Neural Sequence Modeling

Part 2: Recurrent Neural Networks and LSTMs

Aaron Mueller
CAS CS 505: Introduction to Natural Language Processing
Spring 2026
Boston University

Overview of Concepts

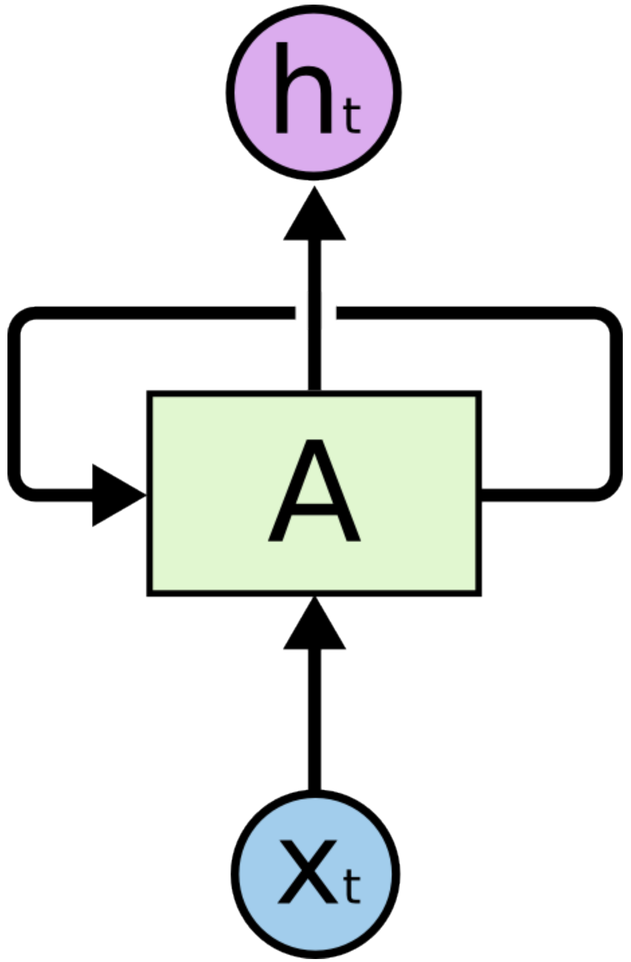
RNNs: A neural architecture for sequential data

Autoregressive: A system that recursively makes predictions given its own outputs

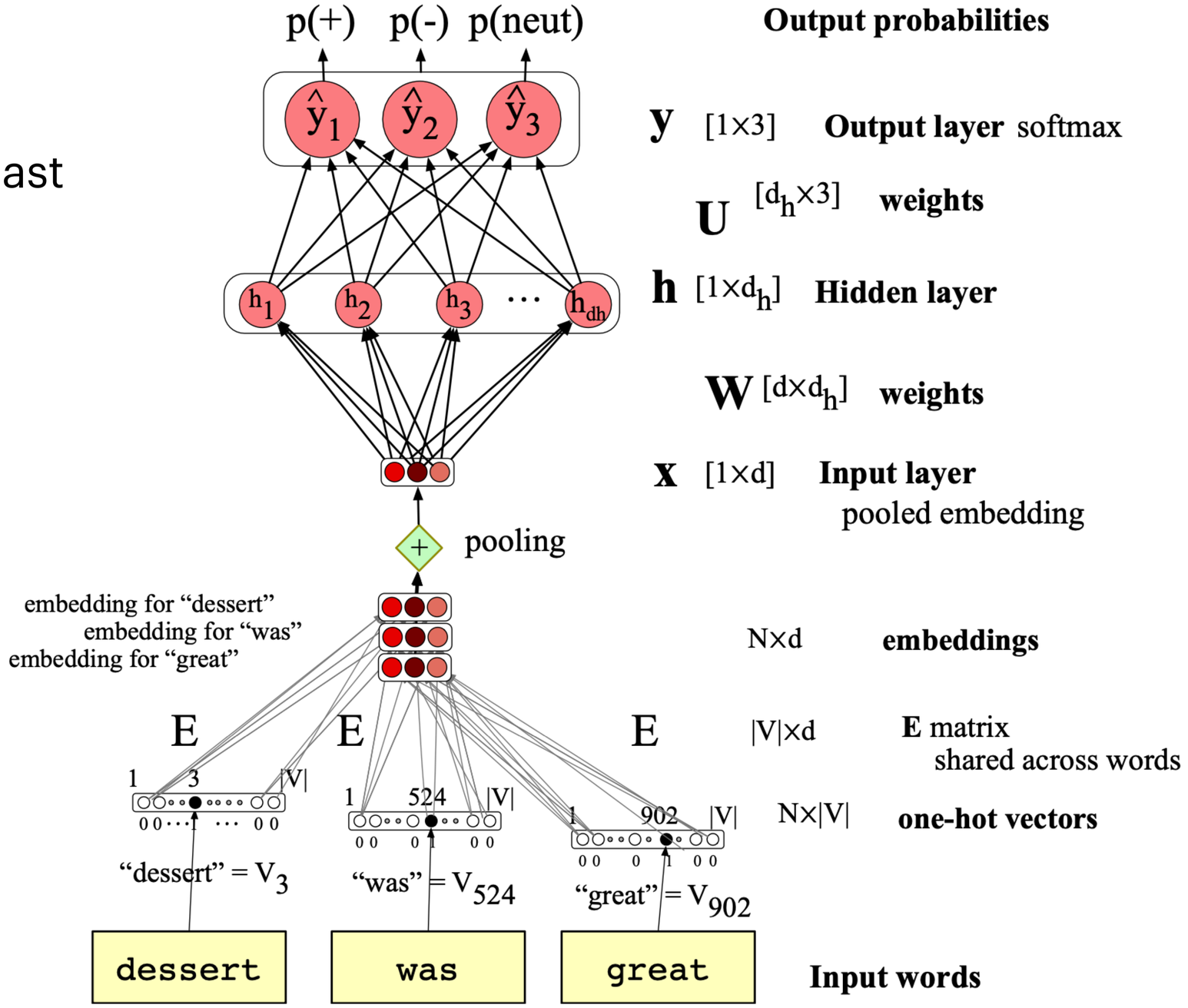
LSTMs: An effective form of RNN that makes extensive use of gates

Gate: A learned mechanism for filtering or transmitting information

Backpropagation through time (BPTT): A method for updating weights in recurrent models

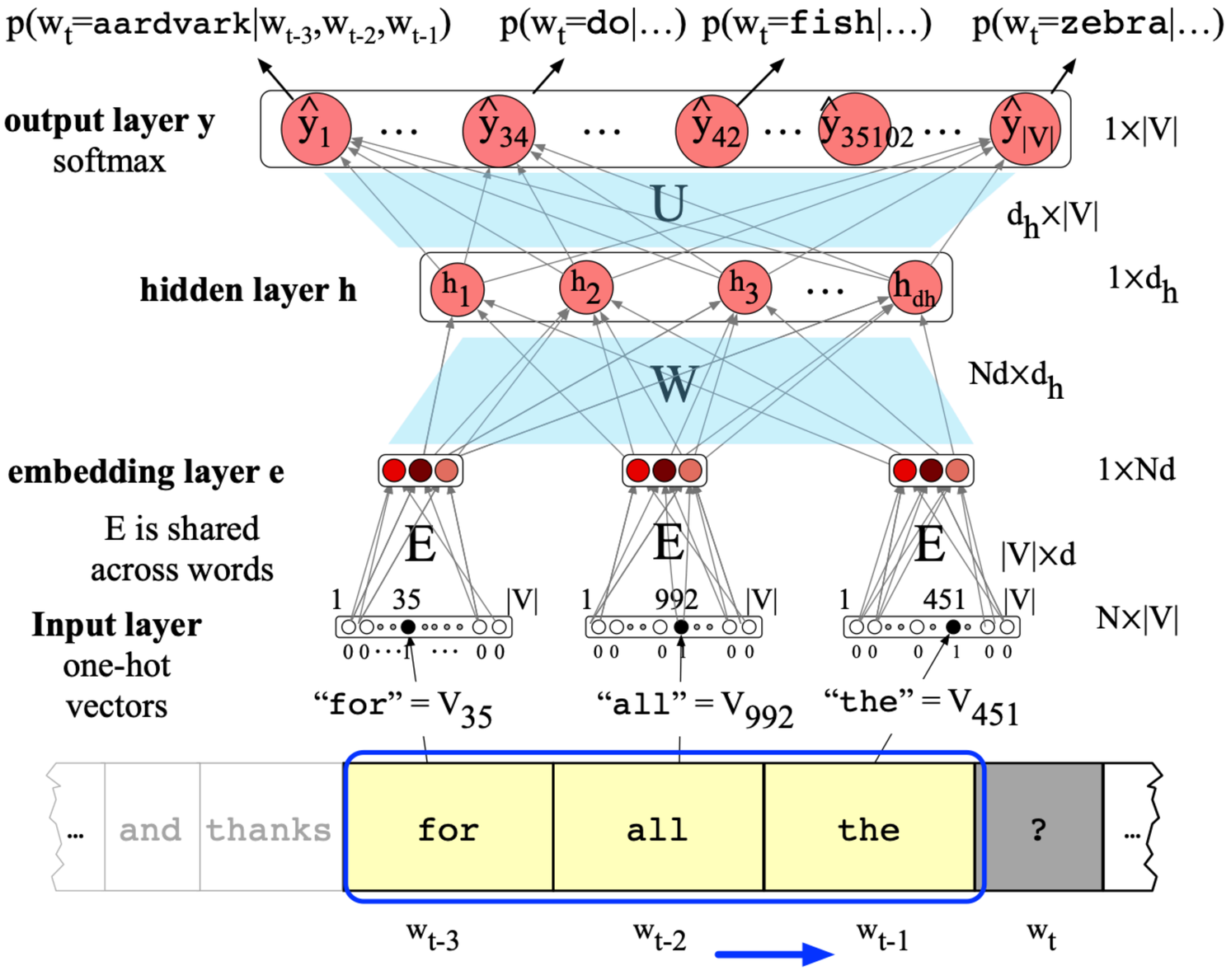


Recall our text classifier from last time:



A *Fixed-window* Neural Language Model

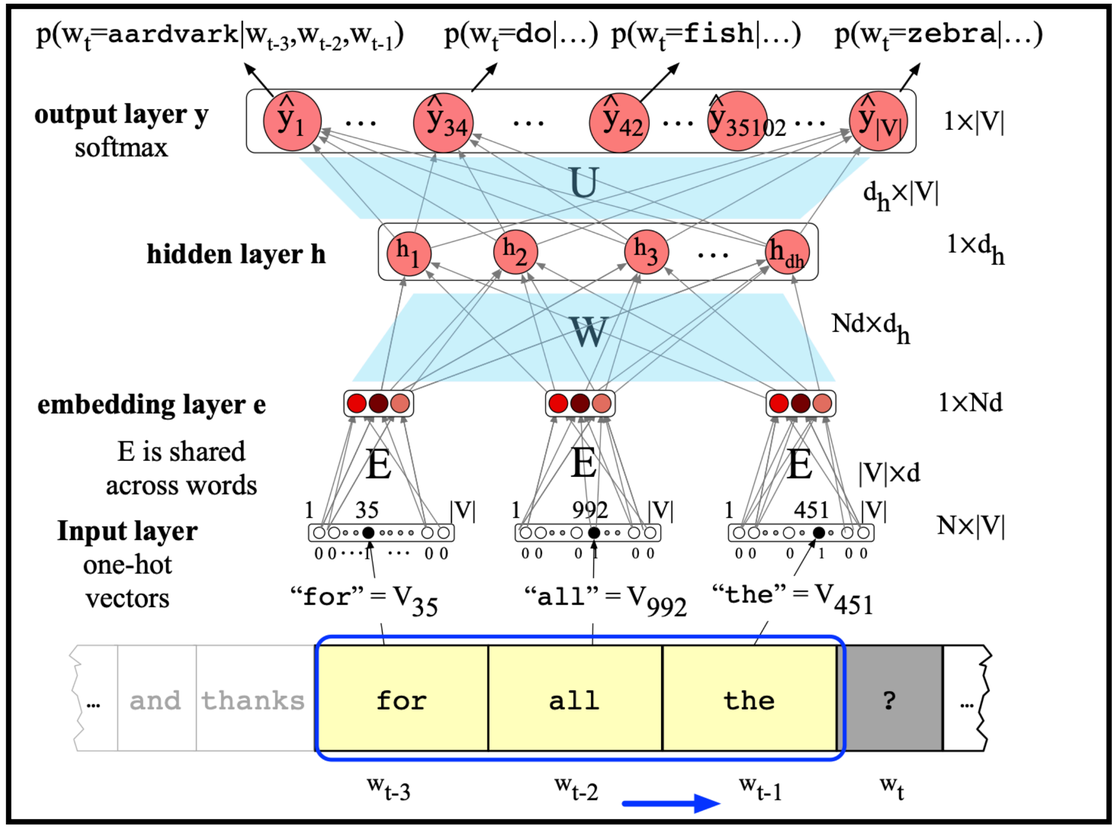
If we replace the classifier output layer with a vocabulary-sized output layer, we now have a language model!



