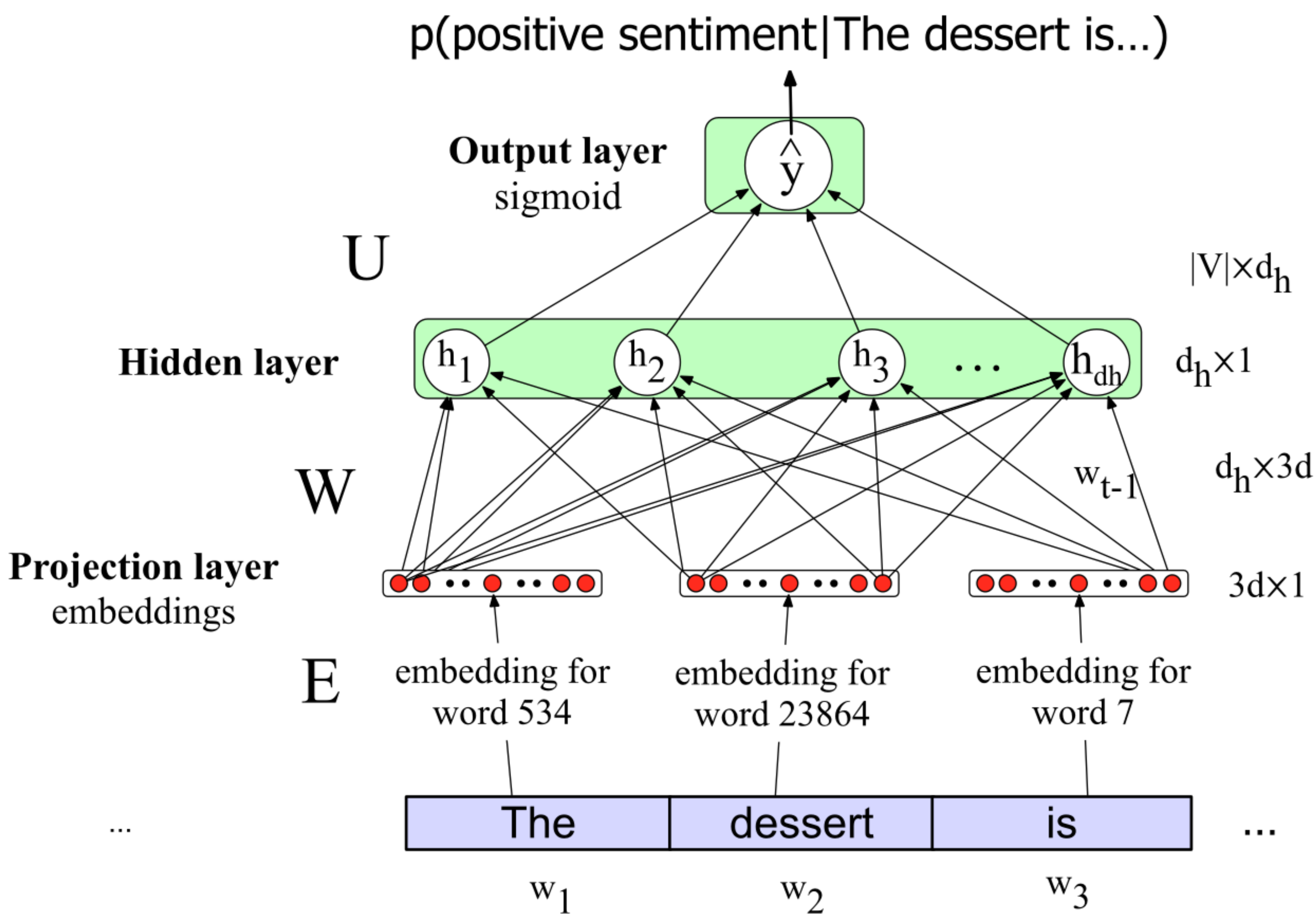
CS 333: Natural Language Processing

Fall 2023

Prof. Carolyn Anderson
Wellesley College

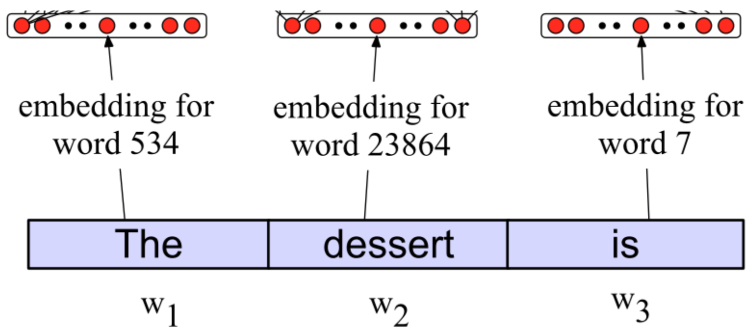
Neural Language Models

Neural Net Classification with embeddings as input features!



Issue: texts come in different sizes

This assumes a fixed size length (3)!

