Recurrent Neural Networks

Wei Xu

(many slides from Greg Durrett)

This Lecture

- Recurrent neural networks

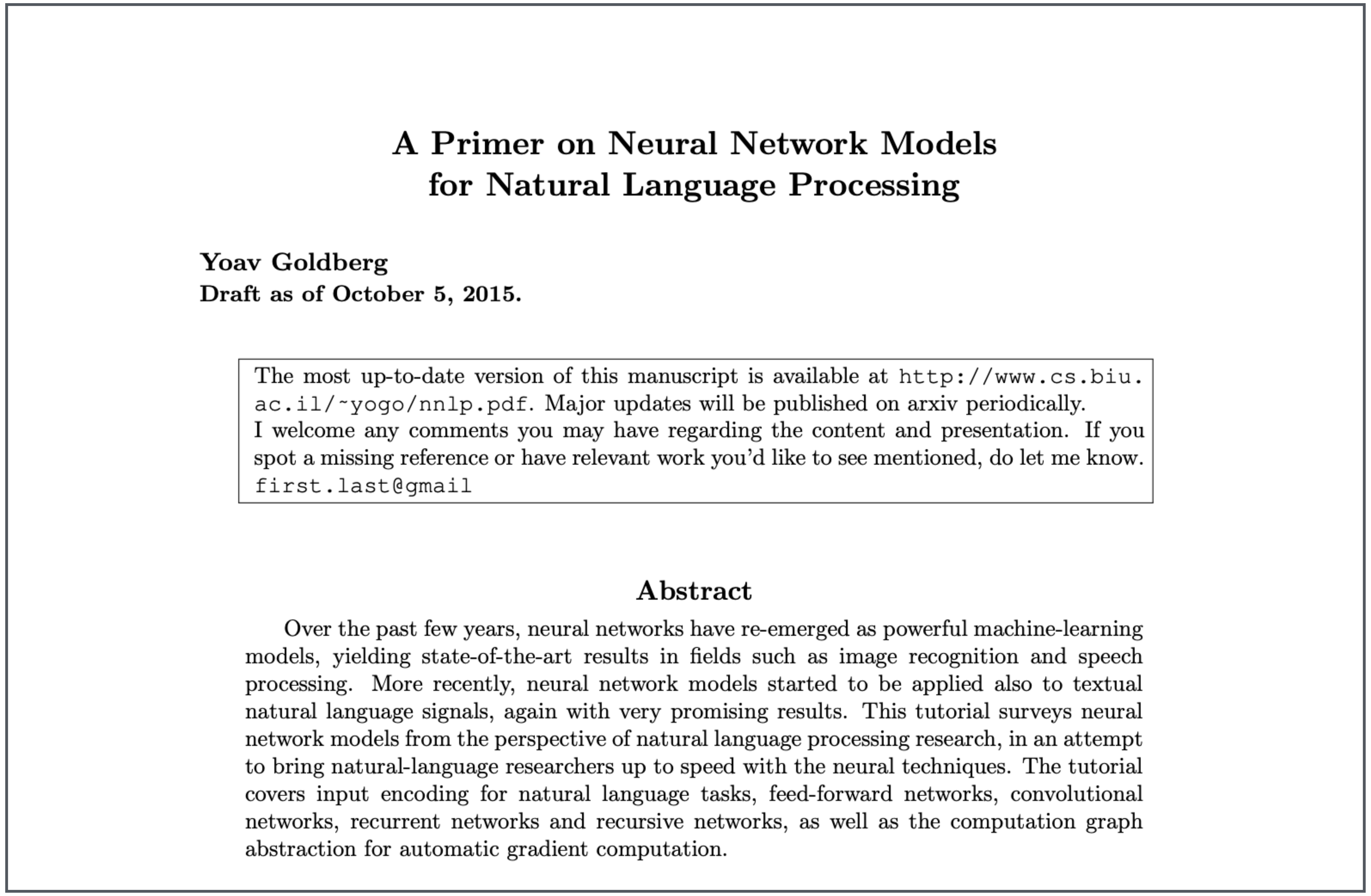
- Vanishing gradient problem

- LSTMs / GRUs

- Applications / visualizations

Readings

- Reading: RNNs
  - Eisenstein 7.6
  - Jurafsky and Martin, Chapter 9
  - Goldberg 10, 11



A Primer on Neural Network Models for Natural Language Processing

Yoav Goldberg
Draft as of October 5, 2015.

The most up-to-date version of this manuscript is available at http://www.cs.biu.ac.il/~yogo/nnlp.pdf. Major updates will be published on arxiv periodically. I welcome any comments you may have regarding the content and presentation. If you spot a missing reference or have relevant work you'd like to see mentioned, do let me know. first.last@gmail

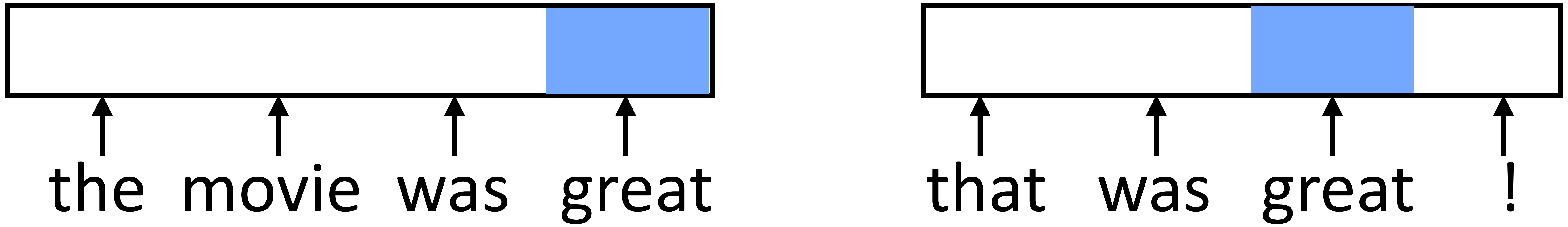
Abstract

Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textual natural language signals, again with very promising results. This tutorial surveys neural network models from the perspective of natural language processing research, in an attempt to bring natural-language researchers up to speed with the neural techniques. The tutorial covers input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.

RNN Basics

RNN Motivation

- Feedforward NNs isn't the best to handle sentences with variable length, words with multiple senses, or same words appear at different positions



- These don't look related (*great* is in two different orthogonal subspaces)

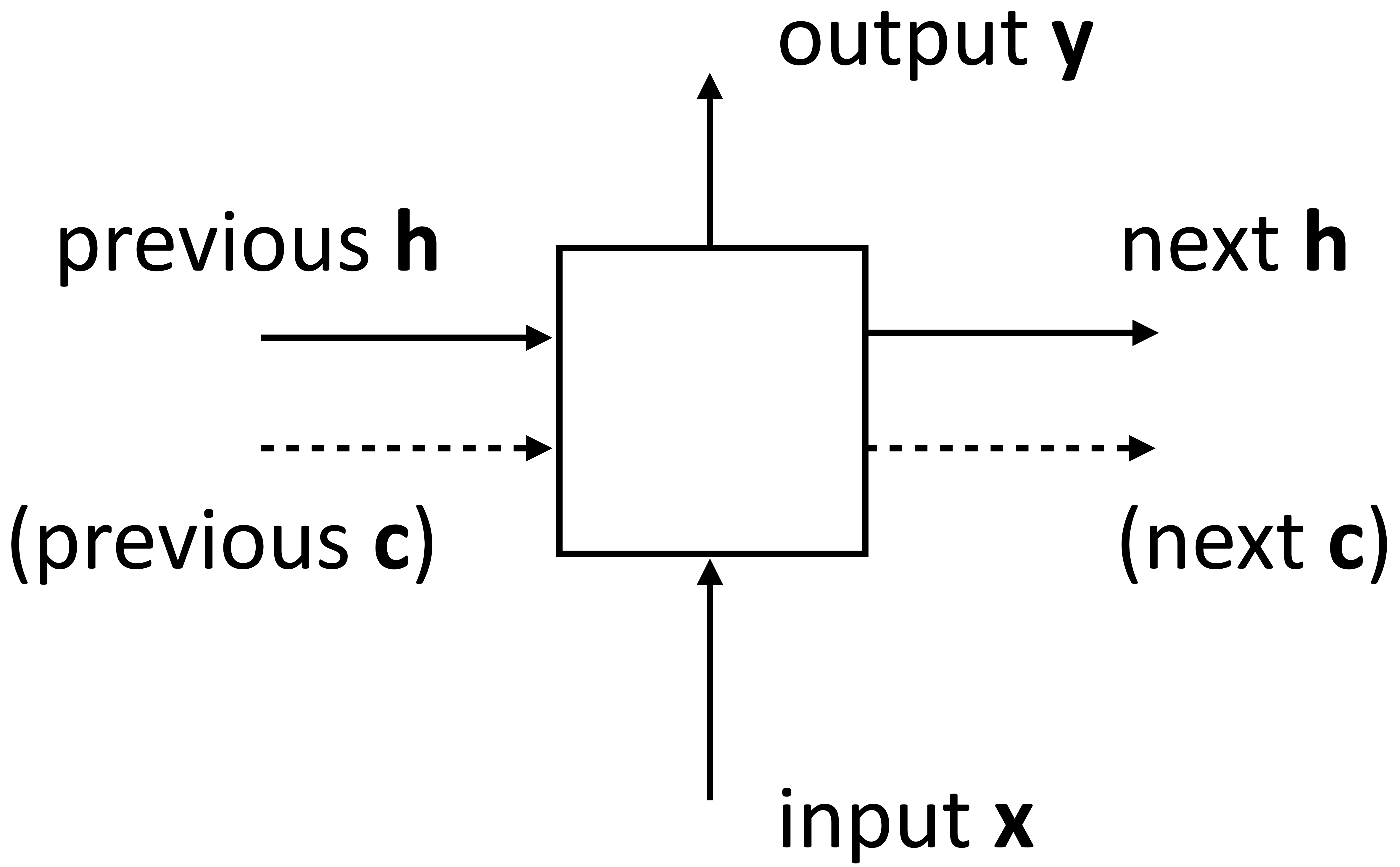
- Instead, we need to:

1) Process each word in a uniform way

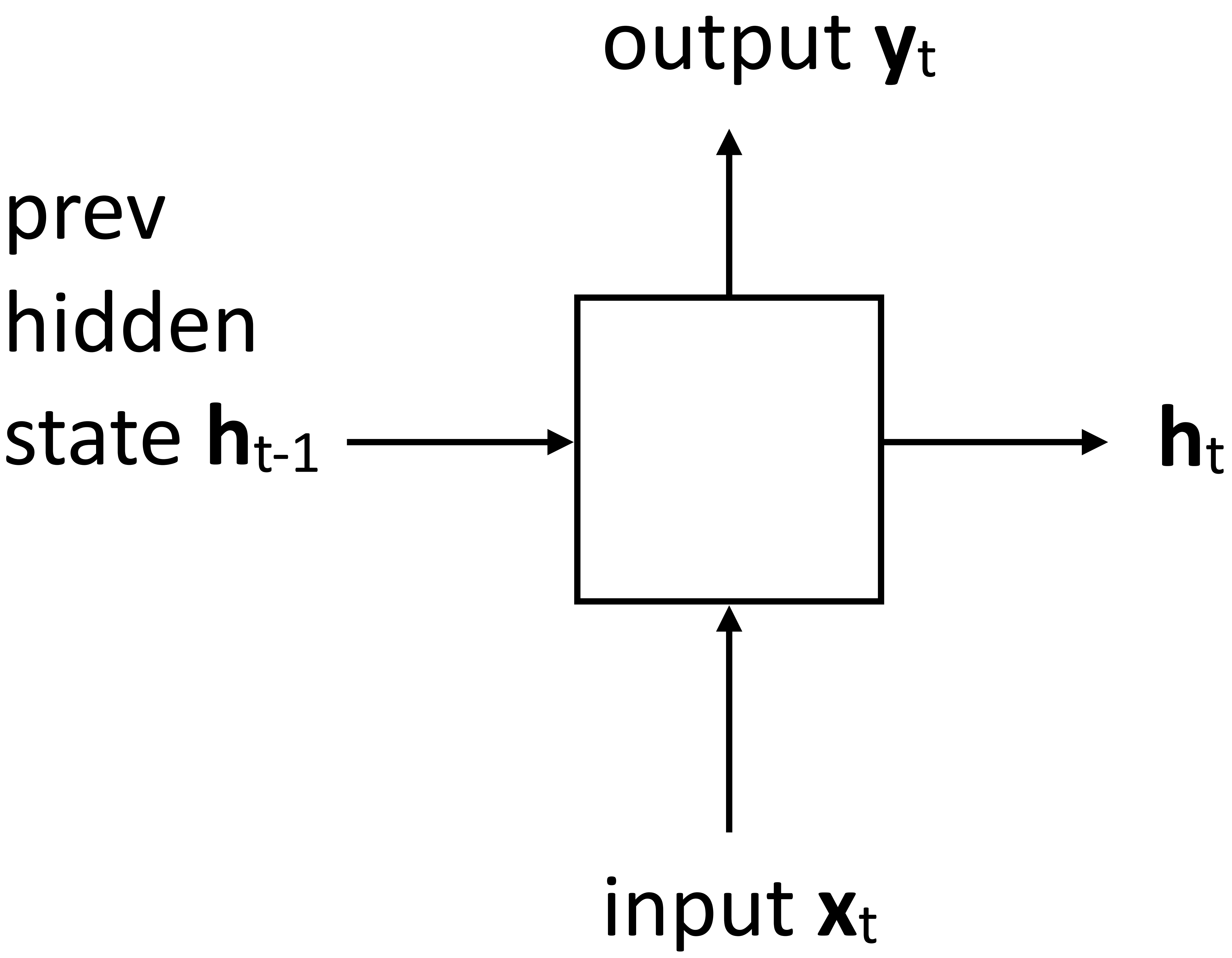
2) …while still exploiting the context that that token occurs in

RNN Abstraction

- Cell that takes some input **x**, has some hidden state **h**, and updates that hidden state and produces output **y** (all vector-valued)



Elman Networks



output $\mathbf{y}_t$

prev hidden state $\mathbf{h}_{t-1}$

$\mathbf{h}_t$

input $\mathbf{x}_t$

$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

- Updates hidden state based on input and current hidden state

$$\mathbf{y}_t = \tanh(U\mathbf{h}_\mathbf{t} + \mathbf{b}_y)$$

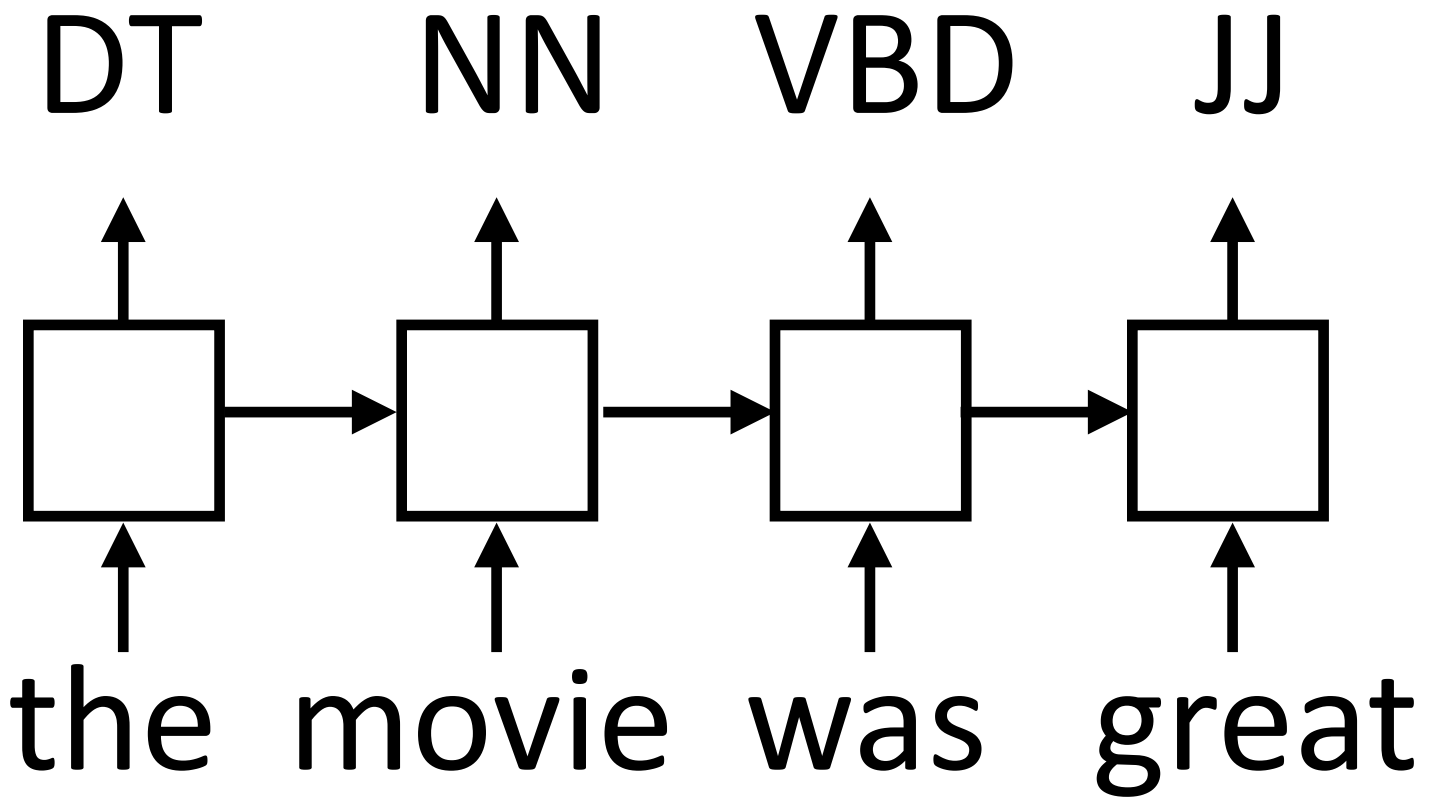
- Computes output from hidden state

- Long history! (invented in the late 1980s)

