Summer semester 2022

344.063 KV Special Topic:

Natural Language Processing with Deep Learning

Language Modeling with Recurrent Neural Networks

Navid Rekab-saz

navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- Neural *n*-gram Language Model
- Recurrent Neural Networks
- Language Modeling with RNN

Some slides are adopted from http://web.stanford.edu/class/cs224n/

Probability

- Conditional probability, given two random variables X and Y:

$$P(Y|X)$$

- Probability distribution
  - For a discrete random variable Y with K states (classes)
    - $0 \leq P(Y_i) \leq 1$
    - $\sum_{i=1}^{K} P(Y_i) = 1$
  - E.g. with $K = 4$ states: $[0.2 \quad 0.3 \quad 0.45 \quad 0.05]$

Agenda

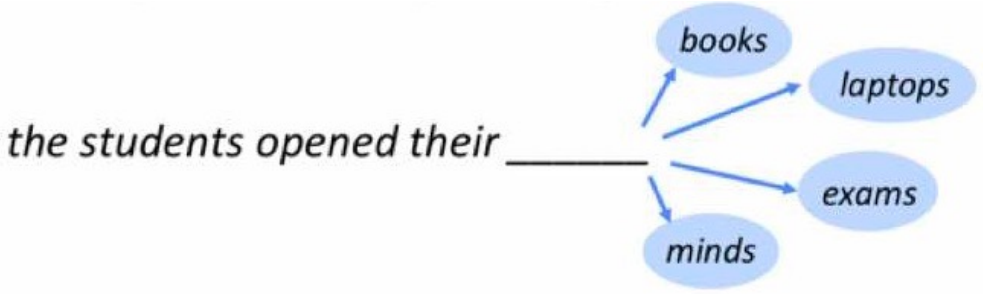
- **Neural *n*-gram Language Model**
- Recurrent Neural Networks
- Language Modeling with RNN

Language Modeling

- Language Modeling is the task of predicting a word/subword/character/etc. given a context:

$$P(v|\text{context})$$

- A Language Model can answer the questions like



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

Language Modeling – formal definition

- 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 $\mathbb{V} = \{v1, v2, \ldots, vN\}$

☞ Note: this is the definition of directed left-to-right language models.

N-gram

- Recall: a *n*-gram is a chunk of *n* consecutive words.

the students opened their _____

- unigrams: *"the", "students", "opened", "their"*
- bigrams: *"the students", "students opened", "opened their"*
- trigrams: *"the students opened", "students opened their"*
- 4-grams: *"the students opened their"*

N-gram Language Model

- Markov assumption: decision at time t depends only on the current state

- In *n*-gram Language Model: predicting $x^{(t+1)}$ depends on preceding *n-1* words

- Without Markovian assumption:

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

- *n*-gram Language Model:

$$P(x^{(t+1)}|\underbrace{x^{(t)}, x^{(t-1)}, \ldots, x^{(t-n+2)}}_{n-1 \text{ words}})$$

N-gram language modeling with neural networks

Recall

- The aim of a *n*-gram Language Model is to calculate:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right)$$

- We can use a feed forward neural network to estimate this probability
- Immediate benefits:
  - Smooth probability estimation
  - Exploiting the semantic space of word embeddings (probably better generalization)

Basic idea from: Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137-1155.

Neural *n*-gram LM – preparing training data

- Preparing training data for a neural 4-gram Language Model in the form of $(\text{context}, \text{next word})$, namely $(x^{(t-2)}x^{(t-1)}x^{(t)}, x^{(t+1)})$:
- For a given text corpus:

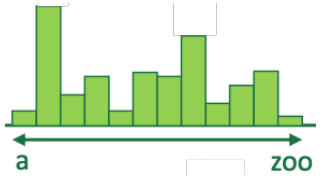
```
a fluffy cat sunbathes on the bank of river …
```

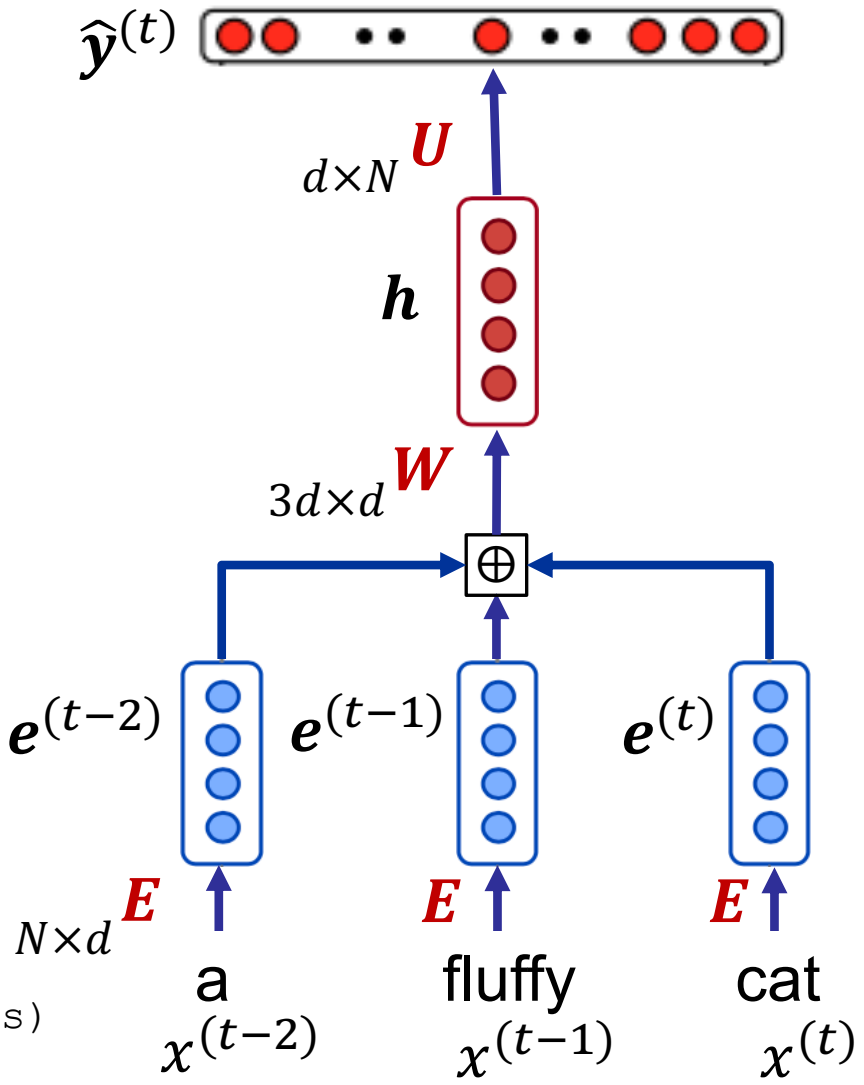
- Training data items would be:

```
(<bos> <bos> <bos>, a)
(<bos> <bos> a, fluffy)
(<bos> a fluffy, cat)
(a fluffy cat, sunbathes)
(fluffy cat sunbathes, on)
(cat sunbathes on, the)
(sunbathes on the, bank)
…
```

Neural *n*-gram Language Model – architecture

$P(\textit{sunbathes}|\textit{a fluffy cat})$





A data item in training data:
`(a fluffy cat, sunbathes)`

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 $\boldsymbol{x}^{(t)}$)
- Concatenation of word embeddings: $\boldsymbol{e} = [\boldsymbol{e}^{(t-2)}, \boldsymbol{e}^{(t-1)}, \boldsymbol{e}^{(t)}]$
- Hidden layer: $\boldsymbol{h} = \tanh(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b})$

Decoder

- Predicted probabilities:
  - Predicted probability distribution:

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

  - Probability of any next word v at step t:

$$P\left(v \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \hat{y}_v^{(t)}$$

Model parameters

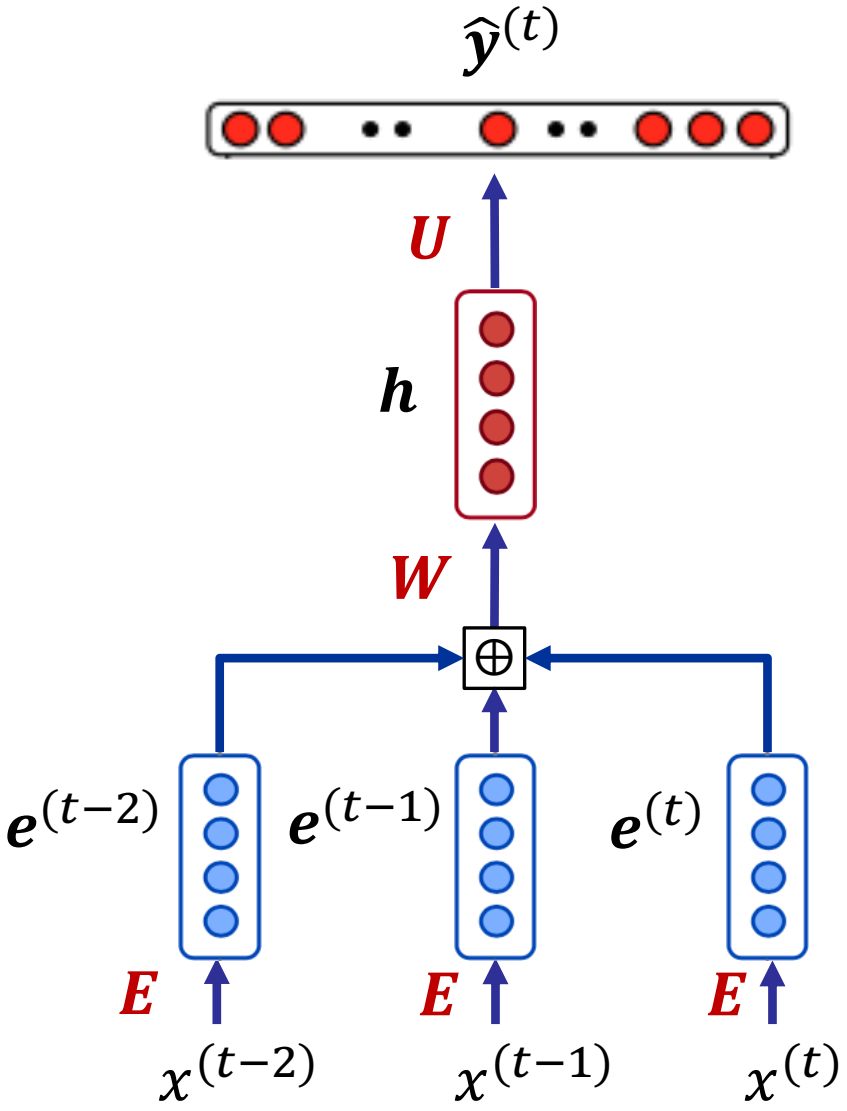
- $\boldsymbol{E} \rightarrow N \times h$
- $\boldsymbol{W} \rightarrow (n \times h) \times h$
- $\boldsymbol{U} \rightarrow h \times N$

