Attention I

AC295

Pavlos Protopapas
Institute for Applied Computational Science, Harvard

Announcements

- Vote!
- Submit your reading questions by Wed 10/28 noon on Ed.
- Exercise was due 10:15 am, next coming up today.
- Project Milestone 1 - due 10/29 - 10:15 AM

Outline

Seq2seq with attention

Transformers

Bert

Outline

Seq2seq with attention

Transformers

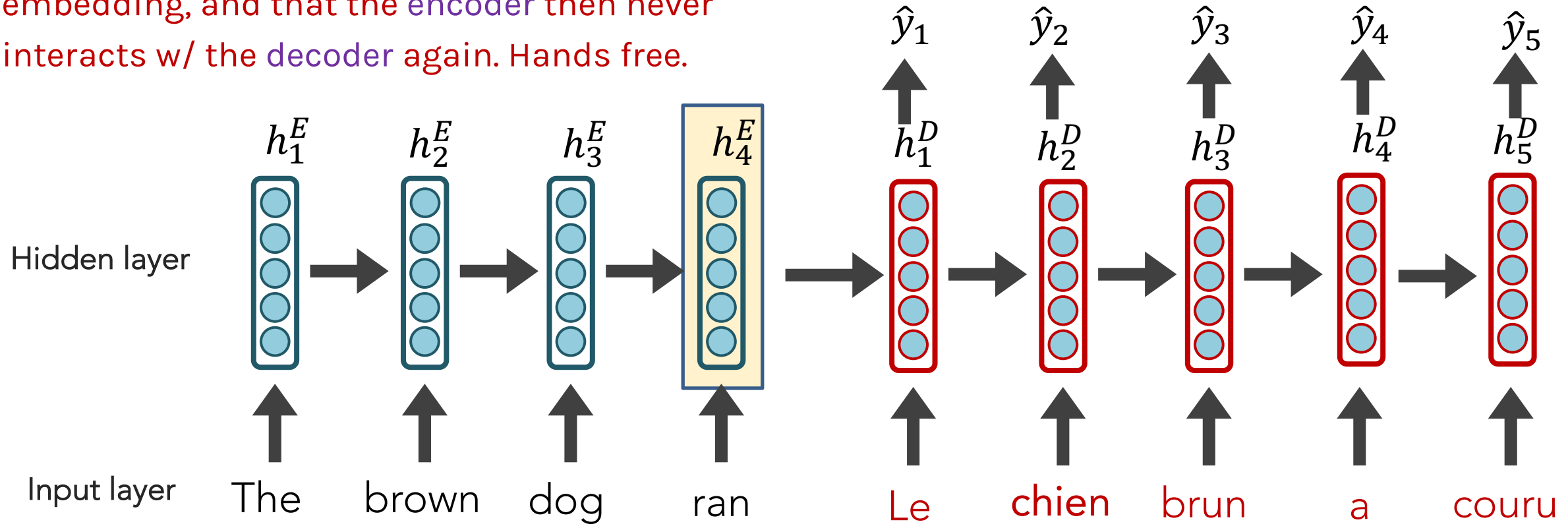
Bert

Sequence-to-Sequence (seq2seq)

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

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.

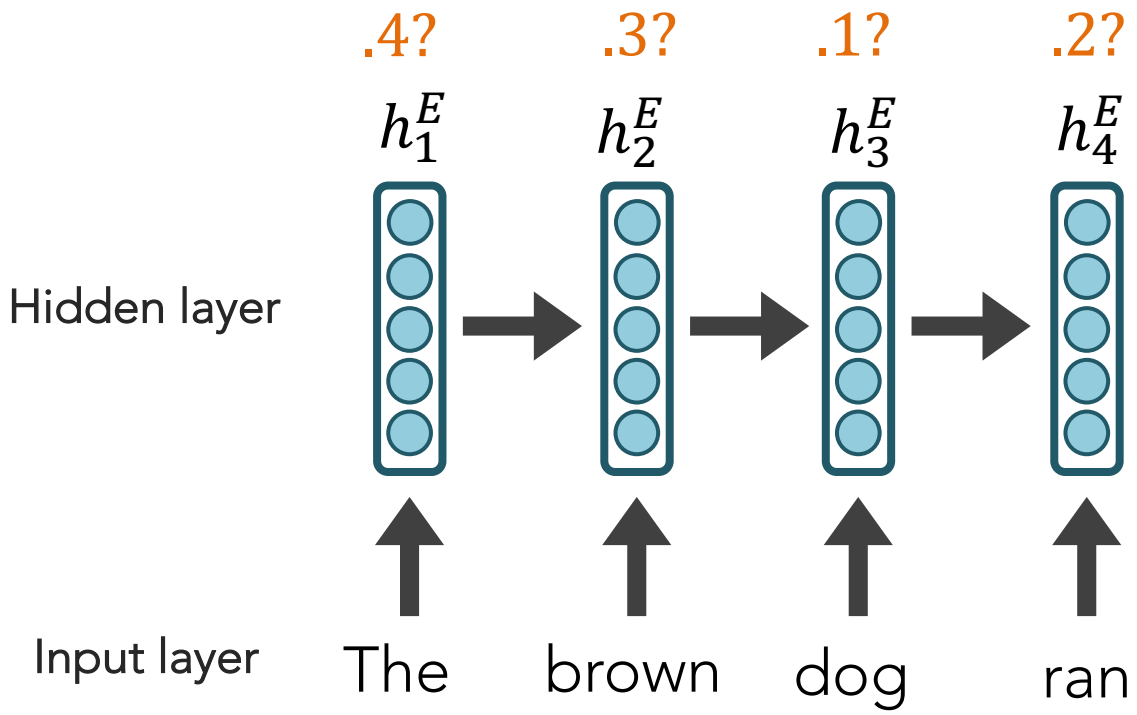


Sequence-to-Sequence (seq2seq)

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

seq2seq + Attention

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

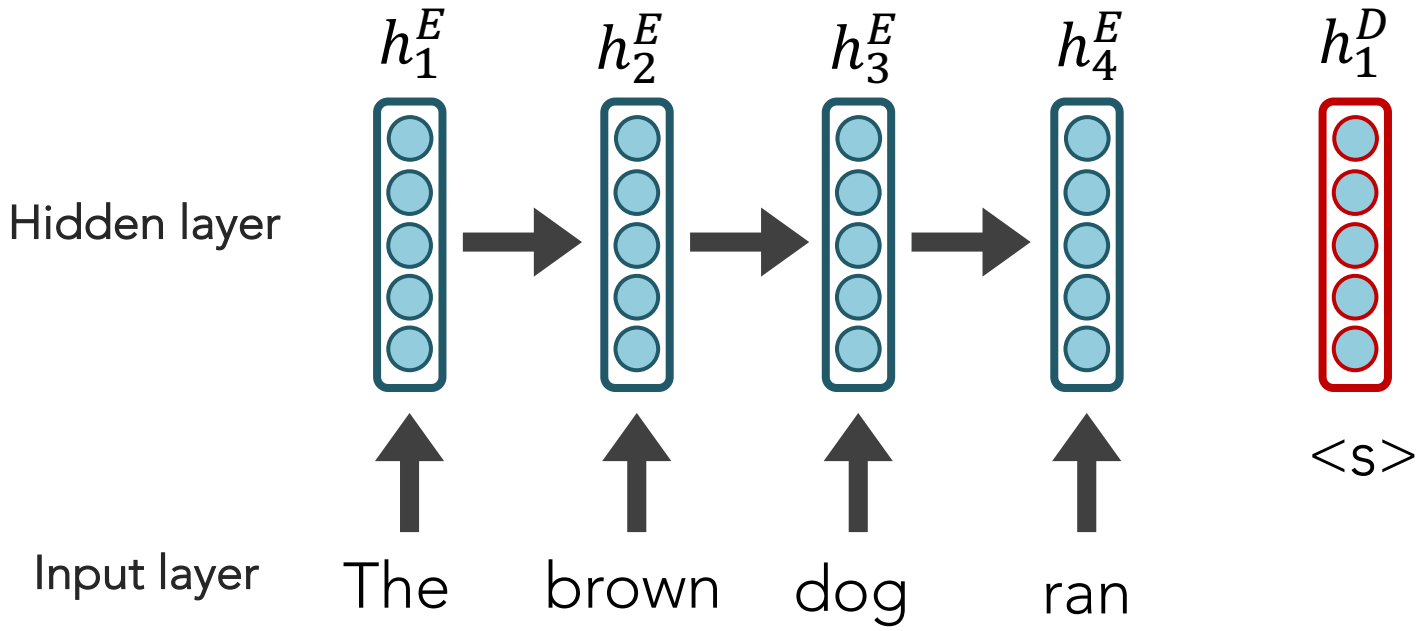


ENCODER RNN

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 previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!

