Lecture 6: LSTMs

Contextualized, Token-based Representations

Harvard

AC295/CS287r/CSCI E-115B

Chris Tanner

Stayin' alive was no jive […] but it was just a dream for the teen, who was a fiend. Started [hustlin’ at] 16, and runnin' up in [LSTM gates].

-- Raekwon of Wu-Tang (1994)

ANNOUNCEMENTS

- HW2 coming very soon. Due in 2 weeks.
- Research Proposals are due in 9 days, Sept 30.
- Office Hours:
  - This week, my OH will be pushed back 30 min: 3:30pm – 5:30pm
  - Please reserve your coding questions for the TFs and/or EdStem, as I hold office hours solo, and debugging code can easily bottleneck the queue.

RECAP: L4

Distributed Representations: dense vectors (aka embeddings) that aim to convey the meaning of tokens:

- “word embeddings” refer to when you have type-based representations
- “contextualized embeddings” refer to when you have token-based representations

RECAP: L4

An auto-regressive LM is one that only has access to the previous tokens (and the outputs become the inputs).

Evaluation: Perplexity

A masked LM can peak ahead, too. It “masks” a word within the context (i.e., center word).

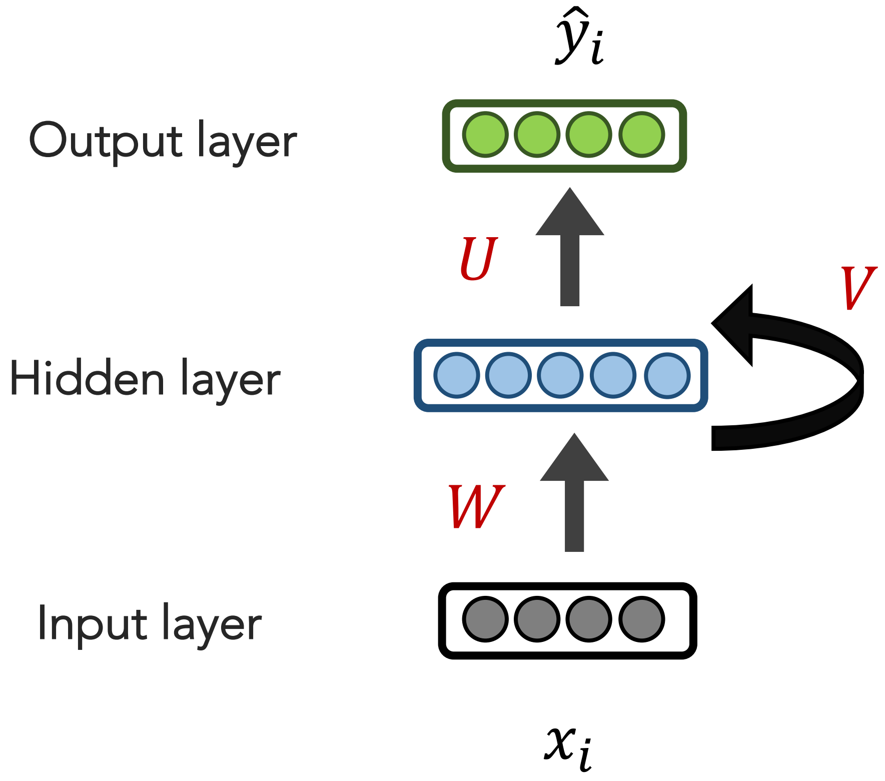
Evaluation: downstream NLP tasks that uses the learned embeddings.

Both of these can produce useful word embeddings.

RECAP: L5

- RNNs help capture more context while avoiding sparsity, storage, and compute issues!

- The hidden layer is what we care about. It represents the word's "meaning".



Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

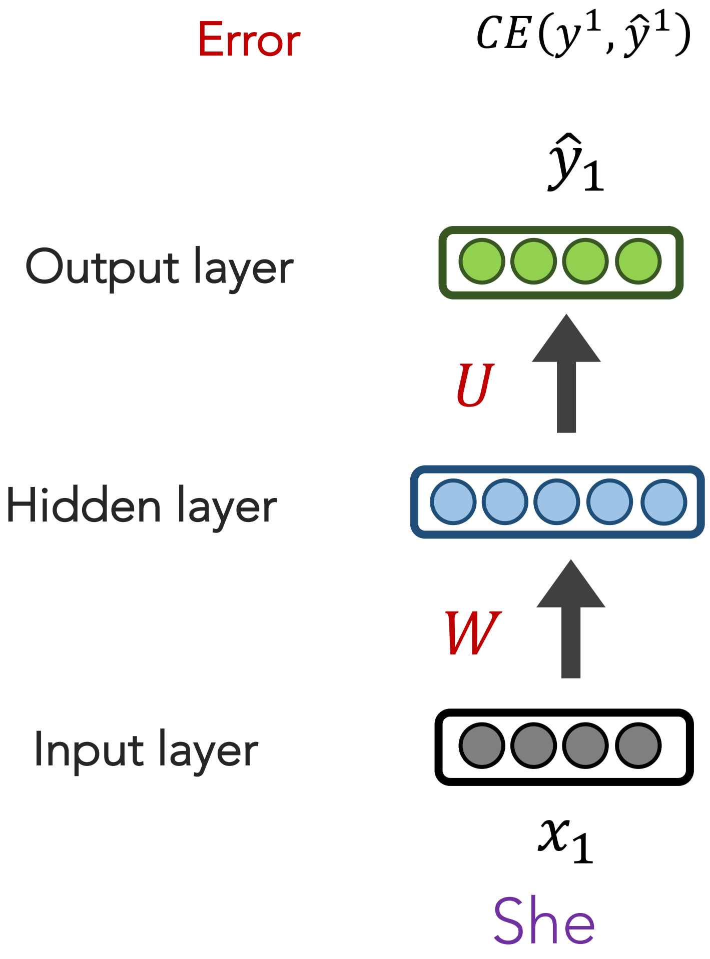
Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

RNN

Training Process

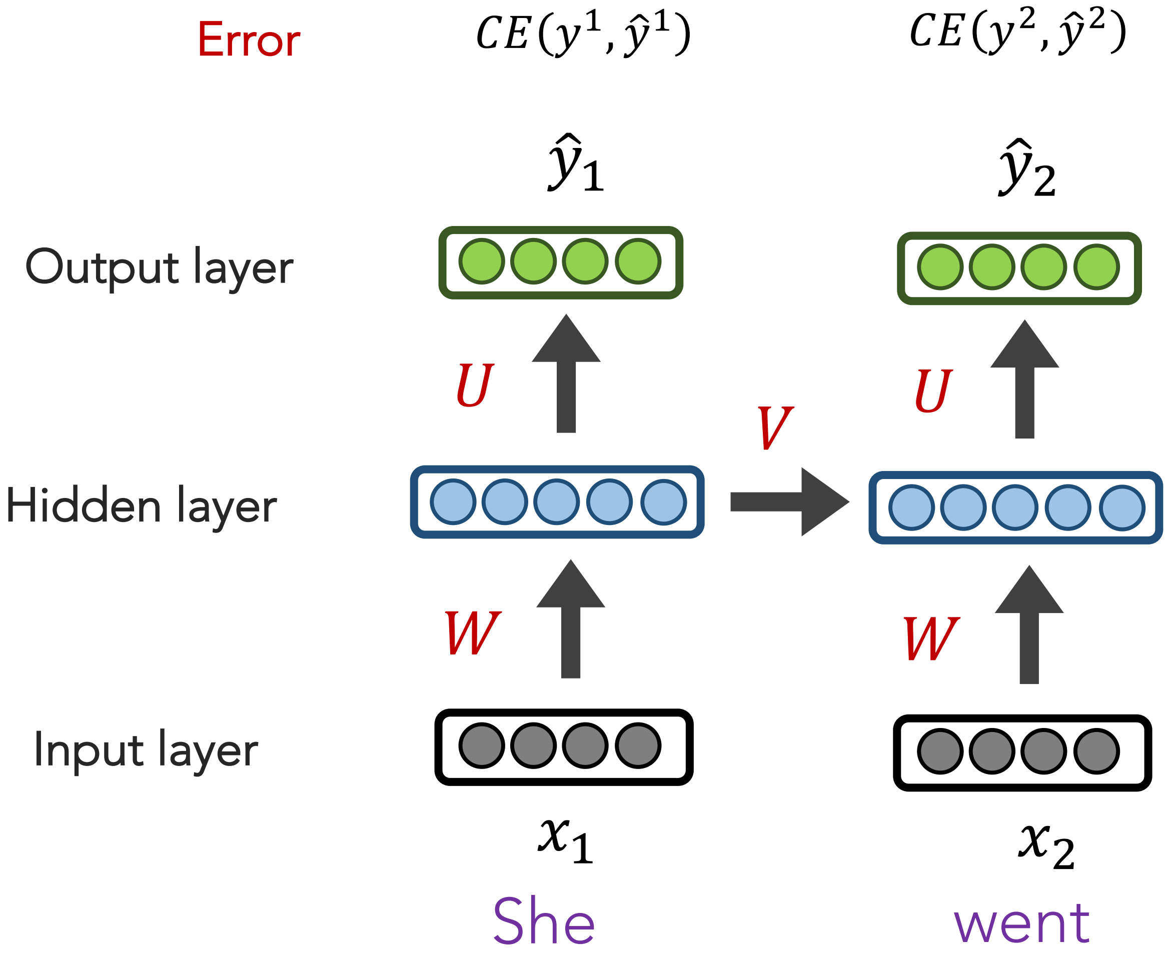
$$CE\left(y^{i}, \hat{y}^{i}\right) = -\sum_{w \in V} y_{w}^{i} \log(\hat{y}_{w}^{i})$$



RNN

Training Process

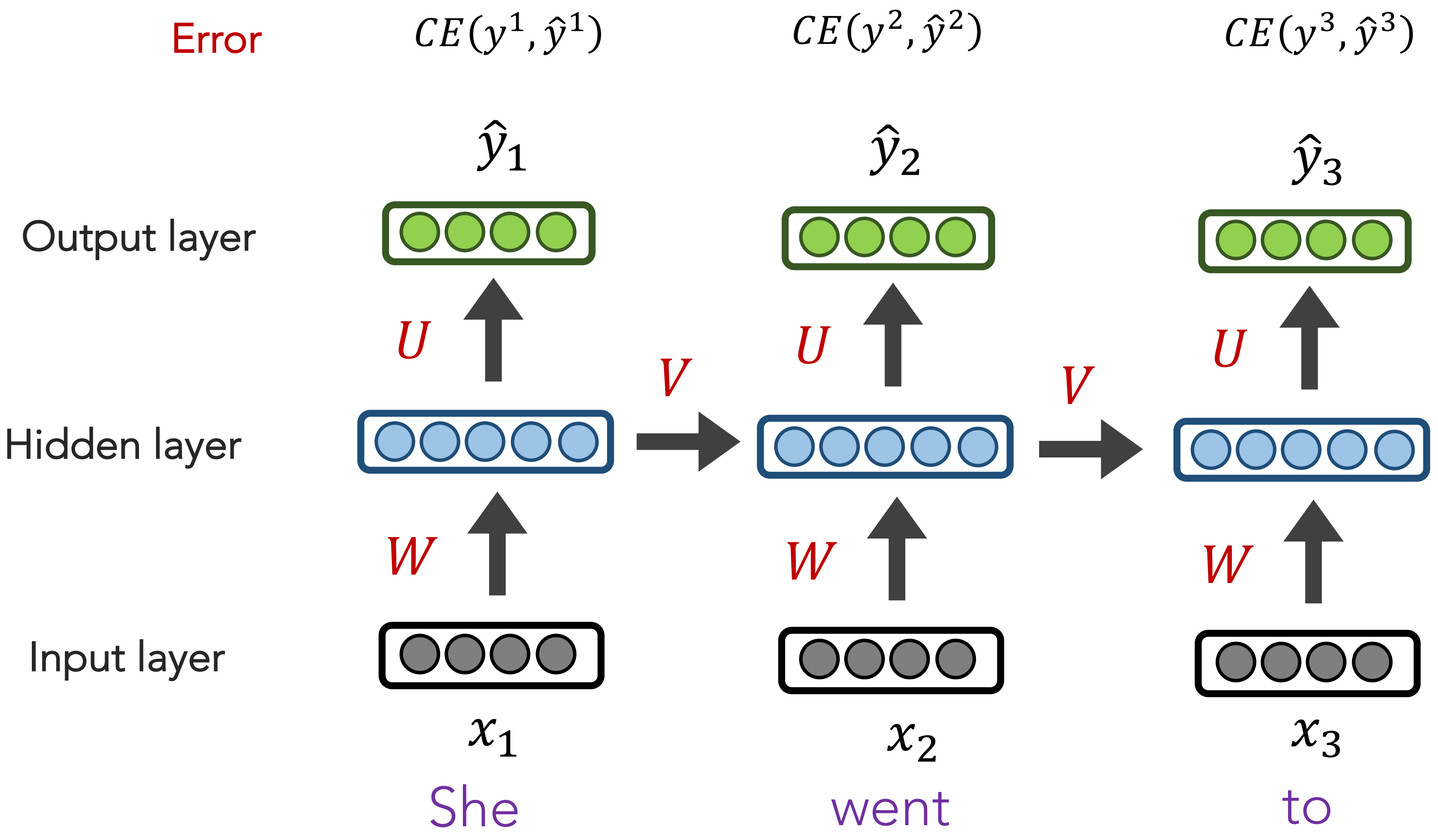
$$CE\left(y^i, \hat{y}^i\right) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

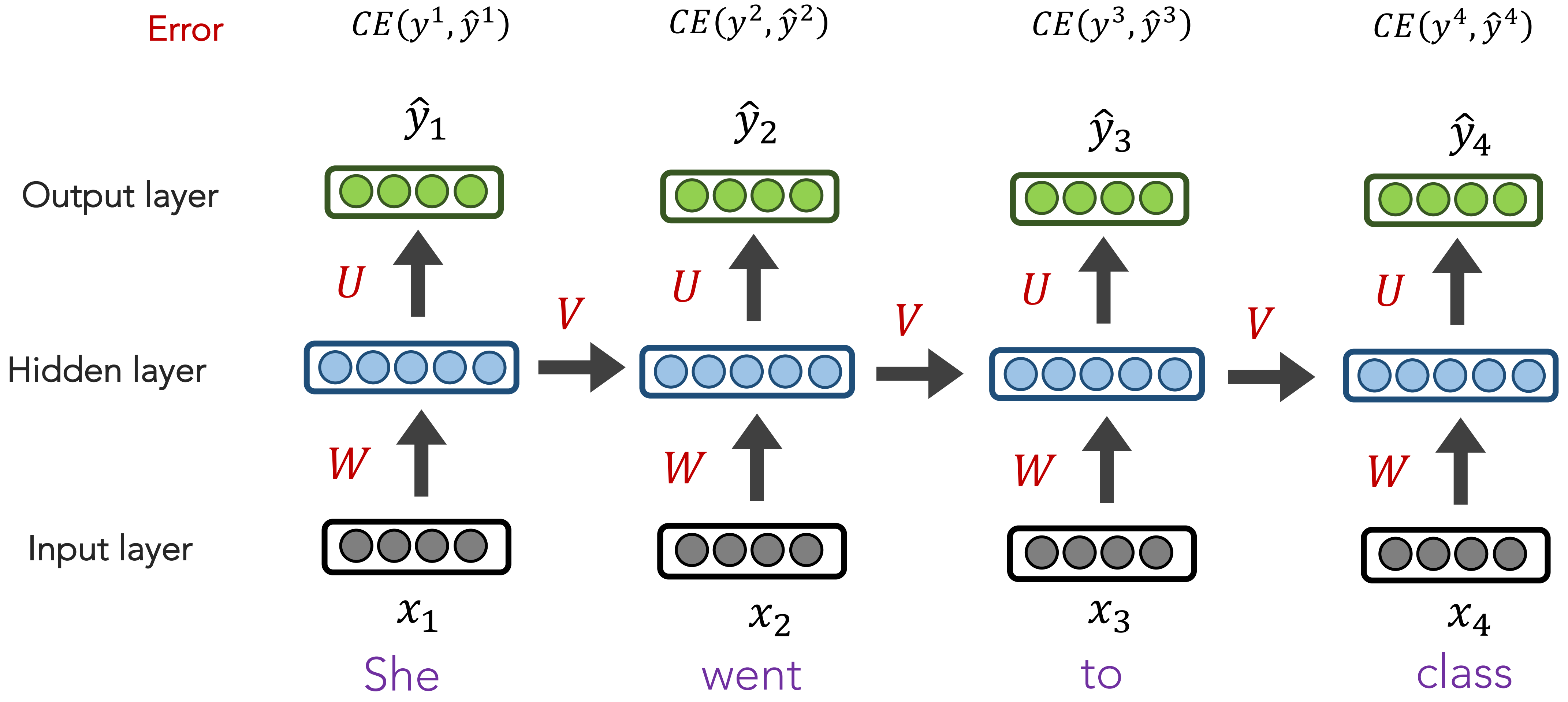
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

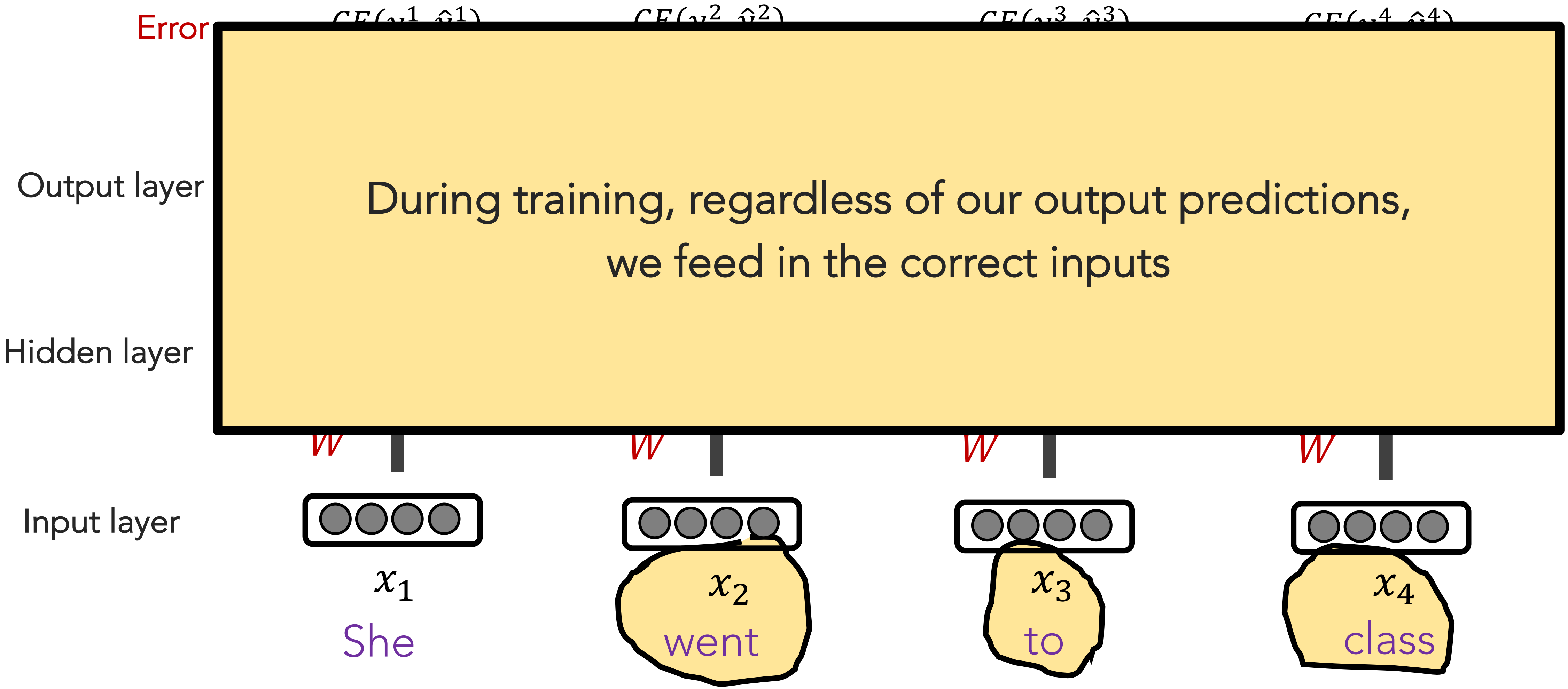
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

$$CE\left(y^i,\hat{y}^i\right) = -\sum_{w\in V} y^i_w \log(\hat{y}^i_w)$$

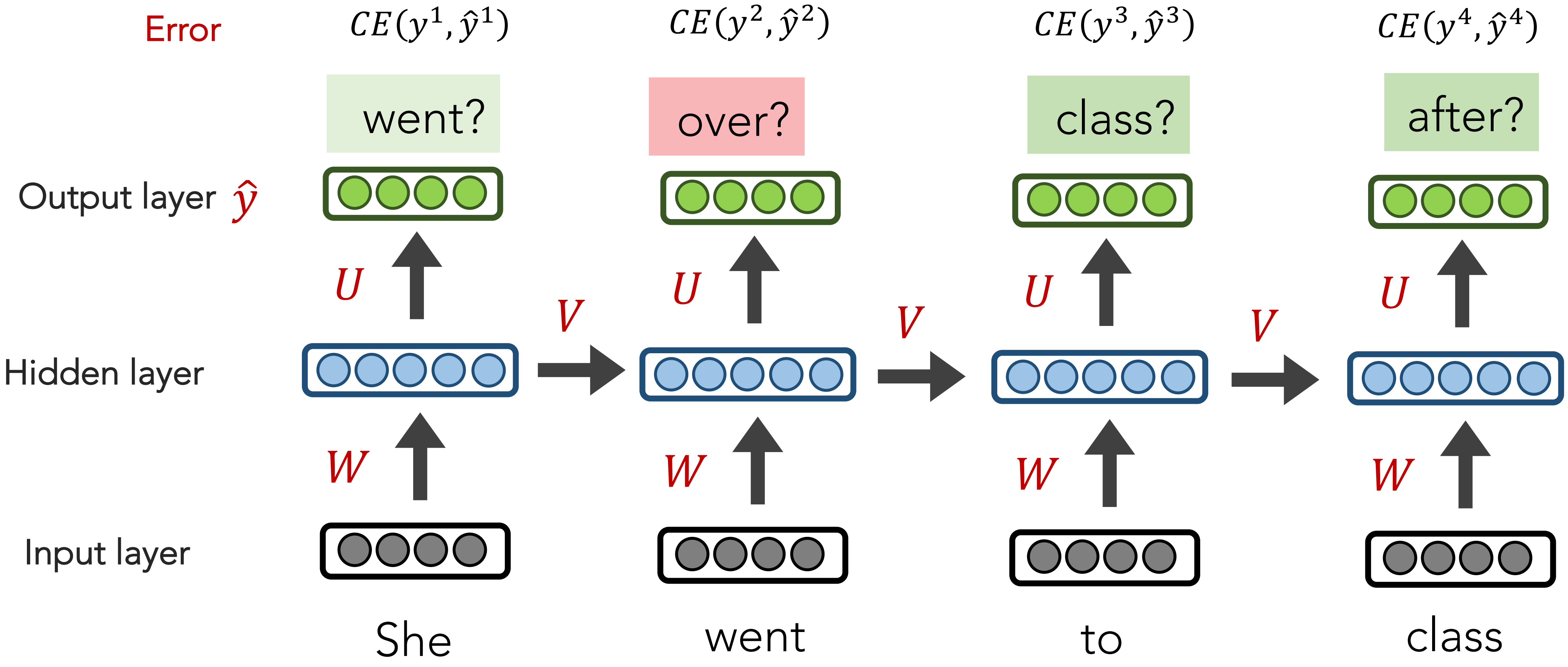


During training, regardless of our output predictions, we feed in the correct inputs

