Modelling Sequential Data with ELMo, BERT, Transformers, and other childhood heroes

Harvard IACS

ComputeFest 2020

Chris Tanner

FOREWORD

We could easily spend an entire semester on this material.

The goal for today is to convey:

- the ubiquity and importance of sequential data
- high-level overview of the most useful, relevant models
- foundation for diving deeper
- when to use which models, based on your data

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

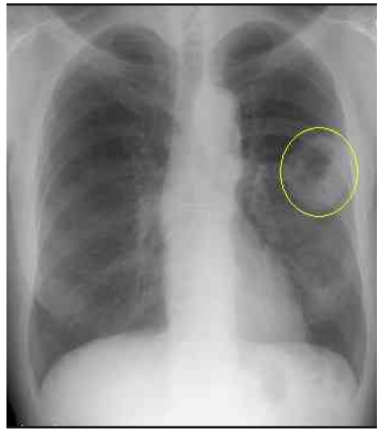
Seq2Seq +Attention

Transformers +BERT

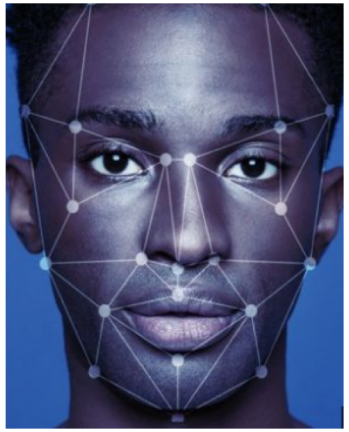
Conclusions

Background

Many classification and regression tasks involve data that is assumed to be independent and identically distributed (i.i.d.). For example:



Detecting lung cancer



Face recognition

Risk of heart attack

Background

Much of our data is inherently **sequential**

scale	examples
WORLD	Natural disasters (e.g., earthquakes) Climate change
HUMANITY	Stock market Flu outbreaks
INDIVIDUAL PEOPLE	Speech recognition Machine Translation (e.g., English -> French) Cancer treatment

Background

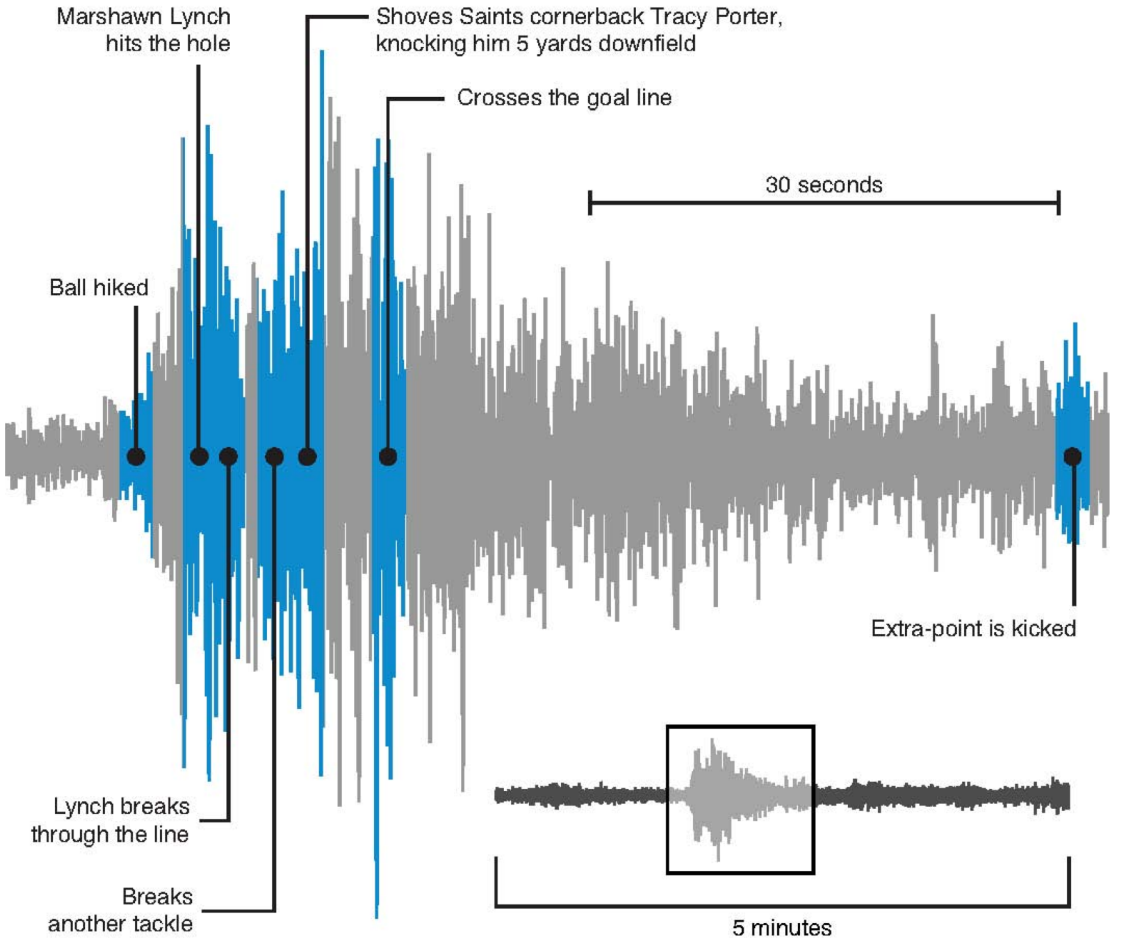
Much of our data is inherently **sequential**

scale	examples
WORLD	Natural disasters (e.g., earthquakes)
	Climate change
HUMANITY	Stock market
	Flu outbreaks
INDIVIDUAL PEOPLE	Speech recognition
	Machine Translation (e.g., English -> French)
	Cancer treatment

Background

Much of our data is inherently **sequential**

PREDICTING EARTHQUAKES



Background

Much of our data is inherently **sequential**

STOCK MARKET PREDICTIONS

Background

Much of our data is inherently **sequential**

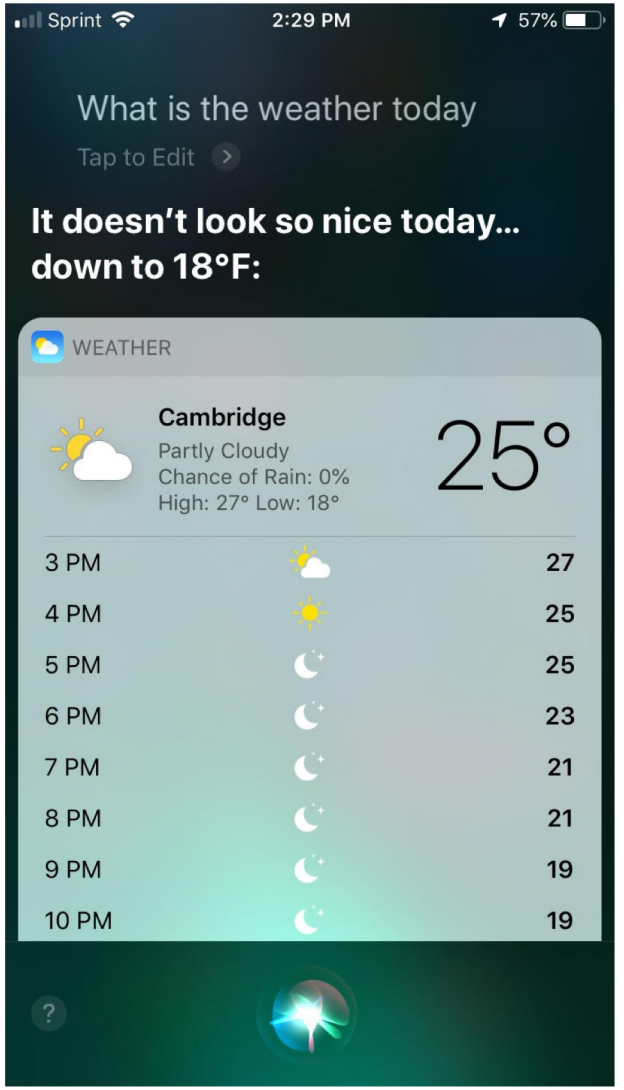
SPEECH RECOGNITION

“What is the weather today?”

“What is the weather two day?”

“What is the whether too day?”

“What is, the Wrether to Dae?”



Background

Regardless of how we model sequential data, keep in mind that we can estimate any time series as follows:

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

Background

Regardless of how we model sequential data, keep in mind that we can estimate any time series as follows:

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

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

Background

Regardless of how we model sequential data, keep in mind that we can estimate any time series as follows:

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

Example

The probability of the following 3-day weather pattern in Seattle:

Day 1

Day 2

Day 3

Example

The probability of the following 3-day weather pattern in Seattle:

Day 1 

Day 2 

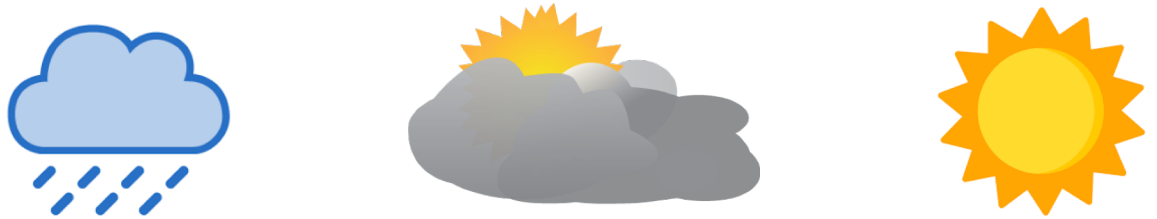
Day 3 

$P($  $) =$

Example

The probability of the following 3-day weather pattern in Seattle:

Day 1 Day 2 Day 3



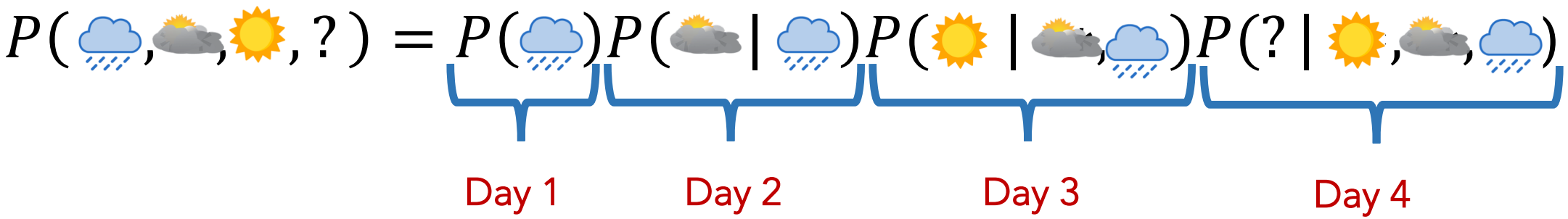
$$P(\text{🌧}, \text{⛅}, \text{☀}) = \underbrace{P(\text{🌧})}_{\text{Day 1}} \underbrace{P(\text{⛅} \mid \text{🌧})}_{\text{Day 2}} \underbrace{P(\text{☀} \mid \text{⛅}, \text{🌧})}_{\text{Day 3}}$$

Example

Why is it useful to accurately estimate the likelihood of any given sequence of length N?

Background

Having the ability to estimate the probability of any sequence of length N allows us to determine the most likely next event (i.e., sequence of length $N + 1$)



IMPORTANT

For the remainder of this lecture, we will use text (natural language) as examples because:

- It's easy to interpret success/failures
- Real-world impact and commonplace usages
- Availability of data to try things yourself

Yet, for any model, you can imagine using any other sequential data

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

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

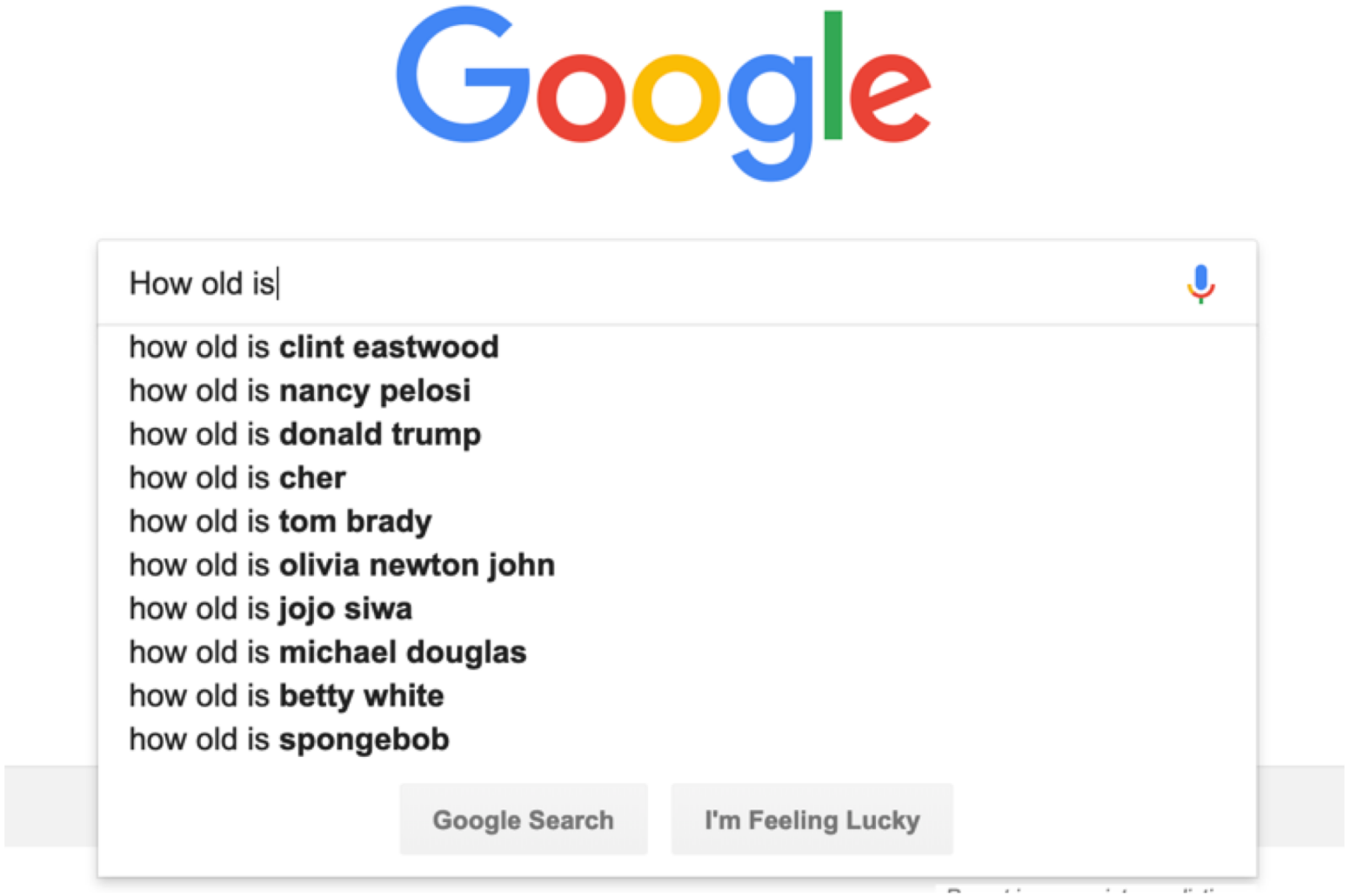
Let $\boldsymbol{X}$ = "Anqi was late for class"

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

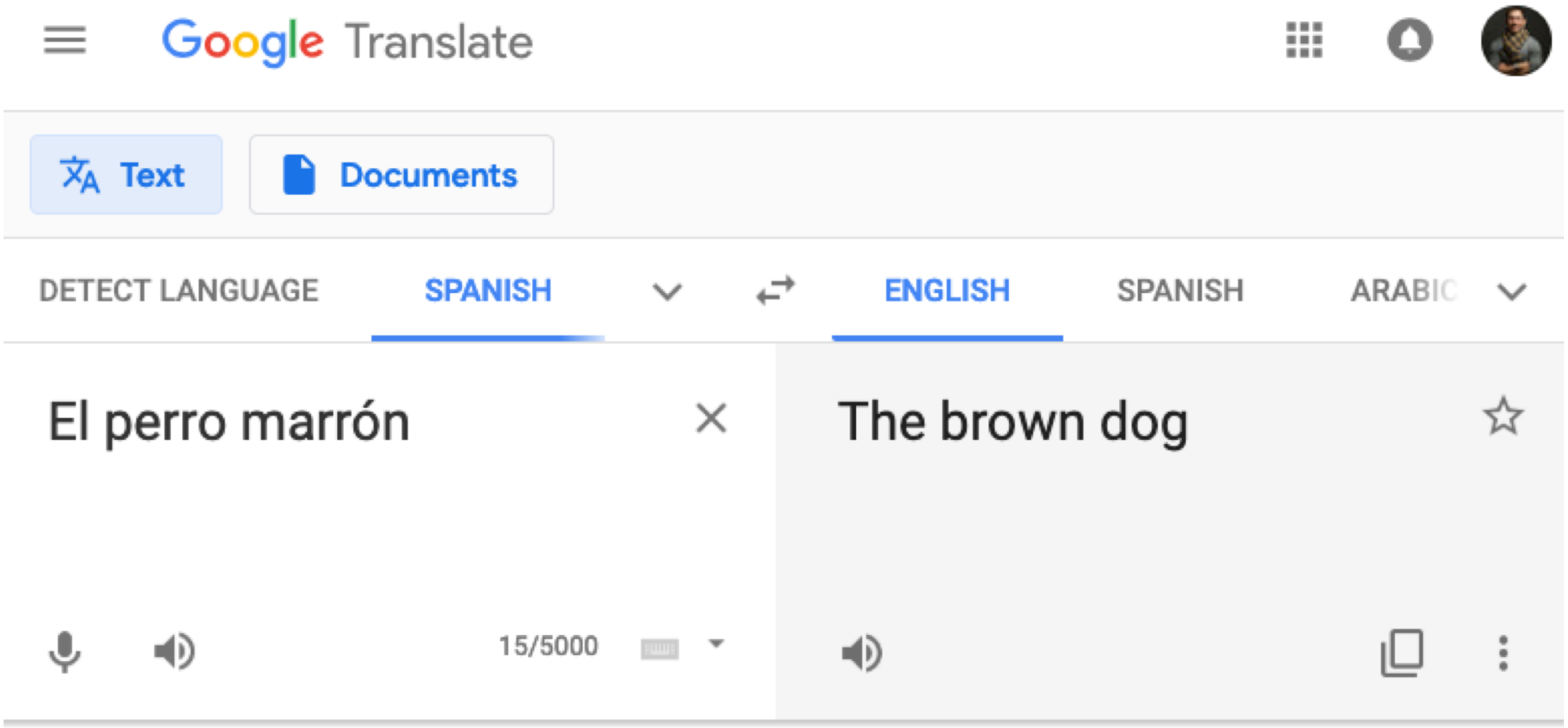


Language Modelling

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

Today, we heavily focus on Language Modelling (LM) because:

1. It's foundational for nearly all NLP tasks
2. Since we're ultimately modelling a sequence, LM approaches are generalizable to any type of data, not just text.

Language Modelling

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others.

Count how often each word occurs (in the training data).

Language Modelling

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

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

w_1 w_2 w_3 w_4 w_5

Language Modelling

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

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

w_1 w_2 w_3 w_4 w_5

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

$$= 0.00015 * 0.01 * 0.004 * 0.03 * 0.0035$$
$$= 6.3\text{x}10^{-13}$$

Language Modelling

How can we build a language model?

Naive Approach: unigram model

Assume each word is independer

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

w_1 w_2 w_3 w_4

You calculate each of these probabilities from the training corpus

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

$$= 0.00015 * 0.01 * 0.004 * 0.03 * 0.0035$$
$$= 6.3\text{x}10^{-13}$$

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

If "the" is the most common word in the training corpus

Language Modelling

How can we build a language model?

Alternative Approach: bigram model

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

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

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

probability

w_1 w_2 w_3 w_4 w_5

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

Language Modelling

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

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

probability

w_1 w_2 w_3 w_4 w_5

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

Language Modelling

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

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

w_1 w_2 w_3 w_4 w_5

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

Language Modelling

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

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

probability

w_1 w_2 w_3 w_4 w_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

How can w

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"

w_1 w_2 w_3 w_4 w_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?

Language Modelling

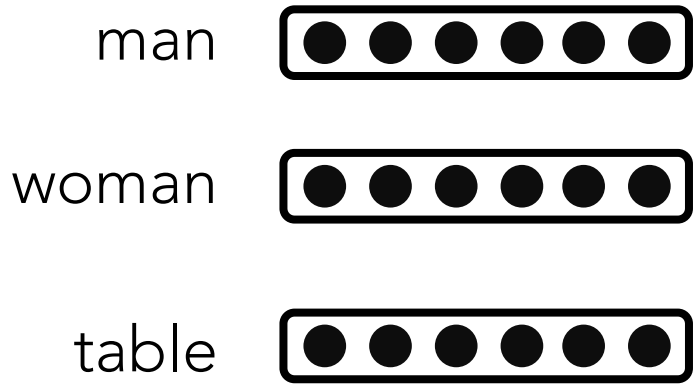
BIGRAM ISSUES?

- **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)

Language Modelling

IDEA: Let's use a neural networks!

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

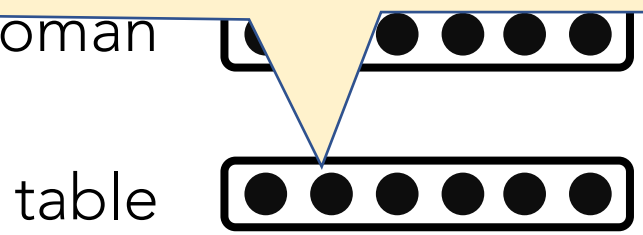


Language Modelling

IDEA: L

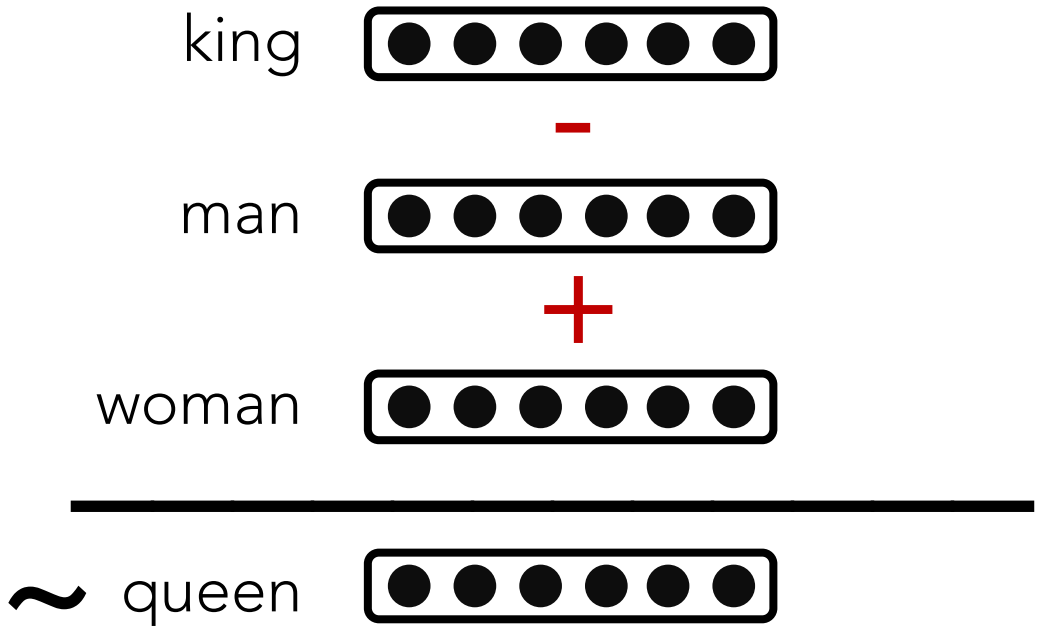
First, ea
(e.g., ve

- 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

How can we use these embeddings to build a 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

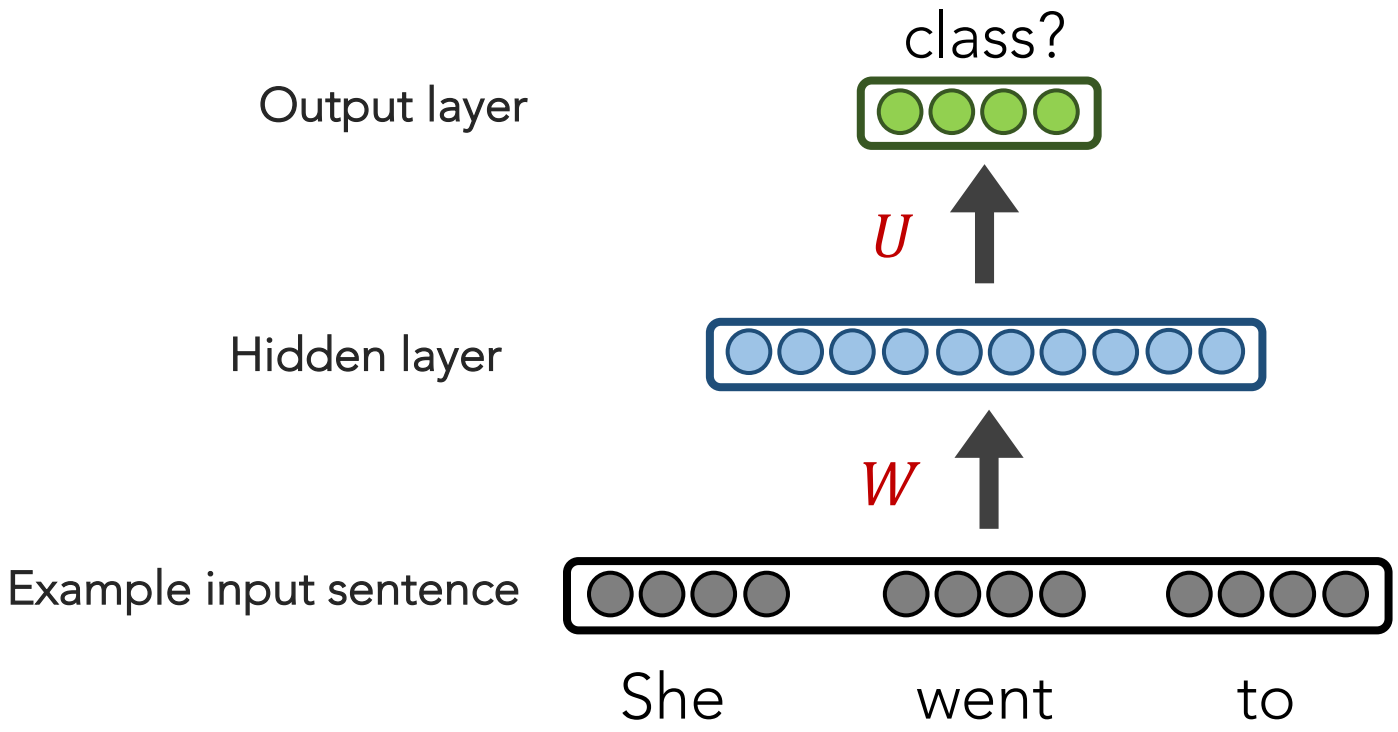
Example input sentence

She went to class

Language Modelling

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

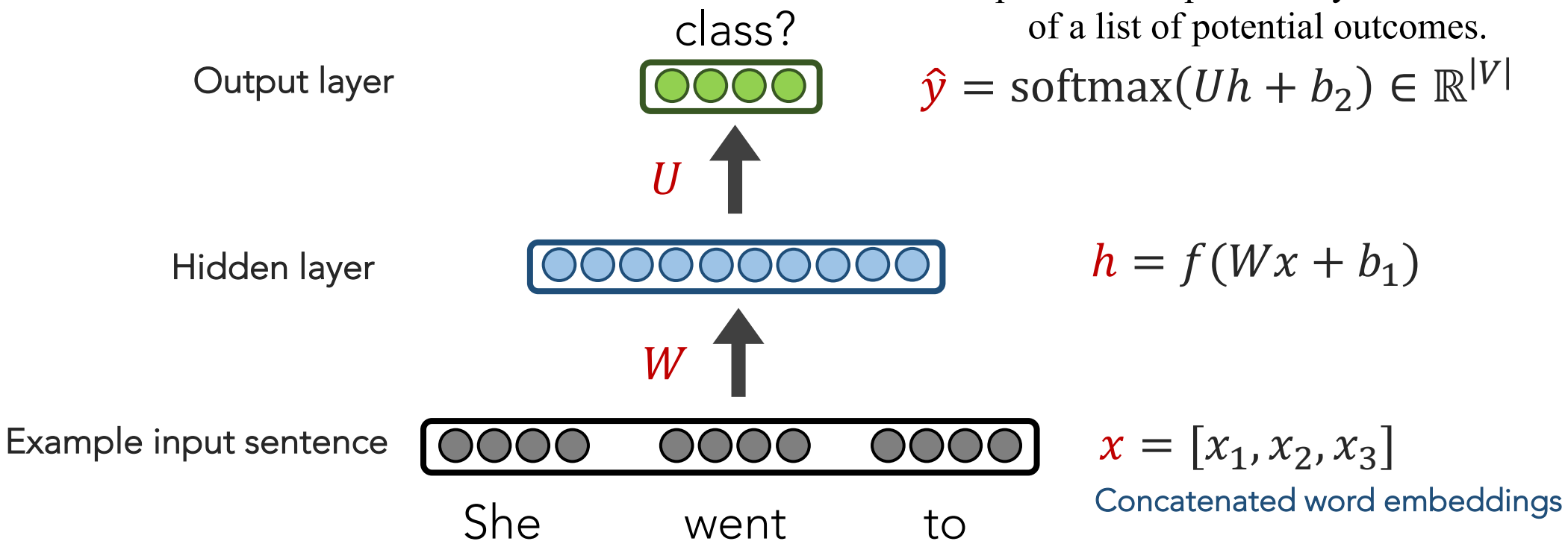


Language Modelling

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

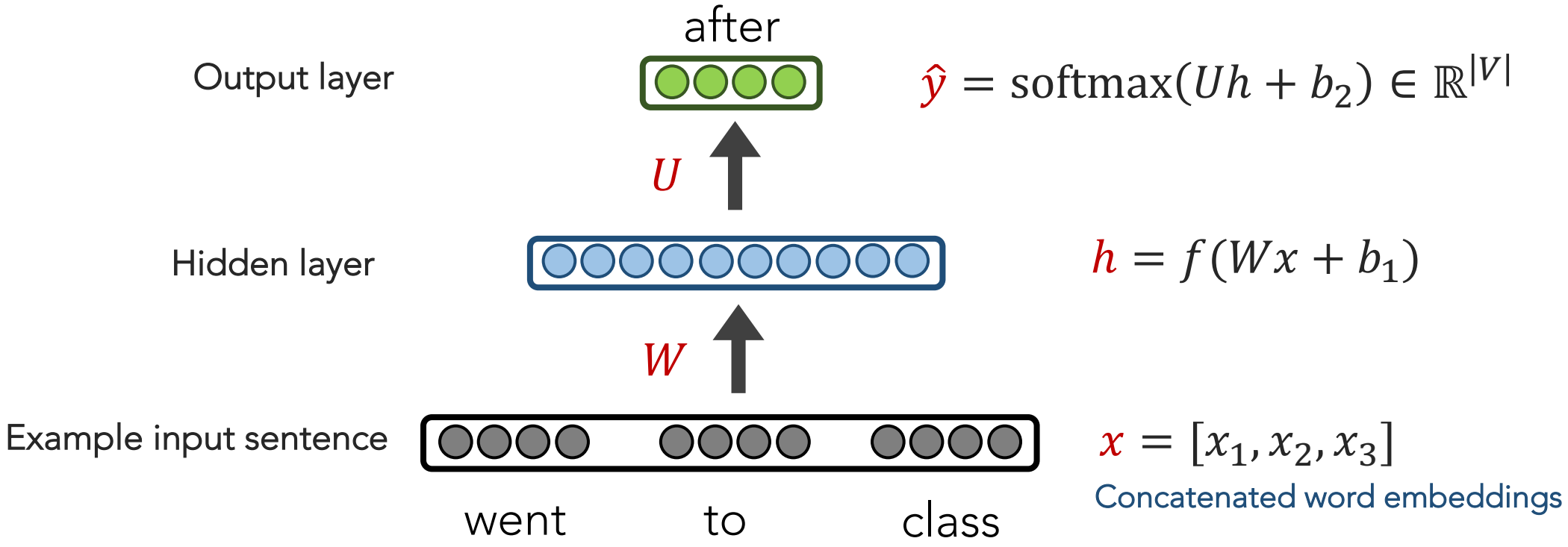
Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.



