Lecture 8: Recurrent Neural Nets

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Sep 24, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements + Logistics

- Today: Project Proposal Due
  - See instructions on website: these will serve as a rubric
    - Task definition, Data and Evaluation (how will you know your approach does well? Think baselines for comparison)
  - Submission link on Brightspace
  - Please do not break format
- HW2 due on 10/13
- HW1 grades will be out by next week
- Quiz 2 next Wednesday
  - Bring along a pen in case you need to show your work! You will be provided scrap paper

Lecture Outline

- Recap: Recurrent Neural Nets (RNNs)
- Training RNNLMs
- The Vanishing Gradient Problem
  - LSTMs
- Applications of RNNs
- Sequence-to-Sequence Modeling

Recap: Recurrent Neural Nets

Recurrent Neural Networks

- Recurrent Neural Networks processes sequences one element at a time:
  - Contains one hidden layer $\mathbf{h}_t$ per time step! Serves as a memory of the entire history…
  - Output of each neural unit at time t based both on
    - the current input at t and
    - the hidden layer from time $t-1$
- As the name implies, RNNs have a recursive formulation
  - dependent on its own earlier outputs as an input!
- RNNs thus don't have
  - the limited context problem that n-gram models have, or
  - the fixed context that feedforward language models have,
  - since the hidden state can *in principle* represent information about all of the preceding words all the way back to the beginning of the sequence

Recurrent Neural Net Language Models

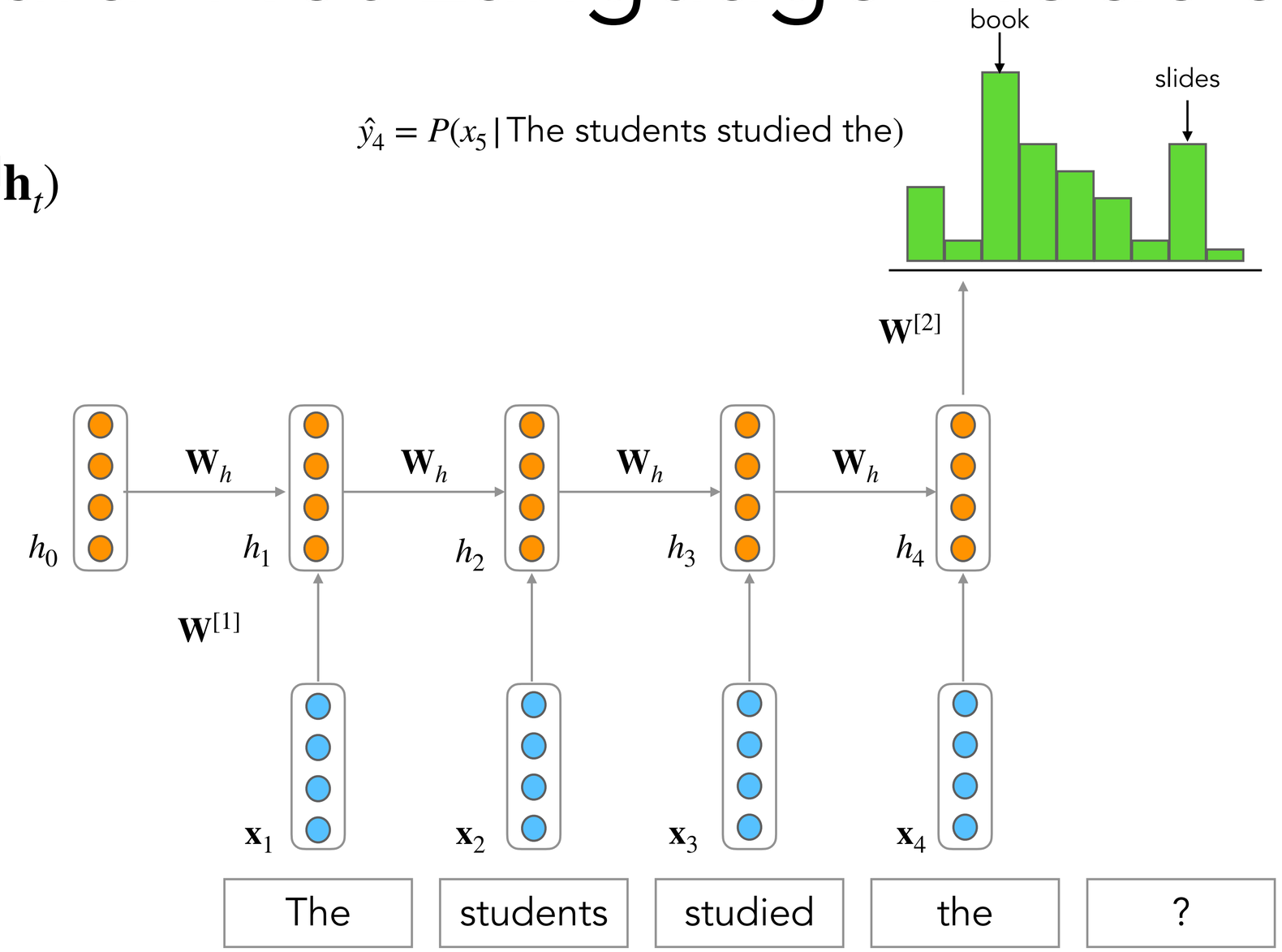
Output layer: $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{W}^{[2]}\mathbf{h}_t)$

Hidden layer:

$\mathbf{h}_t = g(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}^{[1]}\mathbf{x}_t)$