h embeddings dimension

n number of preceding words

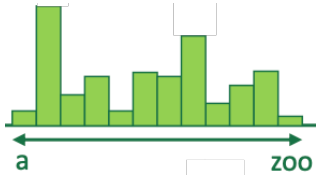
$\boldsymbol{E}$ is called encoder embedding

$\boldsymbol{U}$ is called decoder embedding or output projection



Loss function

$P(\textit{sunbathes}|\textit{a fluffy cat})$



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

$\hat{\boldsymbol{y}}^{(t)}$

$\boldsymbol{U}$

$\boldsymbol{h}$

$\boldsymbol{W}$

$\boldsymbol{e}^{(t-2)}$ $\boldsymbol{e}^{(t-1)}$ $\boldsymbol{e}^{(t)}$

$\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$

a fluffy cat

$x^{(t-2)}$ $x^{(t-1)}$ $x^{(t)}$

A data item in training data:
`(a fluffy cat, sunbathes)`

Training procedure

- Start with a large text corpus: $x^{(1)}, \ldots, x^{(T)}$
- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$ given *n* previous words
- Loss function at t is Negative Log Likelihood of the predicted probability of the word at $x^{(t+1)}$

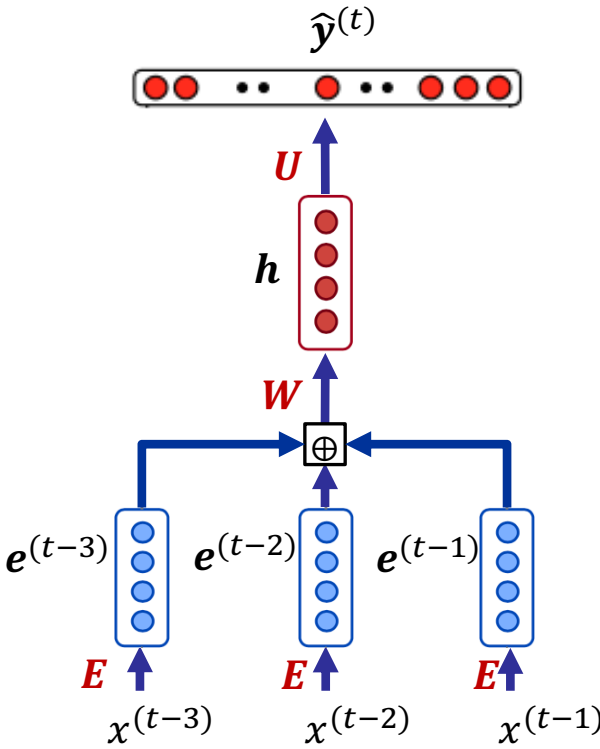
$$\mathcal{L}^{(t)} = -\log \hat{y}^{(t)}_{x^{(t+1)}}$$

- Overall loss is the average over all time steps:

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

Neural *n*-gram LMs – summary

- Neural *n*-gram LMs predict co-occurrence probabilities
- Neural *n*-gram LMs provide a smooth probability distribution
- At inference time, neural *n*-gram LMs require a forward pass



Agenda

- Neural *n*-gram Language Model
- **Recurrent Neural Networks**
- Language Modeling with RNN

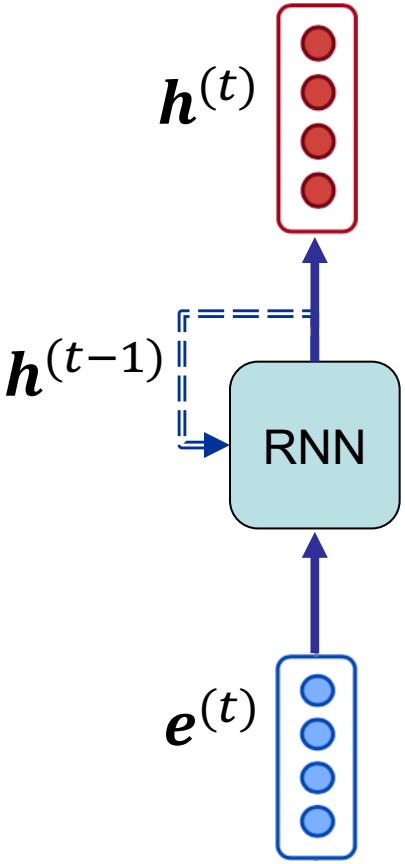
Recurrent Neural Network

- Recurrent Neural Network (RNN) encodes/embeds a sequential input of any size into compositional embeddings

- A sequence can be …
  - a stream of word/subword/character vectors
  - time series
  - etc.

- RNN models …
  - capture dependencies through the sequence
  - apply the same parameters repeatedly
  - output a final embedding but also intermediary embeddings on each time step

Recurrent Neural Networks



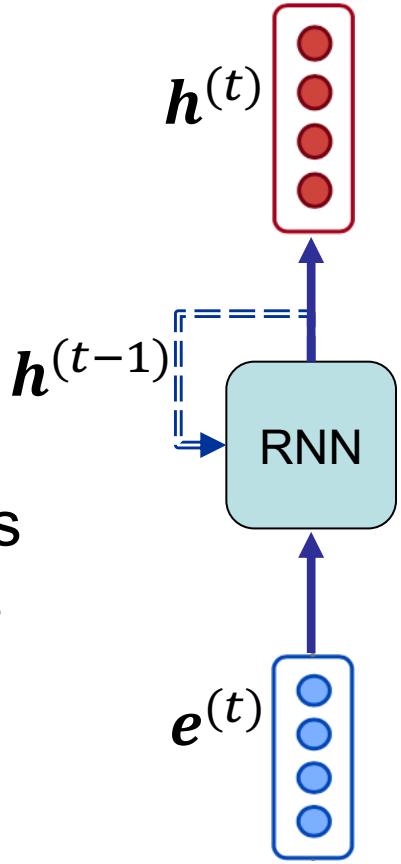
Recurrent Neural Networks

- Output $\boldsymbol{h}^{(t)}$ is a function of input $\boldsymbol{e}^{(t)}$ and the output of the previous time step $\boldsymbol{h}^{(t-1)}$

