CS 333: Natural Language Processing

Fall 2023

Prof. Carolyn Anderson
Wellesley College

Computer Science Colloquium Series | Fall 2023

Supporting Responsible AI Practices in Public Sector Contexts

Anna is a third year PhD student at Carnegie Mellon's Human-Computer Interaction Institute. Her research focuses on improving the design, evaluation, and governance of AI technologies used to inform complex, consequential decisions in real-world organizations. In addition to her research, she will share prior experiences forming collaborations with public sector agencies, doing research internships with industry groups, travelling to conferences, and mentoring undergraduate students. The session will end with an open Q/A discussion on applying to and doing a PhD in Computer Science / Human-Computer Interaction and other topics.

WELLESLEY

Accessibility and Disability Resources:
accessibility@wellesley.edu

Anna Kawakami '21

Nov 2nd, 12:45-2:00 | SCI H401
Lunch will be served

Questions??? eni.mustafara

November 2nd

Neural Language Models

language model review

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

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

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

n-gram models

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

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w_j"* never occurred in data? Then 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* w_j*"* never occurred in data? Then w_j has probability 0!

(Partial) Solution: Add small δ to count for every $w_j \in V$. This is called *smoothing*.

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

Slides adapted from Mohit Iyyer

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!

another issue:

students open their eyes

- We treat all words / prefixes independently of each other!

students opened their ___

pupils opened their ___

scholars opened their ___

undergraduates opened their ___

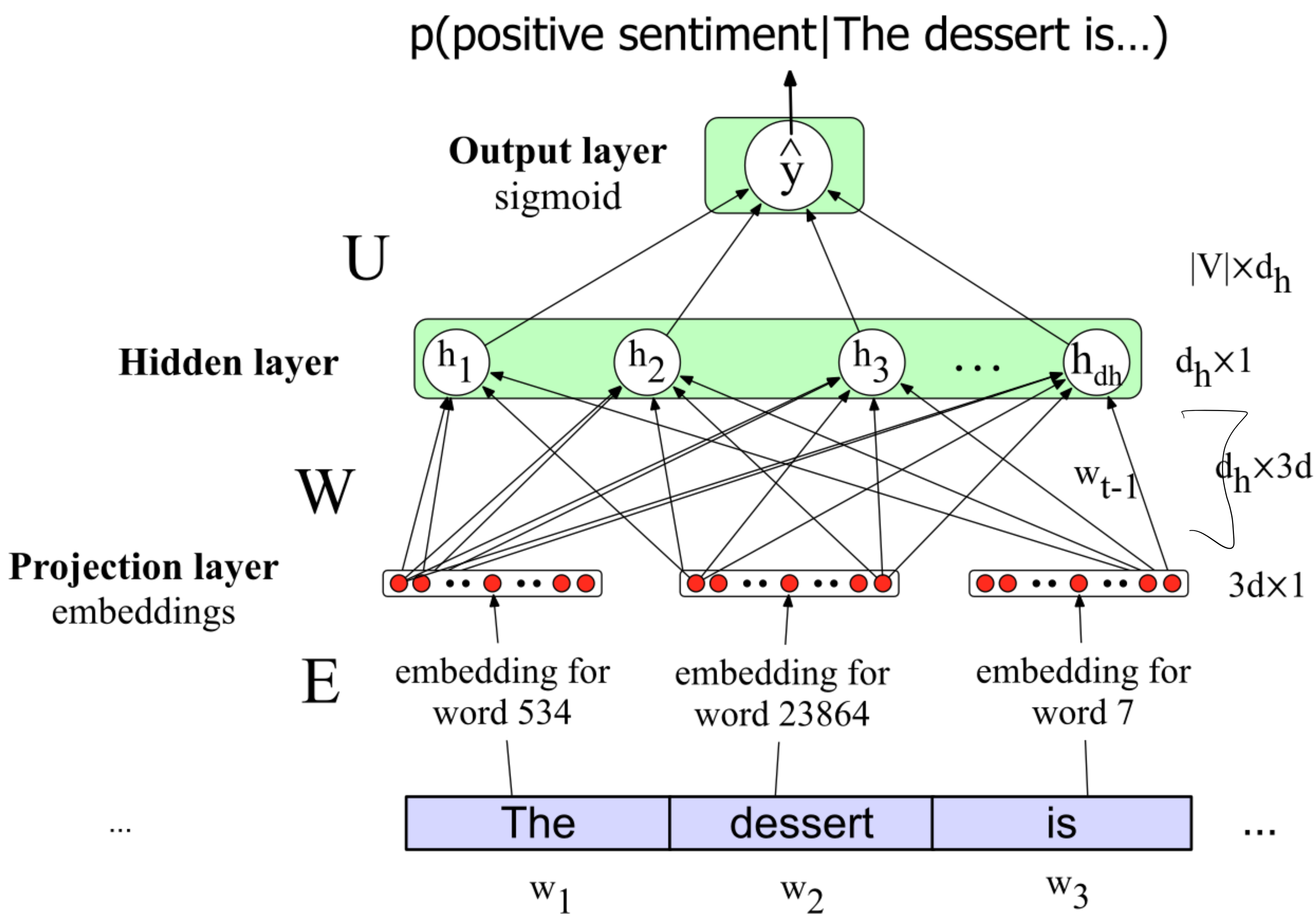
students turned the pages of their ___

students attentively perused their ___

…

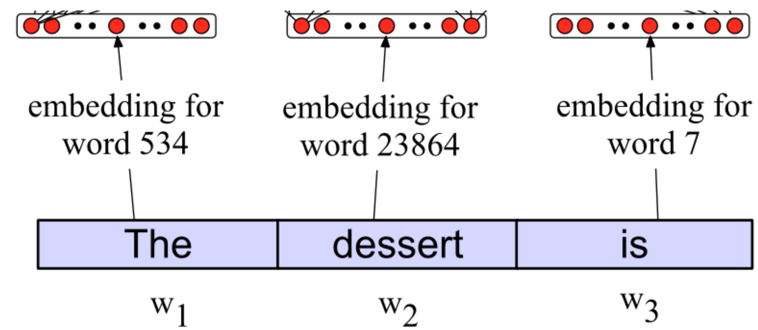
Shouldn't we *share information* across these semantically-similar prefixes?

