北京大学
PEKING UNIVERSITY

Language Models

Kun Yuan

Center for Machine Learning Research @ Peking University

Contents

The embedding materials are from a great course [1]

The language model materials are from a great course [2]

---

[1] The deep learning specialization

[2] Stanford CS224n: Natural Language Processing with Deep Learning

1-hot word representation

- Vocabulary set = {a, aaron, …, zulu}; the size is typically on the order of 10,000
- 1-hot representation is the most natural idea to represent word

Man (5391)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Woman (9853)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

King (4914)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Queen (7159)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Apple (456)

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Orange (6527)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

1-hot word representation

- 1-hot representation ignores semantic relationship

I like orange juice. → I like apple ____.

- “Orange” and “apple” should be close to each other. Language model should fill in “juice”

- But in one-hot representation, apple and orange are not close to each other

Man (5391)	Woman (9853)	King (4914)	Queen (7159)	Apple (456)	Orange (6527)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Semantic representation

北京大学 PEKING UNIVERSITY

- Each word is represented with vectors that involve semantics

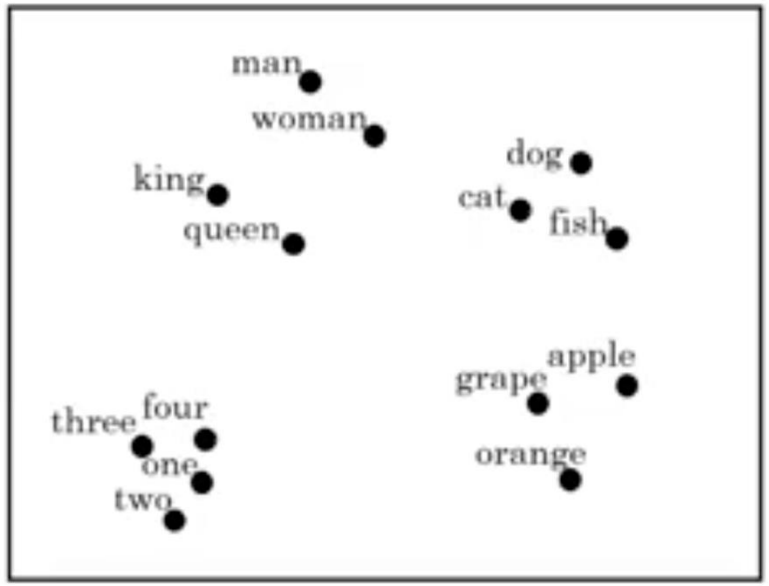
	Man (5391)	Woman (9853)	King (4914)	Queen (7159)	Apple (456)	Orange (6527)
Gender	-1.00	1.00	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.51	0.47	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

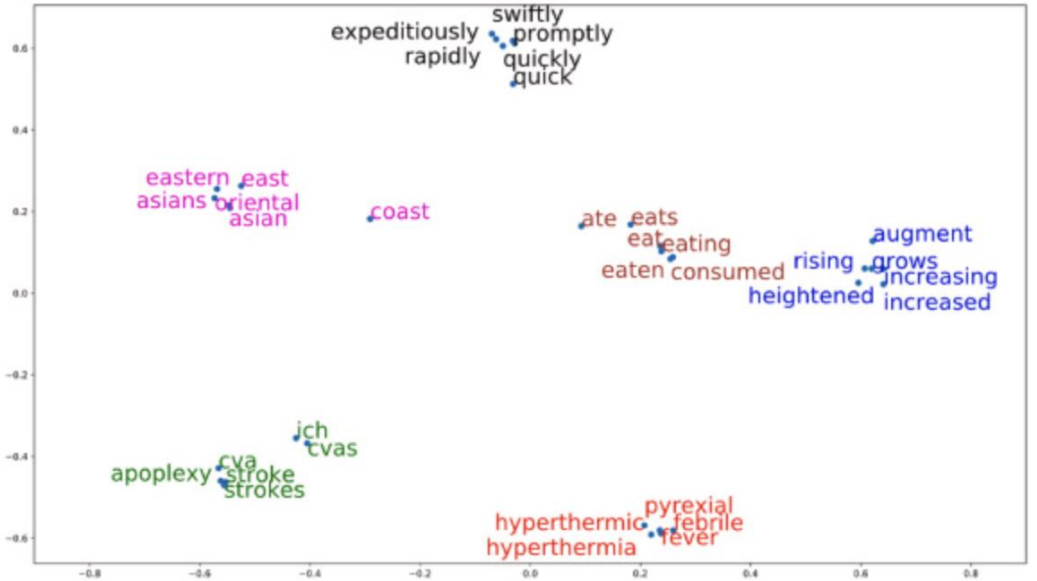
- “Man” is close to “Woman”, “King” is close to “Queen”, and “Apple” is close to “Orange”

- Semantic representation can be much shorter than 1-hot representation

Semantic representation

- Visualization in semantic representation





Semantic representation

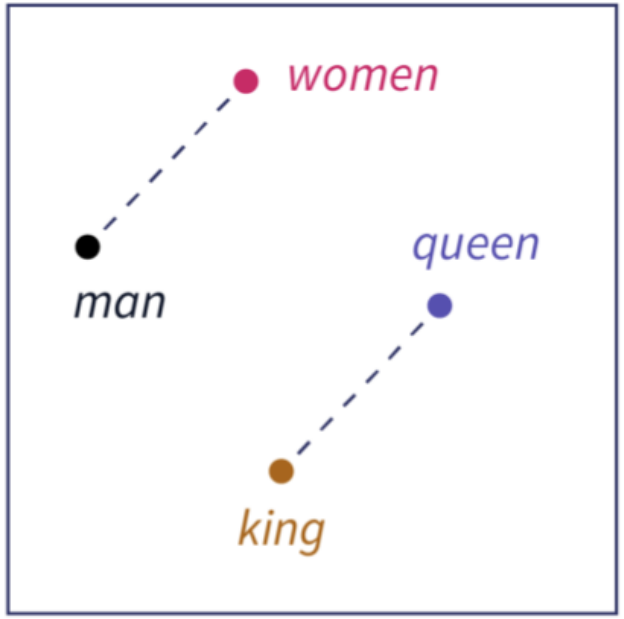
- Sematic representation helps the down-stream tasks

I like orange juice. → I like apple ____.

- Since “orange” and “orange” are close to each other，language model should fill in juice

Semantic representation

- Sematic representation helps find synonyms or antonyms
- Example: given “man” vs “women”, fill in “king” vs “____”



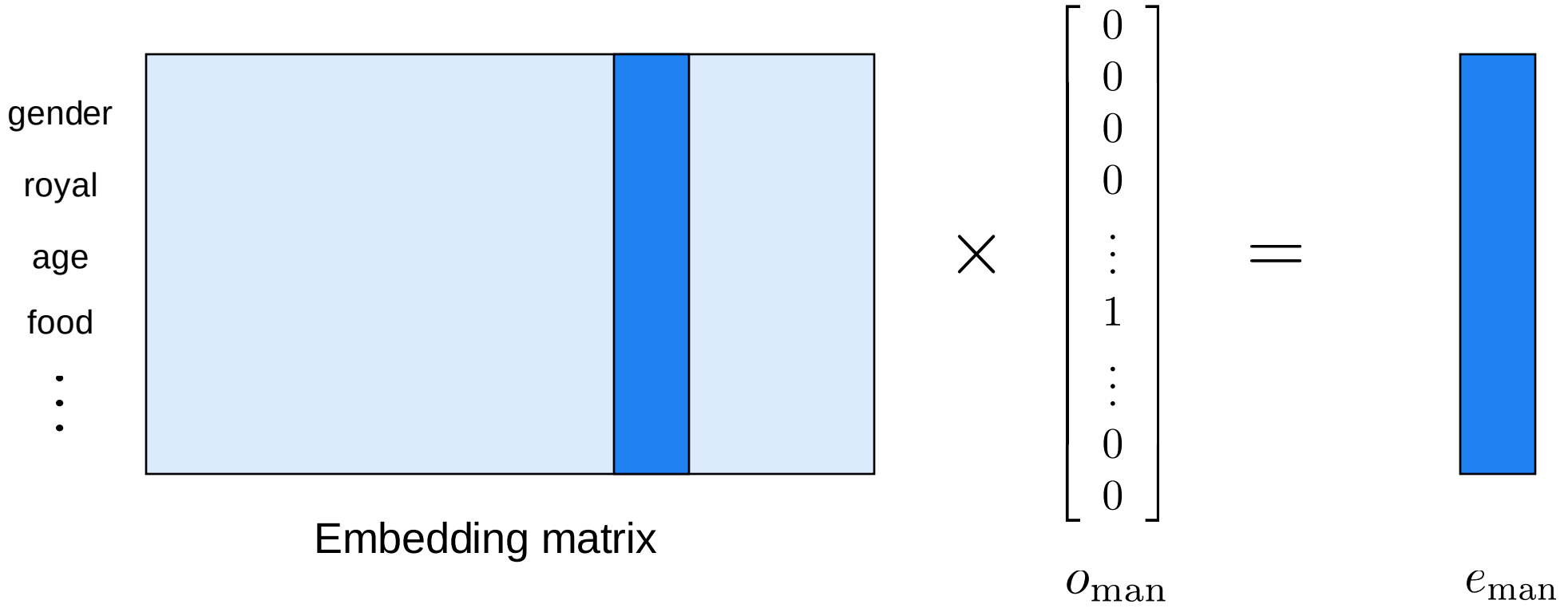
$$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_w$$

$$\rightarrow \quad e_w \approx e_{\text{king}} - e_{\text{man}} + e_{\text{woman}}$$

$$\rightarrow \quad w = \arg\max_u \{e_u, e_{\text{king}} - e_{\text{man}} - e_{\text{woman}}\}$$

1-hot representation to semantic representation

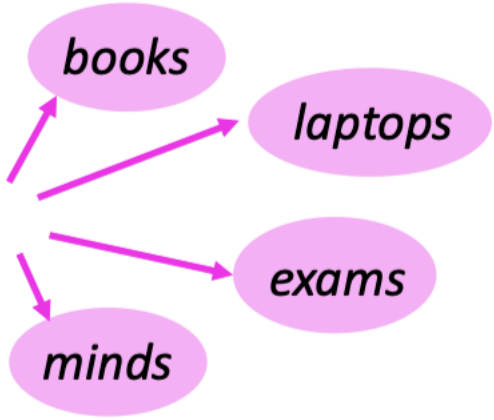
- Given the embedding matrix, we can easily achieve the semantic representation as follows