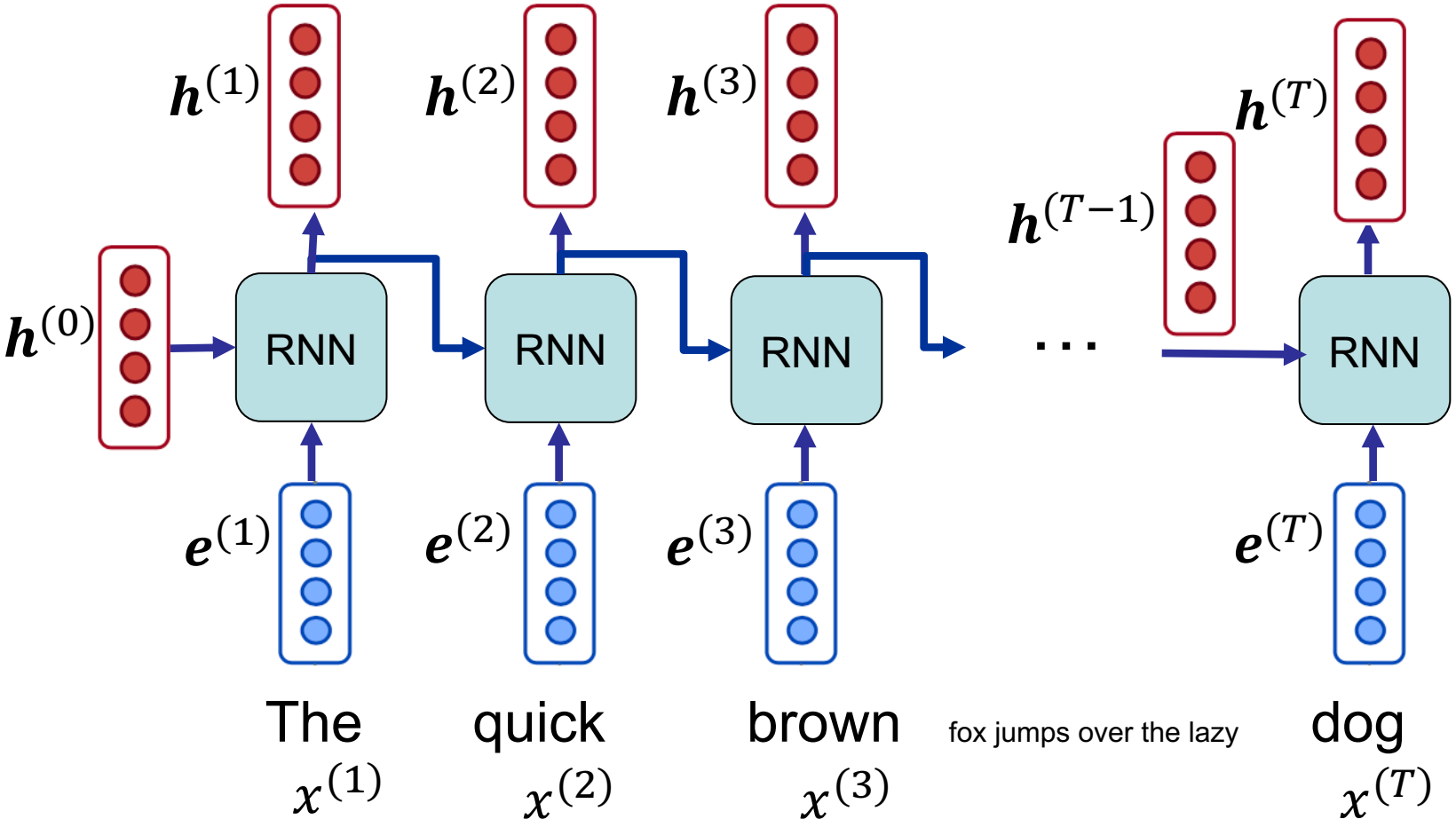
$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

- $\boldsymbol{h}^{(t)}$ is called hidden state

- With hidden state $\boldsymbol{h}^{(t-1)}$, the model accesses to a sort of memory from all previous entities



RNN – Unrolling



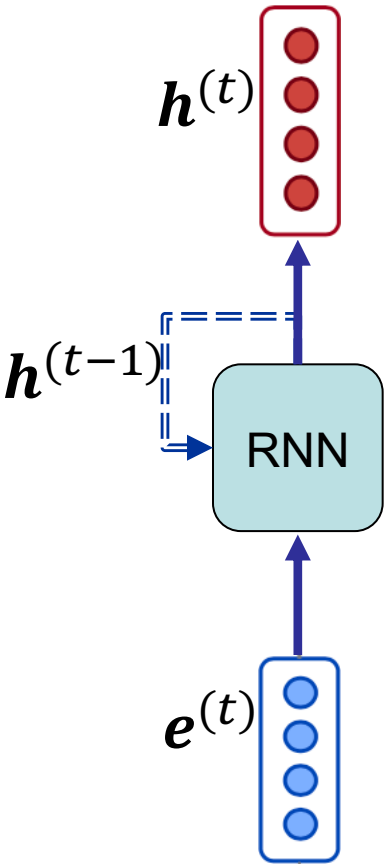
Standard (Elman) RNN

- General form of an RNN function

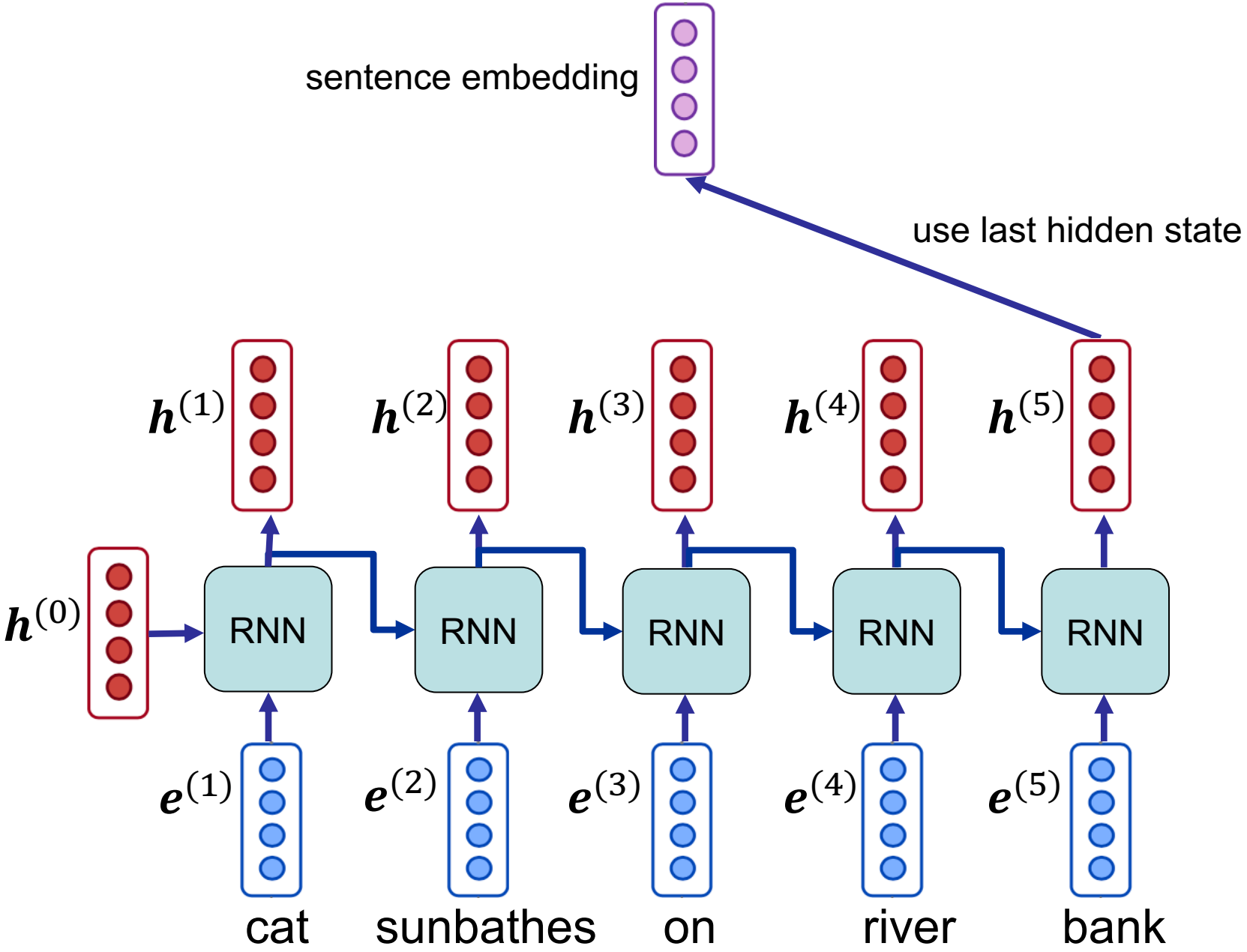
$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

- Standard RNN:
  - linear projection of the previous hidden state $\boldsymbol{h}^{(t-1)}$
  - linear projection of input $\boldsymbol{e}^{(t)}$
  - summing the projections and applying a non-linearity

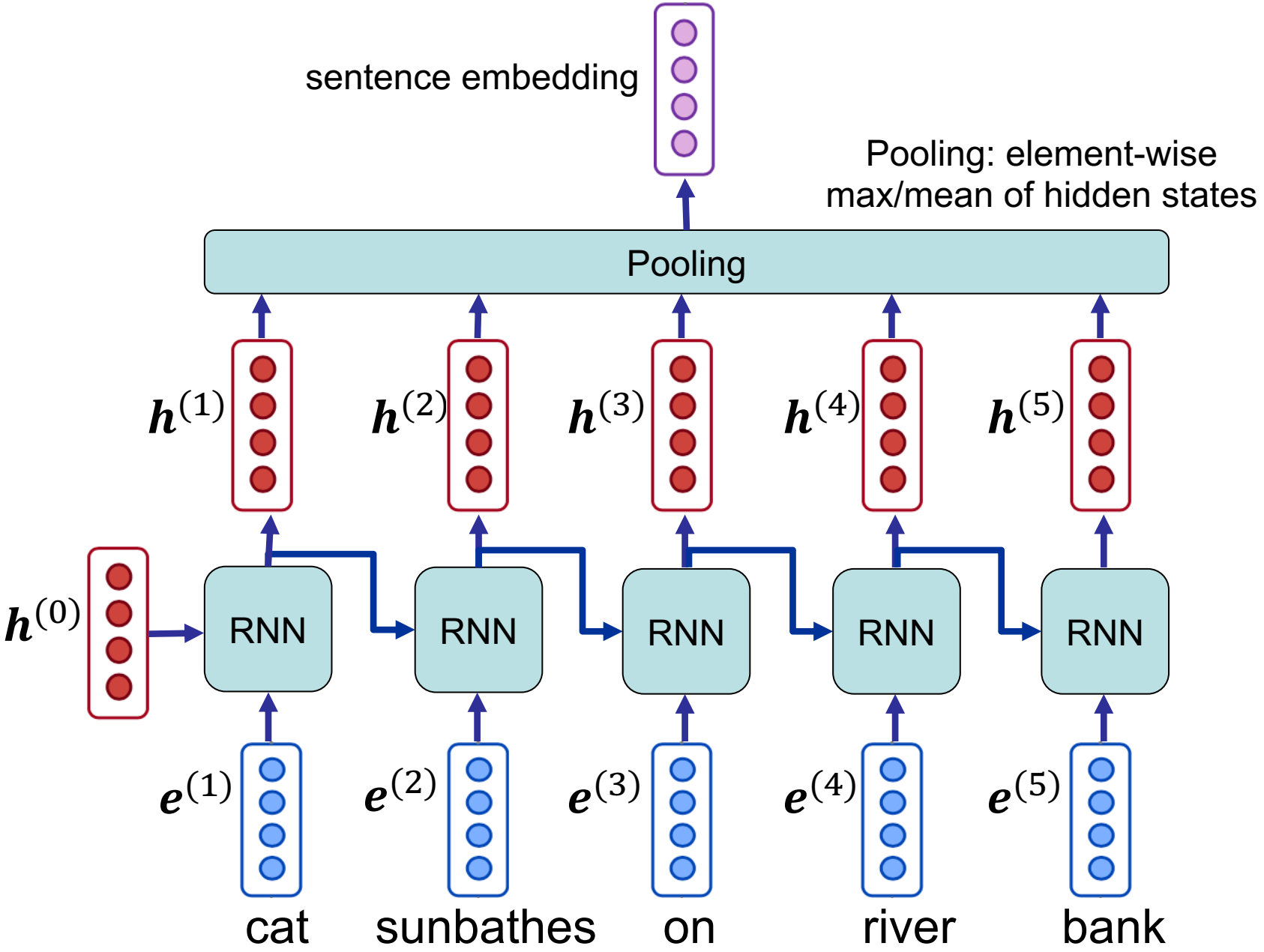
$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b})$$