Neural Net Classification with embeddings as input features!



Issue: texts come in different sizes

This assumes a fixed size length (3)!



len(embedding) × MAX_LEN

Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time

Markov assumption

len(embedding)

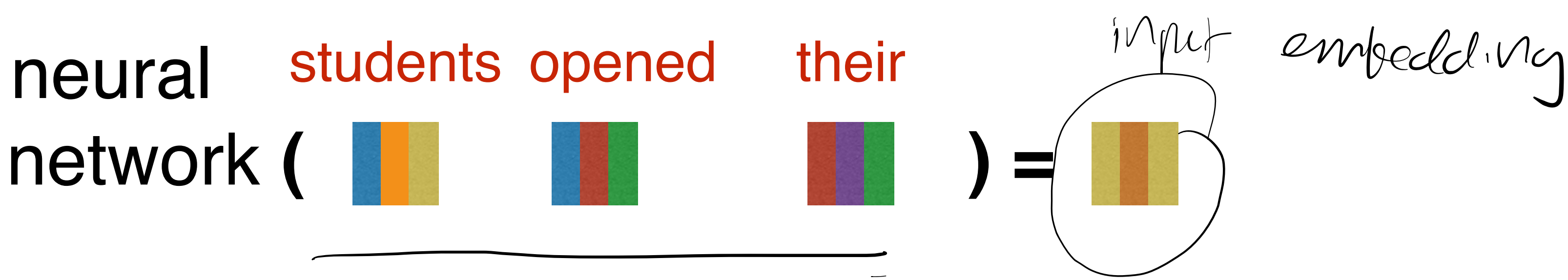
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Bag-of-words

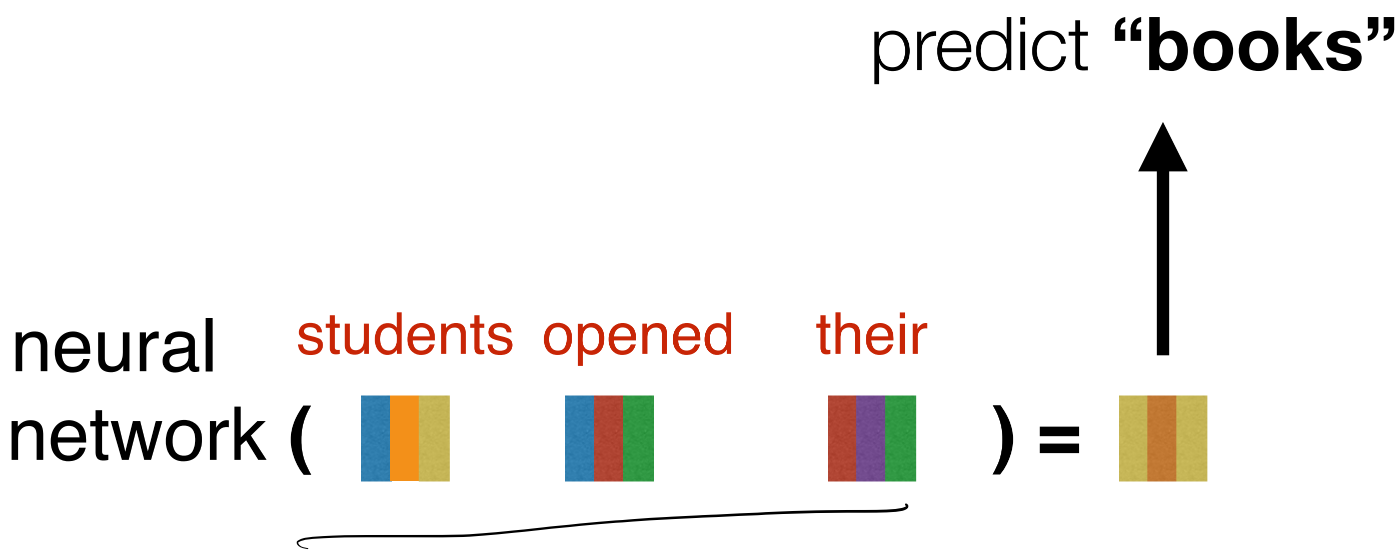
composing embeddings

- neural networks **compose** word embeddings into vectors for phrases, sentences, and documents

