344.063/163 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

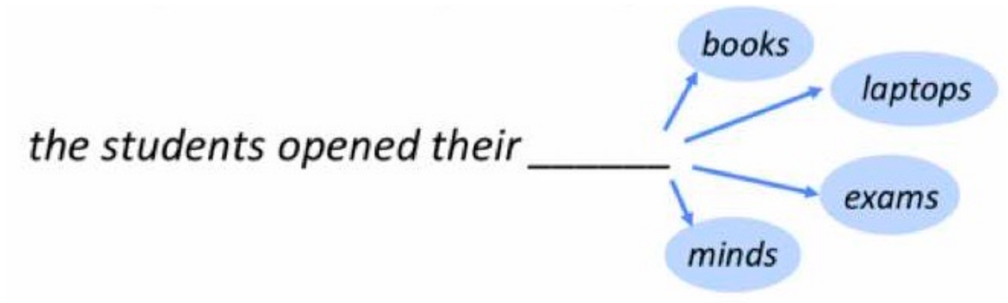
- Recurrent Neural Networks
- Language Modeling with RNN
- Backpropagation Through Time

Language Modeling

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

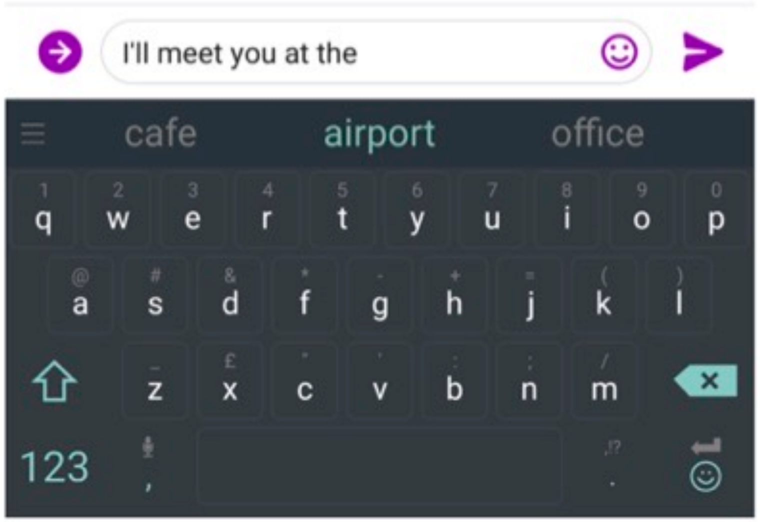
$$P(v|\text{context})$$

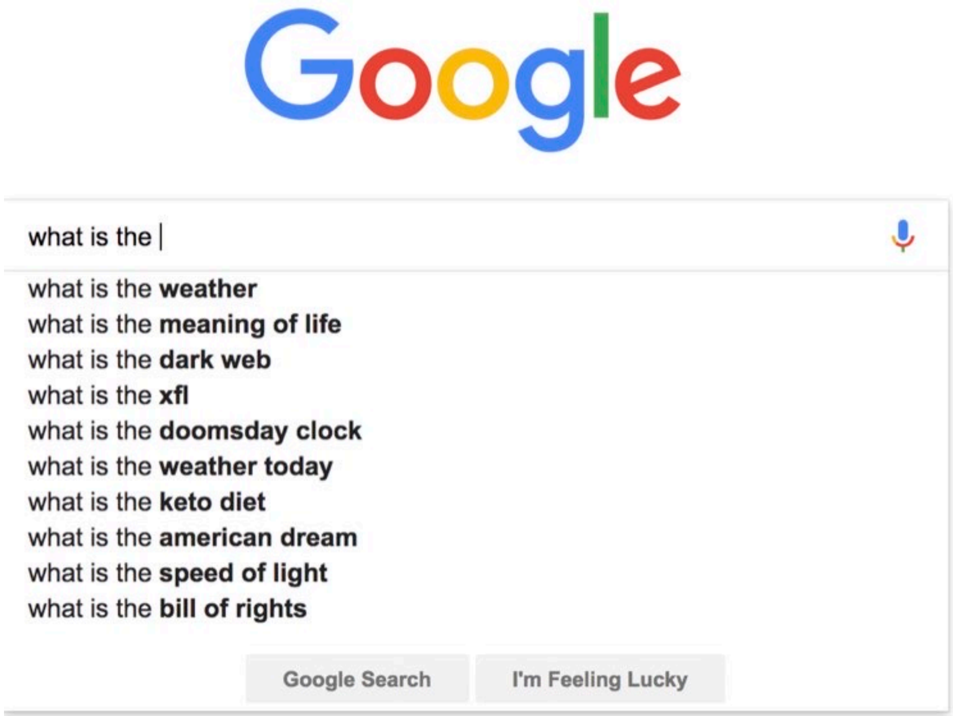
- A Language Model can answer the questions like



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

Language Modeling


I'll meet you at the


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**

Recap from 344.075 Natural Language Processing

Why Language Modeling?

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

- Language Models 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.

Recap from 344.075 Natural Language Processing

Linear Algebra – Dot product

- $\boldsymbol{a} \cdot \boldsymbol{b}^T = c$
  - dimensions: $1 \times d \cdot d \times 1 = 1$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = 5$$

- $\boldsymbol{a} \cdot \boldsymbol{B} = \boldsymbol{c}$
  - dimensions: $1 \times d \cdot d \times e = 1 \times e$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \end{bmatrix}$$

- $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \cdot m \times n = l \times n$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 5 \\ 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 2 \\ 5 & -5 \\ 8 & 13 \end{bmatrix}$$

Element-wise Multiplication

- $\boldsymbol{a} \odot \boldsymbol{b} = \boldsymbol{c}$
  - dimensions: $1 \times d \odot 1 \times d = 1 \times d$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \odot \begin{bmatrix} 3 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 3 & 0 & -6 \end{bmatrix}$$

- $\boldsymbol{A} \odot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \odot l \times m = l \times m$

$$\begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \odot \begin{bmatrix} -1 & 0 \\ 0 & 2 \\ 0.5 & -1 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & 2 \\ 0.5 & 1 \end{bmatrix}$$