Initial hidden state: $\mathbf{h}_0$

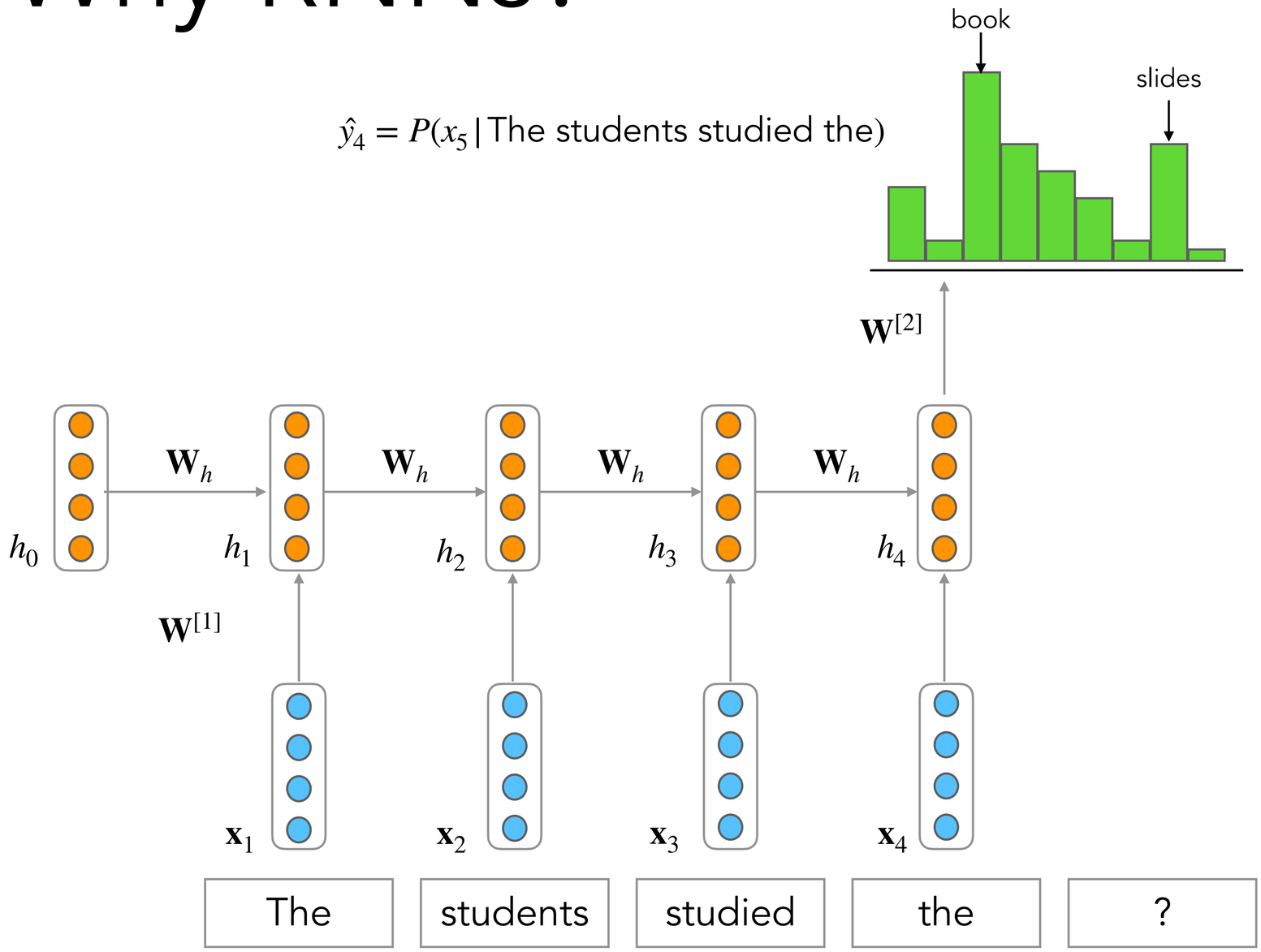
Word Embeddings, $\mathbf{x}_i$



Why RNNs?

RNN Advantages:

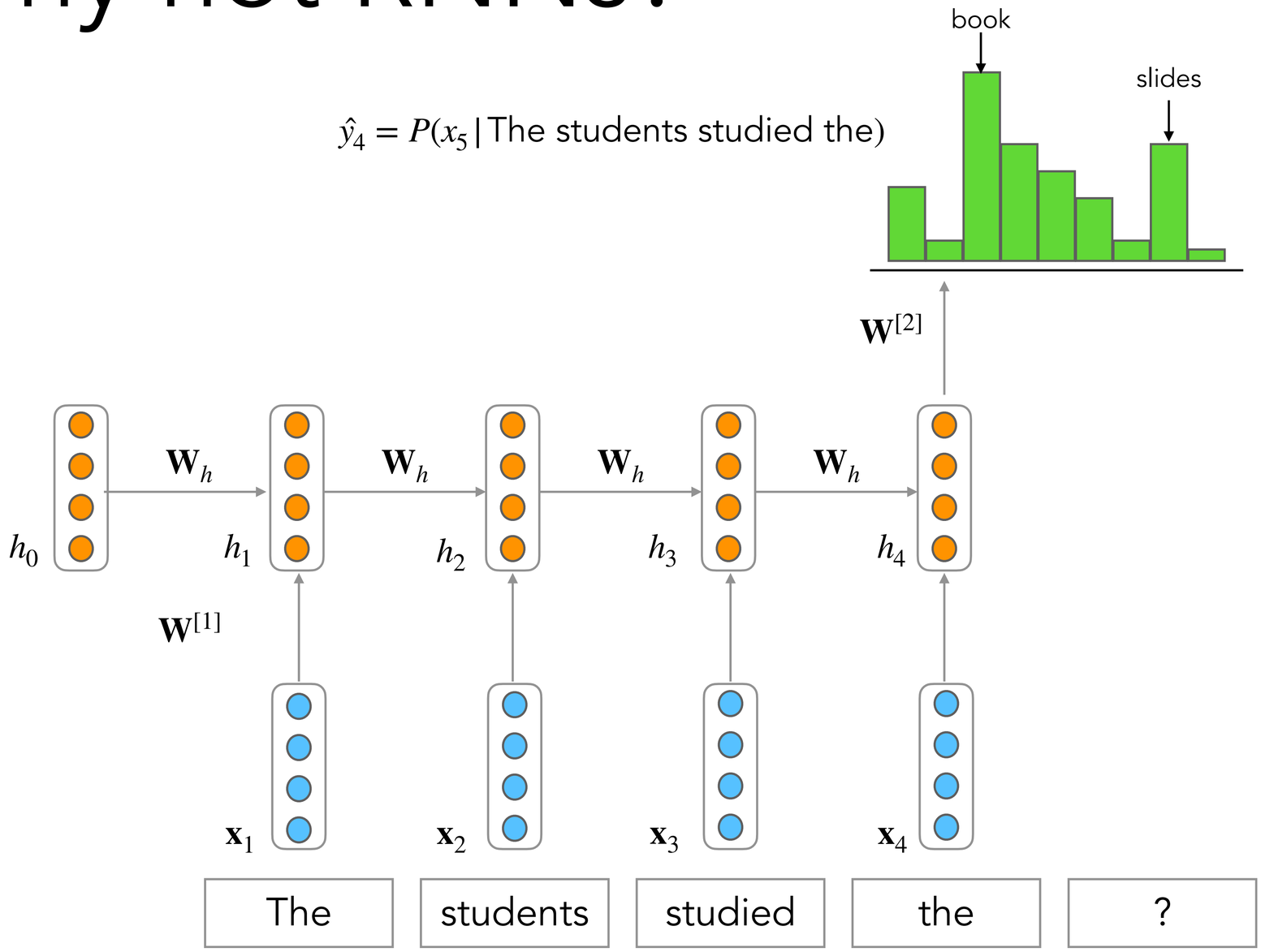
- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- Weights $\mathbf{W}^{[1]}$ are shared (tied) across timesteps → Condition the neural network on all previous words



Why not RNNs?

RNN Disadvantages:

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



Training RNNLMs

Training Outline

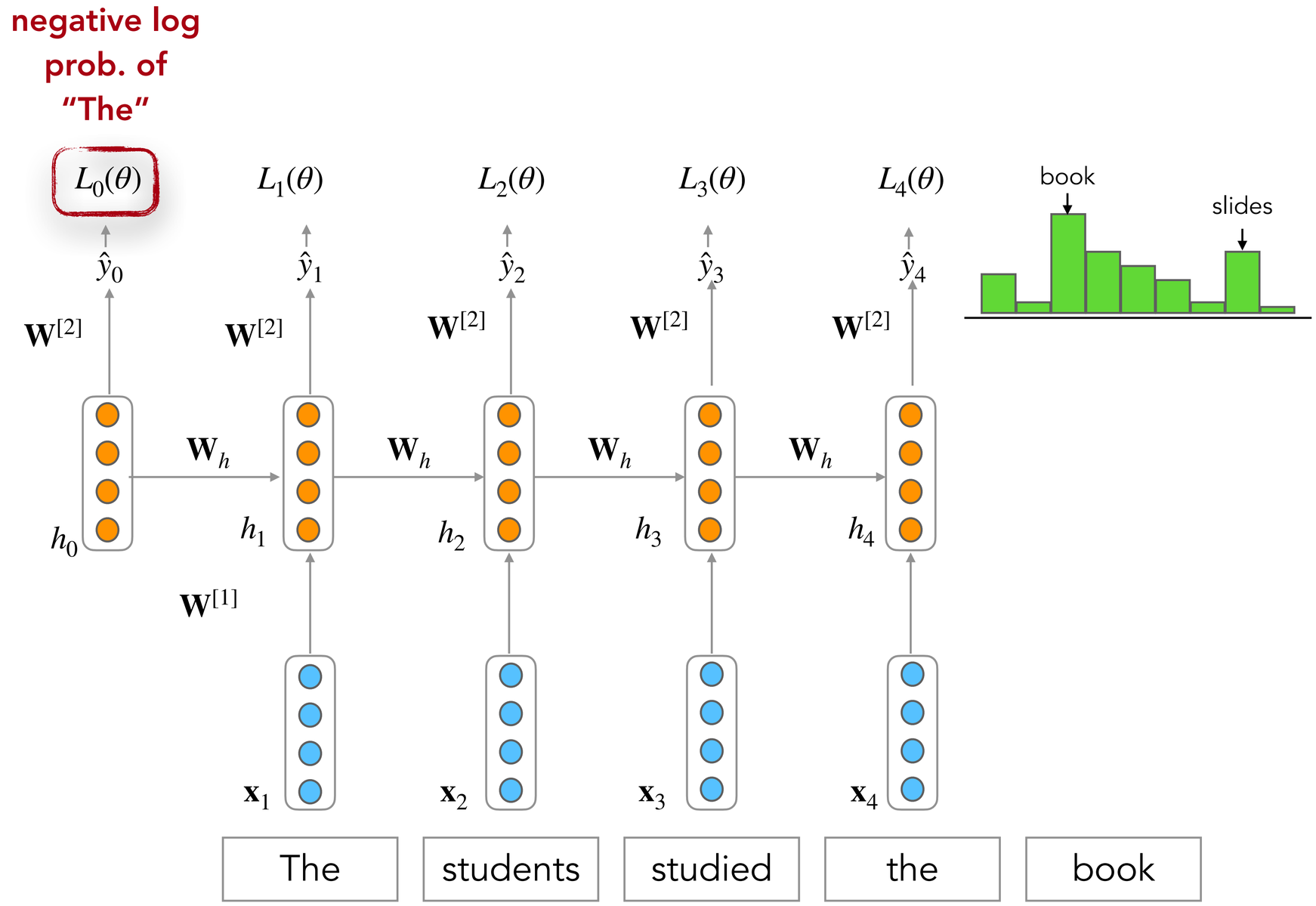
- Get a big corpus of text which is a sequence of words $x_1, x_2, \ldots x_T$
- Feed into RNN-LM; compute output distribution $\hat{y}_t$ for every step t
  - i.e. predict probability distribution of every word, given words so far
- Loss function on step t is usual cross-entropy between our predicted probability distribution $\hat{y}_t$, and the true next word $y_t = x_{t+1}$:

$$L_{CE}(\hat{y}_t, y_t; \theta) = -\sum_{v \in V} \mathbb{I}[y_t = v] \log \hat{y}_t = -\log p_\theta(x_{t+1} \mid x_{\leq t})$$

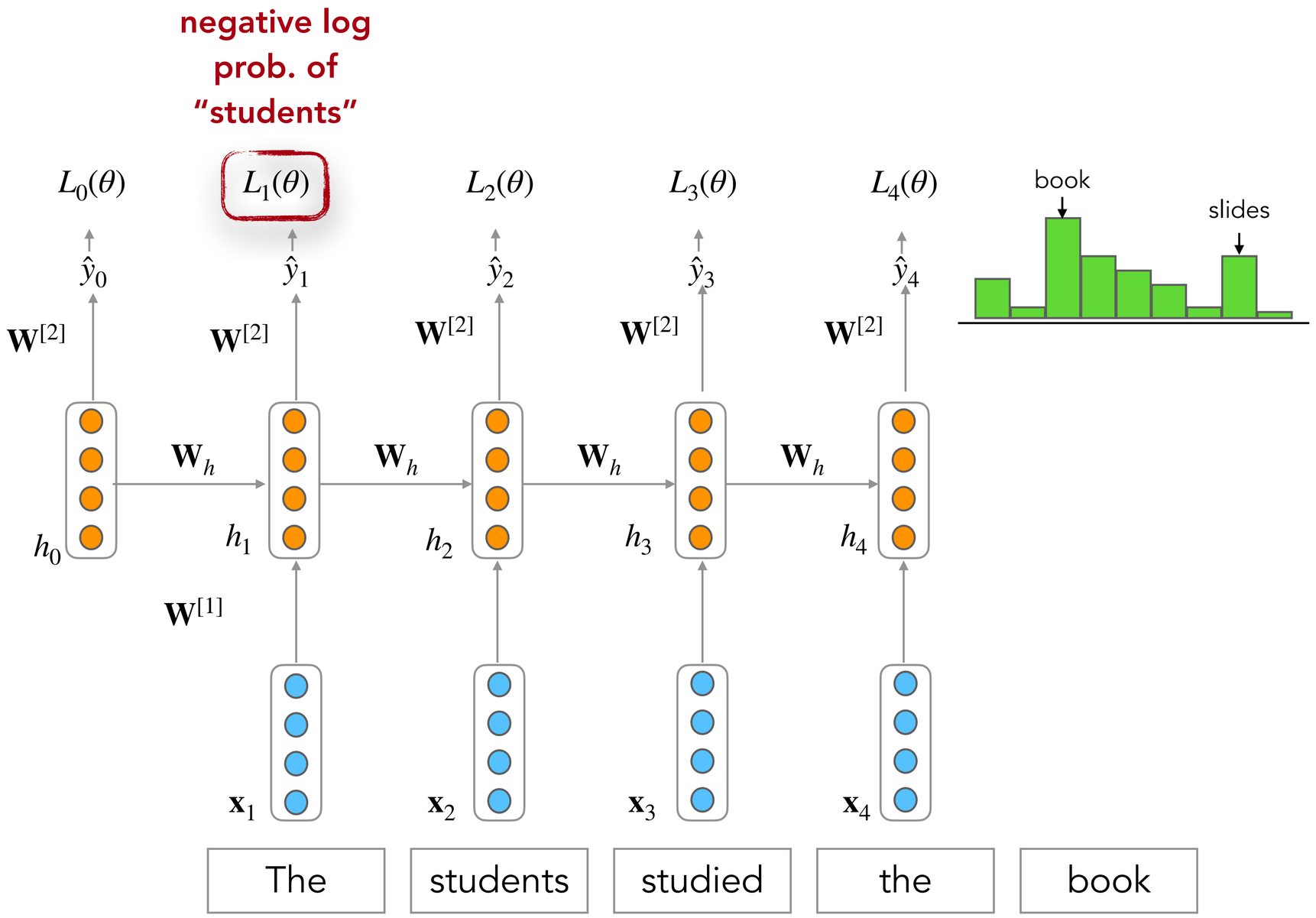
- Average this to get overall loss for entire training set:

$$L(\theta) = \frac{1}{T} \sum_{t=1}^{T} L_{CE}(\hat{y}_t, y_t)$$

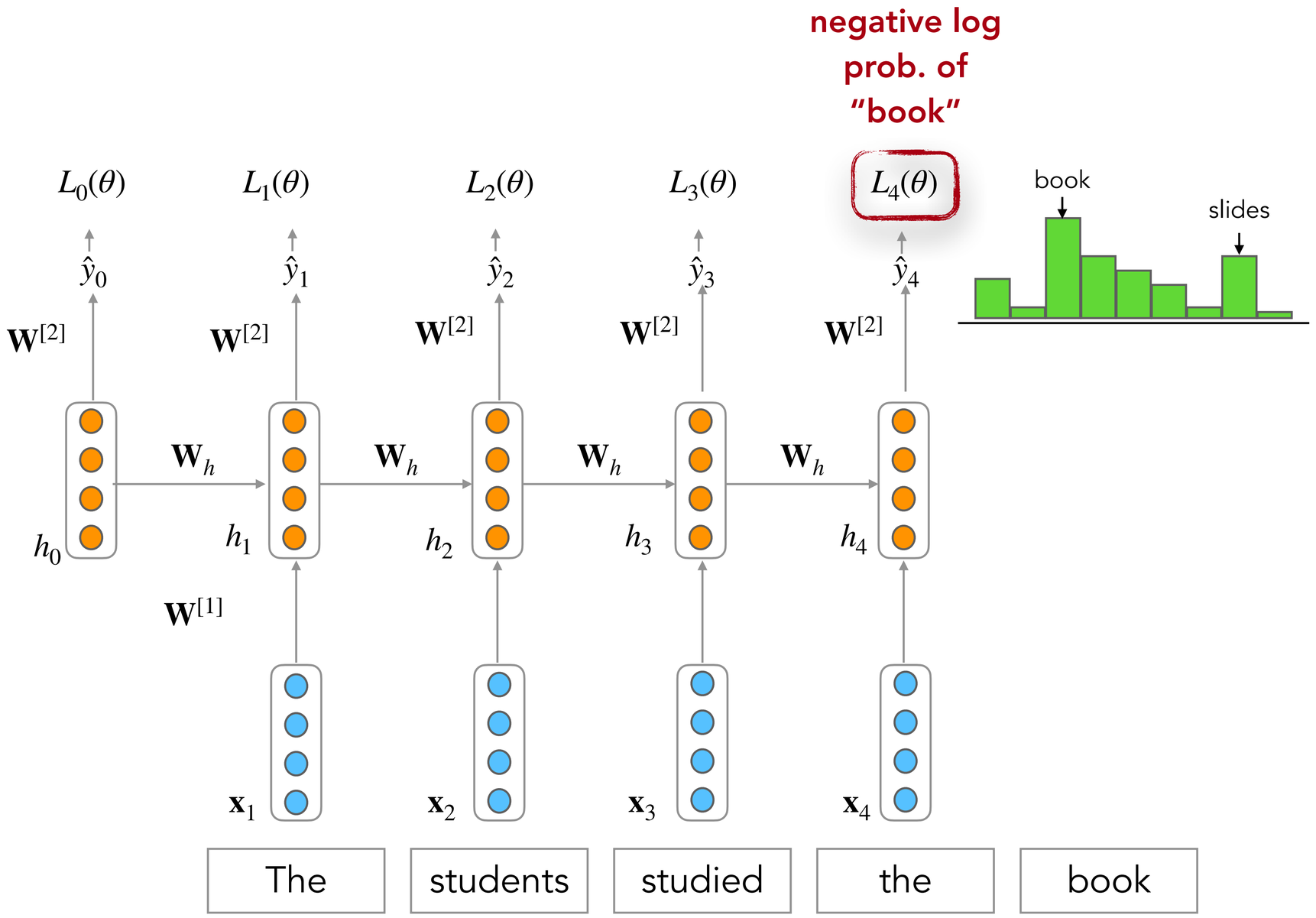
Loss


negative log prob. of "The"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
book
slides
The
students
studied
the
book

Loss

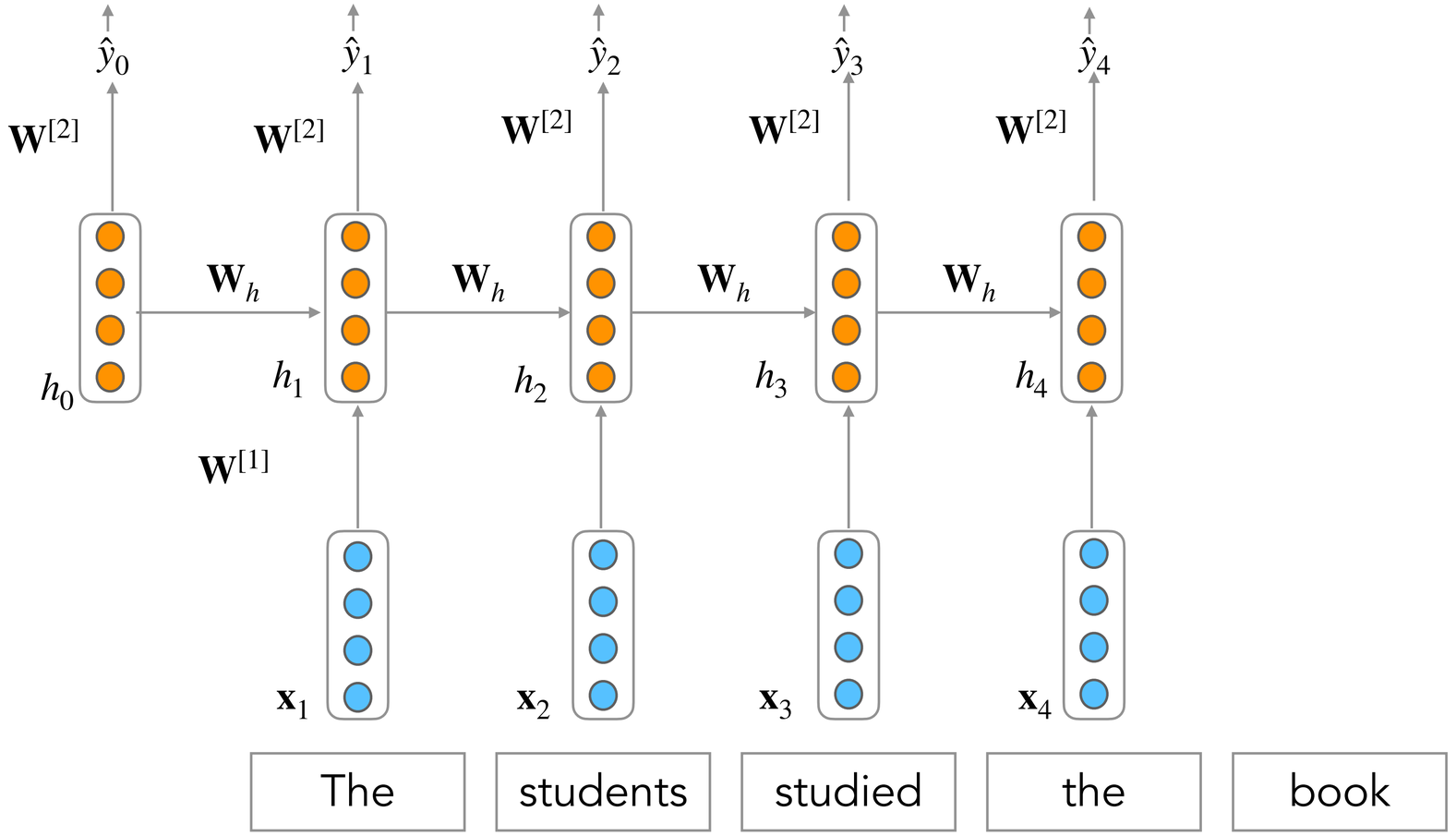

negative log prob. of "students"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
h_0
h_1
h_2
h_3
h_4
$\mathbf{W}^{[1]}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
book
slides
The
students
studied
the
book

Loss


negative log prob. of "book"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
book
slides
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
The
students
studied
the
book

Loss

$$L_0(\theta) + L_1(\theta) + L_2(\theta) + L_3(\theta) + L_4(\theta) + \ldots = L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta)$$



RNNs vs. Other LMs

Table 2. *Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).*

	Penn Corpus		Switchboard	
Model	NN	NN+KN	NN	NN+KN
KN5 (baseline)	-	141	-	92.9
feedforward NN	141	118	85.1	77.5
RNN trained by BP	137	113	81.3	75.4
RNN trained by BPTT	123	106	77.5	72.5

T. **Mikolov**, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur, "Extensions of recurrent neural network language model," *2011 IEEE ICASSP*, doi: 10.1109/ICASSP.2011.5947611.