1. Compose by concatenating: $len(E) \times n$
2. Compose by averaging $len(E)$



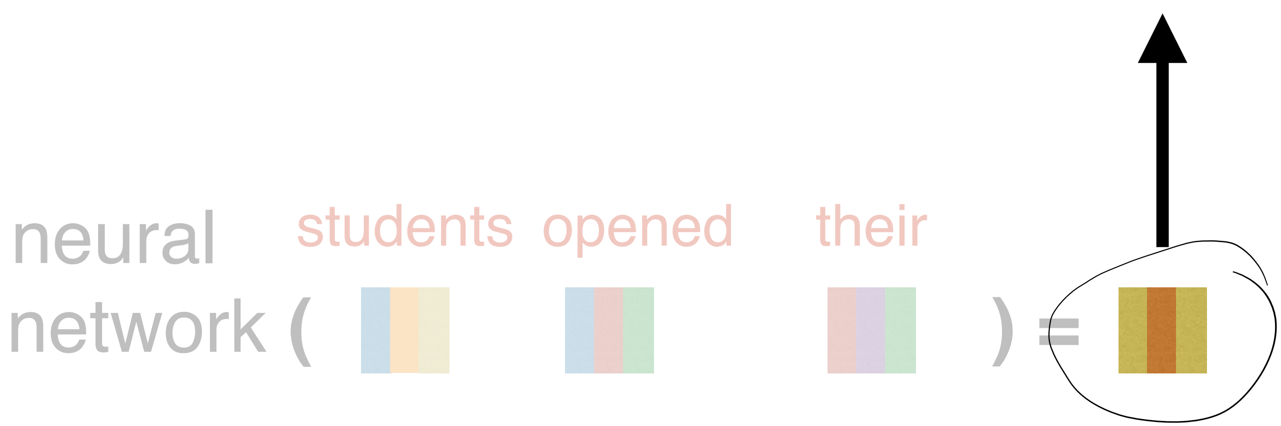
Predict the next word from composed prefix representation



How does this happen? Let's work our way backwards, starting with the prediction of the next word

How does this happen? Let’s work our way backwards, starting with the prediction of the next word

predict **“books”**



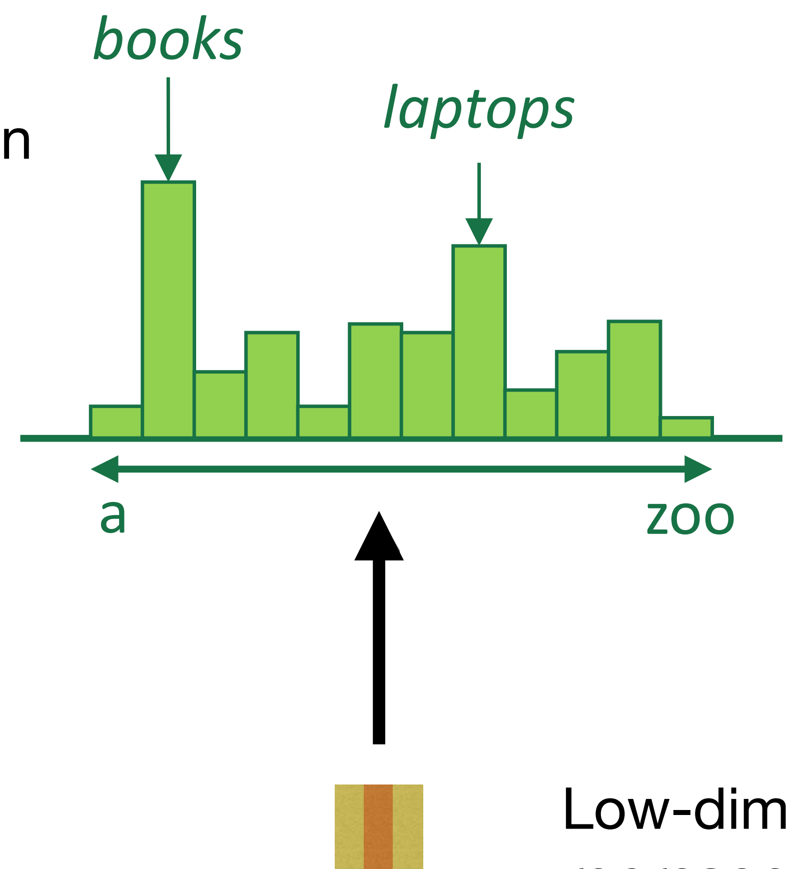
Softmax layer:

convert a vector representation into a probability distribution over the entire vocabulary

Slides adapted from Mohit Iyyer

Vocabulary

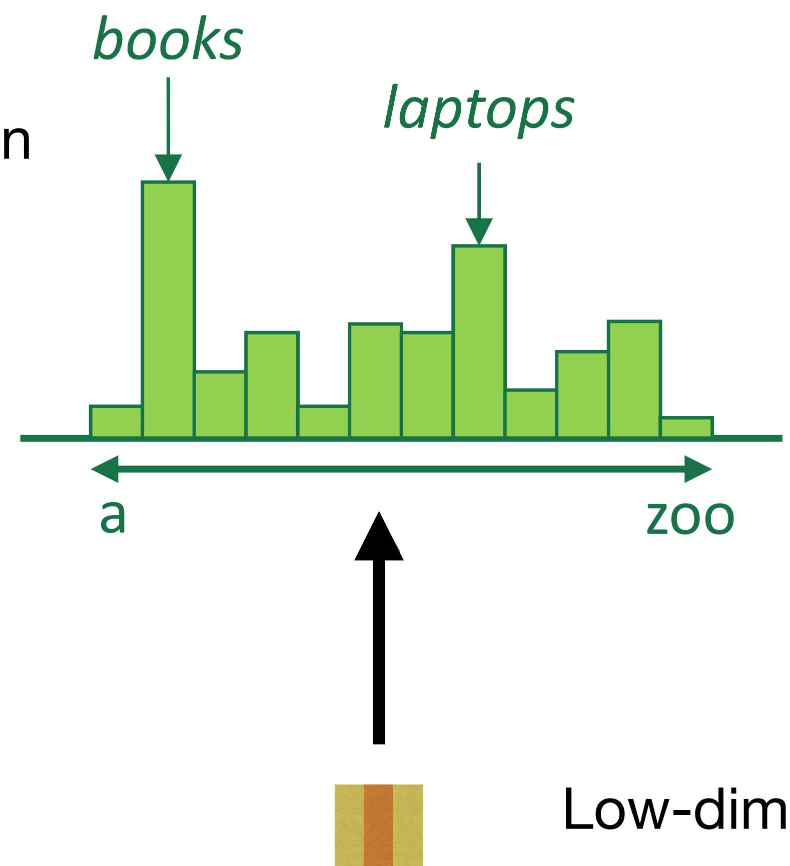
Probability distribution over the entire vocabulary



