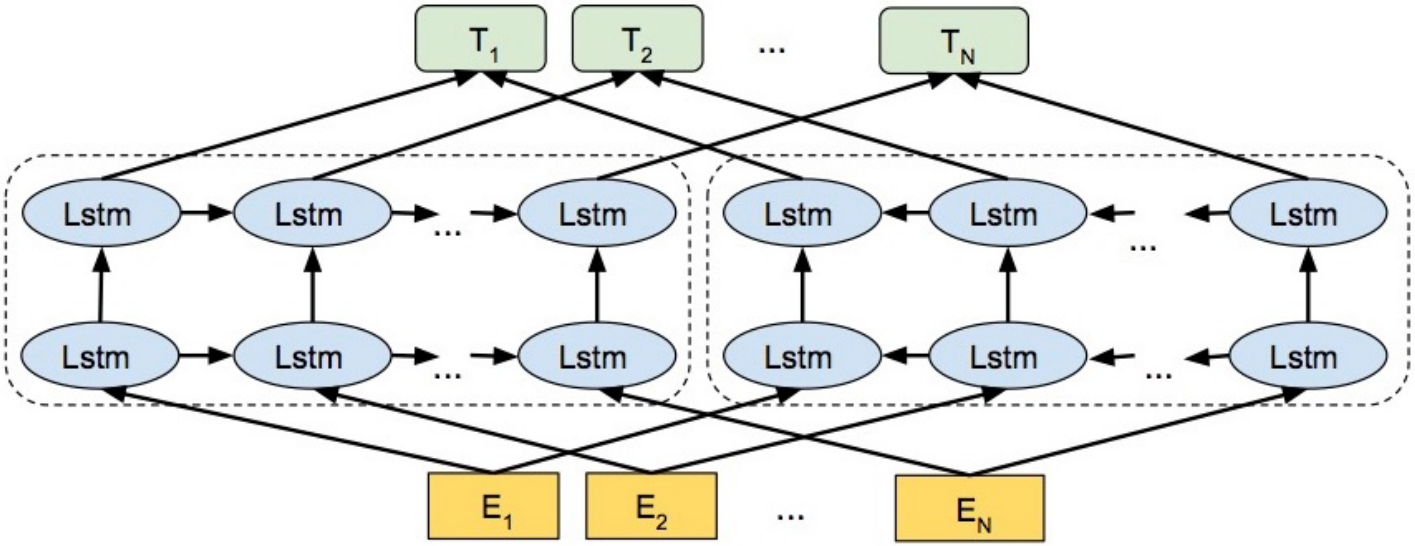
ELMo is a Multi-layer Bidirectional LSTM, trained on a Language Modeling objective



Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. **Deep Contextualized Word Representations**. In *Proc. of NAACL-HTL 2018*

ELMo – contextualized word embedding

- Given the set of input embeddings $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_N$, in each layer j, ELMo outputs a set of contextualized word embeddings $\boldsymbol{h}_1^j, \boldsymbol{h}_2^j, \ldots, \boldsymbol{h}_N^j$
- Contextualized embeddings in higher-levels capture semantic aspects (e.g., word senses), while embeddings in lower-levels model aspects of syntax (e.g., part-of-speech tagging)



ELMo in supervised tasks

- In supervised tasks, ELMo makes use of the embeddings in all layers (not only the last layer)!

- The final word embeddings are the weighted sum of the intermediary hidden states. Embedding of the word at position i:

$$\gamma \sum_{j=1}^{L} \theta_j \boldsymbol{h}_i^j$$

θ_j defines the weight (importance) of each layer

- θ_j values and γ are also model parameters, and are trained end-to-end with whole the model

Model parameters are shown in red

Summary

RNN for Language Modeling

- **Pros**:
  - RNN can process any length input
  - RNN can (in theory) use information from many steps back
  - Model size doesn't increase for longer input sequences

- **Cons**:
  - Recurrent computation is slow → (in its vanilla form) does not fully exploit the parallel computation capability of GPUs
  - Biased toward recent incidents → in practice, RNN has the tendency to use the information of recent steps (harder to access information from many steps back)

Agenda

- Language modeling with RNN
- **seq2seq for abstractive summarization**
- Text decoding

Text Summarization

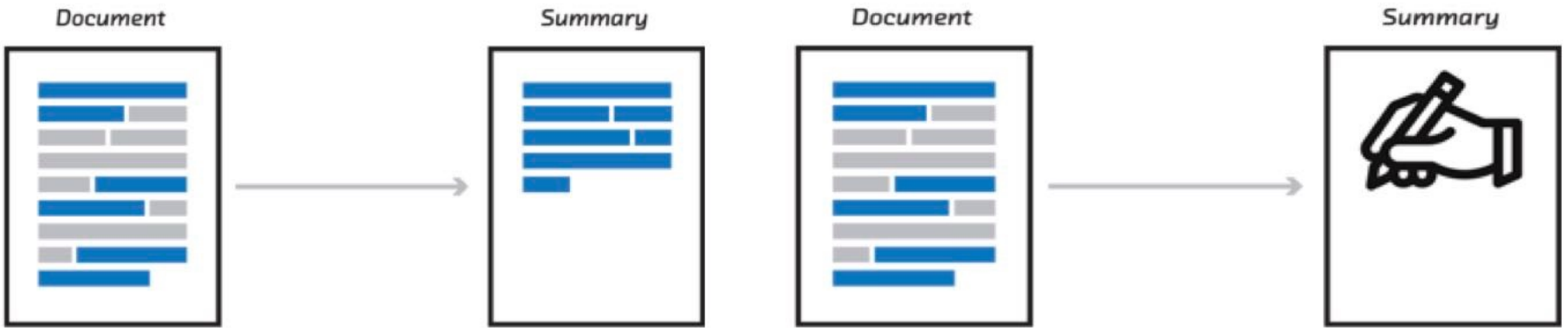
- The task of summarizing the key information content of a text (document) $X = \{x_1, x_2, \ldots, x_N\}$ into a summary $Y = \{y_1, y_2, \ldots, y_M\}$
  - Summary is concise and (much) shorter than document

- 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

CNN/DailyMail dataset 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***
  - …

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

Y and $\hat{Y}$ are the reference and output summary, respectively. n-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

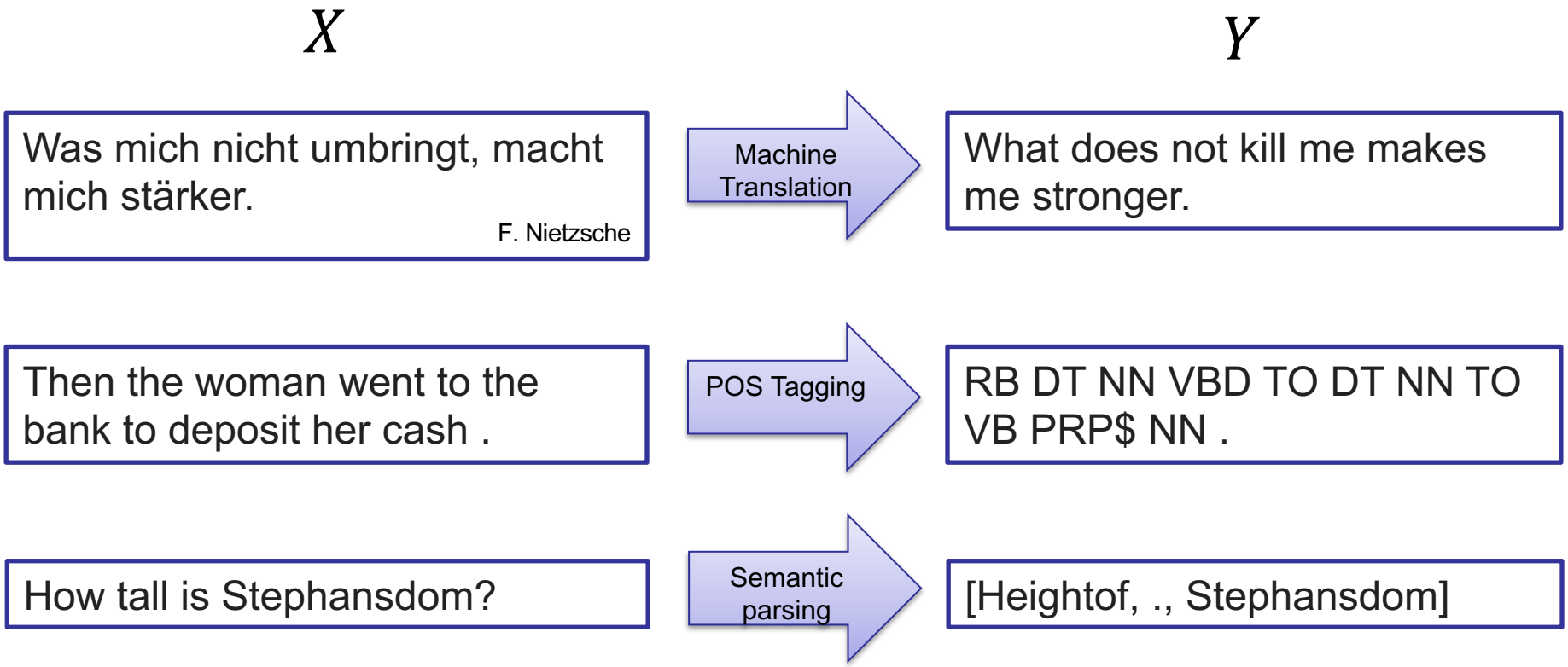
- Y: “police hugged the gunman”
- $\hat{Y}1$: “police hug the gunman”
- $\hat{Y}2$: “the gunman hug police”

- 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$(\hat{Y}2, Y) = 0.5$

Sequence in – sequence out!