N-gram LMs vs. Fixed-window Neural LMs

- **Improvements** over n-gram LMs:
  - No sparsity problem
  - $O(n)$ memory requirements, rather than $O(\exp(n))$

- Remaining **problems**:
  - Fixed window is too small
  - Enlarging window enlarges the weight matrix
  - Window can never be too large!
  - Each embedding uses different rows of the weights. We **don't share weights** across the window positions.



Long-distance dependencies are common.

The **keys** to the **cabinet** **are** on the table.

Yesterday, **Kathy went to the store**. She bought some groceries, paper towels, … Today, Kathy went to school.

Q: Where did Kathy go yesterday?

Translate this to French: The **black cat** near the mat sat on the hat.
Le **chat noir** près du tapis s’est assis sur le chapeau.

How can we predict more than one token?

1. We need to add some sort of “memory” to handle long-range dependencies.

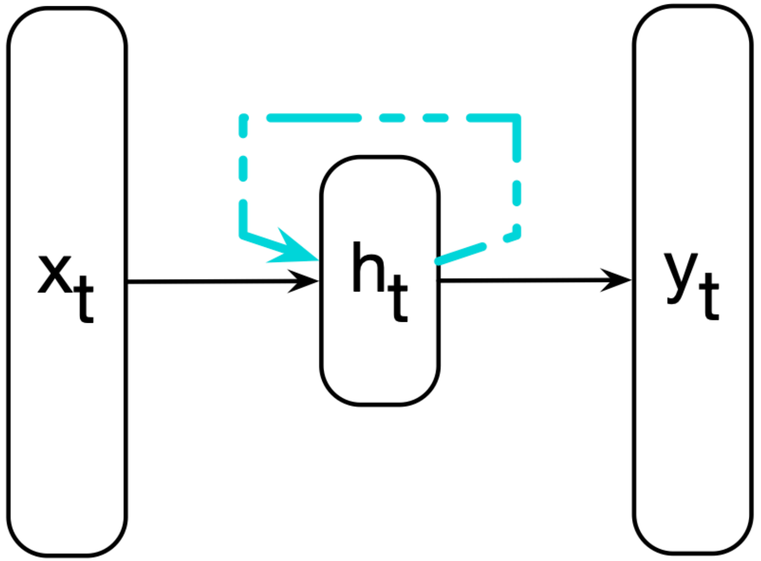
   *Let’s use a separate vector (**context vector**) to keep track of context.*

2. We also need a model that’s capable of tracking information across potentially very long documents.

   *Let’s use a **recurrent** architecture.*

Recurrent Neural Networks (RNNs)

Elman, 1990

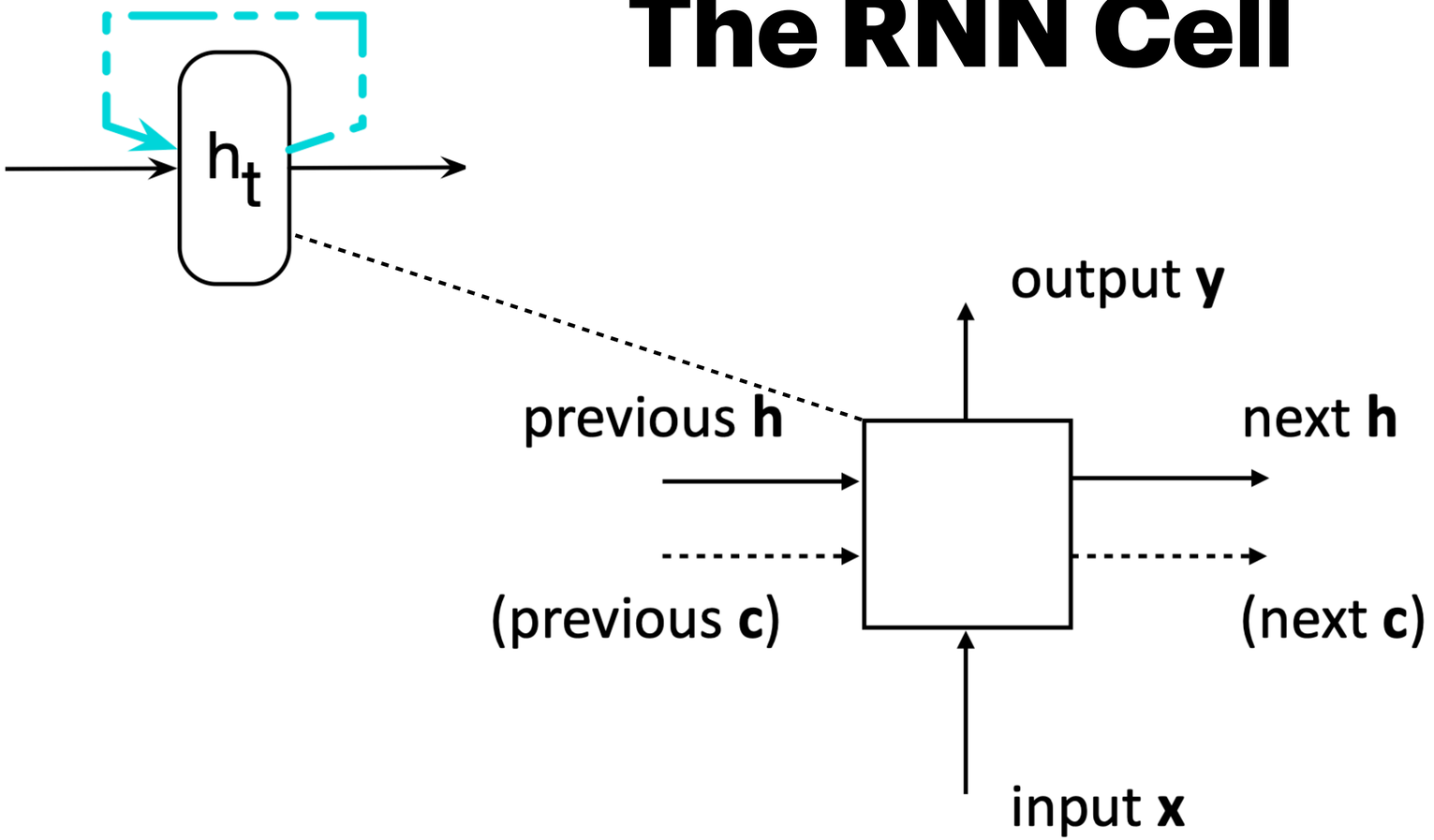


The input layer x_t feeds into a hidden layer $\mathbf{h}_t$ (like before).

However, the hidden layer has a **recurrent** connection that depends on the current token as well as the hidden layer from the previous timestep $\mathbf{h}_{t-1}$.

This hidden state acts as a sort of memory or context vector.

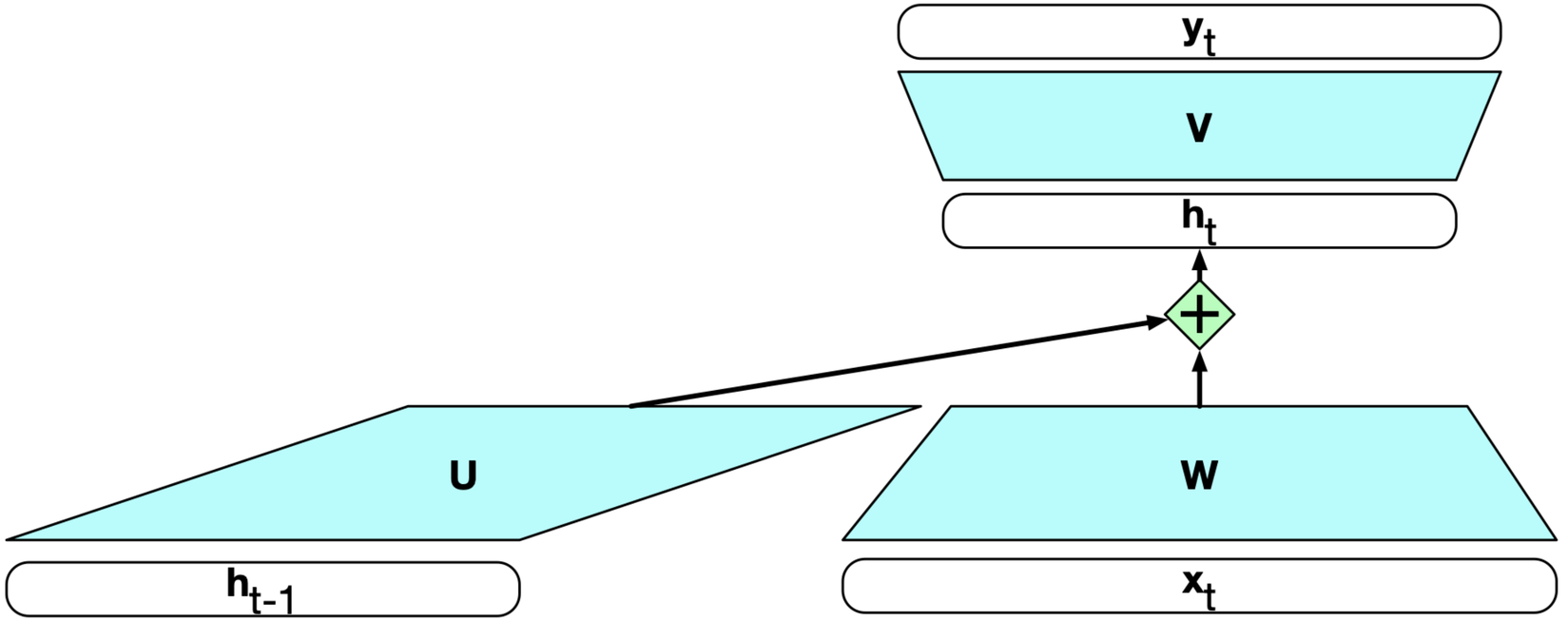
The RNN Cell



Each recurrence takes the previous hidden state $\mathbf{h}_{t-1}$, previous context vector $\mathbf{c}_{t-1}$, and current input x_t.

It outputs the next hidden state $\mathbf{h}_t$, context vector $\mathbf{c}_t$, and an output token $\hat{y}$.