Language Modelling

Neural Approach #1: Feed-forward Neural Net

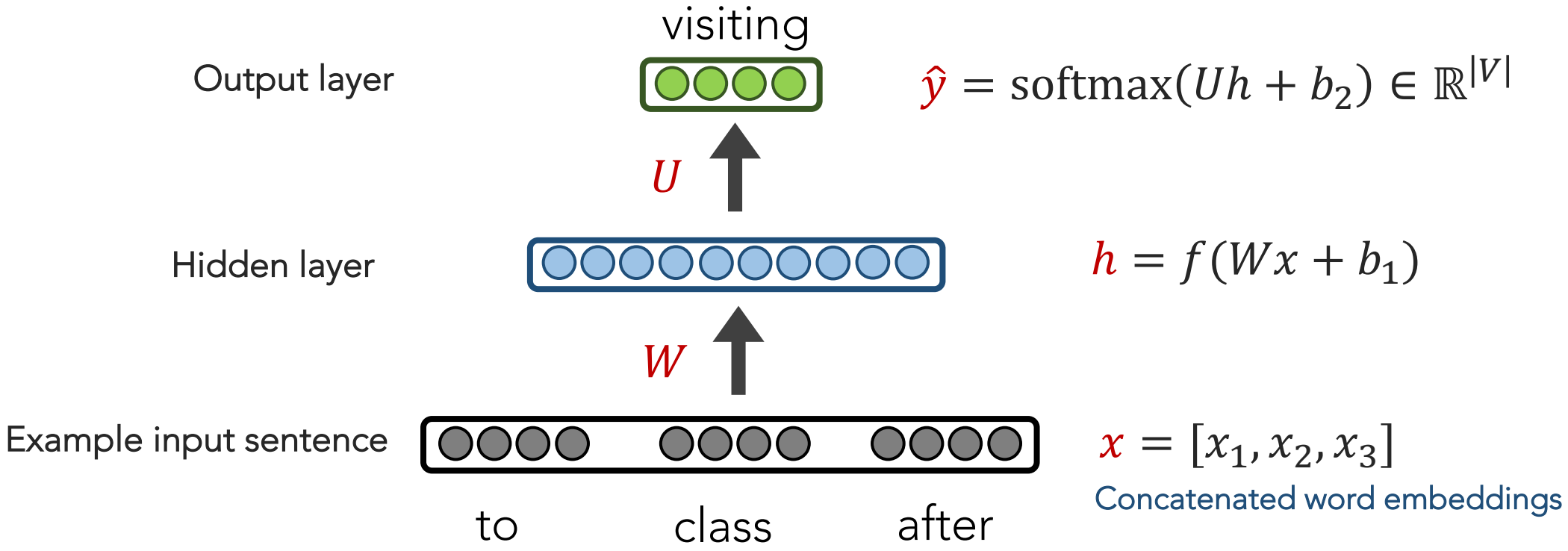
General Idea: using *windows* of words, predict the next word



Language Modelling

Neural Approach #1: Feed-forward Neural Net

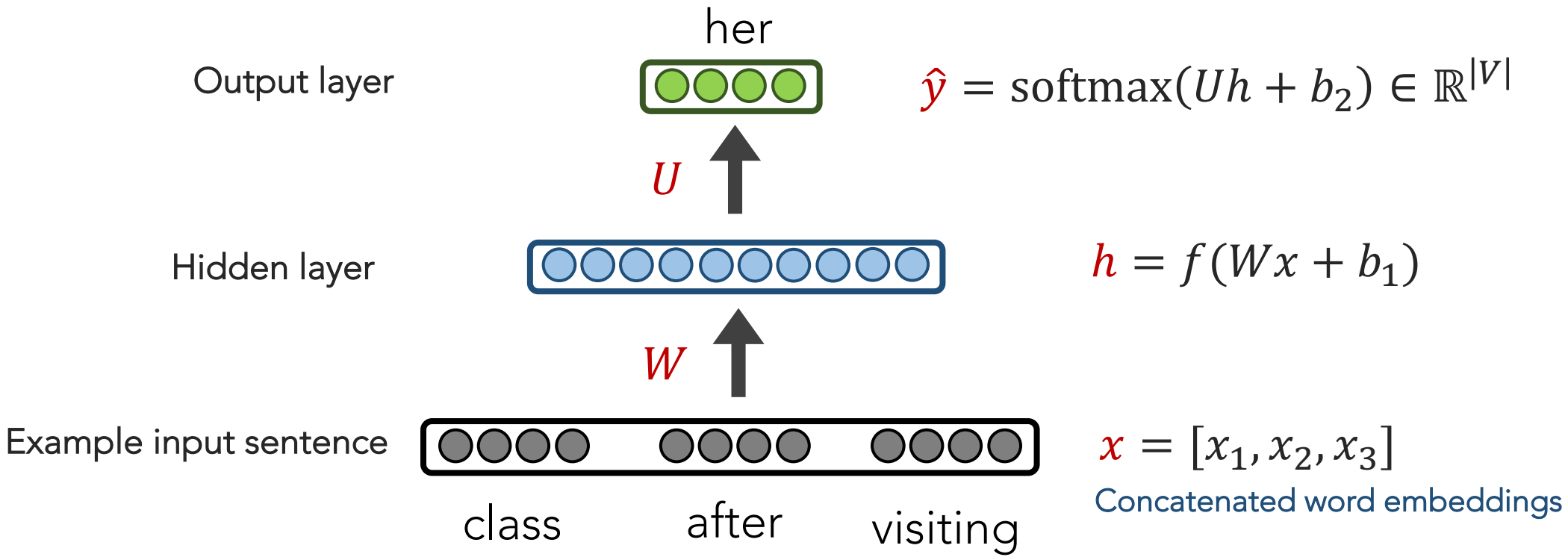
General Idea: using *windows* of words, predict the next word



Language Modelling

Neural Approach #1: Feed-forward Neural Net

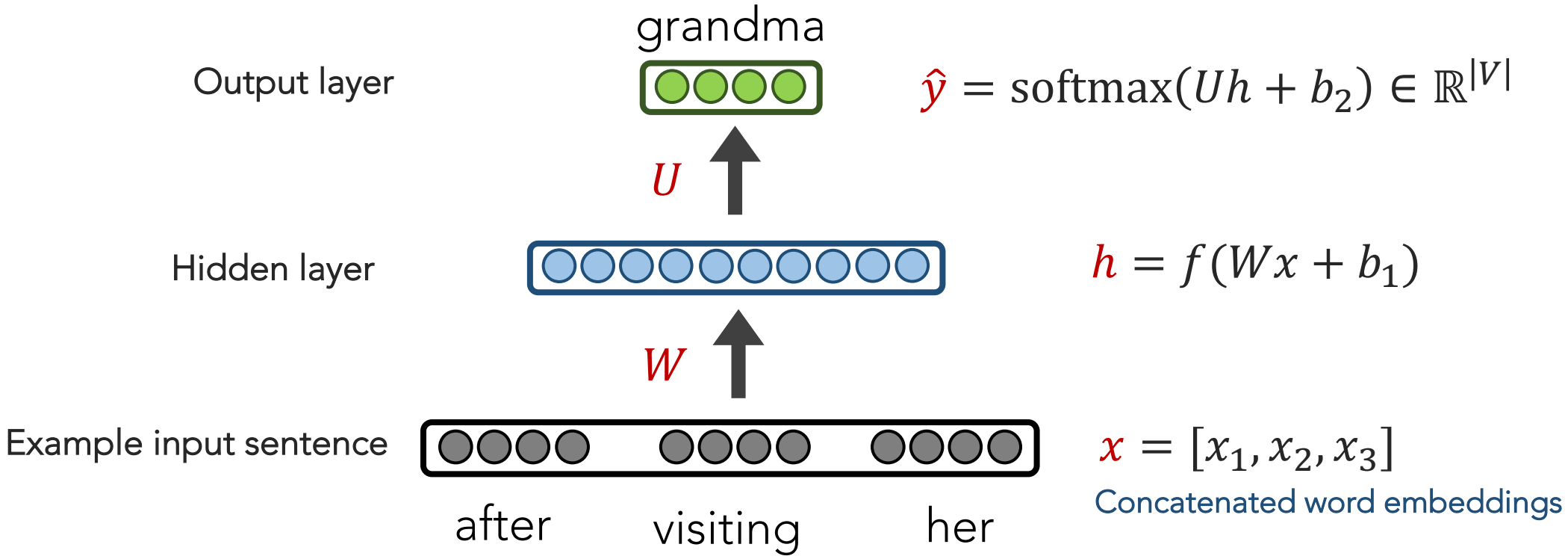
General Idea: using *windows* of words, predict the next word



Language Modelling

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word



Language Modelling

FFNN STRENGTHS?

FFNN ISSUES?

Language Modelling

FFNN STRENGTHS?

- No sparsity issues (it's okay if we've never seen a segment of words)
- No storage issues (we never store counts)

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)

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

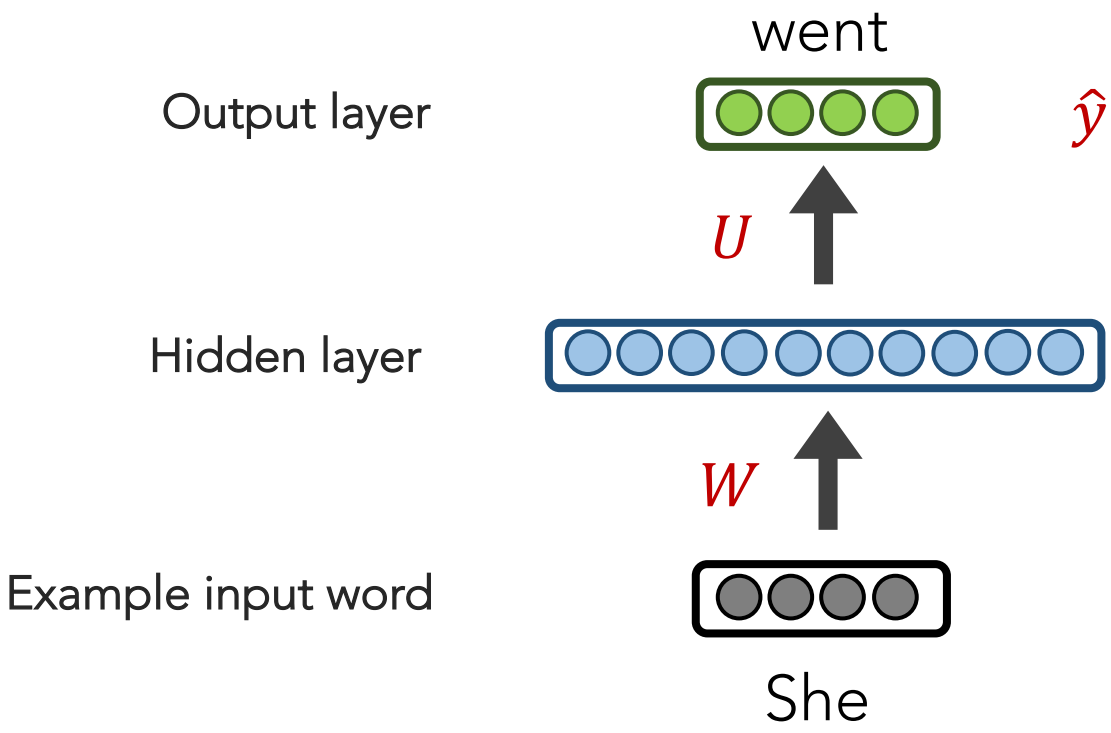
Seq2Seq +Attention

Transformers +BERT

Conclusions

Language Modelling

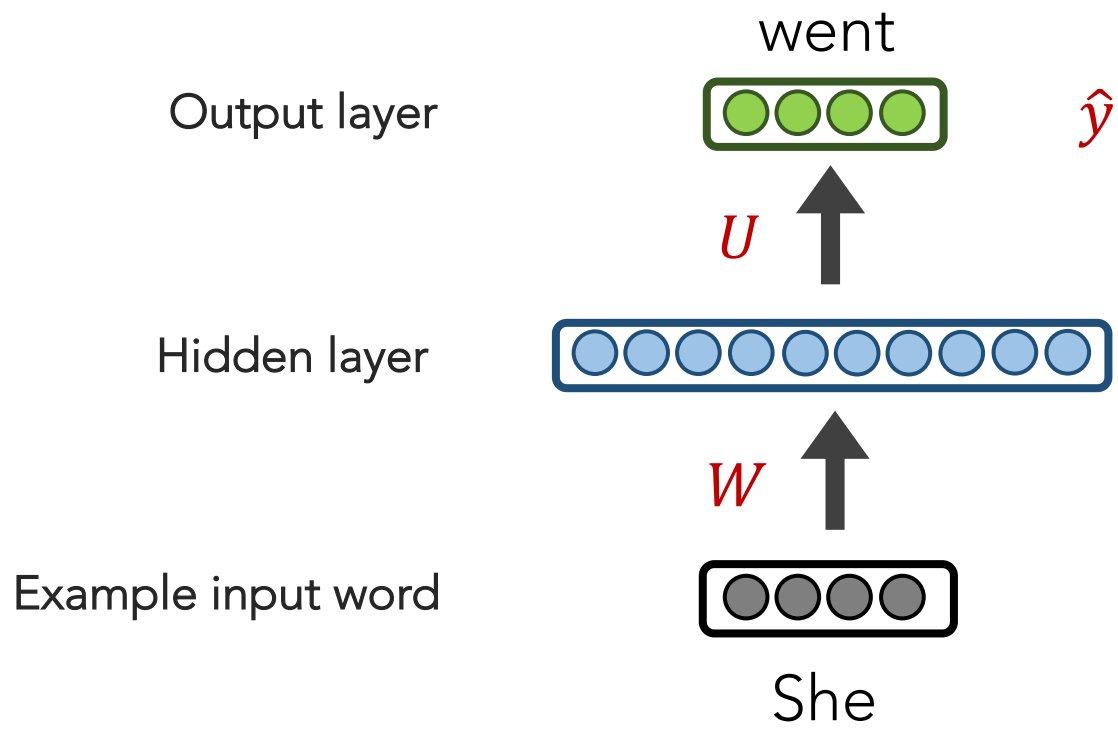
IDEA: for every individual input, output a prediction



Language Modelling

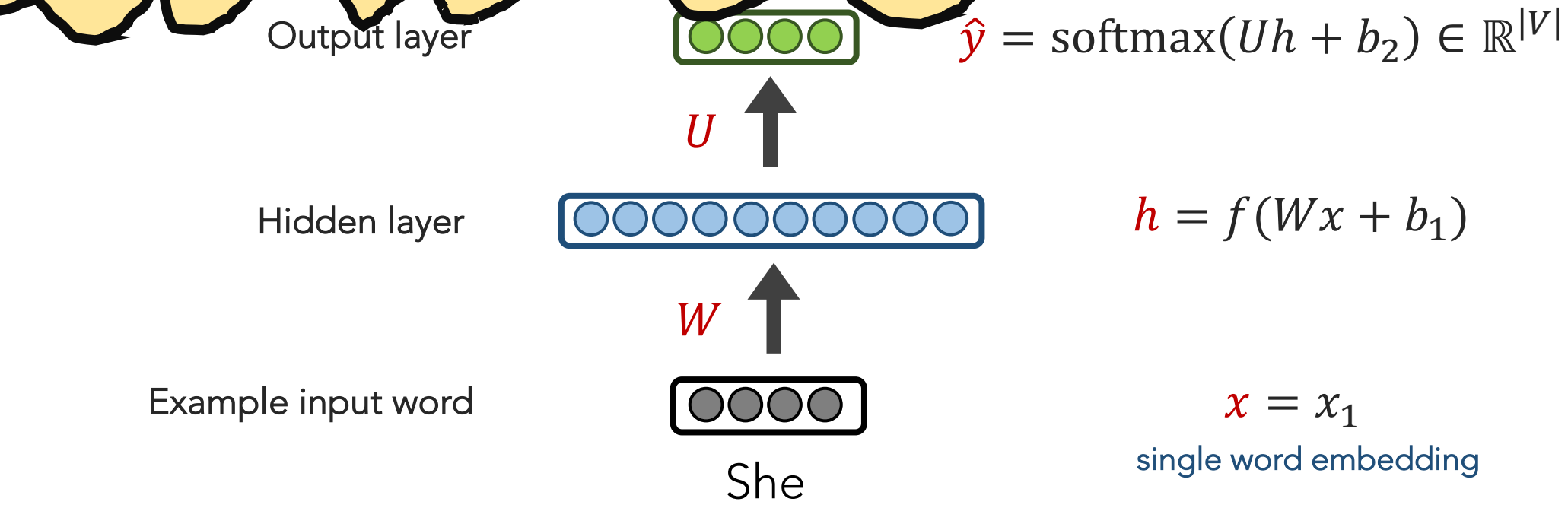
IDEA: for every individual input, output a prediction

Let's use the previous hidden state, too



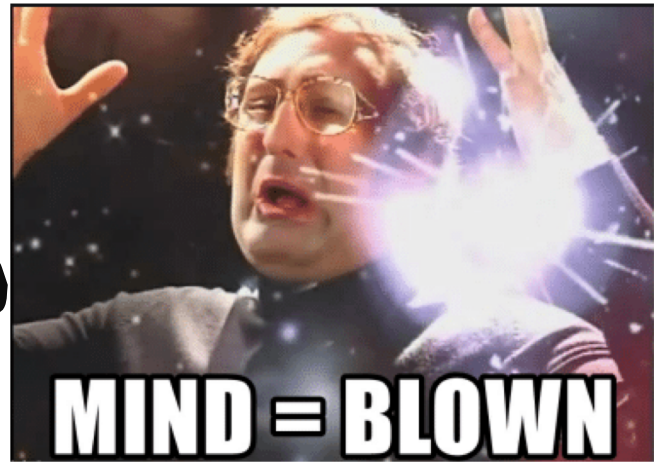
Language Modelling

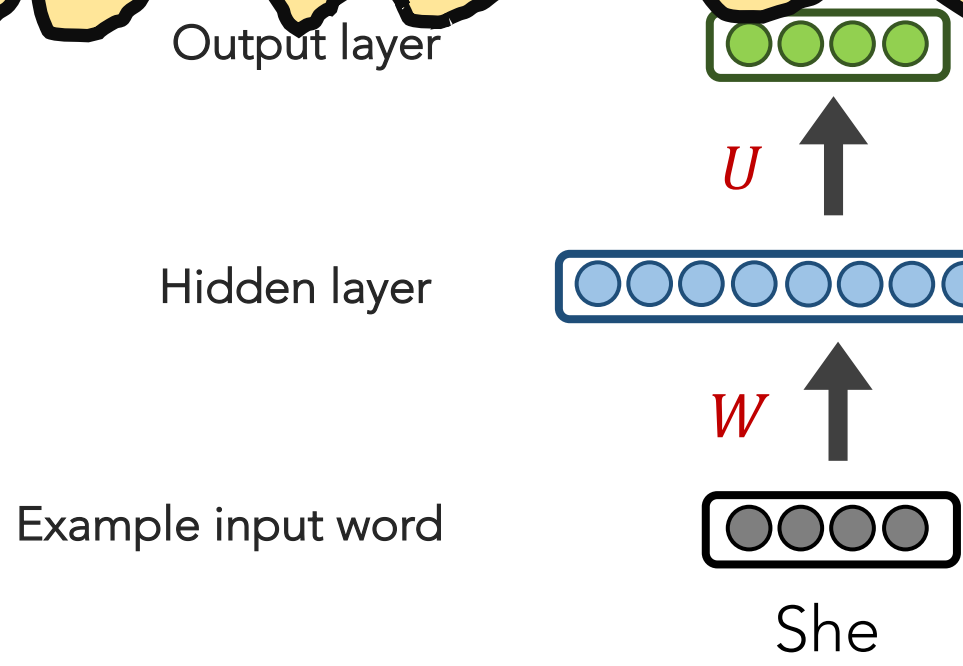
Let's use the previous hidden state, too



Language Modelling

IDEA: Let's use the previous hidden state, too





U

W

$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$

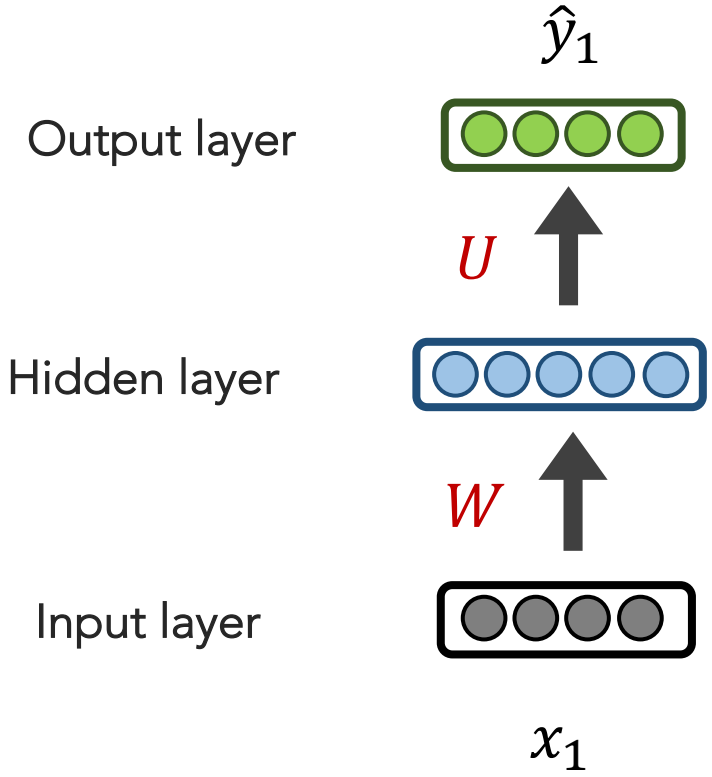
$h = f(Wx + b_1)$

$x = x_1$

single word embedding

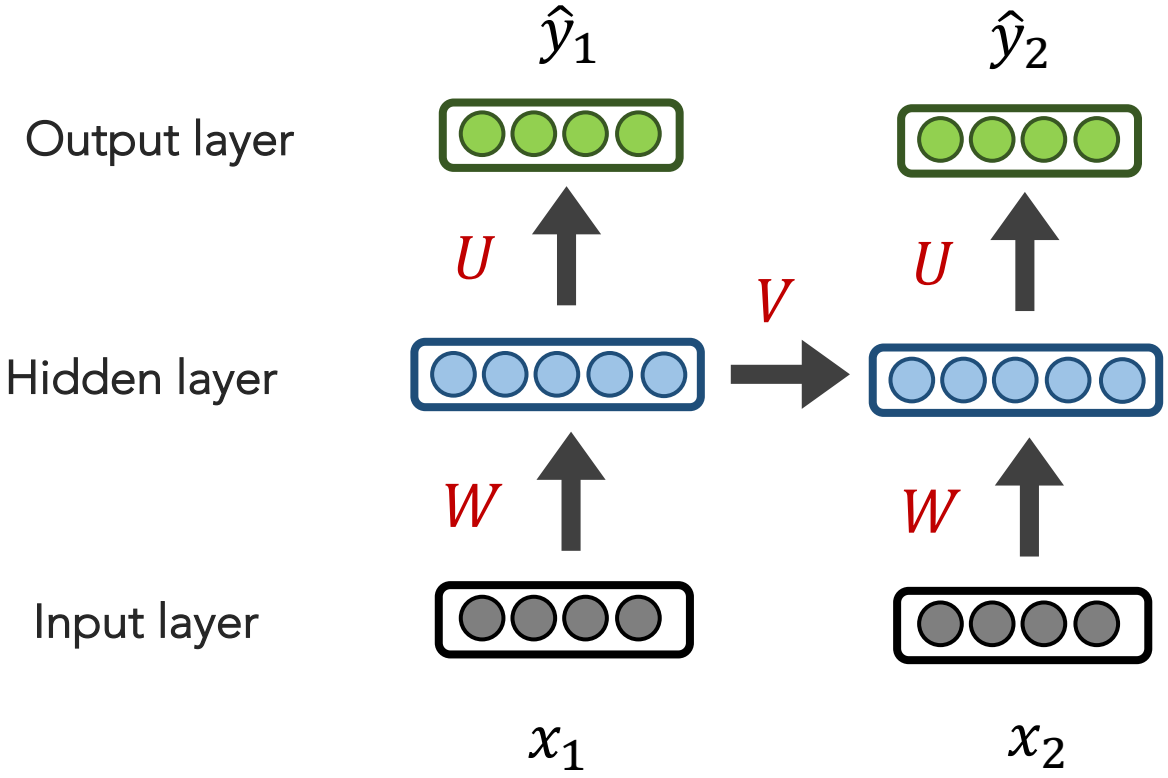
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



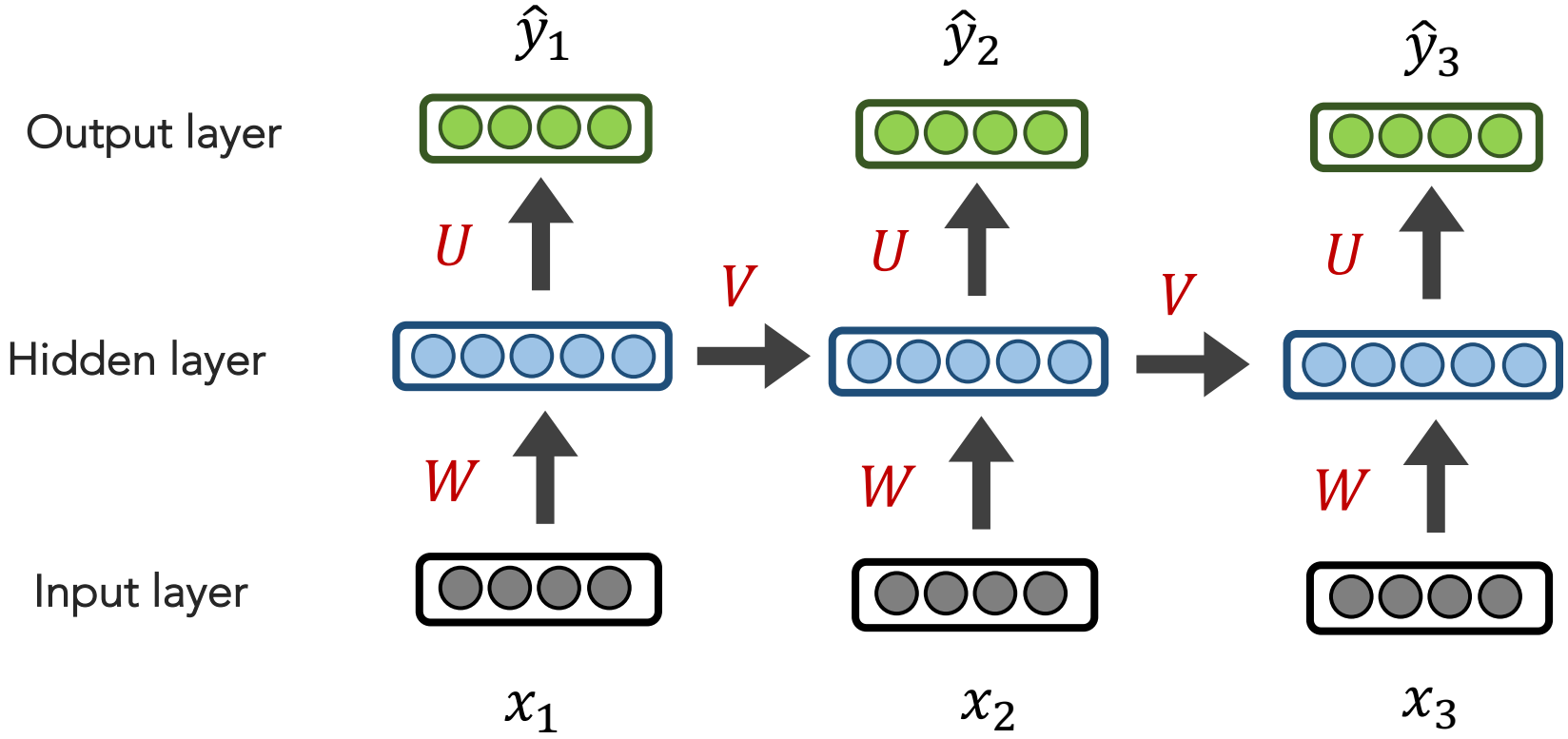
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



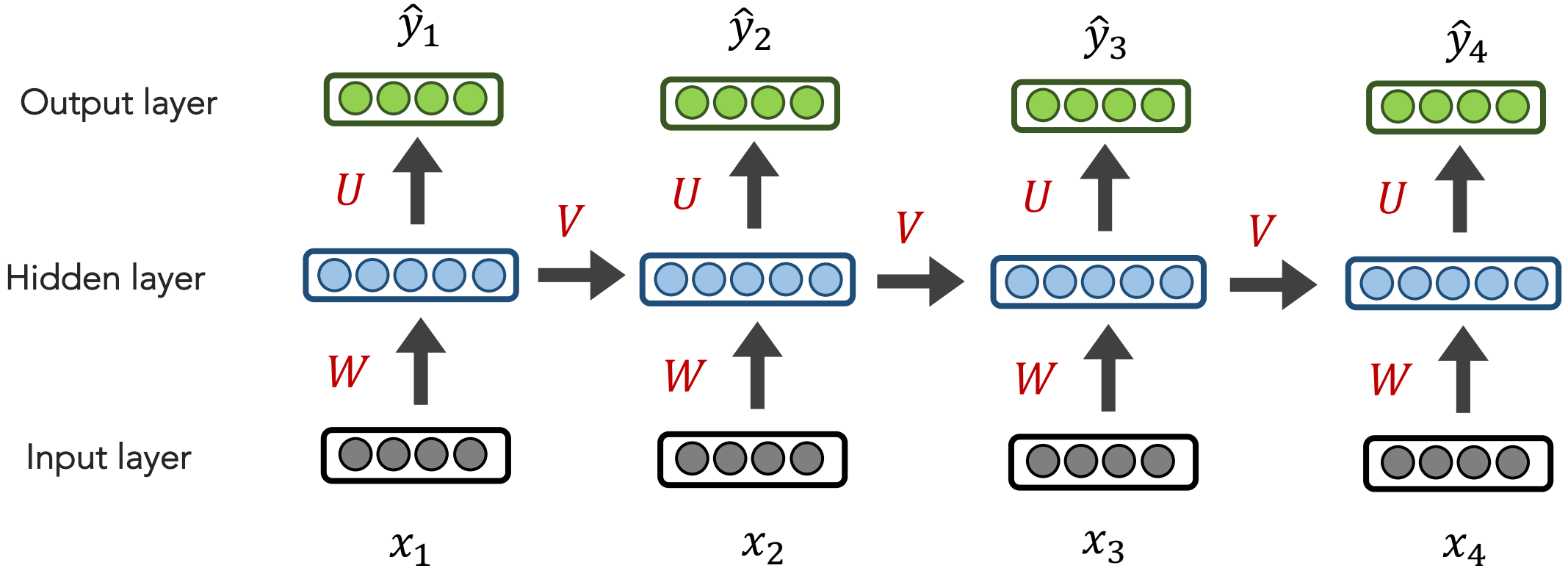
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



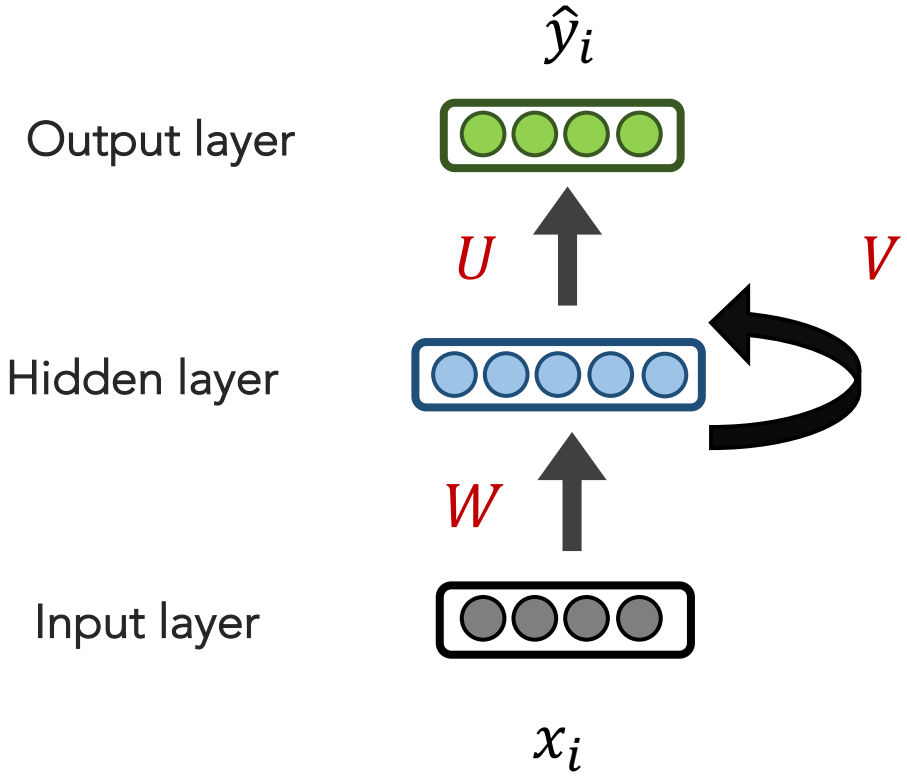
Language Modelling

Neural Approach #2: Recurrent Neural Network (RNN)



Language Modelling

Some people find this abstract view useful.