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



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 $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(L)}\}$,
  - generate the target sequence $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(T)}\}$

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

… and for whole the target sequence:

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

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

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

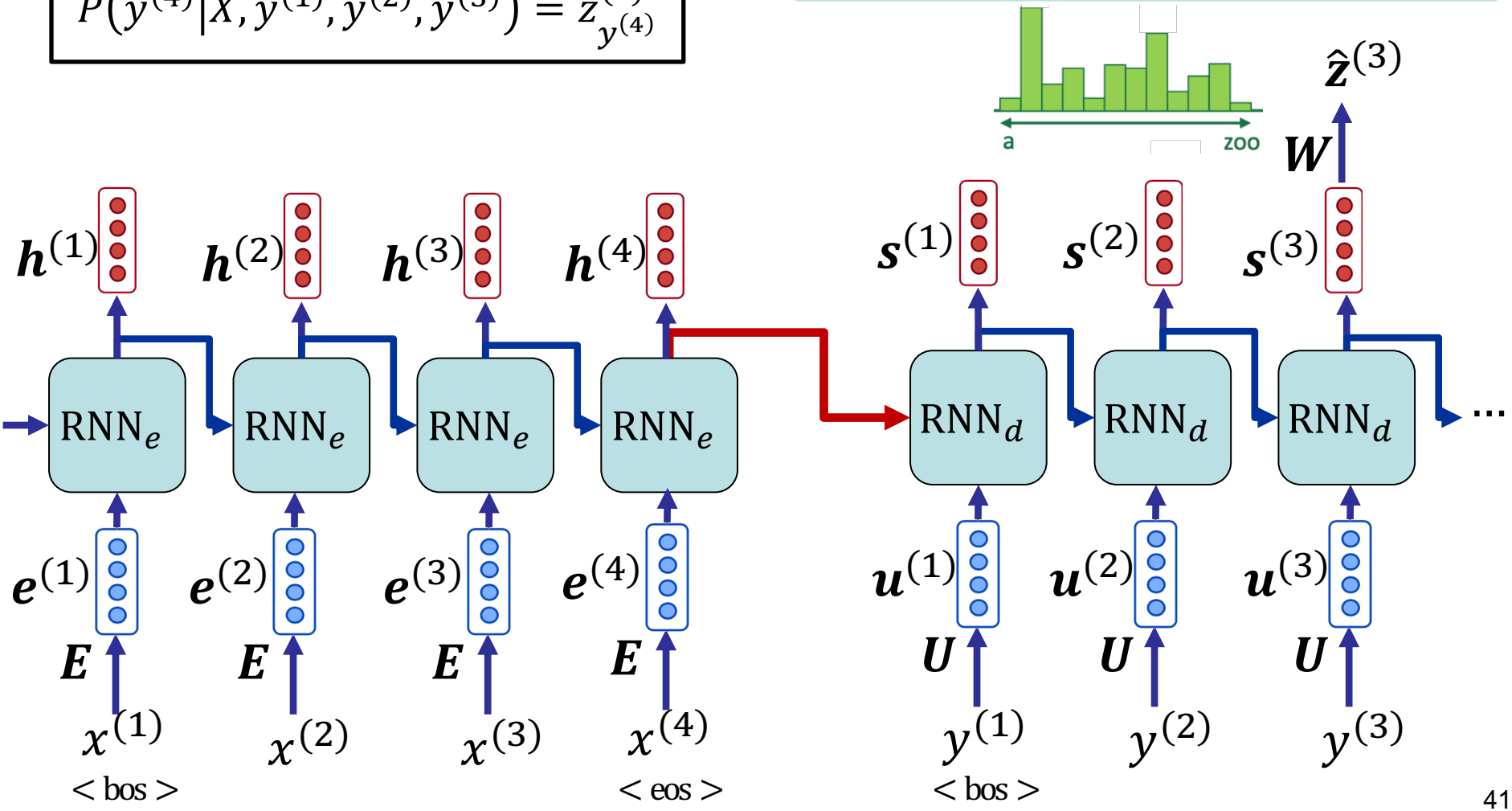
ENCODER

Probability of appearance of the next target word:

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

DECODER

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



Seq2seq with two RNNs – formulation

- 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

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

- Encoder RNN:

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

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

- From words to word embeddings:
  - Decoder embeddings of target words ($\mathbb{V}_d$) at input $\rightarrow$ $\boldsymbol{U}$
  - Embedding of the target word $y^{(t)}$ at time step $t \rightarrow \boldsymbol{u}^{(t)}$
- Decoder RNN: $\boldsymbol{s}^{(t)} = \mathrm{RNN}_d(\boldsymbol{s}^{(t-1)}, \boldsymbol{u}^{(t)})$
  - where the initial hidden state of the decoder RNN is set to the last hidden state of the encoder RNN: $\boldsymbol{s}^{(0)} = \boldsymbol{h}^{(L)}$
- Decoder output prediction
  - Predicted probability distribution of words at the next time step:

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

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

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

Parameters are shown in red

Training Seq2seq

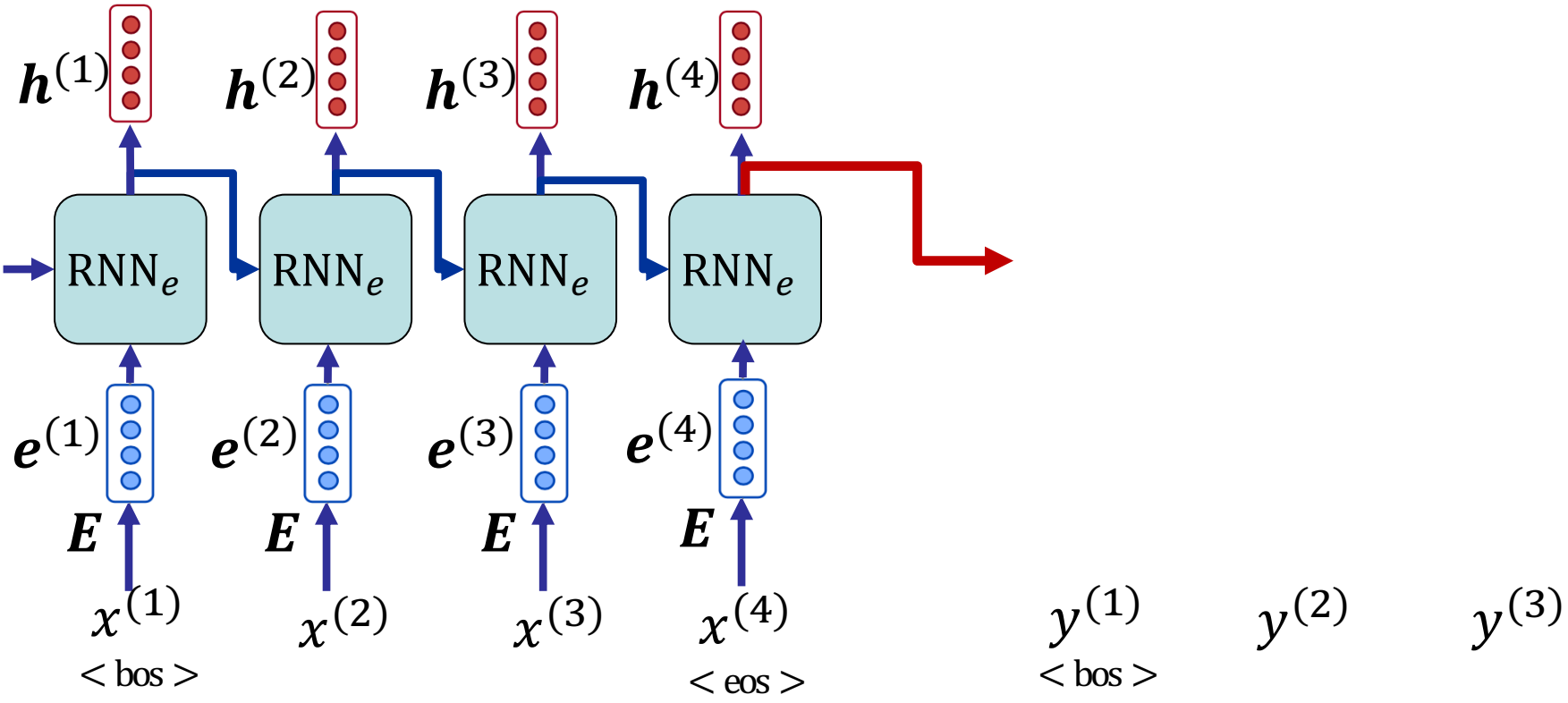
- Training a seq2seq is the same as a Language Model
  - Predict next word, calculate loss, backpropagate, and update parameters
  - Since seq2seq is an end-to-end model, gradient flows from the loss to all parameters, namely both RNNs and all embeddings
- Loss function: Negative Log Likelihood of the predicted probability of the correct next target word $y^{(t+1)}$

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

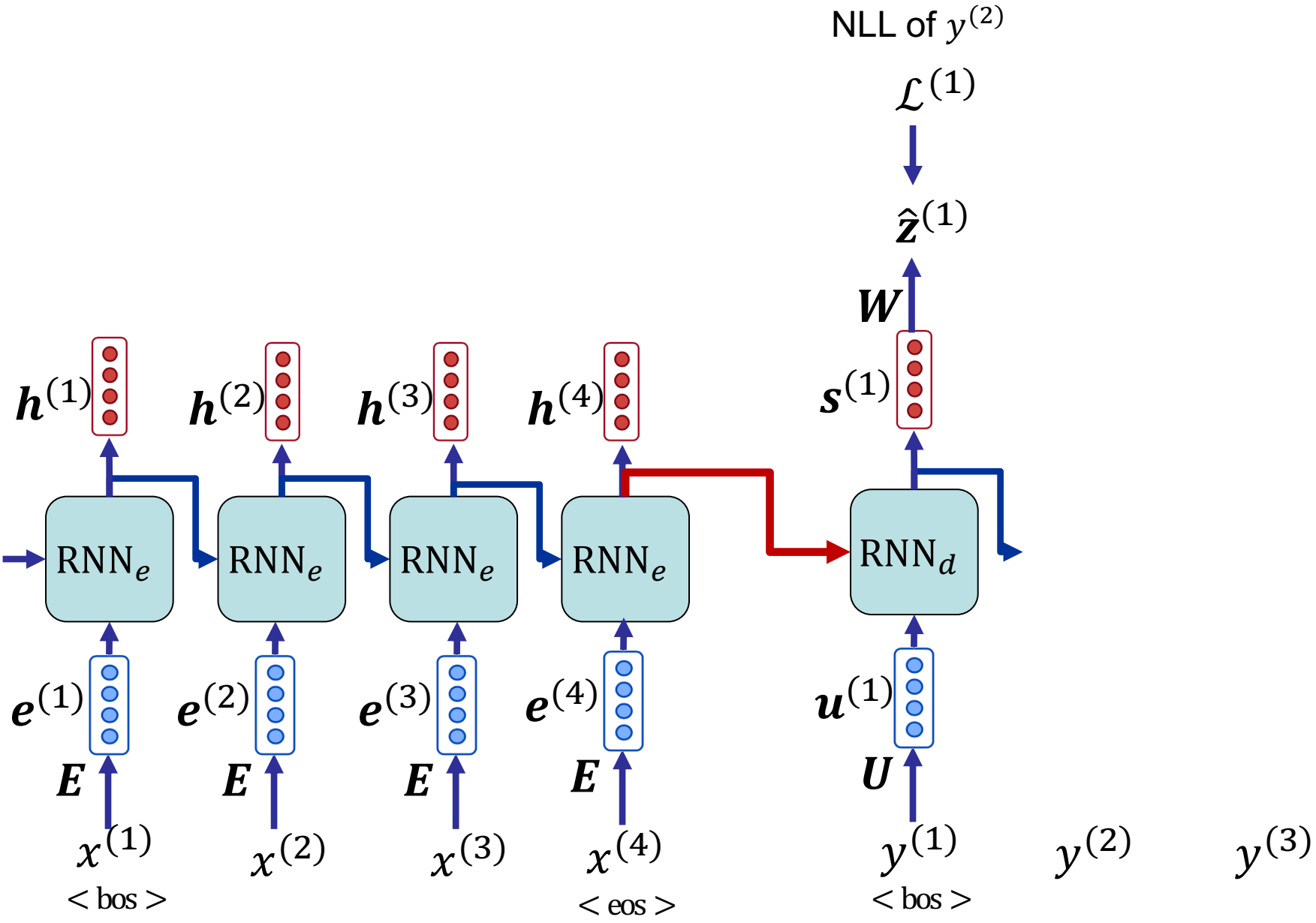
- Overall loss is the average of loss values over the training data in a batch:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{(t)}$$

