Intro: neural networks

CS 685, Fall 2025

Advanced Natural Language Processing

https://people.cs.umass.edu/~brenocon/cs685_f25/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

- Thanks for HW1!
- Midterm #1: will be in-class on either Tu 10/7 or Th 10/9
- HW2: will be released later this week

Exercise #1

- Instructions
  - Please work individually without internet on these questions.
  - Write answers on one sheet of paper or word processing doc. Feel free to complete them after lecture.
  - Submit a PDF to Gradescope, "Exercise 1" before Thursday's lecture.
- **Questions (based on mandatory readings)**
  - 1. In last week's Landauer and Dumais reading, they systematically varied their model's main/only hyperparameter, and examined its impact on downstream task performance. What was that hyperparameter?
  - 2. What was the downstream task?
    (Optional: what was the result of their experiment?)
  - 3. In addition to the logistic sigmoid, what two nonlinear functions for NNs are introduced in today's reading (J&M Chapter 6)?

Target:		levied
Choices:	(a)	imposed
	(b)	believed
	(c)	requested
	(d)	correlated
Solution:	*(a)*	*imposed*

- TOEFL (near-)synonym task
  - Acontextual version
- Choose answer with highest cosine to E("levied")
- https://aclweb.org/aclwiki/TOEFL_Synonym_Questions_(State_of_the_art)
- Hyperparameter sweep!
  - You should always get a cap shape if you've searched enough

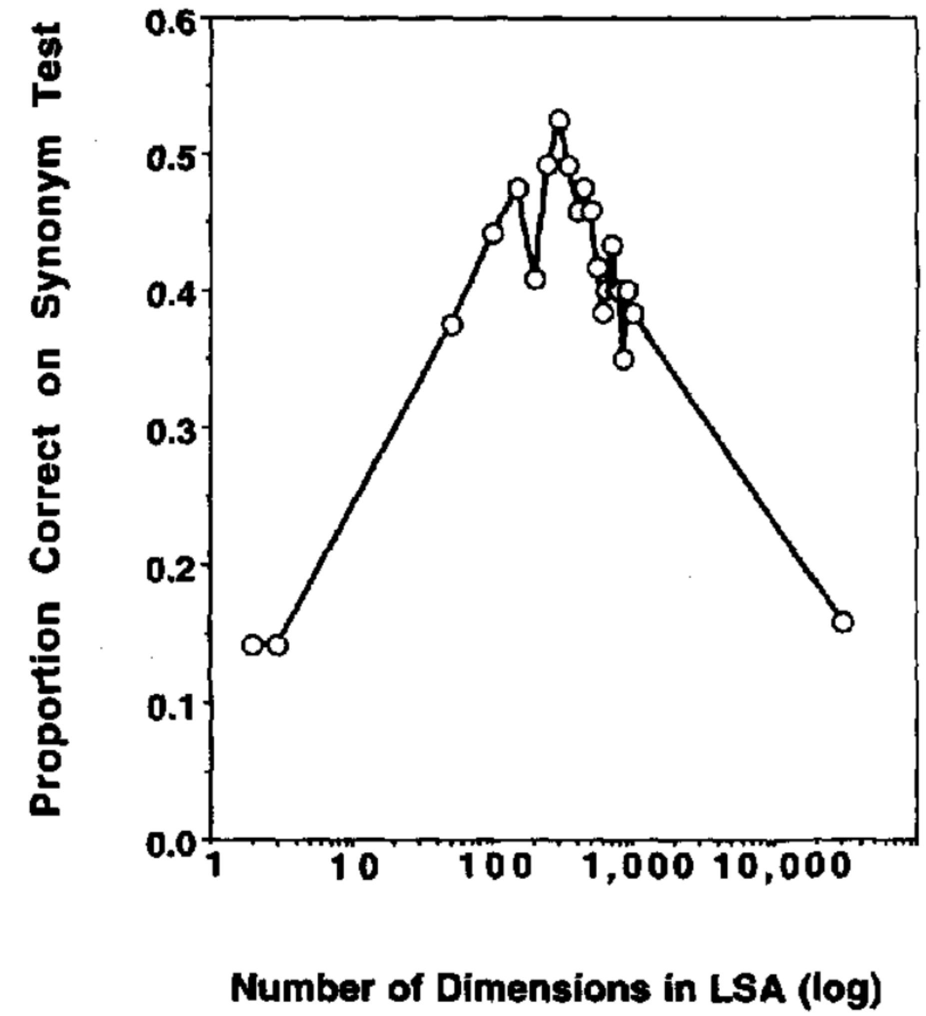


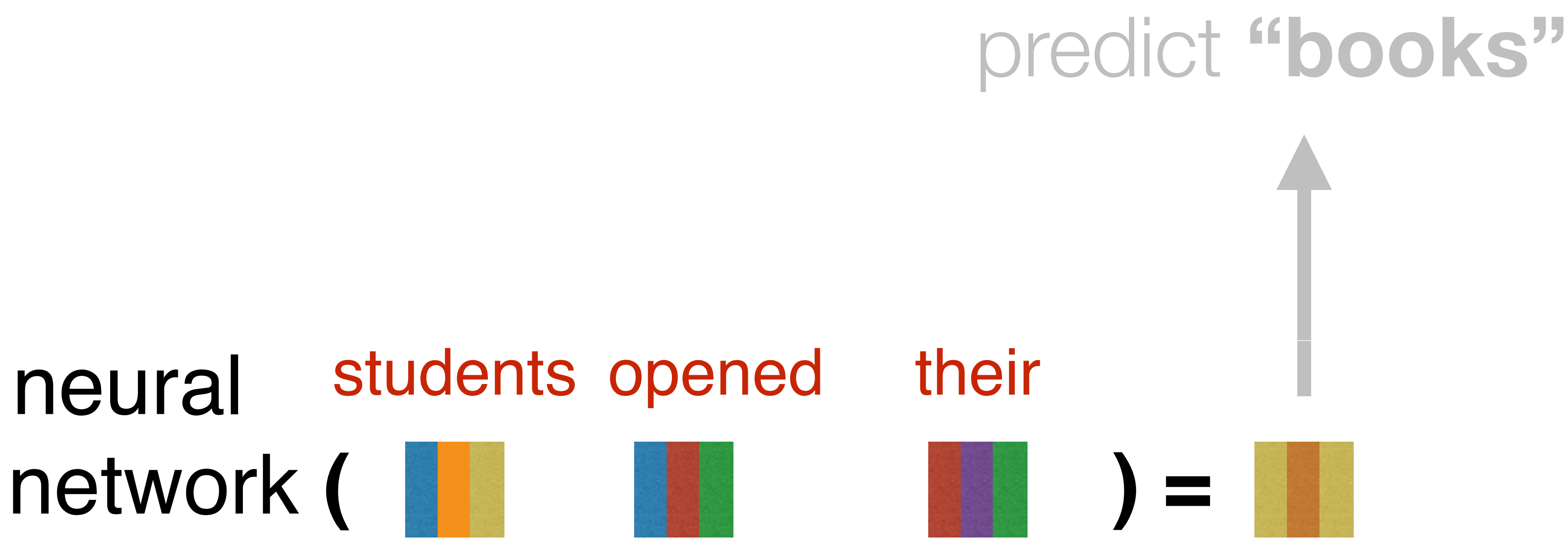
Figure 3. The effect of number of dimensions retained in latent-semantic-analysis (LSA)–singular-value-decomposition (SVD) simulations of word-meaning similarities. The dependent measure is the proportion of 80 multiple-choice synonym test items for which the model chose the correct answer. LSA was trained on text samples from 30,473 articles in an electronic file of text for the *Groliers Academic American Encyclopedia.*

Today: Neural Networks for Language

- Artificial neural network models of language incorporate
  - Word embeddings (done!)
  - Cool/controversial modeling metaphor (new?)
  - Non-linearities or feedforward layers (new)
  - Flexible architectures (new)

  - Learning from predictive loss minimization (review - Thurs)
  - Flexible architectures enabled by autodiff-based learning (new - Thurs)
    - important to deal with sequential and dependency structure in language
    - main architectures:
      - Recurrent NNs
      - Attention and Transformer NNs

how do we compute a prefix representation ***x*** for next-word prediction?



