Lecture 7: seq2seq + Attention

Sequence Generation

Harvard

AC295/CS287r/CSCI E-115B

Chris Tanner

ScHoolboy Q

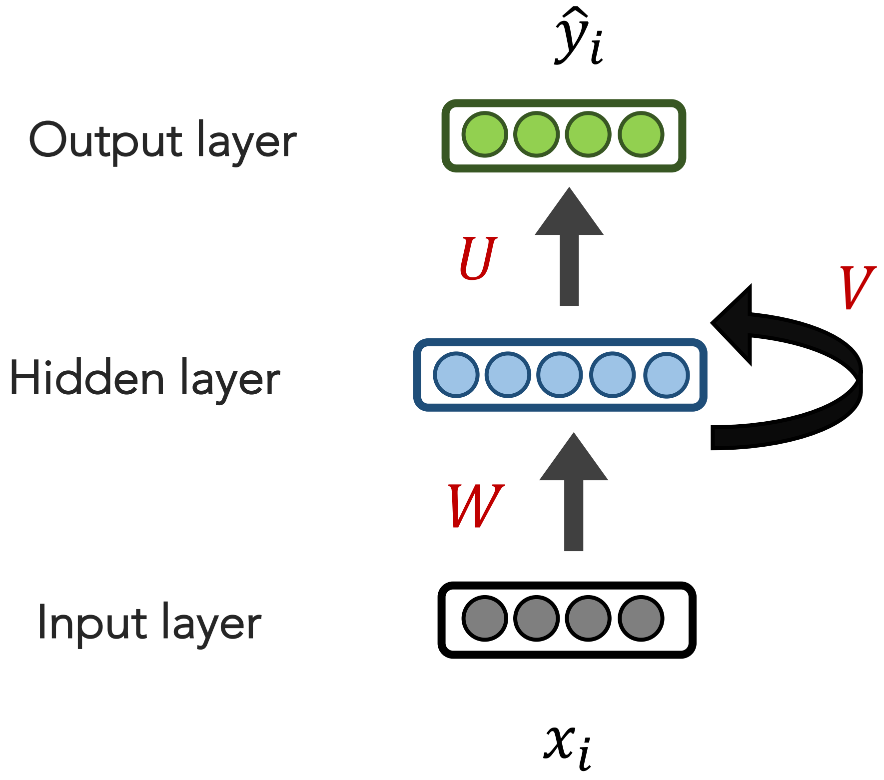
[seq2seq] + Attention (2019)

ANNOUNCEMENTS

- HW2 is out! Determine your mystery language.
- Research Proposals are due in 7 days, Sept 30.
- Office Hours:
  - Today, my OH will be pushed back: 3:30pm – 5:30pm
  - Please reserve your coding questions for the TFs and/or EdStem, as I hold office hours solo, and debugging code can easily bottleneck the queue.
- Saturday @ 9am, I'll host & record a review session. Submit questions on Ed's Sway

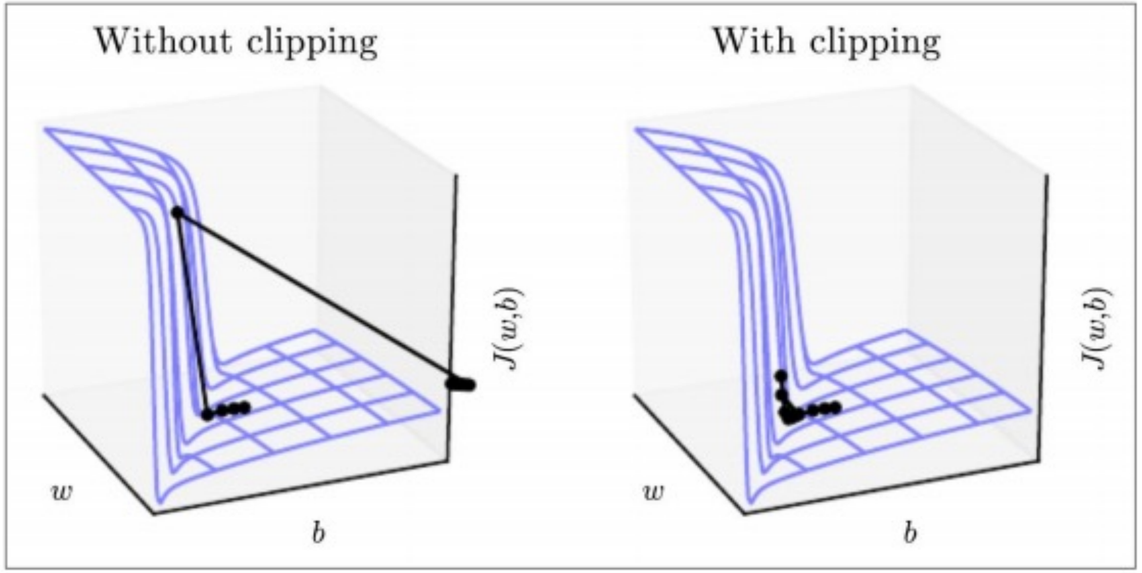
RECAP: L5

- RNNs help capture more context while avoiding sparsity, storage, and compute issues!
- The hidden layer is what we care about. It represents the word's "meaning".
- Often suffers from vanishing/exploding gradients



RECAP: L6

- Gradient Clipping may help all NNs
- LSTMs (1997) are usually much better than vanilla RNNs
  - Captures long-range dependencies
  - Doesn’t suffer as much w/ its gradients



Emilia told her project partner Alan about ___ latest idea.

He tends to stress out and put too much pressure on himself

RECAP: L6

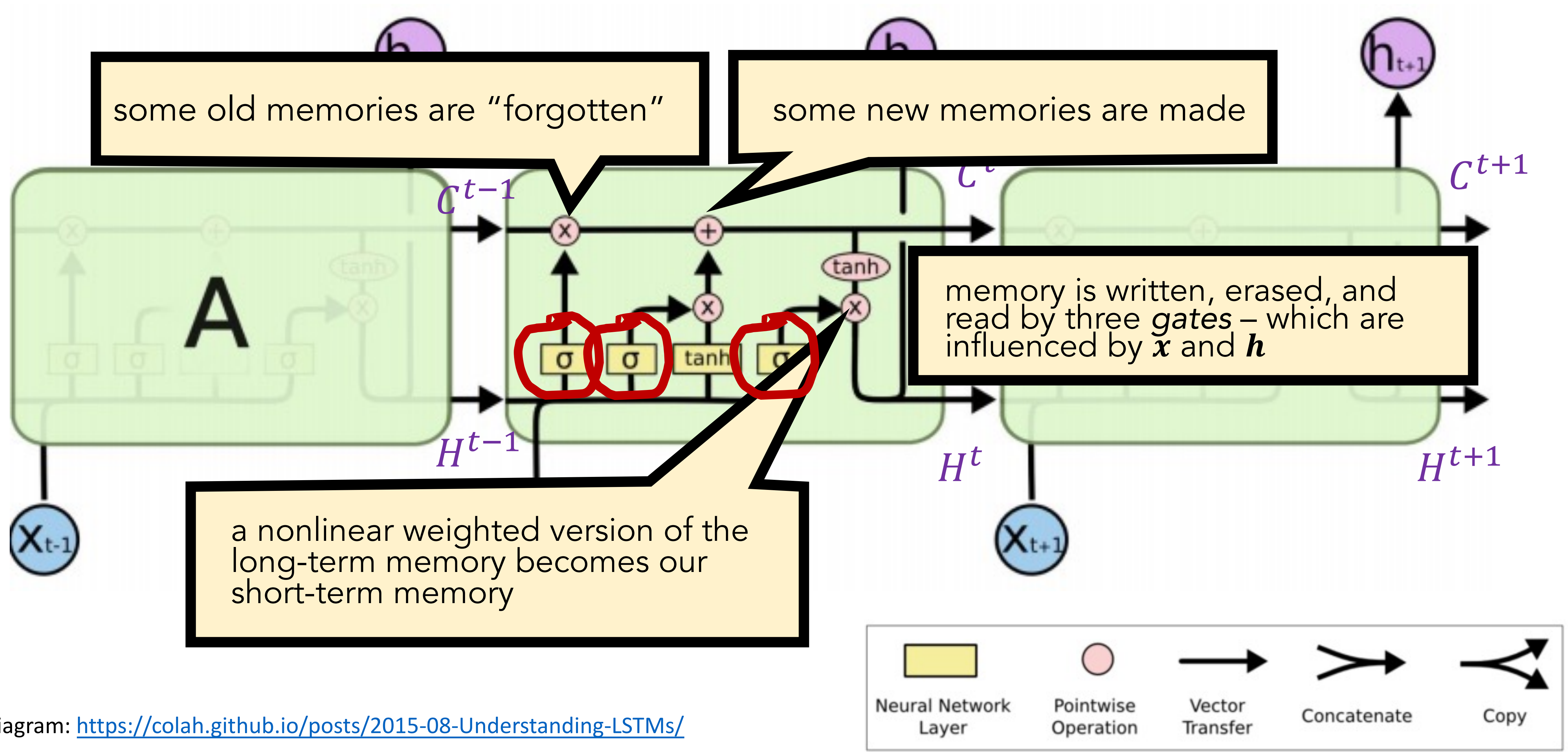


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RECAP: L6

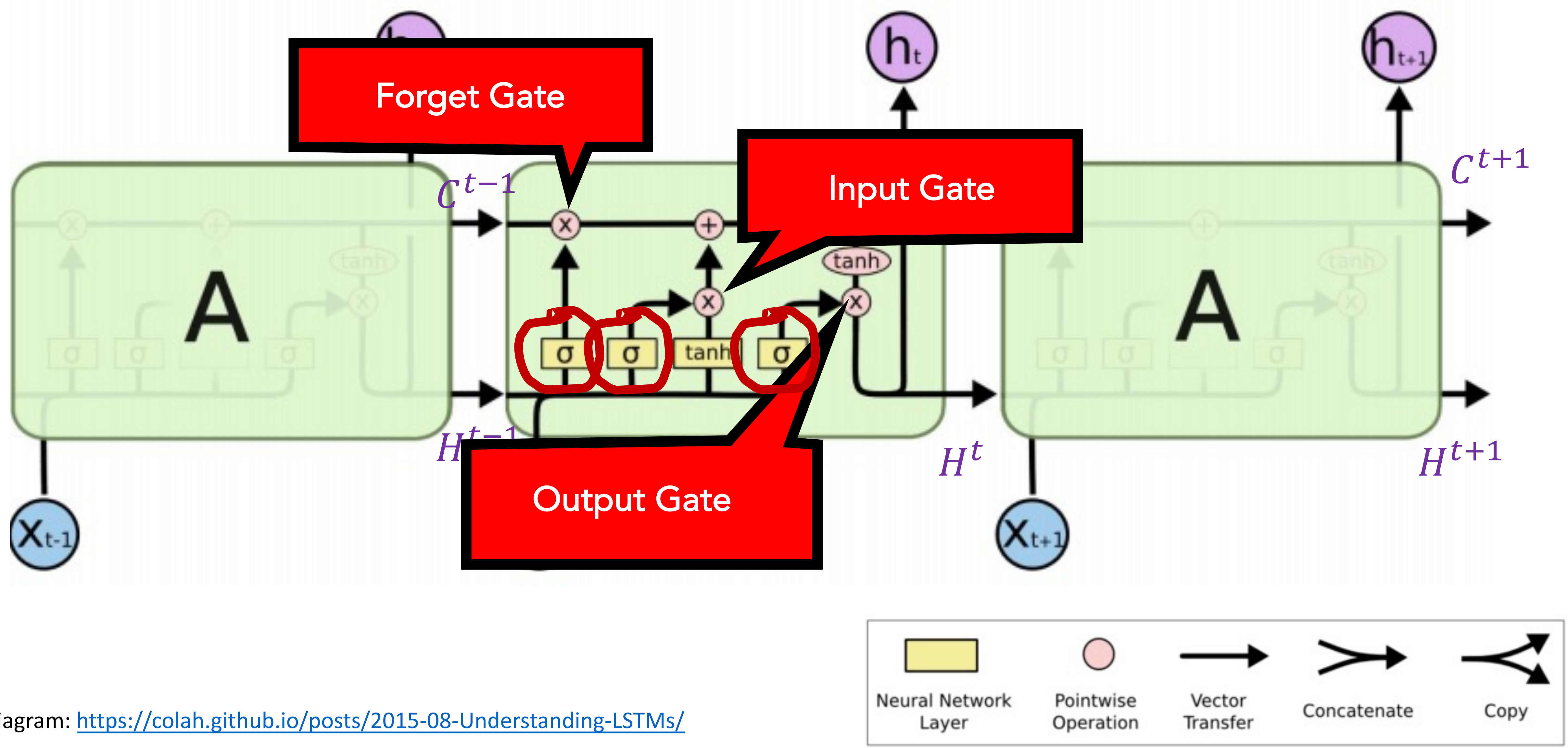


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

LSTMs

They've been around for a while, but essentially unused until 2014!

Juergen Schmidhuber

FOLLOW

The Swiss AI Lab, IDSIA, University of Lugano
Verified email at idsia.ch - Homepage

computer science artificial intelligence reinforcement learning neural networks physics

TITLE	CITED BY	YEAR
Long short-term memory S Hochreiter, J Schmidhuber Neural computation 9 (8), 1735-1780	54040	1997

LSTMs

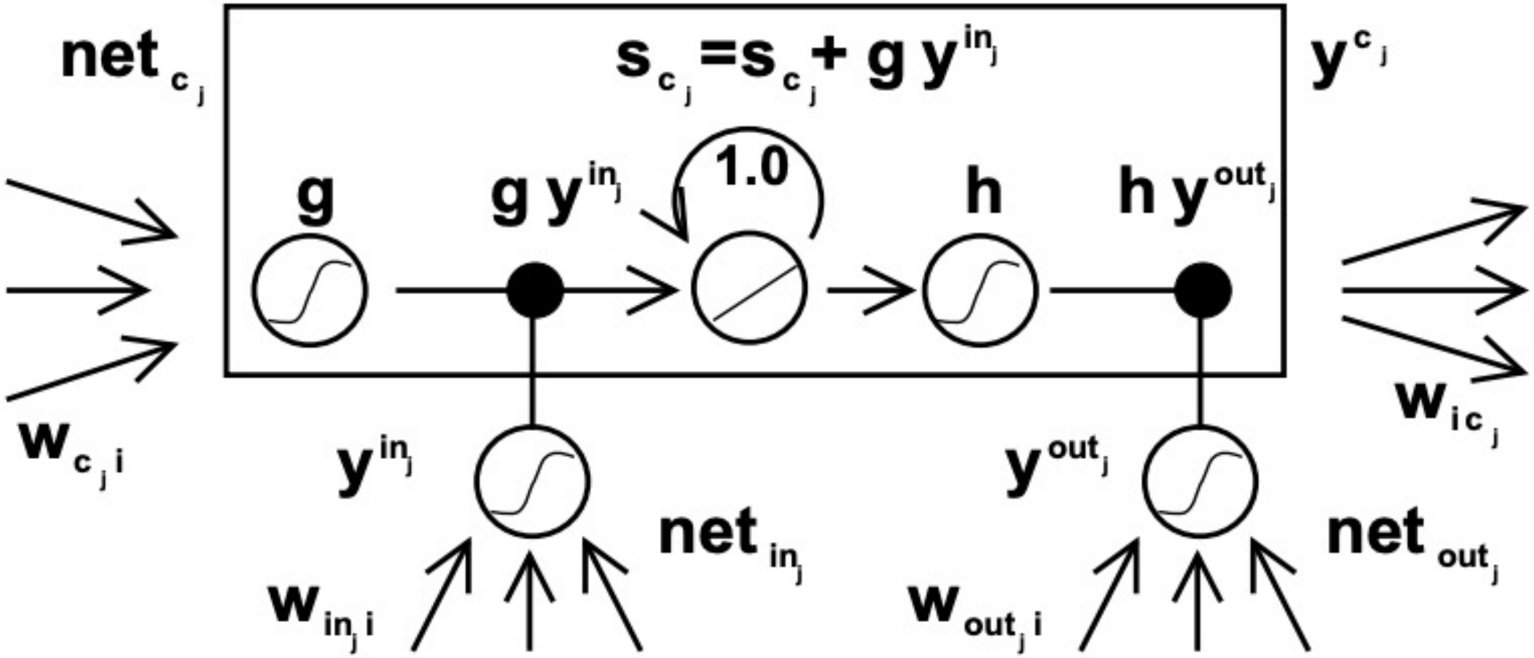


Figure 1: *Architecture of memory cell* c_j *(the box) and its gate units* in_j, out_j. *The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the "constant error carrousel" CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.*

Why gate units? To avoid input weight conflicts, in_j controls the error flow to memory cell c_j's input connections $w_{c_j i}$. To circumvent c_j's output weight conflicts, out_j controls the error flow from unit j's output connections. In other words, the net can use in_j to decide when to keep or override information in memory cell c_j, and out_j to decide when to access memory cell c_j and when to prevent other units from being perturbed by c_j (see Figure 1).

Long short-term memory. Hochreiter and Schmidhuber. Neural Computation (1997).

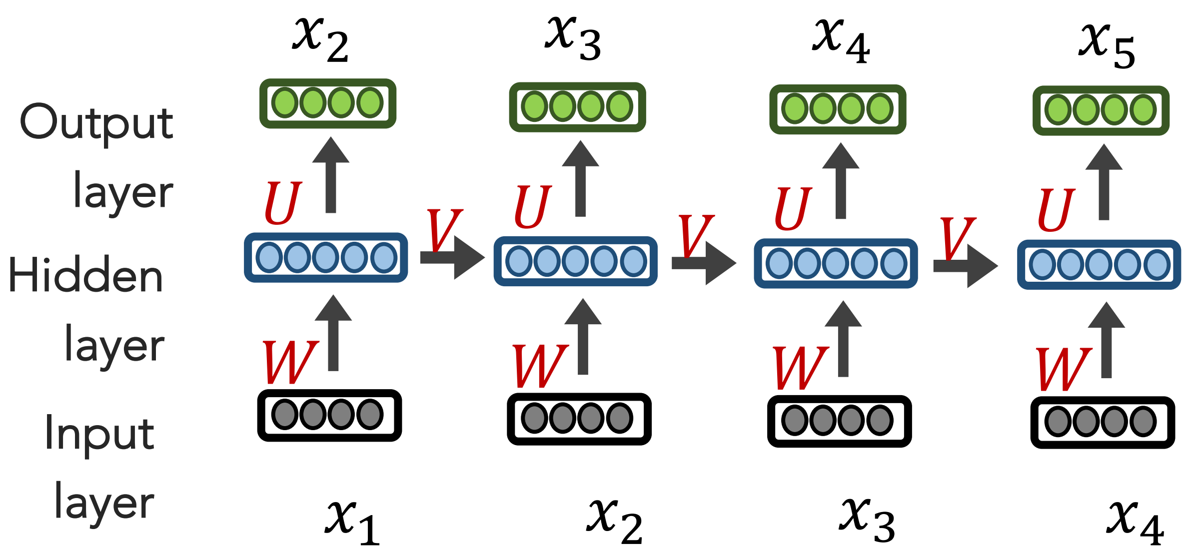
Sequential Modelling

IMPORTANT

If your goal isn't to predict the *next item* in a sequence, you could instead perform classification or regression task using the learned, hidden representations.

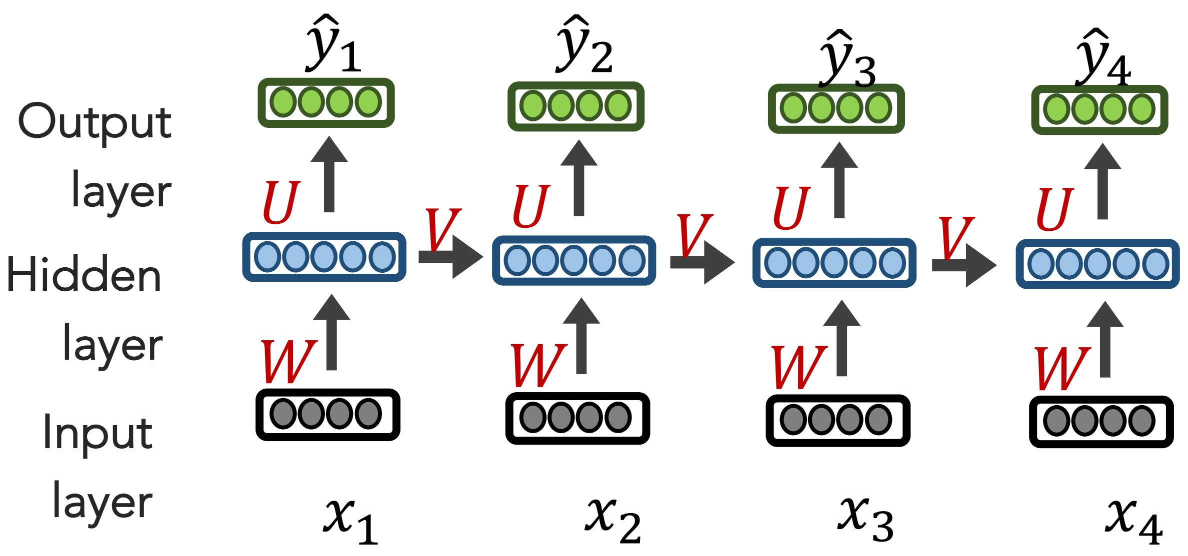
Sequential Modelling

Language Modelling



Auto-regressive

1-to-1 tagging/classification

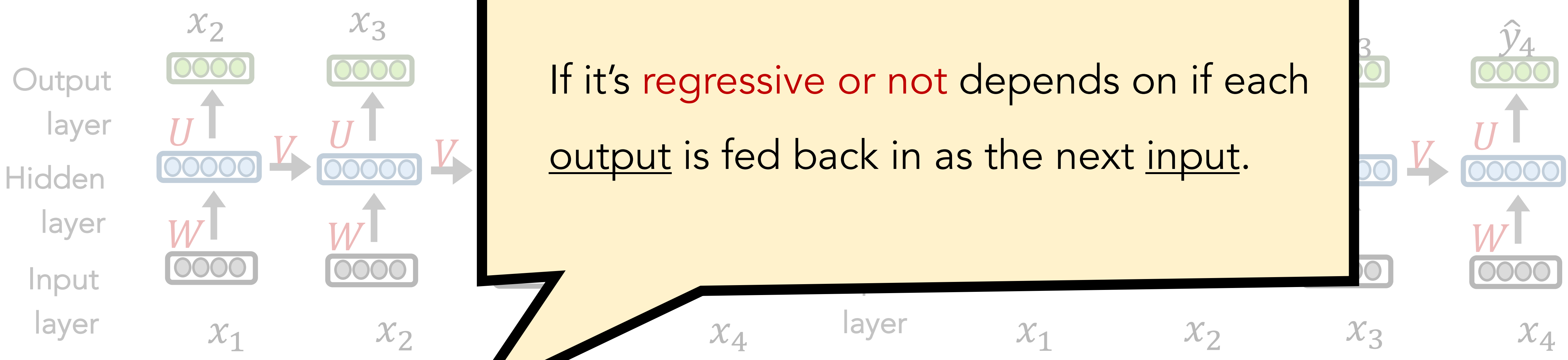


Non-Auto-regressive

Sequential Modelling

Language Modelling

1-to-1 tagging/classification



If it's regressive or not depends on if each output is fed back in as the next input.