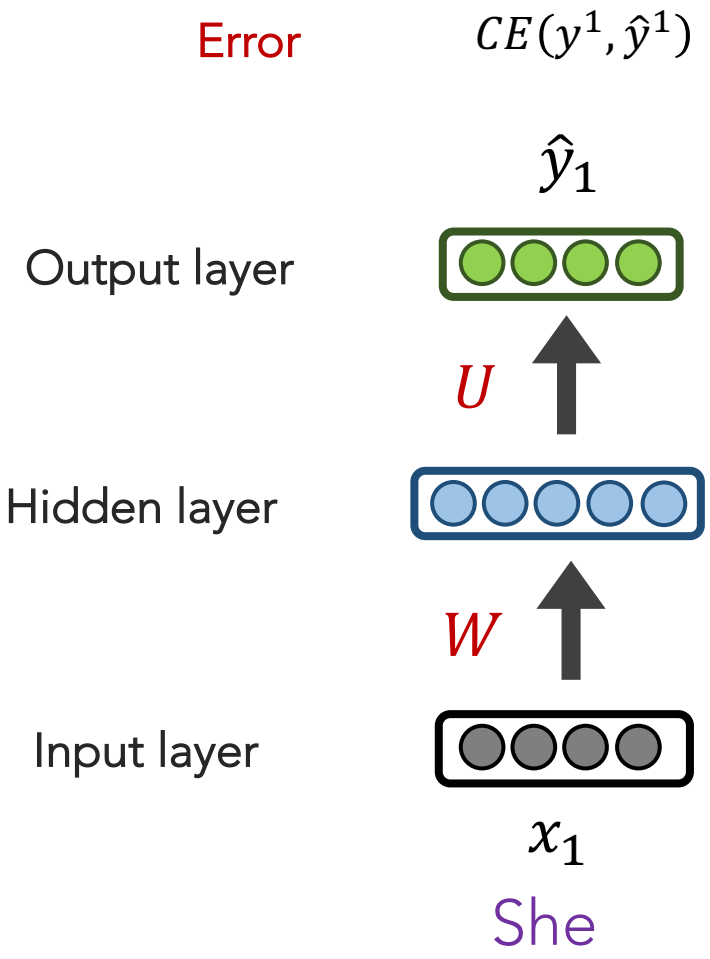


The recurrent loop V conveys that the current hidden layer is influenced by the hidden layer from the previous time step.

RNN

Training Process

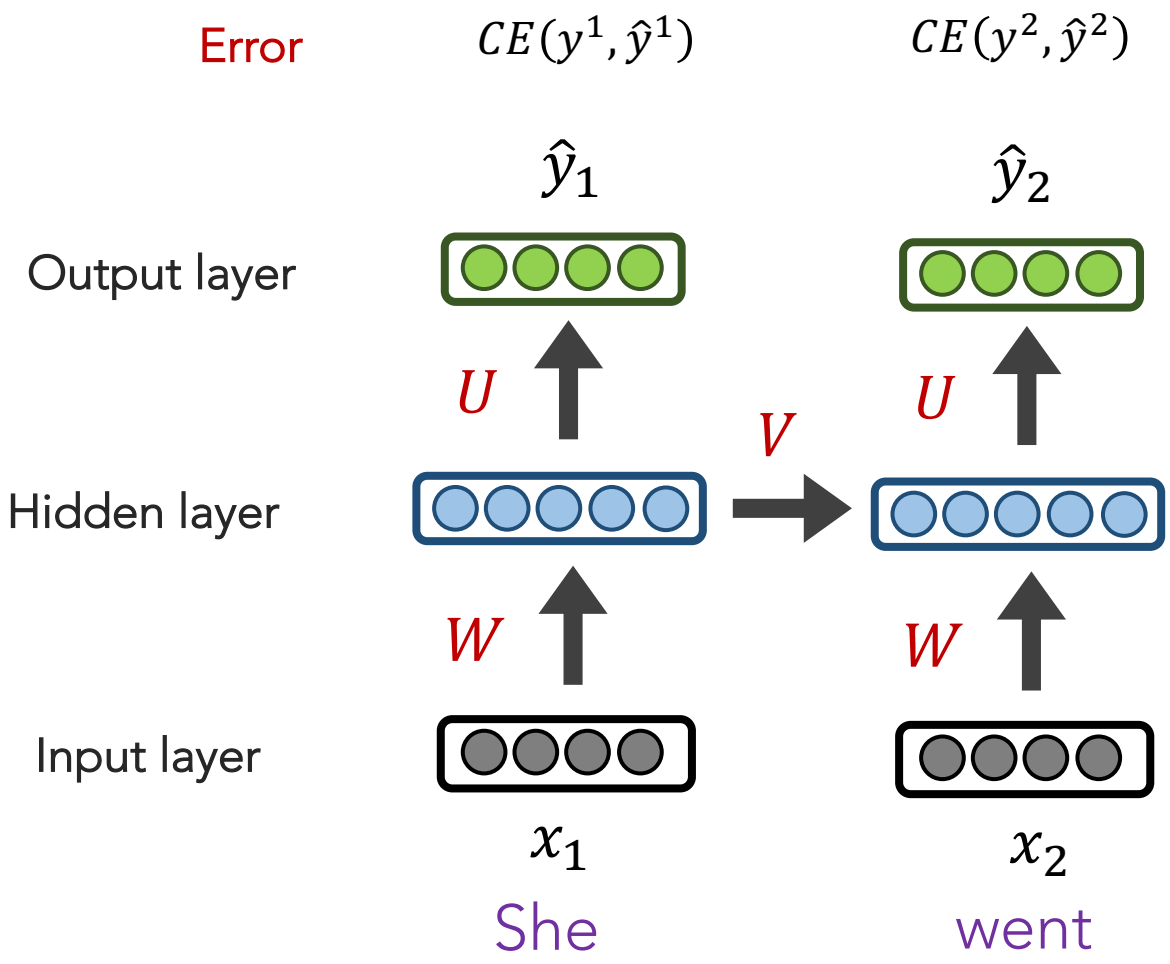
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

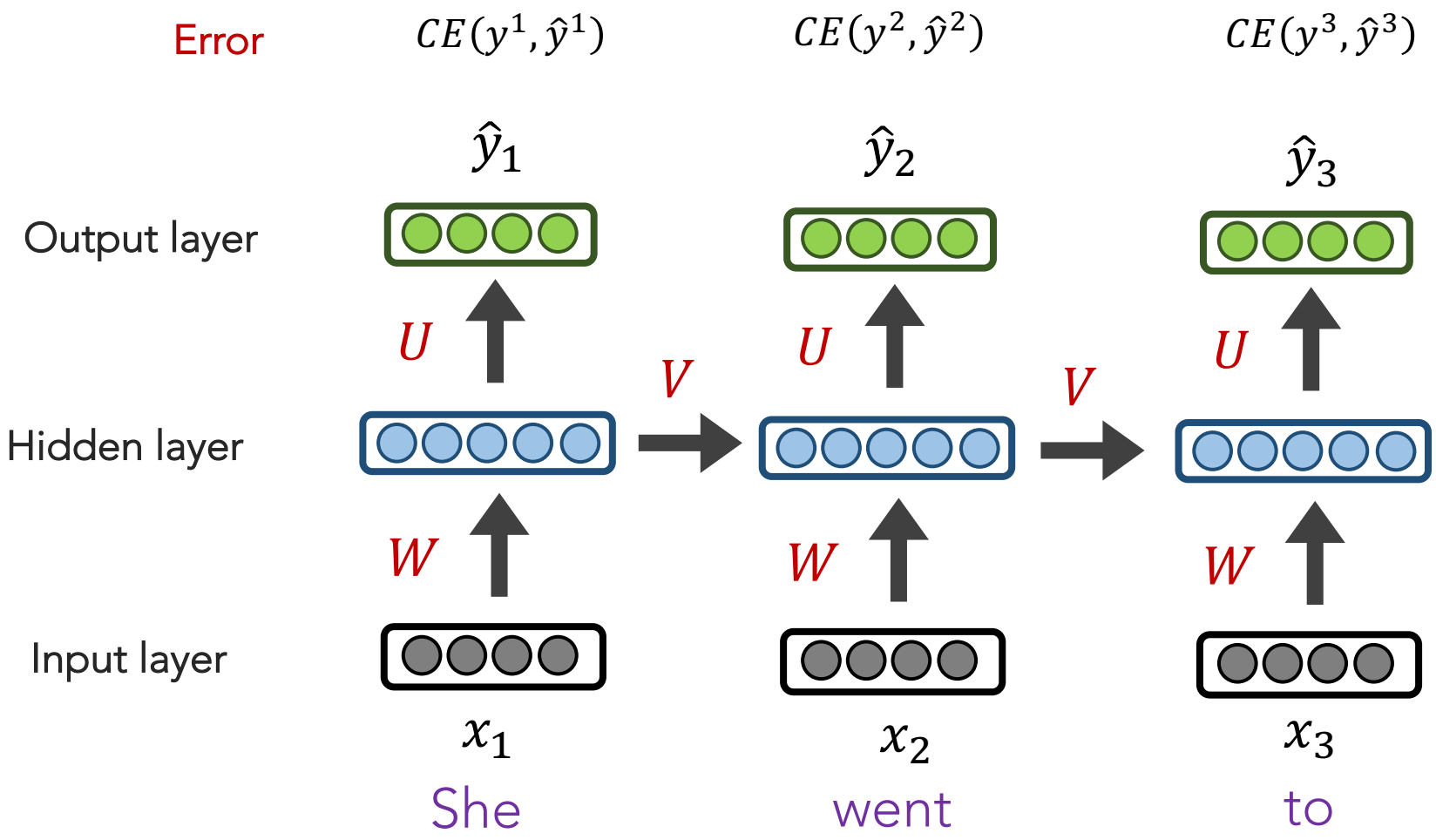
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

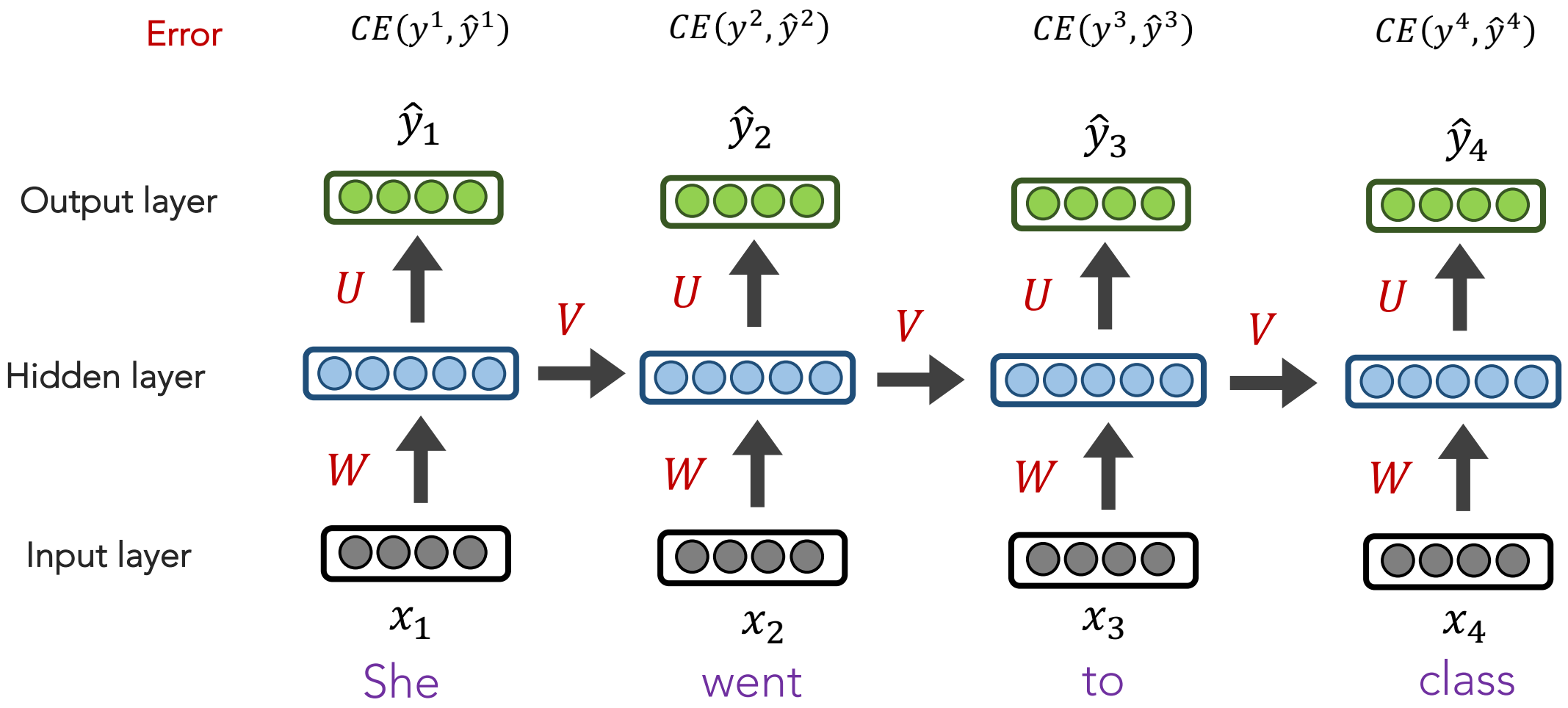
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

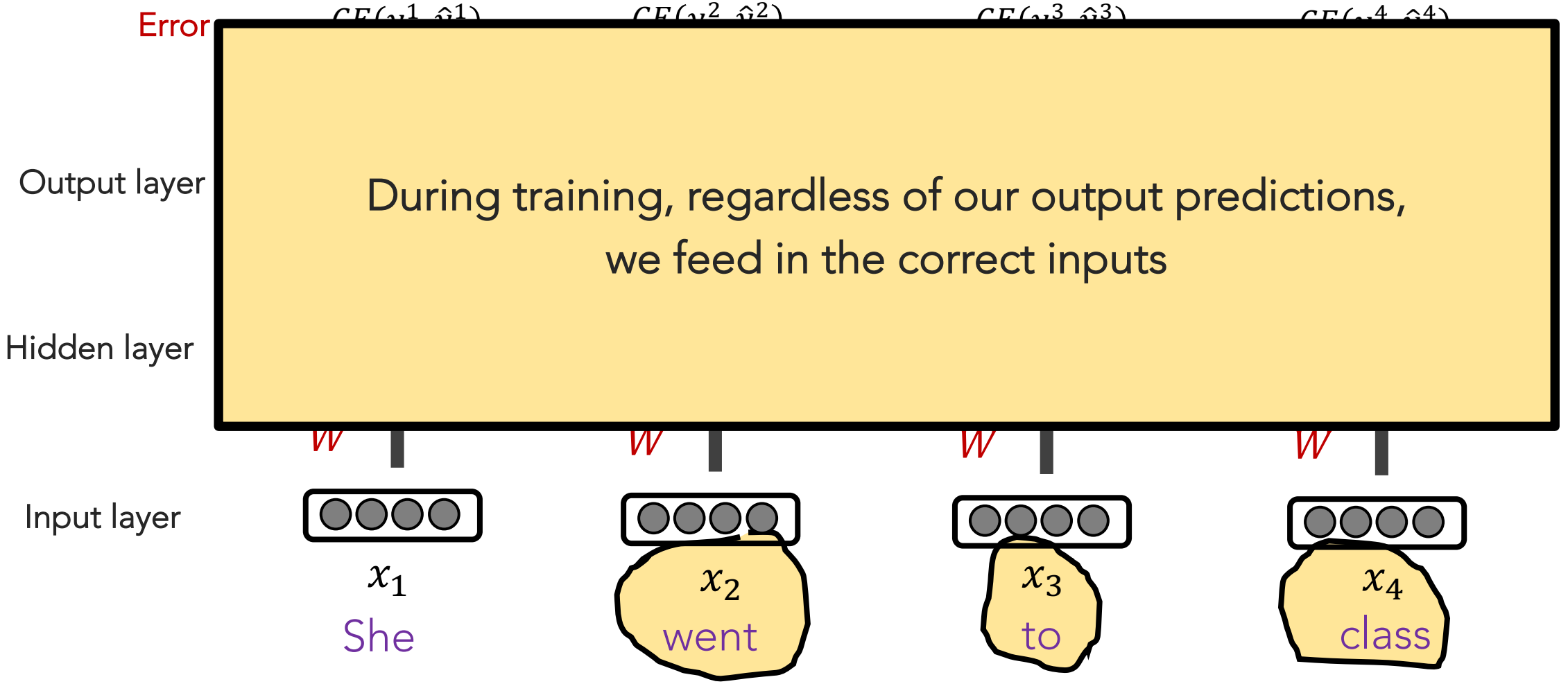
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

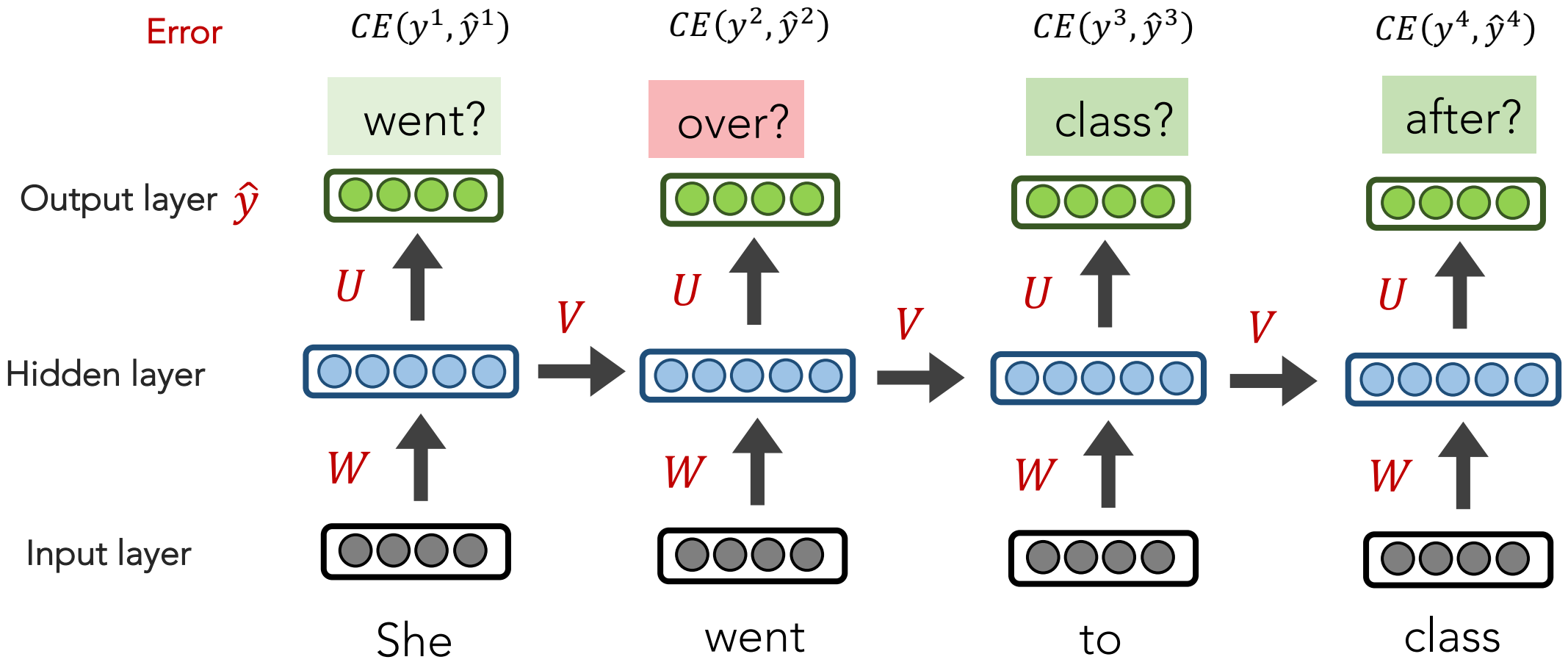
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

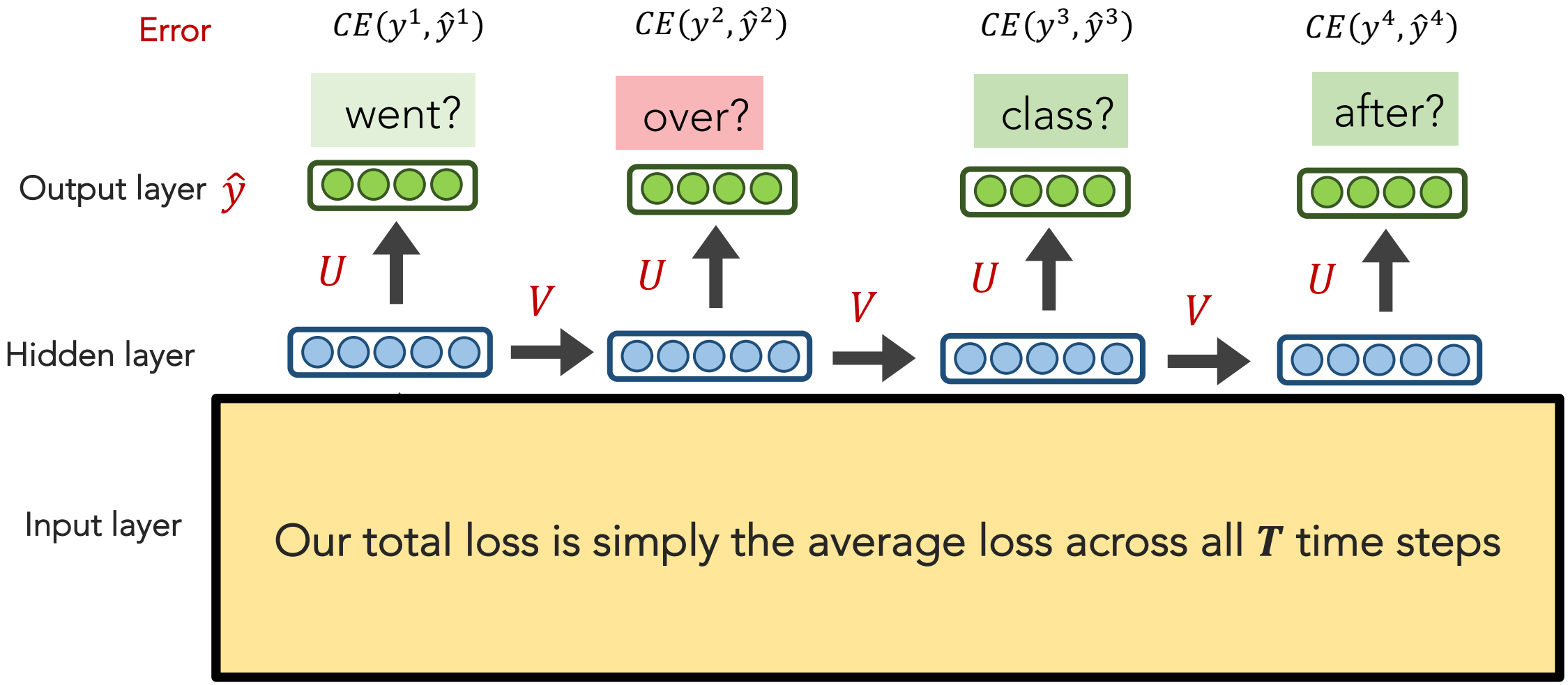
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$

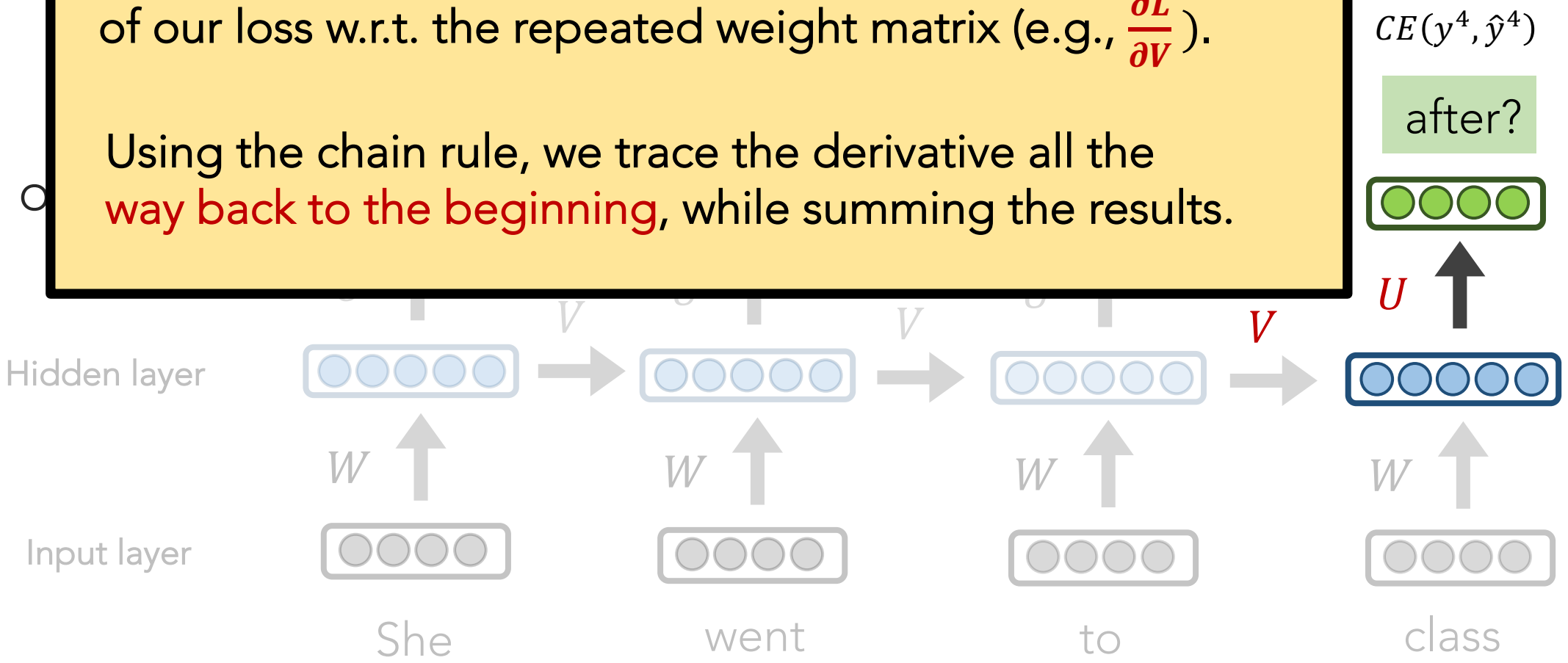


RNN

Training Details

To update our weights (e.g. $\boldsymbol{V}$), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial V}$).

Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.

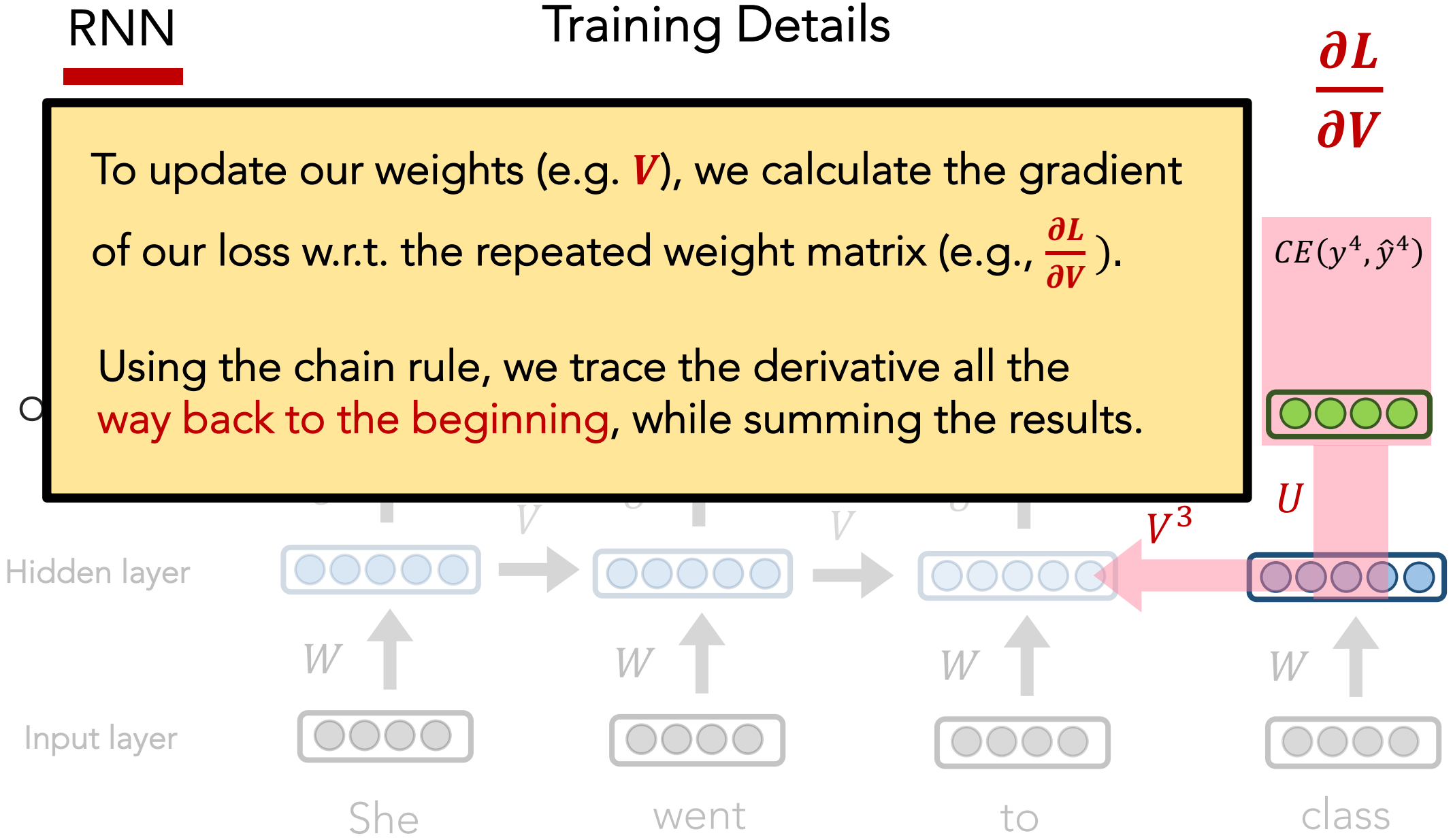


RNN

Training Details

To update our weights (e.g. $\boldsymbol{V}$), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial V}$).

Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.

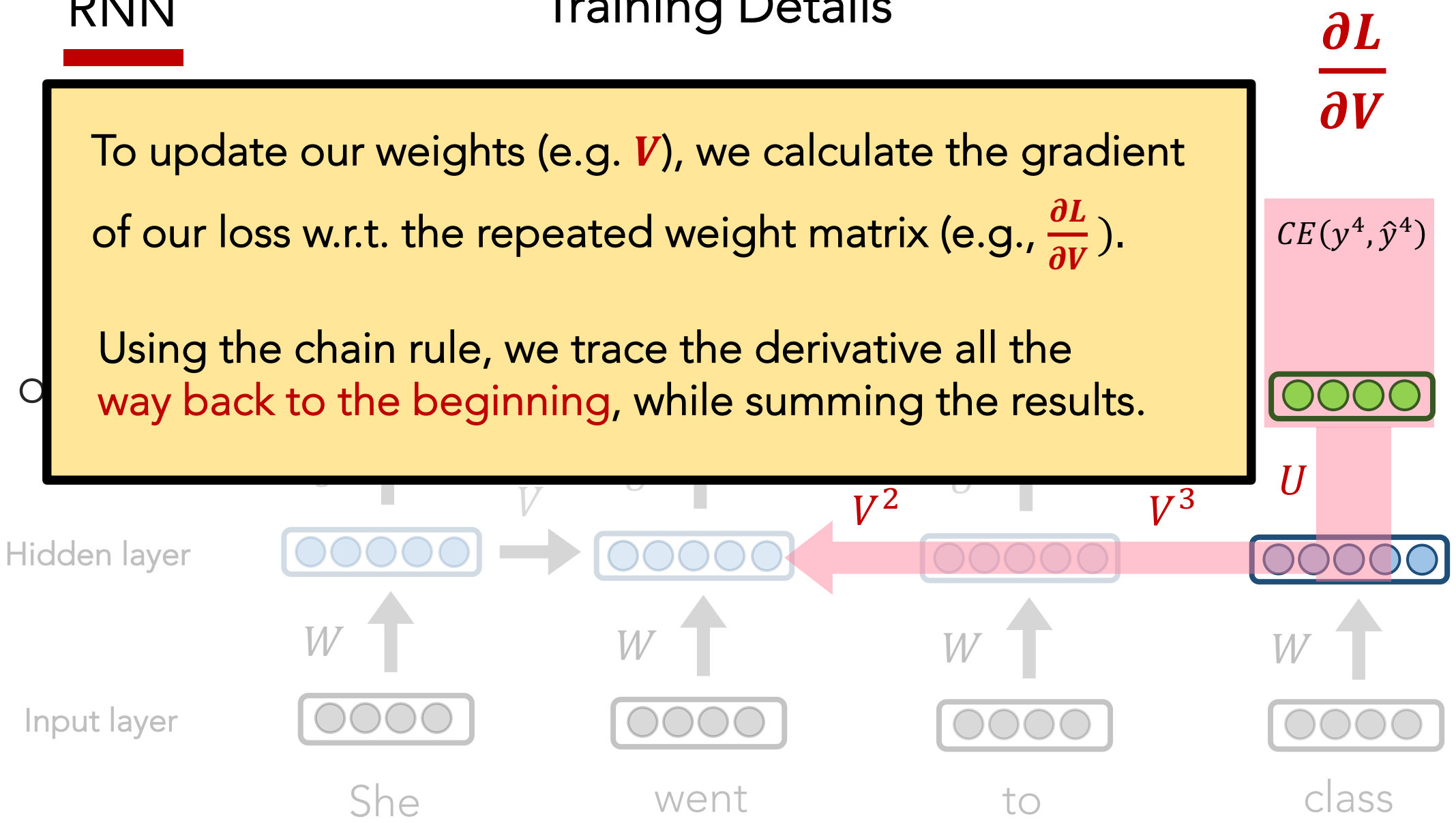


RNN

Training Details

To update our weights (e.g. $\boldsymbol{V}$), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial V}$).

Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.

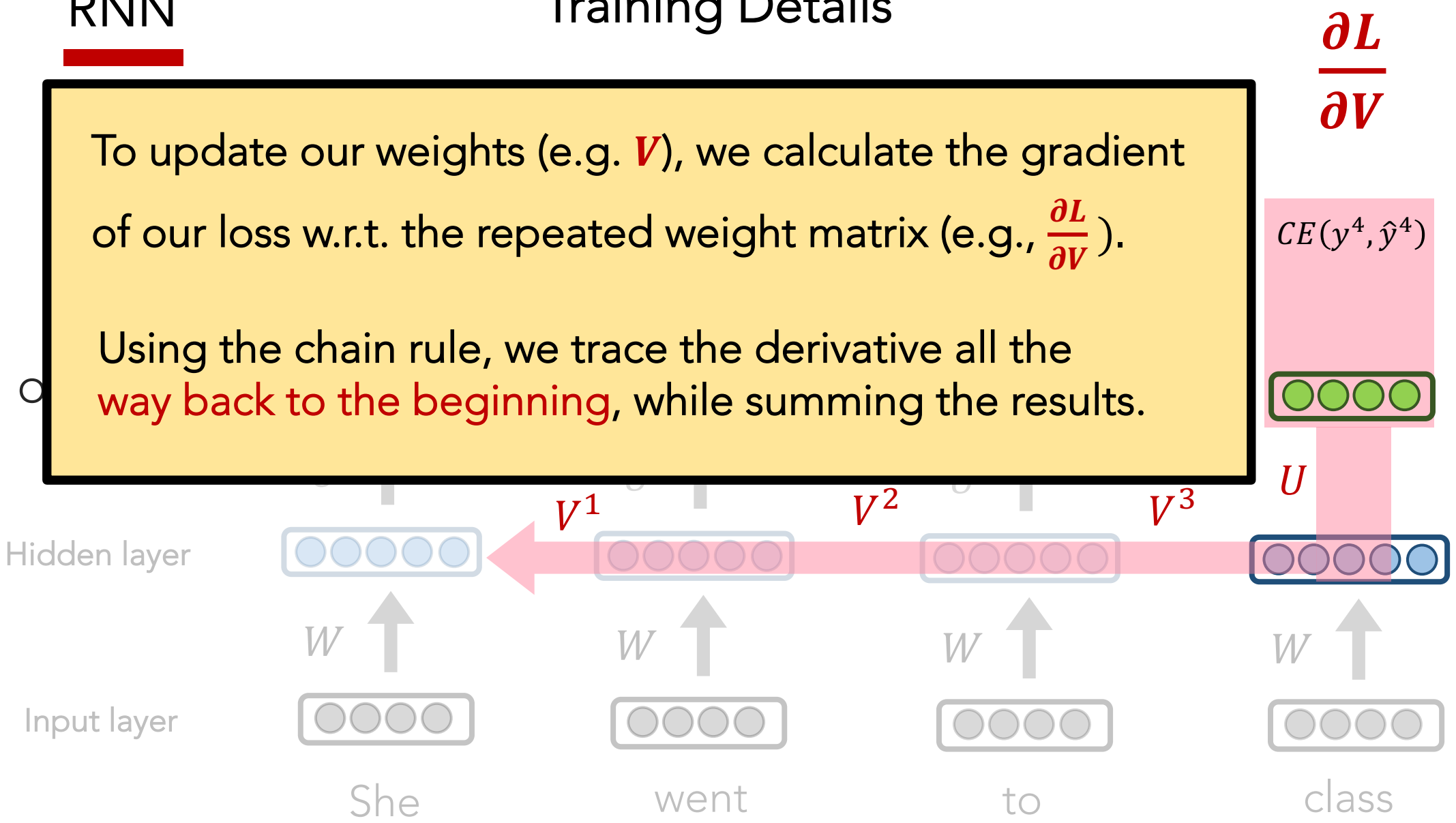


RNN

Training Details

To update our weights (e.g. $\boldsymbol{V}$), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial V}$).

Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.



Training Details

- This backpropagation through time (BPTT) process is **expensive**
- Instead of updating after every timestep, we tend to do so every T steps (e.g., every sentence or paragraph)
- This isn't equivalent to using only a window size T (a la n-grams) because we still have 'infinite memory'

RNN: Generation

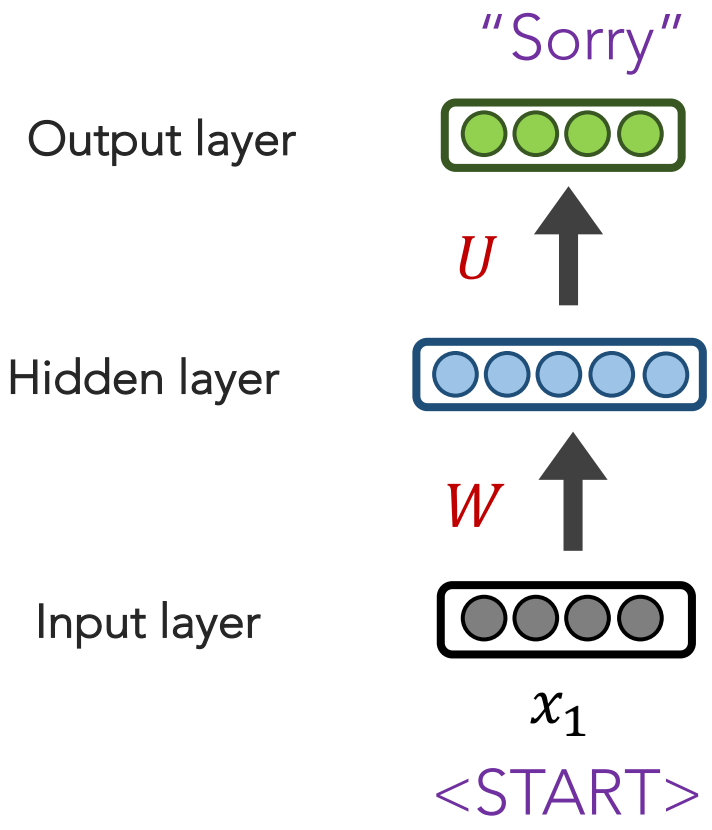
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

Continue until we generate <EOS> symbol.

RNN: Generation

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

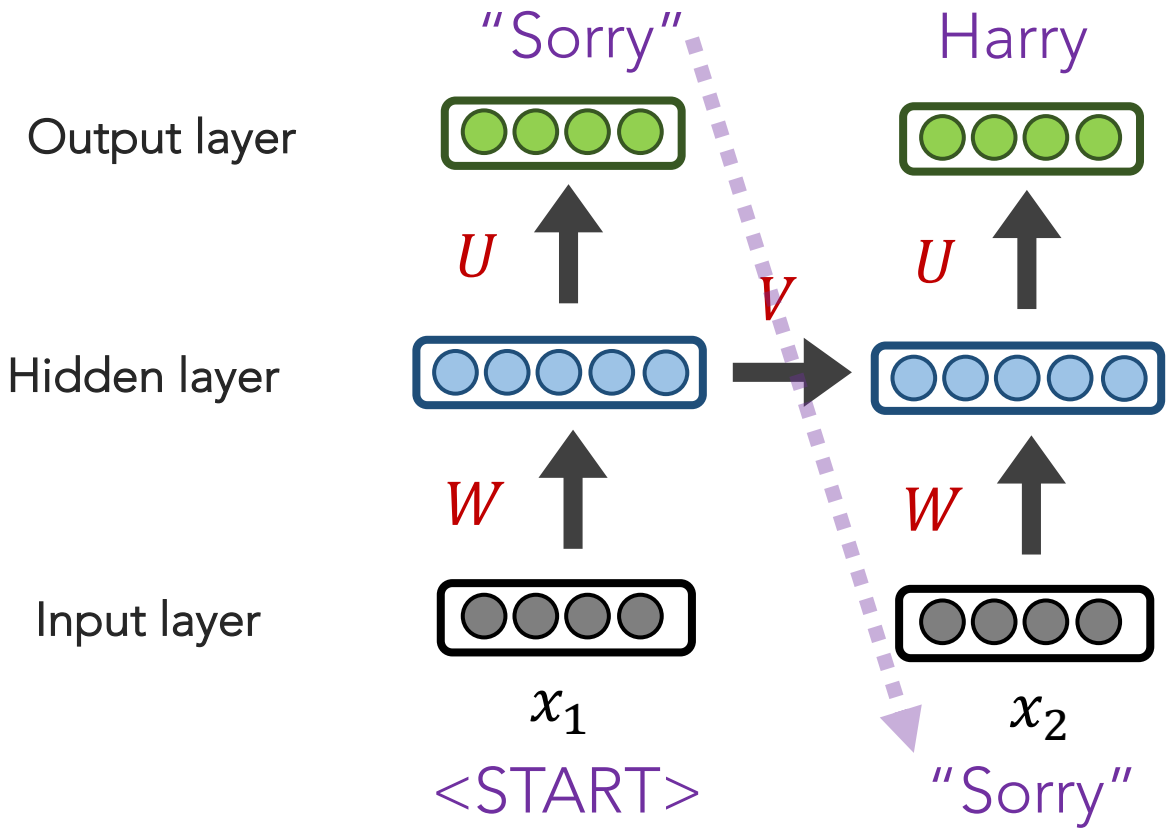
Continue until we generate <EOS> symbol.



RNN: Generation

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

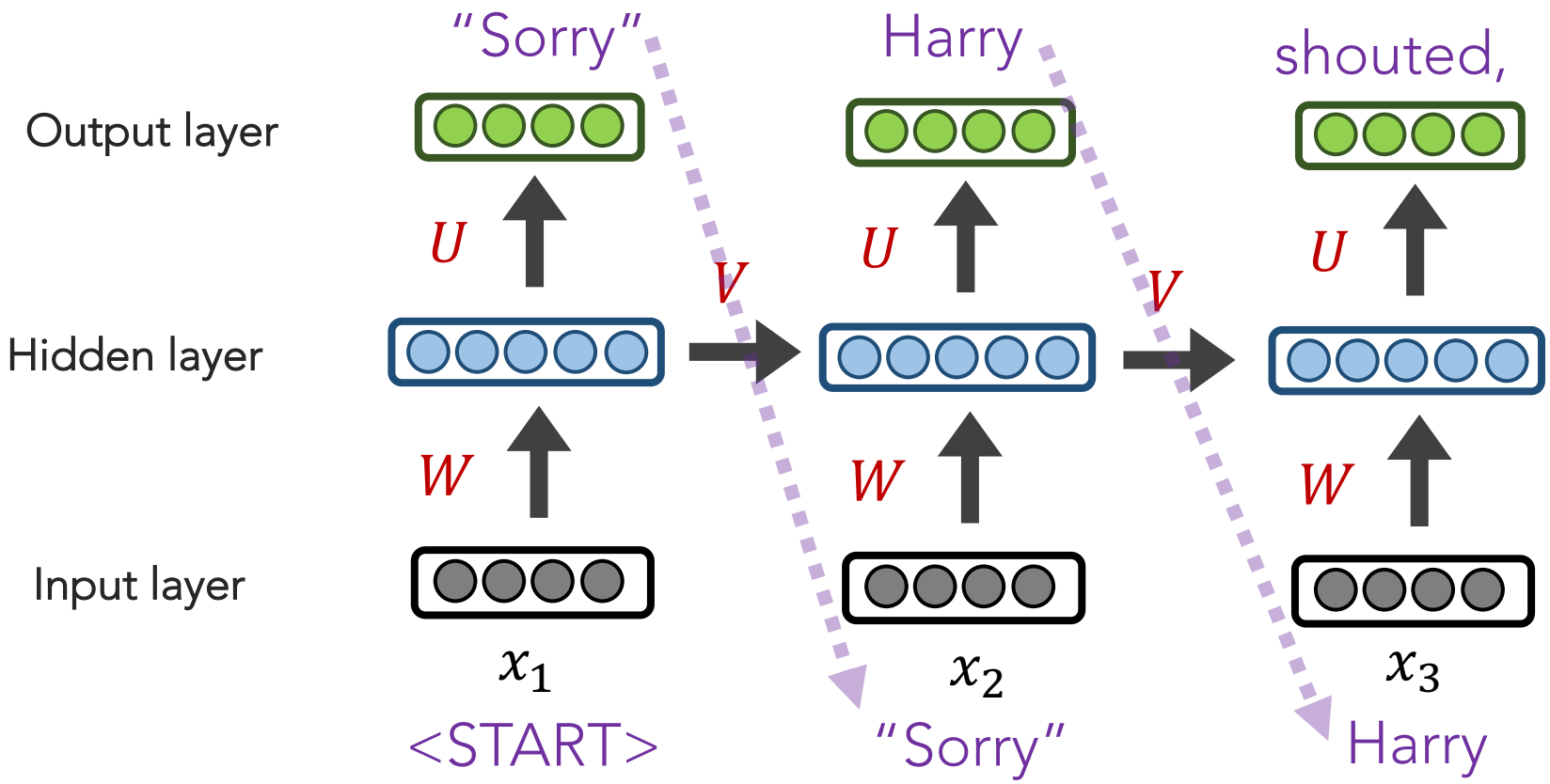
Continue until we generate <EOS> symbol.



RNN: Generation

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

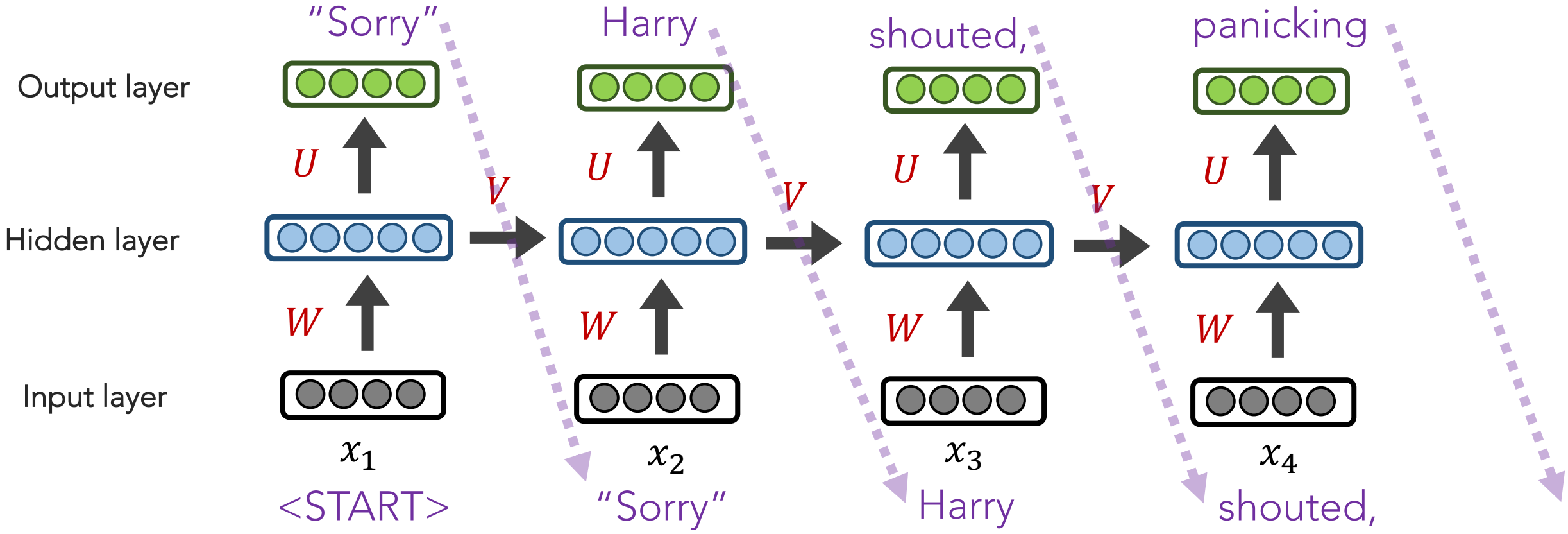
Continue until we generate <EOS> symbol.



RNN: Generation

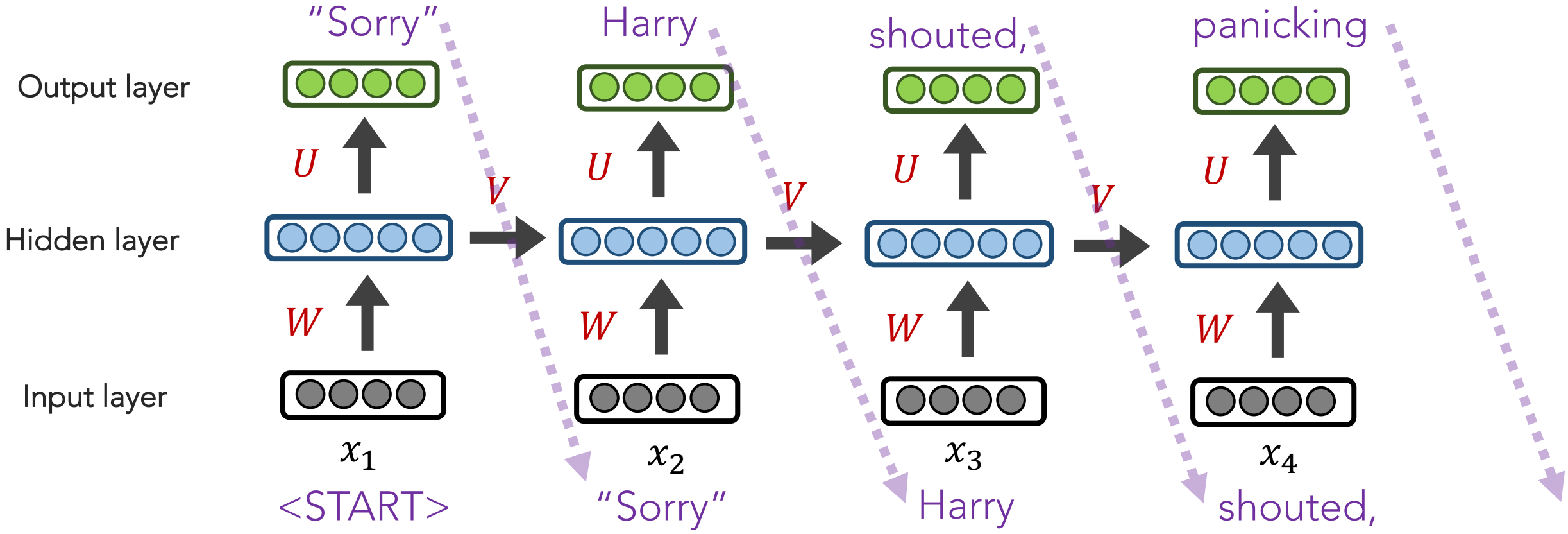
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

