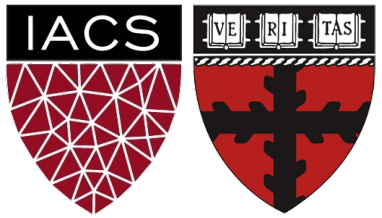
Lecture 24: Attention

NLP Lectures: Part 3 of 4

Harvard IACS

CS109B

Pavlos Protopapas, Mark Glickman, and Chris Tanner



Outline

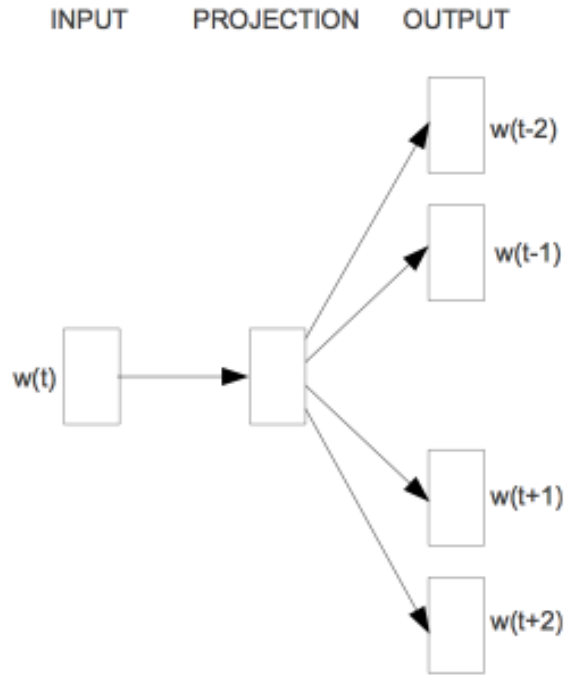
- How to use embeddings
- seq2seq
- seq2seq + Attention
- Transformers (preview)

Outline

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

Previously, we learned about **word embeddings**

millions of books



word2vec

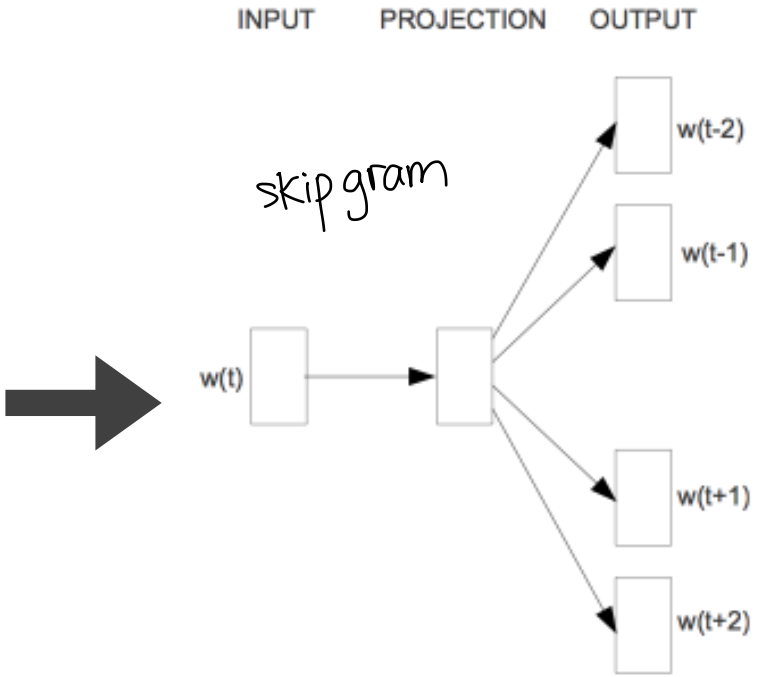
word embeddings (type-based)

approaches:

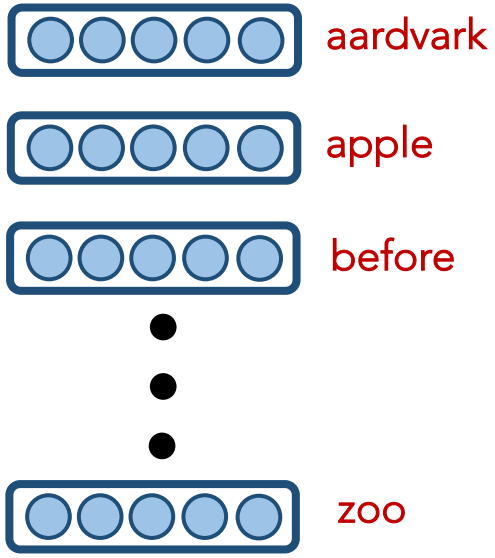
- count-based/DSMs (e.g., SVD, LSA)
- Predictive models (e.g., word2vec, GloVe)

Previously, we learned about **word embeddings**

millions of books



word2vec



word embeddings

word embeddings (type-based)

approaches:

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

one embedding for each word

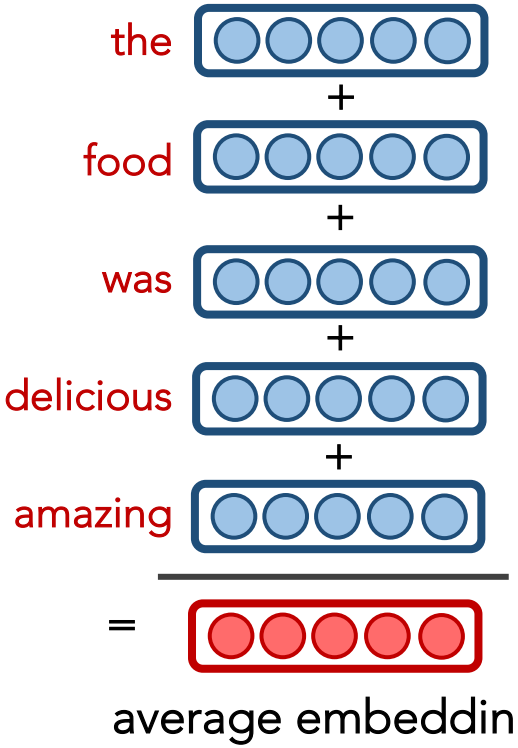
"The food was delicious. Amazing!" → **4.8/5** yelp

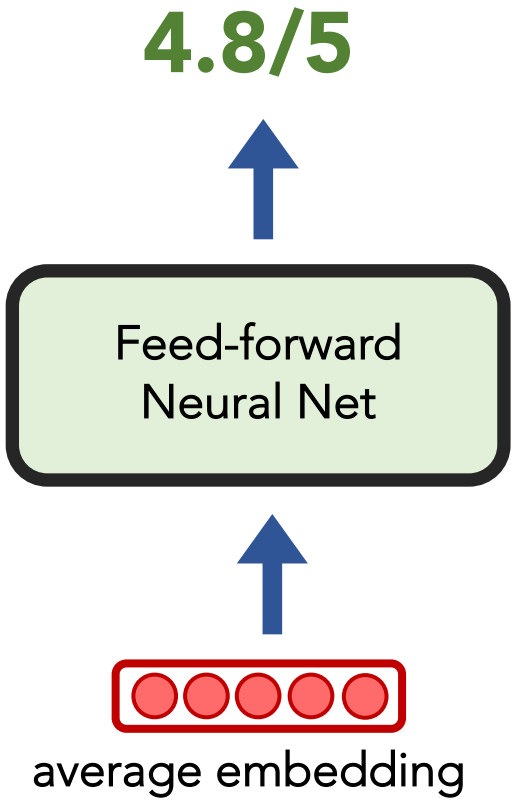
word embeddings (type-based)

approaches:

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

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



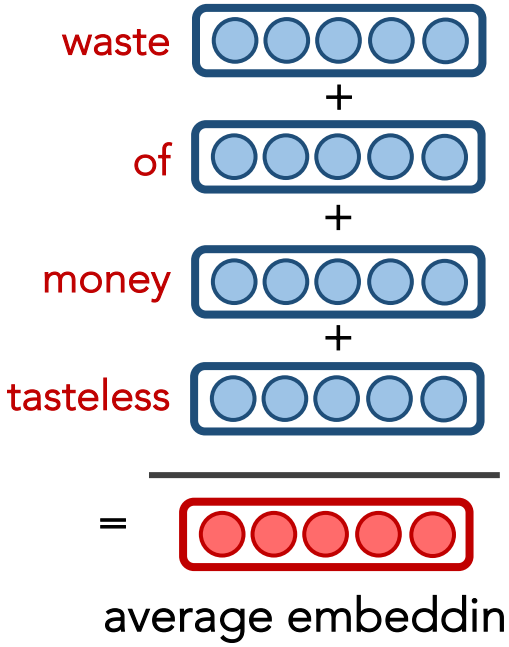


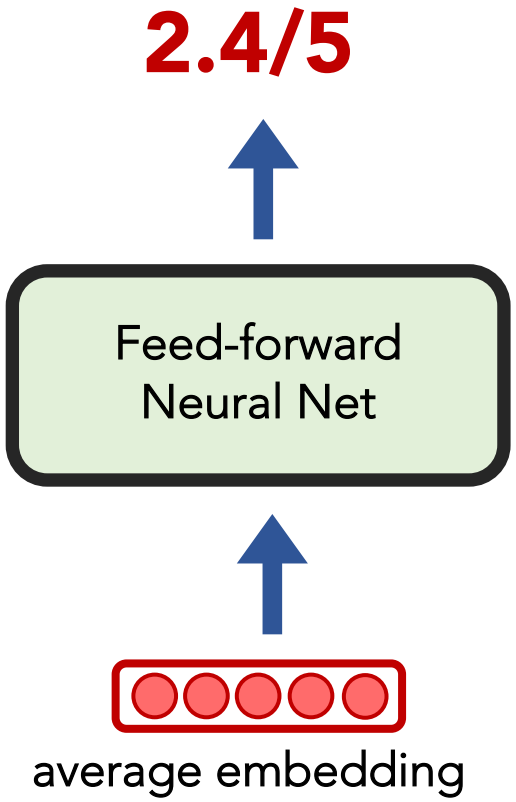
word embeddings (type-based)

approaches:

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

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



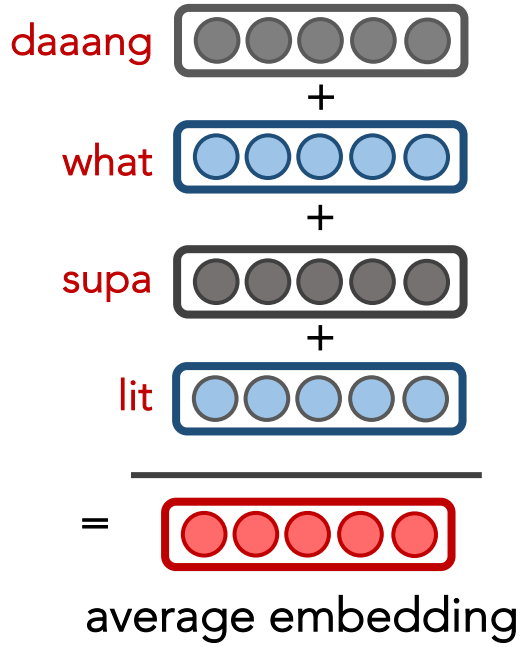


word embeddings (type-based)

approaches:

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

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



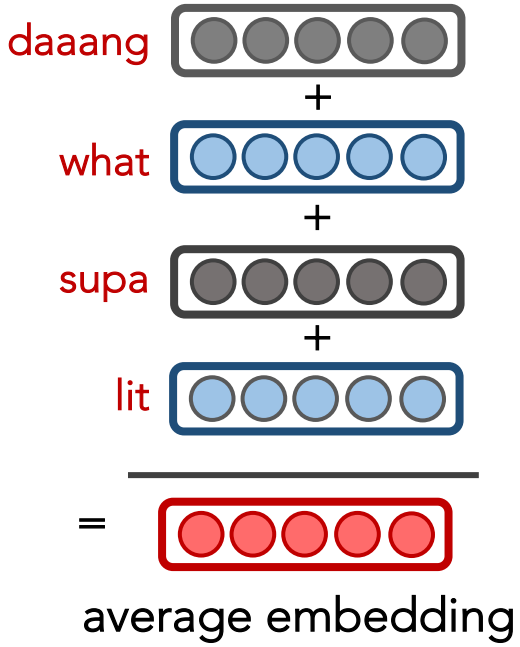
Strengths and weaknesses of word embeddings (type-based)?

word embeddings (type-based)

approaches:

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

external wealth if the training data is not large



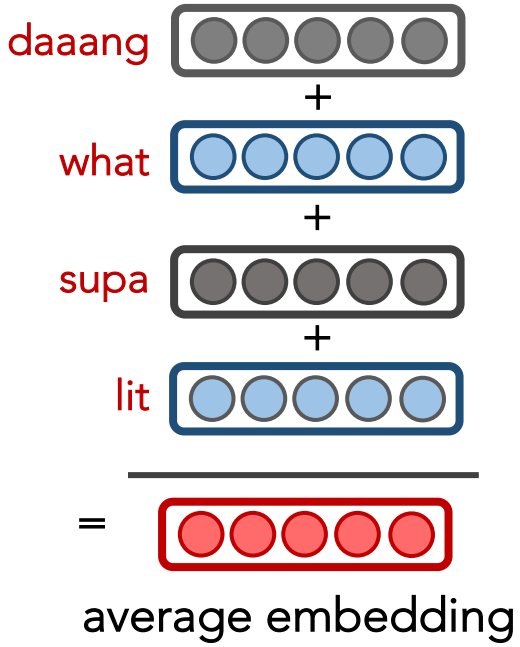
Strengths:

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

word embeddings (type-based)

approaches:

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



Issues:

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

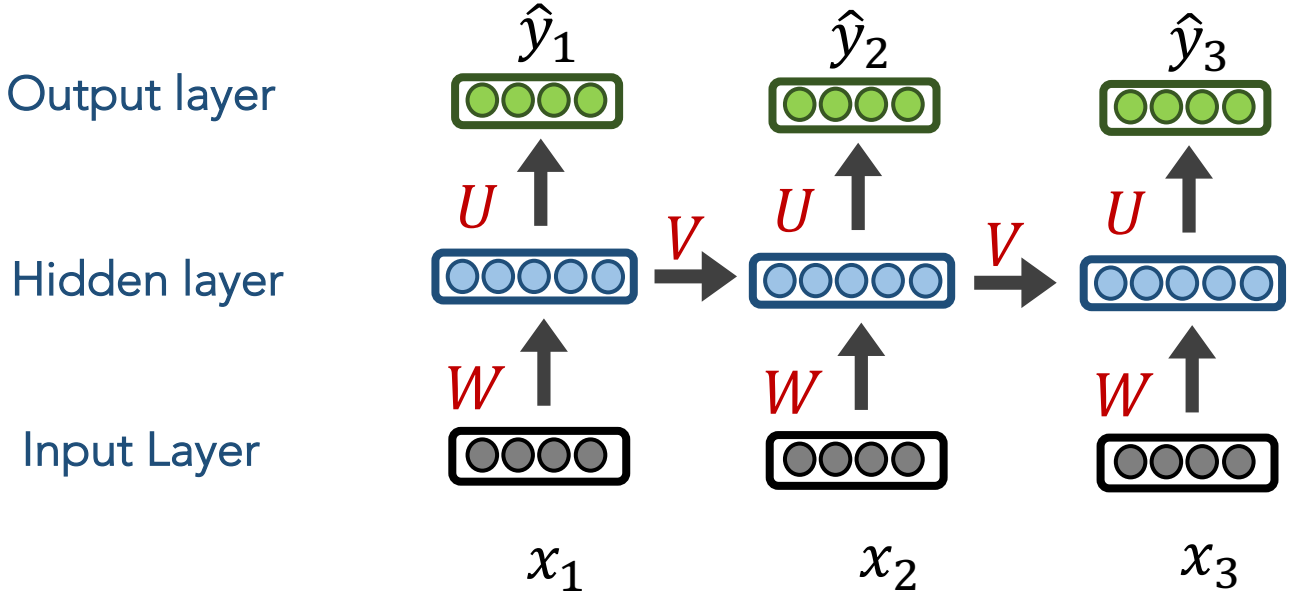
some of the words in the view might be more important than others

word embeddings (type-based)

approaches:

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

Previously, we learned about **word embeddings**



LSTM

contextualized embeddings (token-based)

approaches:

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

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

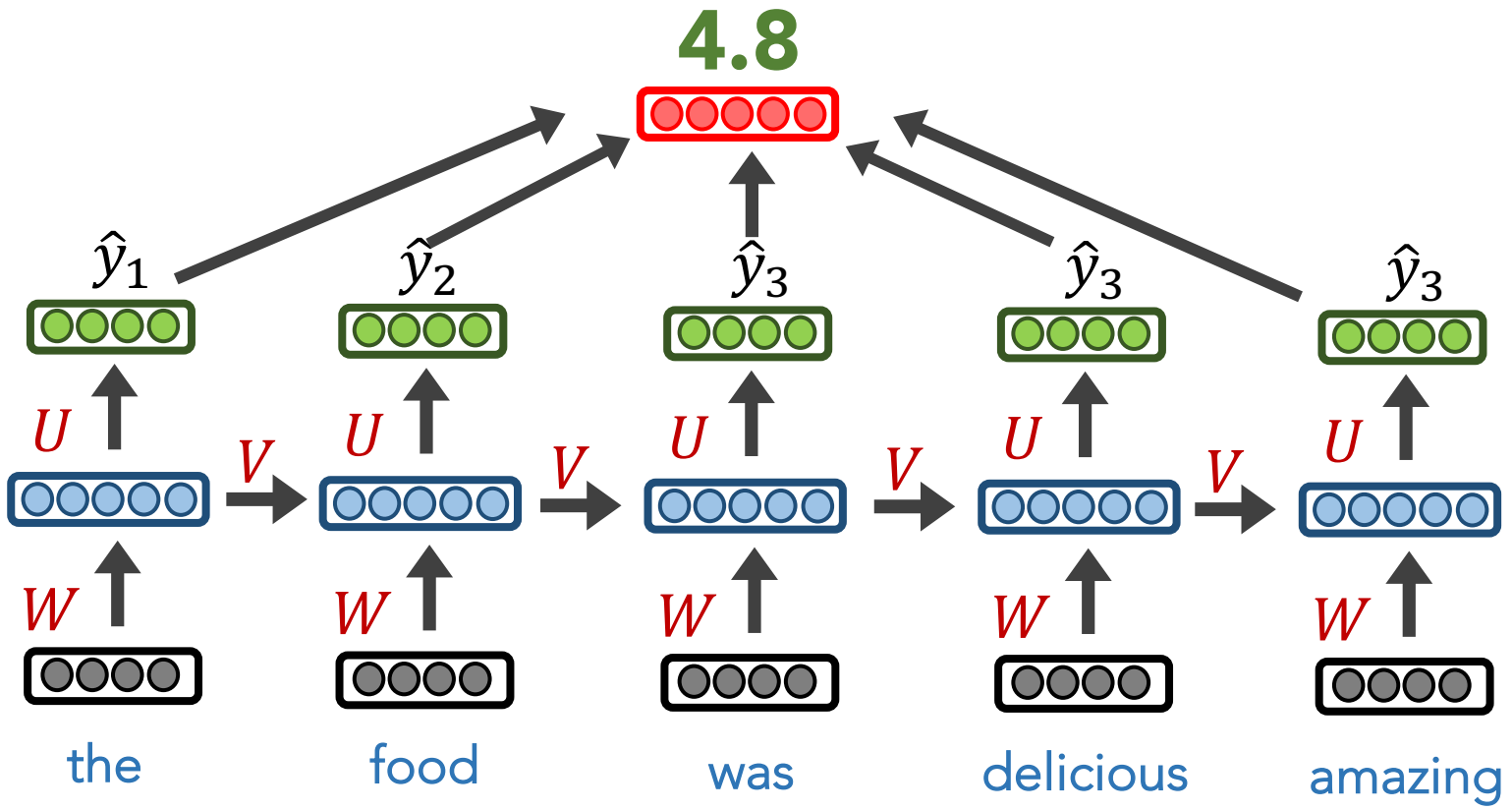
2. Chalawan
108
Thai, Indonesian, Singaporean

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

4. Posto
912
$$ • Italian, Pizza

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

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #1

contextualized embeddings (token-based)

approaches:

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

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

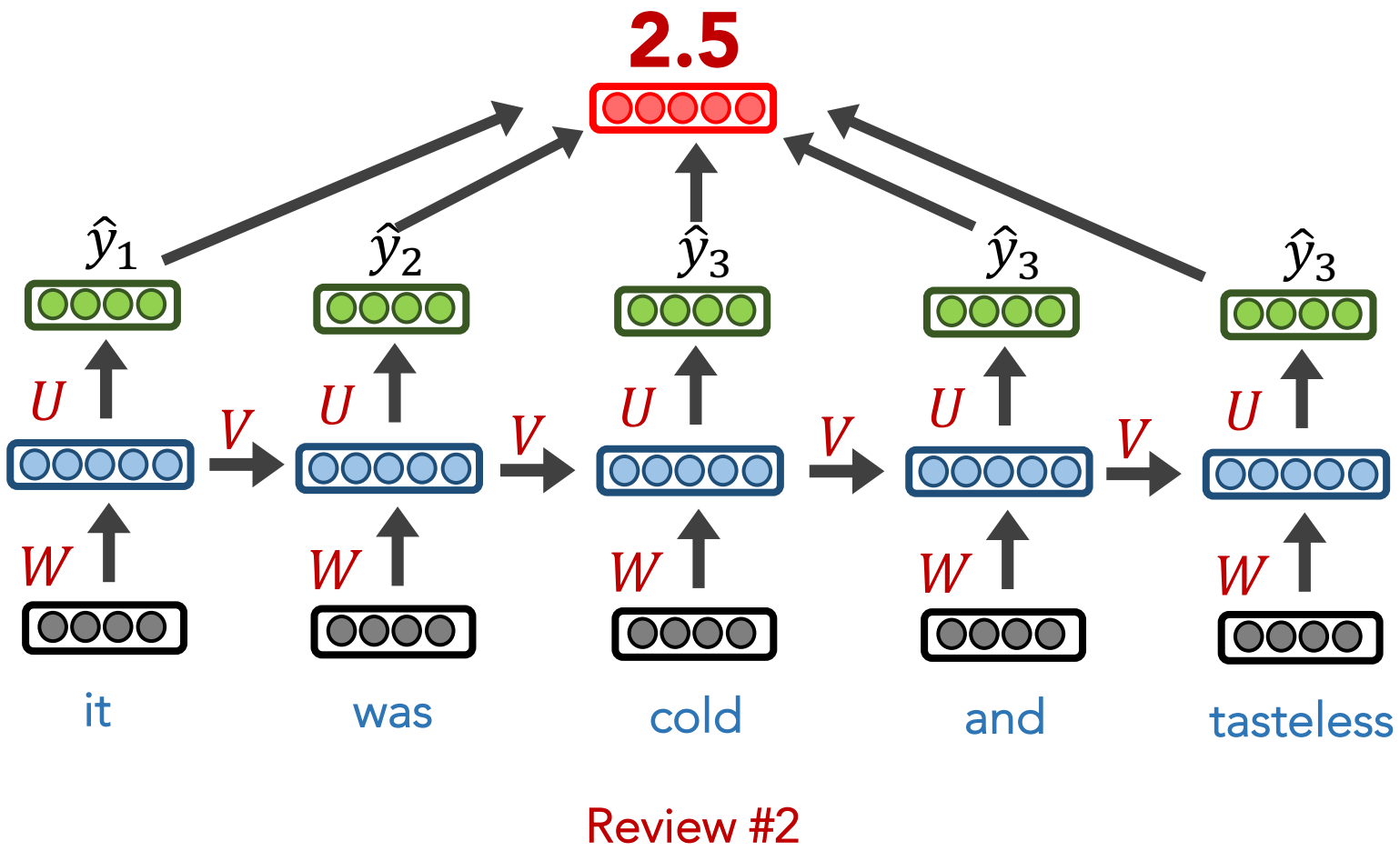
2. Chalawan
108
Thai, Indonesian, Singaporean

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

4. Posto
912
$$ • Italian, Pizza

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

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #2

contextualized embeddings (token-based)

approaches:

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

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

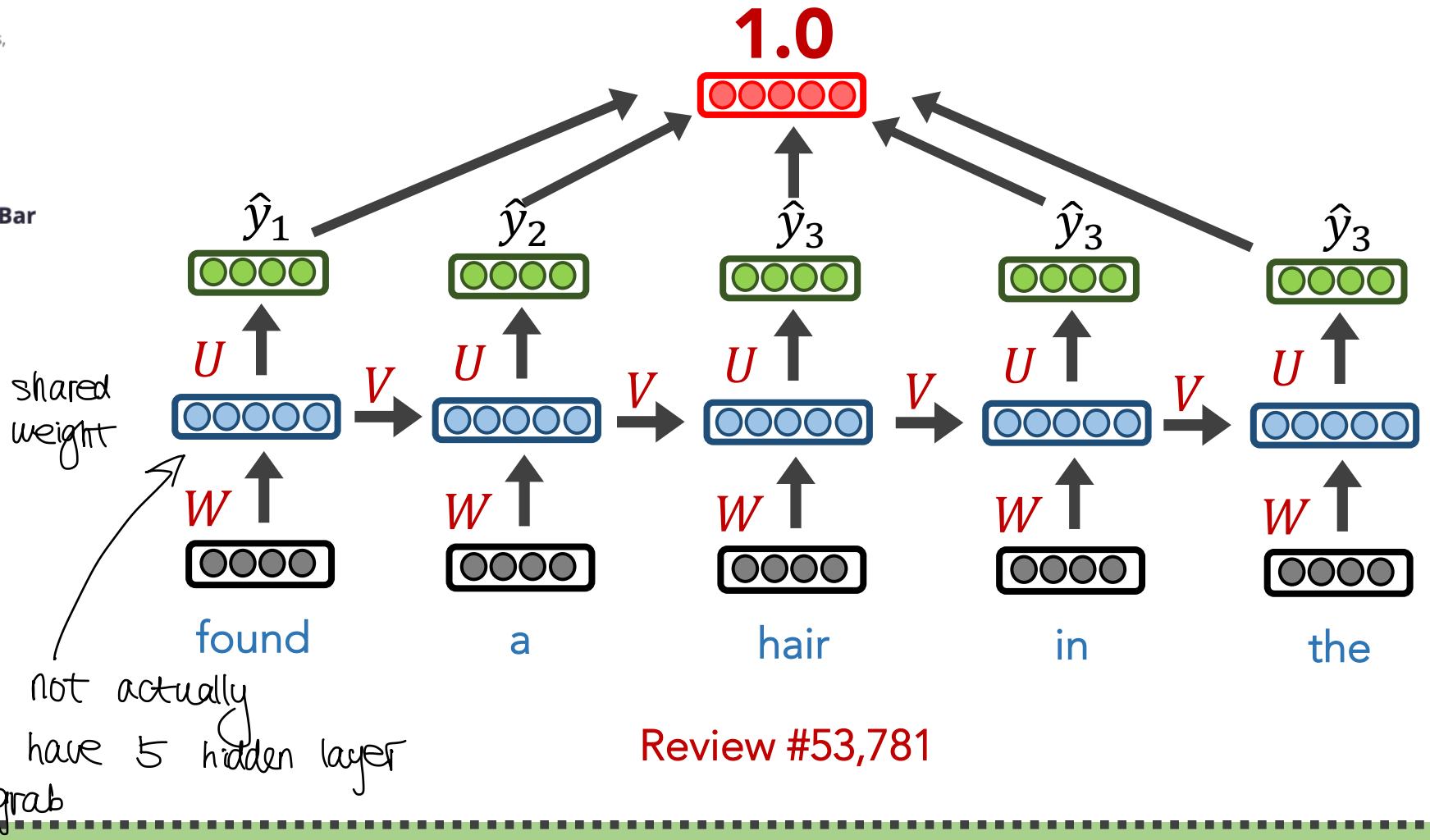
2. Chalawan
108
Thai, Indonesian, Singaporean

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

4. Posto
912
$$ • Italian, Pizza

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

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop



Review #53,781

contextualized embeddings (token-based)

approaches:

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

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

2. Chalawan
108
Thai, Indonesian, Singaporean

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

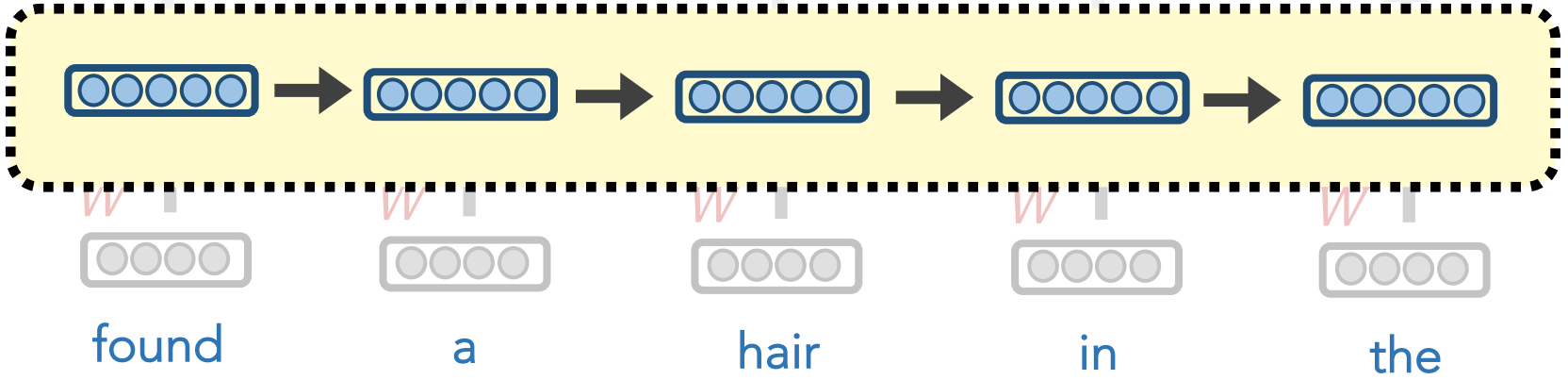
4. Posto
912
$$ • Italian, Pizza

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

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop

grab
(extract it)
for one word
when running codes

Every token in the corpus has a contextualized embedding



Review #53,781

contextualized embeddings (token-based)

approaches:

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

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

2. Chalawan
108
Thai, Indonesian, Singaporean

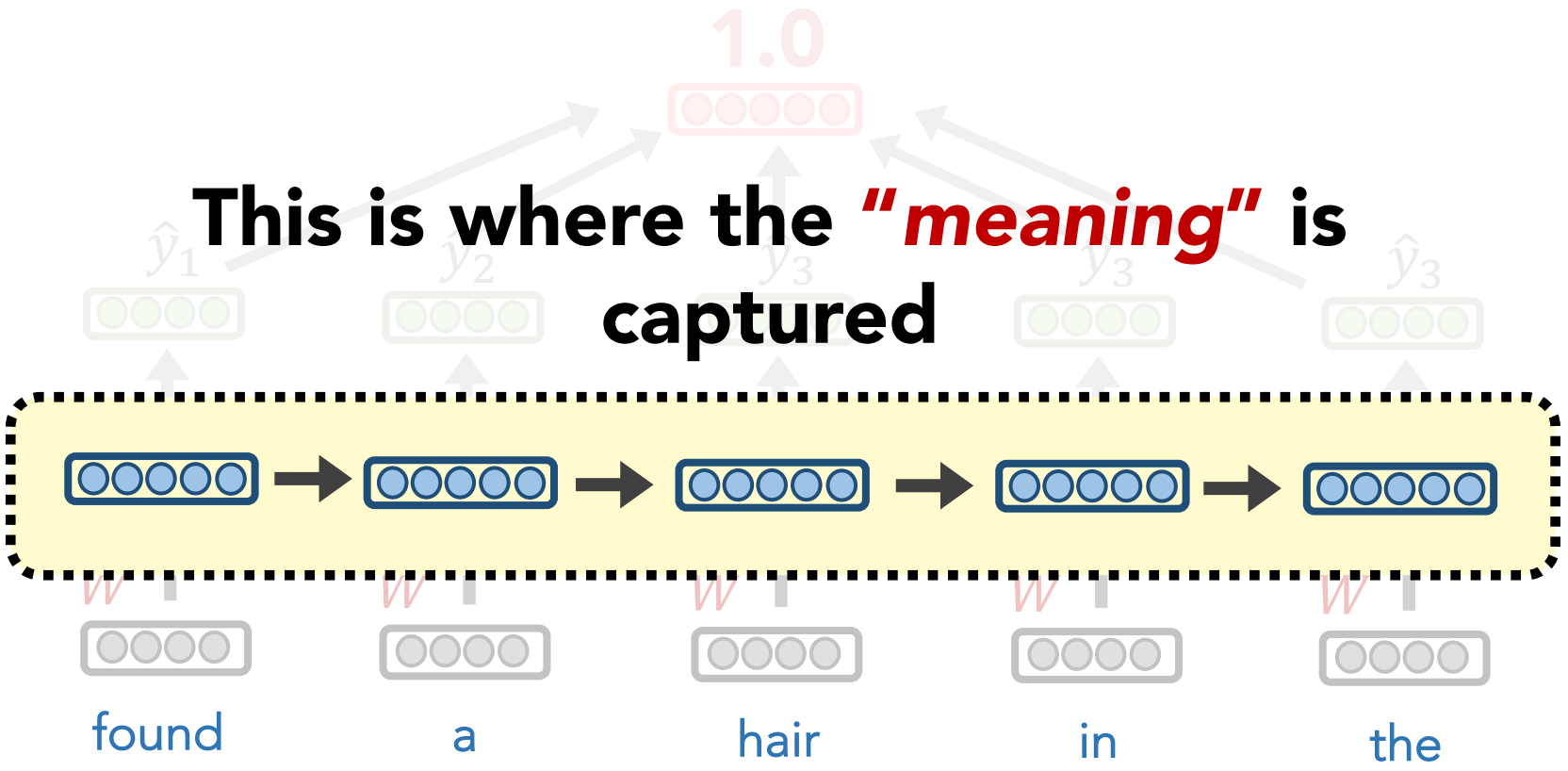
3. Gustazo Cuban Kitchen & Bar
162
Cuban, Tapas/Small Plates, Bars

4. Posto
912
$$ • Italian, Pizza

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

7192. Underdog Hot Chicken
45
Chicken Wings, Chicken Shop

This is where the *"meaning"* is captured



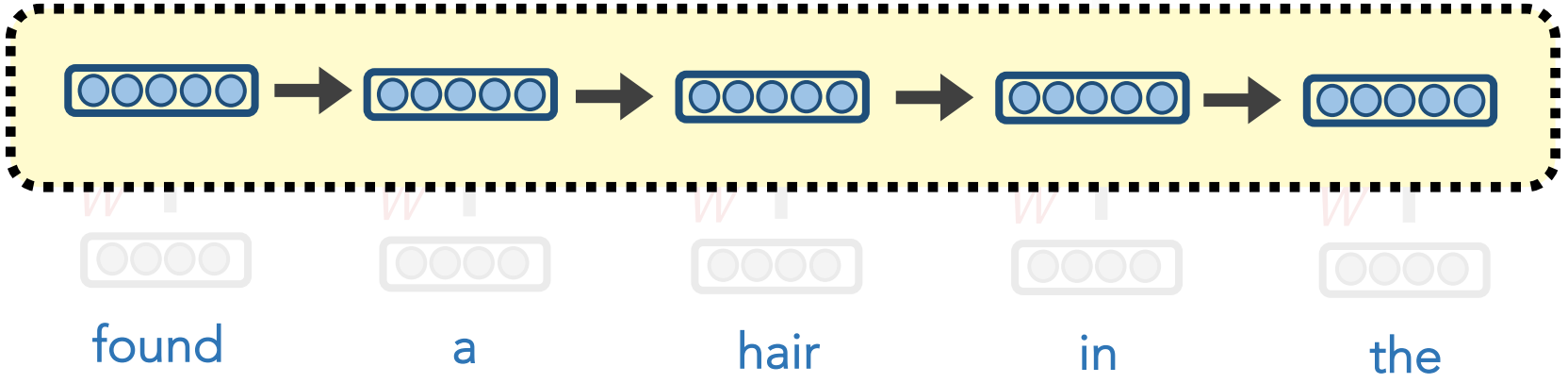
Review #53,781

contextualized embeddings (token-based)

approaches:

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

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



Review #53,781

contextualized embeddings (token-based)

approaches:

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

Strengths:

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

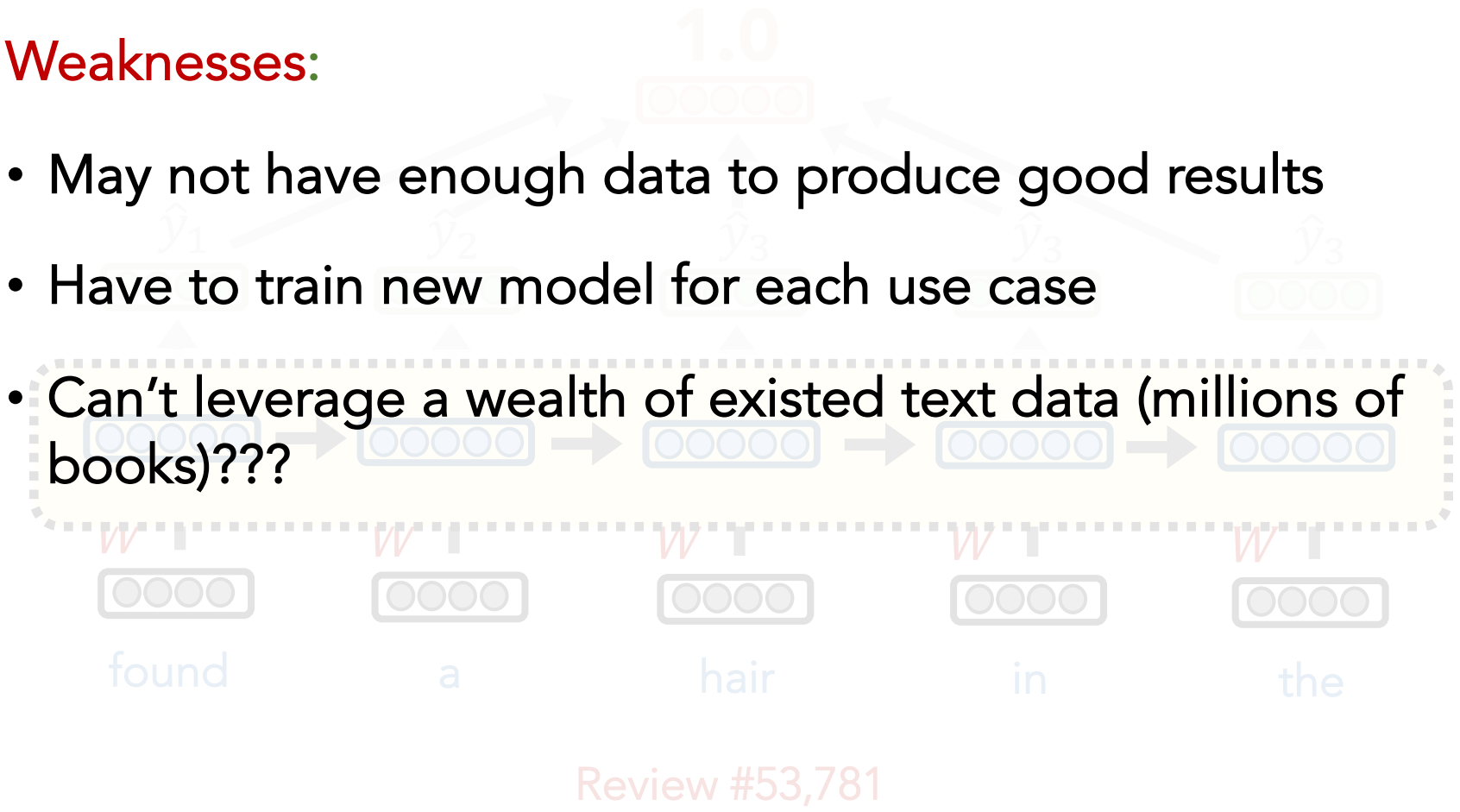
contextualized embeddings (token-based)

approaches:

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

Weaknesses:

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



contextualized embeddings (token-based)

approaches:

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

Weaknesses:

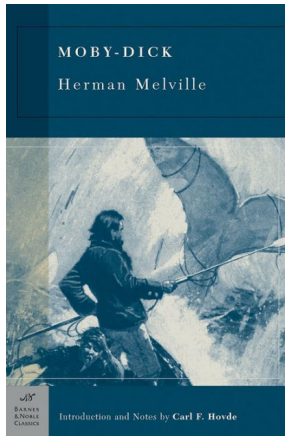
- May not have enough data to produce good results
- Have to train new model for each use case
- ~~Can't leverage a wealth of existed text data (millions of books)???~~

WRONG! We can leverage millions of books!

contextualized embeddings (token-based)

approaches:

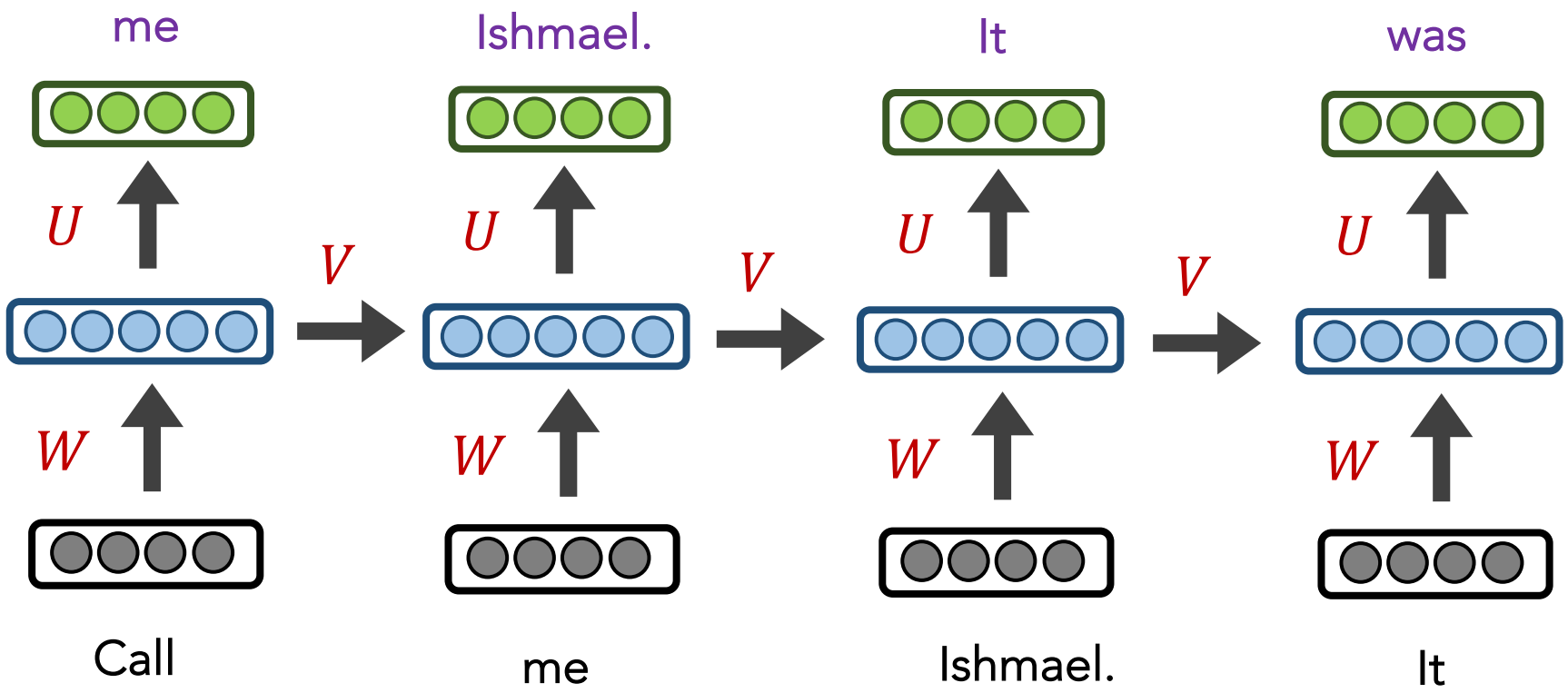
- Predictive models (e.g., BiLSTMs, GPT-2, BERT)

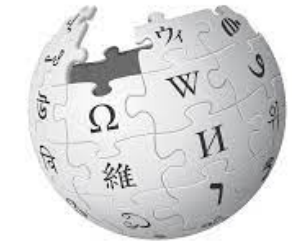


Language Modelling

(let's input 1 million documents)

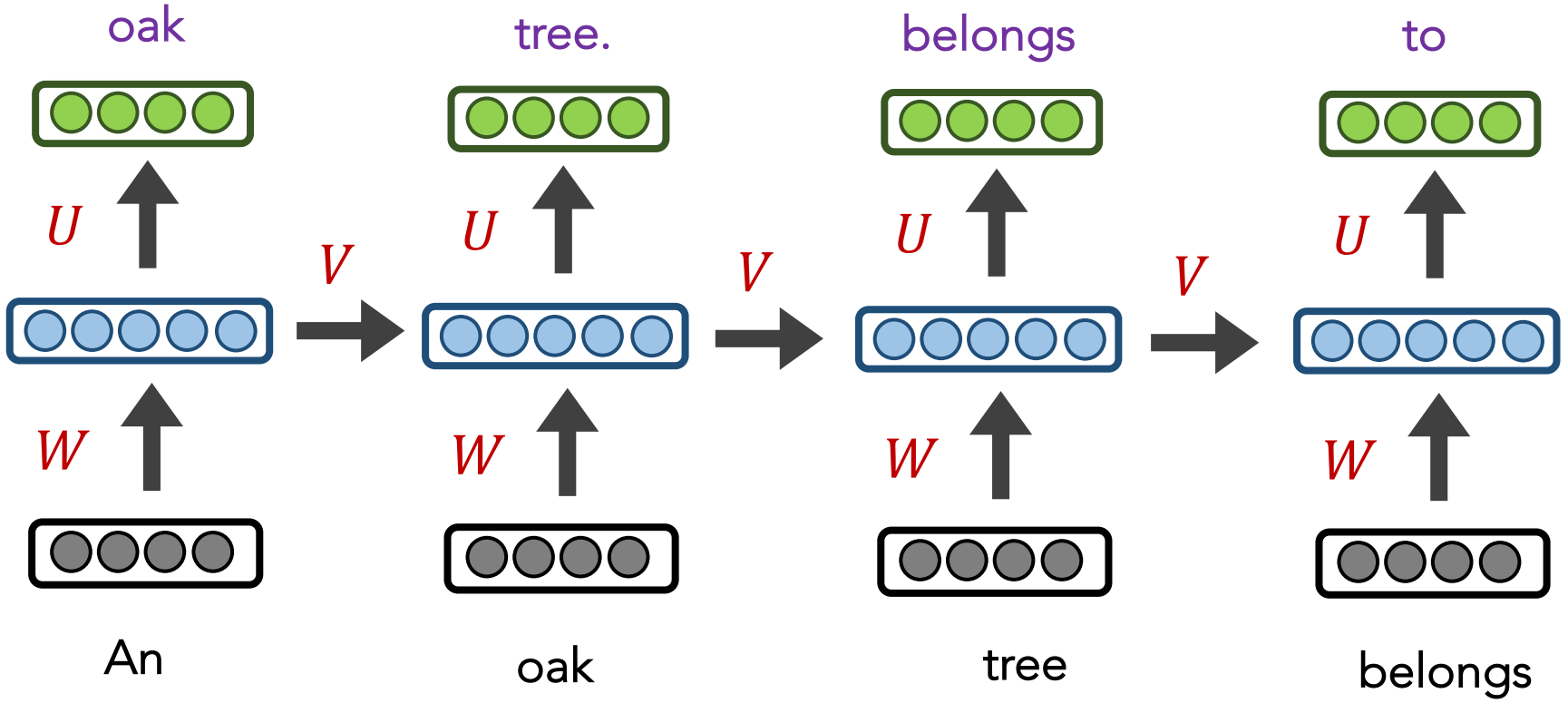
predict the next word





Language Modelling

(let's input 1 million documents)



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

(如果预测 Yelp，没人关心在一本书里的上下文情况)