RNN – Compositional embedding



RNN – Compositional embedding



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)

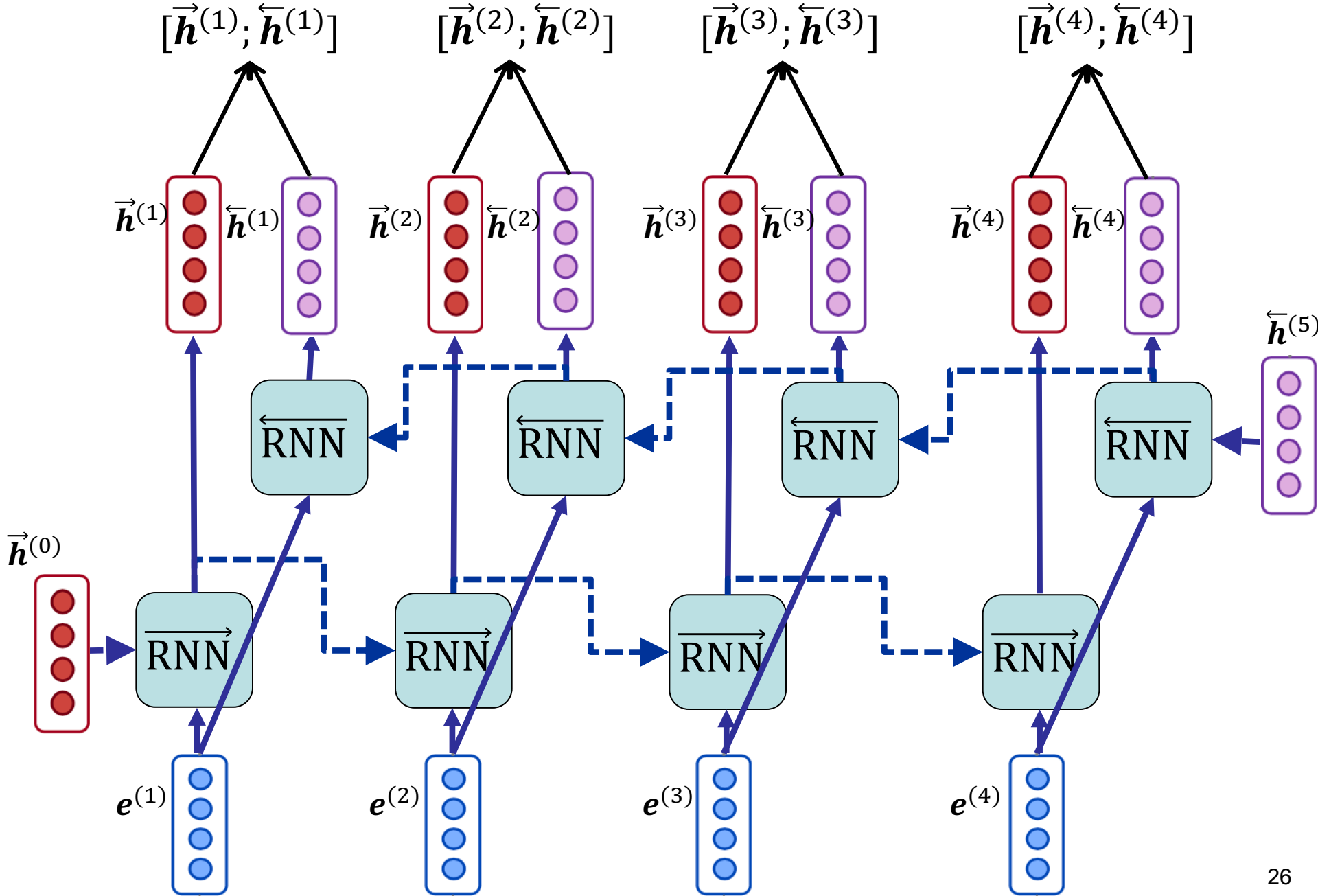
$$\overrightarrow{\boldsymbol{h}}^{(t)} = \overrightarrow{\mathrm{RNN}}\left(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{e}^{(t)}\right)$$

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

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

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

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


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

Agenda

- Neural *n*-gram Language Model
- Recurrent Neural Networks
- **Language Modeling with RNN**

Language Modeling – recap

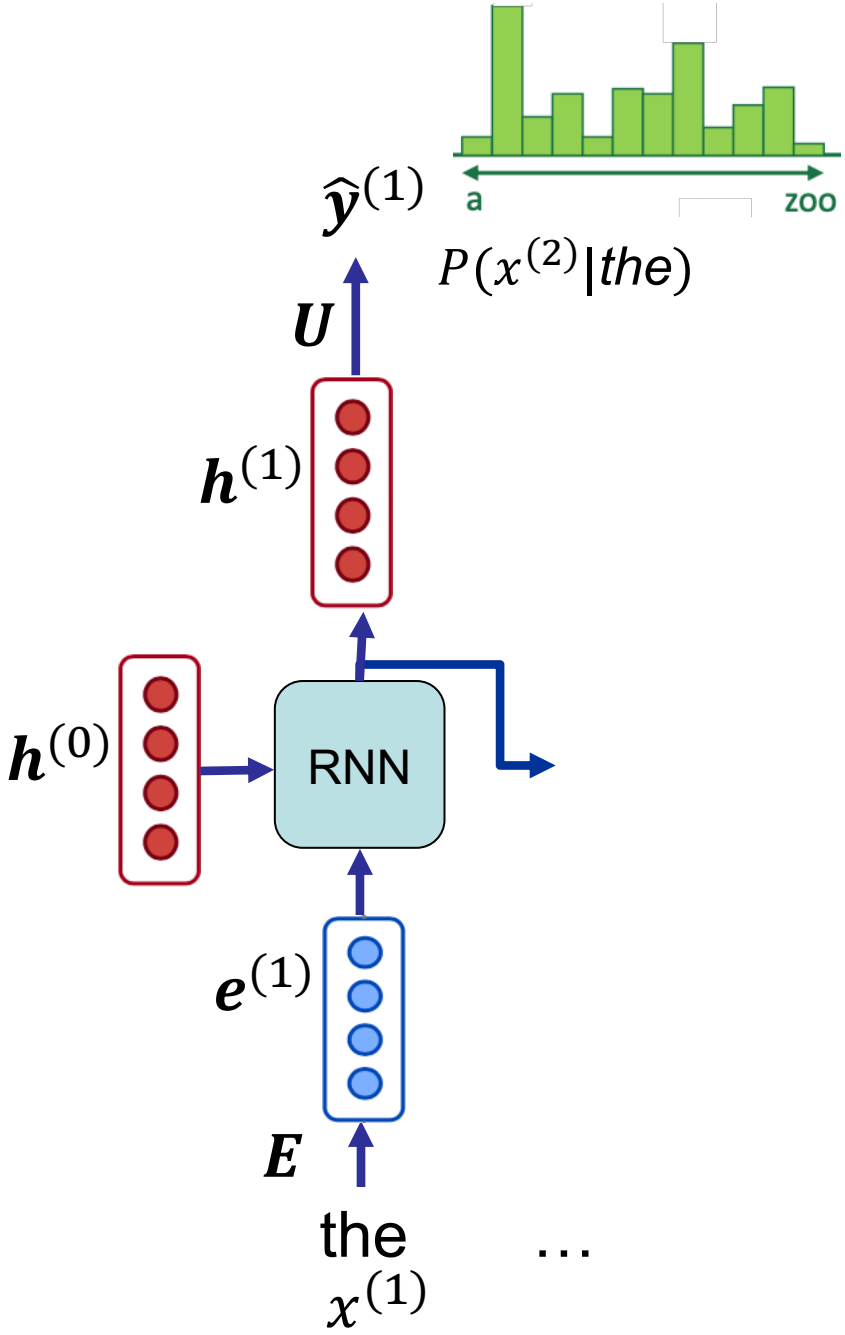
- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, a language model calculates the probability distribution of next word $x^{(t+1)}$ over all words in vocabulary