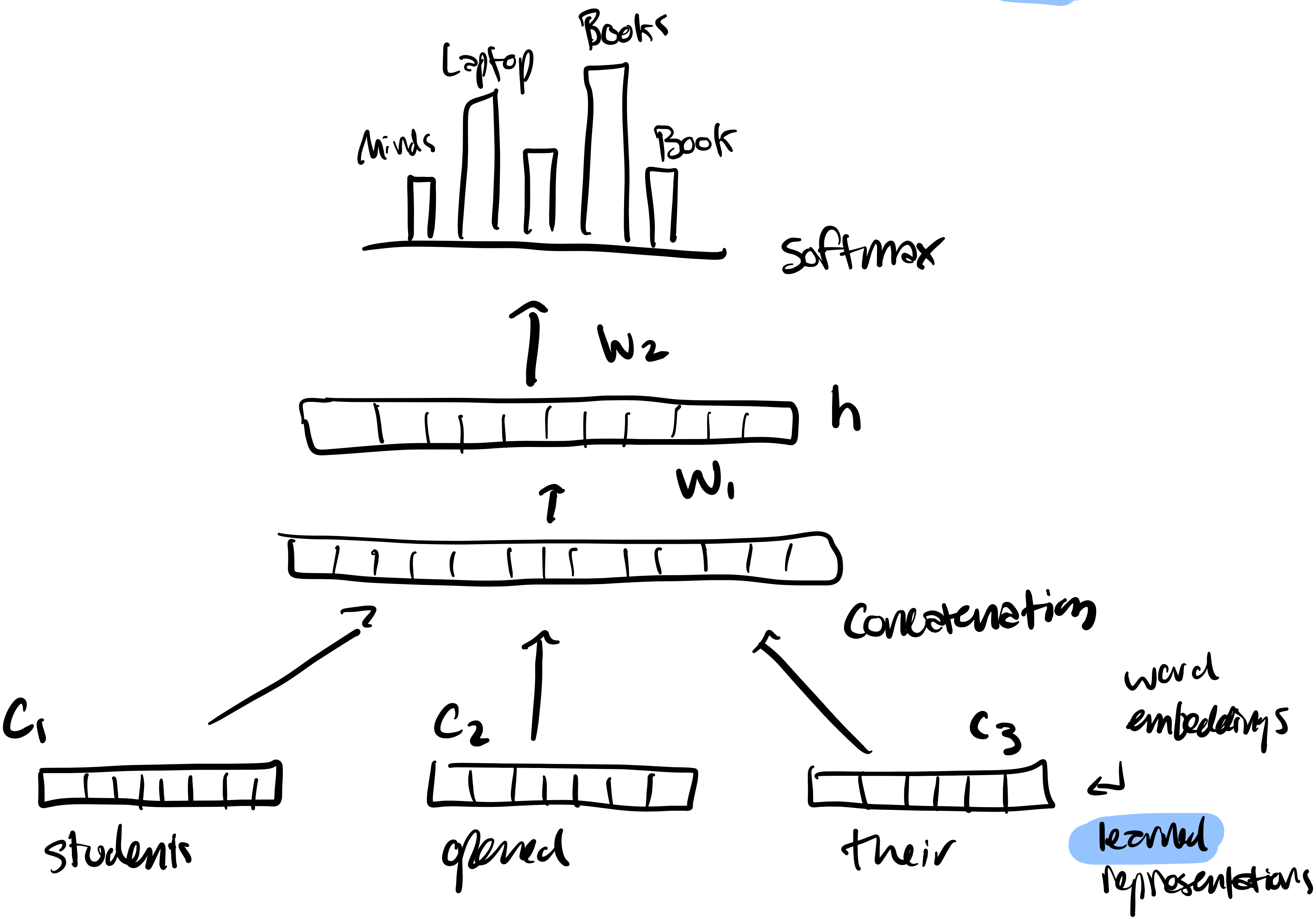


Some simple solutions (more sophisticated solutions later)

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
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

Training a Fixed-Length Neural Language Model

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

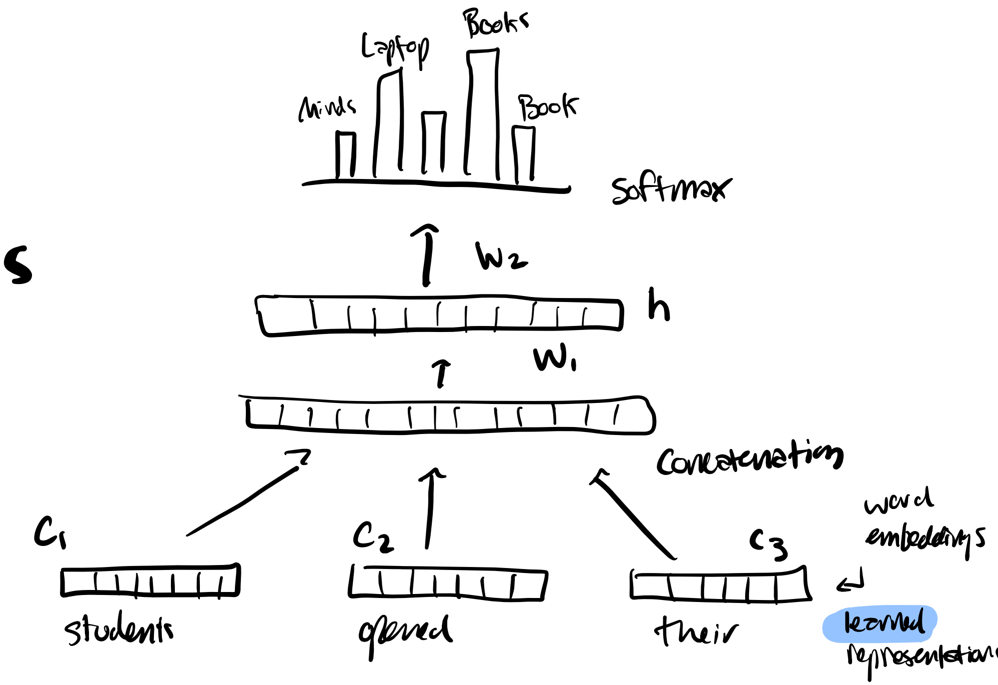


Key Question: what are the parameters?

W_1: input weights

W_2: pre-softmax weights

c_1, c_2, c_3: word embeddings



Feedforward Equations

$$h = f(W_1 \cdot [c_1 ; c_2 ; c_3])$$

$;$ = concatenation

f = non-linear activation

$$O = \text{softmax}(W_2 \cdot h)$$

This tells us how to make a prediction given some input word embedding.

But how do we <u>learn</u> the parameters?

Training a Feedforward Network

In language modeling, our labels come for free!

"I write this sitting in the kitchen sink."

$x_1 \quad y_1 \qquad x_2 \; y_2 \qquad x_3 \; y_3$

Free!

$\theta =$ W_2: pre-softmax weights

W_1: input weights

c_1, c_2, c_3: word embeddings

Goal: Adjust θ to make better predictions on our training data.

Our Friend, Gradient Descent

1) Define a loss function $L(\theta)$ to minimize

Desiderata:
- measure LM performance
- smooth
- differentiable

$\boxed{\text{Cross-entropy loss}}$

Language modeling as a classification task:

Vocabulary is the set of class labels

Our Friend, Gradient Descent

2) Given $L(\theta)$ compute the gradient of L with respect to θ

Gradient tells us the direction of the steepest ascent of L.

Gradient is the same dimensionality as θ: we need to adjust each parameter in θ.

For each param j in θ, gradient tells us how much loss would increase if we increase j.