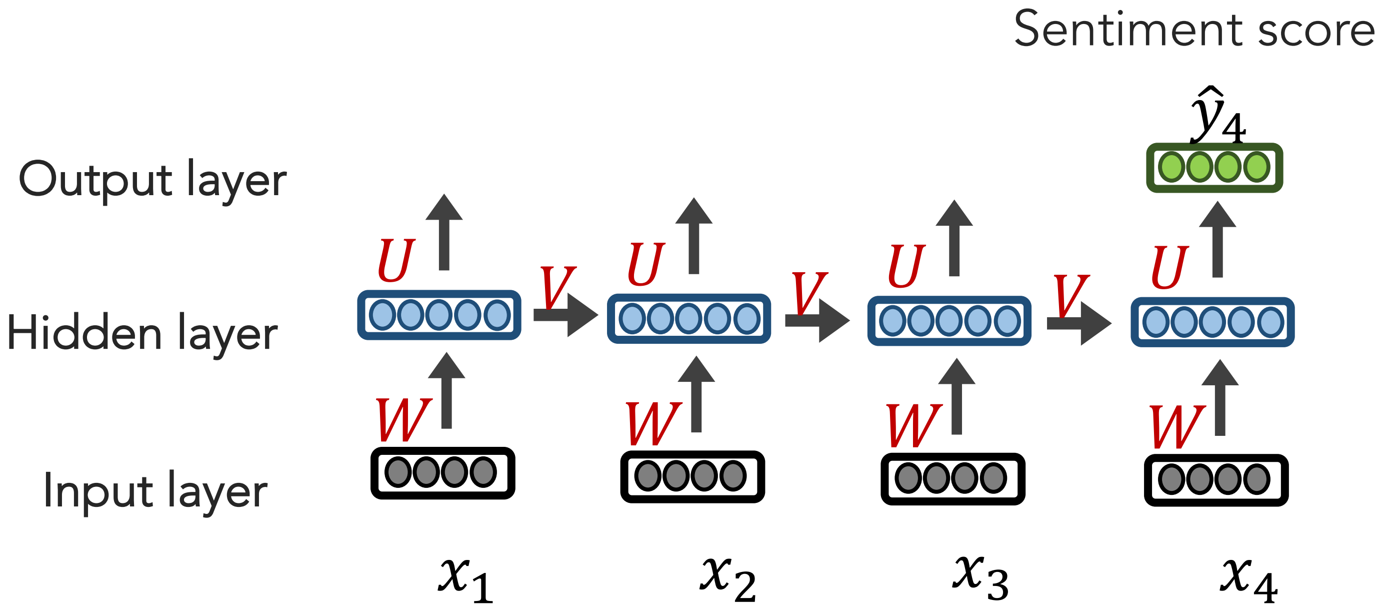
Auto-regressive

Non-Auto-regressive

Sequential Modelling

Regression
Binary classification
Multi-class classification

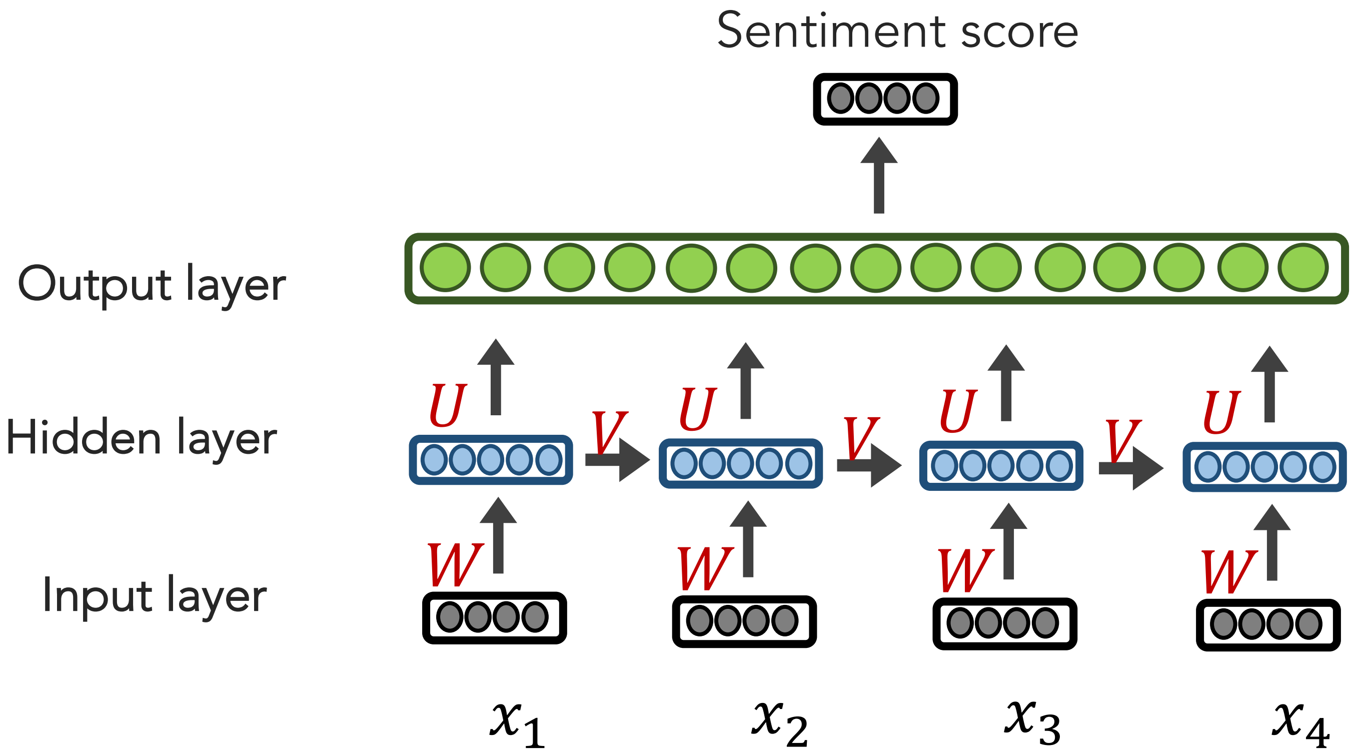
Many-to-1 classification



Sequential Modelling

Regression
Binary classification
Multi-class classification

Many-to-1 classification



Types of Prediction

	input	output
Regression	I love hiking!	0.9
Binary Classification	I love hiking!	Positive or negative
Multi-class Classification	I love hiking!	Very positive, positive, neutral, negative, or very negative
Structured Prediction (difficult scenario when your output has exponential/infinite # of possibilities)	I love hiking!	PRP VBP NN

Types of Prediction (an independent axis)

Unconditioned Prediction: predict some single variable. P(X)

Example: language modelling. X = "I like hiking!"

Conditioned Prediction: predict the probability of an output variable, given the input. P(Y|X)

Example: text classification. Y = positive. X = "I like hiking!"

Types of Prediction (an independent axis)

Unconditioned Prediction: predict some single variable. P(X)

Exa…: language modelling. X = “I like hiking!”

(un)conditioned is referring to if you’re entire model is predicated upon some particular input.

Conditione… …utput variable, given the input. P(Y|X)

Example: text classification. Y = positive. X = “I like hiking!”

Types of Prediction (an independent axis)

Unconditioned Prediction: predict some single variable. P(X)

Example: language modelling. X = "I like hiking!"

Cond… …an output variable,

given…

… = "I like hiking!"

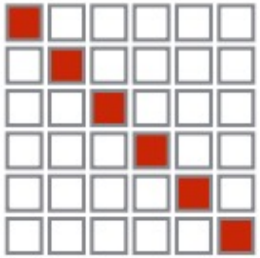
Language modelling is <u>unconditional prediction</u>, but one could do so by making use of **conditional probabilities** of X

Types of Unconditional Prediction

Independent Prediction

$$P(X) = \prod_{i=1}^{|X|} P(x_i)$$

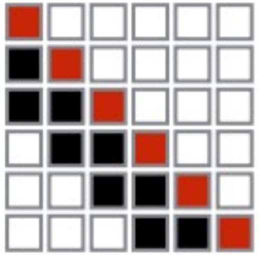
(e.g. unigram model)



Left-to-right Markov Chain (order n-1*)*

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_{i-n+1}, \ldots, x_{i-1})$$

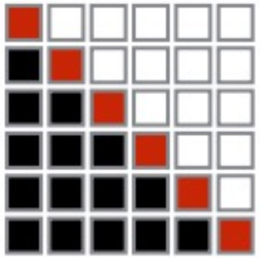
(e.g. n-gram LM, feed-forward LM)



Left-to-right Autoregressive Prediction

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \ldots, x_{i-1})$$

(e.g. RNN LM)

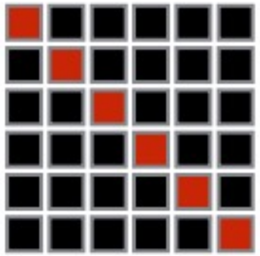


Types of Unconditional Prediction

Bidirectional Prediction

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

(e.g. masked language model)



Formally, a **language model** estimates the probability of a sequence, so this is illegal. It cheats in a manner that we call them **masked language models** (not proper prob. dist and they don't estimate sequences)

Types of Conditional Prediction

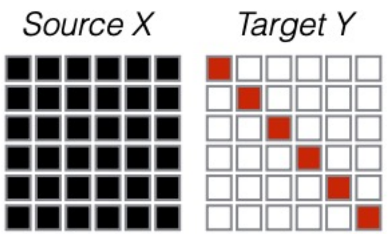
Many-to-1 classification

$$P(y|X)$$

Many-to-many classification

Non-autoregressive Conditioned Prediction

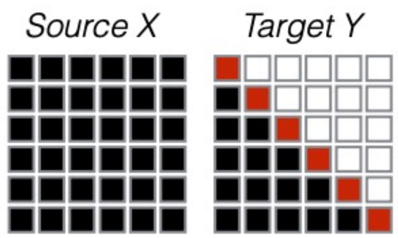
$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X)$$



(e.g. sequence labeling, non-autoregressive MT)

Autoregressive Conditioned Prediction

$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X, y_1, \ldots, y_{i-1})$$



(e.g. seq2seq model)

This concludes the foundation in sequential representation.

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.

(aka this is where things get crazy.)

Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

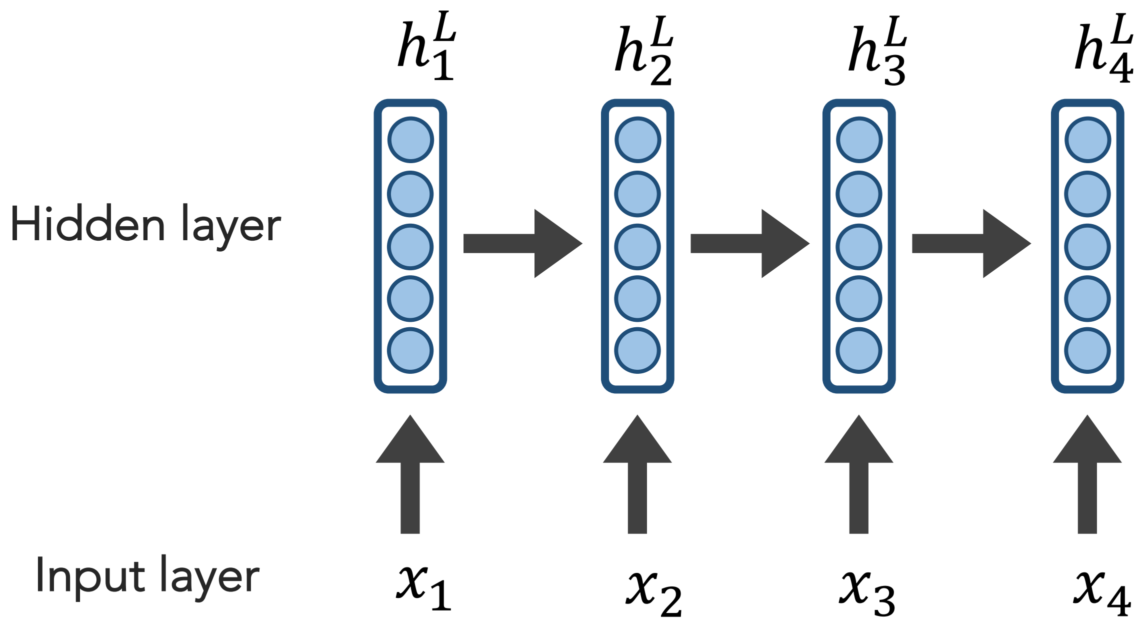
RNN Extensions: Bi-directional LSTMs

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?

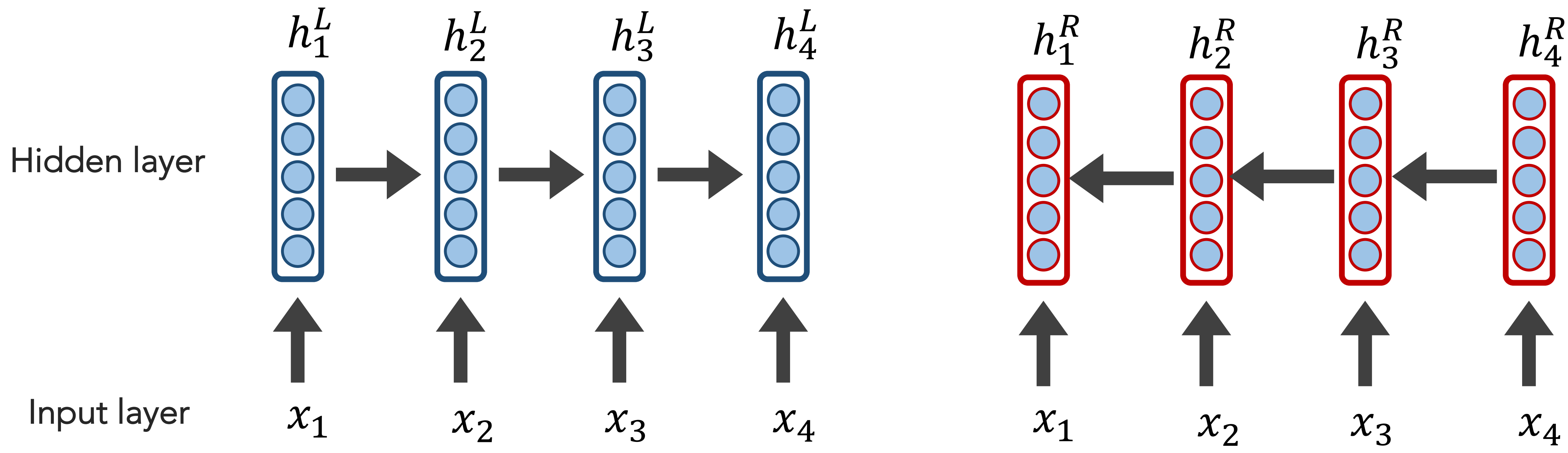
RNN Extensions: Bi-directional LSTMs

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

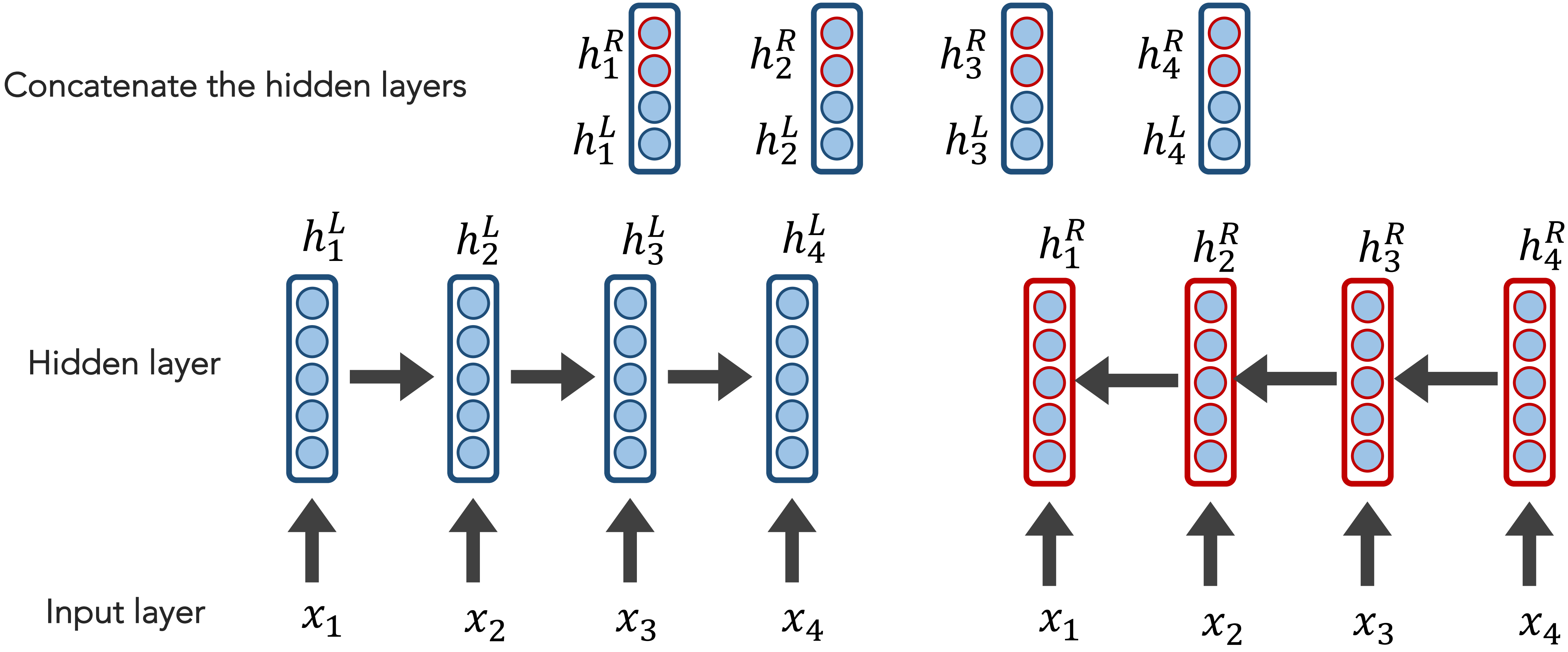


RNN Extensions: Bi-directional LSTMs

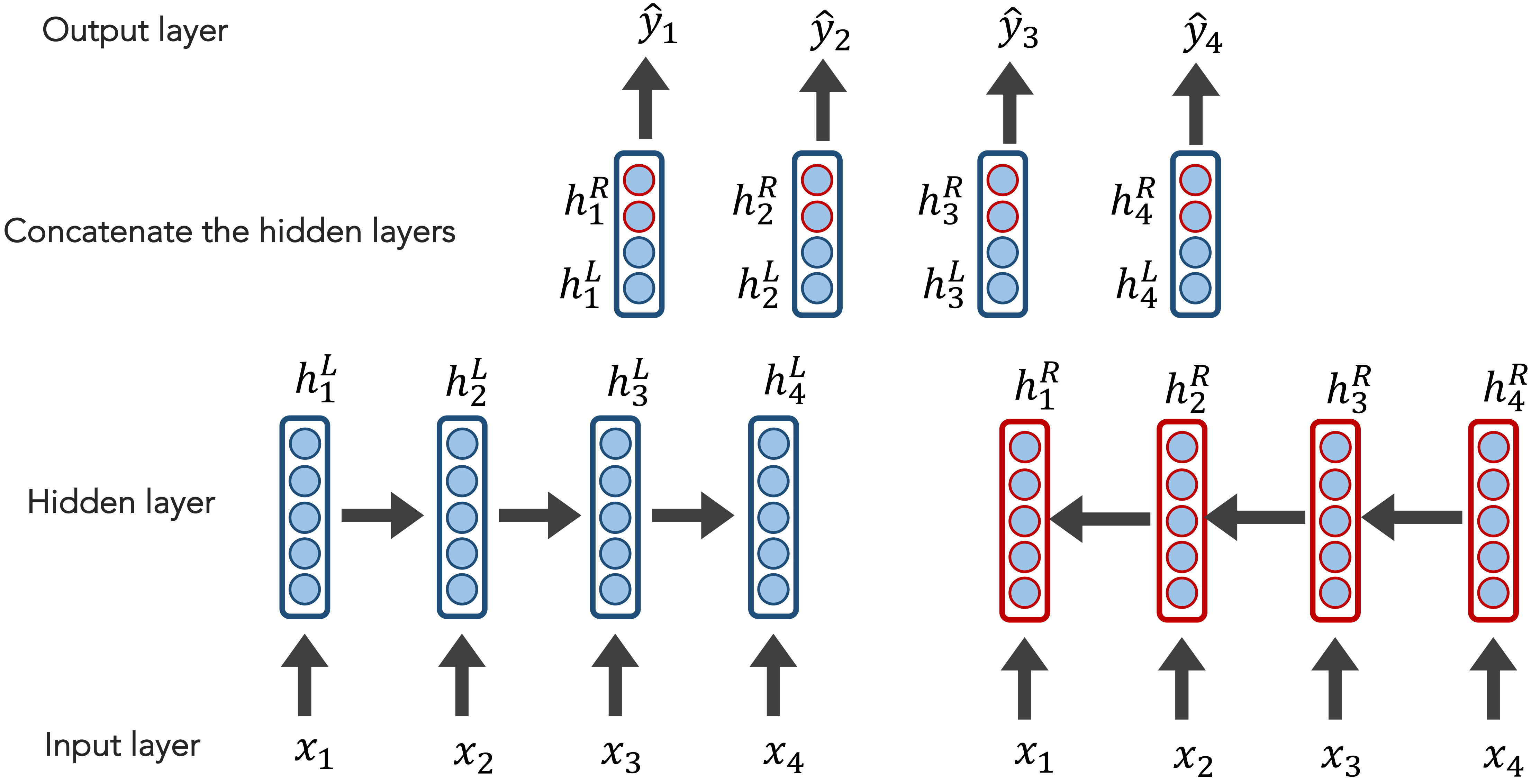
For brevity, let's use the follow schematic to represent an RNN



RNN Extensions: Bi-directional LSTMs



RNN Extensions: Bi-directional LSTMs



RNN Extensions: Bi-directional LSTMs

BI-LSTM STRENGTHS?

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

BI-LSTM ISSUES?

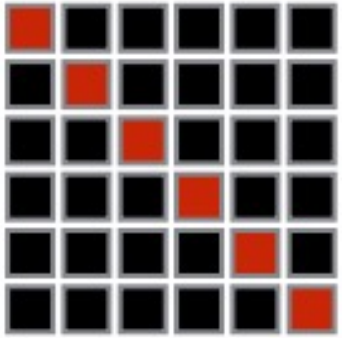
- Slower to train
- Only possible if access to full data is allowed

Type of Unconditional Prediction

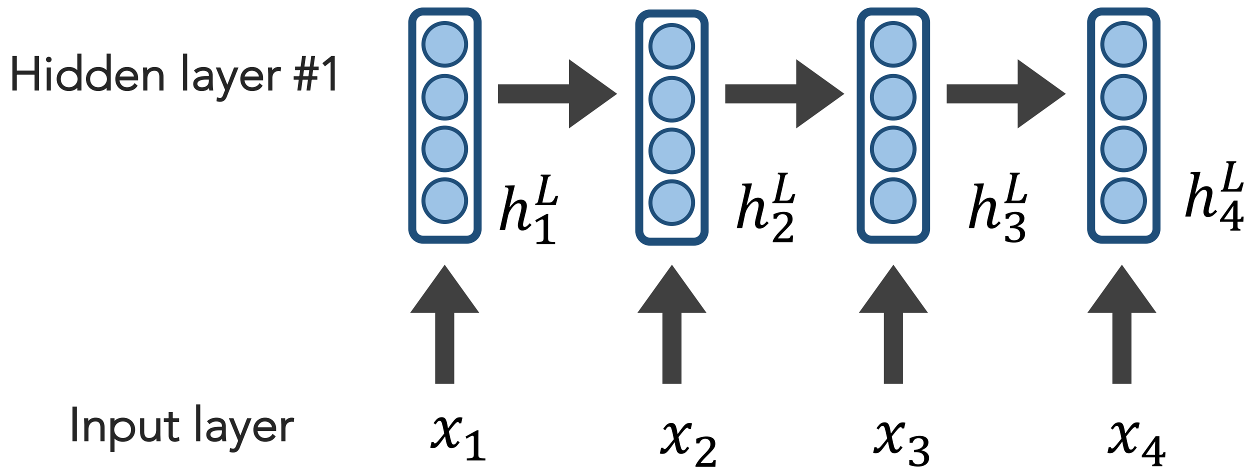
Bidirectional Prediction

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

(e.g. masked language model)



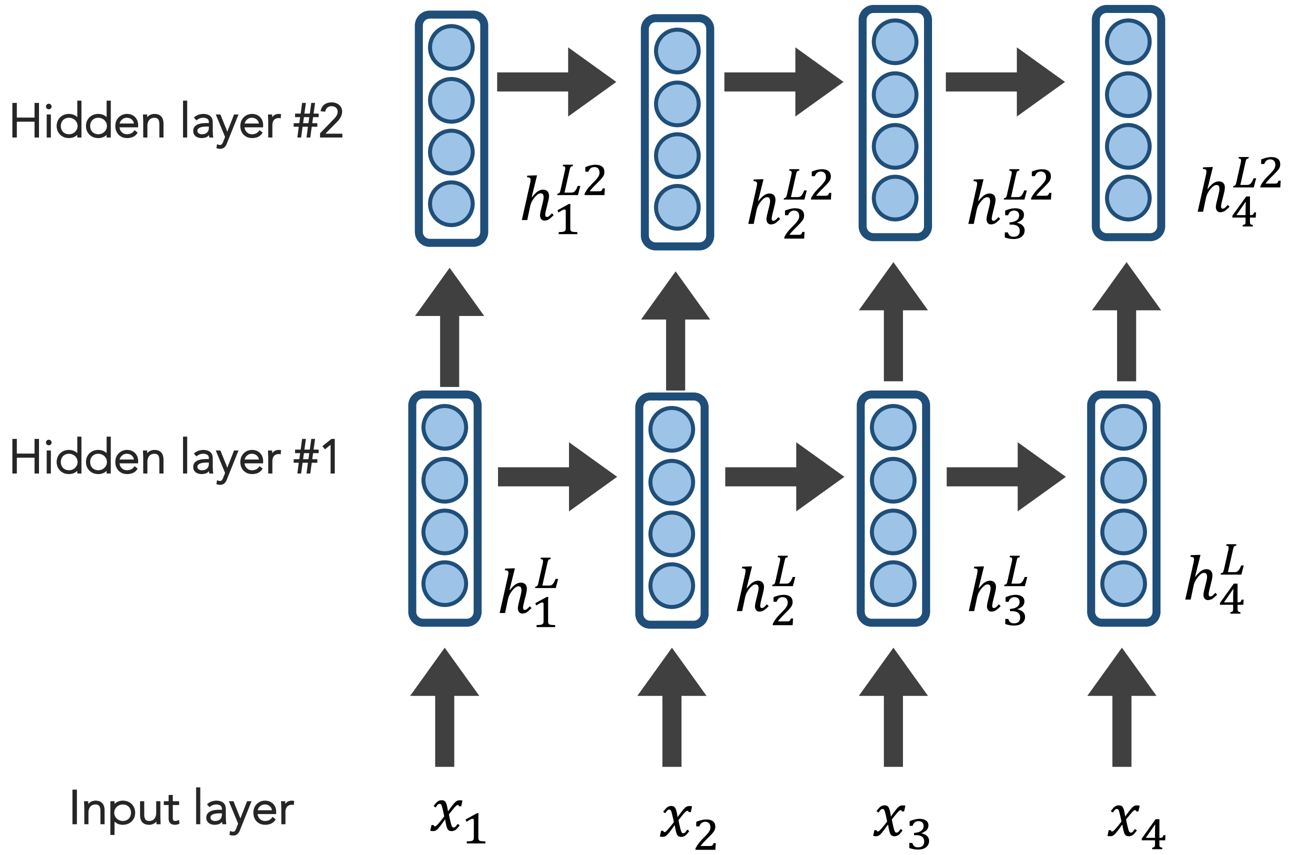
RNN Extensions: Stacked LSTMs



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

Stacking hidden layers provides increased abstractions.