Agenda

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

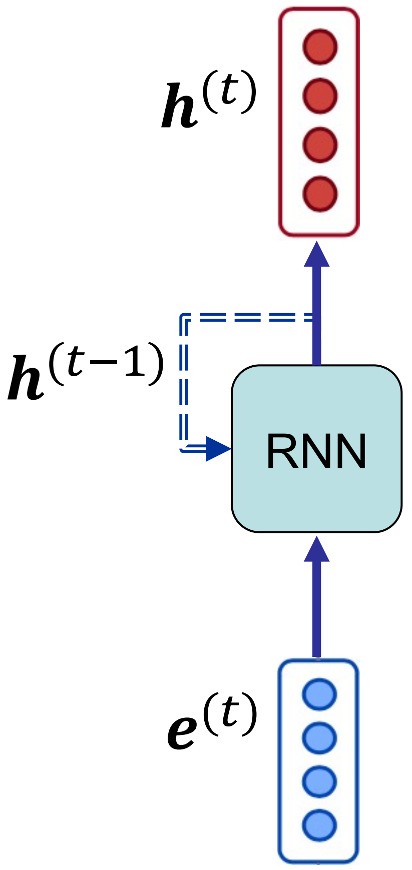
Recurrent Neural Network

- 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



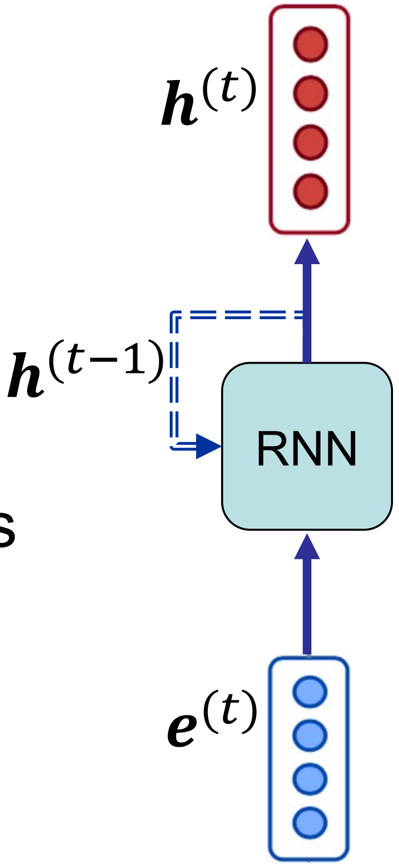
Recurrent Neural Networks

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

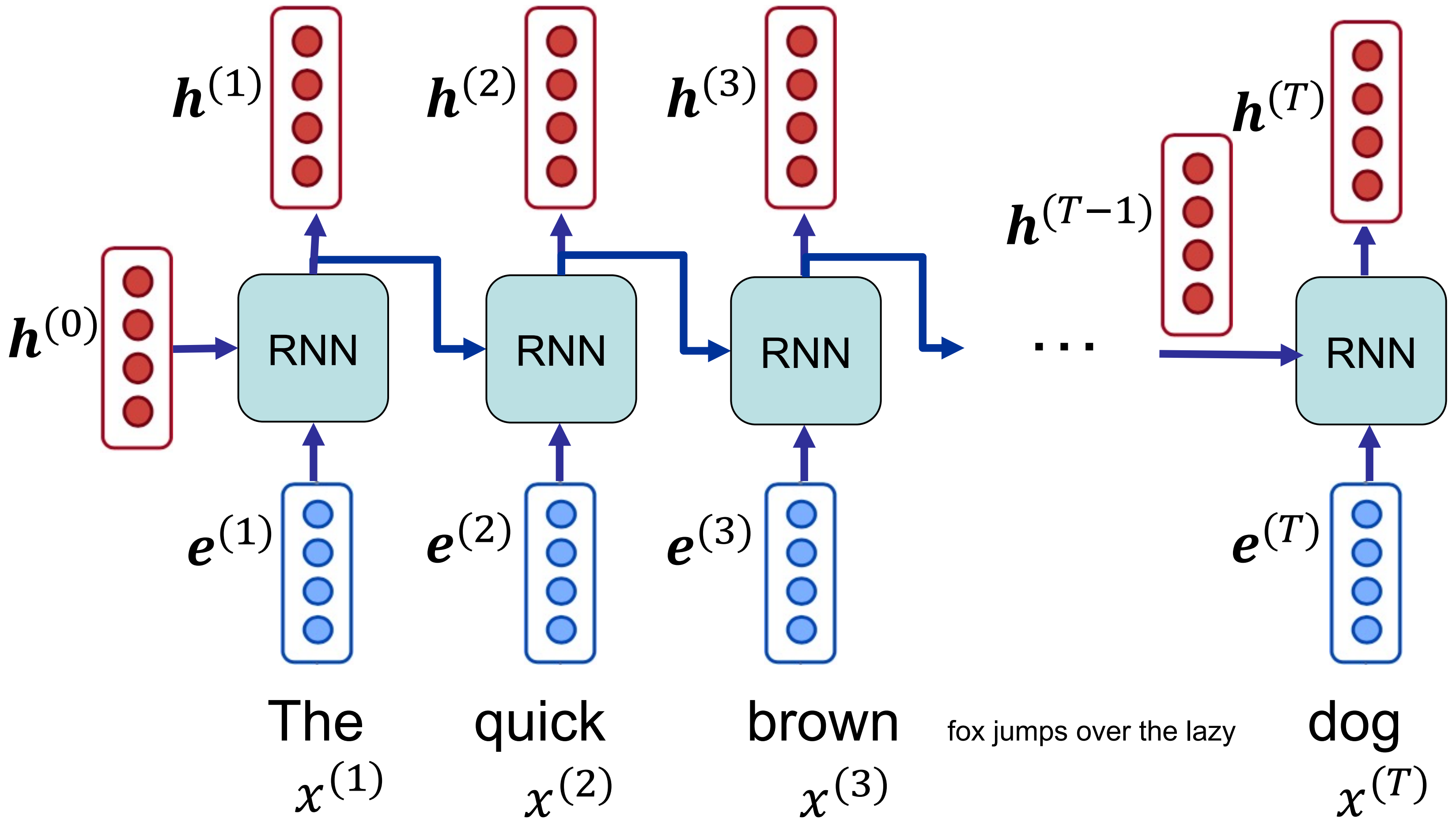
$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

- $\boldsymbol{h}^{(t)}$ 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



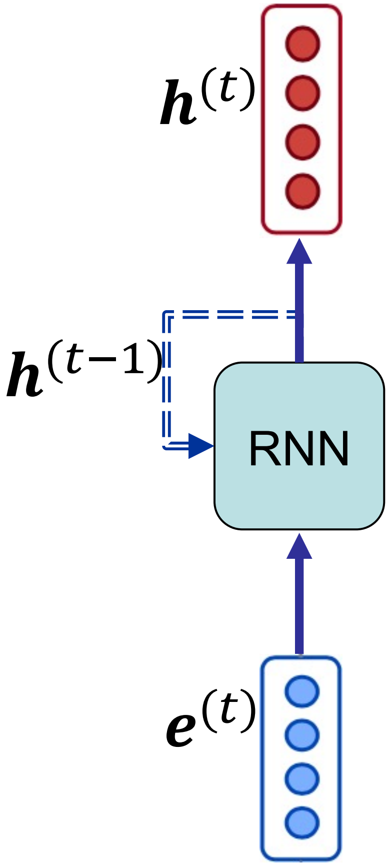
Standard (Elman) RNN

- 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

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



Agenda

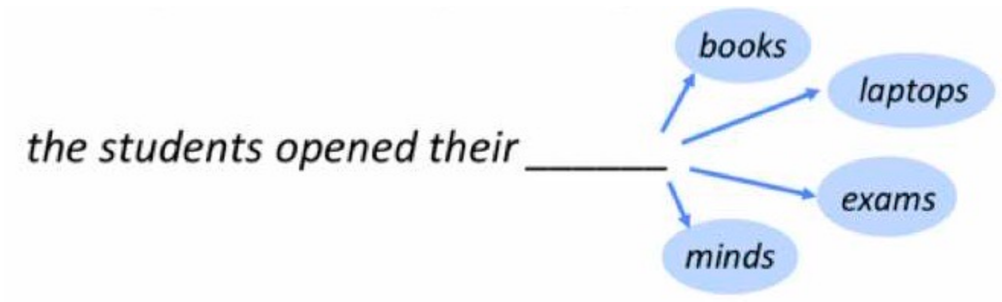
- Recurrent Neural Networks
- **Language Modeling with RNN**
- Backpropagation Through Time

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

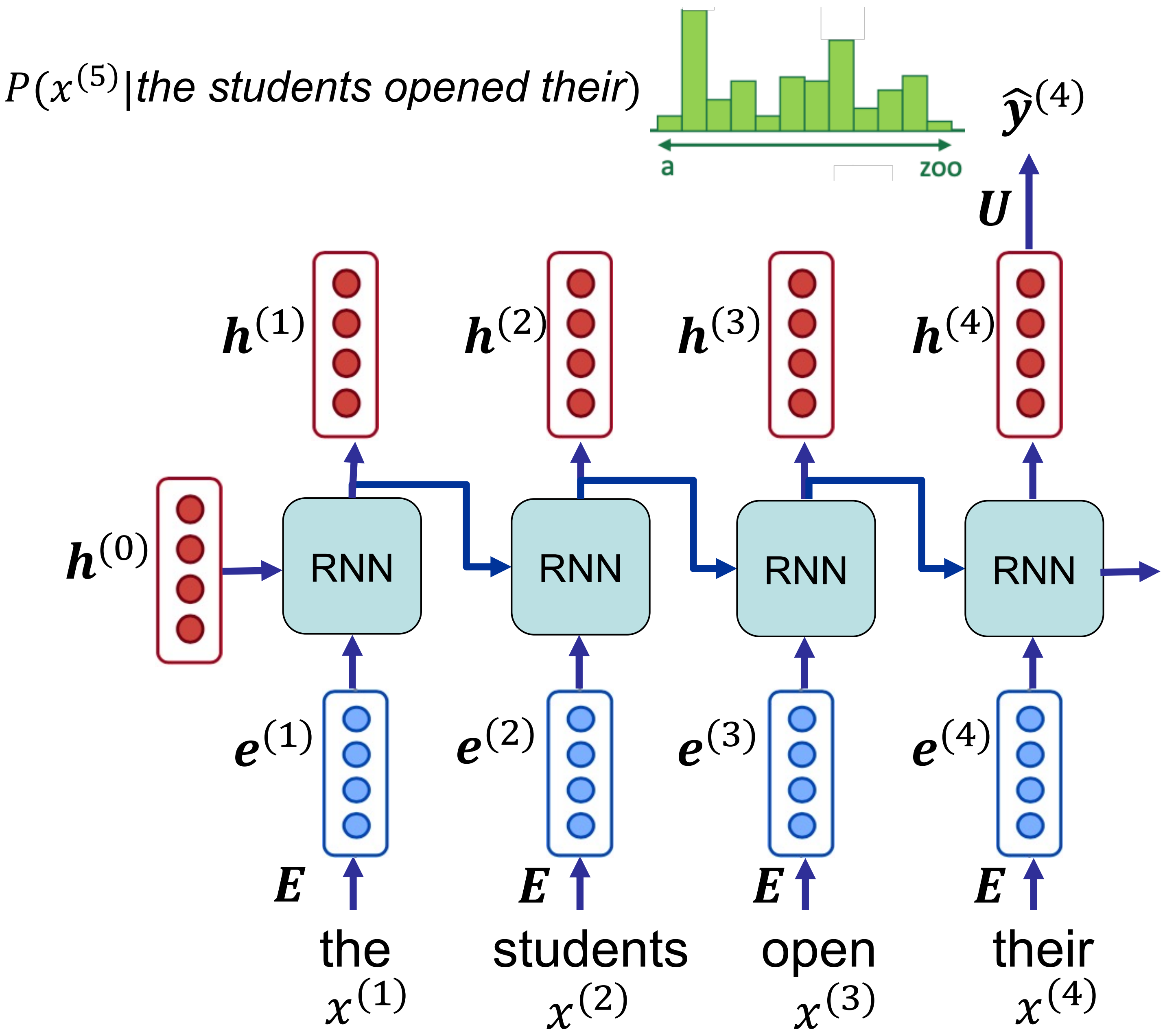
$$P(x^{(t+1)} | x^{(t)}, x^{(t-1)}, \ldots, x^{(1)})$$

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



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

RNN Language Model



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:

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

  - 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: $x^{(1)}, \dots, x^{(T)}$

- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$

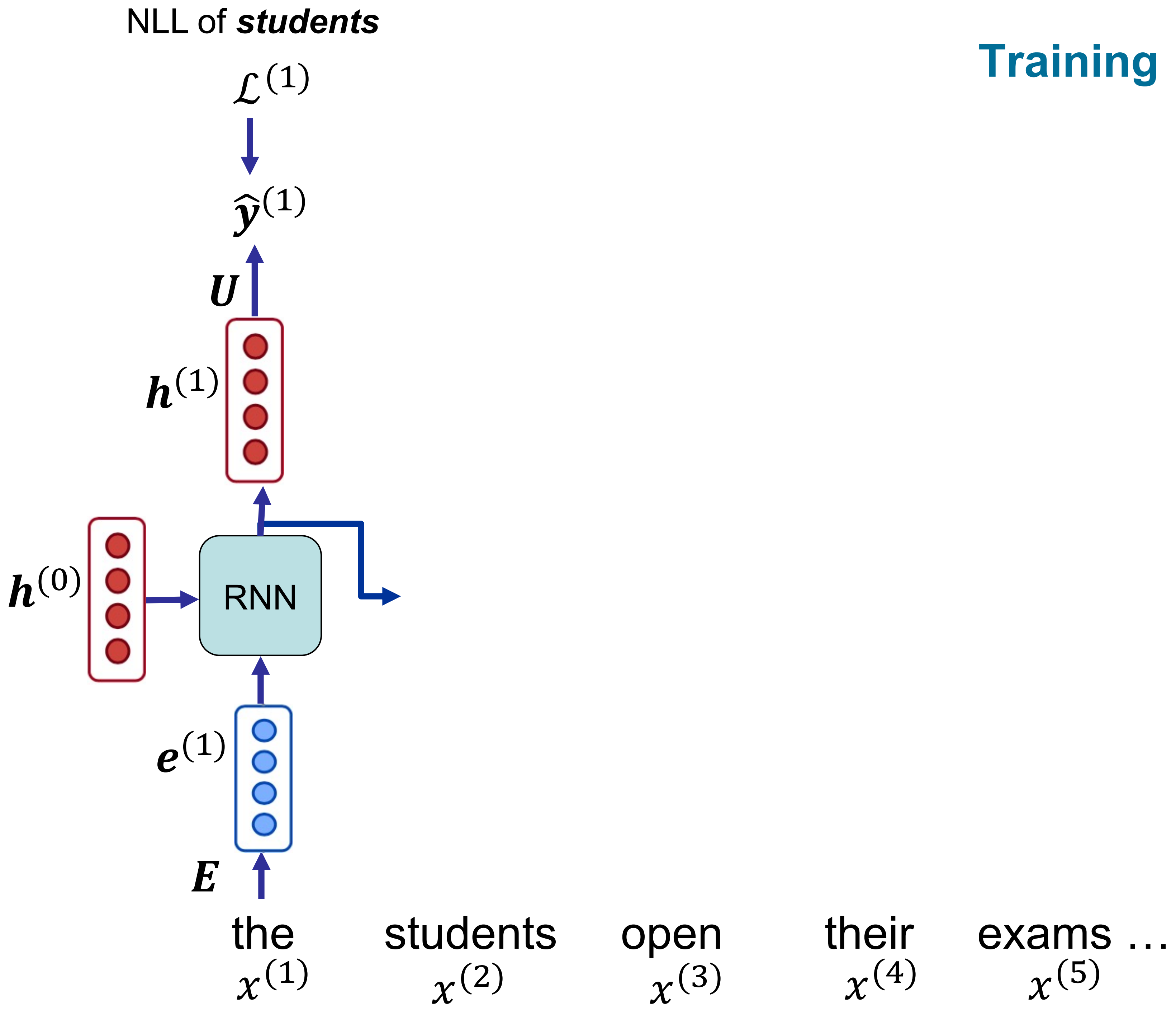
- Calculate the loss function: Negative Log Likelihood of the predicted probability of the true next word $x^{(t+1)}$

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

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

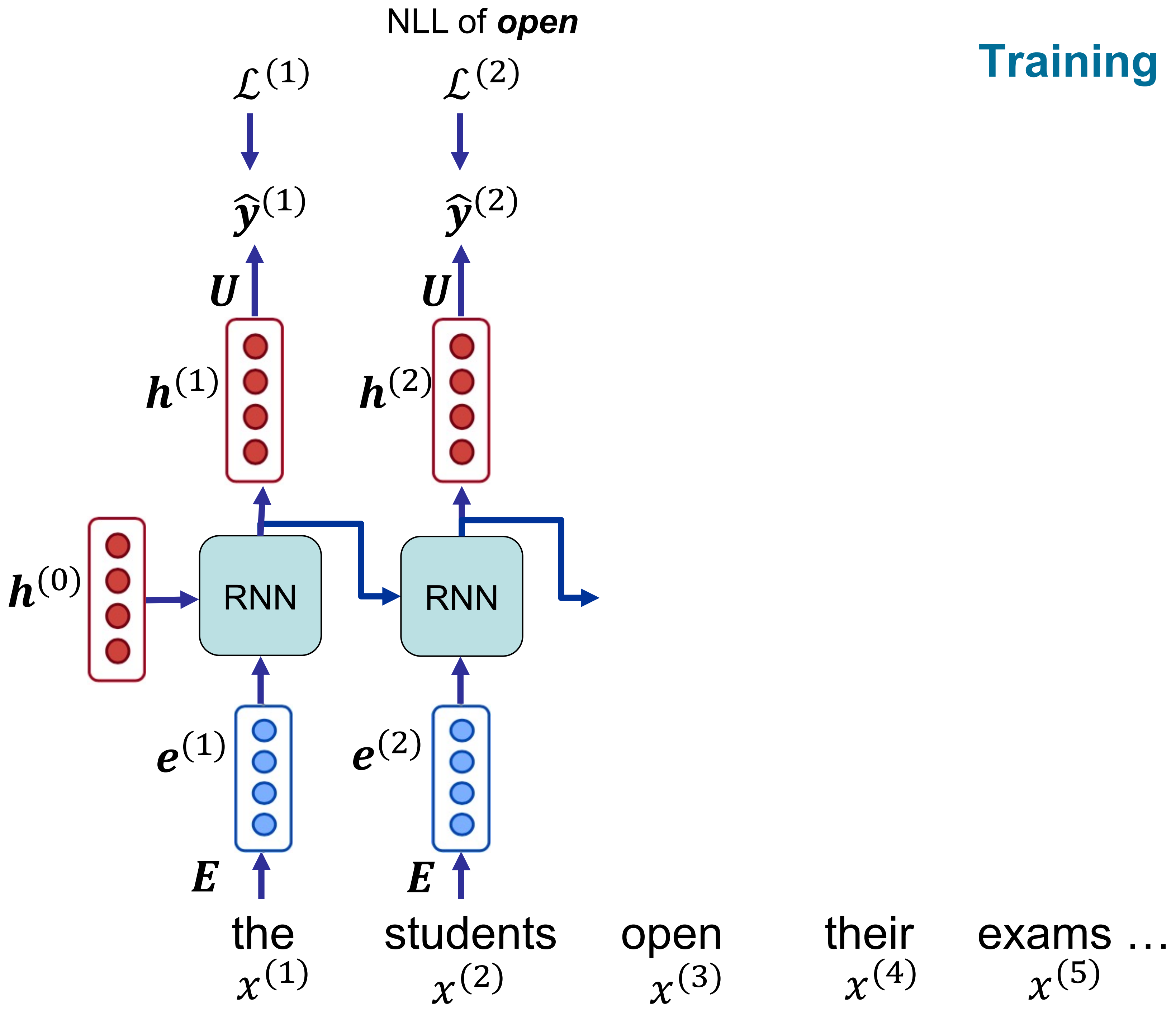
$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}^{(t)}$$