Elman (1990)

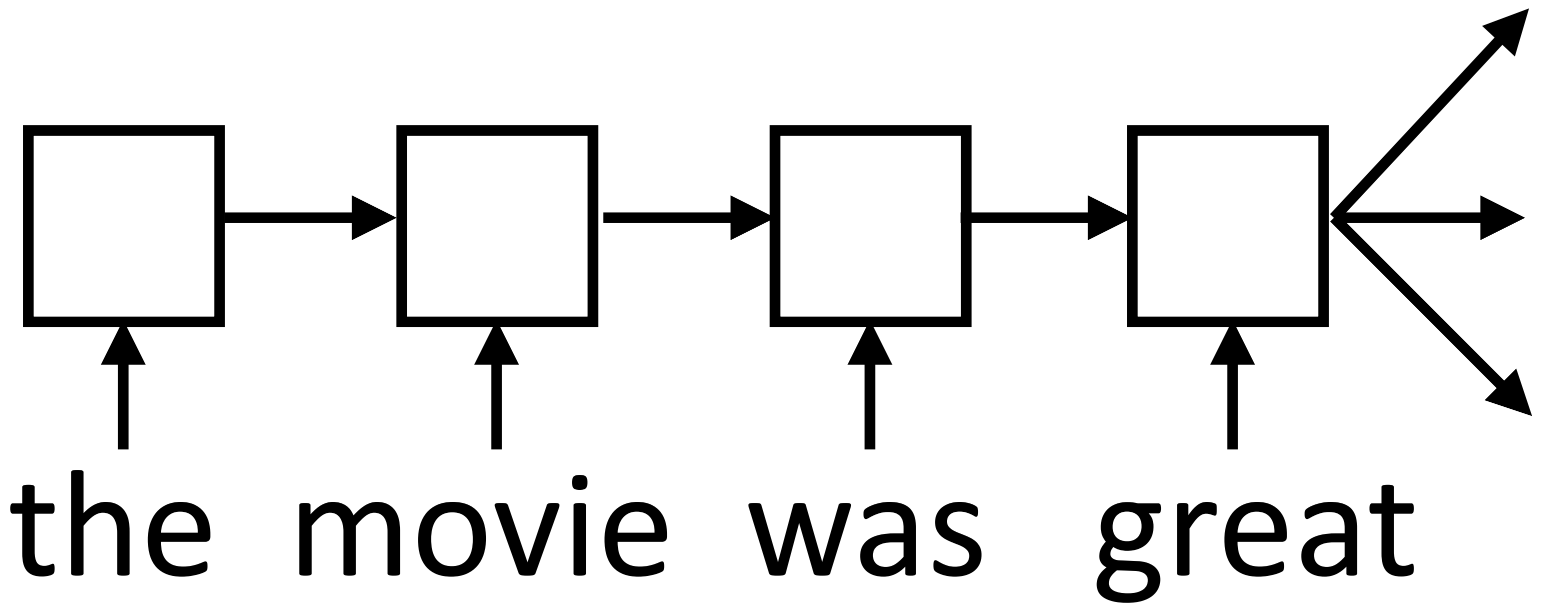
RNN Uses

- Transducer: make some prediction for each element in a sequence



output **y** = score for each tag, then softmax

- Encoder: encode a sequence into a fixed-sized vector and use that for some purpose

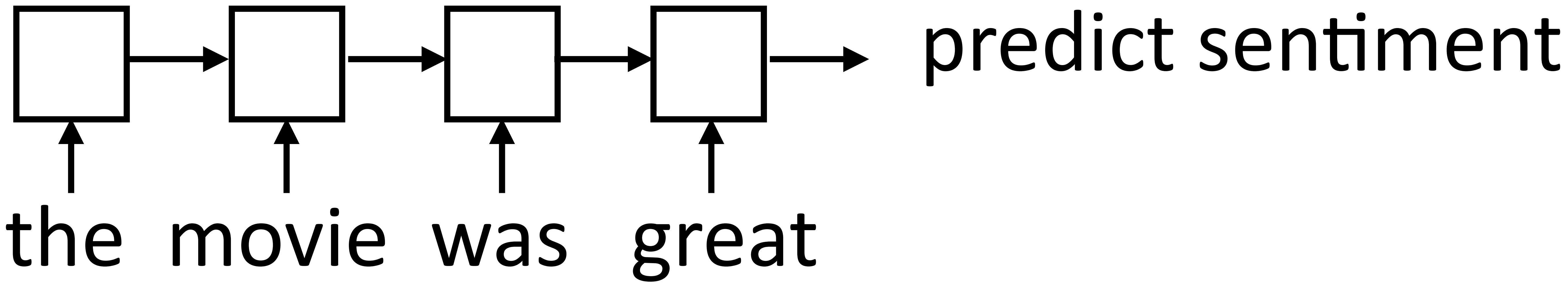


predict sentiment (matmul + softmax)

translate

paraphrase/compress

RNN Intuition

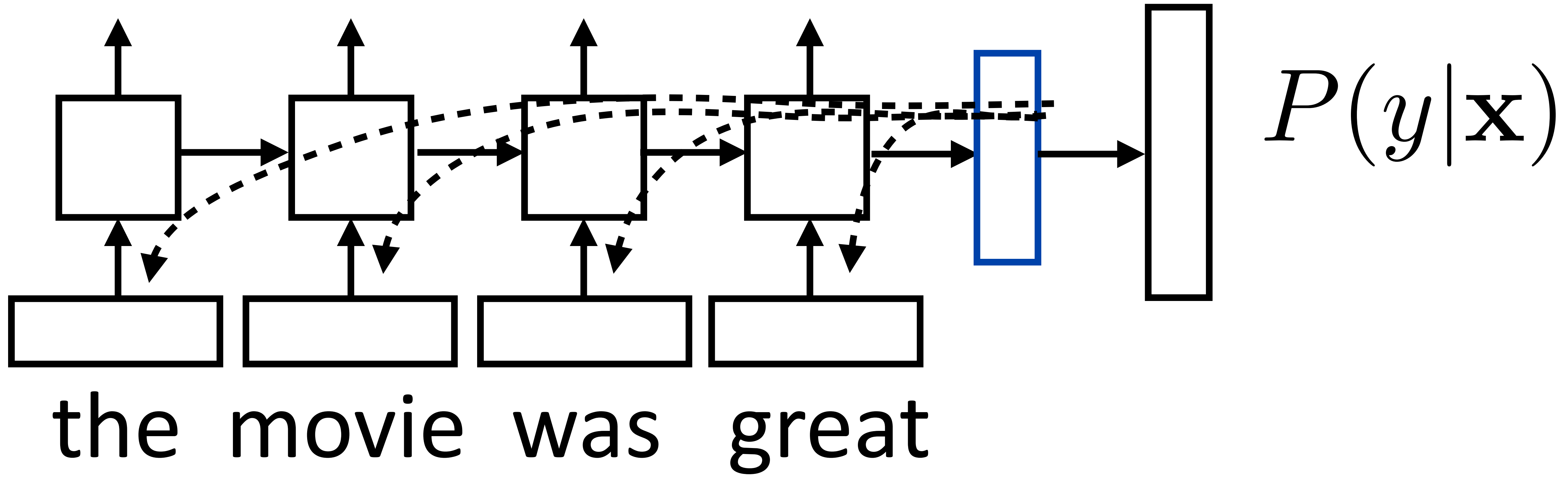


- RNN potentially needs to learn how to "remember" information for a long time!

it was my favorite movie of 2016, though it wasn't without problems -> **+**

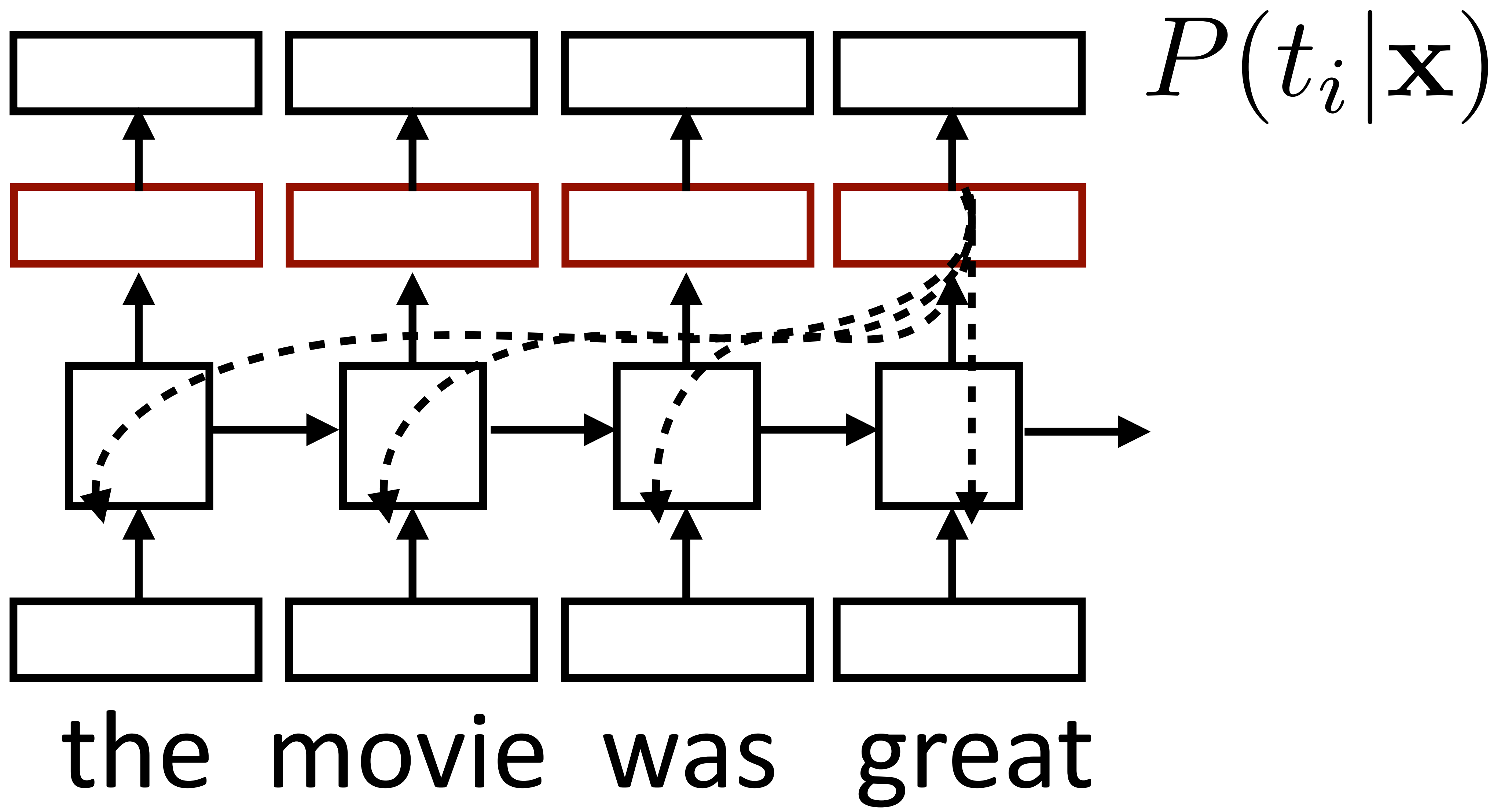
- "Correct" parameter update is to do a better job of remembering the sentiment of *favorite*

Training RNNs



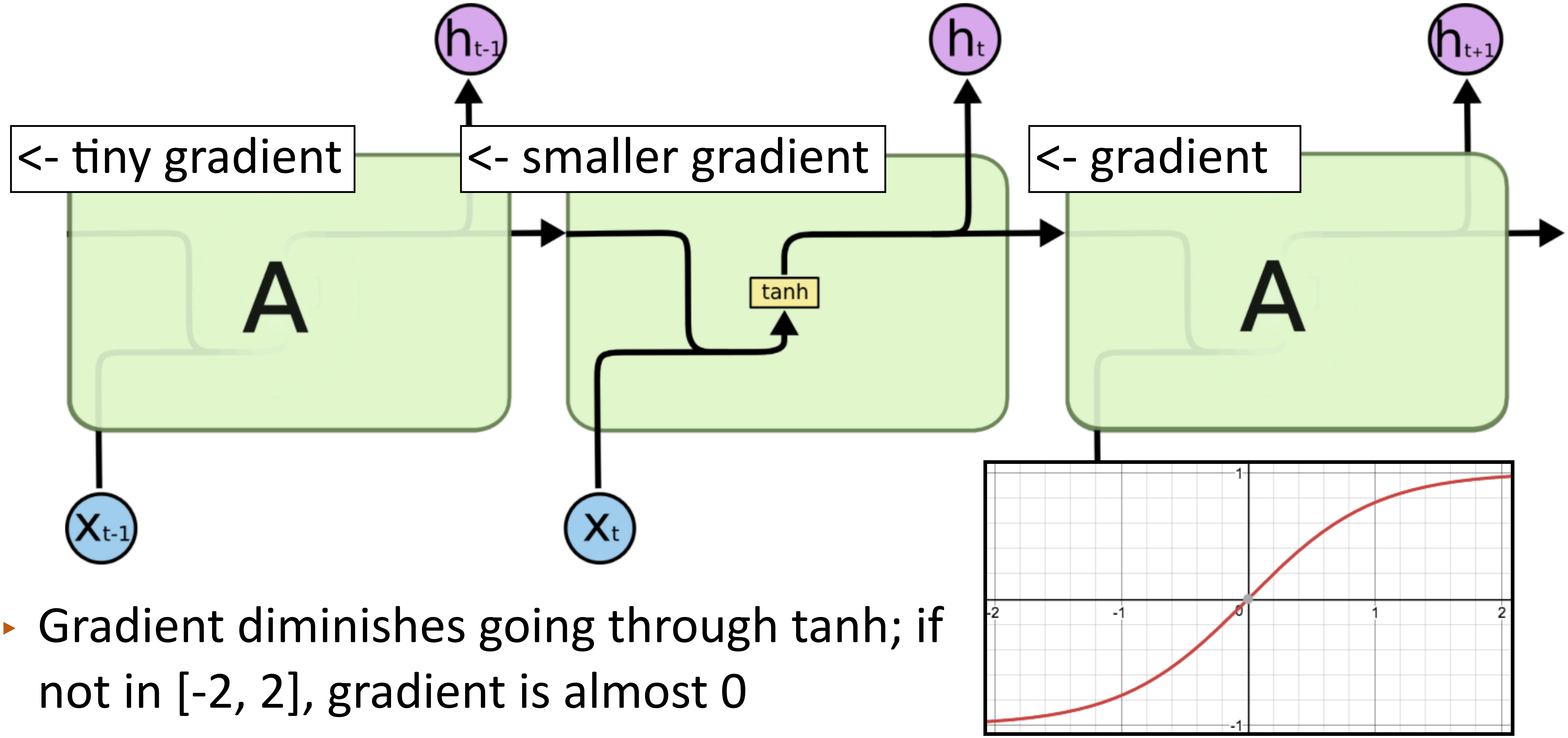
- Example: sentiment analysis
- Loss = negative log likelihood of probability of gold label (softmax or use SVM or other loss)
- "Backpropagation through time": build the network as one big computation graph, some parameters are shared

Training RNNs



- Example: POS tagging, language modeling (predict next word given context)
- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network

Vanishing Gradient



- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0

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

LSTMs/GRUs

A Bit of History

- Long Short-term Memory (Hochreiter & Schmidhuber, 1997)