Language modeling

- **Language Modeling** is the task of predicting what word comes next

the students opened their _____

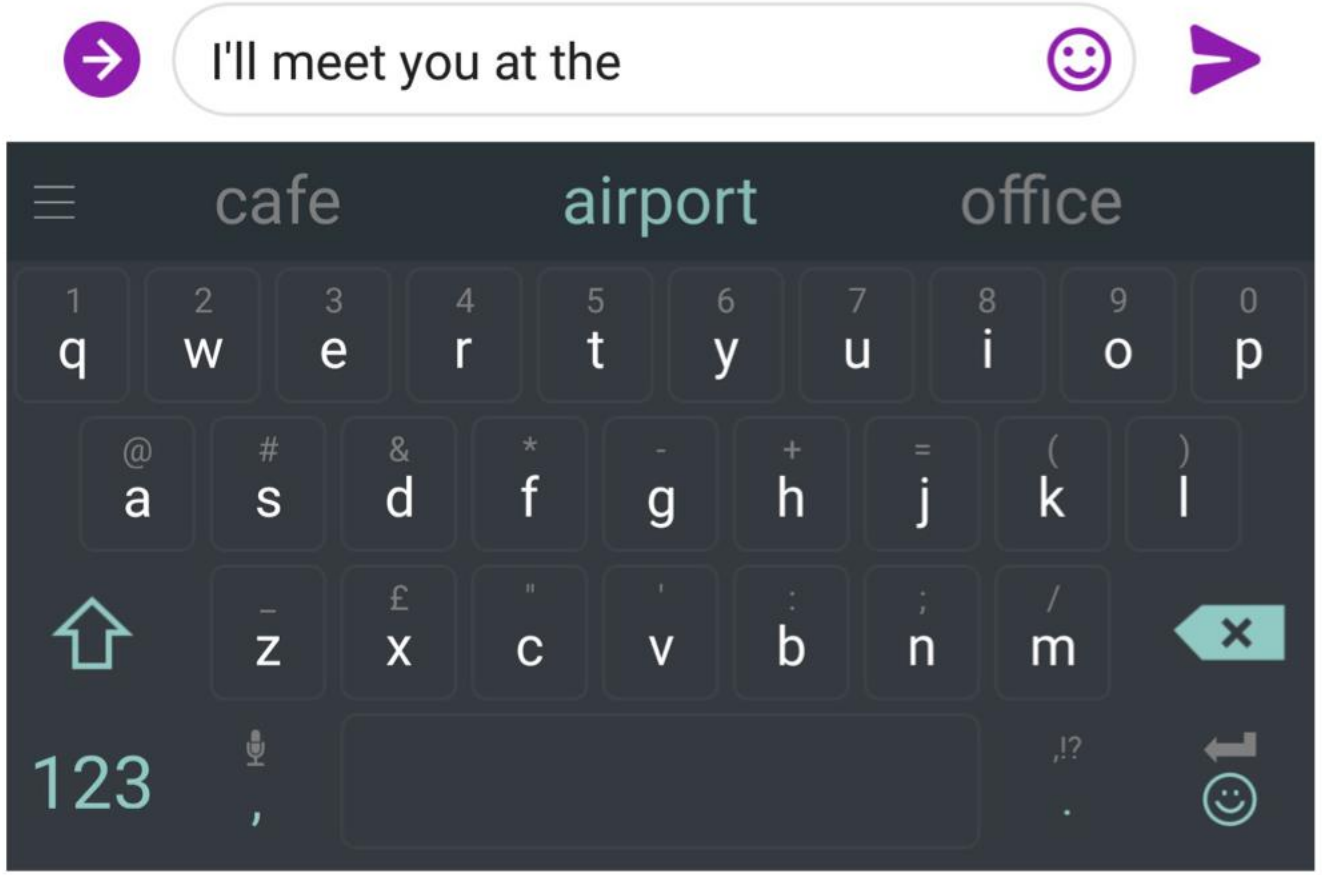


- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(x^{(t+1)} | \ x^{(t)}, \ldots, x^{(1)}) \quad \text{where} \quad x^{(t)} \in \{x_1, \cdots, x_{|V|}\}$$

- A system that does this is called a **Language Model**

You use language models everyday



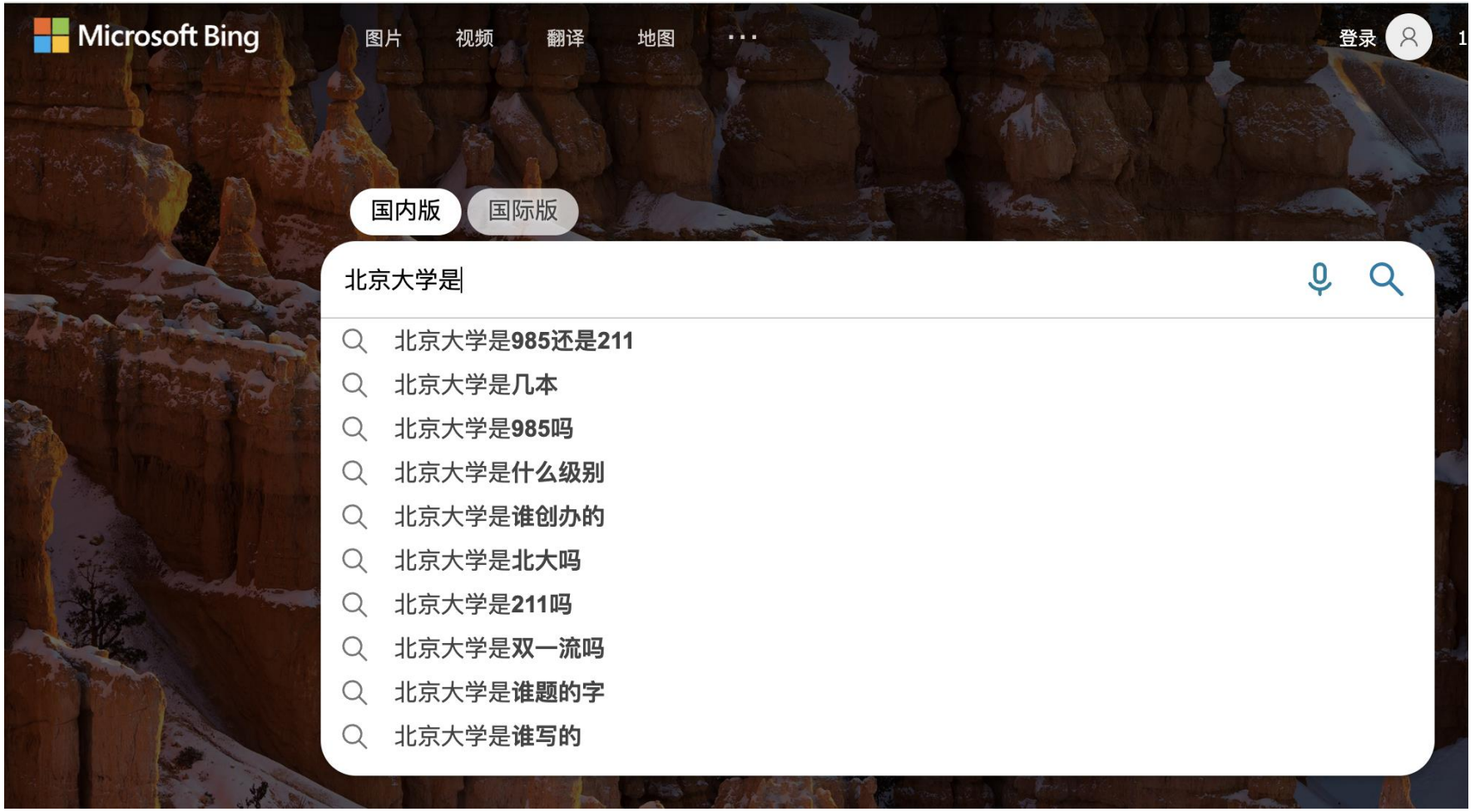
You use language models everyday

Hi Stefan,

Thanks very much for your email

Sans Serif

You use language models everyday



n-gram language model

the students opened their _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - four-grams: “the students opened their”

- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

n-gram language model

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding *n*-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

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 Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _____

get probability distribution

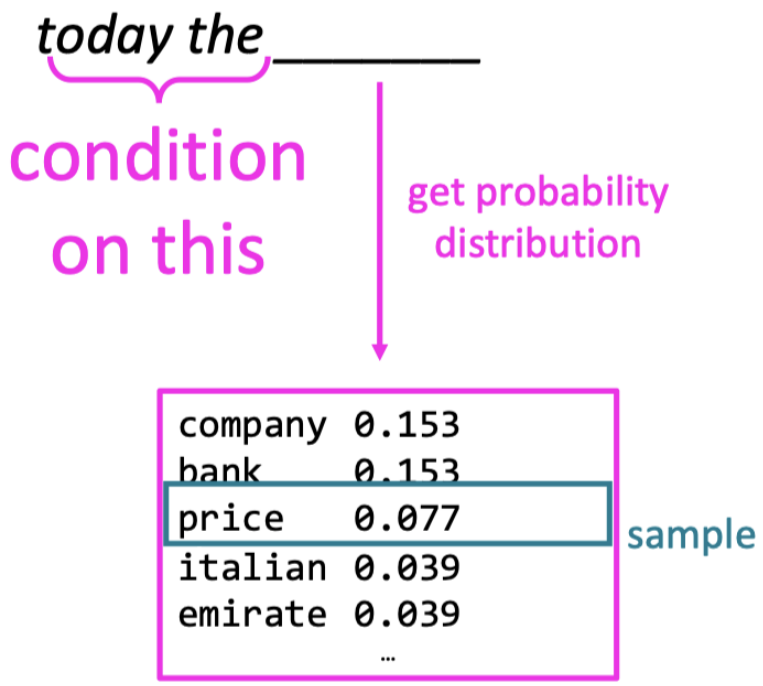
company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

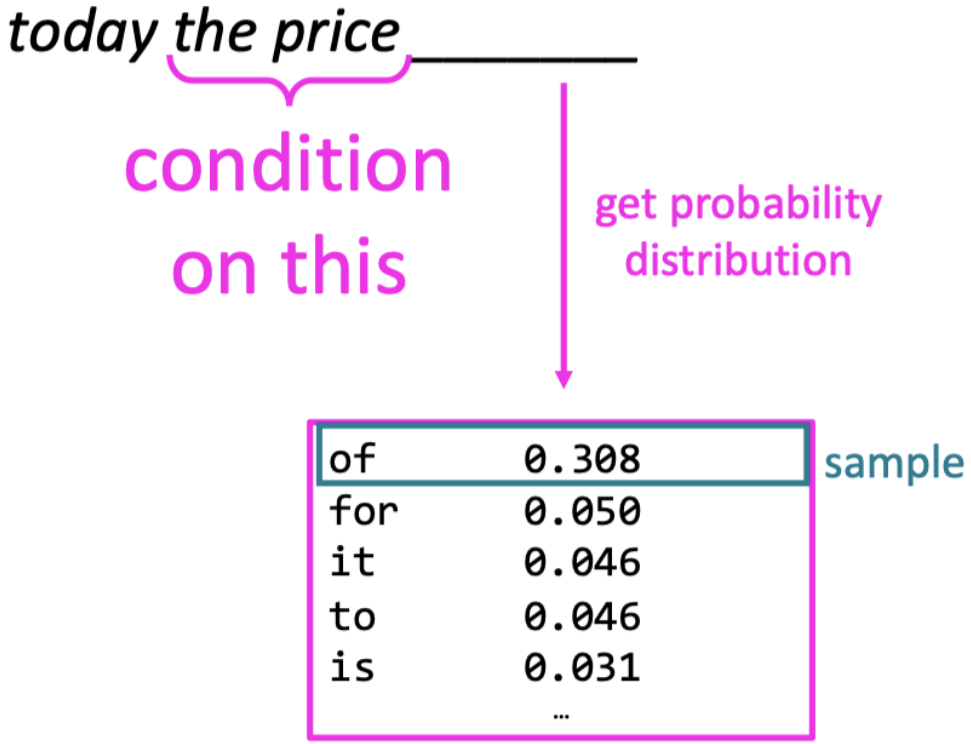
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