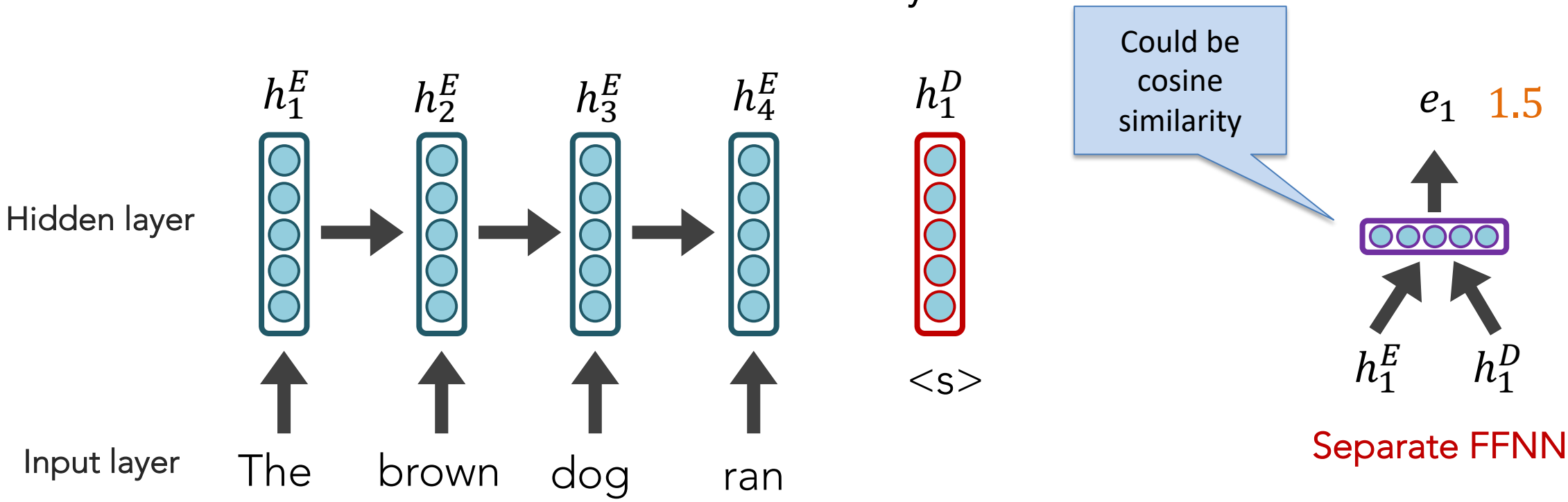


ENCODER RNN

seq2seq + Attention

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

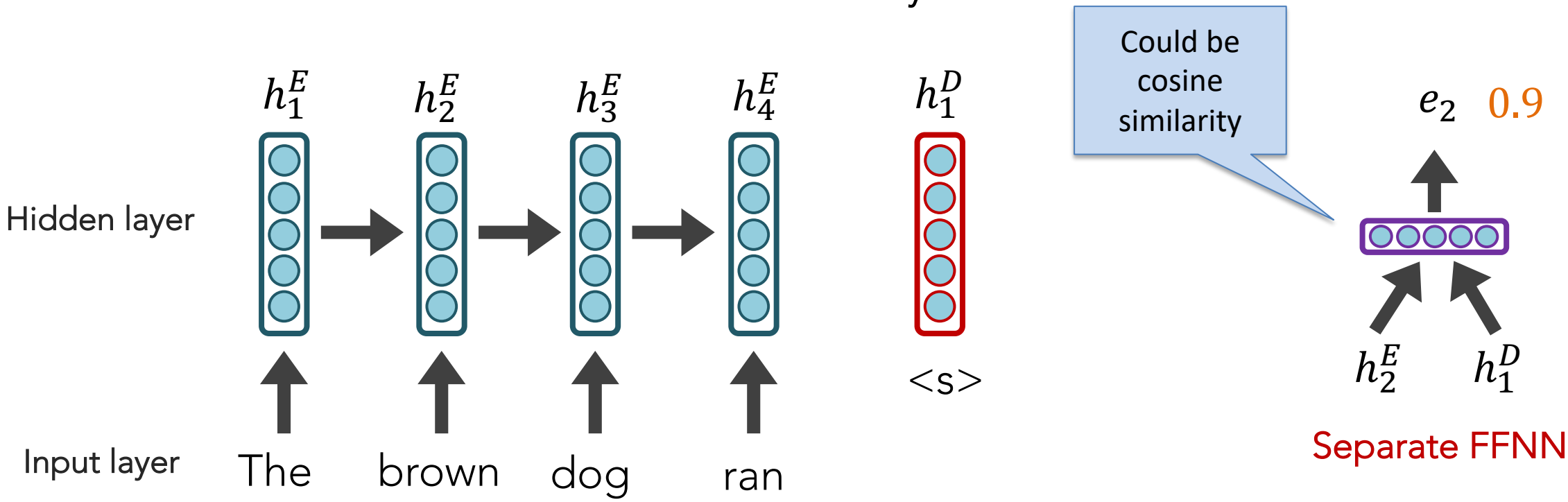
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.



seq2seq + Attention

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

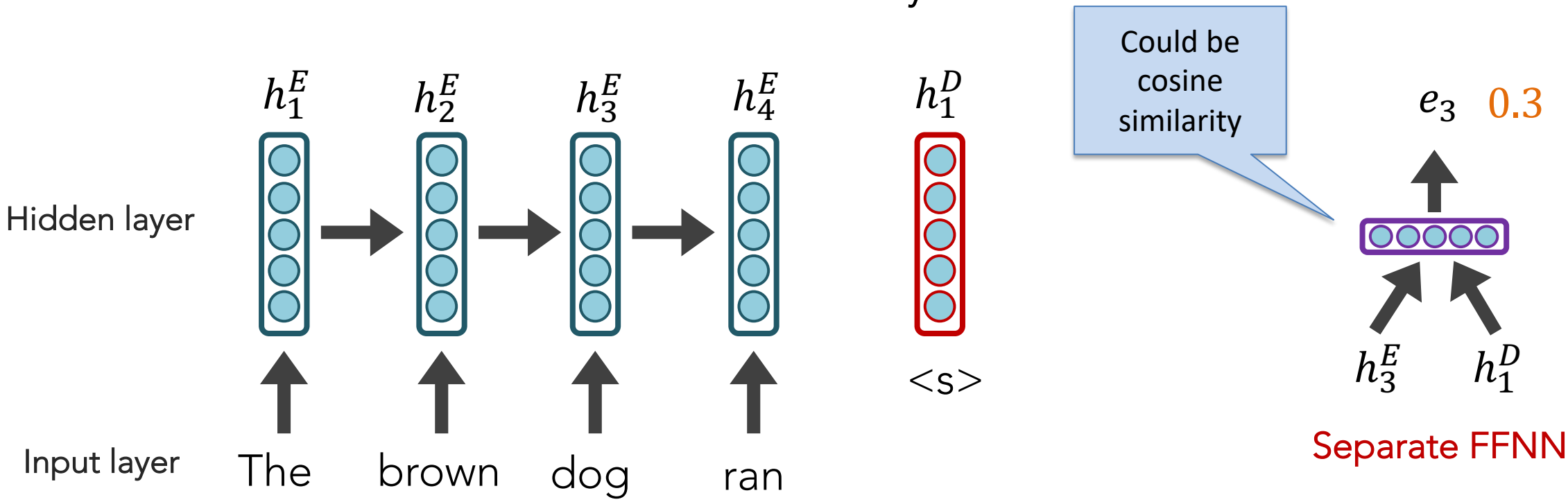
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.



seq2seq + Attention

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

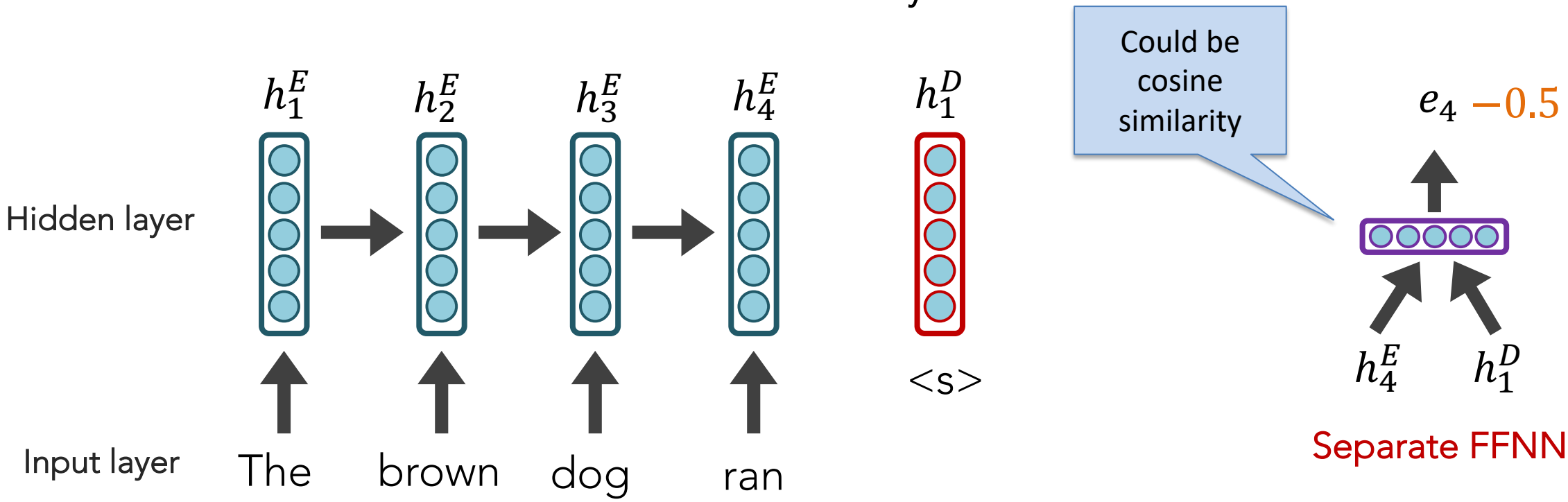
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.