Composition functions

input: sequence of word embeddings corresponding to the tokens of a given prefix

output: single vector

- Element-wise functions (e.g. sum)
- Concatenation

- Feed-forward neural networks
- Recurrent neural networks
- Convolutional neural networks *[skipping]*

- Transformers (self-attention) neural networks

Let's look first at *concatenation* + *feedforward NN*, an easy to understand but limited composition function

review of Bengio's model (non-linearities skipped last week)

A fixed-window neural Language Model

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

discard

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

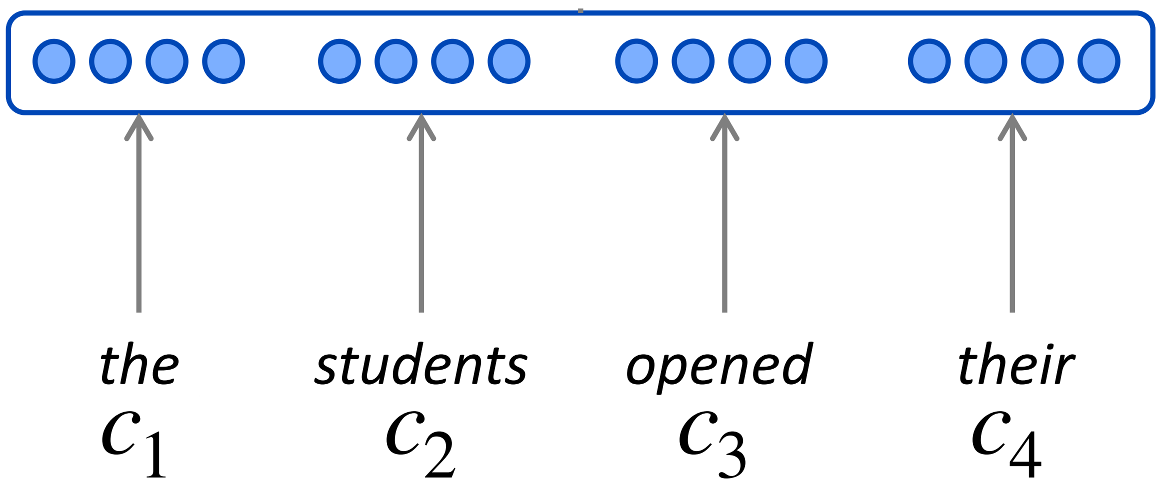
A fixed-window neural Language Model

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



A fixed-window neural Language Model

hidden layer

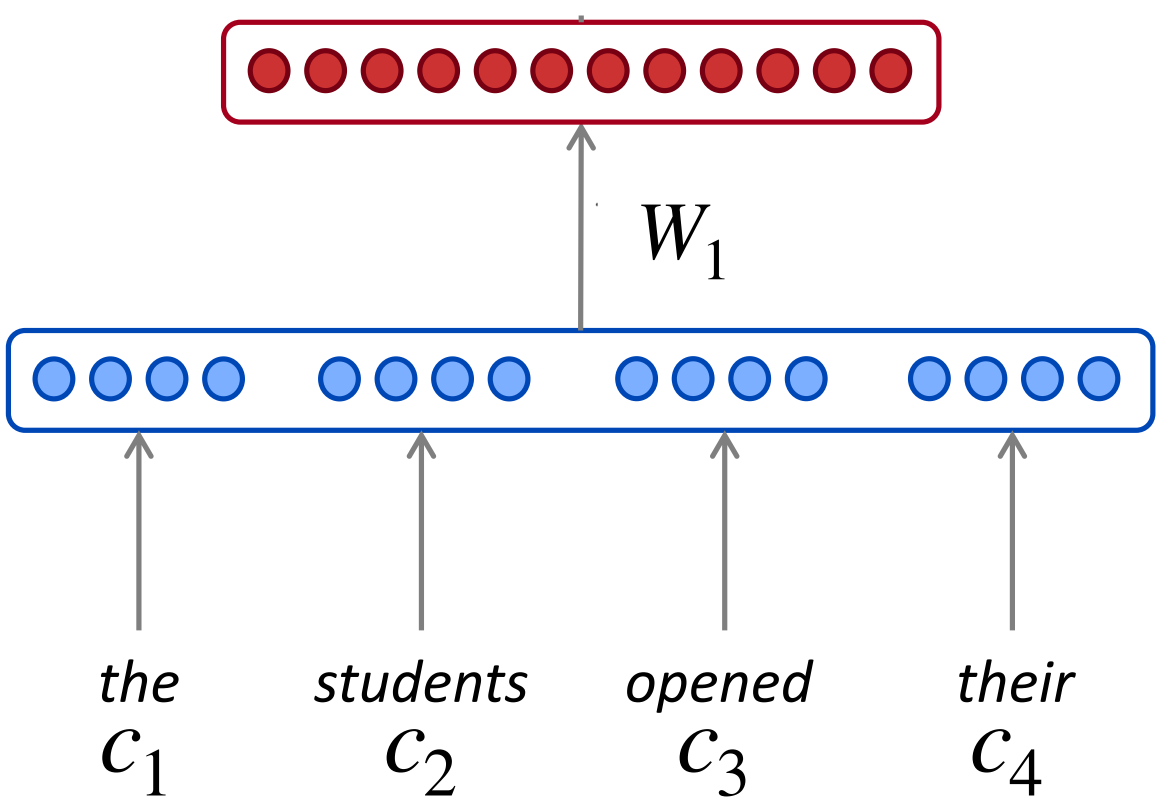
$$h = f(W_1 x)$$

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



A fixed-window neural Language Model

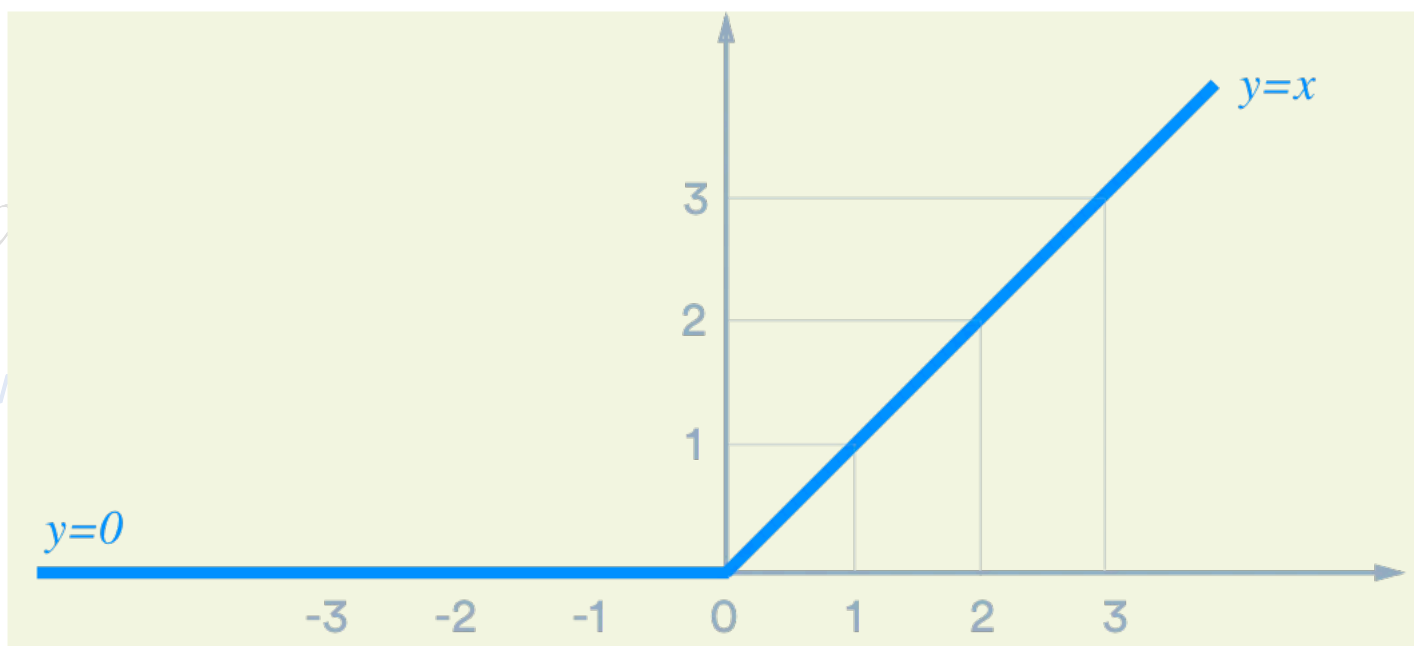
f is a *nonlinearity*, or an element-wise nonlinear function. The most commonly-used choice today is the rectified linear unit (**ReLu**), which is just **ReLu**(x) = max(0, x). Other choices include **tanh** and **sigmoid**.

