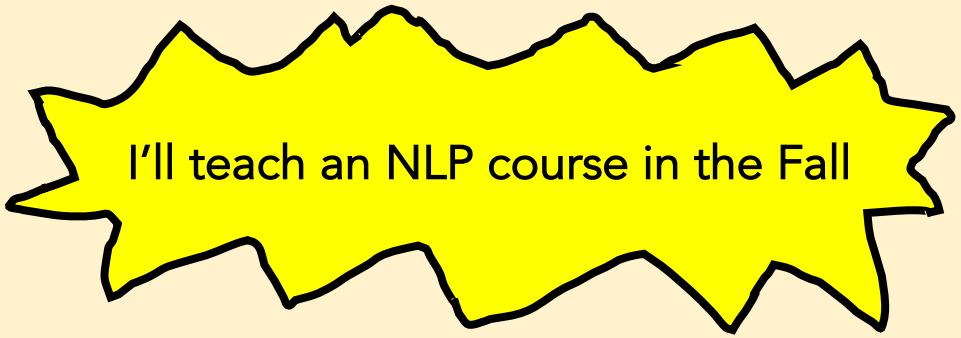
Advances in NLP

Language Modelling, Self-Attention, Transformers

Chris Tanner

HARVARD IACS
COMPUTEFEST
JAN 19–22 2021

FOREWORD



We could easily spend an entire semester on this material.

The goal for today is to convey:

- the ubiquity and importance of text data/NLP
- high-level overview of the most useful, relevant models
- when to use which models, based on your data
- foundation for diving deeper (e.g., our lab notebooks!)

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

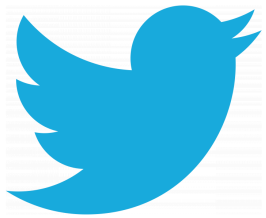
Transformers +BERT and GPT

Conclusions

Our digital world is inundated with text.

How can we leverage it for useful tasks?

62B pages



500M tweets/day

360M user pages

The New York Times

13M articles

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

- Constituency
- Dependency

Discourse

Summarization

Coreference Resolution

Semantics

Sentiment Analysis

Topic Modelling

Named Entity Recognition (NER)

Relation Extraction

Word Sense Disambiguation

Natural Language Understanding (NLU)

Natural Language Generation (NLG)

Machine Translation

Entailment

Question Answering

Language Modelling

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

- Constituency
- Dependency

Semantics

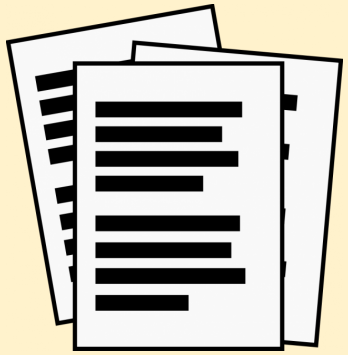
Sentiment Analysis

Topic Modelling

Discourse

Summarization

Coreference Resolution



"Overall, Pfizer's COVID-19 vaccine is very safe and one of the most effective vaccines ever produced"

Question Answering

Language Modelling

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

Constituenc

Dependenc

Discourse

Summarization

Coreference Re

Semantics

Sentiment Analysis

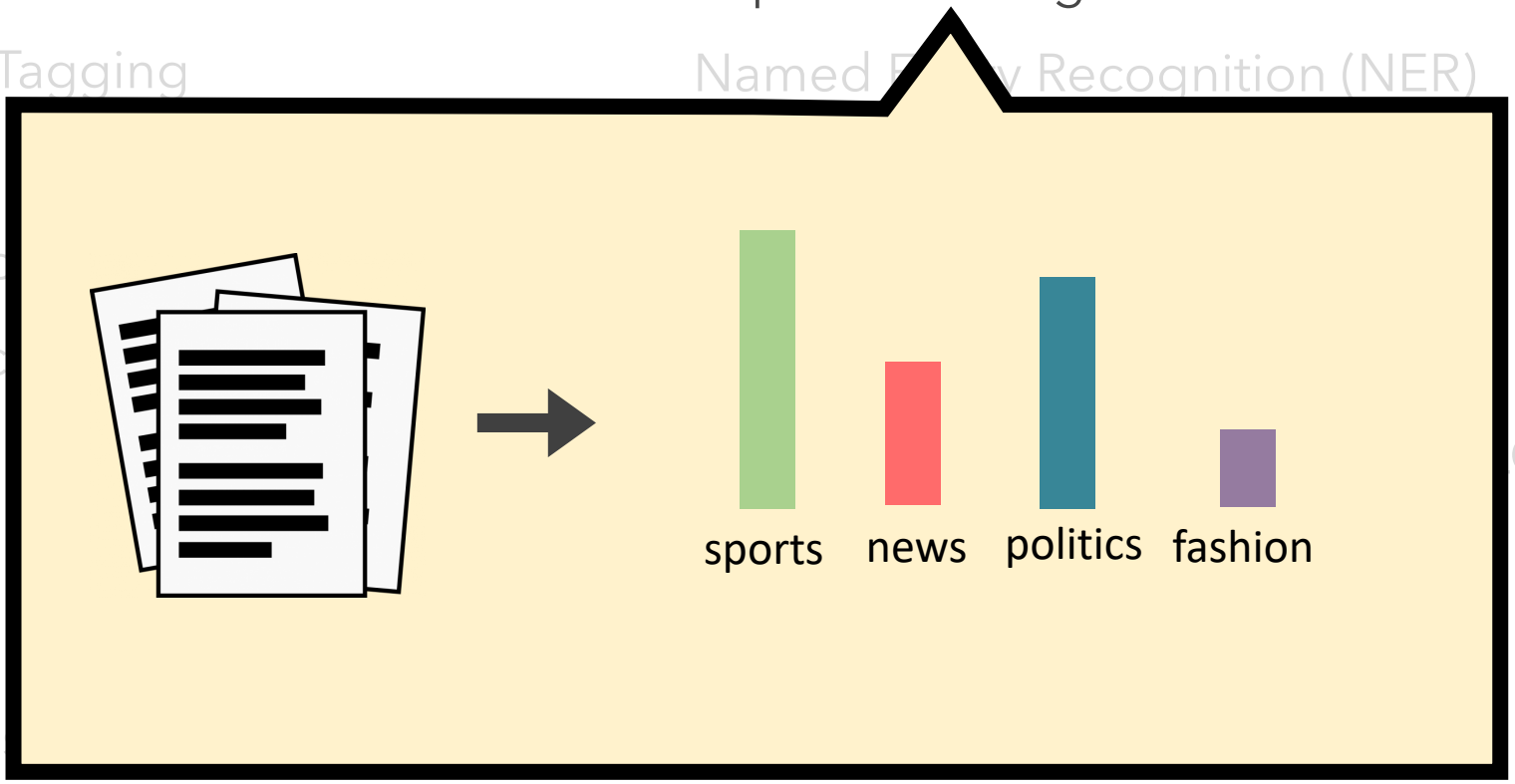
Topic Modelling

Named E y Recognition (NER)

(NLU)

G)

Language Modelling



Common NLP Tasks (aka problems)

Syntax

Morpholog

Word Segm

Part-of-Spe

Parsing

Constit

Dependency

Discourse

Summarization

Coreference Resolution

Semantics

IER)

"Alexa, play Drivers License by Olivia Rodrigo"

"Alexa, play Drivers License by Olivia Rodrigo"

INTENT SONG ARTIST

Natural Language Understanding (NLU)

Natural Language Generation (NLG)

Machine Translation

Entailment

Question Answering

Language Modelling

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

- Constituency
- Dependency

Semantics

Sentiment Analysis

Topic Modelling

Named Entity Recognition (NER)

ng (NLU)

Natural Language Generation (NLG)

Machine Translation

Entailment

Question Answering

Language Modelling

Discourse

Summarization

Coreference Resolution

El perro marrón → The brown dog

SPANISH ENGLISH

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

- Constituency
- Dependency

Discourse

Summarization

Coreference Resolution

Can help with every other task!

Semantics

Sentiment Analysis

Topic Modelling

Named Entity Recognition (NER)

Relation Extraction

Word Sense Disambiguation

Natural Language Understanding (NLU)

Natural Language Generation (NLG)

Machine Translation

Entailment

Question Answering

Language Modelling

Common NLP Tasks (aka problems)

Syntax

Morphology

Word Segmentation

Part-of-Speech Tagging

Parsing

- Constituency
- Dependency

Discourse

Summarization

Coreference Resolution

Semantics

Sentiment Analysis

Topic Modelling

Named Entity Recognition (NER)

Relation Extraction

Word Sense Disambiguation

Natural Language Understanding (NLU)

Natural Language Generation (NLG)

Machine Translation

Entailment

Question Answering

Language Modelling

Can help with every other task!

Language Modelling

A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)

Language Modelling

A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)

Spam

Not Spam

Language Modelling

A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)

Language Modelling

FORMAL DEFINITION

A Language Model estimates the probability of any sequence of words

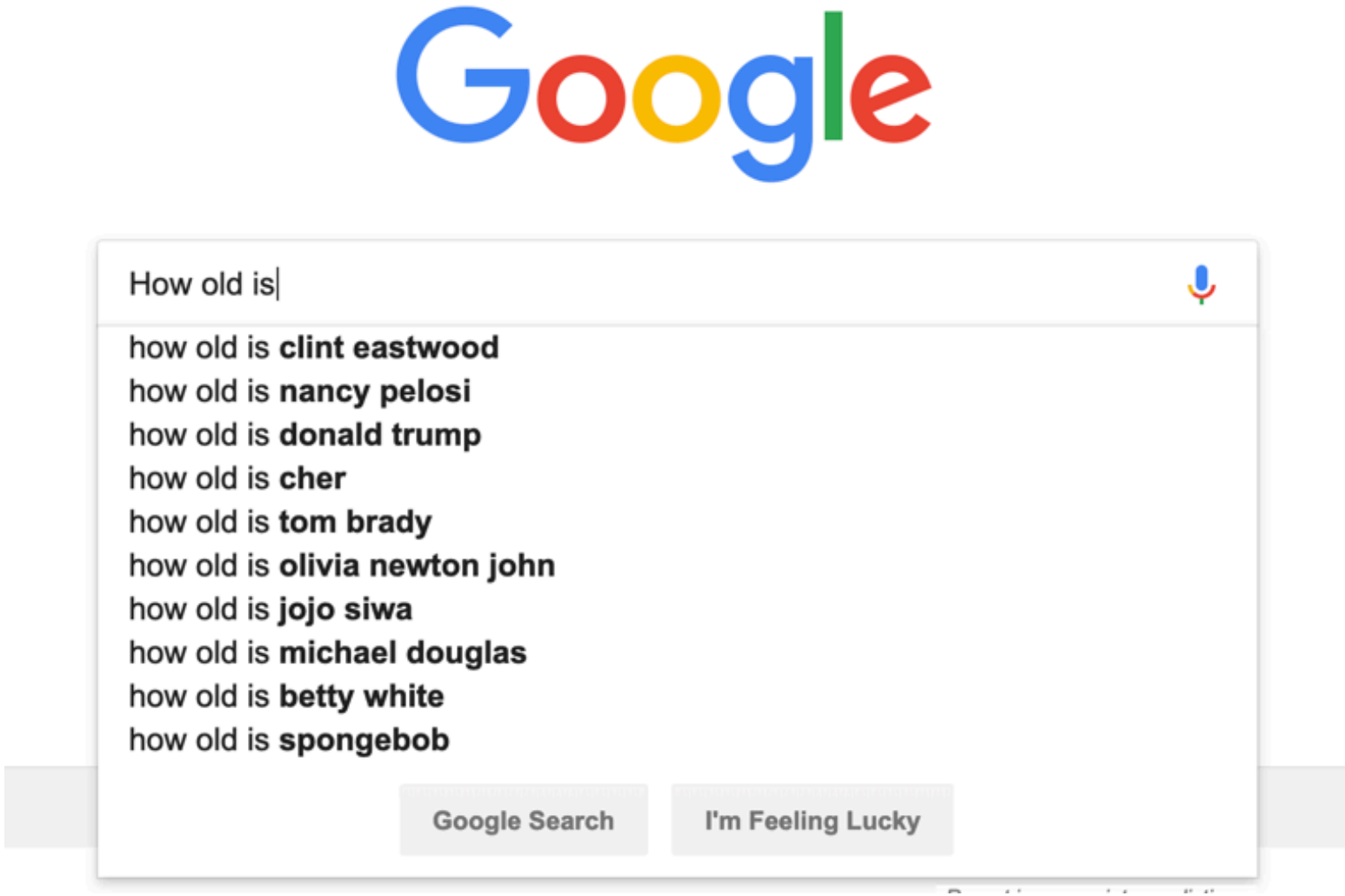
Let $\boldsymbol{X}$ = "Anqi was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{"Anqi was late for class"})$$

Language Modelling

Generate Text

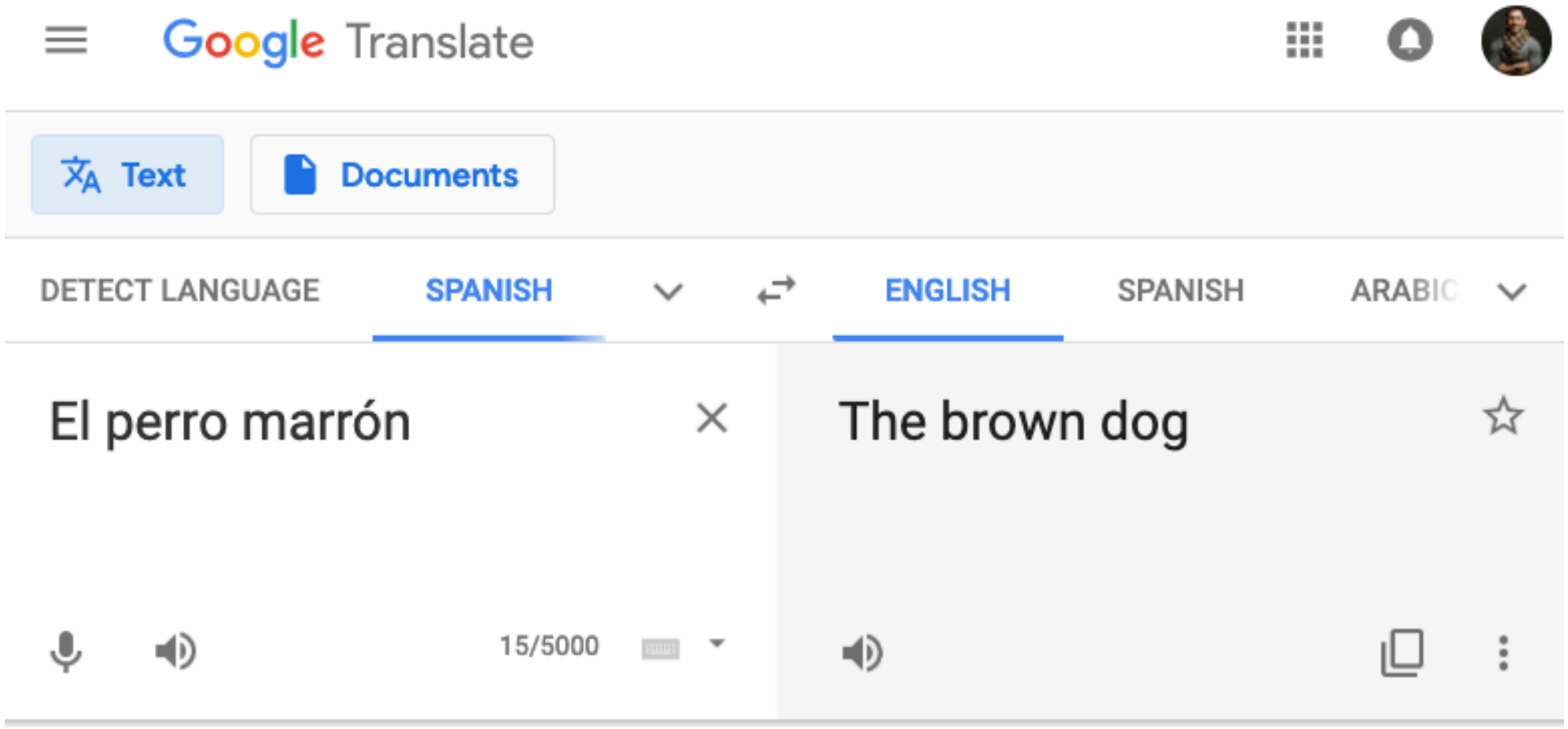


Language Modelling

Generate Text

Language Modelling

Generate Text



Language Modelling

"Drug kingpin El Chapo testified that he gave MILLIONS to Pelosi, Schiff & Killary. The Feds then closed the courtroom doors."

Fake News

Real News

Language Modelling

A Language Model is useful for:

Generating Text

- Auto-complete
- Speech-to-text
- Question-answering / chatbots
- Machine translation

Classifying Text

- Authorship attribution
- Detecting spam vs not spam

And much more!

Language Modelling

How can we build a language model?

Language Modelling

Naive Approach: unigram model

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t)$$

Assumes each word is independent of all others.

Language Modelling

Naive Approach: unigram model

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t)$$

Assumes each word is independent of all others.

Let $\boldsymbol{X}$ = "Anqi was late for class"

x_1 x_2 x_3 x_4 x_5

Language Modelling

Naive Approach: unigram model

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t)$$

Assumes each word is independent of all others.

Let $\boldsymbol{X}$ = "Anqi was late for class"

x_1 x_2 x_3 x_4 x_5

= 0.00015 * 0.01 * 0.004 * 0.03 * 0.0035
= 6.3×10^{-13}

You calculate each of these probabilities from the training corpus

Language Modelling

UNIGRAM ISSUES?

Language Modelling

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{“Anqi was late for class”}) = P(\text{“class for was late Anqi”})$$

Language Modelling

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{“Anqi was late for class”}) = P(\text{“class for was late Anqi”})$$

Sequence generation: What's the most likely next word?

Anqi was late for class _____

Language Modelling

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{“Anqi was late for class”}) = P(\text{“class for was late Anqi”})$$

Sequence generation: What's the most likely next word?

Anqi was late for class _____

Anqi was late for class the

Language Modelling

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{“Anqi was late for class”}) = P(\text{“class for was late Anqi”})$$

Sequence generation: What's the most likely next word?

Anqi was late for class _____

Anqi was late for class the

Anqi was late for class the the

Let’s do better

Language Modelling

Better Approach: n-gram model

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1)$$

Let's factor in context (in practice, a window of size **n**)

Language Modelling

Better Approach: n-gram model

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1)$$

The likelihood of any event occurring hinges upon all prior events occurring

Language Modelling

Better Approach: n-gram model

This compounds for all subsequent events, too

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1)$$

The likelihood of any event occurring hinges upon all prior events occurring

Language Modelling

bi-gram example

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Anqi was late for class"

w_1 w_2 w_3 w_4 w_5

Language Modelling

bi-gram example

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Anqi was late for class"

probability

x_1 x_2 x_3 x_4 x_5

$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Anqi})$

Language Modelling

bi-gram example

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Anqi was late for class"

probability

x_1 x_2 x_3 x_4 x_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Anqi})P(\text{late}|\text{was})$$

Language Modelling

bi-gram example

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Anqi was late for class"

x_1 x_2 x_3 x_4 x_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Anqi})P(\text{late}|\text{was})P(\text{for}|\text{late})$$

Language Modelling

bi-gram example

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Anqi was late for class"

probability

x_1 x_2 x_3 x_4 x_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Anqi})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling

bi-gram exam

You calculate each of these probabilities by simply counting the occurrences

$$P(class \mid for) = \frac{count(\textbf{for class})}{count(\textbf{for})}$$

probability

Let $\boldsymbol{X}$ = "Anqi was late for class"

x_1 x_2 x_3 x_4 x_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Anqi})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling

BIGRAM ISSUES

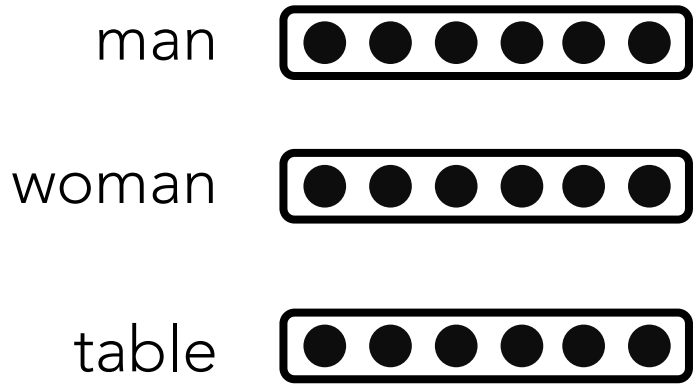
- **Out-of-vocabulary** items are 0 → kills the overall probability
- Always need more context (e.g., trigram, 4-gram), but **sparsity** is an issue (rarely seen subsequences)
- **Storage** becomes a problem as we increase window size
- No semantic information conveyed by counts (e.g., vehicle vs car)

Let’s do better

Language Modelling

IDEA: Let's use neural networks!

First, each word is represented by a word embedding
(e.g., vector of length 200)

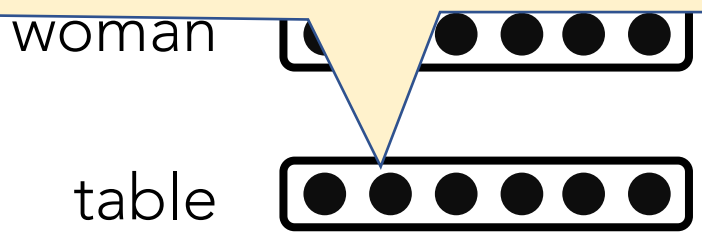


Language Modelling

IDEA: L

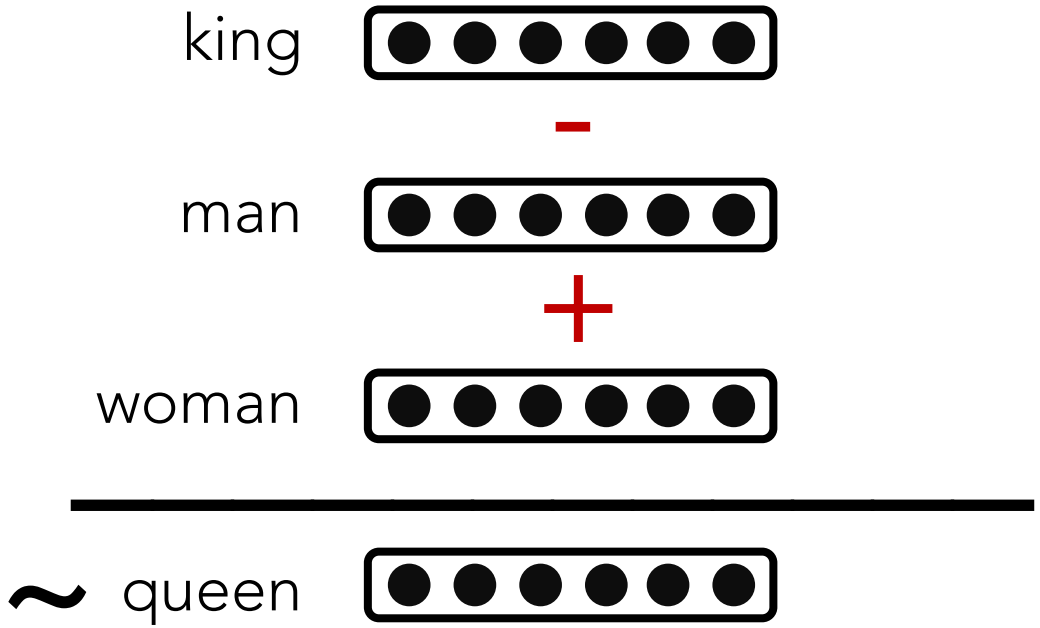
First, ea
(e.g., ve

- Each circle is a specific floating point scalar
- Words that are more semantically similar to one another will have embeddings that are proportionally similar, too
- We can use pre-existing word embeddings that have been trained on gigantic corpora



Language Modelling

These word embeddings are so rich that you get nice properties:

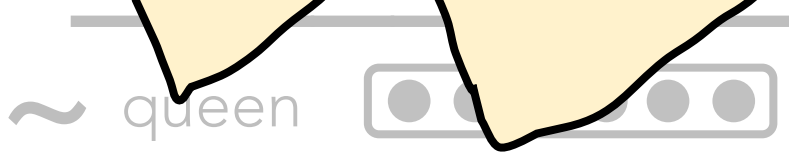


Word2vec: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
GloVe: https://www.aclweb.org/anthology/D14-1162.pdf

Language Modelling

These word embeddings are so rich that you get nice properties:

Takeaway #1:
word embeddings are very useful



Word2vec: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
GloVe: https://www.aclweb.org/anthology/D14-1162.pdf

Language Modelling

How can we use these word embeddings to build a new LM?

Remember, we only need a system that can estimate:

$$P(x_{t+1}|x_t, x_{t-1}, \ldots, x_1)$$

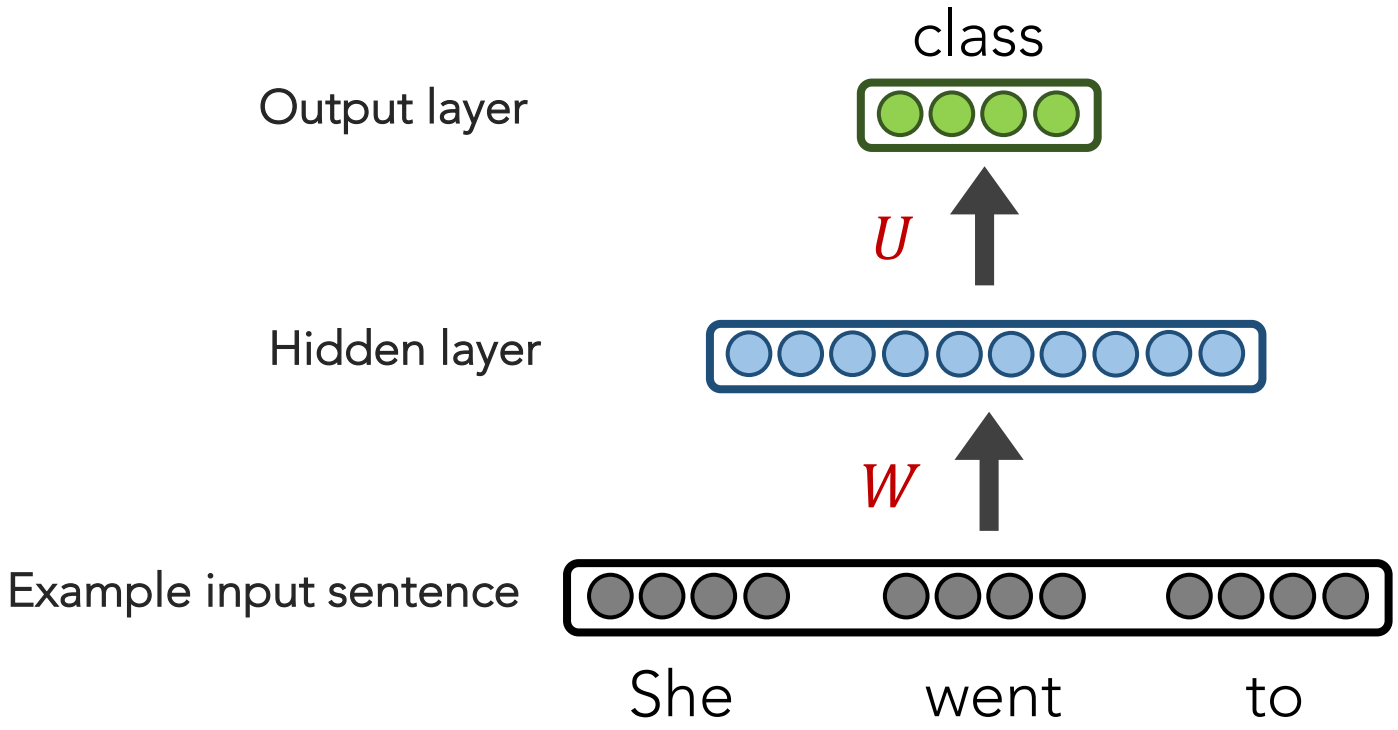
next word previous words

Example input sentence

She went to

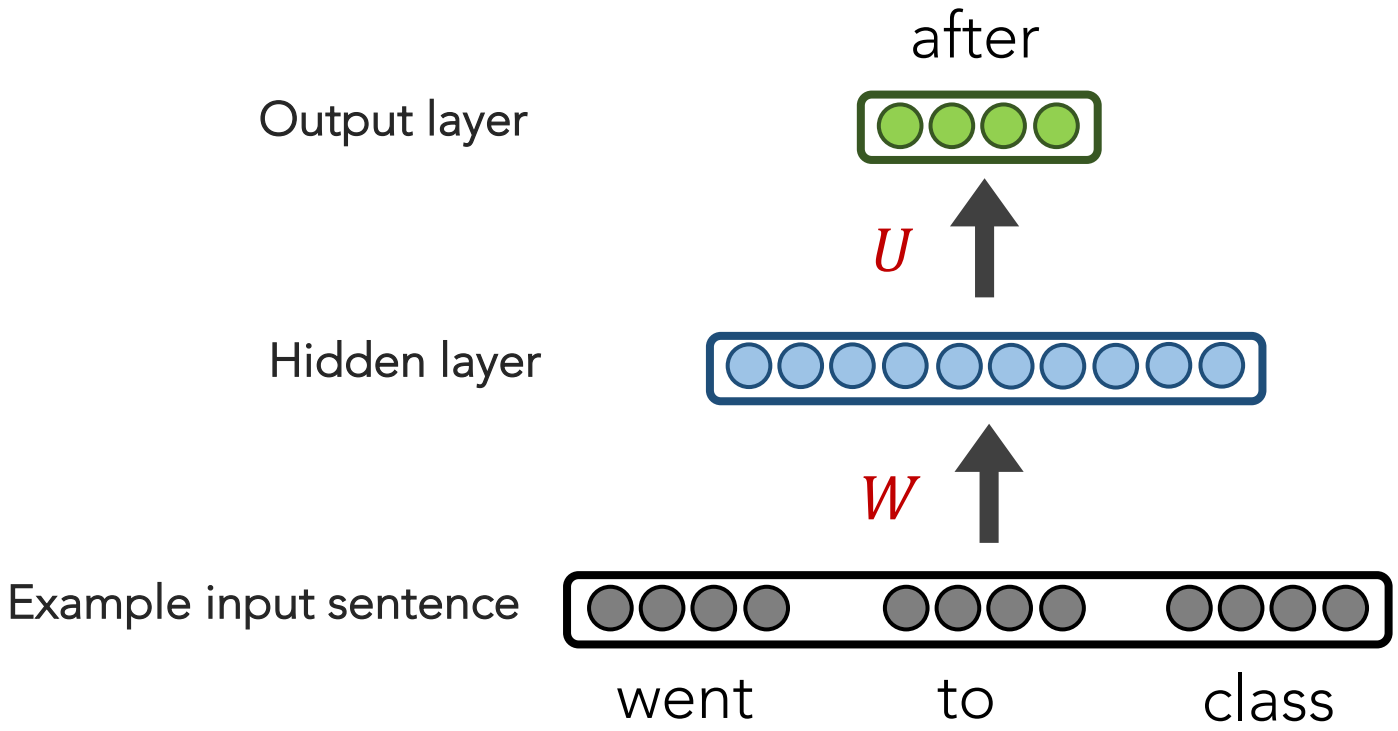
Language Modelling

Neural Approach: Feed-forward Neural Net aren't suitable



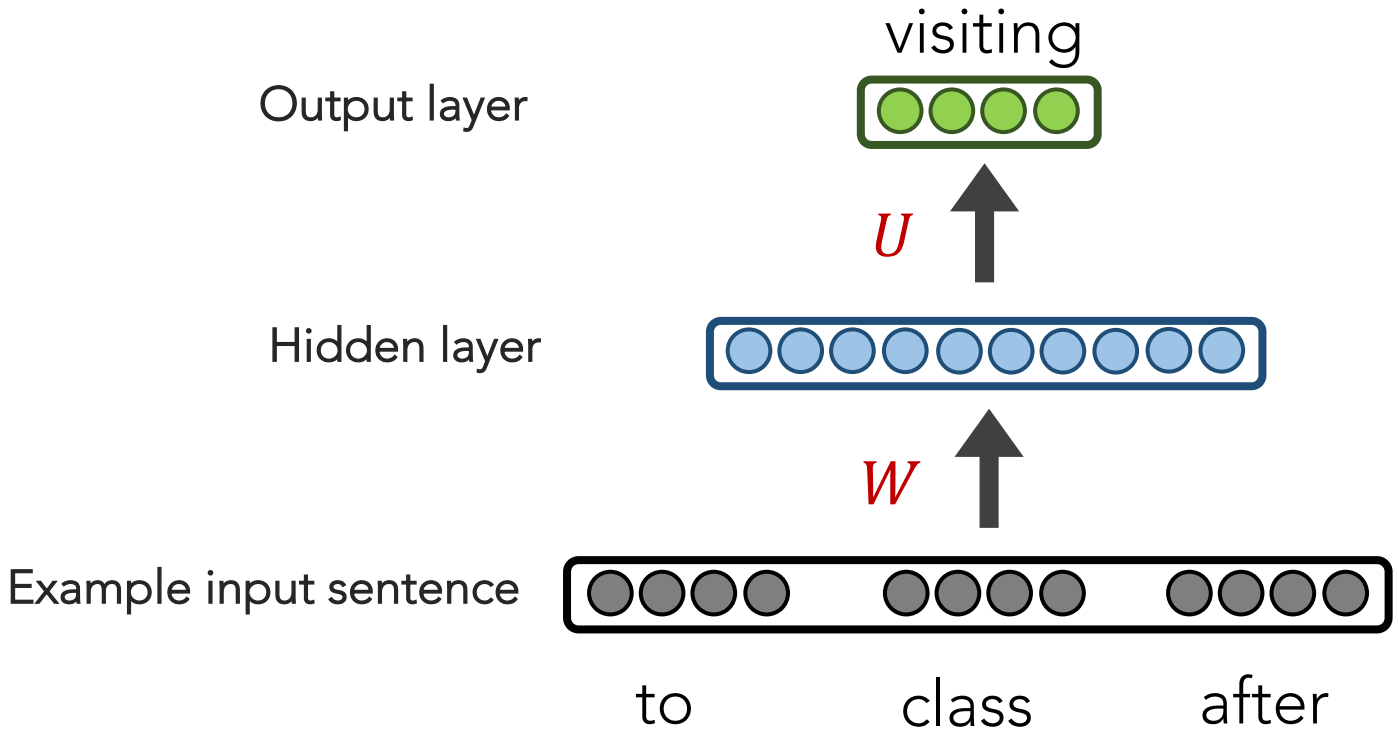
Language Modelling

Neural Approach: Feed-forward Neural Net aren't suitable



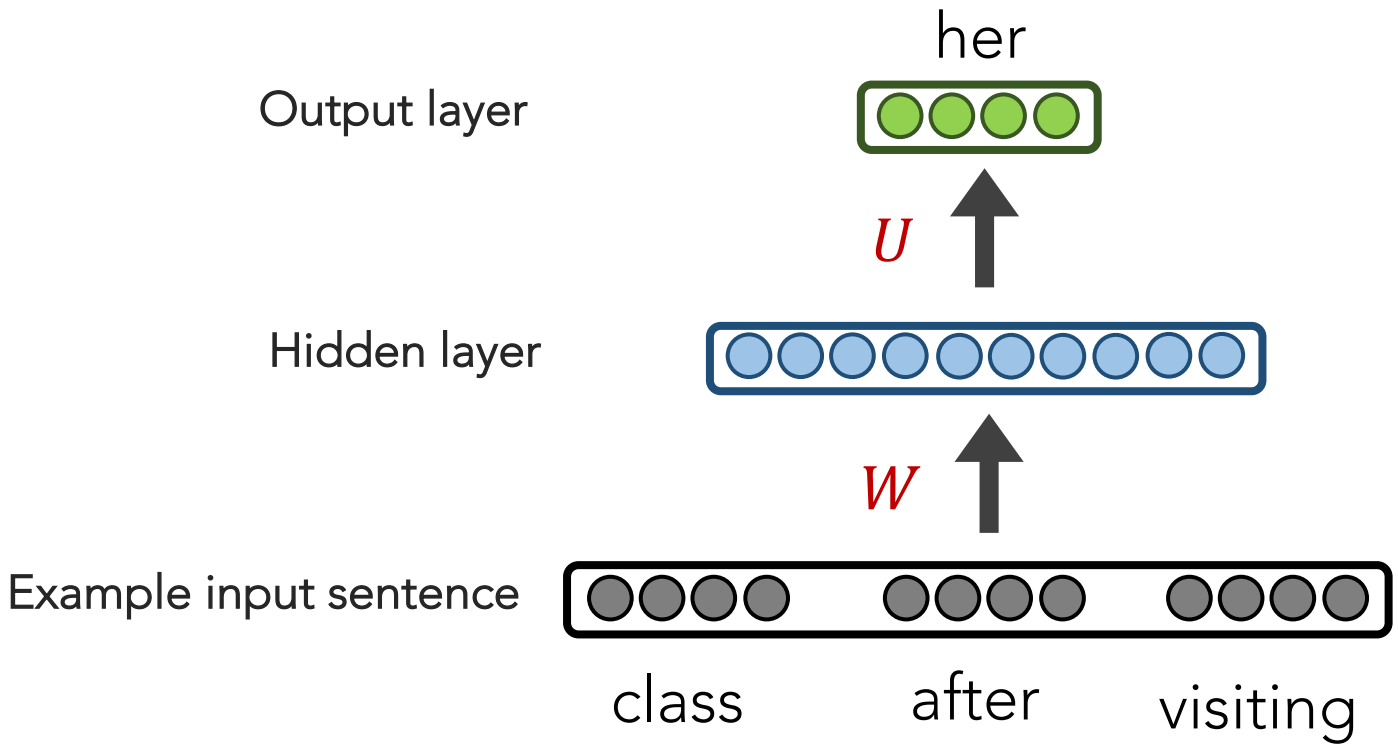
Language Modelling

Neural Approach: Feed-forward Neural Net aren't suitable



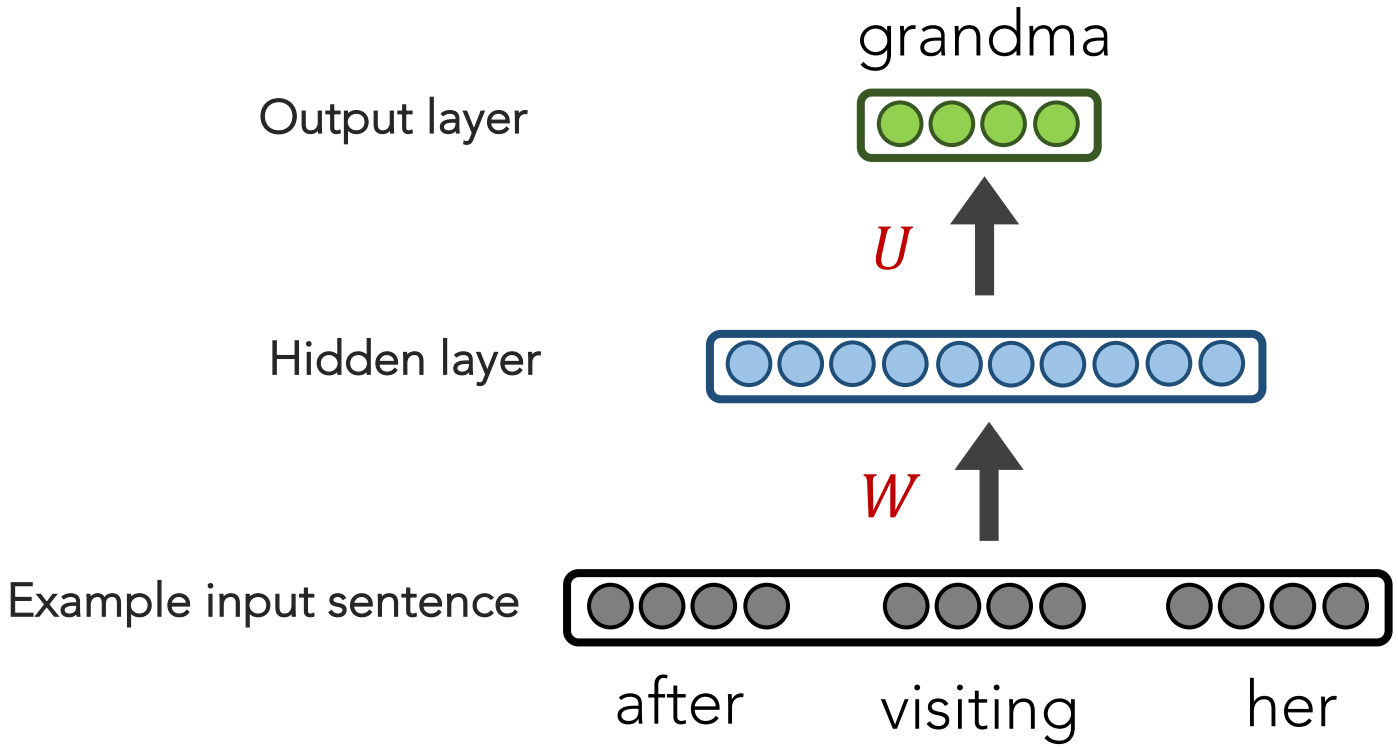
Language Modelling

Neural Approach: Feed-forward Neural Net aren't suitable



Language Modelling

Neural Approach: Feed-forward Neural Net aren't suitable



Language Modelling

FFNN STRENGTHS

- Word embeddings fix the sparsity issue
- Word embeddings fix the storage issue

FFNN ISSUES

- Fixed-window size can never be big enough. Need more context.
- Increasing window size adds many more weights
- The weights awkwardly handle word position
- No concept of time
- Requires inputting entire context just to predict 1 word

Language Modelling

We especially need a system that:

- Has an “infinite” concept of the past, not just a fixed window
- For each new input, output the most likely next event (e.g., word)

Let’s do better

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

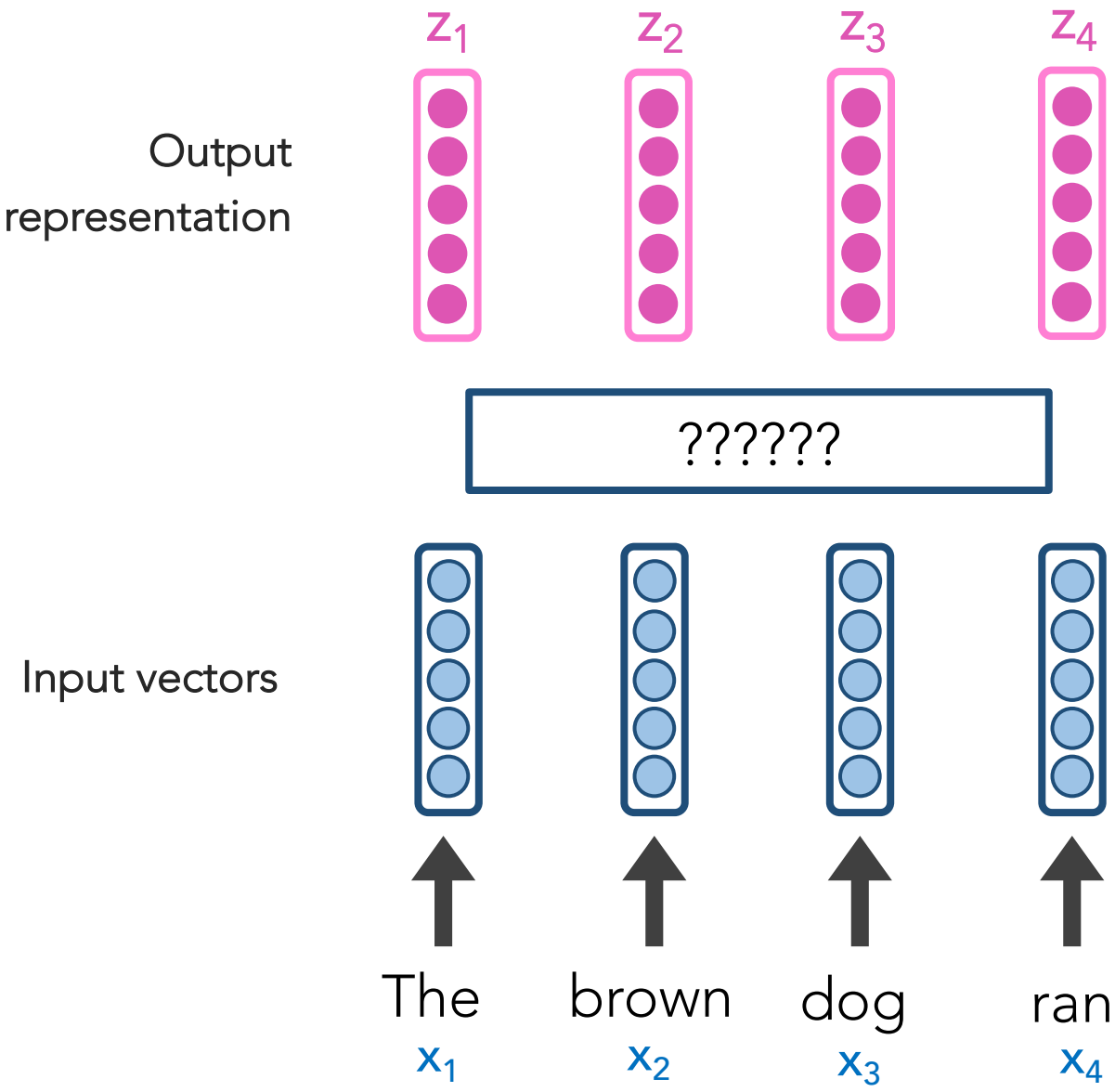
Conclusions

Since we only have an hour, let's jump to the latest, greatest models -- Transformers. For a more comprehensive narrative that builds up, please see the RNNs/LSTMs and Seq2Seq sections (slides are at the end)

Self-Attention

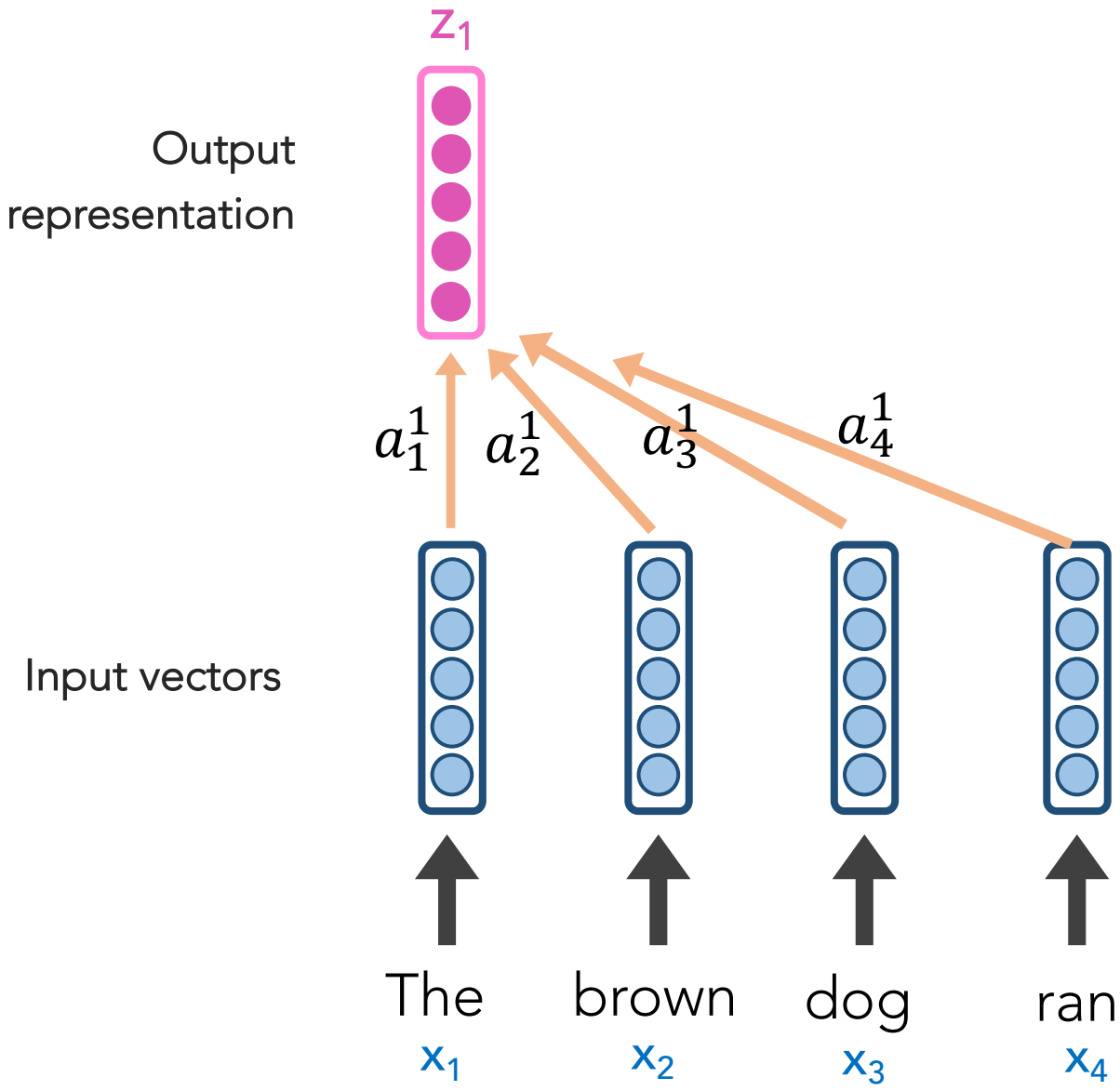
- Models direct relationships between all words in a given sequence (e.g., sentence)
- Each word in a sequence is transformed into an abstract **representation** (embedding) based on the weighted sums of the other words in the same sequence (similar to deep CNN layers)
- Each word for a given layer is effectively addressing, “how much should I be influenced by each of my neighbors”

Self-Attention



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

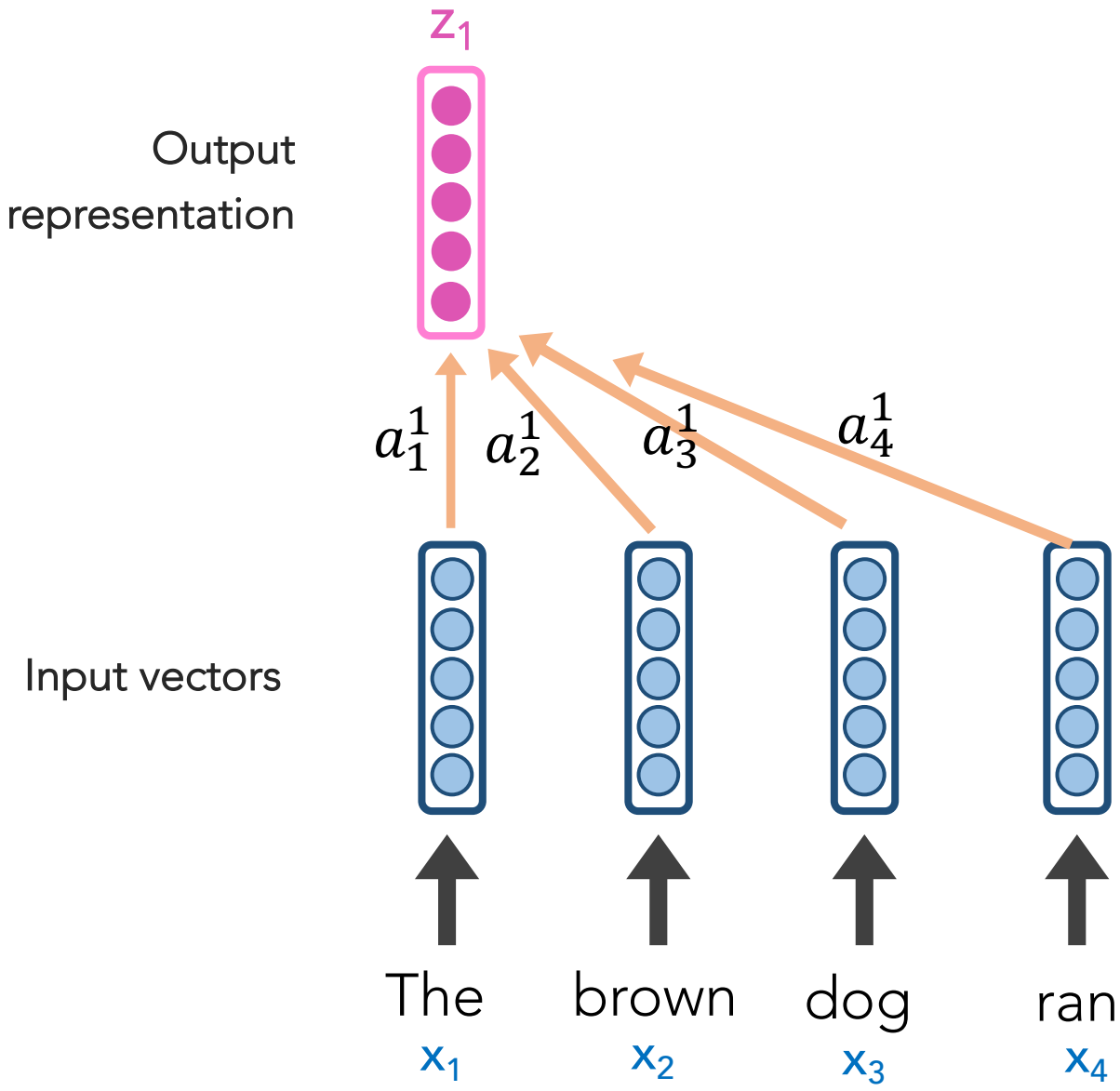
Self-Attention



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

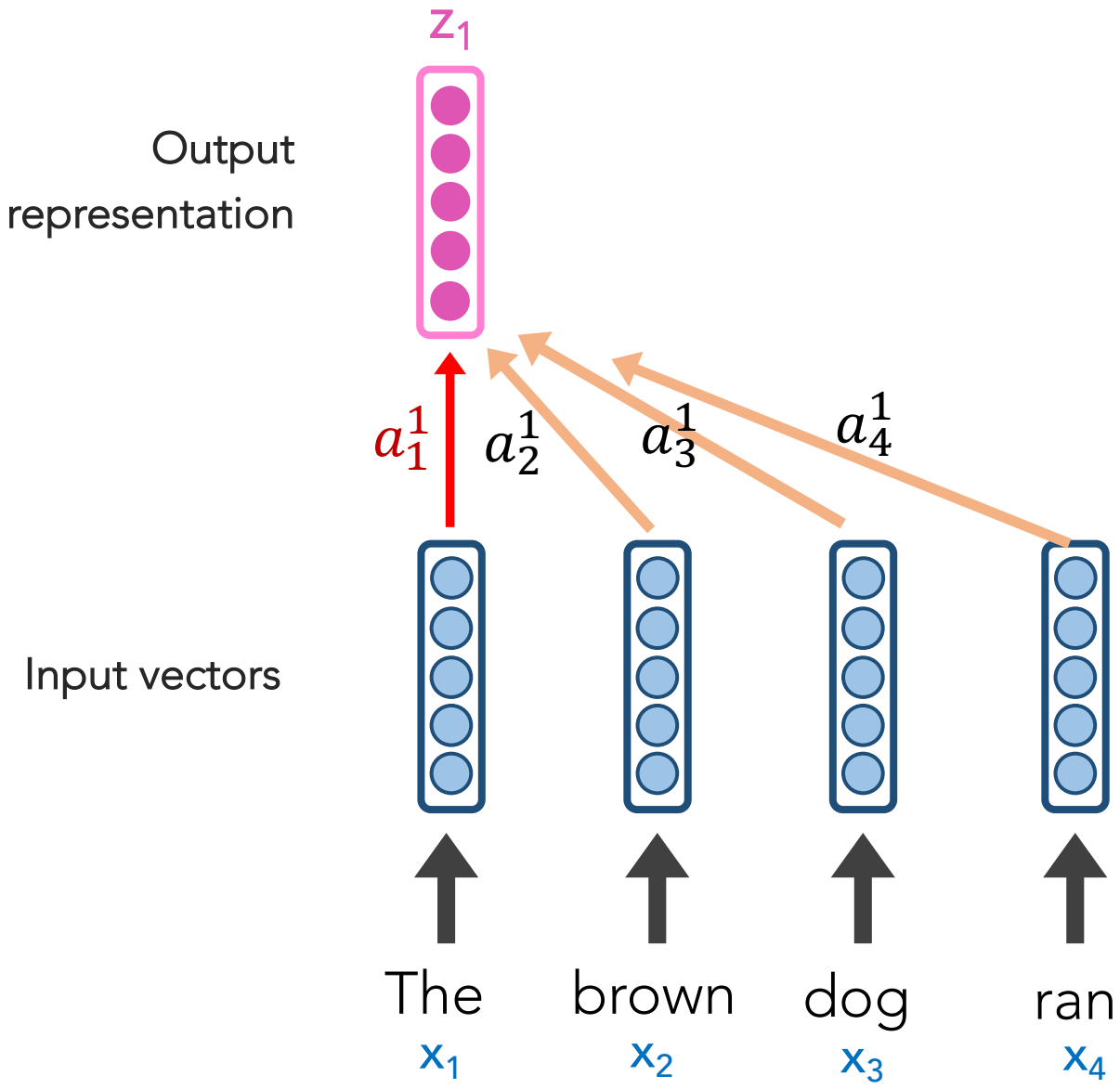


Self-Attention's goal is to create great representations, z_i, of the input

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

a_i^1 isn't a simple 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 q_i
- Key k_i
- Value v_i

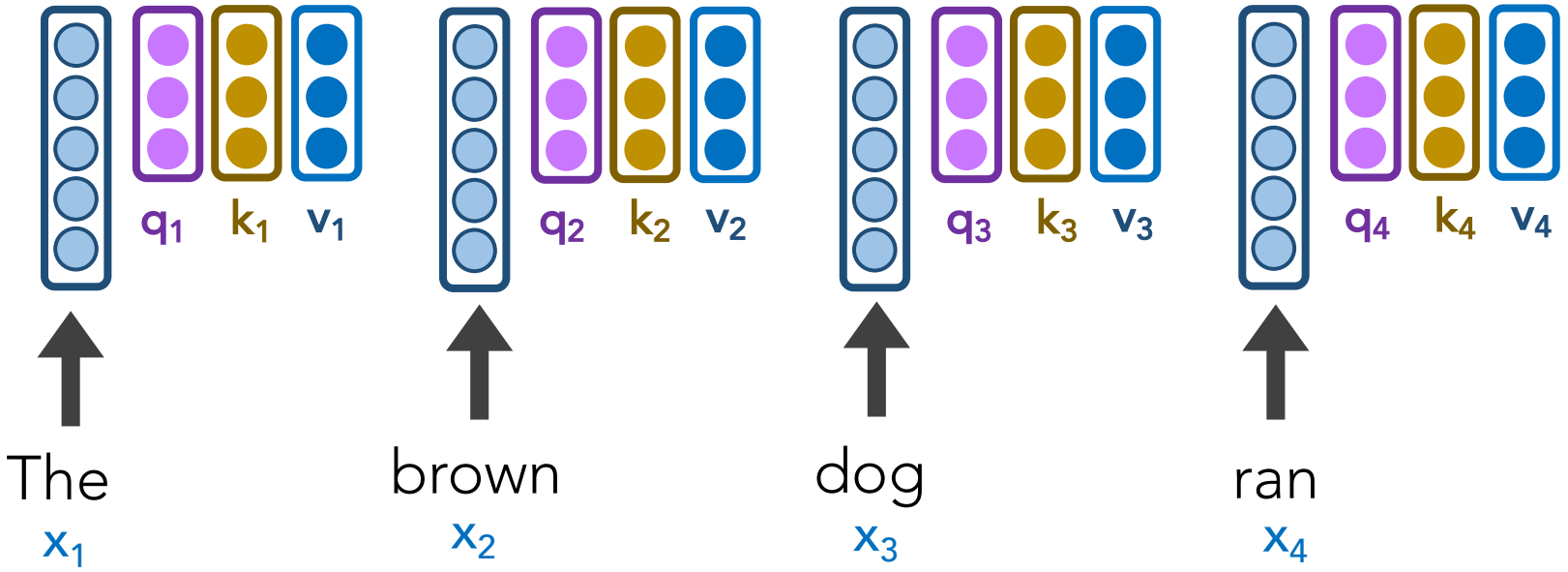
Self-Attention

Step 1: Our model has 3 weight matrices W_q, W_k, W_v which are multiplied by each x_i to create all associated vectors:

$$q_i = w_q x_i$$

$$k_i = w_k x_i$$

$$v_i = w_v x_i$$



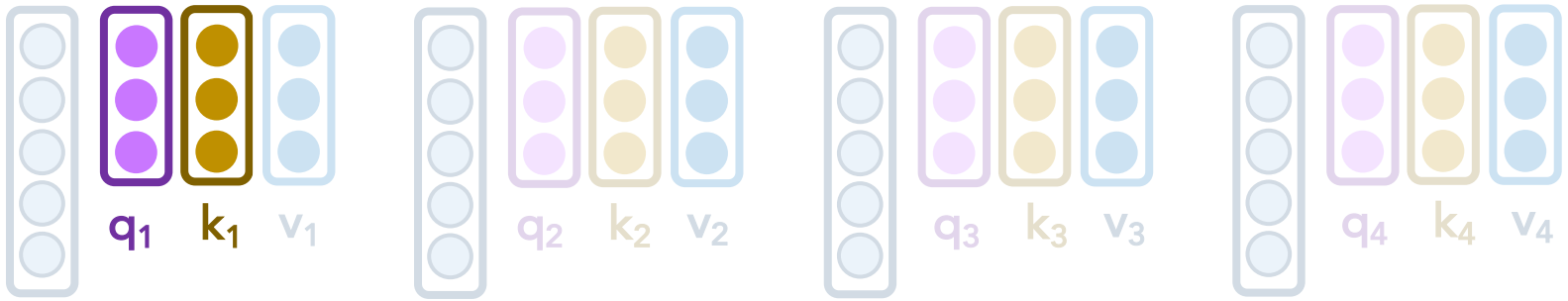
Under the hood, each x_i has 3 small, associated vectors. For example, x_1 has:

- Query q_1
- Key k_1
- Value 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

$s_1 = q_1 \cdot k_1 = 112$



The
x_1

brown
x_2

dog
x_3

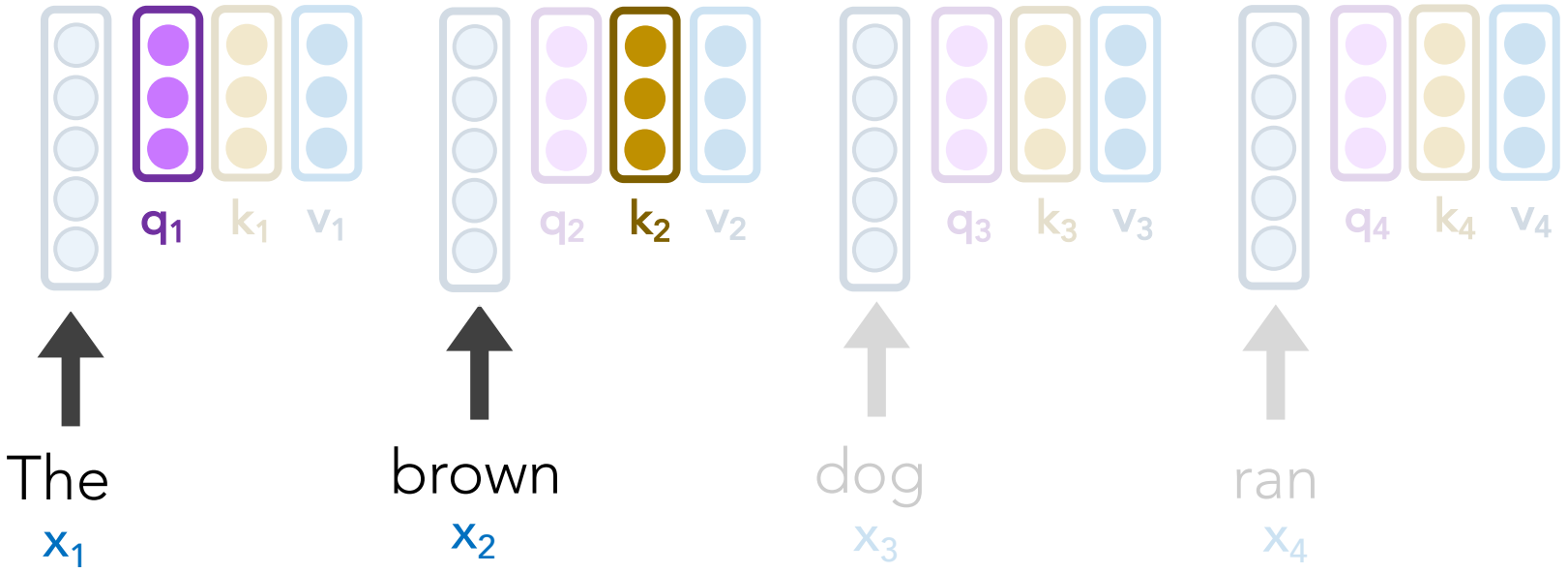
ran
x_4

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

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



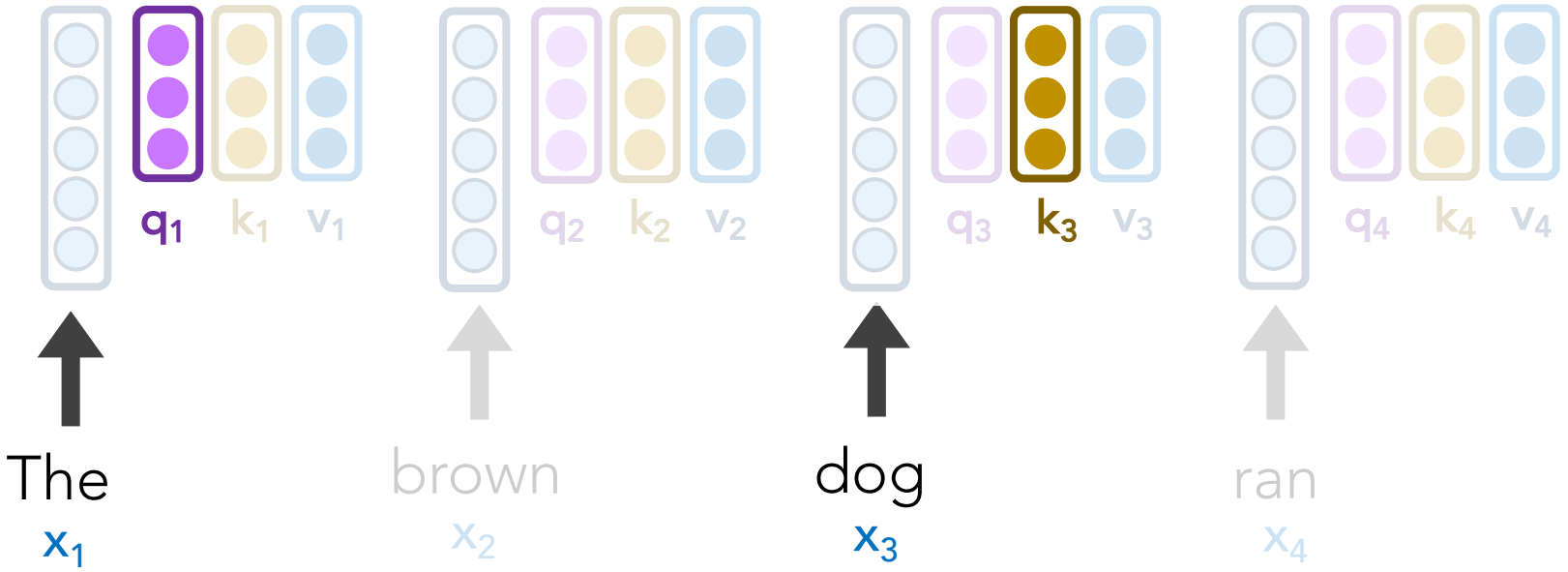
Self-Attention

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

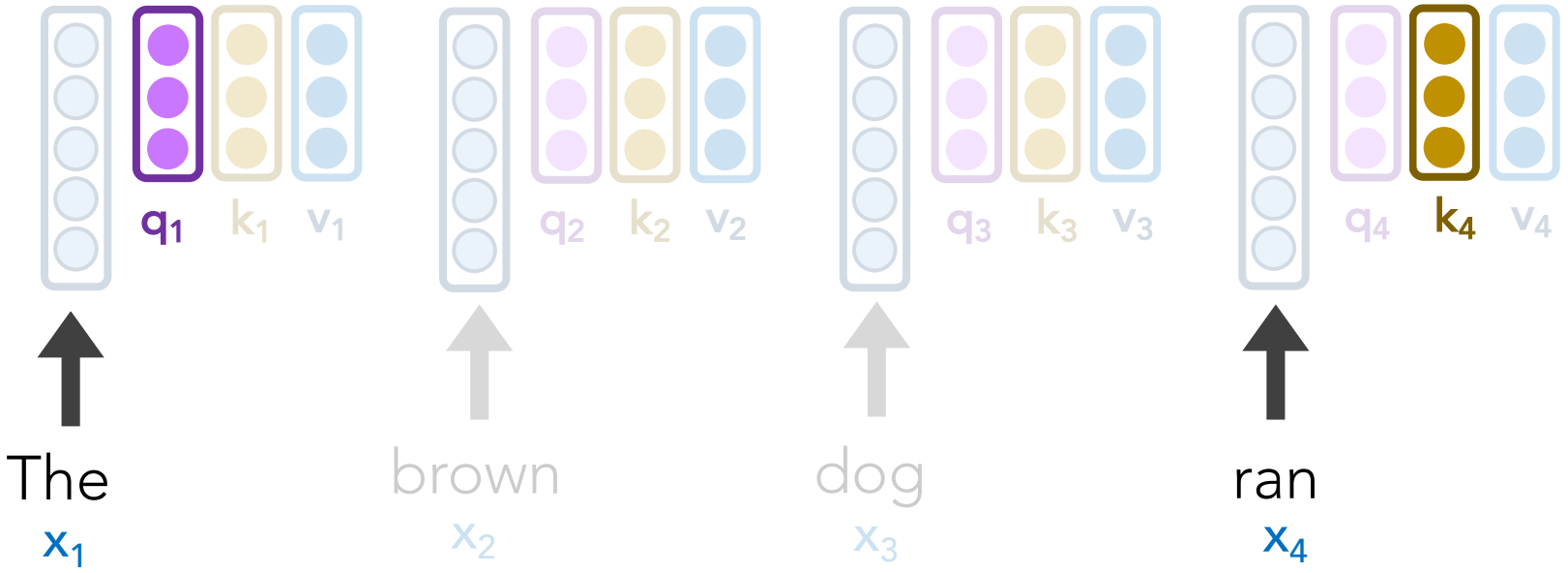
Step 2: For word x_1, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_4 = q_1 \cdot k_4 = 8$

$s_3 = q_1 \cdot k_3 = 16$

$s_2 = q_1 \cdot k_2 = 96$

$s_1 = q_1 \cdot k_1 = 112$



Self-Attention

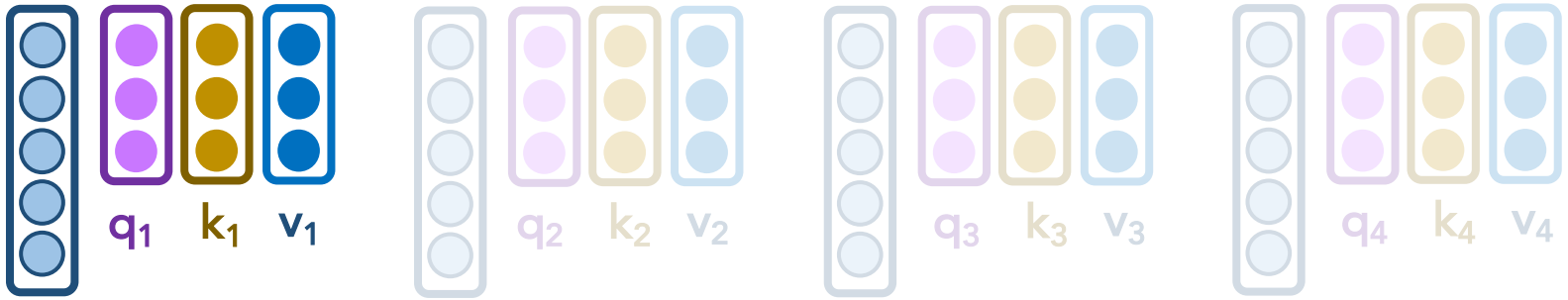
Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

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

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

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

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



The x_1 brown x_2 dog x_3 ran x_4

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

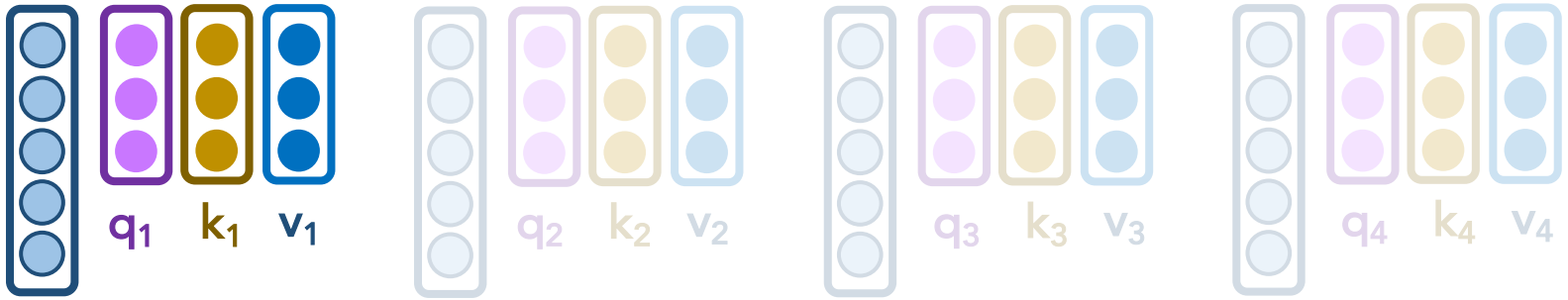
$s_4 = q_1 \cdot k_4 = 8$ $\quad a_4 = \sigma(s_4/8) = 0$

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

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

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

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



The
x_1

brown
x_2

dog
x_3

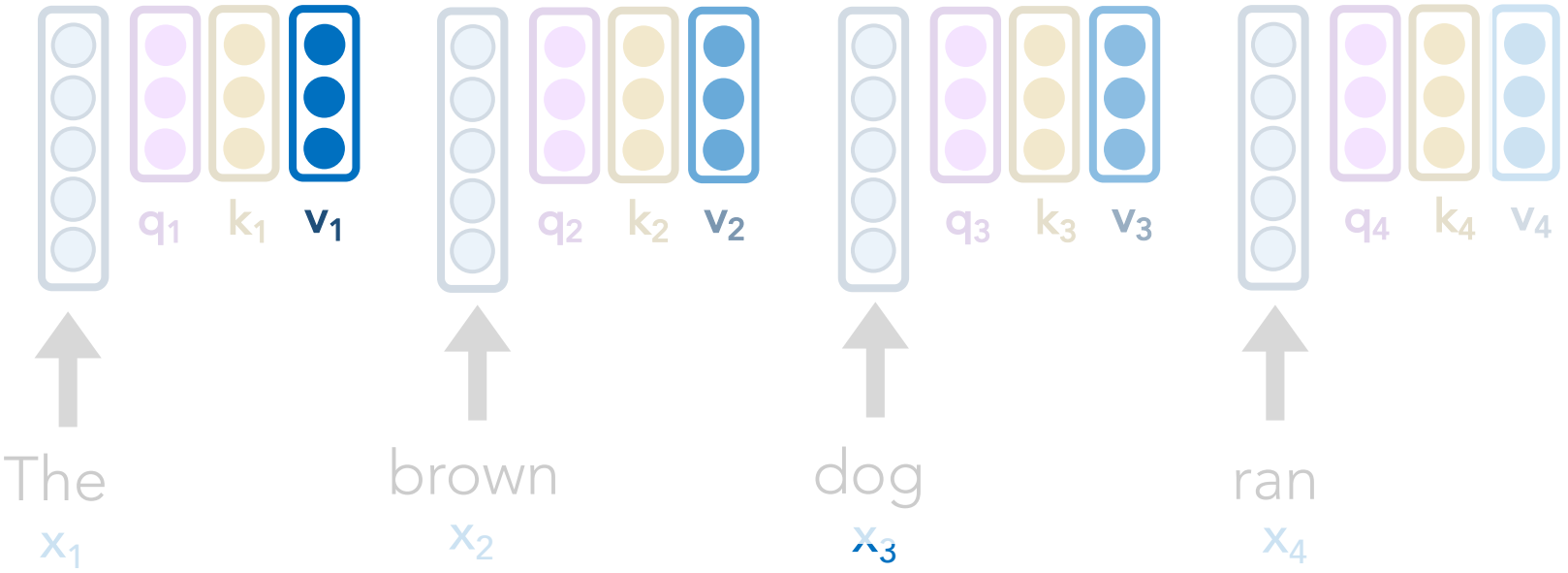
ran
x_4

Self-Attention

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

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

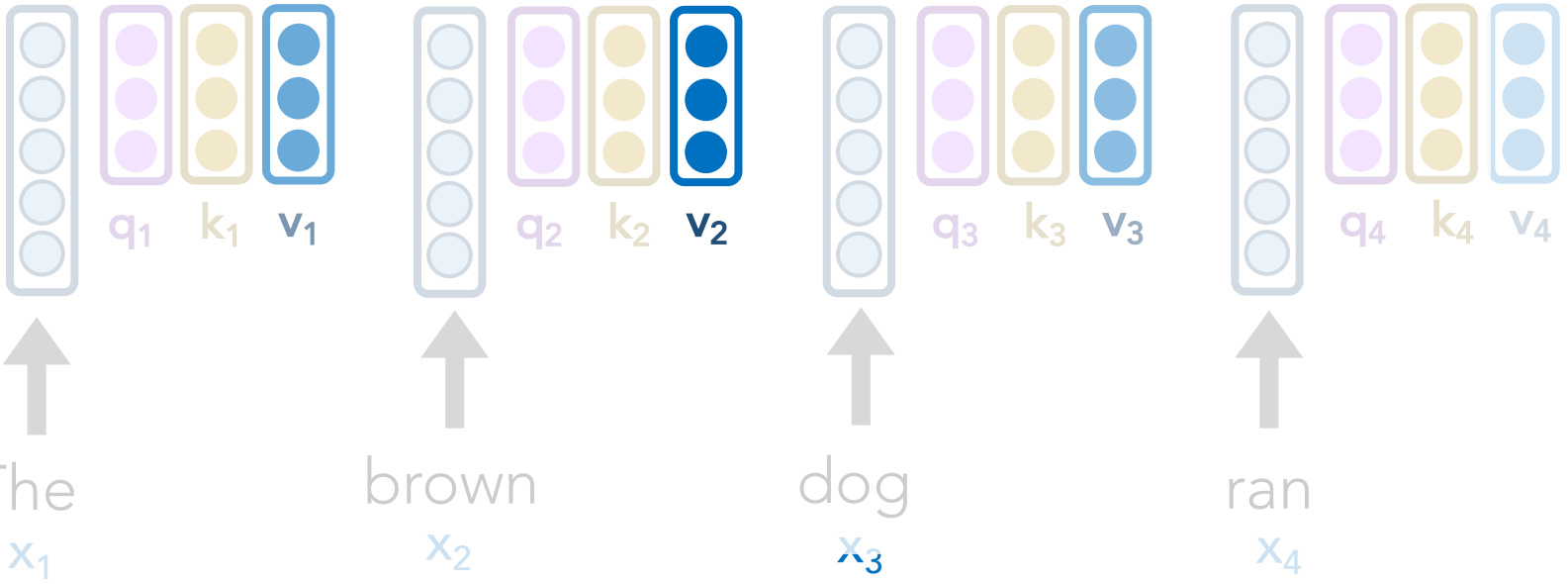
$$= 0.87 \cdot v_1 + 0.12 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



Self-Attention

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

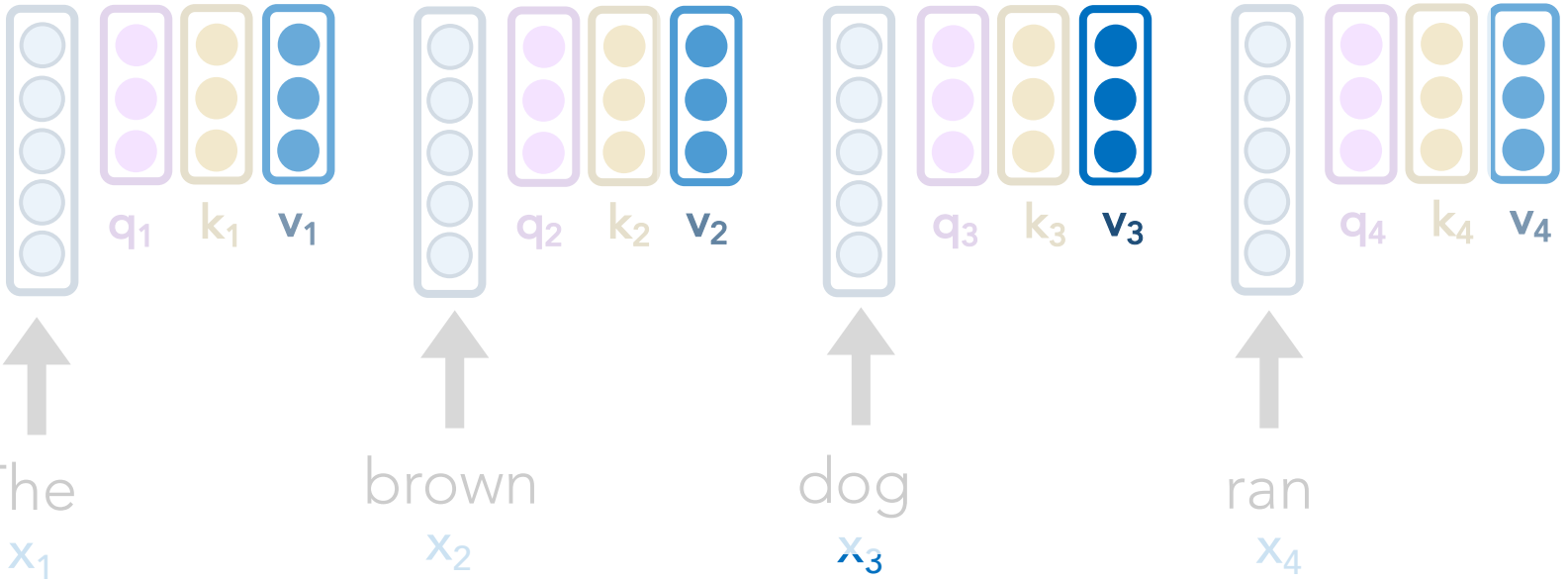
$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$



Self-Attention

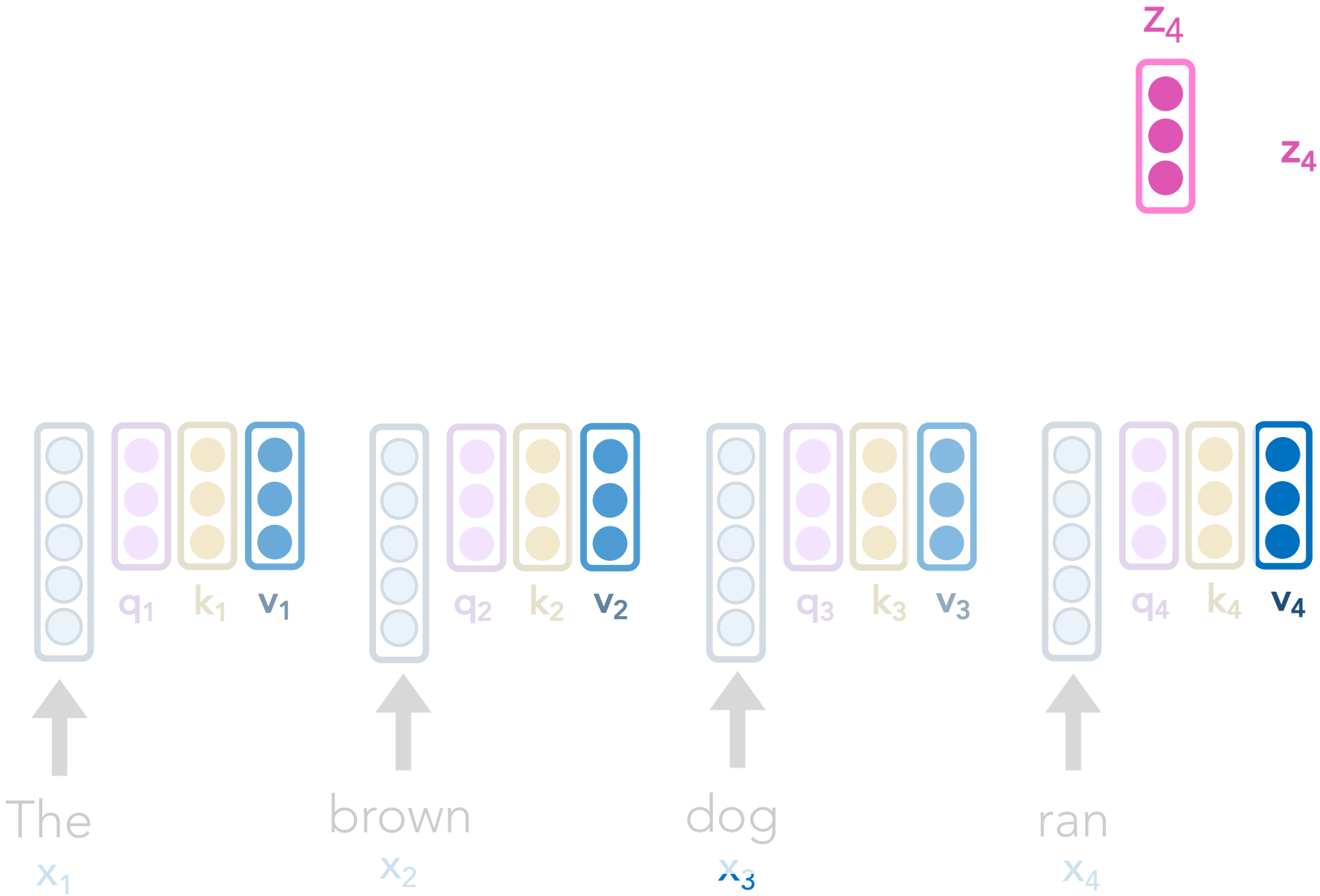
Step 5: We repeat this for all other words, yielding us with great, new z_i representations!

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



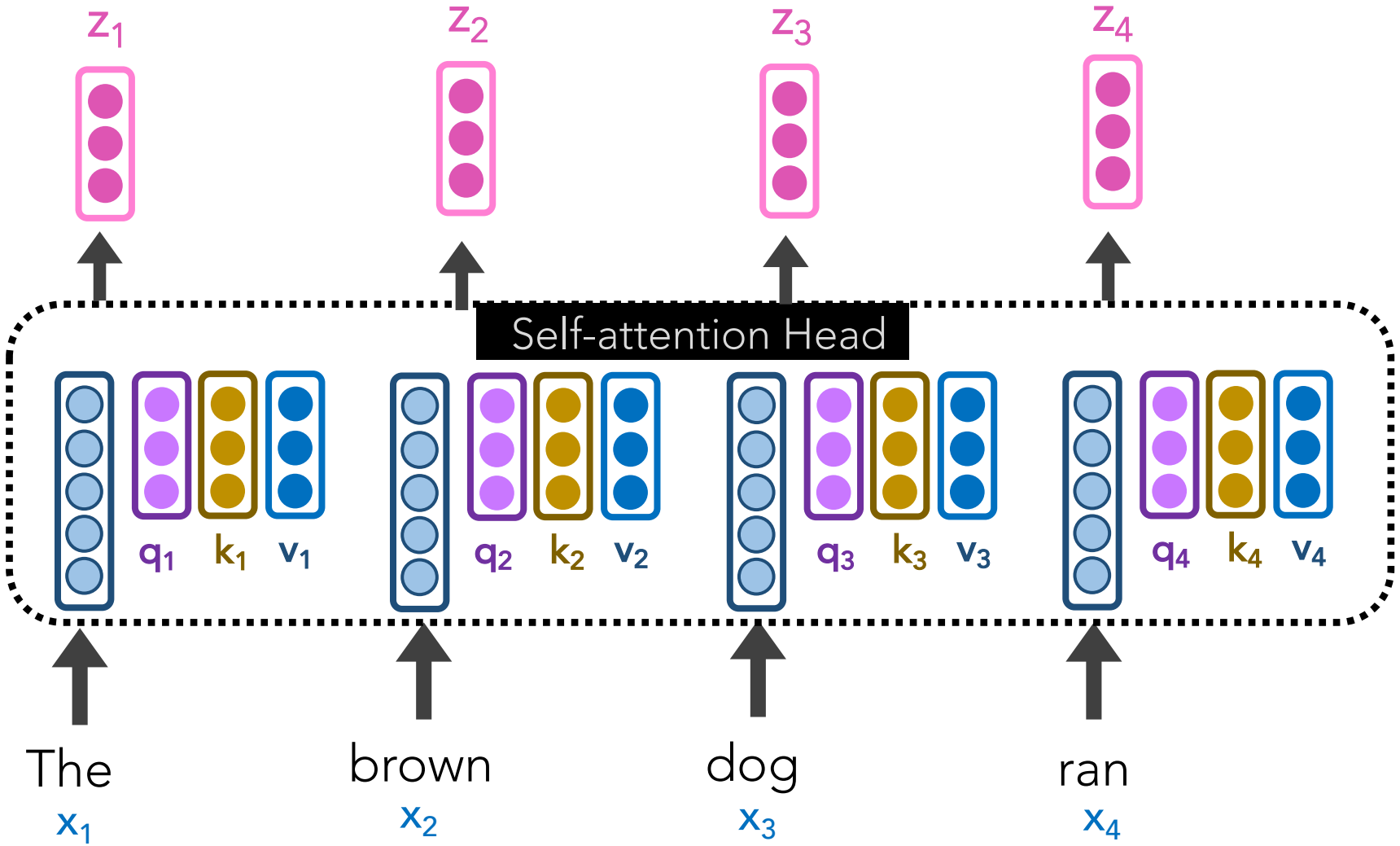
Self-Attention

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



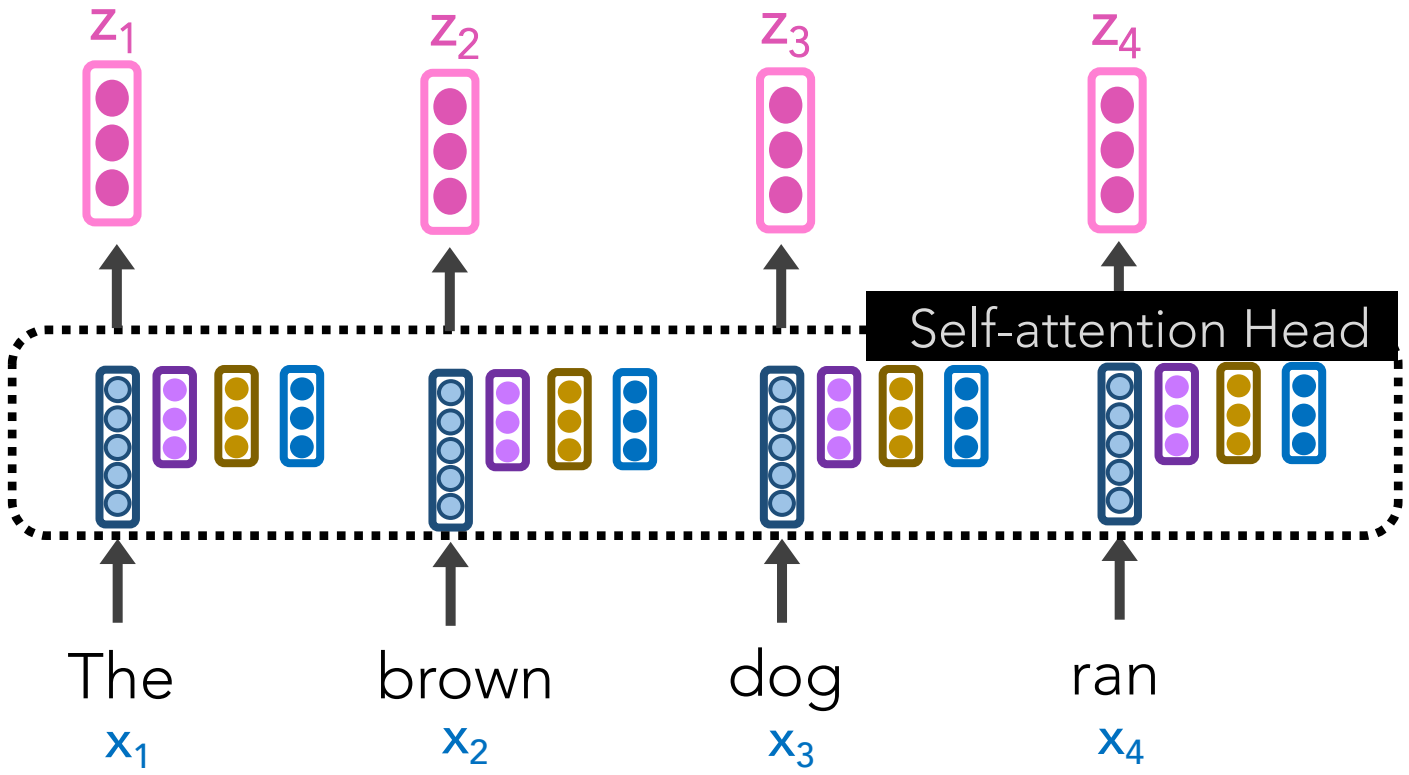
Self-Attention

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



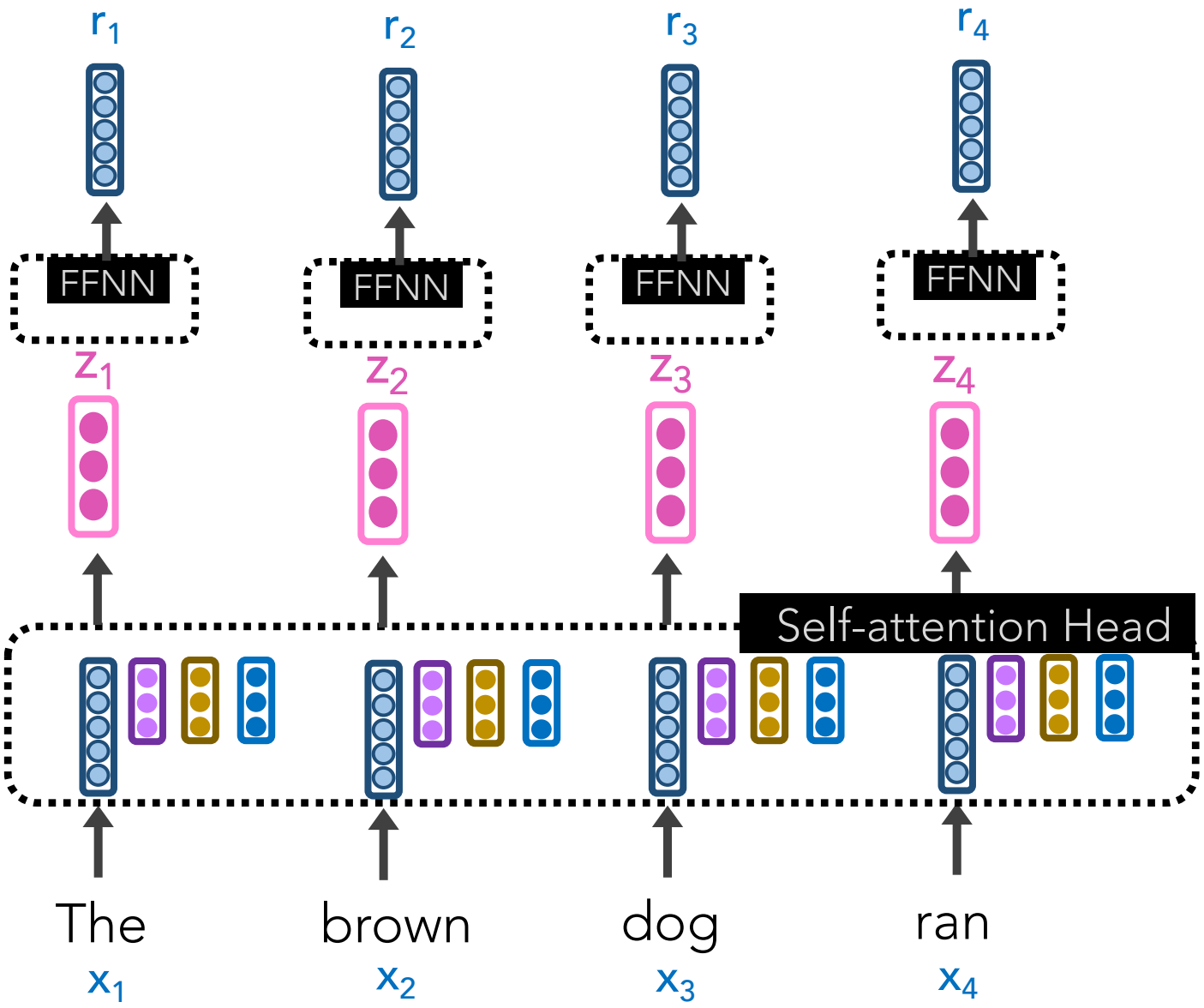

Takeaway #4:
Self-Attention is powerful; allows us to create great, context-aware representations

Self-Attention



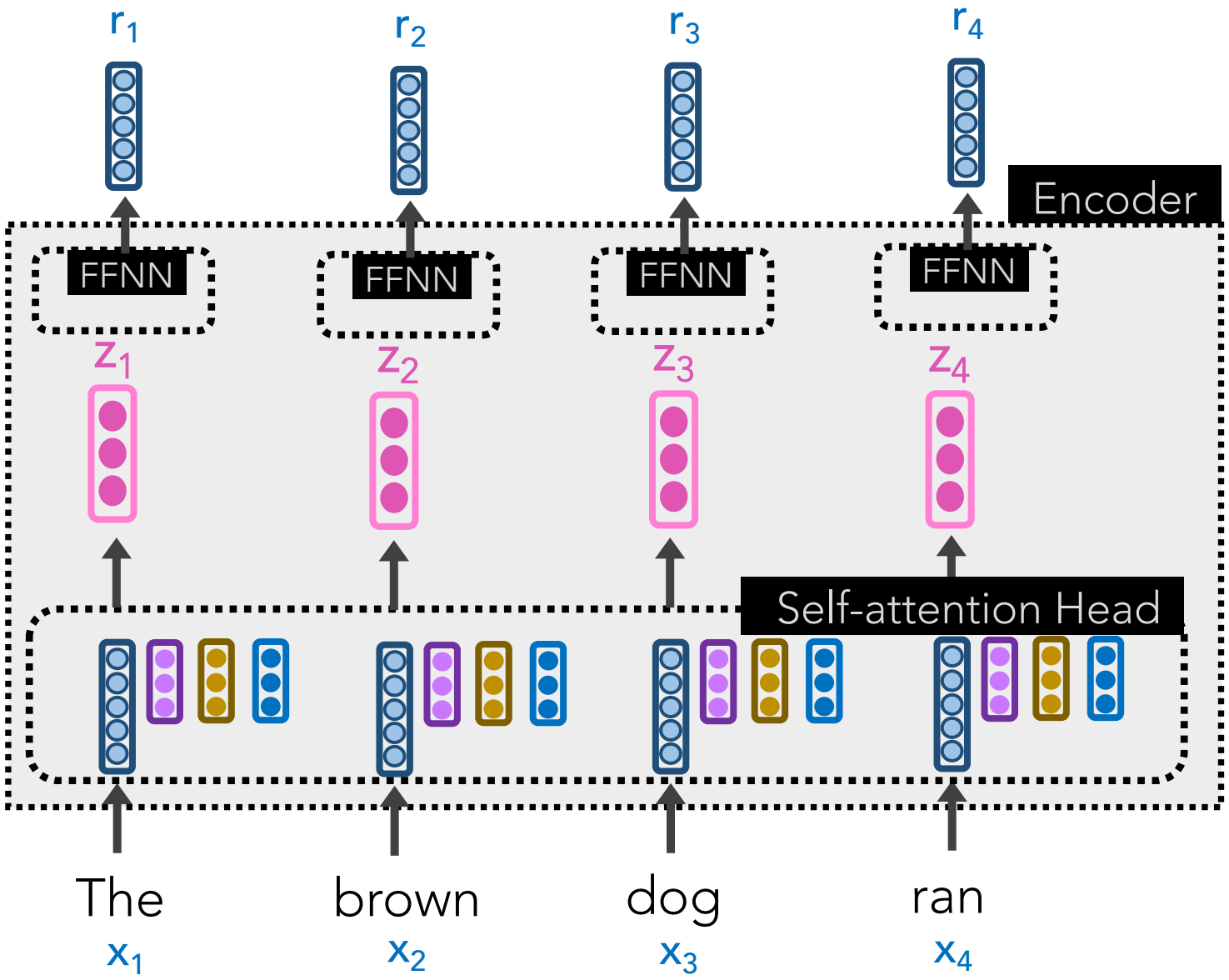
Let's further pass each z_i through the same feed-forward neural net (FFNN)

Self-Attention + FFNN



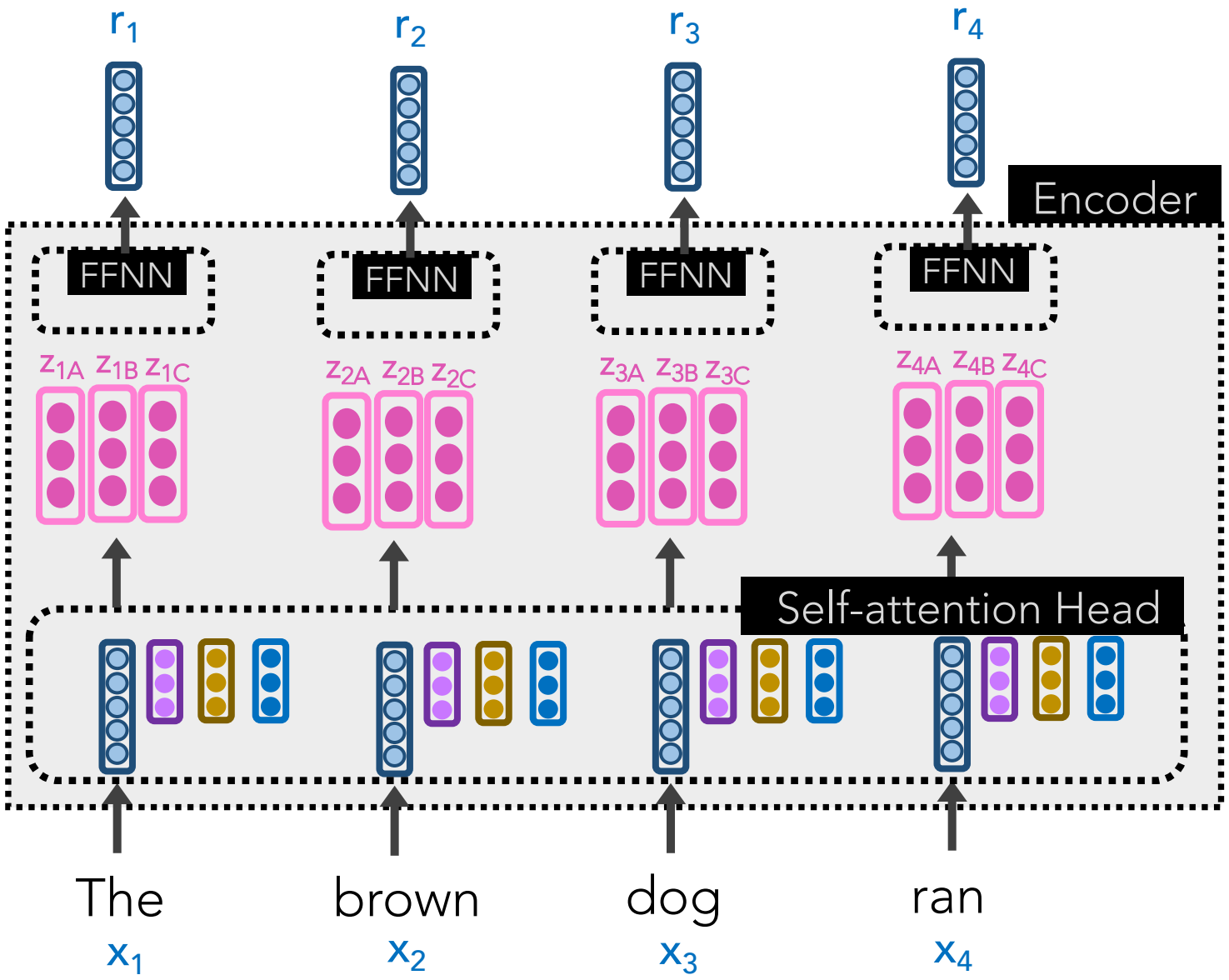
Let's further pass each z_i through the same feed-forward neural net (FFNN)

Transformer Encoder



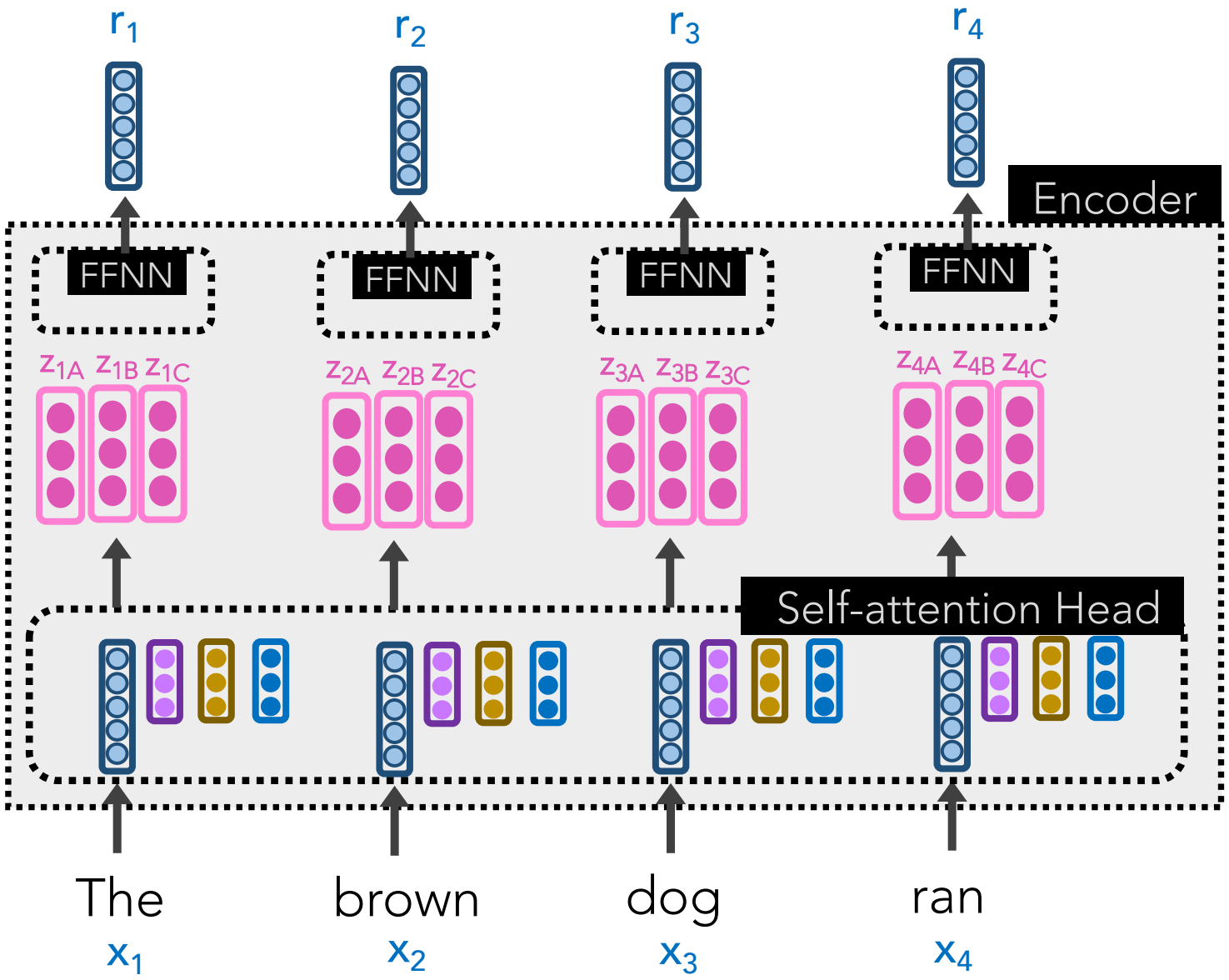
Yay! Our new r_i vectors are our new representations, and this entire process is called an **Transformer Encoder**

Transformer Encoder



The Attention Head produces our z_i vectors. Why rely on this single attention mechanism? We could simultaneously use multiple heads and combine them into our same, single FFNN.

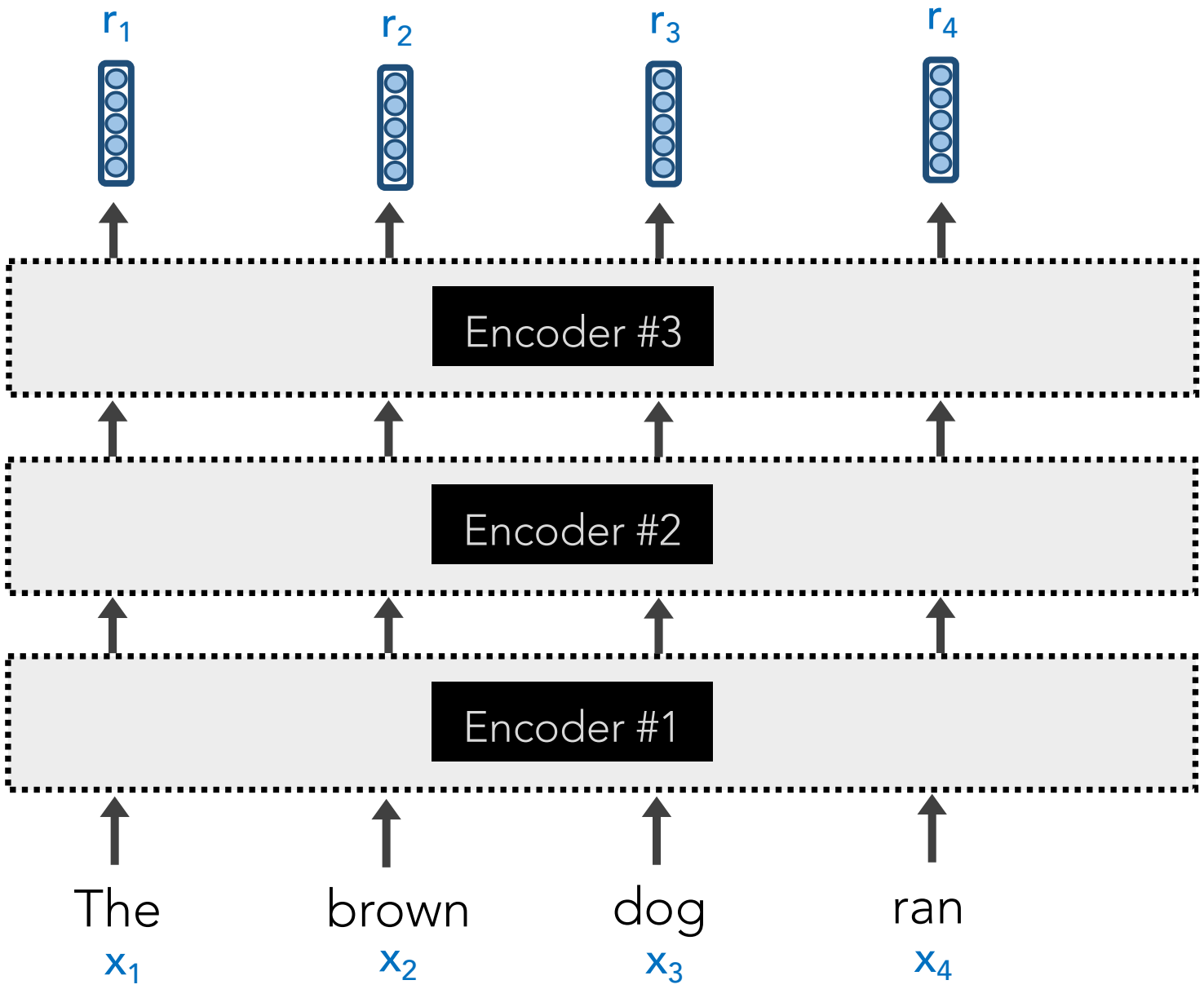
Transformer Encoder



The Attention Head produces our z_i vectors. Why rely on this single attention mechanism? We could simultaneously use multiple heads and combine them into our same, single FFNN.

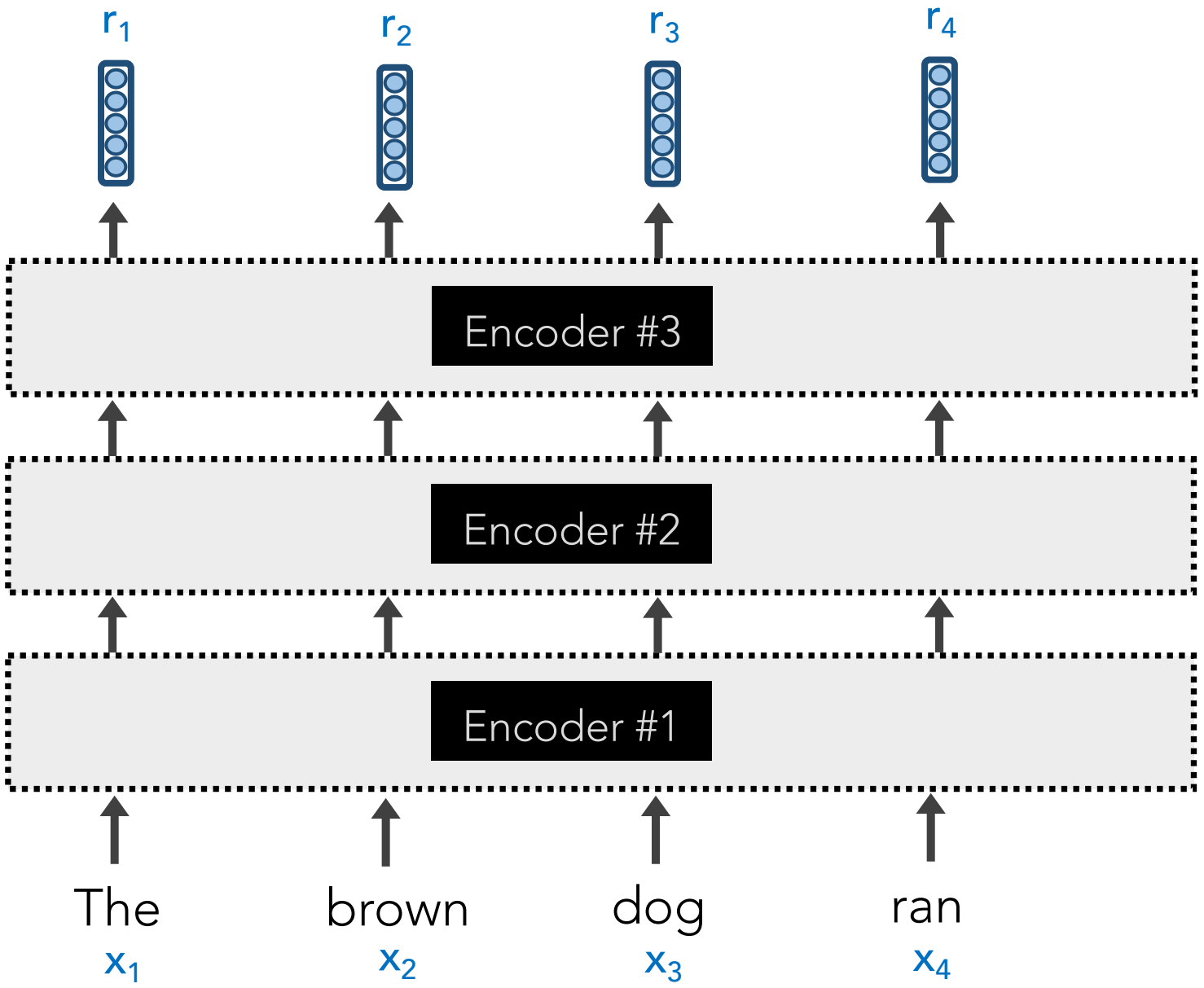
Likewise, why stop with just 1 Transformer Encoder? We could stack several!

Transformer Encoders



To recap: all of this looks fancy, but ultimately it's just producing a very good **context-aware representation** of each word

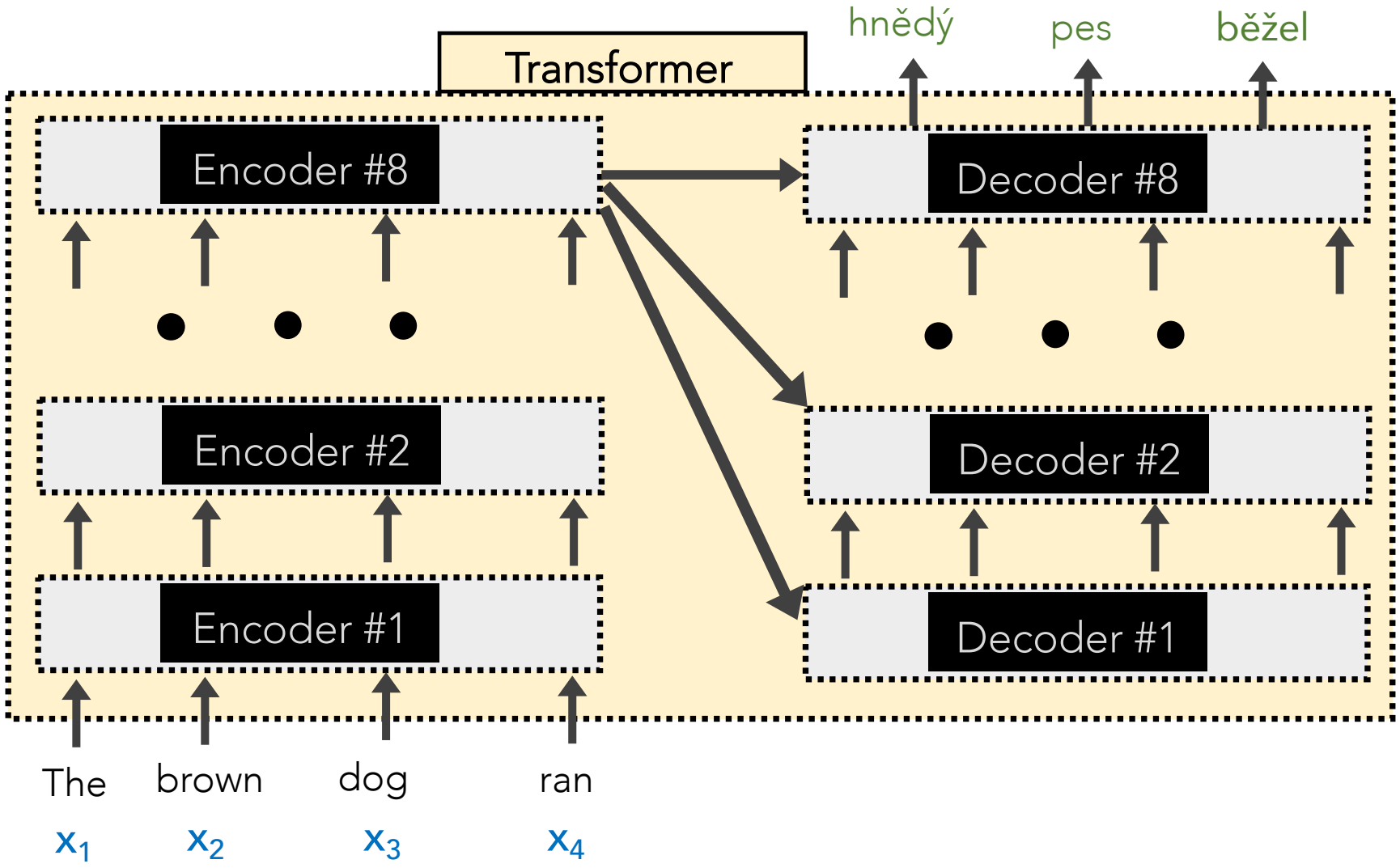
Transformer Encoders



To recap: all of this looks fancy, but ultimately it's just producing a very good **context-aware representation** of each word

If we wish to output a sequence from this, we need Transformer Decoders, too!

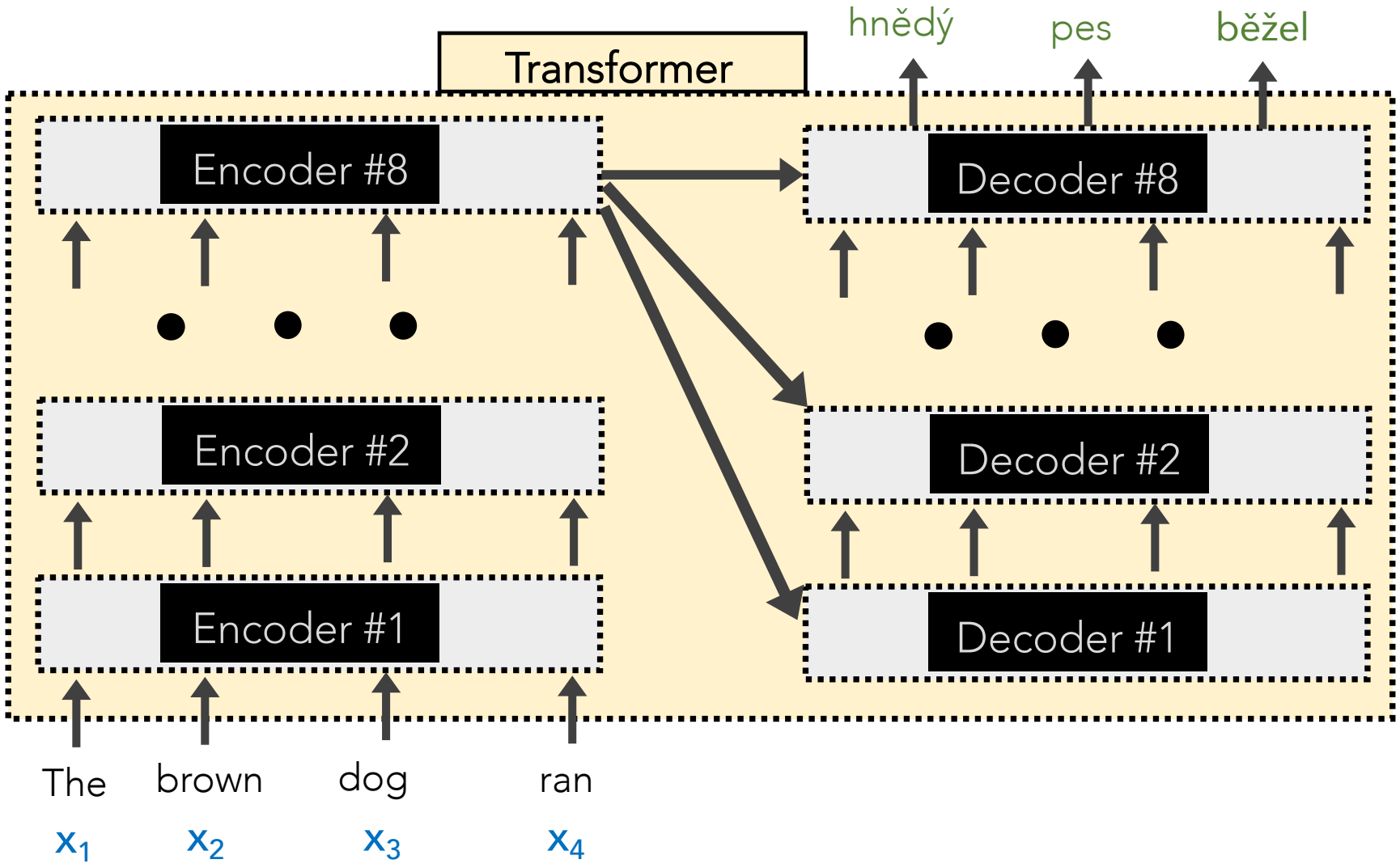
Transformer Encoders and Decoders



Transformer Encoders produce **context-aware representation** of each word

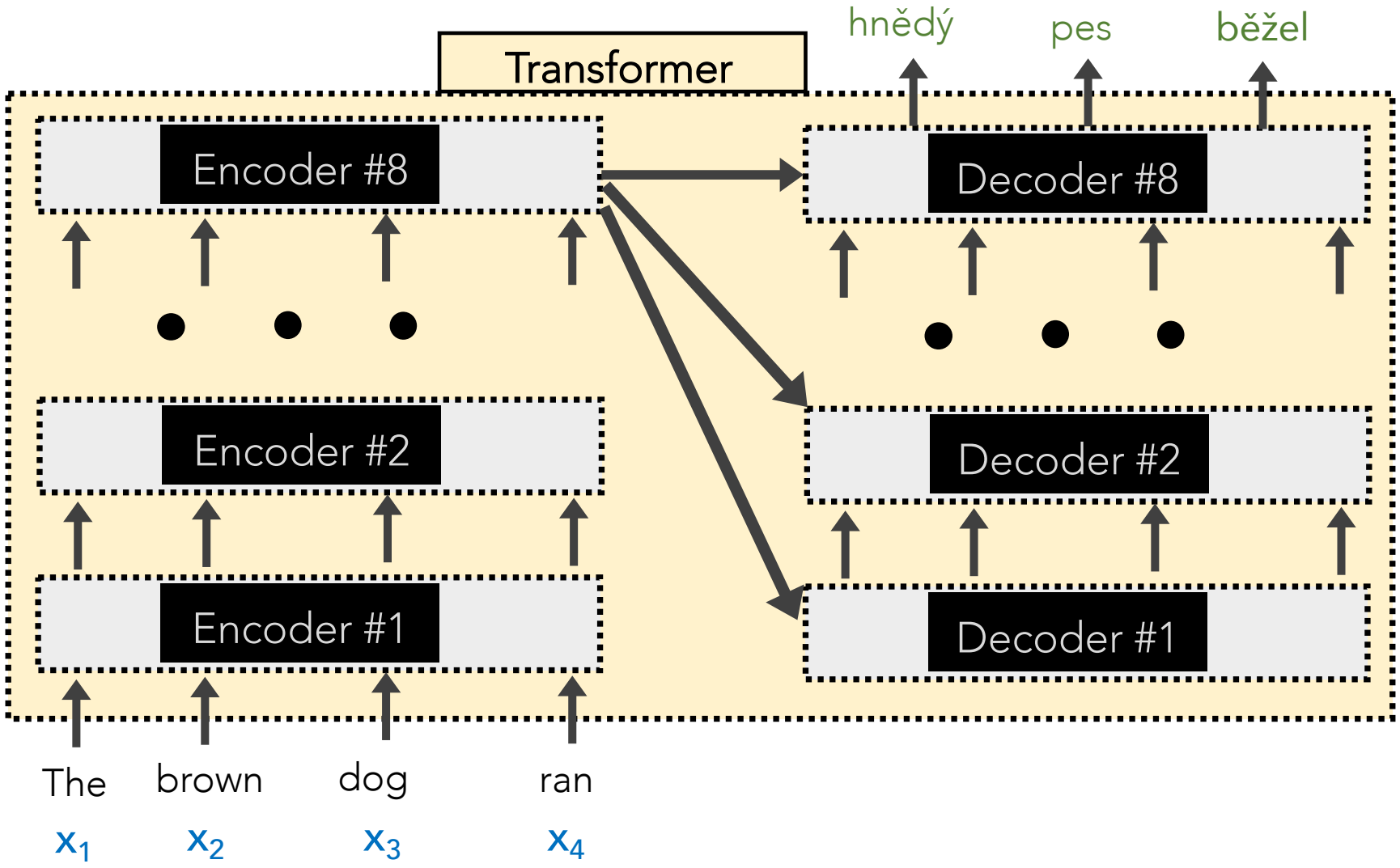
Transformer Decoders generate new sequences of text

Transformer Encoders and Decoders



Transformer Decoders are identical to the Encoders, except they have an additional Attention Head (in between the Self-Attention and FFNN layers) that focuses on parts of the encoder's representations.

Transformer Encoders and Decoders



Additional Details:

- We also input **positional information** into the Encoder for each word
- To help ensure integrity, residual connections are made before each FFNN
- It's common to chunk words into **sub-word pieces**, to prevent sparsity

Transformer

To recap:

- Attention determines which pieces of sequence A are most relevant w.r.t. sequence B
- Self-attention determines which pieces of sequence A are most relevant w.r.t. sequence A
- A Transformer combines both
- Transformers yield state-of-the-art results on most of the common NLP tasks

BERT (a Transformer variant)

What if we don't want to generate a new output sequence, but just use the rich Encoder representations?

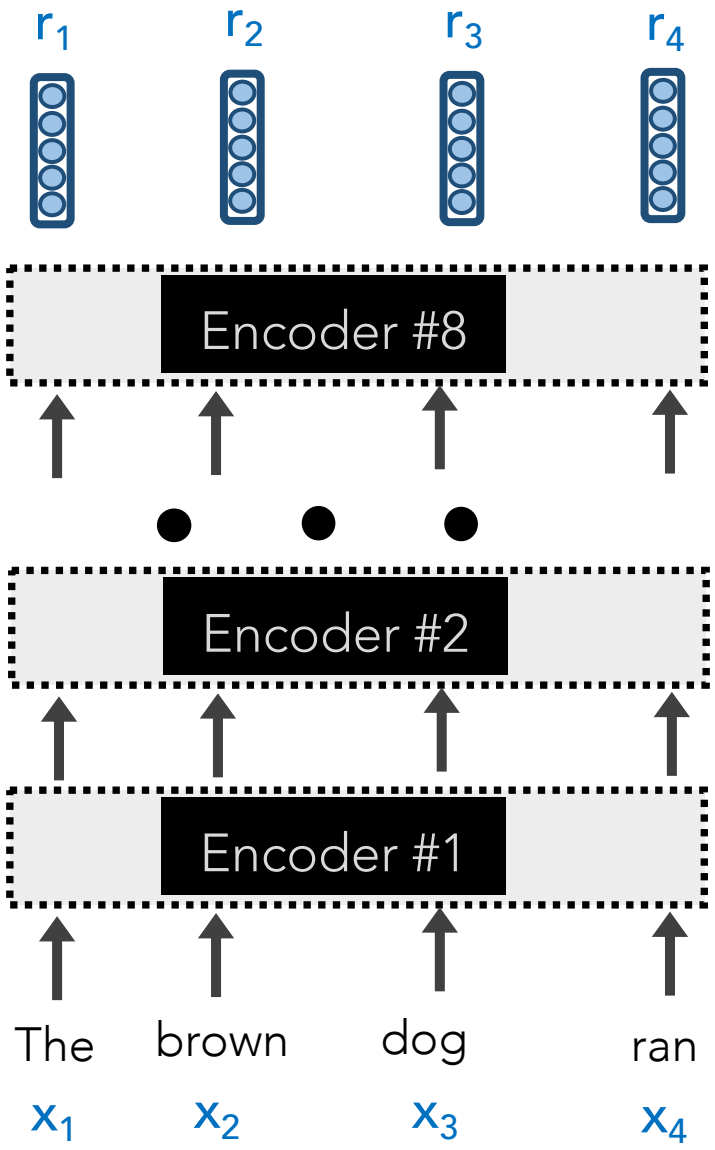
BERT model to the rescue!

Bidirectional Encoder Representations from Transformers

BERT (a Transformer variant)

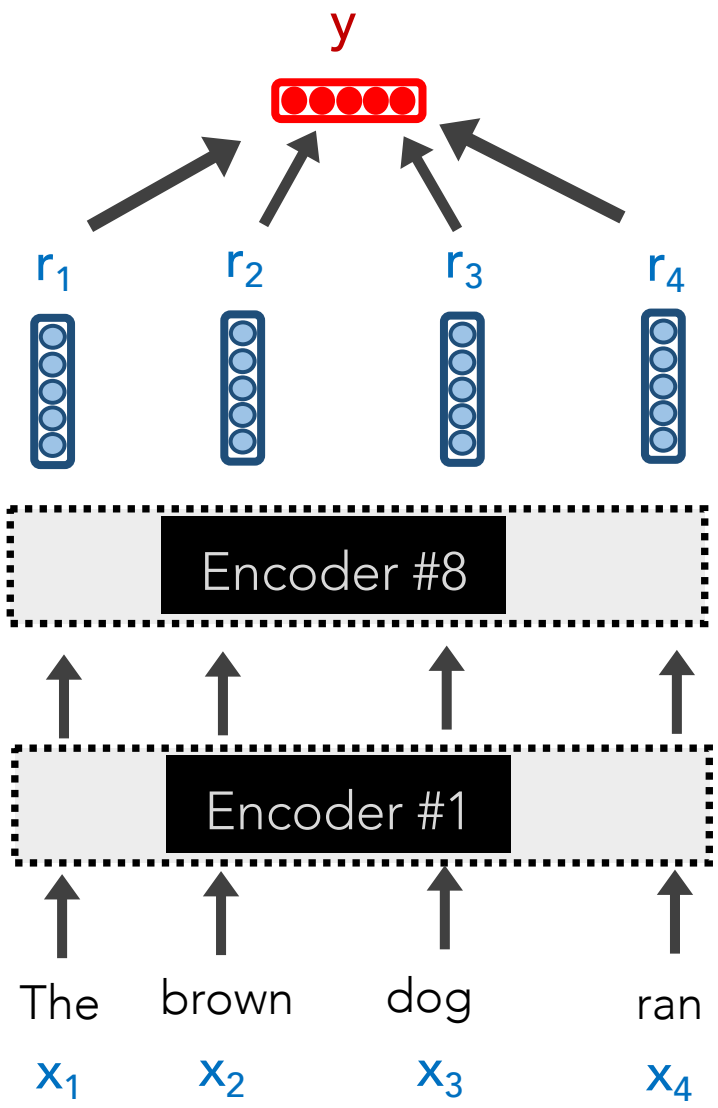
- BERT uses only Transformer Encoders (no Decoders) to learn great representations of word sequences.
- It's bidirectional; can see words at any position
- It's trained by:
  - masking random words (15%) and aiming to predict them
  - Being given two sentences at a time and forced to predict if sentence B logically follows sentence A or not (helps learn coherence)

BERT (a Transformer variant)



Typically, one uses BERT's awesome embeddings to fine-tune toward a different NLP task (this is called Sequential Transfer Learning)

BERT (a Transformer variant)



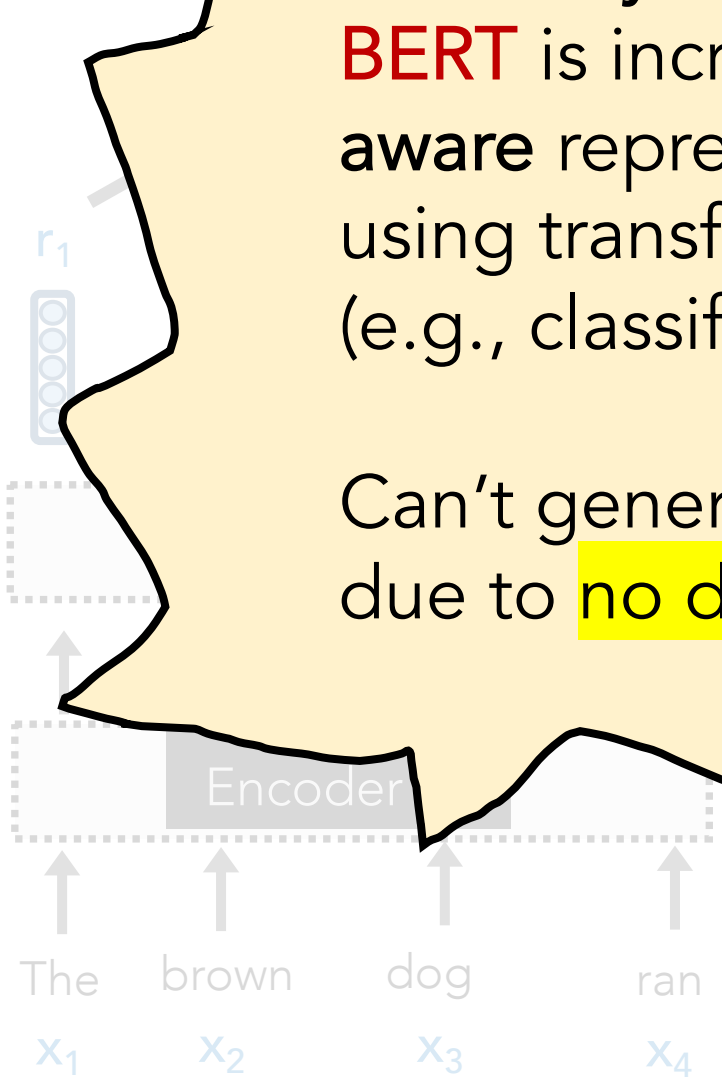
Typically, one uses BERT's awesome embeddings to fine-tune toward a different NLP task (this is called Sequential Transfer Learning)

BERT (a Transformer variant)

Takeaway #5:

BERT is incredible for learning **context-aware** representations of words and using transfer learning for other tasks (e.g., classification).

Can't generate new sentences though, due to no decoders.



Transformer

What if we want to generate a new output sequence?

GPT-2 model to the rescue!

Generative Pre-trained Transformer 2

GPT-2 (a Transformer variant)

- GPT-2 uses only Transformer Decoders (no Encoders) to generate new sequences
- As it processes each word/token, it cleverly masks the “future” words and conditions itself on the previous words
- Can generate text from scratch or from a starting sequence.
- Easy to fine-tune on your own dataset (language)
- GPT-3 is an even bigger version (175B parameters) of GPT-2, but isn’t open-source

GPT-2

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

Takeaway #6:
GPT-2 is astounding for generating realistic-looking new text

Can fine-tune toward other tasks, too.

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

Conclusion

- There has been significant progress in the past few years.
- Some of the complex models are incredible, but rely on having a lot of data and computational resources (e.g., Transformers)
- With all data science and machine learning, it's best to understand your data and your task very well, then clean the data, and start with a simple model (instead of jumping to the most complex model)

Conclusion

Models

- **N-gram**: count statistics; **elementary** sequence modelling
- **FFNN**: fixed-length context window; basic sequence modelling
- **(Vanilla) RNN**: uses context; fair sequence modelling
- **LSTM**: a variant of an RNN that handles long-range context better
- **Seq2Seq**: maps 1 sequence to another (n→m sequences)
- **Attention**: determines which elements in sequence A pertain to sequence B
- **Self-Attention**: determines great representations for items in a sequence
- **Transformers**: learns excellent representation, via a seq2seq framework with self-attention and attention
  - **BERT**: Transformer Encoders that learn great representations and can be fine-tuned on other tasks
  - **GPT-2**: Transformer Decoders that generate realistic text and can be fine-tuned on other tasks

Takeaways #1-3

1. word embeddings are very useful

2. LSTMs are a great starting point for modelling language (or other time series).

Can make:
n → n predictions or n → 1 predictions

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

Attention is powerful; allows us to conditionally weight our focus

Takeaways #4-6

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

5. BERT is incredible for learning **context-aware** representations of words and using transfer learning for other tasks (e.g., classification).

Can't generate new sentences though, due to no decoders.

6. GPT-2 is astounding for generating realistic-looking new text

Can fine-tune toward other tasks, too

Credit & further resources:

- Backprop: http://cs231n.github.io/optimization-2/
- Abigail See's lectures: http://web.stanford.edu/class/cs224n/index.html
- Andrew Ng (Attention): https://www.youtube.com/watch?v=quoGRI-1l0A
- Google (Transformers): https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- Visualizing seq2seq with Attention: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- Illustrated BERT: http://jalammar.github.io/illustrated-bert/
- Illustrated GPT-2: https://jalammar.github.io/illustrated-gpt2/
- Illustrated Transformer: https://jalammar.github.io/illustrated-transformer/
- HuggingFace's Transformers: https://huggingface.co/transformers/summary.html
- Chris McCormick's BERT Series: https://www.youtube.com/watch?v=FKlPCK1uFrc
- Michael Collins' Language Modelling/Transformer series: https://www.youtube.com/watch?v=jfwqRMdTmLo

Conclusion

Questions?

Language Modelling

RNNs/LSTMs

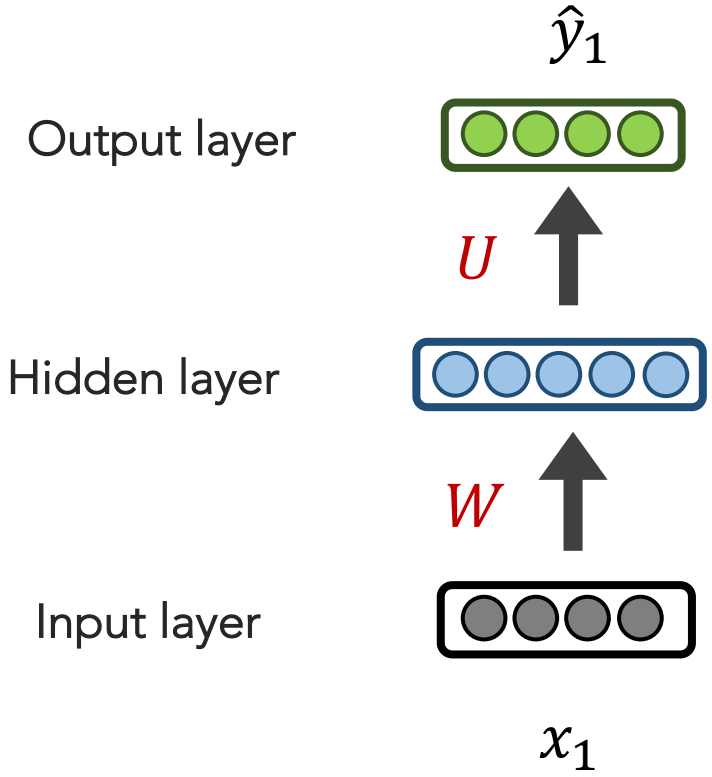
Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

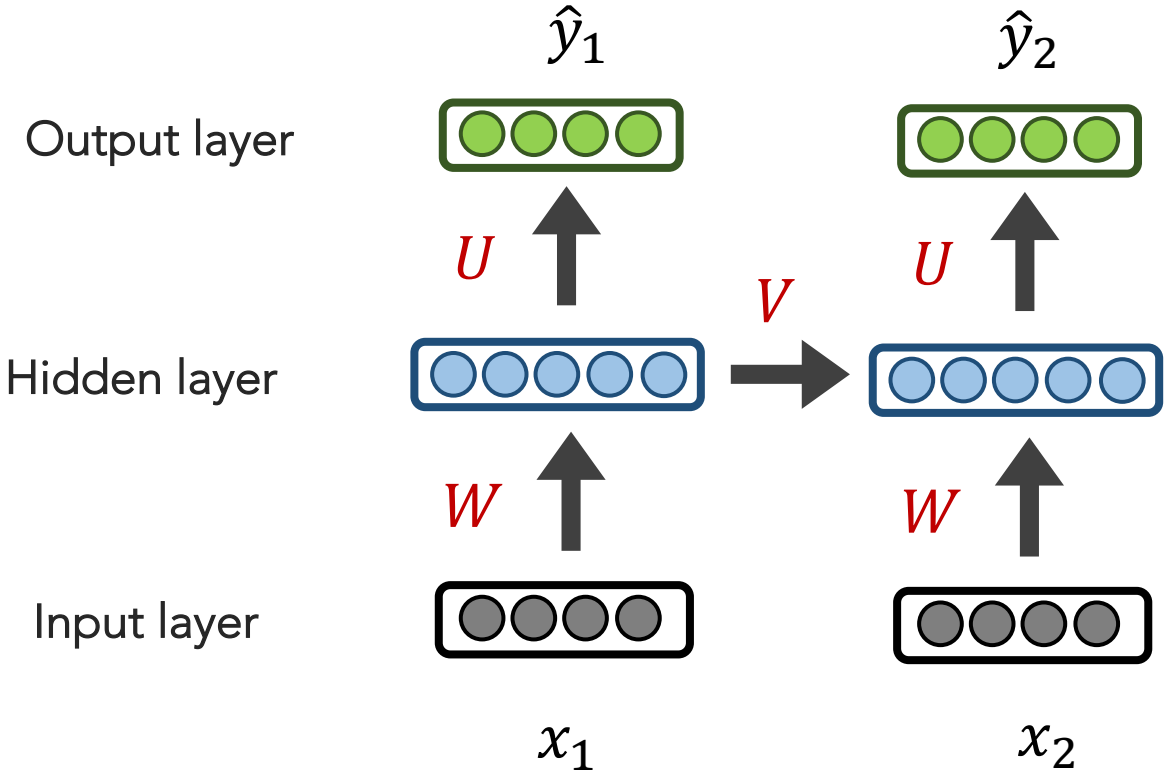
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



Language Modelling

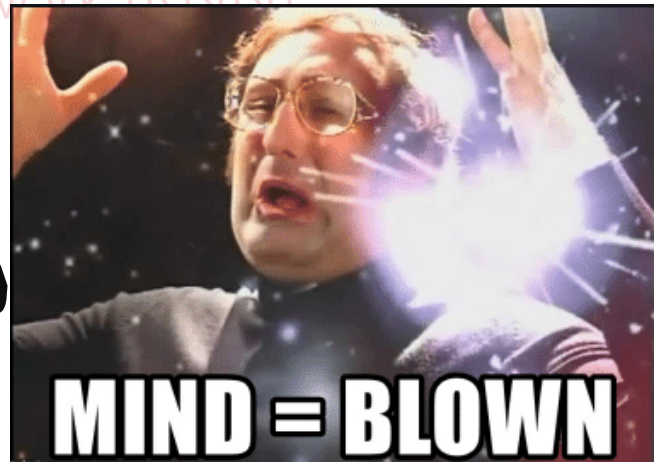
Neural Approach #2: Recurrent Neural Network (RNN)

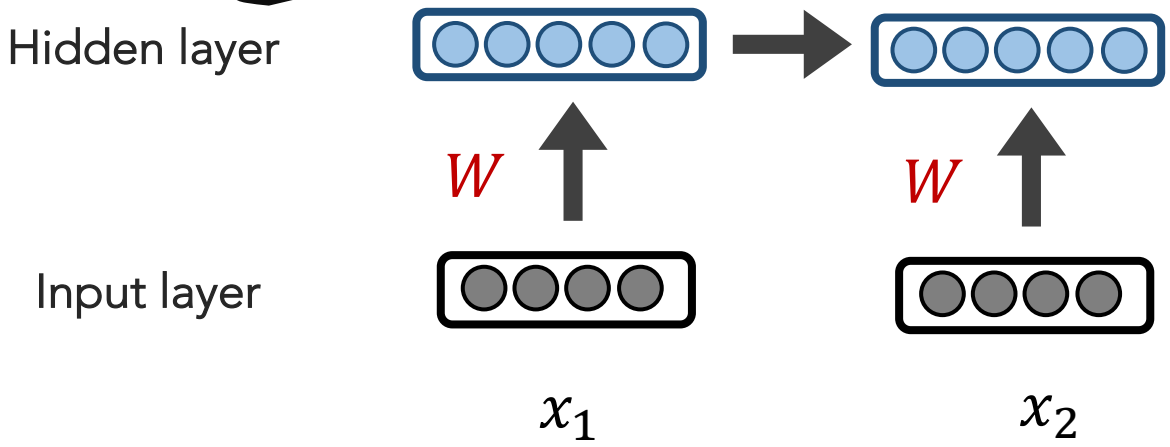


Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)

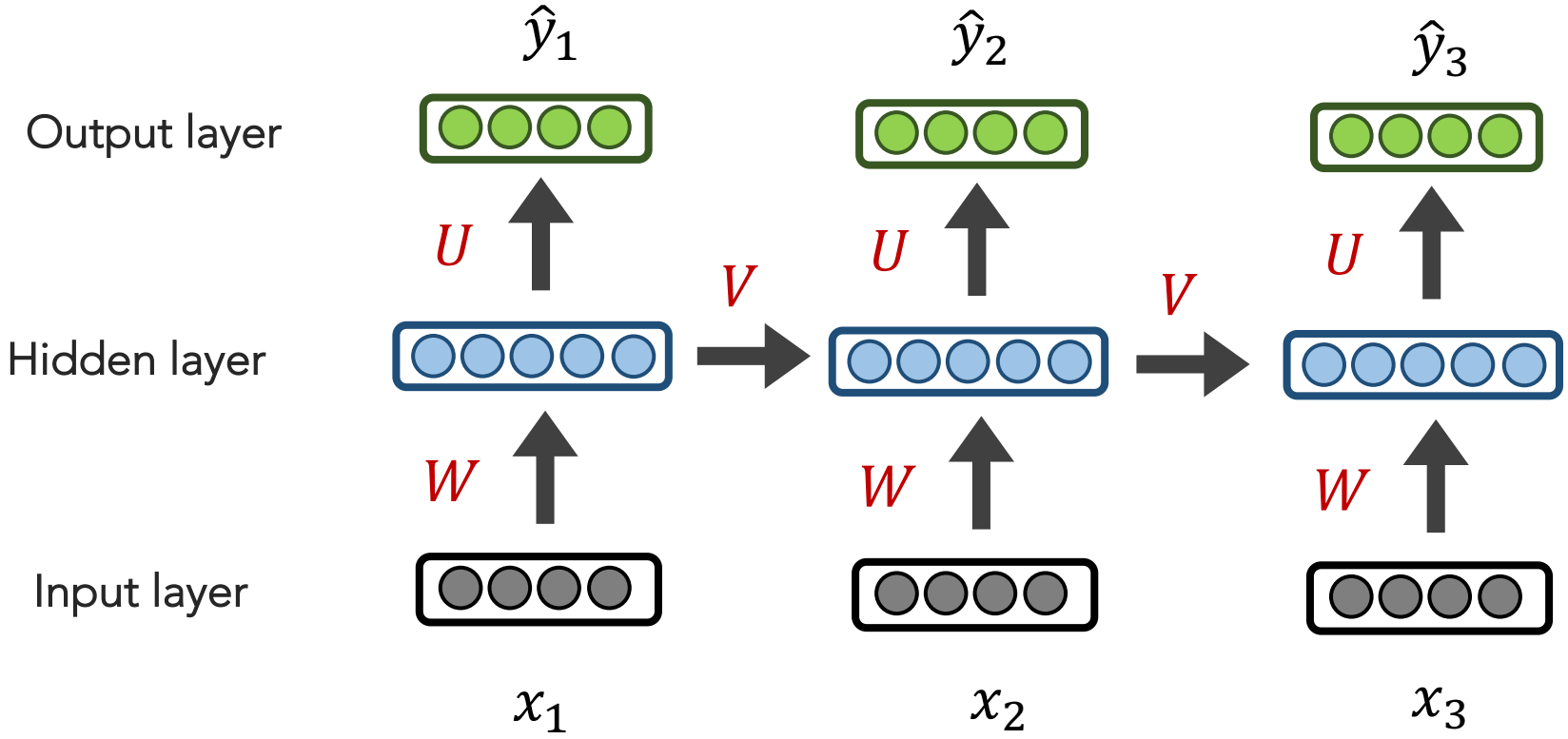
Let's use the previous hidden state, too





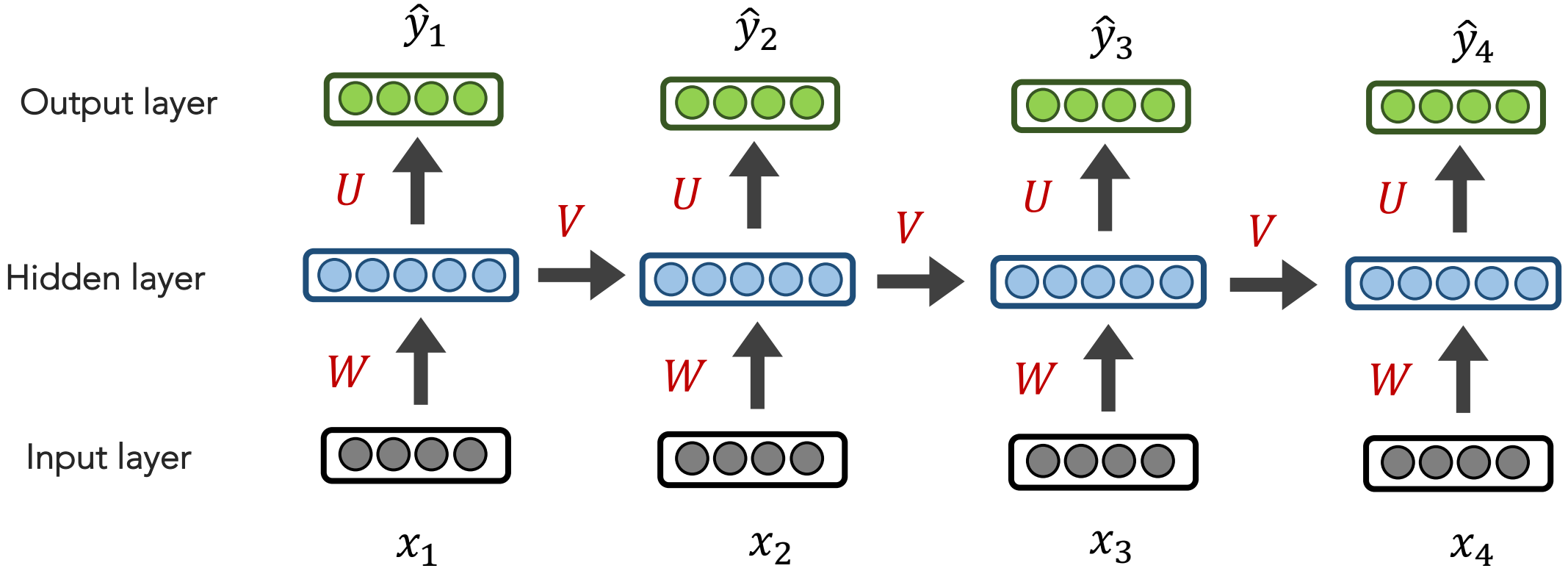
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



RNN: Generation

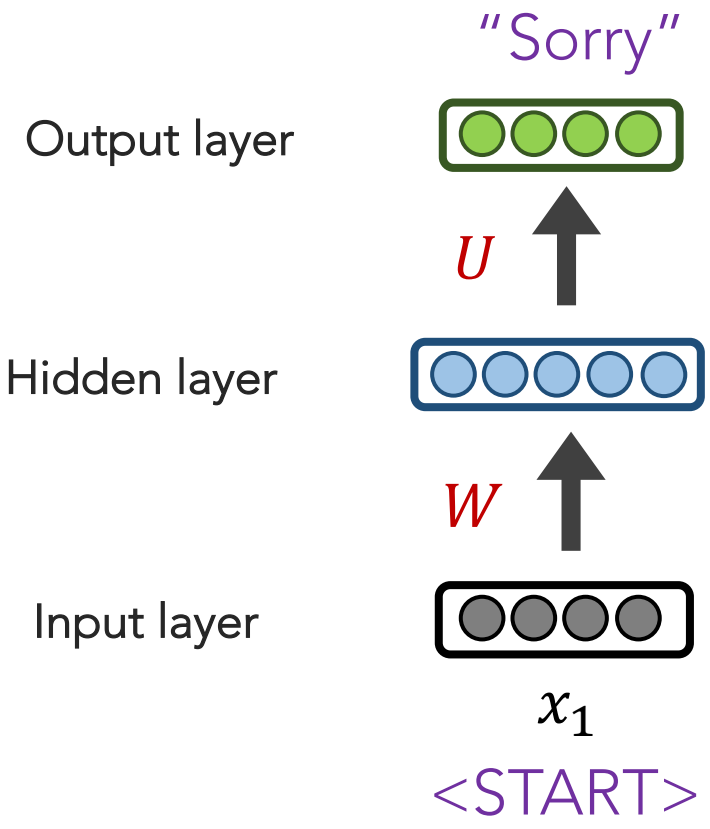
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

Continue until we generate <EOS> symbol.

RNN: Generation

We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

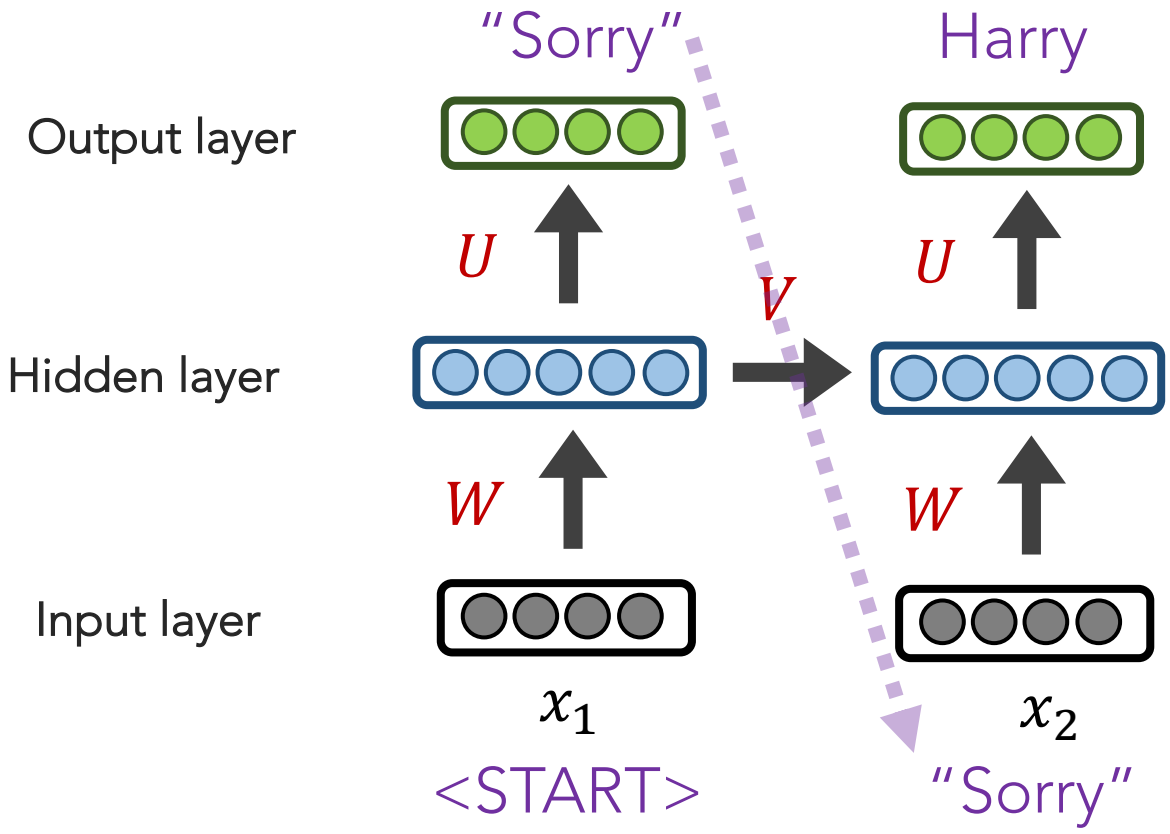
Continue until we generate <EOS> symbol.



RNN: Generation

We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

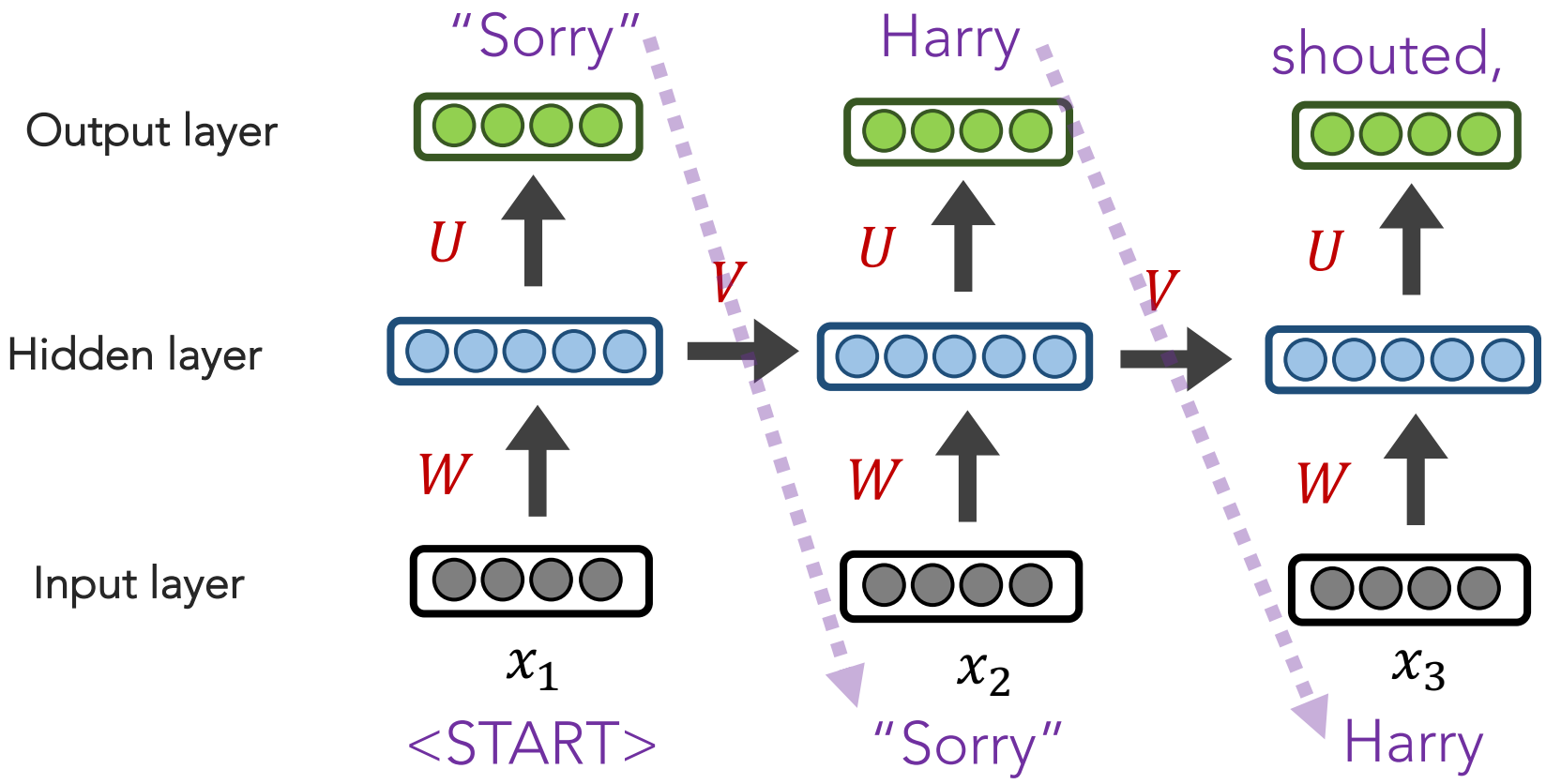
Continue until we generate <EOS> symbol.



RNN: Generation

We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

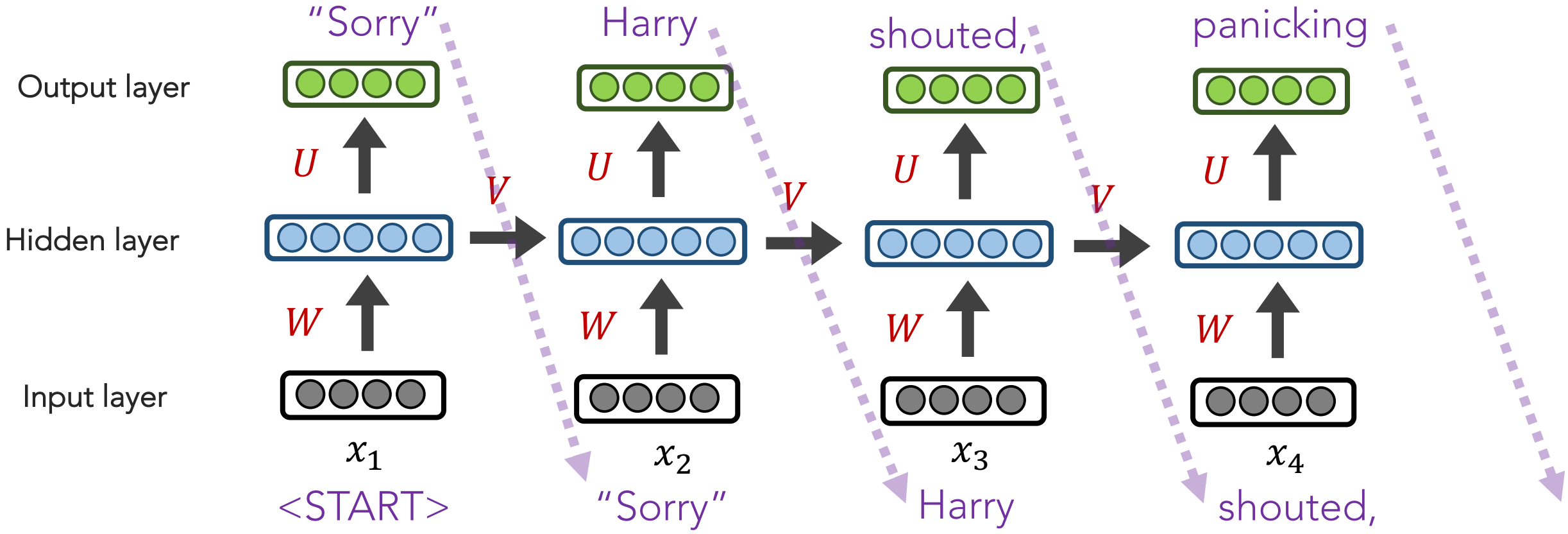
Continue until we generate <EOS> symbol.



RNN: Generation

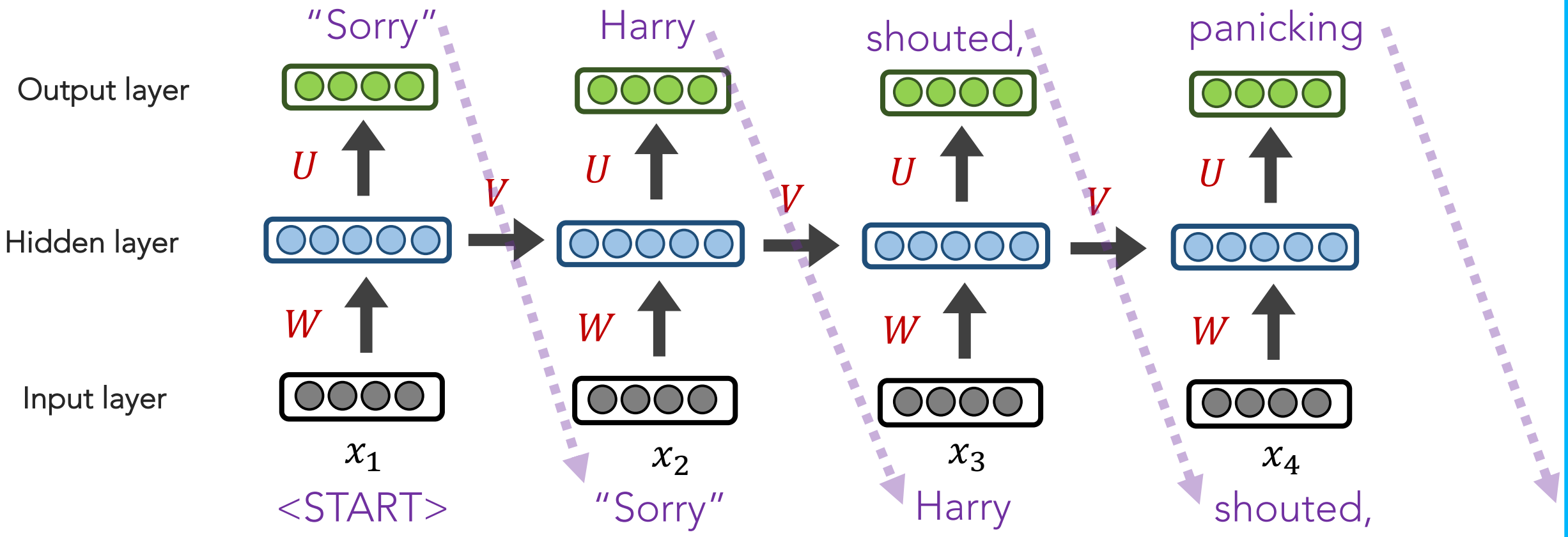
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

Continue until we generate <EOS> symbol.



RNN: Generation

NOTE: the same input (e.g., "Harry") can easily yield different outputs, depending on the context (unlike FFNNs and n-grams).



RNN: Generation

When trained on Harry Potter text, it generates:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

RNNs: Overview

RNN STRENGTHS

- Can handle infinite-length sequences (not just a fixed-window)
- Has a "memory" of the context (thanks to the hidden layer's recurrent loop)
- Same weights used for all inputs, so word order isn't wonky (like FFNN)

RNN ISSUES

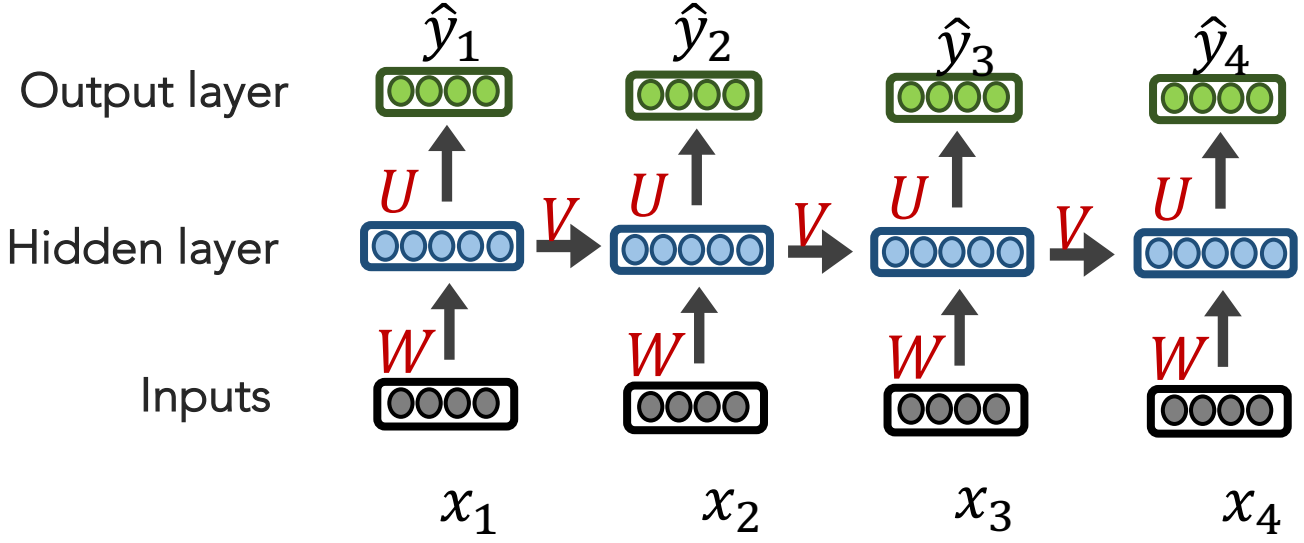
- Slow to train due needing to Backprop Through Time (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- The LSTM variant is way better at making use of long-range context

RNNs/LSTMs

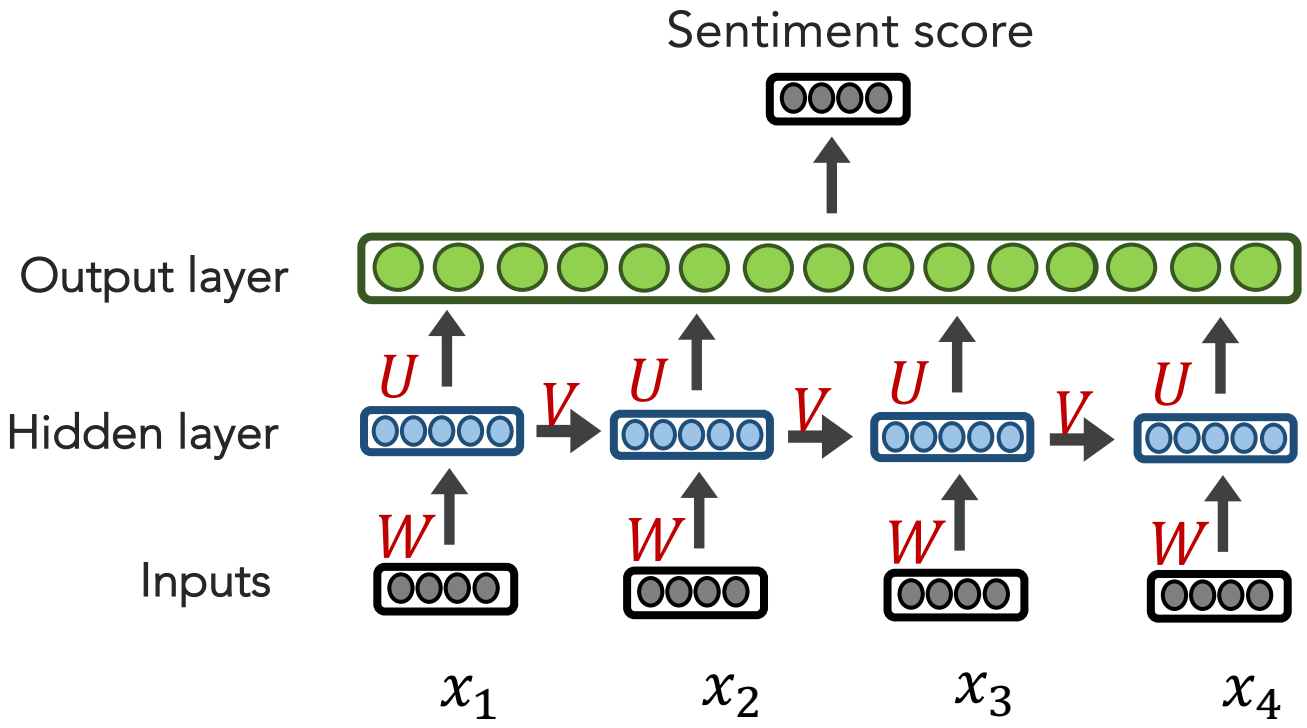
IMPORTANT

If your goal isn't to predict the next item in a sequence, you could instead make a prediction (e.g., classification or regression) for the entire sequence

RNNs/LSTMs for sequential predictions



RNNs/LSTMs for sequent-level predictions

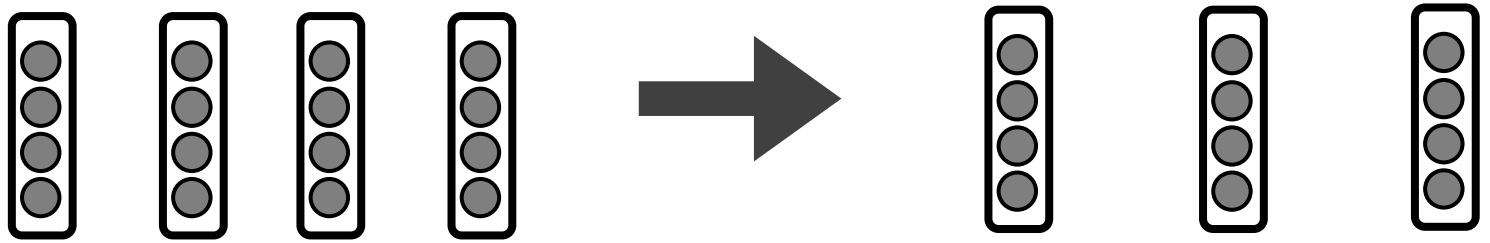


Takeaway #2:
LSTMs are a great starting point for modelling language (or other time series).

Can make:
n → n predictions or
n → 1 predictions

Language Modelling

What if we want to predict a **variable-length** output?
(e.g., n → m predictions)



Thank you for visiting!

Děkujeme za návštěvu!

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

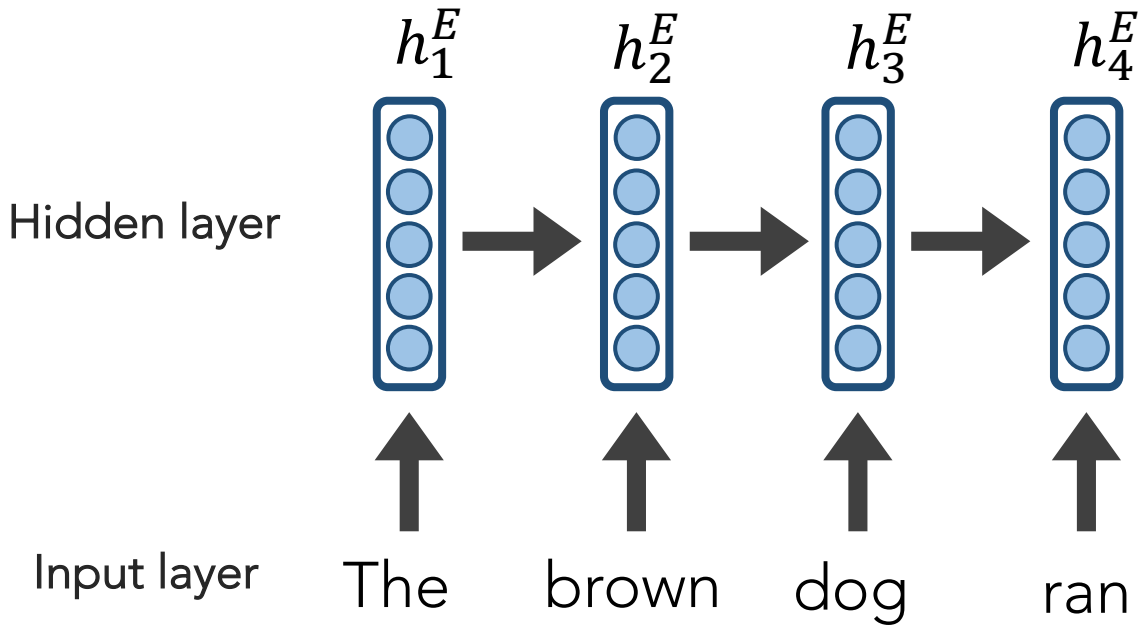
Transformers +BERT and GPT

Conclusions

Sequence-to-Sequence (seq2seq)

- If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly sub-optimal to translate word by word (like our current models are suited to do).
- Instead, let a ***sequence*** of tokens be the unit that we ultimately wish to work with (a sequence of length **N** may emit a sequences of length **M**)
- Seq2seq models are comprised of **2 RNNs**: 1 encoder, 1 decoder

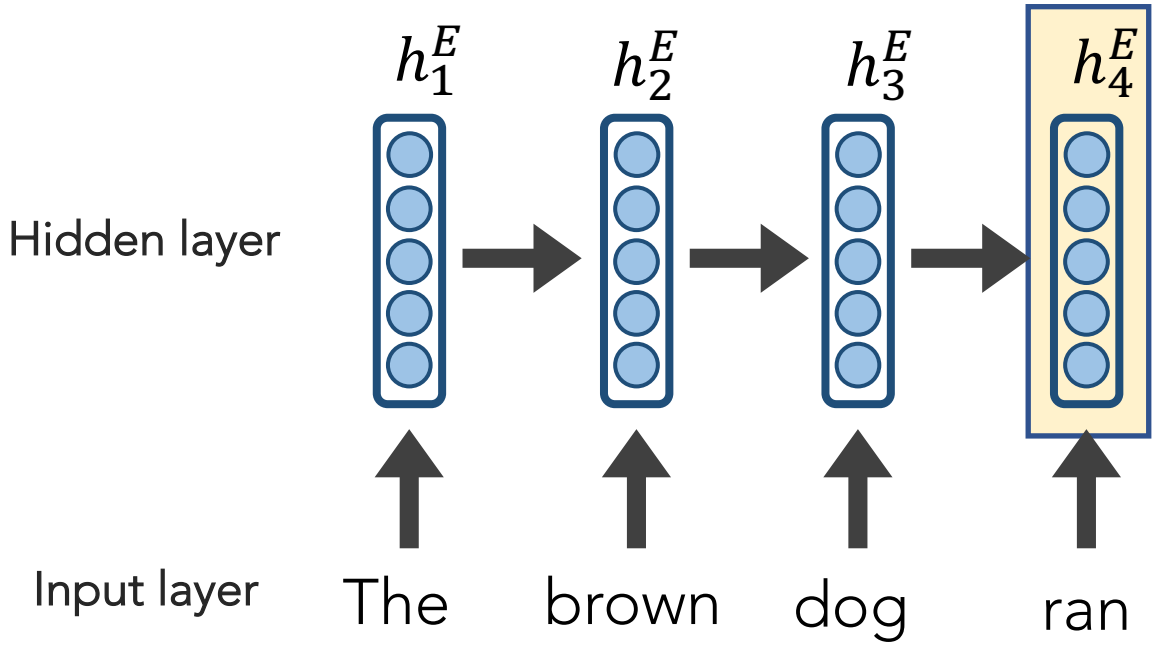
Sequence-to-Sequence (seq2seq)



ENCODER RNN

Sequence-to-Sequence (seq2seq)

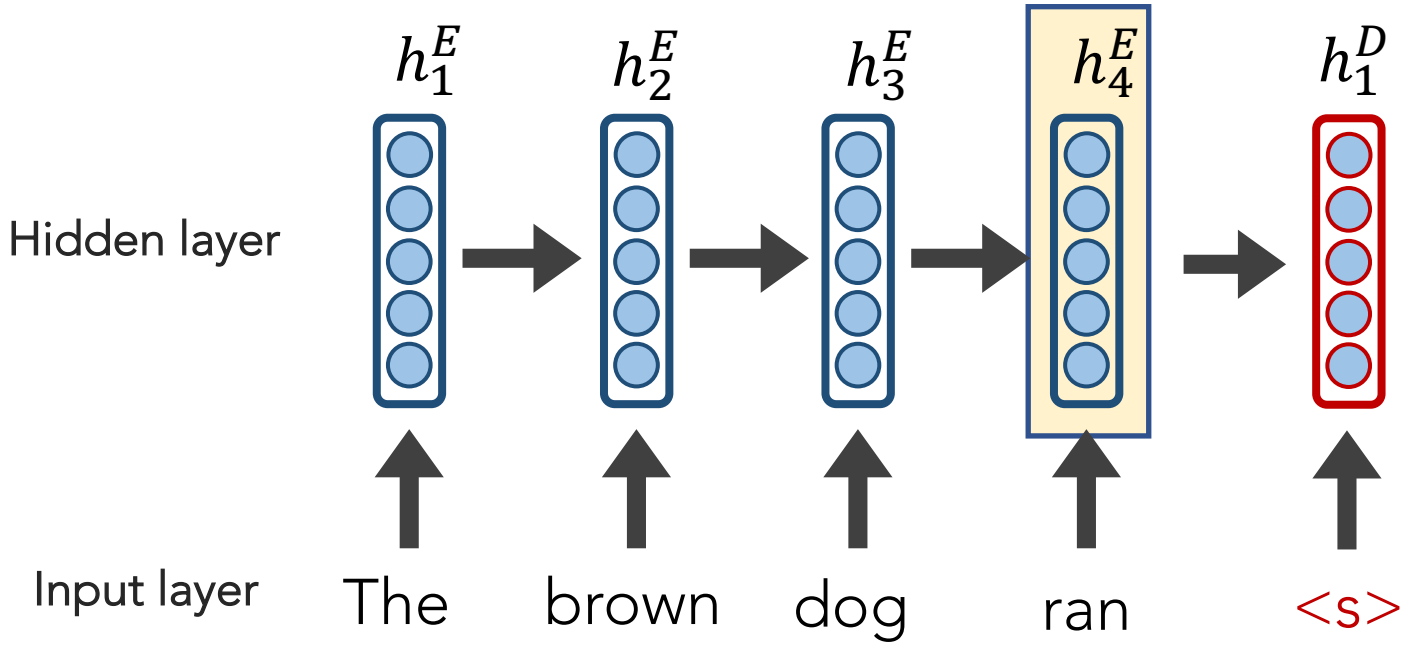
The final hidden state of the encoder RNN
is the initial state of the decoder RNN



ENCODER RNN

Sequence-to-Sequence (seq2seq)

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

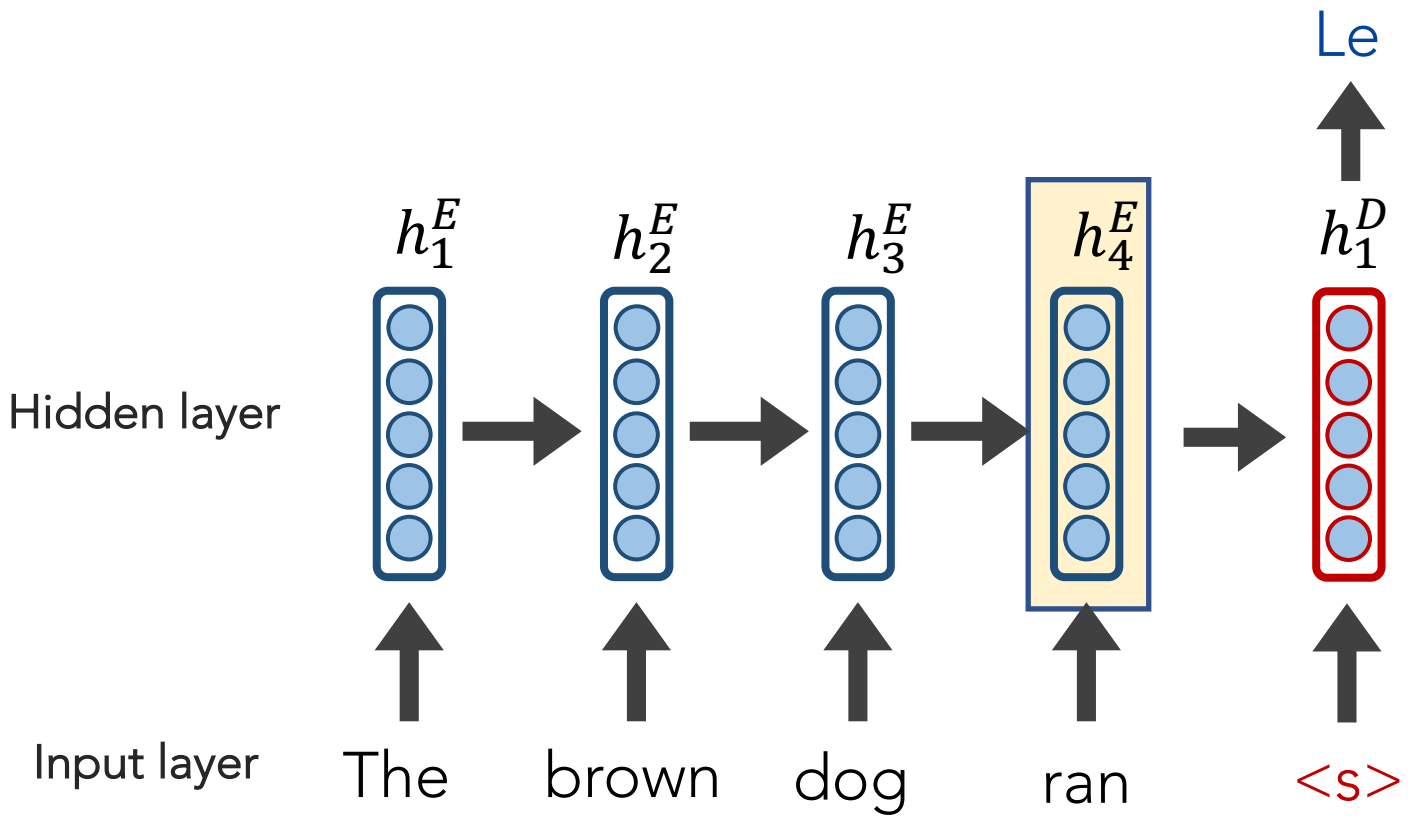


ENCODER RNN

DECODER RNN

Sequence-to-Sequence (seq2seq)

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

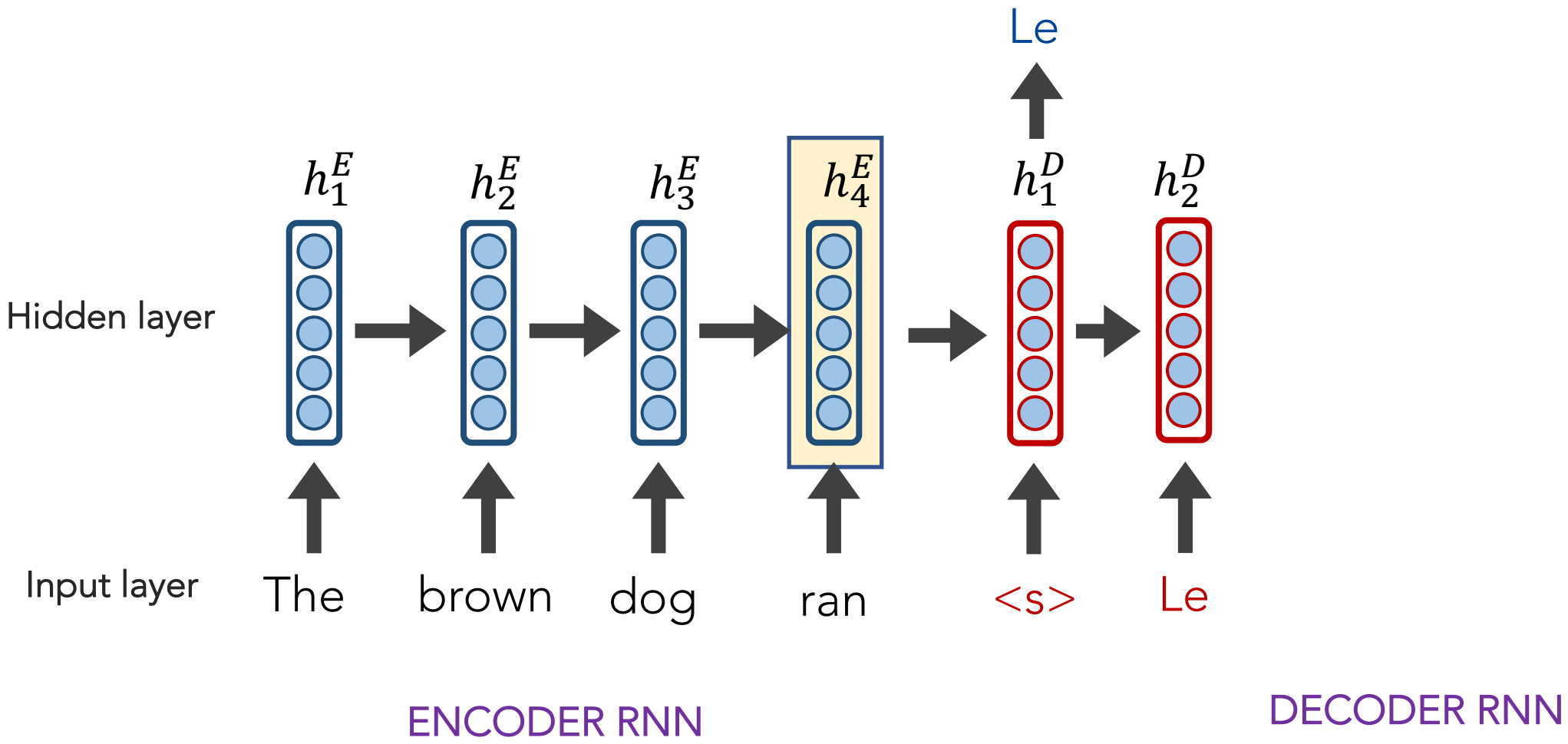


ENCODER RNN

DECODER RNN

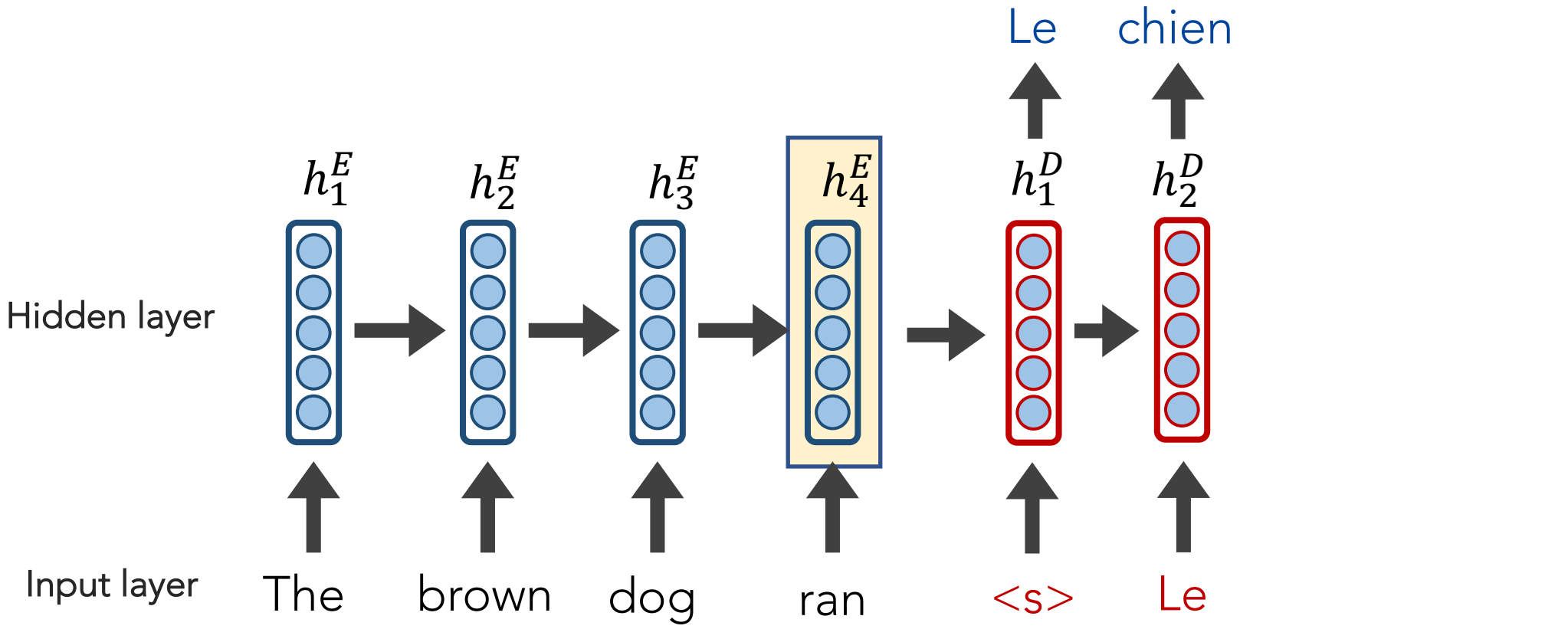
Sequence-to-Sequence (seq2seq)

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



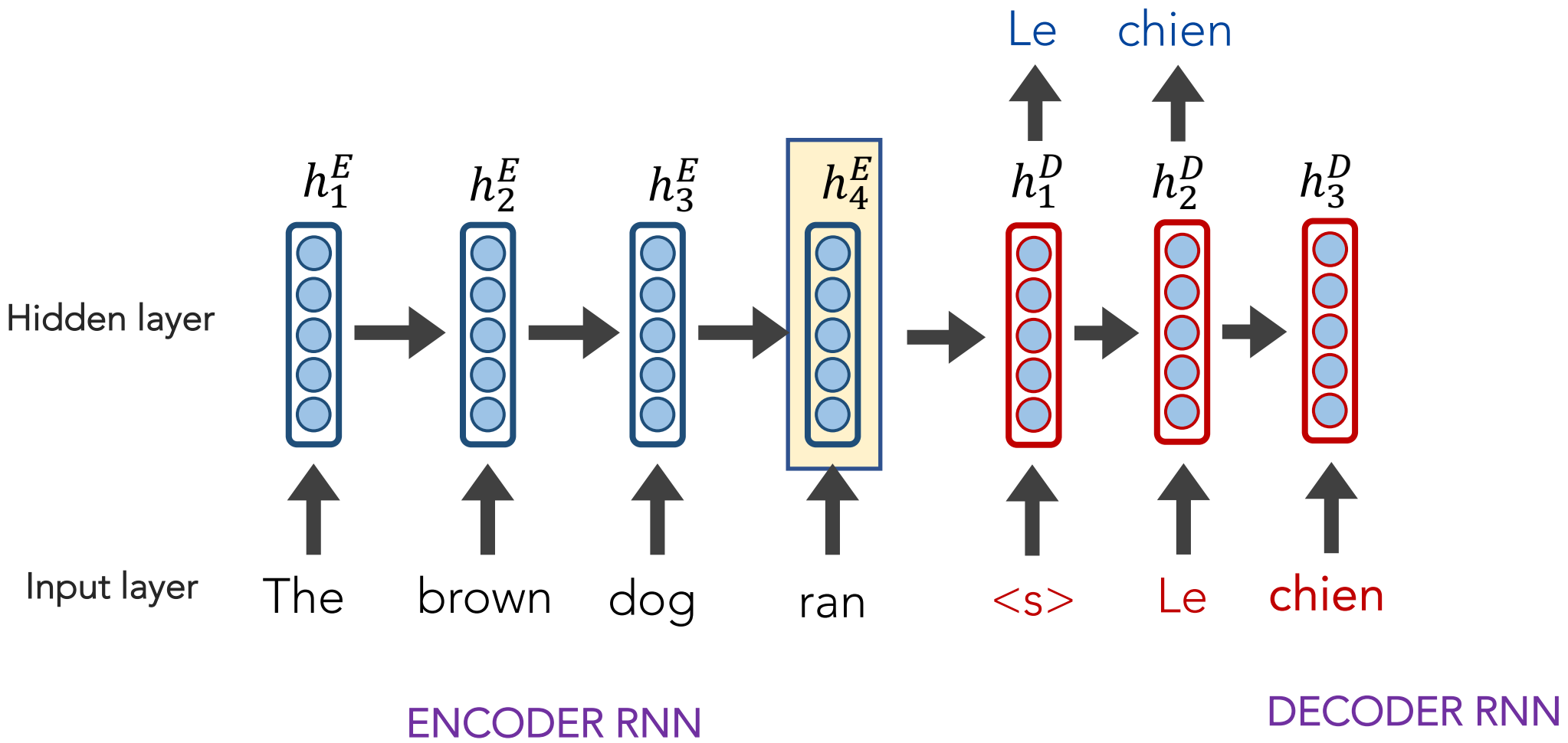
Sequence-to-Sequence (seq2seq)

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



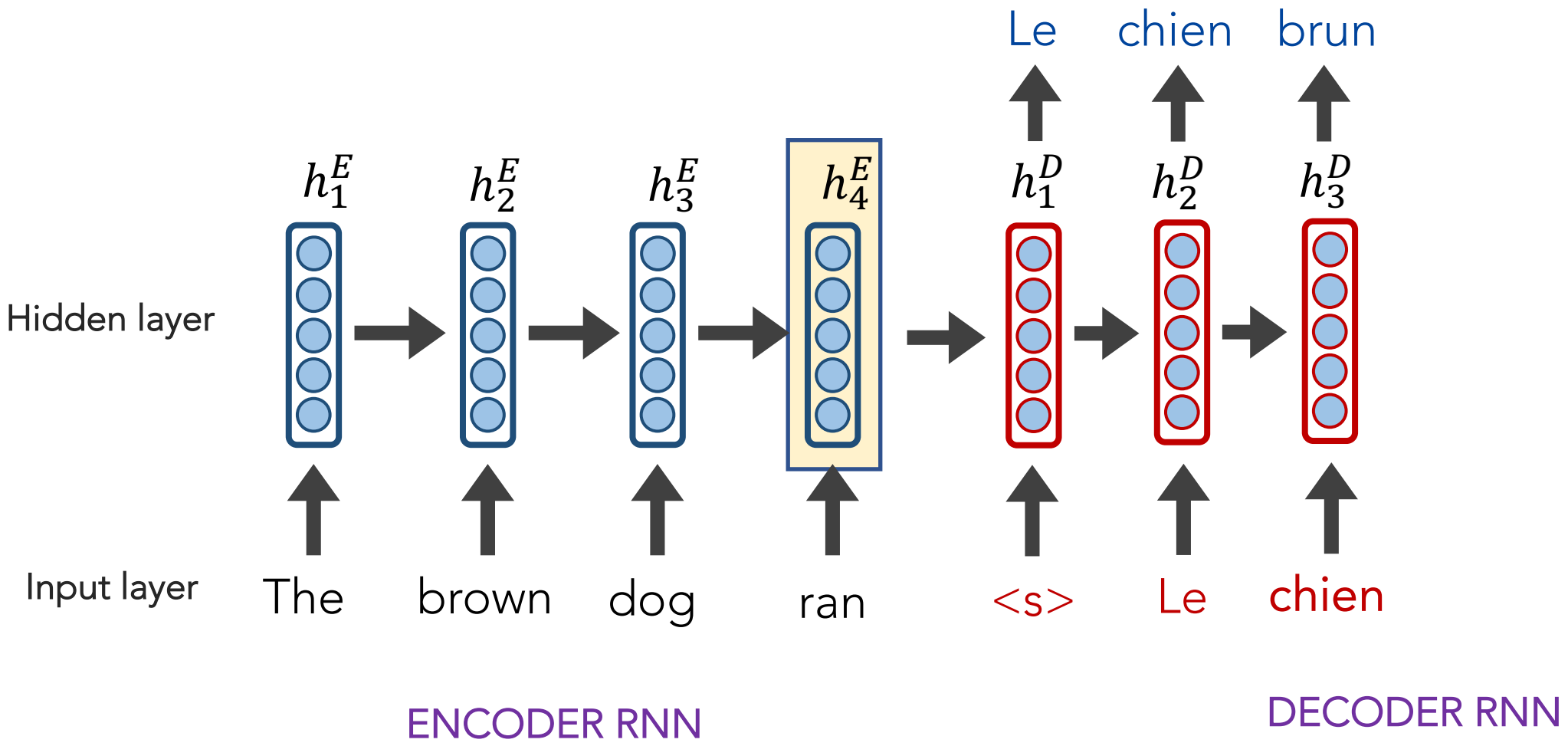
Sequence-to-Sequence (seq2seq)

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



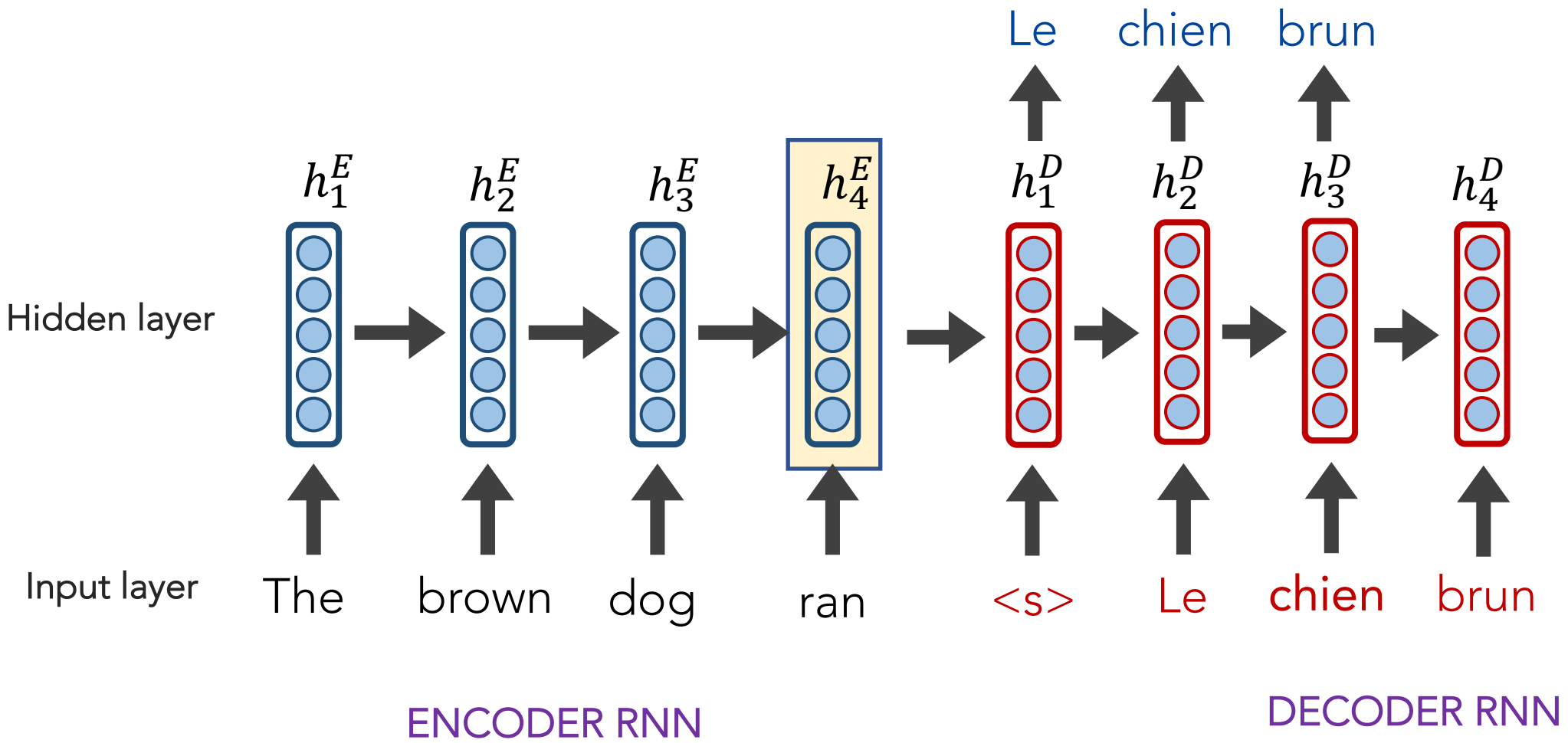
Sequence-to-Sequence (seq2seq)

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



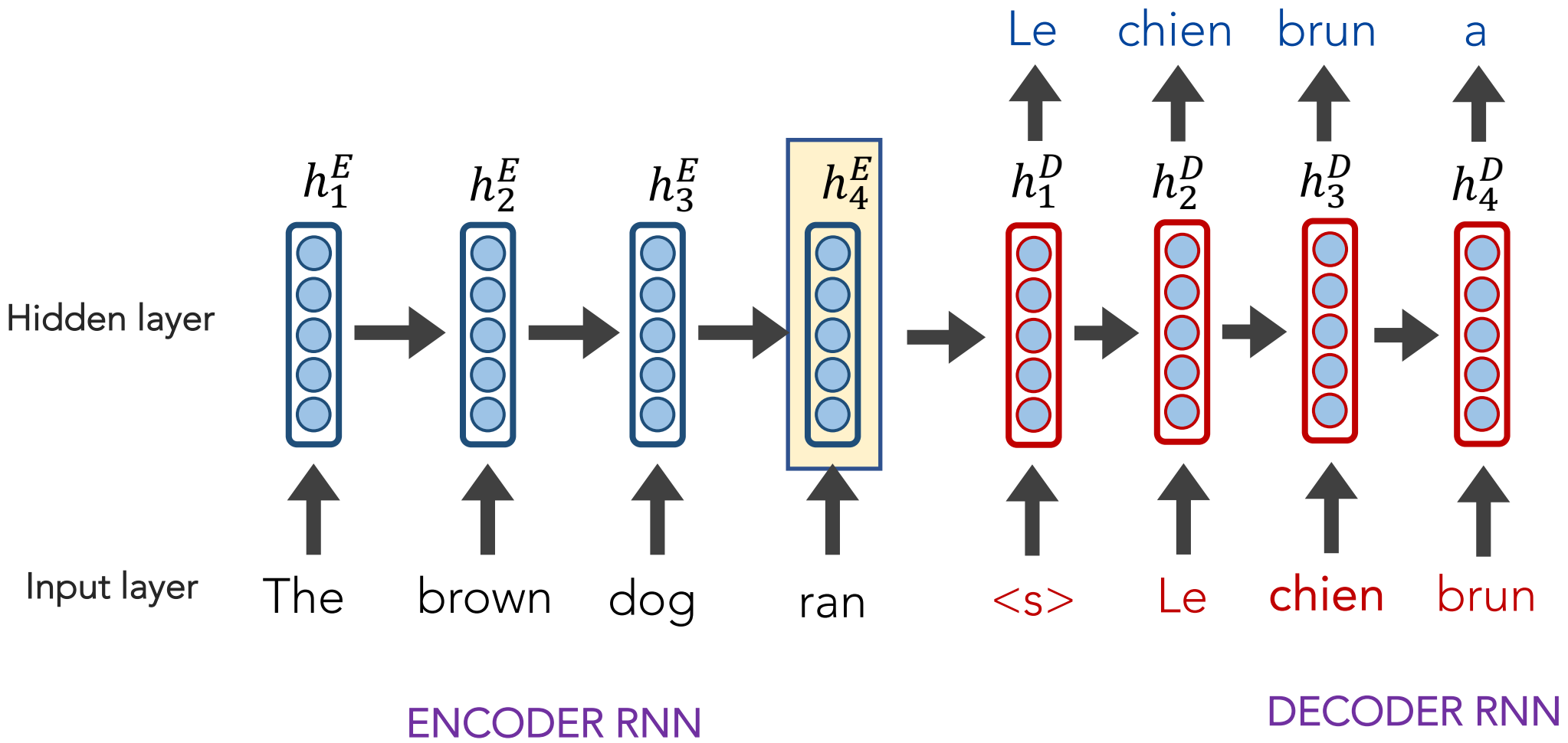
Sequence-to-Sequence (seq2seq)

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



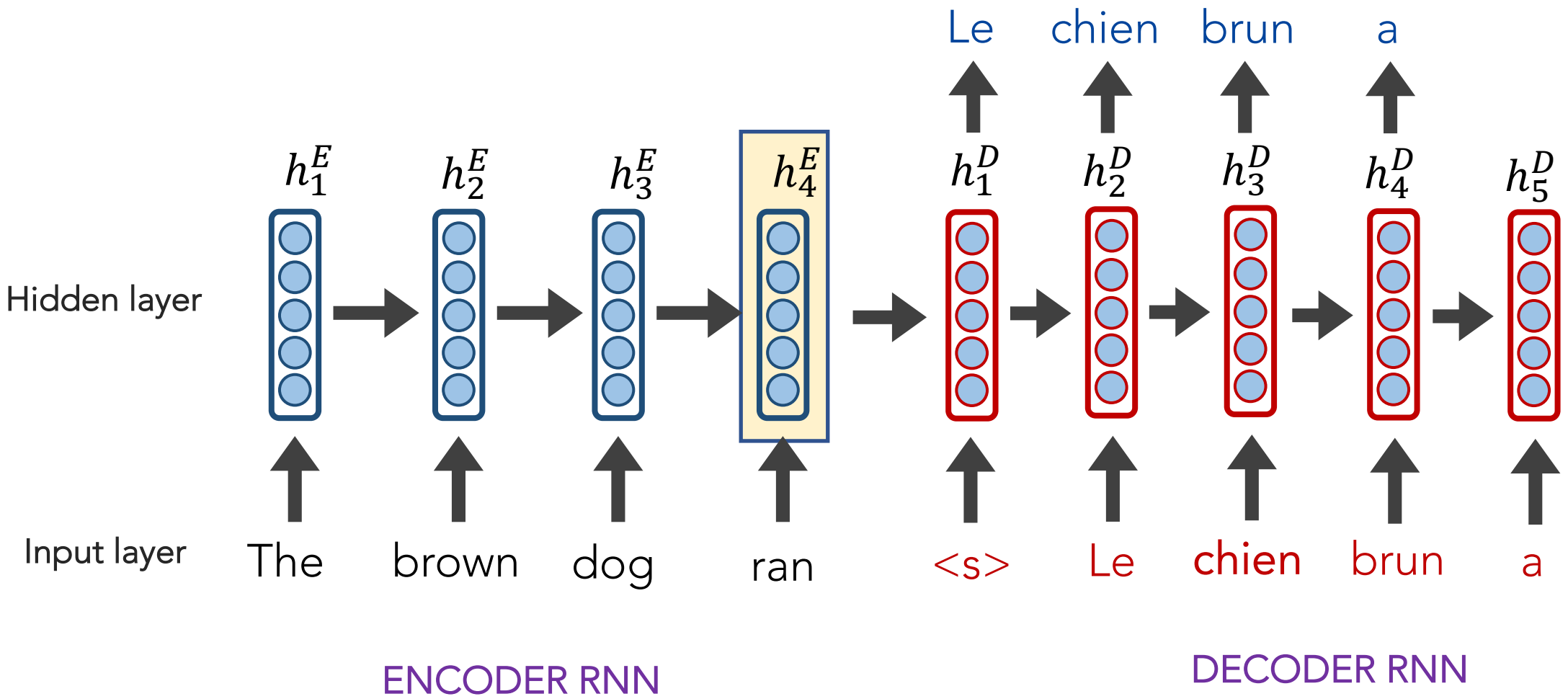
Sequence-to-Sequence (seq2seq)

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



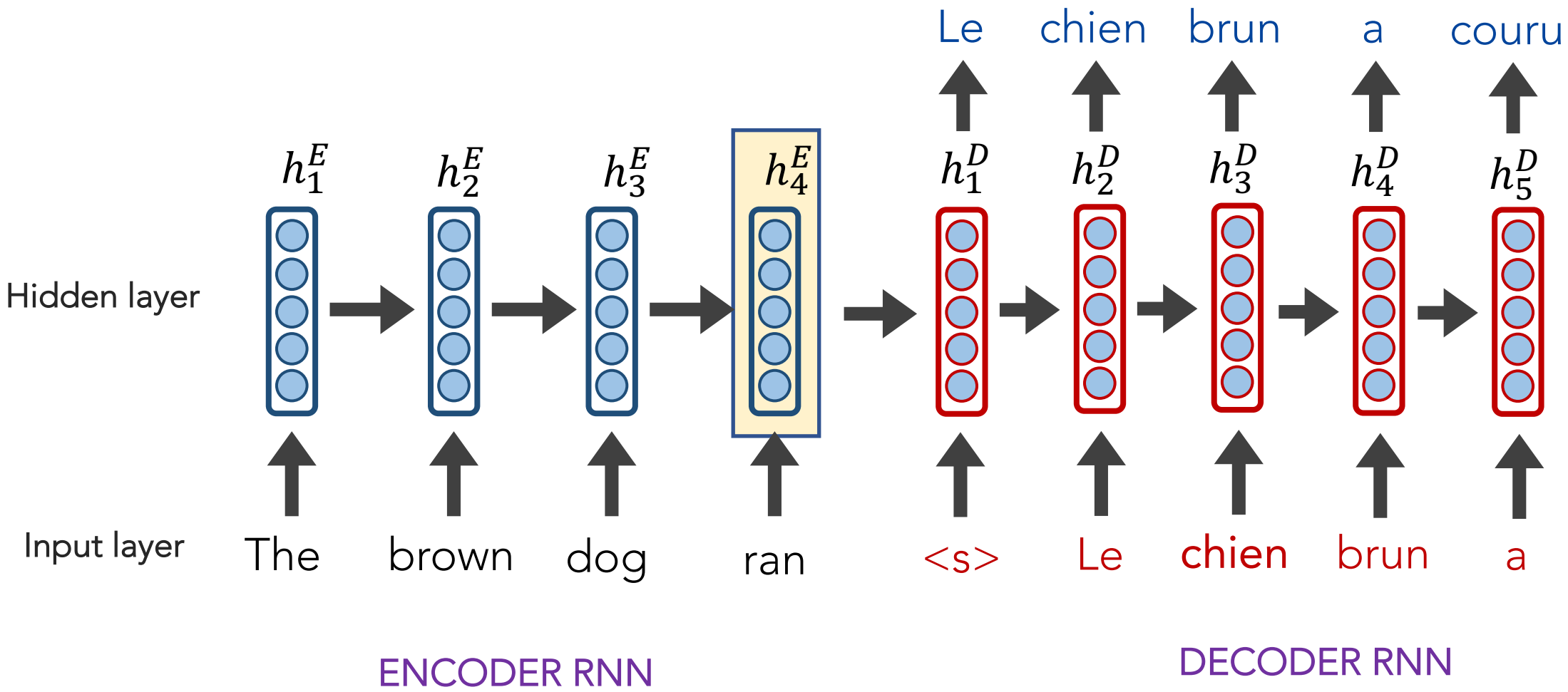
Sequence-to-Sequence (seq2seq)

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

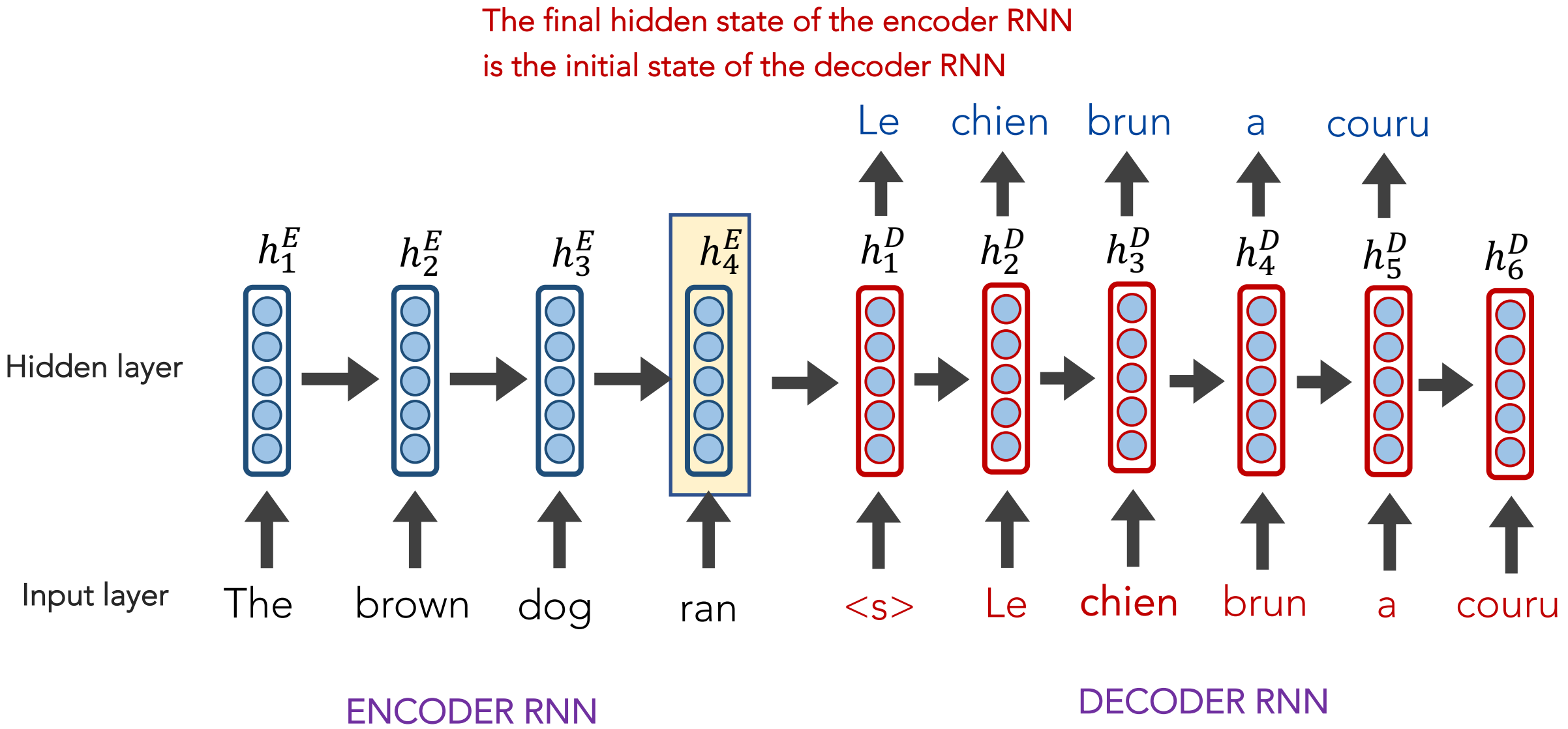


Sequence-to-Sequence (seq2seq)

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

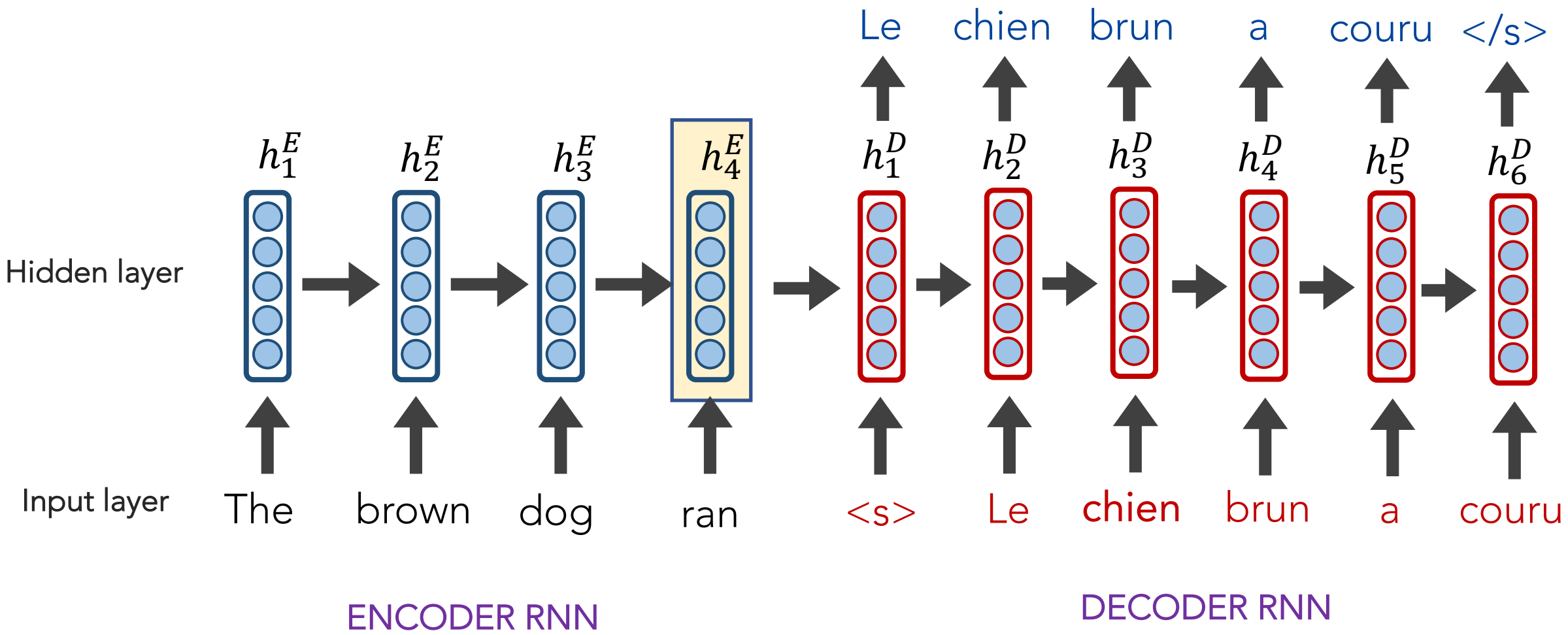


Sequence-to-Sequence (seq2seq)

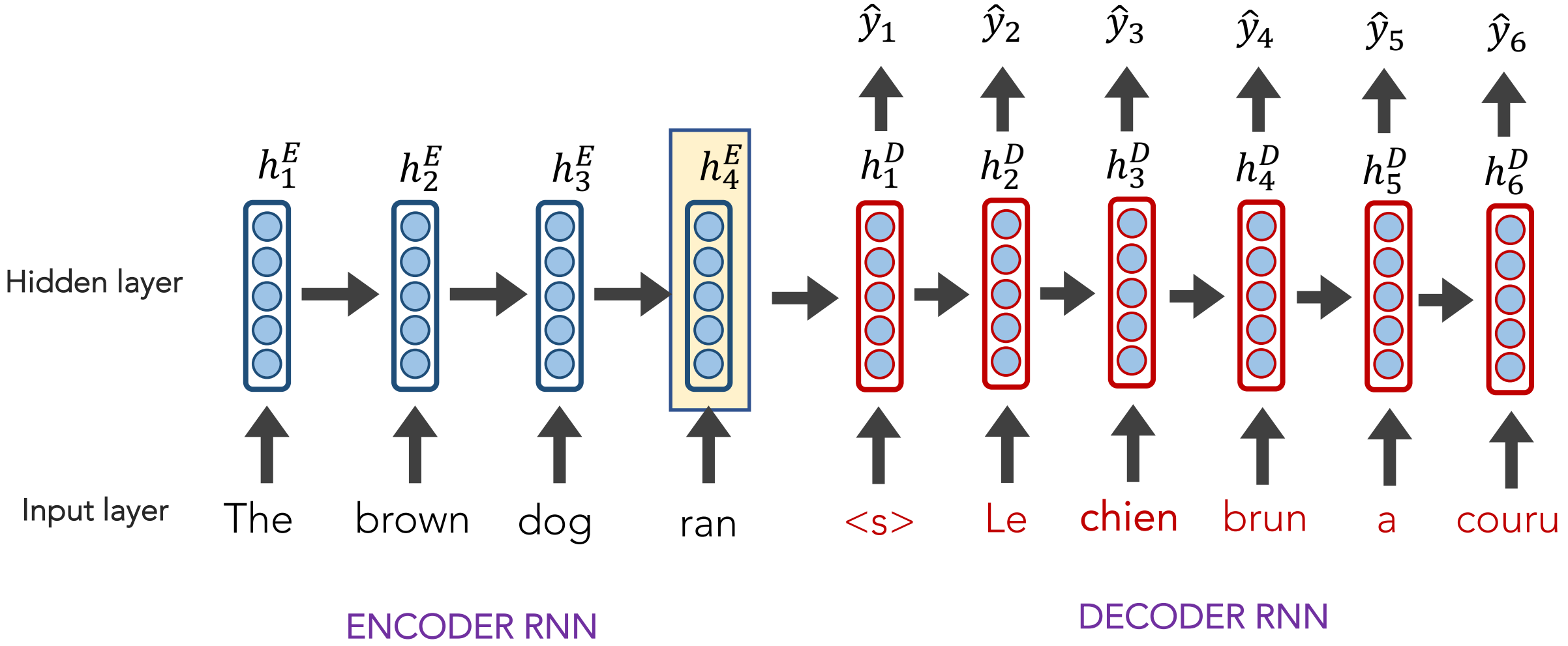


Sequence-to-Sequence (seq2seq)

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

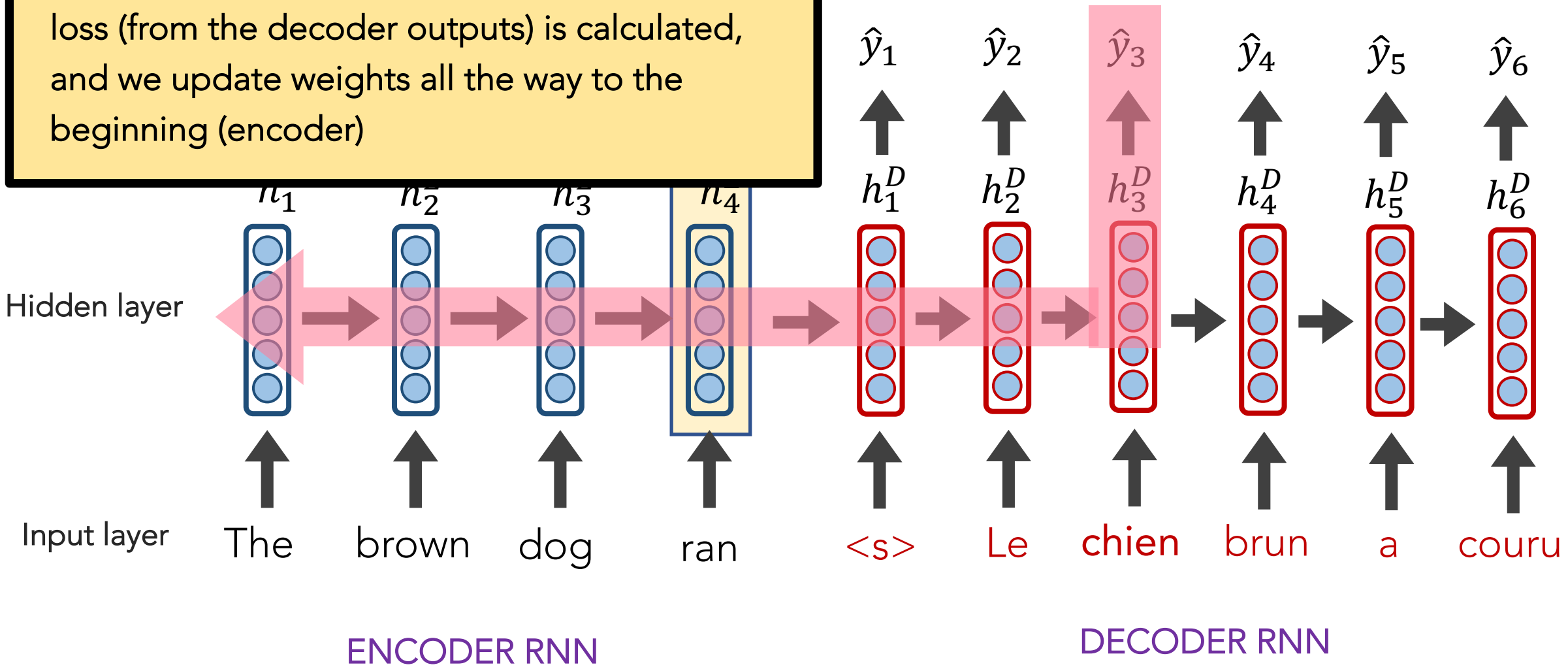


Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

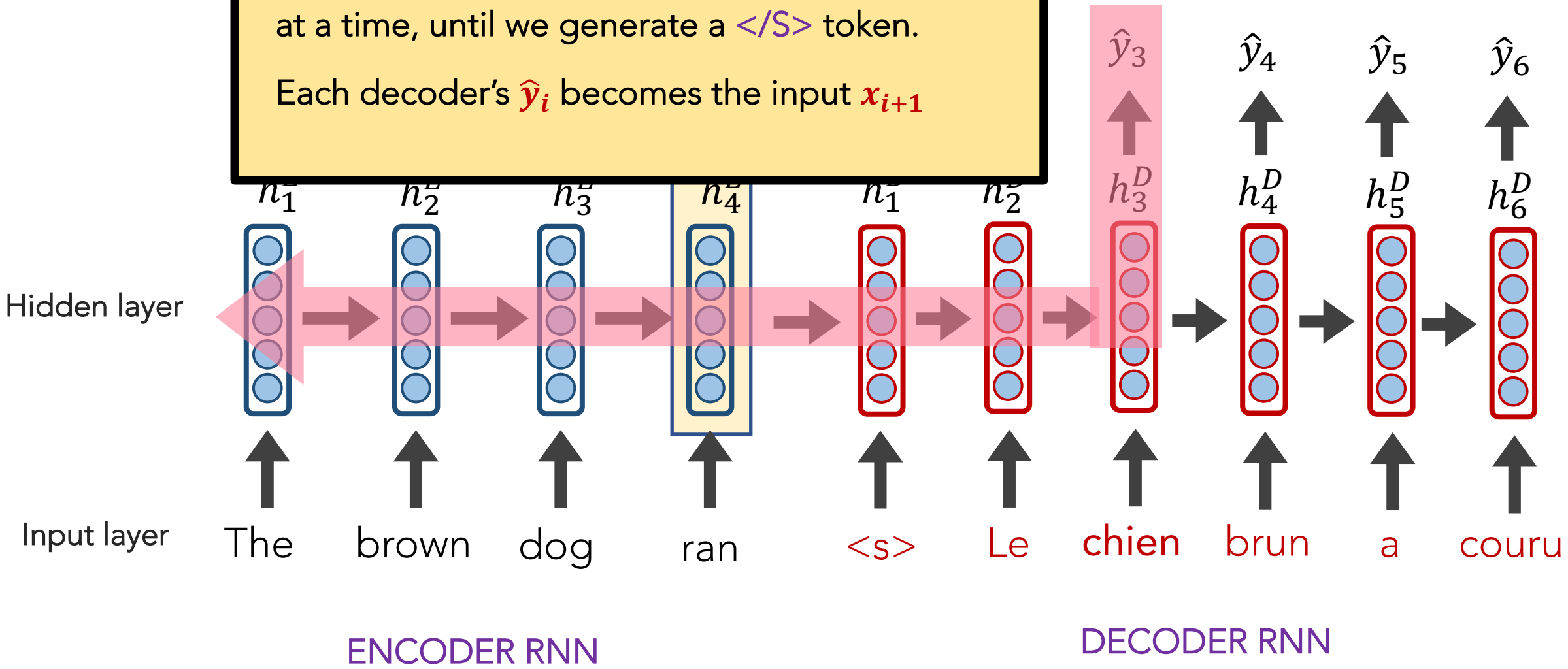
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

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

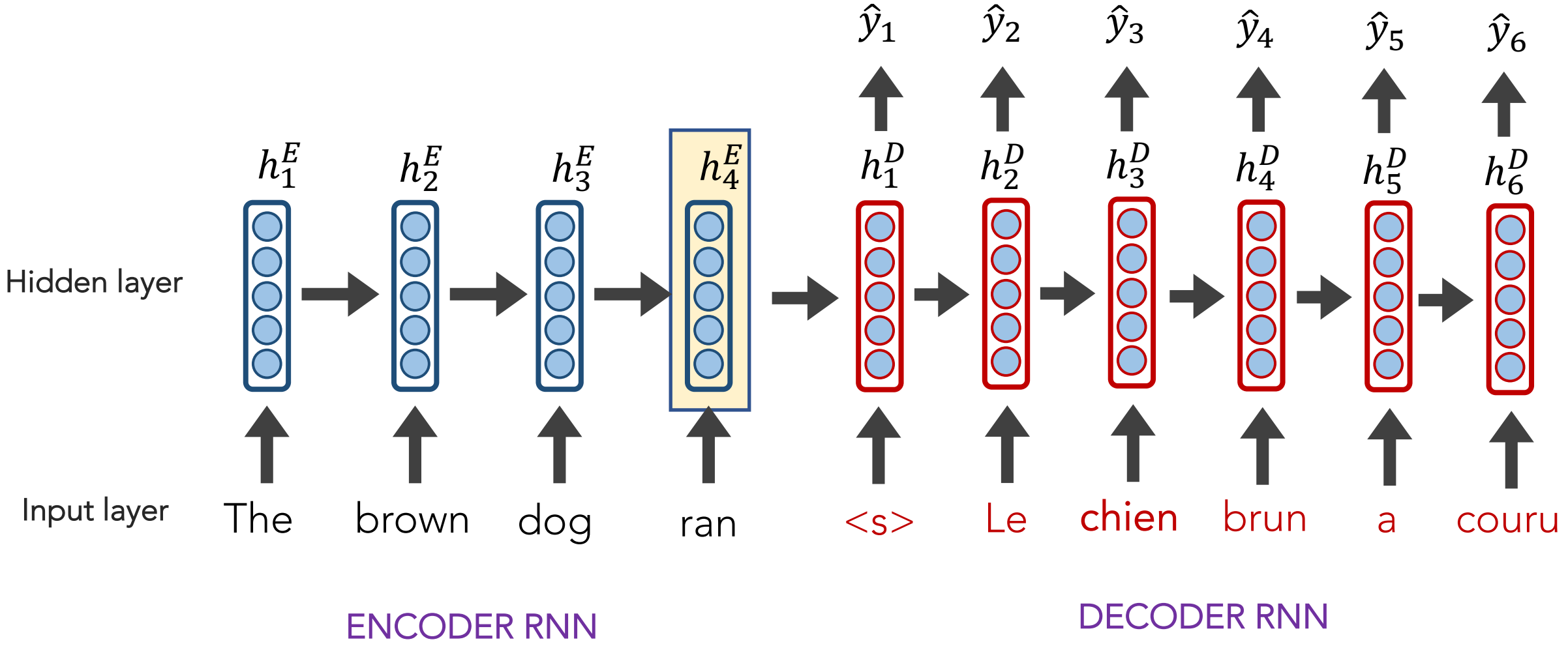
Each decoder's $\hat{y}_i$ becomes the input x_{i+1}



Sequence-to-Sequence (seq2seq)

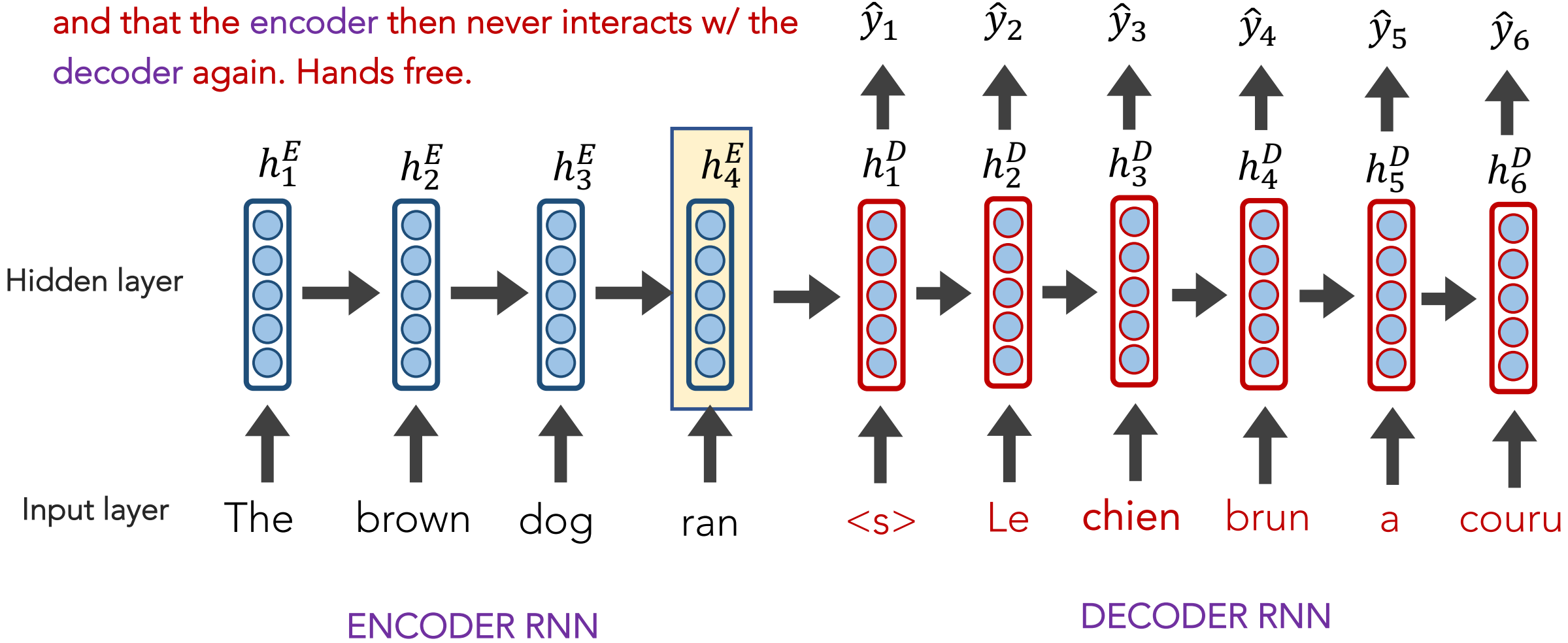
See any issues with this traditional **seq2seq** paradigm?

Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

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



Sequence-to-Sequence (seq2seq)

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

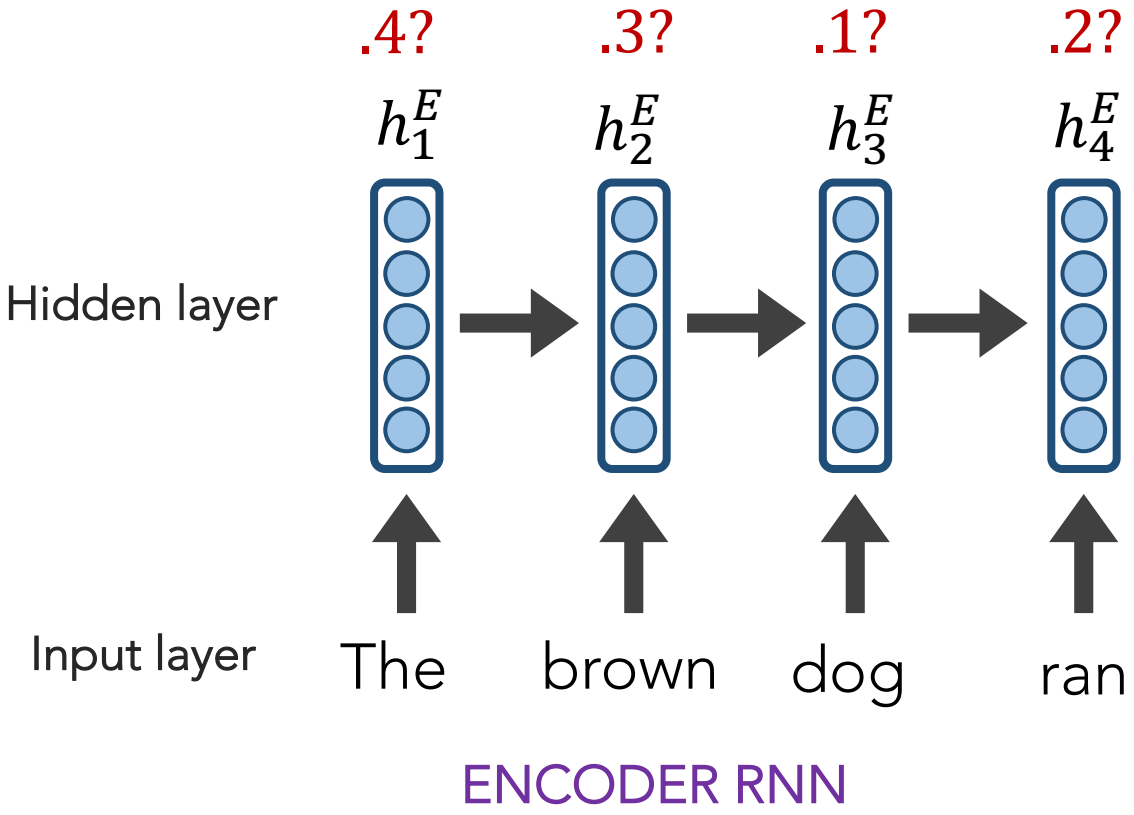
Sequence-to-Sequence (seq2seq)

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

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

seq2seq + Attention

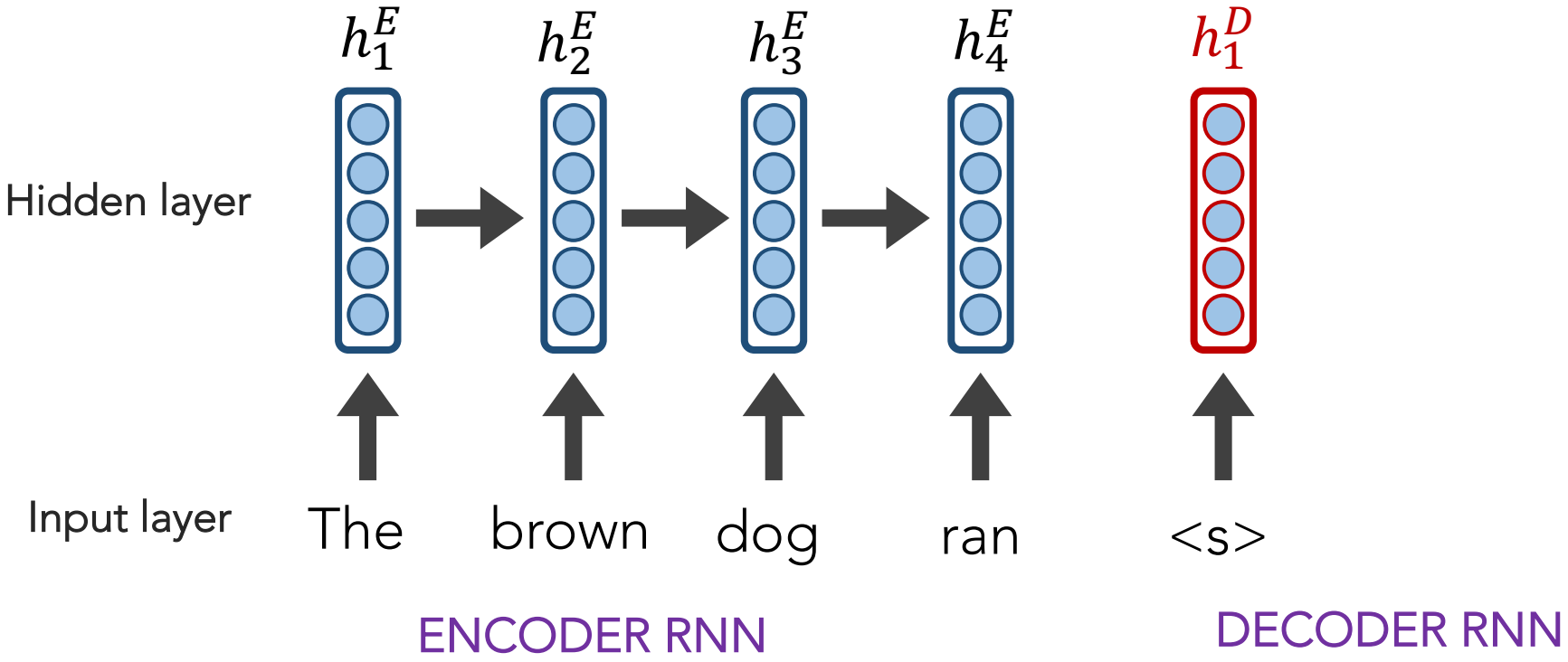
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



seq2seq + Attention

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

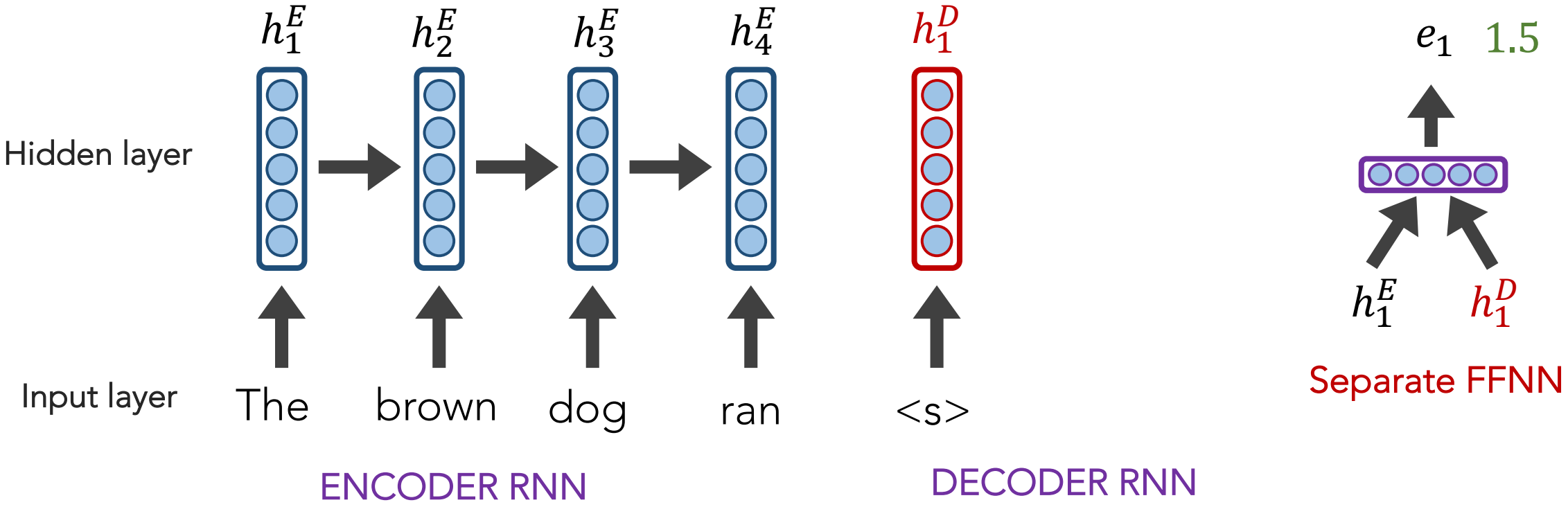
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

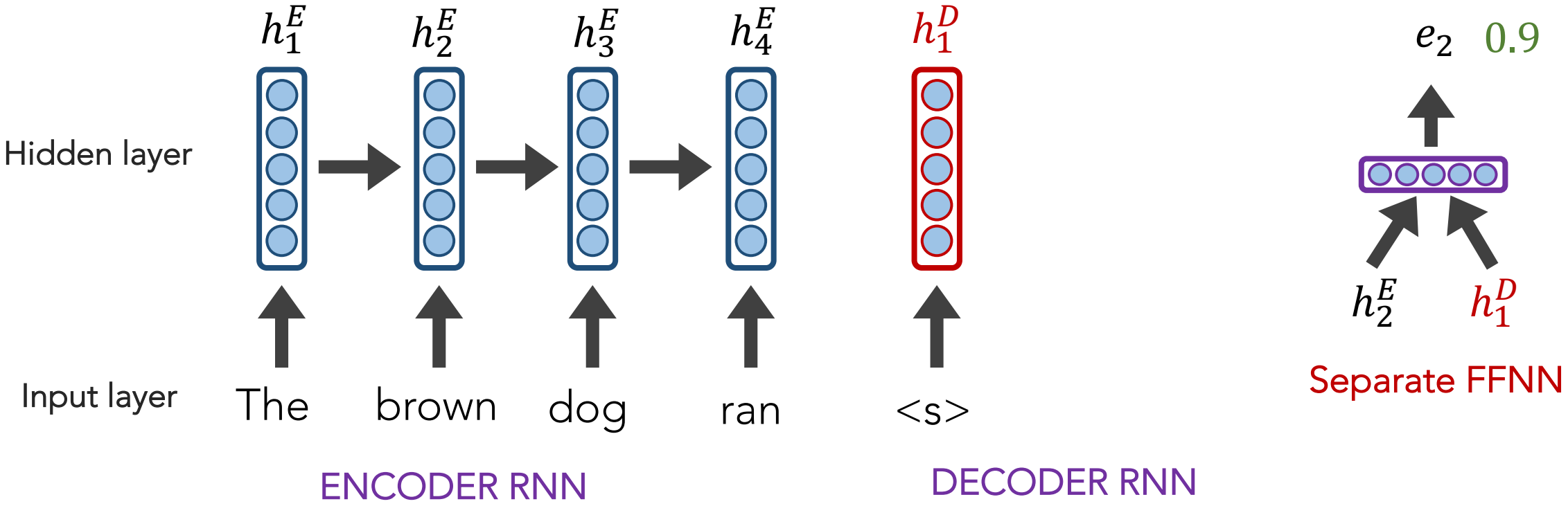
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

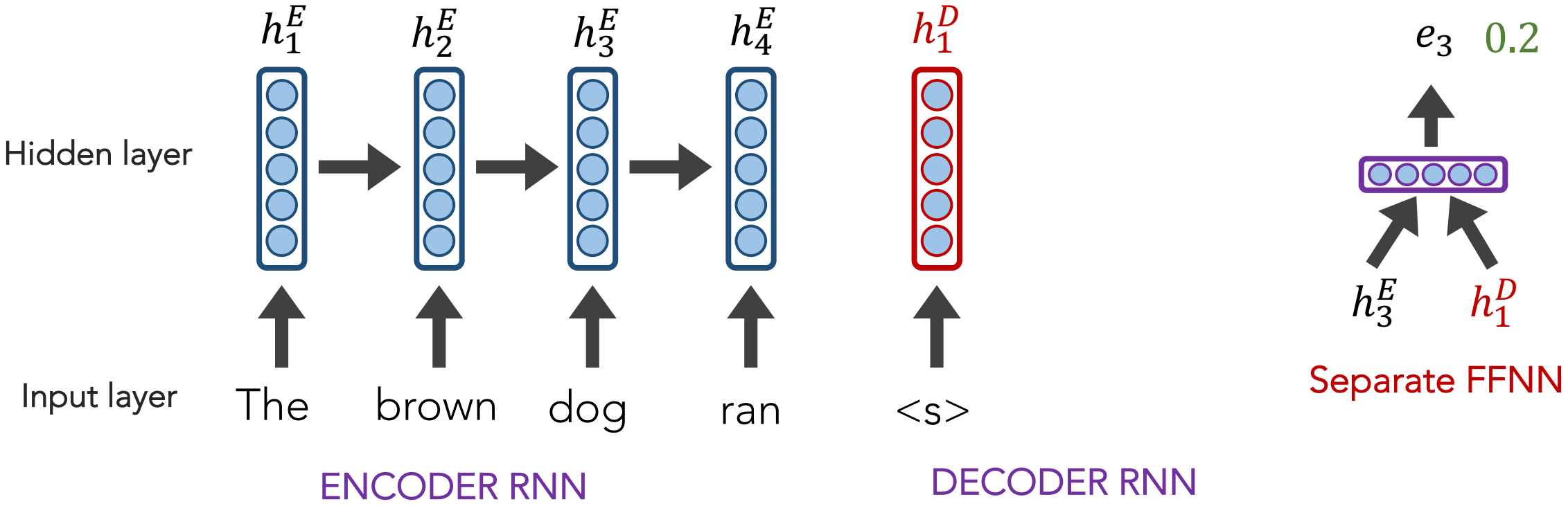
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

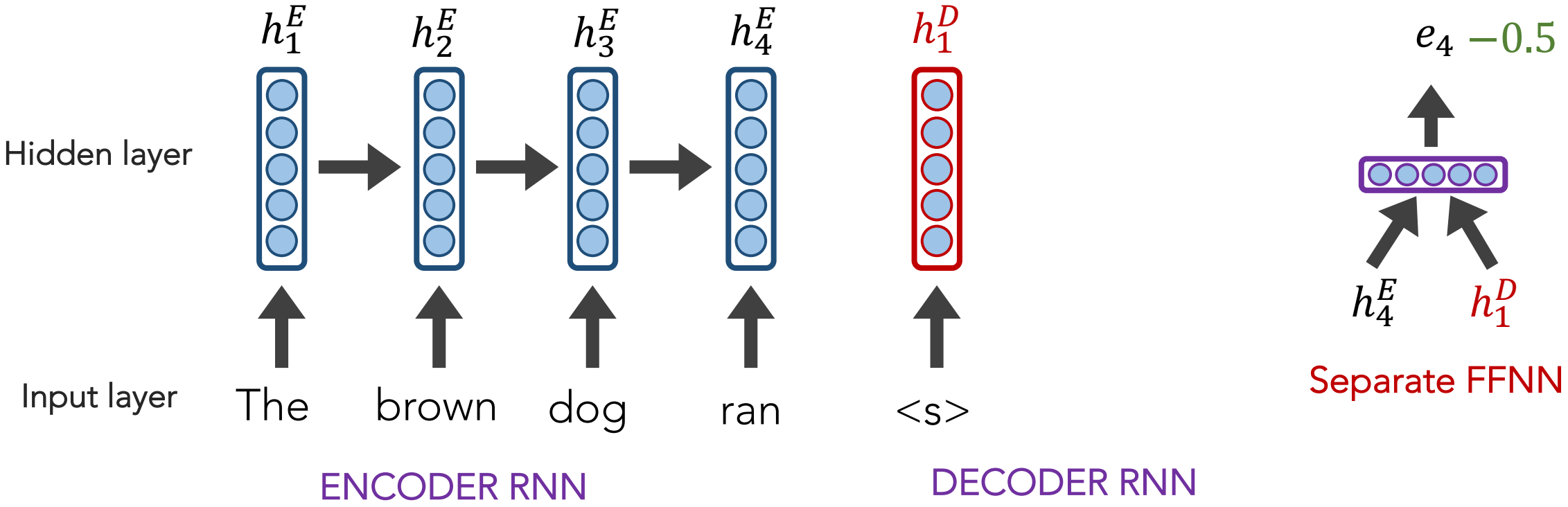
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

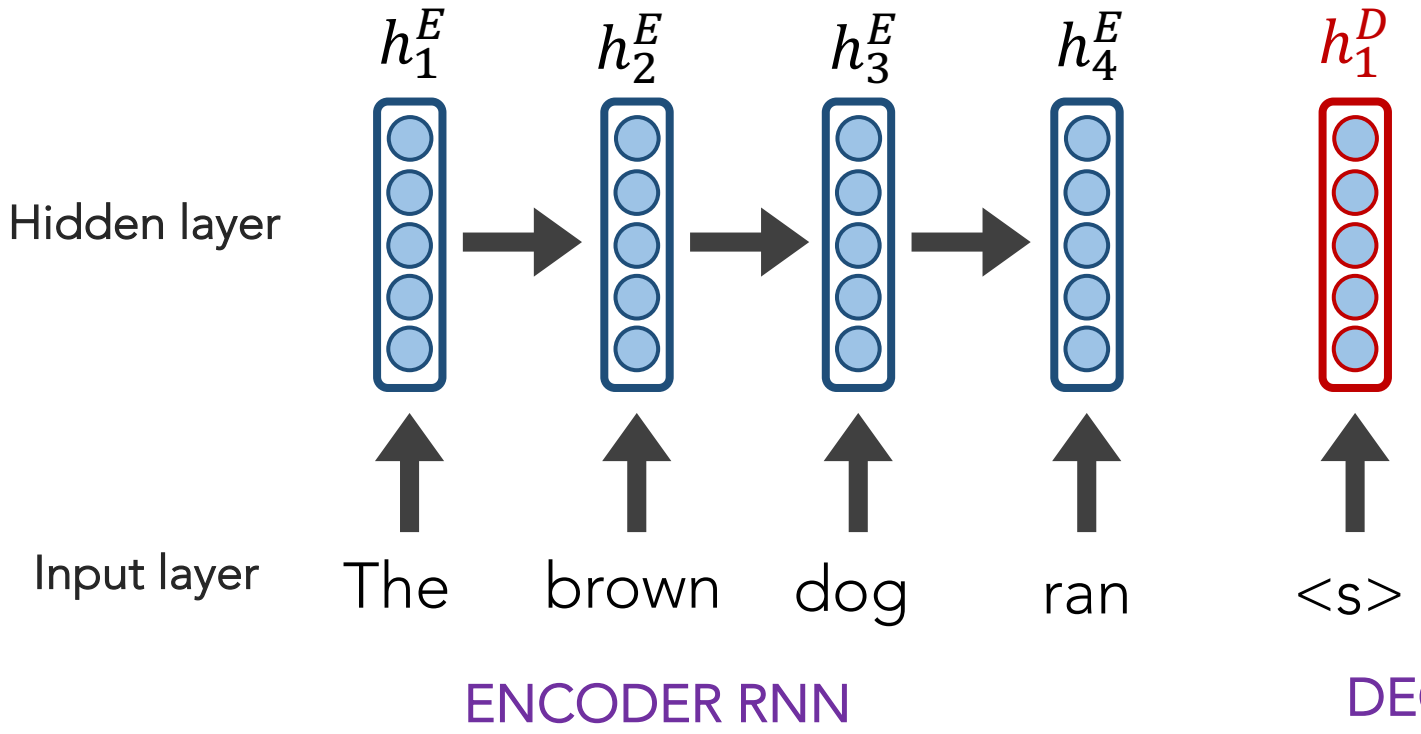
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