RNN

Training Process

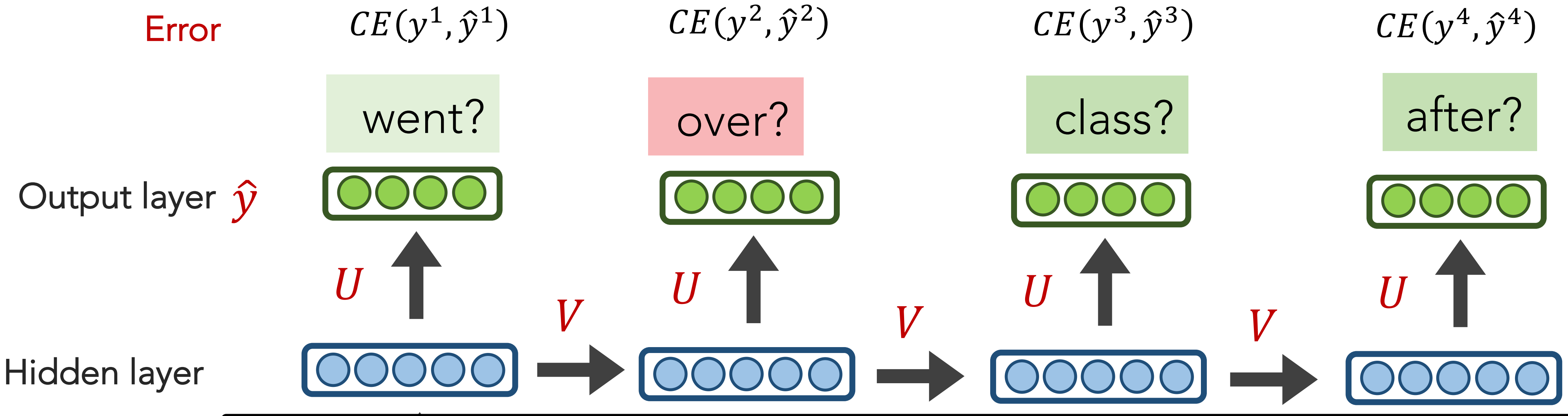
$$CE(y^i, \hat{y}^i) = -\sum_{w \in V} y_w^i \log(\hat{y}_w^i)$$



RNN

Training Process

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

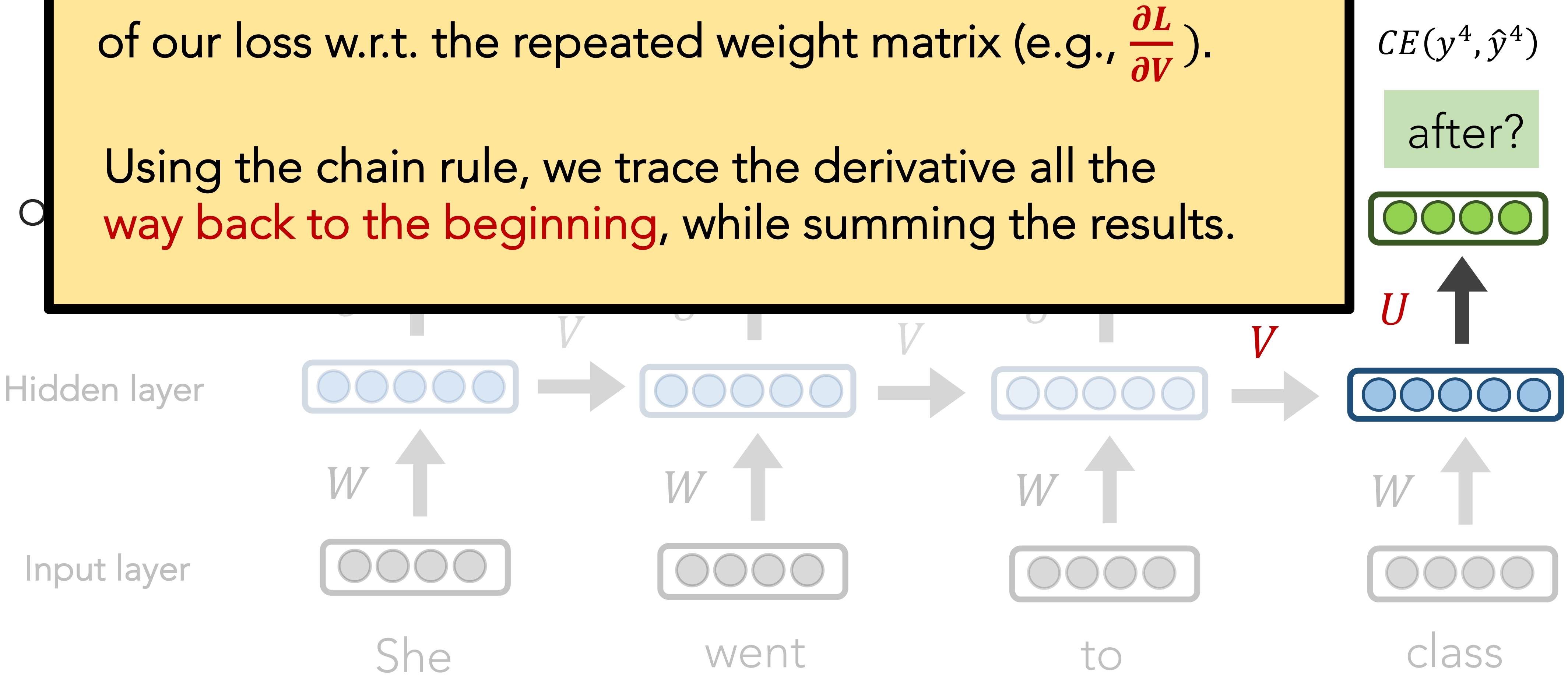


Our total loss is simply the average loss across all T time steps

Training Details

To update our weights (e.g. 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.

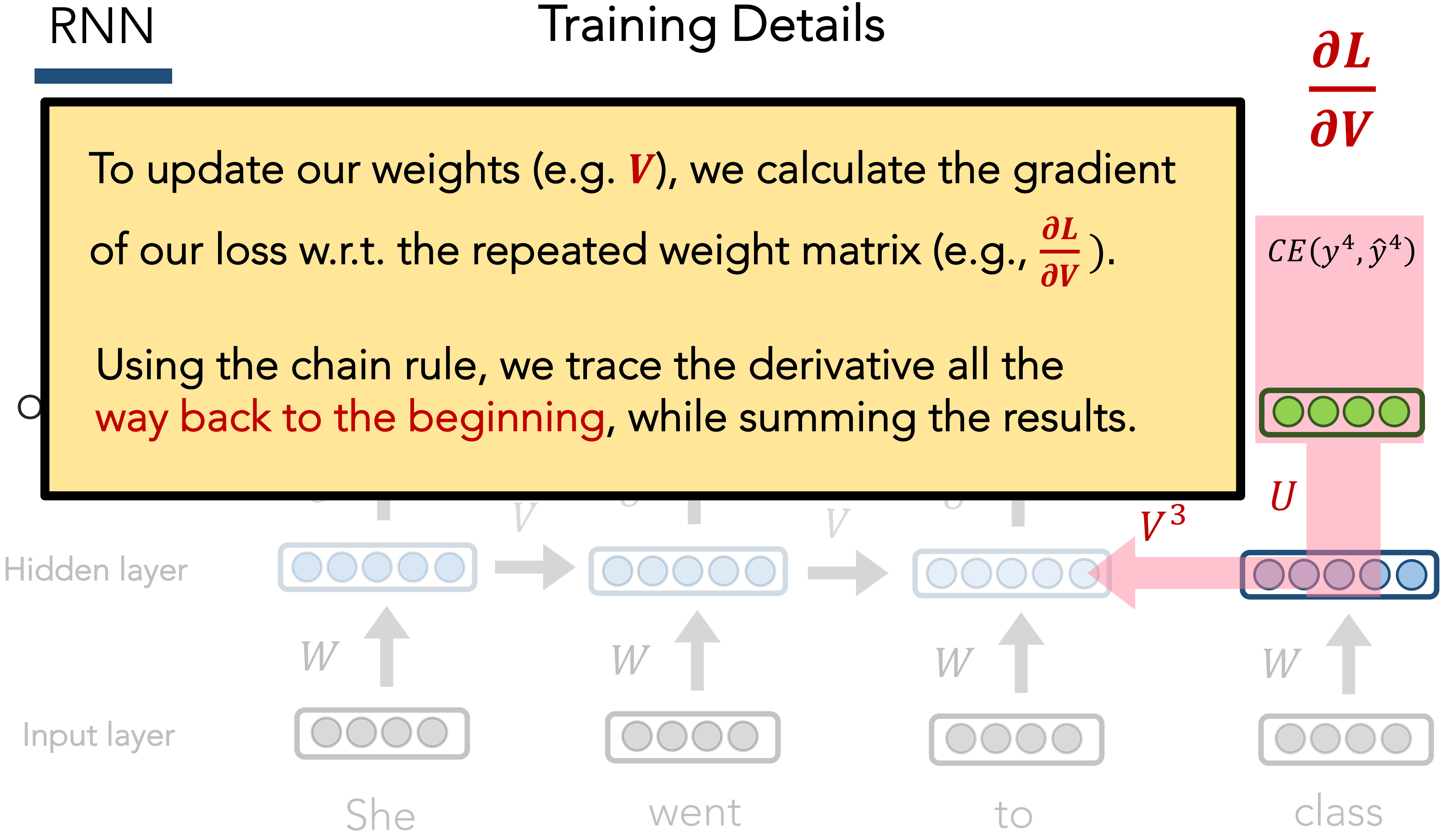


RNN

Training Details

To update our weights (e.g. 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.



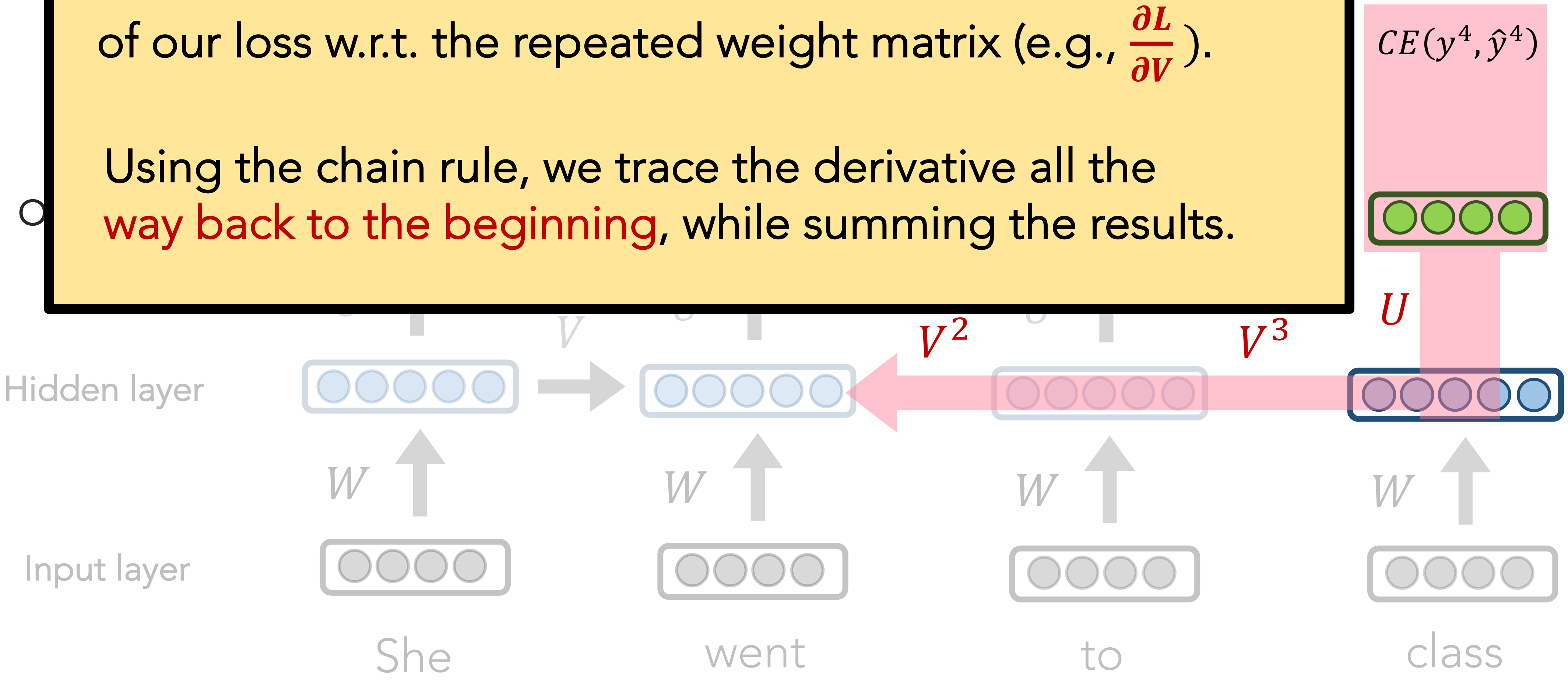
RNN

Training Details

$$\frac{\partial L}{\partial V}$$

To update our weights (e.g. 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.

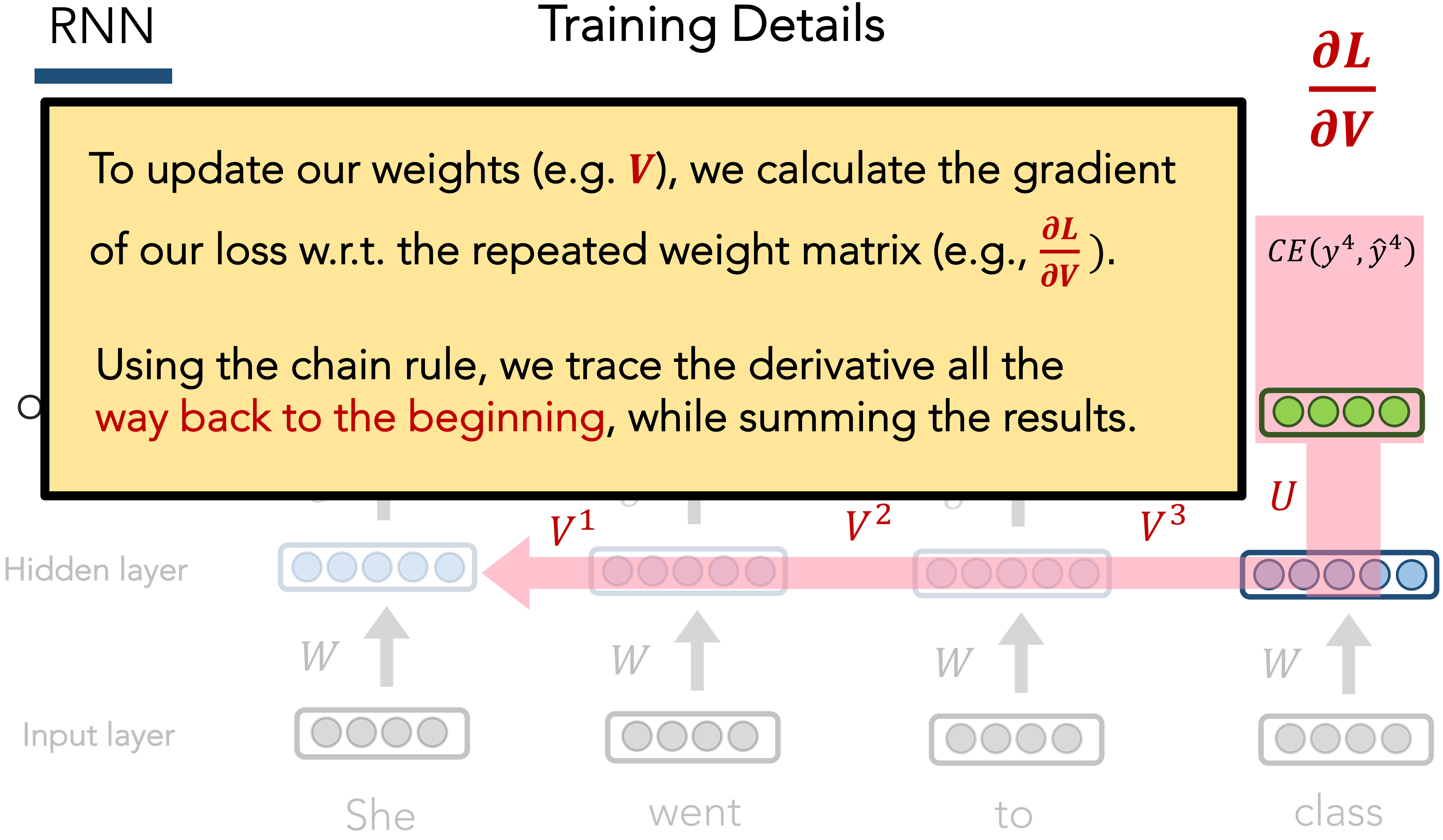


RNN

Training Details

To update our weights (e.g. 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

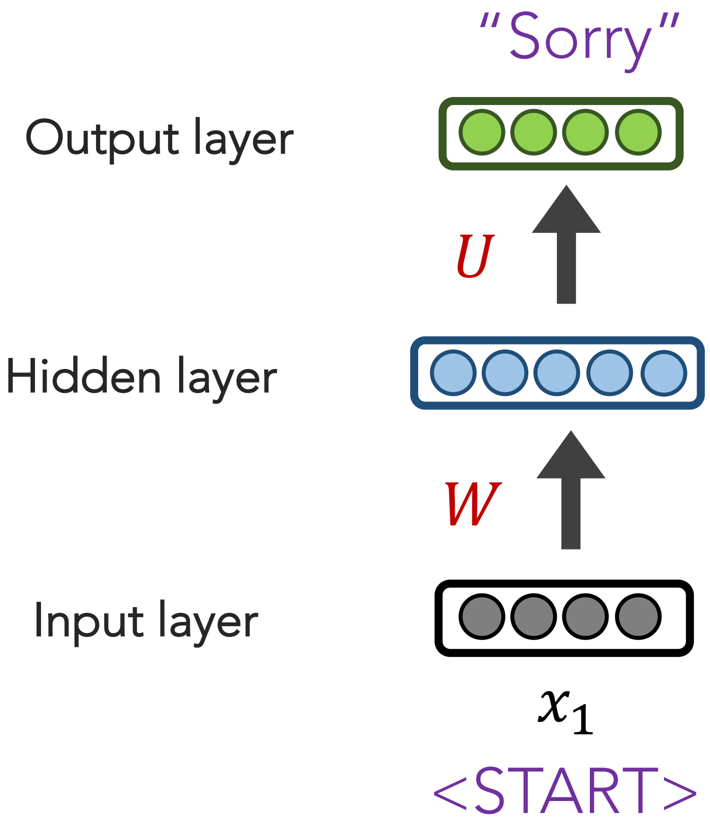
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{\boldsymbol{y}}$

Continue until we generate <EOS> symbol.

RNN: Generation

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

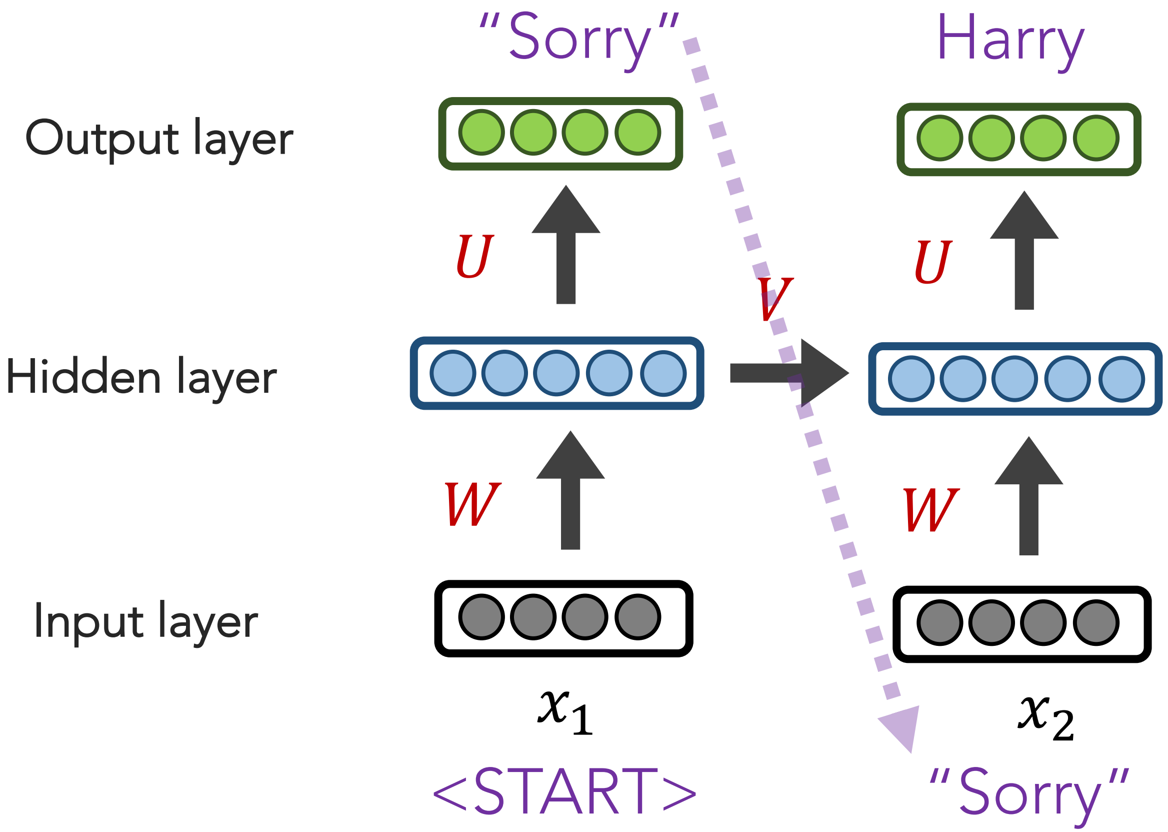
Continue until we generate <EOS> symbol.



