Neural networks & Neural language models

CS 4804: Introduction to AI

Fall 2025

https://tuvllms.github.io/ai-fall-2025/

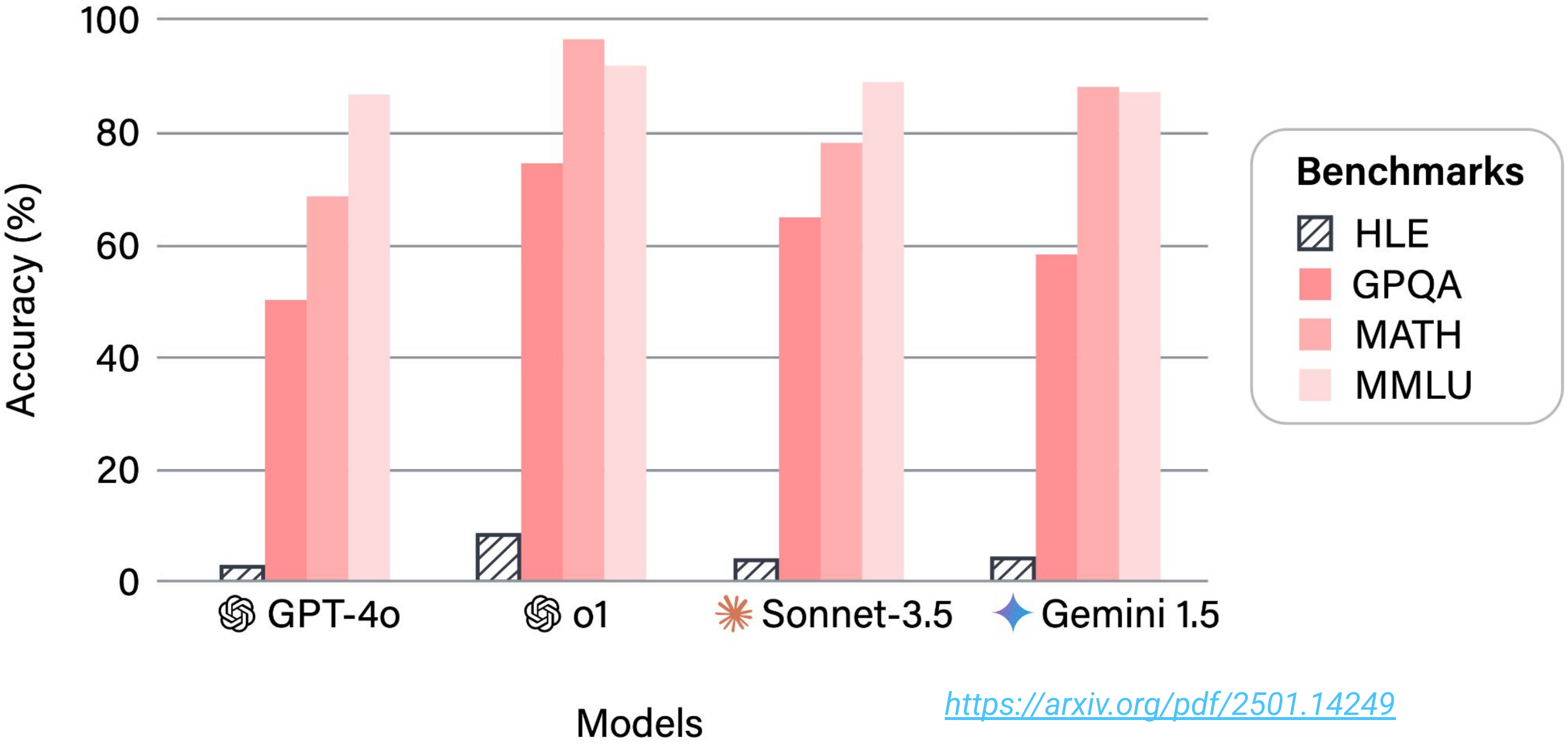
Tu Vu

VIRGINIA TECH.

Logistics

- Office hours starting this week
  - both in-person and via Zoom (links available on Piazza)
- Homework 0 released (due September 16th)
- Final project group
  - Search for teammates on Piazza https://piazza.com/class/meqiibrwtql168/post/5 or reach out to us at cs4804instructors@gmail.com
  - Google form for submitting group information available on Piazza (due September 5th)