Training

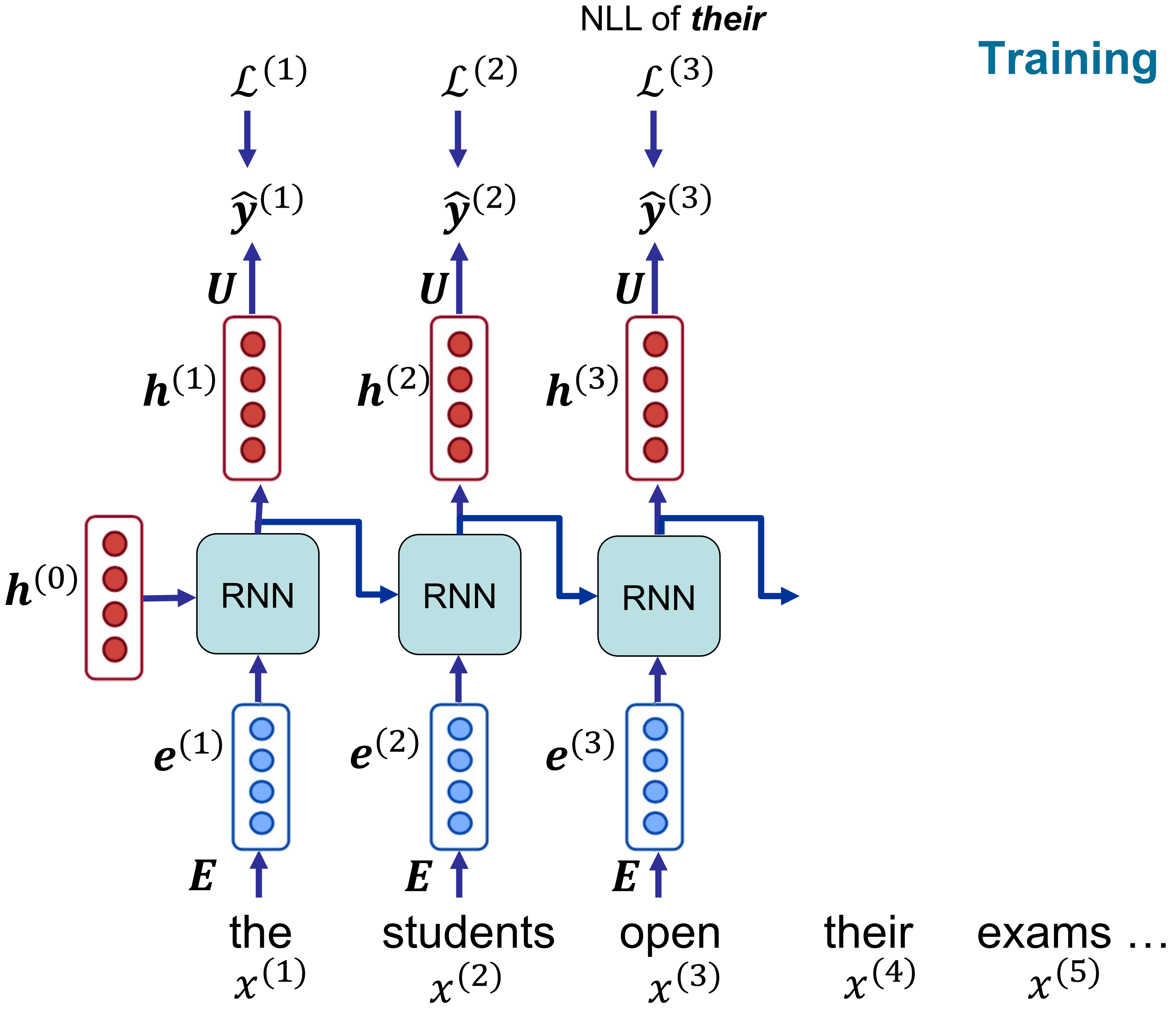


NLL of ***students***
$\mathcal{L}^{(1)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(0)}$
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{E}$
the
$x^{(1)}$
students
$x^{(2)}$
open
$x^{(3)}$
their
$x^{(4)}$
exams …
$x^{(5)}$

Training

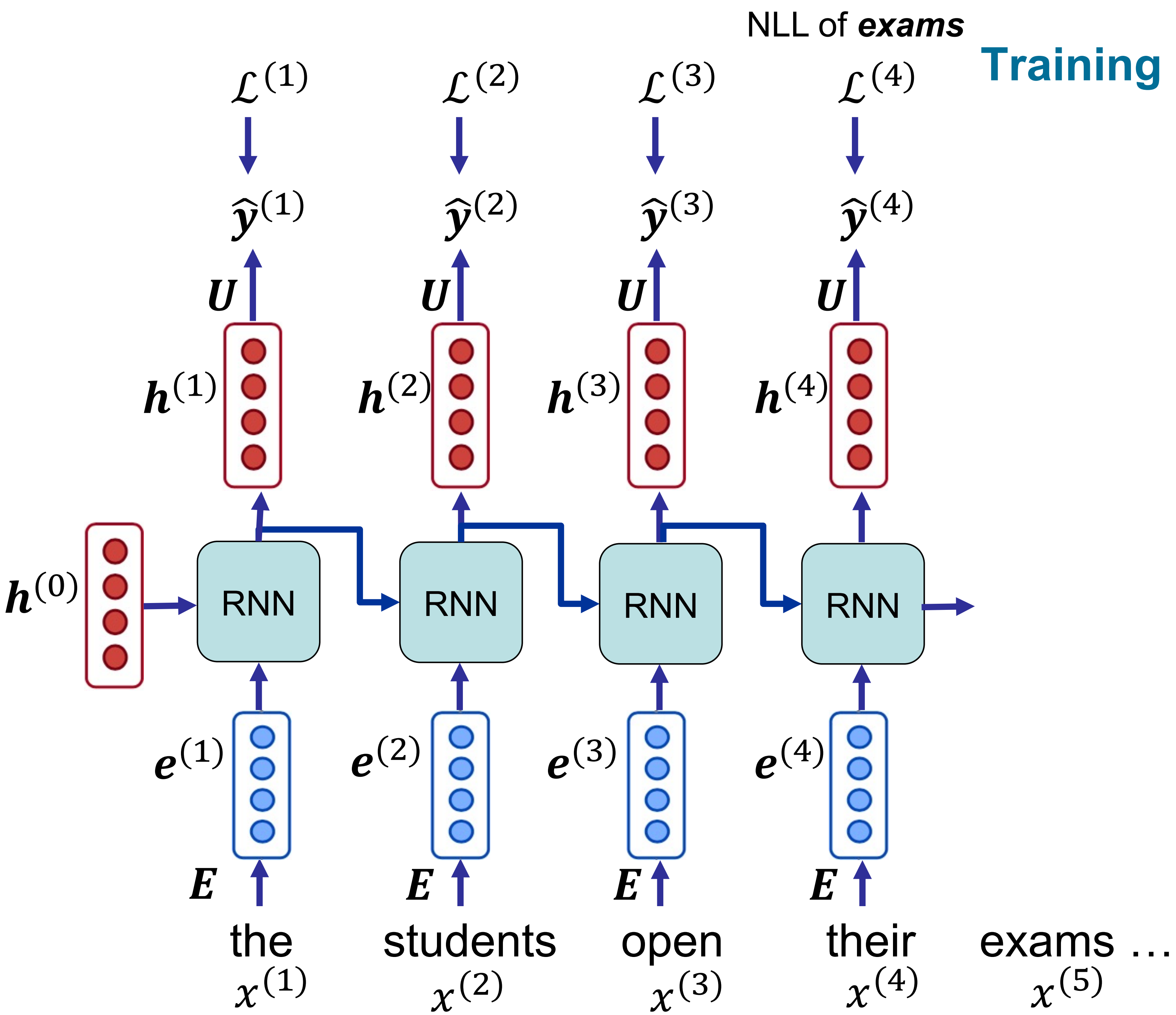

NLL of ***open***
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(0)}$
RNN
RNN
$\boldsymbol{e}^{(1)}$
$\boldsymbol{e}^{(2)}$
$\boldsymbol{E}$
$\boldsymbol{E}$
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***
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)}$
$\mathcal{L}^{(4)}$
$\hat{\boldsymbol{y}}^{(1)}$
$\hat{\boldsymbol{y}}^{(2)}$
$\hat{\boldsymbol{y}}^{(3)}$
$\hat{\boldsymbol{y}}^{(4)}$
$\boldsymbol{U}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
$\boldsymbol{h}^{(0)}$
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 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 – $\boldsymbol{h}^{(0)}$

- 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

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

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

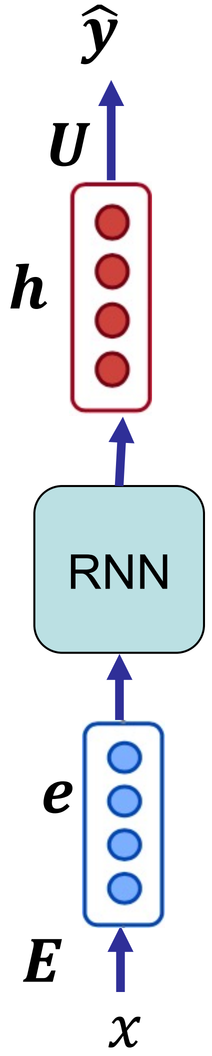
$$\begin{bmatrix} x^{(L+1)} & \dots & x^{(2L)} \\ x^{(L+\tau+1)} & \dots & x^{(\tau+2L)} \end{bmatrix}$$