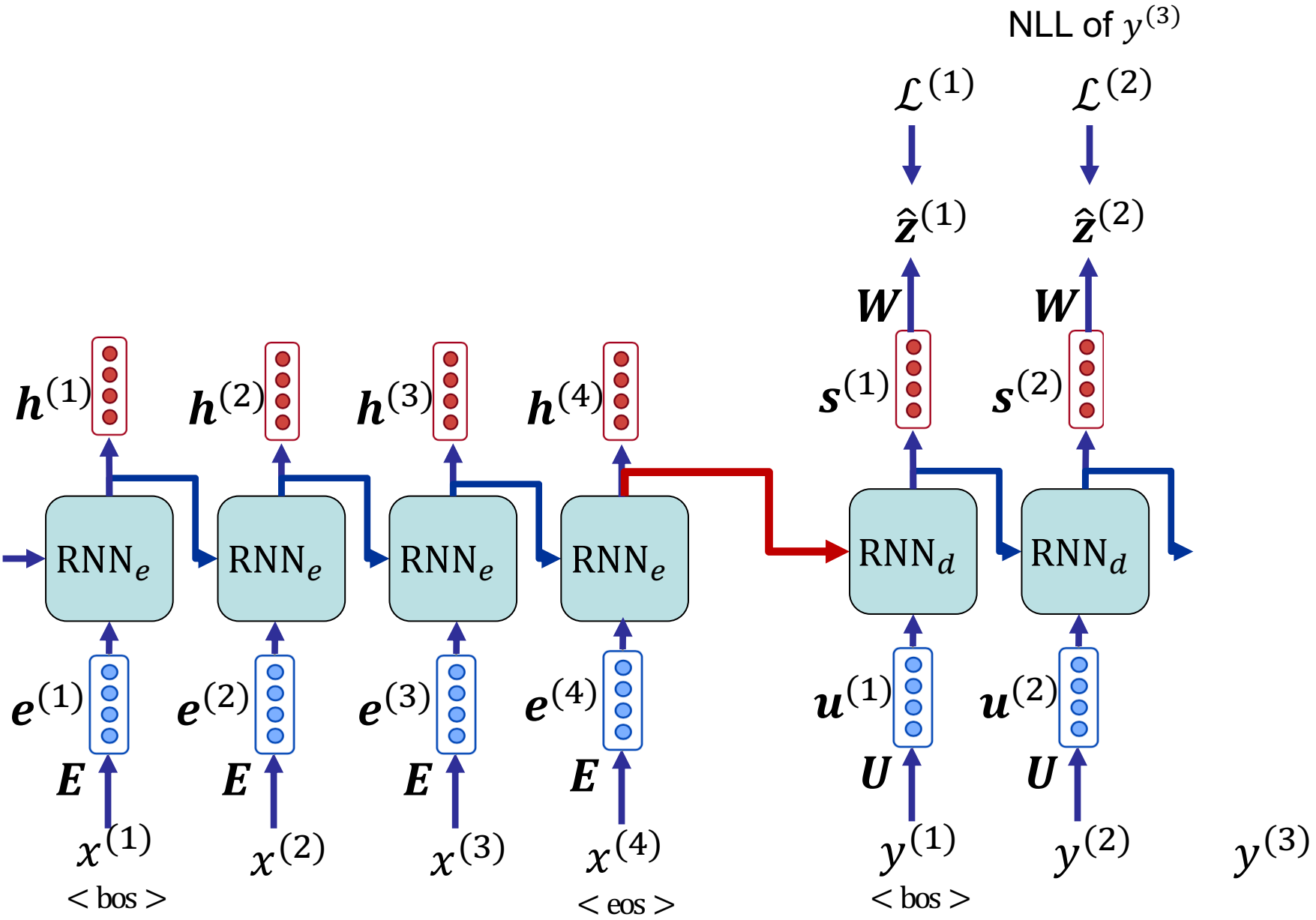
Training Seq2seq



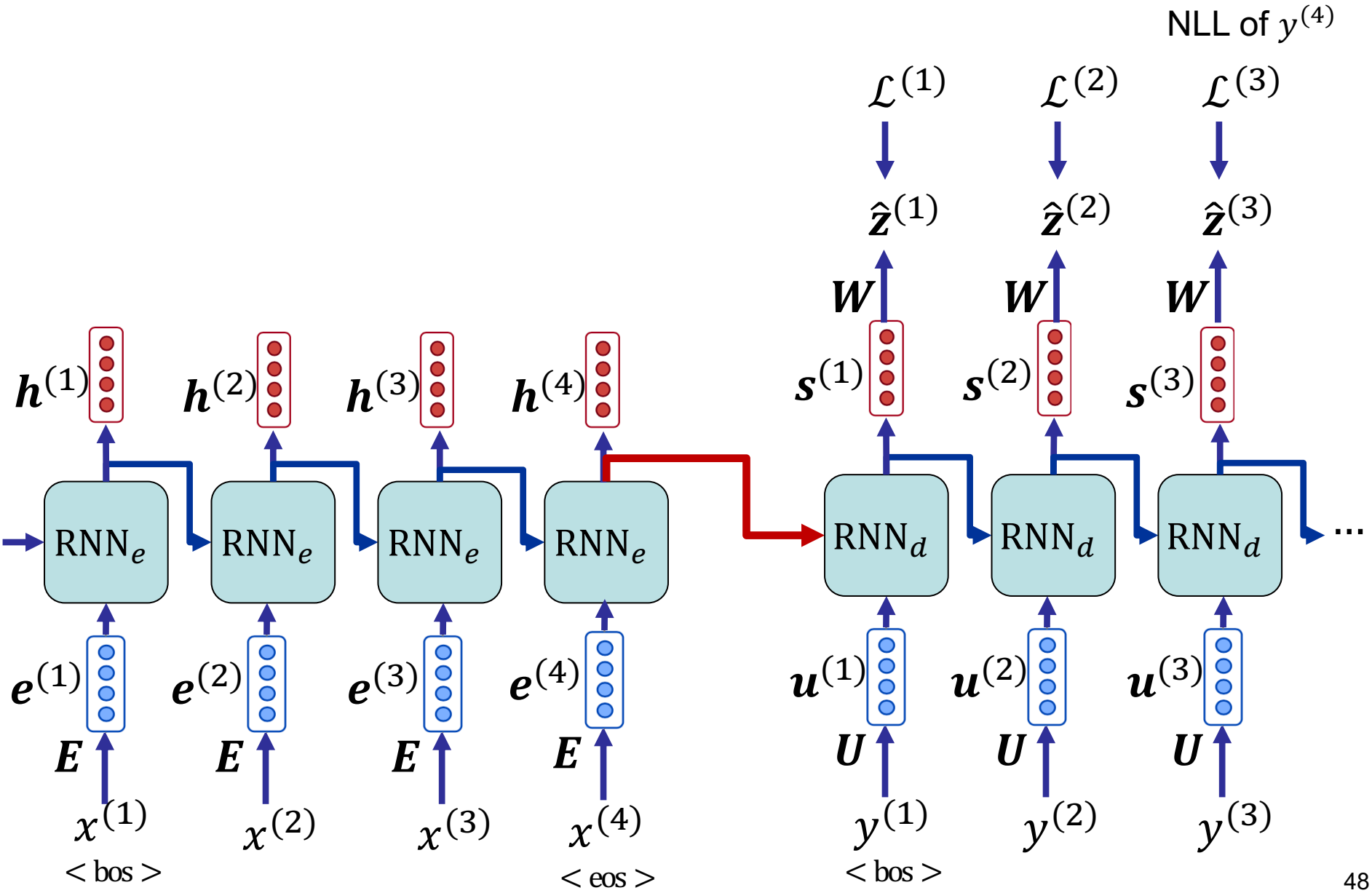
Training Seq2seq



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

Agenda

- Language modeling with RNN
- seq2seq for abstractive summarization
- **Text decoding**

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 $P(Y|X)$:

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

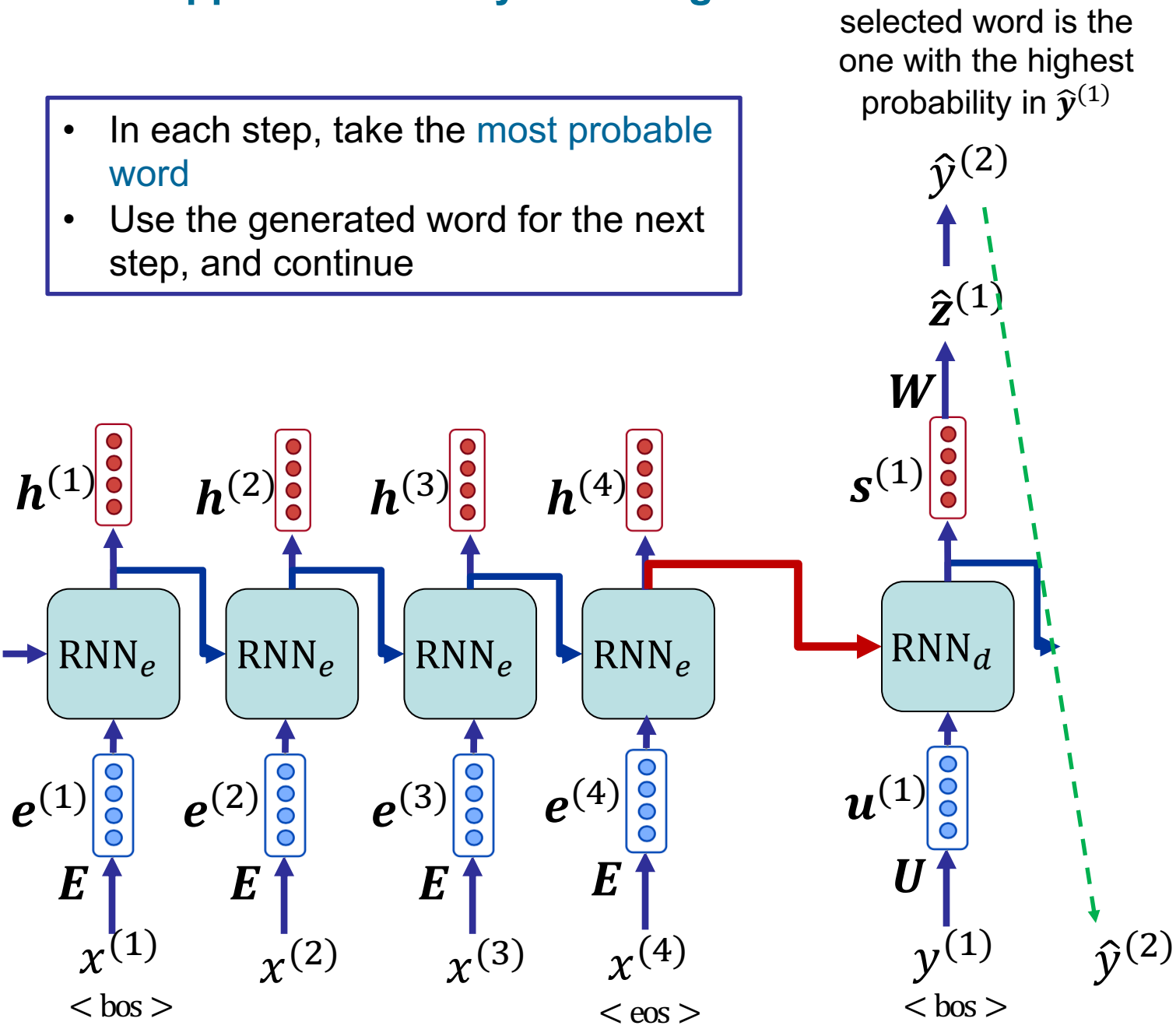
where $P(Y|X)$ for any arbitrary $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(T)}\}$ is:

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

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

A first approach: Greedy decoding

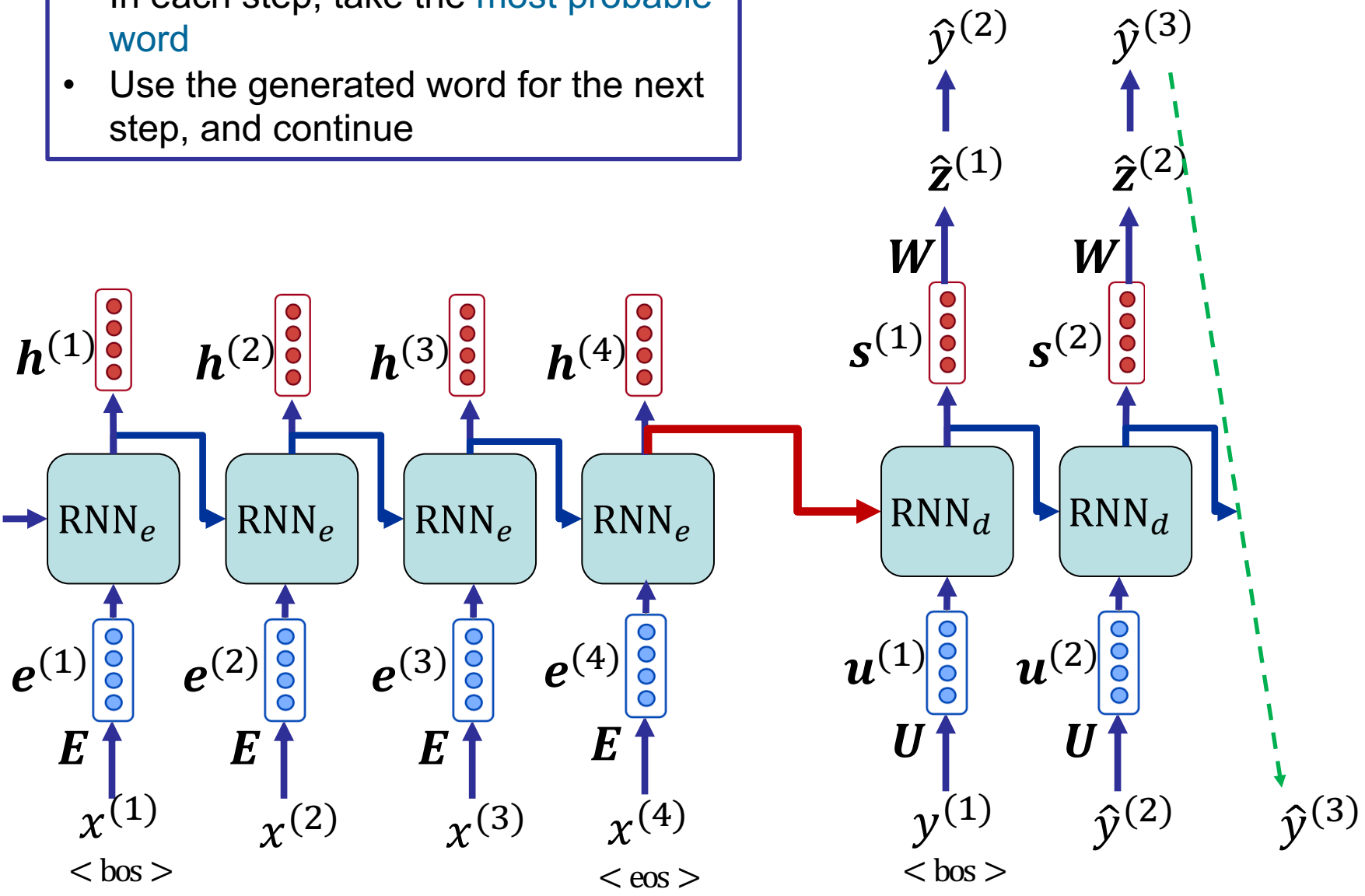
- In each step, take the most probable word
- Use the generated word for the next step, and continue



A first approach: Greedy decoding

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

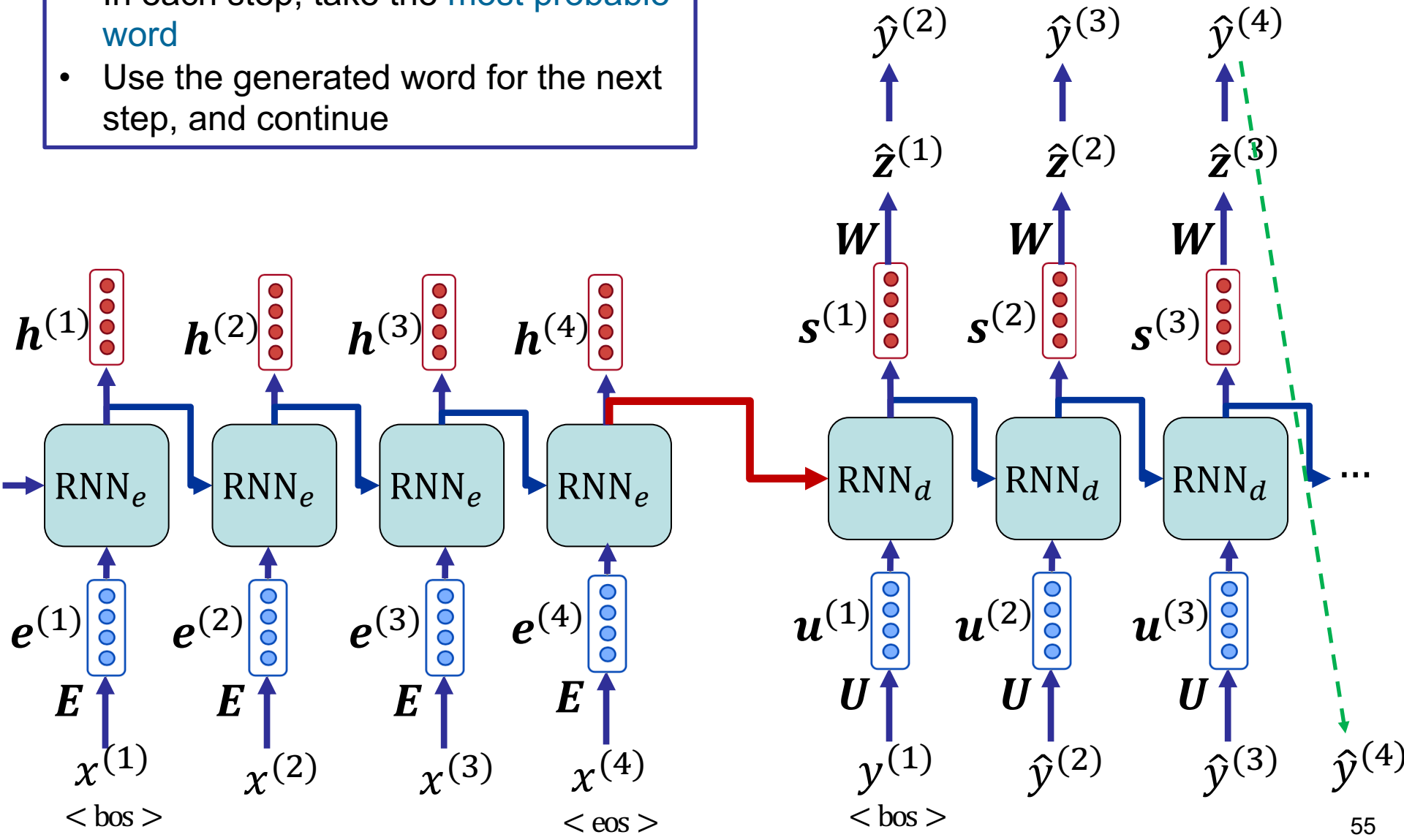
selected word is the one with the highest probability in $\hat{y}^{(2)}$



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 $\hat{y}^{(3)}$



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 $y^{(1\dots l)}$) :

