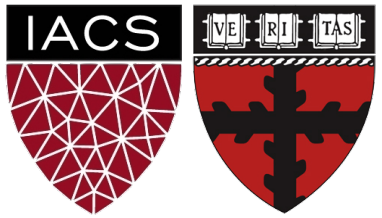
Lecture 24: Attention

NLP Lectures: Part 3 of 4

Harvard IACS

CS109B

Pavlos Protopapas, Mark Glickman, and Chris Tanner



Outline

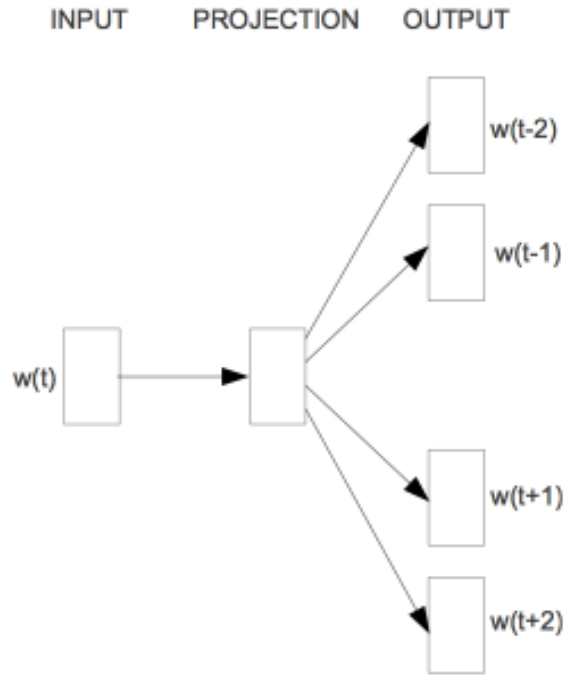
- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Previously, we learned about **word embeddings**

millions of books



word2vec

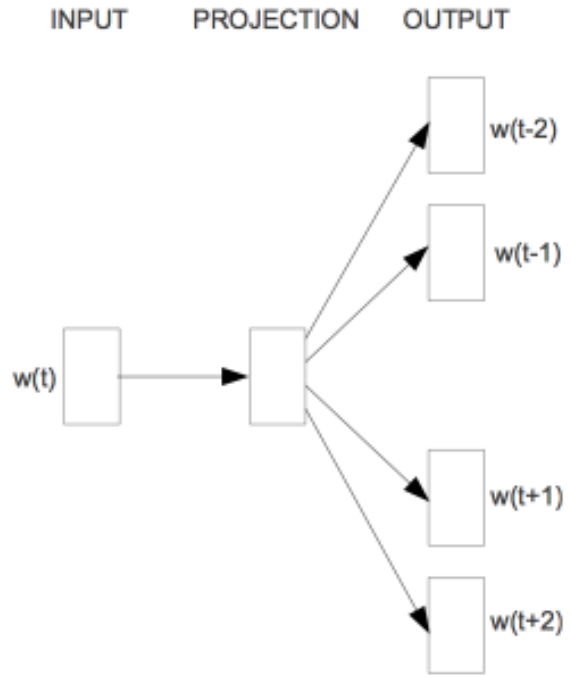
word embeddings (type-based)

approaches:

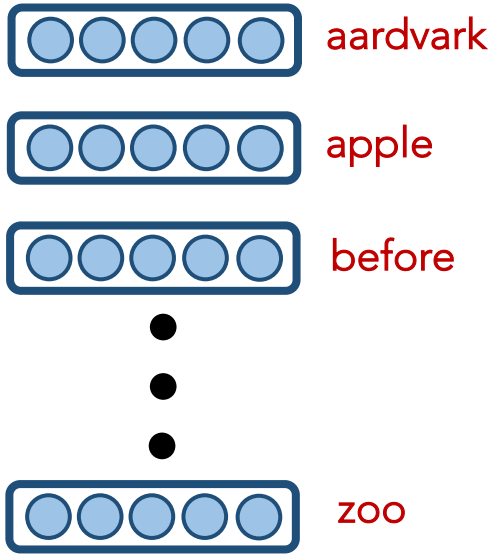
- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

Previously, we learned about **word embeddings**

millions of books



word2vec



word embeddings

word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

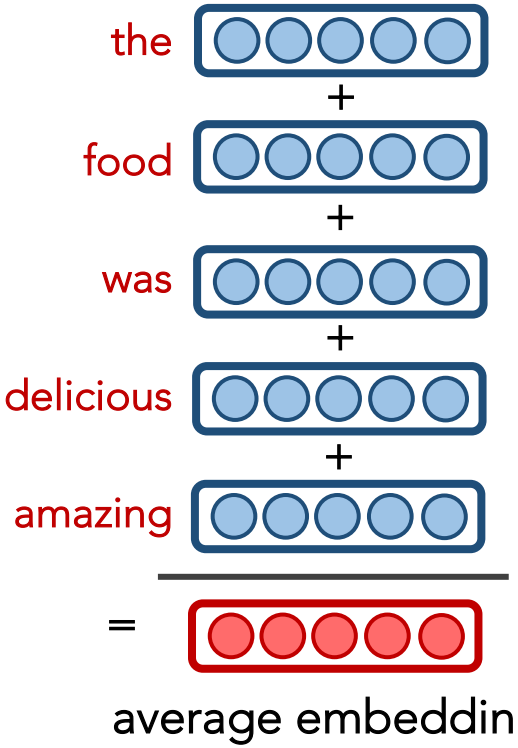
"The food was delicious. Amazing!" → **4.8/5** yelp

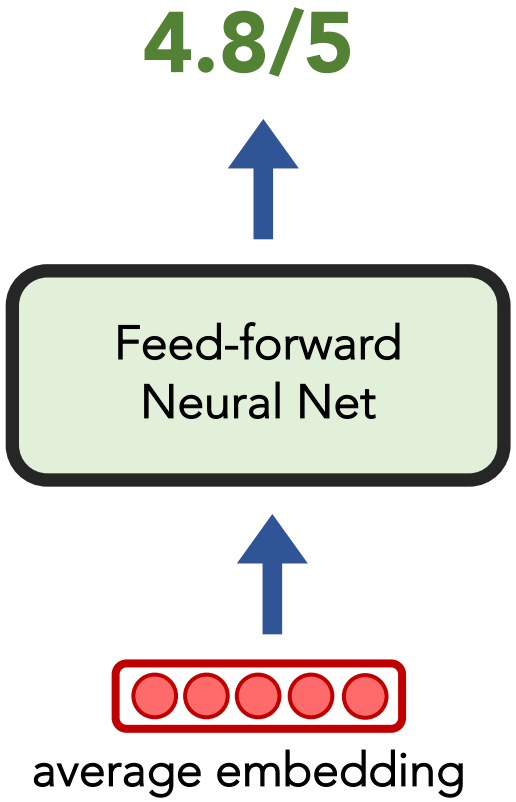
word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

"The food was delicious. Amazing!" → **4.8/5** yelp



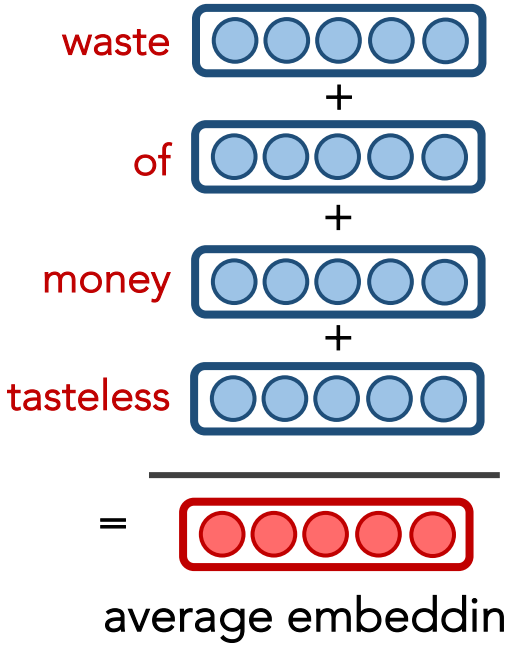


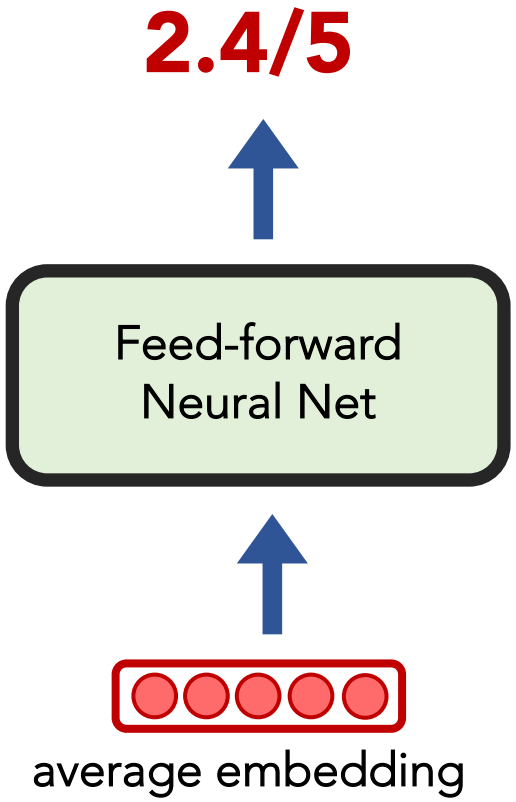
word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

"Waste of money. Tasteless!" → **2.4/5** yelp



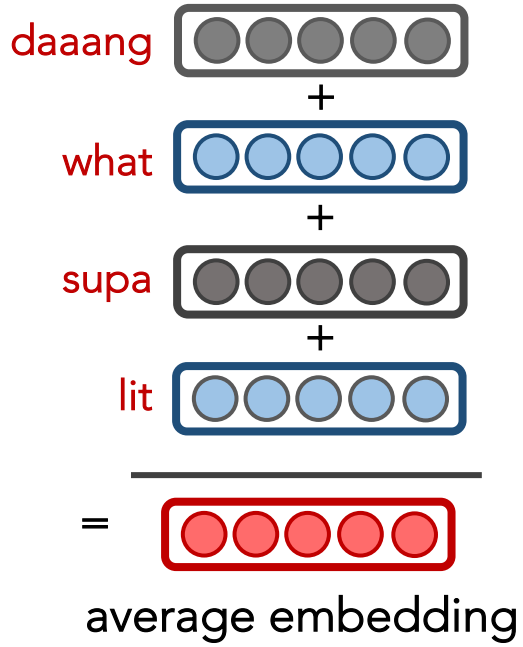


word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

"Daaang. What?! Supa Lit" → **4.9/5** yelp

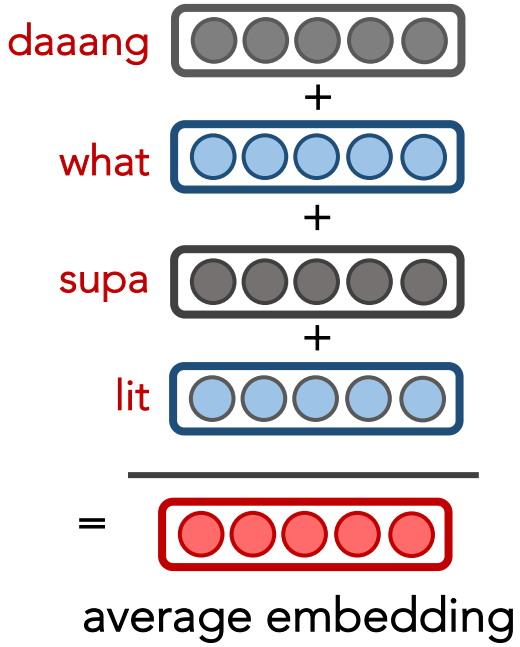


Strengths and weaknesses of word embeddings (type-based)?

word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)



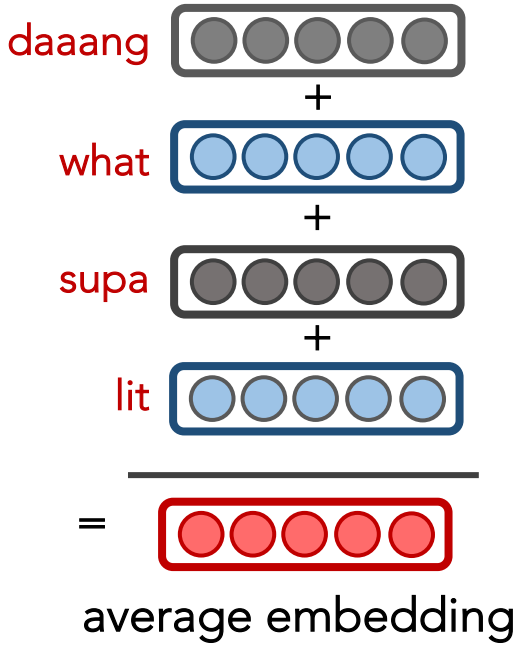
Strengths:

- Leverages tons of existing data
- Don't need to depend on our data to create embeddings

word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)



Issues:

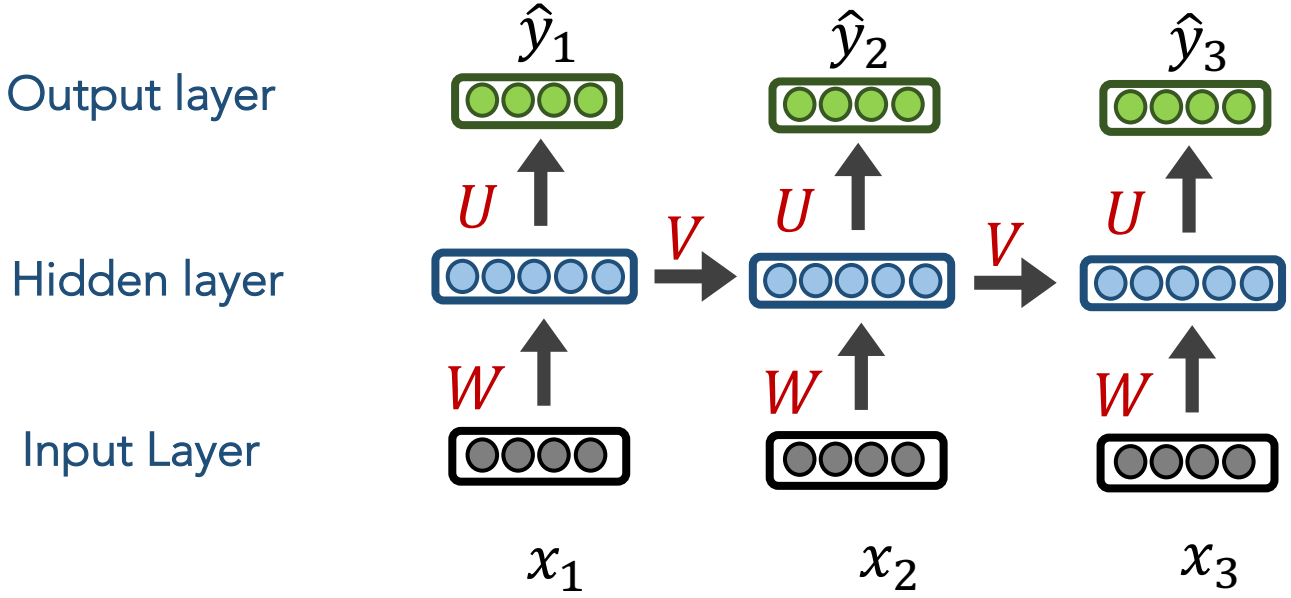
- Out-of-vocabulary (OOV) words
- Not tailored to this dataset

word embeddings (type-based)

approaches:

- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

Previously, we learned about **word embeddings**



LSTM

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

1. Sarma
895
$$$ • Middle Eastern, Tapas/Small Plates, Mediterranean

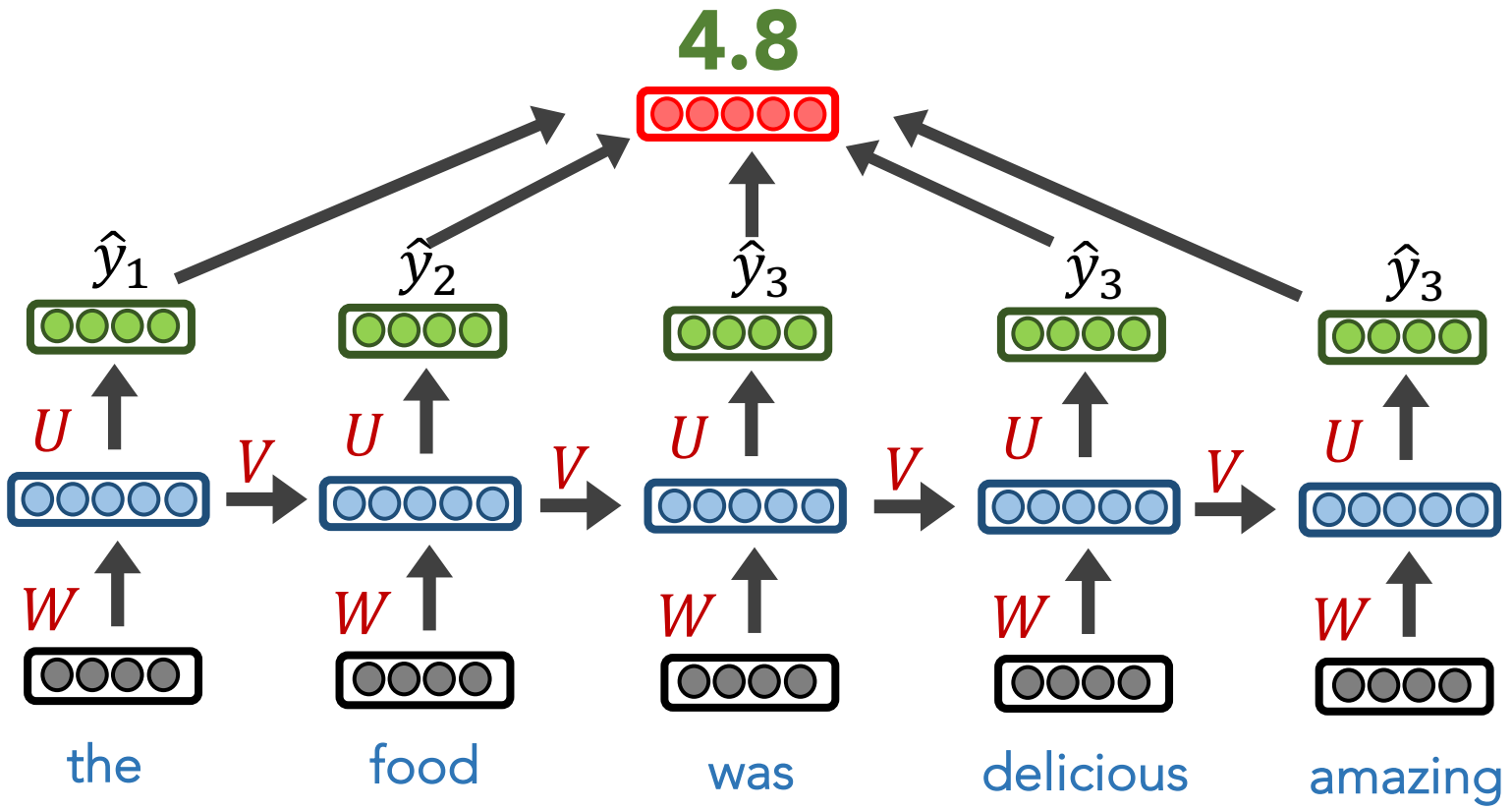
2. Chalawan
108
Thai, Indonesian, Singaporean

3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

5. Avenue Kitchen + Bar
81
$$ • Beer Bar, Pizza, Cocktail Bars

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #1

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

1. Sarma
895
$$$ • Middle Eastern, Tapas/Small Plates, Mediterranean

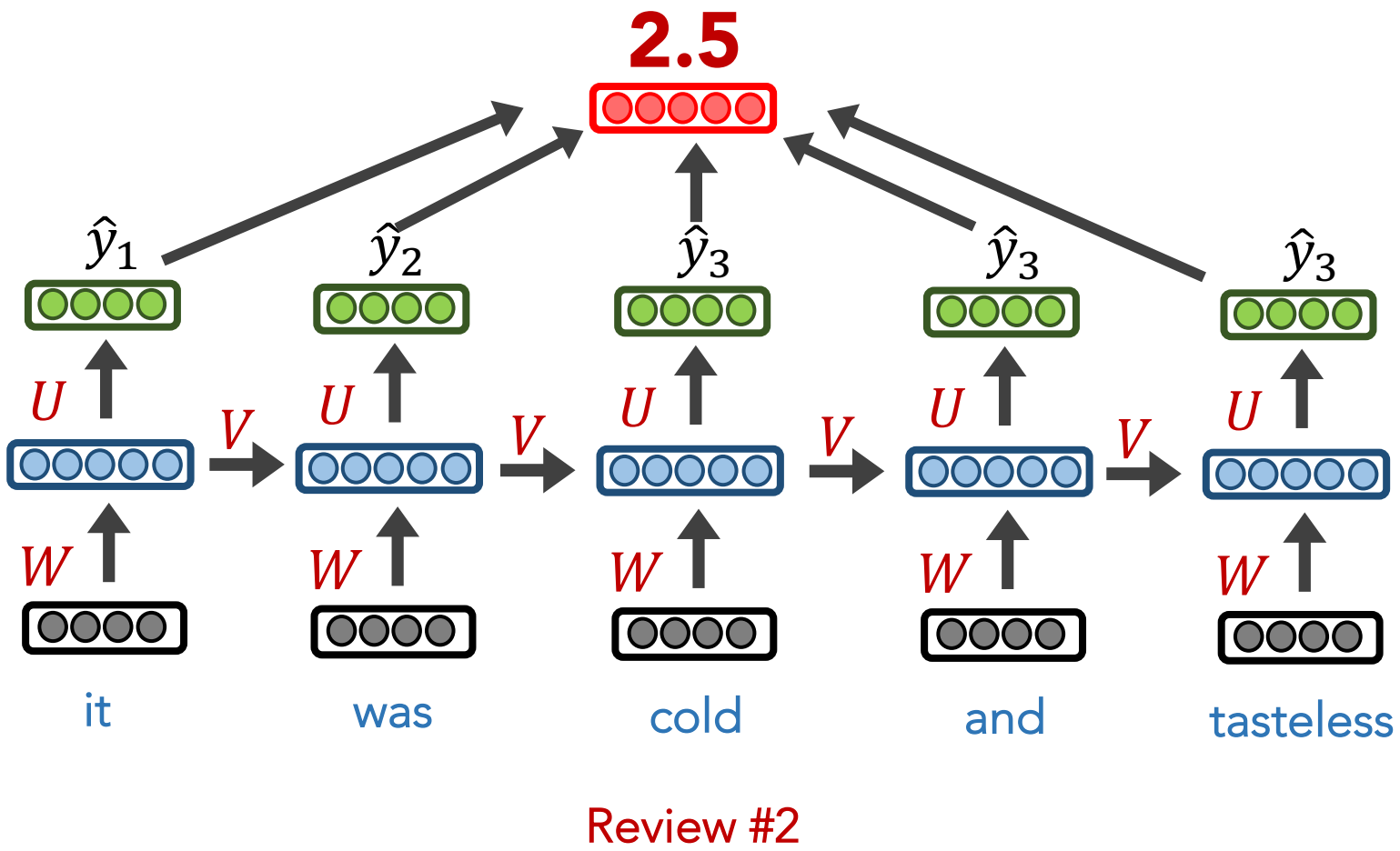
2. Chalawan
108
Thai, Indonesian, Singaporean