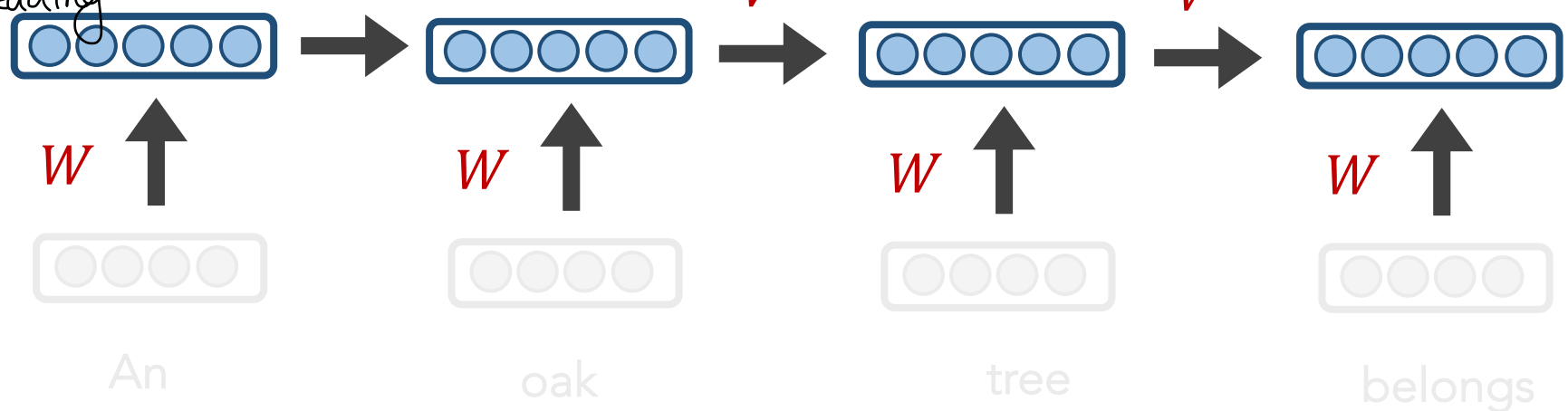
Learn a rich, robust W and V

weight ⇒ told what the embedding is in the future

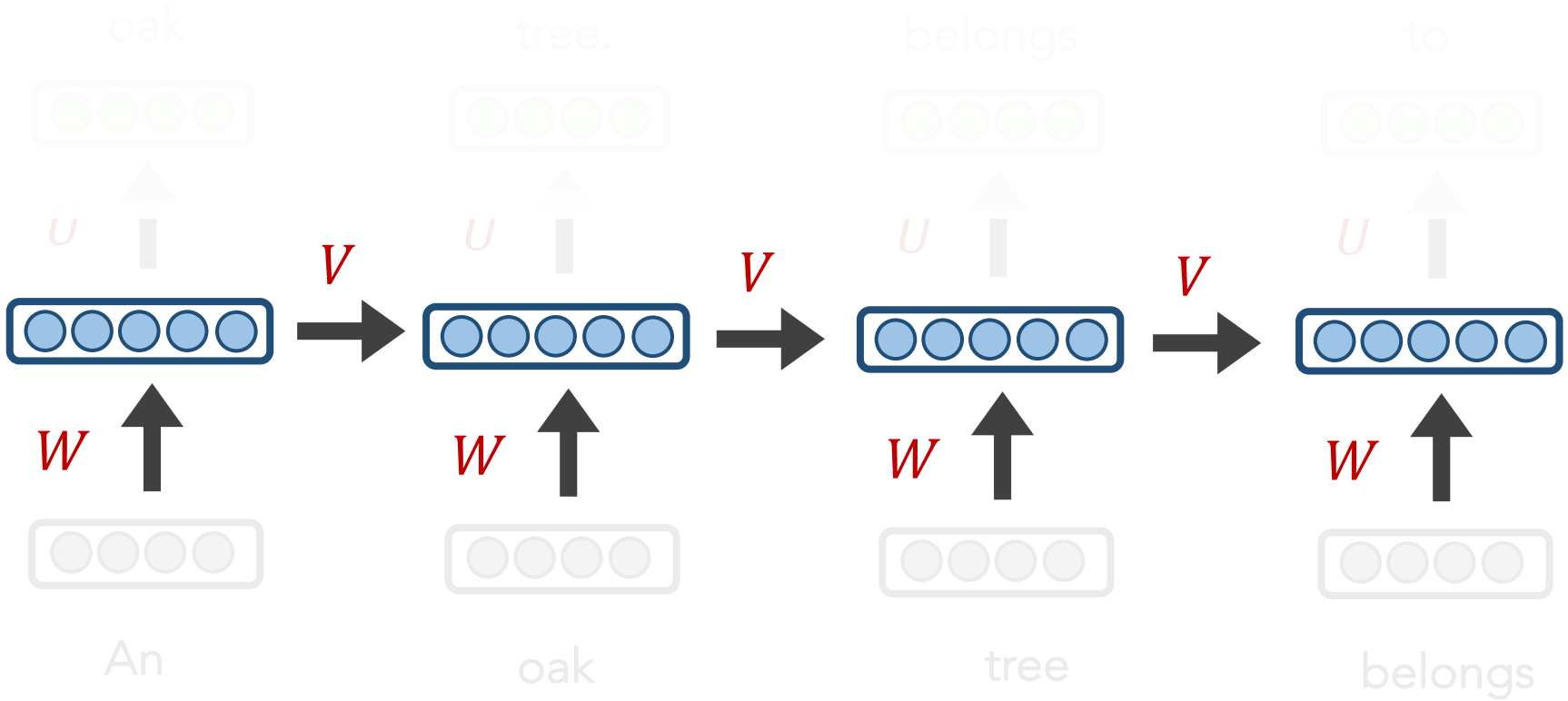
may be CNN involved
longer to train
technical ⇒ set boundary

not seen before
⇓
character embedding

or ngram:
token
special token



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



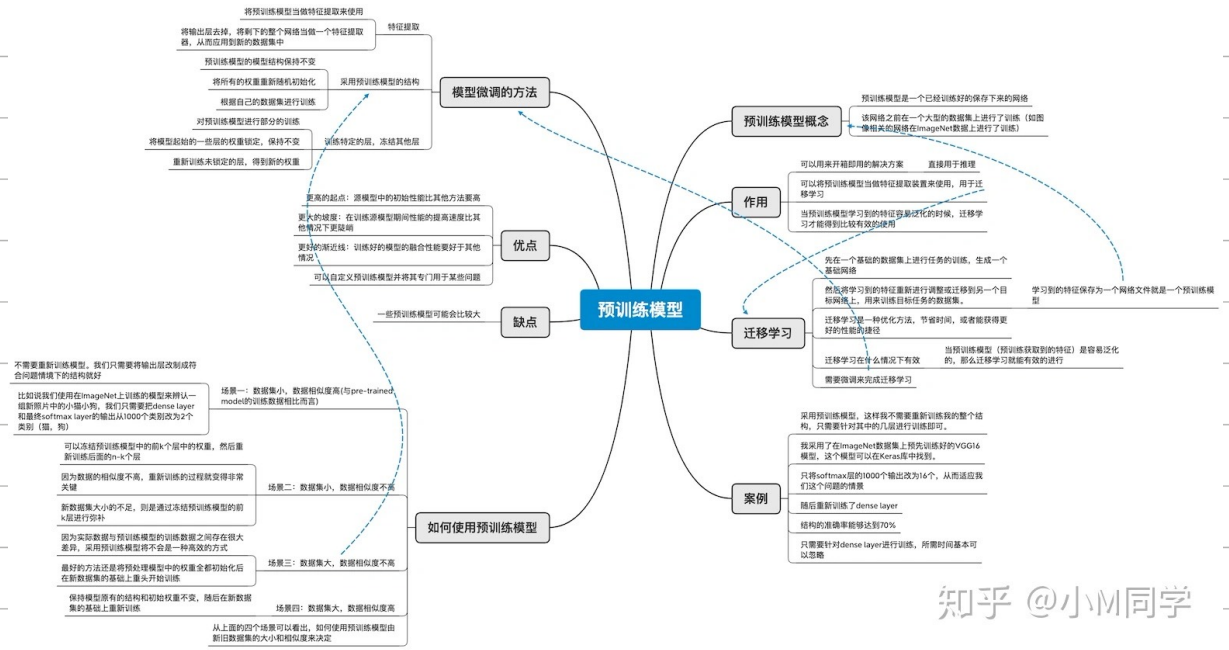
预训练通过自监督学习从大规模数据中获得与具体任务无关的预训练模型。体现某一个词在一个特定上下文中的语义表征。第二个步骤是微调，针对具体的任务修正网络。训练数据可以是文本、文本-图像对、文本-视频对。预训练模型的训练方法可使用自监督学习技术（如自回归的语言模型和自编码技术）。可训练单语言、多语言和多模态的模型。此类模型可经过微调之后，用于支持分类、序列标记、结构预测和序列生成等各项技术，并构建文摘、机器翻译、图片检索、视频注释等应用。

为什么我们要做预训练模型？首先，预训练模型是一种迁移学习的应用，利用几乎无限的文本，学习输入句子的每一个成员的上下文相关的表示，它隐式地学习到了通用的语法语义知识。第二，它可以将从开放领域学到的知识迁移到下游任务，以改善低资源任务，对低资源语言处理也非常有利。第三，预训练模型在几乎所有 NLP 任务中都取得了目前最佳的成果。最后，这个预训练模型+微调机制具备很好的可扩展性，在支持一个新任务时，只需要利用该任务的标注数据进行微调即可，一般工程师就可以实现。

预训练的意思就是提前已经给你一些初始化的参数，这个参数不是随机的，而是通过其他类似数据集上面学得的，然后再用你的数据集进行学习，得到适合你数据集的参数，随机初始化的话，的确不容易得到结果，但是这个结果是因为速度太慢，而不是最终的结果不一样

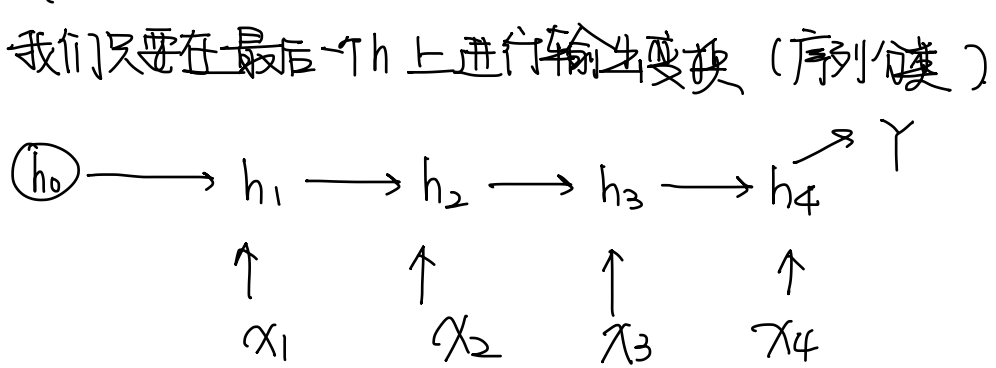
预训练就是先在大量通用语料上训练模型，学习到通用的语言知识，然后再针对性地针对任务进行迁移训练。以前常用的word2vec或者GloVe词向量，还有现在的BERT家族，都是典型的预训练语言模型。

预训练的方法就是使用巨量非标注语料库设计任务进行自监督学习——比如w2v的CBOW和BERT的MLM。这里需要注意的是，自监督学习也是监督式学习的一种，只不过标签来自于数据本身



RECAP

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

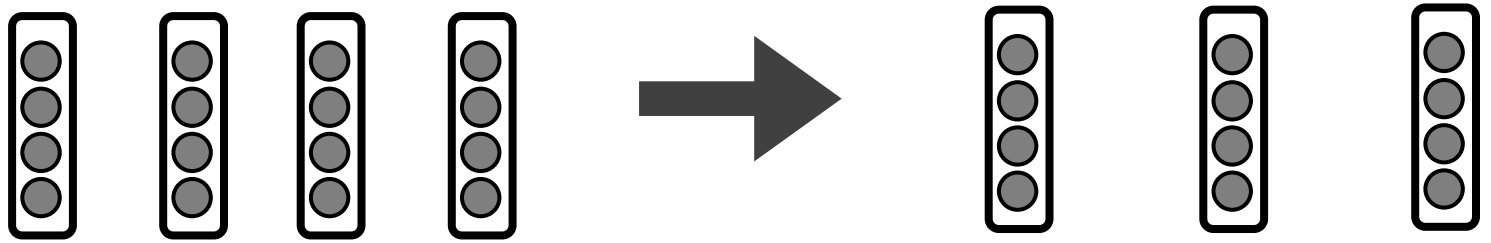


RECAP

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

We want to produce a **variable-length** output

(e.g., n → m predictions)



Thank you for visiting!

Děkujeme za návštěvu!

Outline

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

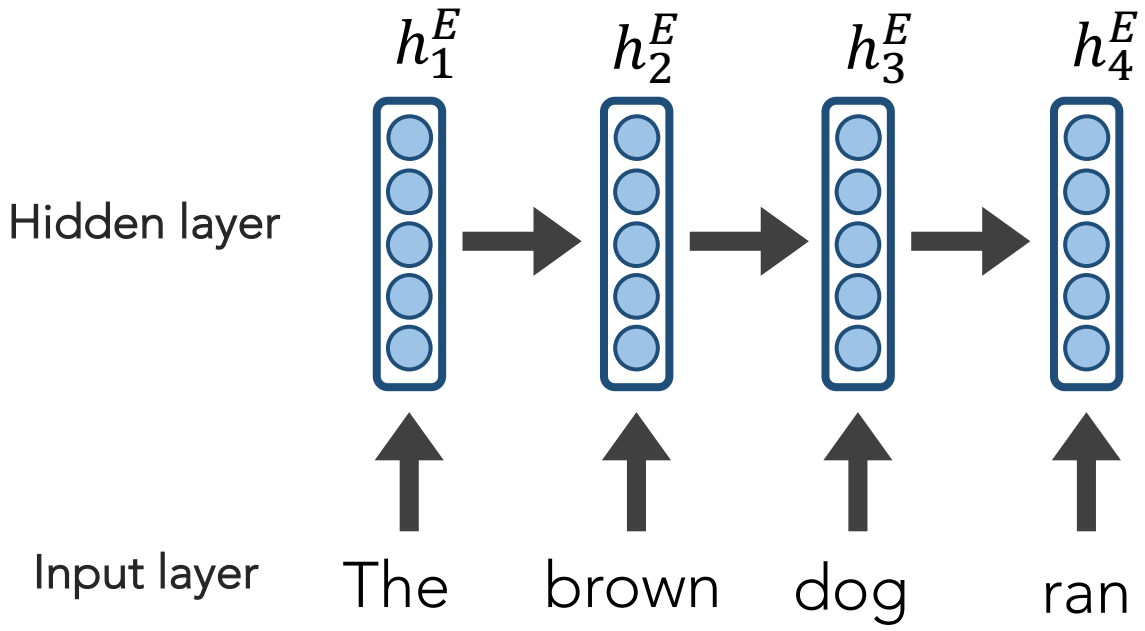
Outline

- How to use embeddings
- seq2seq 解决如何是 N vs M 的问题
- seq2seq + Attention
- Transformers (preview)

Sequence-to-Sequence (seq2seq)

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

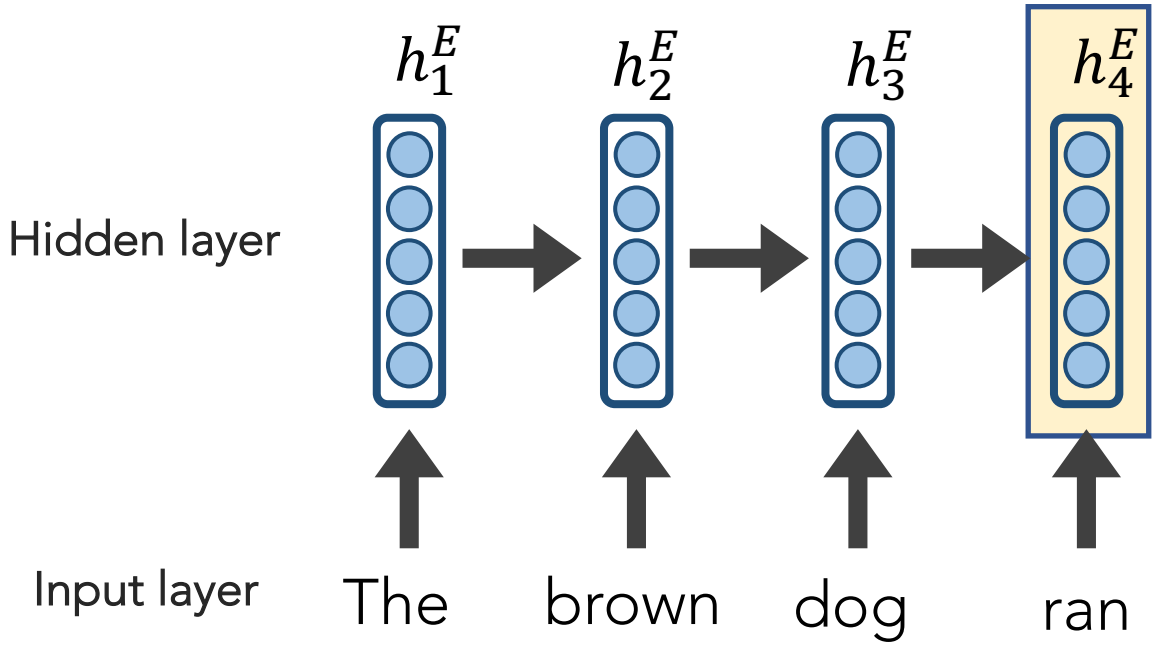
Sequence-to-Sequence (seq2seq)



ENCODER RNN

Sequence-to-Sequence (seq2seq)

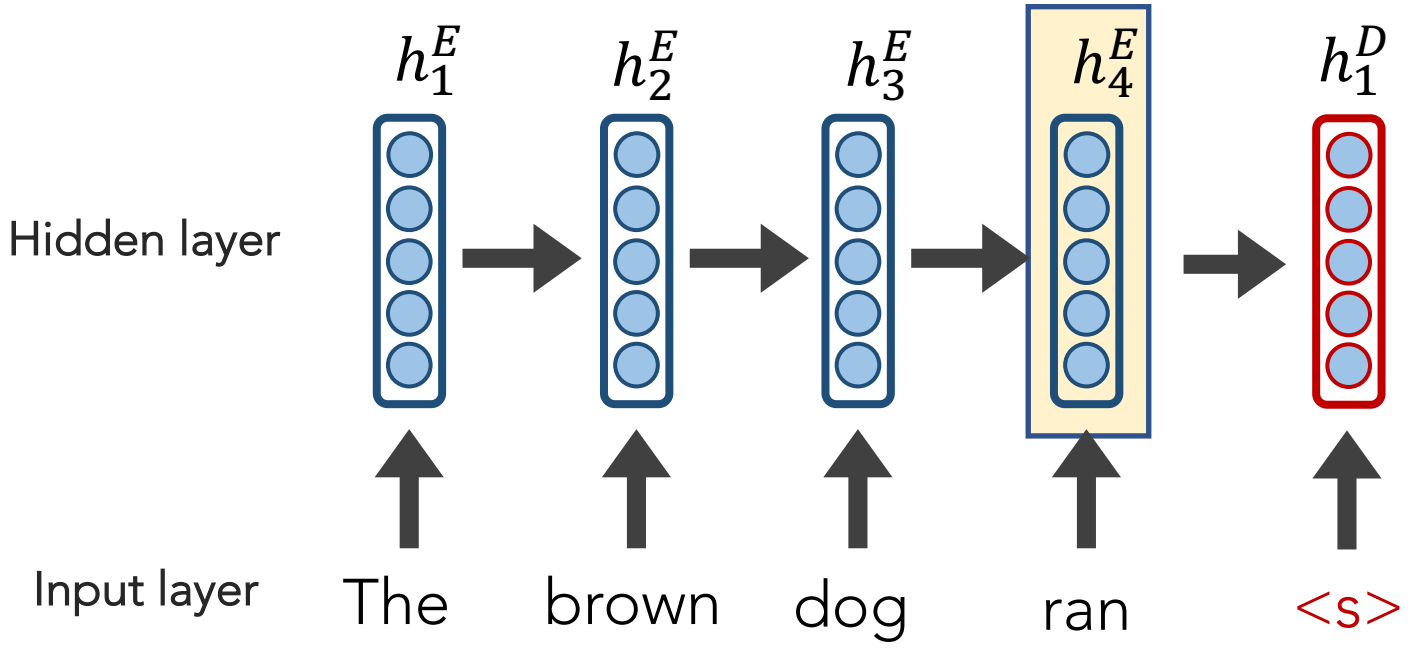
The final hidden state of the encoder RNN
is the initial state of the decoder RNN



ENCODER RNN

Sequence-to-Sequence (seq2seq)

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

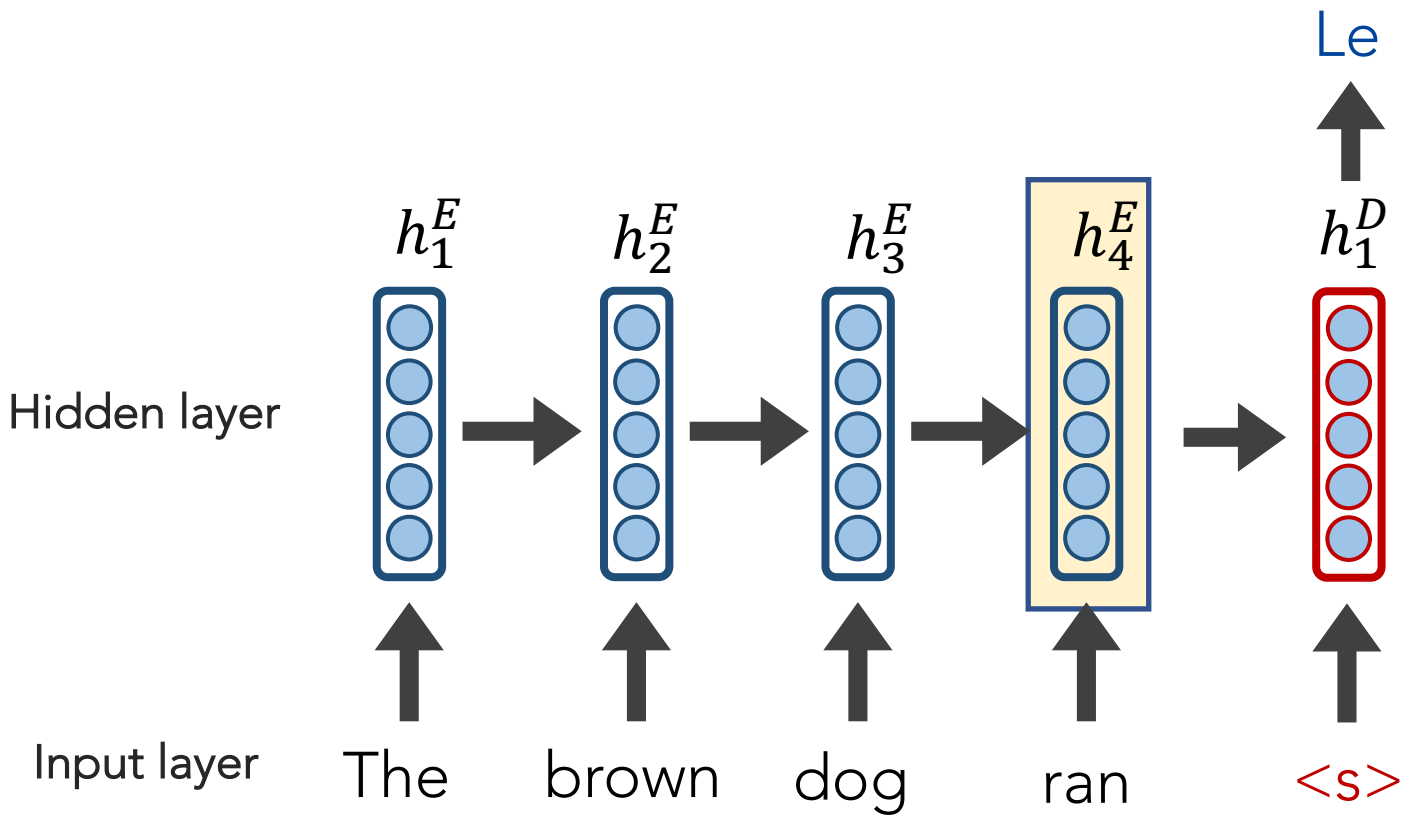


ENCODER RNN

DECODER RNN

Sequence-to-Sequence (seq2seq)

