EECS 230 Deep Learning
Lecture 11: RNN and LSTM

Credit to Daniel Jurafsky for figures of network architectures

Recap: Language models

Many NLP tasks require **natural language output**:

- **Machine translation**
- **Speech recognition**
- **Natural language generation**
- **Spell-checking**

Language models define **probability distributions over** (natural language) **strings or sentences**.

➔ We can use a language model to generate strings

➔ We can use a language model to score/rank candidate strings so that we can choose the best (i.e. most likely) one:
if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Recap: N-gram Language model

N-gram models *assume* each word (event) depends only on the previous n–1 words (events):

Unigram model: $P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$

Trigram model: $P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)})$

Independence assumptions where the n-th event in a sequence depends only on the last n-1 events are called Markov assumptions (of order n–1).

Recap: Learning a language model

We need (a large amount of) text as training data to estimate the parameters of a language model.

The most basic parameter estimation technique: relative frequency estimation (frequency = counts)

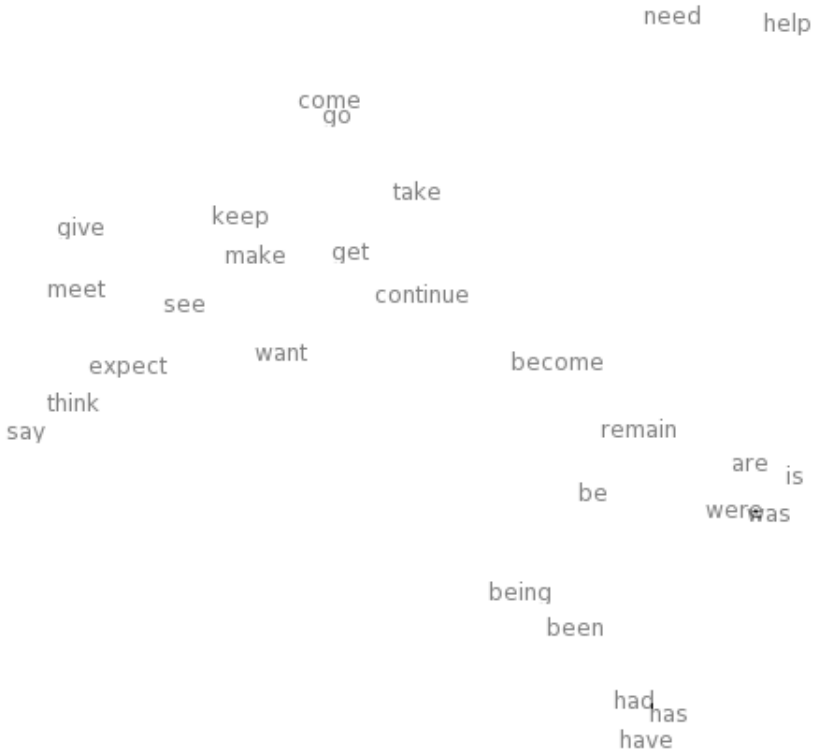
$$P(w^{(i)} = \text{`the'} \mid w^{(i-1)} = \text{`on'}) = C(\text{`on the'}) / C(\text{`on'})$$

Also called Maximum Likelihood Estimation (MLE)

Recap: Word Embedding

expect =

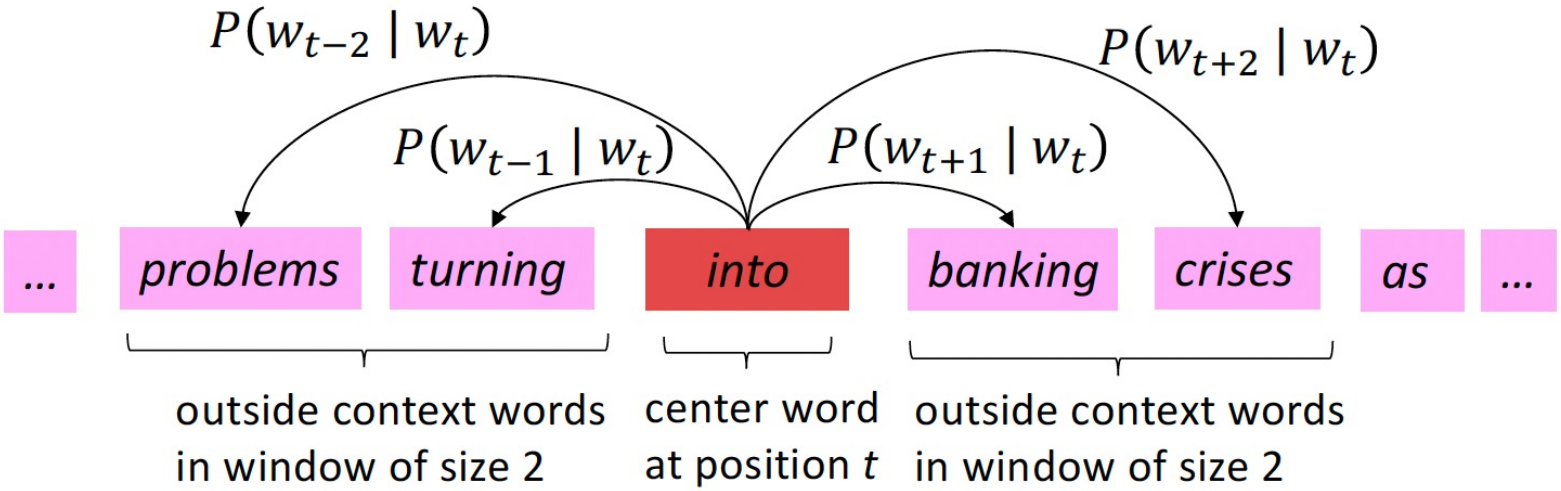
$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



UCMERCED

Recap: Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Recap: Word2Vec Objective Function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Our first neural net for NLP:

A neural n-gram model

Given a fixed-size vocabulary V, an *n*-gram model predicts the probability of the *n*-th word following the preceding *n*–1 words:

$$P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)}, \ldots, w^{i-(n-1)})$$

How can we model this with a neural net?

— **Input layer:** concatenate *n*–1 word vectors

— **Output layer:** a softmax over |V| units

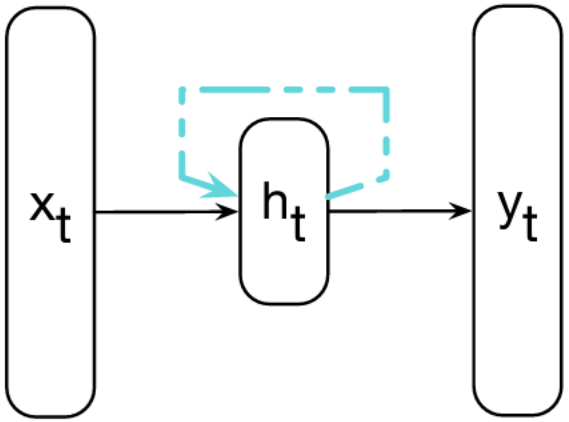
— Feedforward network

Outline

- ❑Recurrent Neural Network
- ❑RNNs as Language Models
- ❑RNNs for other NLP tasks
- ❑Staked and Bi-directional RNN
- ❑LSTM
- ❑Encoder-decoder model

Recurrent Neural Network

- Temporal nature in language processing
- RNN deals with sequential input data stream like language.