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

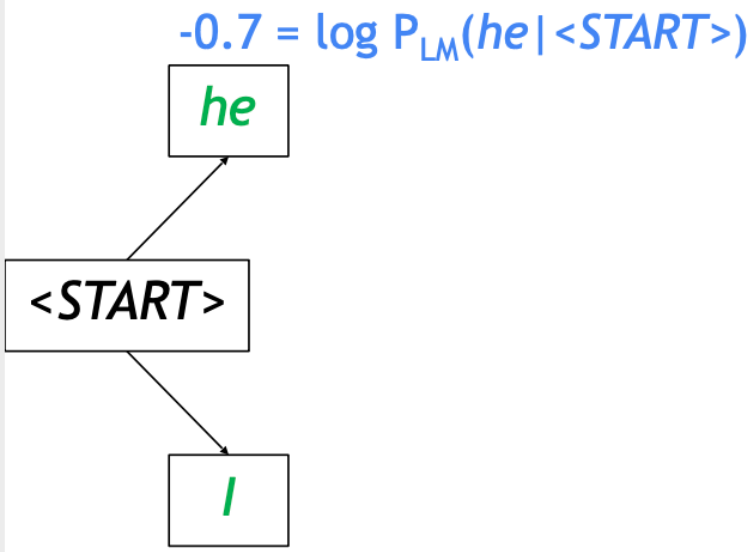
- In each decoding step, we only keep k hypotheses with the highest scores, and don't continue the rest

Beam search decoding – example

<START>

Calculate prob dist of next word

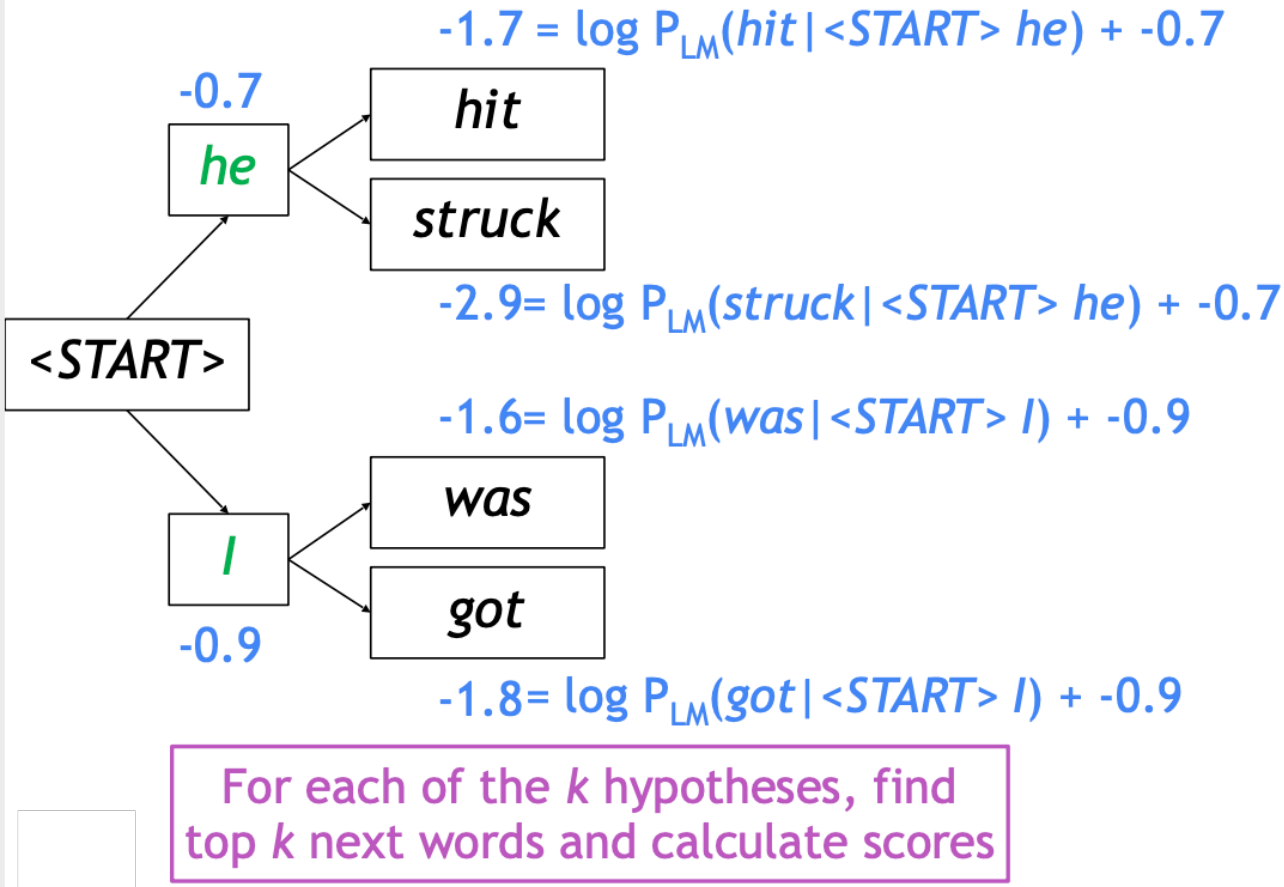
Beam search decoding – example



Take top *k* words and compute scores

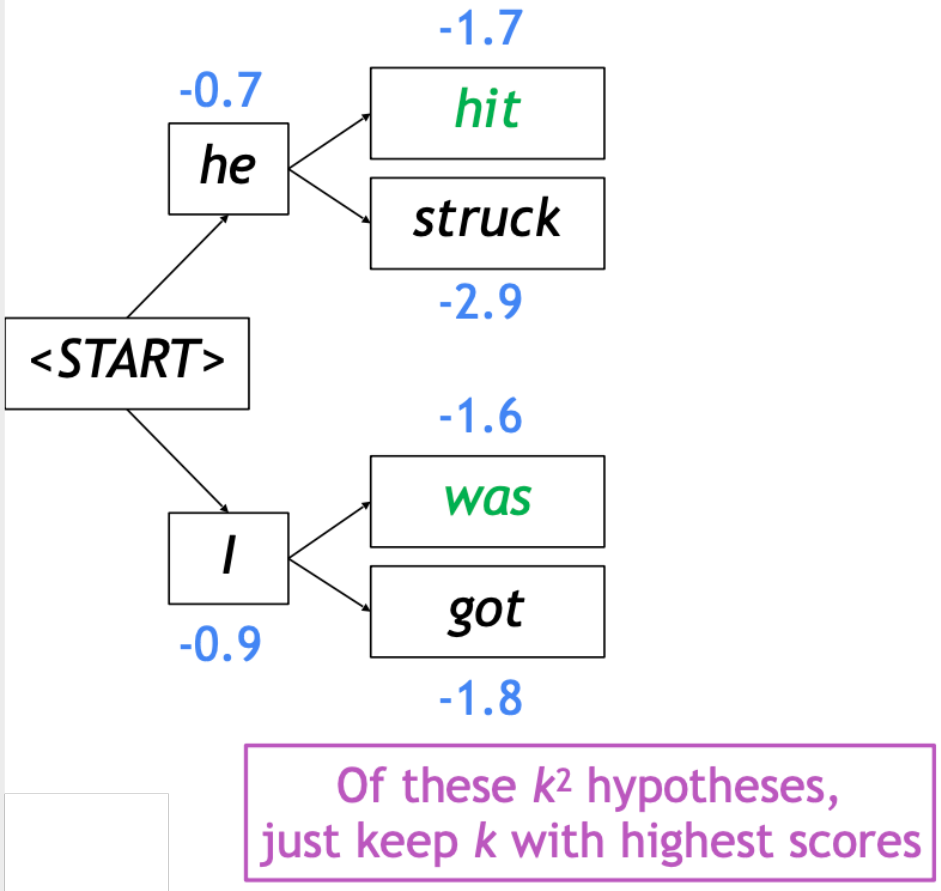
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



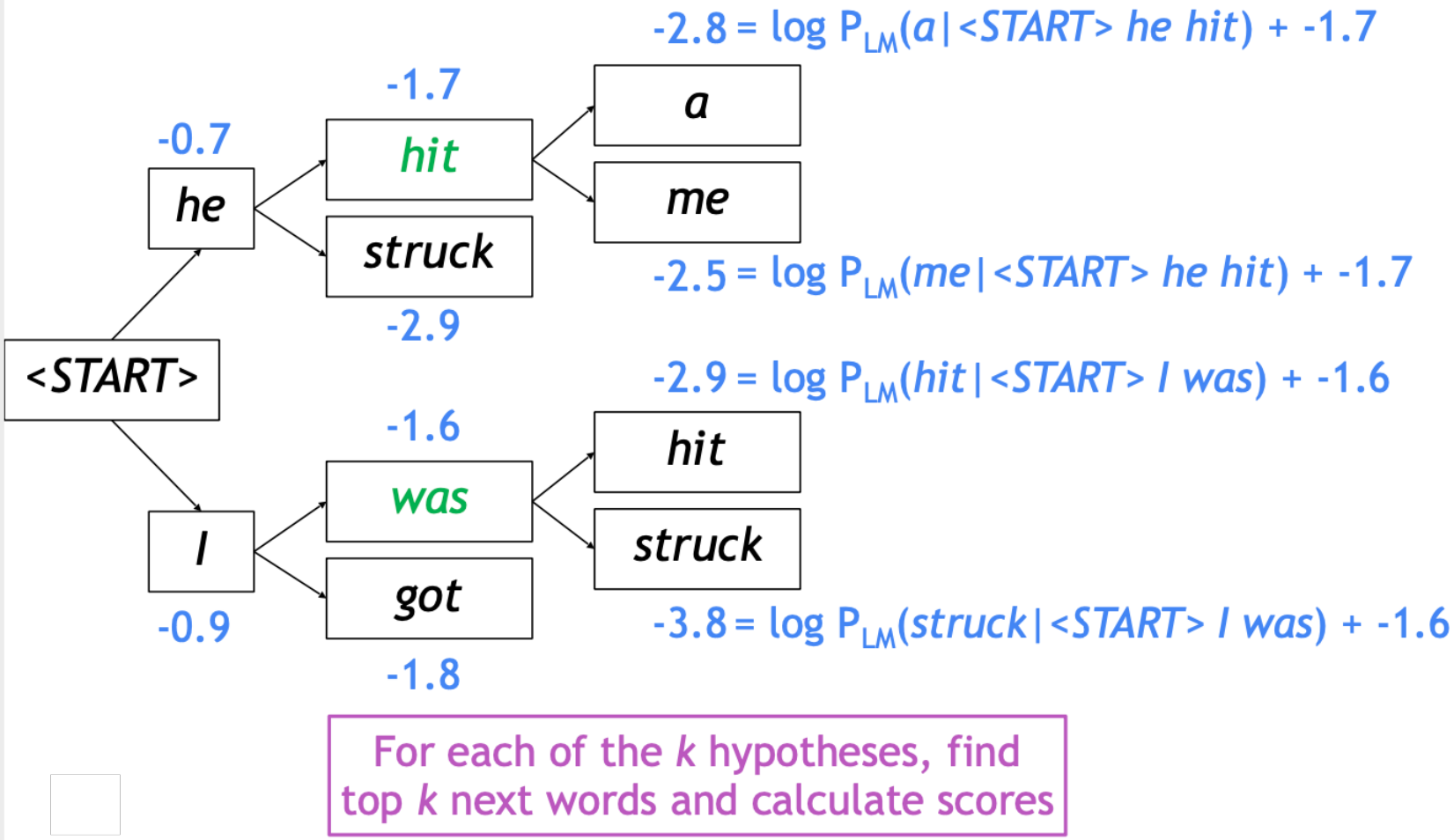
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