seq2seq + Attention

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

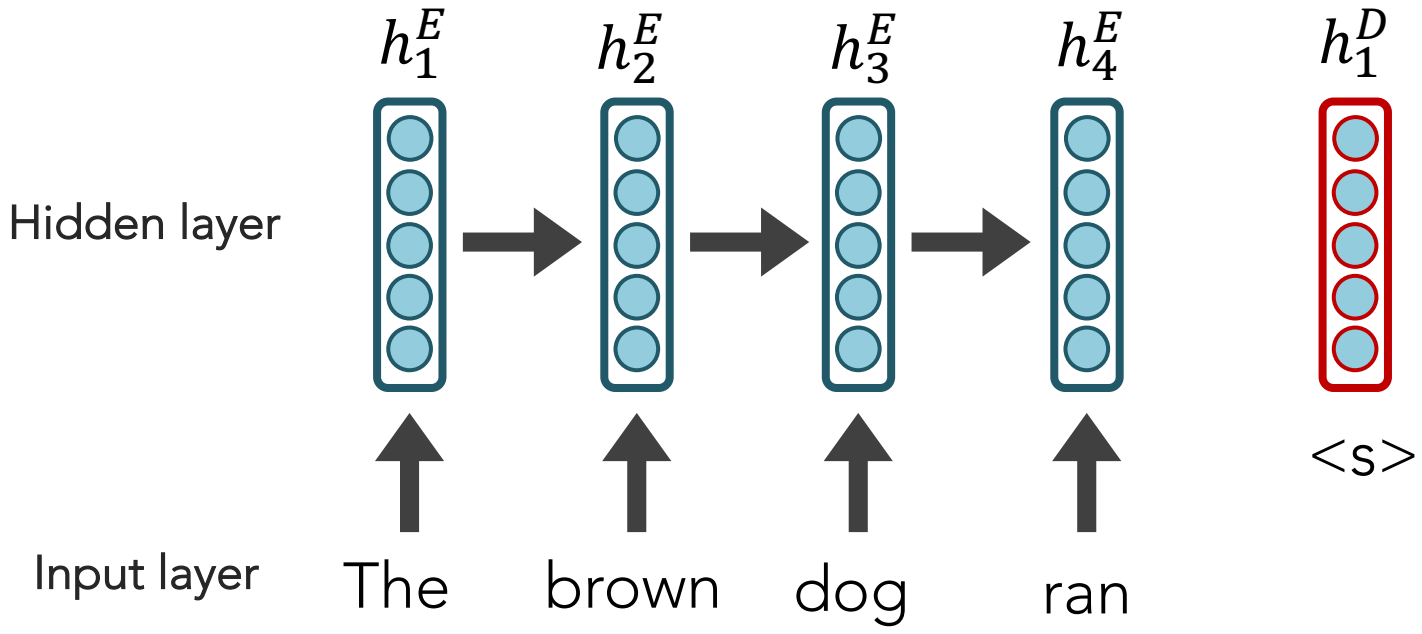
A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.



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 previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!



Attention (raw scores)

e_1 1.5

e_2 0.9

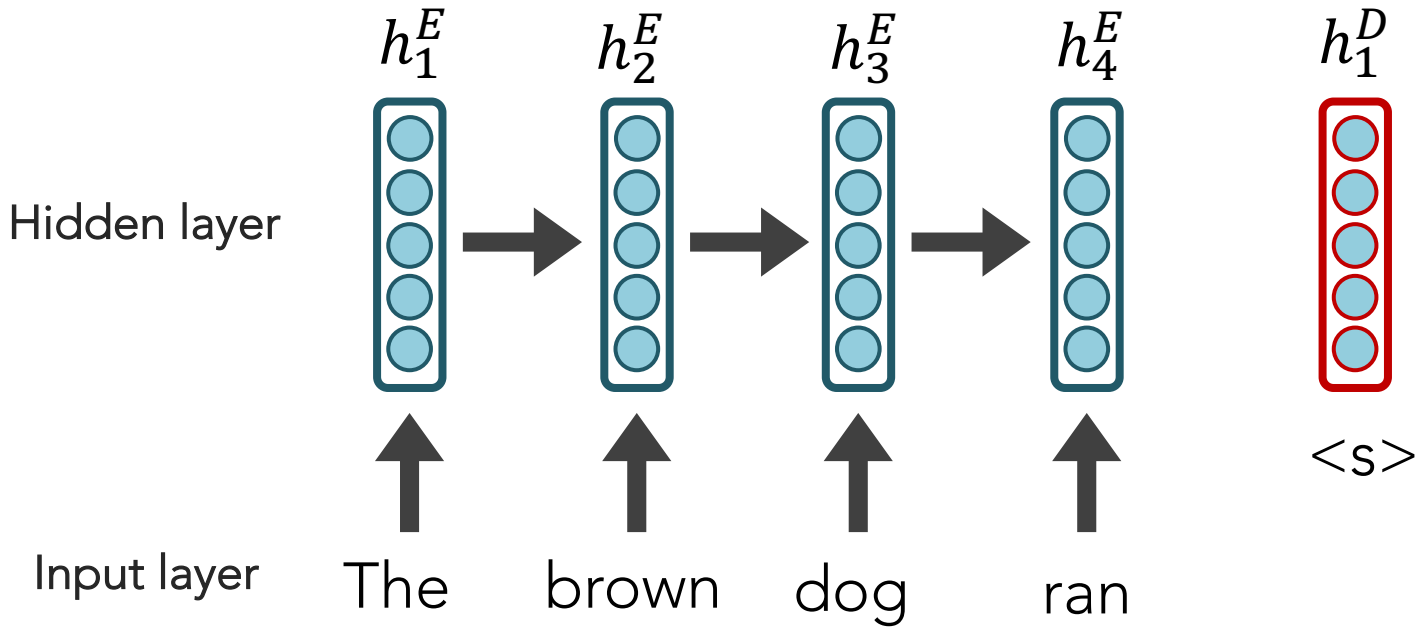
e_3 0.2

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 previous hidden state (our latest 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

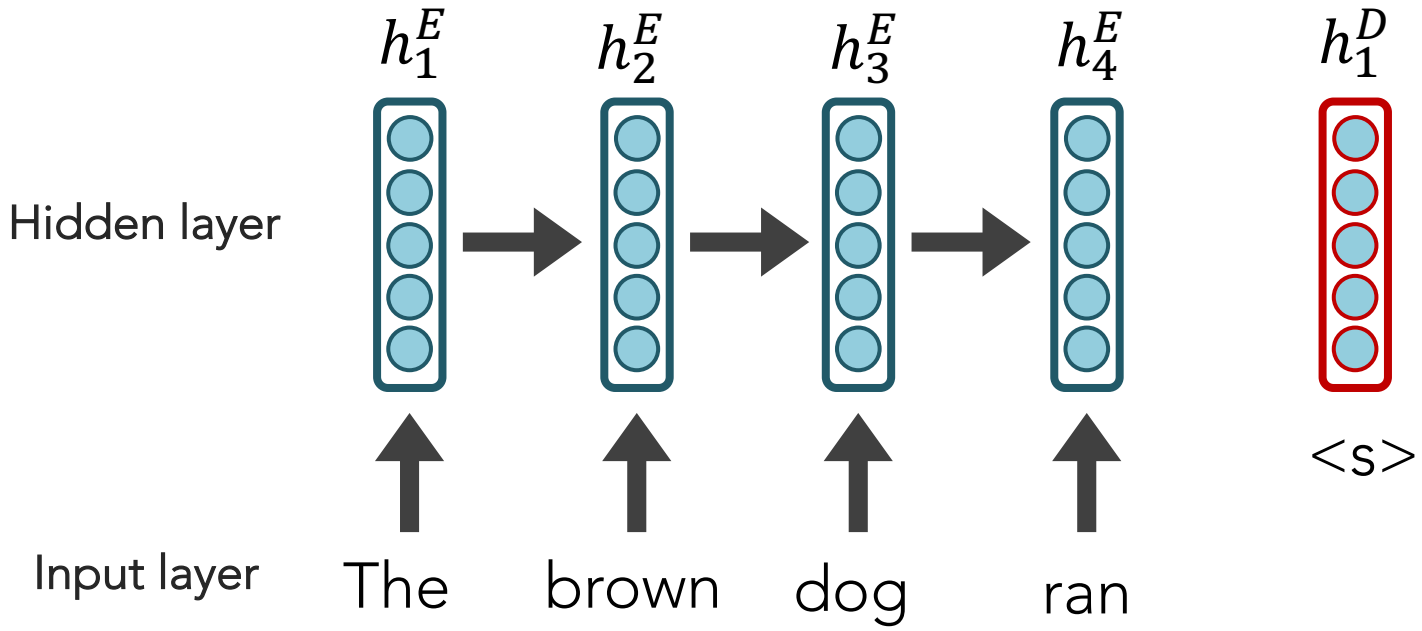
Attention (softmax'd)

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

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 previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!