Our Friend, Gradient Descent

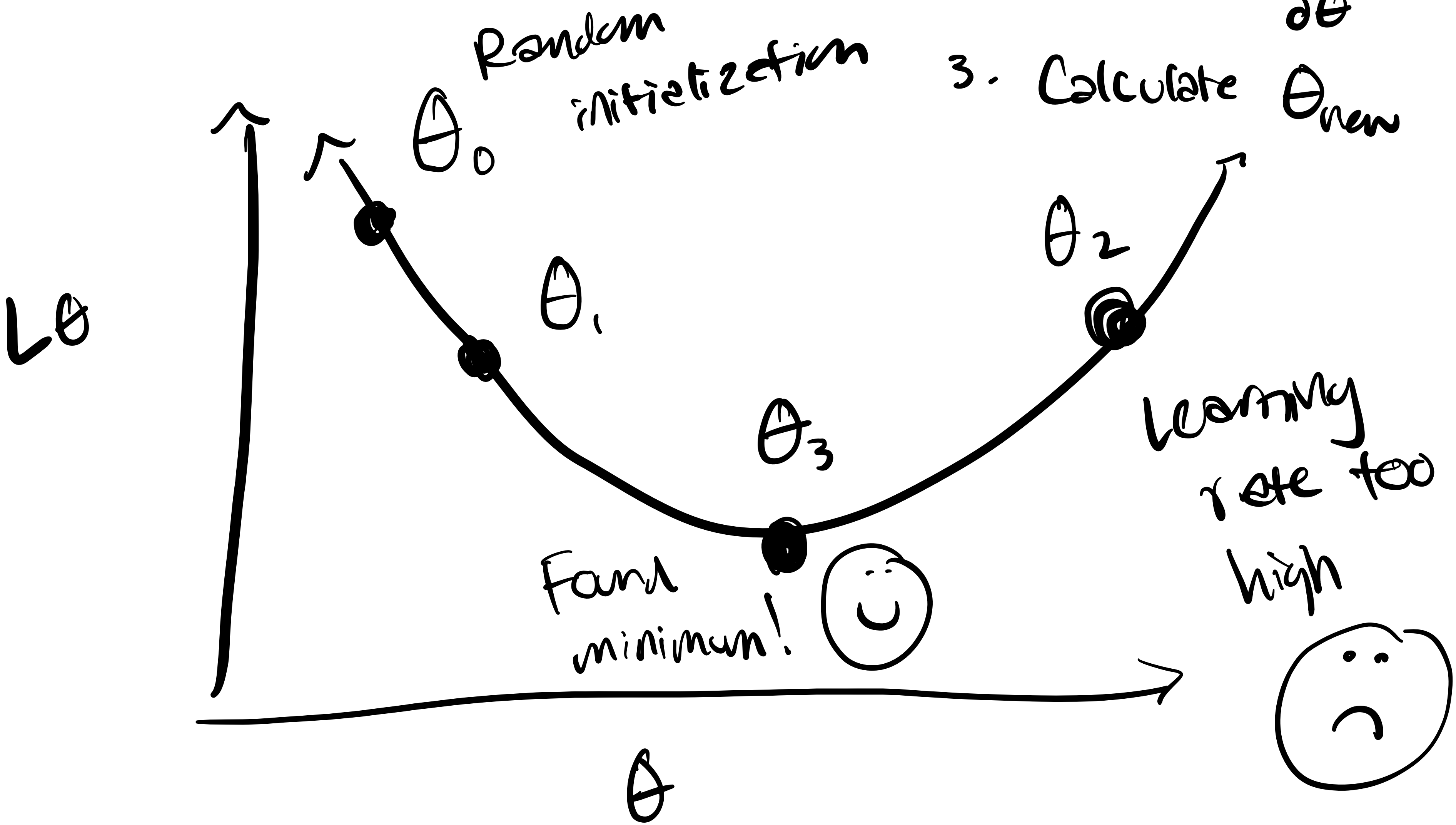
3) Given the gradient $\frac{\partial L}{\partial \theta}$, take a step in the direction of the negative gradient.

This minimizes L.

$$\underbrace{\theta_{new}}_{\text{updated weight}} = \underbrace{\theta_{old}}_{\text{old params}} - \underbrace{\eta}_{\text{learning rate}} \underbrace{\frac{\partial L}{\partial \theta}}_{\text{gradient}}$$

Our Friend, Gradient Descent

1. Calculate $L\theta$
2. Calculate $\frac{\partial L}{\partial \theta}$
3. Calculate θ_{new}



Hyperparameters

- Learning rate (η)

- Batch size (k)

  How often do we update weights?

  - Could update after each x
    - Gradient is poorly estimated
  - Could wait to see all examples
    - Slow but good estimate

Compromise: minibatching

Single Neuron Example

1. Calculate $L\Theta$
2. Calculate $\frac{\partial L}{\partial \Theta}$
3. Calculate Θ_{new}

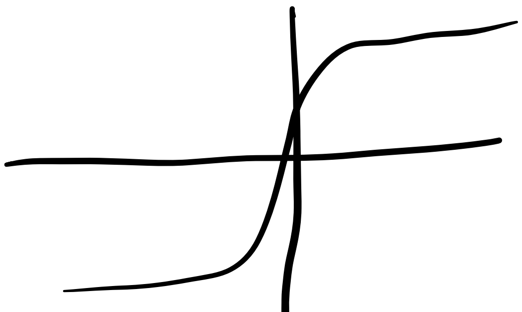
$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} O$$

4, 5.1

Inputs: (x, y)

$$h_1 = \tanh(w_1 \cdot x)$$

$$O = \tanh(w_2 h_1)$$



tanh

Step 1: Compute the Loss

$$L = \frac{1}{2}(y - O)^2$$

Single Neuron Example

Step 2: Compute the gradient

$$\frac{\partial L}{\partial \theta}\,;\quad \frac{\partial L}{\partial w_1}\,,\quad \frac{\partial L}{\partial w_2}$$

$$\boxed{\text{Chain Rule: } \frac{\partial\, g(f(x))}{\partial x} = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x}}$$

Parameter 1: $\frac{\partial L}{\partial w_2}$

$$L = \frac{1}{2}(y-o)^2$$

Single Neuron Example

Step 2: Compute the gradient

Parameter 1: $\frac{\partial L}{\partial w_2}$ $\quad L = \frac{1}{2}(y-o)^2$

$o = \tanh(a)$

$a = w_2 h$

$h = \tanh(b)$

$b = w_1 x$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial a} \cdot \frac{\partial a}{\partial w_2}$$

$$\frac{\partial L}{\partial o} \to -(y-o) \qquad \frac{\partial o}{\partial a} \to (1-o^2) \qquad \frac{\partial a}{\partial w_2} \to h$$

Derivative of tanh: $\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$

Single Neuron Example

Parameter 2: $\frac{\partial L}{\partial w_1}$

For convenience:

$$o = \tanh(a)$$

$$a = w_2 h$$

$$h = \tanh(b)$$

$$b = w_1 x$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial a} \underbrace{\frac{\partial a}{\partial h}}_{w_2} \underbrace{\frac{\partial h}{\partial b}}_{1-h^2} \underbrace{\frac{\partial b}{\partial w_1}}_{x}$$

Single Neuron Example

3) Update the parameters

$$w_{2_{new}} = w_{2_{old}} - \eta \frac{\partial L}{\partial w_2}$$

$$w_{1_{new}} = w_{1_{old}} - \eta \frac{\partial L}{\partial w_1}$$

Why Neural LMs work better than N-gram LMs

Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

Test data:

I forgot to make sure that the dog gets ___

N-gram LM can't predict "fed"!

