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

- Language Modeling with *n*-grams
- Recurrent Neural Networks
- Language Modeling with RNN
- Backpropagation Through Time

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

Agenda

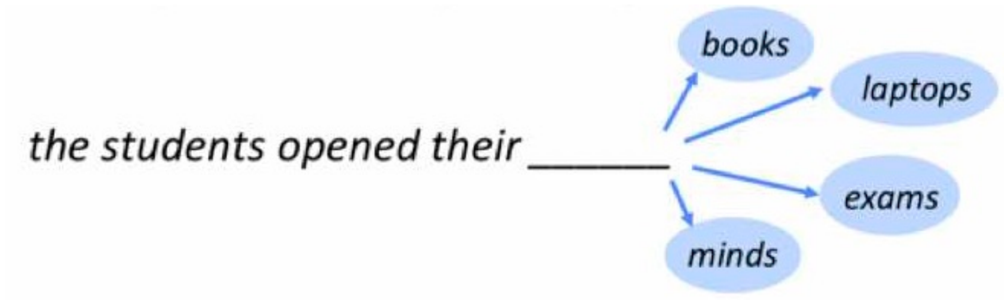
- **Language Modeling with *n*-grams**
- Recurrent Neural Networks
- Language Modeling with RNN
- Backpropagation Through Time

Language Modeling

- Language Modeling is the task of predicting a word (or a subword or character) given a context:

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

- A Language Model can answer the questions like



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

Language Modeling

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

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text
  - How probable is it that someone generates this sentence?!

    “*colorless green ideas sleep furiously*”

- According to a Language Model, the probability of a given text is computed by:

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

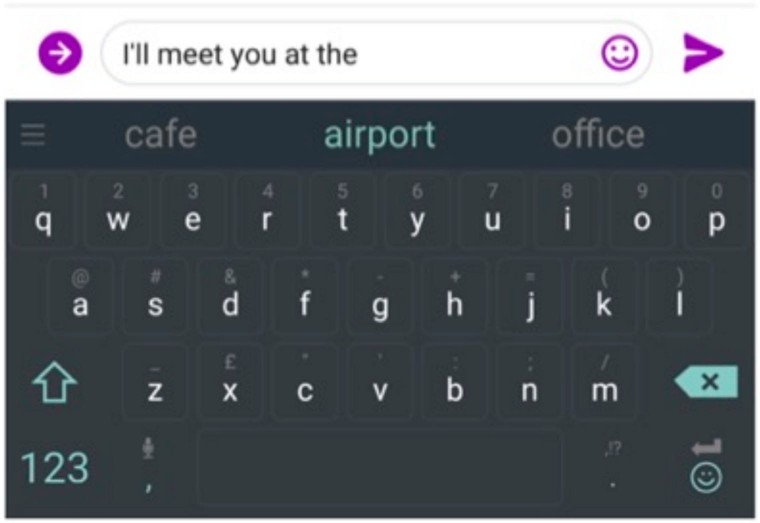
$$P\left(x^{(1)}, \dots, x^{(T)}\right) = \prod_{t=1}^{T} P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)})$$

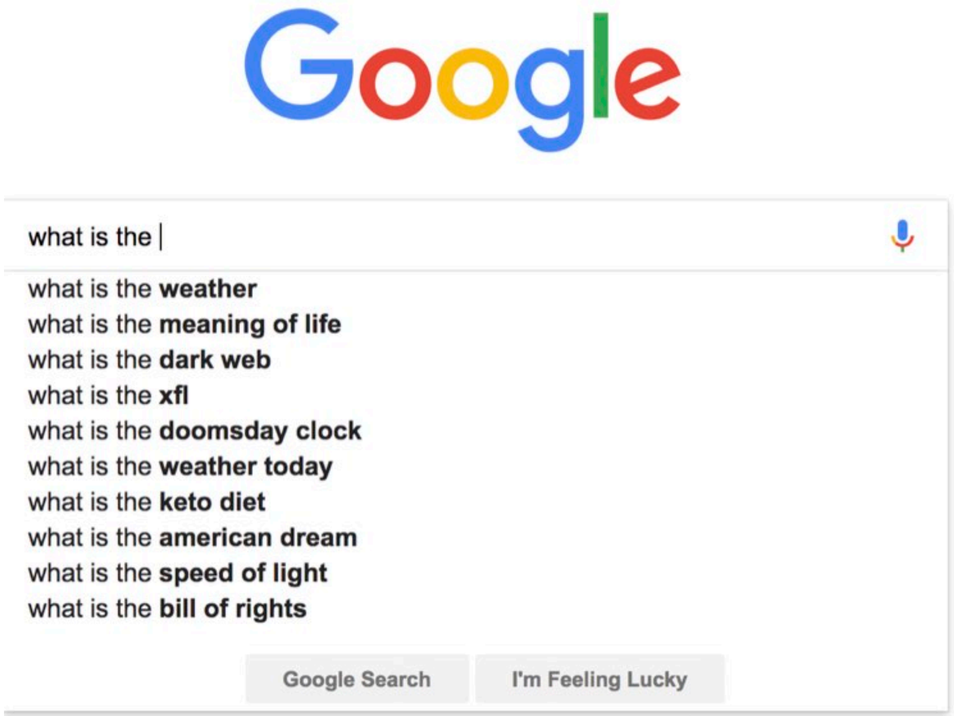
Why Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Spelling/grammar correction
  - Speech recognition
  - Handwriting recognition
  - Machine translation
  - Summarization
  - Dialogue /chatbots
  - etc.

Language Modeling

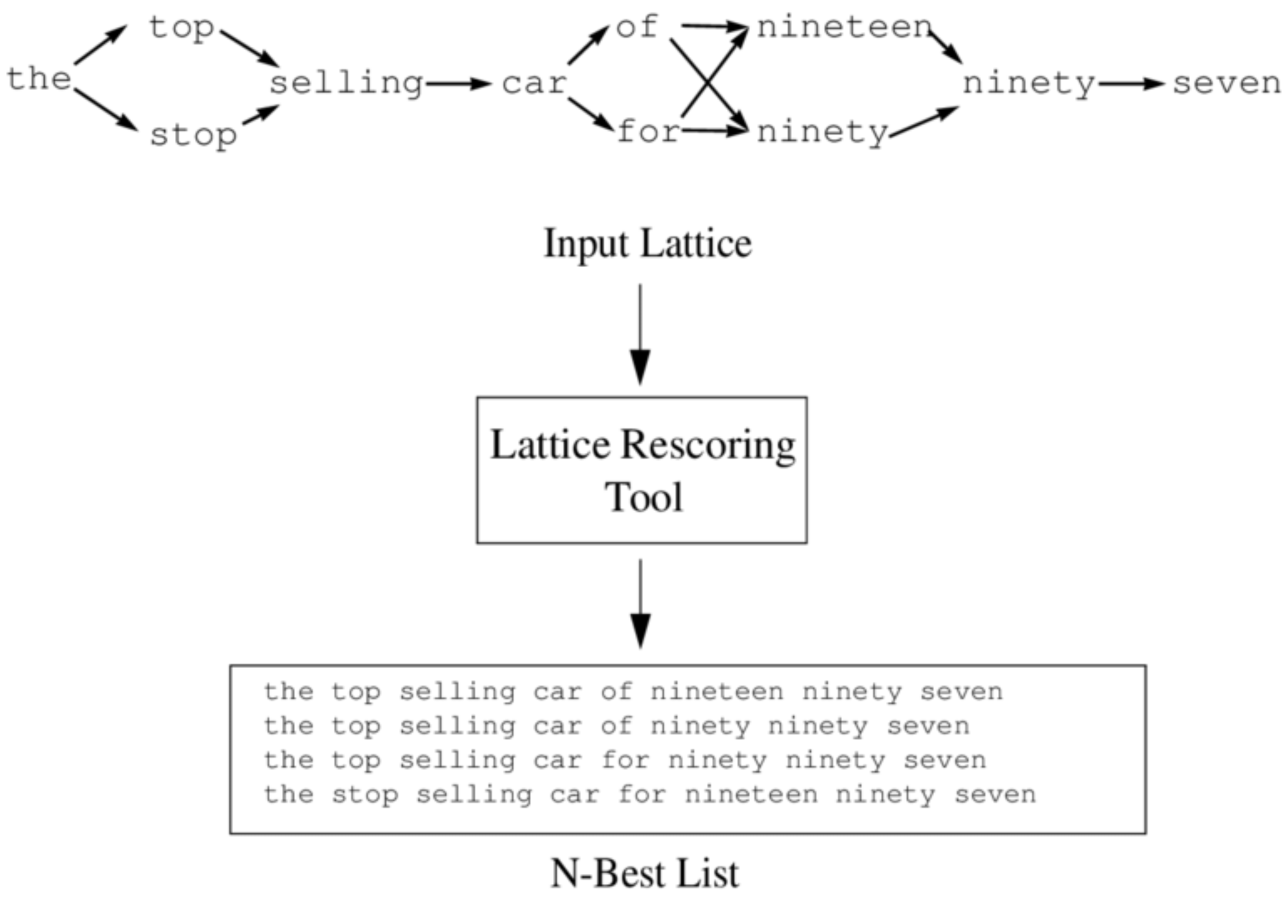



what is the |

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

[link]

Language Modeling



```
the top selling car of nineteen ninety seven
the top selling car of ninety ninety seven
the top selling car for ninety ninety seven
the stop selling car for nineteen ninety seven
```

n-gram Language Model

- 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”

- A *n*-gram Language Model collects frequency statistics of different *n*-grams in a corpus, and use these to calculate probabilities

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:

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

- *n*-gram Language Model:

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

n-gram Language Model

- Based on definition of conditional probability:

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

- The *n*-gram probability is calculated by counting *n*-grams and [*n*–*1*]-grams in a large corpus of text:

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

n-gram Language Model

- Example: learning a 4-gram Language Model

~~as the exam clerk started the clock, the~~ $\underbrace{\textit{students opened their}}_{\text{condition on this}}$ _______