ENCODER RNN

Attention (raw scores)

e_1 1.5

e_2 0.9

e_3 0.2

e_4 −0.5

Attention (softmax'd)

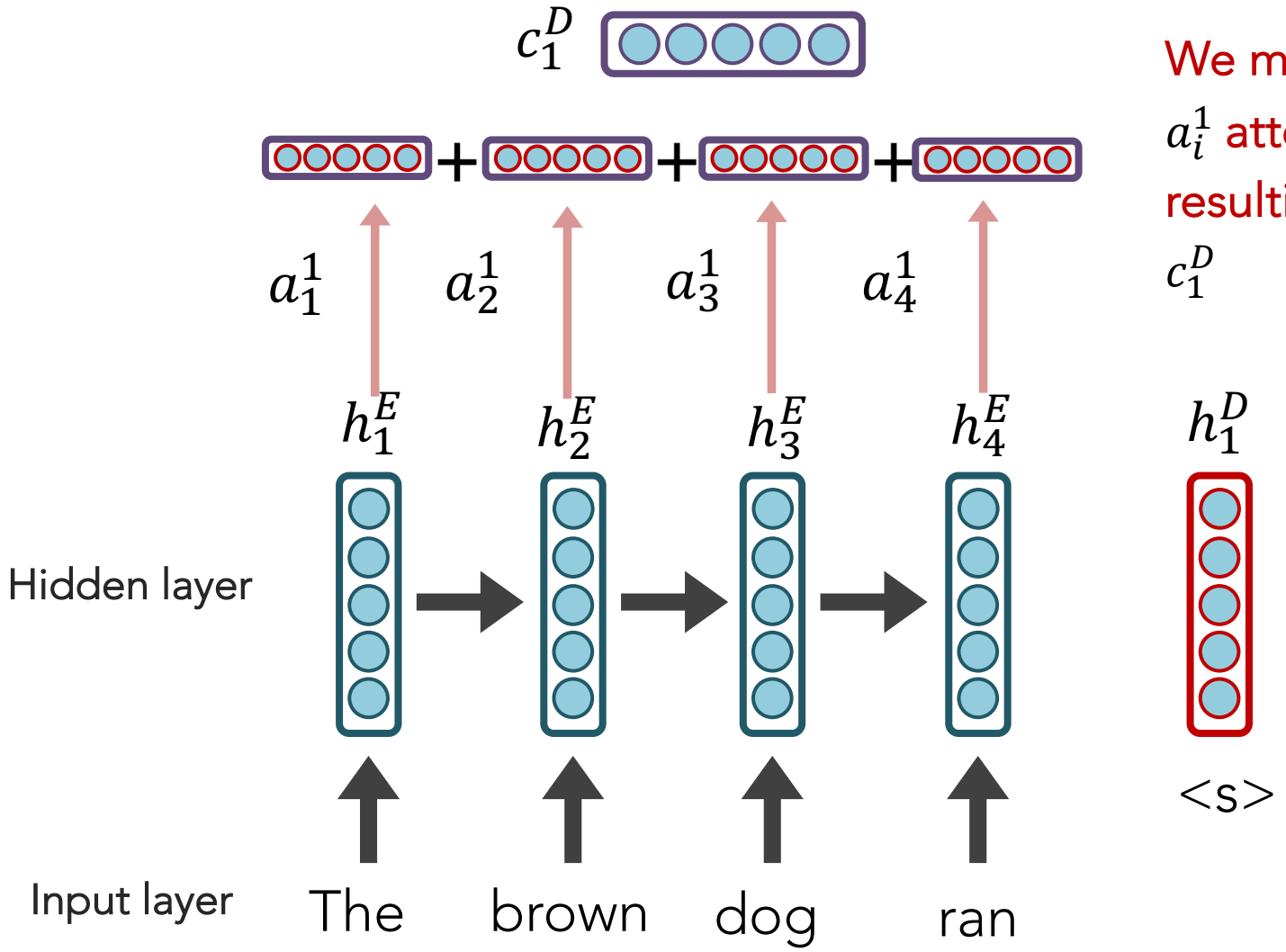
$a_1^1 = 0.51$

$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

seq2seq + Attention



We multiply each hidden layer by its a_i^1 attention weights and then add the resulting vectors to create a context vector c_1^D

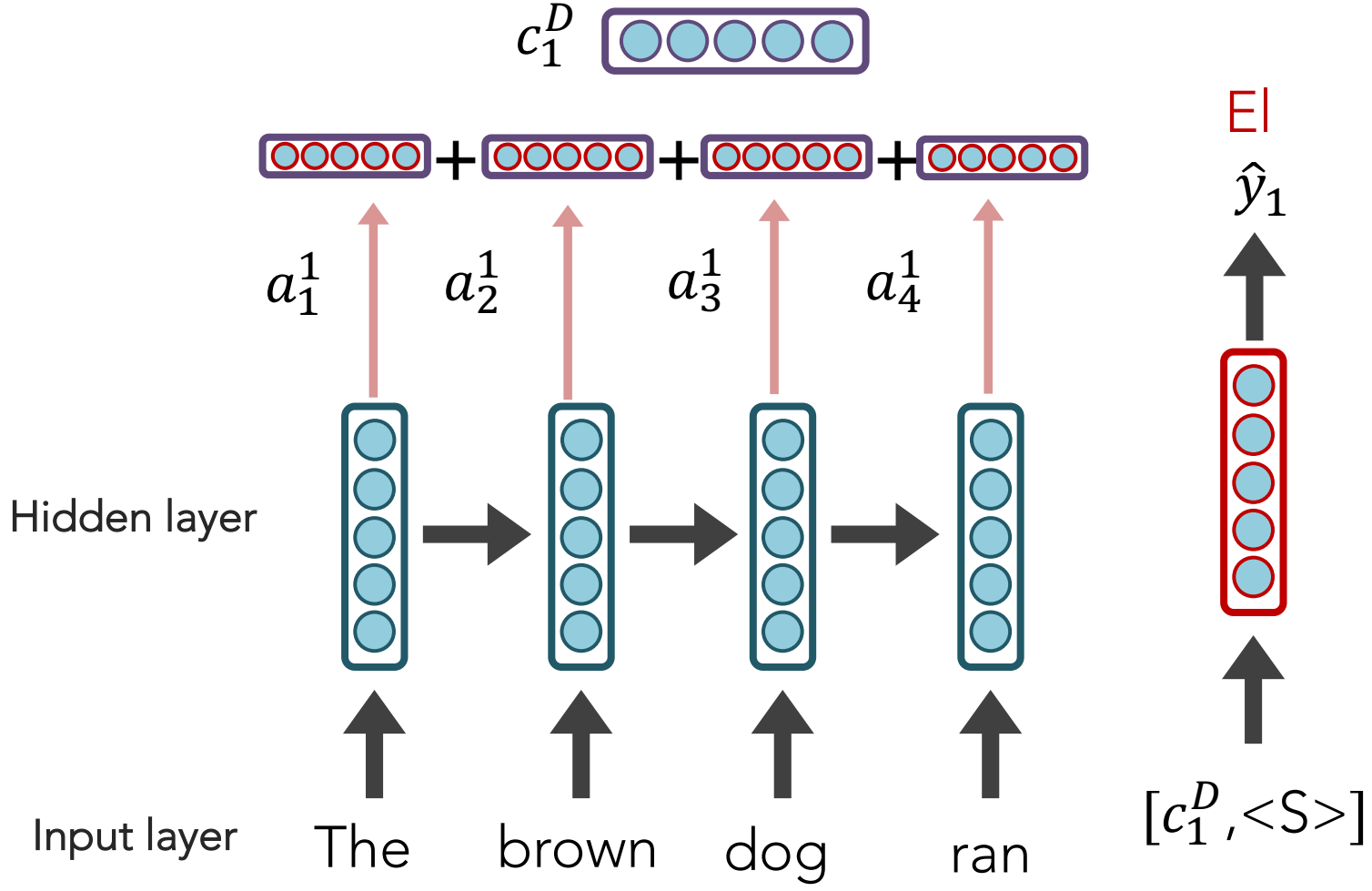
Attention (softmax'd)

