TTIC 31190: Natural Language Processing

Kevin Gimpel

Winter 2016

Lecture 11: Recurrent and Convolutional Neural Networks in NLP

Announcements

- Assignment 3 assigned yesterday, due Feb. 29
- project proposal due Tuesday, Feb. 16
- midterm on Thursday, Feb. 18

Roadmap

- classification
- words
- lexical semantics
- language modeling
- sequence labeling
- neural network methods in NLP
- syntax and syntactic parsing
- semantic compositionality
- semantic parsing
- unsupervised learning
- machine translation and other applications

2-transformation (1-layer) network

$$\boldsymbol{z}^{(1)} = g\left(W^{(0)}\boldsymbol{x} + \boldsymbol{b}^{(0)}\right)$$

$$\boldsymbol{s} = g\left(W^{(1)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(1)}\right)$$

vector of label scores

- we'll call this a "2-transformation" neural network, or a "1-layer" neural network
- input vector is $\boldsymbol{x}$
- score vector is $\boldsymbol{s}$
- one hidden vector $\boldsymbol{z}^{(1)}$ ("hidden layer")

1-layer neural network for sentiment classification

$$\boldsymbol{z}^{(1)} = g\left(W^{(0)}\boldsymbol{x} + \boldsymbol{b}^{(0)}\right)$$

$$\boldsymbol{s} = g\left(W^{(1)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{s} = \begin{bmatrix} \mathrm{score}(\boldsymbol{x}, \mathrm{positive}, \boldsymbol{\theta}) \\ \mathrm{score}(\boldsymbol{x}, \mathrm{negative}, \boldsymbol{\theta}) \end{bmatrix}$$

Use softmax function to convert scores into probabilities

$$\mathrm{softmax}(\boldsymbol{s}) = \begin{bmatrix} \frac{\exp\{s_1\}}{\sum_i \exp\{s_i\}} \\ \cdots \\ \frac{\exp\{s_d\}}{\sum_i \exp\{s_i\}} \end{bmatrix}$$

$$\boldsymbol{s} = \begin{bmatrix} \mathrm{score}(\boldsymbol{x}, \mathrm{positive}, \boldsymbol{\theta}) \\ \mathrm{score}(\boldsymbol{x}, \mathrm{negative}, \boldsymbol{\theta}) \end{bmatrix}$$

$$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s}) = \begin{bmatrix} \frac{\exp\{\mathrm{score}(\boldsymbol{x},\mathrm{positive},\boldsymbol{\theta})\}}{Z} \\ \frac{\exp\{\mathrm{score}(\boldsymbol{x},\mathrm{negative},\boldsymbol{\theta})\}}{Z} \end{bmatrix}$$

$$Z = \exp\{\mathrm{score}(\boldsymbol{x}, \mathrm{positive}, \boldsymbol{\theta})\} + \exp\{\mathrm{score}(\boldsymbol{x}, \mathrm{negative}, \boldsymbol{\theta})\}$$