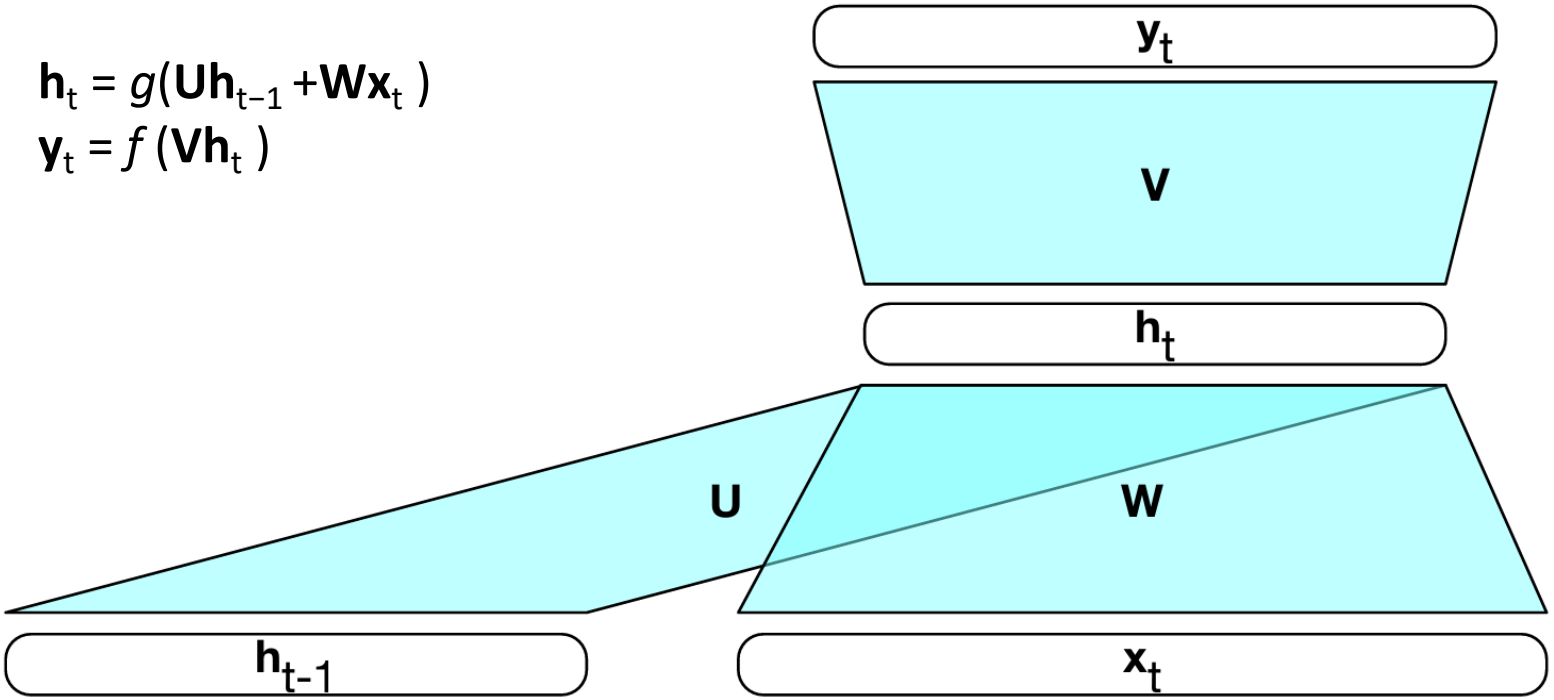


A simple RNN

A Simple Recurrent Neural Network

- RNN illustrated as a feed-forward network

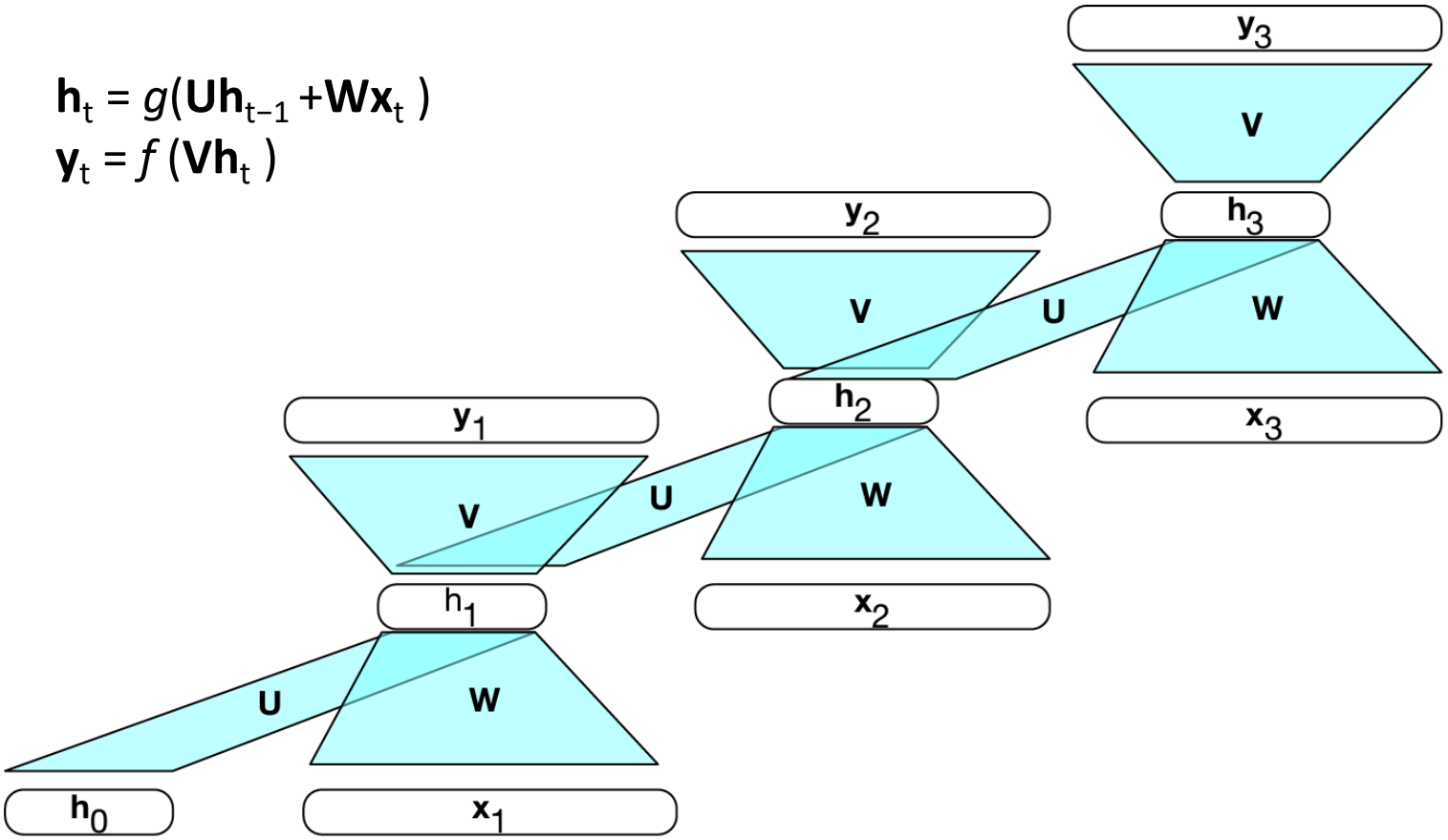
$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$
$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$



A Simple Recurrent Neural Network

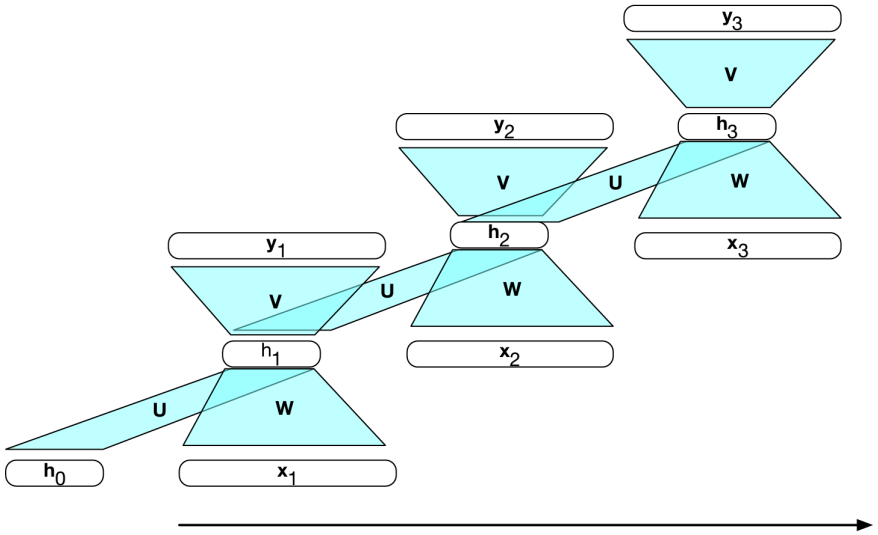
- RNN unrolled in time

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$



How to optimize Recurrent Neural Network?