hidden layer

$h = f(W_1$

concatenated w

$x = [c_1;$



words / one-hot vectors

c_1, c_2, c_3, c_4

the c_1

students c_2

opened c_3

their c_4

A fixed-window neural Language Model

output distribution

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

hidden layer

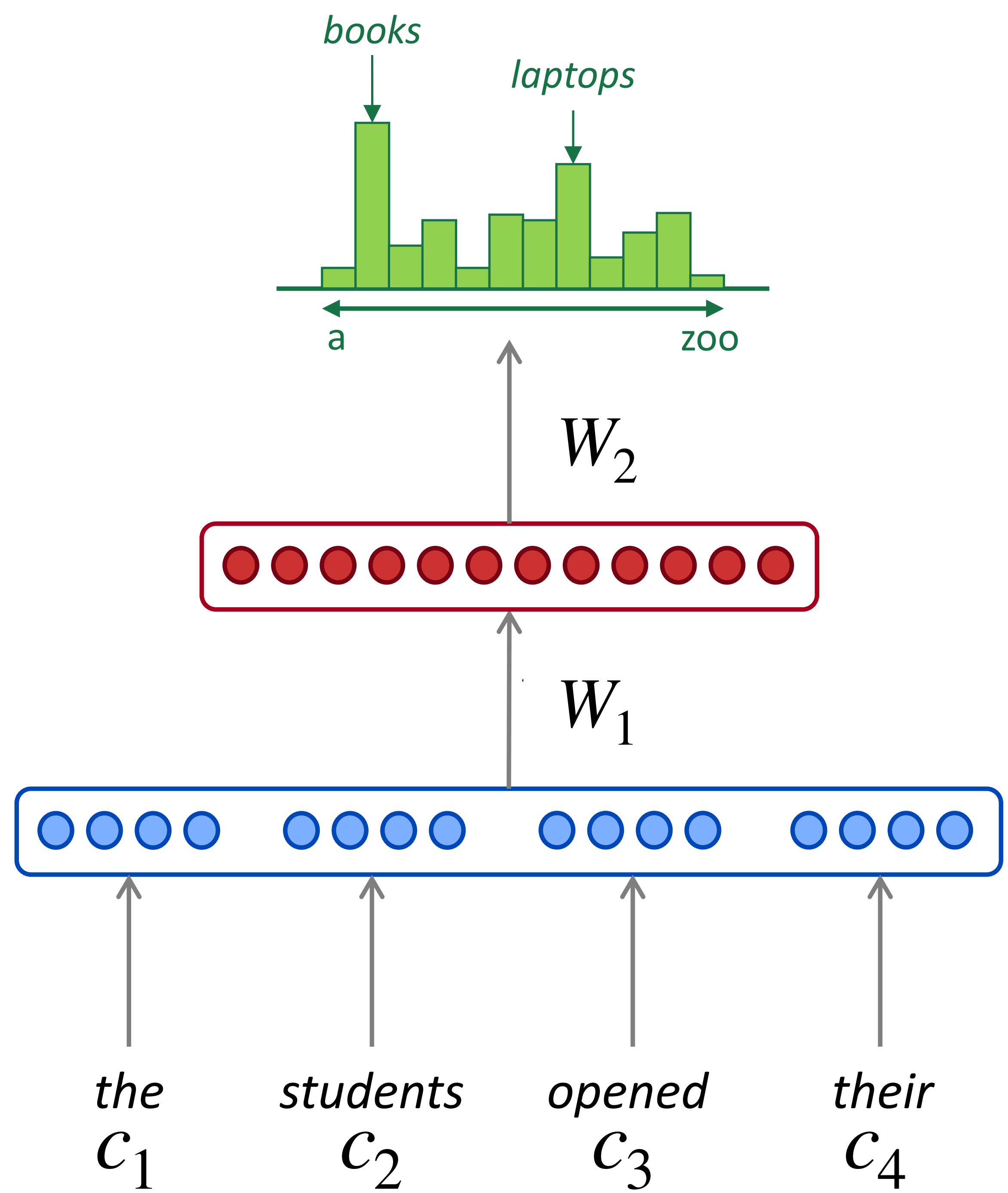
$$h = f(W_1 x)$$

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$

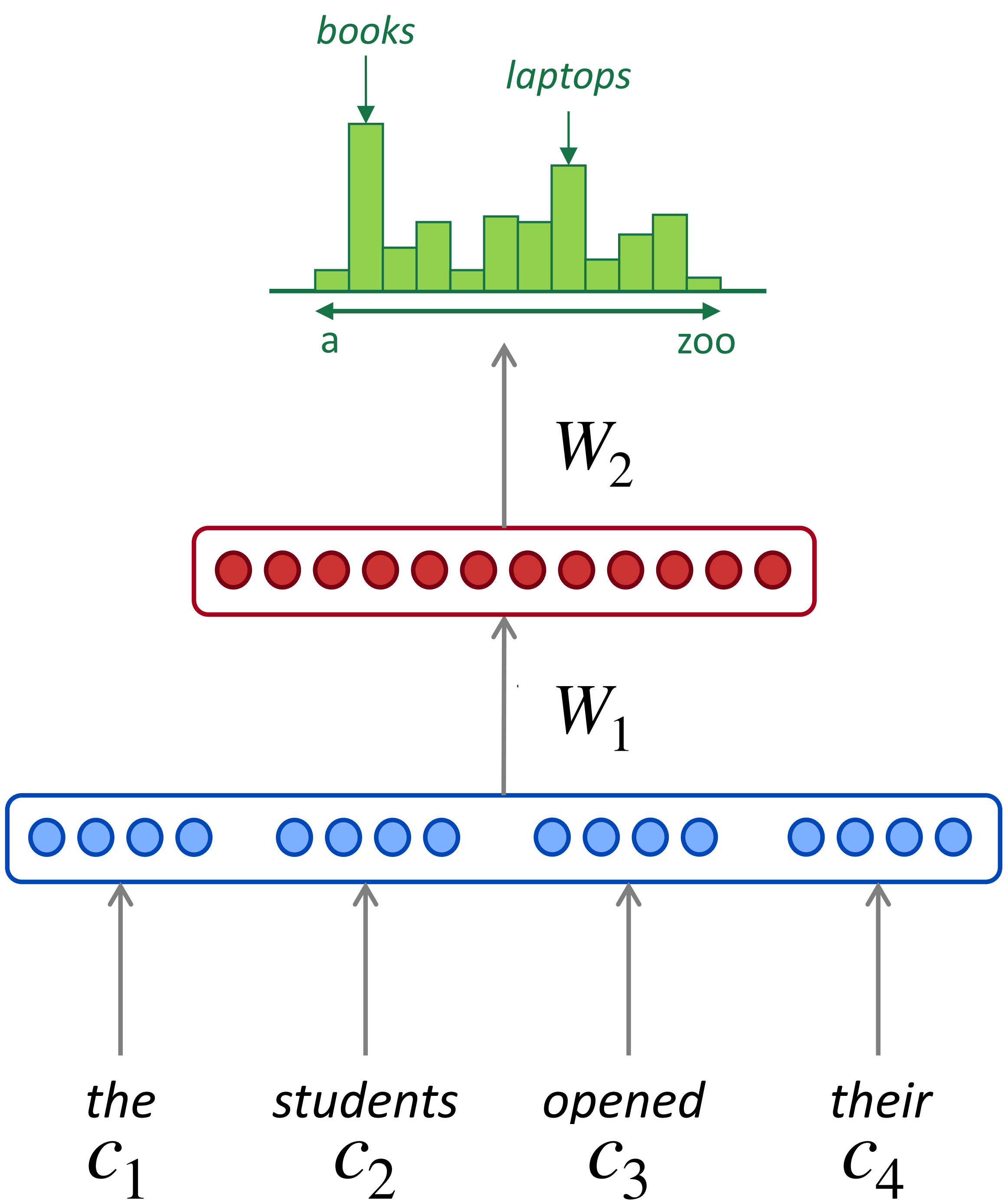


This is also called Feedforward NN , MLP (multilayer perceptron), or fully connected network

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

Improvements over *n*-gram LM:
- No sparsity problem

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.



