Language Modelling

AC295

Pavlos Protopapas
Institute for Applied Computational Science, Harvard

Announcements

- Vote!
- Submit your reading questions by Wed 10/21 noon on Ed.
- Exercise was due 10:15 am, next coming up today.

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

- FFNN, RNNs/LSTMs +ELMo
- Seq2Seq

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

- FFNN, RNNs/LSTMs +ELMo
- Seq2Seq

Common NLP Tasks

Morphological analysis

- Part-of-speech (POS) tagging
- Stemming

Syntactic analysis

- Sentence breaking

Lexical semantics

- Named entity recognition (NER)
- Sentiment analysis

Text and speech processing

- Optical character recognition
- Speech recognition
- Text-to-speech

Common NLP Tasks

Higher-level NLP applications

- Text summarization
- Machine translation
- Natural language generation
- Question answering

https://en.wikipedia.org/wiki/Natural_language_processing#Common_NLP_Tasks

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, RNNs/LSTMs +ELMo

Seq2Seq

Transfer Learning in NLP

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, RNNs/LSTMs +ELMo

Seq2Seq

Language Modelling

We will fist focus on Language Modelling (LM) because:

It's foundational for nearly all NLP tasks

A Language Model estimates the probability of any sequence of words

Let $\boldsymbol{X}$ = "Patrick was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

$\mathrm{P}(\boldsymbol{X}) = P($"Patrick was late for class"$)$

Language Modeling

Regardless of how we model sequential data, keep in mind that we can estimate any sequential data or a time series as follows:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1)$$

This compounds for all subsequent events, too

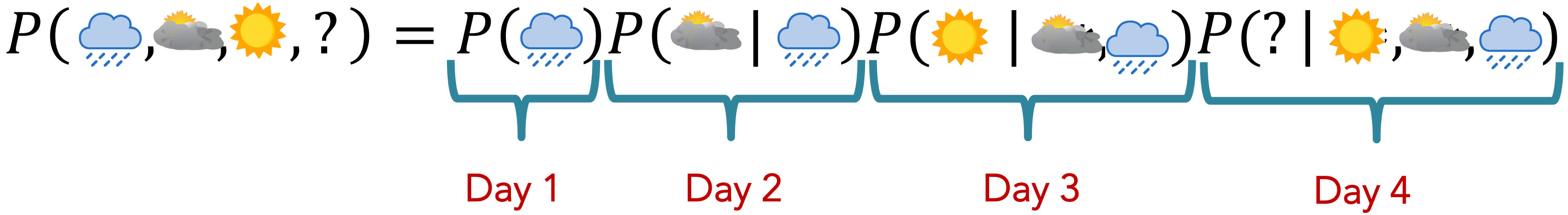
Joint distribution of all measurements

Conditional probability of an event, depends on all of the events that occurred before it.

Example

Why is it useful to accurately estimate the joint probability of any given sequence of length N?

Having the ability to estimate the probability of any sequence of length N allows us to determine the most likely next event (i.e., sequence of length $N + 1$)



Language Modelling: unigrams

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others.

Count how often each word occurs (in the training data).

Language Modelling: unigrams

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

Let $\boldsymbol{X}$ = "Patrick was late for class"

w_1 w_2 w_3 w_4 w_5

Language Modelling: unigrams

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

Let $\boldsymbol{X}$ = "Patrick was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

You calculate each of these probabilities from the training corpus

$$\mathrm{P}(\boldsymbol{X}) = P(\text{Patrick})P(\text{was})P(\text{late})P(\text{for})P(\text{class})$$

= 0.00015 * 0.01 * 0.004 * 0.03 * 0.0035
= $6.3\text{x}10^{-13}$

Language Modelling: unigrams

UNIGRAM ISSUES

Context doesn't play a role at all

$$P(\text{"Patrick was late for class"}) = P(\text{"class for was late Patrick"})$$

Sequence generation: What's the most likely next word?

Patrick was late for class ______

Language Modelling: unigrams

UNIGRAM ISSUES

Context doesn't play a role at all

$$P(\text{"Patrick was late for class"}) = P(\text{"class for was late Patrick"})$$

Sequence generation: What's the most likely next word?

Patrick was late for class ______

Patrick was late for class the

Language Modelling: unigrams

UNIGRAM ISSUES

Context doesn't play a role at all

$$P(\text{"Patrick was late for class"}) = P(\text{"class for was late Patrick"})$$

Sequence generation: What's the most likely next word?

Patrick was late for class _____

Patrick was late for class the

Patrick was late for class the the

Language Modelling: bigrams

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Patrick was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

Language Modelling: bigrams

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Patrick was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Patrick})$$

Language Modelling: bigrams

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Patrick was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Patrick})P(\text{late}|\text{was})$$

Language Modelling: bigrams

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Patrick was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Patrick})P(\text{late}|\text{was})P(\text{for}|\text{late})$$

Language Modelling: bigrams

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Patrick was late $\boxed{\text{for class}}$"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Patrick})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling: bigrams

How can we bu

You calculate each of these probabilities by simply counting the occurrences

$$P(class \mid for) = \frac{count(\textbf{for class})}{count(\textbf{for})}$$

probability

Let $\boldsymbol{X}$ = "Patrick was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Patrick})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling: bigrams

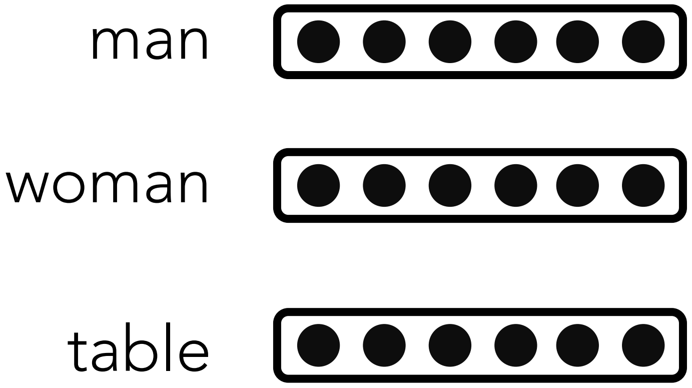
BIGRAM ISSUES?

- **Out-of-vocabulary** items are 0 → kills the overall probability
- Always need more context (e.g., trigram, 4-gram), but **sparsity** is an issue (rarely seen subsequences)
- **Storage** becomes a problem as we increase window size
- No semantic information conveyed by counts (e.g., vehicle vs car)

Language Modelling: neural networks

IDEA: Why not neural networks?!

First, each word is represented by a word embedding (e.g., vector of length 200)



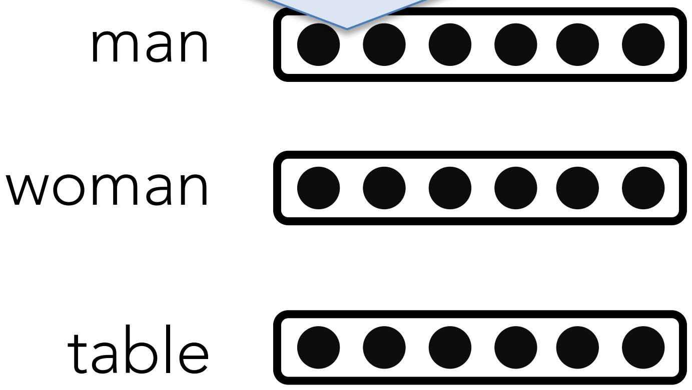
Language Modelling: neural networks

IDEA: Let's use

First, each wo

(e.g., vector of

- Each circle is a specific *floating point* scalar
- Words that are more *semantically similar* to one another will have **embeddings** that are proportionally similar, too
- We can *use pre-existing* word embeddings that have been trained on gigantic corpora



Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, RNNs/LSTMs +ELMo

Seq2Seq

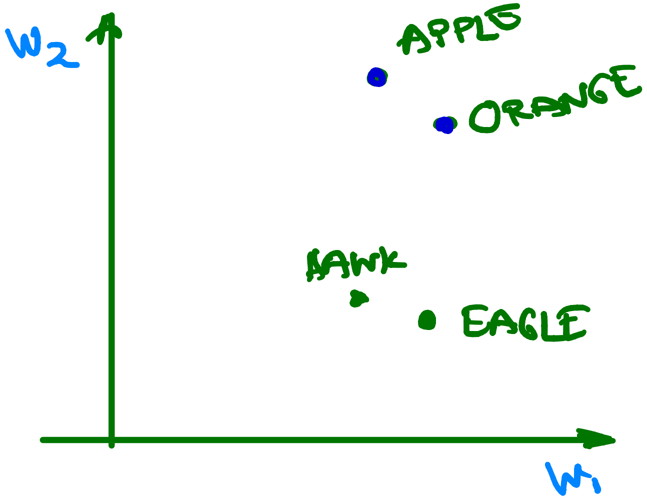
The basics idea

	attributes		
APPLES			
ORANGES			
EAGLE			

Similarity of

Sim(APPLE, ORANGE)

> Sim(APPLE, EAGLE)



How do we measure similarity?

If APPLE is represented by a <u>VECTOR</u>

$\text{cosine}(W^{ORA}, W^{APPLE}) > \text{cosine}(W^{APP.}, W^{EAGLE})$

word $\rightarrow \{w_1, w_2, \ldots, w_N\}$ ← vector that represents the word.

$$W = \begin{bmatrix} w^1_1 & w^1_2 & \cdots & w^1_N \\ w^2_1 & & & w^2_N \\ \vdots & & & \\ w^{|V|}_1 & & & w^{|V|}_N \end{bmatrix}$$

one line per word

LOOKING FOR W ← embedding.

SUCH THAT :

A: SIMILAR WORDS ARE NEAR-BY IN THIS SPACE

B: EVERY RAW HAS A SYMANTIC MEANING

$W^{KING} - W^{MAN} + W^{WOMAN} \sim= W^{QUEEN}$

HOW TO TRAIN (FIND W)

IT HAS BEEN A WONDERFUL DAY

A: EACH WORD HAS AN INDEX

e.g 13 22 512 1032 5 2

W =

B: GOAL IS TO PREDICT $P(x_t | x_{t-1}, ..)$

IT

HAS → trained LM: $\boxed{\text{PREDICT THE NEXT WORD}}$ → "A"

BEEN

$$\begin{array}{cc} \begin{bmatrix} 0.01 \\ 0.23 \\ \vdots \\ \\ \\ \end{bmatrix} & \begin{array}{c} \text{ANT} \\ \vdots \\ \vdots \\ \\ \\ \end{array} \end{array}$$

probabilities

$P(A | \text{"IT HAS BEEN"})$ →

more details:

FIND THE W vector / AKA LOOK UP EMBEDDINGS

1. IT $\rightarrow [w_1^{IT} \ldots\ldots w_N^{IT}]$

HAS $\rightarrow [\quad]$

BEEN $\rightarrow [\quad]$

2. CALCULATE PREDICTION IN $\hat{P}(y=k)$

it has been a wonderful morning.

use two (or more words) to predict next one

and I have a lots and lots of text to train.

Loss function $CE(\hat{P}(y=k), y)$

WAIT: LET'S LOOK BOTH WAYS (LANGUAGES WORK LIKE THIS)

it has been a wonderful morning.

INPUT	OUTPUT
It, has, a, wonderful	been
has, been, wonderful morning	a
	⋮

training set

THIS IS CALLED A CONTINUOUS BAG OF WORDS

ALTERNATIVELY: (SKIP GRAM)

it has been a wonderful morning.

Training set:

input	output
been	it
been	has
been	a
been	wonderful
a	it

Two extra details:

1 output $\hat{P} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ p_{|V|} \end{bmatrix}$ } vector of $|V| \sim 15k$ elements.

$$\text{LOSS} = -\sum \sum_{k} \mathbb{1}(y=k) \log \hat{p}$$ $\Rightarrow$ too expensive

Instead

been $\rightarrow$ $\boxed{\quad\quad}$ $\rightarrow$ 'it'

been $\rightarrow$
it $\rightarrow$ $\boxed{\quad\quad}$ $\rightarrow$ YES / NO

problem we do not have "no" examples
randomly sample negative examples
PROBLE SOLVED

2. TWO EMBEDDINGS:

EMBEDDING

number of words in voc

$$\begin{bmatrix} W_1^1 & \cdots & W_N^1 \\ W_1^2 & & W_N^2 \\ \vdots & & \vdots \\ W_1^{|V|} & & W_N^{|V|} \end{bmatrix}$$

embedding dimension

CONTEXT

$$\begin{bmatrix} C_1^1 & \cdots & C_N^1 \\ C_1^2 & & C_N^2 \\ \vdots & & \vdots \\ C_1^{|V|} & & C_N^{|V|} \end{bmatrix}$$

context dimension

same as emb. dim

WE THROW THIS AWAY ONCE TRAINING IS DONE

Hyper-Parameters:

* WINDOW SIZE:

Smaller window = similar words

Largest window = related words

default: 5

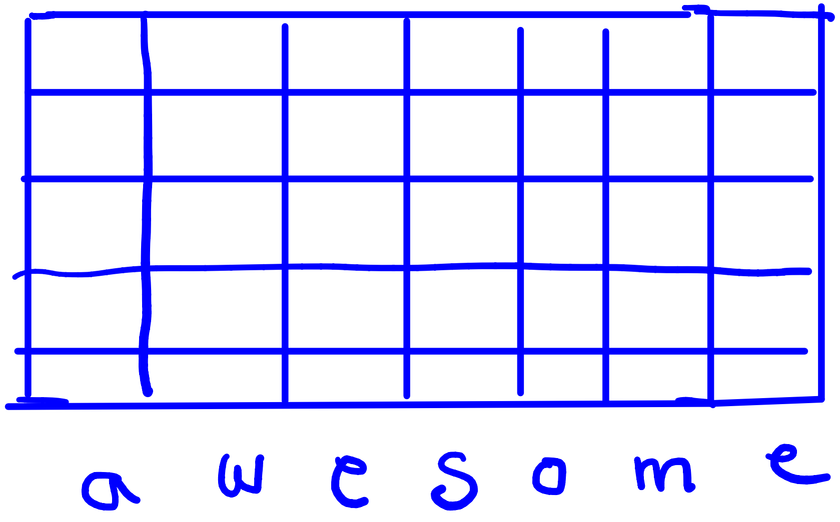
* negative examples: 2-5 enough but 5-20 is recommended

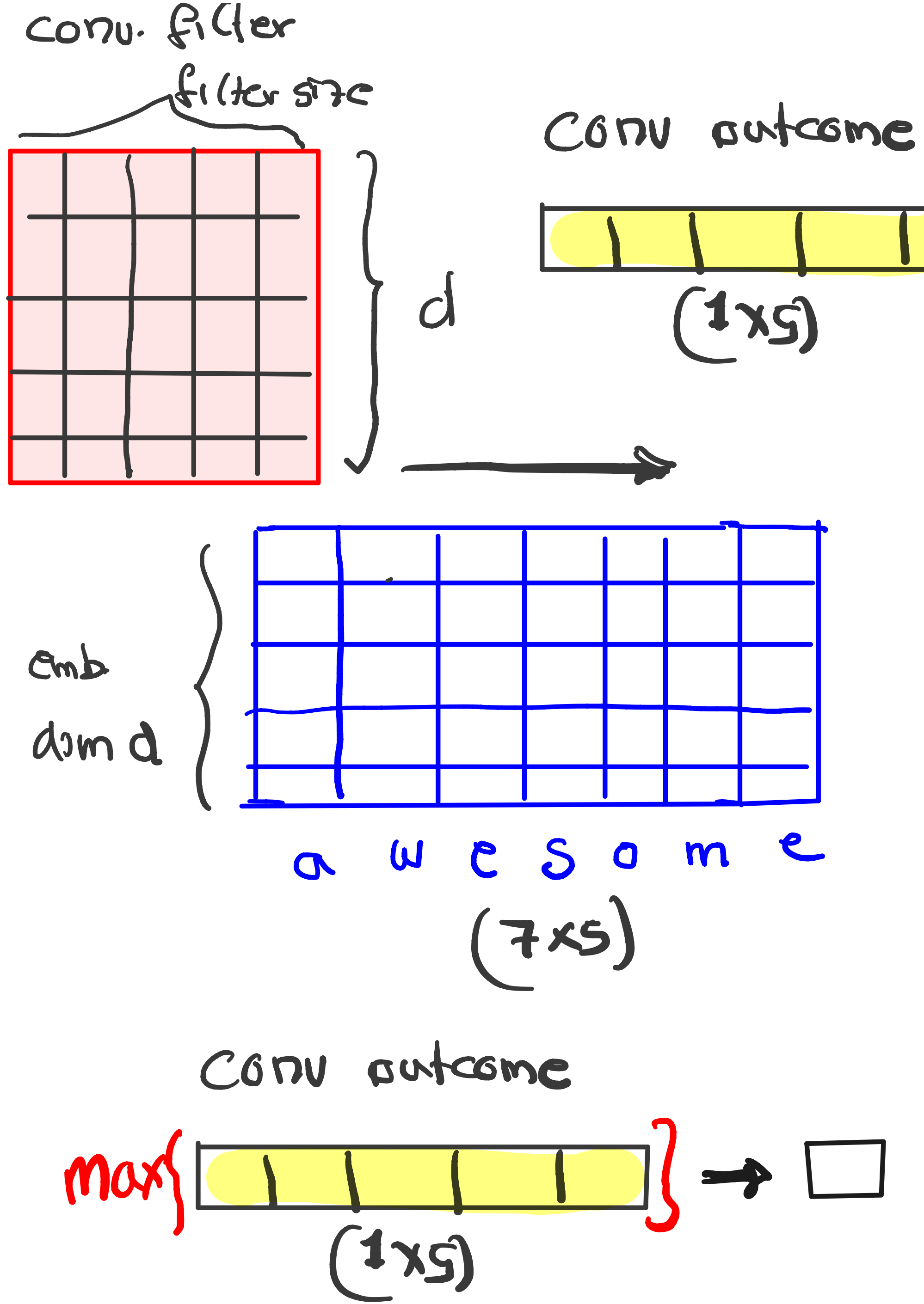
CHARACTER EMBEDDINGS

Instead of word embedding another way is character embedding :

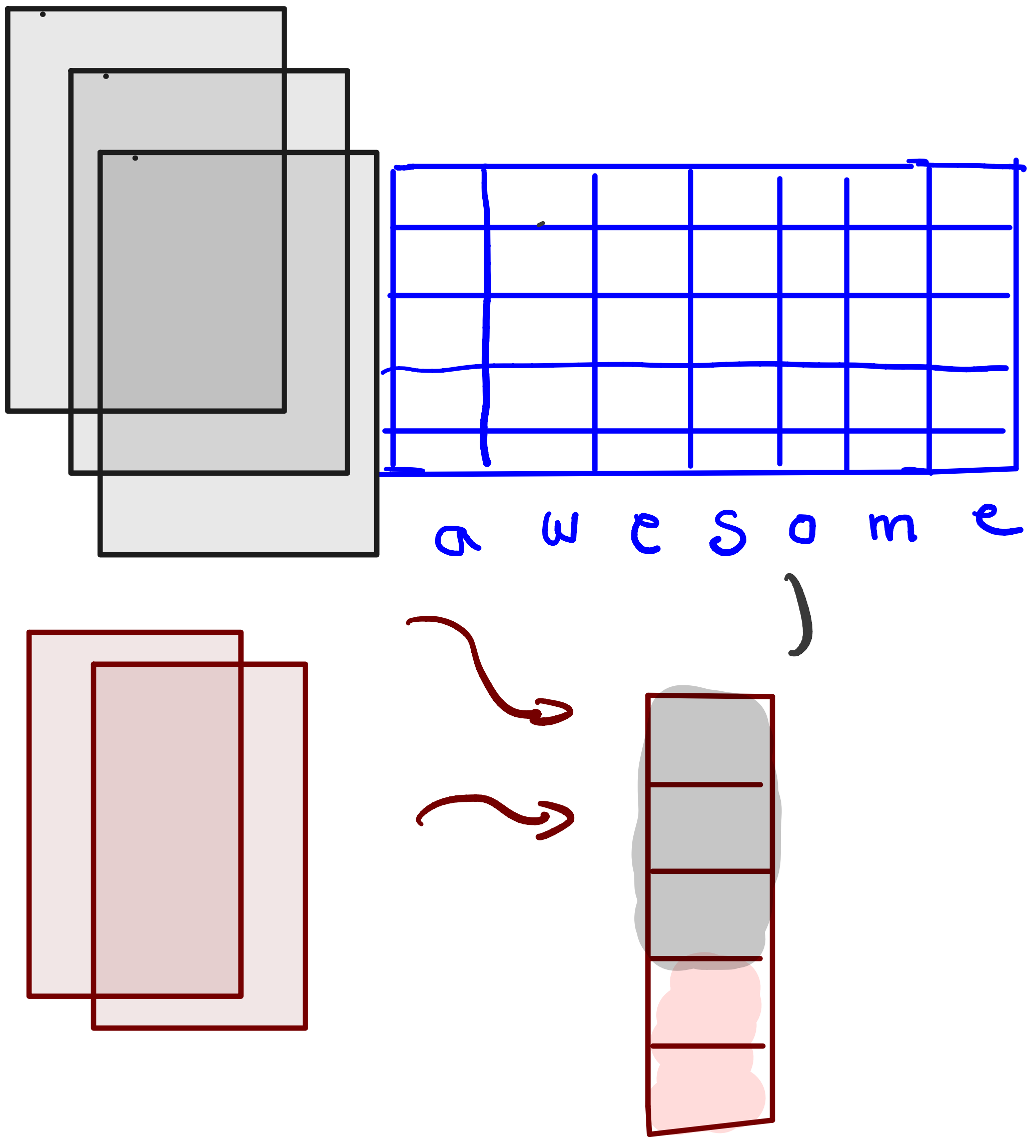
For every word 'awesome' create a matrix $C \in \mathbb{R}^{d \times l}$ ← length of word

character emb.

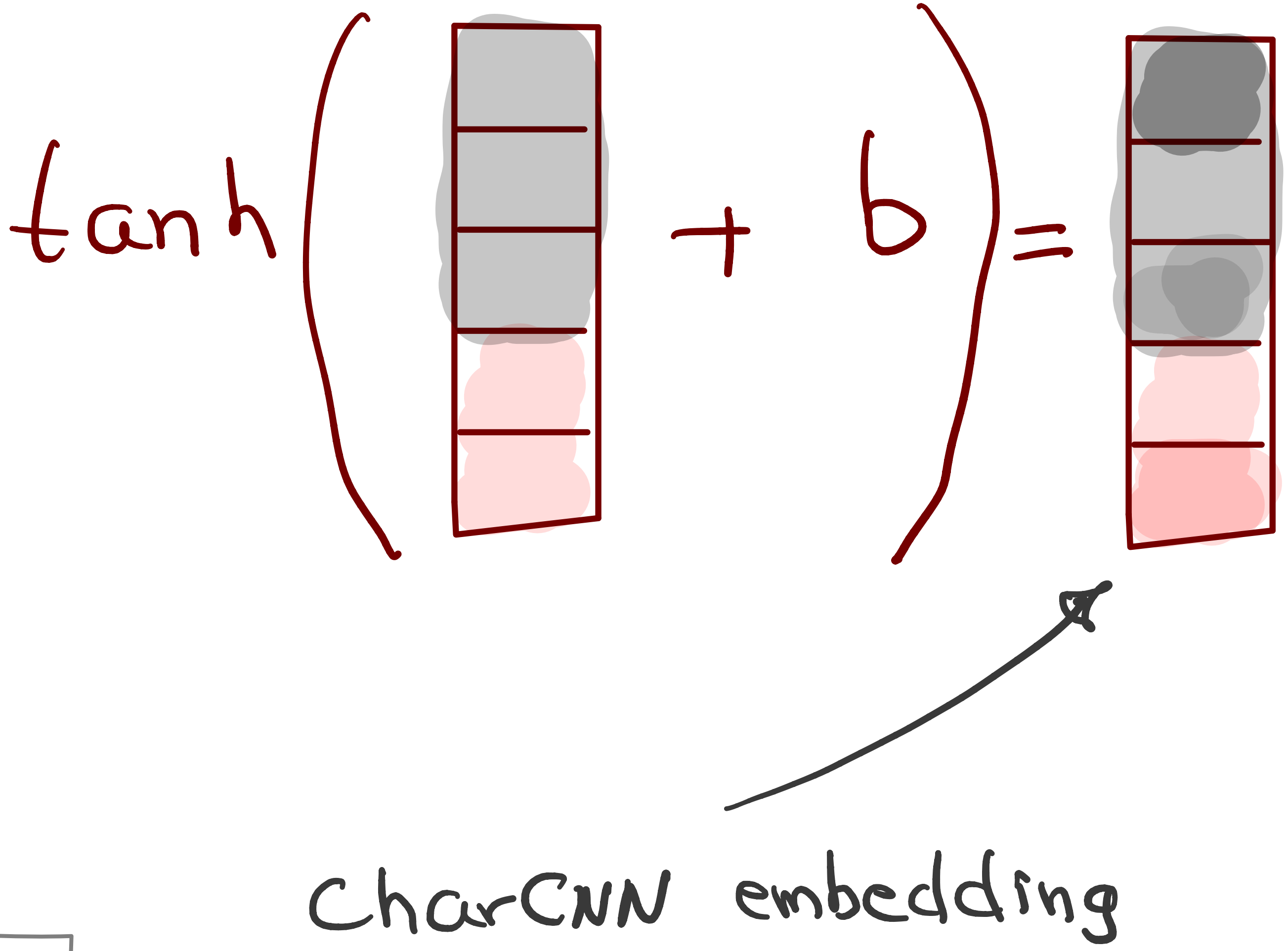



conv. filter
filter size
d
conv outcome
(1x5)
emb
dim d
a w e s o m e
(7x5)
conv outcome
max
(1x5)

multiple filters, each produce one number.