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

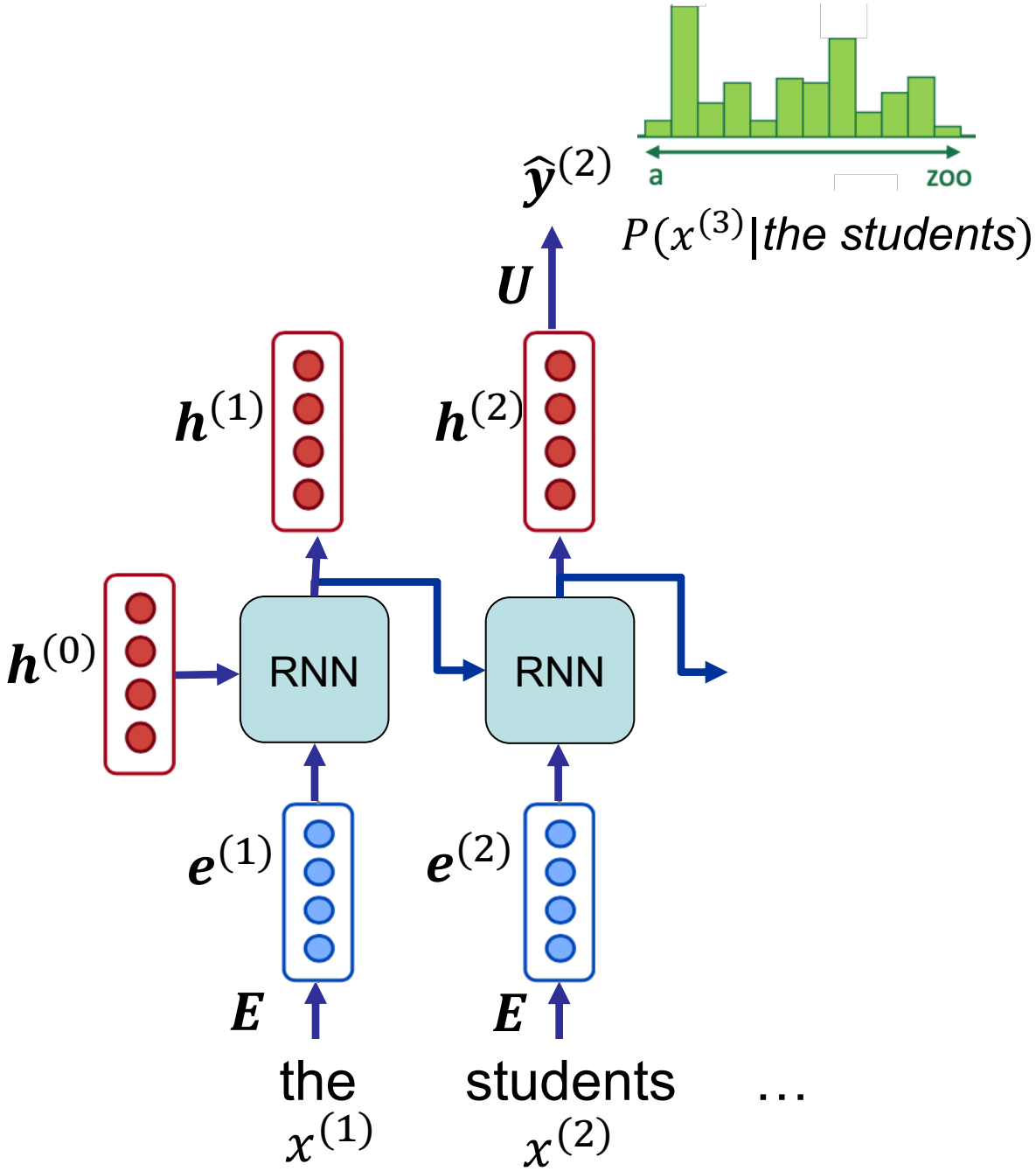
x is any word in the vocabulary $\mathbb{V} = \{v1, v2, \ldots, vN\}$