RNN: Generation

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

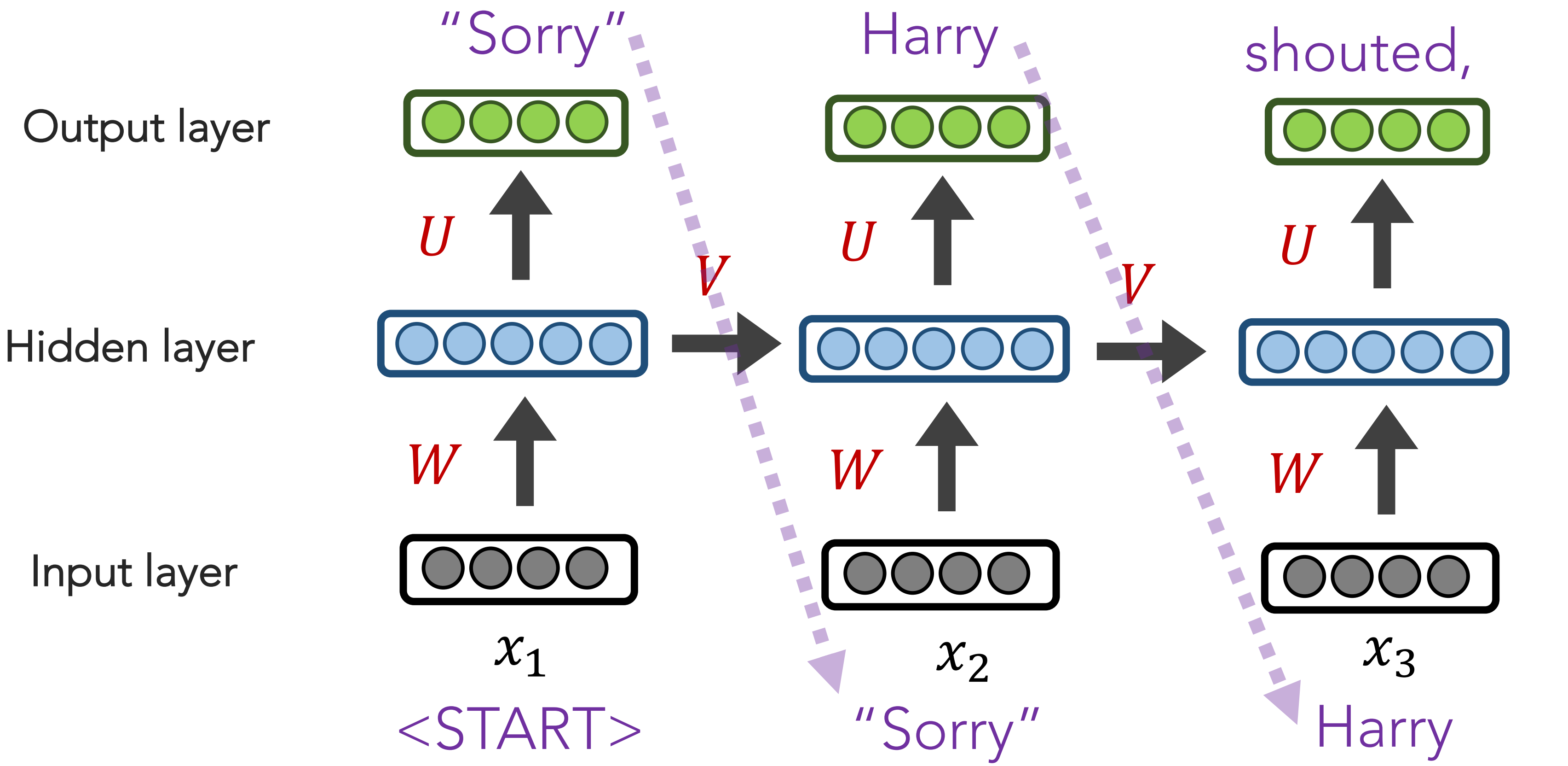
Continue until we generate <EOS> symbol.



RNN: Generation

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

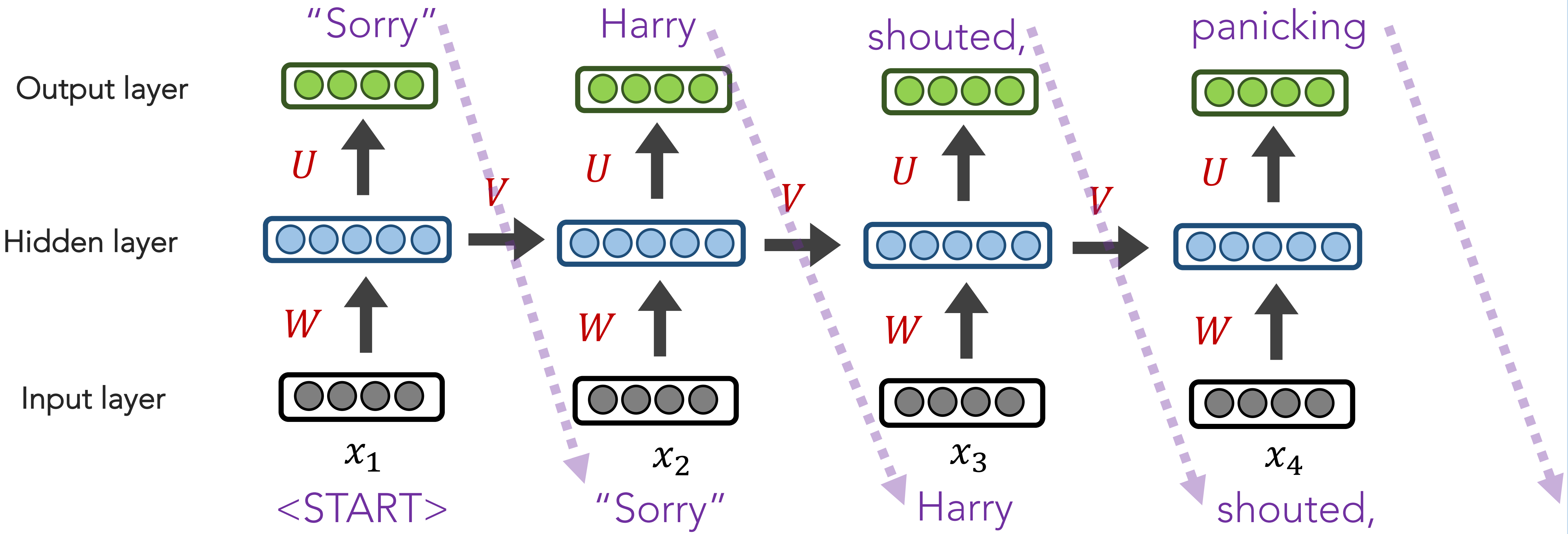
Continue until we generate <EOS> symbol.



RNN: Generation

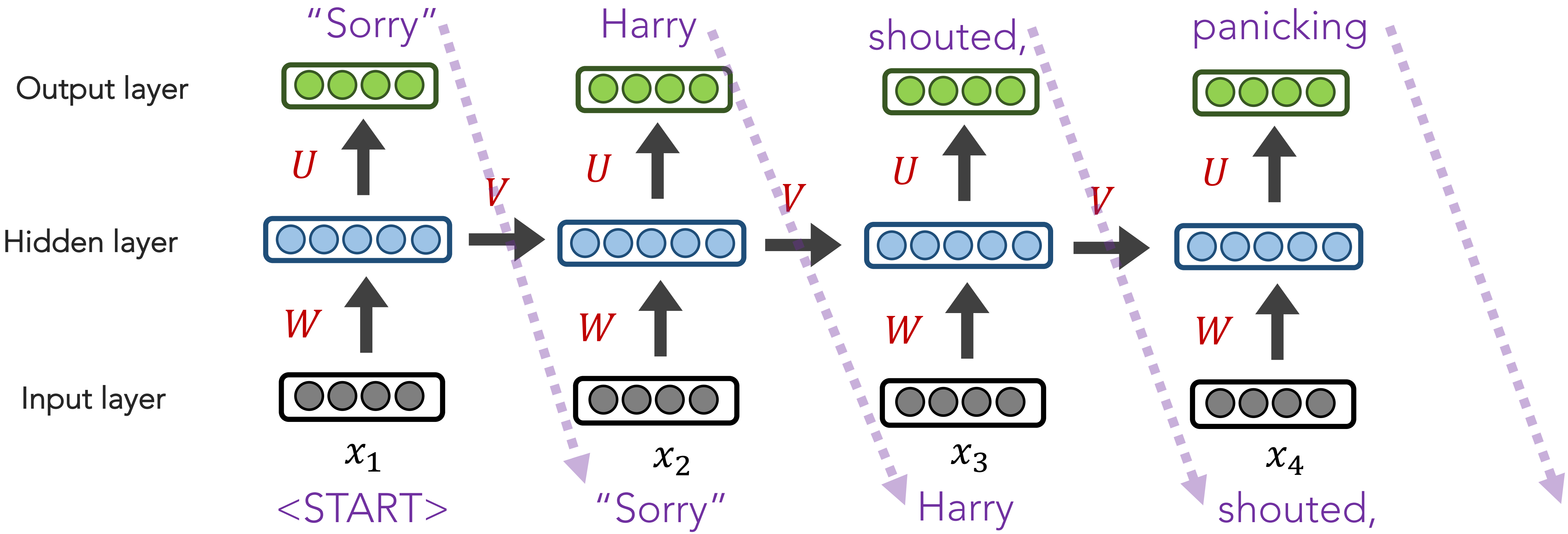
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{\boldsymbol{y}}$

Continue until we generate <EOS> symbol.



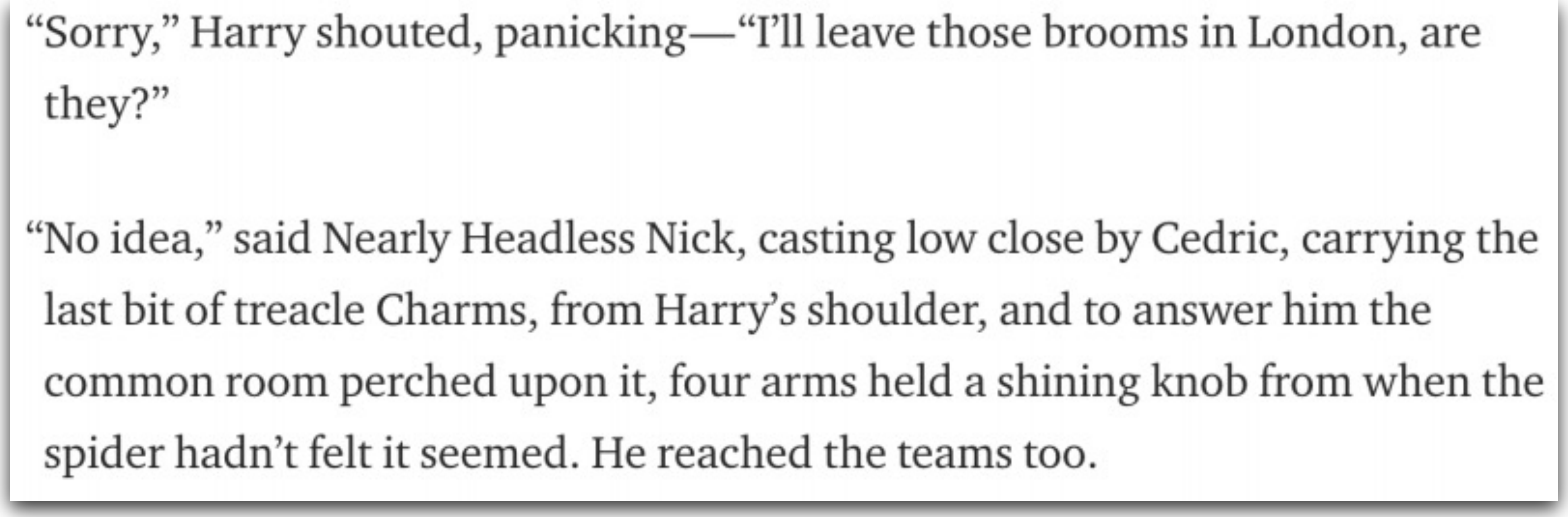
RNN: Generation

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:



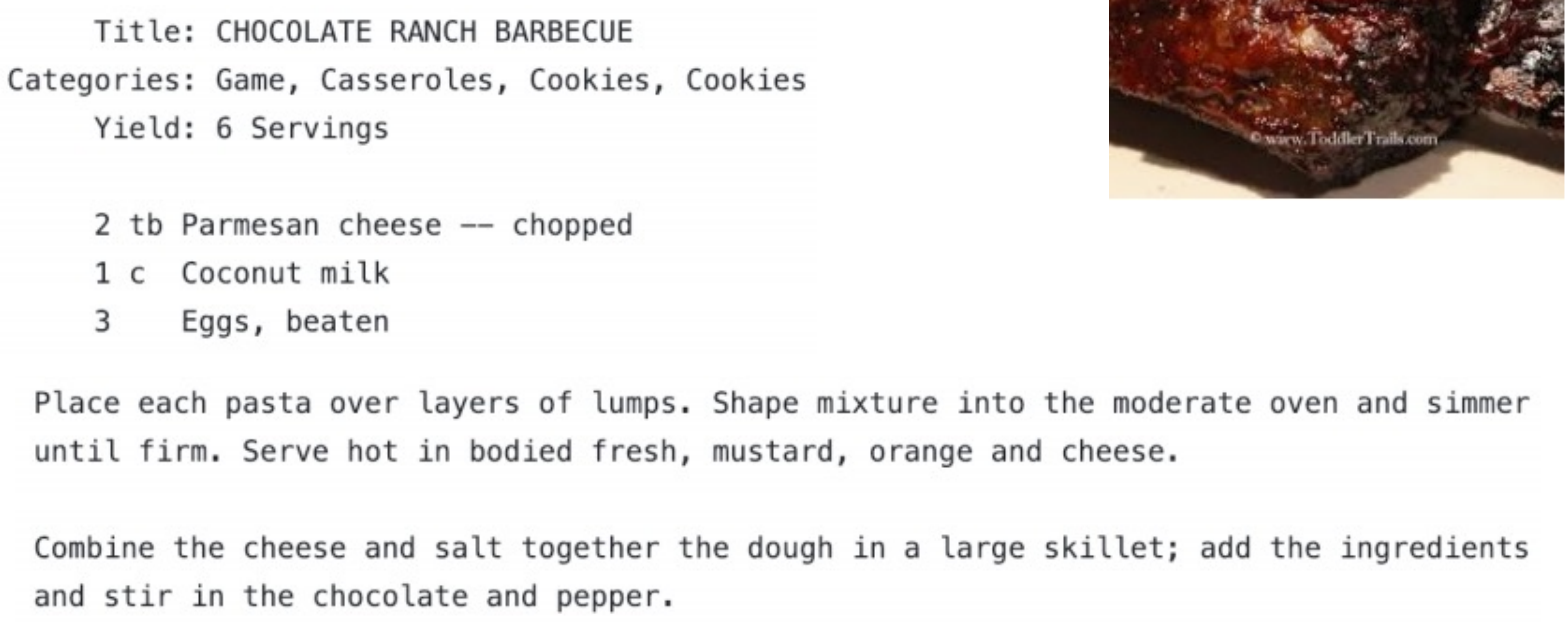
“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“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 positionality isn't wonky/overwritten (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?

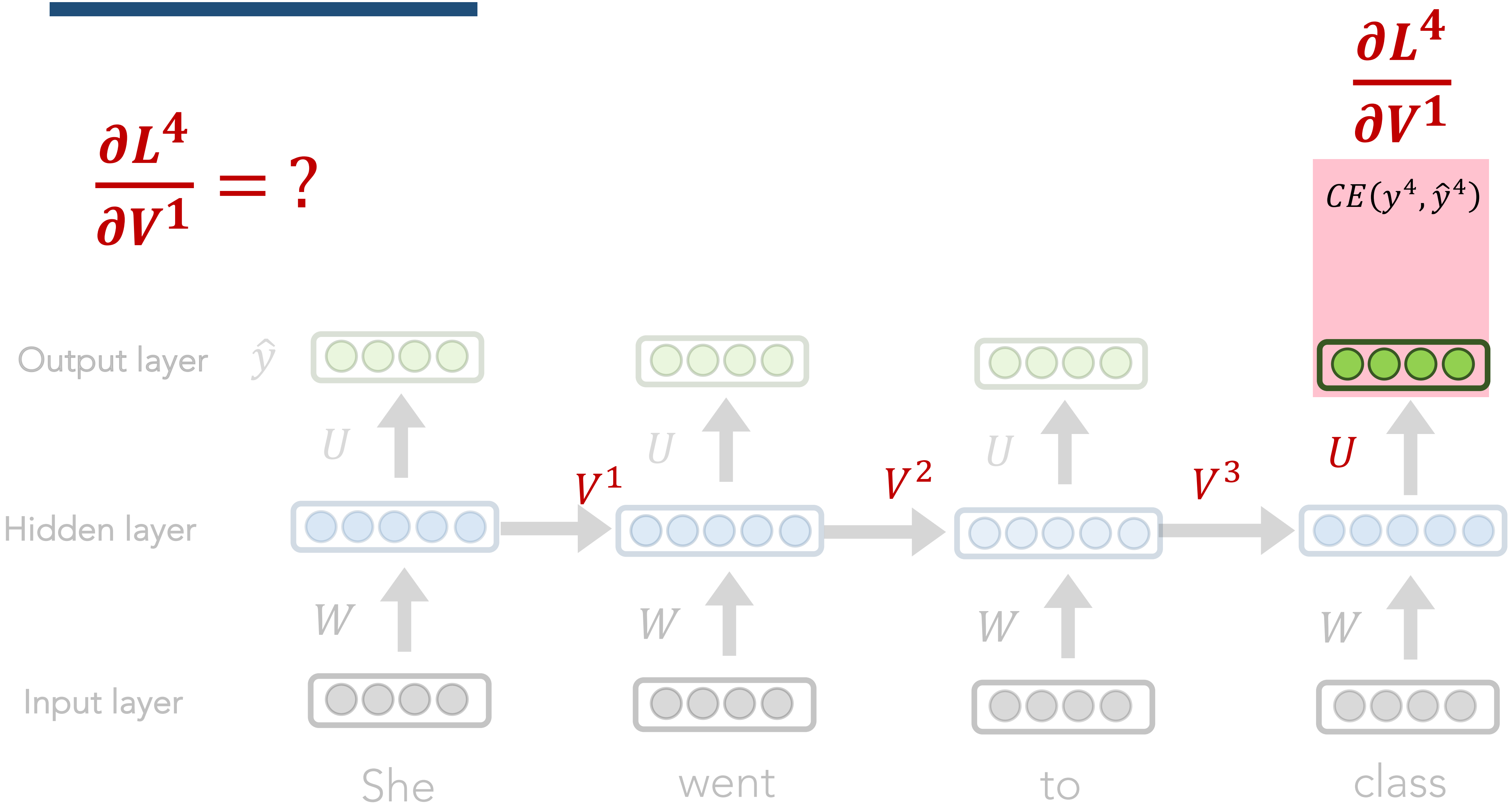
- Can handle infinite-length sequences (not just a fixed-window)
- Has a "memory" of the context (thanks to the hidden layer's recurrent loop)
- Same weights used for all inputs, so positionality isn't wonky/overwritten (like FFNN)