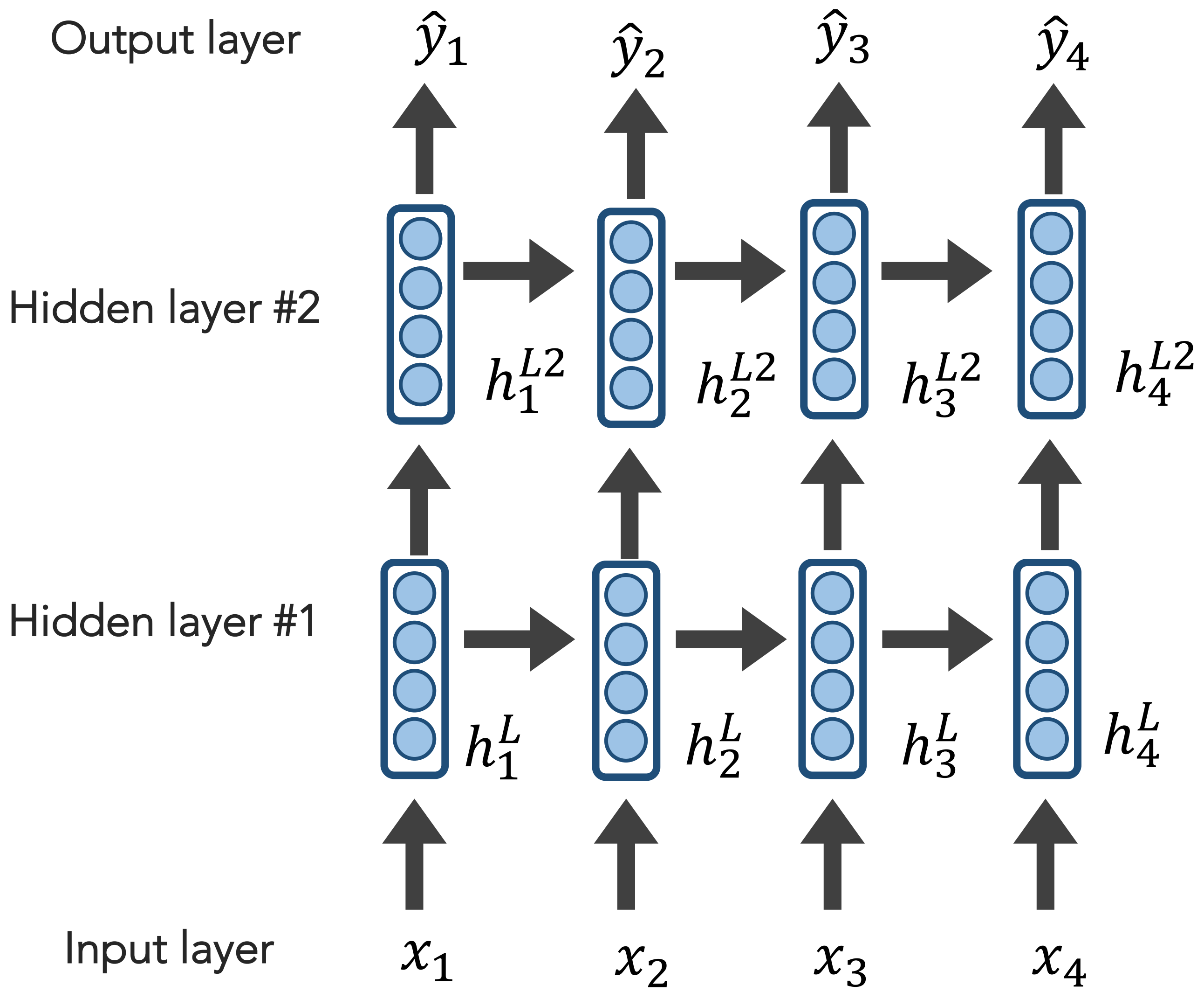
RNN Extensions: Stacked LSTMs



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

Stacking hidden layers provides increased abstractions.

RNN Extensions: Stacked LSTMs



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

Stacking hidden layers provides increased abstractions.

ELMo: Stacked Bi-directional LSTMs

General Idea:

- Goal is to get highly rich, contextualized embeddings (word tokens)
- 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: Stacked Bi-directional LSTMs

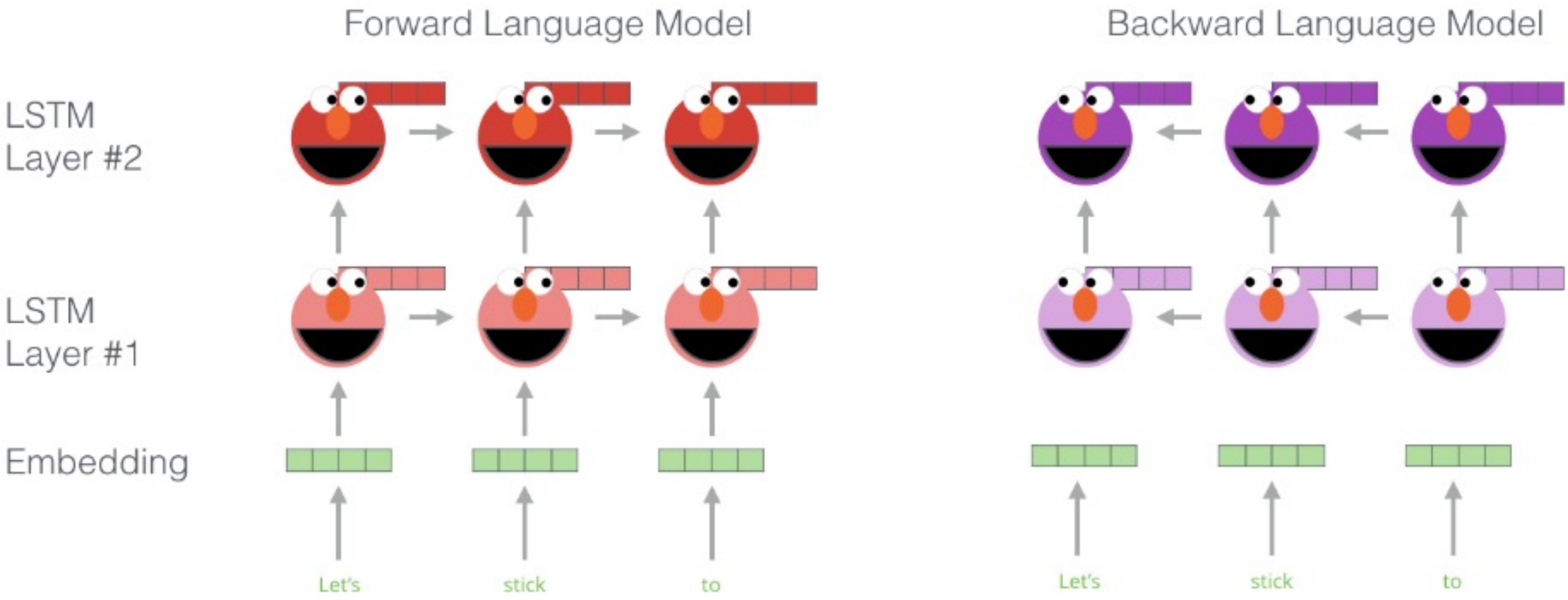


Illustration: http://jalammar.github.io/illustrated-bert/

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

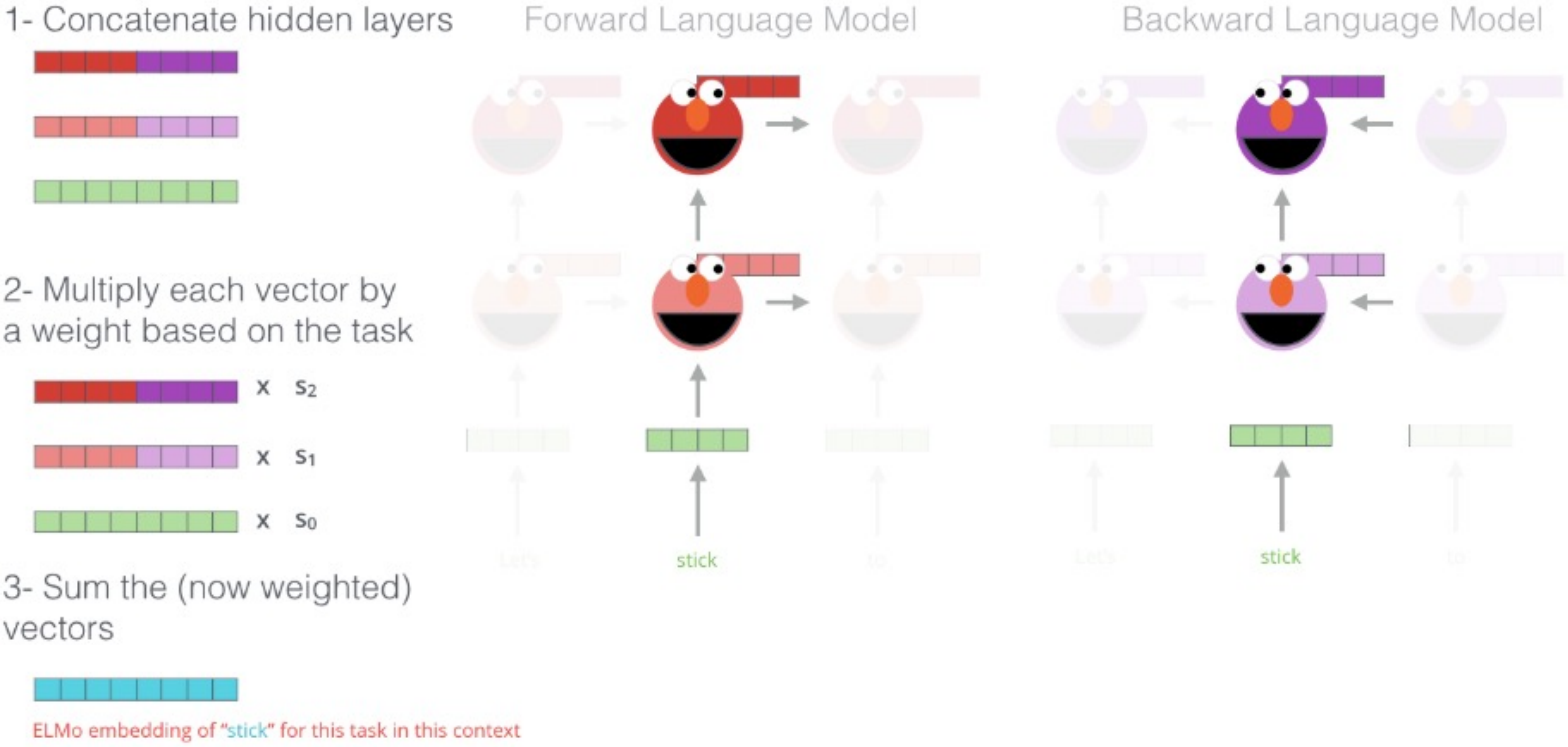


Illustration: http://jalammar.github.io/illustrated-bert/

ELMo: Stacked Bi-directional LSTMs

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Deep contextualized word representations. Peters et al. NAACL 2018.

ELMo: Stacked Bi-directional LSTMs

Model	F_1
WordNet 1st Sense Baseline	65.9
Raganato et al. (2017a)	69.9
Iacobacci et al. (2016)	**70.1**
CoVe, First Layer	59.4
CoVe, Second Layer	*64.7*
biLM, First layer	67.4
biLM, Second layer	*69.0*

Table 5: All-words fine grained WSD F_1. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

Model	Acc.
Collobert et al. (2011)	97.3
Ma and Hovy (2016)	97.6
Ling et al. (2015)	**97.8**
CoVe, First Layer	*93.3*
CoVe, Second Layer	92.8
biLM, First Layer	*97.3*
biLM, Second Layer	96.8

Table 6: Test set POS tagging accuracies for PTB. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

The higher layer seems to learn **semantics** while the lower layer probably captured **syntactic** features

Deep contextualized word representations. Peters et al. NAACL 2018.

ELMo: Stacked Bi-directional LSTMs

- ELMo yielded incredibly good contextualized embeddings, which yielded SOTA results when applied to many NLP tasks.

- Main ELMo takeaway: given enough training data, having tons of explicit connections between your vectors is useful
(system can determine how to best use context)

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

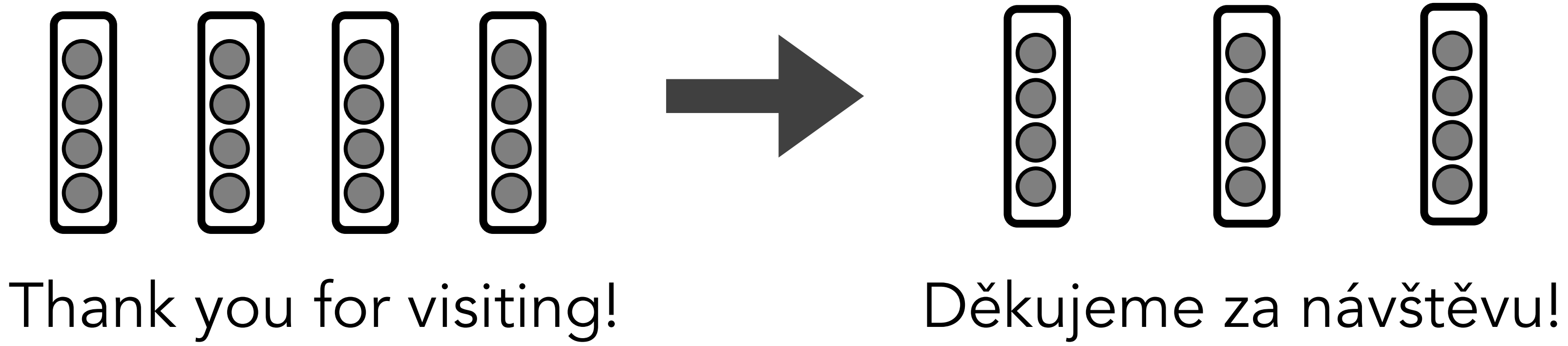
RECAP

- Language Modelling may help us for other tasks
- LSTMs do a great job of capturing "*meaning*", which can be used for almost every task
  - Given a sequence of N words, we can produce 1 output
  - Given a sequence of N words, we can produce N outputs

RECAP

- Language Modelling may help us for other tasks
- LSTMs do a great job of capturing "*meaning*", which can be used for almost every task
  - Given a sequence of N words, we can produce 1 output
  - Given a sequence of N words, we can produce N outputs
  - What if we wish to have M outputs?

We want to produce a **variable-length** output
(e.g., n → m predictions)



Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention

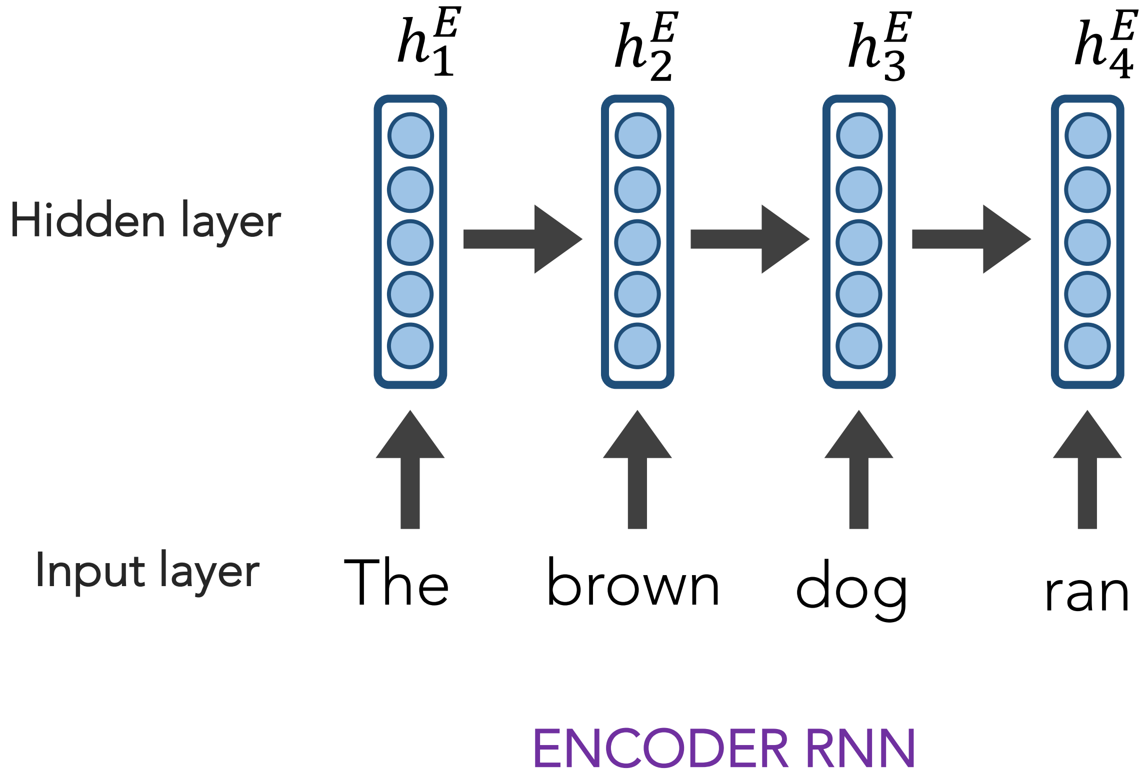
Sequence-to-Sequence (seq2seq)

- 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**)

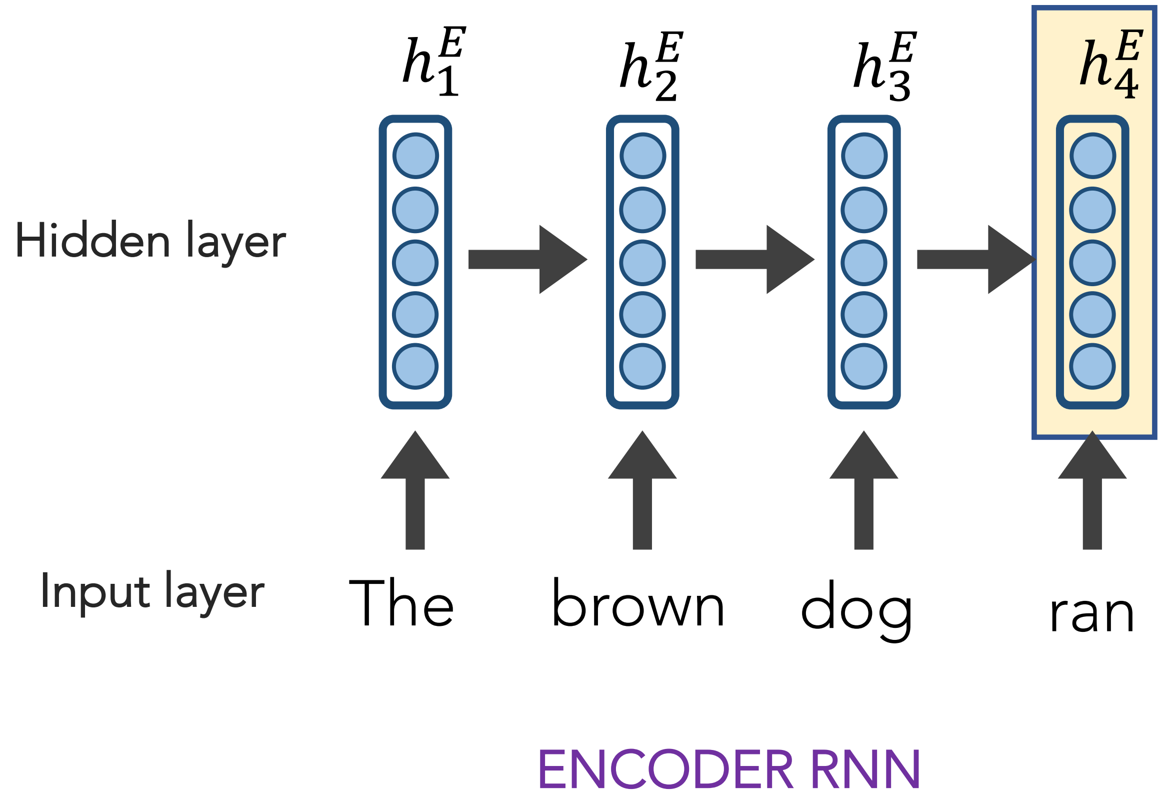
- seq2seq models are comprised of **2 RNNs**: 1 encoder, 1 decoder

Sequence-to-Sequence (seq2seq)



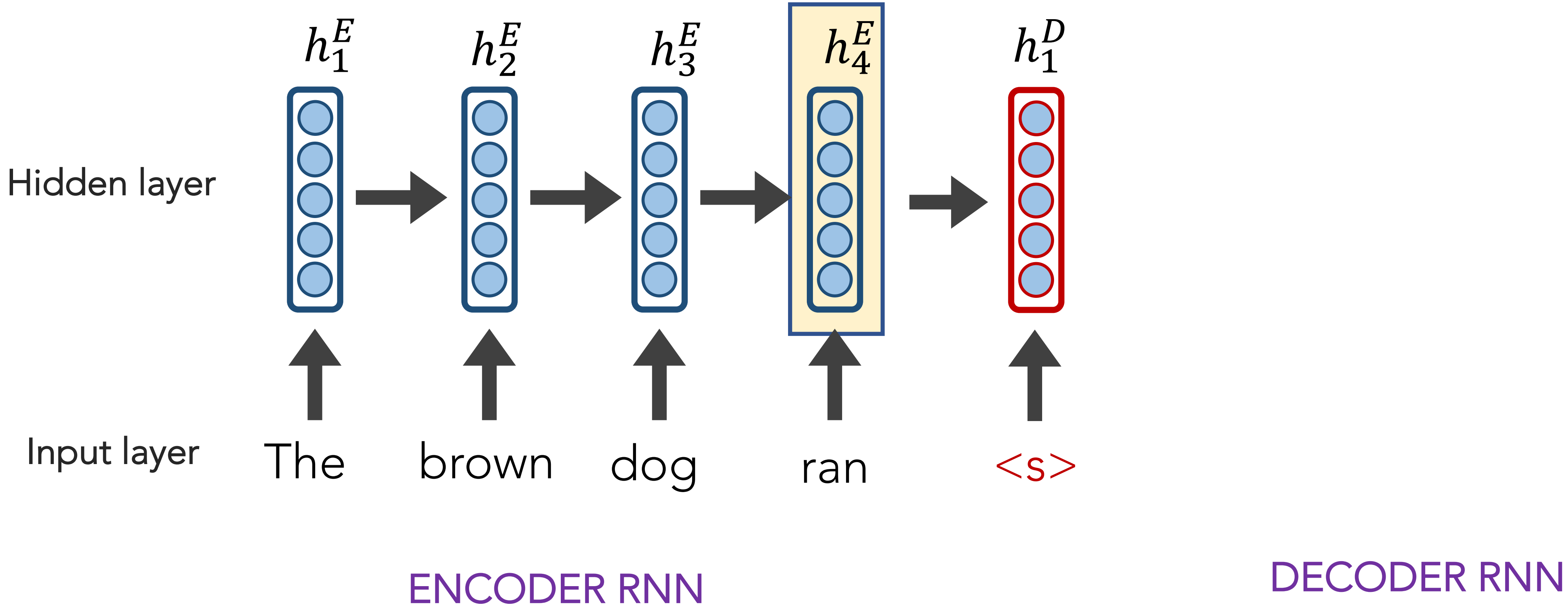
Sequence-to-Sequence (seq2seq)

