Lecture 9: Transformer

From Attention to Self-Attention

Harvard

AC295/CS287r/CSCI E-115B

Chris Tanner

Self [Attention]

-- Mac Miller (2018)

David S. Rosenthal
center for wellness
and health promotion
Harvard University Health Services

https://wellness.huhs.harvard.edu/alcohol-substance-use

basics@huhs.harvard.edu

ANNOUNCEMENTS

- Quizzes 3 have been graded and are logged on Canvas
- HW1 is being graded. Solutions are posted on Canvas -> Files
- HW2 is due next Tues, Oct 5 @ 11:59pm! Determine your mystery language.
- Research Proposals are due tonight, Sept 30 @ 11:59pm.
  - If submitting w/ others, please see the updated Canvas instructions

RESEARCH PROJECTS

- Most research experiences/opportunities are "top-down"
- You're all creative and fully capable.
- Allow yourselves to become comfortable with the unknown.
- It's okay if your Phase 1 Proposals aren't *perfect* ideas. The point is to gain practice with the inquisition and overall process of executing your ideas.
- We'll help provide structure after Phase 1 (e.g., filtering projects, feedback, helping you find the optimal project partners, TF support, etc)

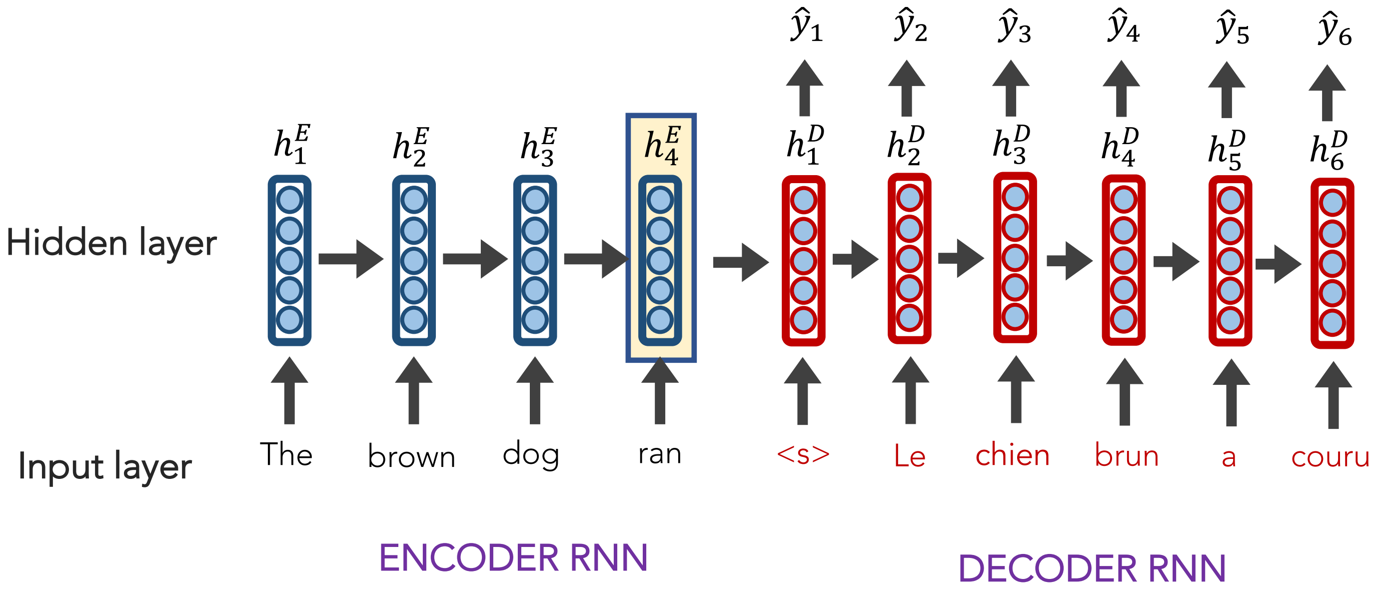
RESEARCH PROJECTS

- I'll filter projects by rating them according to:
  - researchy vs application
  - how grounded/well-reasoned it is
  - technical difficulty (there's a sweet spot)
  - feasibility (e.g., required compute power, data availability, metrics)
  - interestingness / significance

RECAP: L8

seq2seq models

- are a general-purpose encoder-decoder architecture
- can be implemented with RNNs (or Transformers even)
- Allow for n → m predictions
- Natural approach to **Neural MT**
- If implemented end-to-end can be good but slow

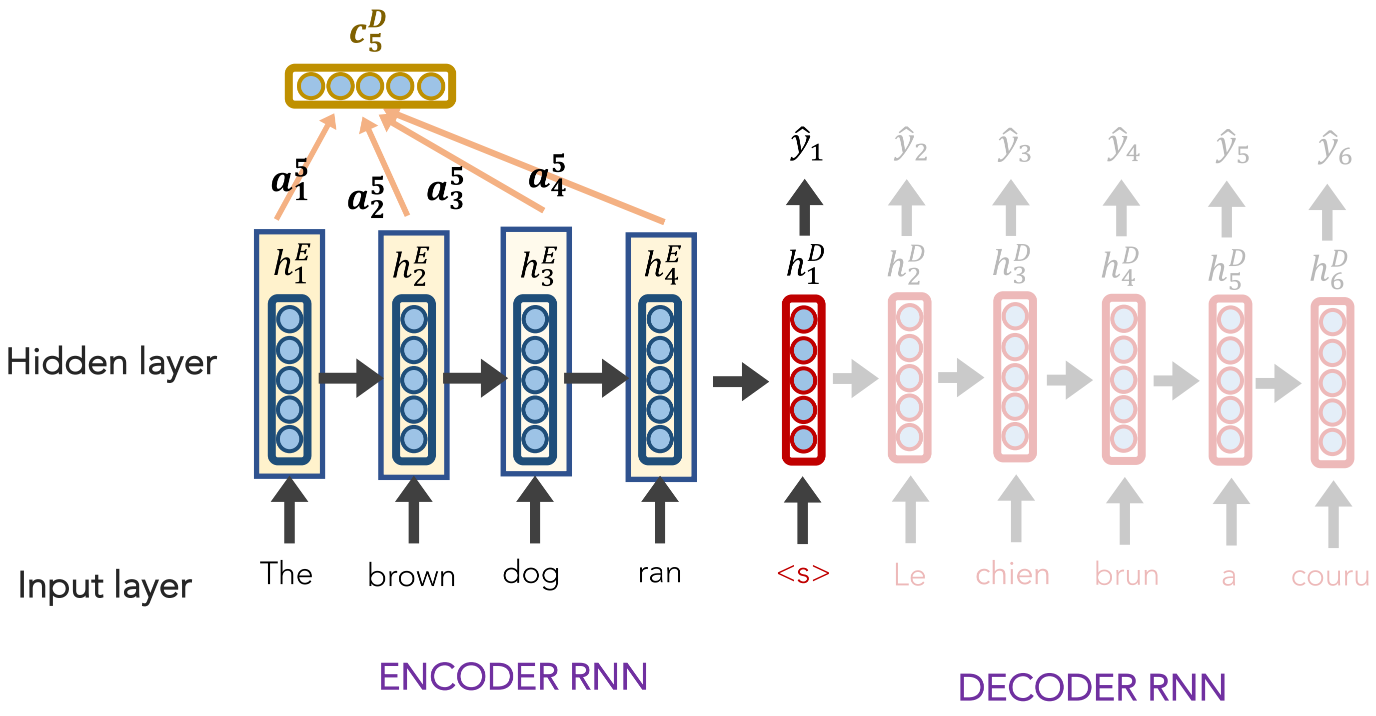


RECAP: L8

seq2seq models

- **Attention** allows a decoder, at each time step, to focus/use different amounts of the encoder's hidden states

- The resulting context vector c_i is used, with the decoder's current hidden state h_i, to predict $\hat{y}_i$



RECAP: L8

MT

$$\mathrm{argmax}_{\mathrm{y}} P(x|y)P(y)$$

- Converts text from a source language x to a target language y
- SMT made huge progress but was brittle
- NMT (starting w/ **LSTM-based seq2seq models**) blew SMT out of the water
- Attention greatly helps **LSTM-based seq2seq models**
- **Next:** Transformer-based seq2seq models w/ Self-Attention and Attention

Outline

- seq2seq + Attention
- Self-Attention

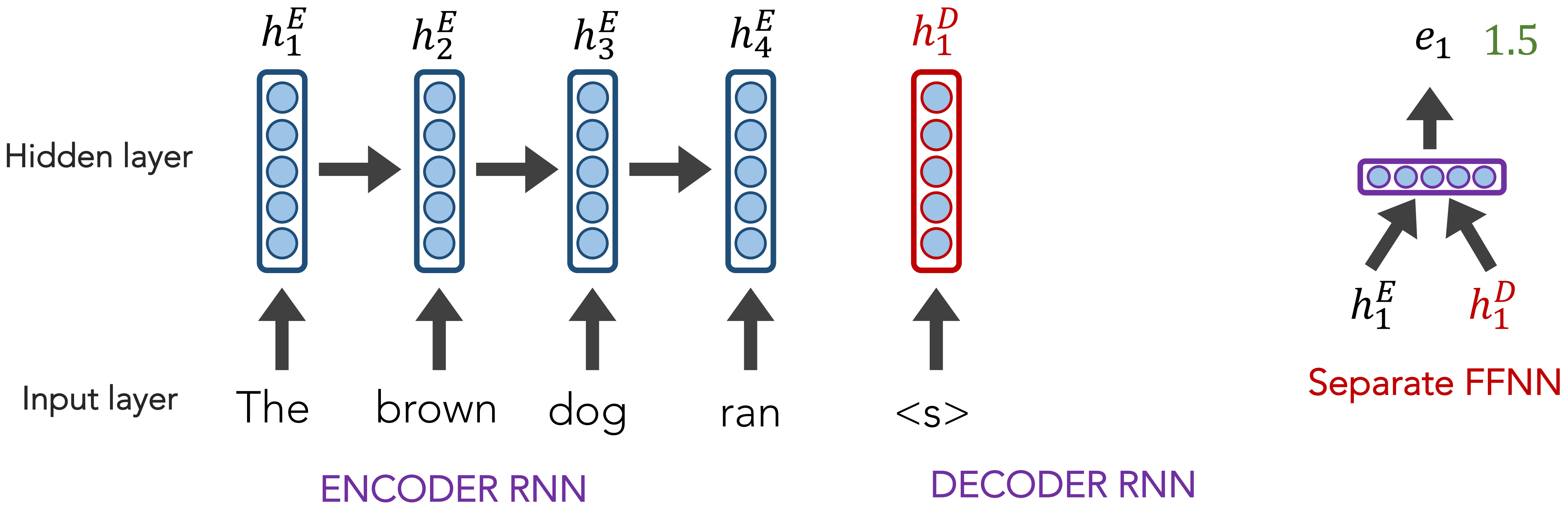
Outline

- seq2seq + Attention
- Self-Attention

seq2seq + Attention

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

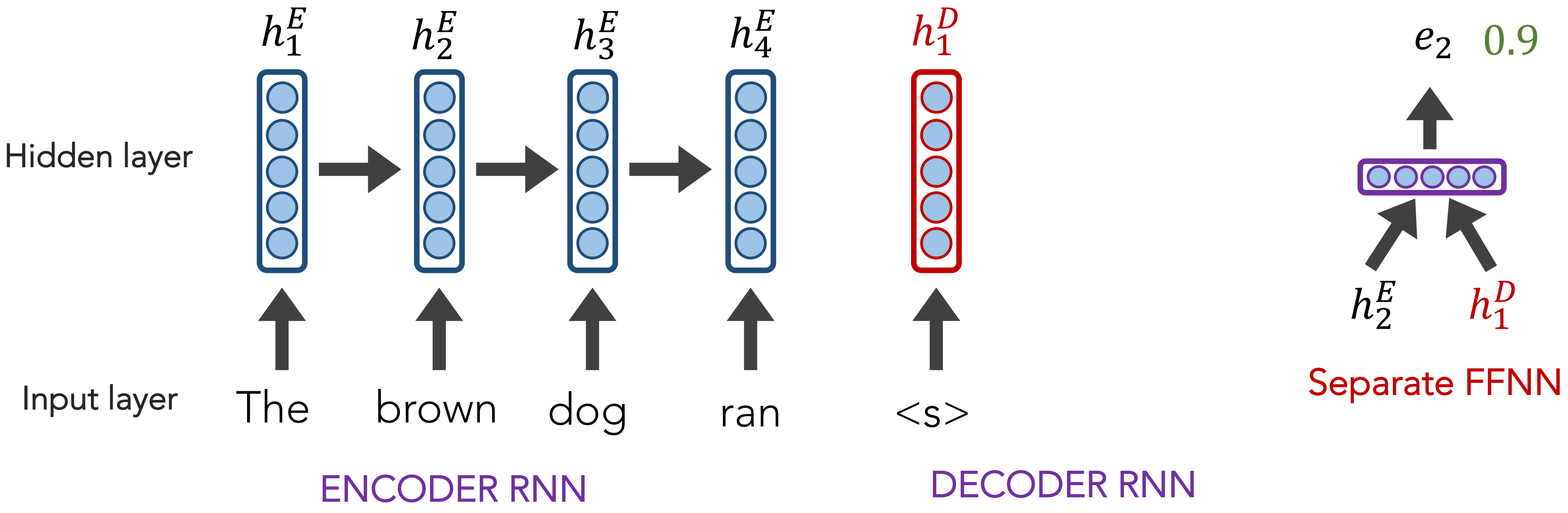
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

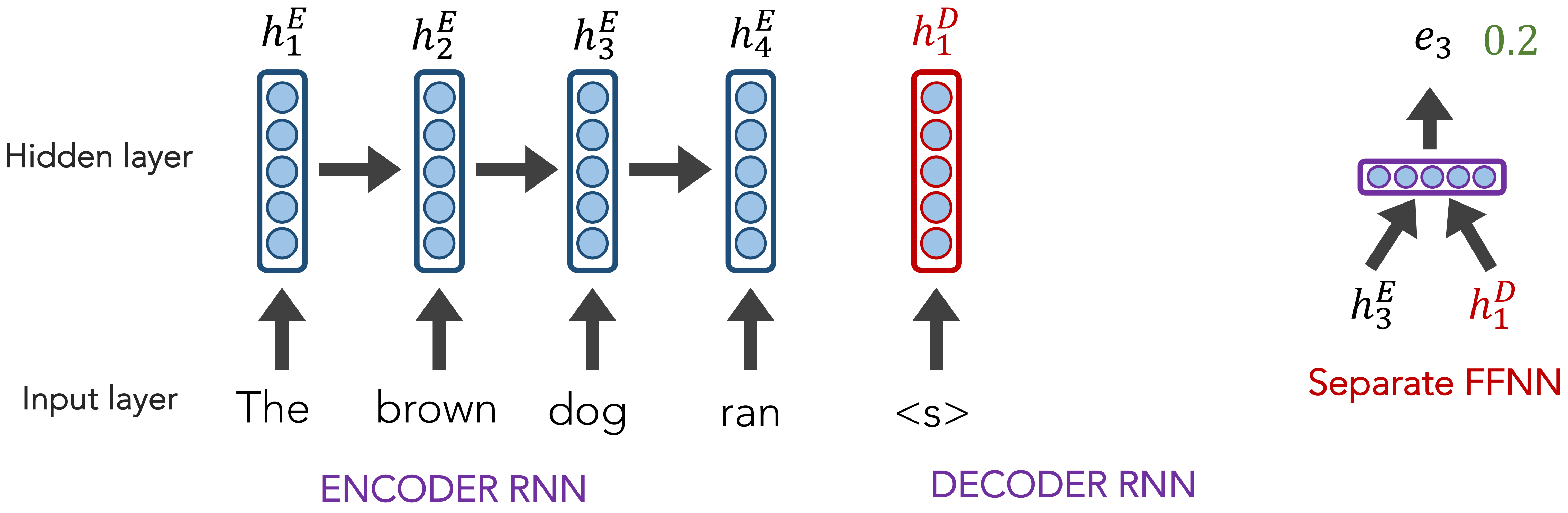
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

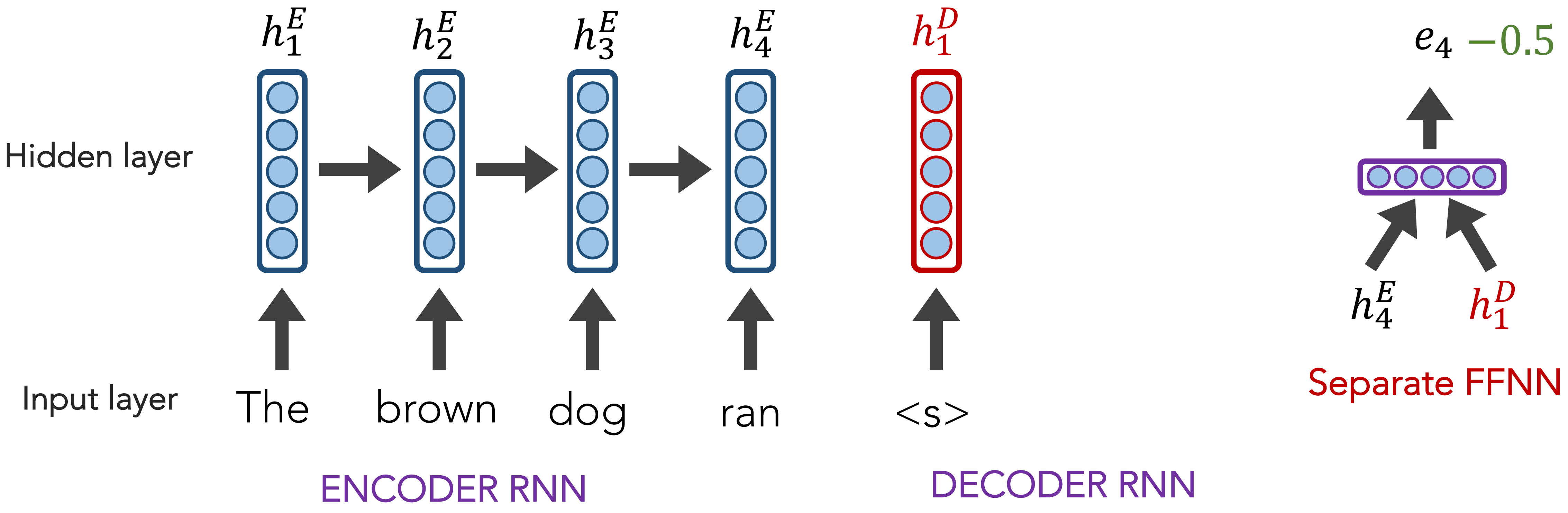
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

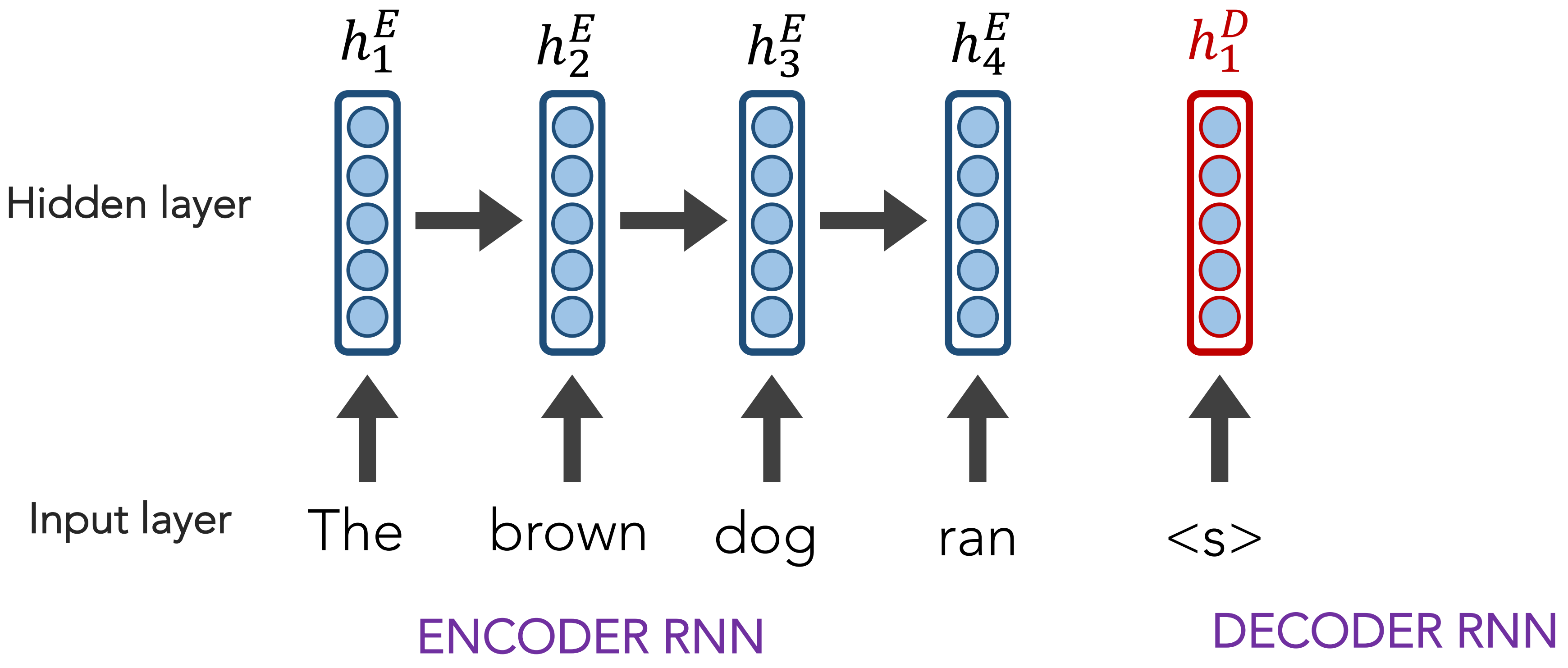
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



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

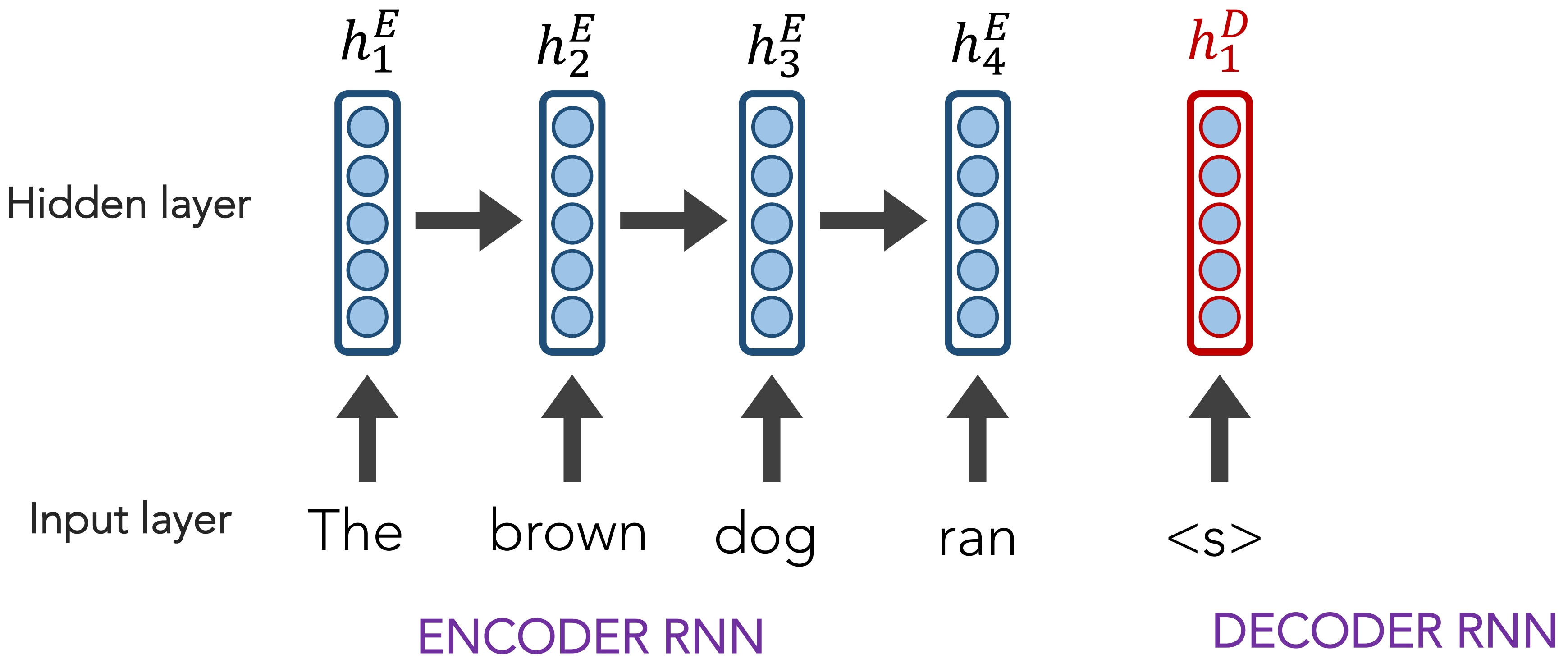
e_3 0.2

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

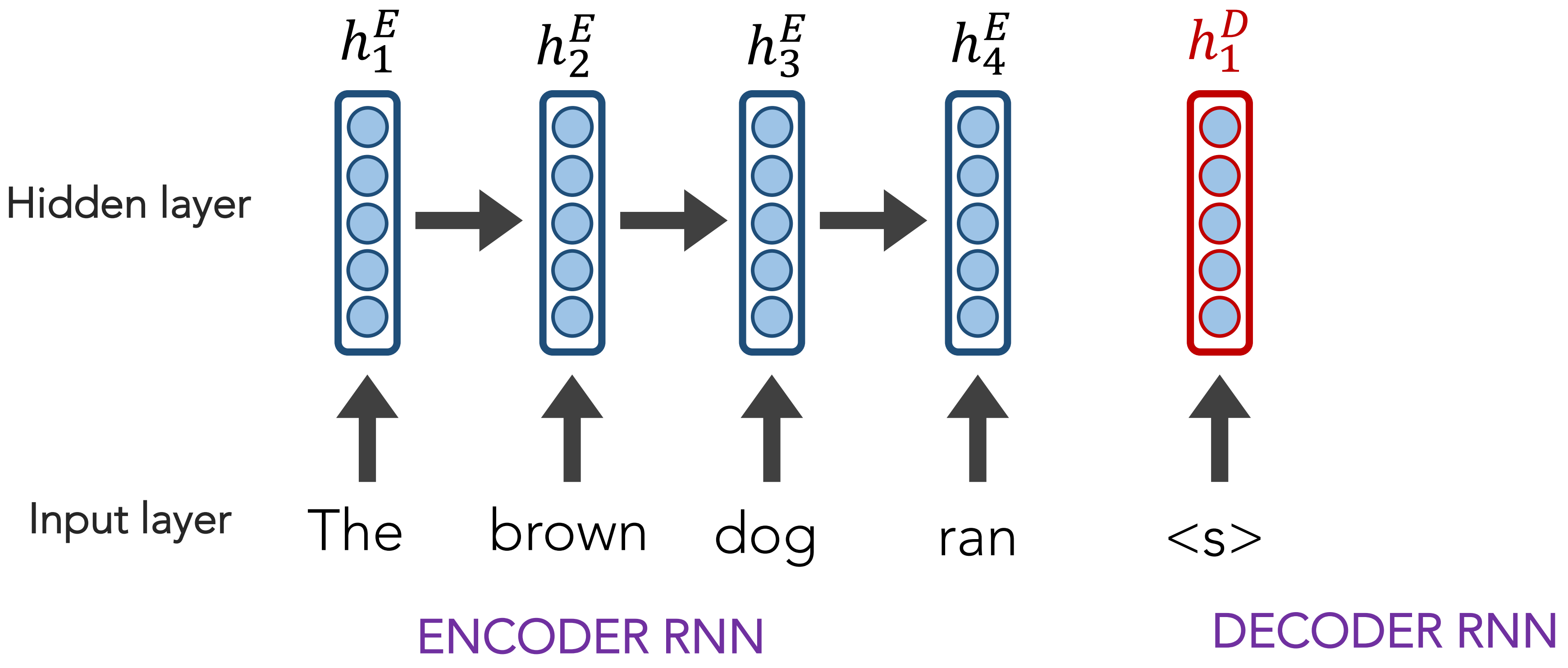
Attention (softmax'd)

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

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!



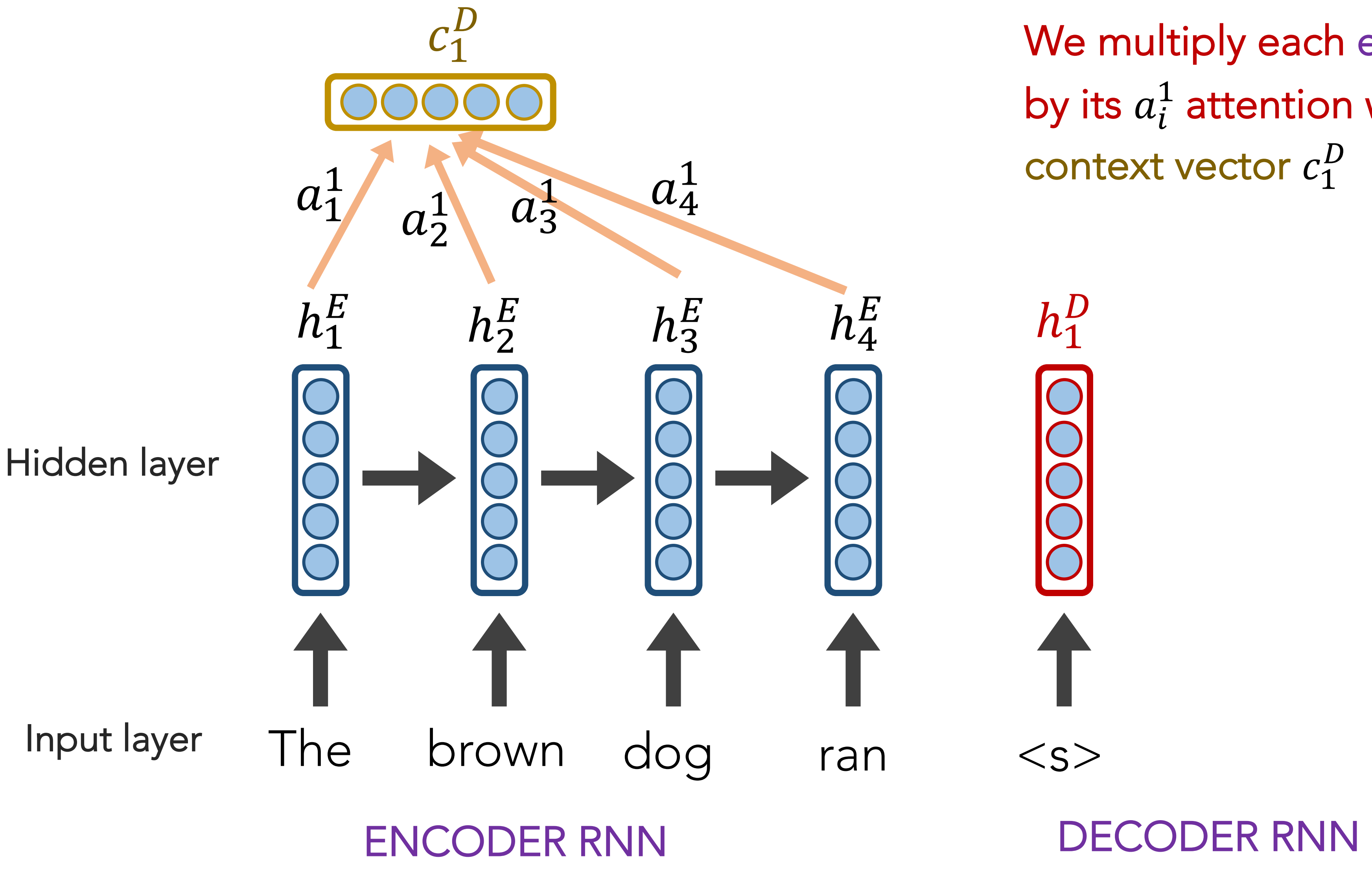
Attention (raw scores)

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)

$a_1^1 = 0.51$

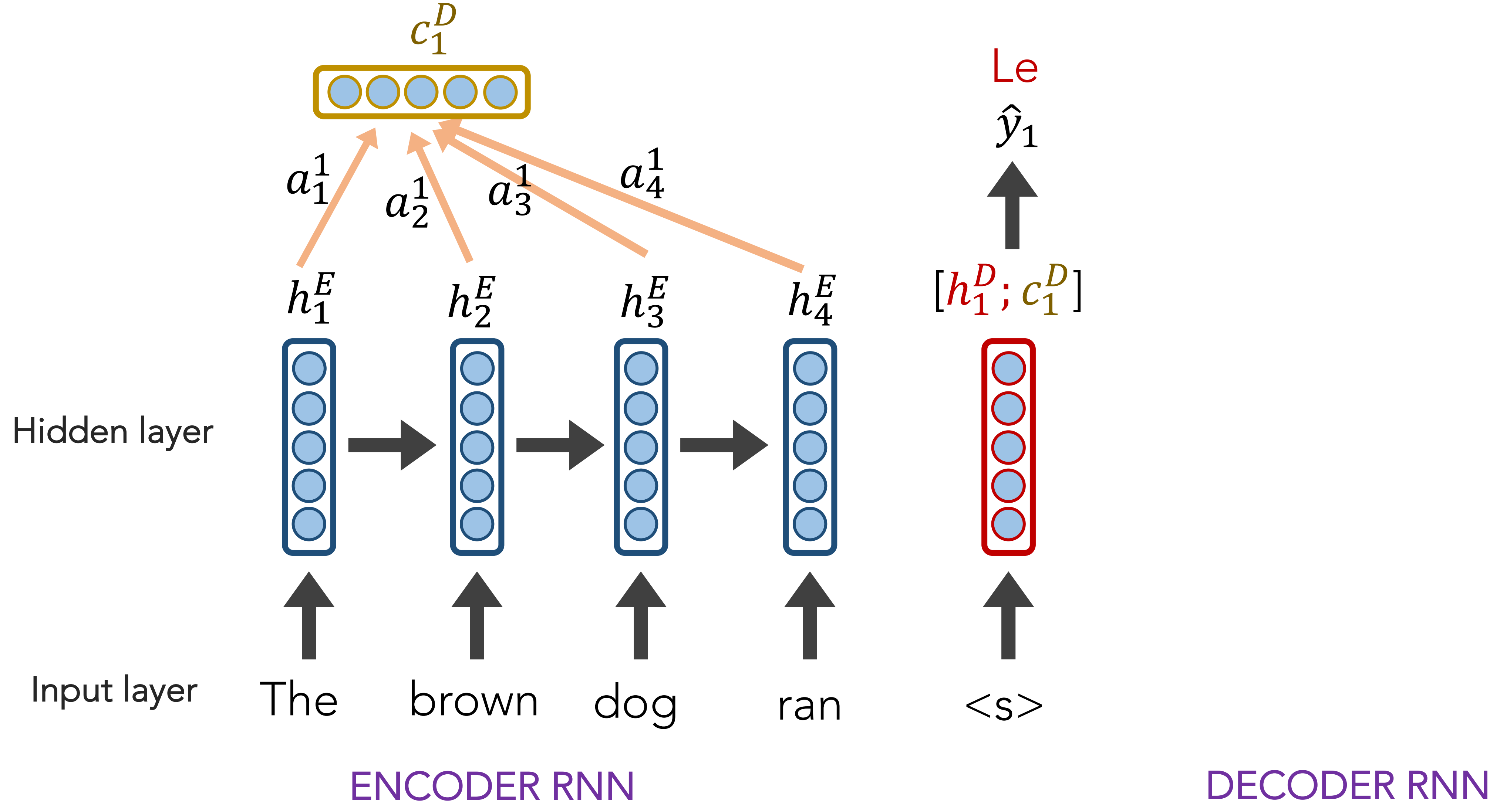
$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

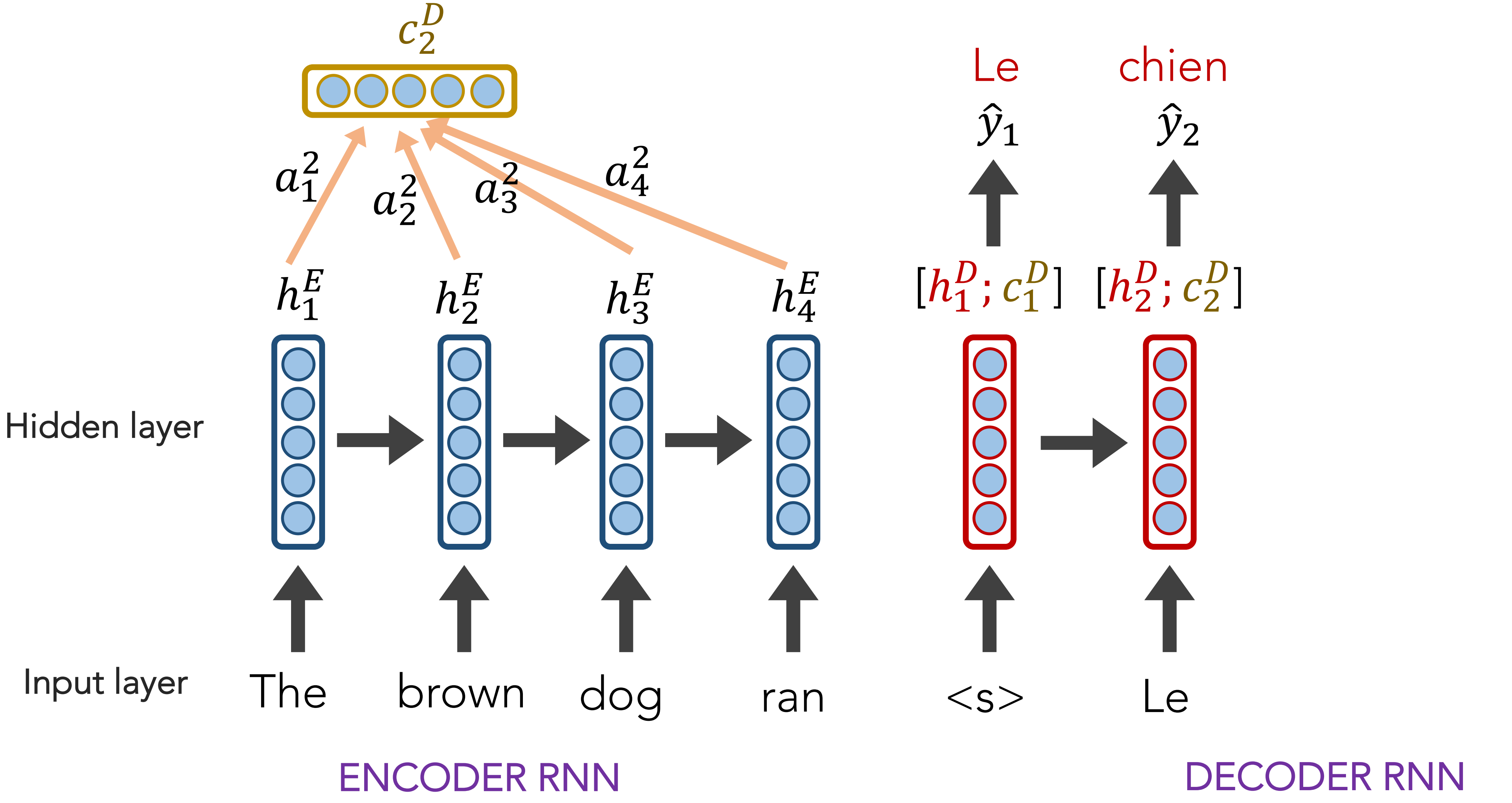
seq2seq + Attention

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



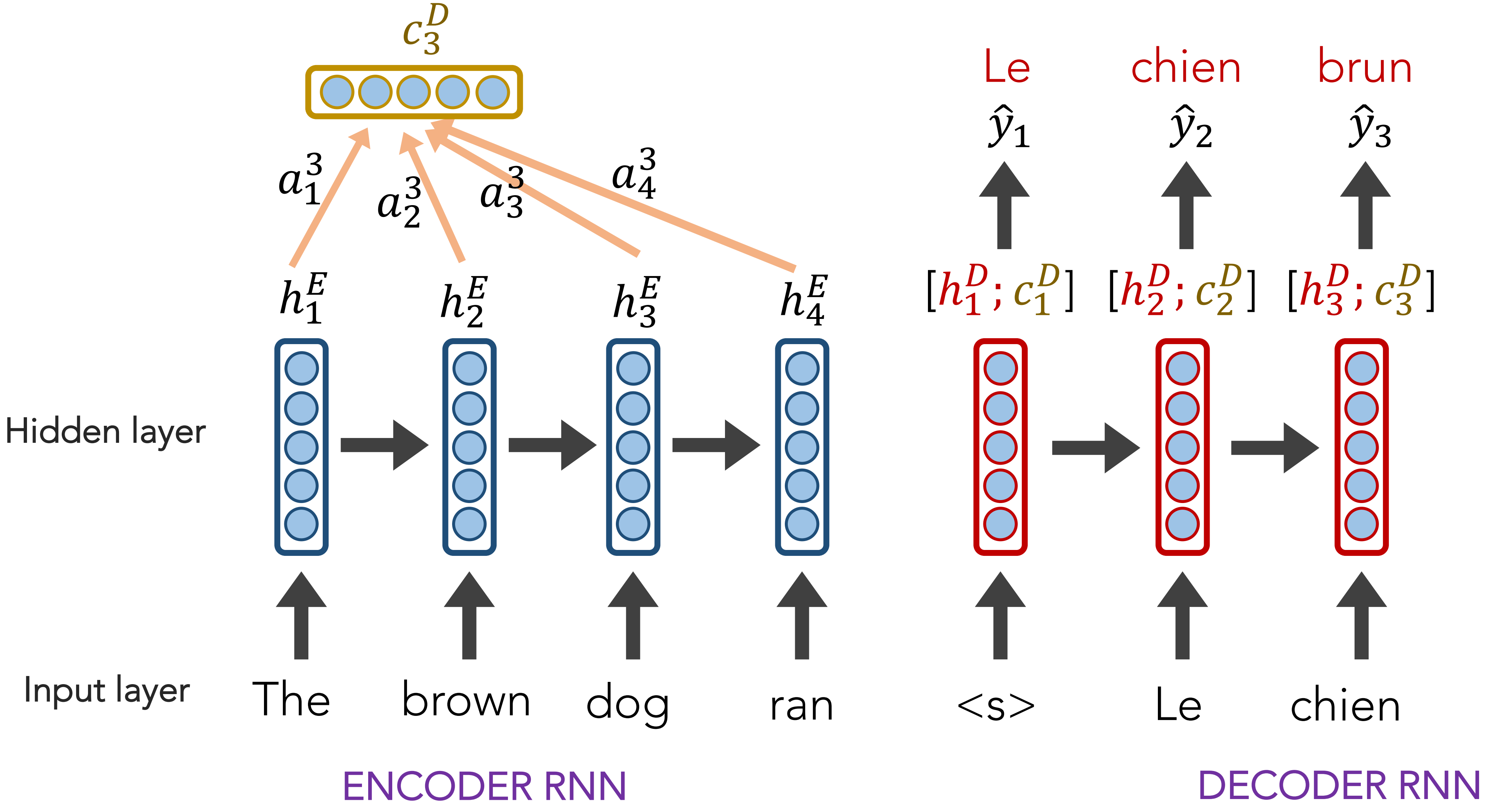
seq2seq + Attention

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



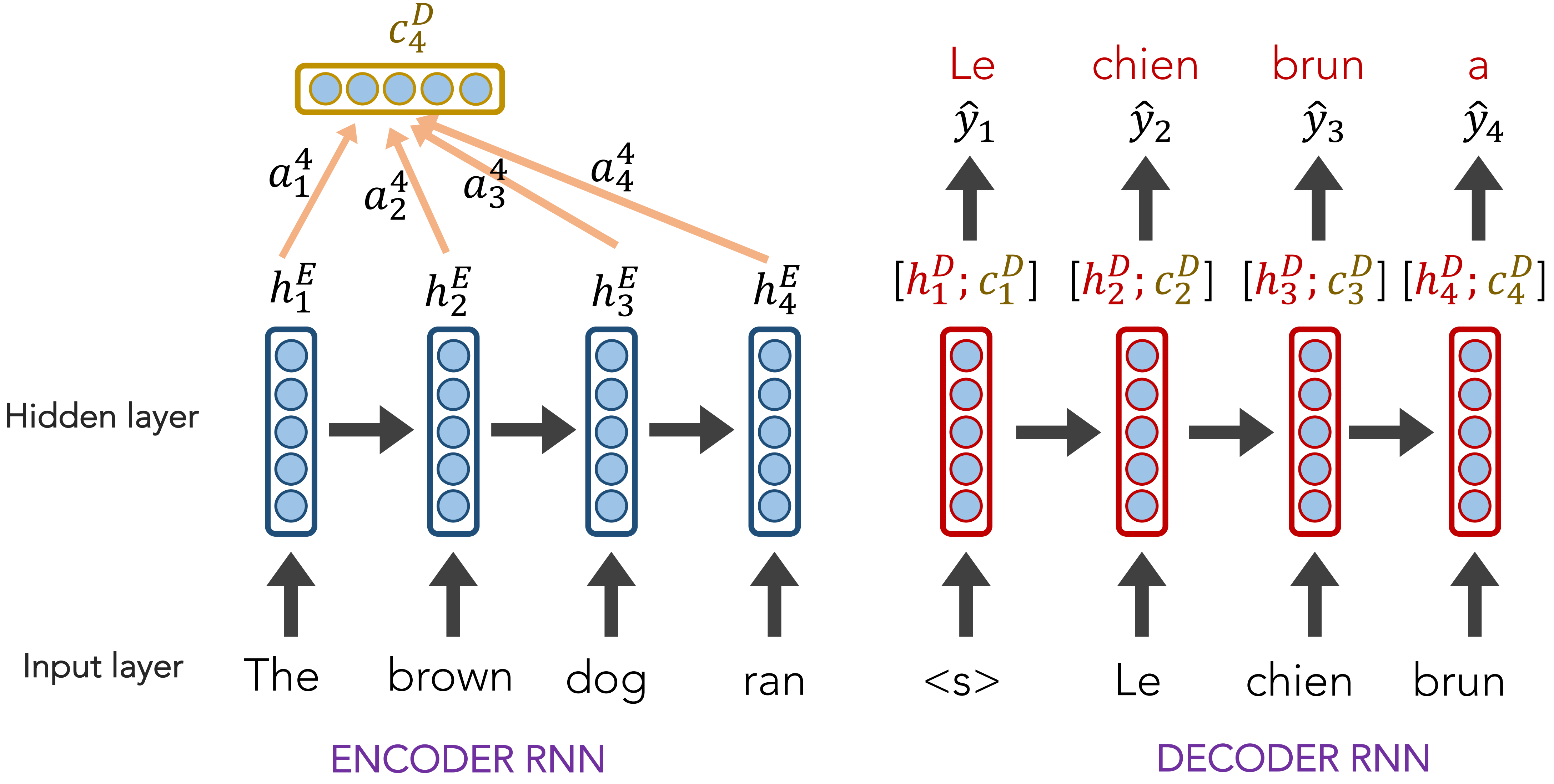
seq2seq + Attention

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



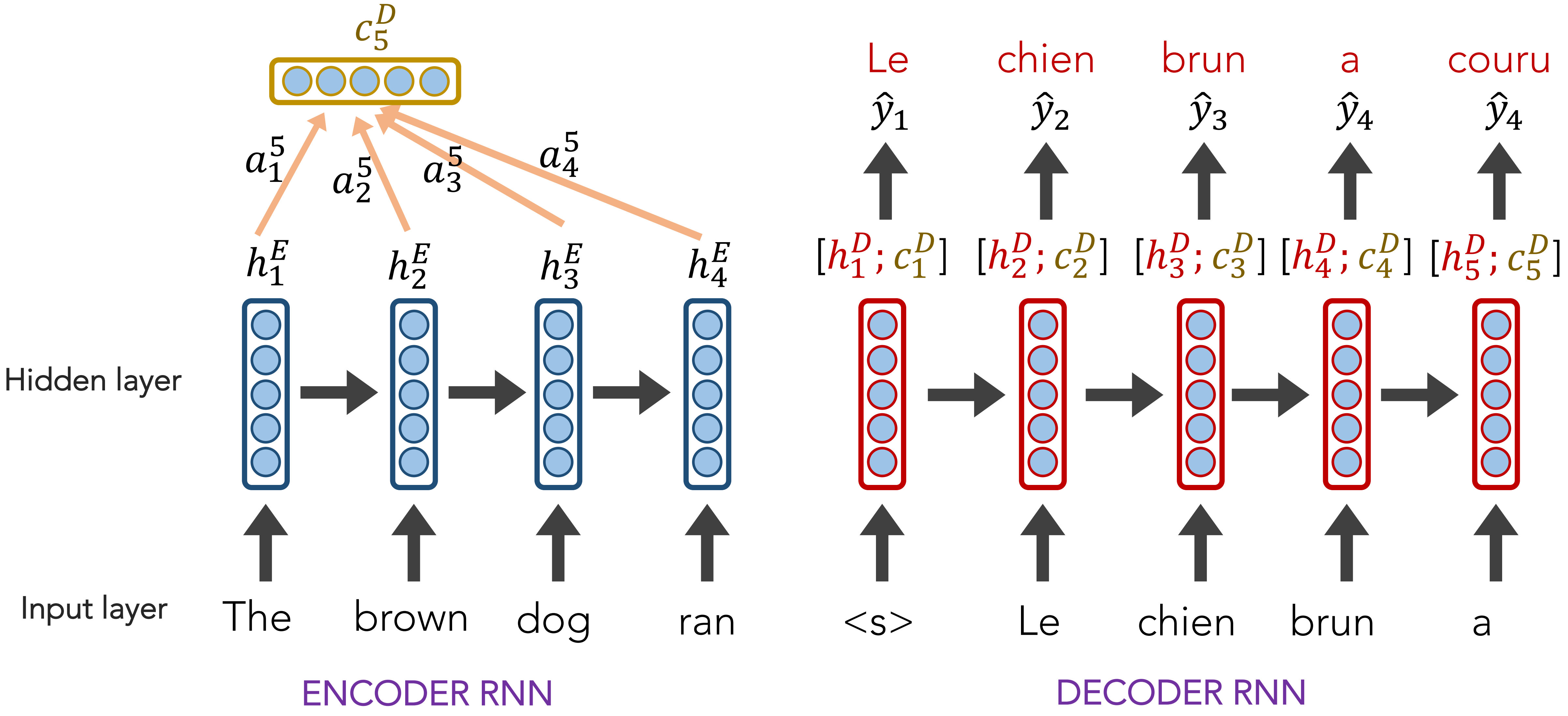
seq2seq + Attention

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

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

"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?"

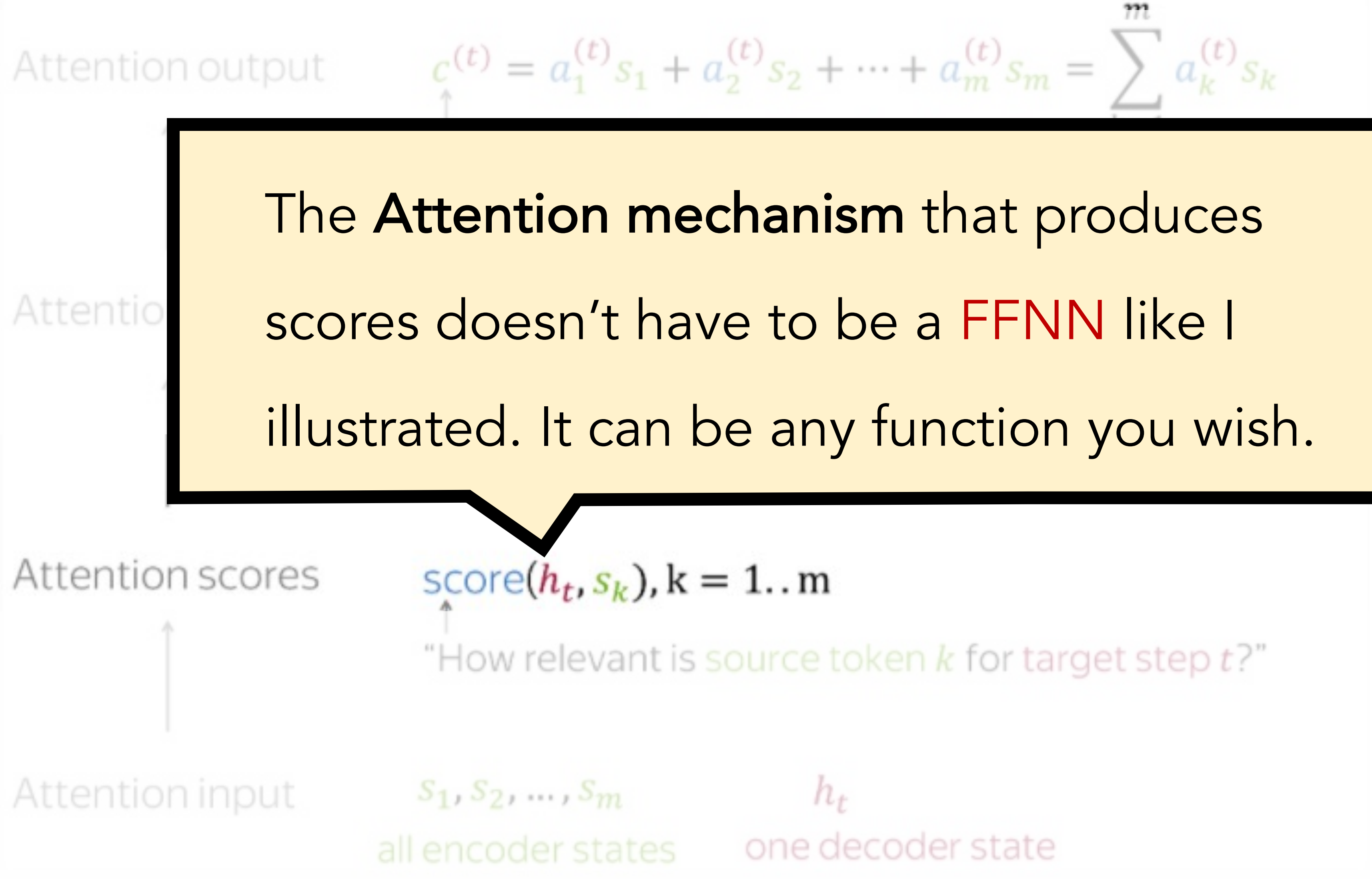
Attention input

$s_1, s_2, \dots, s_m$ all encoder states

h_t one decoder state

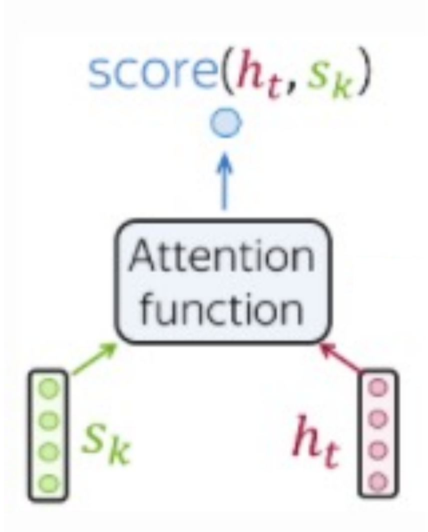
Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

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:

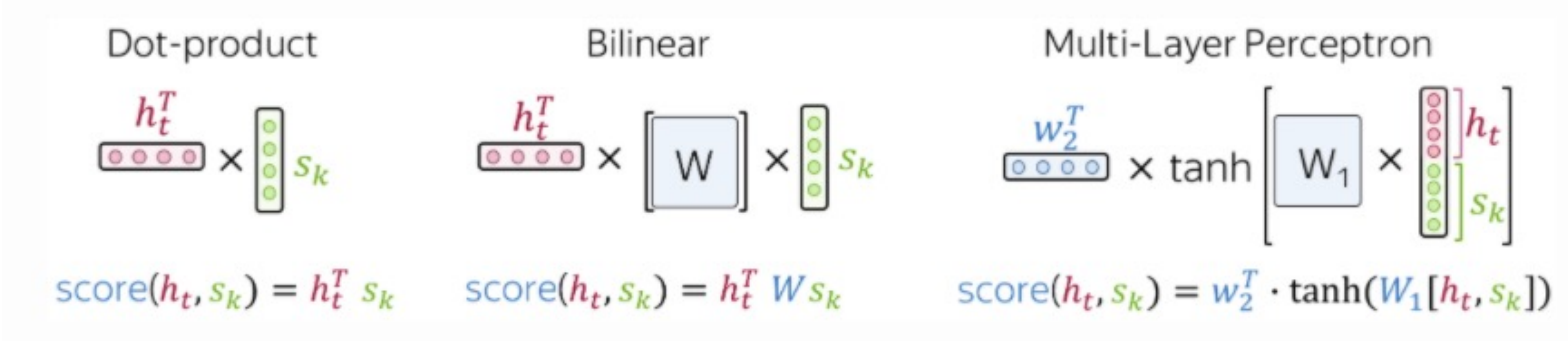


Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

seq2seq + Attention

Attention:

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

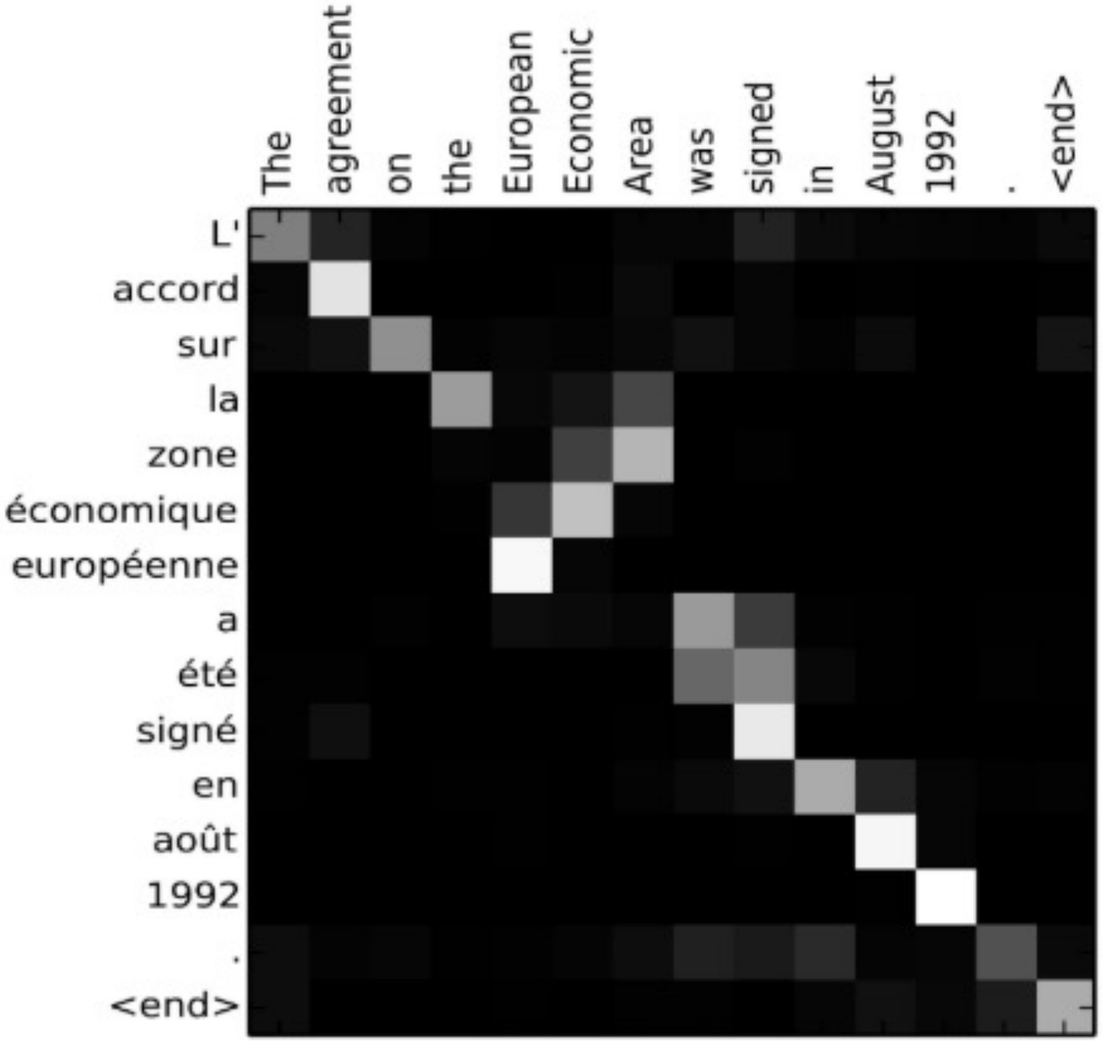


Image source: Fig 3 in [Bahdanau et al., 2015]()

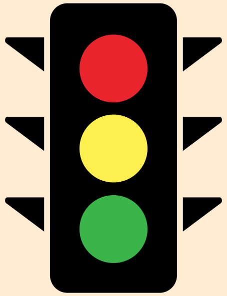
Takeaway:

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]

CHECKPOINT



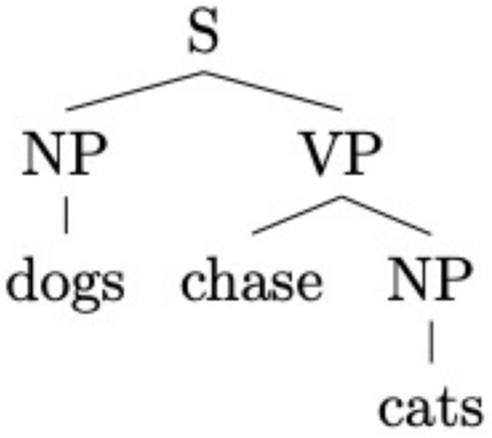
- seq2seq doesn't have to use RNNs/LSTMs
- seq2seq doesn't have to be used exclusively for NMT
- NMT doesn't have to use seq2seq

  (but it's natural and the best we have for now)

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:

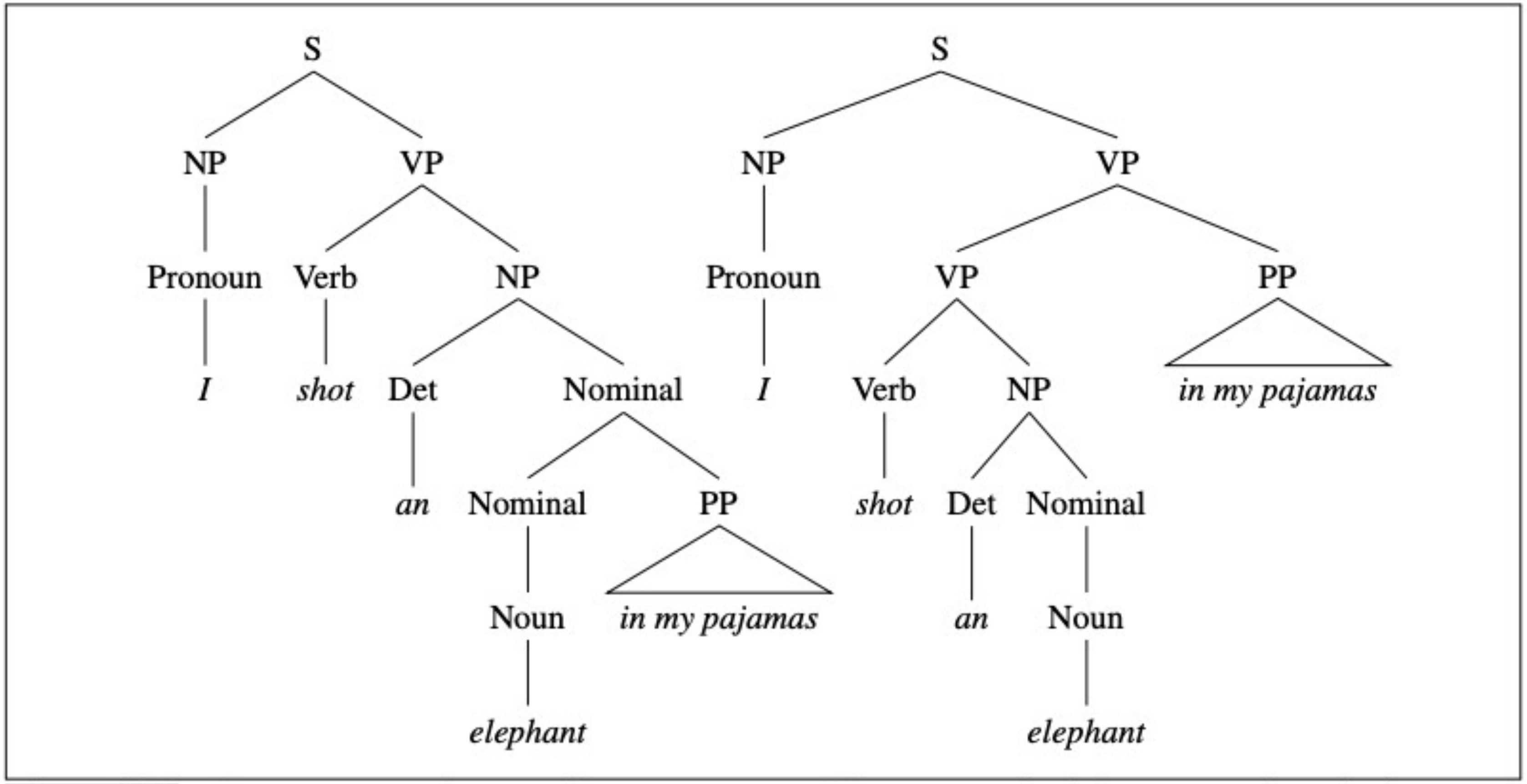


Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

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)

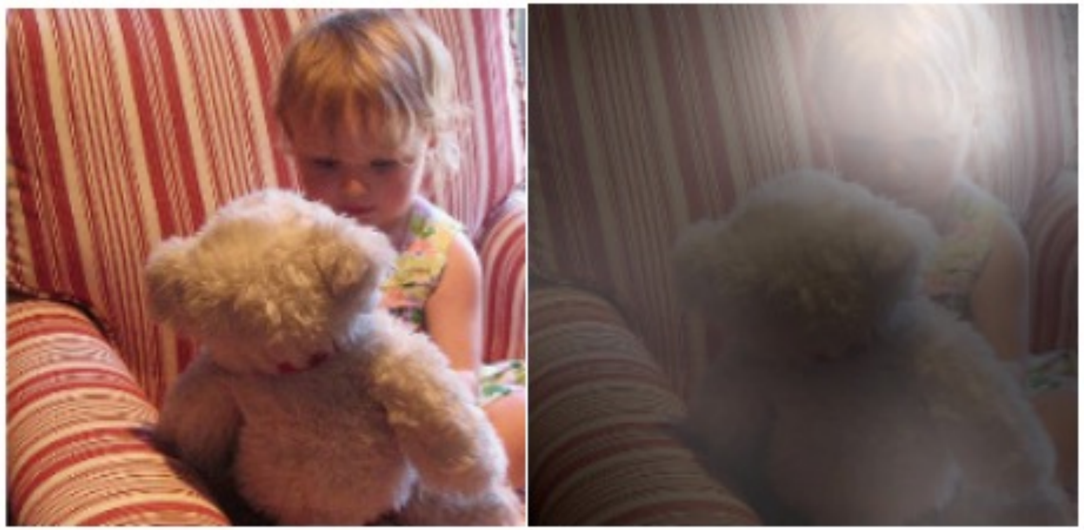
Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Xu et al. CVPR (2016)

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

A large white bird standing in a forest.   A woman holding a clock in her hand.

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:

SUMMARY

- LSTMs are sequential in nature (prohibits parallelization). Very wasteful.
- No <u>explicit</u> modelling of long- and short- range dependencies
- Language is naturally hierarchical

  (can we do better than Stacked LSTMs?)
- Can we apply the concept of Attention to improve our **representations**?

  (i.e., *contextualized representations*)

Outline

- seq2seq + Attention
- Self-Attention

Outline

- seq2seq + Attention
- Self-Attention

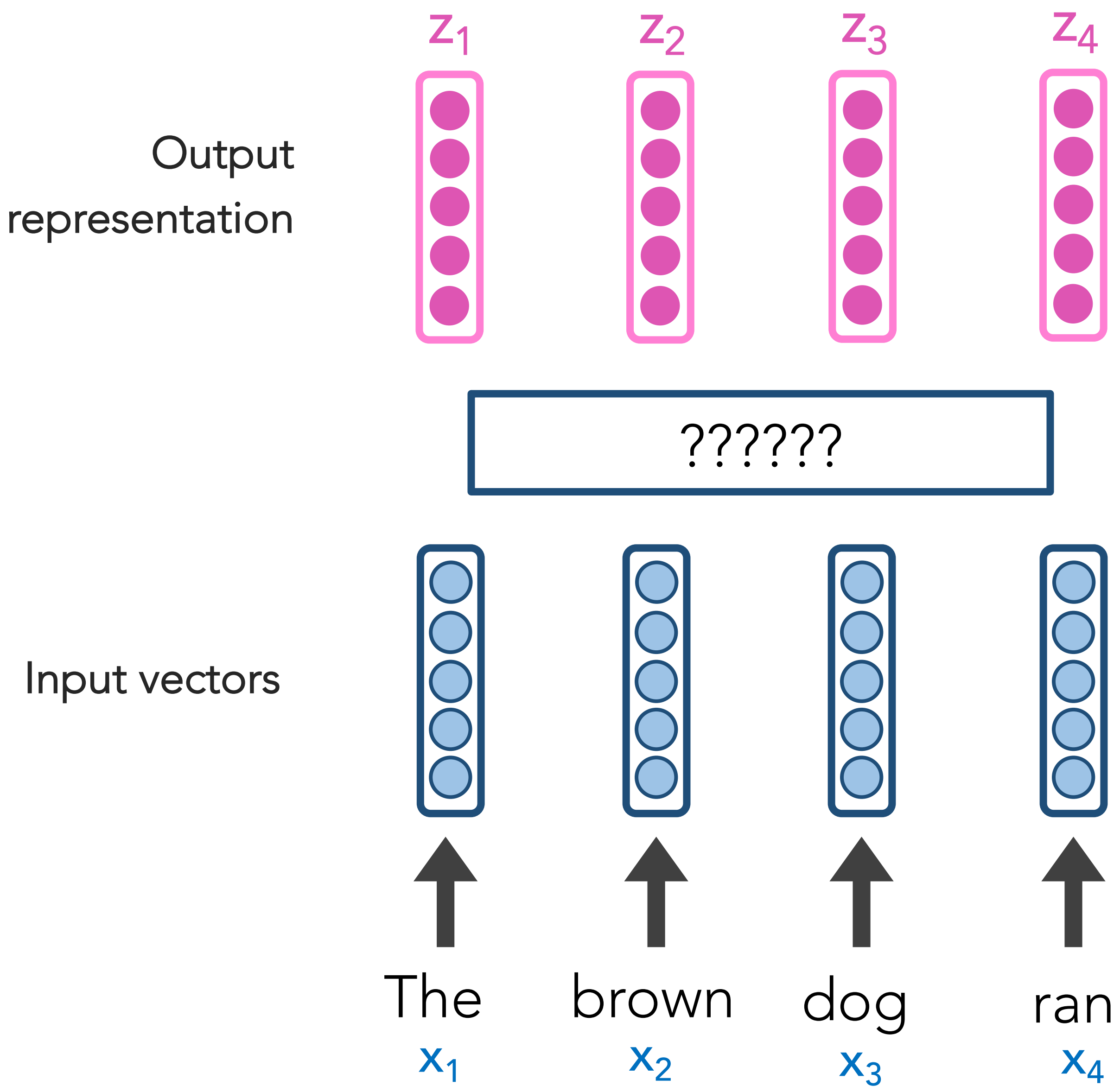
Goals

- Each word in a sequence to be transformed into a rich, abstract **representation** (context embedding) based on the weighted sums of the other words in the same sequence (akin to deep CNN layers)

- Inspired by Attention, we want each word to determine, "how much should I be influenced by each of my neighbors"

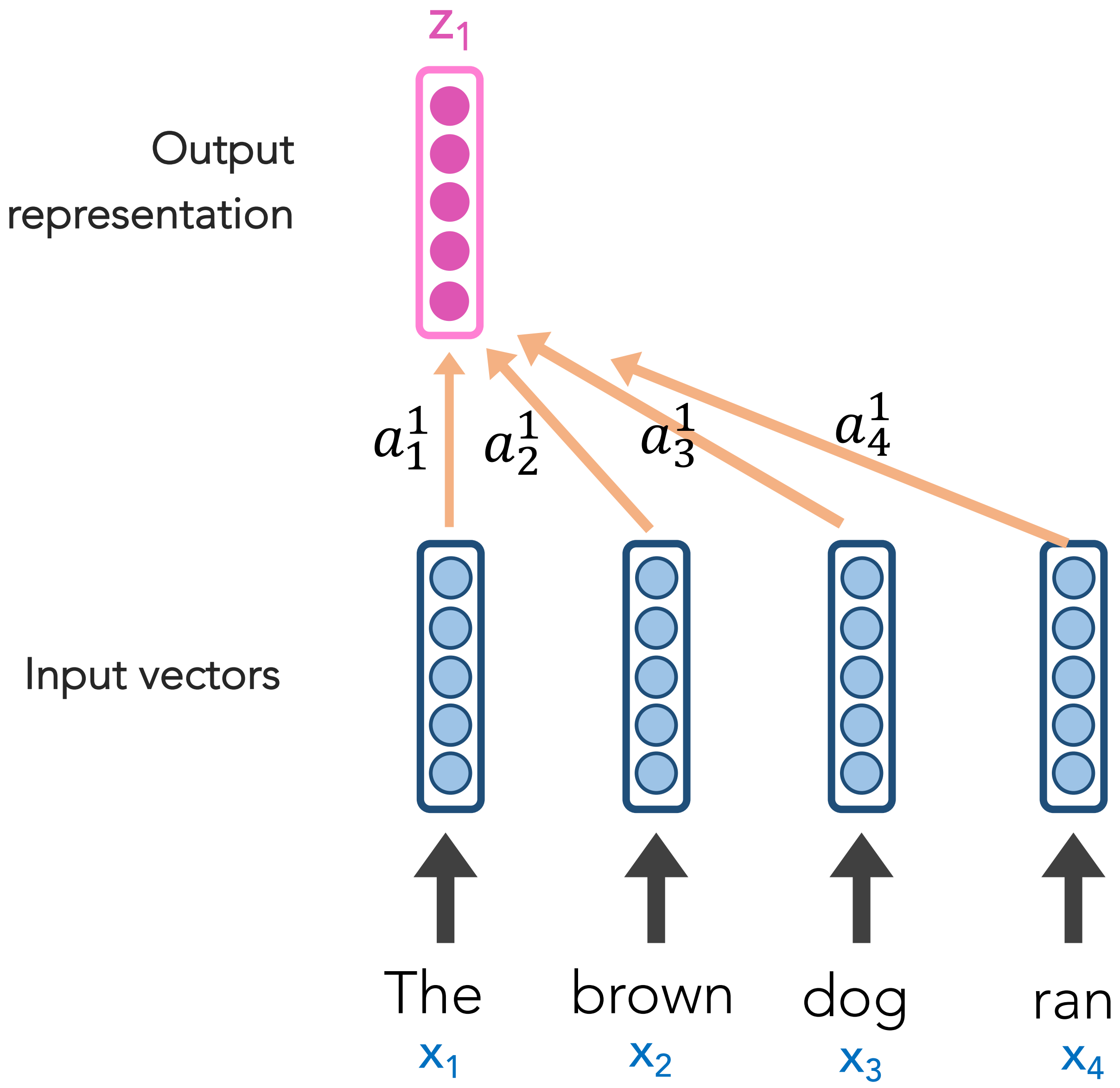
- Want positionality

Self-Attention



Self-Attention's goal is to create great representations, z_i, of the input

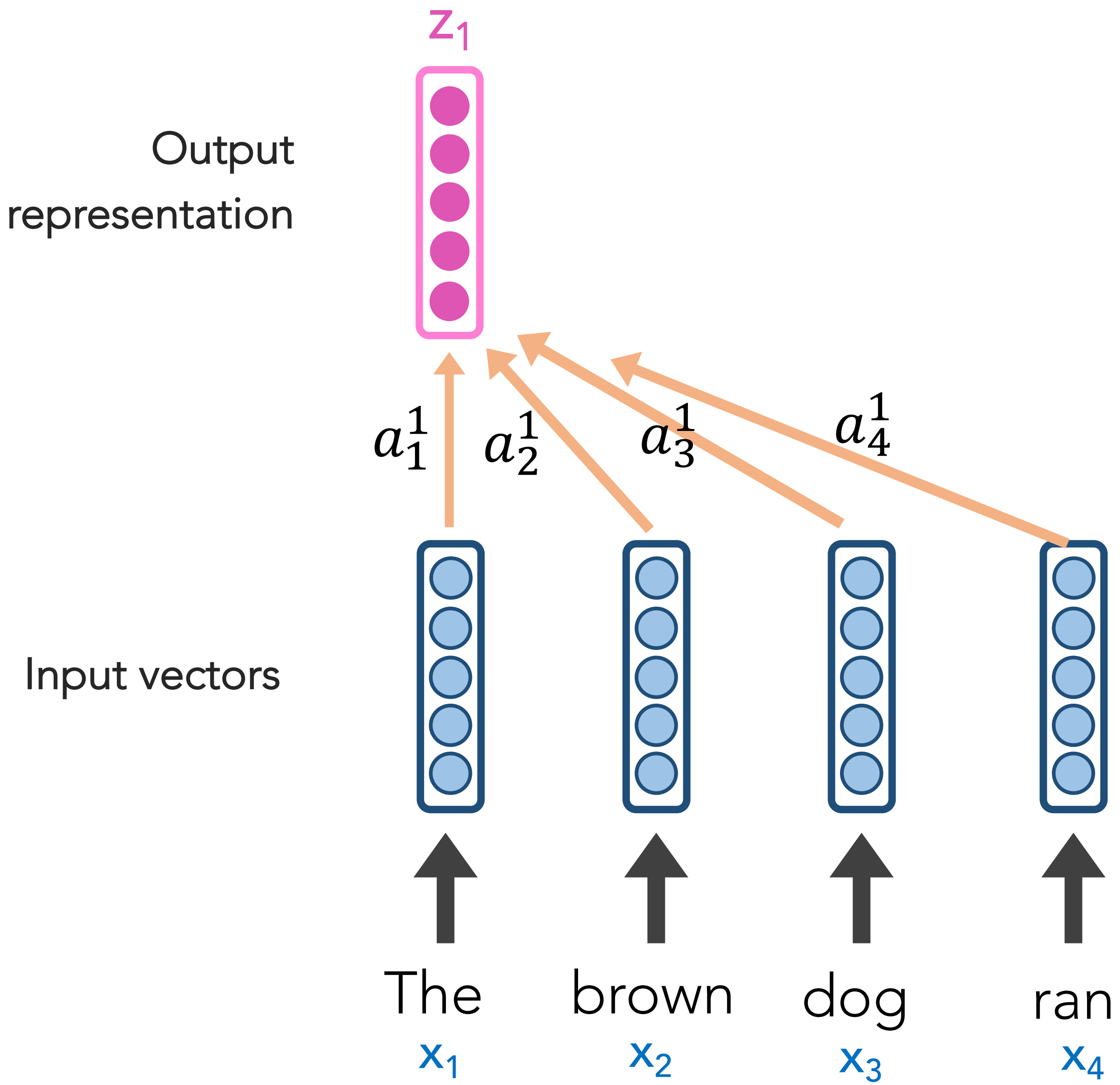
Self-Attention



Self-Attention's goal is to create great representations, z_i, of the input

z_1 will be based on a weighted contribution of x_1, x_2, x_3, x_4

Self-Attention

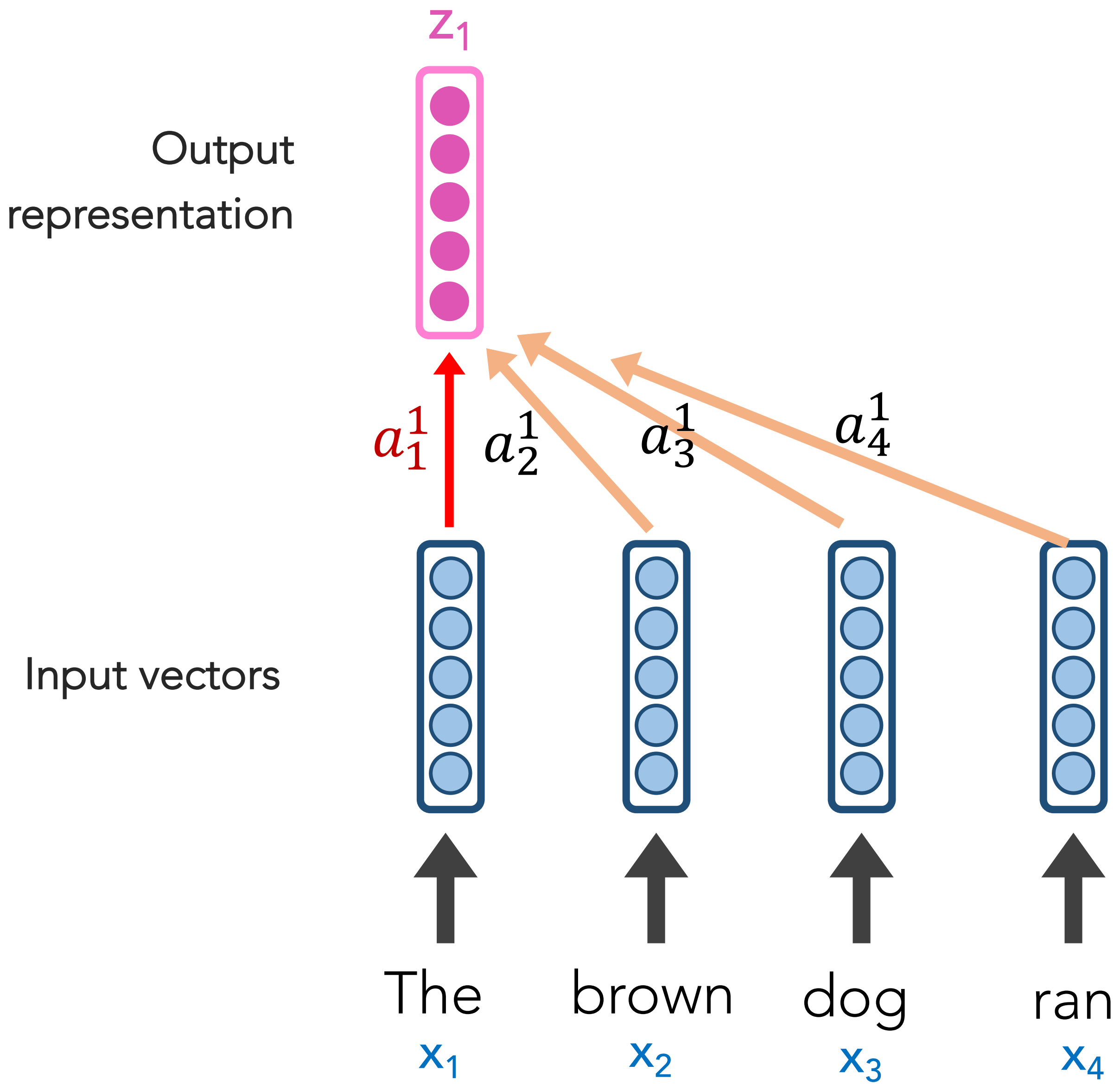


Self-Attention's goal is to create great representations, z_i, of the input

z_1 will be based on a weighted contribution of x_1, x_2, x_3, x_4

a_i^1 is "just" a weight. More is happening under the hood, but it's effectively weighting *versions* of x_1, x_2, x_3, x_4

Self-Attention



Under the hood, each x_i has 3 small, associated vectors.

For example, x_1 has:

- Query $\mathbf{q}_i$
- Key $\mathbf{k}_i$
- Value $\mathbf{v}_i$

Self-Attention

Step 1: Our Self-Attention Head has just 3 weight matrices W_q, W_k, W_v in total. These same 3 weight matrices are multiplied by each x_i to create all vectors:

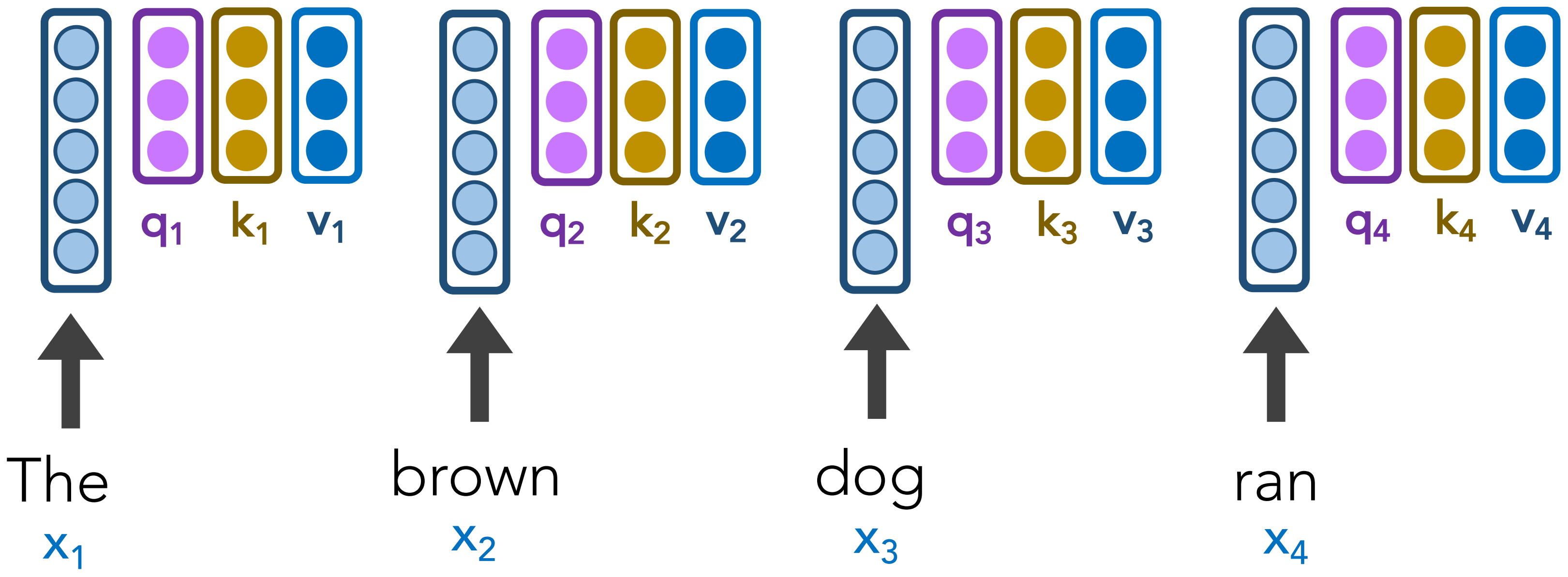
$$q_i = w_q x_i$$

$$k_i = w_k x_i$$

$$v_i = w_v x_i$$

Under the hood, each x_i has 3 small, associated vectors. For example, x_1 has:

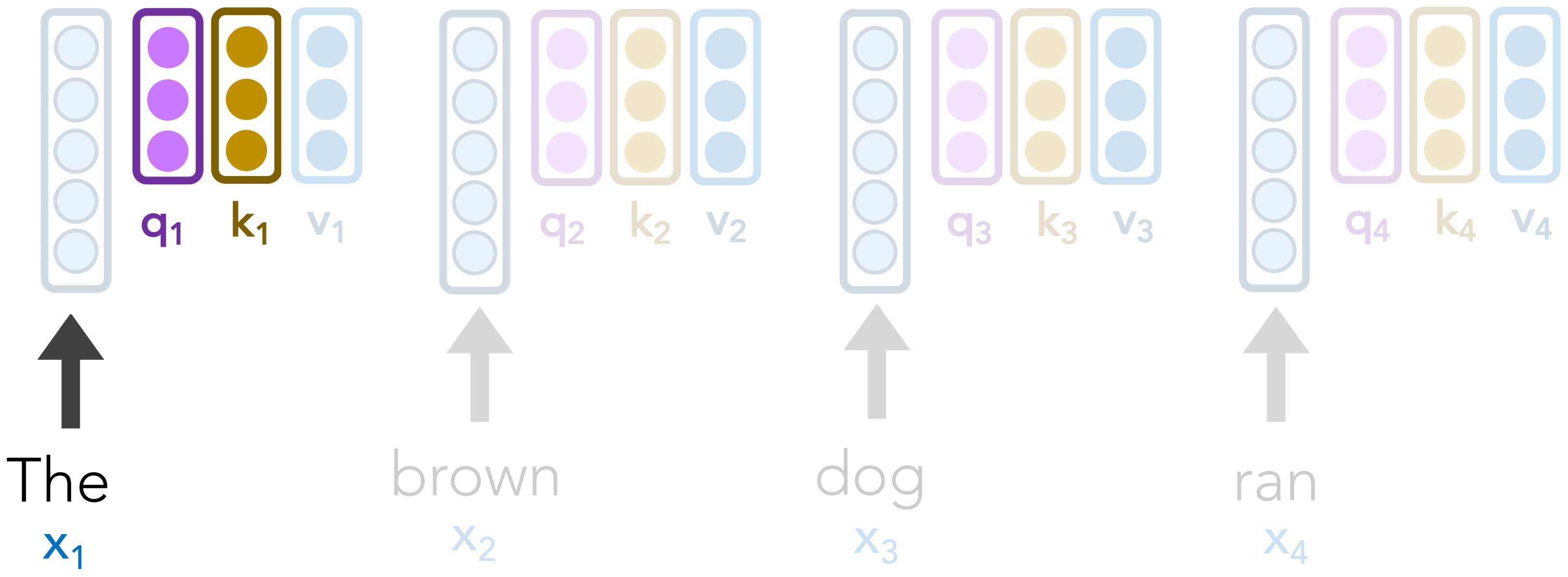
- Query $\mathbf{q_1}$
- Key $\mathbf{k_1}$
- Value $\mathbf{v_1}$



Self-Attention

Step 2: For word x_1, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_1 = q_1 \cdot k_1 = 112$

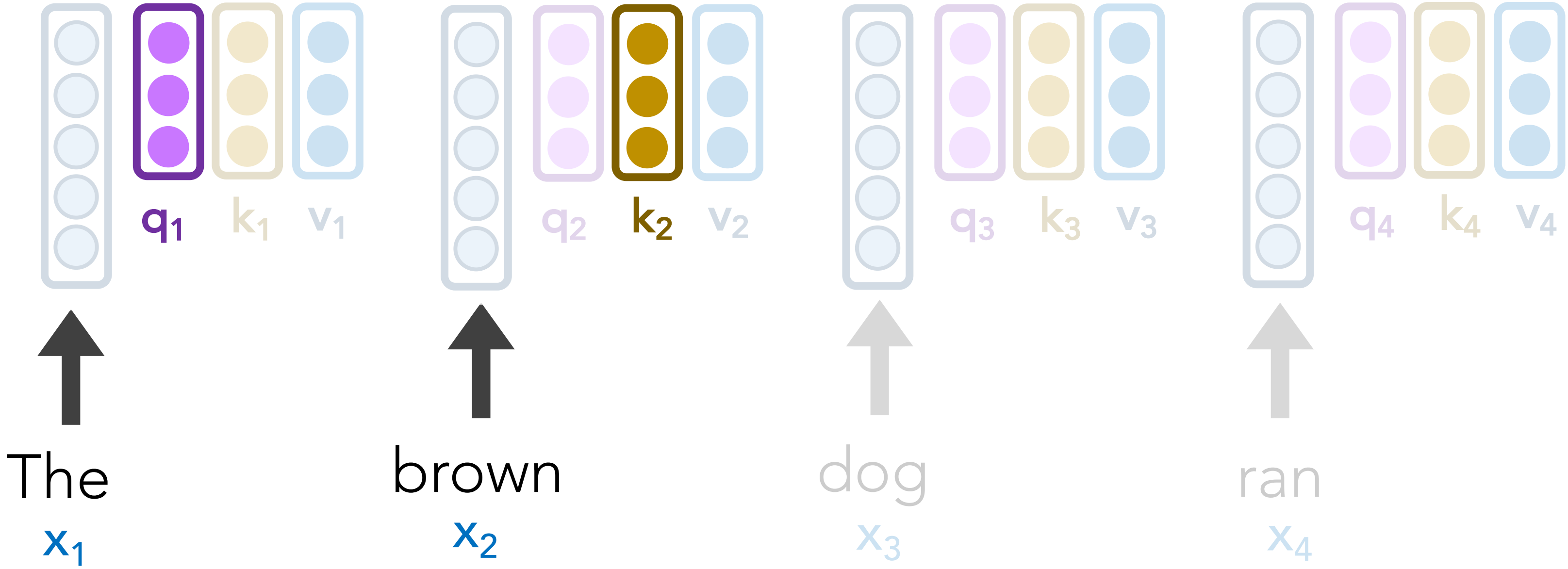


Self-Attention

Step 2: For word x_1, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



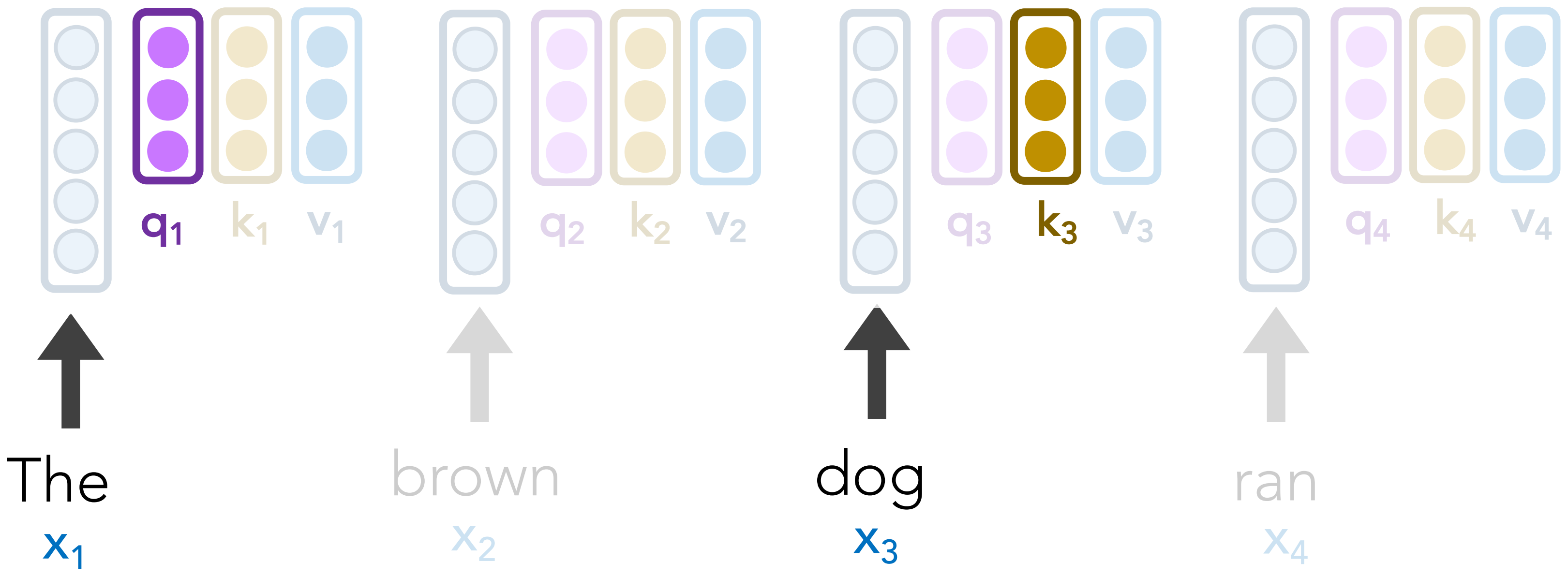
Self-Attention

Step 2: For word x_1, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



Self-Attention

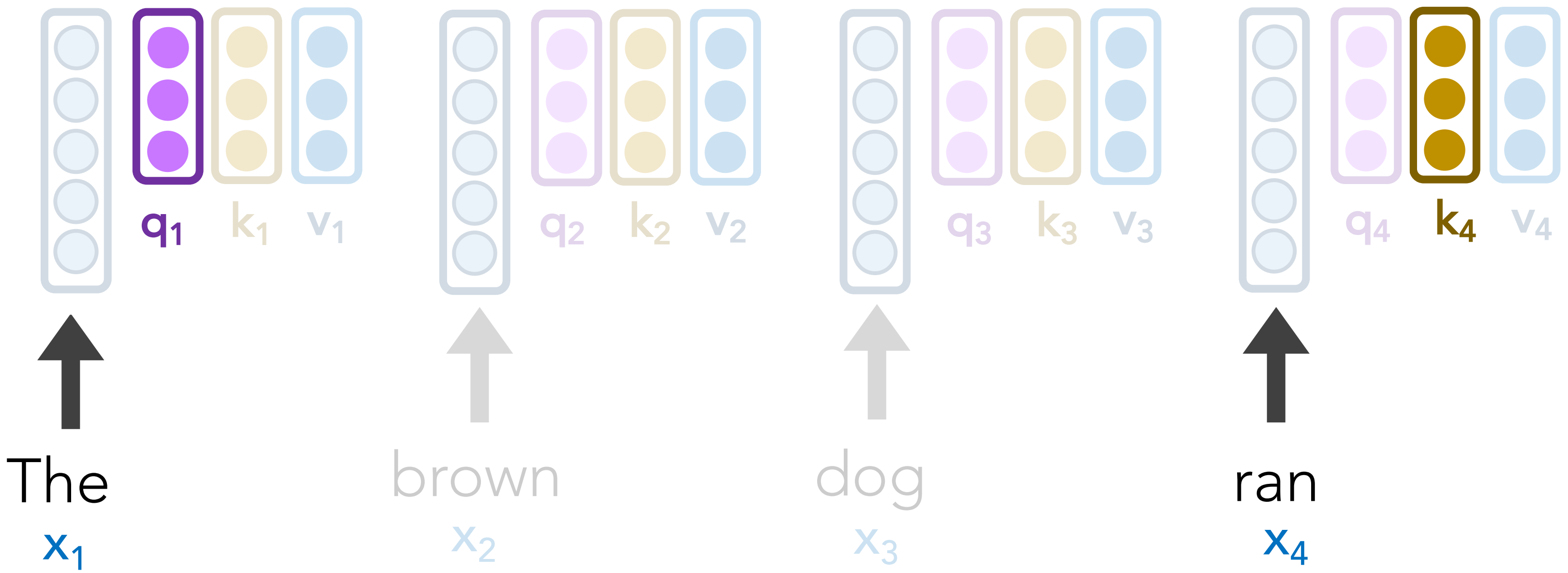
Step 2: For word x_1, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_4 = q_1 \cdot k_4 = 8$

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



Self-Attention

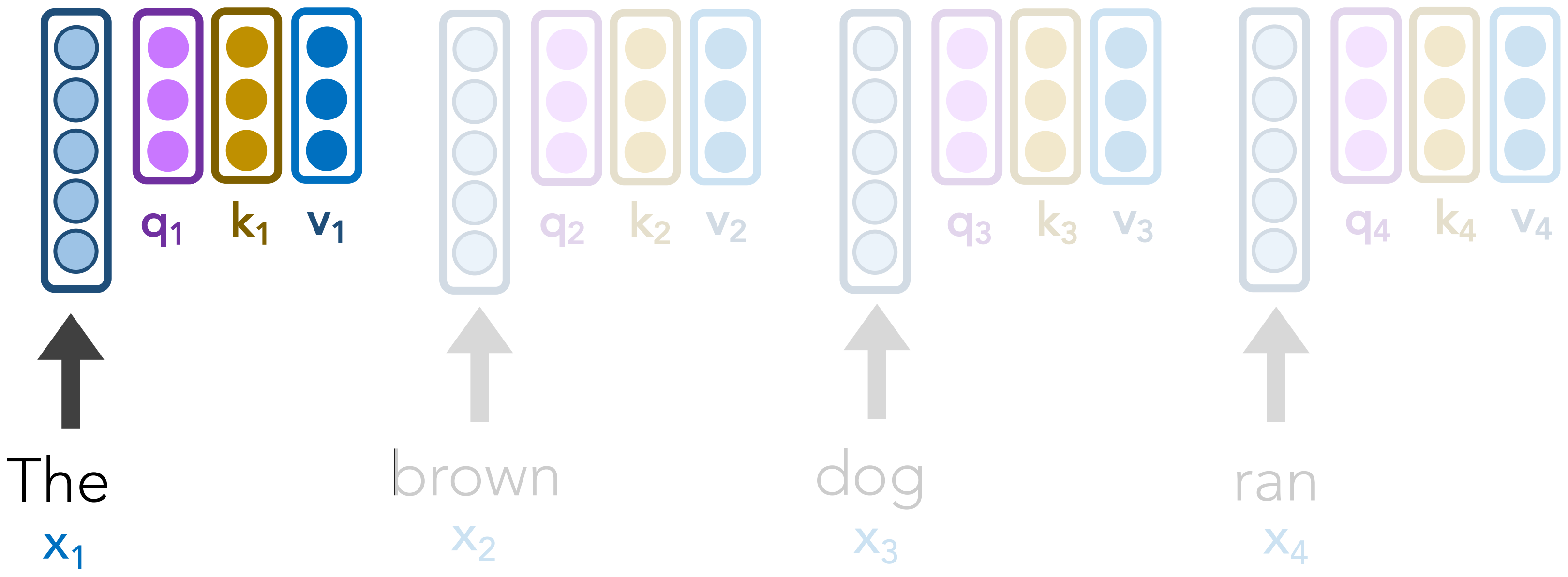
Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_1 \cdot k_4 = 8$ $\quad a_4 = \sigma(s_4/8) = 0$

$s_3 = q_1 \cdot k_3 = 16$ $\quad a_3 = \sigma(s_3/8) = .01$

$s_2 = q_1 \cdot k_2 = 96$ $\quad a_2 = \sigma(s_2/8) = .12$

$s_1 = q_1 \cdot k_1 = 112$ $\quad a_1 = \sigma(s_1/8) = .87$



Self-Attention

Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_1 \cdot k_4 = 8$

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$

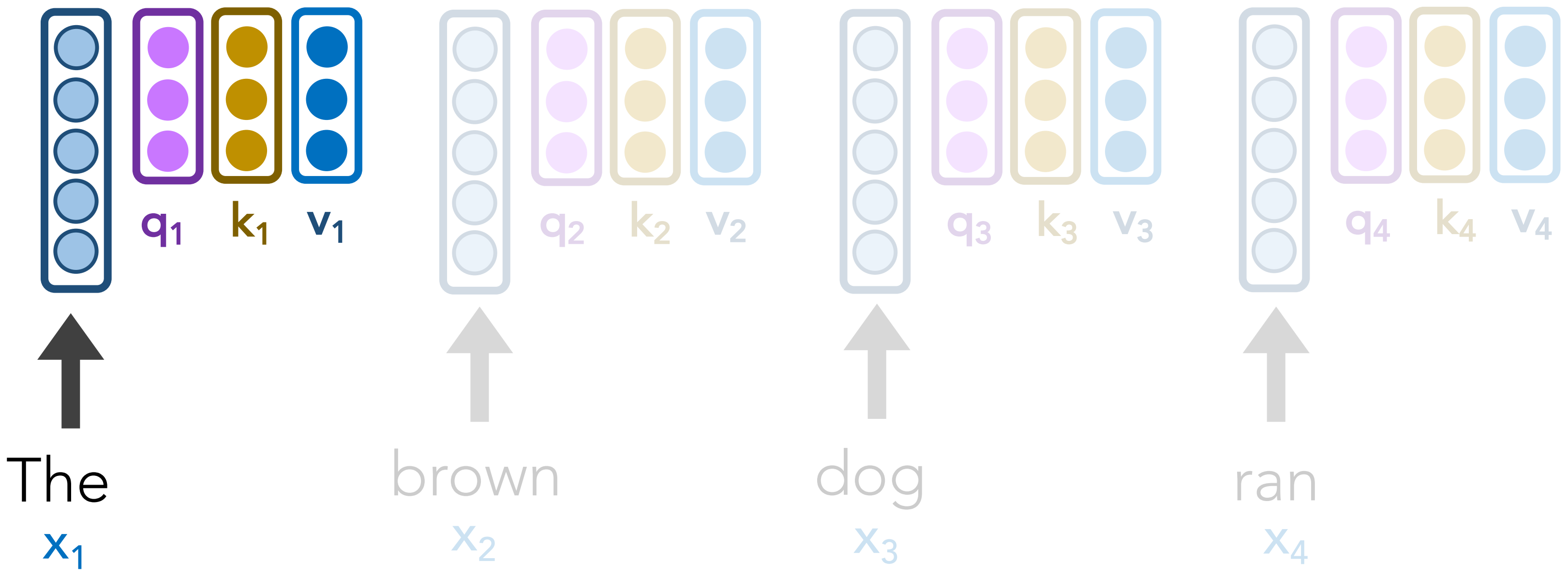
$a_4 = \sigma(s_4/8) = 0$

$a_3 = \sigma(s_3/8) = .01$

$a_2 = \sigma(s_2/8) = .12$

$a_1 = \sigma(s_1/8) = .87$

Instead of these a_i values directly weighting our original x_i word vectors, they directly weight our v_i vectors.



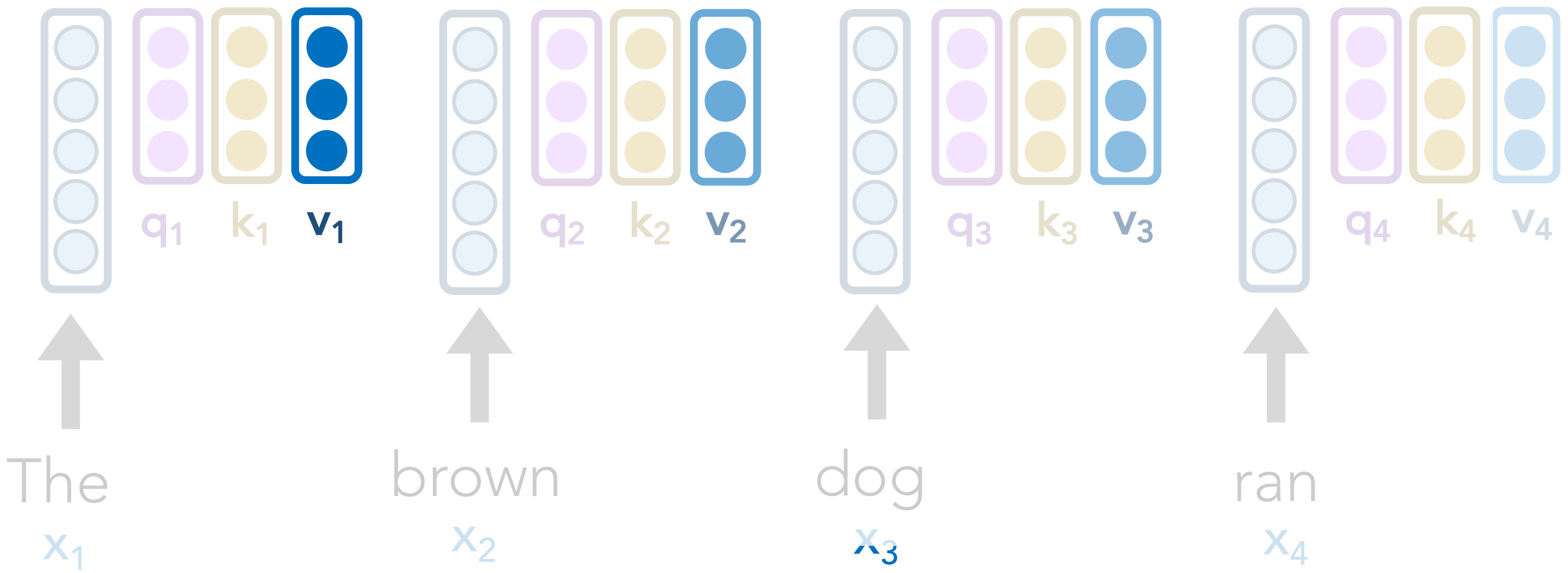
Self-Attention

Step 4: Let's weight our v_i vectors and simply sum them up!

z_1

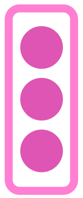
$$z_1 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

$$= 0.87 \cdot v_1 + 0.12 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$

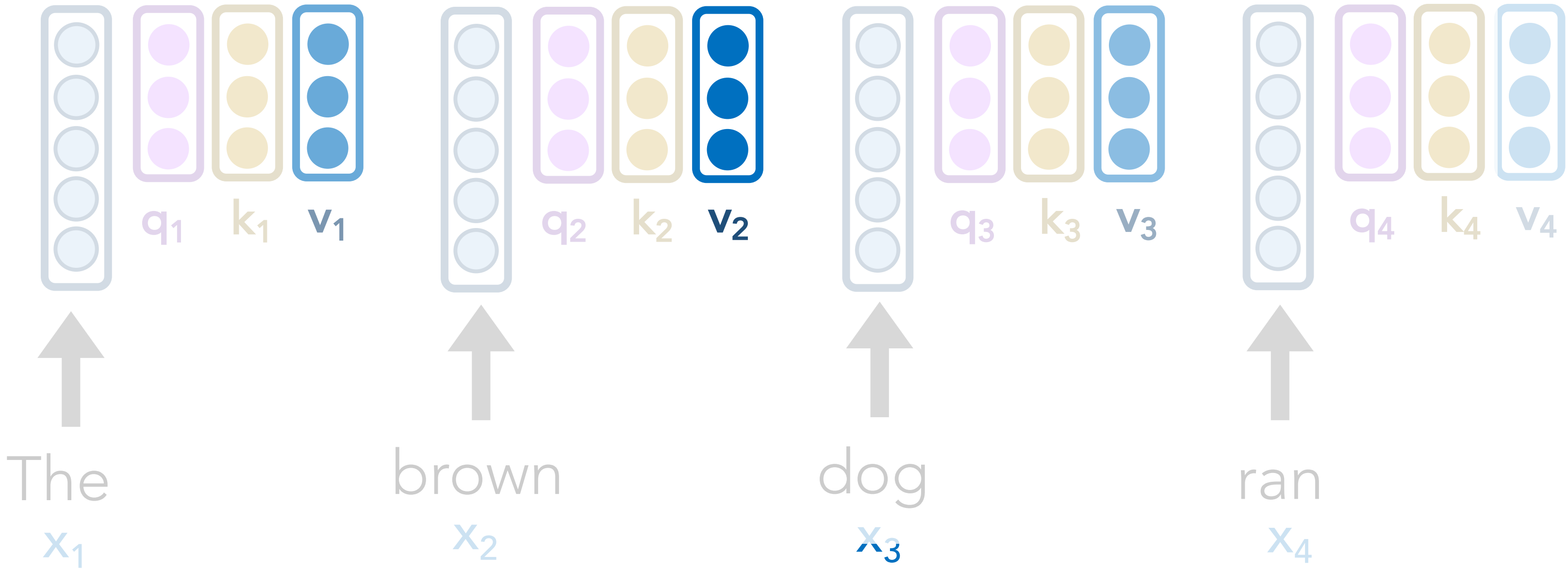


Self-Attention

Step 5: We repeat this for all other words, yielding us with great, new z_i representations!

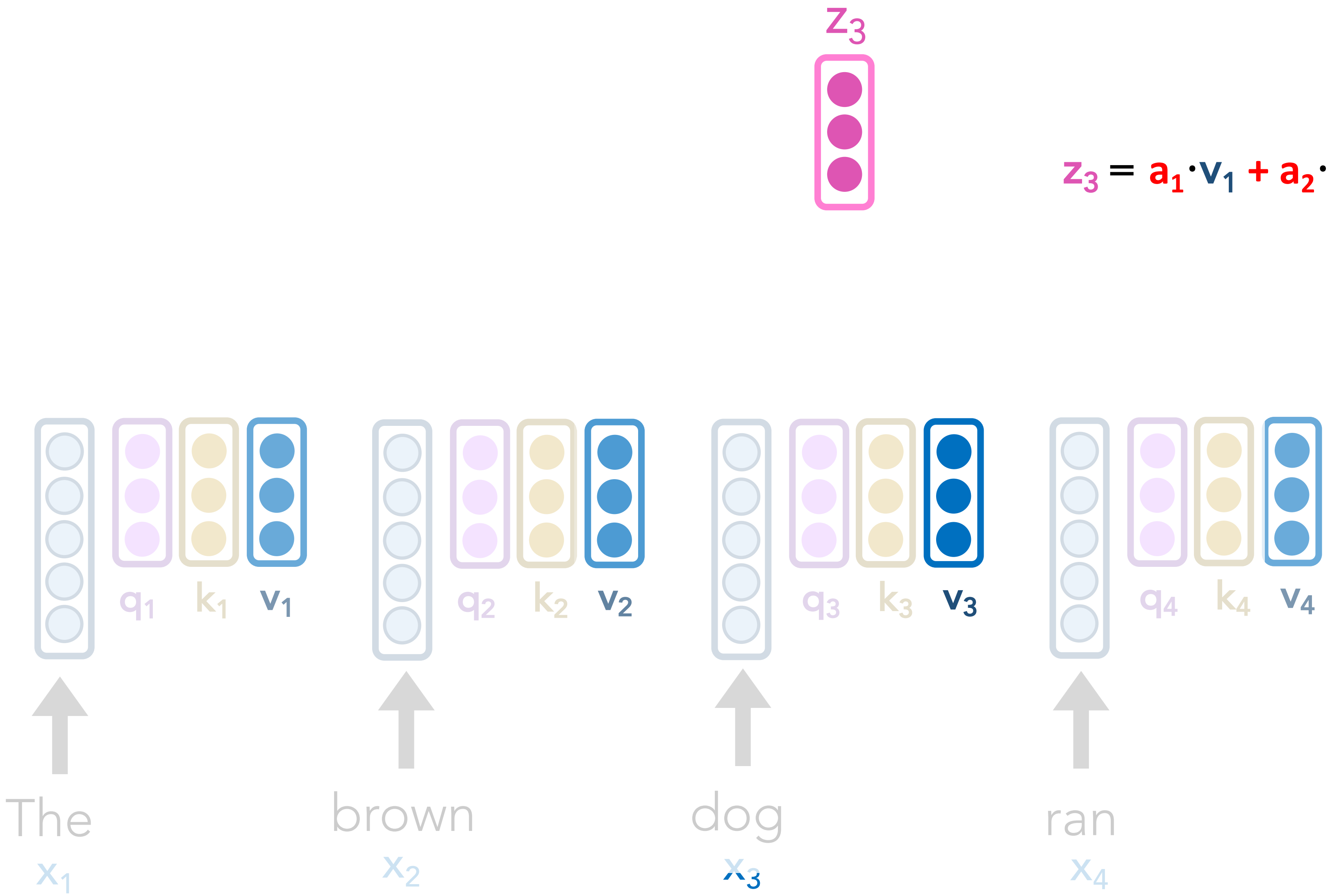


$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$



Self-Attention

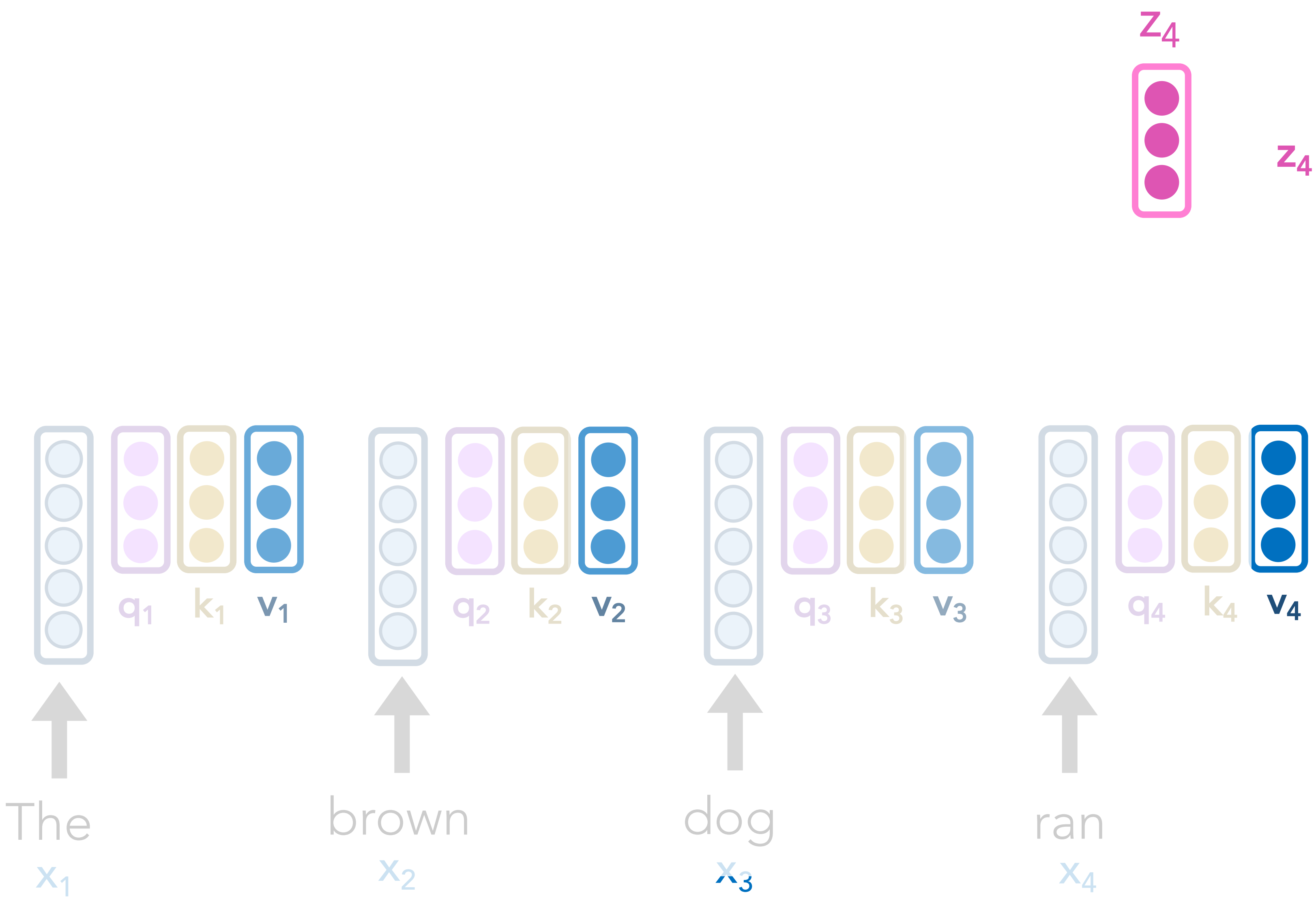
Step 5: We repeat this for all other words, yielding us with great, new z_i representations!



$$z_3 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

Self-Attention

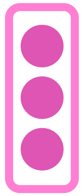
Step 5: We repeat this for all other words, yielding us with great, new z_i representations!



$z_4 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$

Let's illustrate another example:

z_2



$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

Remember, we use the same 3 weight matrices W_q, W_k, W_v as we did for computing z_1.

This gives us q_2, k_2, v_2

Self-Attention

Step 1: Our Self-Attention Head I has just 3 weight matrices W_q, W_k, W_v in total. These same 3 weight matrices are multiplied by each x_i to create all vectors:

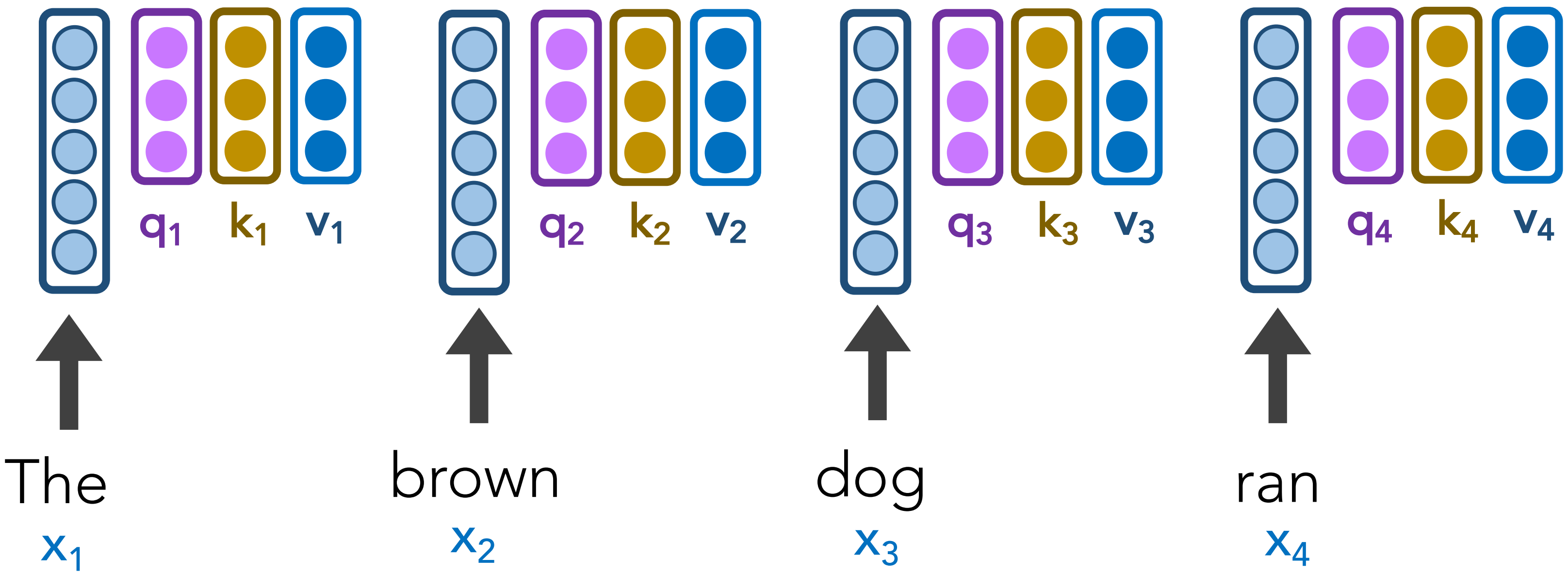
$$q_i = w_q x_i$$

$$k_i = w_k x_i$$

$$v_i = w_v x_i$$

Under the hood, each x_i has 3 small, associated vectors. For example, x_1 has:

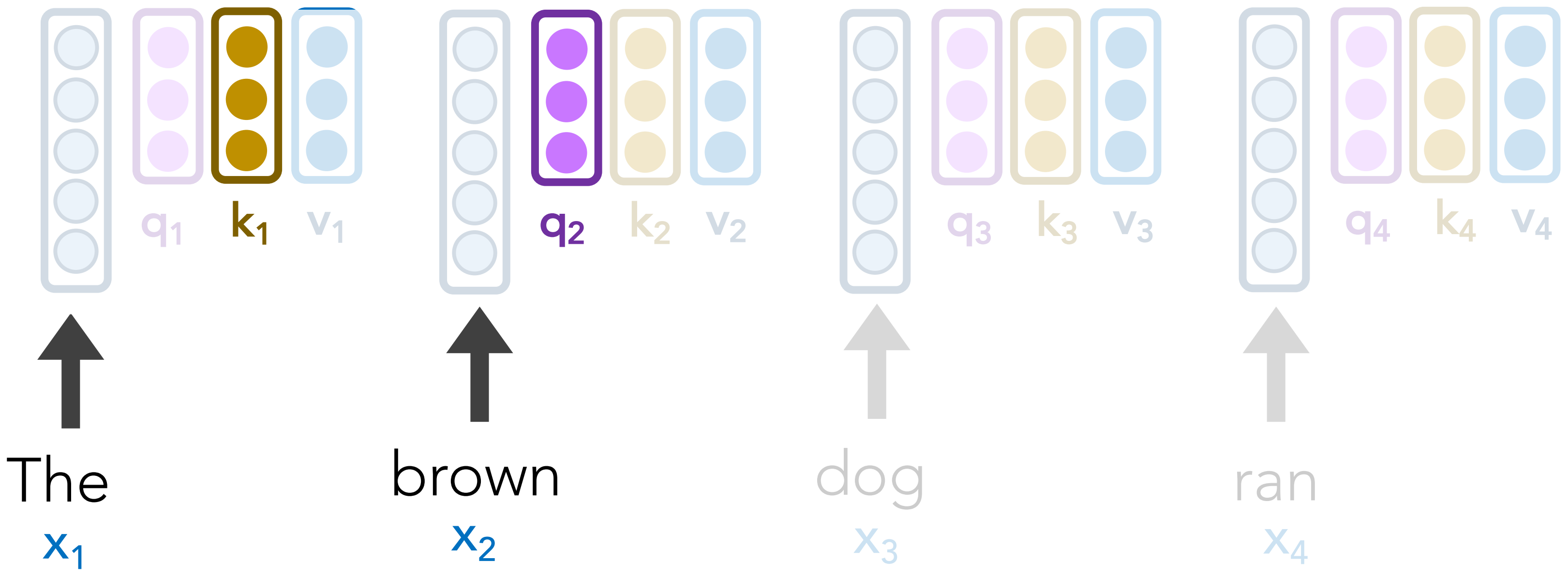
- Query $\mathbf{q_1}$
- Key $\mathbf{k_1}$
- Value $\mathbf{v_1}$



Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_1 = q_2 \cdot k_1 = 92$

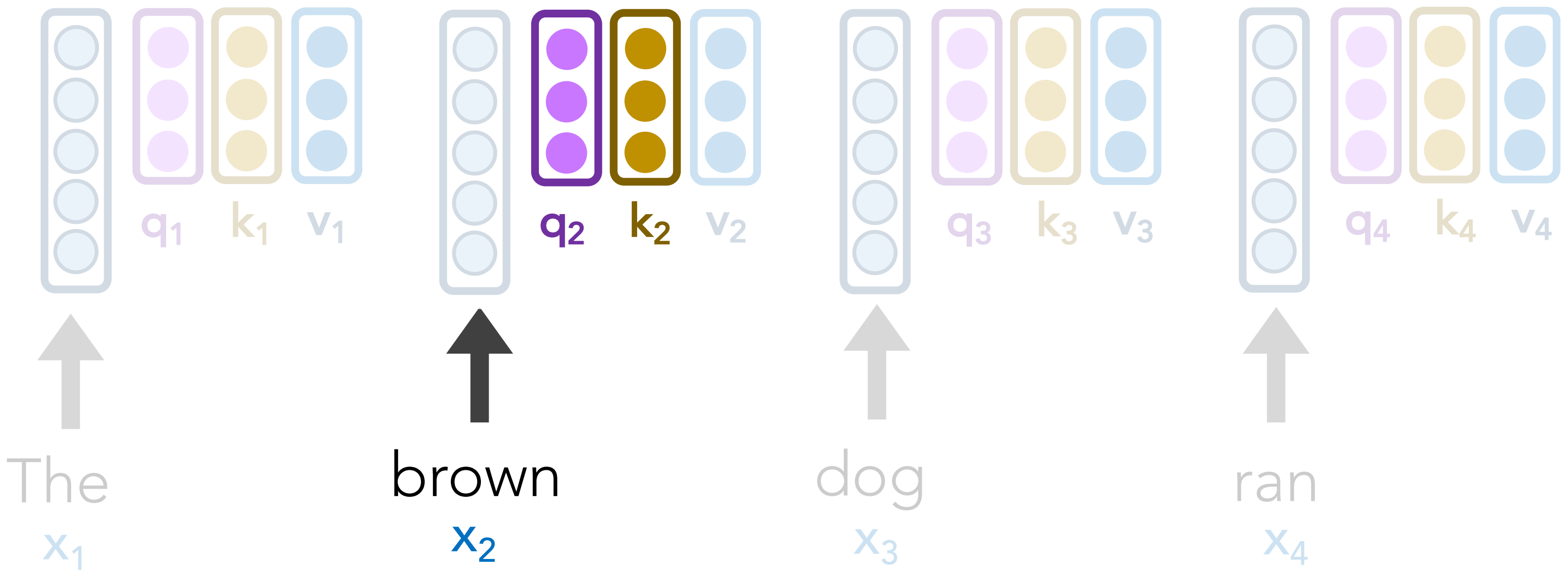


Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



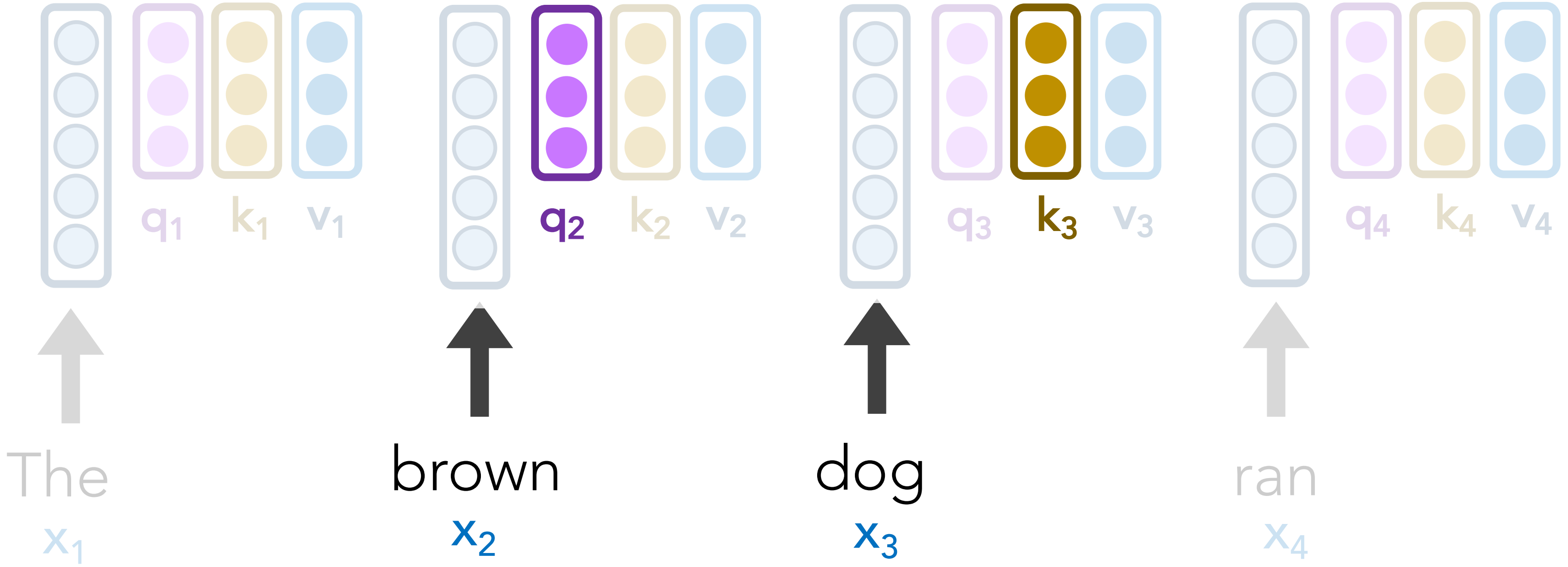
Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



Self-Attention

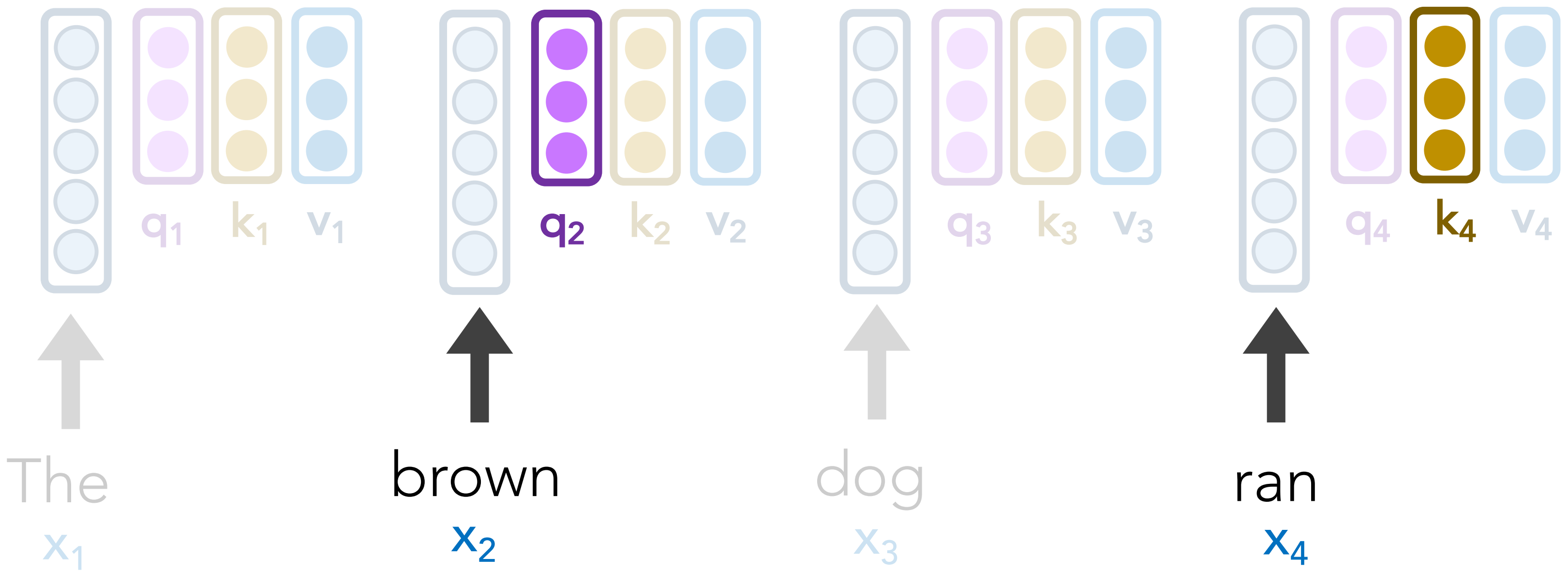
Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_4 = q_2 \cdot k_4 = 8$

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



Self-Attention

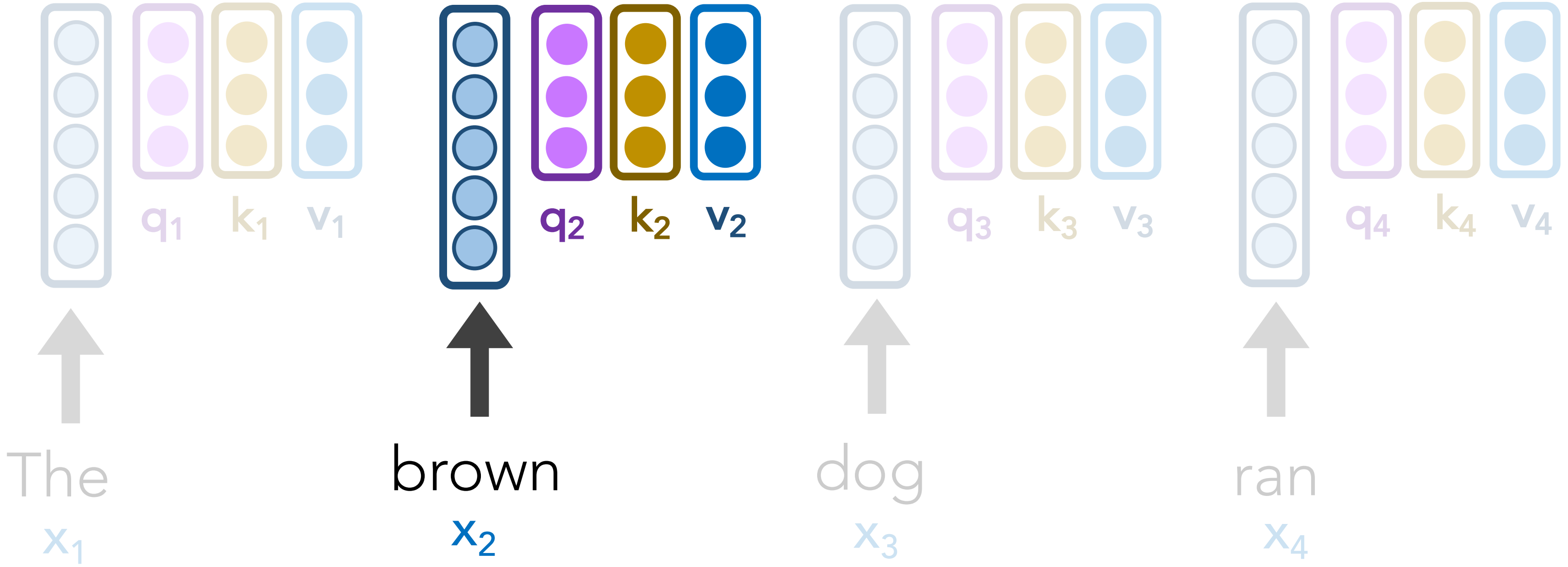
Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_2 \cdot k_4 = 8$ $\quad a_4 = \sigma(s_4/8) = 0$

$s_3 = q_2 \cdot k_3 = 22$ $\quad a_3 = \sigma(s_3/8) = .01$

$s_2 = q_2 \cdot k_2 = 124$ $\quad a_2 = \sigma(s_2/8) = .91$

$s_1 = q_2 \cdot k_1 = 92$ $\quad a_1 = \sigma(s_1/8) = .08$



Self-Attention

Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

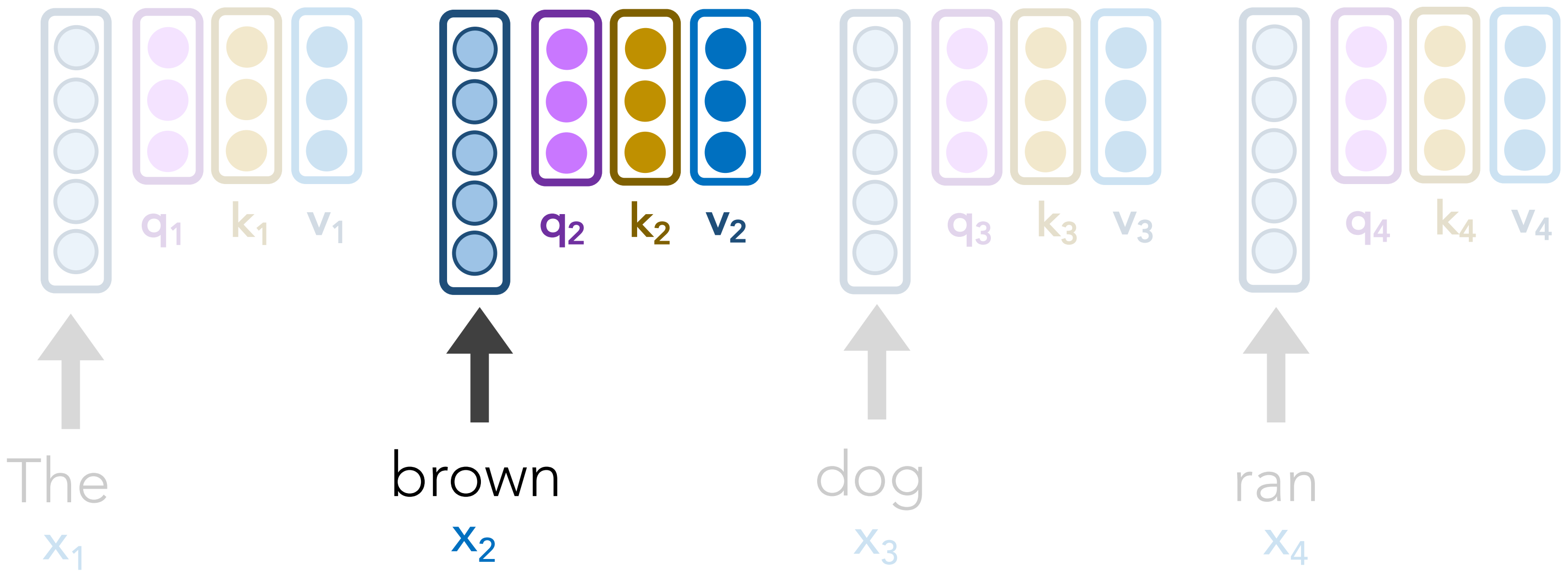
$s_4 = q_2 \cdot k_4 = 8$ $\qquad a_4 = \sigma(s_4/8) = 0$

$s_3 = q_2 \cdot k_3 = 22$ $\qquad a_3 = \sigma(s_3/8) = .01$

$s_2 = q_2 \cdot k_2 = 124$ $\qquad a_2 = \sigma(s_2/8) = .91$

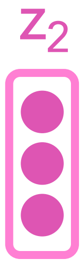
$s_1 = q_2 \cdot k_1 = 92$ $\qquad a_1 = \sigma(s_1/8) = .08$

Instead of these a_i values directly weighting our original x_i word vectors, they directly weight our v_i vectors.



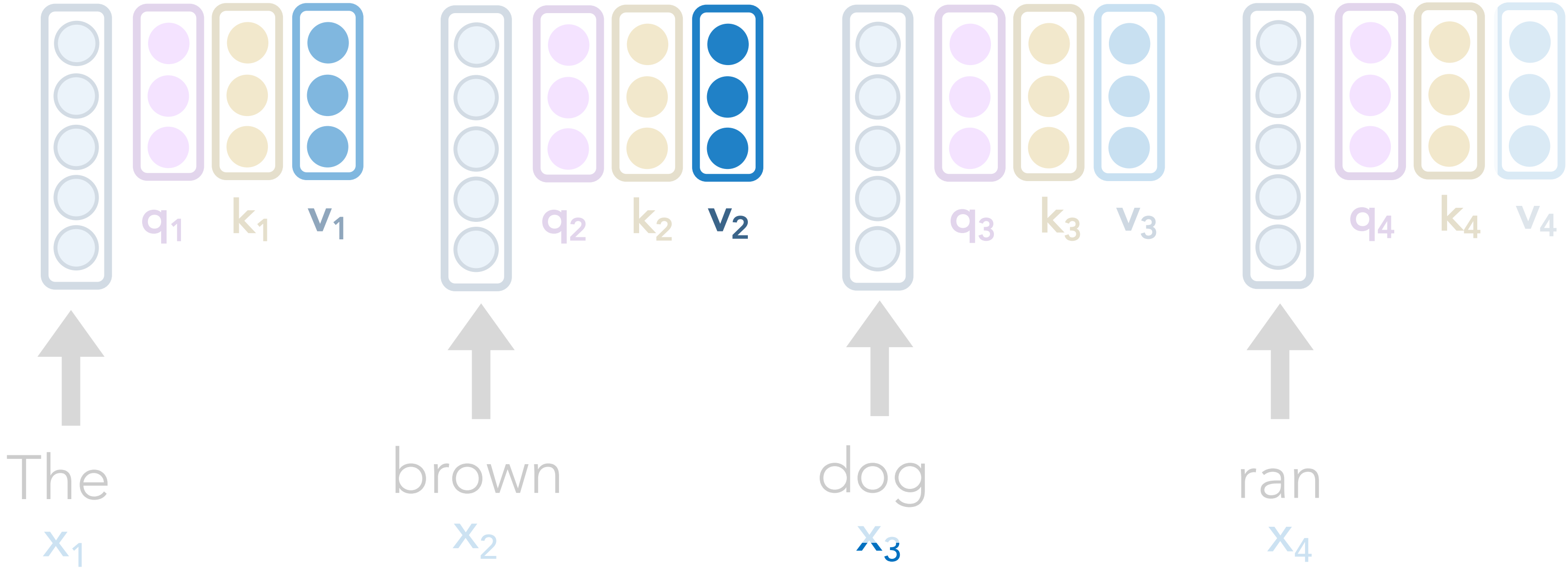
Self-Attention

Step 4: Let's weight our v_i vectors and simply sum them up!



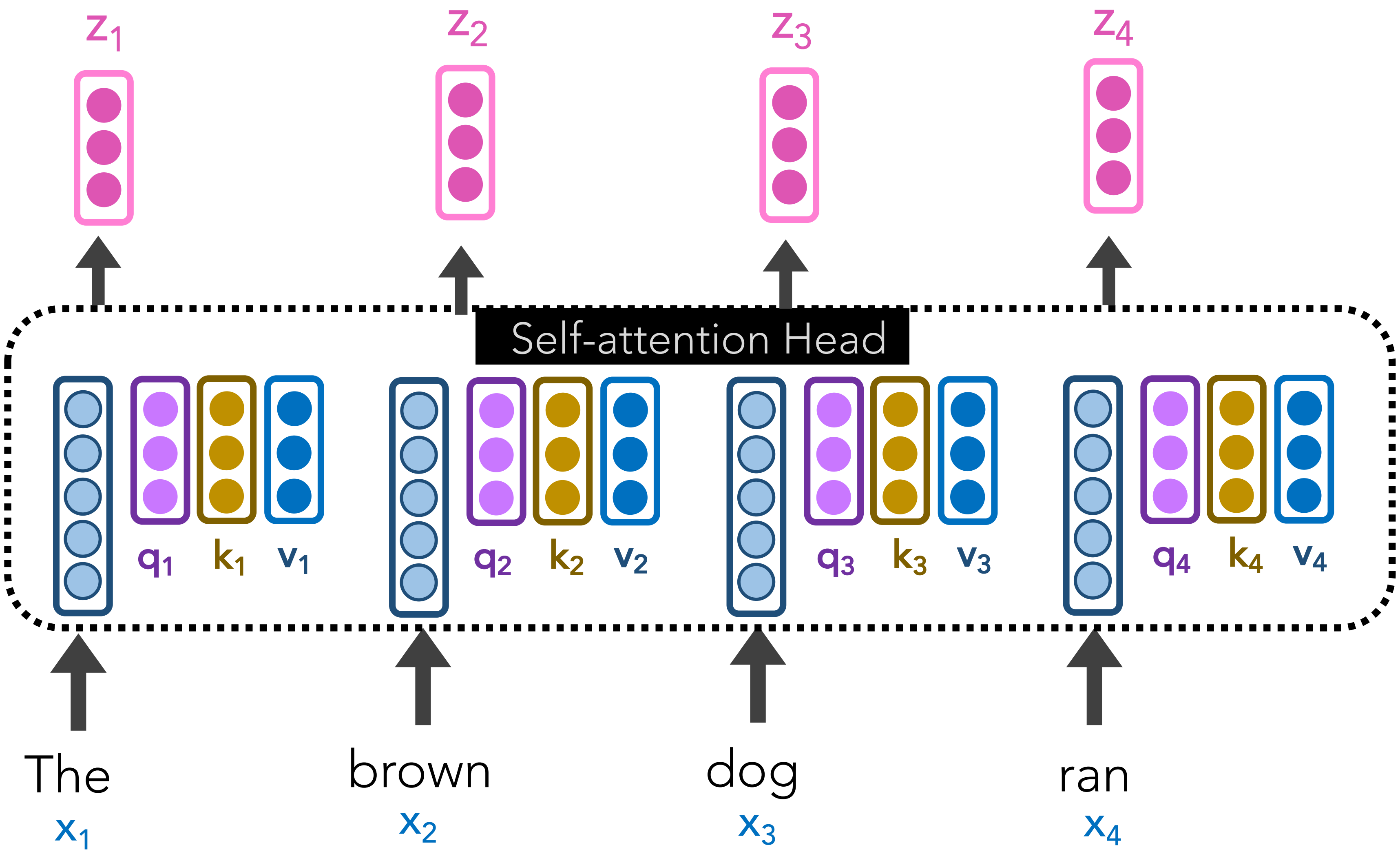
$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

$$= 0.08 \cdot v_1 + 0.91 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



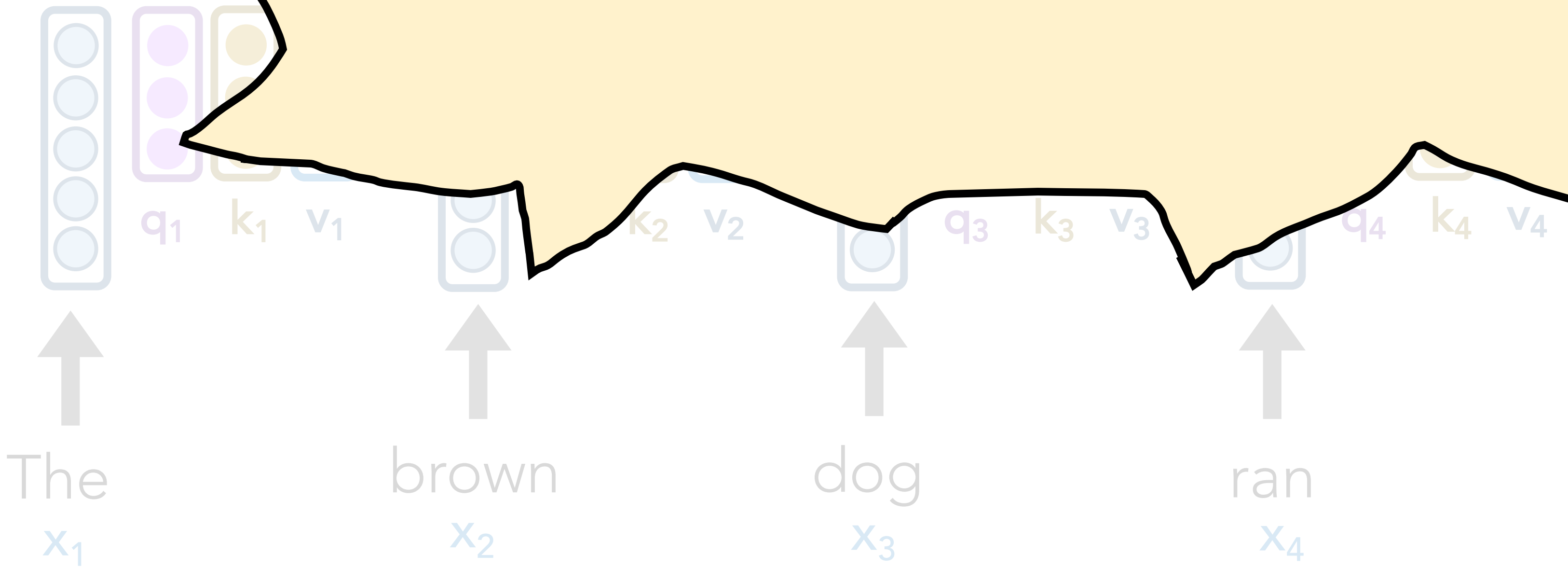
Self-Attention

Tada! Now we have great, new representations z_i via a self-attention head



Takeaway:

Self-Attention is powerful; allows us to create great, context-aware representations



Self-Attention may seem strikingly like Attention in seq2seq models

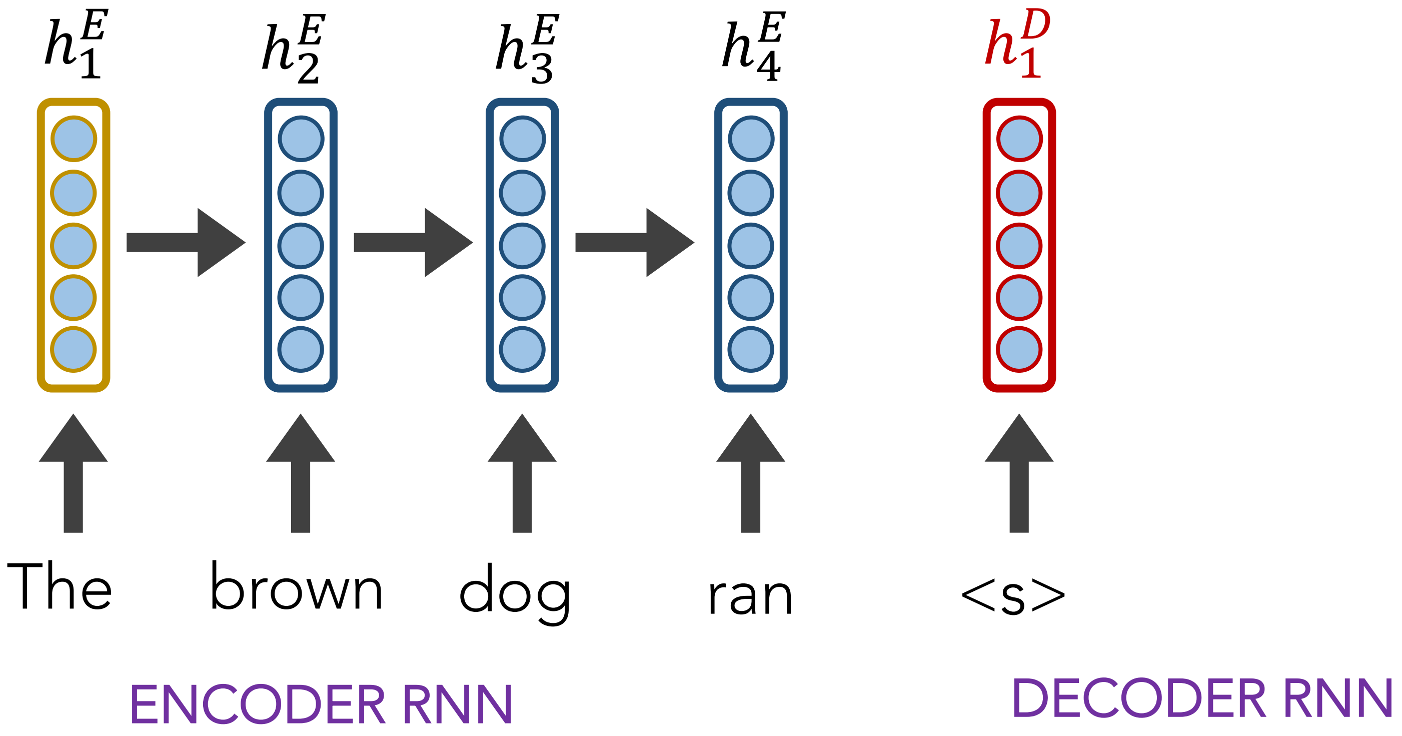
$s_4 = h_1^D * h_4^E$ $\quad a_4 = \sigma(s_4)$

$s_3 = h_1^D * h_3^E$ $\quad a_3 = \sigma(s_3)$

$s_2 = h_1^D * h_2^E$ $\quad a_2 = \sigma(s_2)$

$s_1 = h_1^D * h_1^E$ $\quad a_1 = \sigma(s_1)$

Attention



$s_4 = h_1^D * h_4^E$ $a_4 = \sigma(s_4)$

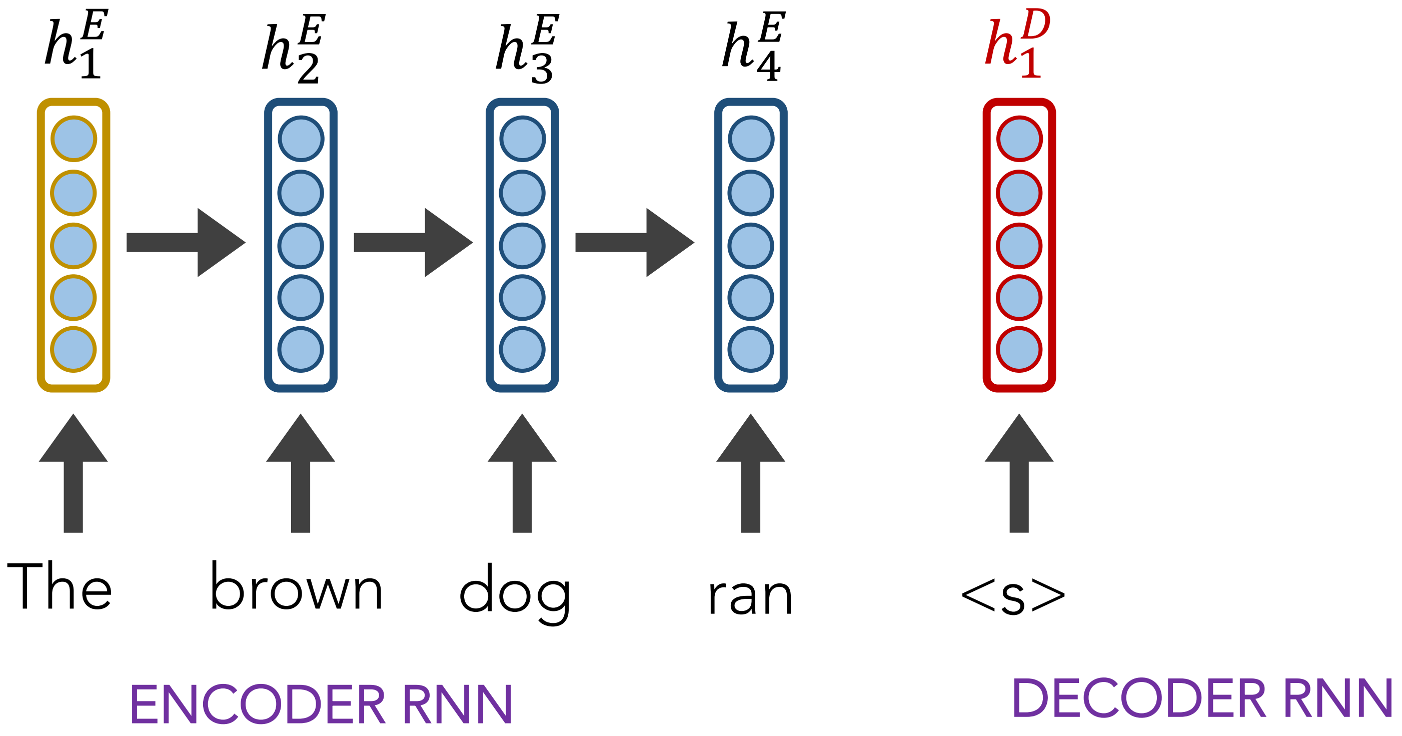
$s_3 = h_1^D * h_3^E$ $a_3 = \sigma(s_3)$

$s_2 = h_1^D * h_2^E$ $a_2 = \sigma(s_2)$

$s_1 = h_1^D * h_1^E$ $a_1 = \sigma(s_1)$

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

Attention



Attention

$s_4 = h_1^D * h_4^E$ $\quad$ $a_4 = \sigma(s_4)$

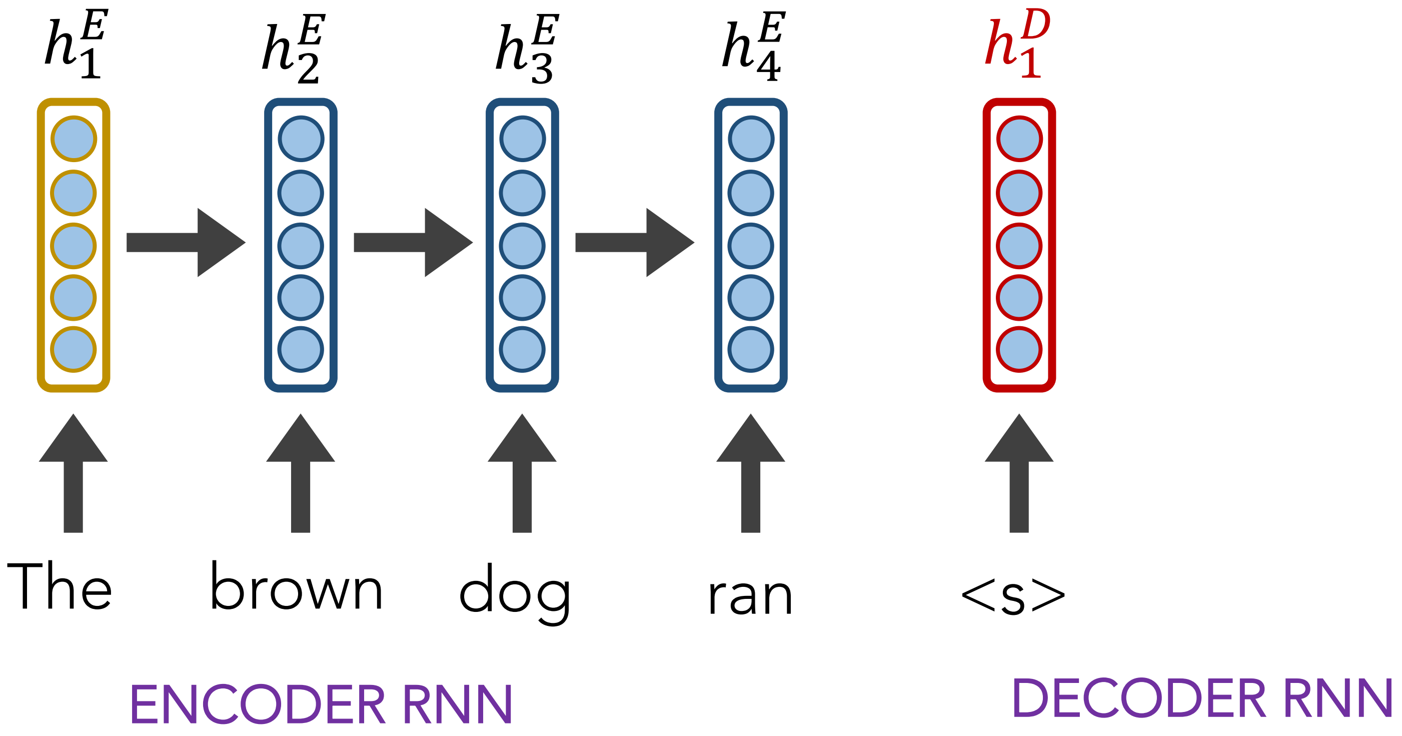
$s_3 = h_1^D * h_3^E$ $\quad$ $a_3 = \sigma(s_3)$

$s_2 = h_1^D * h_2^E$ $\quad$ $a_2 = \sigma(s_2)$

$s_1 = h_1^D * h_1^E$ $\quad$ $a_1 = \sigma(s_1)$

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

$$c_1^D = a_1 \cdot h_1^E + a_2 \cdot h_2^E + a_3 \cdot h_3^E + a_4 \cdot h_4^E$$



$s_4 = q_2 \cdot k_4$ $a_4 = \sigma(s_4/8)$

$s_3 = q_2 \cdot k_3$ $a_3 = \sigma(s_3/8)$

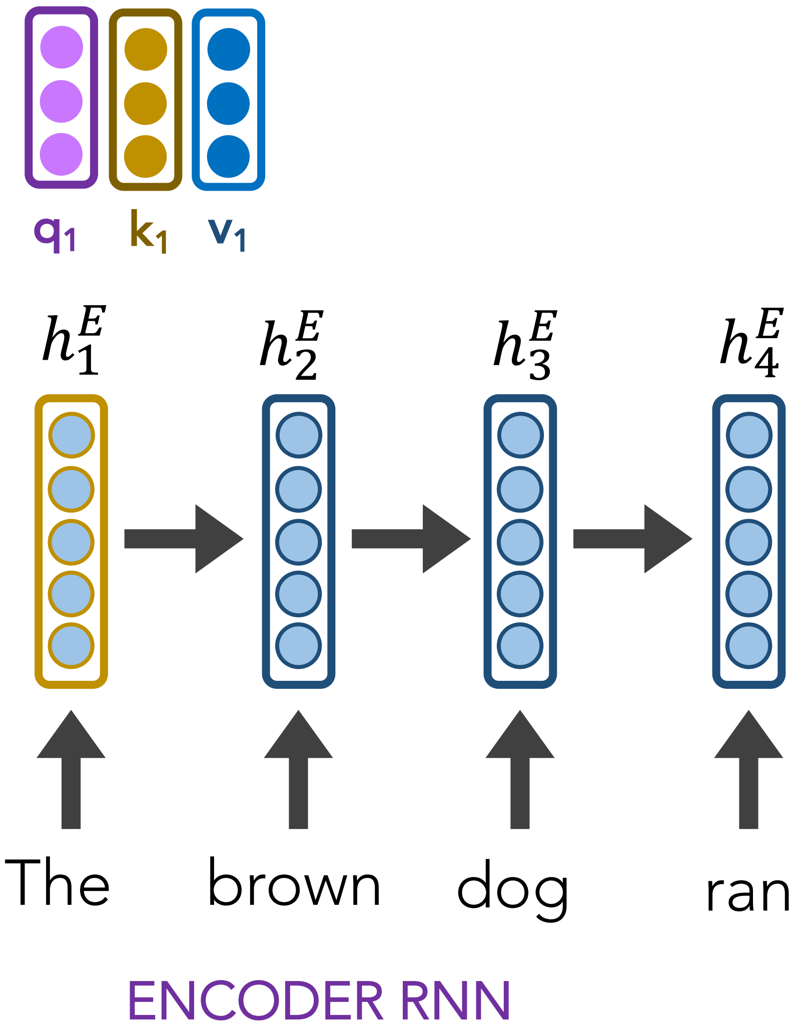
$s_2 = q_2 \cdot k_2$ $a_2 = \sigma(s_2/8)$

$s_1 = q_2 \cdot k_1$ $a_1 = \sigma(s_1/8)$

We multiply each word's value vector by its a_i^1 attention weights to create a better vector z_1

$z_1 = a_1 \cdot v_1^E + a_2 \cdot v_2^E + a_3 \cdot v_3^E + a_4 \cdot v_4^E$

Self-Attention

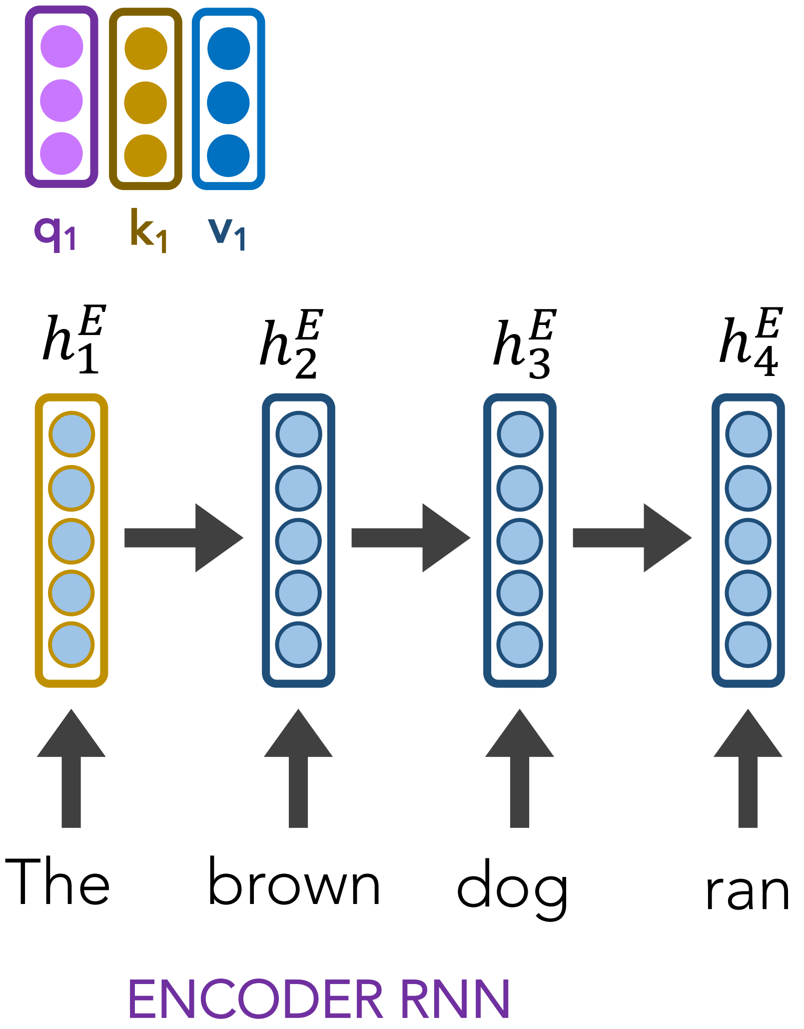


Self-Attention

Self-Attention	Attention	Description
q_i	h_i^D	the probe
k_i	h_i^E	item being compared
v_i	h_i^E	item being weighted

vector by its a_i^1 attention weights to create a better vector z_1

$z_1 = a_1 \cdot v_1^E + a_2 \cdot v_2^E + a_3 \cdot v_3^E + a_4 \cdot v_4^E$



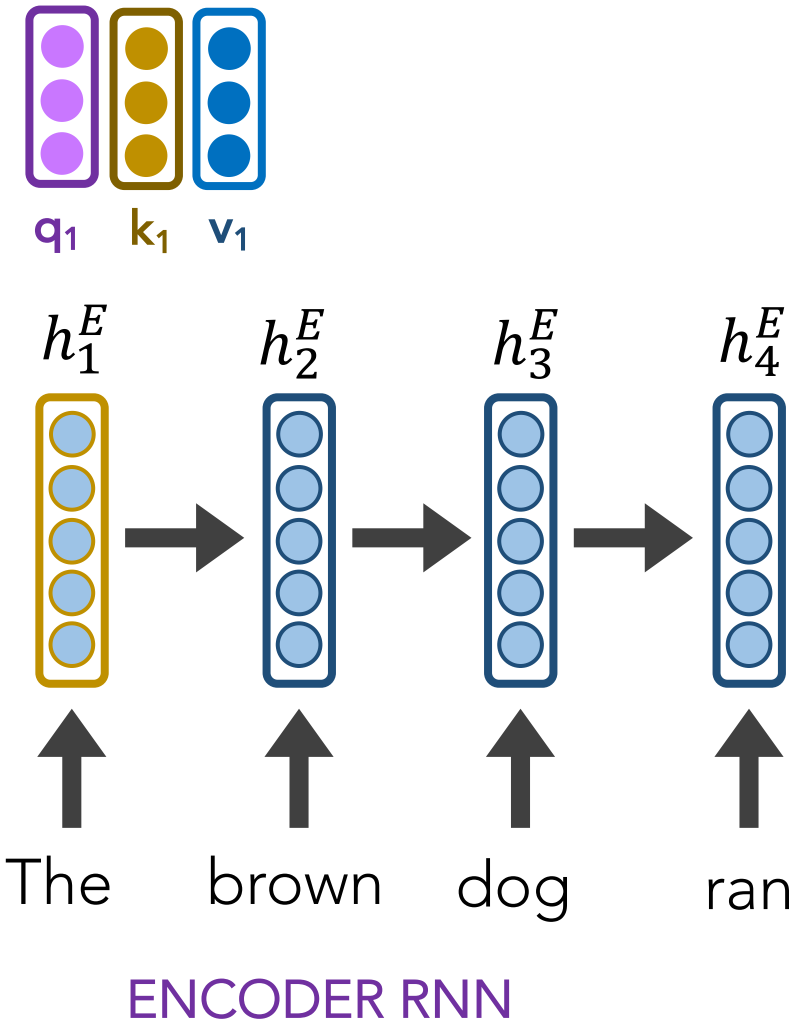
Self-Attention

Self-Attention	Attention	Description
q_i	h_i^D	the probe
k_i	h_i^E	item being compared
v_i	h_i^E	item being weighted

All of these are like surrogates/proxies/abstractions.

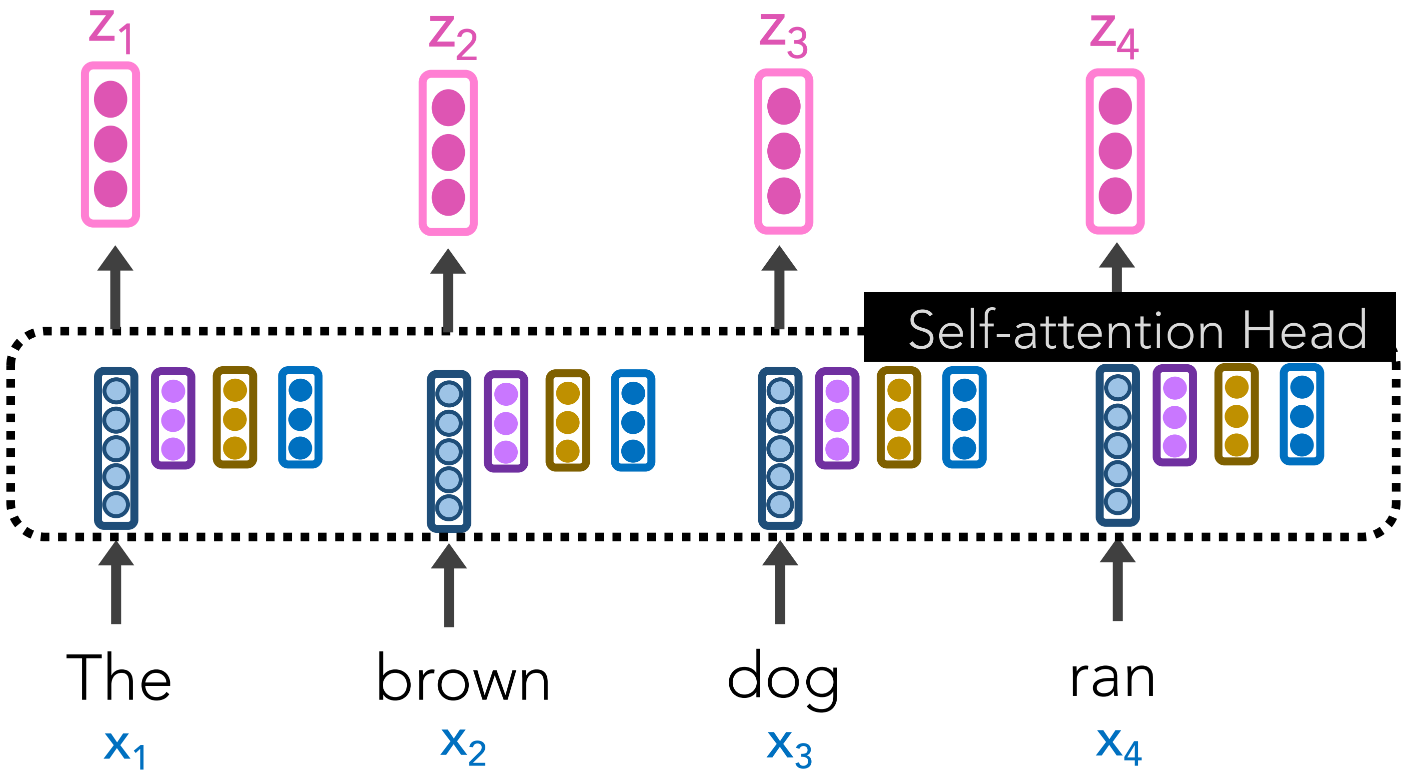
This provides flexibility and fewer constraints.

More room for rich abstractions.

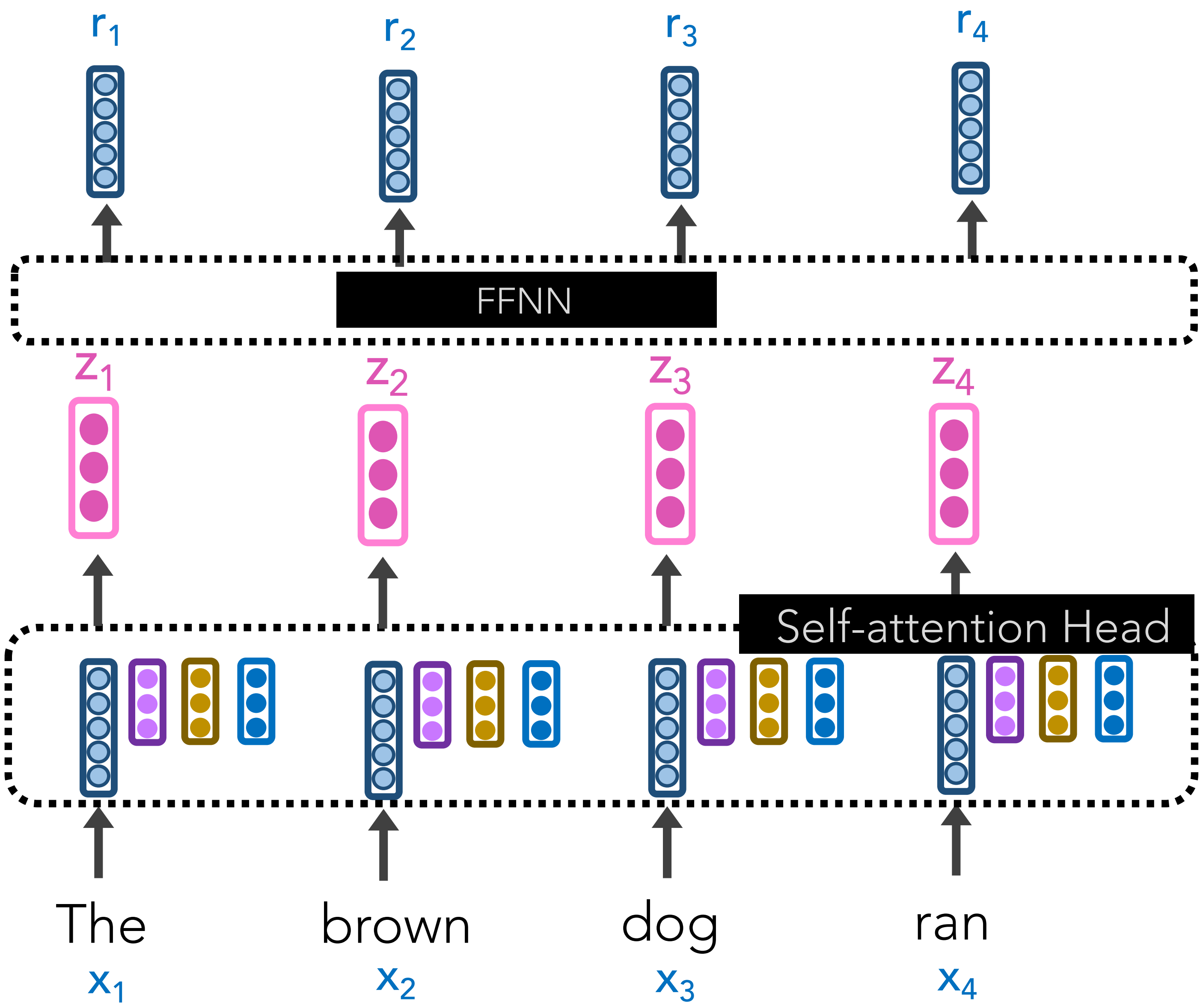


Self-Attention

Let's further pass each z_i through a FFNN

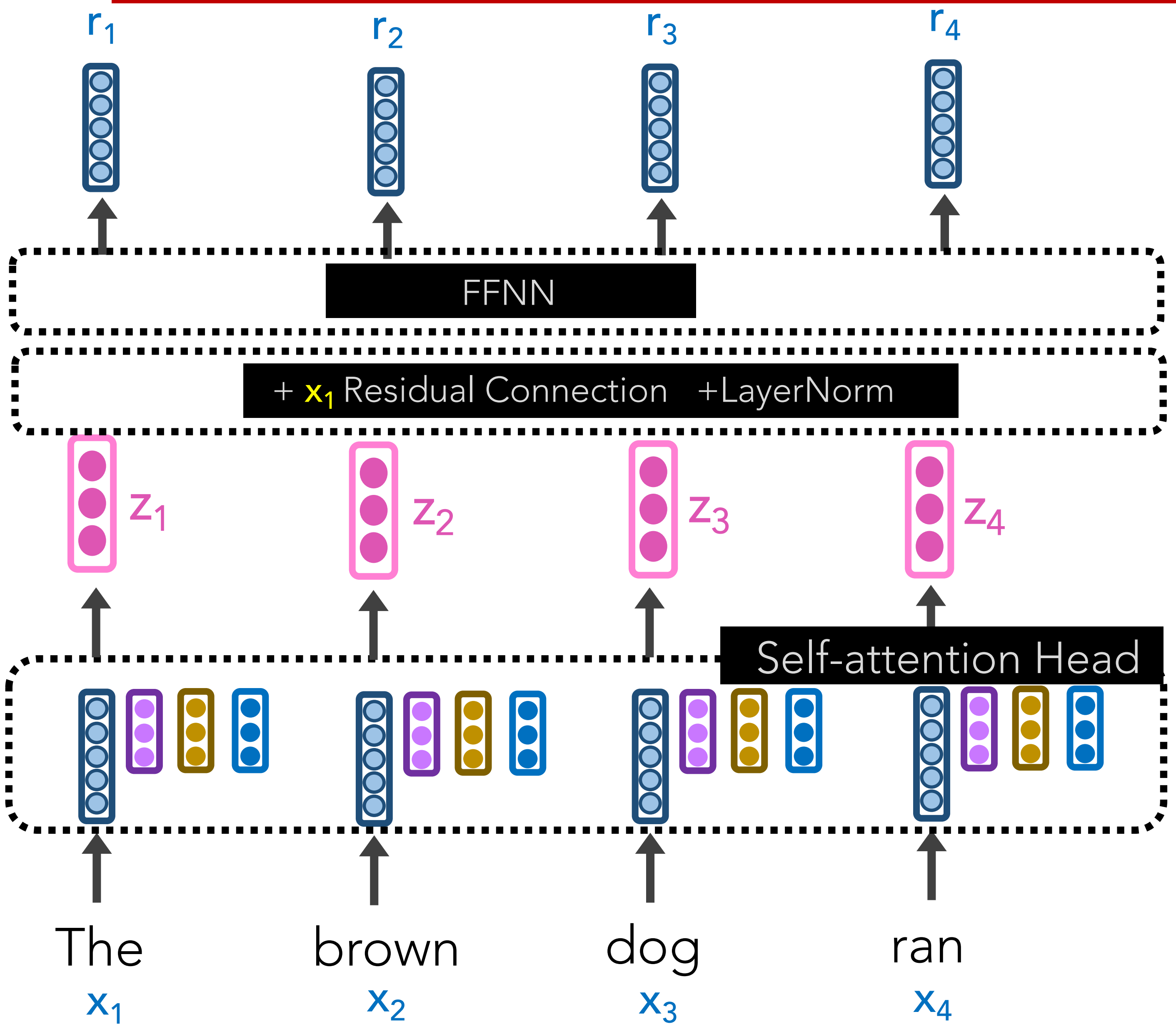


Self-Attention + FFNN



Let's further pass each z_i through a FFNN

Self-Attention + FFNN

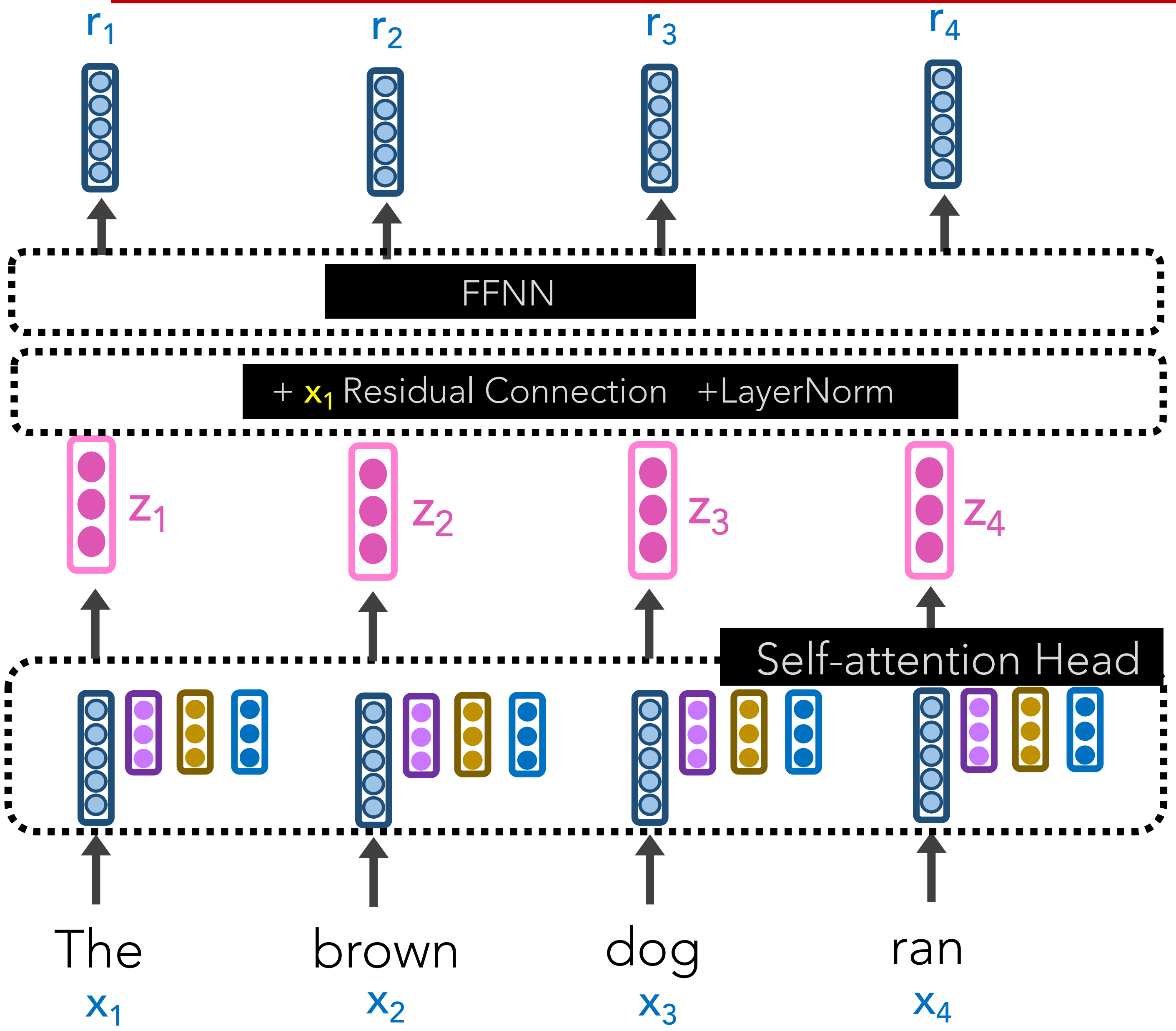


Let's further pass each z_i through a FFNN

We concat w/ a residual connection to help ensure relevant info is getting forward passed.

We perform LayerNorm to stabilize the network and allow for proper gradient flow.

Self-Attention + FFNN



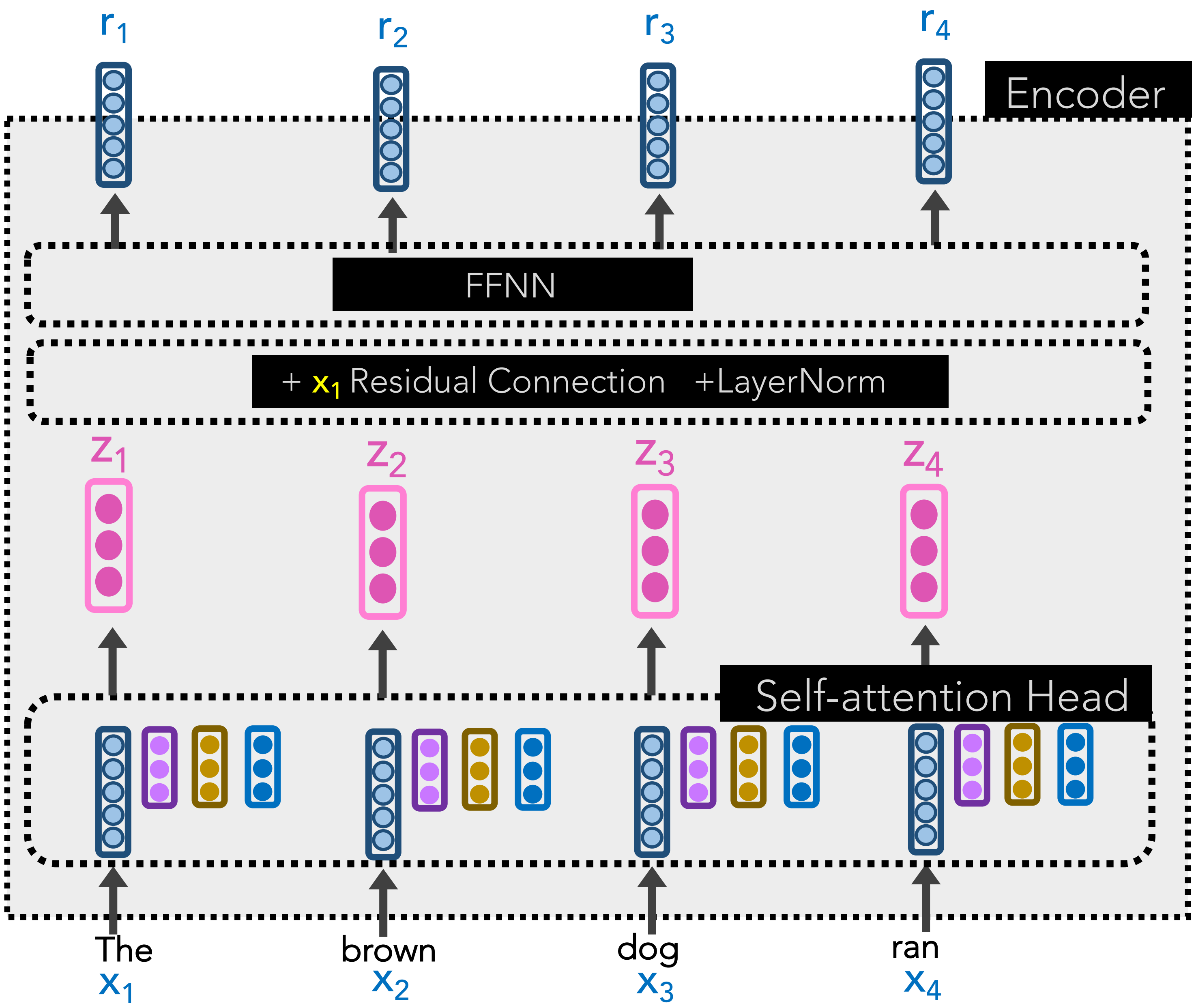
Let's further pass each z_i through a FFNN

We concat w/ a residual connection to help ensure relevant info is getting forward passed.

We perform LayerNorm to stabilize the network and allow for proper gradient flow.

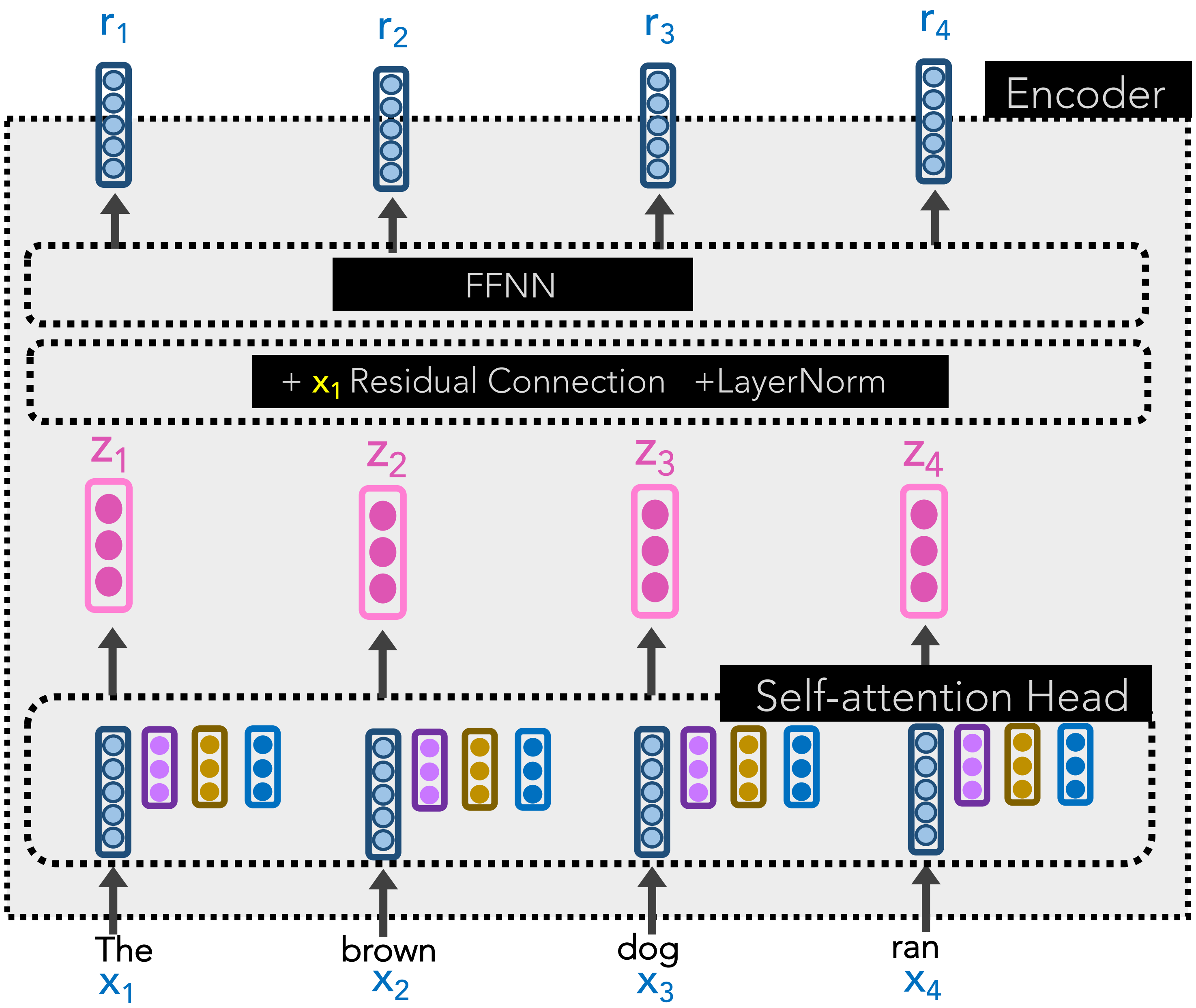
Each z_i can be computed in parallel, unlike LSTMs!

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

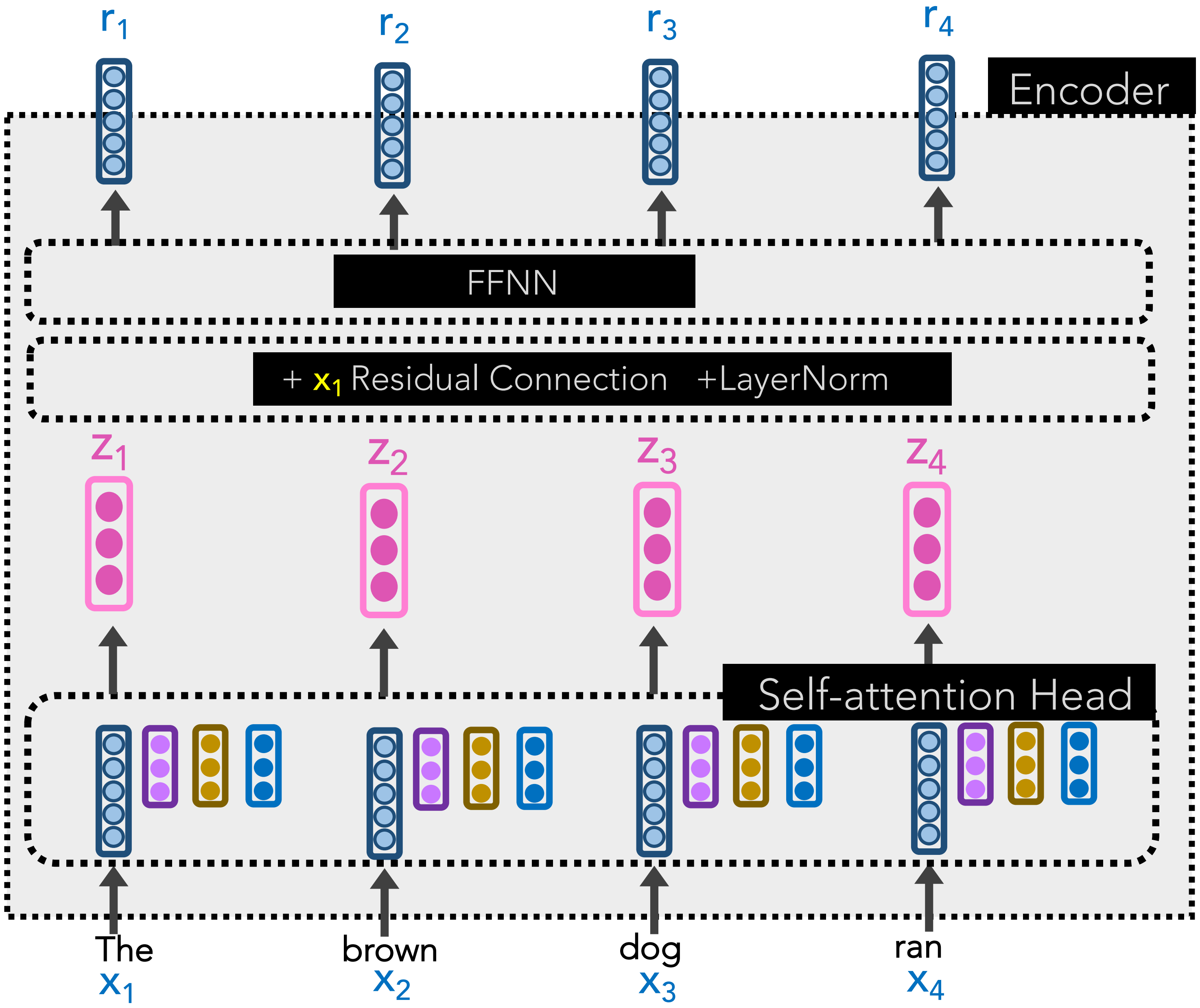
Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

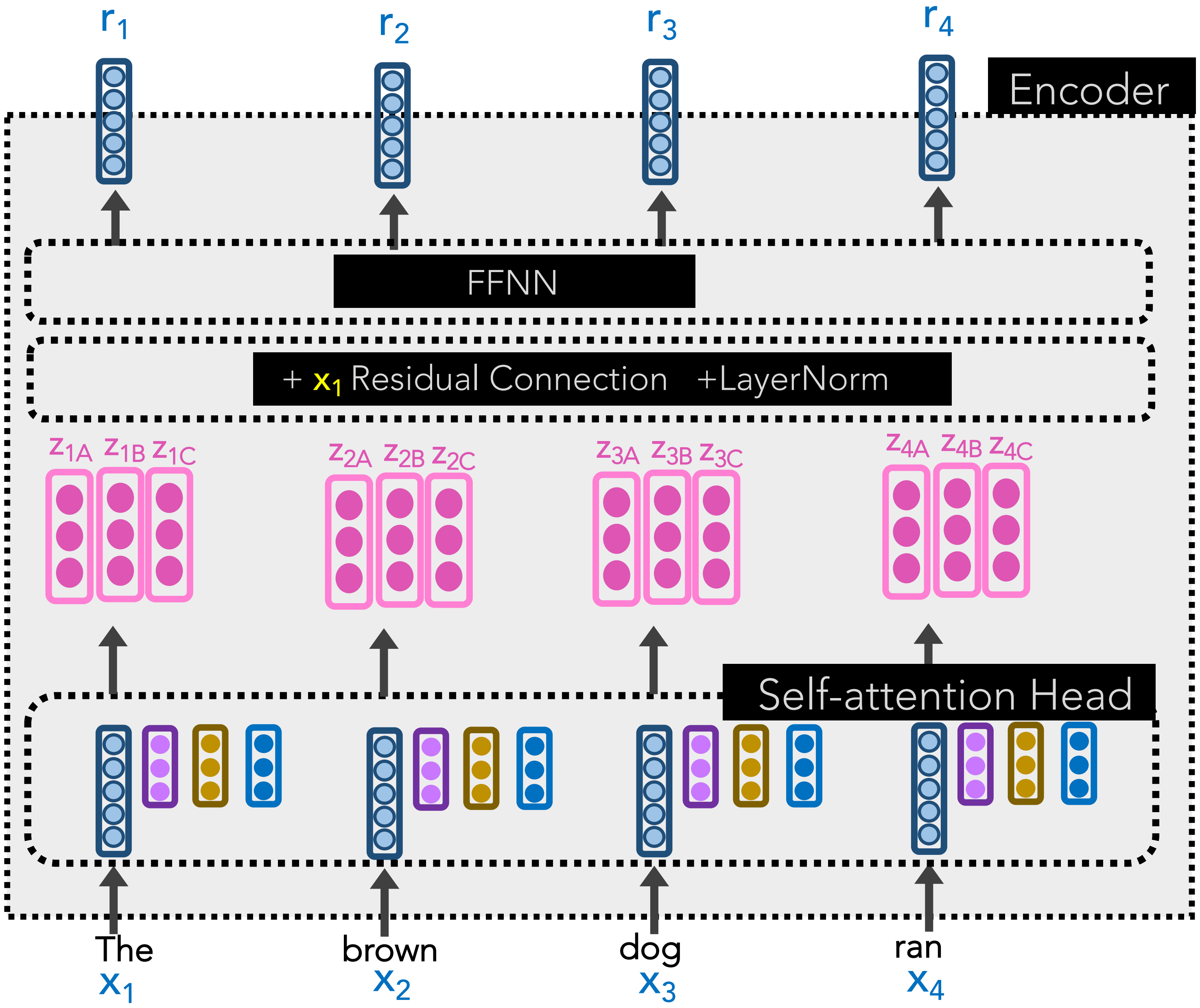
Solution: append each input word x_i with a positional encoding: $\sin(i)\cos(i)$

A Self-Attention Head has just one set of query/key/value weight matrices w_q, w_k, w_v

Words can relate in many ways, so it's restrictive to rely on just one Self-Attention Head in the system.

Let's create Multi-headed Self-Attention

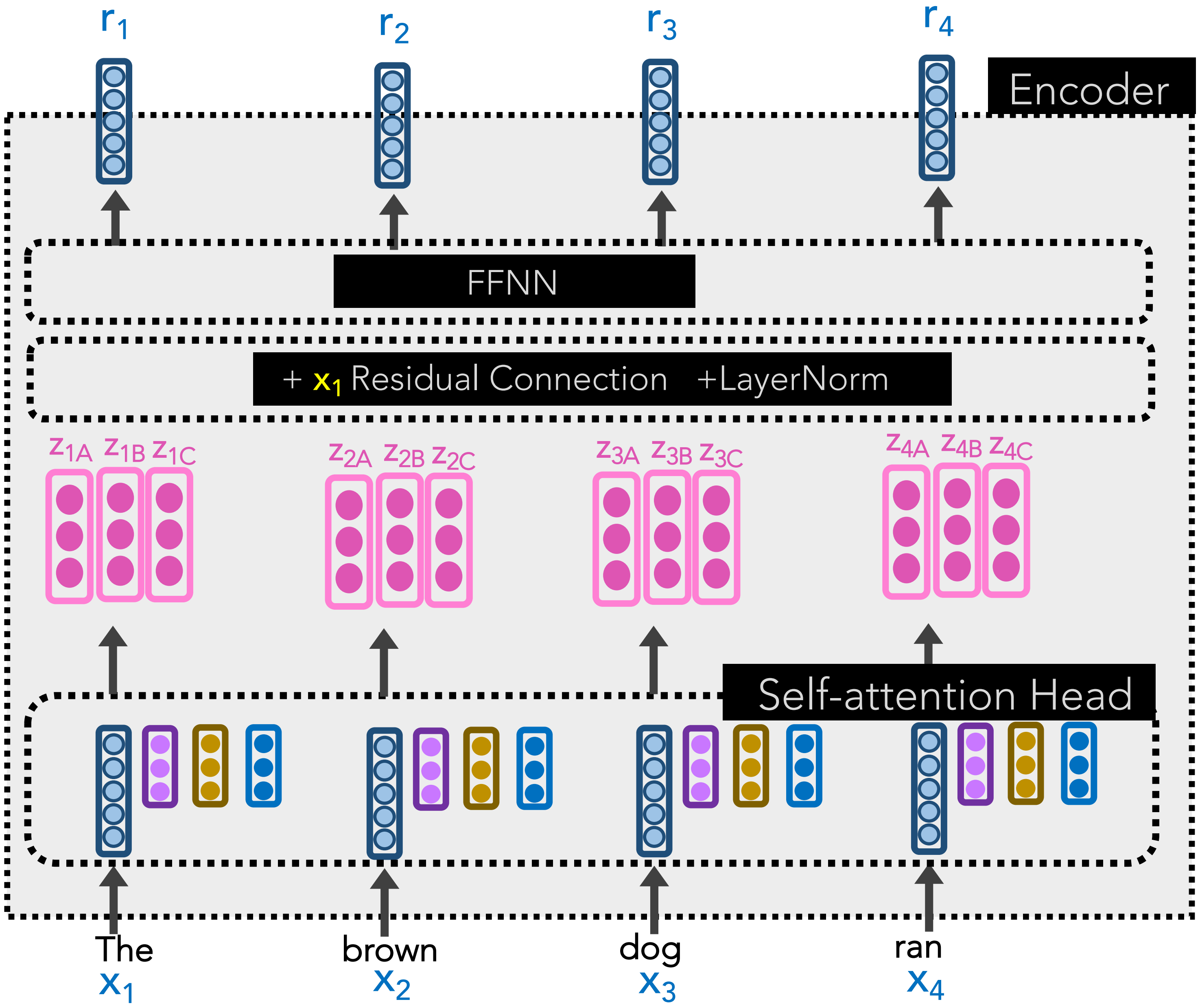
Transformer Encoder



Each Self-Attention Head produces a z_i vector.

We can, in parallel, use multiple heads and concat the z_i's.