Recurrent Neural Networks (RNNs)



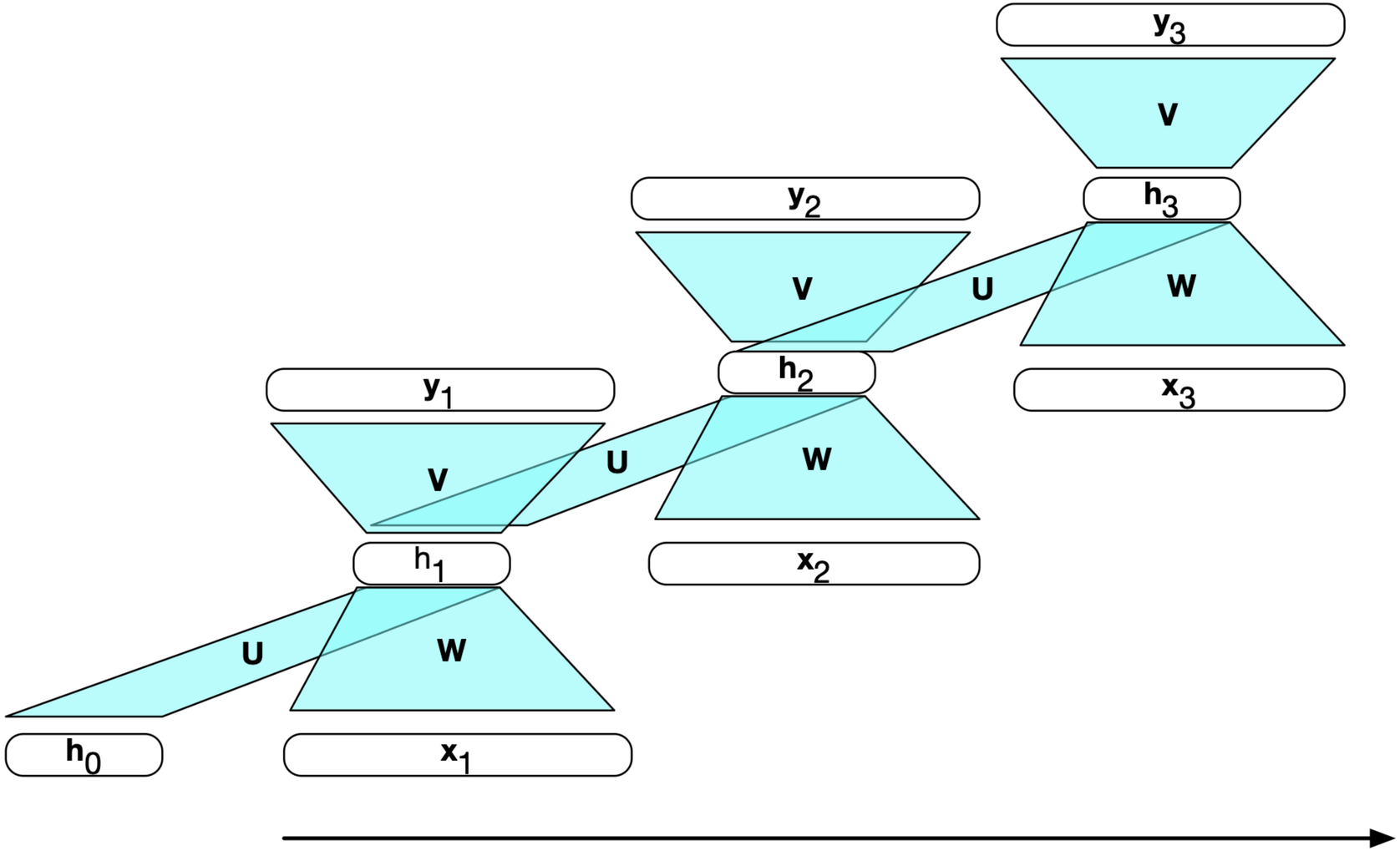
Each token's hidden representation $\mathbf{h}_t$ is a function of the current token's embedding $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$.

Note: the weight matrices U, V, W are *the same* for each timestep t!

$$\mathbf{h}_t = g(U\mathbf{h}_{t-1} + W\mathbf{x}_t)$$

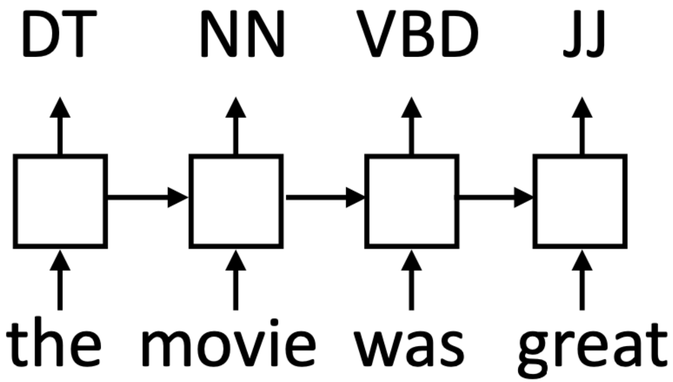
$$\mathbf{y}_t = f(V\mathbf{h}_t)$$

Unrolling an RNN

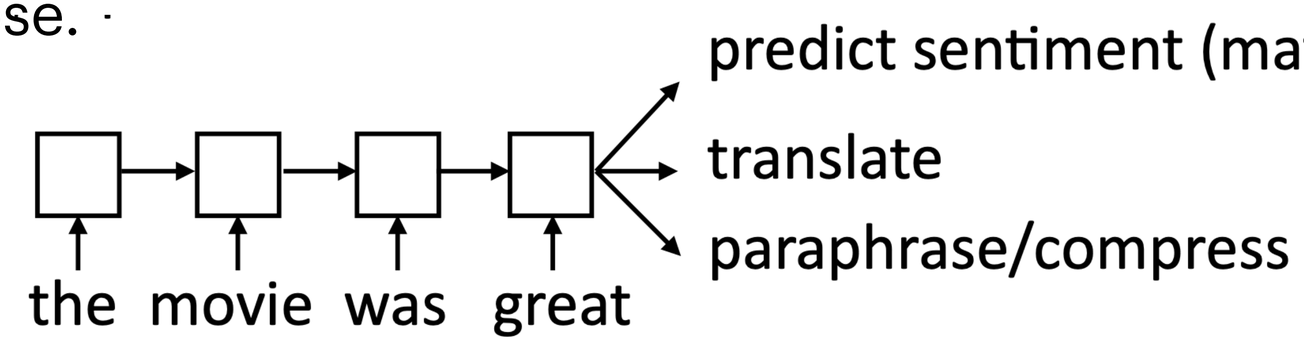


Uses of RNNs

Sequence labeling: predict a label for every element in a sequence.



Encode: encode the entire sequence into a representation, and use that for some downstream purpose.



[Figures from Greg Durrett.]

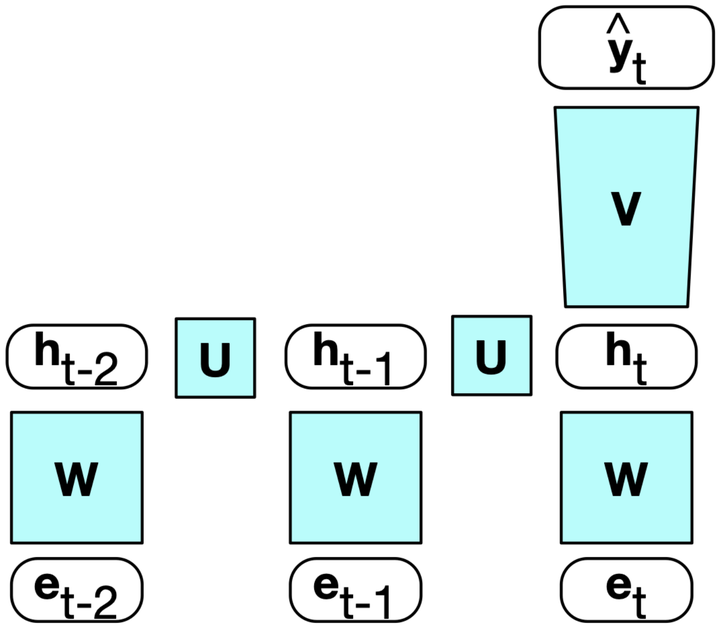
RNN Language Models

- Recall the chain rule for computing sequence probabilities:

$$p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} p(w_i \mid w_{<i})$$

- For n-grams, we had to use a limited context window of $n - 1$.
- For RNNs, the hidden state could potentially include information from *any* timestep; thus, there is technically no limit to how long our context can be!

RNN Language Models



$$\mathbf{e}_t = E\mathbf{x}_t$$

$$\mathbf{h}_t = g(U\mathbf{h}_{t-1} + W\mathbf{x}_t)$$

$$\hat{\mathbf{y}}_t = \text{softmax}(V\mathbf{h}_t)$$

The hidden state has size d; this is a hyperparameter.

Here, V (the "unembedding" matrix) is learned independently of E. Optionally, you can **tie the embeddings** by making V the transpose of E. This allows you to learn fewer parameters.

RNN Language Models

The probability of a token given prior context is:

Probability distribution over tokens

$$\hat{\mathbf{y}}_t[k]$$

Index of the token in the probability distribution

The probability of a sequence is the product of its tokens' conditional probabilities:

$$p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} \hat{\mathbf{y}}_i[w_i]$$

As per usual, we actually work in log space:

$$p(w_1, w_2, \ldots, w_n) = \sum_{i=1}^{n} \log \hat{\mathbf{y}}_i[w_i]$$

Training an RNN

- Much like in classification, we can use a cross-entropy loss to train an RNN:

$$L_{\text{CE}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

- Again, we assume that $p(\mathbf{y}_t[w])$ is always 1, so this simplifies to:

$$L_{\text{CE}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

- In other words, the cross-entropy loss at a given timestep t is the negative log-probability of the correct next token.

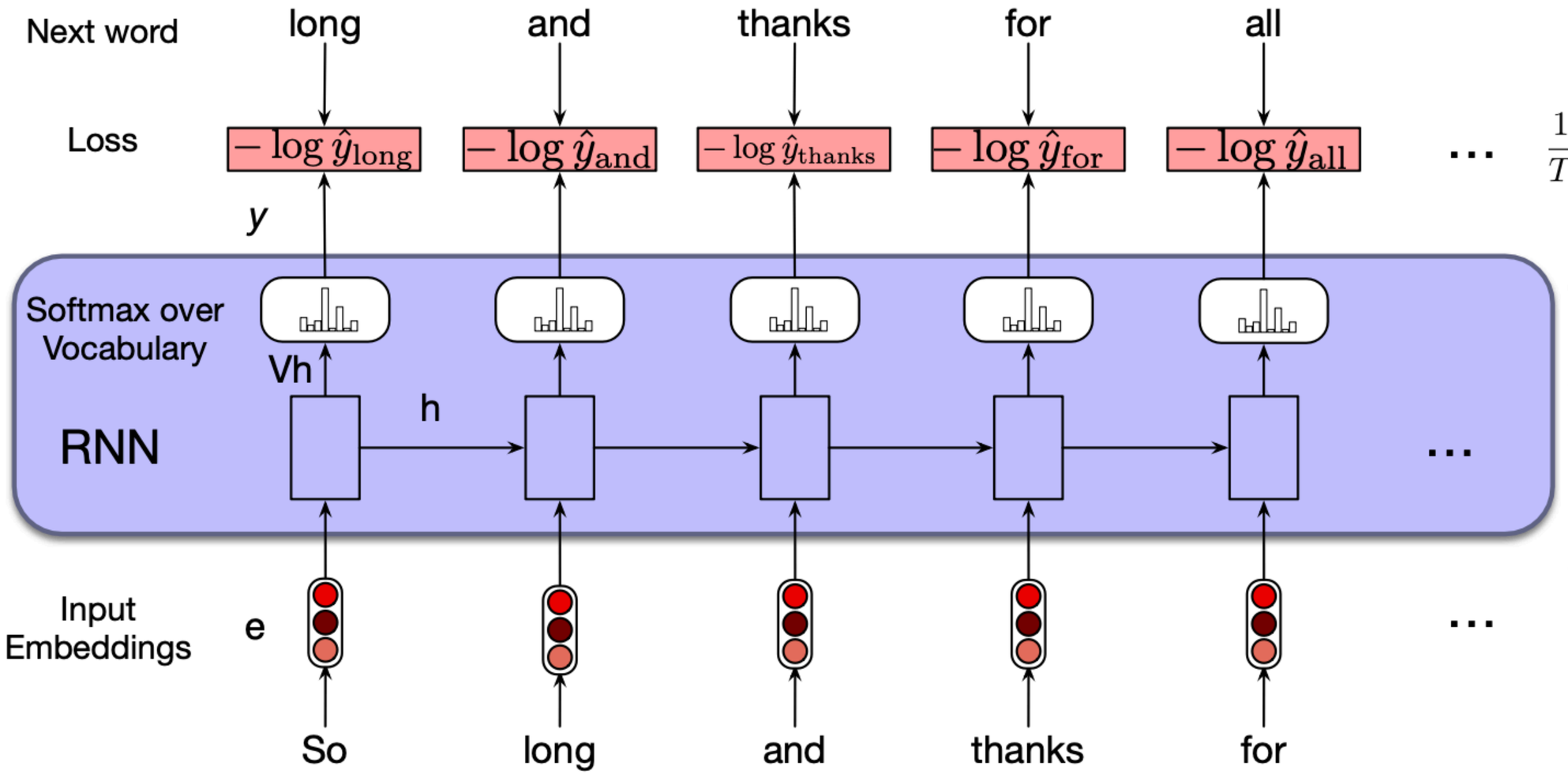
Training an RNN

1. Get a dataset D consisting of documents.
   - Each document consists of sequences of words $(w_1, \ldots, w_T)$.
2. Feed into RNN, compute probability distribution over tokens $\hat{y}_t$ *for each timestep* t.
3. Compute the cross entropy loss at every position.
4. Average the loss across timesteps:

$$L_{\text{CE}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = \frac{1}{T} \sum_{t=1}^{T} - \log \hat{\mathbf{y}}_t[w_{t+1}]$$

The Forward Pass

Do forward inference for each timestep, accumulating the loss at each step.