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

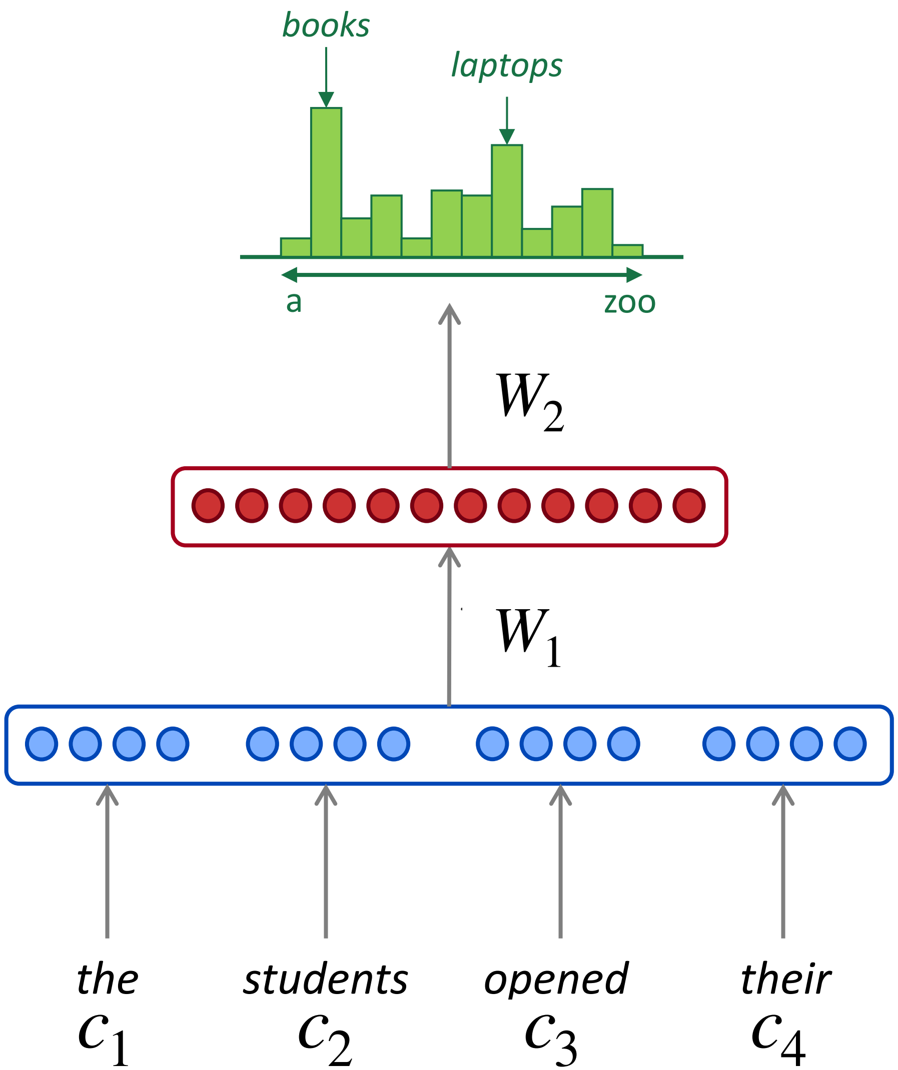
how does this compare to a normal n-gram model?

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 W
- Window can never be large enough!
- Each c_i uses different rows of W. We don't share weights across the window.



Slides adapted from Mohit Iyyer

How Do We Train Neural Networks?

Recurrent Neural Networks!

A RNN Language Model

hidden state

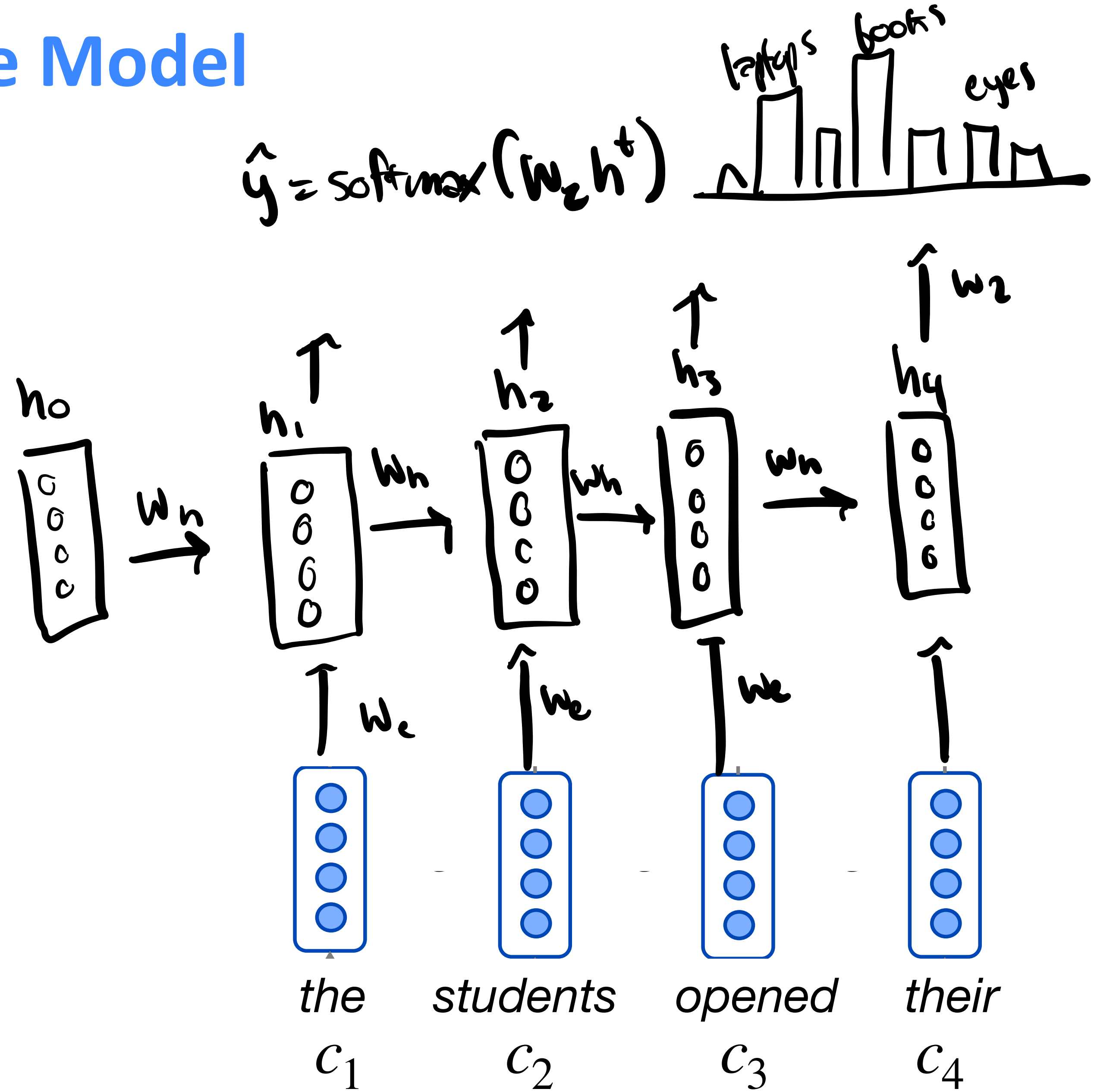
$$h^{(t)} = f(W_h \cdot h^{(t-1)} + W_e c_t)$$