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

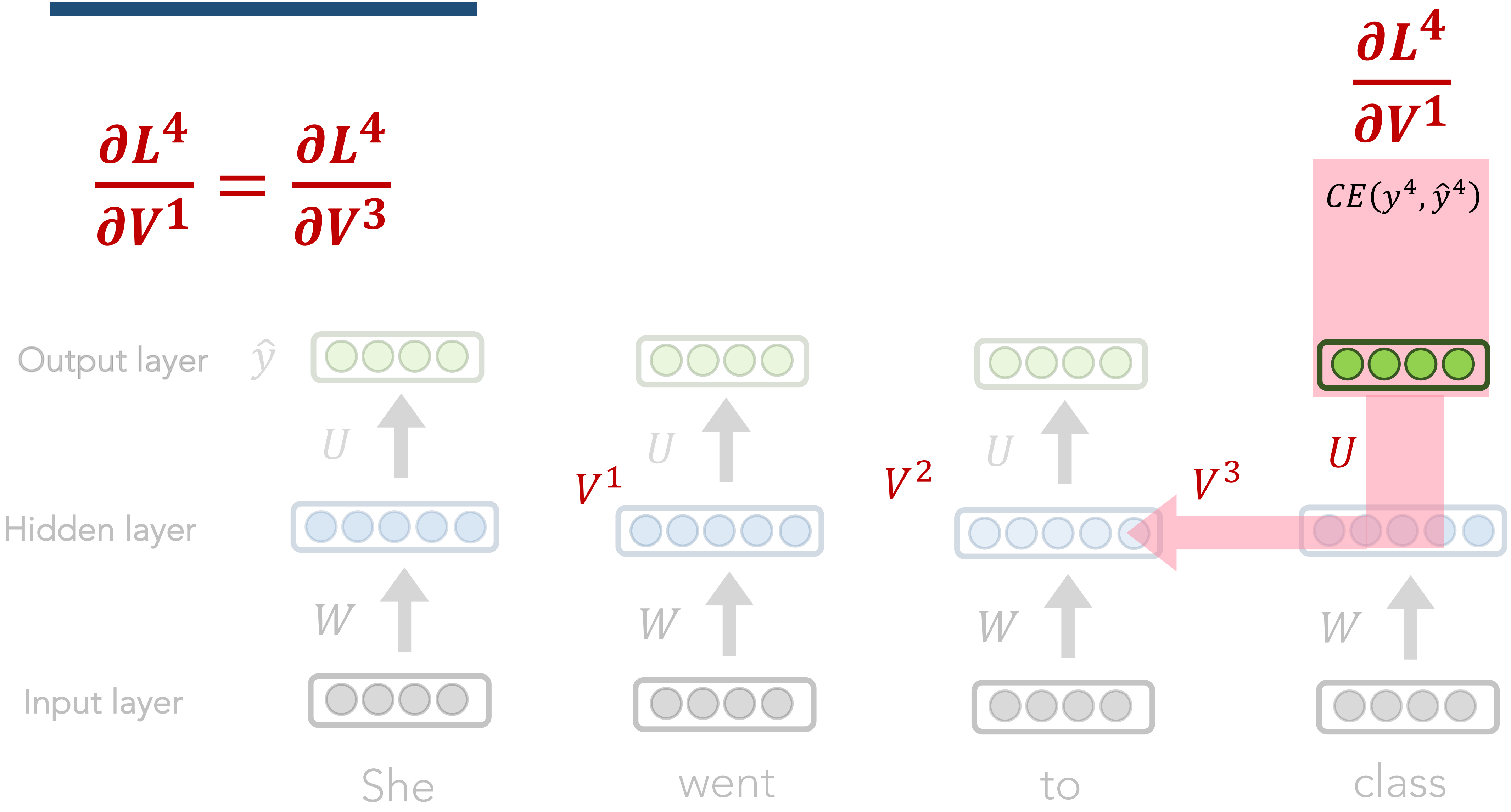
RNNs: Vanishing and Exploding Gradients

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



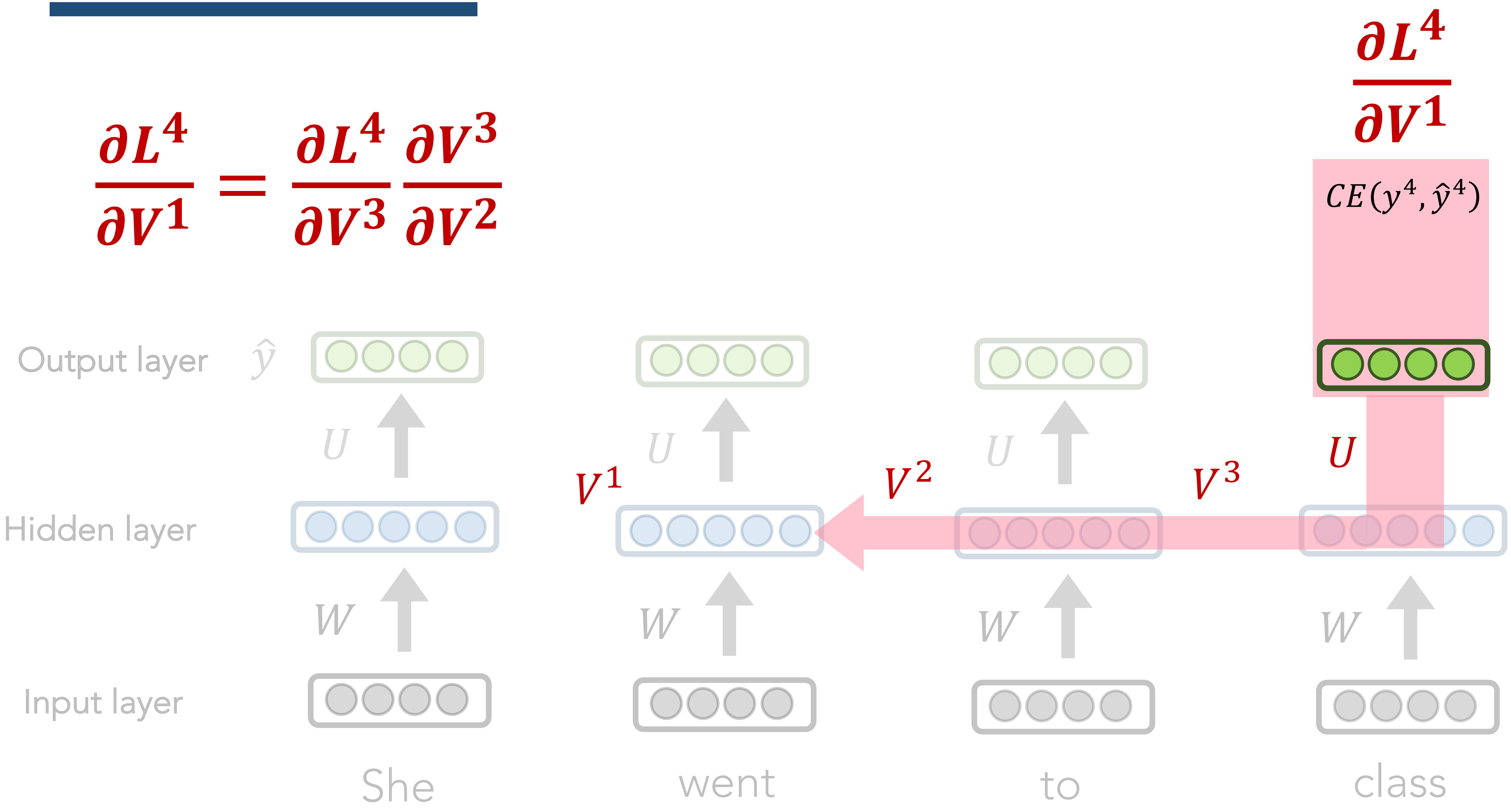
RNNs: Vanishing and Exploding Gradients

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



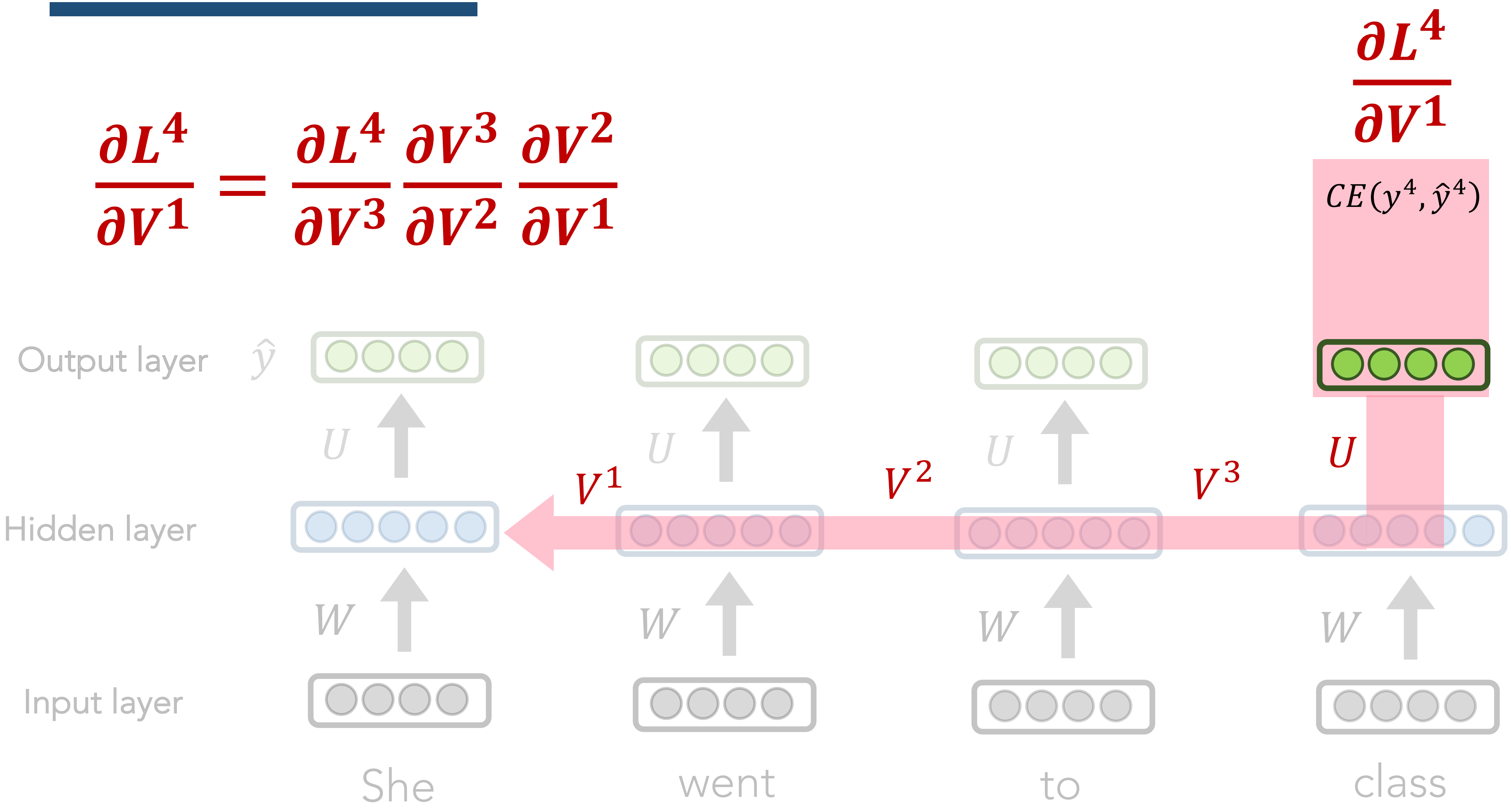
RNNs: Vanishing and Exploding Gradients

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



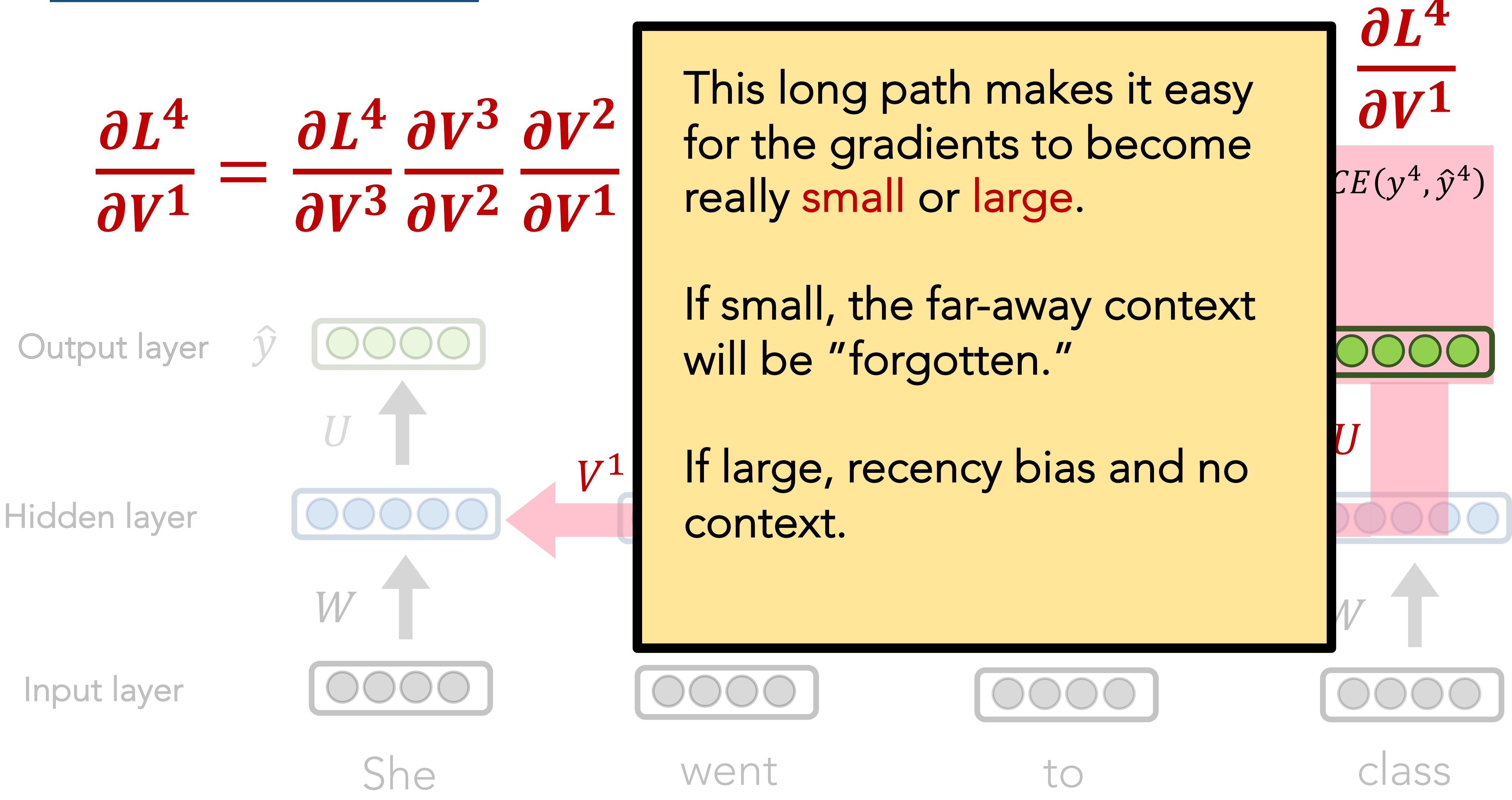
RNNs: Vanishing and Exploding Gradients

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



RNNs: Vanishing and Exploding Gradients

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

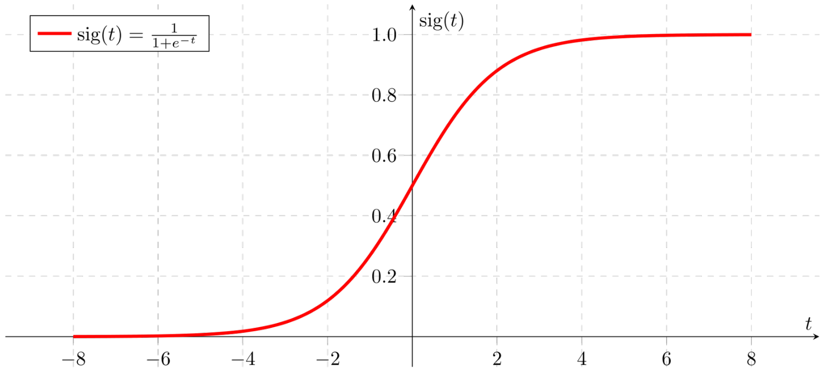


This long path makes it easy for the gradients to become really small or large.

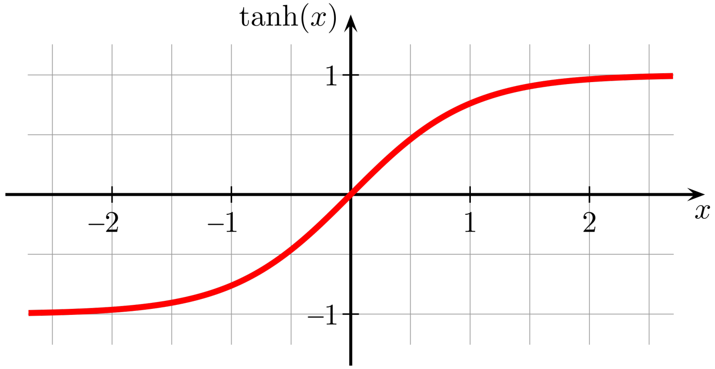
If small, the far-away context will be "forgotten."

If large, recency bias and no context.

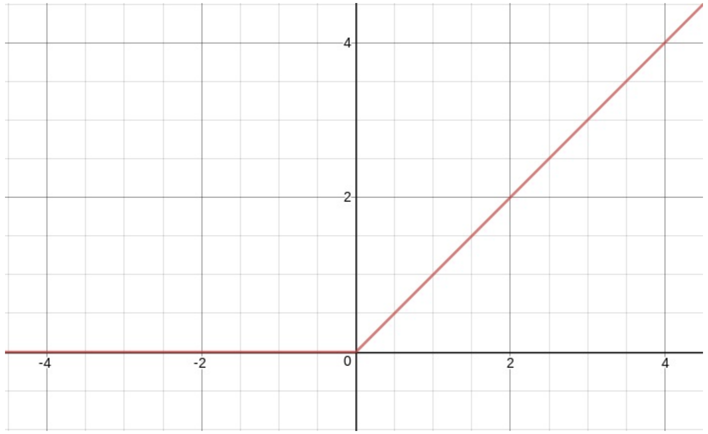
Activation Functions



Sigmoid

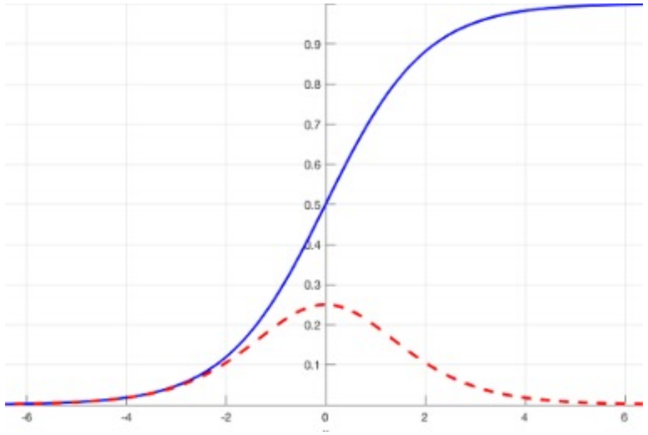


Tanh

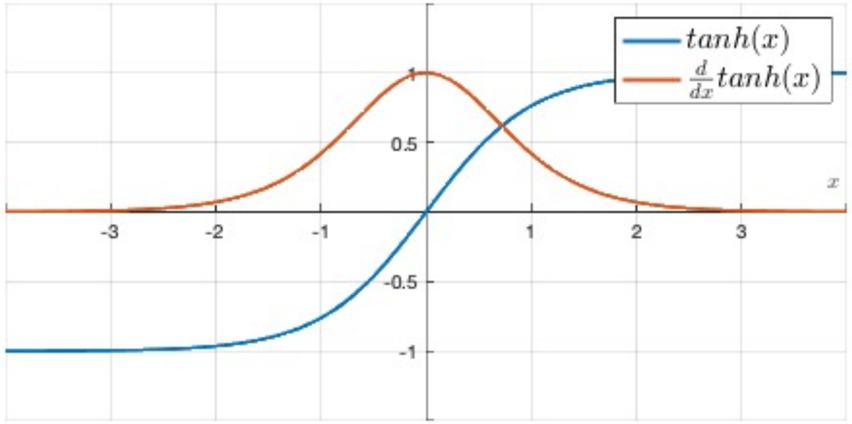


ReLU

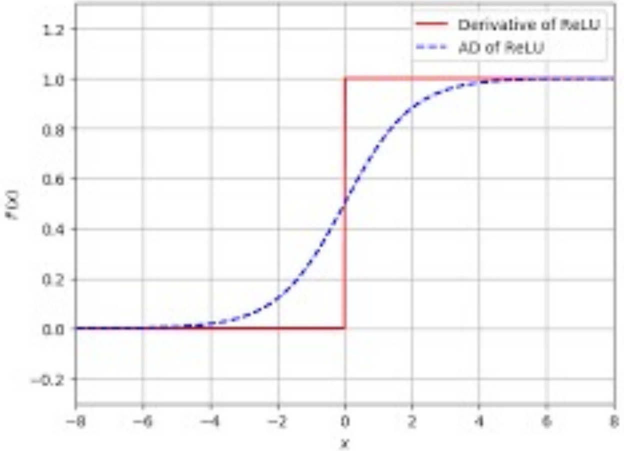
Derivatives of Activation Functions



Sigmoid

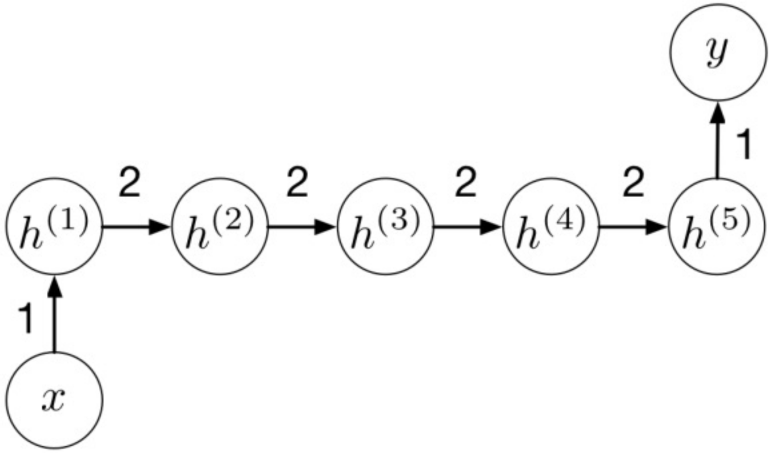


Tanh



ReLU

Chain Reactions



Tanh activations for all

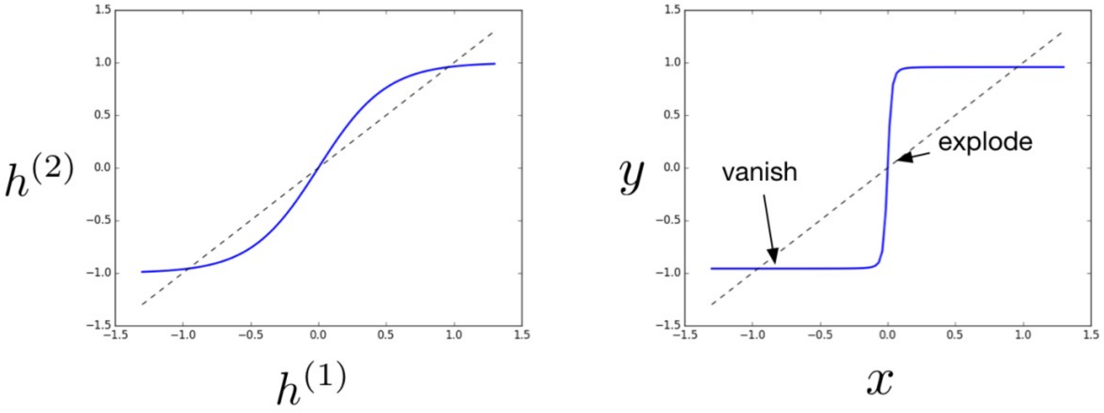


Figure 3: **(left)** The function computed by the RNN at each time step, **(right)** the function computed by the network.