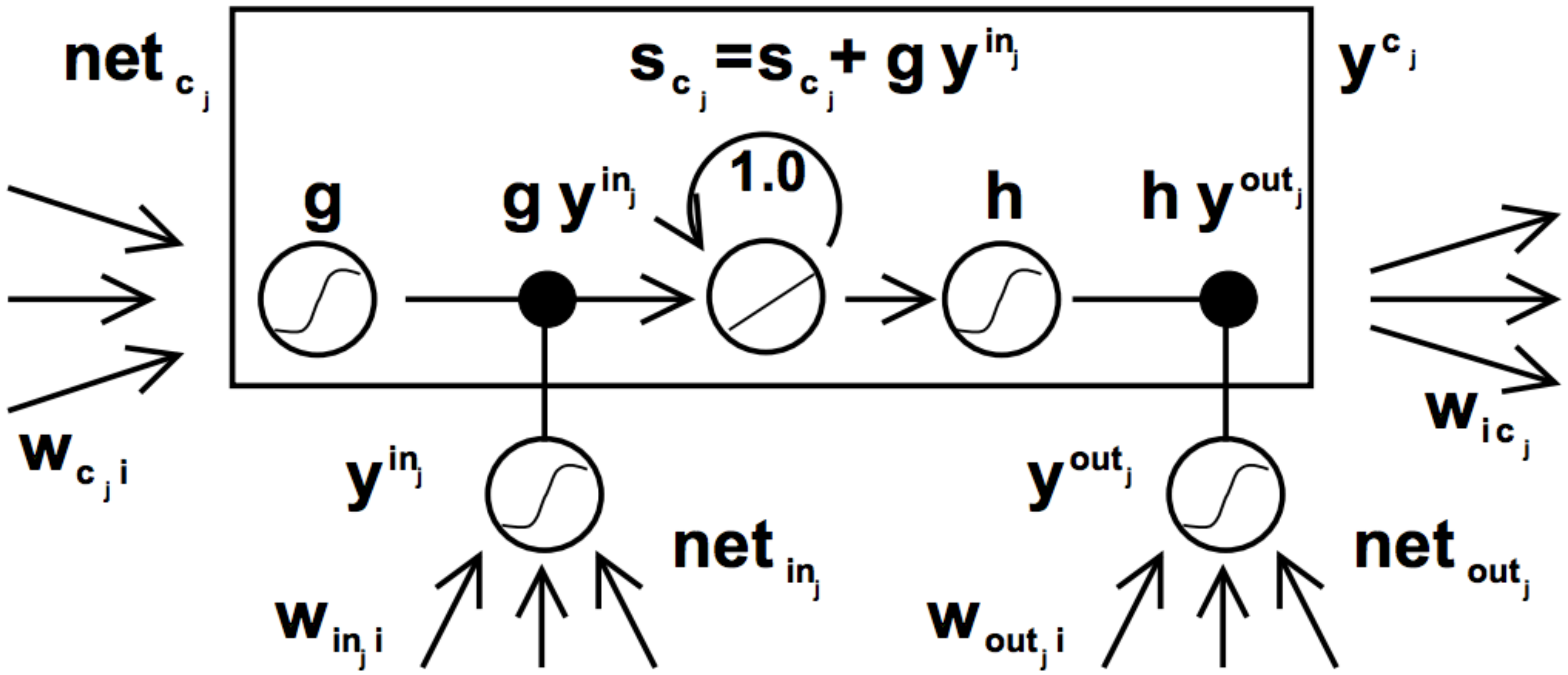


Figure 1: *Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the "constant error carrousel" CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.*

Gated Connections

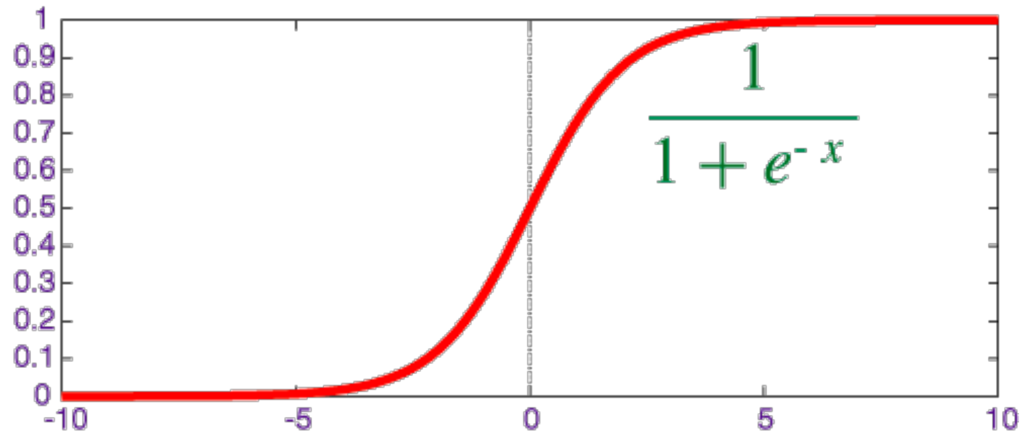
- Designed to fix “vanishing gradient” problem using *gates*

$$\mathbf{h}_t = \mathbf{h}_{t-1} \odot \mathbf{f} + \mathrm{func}(\mathbf{x}_t)$$

gated

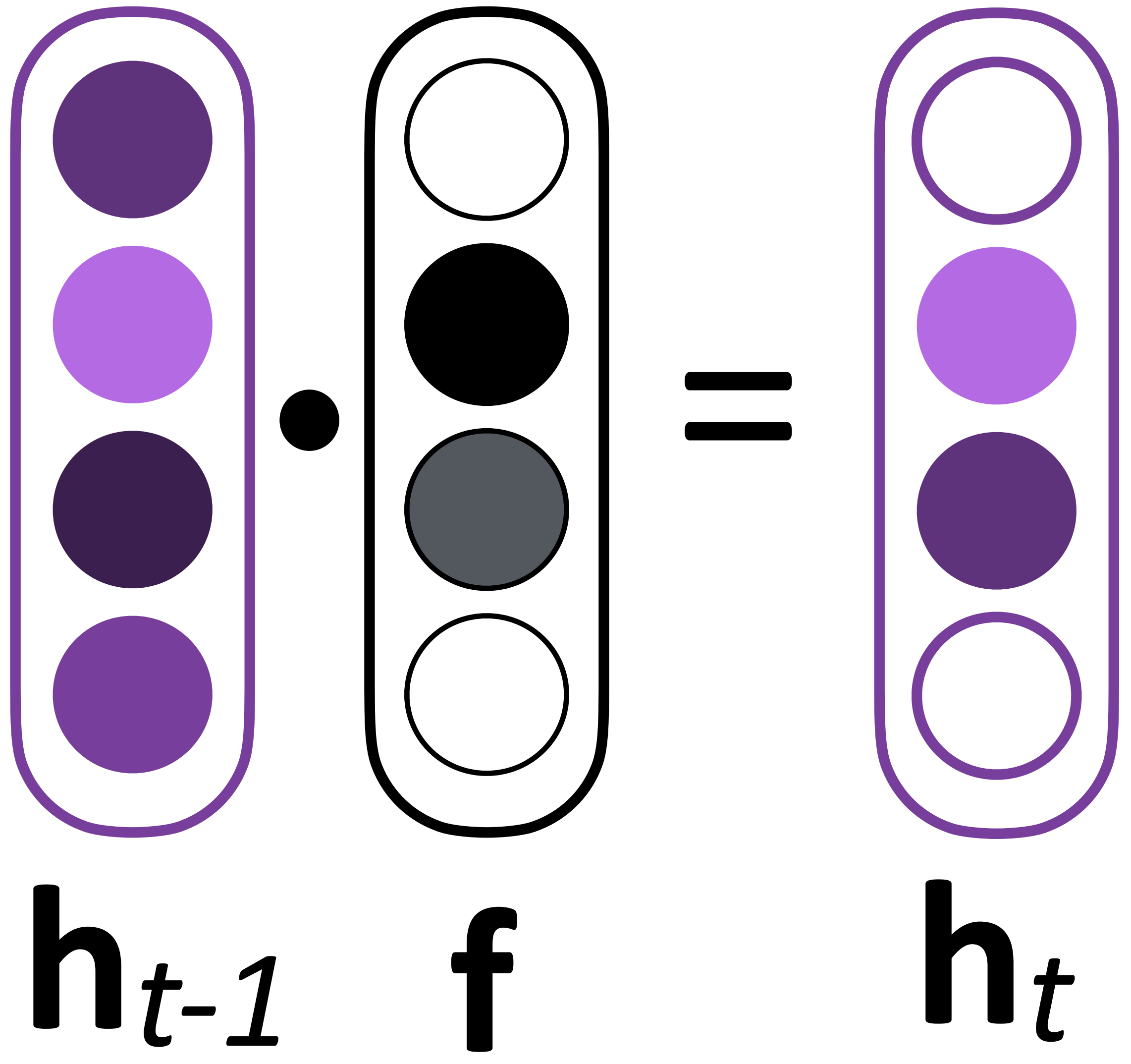
$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

Elman



- Vector-valued “forget gate” **f** computed based on input and previous hidden state

$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$



- Sigmoid: elements of **f** are in (0, 1)

- If **f** ≈ **1**, we simply sum up a function of all inputs — gradient doesn’t vanish!

LSTMs

- Long short-term memory network: hidden state as a “short-term” memory

- “Cell” **c** in addition to hidden state **h**

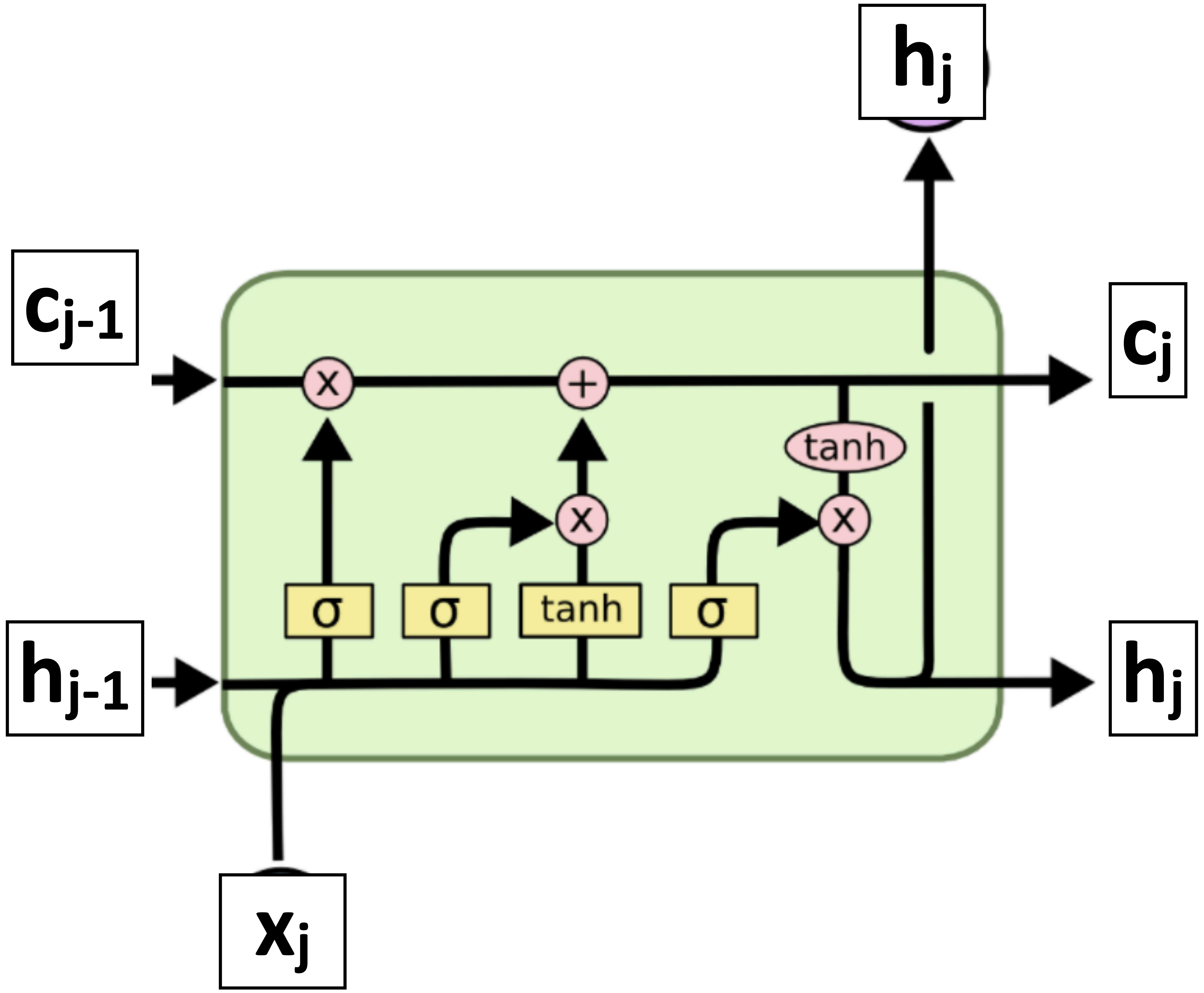
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- Vector-valued forget gate **f** depends on the **h** hidden state

$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$

- Basic communication flow: **x** -> **c** -> **h ->** output**,** each step of this process is gated in addition to gates from previous timesteps

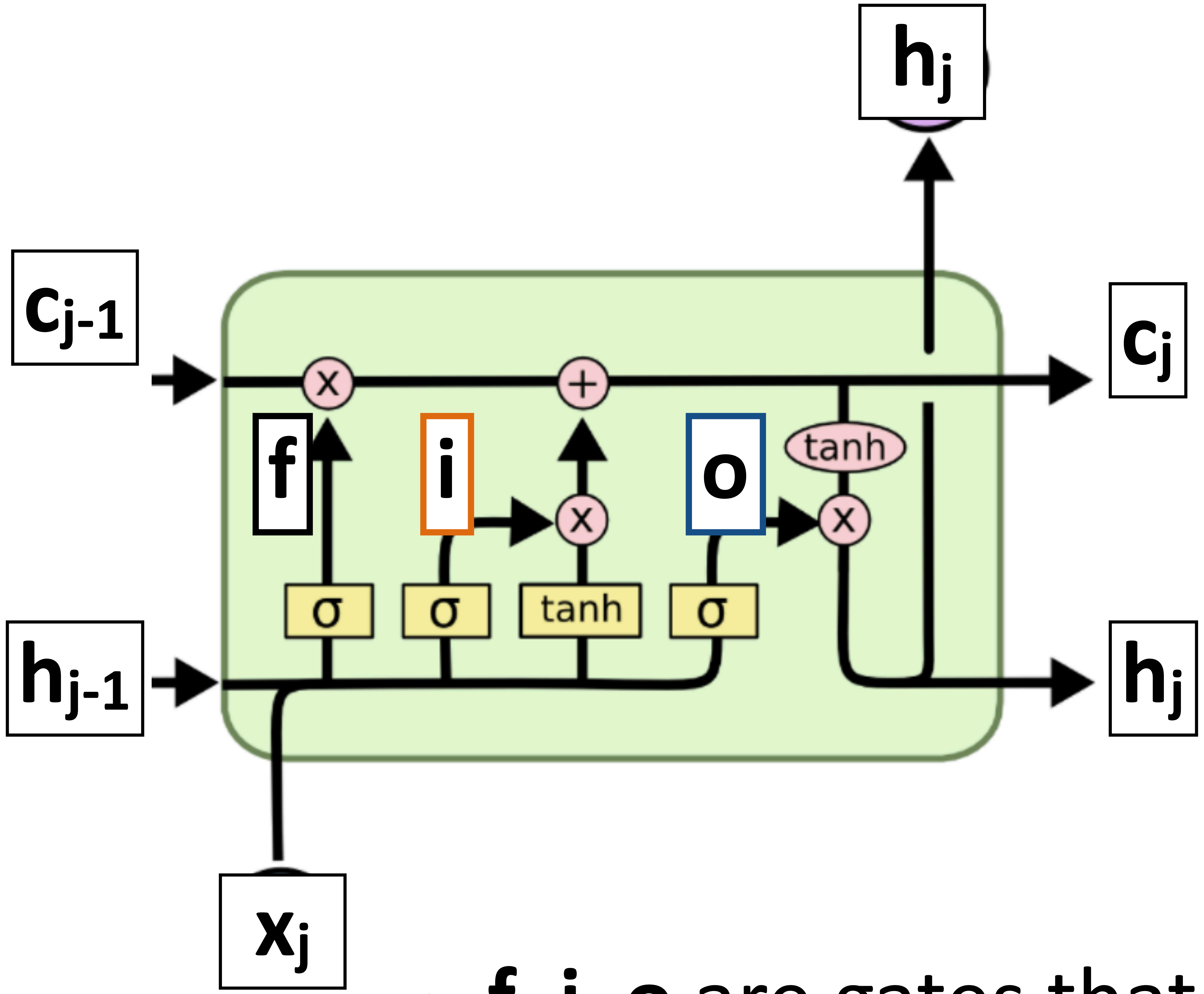
LSTMs



Hochreiter & Schmidhuber (1997)

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

LSTMs



$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\mathbf{i} = \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}})$$