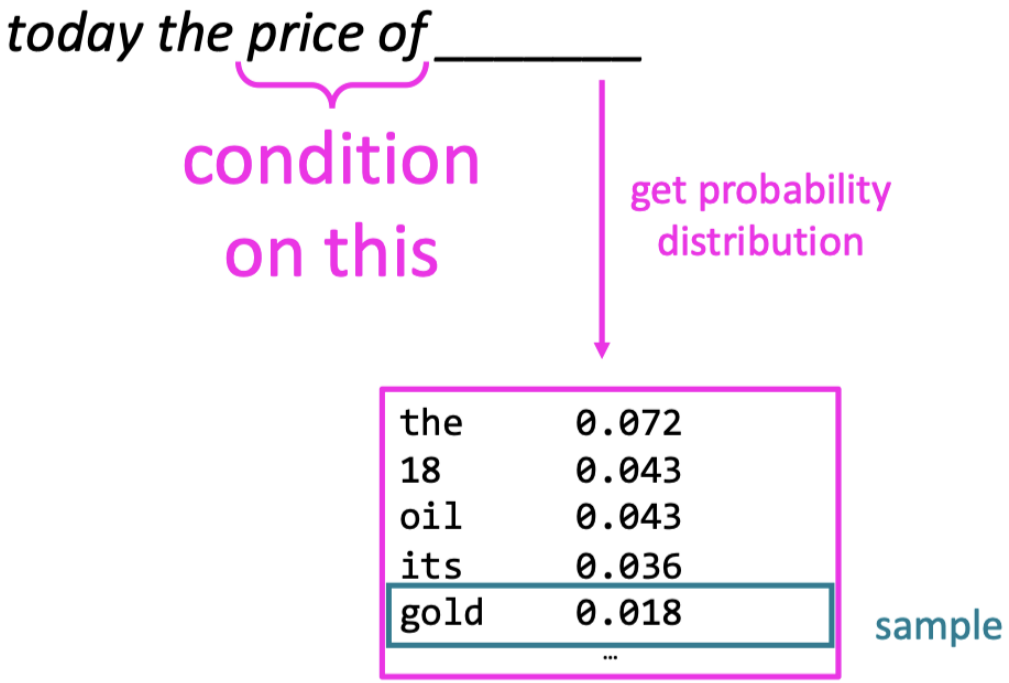
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Generating text with a n-gram Language Model

北京大学 PEKING UNIVERSITY

You can also use a Language Model to generate text

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 .

Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size...

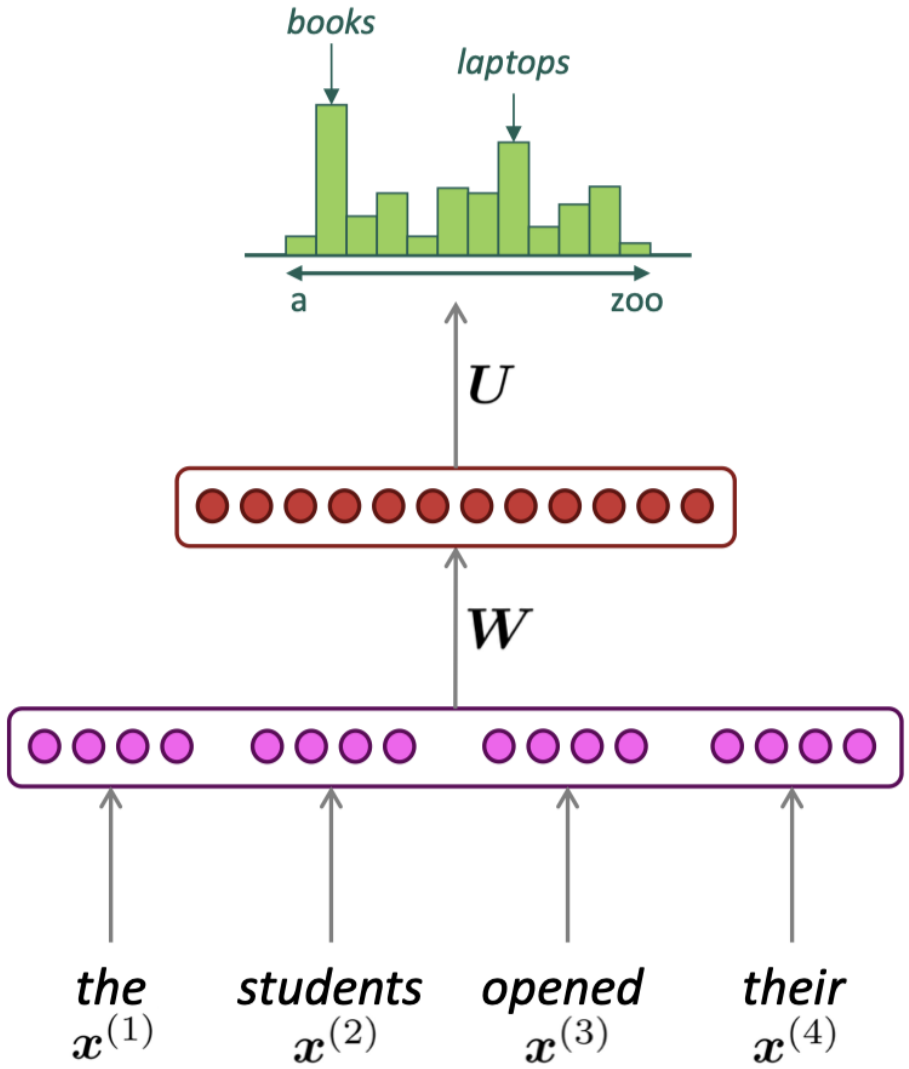
Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

A neural language model



output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

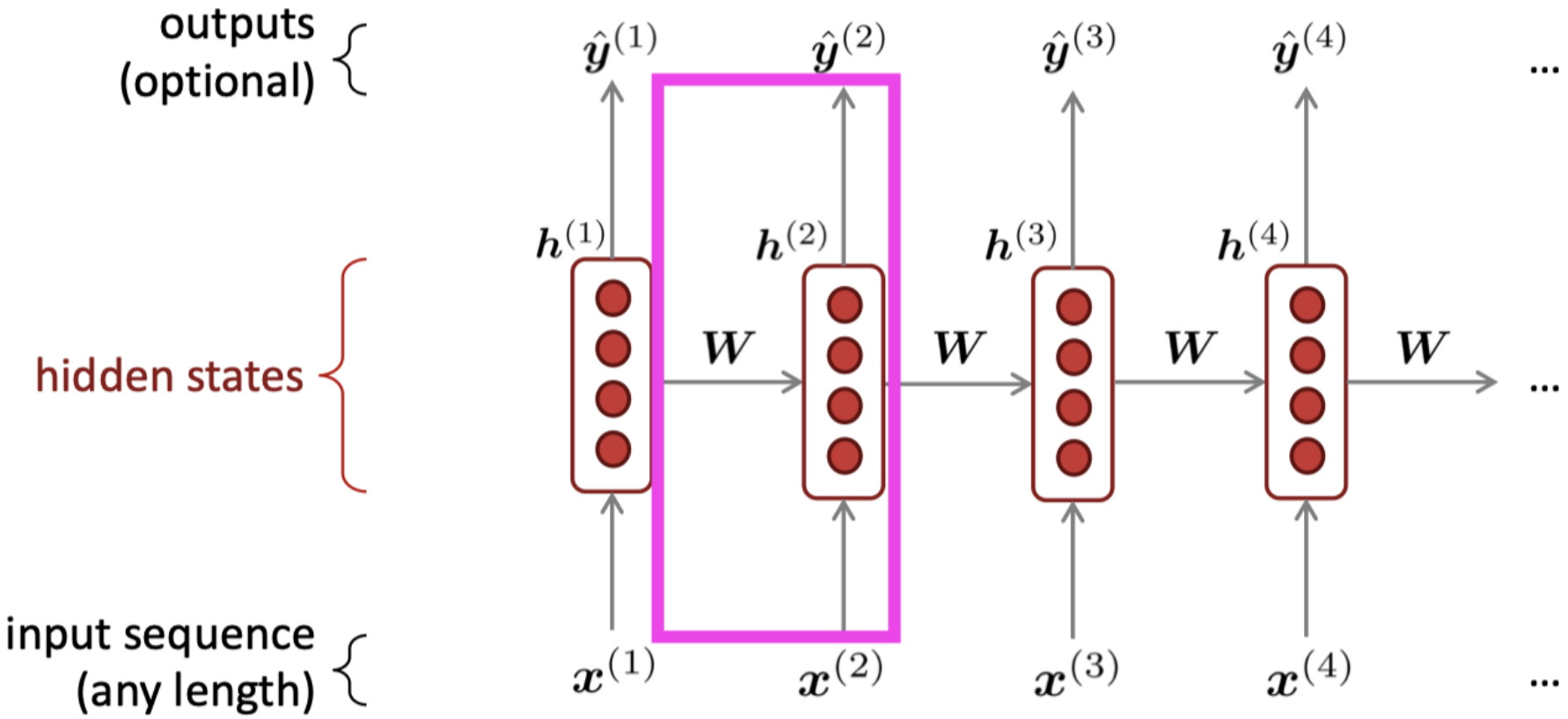
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