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

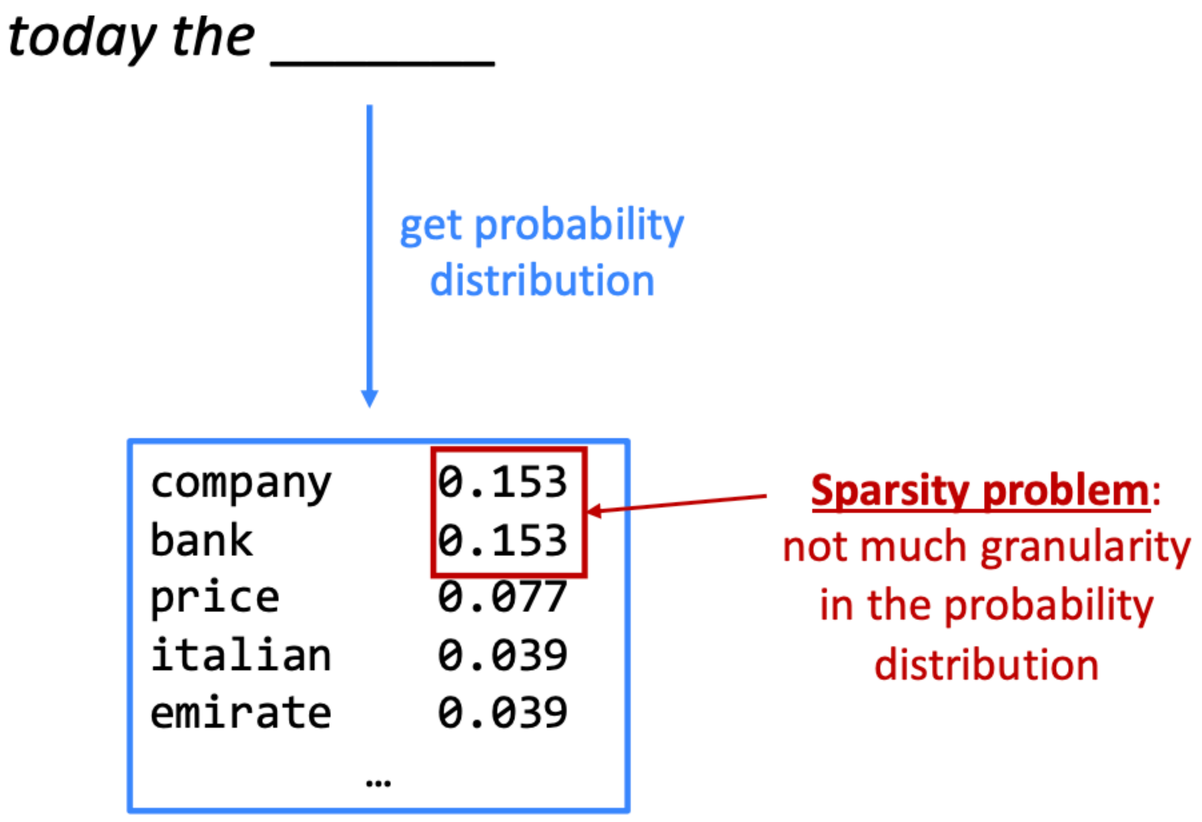
- For example, suppose that in the corpus:
  - "*students opened their*" occurred 1000 times
  - "*students opened their books*" occurred 400 times
    - P(*books* | *students opened their*) = 0.4
  - "*students opened their exams*" occurred 100 times
    - P(*exams* | *students opened their*) = 0.1

n-gram Language Model – problems

- Sparsity
  - If nominator „*students opened their* v" never occurred in corpus
    - Smoothing: add small hyper-parameter δ to all words
  - If denominator „*students opened their*" never occurred in corpus
    - Backoff: condition on "*students opened*" instead
  - Increasing *n* makes sparsity problem worse!
- Storage
  - The model needs to store all *n*-grams (from unigram to *n*-gram), observed in the corpus
  - Increasing *n* worsens the storage problem radically!

n-gram Language Models – generating text

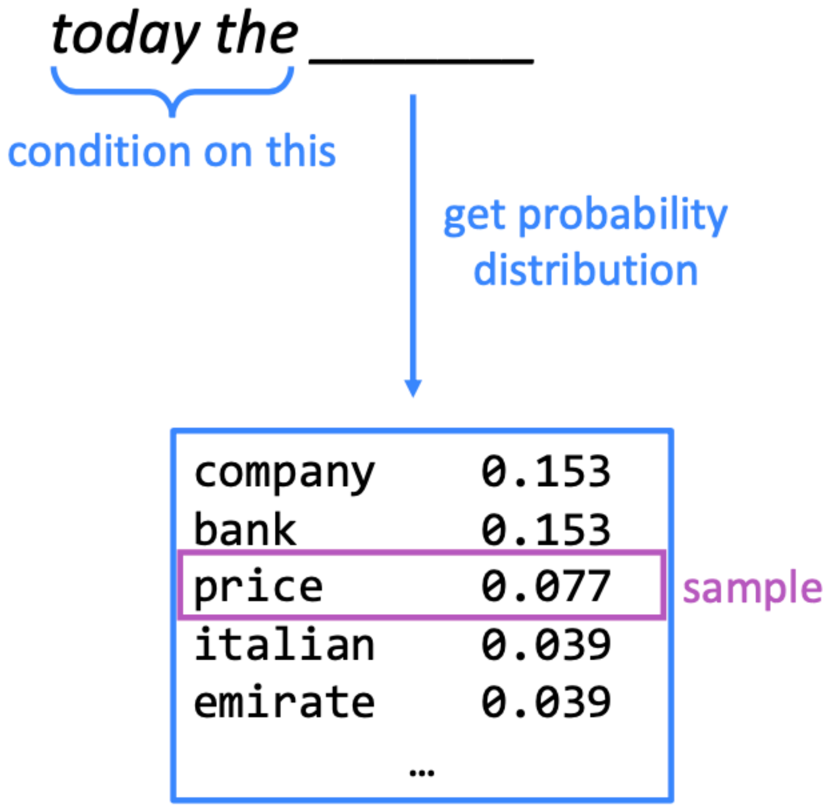
- A trigram Language Model trained on Reuters corpus (1.7 M words)


today the ________
get probability distribution
Sparsity problem: not much granularity in the probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

n-gram Language Models – generating text

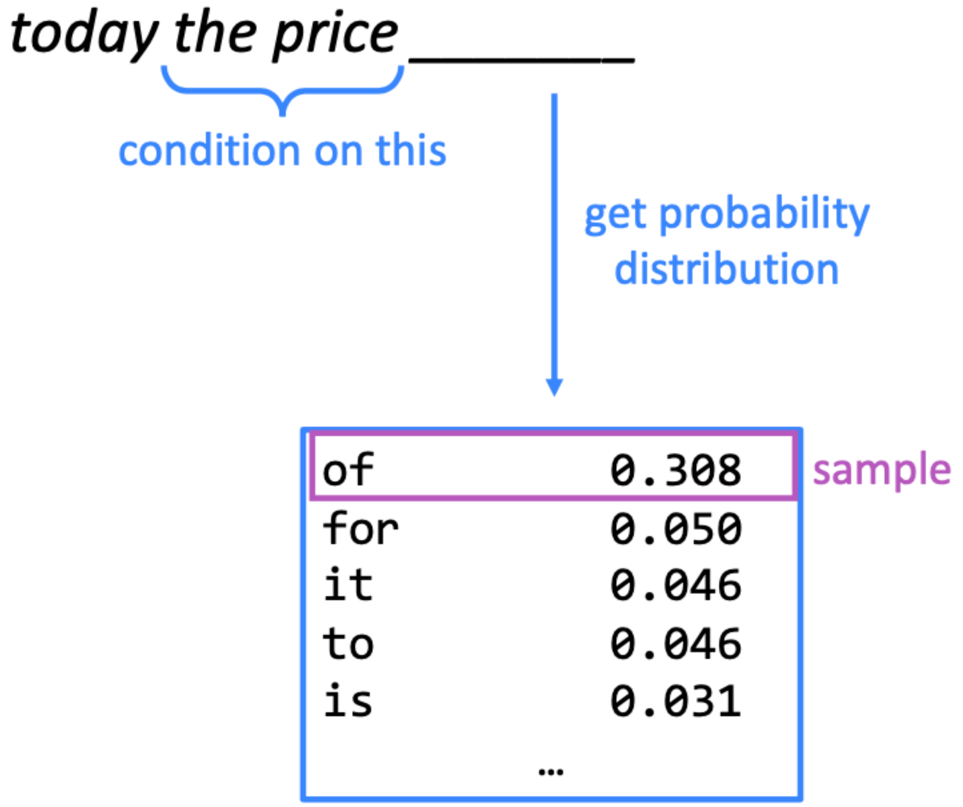
- Generating text by sampling from the probability distributions



company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

n-gram Language Models – generating text

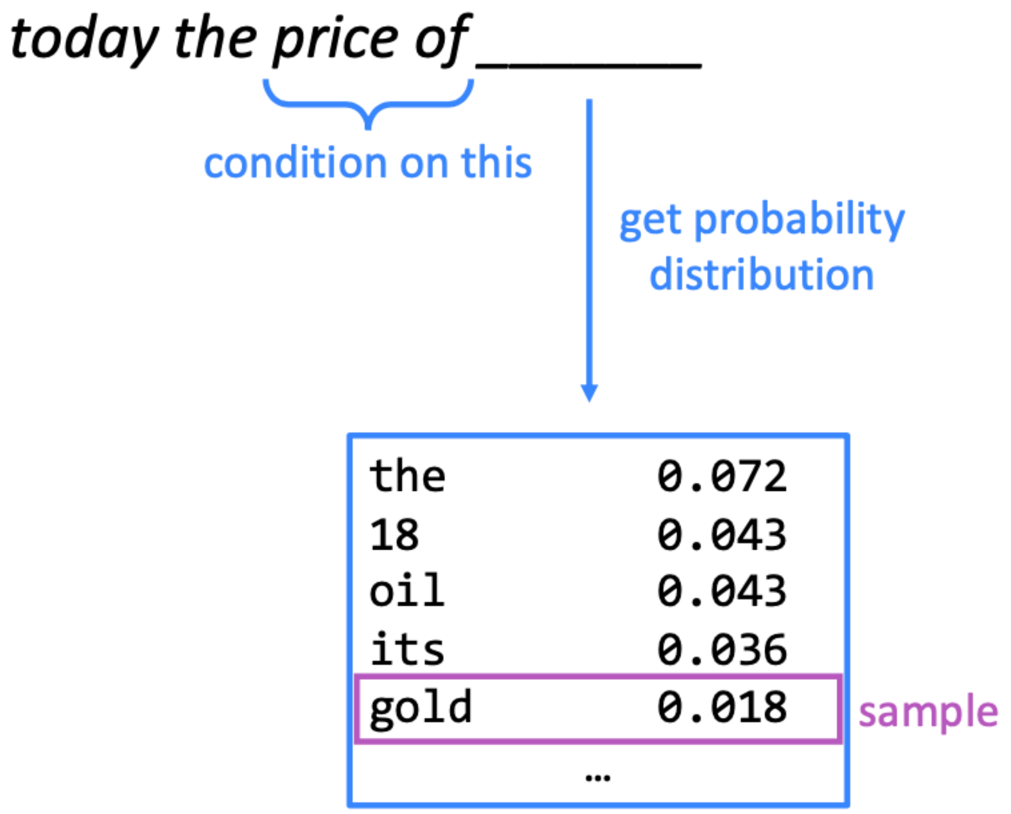
- Generating text by sampling from the probability distributions



of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

n-gram Language Models – generating text

- Generating text by sampling from the probability distributions



the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

n-gram Language Models – generating text

- Generating text by sampling from the probability distributions

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

- Very good in syntax … but incoherent!
- Increasing *n* makes the text more coherent but also intensifies the discussed issues

Agenda

- Language Modeling with *n*-grams
- **Recurrent Neural Networks**
- Training Language Models with RNN
- Backpropagation Through Time