$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$
$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$



❑Backpropagation through time

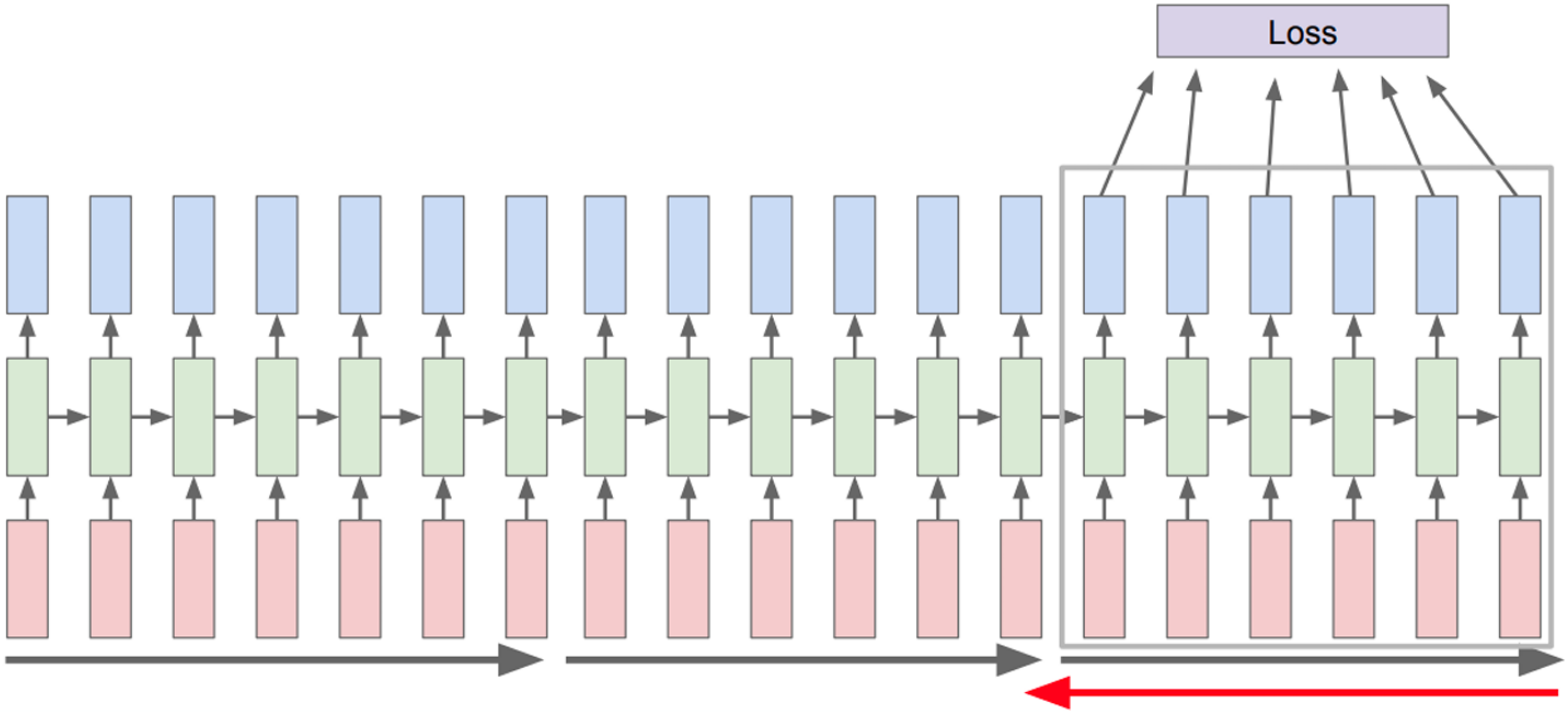
$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

UCMERCED

Truncated backpropagation through time

- Backpropagation is very expensive for long sequences



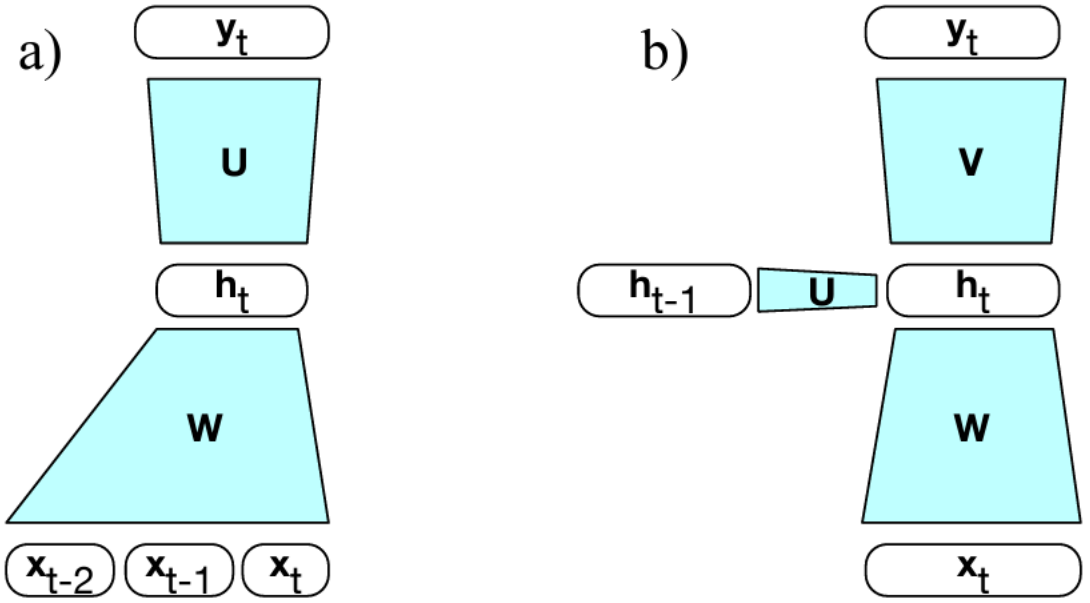
- Run forward and backward through chunks of the sequence instead of whole sequence

RNNs as Language Models

- Language models predict the next word in a sequence given some preceding context.