h_0 is the initial hidden state

word embeddings

c_1, c_2, c_3, c_4

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



Slides adapted from Mohit Iyyer

A RNN Language Model

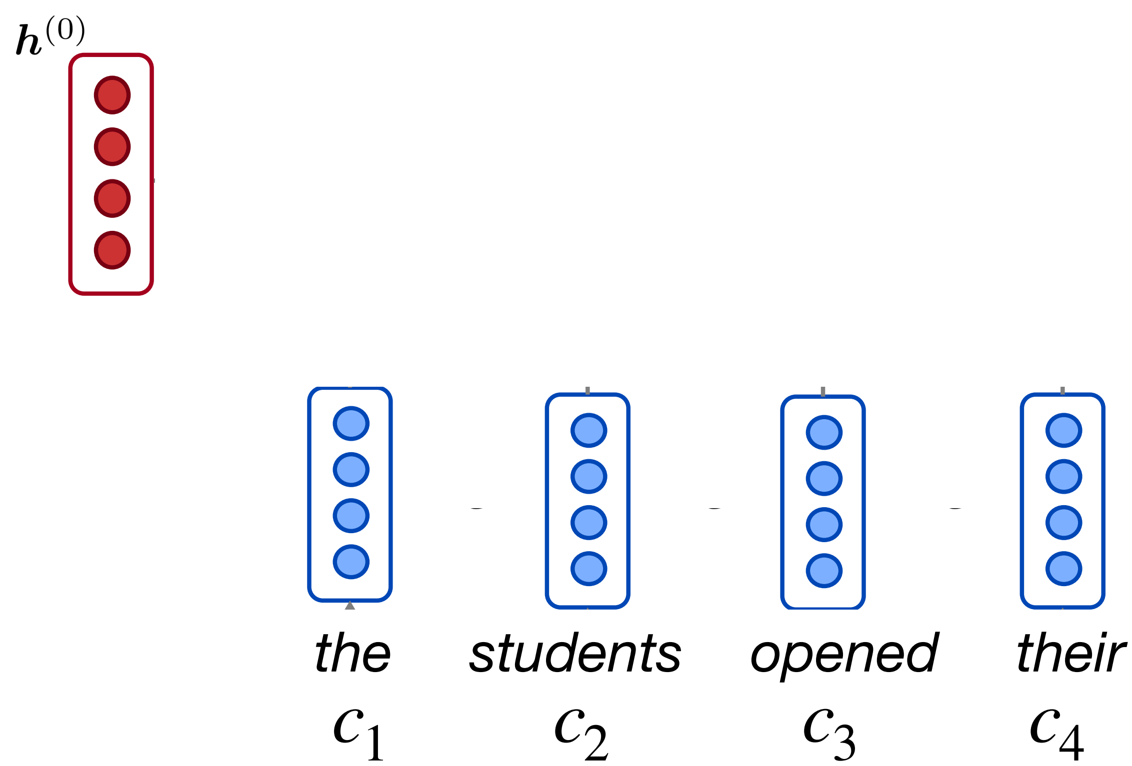
hidden states

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

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

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

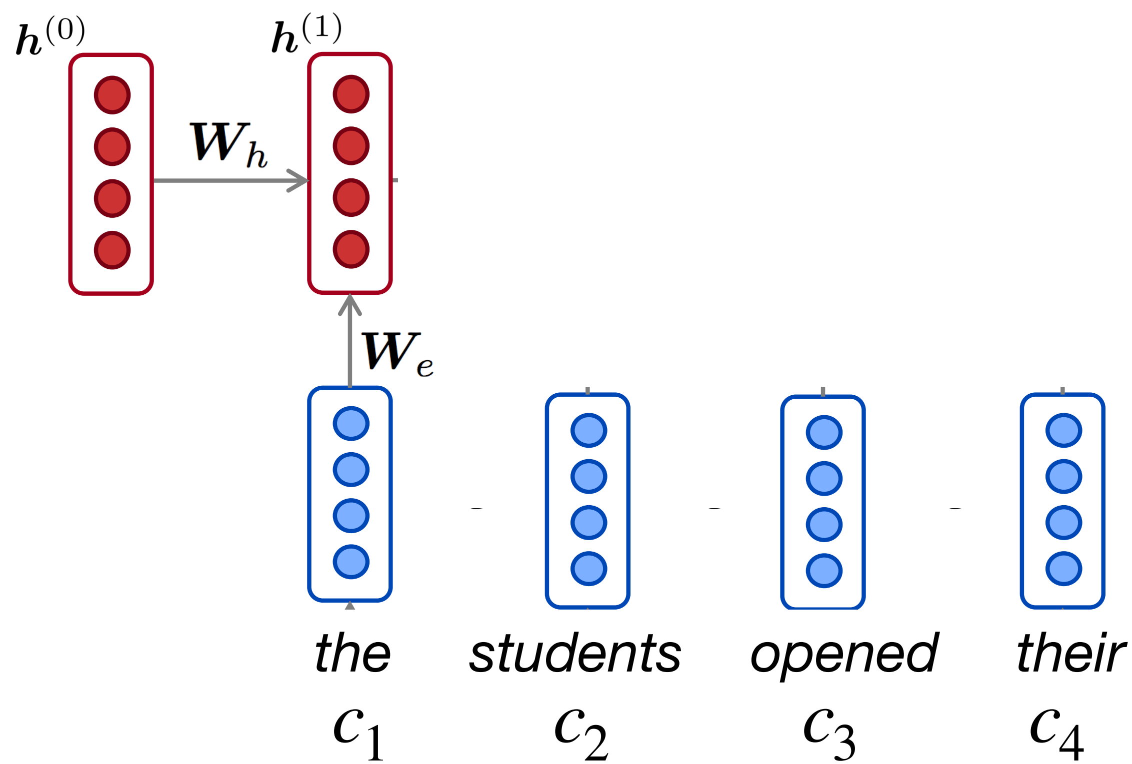
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