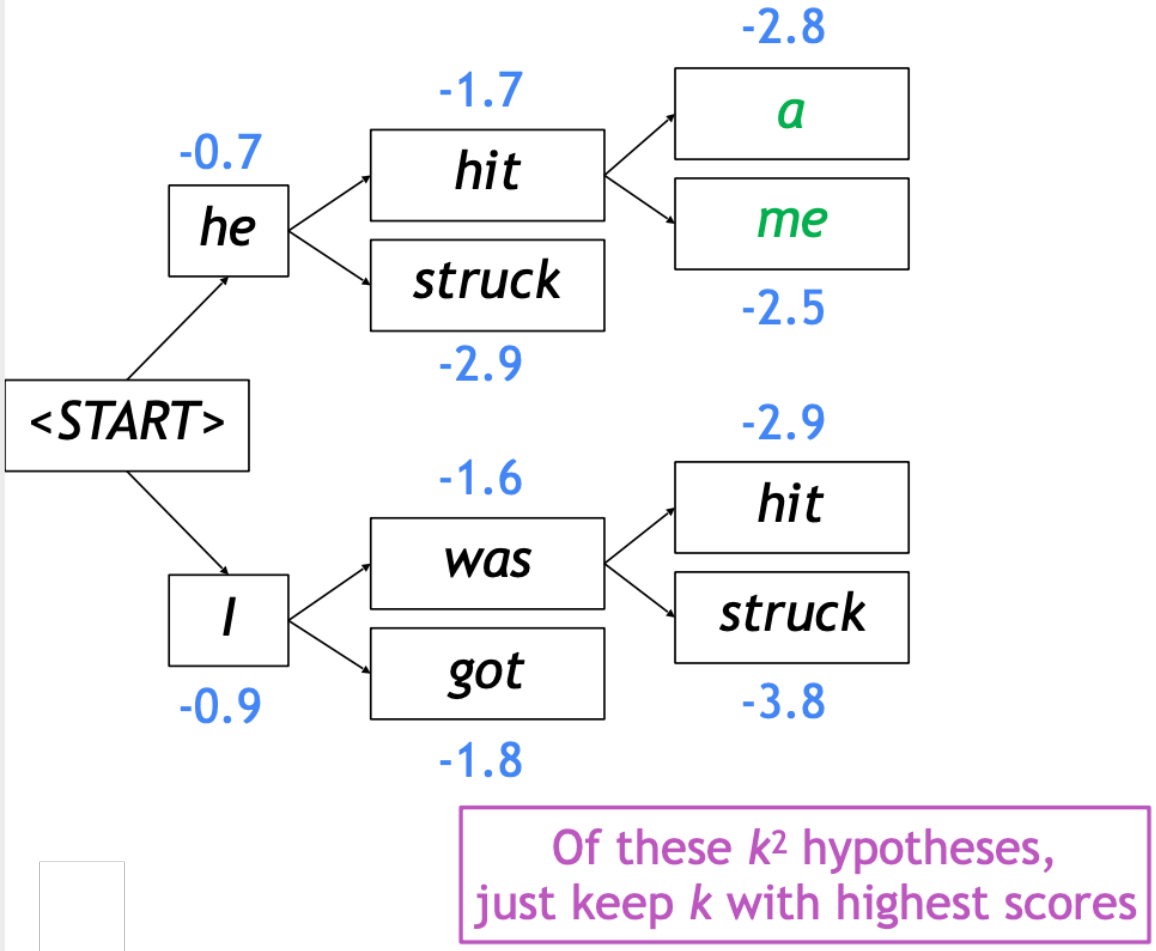
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



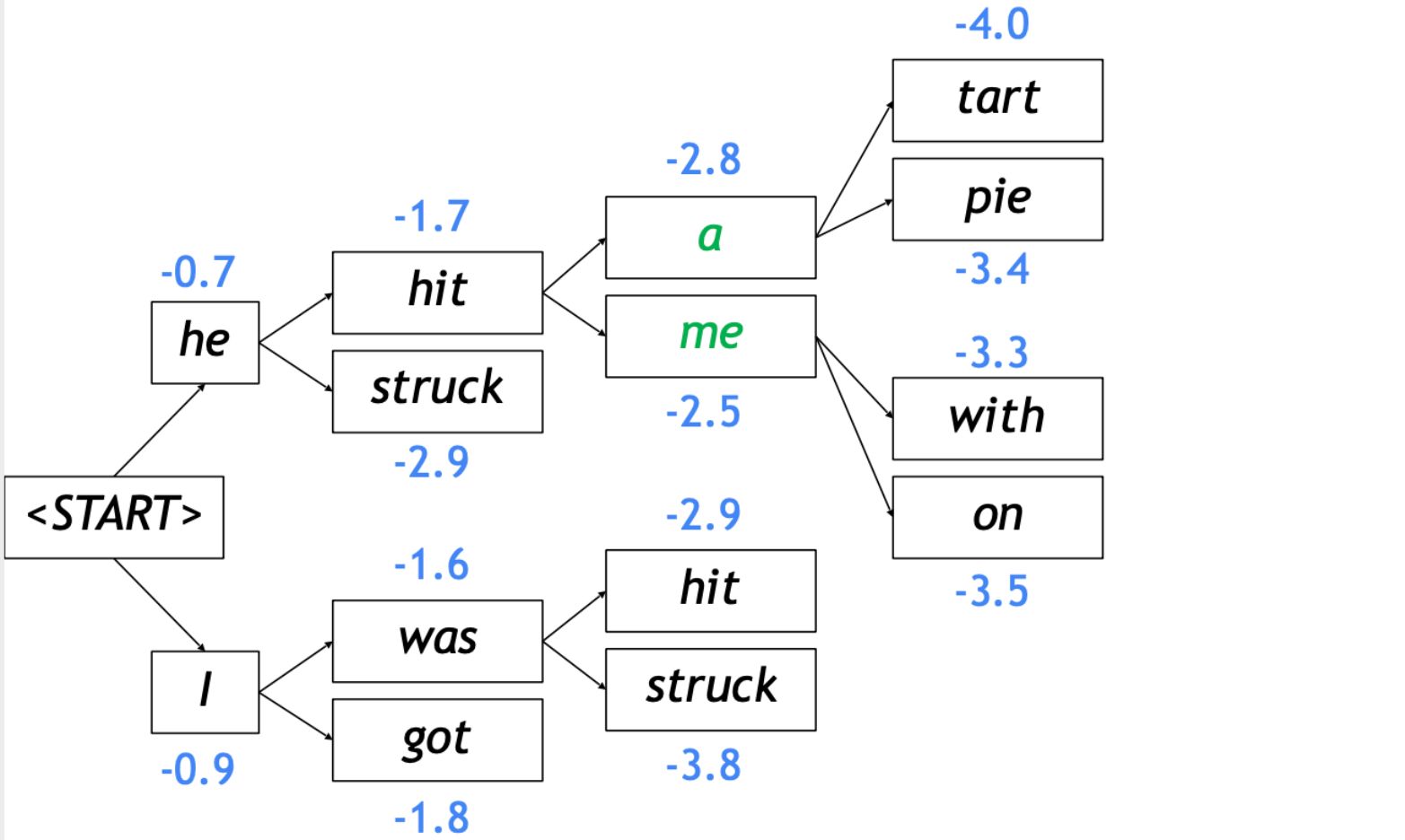
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