Neural Networks for Twitter Part-of-Speech Tagging

other verb det noun pronoun

intj pronoun prep adj prep verb

ikr smh he asked fir yo last name so he can

add u on fb lololol

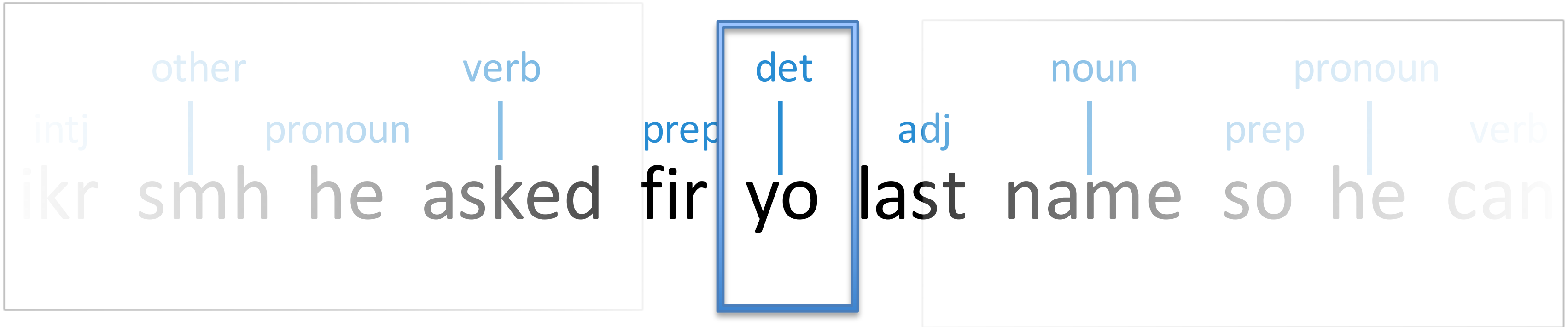
verb prep intj

pronoun proper noun

adj = adjective
prep = preposition
intj = interjection

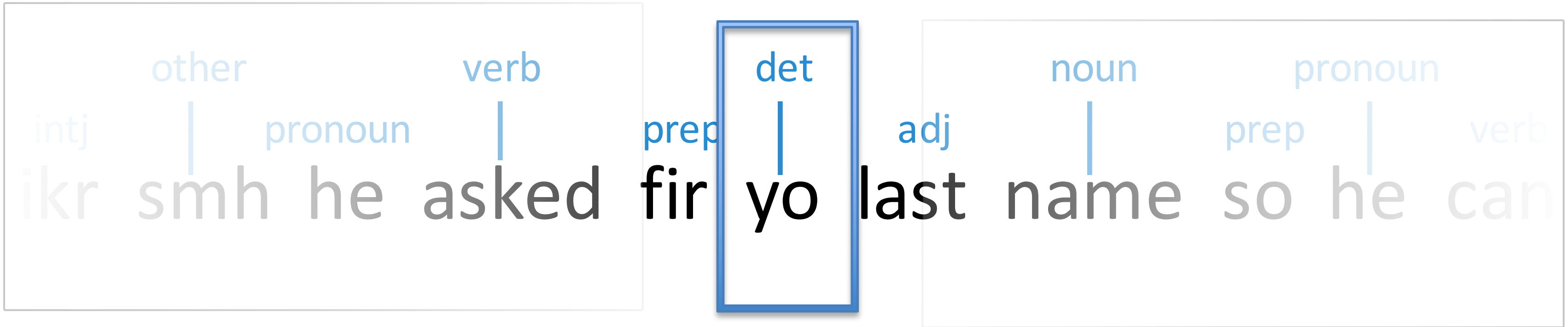
- in Assignment 3, you'll build a neural network classifier to predict a word's POS tag based on its context

Neural Networks for Twitter Part-of-Speech Tagging



- e.g., predict tag of *yo* given context
- what should the input ***x*** be?
  - it has to be independent of the label
  - it has to be a **fixed-length** vector

Neural Networks for Twitter Part-of-Speech Tagging



- e.g., predict tag of *yo* given context
- what should the input ***x*** be?

$$\boldsymbol{x} = \underbrace{[0.4\ \ 0.1\ \ \ldots\ \ 0.9]}_{\text{word vector for } \textit{yo}}{}^\top$$