3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

5. Avenue Kitchen + Bar
81
$$ • Beer Bar, Pizza, Cocktail Bars

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #2

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

1. Sarma
895
$$$ • Middle Eastern, Tapas/Small Plates, Mediterranean

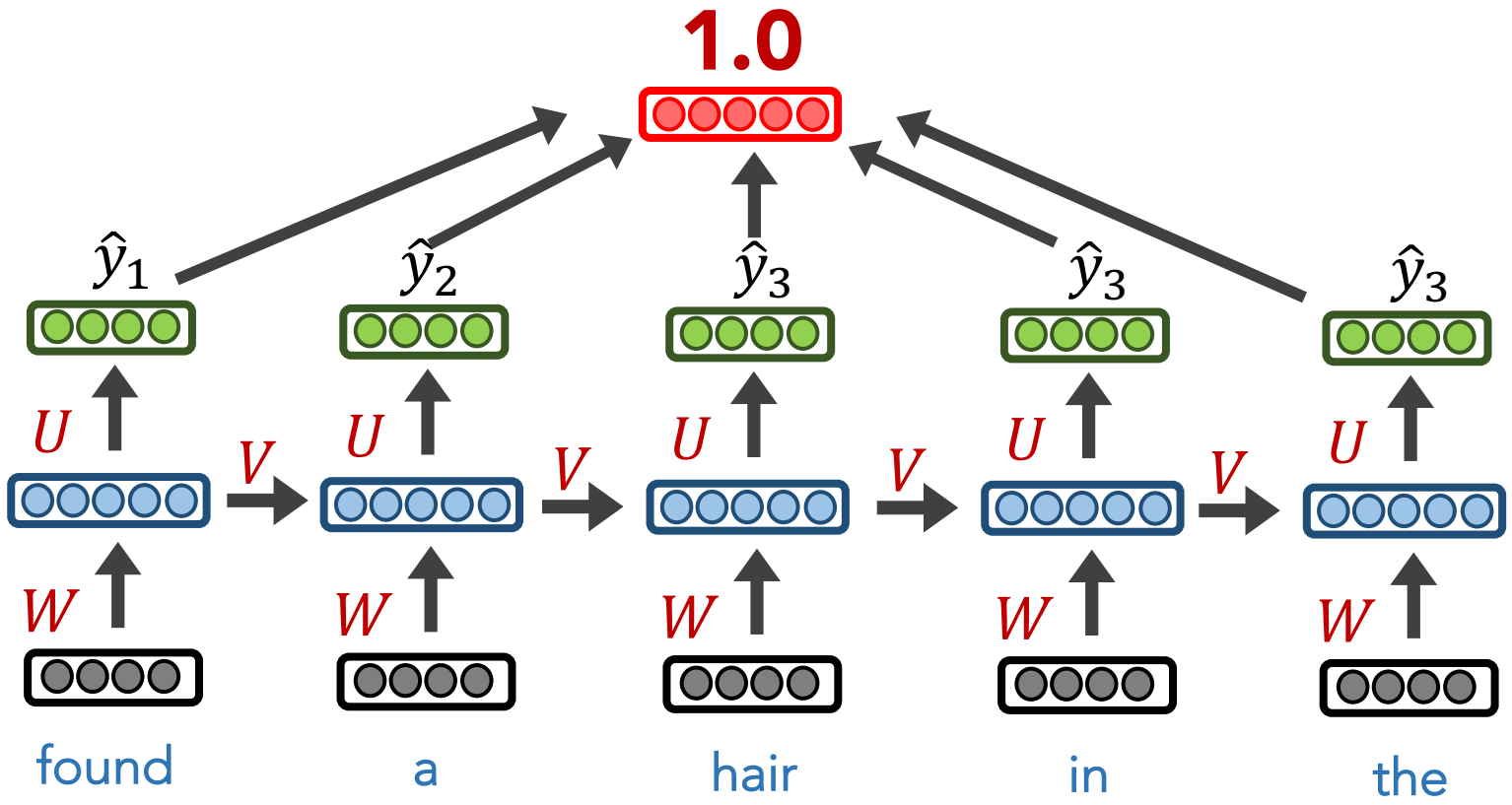
2. Chalawan
108
Thai, Indonesian, Singaporean

3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

5. Avenue Kitchen + Bar
81
$$ • Beer Bar, Pizza, Cocktail Bars

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #53,781

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

1. Sarma
895
$$$ • Middle Eastern, Tapas/Small Plates, Mediterranean

2. Chalawan
108
Thai, Indonesian, Singaporean

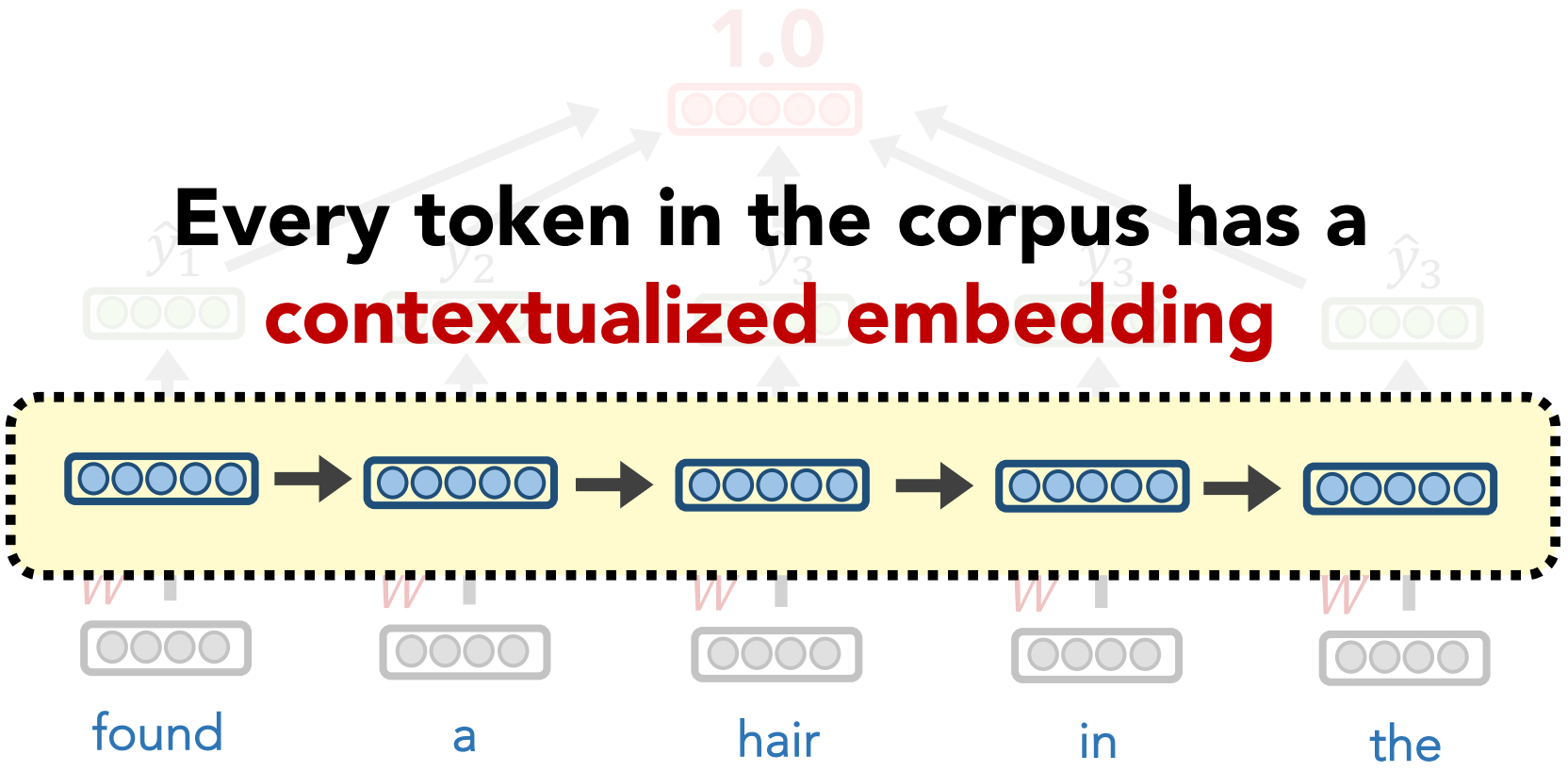
3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

5. Avenue Kitchen + Bar
81
$$ • Beer Bar, Pizza, Cocktail Bars

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop

Every token in the corpus has a contextualized embedding



Review #53,781

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

1. Sarma
895
$$$ • Middle Eastern, Tapas/Small Plates, Mediterranean

2. Chalawan
108
Thai, Indonesian, Singaporean

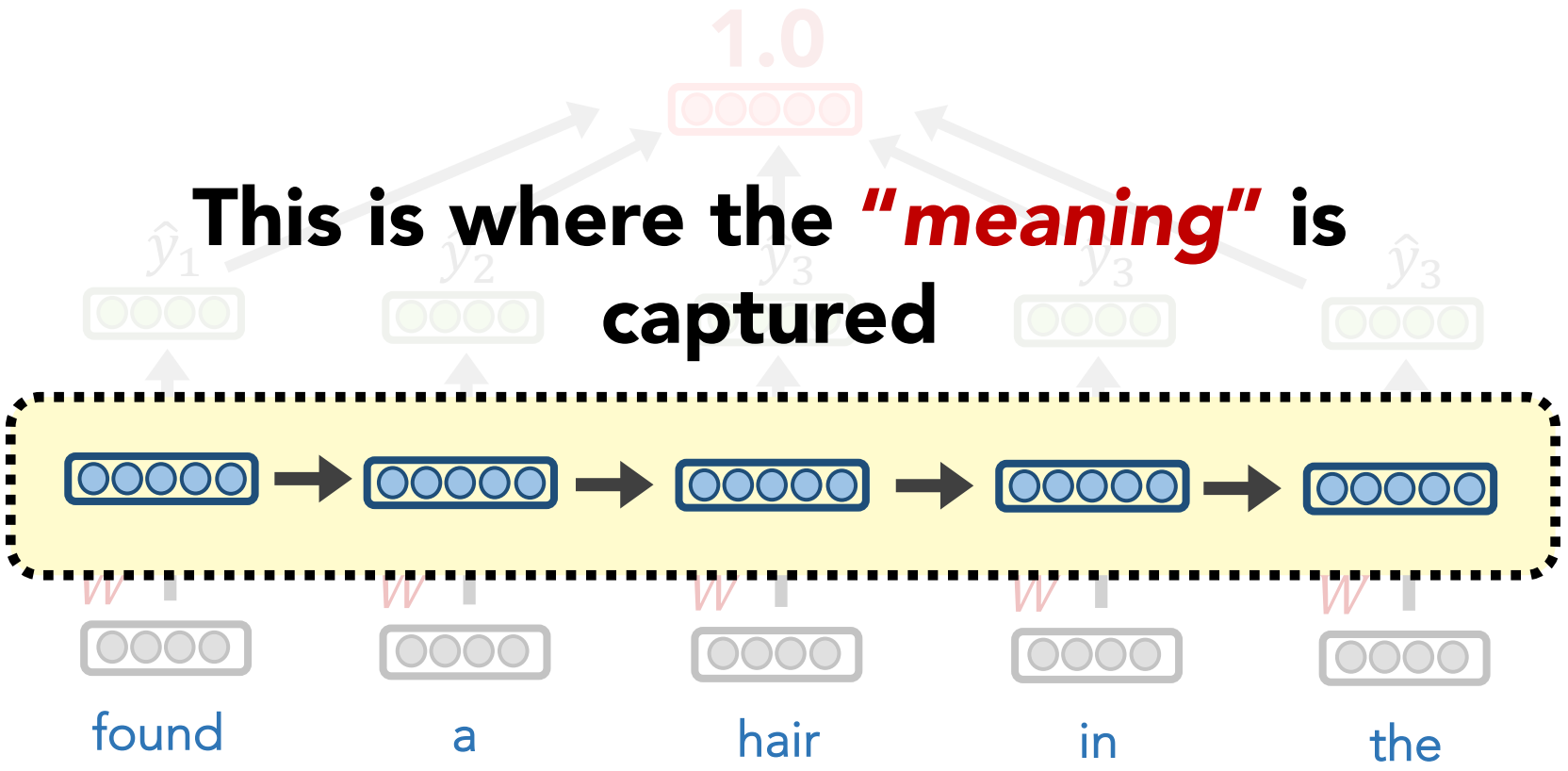
3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

5. Avenue Kitchen + Bar
81
$$ • Beer Bar, Pizza, Cocktail Bars

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop

This is where the *"meaning"* is captured



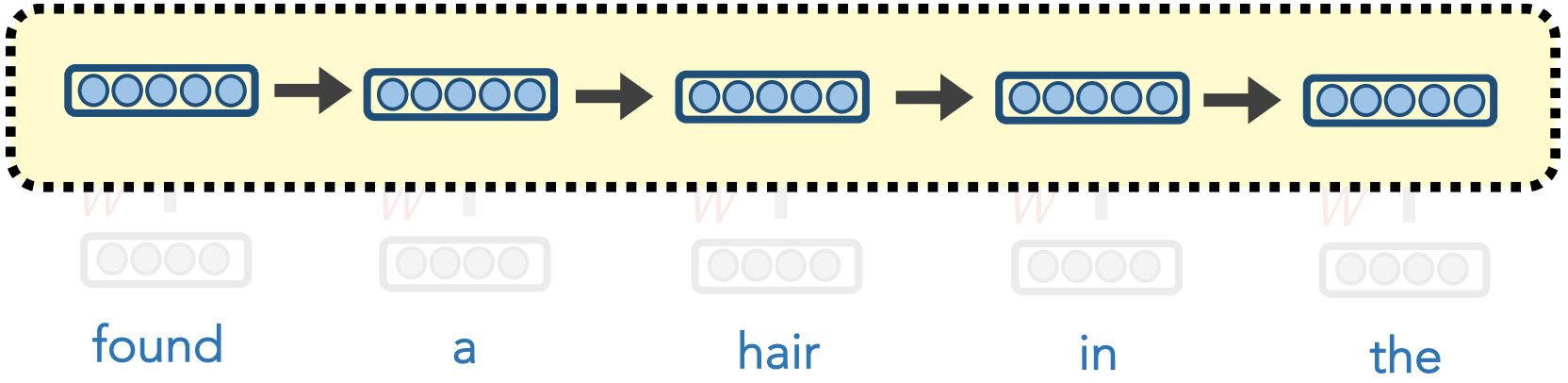
Review #53,781

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

Strengths and weaknesses of contextualized embeddings (aka token-based)?



Review #53,781

contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

Strengths:

- Tailored to your particular corpus
- No out-of-vocabulary (OOV) words

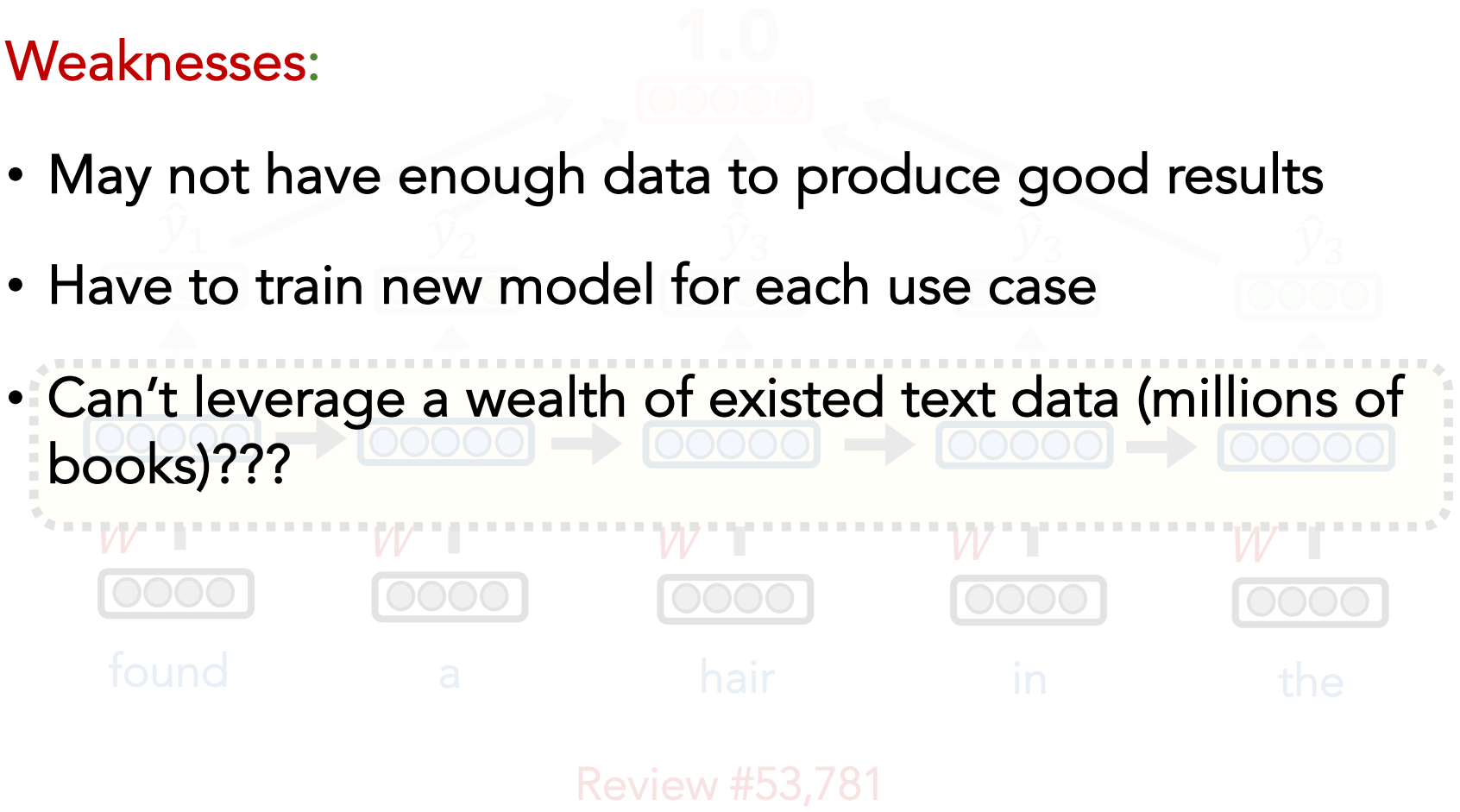
contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

Weaknesses:

- May not have enough data to produce good results
- Have to train new model for each use case
- Can't leverage a wealth of existed text data (millions of books)???



contextualized embeddings (token-based)

approaches:

- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

Weaknesses:

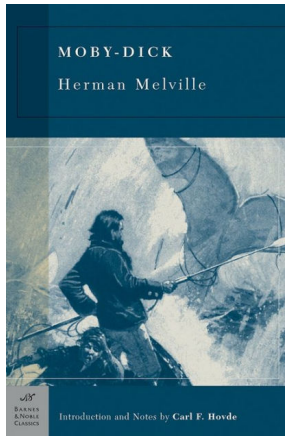
- May not have enough data to produce good results
- Have to train new model for each use case
- ~~Can't leverage a wealth of existed text data (millions of books)???~~

WRONG! We can leverage millions of books!

contextualized embeddings (token-based)

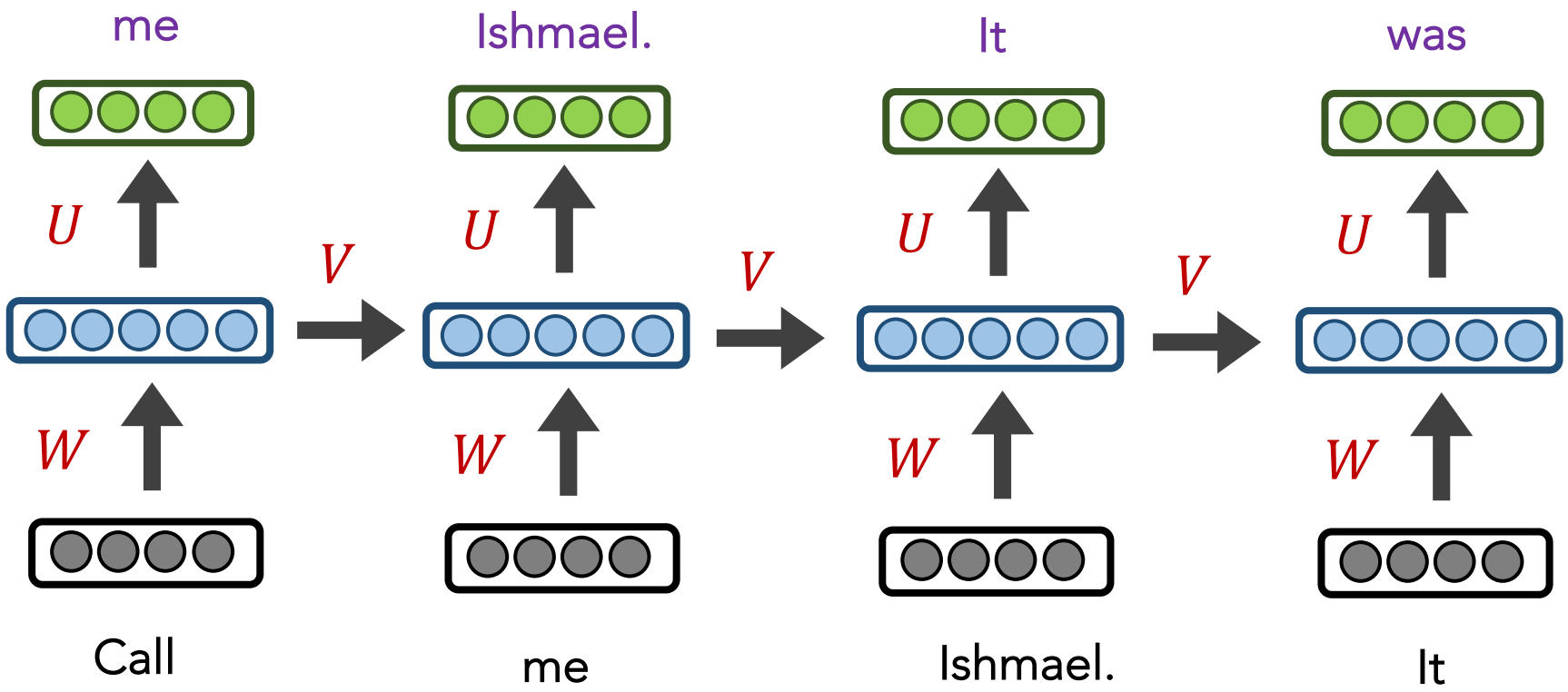
approaches:

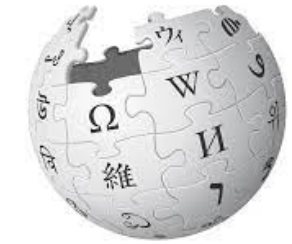
- Predictive models (e.g., BiLSTMs, GPT-2, BERT)



