# Lecture 6: Transformers, Part I

From Self-Attention to BERT

MIT

6.861*

Yoon Kim, Jacob Andreas, and Chris Tanner

# Self [Attention]

-- Mac Miller (2018)

MITMedical

Urgent Care
617-253-1311

Urgent Mental Health Concerns
617-253-2916

# ANNOUNCEMENTS

- Website has tons of Research Project info, including examples of good projects
- HW2 is being finalized; will be released later this week

# RESEARCH PROJECTS

- Most research experiences/opportunities are "top-down"
- You're all creative and fully capable.
- Allow yourselves to become comfortable with the unknown.

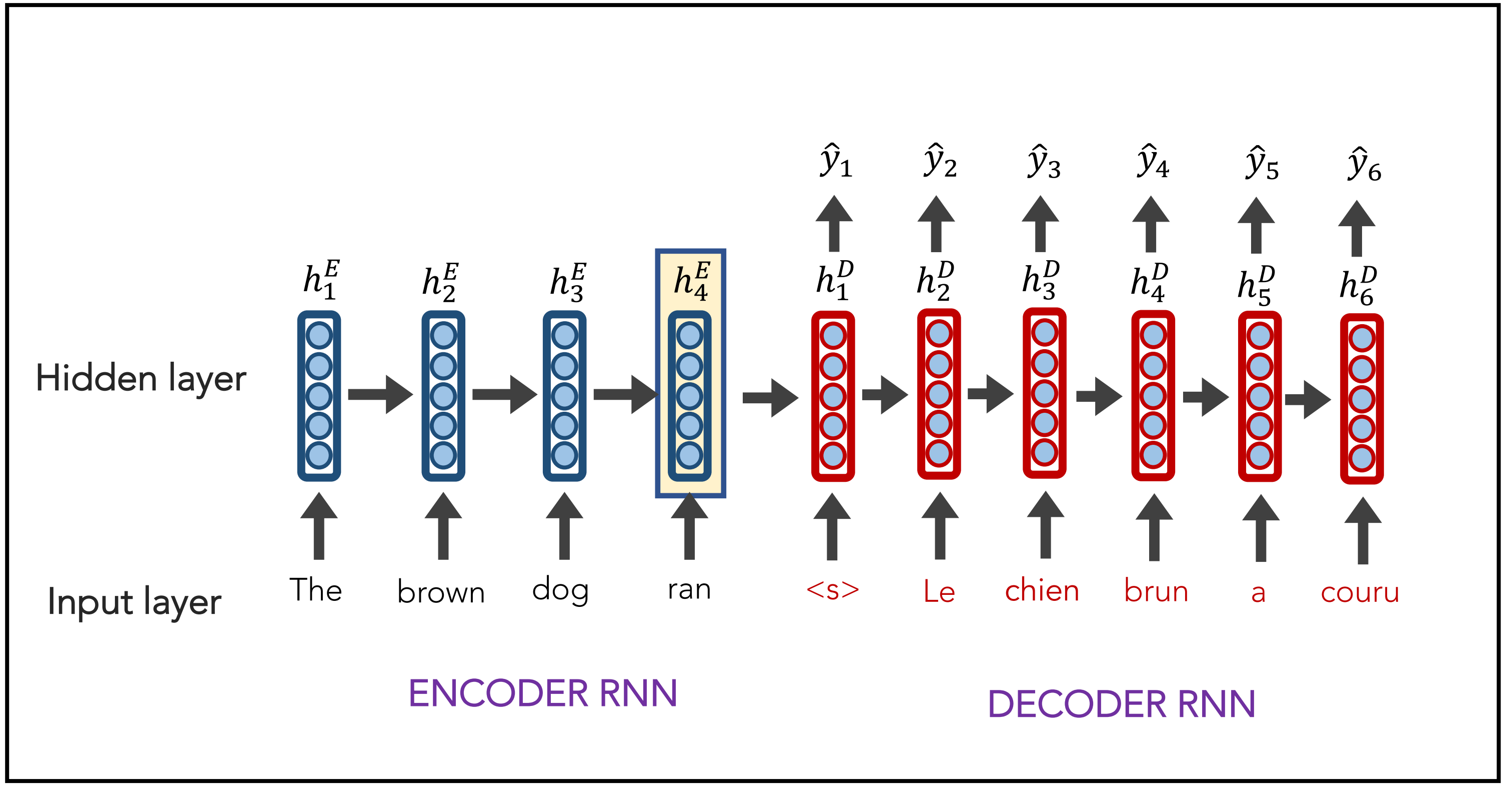
# RESEARCH PROJECTS

- We will try to provide feedback about project proposals based on:
  - 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: L5

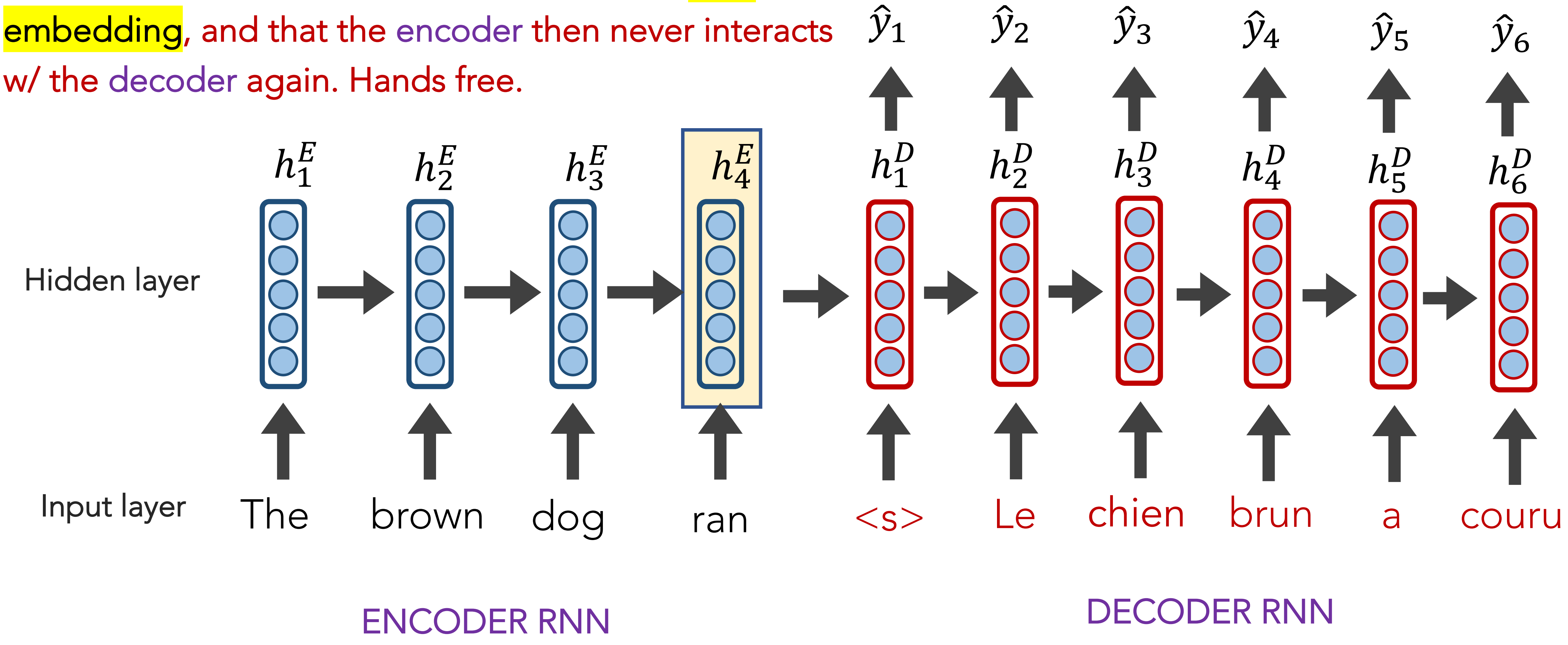
## seq2seq models

- 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: L5 seq2seq models

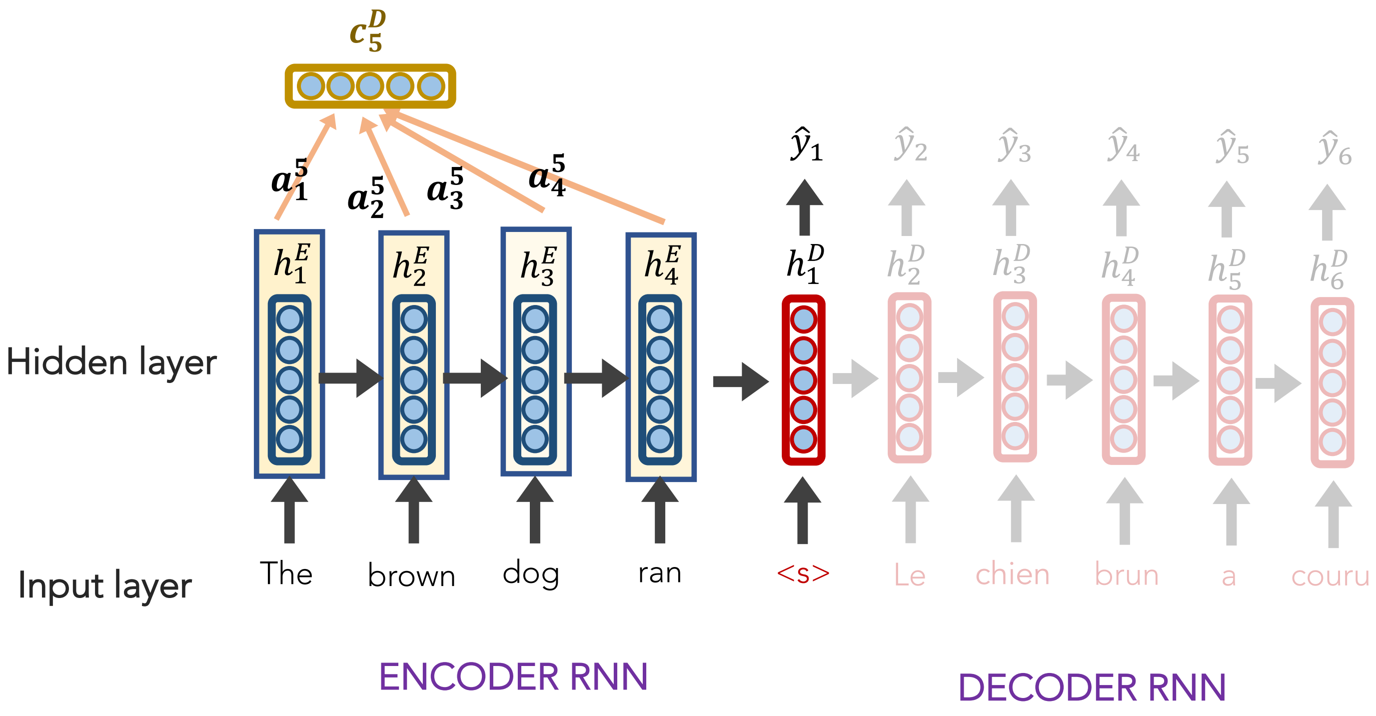
It's absurd that the entire "meaning" of the 1st sequence is expected to be packed into this one embedding, and that the encoder then never interacts w/ the decoder again. Hands free.



# RECAP: L5 seq2seq models

- **Attention** allows a decoder, at each time step, to focus on (pay "attention" to) *a distribution* 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: L5 Machine Translation (MT)

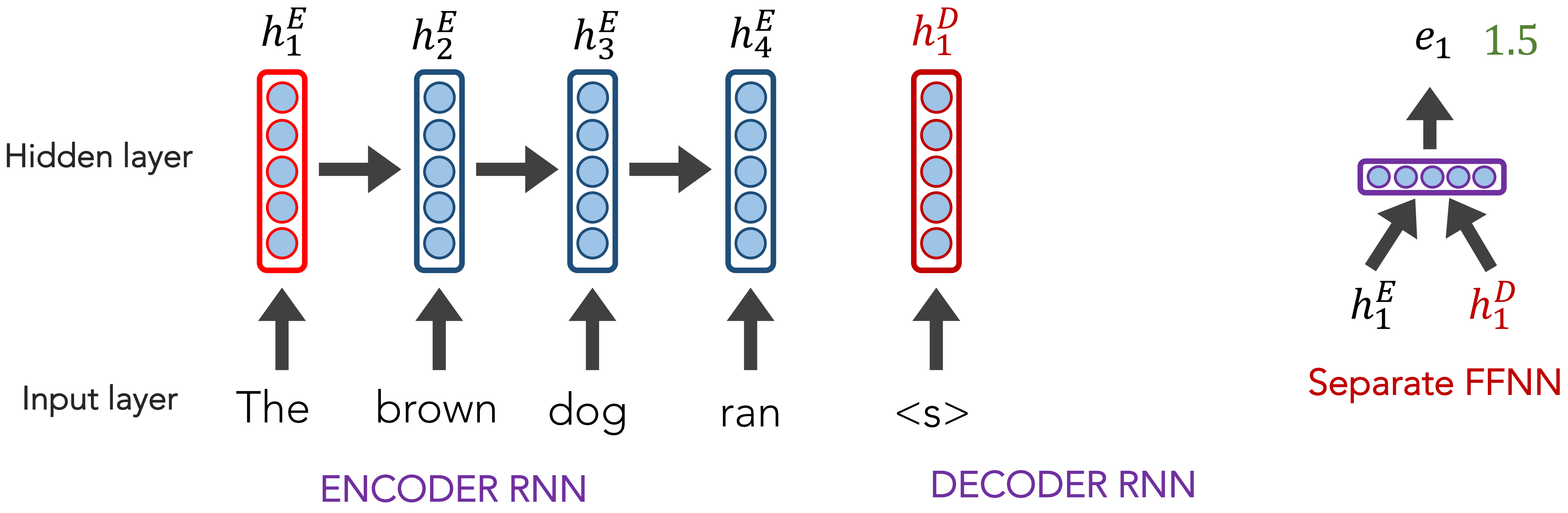
$$\mathrm{argmax}_y P(x|y)P(y)$$

- 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

# seq2seq + Attention

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

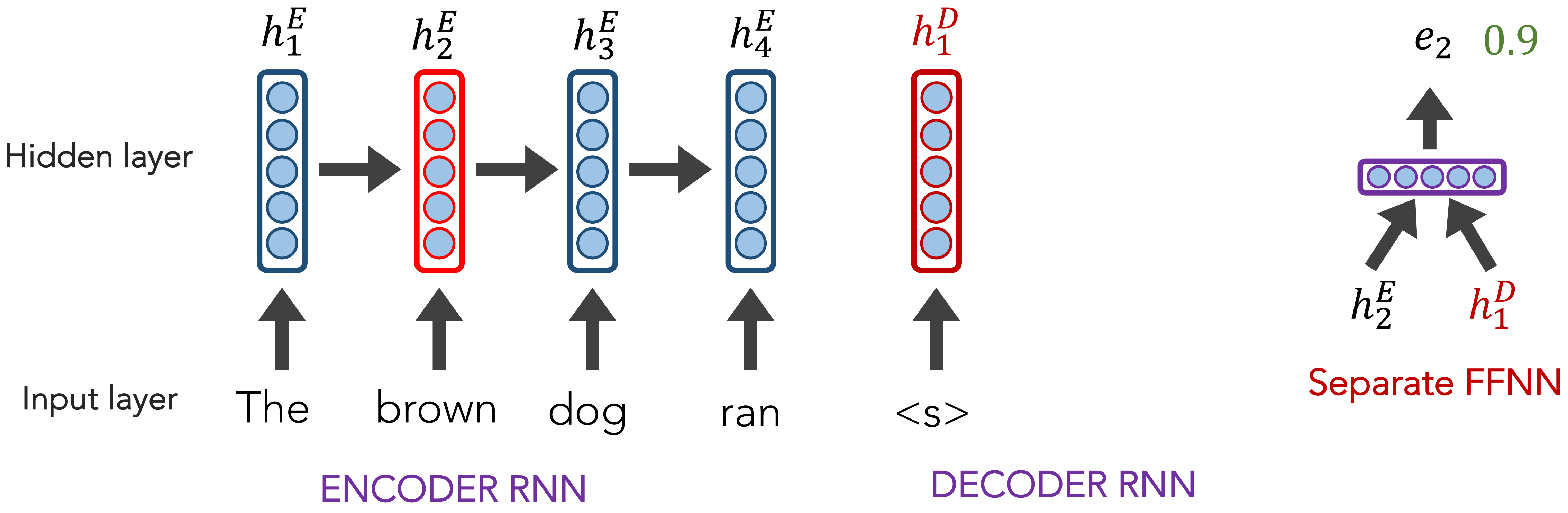
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



# seq2seq + Attention

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

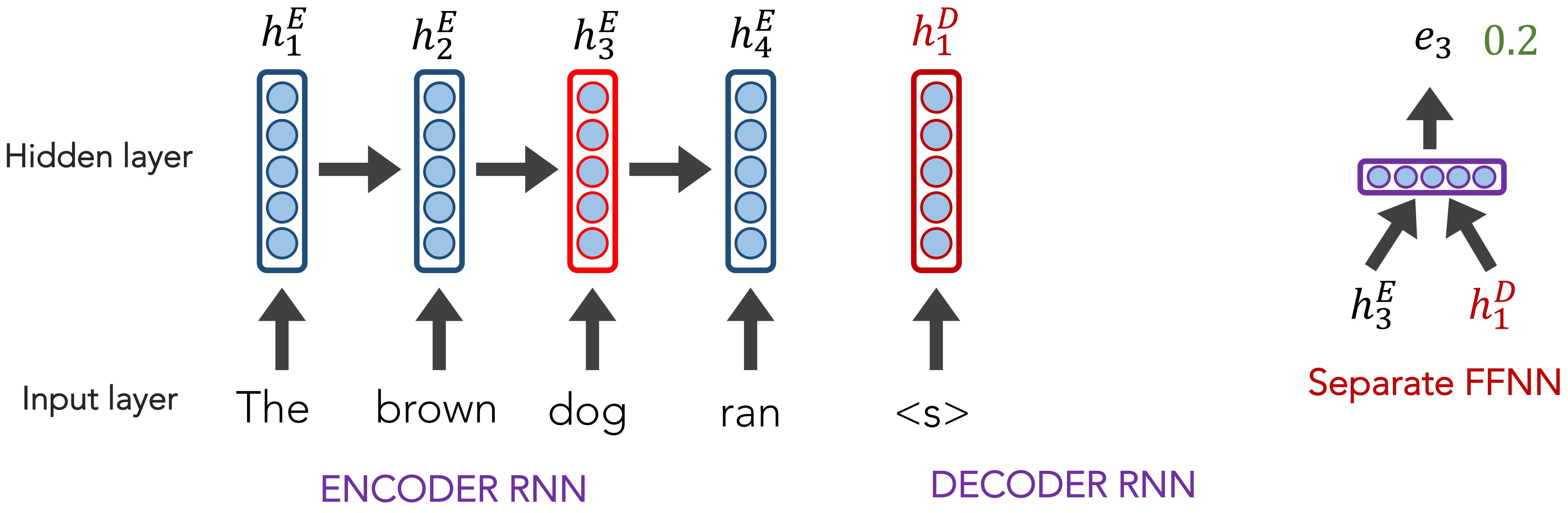
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



# seq2seq + Attention

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

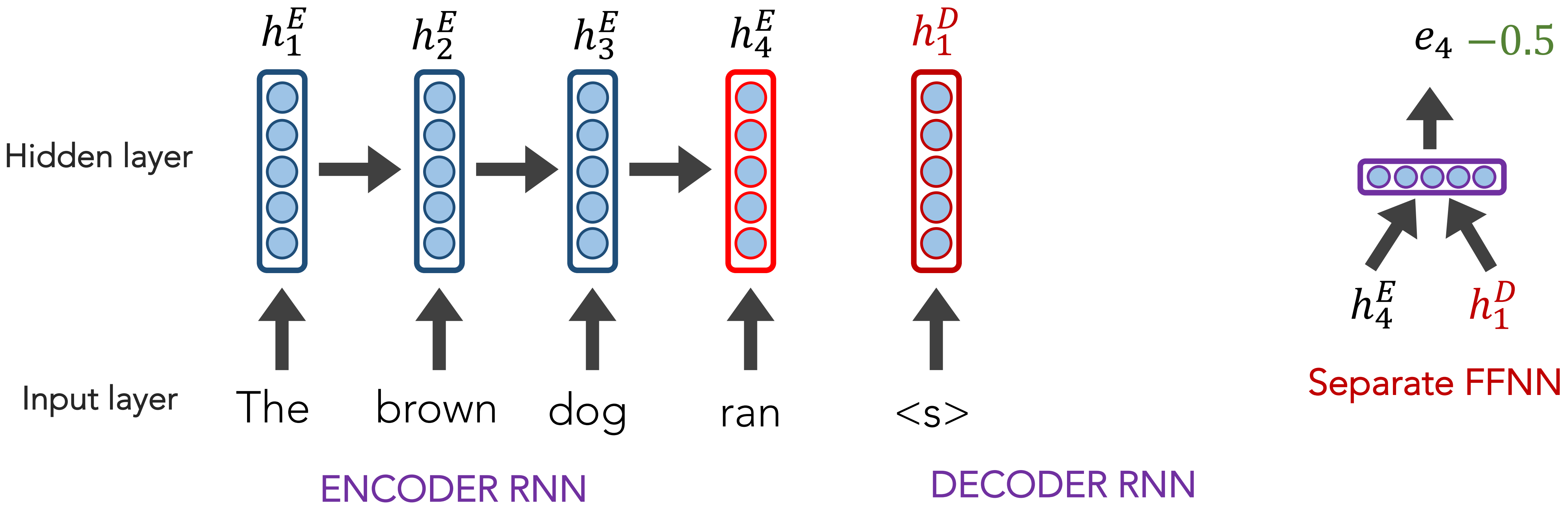
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



# seq2seq + Attention

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

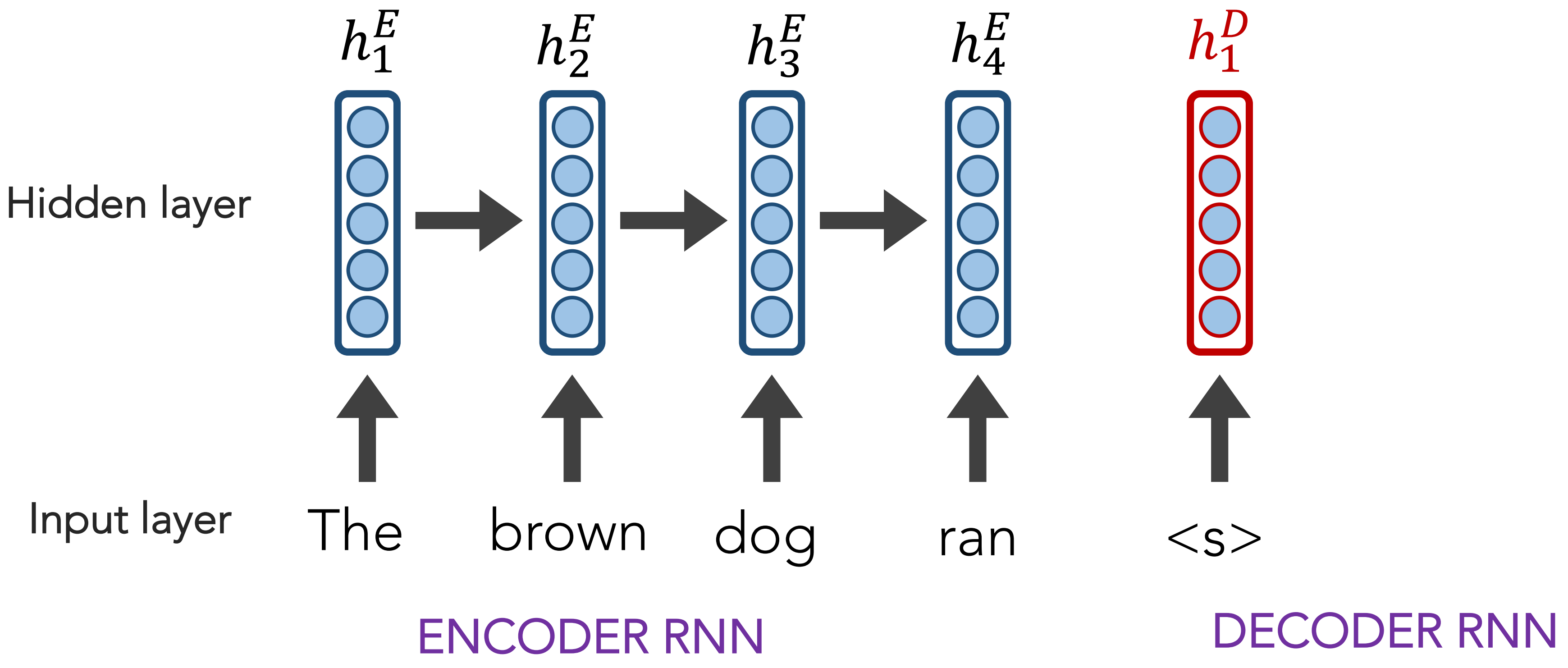
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



# 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)

$e_1$ $1.5$

$e_2$ $0.9$

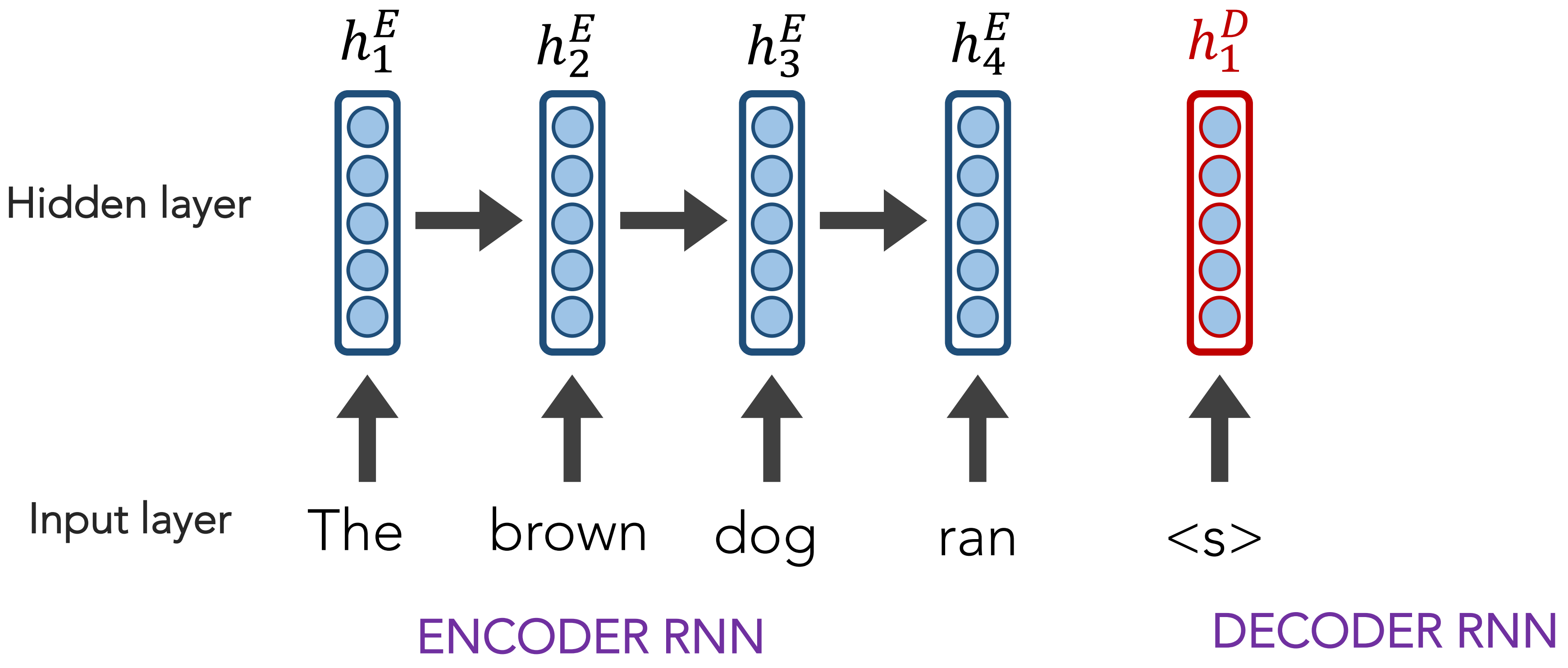
$e_3$ $0.2$

$e_4$ $-0.5$

# 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

$e_4$ $-0.5$

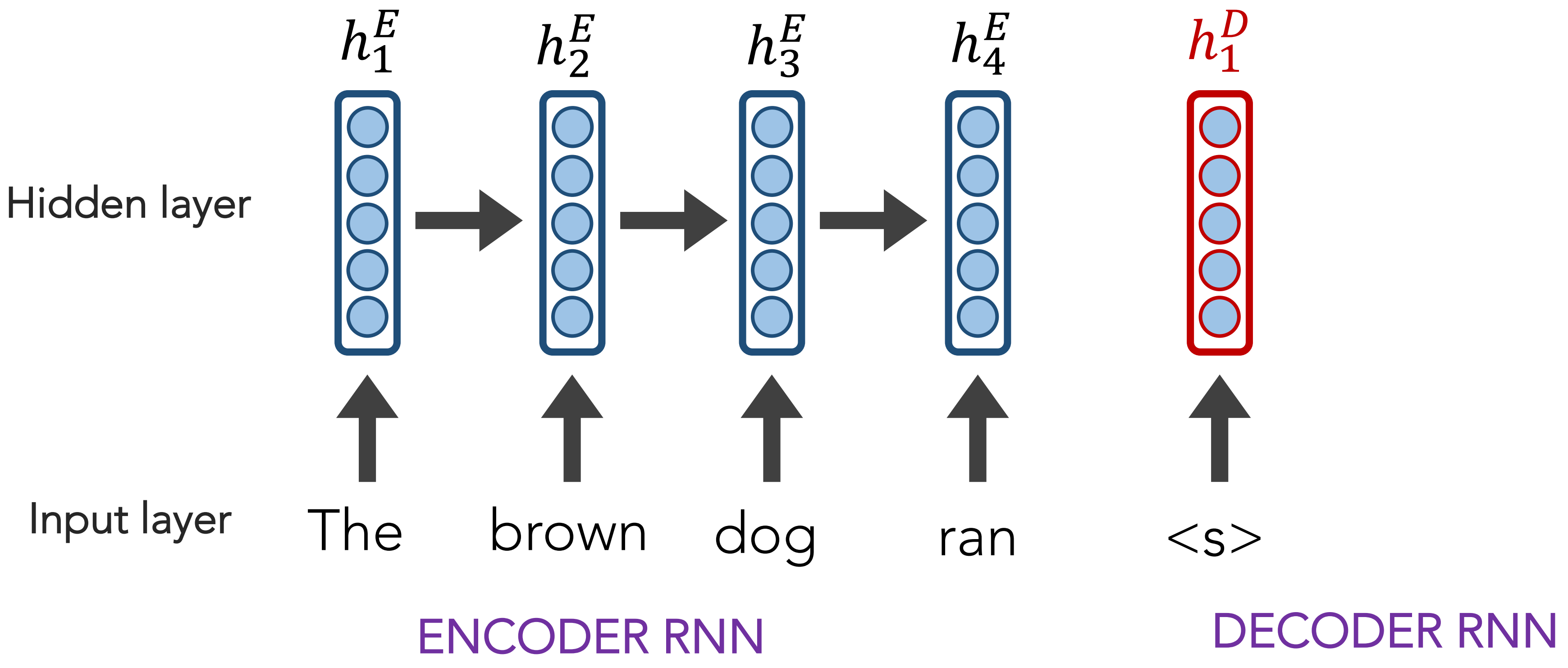
Attention (softmax'd)