Recurrent Neural Network

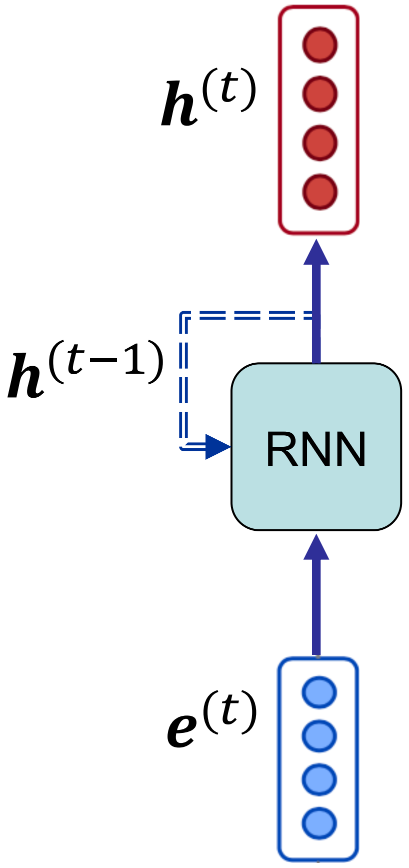
- Recurrent Neural Network (RNN) encodes/embeds a sequential input of any size like …
  - Sequence of word/subword/character vectors
  - Time series

… into compositional embeddings

- RNN captures dependencies through the sequence by applying the same parameters repeatedly …

- RNN outputs a final embedding but also intermediary embeddings on each time step

Recurrent Neural Networks

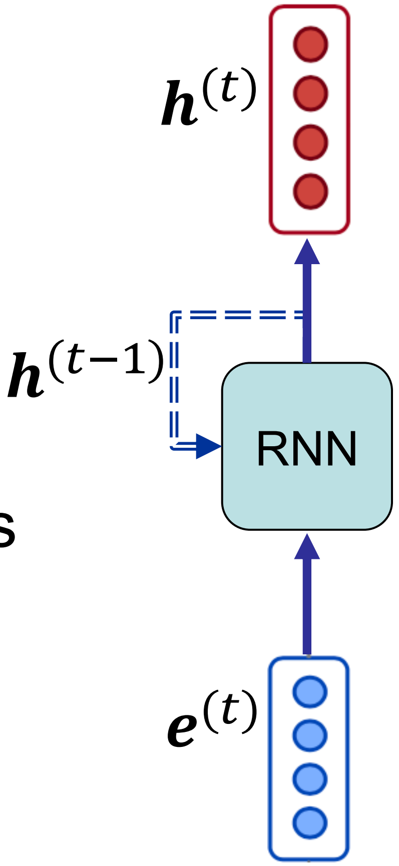


Recurrent Neural Networks

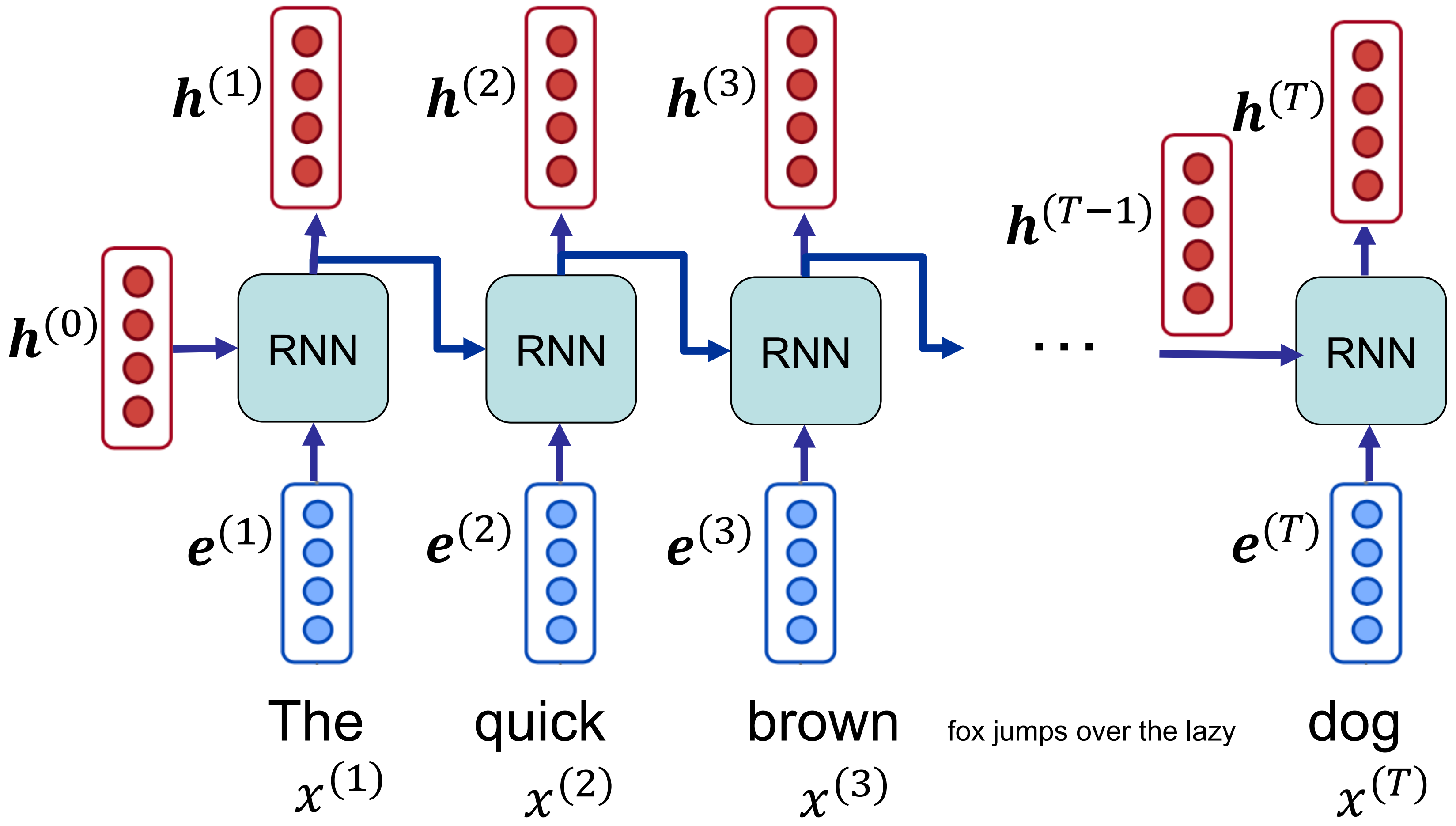
- Output $\boldsymbol{h}^{(t)}$ is a function of input $\boldsymbol{e}^{(t)}$ and the output of the previous time step $\boldsymbol{h}^{(t-1)}$

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

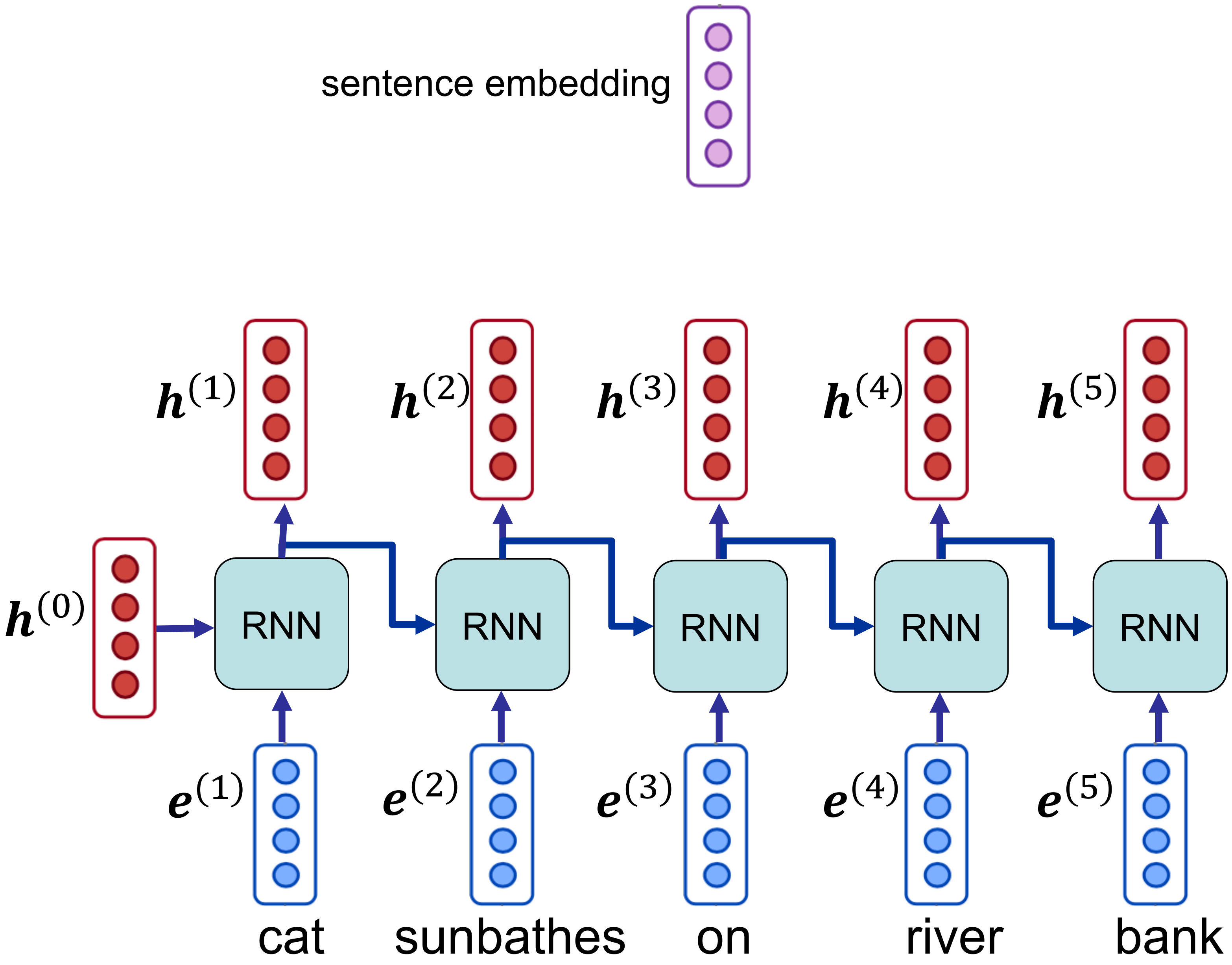
- $\boldsymbol{h}^{(t)}$ is called hidden state
- With hidden state $\boldsymbol{h}^{(t-1)}$, the model accesses to a sort of memory from all previous entities