Language Modelling

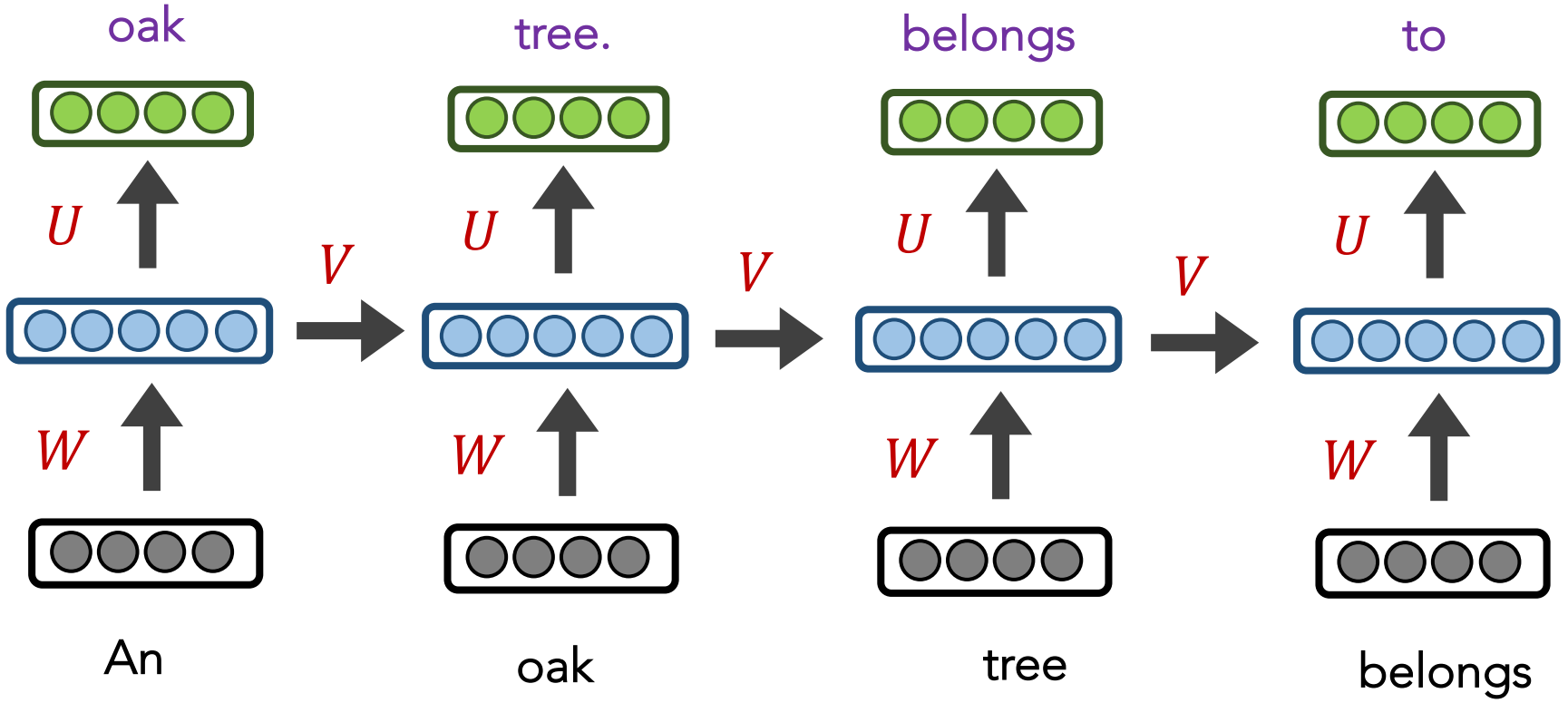
(let's input 1 million documents)





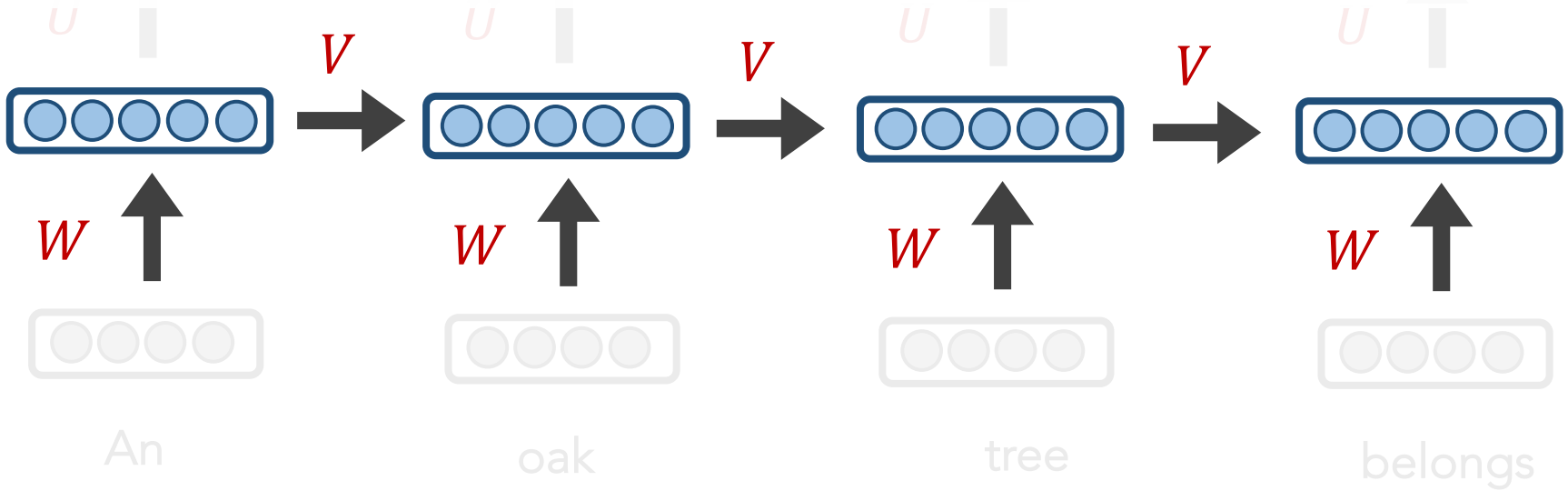
Language Modelling

(let's input 1 million documents)

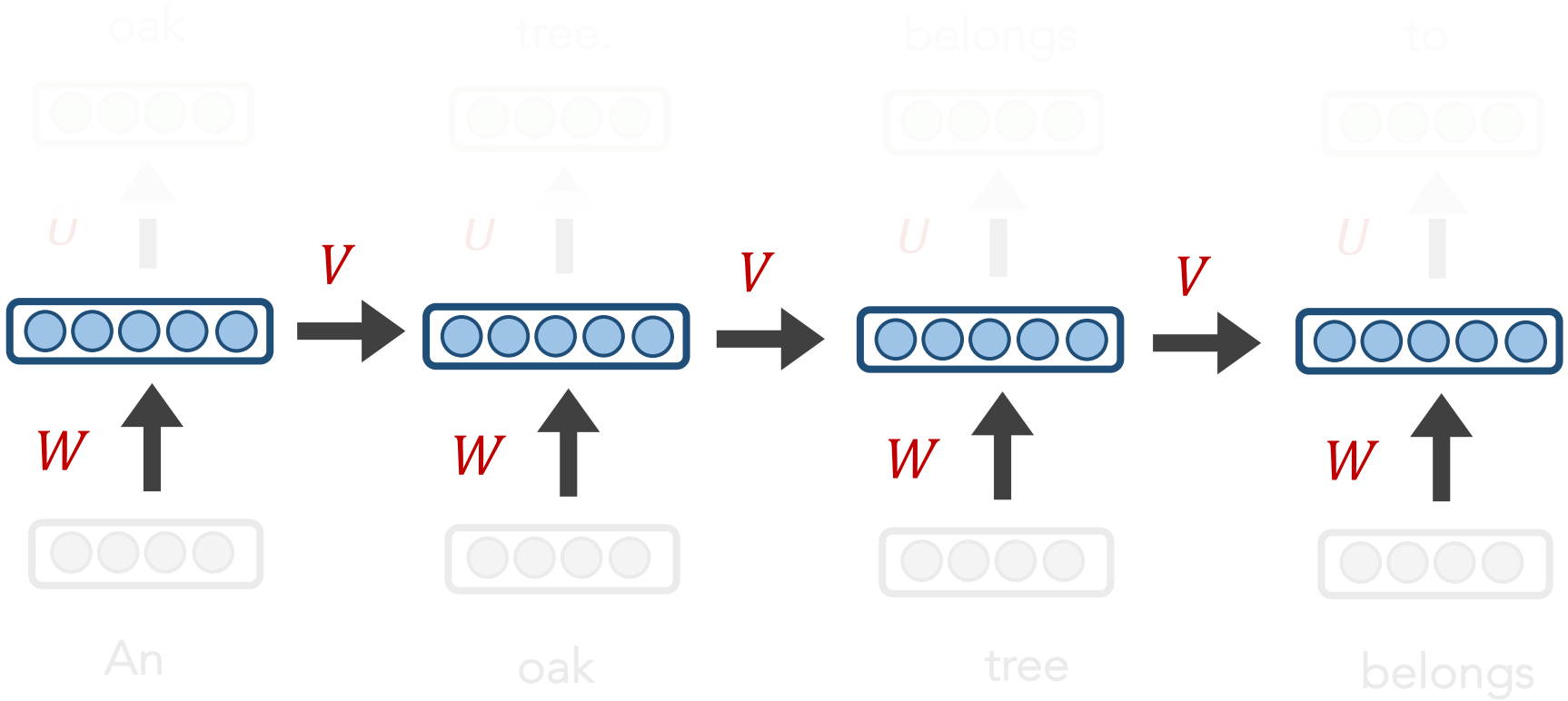


The contextualized embeddings for 1 million docs aren't useful to us for a new task (e.g., predicting Yelp reviews), but the learned weights could be!

Learn a rich, robust W and V



Using these "pre-trained" W and V, we can possibly increase our performance on other tasks (e.g., Yelp reviews), since they're very experienced with producing/capturing "meaning"



RECAP

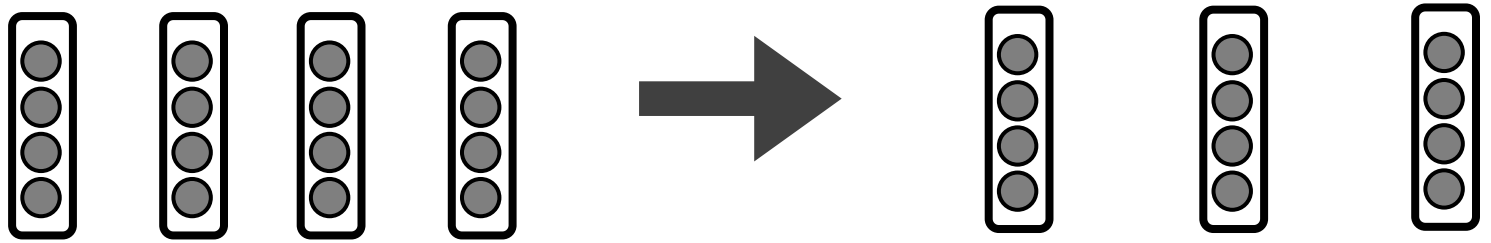
- Language Modelling may help us for other tasks
- LSTMs do a great job of capturing "*meaning*", which can be used for almost every task
  - Given a sequence of N words, we can produce 1 output
  - Given a sequence of N words, we can produce N outputs

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)



Thank you for visiting!

Děkujeme za návštěvu!

Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

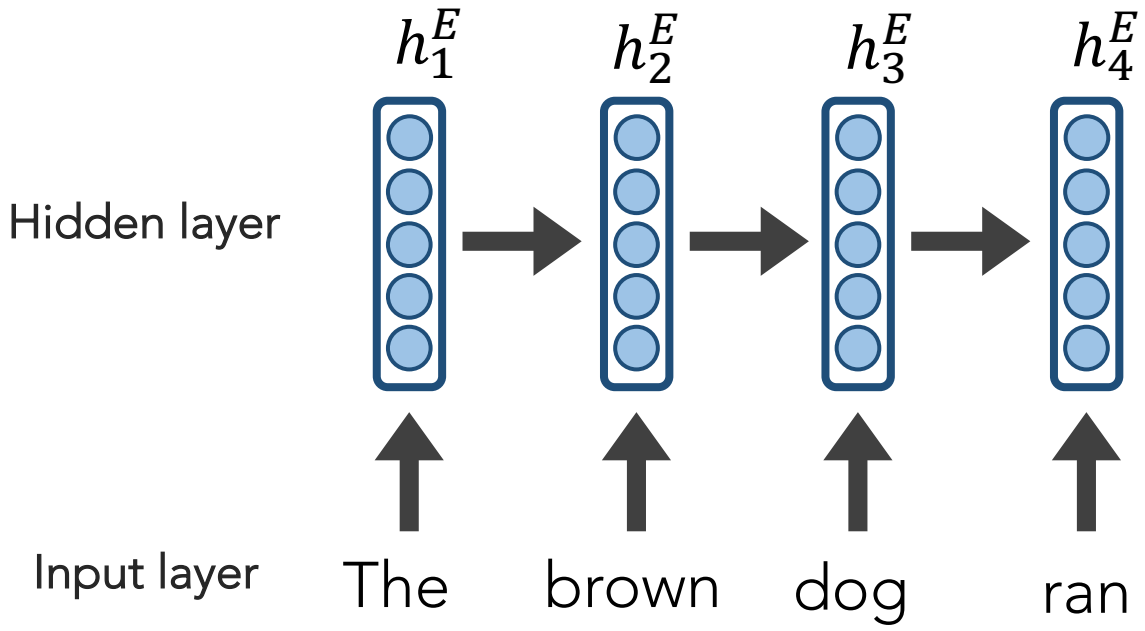
Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

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

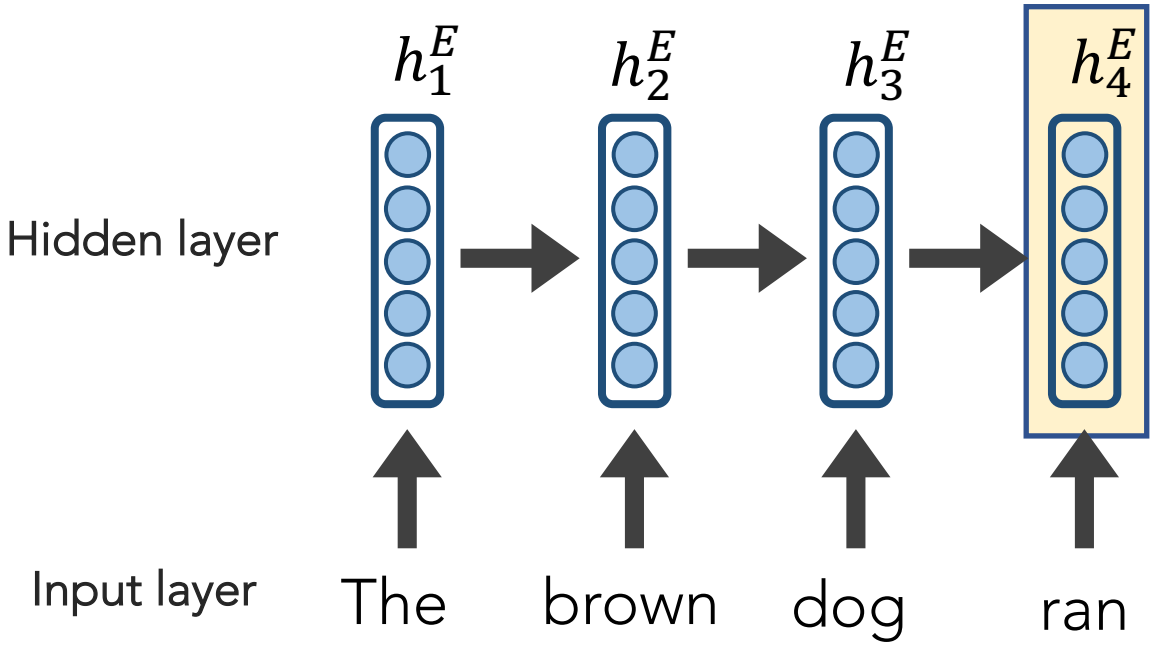
Sequence-to-Sequence (seq2seq)



ENCODER RNN

Sequence-to-Sequence (seq2seq)

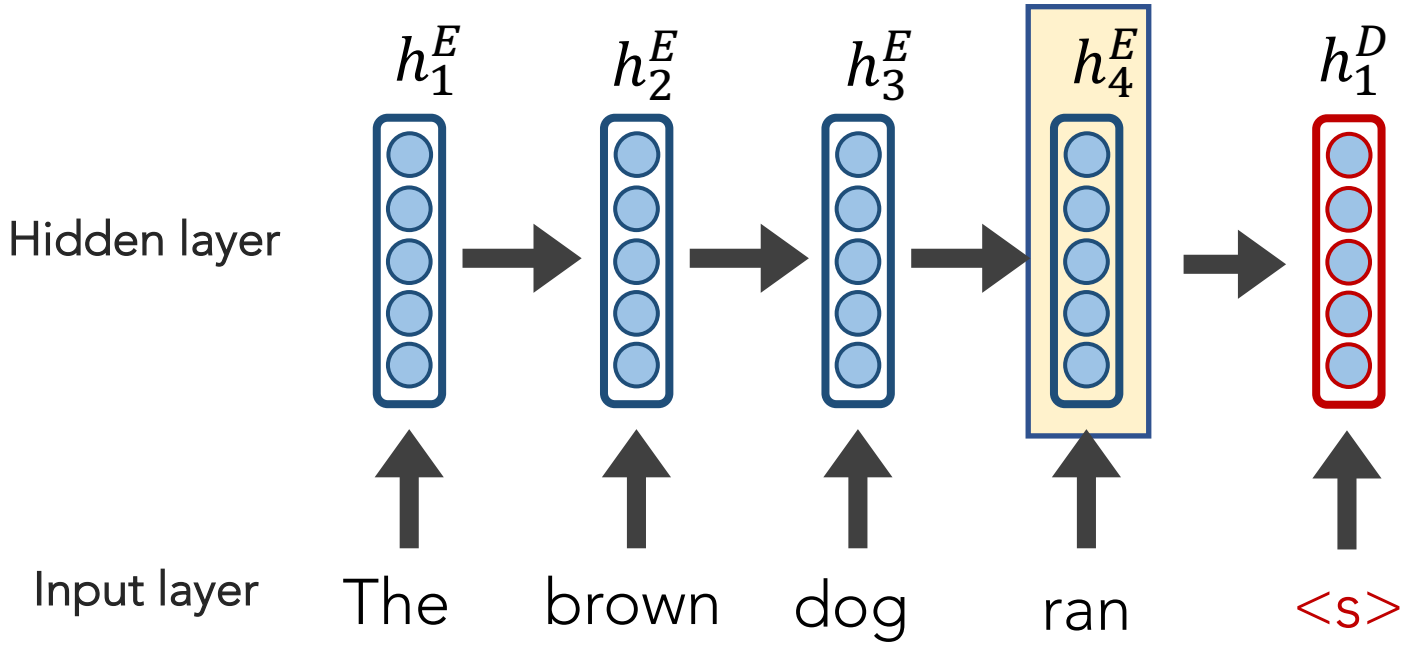
The final hidden state of the encoder RNN
is the initial state of the decoder RNN



ENCODER RNN

Sequence-to-Sequence (seq2seq)

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

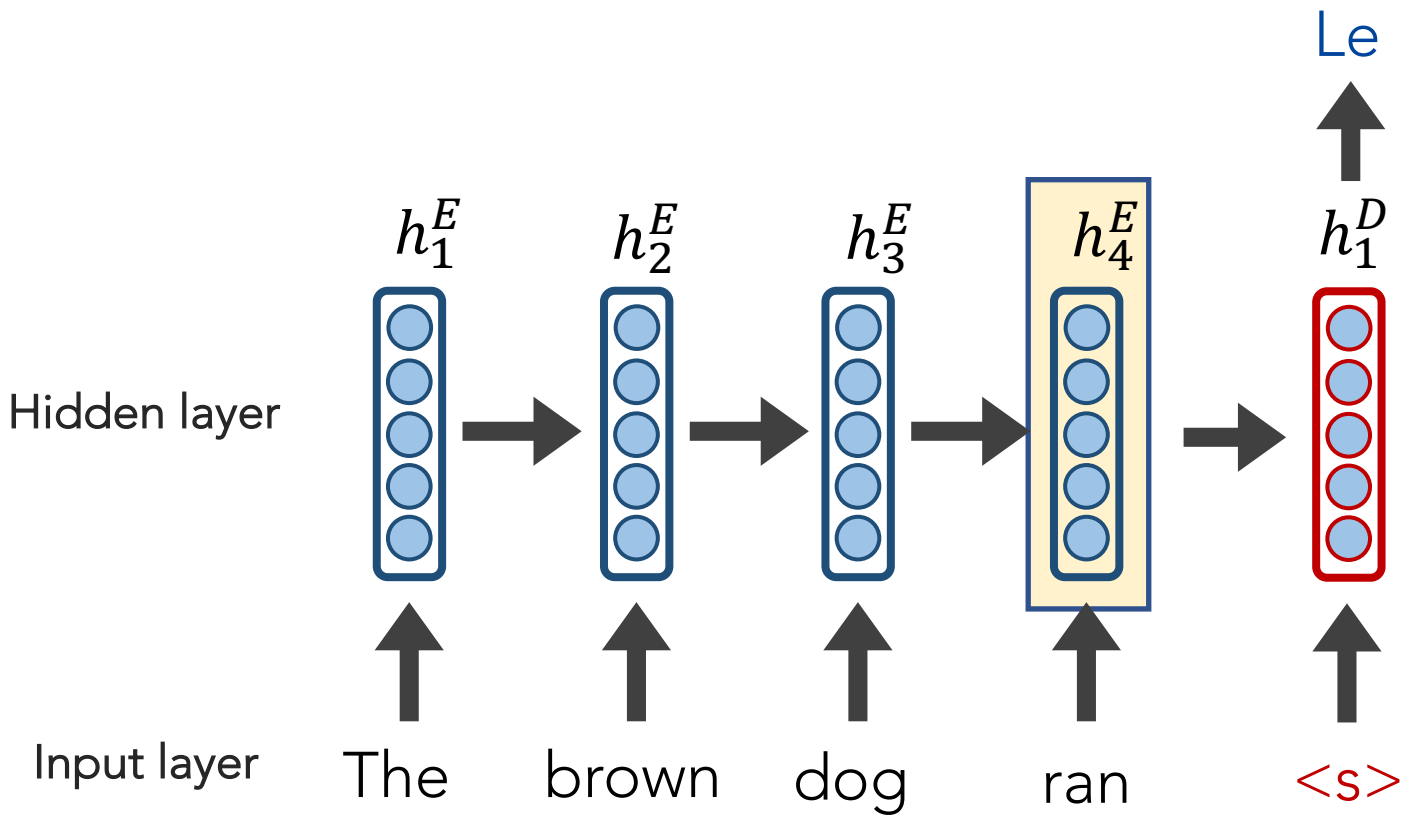


ENCODER RNN

DECODER RNN

Sequence-to-Sequence (seq2seq)