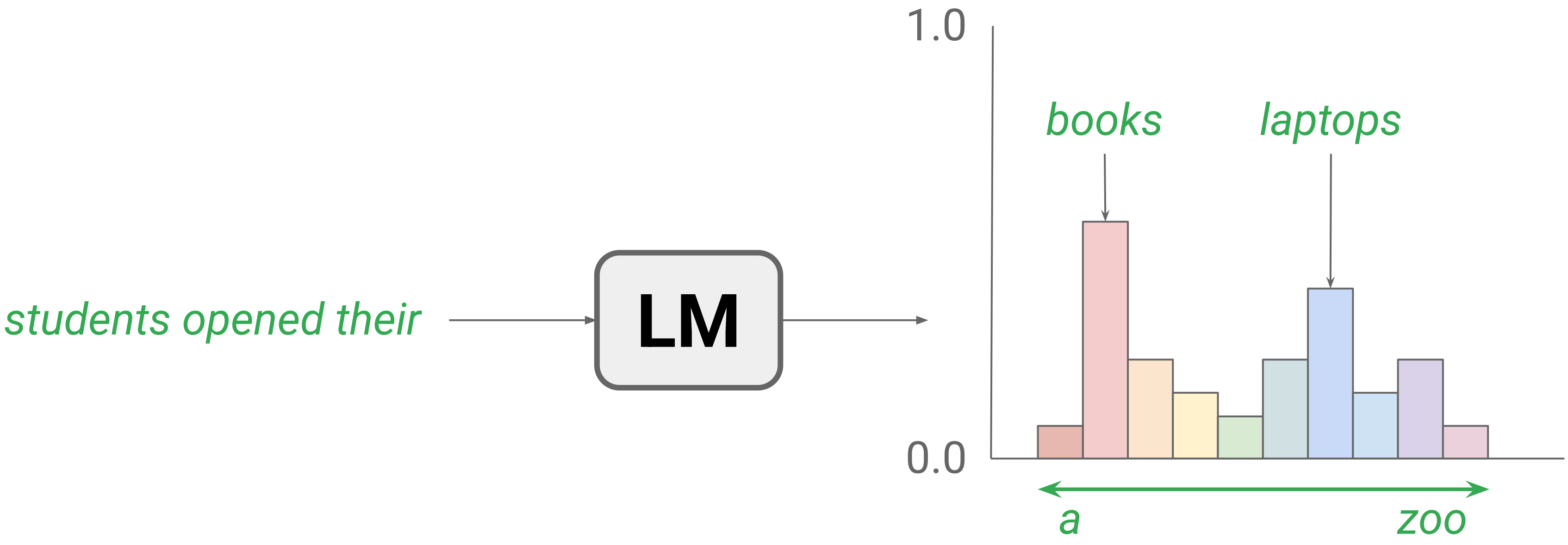
Humanity's Last Exam



https://arxiv.org/pdf/2501.14249

Language modeling review

- Predict **the next word**, or a ***probability distribution*** over possible next words



Language modeling review (cont'd)

- Language models
  - compute

$$P(w_1, w_2, \ldots, w_n)$$

or

$$P(w_j | w_1, w_2, \ldots, w_{j-1})$$

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2 | w_1) \times \ldots \times P(w_n | w_1, w_2, \ldots, w_{n-1})$$

N-gram language models review

- Use maximum likelihood estimation (MLE)

$$\boldsymbol{P}(\text{“laptops”} \mid \text{“students opened their”}) = \frac{\boldsymbol{Count}(\text{“students opened their laptops”})}{\boldsymbol{Count}(\text{“students opened their”})}$$

Perplexity

$$\text{perplexity}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

We normalize by the number of words N by taking the Nth root

Or we can use the chain rule to expand the probability of W:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

Perplexity as Weighted Average Branching Factor

- Suppose a sentence consists of random digits.
  What is the perplexity of this sentence for a model that assigns a probability of 1/10 to each digit?

$$\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10^N}\right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}$$

Given any prefix, how many next words does the model consider reasonable?

In practice, we use log probs

$$log \prod p(w_i|w_{i-1}) = \sum logp(w_i|w_{i-1})$$

logs to avoid numerical underflow

sentence: I love love love love love the movie

$$p(\text{i}) \cdot p(\text{love})^5 \cdot p(\text{the}) \cdot p(\text{movie}) \quad = 5.95374181\text{e-}7$$

$$\log p(\text{i}) + 5 \log p(\text{love}) + \log p(\text{the}) + \log p(\text{movie})$$

$$= -14.3340757538$$

source: Mohit Iyyer

In practice, we use log probs (cont'd)

$$perplexity(W) = exp(-\frac{1}{N}\sum_{i}^{N} logp(w_i|w_{<i}))$$

perplexity is the exponentiated token-level negative log-likelihood

Problems with n-gram language models

$$P(\text{“laptops”} \mid \text{“students opened their”}) = \frac{\mathbf{Count}(\text{“students opened their laptops”})}{\mathbf{Count}(\text{“students opened their”})}$$

What if *“students opened their laptops”* *never occurred in training data?*

Problems with n-gram language models (cont'd)

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Need to store V^n counts for an n-gram model!

source: Jurafsky and Martin

Problems with n-gram language models (cont'd)

- Treat semantically similar prefixes independently of each other

"students opened their ___"

"pupils opened their ___"

"scholars opened their ___"

"students began reading their ___"

Shouldn't we share information across these prefixes?