$$a_1^1 = 0.51$$
$$a_2^1 = 0.28$$
$$a_3^1 = 0.14$$
$$a_4^1 = 0.07$$

ENCODER RNN

seq2seq + Attention

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

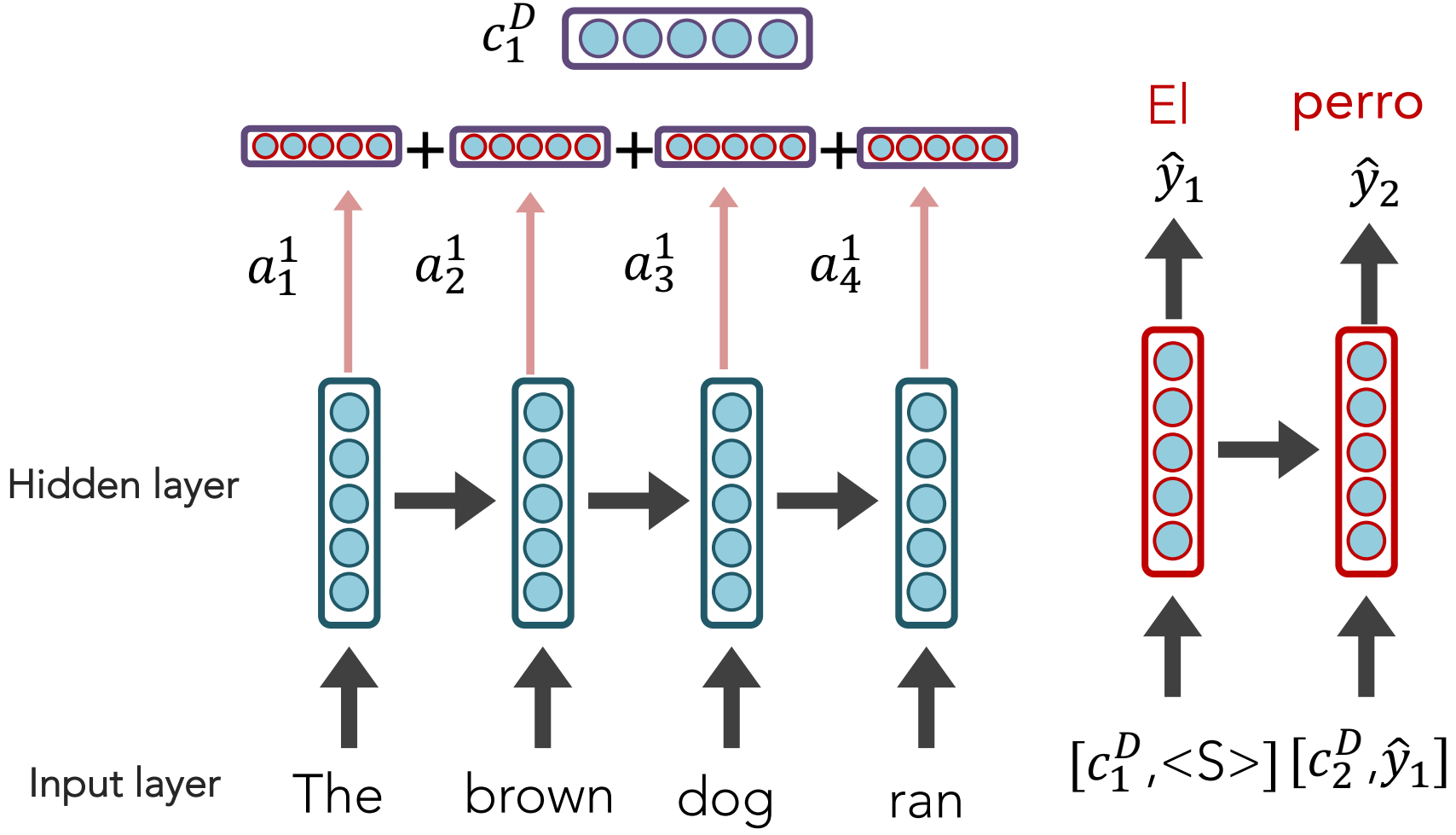


ENCODER RNN

DECODER RNN

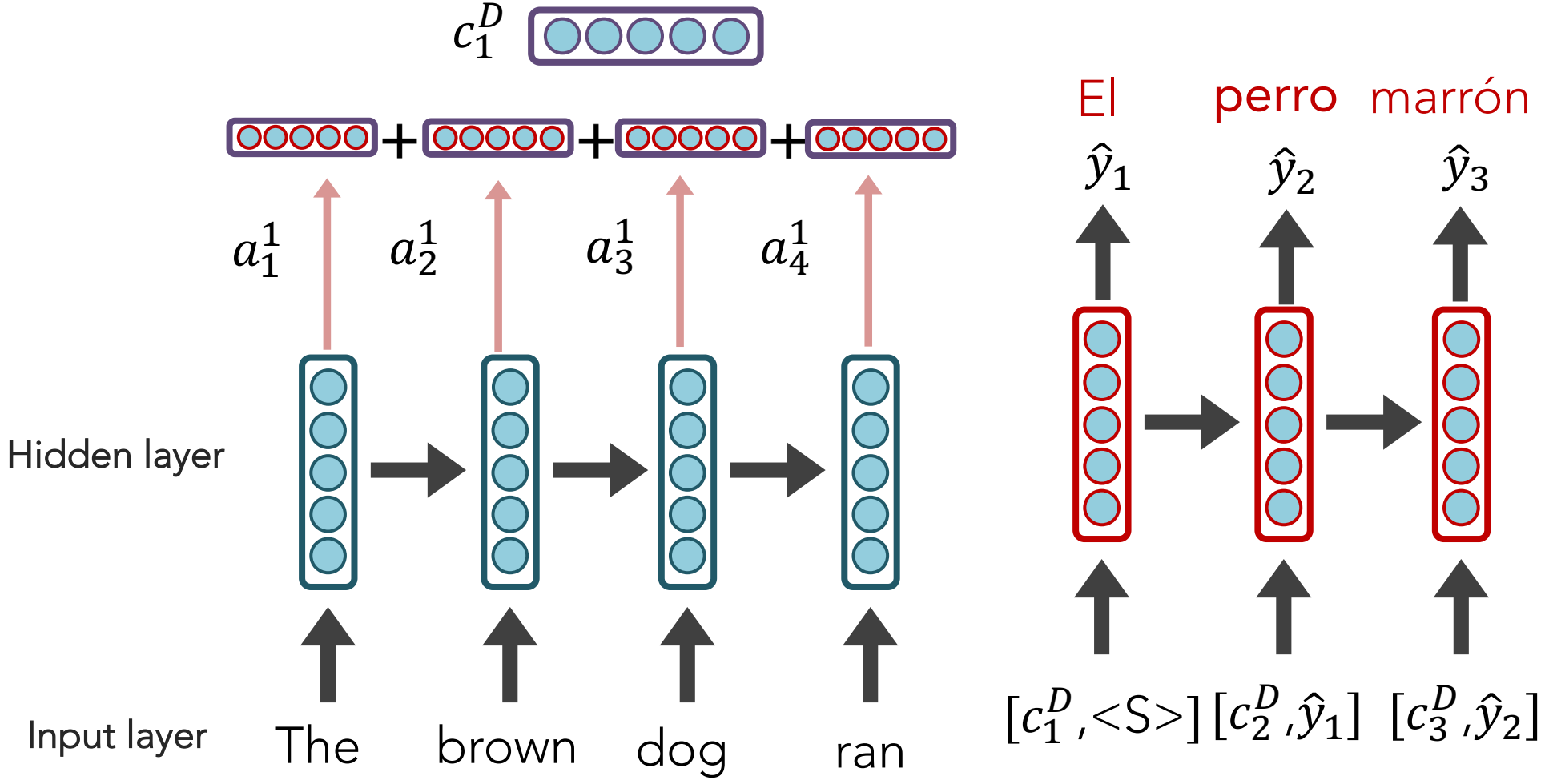
seq2seq + Attention

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



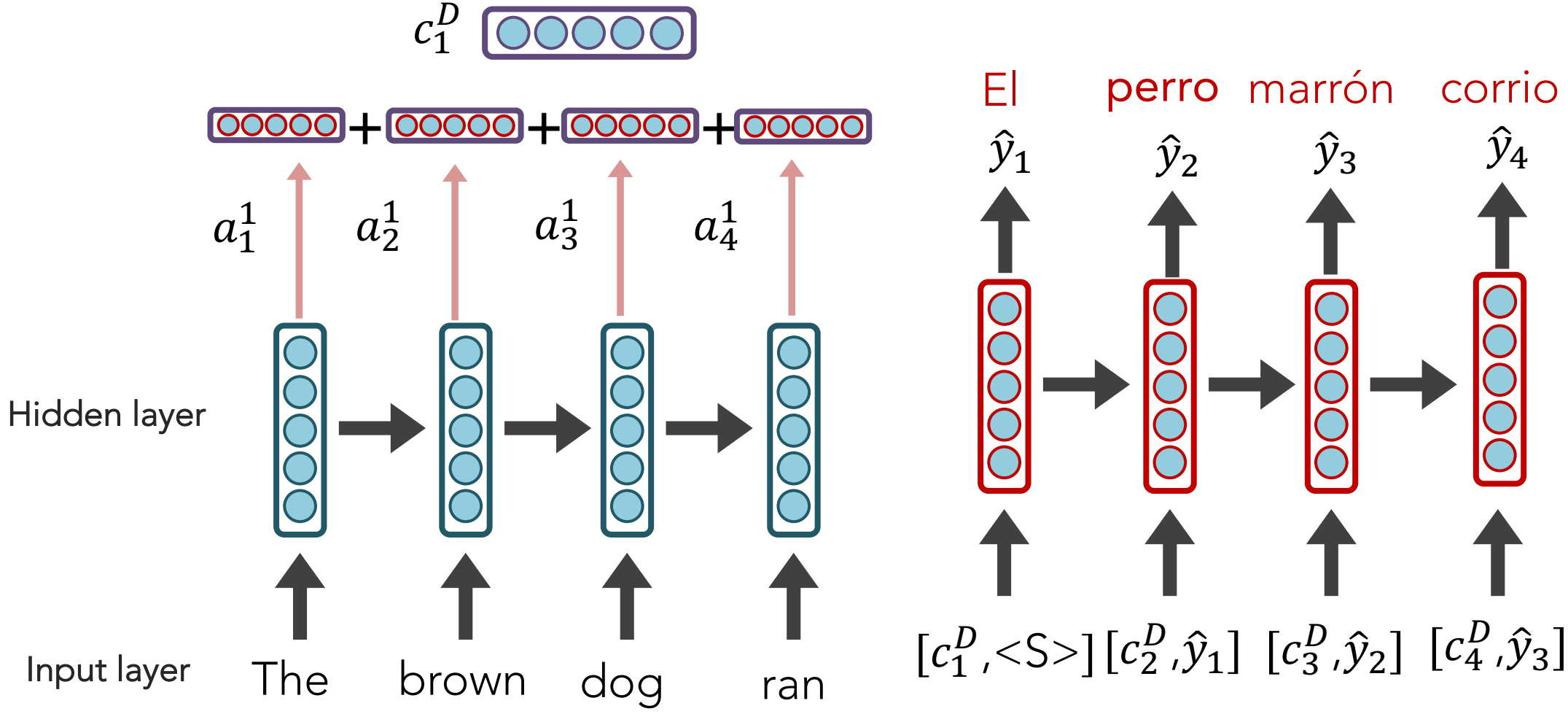
seq2seq + Attention

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



seq2seq + Attention

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



seq2seq + Attention

Attention:

- greatly improves seq2seq results
- allows us to visualize the contribution each word gave during each step of the decoder

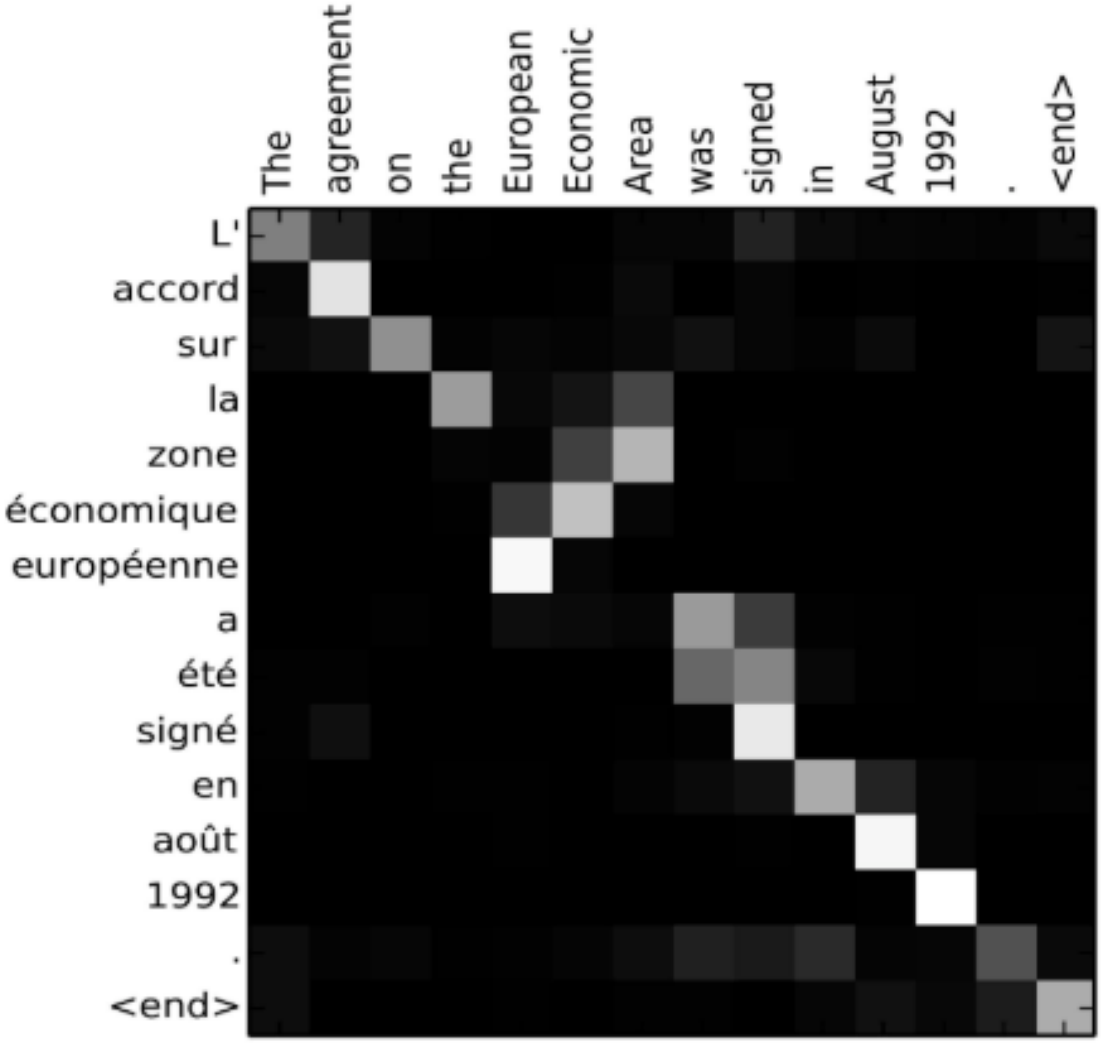


Image source: Fig 3 in Bahdanau et al., 2015

More reading

- Kalchbrenner and Blunsom (2013), Sutskever *et. al* (2014) and Cho. *et. al* (2014b)

- Bahdanau et. al (2015) https://arxiv.org/abs/1409.0473
  seq2seq with bidirectional GRU encoder + attention, score is FFNN.

- Luong et. al (2015) https://arxiv.org/abs/1508.04025
  Two-stacked LSTMs for the encoder and decoder, score experimented with cosine and FFNN. Context vector and output goes through another FFNN.

- Google's Neural Machine Translation (GNMT)
  https://arxiv.org/pdf/1609.08144.pdf
  8 LSTMs, where the first is bidirectional (whose outputs are concatenated), and a residual connection exists between outputs from consecutive layers (starting from the 3rd layer). The decoder is a *separate* stack of 8 unidirectional LSTMs.

Outline

Seq2seq with attention

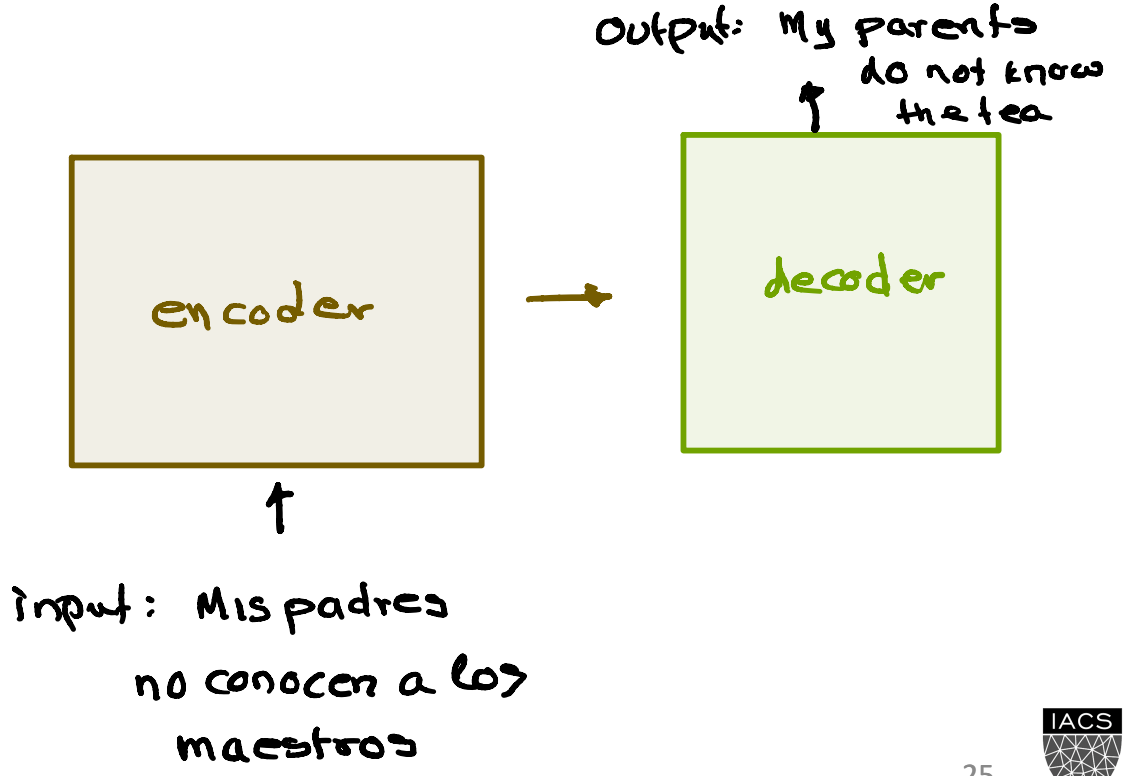
Transformers

Bert

Transformers

The Transformer is a model that uses attention to boost the speed with which seq2seq with attention models can be trained. The biggest benefit, however, comes from how The Transformer lends itself to parallelization. We will break it apart and look at how it functions.

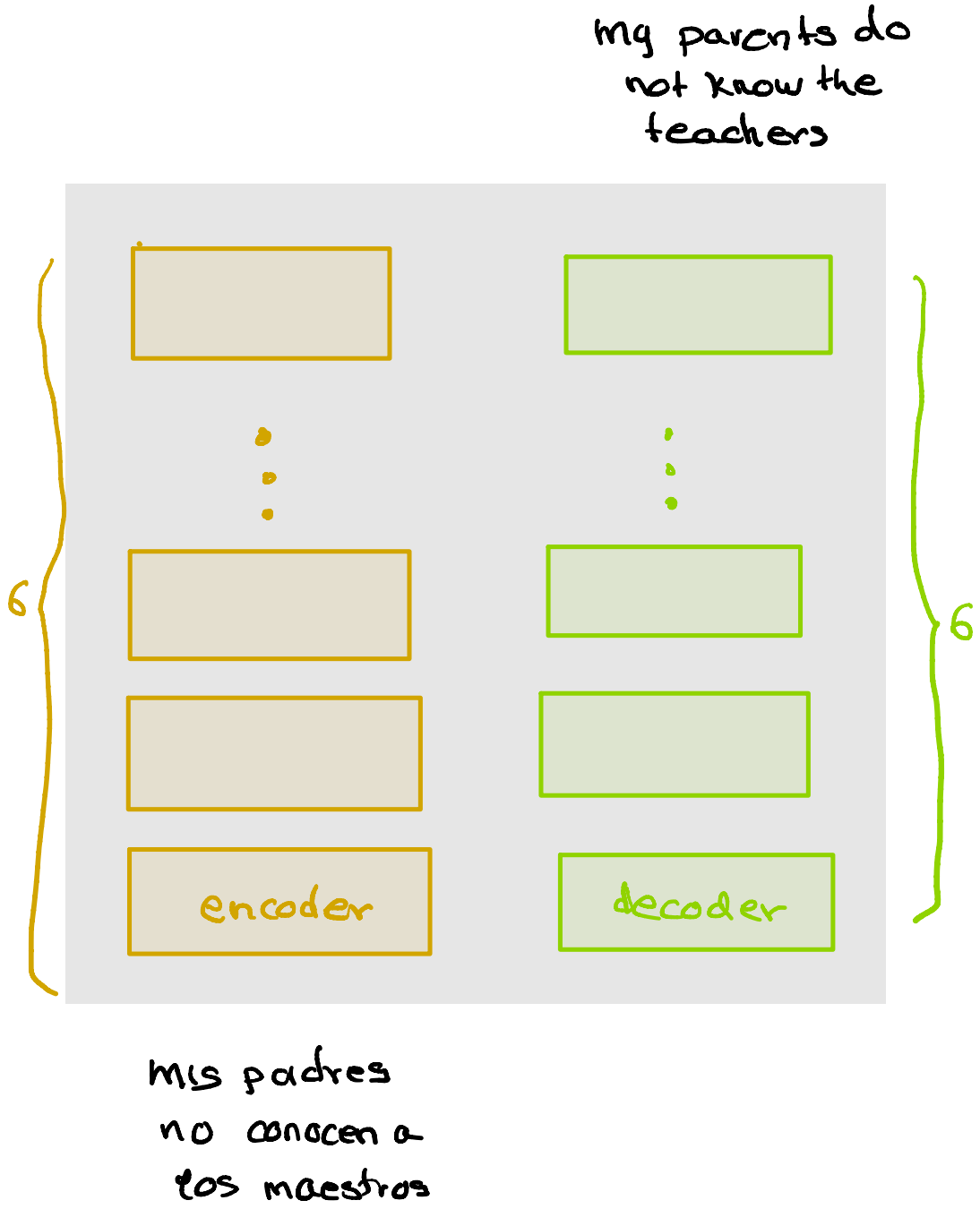
In its heart it contains an encoding component, a decoding component, and connections between them.



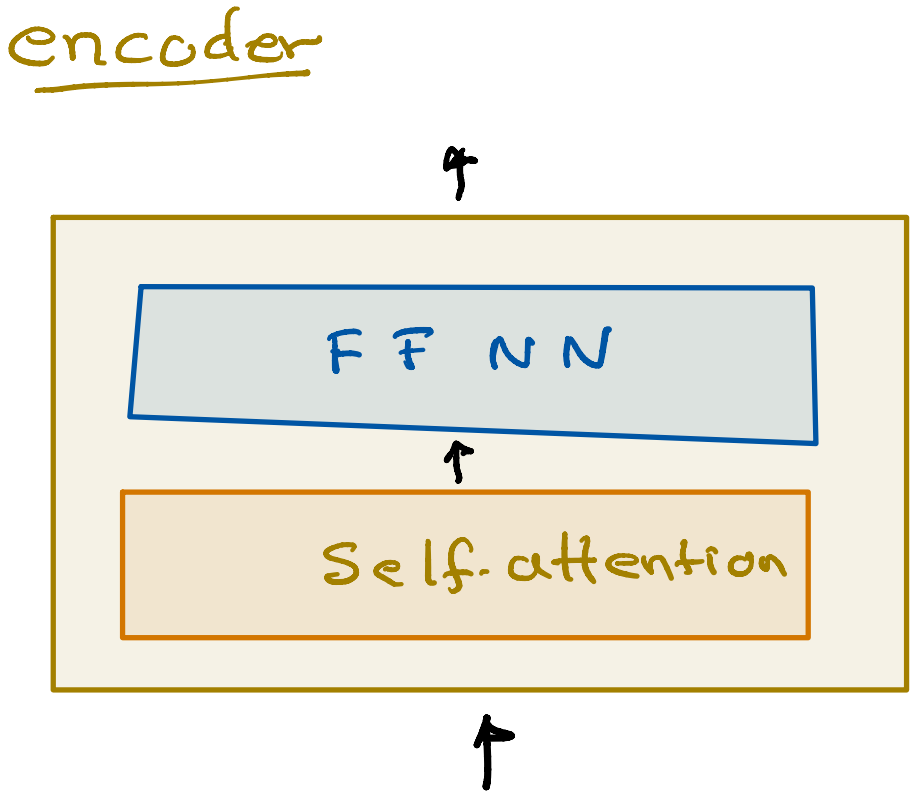
The encoding is a stack of encoders.

The original paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements).

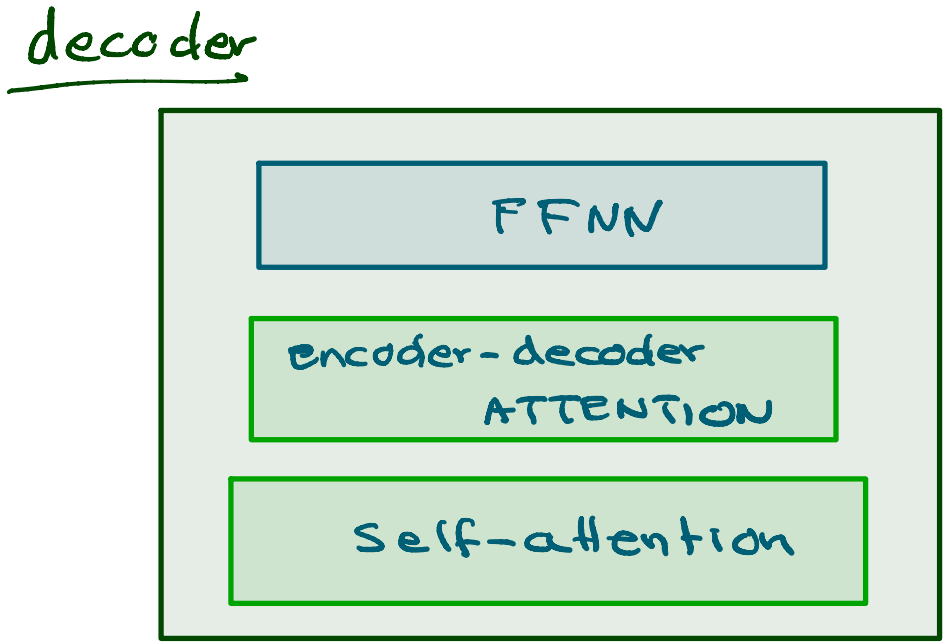
The decoding is a stack of decoders of the same number.