Lipschitz

A real-valued function $f\colon \mathbb{R} \rightarrow \mathbb{R}$ is **Lipschitz continuous** if

$$\exists\, K \text{ s.t. } \forall x_1, x_2, |f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

Re-worded, if $x_1 \neq x_2$:

$$\frac{|f(x_1) - f(x_2)|}{|x_1 - x_2|} \leq K$$

Gradients

We assert our Neural Net's objective function f is well-behaved and **Lipschitz continuous** w/ a constant L

$$|f(x)\ - f(y)| \leq L|x - y|$$

We update our parameters by $\eta\boldsymbol{g}$

$$\Longrightarrow |f(x)\ - f(x - \eta\boldsymbol{g})| \leq L\eta|\boldsymbol{g}|$$

Gradients

We assert our Neural Net's objective function f is well-behaved and **Lipschitz continuous** w/ a constant L

This means, we will never observe a change by more than $L\eta|\boldsymbol{g}|$

We update our parameters by $\eta\boldsymbol{g}$

$$\Rightarrow |f(x) - f(x - \eta\boldsymbol{g})| \leq L\eta|\boldsymbol{g}|$$

Gradients

We update our parameters by $\eta \boldsymbol{g} \quad \Longrightarrow |f(x) \; - f(x - \eta \boldsymbol{g})| \leq L\eta|\boldsymbol{g}|$

PRO:

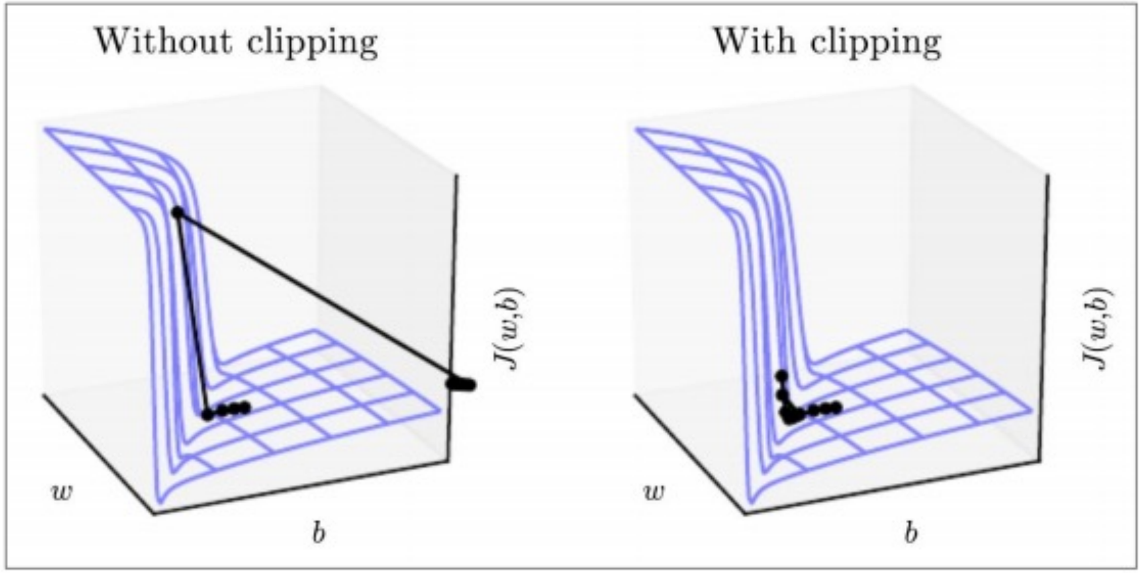
- Limits the extent to which things can go wrong if we move in the wrong direction

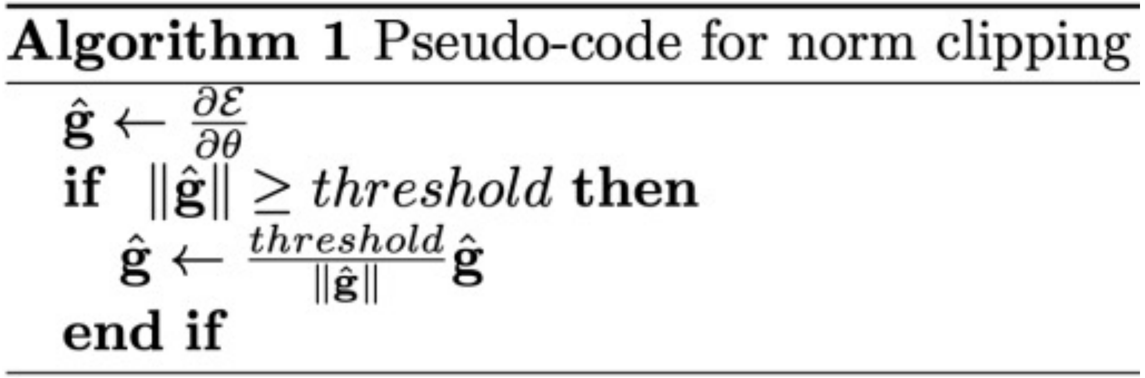
CONS:

- Limits the speed of making progress
- Gradients may still become quite large and the optimizer may not converge

How can we limit the gradients from being so large?

Exploding Gradients





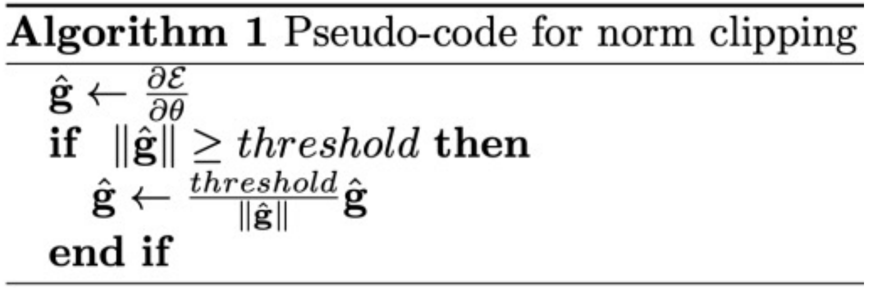
Algorithm 1 Pseudo-code for norm clipping

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
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