A neural language model with a fixed window length

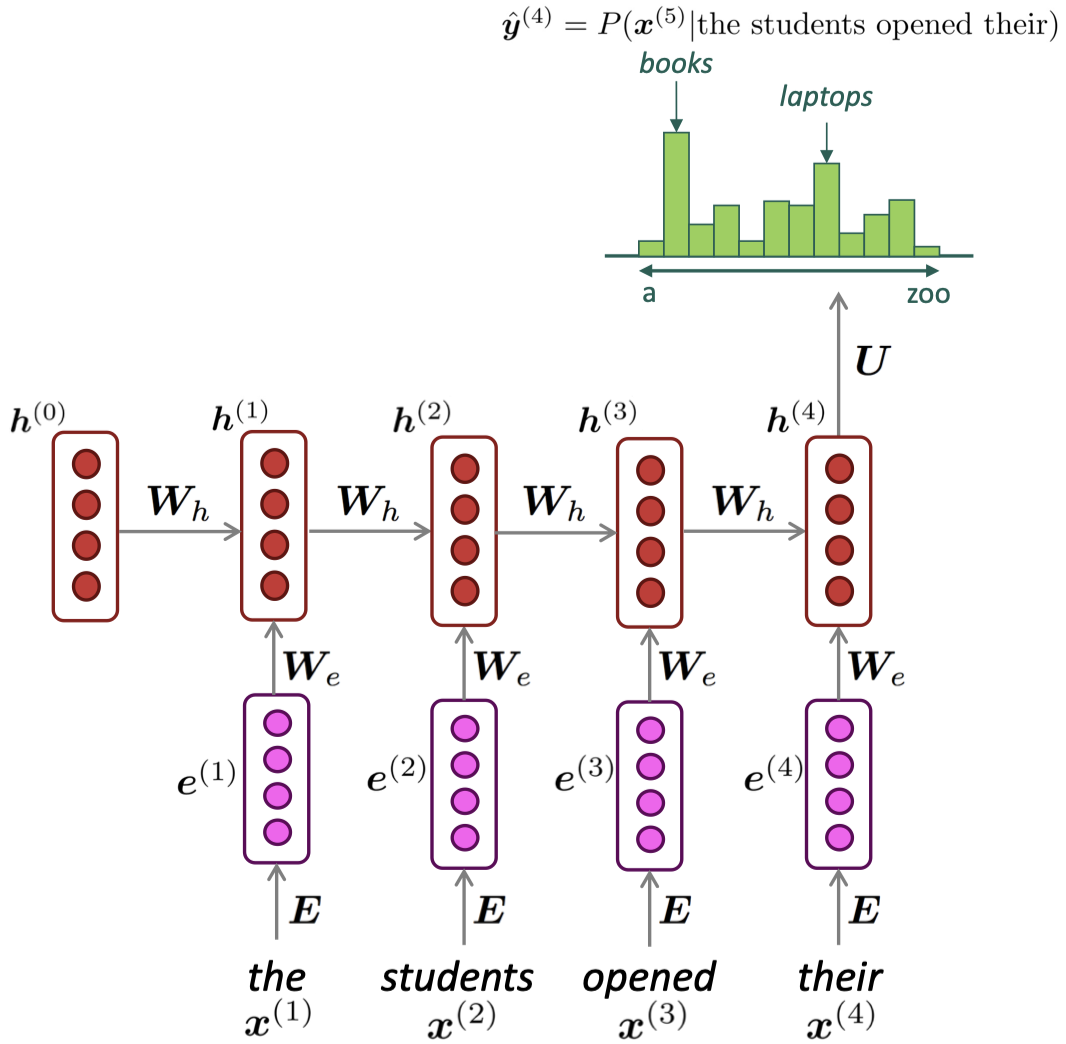
When the window size is large, the model and memory costs will get large

Typically, the window size is too small; cannot reflect long-term dependency

Recurrent neural networks (RNN)



Recurrent neural networks (RNN)



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

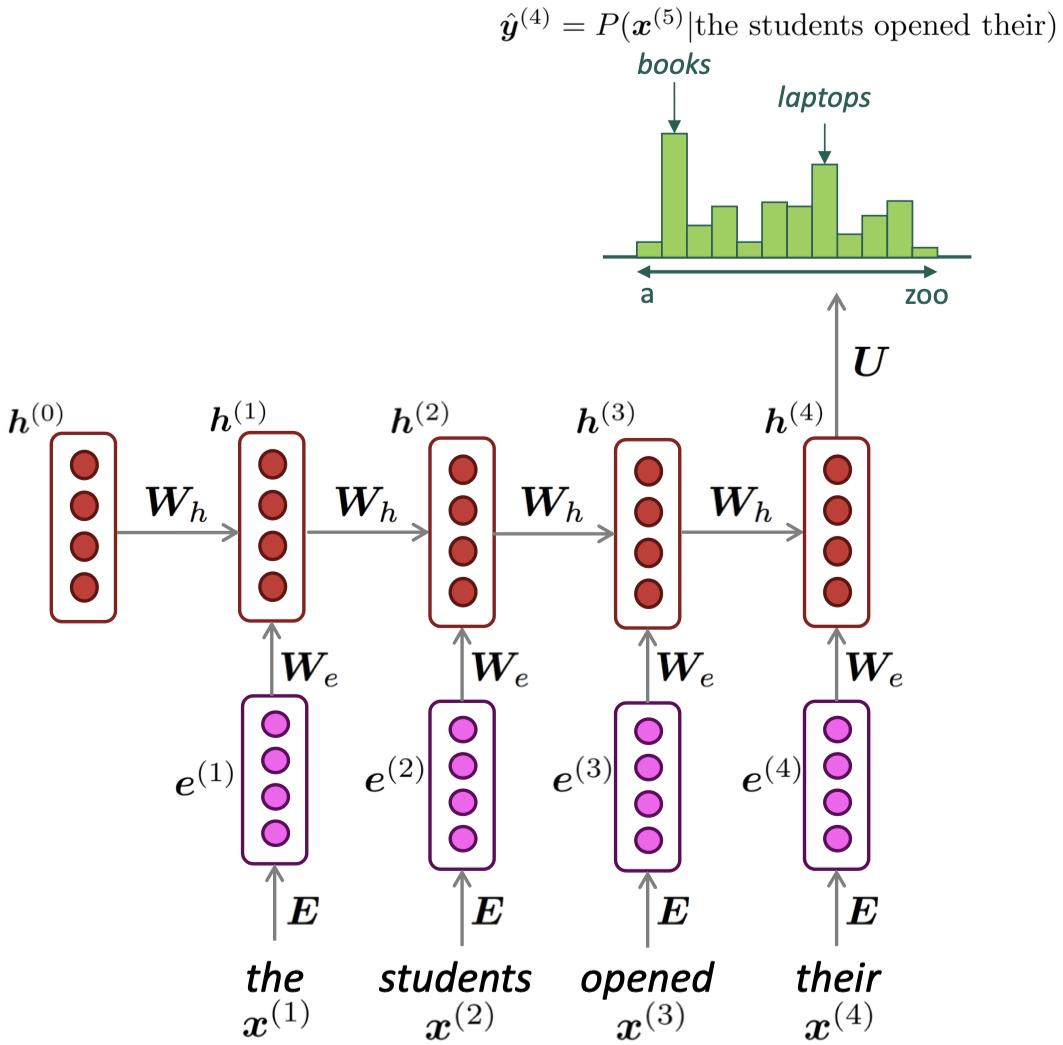
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

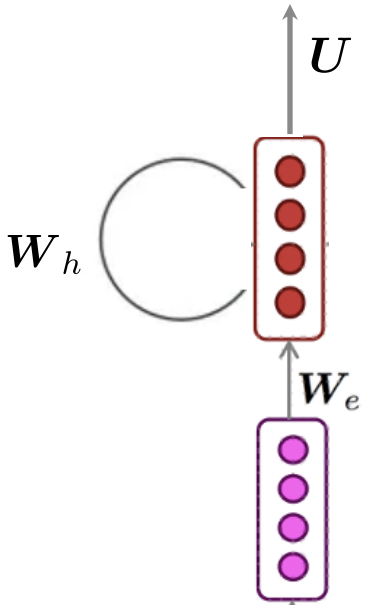
words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

More concise representations



Model weights: $\{\boldsymbol{W}_h, \boldsymbol{W}_e, \boldsymbol{U}, b_1, b_2\}$

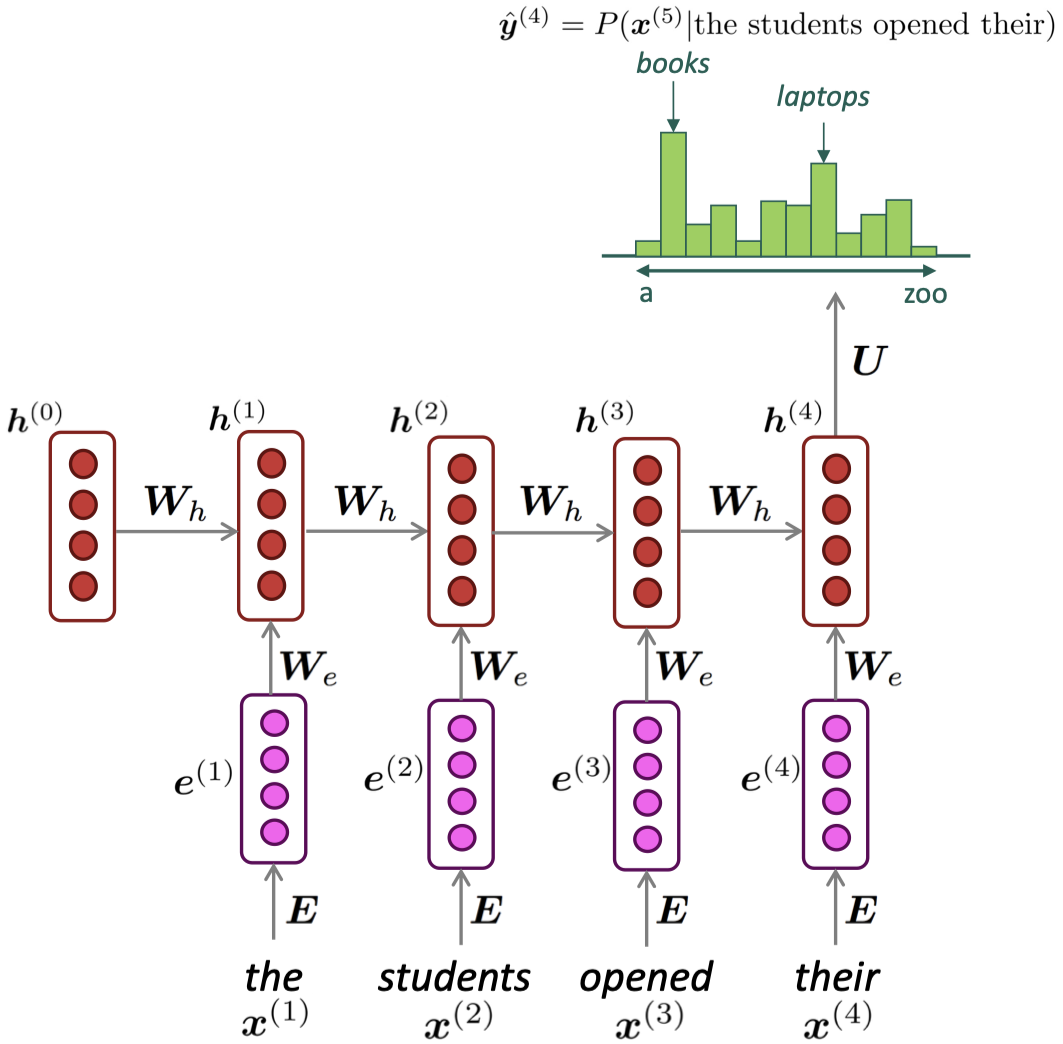


$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

RNN benefits



RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context

Training an RNN language model

- Get a big corpus of text which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for *every step t*.
  - i.e., predict probability dist of *every word*, given words so far