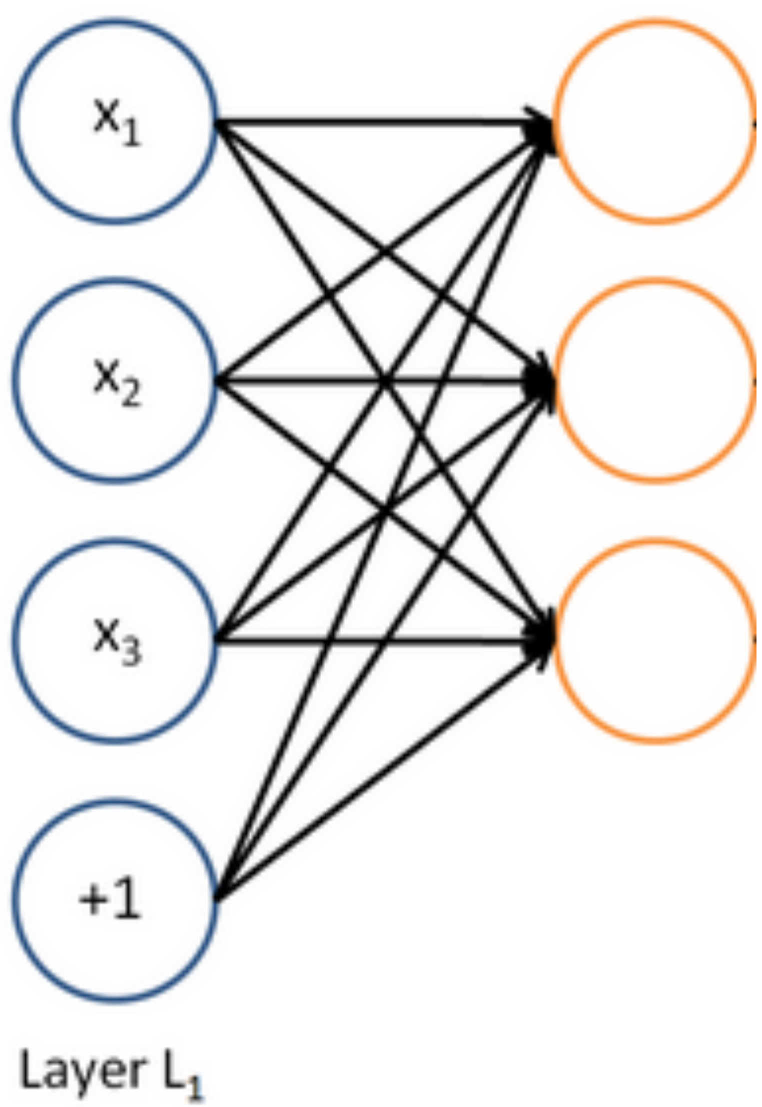
NN non-linearities

- Why use an element-wise "squashing" function?

https://playground.tensorflow.org/

Feedforward (MLP) Neural Network

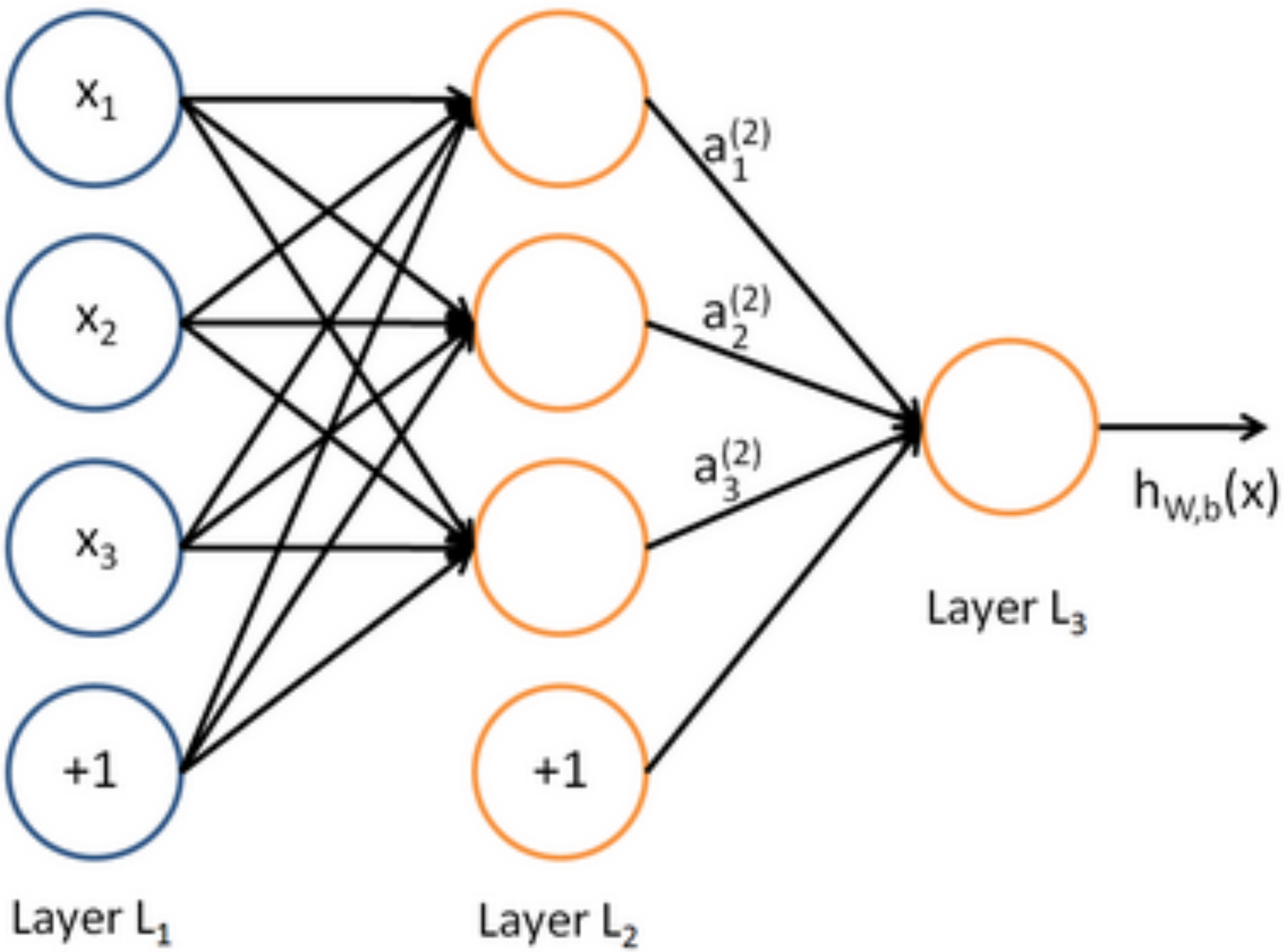
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