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

RNN: Generation

When trained on recipes


```
      Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
      Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten

  Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
  until firm. Serve hot in bodied fresh, mustard, orange and cheese.

  Combine the cheese and salt together the dough in a large skillet; add the ingredients
  and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

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 (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- Has trouble actually making use of long-range context

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)

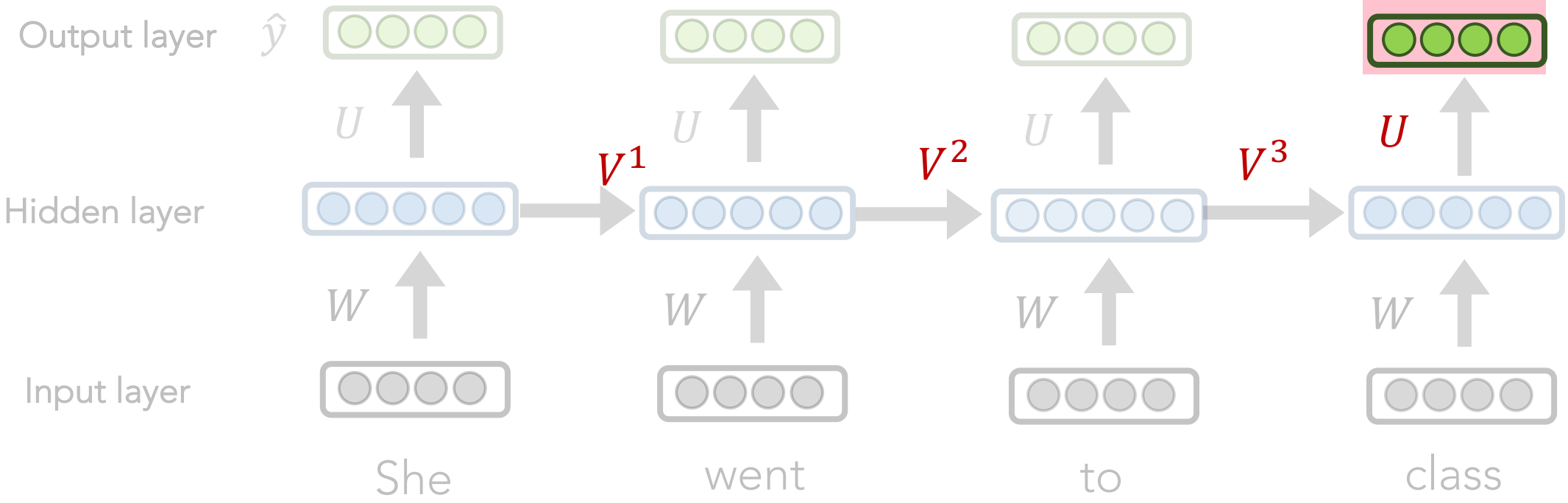
RNN ISSUES?

- Slow to train (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- Has trouble actually making use of long-range context

RNNs: Vanishing and Exploding Gradients

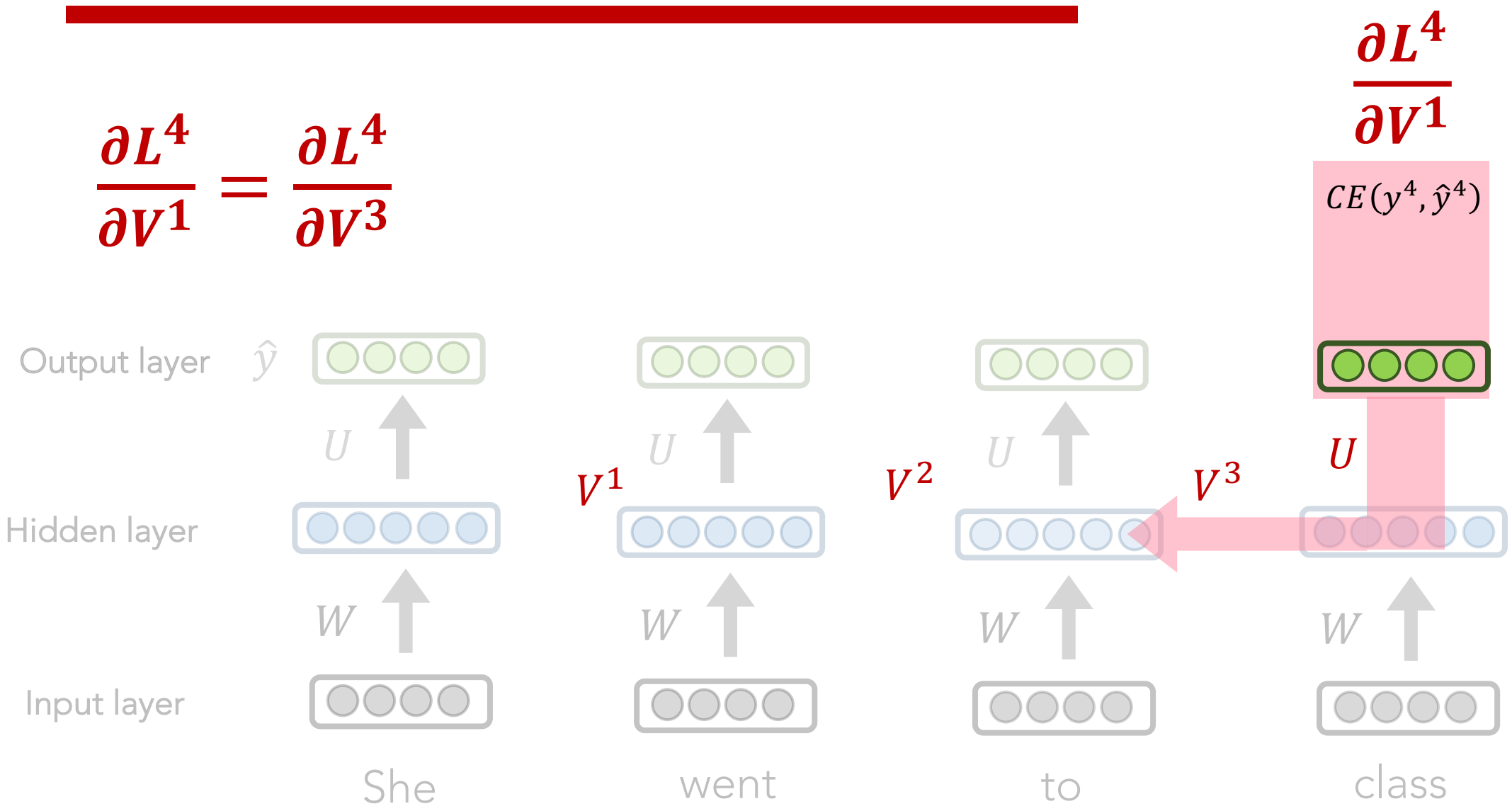
$$\frac{\partial L^4}{\partial V^1} = ?$$

$$\frac{\partial L^4}{\partial V^1}$$



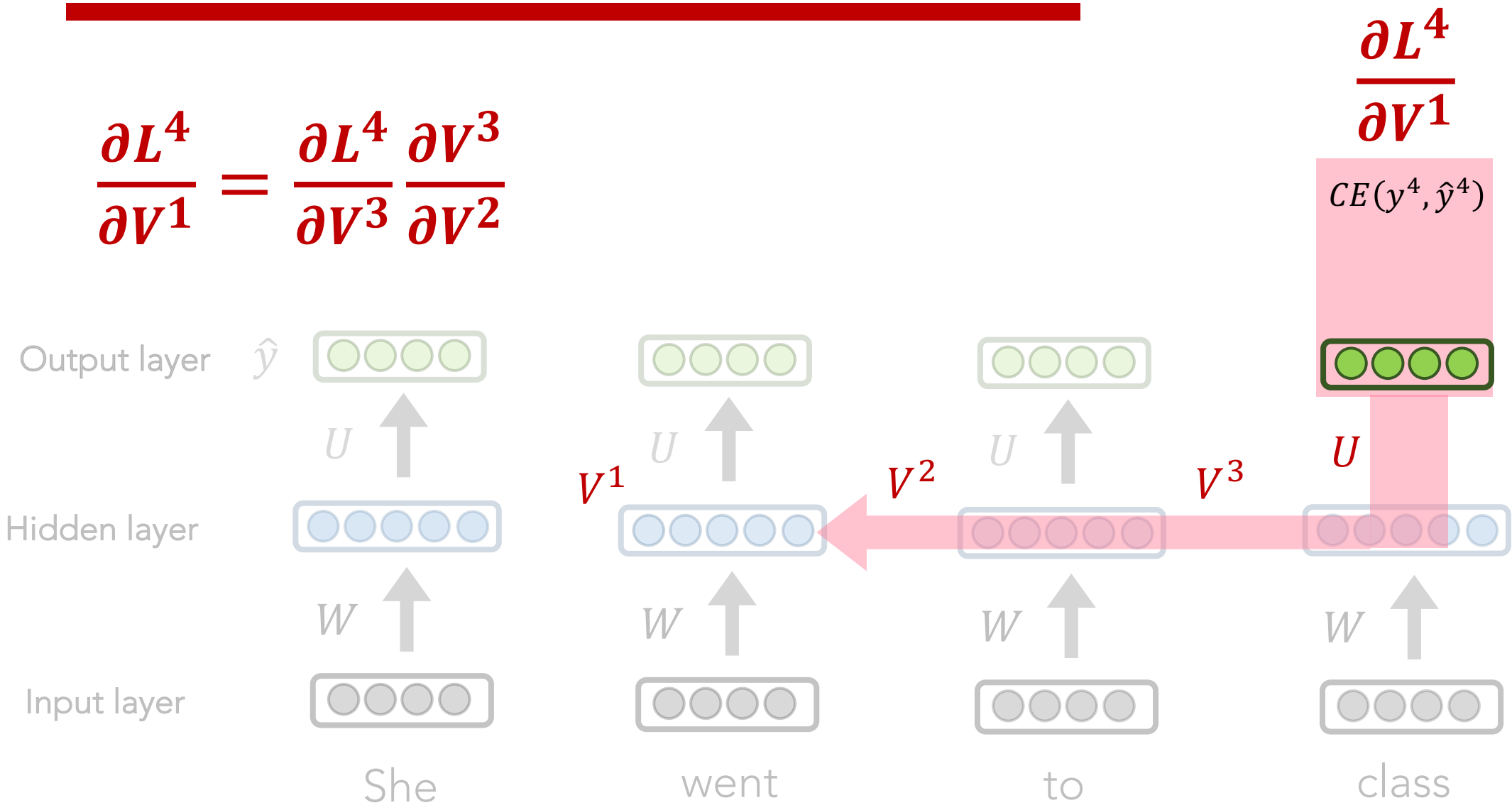
RNNs: Vanishing and Exploding Gradients

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}$$



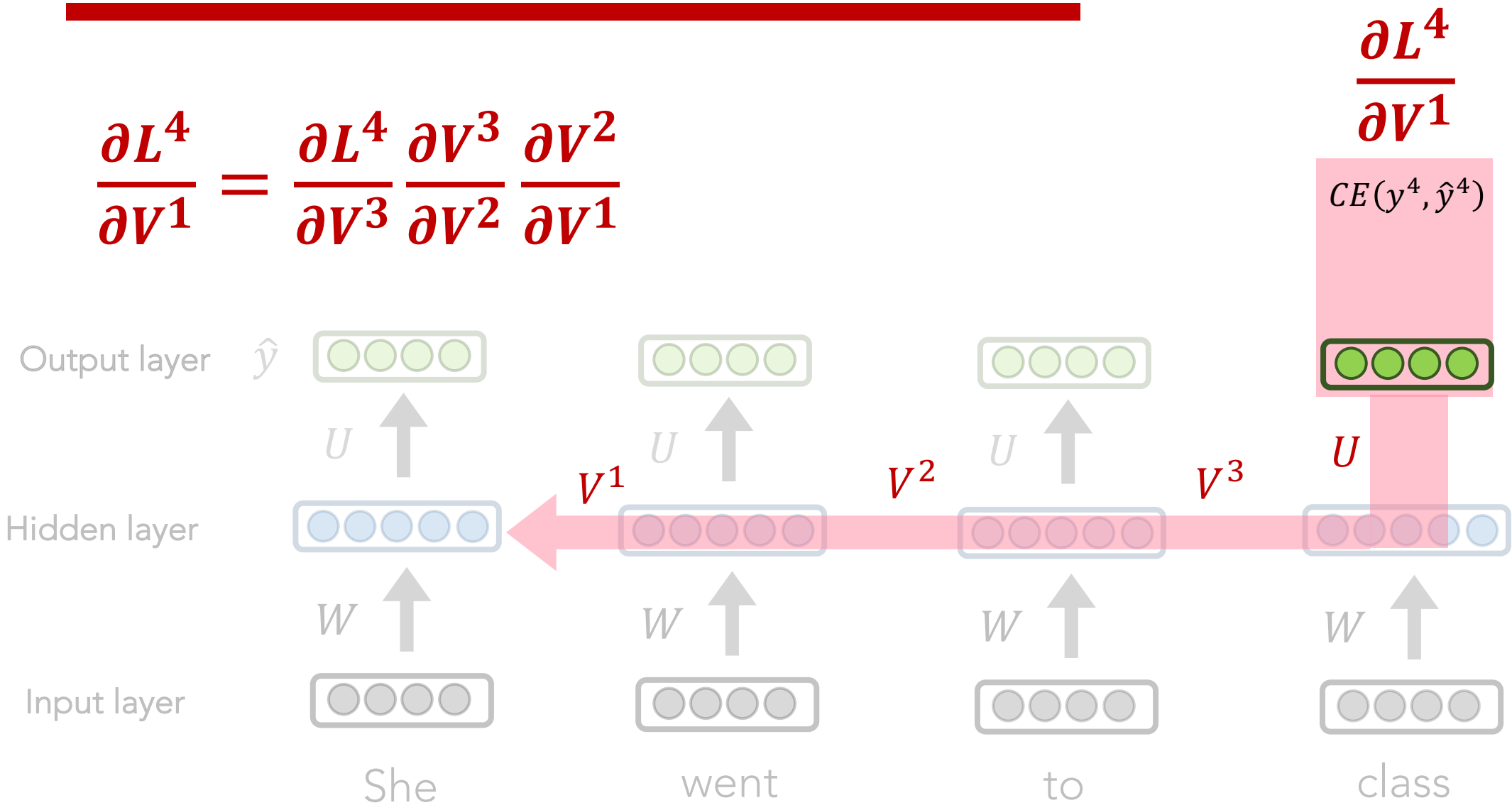
RNNs: Vanishing and Exploding Gradients

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}\frac{\partial V^3}{\partial V^2}$$



RNNs: Vanishing and Exploding Gradients

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}\frac{\partial V^3}{\partial V^2}\frac{\partial V^2}{\partial V^1}$$



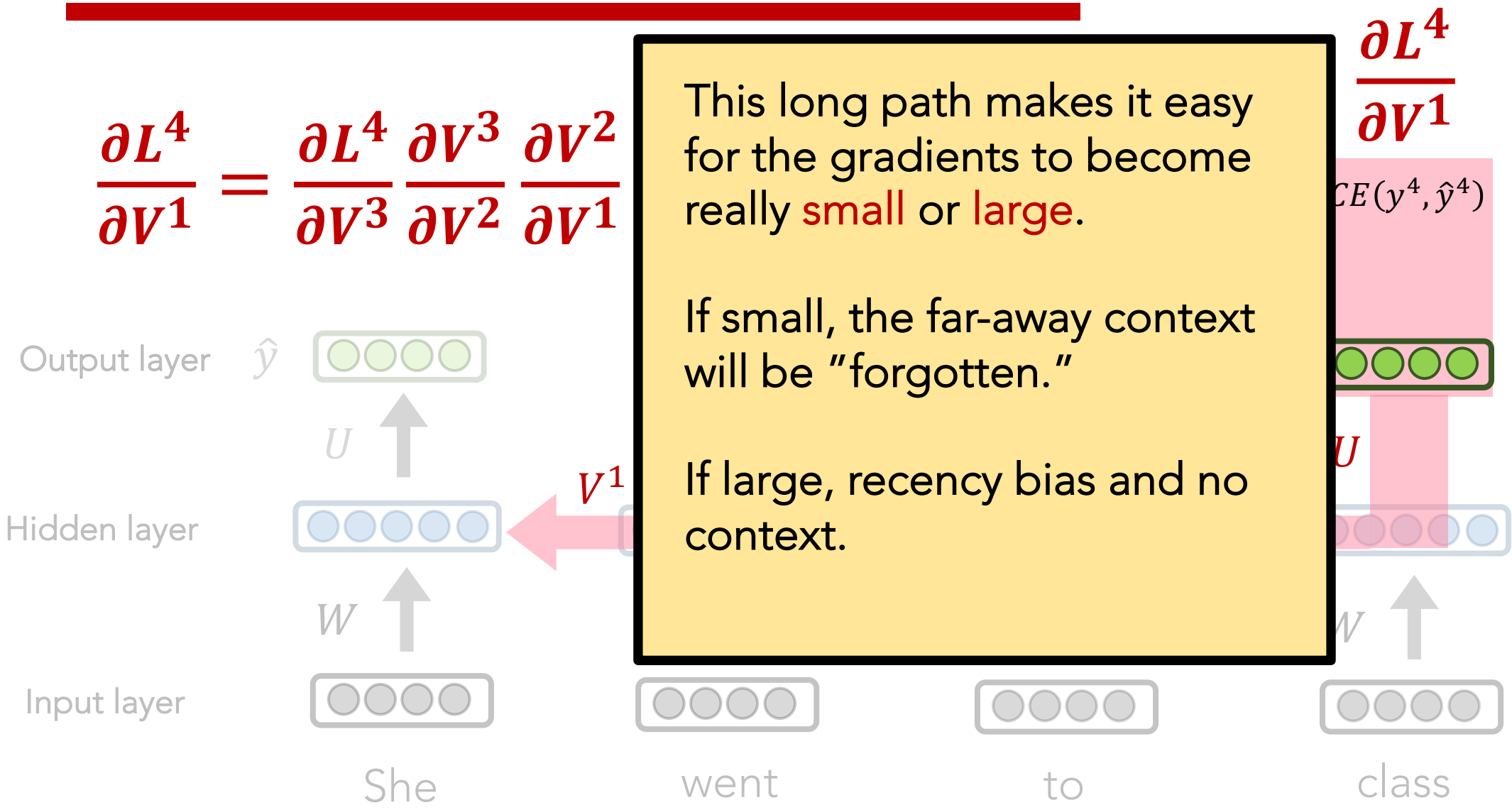
RNNs: Vanishing and Exploding Gradients

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}\frac{\partial V^3}{\partial V^2}\frac{\partial V^2}{\partial V^1}$$

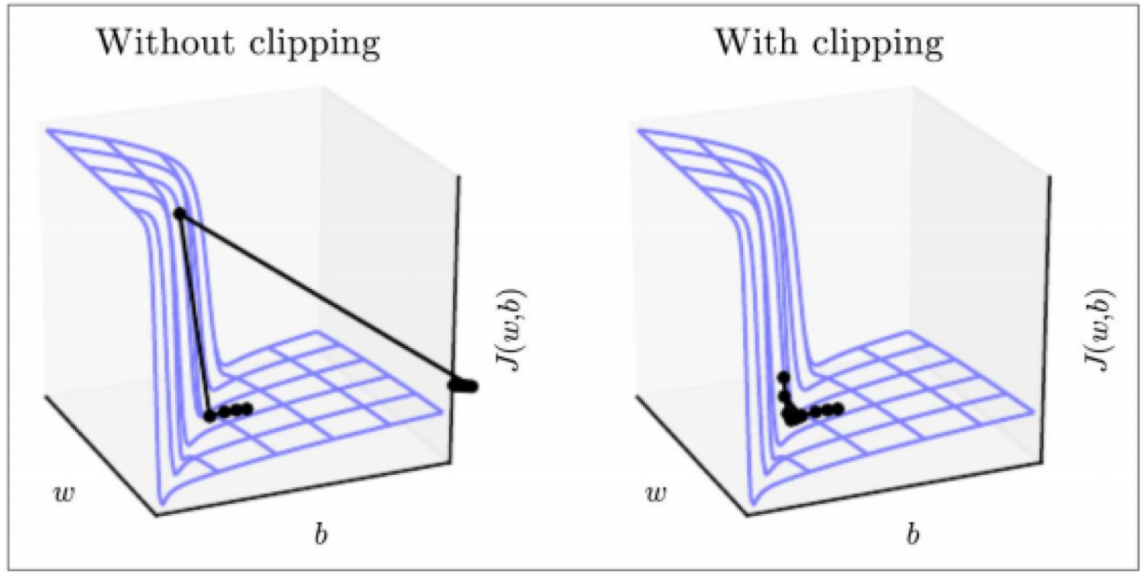
This long path makes it easy for the gradients to become really small or large.

If small, the far-away context will be "forgotten."

If large, recency bias and no context.



Exploding Gradients



Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Source: https://www.deeplearningbook.org/contents/rnn.html

Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

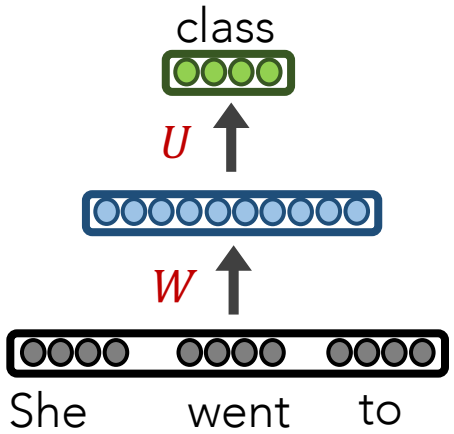
Vanishing Gradients

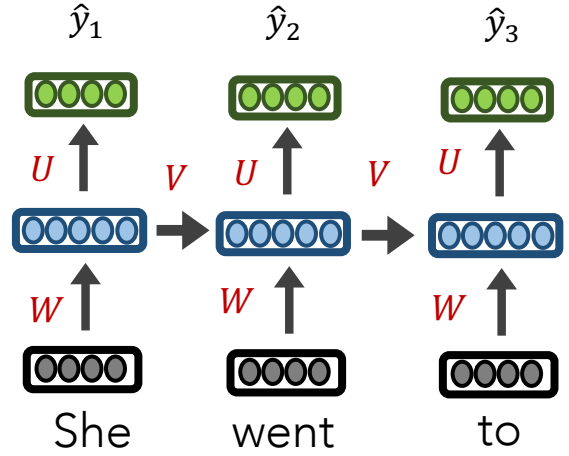
To address RNNs' finnicky nature with long-range context, we turn to an RNN variant named LSTMs (long short-term memory)

But first, let's recap what we've learned so far

Sequential Modelling (so far)

$$P(\text{went}|She) = \frac{\text{count}(She\ went)}{\text{count}(She)}$$





n-grams

- Basic counts; fast
- Fixed window size
- Sparsity & storage issues
- Not robust

FFNN

- Kind of robust… almost
- Fixed window size
- Weirdly handles context positions
- No "memory" of past

RNN

- Handles infinite context (in theory)
- Robust to rare words
- Slow
- Difficulty with long context

Background

Language Modelling

RNNs/LSTMs +ELMo

As a reminder, this is where we are

Seq2Seq +Attention

Transformers +BERT

Conclusions

Long short-term memory (LSTM)

- A type of RNN that is designed to better handle **long-range dependencies**
- In "vanilla" RNNs, the hidden state is perpetually being rewritten
- In addition to a traditional **hidden state** h, let's have a dedicated **memory cell** c for long-term events. More power to relay sequence info.

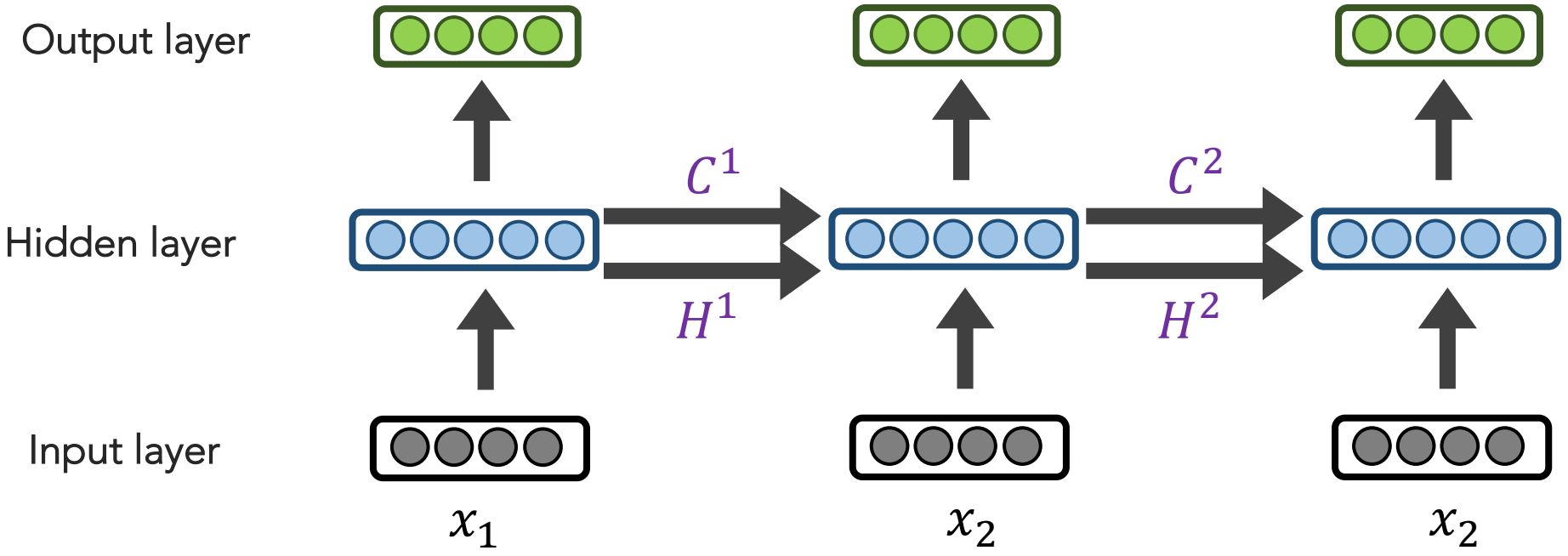
Long short-term memory (LSTM)