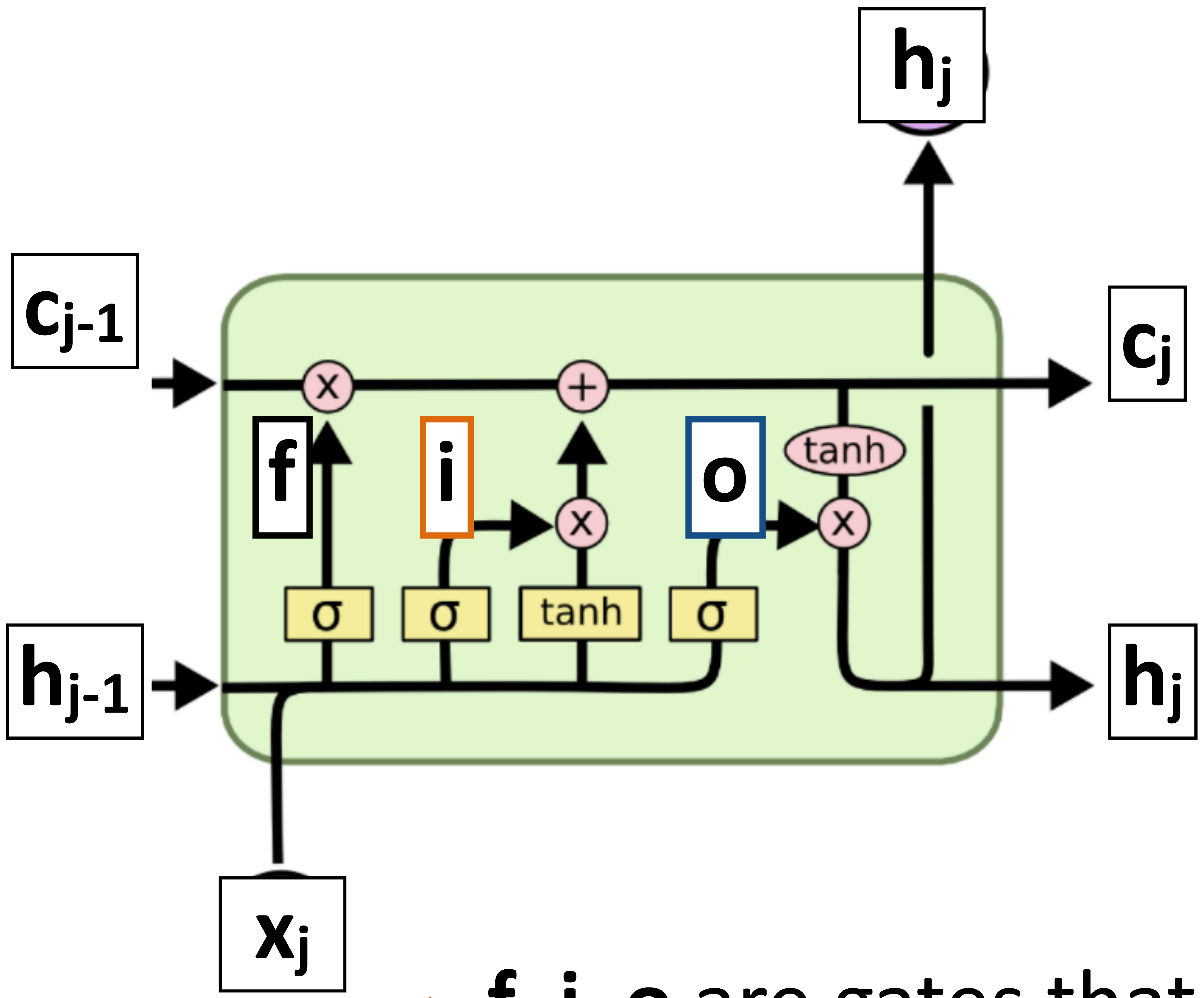
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **f**, **i**, **o** are gates that control information flow

Hochreiter & Schmidhuber (1997)

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

LSTMs



$$f = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

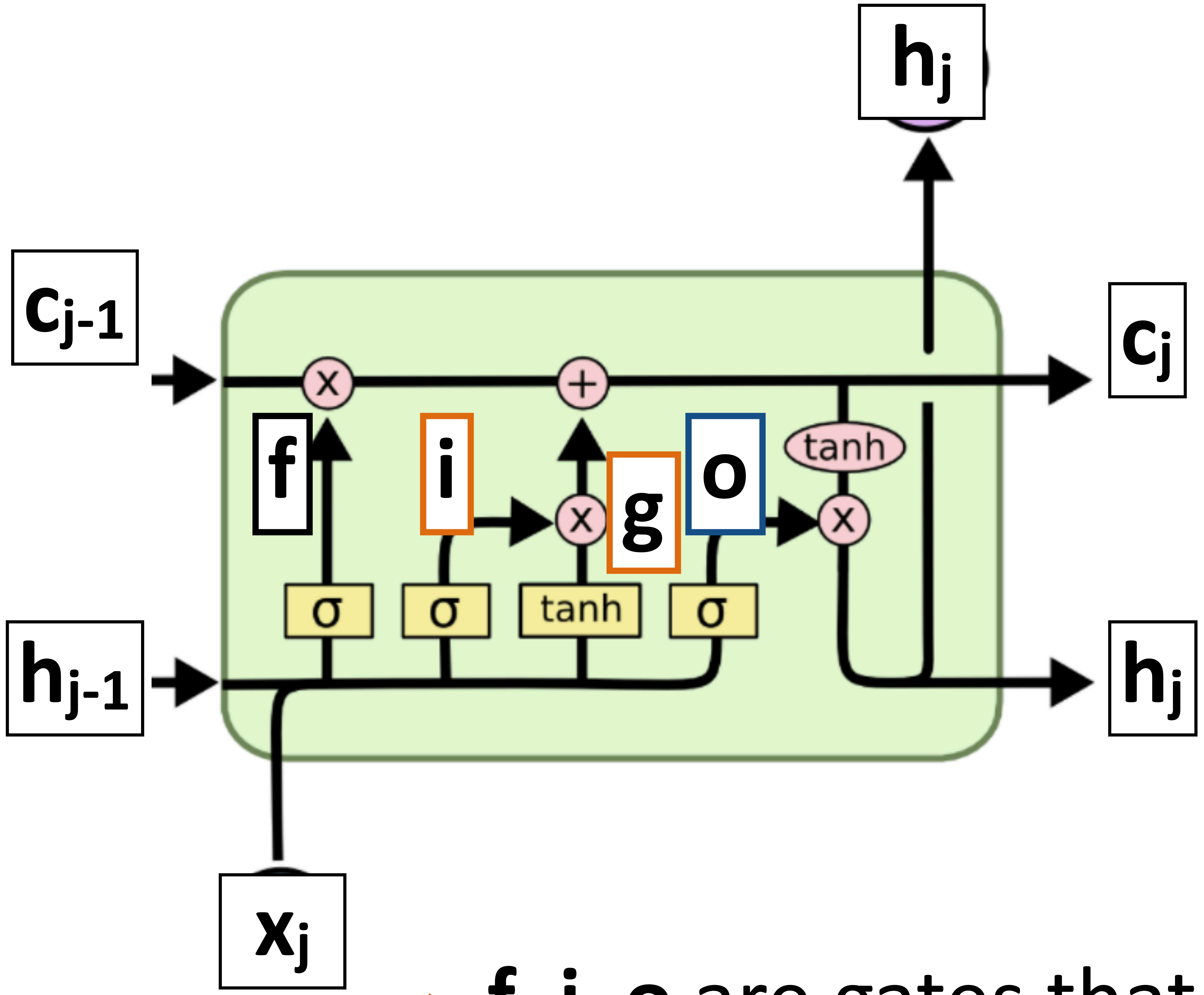
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **f**, **i**, **o** are gates that control information flow
- **g** reflects the main computation of the cell

Hochreiter & Schmidhuber (1997)

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

LSTMs



$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \boxed{\mathbf{g} \odot \mathbf{i}}$$

$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

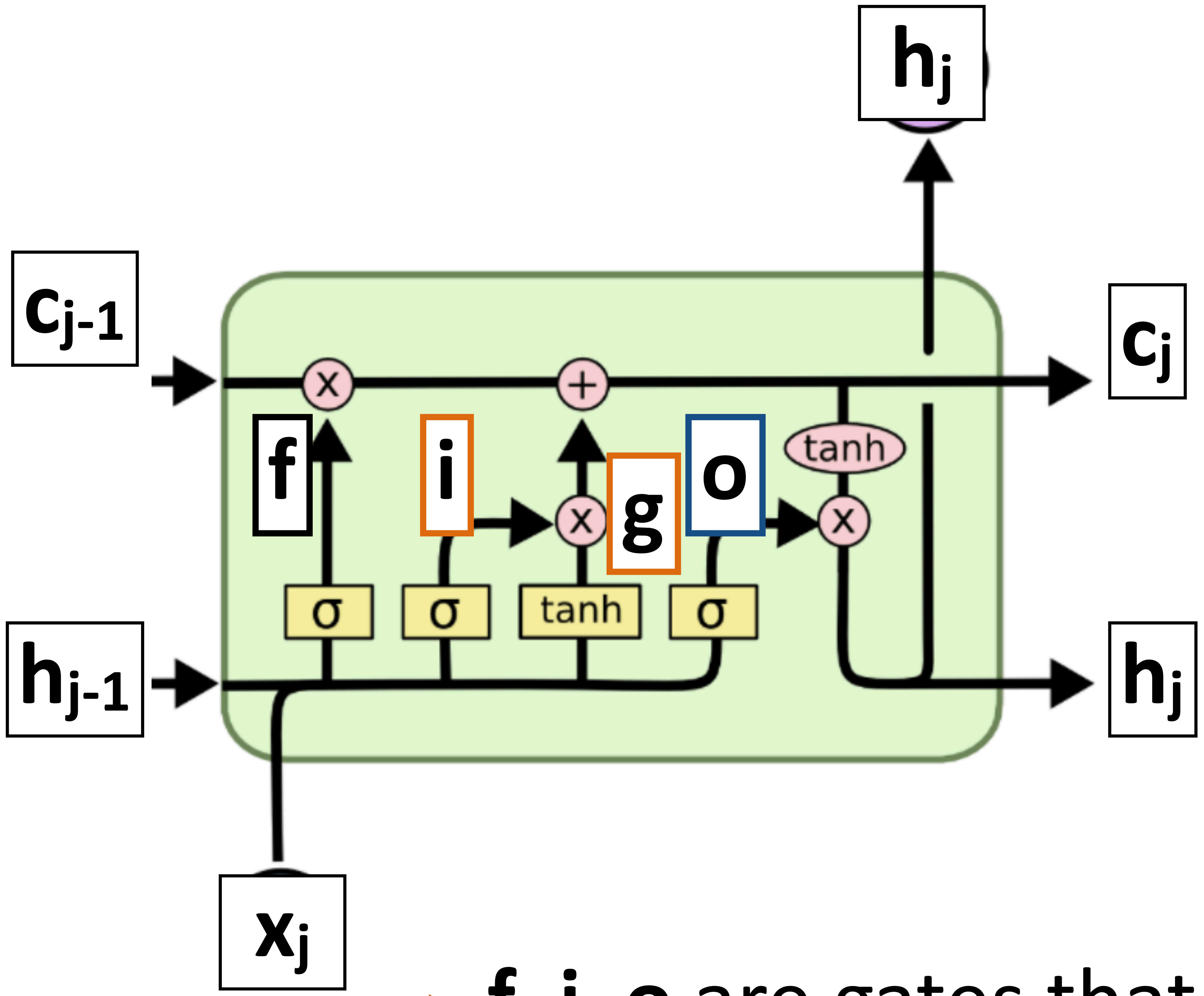
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **f**, **i**, **o** are gates that control information flow
- **g** reflects the main computation of the cell

Hochreiter & Schmidhuber (1997)

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

LSTMs



$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \boxed{\mathbf{g} \odot \mathbf{i}}$$

$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

$$\mathbf{h_j} = \tanh(\mathbf{c_j}) \odot \mathbf{o}$$

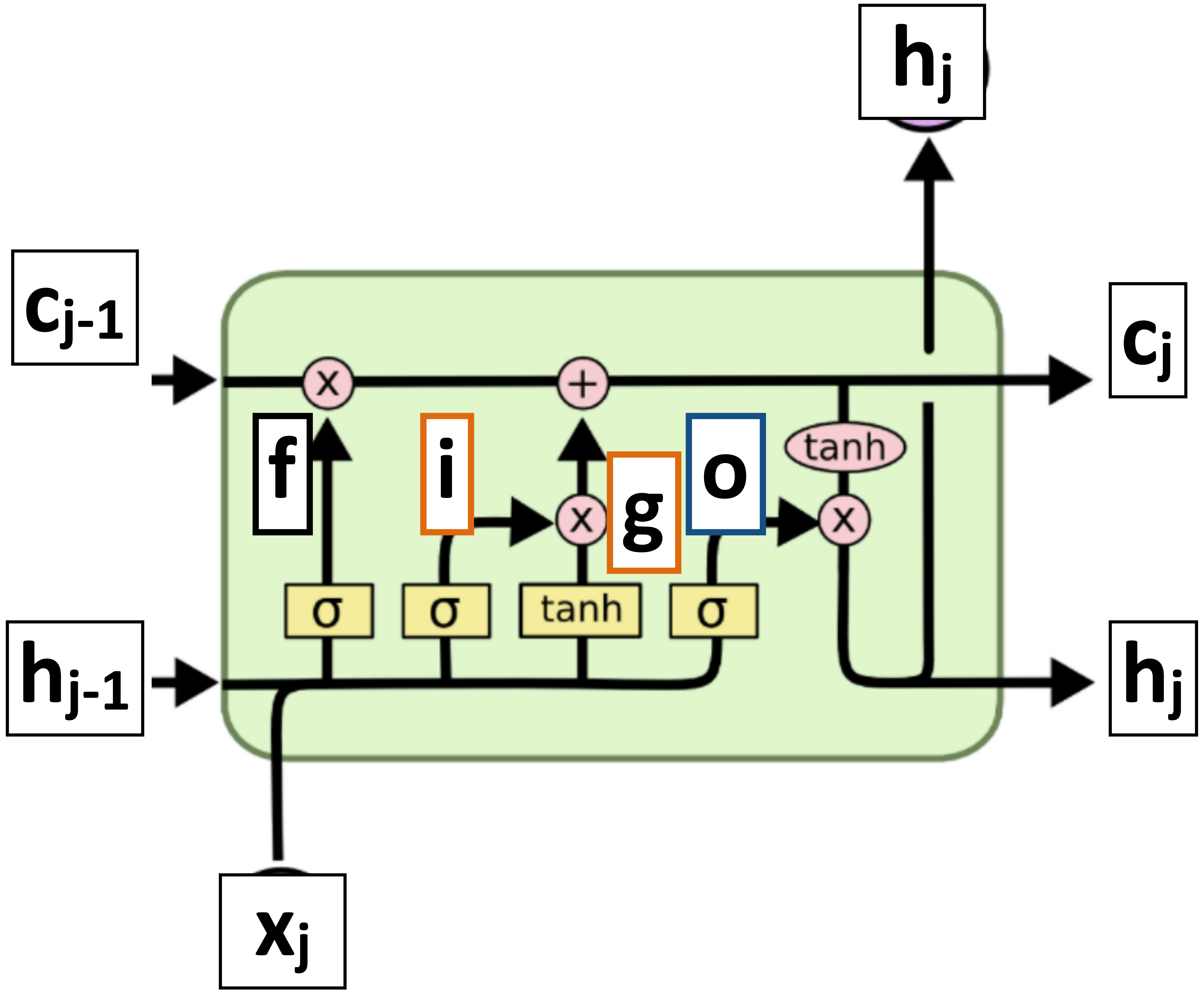
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **f**, **i**, **o** are gates that control information flow
- **g** reflects the main computation of the cell

Hochreiter & Schmidhuber (1997)

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

LSTMs



$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{f} = \sigma(\mathbf{x_j} \mathbf{W^{xf}} + \mathbf{h_{j-1}} \mathbf{W^{hf}})$$

$$\mathbf{g} = \tanh(\mathbf{x_j} \mathbf{W^{xg}} + \mathbf{h_{j-1}} \mathbf{W^{hg}})$$

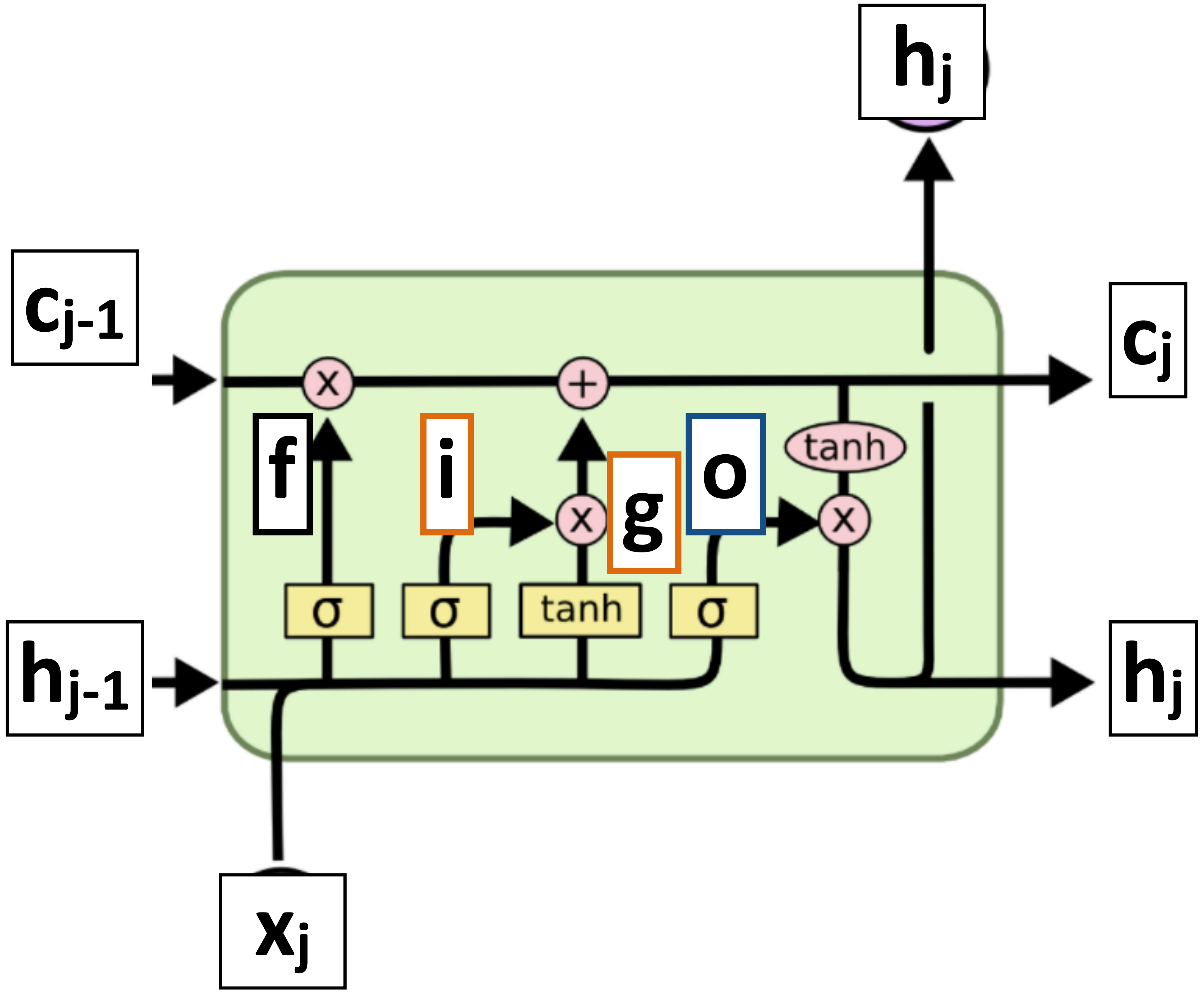
$$\mathbf{i} = \sigma(\mathbf{x_j} \mathbf{W^{xi}} + \mathbf{h_{j-1}} \mathbf{W^{hi}})$$

$$\mathbf{h_j} = \tanh(\mathbf{c_j}) \odot \mathbf{o}$$

$$\mathbf{o} = \sigma(\mathbf{x_j} \mathbf{W^{xo}} + \mathbf{h_{j-1}} \mathbf{W^{ho}})$$