Feedforward (MLP) Neural Network

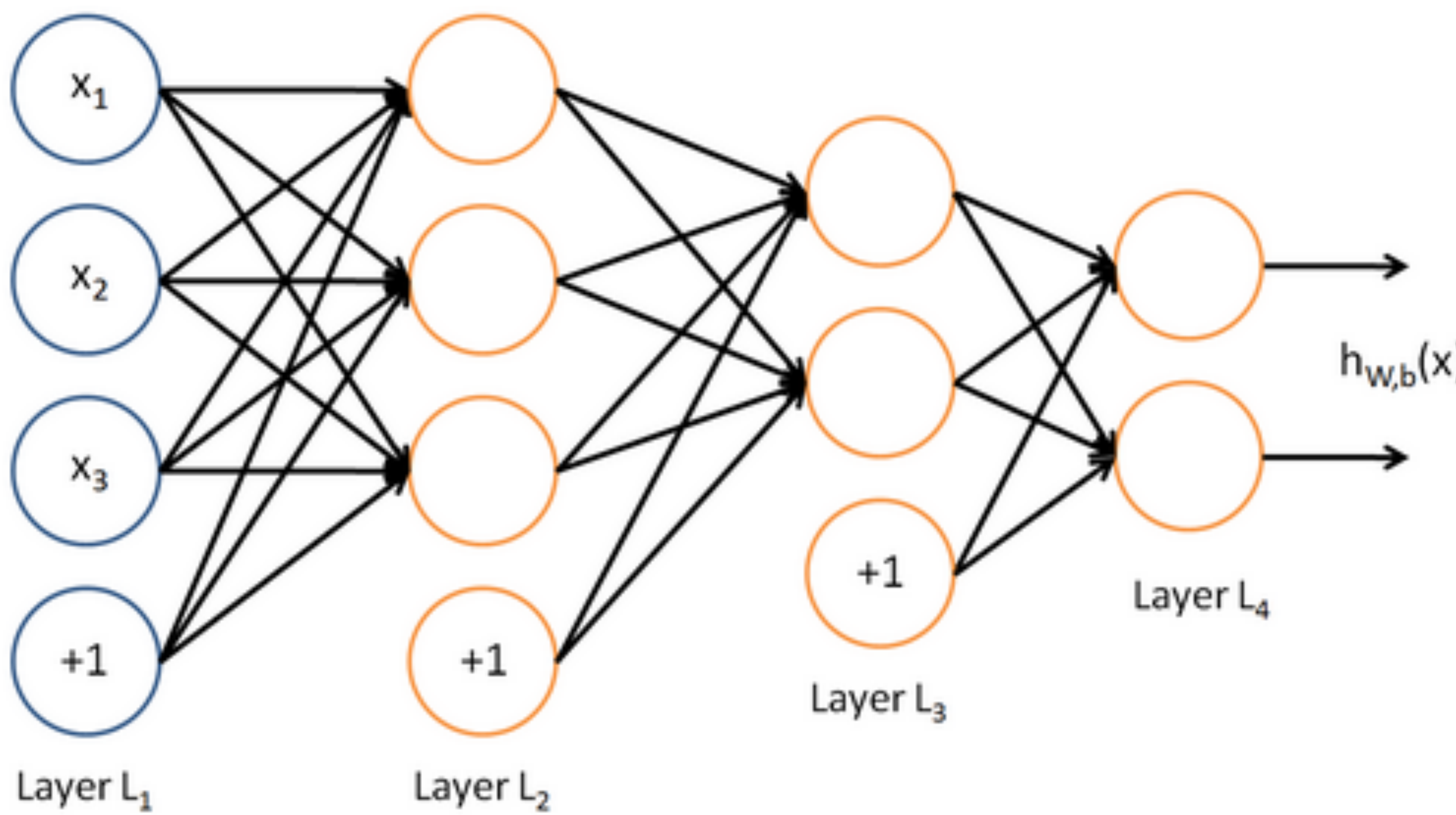
… which we can feed into another logistic regression function



It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

Feedforward (MLP) Neural Network

Before we know it, we have a multilayer neural network….



Question

- Say we have a many-layered feedforward network
  - y = g(A g(B g(C x)))
- Can we use the identity function g(x)=x for the elementwise non-linearity layer?

- Visual demo of the potential usefulness NN non-linearities (for non-language data, at least)

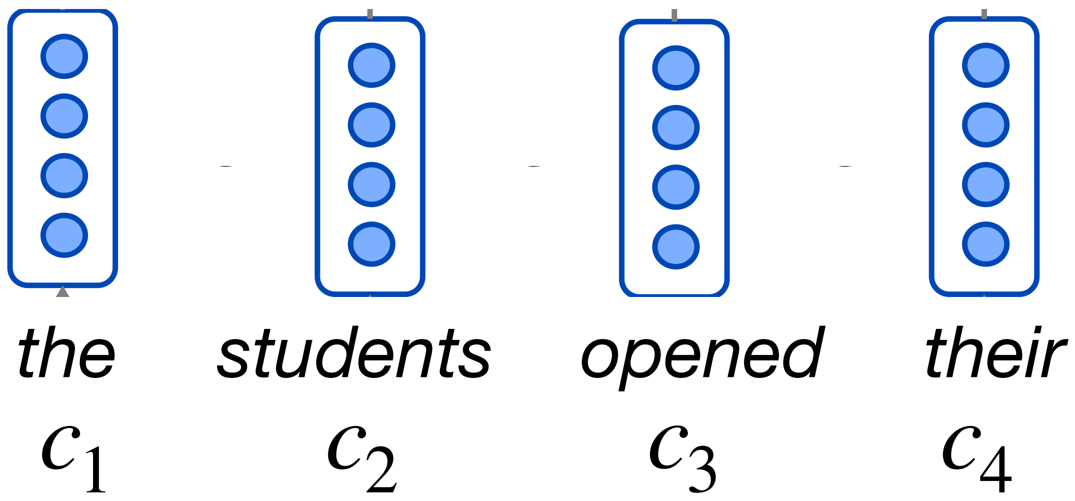
https://playground.tensorflow.org/

Recurrent Neural Networks!