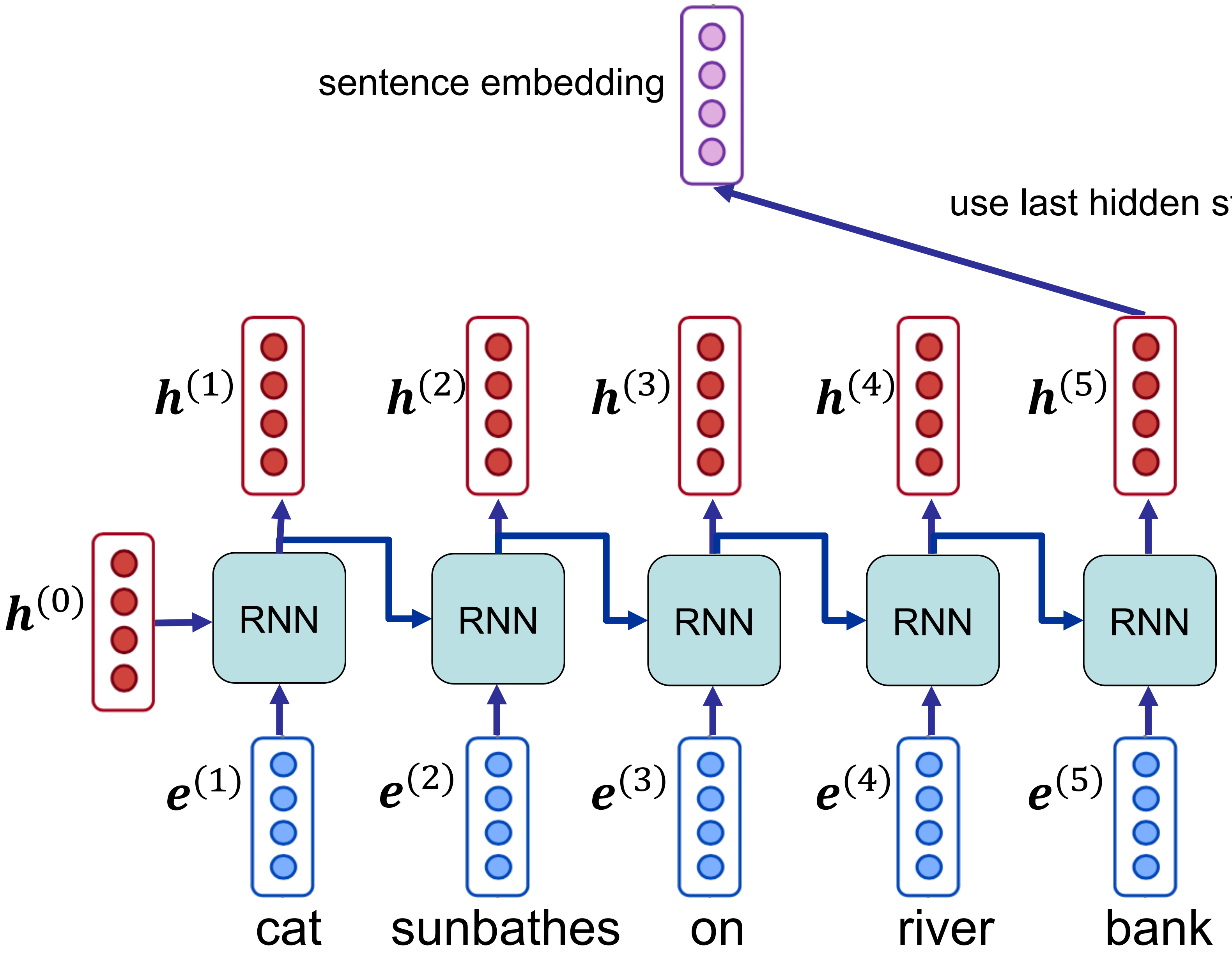
RNN – Unrolling



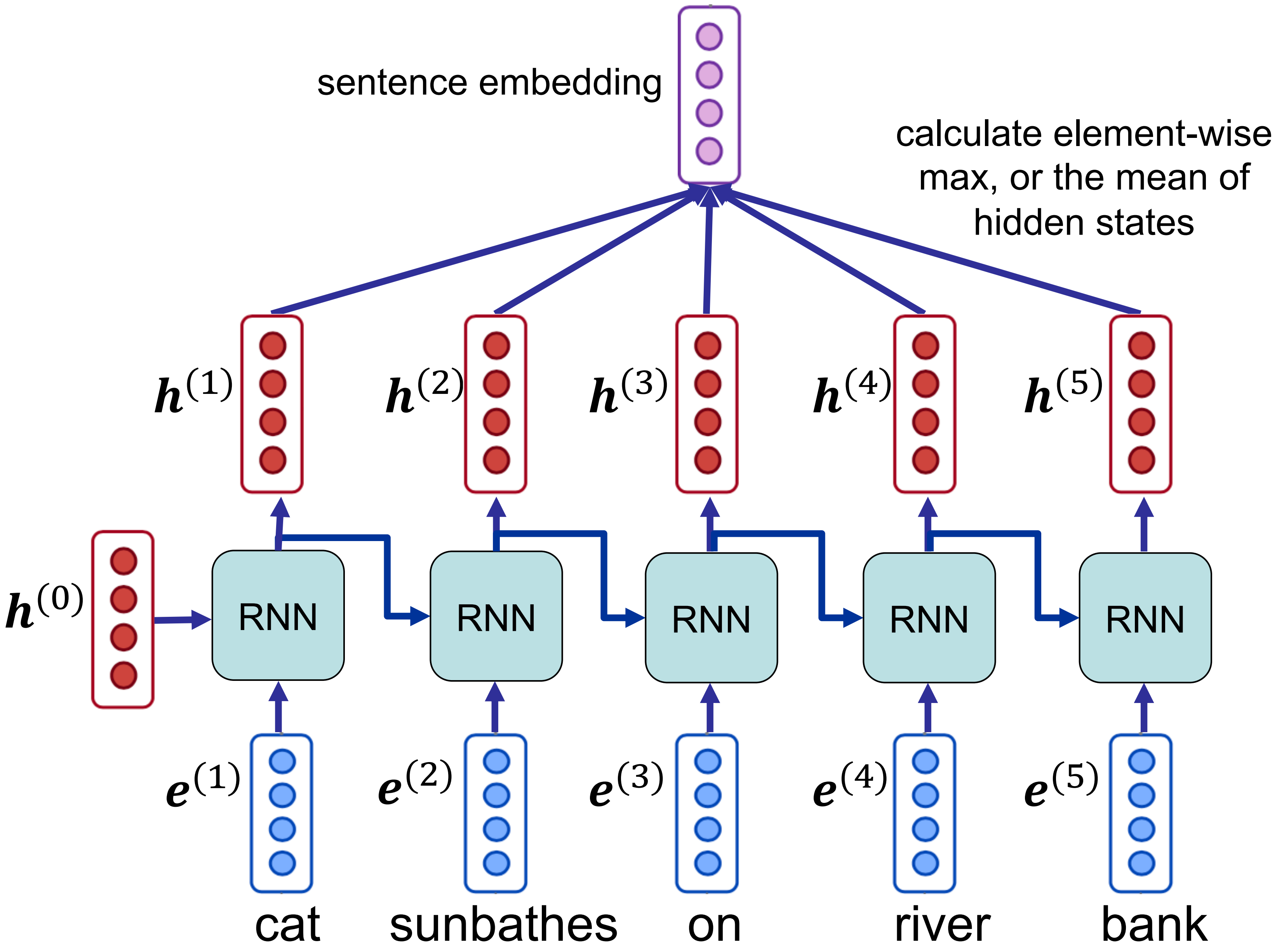
RNN – Compositional embedding



RNN – Compositional embedding



RNN – Compositional embedding



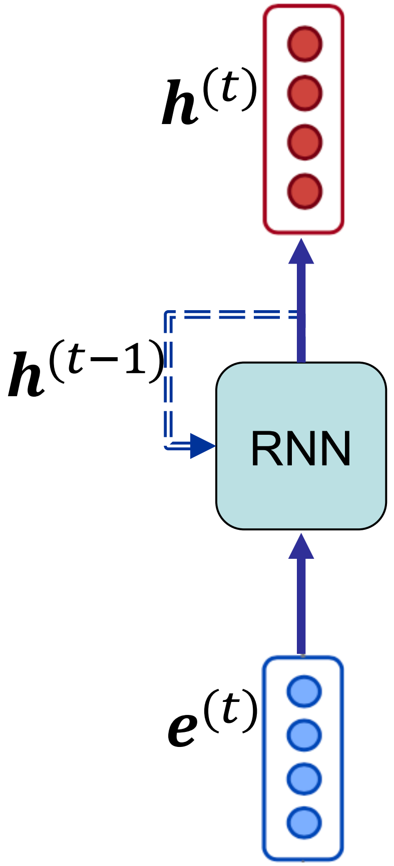
Standard (Elman) RNN

- 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

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

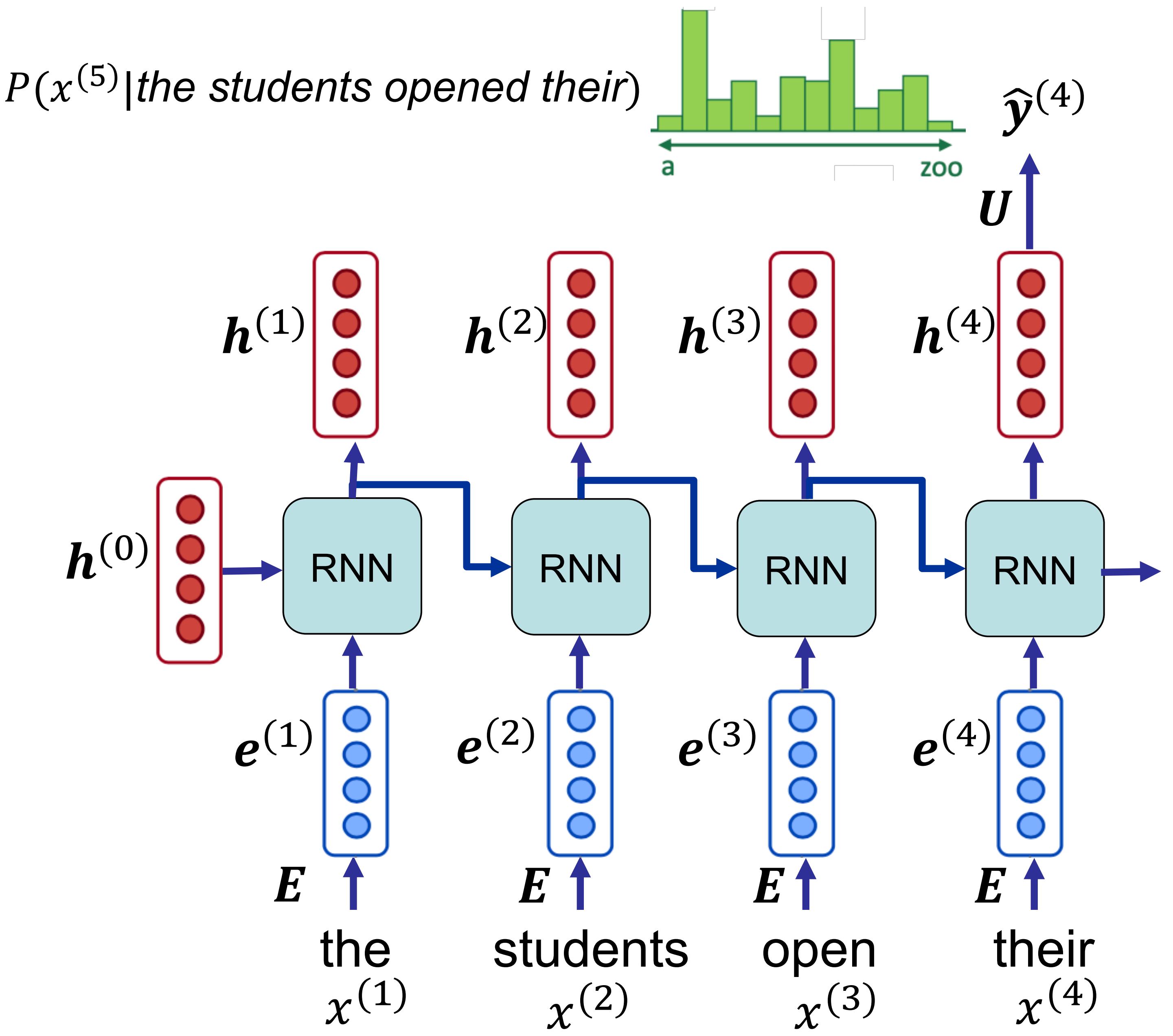


Agenda

- Language Modeling with *n*-grams
- Recurrent Neural Networks
- **Language Modeling with RNN**
- Backpropagation Through Time