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



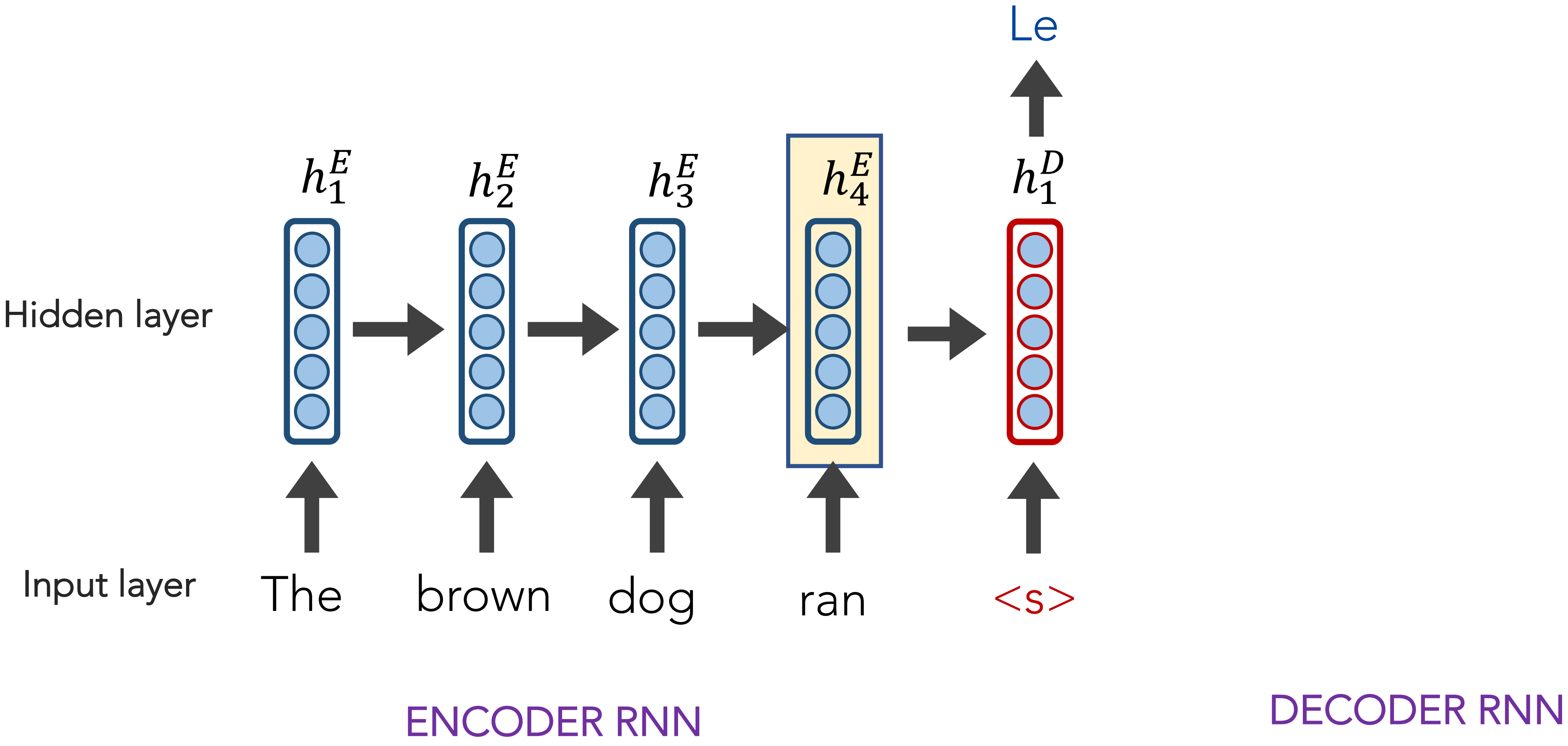
Sequence-to-Sequence (seq2seq)

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



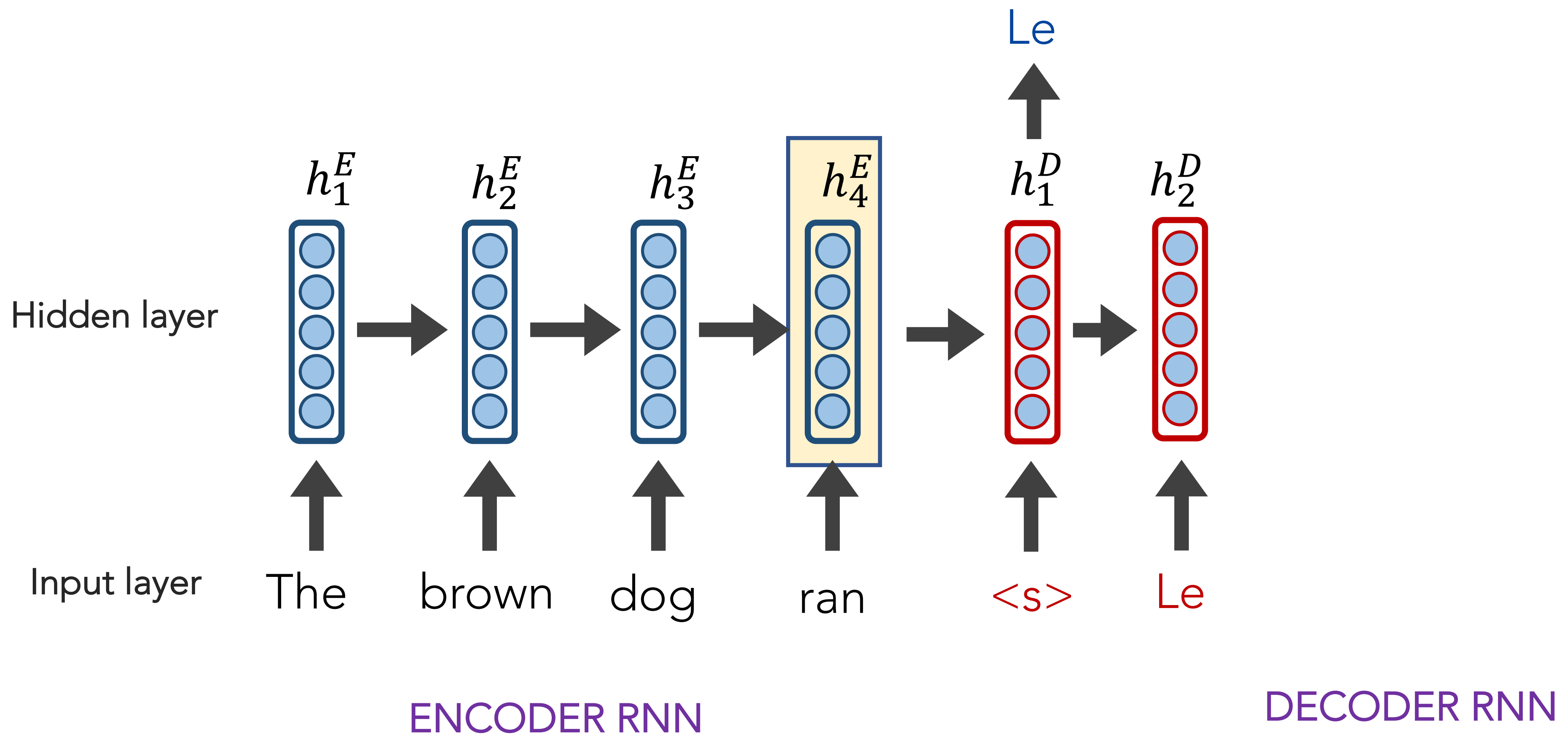
Sequence-to-Sequence (seq2seq)

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



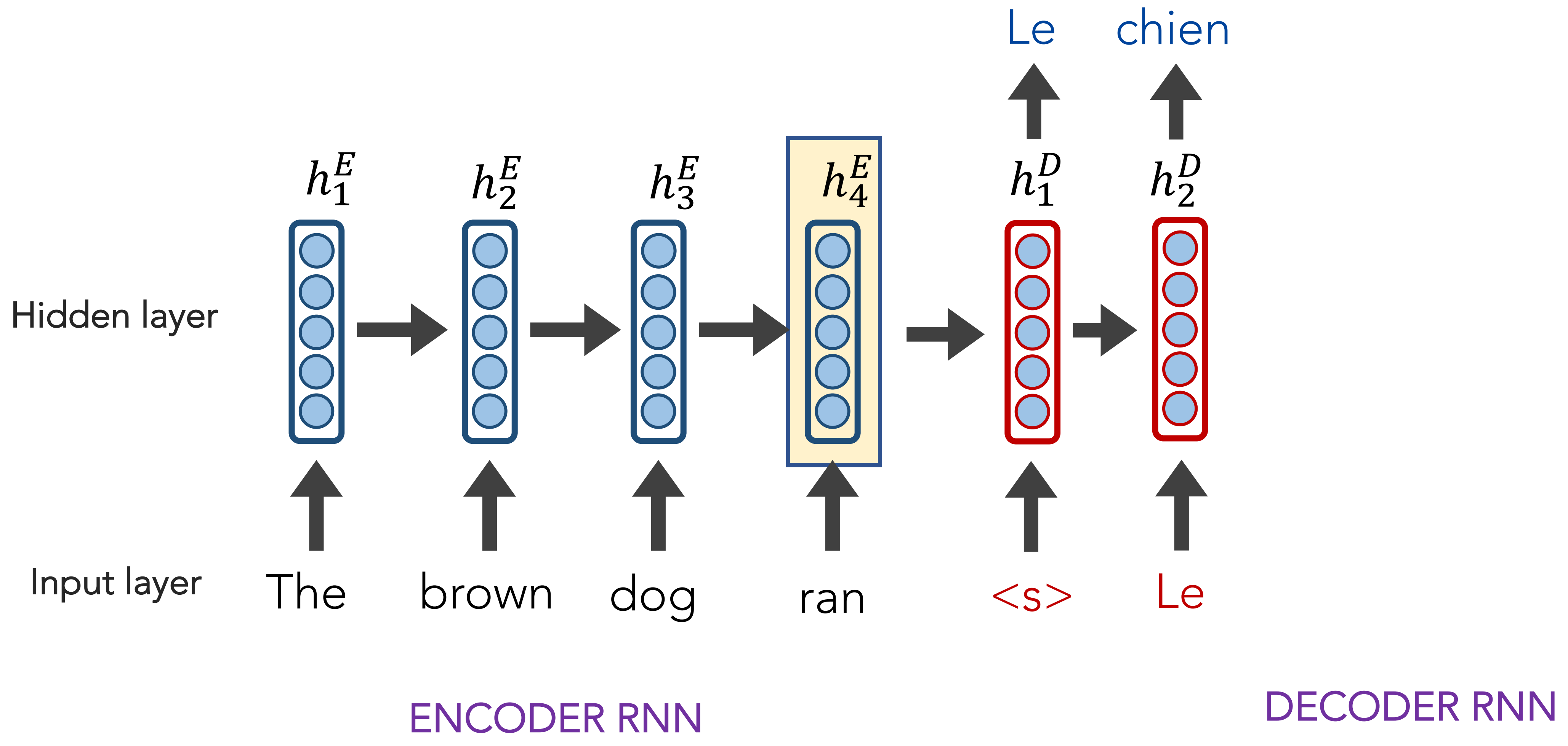
Sequence-to-Sequence (seq2seq)

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

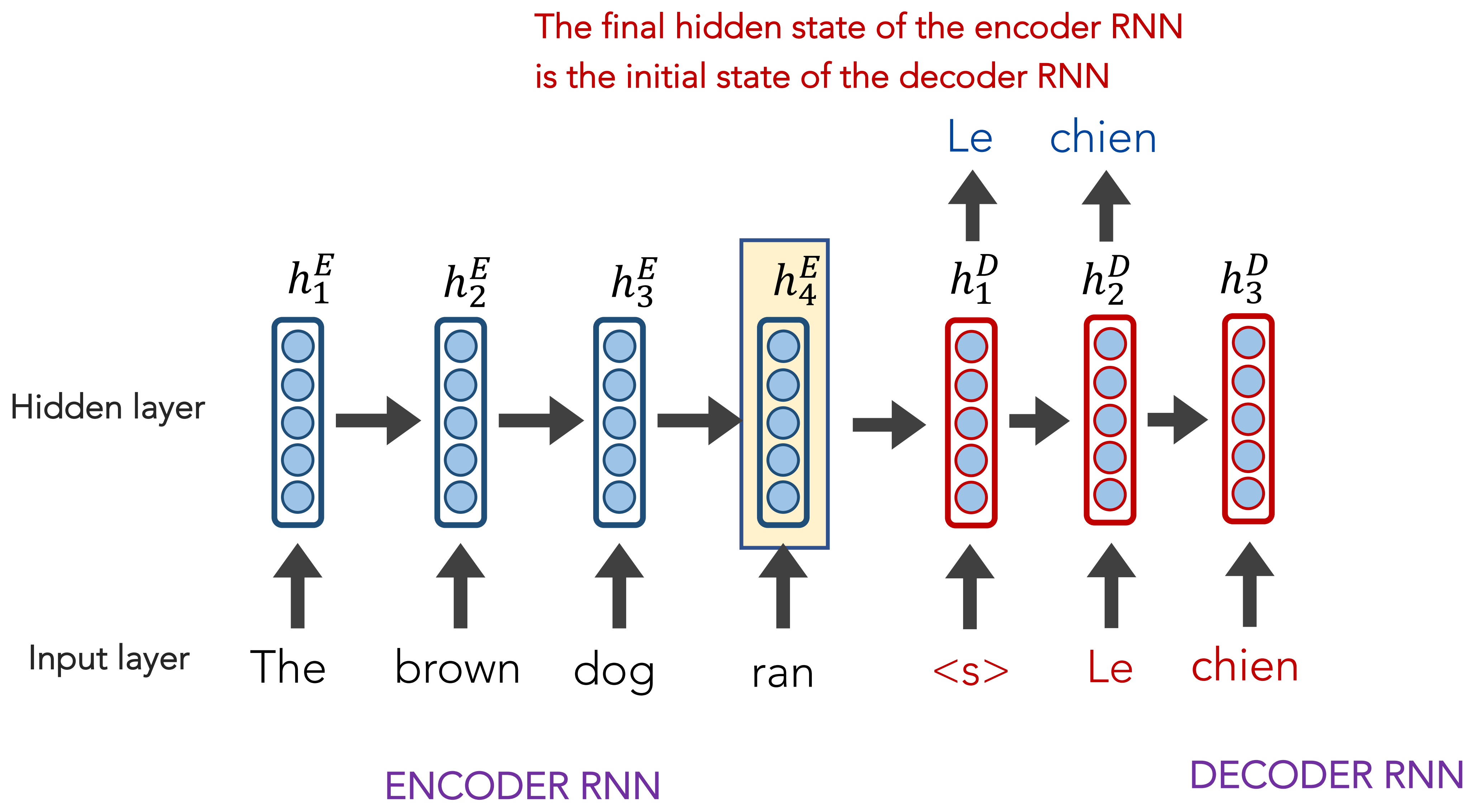


Sequence-to-Sequence (seq2seq)

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

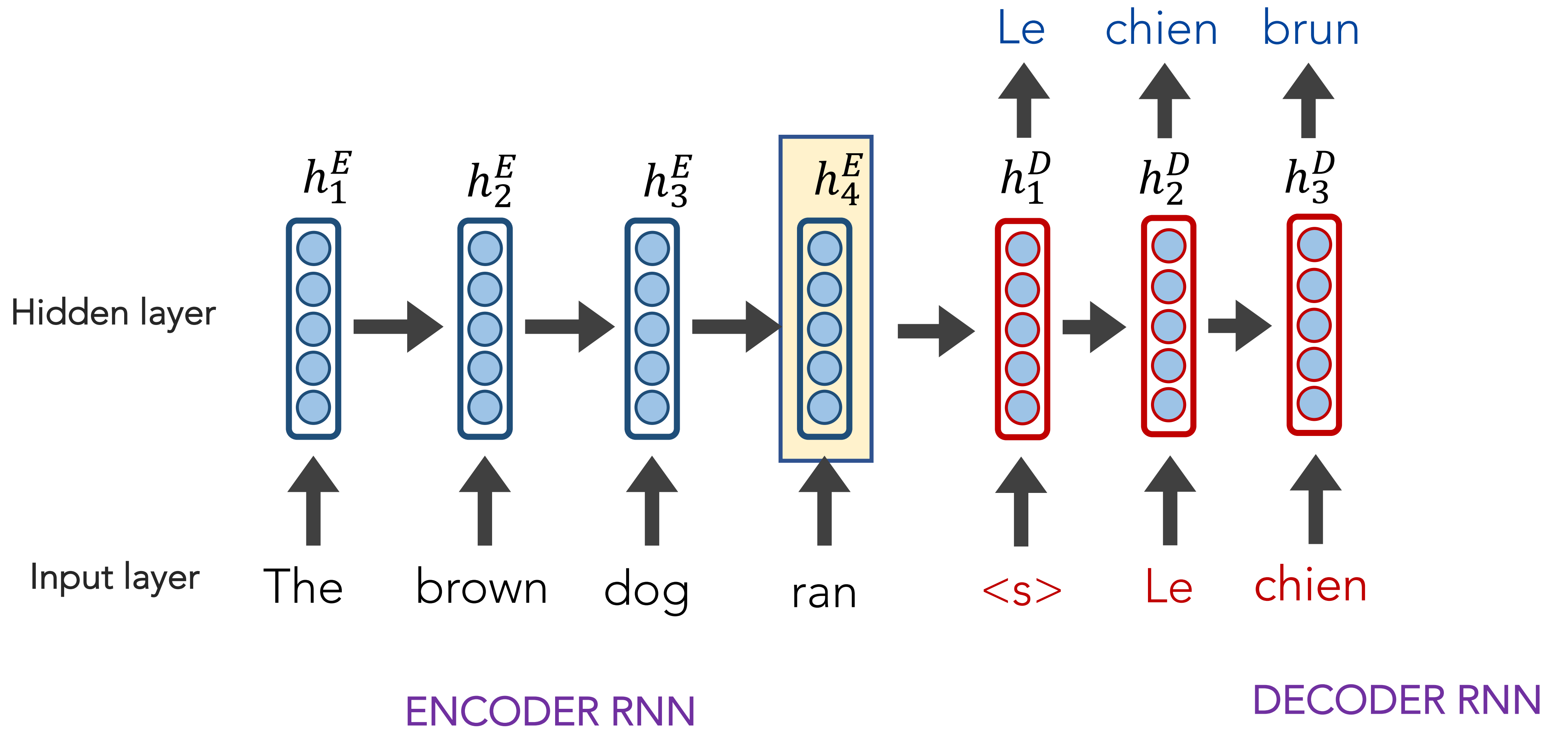


Sequence-to-Sequence (seq2seq)

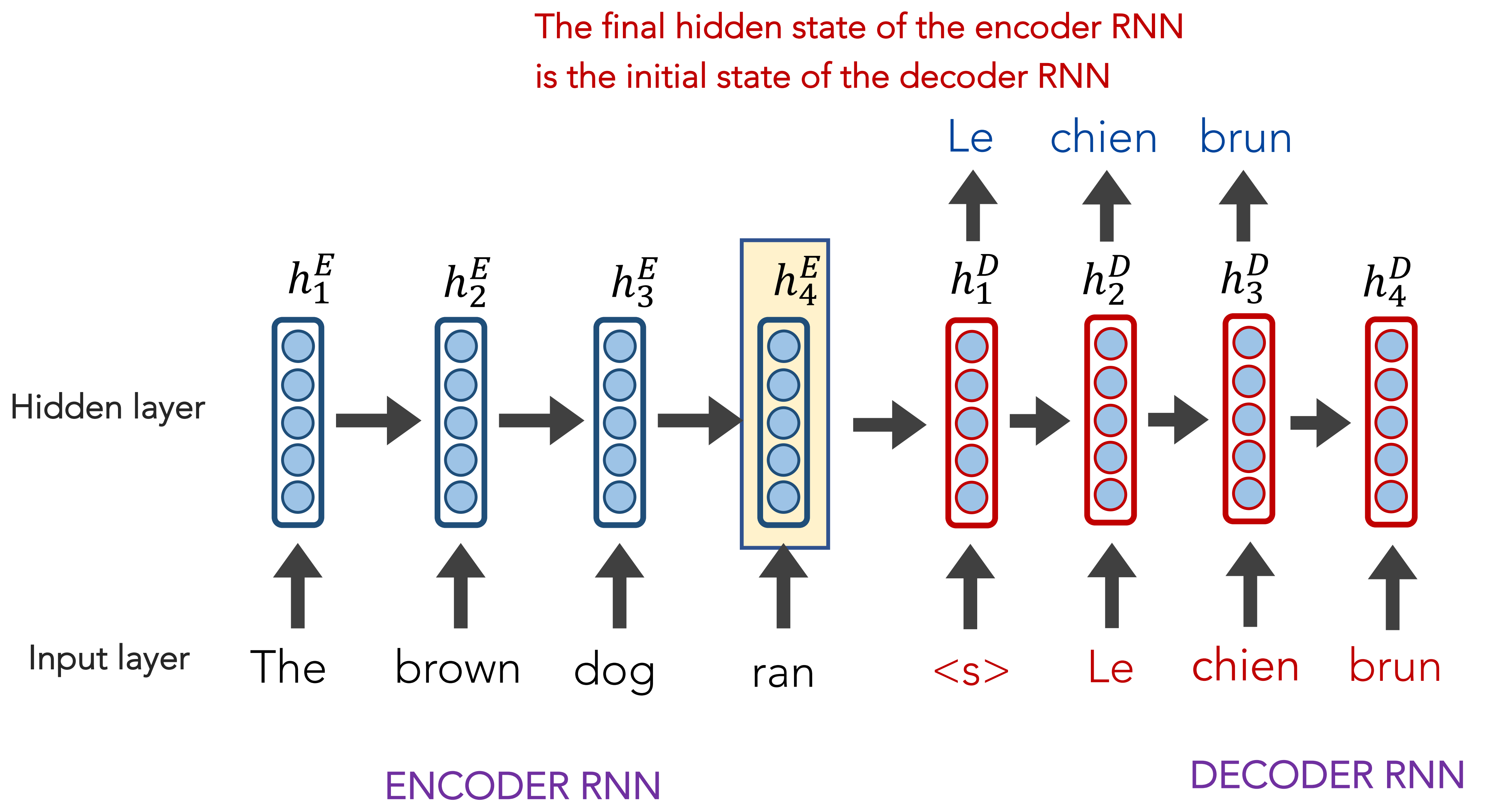


Sequence-to-Sequence (seq2seq)

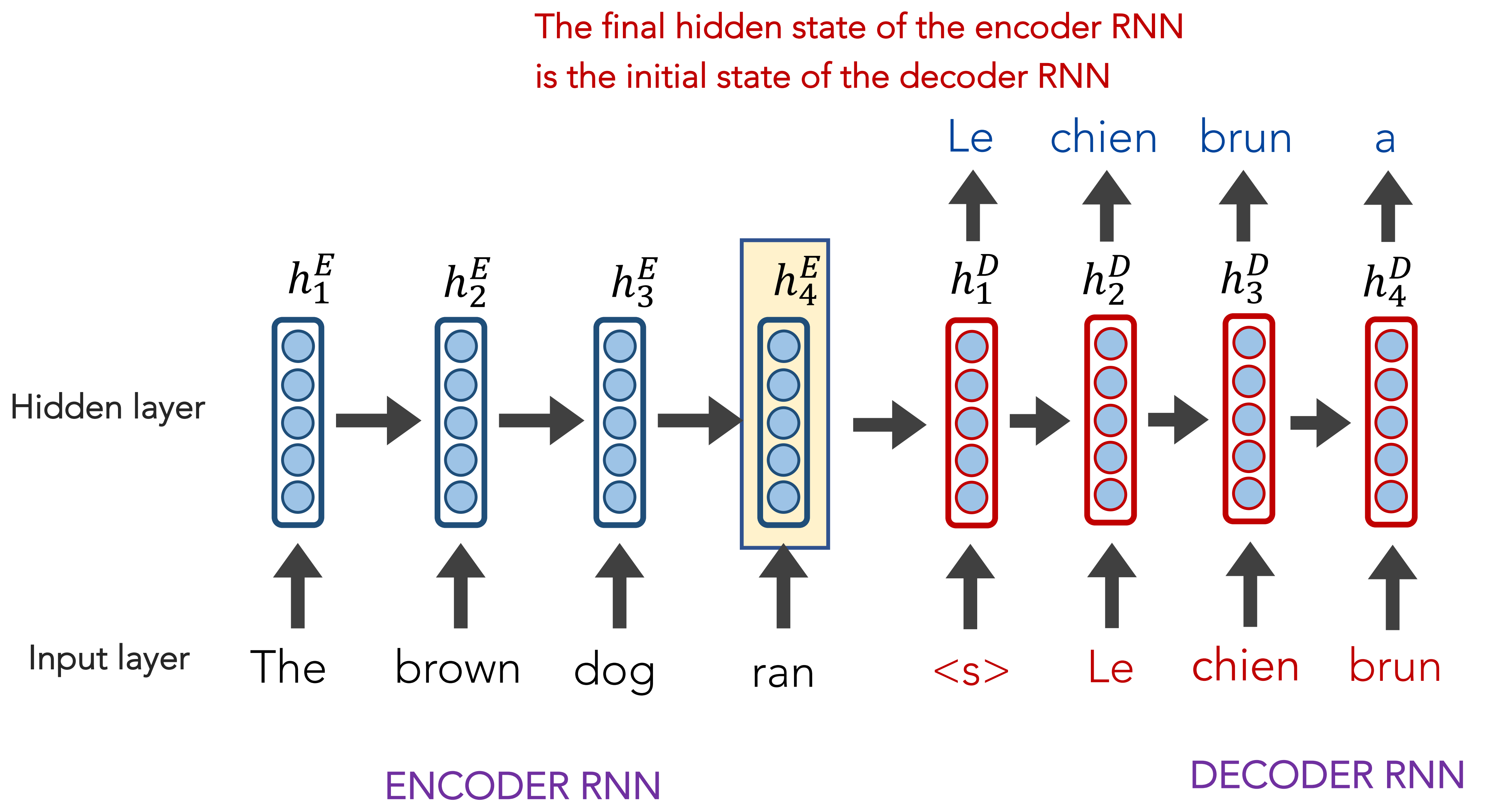
The final hidden state of the encoder RNN
is the initial state of the decoder RNN