A RNN Language Model

word embeddings

c_1, c_2, c_3, c_4



A RNN Language Model

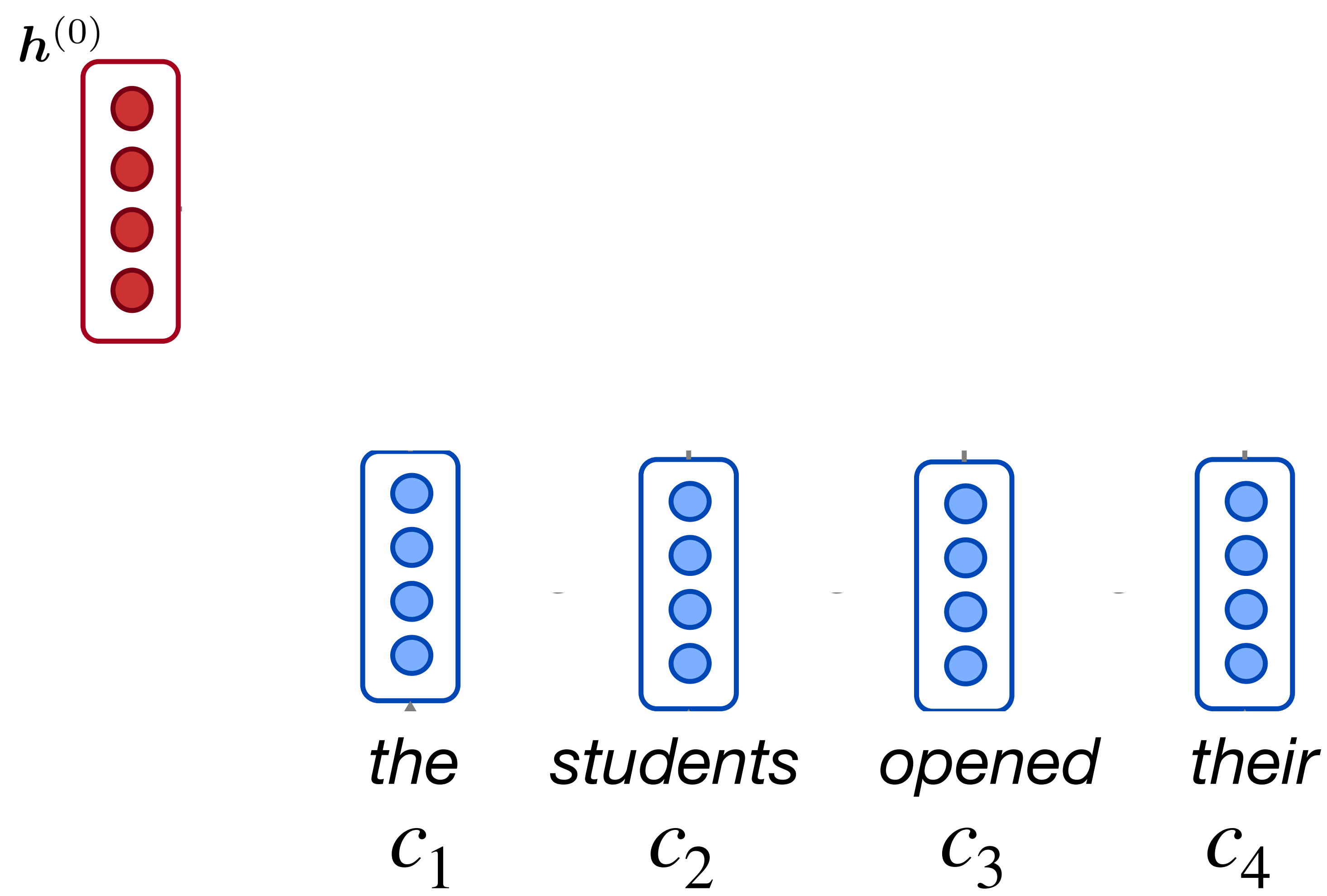
hidden states

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

h(0) is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

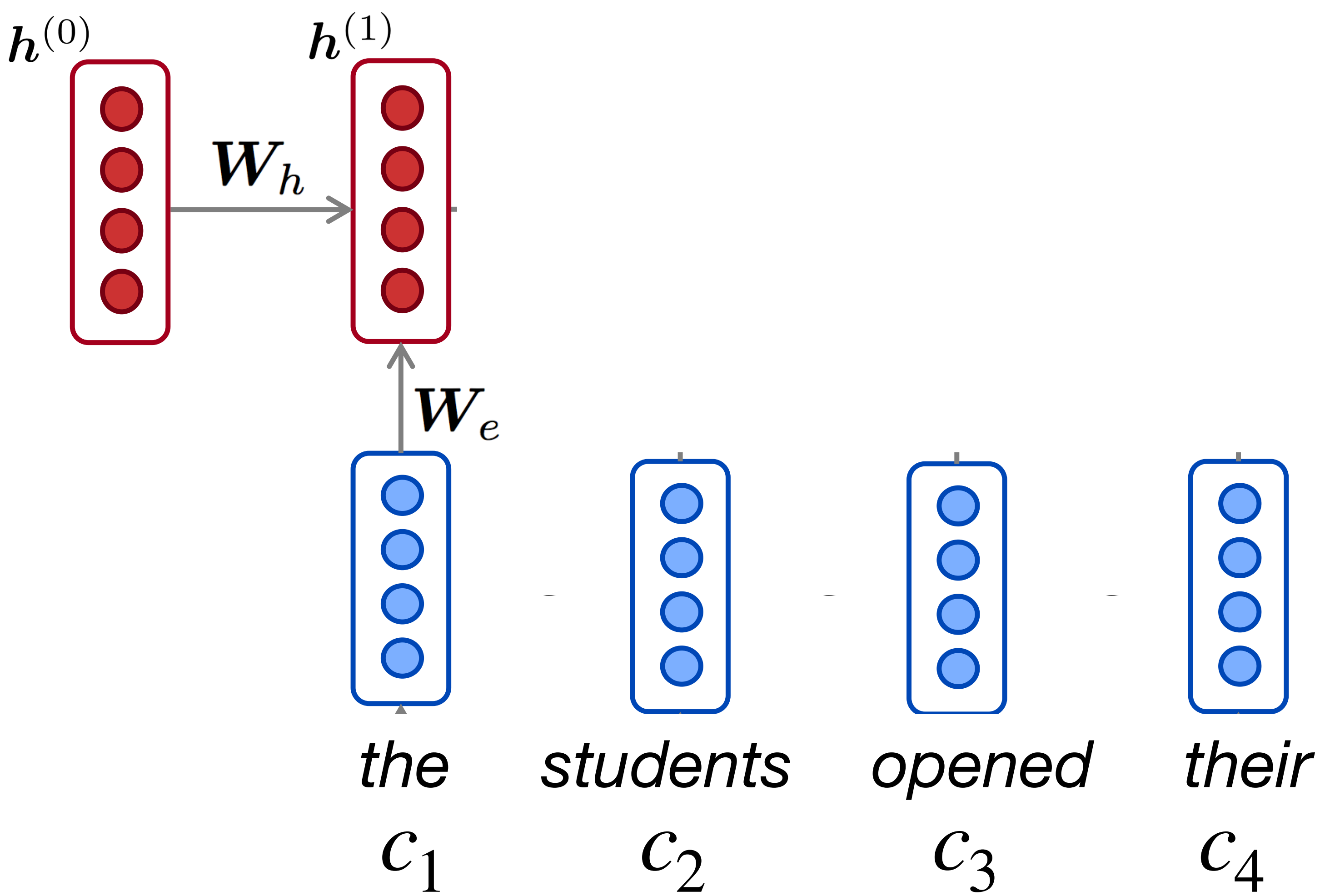
hidden states

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

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

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

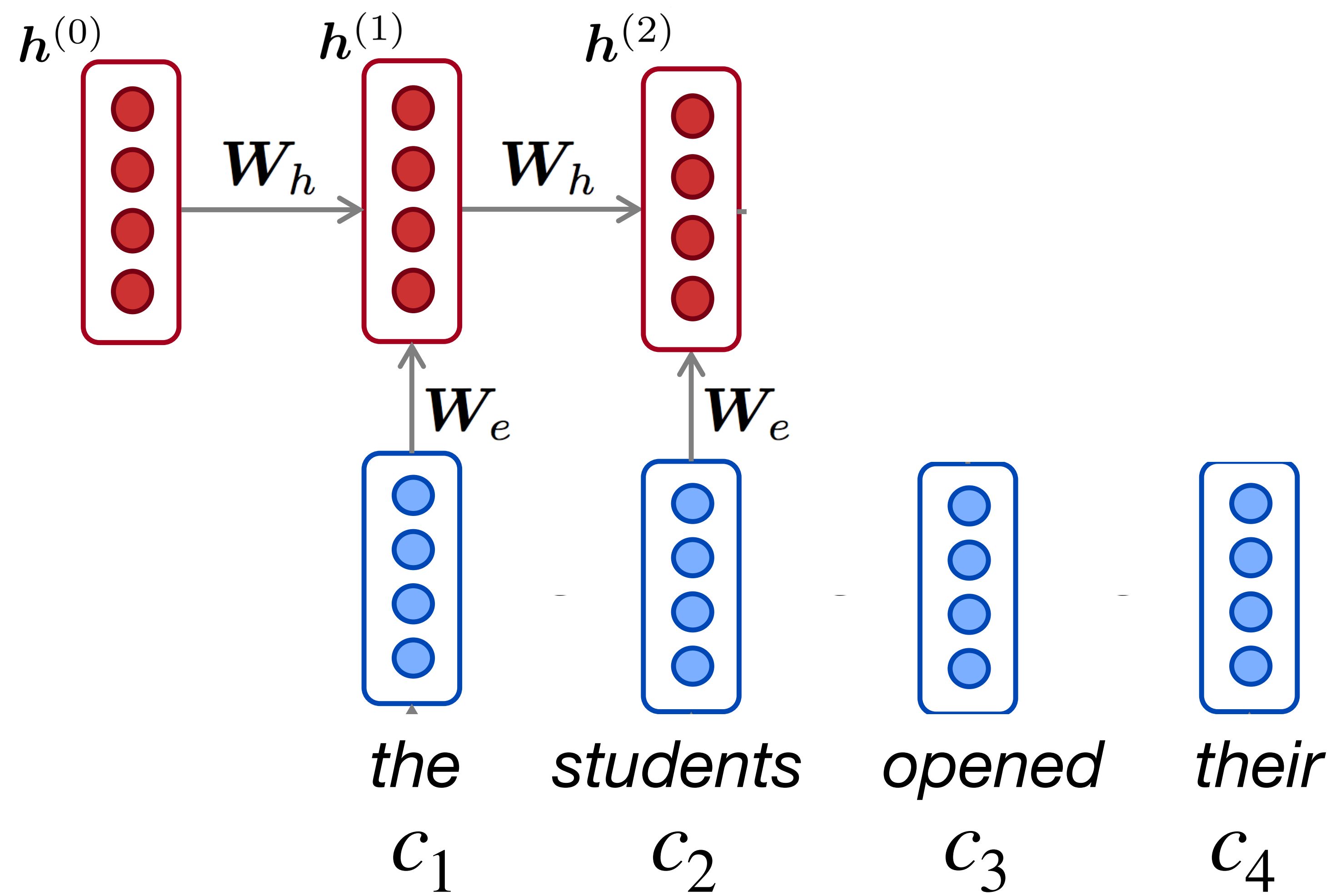
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