RNN Language Model

$P(x^{(5)}|\textit{the students opened their})$



RNN Language Model

- Encoder
  - word at time step $t \rightarrow x^{(t)}$
  - One-hot vector of $x^{(t)} \rightarrow \boldsymbol{x}^{(t)} \in \mathbb{R}^{|\mathbb{V}|}$
  - Word embedding $\rightarrow \boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{E}$
- RNN

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

- Decoder
  - Predicted probability distribution:

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

  - Probability of any word v at step t:

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

Training an RNN Language Model

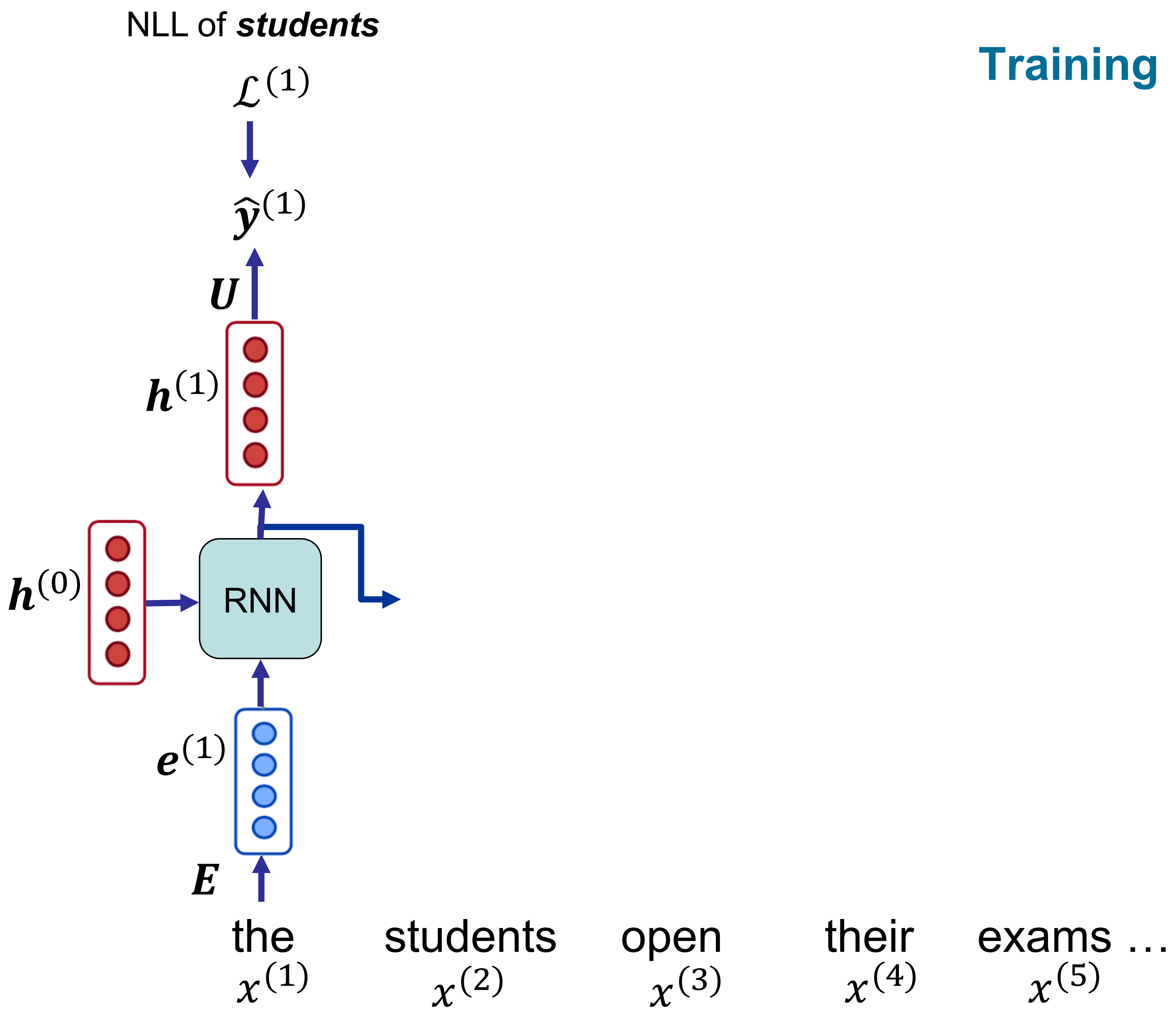
- Start with a large text corpus: $x^{(1)}, \ldots, x^{(T)}$
- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$
- Calculate the loss function: Negative Log Likelihood of the predicted probability of the true next word $x^{(t+1)}$

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

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

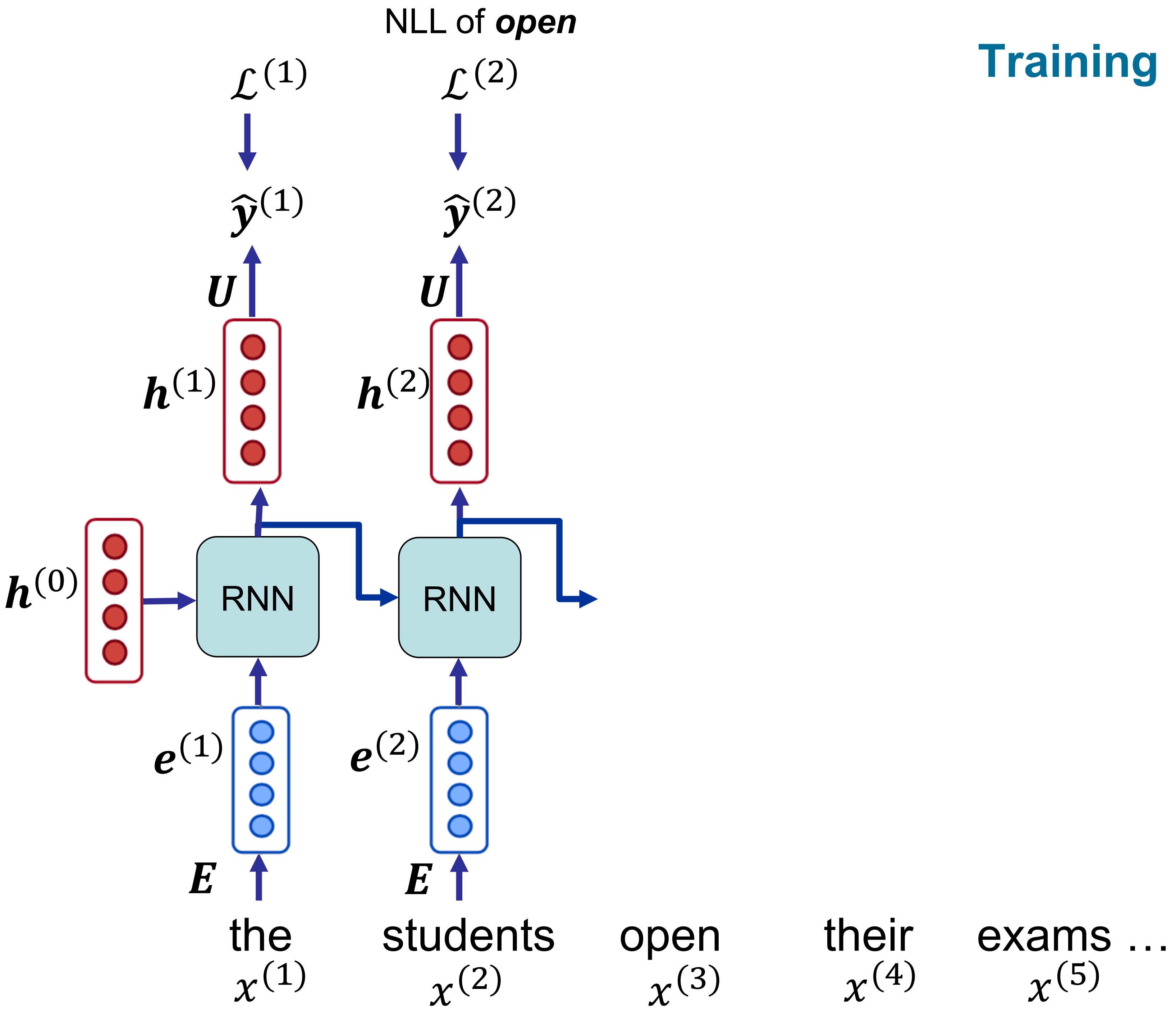
$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}^{(t)}$$