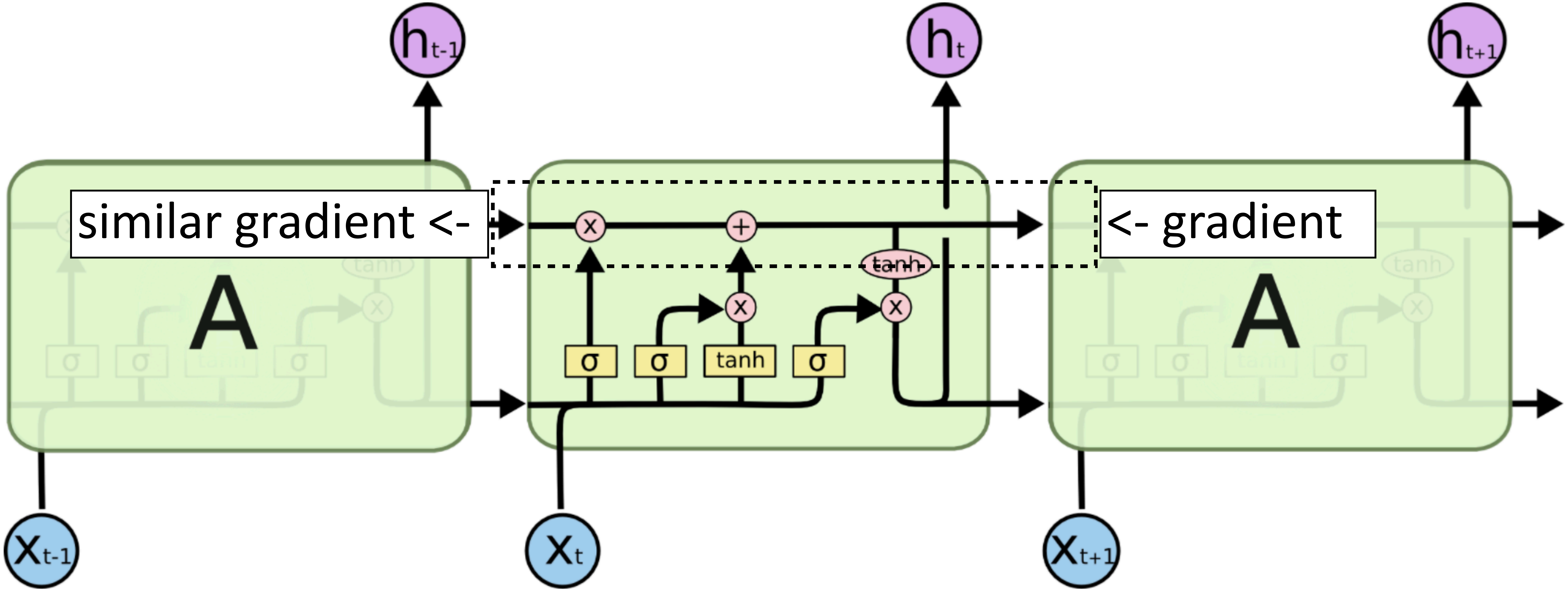
- Can we ignore the old value of **c** for this timestep?
- Can an LSTM sum up its inputs **x**?
- Can we ignore a particular input **x**?

LSTMs



- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

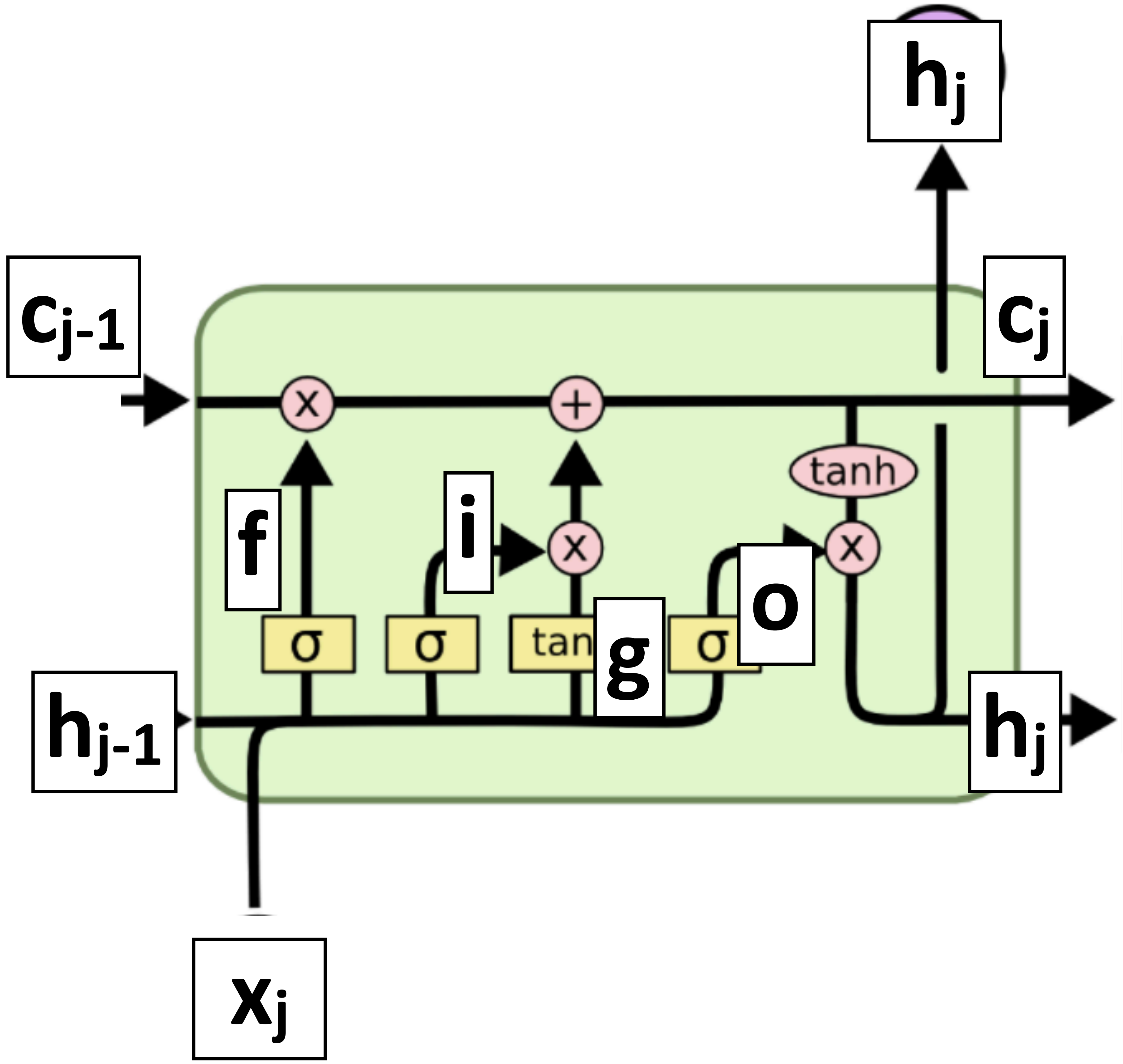
LSTMs

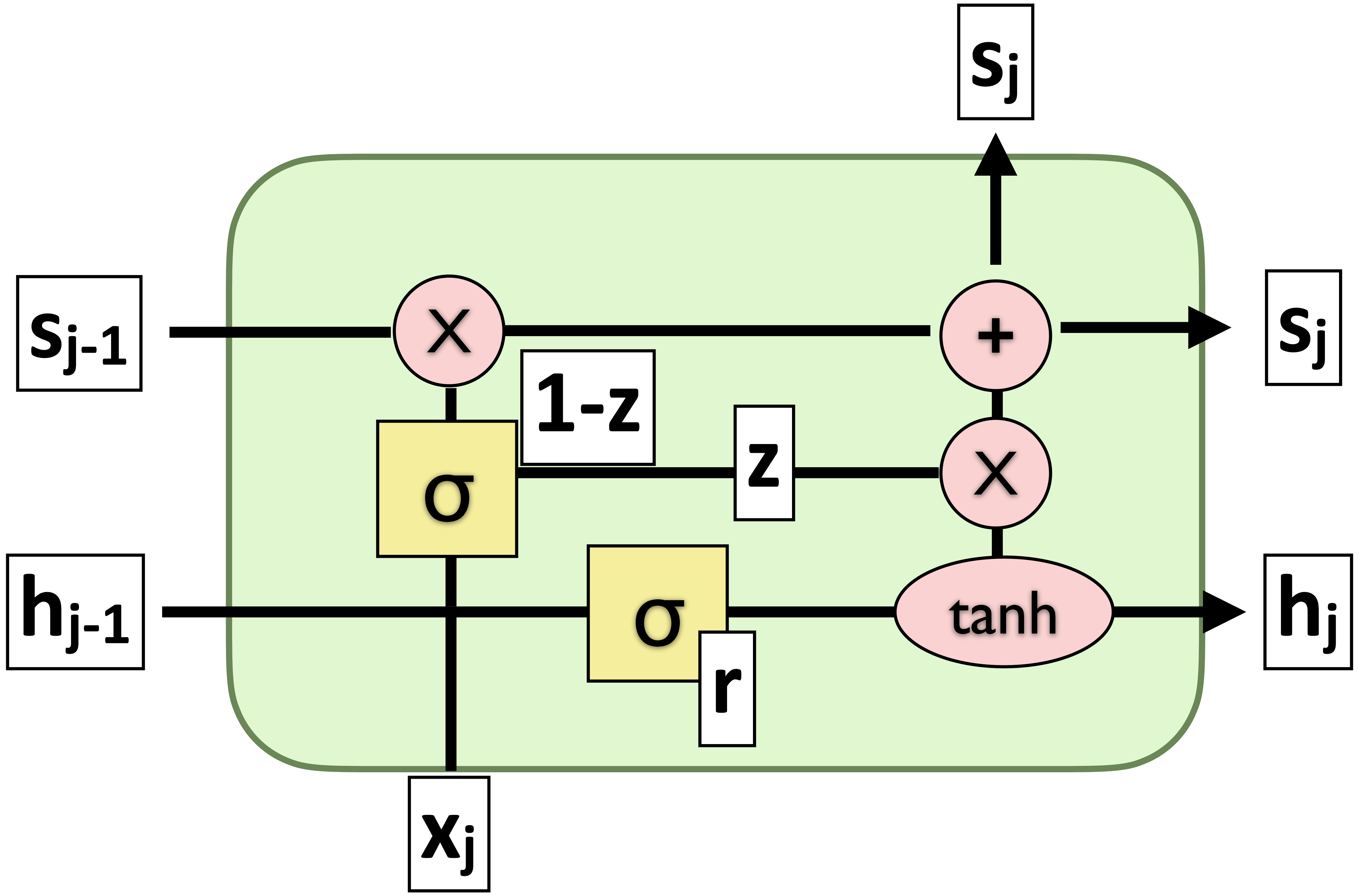


- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

"A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation" Gang Chen (2018)
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Units (GRUs)





- LSTM: more complex and slower, may work a bit better

- GRU: faster, a bit simpler
- Two gates: **z** (forget, mixes **s** and **h**) and **r** (mixes **h** and **x**)

GRUs

- Also solves the vanishing gradient problem, simpler than LSTM

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

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

- **z** controls mixing of hidden state **h** with new input **x**

- Faster to train and sometimes work better (most times not) than LSTMs

- Other variants of LSTMs:
  - multiplicative LSTMs, rotational unit of memory (RUM), …

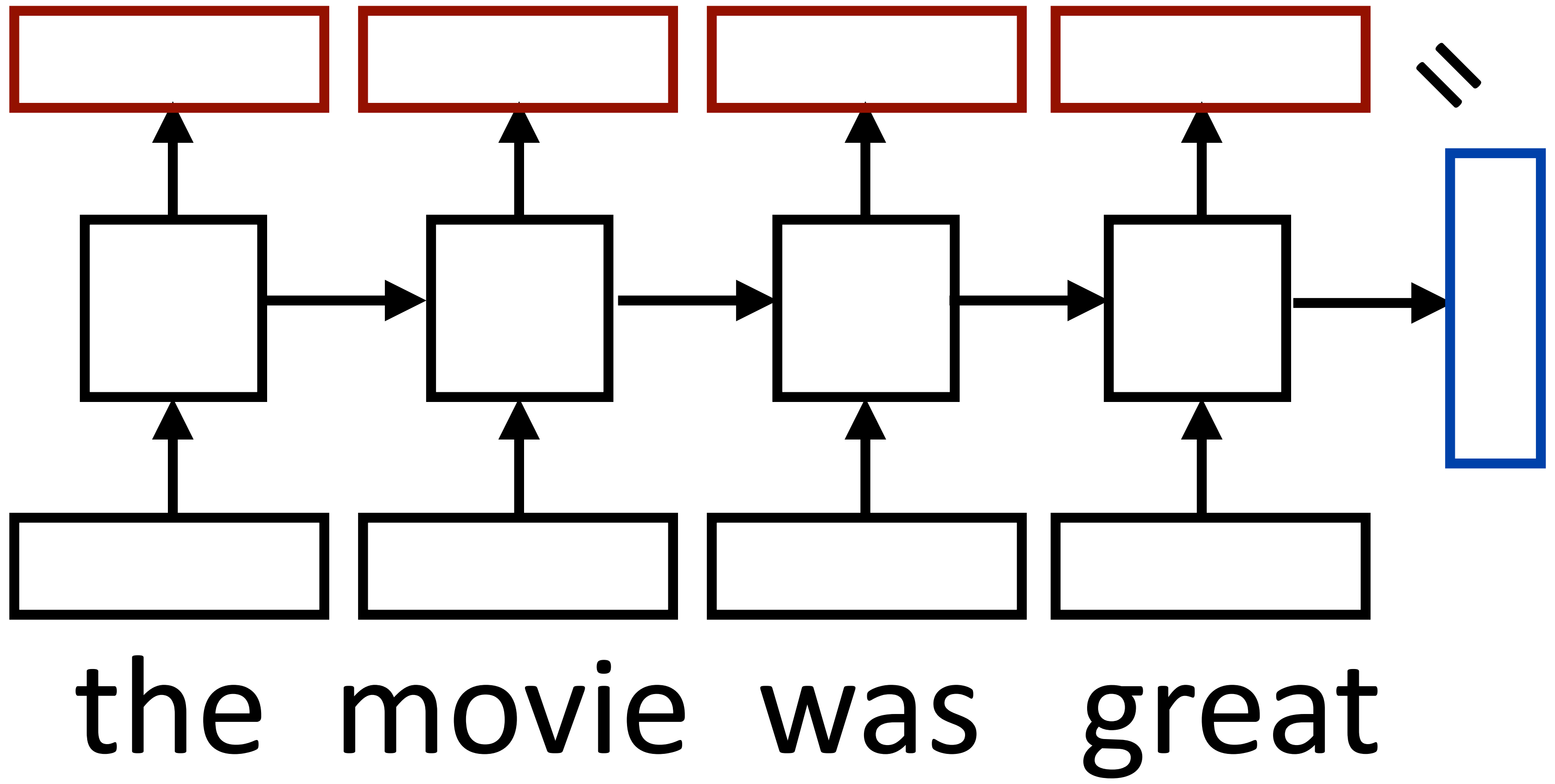
Cho et al. (2014)

Applications

What can LSTMs model?