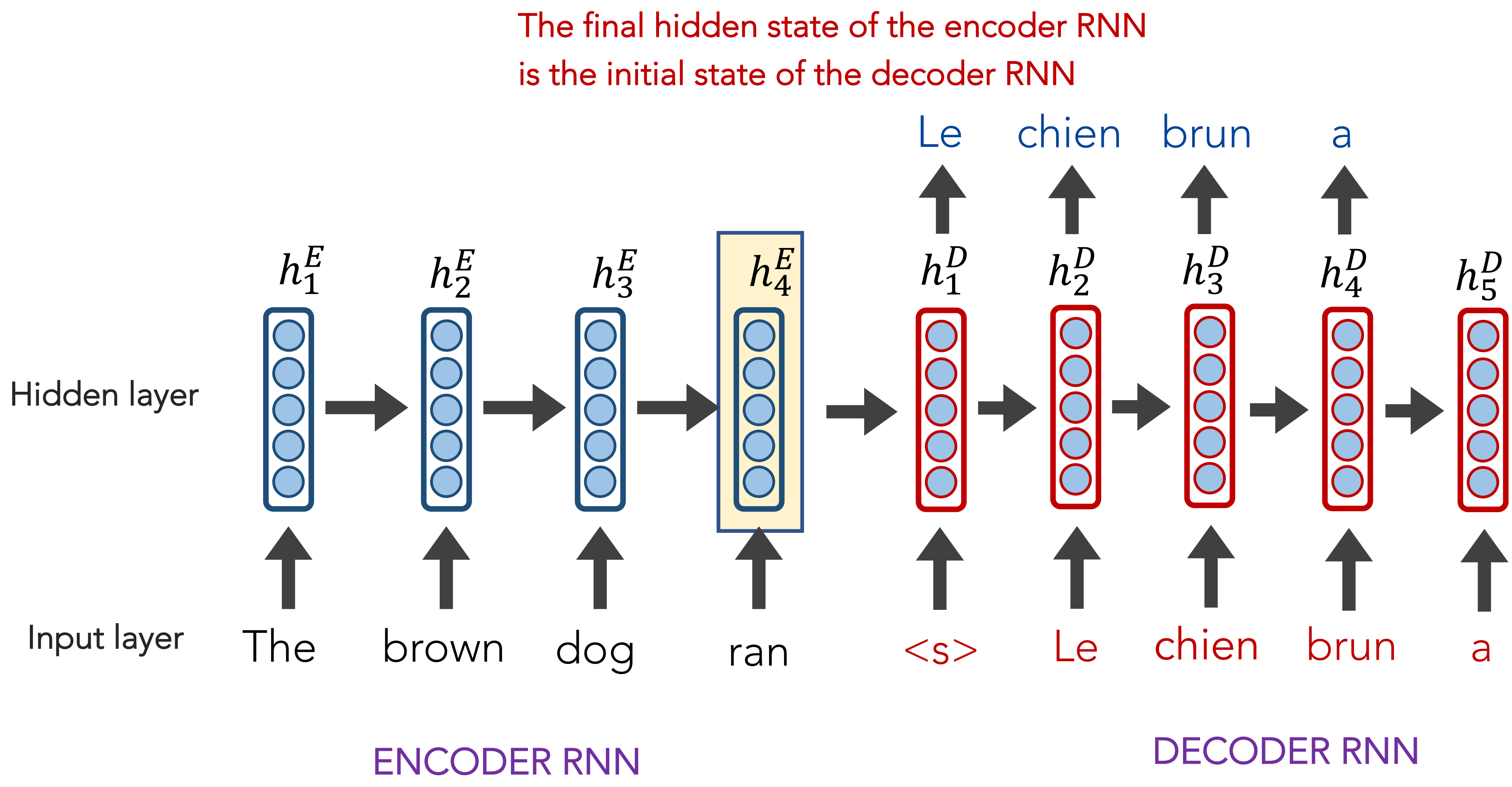
Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

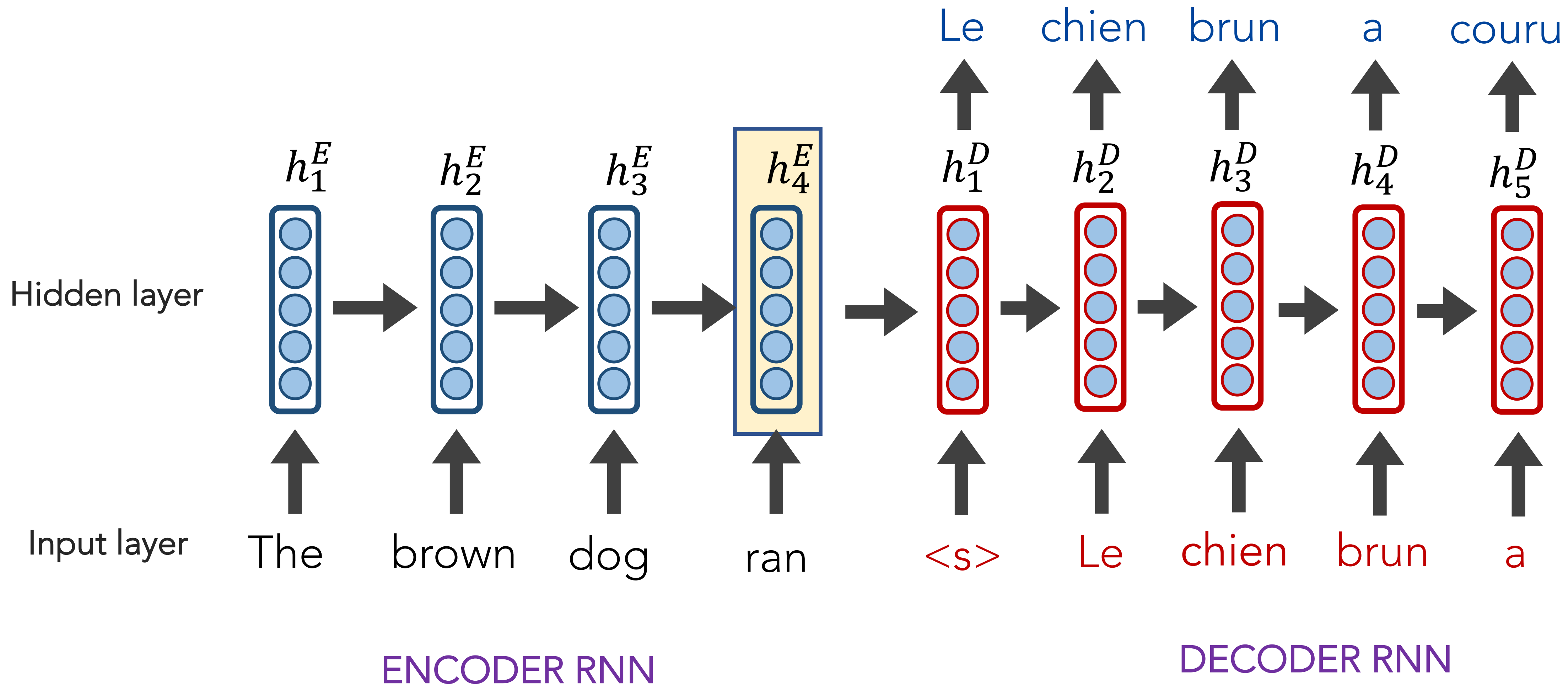


Sequence-to-Sequence (seq2seq)



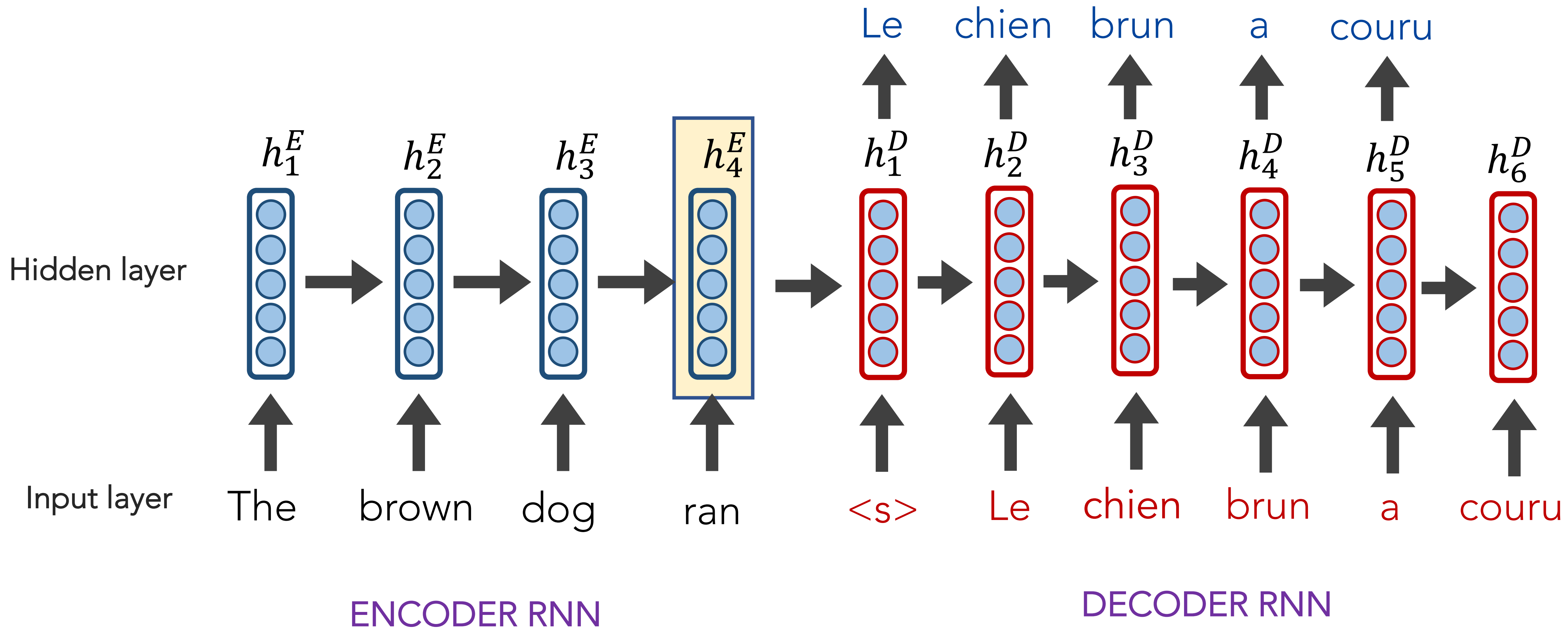
Sequence-to-Sequence (seq2seq)

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



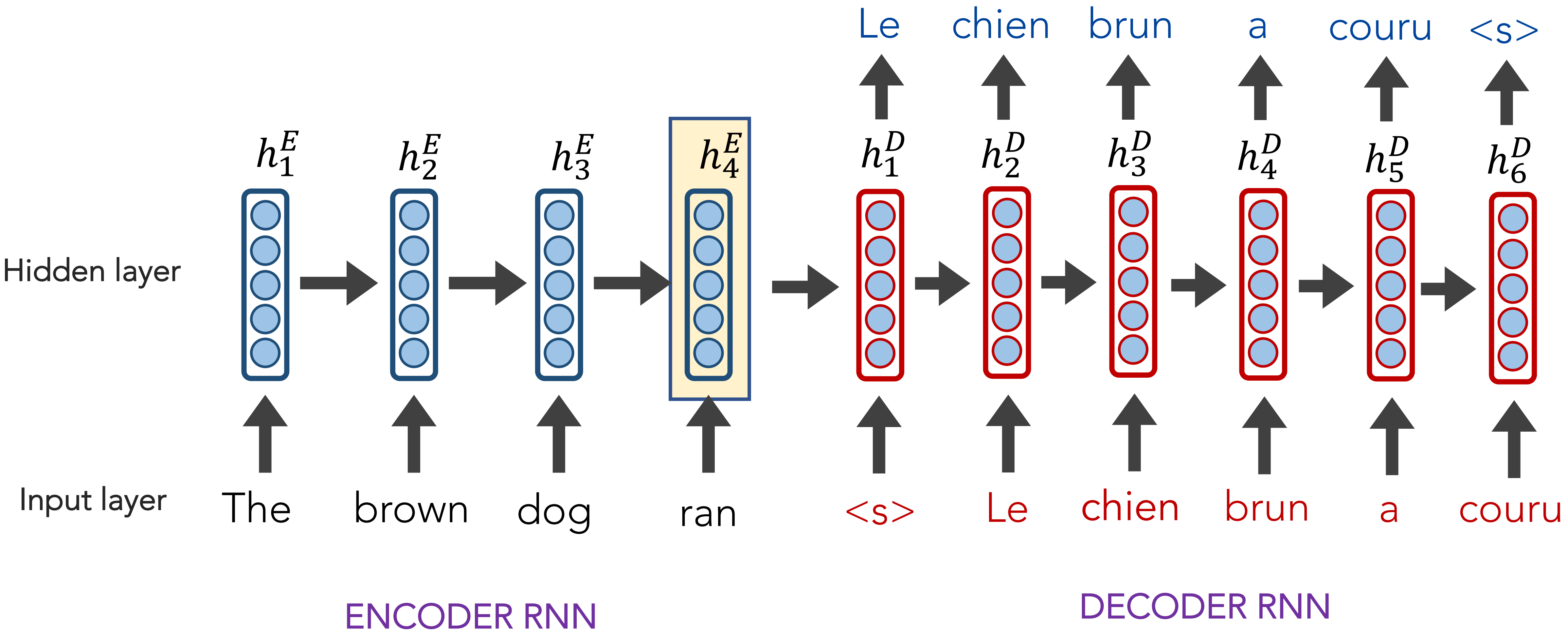
Sequence-to-Sequence (seq2seq)

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

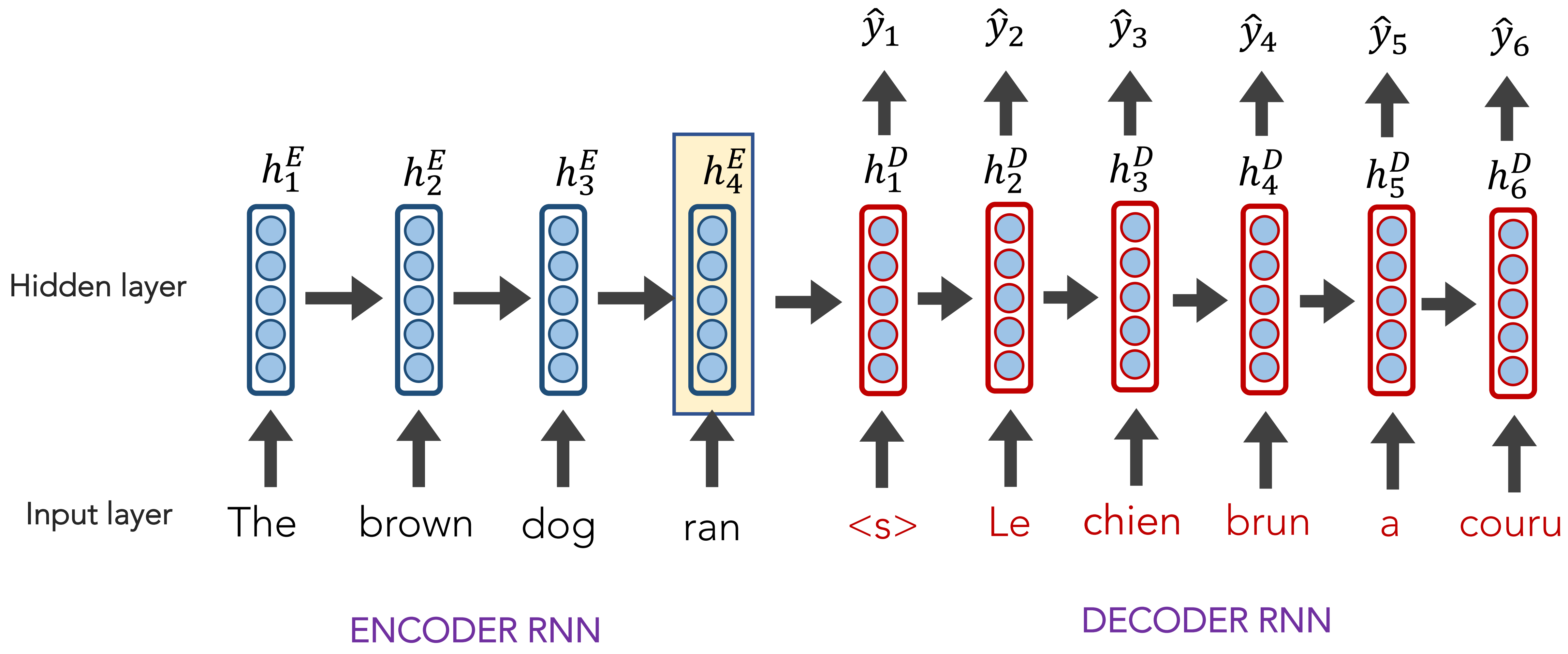


Sequence-to-Sequence (seq2seq)

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

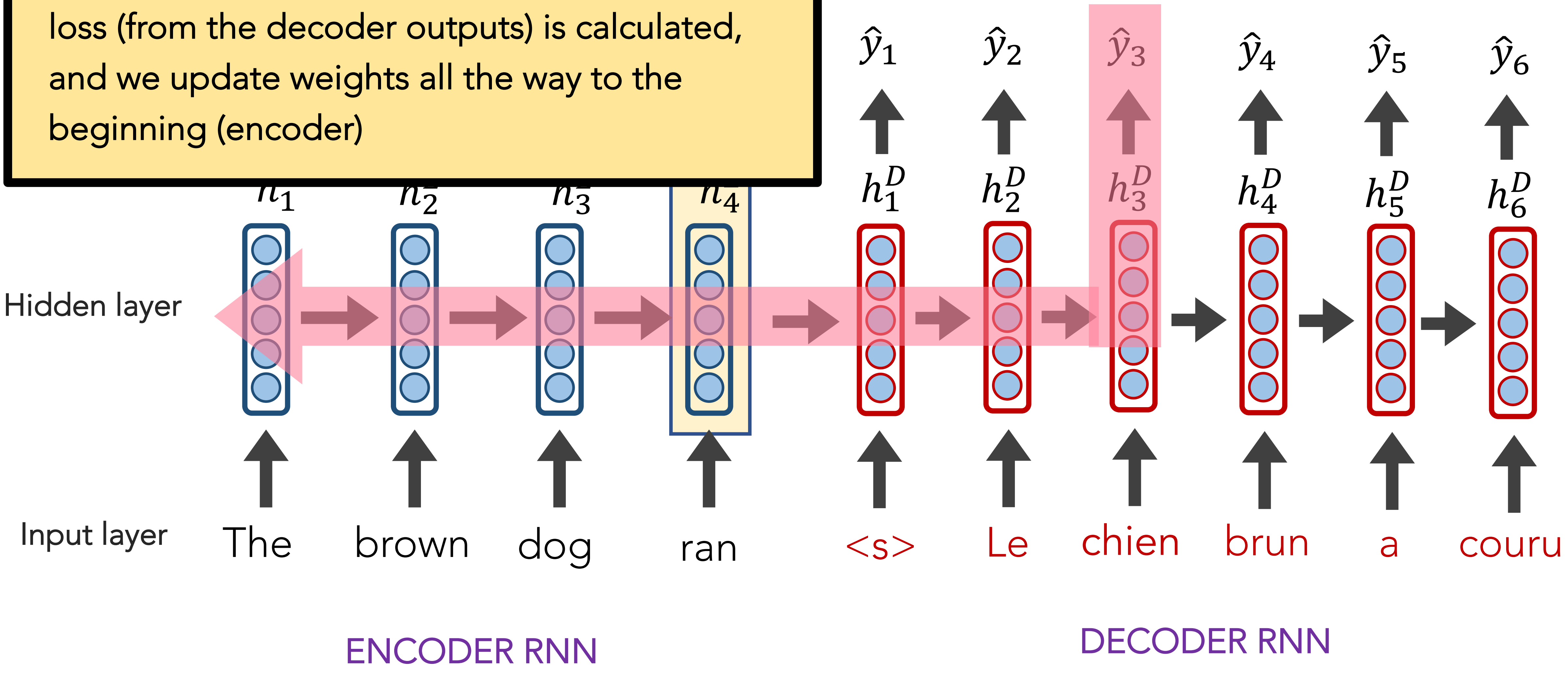


Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

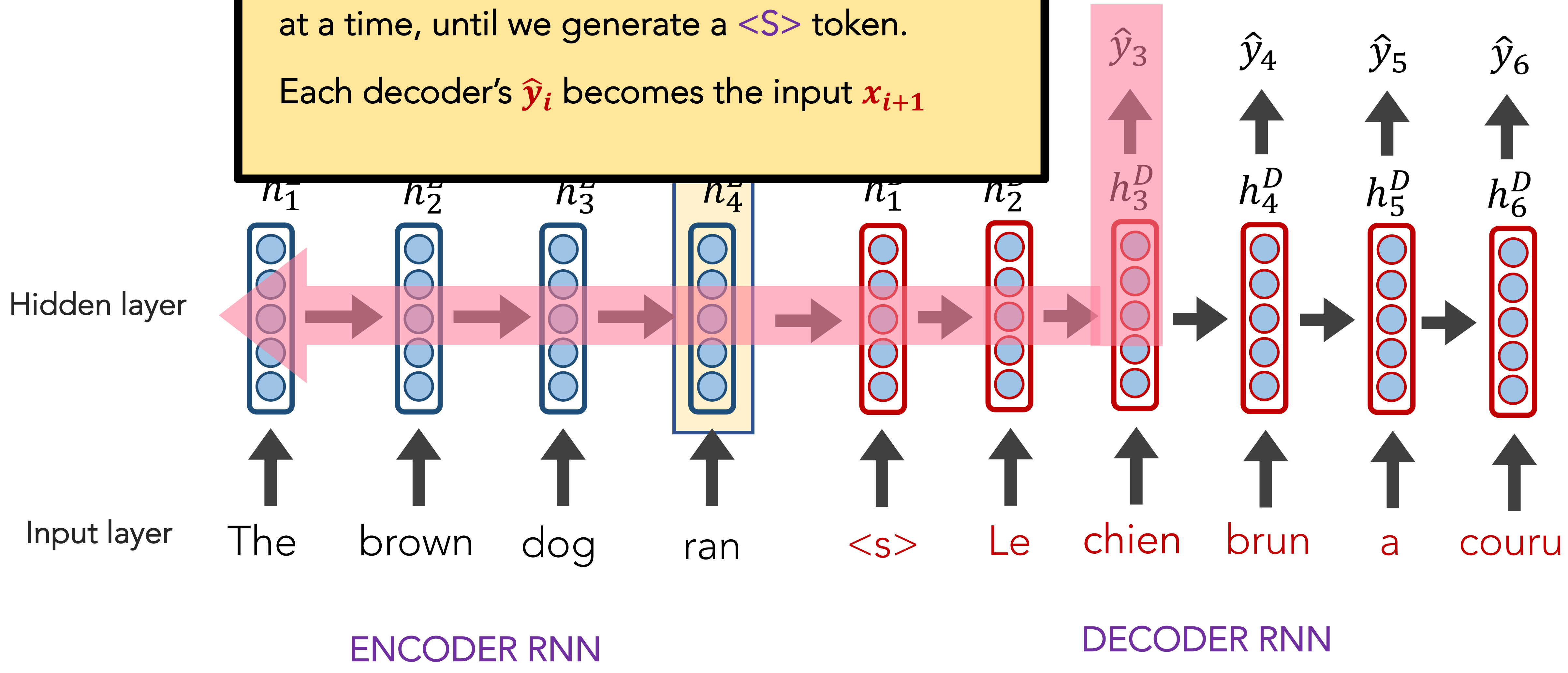
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

Testing generates decoder outputs one word at a time, until we generate a <S> token.