Teacher forcing: Regardless of what the model predicted, give it the correct prior context when predicting the next token.

Backpropagation Through Time (BPTT)

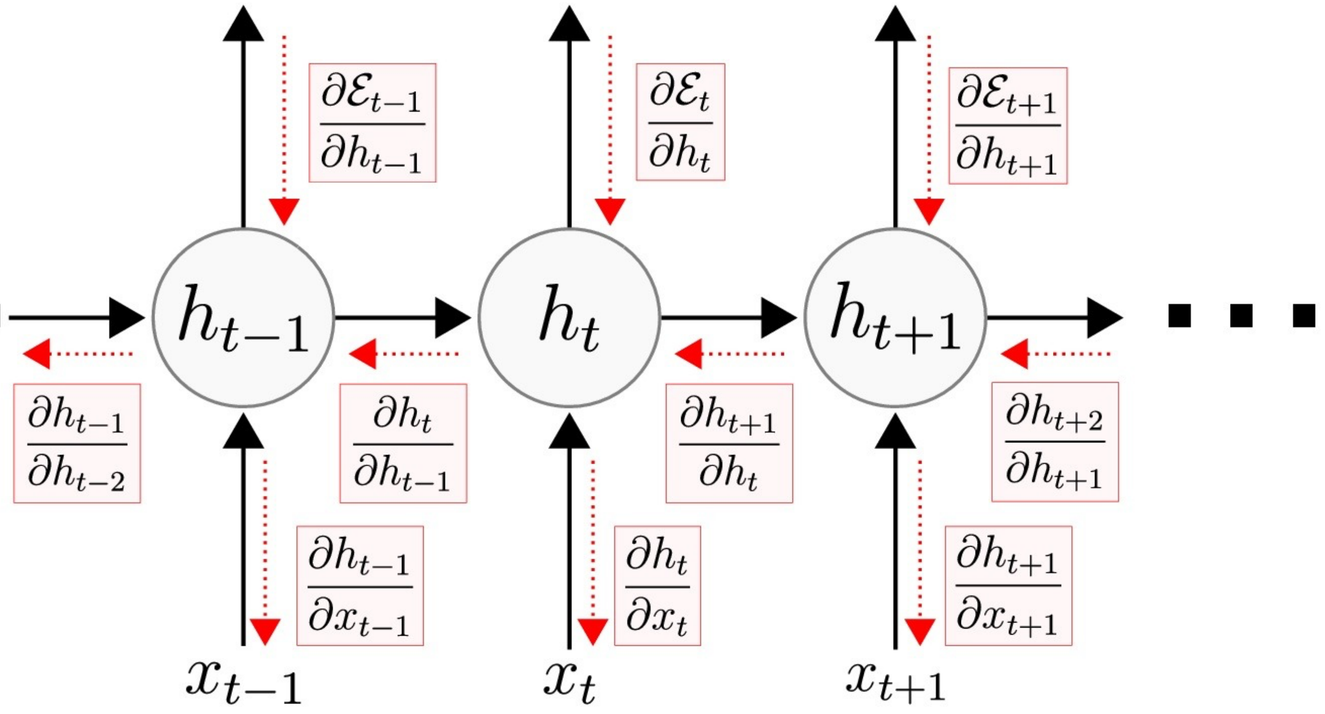
- Backpropagation is straightforward when we only have one prediction to worry about.

- However, RNNs make a bunch of predictions, each of which depends on the previous timestep. How does backpropagation work in this case?
- Basically, we must backpropagate starting from the final prediction, and work our way backwards to the first timestep. This is called **backpropagation through time (BPTT).**
    - This means there is now an $O(n)$ factor in backprop's time and space complexity.

Backpropagation Through Time (BPTT)

1. Do forward inference for each timestep, accumulating the loss at each step.

2. Process the sequence in reverse, computing required gradients step-by-step.
   - If the sequence is short enough, we can do this all in parallel!
   - If not, we can break it into chunks and use each chunk as a separate training example.



RNNs vs. N-grams

Perplexity

Word Error Rate

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

Recurrent neural networks *significantly* outperform 5-gram LMs with Kneser-Ney smoothing!

The Unreasonable Effectiveness of RNNs

Train an RNN on ≈100MB of Wikipedia data:

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```

Model learns to open and close parentheses correctly!

(fake Yahoo link)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

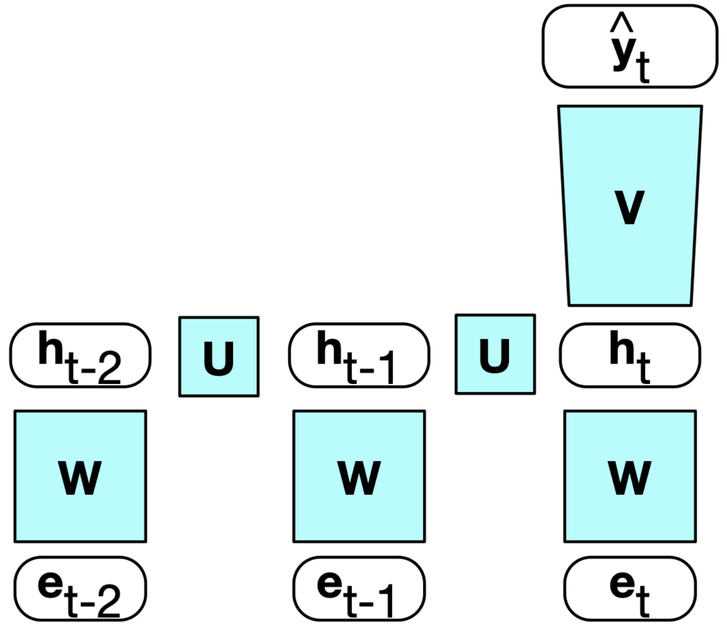
The Unreasonable Effectiveness of RNNs

The same RNN learns to produce correctly formatted XML.