$$P(\text{fish} \mid \text{Thanks for all the})$$

- RNN Language Model

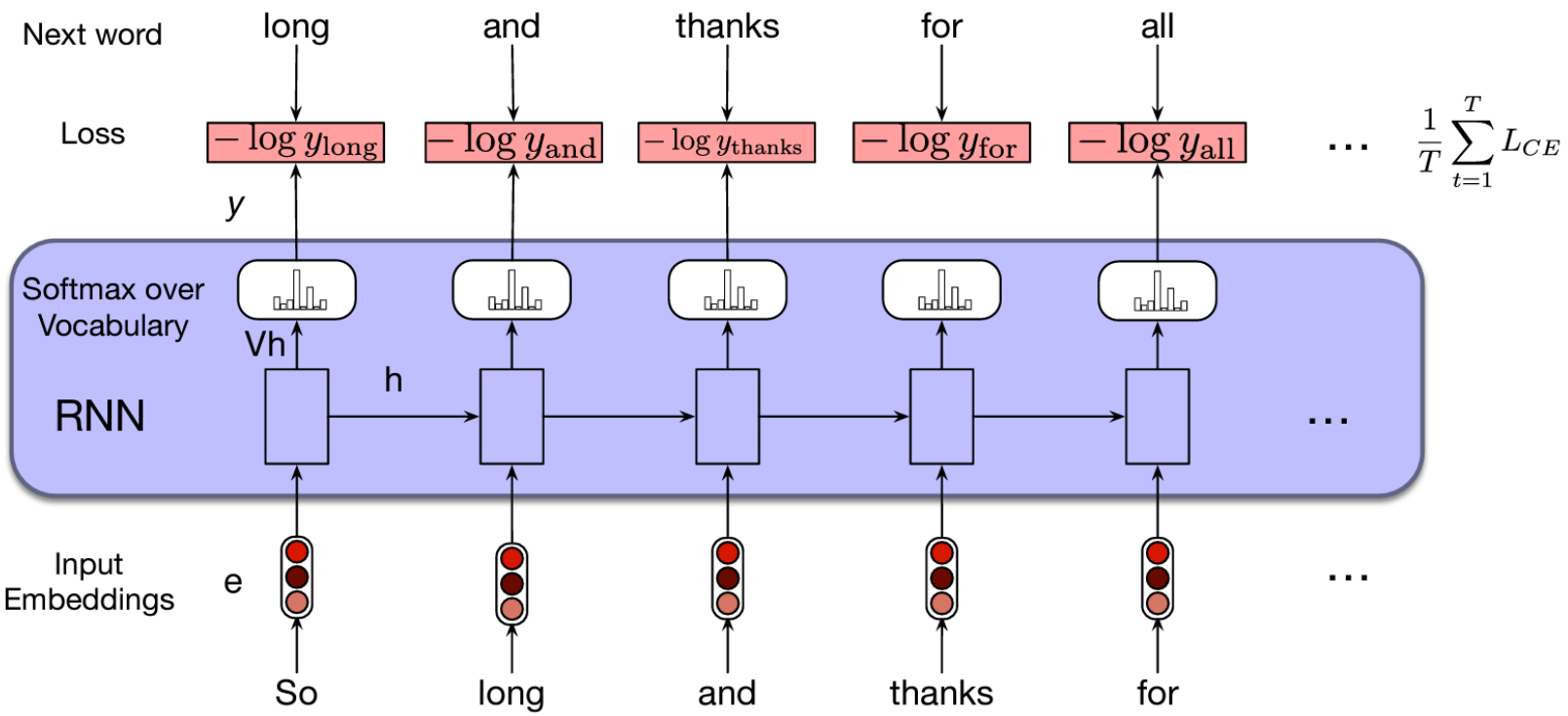


$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

UCMERCED

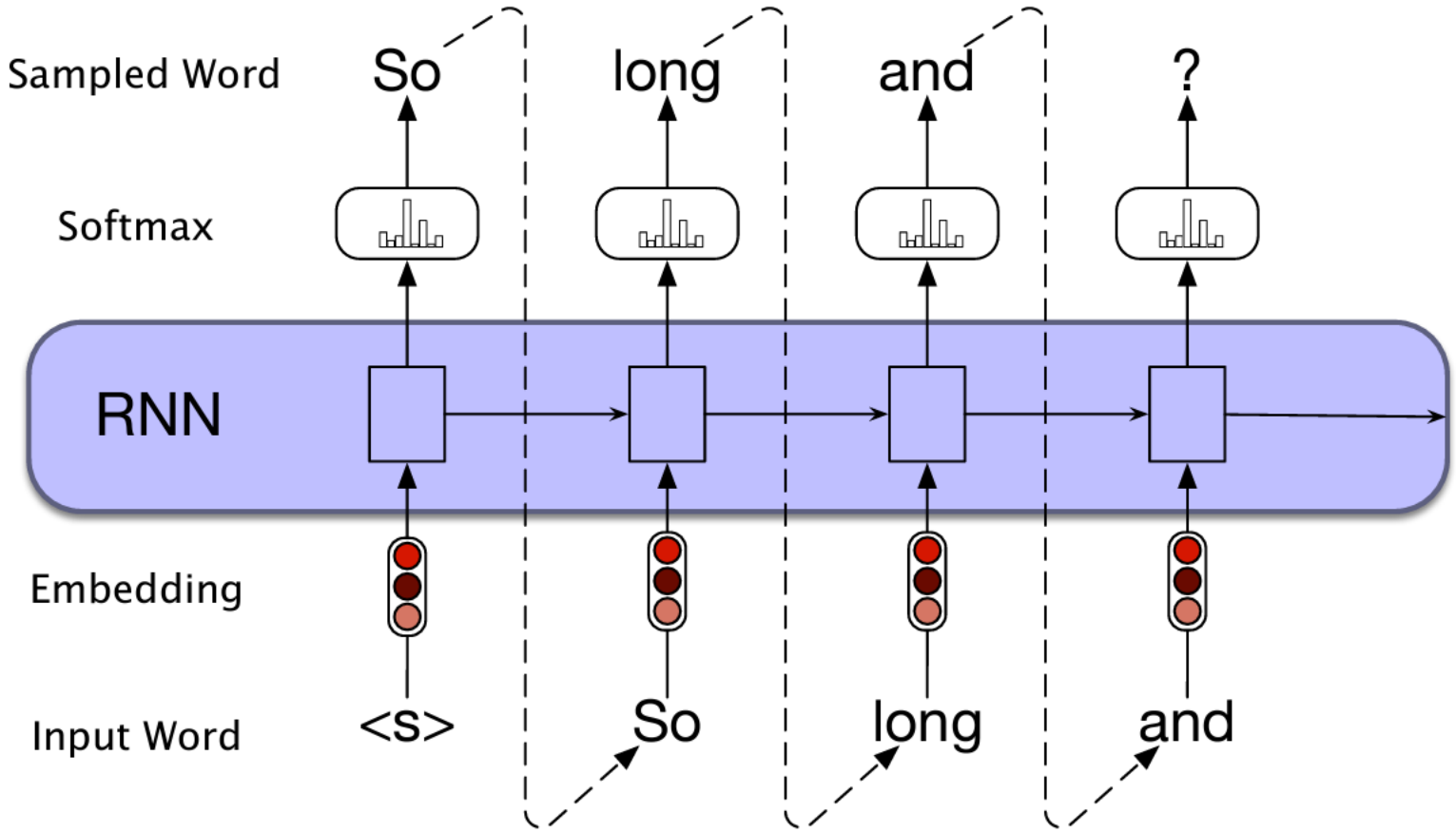
Training an RNN Language Model

- Maximum likelihood estimation



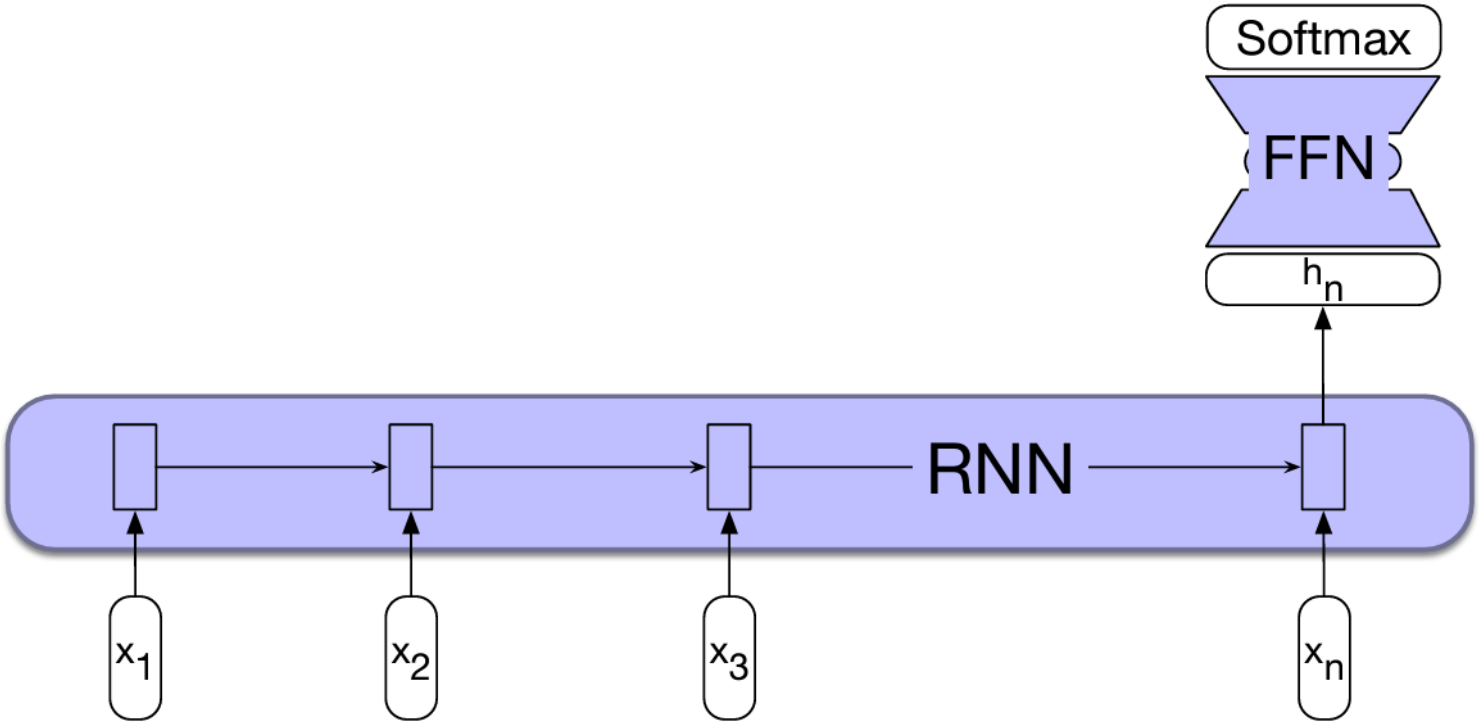