Matrix-vector multiplication

Matrix A (dimensions 4×3):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

Vector x (dimensions 3×1):

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Resulting vector b (dimensions 4×1):

$$b = A \cdot x = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 \end{bmatrix}$$

Softmax function

For a vector $y = [y_1, y_2, \ldots, y_V]$ of dimension V, the softmax transformation is calculated as:

$$\text{softmax}(y) = \left[\frac{e^{y_1}}{\sum e^y}, \frac{e^{y_2}}{\sum e^y}, \ldots, \frac{e^{y_V}}{\sum e^y}\right]$$

where $\sum e^y = e^{y_1} + e^{y_2} + \cdots + e^{y_V}$.

Word representations / embeddings

- High-dimensional / sparse / one-hot representations
- Low-dimensional / dense representations

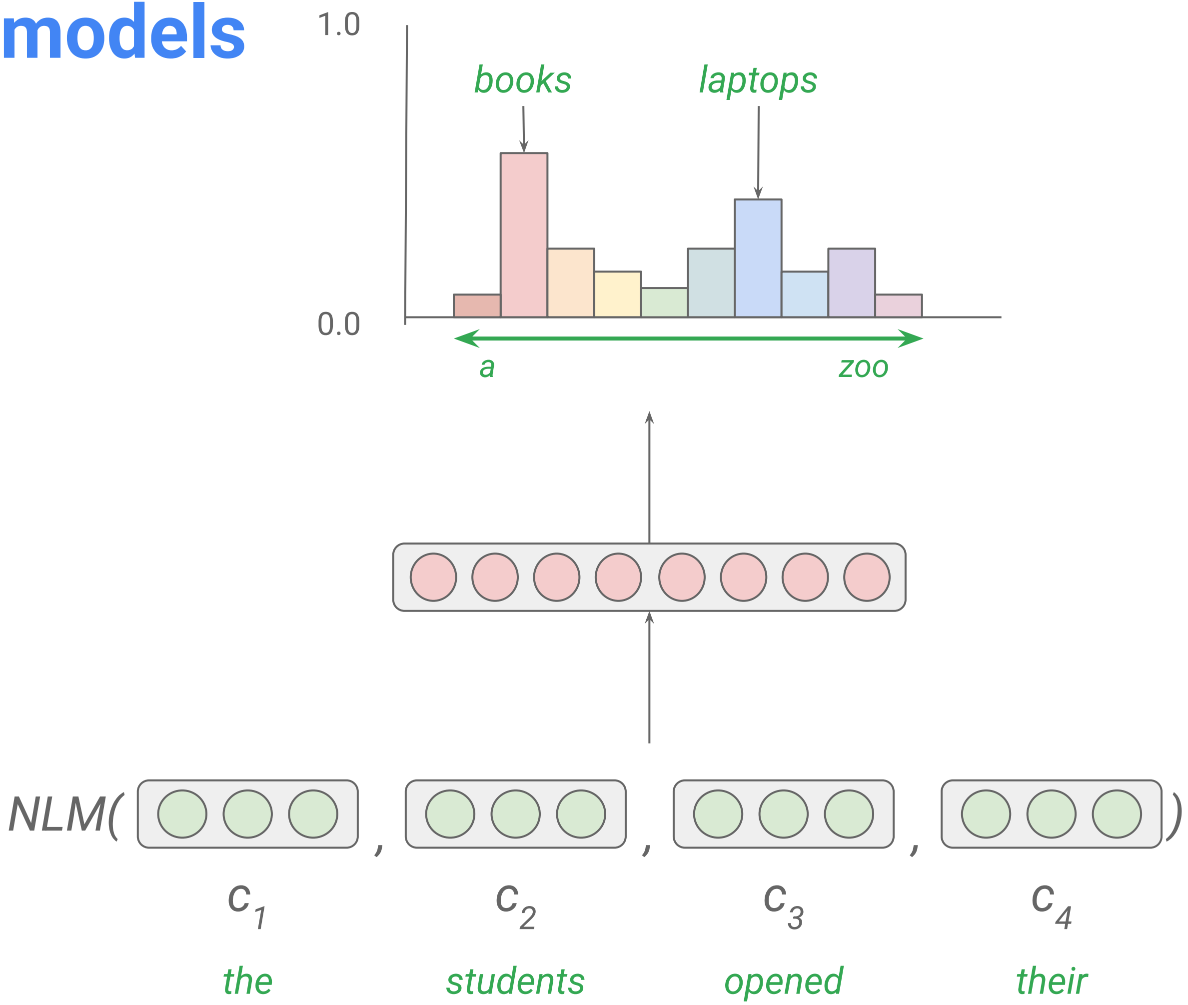
Word representations / embeddings (cont'd)

```
#What is the vector representation for a word?
w2v_model['computer']
```

```
array([ 1.07421875e-01, -2.01171875e-01,  1.23046875e-01,  2.11914062e-01,
       -9.13085938e-02,  2.16796875e-01, -1.31835938e-01,  8.30078125e-02,
        2.02148438e-01,  4.78515625e-02,  3.66210938e-02, -2.45361328e-02,
        2.39257812e-02, -1.60156250e-01, -2.61230469e-02,  9.71679688e-02,
       -6.34765625e-02,  1.84570312e-01,  1.70898438e-01, -1.63085938e-01,
       -1.09375000e-01,  1.49414062e-01, -4.65393066e-04,  9.61914062e-02,
        1.68945312e-01,  2.60925293e-03,  8.93554688e-02,  6.49414062e-02,
        3.56445312e-02, -6.93359375e-02, -1.46484375e-01, -1.21093750e-01,
       -2.27539062e-01,  2.45361328e-02, -1.24511719e-01, -3.18359375e-01,
       -2.20703125e-01,  1.30859375e-01,  3.66210938e-02, -3.63769531e-02,
       -1.13281250e-01,  1.95312500e-01,  9.76562500e-02,  1.26953125e-01,
        6.59179688e-02,  6.93359375e-02,  1.02539062e-02,  1.75781250e-01,
       -1.68945312e-01,  1.21307373e-03, -2.98828125e-01, -1.15234375e-01,
        5.66406250e-02, -1.77734375e-01, -2.08984375e-01,  1.76757812e-01,
        2.38037109e-02, -2.57812500e-01, -4.46777344e-02,  1.88476562e-01,
        5.51757812e-02,  5.02929688e-02, -1.06933594e-01,  1.89453125e-01,
       -1.16210938e-01,  8.49609375e-02, -1.71875000e-01,  2.45117188e-01,
       -1.73828125e-01, -8.30078125e-03,  4.56542969e-02, -1.61132812e-02,
        1.86523438e-01, -6.05468750e-02, -4.17480469e-02,  1.82617188e-01,
        2.20703125e-01, -1.22558594e-01, -2.55126953e-02, -3.08593750e-01,
        9.13085938e-02,  1.60156250e-01,  1.70898438e-01,  1.19628906e-01,
        7.08007812e-02, -2.64892578e-02, -3.08837891e-02,  4.06250000e-01,
       -1.01562500e-01,  5.71289062e-02, -7.26318359e-03, -9.17968750e-02,
       -1.50390625e-01, -2.55859375e-01,  2.16796875e-01, -3.63769531e-02,
        2.24609375e-01,  8.00781250e-02,  1.56250000e-01,  5.27343750e-02,
```