At each each time step t, we have a hidden state h^t and cell state c^t:

- Both are vectors of length **n**
- cell state c^t stores long-term info
- At each time step t, the LSTM erases, writes, and reads information from the cell c^t
- c^t never undergoes a nonlinear activation though, just – and +

Long short-term memory (LSTM)

C and H relay long- and short-term memory to the hidden layer, respectively. Inside the hidden layer, there are many weights.



Inside an LSTM Hidden Layer

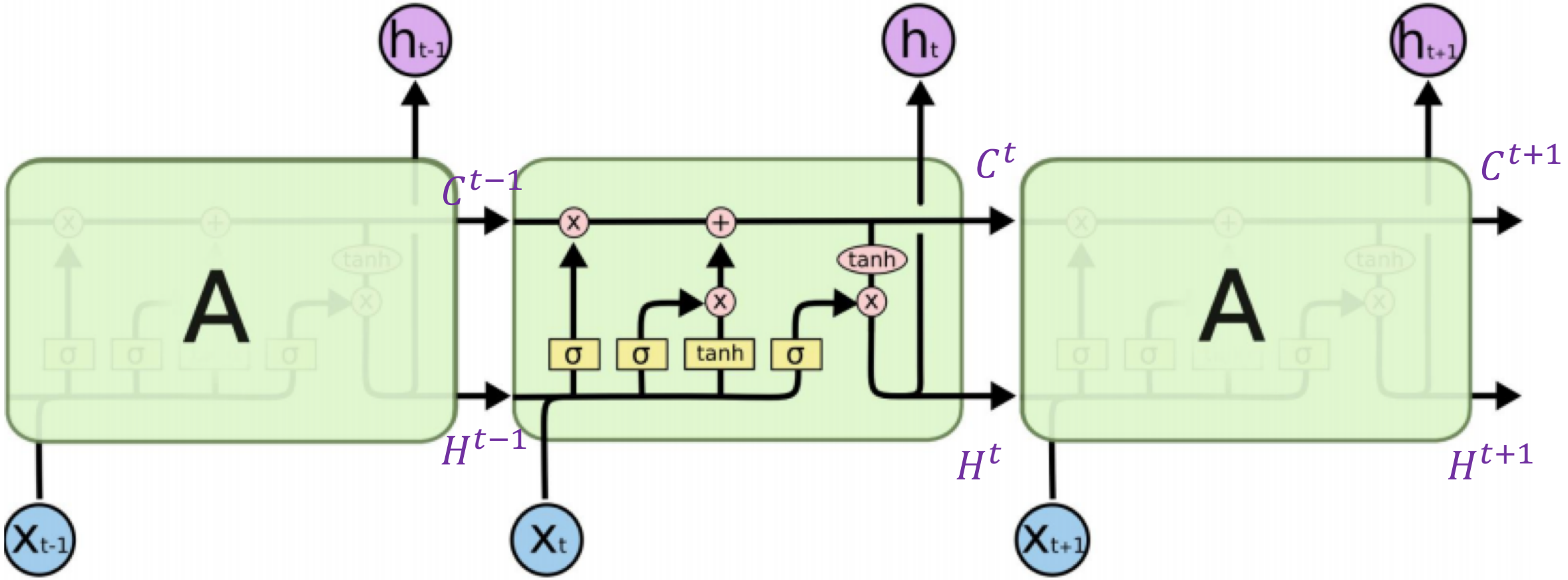


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

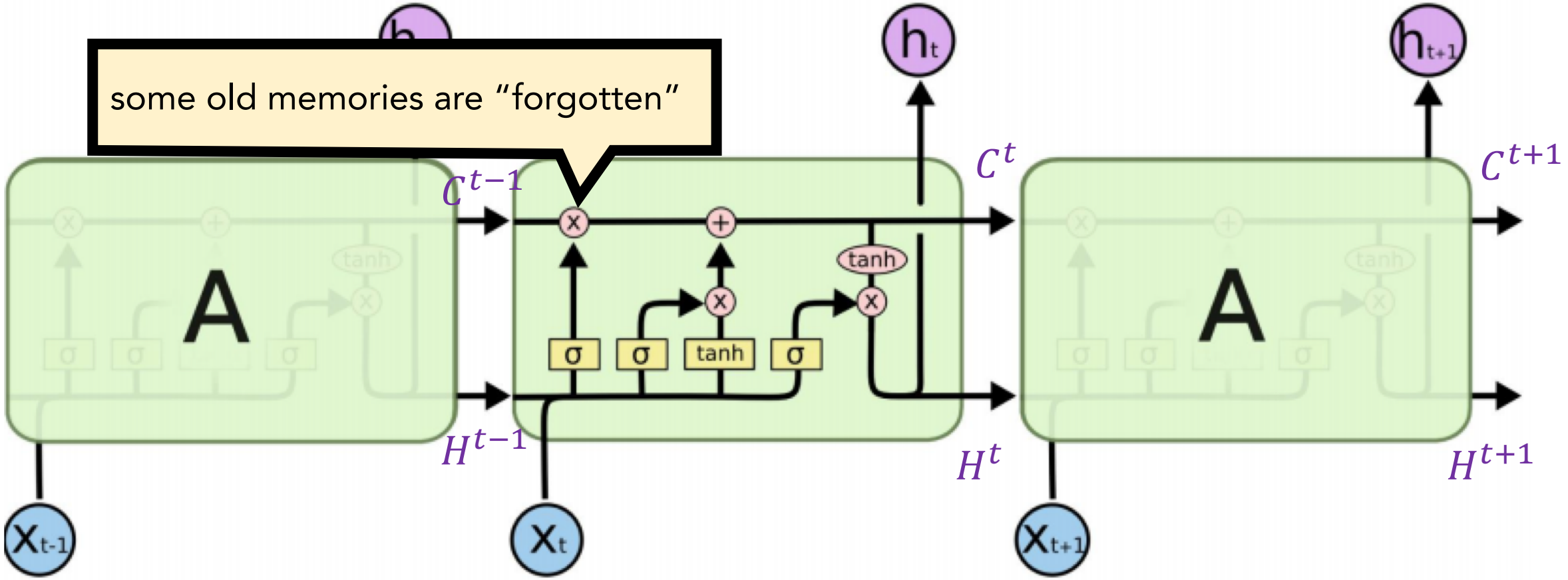


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

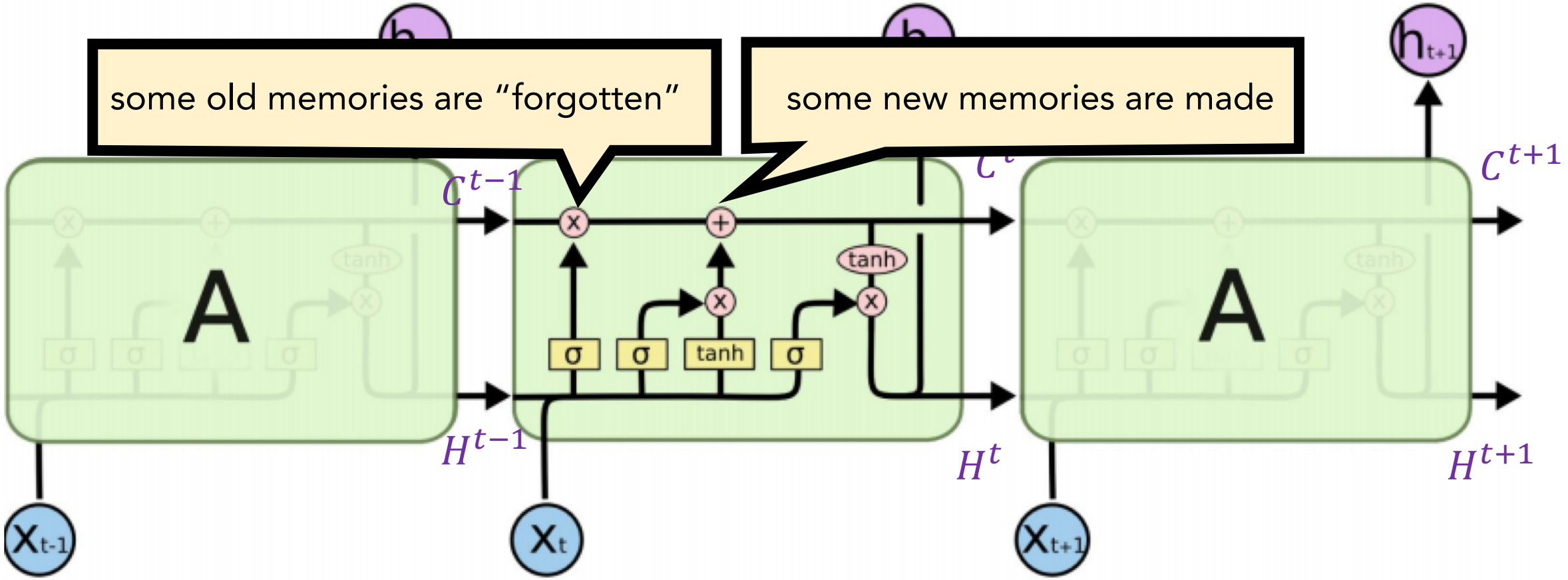


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

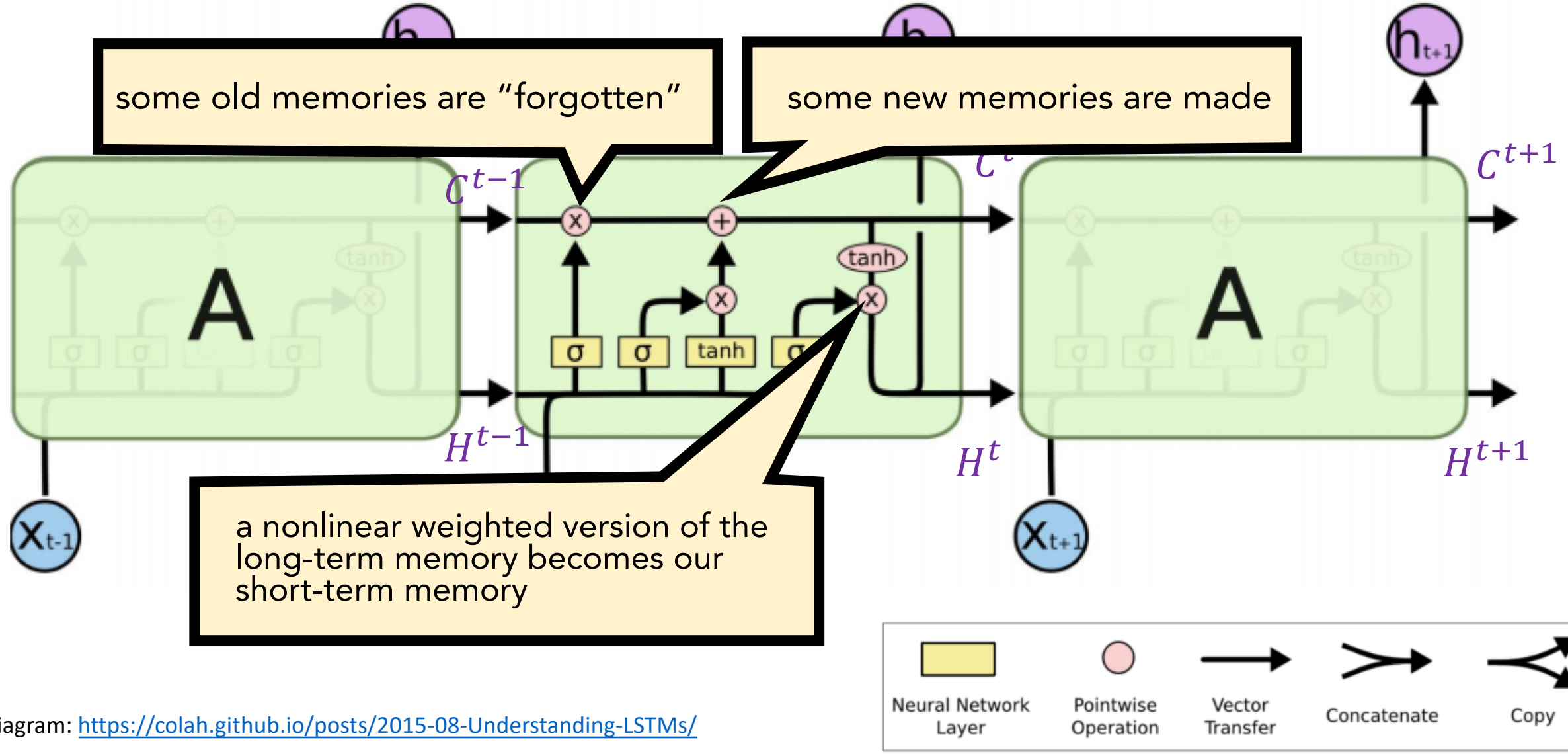


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

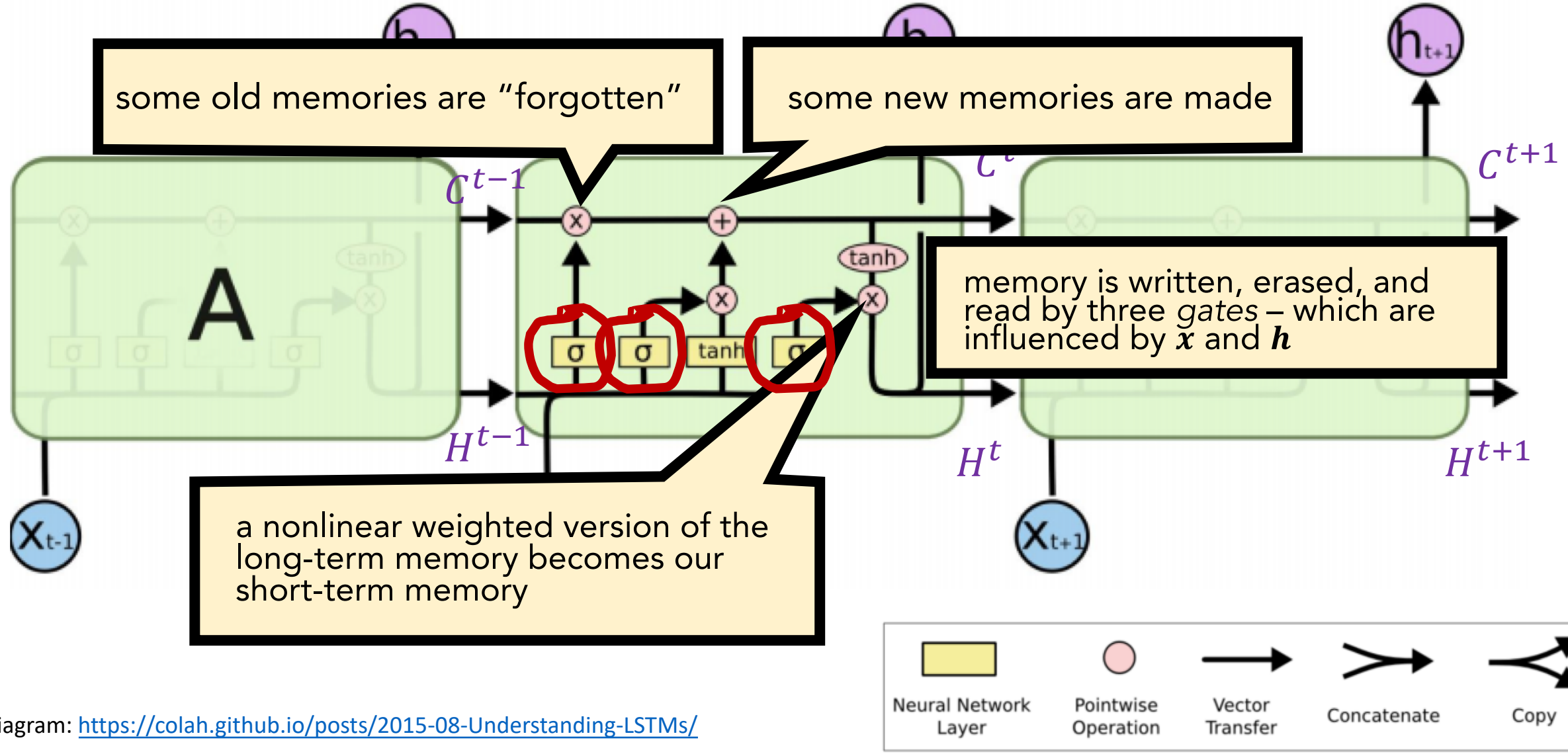


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short-term memory (LSTM)

It's still possible for LSTMs to suffer from vanishing/exploding gradients, but it's way less likely than with vanilla RNNs:

- If RNNs wish to preserve info over long contexts, it must delicately find a recurrent weight matrix W_h that isn't too large or small
- However, LSTMs have 3 separate mechanism that adjust the flow of information (e.g., forget gate, if turned off, will preserve all info)

Long short-term memory (LSTM)

LSTM STRENGTHS?

- Almost always outperforms vanilla RNNs
- Captures long-range dependencies shockingly well

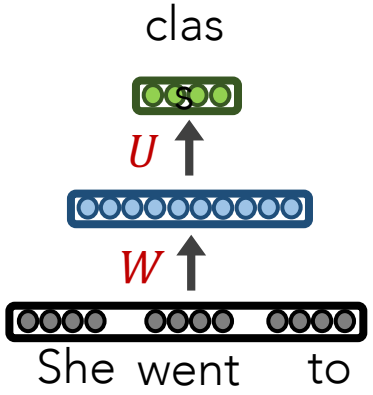
LSTM ISSUES?

- Has more weights to learn than vanilla RNNs; thus,
- Requires a moderate amount of training data (otherwise, vanilla RNNs are better)
- Can still suffer from vanishing/exploding gradients

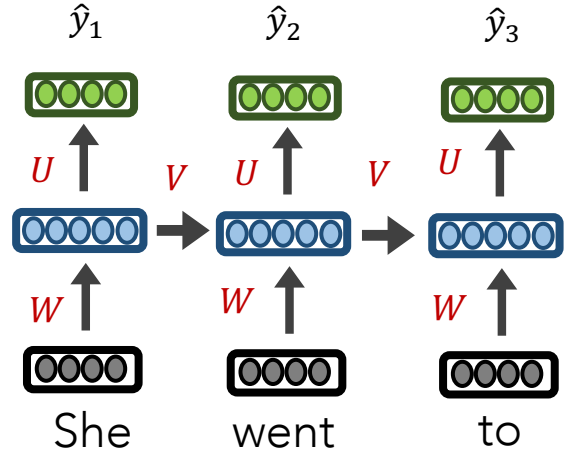
Sequential Modelling

$$P(\text{went}|She) = \frac{\text{count}(She\ went)}{\text{count}(She)}$$

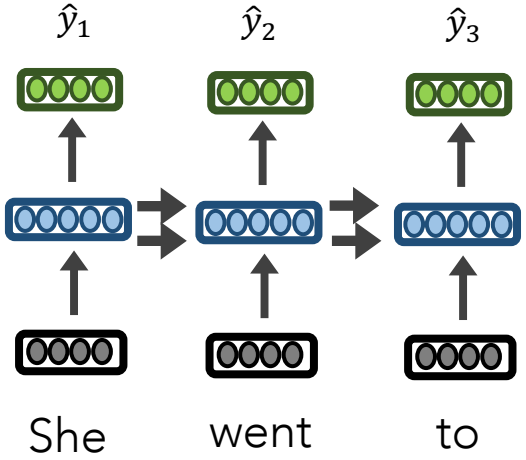
n-grams



FFNN



RNN



LSTM

Sequential Modelling

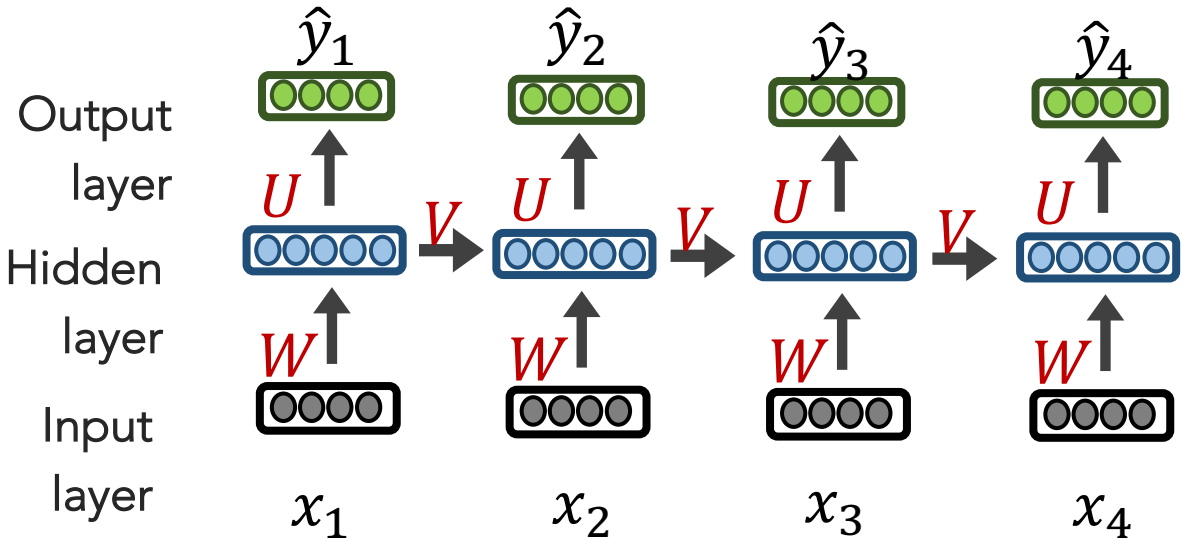
IMPORTANT

If your goal isn't to predict the next item in a sequence, and you rather do some other classification or regression task using the sequence, then you can:

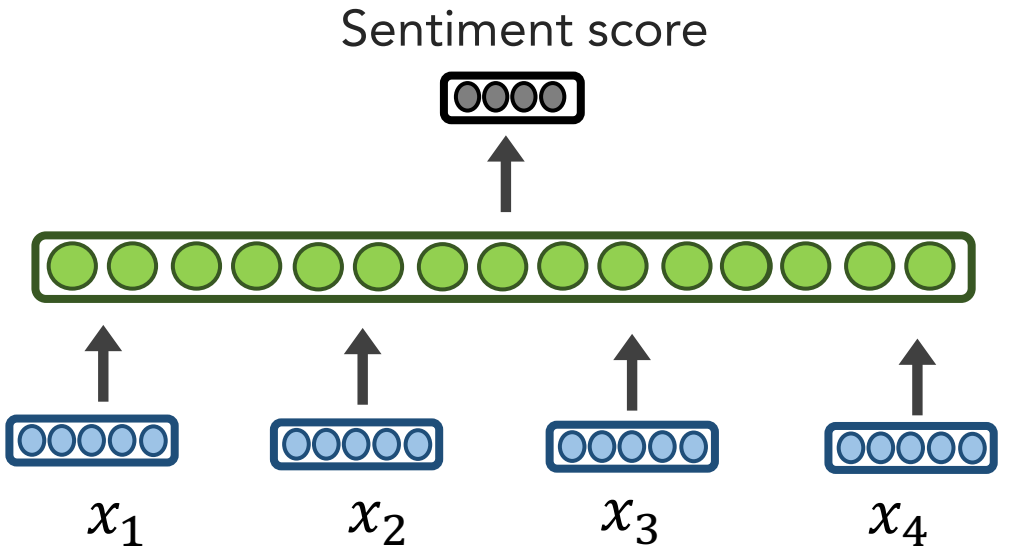
- Train an aforementioned model (e.g., LSTM) as a language model
- Use the **hidden layers** that correspond to each item in your sequence

Sequential Modelling

1. Train LM to learn hidden layer embeddings

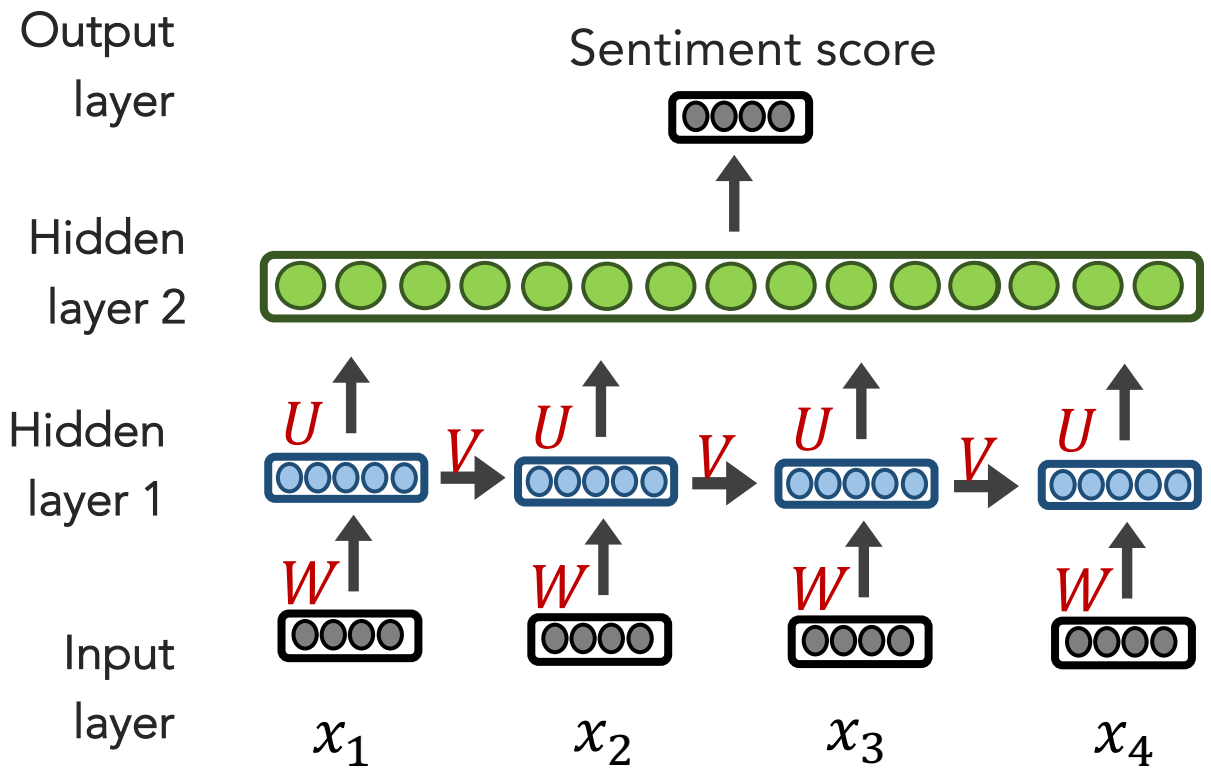


2. Use hidden layer embeddings for other tasks



Sequential Modelling

Or jointly learn hidden embeddings toward a particular task (end-to-end)



You now have the foundation for modelling sequential data.

Most state-of-the-art advances are based on those core RNN/LSTM ideas. But, with tens of thousands of researchers and hackers exploring deep learning, there are many tweaks that haven proven useful.

(This is where things get crazy.)

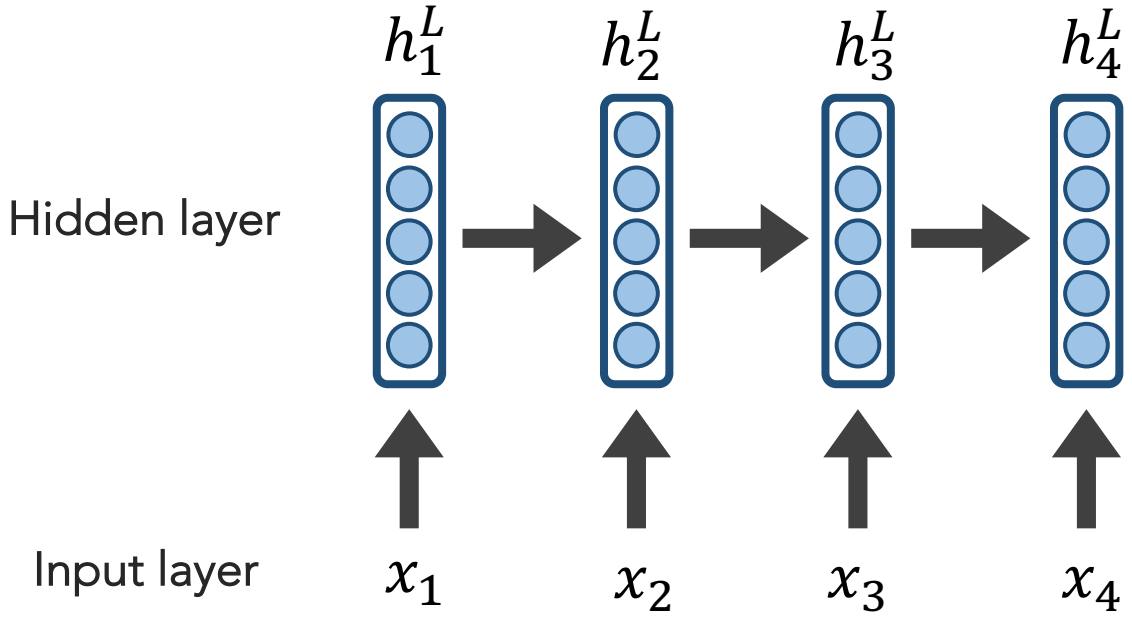
RNN Extensions: Bi-directional LSTMs

RNNs/LSTMs use the left-to-right context and sequentially process data.

If you have full access to the data at testing time, why not make use of the flow of information from right-to-left, also?

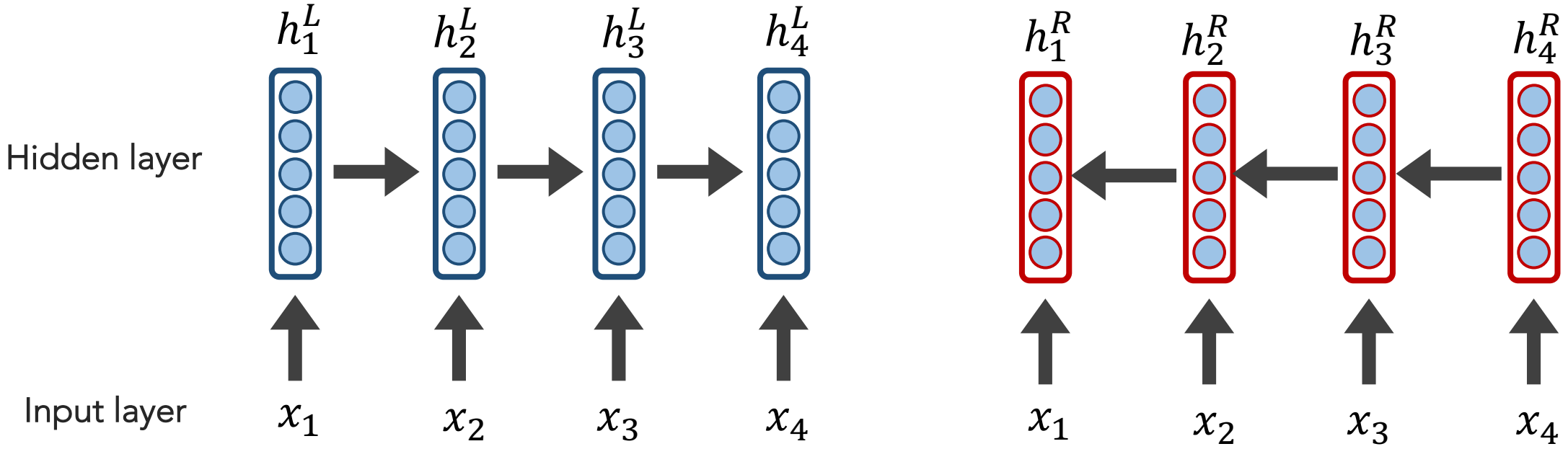
RNN Extensions: Bi-directional LSTMs

For brevity, let's use the follow schematic to represent an RNN

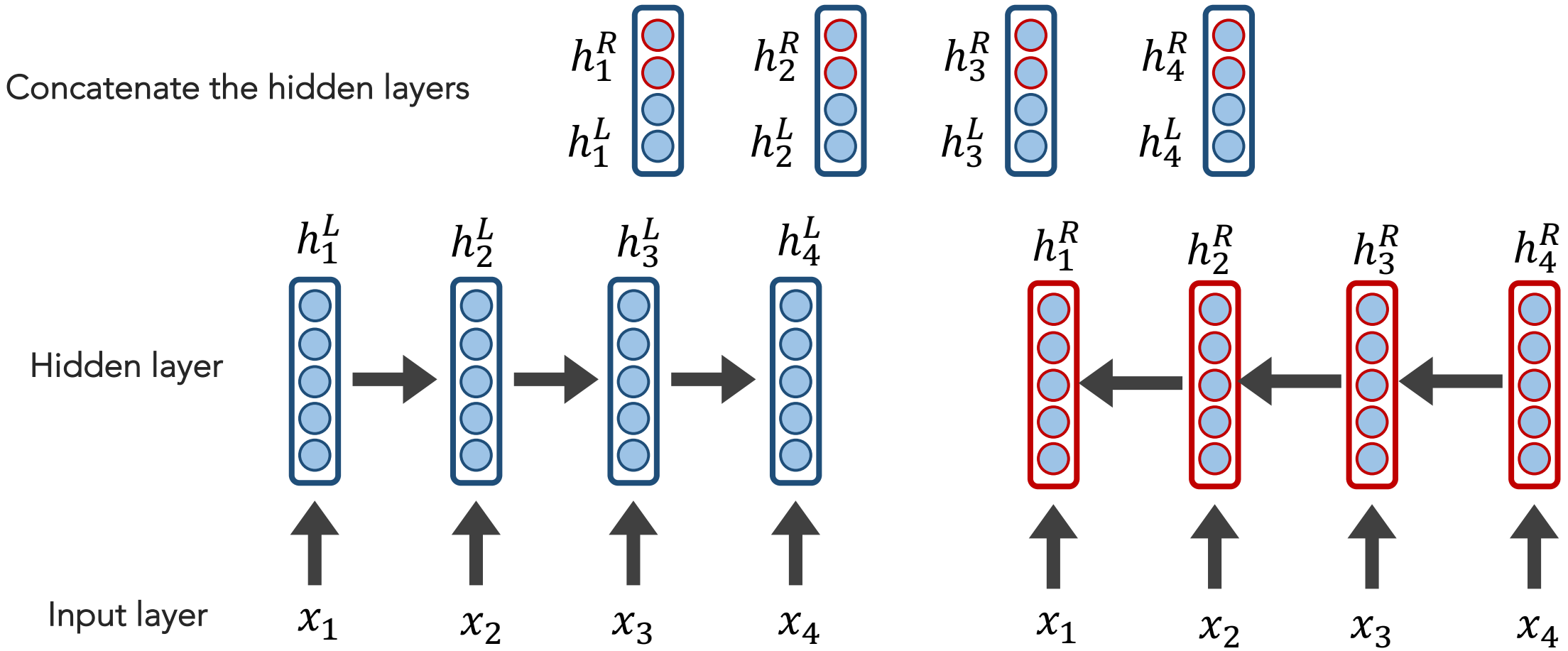


