language modeling: neural language models

CS 585, Fall 2018

Introduction to Natural Language Processing

http://people.cs.umass.edu/~miyyer/cs585/

Mohit Iyyer

College of Information and Computer Sciences

University of Massachusetts Amherst

many slides from Richard Socher and Matt Peters

language model review

- Goal: compute the probability of a sentence or sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

- Related task: probability of an upcoming word:

  $P(w_5|w_1,w_2,w_3,w_4)$

- A model that computes either of these:

  $P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model** or **LM**

$$p(w_j \mid \text{students opened their}) = \frac{\text{count(students opened their } w_j\text{)}}{\text{count(students opened their)}}$$

what is the order of this n-gram model? (i.e., what is n?)

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their $\boldsymbol{w}_j$"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

$$p(w_j \mid \text{students opened their}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}$$

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}_j$*"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

$$p(w_j \mid \text{students opened their}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}$$

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}_j$*"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}_j$!

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}_j$*"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}_j$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}_j$*"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}_j$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse.* Typically we can't have *n* bigger than 5.

Problems with n-gram Language Models

Storage: Need to store count for all possible *n*-grams. So model size is O(exp(*n*)).

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

Increasing *n* makes model size huge!

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!

*** Try for yourself:** https://nlpforhackers.io/language-models/

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the ________

condition on this

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today $\underbrace{\textit{the price}}_{\text{condition on this}}$ ________

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Generating text with a n-gram Language Model

- 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 .

Incoherent! We need to consider more than 3 words at a time if we want to generate good text.

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

How to build a *neural* Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?

similar to the deep averaging network
we saw for text classification!

A fixed-window neural Language Model

~~*as the proctor started the clock*~~ *the students opened their* ______

discard

$\underbrace{\textit{the students opened their}}_{\text{fixed window}}$

A fixed-window neural Language Model

output distribution

$$\hat{y} = \text{softmax}(W_2 h + b_2)$$

hidden layer

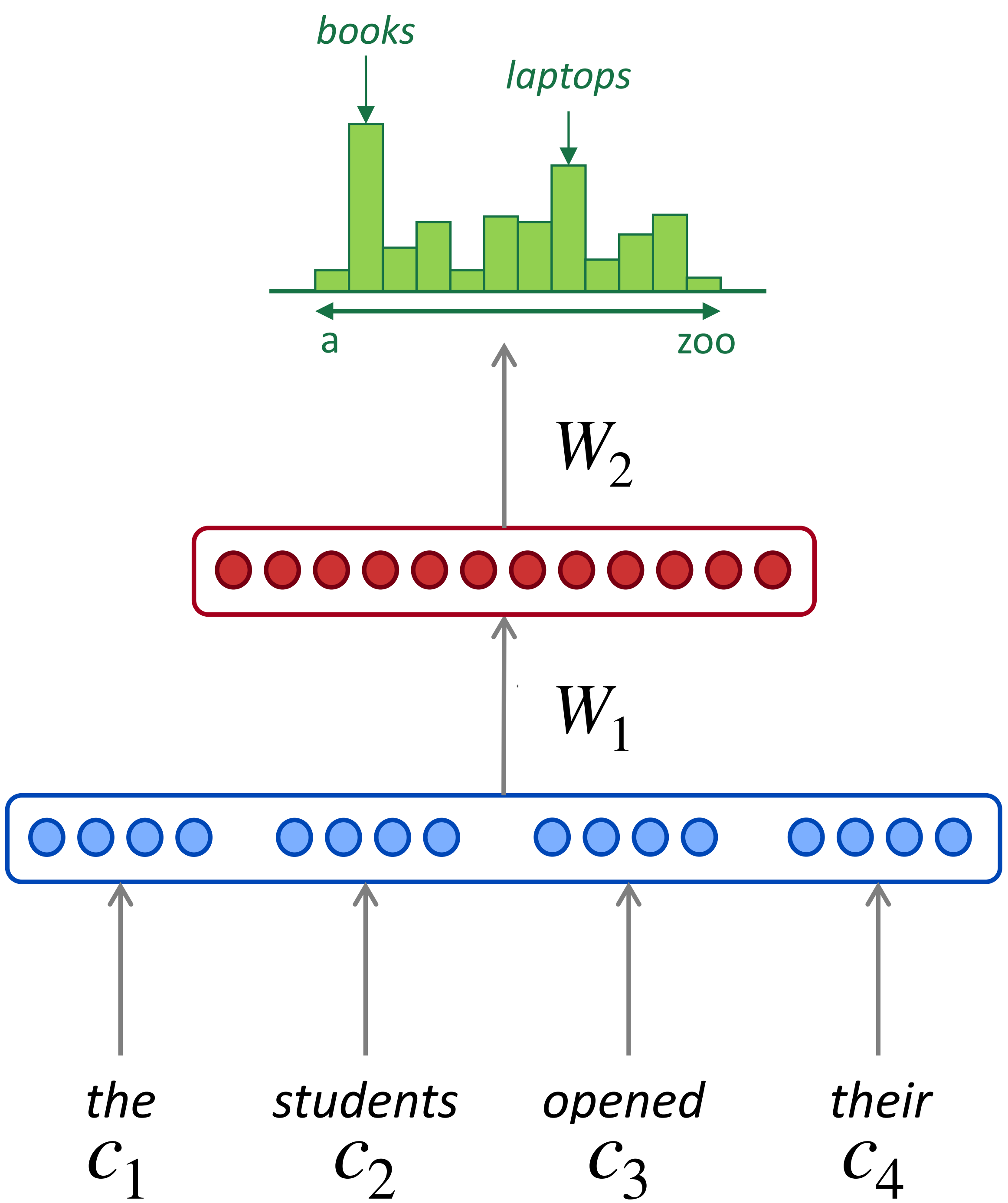
$$h = f(W_1 c + b_1)$$

concatenated word embeddings

$$c = [c_1; c_2; c_3; c_4]$$

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



A fixed-window neural Language Model

how does this differ from a DAN?