Slides adapted from Mohit Iyyer

A RNN Language Model

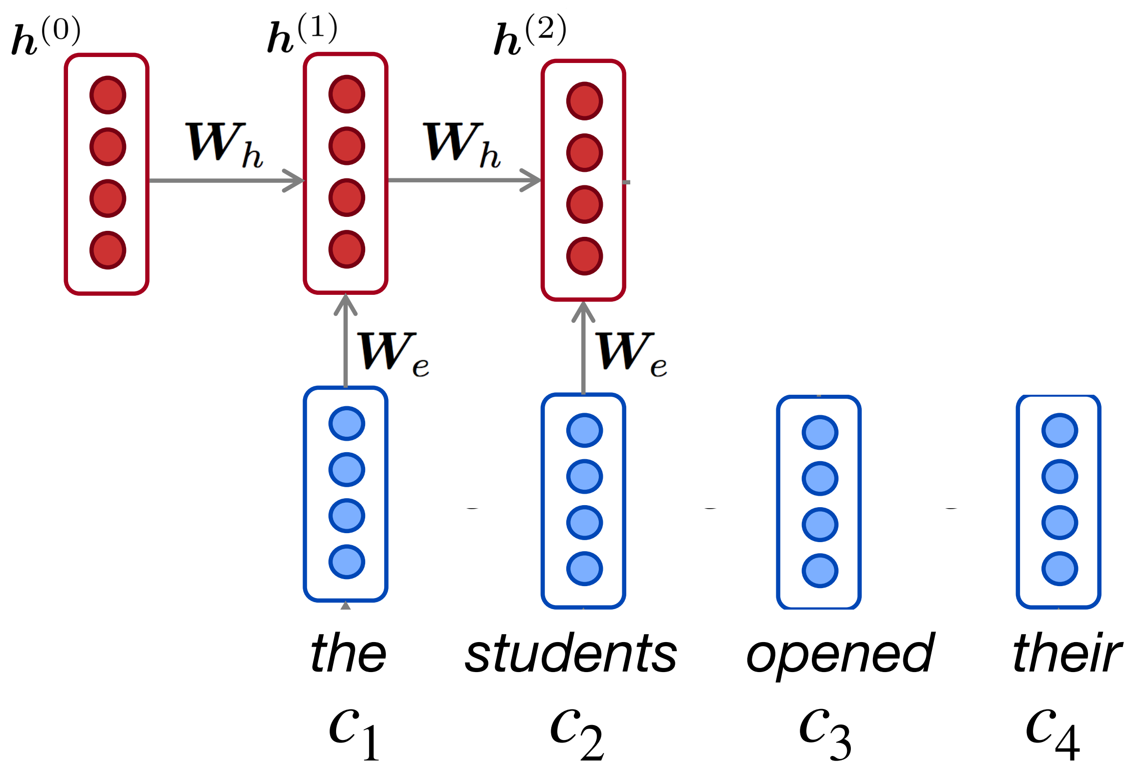
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



Slides adapted from Mohit Iyyer

A RNN Language Model

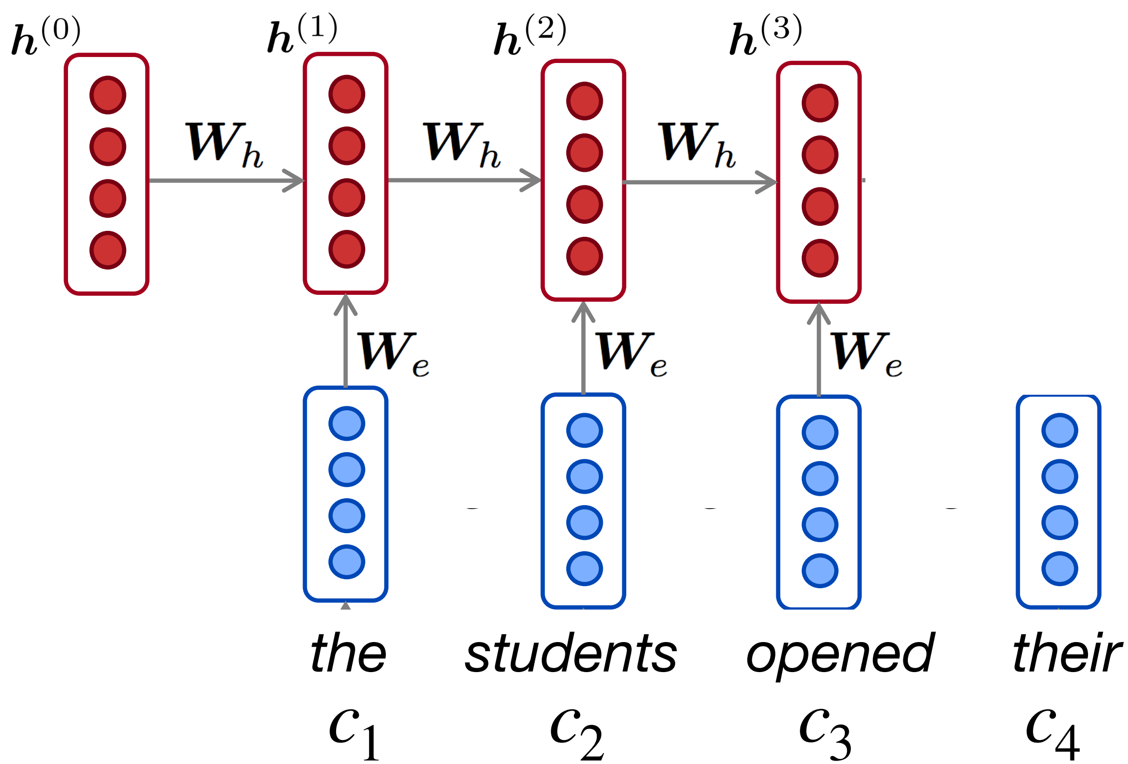
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