Add Bias and Apply Activation



Usage of embeddings

- the pre-trained word2vec and other embeddings (such as GloVe) are used everywhere in NLP today.
- the ideas have been used elsewhere as well. AirBnB and Anghami model sequences of listings and songs using word2vec like techniques.
- Alibaba and Facebook use word2vec and graph embeddings for recommendations and social network analysis.

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

- **FFNN**, RNNs/LSTMs +ELMo
- Seq2Seq

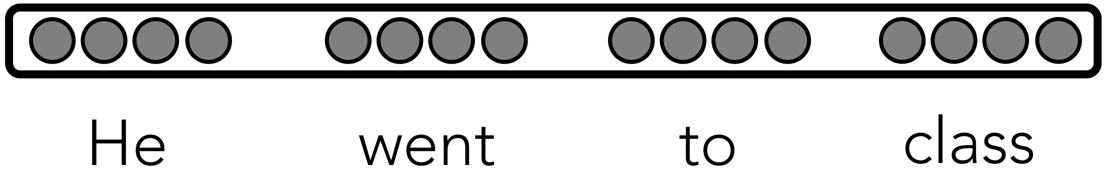
Language Modelling: neural networks

How can we use these embeddings to build a LM?

Remember, we only need a system that can estimate:

$$P(\underbrace{x_{t+1}}_{\text{next word}} | \underbrace{x_t, x_{t-1}, \dots, x_1}_{\text{previous words}})$$

Example input sentence

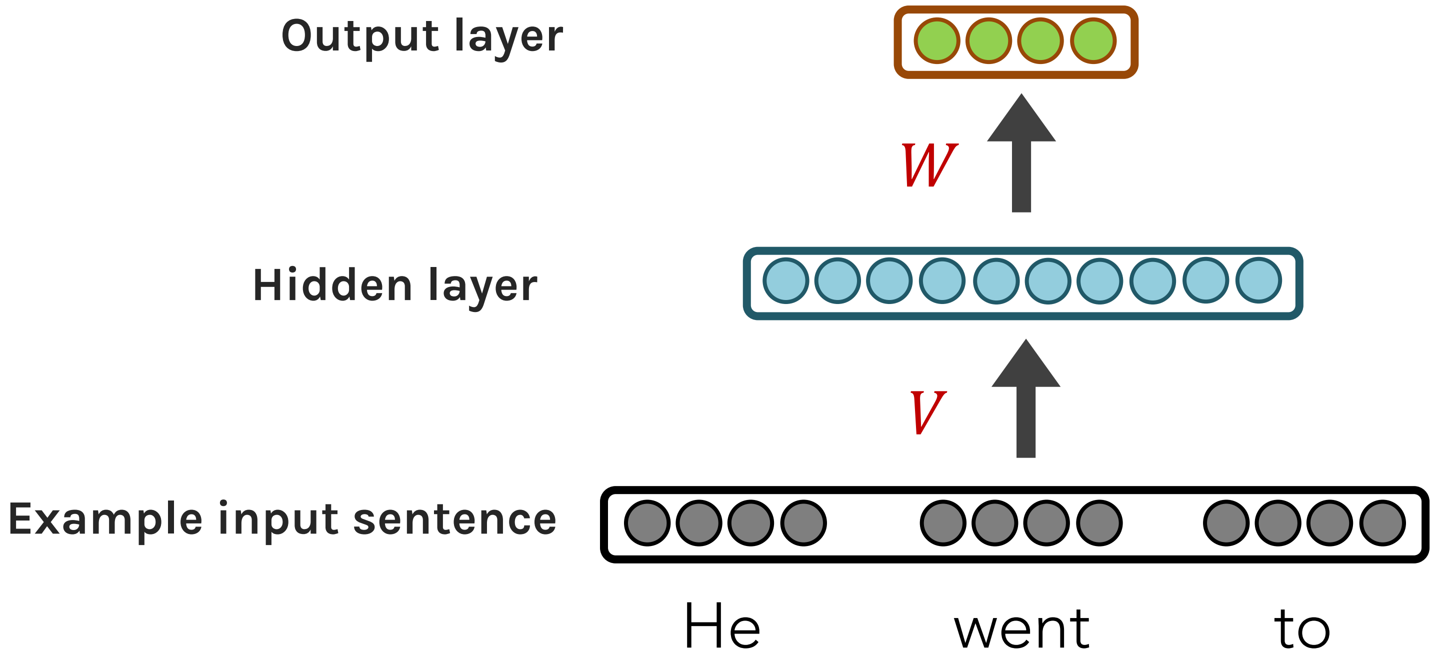


Language Modelling: Feed-forward Neural Net

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

Ex. *He went to class after breakfast.*

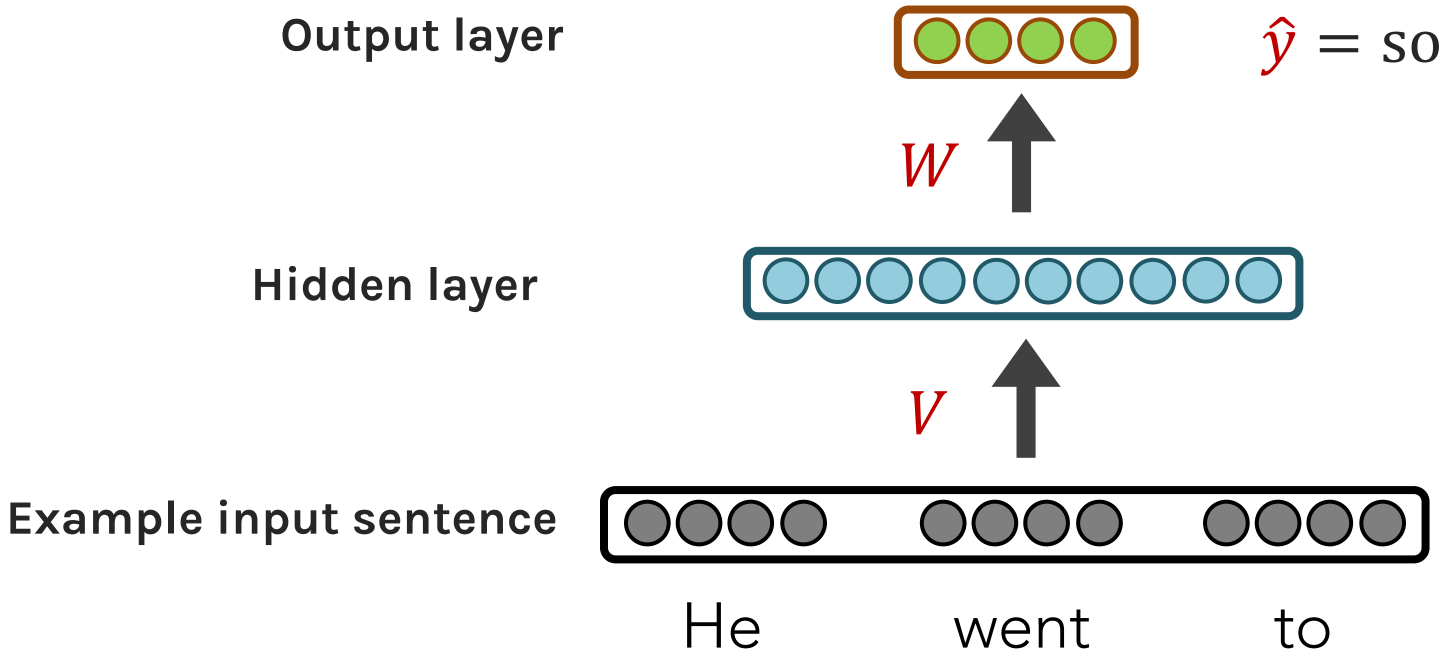


Language Modelling: Feed-forward Neural Net

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

Ex. *He went to class after breakfast.*



$$\hat{y} = \mathrm{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$$

$$h = f(Vx + b_1)$$

$$x = [x_1, x_2, x_3]$$

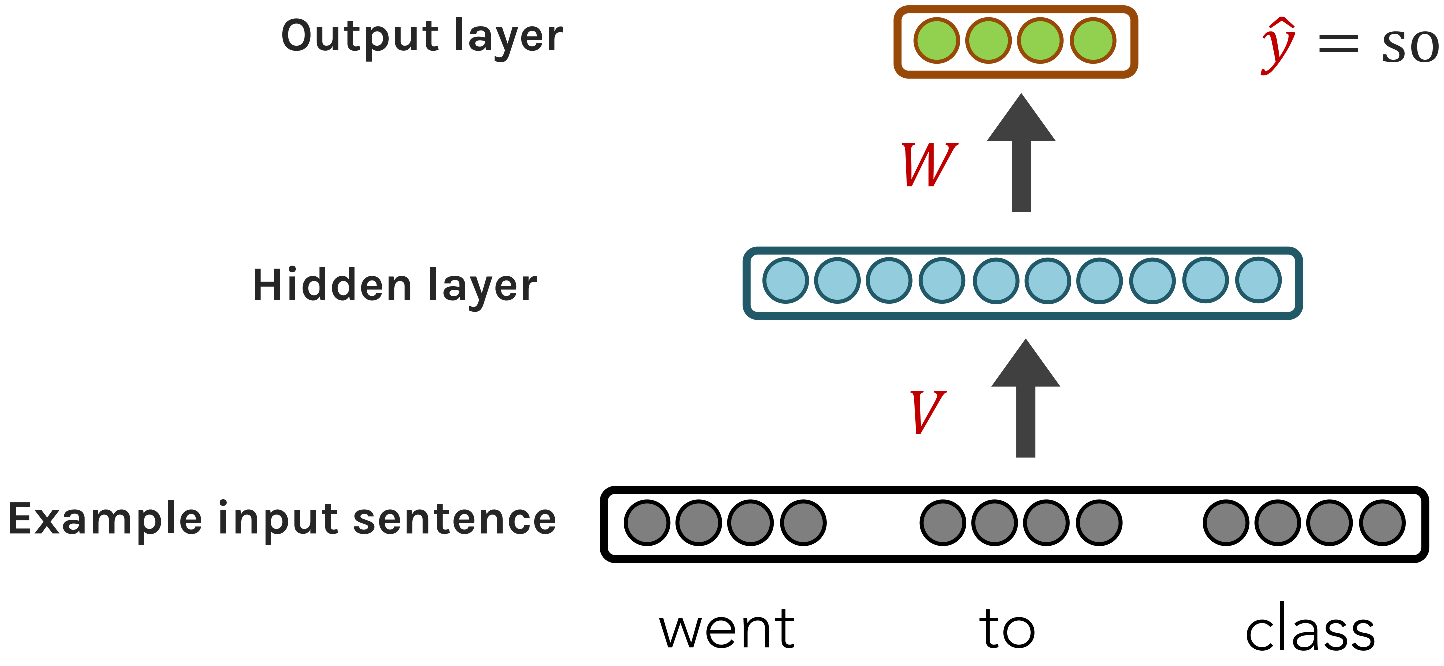
Concatenated word embeddings

Language Modelling: Feed-forward Neural Net

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

Ex. *He went to class after breakfast.*



$\hat{y} = \mathrm{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$

$h = f(Vx + b_1)$

$x = [x_1, x_2, x_3]$

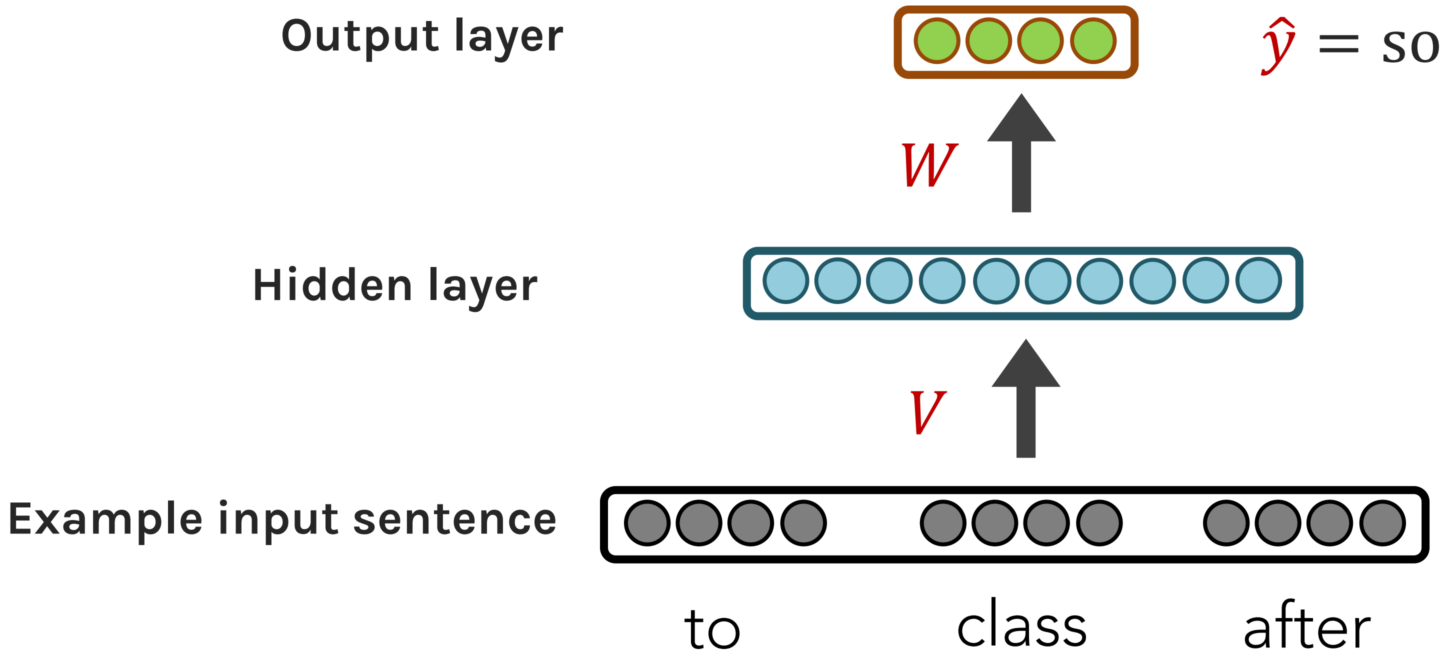
Concatenated word embeddings

Language Modelling: Feed-forward Neural Net

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

Ex. *He went to class after breakfast.*



$\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$

$h = f(Vx + b_1)$

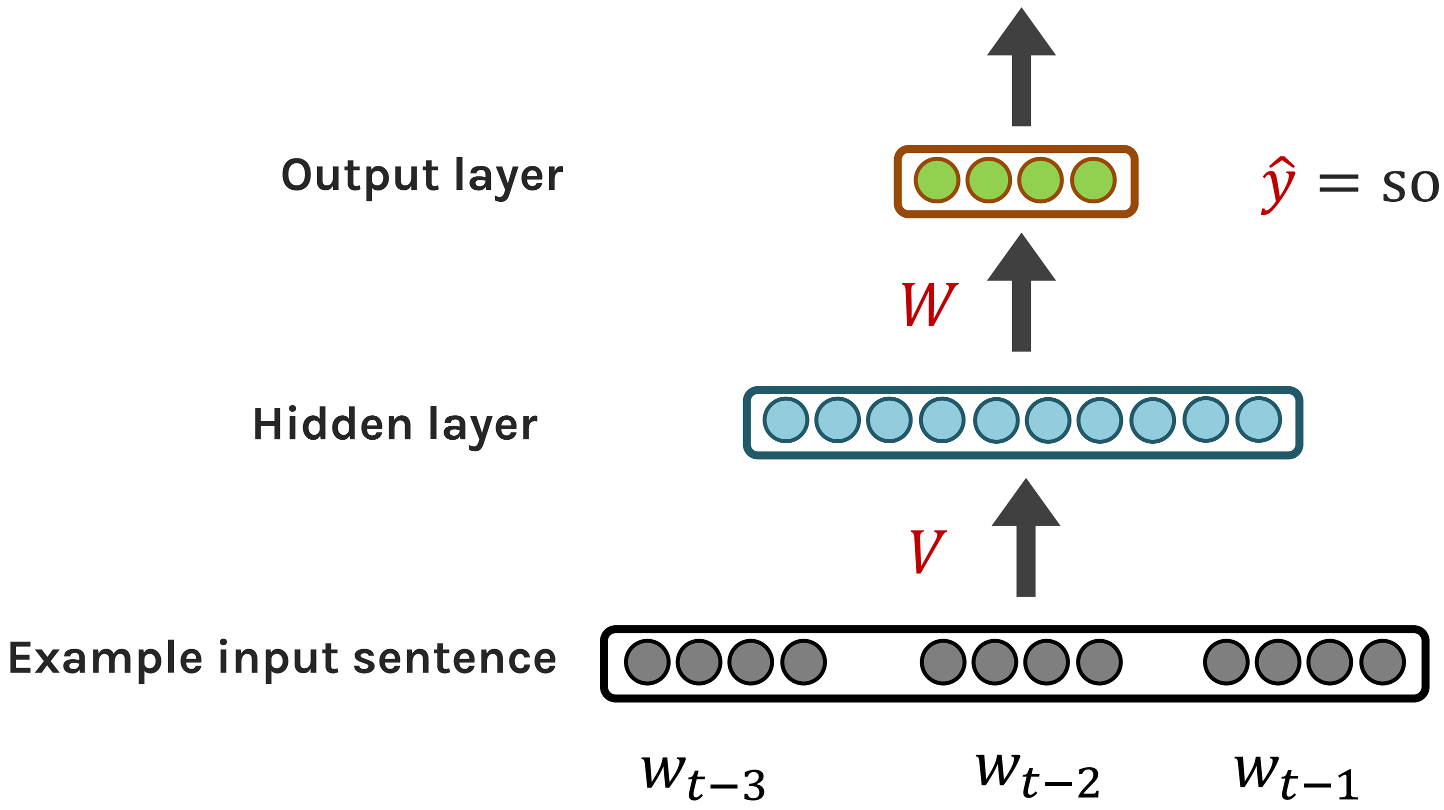
$x = [x_1, x_2, x_3]$

Concatenated word embeddings

Language Modelling: Feed-forward Neural Net

Training

$$L = CE(\hat{y}_t, y_t)$$



$$\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$$

$$h = f(Vx + b_1)$$

$$x = [x_1, x_2, x_3]$$

Concatenated word embeddings

Language Modelling: Feed-forward Neu

$\hat{y}$: are probabilities for each possible word in the vocabulary.

y: is a vector of 0's and 1.

Training

$$L = \sum_{t=3}^{t} CE(\hat{y}_t, y_t)$$

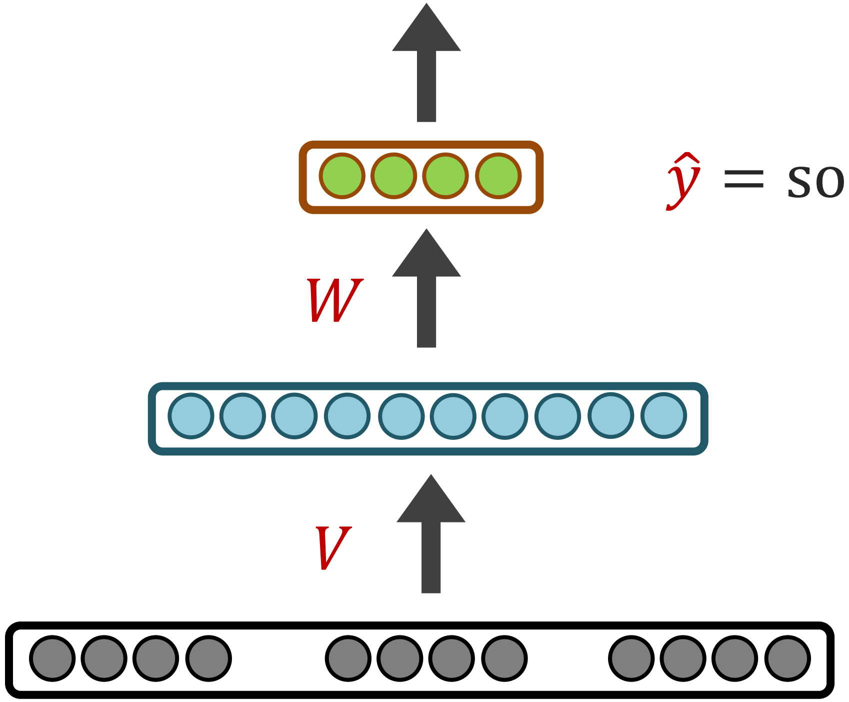
Output layer

$$\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$$

Hidden layer

$$h = f(Vx + b_1)$$

Example input sentence



$$x = [x_1, x_2, x_3]$$

Concatenated word embeddings

w_{t-3} w_{t-2} w_{t-1}

Language Modelling : Feed-forward Neural Net

FFNN STRENGTHS?

- No sparsity issues (it's okay if we've never seen a segment of words)
- No storage issues (we never store counts)

FFNN ISSUES?