output distribution

$$\hat{y} = \mathrm{softmax}(W_2 h + b_2)$$

hidden layer

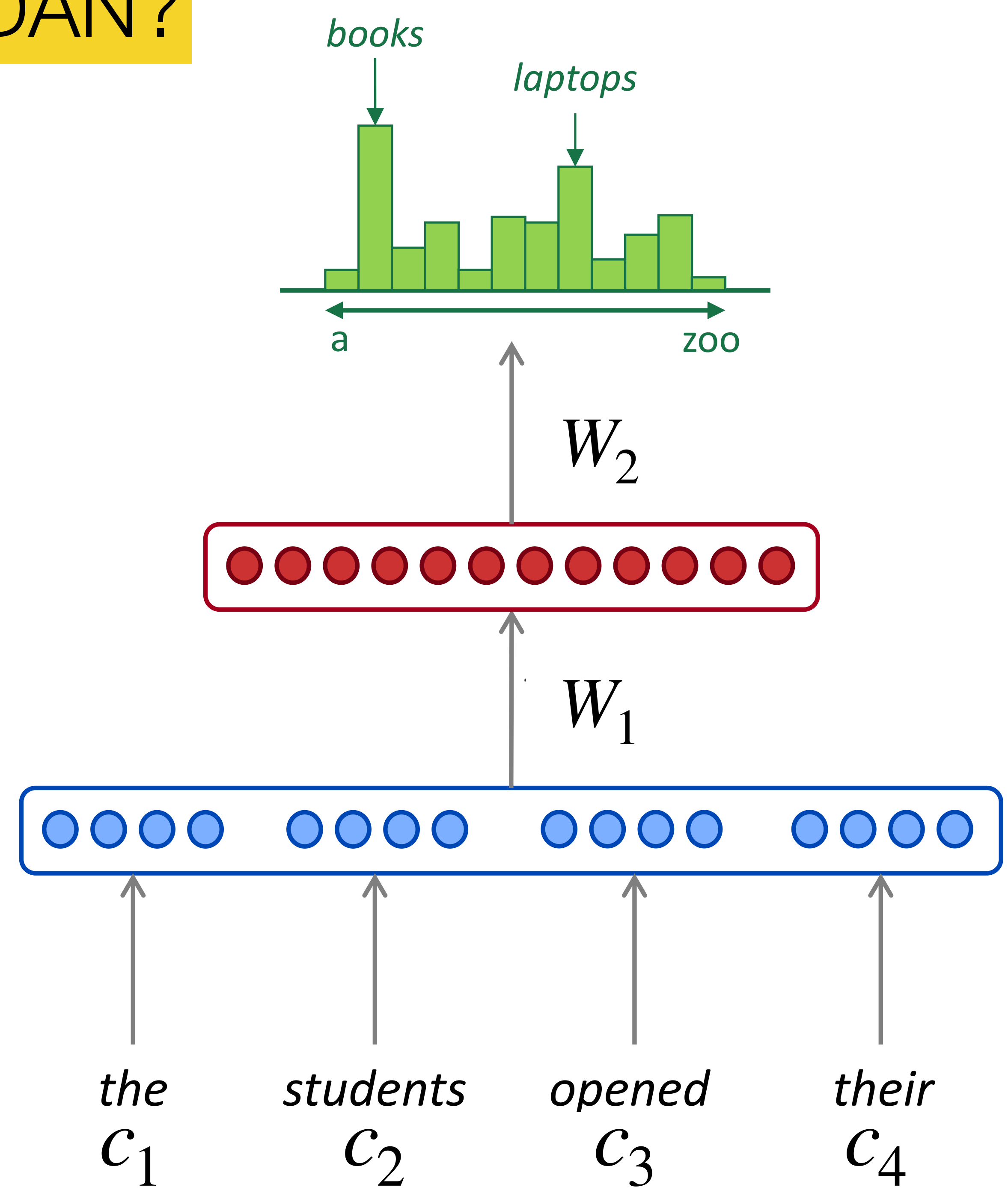
$$h = f(W_1 c + b_1)$$

concatenated word embeddings

$$c = [c_1; c_2; c_3; c_4]$$

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



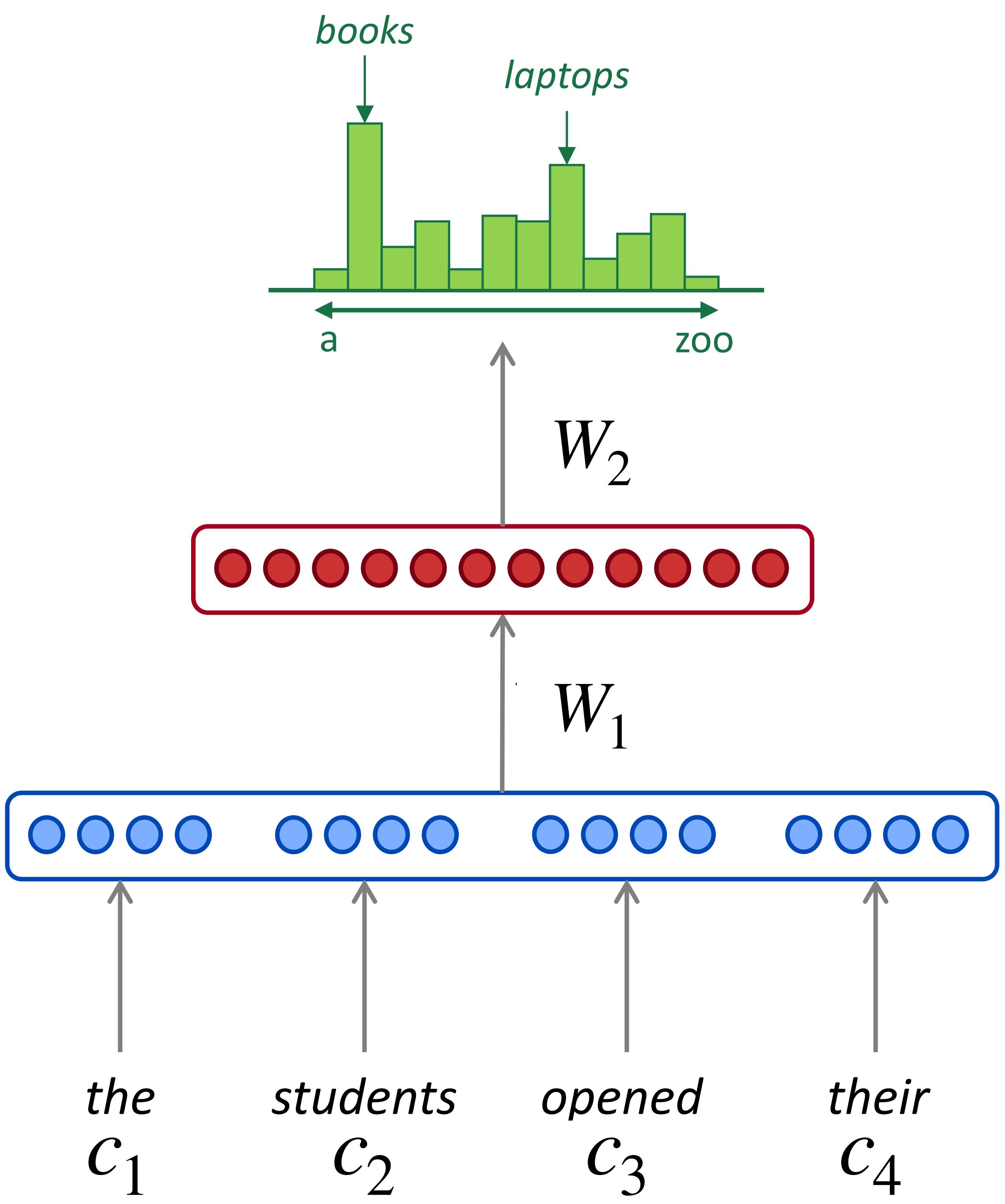
how does this compare to a normal n-gram model?

Improvements over *n*-gram LM:

- No sparsity problem
- Model size is O(*n*) not O(exp(*n*))

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- Each c_i uses different rows of $\boldsymbol{W}$. We don't share weights across the window.



Recurrent Neural Networks!

A RNN Language Model

output distribution

$$\hat{y} = \text{softmax}(W_2 h^{(t)} + b_2)$$

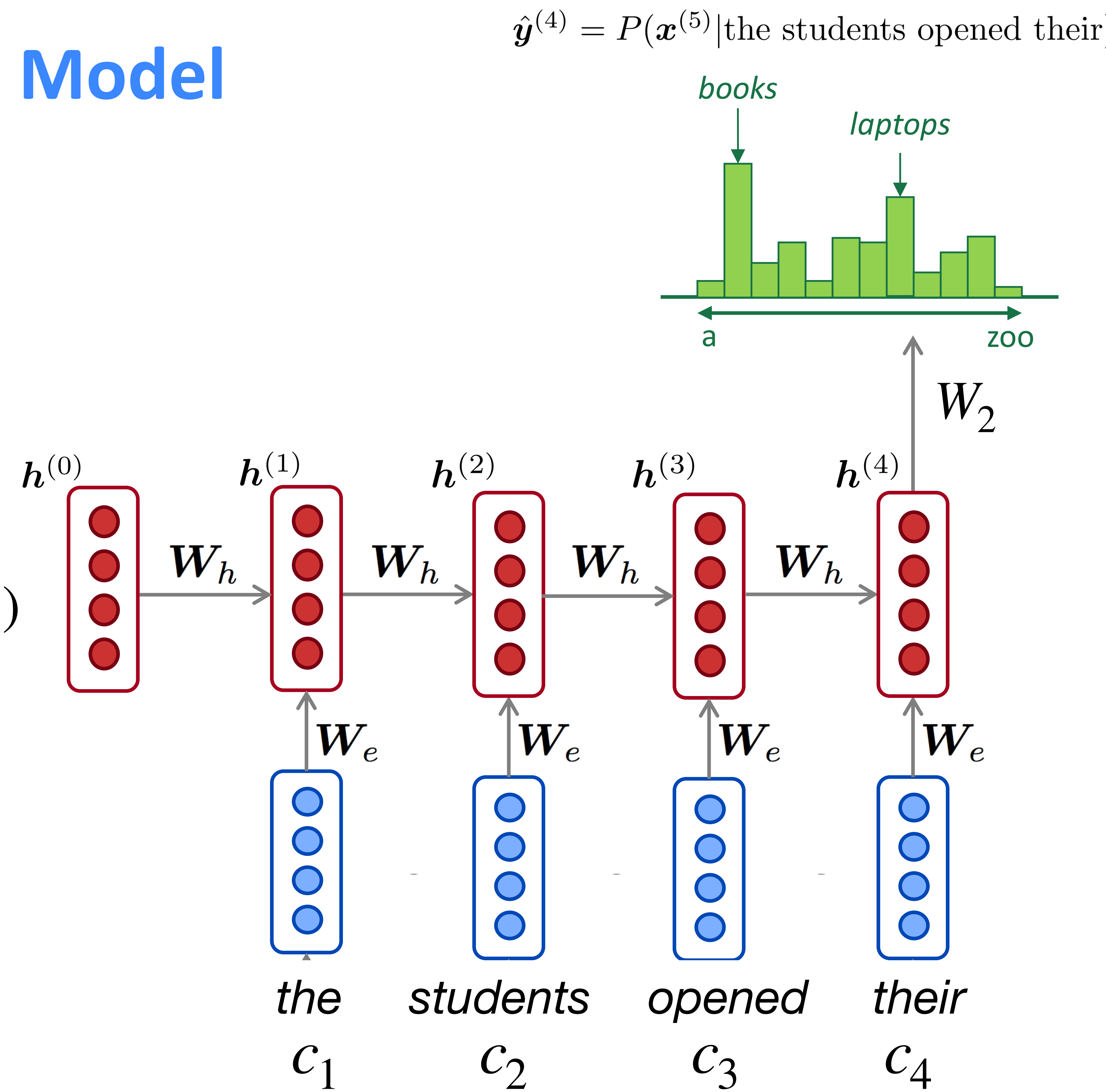
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t + b_1)$$