Slides adapted from Mohit Iyyer

A RNN Language Model

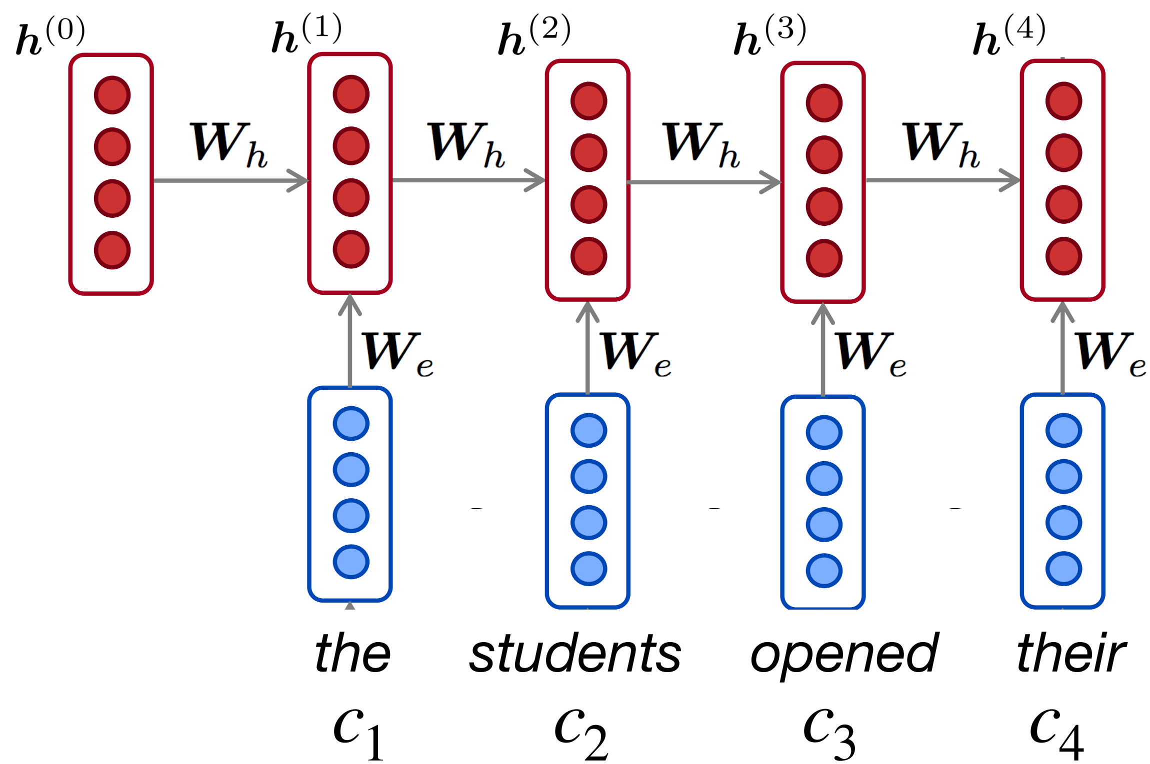
hidden states

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

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

word embeddings

c_1, c_2, c_3, c_4



Slides adapted from Mohit Iyyer

A RNN Language Model

output distribution

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

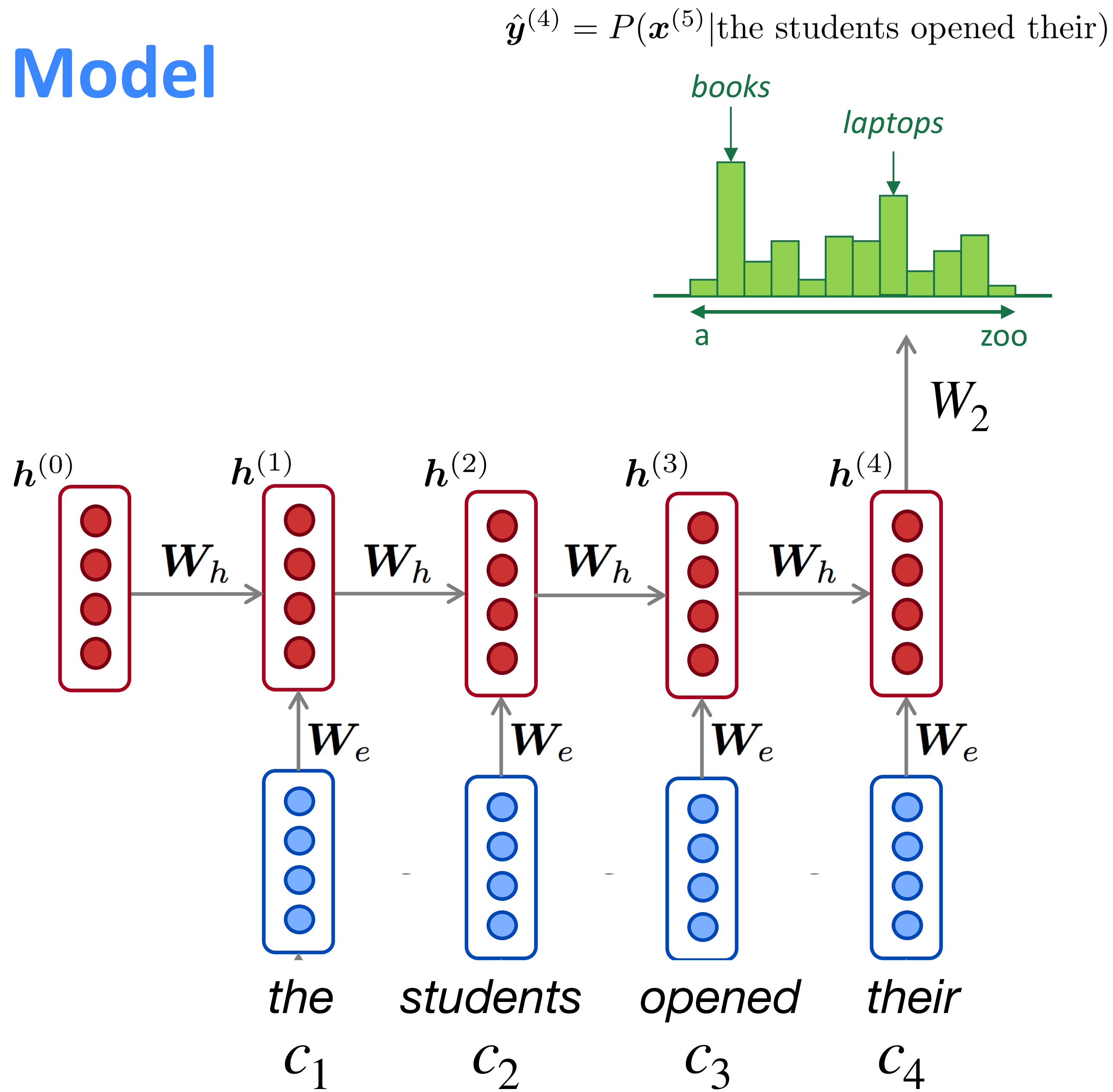
hidden states

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

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

word embeddings

$$c_1, c_2, c_3, c_4$$



Slides adapted from Mohit Iyyer

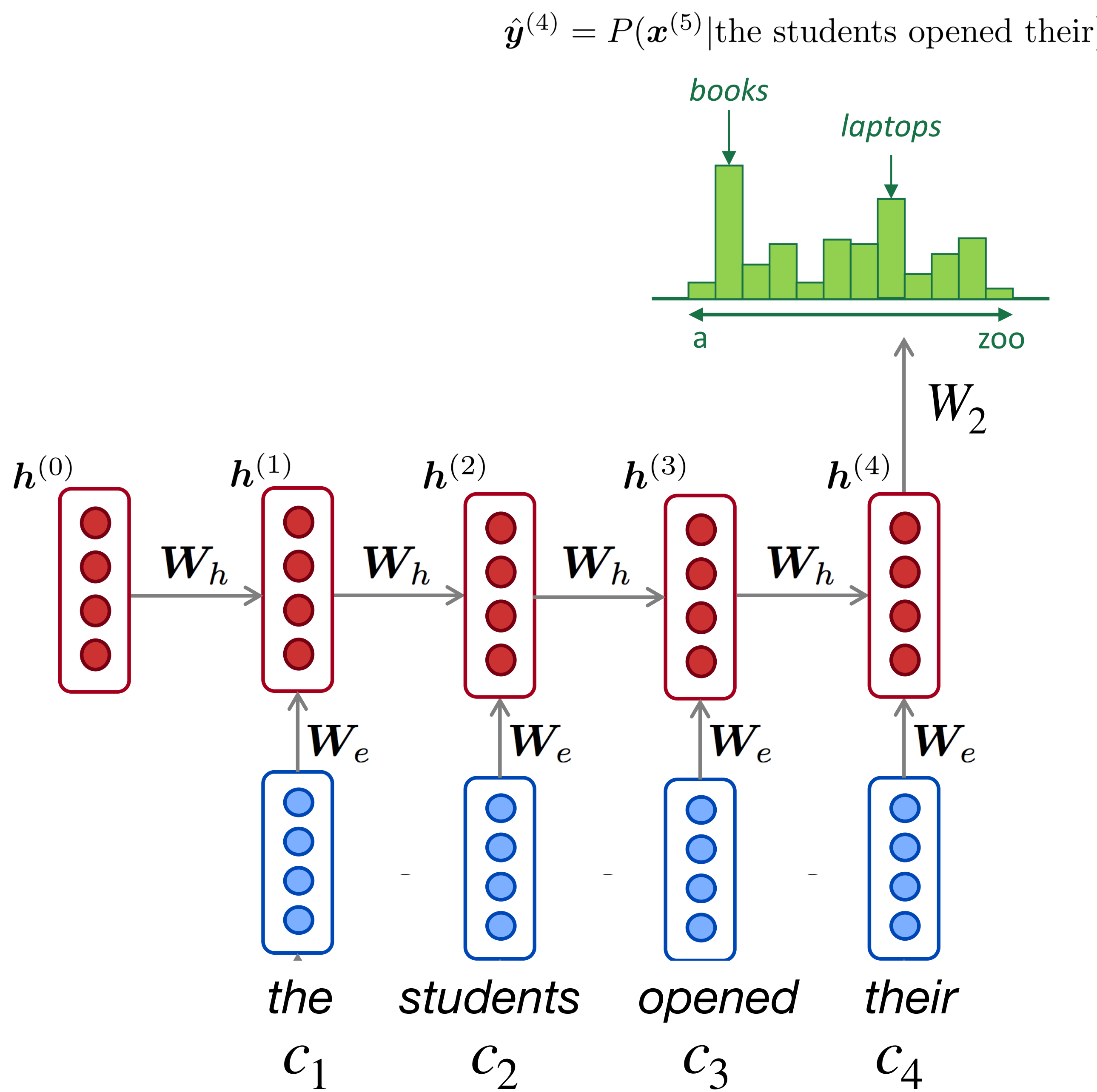
why is this good?

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

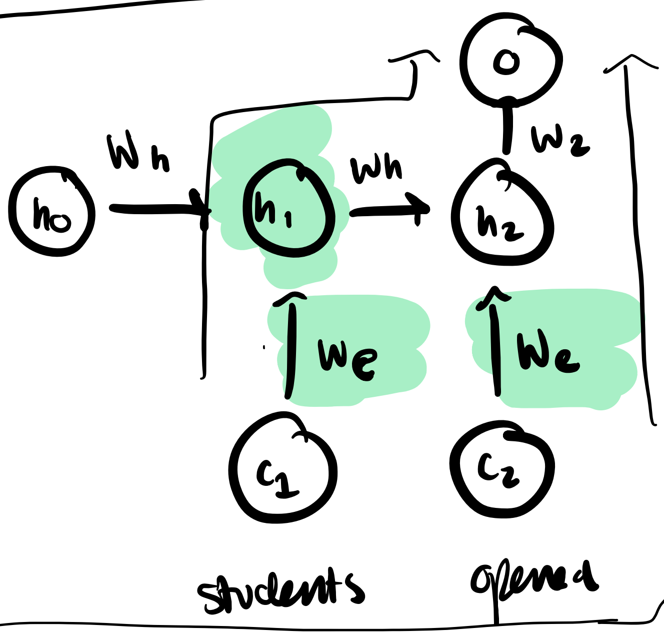
RNN **Disadvantages**:

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



Slides adapted from Mohit Iyyer

Recurrent Neural Network



Parameters

W_h : hidden state weights

W_e : embedding weights

W_2 : pre-softmax weights

c_1, c_2 : embeddings

$$L = \frac{1}{2}(y-o)^2$$

$$o = W_2 h_2$$