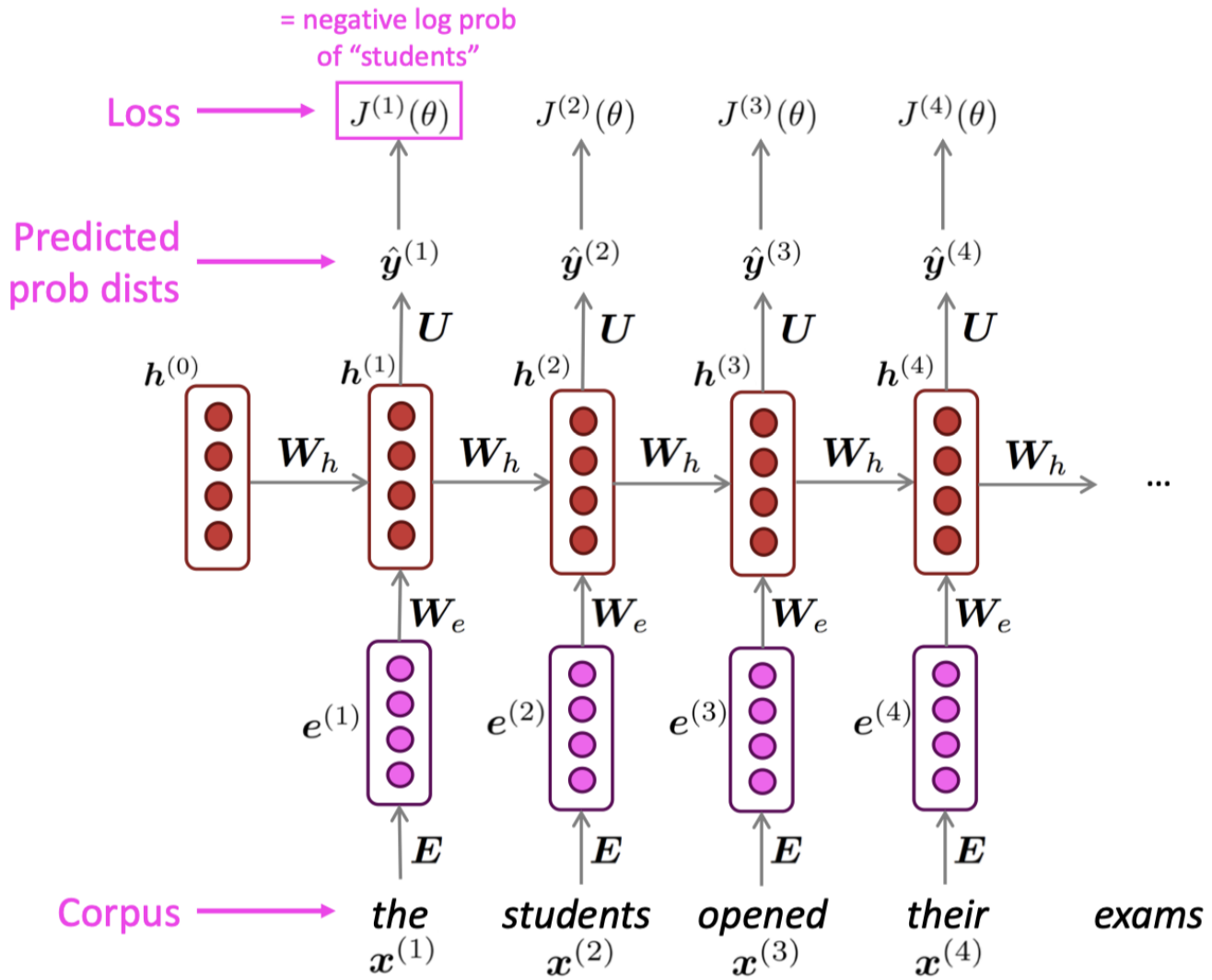
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

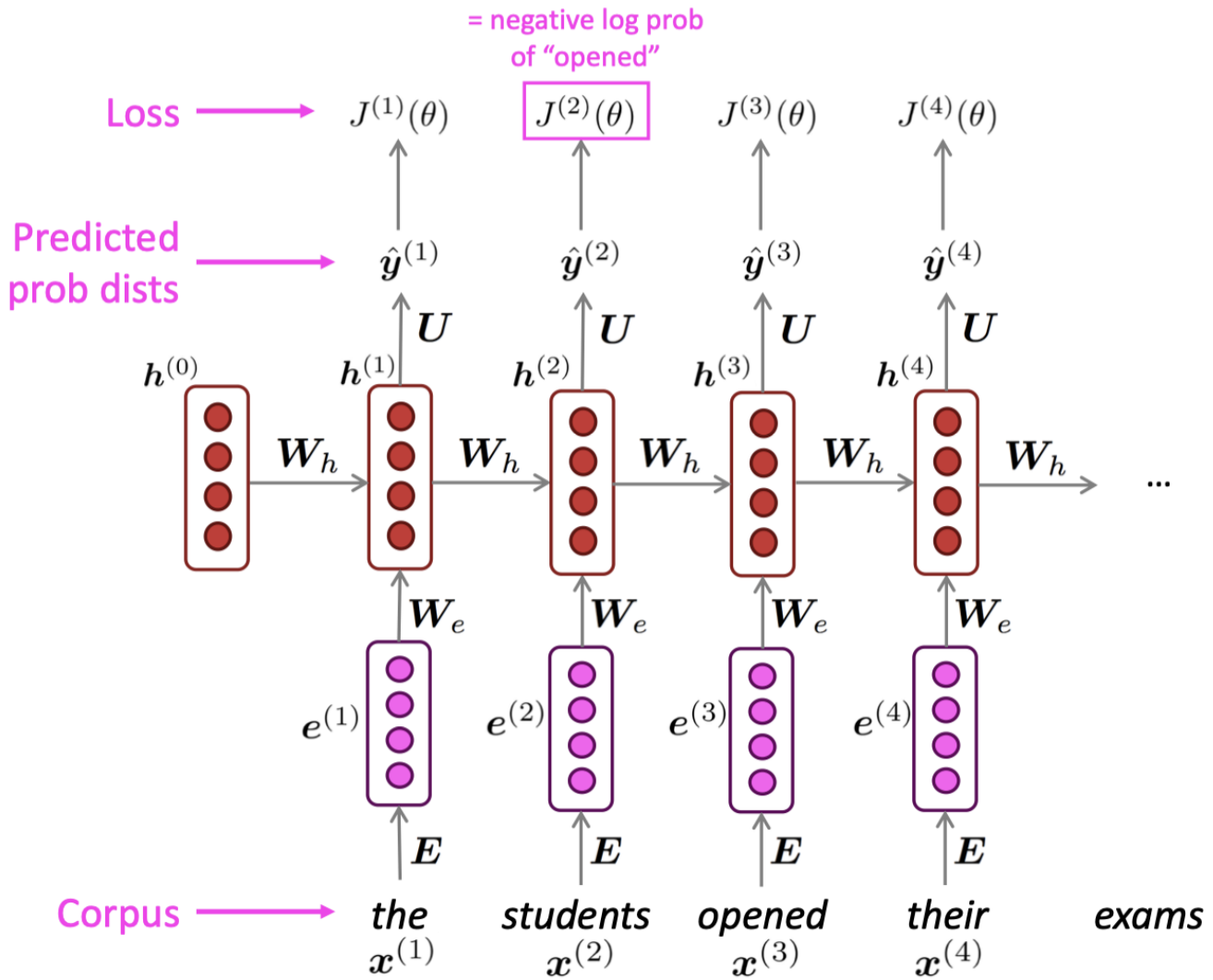
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