$h^{(0)}$ is initial hidden state!

word embeddings

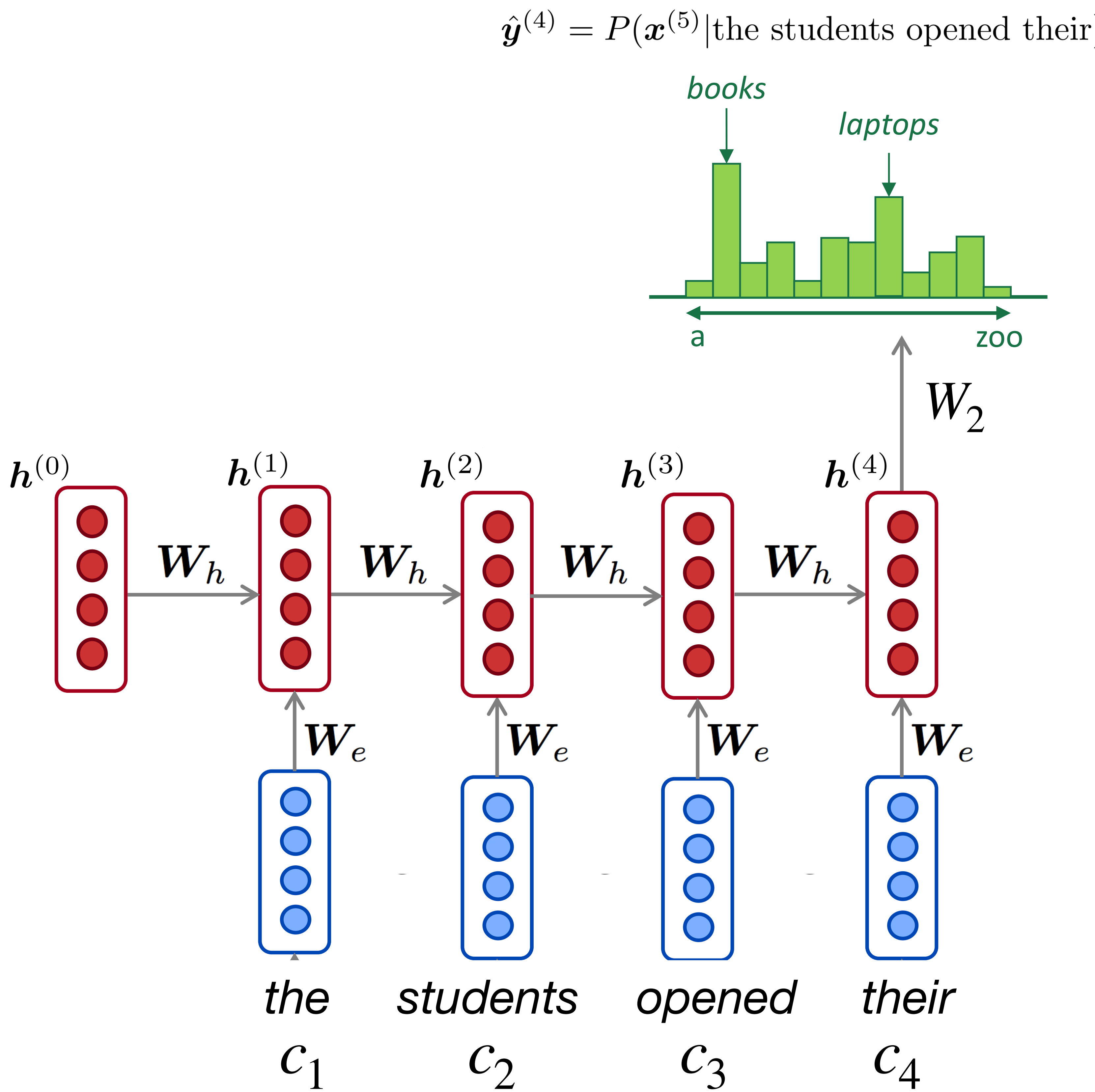
$$c_1, c_2, c_3, c_4$$



why is this good?

RNN **Advantages**:
- Can process any length input
- Model size doesn't increase for longer input
- Computation for step *t* can (in theory) use information from many steps back
- Weights are shared across timesteps → representations are shared

RNN **Disadvantages**:
- Recurrent computation is slow
- In practice, difficult to access information from many steps back



Training a RNN Language Model

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

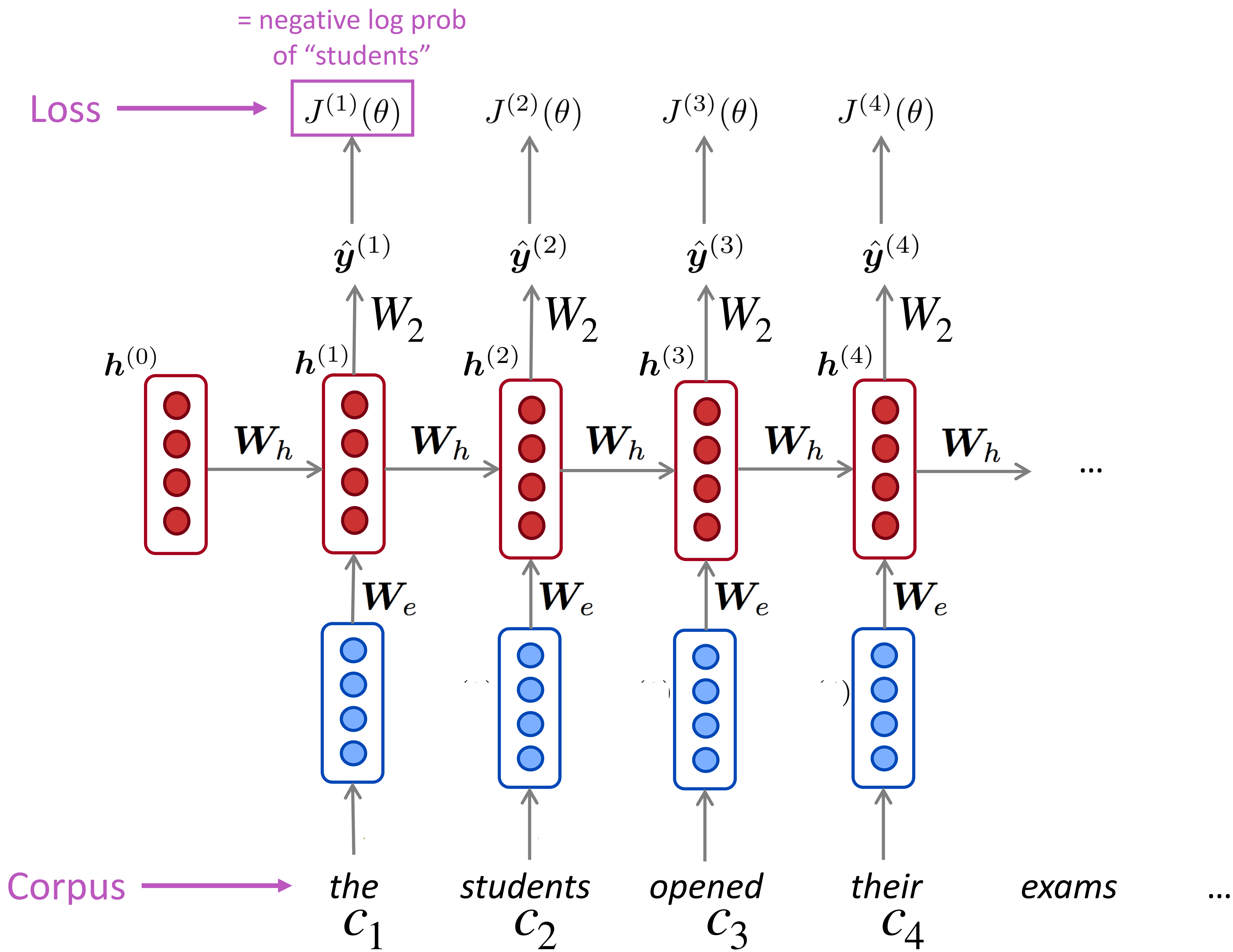
- Loss function on step *t* is usual cross-entropy between our predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)} = \boldsymbol{x}^{(t+1)}$:

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

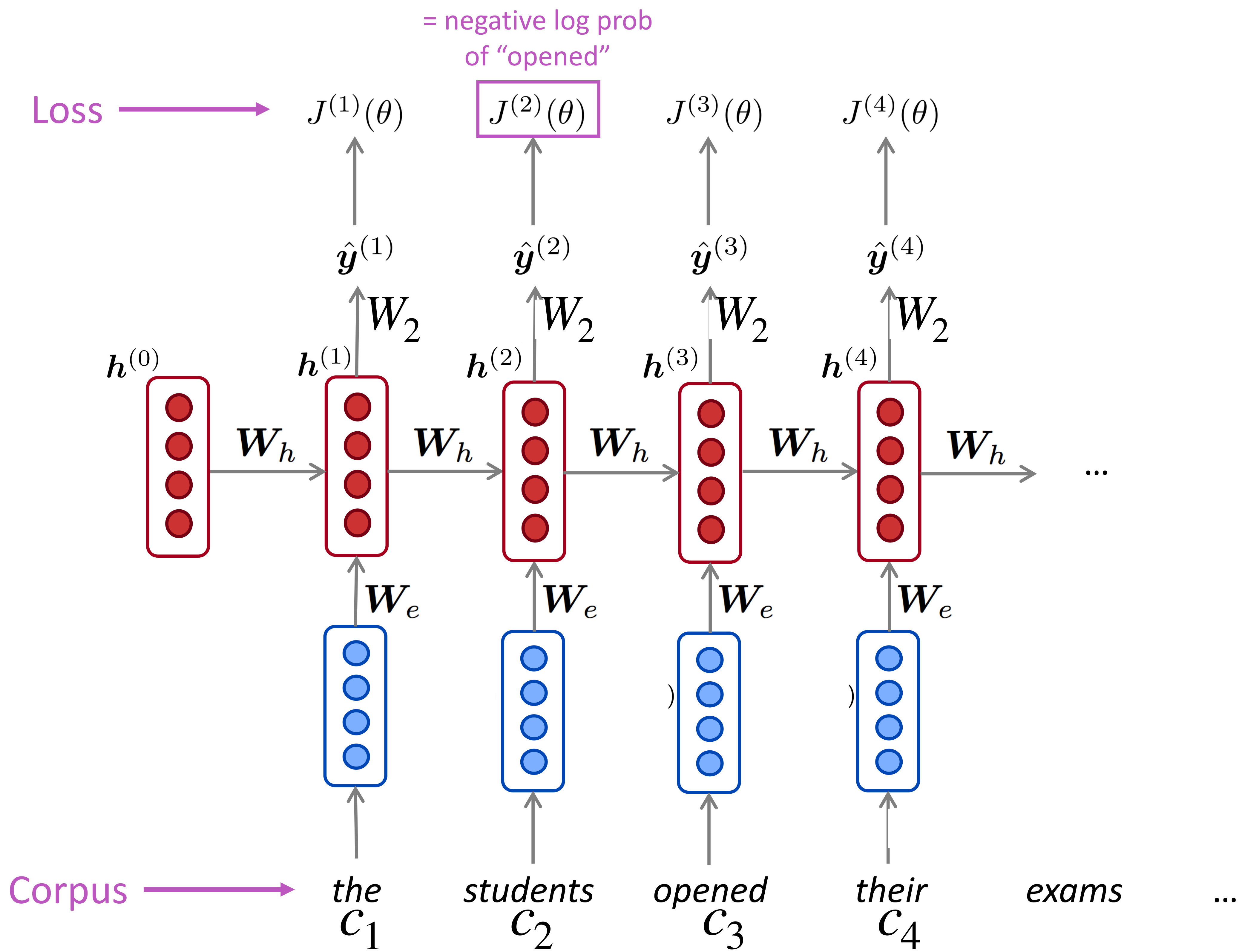
- Average this to get overall loss for entire training set:

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

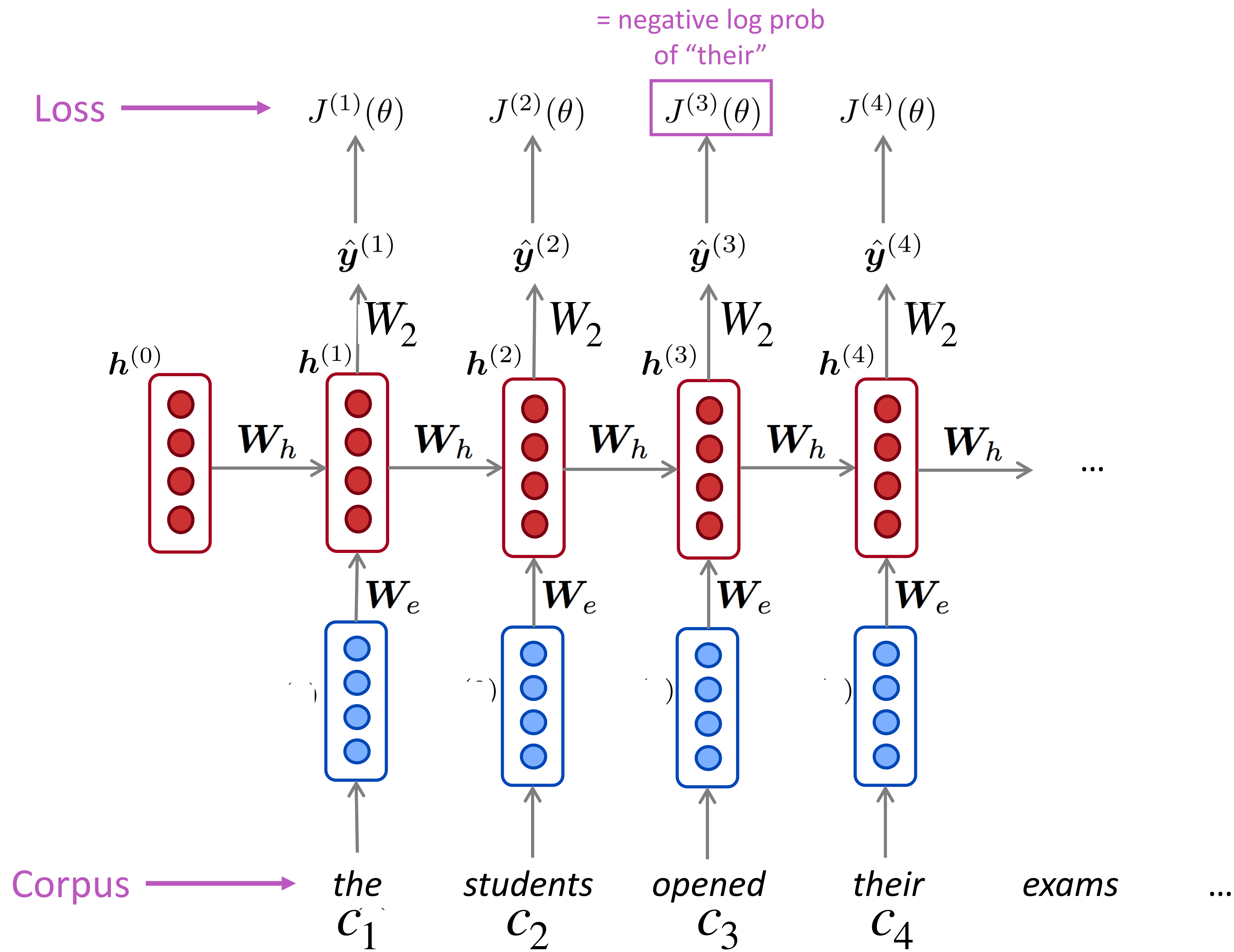
Training a RNN Language Model



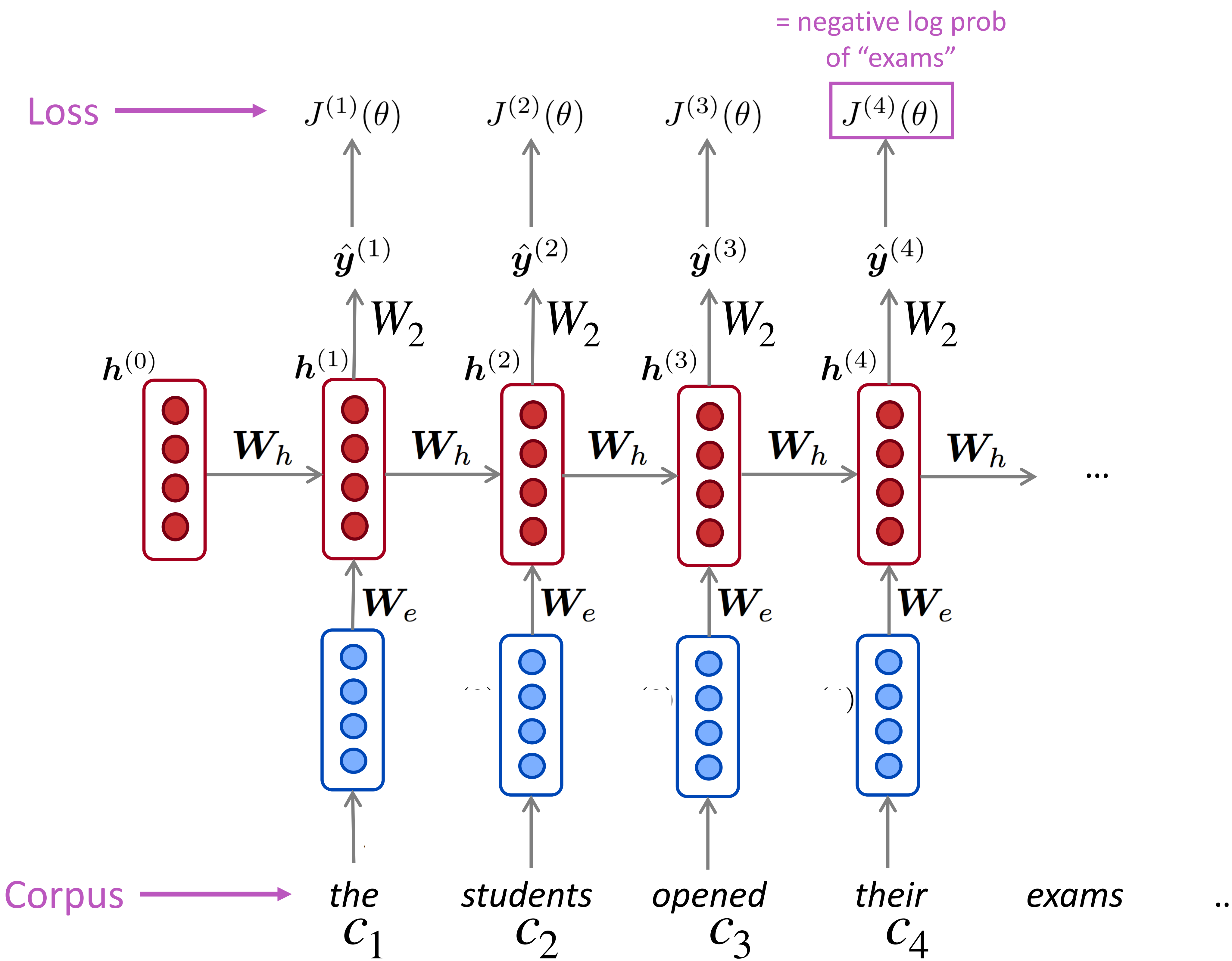
Training a RNN Language Model



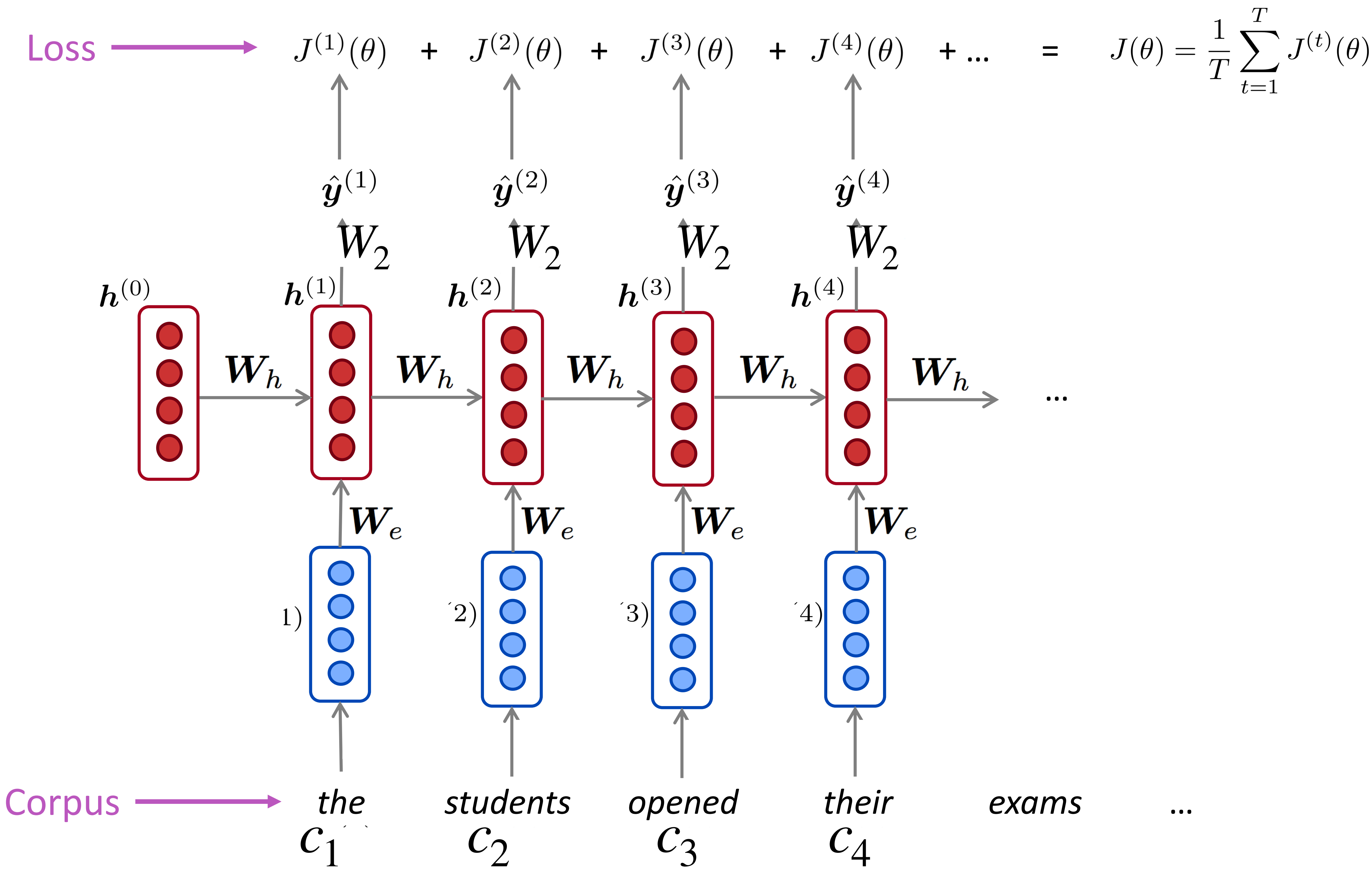
Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model

- However: Computing loss and gradients across entire corpus is too expensive!
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

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

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

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

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

Good afternoon. God bless you.

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. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretcks of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives.

Thank you very much. God bless you, and God bless the United States of America.

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

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a 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.

https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

RNNs have greatly improved perplexity

n-gram model

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

can we use language models to produce word embeddings?

Deep contextualized word representations. Peters et al., NAACL 2018

Word vectors are ubiquitous

Most if not all current state-of-the-art NLP systems use pre-trained word embeddings*

* With the exception of data rich tasks like machine translation

ALLEN INSTITUTE *for* ARTIFICIAL INTELLIGENCE

word2vec represents each word as a **single vector**

play = [0.2, -0.1, 0.5, ...]
bank = [-0.3, 1.4, 0.7, ...]
run = [-0.5, -0.3, -0.1, ...]

Single vector per word

The new-look ***play*** area is due to be completed by early spring 2010 .

Single vector per word

Gerrymandered congressional districts favor representatives who ***play*** to the party base .

Single vector per word

The freshman then completed the three-point ***play*** for a 66-63 lead .

Nearest neighbors

play = [0.2, -0.1, 0.5, ...]

Nearest Neighbors

playing
game
games
played
players
plays
player
Play
football
multiplayer

Multiple senses entangled

play = [0.2, -0.1, 0.5, ...]

Nearest Neighbors

VERB

playing
game
games
played
players

plays
player
Play
football
multiplayer

Multiple senses entangled

play = [0.2, -0.1, 0.5, ...]

Nearest Neighbors

playing	**VERB**	plays
game	**NOUN**	player
games		Play
played		football
players		multiplayer

Multiple senses entangled

play = [0.2, -0.1, 0.5, ...]

Nearest Neighbors

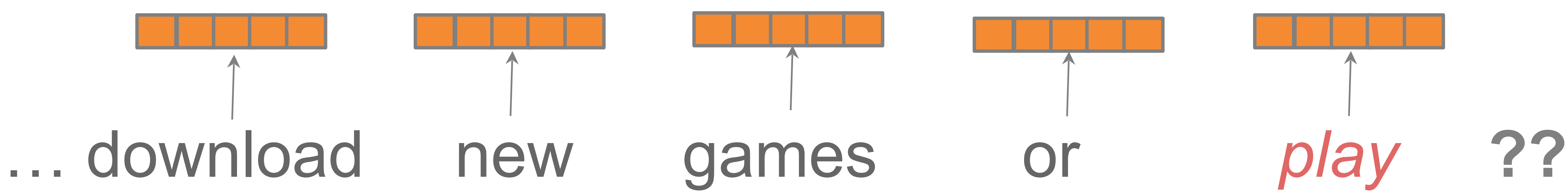
playing	**VERB**	plays
game	**NOUN**	player
games	**ADJ**	Play
played		football
players		multiplayer

Deep bidirectional language model

… download new games or *play* ??