Generation with RNN Language Model

- Autoregressive (casual) generation
- Unlike teacher forcing during training



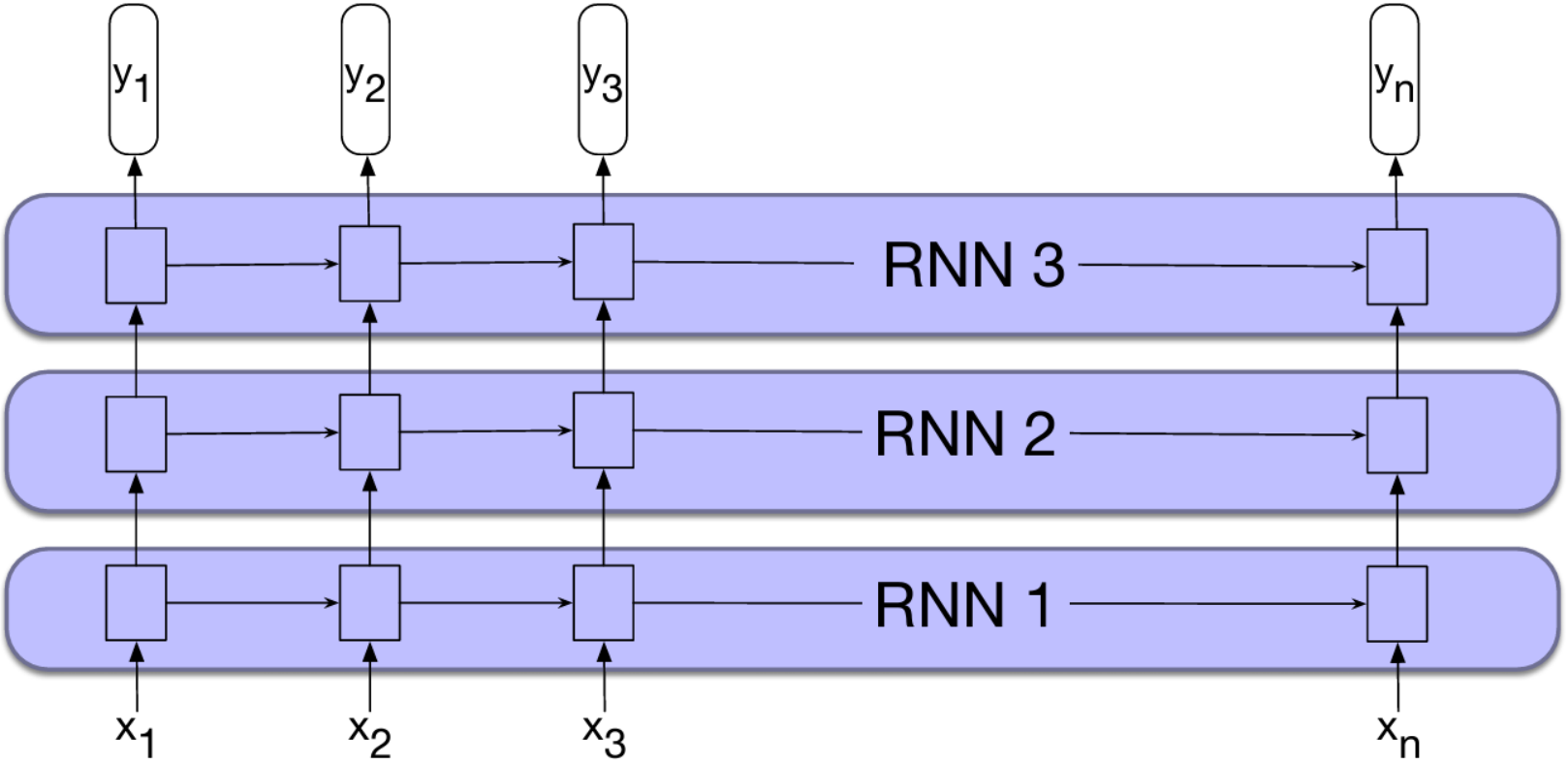
RNNs for other NLP tasks

- RNN for sequence classification
- Commonly called text classification, like sentiment analysis or spam detection