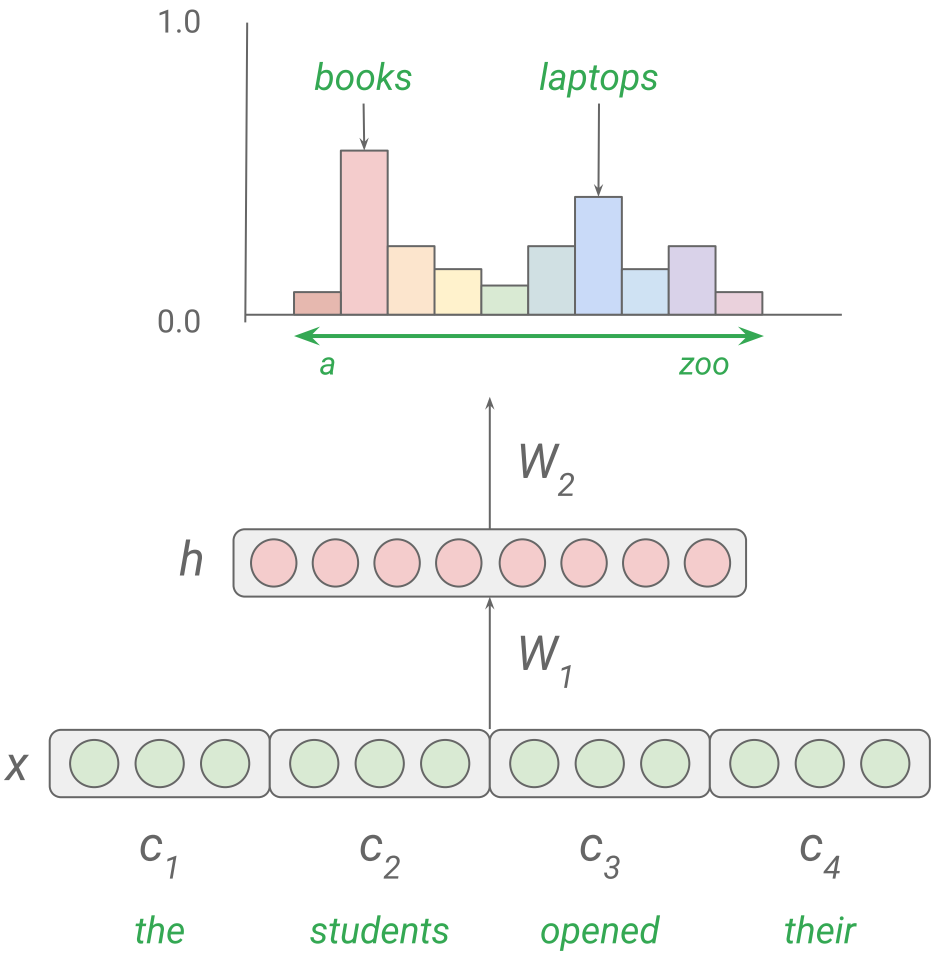
Connected to Python 3 Google Compute Engine backend