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

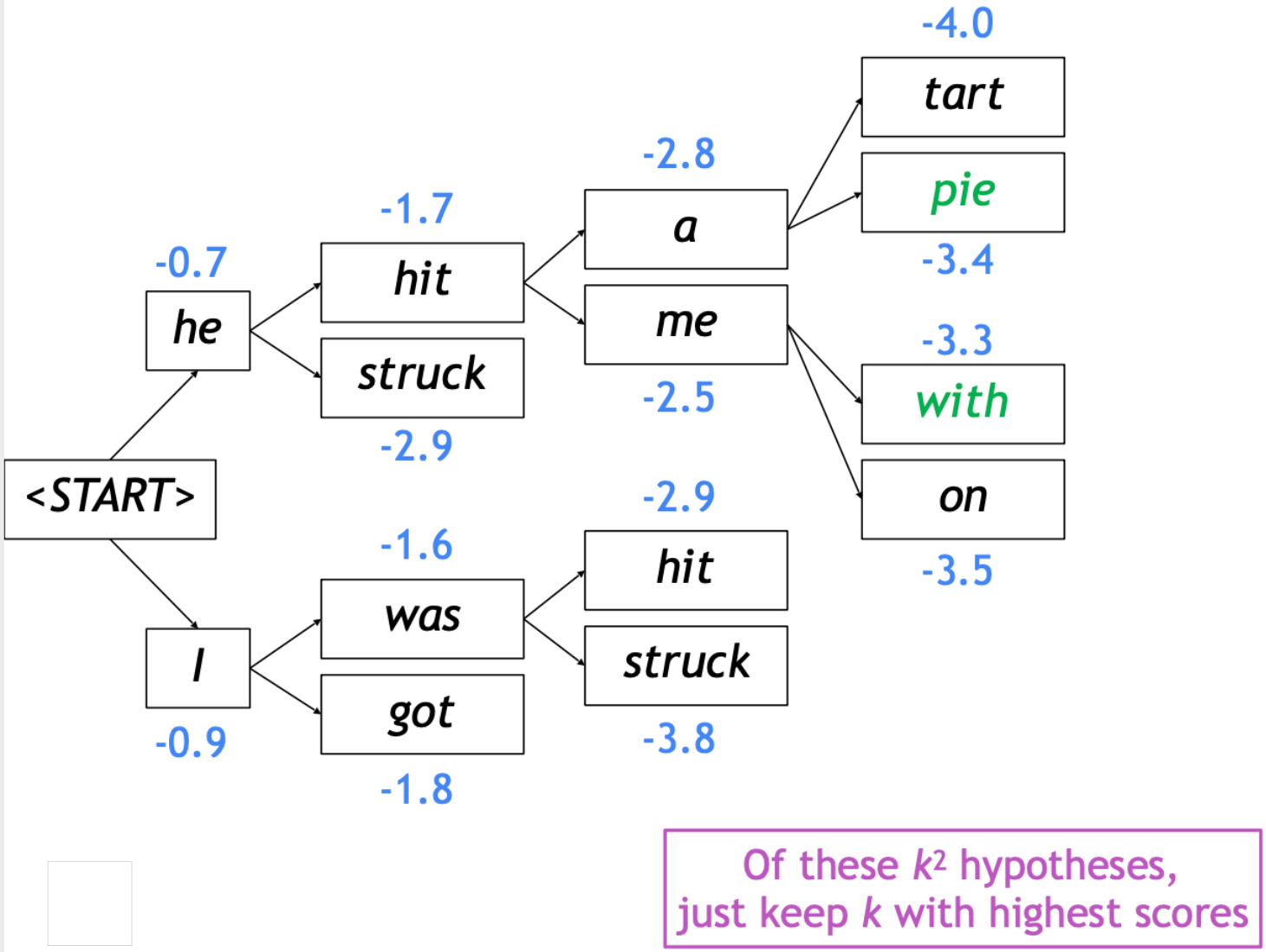
Beam search decoding – example



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

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

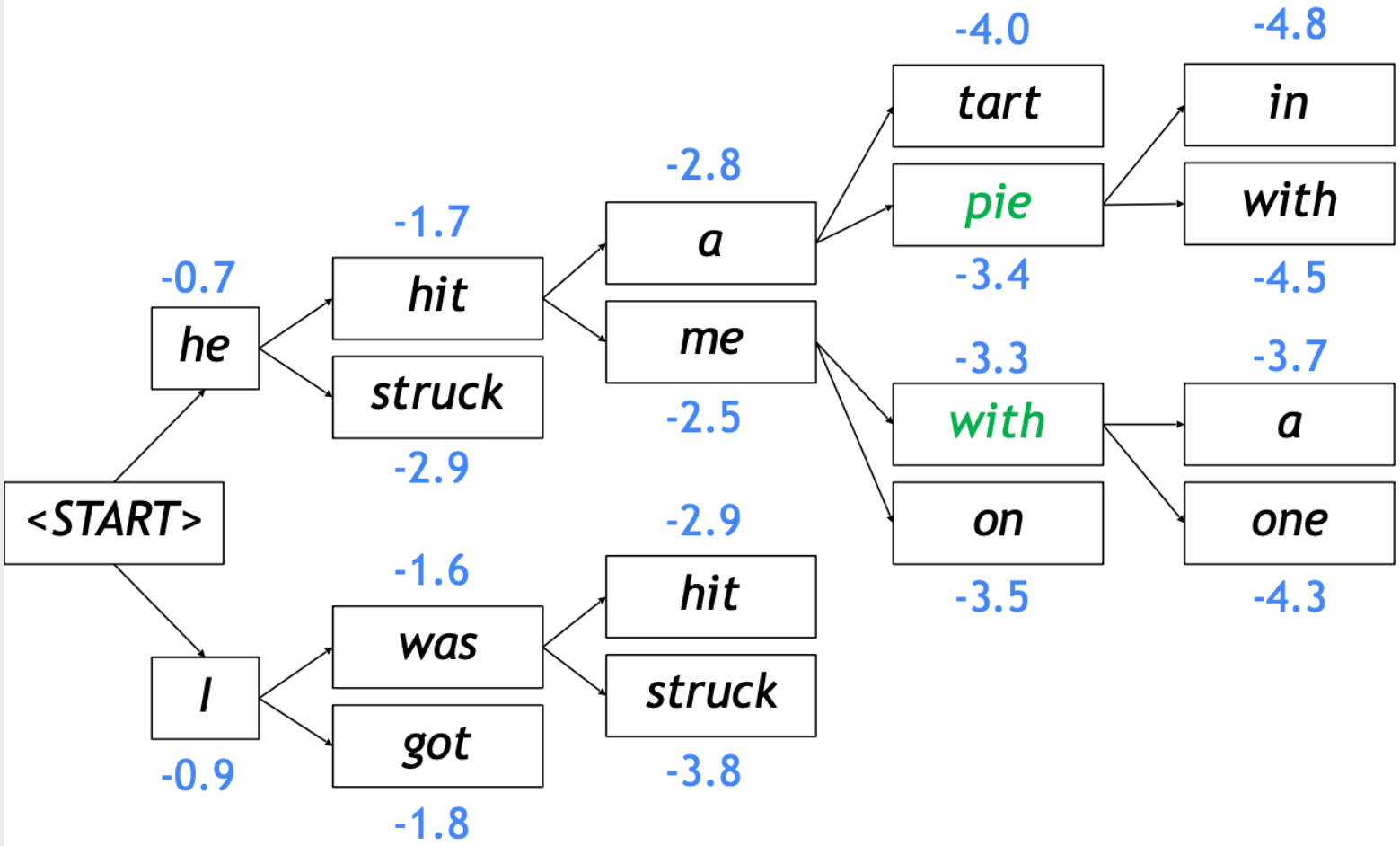
Beam search decoding – example



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

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

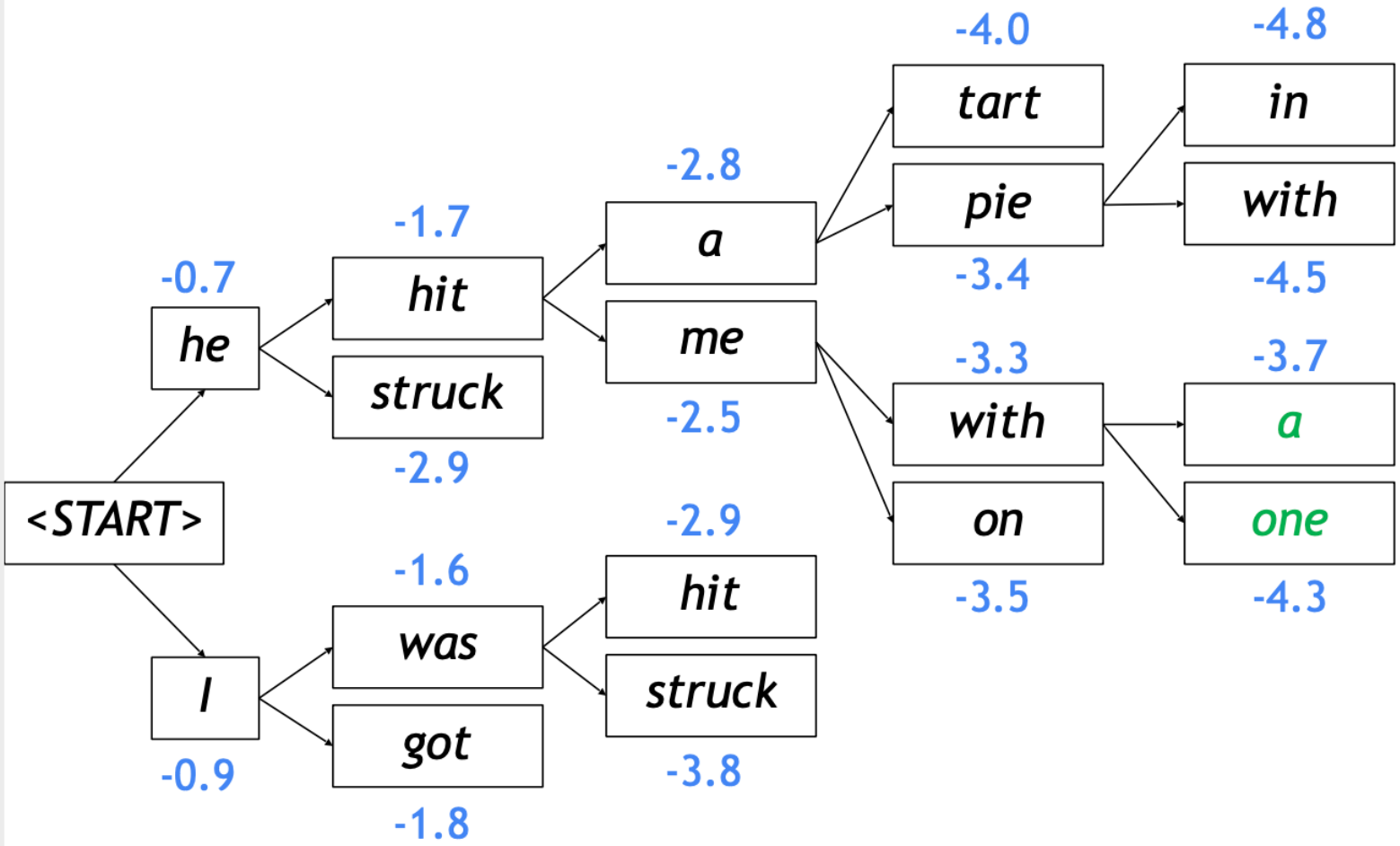
Beam search decoding – example



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