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

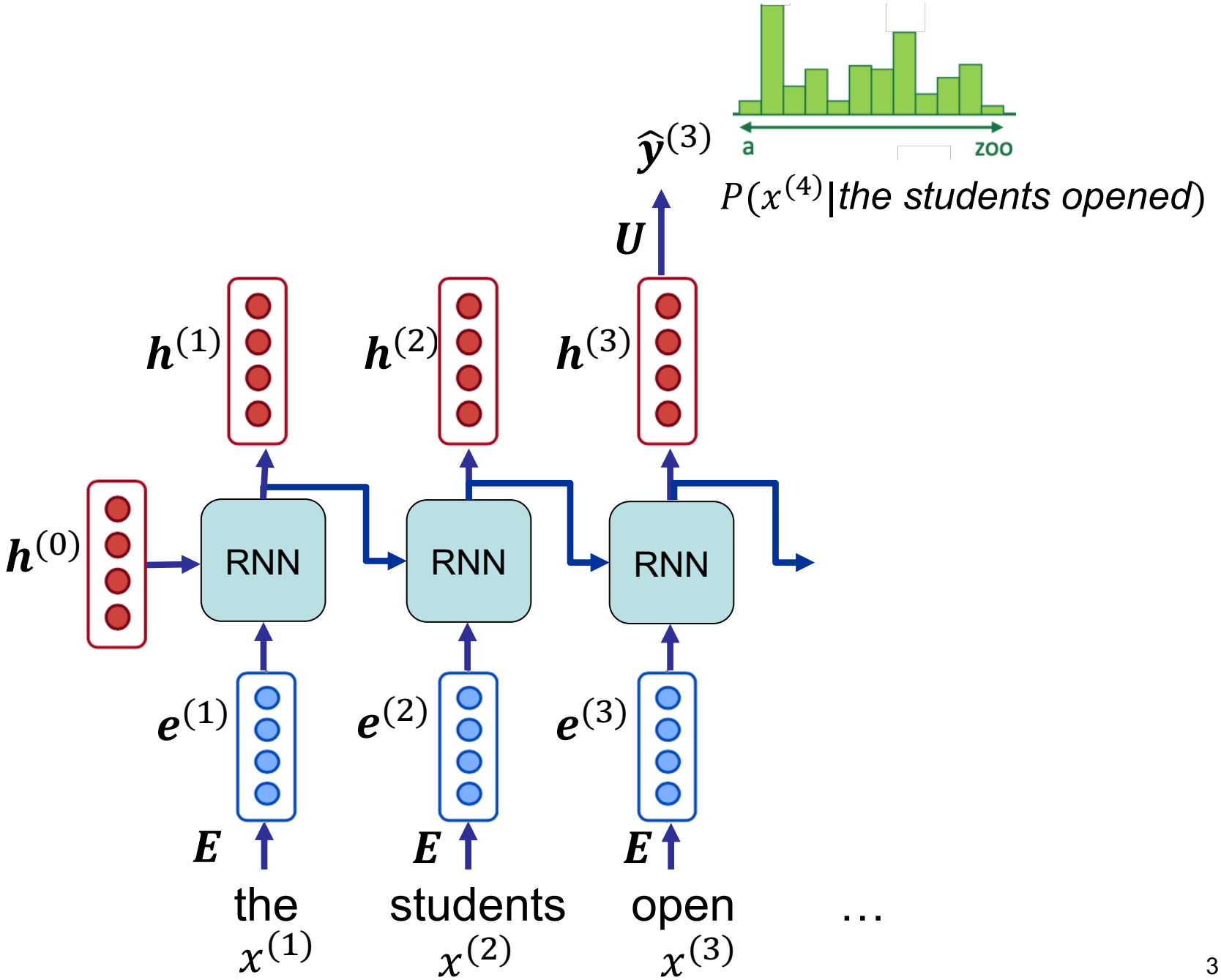
RNN Language Model



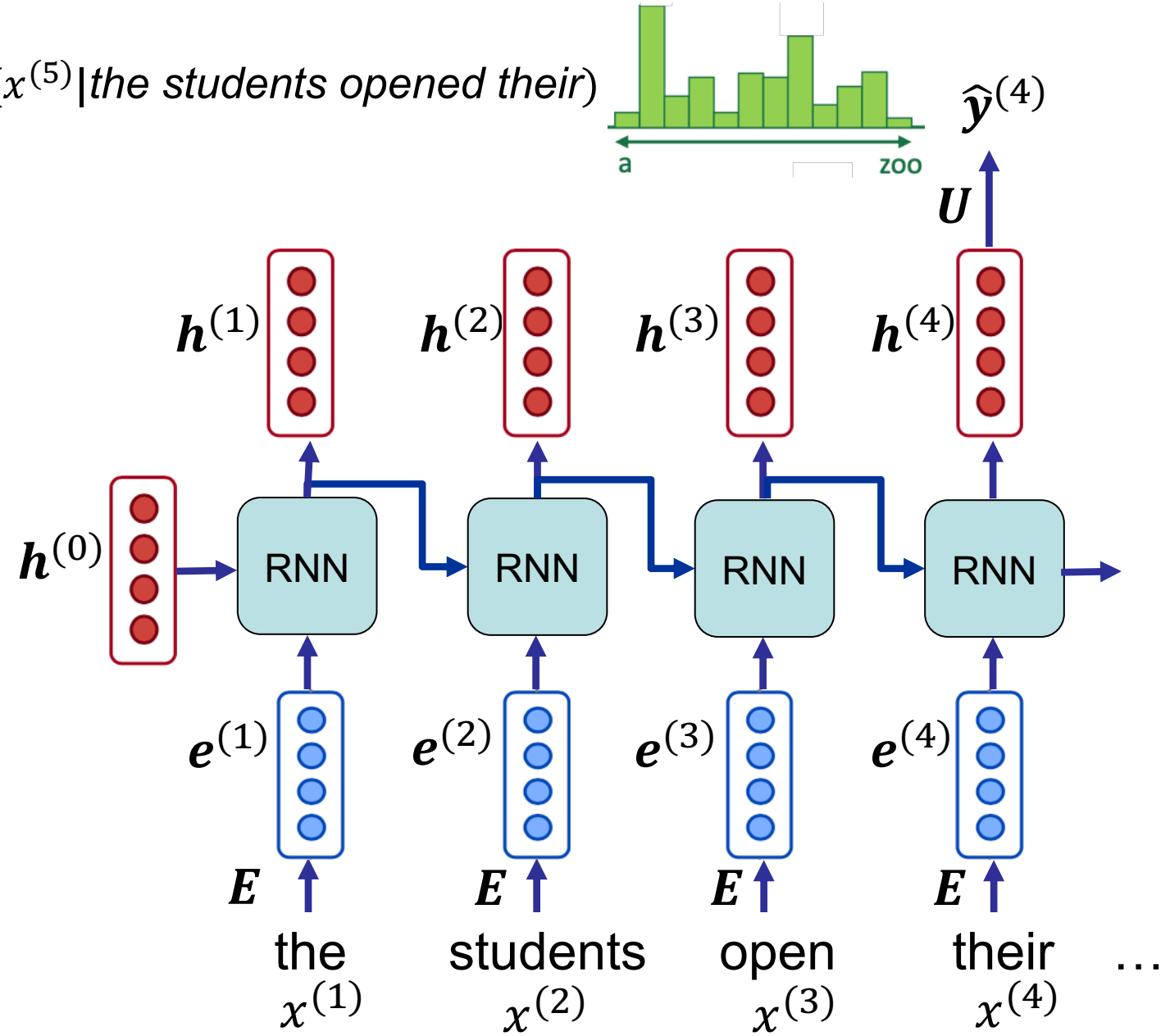
RNN Language Model



RNN Language Model



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:

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

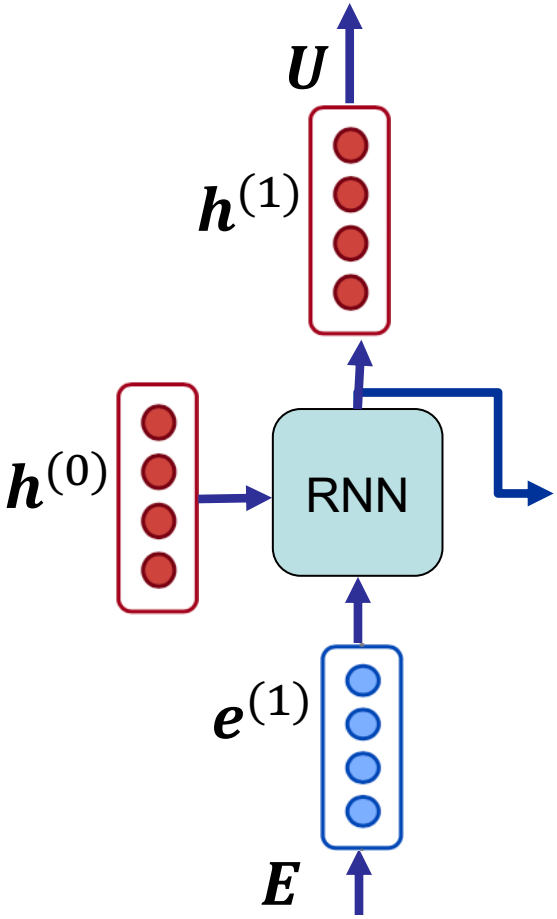
  - Probability of any word v at step t:

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

Training

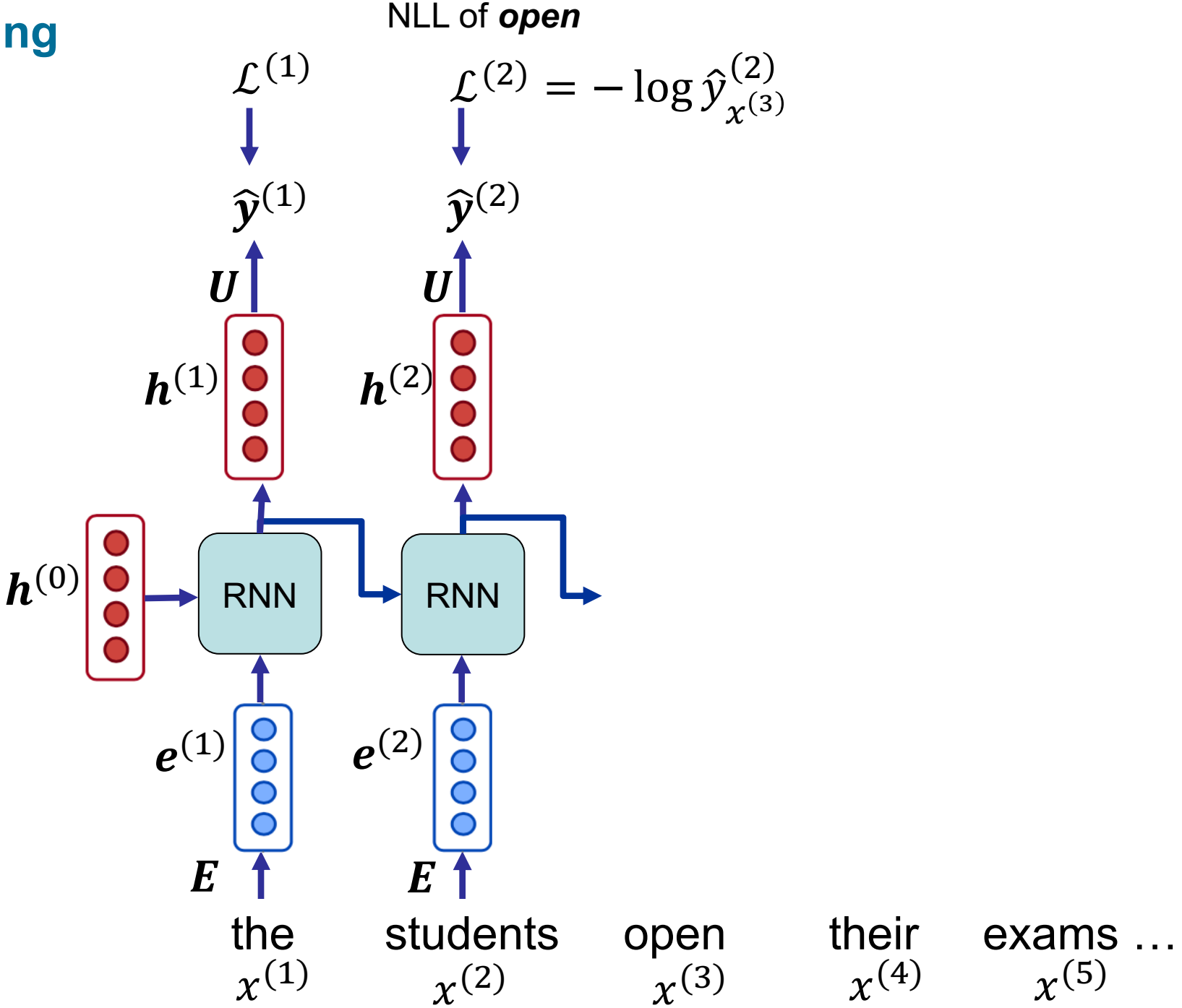
NLL of ***students***

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

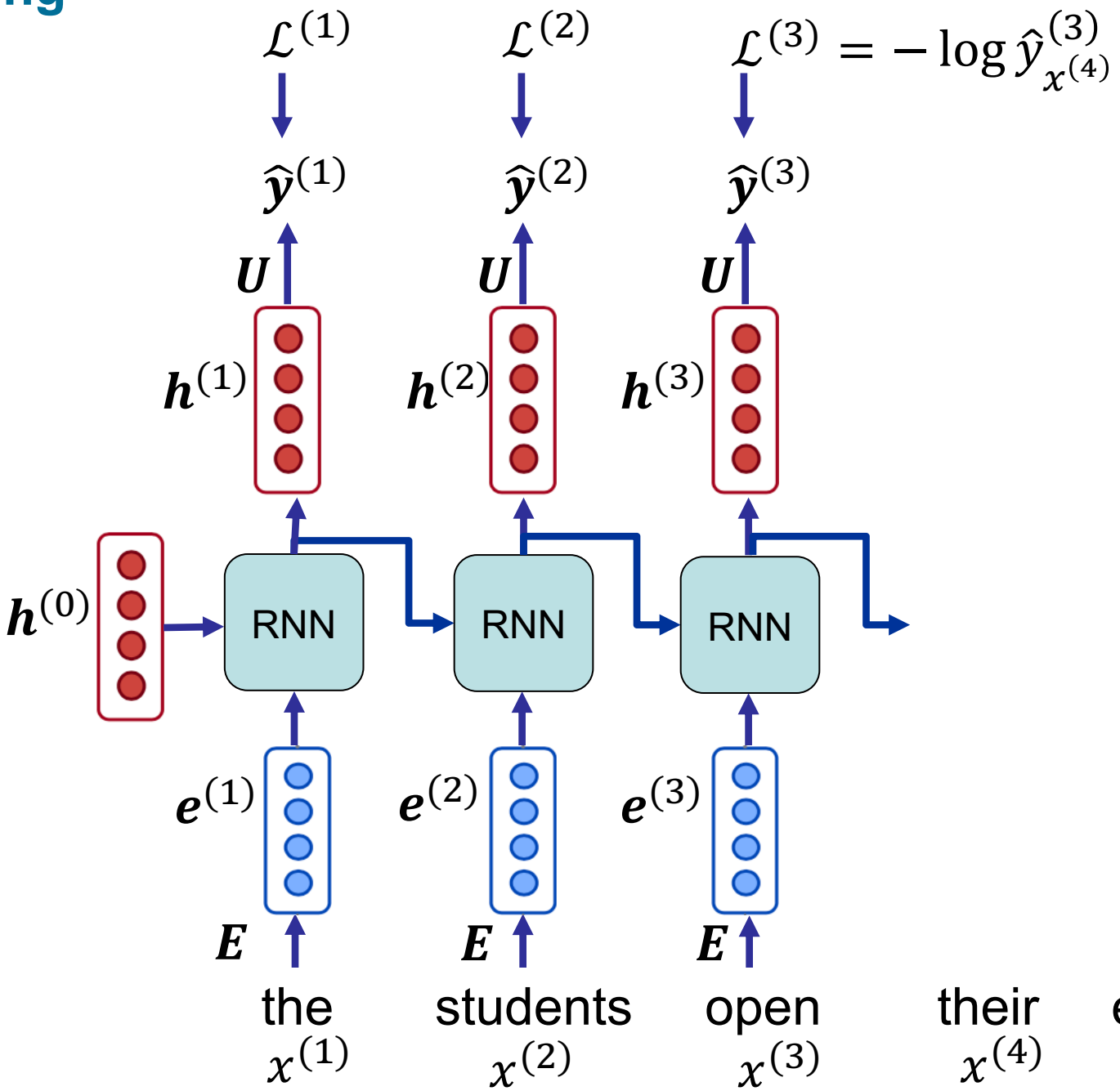


the $x^{(1)}$ students $x^{(2)}$ open $x^{(3)}$ their $x^{(4)}$ exams $x^{(5)}$ …

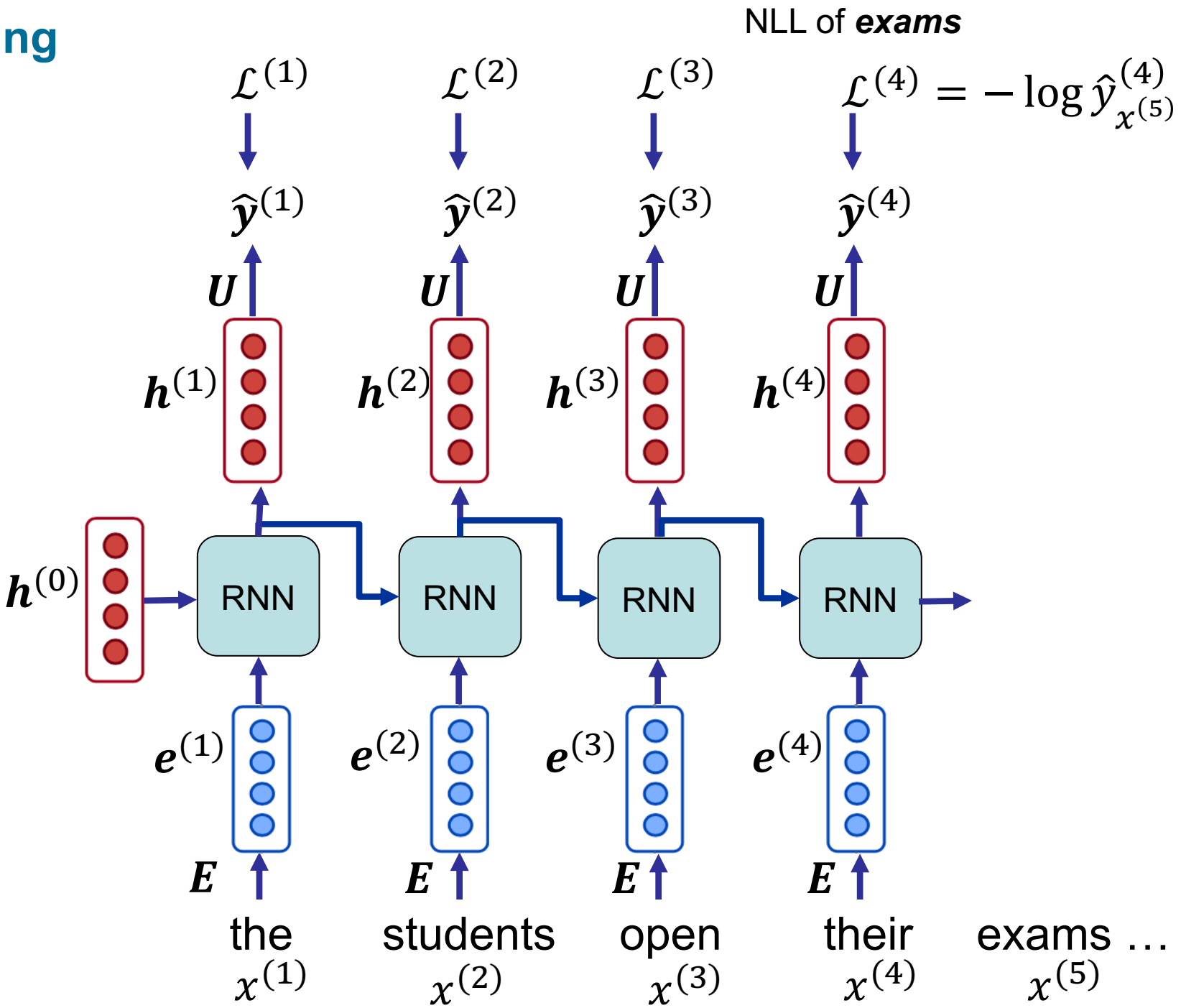
Training


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

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

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

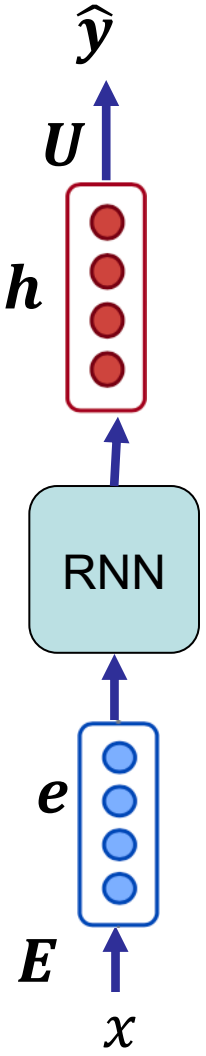
Training RNN Language Model – Parameters

- 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



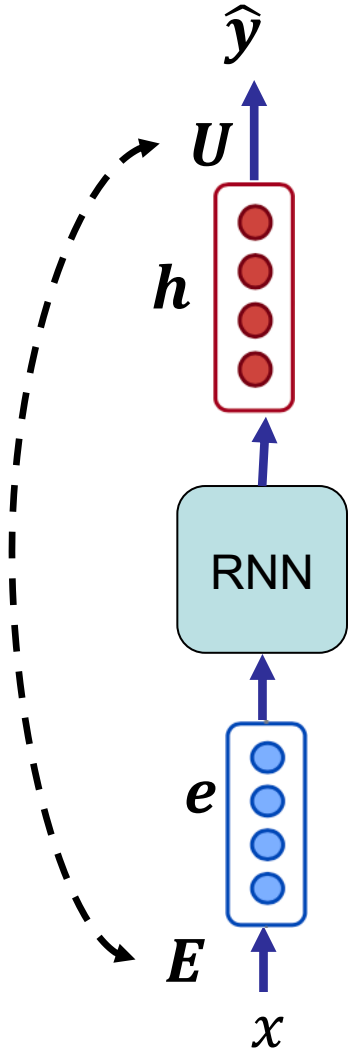
Training RNN Language Model – Weight Tying