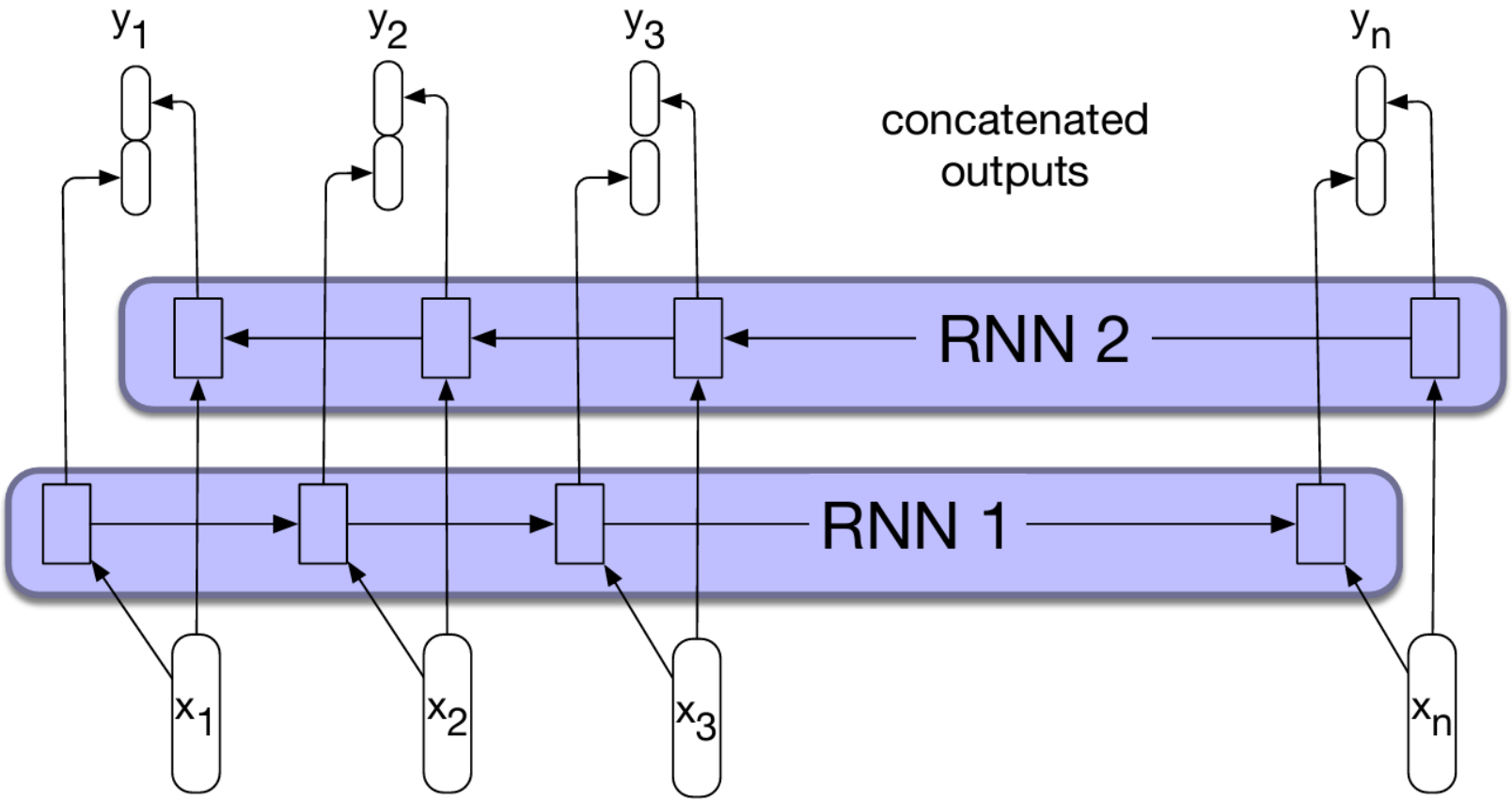
Stacked RNN

- Outputs from one RNN as an input sequence to another one



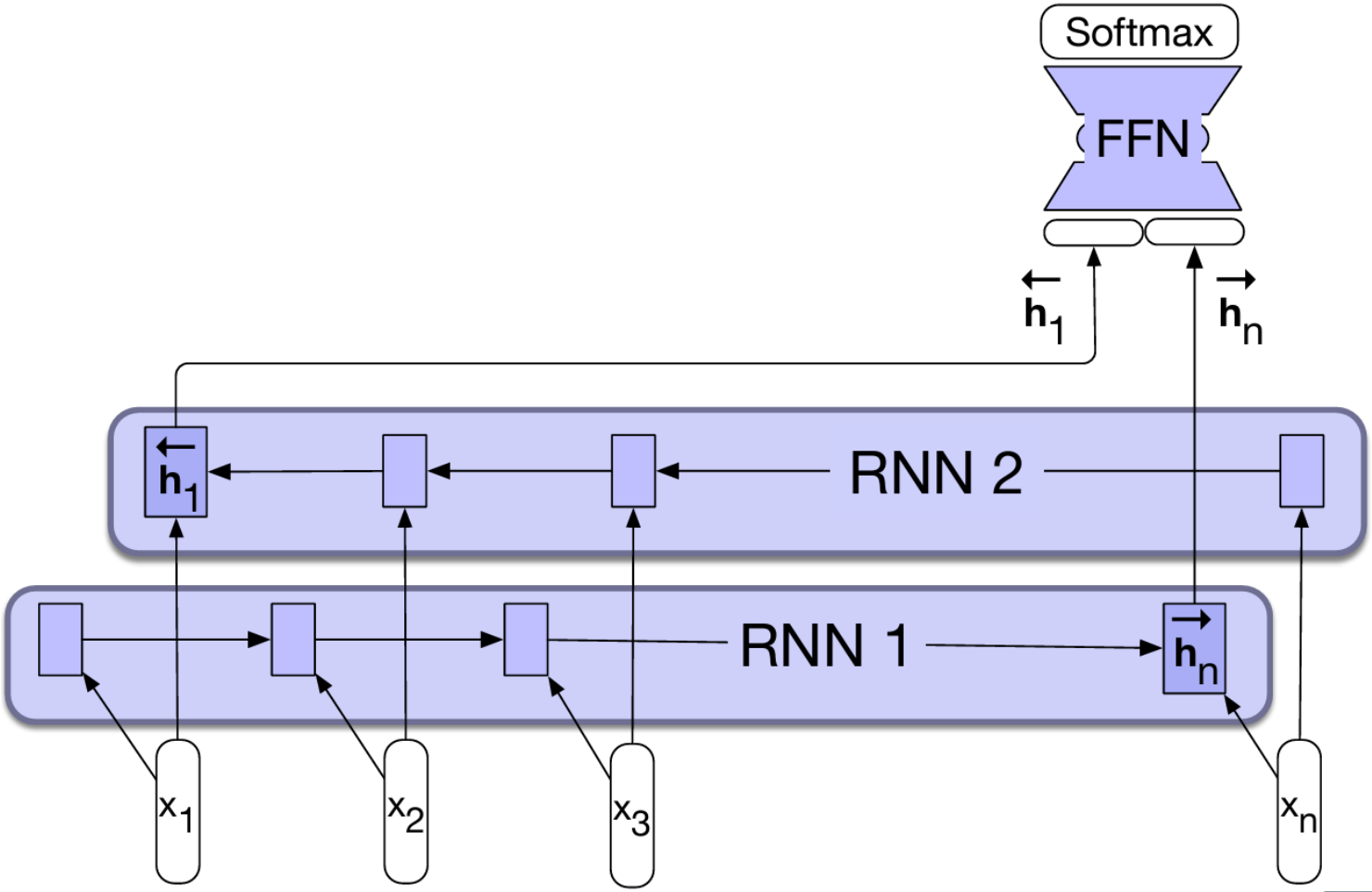
Bidirectional RNN

- Two independent RNN, one forward, another backward



Bidirectional RNN for sequence classification

- Combine hidden states from first & last word



Vanishing/exploding gradients

- Consider the gradient of L_t at step t, with respect to the hidden state $\mathbf{h}_k$ at some previous step k (k<t):

$$\frac{\partial L_t}{\partial \mathbf{h}_k} = \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{t \geq j > k} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right)$$

- Recurrent multiplication
- Gradients too small (vanishing gradient) or too large (exploding gradient)

Exploding gradients

- What is the problem?
- We take a very large step in SGD
- Solution: Gradient clipping

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Vanishing gradients

- What is the problem?
- Parameters barely get updated (no learning)
- Solution:
  - LSTMs: Long short-term memory networks
  - GRUs: Gated recurrent units

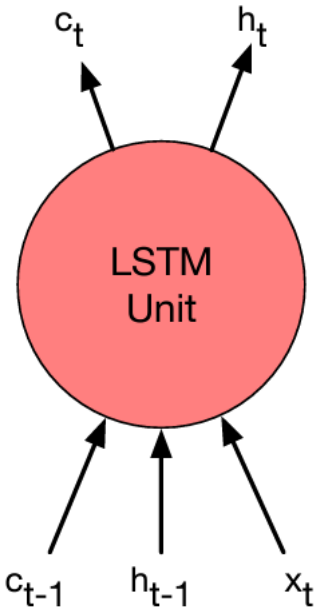
Local vs distant information

- Hidden states tend to contain local information
- But distant information is critical

  "The flights the airline was canceling were full"
- Should predict "were" given distant information (flights)

Long Short-term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem
- Basic idea: turning multiplication into addition
- Use "gates" to control how much information to add/erase
- At each timestep, there is a hidden state $\mathbf{h}_t$ (local information) and also a cell state $\mathbf{C}_t$ (distant information)



UCMERCED

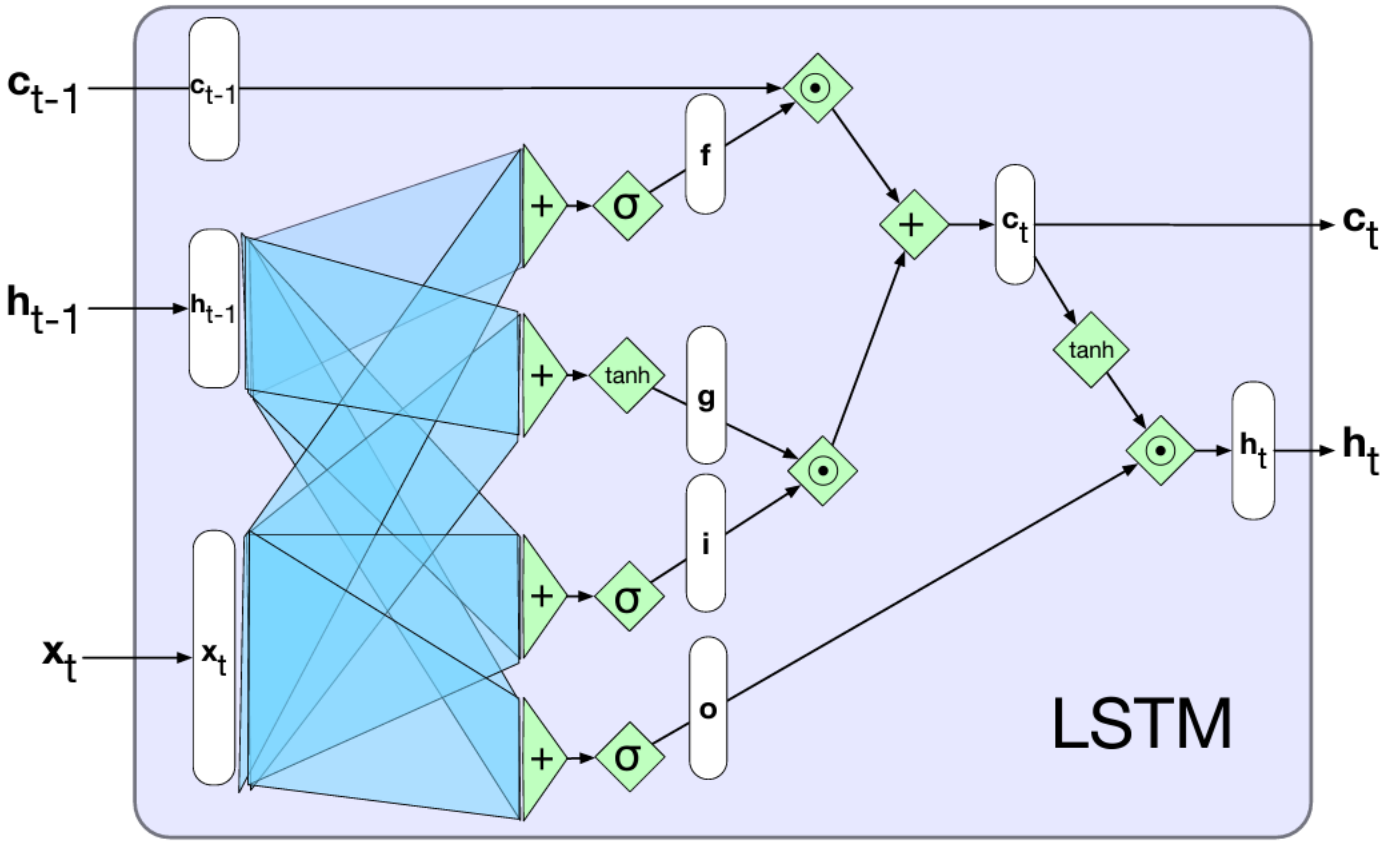
Long Short-term Memory (LSTM)