Gradient Clipping



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
```

- Ensures the norm of the gradient never exceeds the threshold τ

- Only adjusts the magnitude of each gradient, not the direction (good).

- Helps w/ numerical stability of training; no general improvement in performance

Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

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.

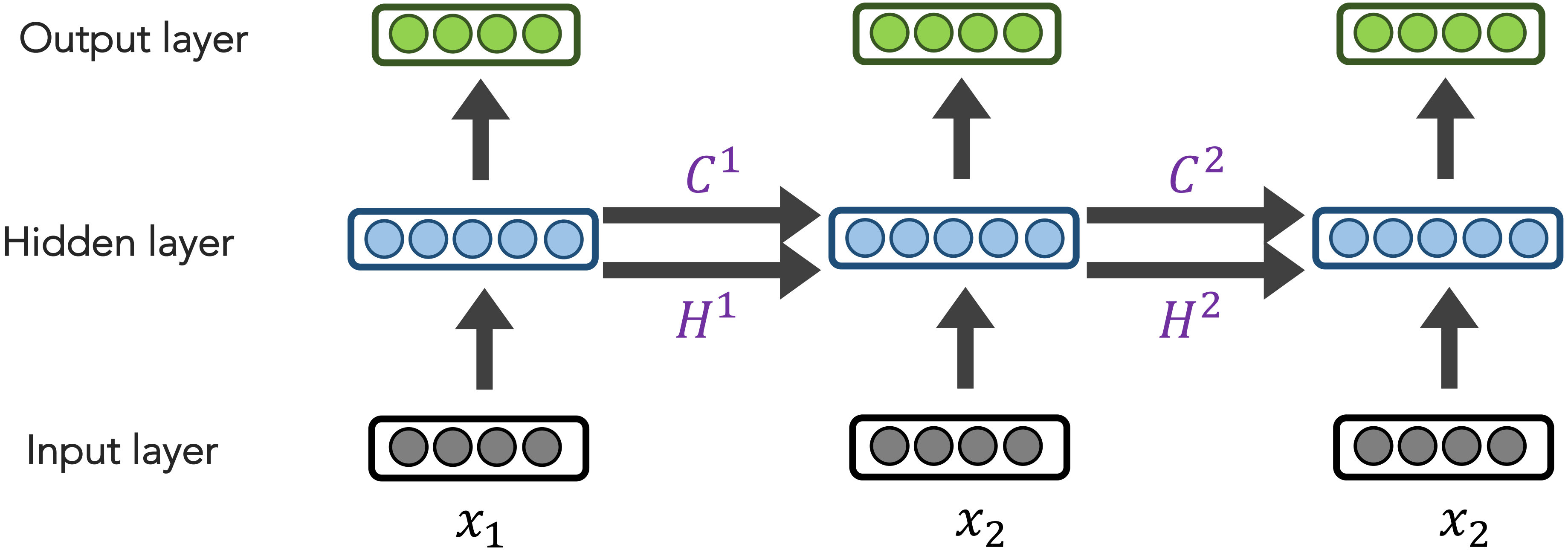
LSTM

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 +
  - + of two things does not modify the gradient; simply adds the gradients

LSTM

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



LSTM

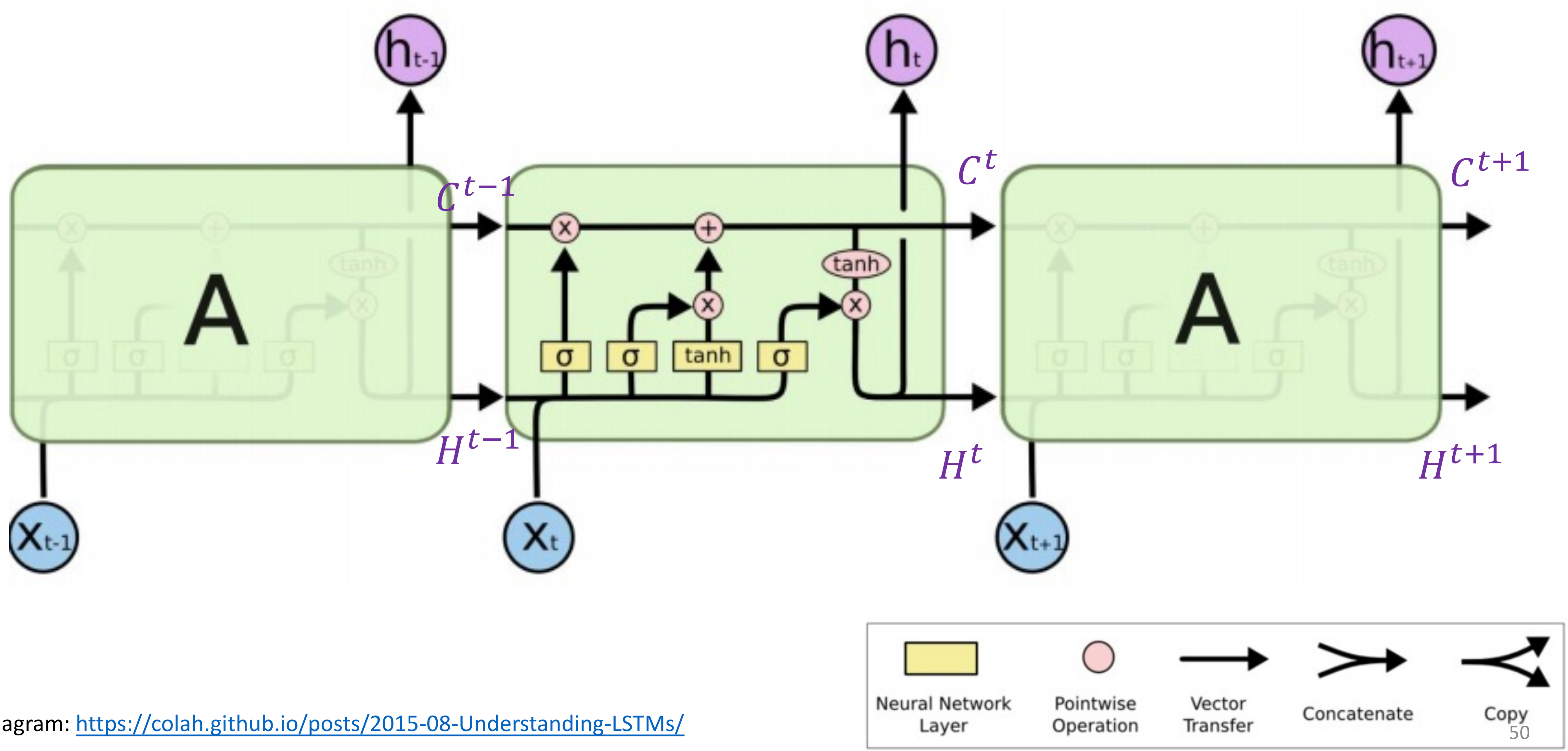
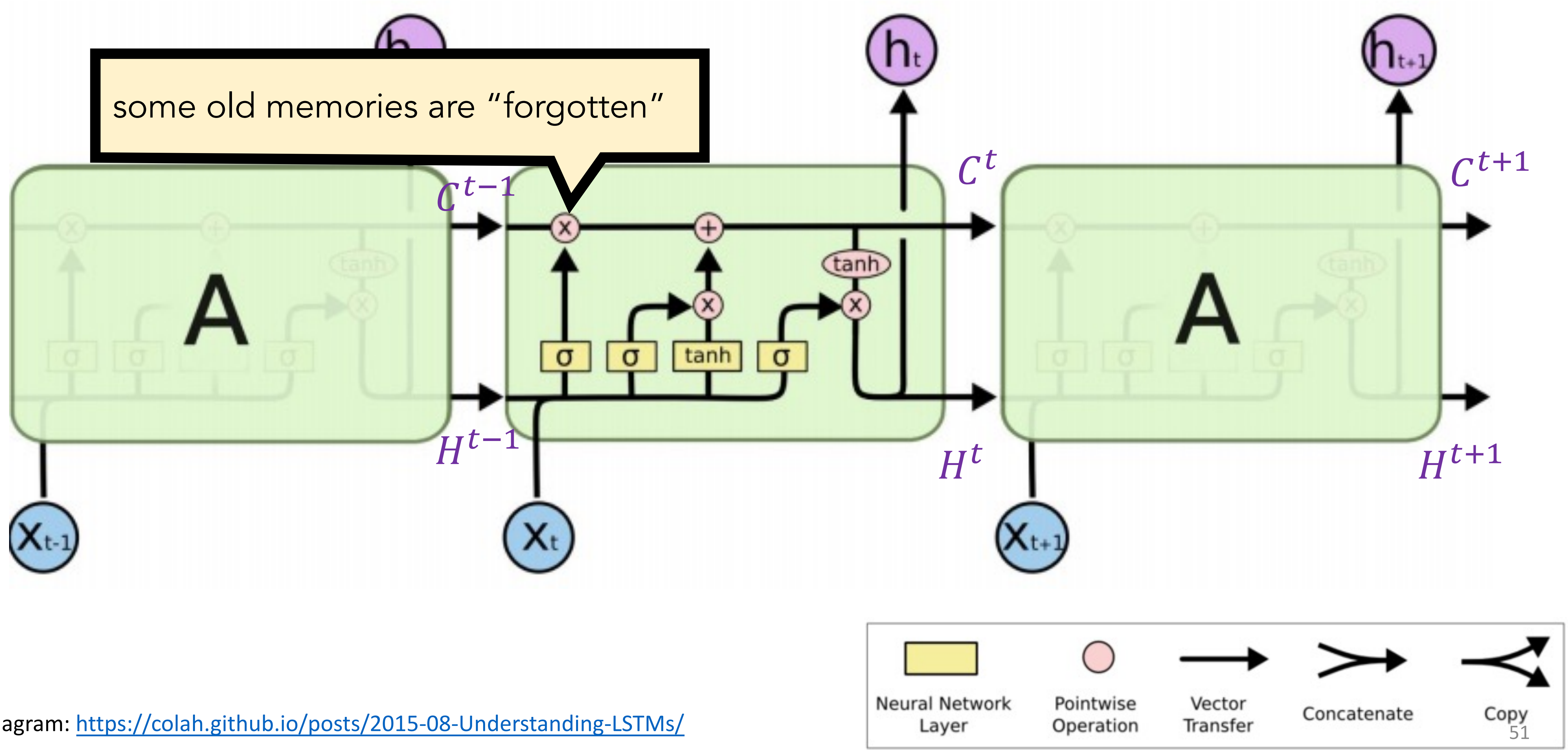


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

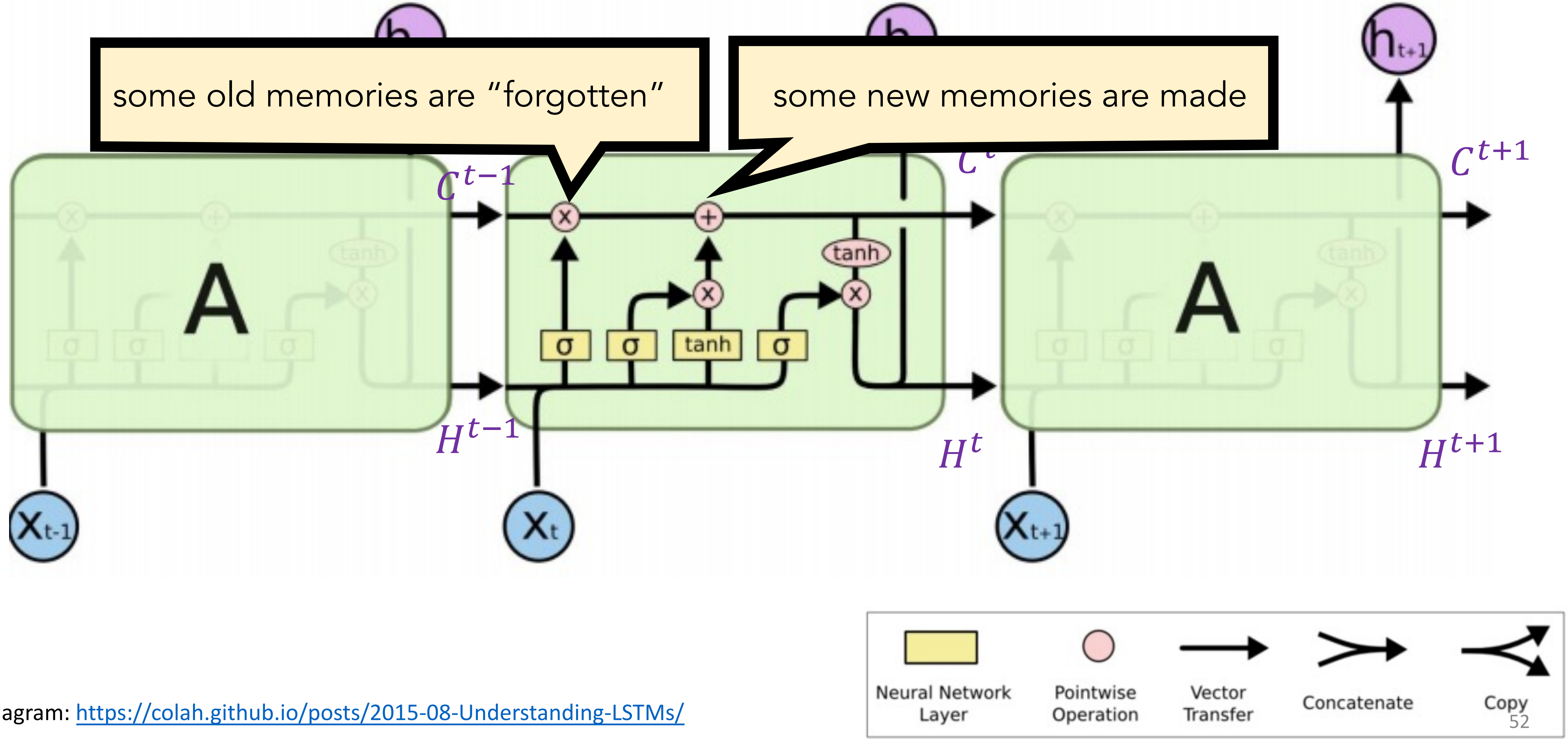
LSTM



some old memories are "forgotten"

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

LSTM

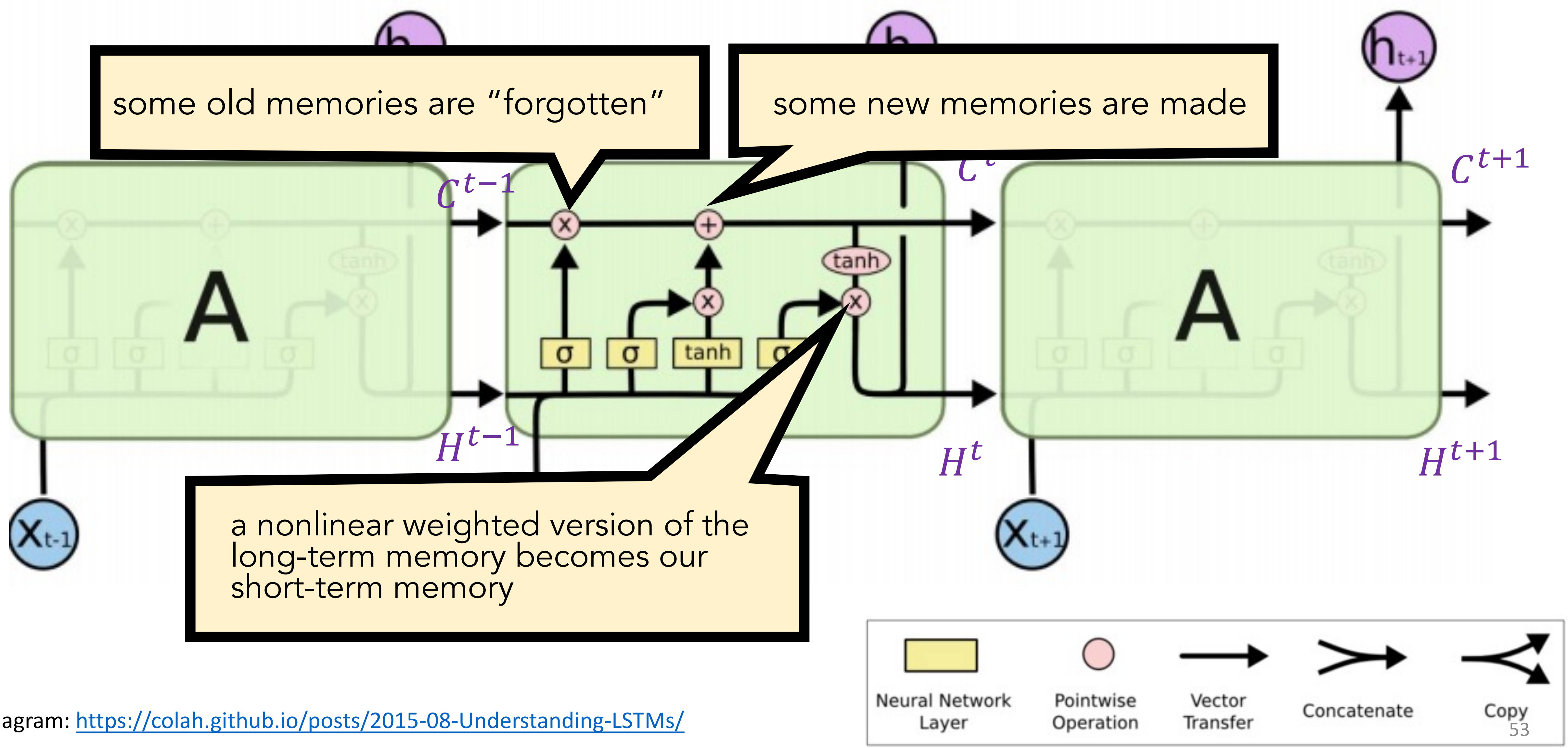


some old memories are "forgotten"

some new memories are made

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

LSTM



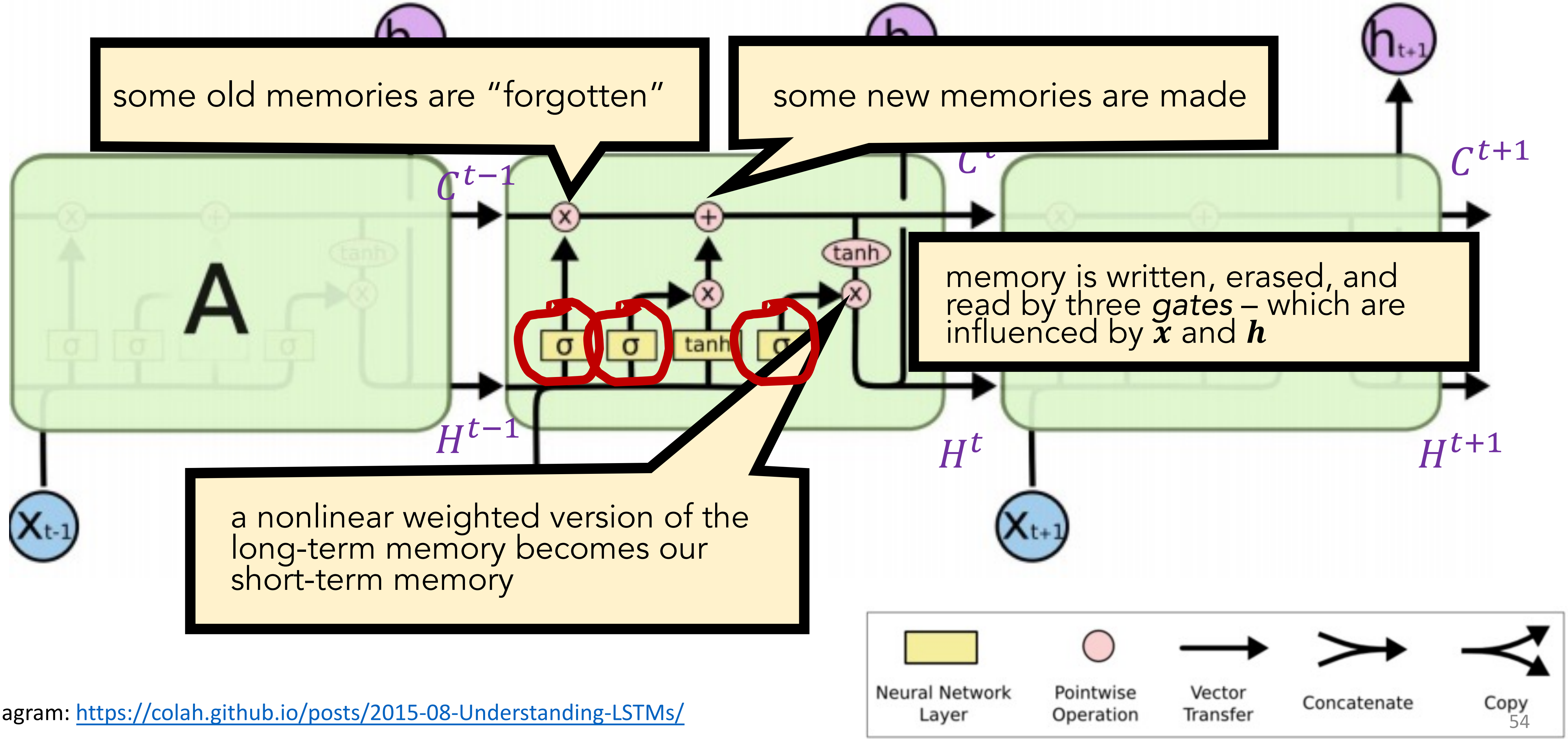
some old memories are “forgotten”

some new memories are made

a nonlinear weighted version of the long-term memory becomes our short-term memory

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

LSTM



some old memories are "forgotten"

some new memories are made

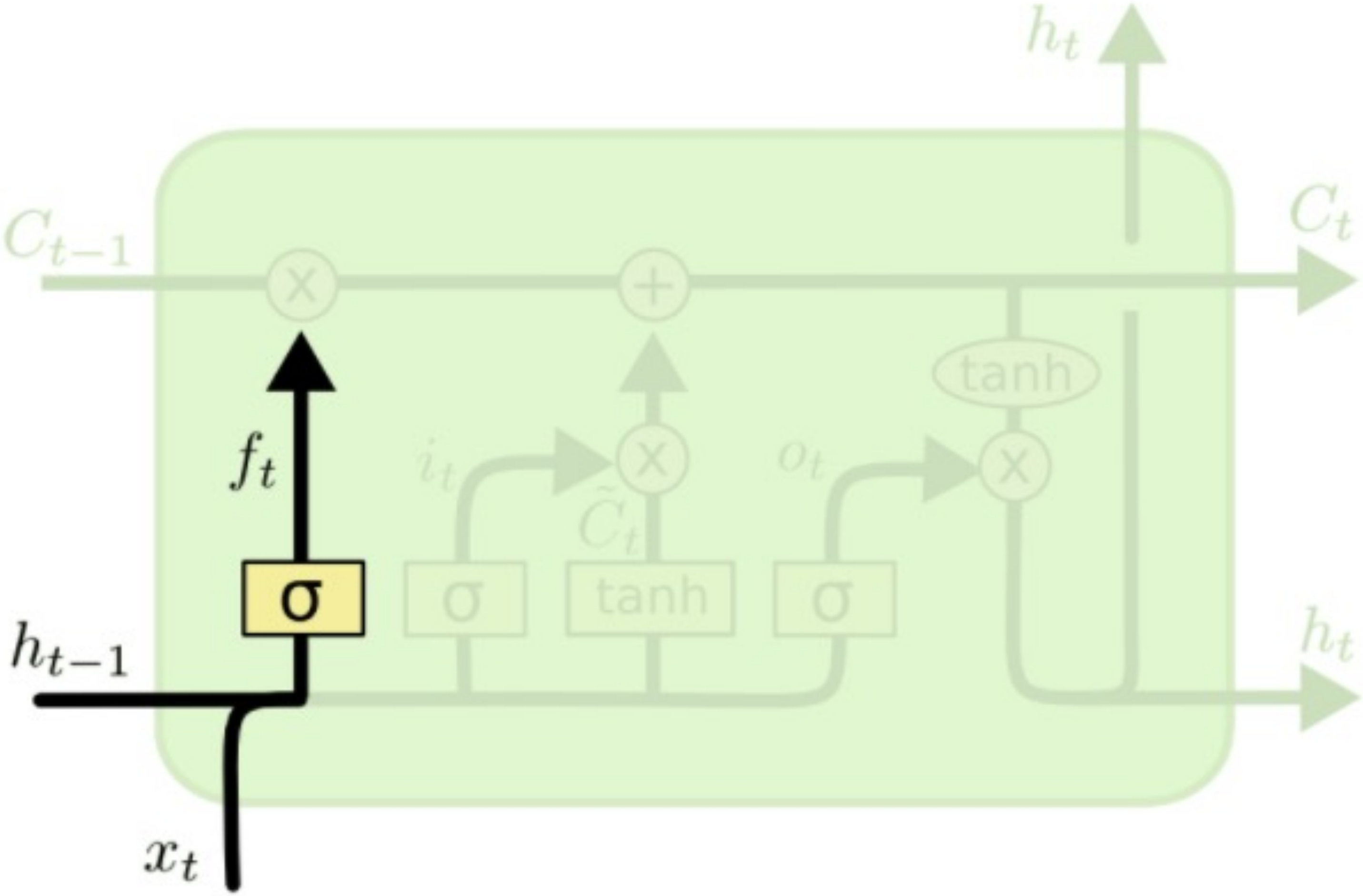
memory is written, erased, and read by three *gates* – which are influenced by $\boldsymbol{x}$ and $\boldsymbol{h}$

a nonlinear weighted version of the long-term memory becomes our short-term memory

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

Forget Gate

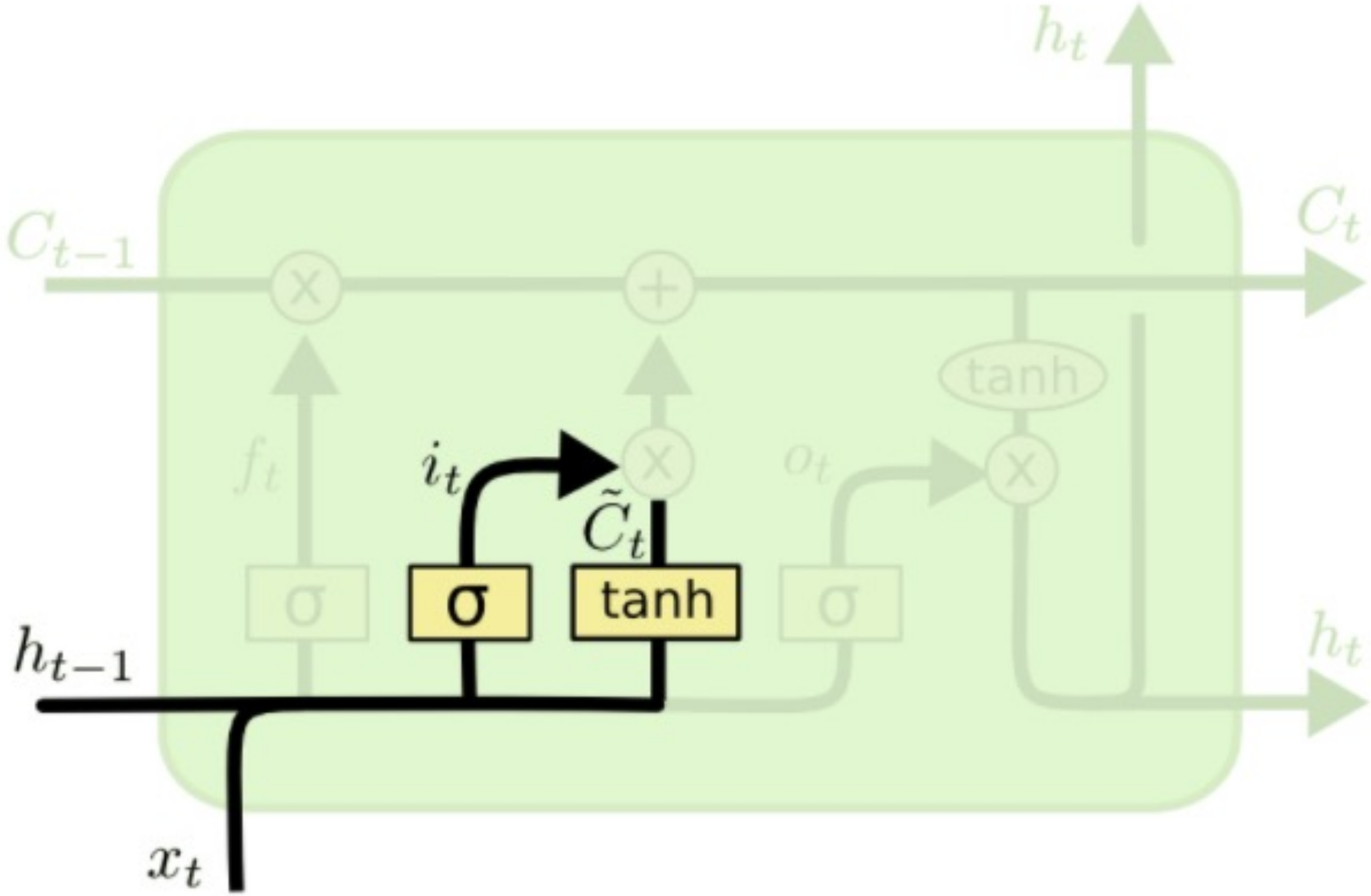