RNN Extensions: Bi-directional LSTMs

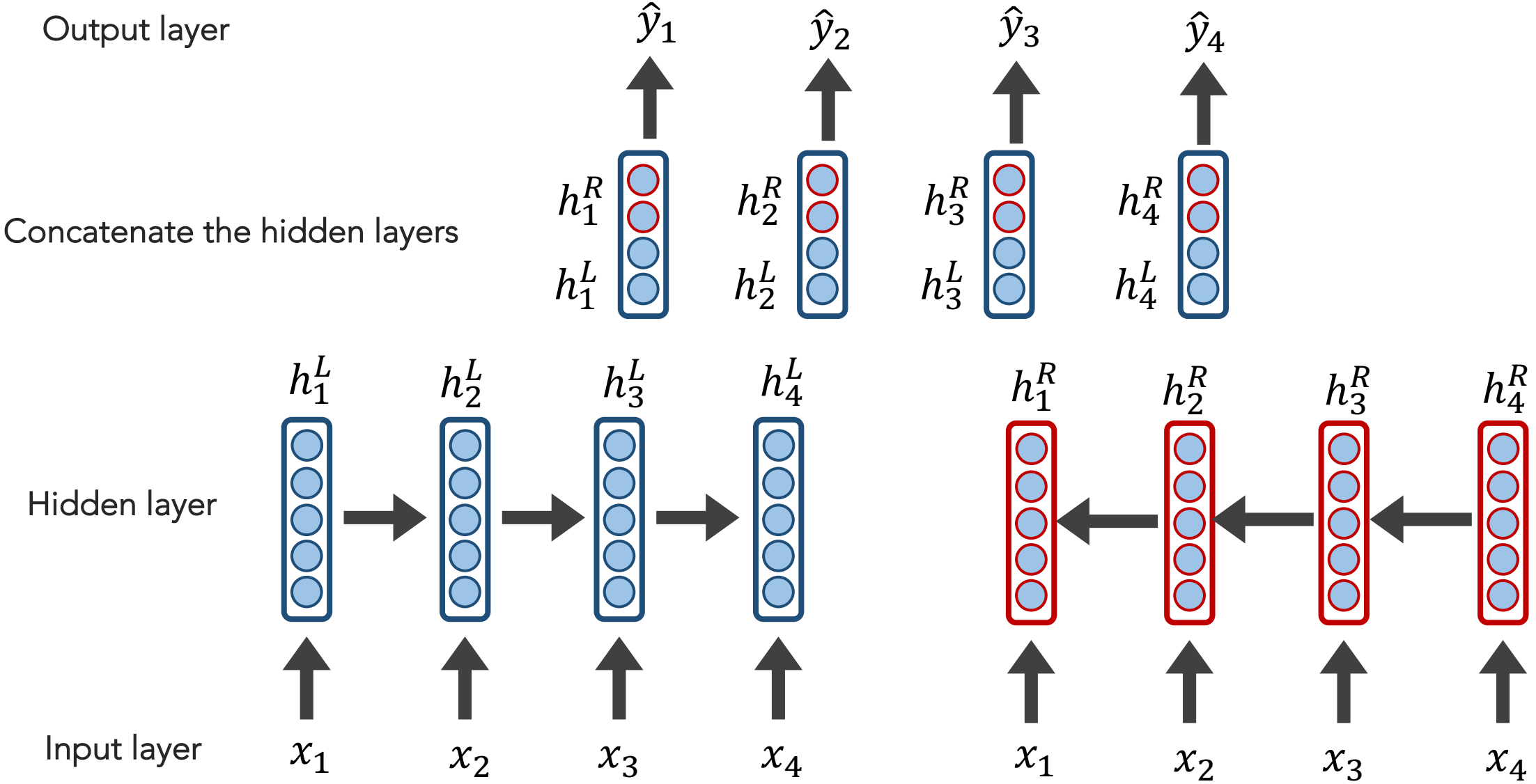
For brevity, let's use the follow schematic to represent an RNN



RNN Extensions: Bi-directional LSTMs



RNN Extensions: Bi-directional LSTMs



RNN Extensions: Bi-directional LSTMs

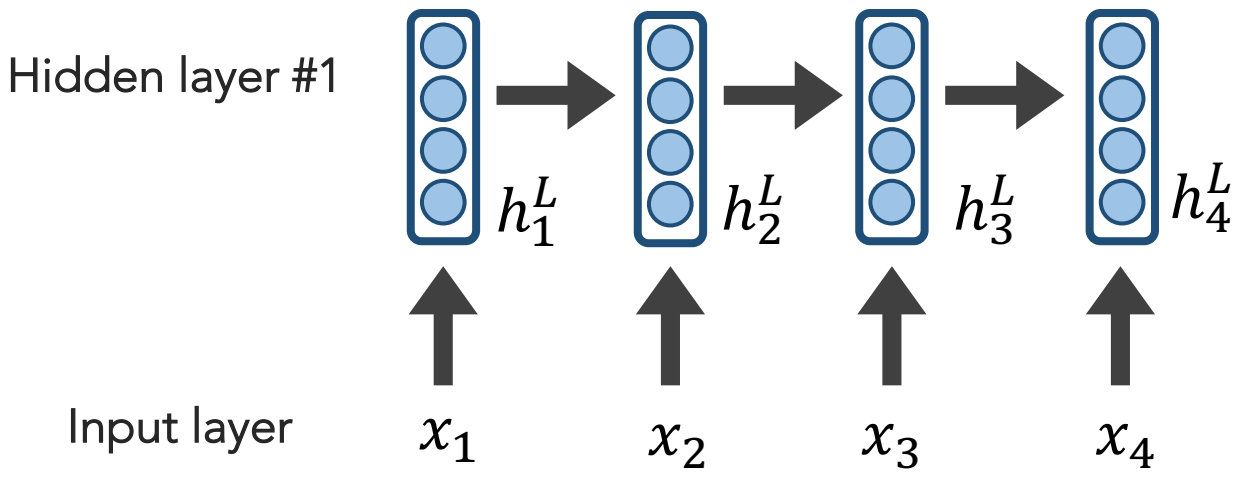
BI-LSTM STRENGTHS?

- Usually performs at least as well as uni-directional RNNs/LSTMs

BI-LSTM ISSUES?

- Slower to train
- Only possible if access to full data is allowed

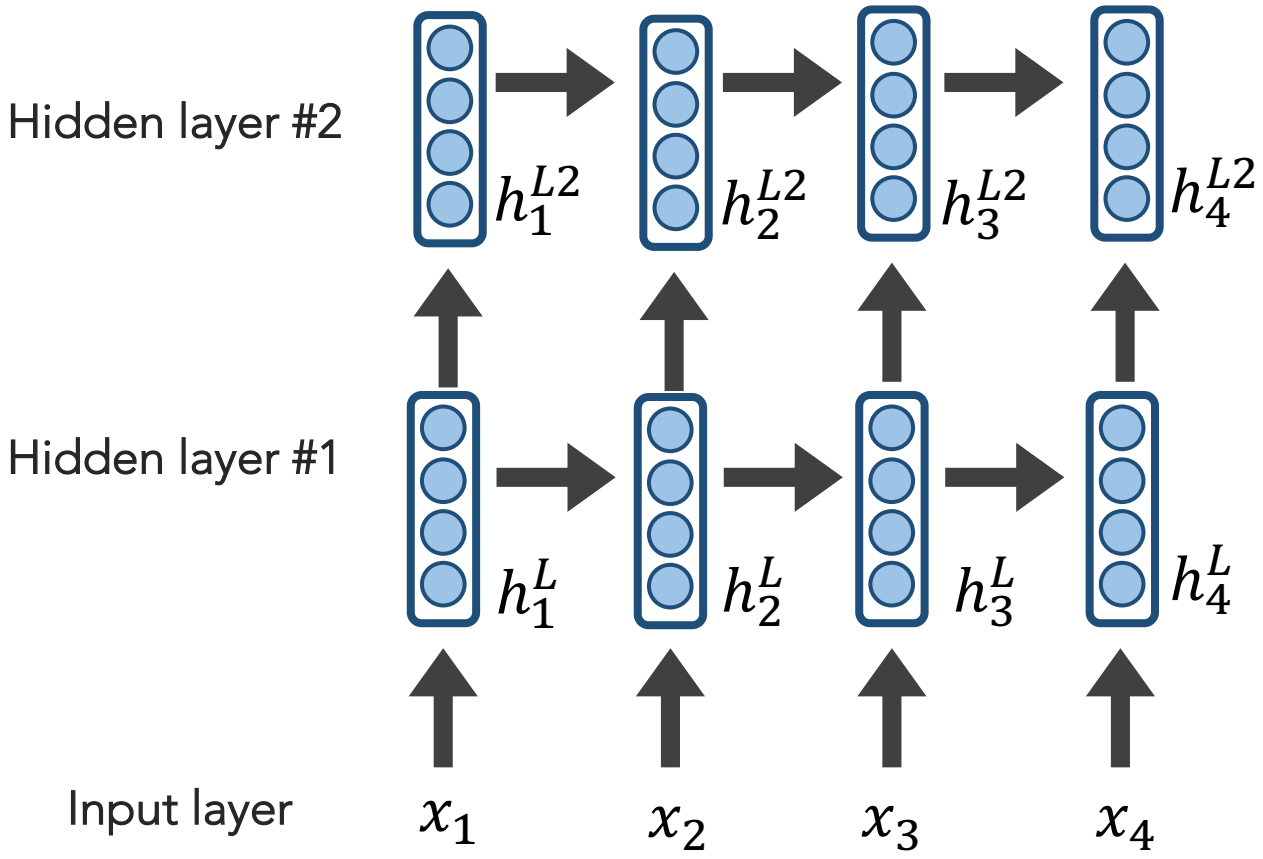
RNN Extensions: Stacked LSTMs



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

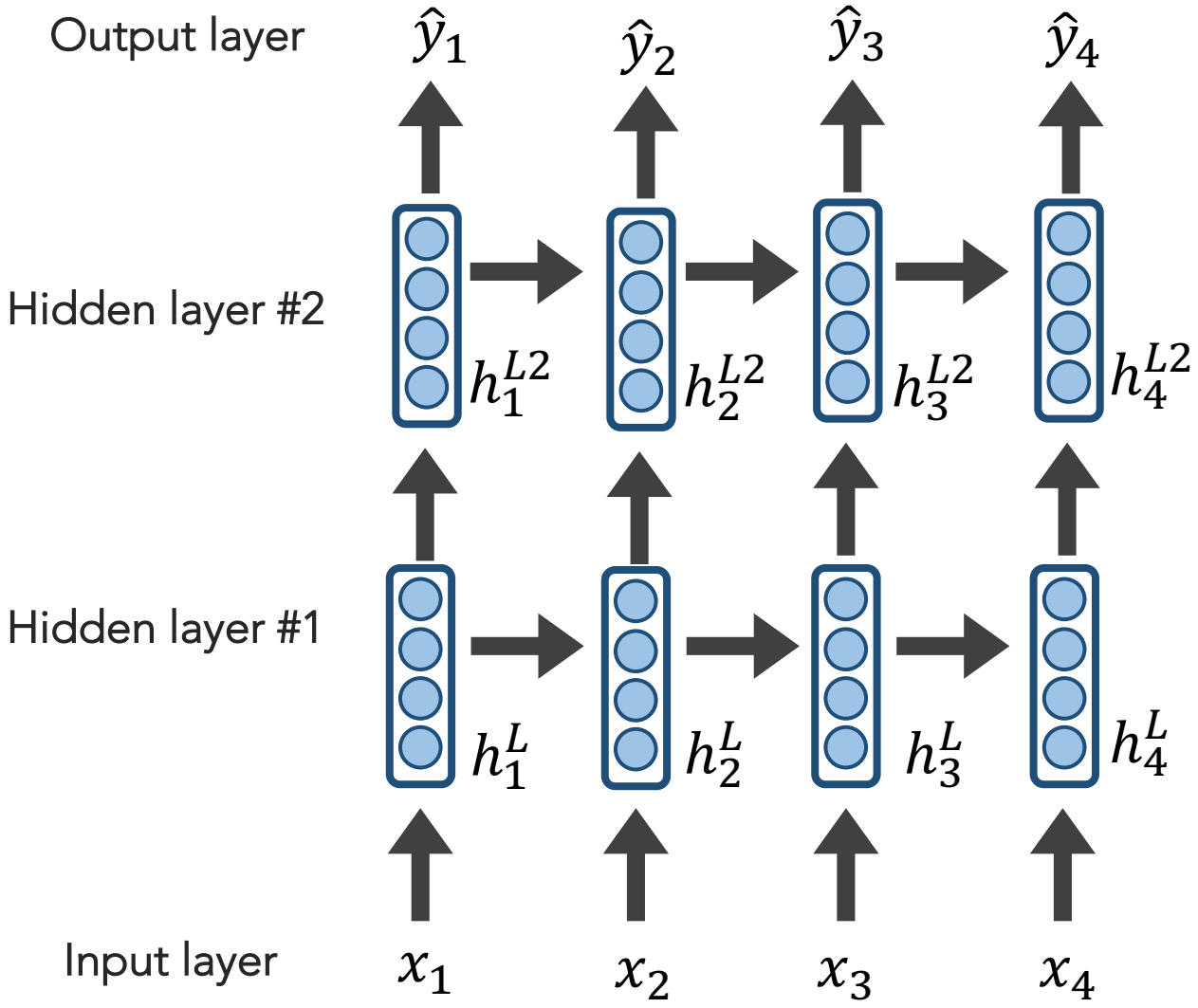
RNN Extensions: Stacked LSTMs



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

RNN Extensions: Stacked LSTMs



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

ELMo: Stacked Bi-directional LSTMs

General Idea:

- Goal is to get highly rich embeddings for each word (unique type)
- Use both directions of context (bi-directional), with increasing abstractions (stacked)
- Linearly combine all abstract representations (hidden layers) and optimize w.r.t. a particular task (e.g., sentiment classification)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

ELMo: Stacked Bi-directional LSTMs

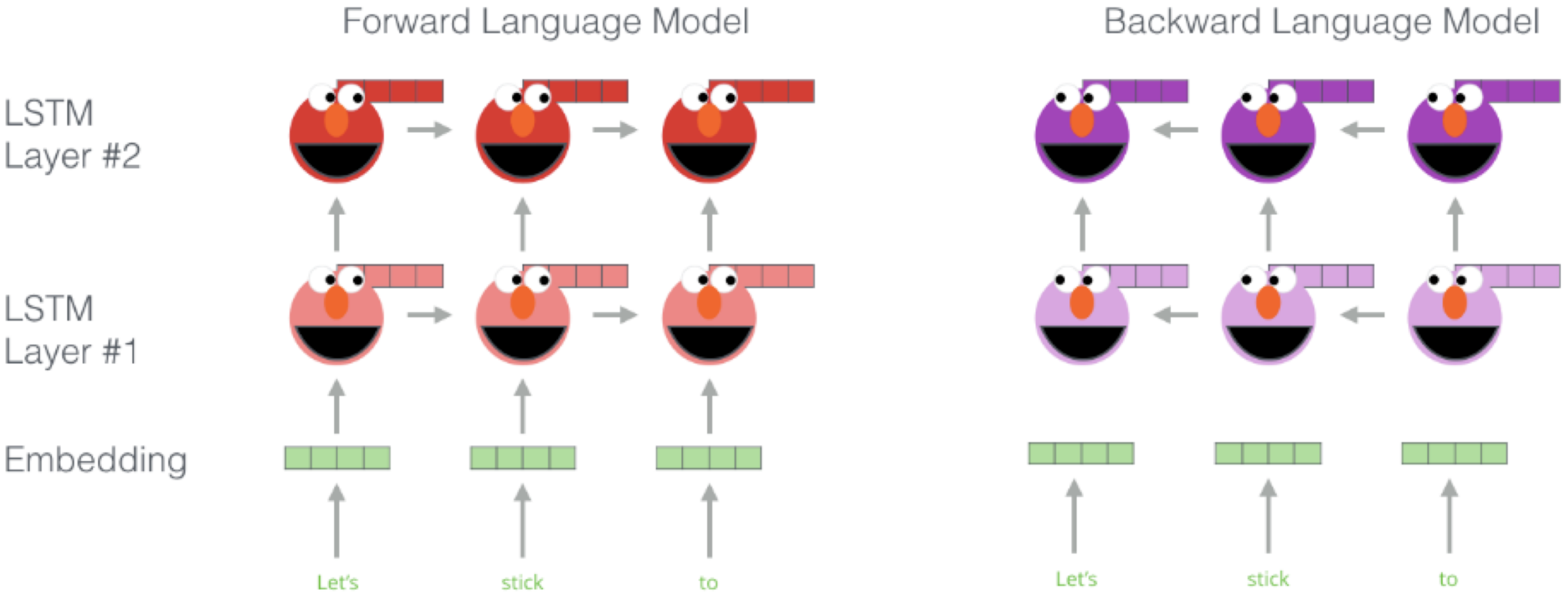


Illustration: http://jalammar.github.io/illustrated-bert/

Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context

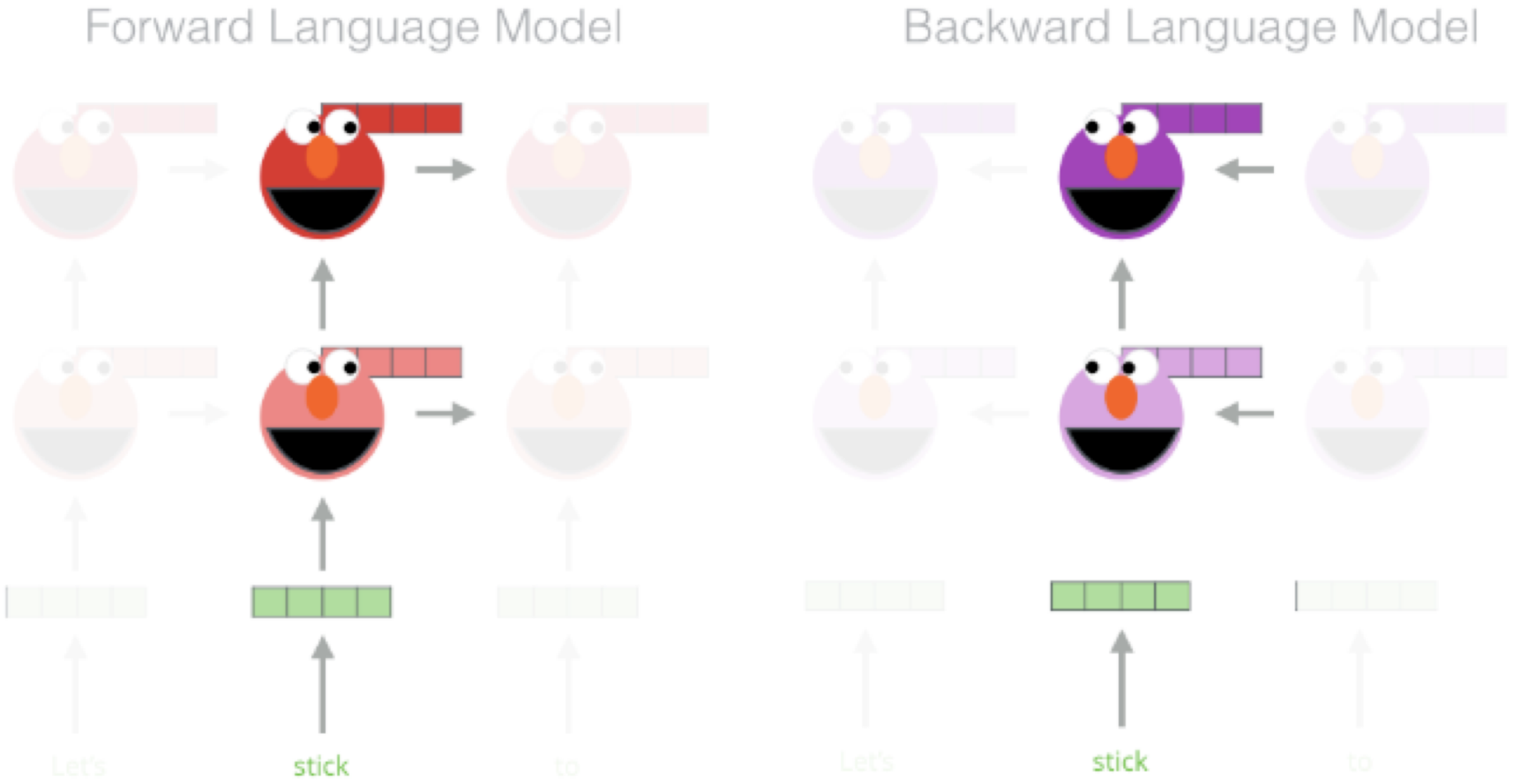


Illustration: http://jalammar.github.io/illustrated-bert/

ELMo: Stacked Bi-directional LSTMs

- ELMo yielded incredibly good word embeddings, which yielded state-of-the-art results when applied to many NLP tasks.
- Main ELMo takeaway: given enough training data, having tons of explicit connections between your vectors is useful
  (system can determine how to best use context)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

REFLECTION

So far, for all of our sequential modelling, we have been concerned with emitting 1 output per input datum.

Sometimes, a *sequence* is the smallest granularity we care about though (e.g., an English sentence)

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

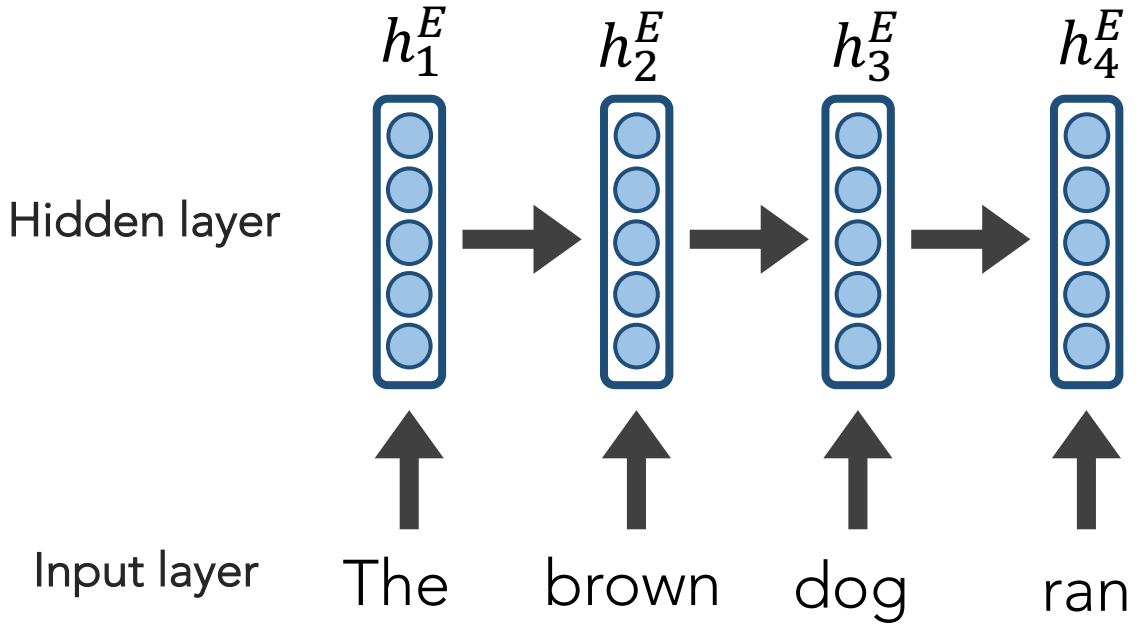
Transformers +BERT

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

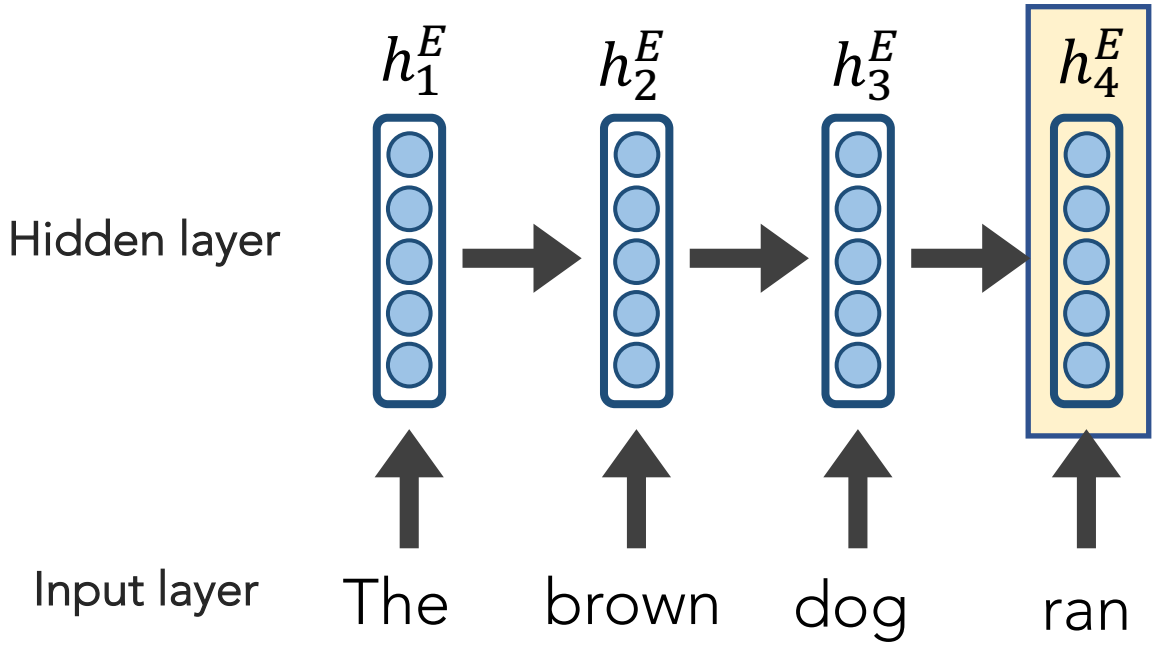
Sequence-to-Sequence (seq2seq)



ENCODER RNN

Sequence-to-Sequence (seq2seq)

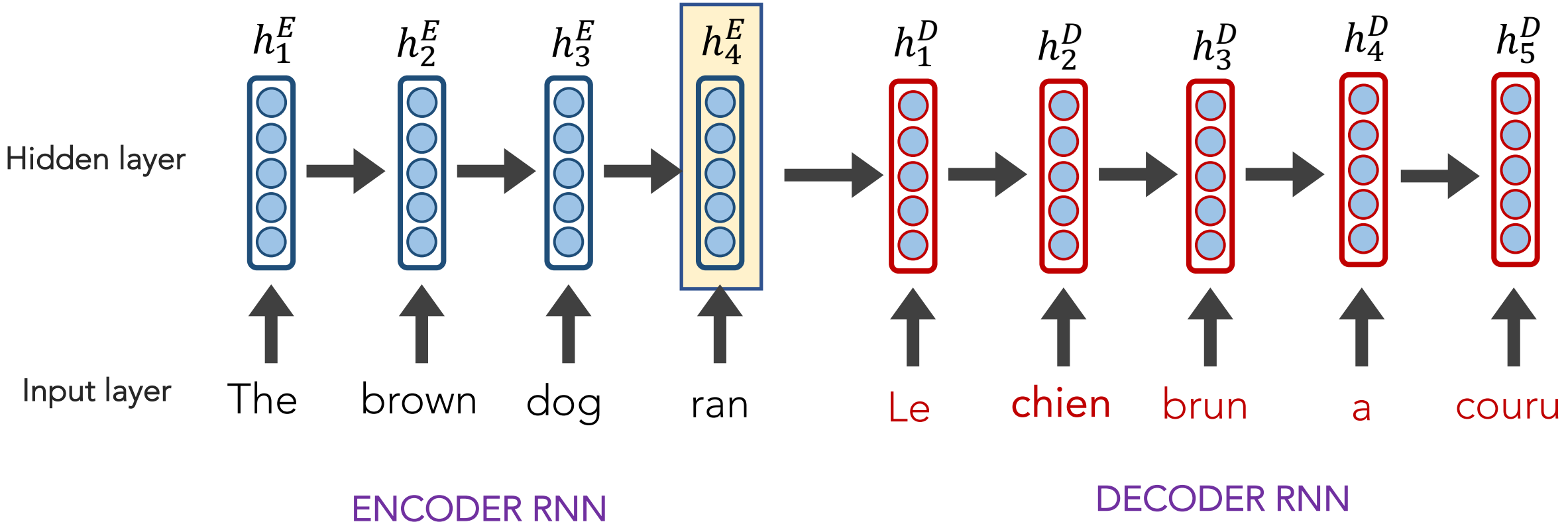
The final hidden state of the encoder RNN
is the initial state of the decoder RNN



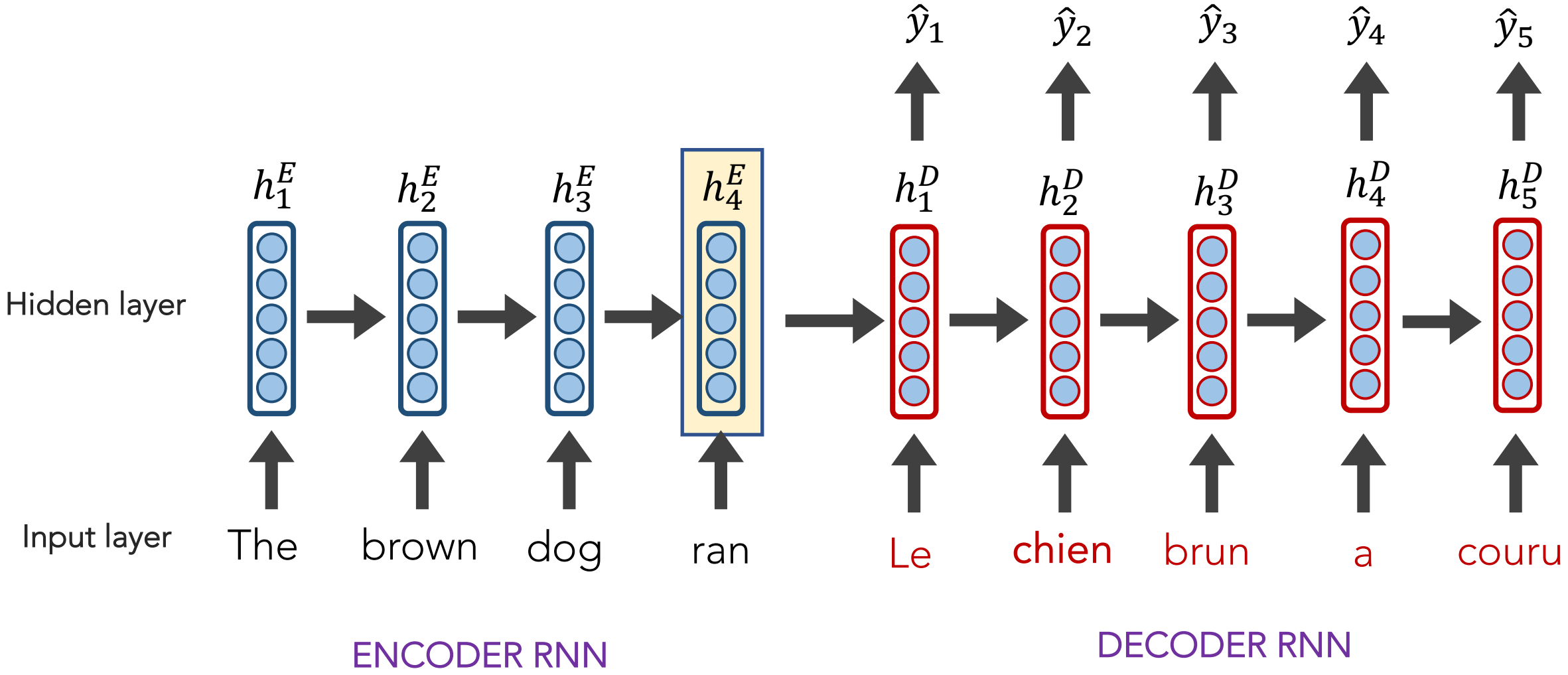
ENCODER RNN

Sequence-to-Sequence (seq2seq)

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

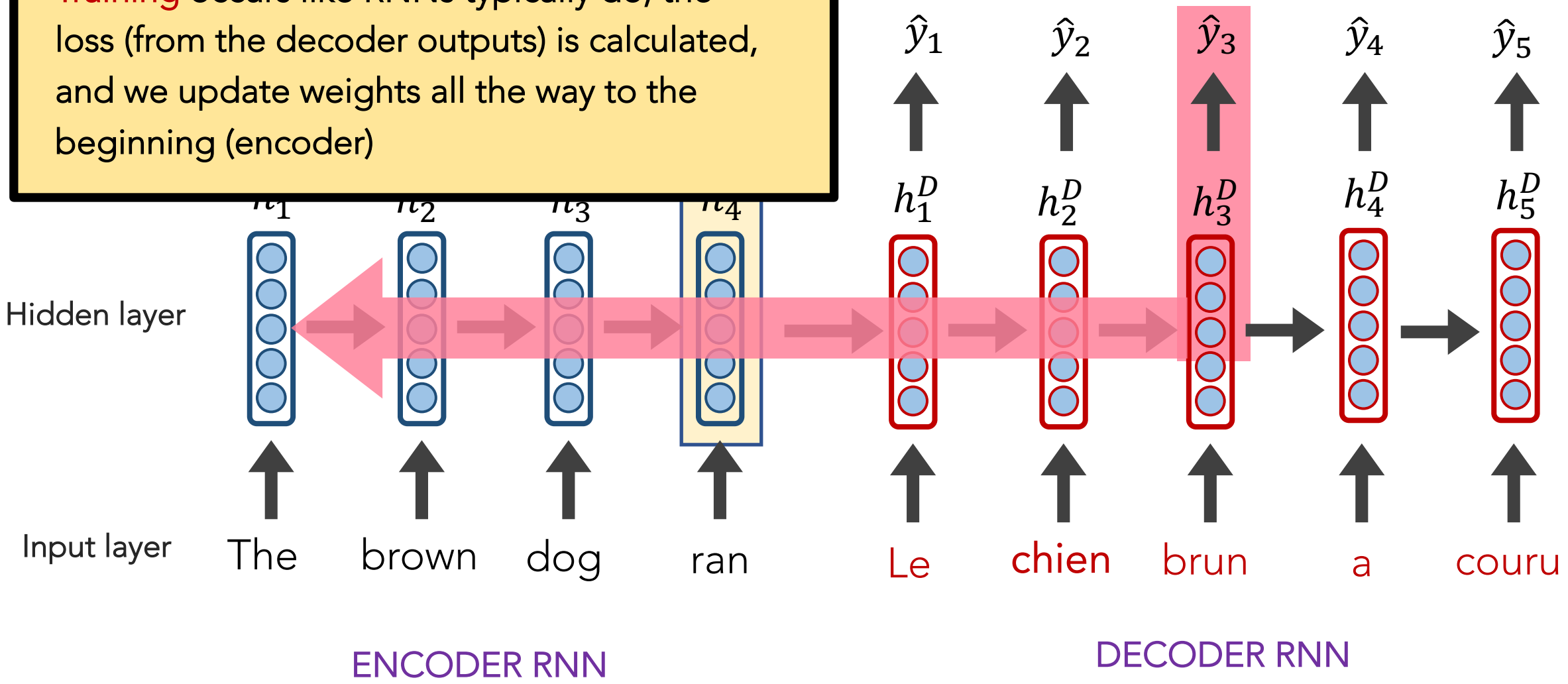


Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

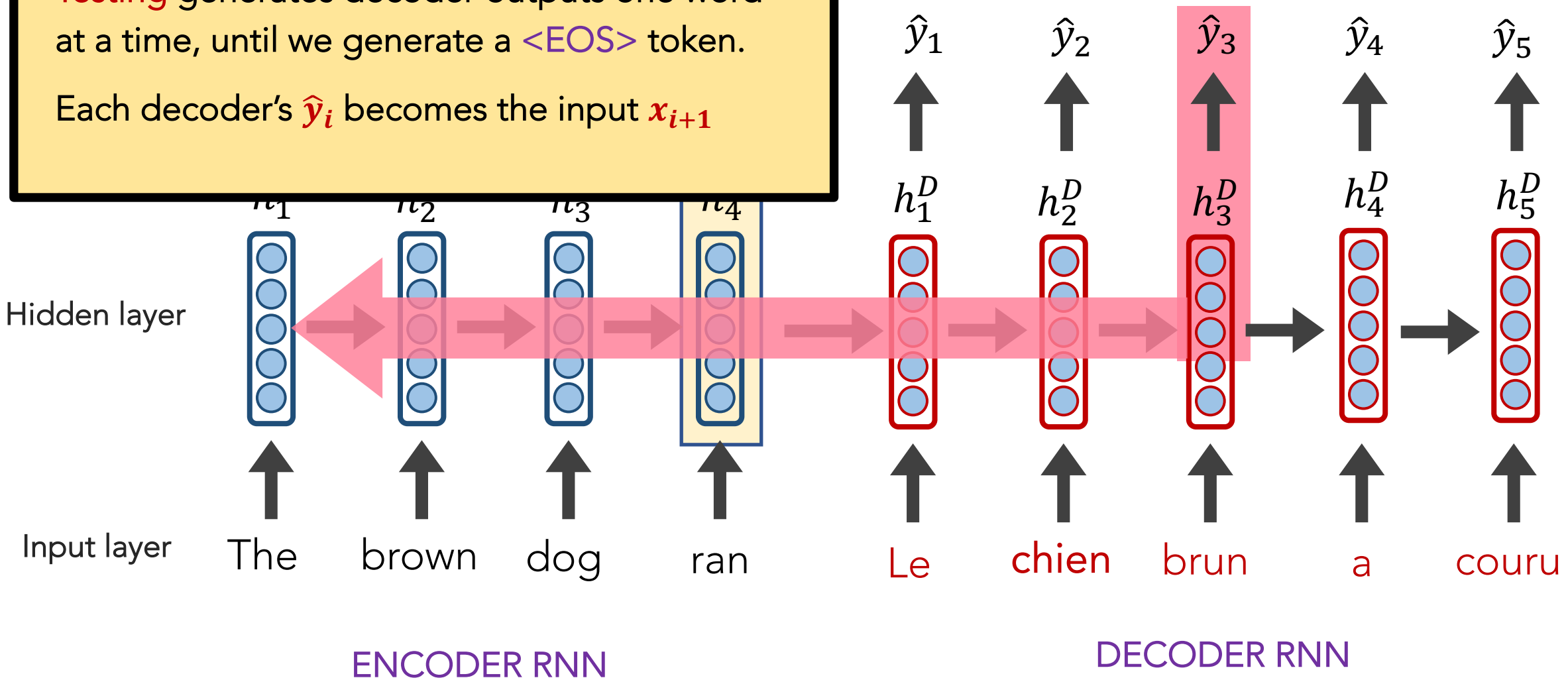
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

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

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

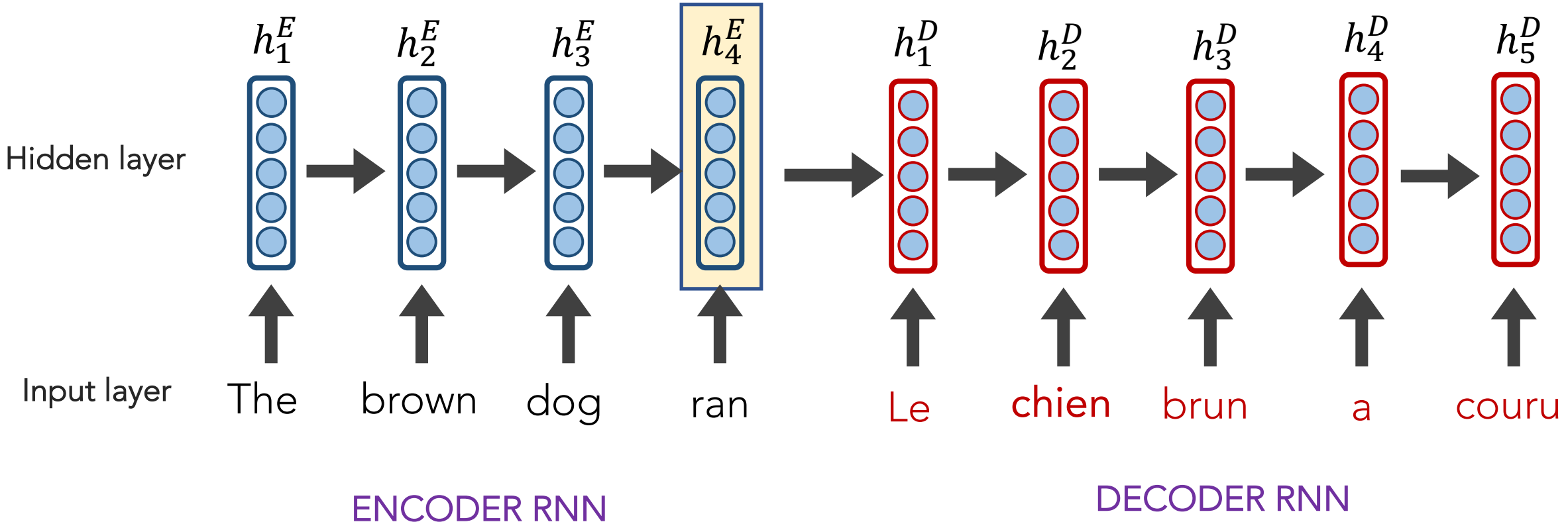


Sequence-to-Sequence (seq2seq)

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

Sequence-to-Sequence (seq2seq)

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

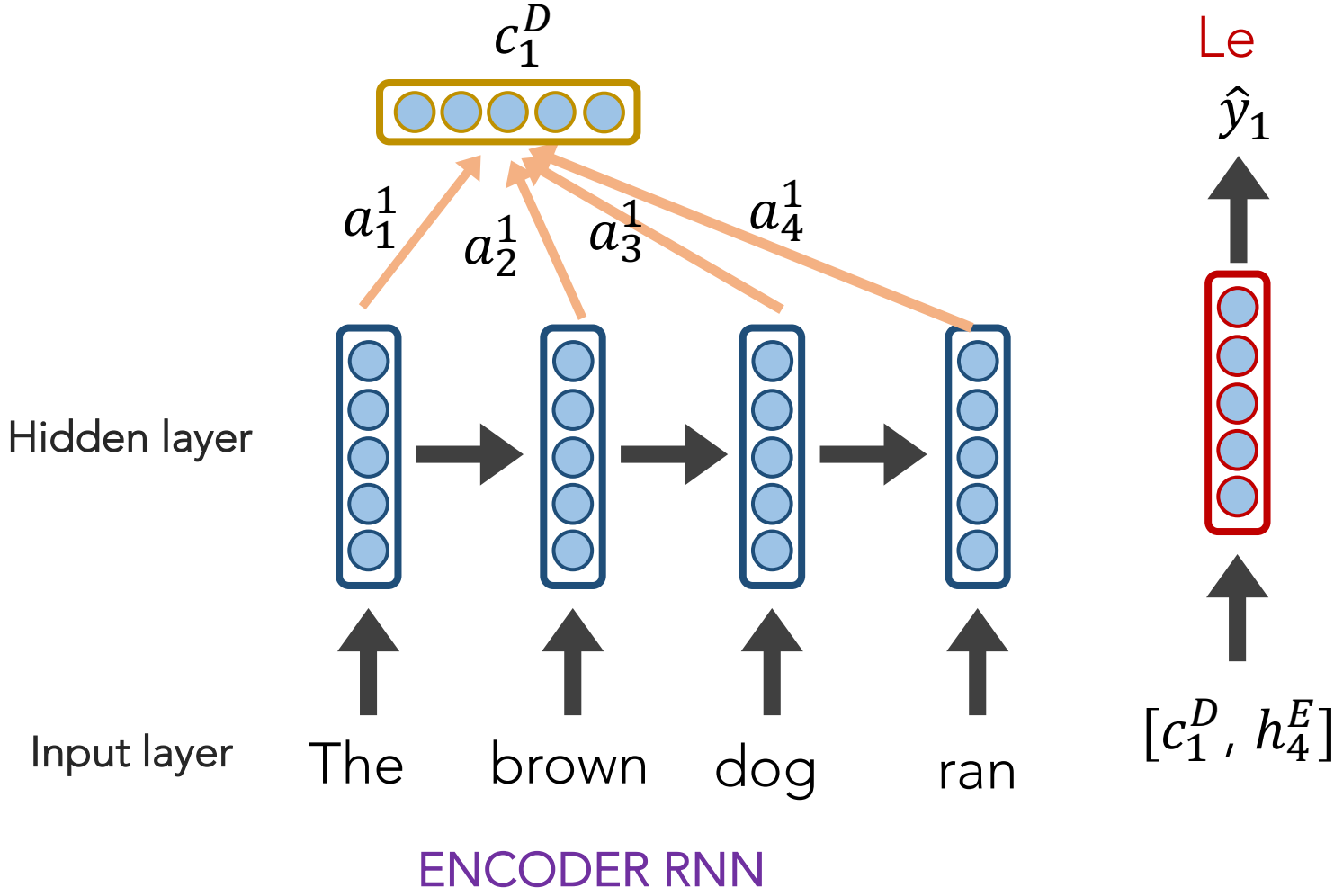


Sequence-to-Sequence (seq2seq)

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

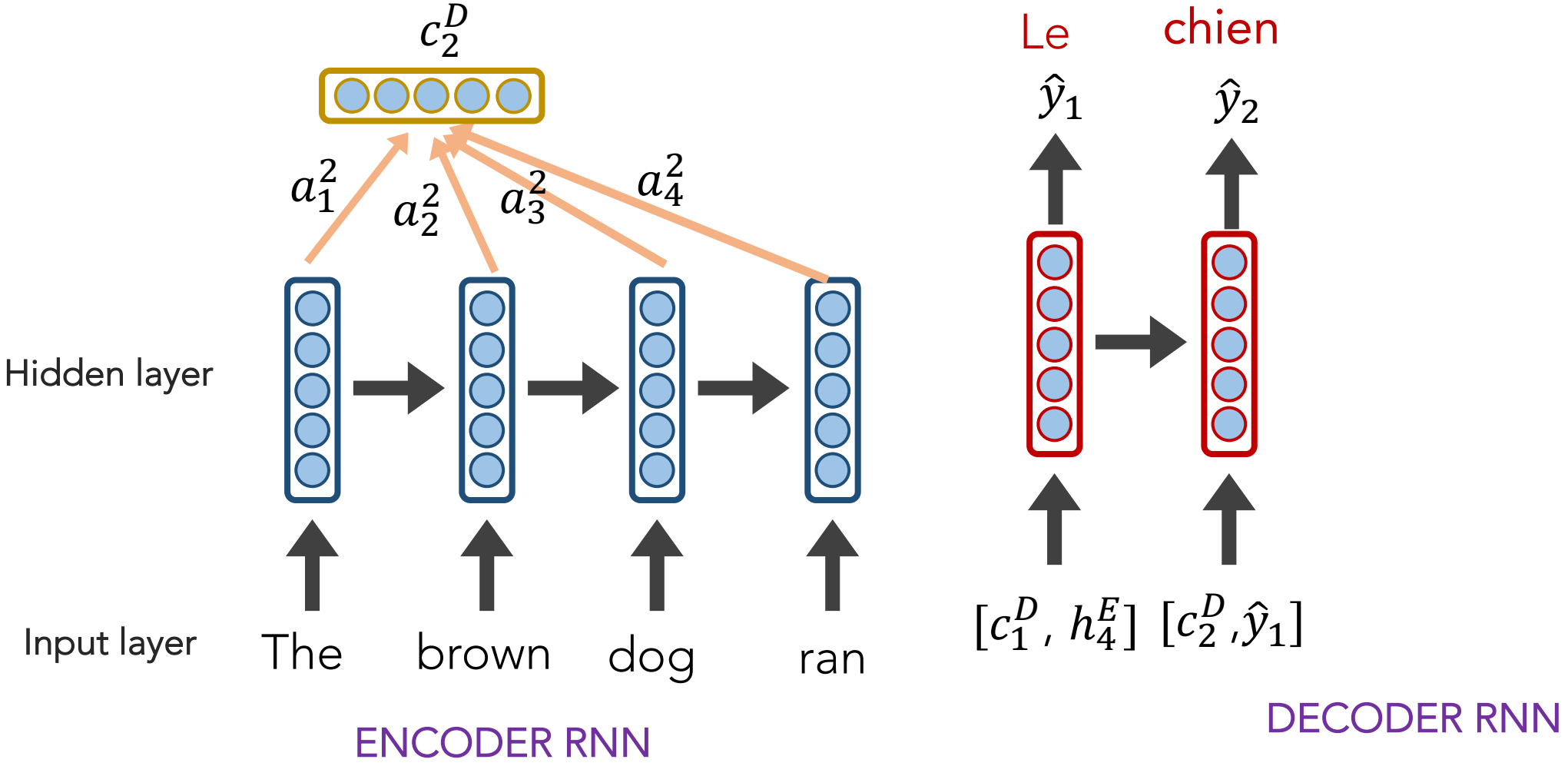
seq2seq + Attention

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