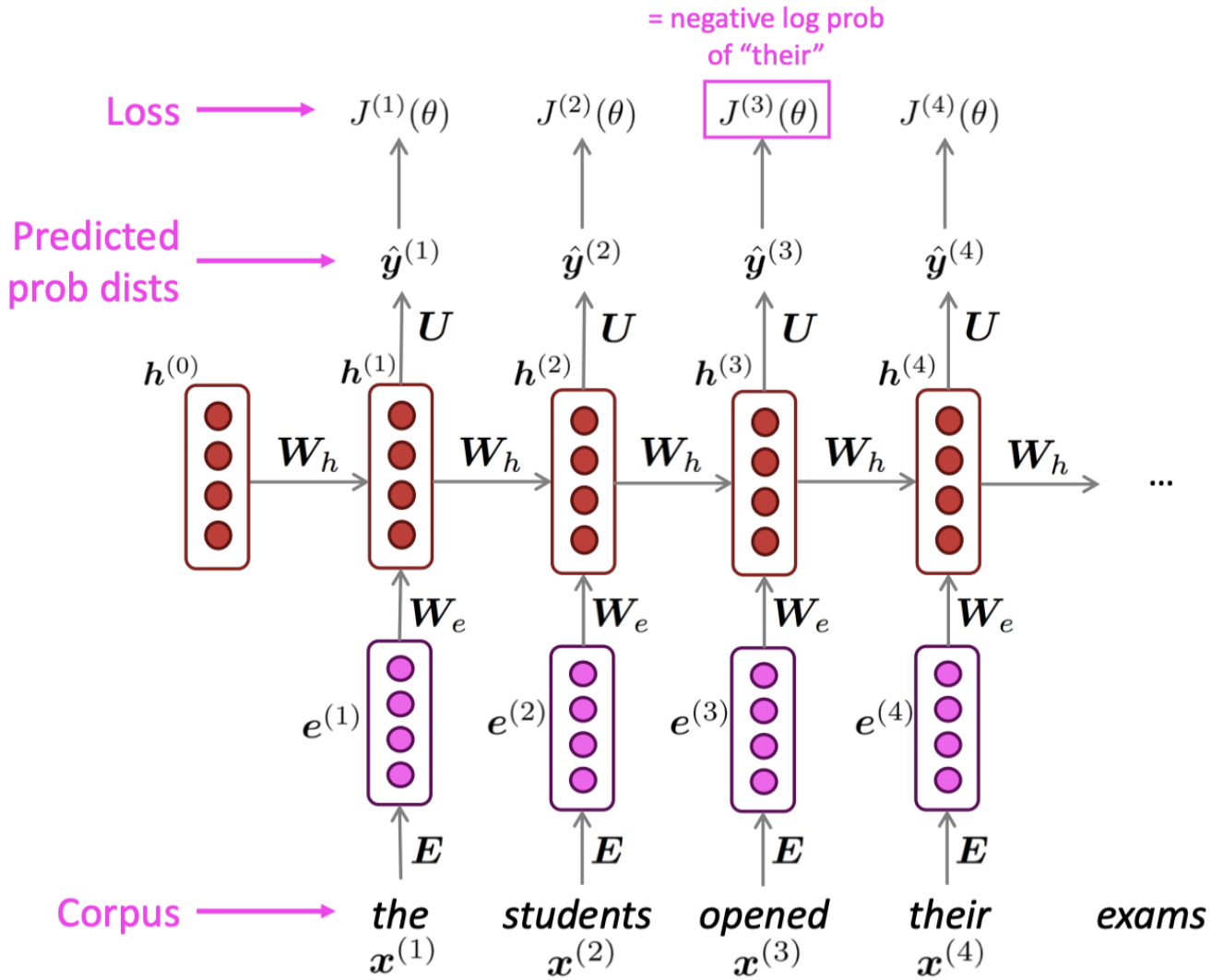
Training an RNN language model



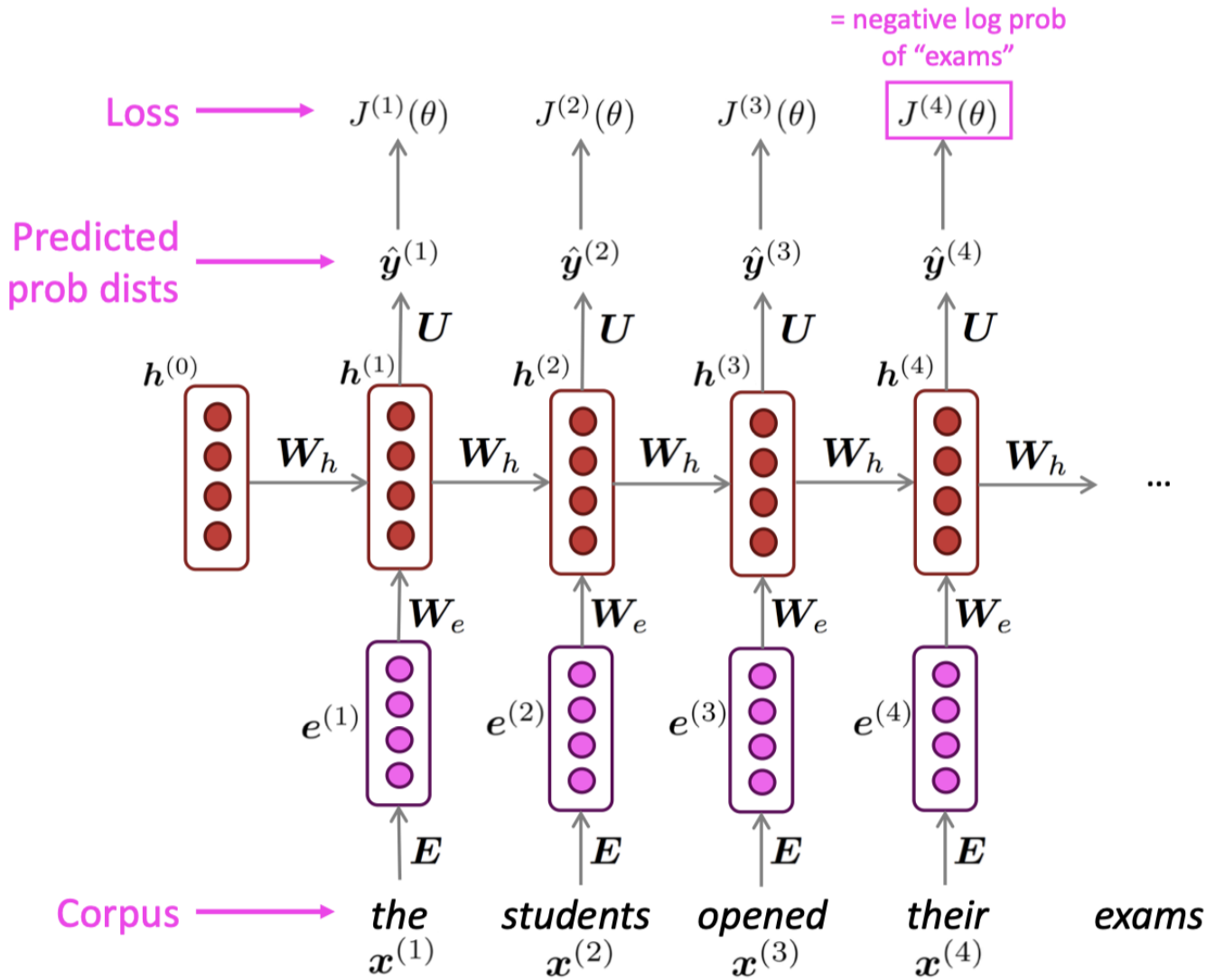
Training an RNN language model



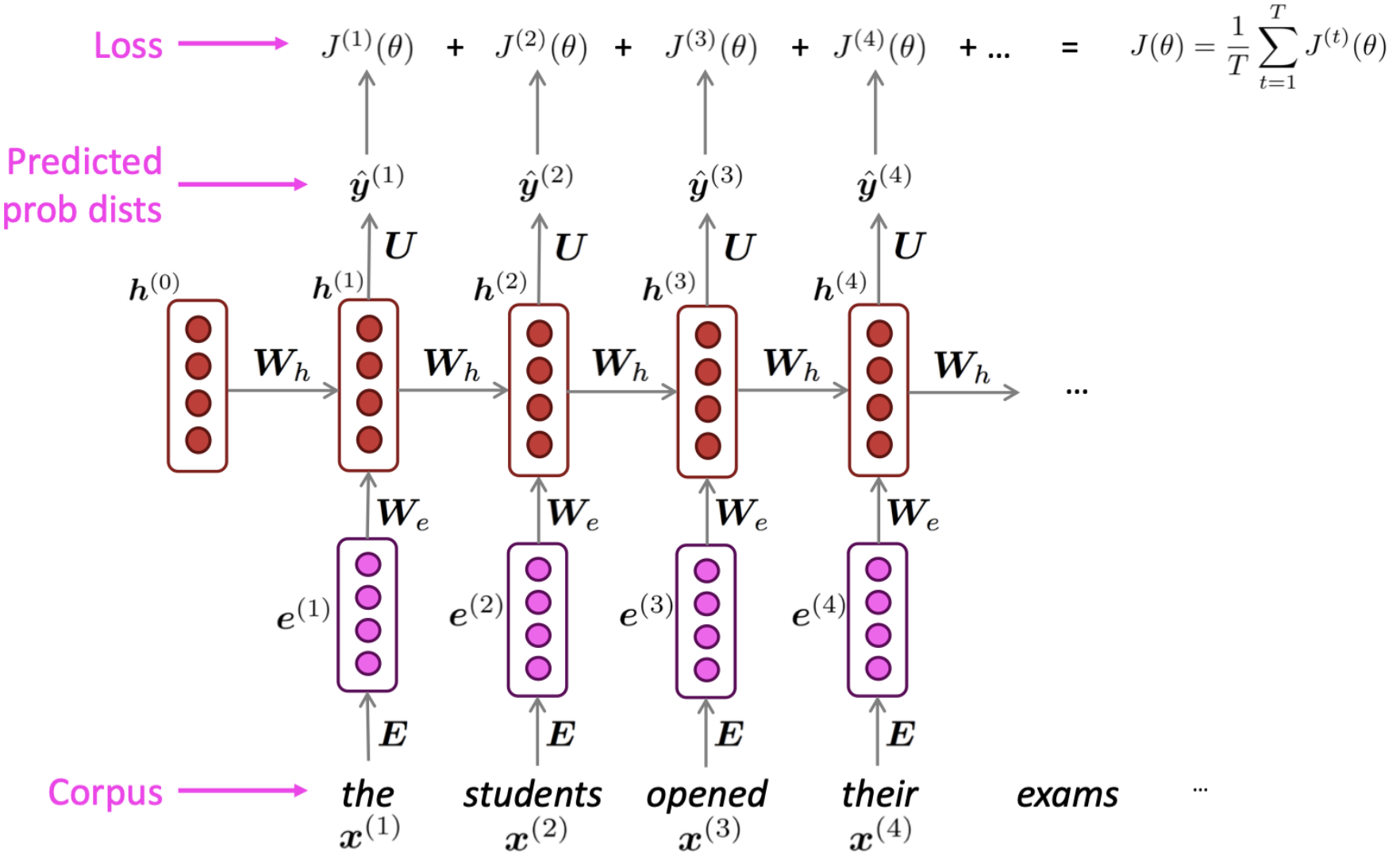
Training an RNN language model



Training an RNN language model



Training an RNN language model



Training an RNN language model

北京大学 PEKING UNIVERSITY

- However: Computing loss and gradients across entire corpus $x^{(1)}, \ldots, x^{(T)}$ at once is too expensive (memory-wise)!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $x^{(1)}, \ldots, x^{(T)}$ as a sentence (or a document)

- Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

A batch of sentences

The inputs to the Transformer are arrays of shape (B,T)

- B is the batch size (e.g. 4 here)
- T is the maximum context length (e.g. 10 here)

Training sequences are laid out as rows, delimited by special <|endoftext|> tokens

Row 1: Here is an example document 1 showing some tokens.
Row 2: Example document 2<|endoftext|>Example document 3<|endoftext|>Example document
Row 3: This is some random text just for example<|endoftext|>This
Row 4: 1,2,3,4,5

One training batch, array of shape (B,T)

T = 10

B = 4

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

Generating text with an RNN Language Model

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

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.

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

Generating text with an RNN Language Model

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Generating text with an RNN Language Model