Neural language models



output distribution

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



Composition functions

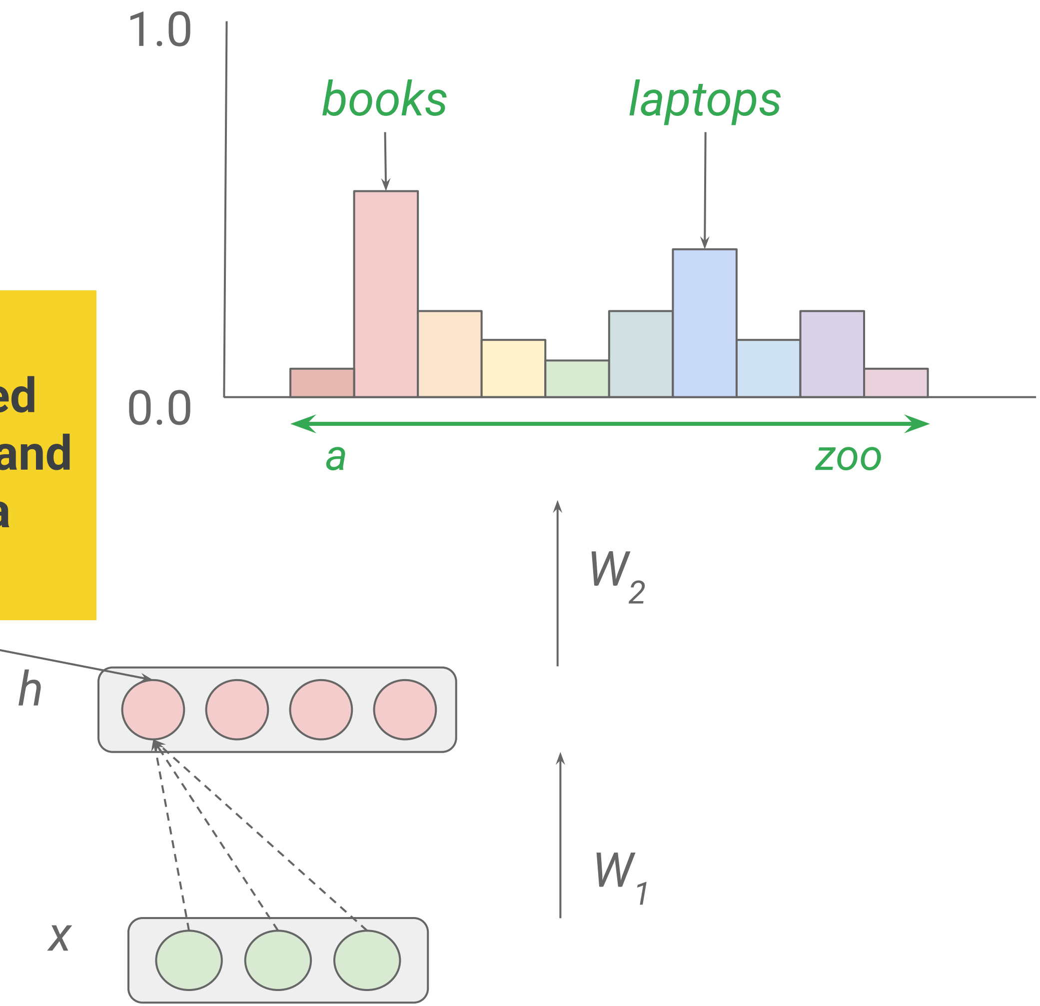
- Element-wise functions
  - e.g., just sum up all of the word embeddings
- Concatenation
- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Transformers