Training RNN Language Model – Parameters

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

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

- Encoder and decoder embeddings have most of the parameters

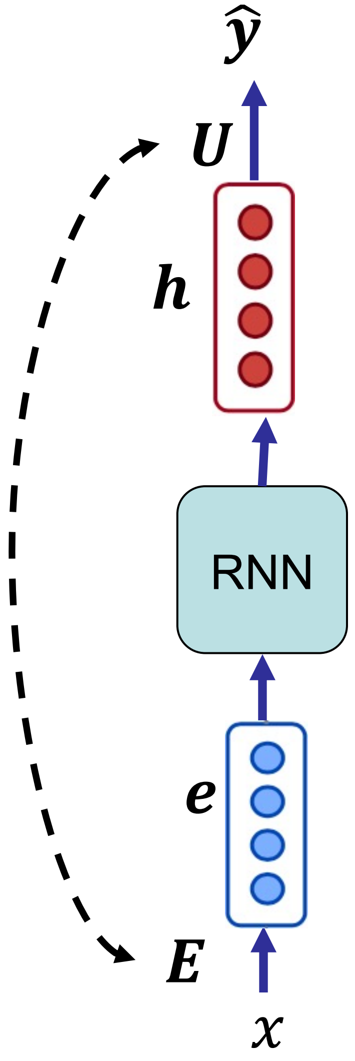


Training RNN Language Model – Weight Tying

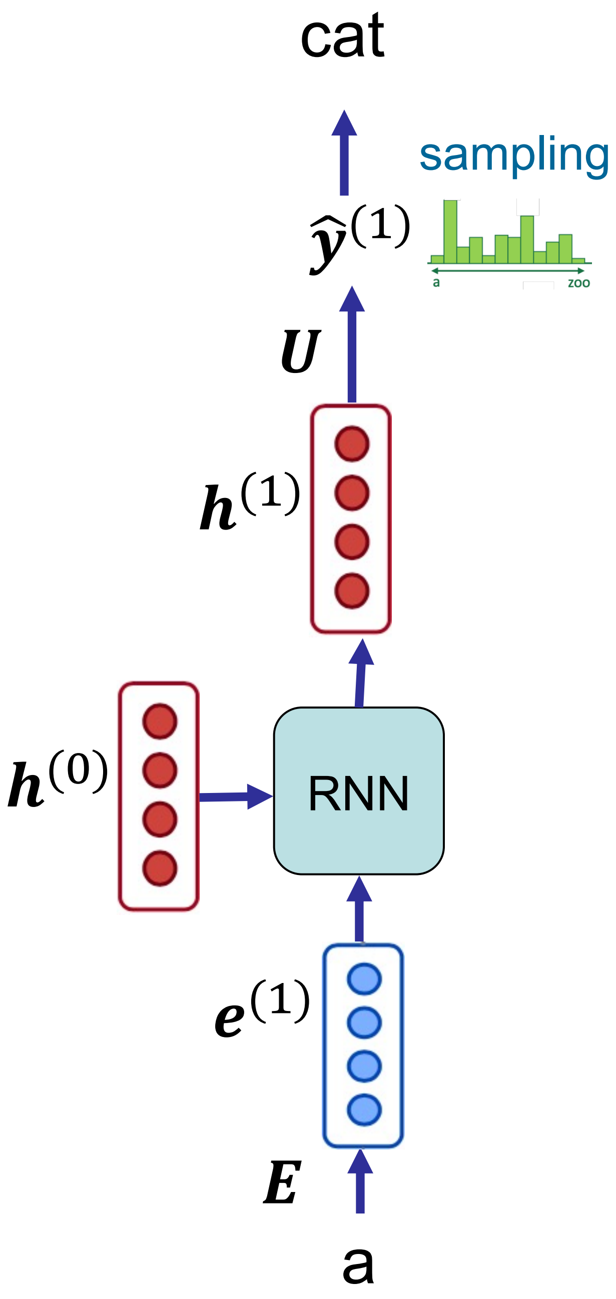
- Parameters in a vanilla RNN (bias terms discarded)
  - $\boldsymbol{E} \rightarrow |\mathbb{V}| \times h$
  - $\boldsymbol{U} \rightarrow h \times |\mathbb{V}|$
  - $\boldsymbol{W}_i \rightarrow d \times h$
  - $\boldsymbol{W}_h \rightarrow h \times h$

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

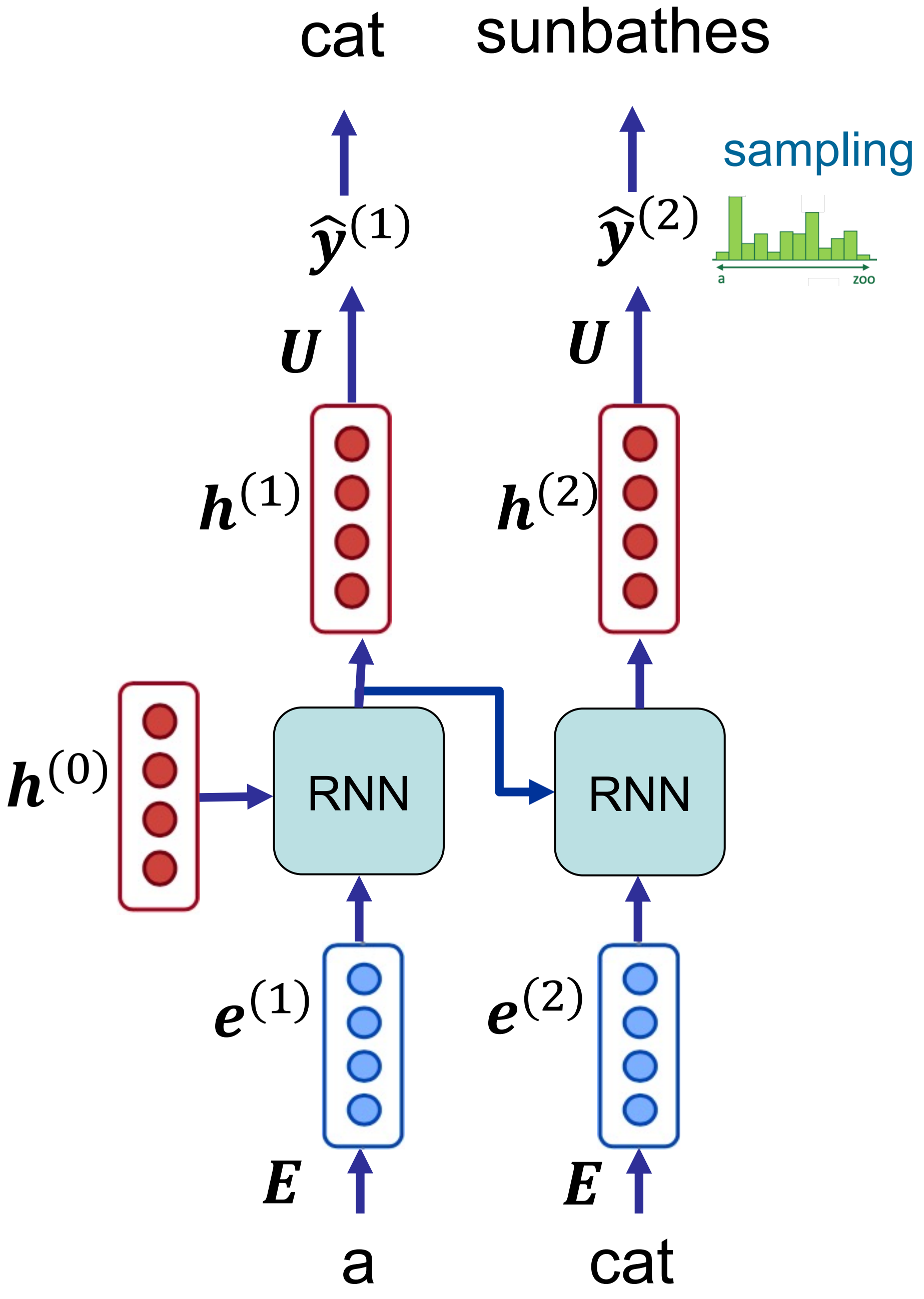
- Weight tying: set the decoder parameters the same as encoder parameters (saving $|\mathbb{V}| \times h$ decoding parameters)
  - In this case d must be equal to h
  - If $d \neq h$, usually a linear projects the output vector from h to d dimensions