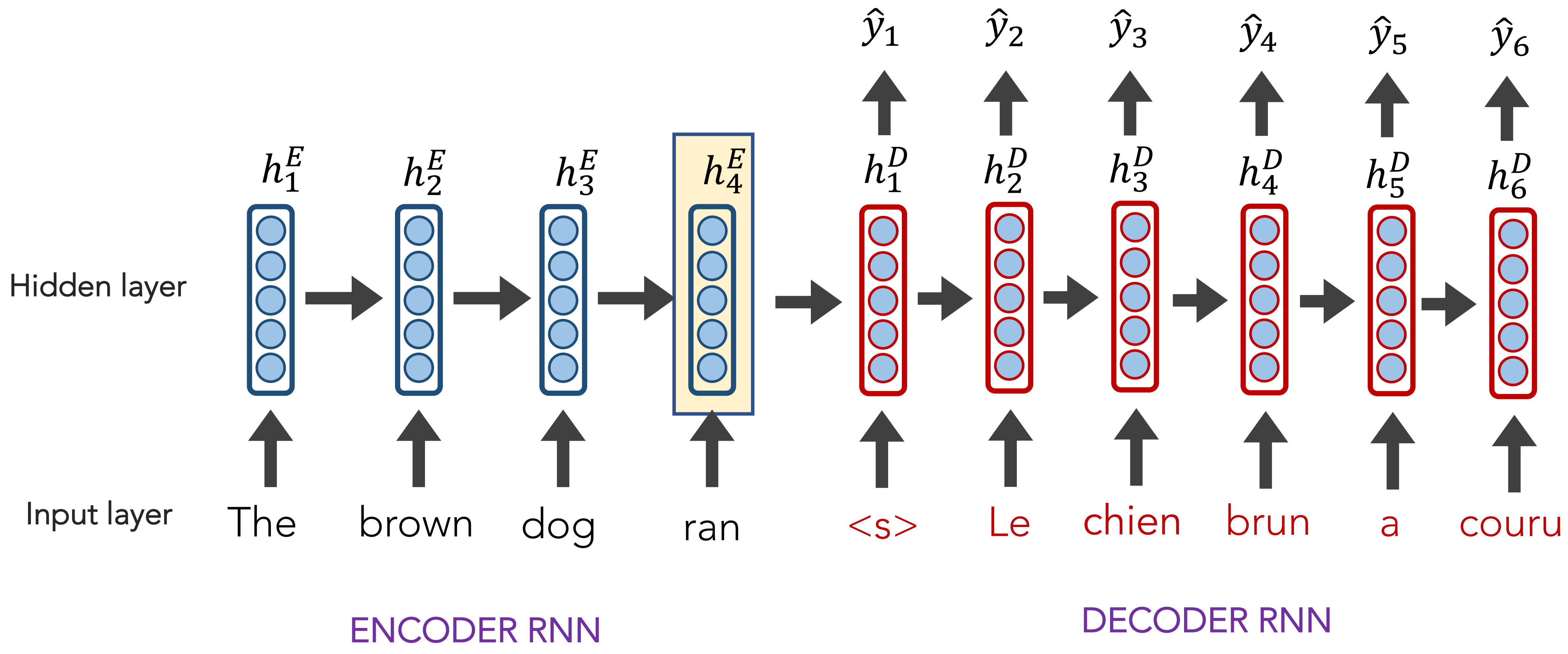
Each decoder's $\hat{\boldsymbol{y}}_i$ becomes the input $\boldsymbol{x}_{i+1}$



Sequence-to-Sequence (seq2seq)

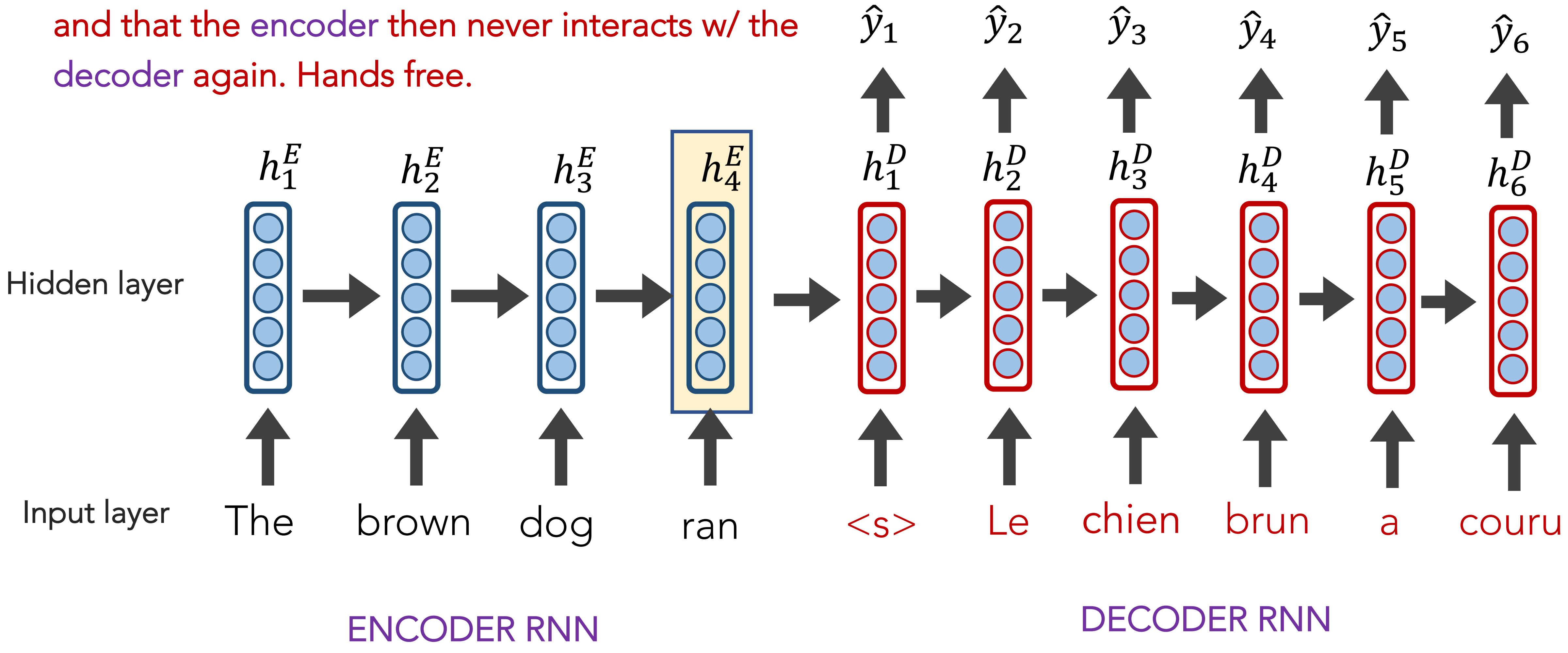
See any issues with this traditional **seq2seq** paradigm?

Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

It's crazy that 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. Hands free.



Sequence-to-Sequence (seq2seq)

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

Sequence-to-Sequence (seq2seq)

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

Intuition: when we (humans) translate a sentence, we don't just consume the original sentence, reflect on the meaning of the last word, then regurgitate in a new language; we continuously think back at the original sentence while focusing on different parts.

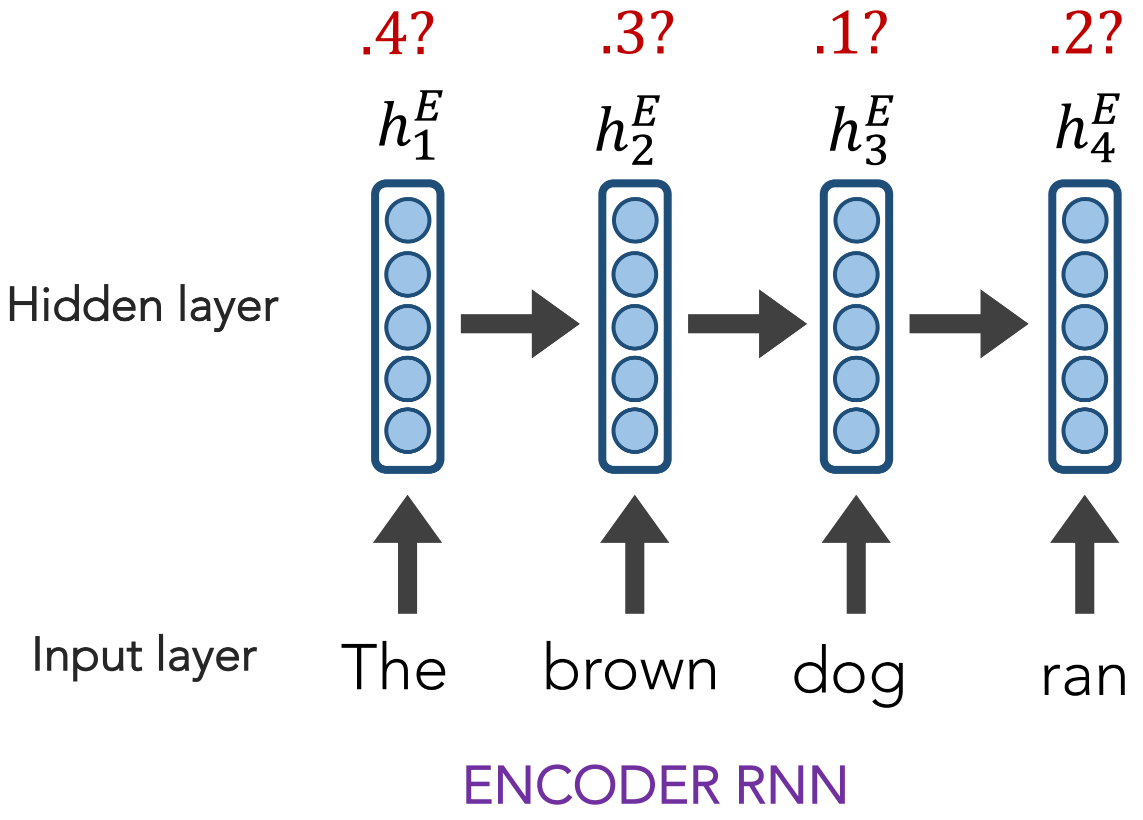
Attention

The concept of attention within cognitive neuroscience and psychology dates back to the 1800s. [William James, 1890].

Nadaray-Watson kernel regression proposed in 1964. It *locally weighted* its predictions.

seq2seq + Attention

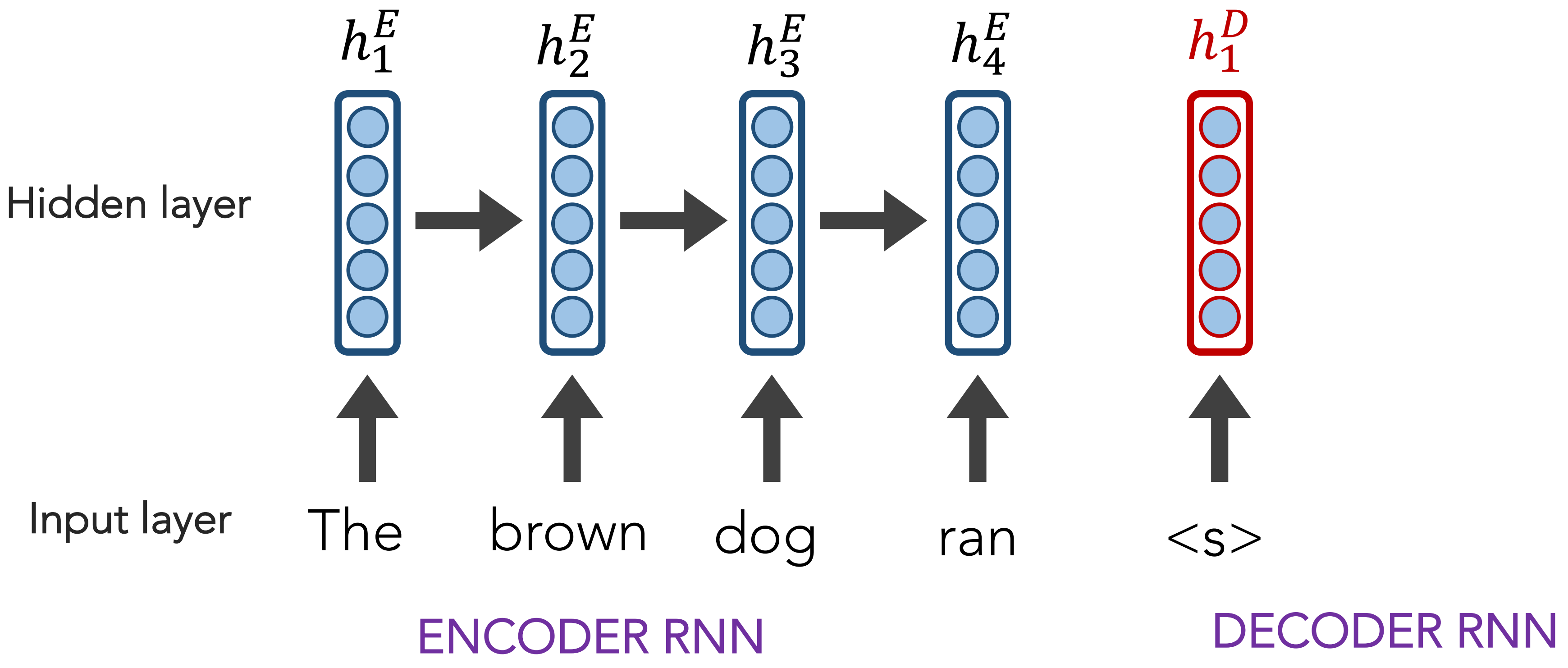
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

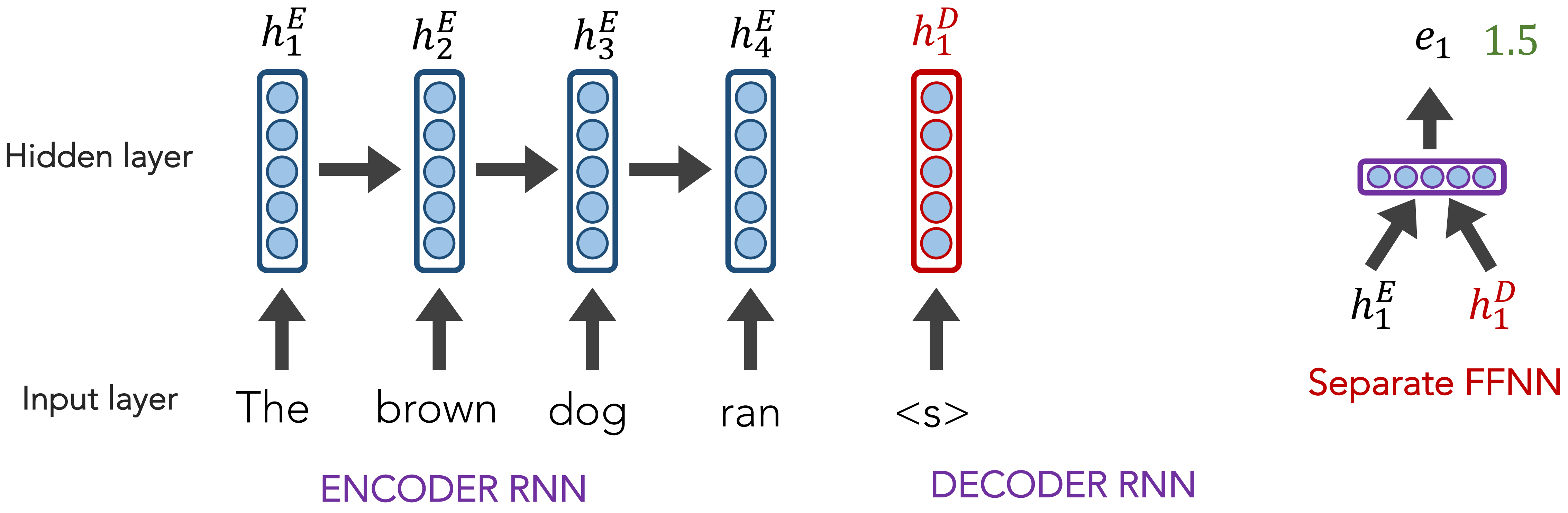
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

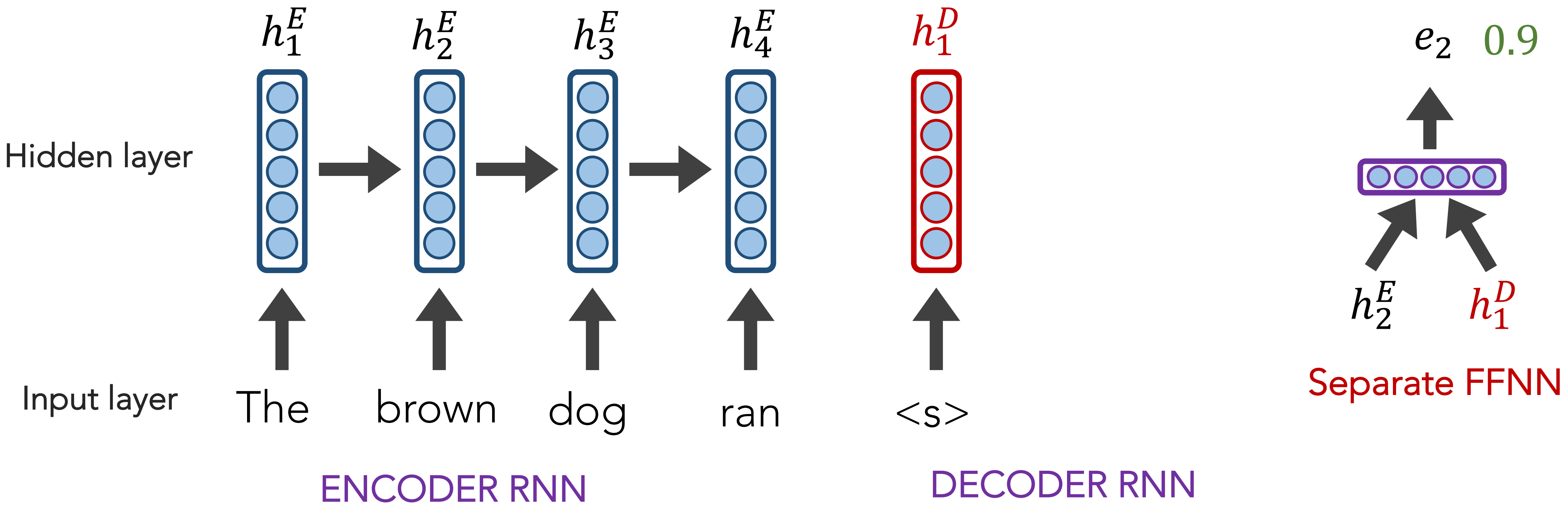
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

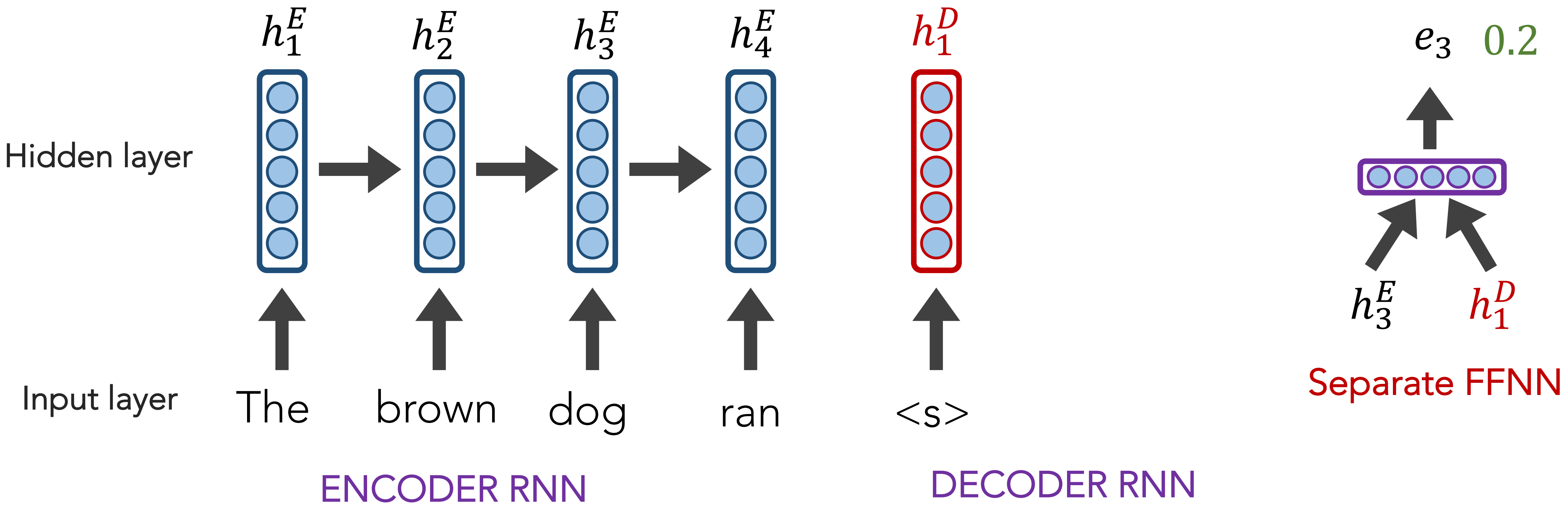
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

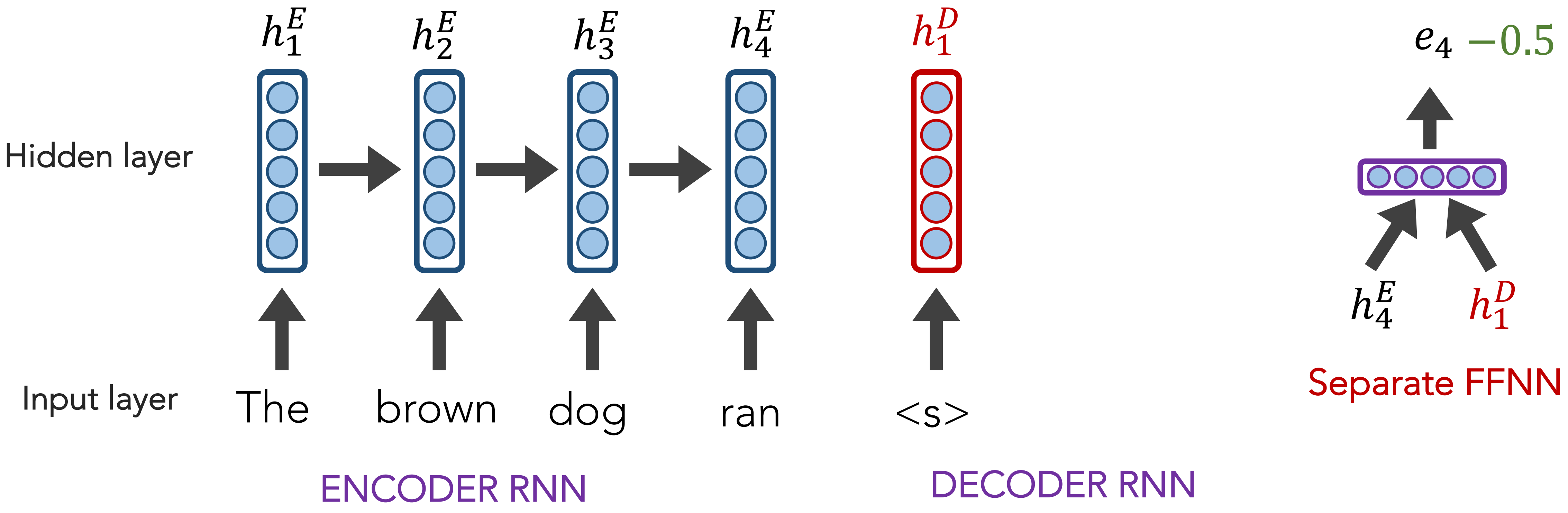
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

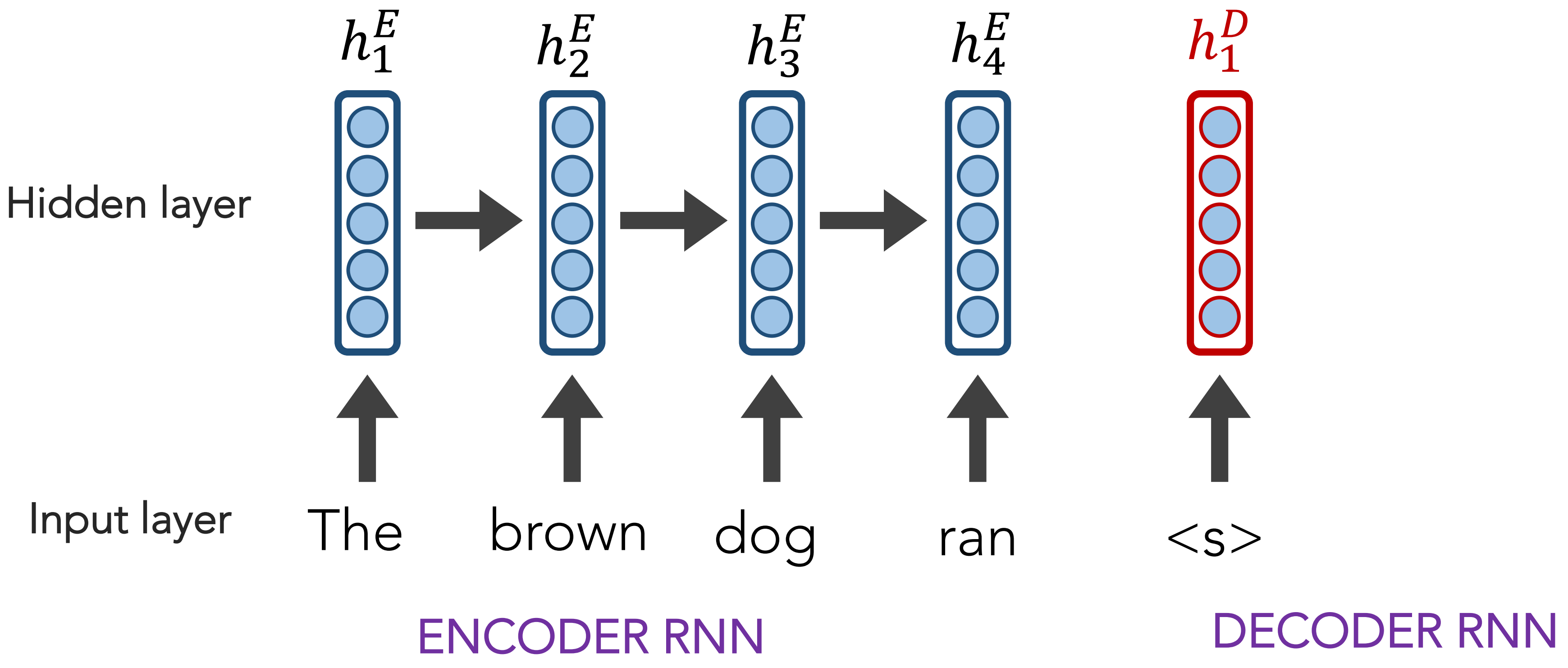
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