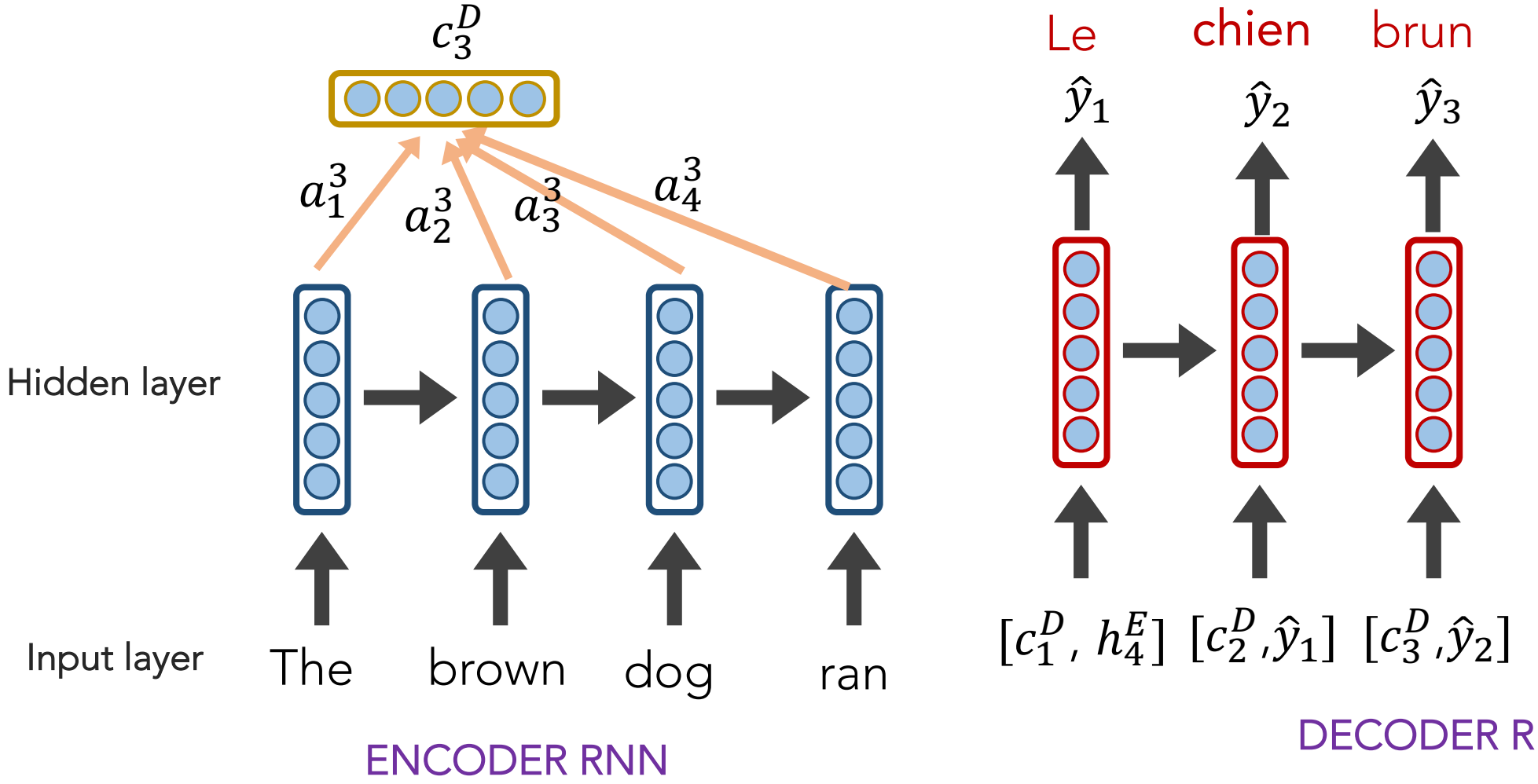
seq2seq + Attention

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



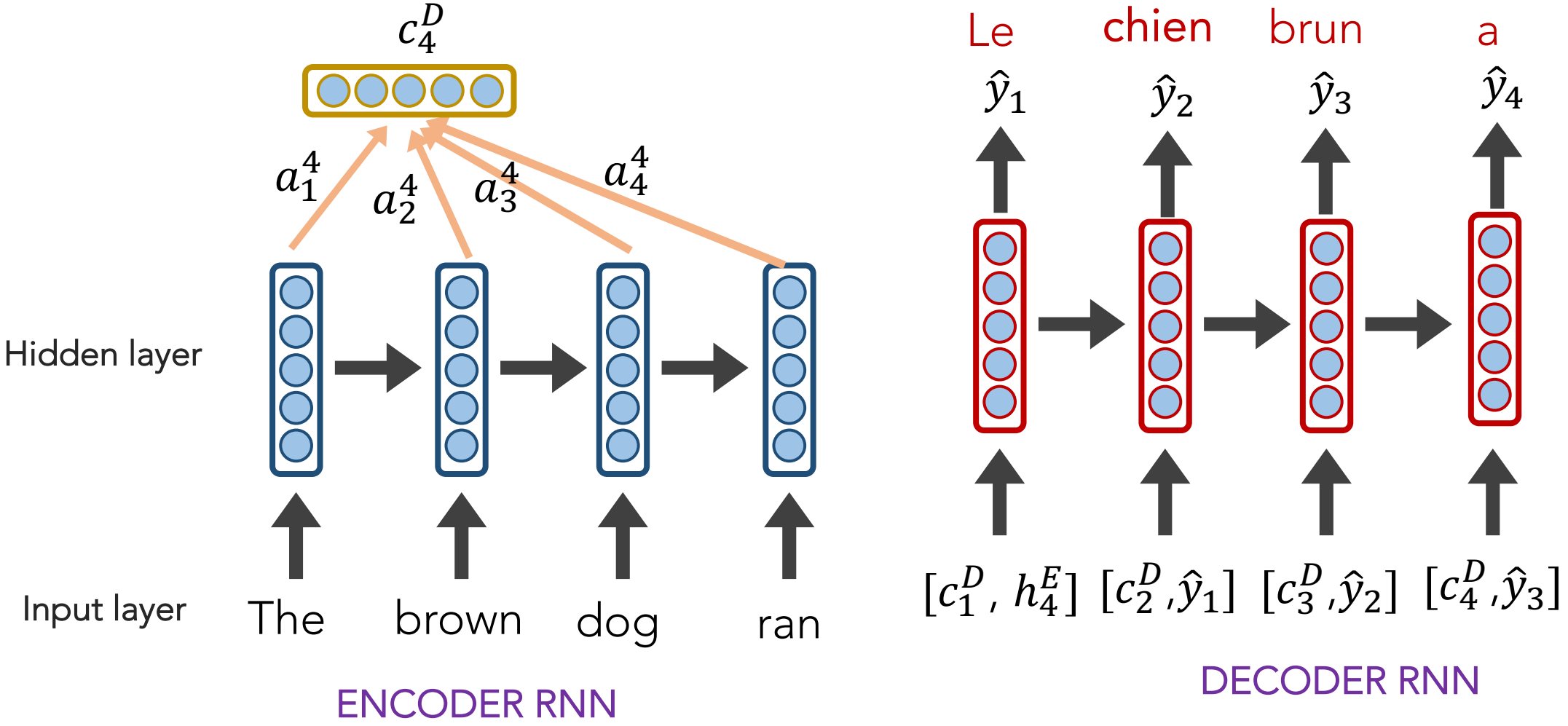
seq2seq + Attention

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



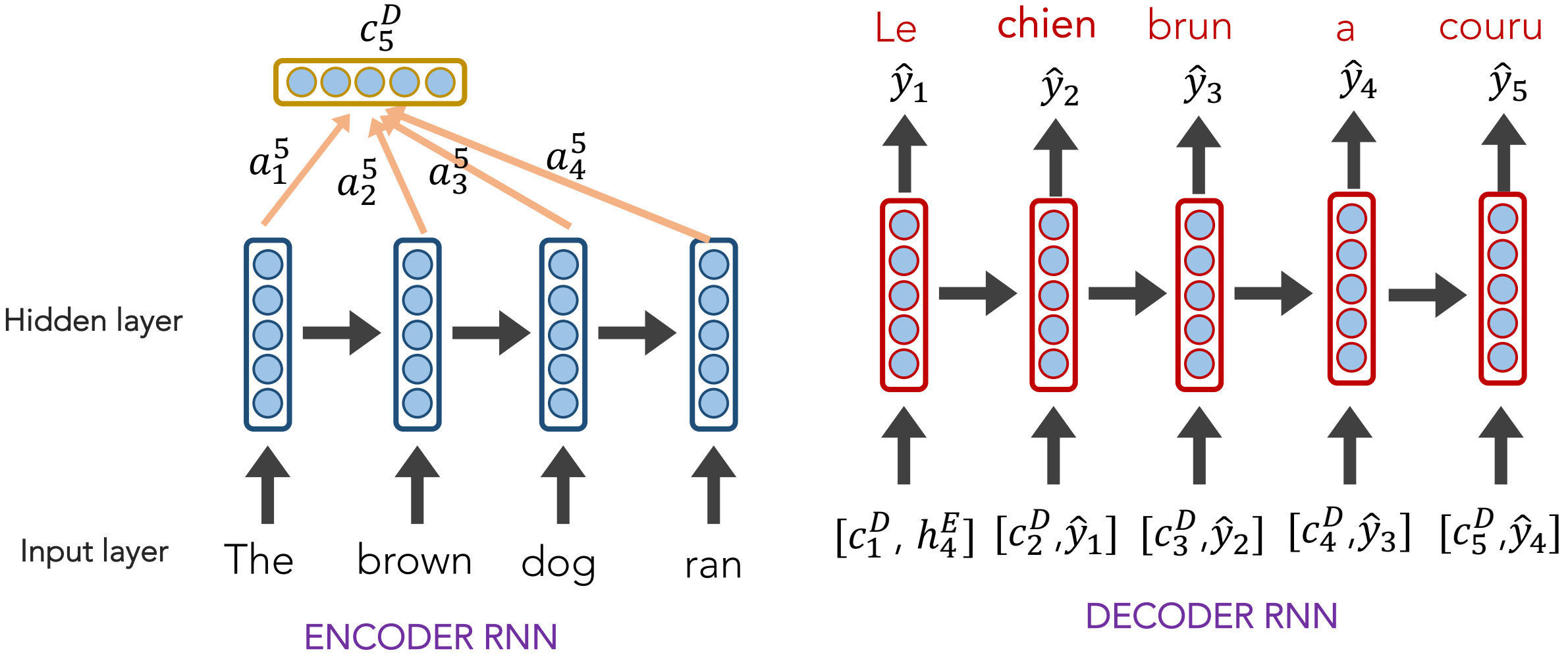
seq2seq + Attention

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



seq2seq + Attention

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



seq2seq + Attention

Attention:

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

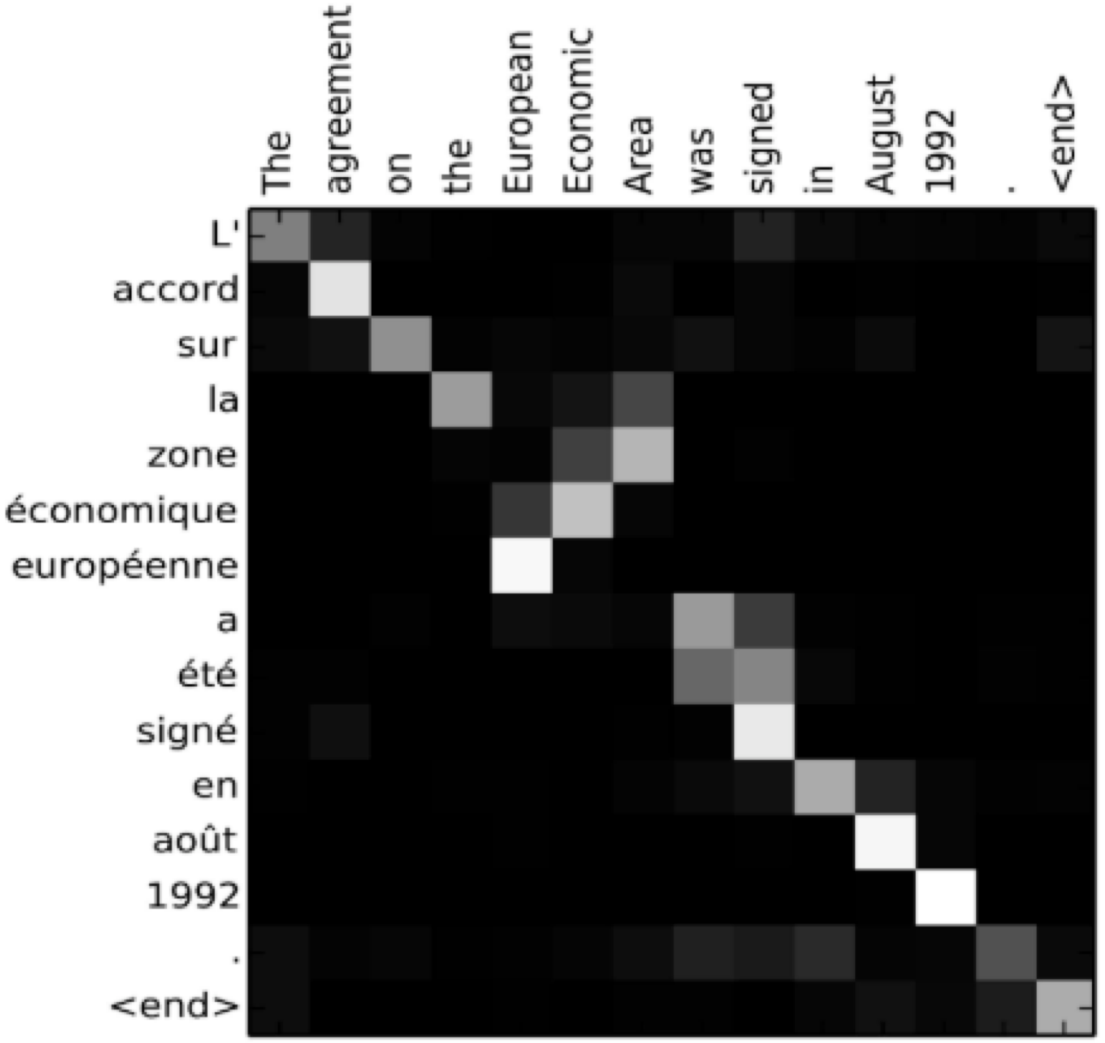


Image source: Fig 3 in [Bahdanau et al., 2015]

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

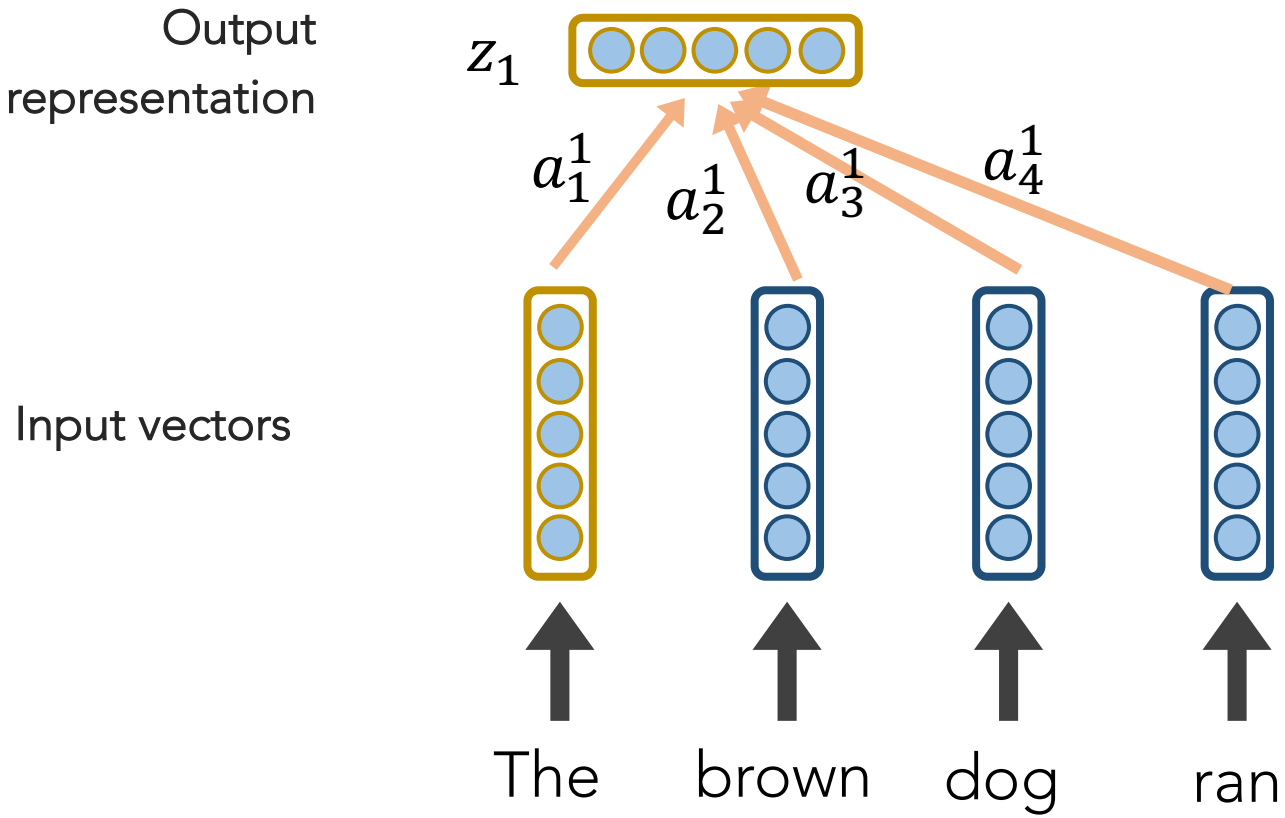
Transformers +BERT

Conclusions

Self-Attention

- Models direct relationships between all words in a given sequence (e.g., sentence)
- Does not concern a seq2seq (i.e., encoder-decoder RNN) framework
- Each word in a sequence can be transformed into an abstract **representation** (embedding) based on the weighted sums of the other words in the same sequence

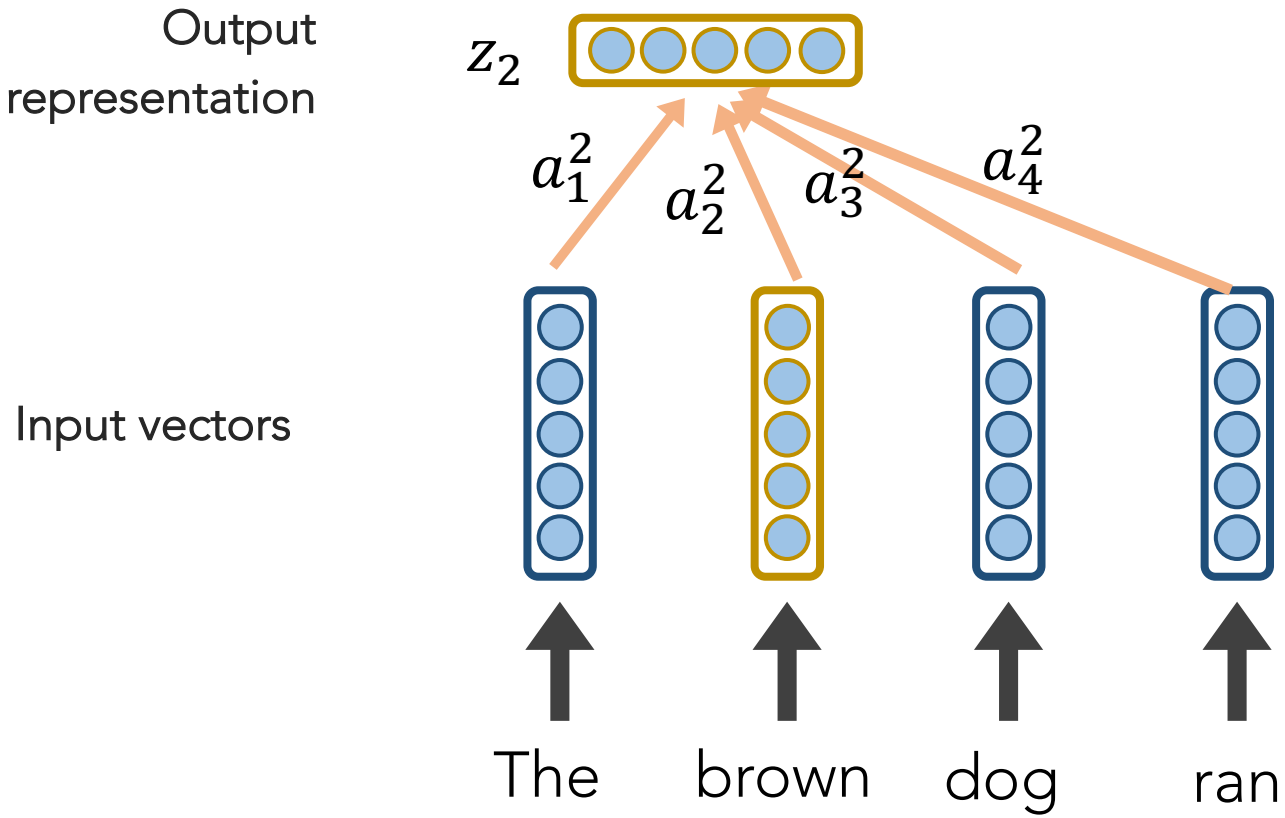
Self-Attention



This is a large simplification.

The representations are created from using Query, Key, and Value vectors, produced from learned weight matrices during Training

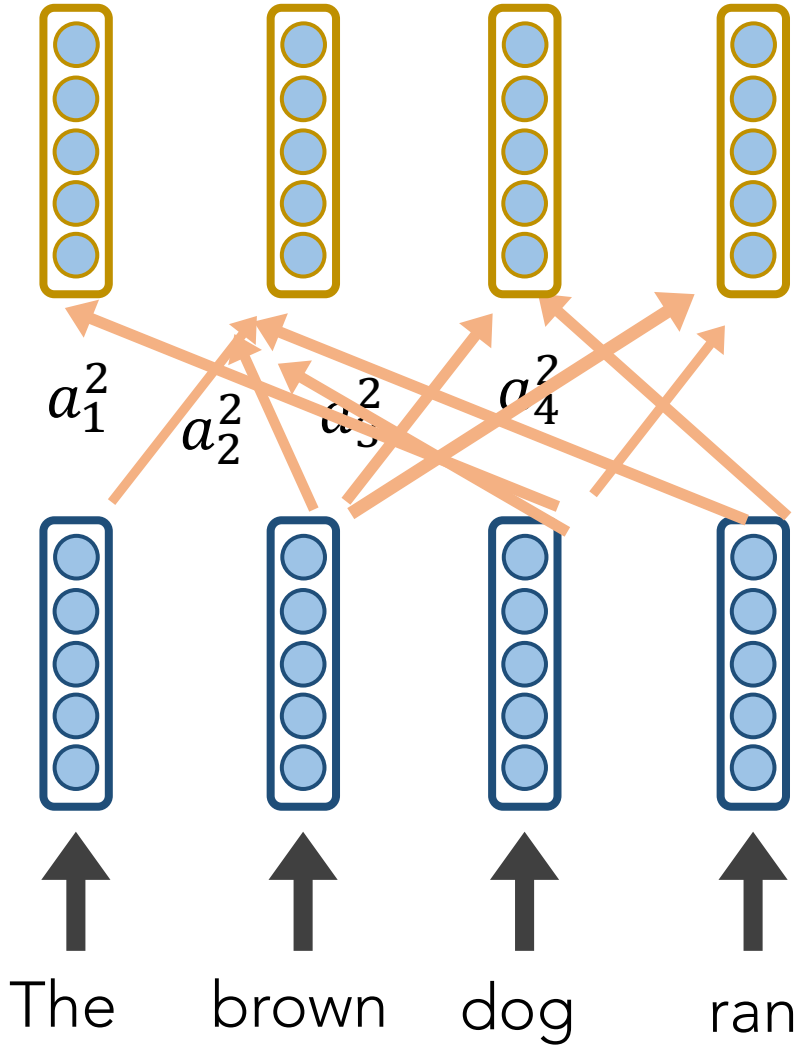
Self-Attention



This is a large simplification.

The representations are created from using Query, Key, and Value vectors, produced from learned weight matrices during Training

Self-Attention



This is a large simplification.

The representations are created from using Query, Key, and Value vectors, produced from learned weight matrices during Training

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; it has an encoder-decoder component, yet also uses self-attention to refine each respective sequence's representation

Transformer

- Transformers yield the best results for machine translation (seq2seq)
- Transformers handle long-range dependencies better than LSTMs
- BERT is an example of a Transformer

BERT

- BERT is like the encoder portion of a Transformer
- Uses self-attention
- Uses bi-directional conditioning to perform language modelling
- Yet, it doesn't see its own words because it cleverly masks 15% of its words
- Fine-tunes on a sentence/entailment task
- BERT provides generalized contextual embeddings which can be fine-tuned toward other classification tasks (e.g., sentiment classification)

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

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**: great contextual usage; great sequence modelling
- **Seq2Seq**: maps 1 sequence to another
- **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 and self-attention

Sequential Modelling

LSTM

RNN

FFNN

n-grams

Sequence Modelling
(1-to-1 mapping)

Transformer

LSTM w/ attention

seq2seq

Conclusion

Questions?

Credit & further resources:

- Backprop: http://cs231n.github.io/optimization-2/
- Abigail See's lectures: http://web.stanford.edu/class/cs224n/index.html
- Illustrated BERT: http://jalammar.github.io/illustrated-bert/
- Andrew Ng (Attention): https://www.youtube.com/watch?v=quoGRI-1l0A
- Illustrated Transformer: https://jalammar.github.io/illustrated-transformer/
- Google (Transformers): https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- ELMo paper: https://arxiv.org/pdf/1802.05365.pdf