- Fixed-window size can never be big enough. Need more context.
- Increasing window size adds many more weights
- The weights awkwardly handle word position
- No concept of time
- Requires inputting entire context just to predict one word

Language Modelling

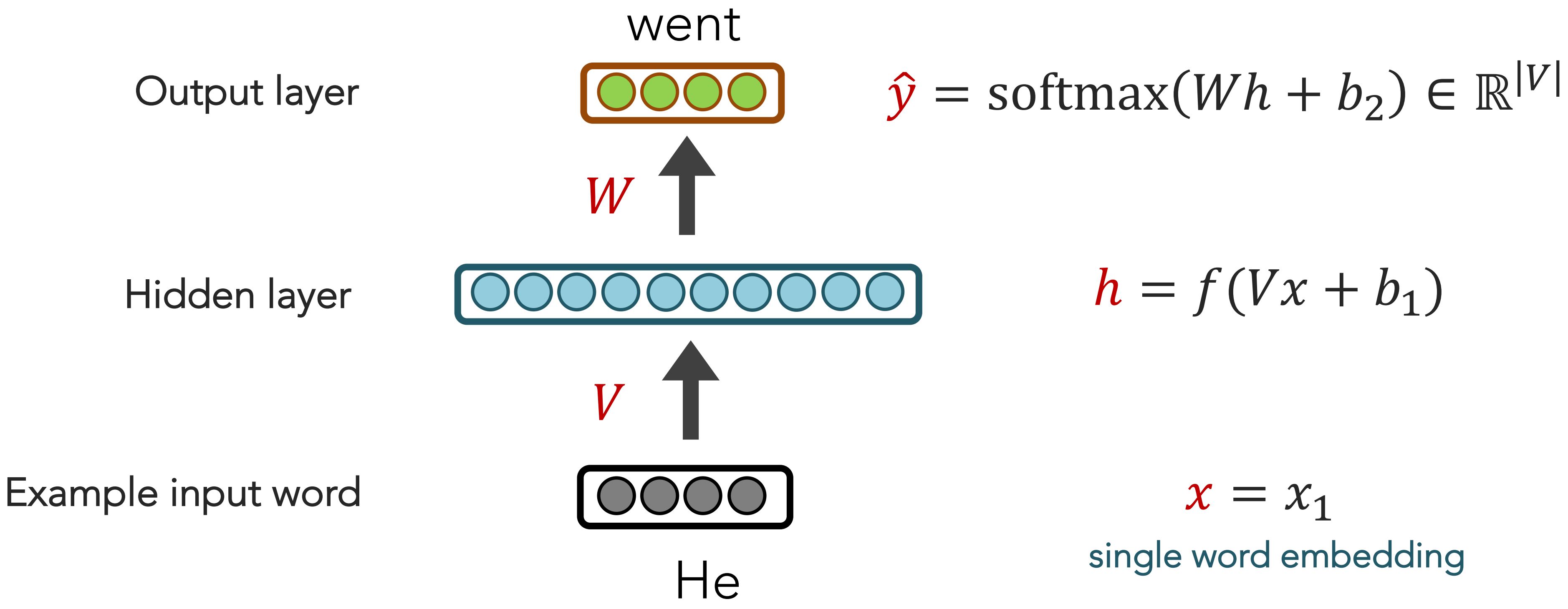
We especially need a system that:

- Has an “infinite” concept of the past, not just a fixed window
- For each new input, output the most likely next event (e.g., word)

Language Modelling

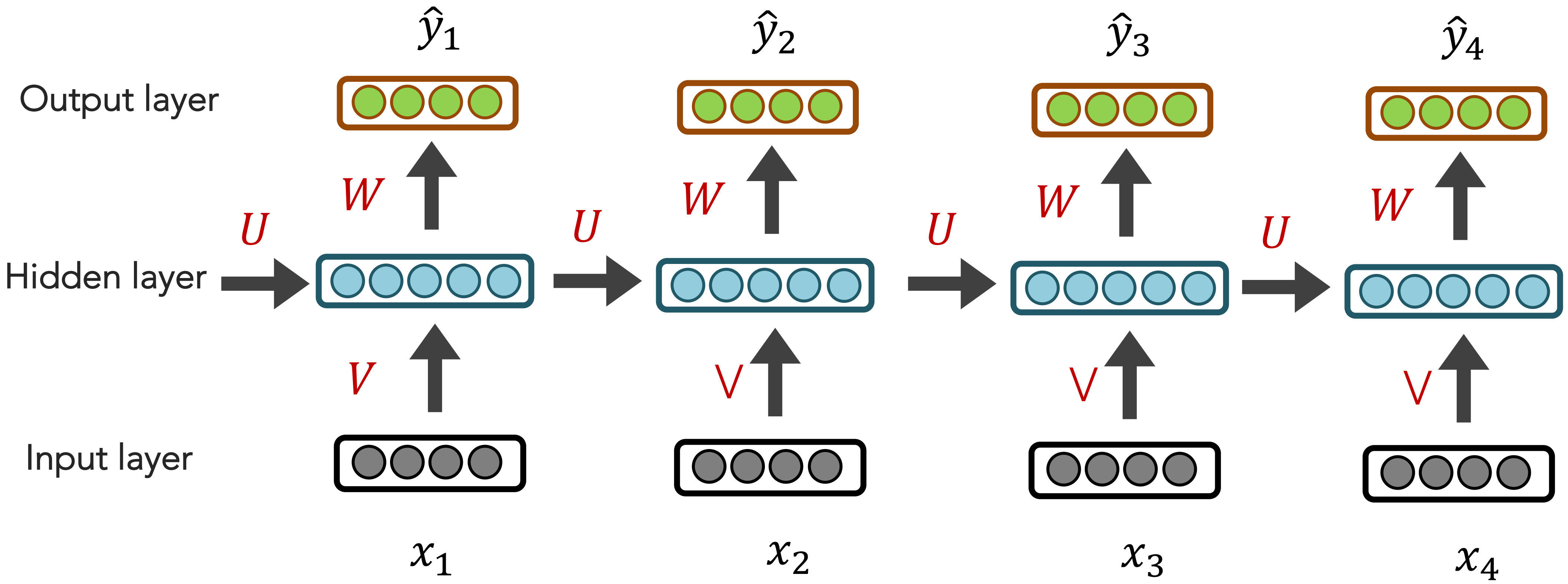
IDEA: for every individual input, output a prediction

Let's use the previous hidden state, too



Language Modelling: RNNs

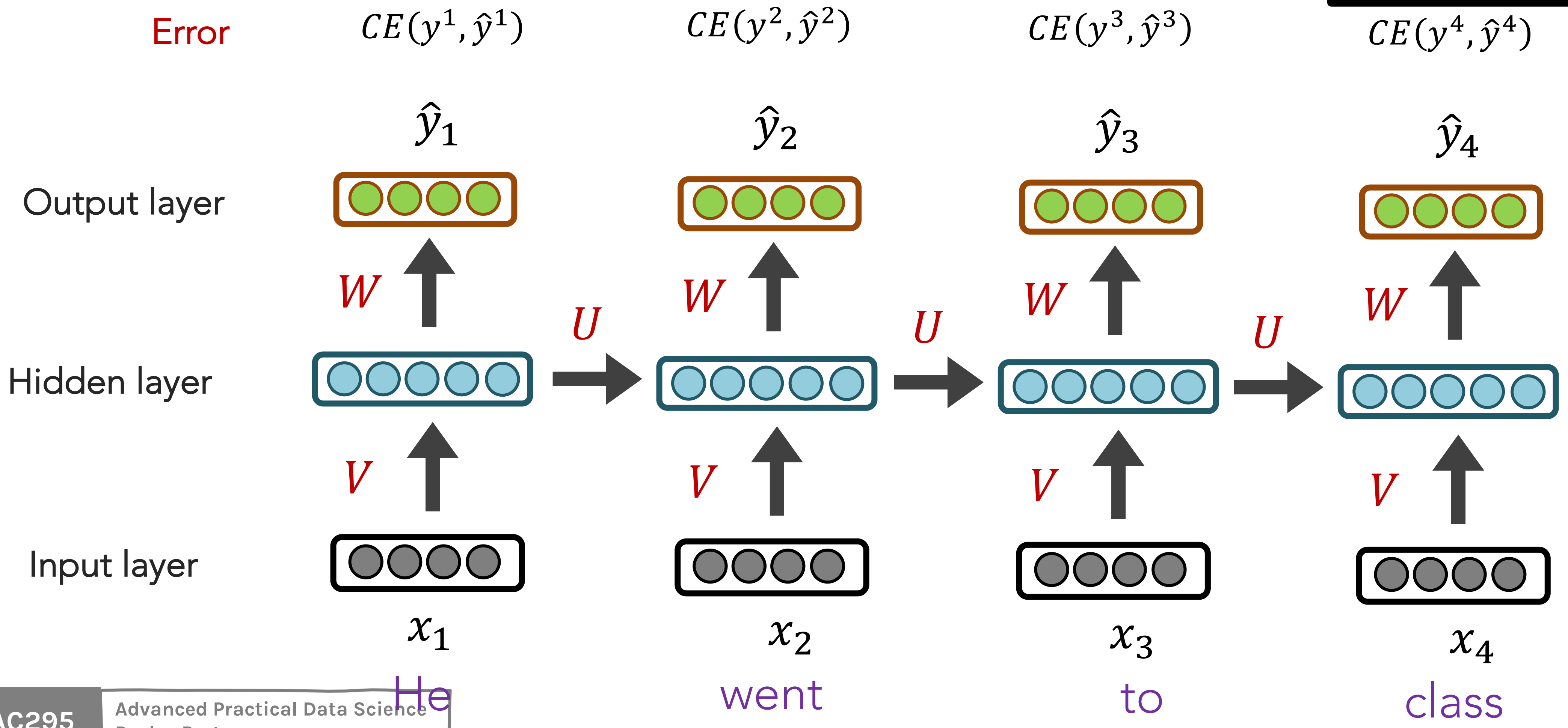
Neural Approach #2: Recurrent Neural Network (RNN)



RNN (review)

Training Process

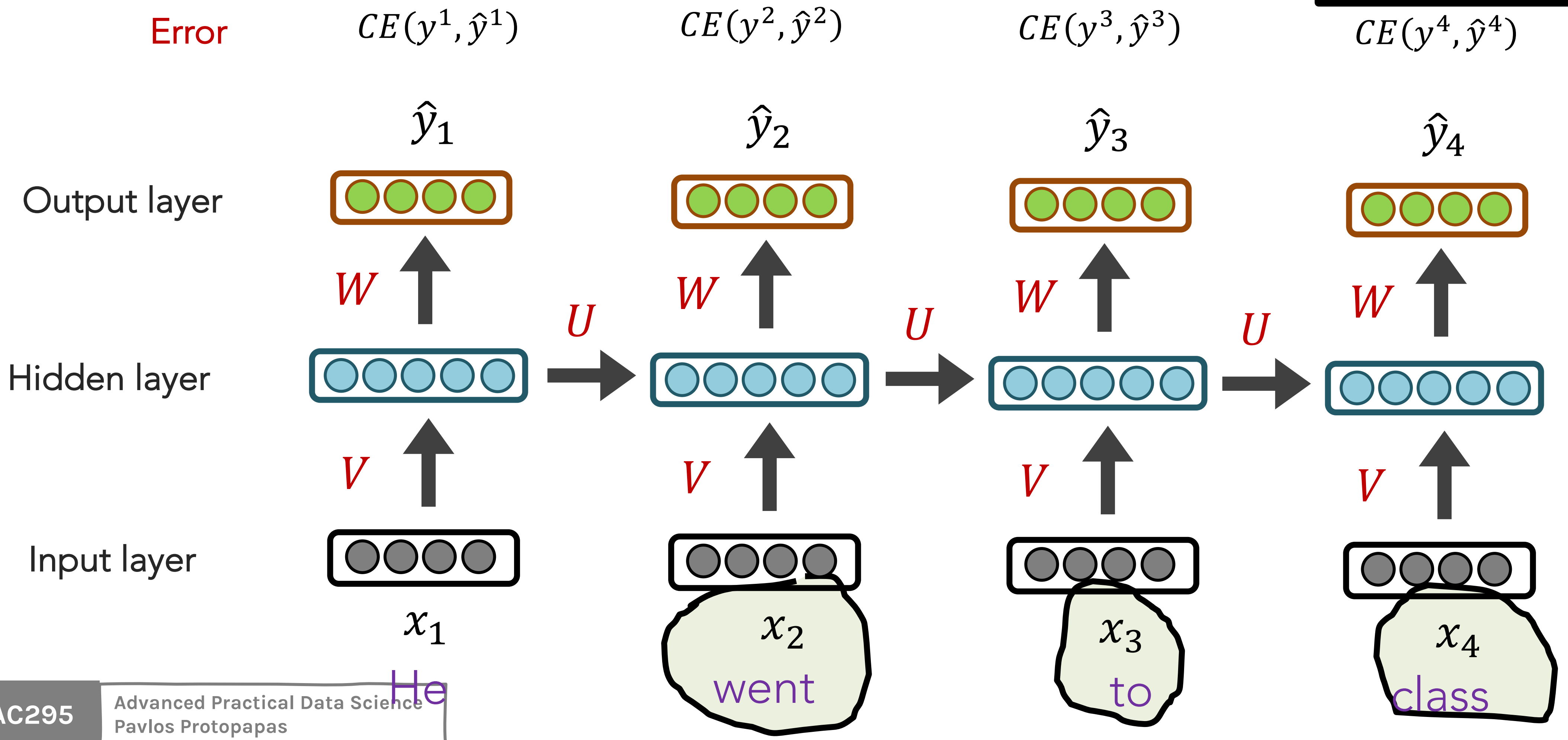
$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$



RNN (review)

Training Process

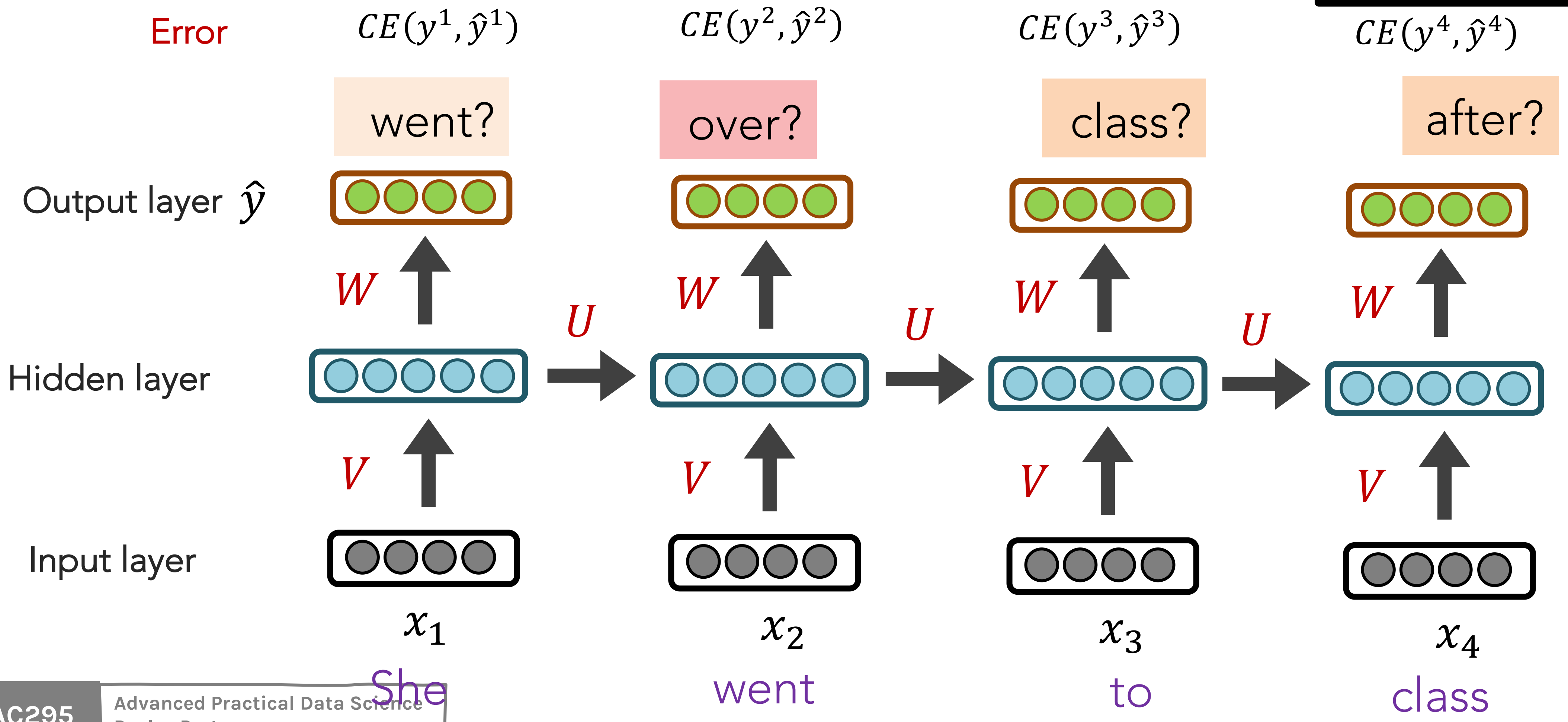
$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$



RNN (review)

Training Process

$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$

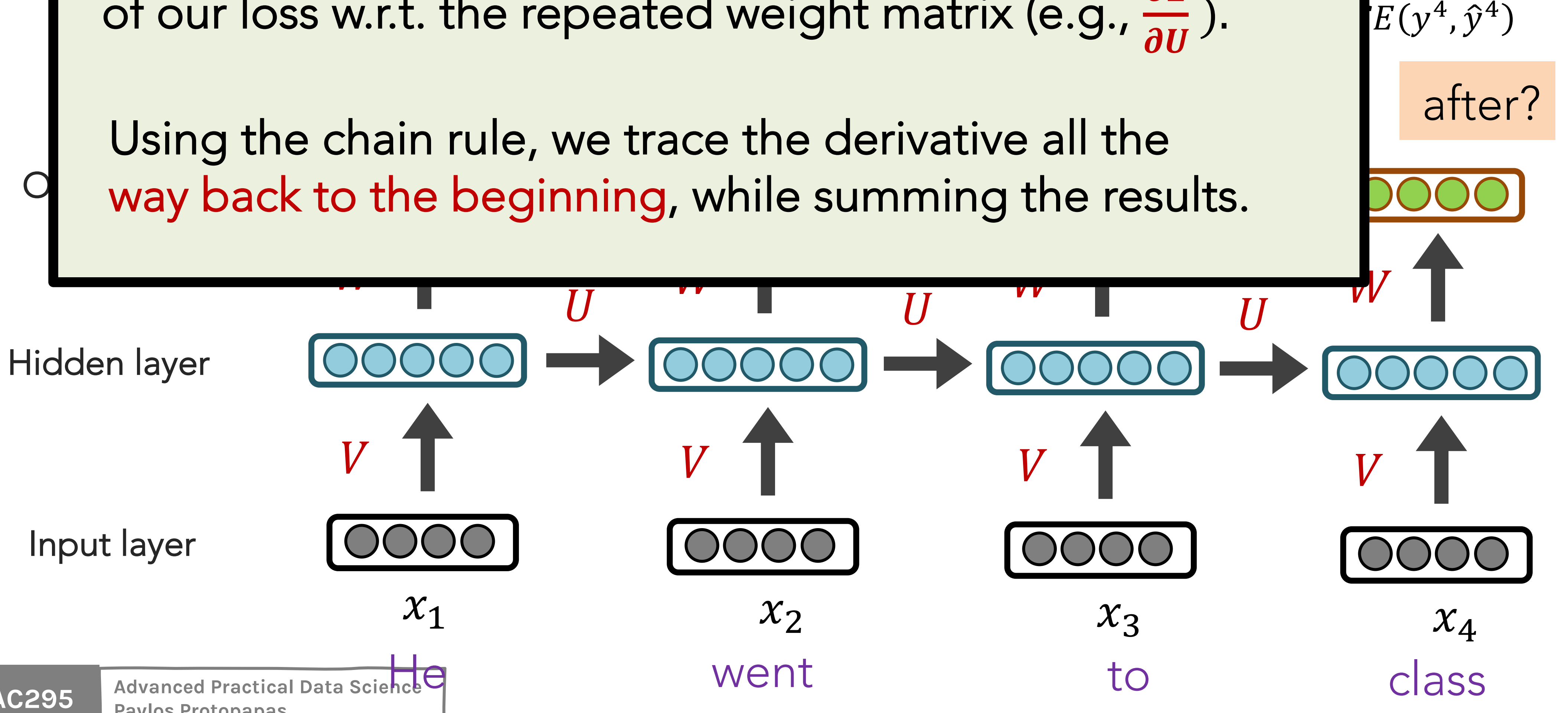


RNN (review)

To update our weights (e.g. $\boldsymbol{U}$), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial U}$).

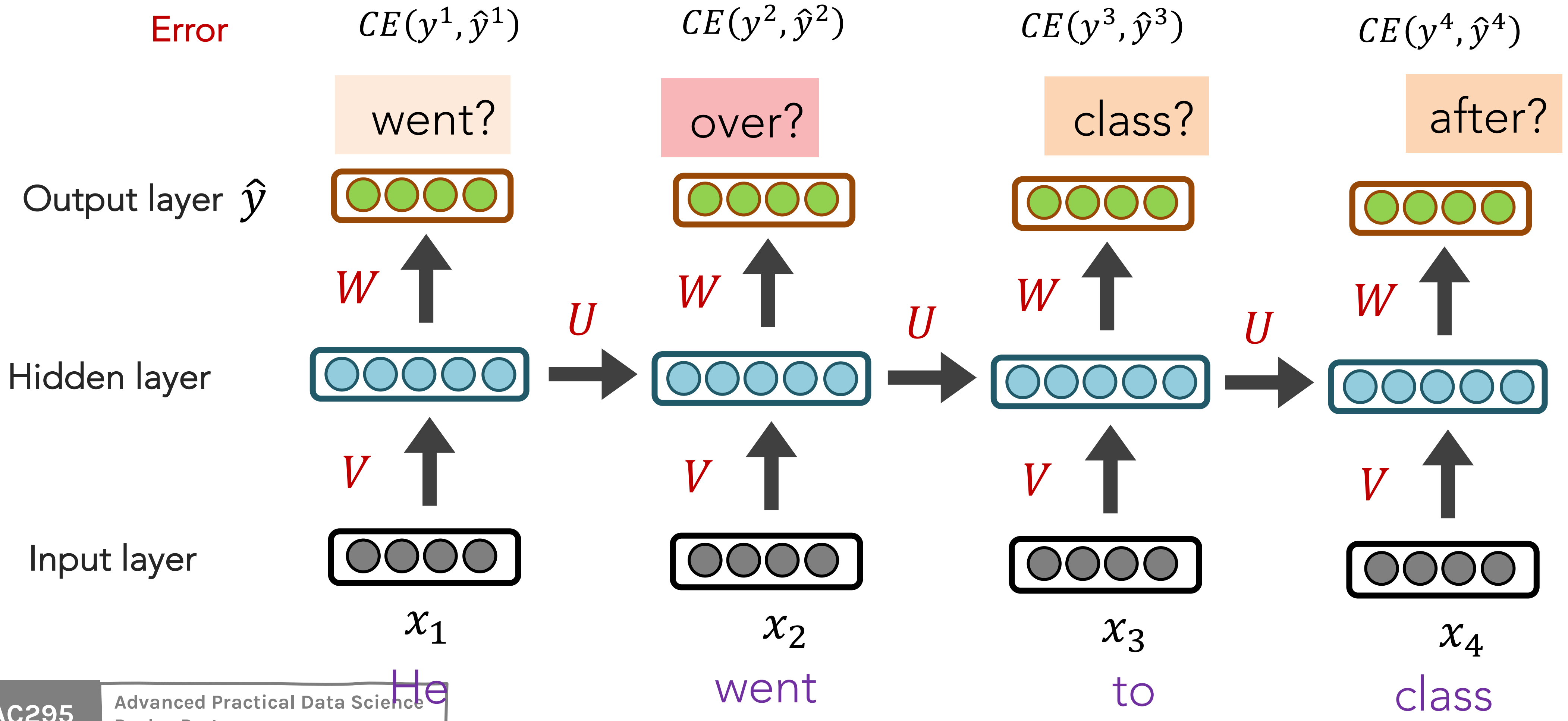
Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.

$E(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$



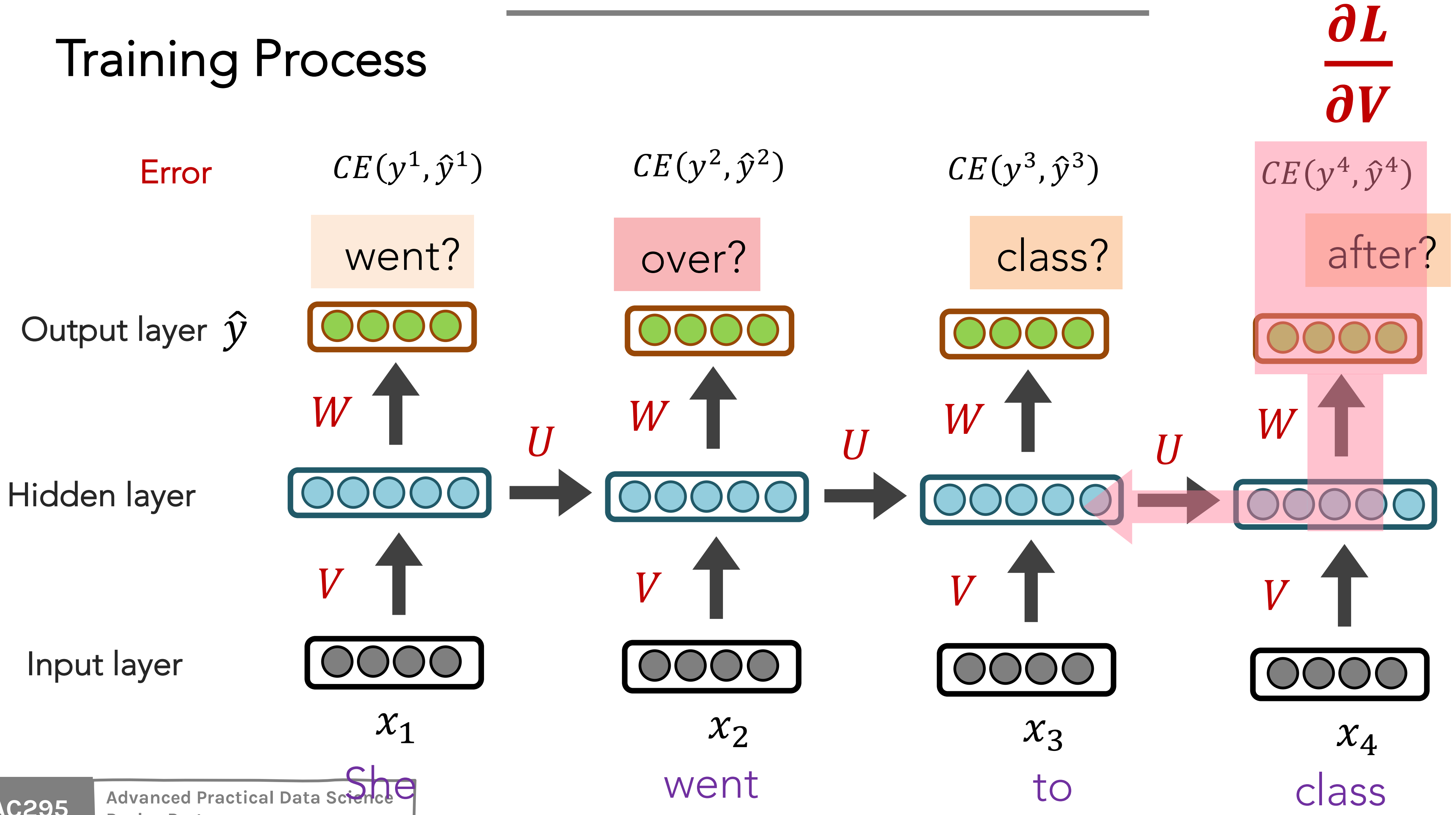
RNN (review)

Training Process



RNN (review)

Training Process



RNN (review)

- This backpropagation through time (BPTT) process is **expensive**
- Instead of updating after every timestep, we tend to do so every T steps (e.g., every sentence or paragraph)
- This isn't equivalent to using only a window size T (a la n-grams) because we still have 'infinite memory'

RNN: Generation

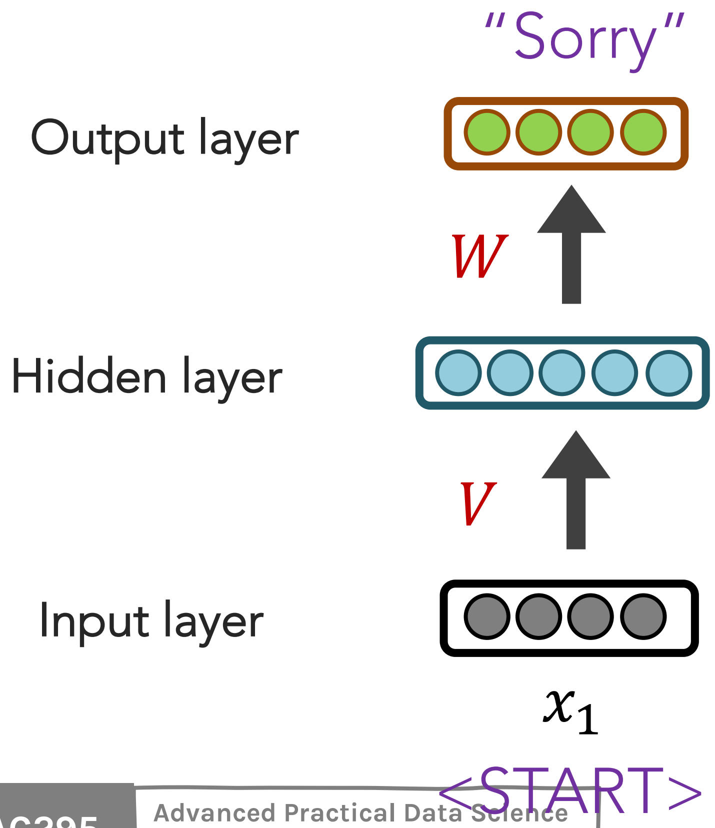
We can generate the most likely next event (e.g., word) by sampling from $\hat{y}$

Continue until we generate <EOS> symbol.