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



Attention (raw scores)

e_1 1.5

e_2 0.9

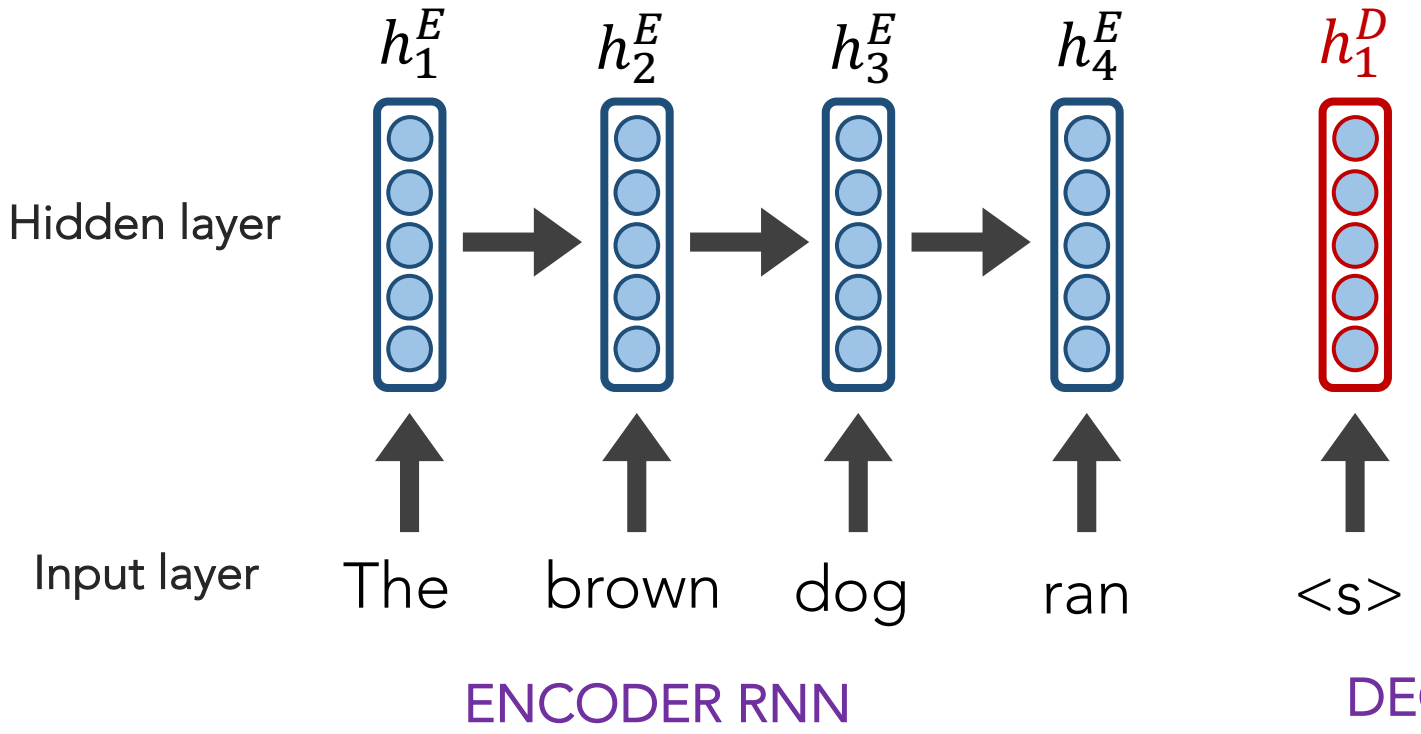
e_3 0.2

e_4 −0.5

seq2seq + Attention

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

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



Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

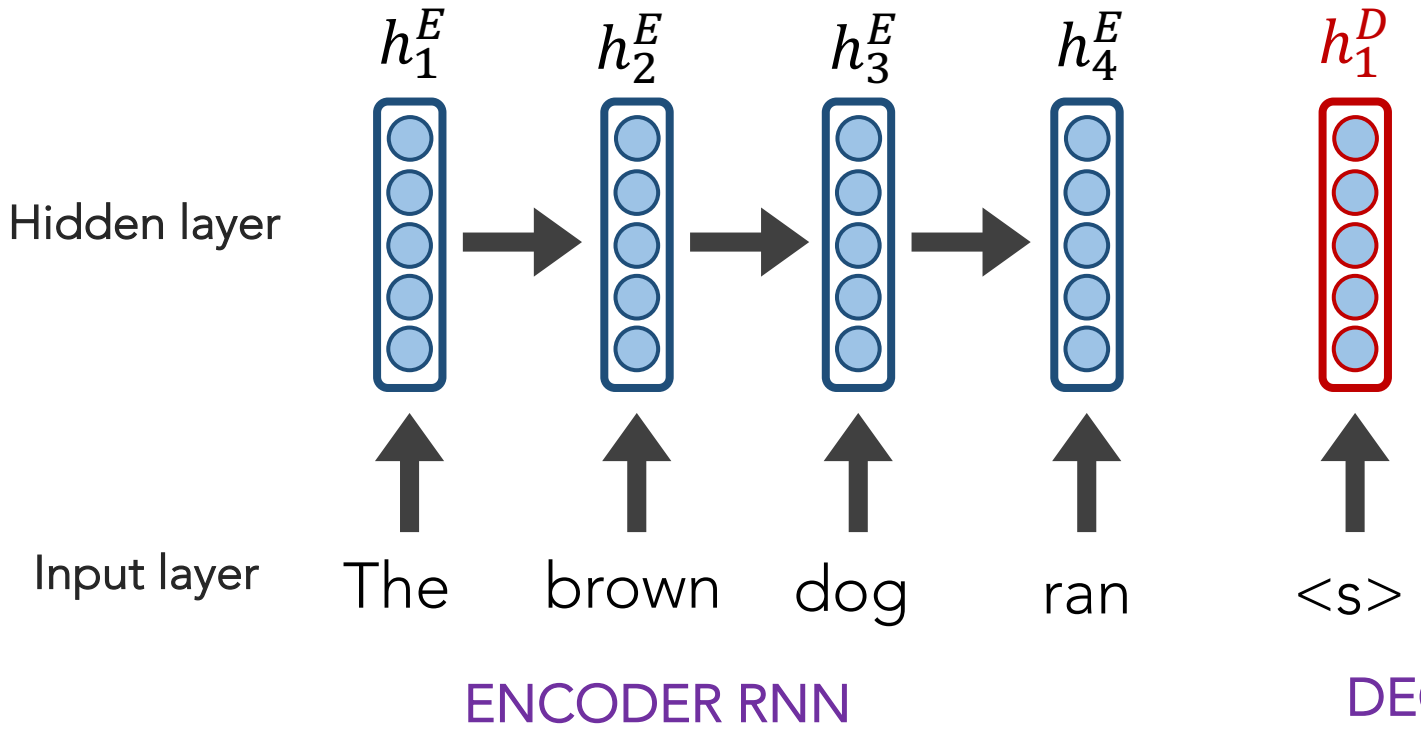
Attention (softmax'd)

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

seq2seq + Attention

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

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



Attention (raw scores)

e_1 1.5

e_2 0.9

e_3 0.2

e_4 −0.5

Attention (softmax'd)

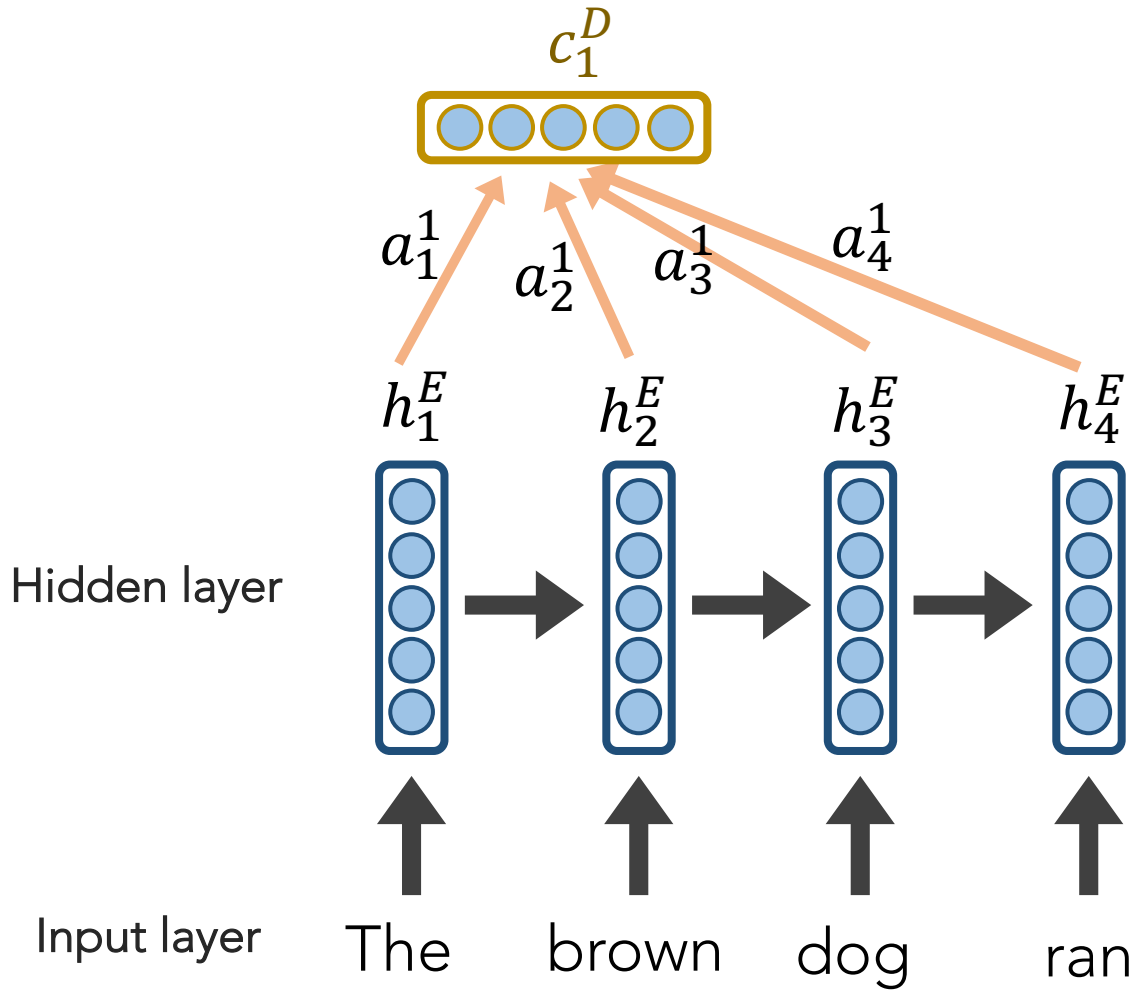
$a_1^1 = 0.51$

$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

seq2seq + Attention



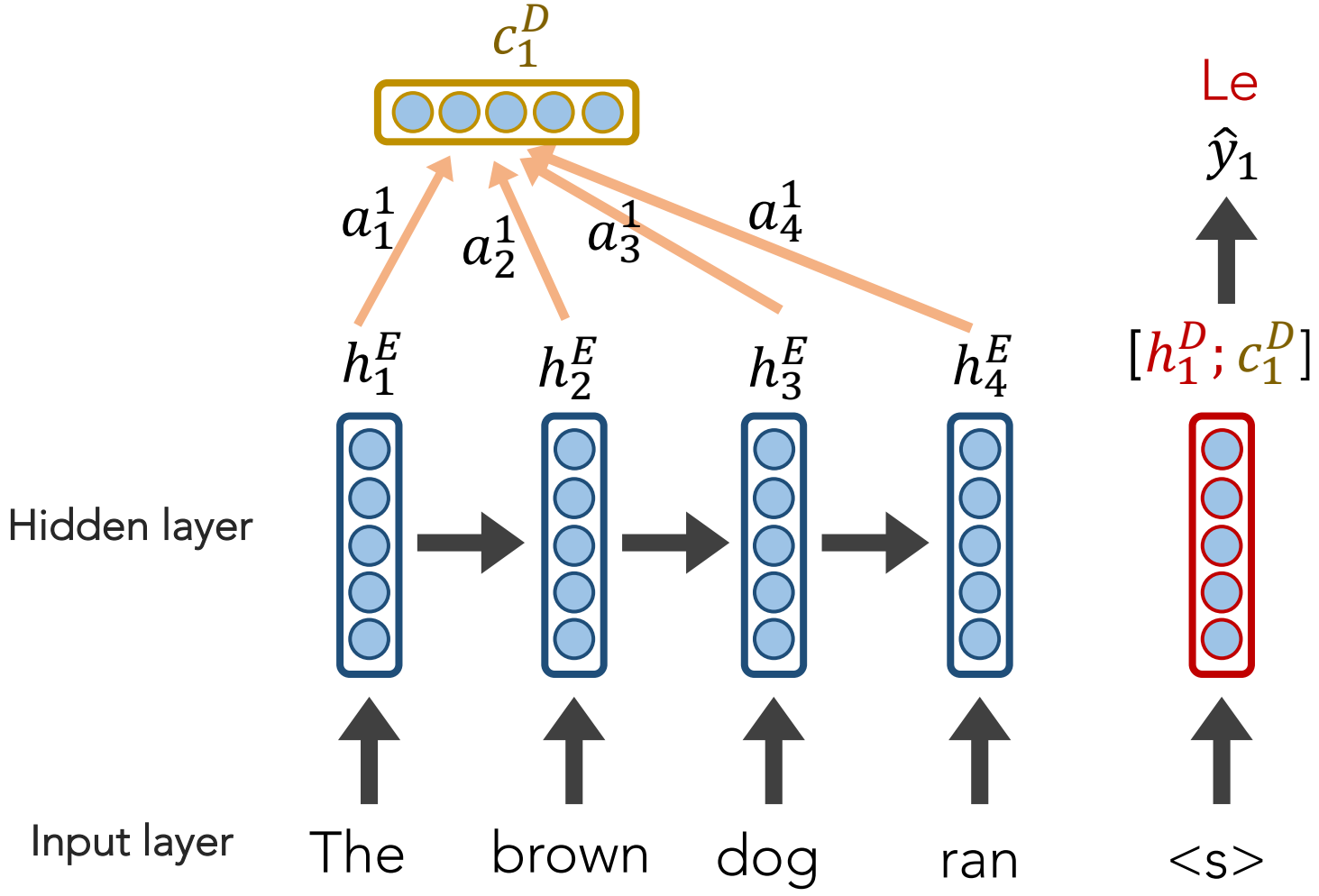
We multiply each encoder's hidden layer by its a_i^1 attention weights to create a context vector c_1^D

Attention (softmax'd)

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

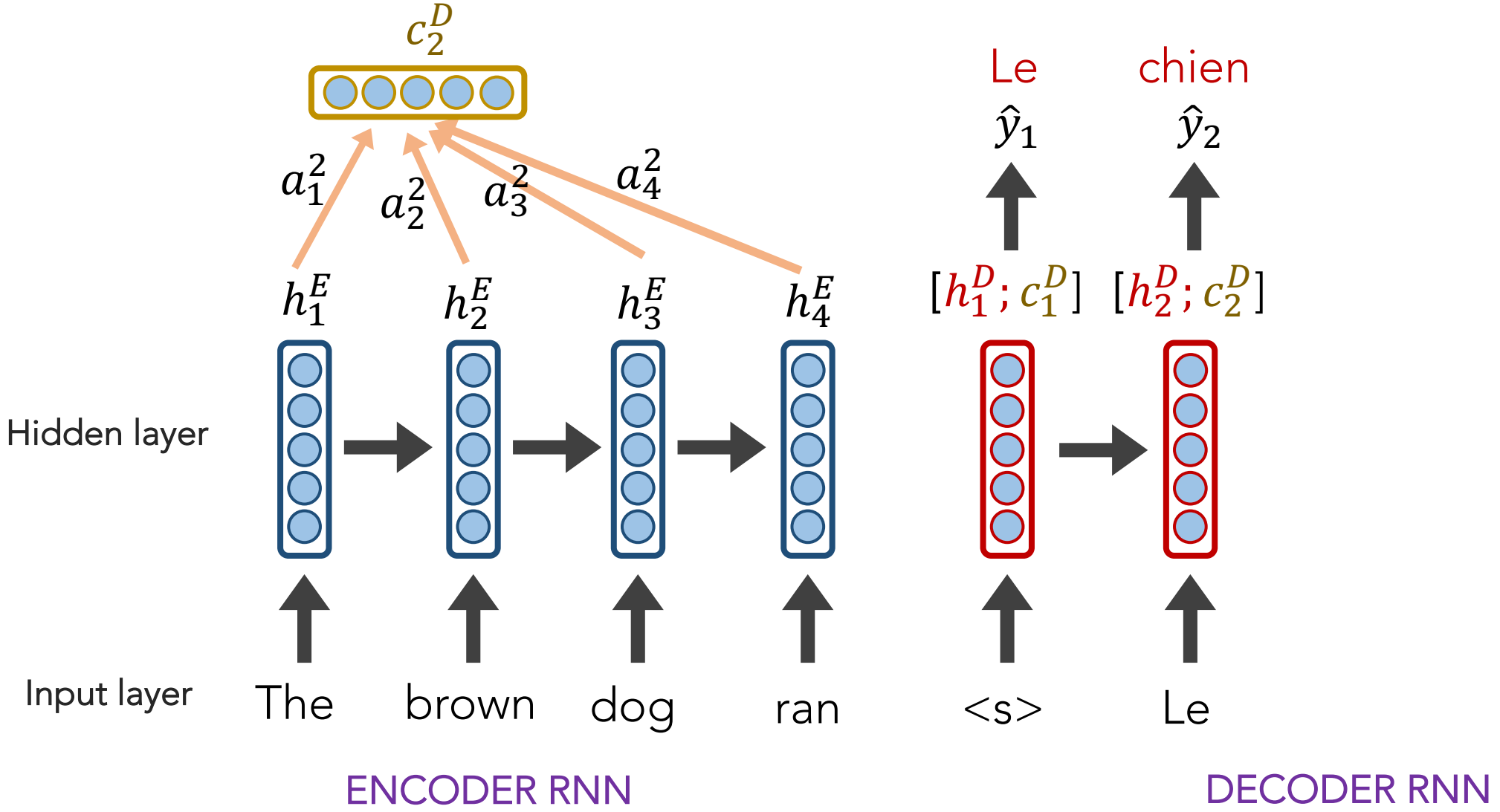
seq2seq + Attention

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



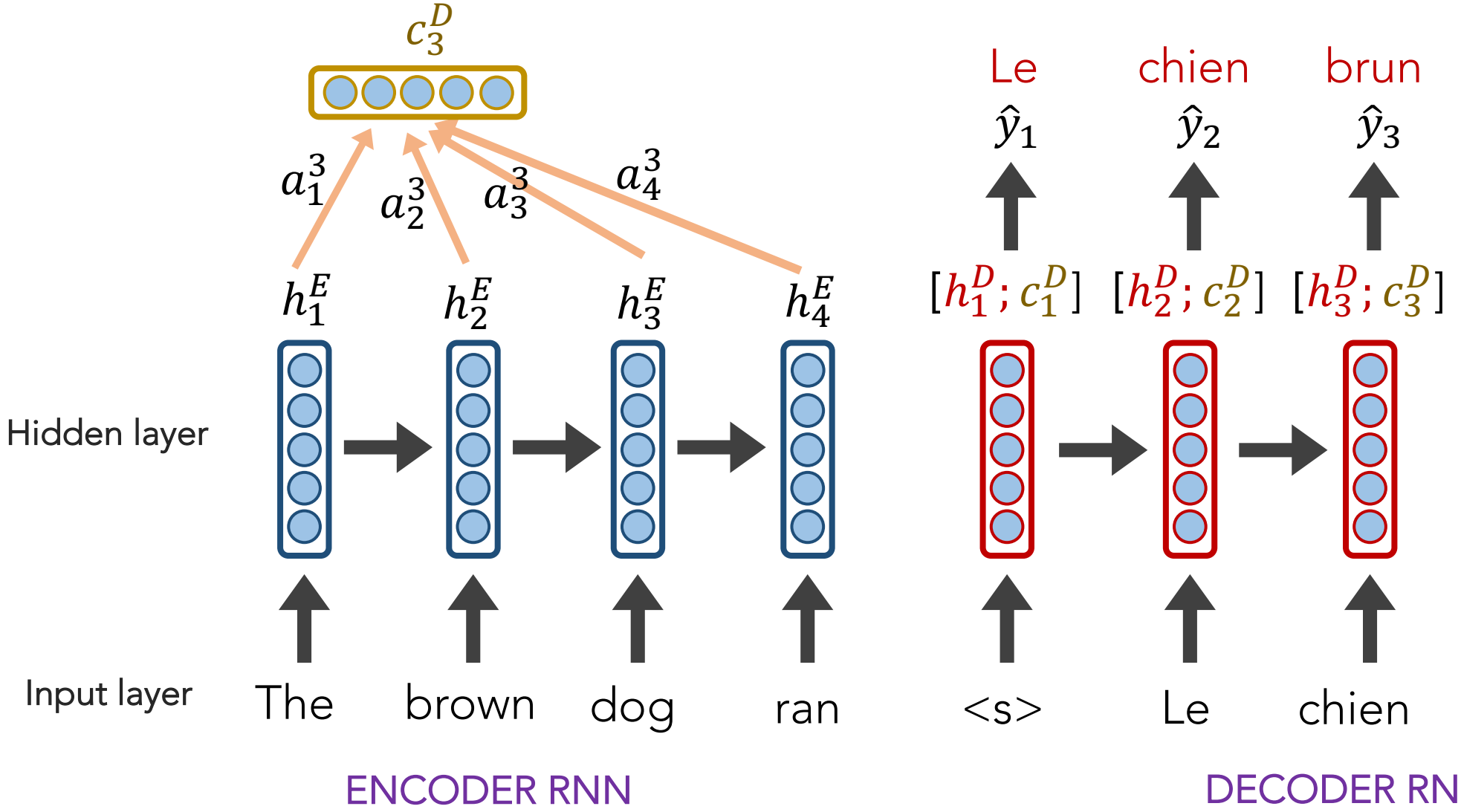
seq2seq + Attention

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



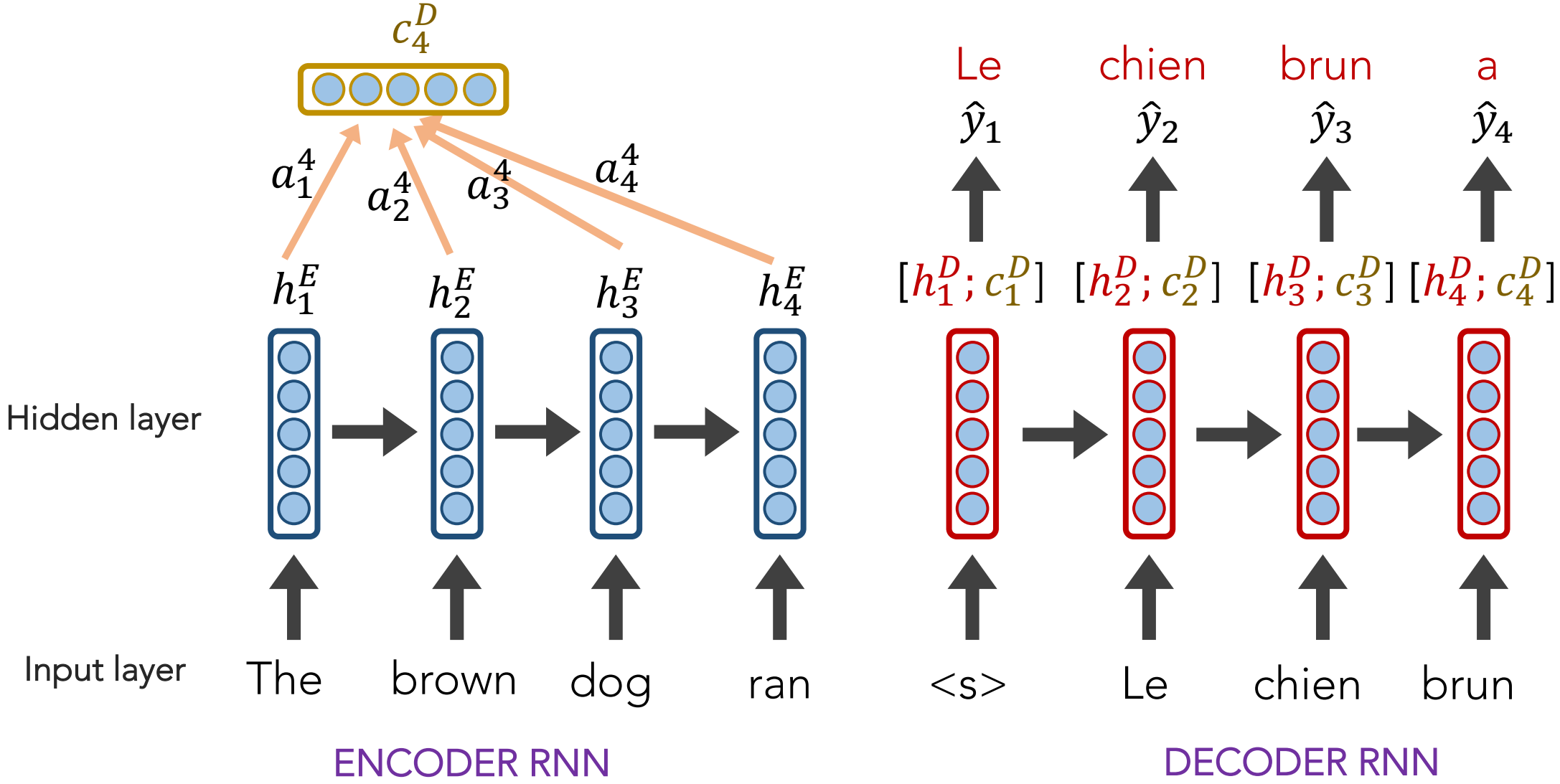
seq2seq + Attention

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



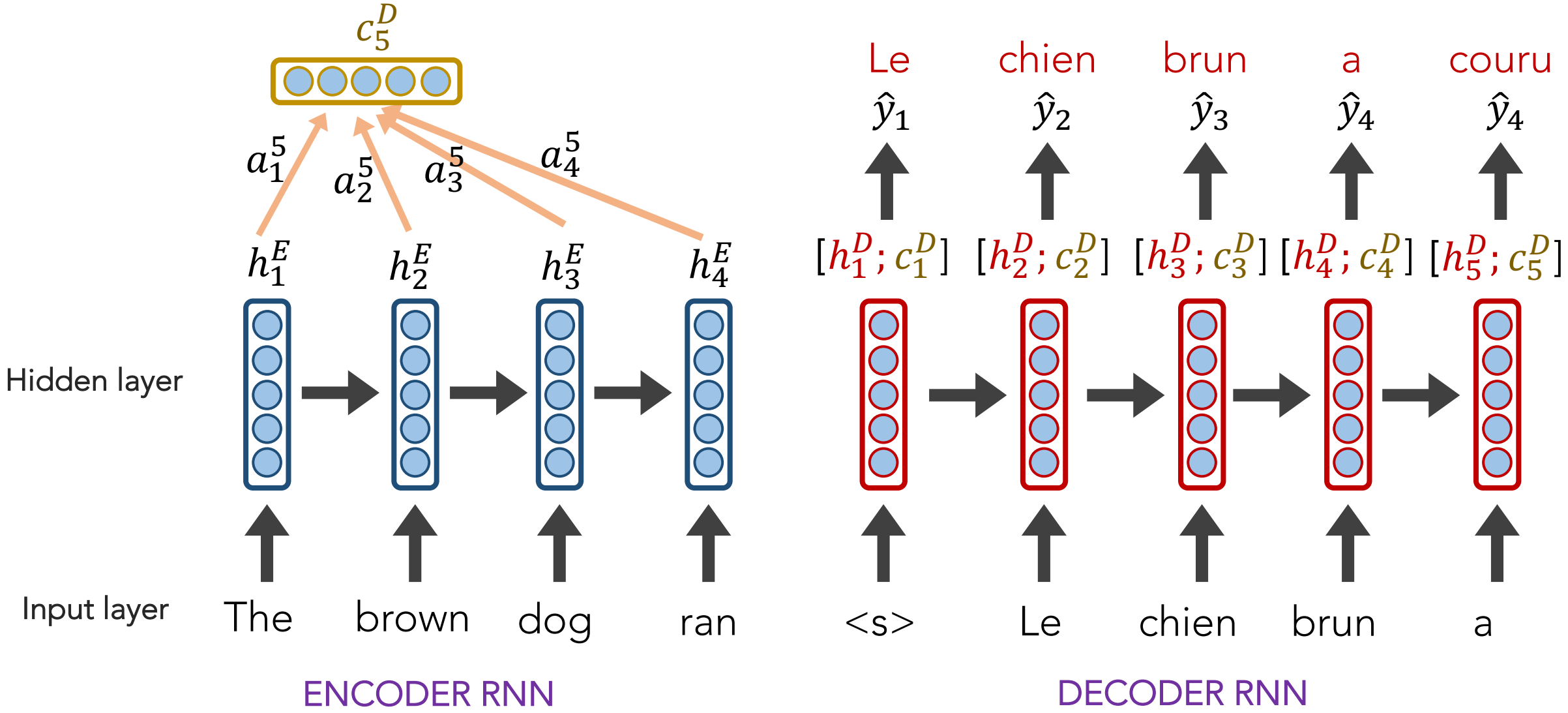
seq2seq + Attention

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



seq2seq + Attention

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



seq2seq + Attention

Attention:

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

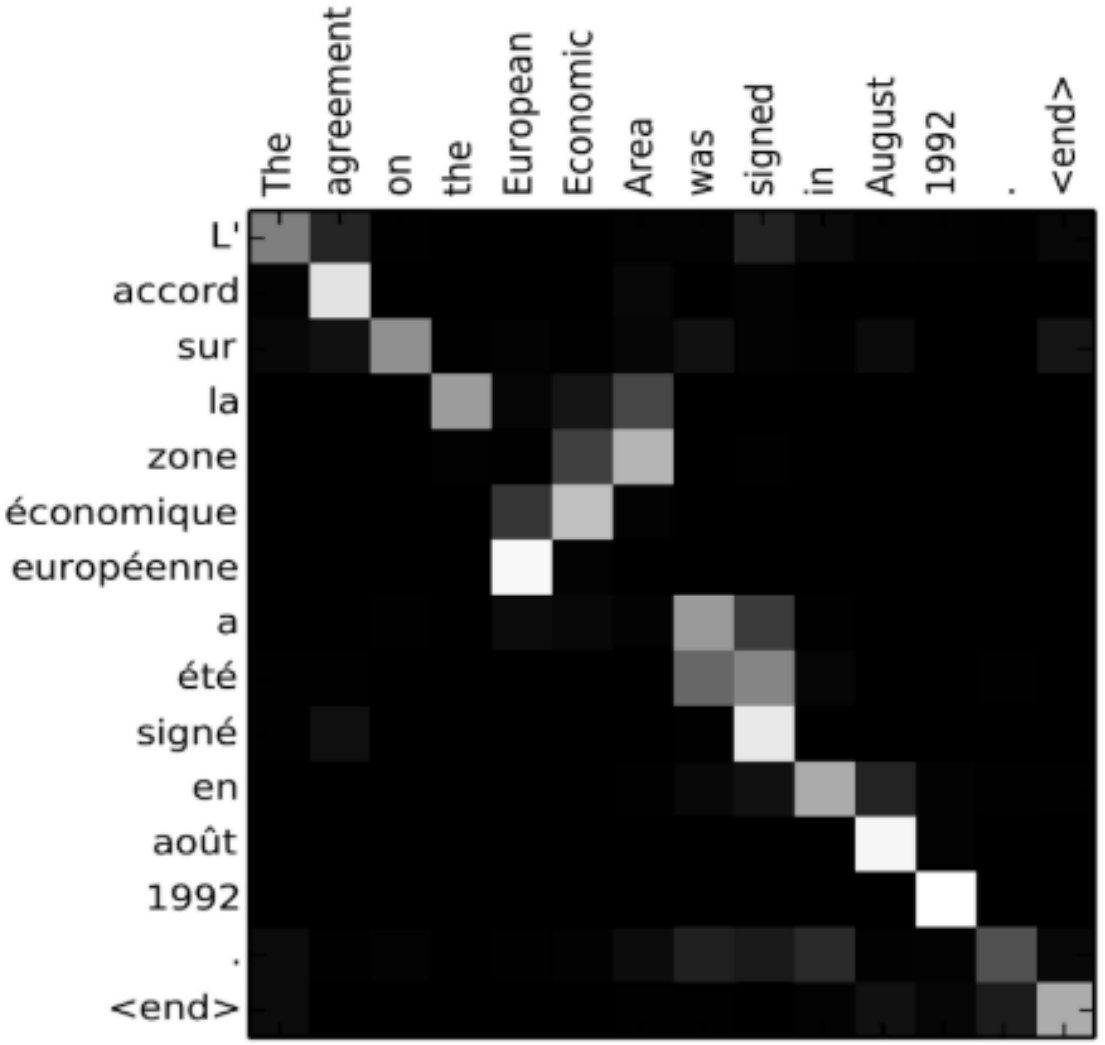


Image source: Fig 3 in [Bahdanau et al., 2015]

Takeaway #3:

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

Attention is powerful; allows us to conditionally weight our focus

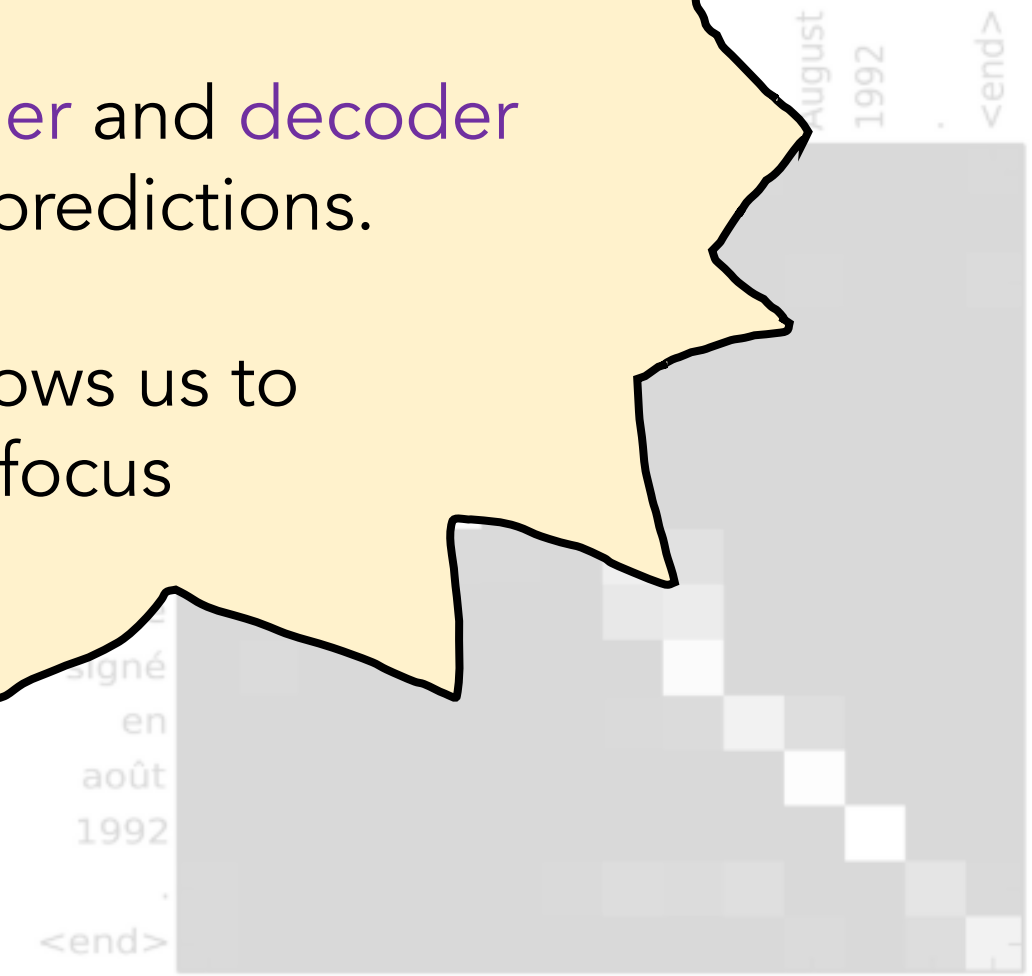


Image source: Fig 3 in Bahdanau et al., 2015

Sequence-to-Sequence (seq2seq)

seq2seq + **Attention** is great, but can we create better *encoder representations* and *decoder representations*?

Language Modelling

RNNs/LSTMs

Seq2Seq +Attention

Transformers +BERT and GPT

Conclusions