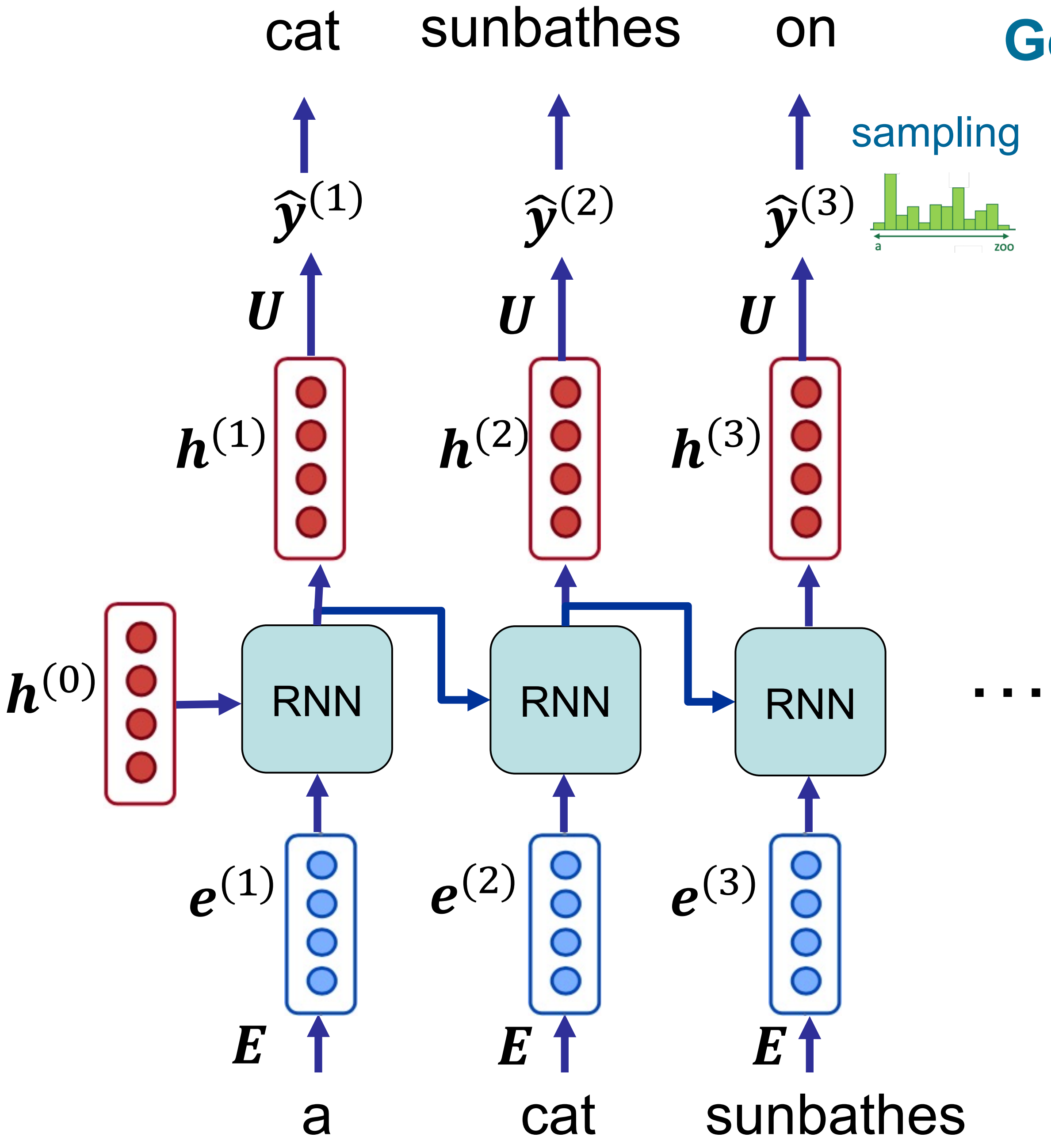
Generating text



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

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

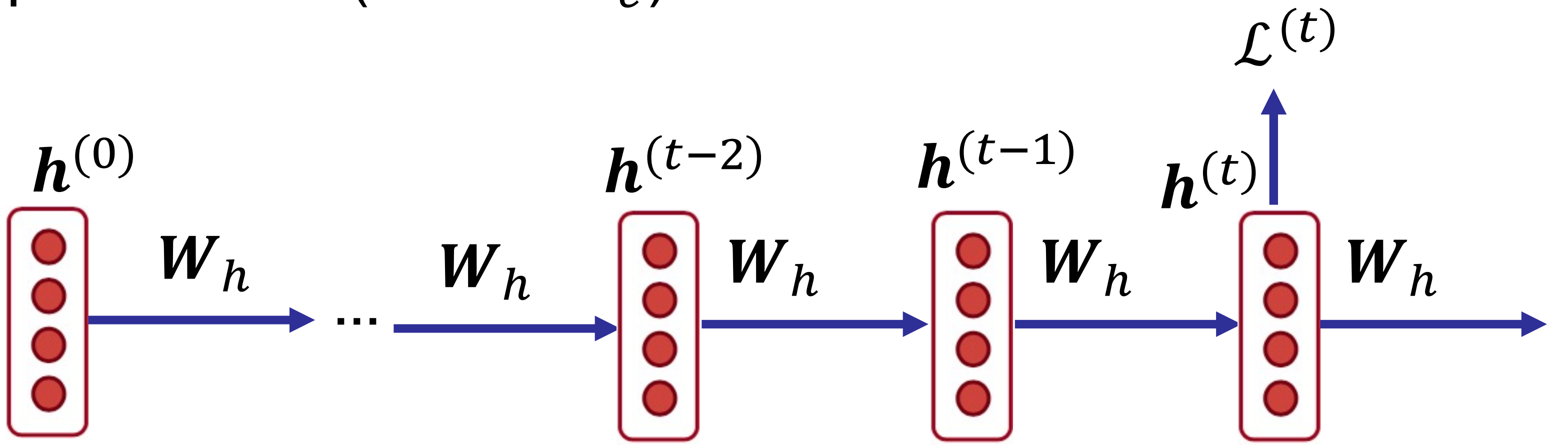
- **Cons**:
  - Recurrent computation is slow → (in its vanilla form) does not fully exploit the parallel computation capability of GPUs
  - Biased toward recent incidents → in practice, RNN has the tendency to use the information of recent steps (harder to access information from many steps back)

Agenda

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

Backpropagation for RNNs

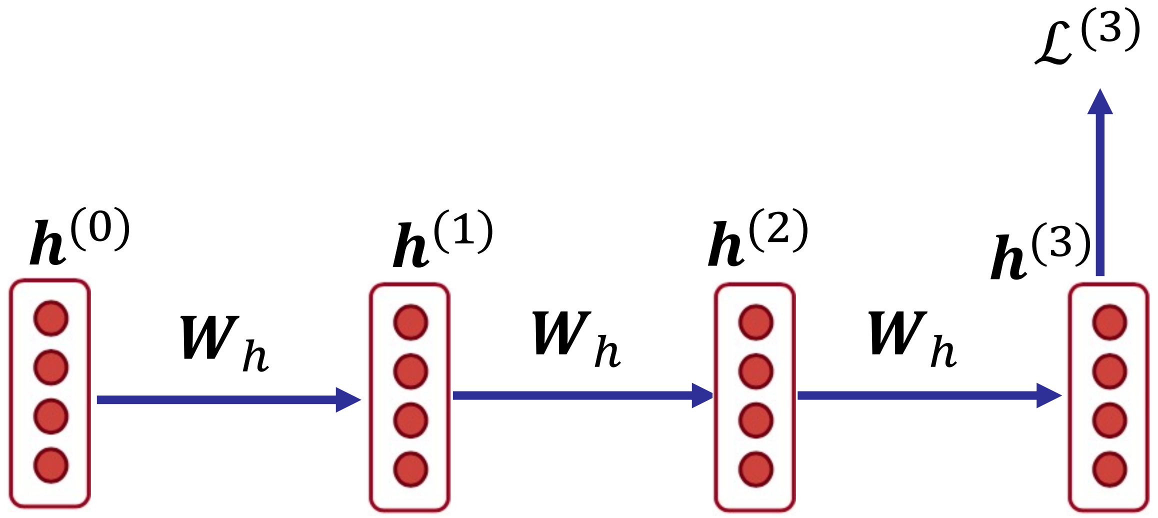
- Unrolling the computation graph of RNN
- Simplified: the interactions with $\boldsymbol{U}$ and also input parameters ($\boldsymbol{E}$ and $\boldsymbol{W}_e$) are removed



- What is …

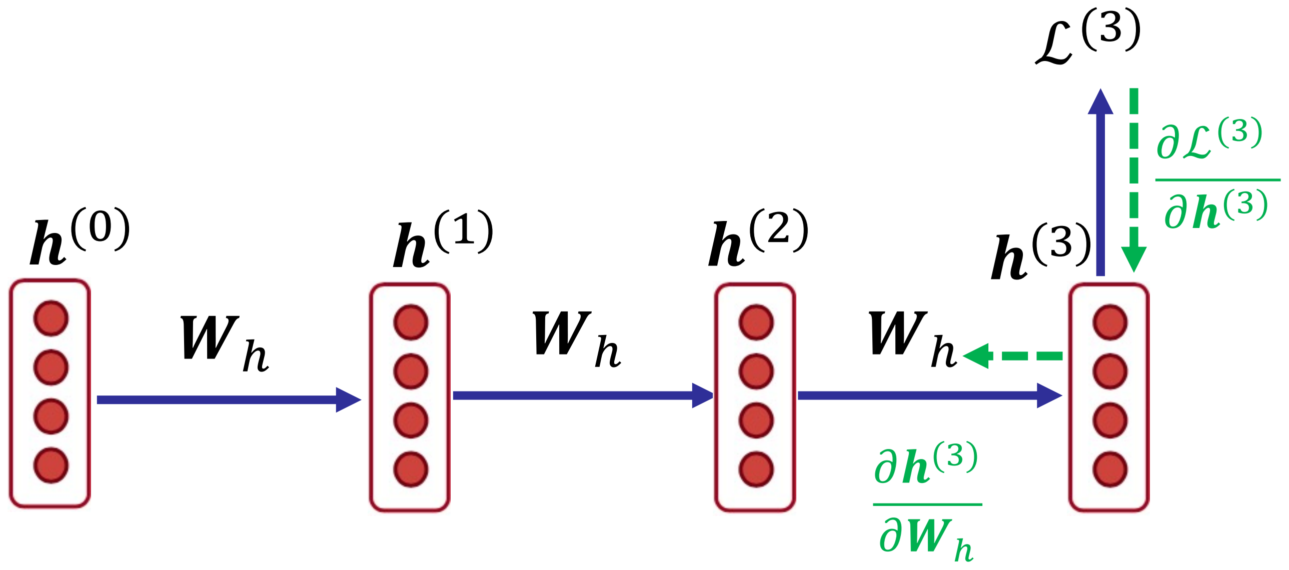
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = ?$$