RNN: Generation

We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

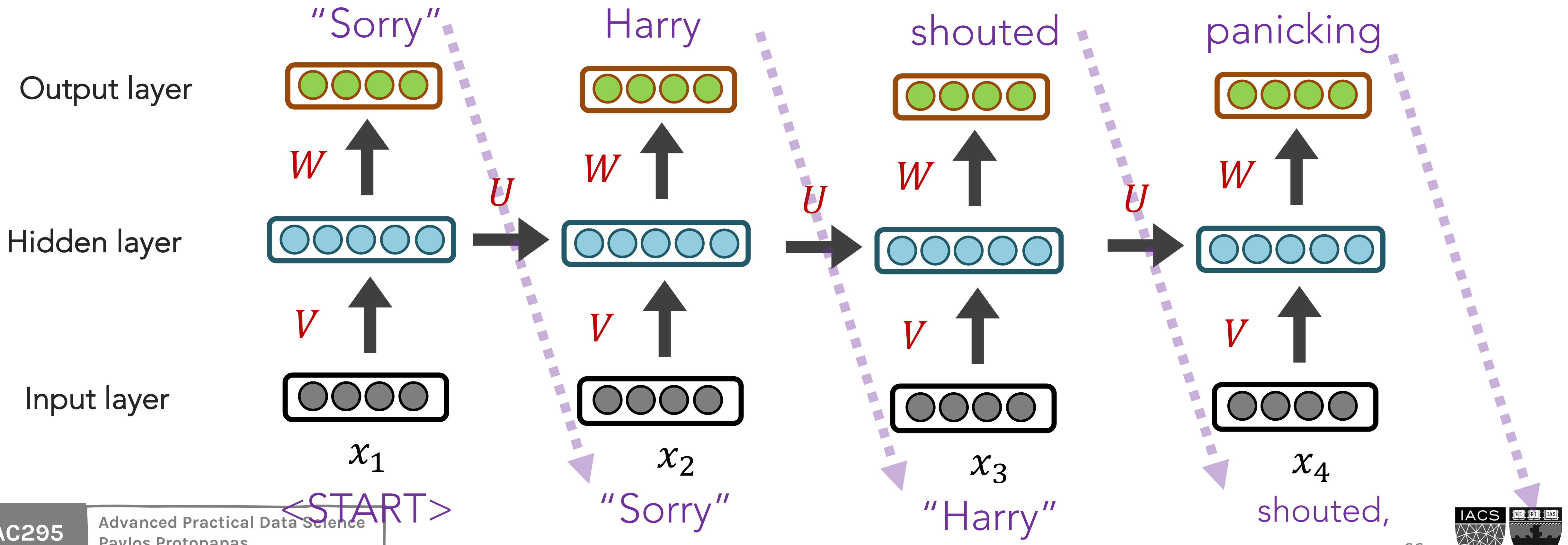
Continue until we generate **<EOS>** symbol.



RNN: Generation

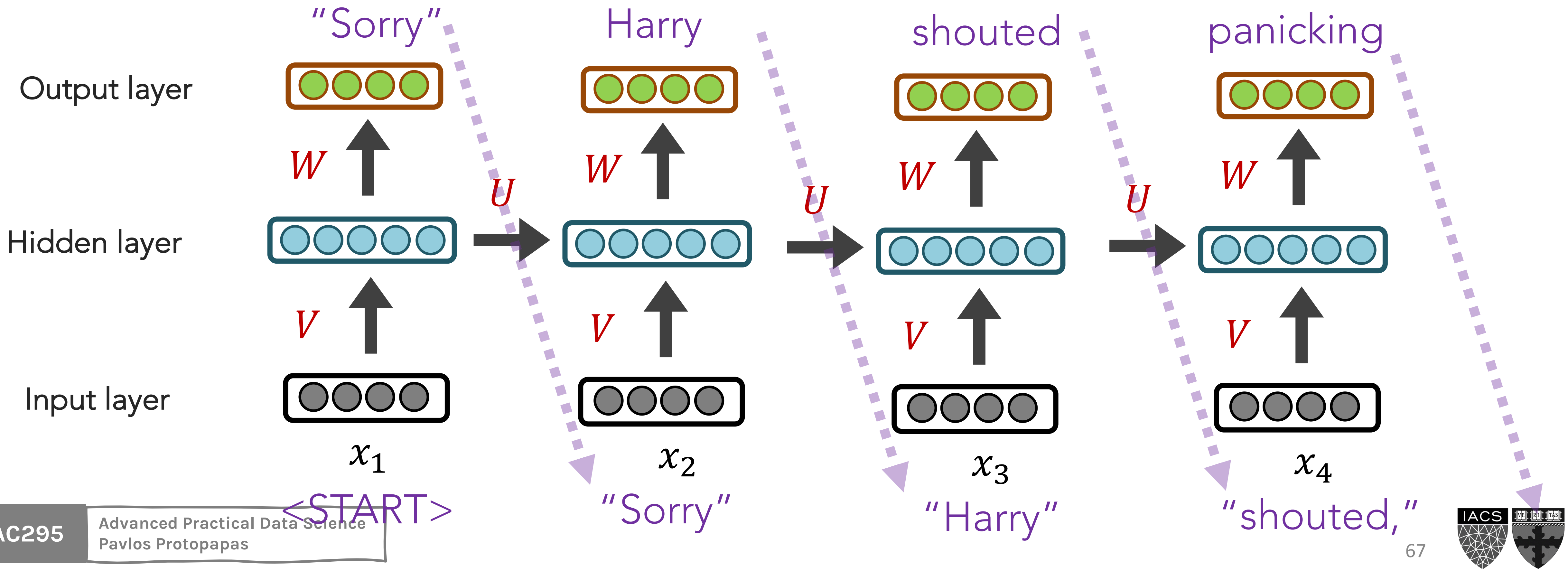
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

Continue until we generate **<EOS>** symbol.



RNN: Generation

NOTE: the same input (e.g., **"Harry"**) can easily yield different outputs, depending on the context (unlike FFNNs and n-grams).



RNN: Generation

When trained on Harry Potter text, it generates:

“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

RNN: Generation

When trained on recipes

```
     Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
     Yield: 6 Servings

     2 tb Parmesan cheese -- chopped
     1 c  Coconut milk
     3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

RNNs: Overview

RNN STRENGTHS?

- Can handle infinite-length sequences (not just a fixed-window)
- Has a "memory" of the context (thanks to the hidden layer's recurrent loop)
- Same weights used for all inputs, so word order isn't wonky (like FFNN)

RNN ISSUES?

- Slow to train (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- Has trouble actually making use of long-range context

RNNs: Vanishing and Exploding Gradients (review)

To address RNNs' finnicky nature with long-range context, we turned to an RNN variant named **LSTMs (long short-term memory)**

But first, let's recap what we've learned so far

Outline

NLP Tasks

Transfer Learning in NLP

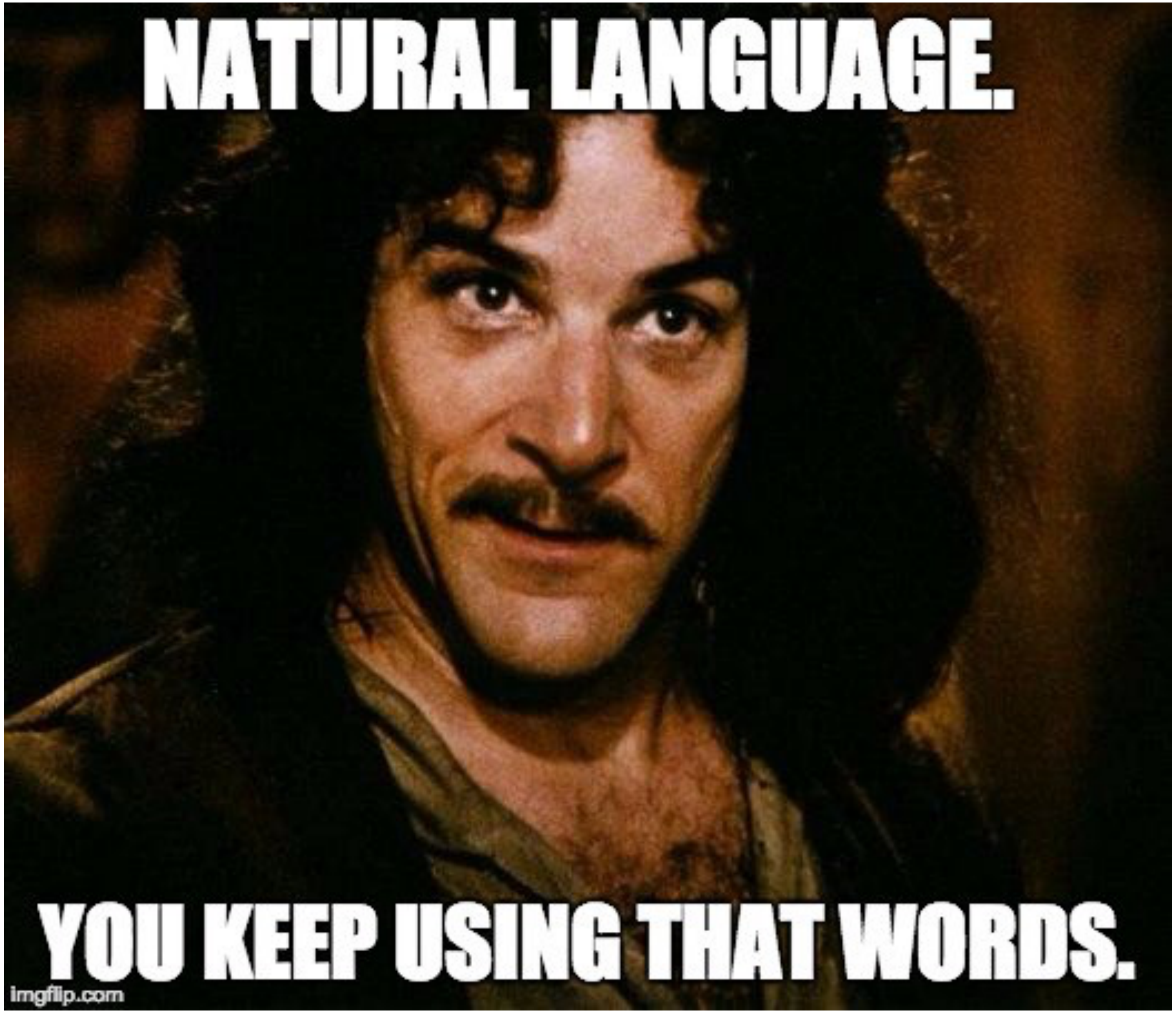
Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, **RNNs/LSTMs** +ELMo

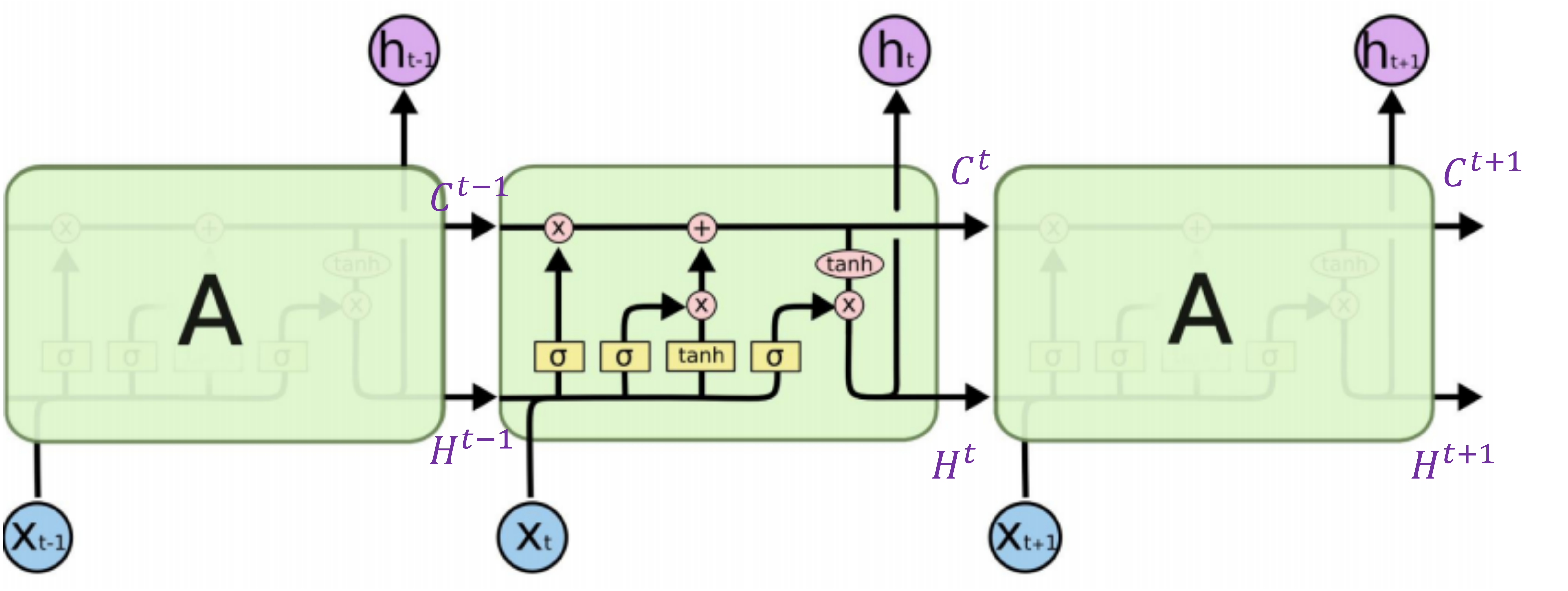
Seq2Seq


NATURAL LANGUAGE.
YOU KEEP USING THAT WORDS.
imgflip.com

Long short-term memory (LSTM)

- A type of RNN that is designed to better handle **long-range dependencies**
- In ”vanilla” RNNs, the hidden state is perpetually being rewritten
- In addition to a traditional **hidden state h**, let’s have a dedicated **memory cell c** for long-term events. More power to relay sequence info.

Inside an LSTM Hidden Layer



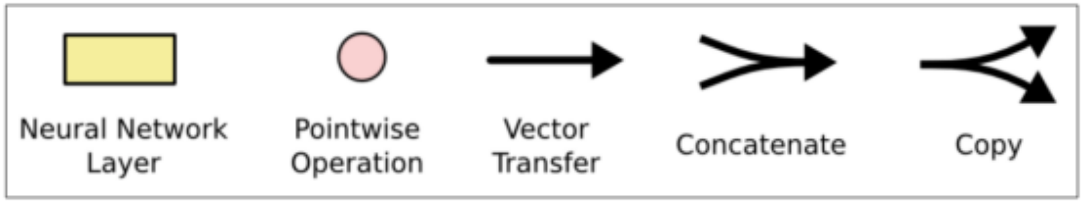
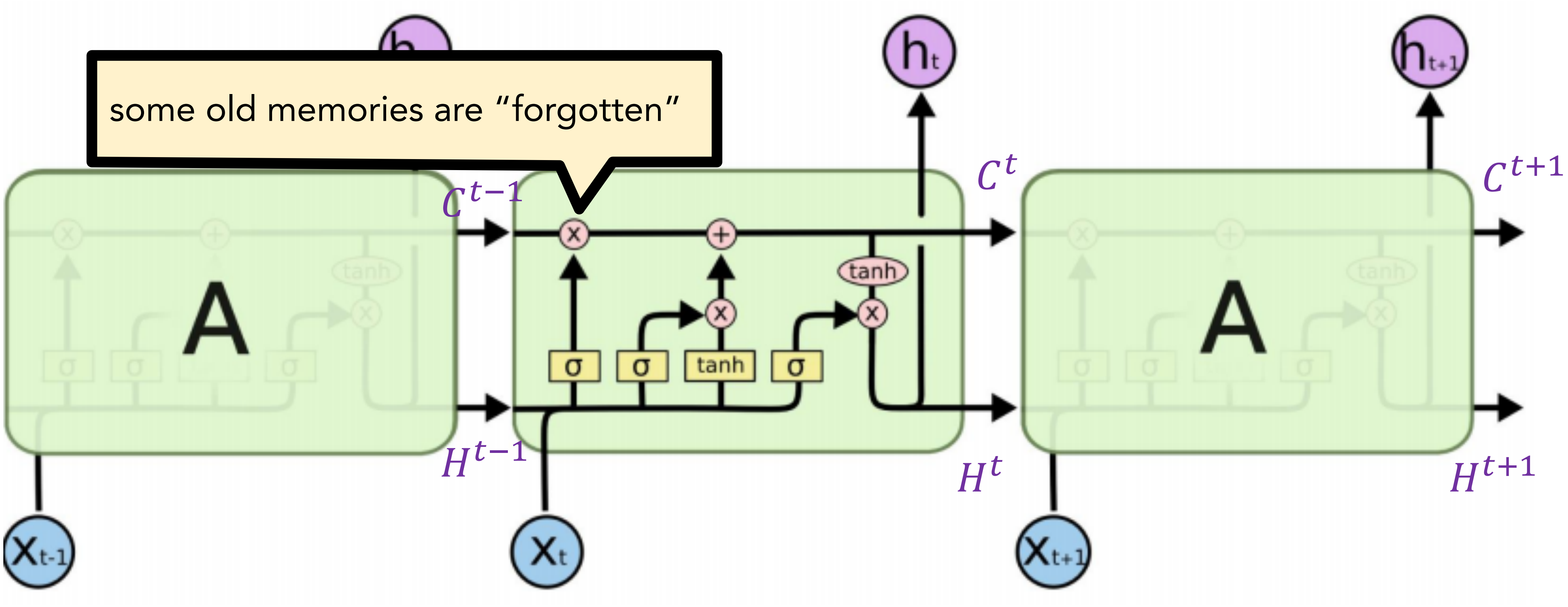


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer



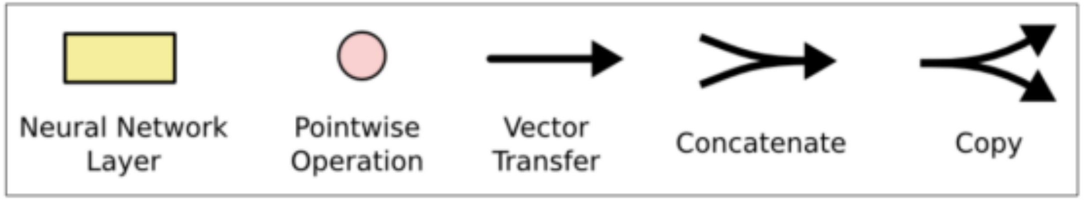


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

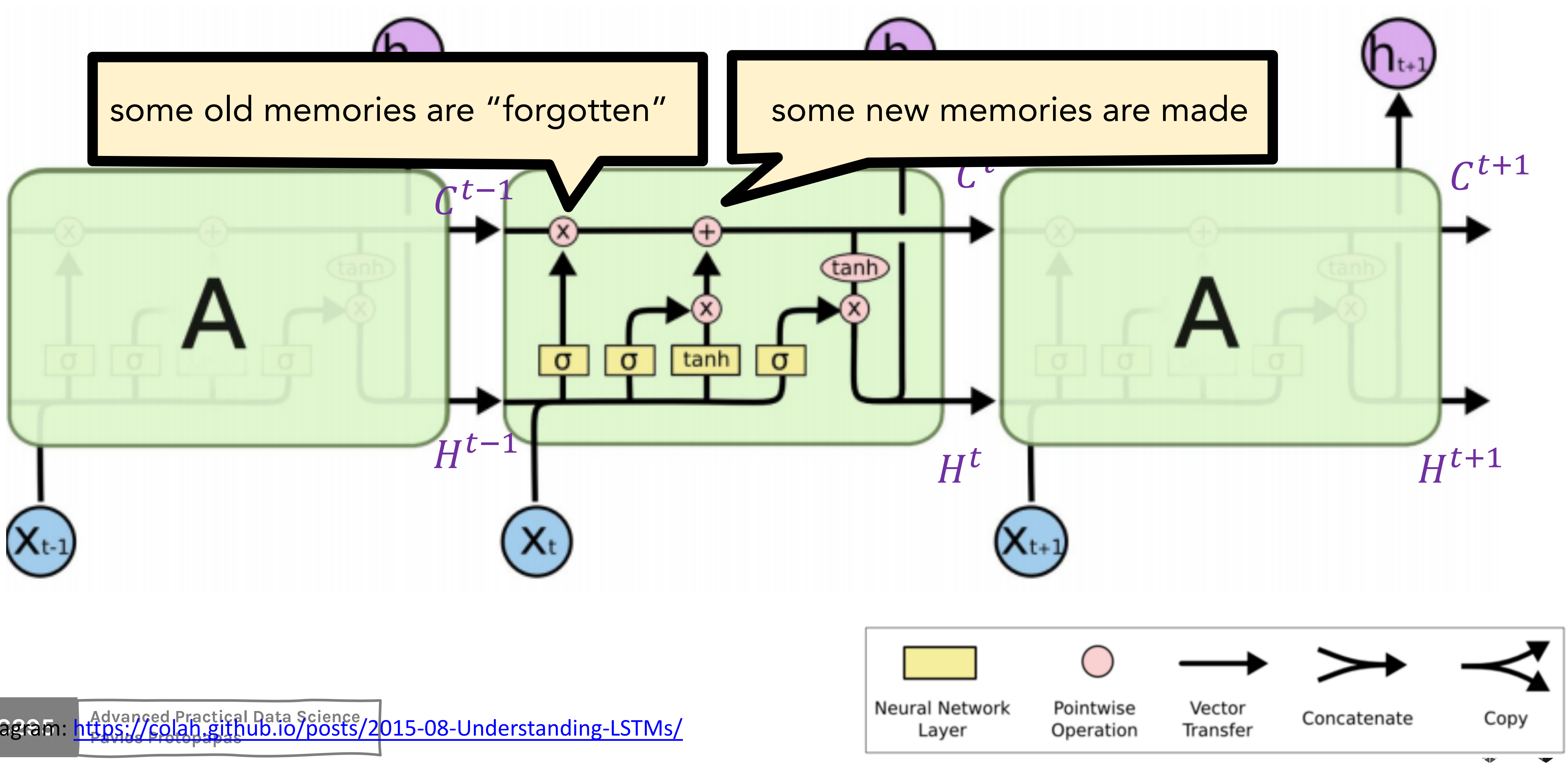


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

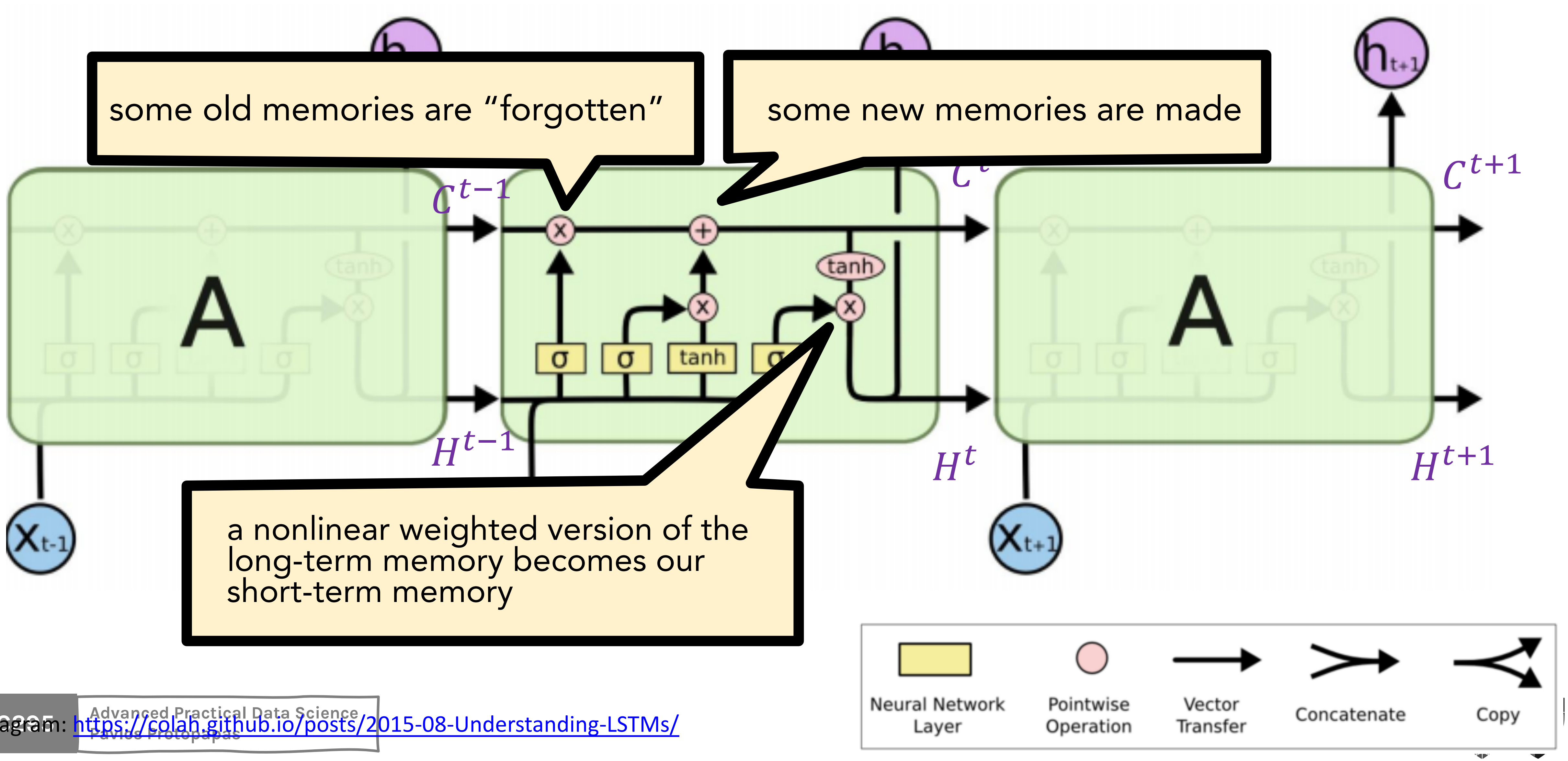


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

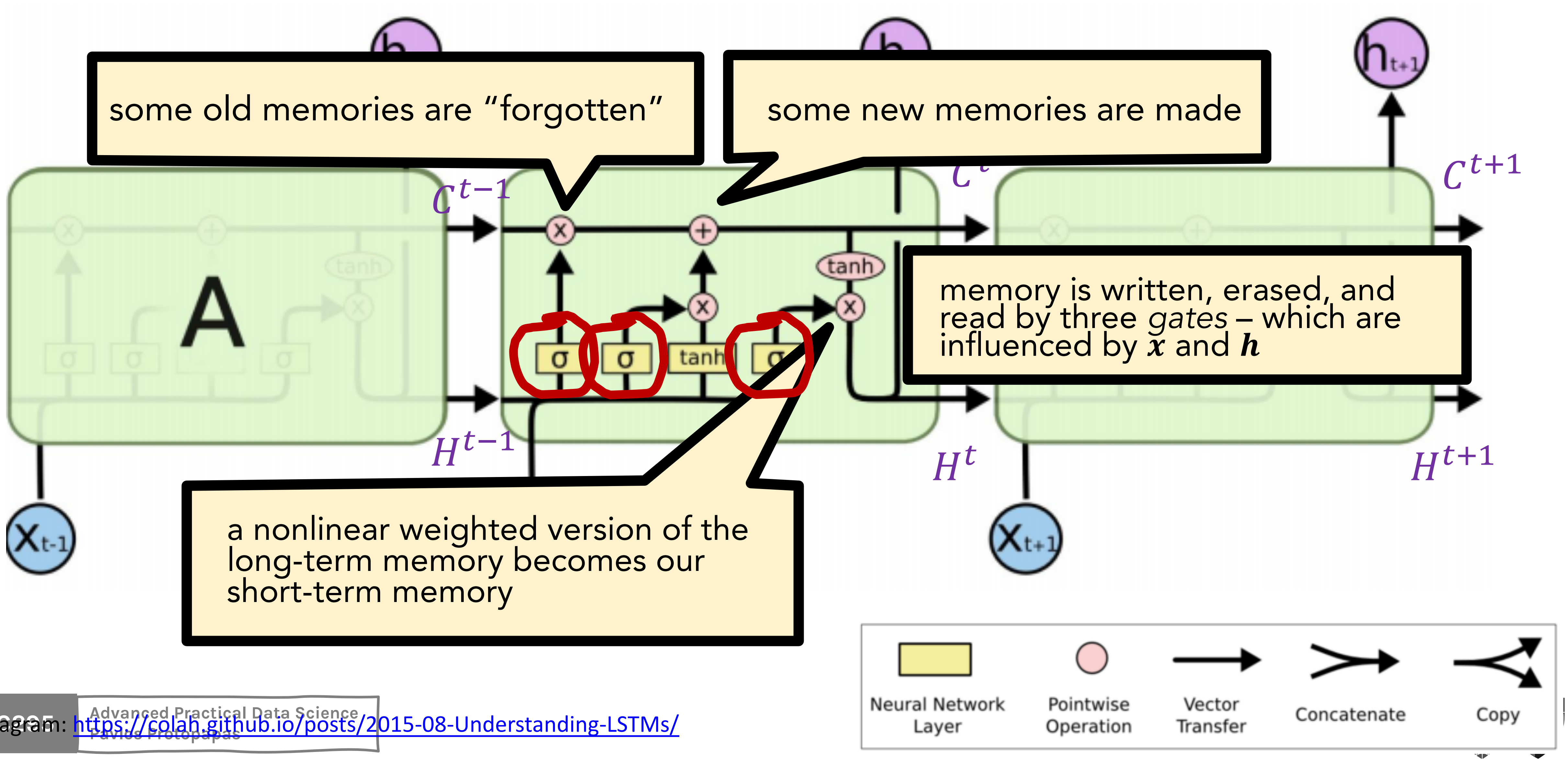


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

It's still possible for LSTMs to suffer from vanishing/exploding gradients, but it's way less likely than with vanilla RNNs:

- If RNNs wish to preserve info over long contexts, it must delicately find a recurrent weight matrix W_h that isn't too large or small

- However, LSTMs have 3 separate mechanism that adjust the flow of information (e.g., forget gate, if turned off, will preserve all info)