Low-dimensional representation of "students opened their"

$P(w_i \mid \text{vector for "students opened their"})$

Probability distribution over the entire vocabulary



Low-dimensional representation of “students opened their”

Let’s say our output vocabulary consists of just four words: “books”, “houses”, “lamps”, and “stamps”.

books houses lamps stamps

$\langle 0.6, 0.2, 0.1, 0.1 \rangle$

We want to get a probability distribution over these four words

Let’s say our output vocabulary consists of just four words: “books”, “houses”, “lamps”, and “stamps”.

books	houses	lamps	stamps
<0.6,	0.2,	0.1,	0.1>

We want to get a probability distribution over these four words

SOFTMAX

books	houses	lamps	stamps
0.2	0.05	0.7	0.05

$\langle 0.2, 0.05, 0.7, 0.05 \rangle$

$z = W \cdot X = \langle 1.8, -11.9, 12.9, -8.9 \rangle$

$W \cdot X$ (dot product)

$W_1 X = \langle 1.2 \;\; -0.3 \;\; 0.9 \rangle$

$\langle -2.3, 0.9, 5.4 \rangle$

$= 1.2 * -2.3 + -0.3 * 0.9$

$+ \; 0.9 * 5.4$

$= 1.8$

$$W = \begin{array}{cl} \begin{Bmatrix} 1.2 & -0.3 & 0.9 \\ 0.2 & 0.4 & -2.2 \\ 8.9 & -1.9 & 6.5 \\ 4.5 & 2.2 & -0.1 \end{Bmatrix} & \begin{array}{l} \leftarrow \text{books} \\ \leftarrow \text{houses} \\ \leftarrow \text{lamps} \\ \leftarrow \text{stamps} \end{array} \end{array}$$

X: $\langle -2.3, 0.9, 5.4 \rangle$

not human-interpretable!

"students opened their"

start with a small vector representation of the sentence prefix start with a small vector representation of the sentence prefix

Low-dimensional representation of "students opened their"

just like in regression, we will learn a set of weights

Low-dimensional representation of "students opened their"

$$\mathbf{W} = \left\{ \begin{matrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{matrix} \right\}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

Here’s an example 3-d prefix vector

$$\mathbf{W} = \left\{ \begin{matrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{matrix} \right\}$$

first, we'll project our 3-d prefix representation to 4-d with a matrix-vector product

$$\boldsymbol{x} = <-2.3, 0.9, 5.4>$$

Here's an example 3-d prefix vector

Slides adapted from Mohit Iyyer

$$\mathbf{W} = \left\{ \begin{array}{lll} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{array} \right\}$$

$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$

intuition: each dimension of $\boldsymbol{x}$ corresponds to a *feature* of the prefix

intuition: each row of **W** contains *feature weights* for a corresponding word in the vocabulary

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

intuition: each dimension of ***x*** corresponds to a *feature* of the prefix

intuition: each row of **W** contains *feature weights* for a corresponding word in the vocabulary

$$\mathbf{W} = \begin{array}{cl} \left\{\begin{array}{ccc} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{array}\right\} & \begin{array}{l} \text{books} \\ \text{houses} \\ \text{lamps} \\ \text{stamps} \end{array} \end{array}$$

$$\boldsymbol{x} = <-2.3, 0.9, 5.4>$$

intuition: each dimension of ***x*** corresponds to a *feature* of the prefix

intuition: each row of **W** contains *feature weights* for a corresponding word in the vocabulary

CAUTION: we can't easily *interpret* these features! For example, the second dimension of ***x*** likely does not correspond to any linguistic property

$$\mathbf{W} = \left\{ \begin{array}{ccc} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{array} \right\} \begin{array}{l} \text{books} \\ \text{houses} \\ \text{lamps} \\ \text{stamps} \end{array}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

intuition: each dimension of ***x*** corresponds to a *feature* of the prefix

Slides adapted from Mohit Iyyer

$$\mathbf{W} = \left\{ \begin{matrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{matrix} \right\}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

now we compute the output for this layer by taking the dot product between x and W

$\mathbf{W}\boldsymbol{x}$ = <1.8, -11.9, 12.9, -8.9>

How did we compute this? Just the dot product of each row of **W** with ***x***!

$$\mathbf{W} = \left\{ \begin{matrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{matrix} \right\}$$

1.2 * -2.3
+ -0.3 * 0.9
+ 0.9 * 5.4

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

Slides adapted from Mohit Iyyer

Okay, so how do we go from this 4-d vector to a probability distribution?