$$h_2 = \tanh(W_h h_1 + W_e c_2)$$

$$h_1 = \tanh(W_h h_0 + W_e c_1)$$

$$a = W_e c_2$$

$$b = W_h h_1$$

Goal: $\frac{\partial L}{W_e}$, $\frac{\partial L}{W_h}$, $\frac{\partial L}{W_2}$, $\frac{\partial L}{\partial c_1}$, $\frac{\partial L}{\partial c_2}$

√Goal 1:

$$\frac{\partial L}{\partial W_2} = \underbrace{\frac{\partial L}{\partial o}}_{-(y-o)} \underbrace{\frac{\partial o}{\partial W_2}}_{h_2} = -(y-o)h_2$$

√Goal 2: $\frac{\partial L}{\partial c_2}$

$$\frac{\partial L}{\partial c_2} = \underbrace{\frac{\partial L}{\partial o}}_{-(y-o)} \cdot \underbrace{\frac{\partial o}{\partial h_2}}_{W_2} \cdot \underbrace{\frac{\partial h_2}{\partial a}}_{(1-h_2^2)} \cdot \underbrace{\frac{\partial a}{\partial c_2}}_{W_e}$$

Goal:

$$\frac{\partial L}{\partial W_e} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_2} \cdot \frac{\partial h_2}{\partial a}\frac{\partial a}{\partial W_e} + \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_2} \cdot \frac{\partial h_2}{\partial a}\frac{\partial a}{\partial h_1}\frac{\partial h_1}{\partial b} \cdot \frac{\partial b}{\partial W_e}$$