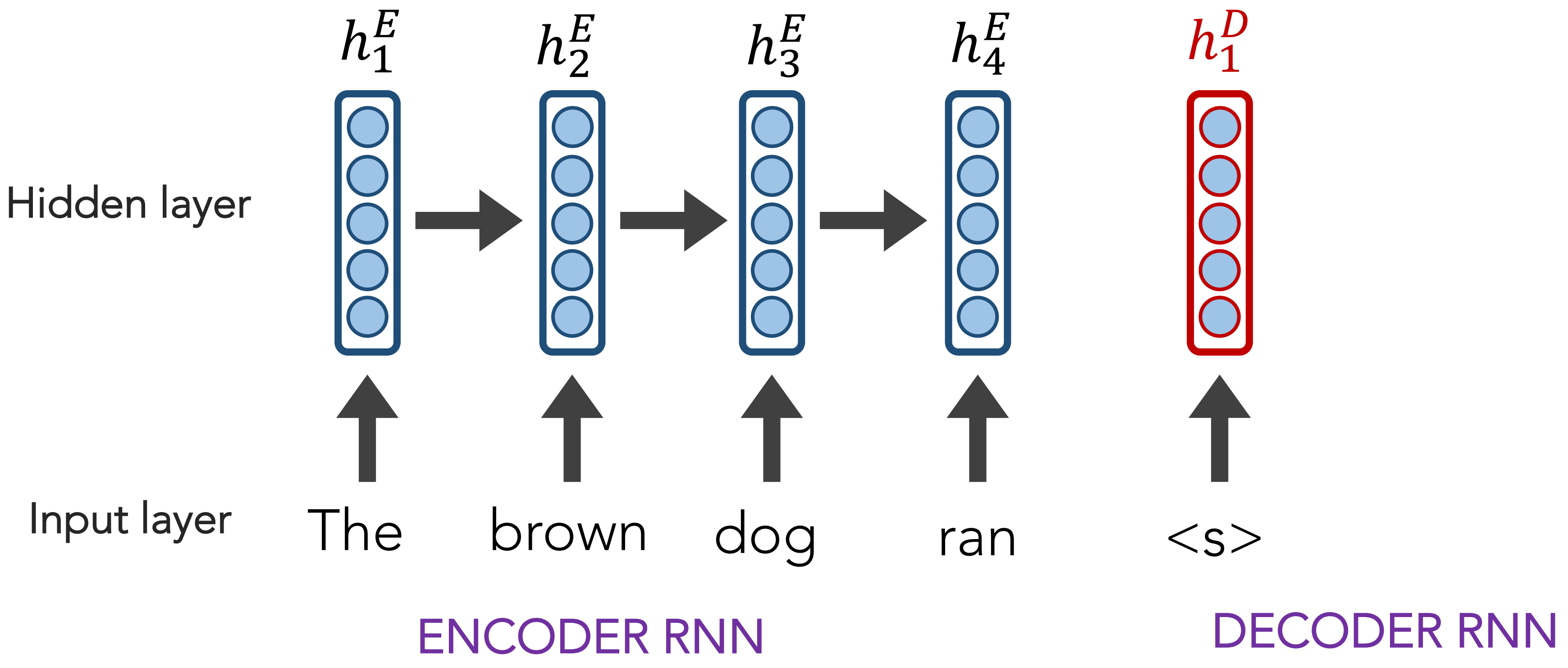
Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 -0.5

seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

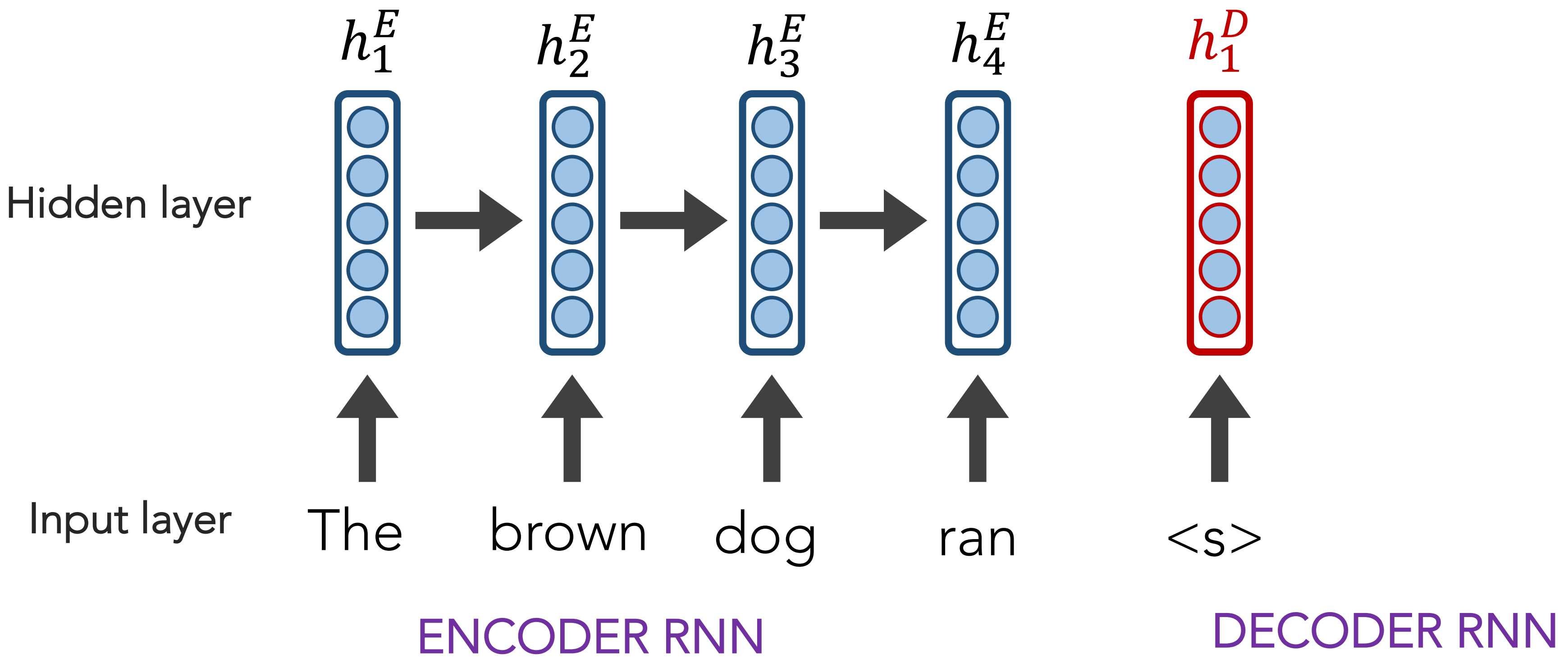
Attention (softmax'd)

$$a_i^1 = \frac{\exp(e_i)}{\sum_i^N \exp(e_1)}$$

seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



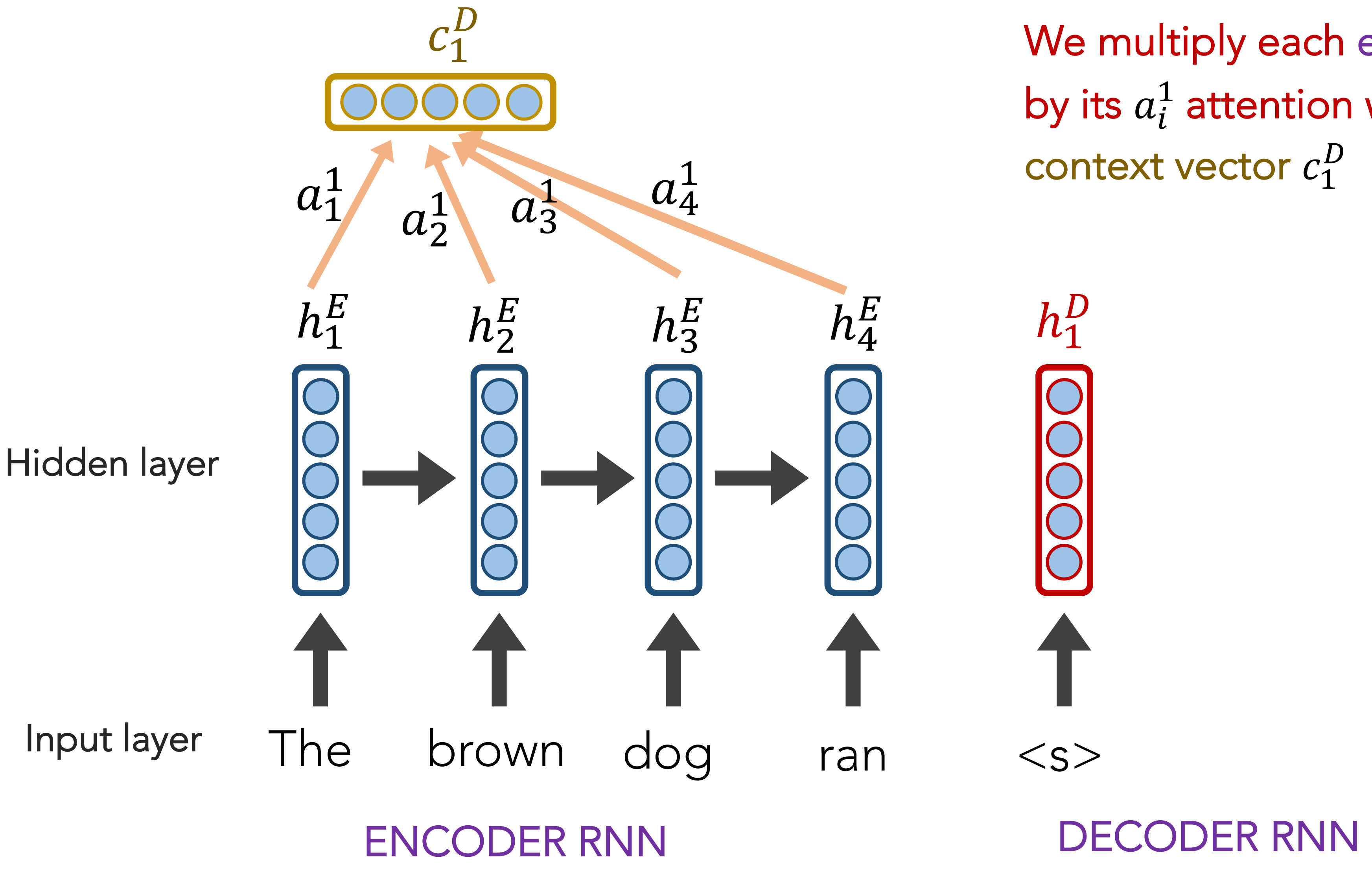
Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 -0.5

Attention (softmax'd)

$a_1^1 = 0.51$
$a_2^1 = 0.28$
$a_3^1 = 0.14$
$a_3^1 = 0.07$

seq2seq + Attention



We multiply each encoder's hidden layer by its a_i^1 attention weights to create a context vector c_1^D

Attention (softmax'd)

$a_1^1 = 0.51$

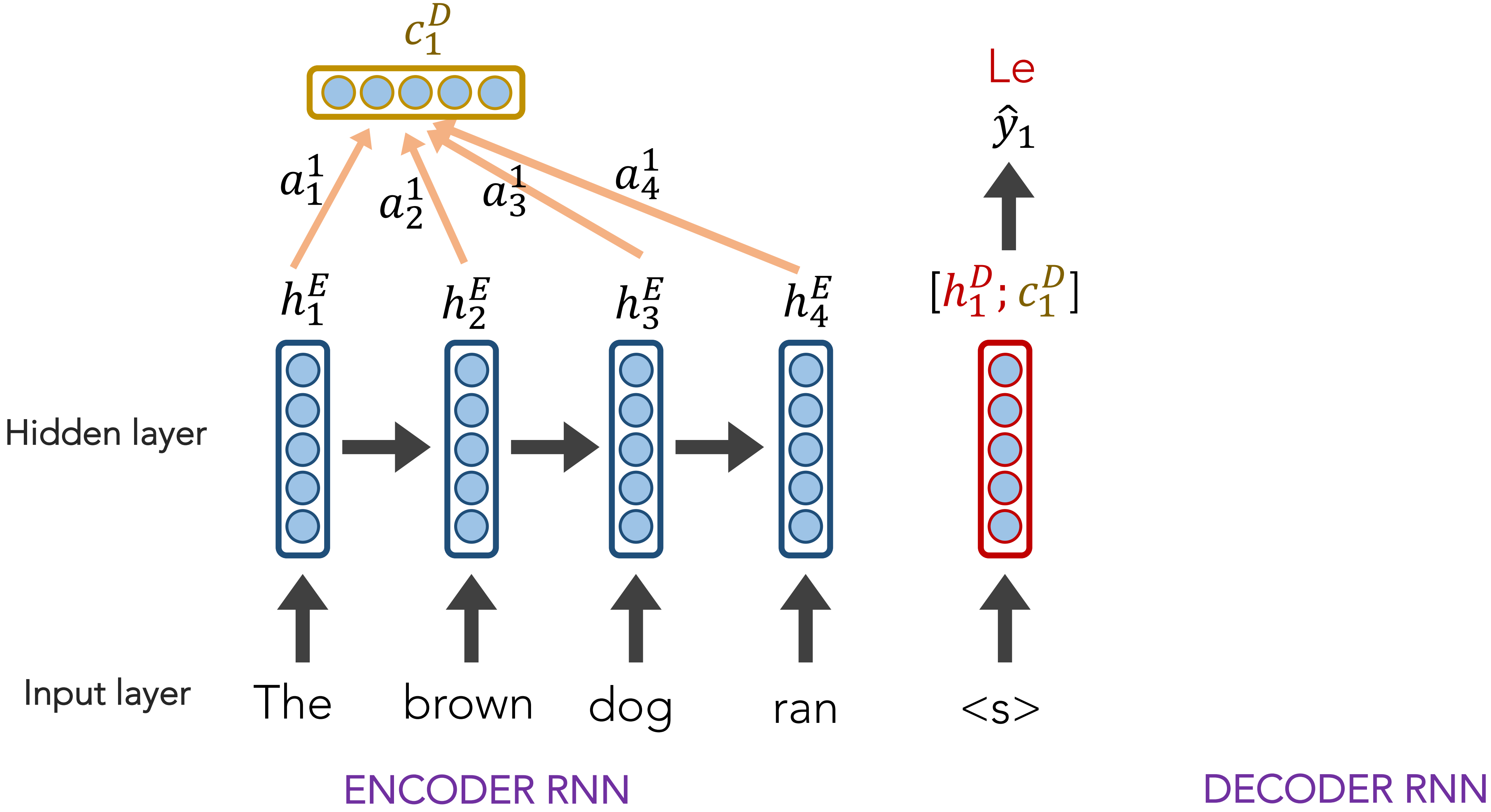
$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

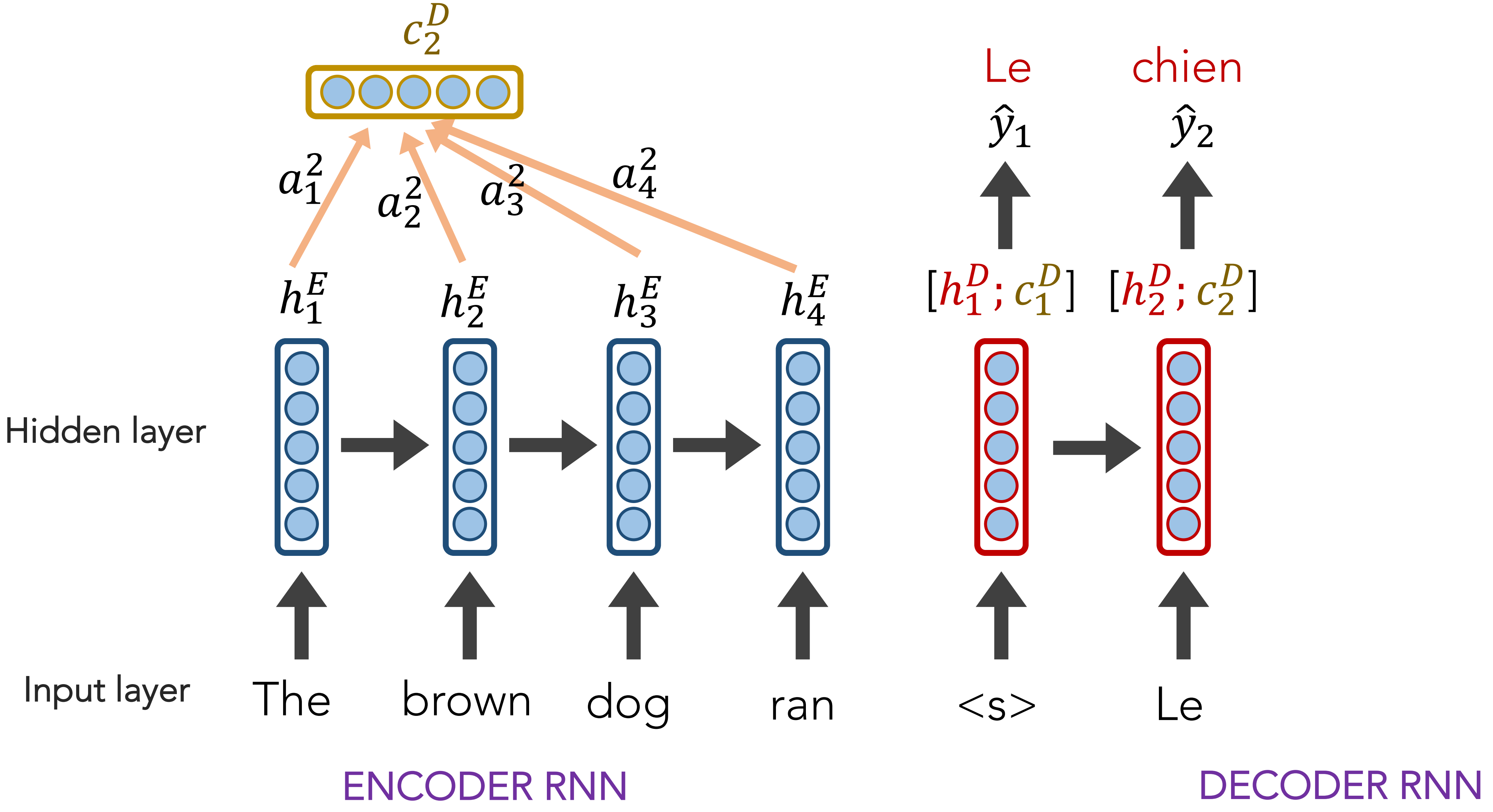
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



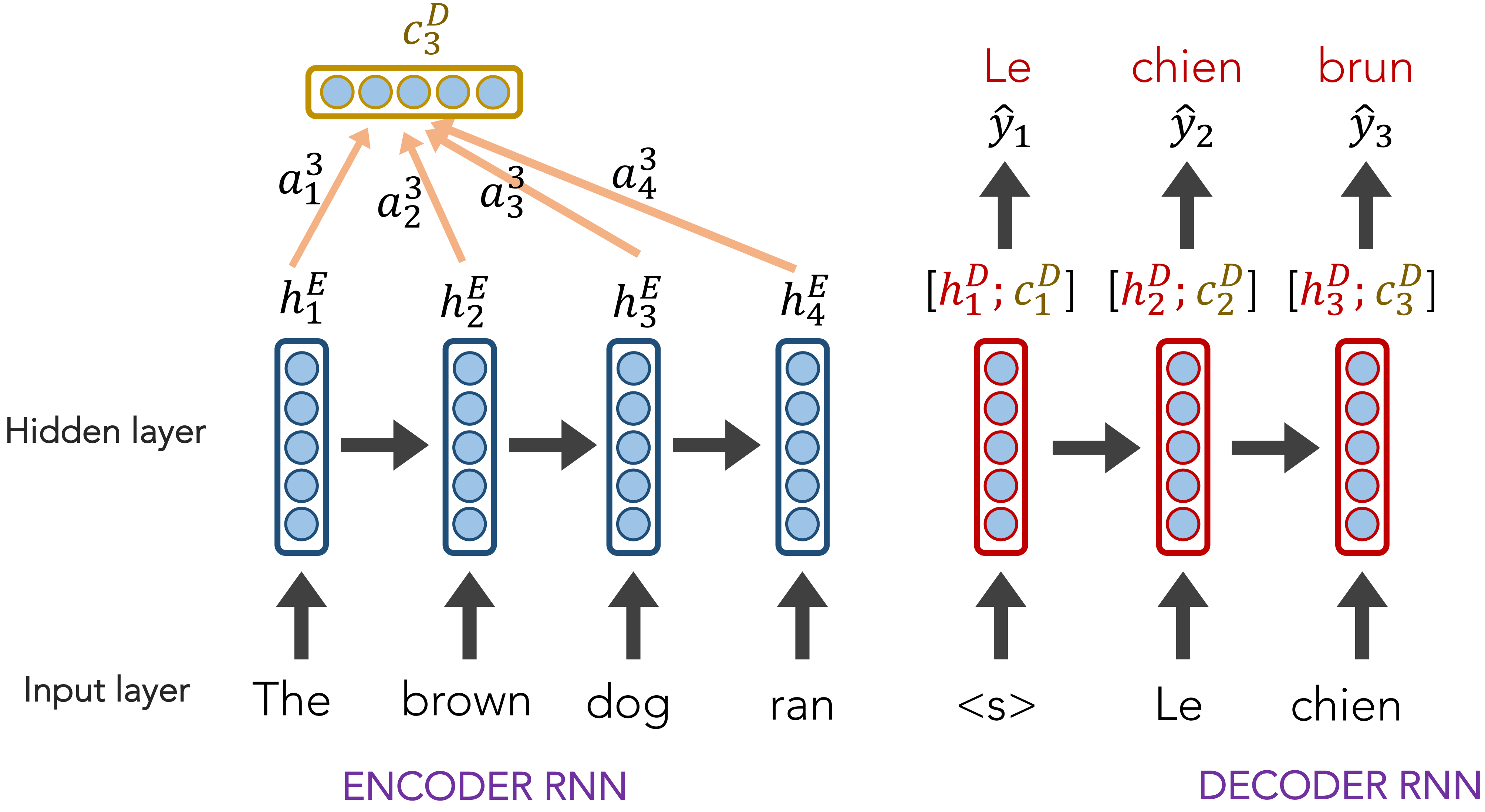
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



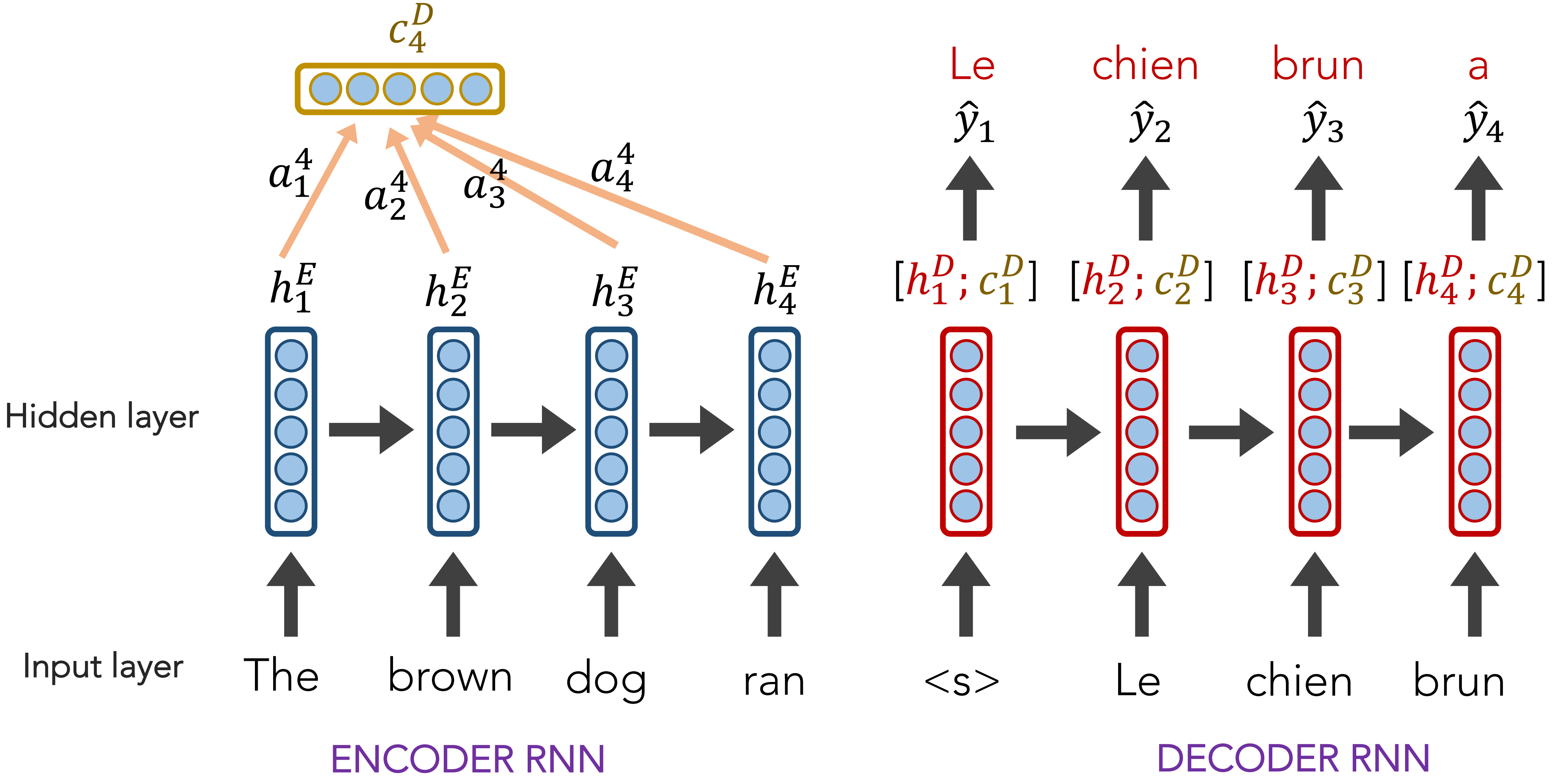
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