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

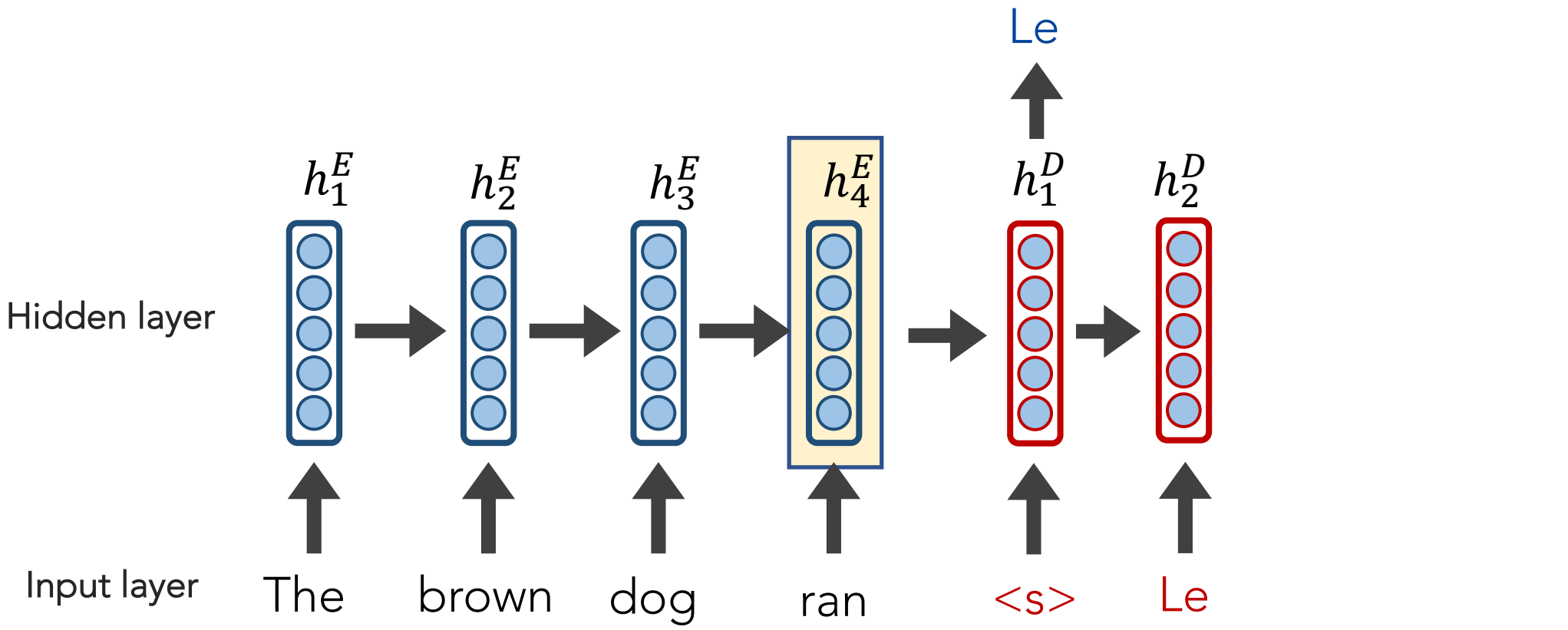


ENCODER RNN

DECODER RNN

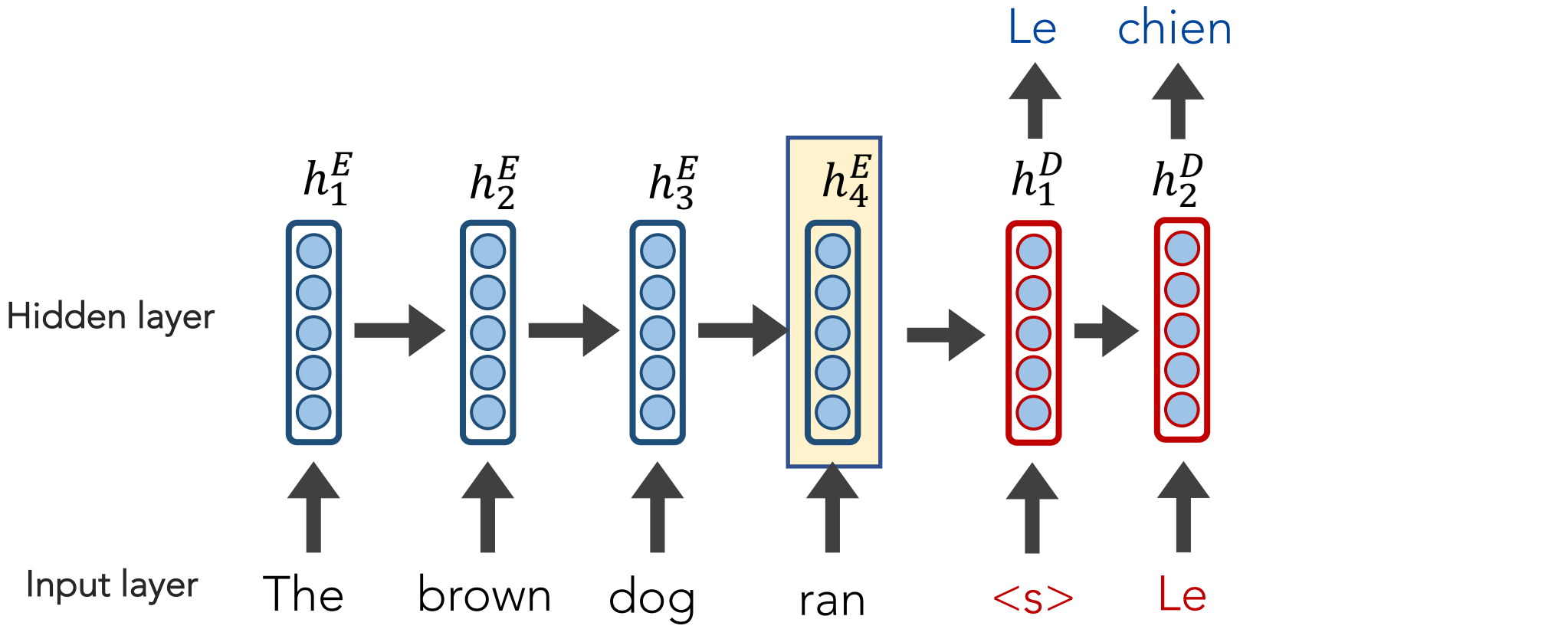
Sequence-to-Sequence (seq2seq)

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



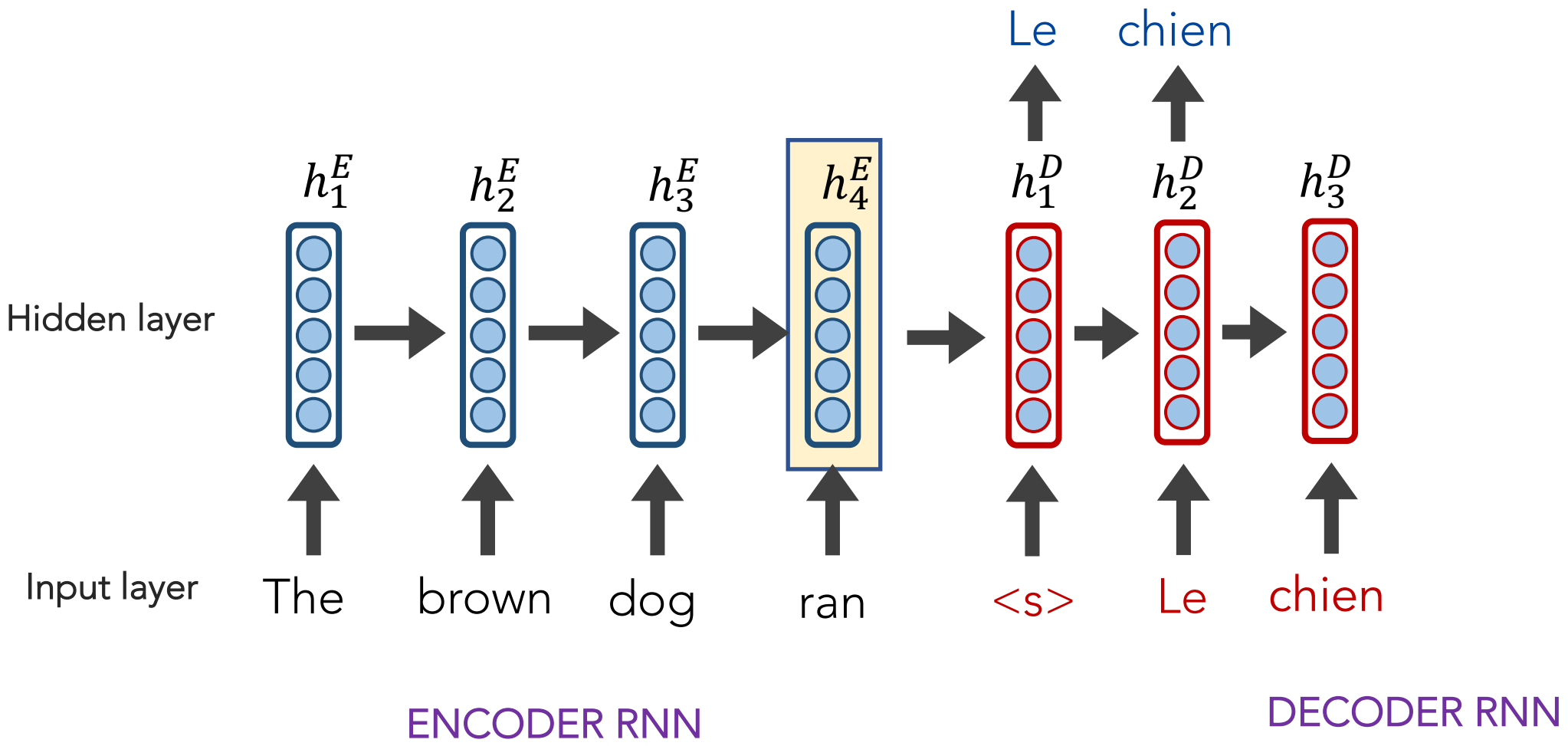
Sequence-to-Sequence (seq2seq)

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



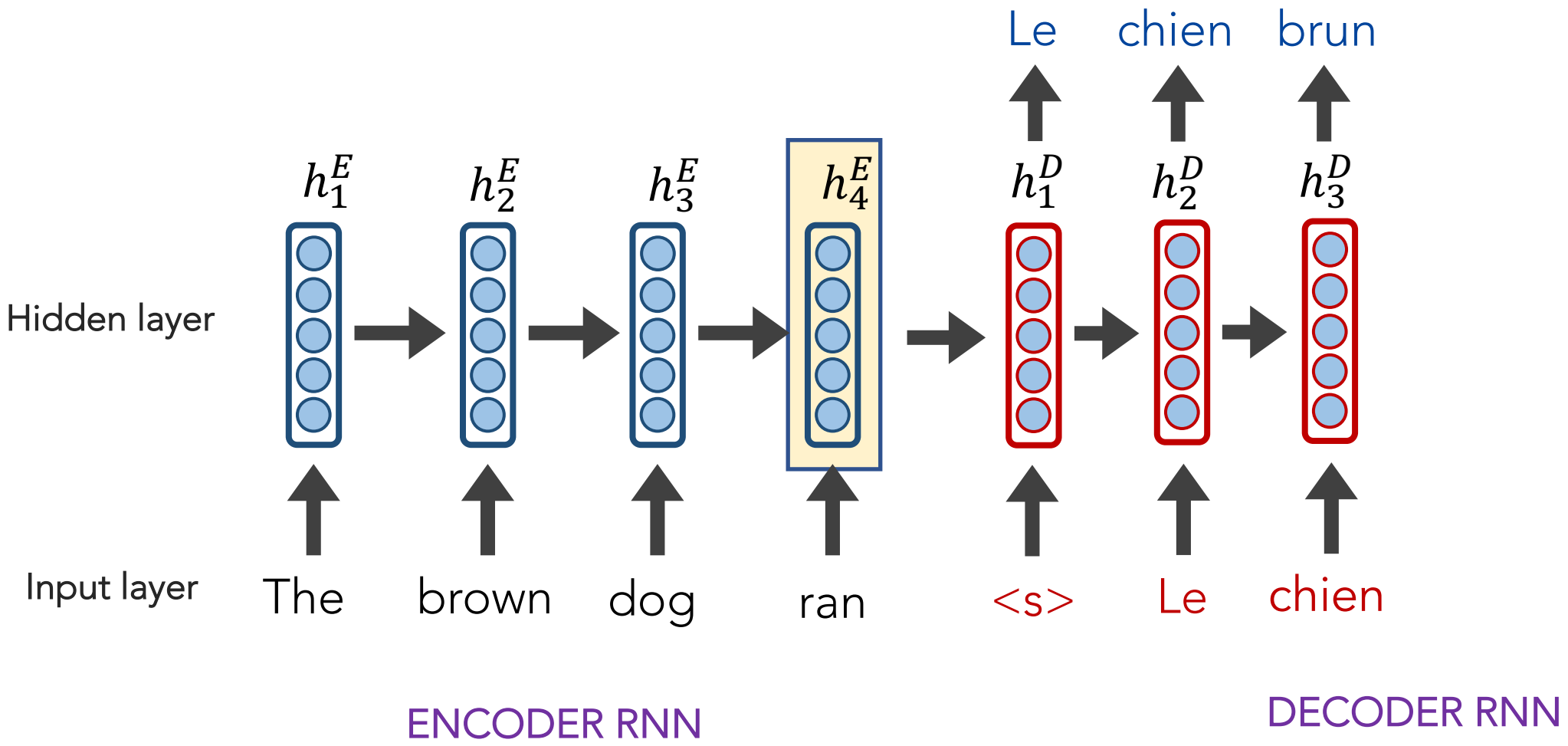
Sequence-to-Sequence (seq2seq)

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



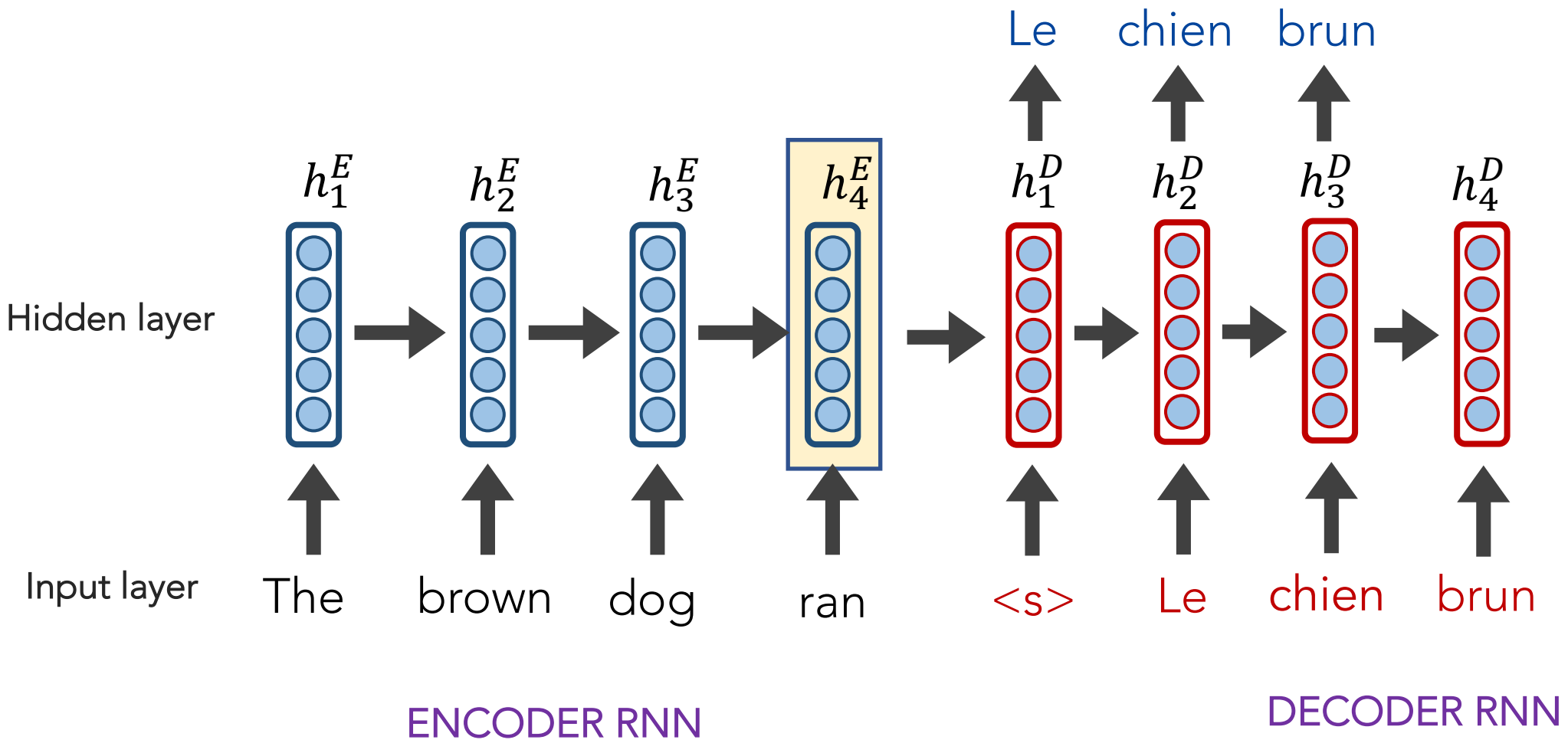
Sequence-to-Sequence (seq2seq)

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



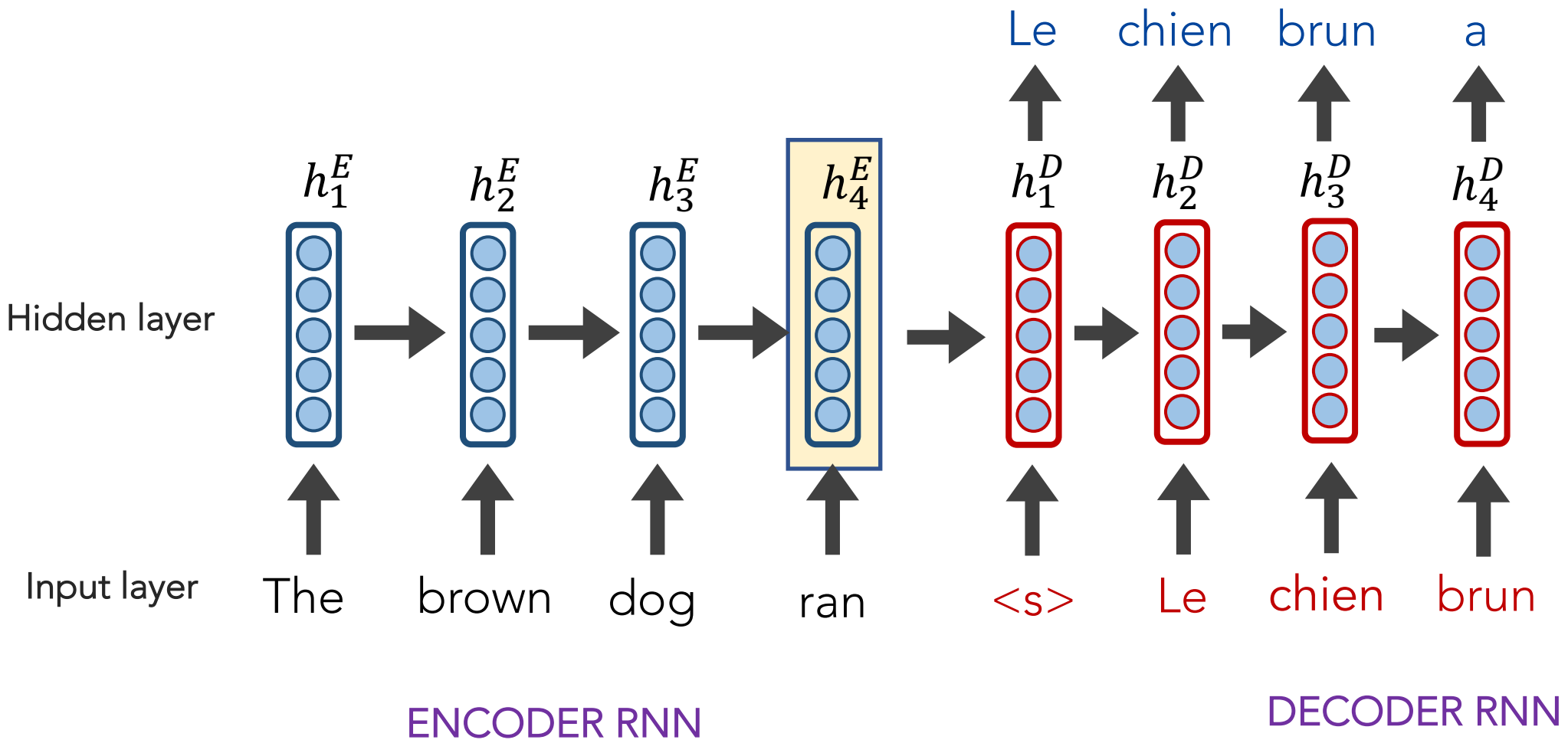
Sequence-to-Sequence (seq2seq)

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



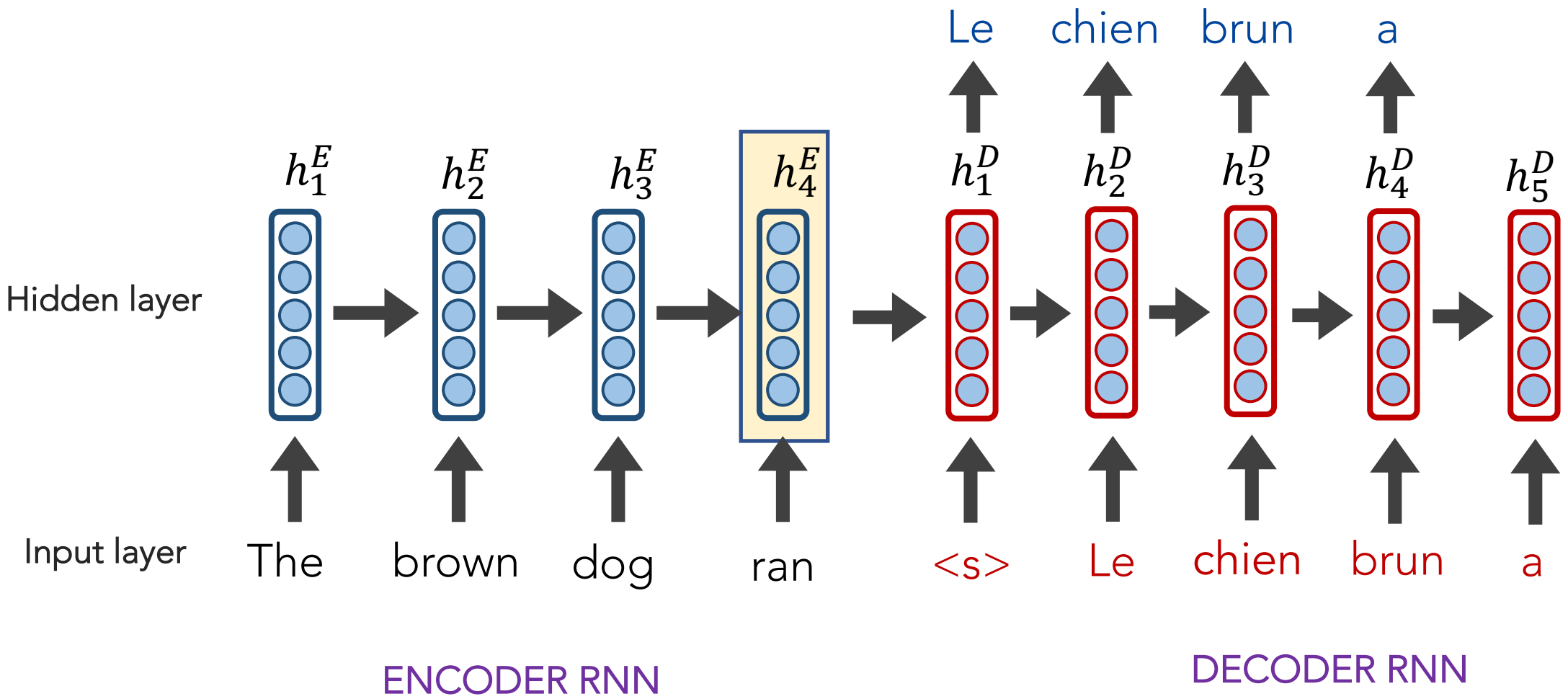
Sequence-to-Sequence (seq2seq)

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

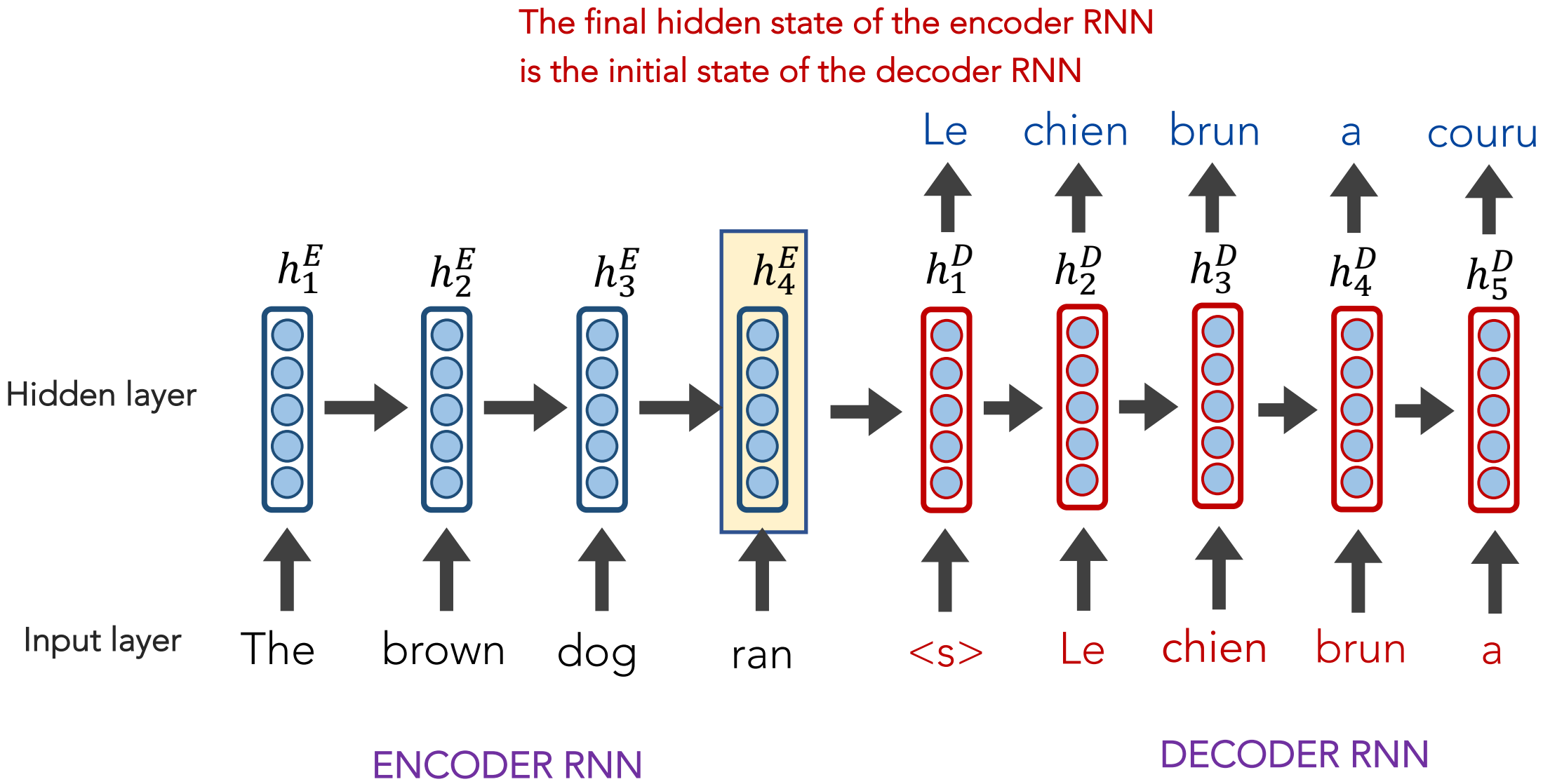


Sequence-to-Sequence (seq2seq)