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

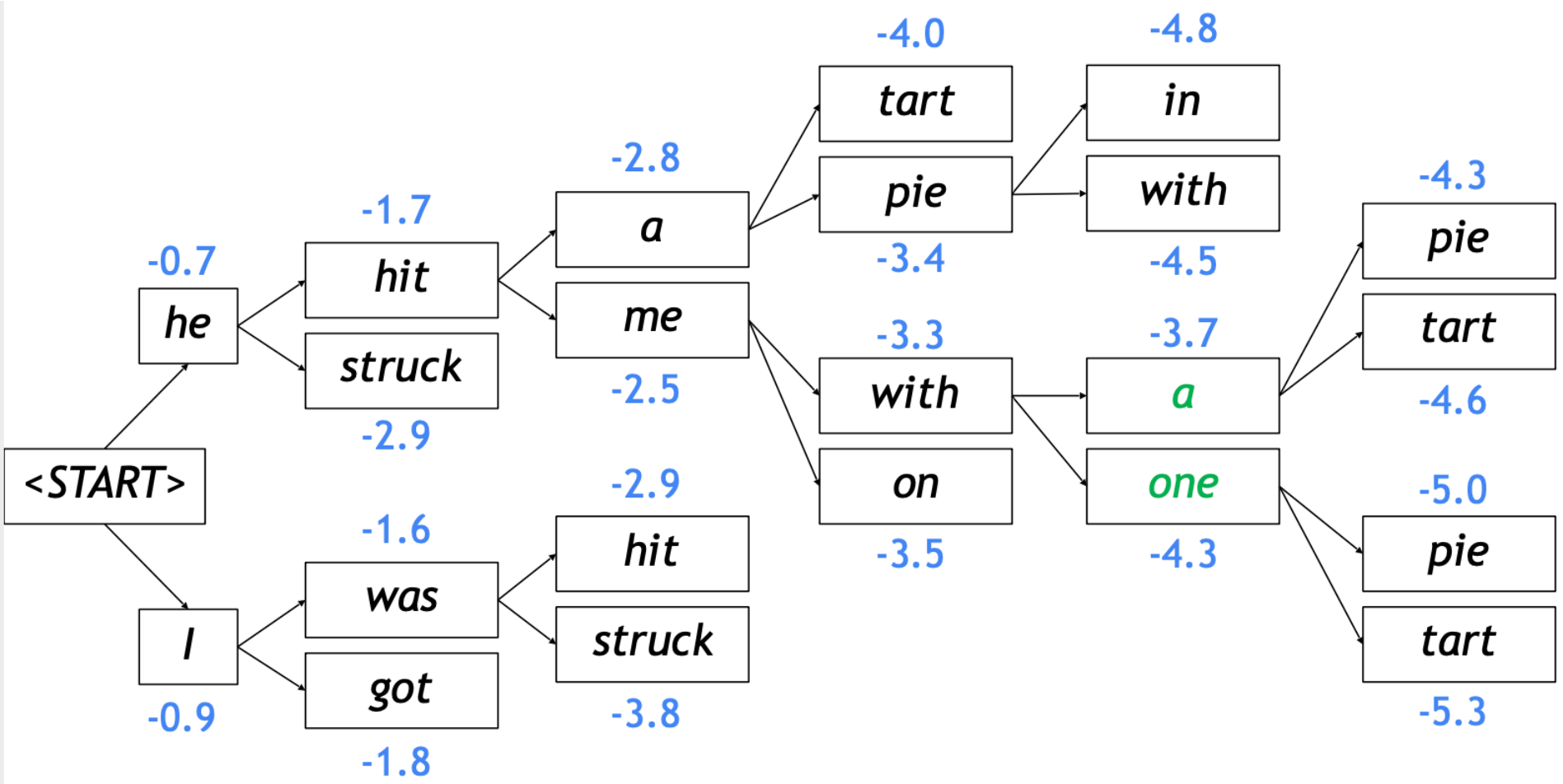
Beam search decoding – example



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

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

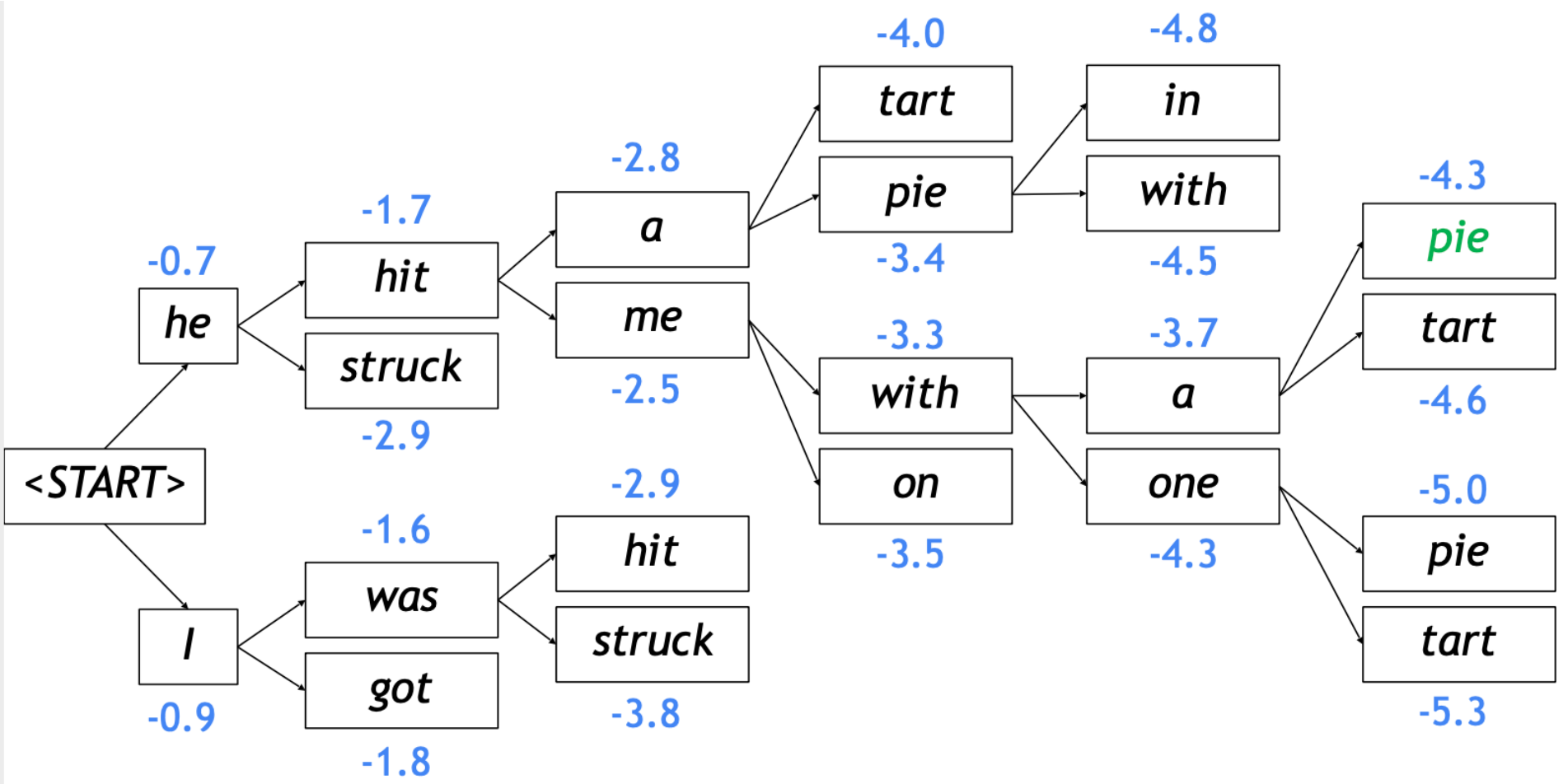
Beam search decoding – example



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

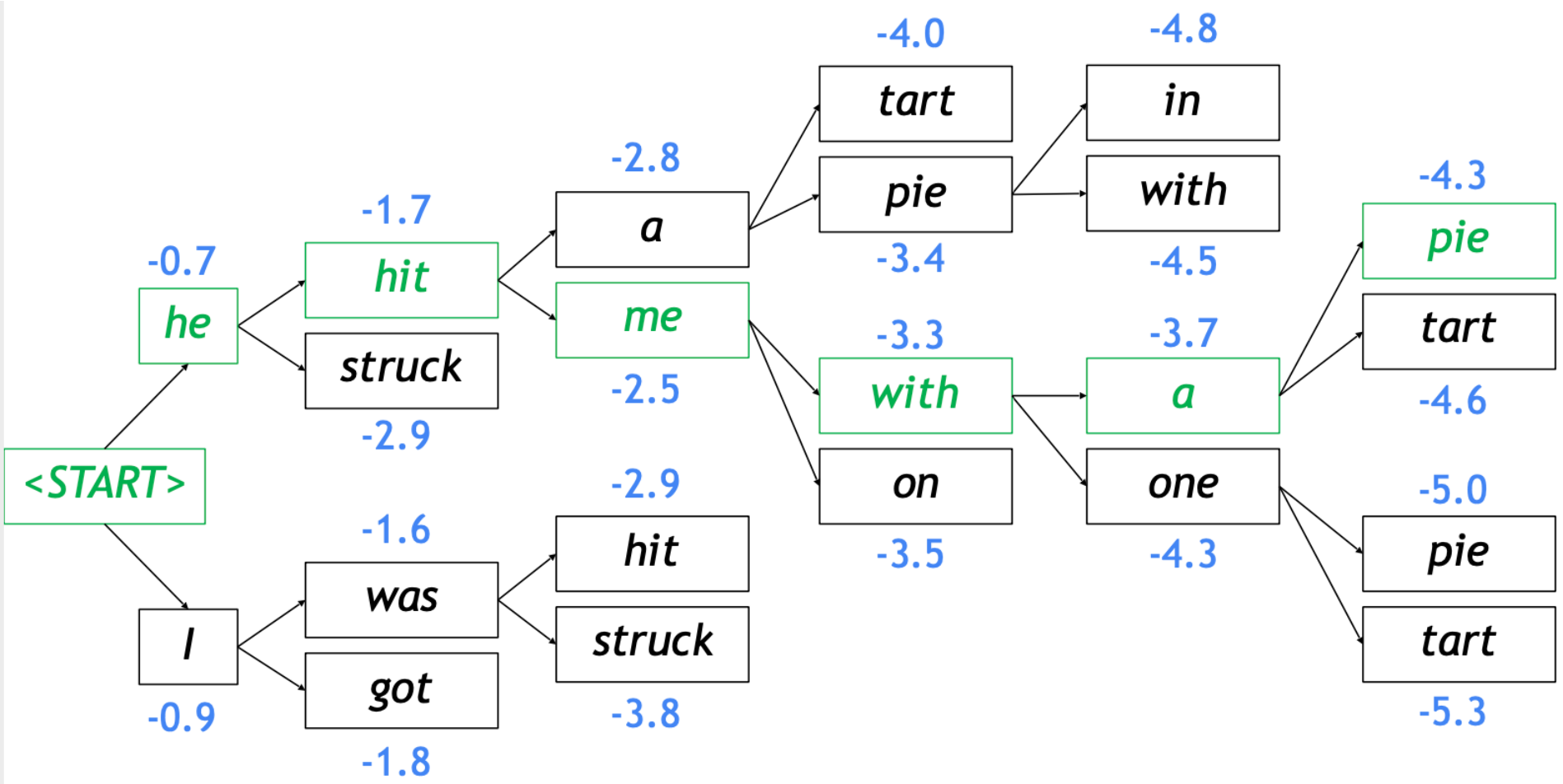
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf

Beam search decoding – example



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)