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

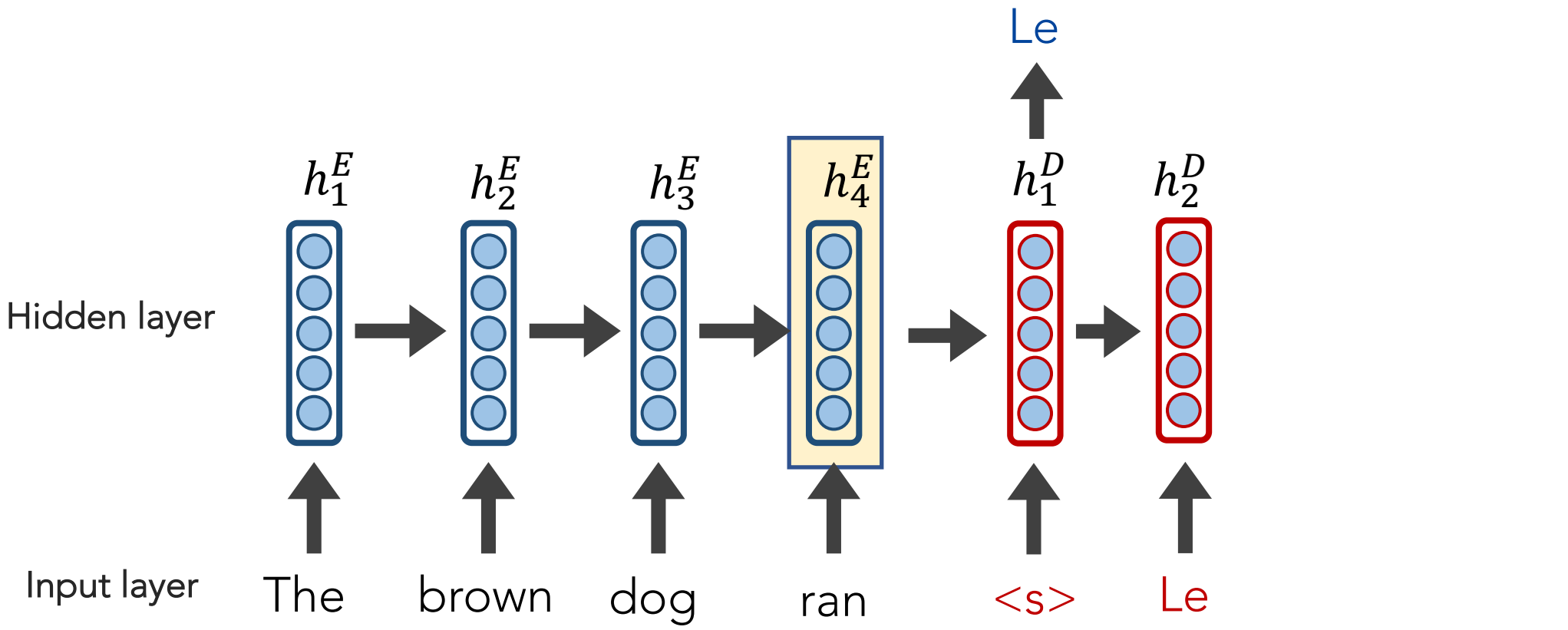


ENCODER RNN

DECODER RNN

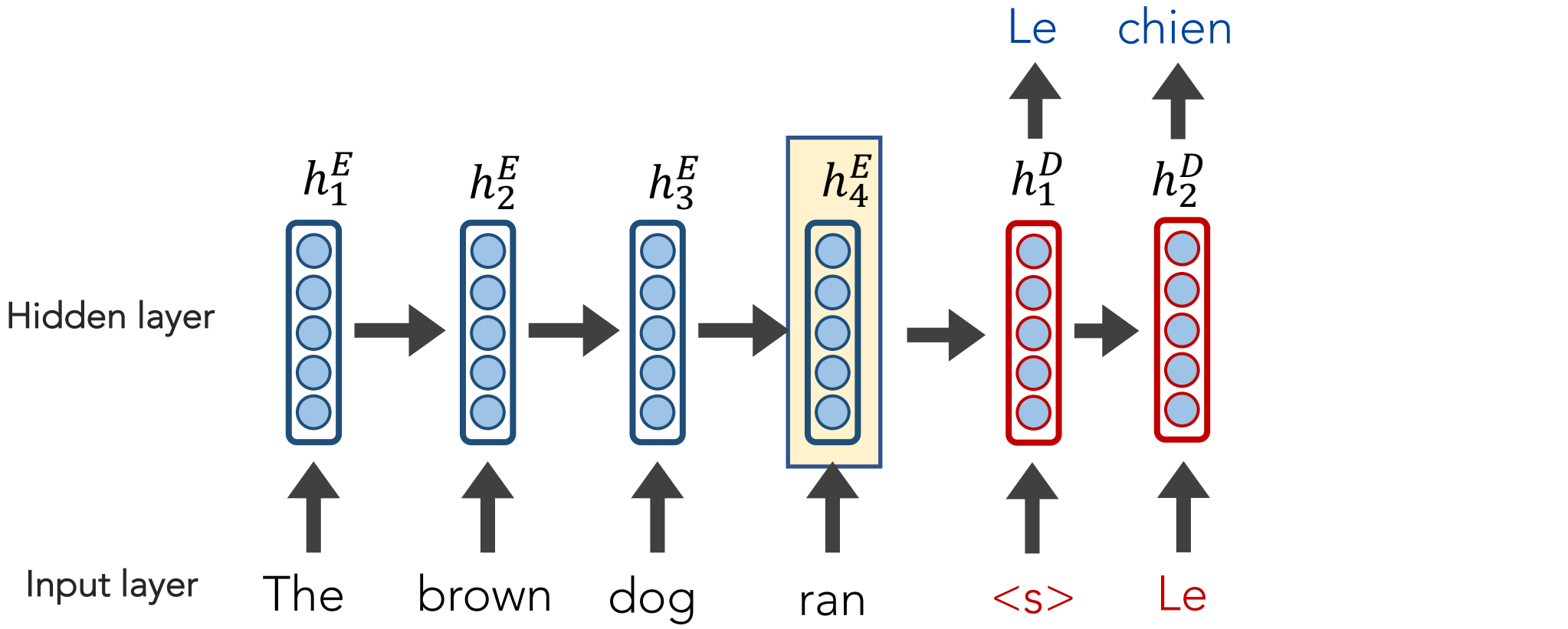
Sequence-to-Sequence (seq2seq)

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



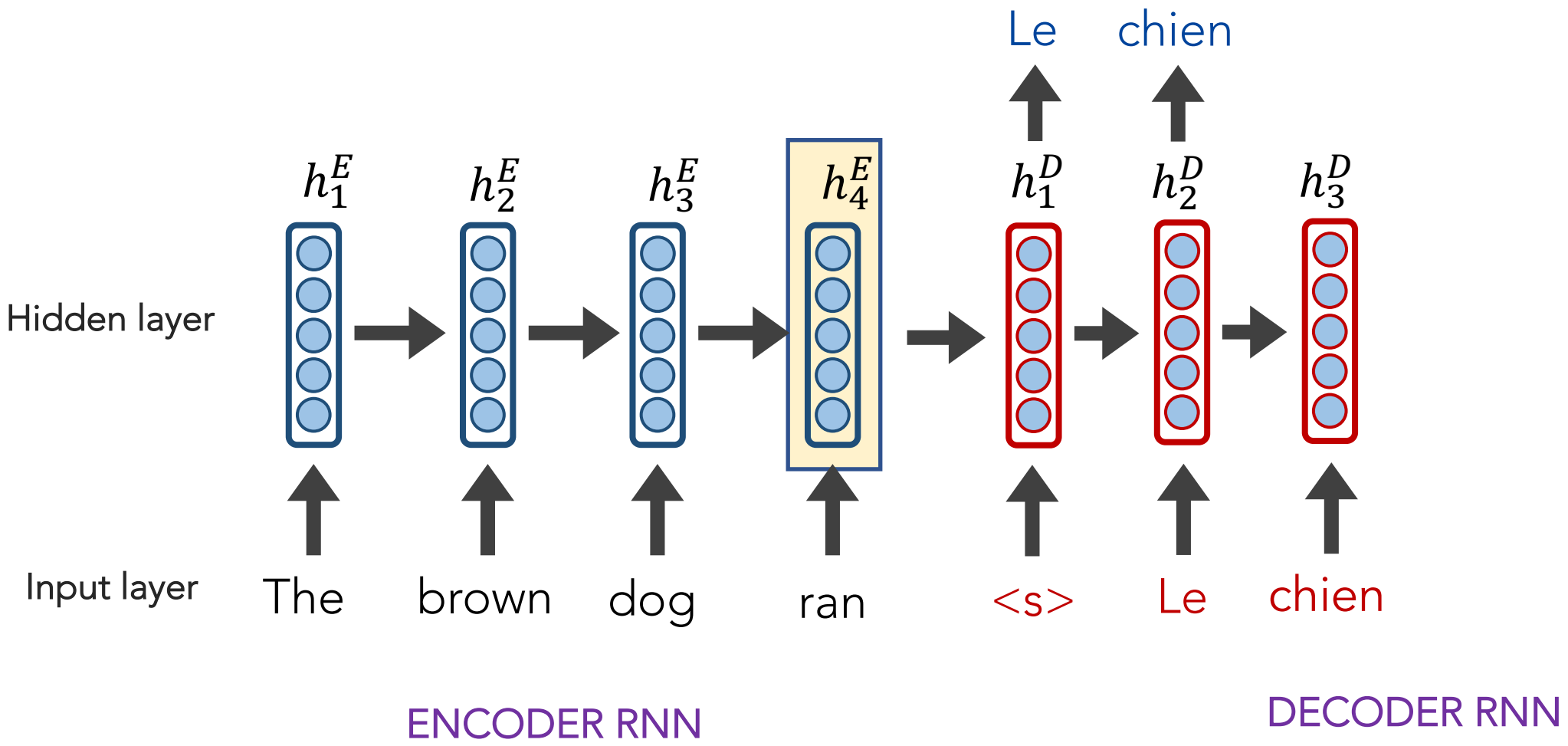
Sequence-to-Sequence (seq2seq)

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



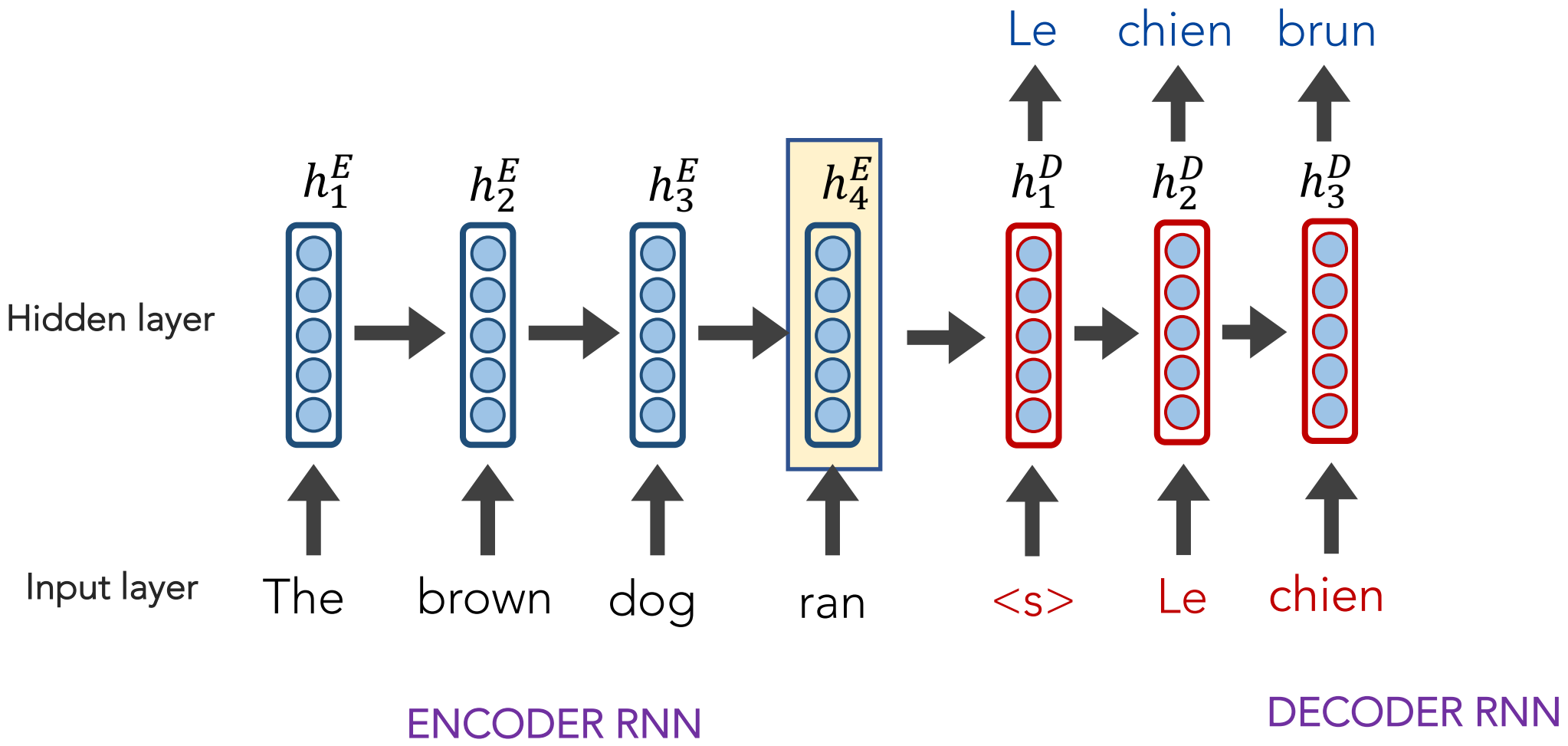
Sequence-to-Sequence (seq2seq)

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



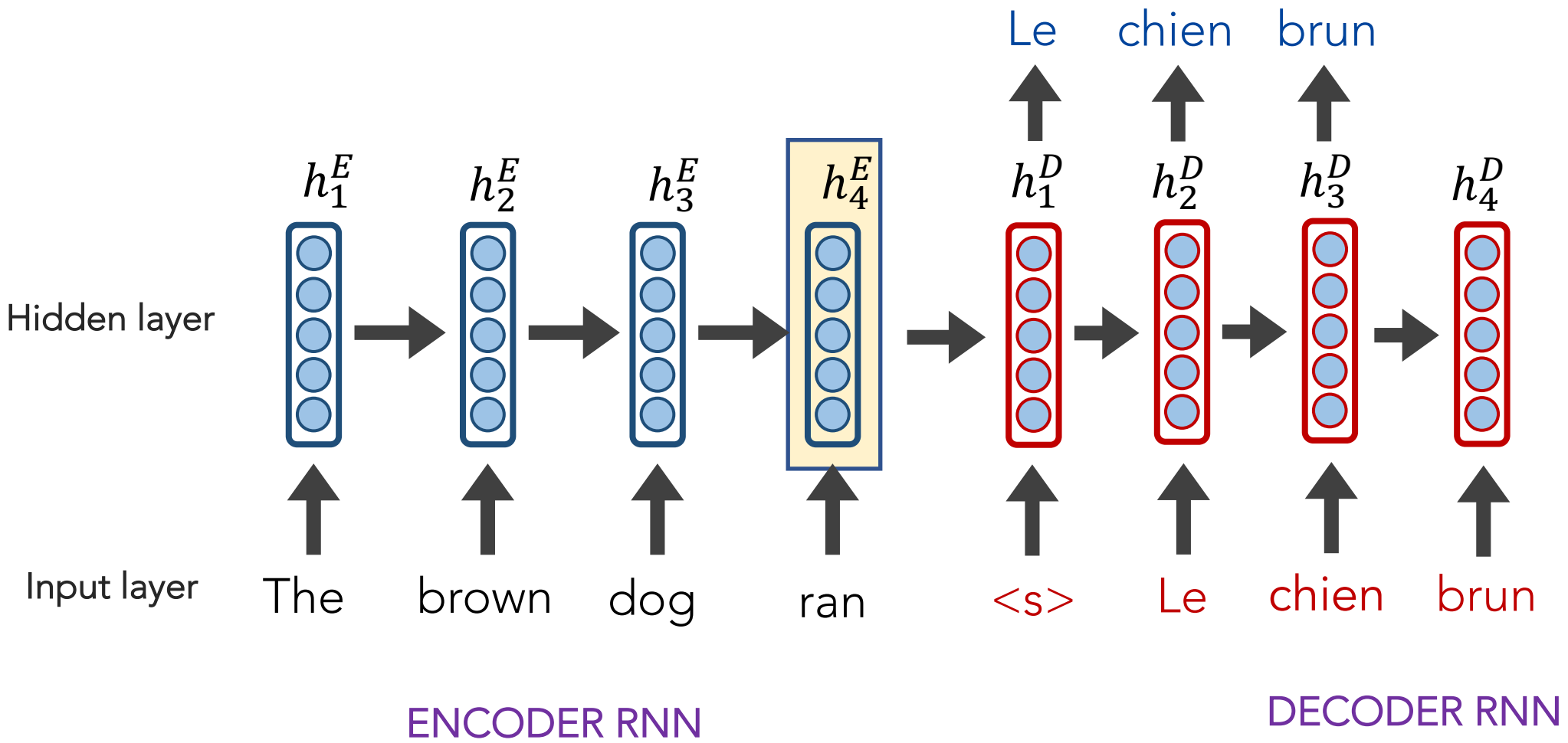
Sequence-to-Sequence (seq2seq)

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



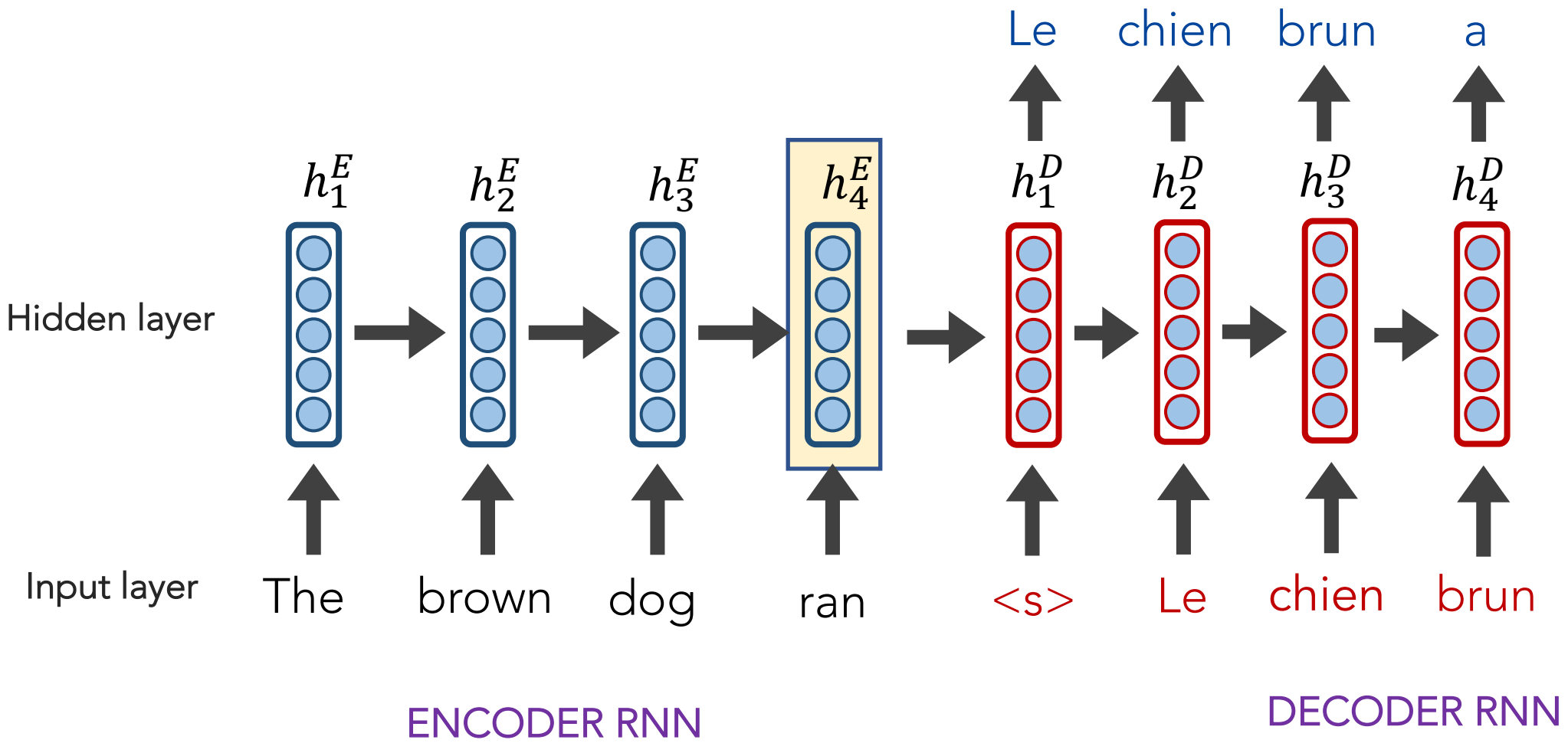
Sequence-to-Sequence (seq2seq)

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



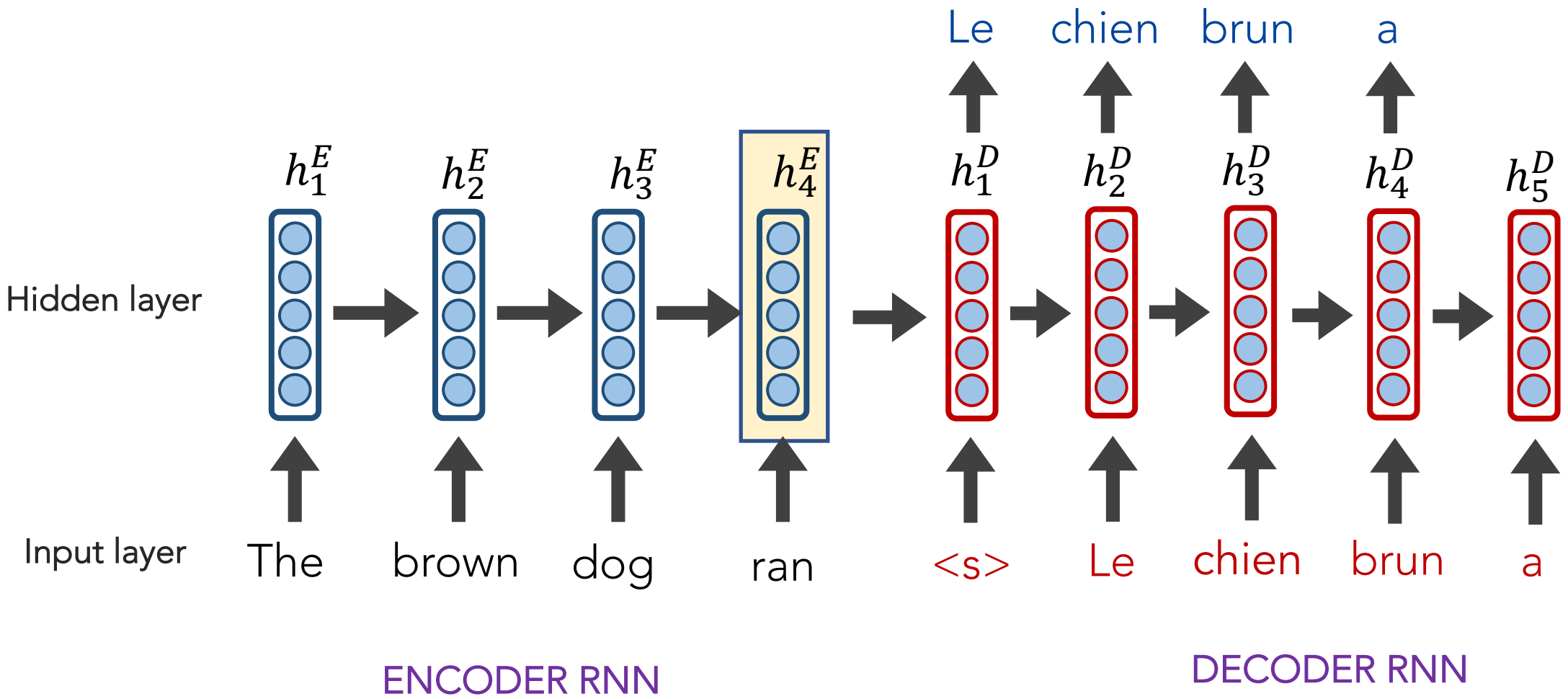
Sequence-to-Sequence (seq2seq)