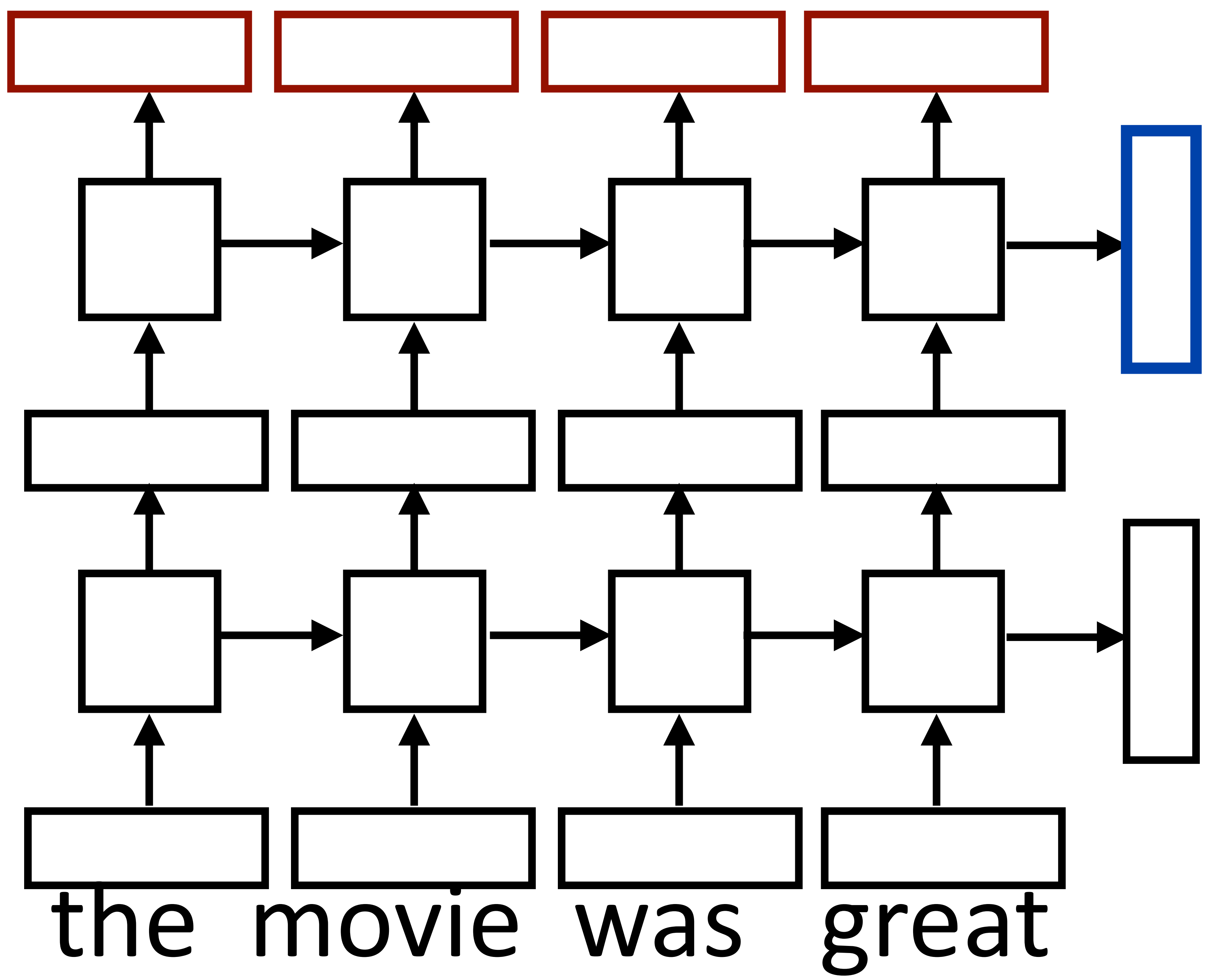
- Sentence classification (e.g., sentiment)
  - Encode one sentence, predict
- Sentence pair classification (e.g., paraphrase identification, NLI)
  - Encode two sentences, predict
- Sequential tagging (e.g., POS/NER), or Language models
  - Move left-to-right, per-token prediction
- Translation/Generation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)

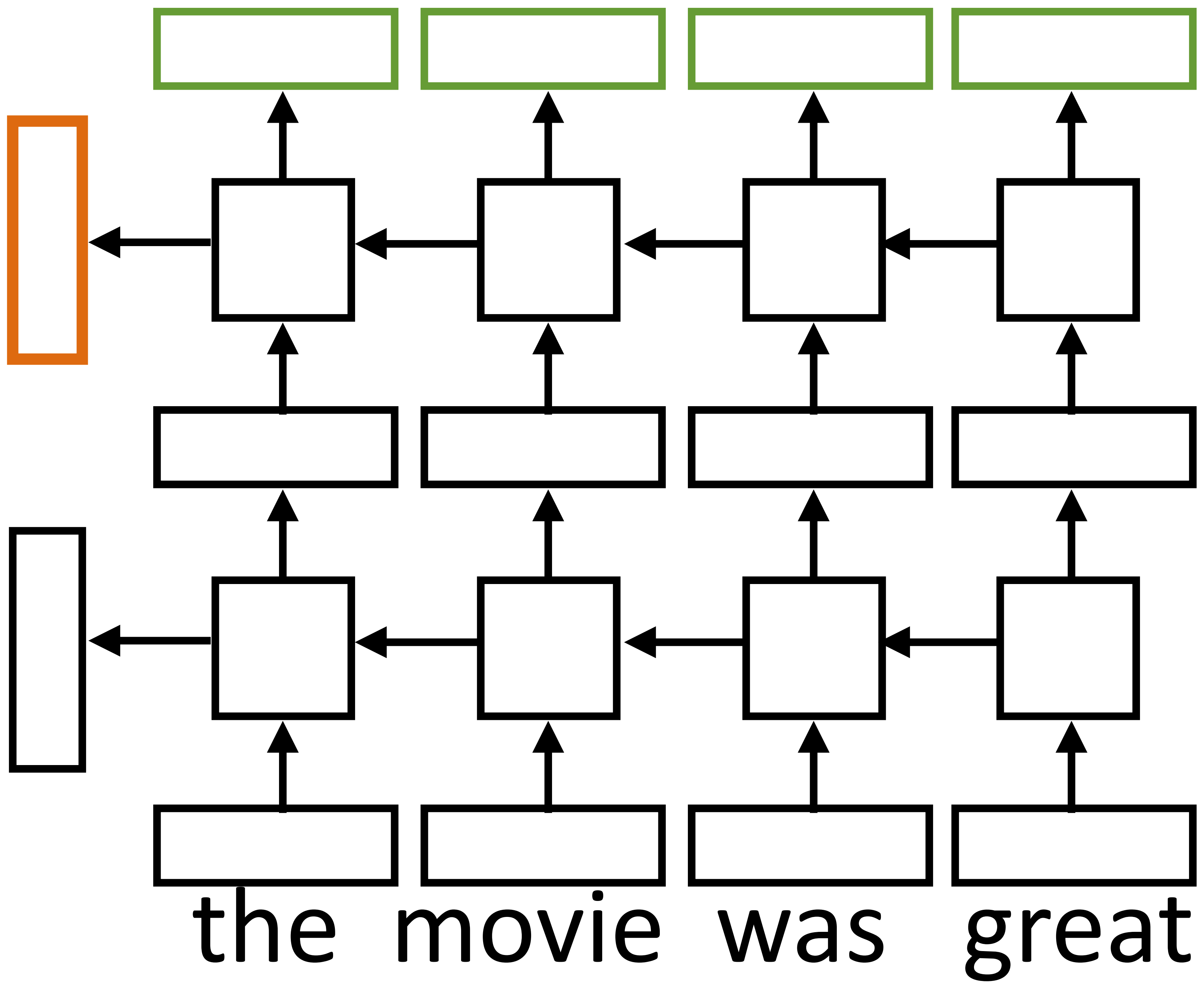
What do RNNs produce?



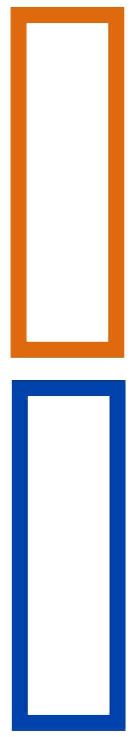
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

Multilayer Bidirectional RNN

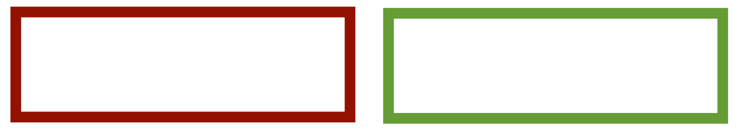




- Sentence classification based on concatenation of both final outputs



- Token classification based on concatenation of both directions' token representations



Natural Language Inference

Premise		Hypothesis
A boy plays in the snow	*entails*	A boy is outside
A man inspects the uniform of a figure	*contradicts*	The man is sleeping
An older and younger man smiling	*neutral*	Two men are smiling and laughing at cats playing

- Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)

- Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)

SNLI Dataset

- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process

100D LSTM: 78% accuracy

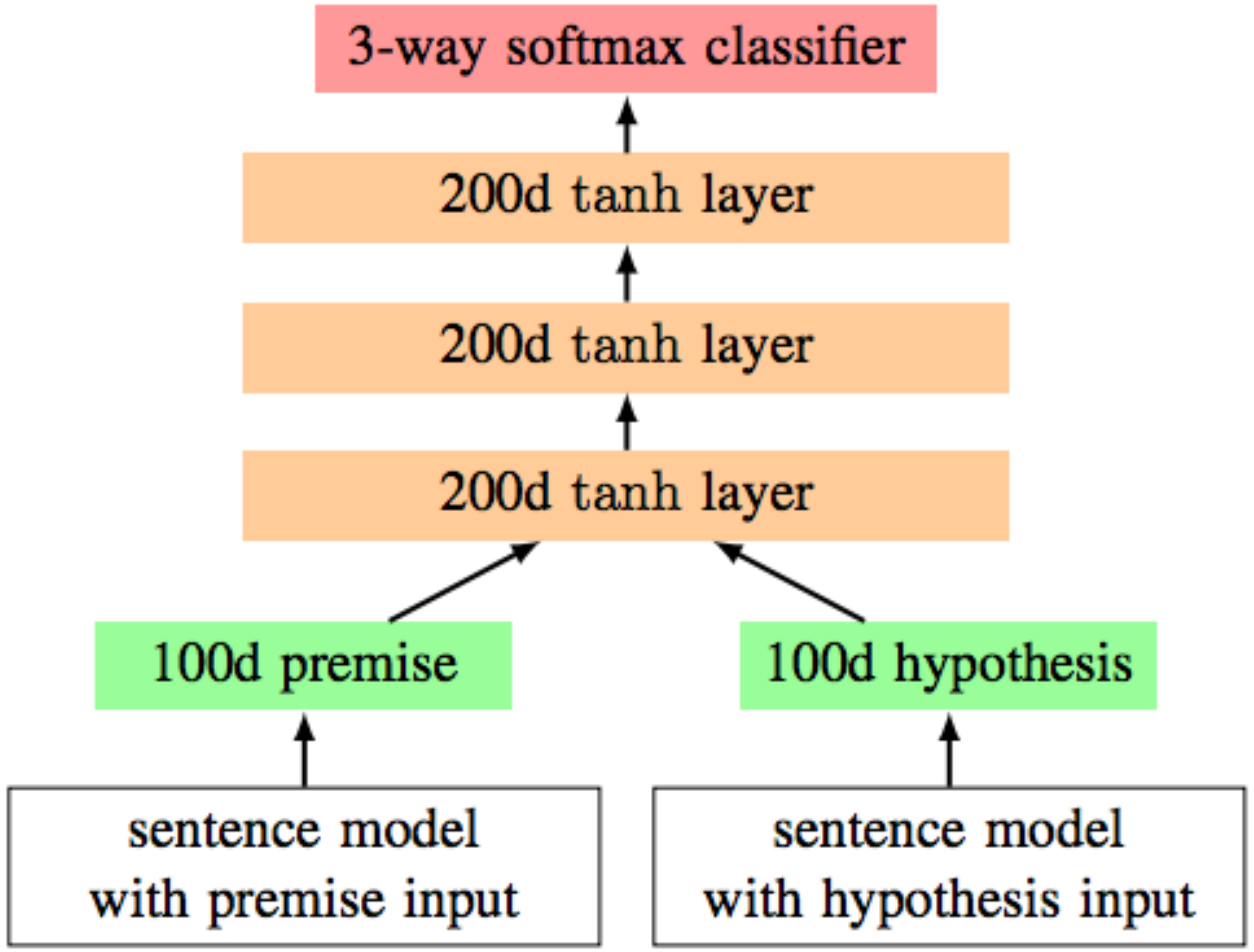
300D LSTM: 80% accuracy

(Bowman et al., 2016)

300D BiLSTM: 83% accuracy

(Liu et al., 2016)

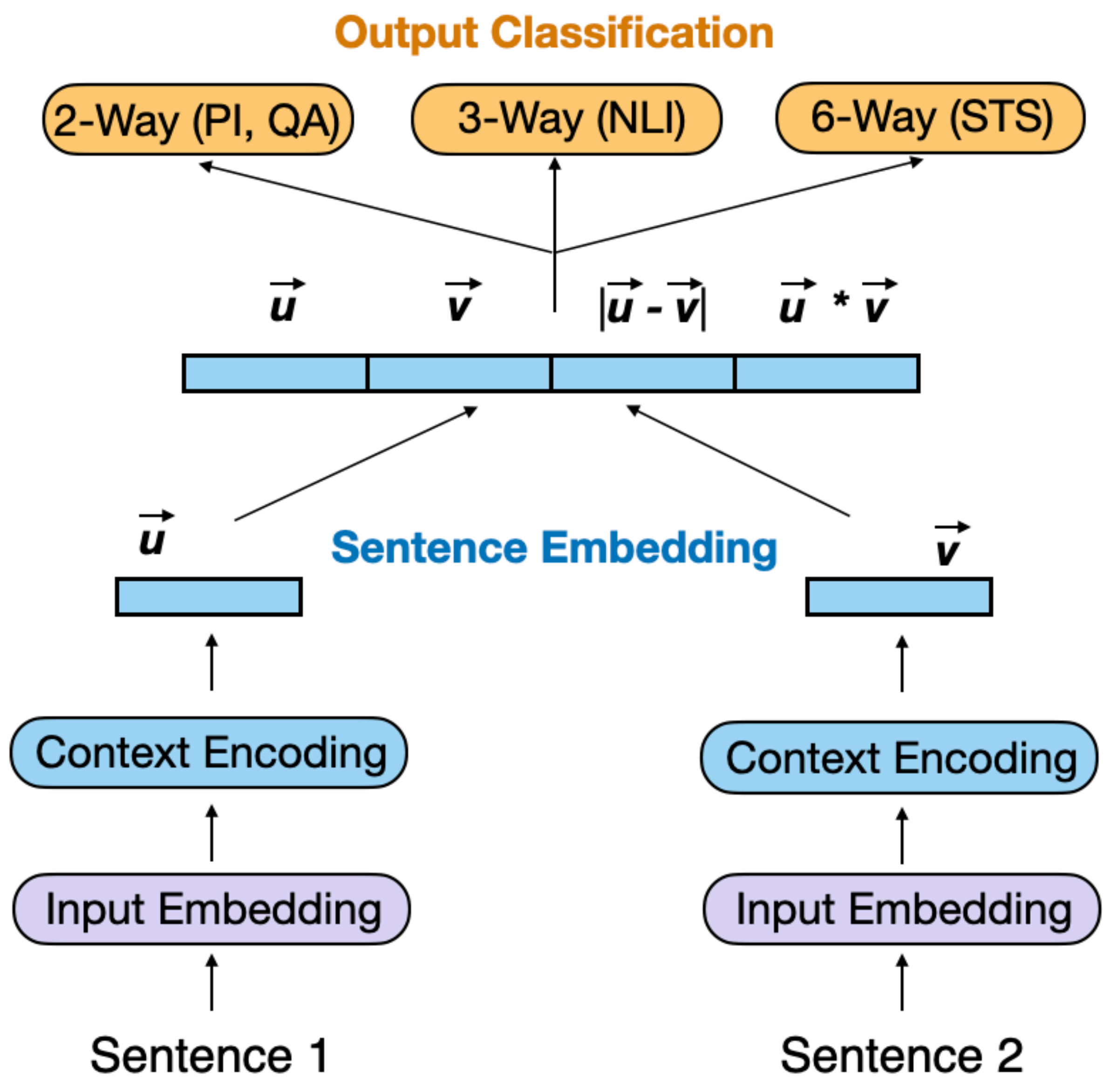
- Later: better models for this



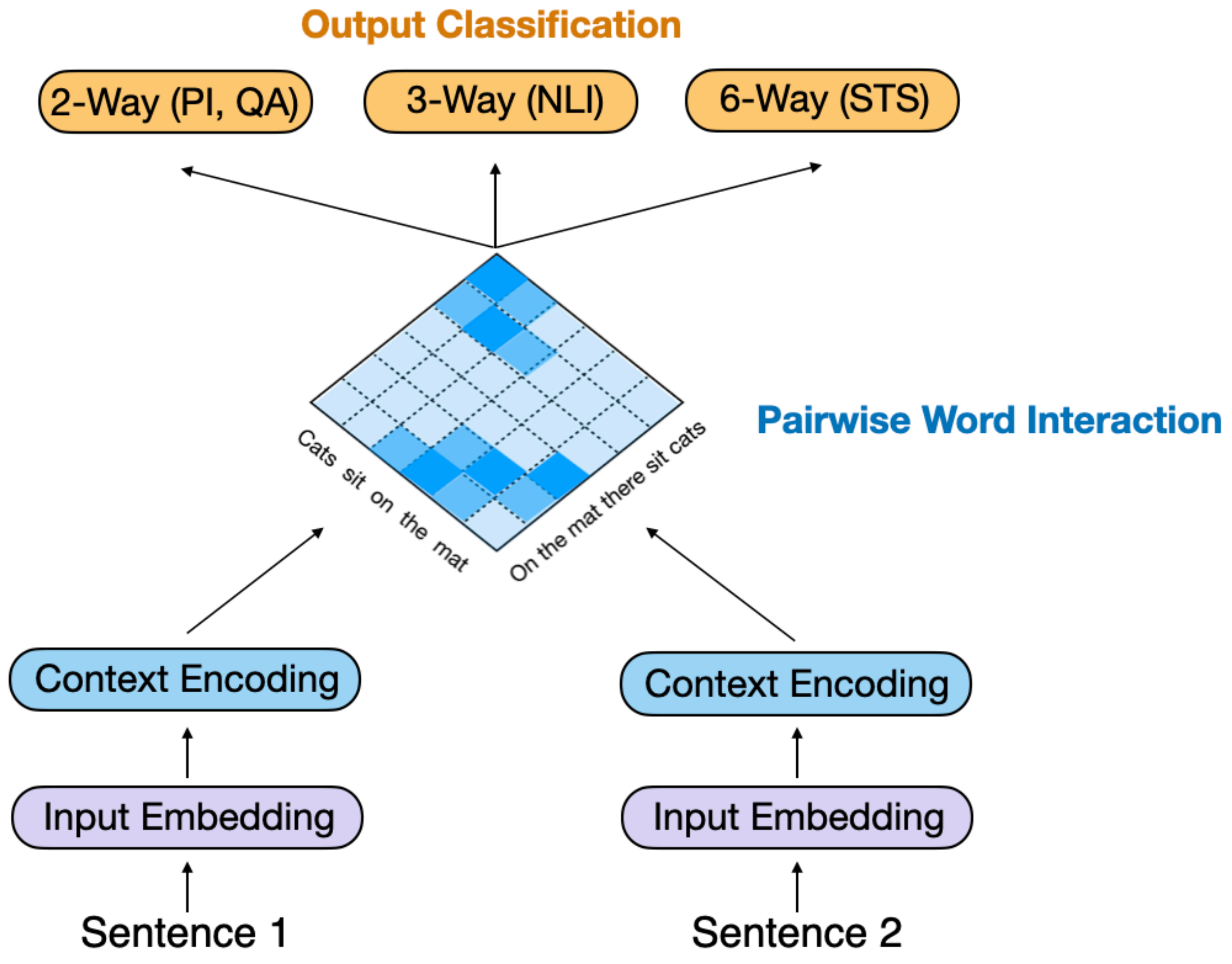
Bowman et al. (2015)