Practical Issues with training RNNs

- Computing loss and gradients across entire corpus is too expensive!
- Recall: mini-batch Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
- Solution: consider chunks of text.
  - In practice, consider $x_1, x_2, \ldots x_T$ for some T as a "sentence" or "single data instance"

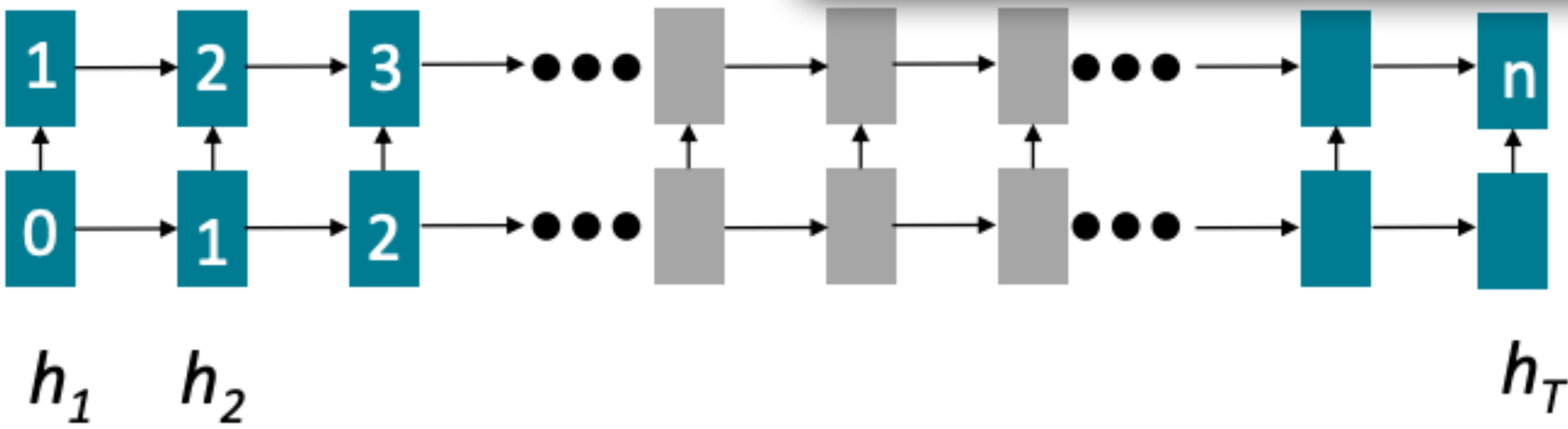
$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_{CE}(\hat{y}_t, y_t)$$

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

Training RNNs is hard: Parallelizability

- Forward and backward passes have **O(sequence length)** unparallelizable operations!
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed

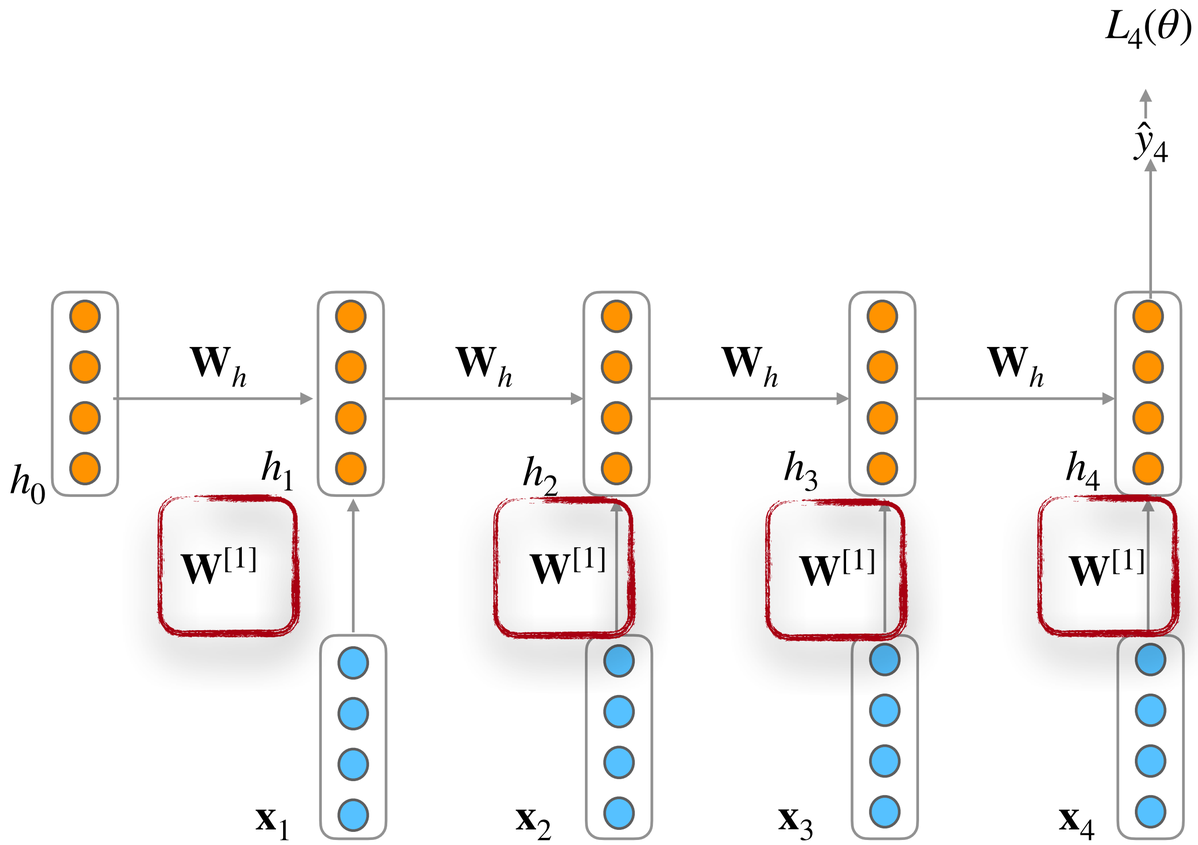
Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

Training RNNs is hard: Gradients

- Multiply the same matrix at each time step during forward propagation
- Ideally inputs from many time steps ago can modify output y
- This leads to something called the **vanishing gradient problem**

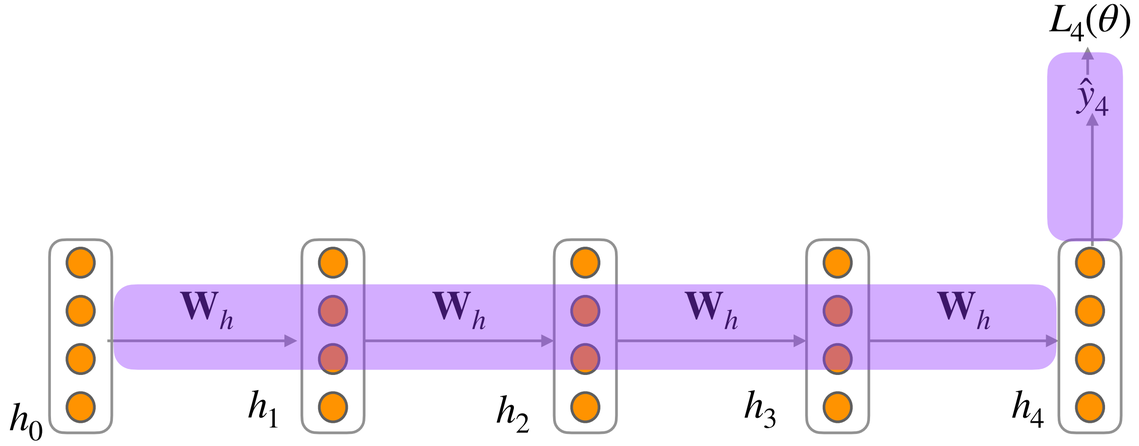


Lecture Outline

- Recap: Recurrent Neural Nets (RNNs)
- Training RNNLMs
- The Vanishing Gradient Problem
  - LSTMs
- Applications of RNNs
- Sequence-to-Sequence Modeling

The Vanishing Gradient Problem and LSTMs

The Vanishing Gradient Problem: Intuition



When these gradients are small, the gradient signal gets smaller and smaller as it backpropagates further…

$$\frac{\partial L_4}{\partial h_0} = \frac{\partial h_1}{\partial h_0} \times \frac{\partial L_4}{\partial h_1}$$

$$= \frac{\partial h_1}{\partial h_0} \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial L_4}{\partial h_2}$$

$$= \frac{\partial h_1}{\partial h_0} \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial L_4}{\partial h_3}$$

$$= \frac{\partial h_1}{\partial h_0} \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_4}{\partial h_3} \times \frac{\partial L_4}{\partial h_4}$$