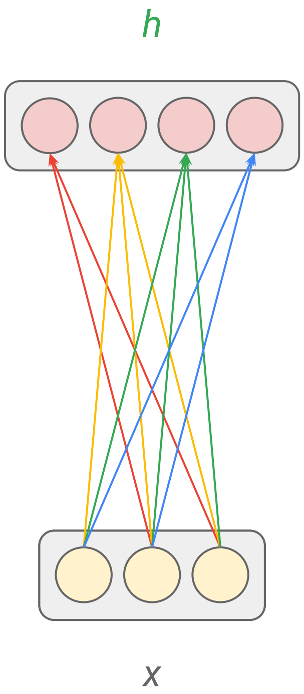
Feedforward neural language model

hidden layer

$h = f(W_1 x)$

hidden unit: taking a weighted sum of its inputs and then applying a non-linearity





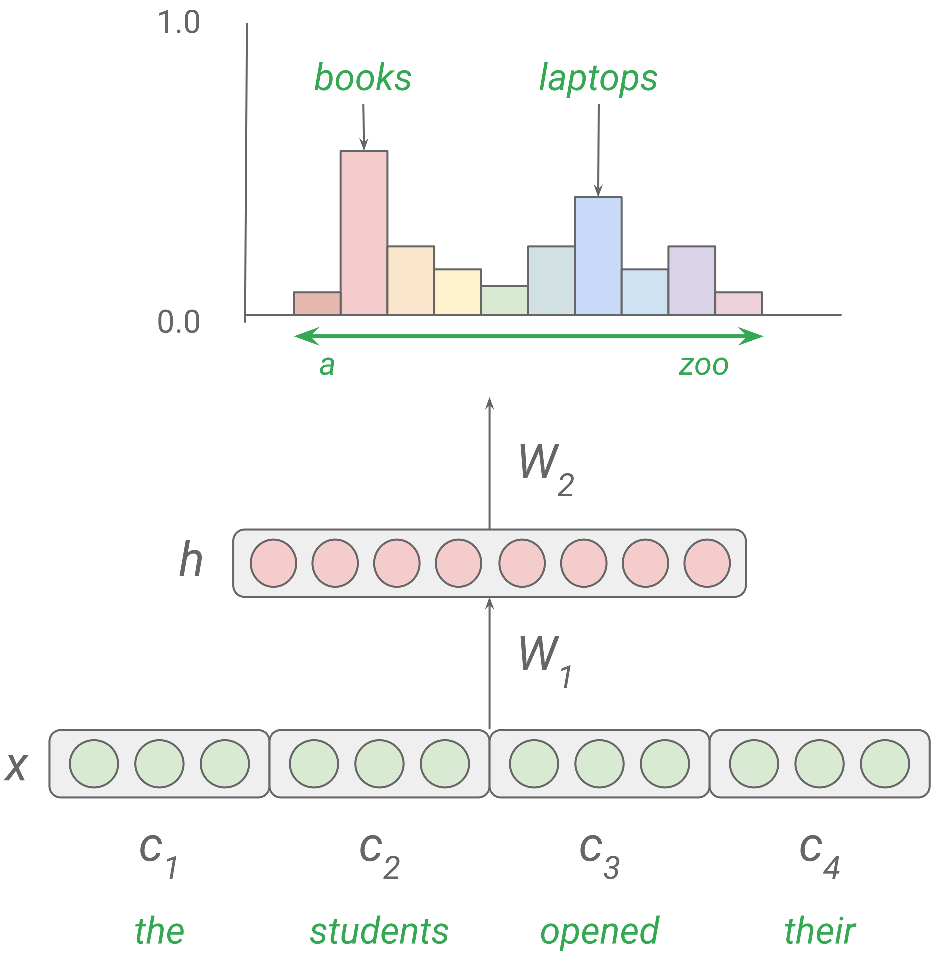
hidden layer

$$h = f(W_1 x)$$