A RNN Language Model

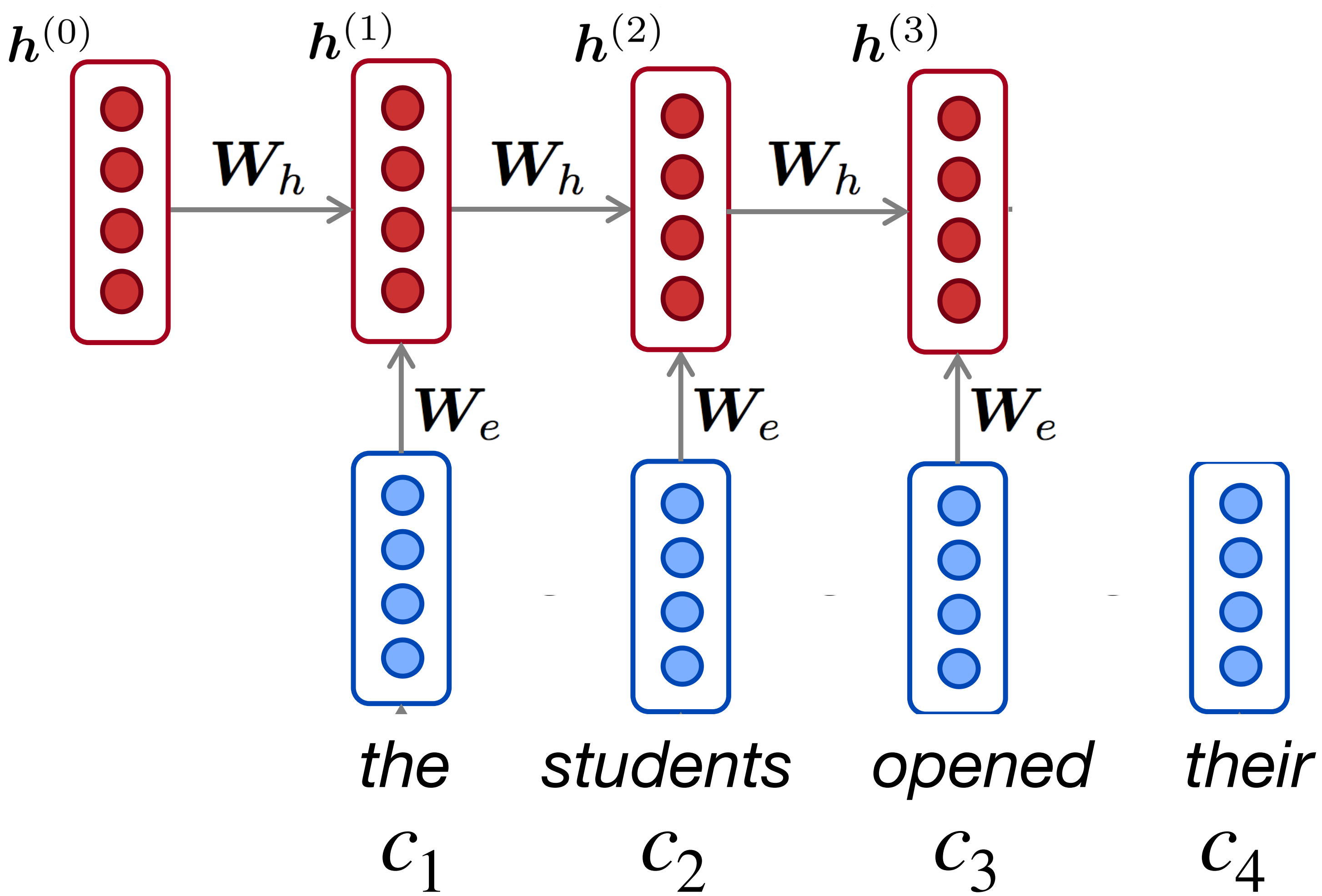
hidden states

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

h(0) is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

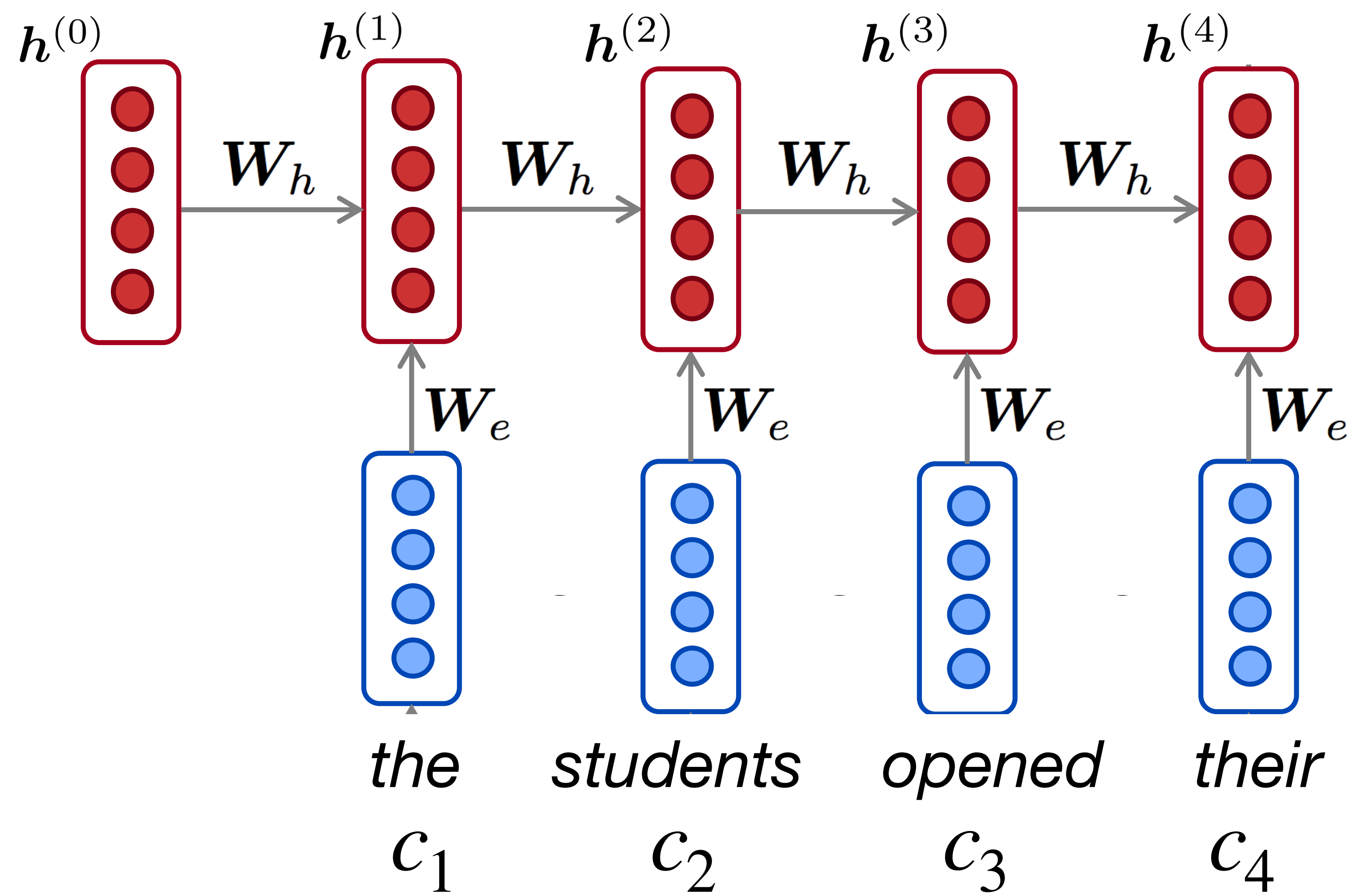
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