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

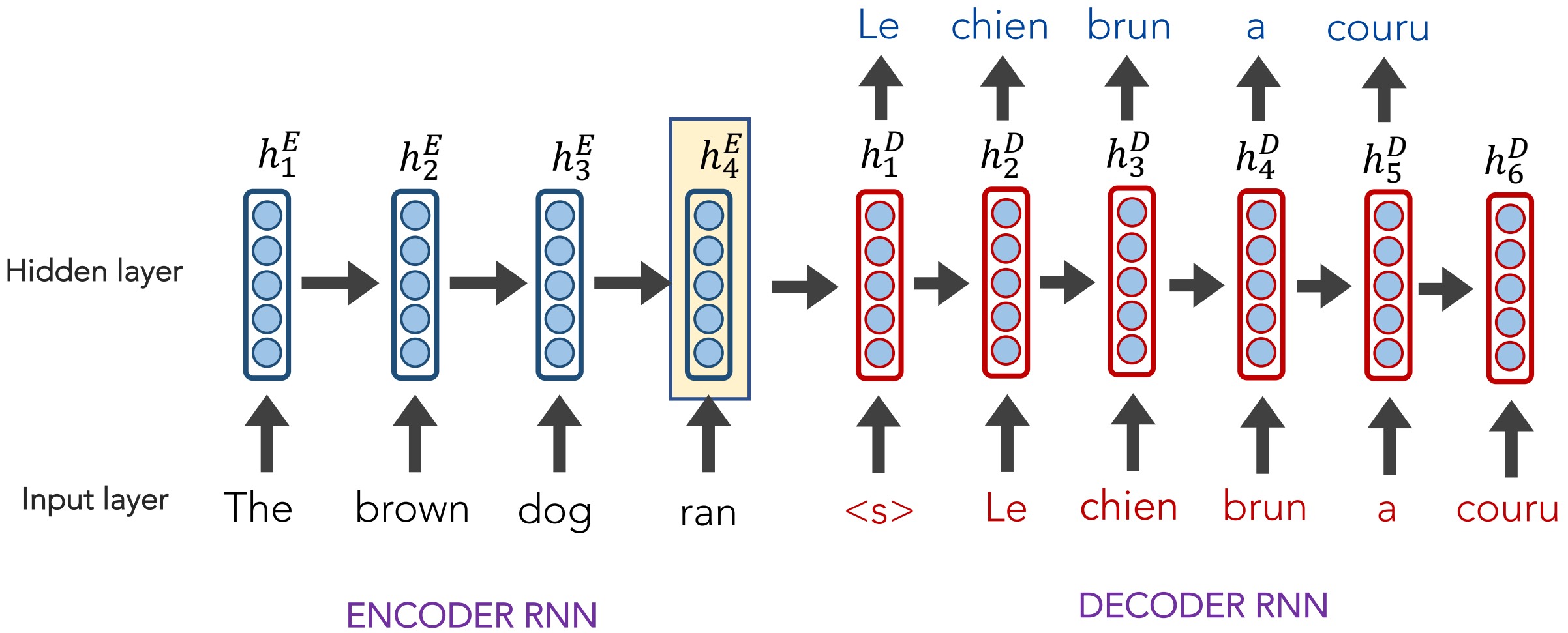


Sequence-to-Sequence (seq2seq)



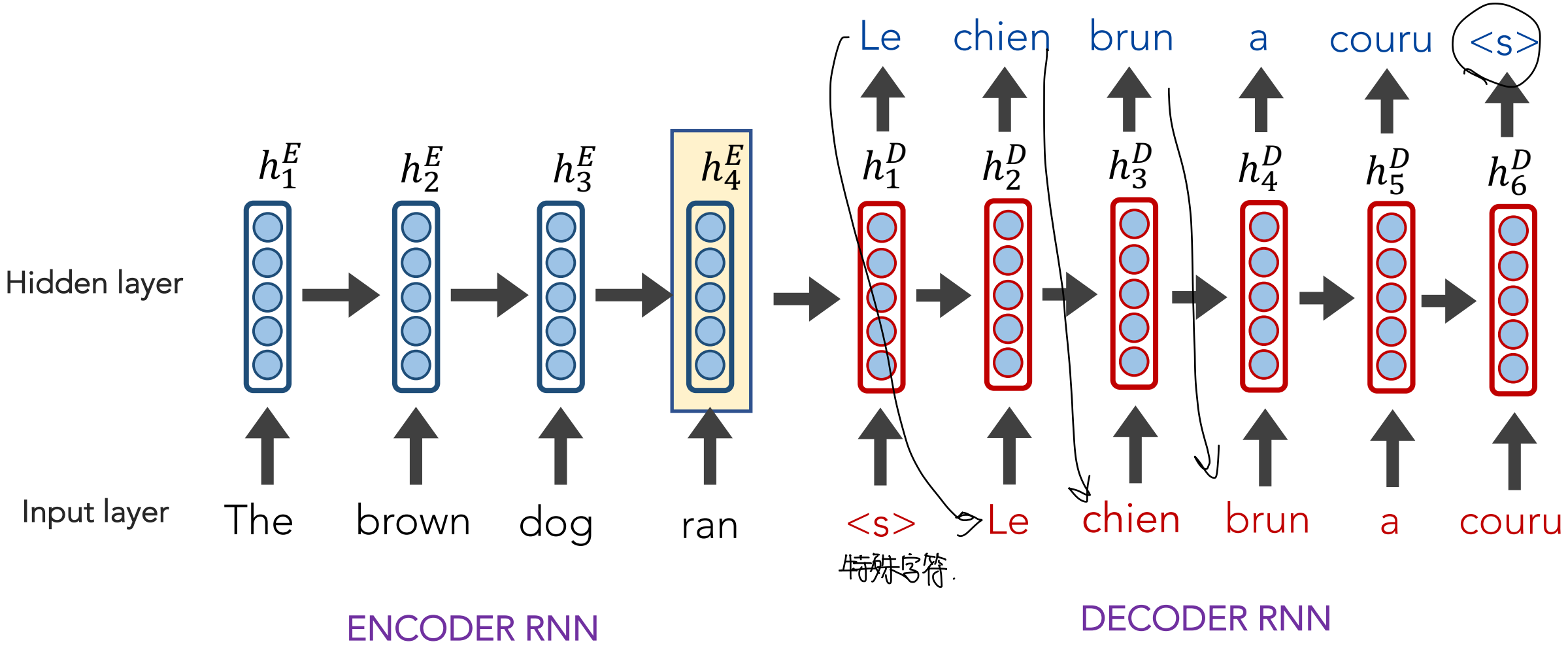
Sequence-to-Sequence (seq2seq)

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

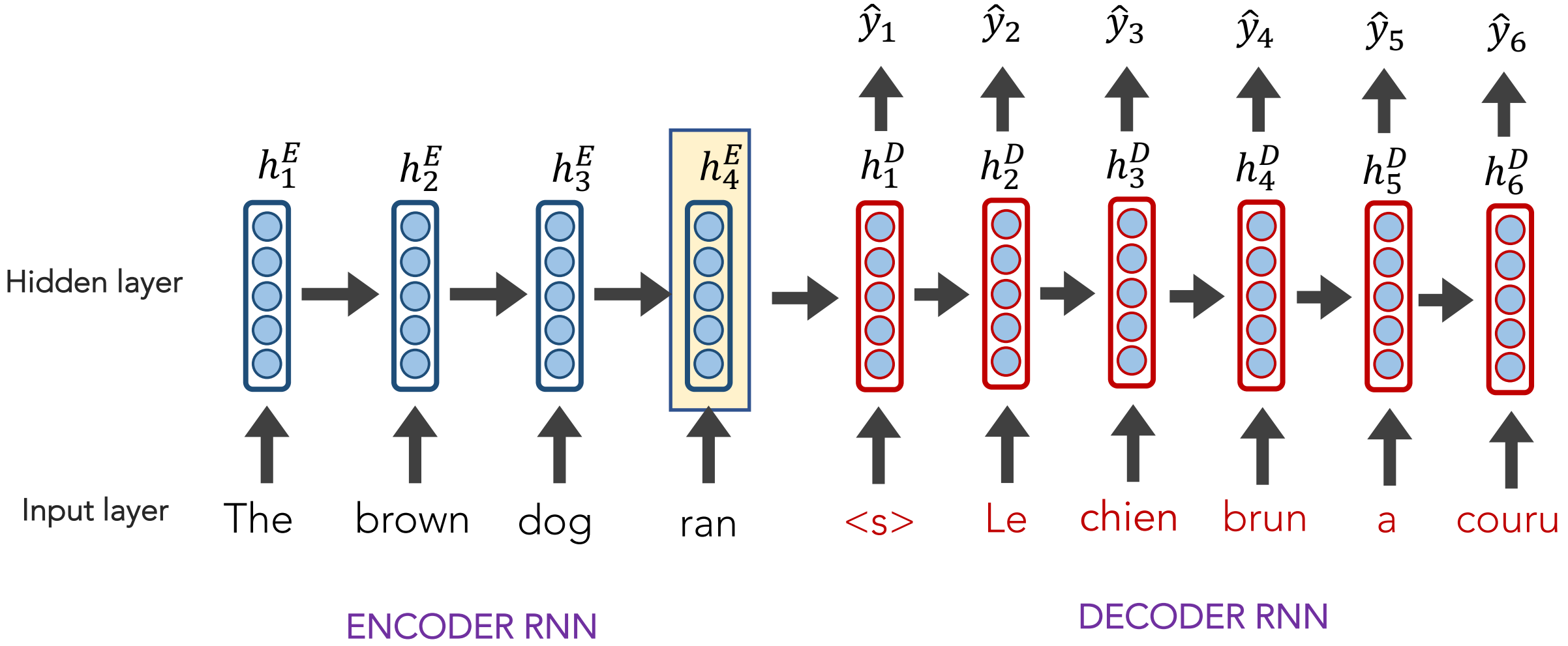


Sequence-to-Sequence (seq2seq)

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

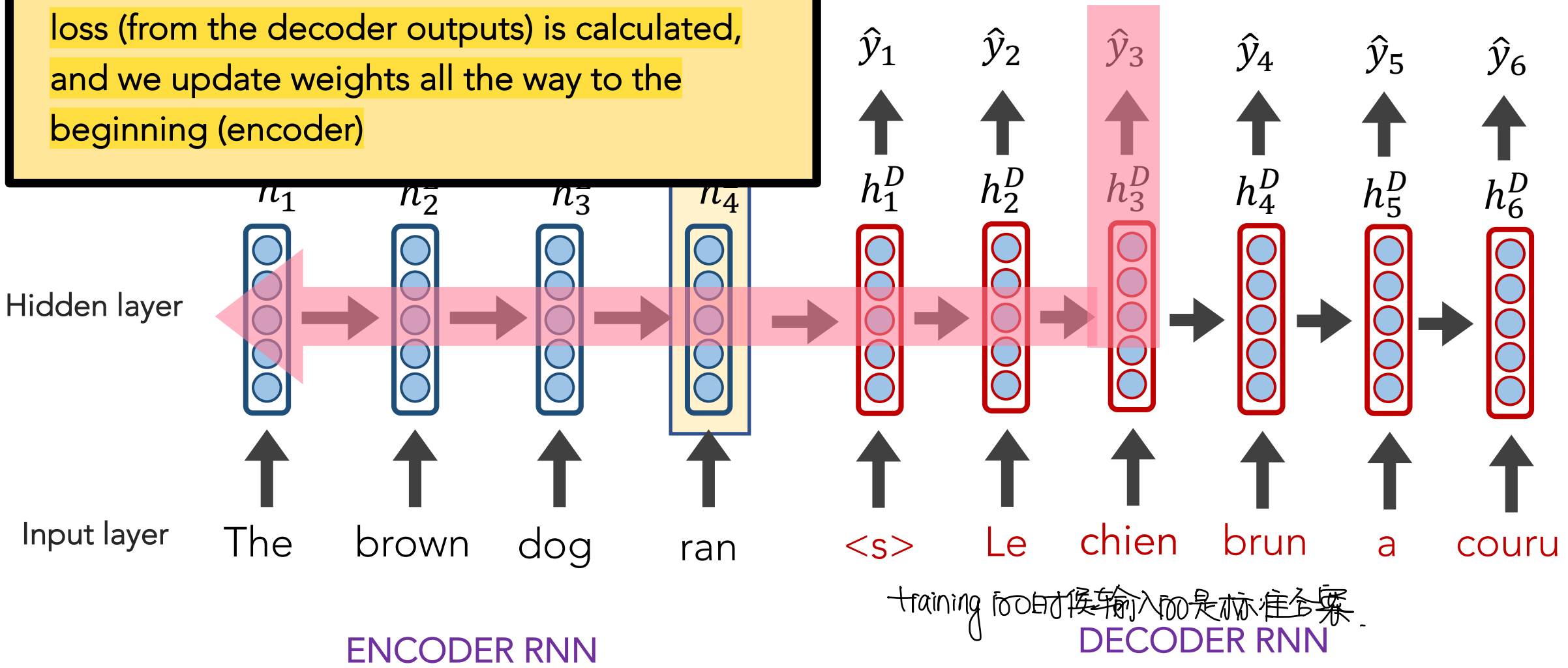


Sequence-to-Sequence (seq2seq)



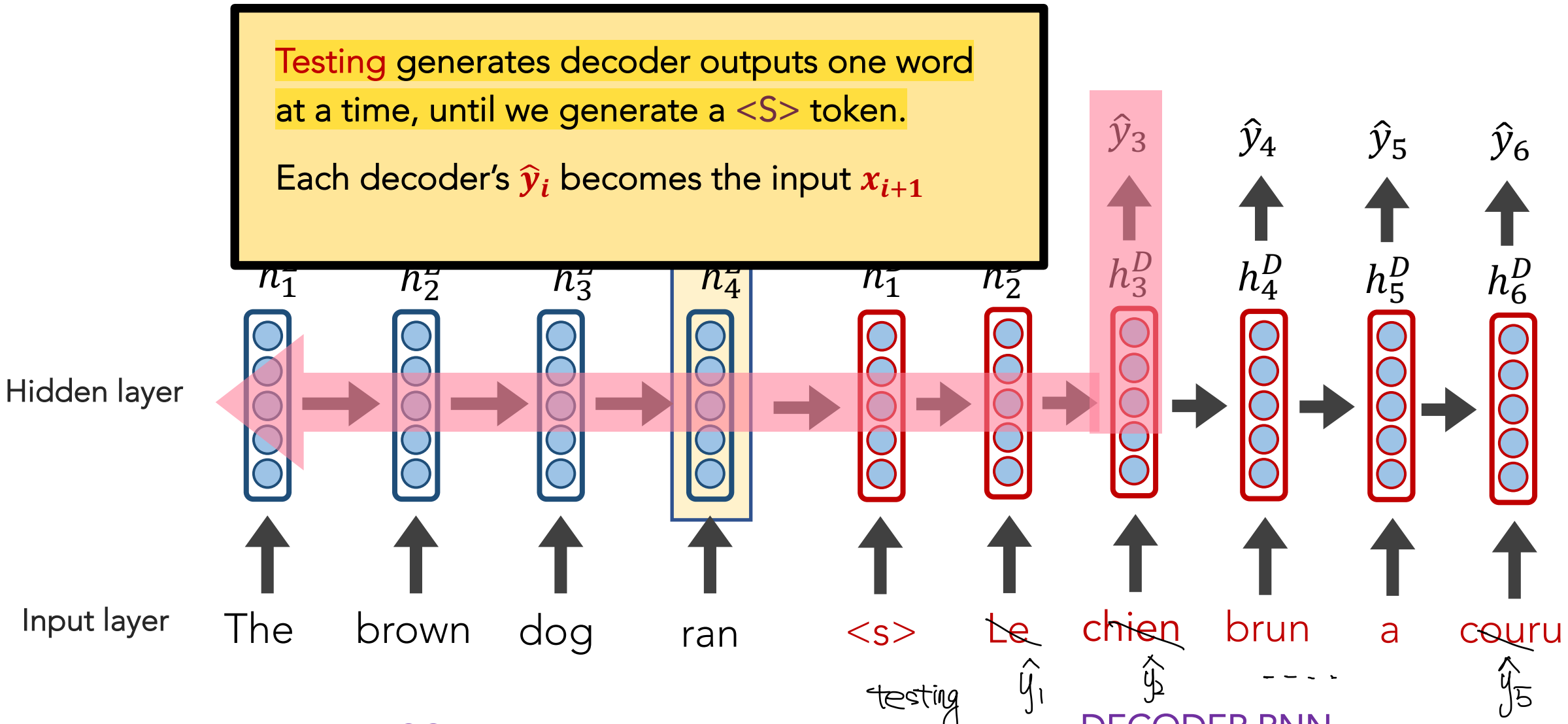
Sequence-to-Sequence (seq2seq)

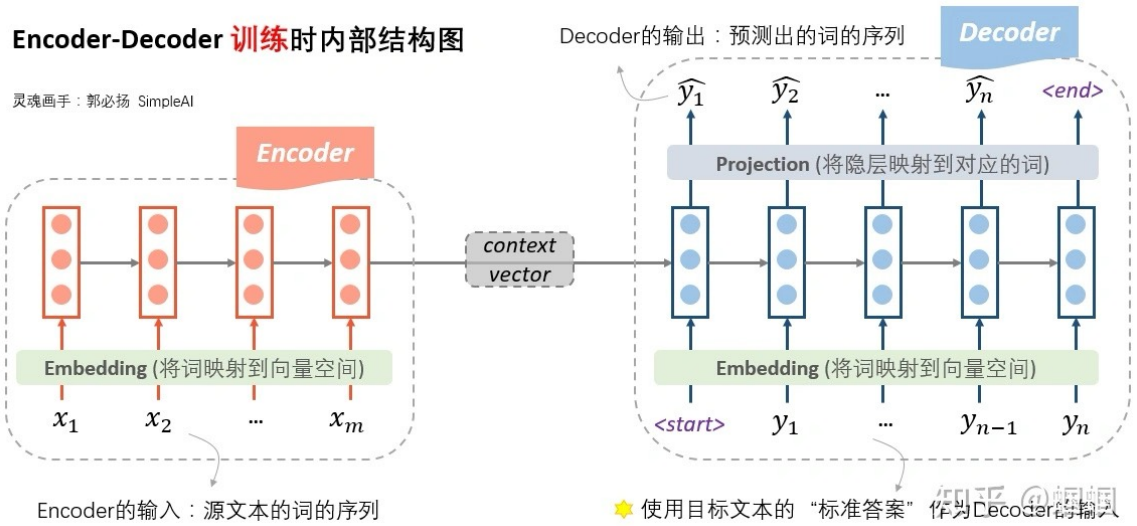
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)

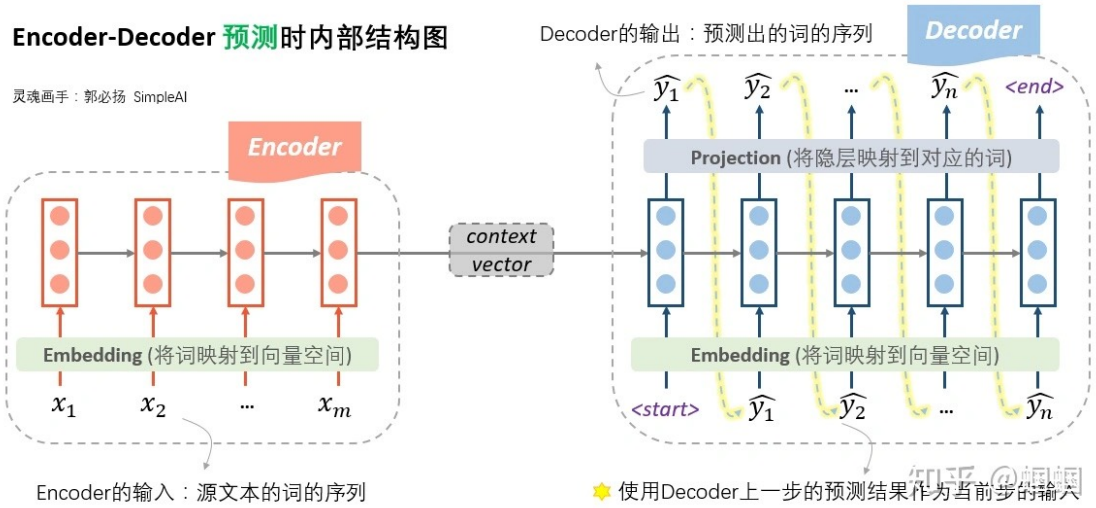


training 的时候输入的是标准答案

Sequence-to-Sequence (seq2seq)







这张图，展示了在「训练时」，seq2seq内部的详细结构。

在Encoder端，我们将source文本的词序列先经过embedding层转化成向量，然后输入到一个RNN结构（可以是普通RNN，LSTM，GRU等等）中。另外，这里的RNN也可以是多层、双向的。经过了RNN的一系列计算，最终隐层的输入，就作为源文本整体的一个表示向量，称为「context vector」。

Decoder端的操作就稍微复杂一些了。首先，Decoder的输入是什么呢？Decoder的输入，训练和测试时是不一样的！「在训练时，我们使用真实的目标文本，即“标准答案”作为输入」（注意第一步使用一个特殊的<start>字符，表示句子的开头）。每一步根据当前正确的输出词、上一步的隐状态来预测下一步的输出词。

预测时，Encoder端没什么变化，在Decoder端，由于此时没有所谓的“真实输出”或“标准答案”了，所以只能「自产自销：每一步的预测结果，都送给下一步作为输入」，直至输出<end>就结束。如果你对我之前写的笔记很熟悉的话，会发现，「这时的Decoder就是一个语言模型」。由于这个语言模型是根据context vector来进行文本的生成的，因此这种类型的语言模型，被称为“条件语言模型”：Conditional LM。正因为如此，在训练过程中，我们可以使用一些预训练好的语言模型来对Decoder的参数进行初始化，从而加快迭代过程。

我们称这两种模式，根据标准答案来decode的方式为「teacher forcing」，而根据上一步的输出作为下一步输入的decode方式为「free running」。

其实，free running的模式真的不能在训练时使用吗？——当然是可以的！从理论上没有任何的问题，又不是不能跑。但是，在实践中人们发现，这样训练太南了。因为没有任何的引导，一开始会完全是瞎预测，正所谓“一步错，步步错”，而且越错越离谱，这样会导致训练时的累积损失太大（「误差爆炸」问题，exposure bias），训练起来就很费劲。这个时候，如果我们能够在每一步的预测时，让老师来指导一下，即提示一下上一个词的正确答案，decoder就可以快速步入正轨，训练过程也可以更快收敛。因此大家把这种方法称为teacher forcing。所以，这种操作的目的就是为了使得训练过程更容易。