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

# 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!



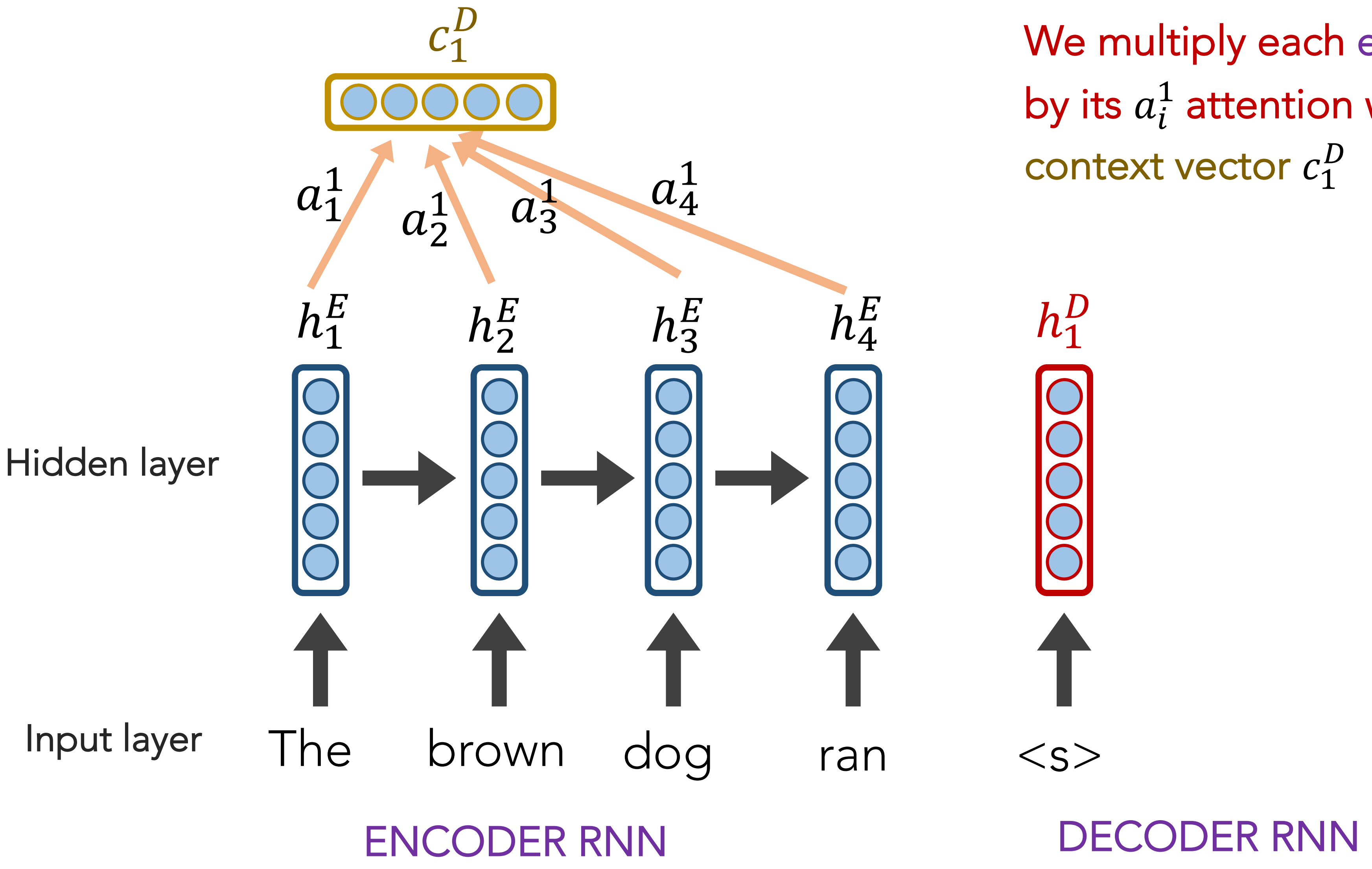

Attention (raw scores)

$e_1$ 1.5
$e_2$ 0.9
$e_3$ 0.2
$e_4$ $-0.5$

Attention (softmax'd)

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

# 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)

$a_1^1 = 0.51$

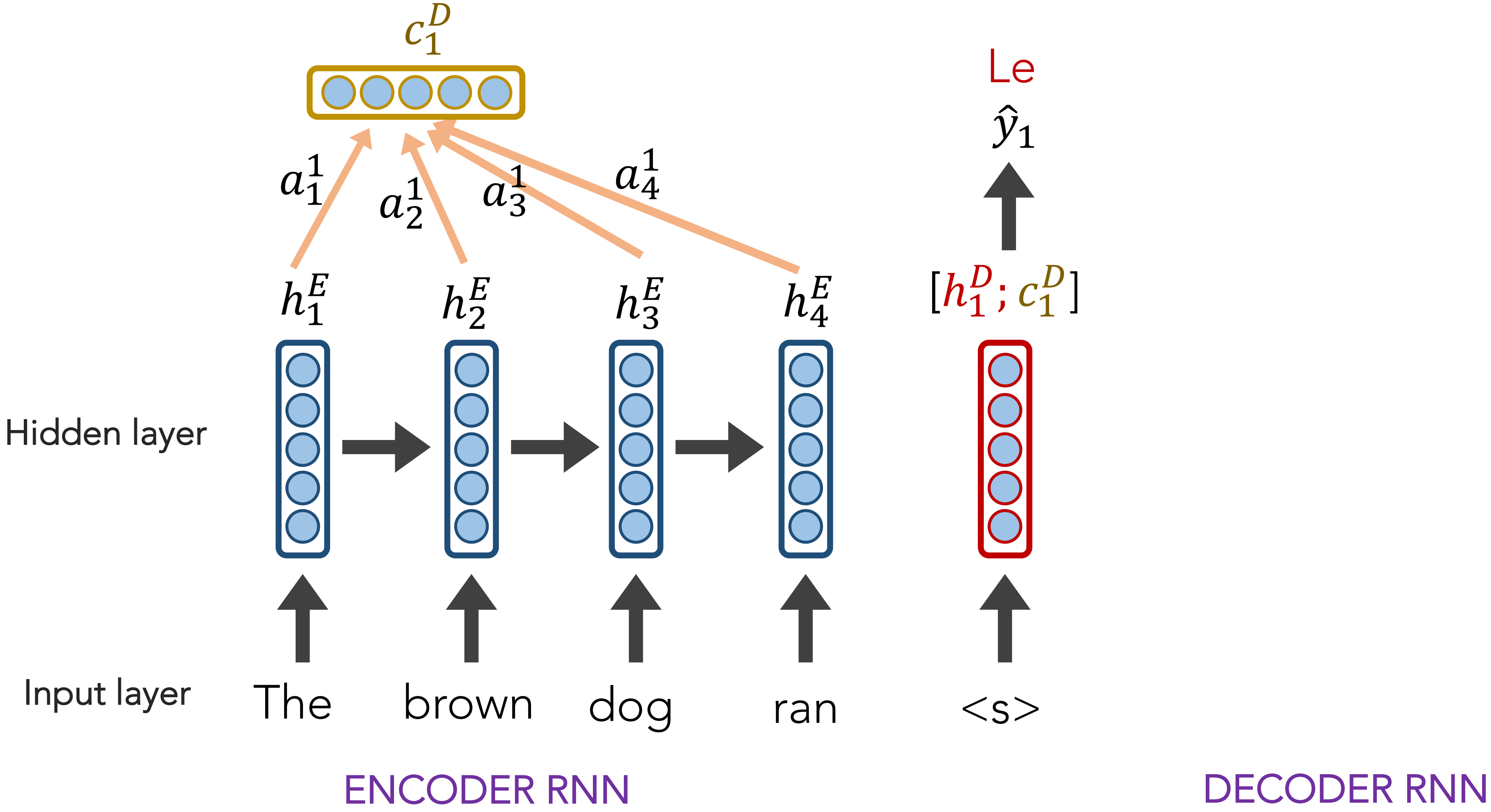
$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_4^1 = 0.07$

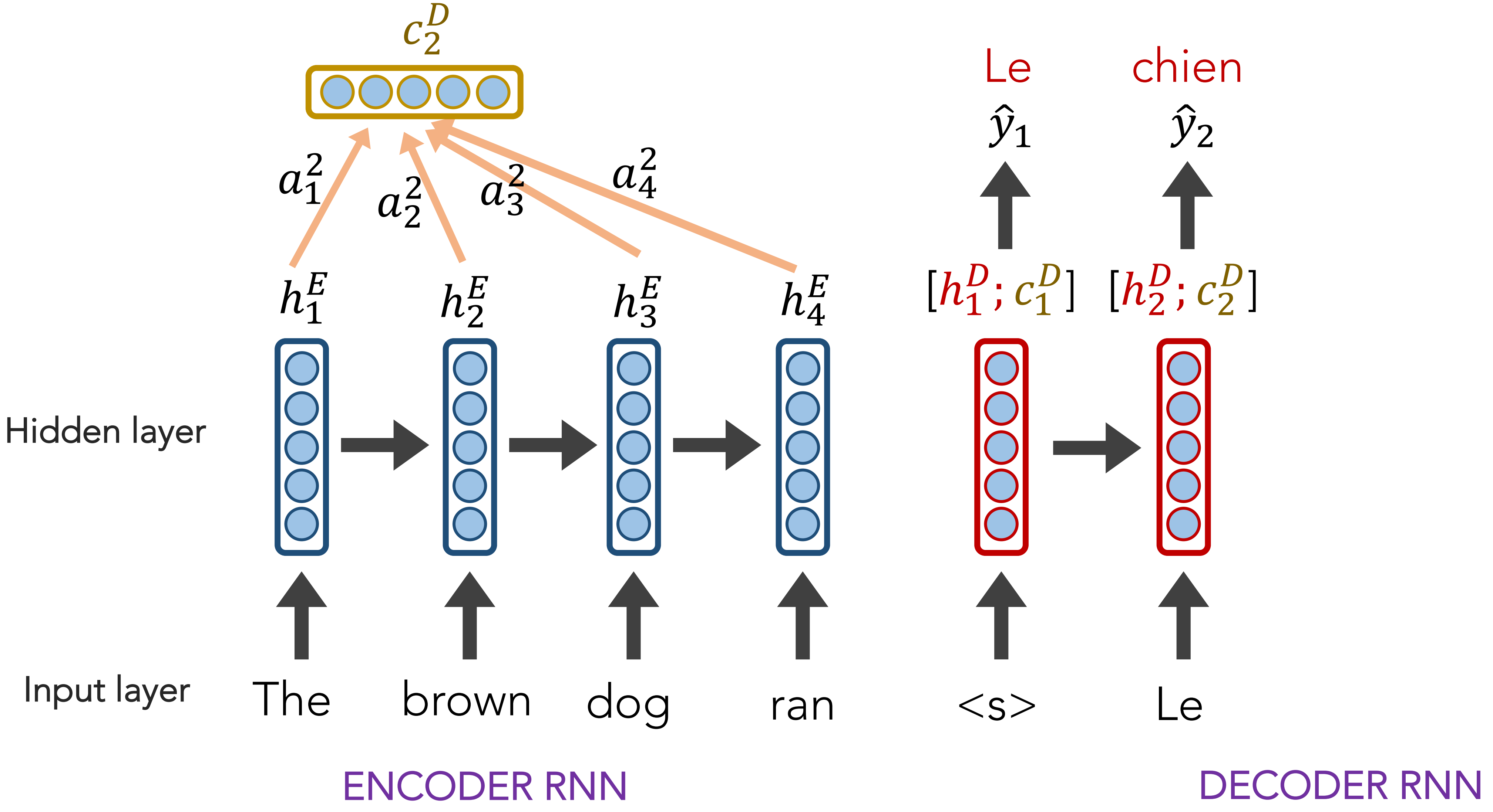
# seq2seq + Attention

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



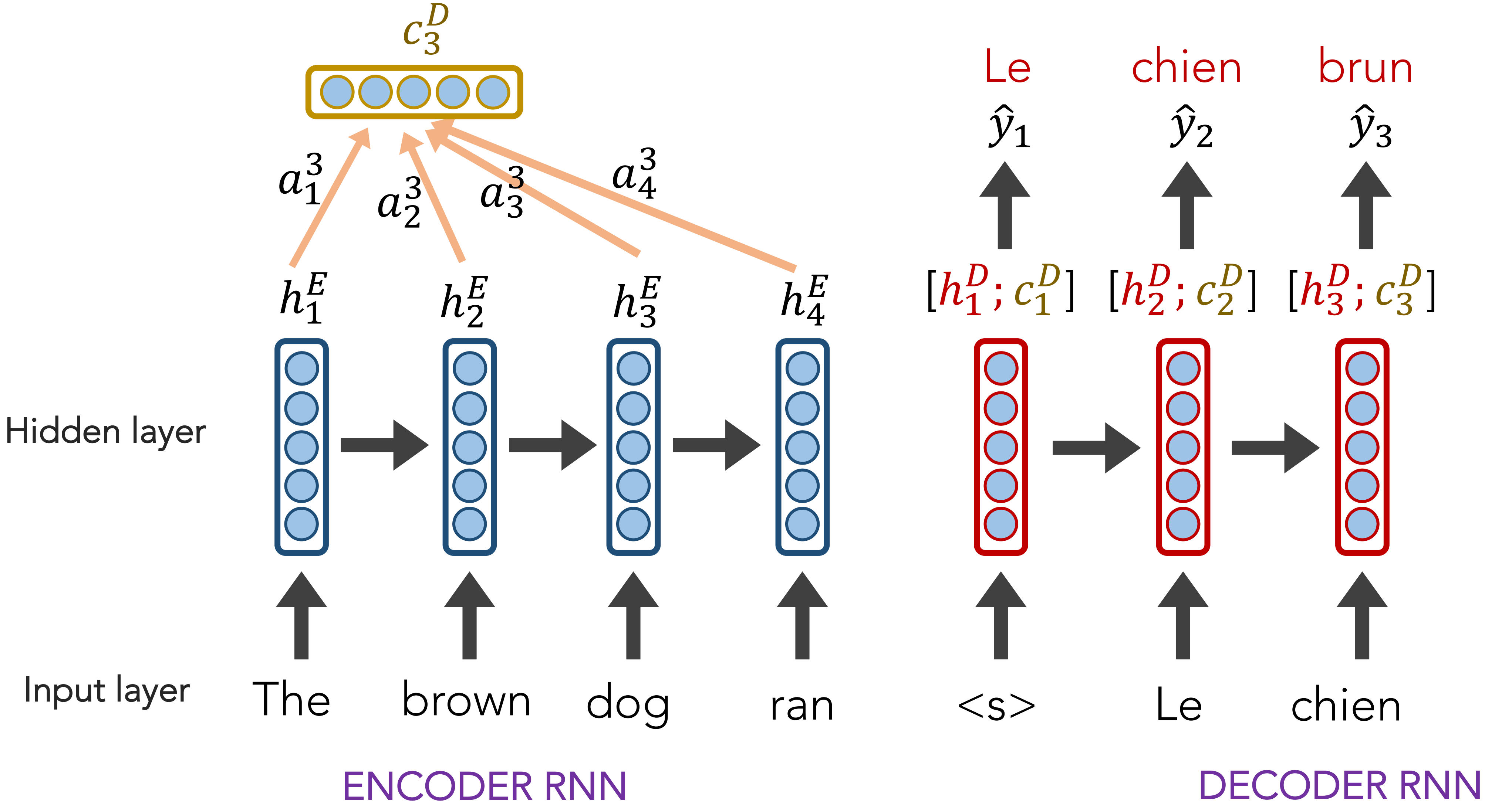
# seq2seq + Attention

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



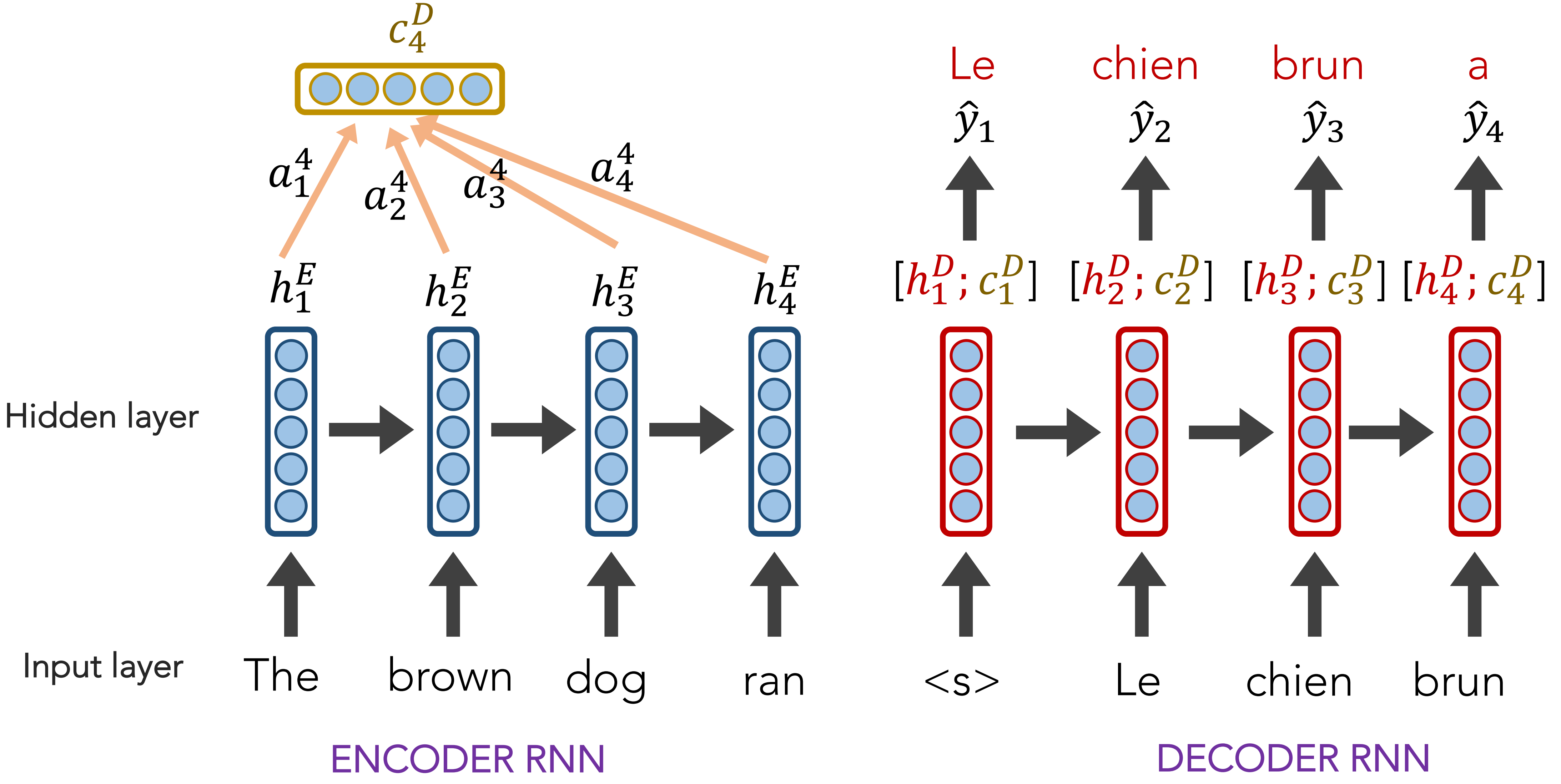
# seq2seq + Attention

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



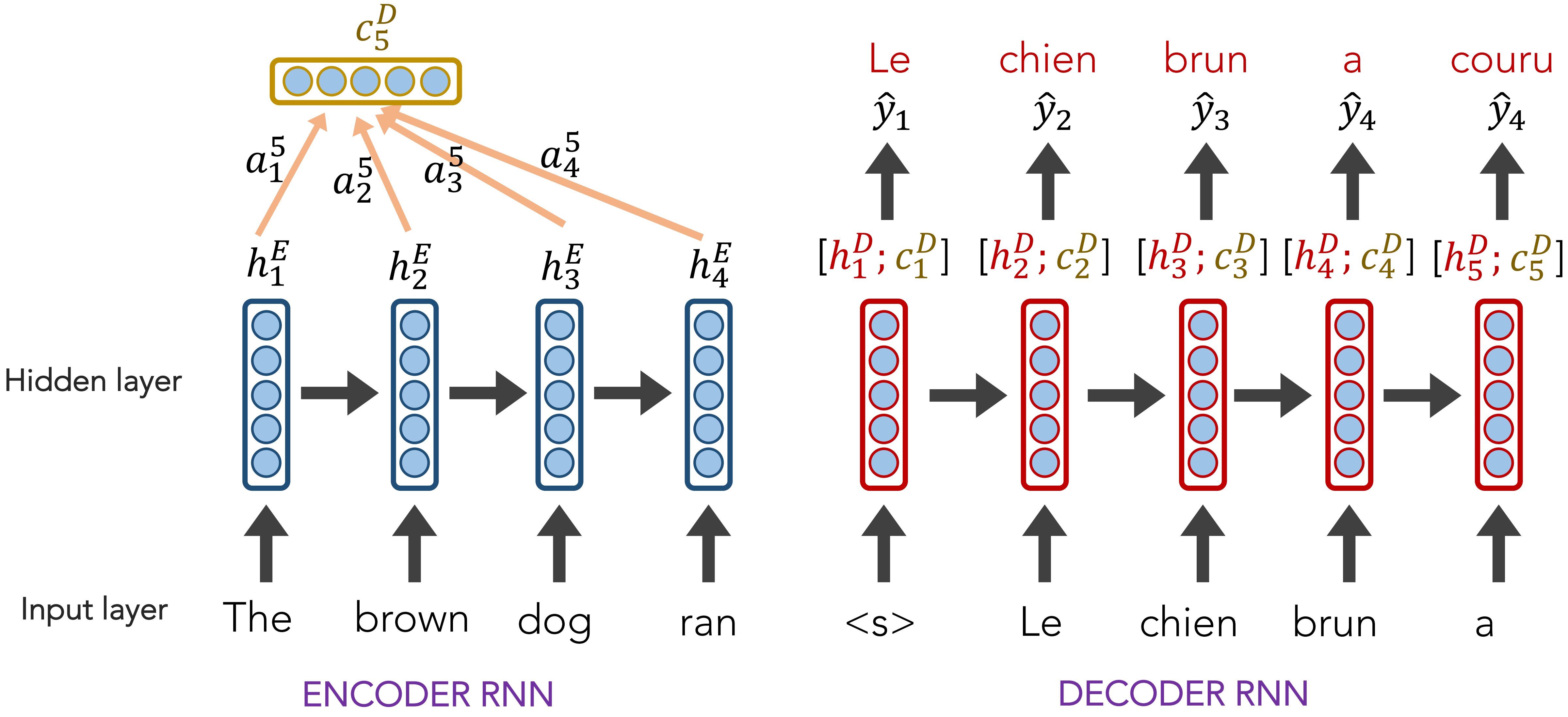
# seq2seq + Attention

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 $t$?"

↑

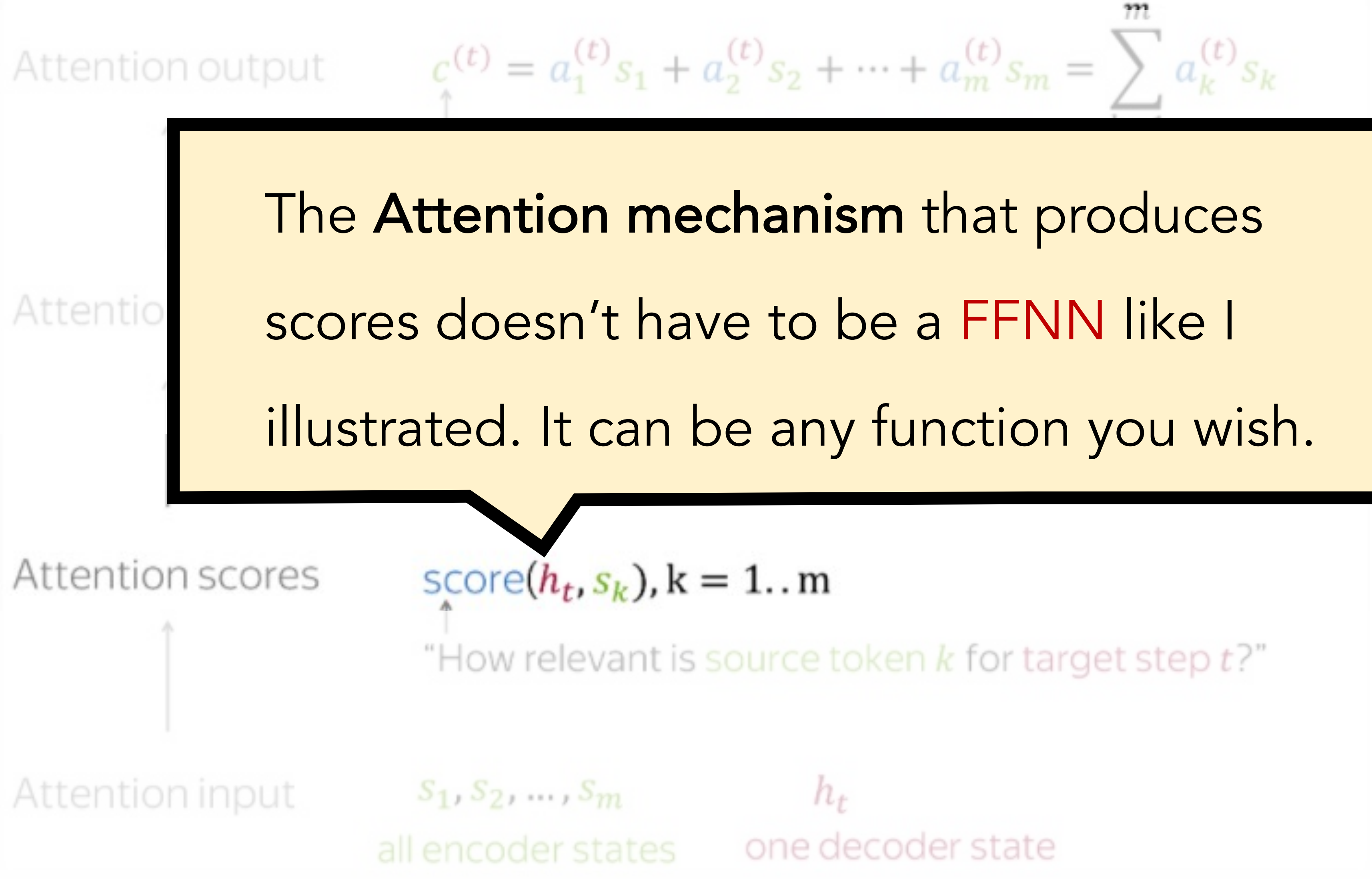
**Attention input**

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

$h_t$ — one decoder state

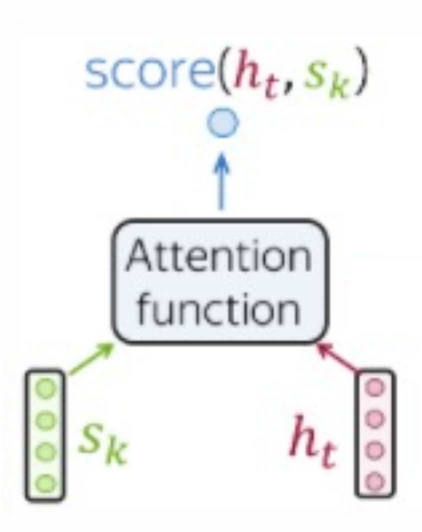
Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

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:

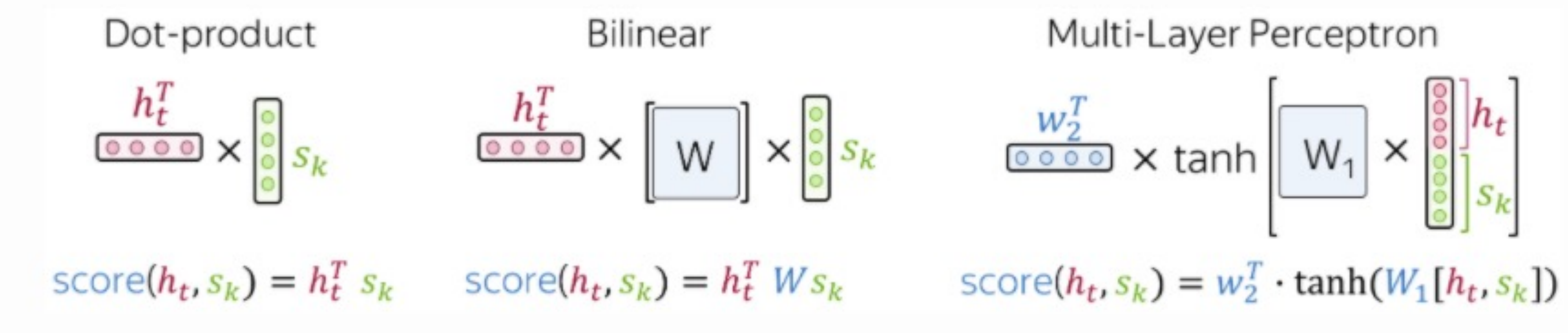
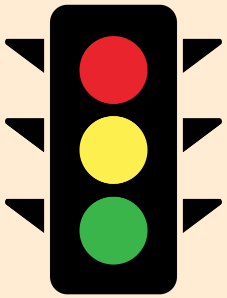




**Photo credit**: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

# 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)

# RECAP 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…

# RECAP SUMMARY

- LSTMs are sequential in nature (prohibits parallelization). Very wasteful.
- No <u>explicit</u> modelling of long- and short- range dependencies
- Language is naturally sequential, with hierarchical structure of meaning (can we do better than Stacked-LSTMs?)
- Can we apply the concept of Attention to improve our **representations**? (i.e., *contextualized representations*)