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:

```
      Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
      Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

Training RNN is typically challenging

Gradient is either vanishing or exploding!

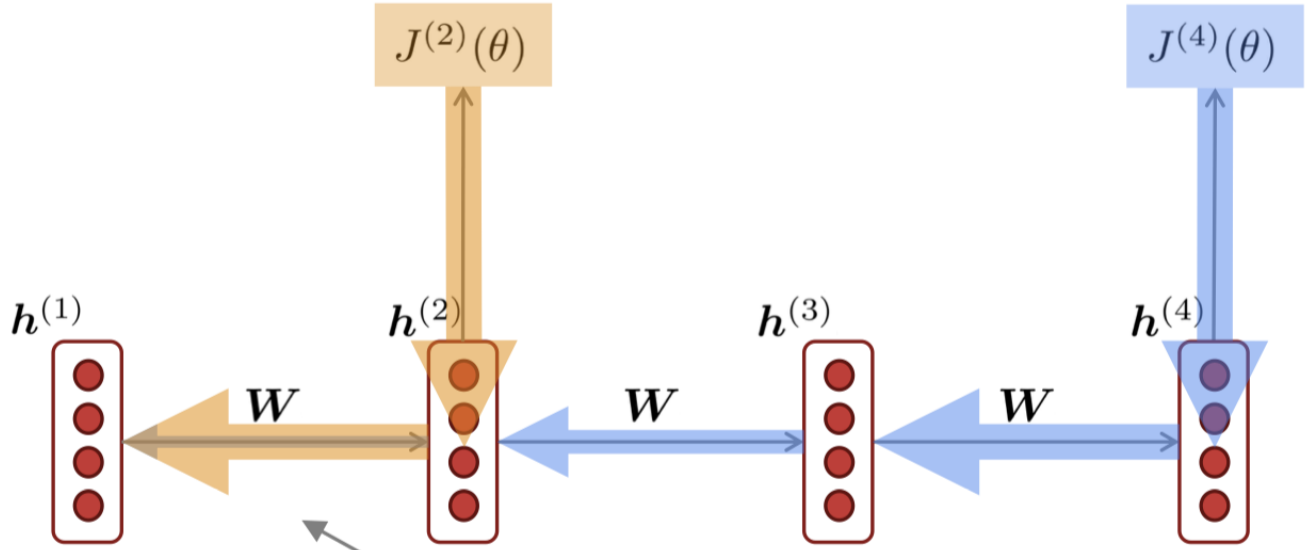
We will check this phenomenon in a different set of slides

Several ways to fix this issue: **residual connections**, **LSTMs**, **Attentions, etc.**

Training RNN is typically challenging

北京大学 PEKING UNIVERSITY

Vanishing gradient



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

Effect of vanishing gradient on RNN-LM

北京大学 PEKING UNIVERSITY

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if the gradient is small, the model can't learn this dependency
  - So, the model is unable to predict similar long-distance dependencies at test time

Recap

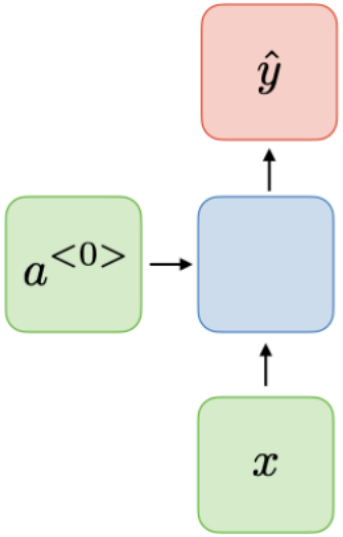
- **Language Model**: A system that predicts the next word

- **Recurrent Neural Network**: A family of neural networks that:
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step

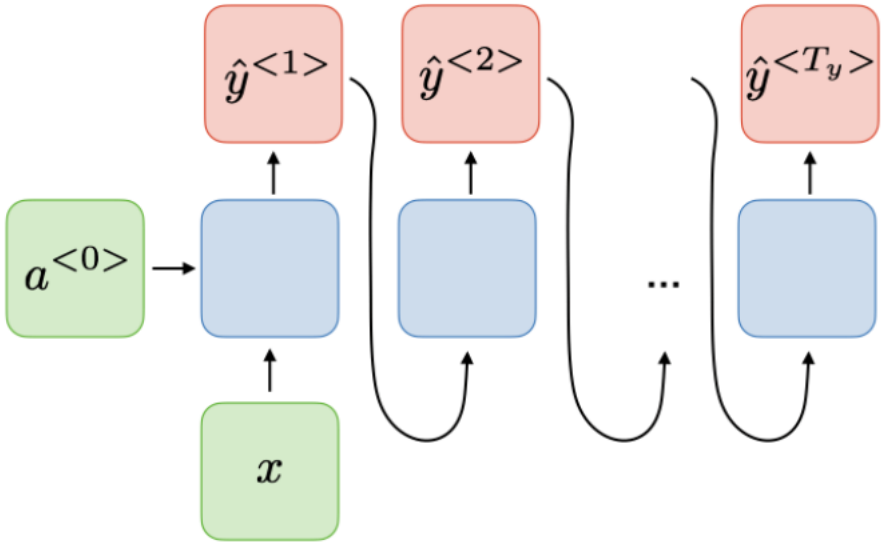
- We've shown that RNNs are a great way to build a LM (despite some problems)

- RNNs are also useful for much more!

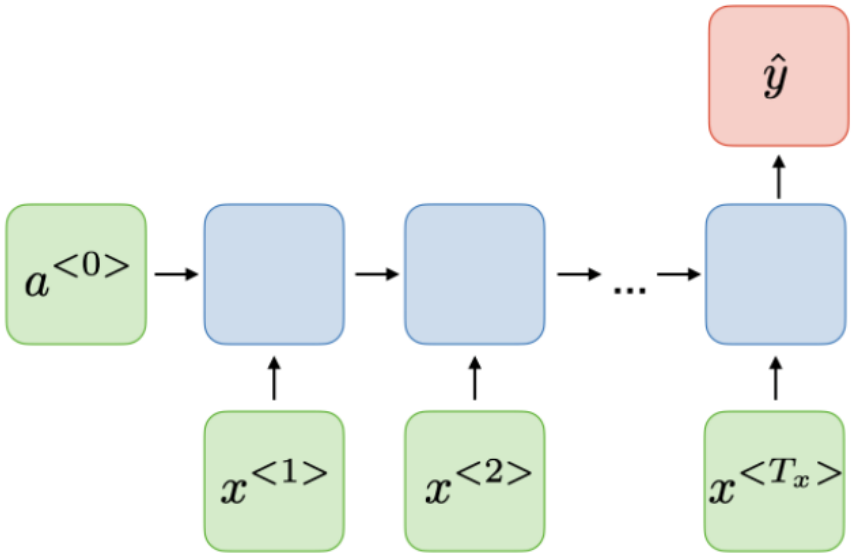
RNN

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network

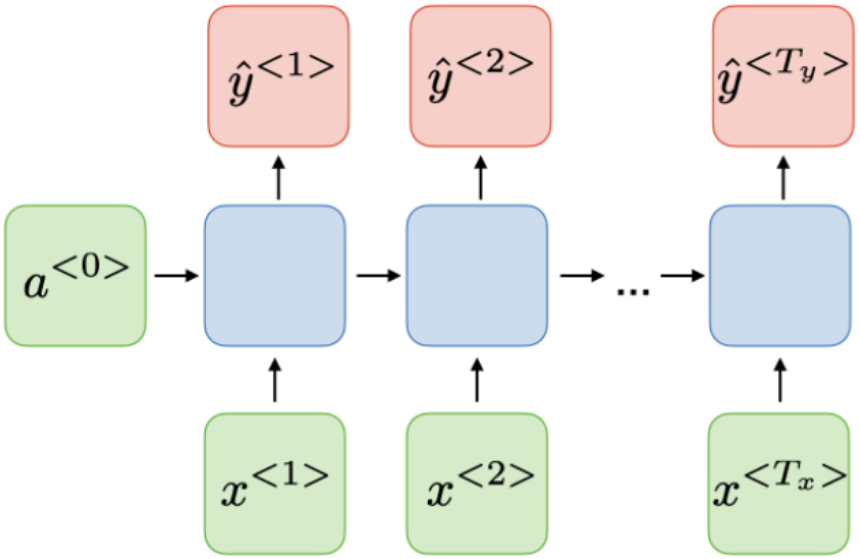
RNN

One-to-many $T_x = 1, T_y > 1$		Music generation

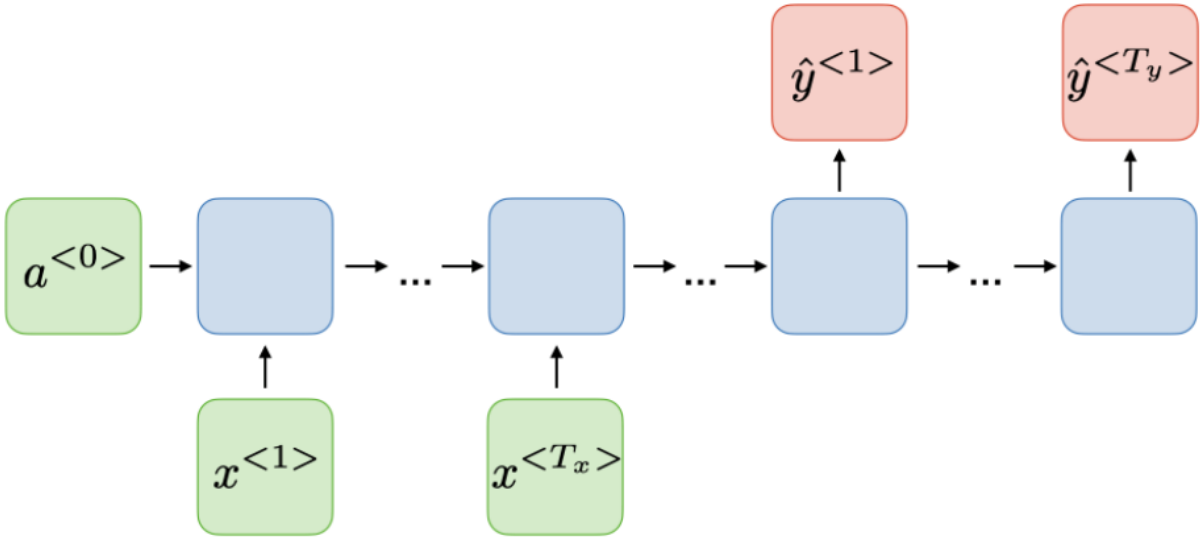
RNN

Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