Scheduled sampling

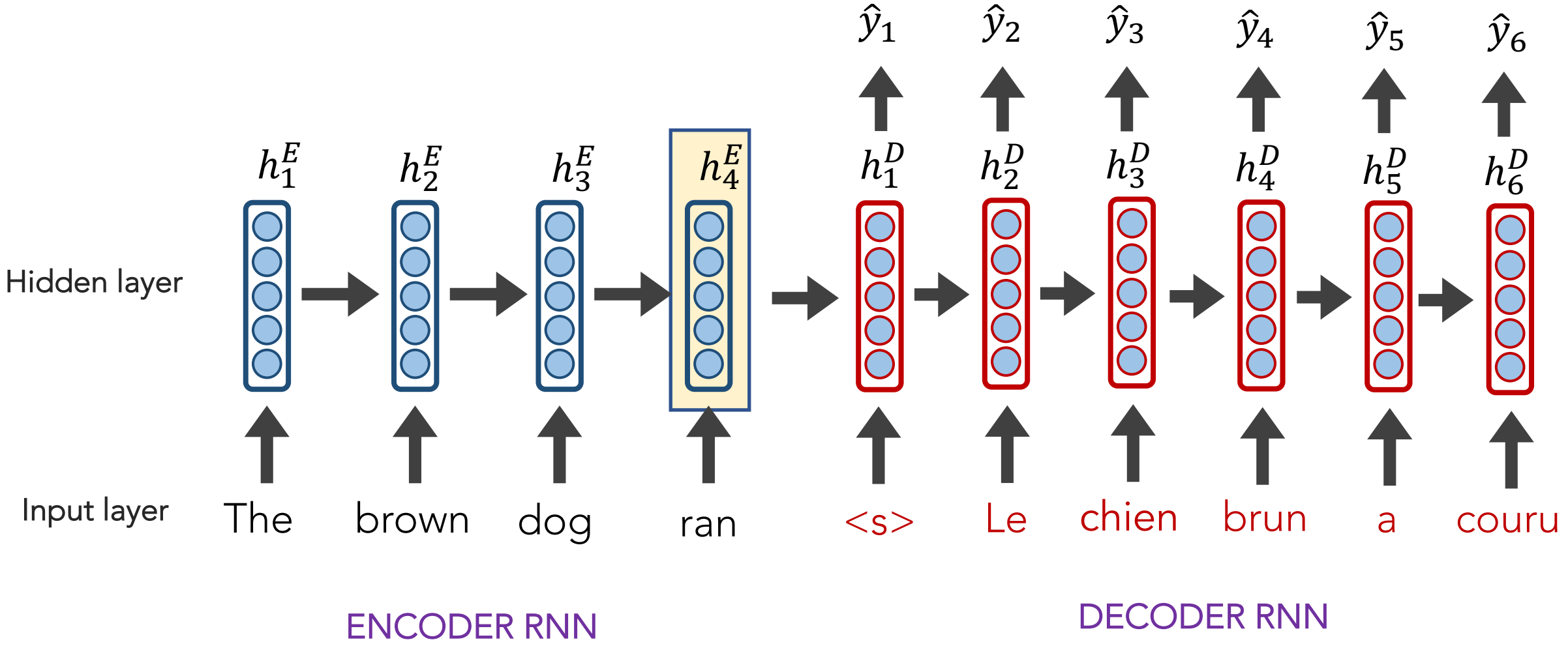
Sequence-to-Sequence (seq2seq)

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

在Encoder-Decoder结构中，Encoder把所有的输入序列都编码成一个统一的语义特征c再解码，因此， c中必须包含原始序列中的所有信息，它的长度就成了限制模型性能的瓶颈。如机器翻译问题，当要翻译的句子较长时，一个c可能存不下那么多信息，就会造成翻译精度的下降。

Attention机制通过在每个时间输入不同的c来解决这个问题，下图是带有Attention机制的Decoder

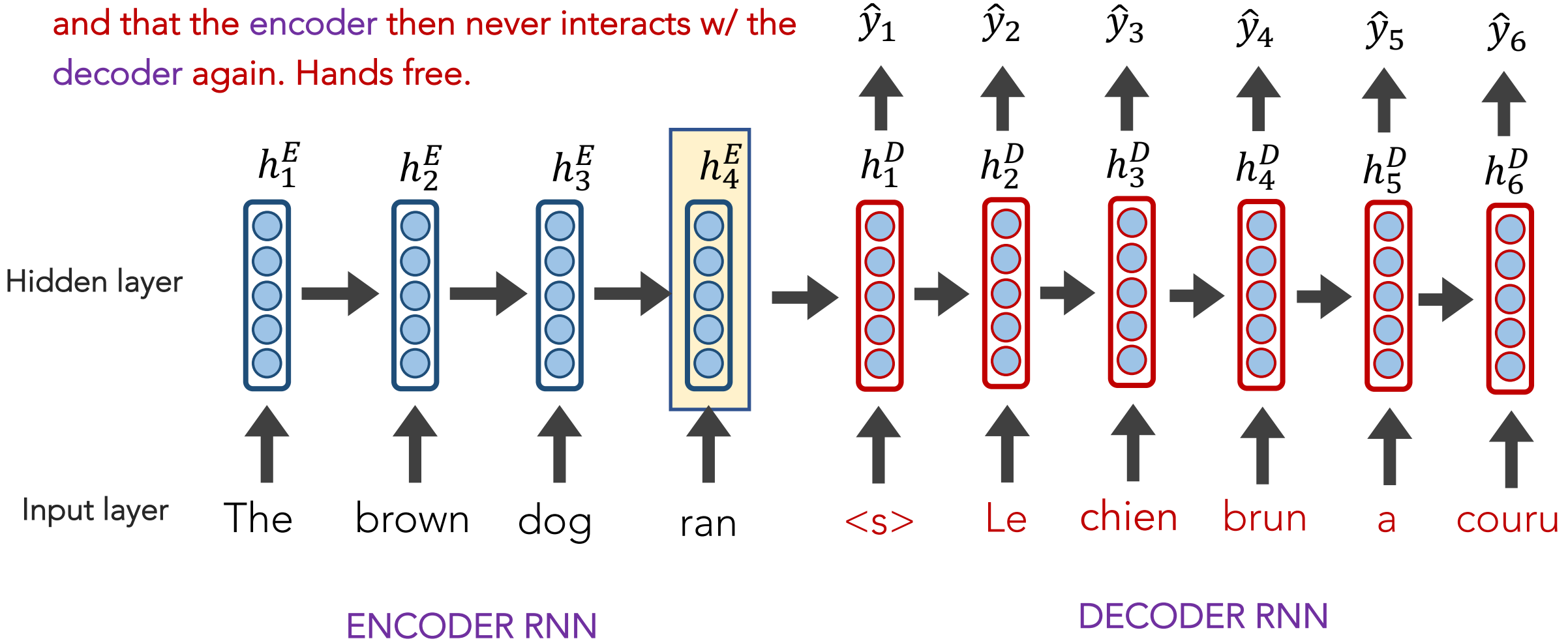
Sequence-to-Sequence (seq2seq)



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.

⇒ not much more 256 dimensions



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 便反哺 while focusing on different parts.

Outline

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

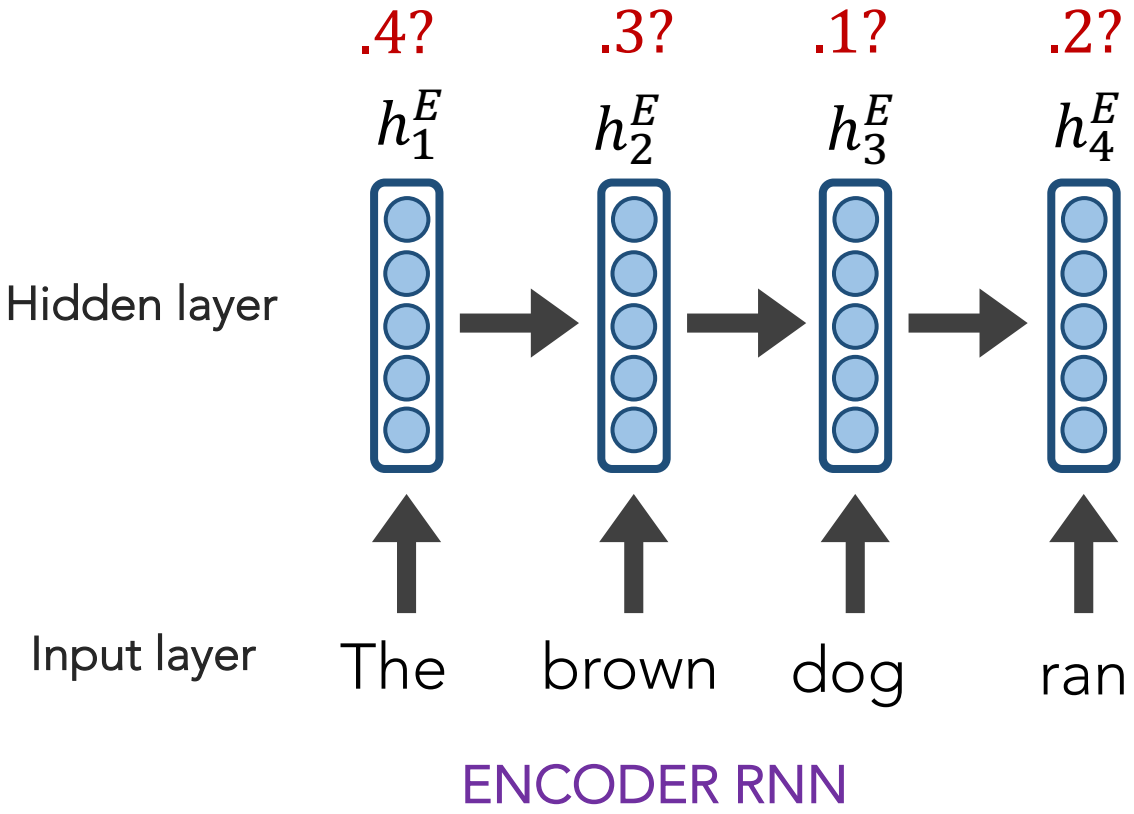
Outline

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

Attention模型中，当我们翻译当前词语时，我们会寻找源语句中相对应的几个词语，并结合之前的已经翻译的部分作出相应的翻译，如下图所示，当我们翻译“knowledge”时，只需将注意力放在源句中“知识”的部分，当翻译“power”时，只需将注意力集中在"力量“。这样，当我们decoder预测目标翻译的时候就可以看到encoder的所有信息，而不仅局限于原来模型中定长的隐藏向量，并且不会丧失长程的信息。

seq2seq + Attention

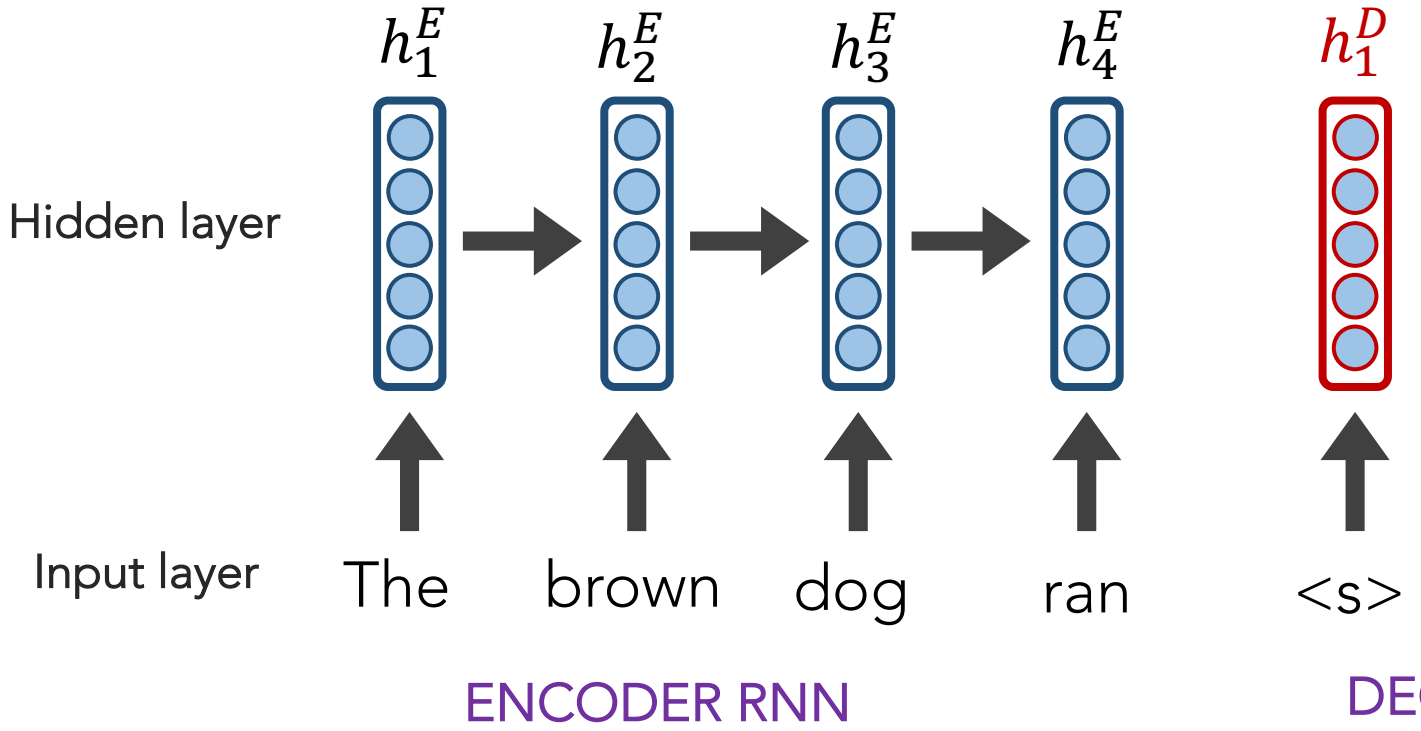
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



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!



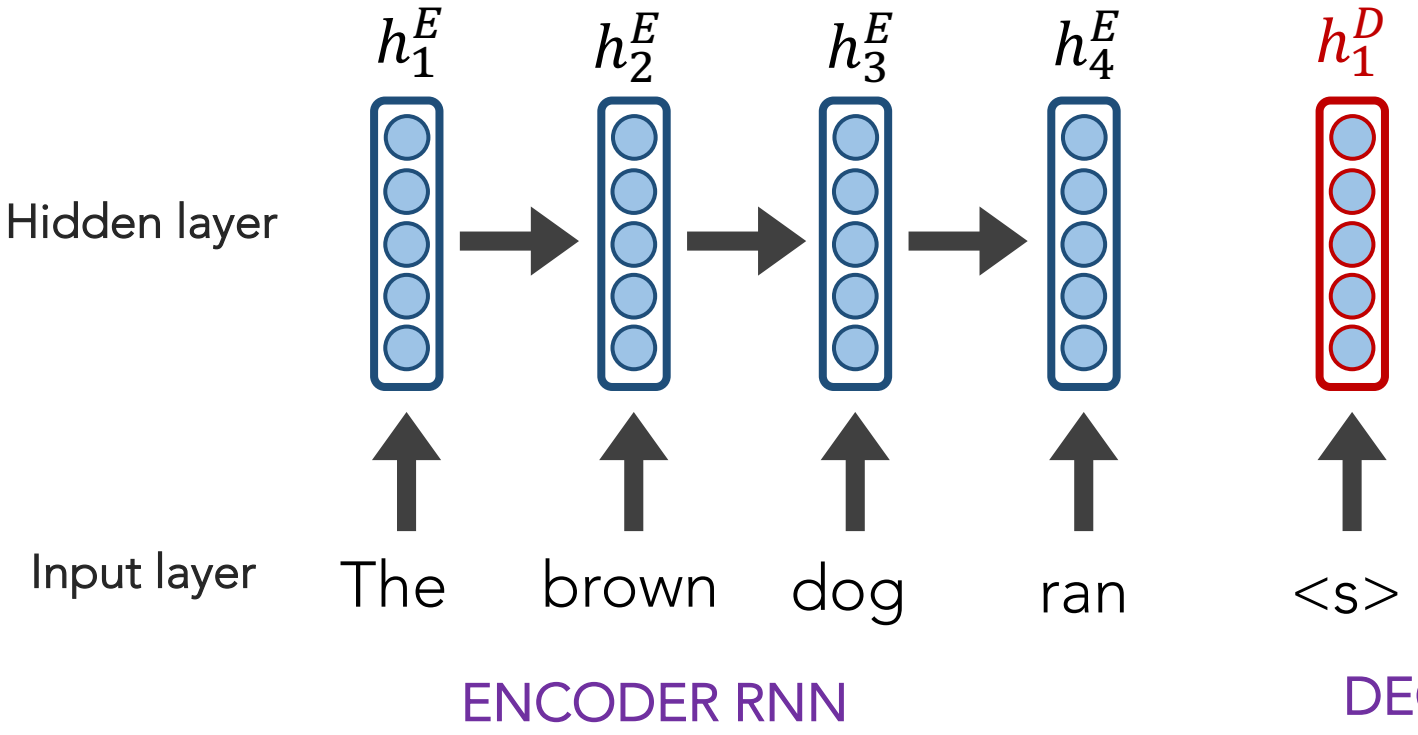
这里关键的操作是计算encoder与decoder state之间的关联性的权重，得到Attention分布，从而对于当前输出位置得到比较重要的输入位置的权重，在预测输出时相应的会占较大的比重。

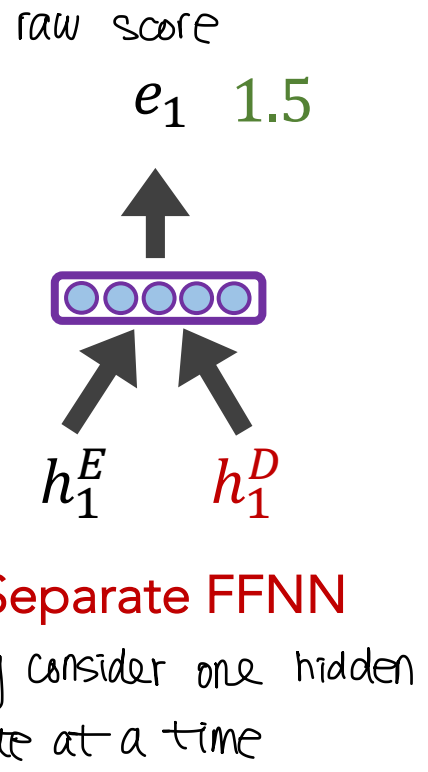
通过Attention机制的引入，我们打破了只能利用encoder最终单一向量结果的限制，从而使模型可以集中在所有对于下一个目标单词重要的输入信息上，使模型效果得到极大的改善。还有一个优点是，我们通过观察attention 权重矩阵的变化，可以更好地知道哪部分翻译对应哪部分源文字，有助于更好的理解模型工作机制

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!

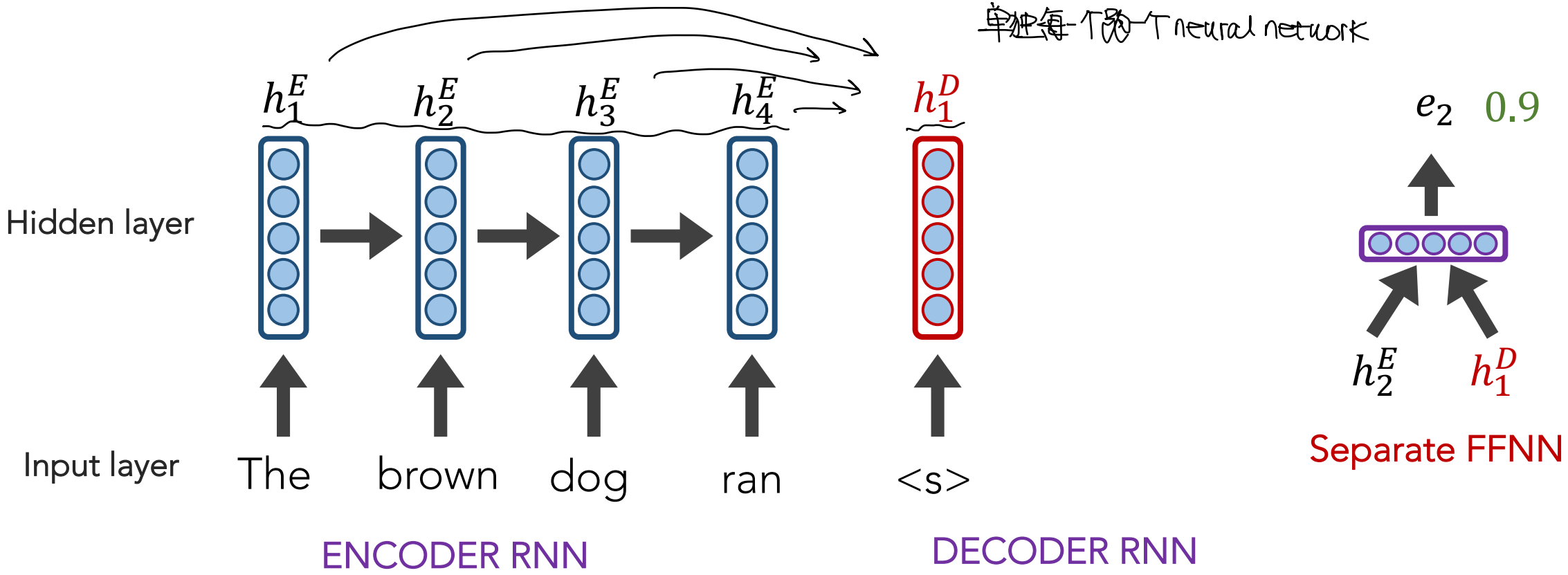




seq2seq + Attention

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

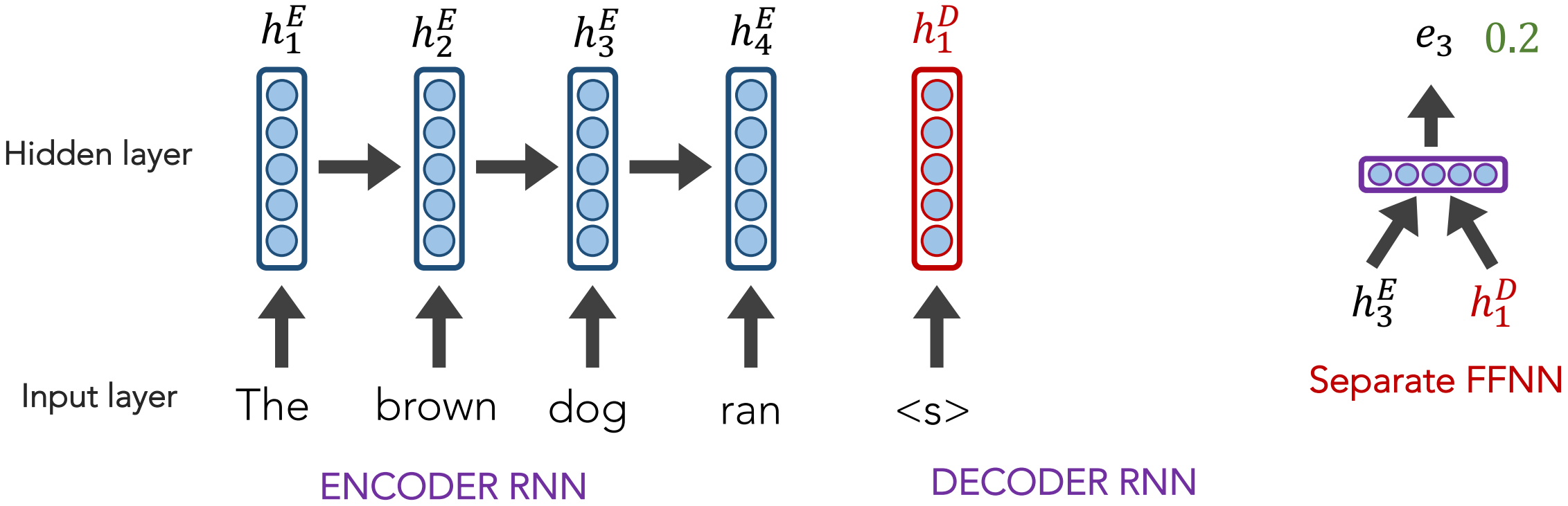
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