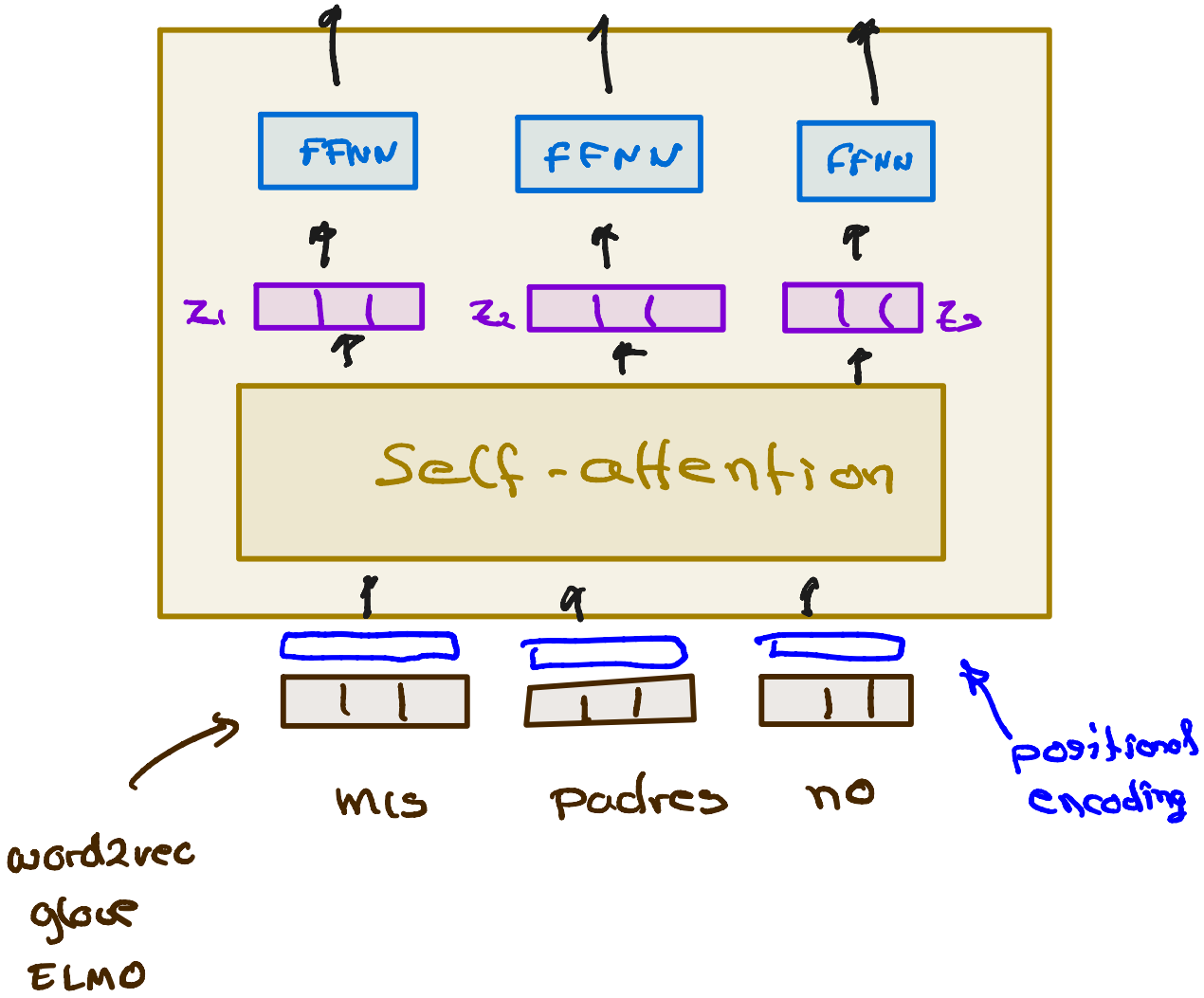


The encoder's inputs goes through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.



The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in seq2seq models).



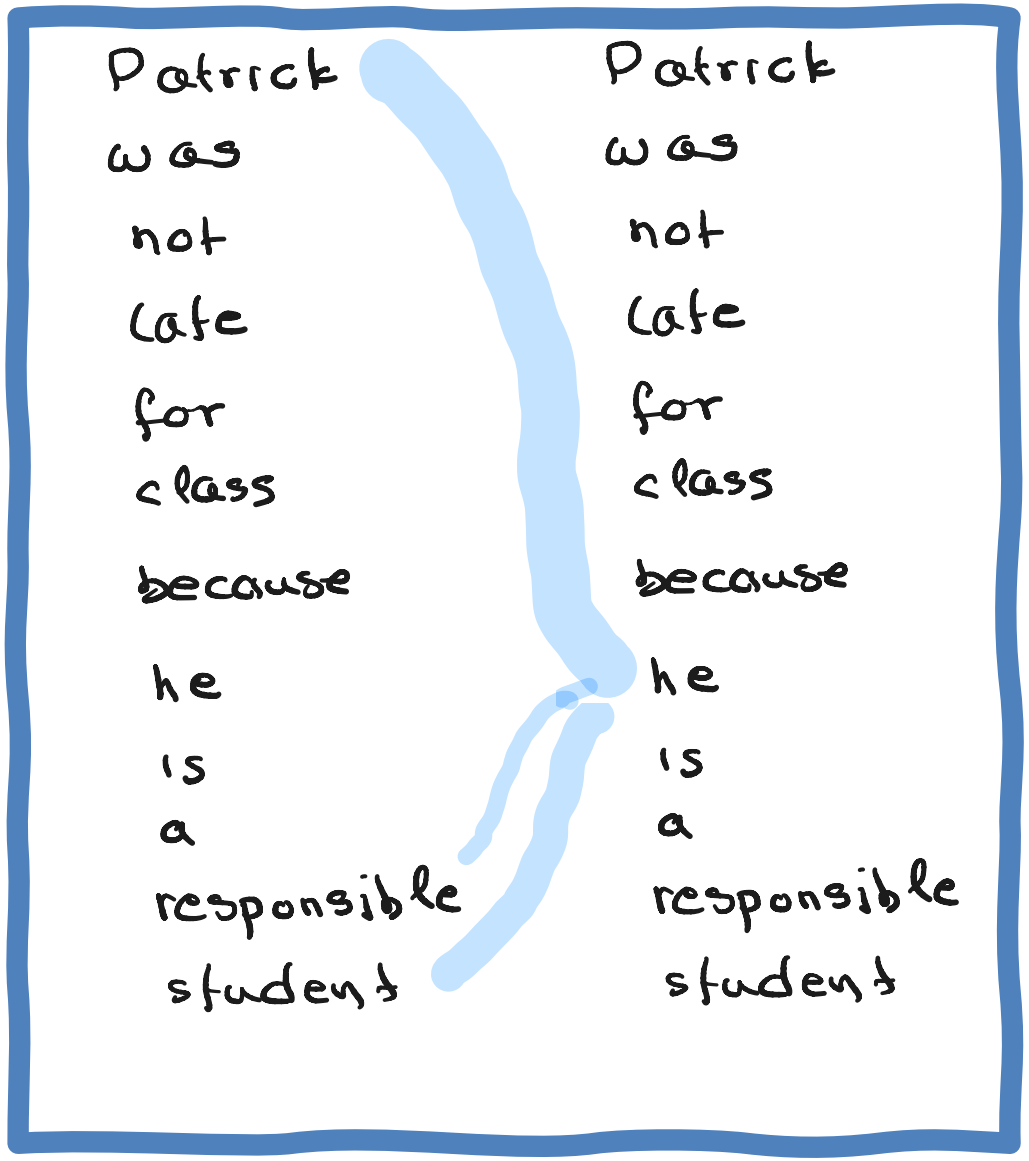


A key property of the Transformer, which is that the word in each position flows through its own path in the encoder. There are dependencies between these paths in the self-attention layer.

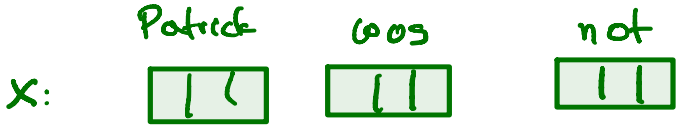
The feed-forward layer does not have those dependencies, Therefore, the various paths can be executed in parallel while flowing through the feed-forward layer.

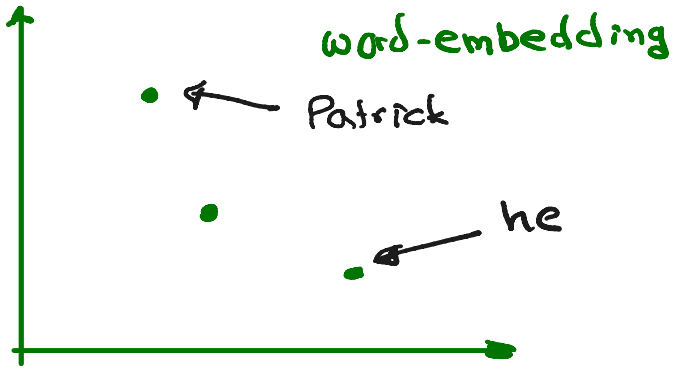
Patrick was not late for class because he is a responsible student

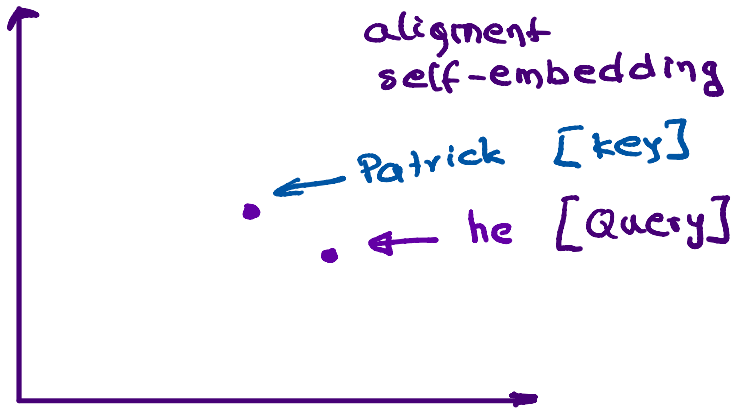
What does "he" in this sentence refer to? Is it referring to the class or to the student? It's a simple question to a human, but not as simple to an algorithm.
When the model is processing the word "he", self-attention allows it to associate "he" with "Patrick".



self-attention







Query:

$$Q = X W^{Q}$$

transformation

$n_w \times n_a$ $\quad$ $n_w \times n_e$ $\quad$ $n_e \times n_a$

key:

$$K = X W^{K}$$

Value:

$$V = X W^{V}$$

Similarity:

$$Z^{T} = \text{softmax}\left(\frac{Q^{T} \times K}{\sqrt{d}}\right) \cdot V^{T}$$

$n_a \times n_e$ $\quad$ $n_e \times n_a$ $\quad$ $n_w \times n_a$

$n_w \times n_a$

More heads:

We do the same 8 times

- HEAD 0 HEAD 1 HEAD 2

Z_0 Z_1 Z_2

- Concat and linearly combine

$Z =$  $\times W^Z$

$\text{Softmax}(Q \times K)$

↳ attention