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

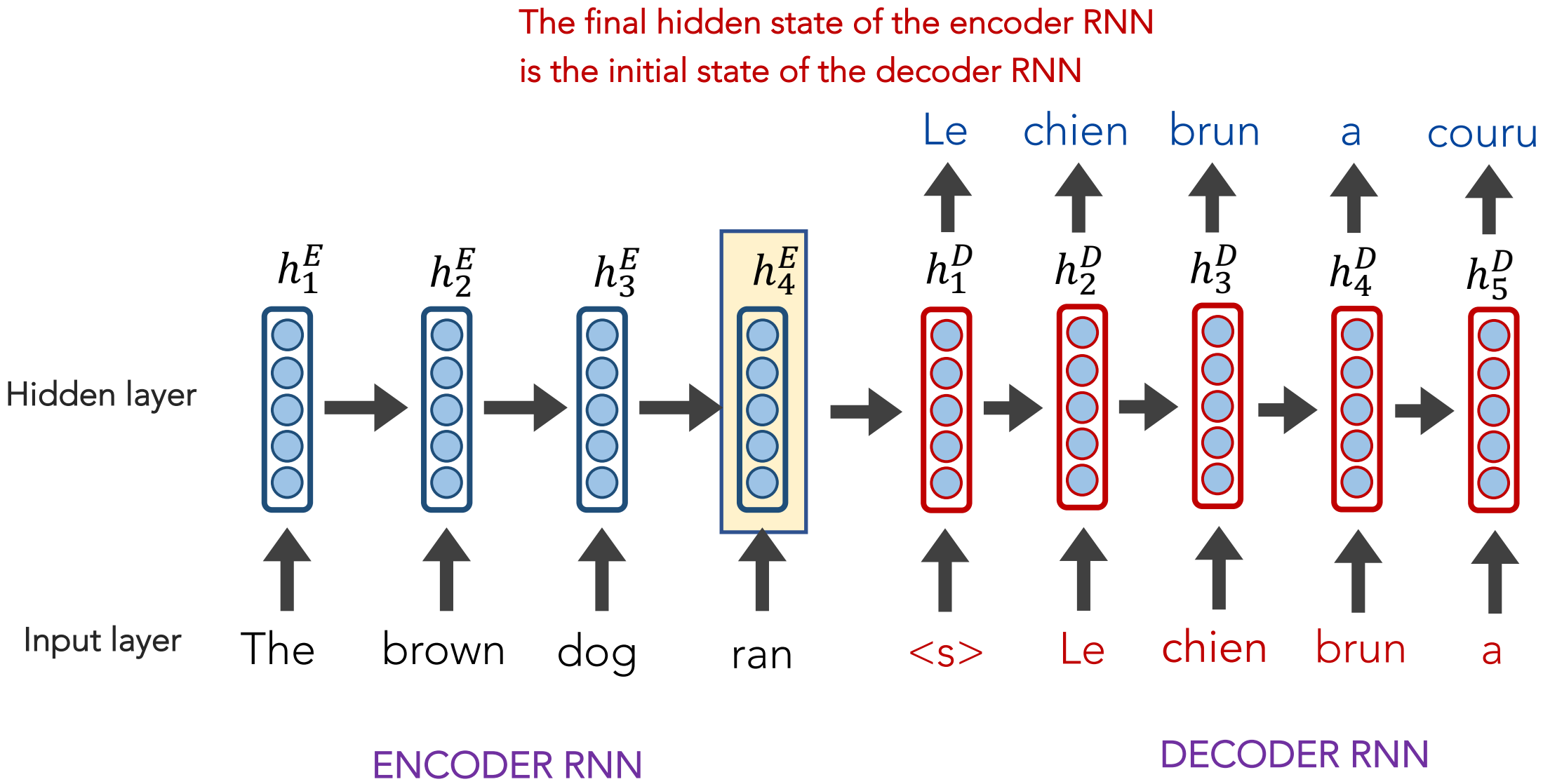


Sequence-to-Sequence (seq2seq)

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

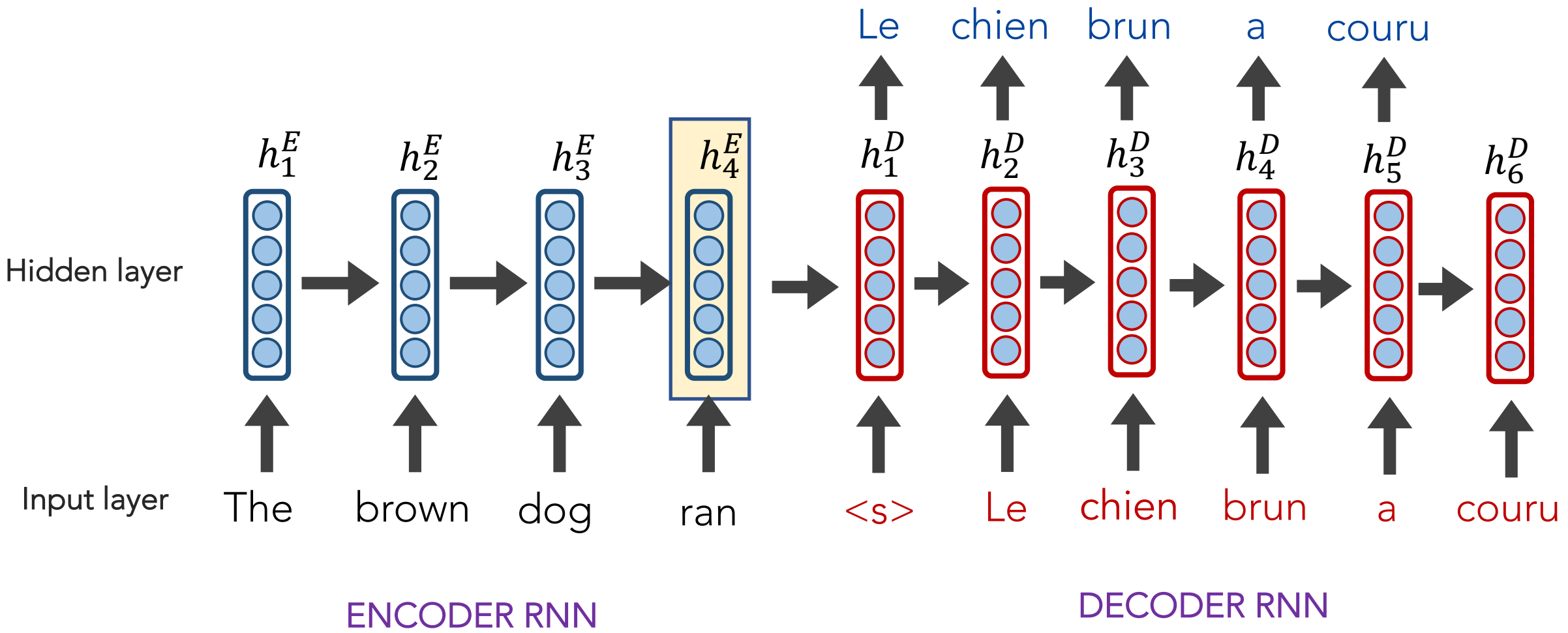


Sequence-to-Sequence (seq2seq)



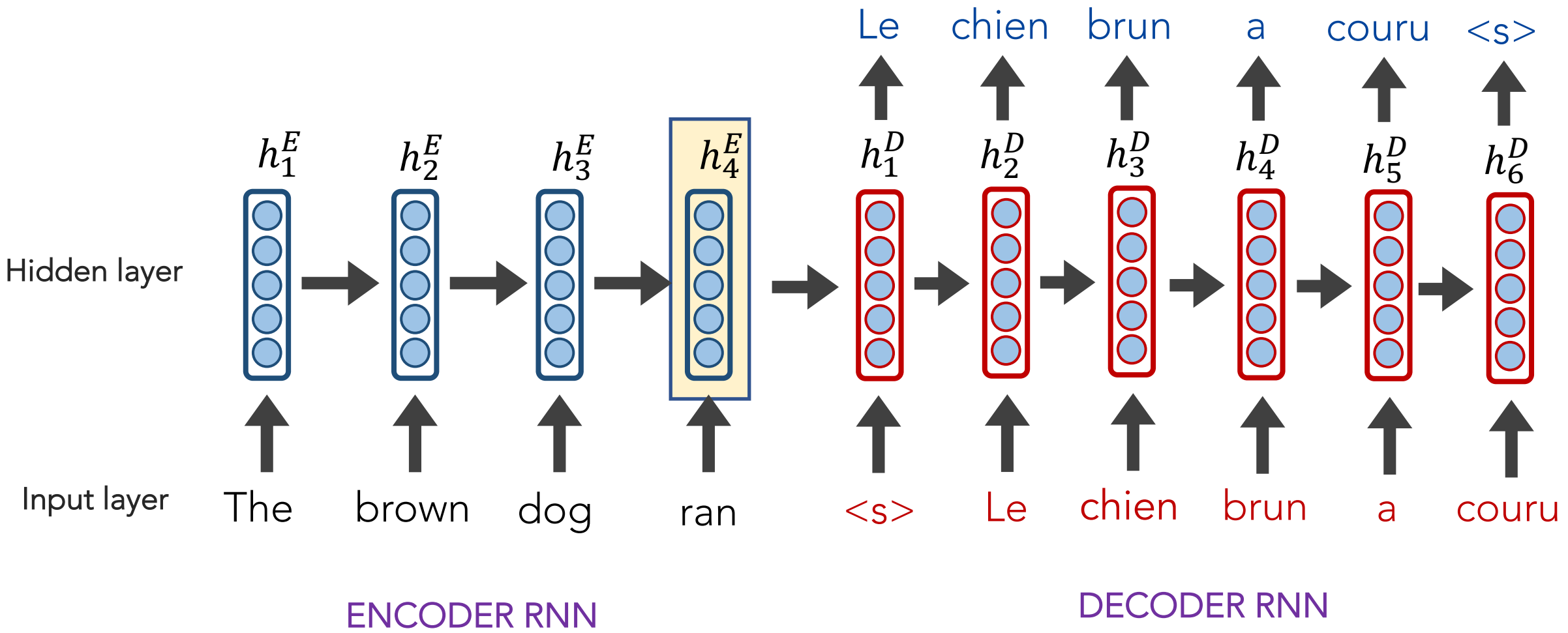
Sequence-to-Sequence (seq2seq)

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

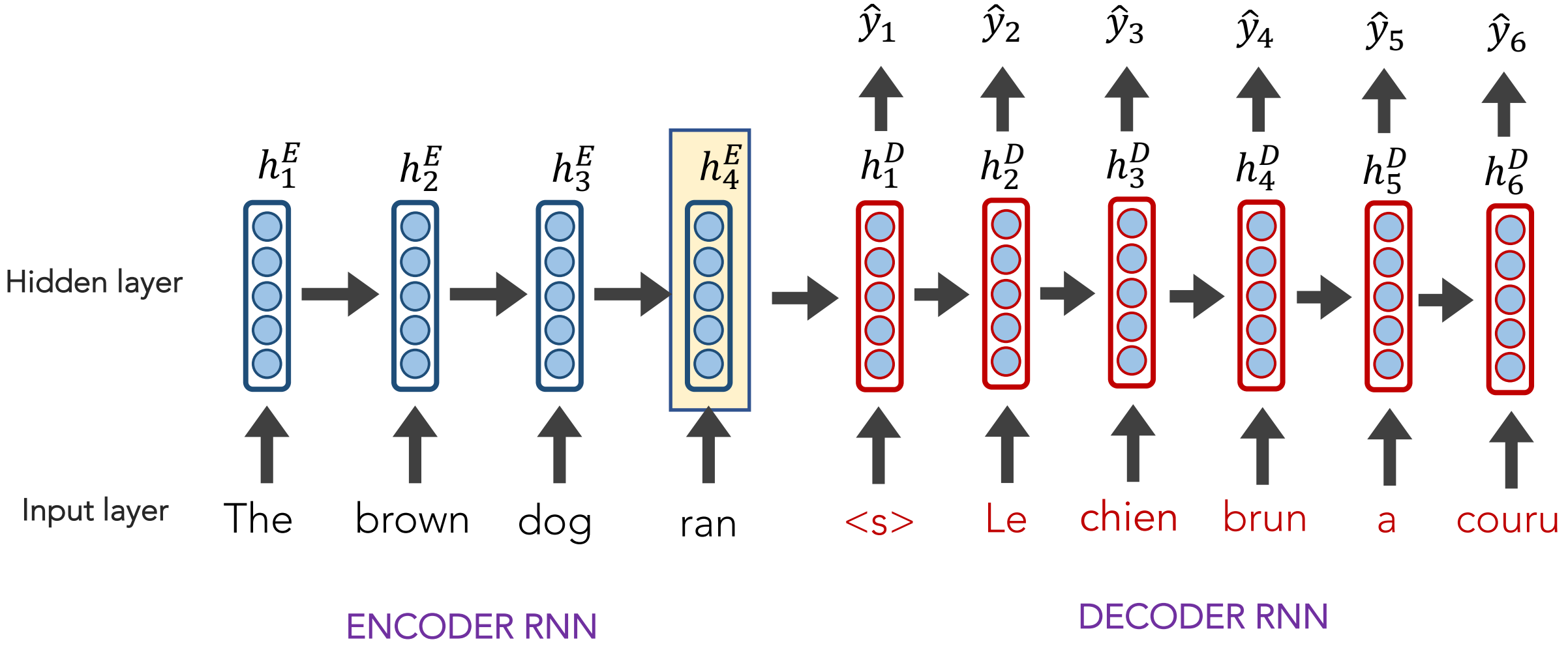


Sequence-to-Sequence (seq2seq)

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

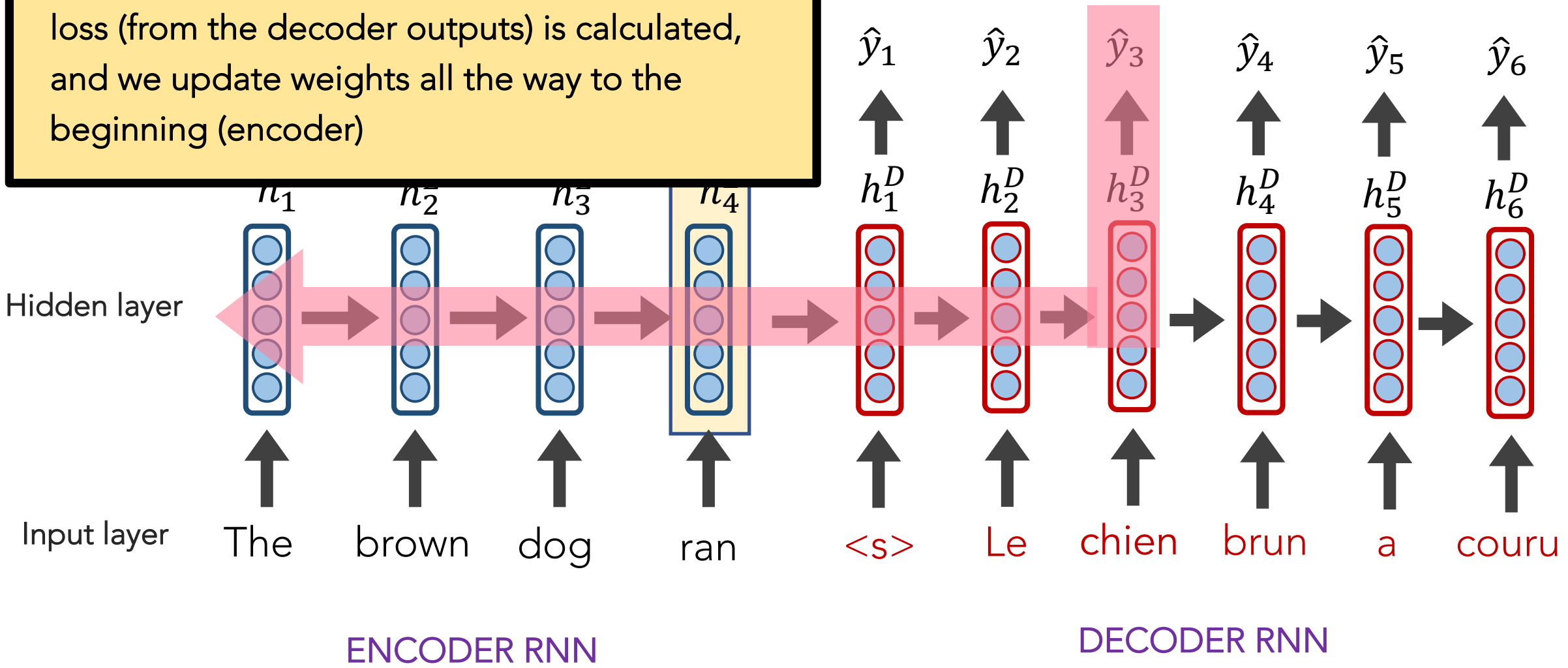


Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

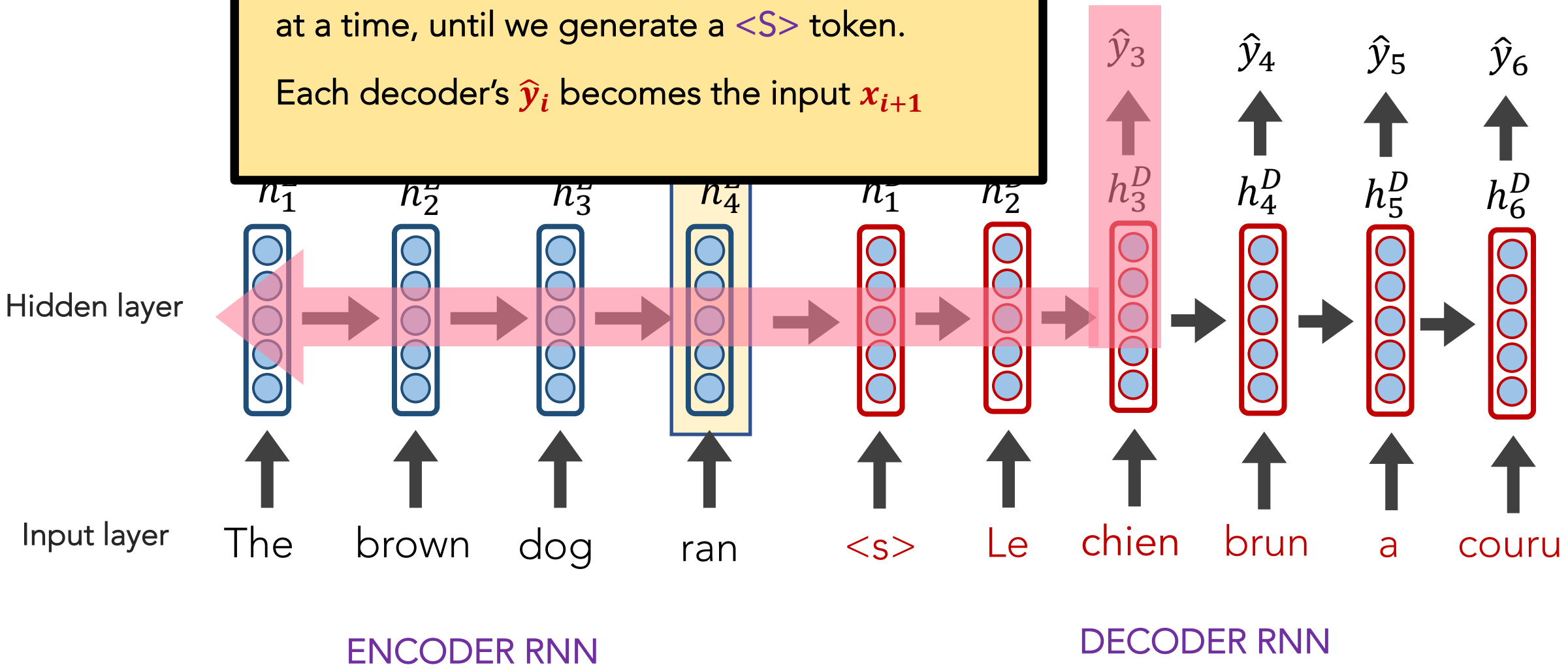
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

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

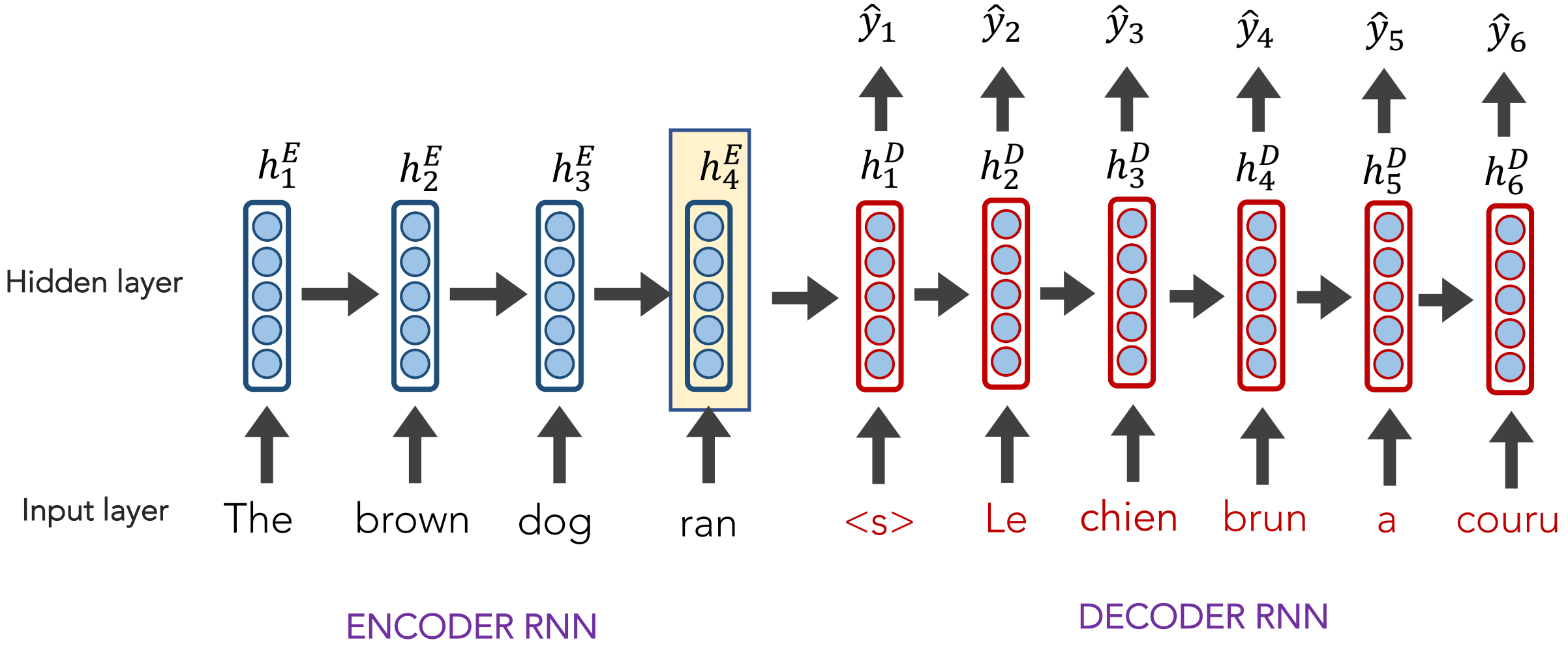
Each decoder's $\hat{y}_i$ becomes the input x_{i+1}