Neural Networks for Twitter Part-of-Speech Tagging

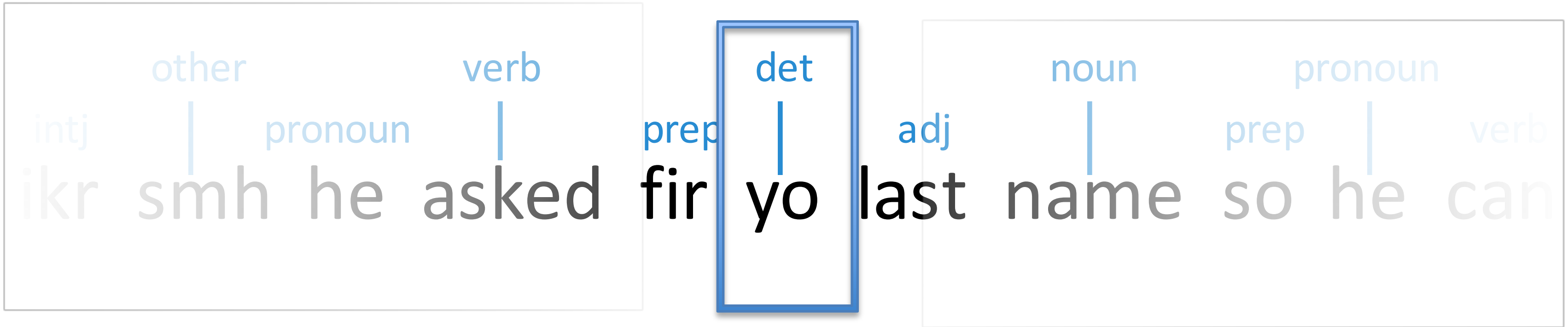
intj other pronoun verb prep det adj noun prep pronoun verb

ikr smh he asked fir yo last name so he can

- e.g., predict tag of *yo* given context
- what should the input ***x*** be?

$$\boldsymbol{x} = [\underbrace{-0.2 \;\; 0.5 \;\; \ldots \;\; 0.8}_{\text{word vector for } \textit{fir}} \;\; \underbrace{0.4 \;\; 0.1 \;\; \ldots \;\; 0.9}_{\text{word vector for } \textit{yo}}]^\top$$

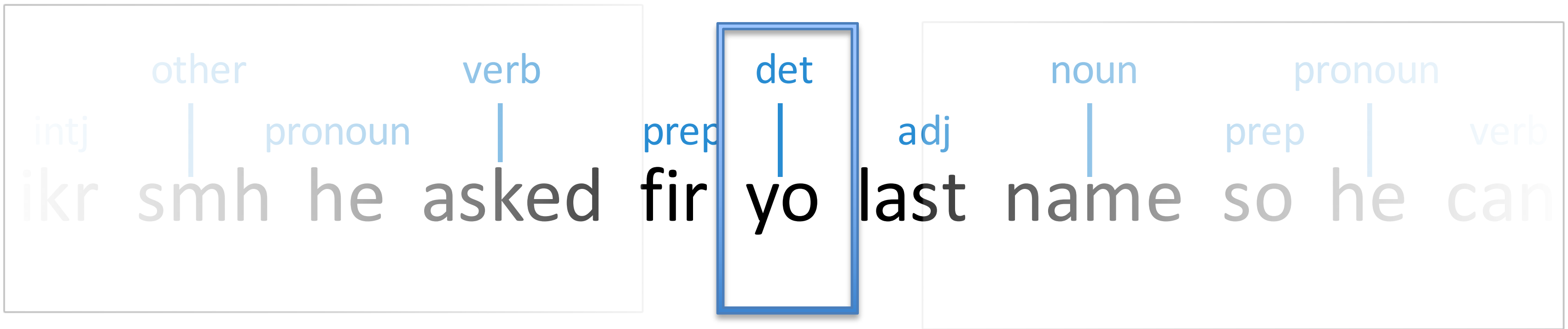
Neural Networks for Twitter Part-of-Speech Tagging



- when using word vectors as part of input, we can also treat them as more parameters to be learned!
- this is called “updating” or “fine-tuning” the vectors (since they are initialized using something like word2vec)

$$\boldsymbol{x} = [\underbrace{-0.2 \;\; 0.5 \;\; \ldots \;\; 0.8}_{\text{word vector for } \textit{fir}} \;\; \underbrace{0.4 \;\; 0.1 \;\; \ldots \;\; 0.9}_{\text{word vector for } \textit{yo}}]^\top$$

Neural Networks for Twitter Part-of-Speech Tagging



- let's use the center word + two words to the right:

$$\boldsymbol{x} = [\underbrace{0.4 \;\dots\; 0.9}_{\text{vector for } \textit{yo}}\ \underbrace{0.2 \;\dots\; 0.7}_{\text{vector for } \textit{last}}\ \underbrace{0.3 \;\dots\; 0.6}_{\text{vector for } \textit{name}}]^\top$$

- if *name* is to the right of *yo*, then *yo* is probably a form of *your*
- but our ***x*** above uses separate dimensions for each position!
  - i.e., *name* is two words to the right
  - what if *name* is one word to the right?