Gradient signal from far away is lost because it's much smaller than gradient signal from close-by

The Vanishing Gradient Problem: Effects

- In practice, no long-term / long-range effects, contrary to the RNN promise
- Example language modeling task
  - To learn from this training example, the RNN-LM needs to model the dependency between "tickets" on the 7th step and the target word "tickets" at the end
- But if the gradient is small, the model can't learn this dependency
  - So, the model is unable to predict using similar long-distance dependencies at test time
- In practice a simple RNN will only condition ~7 tokens back [vague rule-of-thumb]

When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____

The Vanishing Gradient Problem: Fixes

- The main problem is that it is too difficult for the RNN to learn to preserve information over many timesteps
- In a vanilla RNN, the hidden state is constantly being rewritten

$$\mathbf{h}_t = reLU(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}^{[1]}\mathbf{x}_t)$$

New design: equip an RNN with separate memory data structure

Solution? Think data structures…

Long Short-Term Memory RNNs (LSTMs)

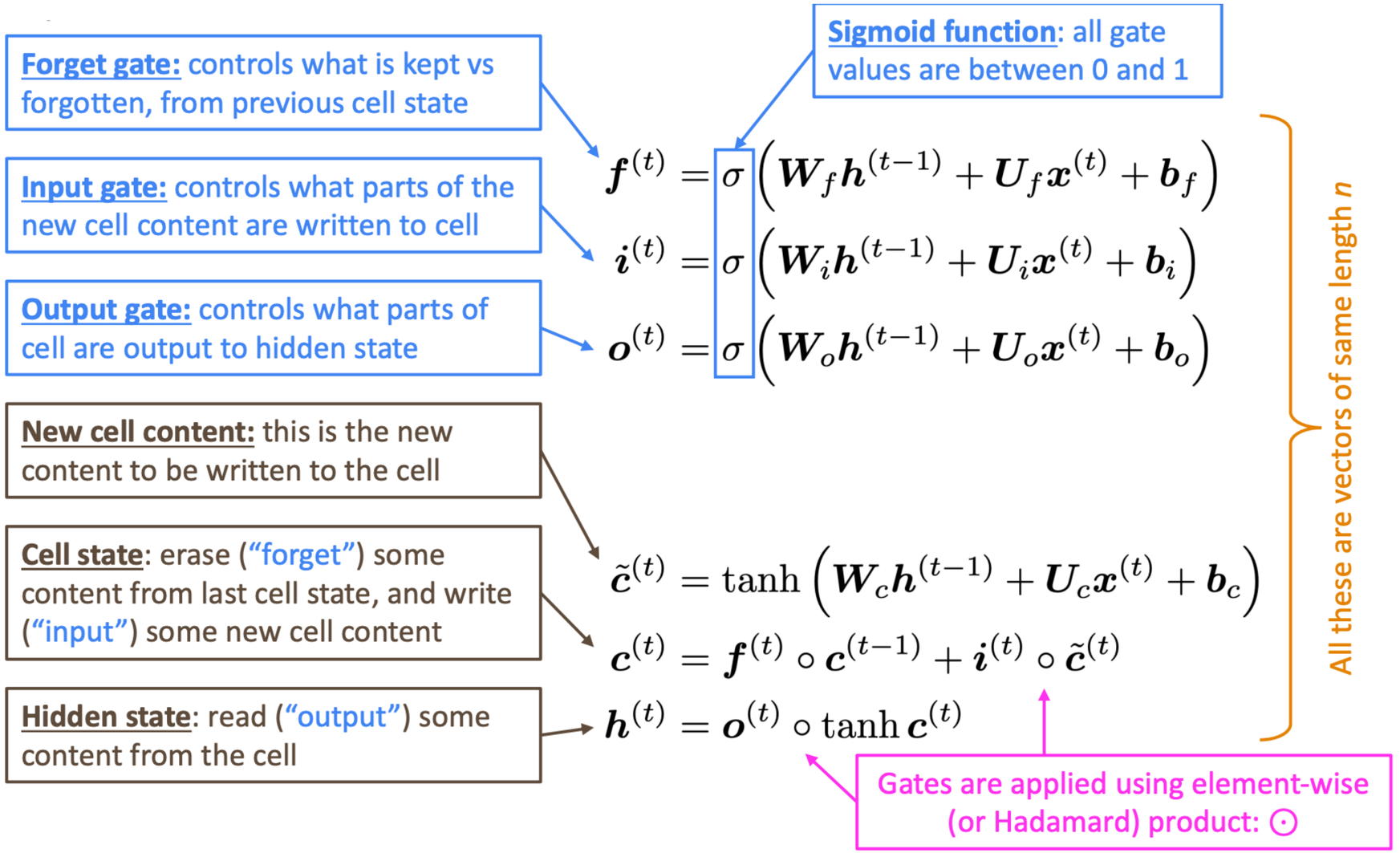
- At time step t, introduces a new cell state $\mathbf{c}_t \in \mathbb{R}^d$
  - In addition to a hidden state $\mathbf{h}_t \in \mathbb{R}^d$
  - The cell stores long-term information (memory)
  - The LSTM can read, erase, and write information from the cell!
    - The cell becomes conceptually rather like RAM in a computer

- The selection of which information is erased/written/read is controlled by three corresponding gates:
  - Input gate $\mathbf{i}_t \in \mathbb{R}^d$, Output gate $\mathbf{o}_t \in \mathbb{R}^d$ and Forget gate $\mathbf{f}_t \in \mathbb{R}^d$
  - Each *element* of the gates can be open (1), closed (0), or somewhere in between
  - The gates are dynamic: their value is computed based on the current context