Training

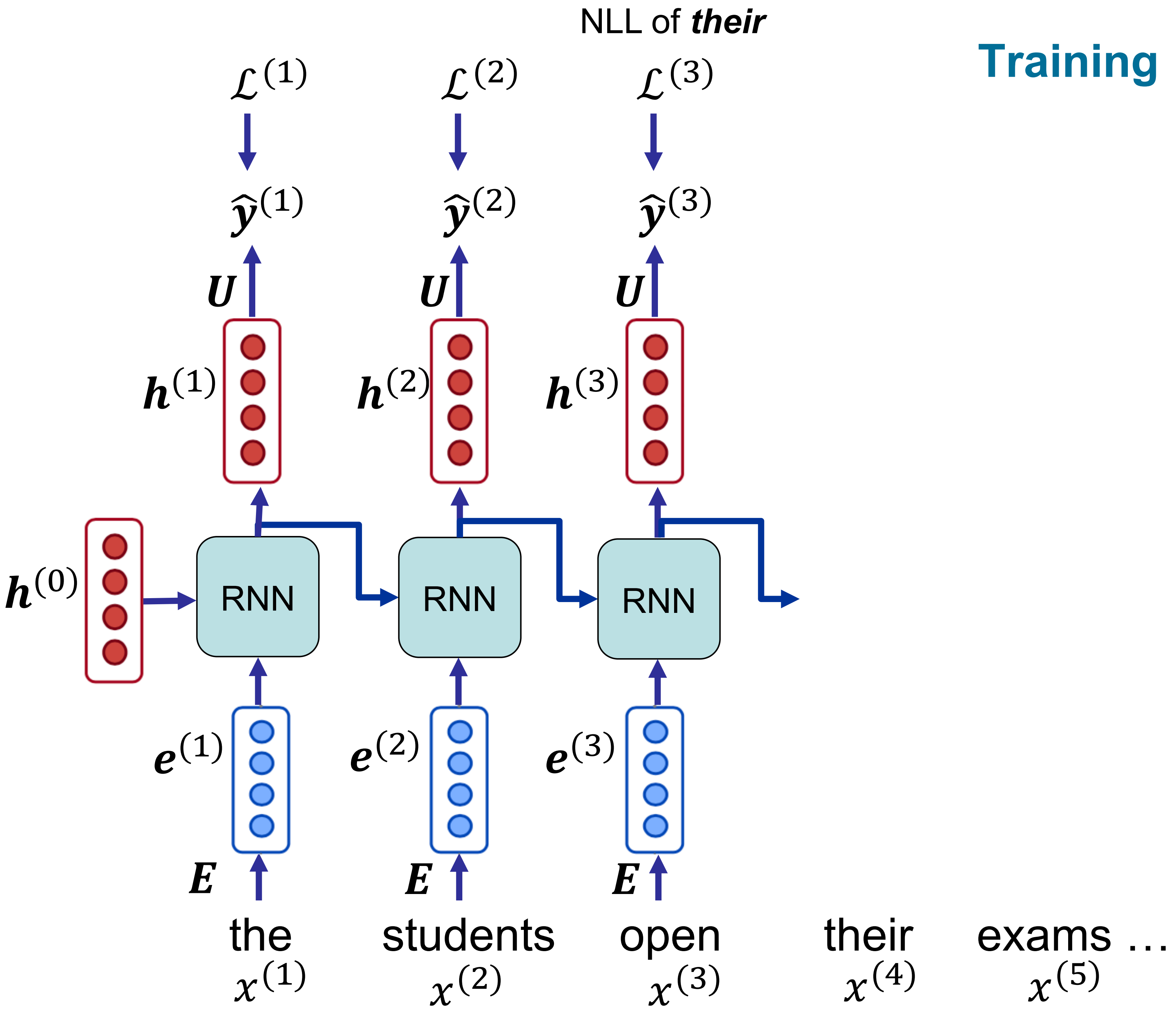


NLL of ***students***
$\mathcal{L}^{(1)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(0)}$
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{E}$
the
$x^{(1)}$
students
$x^{(2)}$
open
$x^{(3)}$
their
$x^{(4)}$
exams …
$x^{(5)}$

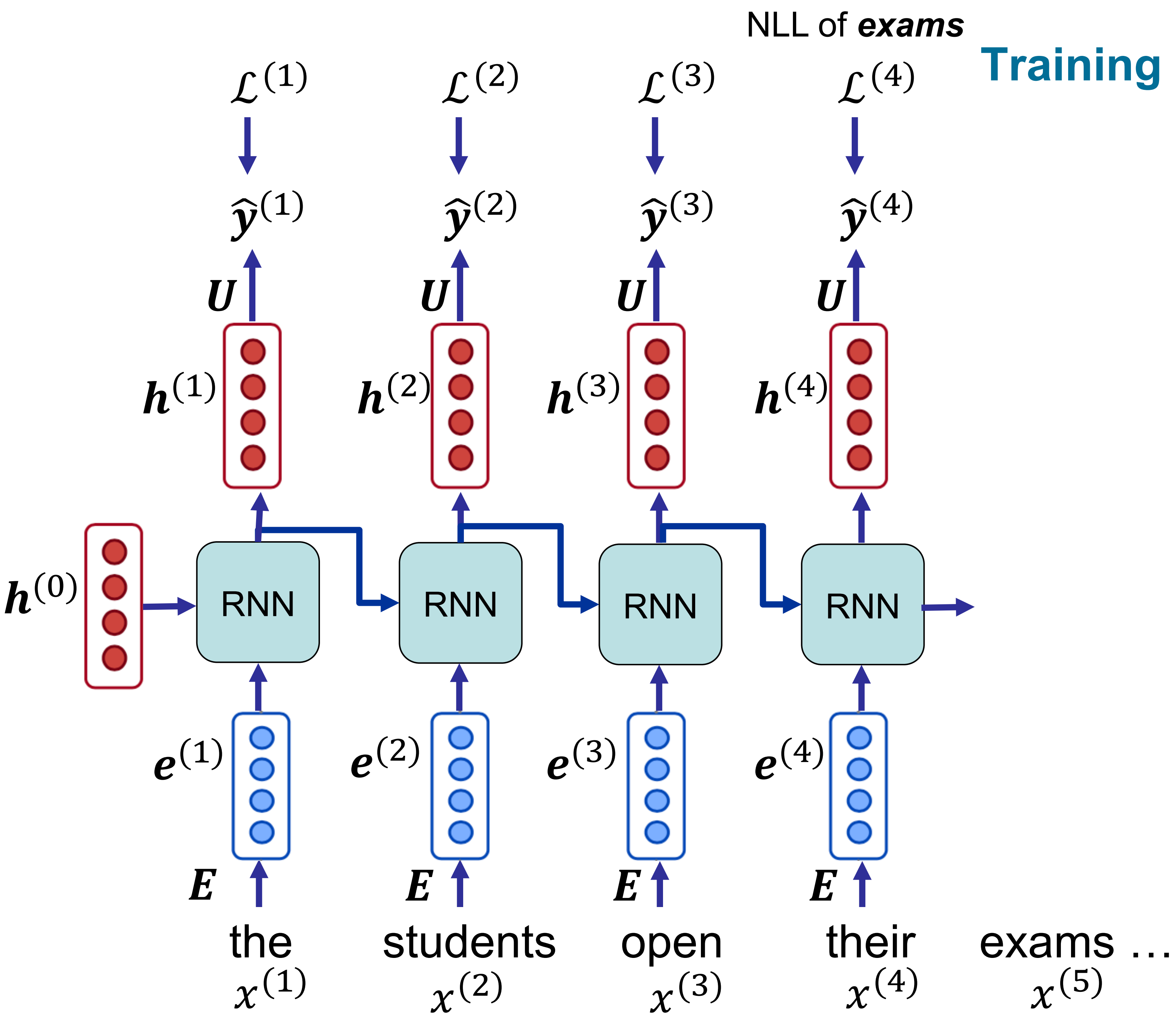
Training


NLL of ***open***
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(0)}$
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***
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(0)}$
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***
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)}$
$\mathcal{L}^{(4)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\hat{\boldsymbol{y}}^{(4)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
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 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, the corpus should be splitted into sub-corpora, where 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 is:

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

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

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

- 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

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

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

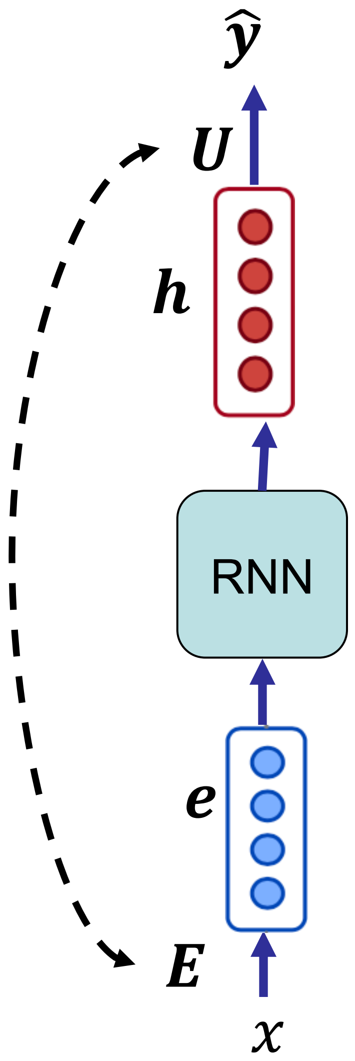
$$\begin{bmatrix} x^{(L+1)} & \ldots & x^{(2L)} \\ x^{(L+\tau+1)} & \ldots & x^{(\tau+2L)} \end{bmatrix}$$

Training RNN Language Model – Weight Tying

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

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

- Encoder and decoder embeddings contain the most parameters

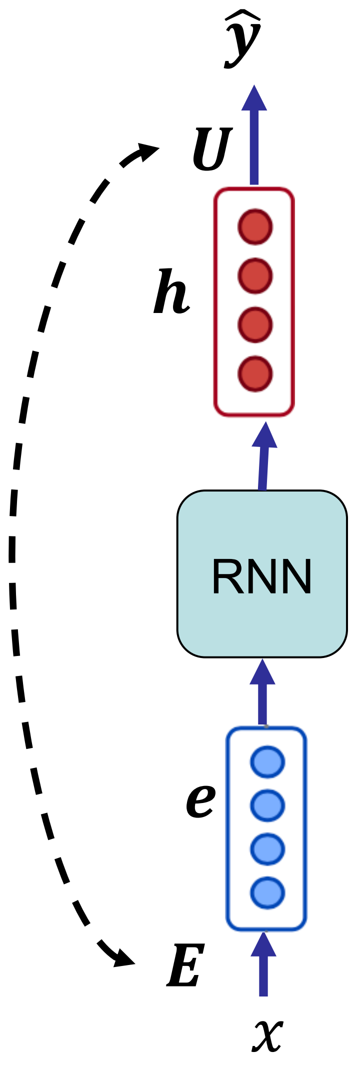


Training RNN Language Model – Weight Tying

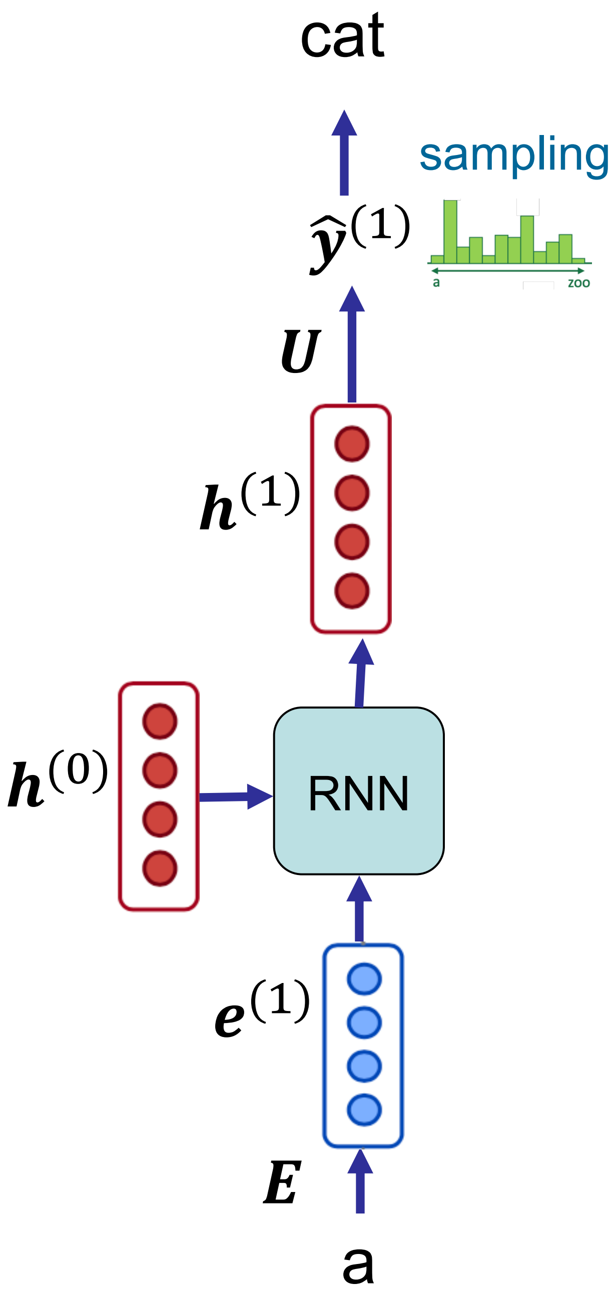
- Parameters in a vanilla RNN (bias terms discarded)
  - $\boldsymbol{E} \rightarrow |\mathbb{V}|\times h$
  - $\boldsymbol{U} \rightarrow h\times|\mathbb{V}|$
  - $\boldsymbol{W}_i \rightarrow d\times h$
  - $\boldsymbol{W}_h \rightarrow h\times h$