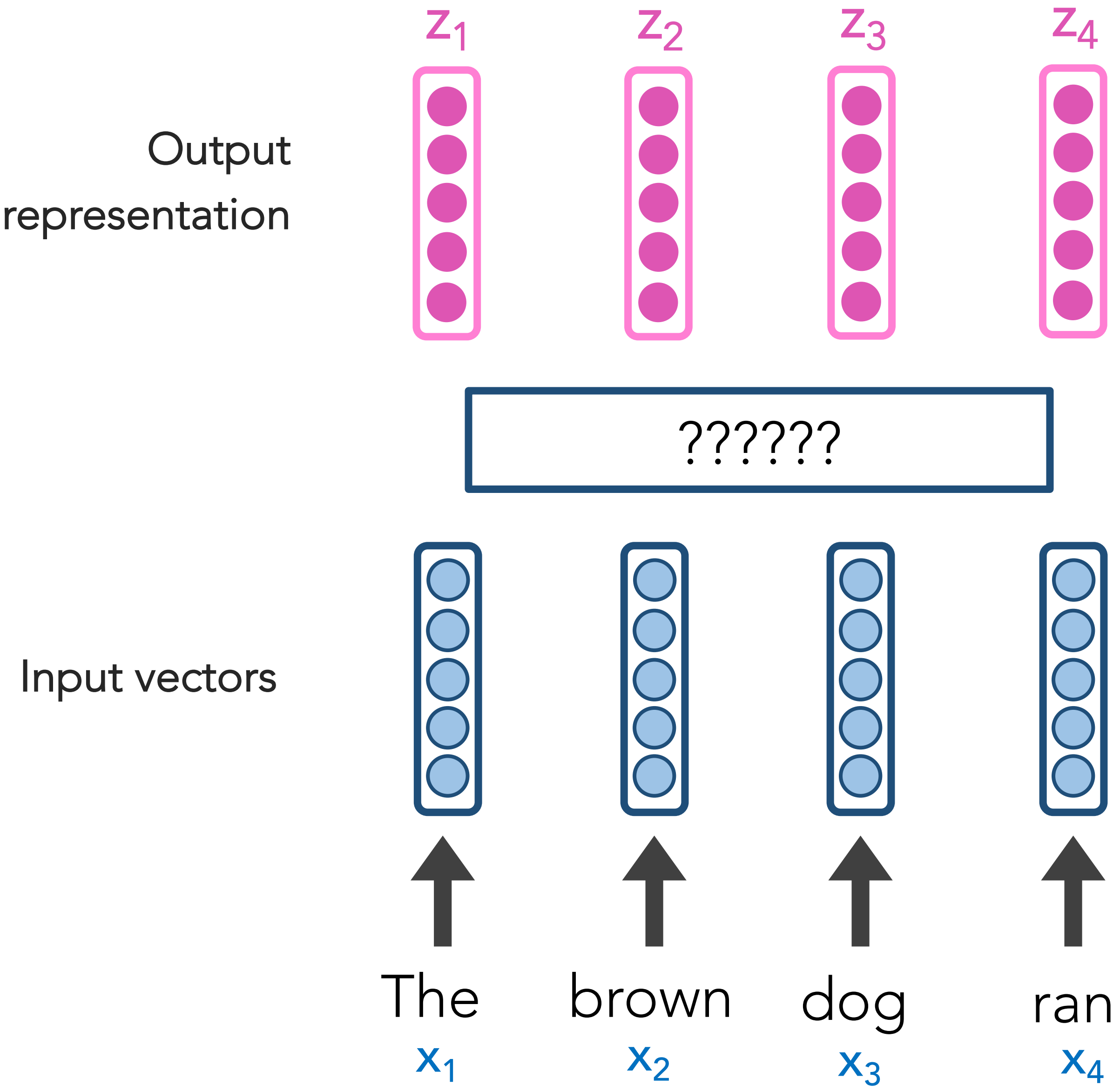
# Goals

- 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

# Outline

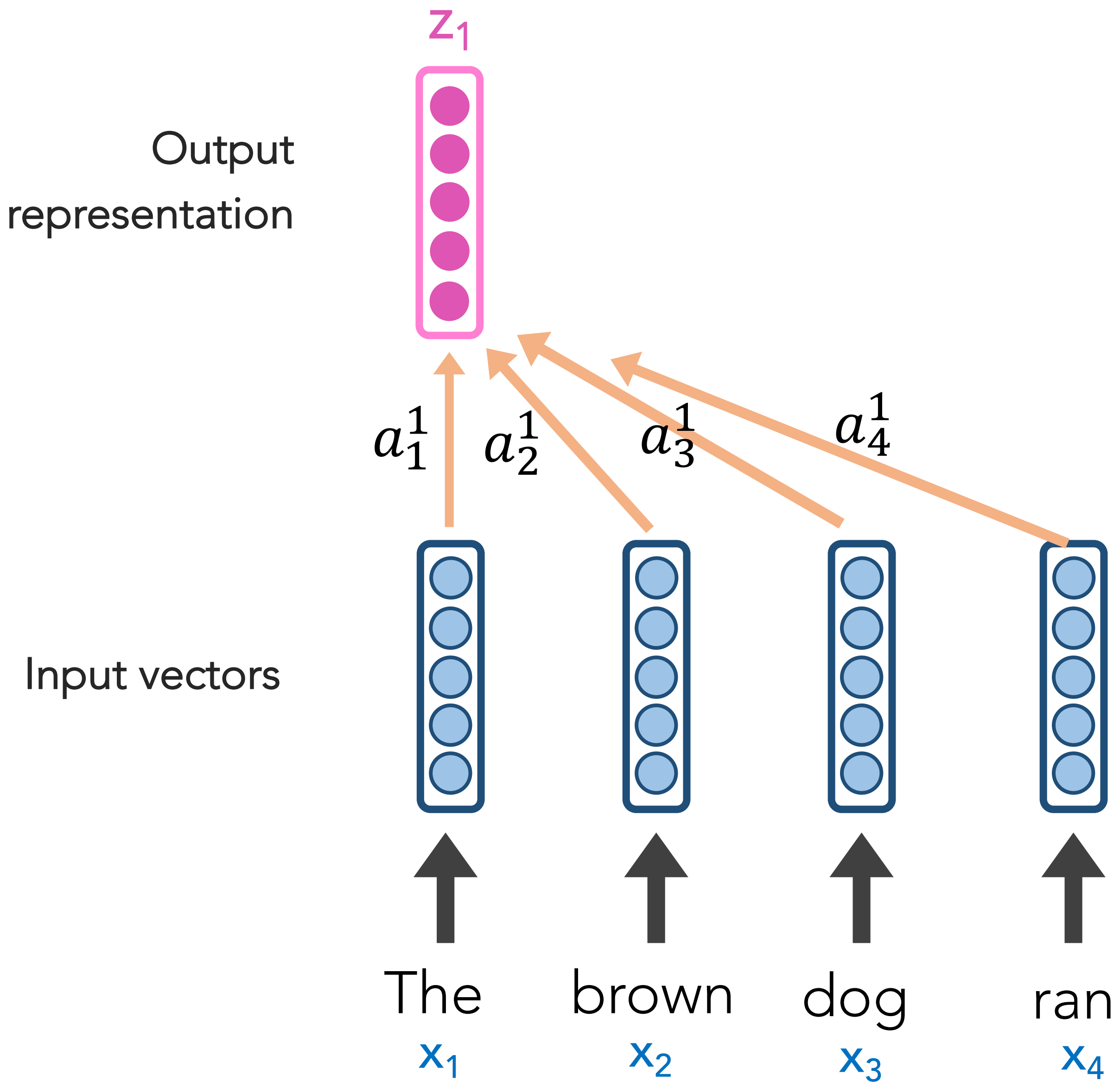
- Self-Attention
- Transformer

# Self-Attention




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

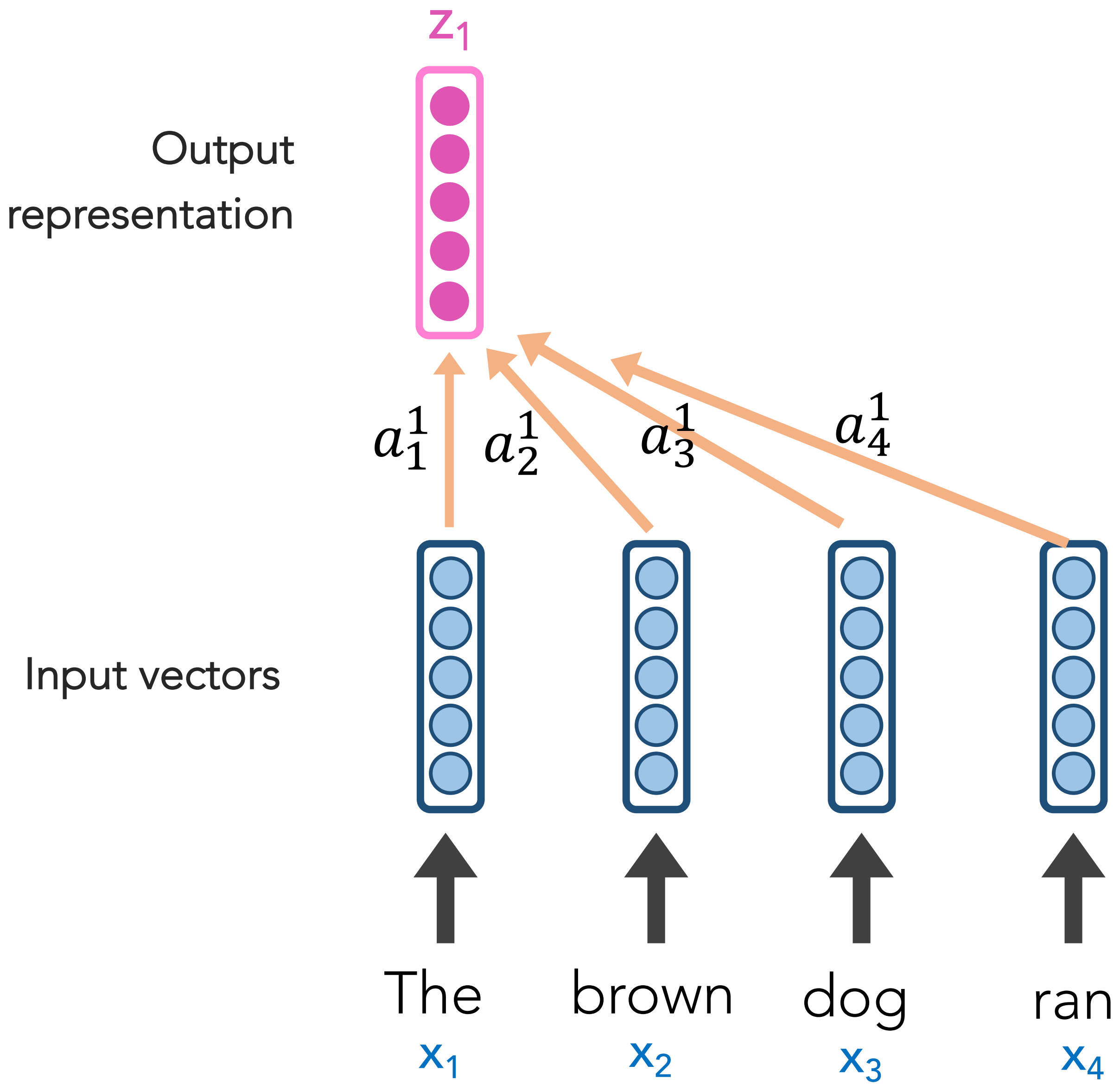
# Self-Attention




**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

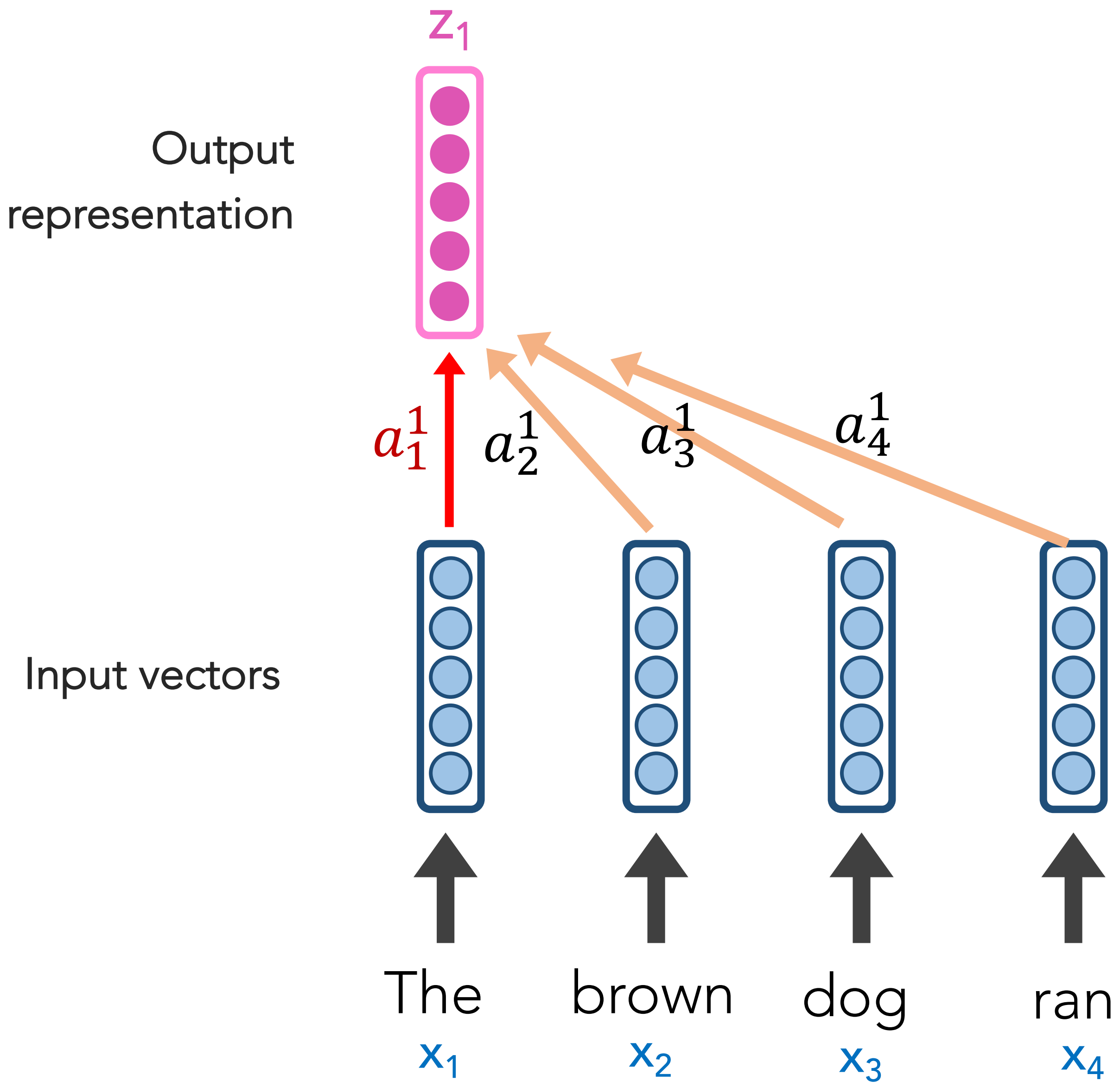



**Self-Attention**'s goal is to create great representations, $z_i$, of the input

$z_1$ 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 $\mathbf{v}_i$

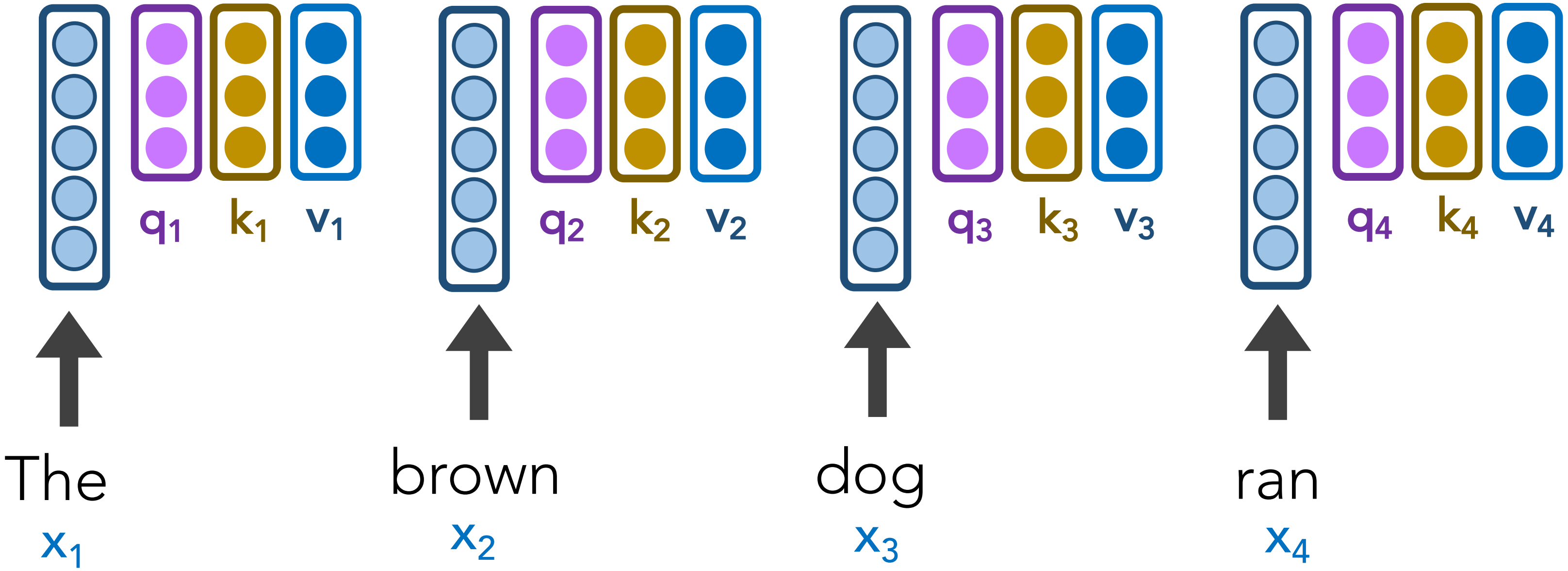
# Self-Attention

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:

$$q_i = W_q x_i$$

$$k_i = W_k x_i$$

$$v_i = W_v x_i$$



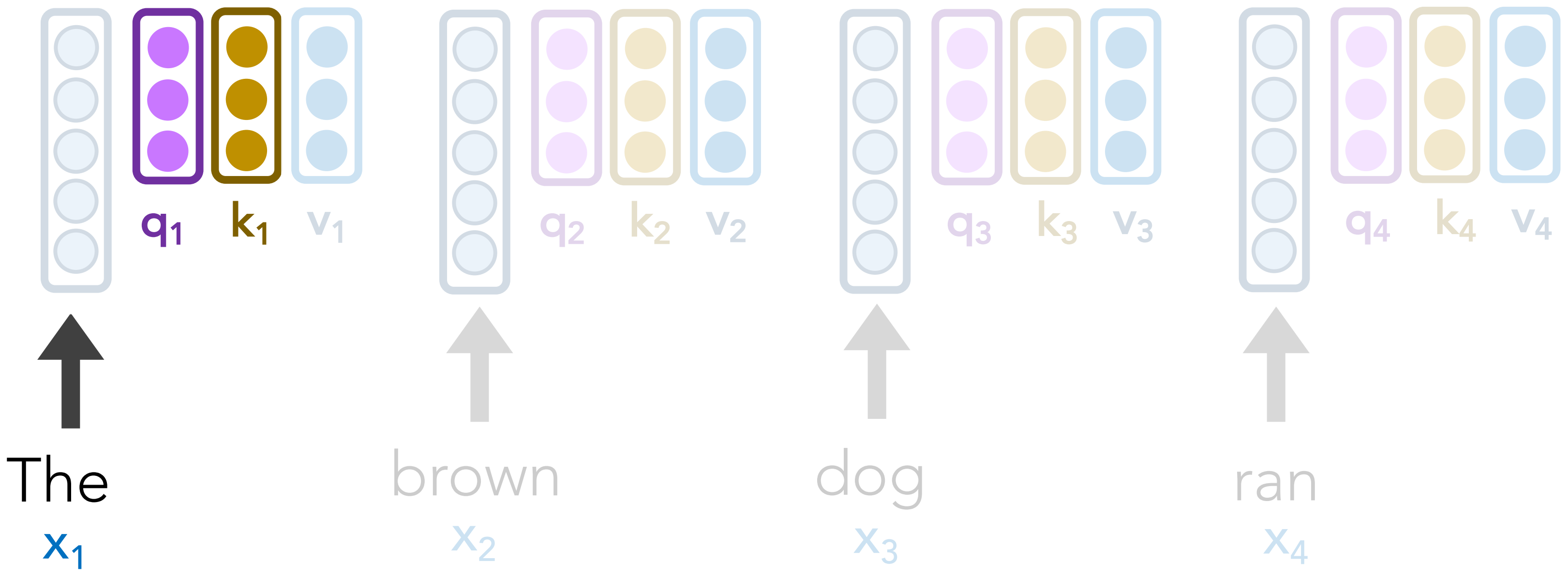

Under the hood, each $x_i$ has 3 small, associated vectors. For example, $x_1$ has:

- 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" $v_i$

$s_1 = q_1 \cdot k_1 = 112$

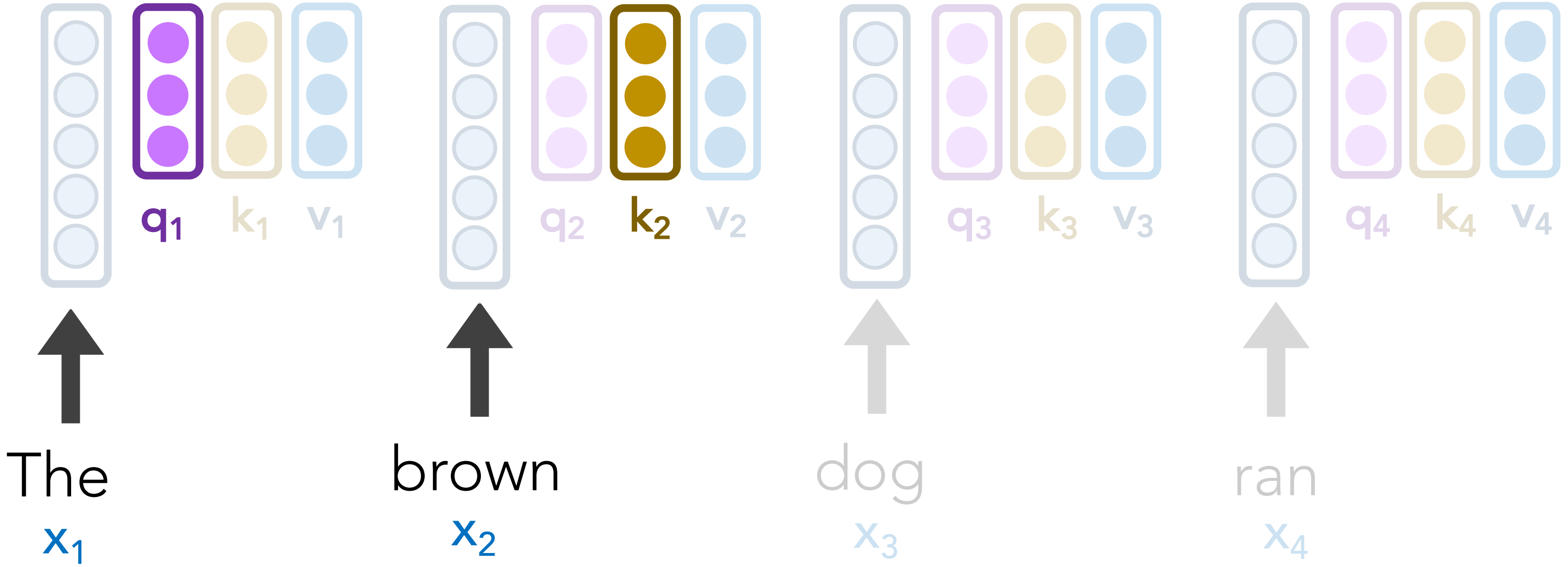


# Self-Attention

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" $v_i$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



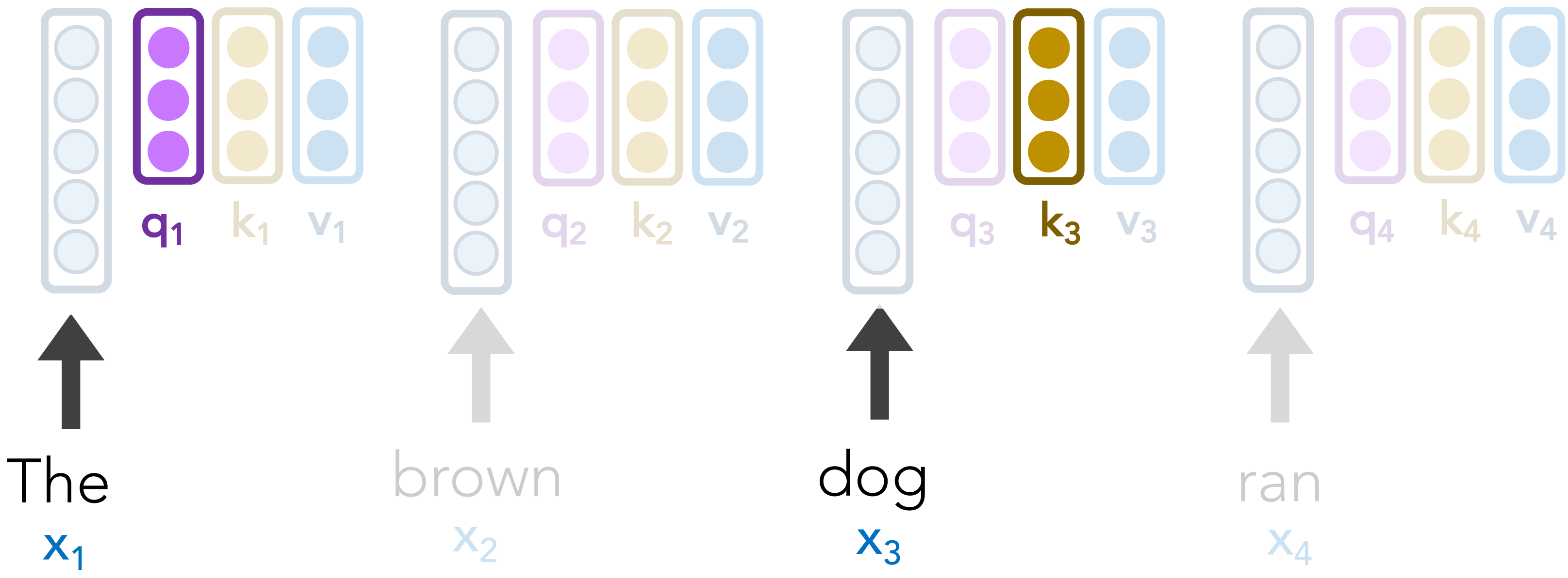
# Self-Attention

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

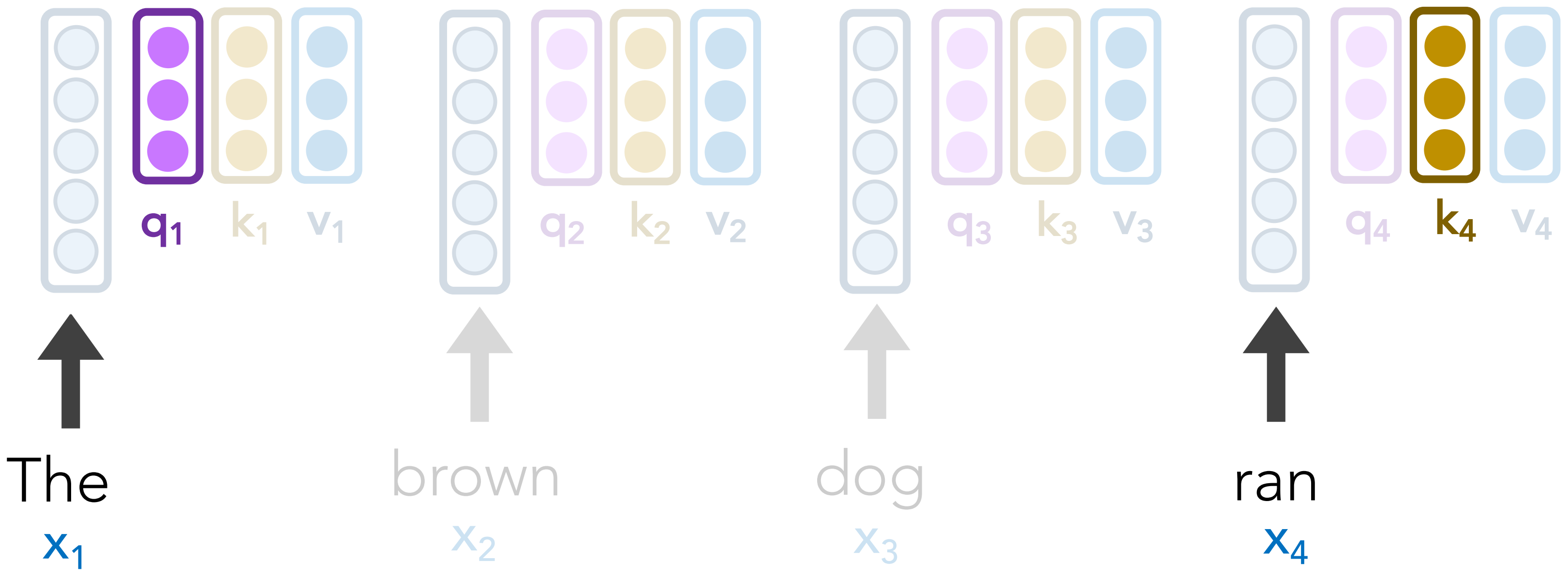
Step 2: For word $x_1$, let's calculate the scores $s_1$, $s_2$, $s_3$, $s_4$, which represent how much attention to pay to each respective "word" $v_i$