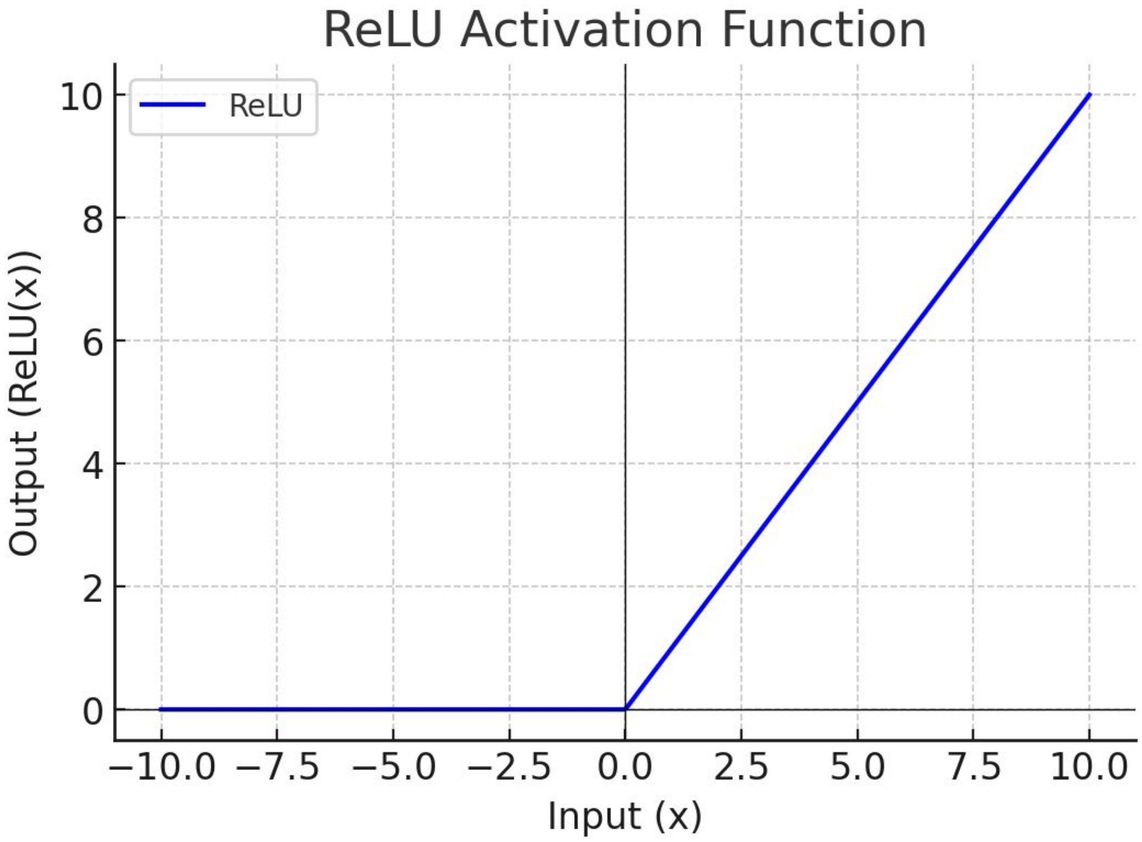
f is a non-linear activation function to model non-linear relationships between words

output distribution

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

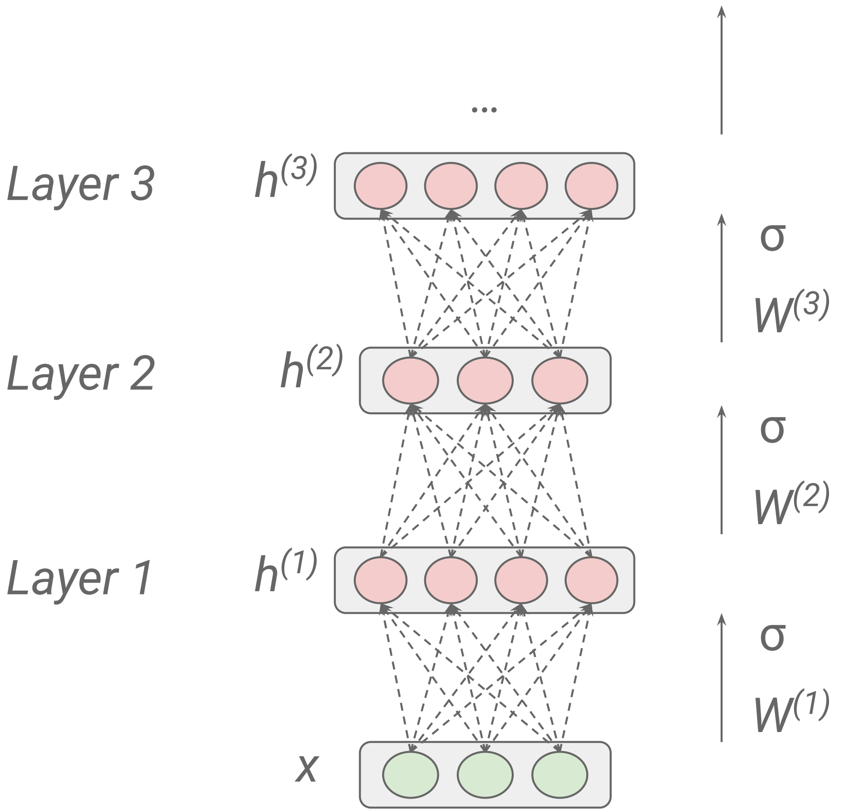


Activation functions



Rectified
Linear Unit
(ReLU)