$\mathbf{W}\boldsymbol{x} = \langle 1.8, -11.9, 12.9, -8.9 \rangle$

$$\mathbf{W}\boldsymbol{x} = \langle 1.8,\ -11.9,\ 12.9,\ -8.9 \rangle$$

books houses lamps stamps

RESULT: <0.6, 0.2, 0.1, 0.1>

$$\mathbf{W}\boldsymbol{x} = \langle 1.8,\ -11.9,\ 12.9,\ -8.9 \rangle$$

	books	houses	lamps	stamps
RESULT:	0.6	0.2	0.1	0.1

$$\langle 0.6,\ 0.2,\ 0.1,\ 0.1 \rangle$$

Given a *d*-dimensional vector representation ***x*** of a prefix, we do the following to predict the next word:

1. Project it to a *V*-dimensional vector using a matrix-vector product (a.k.a. a “linear layer”, or a “feedforward layer”), where *V* is the size of the vocabulary

2. Apply the softmax function to transform the resulting vector into a probability distribution

So far, this is just multi-class regression on word embeddings!

Now that we know how to predict **“books”**, let’s focus on how to compute the prefix representation ***x*** in the first place!

Composition functions

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

output: single vector

- Element-wise functions
  - e.g., just sum up all of the word embeddings!
- Concatenation
- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Transformers