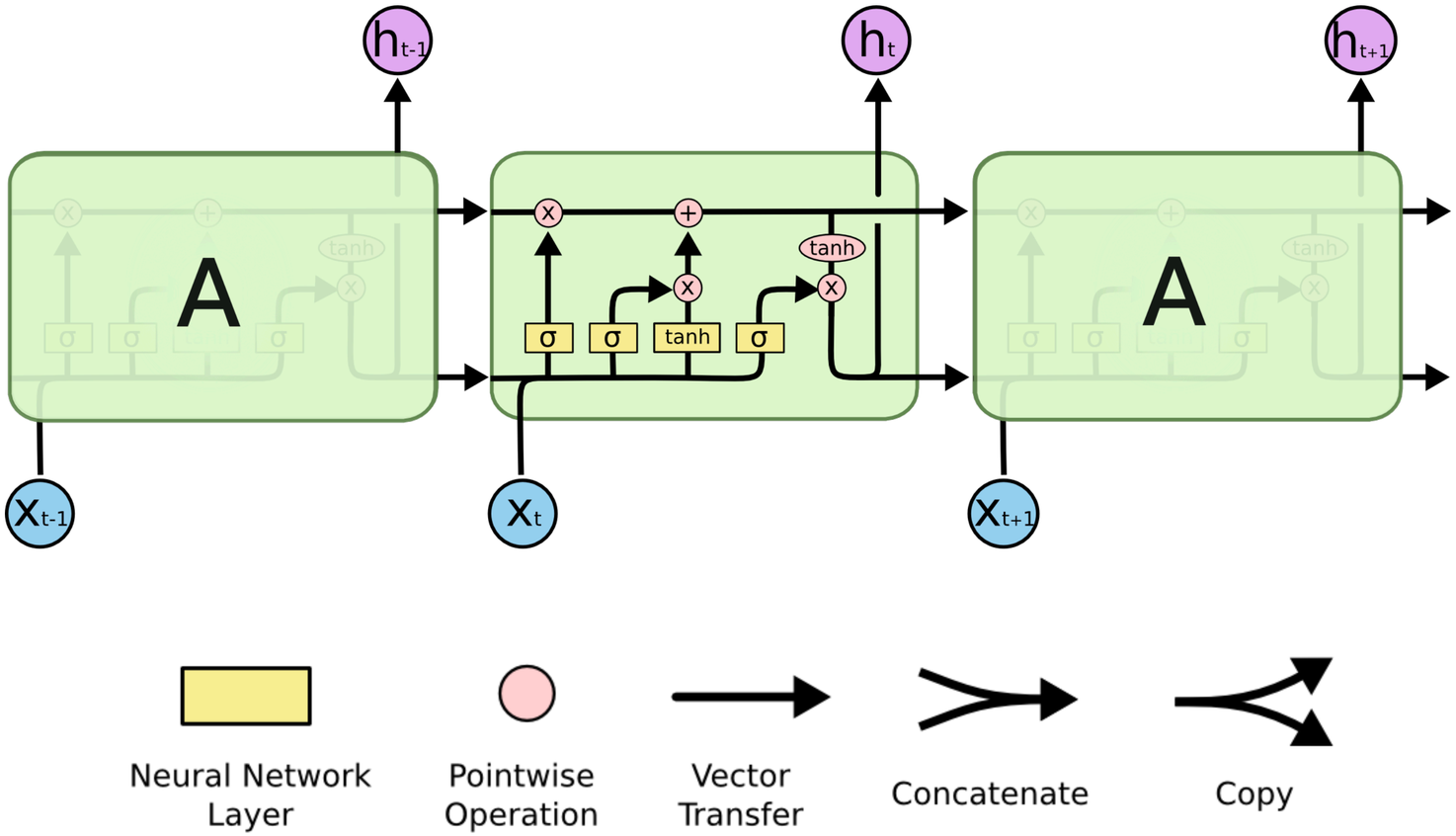
LSTMs

Given a sequence of inputs x_t , we will compute a sequence of hidden states h_t and cell states c_t

At timestep t :