- Gate: feedforward layer, followed by a sigmoid activation function, followed by a pointwise multiplication with the layer being gated
- For example, output gate (What to output for hidden state)

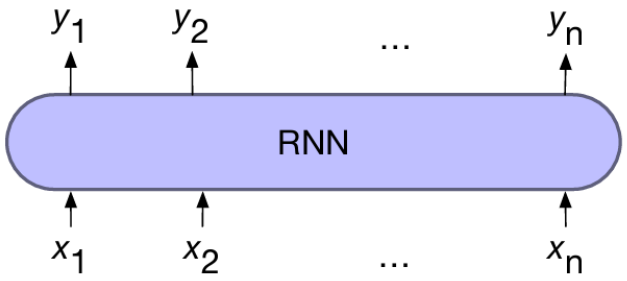
$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

- Other gates
  - Forget gate
  - Add gate
  - Input gate

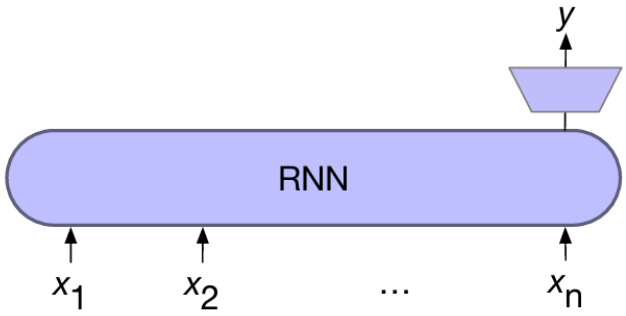
Long Short-term Memory (LSTM)