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

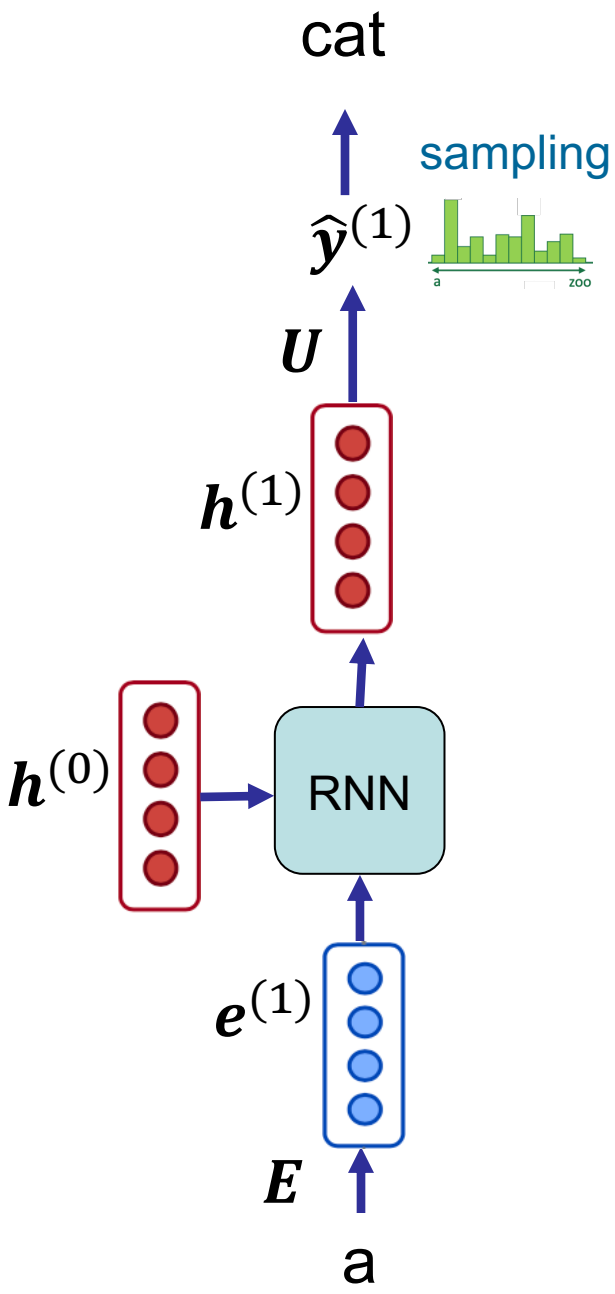
d: dimension of input embedding

h: dimension of hidden vectors

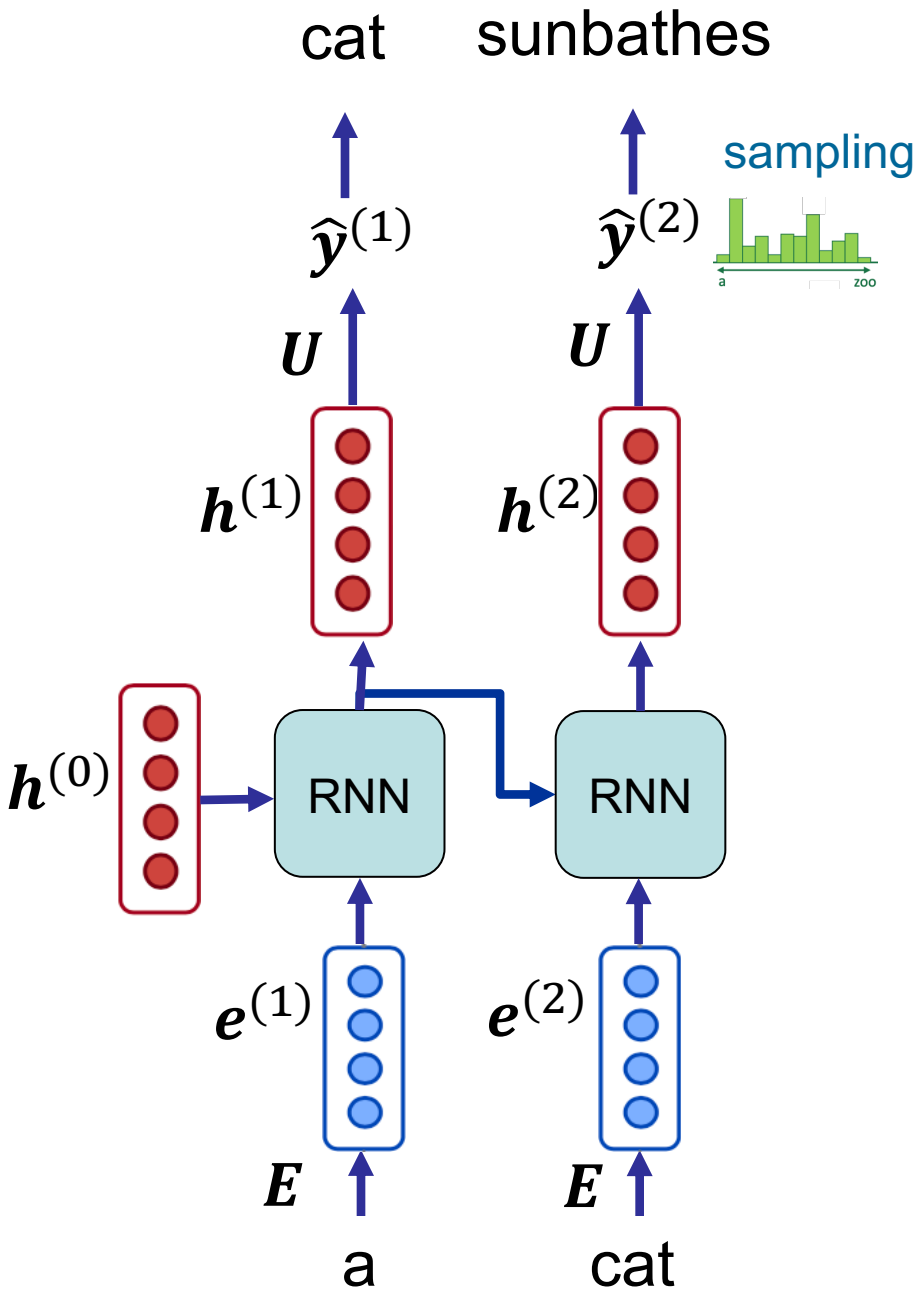
- Weight tying: set the decoder parameters the same as encoder parameters (saving $N \times h$ decoding parameters)
  - In this case d must be equal to h
  - If $d \neq h$, usually a linear projects the output vector from h to d dimensions



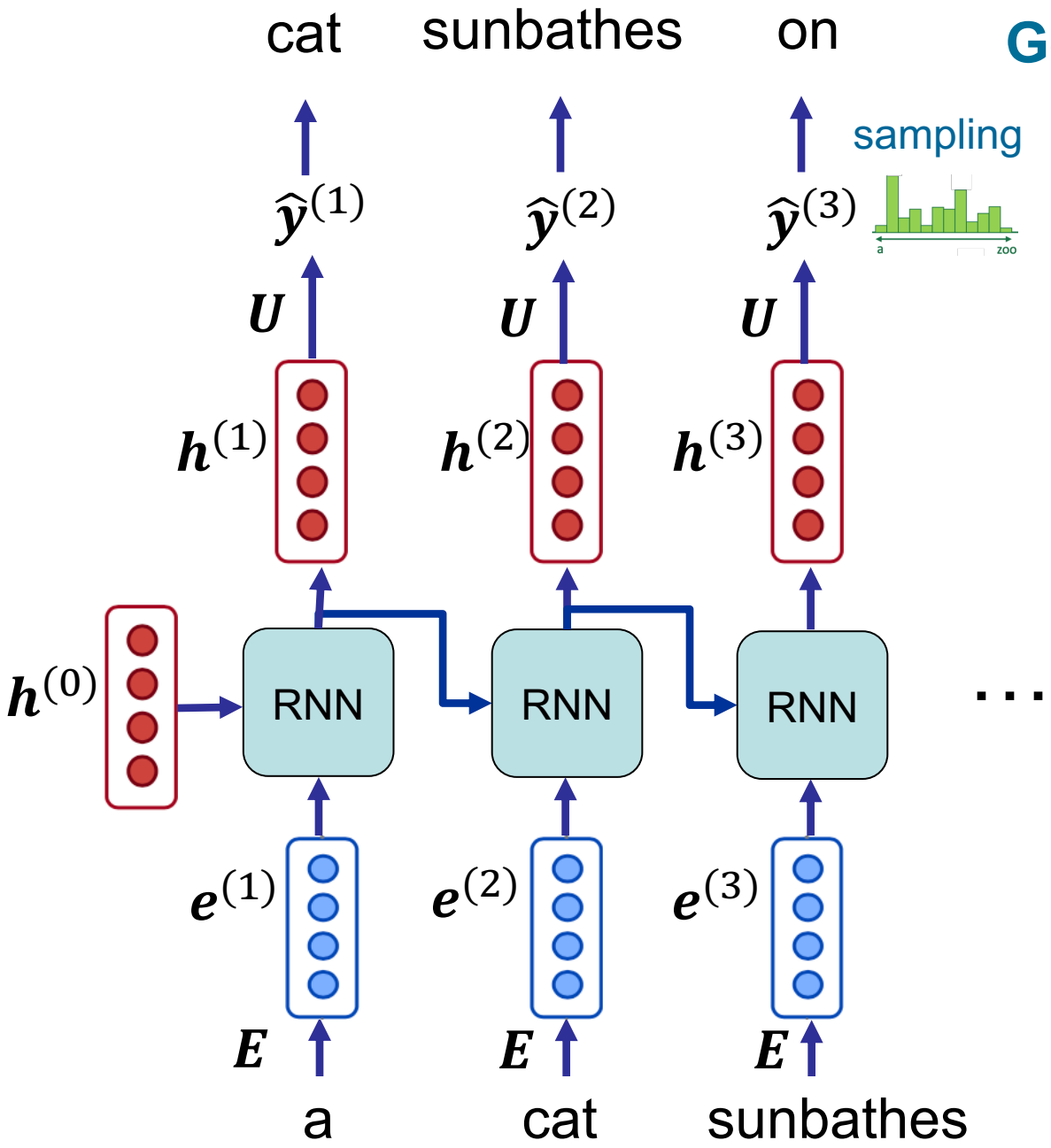
Generating text



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

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)