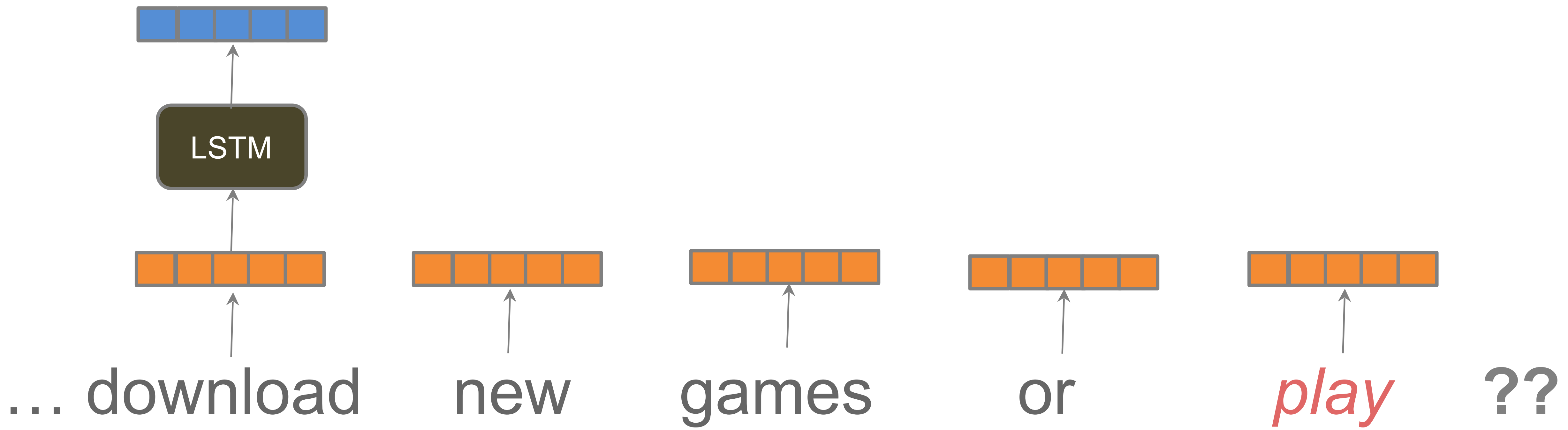
ALLEN INSTITUTE *for* ARTIFICIAL INTELLIGENCE

Deep bidirectional language model

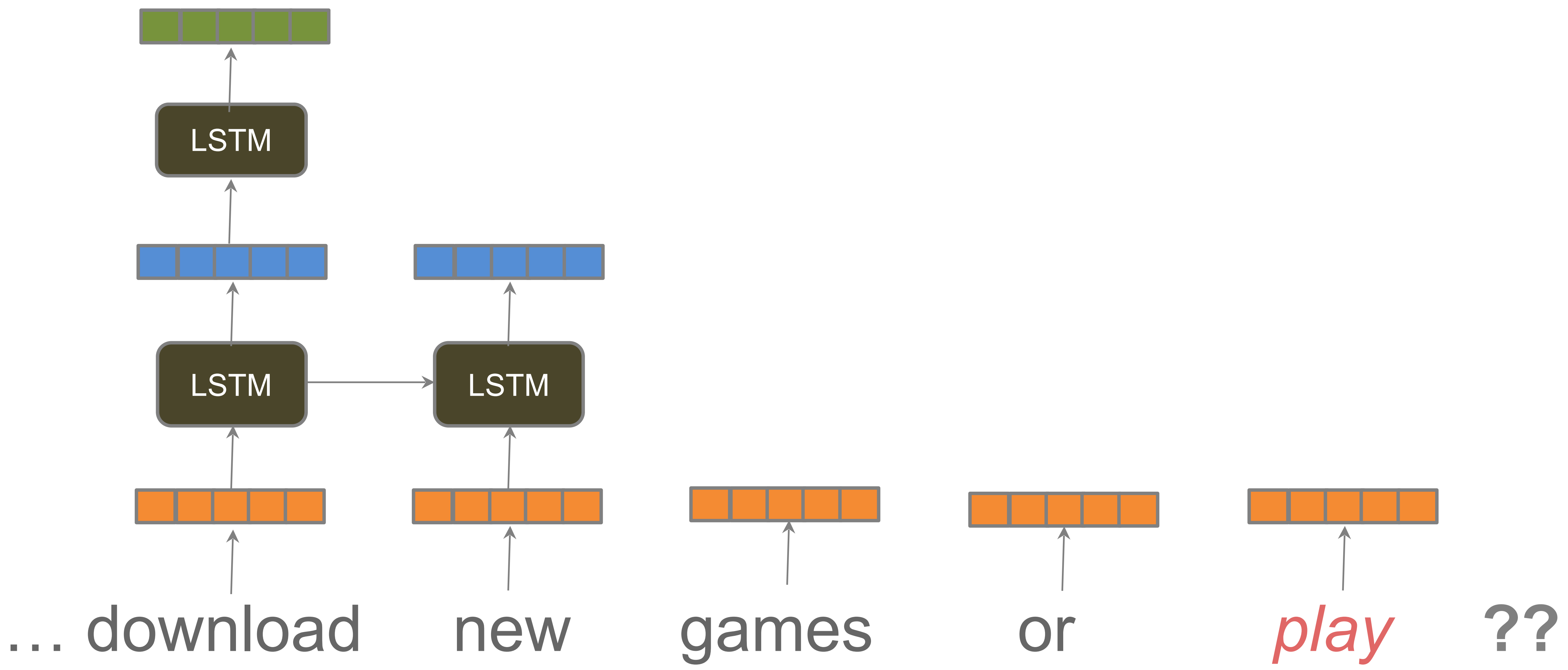


Deep bidirectional language model

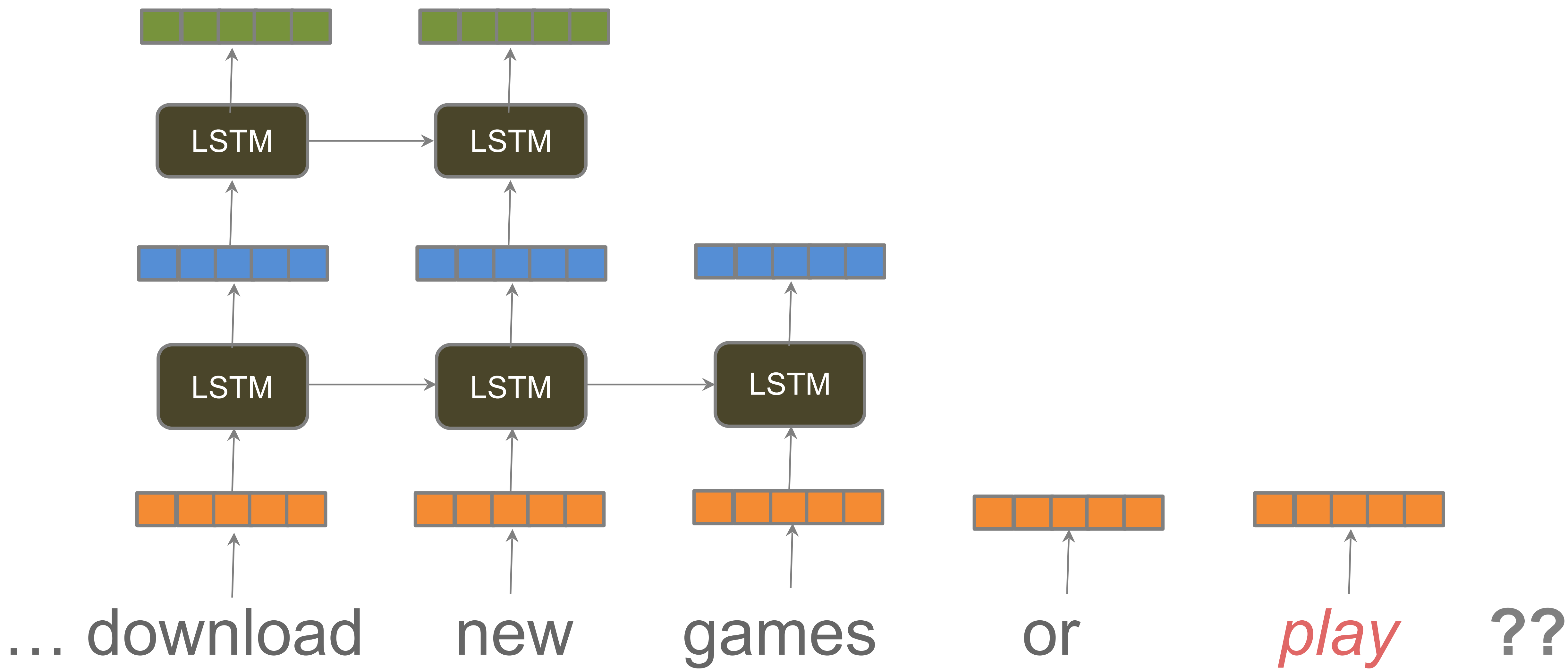
Note: an LSTM is a more powerful variant of the RNN we just learned about! We won't go into how LSTMs work in this class.