Sequence-to-Sequence (seq2seq)

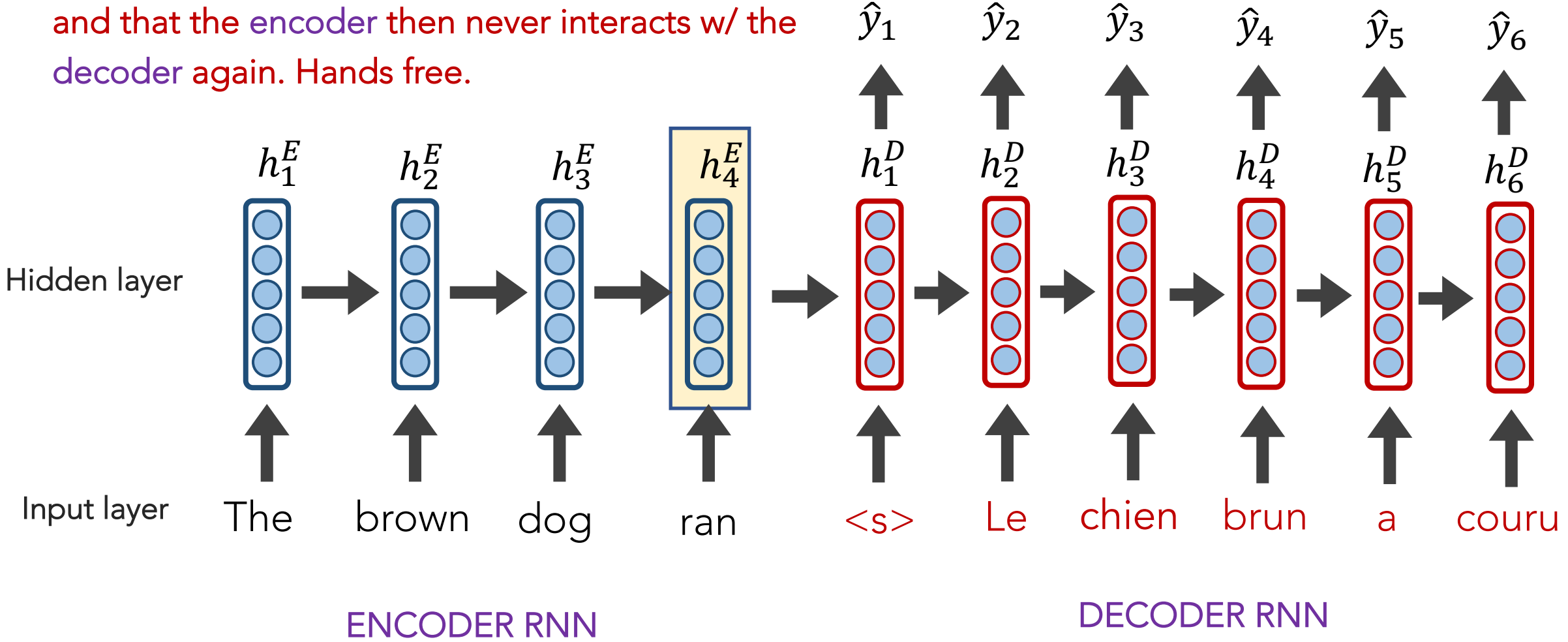
See any issues with this traditional **seq2seq** paradigm?

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 then regurgitate in a new language; we continuously look back at the original while focusing on different parts.

Outline

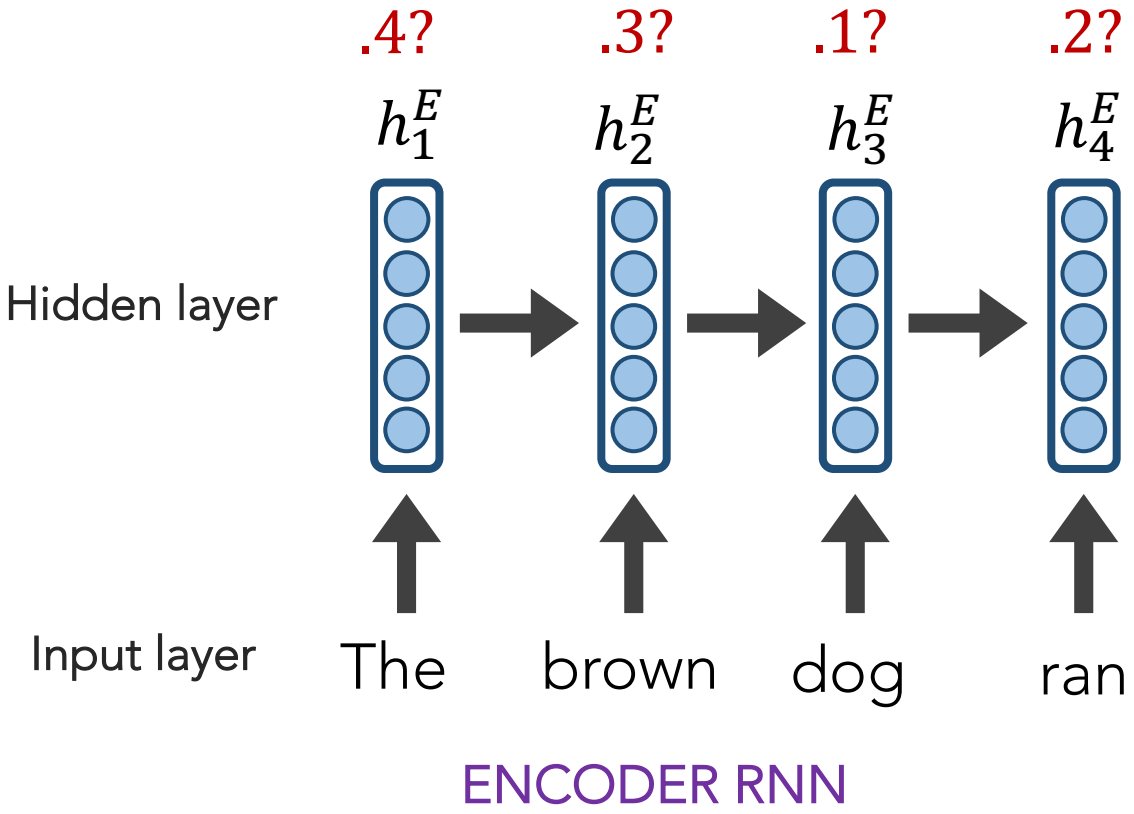
- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

seq2seq + Attention

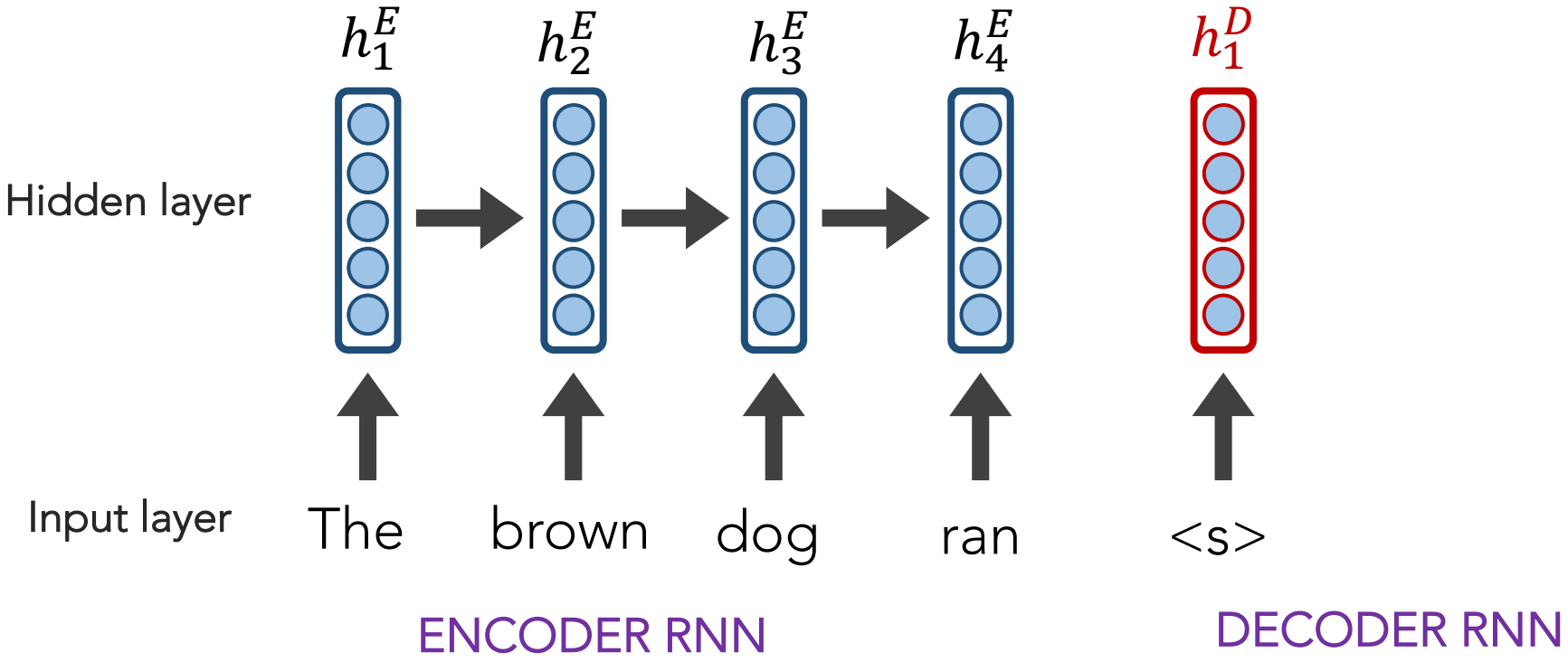
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



seq2seq + Attention

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

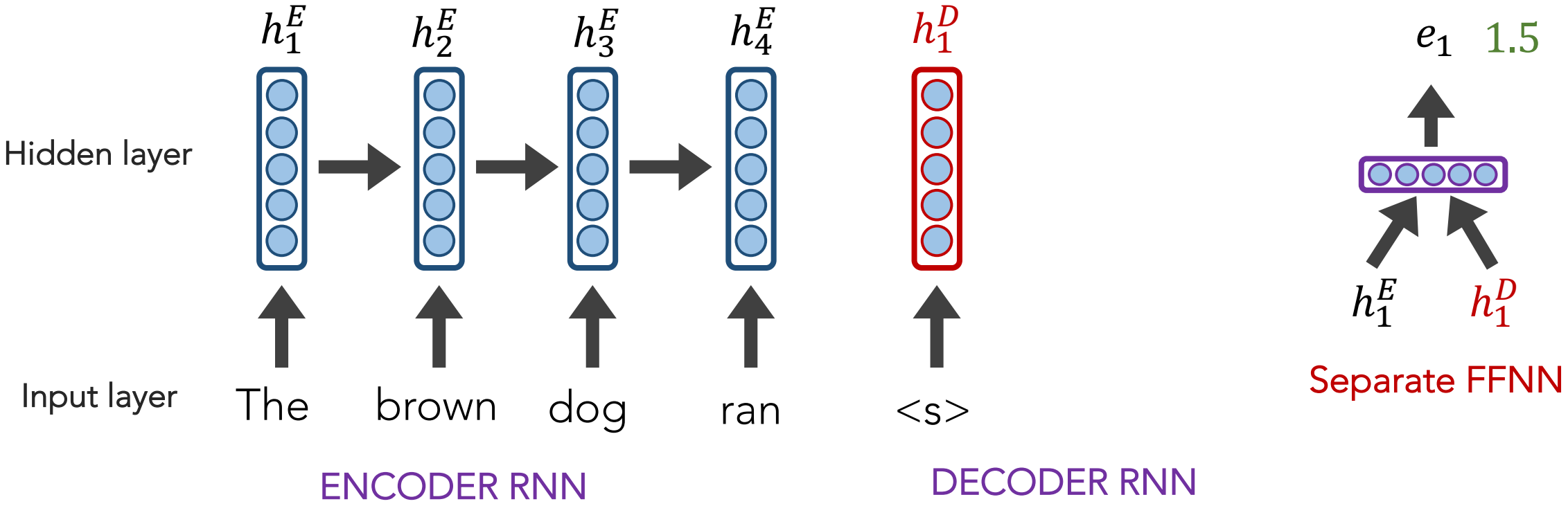
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

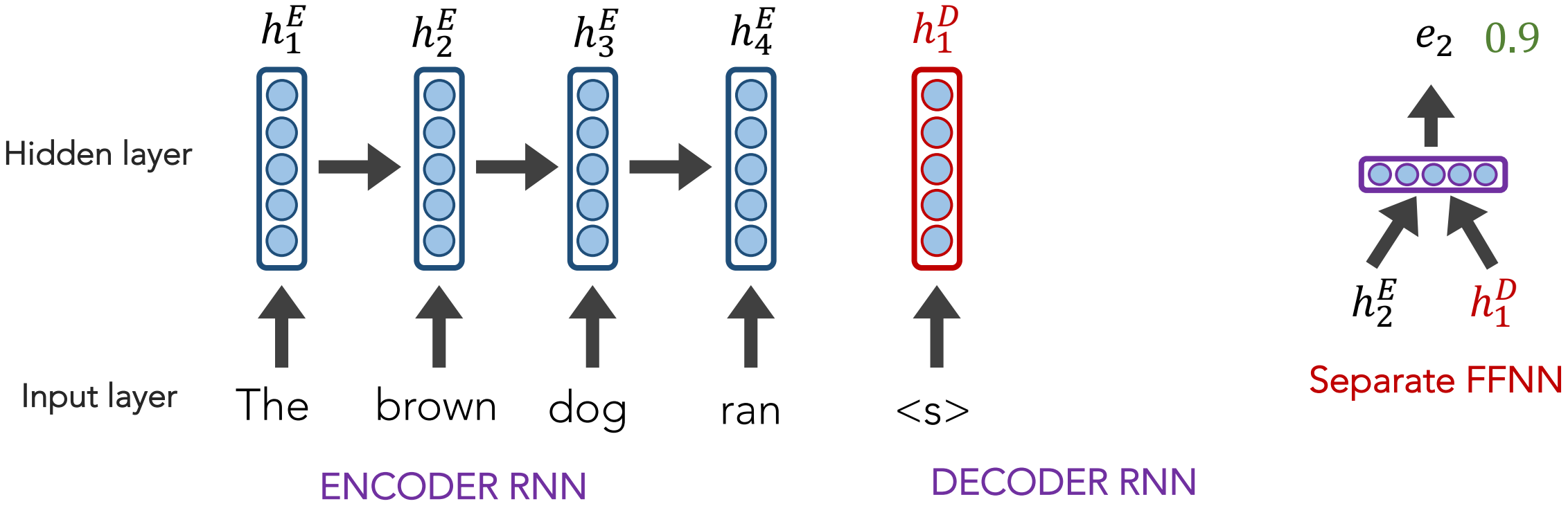
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

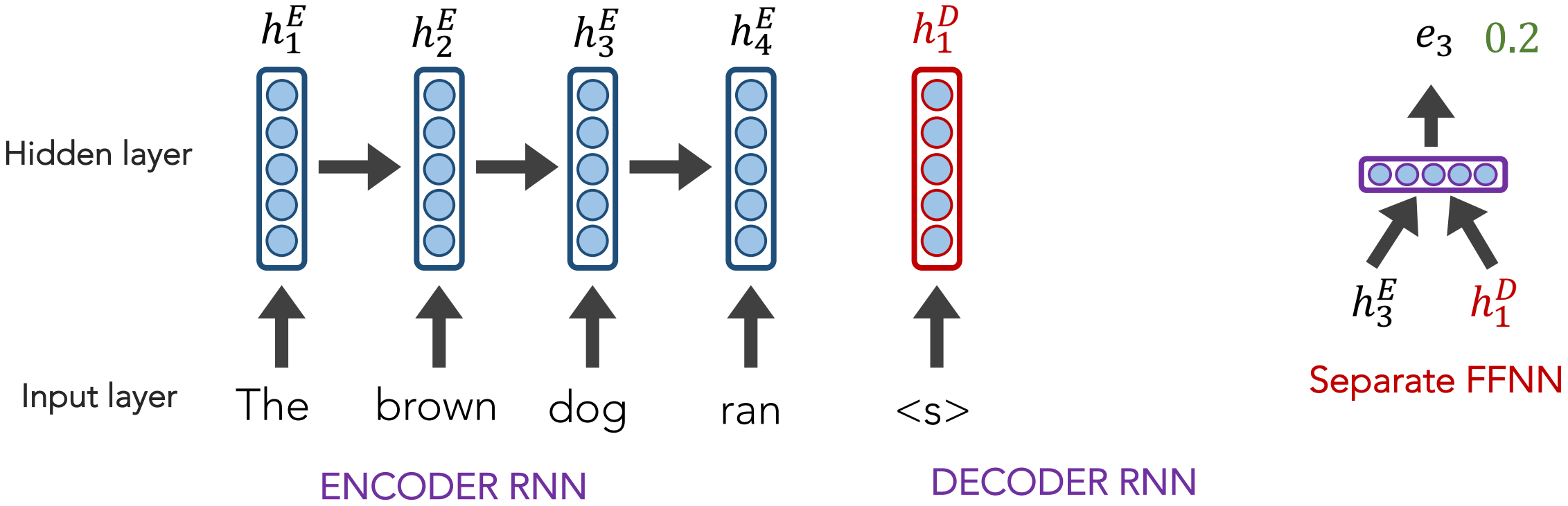
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

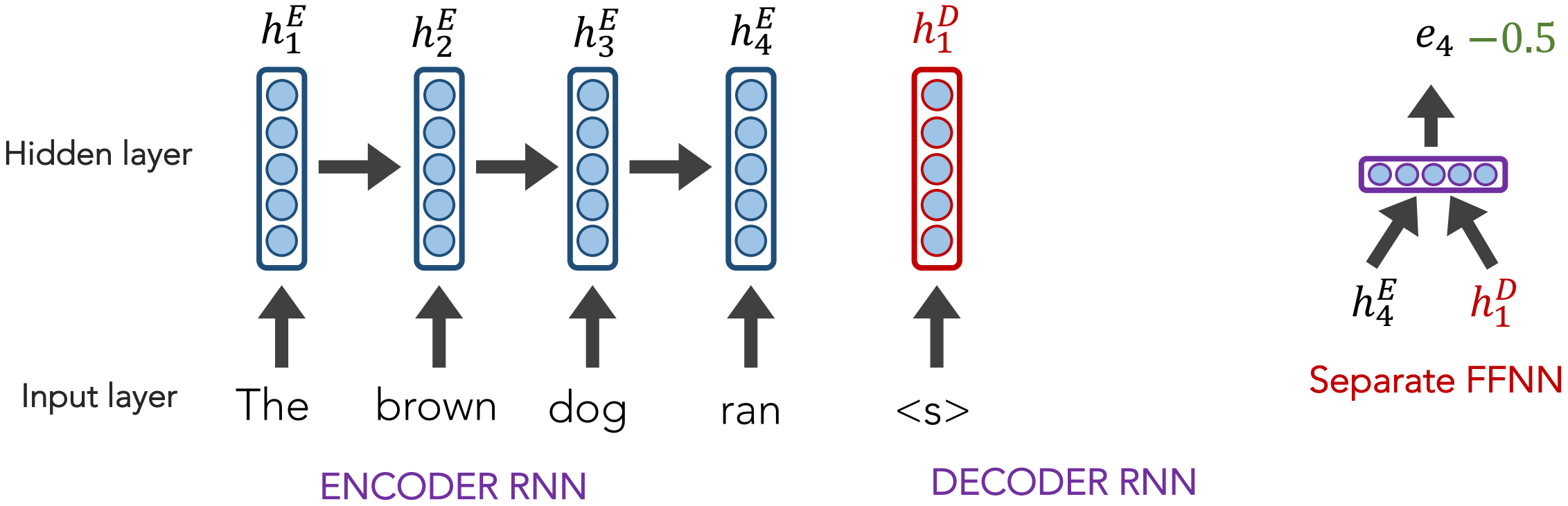
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

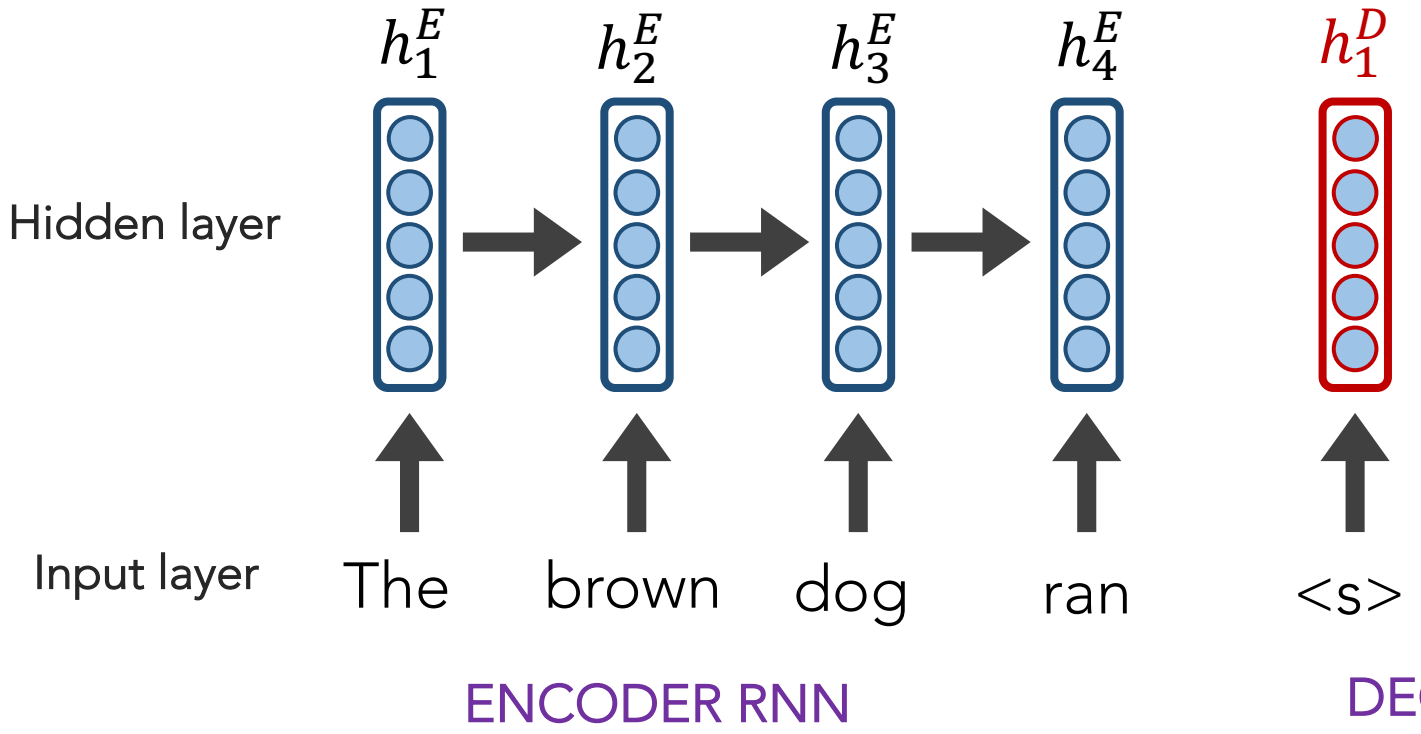
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



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!