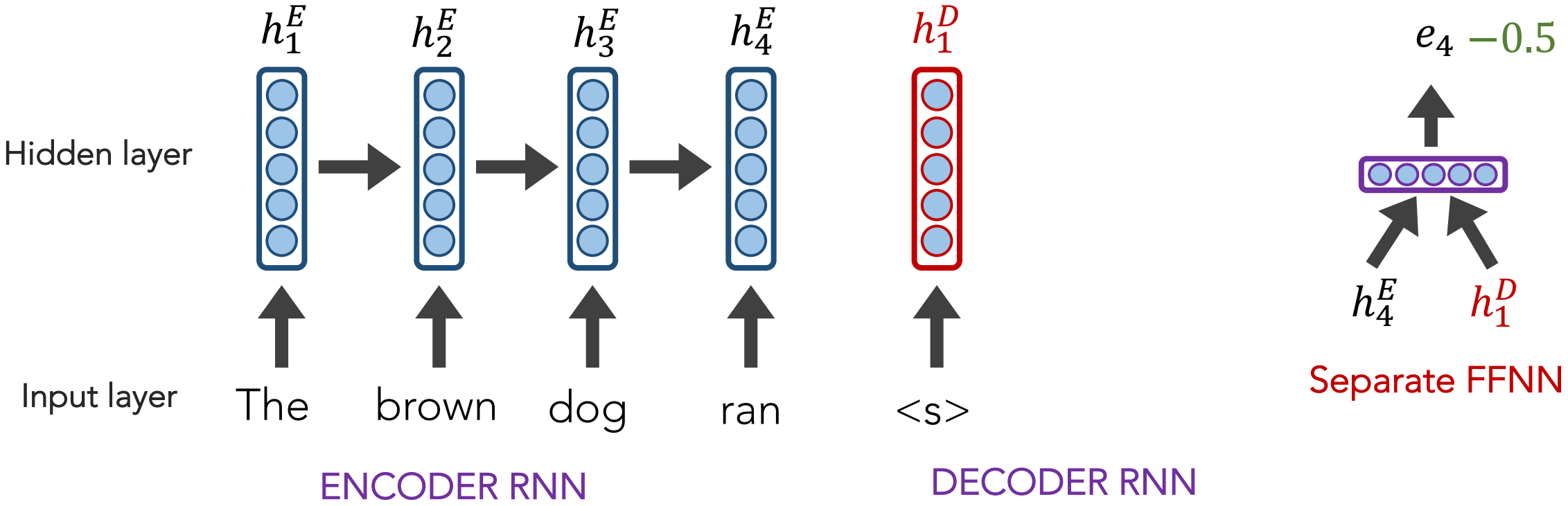
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

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

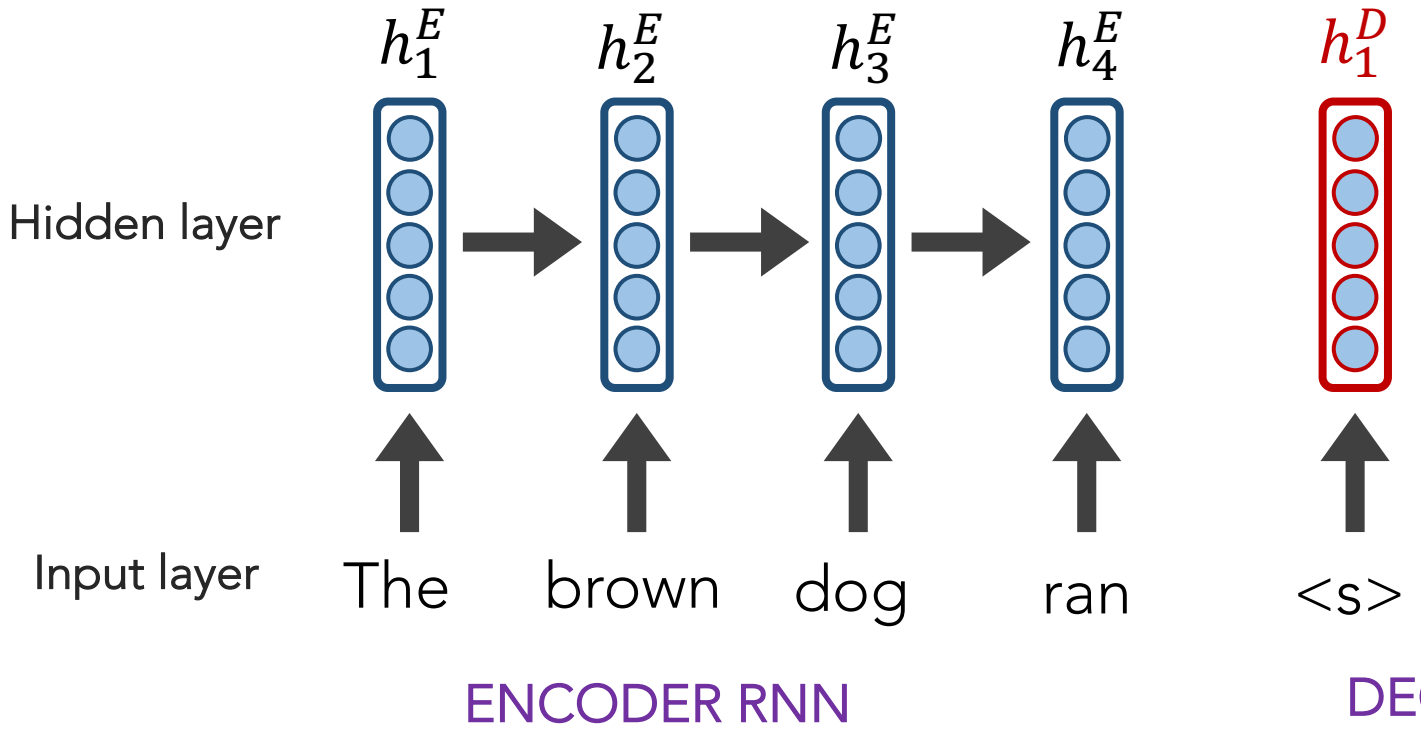
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



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)

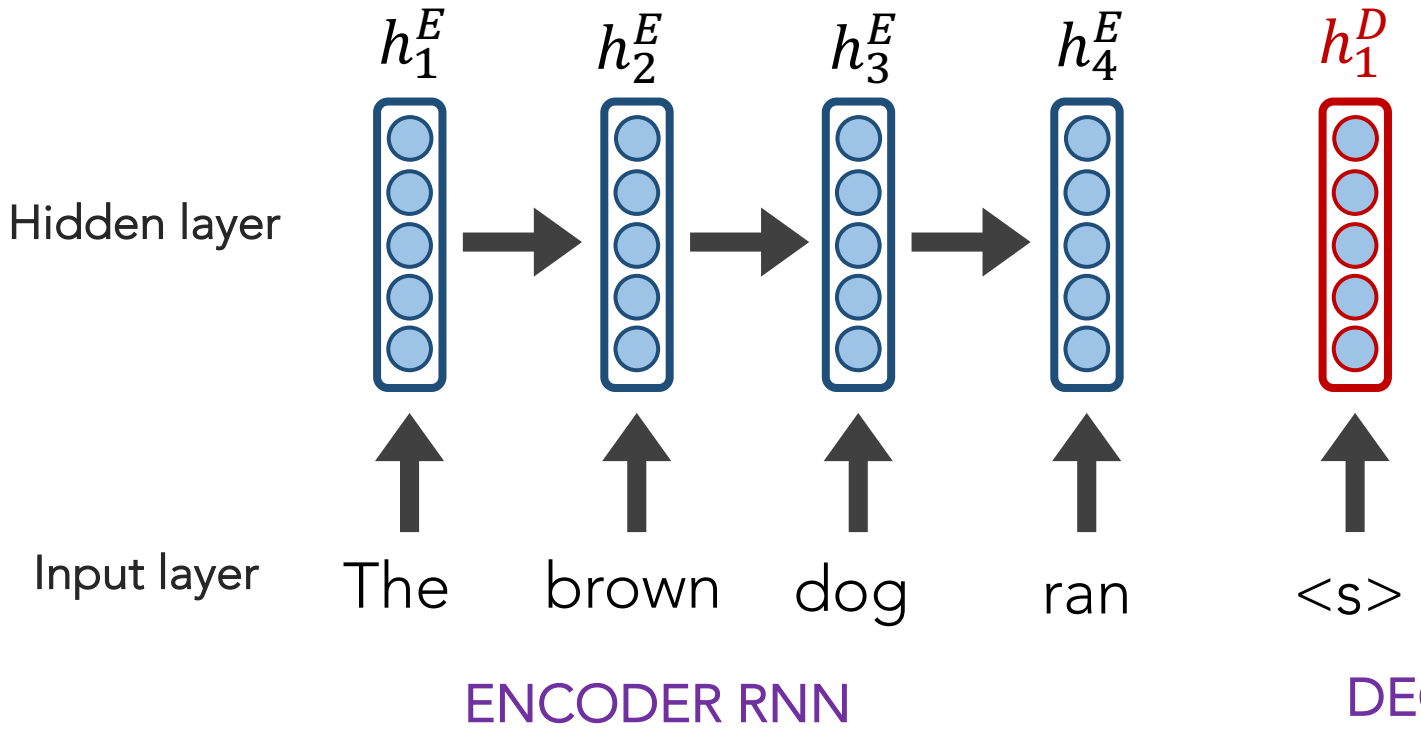
e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

no guarantee that the scores sum to 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

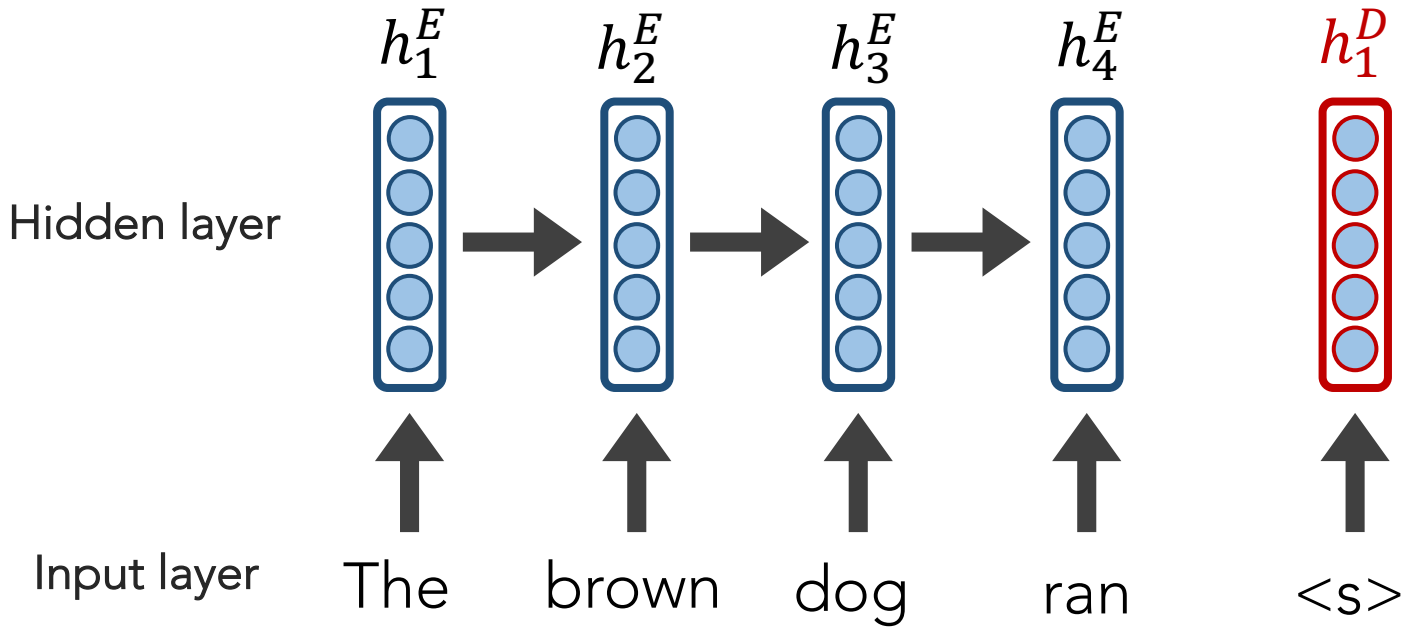
Attention (softmax'd)

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

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

sum to 1

Attention (softmax'd)

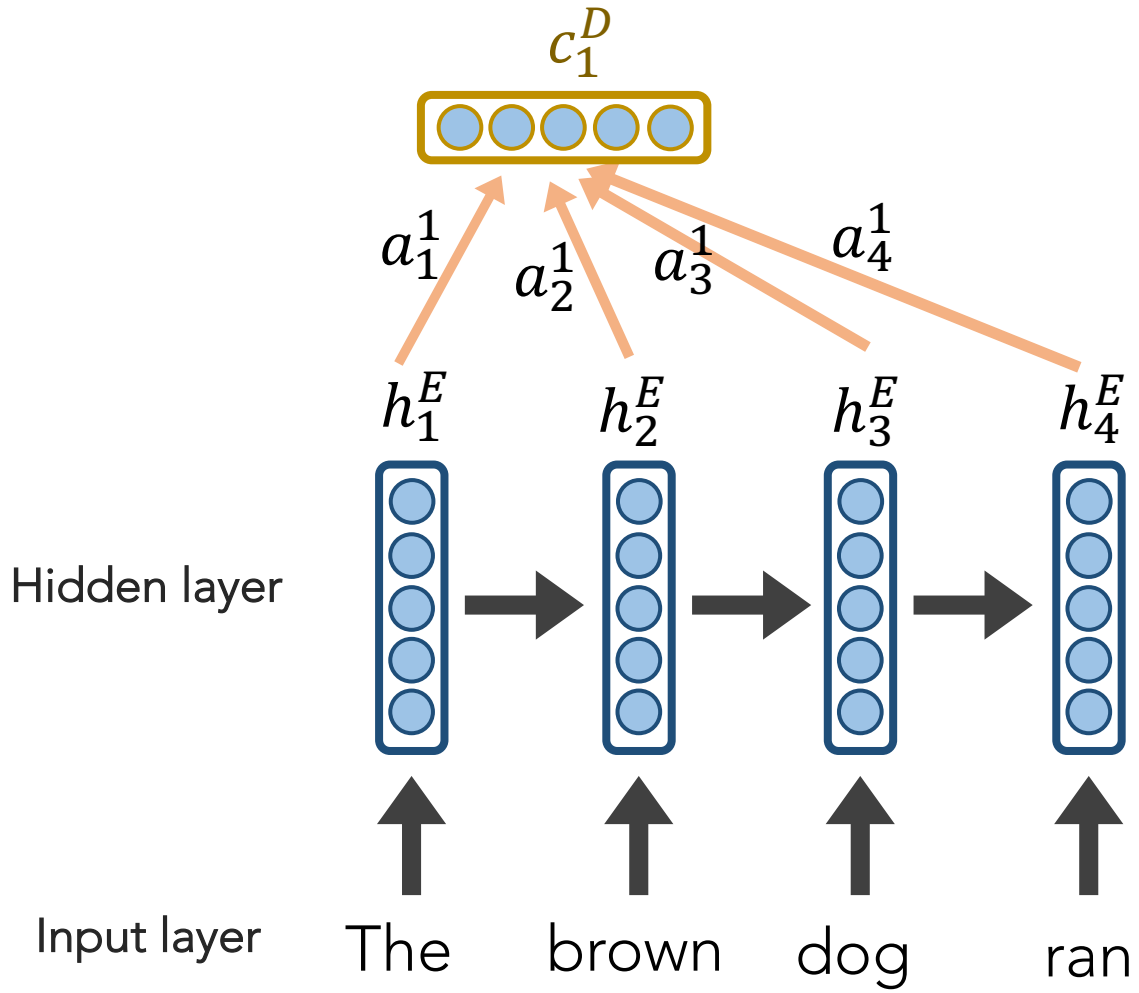
$a_1^1 = 0.51$

$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

seq2seq + Attention



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

Attention (softmax'd)

$a_1^1 = 0.51$

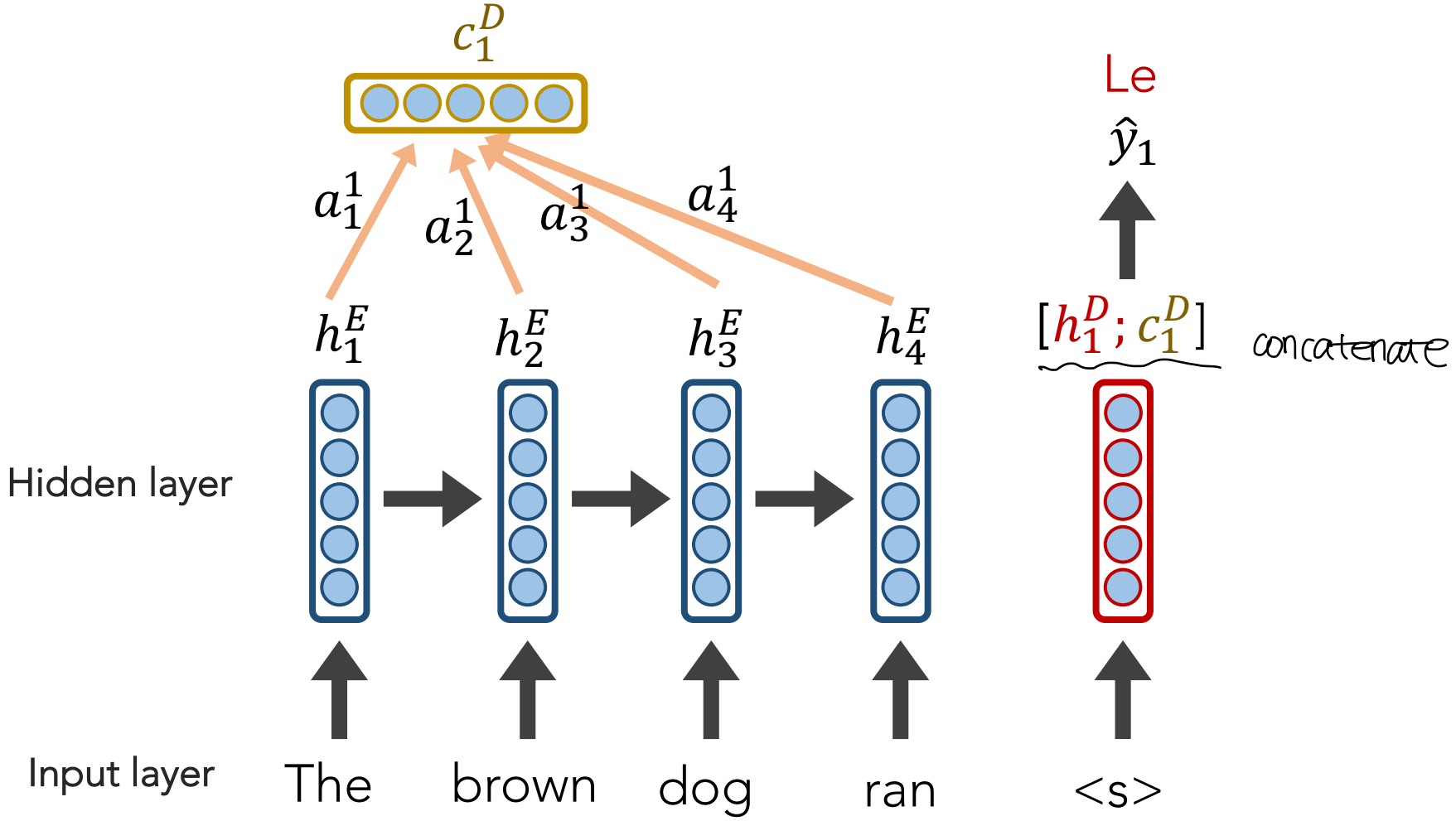
$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