Sentence Pair Classification

Type I: Sentence Encoding-based Models



Type II: Word Interaction-based Models



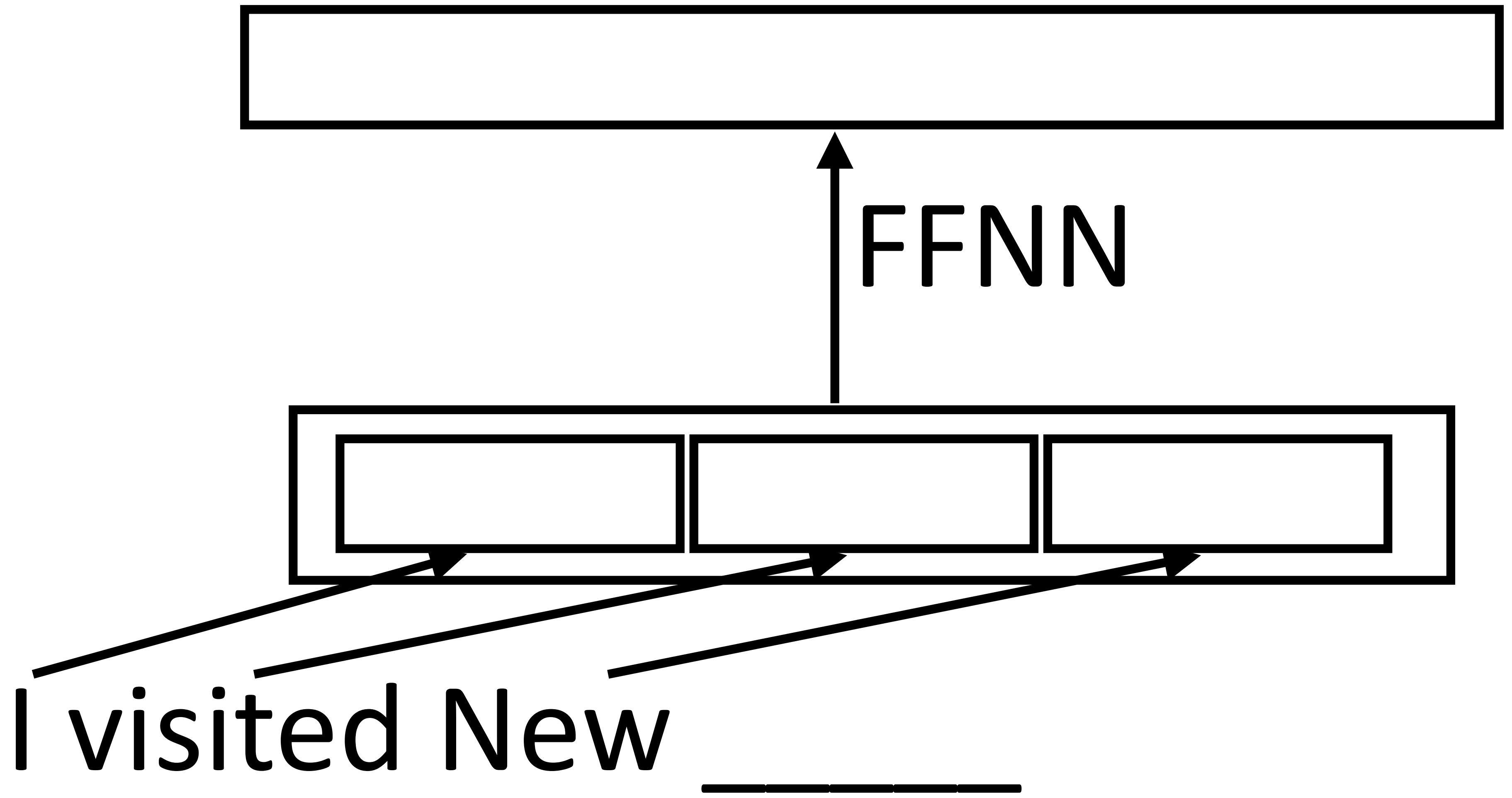
- semantic relation between two sentences depends largely on aligned words/phrases

Wuwei Lan, Wei Xu. "Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering" (COLING 2018)

RNN Language Modeling

Neural Language Models

- Early work: feedforward neural networks looking at context



FFNN

I visited New ______

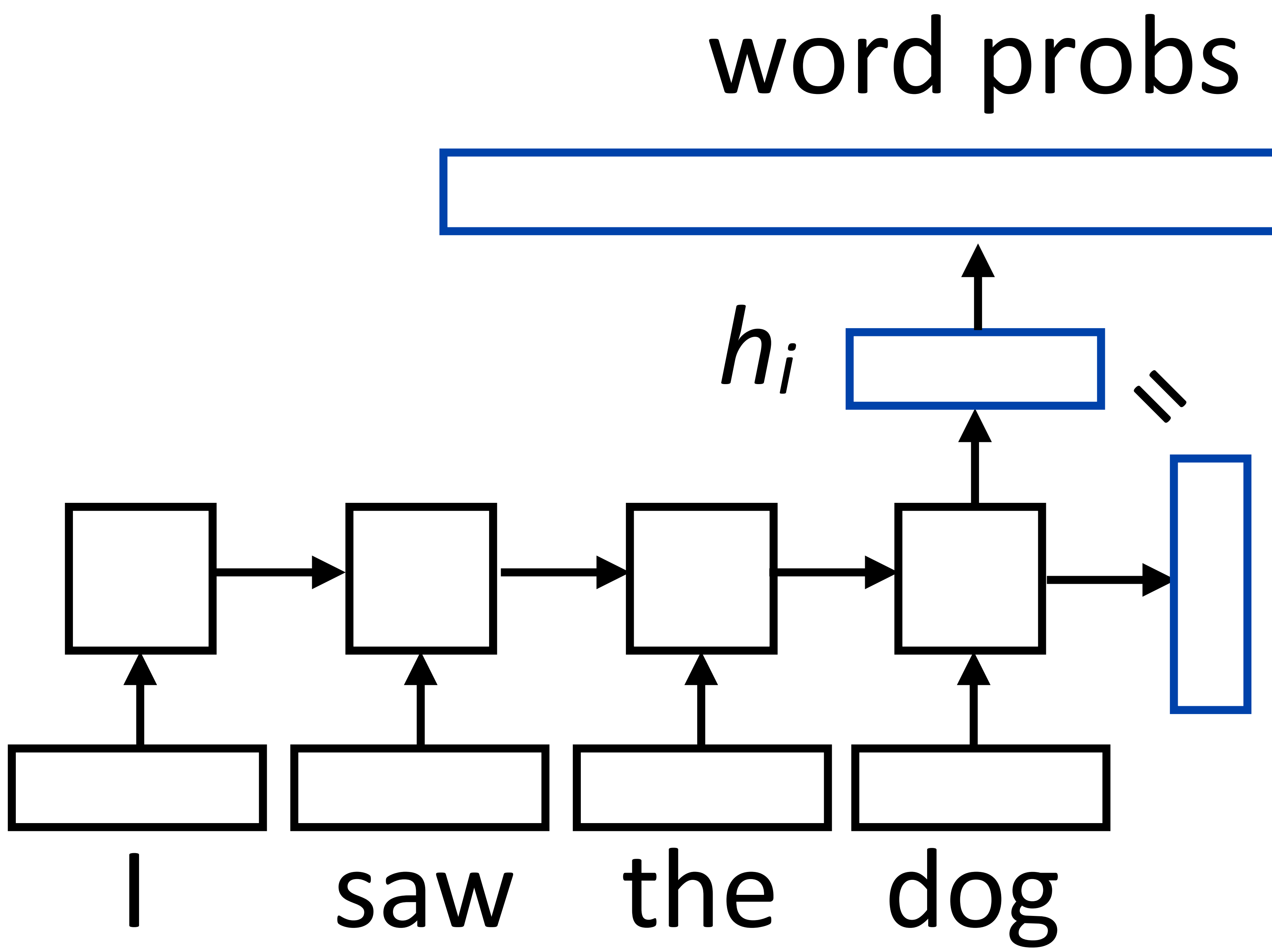
$$P(w_i | w_{i-n}, \ldots, w_{i-1})$$

- Slow to train over lots of data!

- Still only look at a fixed window of information…can we use more?

Bengio (2003), Mnih and Hinton (2003)

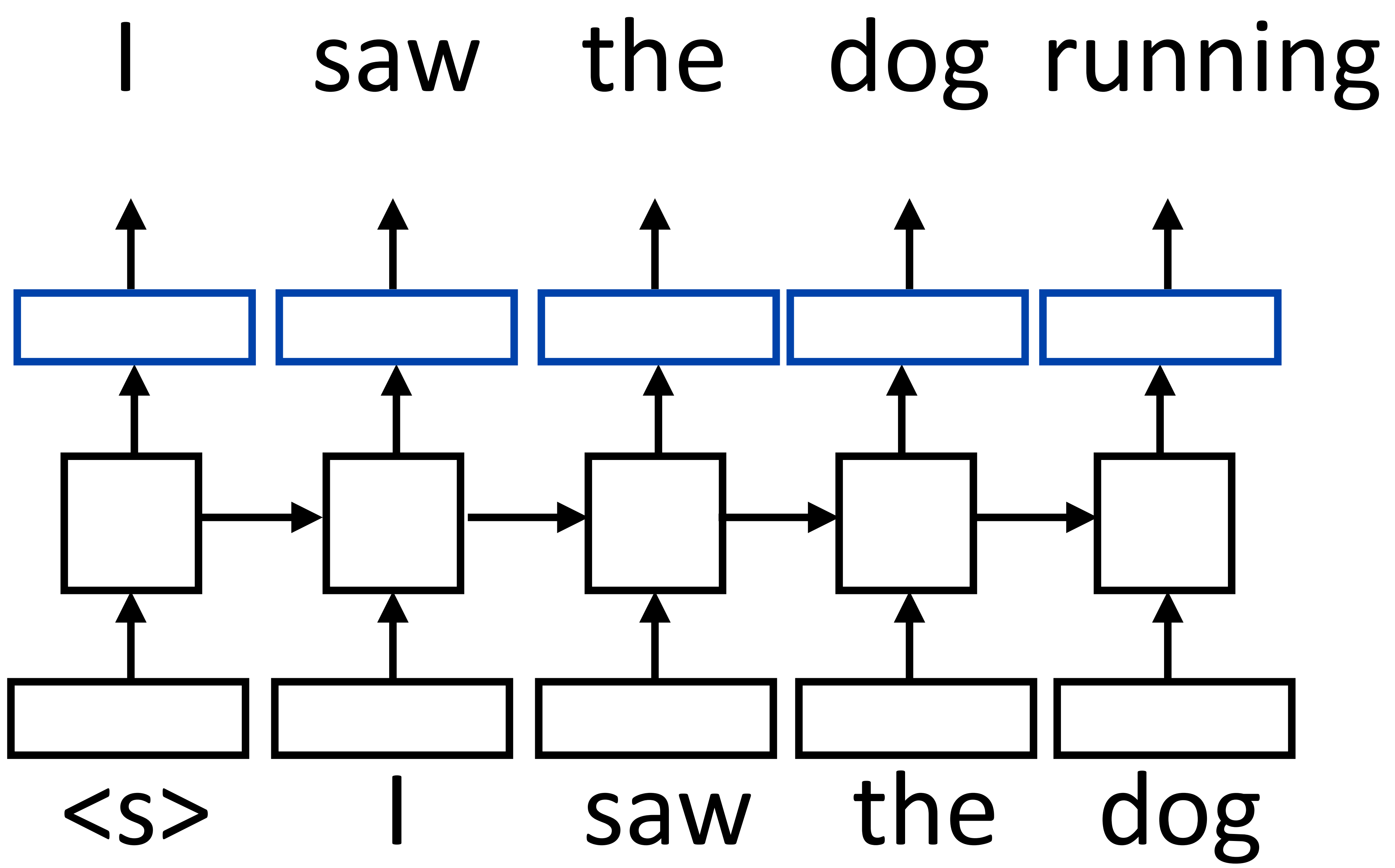
RNN Language Modeling



$$P(w|\text{context}) = \text{softmax}(W\mathbf{h}_i)$$

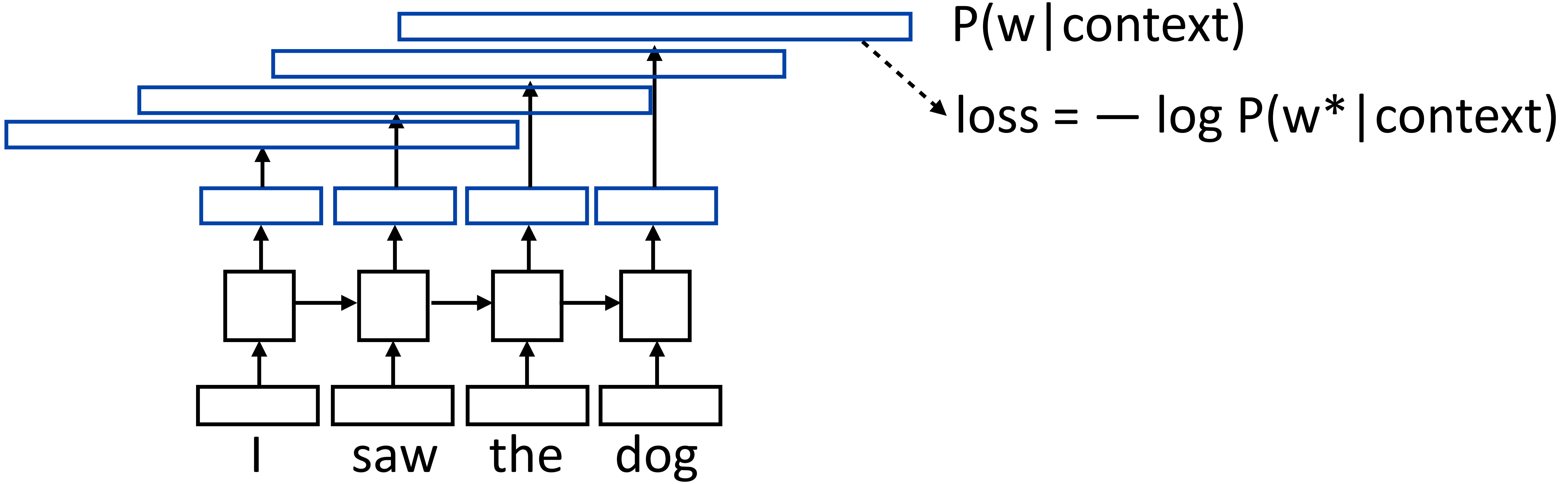
- *W* is a (vocab size) x (hidden size) matrix