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

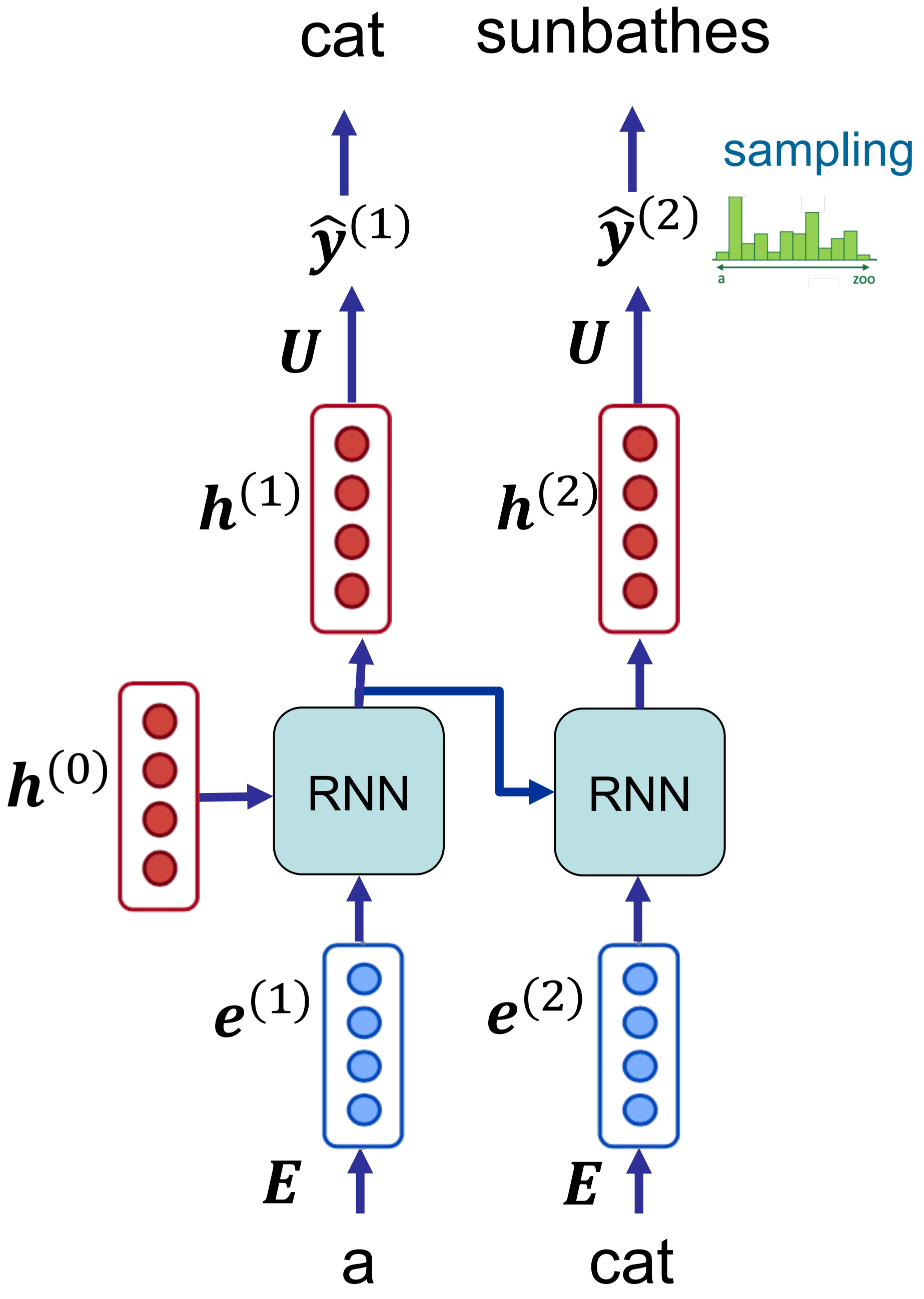
- Weight tying: set the decoder parameters the same as encoder parameters (saving $|\mathbb{V}|\times h$ decoding parameters)
  - In this case d must be equal to h
  - If $d \neq h$, usually a linear projects the output vector from h to d dimensions



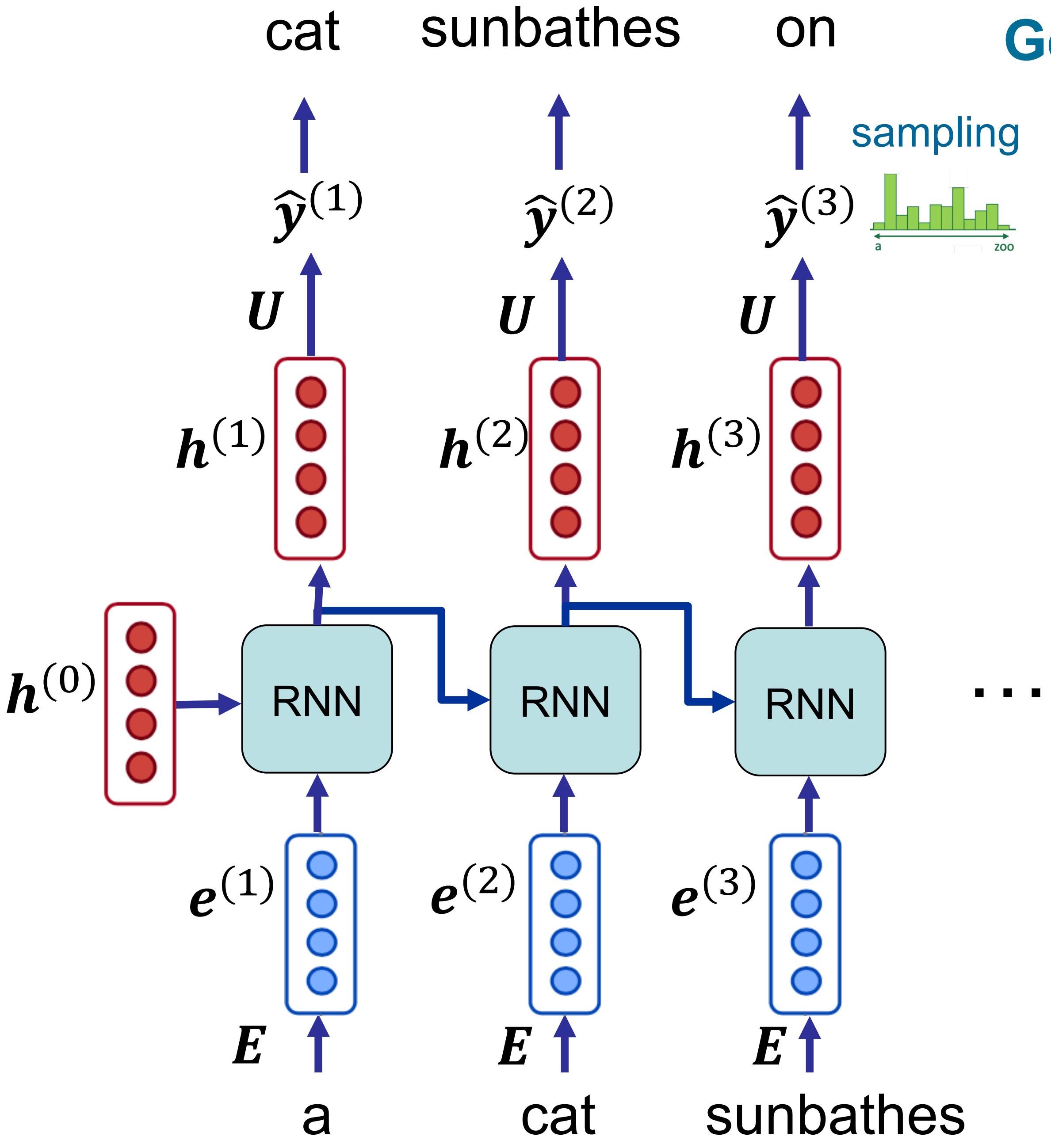
Generating text



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.

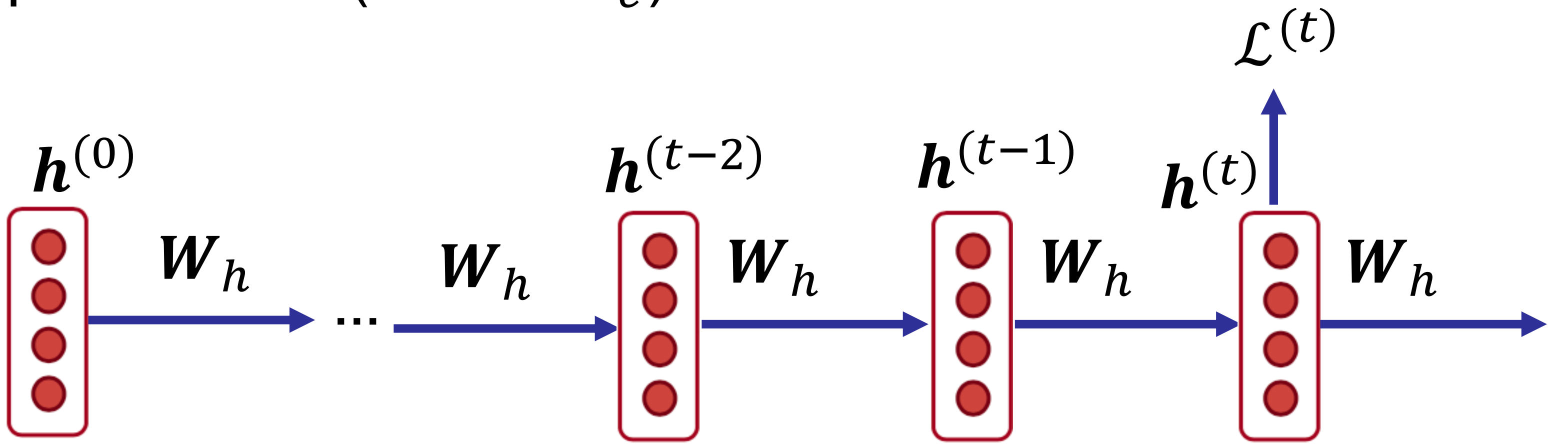
https://github.com/ppramesi/RoboTrumpDNN

Agenda

- Language Modeling with *n*-grams
- Recurrent Neural Networks
- Language Modeling with RNN
- **Backpropagation Through Time**