Backpropagation for RNNs



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

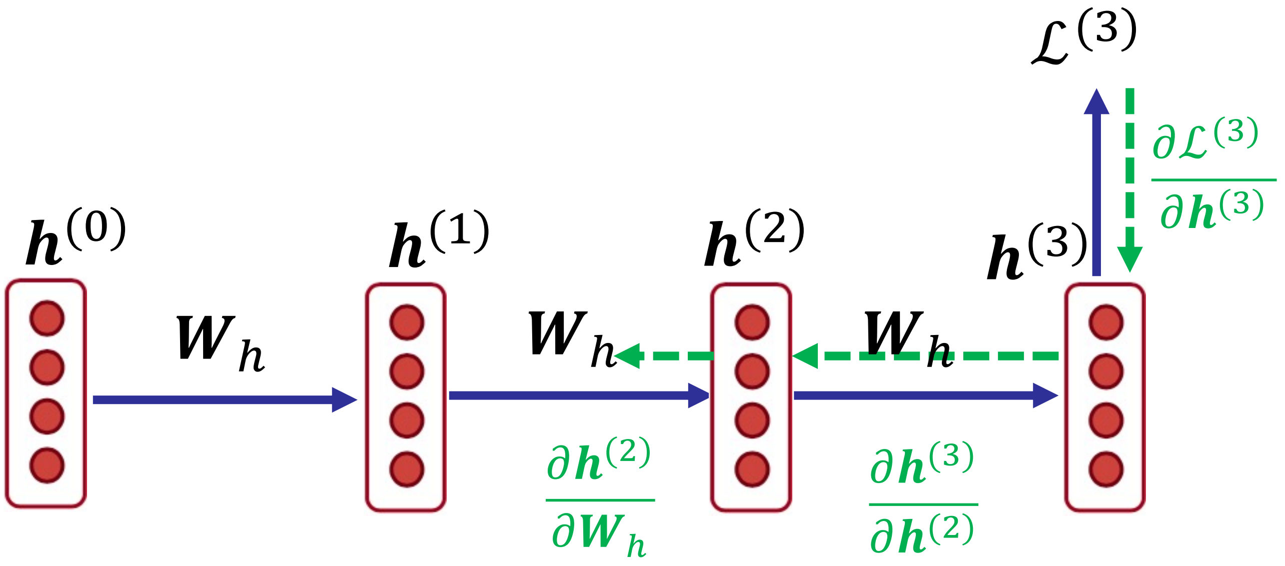
Backpropagation for RNNs



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

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

Backpropagation for RNNs

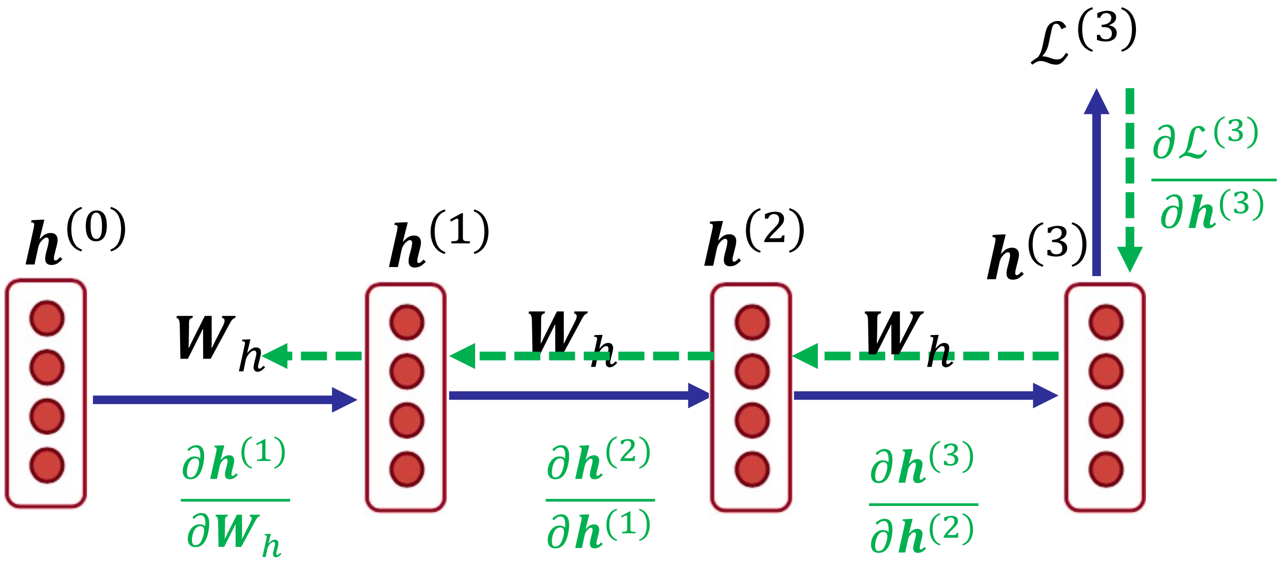


$\mathcal{L}^{(3)}$
$\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$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

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

- 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 (BPTT)

- 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)

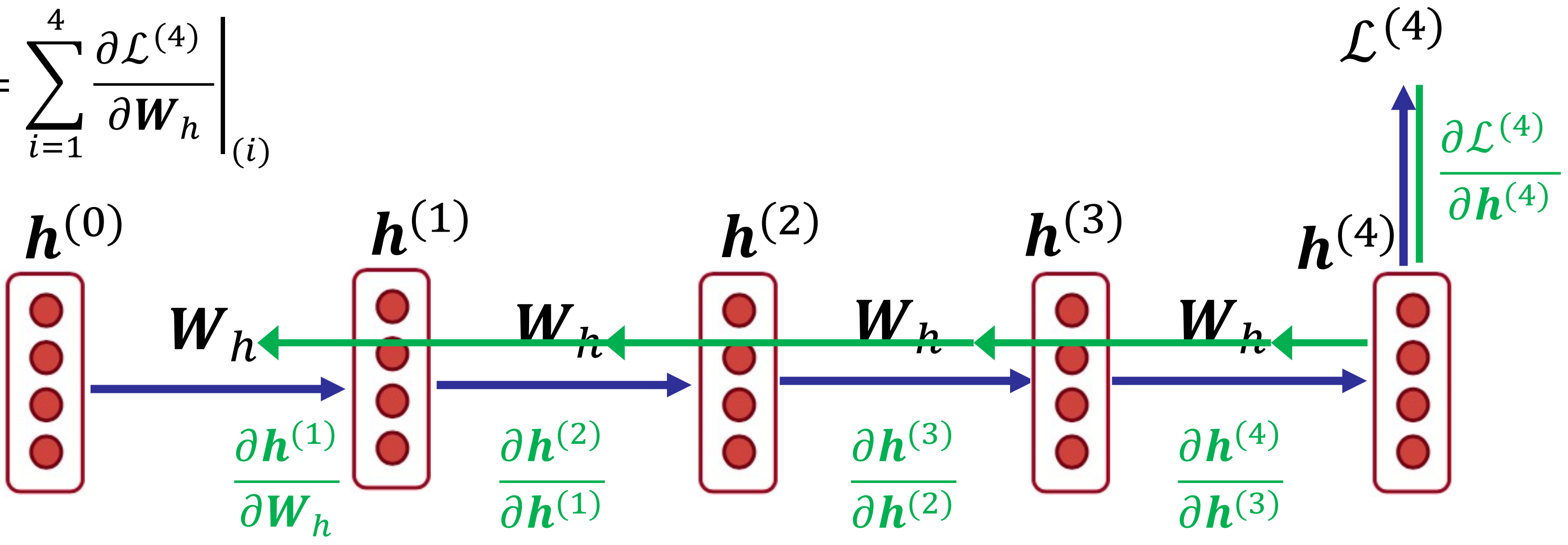
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left. \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} \right|_{(i)}$$

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

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

Backpropagation Through Time (BPTT) – all in one!

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{4} \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$



$\mathcal{L}^{(4)}$
$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}$
$\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}$
$\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}$

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(4)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{W}_h}$$

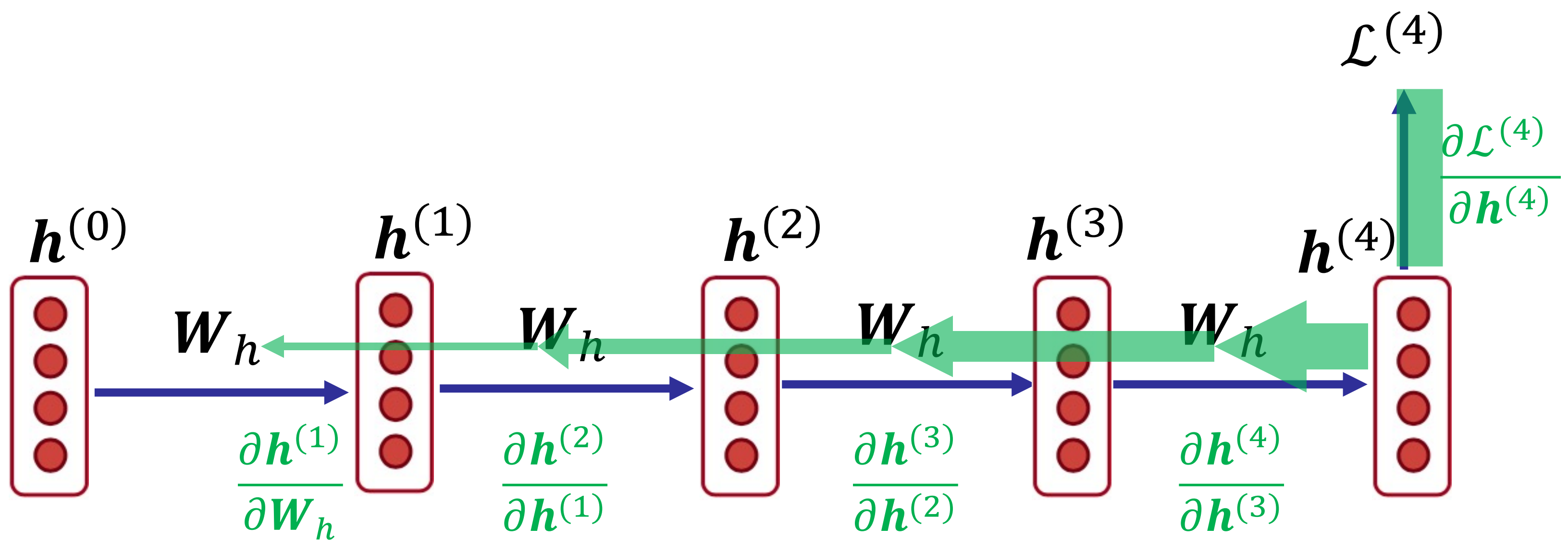
$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(3)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{W}_h}$$

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(2)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$$

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(1)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