Long short-term memory (LSTM)

LSTM STRENGTHS?

- Almost always outperforms vanilla RNNs
- Captures long-range dependencies shockingly well

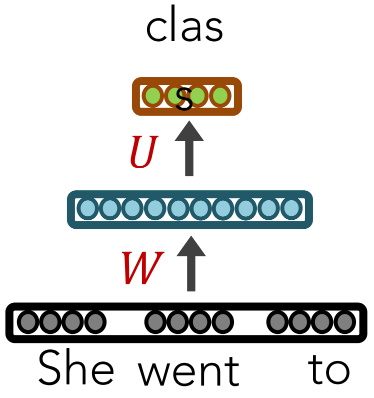
LSTM ISSUES?

- Has more weights to learn than vanilla RNNs; thus,
- Requires a moderate amount of training data (otherwise, vanilla RNNs are better)
- Can still suffer from vanishing/exploding gradients

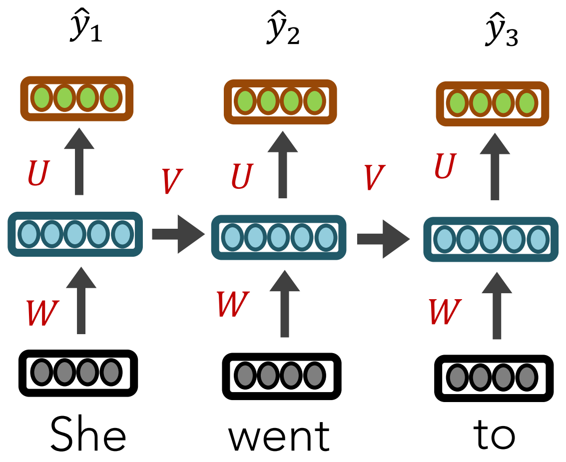
Sequential Modelling

$$P(\text{went}|She) = \frac{\text{count}(She\ went)}{\text{count}(She)}$$

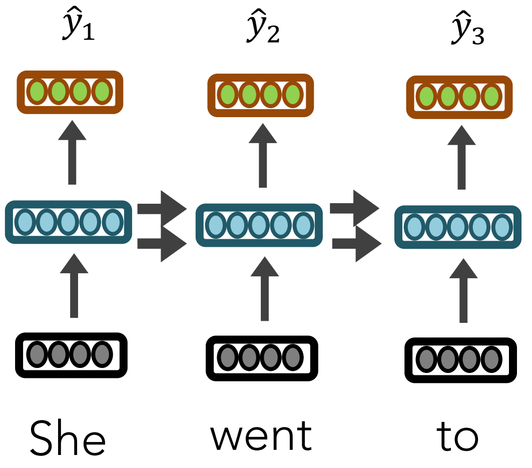
n-grams



FFNN



RNN



LSTM

Sequential Modelling

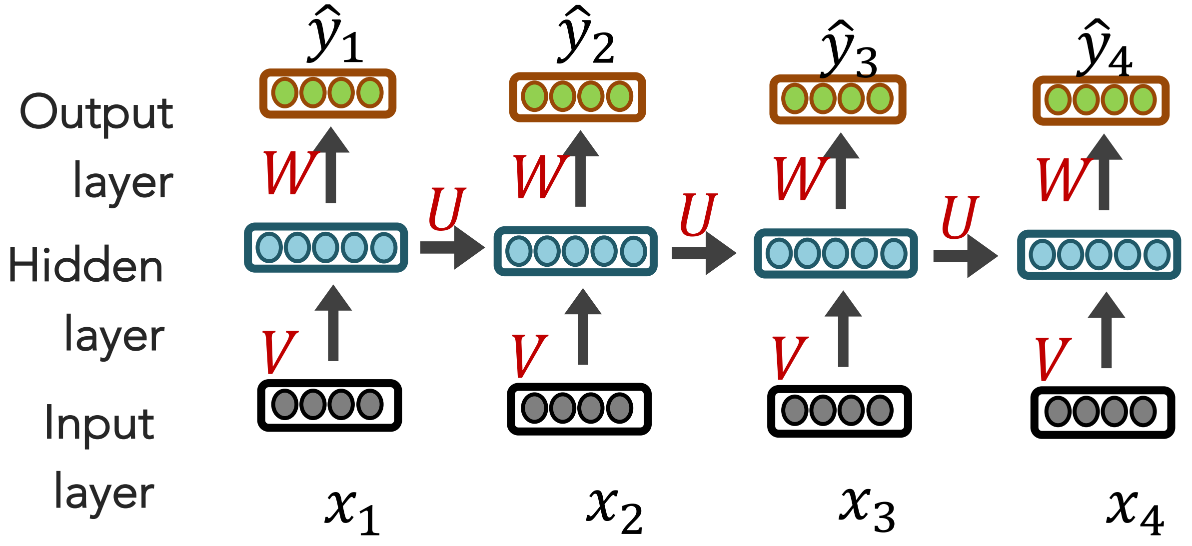
IMPORTANT

If your goal isn't to predict the next item in a sequence, and you rather do some other classification or regression task using the sequence, then you can:

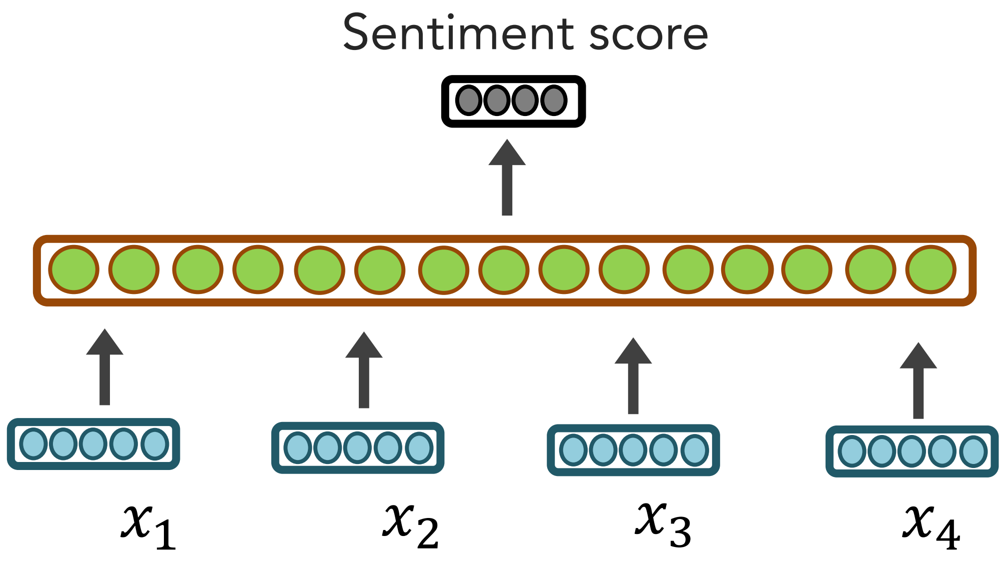
- Train an aforementioned model (e.g., LSTM) as a language model
- Use the **hidden layers** that correspond to each item in your sequence

Sequential Modelling

1. Train LM to learn hidden layer embeddings

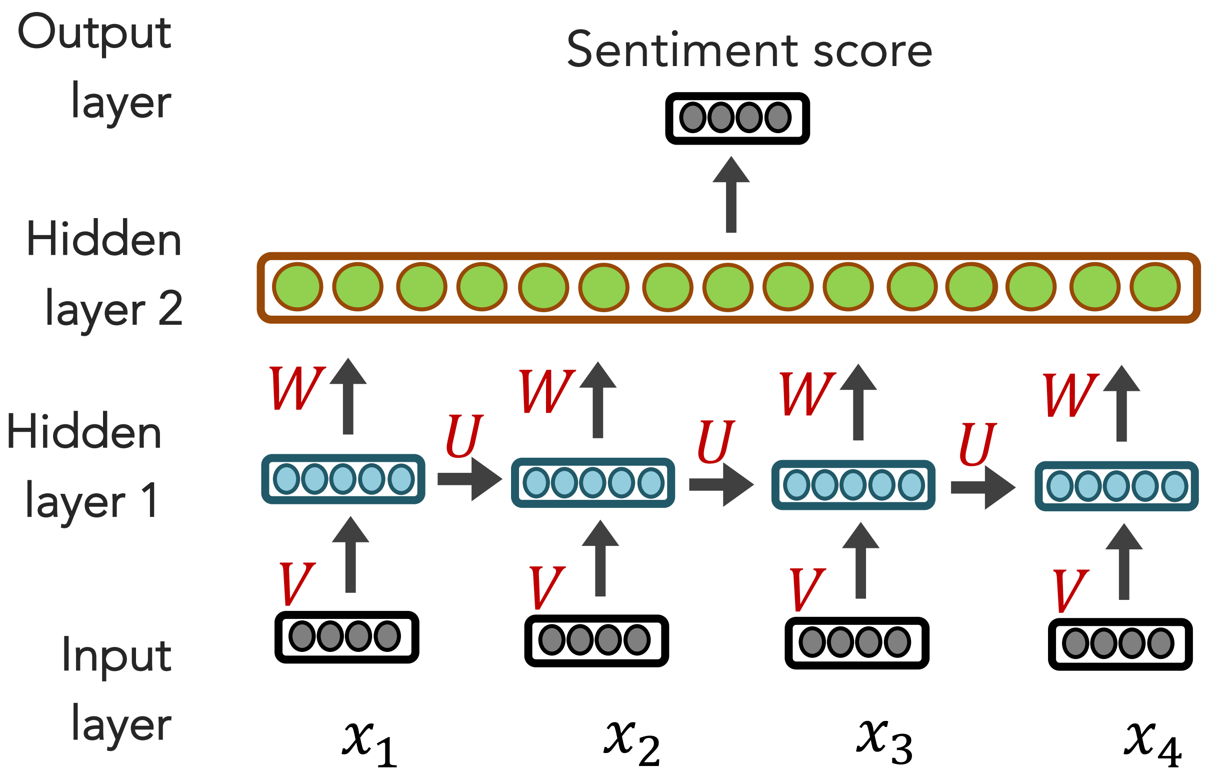


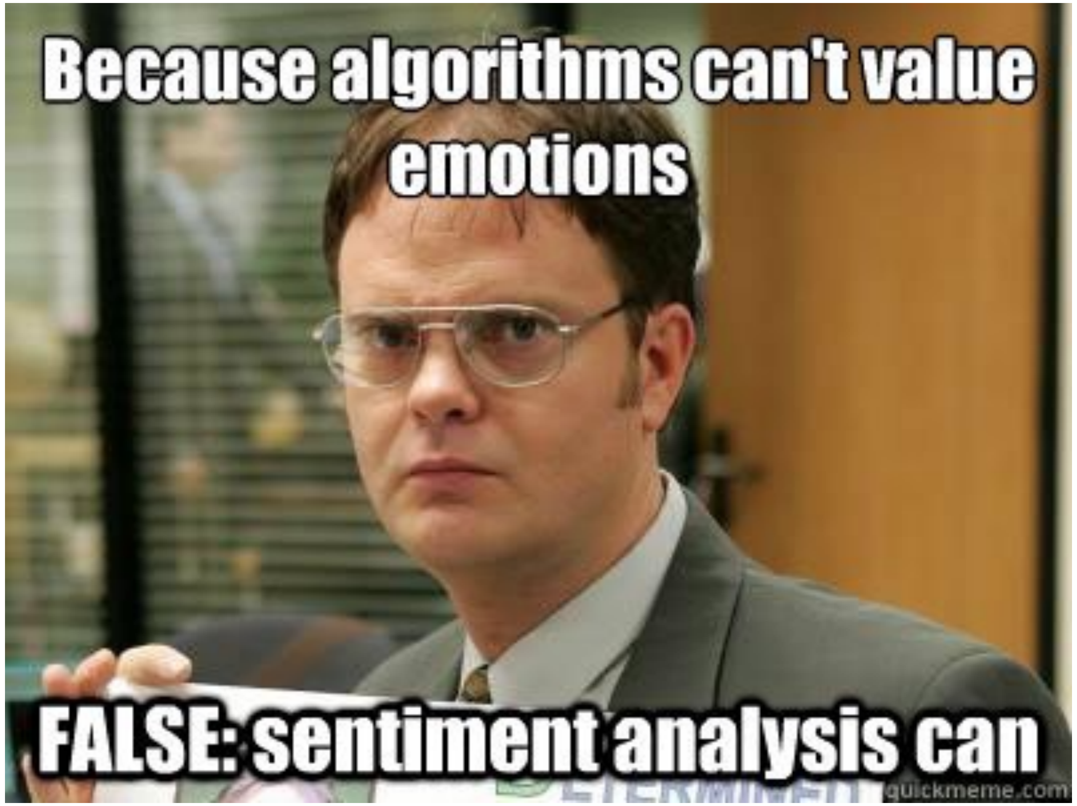
2. Use hidden layer embeddings for other tasks



Sequential Modelling

Or jointly learn hidden embeddings toward a particular task (end-to-end)




Because algorithms can't value emotions
FALSE: sentiment analysis can
quickmeme.com

You now have the foundation for modelling sequential data.

Most state-of-the-art advances are based on those core RNN/LSTM ideas. But, with tens of thousands of researchers and hackers exploring deep learning, there are many tweaks that haven proven useful.

(This is where things get crazy.)

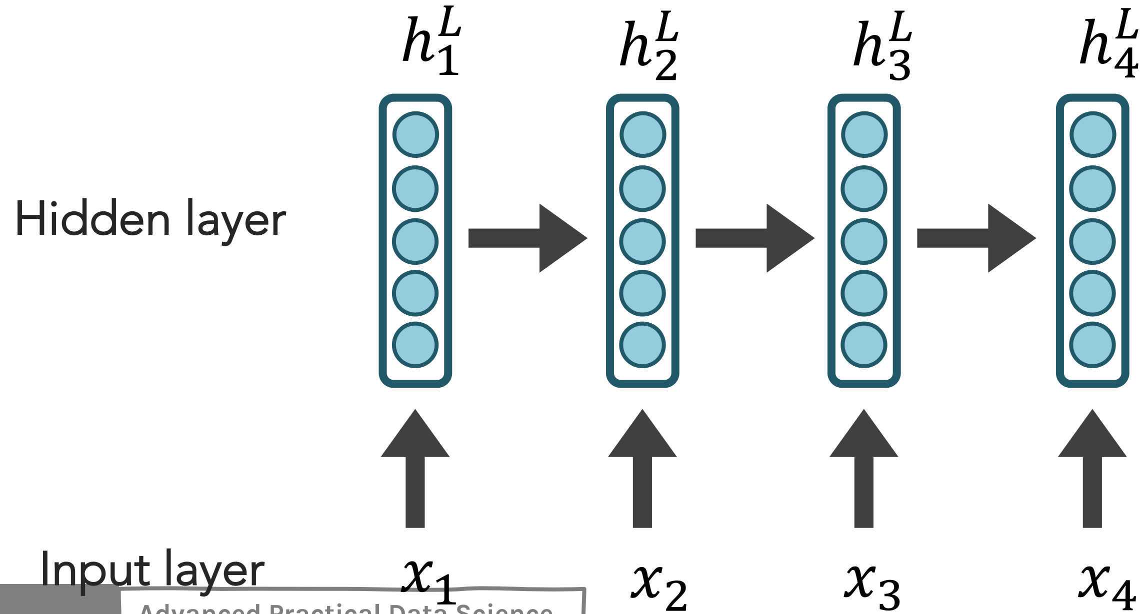
Bi-directional (review)

RNNs/LSTMs use the left-to-right context and sequentially process data.

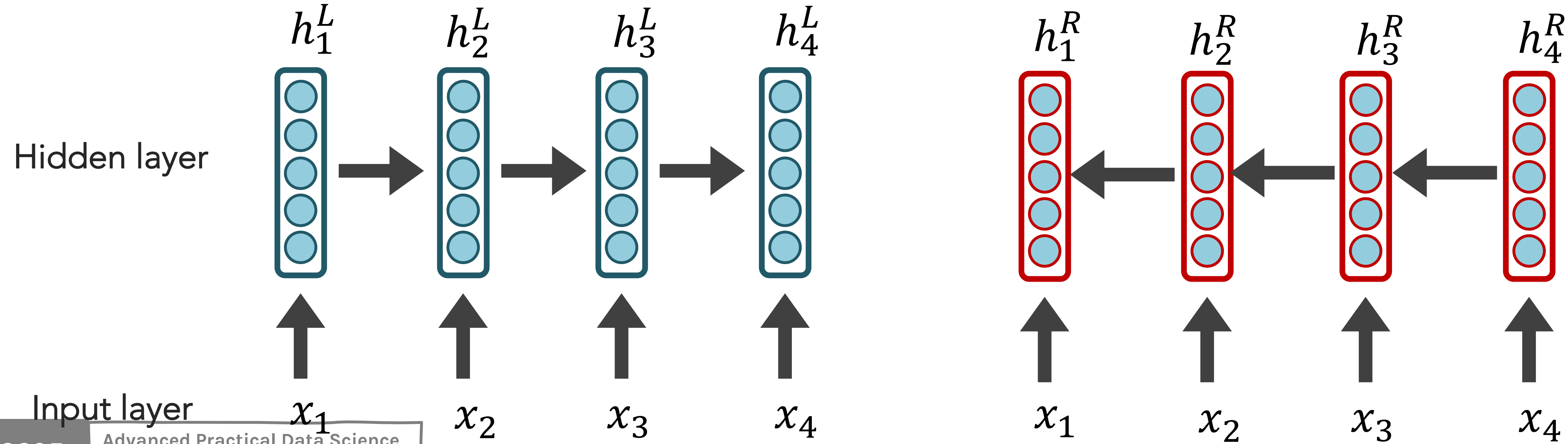
If you have full access to the data at testing time, why not make use of the flow of information from right-to-left, also?

RNN Extensions: Bi-directional LSTMs (review)

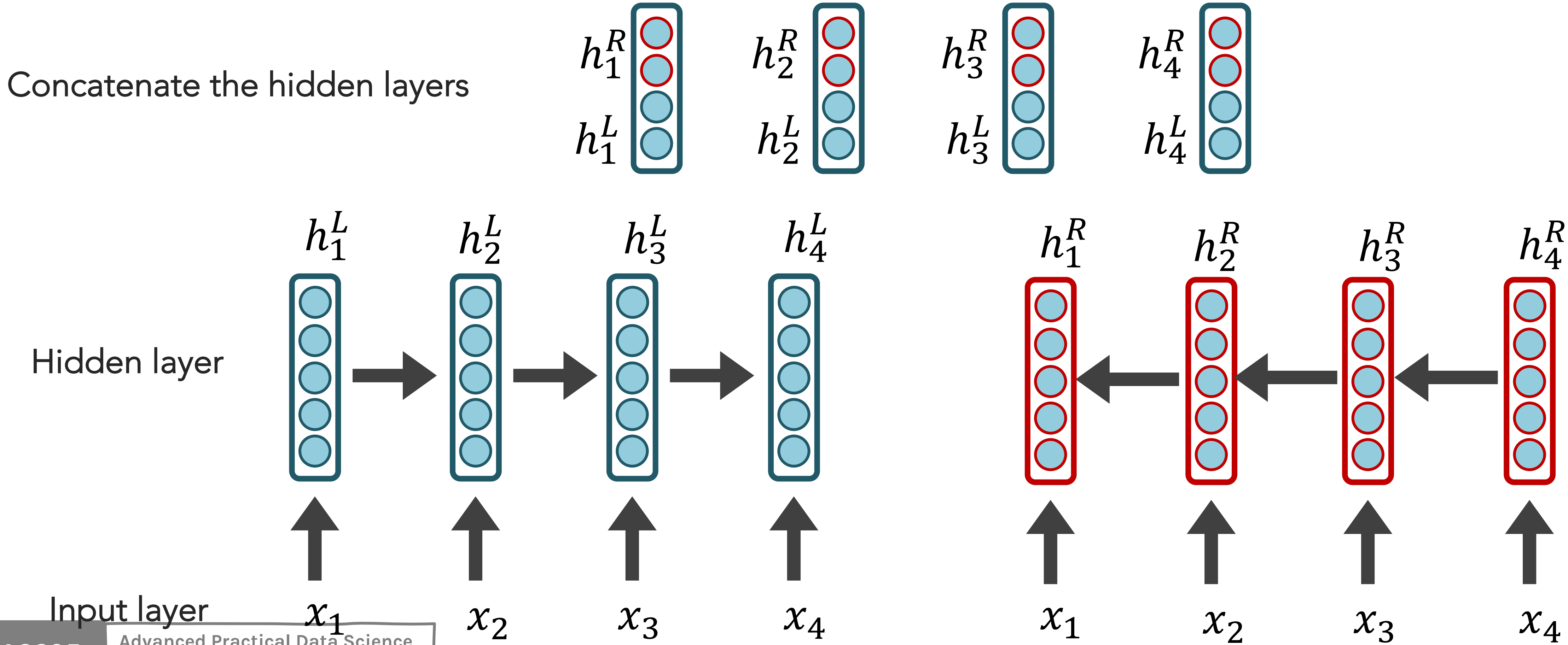
For brevity, let's use the follow schematic to represent an RNN



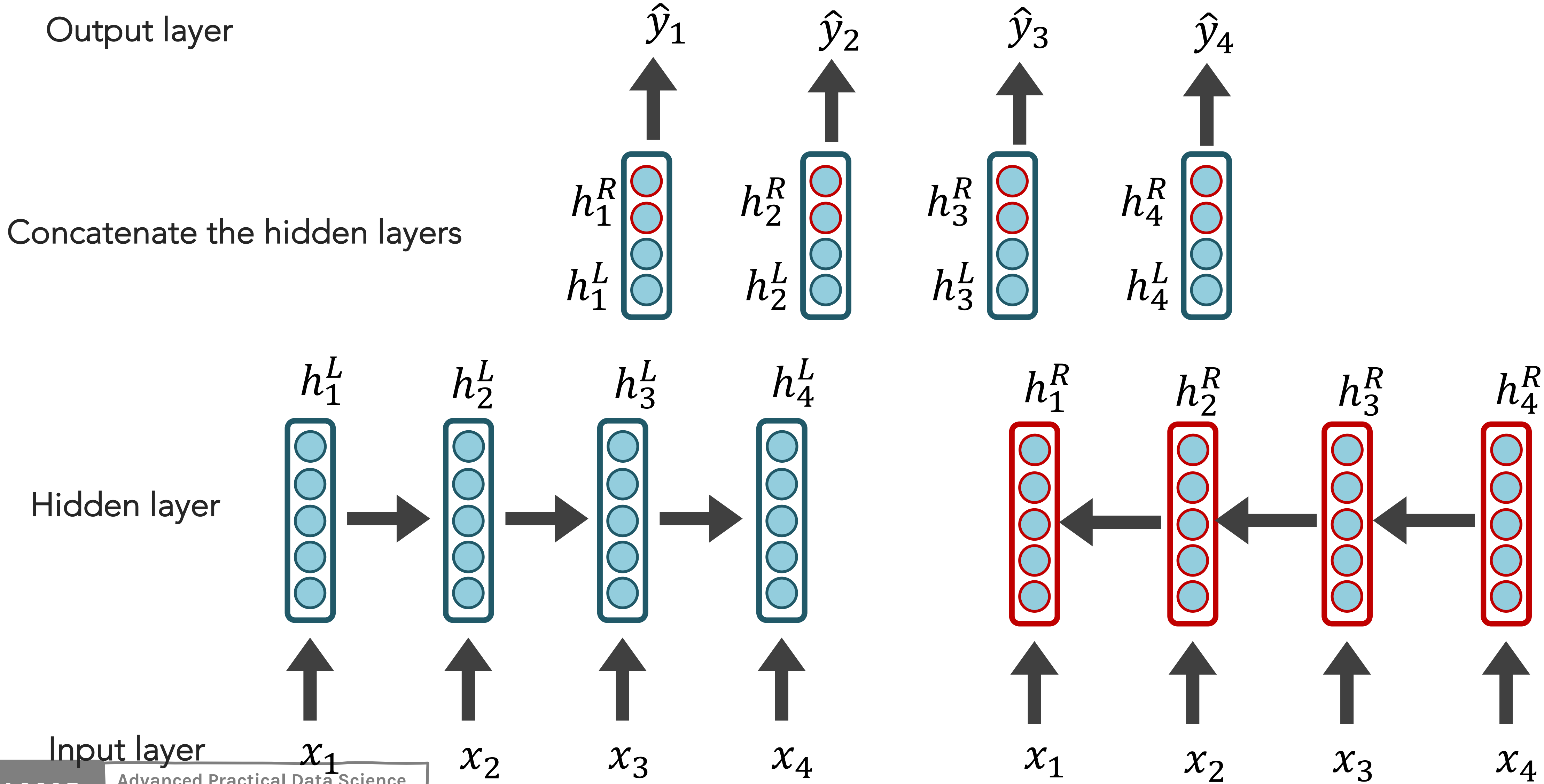
RNN Extensions: **Bi-directional LSTMs (review)**



RNN Extensions: Bi-directional LSTMs (review)



RNN Extensions: Bi-directional LSTMs (review)



RNN Extensions: Bi-directional LSTMs (review)

BI-LSTM STRENGTHS?

- Usually performs at least as well as uni-directional RNNs/LSTMs

BI-LSTM ISSUES?

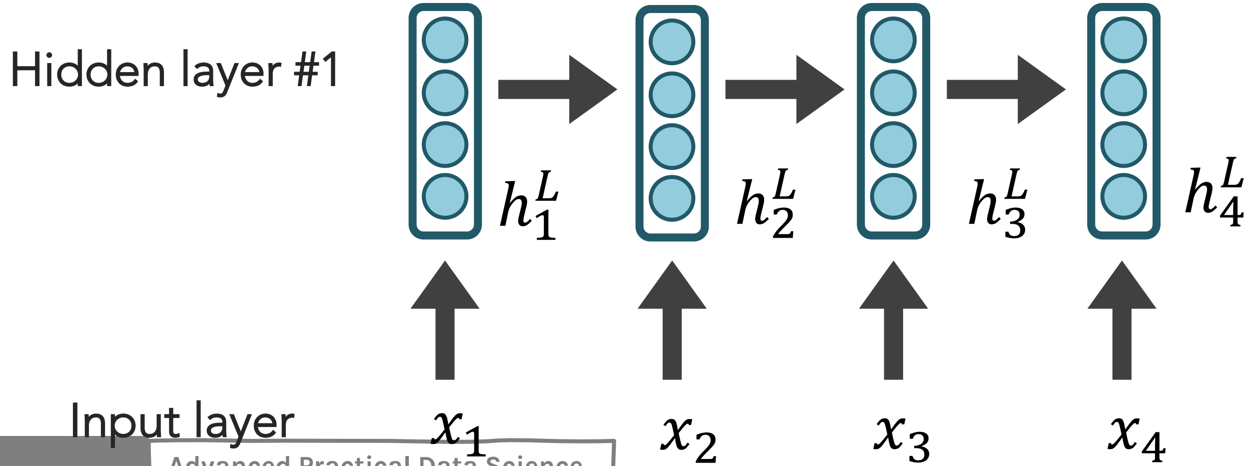