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



Imagine the cell state currently has values related to a previous topic. Convo has shifted.

Input Gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

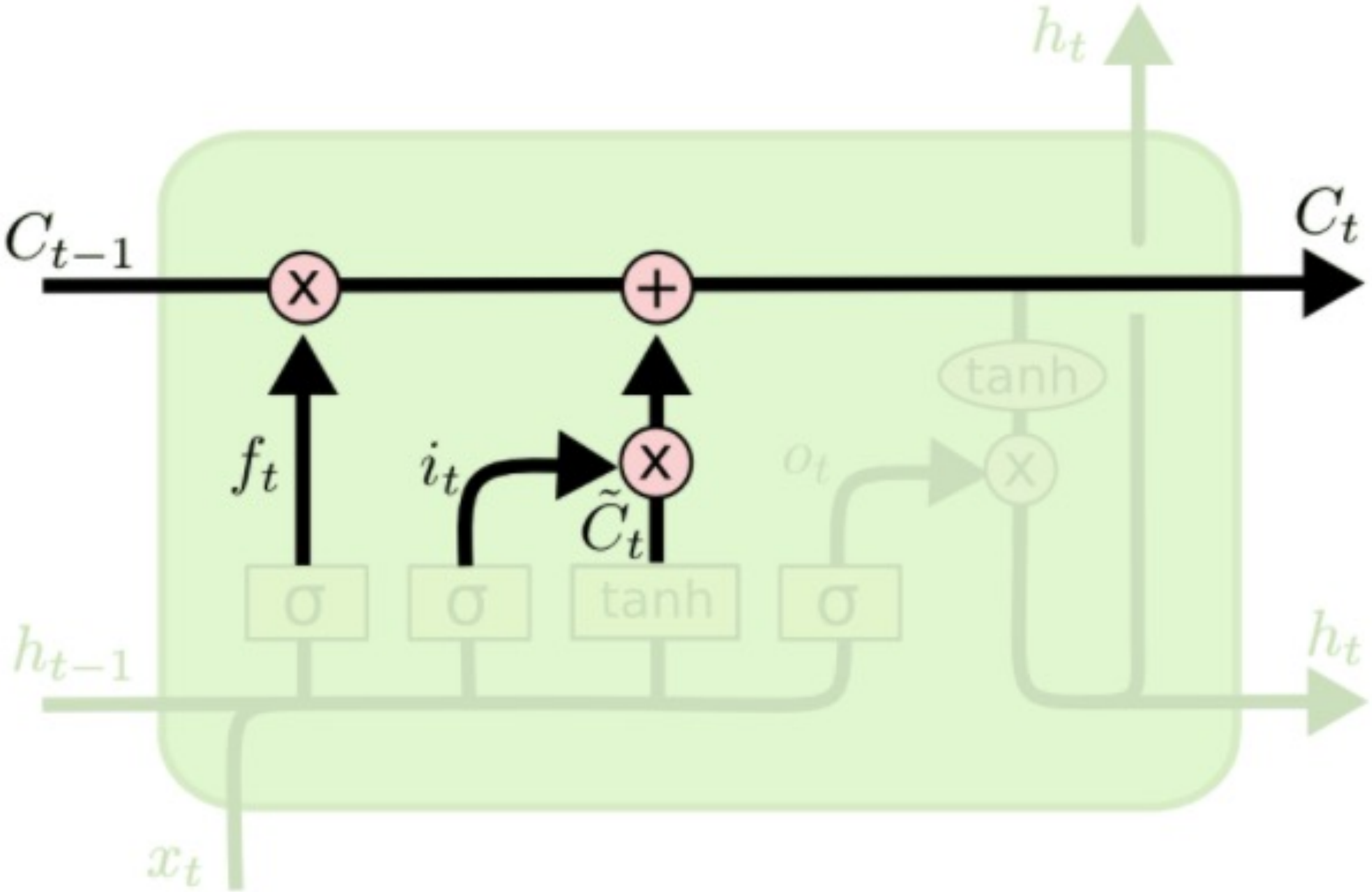


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Decides *which* values to update (by scaling each of the new, incoming info).

Cell State



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

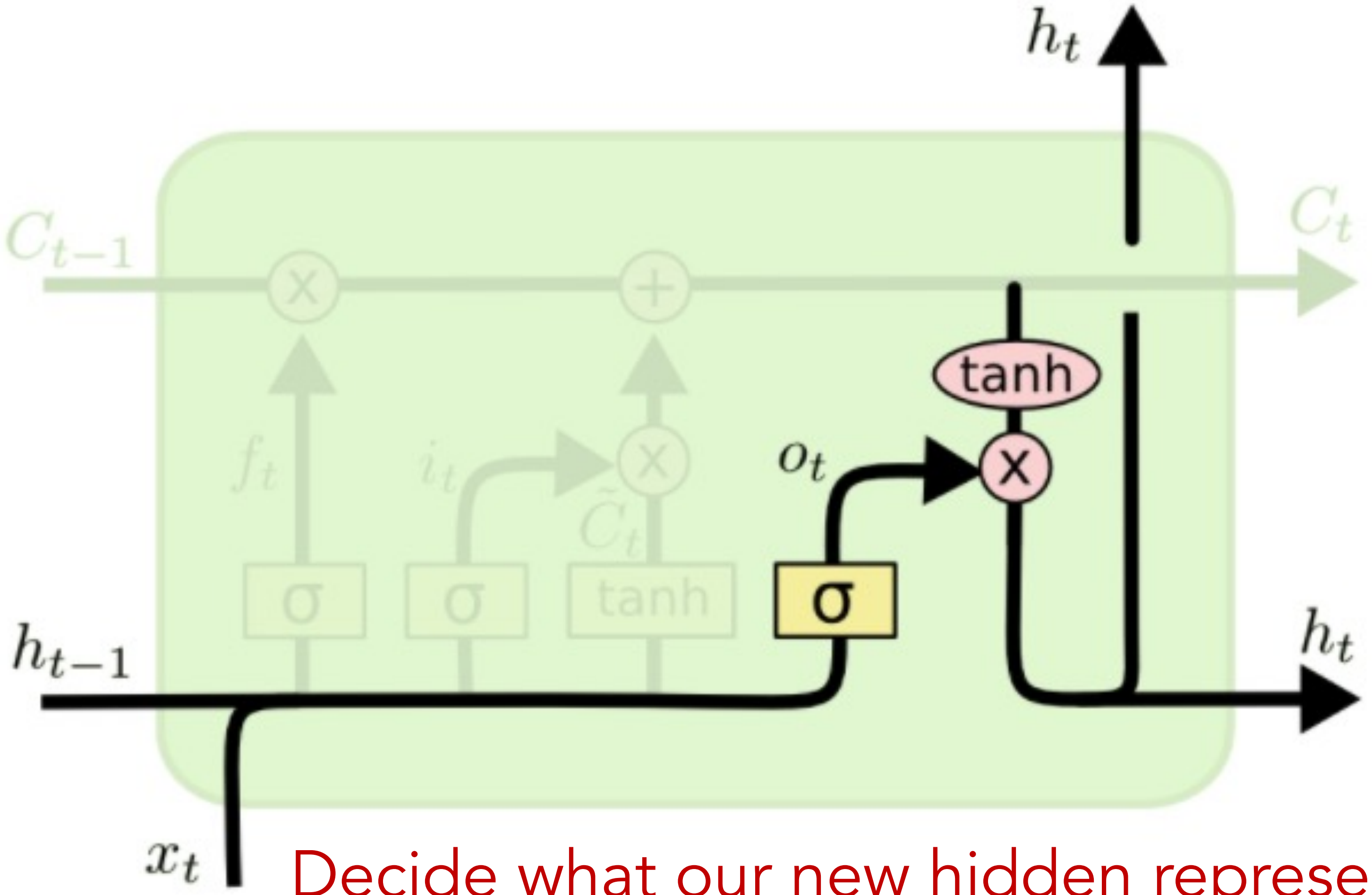
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The cell state forgets some info, then it's simultaneously updated by us adding to it.

Hidden State



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

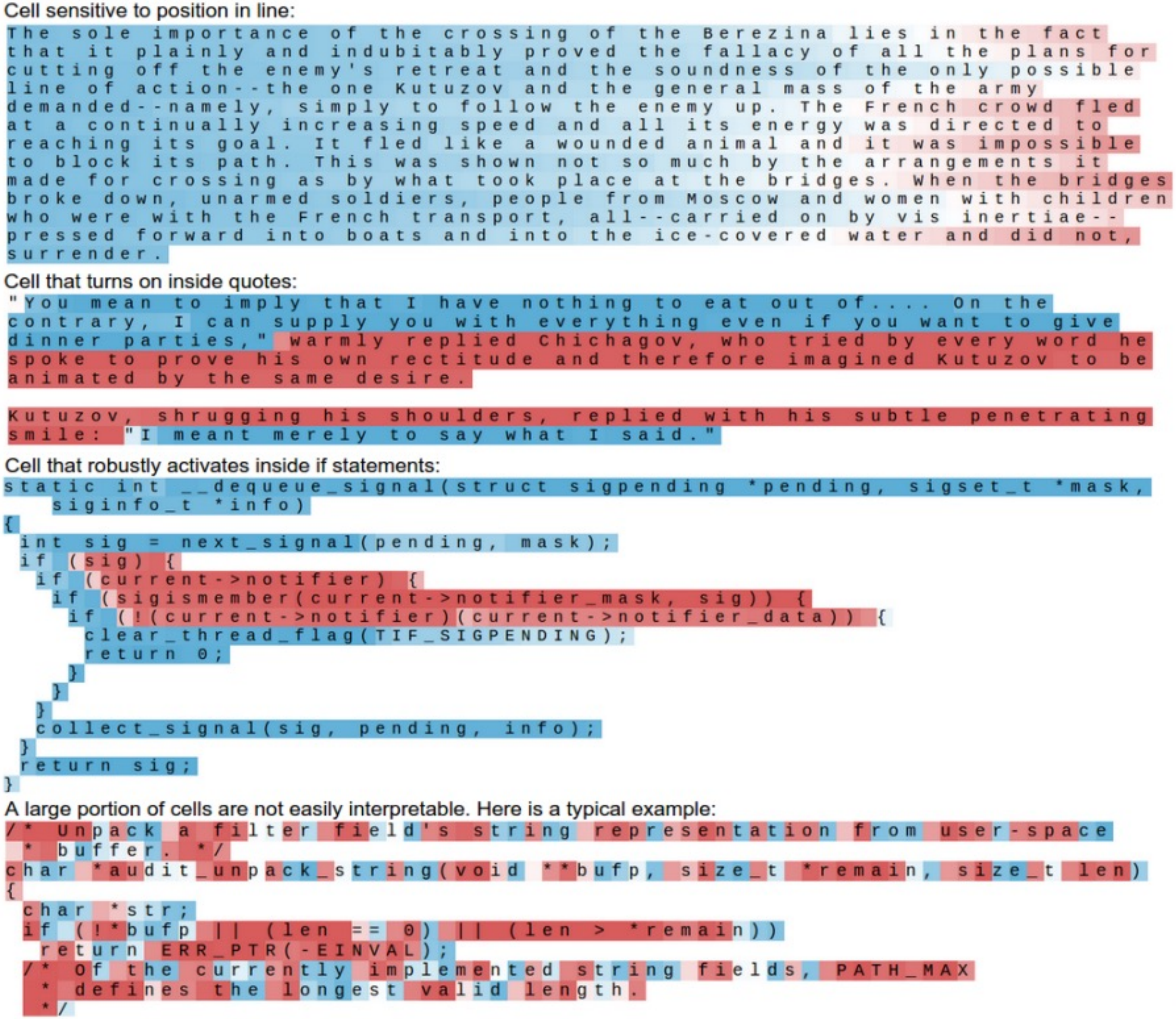
$$h_t = o_t * \tanh\left(C_t\right)$$

Decide what our new hidden representation will be. Based on:
- filtered version of the cell state, and
- weighted version of recurrent, hidden layer

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)



Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."

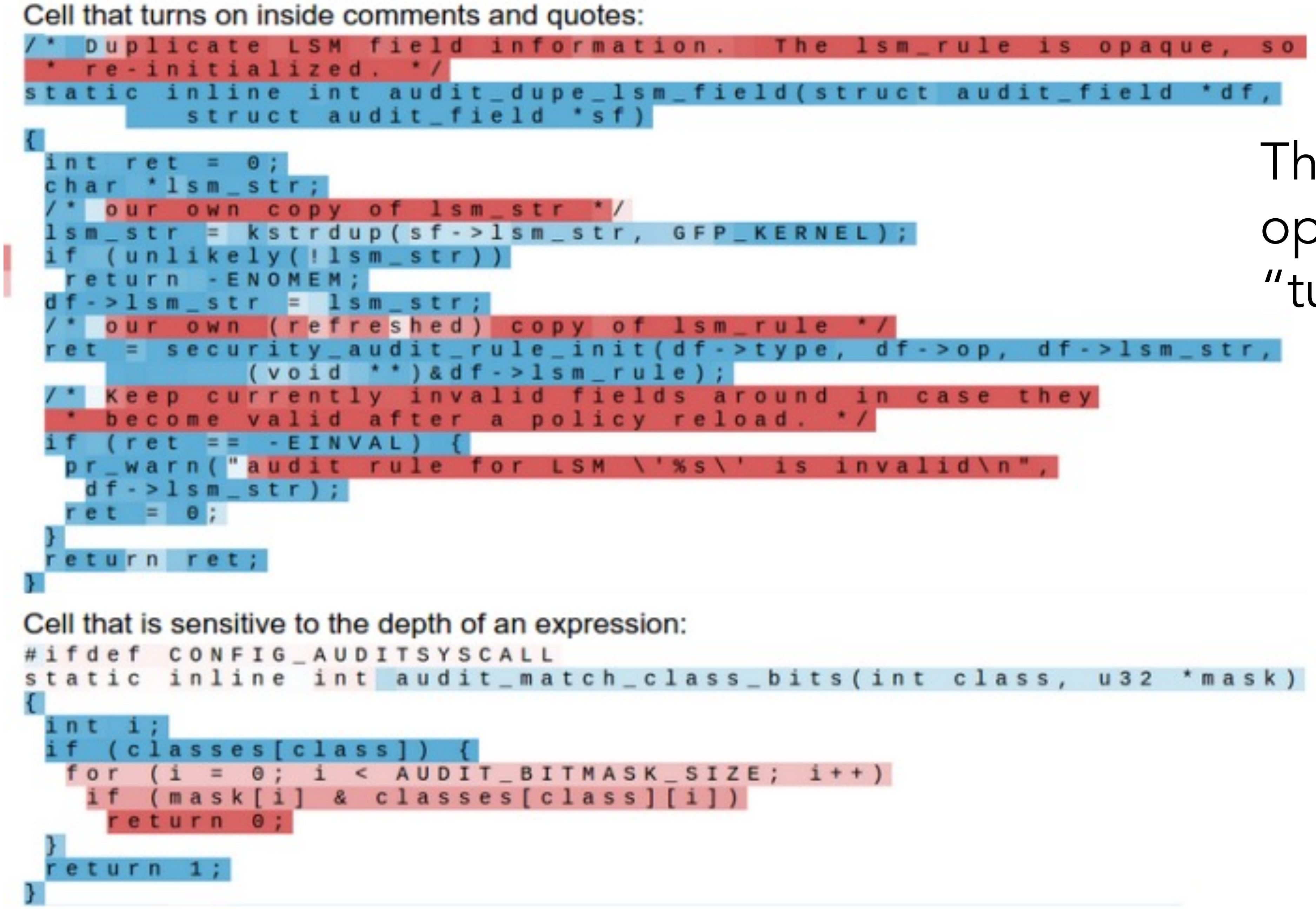
Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

The cell learns an operative time to "turn on".



Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
          struct audit_field *sf)
{
 int ret = 0;
 char *lsm_str;
 /* our own copy of lsm_str */
 lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
 if (unlikely(!lsm_str))
  return -ENOMEM;
 df->lsm_str = lsm_str;
 /* our own (refreshed) copy of lsm_rule */
 ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
              (void **)&df->lsm_rule);
 /* Keep currently invalid fields around in case they
  * become valid after a policy reload. */
 if (ret == -EINVAL) {
  pr_warn("audit rule for LSM \'%s\' is invalid\n",
   df->lsm_str);
  ret = 0;
 }
 return ret;
}
```