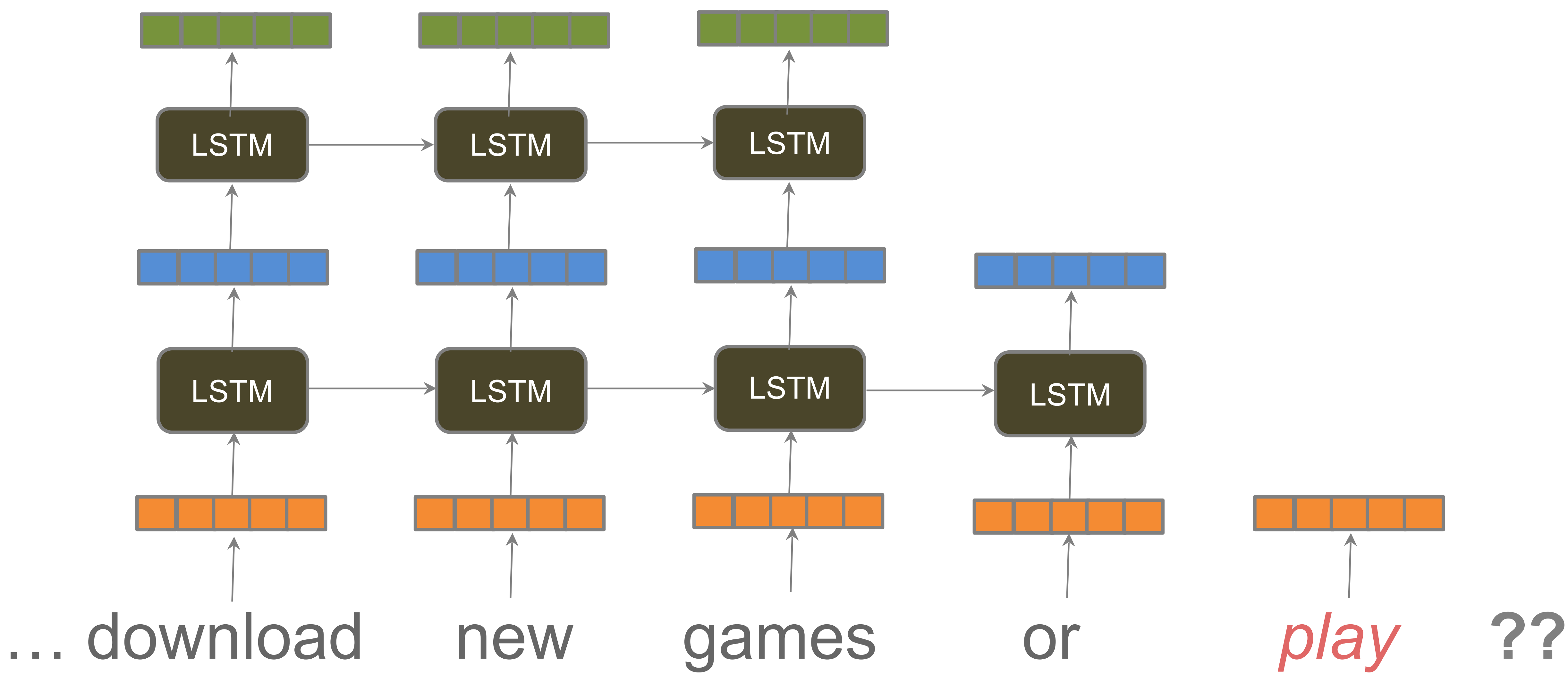
Deep bidirectional language model



Deep bidirectional language model

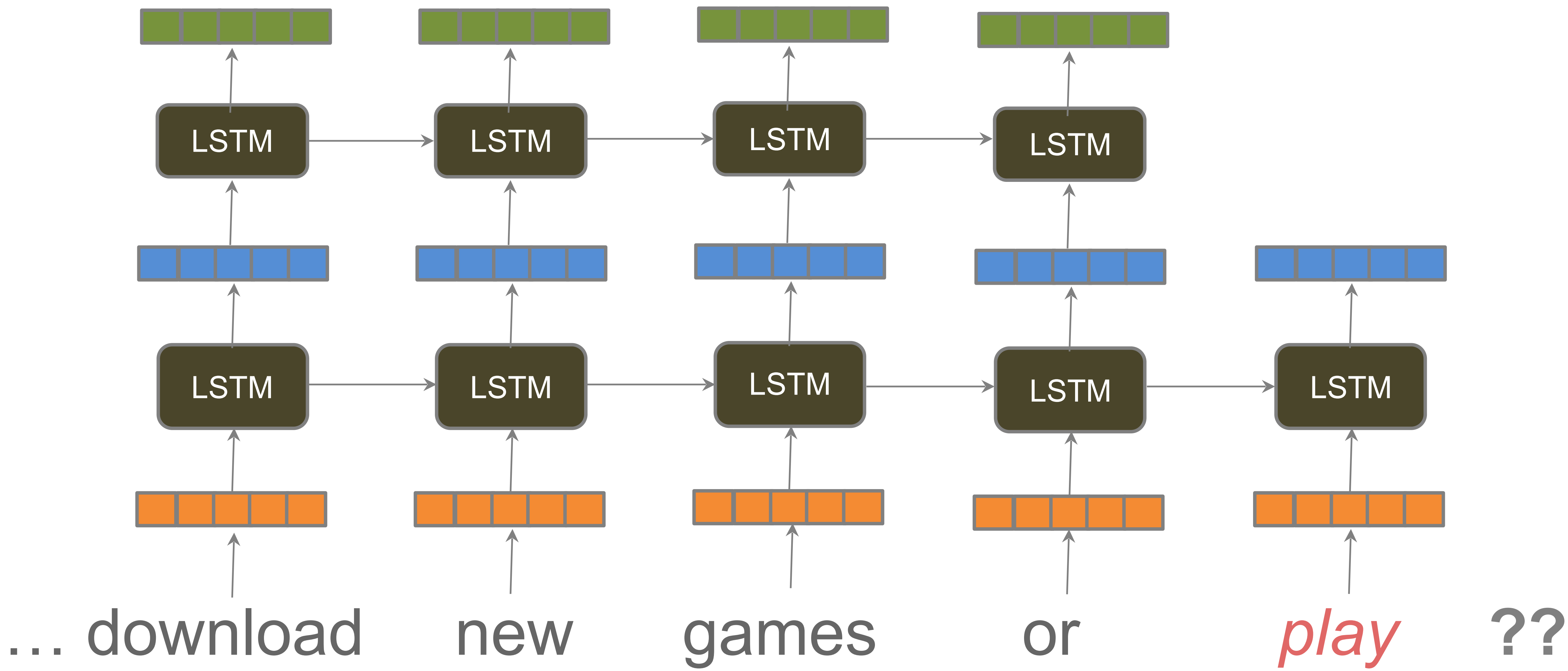


Deep bidirectional language model



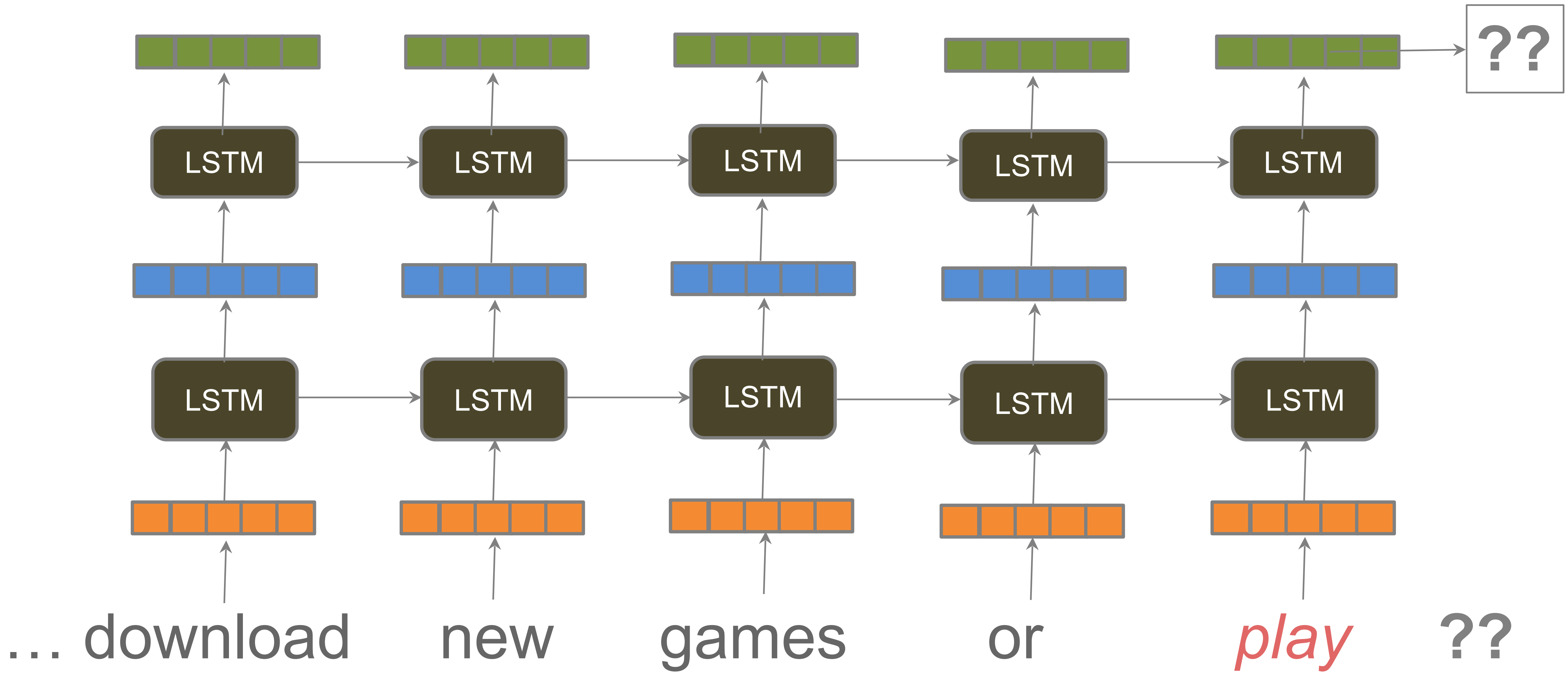
ALLEN INSTITUTE *for* ARTIFICIAL INTELLIGENCE