- Slower to train
- Only possible if access to full data is allowed

Deep RNN (review)

LSTMs units can be arranged in layers, so that the output of each unit is the input to the other units. This is called **a deep RNN**, where the adjective “deep” refers to these multiple layers.

- Each layer feeds the LSTM on the next layer
- First time step of a feature is fed to the first LSTM, which processes that data and produces an output (and a new state for itself).
- That output is fed to the next LSTM, which does the same thing, and the next, and so on.
- Then the second time step arrives at the first LSTM, and the process repeats.

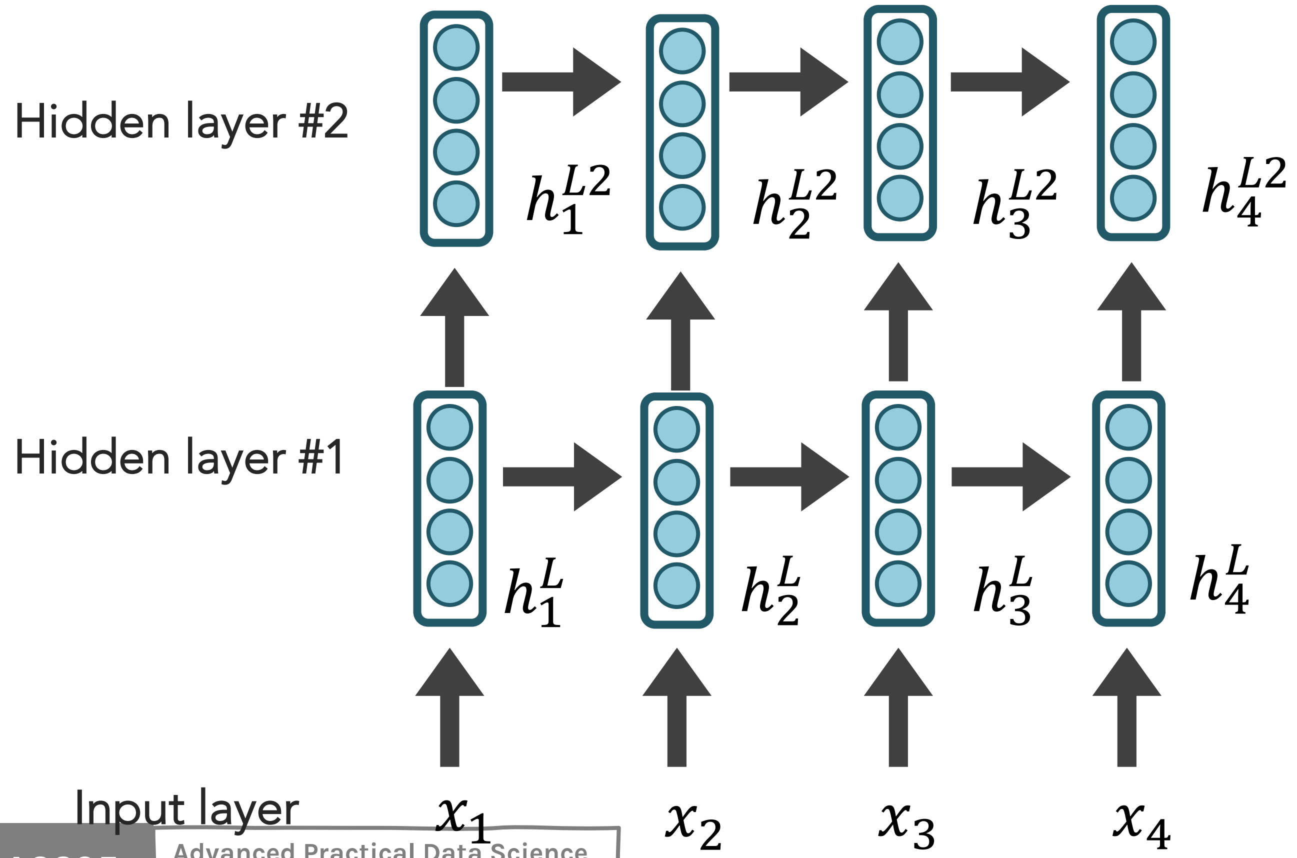
Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

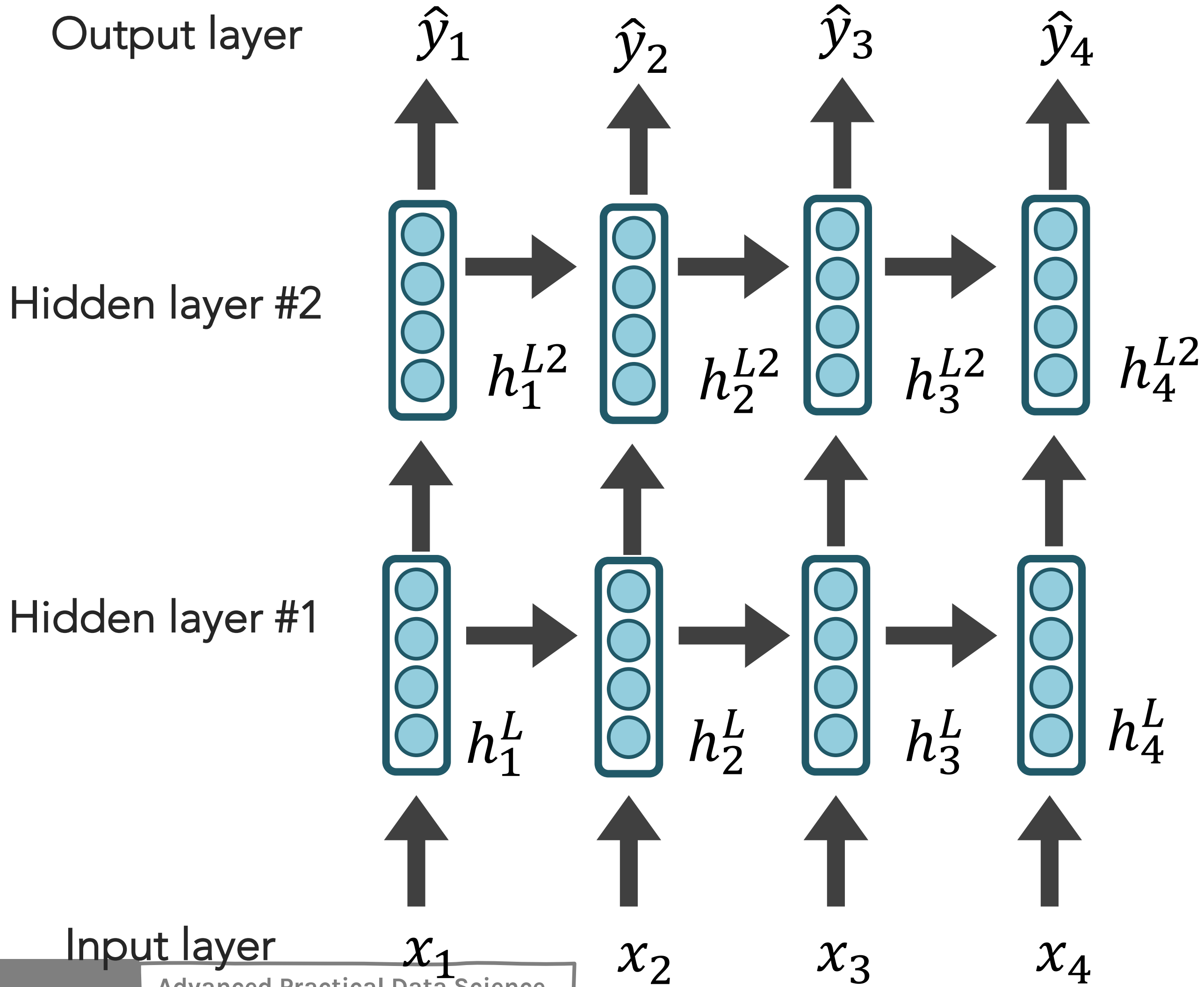
Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, RNNs/LSTMs +**ELMo**

Seq2Seq

ELMo: Stacked Bi-directional LSTMs

General Idea:

- Goal is to get highly rich embeddings for each word (unique type)
- Use both directions of context (bi-directional), with increasing abstractions (stacked)
- Linearly combine all abstract representations (hidden layers) and optimize w.r.t. a particular task (e.g., sentiment classification)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

ELMo: Stacked Bi-directional LSTMs

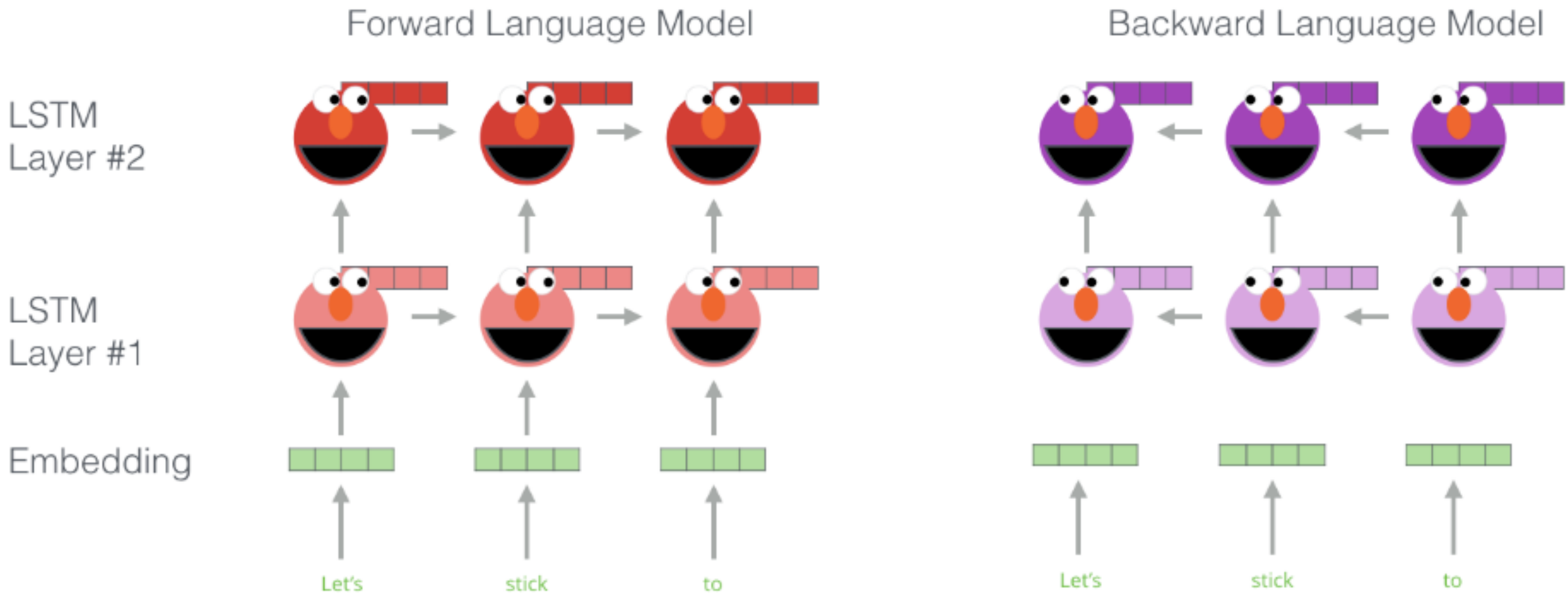


Illustration: http://jalammar.github.io/illustrated-bert/

Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context

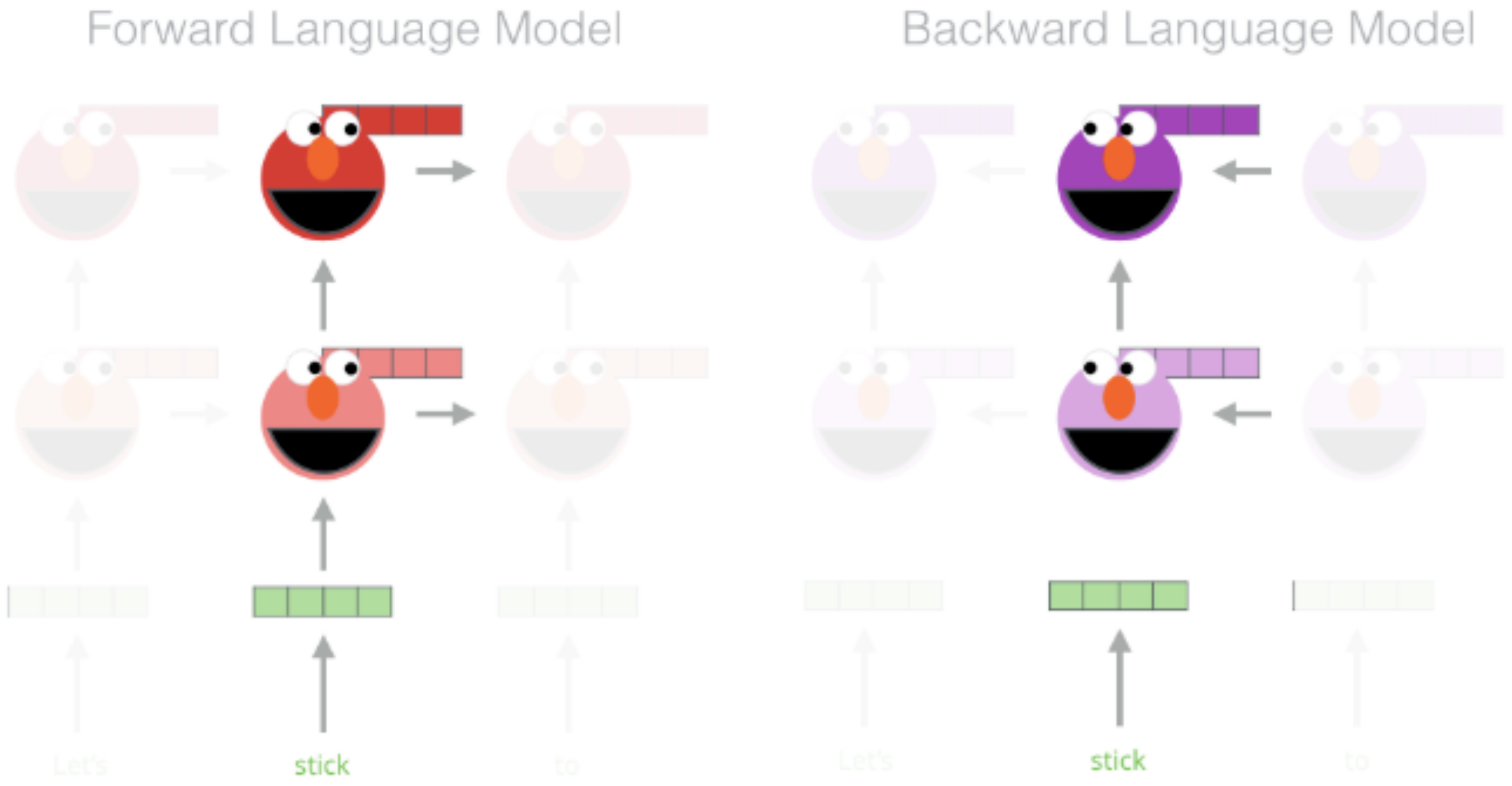


Illustration: http://jalammar.github.io/illustrated-bert/

ELMo: Stacked Bi-directional LSTMs

- ELMo yields incredibly good word embeddings, which yields state-of-the-art results when applied to many NLP tasks.
- Main ELMo takeaway: ELMo does not give you a matrix embedding as with Word2Vec, but a trained bidirectional LSTM to be used with the task.

REFLECTION

So far, for all of our sequential modelling, we have been concerned with emitting 1 output per input datum.

Sometimes, a *sequence* is the smallest granularity we care about though (e.g., an English sentence)

Outline

NLP Tasks

Transfer Learning in NLP

Language Modelling, n-grams

Word Embeddings (character embeddings)

Neural Networks LM:

FFNN, RNNs/LSTMs +ELMo

Seq2Seq


WAS THAT...
SARCASM?

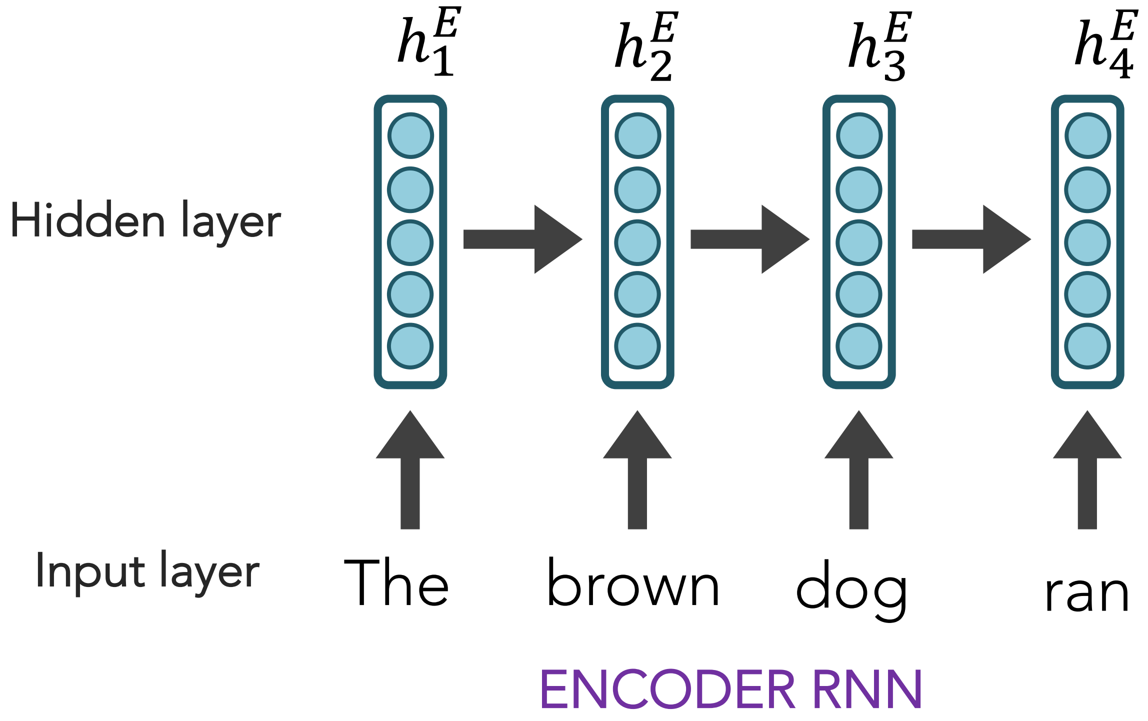
Sequence-to-Sequence (seq2seq)

- If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly sub-optimal to translate word by word (like our current models are suited to do).