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT [[Christianity]]</text>
  </revision>
</page>
```

What Can You Fit in a Word Vector?

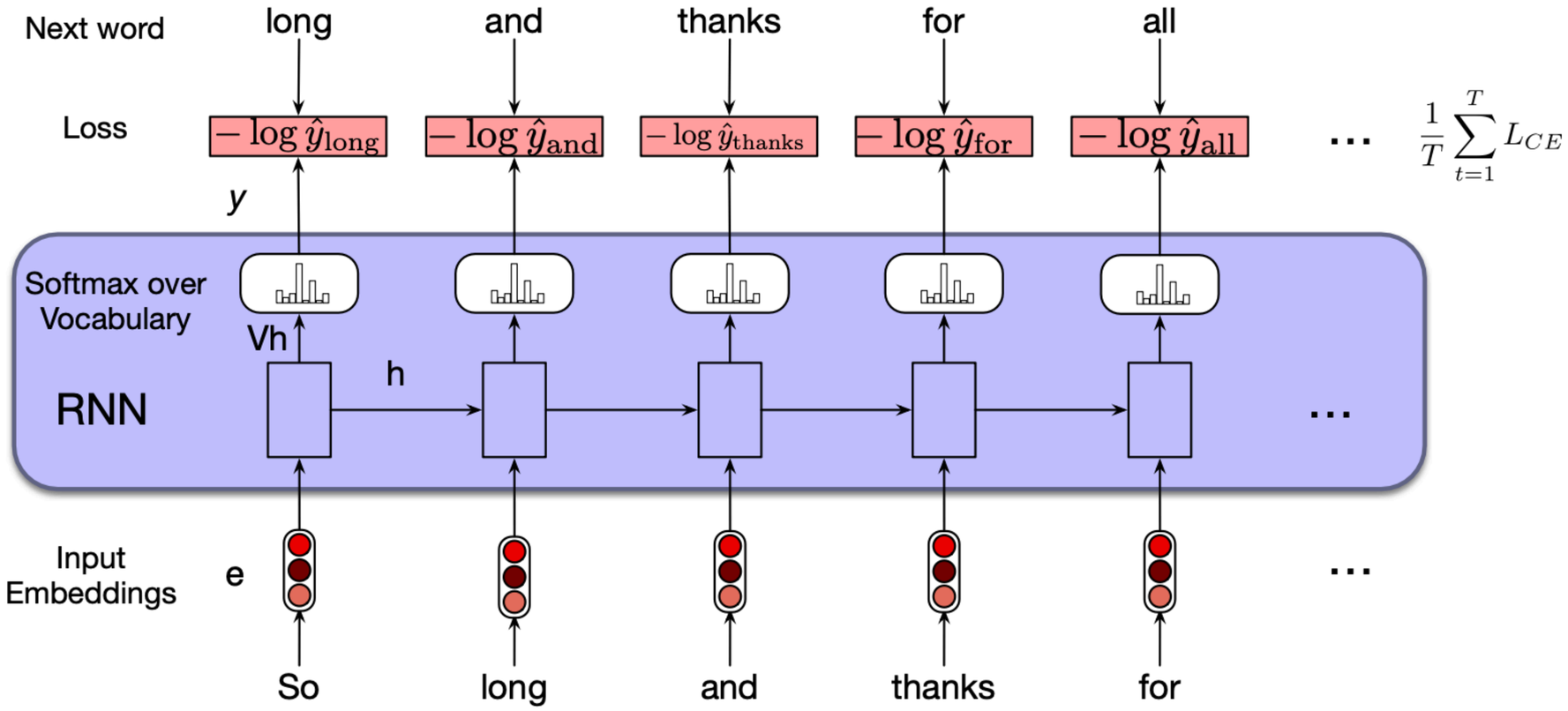
- We have one vector $\mathbf{h}$ to handle the entire input. What can it hold?
  - Word meaning: definitely!
  - Phrase meaning: very well, but not perfectly
  - Sentence meaning: iffy
  - Document meaning: probably not
- We're doing significantly better, but can't effectively handle super long context quite yet.



"You can't cram the meaning of a whole %&! $ing sentence into a single $&!*ing vector!"

—Raymond Mooney, 2014

Why RNNs Are Cursed



In reality, it is *very hard* to make use of distant context.

It's also *very hard* to learn well with long sequences: the same matrix is getting multiplied over and over and over...

Vanishing and Exploding Gradients, Revisited

- Imagine you have a sequence that's t tokens long. W and $\mathbf{h}$ are a weight matrix and hidden state, respectively.
    - β_W and $\beta_{\mathbf{h}}$ are the upper bound of the norms of these objects.
- The partial derivative will be $(\beta_W \beta_{\mathbf{h}})^{t-k}$
    - When t is large, this can get *huge* (explode) or shrink to nothing (vanish)

In both cases, nothing of importance is learned. If vanishing, updates to the parameters become 0. If exploding, the parameters are jumping around too much and too fast to be useful.

Long Short-term Memory Networks (LSTMs)

Motivation

- *Idea*: Instead of learning what to keep or forget at the same time, let's divide context management into two subproblems:
    1. Forget information that is no longer needed
    2. Add information that is likely to be needed later

- This is less about the architecture and more about context management.
- Let's implement this functionality via **gates** (implemented as additional weights).

Long Short-term Memory Networks (LSTMs)

We decide what information to extract:

$$\mathbf{g}_t = \sigma(U_g \mathbf{h}_{t-1} + W_g \mathbf{x}_t)$$

The **add gate** selects information to add to the current context:

$$\mathbf{i}_t = \sigma(U_i \mathbf{h}_{t-1} + W_i \mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

We add the modified context to get the new context vector:

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

Finally, the **output gate** decides what information is needed for the current hidden state:

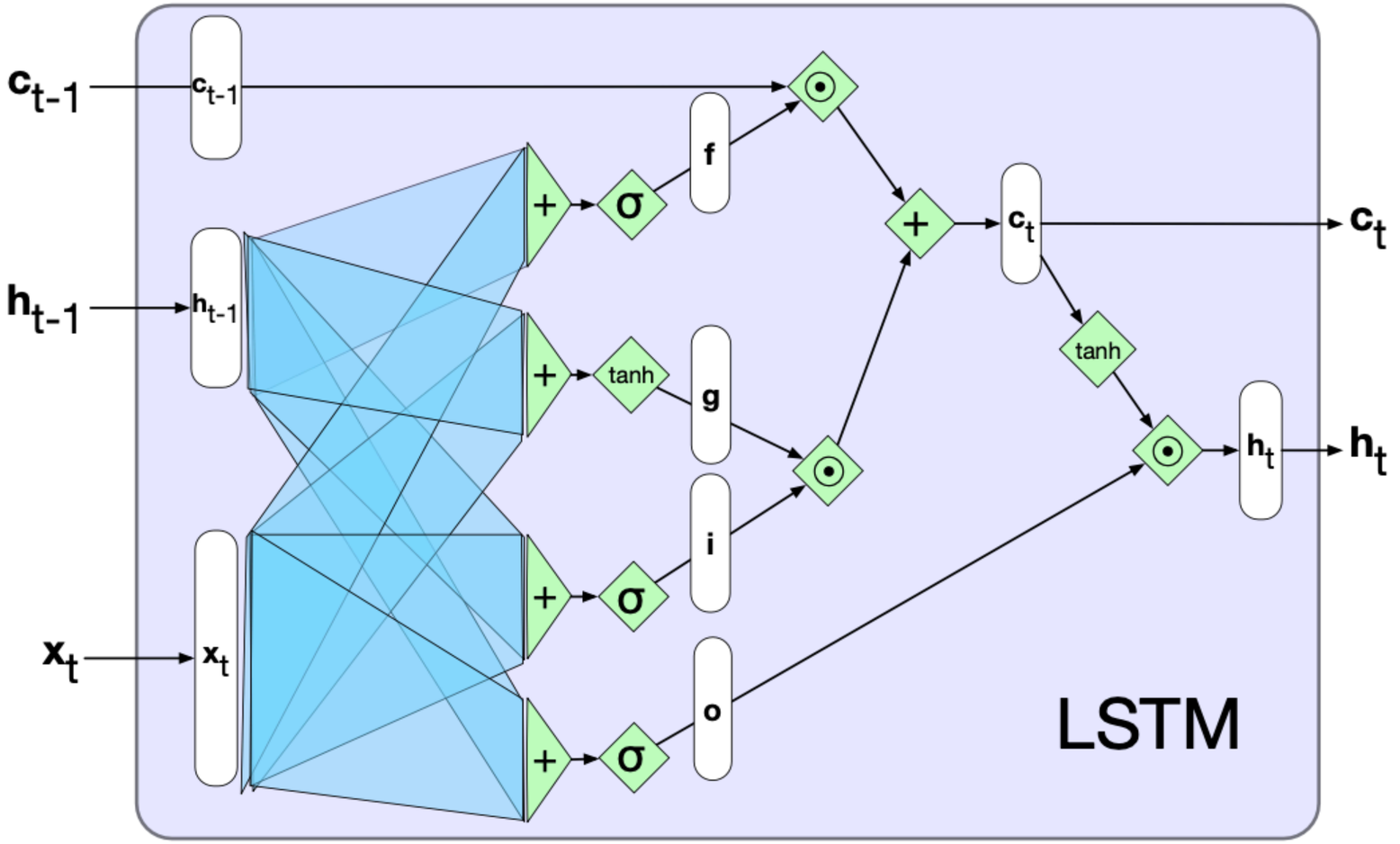
$$\mathbf{o}_t = \sigma(U_o \mathbf{h}_{t-1} + W_o \mathbf{x}_t)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

First, the **forget gate** decides what information to delete:

$$\mathbf{f}_t = \sigma(U_f \mathbf{h}_{t-1} + W_f \mathbf{x}_t)$$

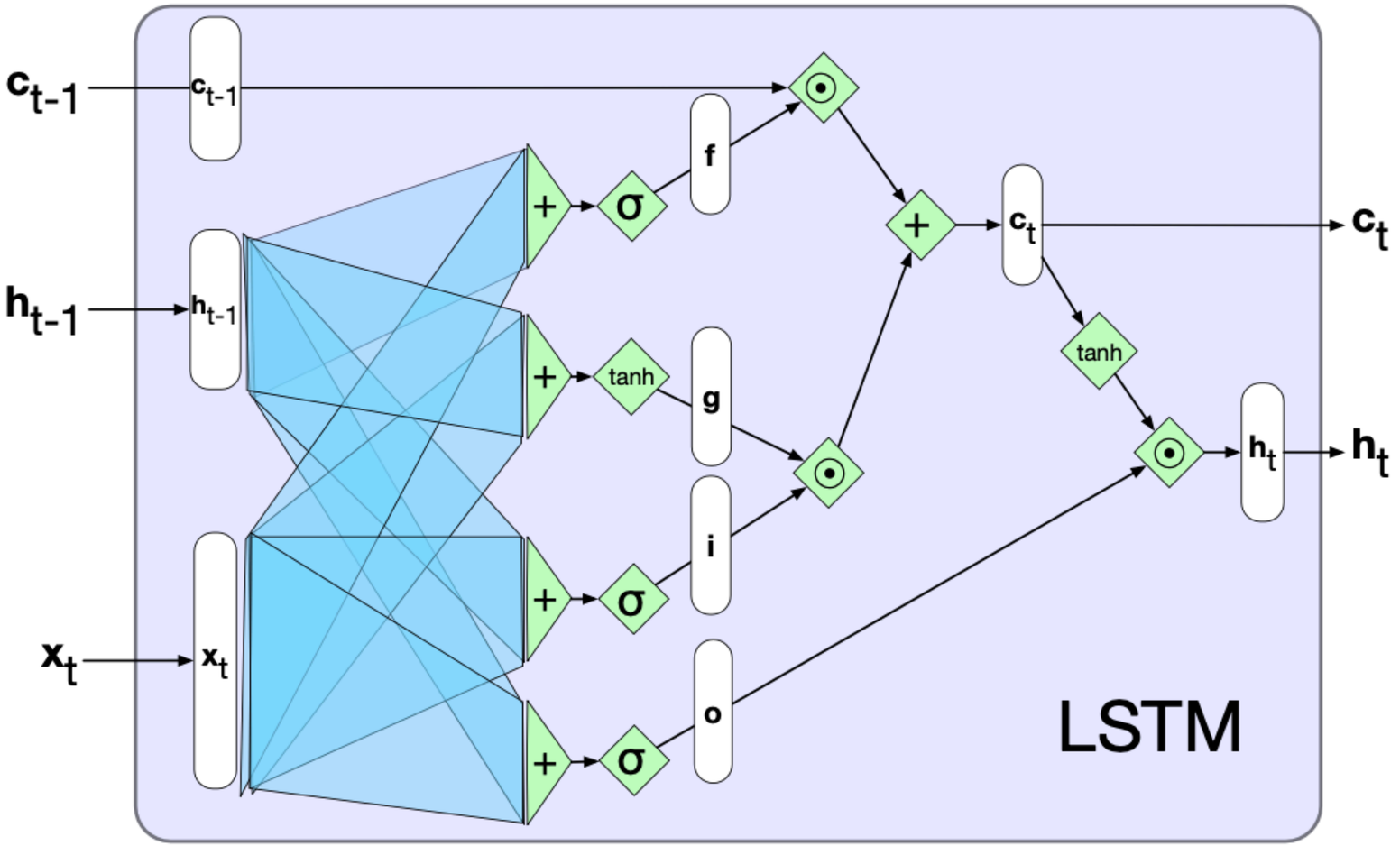
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$



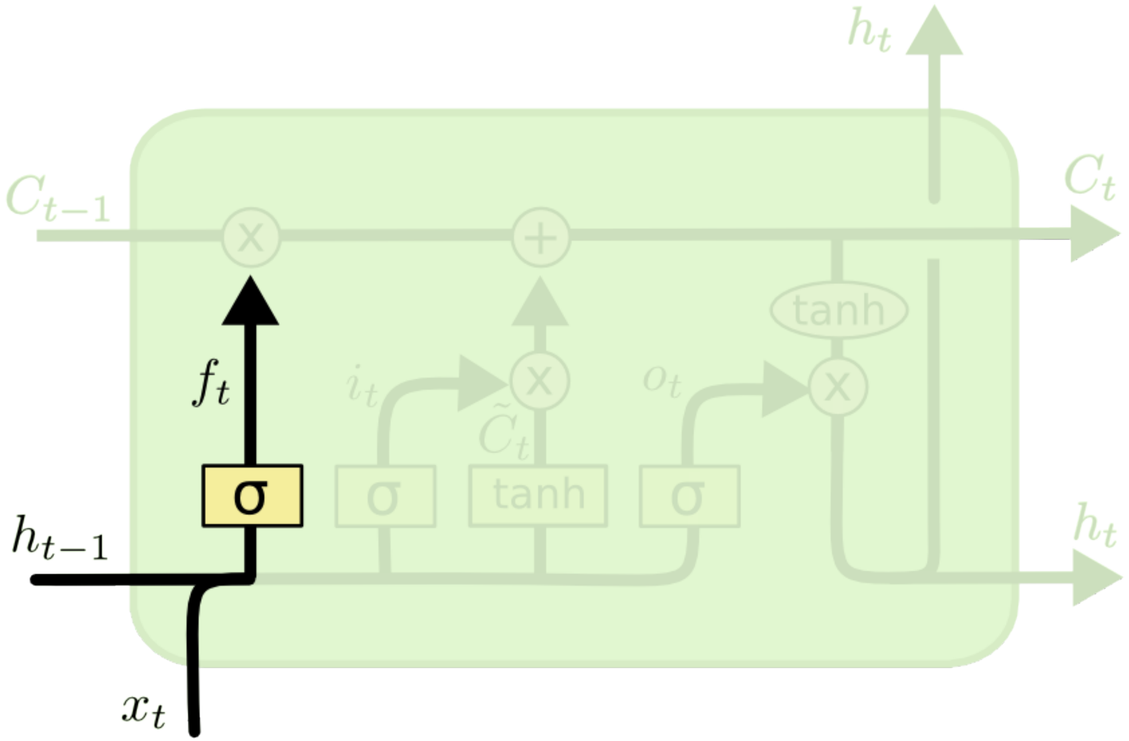
Long Short-term Memory Networks (LSTMs)

- You can think of the $\mathbf{c}$ as being your **long-term memory**: what should we hold onto for later?

- You can think of $\mathbf{h}$ as the **short-term memory**: what's needed to predict the very next token?



Understanding LSTM Gates

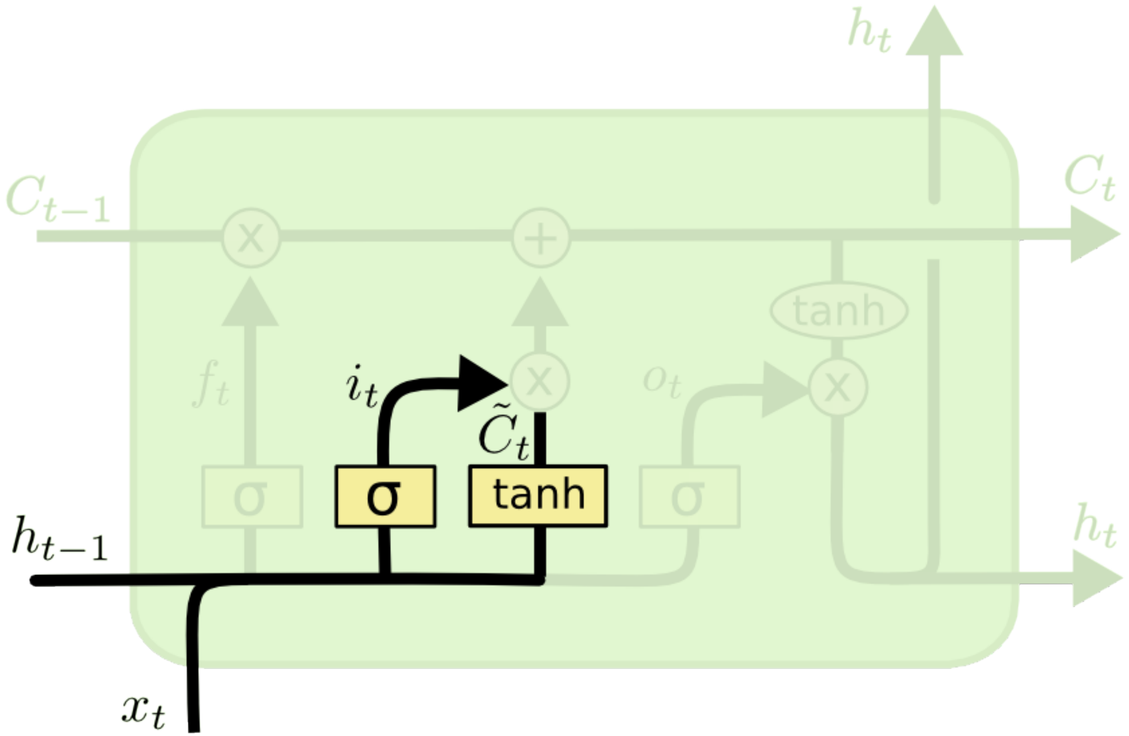


Forget gate:

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

Based on the current input, how much of the previous hidden state can we throw away?

Understanding LSTM Gates



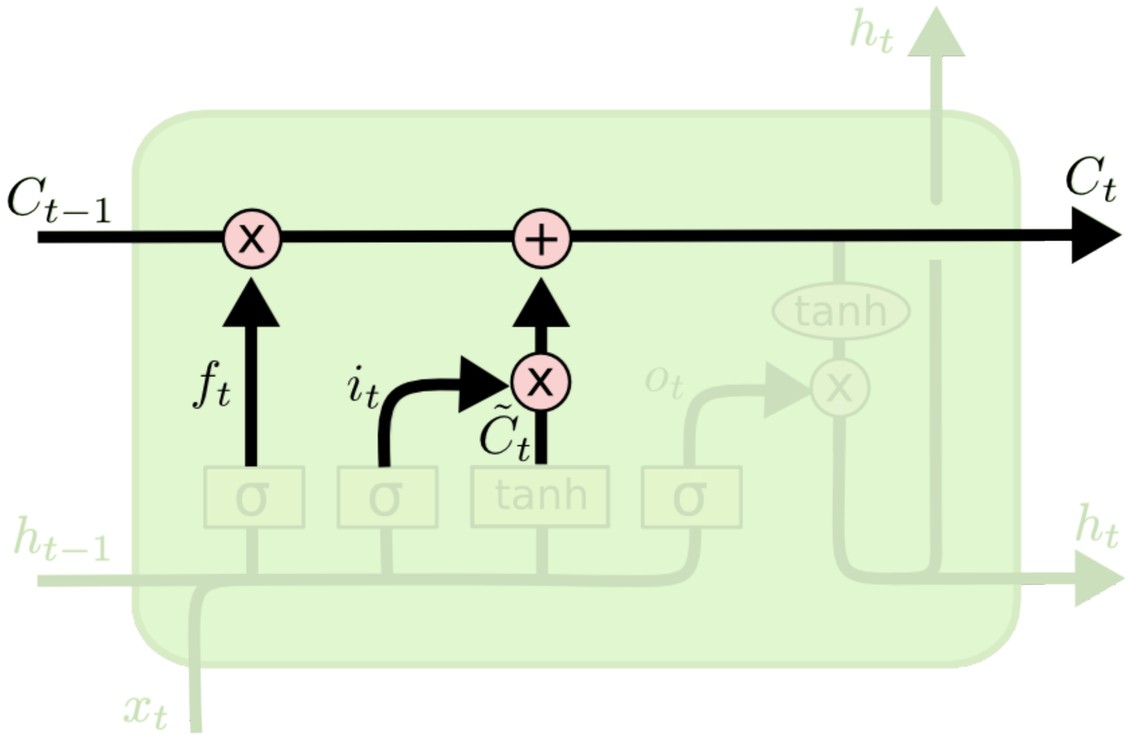
Input gate:

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

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

How much of the information in the current input should be incorporated into the context vector?

Understanding LSTM Gates

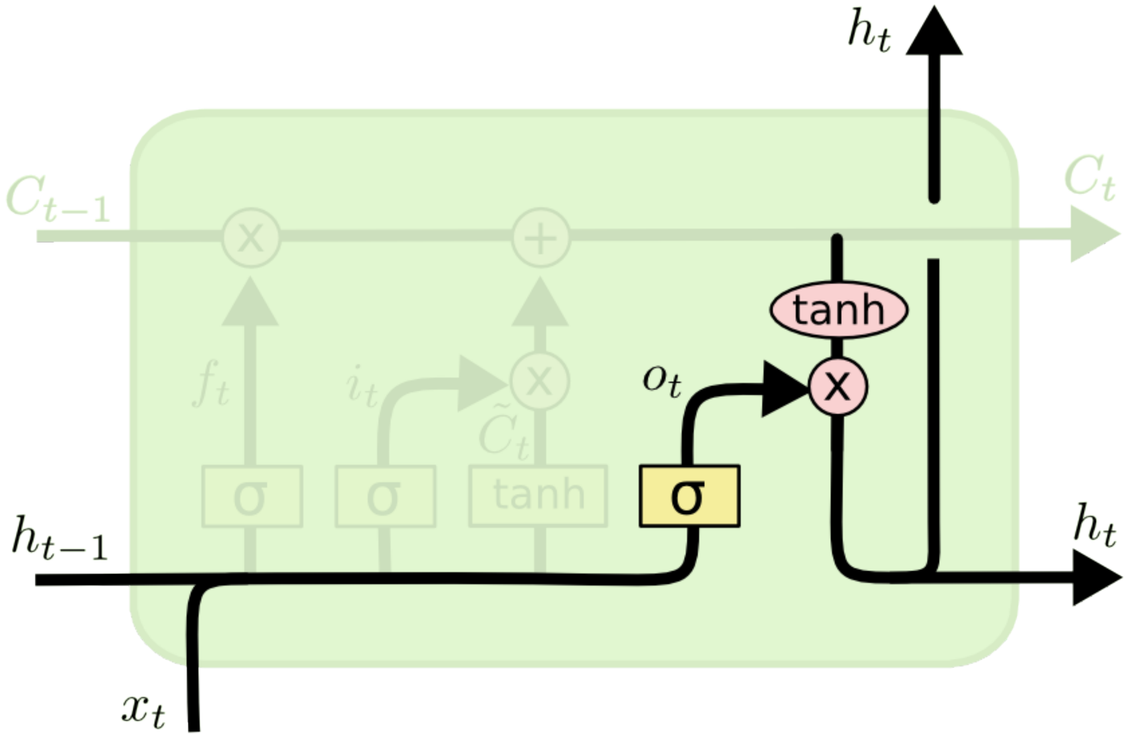


Context vector update:

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

Update the context vector: forget whatever the forget gate said to, add whatever the input gate said to.

Understanding LSTM Gates



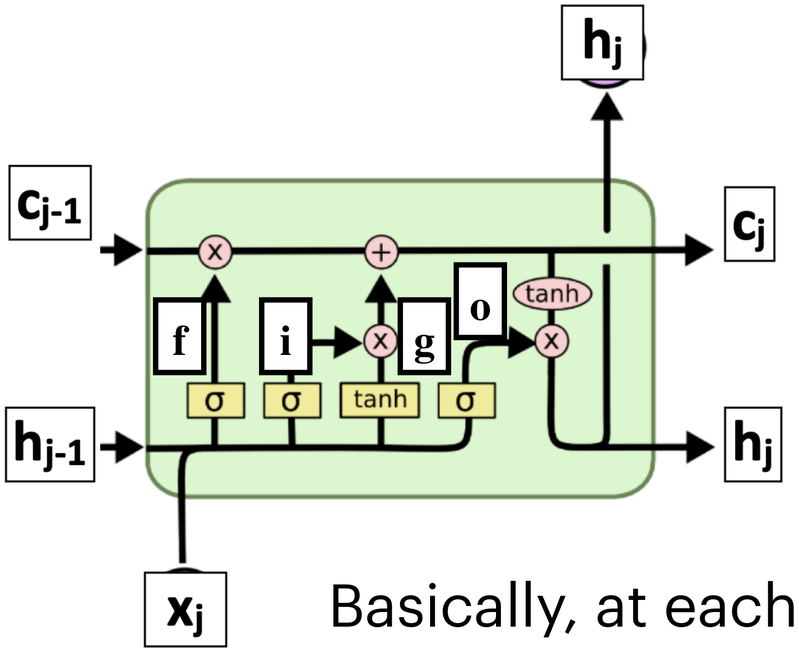
Output gate:

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

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

Decide what to output: run the context vector through a tanh, run the hidden state through an output gate, and multiply these all together to get the next hidden state.

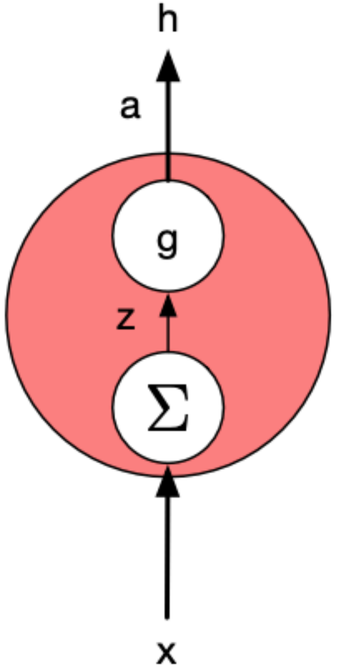