Deep bidirectional language model

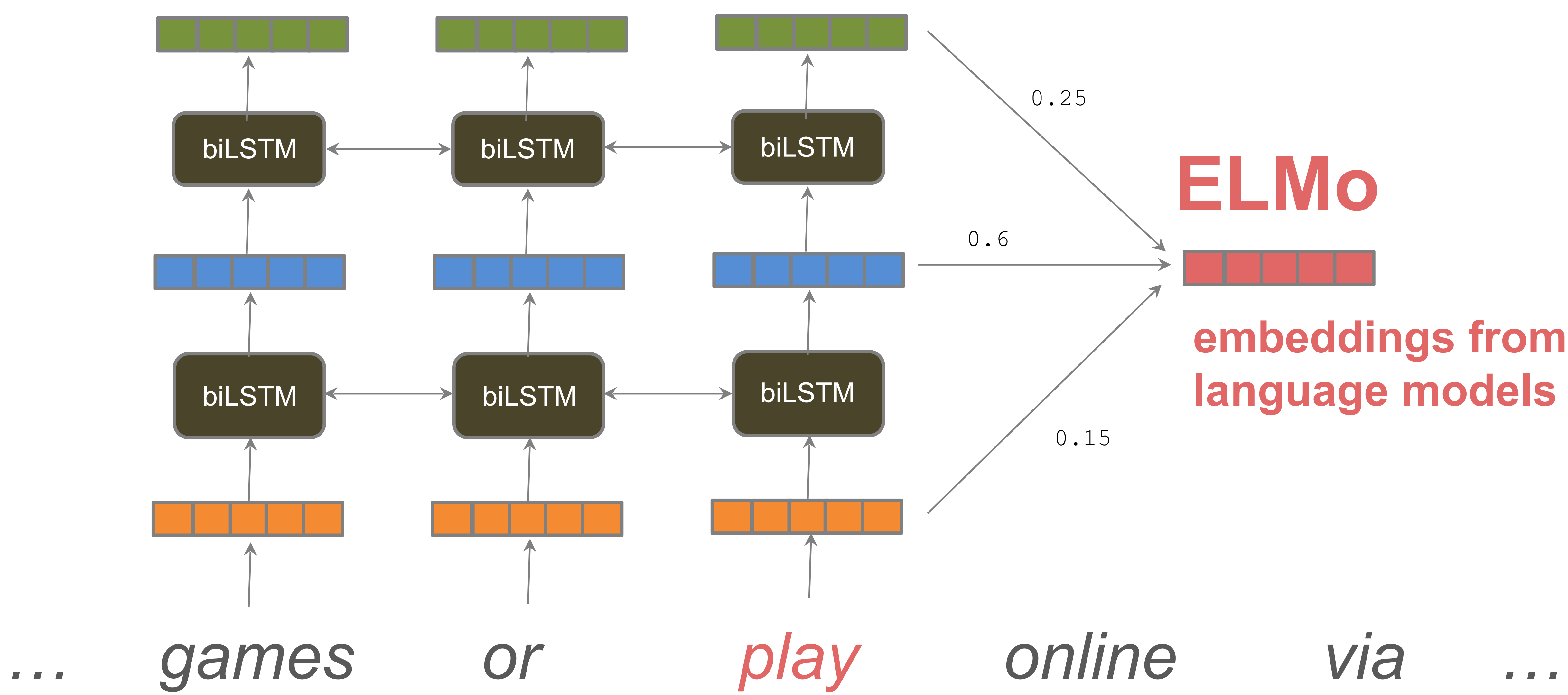


ALLEN INSTITUTE
for ARTIFICIAL INTELLIGENCE

Deep bidirectional language model



Use all layers of language model

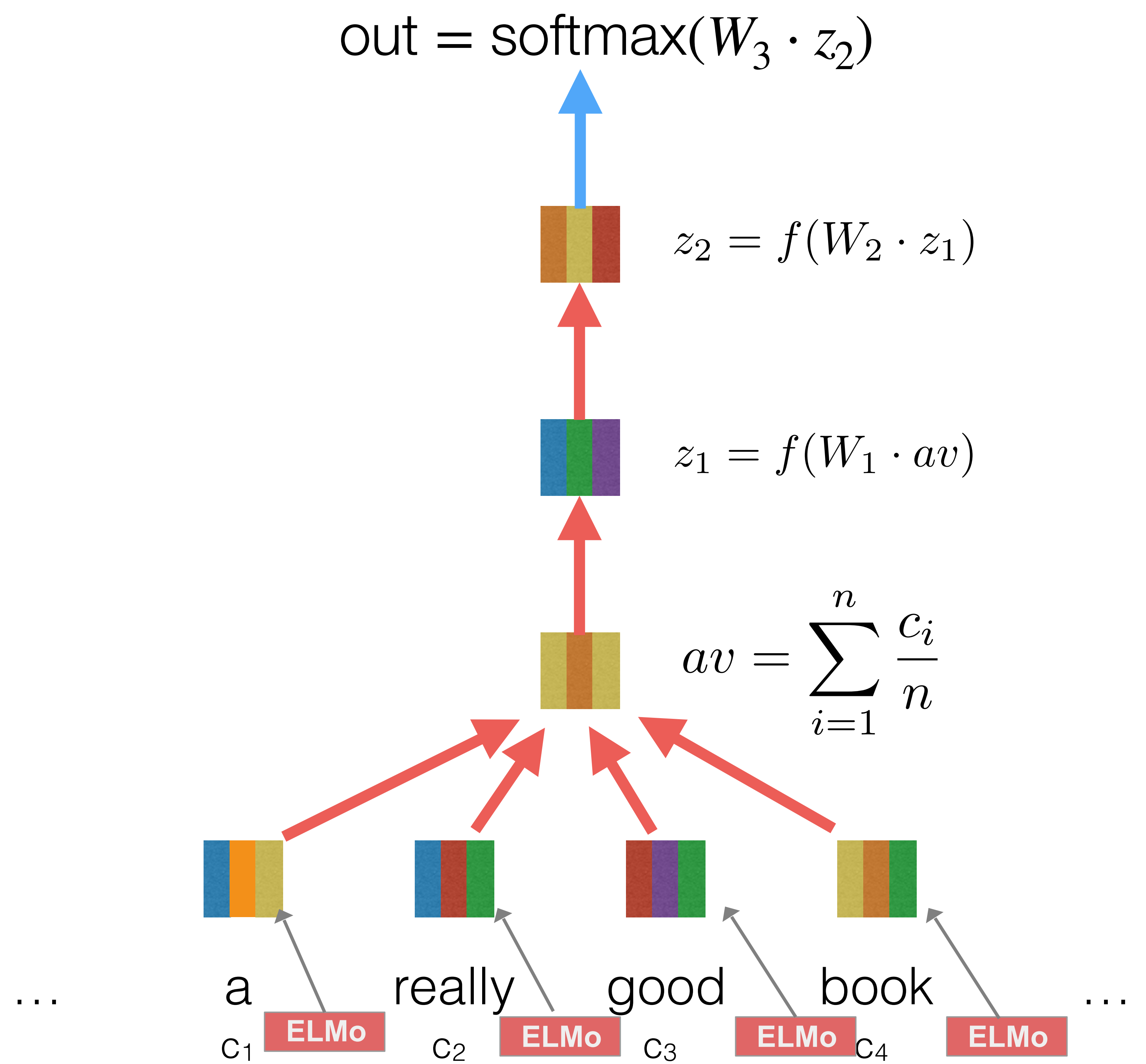


Contextual representations

ELMo representations are **contextual** – they depend on the entire sentence in which a word is used.

how many different embeddings does ELMo compute for a given word?

how to use ELMo in NLP tasks?



ELMo improves NLP tasks

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

ALLEN INSTITUTE *for* ARTIFICIAL INTELLIGENCE

Large-scale recurrent neural language models learn contextual representations that capture basic elements of semantics and syntax

Adding ELMo to existing state-of-the-art models provides significant performance improvement on all NLP tasks.

TensorFlow™

```
elmo = hub.Module("https://tfhub.dev/google/elmo/1", trainable=True)
embeddings = elmo(
    ["the cat is on the mat", "dogs are in the fog"],
    signature="default",
    as_dict=True)["elmo"]
```