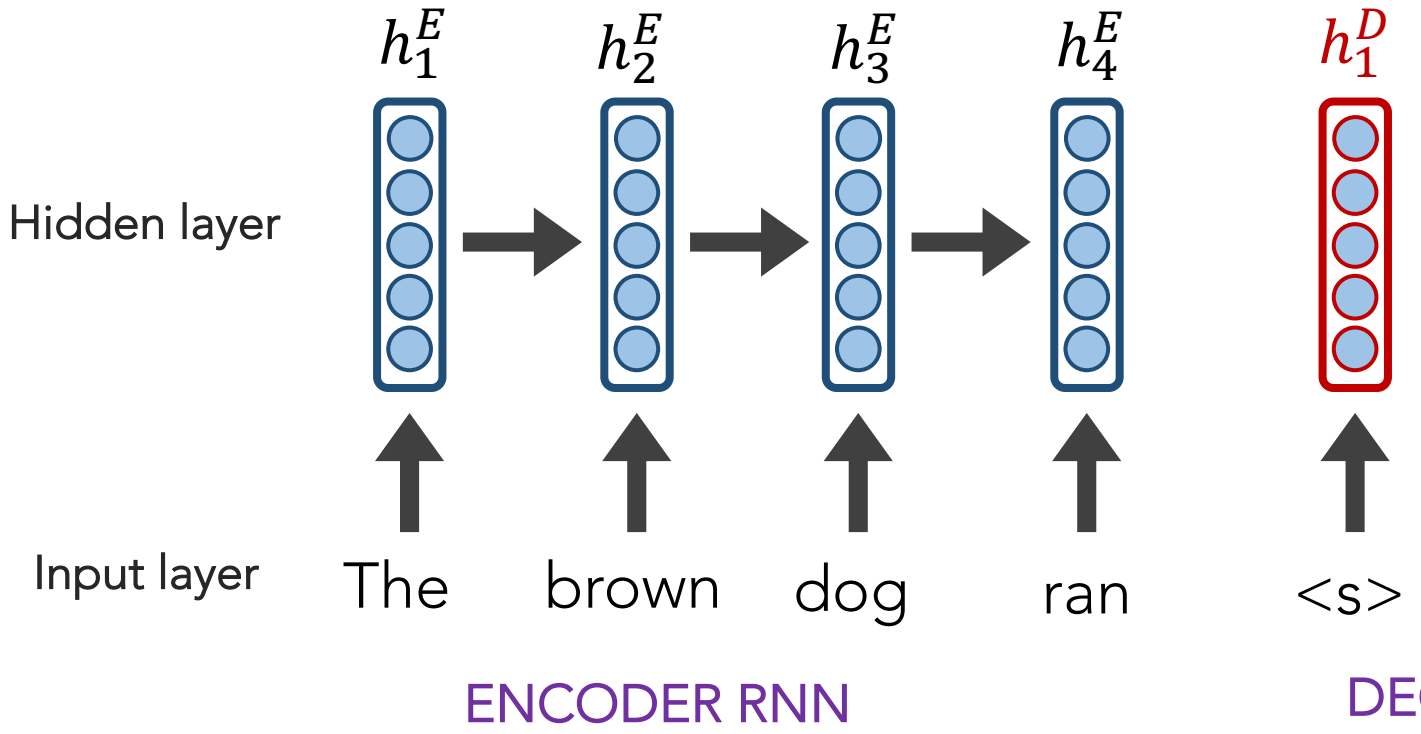
Attention (raw scores)

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

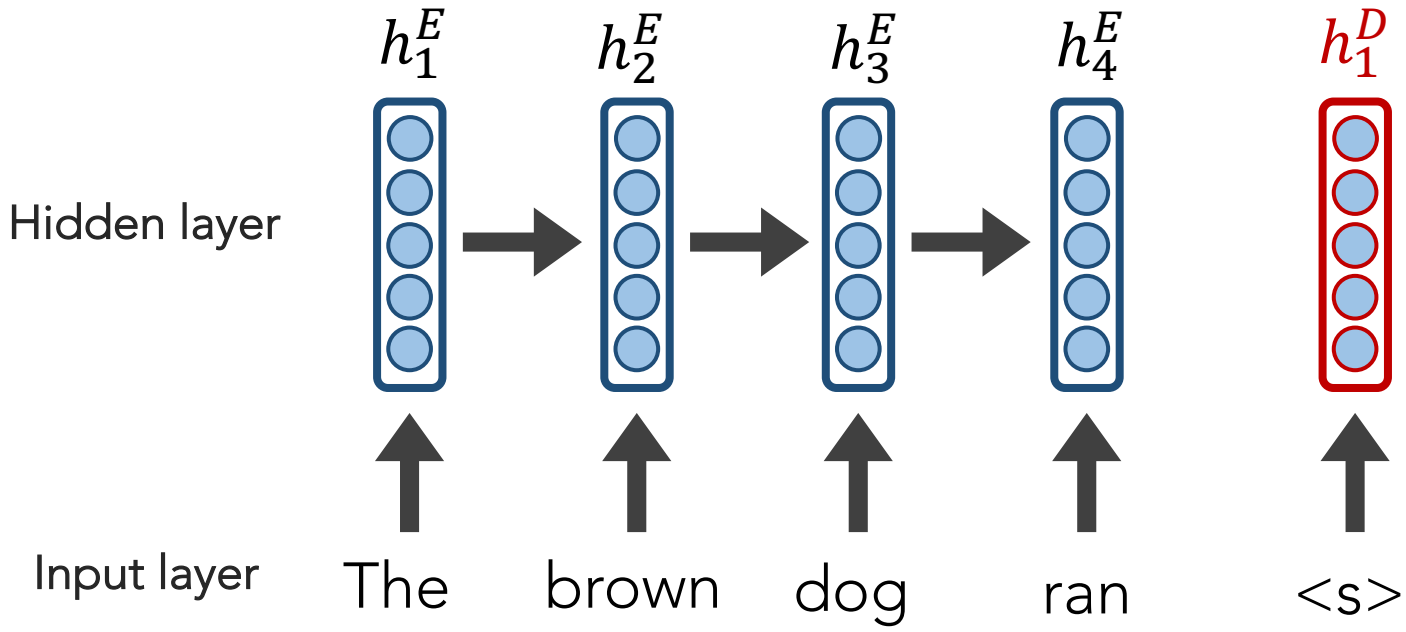
Attention (softmax'd)

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

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)

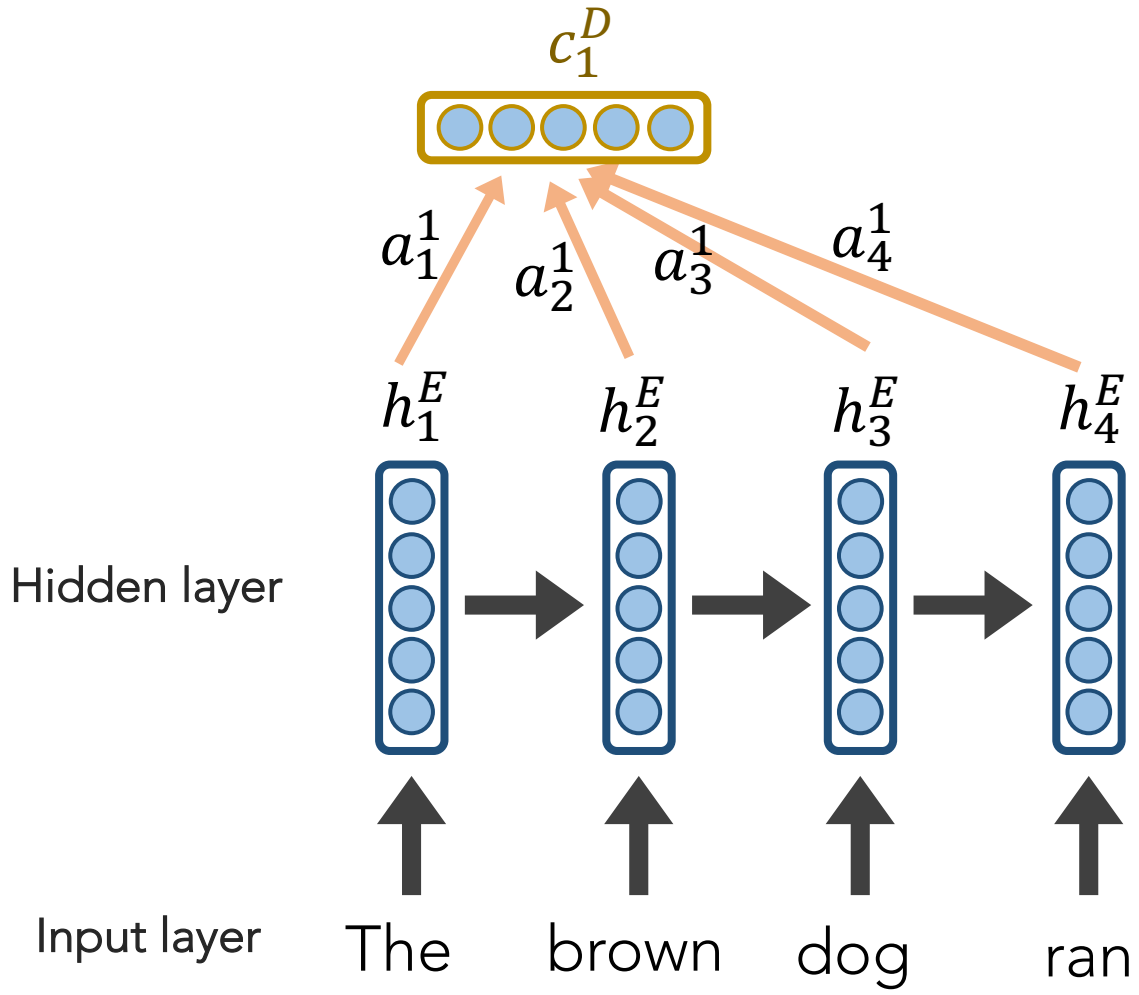
$a_1^1 = 0.51$

$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

seq2seq + Attention



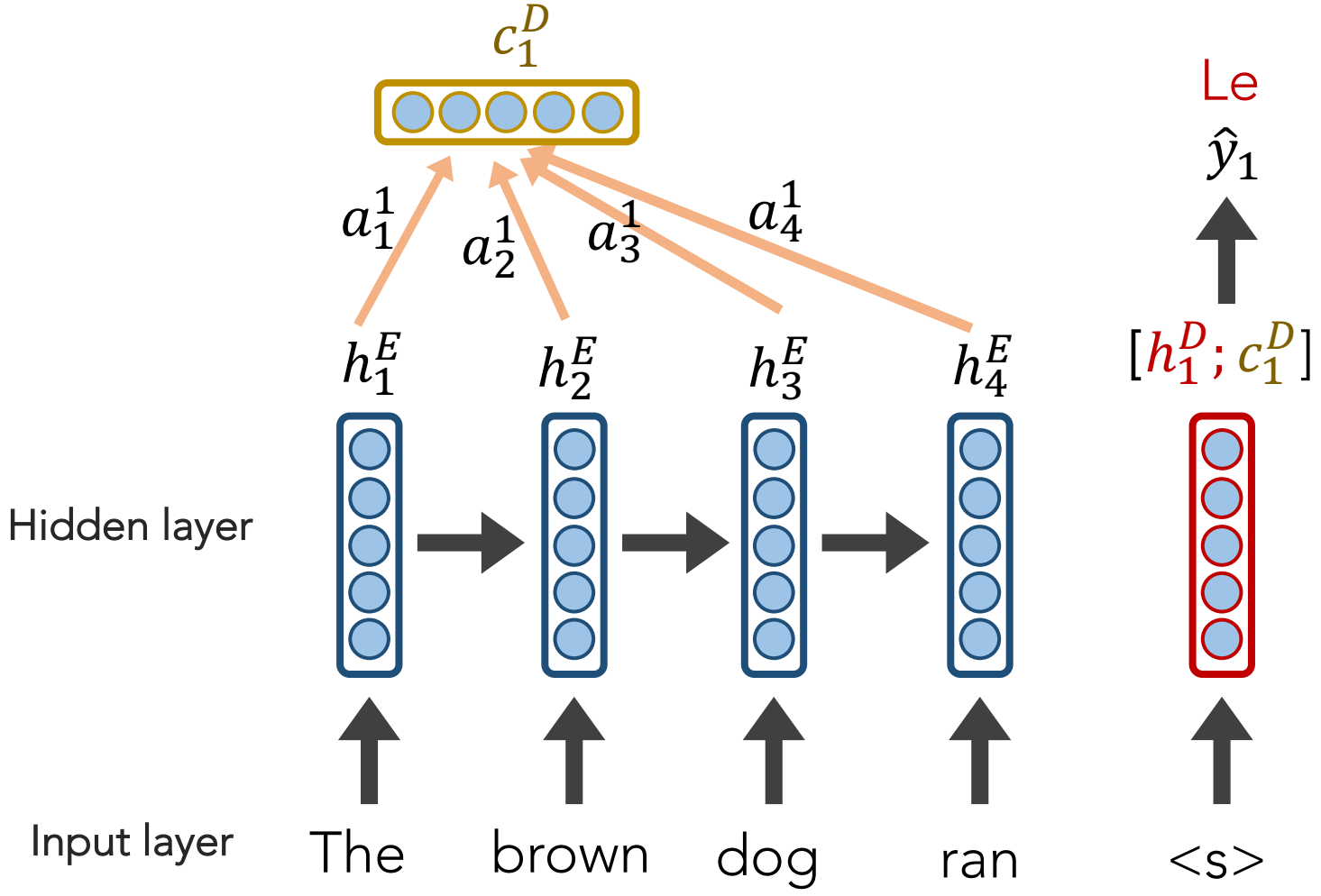
We multiply each encoder's hidden layer by its a_i^1 attention weights to create a context vector c_1^D

Attention (softmax'd)