Deep neural networks



hierarchical representations, where each layer builds upon the previous one

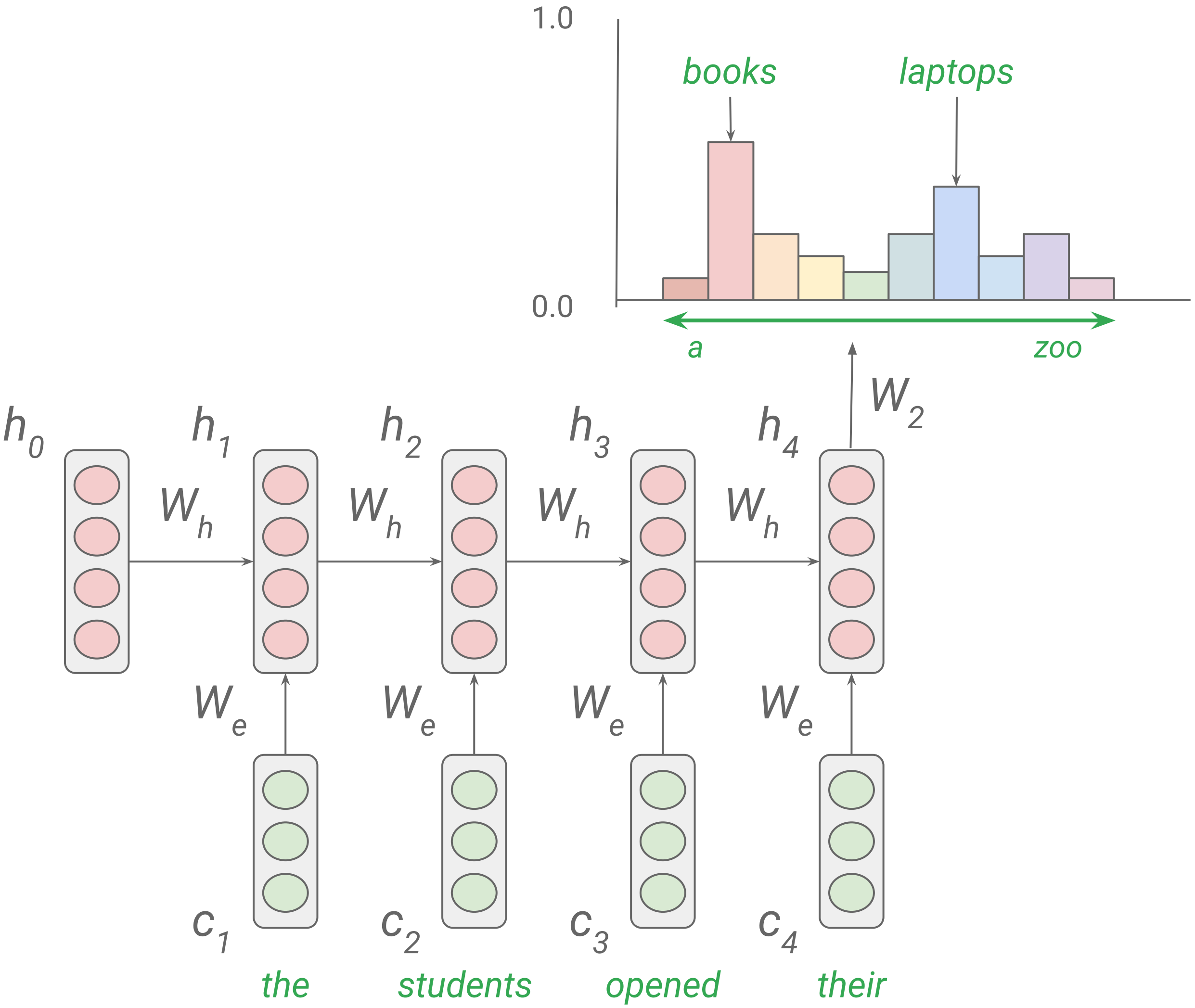
Recurrent neural networks (RNNs)

hidden states

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

output distribution

$$\hat{y} = softmax(W_2 h^{(n-1)})$$



Recurrent neural networks (RNNs)

- RNNs advantages
  - can handle much longer histories
  - can generalize better over contexts of similar words
  - are more accurate at word-prediction

- RNNs disadvantages
  - are much more complex
  - are slower and need more energy to train
  - and are less interpretable than n-gram models

Thank you!