- Instead, let a ***sequence*** of tokens be the unit that we ultimately wish to work with (a sequence of length **N** may emit a sequences of length **M**)

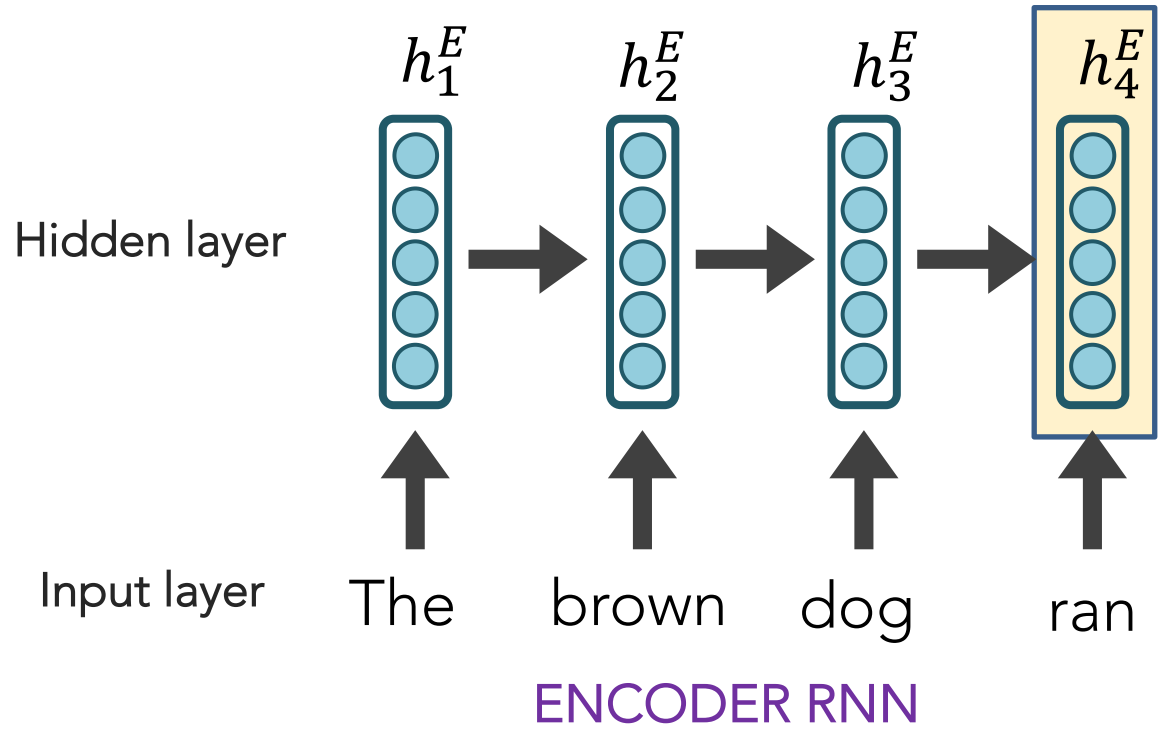
- Seq2seq models are comprised of **2 RNNs**: 1 encoder, 1 decoder

Sequence-to-Sequence (seq2seq)



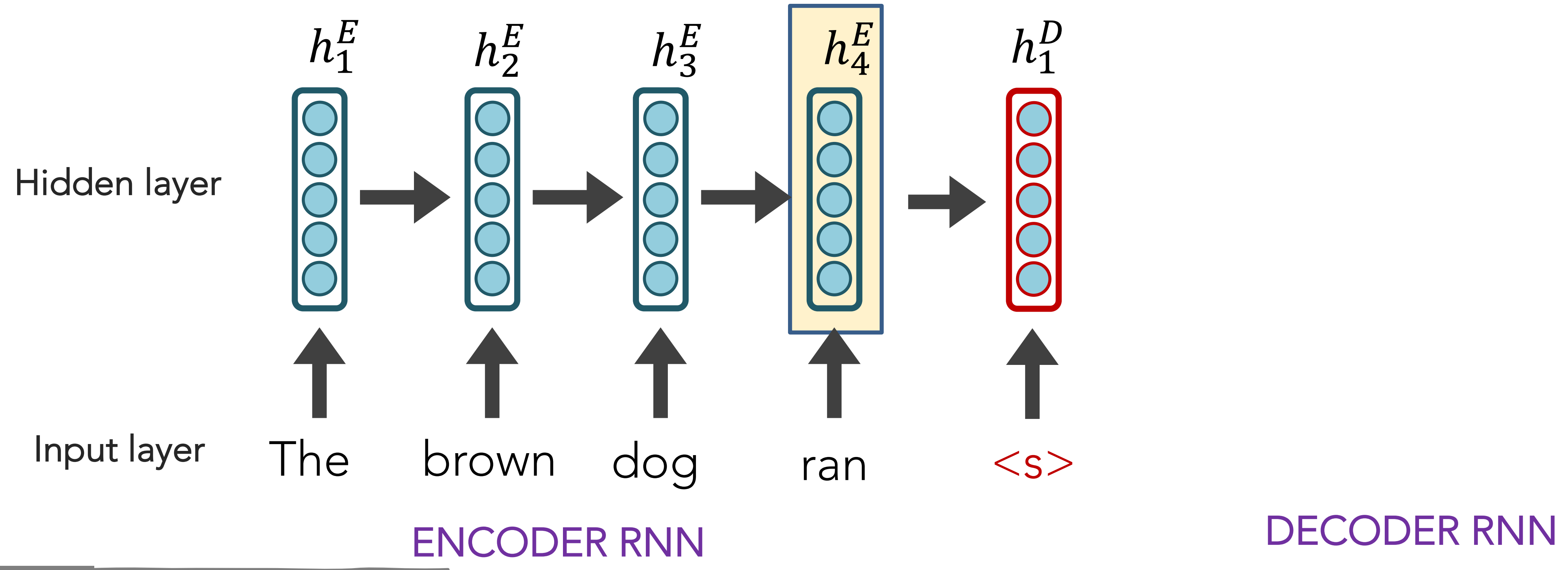
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN
is the initial state of the decoder RNN



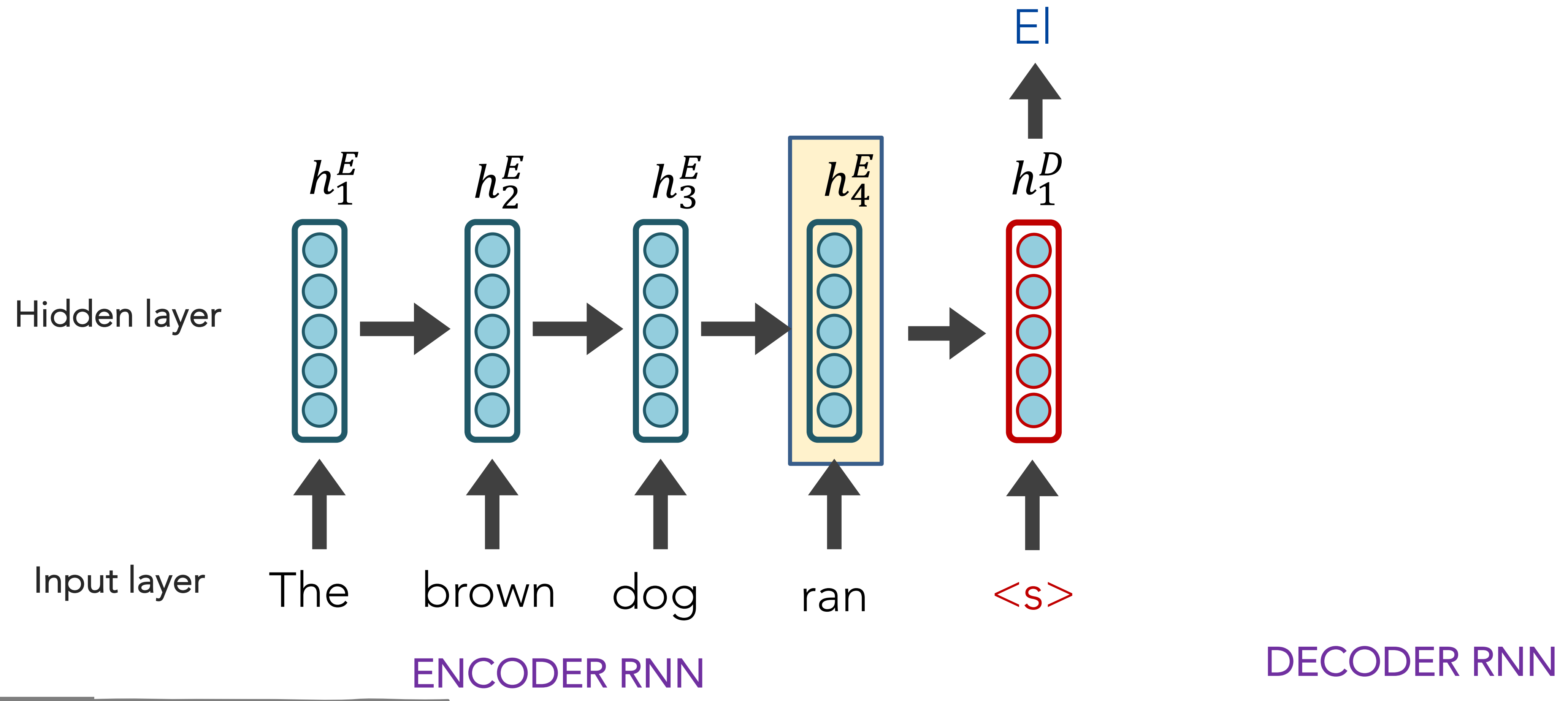
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN



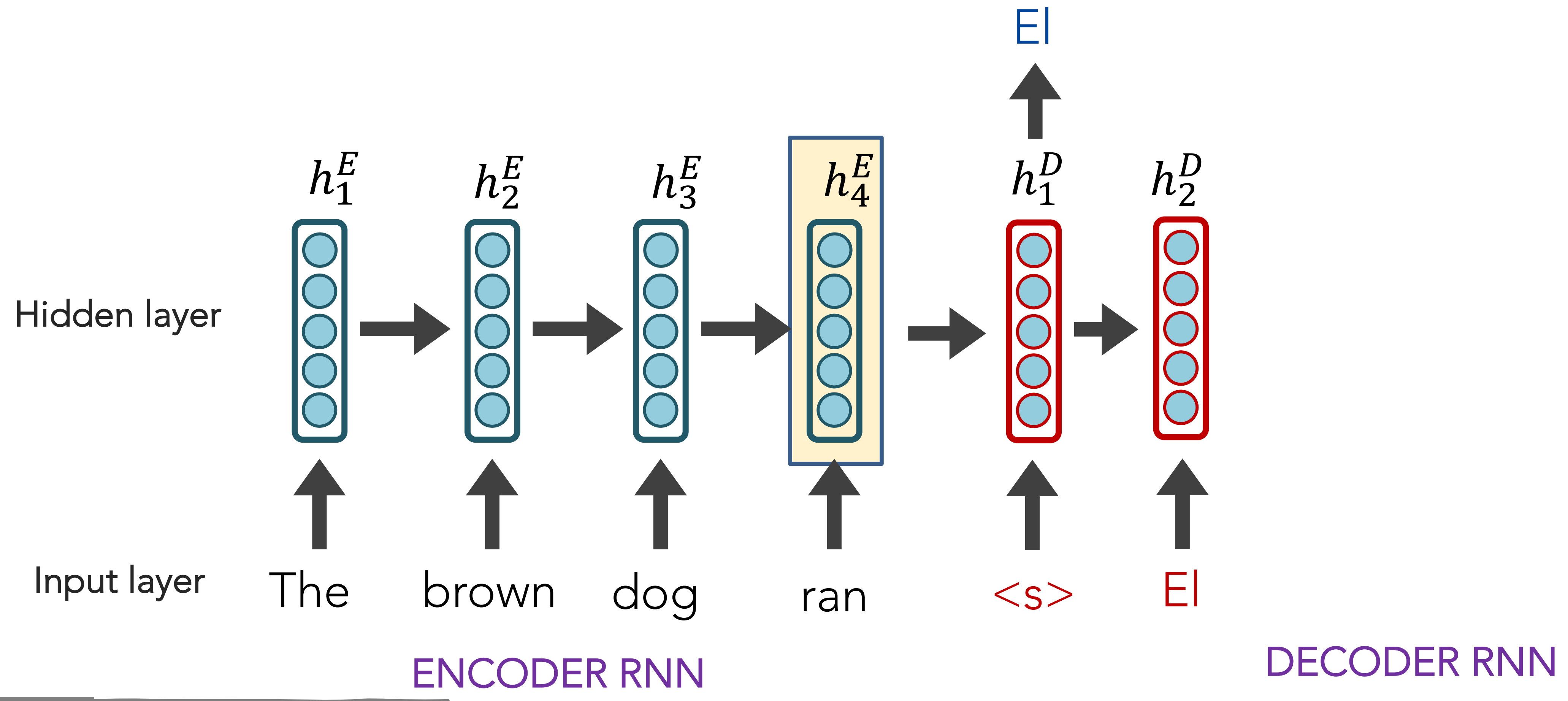
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN
is the initial state of the decoder RNN



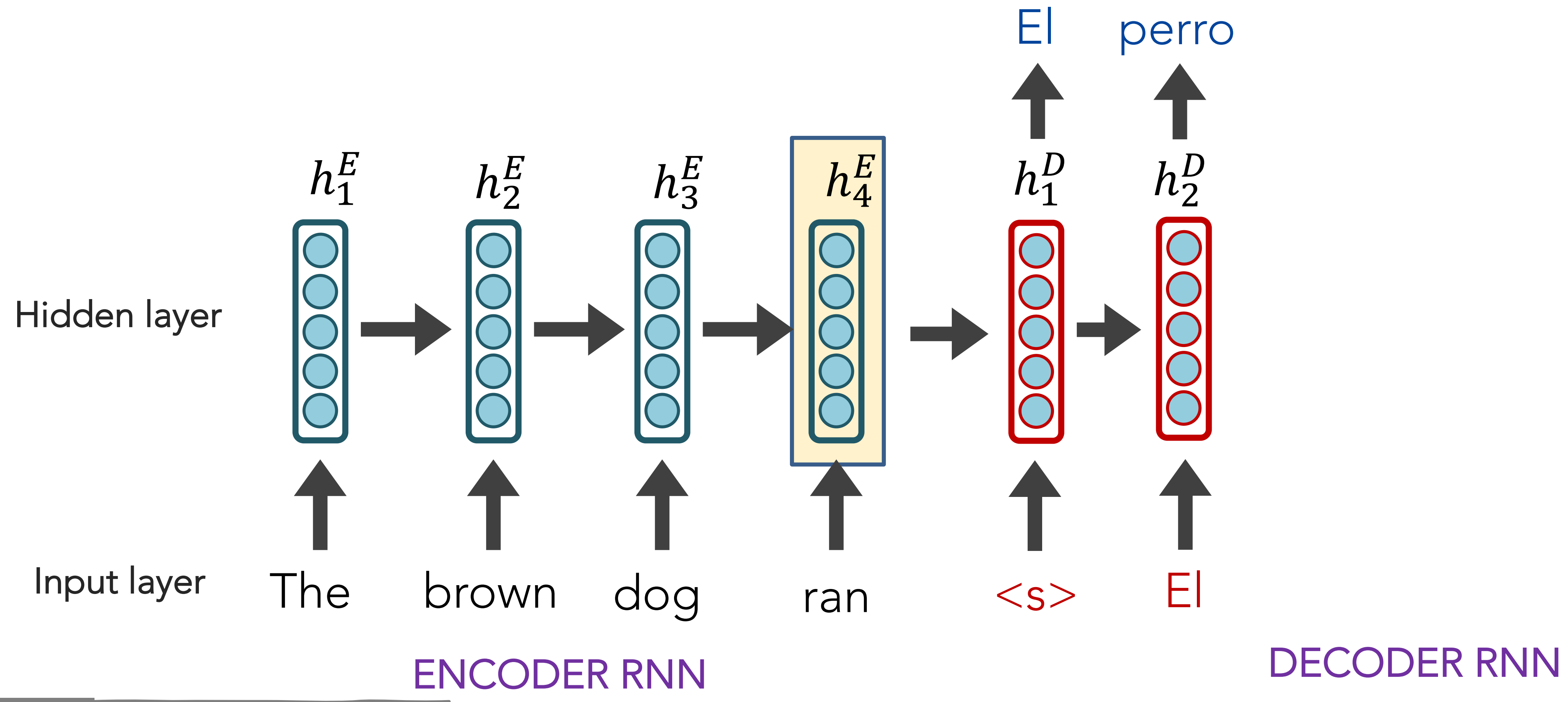
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN



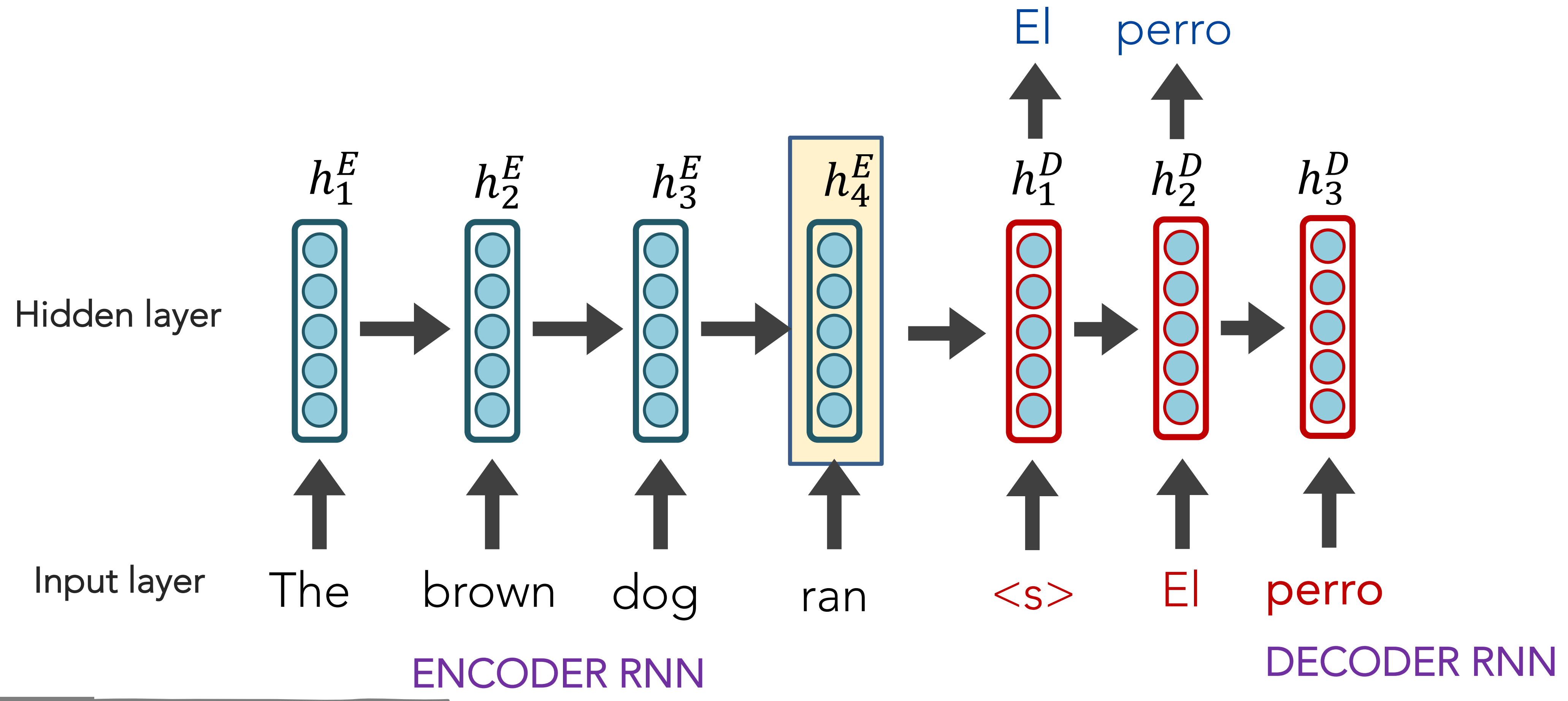
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN
is the initial state of the decoder RNN



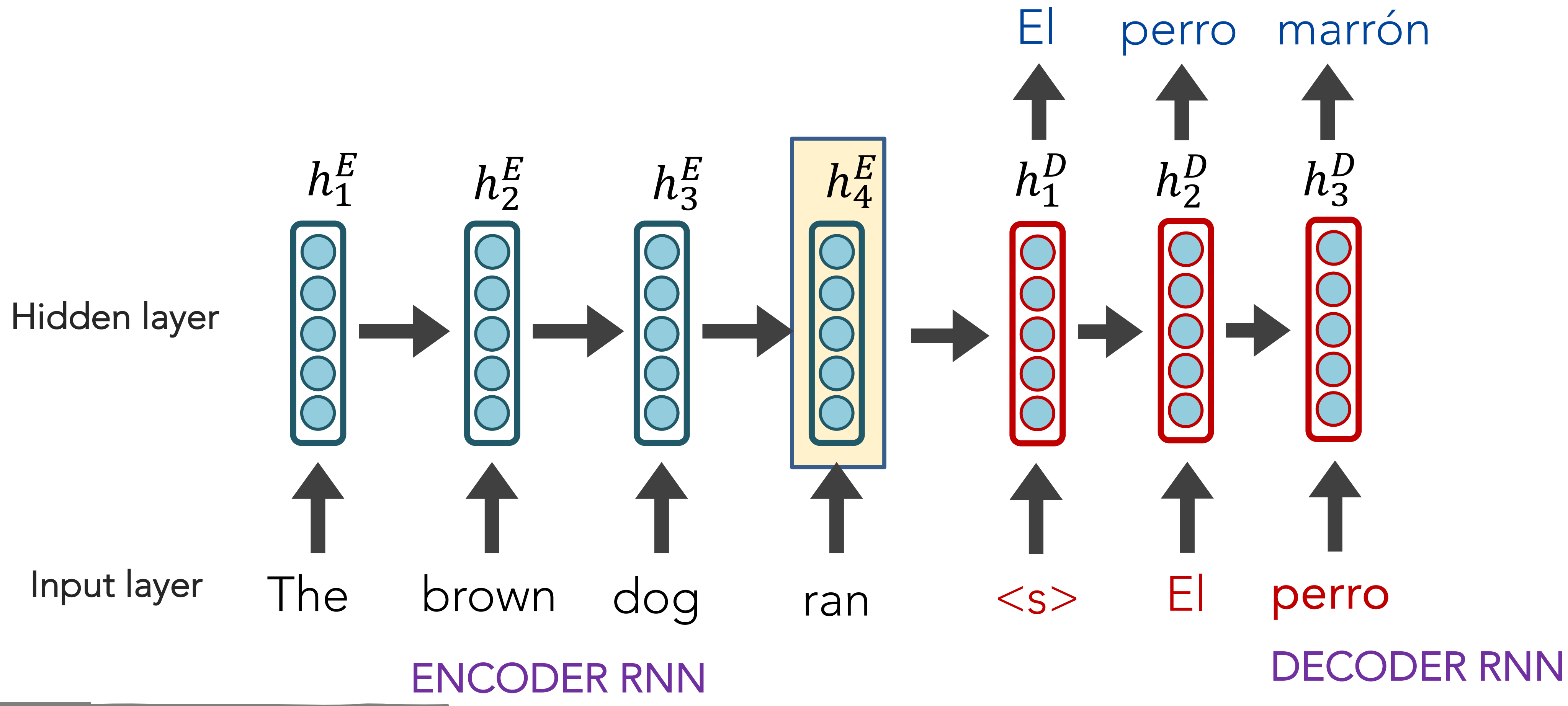
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN

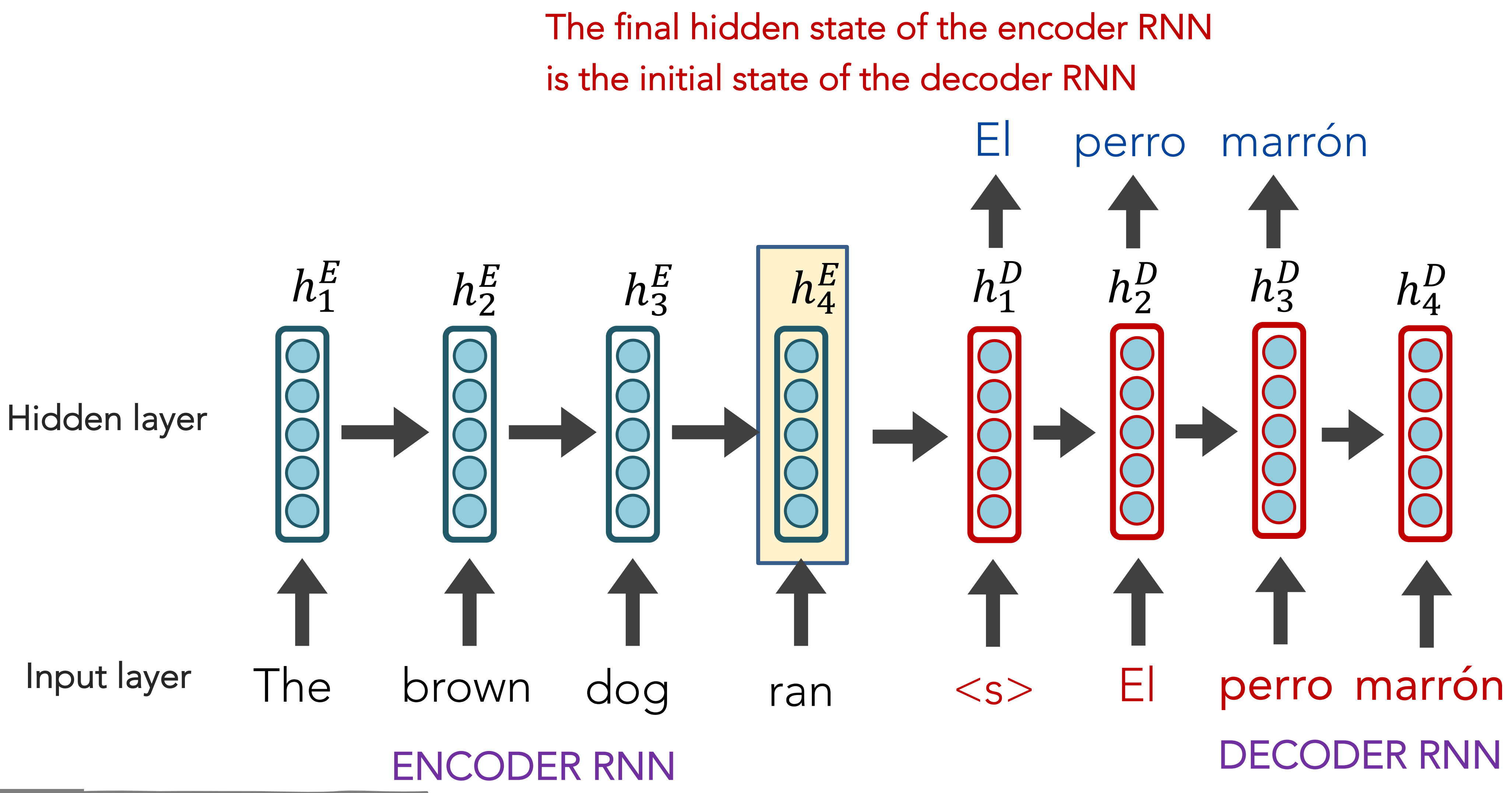


Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN

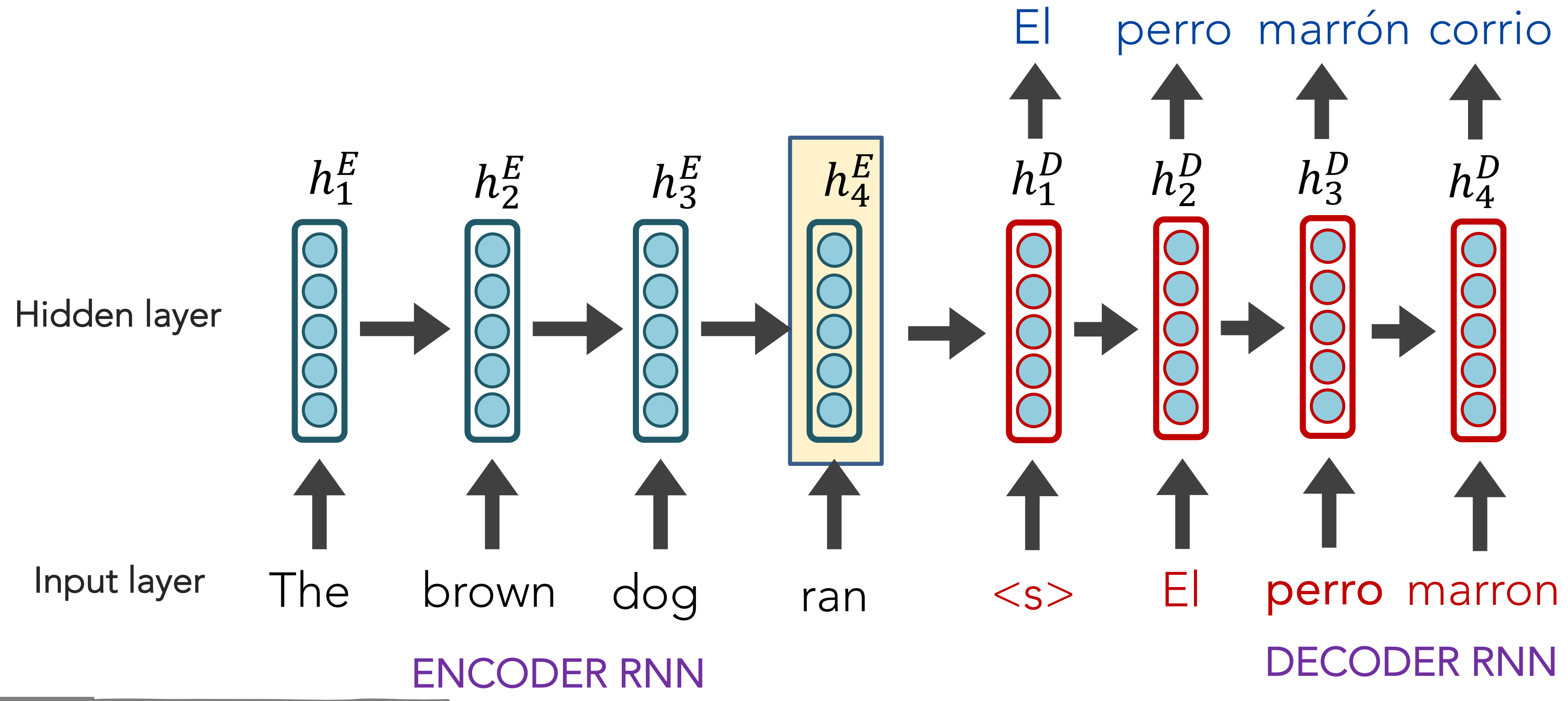


Sequence-to-Sequence (seq2seq)



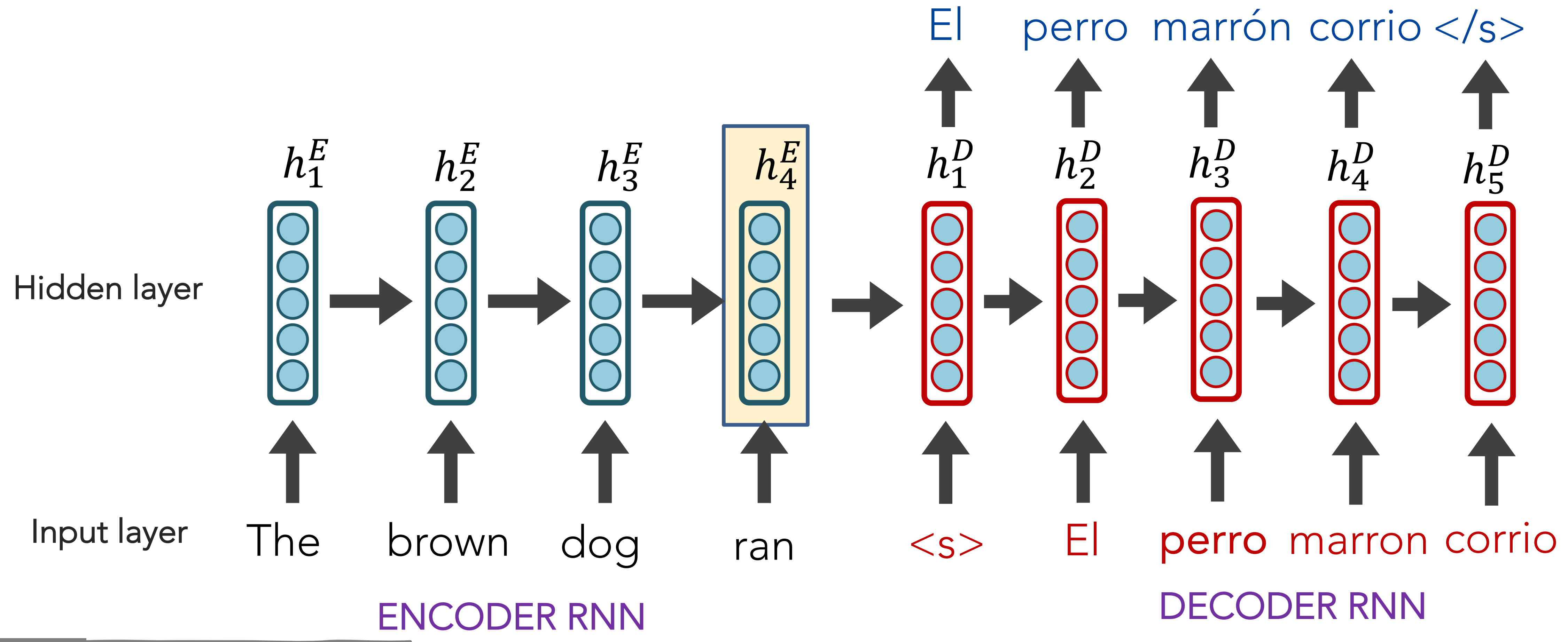
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN



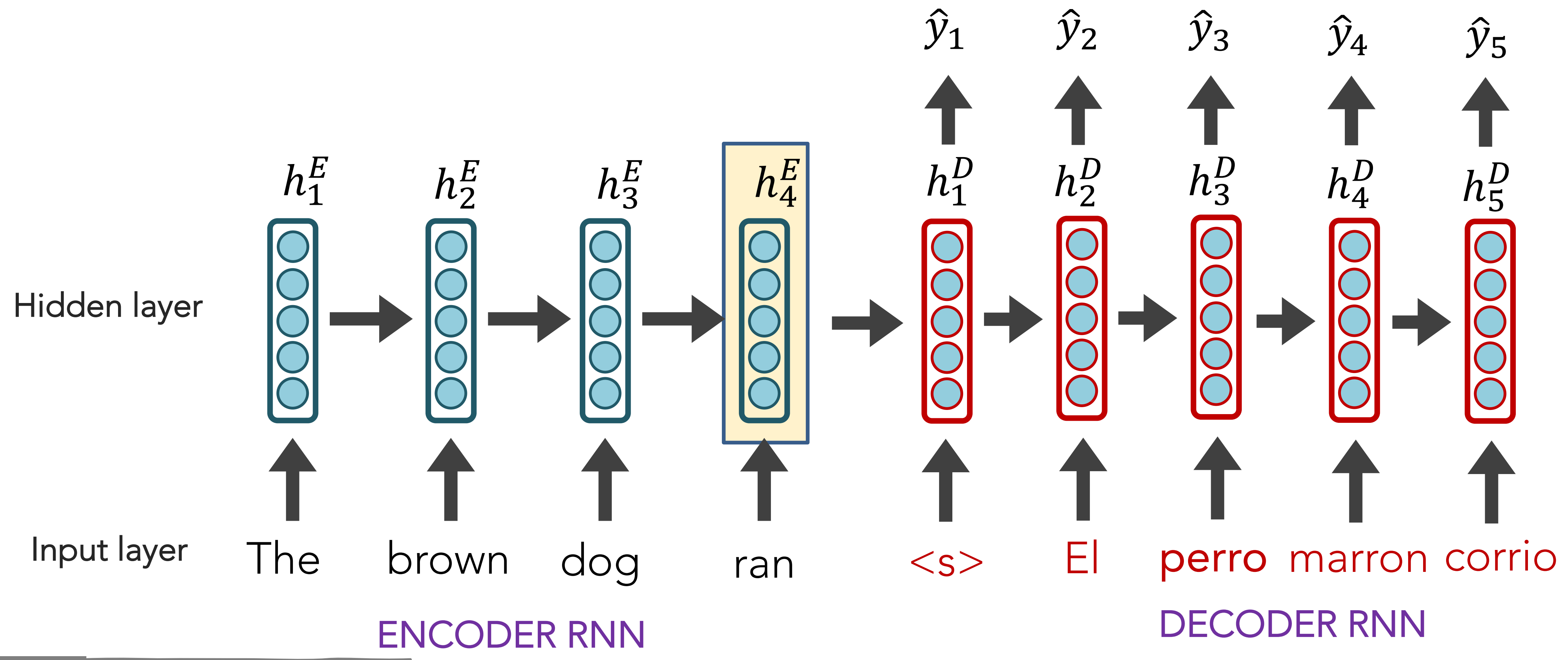
Sequence-to-Sequence (seq2seq)

The final hidden state of the decoder RNN is </s>



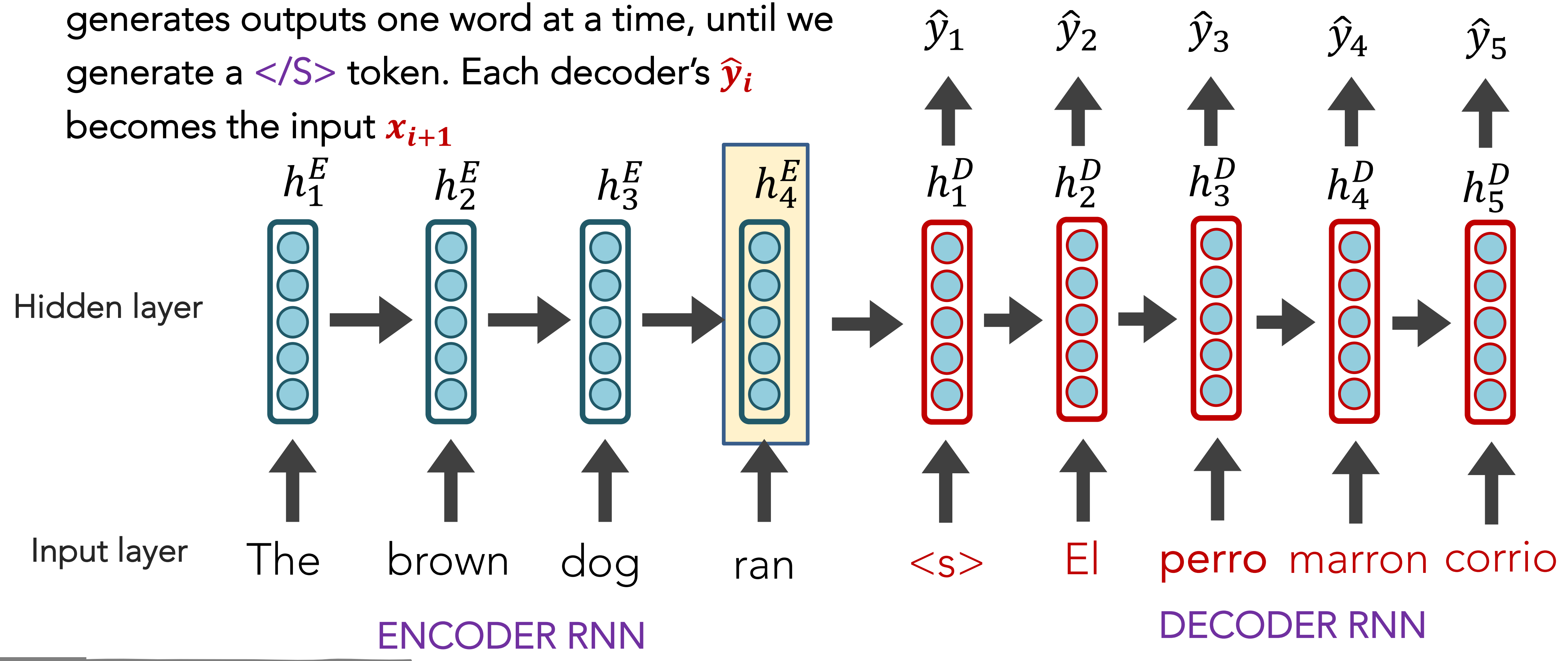
Sequence-to-Sequence (seq2seq)

Training occurs like RNNs typically do.



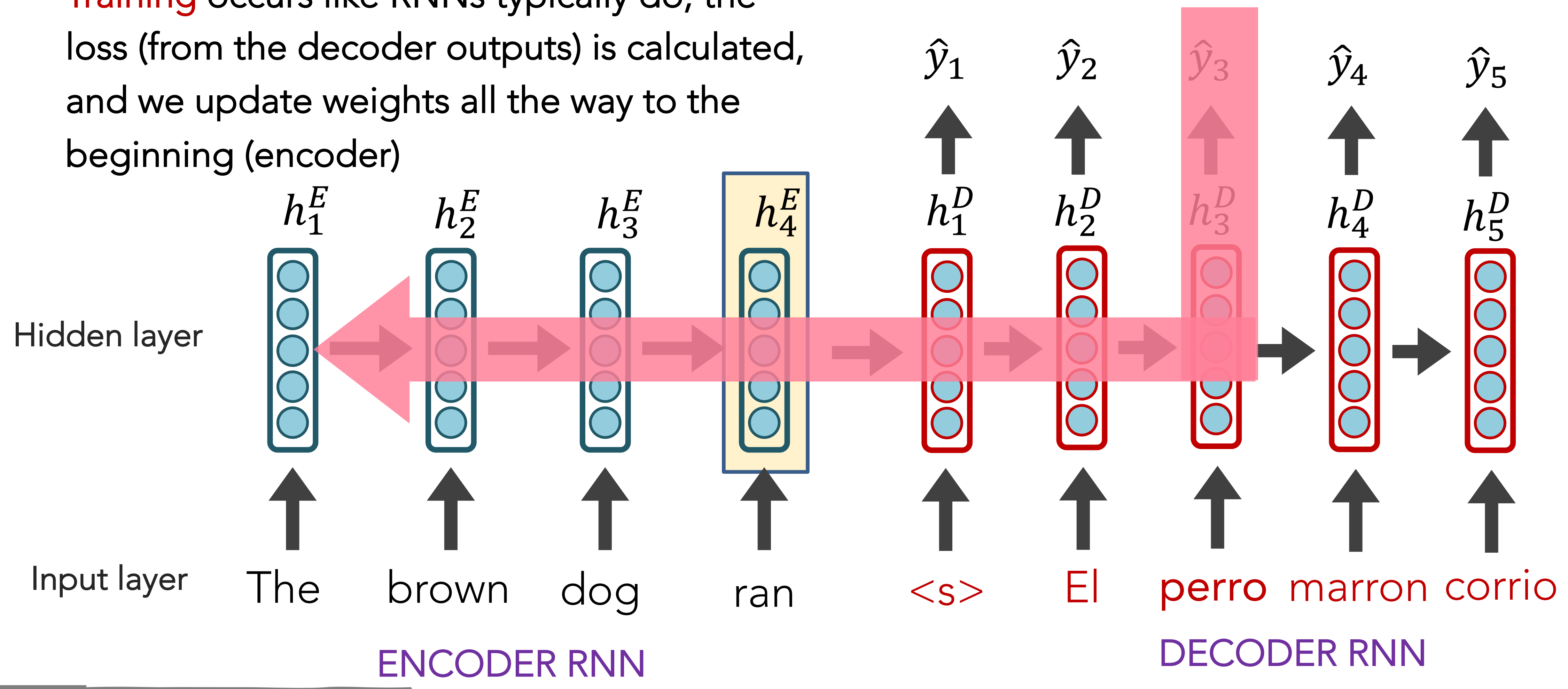
Sequence-to-Sequence (seq2seq)

Training occurs like RNNs typically do. Decoder generates outputs one word at a time, until we generate a </S> token. Each decoder's $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ becomes the input $\boldsymbol{x}_{\boldsymbol{i+1}}$



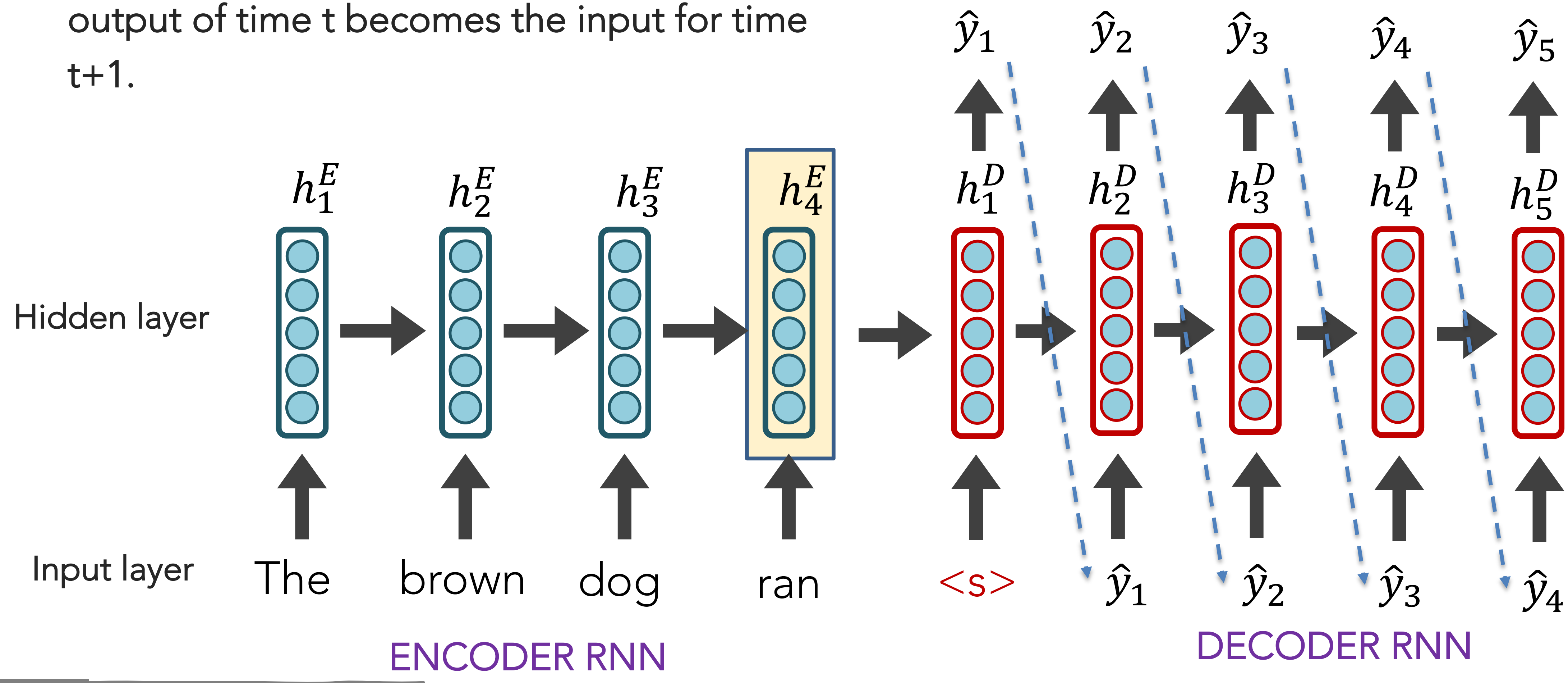
Sequence-to-Sequence (seq2seq)

Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

Generation: During actual translation, the output of time t becomes the input for time t+1.



Sequence-to-Sequence (seq2seq)

The entire "meaning" of the 1st sequence is expected to be packed into this one embedding, and that the encoder then never interacts w/ the decoder again!

Instead, what if the decoder, at each step, sees or pays attention to all of the encoder's hidden states?

Also, RNNs are sequential and can not be parallelized.

Instead, what if we capture long and short memories in some other way?

Thank you