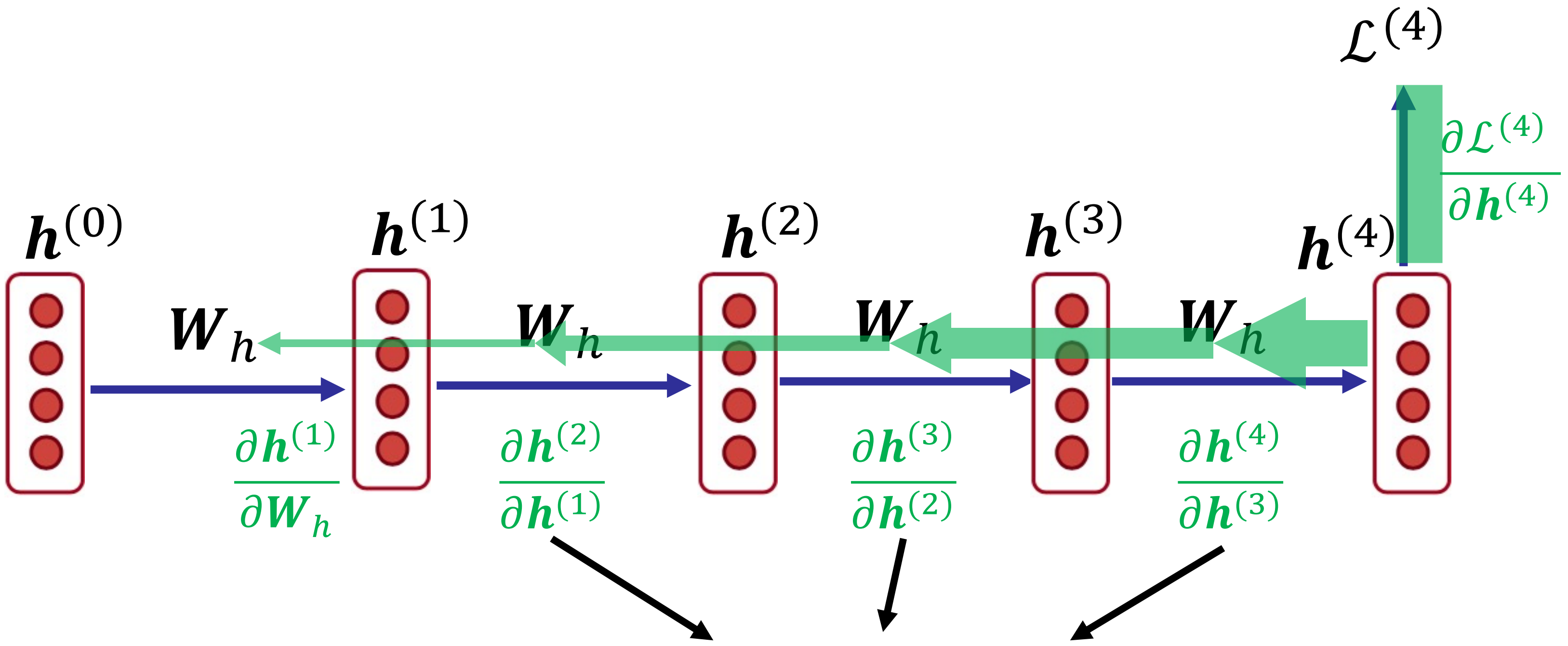
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}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

Vanishing/Exploding gradient


$\mathcal{L}^{(4)}$
$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}$
$\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}$
$\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}$

- In practice, the gradient regarding each time step becomes smaller and smaller as it goes back in time → Vanishing gradient
- While less often, this may also happen other way around: the gradient regarding further time steps becomes larger and larger→ Exploding gradient

Vanishing/Exploding gradient – why?


$\mathcal{L}^{(4)}$
$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}}$
$\boldsymbol{h}^{(0)}$
$\boldsymbol{h}^{(1)}$
$\boldsymbol{h}^{(2)}$
$\boldsymbol{h}^{(3)}$
$\boldsymbol{h}^{(4)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$
$\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}$
$\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}$
$\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}$

If these gradients are small, their multiplication gets smaller. As we go further back, the final gradient contains more of these!

$$\left.\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\right|_{(1)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

Vanishing/Exploding gradient – why?

- What is $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$?!
- Recall the definition of RNN:

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

- Let’s replace sigmoid (σ) with a simple linear activation ($y = x$) function.

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

- In this case:

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \boldsymbol{W}_h$$

Vanishing/Exploding gradient – why?

- Recall the BPTT formula (for the simplified case):

$$\left.\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(i)} = \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \underbrace{\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

- Given $l = t - i$, the BPTT formula can be rewritten as:

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

If weights in $\boldsymbol{W}_h$ are small (i.e. eigenvalues of $\boldsymbol{W}_h$ are smaller then 1), these term gets *exponentially* smaller

Why is vanishing/exploding gradient a problem?

- Vanishing gradient
  - Gradient signal from faraway “fades away” and becomes insignificant in comparison with the gradient signal from close-by
  - Long-term dependencies are not captured, since model weights are updated only with respect to near effects

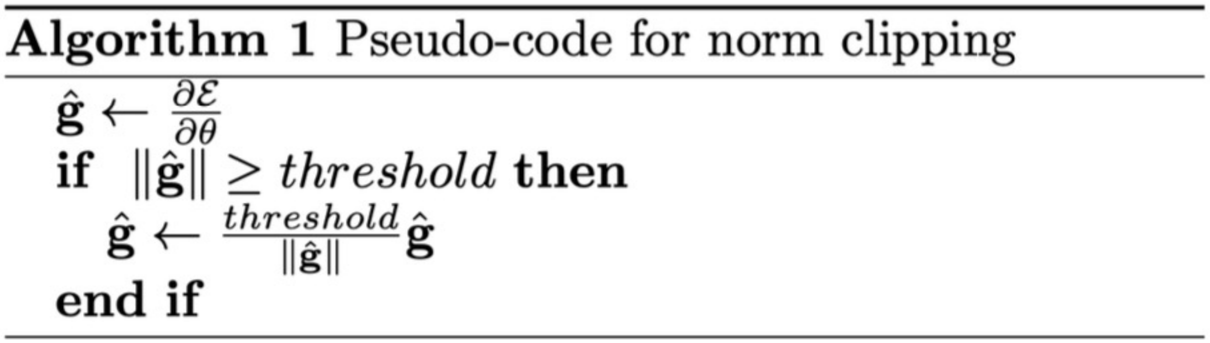
  → one approach to address it: RNNs with gates – LSTM, GRU

- Exploding gradient
  - Gradients become too big → SGD update steps become too large
  - This causes (large loss values and) large updates on parameters, and eventually unstable training

  → main approach to address it: Gradient clipping

Gradient clipping

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale the gradient down



Algorithm 1 Pseudo-code for norm clipping

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if
```

- Intuition: take the step in the same direction, but with a smaller step

Problem with vanilla RNN – summary

- It is too difficult for the hidden state of vanilla RNN to learn and preserve information of several time steps
  - In particular as new contents are constantly added to the hidden state in every step

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

In every step, input vector "adds" new content to hidden state

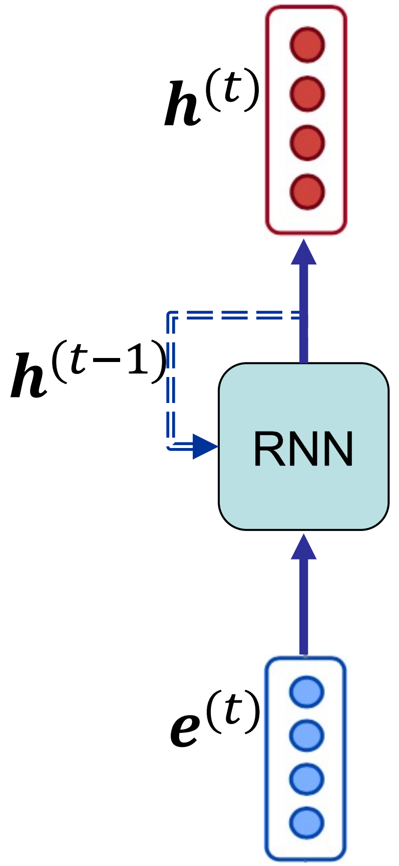
Agenda

- Recurrent Neural Networks
- Language Modeling with RNN
- Backpropagation Through Time

Recap

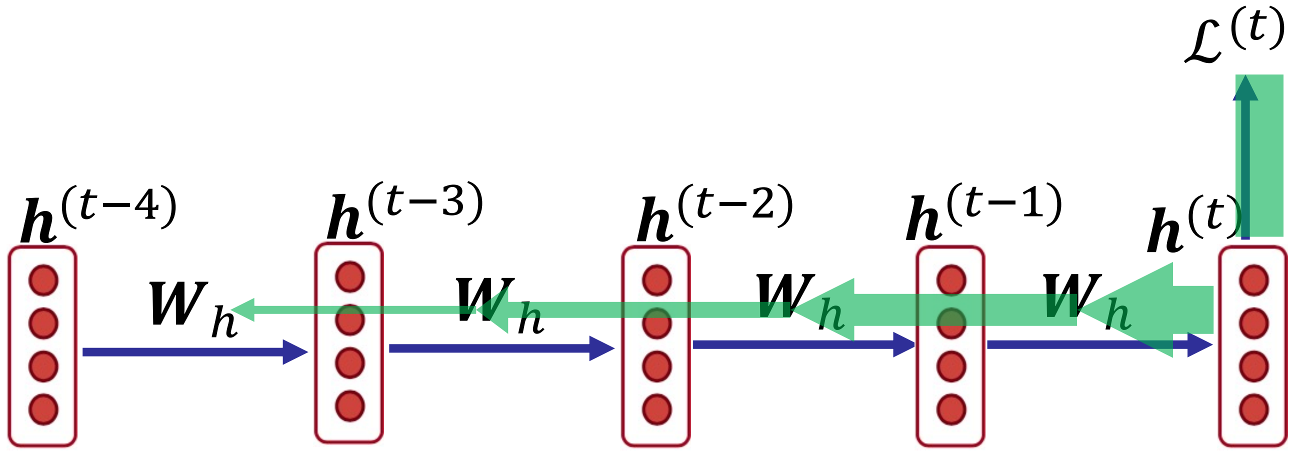
- A Language Model calculates …
  - the probability of the appearance of a word/subword/character given its context

- Recurrent Neural Network
  - Predicts next entity (word/subword/character) based on a *memory* of previous steps and the given input



Recap

- Backpropagation Through Time



$\mathcal{L}^{(t)}$
$\boldsymbol{h}^{(t-4)}$
$\boldsymbol{h}^{(t-3)}$
$\boldsymbol{h}^{(t-2)}$
$\boldsymbol{h}^{(t-1)}$
$\boldsymbol{h}^{(t)}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$
$\boldsymbol{W}_h$