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

A fixed-window neural Language Model

$$\hat{y} = \text{softmax}(W_3 h_2)$$

$$h_2 = f(W_2 h_1)$$

$$h_1 = f(W_1 x)$$

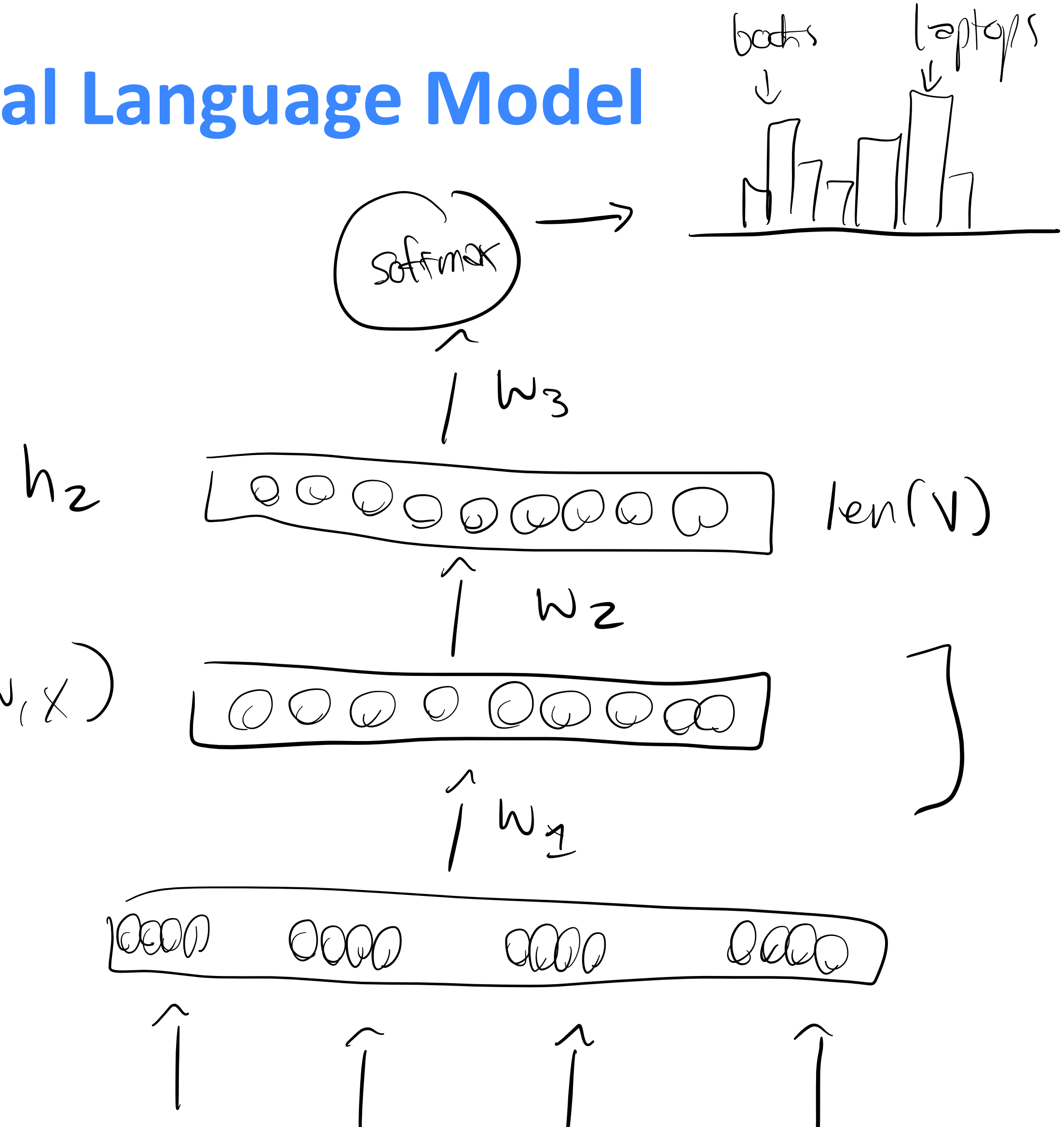
f: ReLU

tanh

sigmoid

~~softmax~~

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



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

discard

fixed window

c_1 c_2 c_3 c_4

Slides adapted from Mohit Iyyer

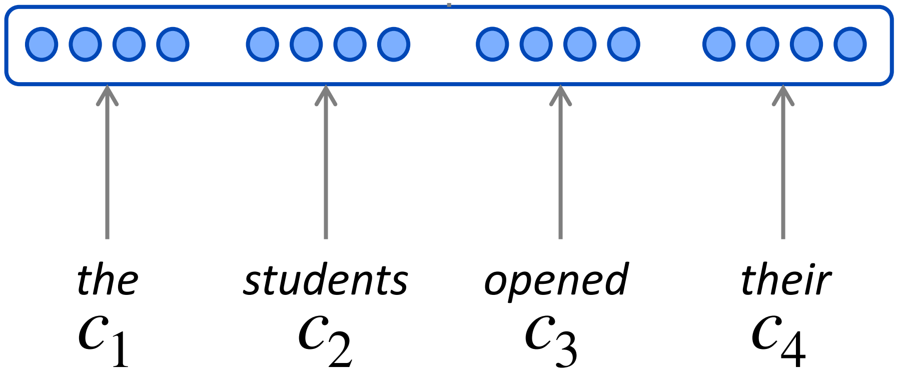
A fixed-window neural Language Model

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



Slides adapted from Mohit Iyyer

A fixed-window neural Language Model

