Lecture 7:
Recurrent Neural Networks

Instructor: Xiang Ren

USC CSCI 444 NLP

2026 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Logistics / Announcements

- HW2 released today, due on 3/4
- Project Proposal due on 2/11
- HW1 graded by 2/18

Feb 11	Recurrent Neural Nets	J&M, Chap 13;	Project Proposal Due
Feb 16	Presidents Day		
Feb 18	Seq2Seq and Attention	J&M, Chap 8;	
Feb 23	Transformers - Building Blocks	J&M, Chap 8;	
Feb 25	PyTorch for Transformers		
Mar 2	Transformer Language Models	J&M Chap 8;	
Mar 4	Tokenization	J&M, Chap 2.5;	HW2 Due

Recap

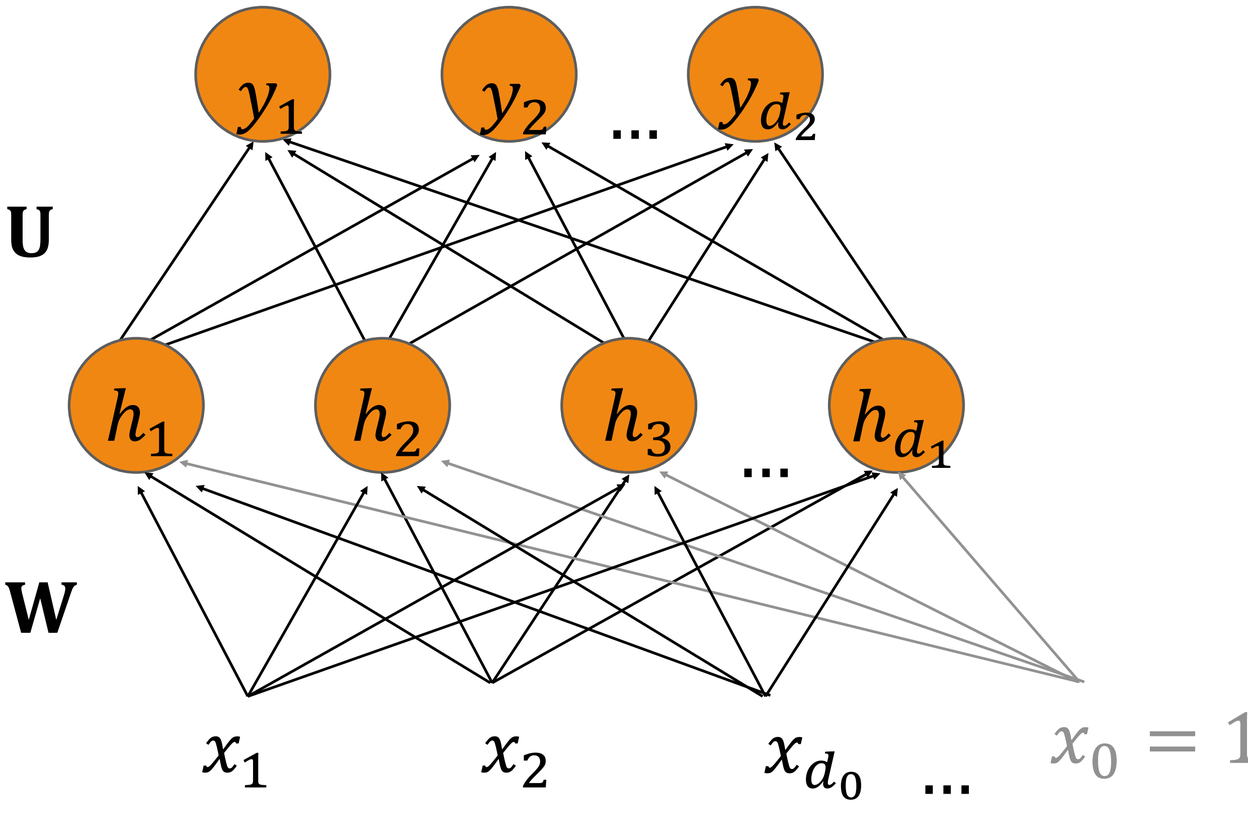
Two-layer FFNN: Notation

Output layer: $\mathbf{y} = \mathit{softmax}(\mathbf{U} \cdot \mathbf{h})$

Hidden layer: $\mathbf{h} = g(\mathbf{W}\mathbf{x}) = g(\sum_{i=0}^{d_0} W_{ji} x_i)$

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



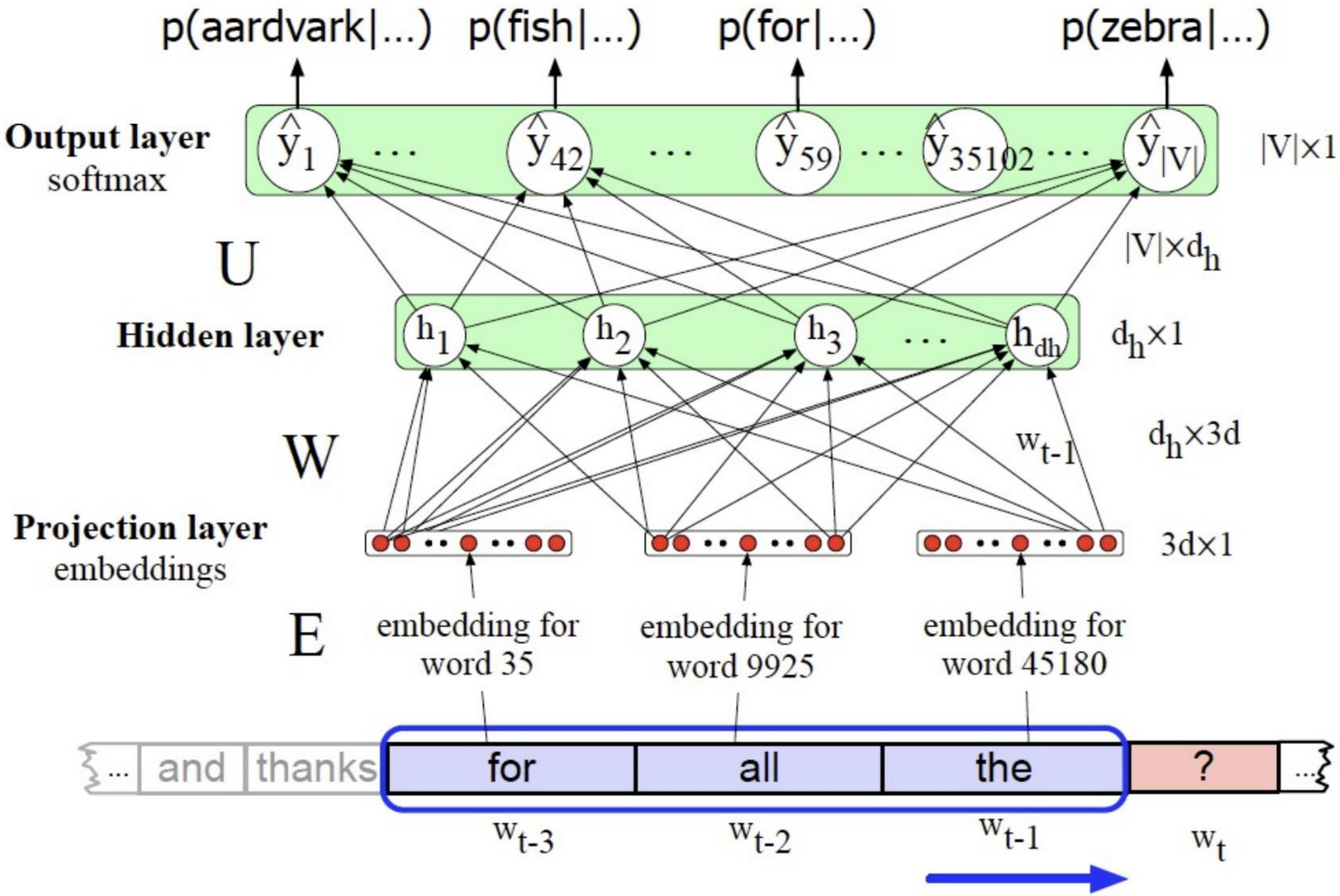
Data: Feedforward Language Model

- Self-supervised
- Computation is divided into time steps t, where different sliding windows are considered
- $x_t = (w_{t-1}, \ldots, w_{t-M+1})$ for the context
  - represent words in this prior context by their embeddings, rather than just by their word identity as in n-gram LMs
  - allows neural LMs to generalize better to unseen data / similar data
  - All embeddings in the context are concatenated
- $y_t = w_t$ for the next word
  - Represented as a one hot vector of vocabulary size where only the ground truth gets a value of 1 and every other element is a 0

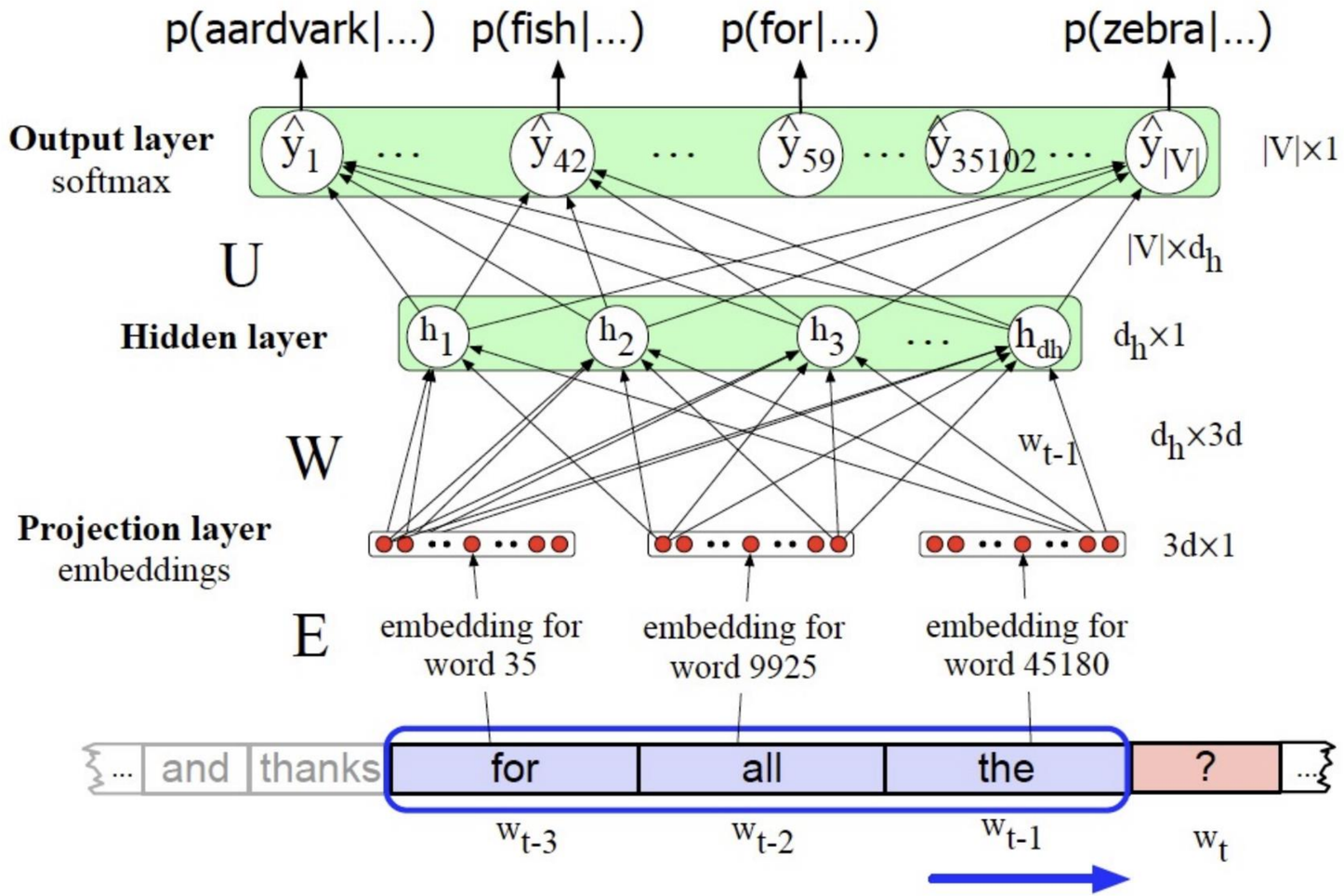
One-hot vector

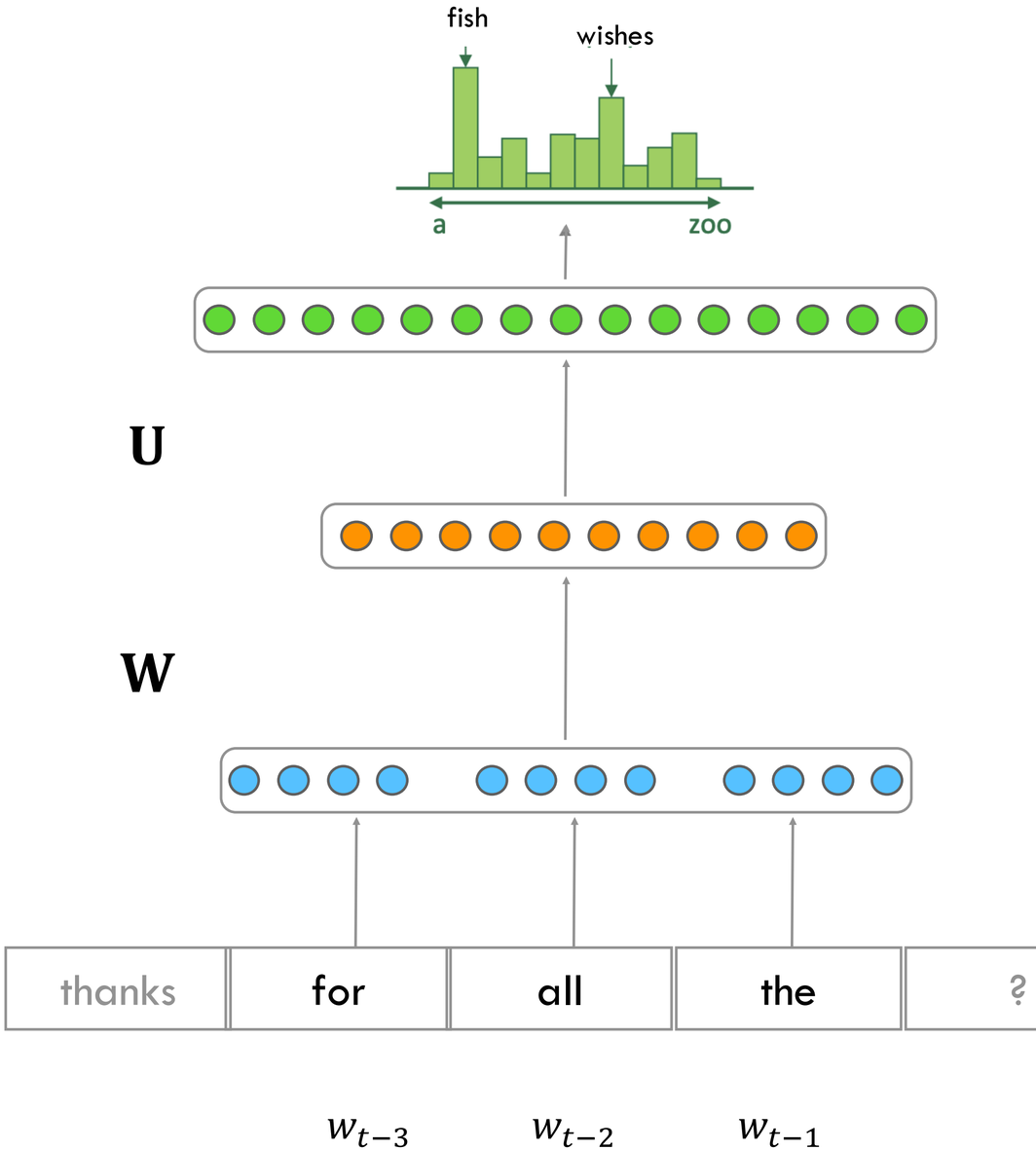
Feedforward Neural LM

- Sliding window of size 4 (including the target word)
- Every feature in the embedding vector connected to every single hidden unit
- Projection / embedding layer is a kind of input layer
  - This is where we plug in our word2vec embeddings
  - May or may not update embedding weights



Simplified Representation





Intuition: Training a 2-layer Network

Training instance $\mathbf{y}$

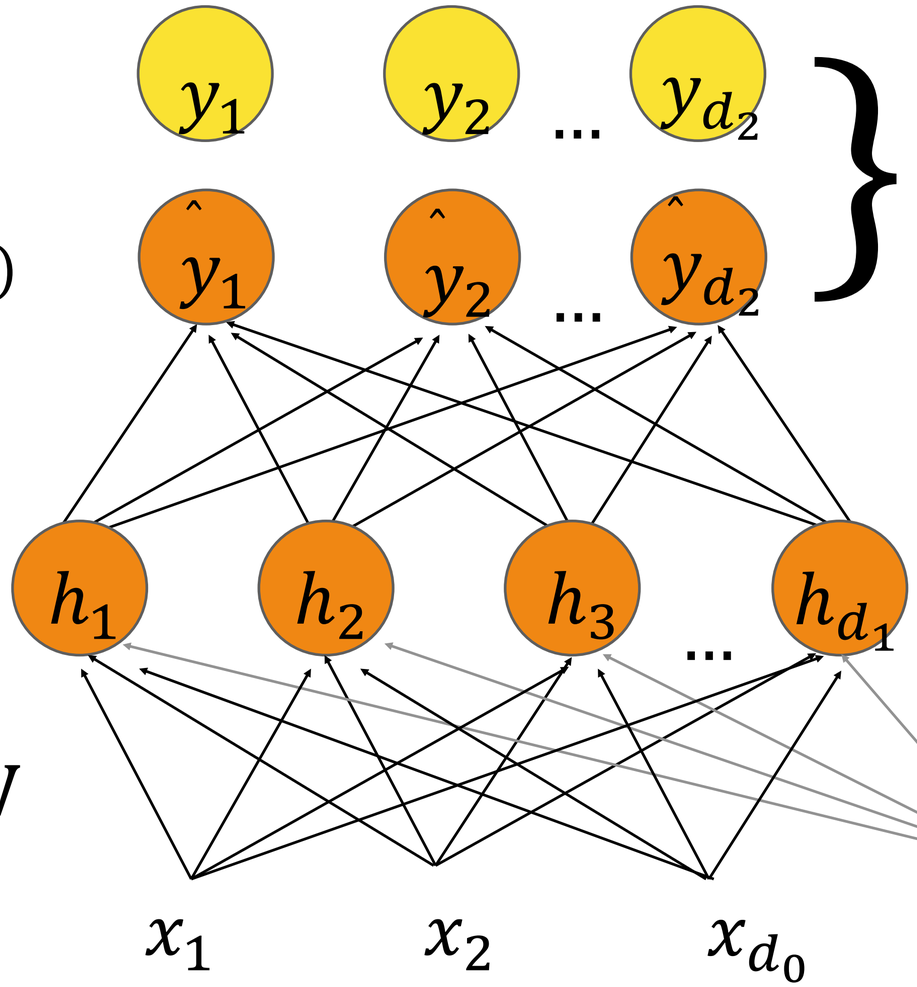
Model Output $\hat{\mathbf{y}} = softmax(\mathbf{U} \cdot \mathbf{h})$

Forward Pass

$\mathbf{U}$

$\mathbf{W}$

Training instance $\mathbf{x}$

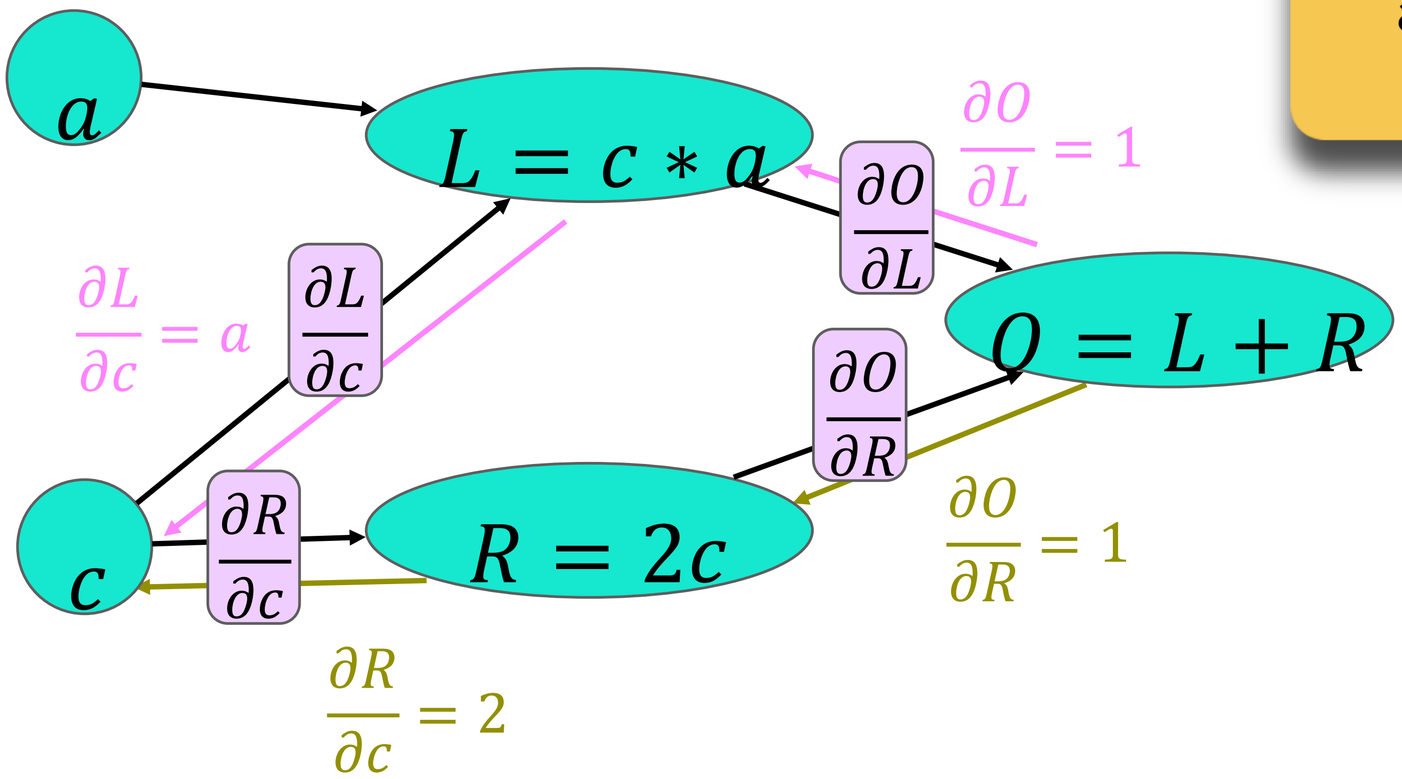


Loss function $L(\hat{\mathbf{y}}, \mathbf{y})$

Backward Pass

Example: Two Paths

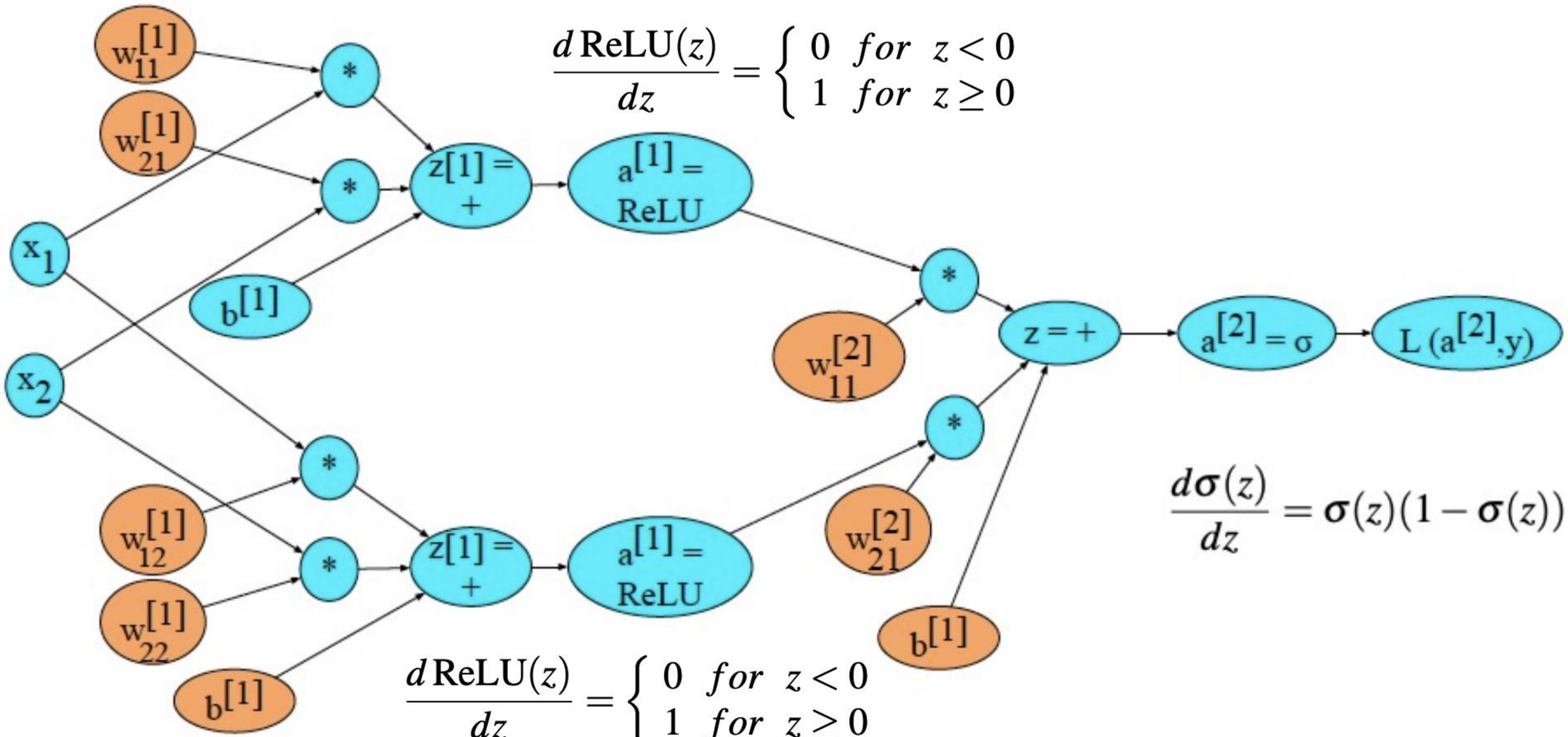
When multiple branches converge on a single node we will add these branches



$$\frac{\partial O}{\partial c} = \frac{\partial O}{\partial L}\frac{\partial L}{\partial c} + \frac{\partial O}{\partial R}\frac{\partial R}{\partial c}$$

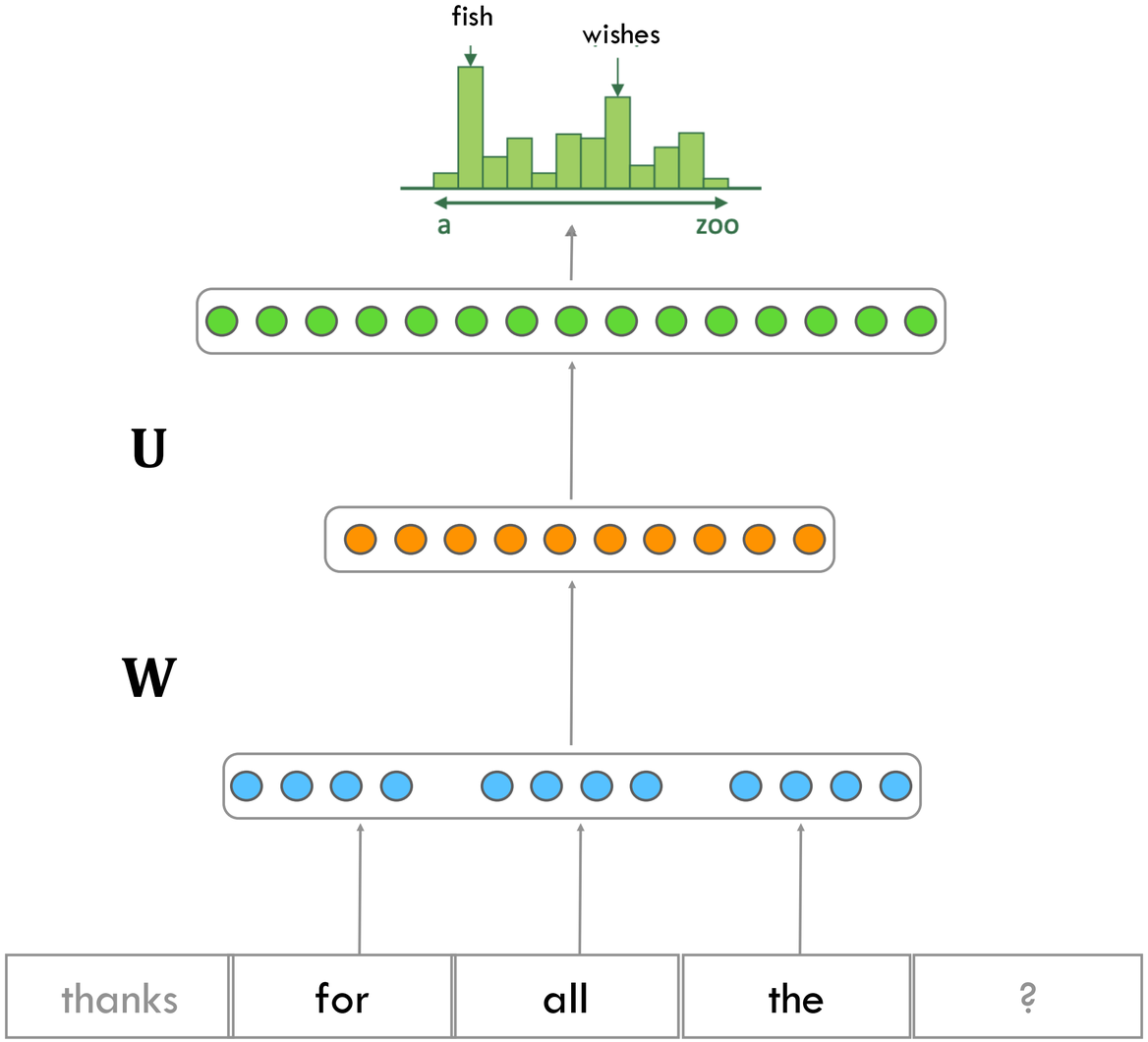
Such cases arise when considering regularized loss functions

2 layer MLP with 2 input features



Feedforward LMs: Windows

- The goodness of the language model depends on the size of the sliding window!
- Fixed window can be too small
- Enlarging window enlarges $\mathbf{W}$
- Each word uses different rows of $\mathbf{W}$. We don't share weights across the window.
- Window can never be large enough!



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
- 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

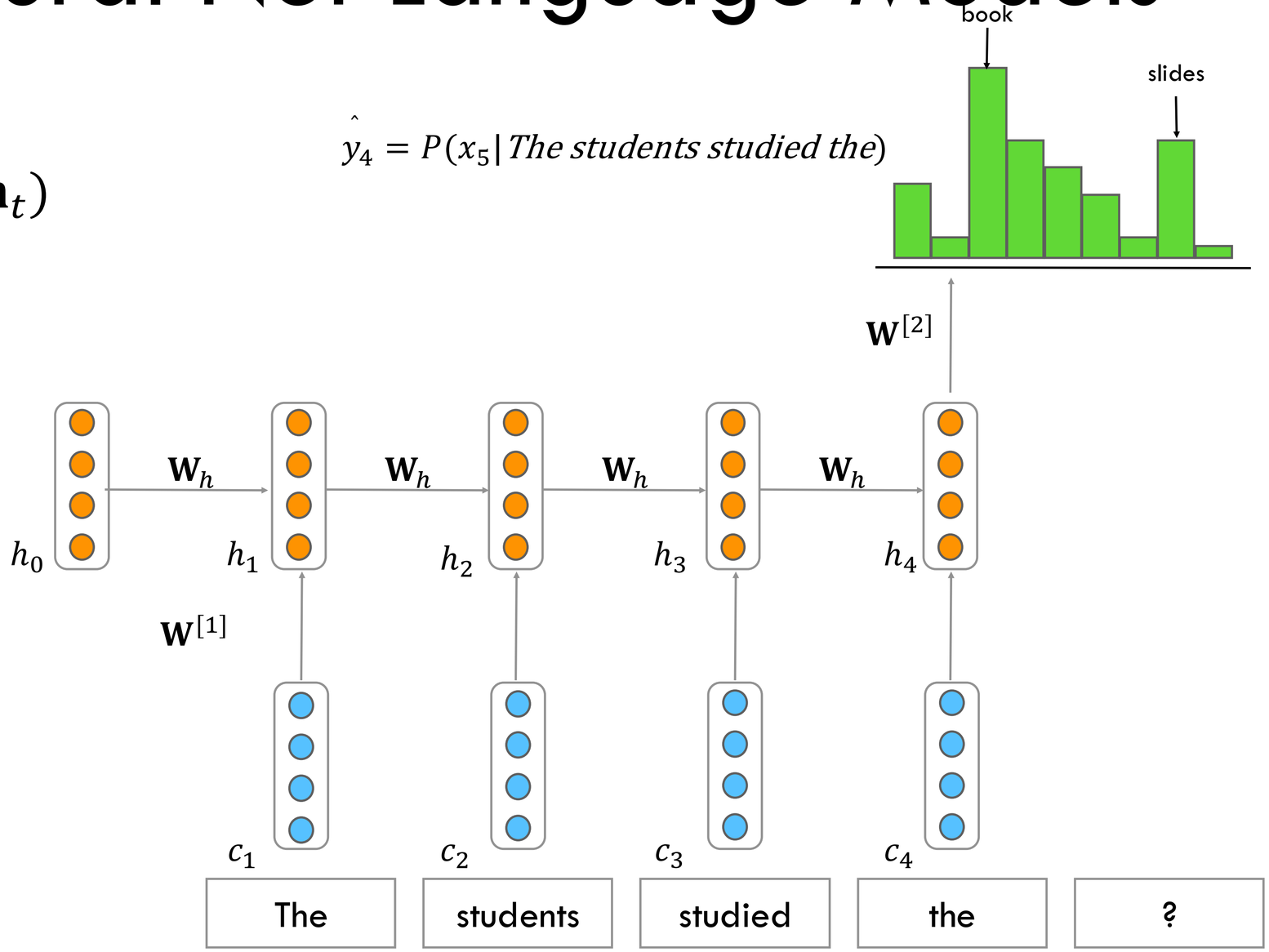
Recurrent Neural Net Language Models

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

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

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

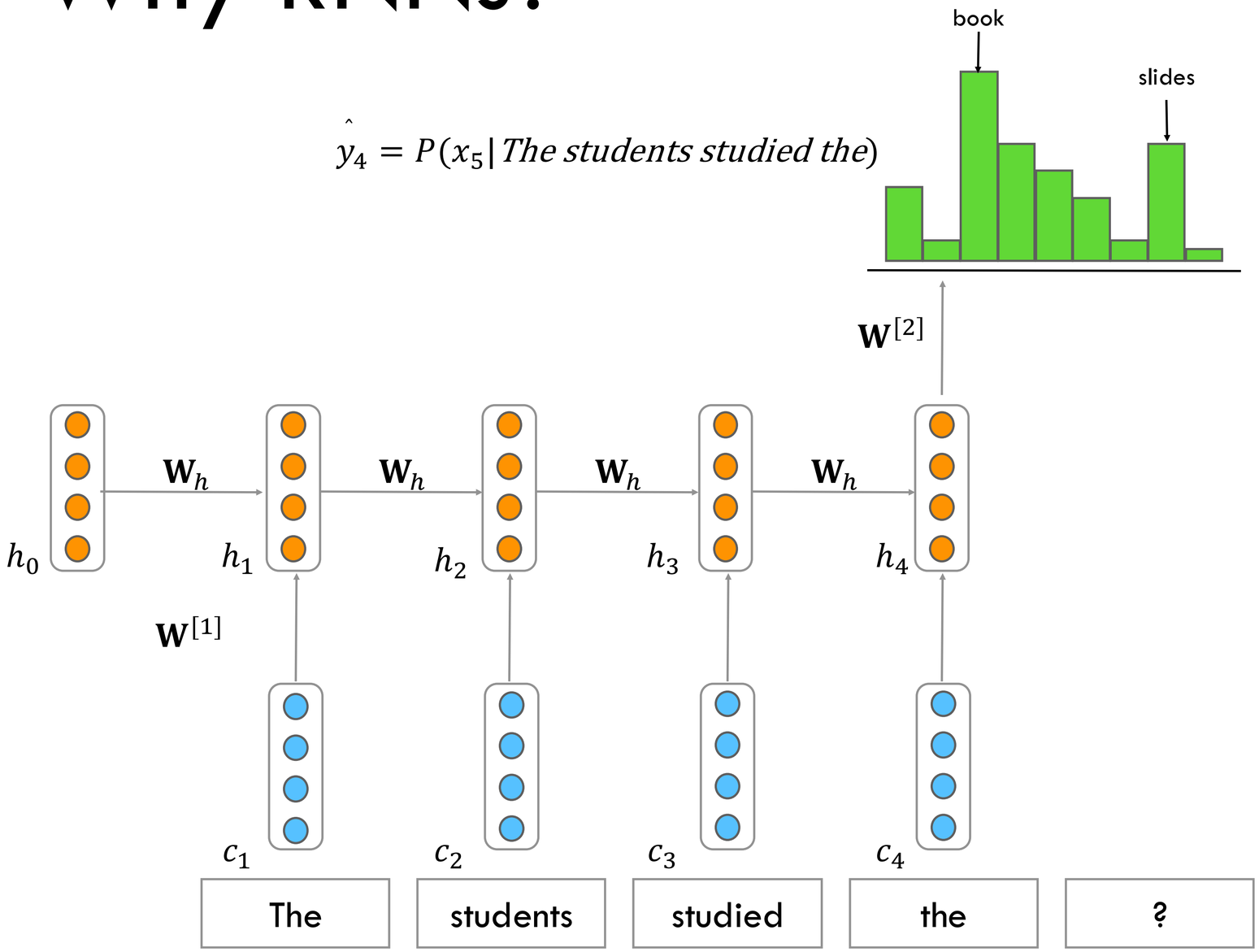
Word Embeddings, $\mathbf{c}_i$



Why RNNs?

RNN Advantages:

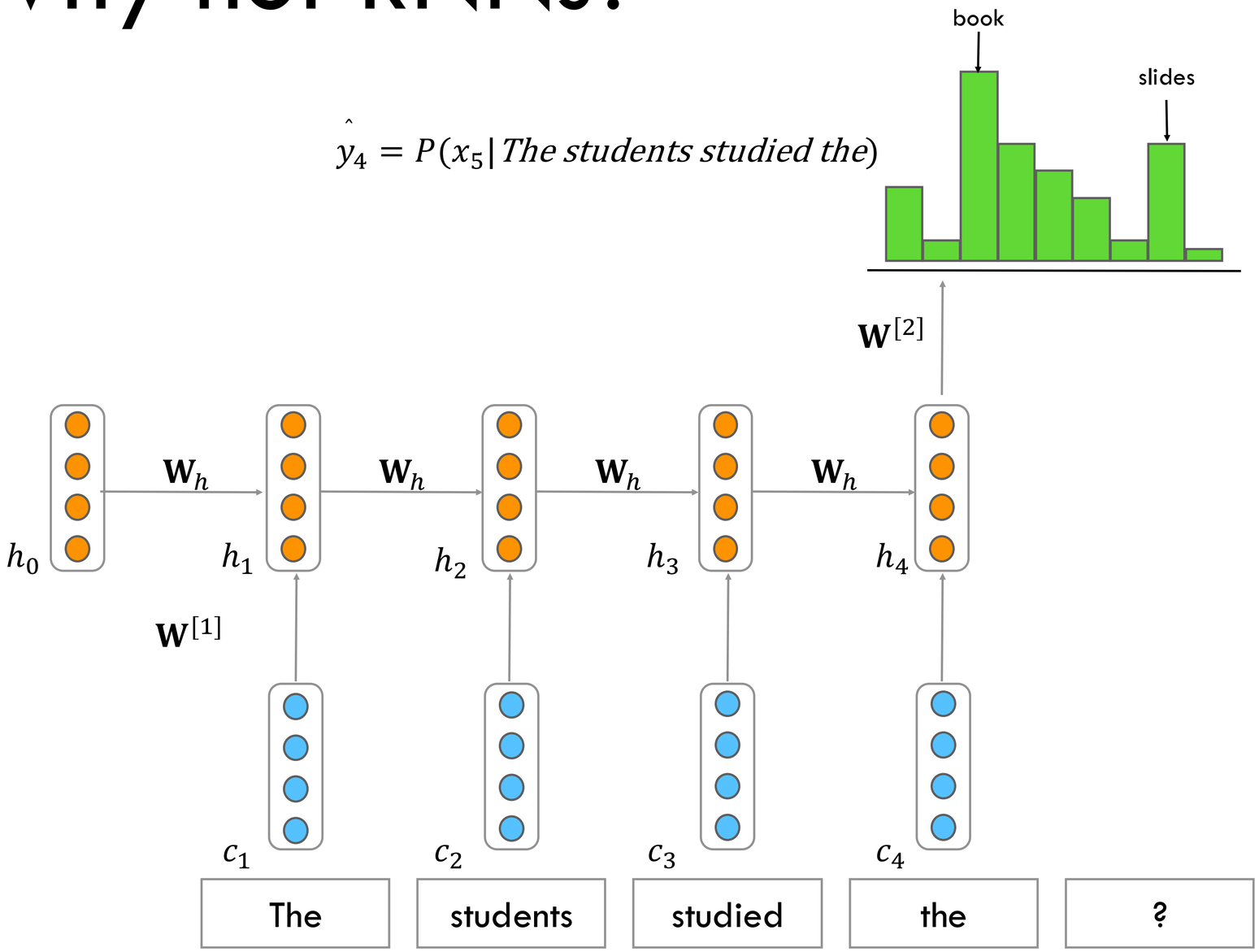
- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- Weights $\mathbf{W}^{[1]}$ are shared (tied) across timesteps → Condition the neural network on all previous words



Why not RNNs?

RNN Disadvantages:

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



Training RNNLMs

Training Outline

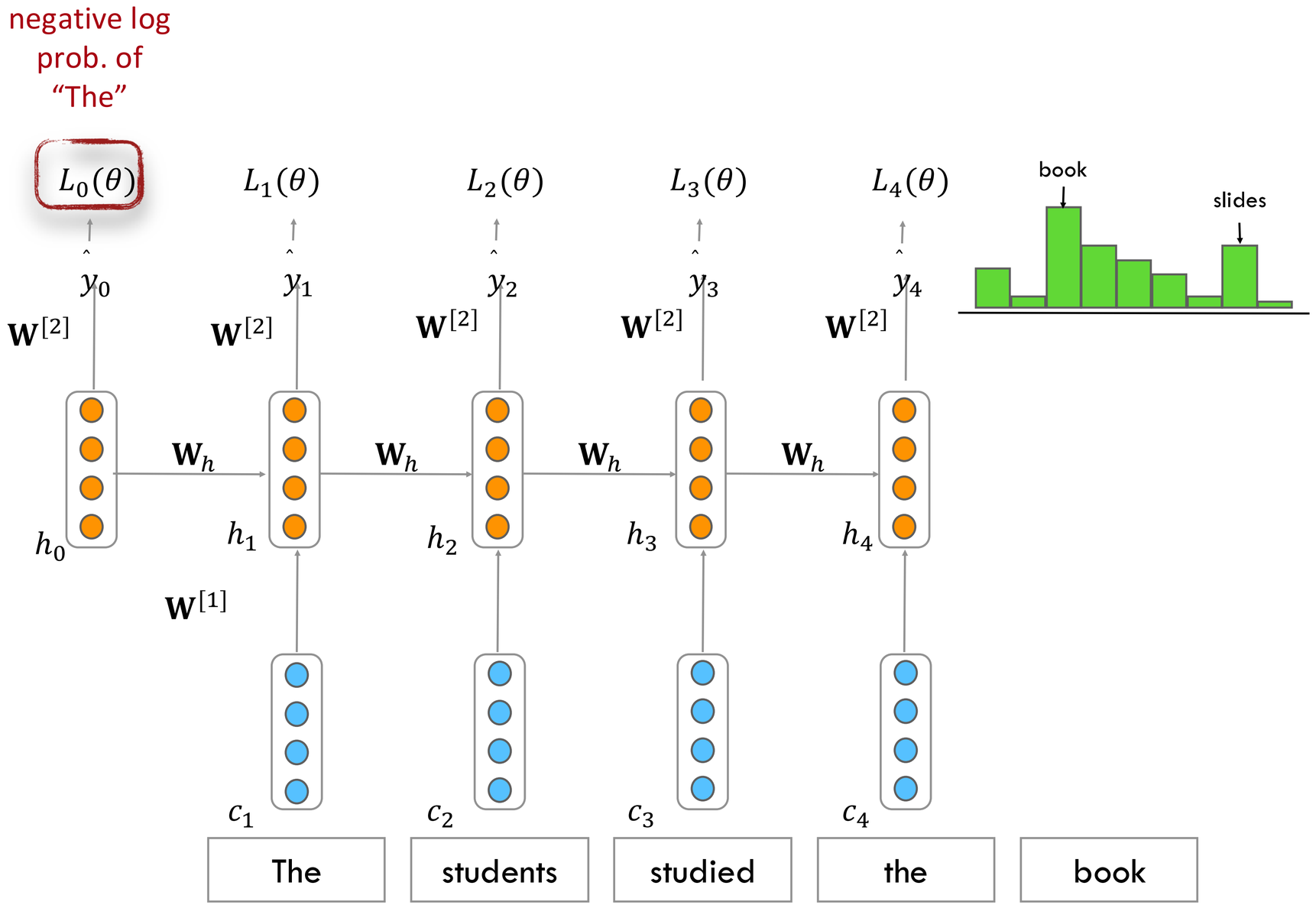
- Get a big corpus of text which is a sequence of words $x_1, x_2, \ldots x_T$
- Feed into RNN-LM; compute output distribution $\hat{y}_t$ for every step t
  - i.e. predict probability distribution of every word, given words so far
- Loss function on step t is usual cross-entropy between our predicted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

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

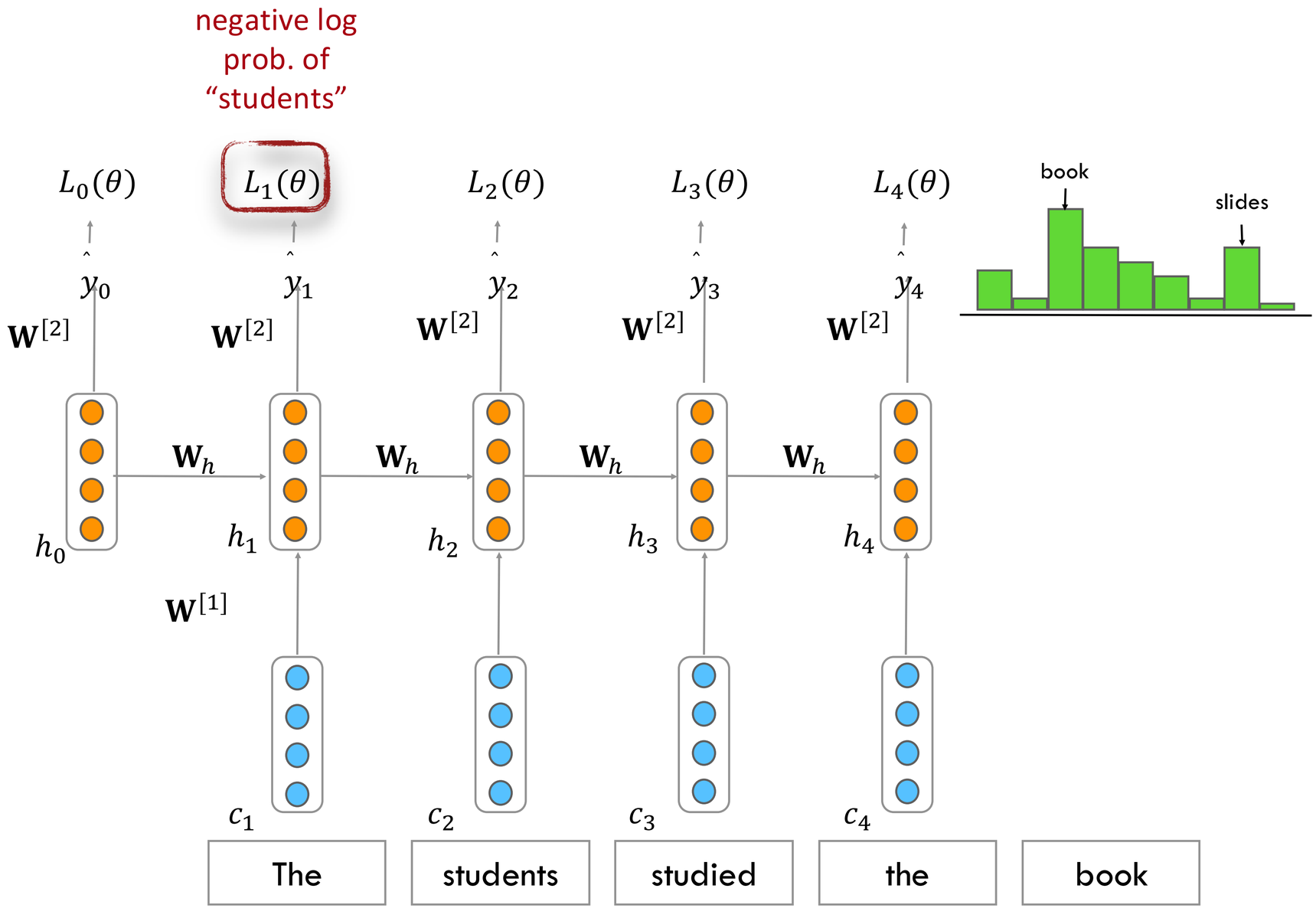
- Average this to get overall loss for entire training set:

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

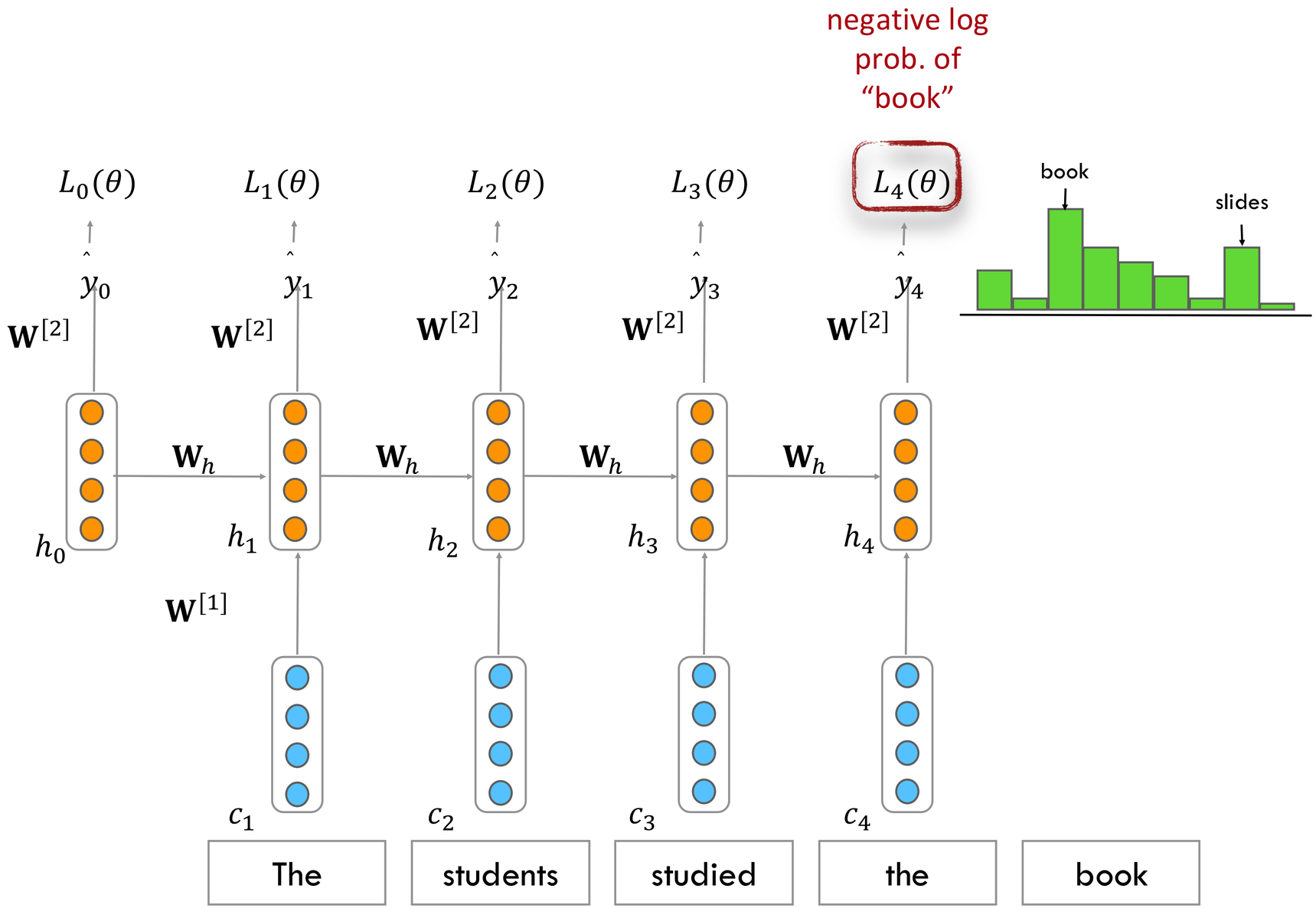
Loss


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
c_1
c_2
c_3
c_4
The
students
studied
the
book
book
slides

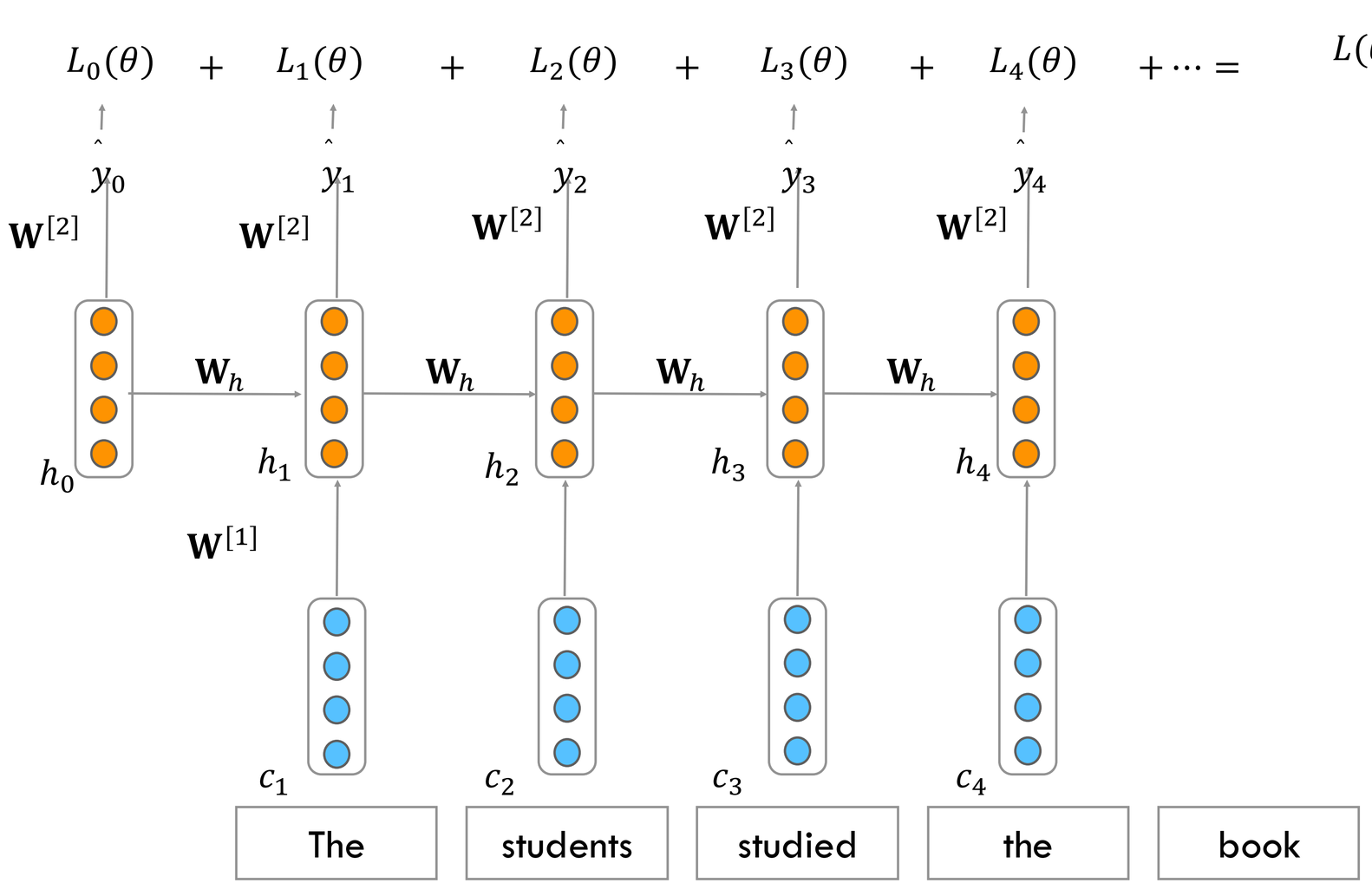
Loss


negative log prob. of "students"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
c_1
c_2
c_3
c_4
The
students
studied
the
book
book
slides

Loss

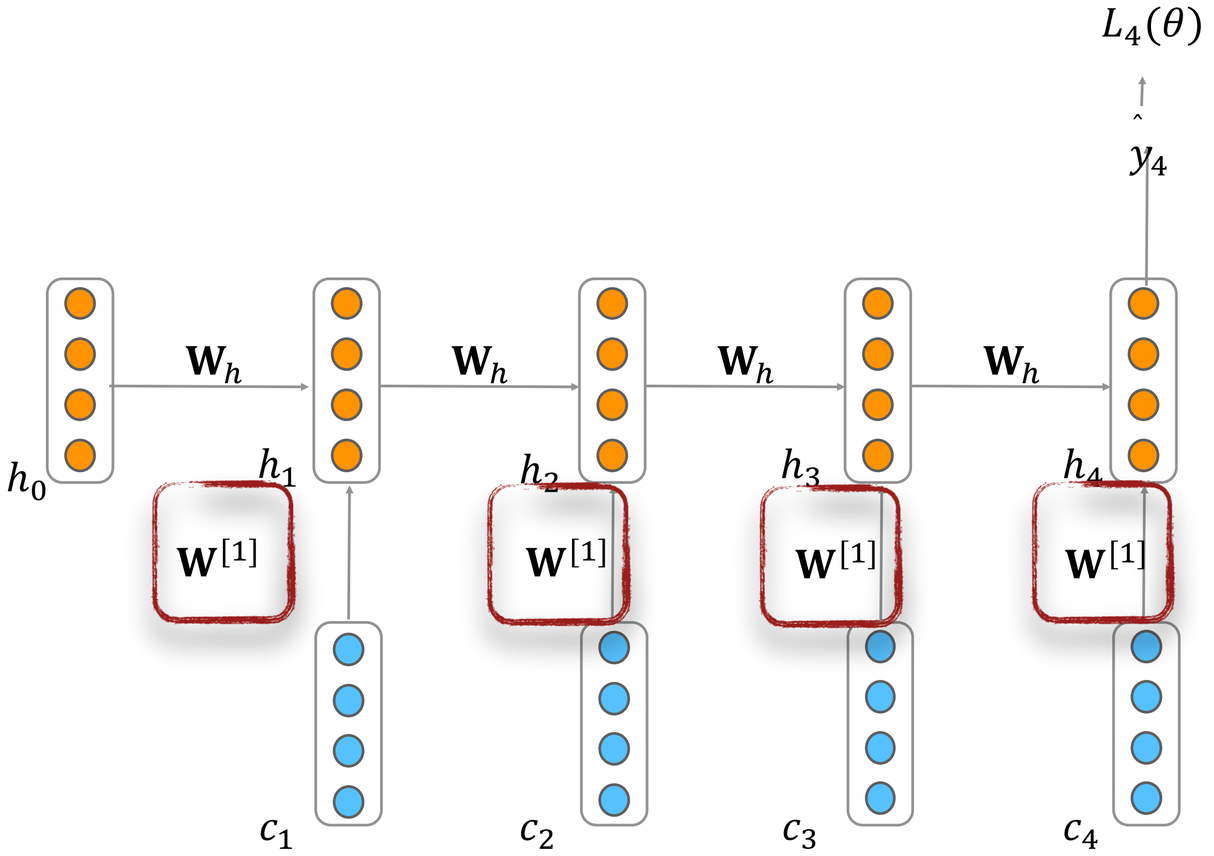

negative log prob. of "book"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
book
slides
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
c_1
c_2
c_3
c_4
The
students
studied
the
book

Loss


$L_0(\theta)$ + $L_1(\theta)$ + $L_2(\theta)$ + $L_3(\theta)$ + $L_4(\theta)$ + ⋯ =
$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta)$
$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
h_0 h_1 h_2 h_3 h_4
$\mathbf{W}^{[1]}$
c_1 c_2 c_3 c_4
The students studied the book

Training RNNs is hard

- 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



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

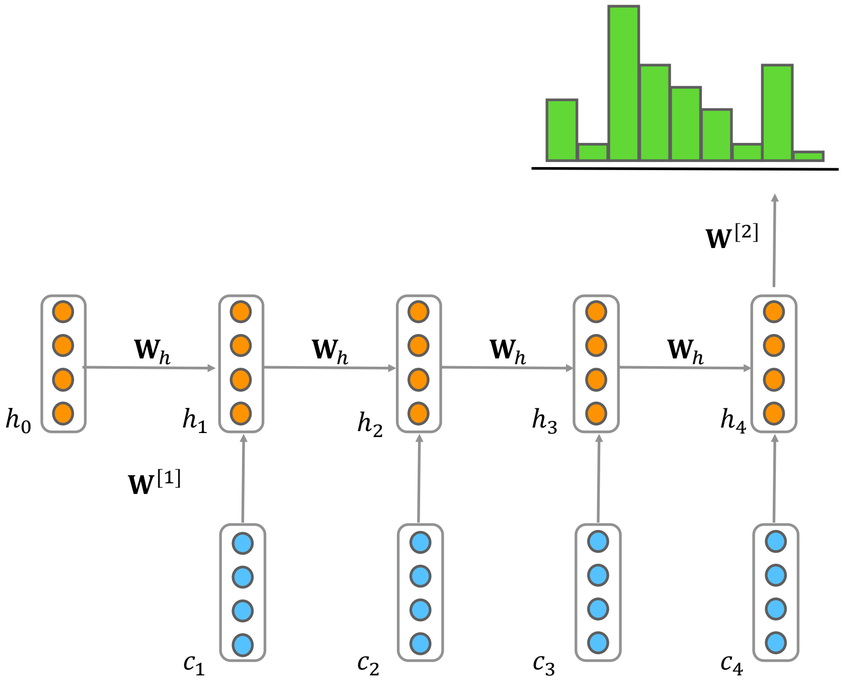
- Computing loss and gradients across entire corpus is too expensive!
- Recall: mini-batch Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
- Solution: consider chunks of text.
  - In practice, consider $x_1, x_2, \ldots x_T$ for some T as a "sentence"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

Summarizing RNNs

- 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
- Training can be expensive and might lead to vanishing gradients
- More advanced architectures: LSTMs (Long Short-Term Memories)



Can be applied to both classification and generation tasks

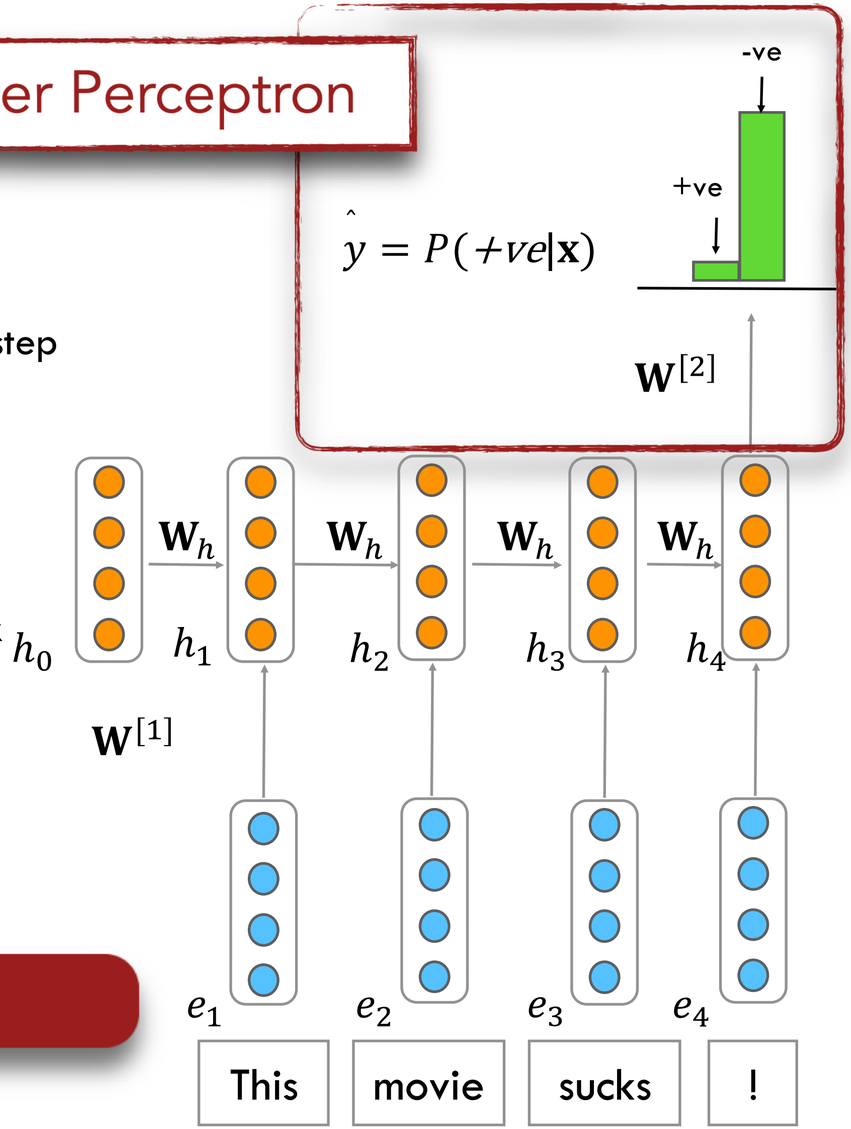
Applications

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 embedding at each time step
- Hidden layer 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

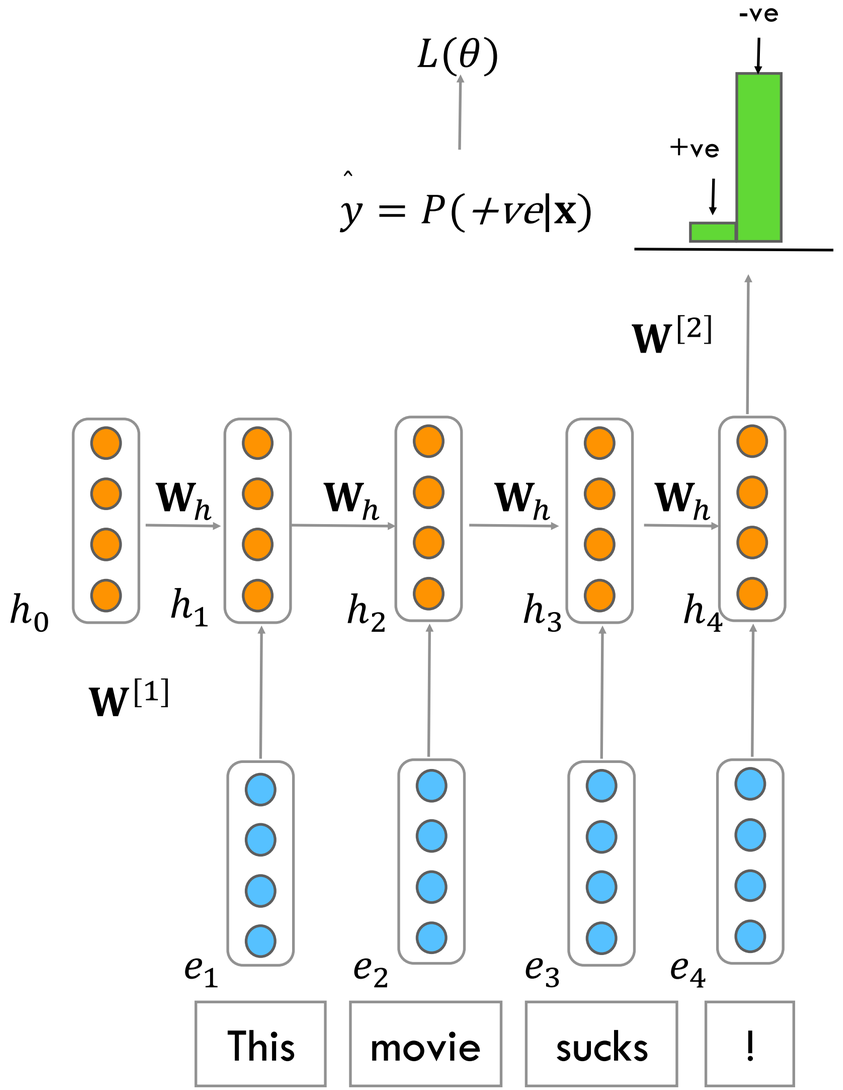
Mean pooling

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



Generation with RNNLMs

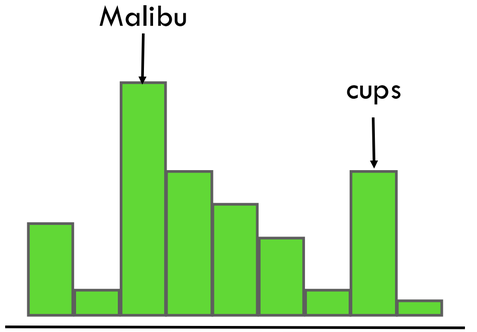
- Similar to sampling from n-gram LMs
- First randomly sample a word to begin a sequence based on its suitability as the start of a sequence
- Then continue to sample words conditioned on our previous choices until
  - we reach a pre-determined length,
  - or an end of sequence token is generated

Remember sampling from n-gram LMs?

1. Choose a random bigram (<s>, w) according to its probability
2. Now choose a random bigram (w, x) according to its probability…and so on until we choose </s>

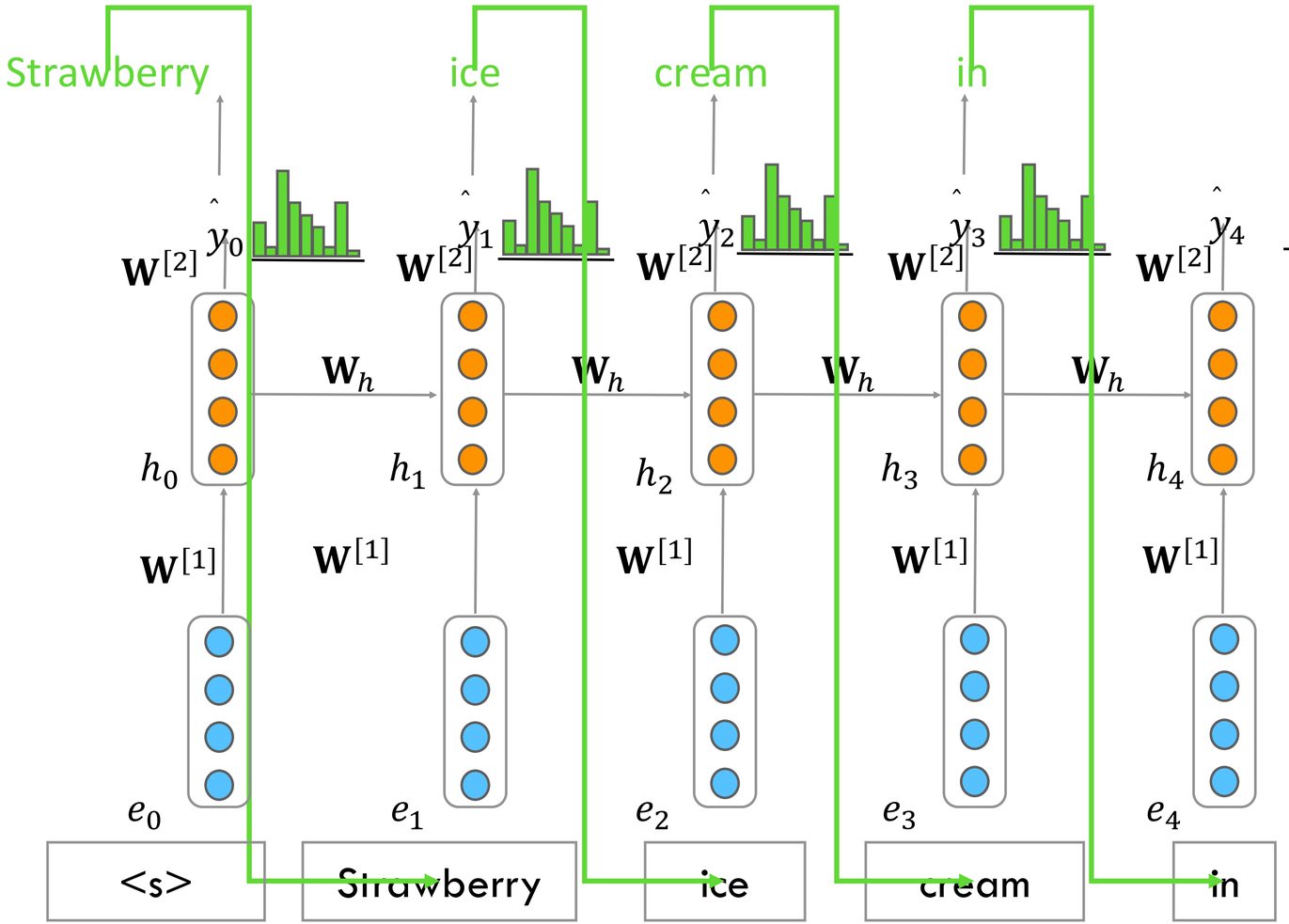
```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

$\hat{y}_4 = P(x_5 | Strawberry\ ice\ cream\ in)$



$\text{arg}max\hat{y}_i$

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



Generation with RNNLMs

1. Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>, as the first input.

2. Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.

3. Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.

Repeated sampling of the next word conditioned on previous choices

Autoregressive Generation

RNNLMs are Autoregressive Models

- Model that predicts a value at time t based on a function of the previous values at times $t-1$, $t-2$, and so on
- Word generated at each time step is conditioned on the word selected by the network from the previous step
- State-of-the-art generation approaches are all autoregressive!
    - Machine translation, question answering, summarization
- Key technique: prime the generation with the most suitable context

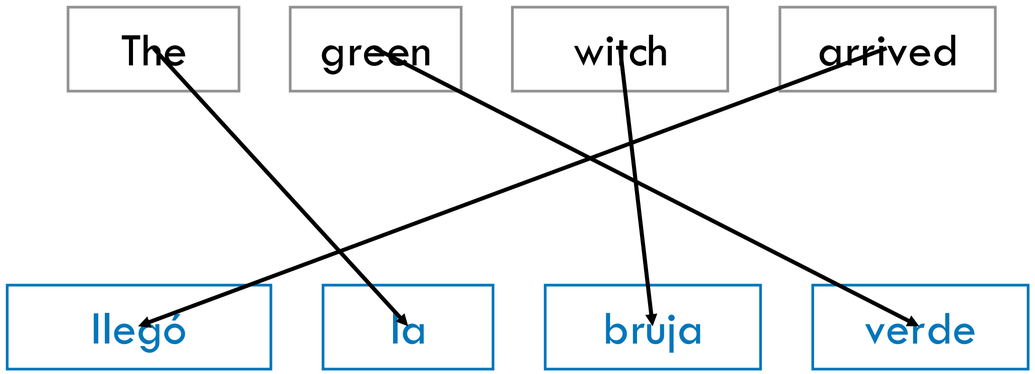
Can do better than <s>!

Provide rich task-appropriate context!

(Neural) Machine Translation

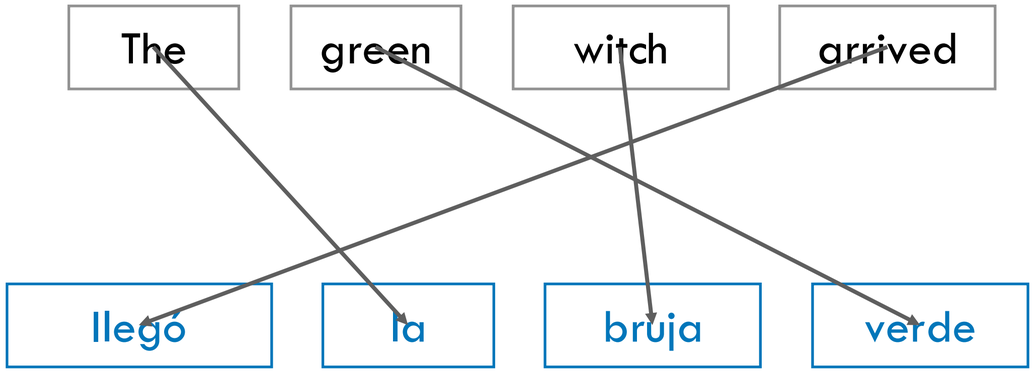
Provide rich task-appropriate context!

- Sequence Generation Problem (as opposed to sequence classification)
  - $\mathbf{x}$ = Source sequence of length n
  - $\mathbf{y}$ = Target sequence of length m
- Different from regular generation from an LM
  - Since we expect the target sequence to serve a specific utility (translate the source)



Sequence-to-Sequence (Seq2seq)

Sequence-to-Sequence Generation



- Mapping between a token in the input and a token in the output can be very indirect
  - in some languages the verb appears at the beginning of the sentence; e.g. Arabic, Hawaiian
  - in other languages at the end; e.g. Hindi
  - in other languages between the subject and the object; e.g. English
- Does not necessarily align in a word-word way!

Need a special architecture to summarize the entire context!

Sequence-to-Sequence Models

- Models capable of generating contextually appropriate, arbitrary length, output sequences given an input sequence.
- The key idea underlying these networks is the use of an encoder network that takes an input sequence and creates a contextualized representation of it, often called the context.
- This representation is then passed to a decoder network which generates a task- specific output sequence.

Encoder-Decoder Networks

Sequence-to-Sequence Modeling with Encoder-Decoder Networks

Encoder-Decoder Networks

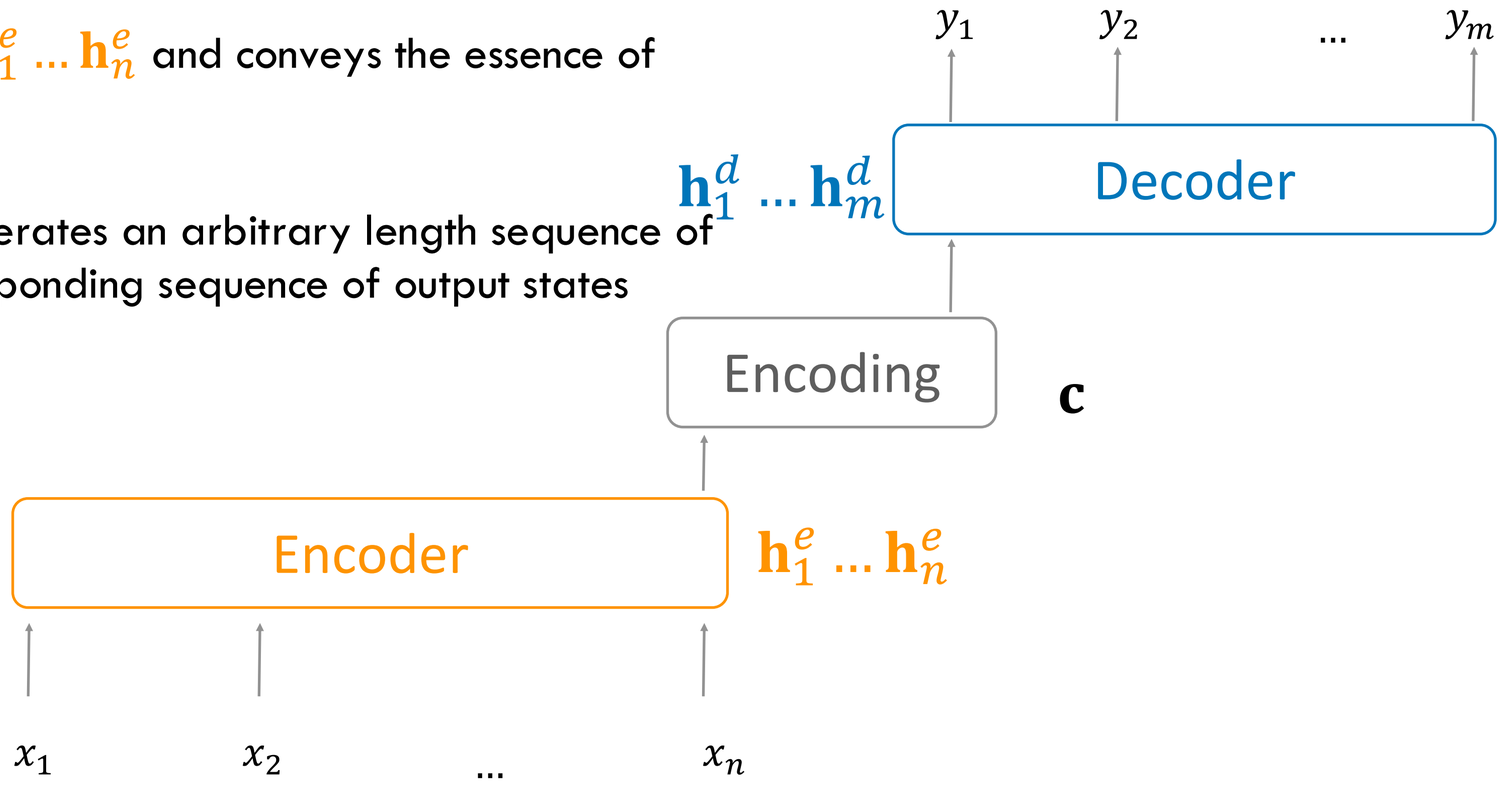
Encoder-decoder networks consist of three components:

1. An encoder that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_1^e \dots \mathbf{h}_n^e$

2. A encoding vector, $\mathbf{c}$ which is a function of $\mathbf{h}_1^e \dots \mathbf{h}_n^e$ and conveys the essence of the input to the decoder

3. A decoder which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_1^d \dots \mathbf{h}_m^d$, from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained

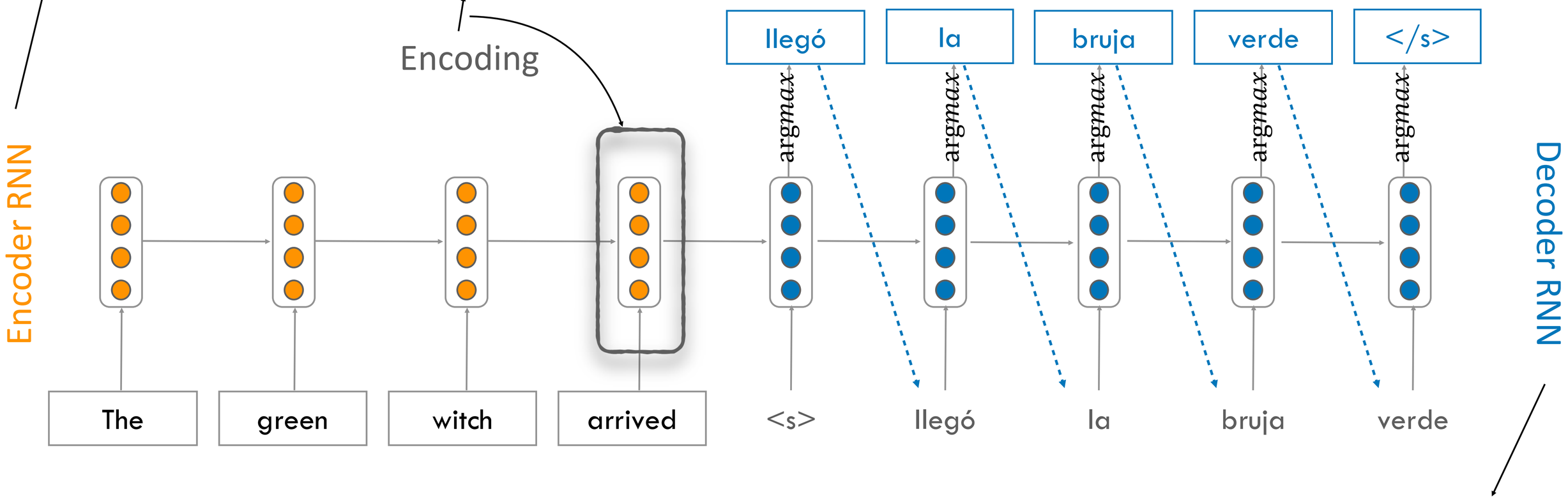
Encoders and decoders can be made of FFNNs, RNNs, or Transformers



Produces an encoding of the source sequence

Represents input sequence. Provides initial hidden state for Decoder RNN

Target Sentence **y**



Source Sentence **x**

Language Model that produces the target sentence conditioned on the encoding

negative log prob. of "llegó"

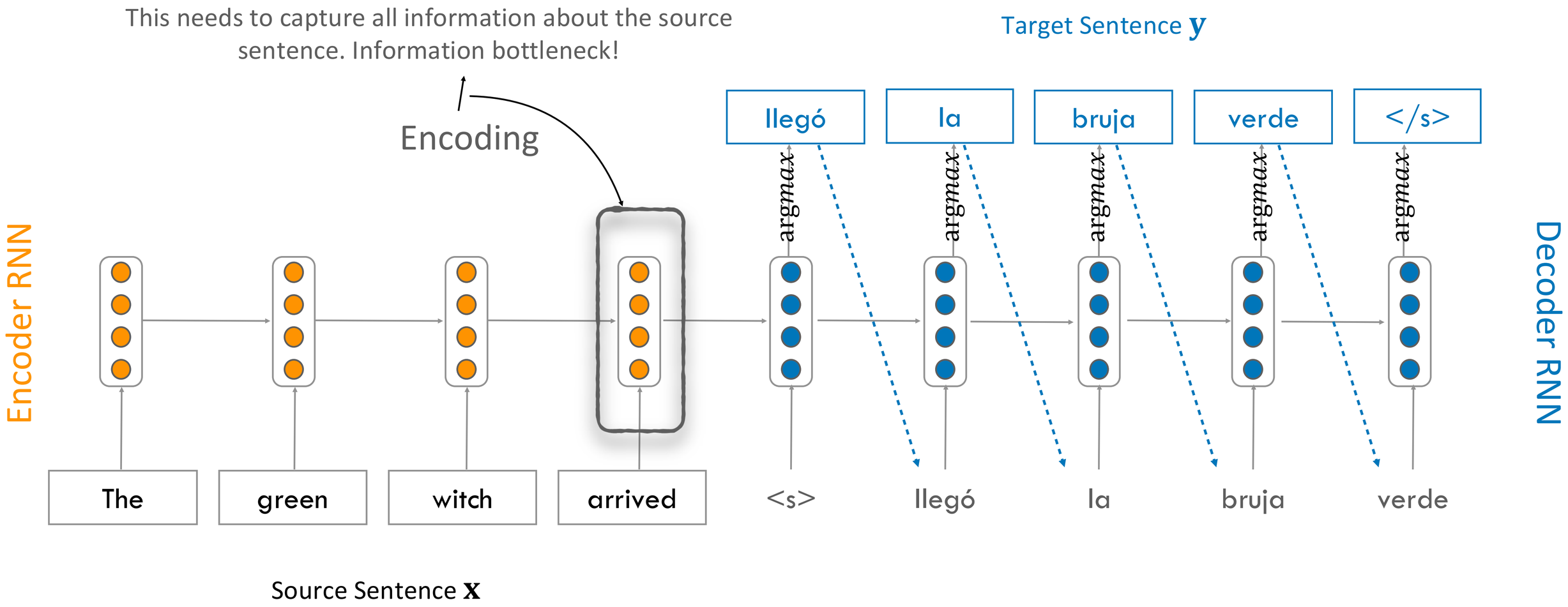
negative log prob. of "</s>"

Loss

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

Source Sentence $\mathbf{x}$

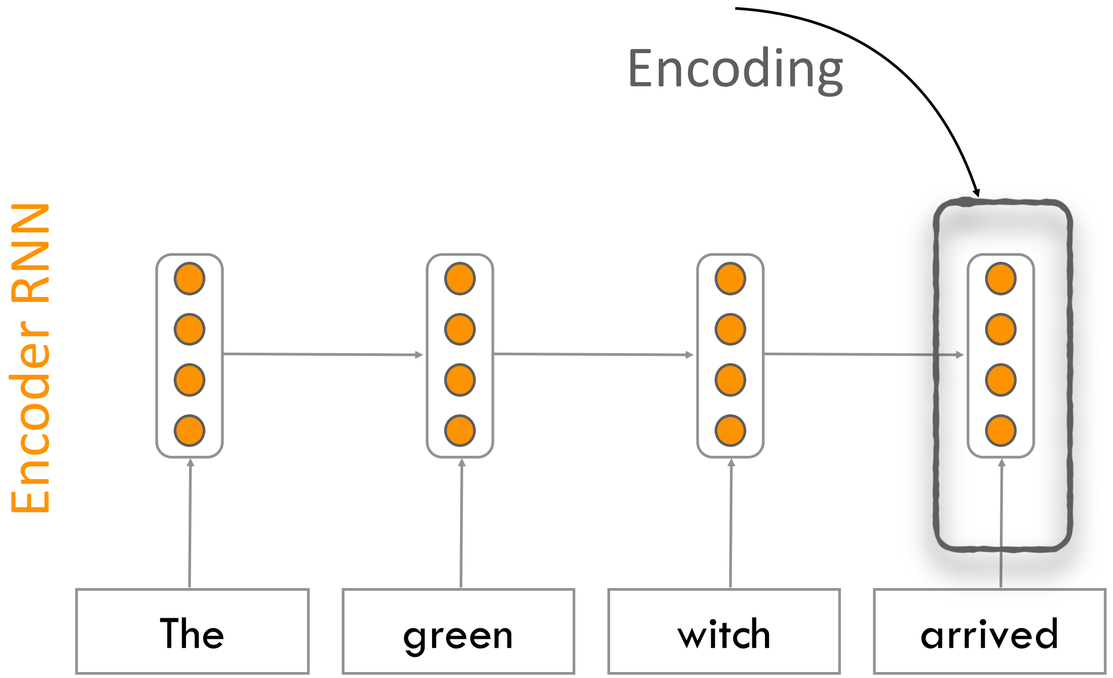
Target Sentence $\mathbf{y}$

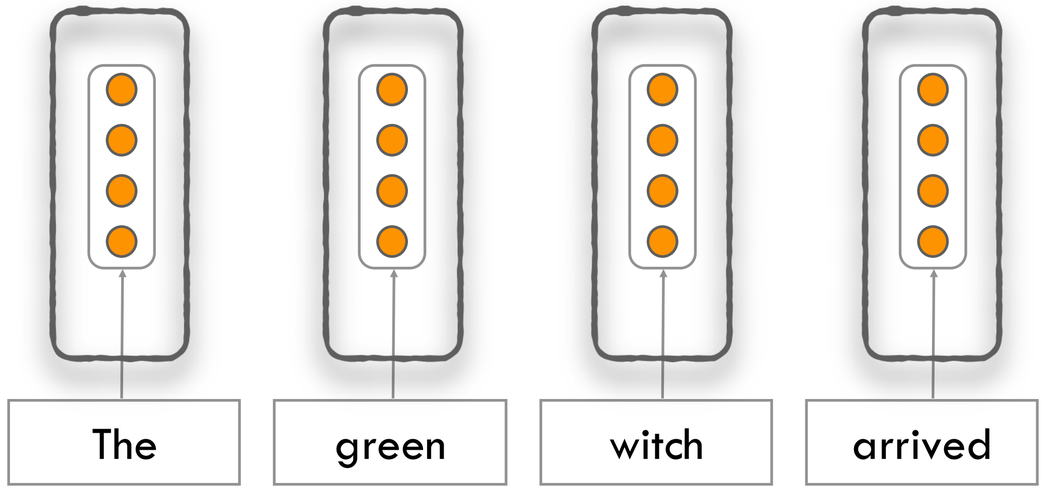

This needs to capture all information about the source sentence. Information bottleneck!
Encoding
Target Sentence y
llegó
la
bruja
verde
</s>
argmax
Encoder RNN
Decoder RNN
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence x

"you can't cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!"

– Ray Mooney, Professor of Computer Science, UT Austin

Information Bottleneck: One Solution





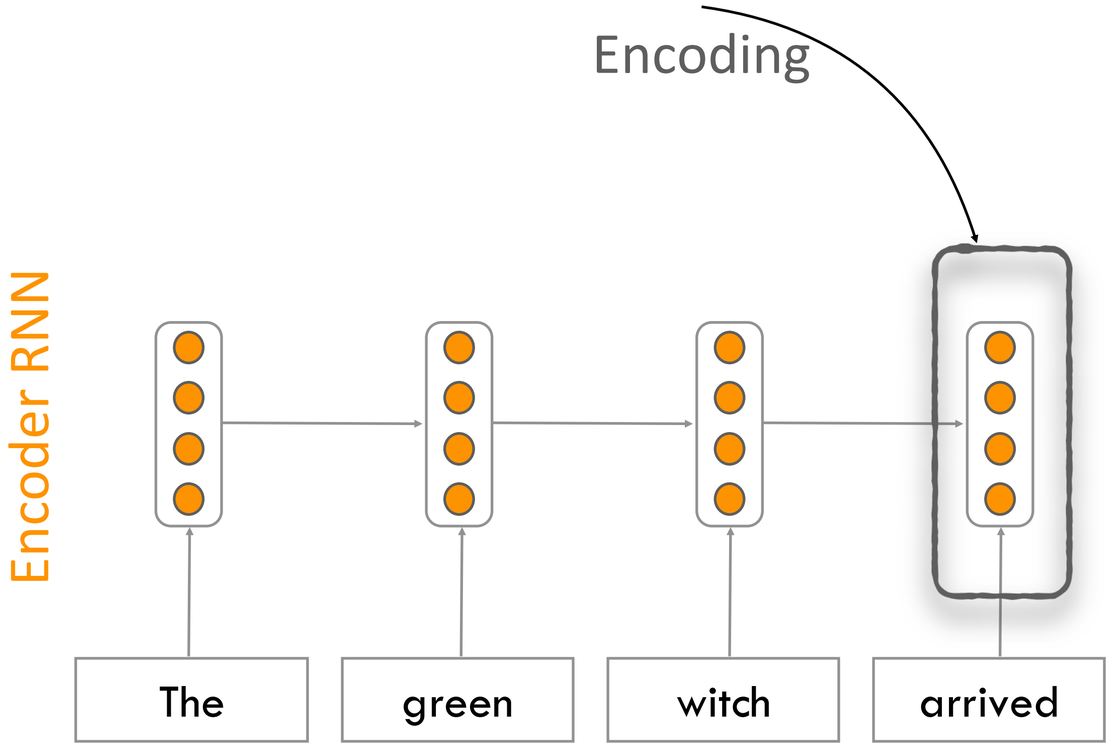
What if we had access to all hidden states?

How to create this?

Attention Mechanism

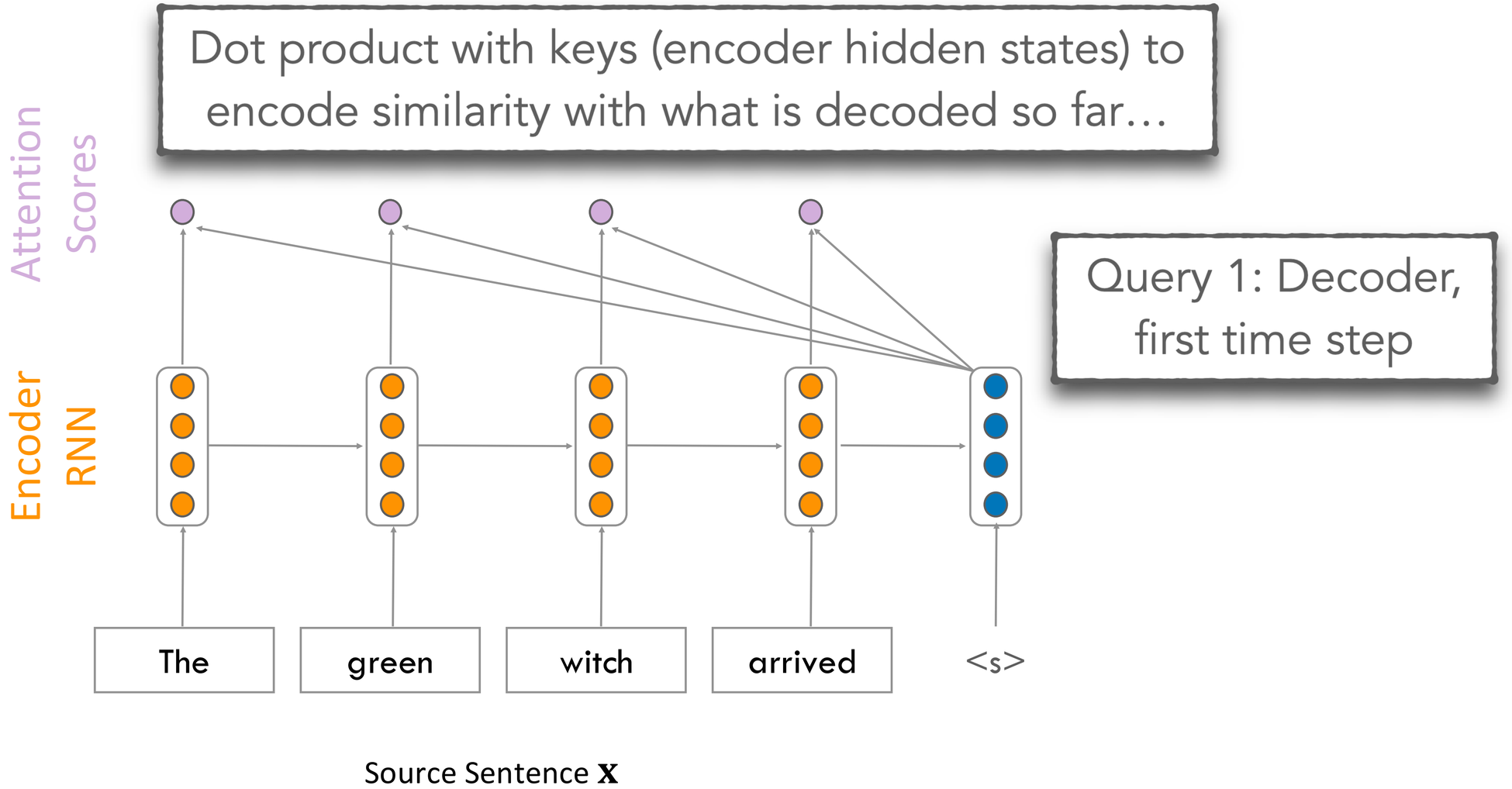
Attention Mechanism

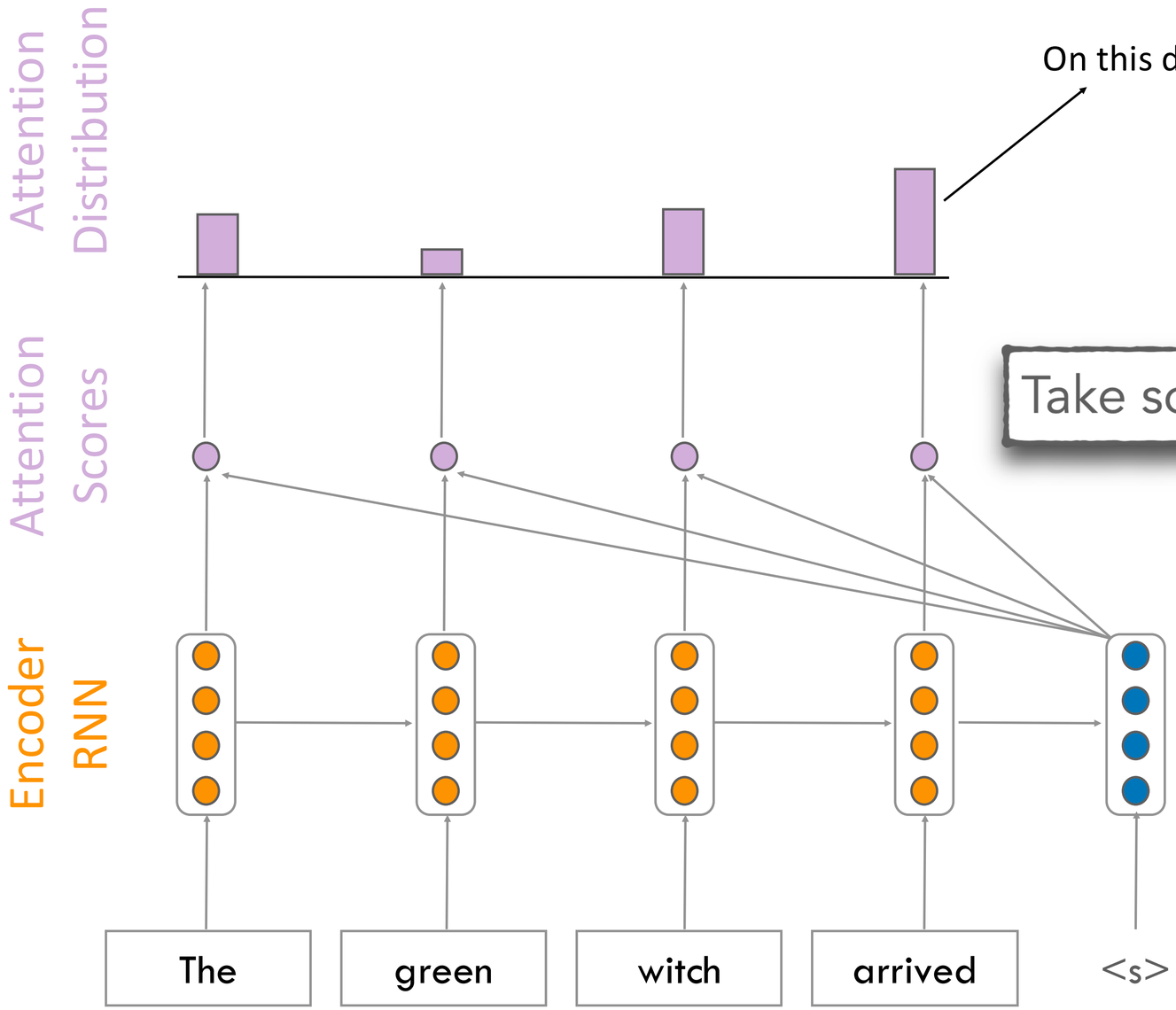
- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Single fixed-length vector $\mathbf{c}_t$ by taking a weighted sum of all the encoder hidden states
  - One per time step of the decoder!
- In general, we have a single query vector and multiple key vectors.
- We want to score each query-key pair



Bahdanau et al., 2015

Seq2Seq with Attention

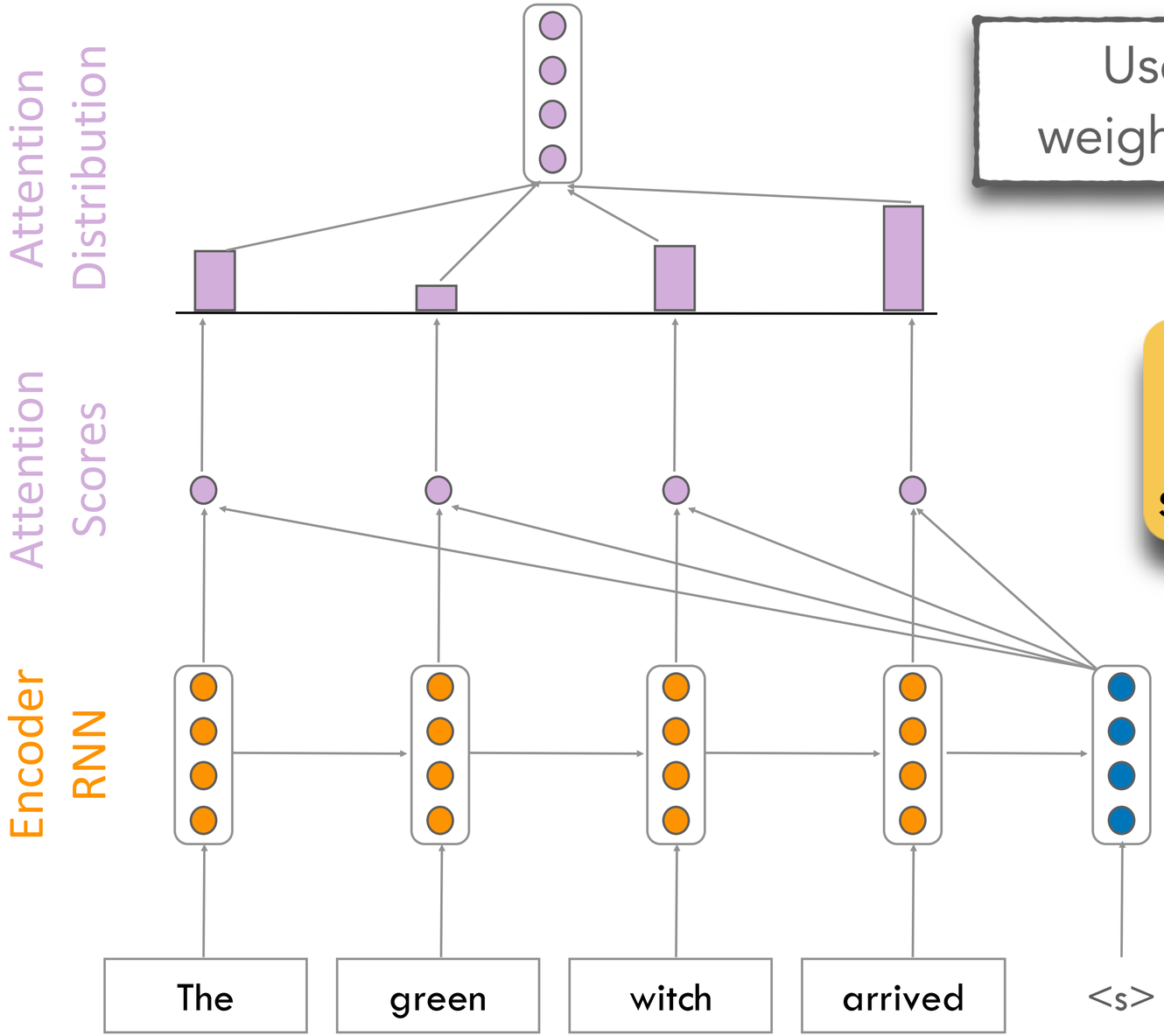




On this decoder tilmestep we are mostly focusing on the source token “arrived”

Take softmax to turn into probability distribution

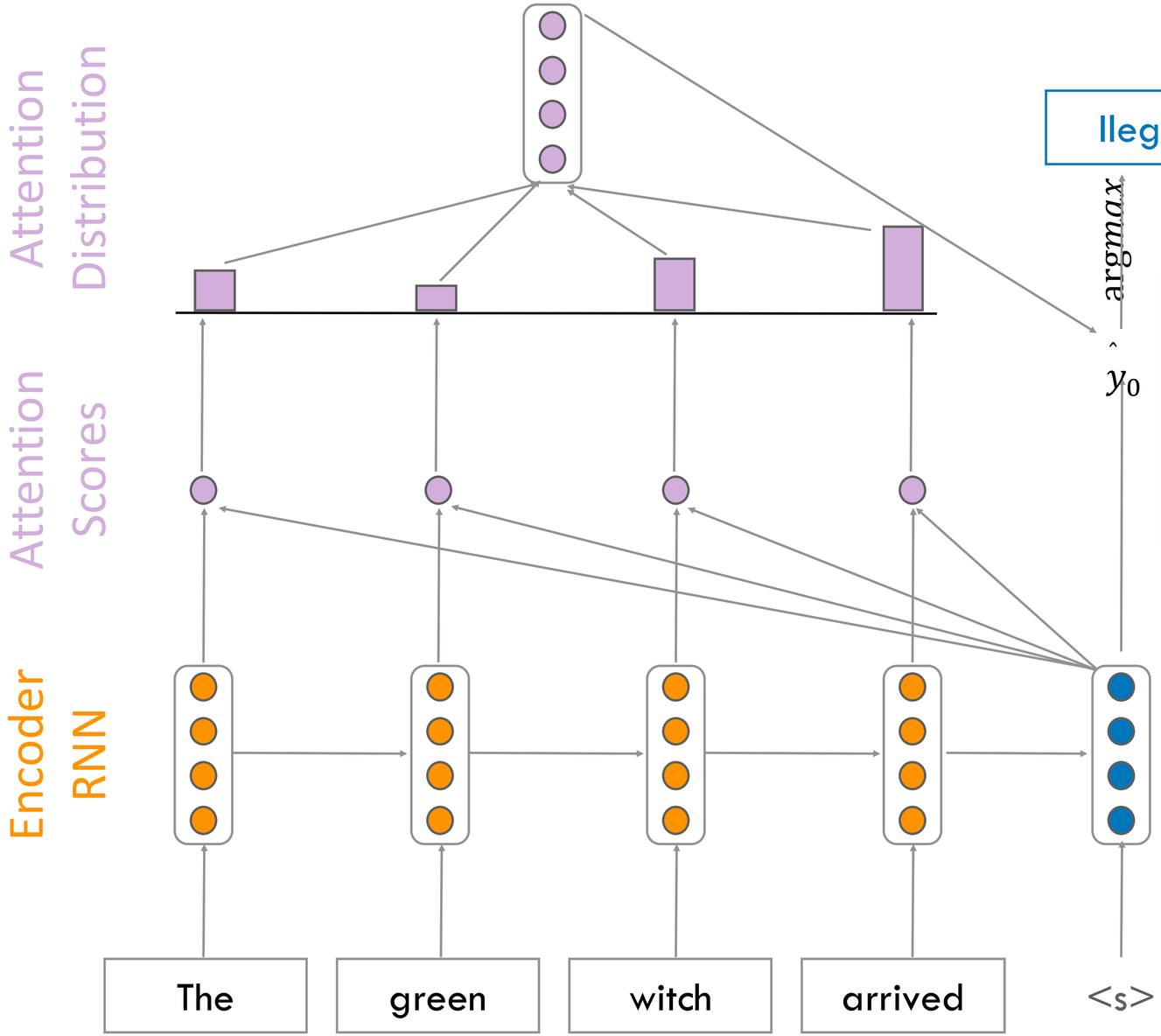
Source Sentence $\mathbf{x}$



Use the attention distribution to take a weighted sum of the encoder hidden states.

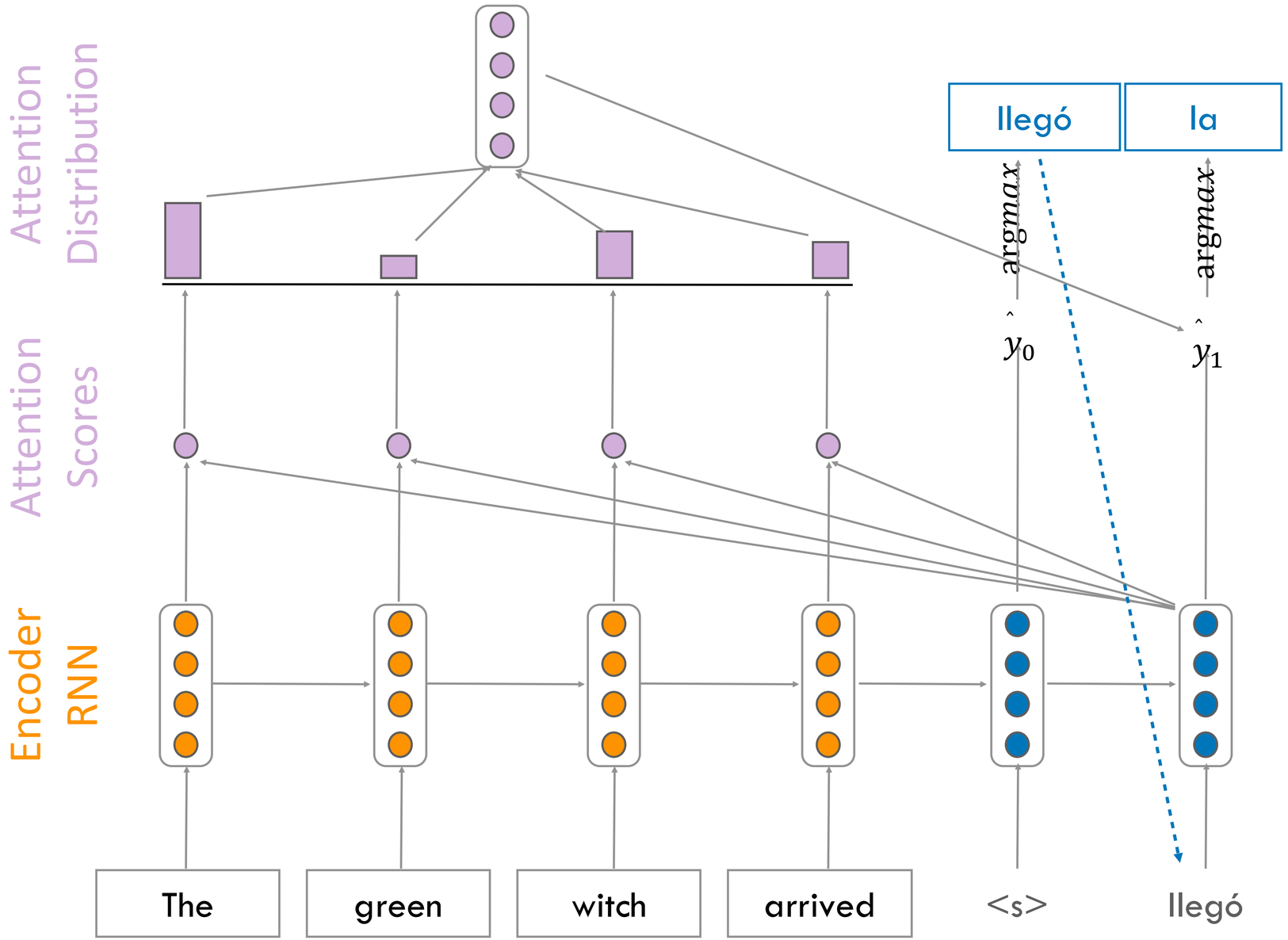
The attention output mostly contains information the hidden states that received high attention.

Source Sentence $\mathbf{x}$



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

Source Sentence $\mathbf{x}$

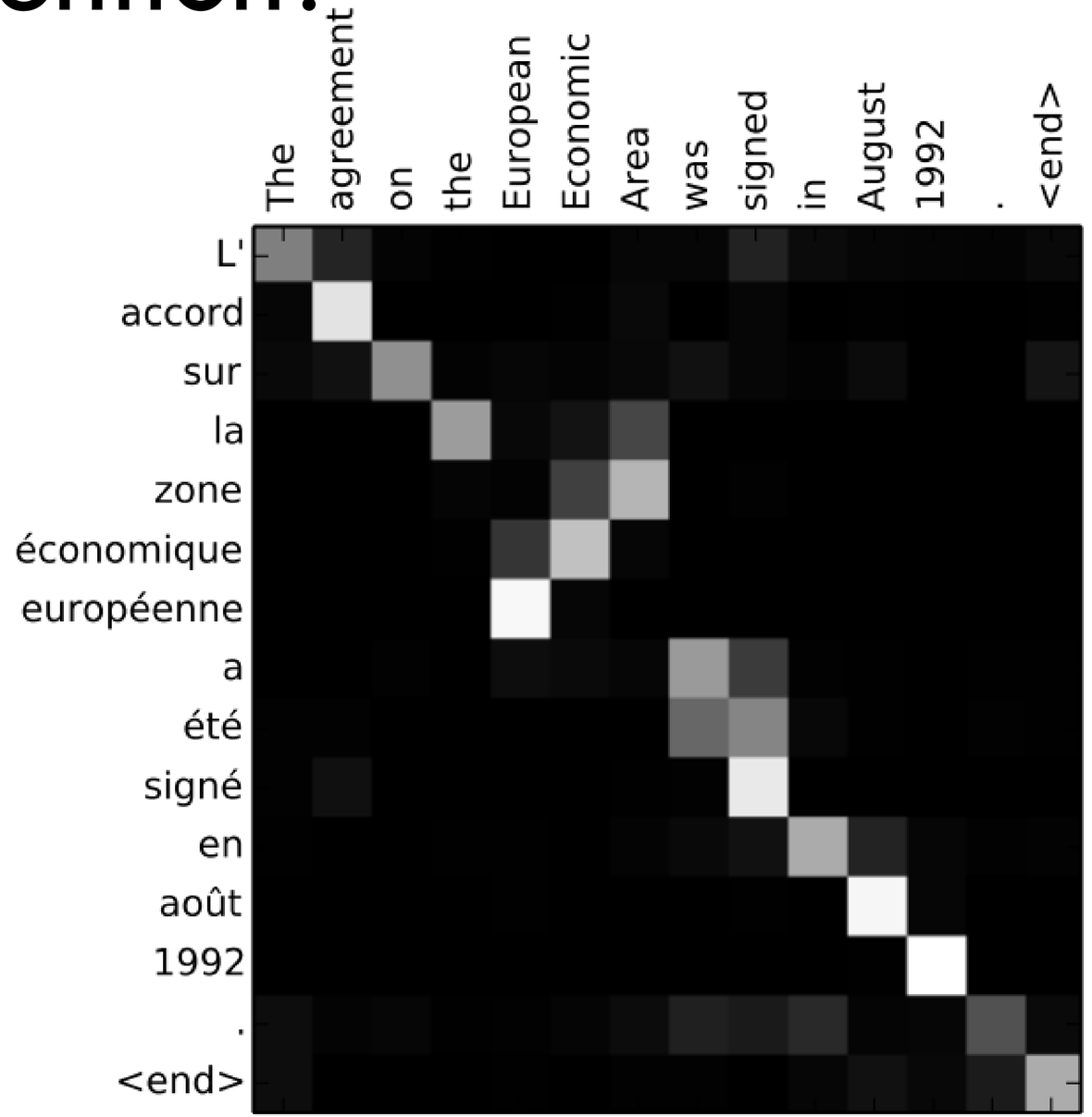


Query 2: Decoder, second time step

Source Sentence **x**

Why Attention?

- Attention significantly improves neural machine translation performance
  - Very useful to allow decoder to focus on certain parts of the source
- Attention solves the information bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
  - Attention provides some interpretability
    - By inspecting attention distribution, we can see what the decoder was focusing on →
    - We get alignment for free! We never explicitly trained an alignment system! The network just learned alignment by itself



Announcements + Logistics

- HW2 is out!
- Project proposal is due by Feb 11, 11:59 PM PT