Understanding LSTM Gates



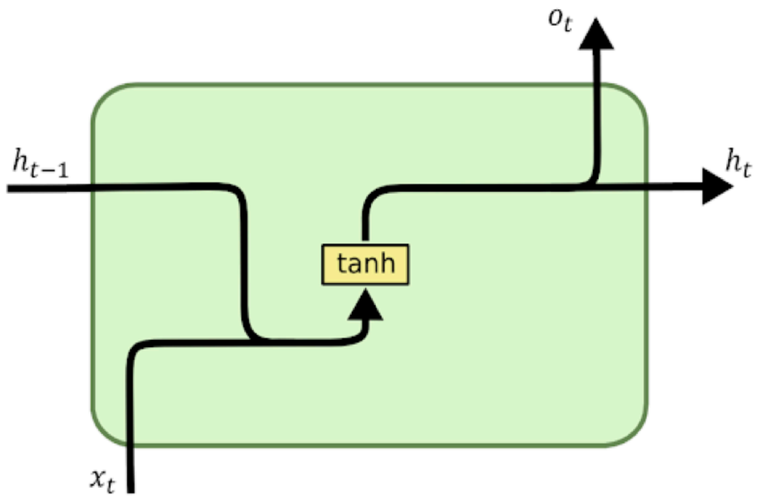
- The $\mathbf{f}$, $\mathbf{i}$, $\mathbf{o}$ gates control information flow across time
- $\mathbf{g}/\tilde{C}$ is the main computation of new information

Basically, at each timestep, we're computing:

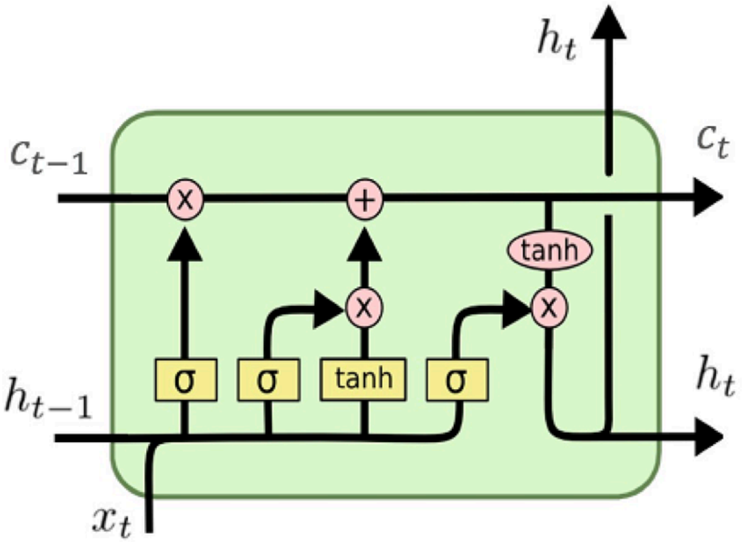
- How much can we ignore old values of $\mathbf{c}$ for this timestep?
- How much should we incorporate the input $\mathbf{x}$ at this timestep?
- What should we output at this timestep?



Feedforward NNs (FNNs)



RNNs



LSTMs

To recap: FNNs (from Tuesday) take the current input and learn to output some hidden vector.

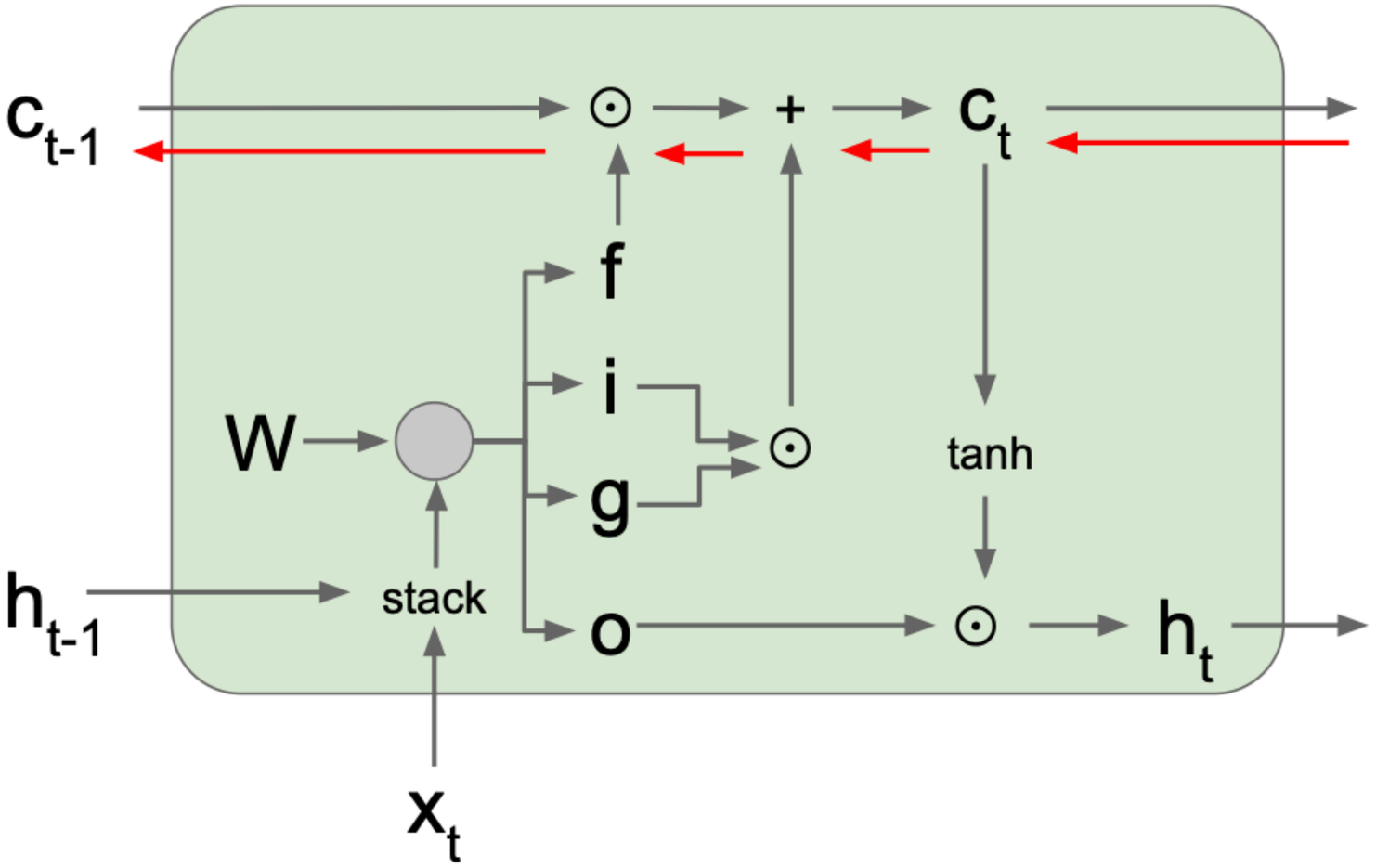
RNNs take the current input and previous hidden vector, and output a new hidden vector.

LSTMs take the current input, previous hidden vector, and previous context vector, and output a new hidden vector and new context vector.

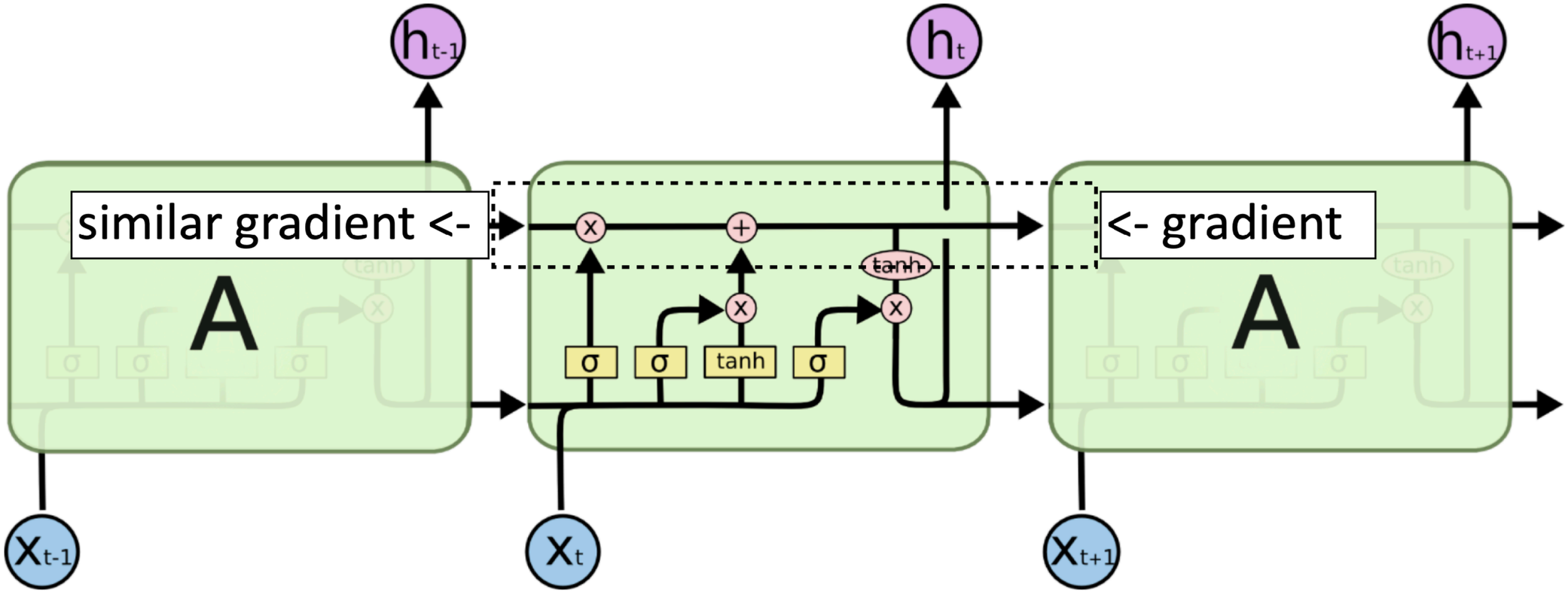
LSTM Gradients

- Backpropagating through the context vector $\mathbf{c}$ does not require us to multiply by W!

- We just need elementwise multiplications with the forget gate $\mathbf{f}$.
  - *Much* easier to avoid exploding or vanishing gradients.

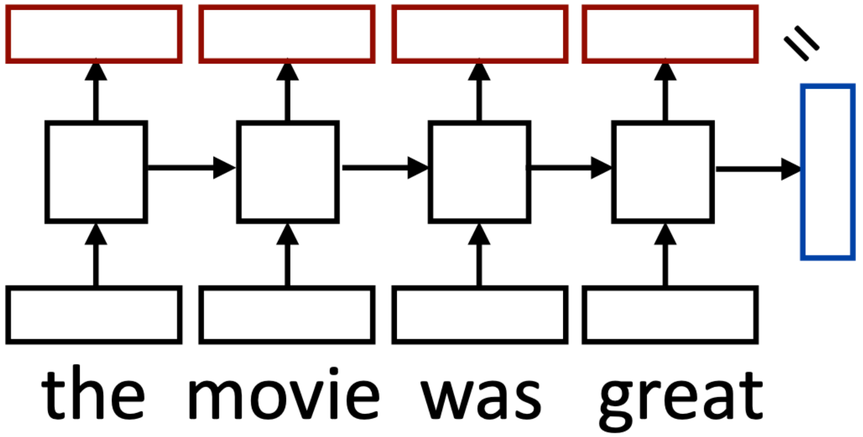


LSTM Gradients



The gradient still diminishes over time, but in a more controlled way, and by less than in RNNs.

What can LSTMs do?



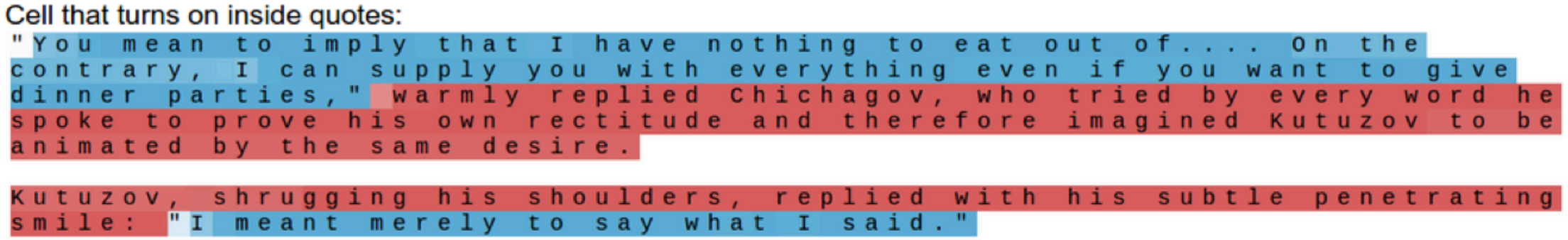
- Encode a sentence
  - Sentence classification
- Encode *pairs of* sentences
  - Paraphrase identification, natural language inference
- Predict a label for each word in the sentence
  - POS tagging, language modeling
- Translation, generation

VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu

VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu



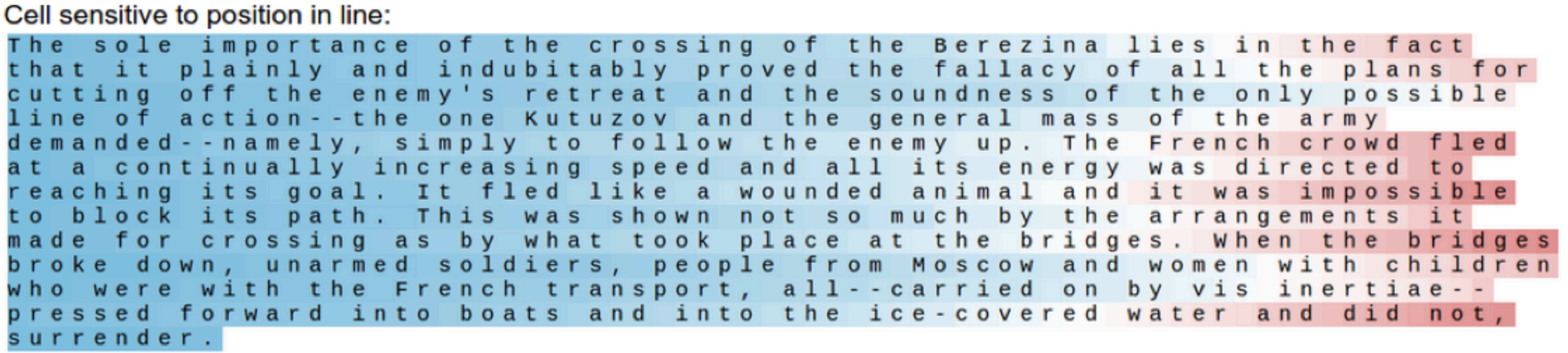
This shows the activations of a dimension of the context vector ("cell").

Blue: positive

Red: negative

VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu



VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu

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;
}
```

VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu

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

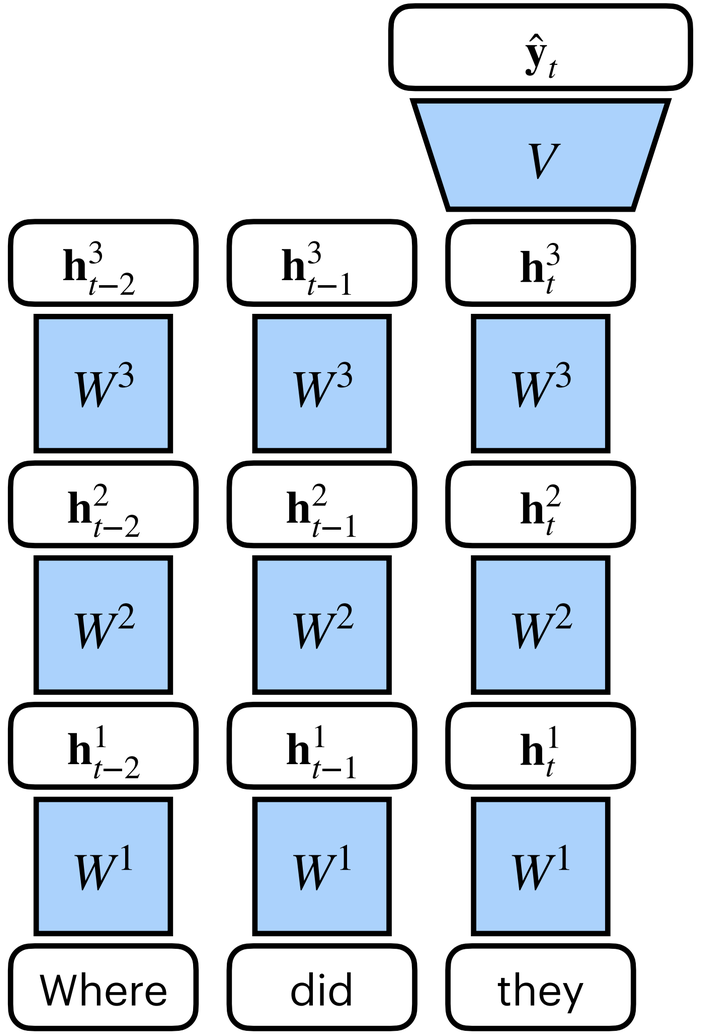
```
/*  Unpack  a  filter  field's  string  representation  from  user-space
 *  buffer.  */
char  *audit_unpack_string(void  **bufp,  size_t  *remain,  size_t  len)
{
  char  *str;
  if  (!*bufp  ||  (len  ==  0)  ||  (len  >  *remain))
    return  ERR_PTR(-EINVAL);
  /*  Of  the  currently  implemented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

Deep RNNs

Instead of a single hidden layer, you can have many.

This allows the model to learn more abstract representations of the inputs.

In practice, I would strongly recommend doing this if your dataset is big enough!



The Future Matters

I was _____.

I was _____ happy that I got to see them before they left.

I was _____ already when they arrived.

The Future Matters

I was interested.

I was very happy that I got to see them before they left.

I was there already when they arrived.

The Future Matters

I was **interested**.

I was **very** happy that I got to see them before they left.

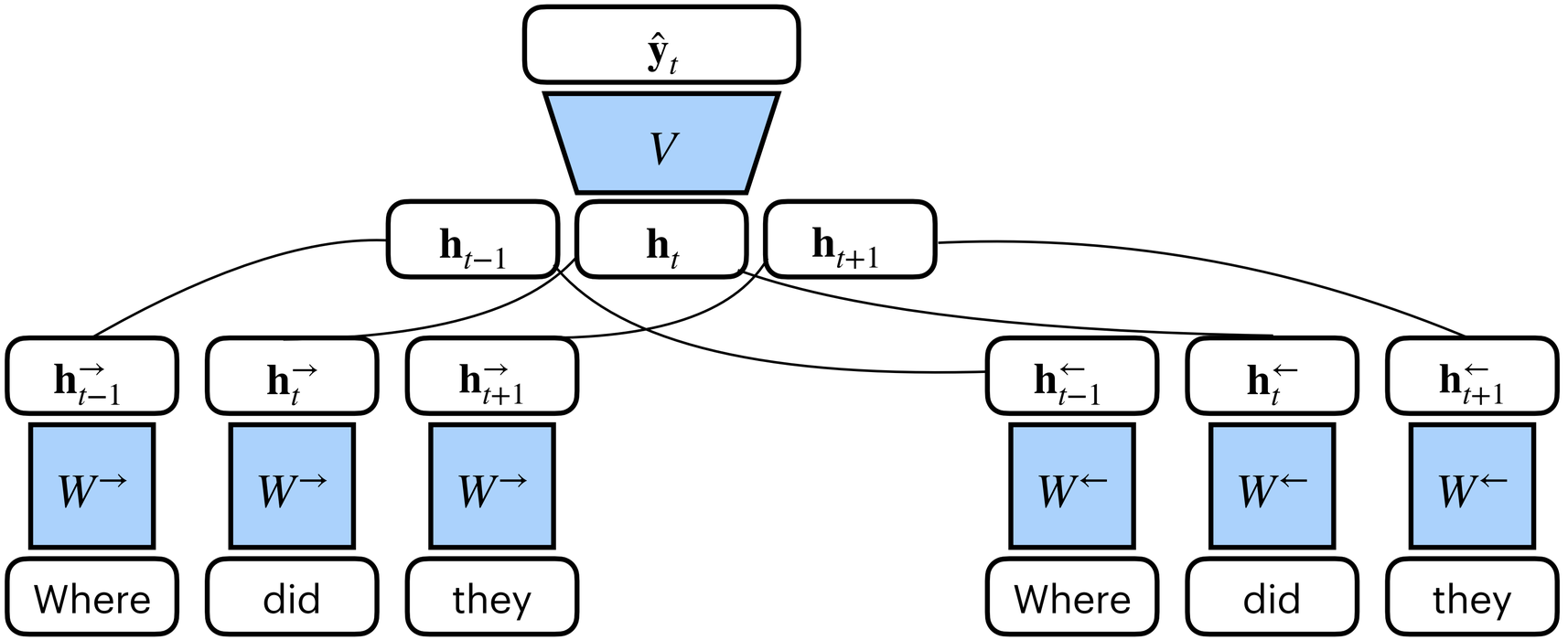
I was **there** already when they arrived.

So far, our RNNs have gone just left-to-right.

Clearly, access to future tokens helps us predict a token more accurately.

Idea: let's do a left-to-right pass *and* a right-to-left pass.

Bidirectional RNNs

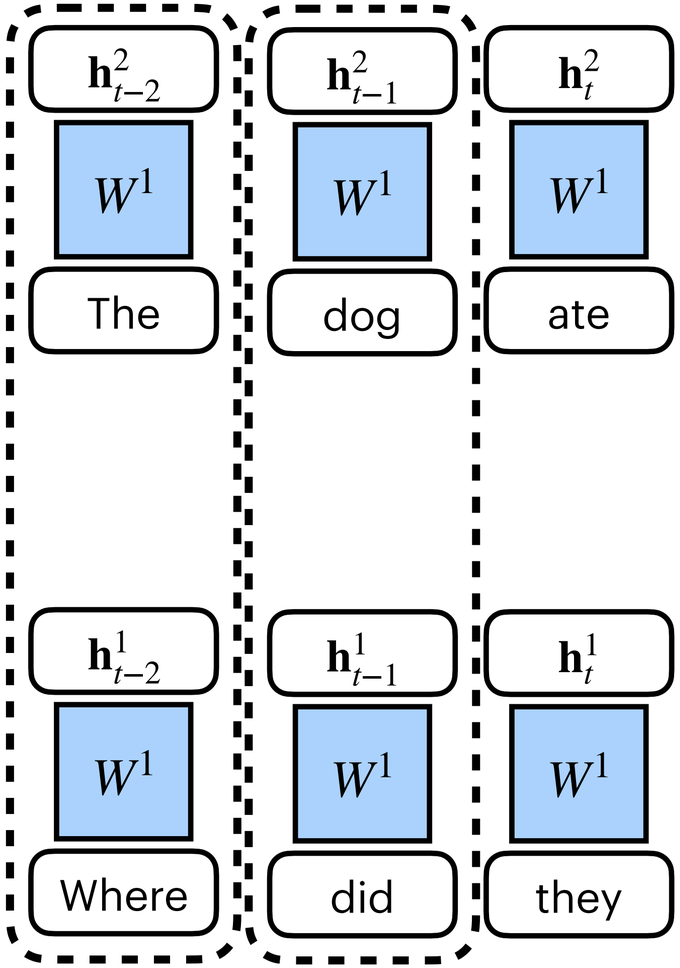


Take one RNN that goes left-to-right → and one that goes right-to-left ←.

For a given timestep, concatenate their hidden states: $\mathbf{h}_t = \mathbf{h}^{\rightarrow}_t \oplus \mathbf{h}^{\leftarrow}_t$

These work well for sequence classification, but aren't good for generation.

Batching

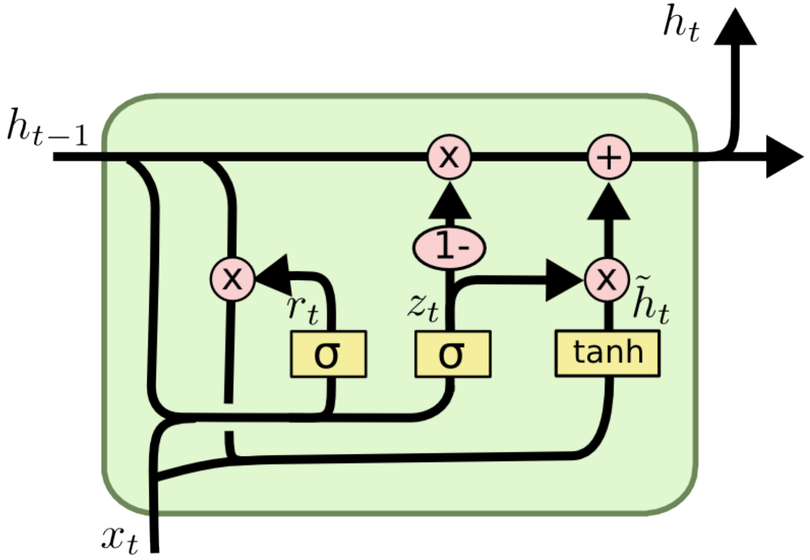


We can parallelize across b examples (the batch size).

However, we *cannot* parallelize across timesteps.
Every timestep depends on previous timesteps.

Other RNNs

- You may encounter **Gated Recurrent Units** (GRUs)
  - Similar in spirit to LSTMs, but fewer gates and more computationally efficient
  - Also addresses the vanishing/exploding gradients problem
  - Performs similarly to LSTMs in many cases



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Summary

- **Recurrent neural networks (RNNs)** allow us to consider *way more* prior context than n-grams.
    - Our first neural language models!

- Vanilla RNN LMs are better than n-gram LMs, but they have issues: most notably, **vanishing/exploding gradients** and bad long-distance dependency tracking
- **LSTMs** improve upon the vanilla RNN via the extensive use of **gates**
- RNN LMs can learn surprisingly sophisticated concepts

Homework 1

- You will implement (at least) two types of language model: an n-gram, an RNN, and (optionally, for extra credit) an LSTM.
  - Don’t leave this to the last minute!
  - Recommended schedule:
    - Week 1: BPE tokenizer, n-gram language model
    - Week 2: RNN LM (and optionally, LSTM)
    - +2 days: conceptual questions about embeddings

- Recommendations:
  - Use Google Colab for debugging. GPUs will speed things up.
  - When debugging, keep the number of epochs, vocab size, and dataset size low. Increase these later to get your final scores.