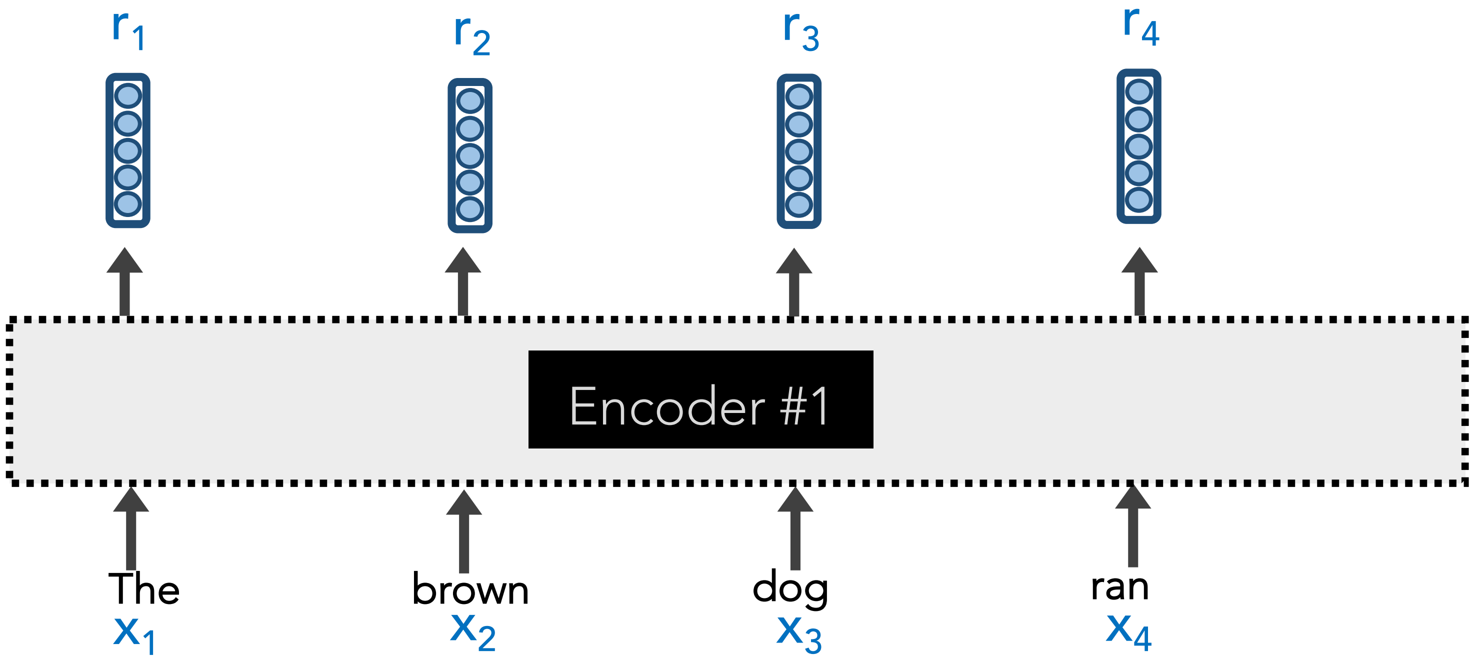
Transformer Encoder



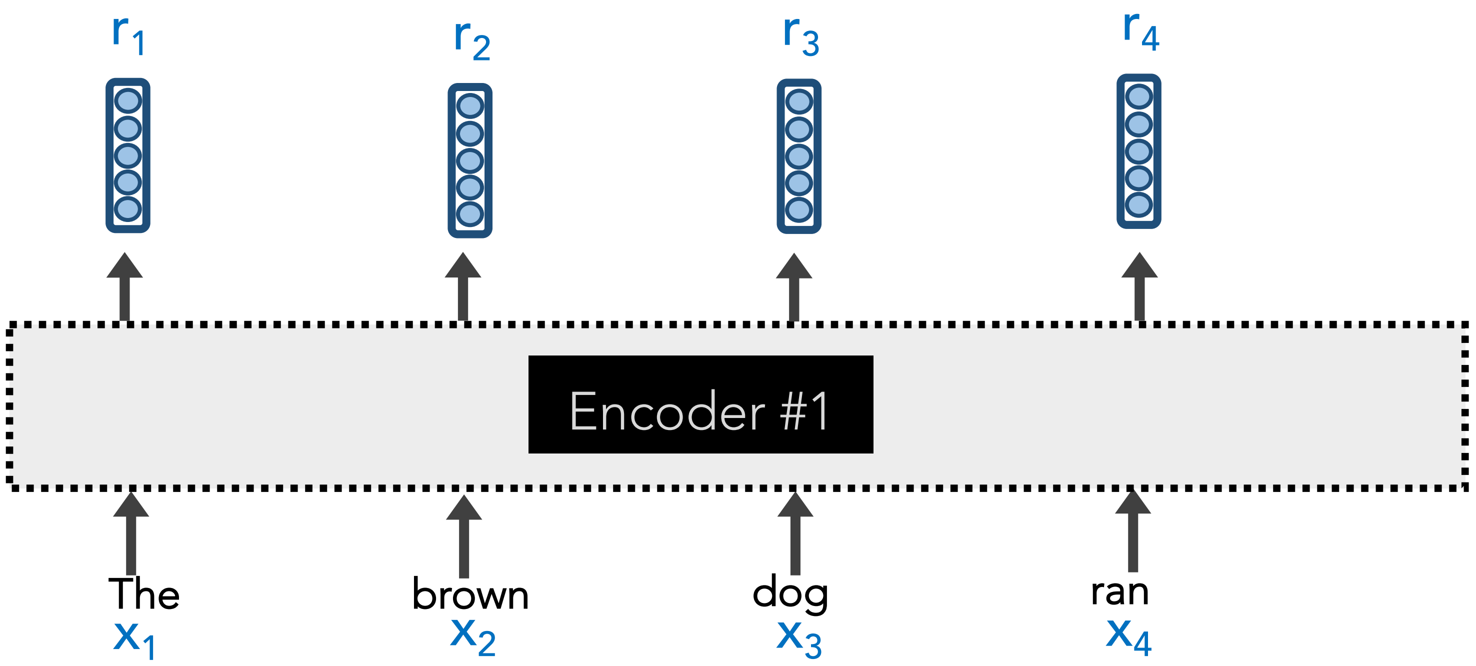
To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Transformer Encoder

To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i



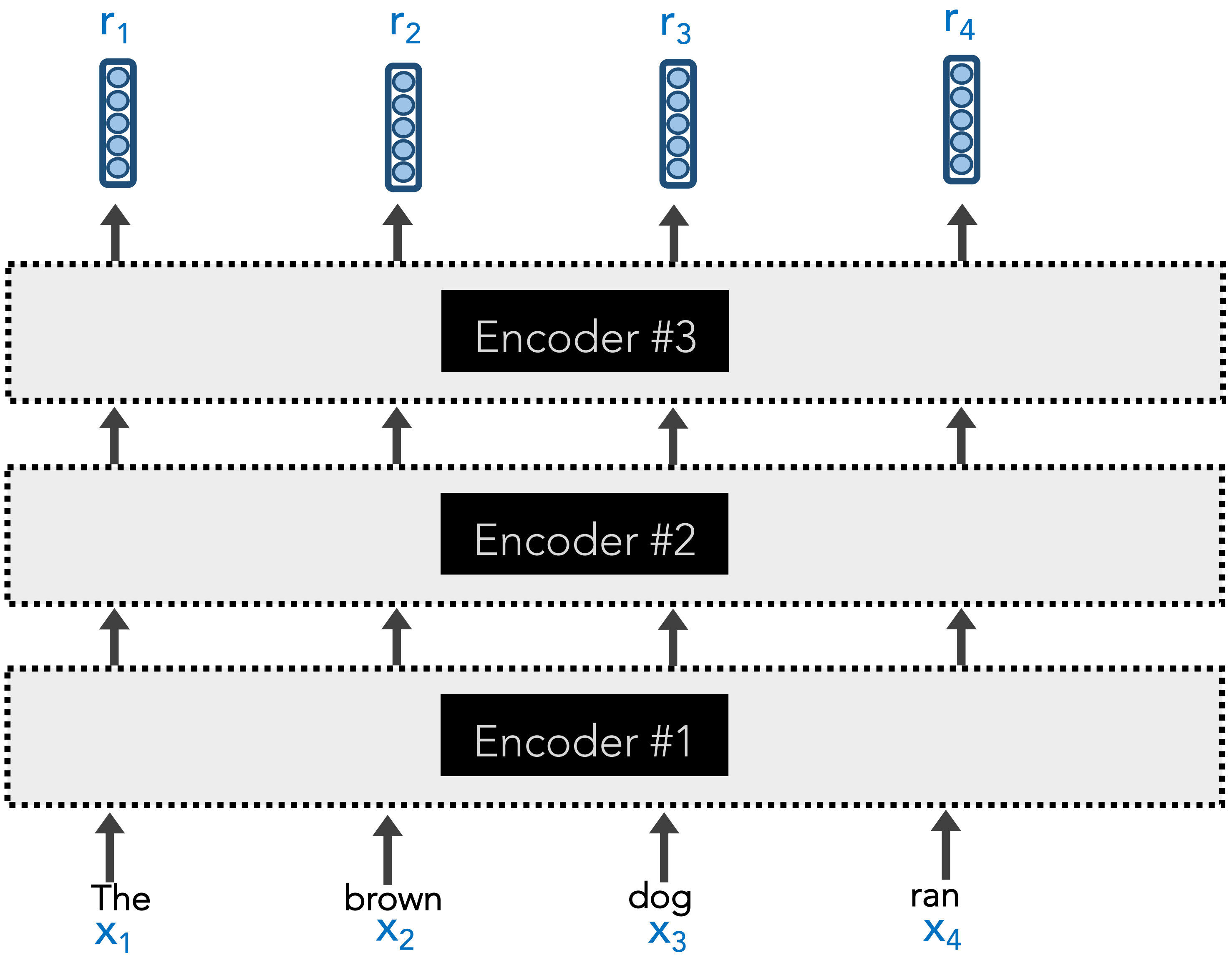
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

Transformer Encoder

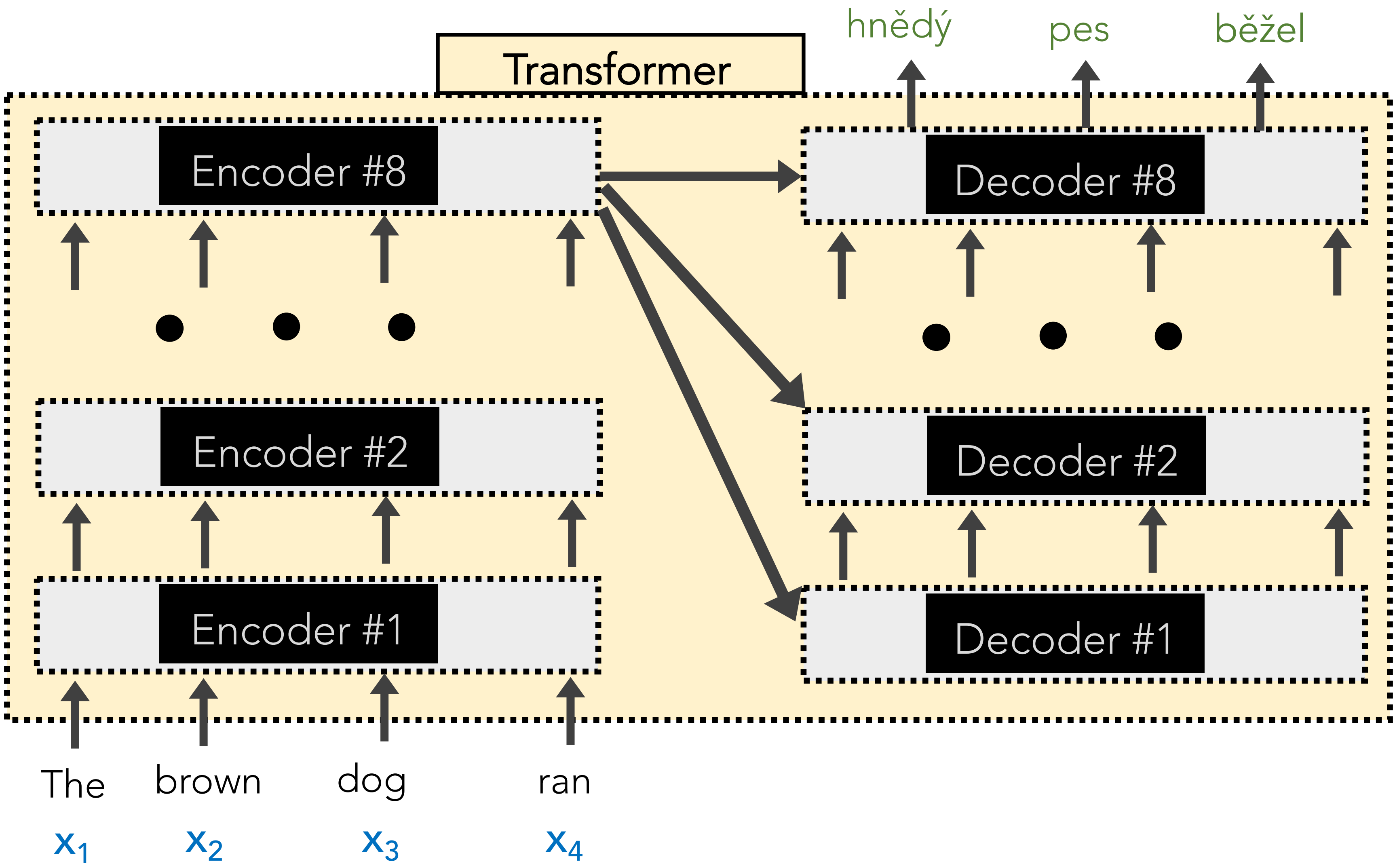


To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

The original Transformer model was intended for Machine Translation, so it had Decoders, too

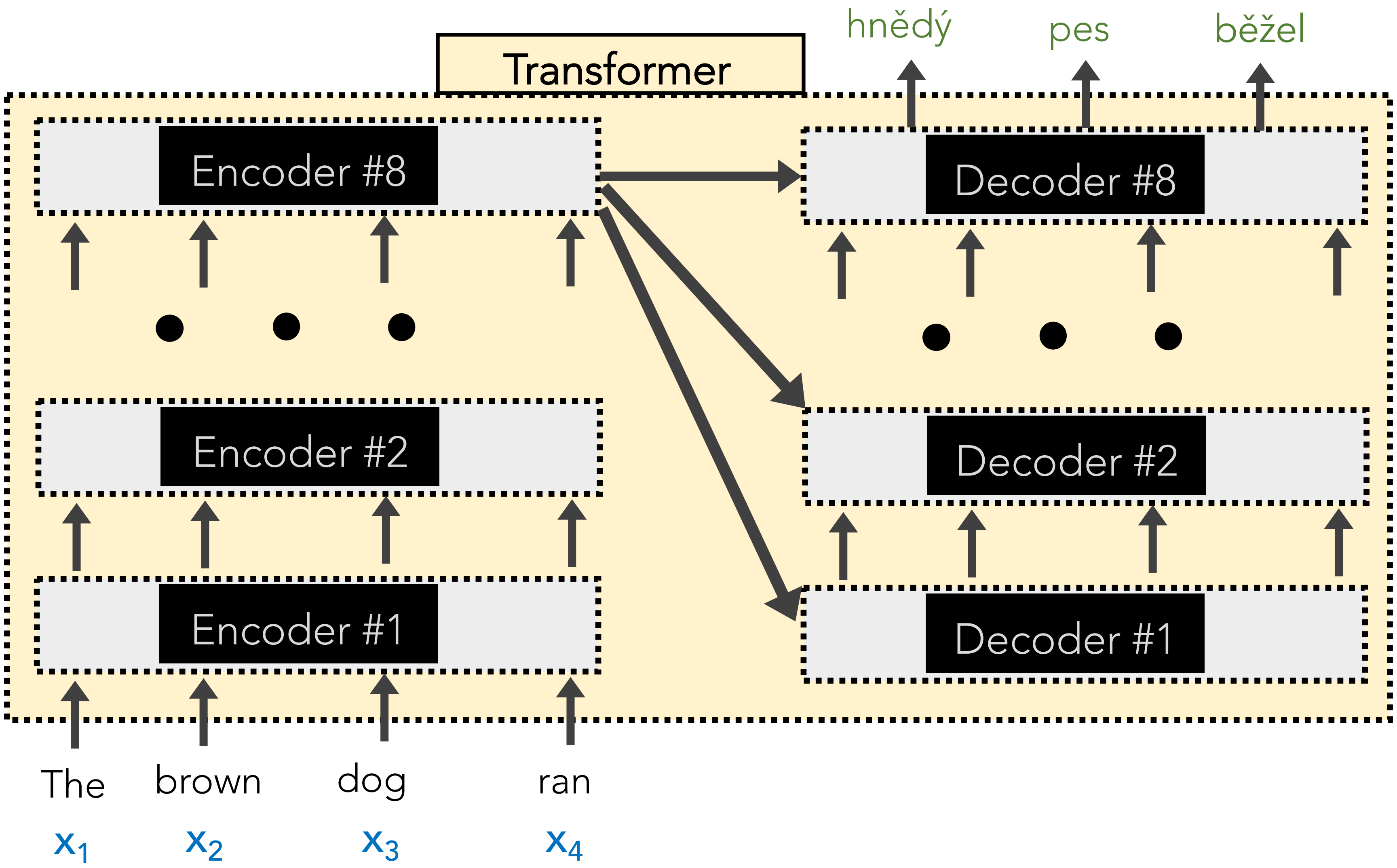
Transformer Encoders and Decoders



Transformer Encoders produce contextualized embeddings of each word

Transformer Decoders generate new sequences of text

Transformer Encoders and Decoders

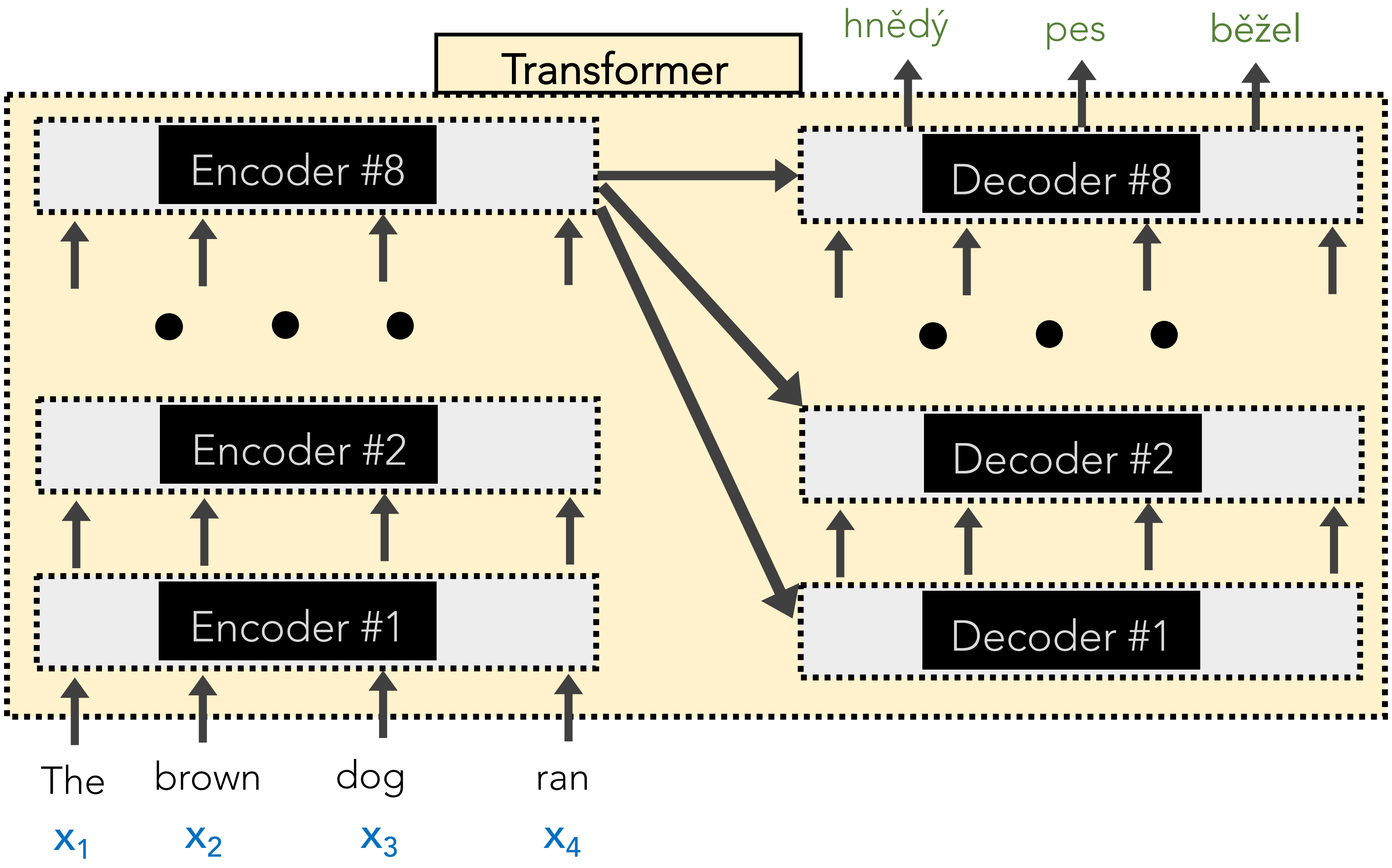


NOTE

Transformer Decoders are identical to the Encoders, except they have an additional Attention Head in between the Self-Attention and FFNN layers.

This additional Attention Head **focuses on parts of the encoder's representations.**

Transformer Encoders and Decoders

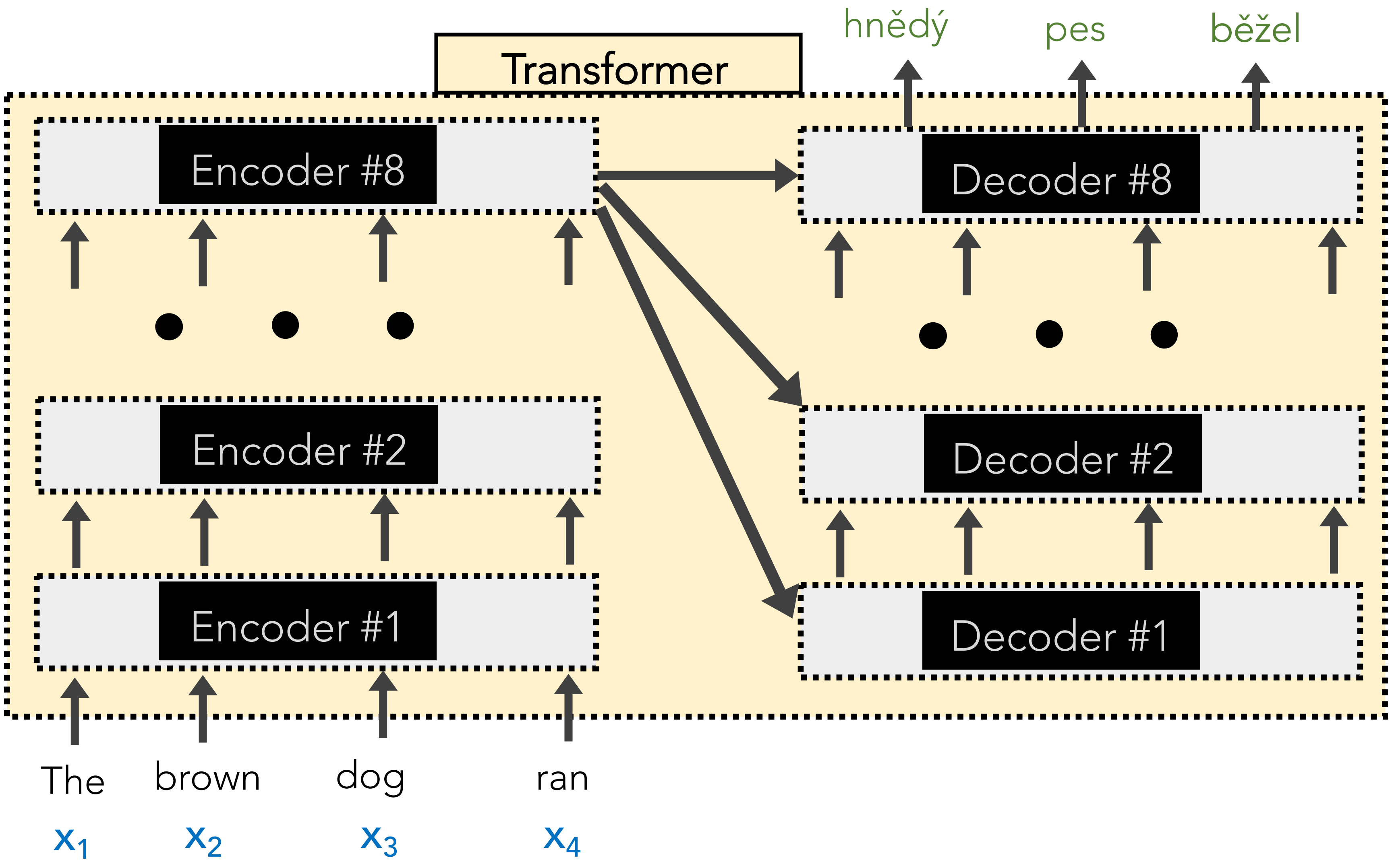


NOTE

The query vector for a Transformer Decoder's Attention Head (not Self-Attention Head) is from the output of the previous decoder layer.

However, the key and value vectors are from the Transformer Encoders' outputs.

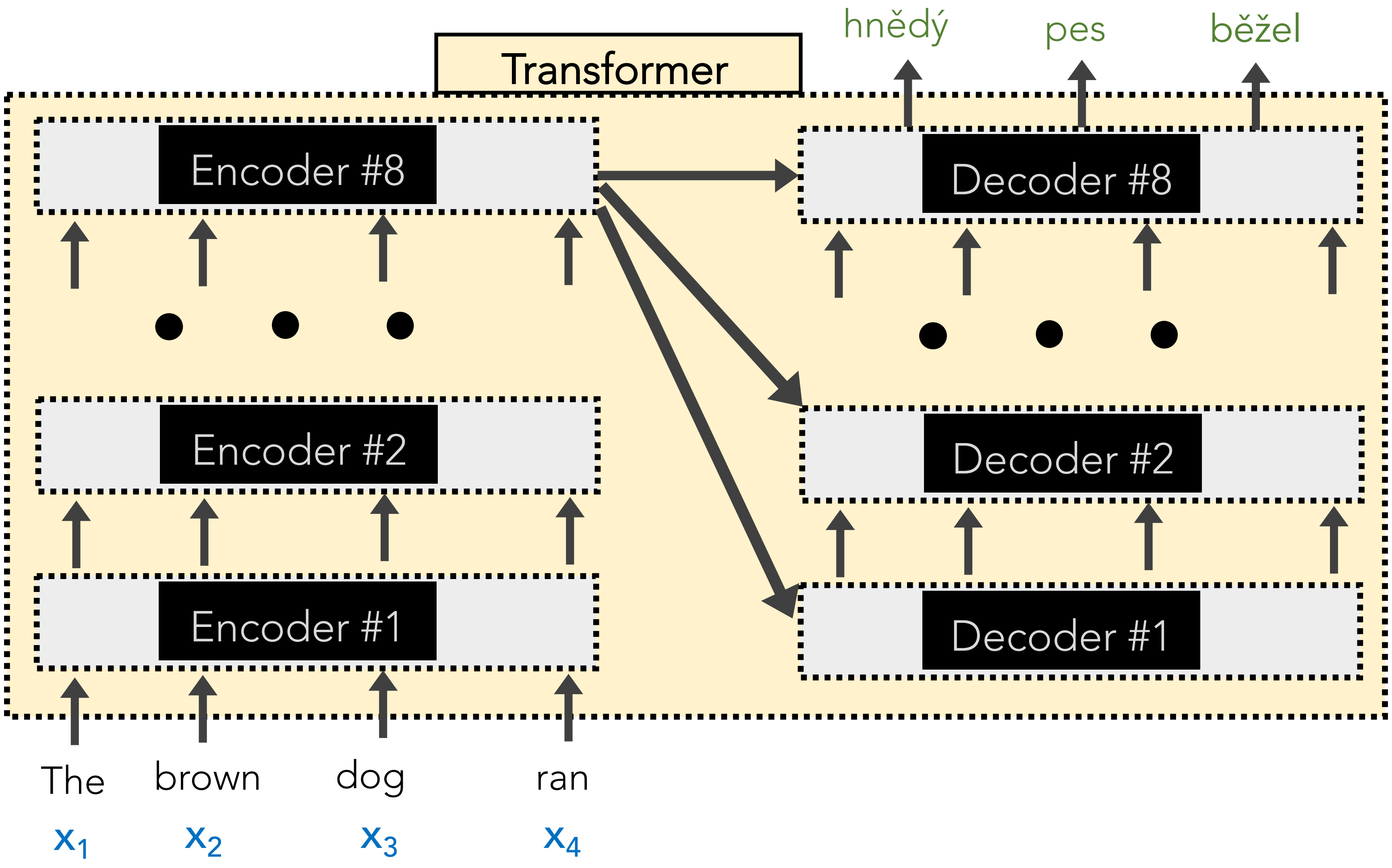
Transformer Encoders and Decoders



NOTE

The query, key, and value vectors for a Transformer Decoder's **Self-Attention Head** (not Attention Head) are all from the output of the previous decoder layer.

Transformer Encoders and Decoders



IMPORTANT

The Transformer Decoders have positional embeddings, too, just like the Encoders.

Critically, each position is only allowed to attend to the previous indices. This *masked* Attention preserves it as being an auto-regressive LM.

Loss Function: cross-entropy (predicting translated word)

Training Time: ~4 days on (8) GPUs

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Machine Translation results: state-of-the-art (at the time)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Machine Translation results: state-of-the-art (at the time)

You can train to translate from Language A to Language B.

Then train it to translate from Language B. to Language C.

Then, without training, it can translate from Language A to Language C

- What if we don't want to decode/translate?
- Just want to perform a particular task (e.g., classification)
- Want even more robust, flexible, rich representation!
- Want positionality to play a more explicit role, while not being restricted to a particular form (e.g., CNNs)