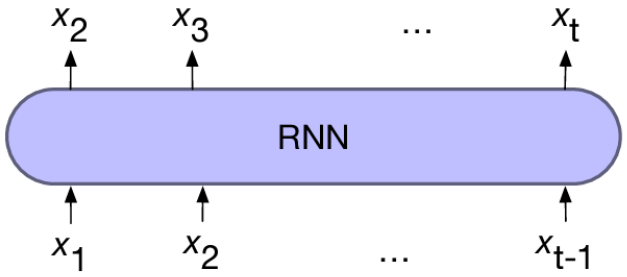
Summary: Common RNN NLP architectures



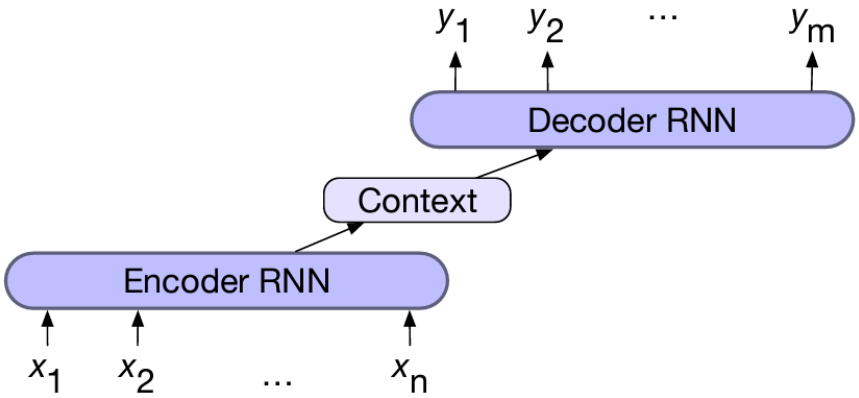
a) sequence labeling



b) sequence classification



c) language modeling

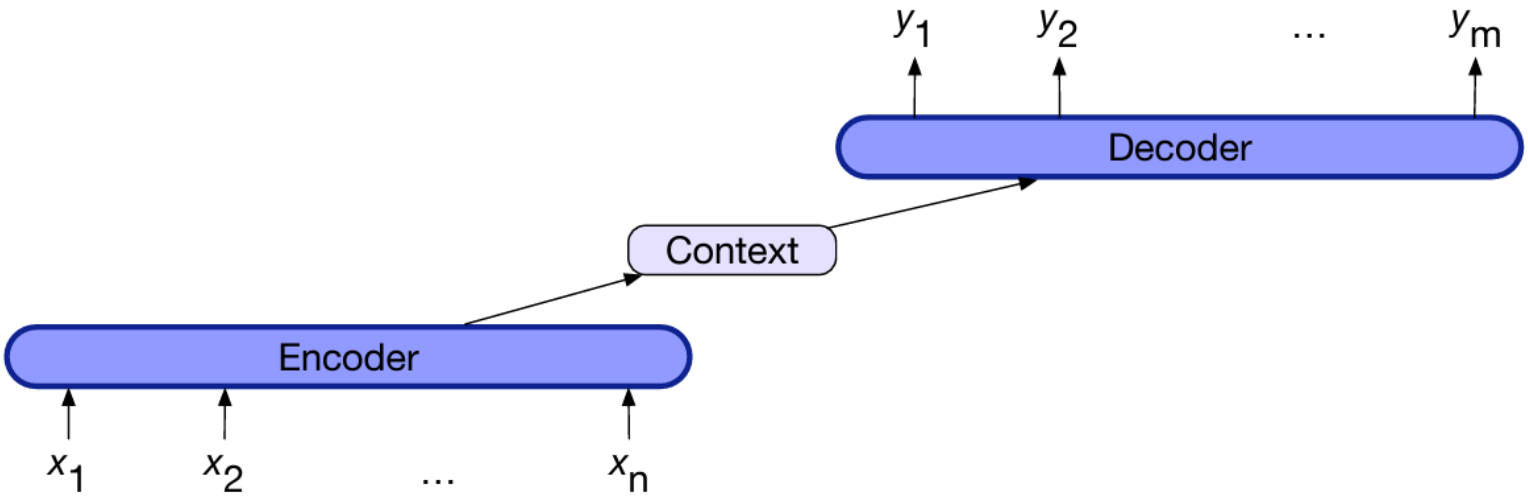


d) encoder-decoder

UCMERCED

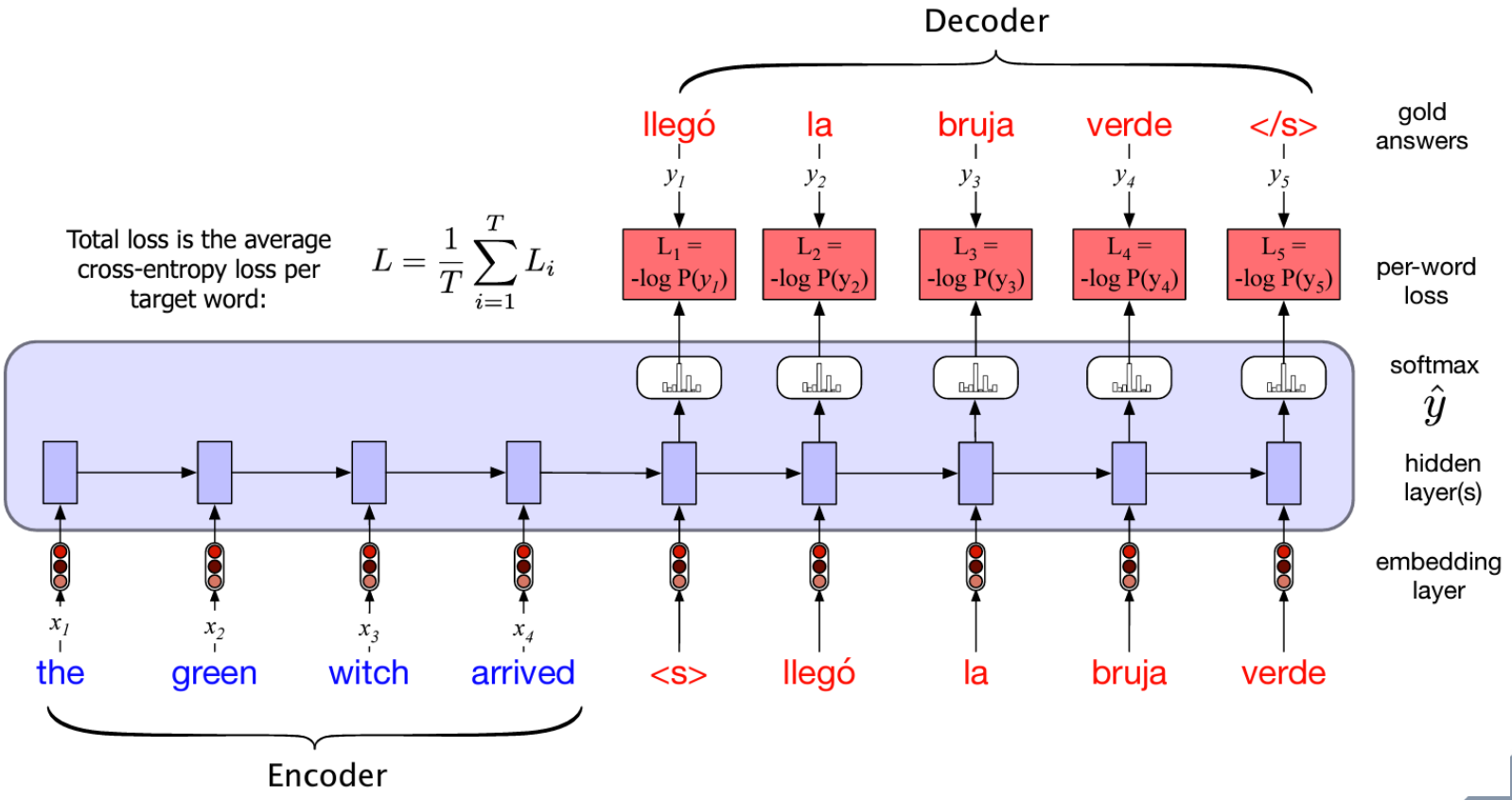
Encoder Decoder Architecture

- Arbitrary length output given an input sequence
- A.K.A. sequence-to-sequence network
- Context vector conveys the essence of the input to the decoder



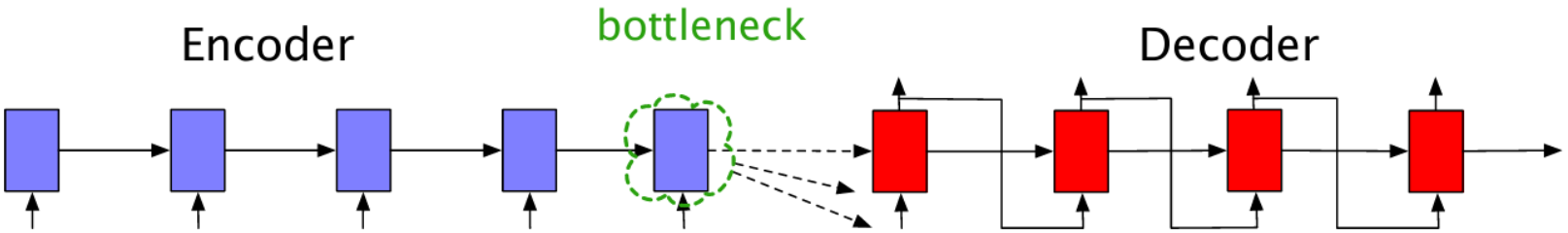
Encoder Decoder Architecture

- Training an encoder-decoder for machine translation



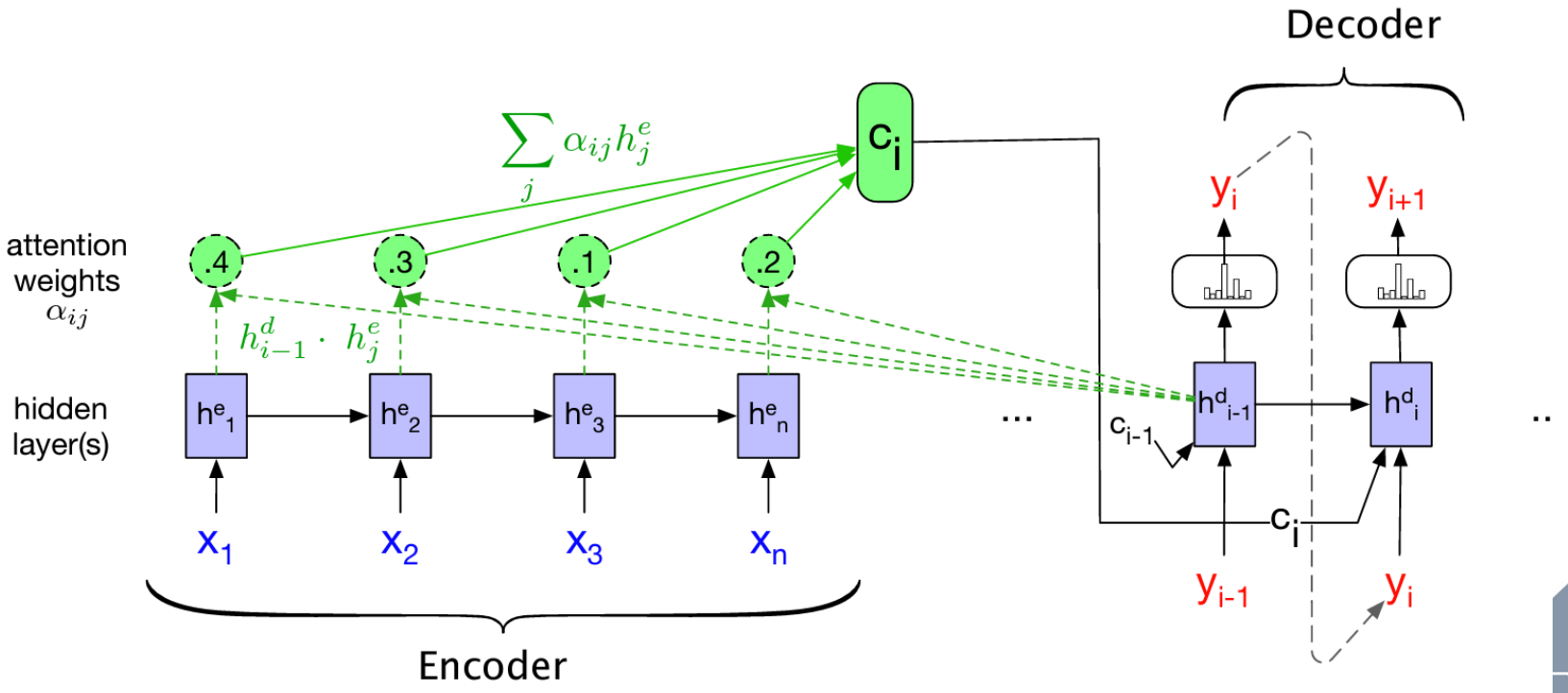
Problem of Encoder-decoder architecture

- Context vector encodes EVERYTHING about input sequence
- Context vector acts as a bottleneck



Attention Mechanism

- Each output in decoder accesses all the hidden states from the encoder, not just the last state
- Each output attends to all input



What's next?

- ❑Self-attention
- ❑Transformer block
- ❑Transformer architecture
- ❑Large Language Model
- ❑Vision Transformers