Features and Filters

- we could use a feature that returns 1 if *name* is to the right of the center word, but that does not use the word's embedding
- how do we include a feature like "a word similar to *name* appears somewhere to the right of the center word"?
- rather than always specify relative position and embedding, we want to add **filters** that look for words like *name* **anywhere in the window (or sentence!)**

Filters

- for now, think of a filter as a vector in the word vector space
- the filter matches a particular region of the space
- “match” = “has high dot product with”

Convolution

- convolutional neural networks use a bunch of such filters
- each filter is matched against (dot product computed with) each word in the entire context window or sentence
- e.g., a single filter $\boldsymbol{w}$ is a vector of same length as word vectors

Convolution

$$\boldsymbol{w}$$

$$\boldsymbol{x} = [\underbrace{0.4 \ \ldots \ 0.9}_{\text{vector for } \textit{yo}} \ \underbrace{0.2 \ \ldots \ 0.7}_{\text{vector for } \textit{last}} \ \underbrace{0.3 \ \ldots \ 0.6}_{\text{vector for } \textit{name}}]^\top$$

$$c_1 = \boldsymbol{w} \cdot \boldsymbol{x}_{1:d}$$

Convolution

$$\boldsymbol{w}$$

$$\boldsymbol{x} = [\underbrace{0.4\ \ldots\ 0.9}_{\text{vector for } \textit{yo}}\ \underbrace{0.2\ \ldots\ 0.7}_{\text{vector for } \textit{last}}\ \underbrace{0.3\ \ldots\ 0.6}_{\text{vector for } \textit{name}}]^\top$$

$$c_2 = \boldsymbol{w} \cdot \boldsymbol{x}_{d+1:2d}$$

Convolution

$$\boldsymbol{w}$$

$$\boldsymbol{x} = [\underbrace{0.4 \ \ldots \ 0.9}_{\text{vector for } \textit{yo}} \ \underbrace{0.2 \ \ldots \ 0.7}_{\text{vector for } \textit{last}} \ \underbrace{0.3 \ \ldots \ 0.6}_{\text{vector for } \textit{name}}]^\top$$

$$c_3 = \boldsymbol{w} \cdot \boldsymbol{x}_{2d+1:3d}$$

Convolution

$\boldsymbol{c}$ = "feature map", has an entry for each word position in context window / sentence

$$\boldsymbol{x} = [\underbrace{0.4\ \ldots\ 0.9}_{\text{vector for } \textit{yo}}\ \underbrace{0.2\ \ldots\ 0.7}_{\text{vector for } \textit{last}}\ \underbrace{0.3\ \ldots\ 0.6}_{\text{vector for } \textit{name}}]^\top$$

$$c_1 = \boldsymbol{w} \cdot \boldsymbol{x}_{1:d}$$

$$c_2 = \boldsymbol{w} \cdot \boldsymbol{x}_{d+1:2d}$$

$$c_3 = \boldsymbol{w} \cdot \boldsymbol{x}_{2d+1:3d}$$

Pooling

$\boldsymbol{c}$ = "feature map", has an entry for each word position in context window / sentence

how do we convert this into a fixed-length vector?