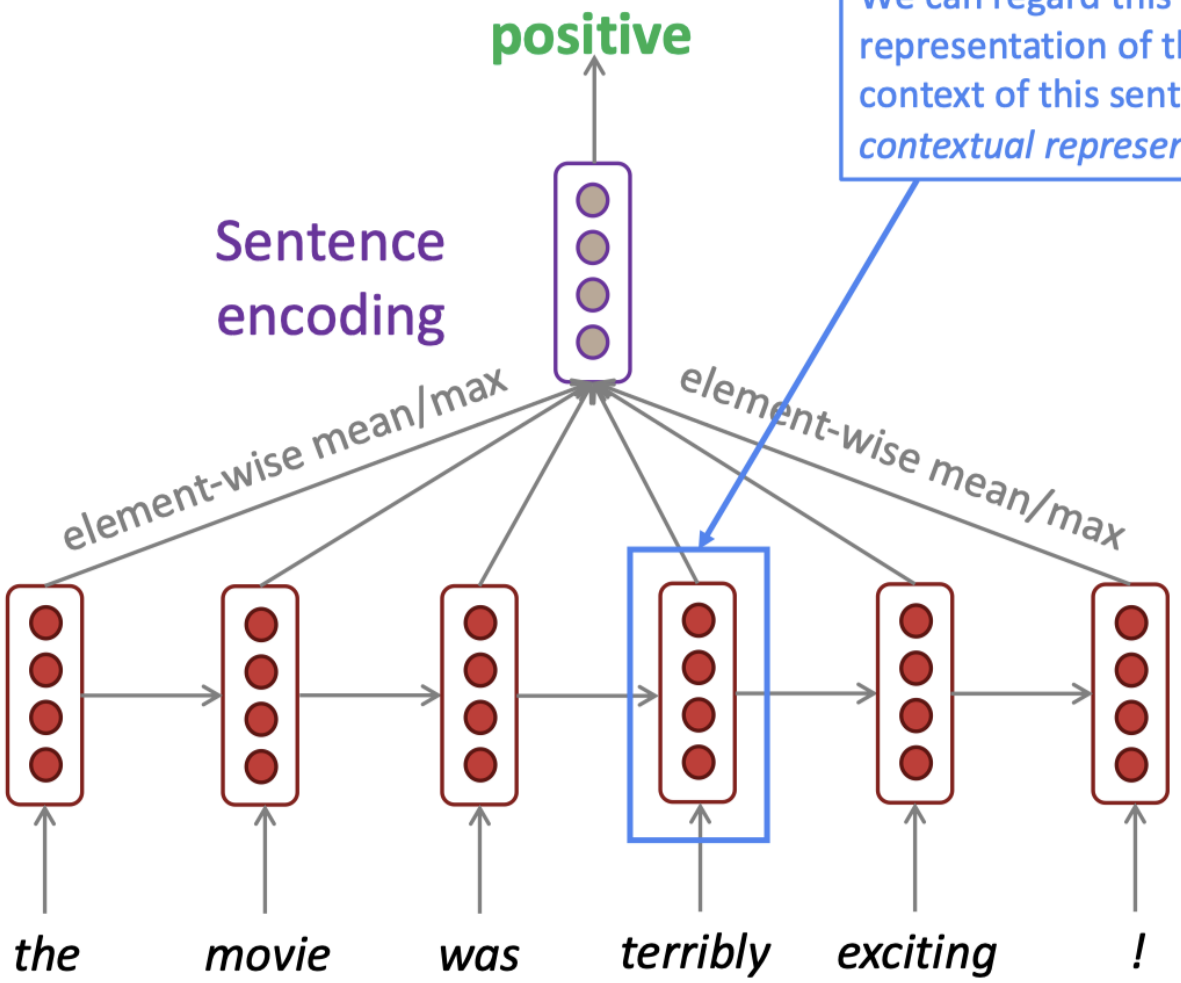
RNN

Many-to-many $T_x = T_y$		Name entity recognition

RNN

Many-to-many $T_x \neq T_y$		Machine translation

Bidirectional RNN



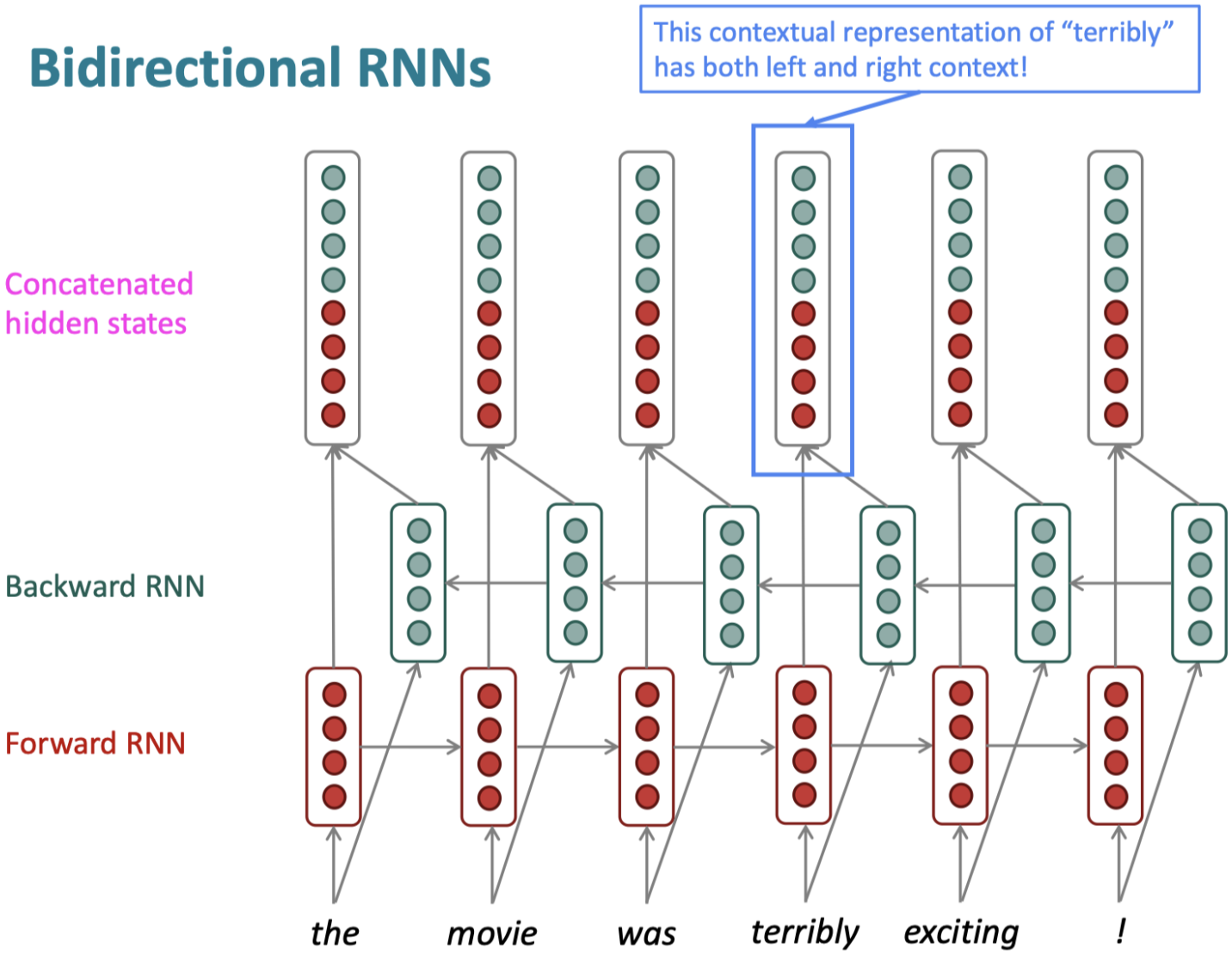
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

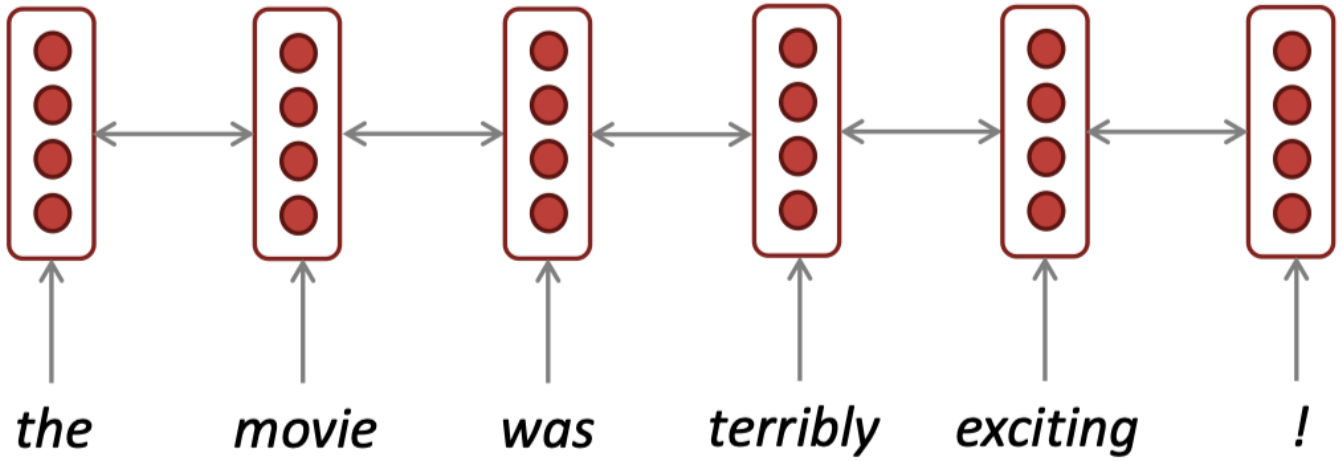
Bidirectional RNN



Bidirectional RNN

北京大学 PEKING UNIVERSITY

Simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

Bidirectional RNN

北京大学 PEKING UNIVERSITY

- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.

- If you do have entire input sequence (e.g., any kind of encoding), bidirectionality is powerful (you should use it by default).

- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.
  - You will learn more about transformers, including BERT, in a couple of weeks!

Muti-layer RNNs

北京大学 PEKING UNIVERSITY

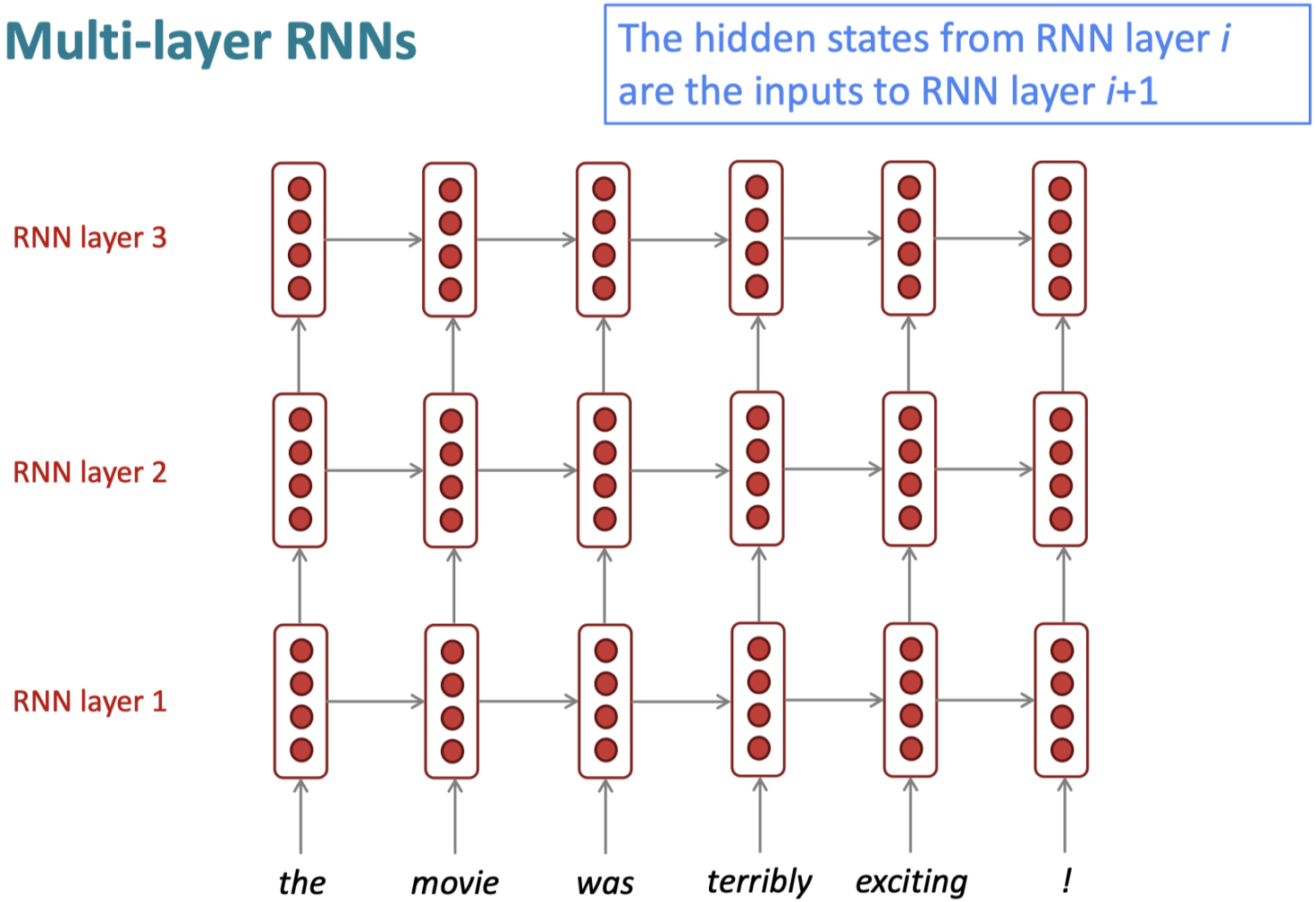
- RNNs are already “deep” on one dimension (they unroll over many timesteps)

- We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.

- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer RNNs



Multi-layer RNNs in practice

北京大学 PEKING UNIVERSITY

- Multi-layer or stacked RNNs allow a network to compute more complex representations
  - they work better than just have one layer of high-dimensional encodings!
    - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

- High-performing RNNs are usually multi-layer (but aren't as deep as convolutional or feed-forward networks)

- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)

- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections