Lecture 4: RNNs + Attention

Cornell Bowers C·IS
Computer Science

Tanya Goyal

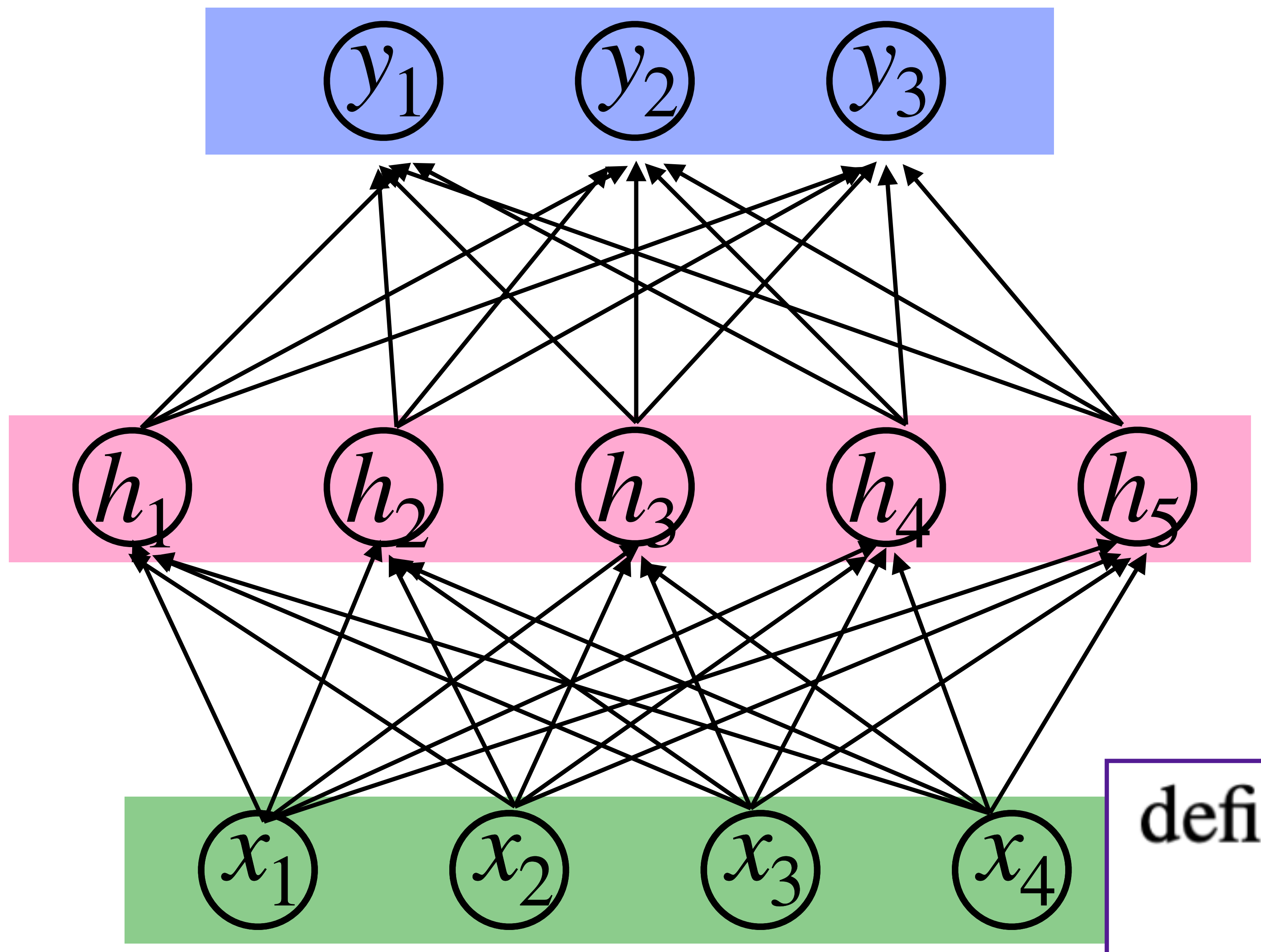
CS 4740 (and crosslists): Introduction to Natural Language Processing

Upcoming deadlines + Today

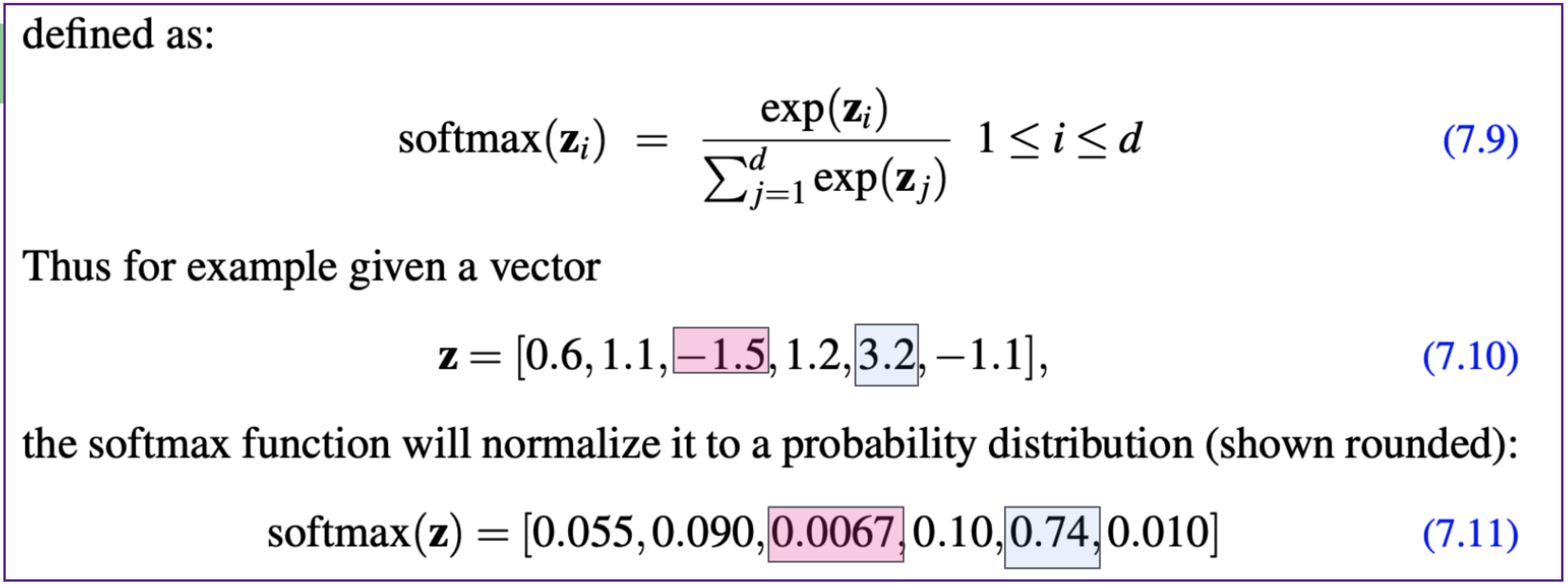
- **Deadlines**
  - Milestone submission due March 11, 11.59 p.m.
    - No slip days allowed for the milestone.
  - Final submission due March 20, 11.59 p.m.
    - At max 2 slip days can be used.

- **Today**
  - Recurrent Neural Networks
  - Attention

Recap:FFNNs



- Input layer $x = [x_1 x_2 \ldots x_4]^T$
- Hidden Layer:
  - $h = f(Wx) \quad \in R^5$
  - $W \in \mathbb{R}^{5 \times 4}$



defined as:

$$\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{d} \exp(\mathbf{z}_j)} \quad 1 \leq i \leq d \tag{7.9}$$

Thus for example given a vector

$$\mathbf{z} = [0.6, 1.1, \boxed{-1.5}, 1.2, \boxed{3.2}, -1.1], \tag{7.10}$$

the softmax function will normalize it to a probability distribution (shown rounded):

$$\text{softmax}(\mathbf{z}) = [0.055, 0.090, \boxed{0.0067}, 0.10, \boxed{0.74}, 0.010] \tag{7.11}$$

FFNNs

Usually, input is a **batch.**

Batch size: b, i.e. b datapoints.

Input = $x = \begin{bmatrix} x_1^1 \; x_2^1 \; x_3^1 \cdots x_n^1 \\ x_1^2 \; x_2^2 \; x_3^2 \cdots x_n^2 \\ \cdot\,\cdot \\ x_1^b \; x_2^b \; x_3^b \cdots x_n^b \end{bmatrix}$

$h = f(Wx)$ $\rightarrow$ dimension of h?

$z = Uh$

$y = \text{softmax}(z)$

Training a FFNN

1. Forward Pass: Pass the input through the network. Compute the predicted output probabilities $\hat{y}$

2. Compute Loss $L(y, \hat{y})$

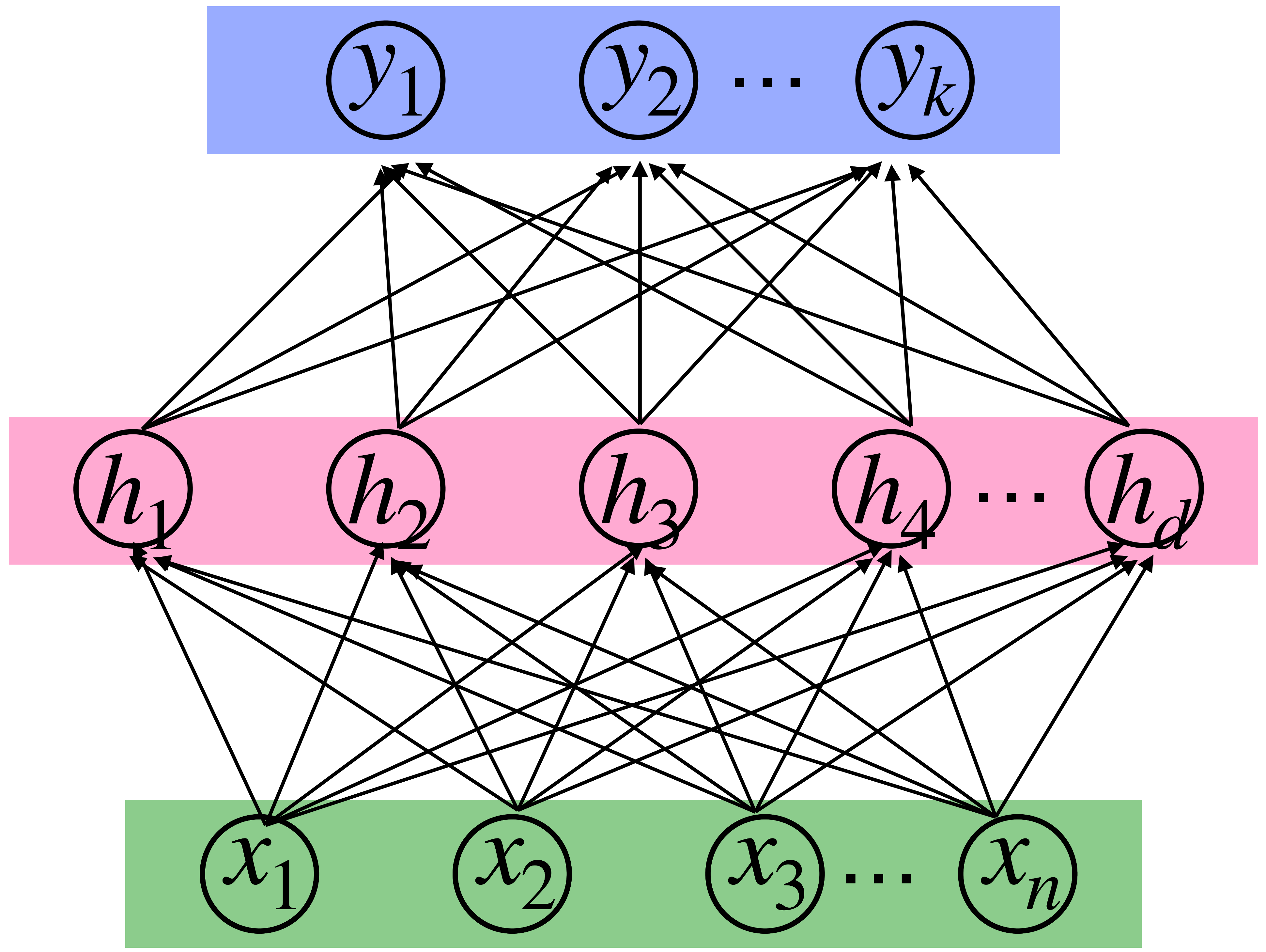
$$L_{\text{CE}} = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$ **(cross entropy loss)**

Note that y_k, i.e. the gold distribution, is 1 for the gold label and 0 for all other k.

Simplifies to $-\log \hat{y}_{\text{gold}}$ **(negative log likelihood)**

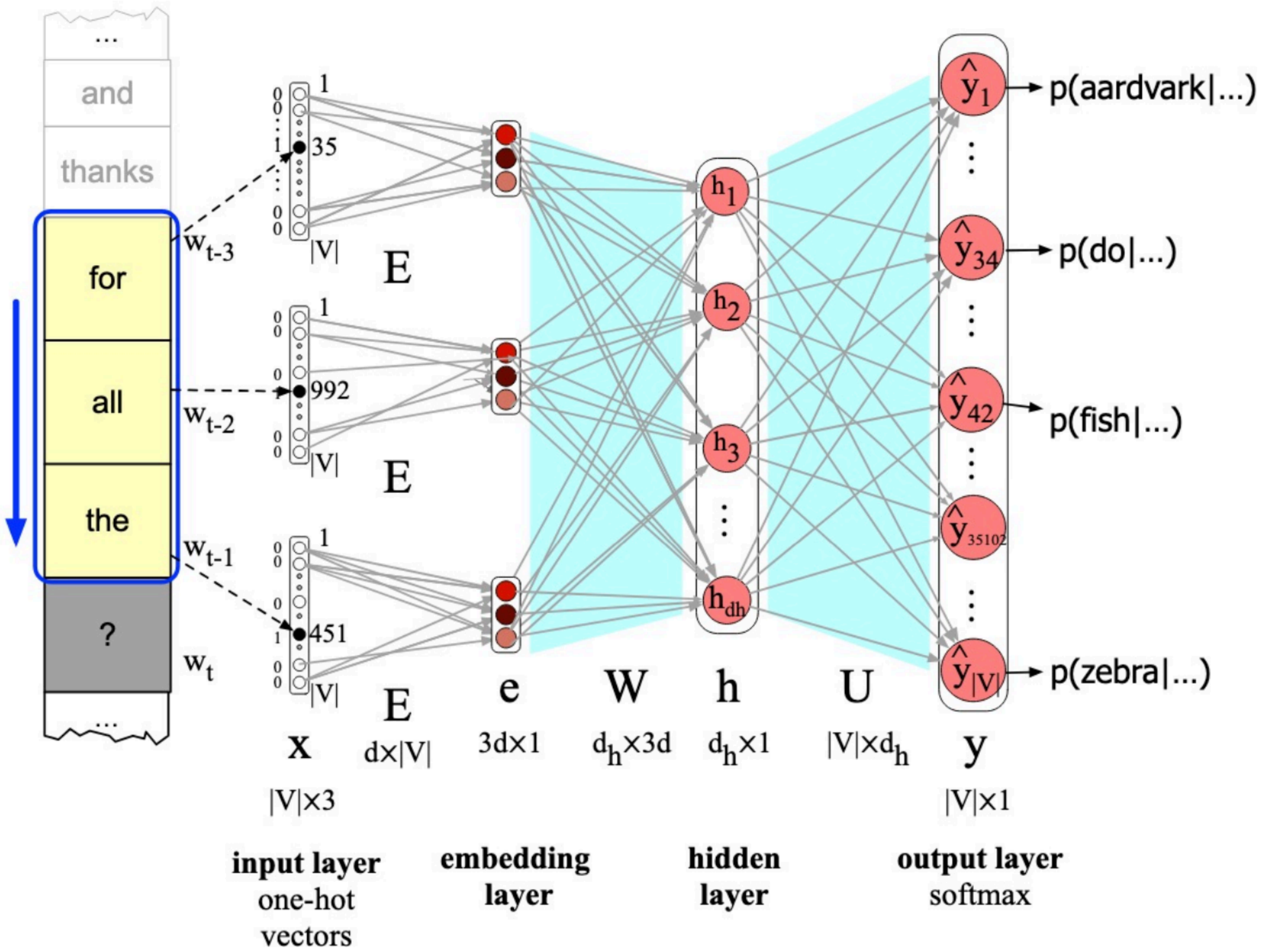
3. Back-propagate and update all parameters.

y_i = Prob. of predicting label i.



Recap: FFNN for language modeling

Language Modeling Task: FFNN should output $P(w_n | w_1 \ldots . w_{n-1})$, i.e. prob. distribution over the vocabulary of predicting a particular word as the next word of a sequence.

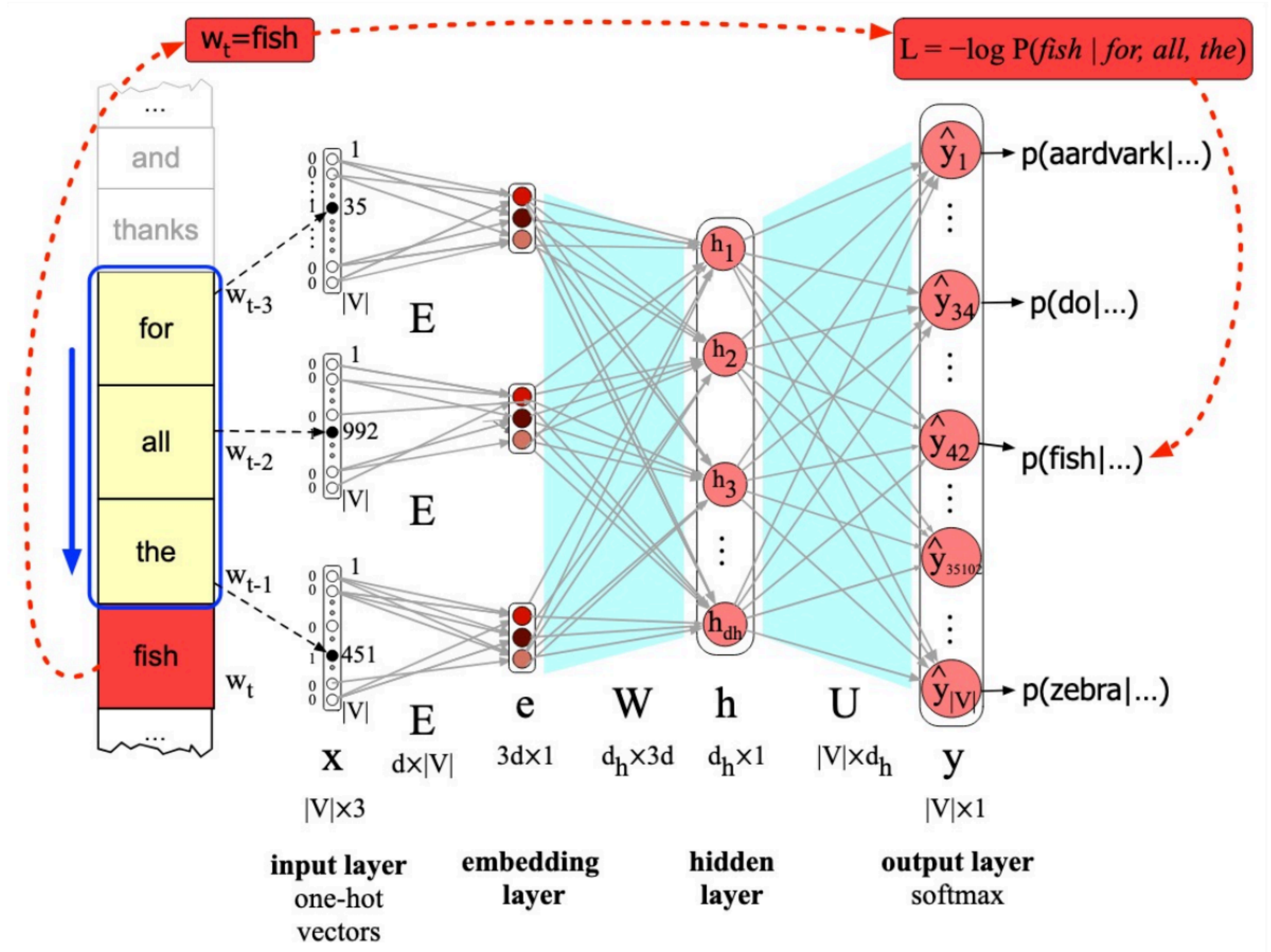


This is a 4-gram language model.

How should we change this architecture to train a language model that avoids making the Markov assumption?

FFNN for language modeling

Language Modeling Task: FFNN should output $P(w_n | w_1 \ldots . w_{n-1})$, i.e. prob. distribution over the vocabulary of predicting a particular word as the next word of a sequence.



We can learn better word embeddings by **back-propagating all the way back to embeddings.**

- Modify W, U and E during updates.

- For some tasks, it's ok to freeze the embedding layer E with initial word2vec/other learned embeds.
  - Hold E constant, only modify W, U.

Count based N-grams v/s FFNN for language modeling

- Advantages of FFNNs over N-grams.
  - Do we need smoothing for FFNNs?
  - Do we need to handle unknown words for FFNNs?

  - Can handle longer histories using a deep averaging network.
  - Can generalize over contexts of similar words.
  - Higher predictive accuracy

- Disadvantage of FFNNs over N-grams?
  - Slower to train

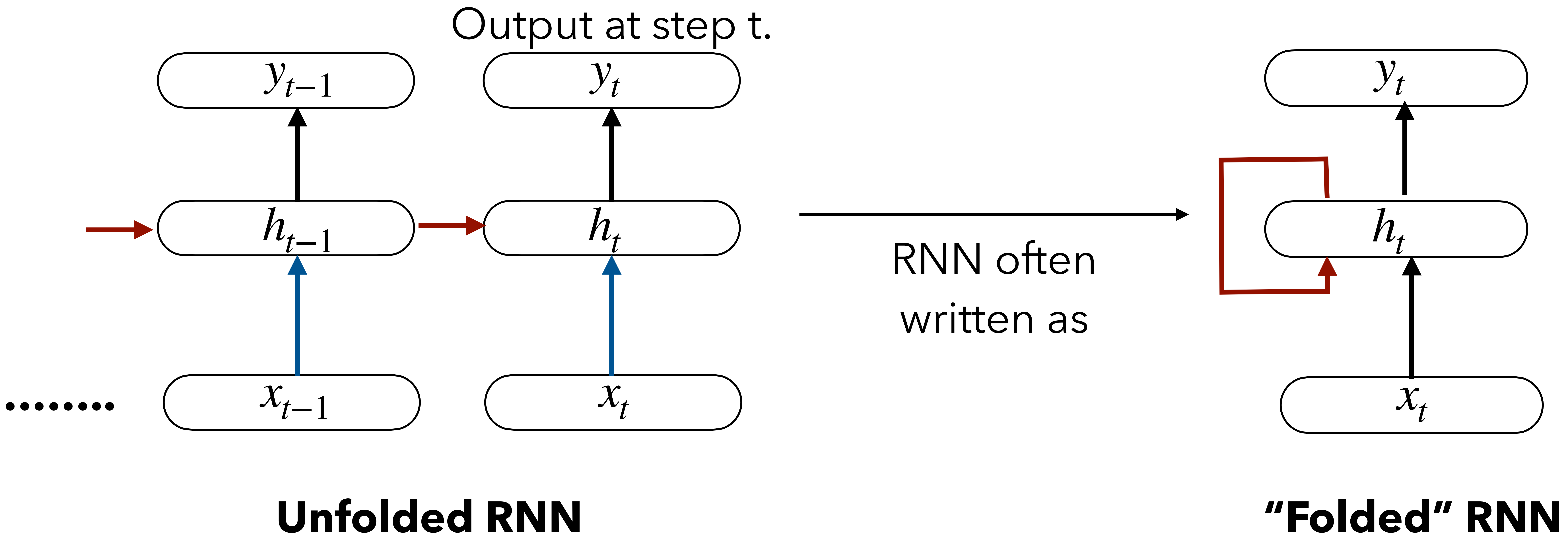
Limitations of FFNN-based Language Models

- We have discussed two options for FFNNs –
  1. Train a N-gram language model. Loses information from distant contexts, i.e. more than (N-1) words away.
  2. Train a deep averaging network. Loses order information.

- Neither is ideal.
- Can fix these problems with **Recurrent Neural Networks** (RNNs)

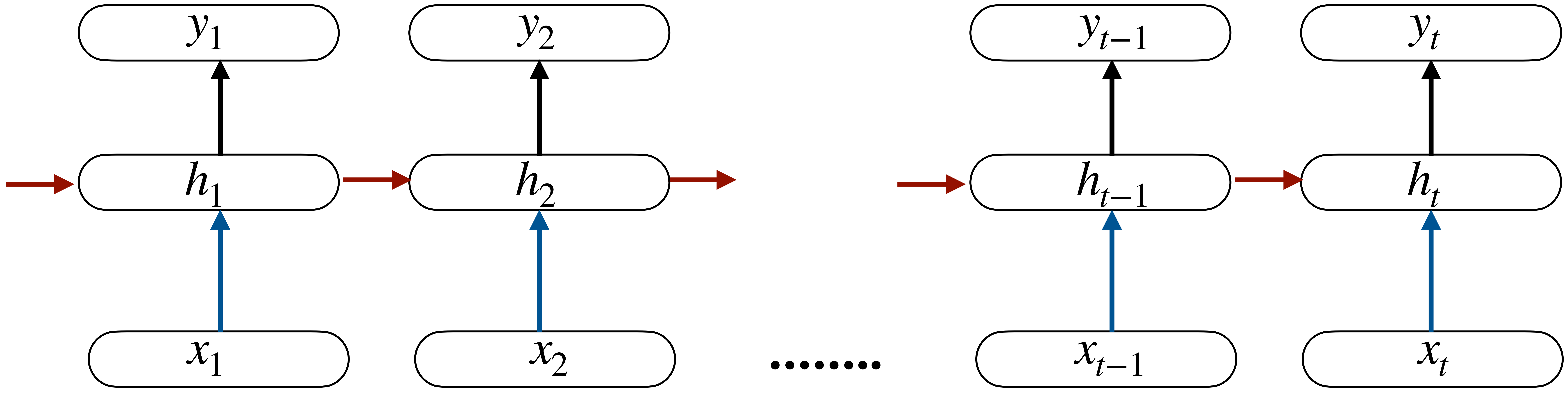
RNNs: High-level Idea

- "Recurrent": A network that contains a cycle.
- Hidden layer activation depends on
  - Input/last layer **and** activation of the hidden layer at the previous time step.



Simple RNNs

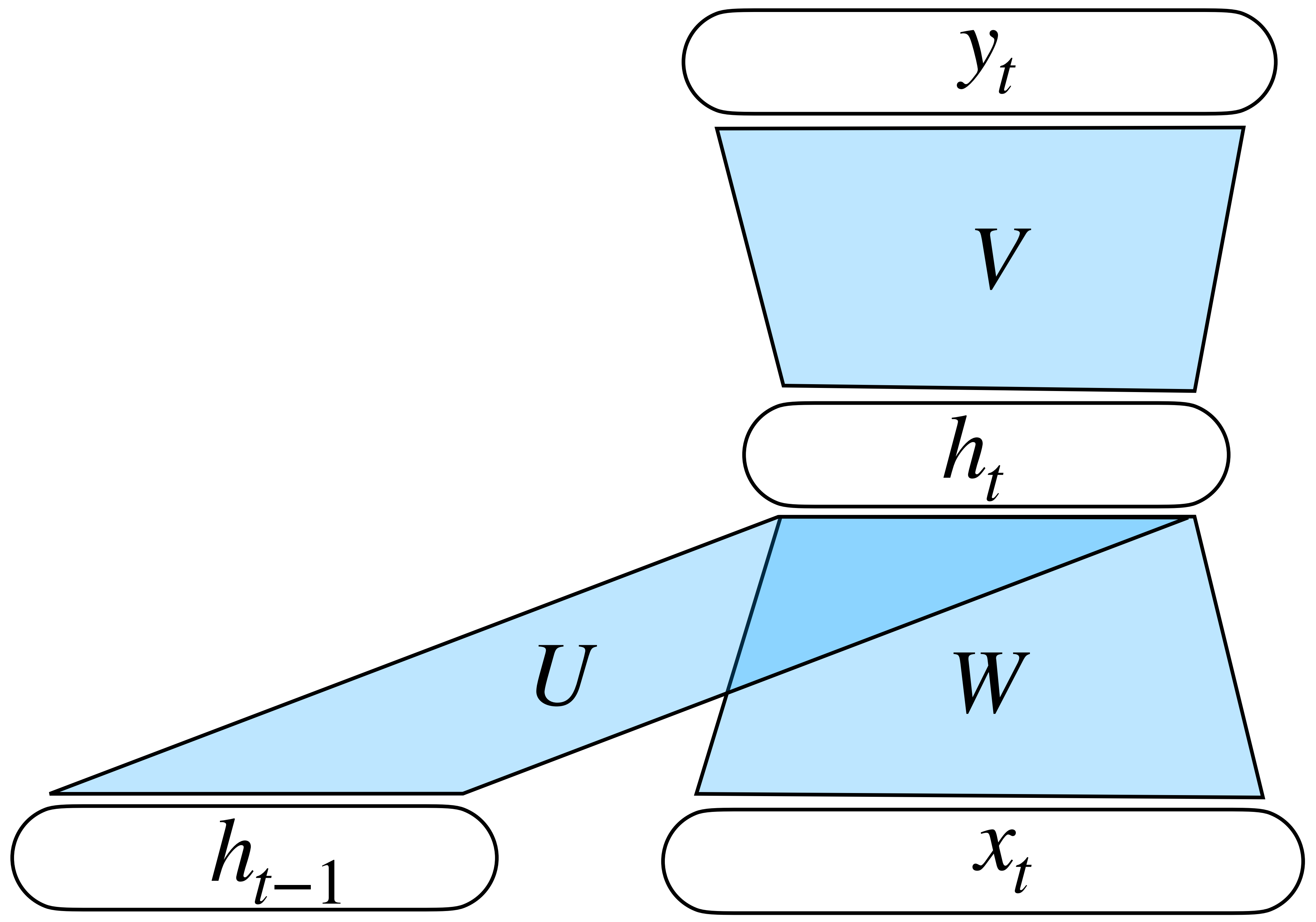
- Hidden layer from the previous time step acts as a “memory”.
- Stores information about entire previous context.
- Includes information extending back to the beginning of the sequence.



RNNs: The computation structure

- Similar to feedforward computations.
- U determines how the past context is used.

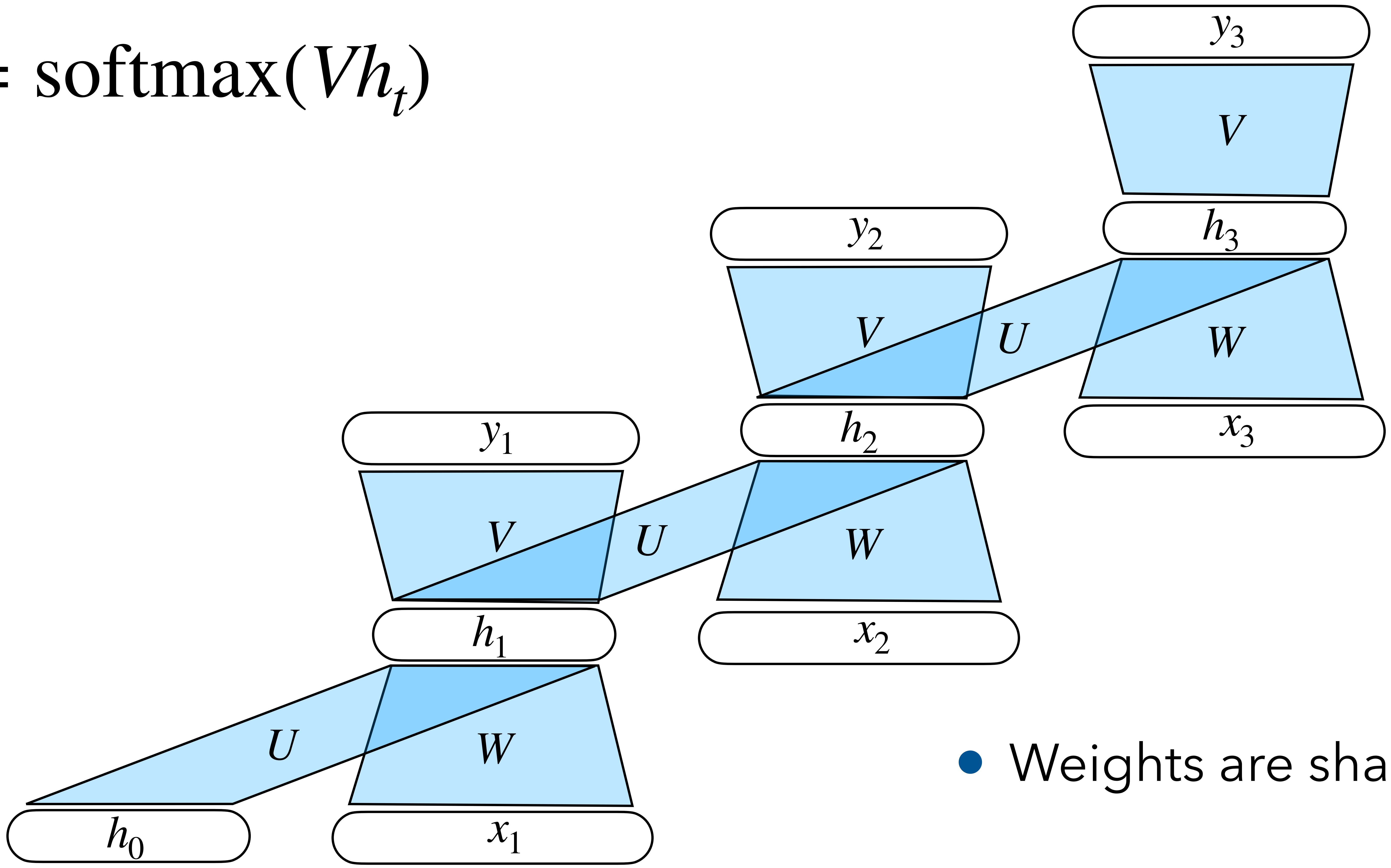
$$h_t = f(Uh_{t-1} + Wx_t)$$
$$y_t = \text{softmax}(Vh_t)$$



Unrolling the RNN in time

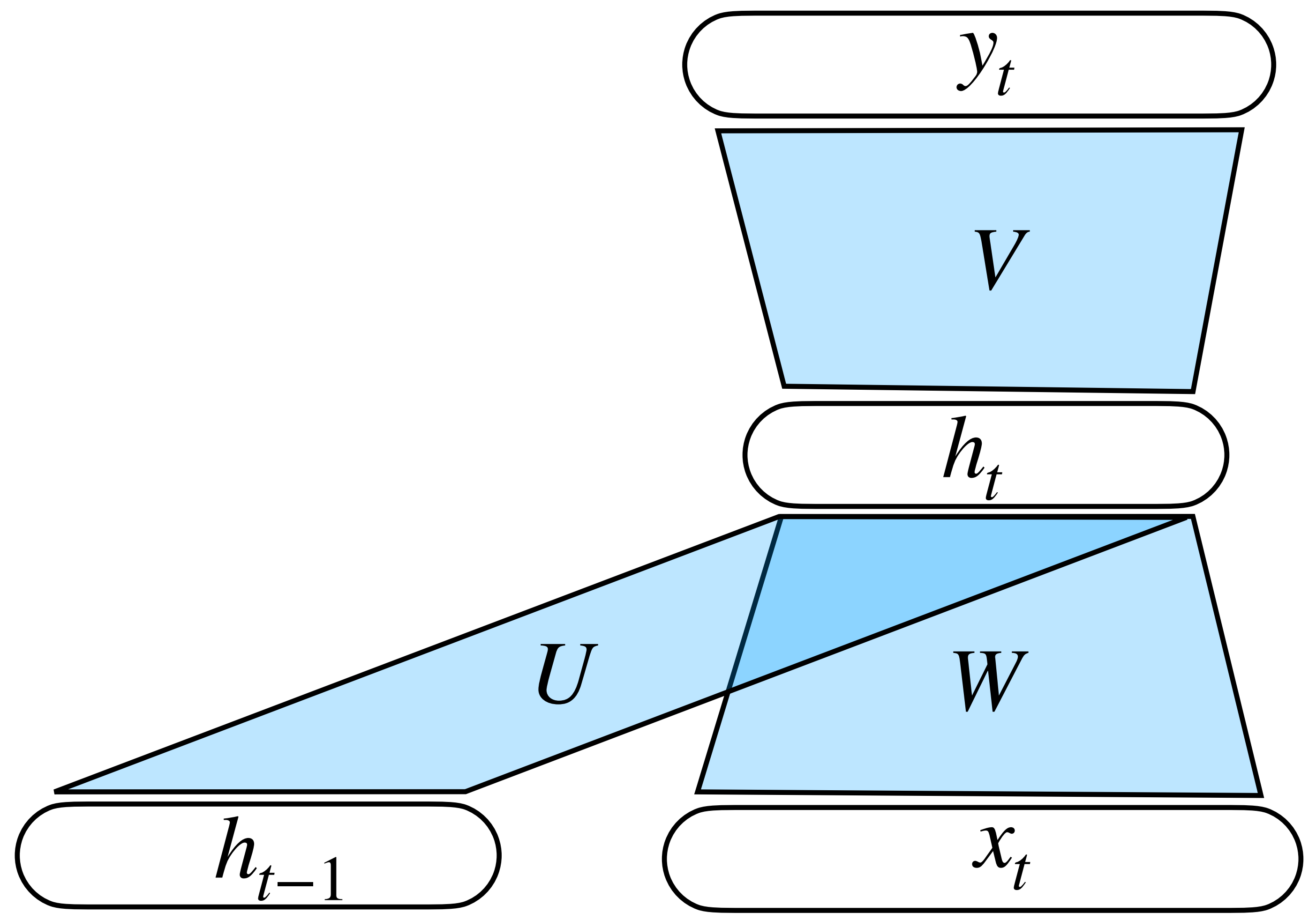
$$h_t = f(Uh_{t-1} + Wx_t)$$

$$y_t = \text{softmax}(Vh_t)$$



- Weights are shared across the time steps

More Quiz



Q1. What are the dimensions of U, W and V for binary classification?

(Assume x is [d1 x 1] and h is [d2 x 1])

Q2. The entire history has an influence on h_t. True or False?

A2. In theory, yes.

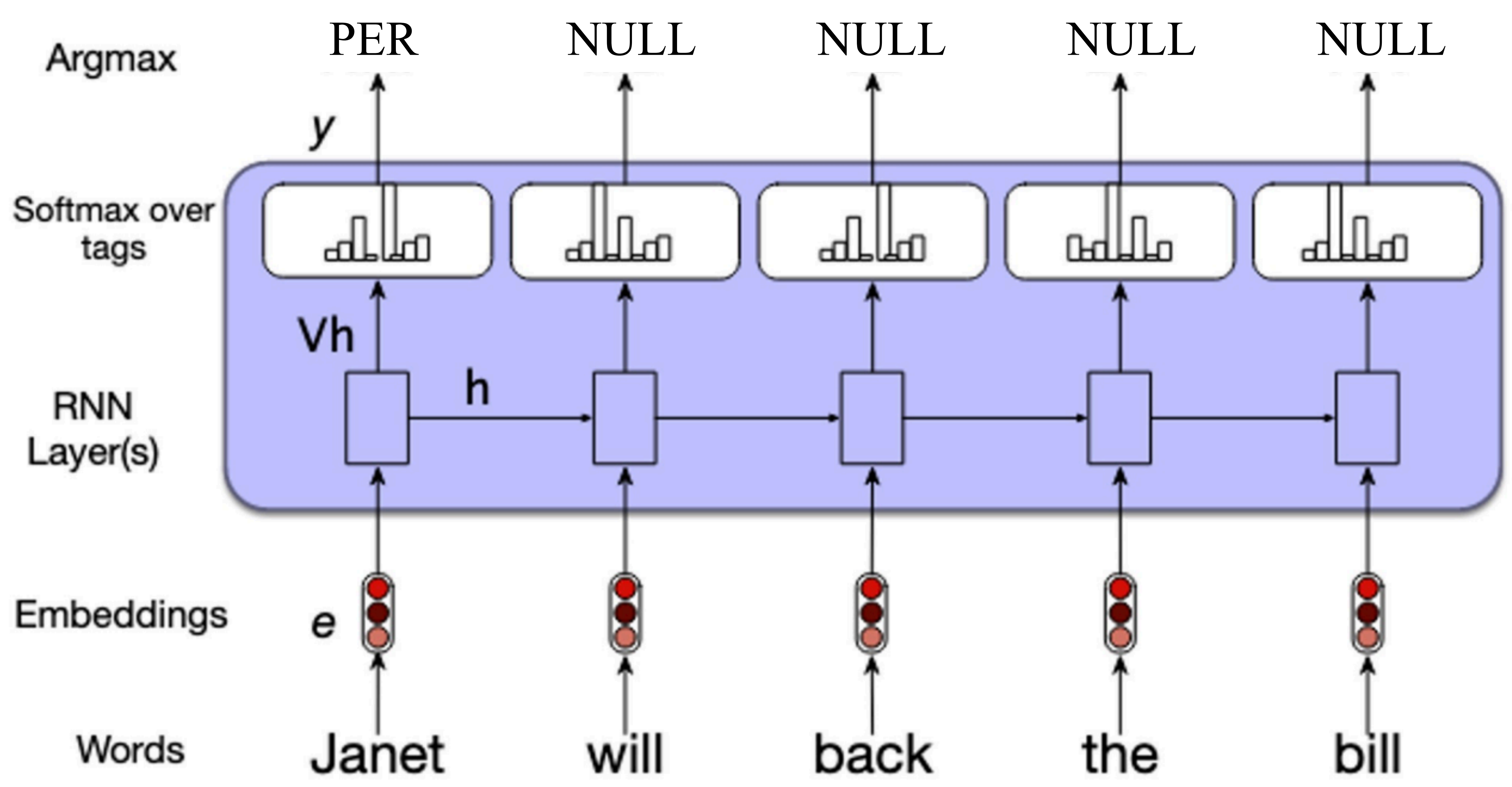
$$\begin{aligned} h_t &= f(Uh_{t-1} + \cdots) \\ &= f(U(f(Uh_{t-2} + \cdots) + \cdots)) \\ &= f(Uf(U(f(Uh_{t-3} + \cdots) + \cdots))) + \cdots) \\ &= f(Uf(U(f(\cdots \quad Uf(Uh_0 \ + \cdots) + \cdots))) + \cdots) \end{aligned}$$

RNNs algorithm

- Input: sequence of words $x = x_1 x_2 \cdots x_N$
- Output: sequence of labels $y = y_1 y_2 \cdots y_N$

```
**function** *ForwardPassRNN*(x, network)
    $h_0 \leftarrow 0$
    **for** $t \leftarrow 0$ to $N$ **do**
        $h_t = f(Uh_{t-1} + Wx_t)$
        $\hat{y}_t = \text{softmax}(Vh_t)$
    **Return** $\hat{y}$
```

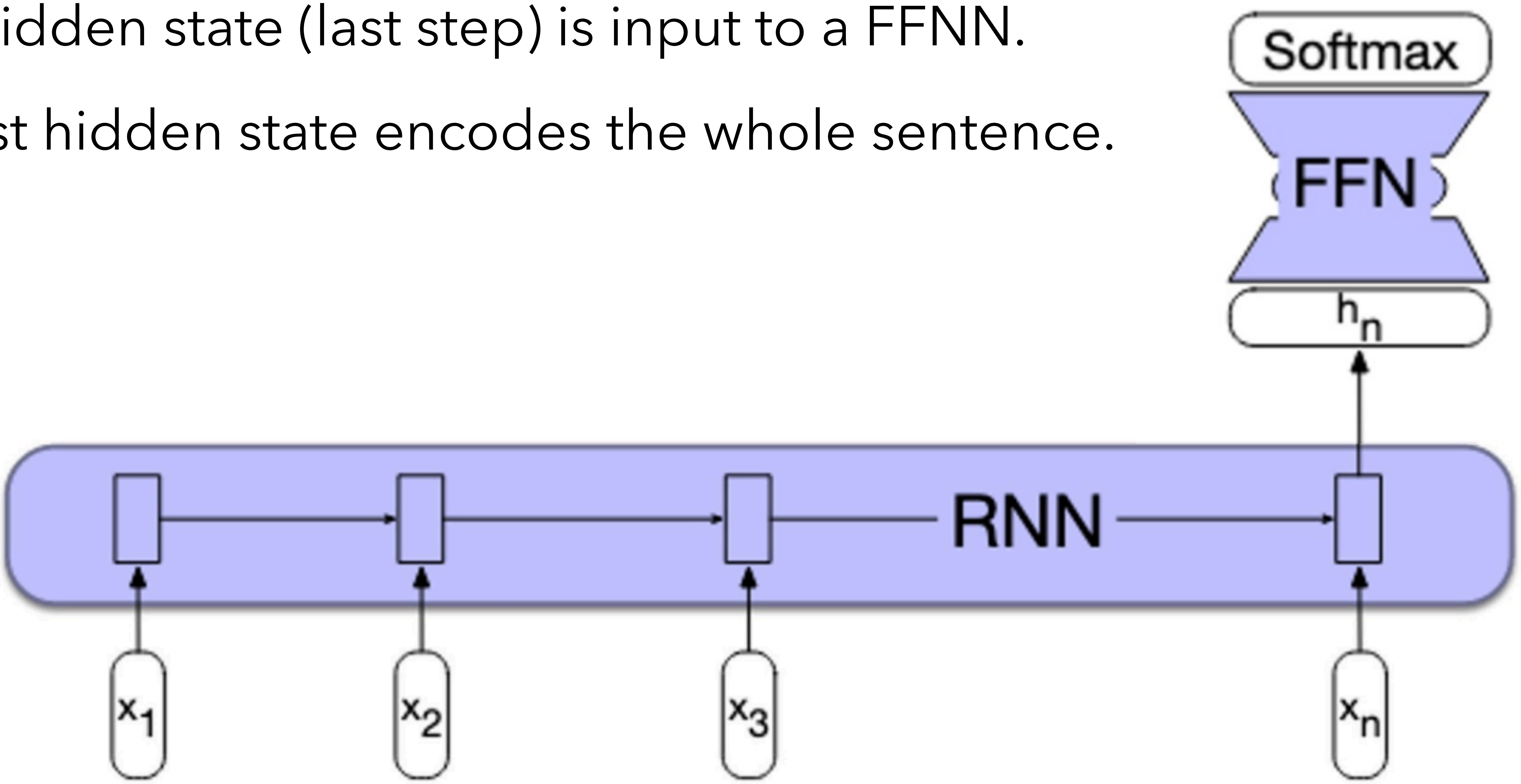
Q. What is y for the named entity recognition (NER) task?



RNNs for Sentence Classification

Q. Design an RNN architecture for the sentence sentiment classification task?

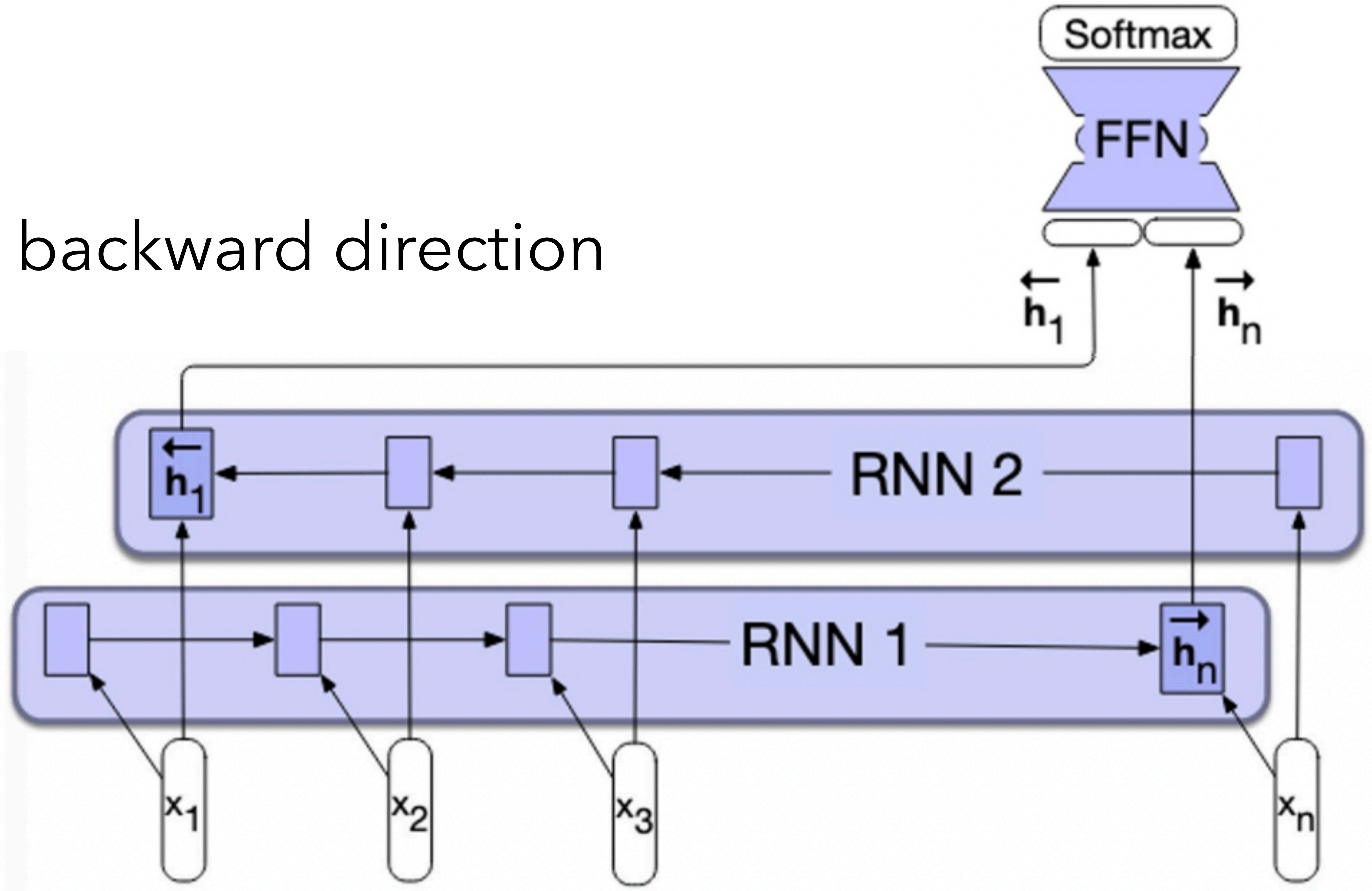
- No intermediate output tokens, only one output at the last step.
- Final hidden state (last step) is input to a FFNN.
- The last hidden state encodes the whole sentence.



Bidirectional RNN

- For token-level classification, the hidden state is a function of only the left context. But it might be useful to encode the right context as well.
- E.g. "London lives in Ithaca" → It is only clear that London has label PER based on the right context.
- Solution: **bidirectional RNNs**
  - Train two RNNs, one each in forward ing backward direction
  - Normally use concatenation.
  - Can be used for both seq. classification

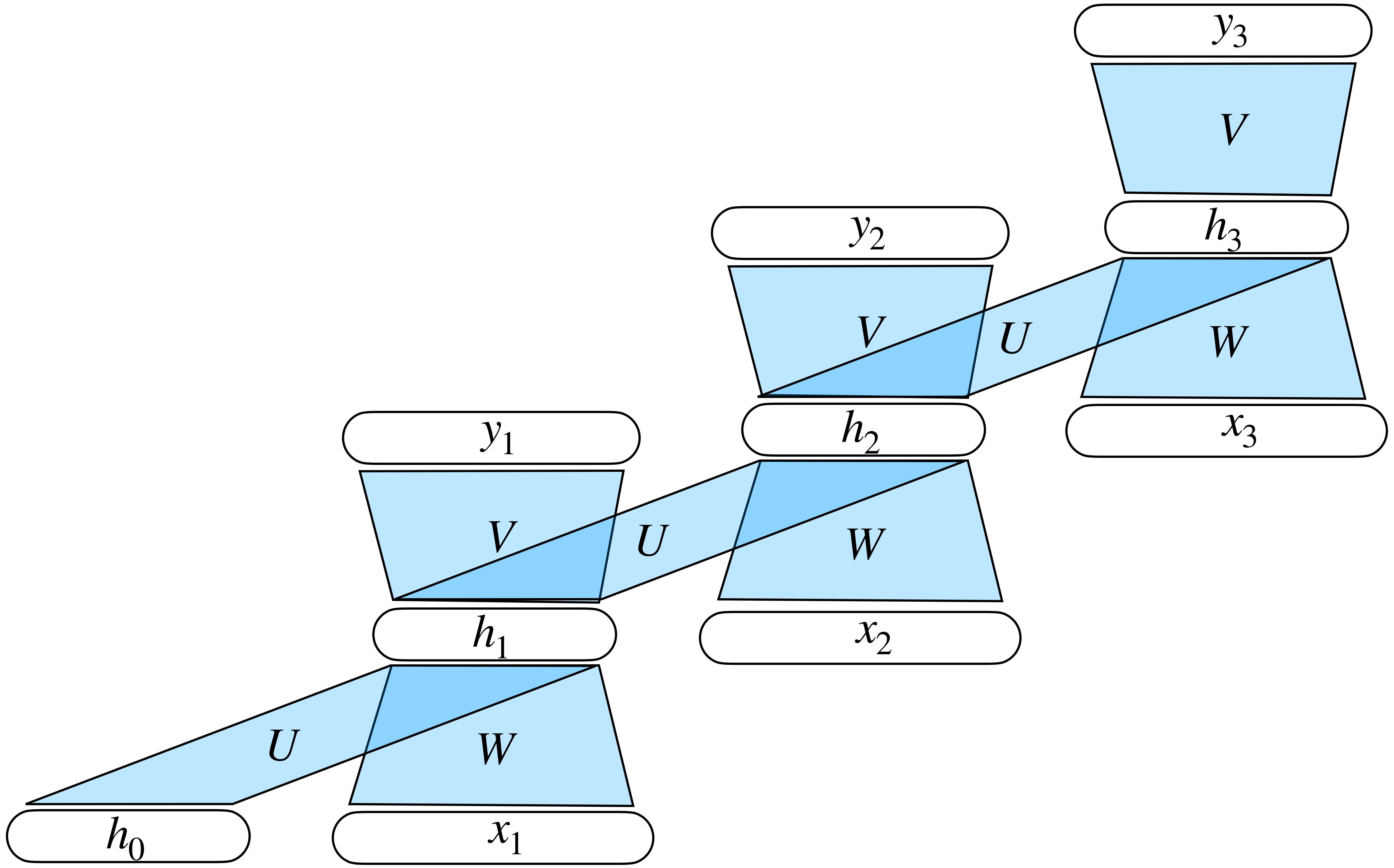
  (See figure) or token-level classification.



Training an RNN

As with FFNNs, three steps.

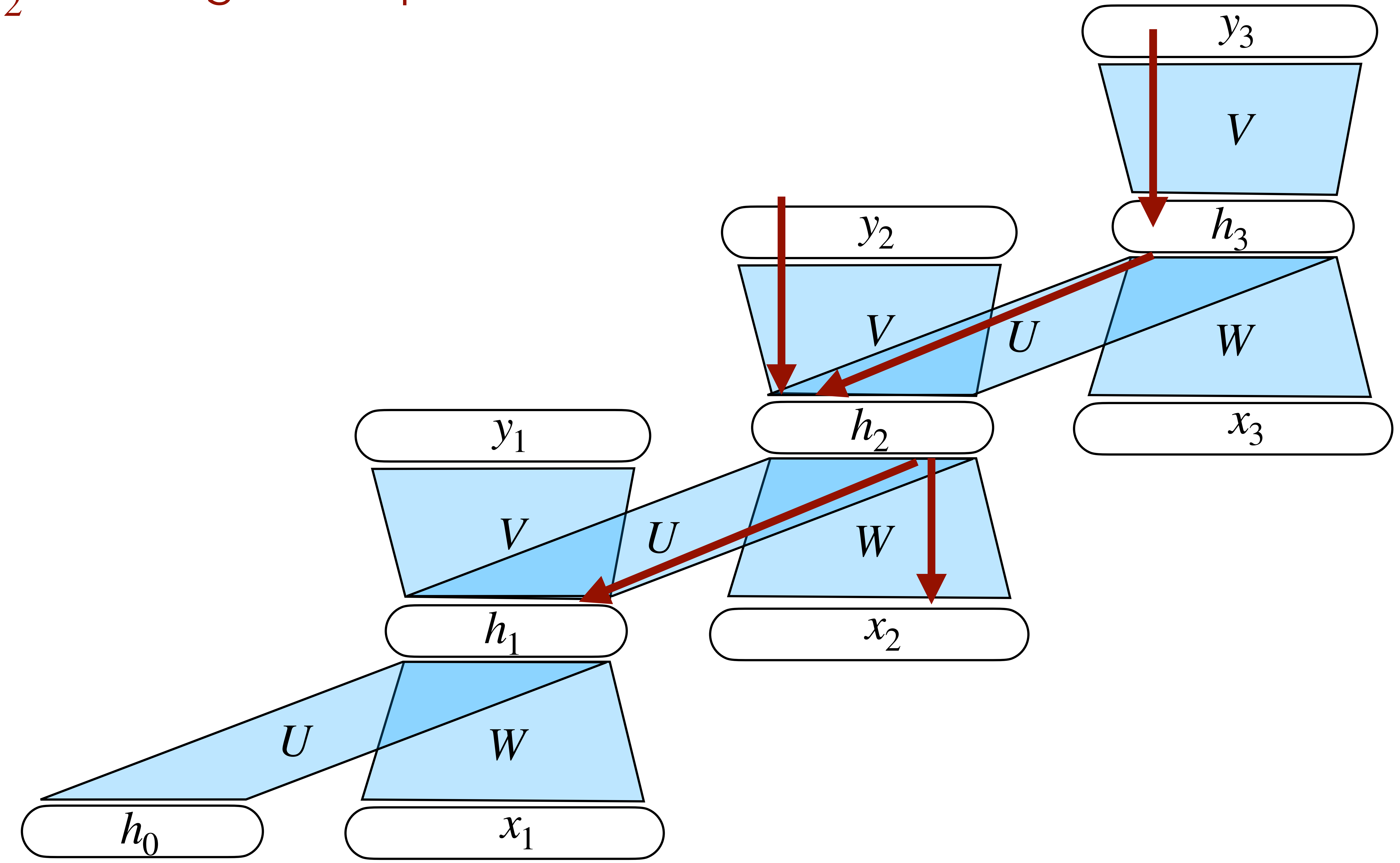
1. Forward pass to get predicted outputs $\hat{y}_1, \hat{y}_2, \hat{y}_3$
2. Compute loss L (cross entropy) **for each time step.**
3. Back-propagate the loss and update parameters: U, W, V and (maybe) E.



Training an RNN

Consider time step 2: Let y_2^* be the gold output.
We compute $L(\hat{y}_2, y_2^*)$.

Flow of back propagated errors needed for updating U, V, W at <u>time step 2.</u>



RNNs for Langauge Modeling

Q. Design an RNN architecture for the language modeling task?

Qa. What is the **input sequence** x and **output sequence** y for the task?

Autoregressive generation:
One word is generated at each time step, conditioning on the word generated in the previous time step.

RNNs for Langauge Modeling

Q. Design an RNN architecture for the language modeling task?

Qa. What is the **input sequence** x and **output sequence** y for the task?

Training an RNN:

Use **teacher forcing**, i.e. use the predicted word at each time step for backprop, but supply the gold sequence as input.

Break

Encoder-Decoder RNNs

Can we think of generation tasks where we do not have an output label y_i corresponding to each input token x_i?

Answers: Summarization, Neural Machine Translation, etc.

Suppose we are translating from English to French. We need an RNN model that takes in the **entire** input, and then generates the output conditioned on the input.

An encoder-decoder architecture is appropriate for this.

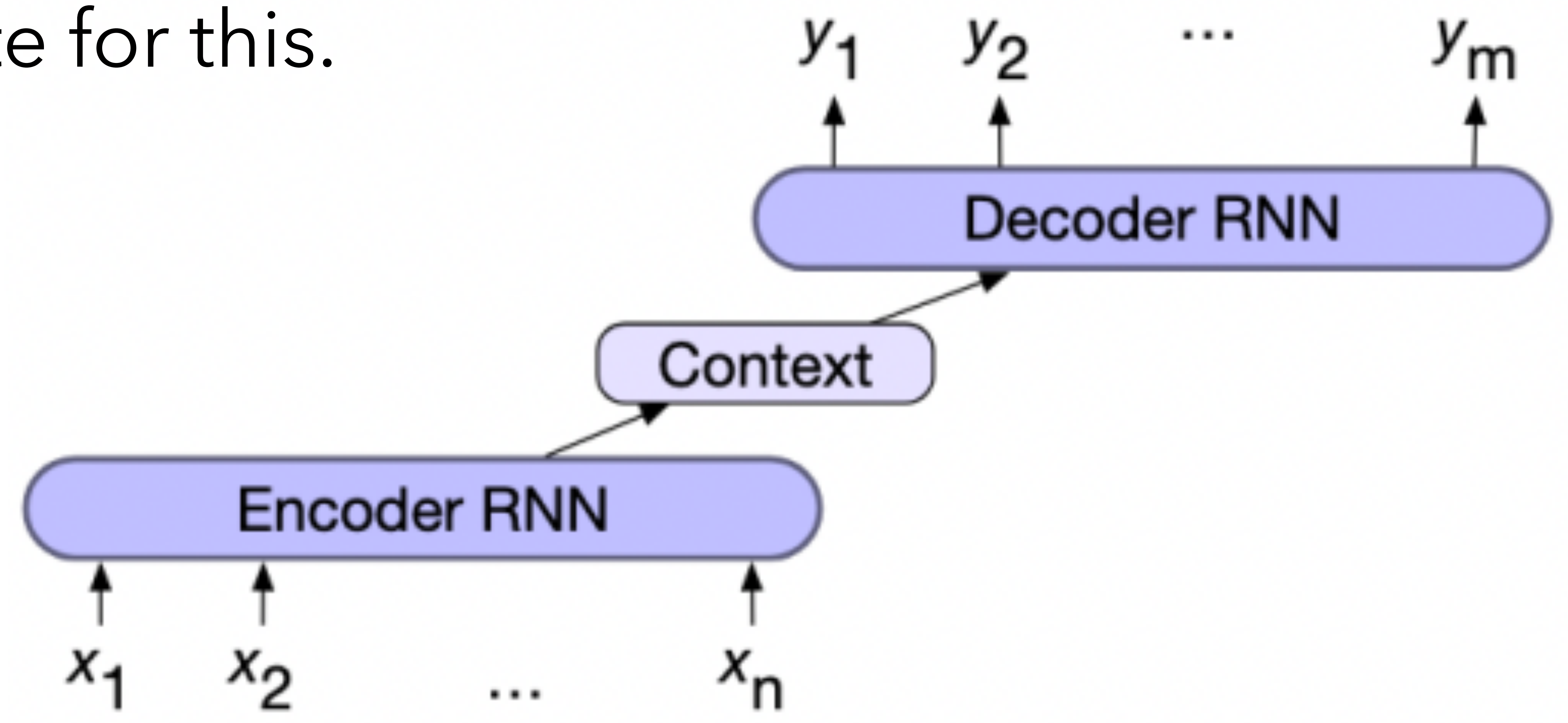
Encoder-Decoder RNNs

Can we think of generation tasks where we do not have an output label y_i corresponding to each input token x_i? Summarization, Neural Machine Translation, etc.

Suppose we are translating from English to French. We need an RNN model that takes in the **entire** input, and then generates the output conditioned on the input.

An **encoder-decoder** architecture is appropriate for this.

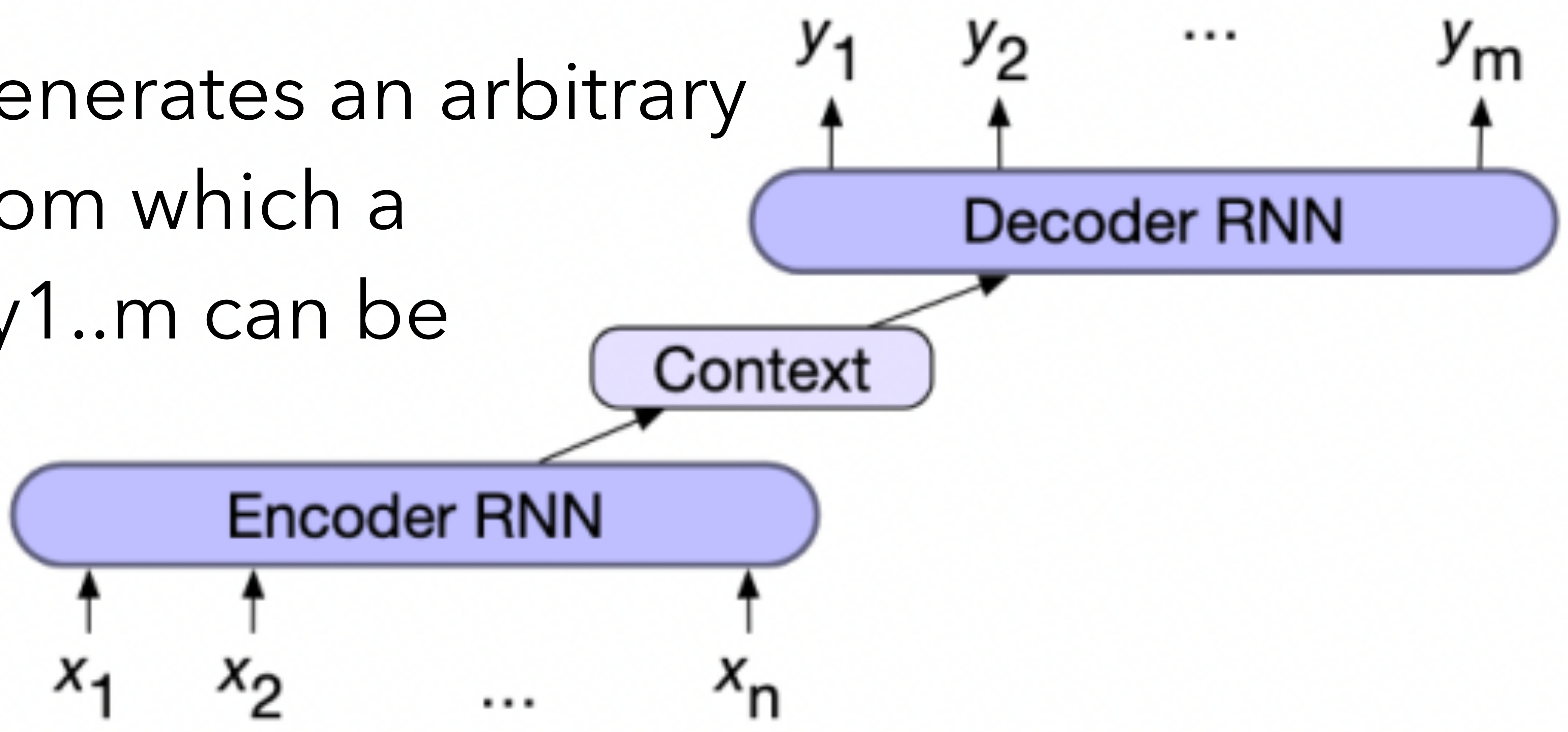
- **Encoder**: an RNN that maps the input sequence to a "encoding" vector representing the context/input.
- **Decoder**: an RNN that maps from the **context** to the output sequence **y**.
- Commonly knows as seq2seq models.



Encoder-Decoder RNNs

An **encoder-decoder** has 3 components:

- An encoder that accepts an input sequence, x1..n, and generates a corresponding sequence of contextualized representations, h1..n
- A context vector, c, which is a function of h1..n, and conveys the essence of the input to the decoder
- A decoder, which accepts c as input and generates an arbitrary length sequence of hidden states h1..m, from which a corresponding sequence of output states y1..m can be obtained

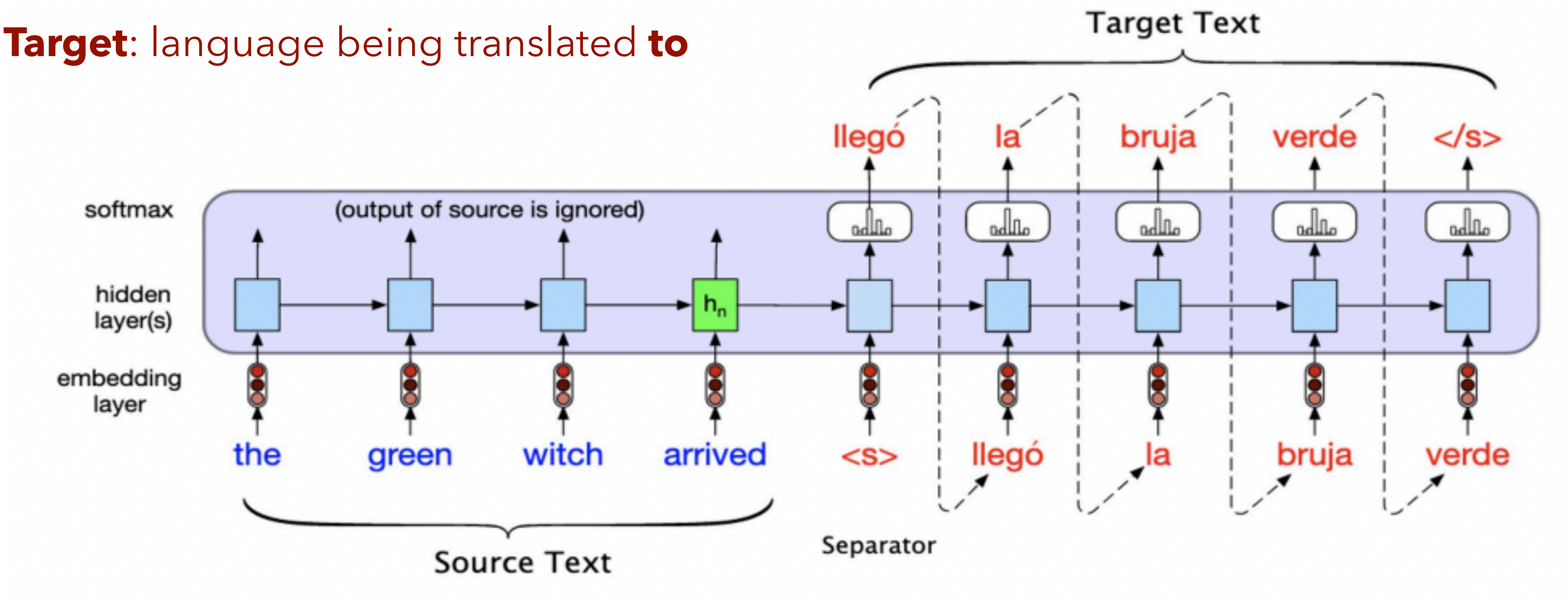


Encoder-Decoder RNNs: Example Machine Translation

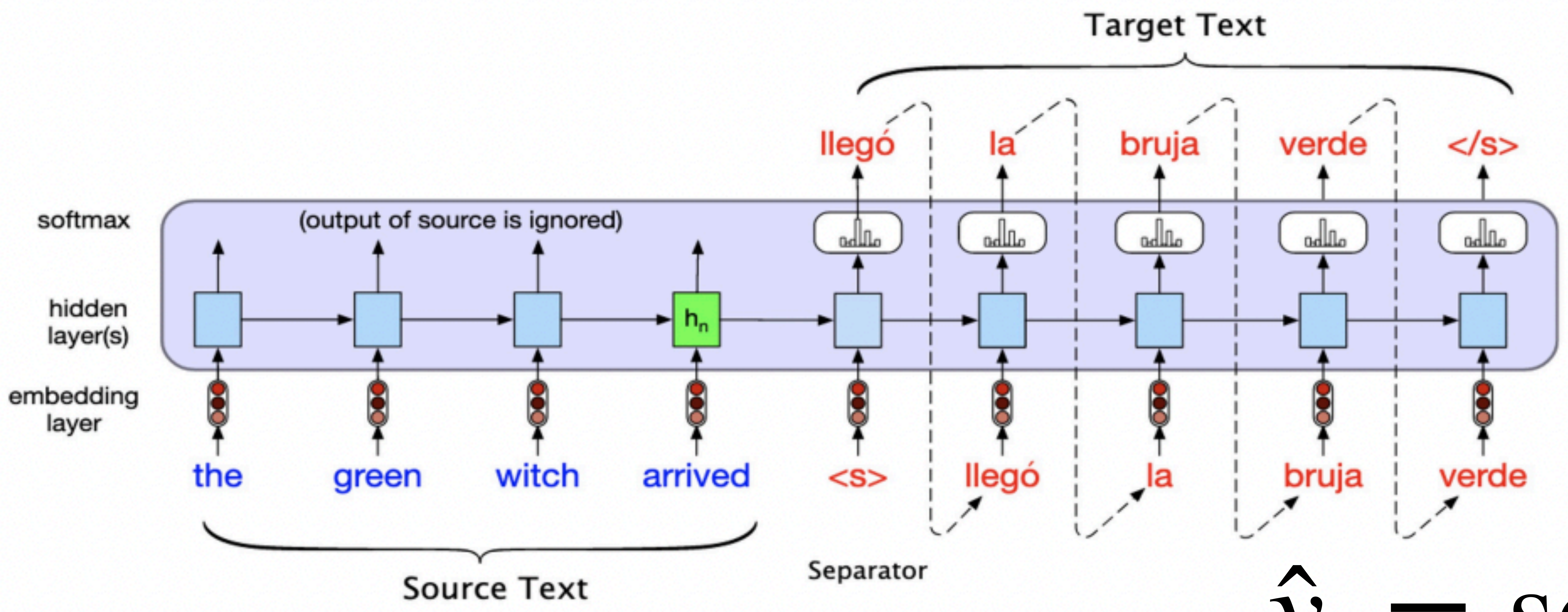
Training data: pairs of translated sentences (source lang. text, target lang. text)

Source: language being translated **from**

Target: language being translated **to**



Encoder-Decoder RNNs: Example Machine Translation



$$c = h_n^e$$

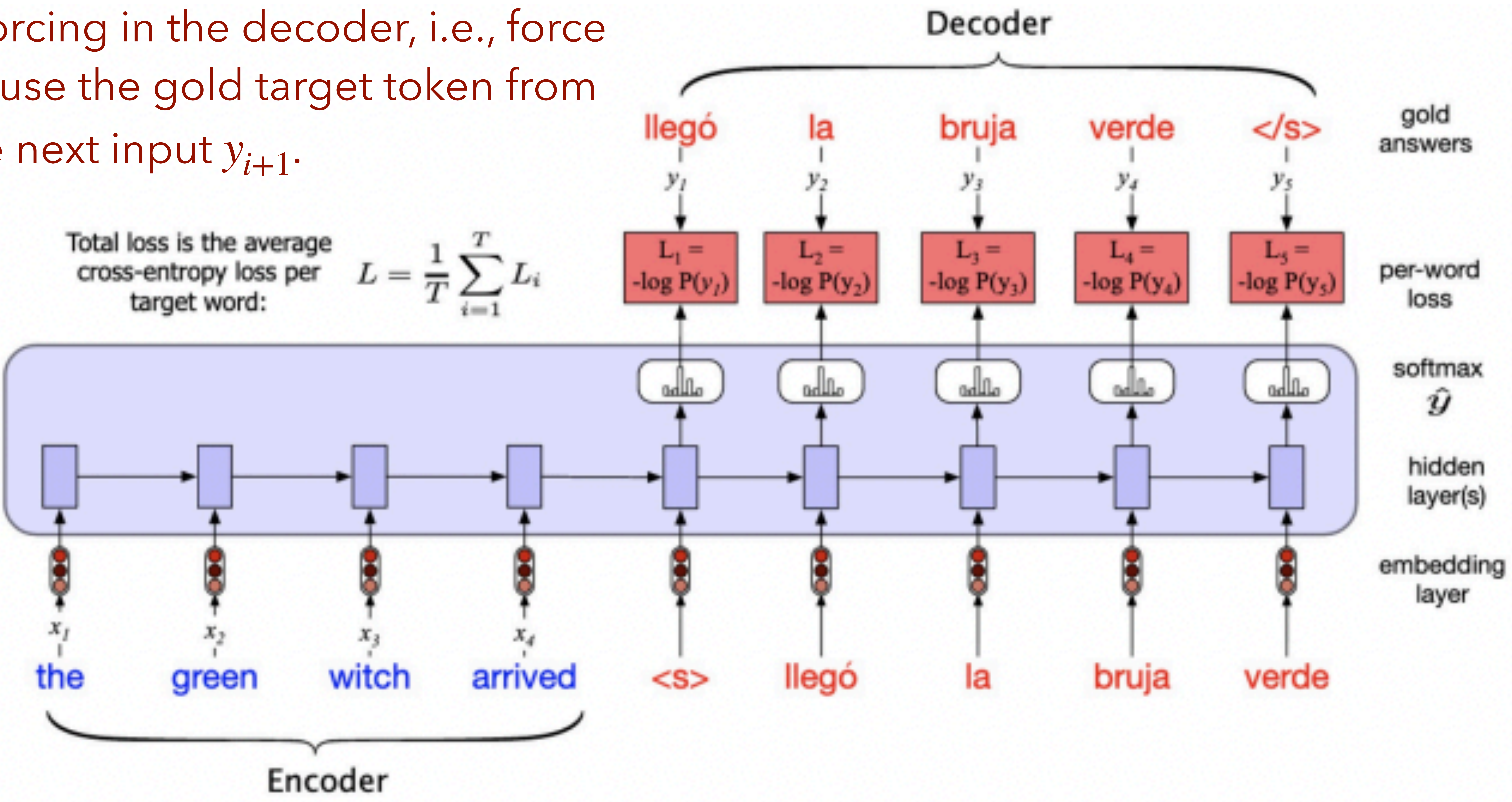
$$h_0^d = c$$

$$\hat{y}_i = \mathrm{softmax}(Vh_i^d)$$

What is the loss for neural machine translation?

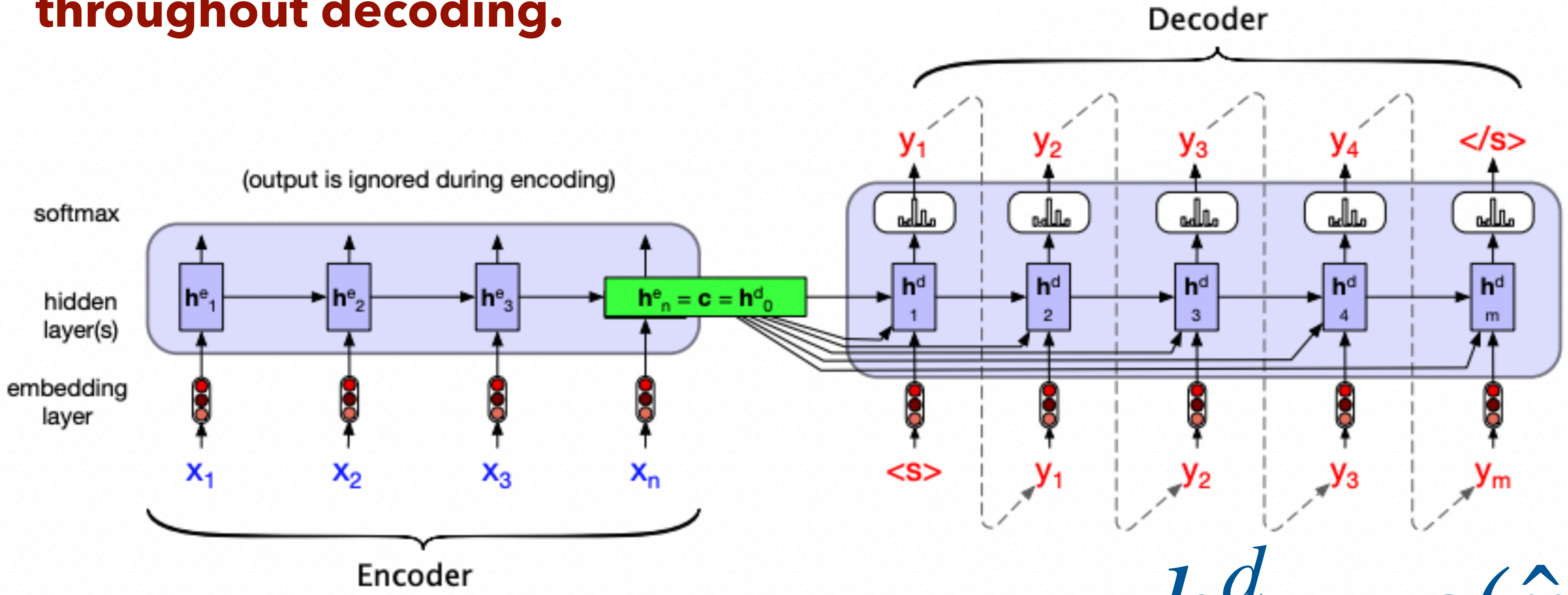
Training an Encoder-Decoder model

Use teacher forcing in the decoder, i.e., force the system to use the gold target token from training as the next input y_{i+1}.



Training an Encoder-Decoder model

We can make the context available throughout decoding.



$$c = h^e_n$$

$$h^d_0 = c$$

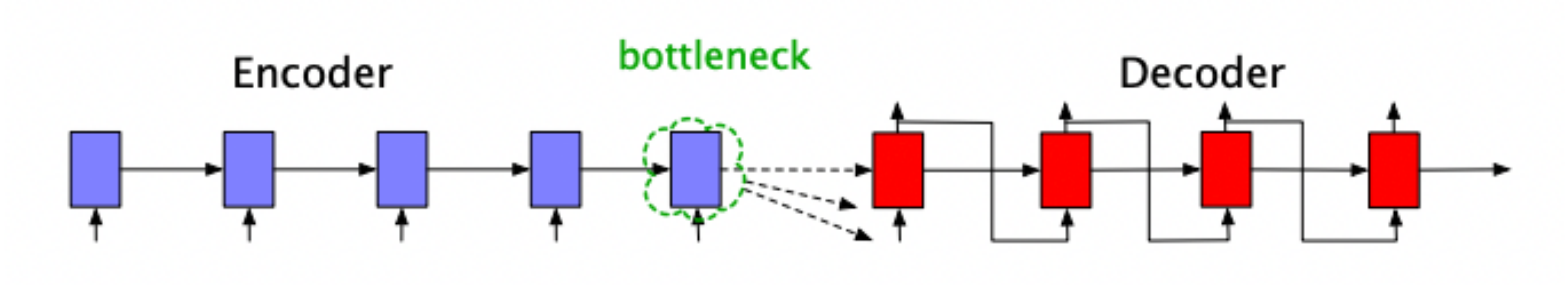
$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c)$$

$$\hat{y}_i = \mathrm{softmax}(Vh^d_i)$$

Break

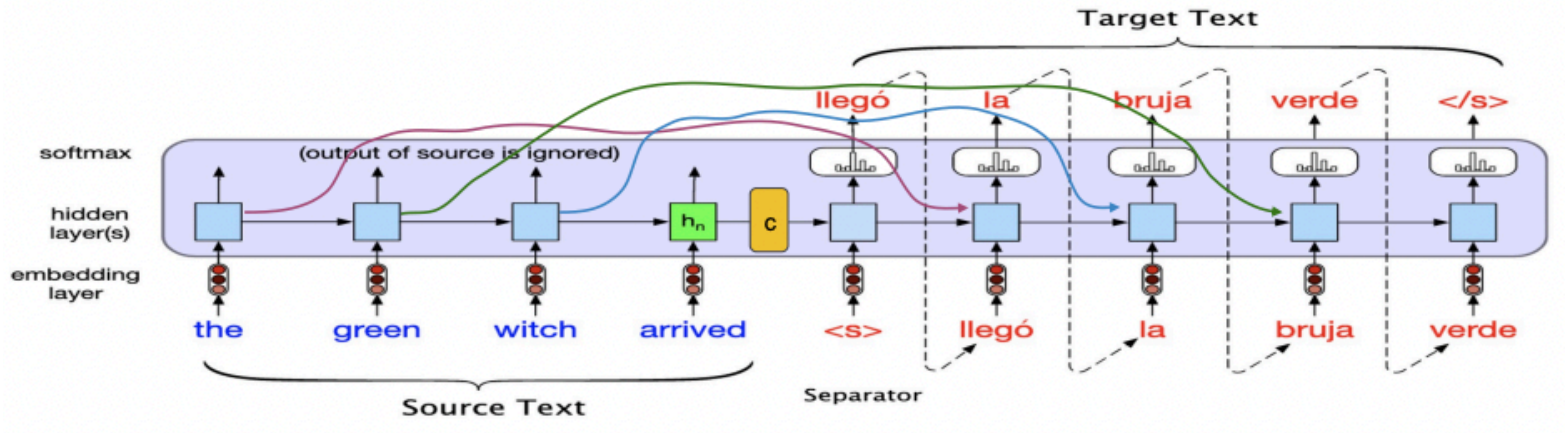
Encoder-decoders (as presented so far) are making a tradeoff

- By construction, we are compressing the **entire** context (which can be very long) into a single context vector c.
  - But what if we need to recall a very specific fact?



Possible Alternative: Attention to relevant encoder states

- If we knew which relevant encoder state h_j^e to use for our particular decoding step, we could use that instead of a fixed c .



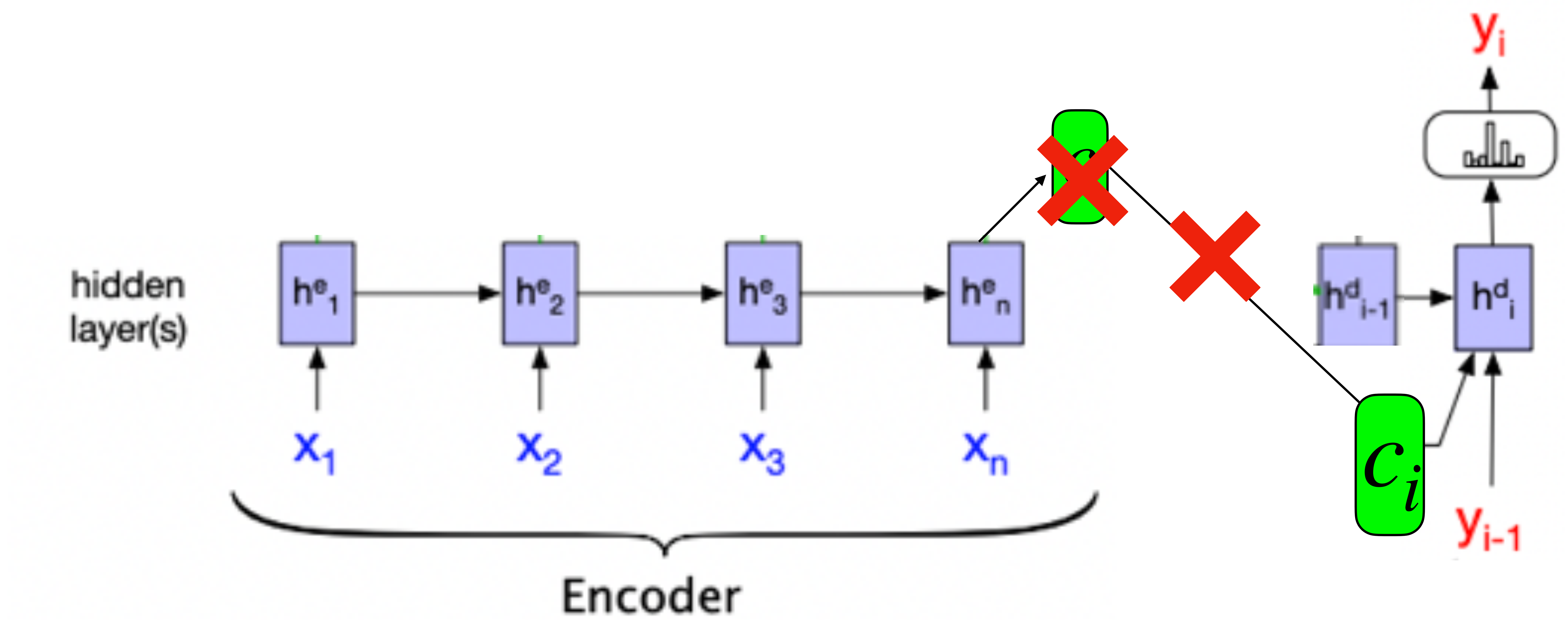
Possible Alternative: Attention to relevant encoder states

- If we knew which relevant encoder state h_j^e to use for our particular decoding step, we could use that instead of a fixed c .
- How to find which encoder state h_j^e is most relevant to our current decoding decision h_i^d?
  - One possible solution: use cosine similarity.?

More generally, need a method for computing c that can…

- Take the entire encoder context into account
- Dynamically update it during the course of decoding to focus on the most relevant tokens in the input
- Be embodied in a fixed-size vector

Attention: Solution we are building up to



Dynamically change the context vector at each decode step.

$$\cancel{h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c)}$$

$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

How do we dynamically construct c_i?

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

- Let c_t be a **weighted** average of the encoder all states h_i^e (instead of the last encoder state).

$$c_i = \sum_j w_j^i \, . \, h_j^e$$

- The weights w_j^i, i.e. how important is h_j^e at decoder step I, needs to be decided dynamically.

Pseudocode

- Decide w_i by scoring each encoder state against the previous decoder state h_{i-1}^d

$$\text{score}(h_{i-1}^d, h_j^e)$$

- Softmax the scores to get weights.

$$w_j^i = \text{softmax}[\text{score}(h_{i-1}^d, h_j^e) \forall j \in \text{encoder hidden states}]$$

- Take the weighted average.

How do we dynamically construct c_i?

How do we get $\text{score}(h^d_{i-1}, h^e_j)$?

Multiple options:

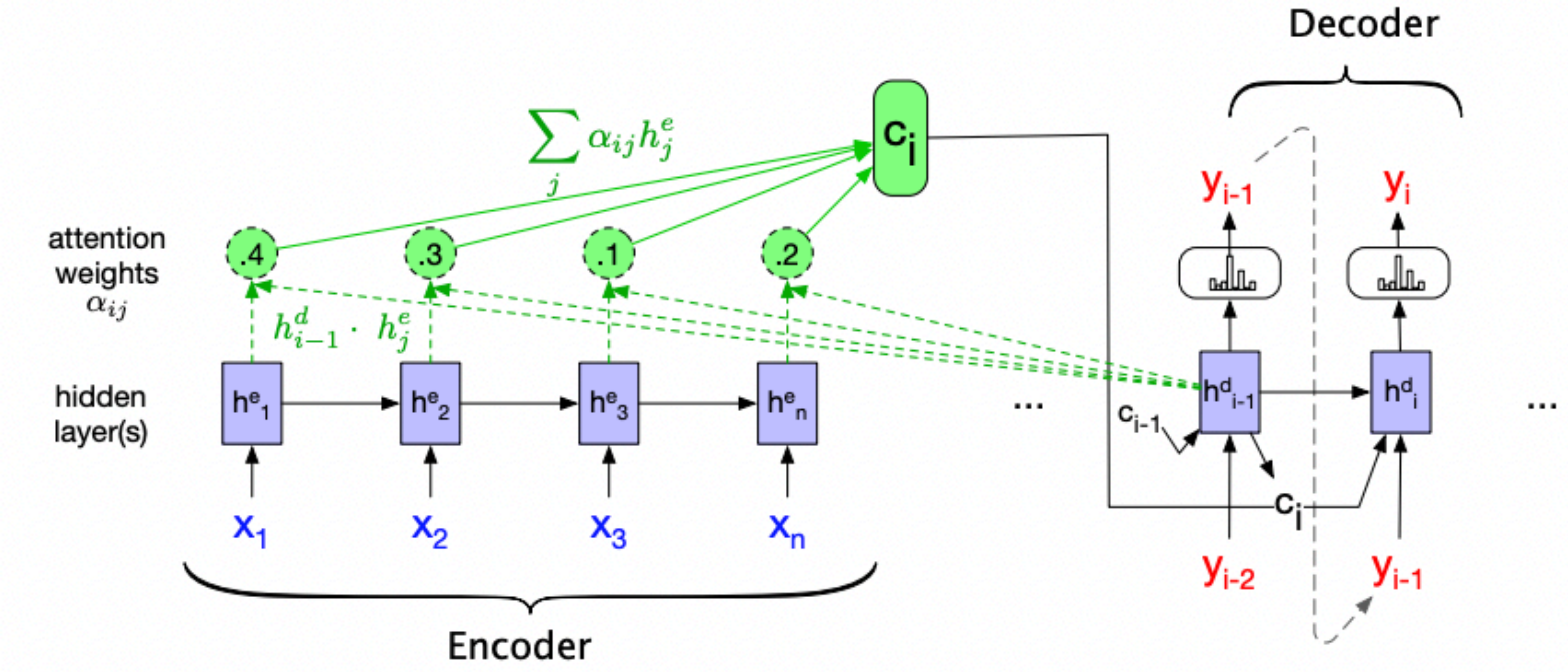
- Dot-product attention ("similarity between the prev decoder state and encoder state). Also called **Luong Attention**.

  $$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \,.\, h^e_j$$

- Some other function of h^d_{i-1} & h^e_j
  - E.g. $\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} W_s \, h^e_j$ (W_s is learned).
  - Bahdanau attention uses a FFNN with h^d_{i-1} & h^e_j as inputs. (Bahdanau et al. introduced the idea of attention for RNNs)

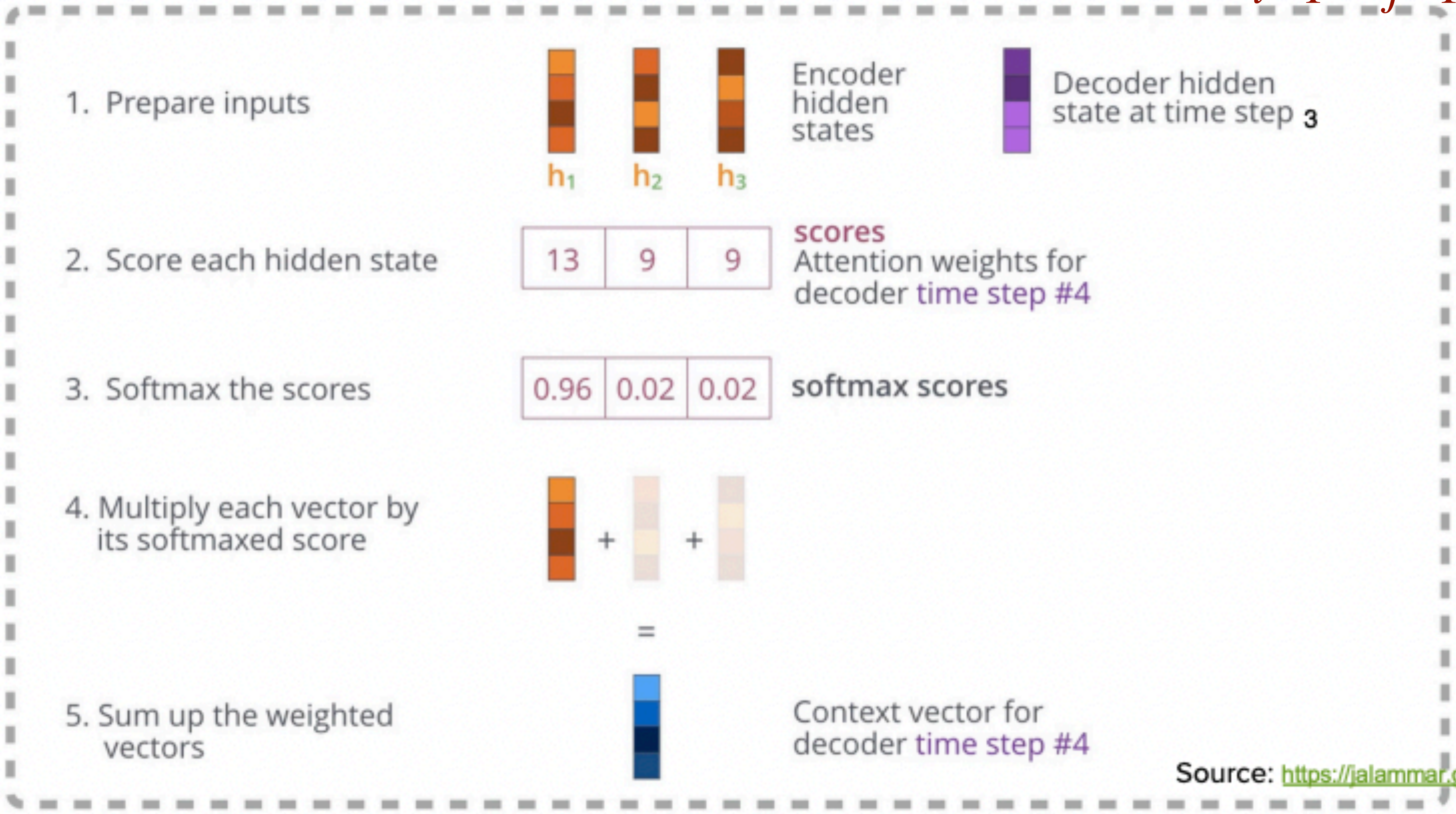
Putting it all together: Luong Attention

For time step i of decoder.



Putting it all together: Luong Attention

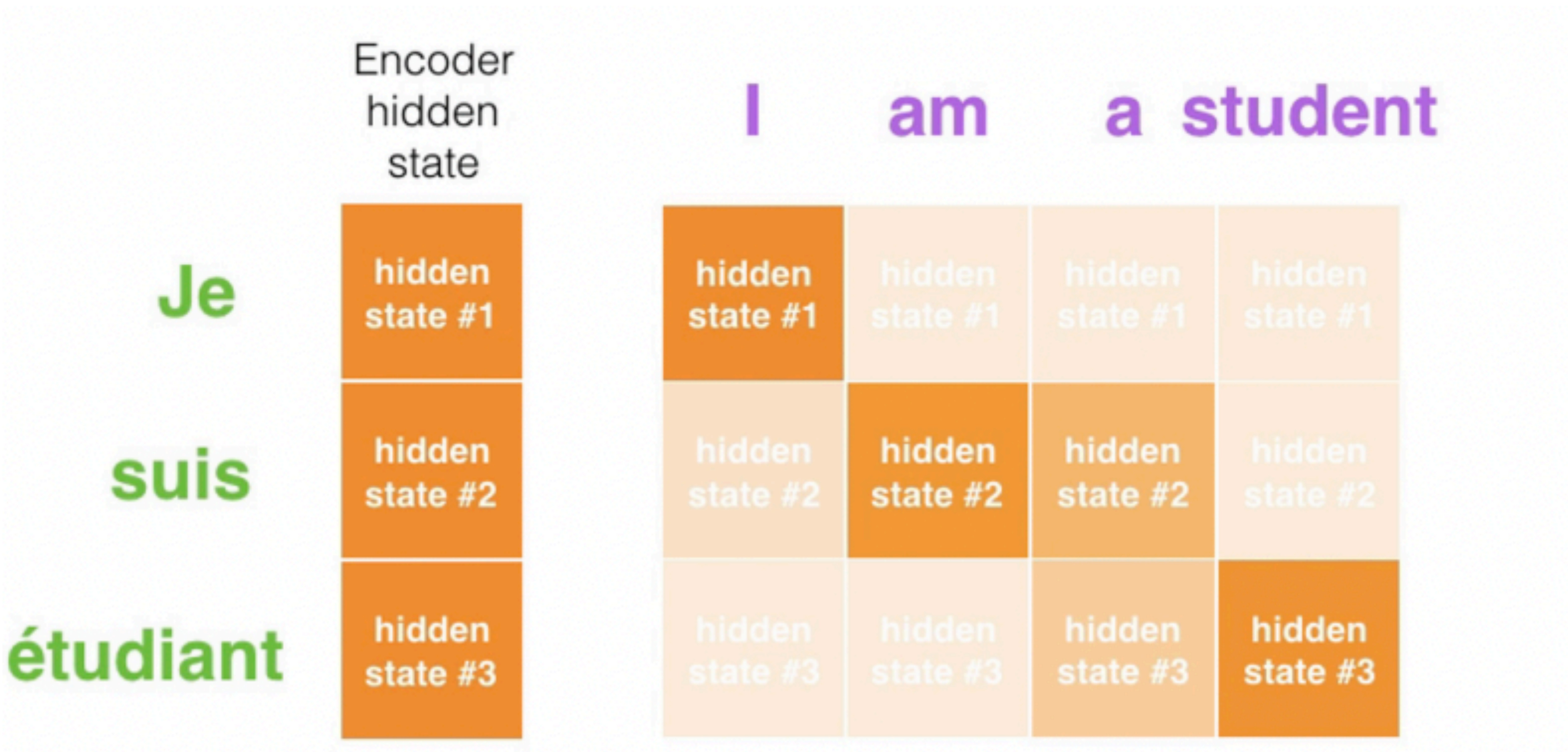
For time step i of decoder (only shows how to compute $\mathbf{c}_\mathrm{i} = f(h^d_{i-1}, h^d_{j=1:n})$



1. Prepare inputs
Encoder hidden states
Decoder hidden state at time step 3
h1
h2
h3
2. Score each hidden state
13
9
9
scores
Attention weights for decoder time step #4
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

Source: https://jalammar.github.io/

What is actually does?



Source: https://jalammar.github.io/

Today's takeaways

- RNNs help us model sequences.
  - Useful for any task where order information is important.
  - Can be used for token-level classification, seq-level classification or seq2seq tasks.

- Architecture:
  - Simple RNN
  - Bidirectional RNN
  - Multi-layer RNN (can stack multiple RNN layers on top of each other)
  - RNNs w/ Attention

So RNNs can really take arbitrarily long history into account?

In practice, no.

- Vanishing gradient (more on this later).
- It can be very slow. To process 10000 tokens, we need to auto regressively compute h_t for each time step.

Slide Acknowledgements

- Earlier versions of this course offerings including materials from Claire Cardie, Marten van Schijndel, Lillian Lee
- CS 288 course by Alane Suhr (UC Berkeley).