$s_4 = q_1 \cdot k_4 = 8$

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



# Self-Attention

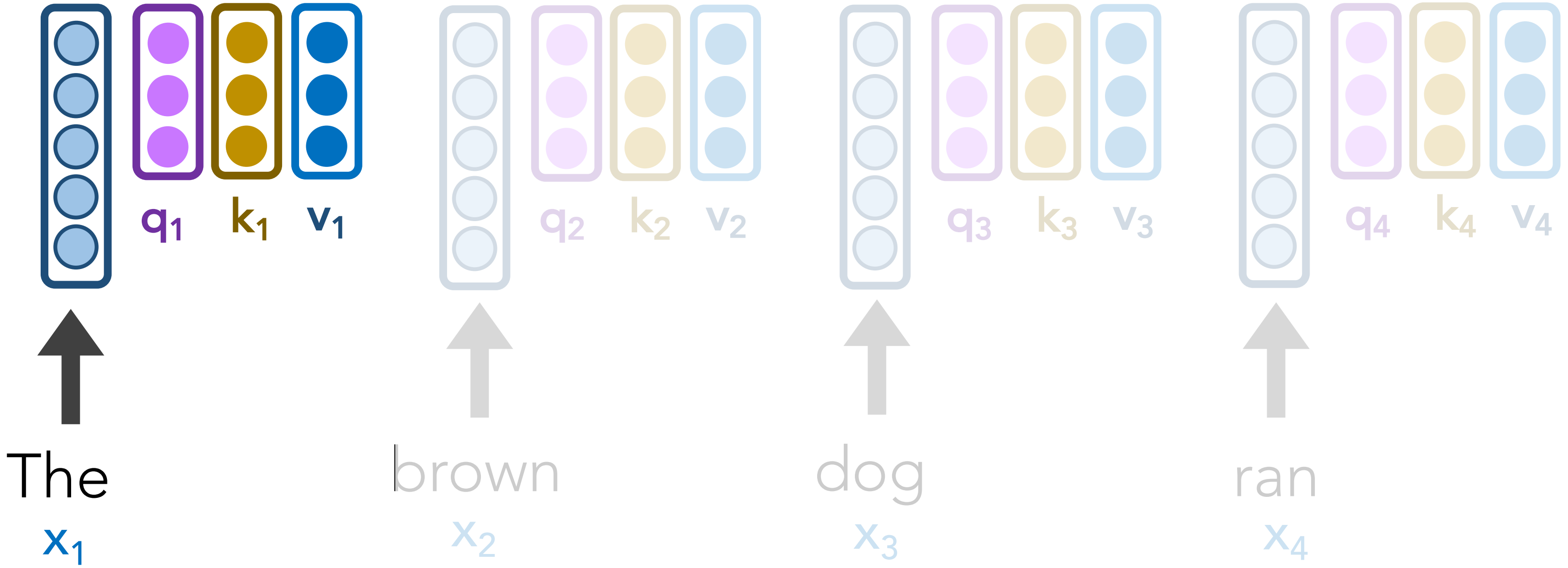
**Step 3:** Our scores $s_1, s_2, s_3, s_4$ don't sum to 1. Let's divide by $\sqrt{len(\boldsymbol{k_i})}$ and softmax() it

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

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

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

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



# 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

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

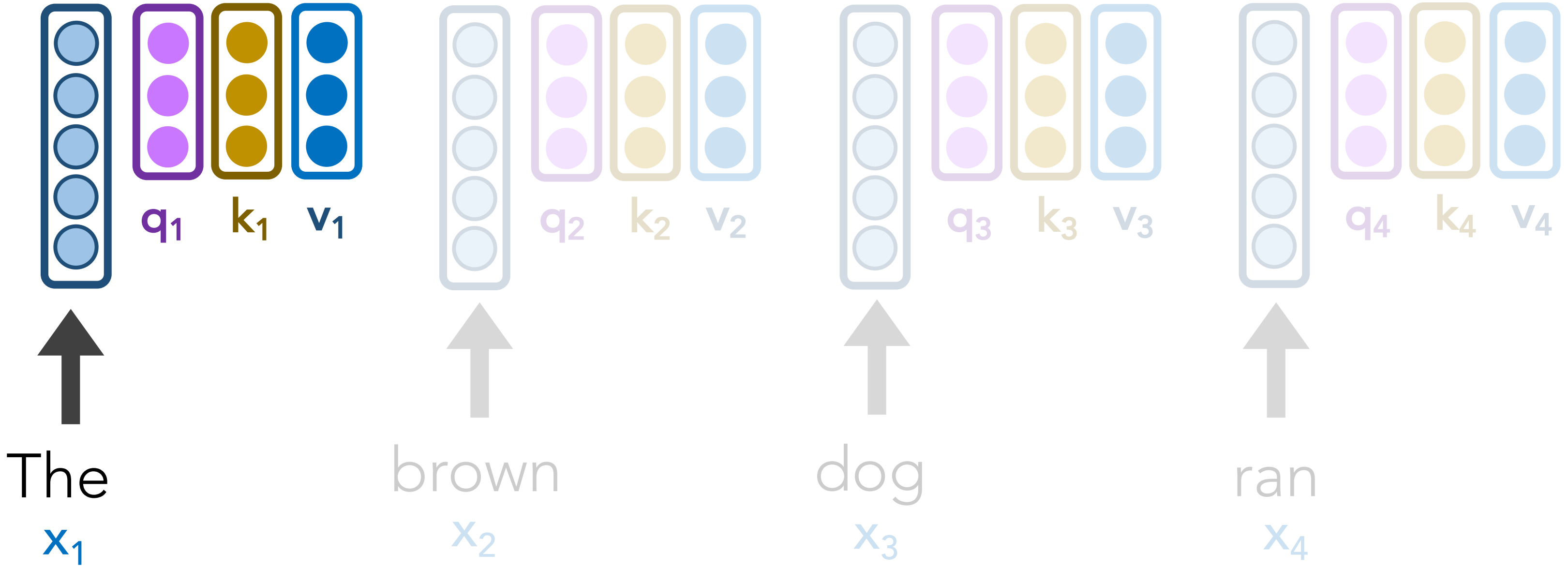
$s_3 = q_1 \cdot k_3 = 16$ $\quad a_3 = \sigma(s_3/8) = .01$

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

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

Dot-product of $q_i \cdot k_j$ grows large in magnitude; thus, inputs to softmax() can be large, and in turn yield small gradients.

Dividing by $\sqrt{len(k_i)}$ helps.



# 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

$s_4 = q_1 \cdot k_4 = 8$

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$

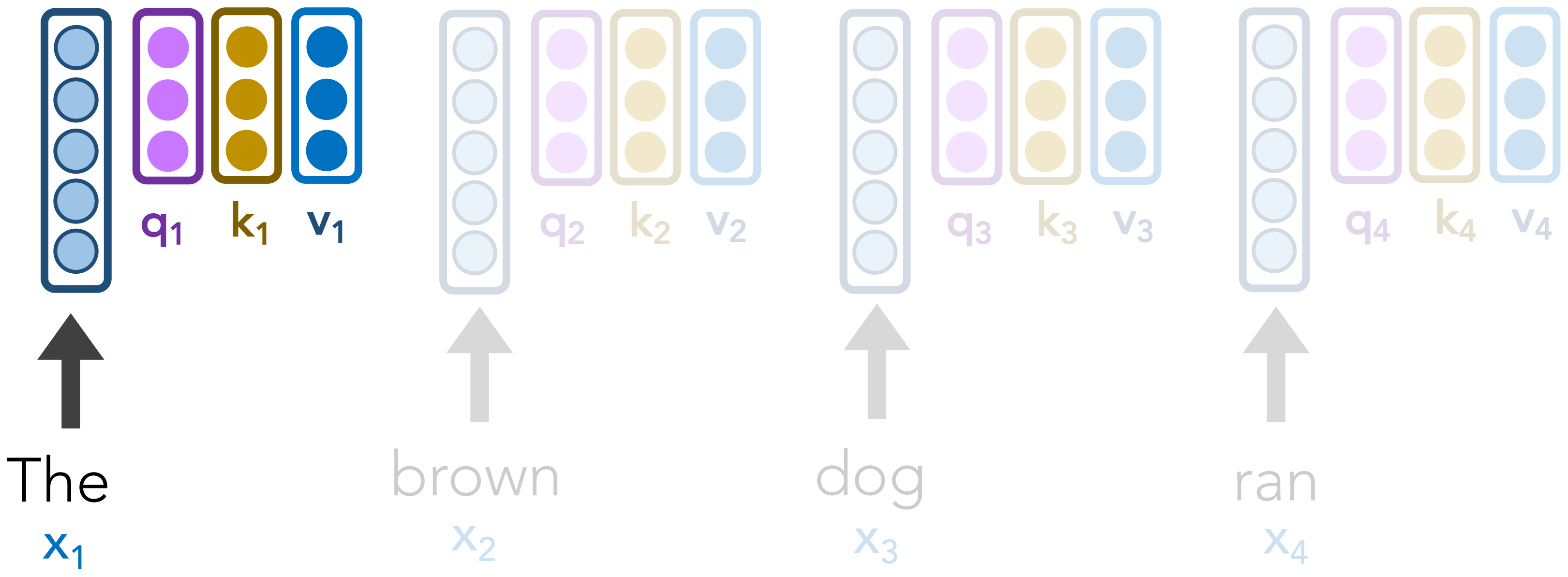
$a_4 = \sigma(s_4/8) = 0$

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

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

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

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

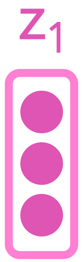


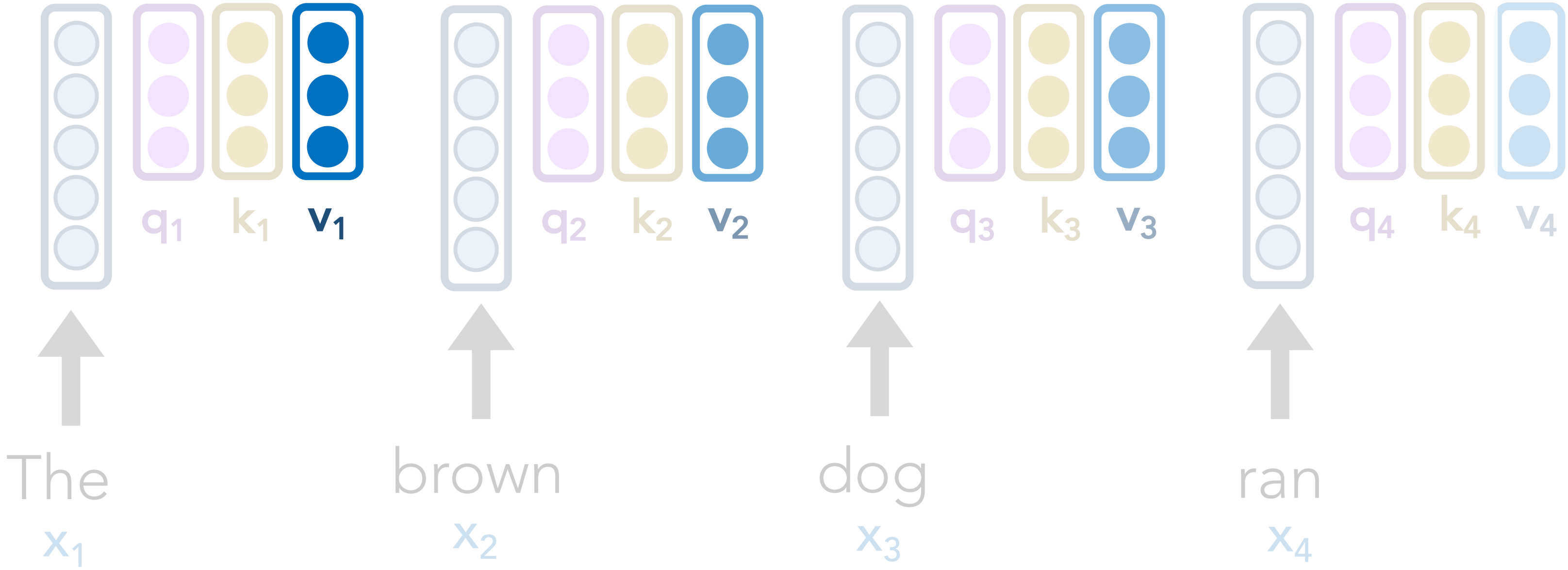
# Self-Attention

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

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

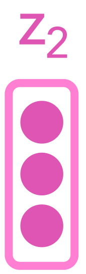
$$= 0.87 \cdot v_1 + 0.12 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



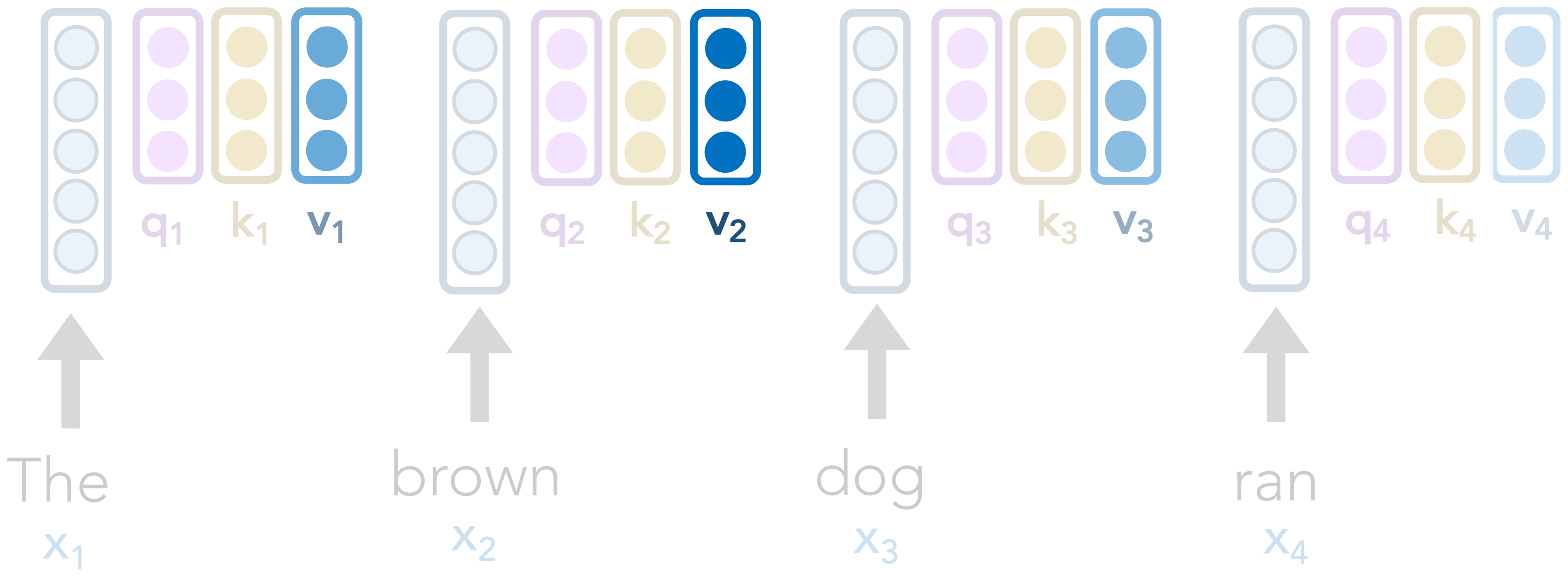



# Self-Attention

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

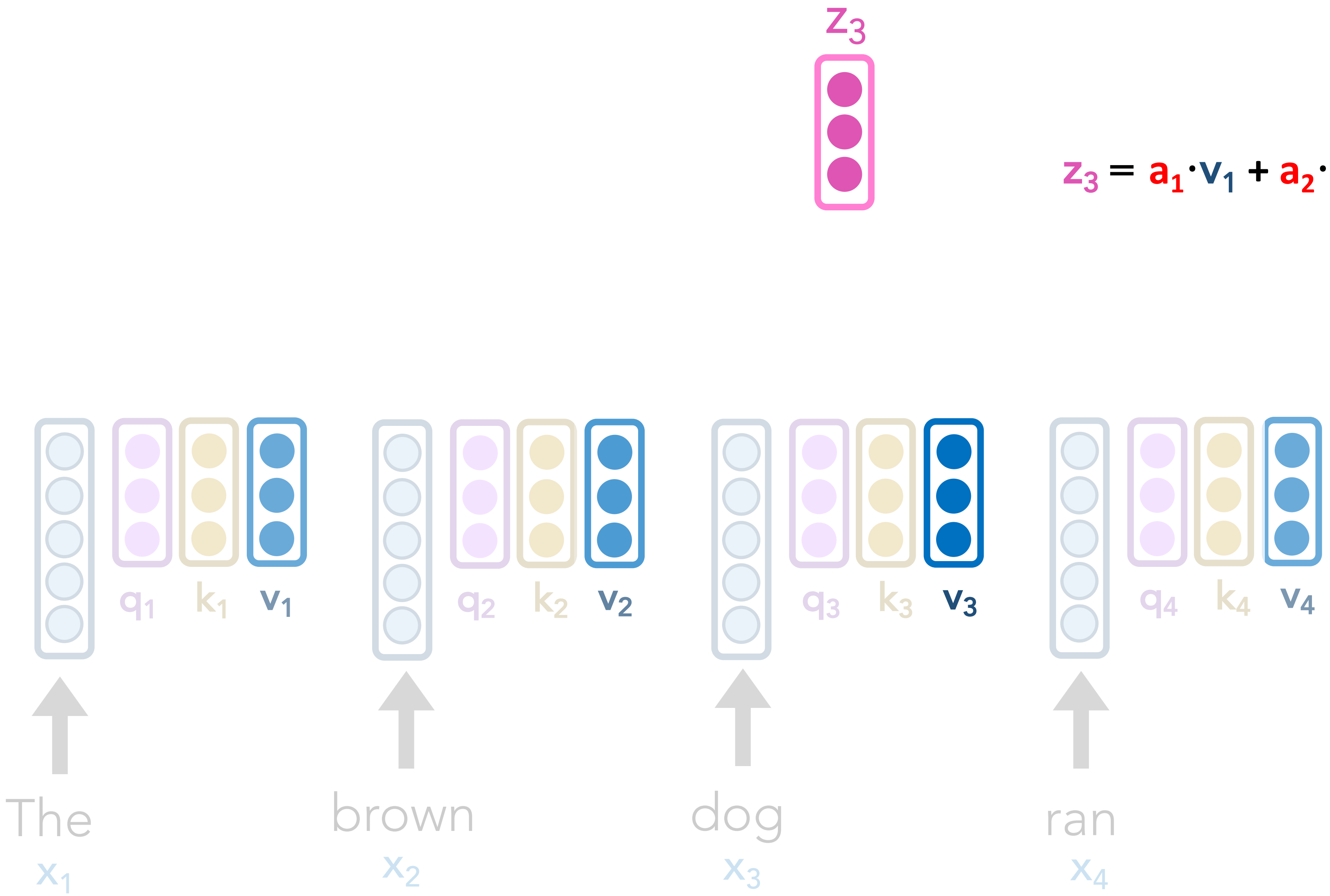



$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$



# Self-Attention

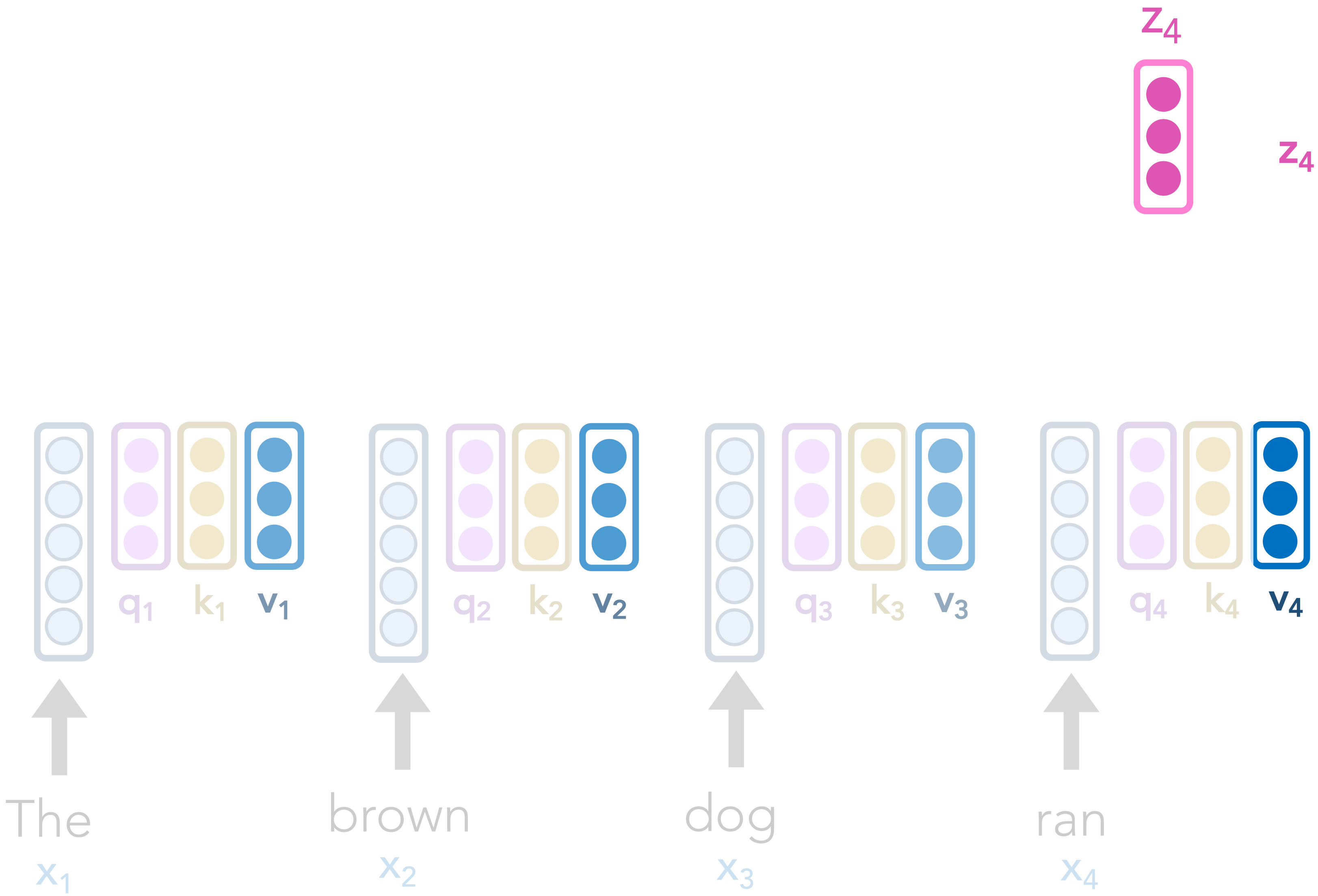
Step 5: We repeat this for all other words, yielding us with great, new $z_i$ representations!




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

# Self-Attention

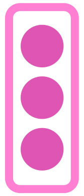
Step 5: We repeat this for all other words, yielding us with great, new $z_i$ representations!




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

# 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 $q_2$, $k_2$, $v_2$

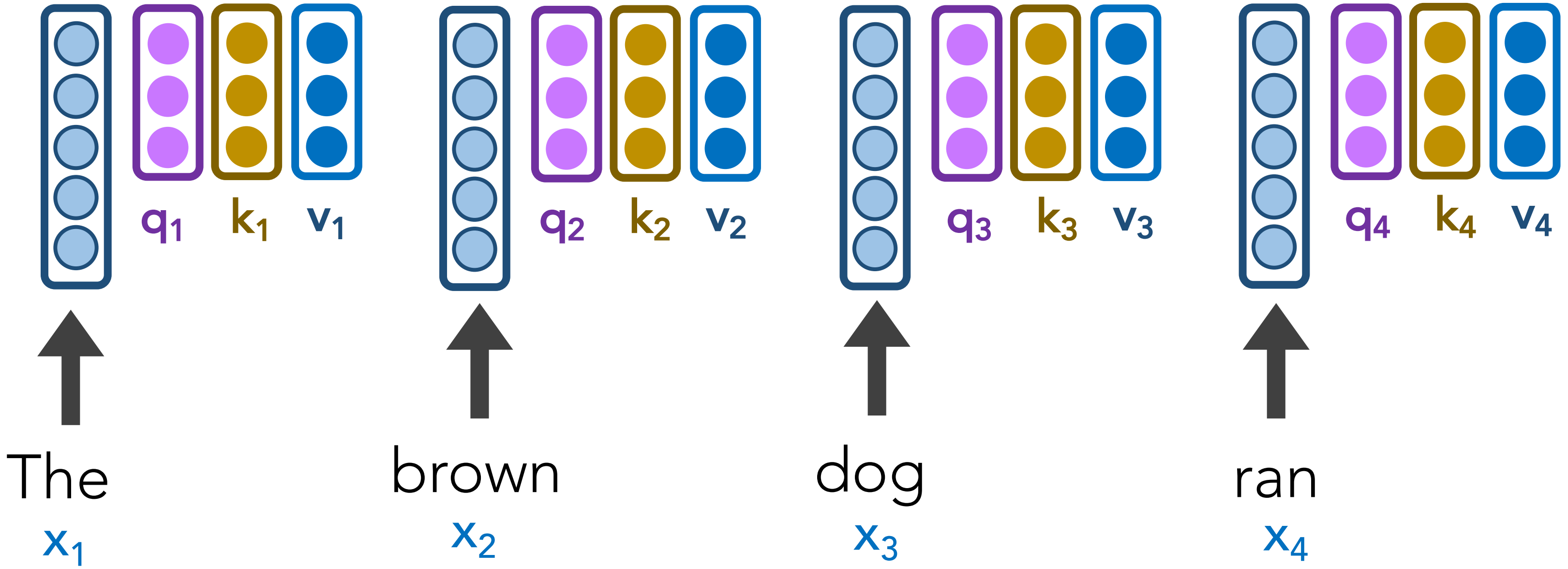
# Self-Attention

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:

$$q_i = W_q x_i$$

$$k_i = W_k x_i$$

$$v_i = W_v x_i$$



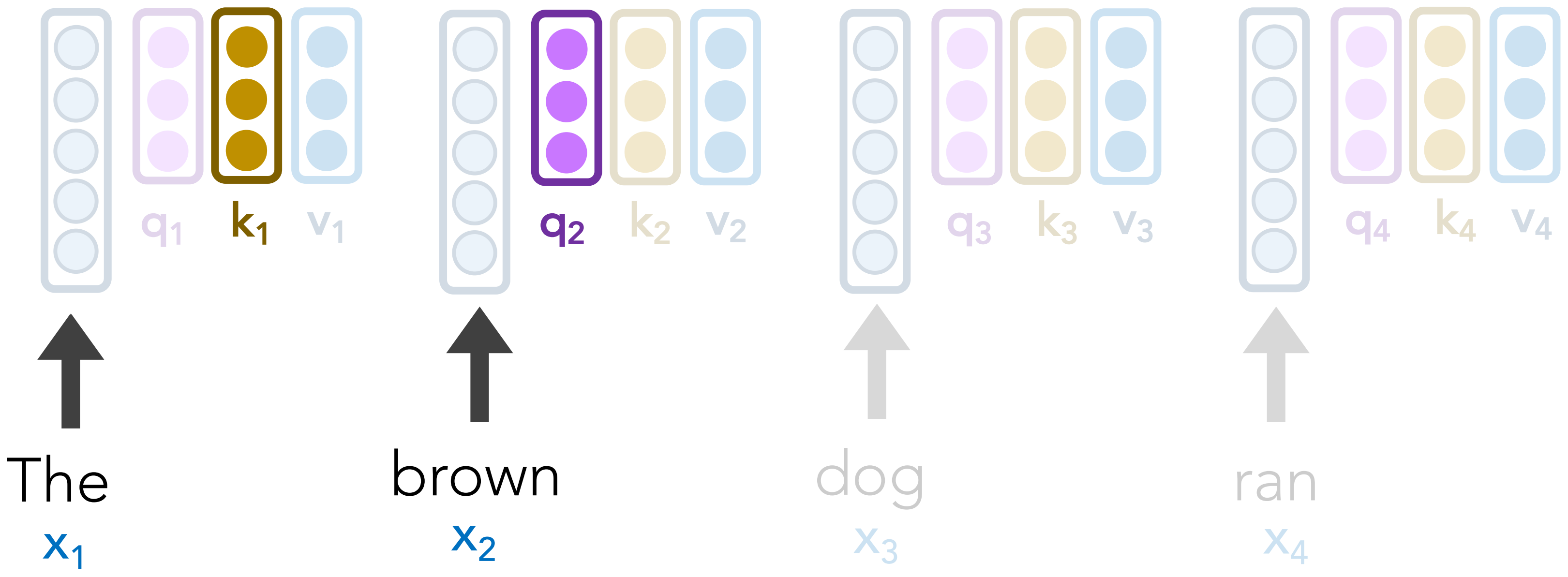

Under the hood, each $x_i$ has 3 small, associated vectors. For example, $x_1$ has:

- 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" $v_i$

$s_1 = q_2 \cdot k_1 = 92$

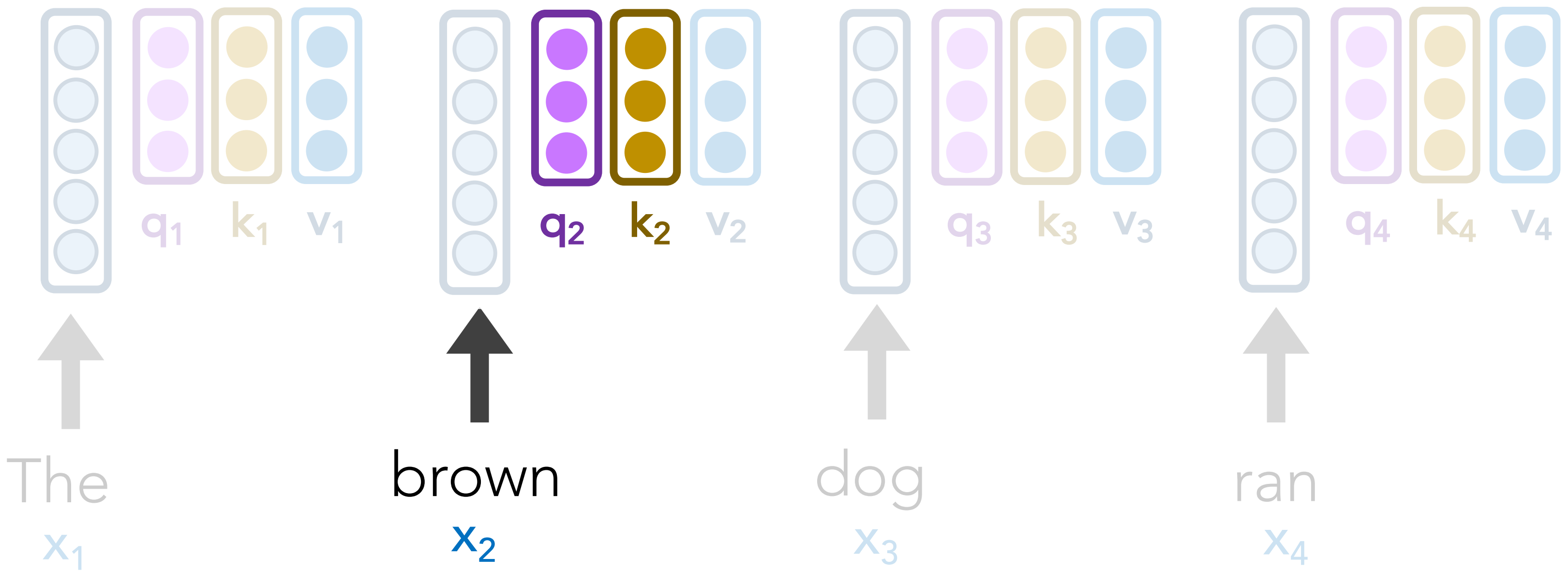


# Self-Attention

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" $v_i$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



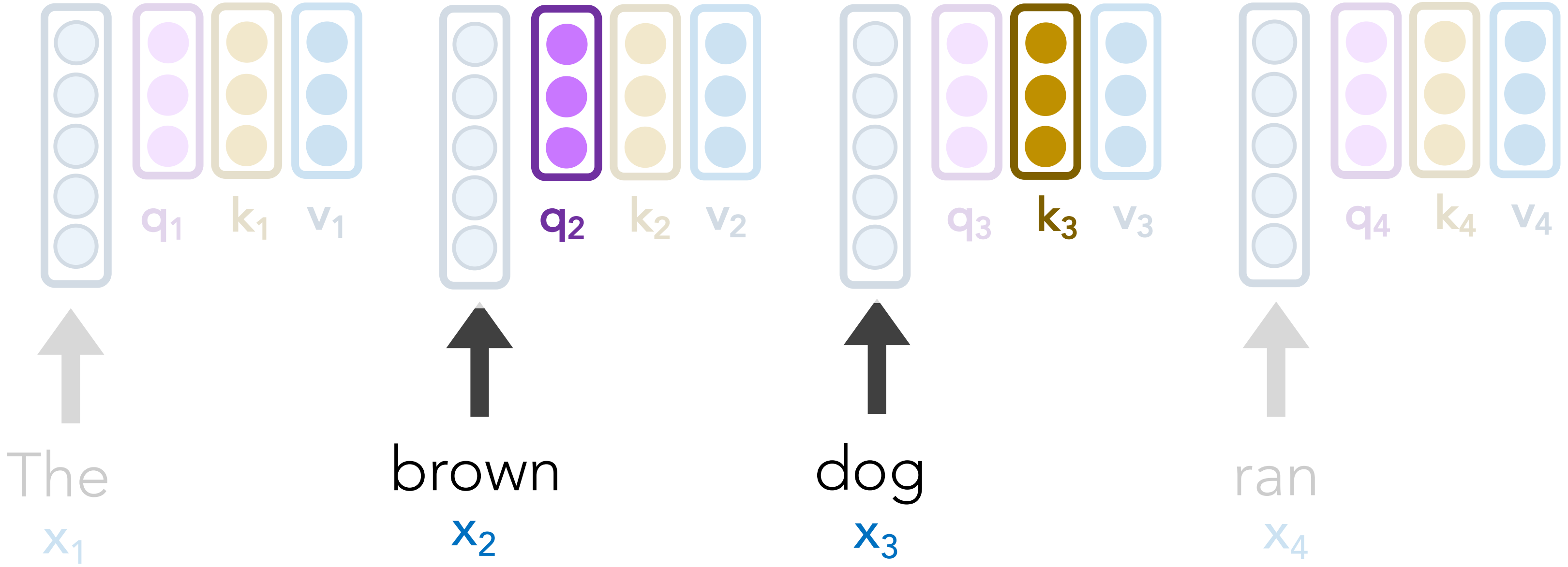
# Self-Attention

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

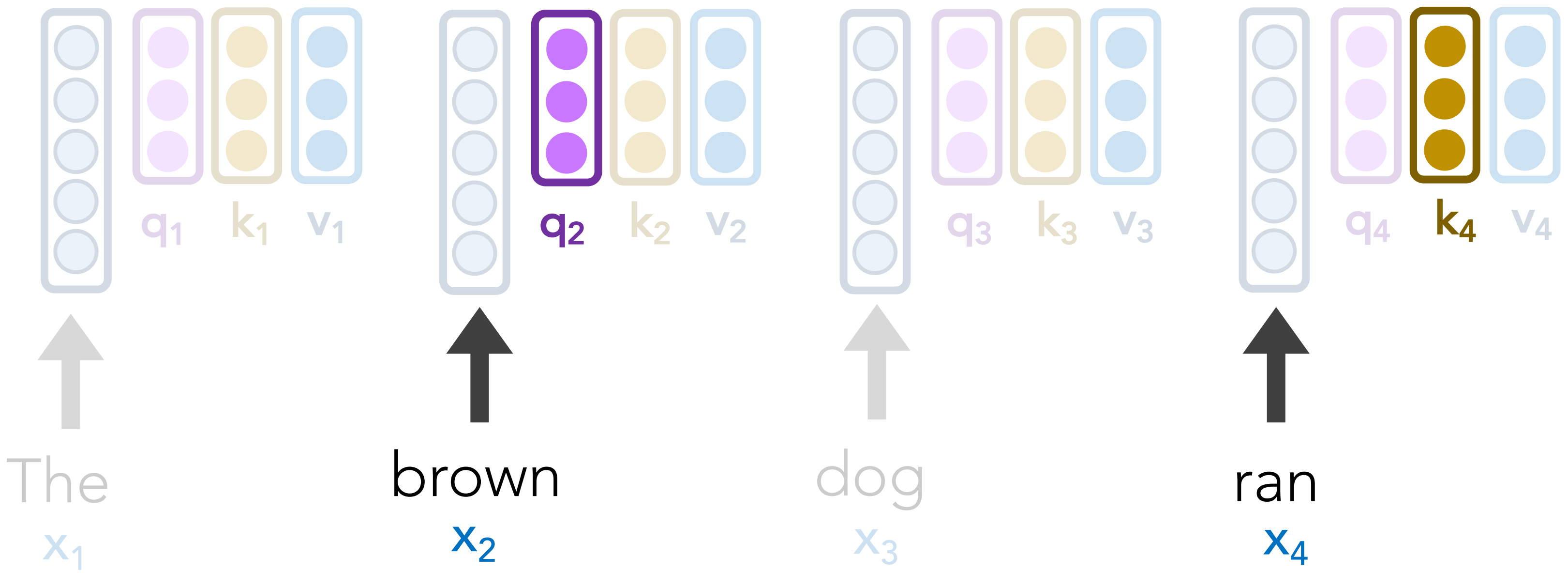
Step 2: For word $x_2$, let's calculate the scores $s_1$, $s_2$, $s_3$, $s_4$, which represent how much attention to pay to each respective "word" $v_i$

$s_4 = q_2 \cdot k_4 = 8$

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



# Self-Attention

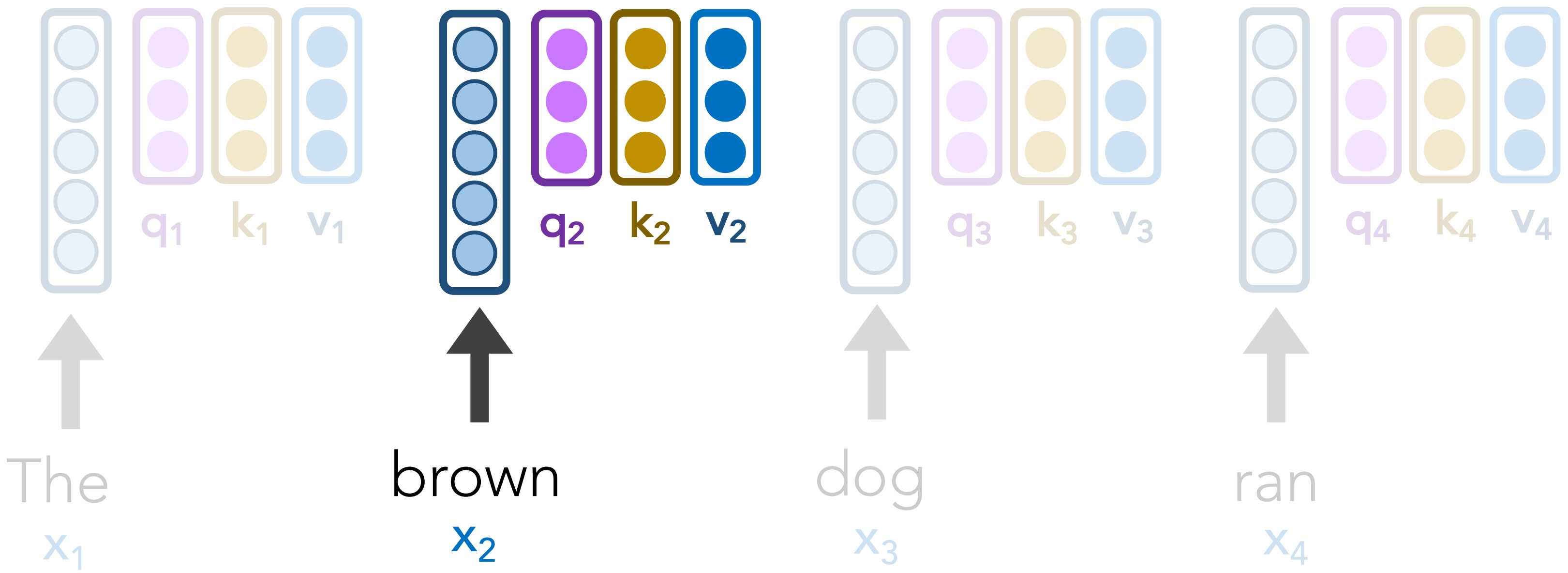
Step 3: Our scores $s_1, s_2, s_3, s_4$ don't sum to 1. Let's divide by $\sqrt{len(\boldsymbol{k_i})}$ and softmax() it

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

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

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

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



# 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

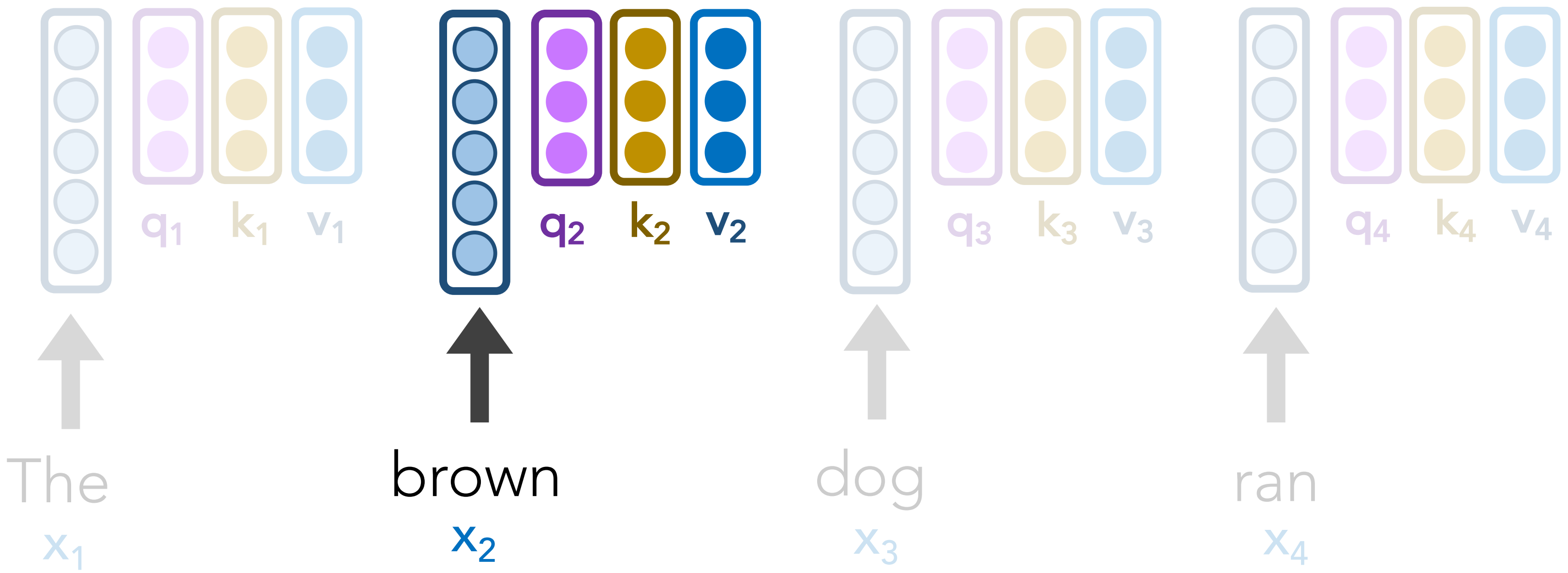
$s_4 = q_2 \cdot k_4 = 8$ $\qquad a_4 = \sigma(s_4/8) = 0$

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

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

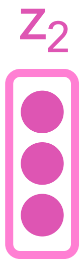
$s_1 = q_2 \cdot k_1 = 92$ $\qquad a_1 = \sigma(s_1/8) = .08$

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



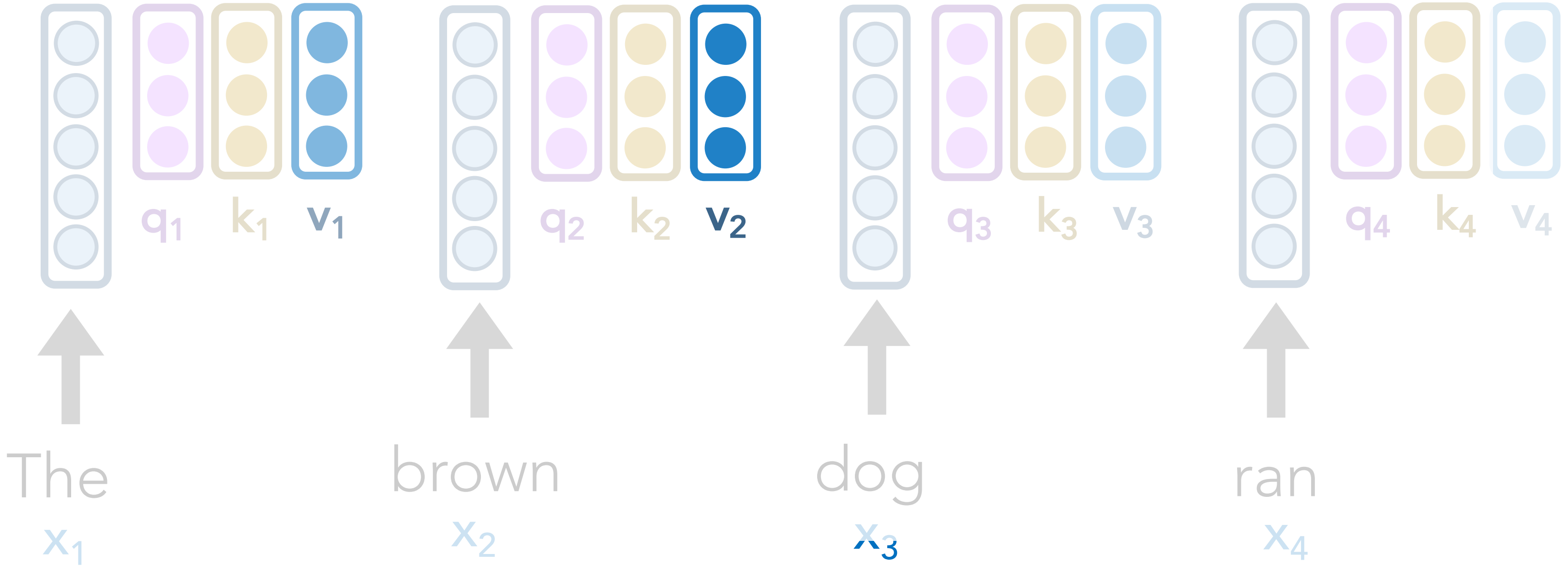
# Self-Attention

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



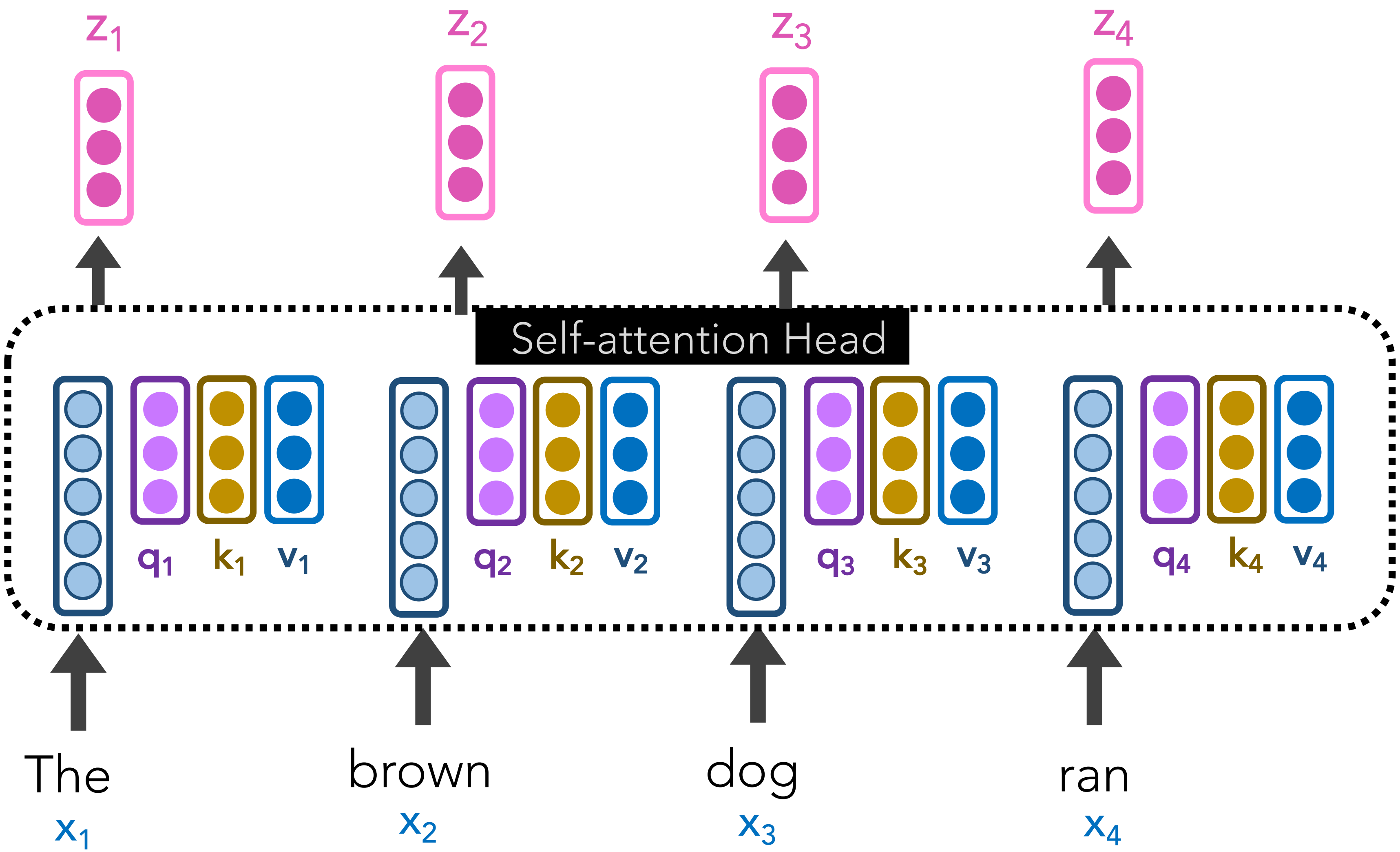

$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

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



# Self-Attention

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



# Self-Attention

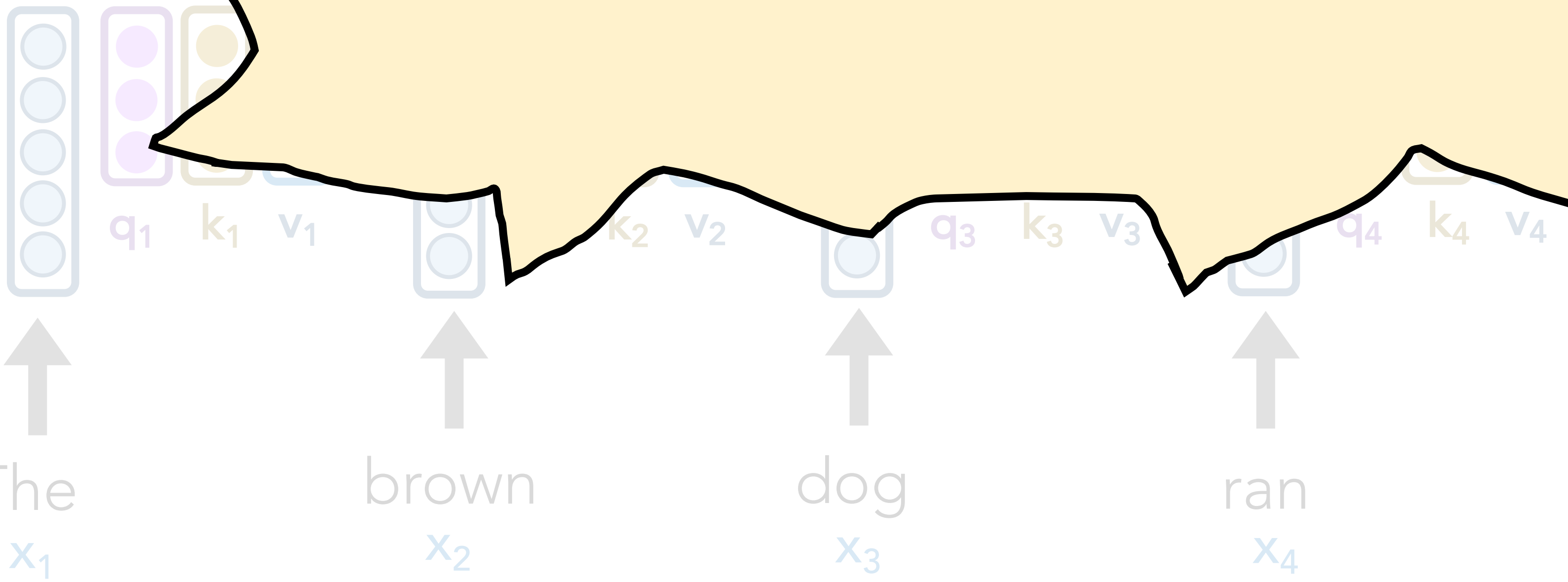
Implementation/technical detail:

All $z_i$'s can be calculated at the same time, via matrix multiplications

$$Z = softmax\left(\frac{Q * K^T}{\sqrt{len(k_o)}}\right) * V$$

Takeaway:

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



# Self-Attention may seem strikingly like Attention in seq2seq models

Q: What are the key, query, value vectors in the Attention setup?

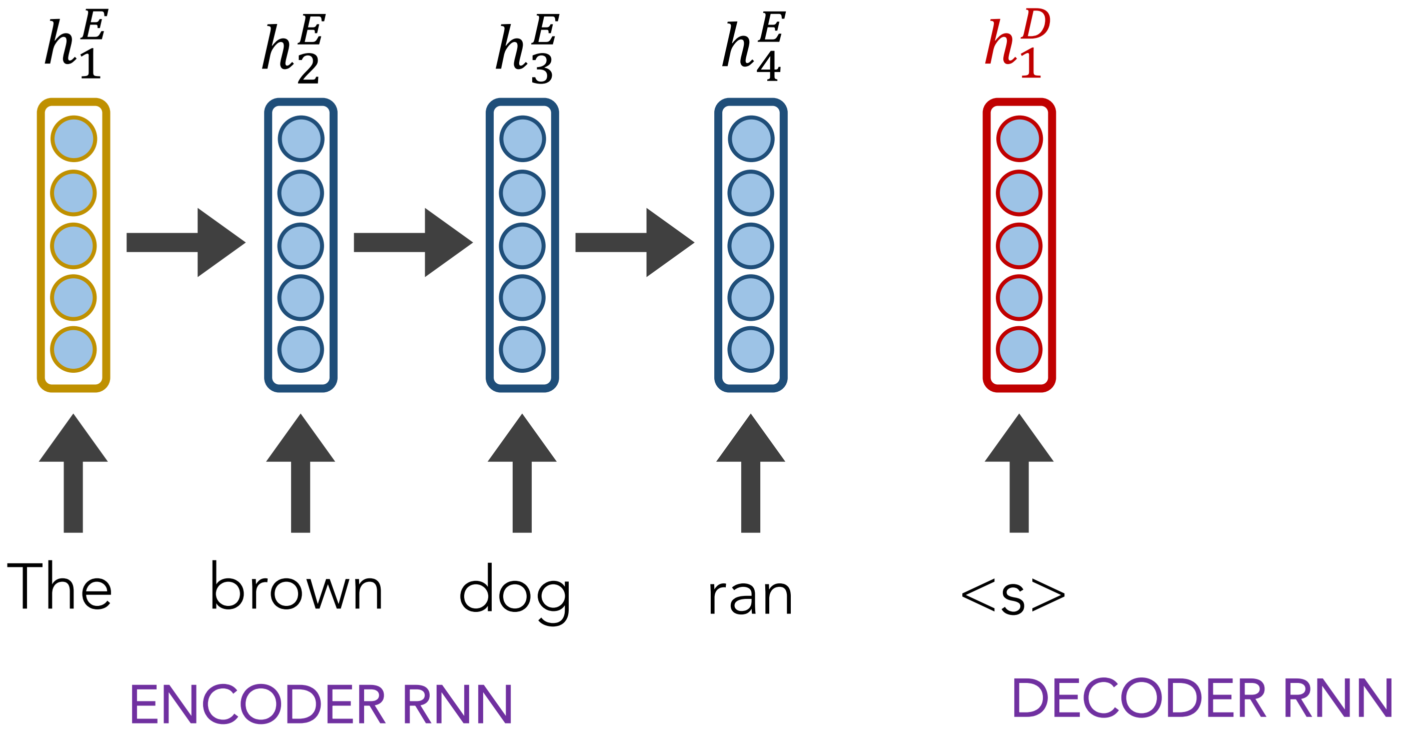
# Attention

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

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

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

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



# Attention

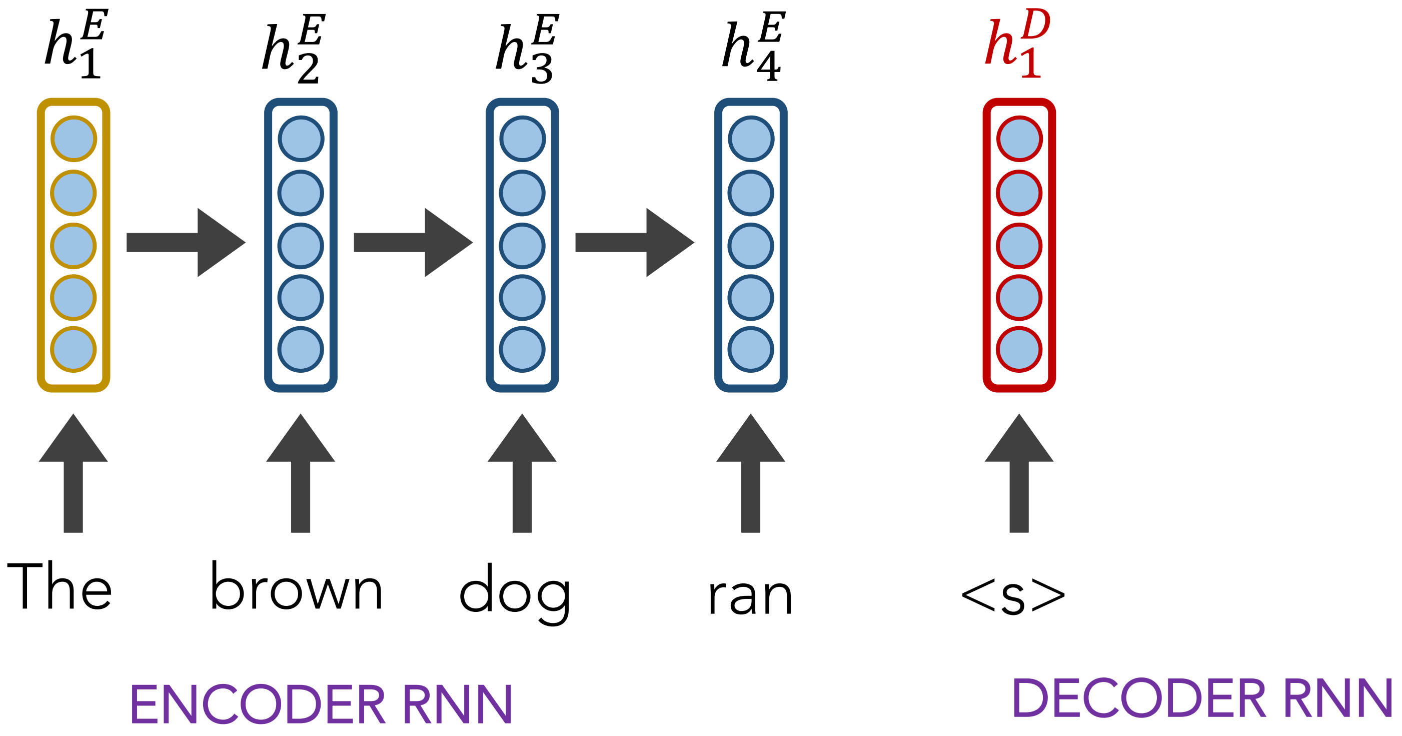
$s_4 = h_1^D * h_4^E$ $\quad a_4 = \sigma(s_4)$

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

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

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

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



# Attention

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

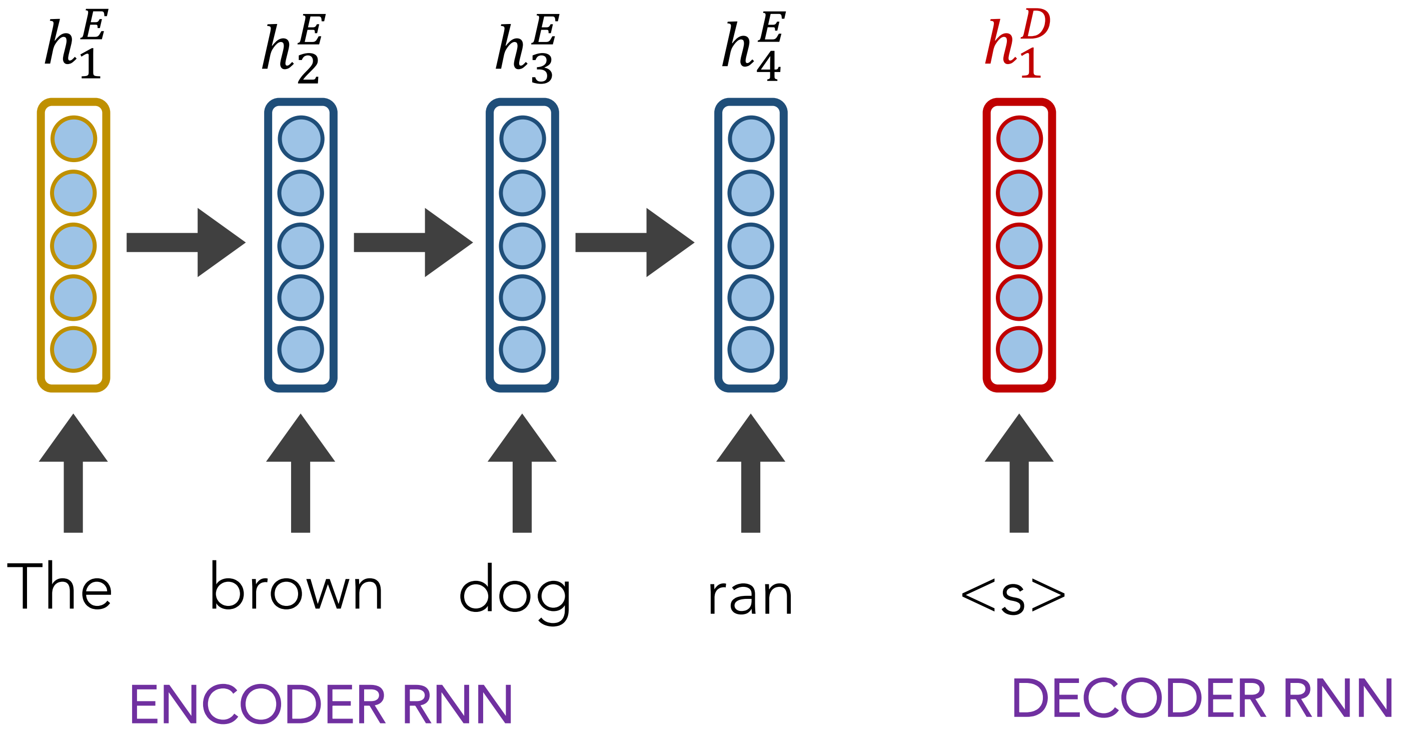
$s_3 = h_1^D * h_3^E$ $\quad$ $a_3 = \sigma(s_3)$

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

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

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

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



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

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

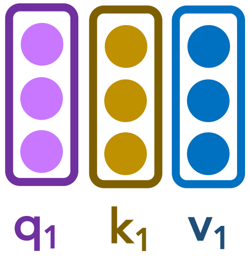
$s_2 = q_2 \cdot k_2$ $a_2 = \sigma(s_2/8)$

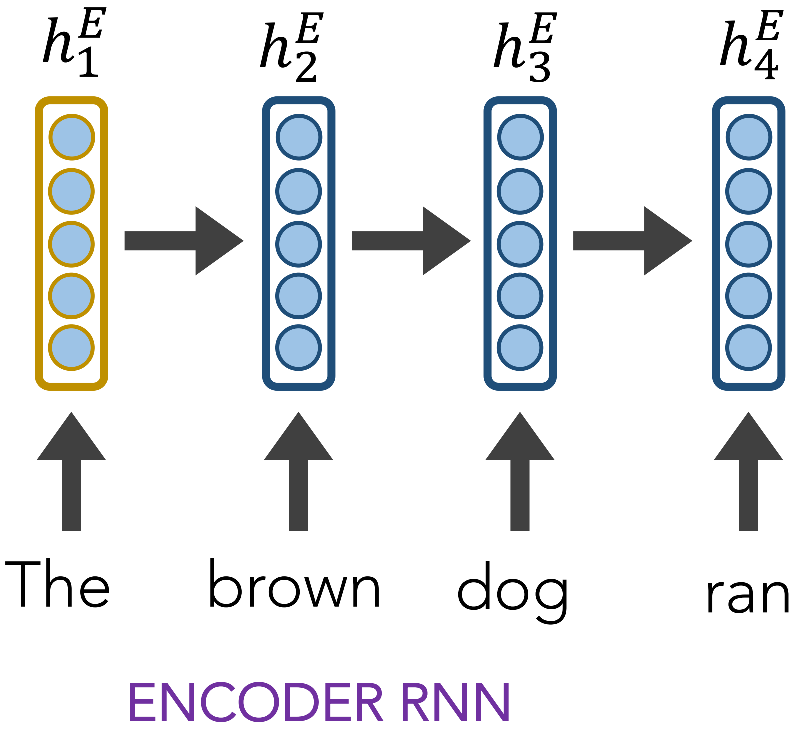
$s_1 = q_2 \cdot k_1$ $a_1 = \sigma(s_1/8)$

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



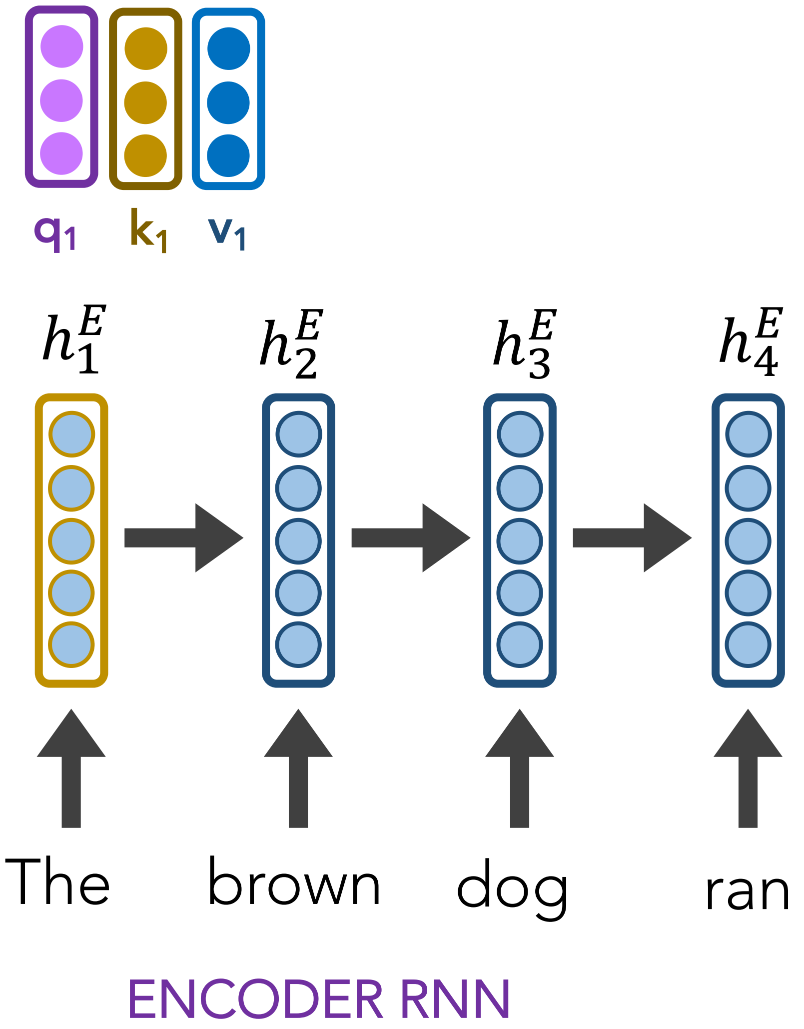



# Self-Attention

| 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 | Attention | Description |
|---|---|---|
| $q_i$ | $h_i^D$ | the probe (i.e., asks for information) |
| $k_i$ | $h_i^E$ | item being compared (i.e, answers "I have info) |
| $v_i$ | $h_i^E$ | item being weighted (i.e., gives the information) |

All of these are like surrogates/proxies/abstractions.

This provides flexibility and fewer constraints.

More room for rich abstractions.

# Outline

- Self-Attention
- Transformer

# Outline

Self-Attention

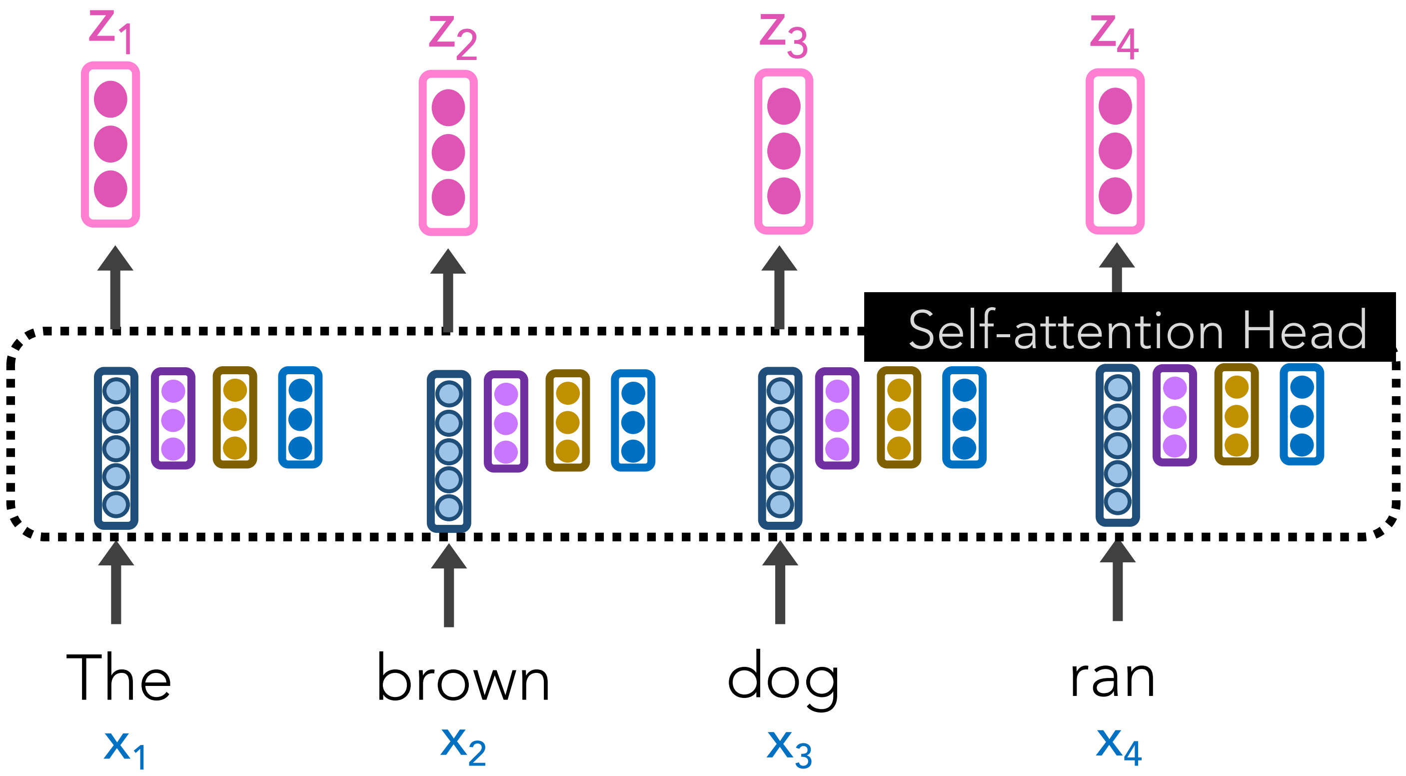
Transformer

# Transformer

[Attention is all you need](). Vaswani et al., 2017

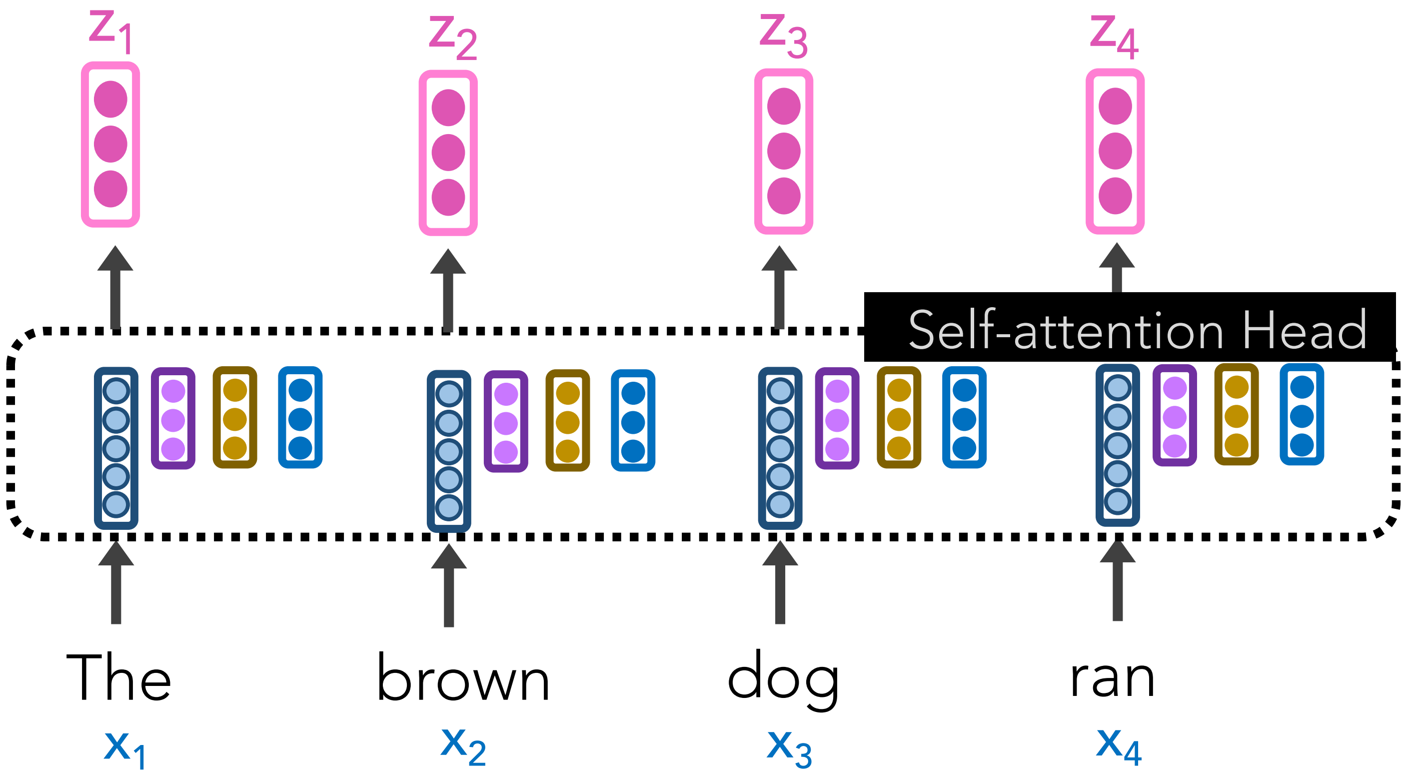
| CITED BY | YEAR |
|---|---|
| 90141 | 2017 |

# Self-Attention




**Q:** Do we see any shortcomings with this Self-Attention Head?

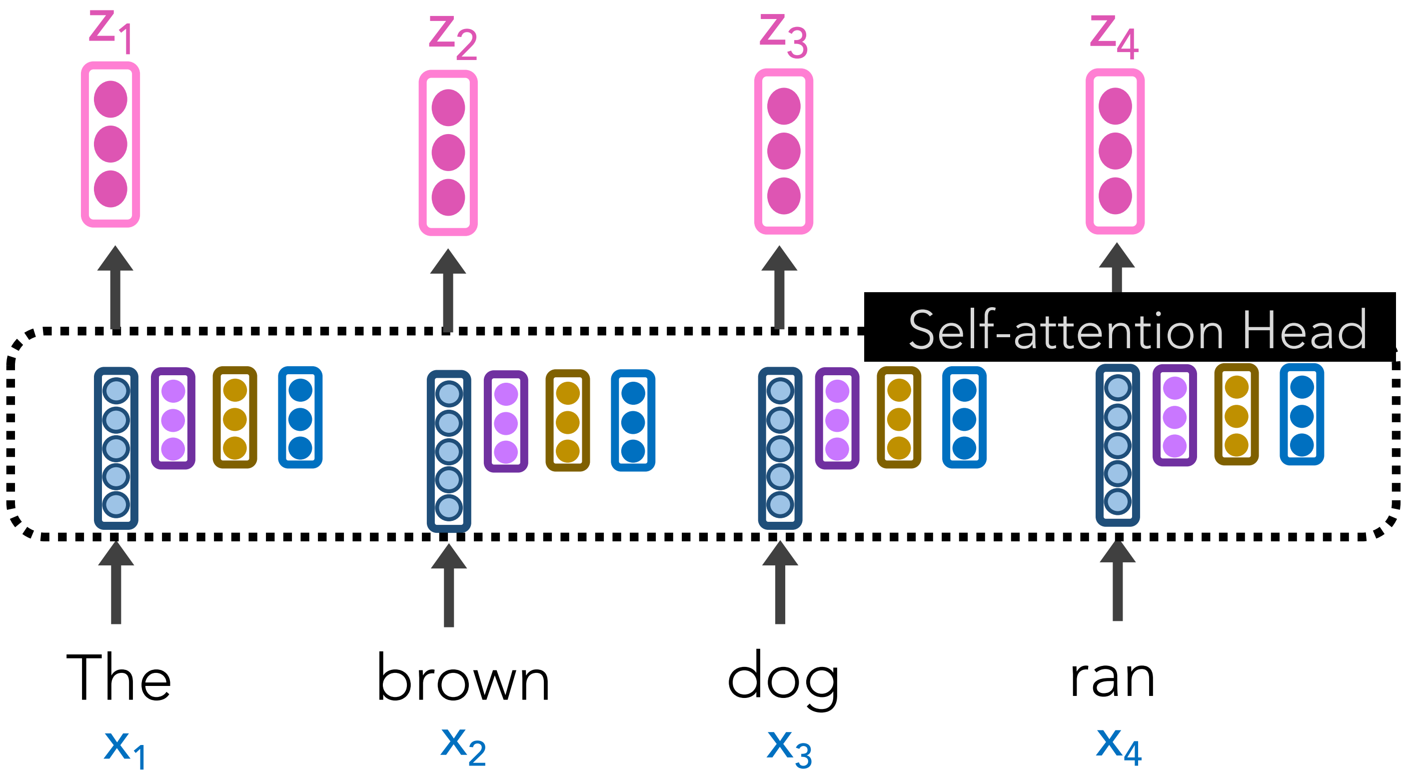
# Self-Attention




A:

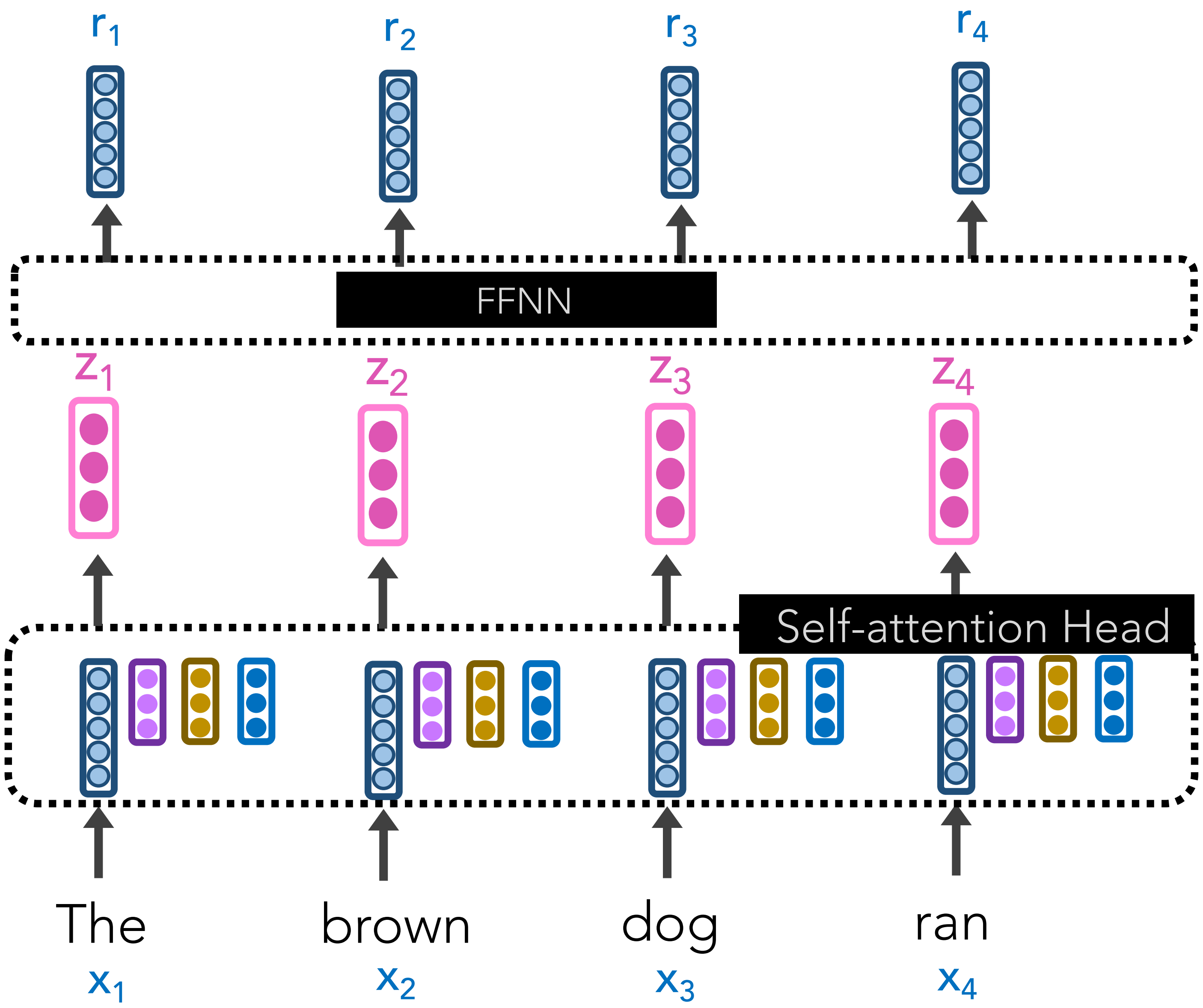
- There are only linear interactions; no non-linear relationship captured
- Position agnostic (BoW)

# Self-Attention



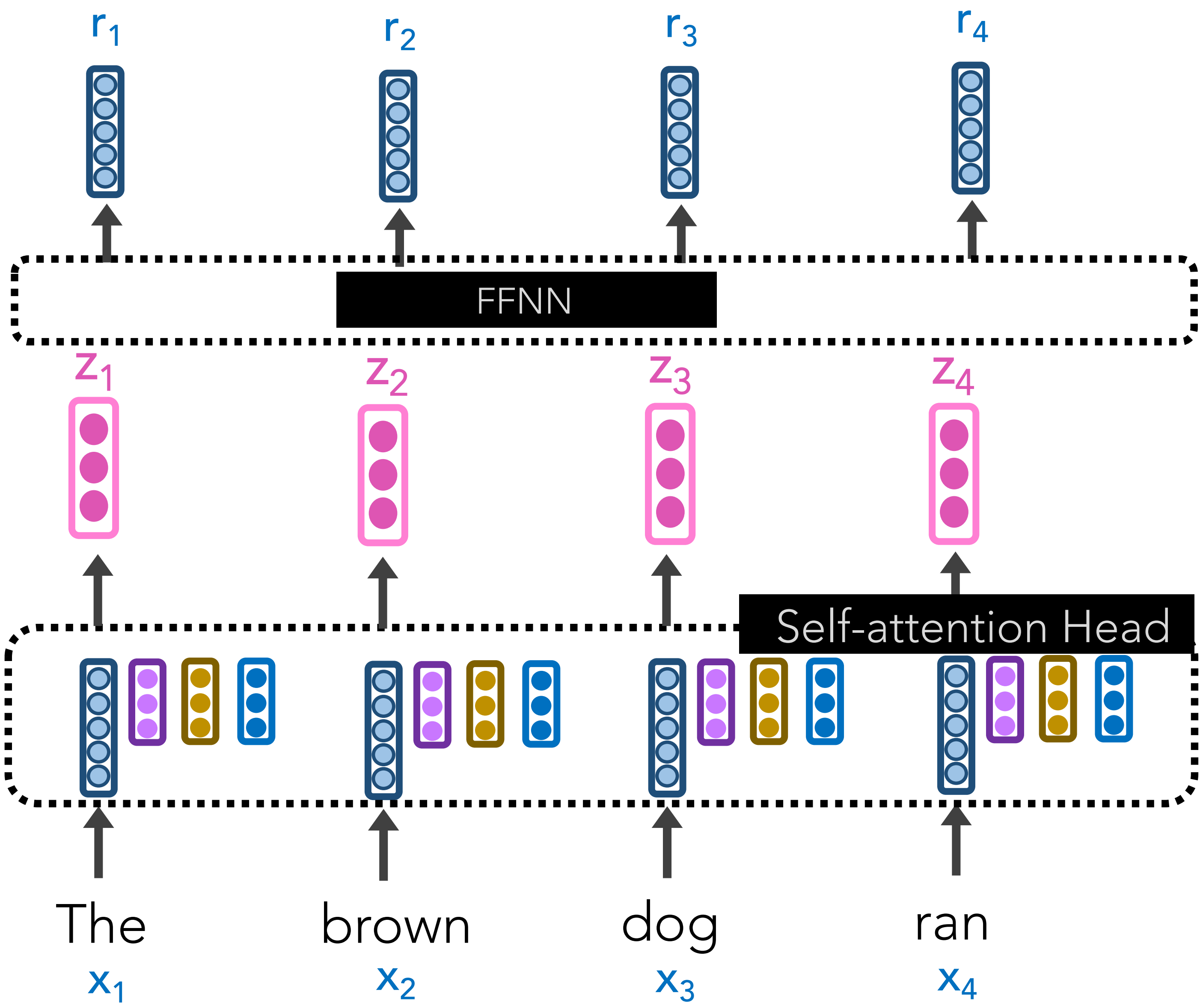

Let's further pass all the $z_i$'s through a FFNN

# Self-Attention + FFNN




Let's further pass all the $z_i$'s through a FFNN

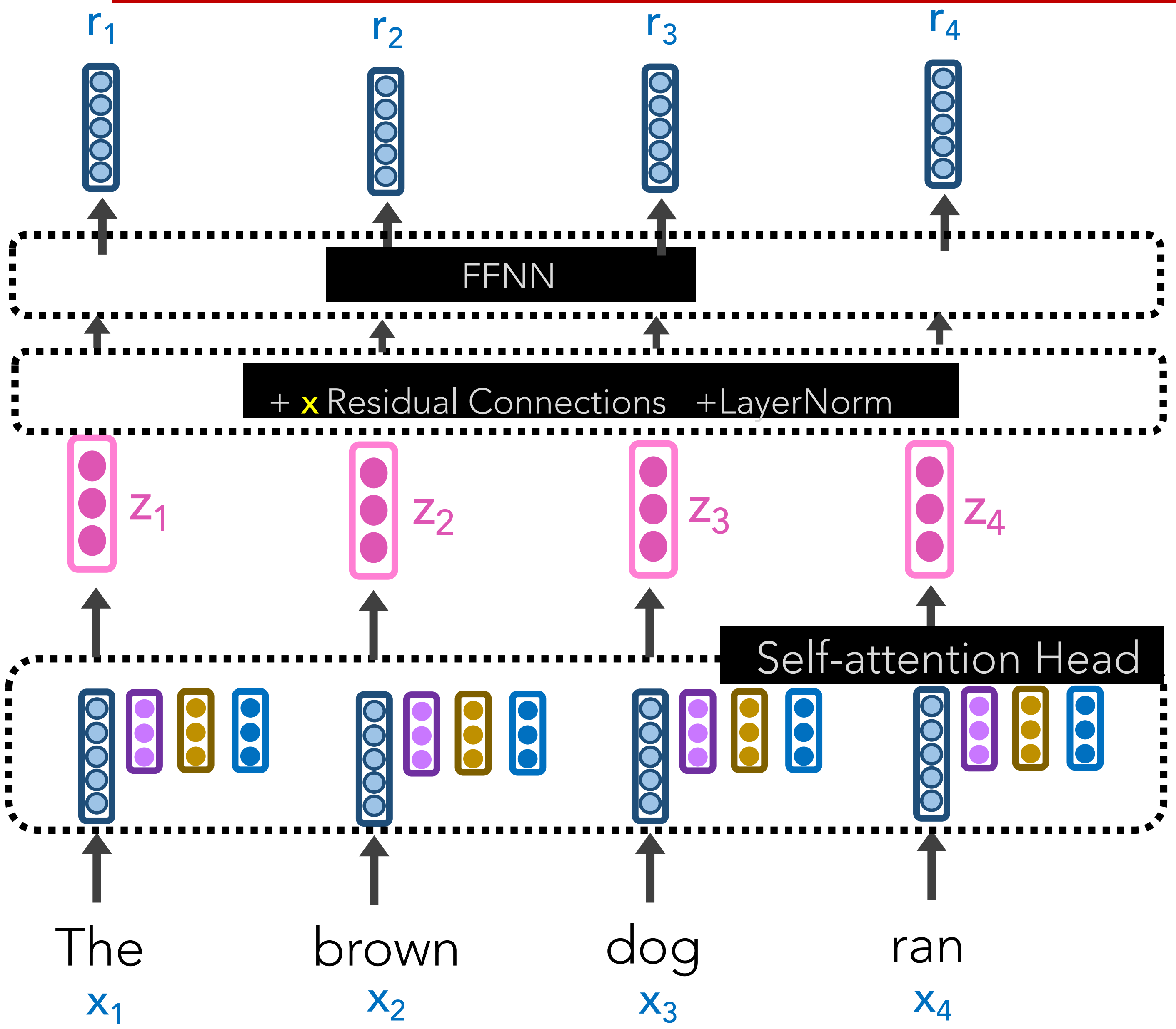
# Self-Attention + FFNN




Let's further pass all the $z_i$'s through a FFNN

But first, let's modifier our inputs into the FFNN to help ensure we don't lose precious info and that the values are reasonable (normalized)

# Self-Attention + FFNN



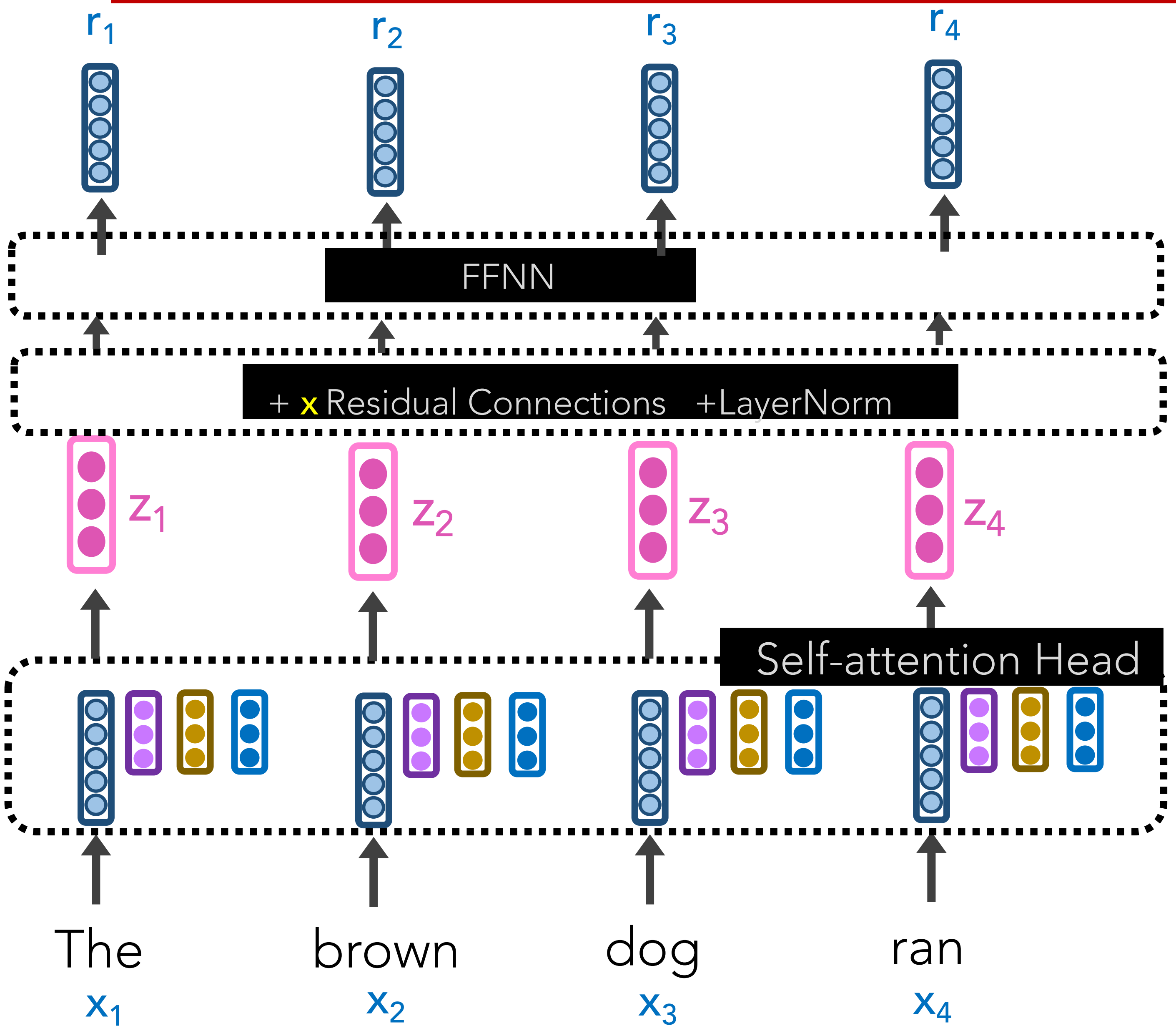

Let's further pass all the $z_i$'s through a FFNN

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

A **residual connection (aka skip connection)** allows the input $x$ to <u>skip</u> the computation at hand $f()$ and directly contribute to the output

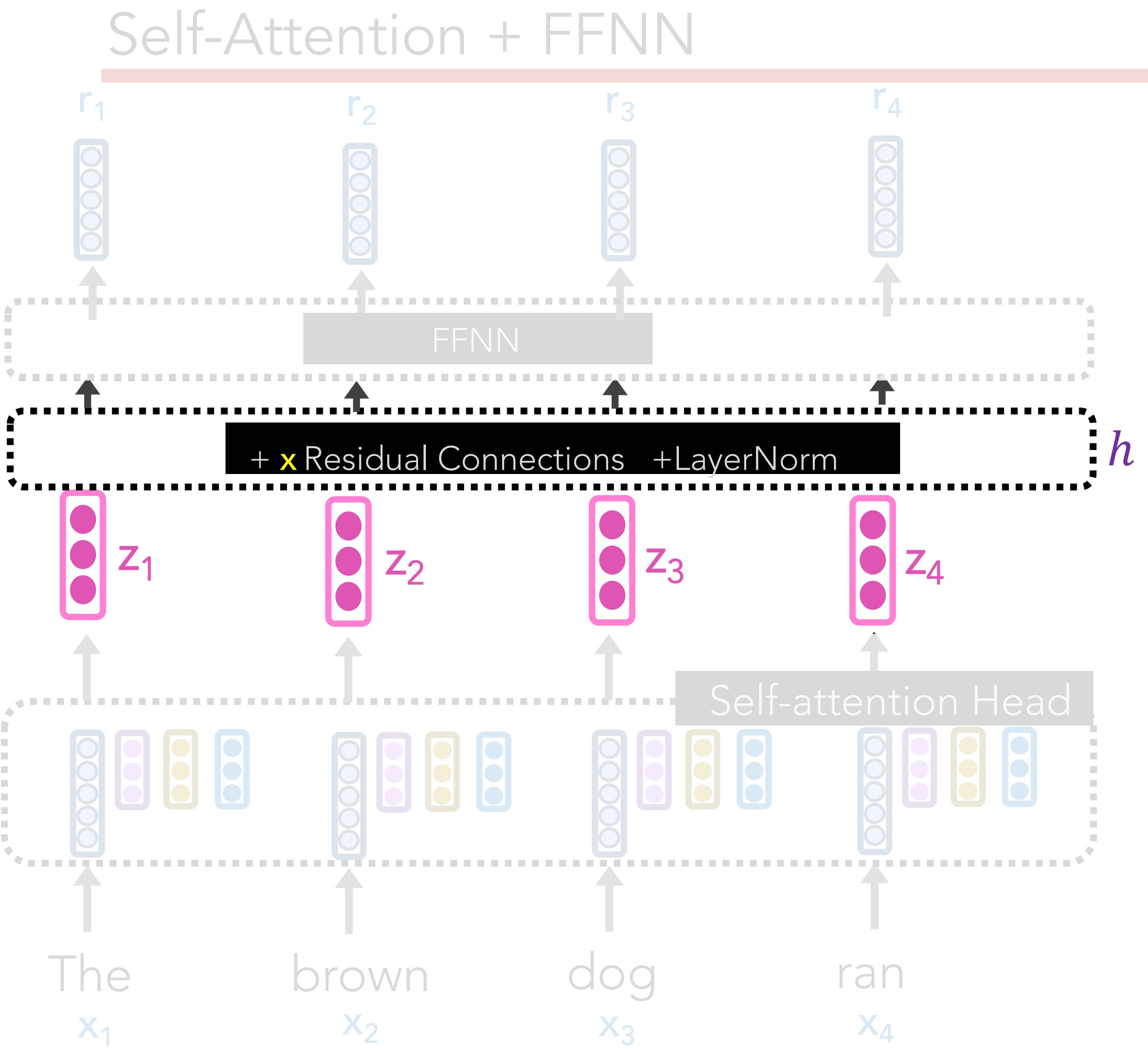
$$f_{residual}(x) = f(x) + x$$

# Self-Attention + FFNN




Let's further pass all the $z_i$'s through a FFNN

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

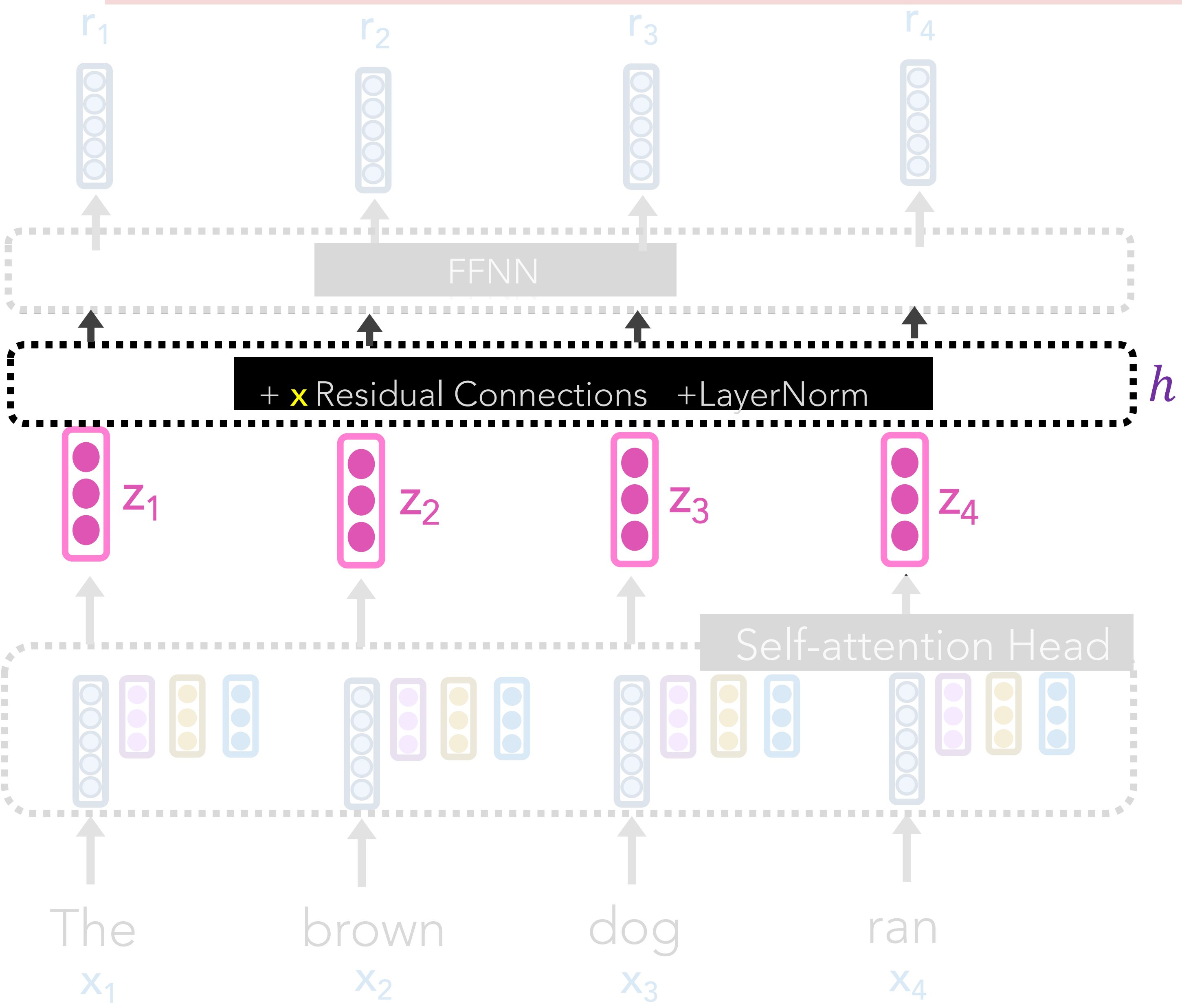



Let's further pass all the $z_i$'s through a FFNN

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

$$LayerNorm(h) = \frac{h - \hat{u}}{\sigma}$$

# Self-Attention + FFNN




Let's further pass all the $z_i$'s through a FFNN

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

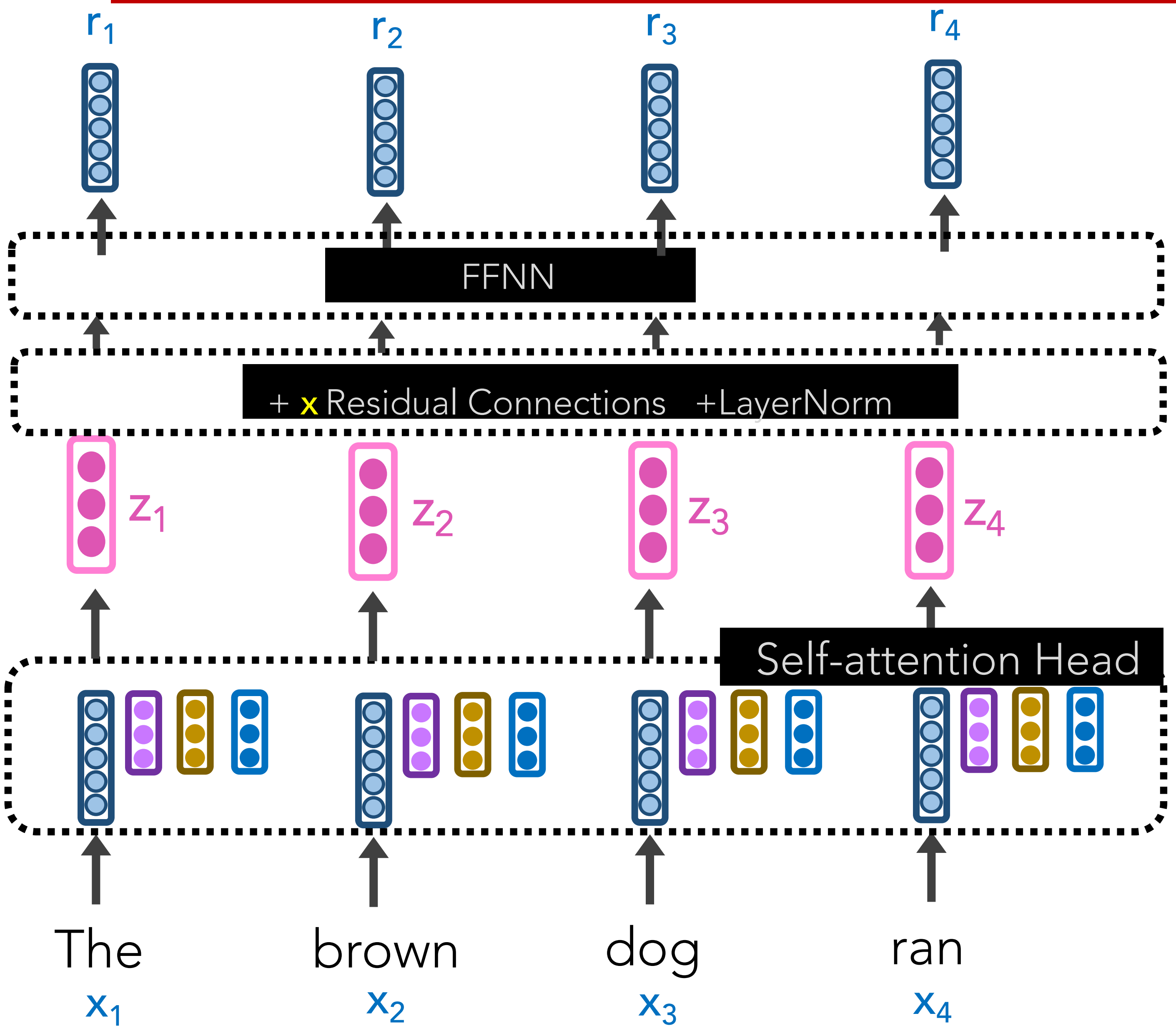
$$LayerNorm(h) = \frac{h - \hat{u}}{\sigma}$$

$$h_{pre-norm} = f(LayerNorm(h)) + h$$

$$h_{post-norm} = LayerNorm(f(h) + h)$$

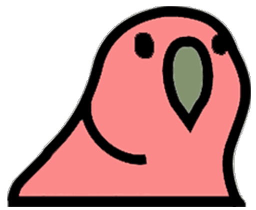
$h_{pre-norm}$ tends to work better and faster in practice. Xiong et al., 2020

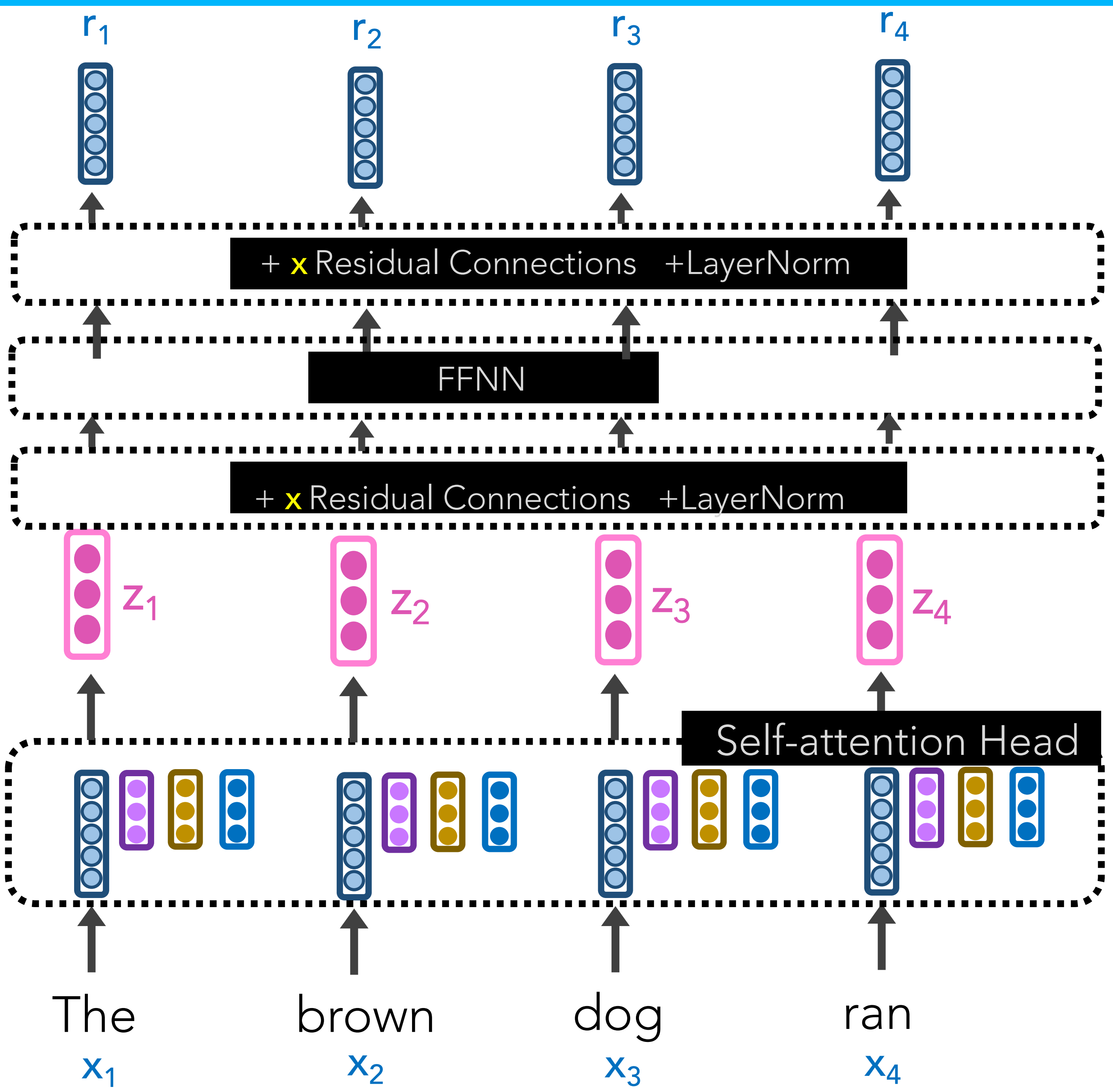
# Self-Attention + FFNN




Let's further pass all the $z_i$'s through a FFNN

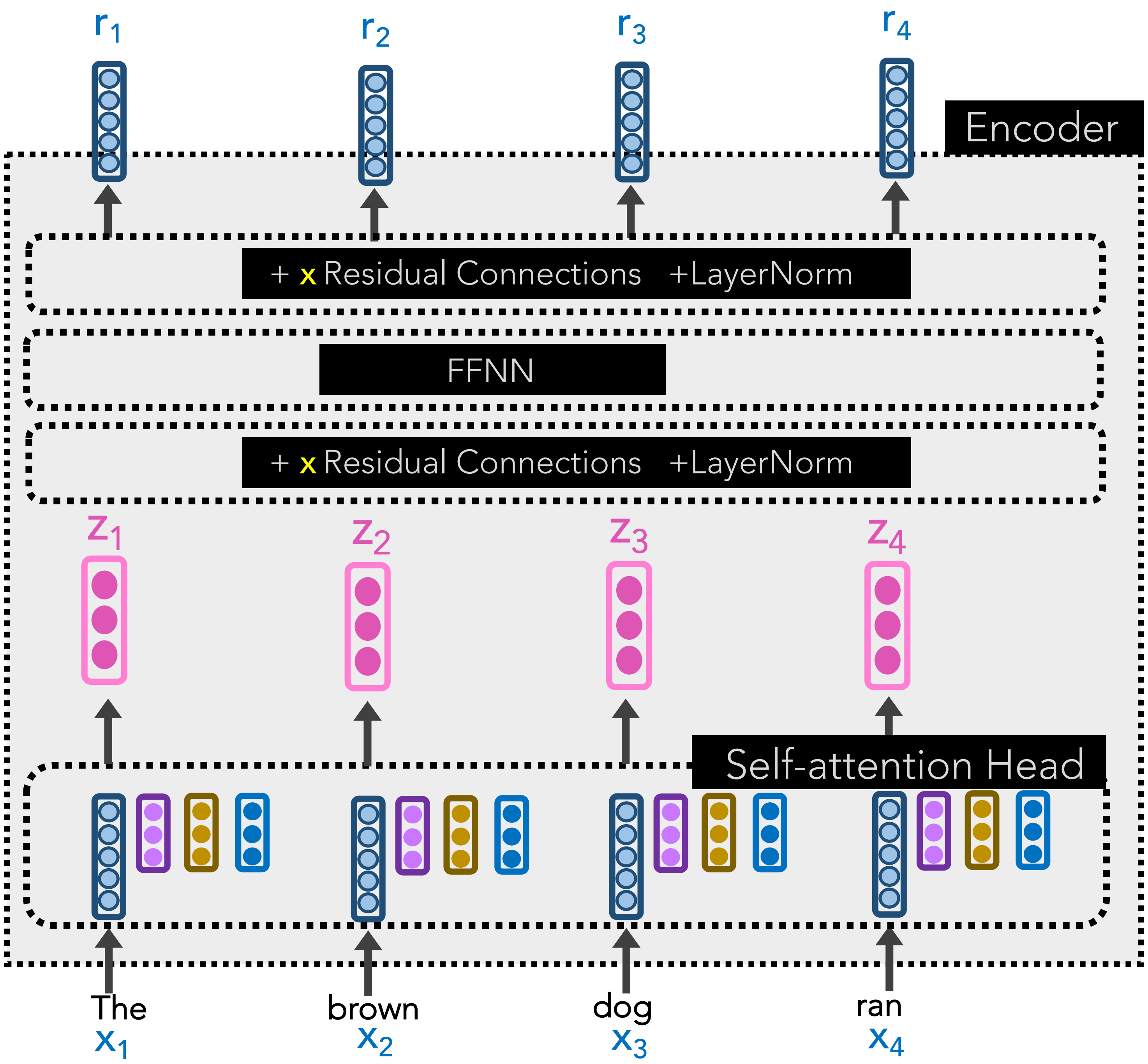
Each $z_i$ can be computed in parallel, unlike LSTMs!



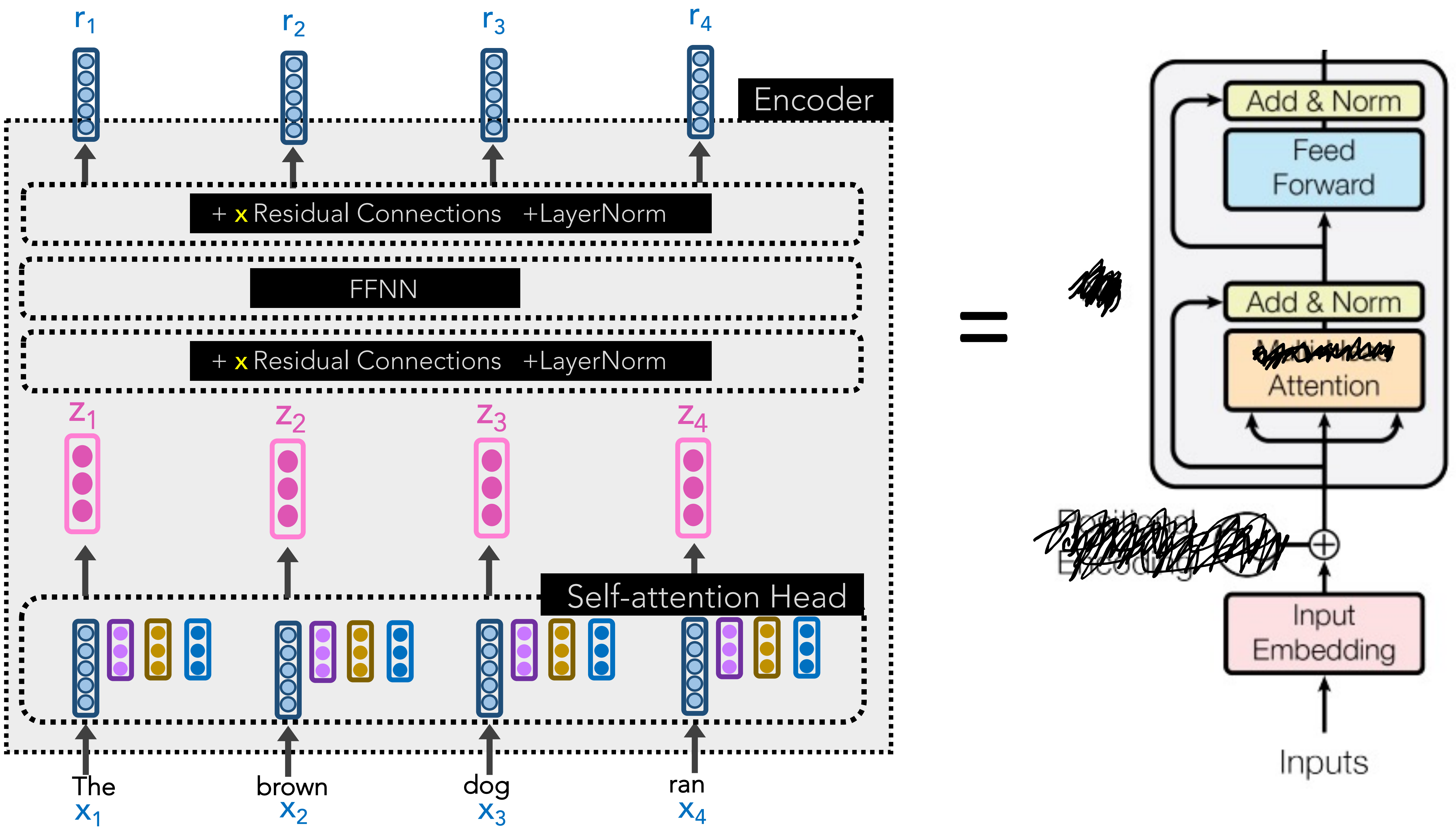



Let's further pass all the $z_i$'s through a FFNN

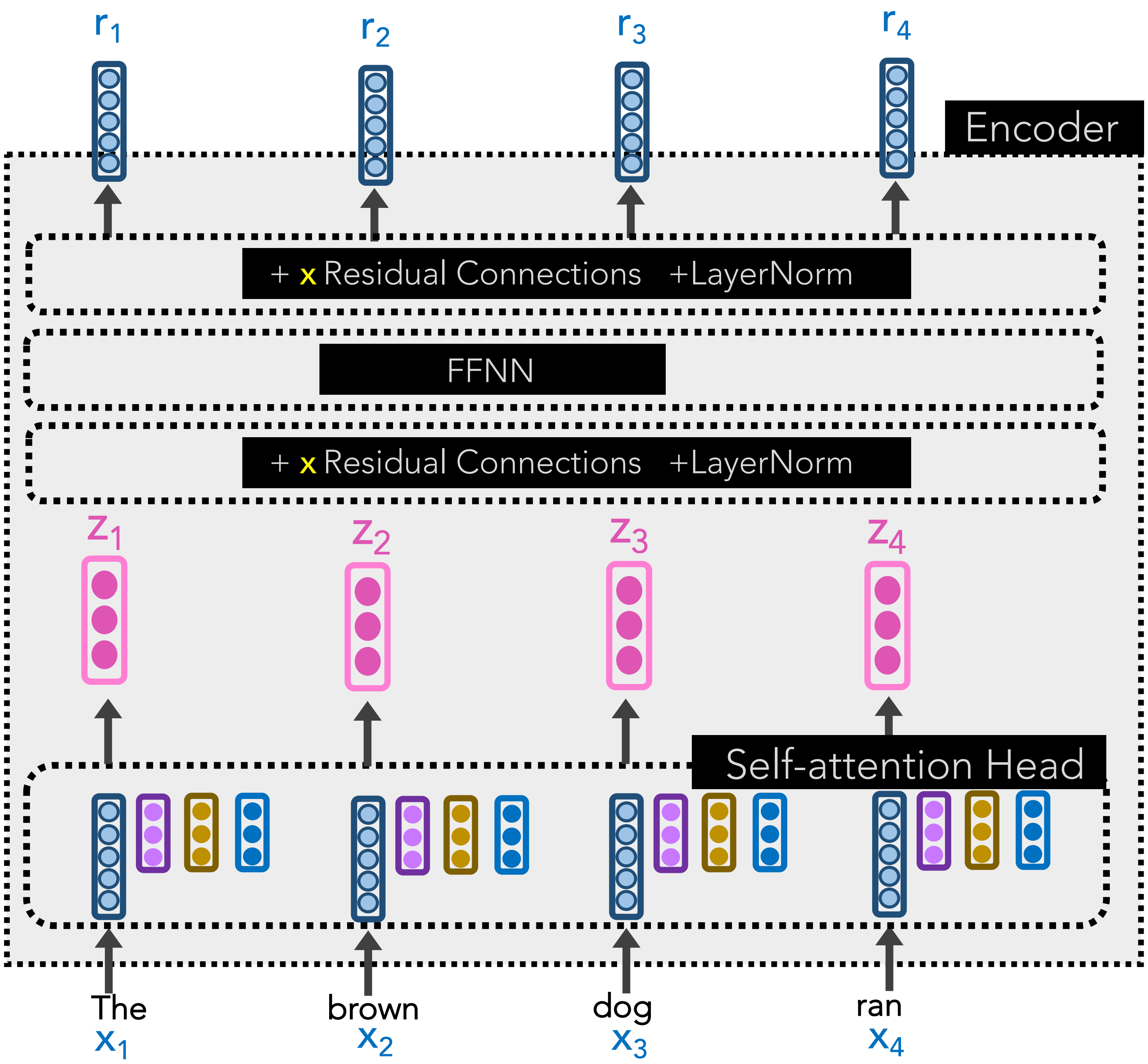
We additionally add a **Residual Connection and LayerNorm transformation** after the FFNN, too.




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

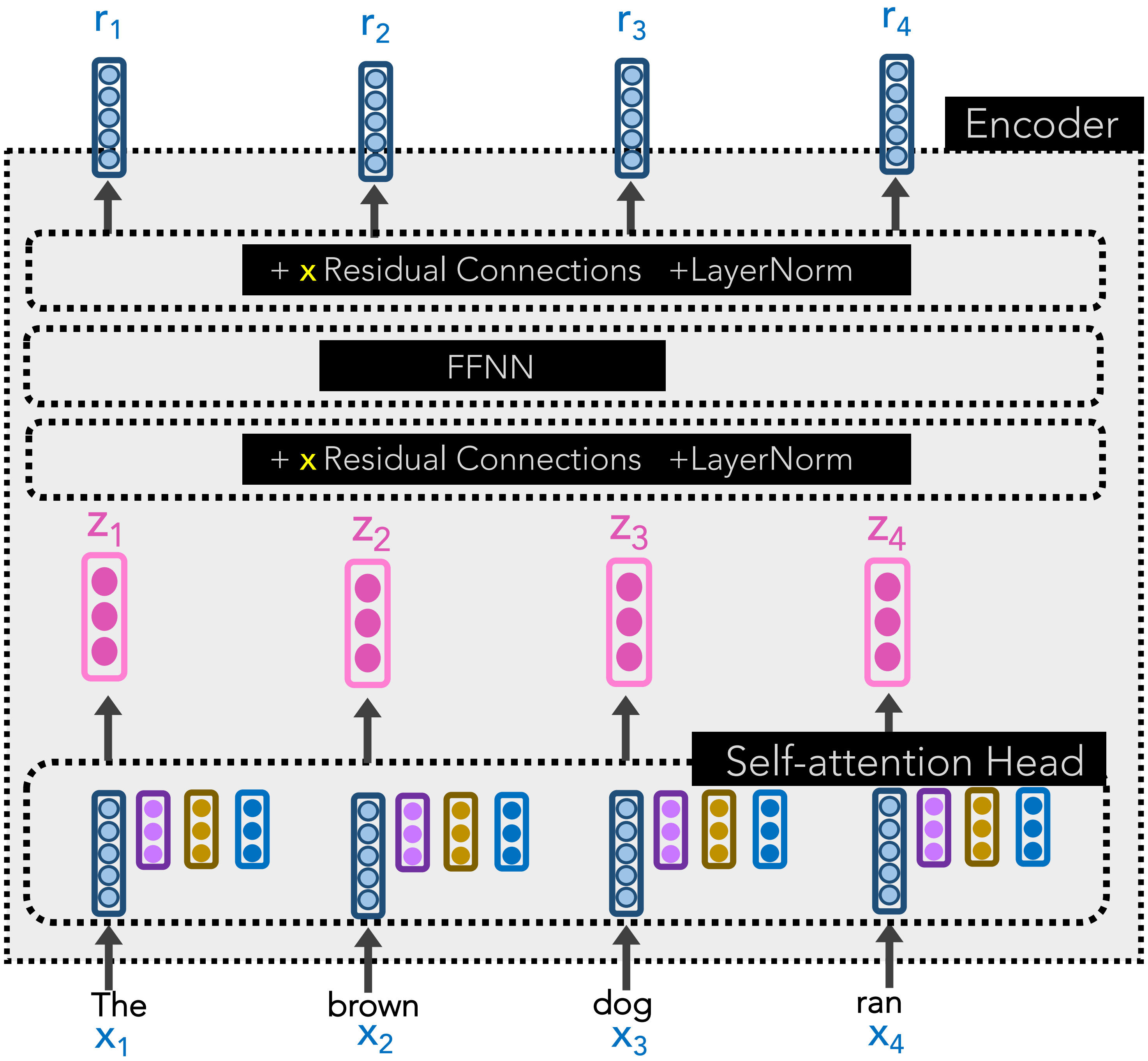


r1
r2
r3
r4
Encoder
+ x Residual Connections +LayerNorm
FFNN
+ x Residual Connections +LayerNorm
z1
z2
z3
z4
Self-attention Head
The
x1
brown
x2
dog
x3
ran
x4
=
Add & Norm
Feed Forward
Add & Norm
Attention
Input Embedding
Inputs




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"




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

**Problem:** there is no concept of <u>positionality</u>. Words are weighted as if a "bag of words"

**Solution:** add to each input word $x_i$ a positional encoding such as $\sim \sin(i) \cos(i)$

# Position Encodings

## Many ways to construct positional embeddings

Key characteristics we want:

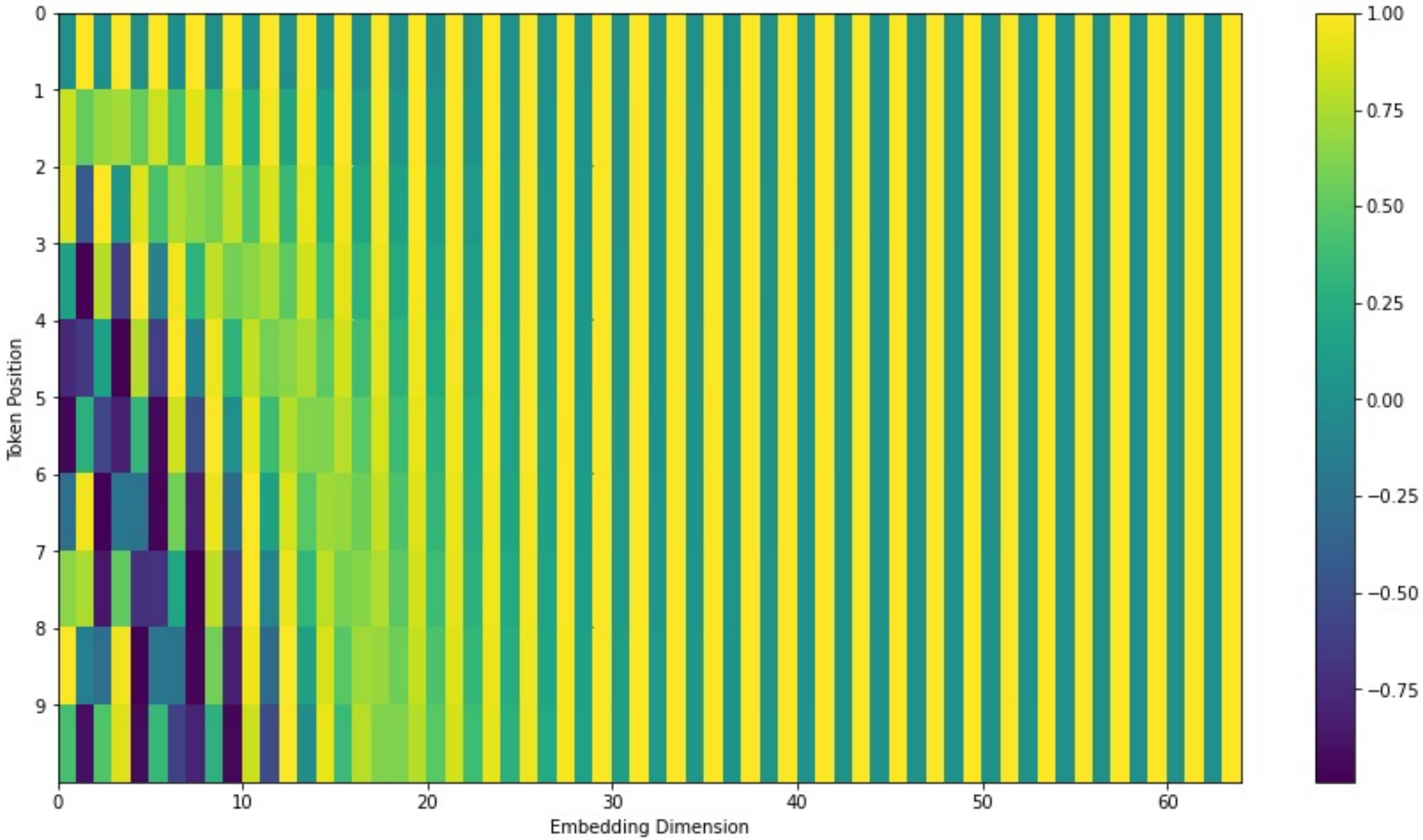
- Sequential positioning info (e.g., index 4 is after 3)
- Some aspects of both absolute and relative positioning
- Not susceptible to lengths we saw during training time

$$\check{x}_i = x_i + p_i$$

Usually we <u>add</u> positional embeddings $p_i$ to our inputs $x_i$,
but you could <u>concatenate</u> if you wish

# Position Encodings

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$




Can handle repeatability, but these embeddings are hardcoded – ideally would be learnable, too.

https://jalammar.github.io/illustrated-transformer/

[Transformer Language Models without Positional Encodings Still Learn Positional Information (Haviv et al., 2022)]

# Position Encodings

Learnable positional embeddings!

Learn $p \in \mathbb{R}^{dxn}$, where $n$ is the # of positions represented

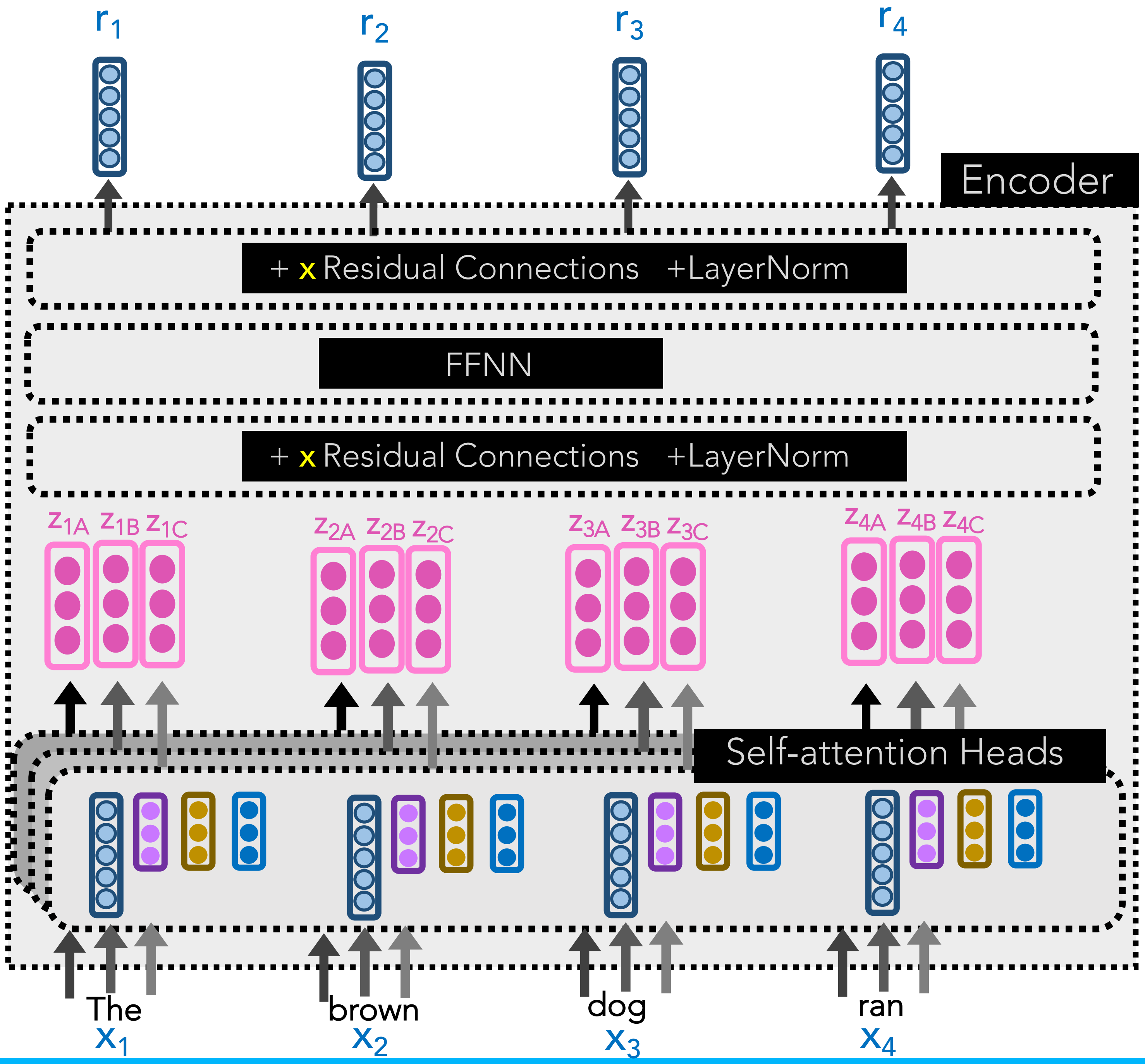
Each position gets to learn how to best fit/assist the data's representation, but we're limited to a fixed $n$ positions

$$\check{x}_i = x_i + p_i$$

A Self-Attention Head has just one set of query/key/value weight matrices $W_{q,}$ $W_{k,}$ $W_v$

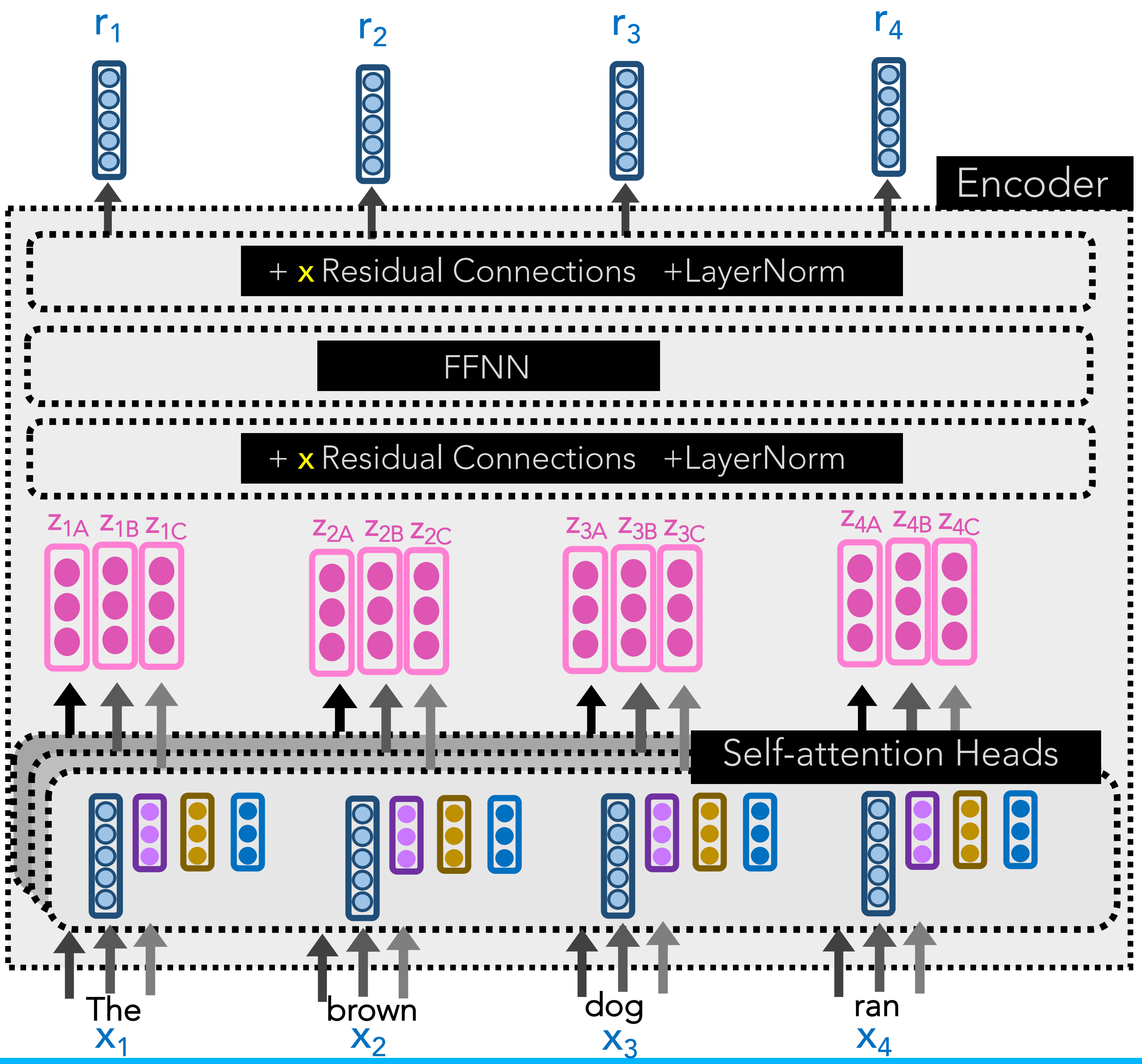
Words can relate in many ways, so it's restrictive to rely on just one Self-Attention Head in the system.

Let's create Multi-headed Self-Attention




Each Self-Attention Head produces a $z_i$ vector.

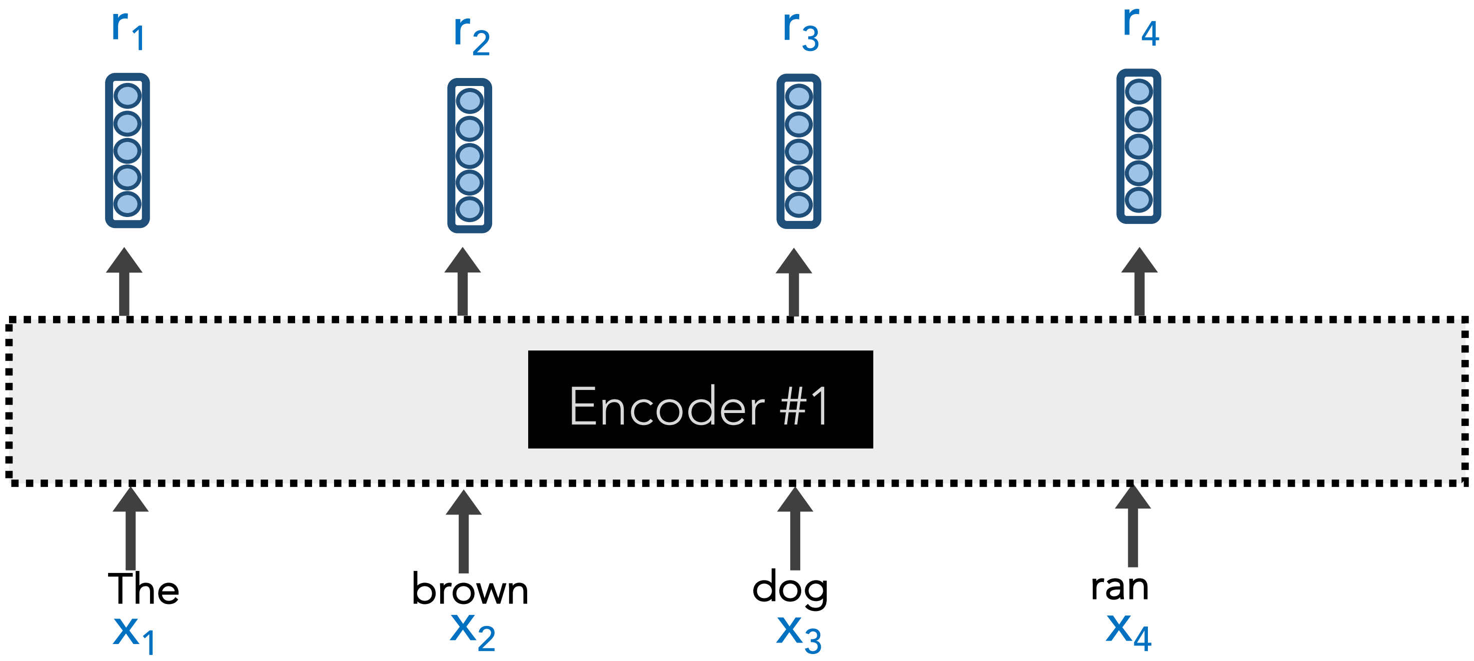
We can, in parallel, use multiple heads and concat the $z_i$'s.




**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

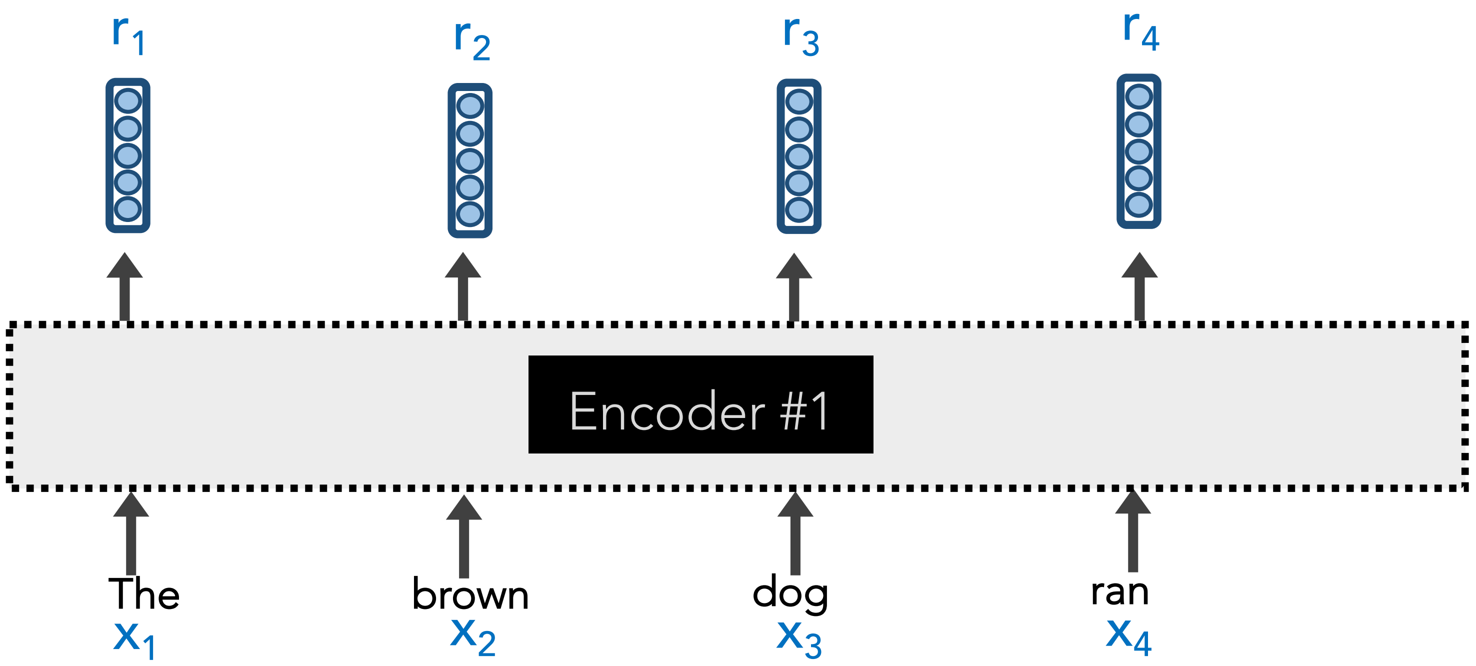
Within each Encoder, we have Positional Embeddings and Multi-Headed Attention




**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

Within each Encoder, we have Positional Embeddings and Multi-Headed Attention

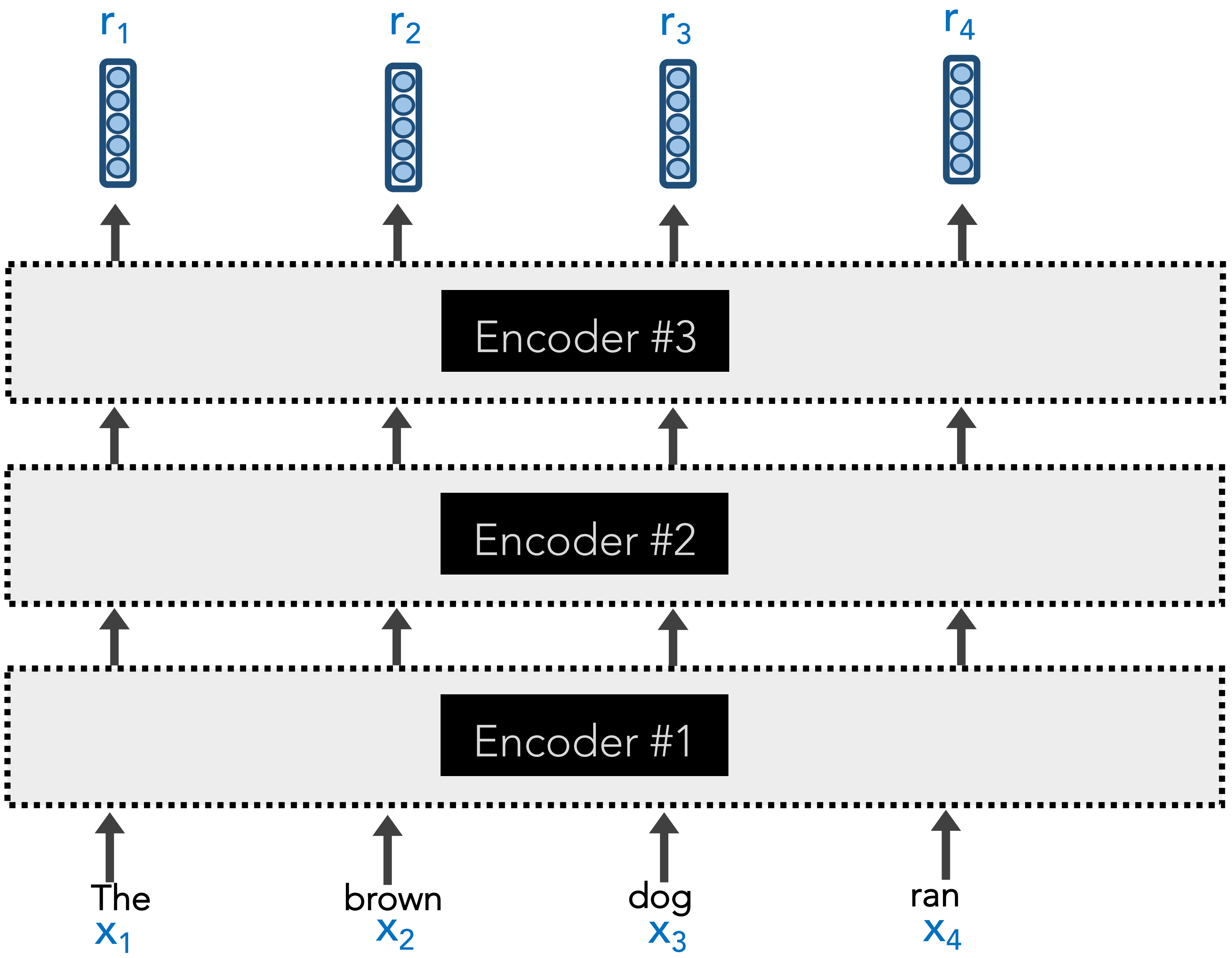



**To recap:** all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** $r_i$ of each word $x_i$

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

# Transformer Encoder

Within each Encoder, we have Positional Embeddings and Multi-Headed Attention

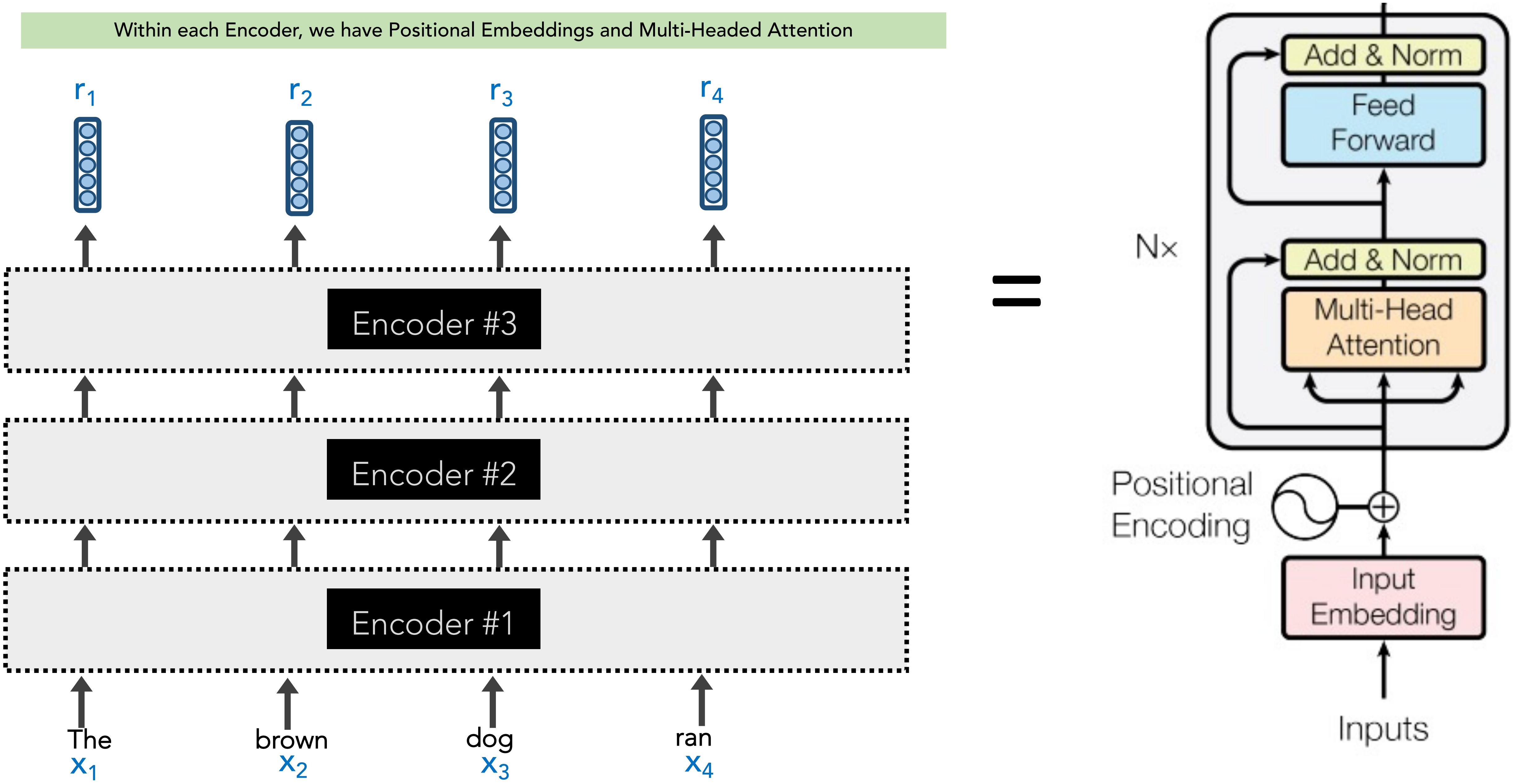



**To recap:** all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** $r_i$ of each word $x_i$

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

# Transformer Encoder

Within each Encoder, we have Positional Embeddings and Multi-Headed Attention



n = sequence length

d = length of representation (vector)

Q: Is the complexity of self-attention good?

| 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) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

https://arxiv.org/pdf/1706.03762.pdf

**Important:** when learning dependencies b/w words, you don't want long paths. Shorter is better.

Self-attention connects all positions with a constant # of sequentially executed operations, whereas RNNs require $O(n)$.

| 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) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

$n$ = sequence length ~20 - 70

$d$ = representation dimension. ~1024

$k$ = kernel size of convolutions

- 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)

# THOUGHT EXERCISE 1

How can we design a Transformer Decoder for seq2seq learning (i.e., **N-to-M** predictions)?

e.g., machine translation, such as converting English to French

**Training data:** hundreds of thousands of annotated sentence pairs (English and their French translations)

## HINT

*The assumptions of Self-Attention*

# THOUGHT EXERCISE 2

How can we chunk (aka tokenize) our input words such that they are comprised of discrete, meaningful sub-word units?

i.e., akin to syllables

**Training data:** millions of unannotated, natural sentences found on the Internet

# HINT

*Remember, NLP has accelerated since the 90s, when the field shifted toward statistical approaches*

# Summary

- Transformers allow for more complete, free access to everything (unless masked) at once

- It's very useful to pre-train a large unsupervised/self-supervised LM then fine-tune on your particular task (replace the top layer, so that it can work)