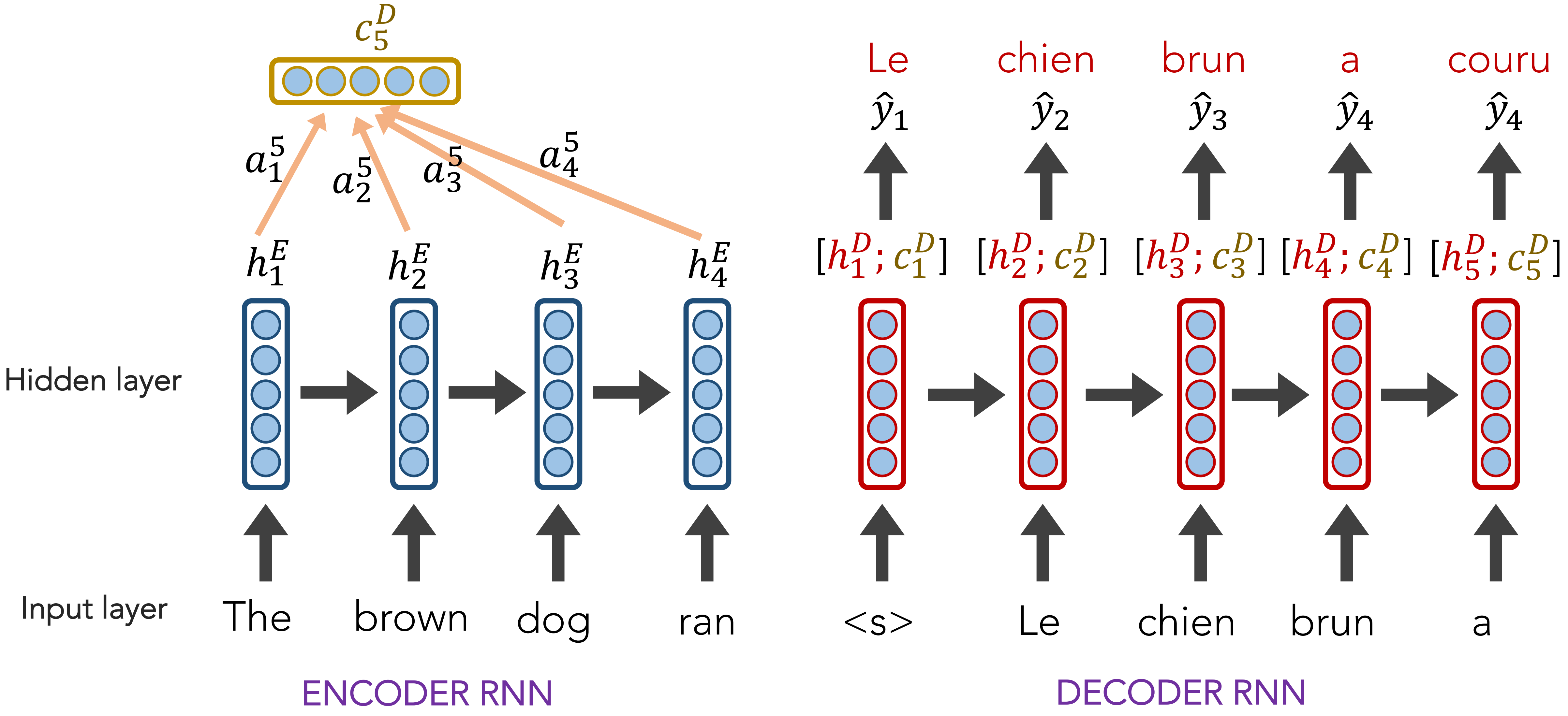
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



For convenience, here's the Attention calculation summarized on 1 slide

Attention output

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$$

"source context for decoder step t"

(weighted sum)

Attention weights

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \mathrm{k} = 1..\mathrm{m}$$

"attention weight for source token k at decoder step t"

(softmax)

Attention scores

$$\text{score}(h_t, s_k), \mathrm{k} = 1..\mathrm{m}$$

"How relevant is source token k for target step t?"

Attention input

$s_1, s_2, \ldots, s_m$ — all encoder states

h_t — one decoder state

Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

For convenience, here's the Attention calculation summarized on 1 slide

The **Attention mechanism** that produces scores doesn't have to be a FFNN like I illustrated. It can be any function you wish.

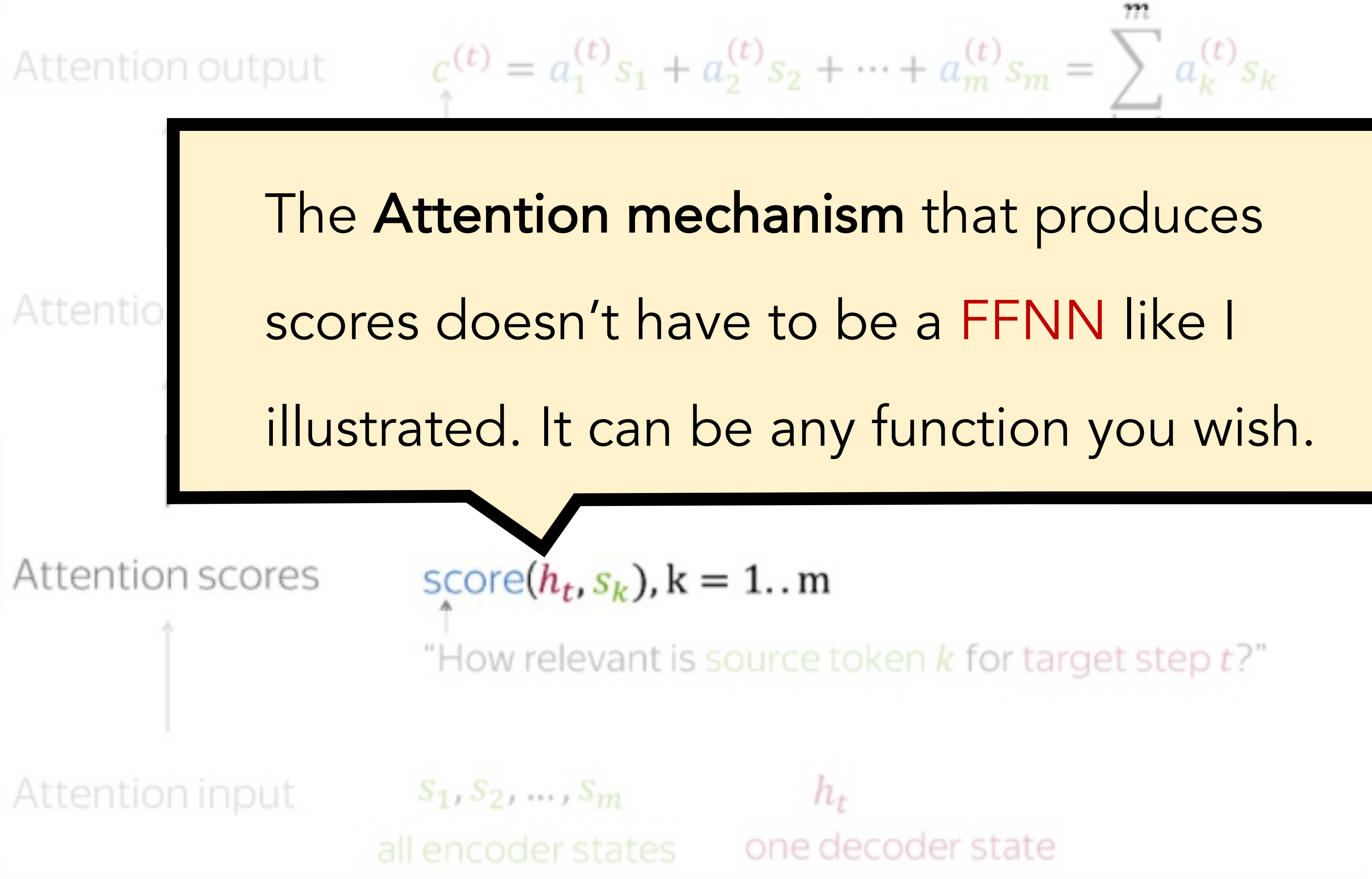
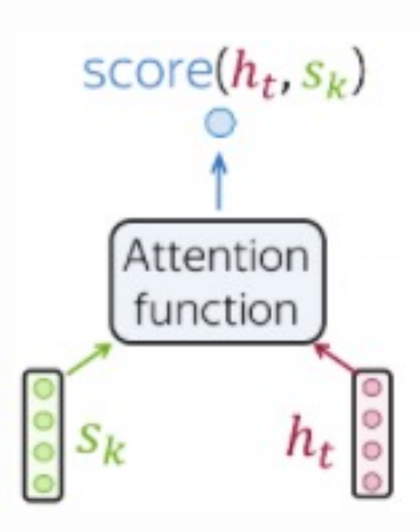


Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html



Popular Attention Scoring functions:

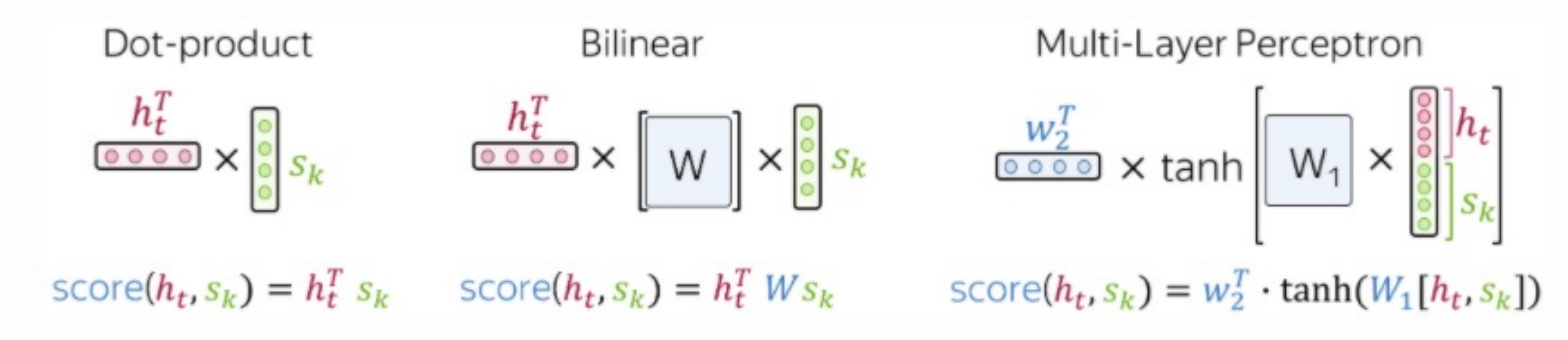


Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

seq2seq + Attention

Attention:

- greatly improves seq2seq results
- allows us to visualize the contribution each encoding word gave for each decoder's word

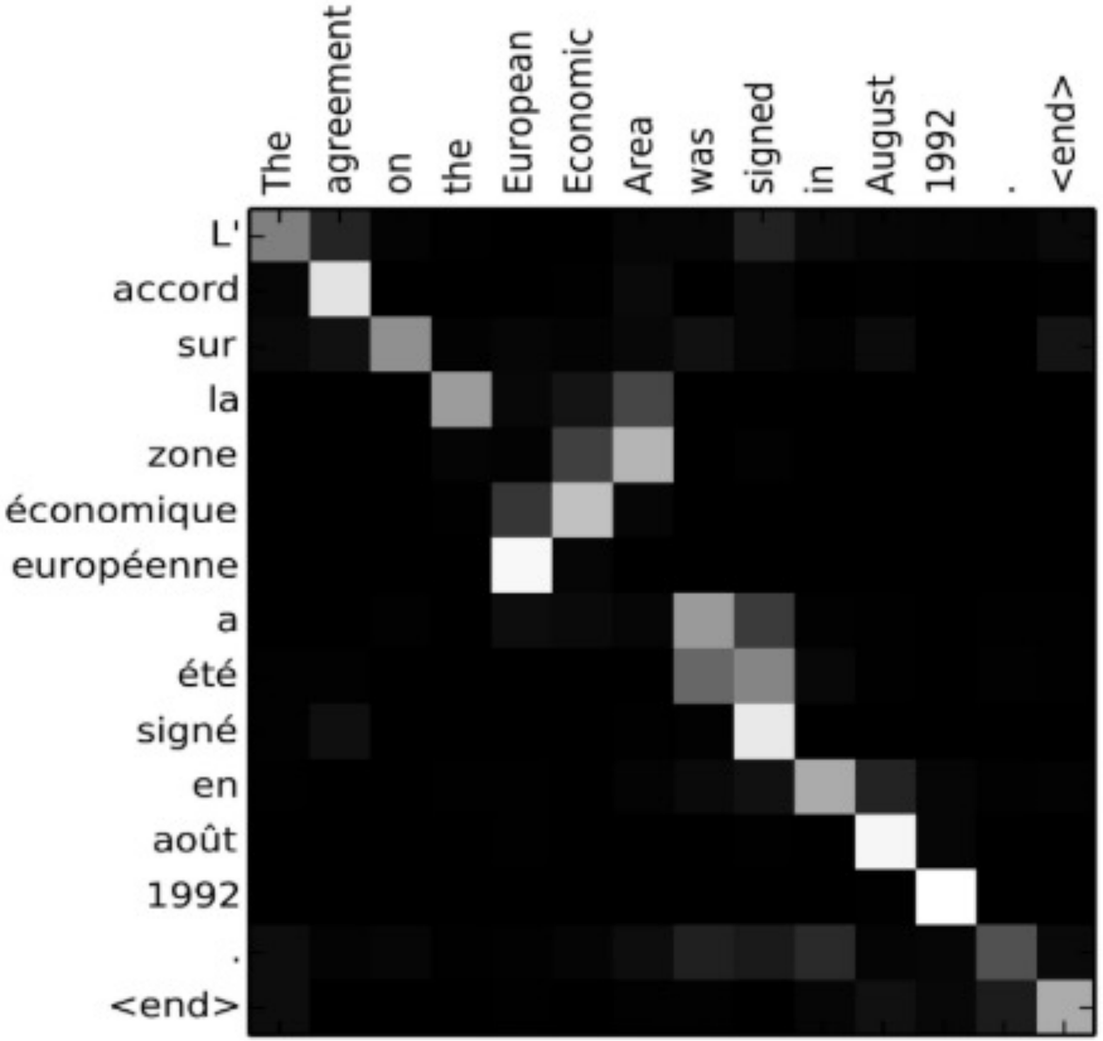


Image source: Fig 3 in [Bahdanau et al., 2015]

Takeaway:

Having a separate encoder and decoder allows for n → m length predictions.

Attention is powerful; allows us to conditionally weight our focus

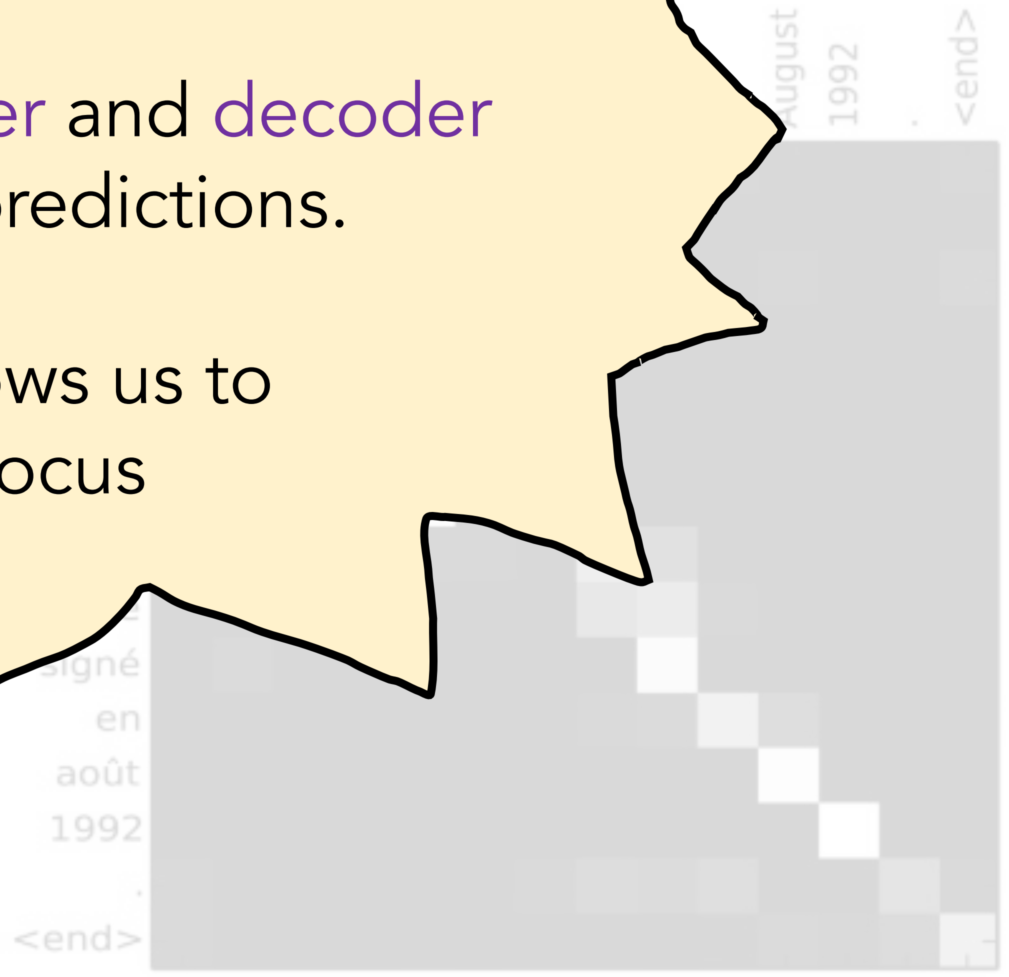
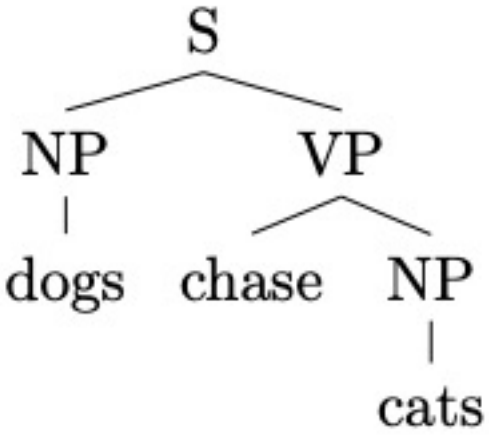


Image source: Fig 3 in Bahdanau et al., 2015

Constituency Parsing

Input: dogs chase cats

Output:



or a flattened representation

$$(\mathrm{S}\ (\mathrm{NP}\ \text{dogs}\ )_{\mathrm{NP}}\ (\mathrm{VP}\ \text{chase}\ (\mathrm{NP}\ \text{cats}\ )_{\mathrm{NP}}\ )_{\mathrm{VP}}\ )_{\mathrm{S}}$$

Constituency Parsing

Input: I shot an elephant in my pajamas

Output:

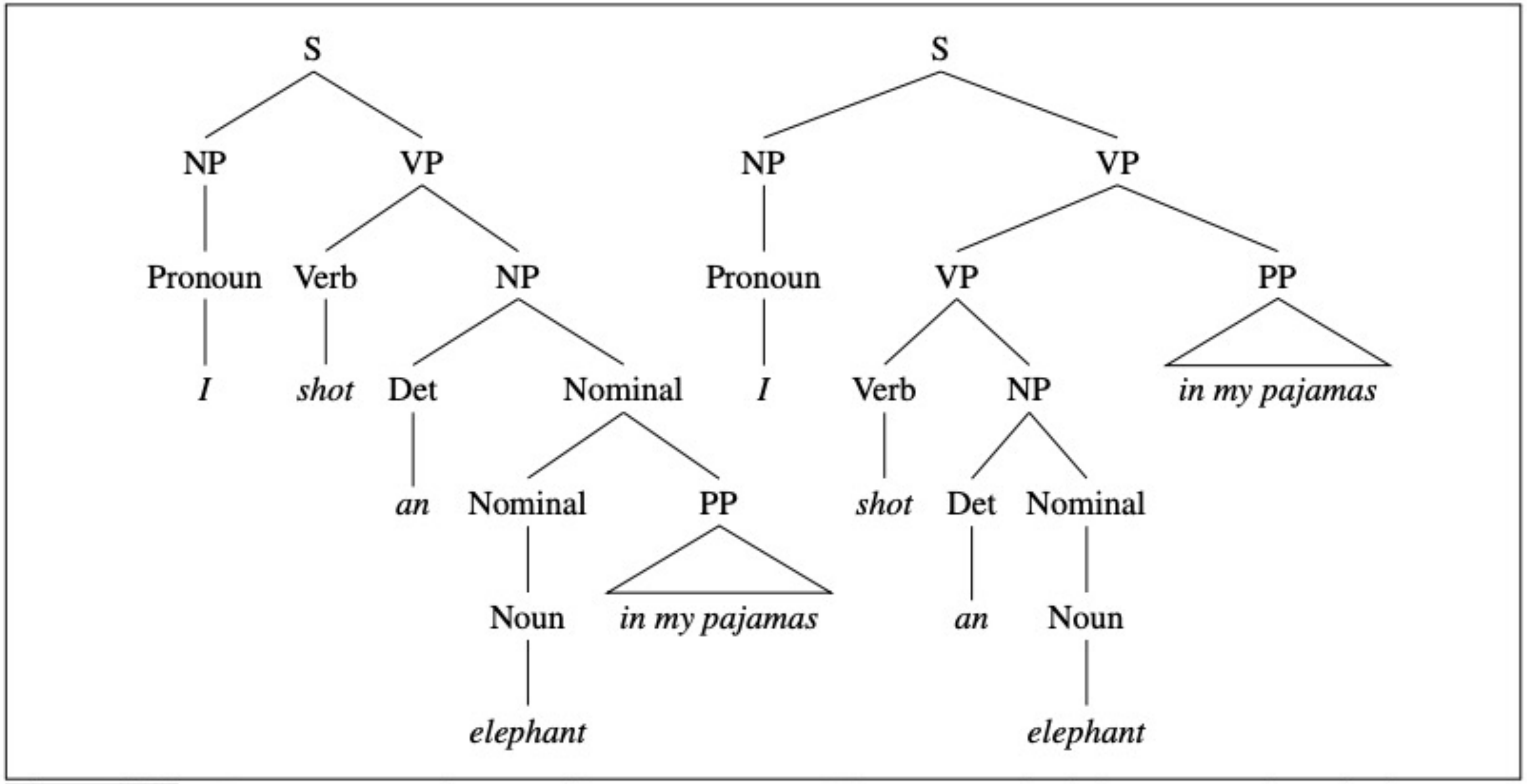


Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

https://web.stanford.edu/~jurafsky/slp3/13.pdf

Results

Model	English			Chinese		
	LR	LP	F1	LR	LP	F1
Shen et al. (2018)	92.0	91.7	91.8	86.6	86.4	86.5
Fried and Klein (2018)	-	-	92.2	-	-	87.0
Teng and Zhang (2018)	92.2	92.5	92.4	86.6	88.0	87.3
Vaswani et al. (2017)	-	-	92.7	-	-	-
Dyer et al. (2016)	-	-	93.3	-	-	84.6
Kuncoro et al. (2017)	-	-	93.6	-	-	-
Charniak et al. (2016)	-	-	93.8	-	-	-
Liu and Zhang (2017b)	91.3	92.1	91.7	85.9	85.2	85.5
Liu and Zhang (2017a)	-	-	94.2	-	-	86.1
Suzuki et al. (2018)	-	-	94.32	-	-	-
Takase et al. (2018)	-	-	94.47	-	-	-
Fried et al. (2017)	-	-	94.66	-	-	-
Kitaev and Klein (2018)	94.85	95.40	95.13	-	-	-
Kitaev et al. (2018)	95.51	96.03	95.77	91.55	91.96	91.75
Zhou and Zhao (2019) (BERT)	95.70	95.98	95.84	**92.03**	92.33	92.18
Zhou and Zhao (2019) (XLNet)	96.21	96.46	96.33	-	-	-
Our work	**96.24**	**96.53**	**96.38**	91.85	**93.45**	**92.64**

Table 3: Constituency Parsing on PTB & CTB test sets.

https://aclanthology.org/2020.findings-emnlp.65.pdf

Image Captioning

Input: image

Output: generated text

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Xu et al. CVPR (2016)

Image Captioning

Input: image

Output: generated text

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Xu et al. CVPR (2016)

Image Captioning

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Xu et al. CVPR (2016)

Image Captioning

A woman is sitting at a table with a large pizza.

A person is standing on a beach with a surfboard.

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Xu et al. CVPR (2016)

SUMMARY

- LSTMs yielded state-of-the-art results on most NLP tasks (2014-2018)
- seq2seq+Attention was an even more revolutionary idea (Google Translate used it)
- Attention allows us to place appropriate weight to the encoder's hidden states
- But, **LSTMs** require us to iteratively scan each word and wait until we're at the end before we can do anything

BACKUP