use **pooling**:

max-pooling: returns maximum value in $\boldsymbol{c}$

average pooling: returns average of values in $\boldsymbol{c}$

vector for *yo* vector for *last* vector for *name*

$$c_1 = \boldsymbol{w} \cdot \boldsymbol{x}_{1:d}$$

$$c_2 = \boldsymbol{w} \cdot \boldsymbol{x}_{d+1:2d}$$

$$c_3 = \boldsymbol{w} \cdot \boldsymbol{x}_{2d+1:3d}$$

Pooling

$\boldsymbol{c}$ = "feature map", has an entry for each word position in context window / sentence

how do we convert this into a fixed-length vector?
use **pooling**:

max-pooling: returns maximum value in $\boldsymbol{c}$

average pooling: returns average of values in $\boldsymbol{c}$

vector for *yo* vector for *last* vector for *name*

$$c_1 = \boldsymbol{w} \cdot \boldsymbol{x}_{1:d}$$

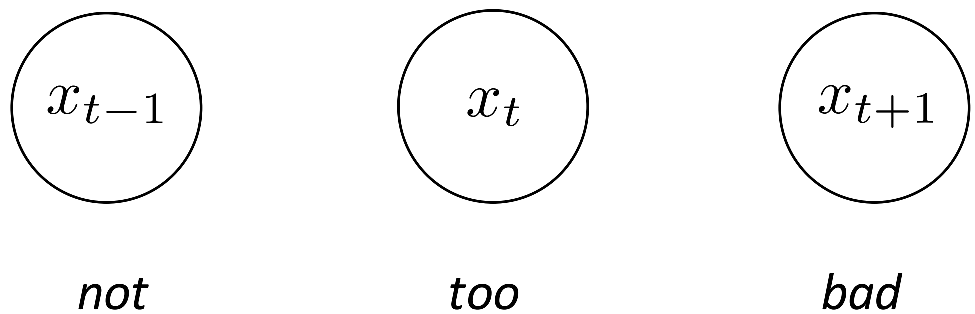
then, this single filter $\boldsymbol{w}$ produces a single feature value (the output of some kind of pooling).
in practice, we use many filters of many different lengths (e.g., *n*-grams rather than words).

Convolutional Neural Networks

- convolutional neural networks (**convnets** or **CNNs**) use filters that are "convolved with" (matched against all positions of) the input
- informally, think of convolution as "perform the same operation everywhere on the input in some systematic order"
- "convolutional layer" = set of filters that are convolved with the input vector (whether ***x*** or hidden vector)
- could be followed by more convolutional layers, or by a type of pooling
- often used in NLP to convert a sentence into a feature vector