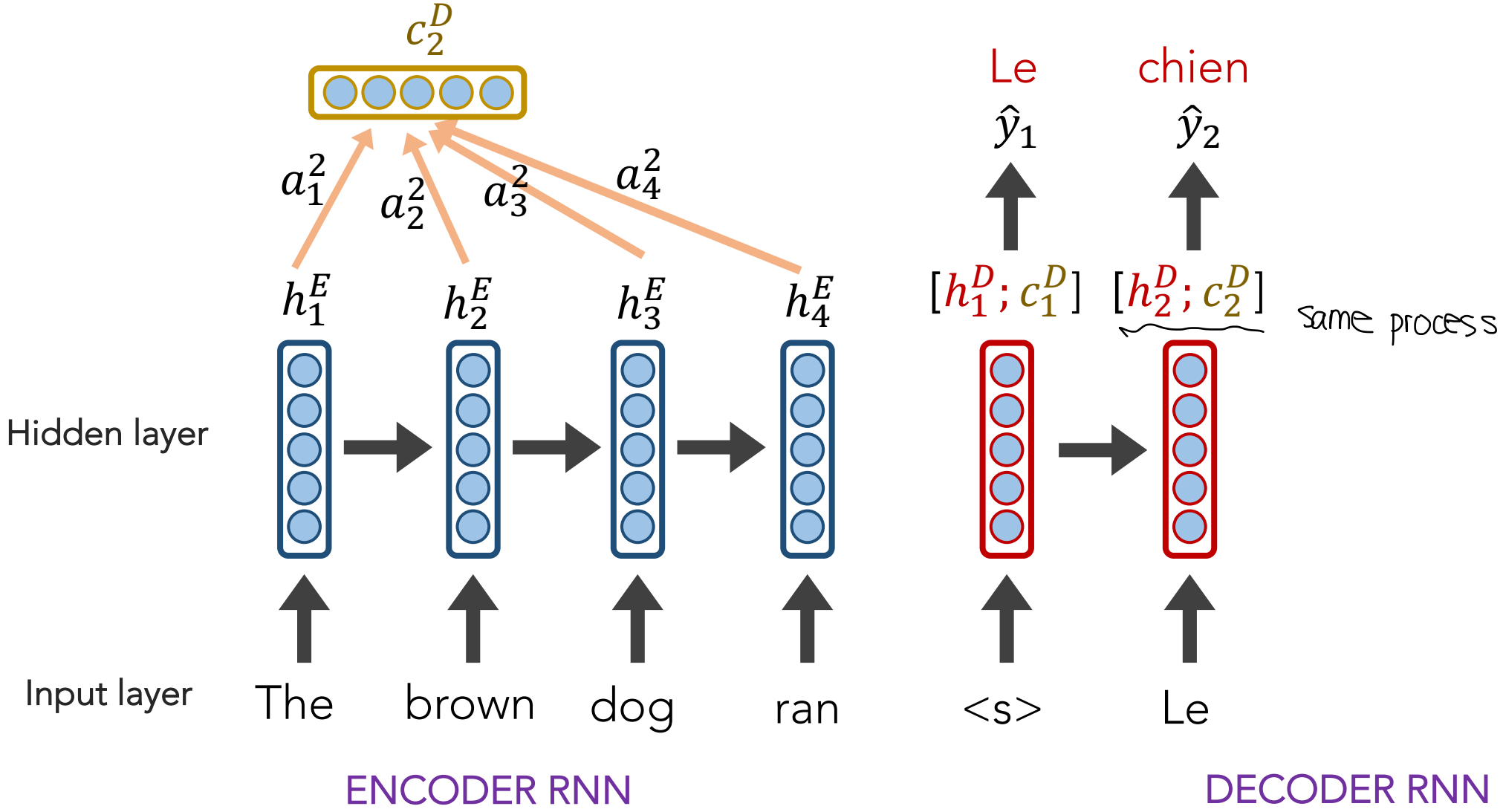
seq2seq + Attention

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



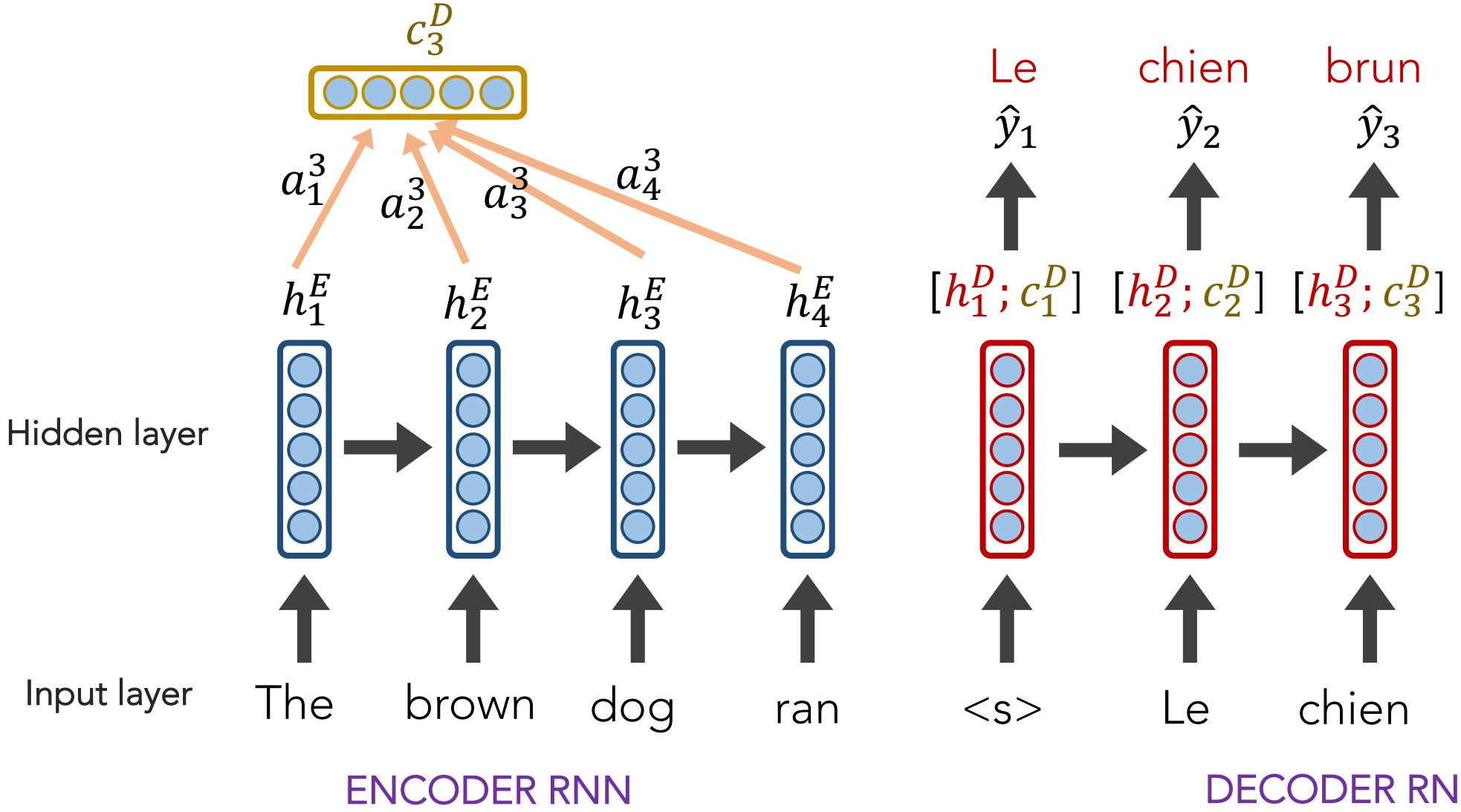
seq2seq + Attention

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



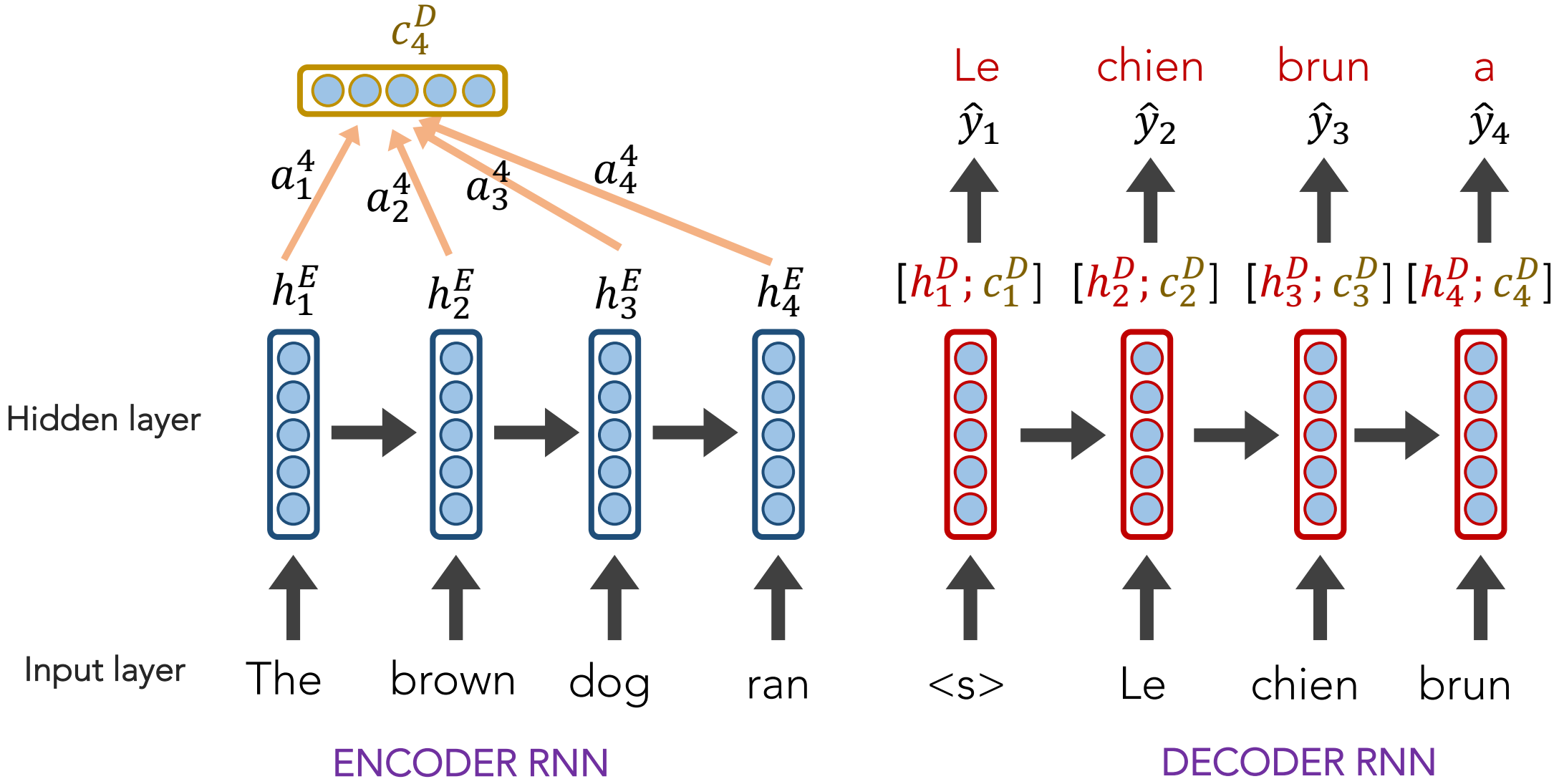
seq2seq + Attention

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



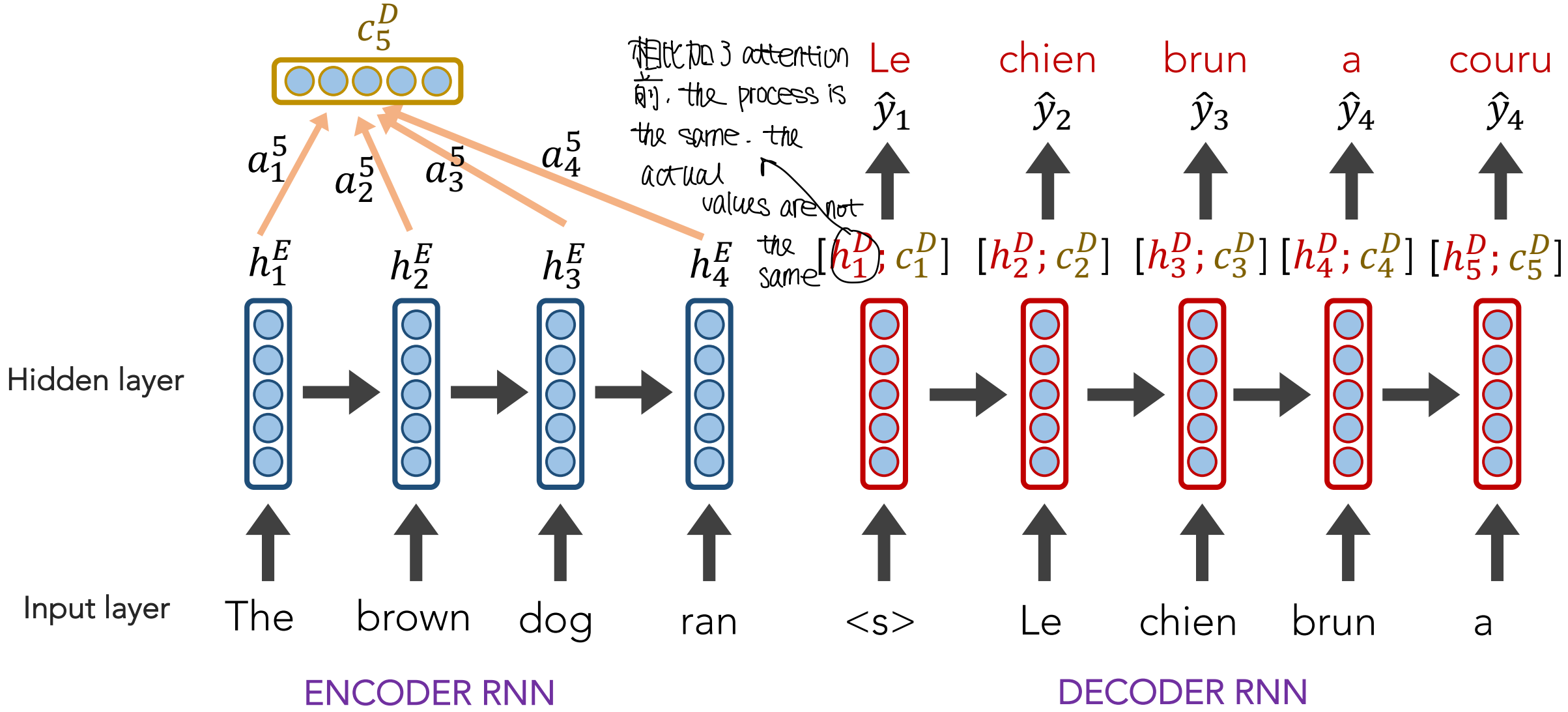
seq2seq + Attention

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



seq2seq + Attention

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



For convenience, here's the Attention calculation summarized on 1 slide

Attention output

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

"source context for decoder step t"

(weighted sum)

Attention weights

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

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

(softmax)

Attention scores

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

"How relevant is source token k for target step t?"

Attention input

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

h_t one decoder state

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

For convenience, here's the Attention calculation summarized on 1 slide

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

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

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

"How relevant is source token k for target step t?"

Attention input $s_1, s_2, \ldots, s_m$ all encoder states; h_t one decoder state

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

Popular Attention Scoring functions:

Dot-product

$h_t^T \times s_k$

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

Bilinear

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

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

Multi-Layer Perceptron

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

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

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

seq2seq + Attention

Attention:

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

look at the attention weight as you are translating

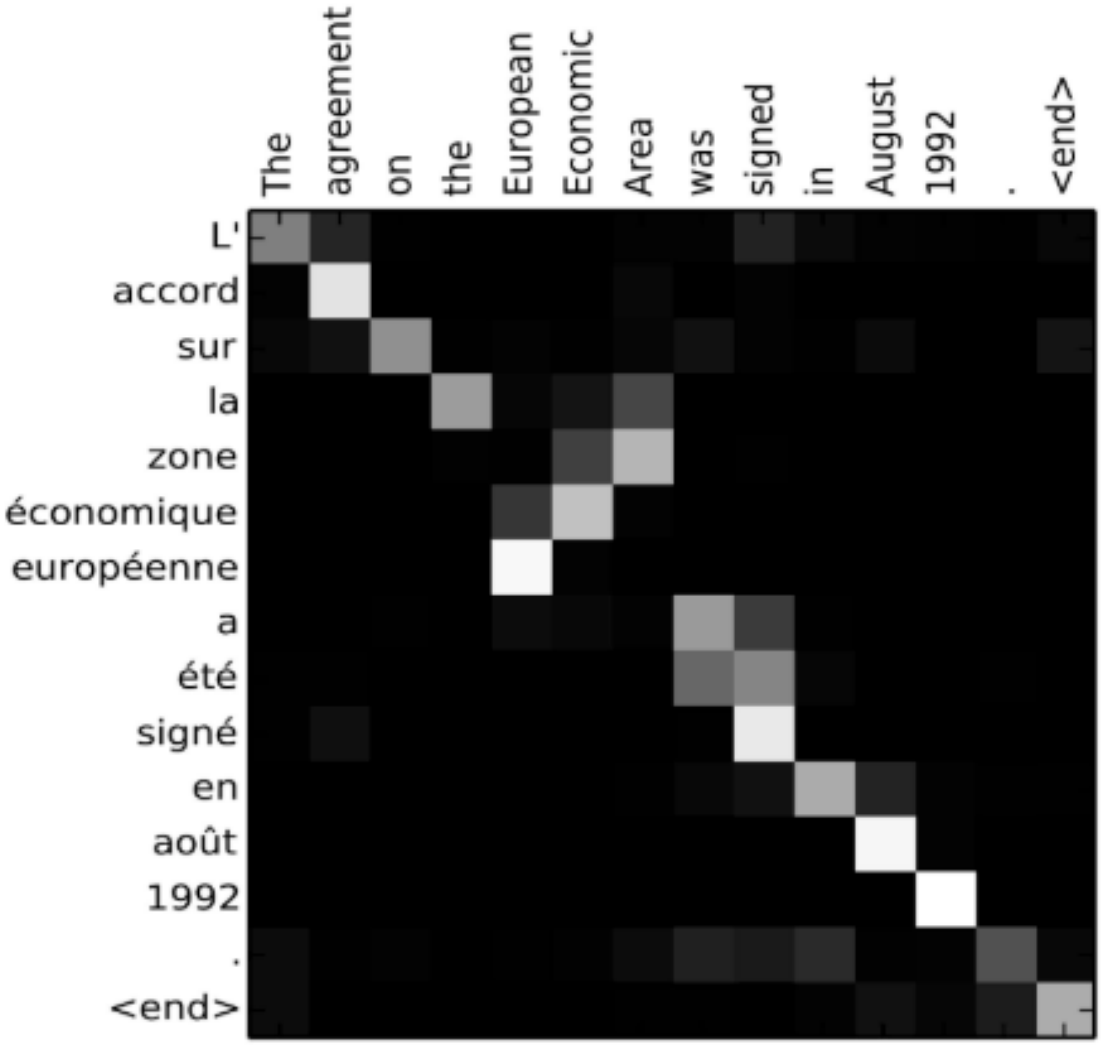


Image source: Fig 3 in [Bahdanau et al., 2015]

Takeaway:

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

Attention is powerful; allows us to conditionally weight our focus

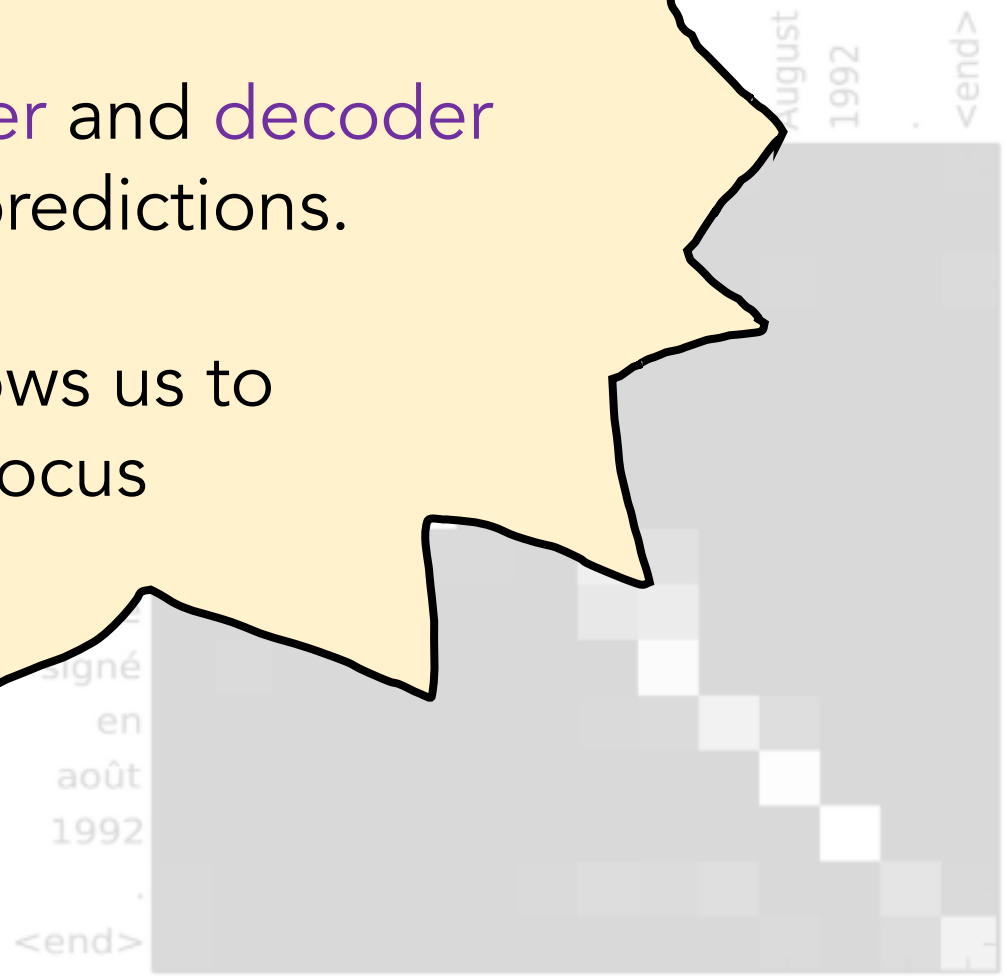


Image source: Fig 3 in Bahdanau et al., 2015

SUMMARY

- LSTMs yielded state-of-the-art results on most NLP tasks (2014-2018)
- seq2seq+Attention was an even more revolutionary idea (Google Translate used it)
- Attention allows us to place appropriate weight to the encoder's hidden states
- But, **LSTMs** require us to iteratively scan each word and wait until we're at the end before we can do anything

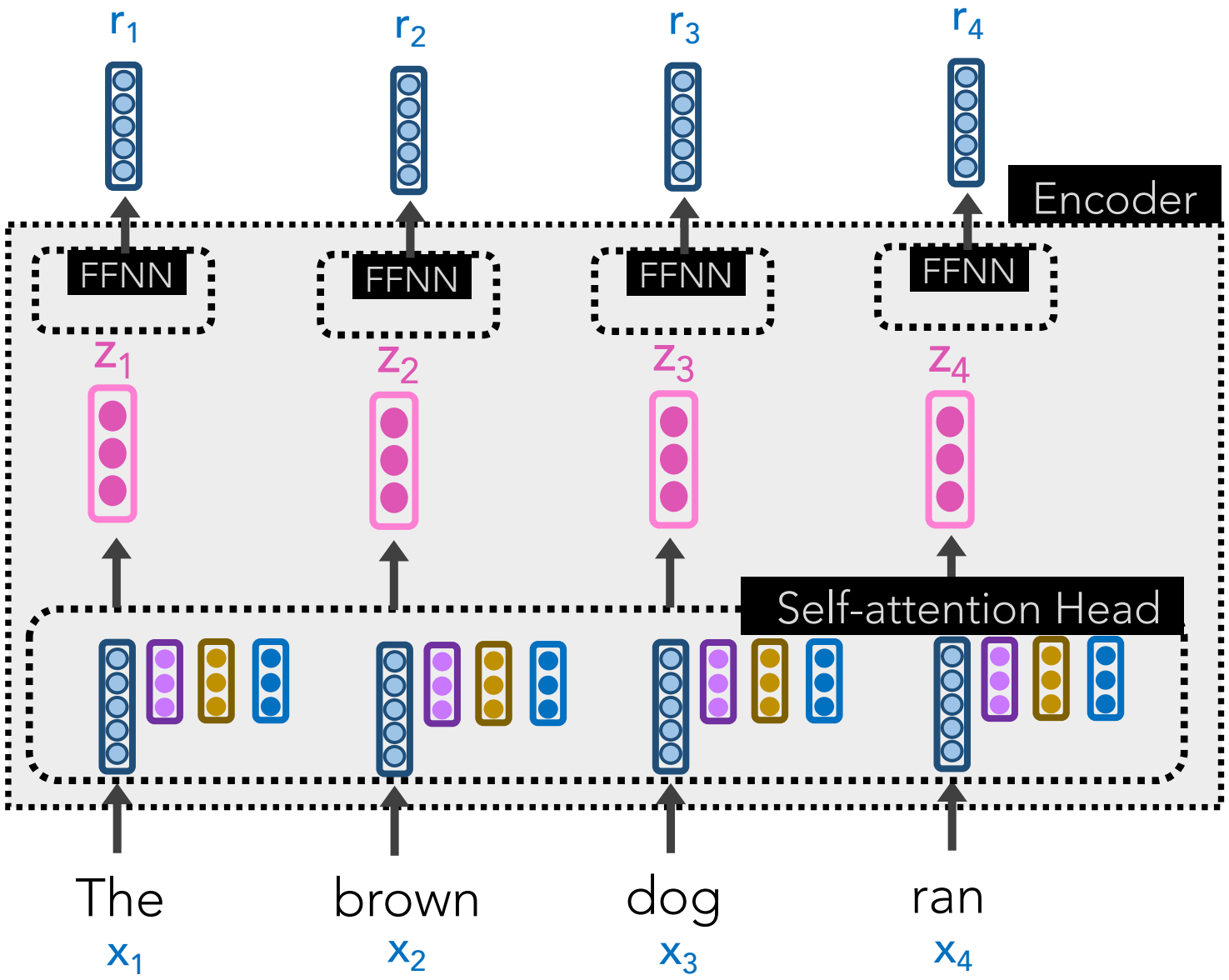
Outline

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

Outline

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

Transformer Encoder



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

Only focus on itself

how much attention do I focus on different part

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

Its results are unbelievably good.

BERT (a Transformer variant)

not autoregressive model . not just the things in front of you

BERT is trained on a lot of text data: Bidirectional

Yay, for transfer learning!

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

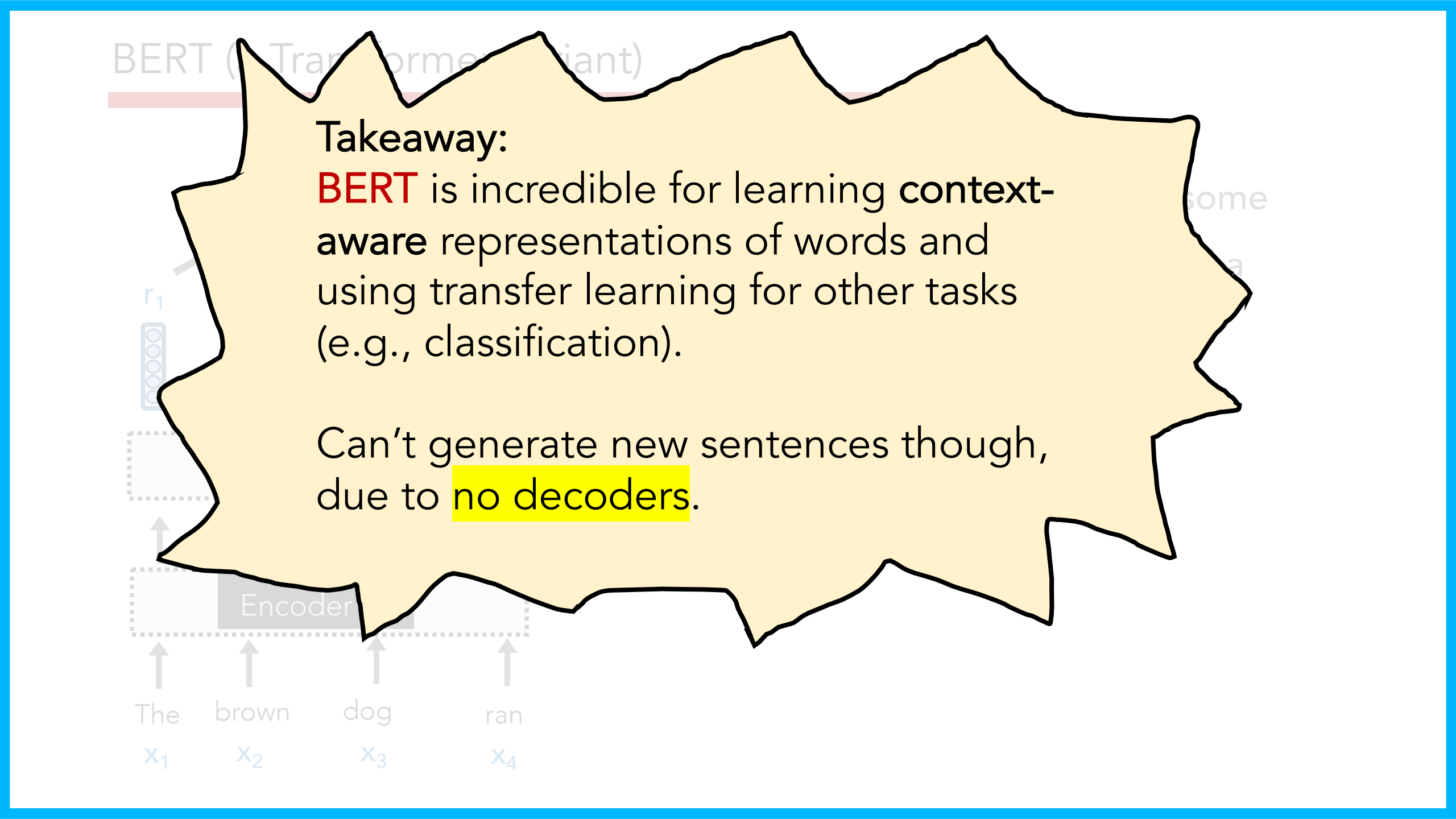
not the contextulized embeddings that we use. but the weight we care. use the weights ⇒ already in a good places

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

110M parameters!

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

340M parameters!


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

Transformer

What if we want to generate a new output sequence?

GPT-2 model to the rescue!

Generative Pre-trained Transformer 2

GPT-2 (a Transformer variant)

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

GPT-2

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

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

Can fine-tune toward other tasks, too.

GPT-2 (a Transformer variant)

GPT-2 is:

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

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

Yay, for transfer learning!

QUESTIONS?