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

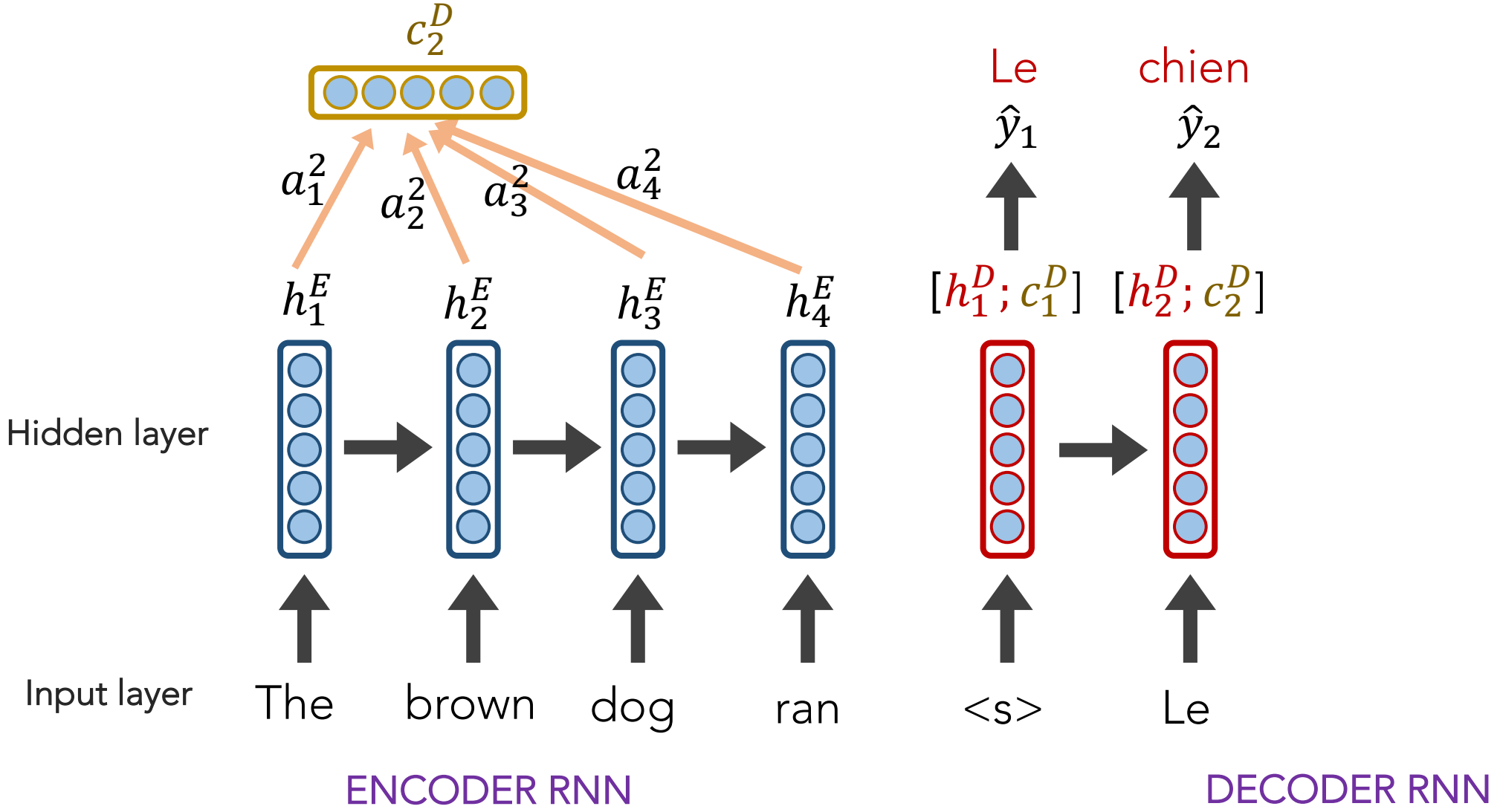
seq2seq + Attention

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



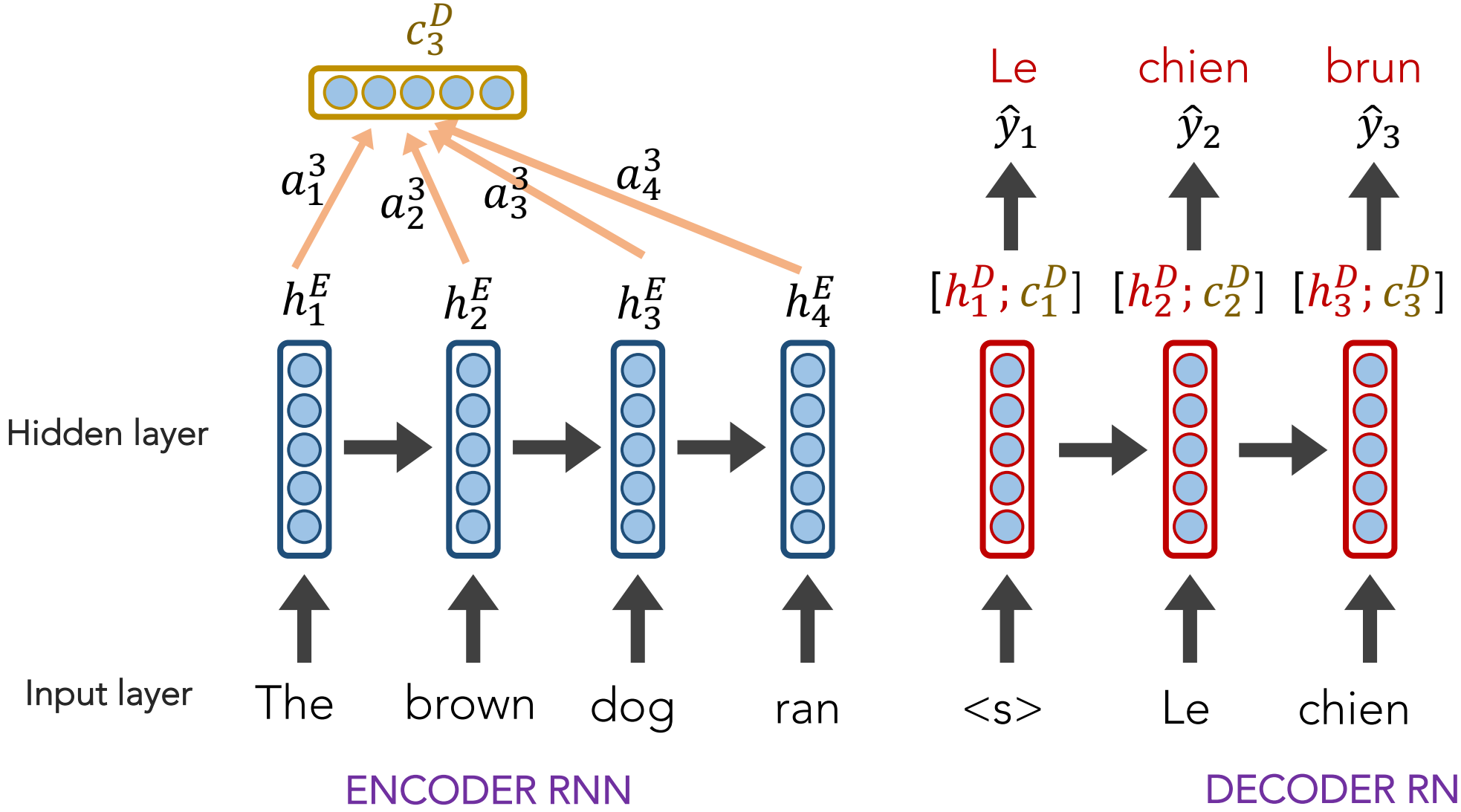
seq2seq + Attention

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



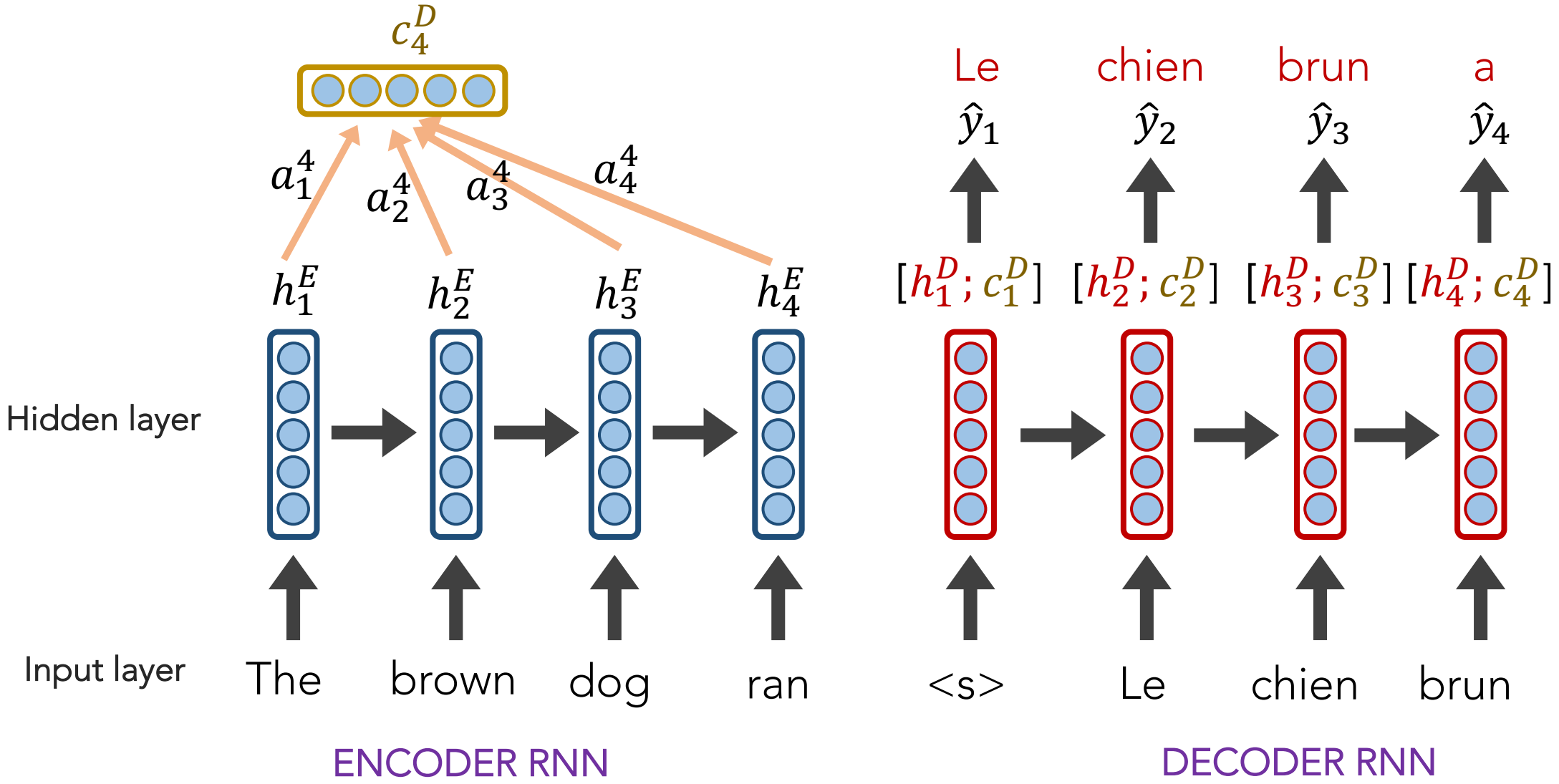
seq2seq + Attention

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



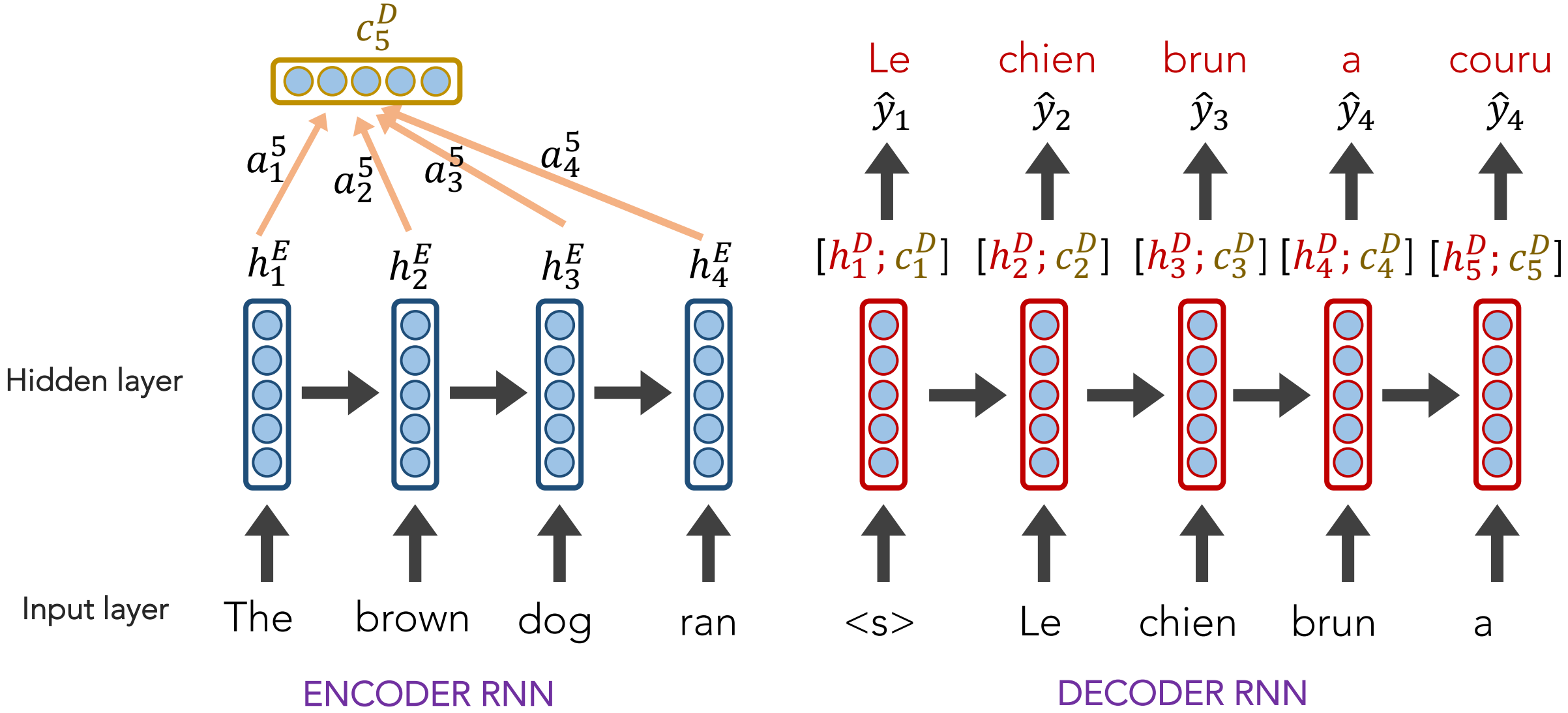
seq2seq + Attention

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

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

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

(softmax)

Attention scores

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

"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

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

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

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

"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

Popular Attention Scoring functions:

Dot-product

$h_t^T \times s_k$

$\text{score}(h_t, s_k) = h_t^T s_k$

Bilinear

$h_t^T \times [W] \times s_k$

$\text{score}(h_t, s_k) = h_t^T W s_k$

Multi-Layer Perceptron

$w_2^T \times \tanh\left[W_1 \times \begin{bmatrix} h_t \\ s_k \end{bmatrix}\right]$

$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

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

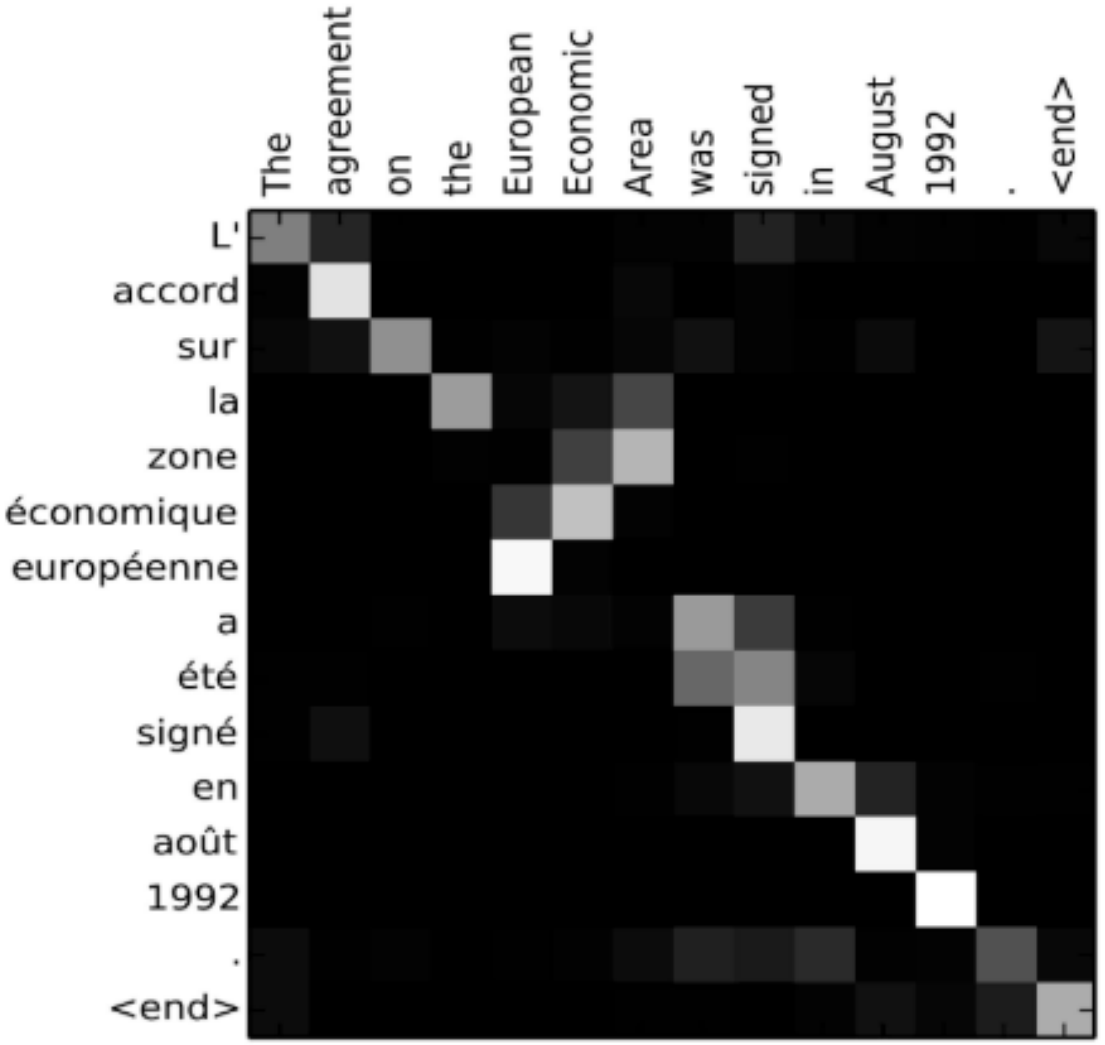


Image source: Fig 3 in [Bahdanau et al., 2015]

Takeaway:

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

Attention is powerful; allows us to conditionally weight our focus

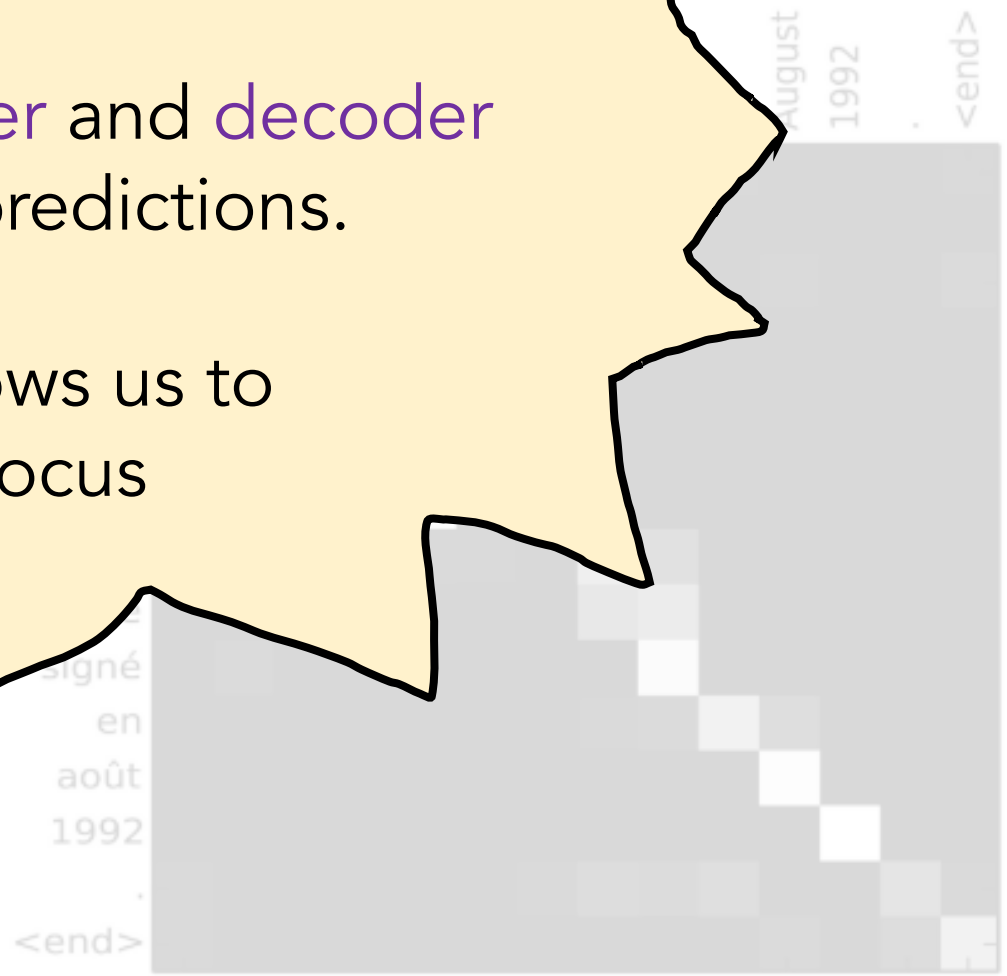


Image source: Fig 3 in Bahdanau et al., 2015

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

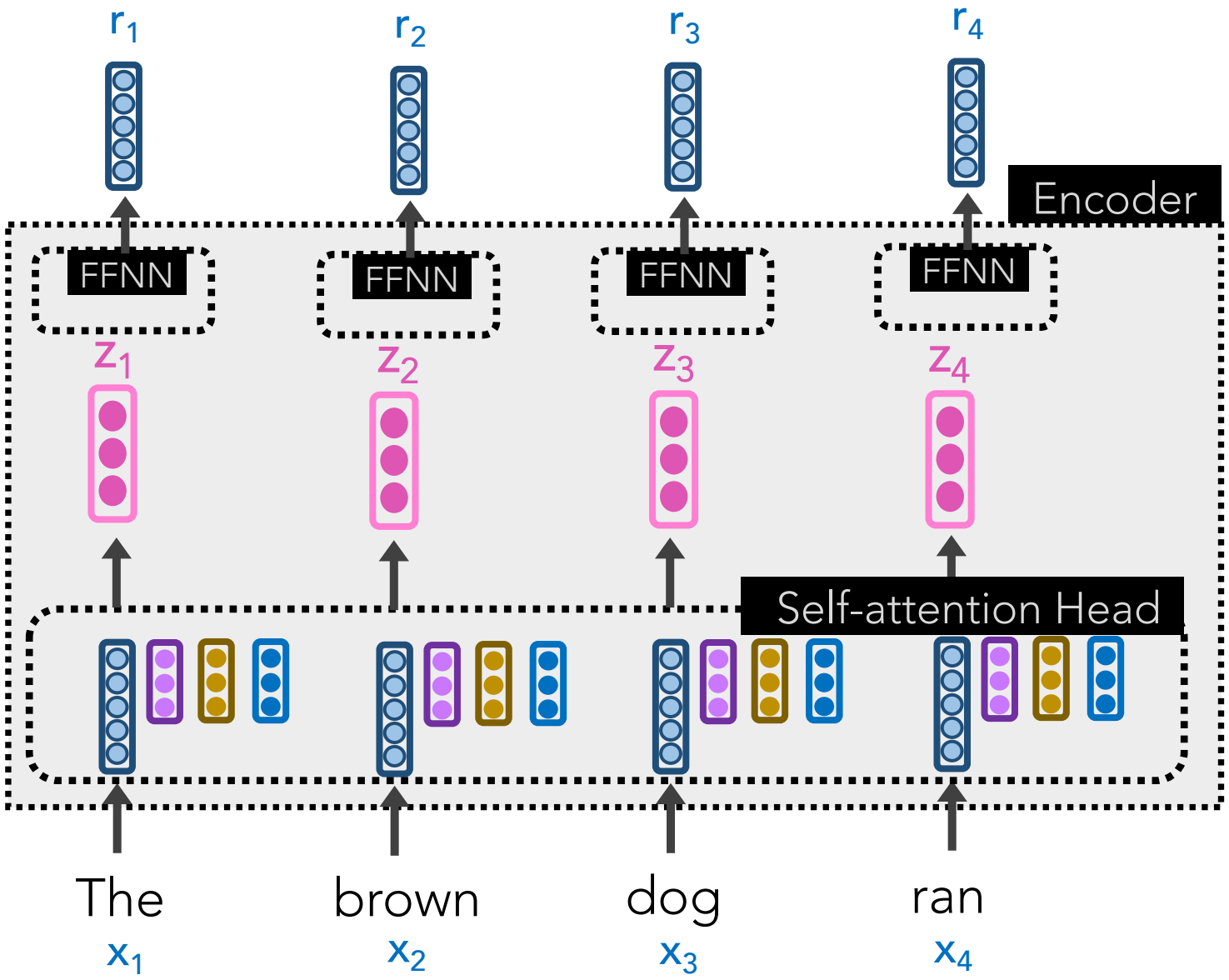
Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Outline

- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Transformer Encoder



Transformer Encoder uses attention on itself (self-attention) to create very rich embeddings which can be used for any task.

BERT is a *Bidirectional* **Transformer Encoder.** You can attach a final layer that performs whatever task you're interested in (e.g., Yelp reviews).

Its results are unbelievably good.

BERT (a Transformer variant)

BERT is trained on a lot of text data:

- BooksCorpus (800M words)
- English Wikipedia (2.5B words)

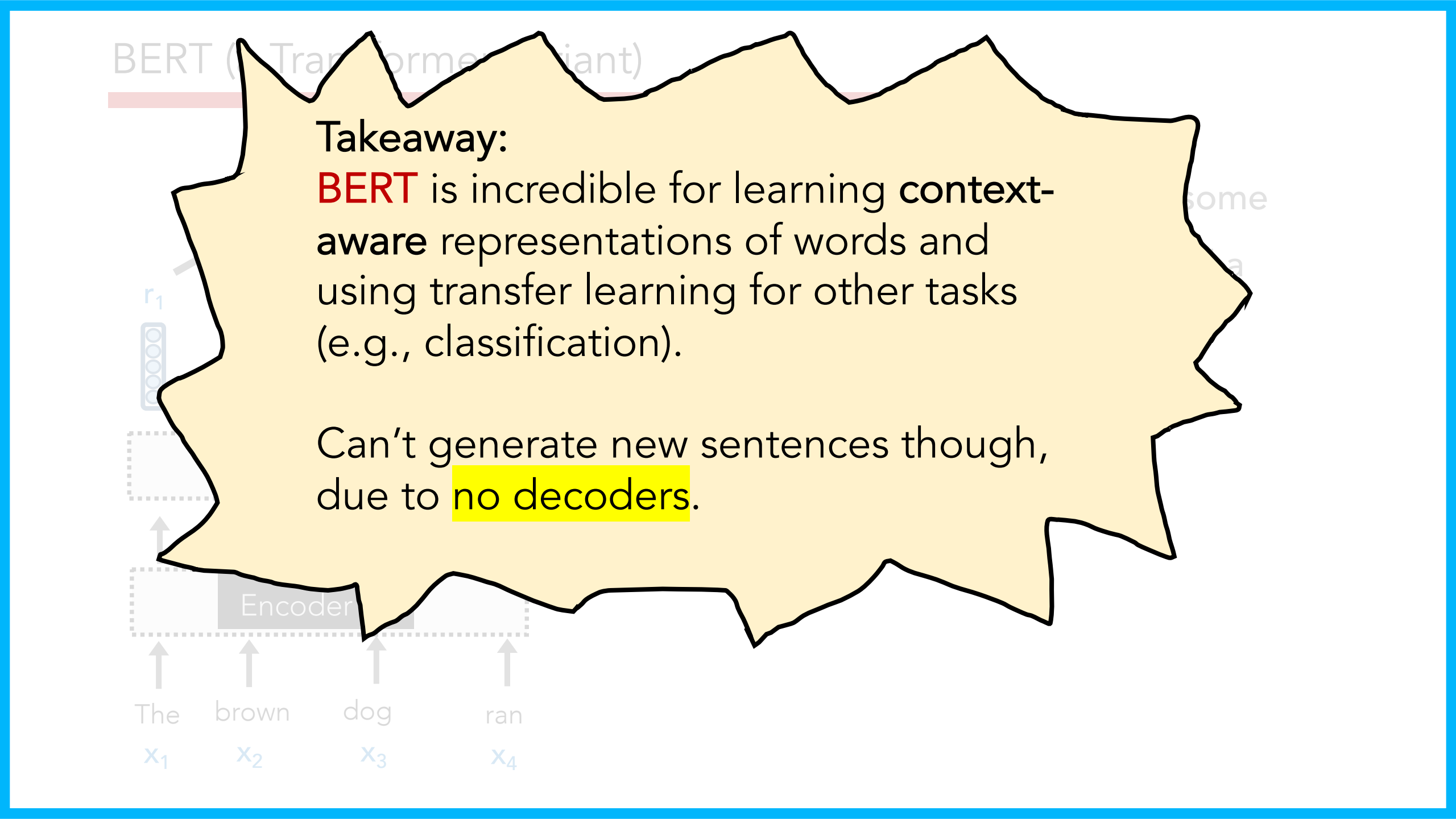
Yay, for transfer learning!

BERT-Base model has 12 transformer blocks, 12 attention heads,

110M parameters!

BERT-Large model has 24 transformer blocks, 16 attention heads,

340M parameters!


BERT (a Transformer variant)
Takeaway:
BERT is incredible for learning context-aware representations of words and using transfer learning for other tasks (e.g., classification).
Can't generate new sentences though, due to no decoders.
Encoder
The
brown
dog
ran

Transformer

What if we want to generate a new output sequence?

GPT-2 model to the rescue!

Generative Pre-trained Transformer 2

GPT-2 (a Transformer variant)

- GPT-2 uses only Transformer Decoders (no Encoders) to generate new sequences
- As it processes each word/token, it cleverly masks the “future” words and conditions itself on the previous words
- Can generate text from scratch or from a starting sequence.
- Easy to fine-tune on your own dataset (language)

GPT-2

- GPT-2 u
new
- As it p
word
- Can g
- Easy
- GPT-3 is an eve bigger version of GP but isn't open-source

Takeaway:
GPT-2 is astounding for generating realistic-looking new text

Can fine-tune toward other tasks, too.

GPT-2 (a Transformer variant)

GPT-2 is:

- trained on 40GB of text data (8M webpages)!
- 1.5B parameters

GPT-3 is an even bigger version (175B parameters) of GPT-2, but isn't open-source

Yay, for transfer learning!

QUESTIONS?