Backpropagation for RNNs

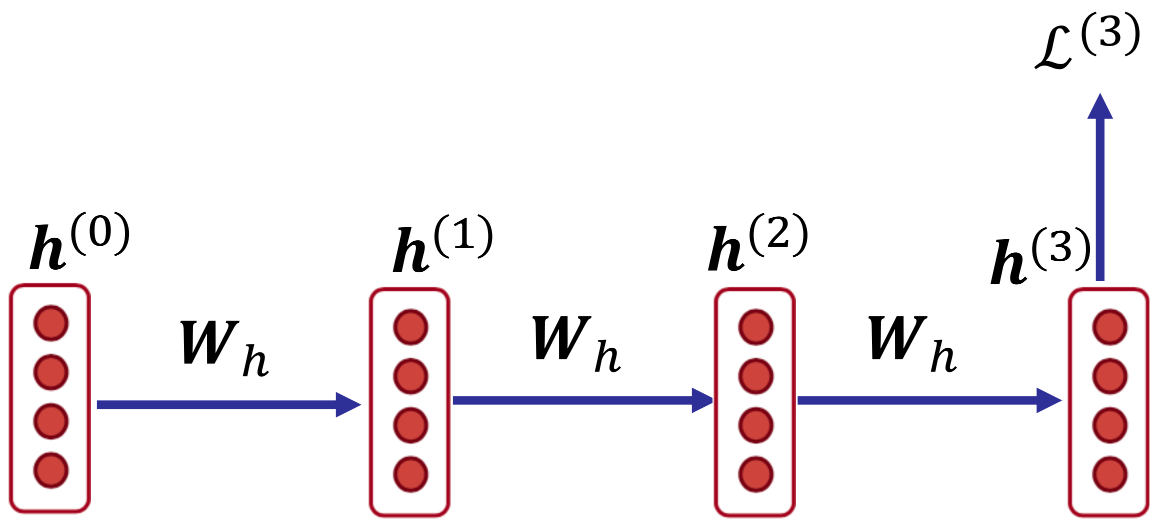
- Unrolling the computation graph of RNN
- Simplified: the interactions with $\boldsymbol{U}$ and also input parameters ($\boldsymbol{E}$ and $\boldsymbol{W}_e$) are removed



- What is …

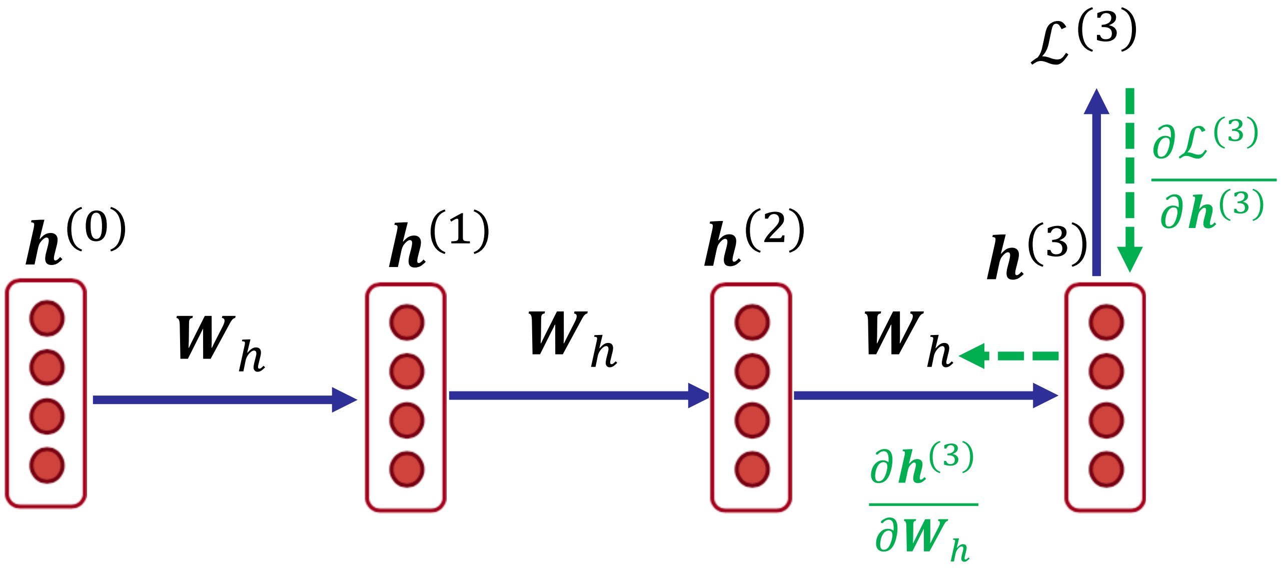
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = ?$$

Backpropagation for RNNs



$$\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h} = ?$$

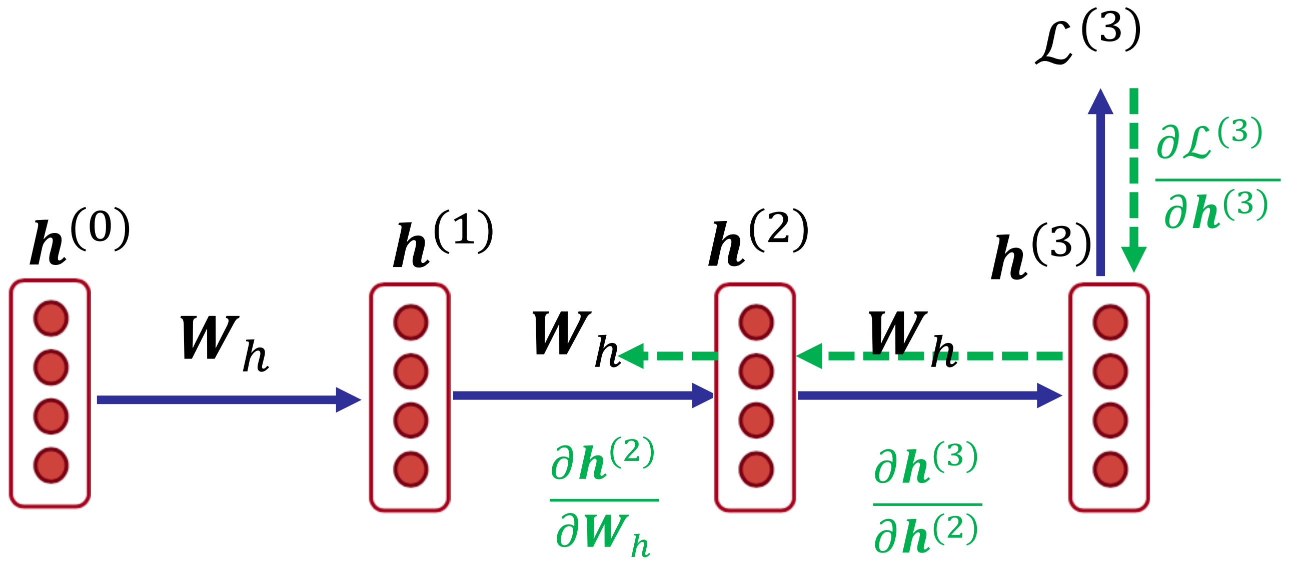
Backpropagation for RNNs



$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(3)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 3

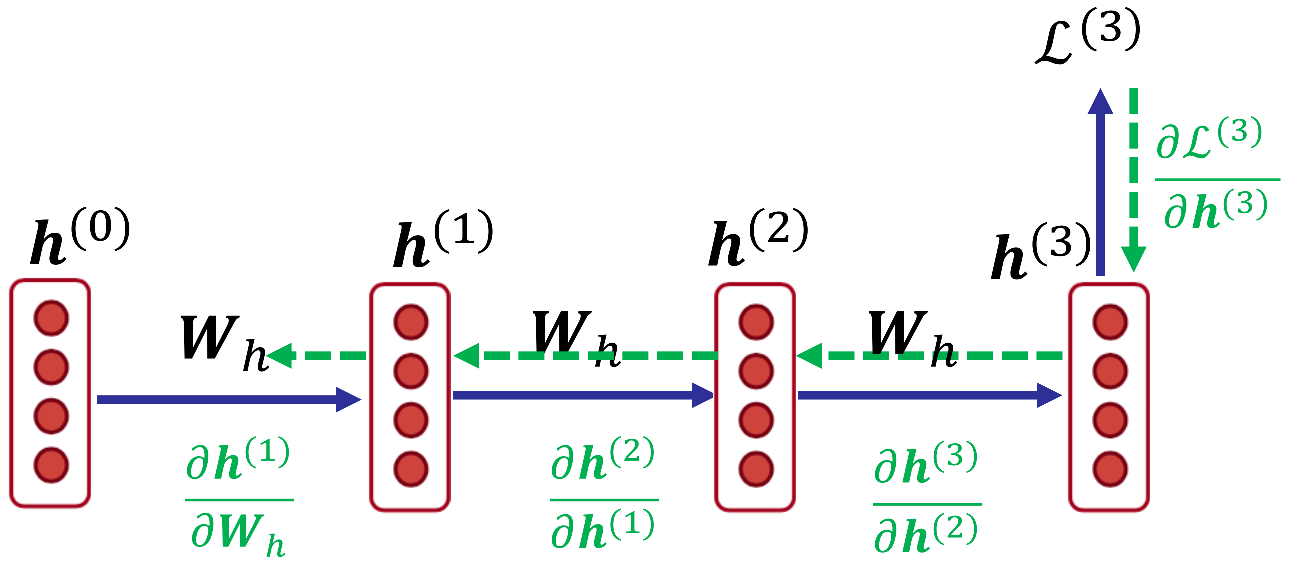
Backpropagation for RNNs


$\mathcal{L}^{(3)}$
$\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$
$\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}$

$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(2)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 2

Backpropagation for RNNs


$\mathcal{L}^{(3)}$
$\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}$
$\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}$

$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(1)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 1

Backpropagation Through Time

- Final gradient is the sum of the gradients regarding the model parameters (such as $\boldsymbol{W}_h$) from the current time step back to the beginning of corpus (or batch)

$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left. \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} \right|_{(i)}$$

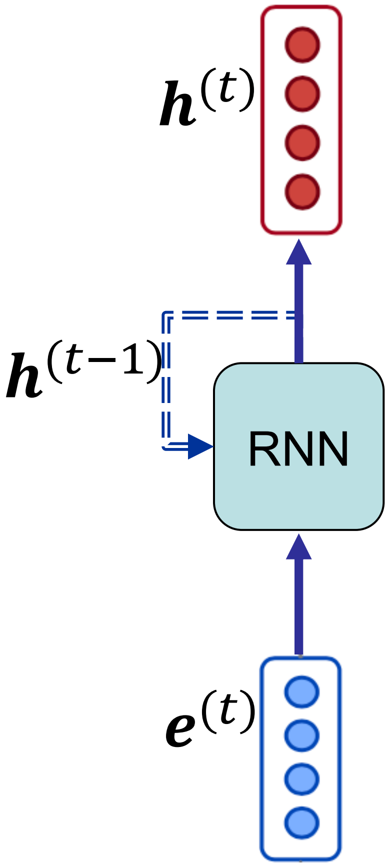
- In this simplified case, this can be written as:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

Summary

- A Language Model calculates …
  - the probability of the appearance of a word/subword/character given its context
  - the probability of a stream of text

- Recurrent Neural Network
  - Predicts next entity (word/subword/character) based on a *memory* of previous steps and the given input
  - Backpropagation Through Time in each step t updates model parameters t times, from the current point back to the beginning of the sequence



Summary

<u>RNN for Language Modeling</u>

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

- **Cons**:
  - Recurrent computation is slow → (in its vanilla form) does not fully exploit the parallel computation capability of GPUs
  - In practice, difficult to access information from many steps back