Training RNNLMs



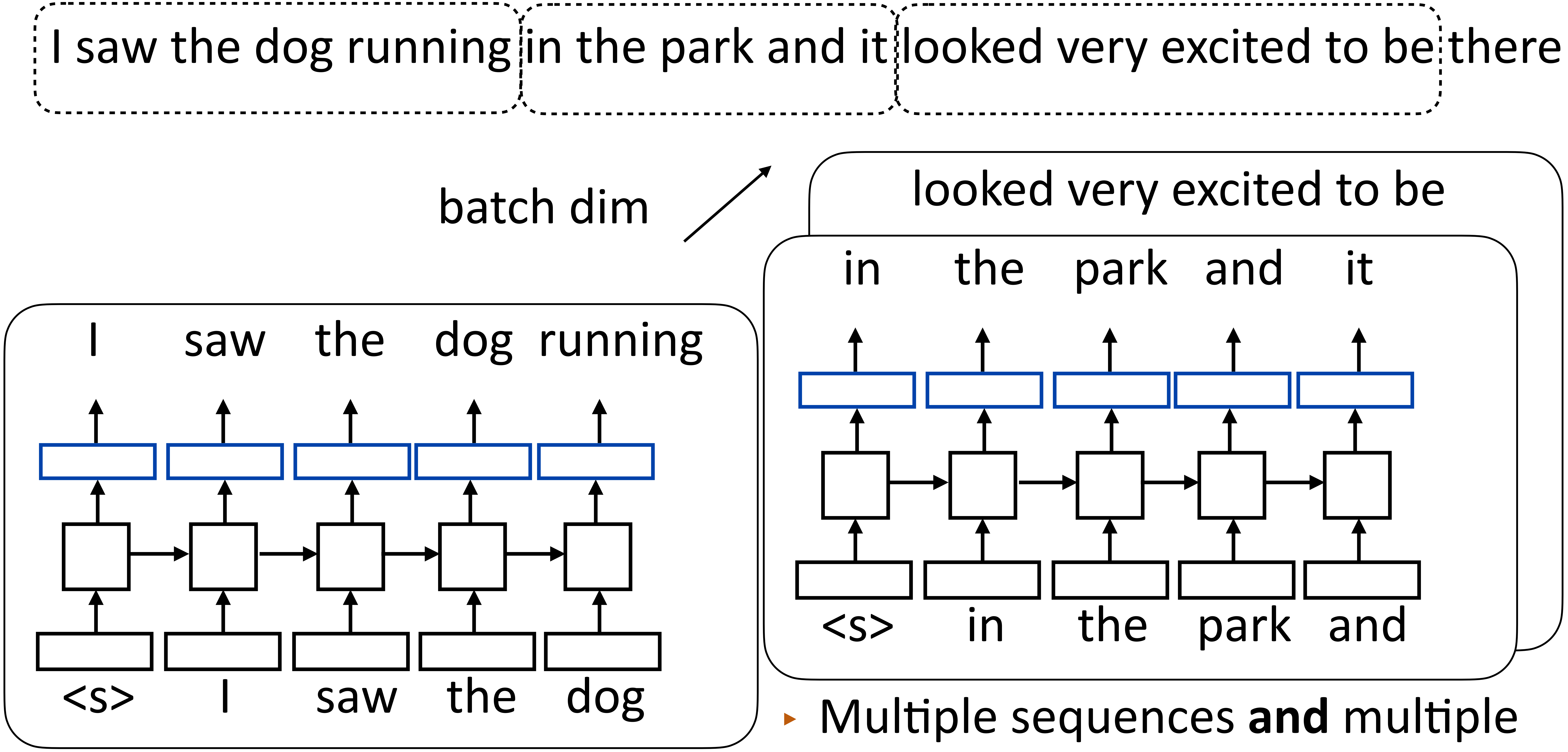
- Input is a sequence of words, output is those words shifted by one,
- Allows us to efficiently batch up training across time (one run of the RNN)

Training RNNLMs



- Total loss = sum of negative log likelihoods at each position
- Backpropagate through the network to simultaneously learn to predict next word given previous words at all positions

Batched LM Training

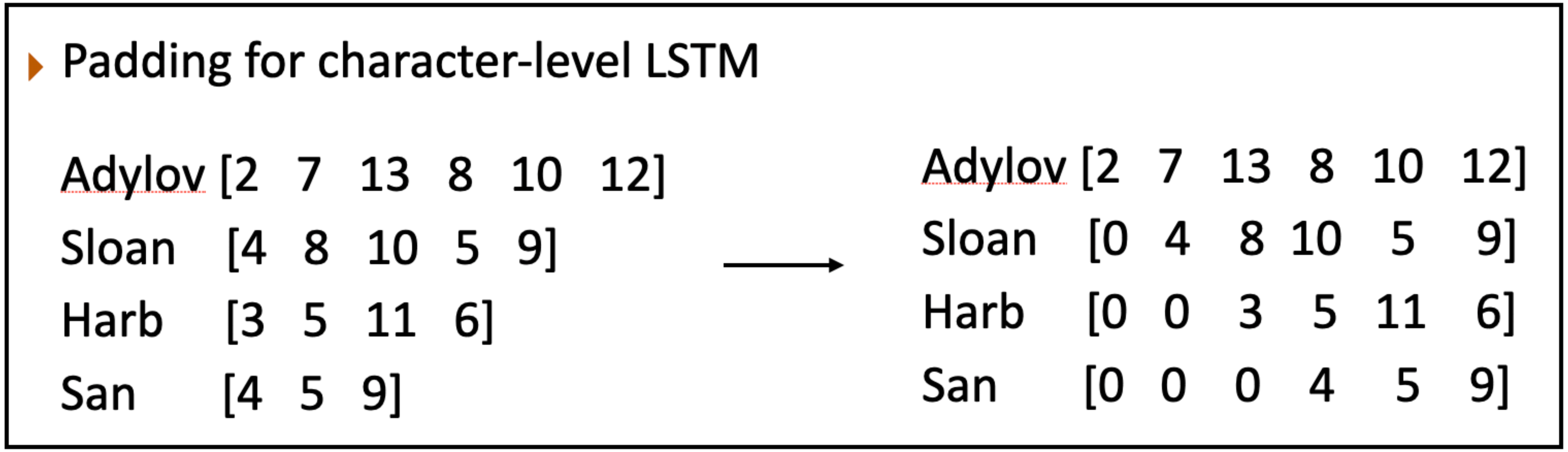


- Multiple sequences **and** multiple timestamps per sequence

Padding

- Prepending or appending zeros

- To create batches of equal length for faster training time



- Padding for character-level LSTM

Adylov	[2 7 13 8 10 12]	→	Adylov	[2 7 13 8 10 12]
Sloan	[4 8 10 5 9]		Sloan	[0 4 8 10 5 9]
Harb	[3 5 11 6]		Harb	[0 0 3 5 11 6]
San	[4 5 9]		San	[0 0 0 4 5 9]

LM Evaluation

- Accuracy doesn't make sense — predicting the next word is generally impossible so accuracy values would be very low

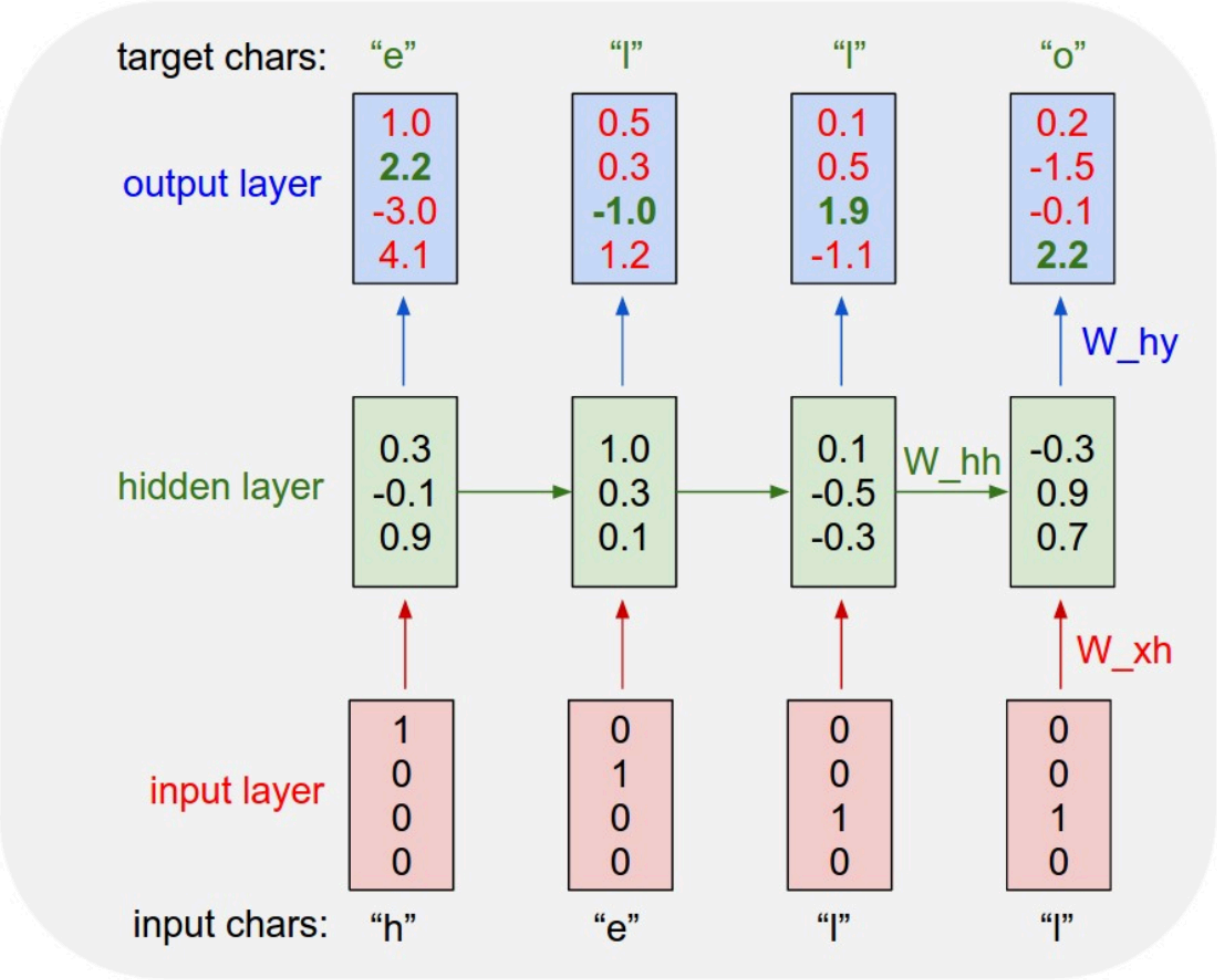
- Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

$$\frac{1}{n}\sum_{i=1}^{n}\log P(w_i|w_1,\ldots,w_{i-1})$$

- Perplexity: exp(average negative log likelihood). Lower is better
  - Suppose we have probs 1/4, 1/3, 1/4, 1/3 for 4 predictions
  - Avg NLL (base e) = 1.242   Perplexity = 3.464 ← geometric mean of denominators

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code



An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

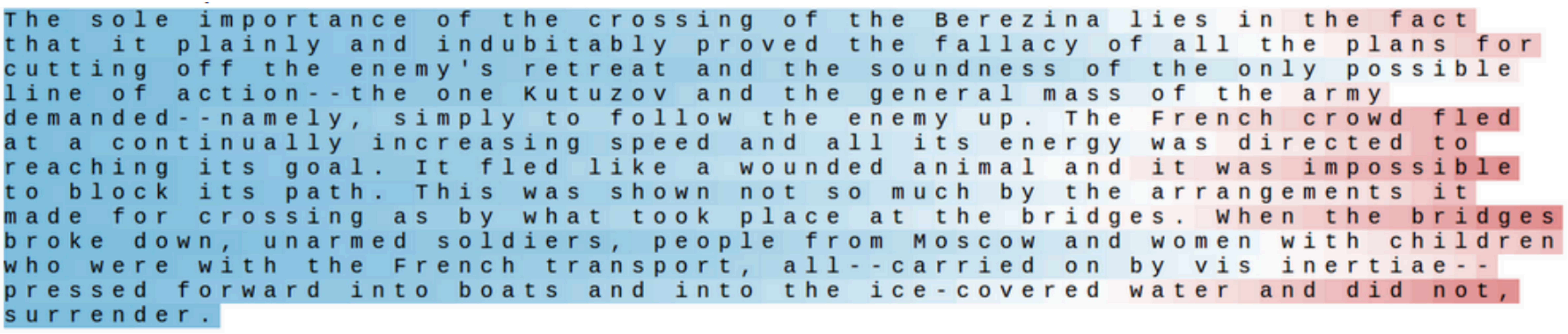
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

- Visualize activations of specific cells (components of **c**) to understand them

- Counter: know when to generate \n

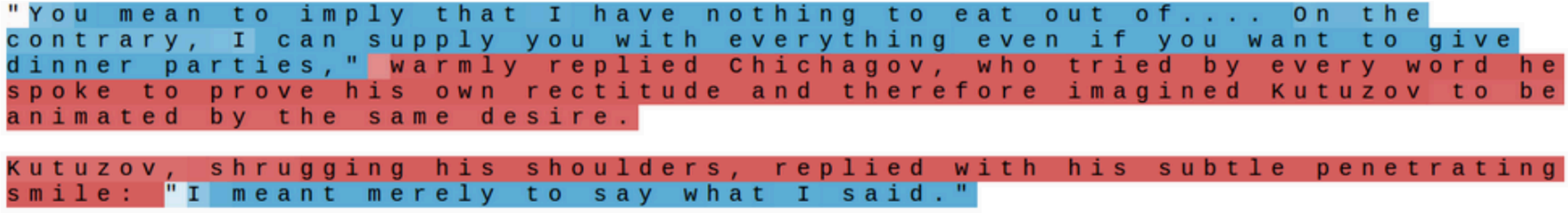


```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply 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.
```

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

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: tells us if we’re in a quote or not



"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."

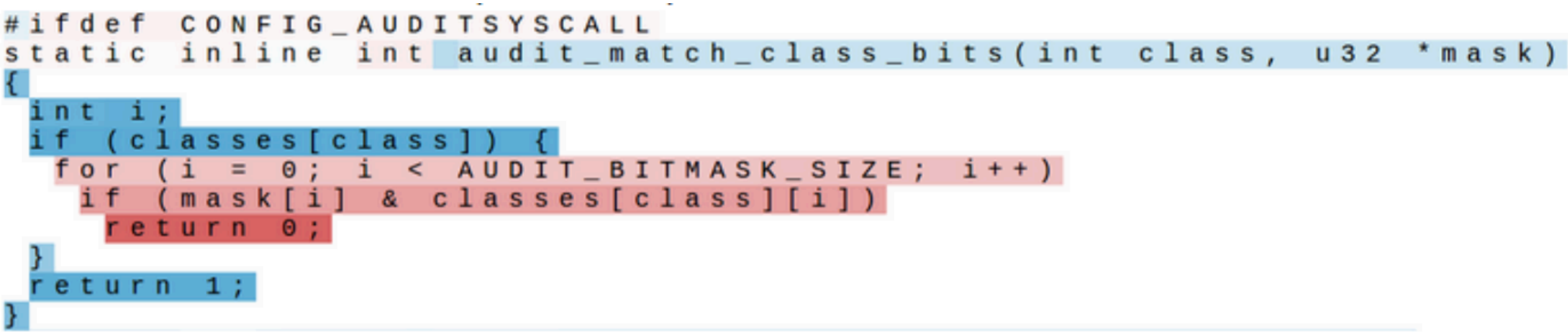
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

- Visualize activations of specific cells to see what they track

- Stack: activation based on indentation



```
#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;
}
```

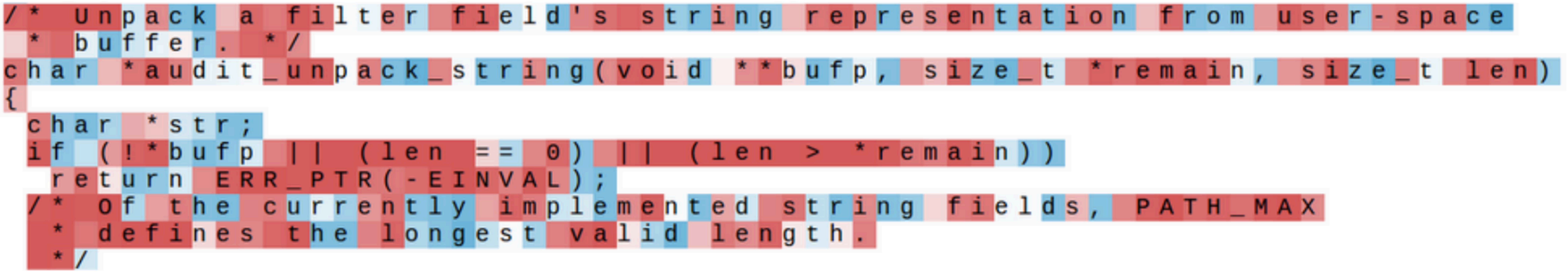
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

- Visualize activations of specific cells to see what they track

- Uninterpretable: probably doing double-duty, or only makes sense in the context of another activation



```
/* 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.
  */
```

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

Applications of Language Modeling

- All generation tasks: translation, dialogue, text simplification, paraphrasing, etc.
- Grammatical error correction
- Predictive text
- Pretraining! (more later in the course)
  - Language modeling involves predicting words given context.
  - Learning a neural network to do this induces useful representations for other tasks, similar to word2vec/GloVe.
  - ELMo, BERT, RoBERTa, GPT-2, GPT-3, BART, T5 …

Takeaways

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector

- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation

- Next time: CNNs and neural CRFs