hidden layer

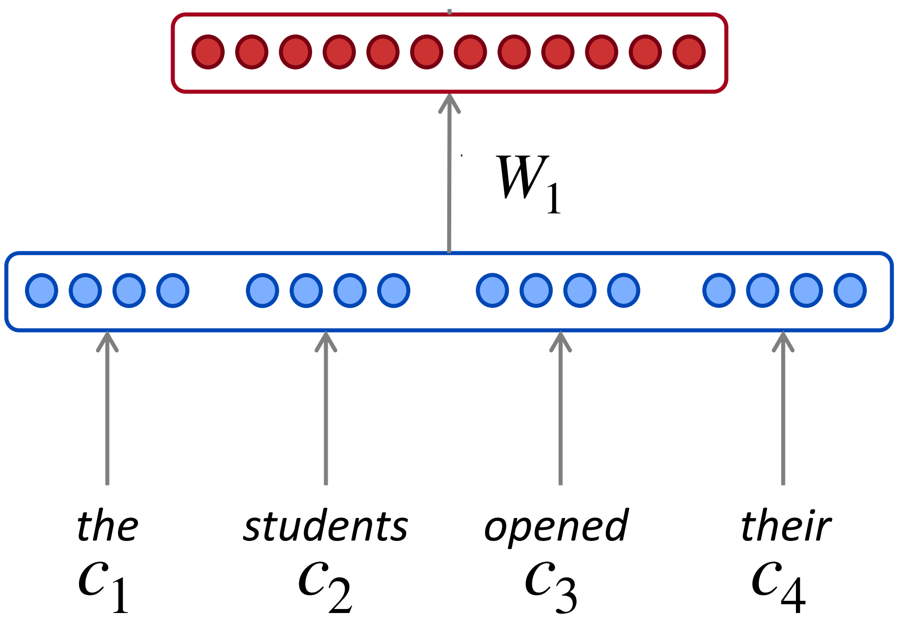
$$h = f(W_1 x)$$

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



Slides adapted from Mohit Iyyer

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

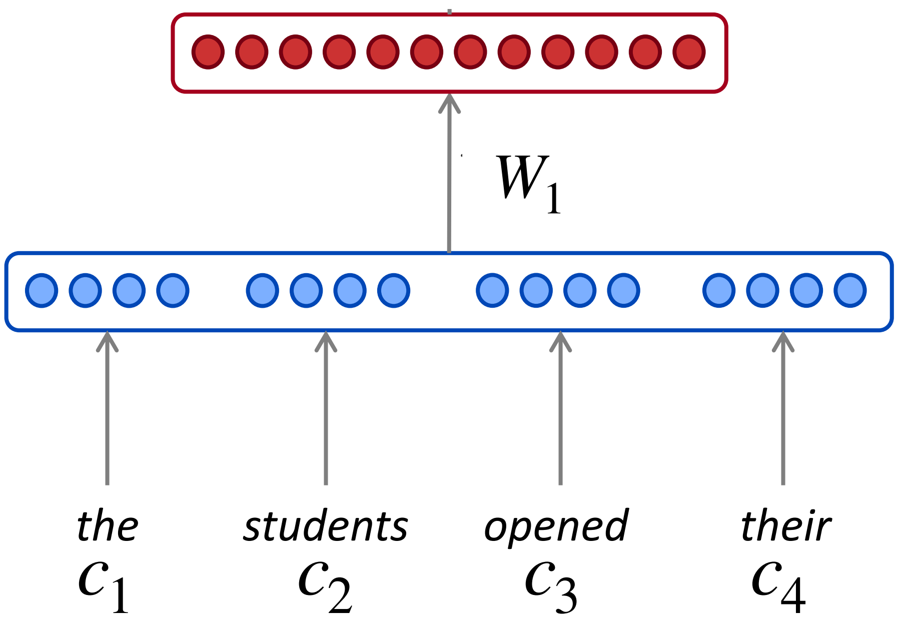
$$h = f(W_1 x)$$

concatenated word embeddings

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

words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



Slides adapted from Mohit Iyyer

A fixed-window neural Language Model

output distribution

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

hidden layer

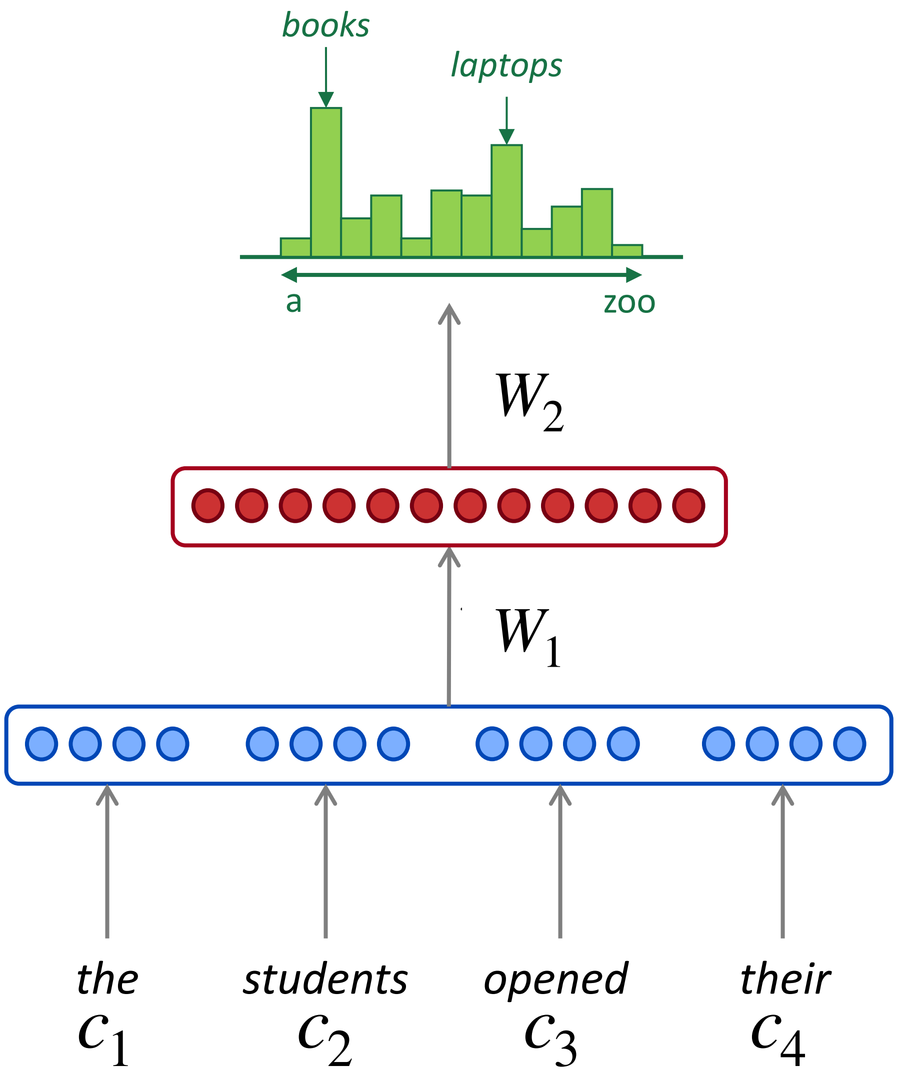