A RNN Language Model

$$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)} | \text{the students opened their})$$

output distribution

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

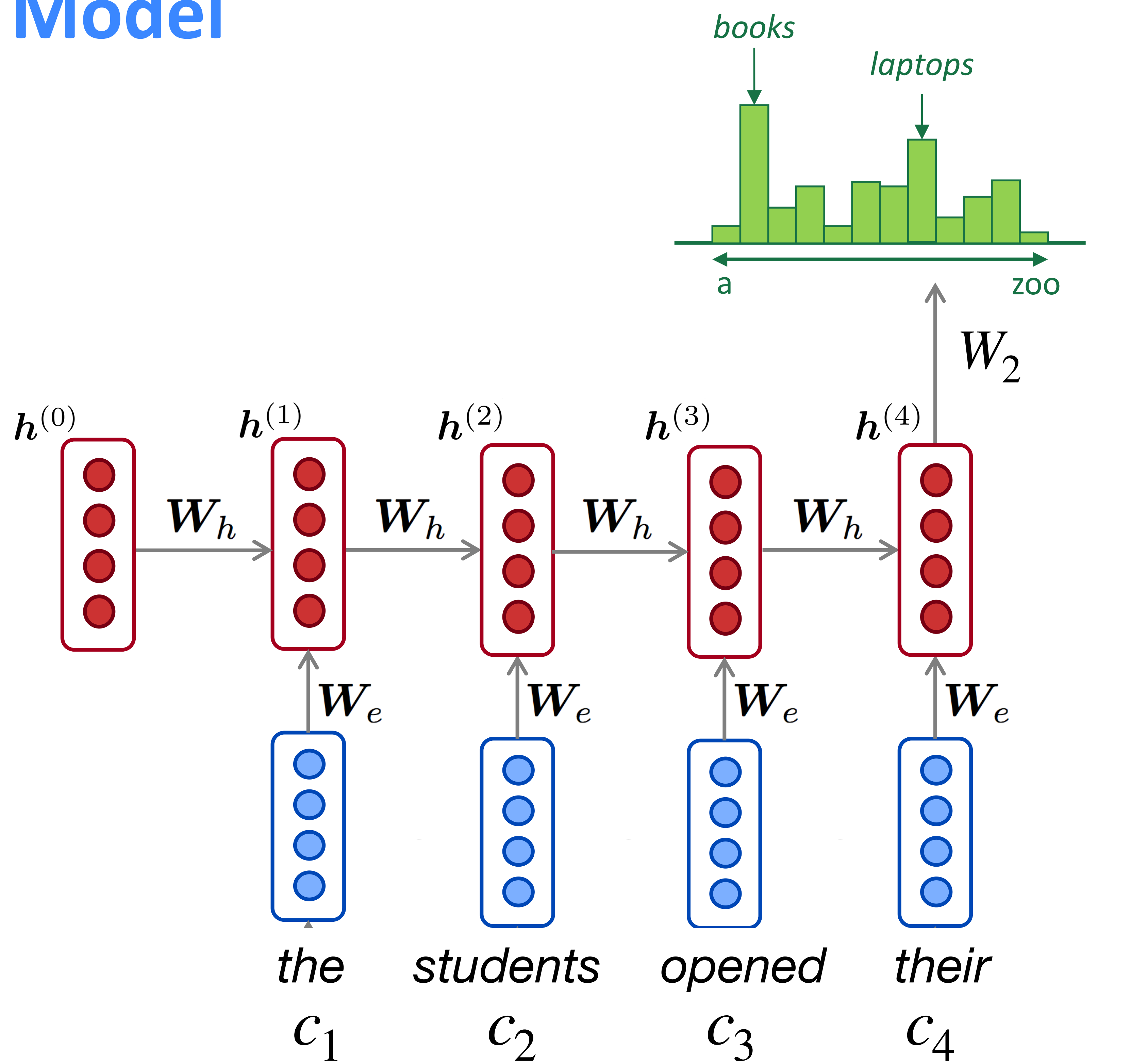
hidden states

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

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

word embeddings

$$c_1, c_2, c_3, c_4$$



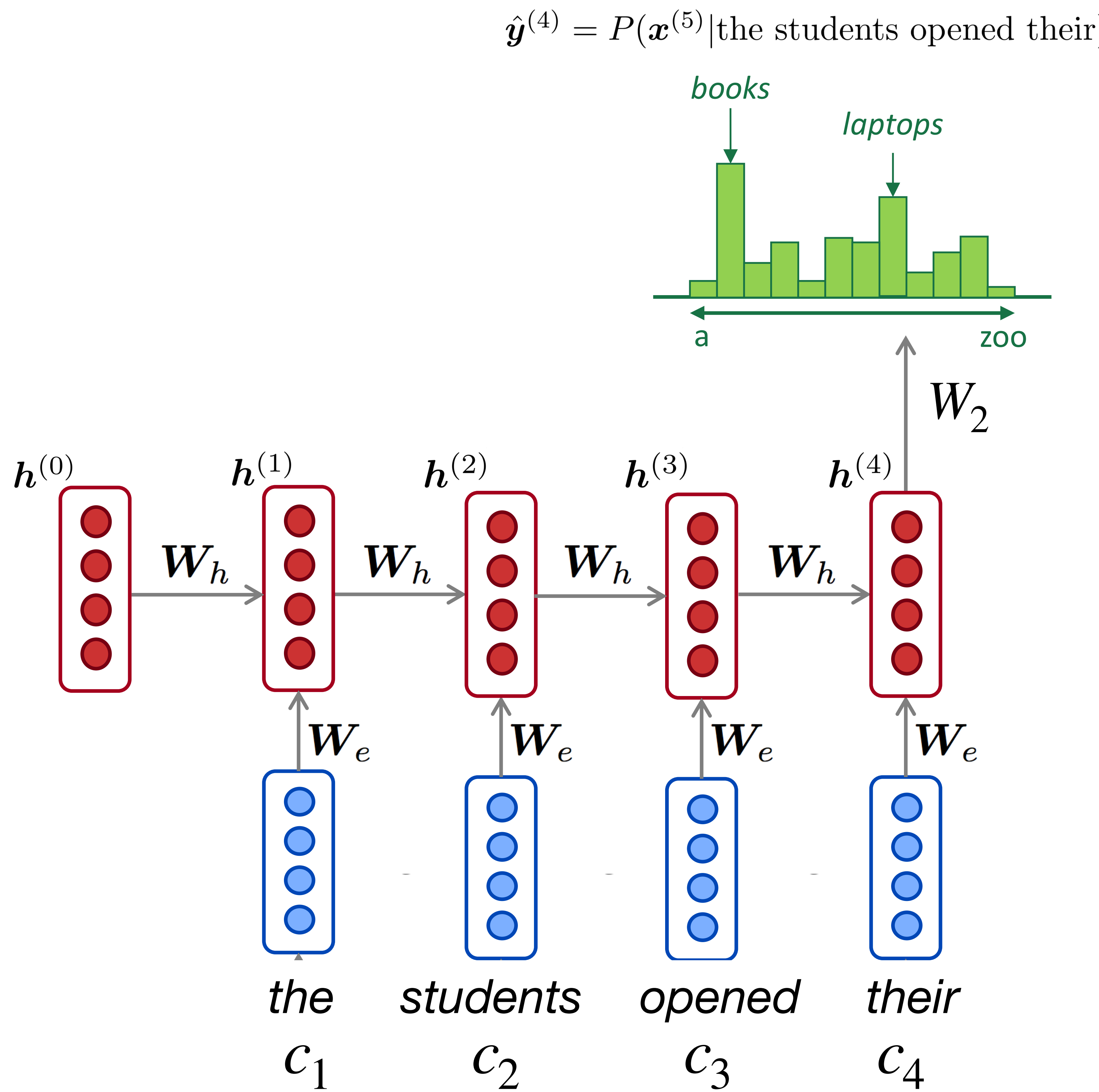
why is this good?

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



- stopped here 9/16

Gradient-based learning

- Goal: learn all model parameters W
- Loss function L(W) based on dataset
- Choose W to minimize L(W) be following the negative gradient of the loss
- Intuition: cross-entropy gradient shifts probability mass to the data

- Next time: gradient learning for arbitrary NN functions; issues and solutions