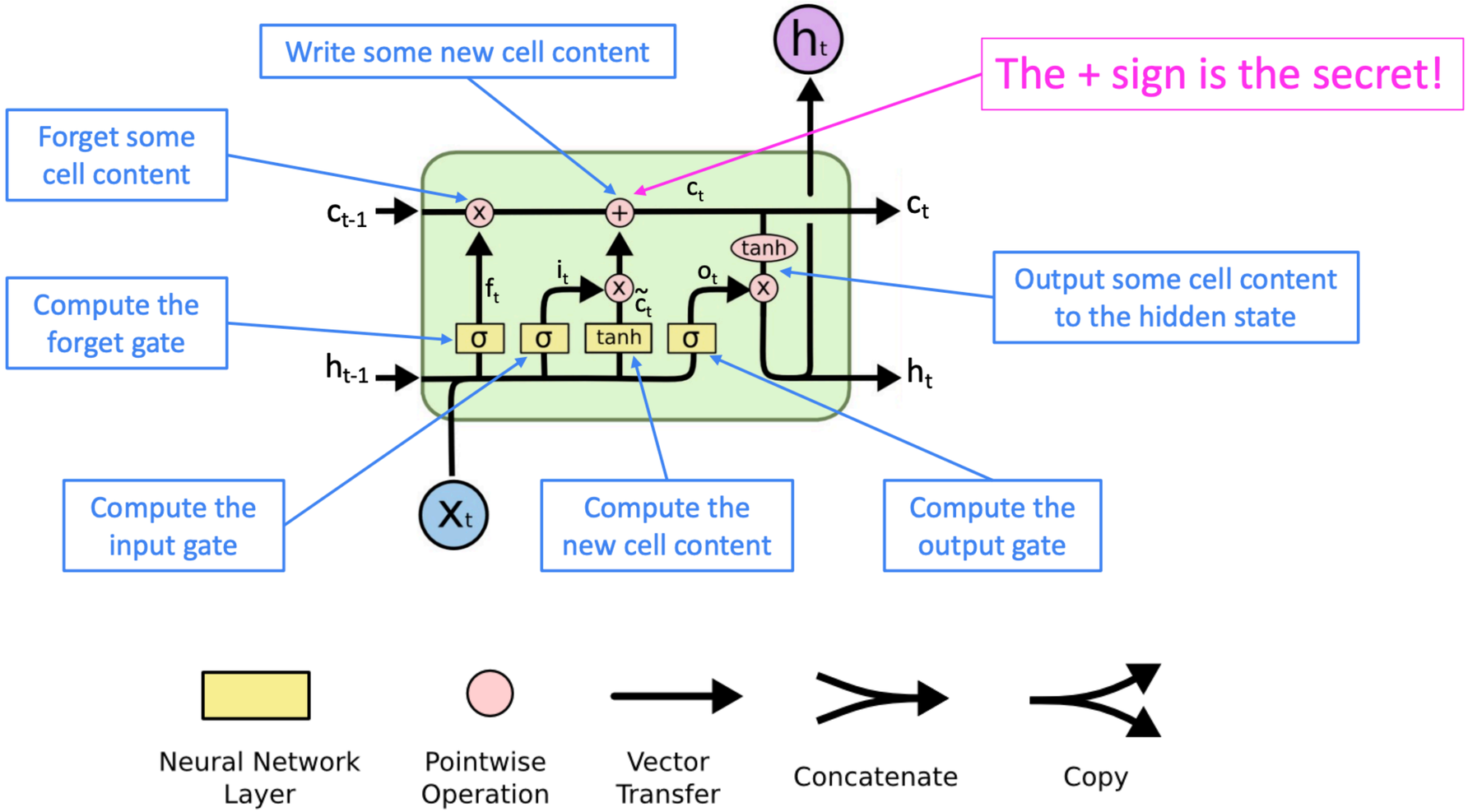


LSTMs: A Visual Representation



Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

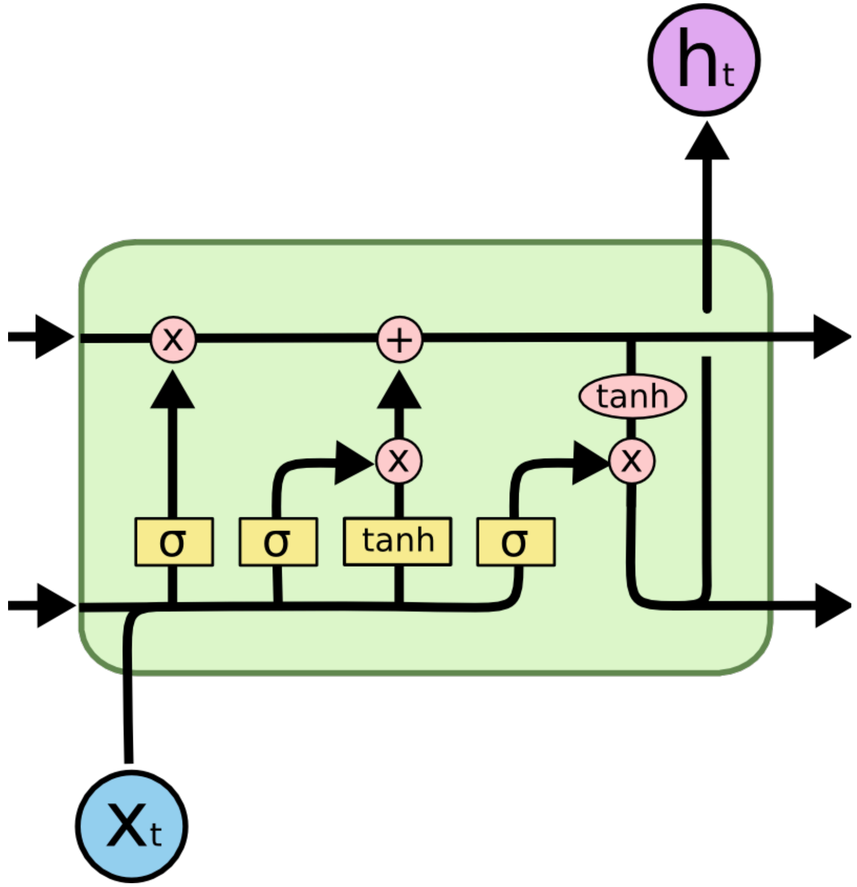
LSTMs: A Visual Representation



Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

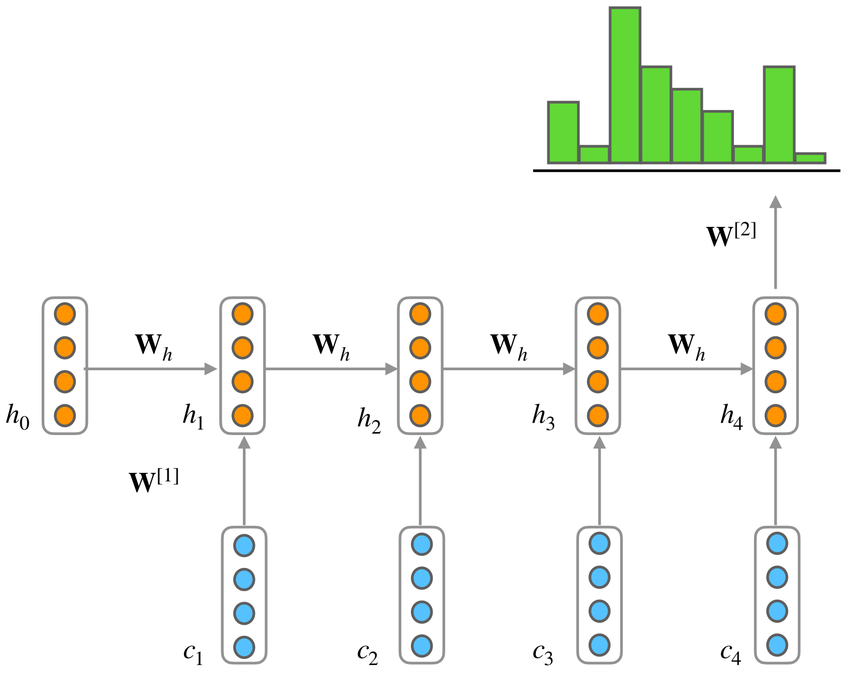
LSTMs: Summary

- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely
- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks
  - We'll look into machine translation next!



Summarizing RNNs

- Recurrent Neural Networks processes sequences one element at a time
- RNNs do not have
  - the limited context problem of n-gram models
  - the fixed context limitation of feedforward LMs
  - since the hidden state can *in principle* represent information about all of the preceding words all the way back to the beginning of the sequence
- But training RNNs is hard
  - Vanishing gradient problem
  - LSTMs address it by incorporating a memory



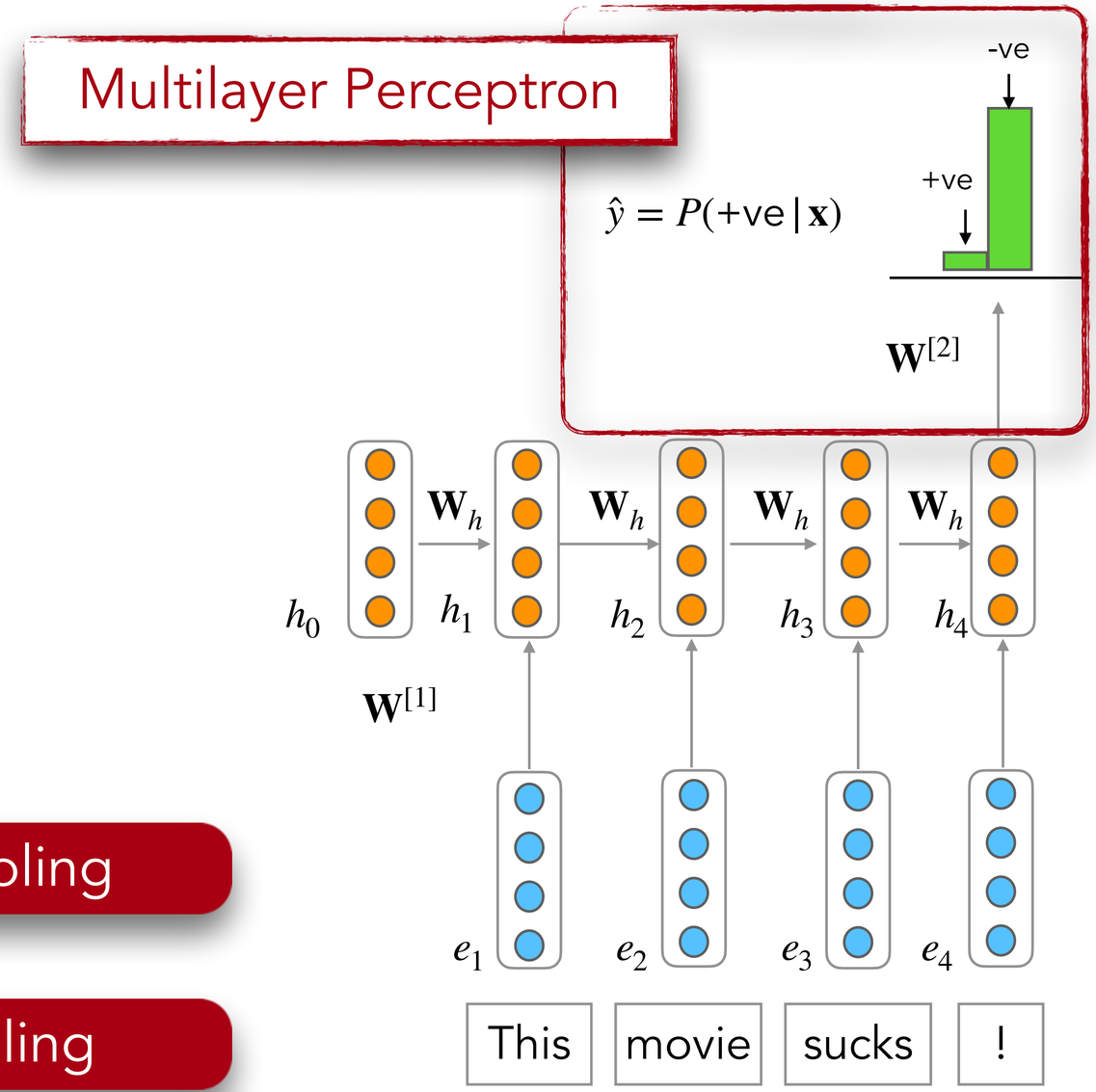
Applications of RNNs

RNNs for Sequence Classification

- $\mathbf{x}$ = Entire sequence / document of length n
- y = (Multivariate) labels
- Pass $\mathbf{x}$ through the RNN one word at a time generating a new hidden state at each time step
- Hidden state for the last token of the text, $\mathbf{h}_n$ is a compressed representation of the entire sequence
- Pass $\mathbf{h}_n$ to a feedforward network (or multilayer perceptron) that chooses a class via a softmax over the possible classes
- Better sequence representations?
  - could also average all $\mathbf{h}_i$'s or
  - consider the maximum element along each dimension

Mean pooling

Max pooling



Training RNNs for Sequence Classification

- Don't need intermediate outputs for the words in the sequence preceding the last element
- Loss function used to train the weights in the network is based entirely on the final text classification task
  - Cross-entropy loss
- Backprop: error signal from the classification is backpropagated all the way through the weights in the feedforward classifier through, to its input, and then through to the three sets of weights in the RNN