$$h = f(W_1 x)$$

concatenated word embeddings

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

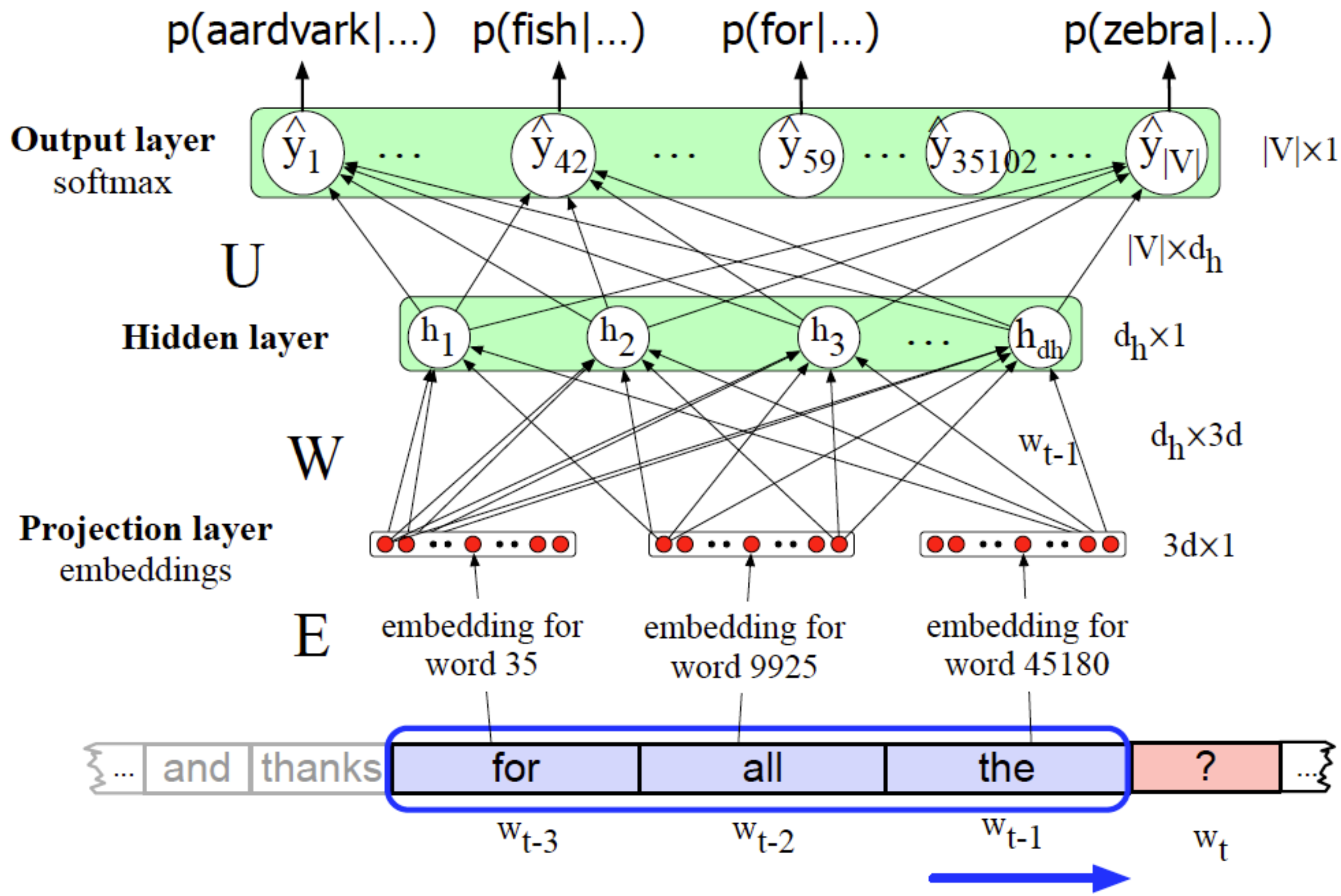
words / one-hot vectors

$$c_1, c_2, c_3, c_4$$



Slides adapted from Mohit Iyyer

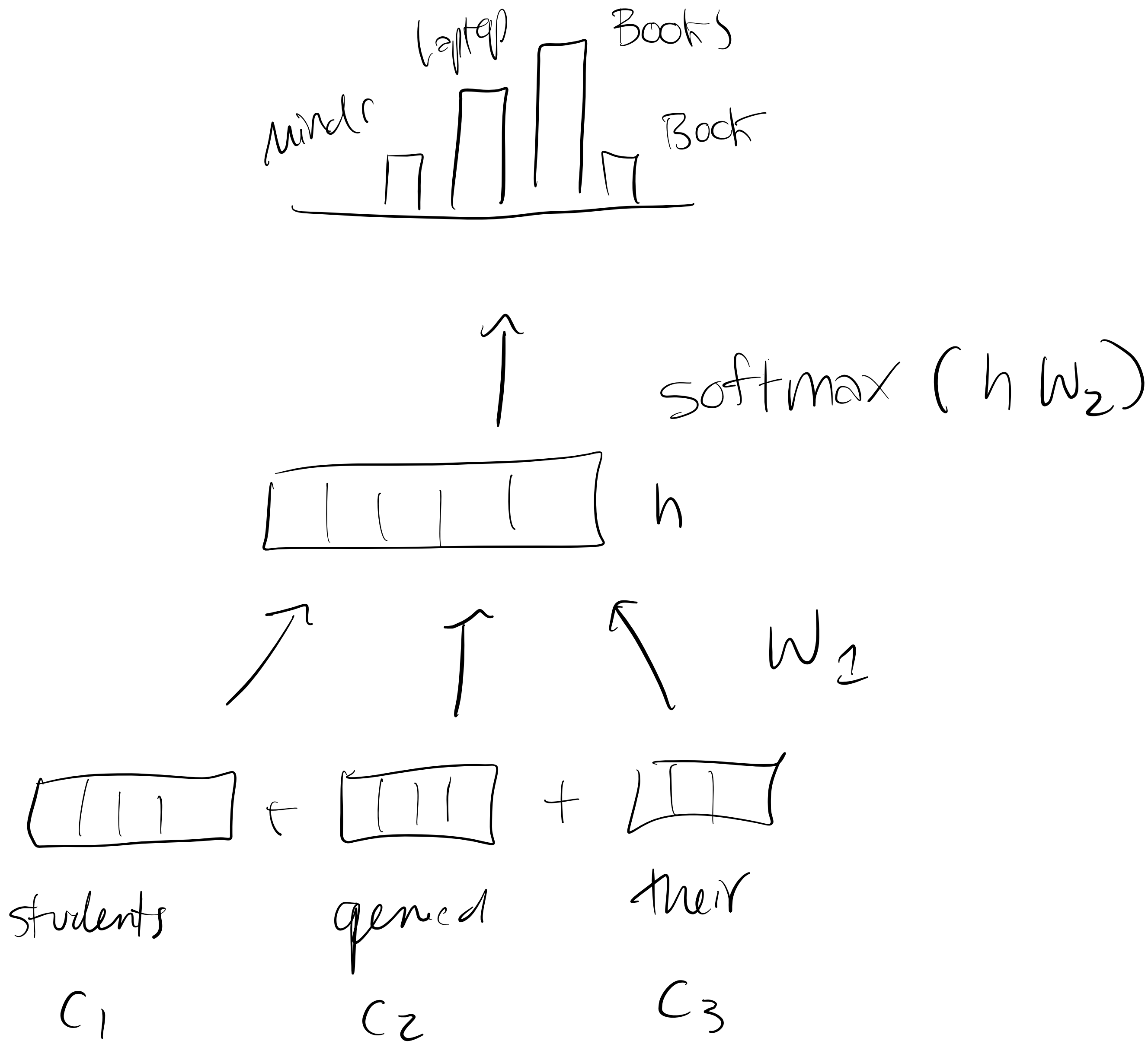
Neural Language Model



Slides borrowed from Jurafsky & Martin Edition 3

Training a Fixed-Length Neural Language Model

Goal: given "students open their", predict "books"



Key Question: what are the parameters?

W_2: pre-softmax weights

W_1: input weights

C_1, C_2, C_3: word embeddings

1. randomly initialize
2. learn weight by updating during training