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
 int i;
 if (classes[class]) {
  for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
   if (mask[i] & classes[class][i])
    return 0;
 }
 return 1;
}
```

The cell learns an operative time to "turn on".

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

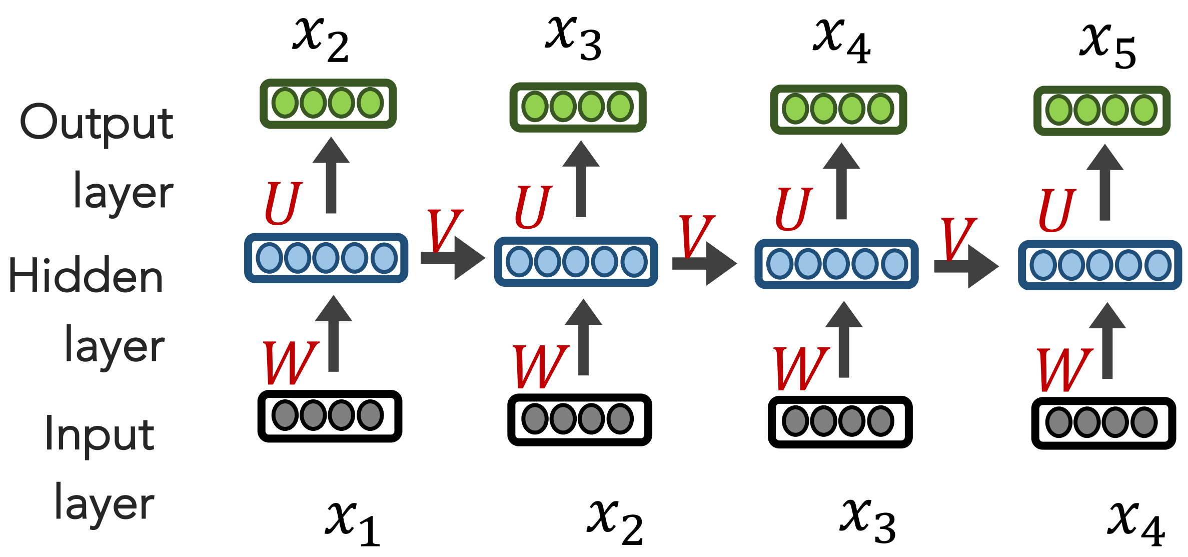
IMPORTANT

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

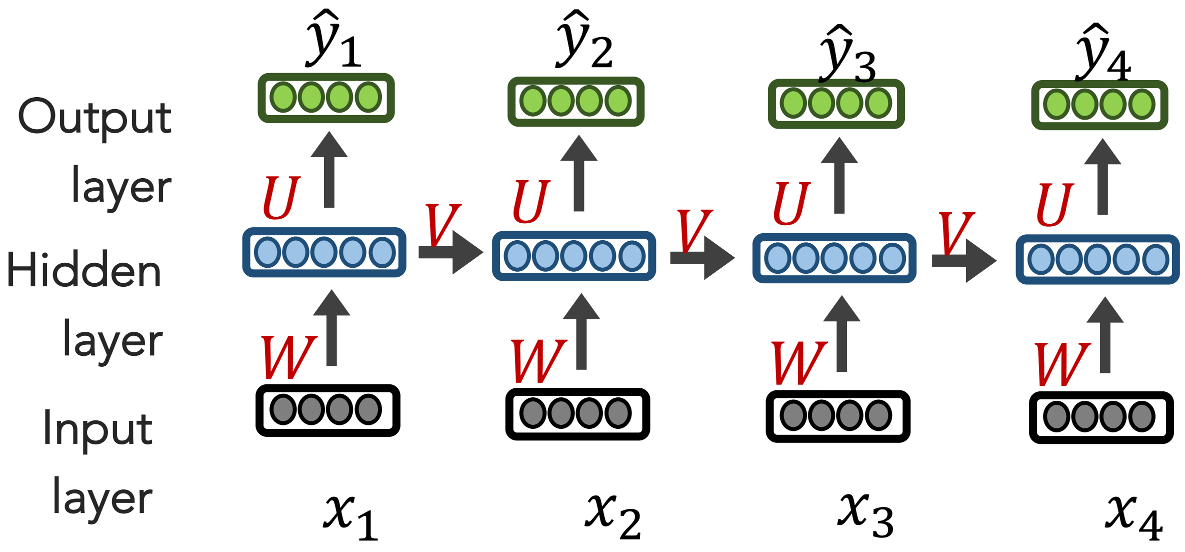
Sequential Modelling

Language Modelling



Auto-regressive

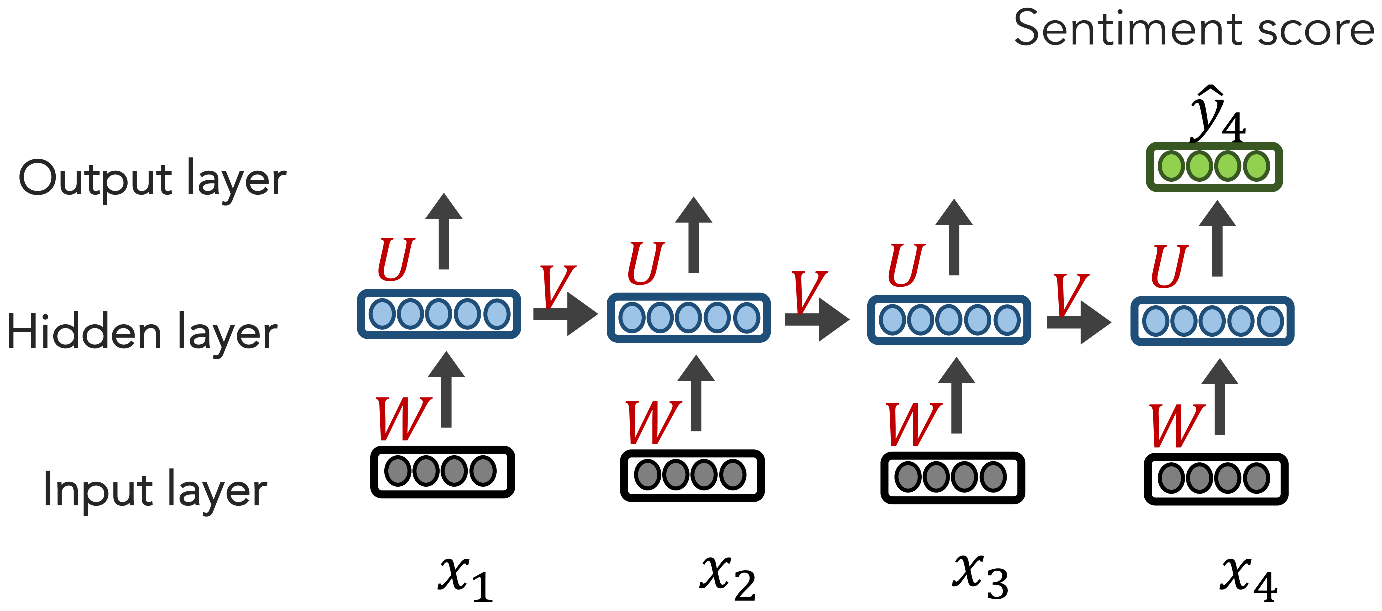
1-to-1 tagging/classification



Non-Auto-regressive

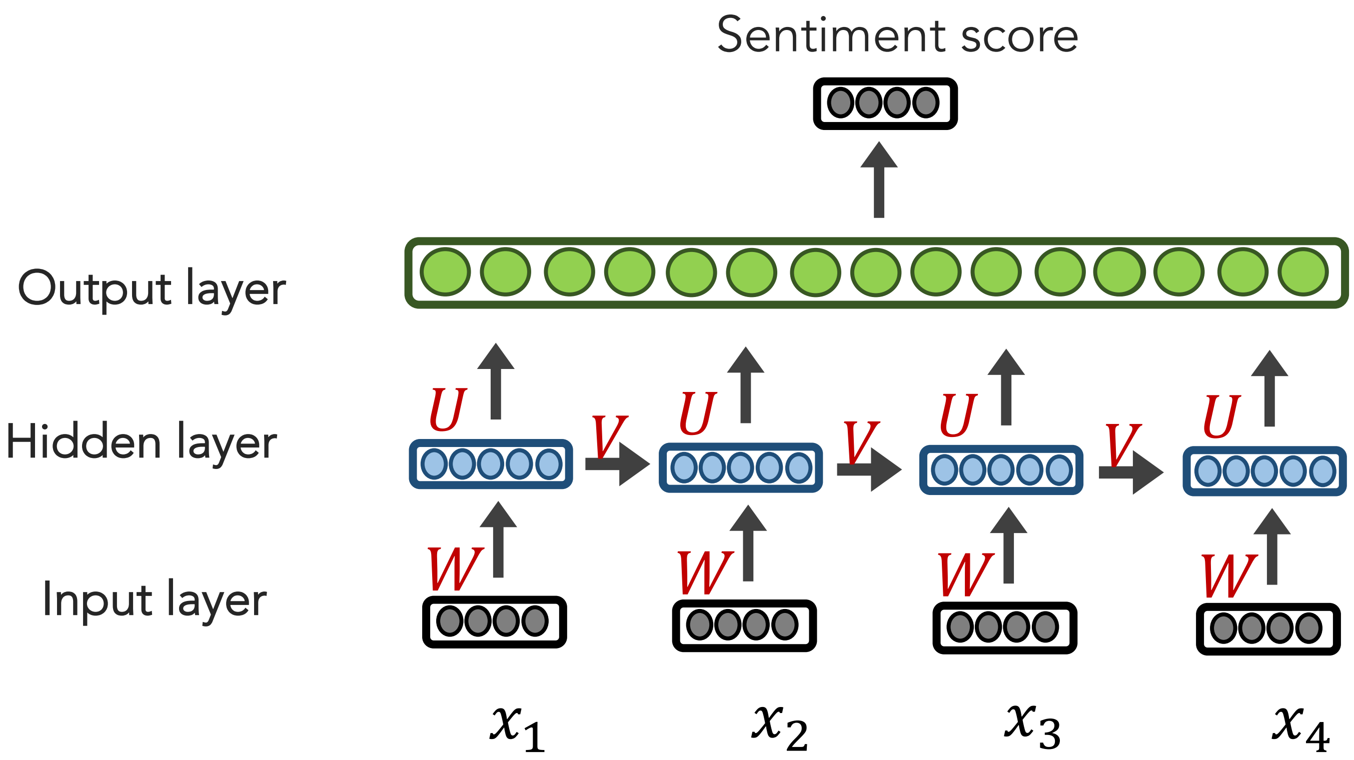
Sequential Modelling

Many-to-1 classification



Sequential Modelling

Many-to-1 classification



This concludes the foundation in sequential representation.

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.)

Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

Outline

- Recurrent Neural Nets (RNNs)
- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo

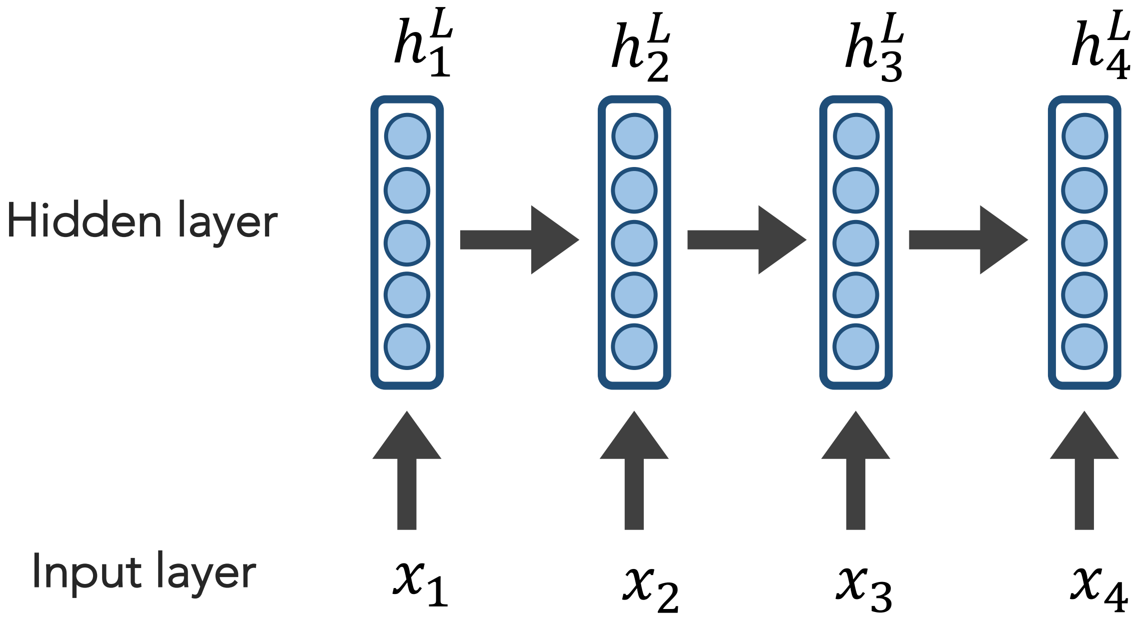
RNN Extensions: Bi-directional LSTMs

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?

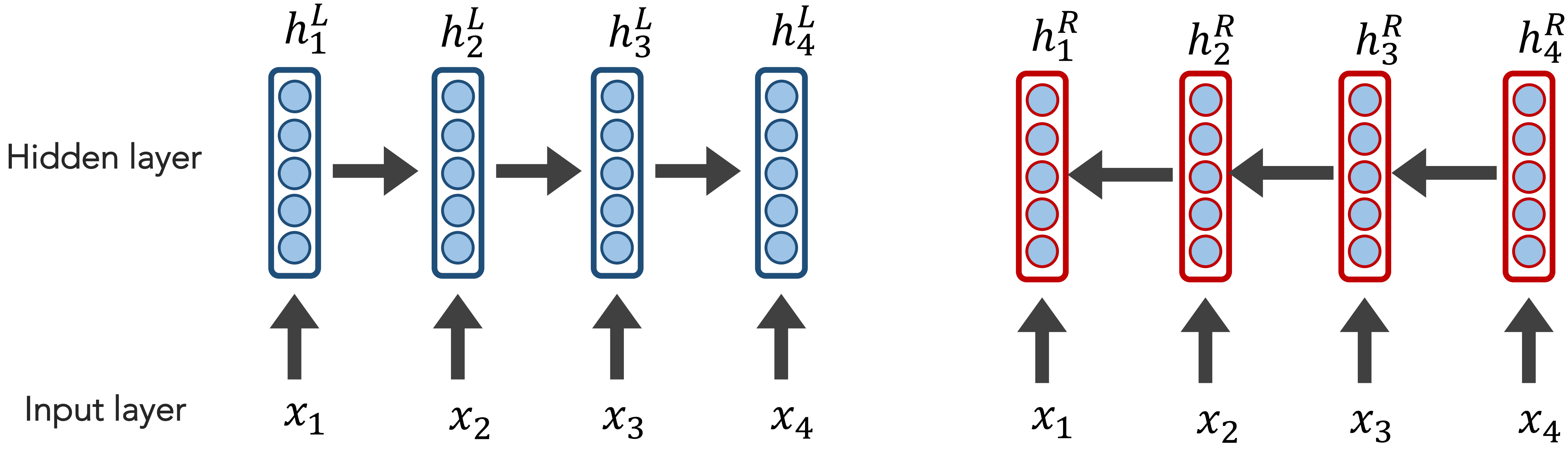
RNN Extensions: Bi-directional LSTMs

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

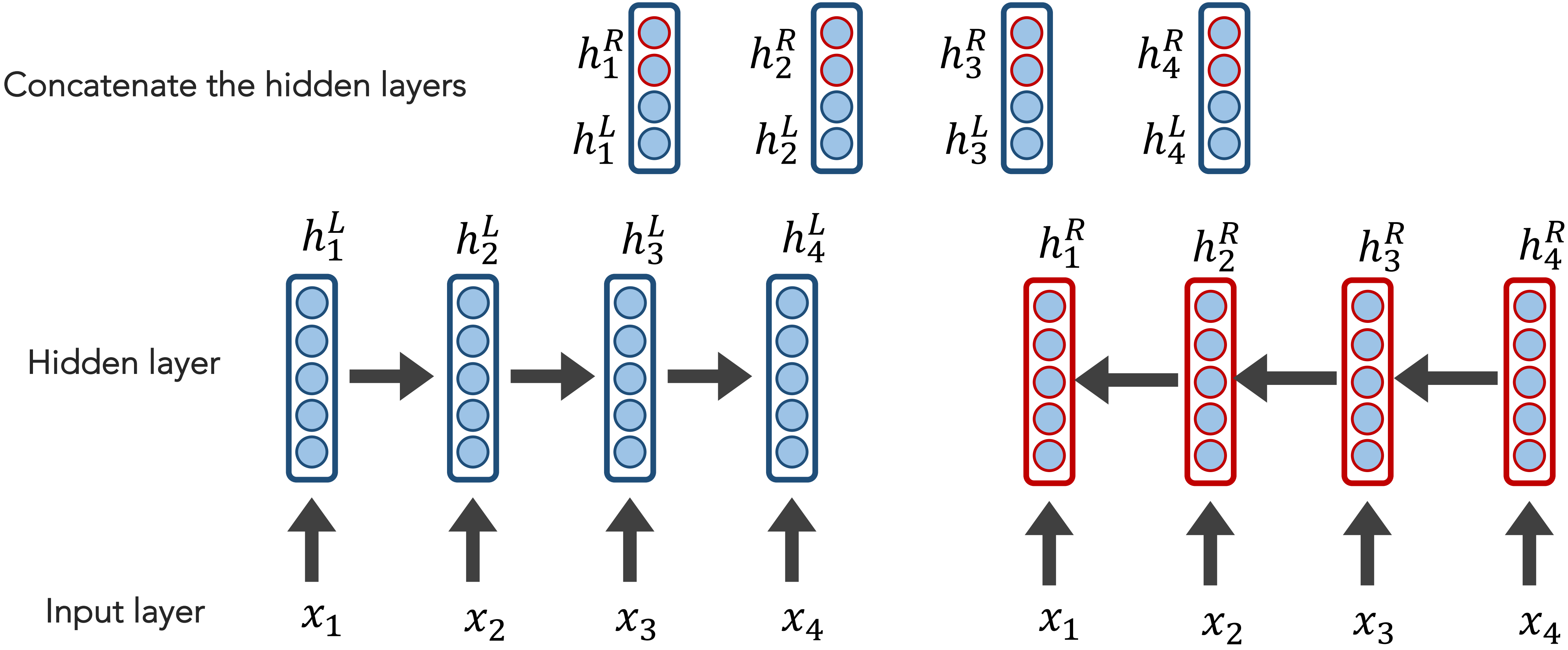


RNN Extensions: Bi-directional LSTMs

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



RNN Extensions: Bi-directional LSTMs

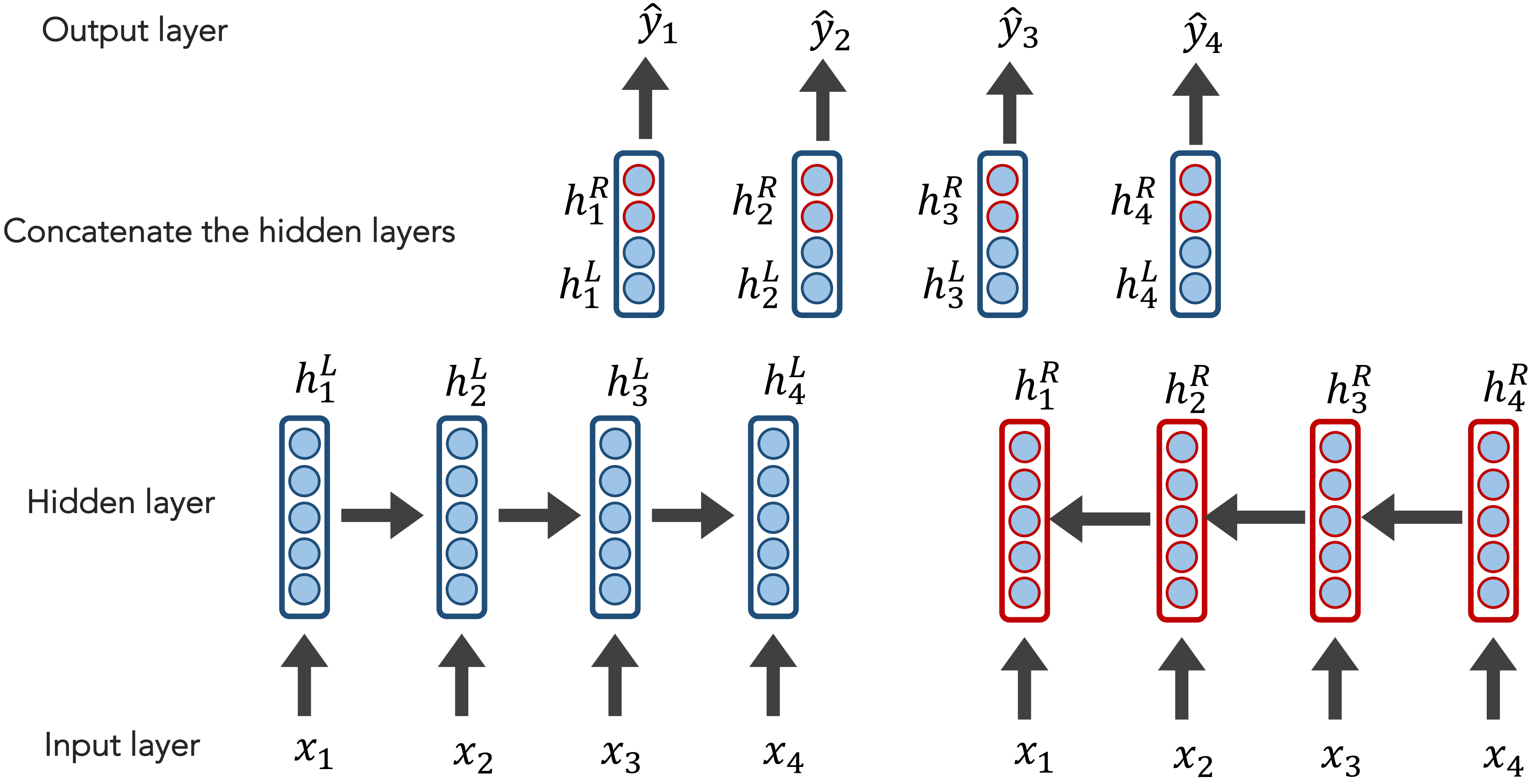


Concatenate the hidden layers

Hidden layer

Input layer

RNN Extensions: Bi-directional LSTMs



RNN Extensions: Bi-directional LSTMs

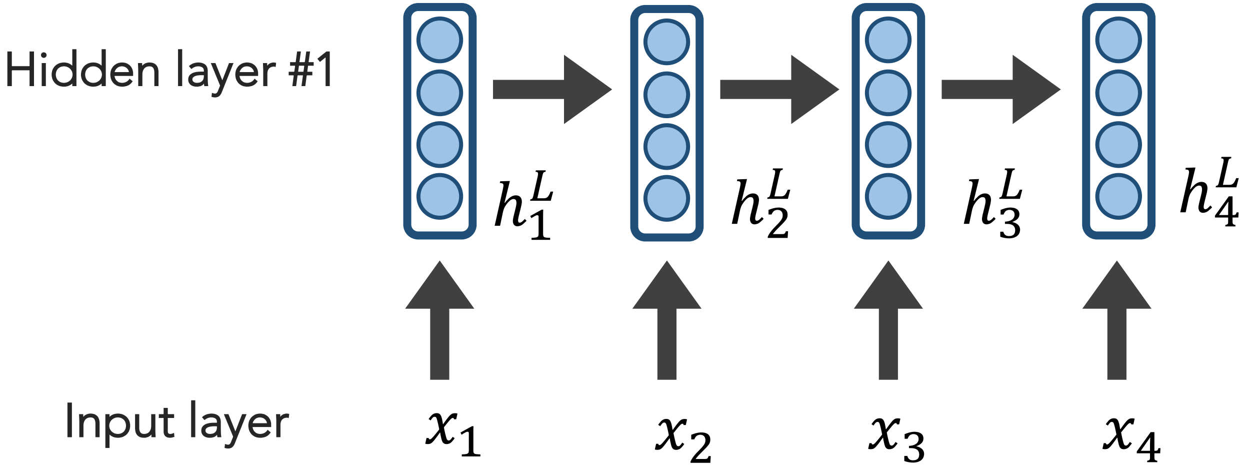
BI-LSTM STRENGTHS?

- 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

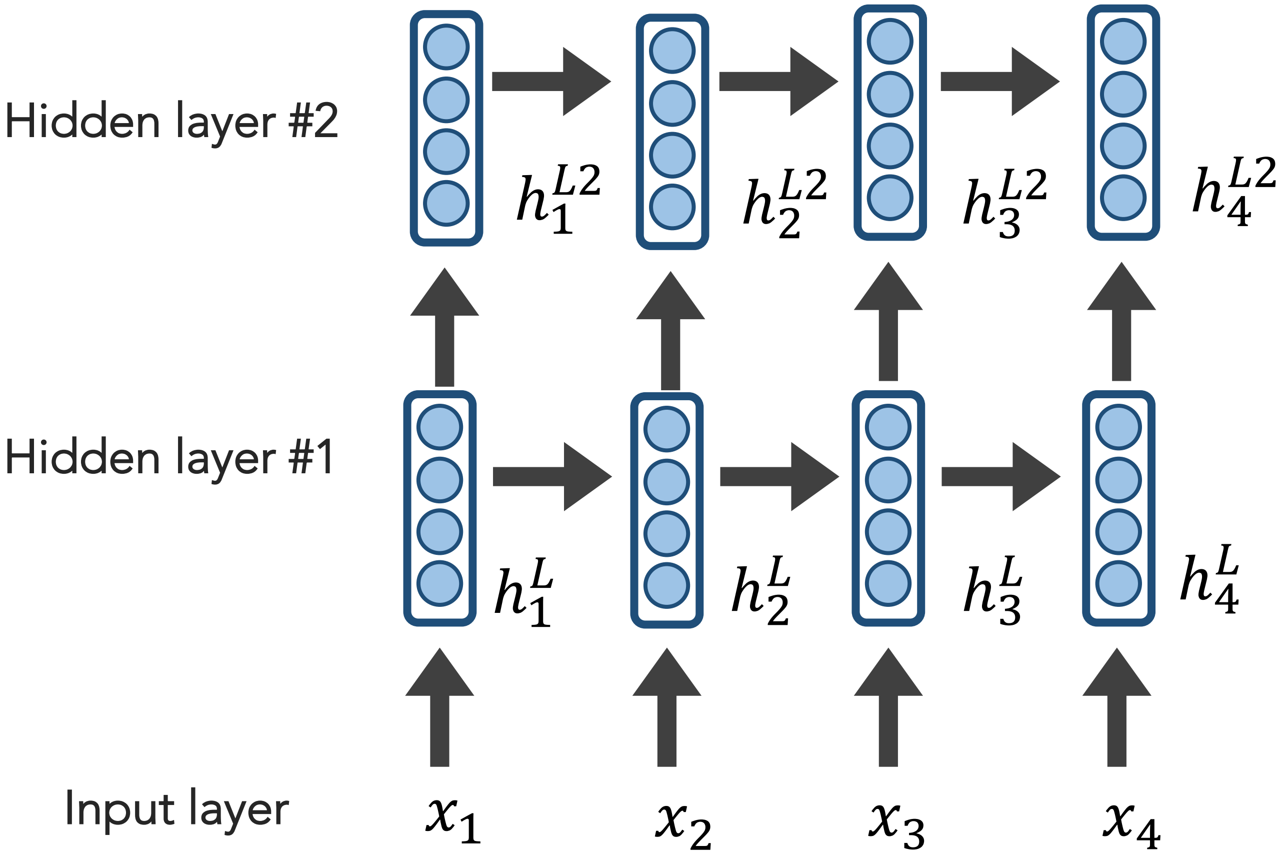
RNN Extensions: Stacked LSTMs



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

Stacking hidden layers provides increased abstractions.

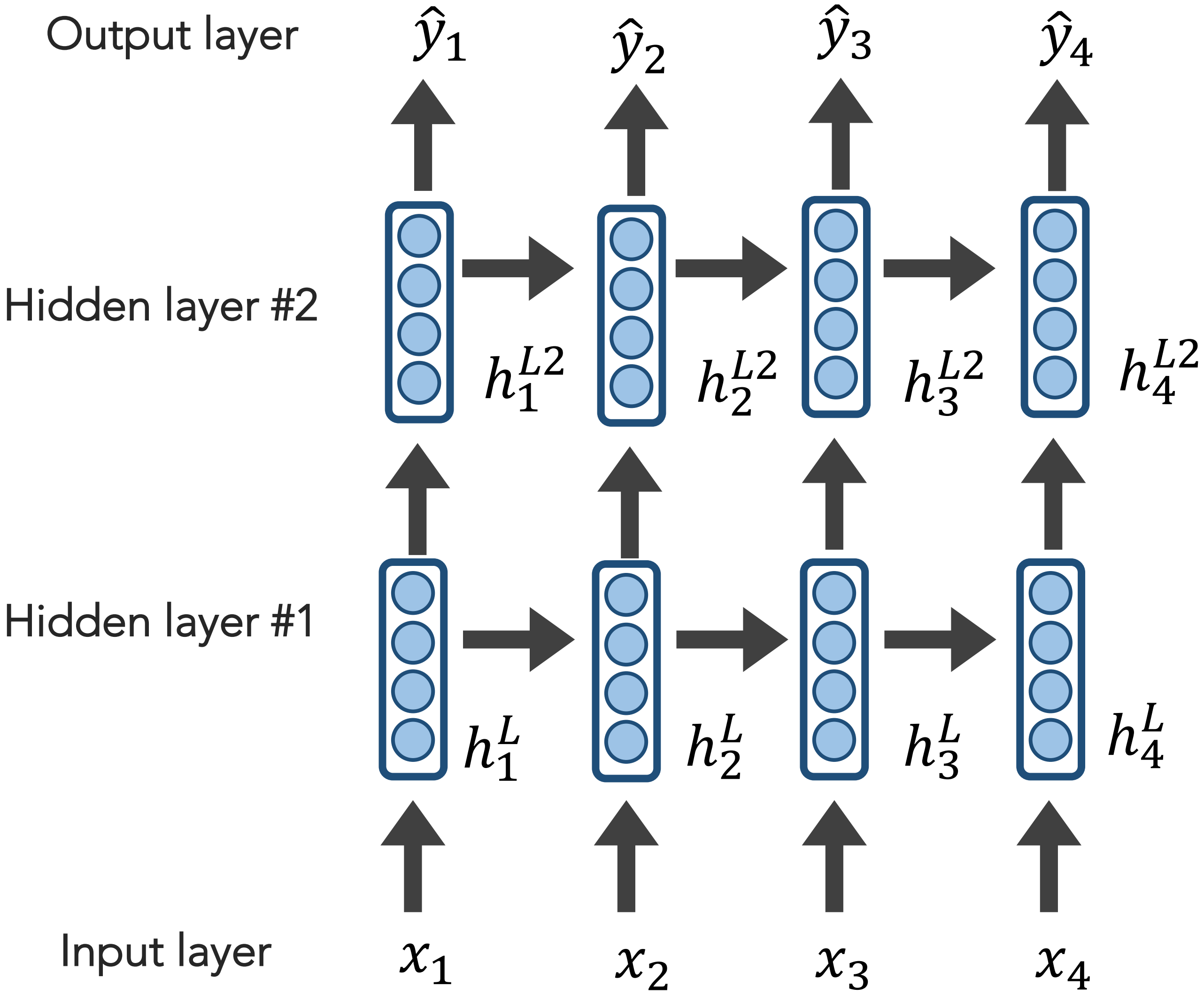
RNN Extensions: Stacked LSTMs



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

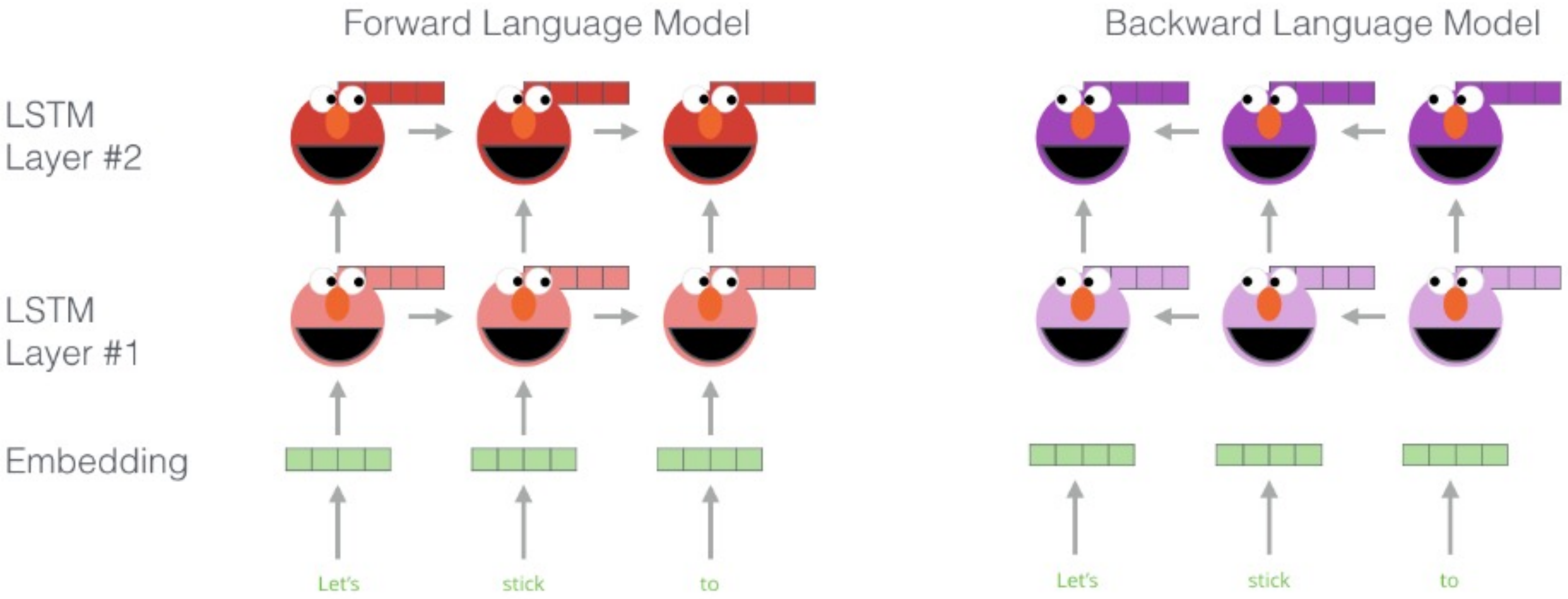
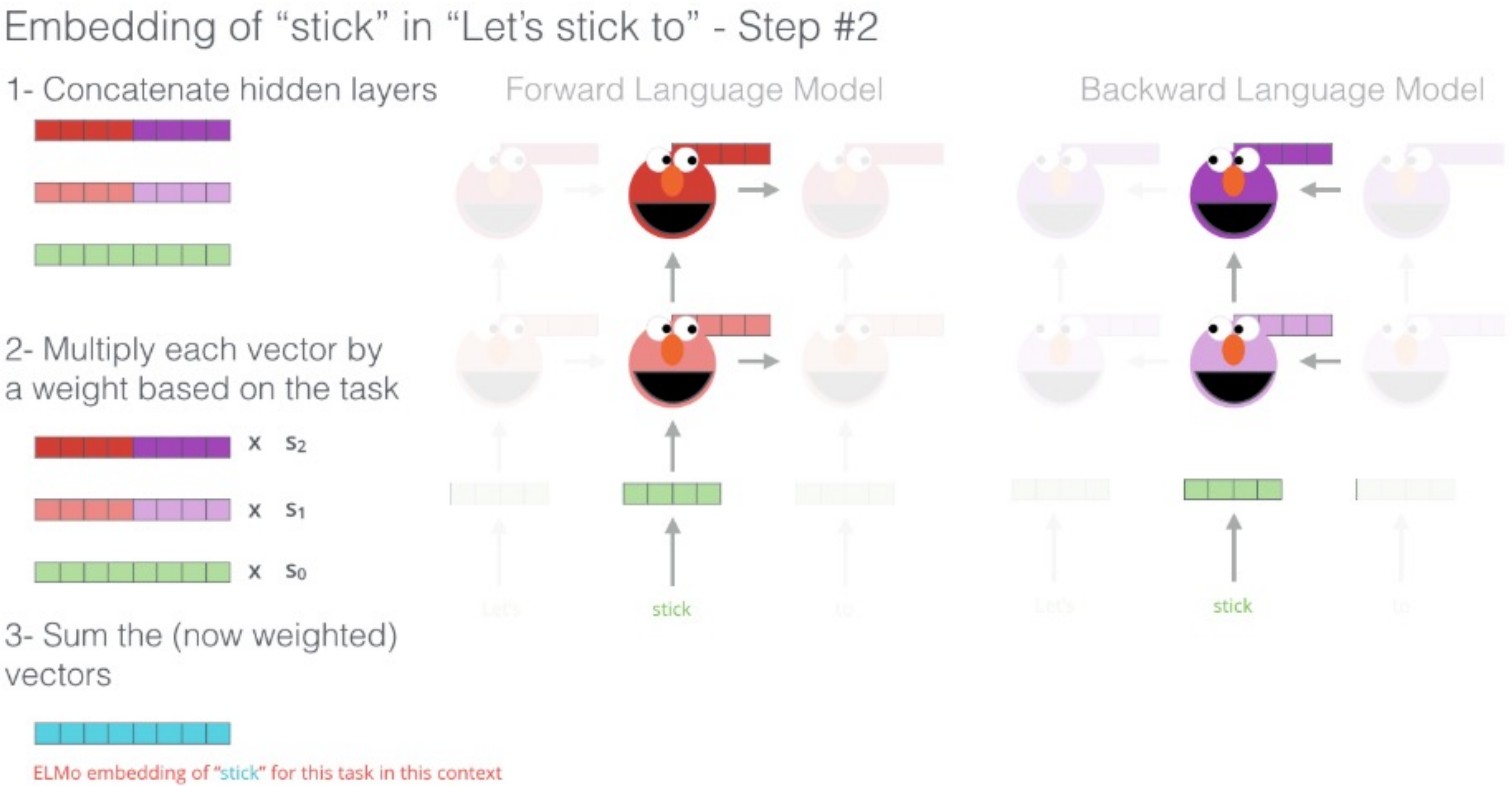


Illustration: http://jalammar.github.io/illustrated-bert/



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

x s_2

x s_1

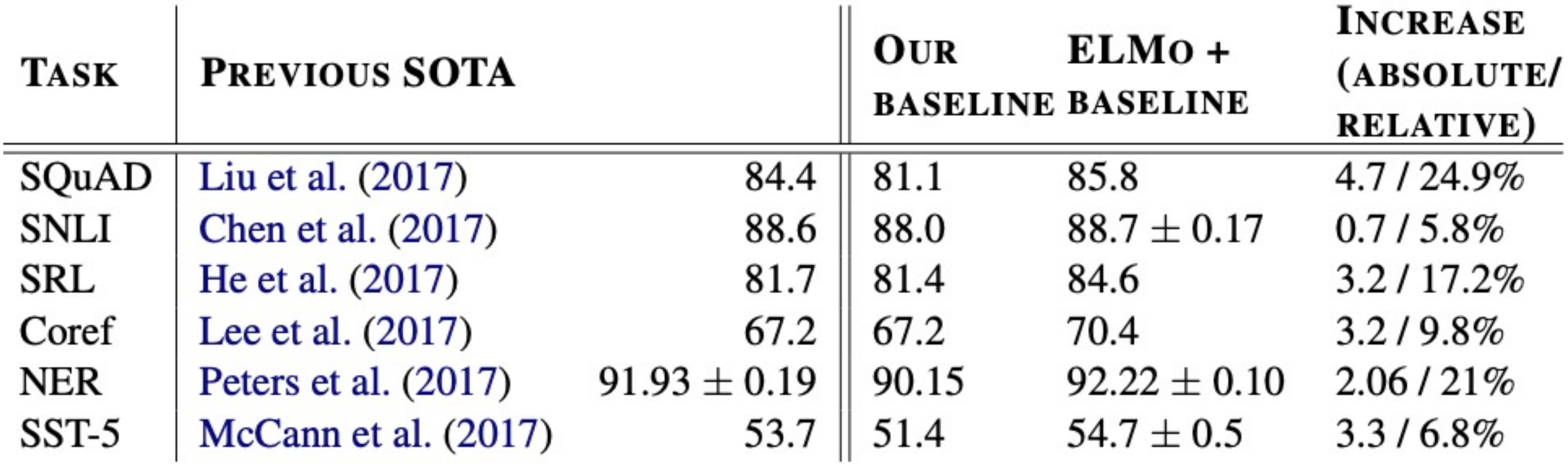
x s_0

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



TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Deep contextualized word representations. Peters et al. NAACL 2018.

ELMo: Stacked Bi-directional LSTMs

- ELMo yielded incredibly good contextualized embeddings, which yielded SOTA 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

SUMMARY

- Distributed Representations can be:
  - Type-based ("word embeddings")
  - Token-based ("contextualized representations/embeddings")
- **Type-based models** include Bengio's 2003 and word2vec 2013
- **Token-based models** include RNNs/LSTMs, which:
  - demonstrated profound results in 2015 onward.
  - it can be used for essentially any NLP task.

BACKUP