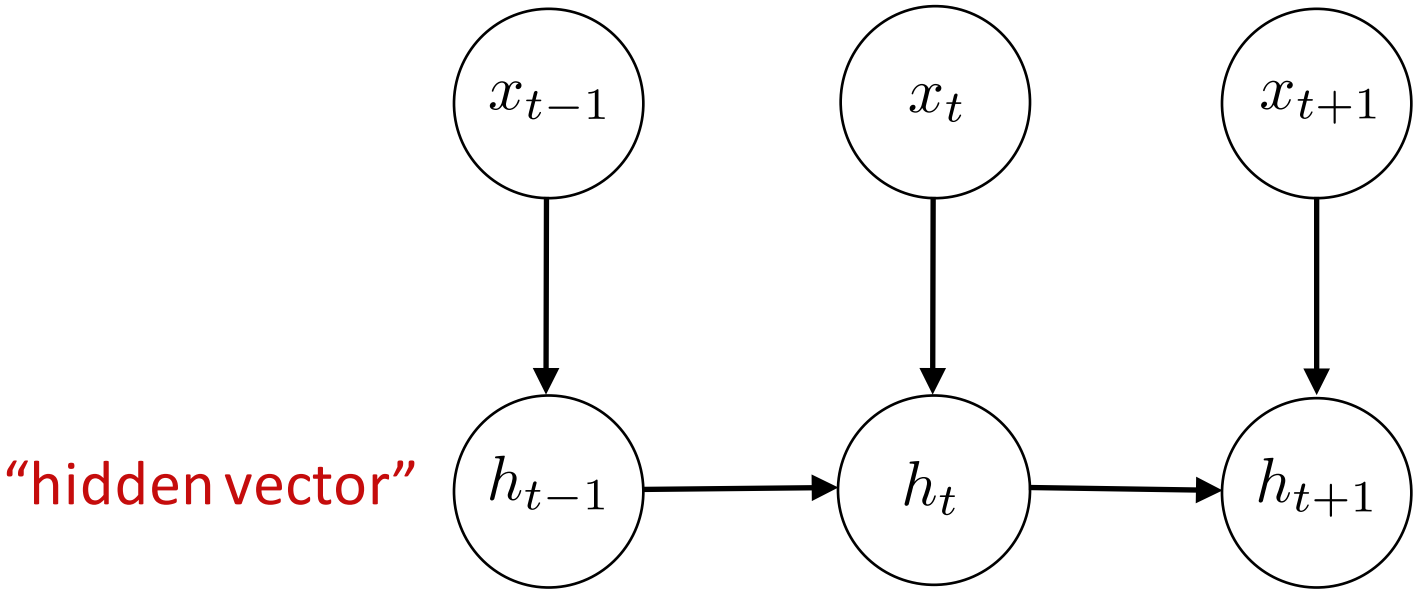
Recurrent Neural Networks

Input is a sequence:



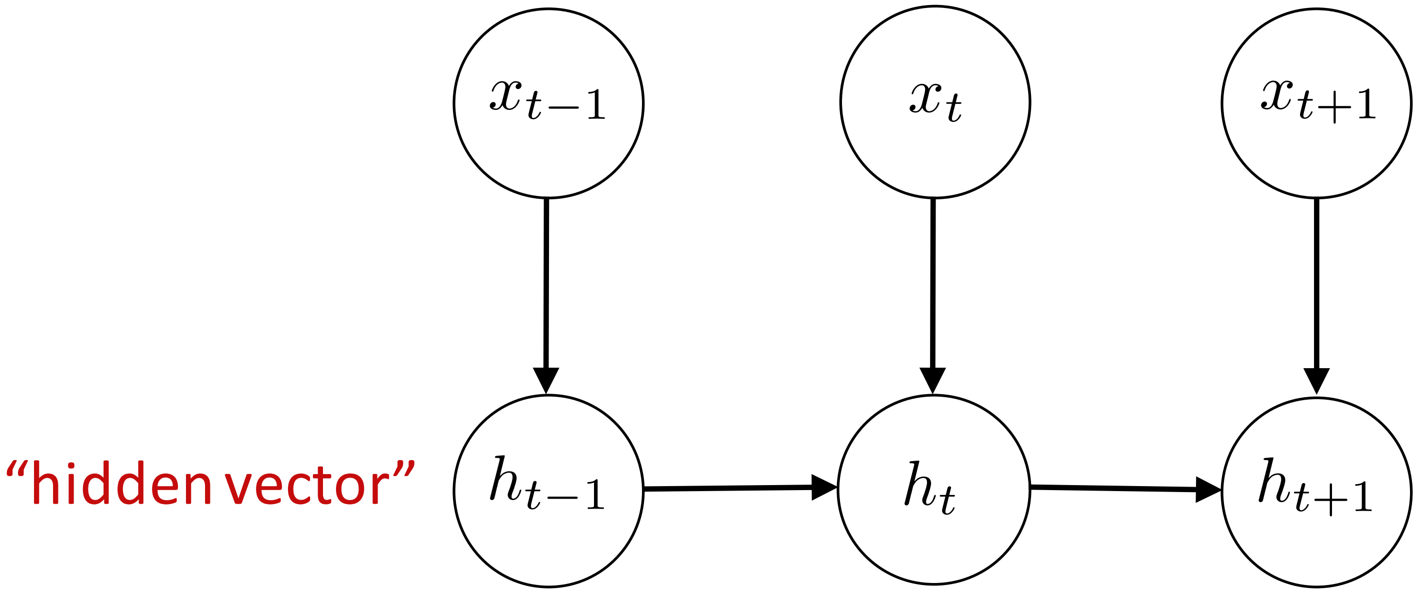
Recurrent Neural Networks

Input is a sequence:



Recurrent Neural Networks

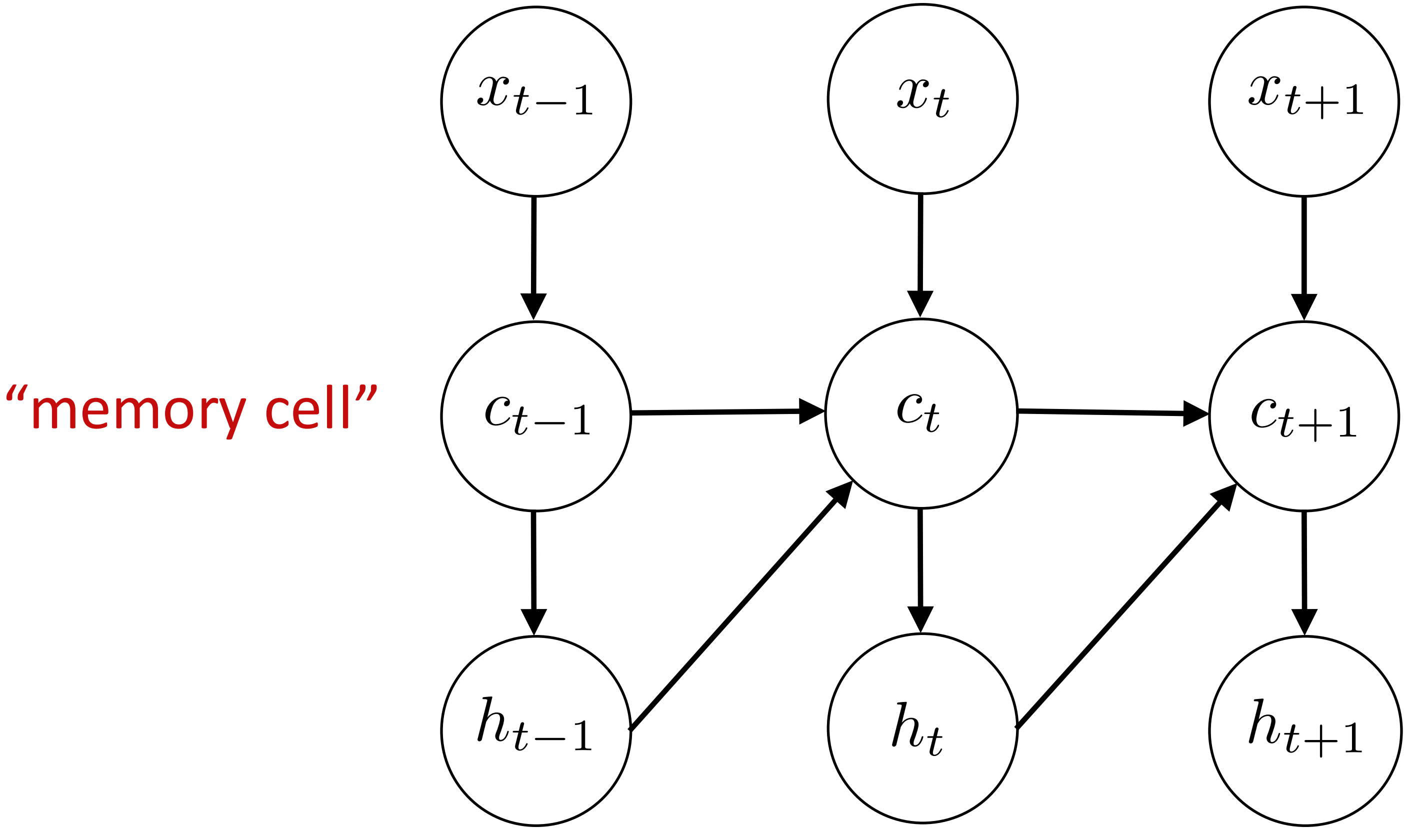
$$h_t = \tanh\left(W^{(xh)} x_t + W^{(hh)} h_{t-1} + b^{(h)}\right)$$



Disclaimer

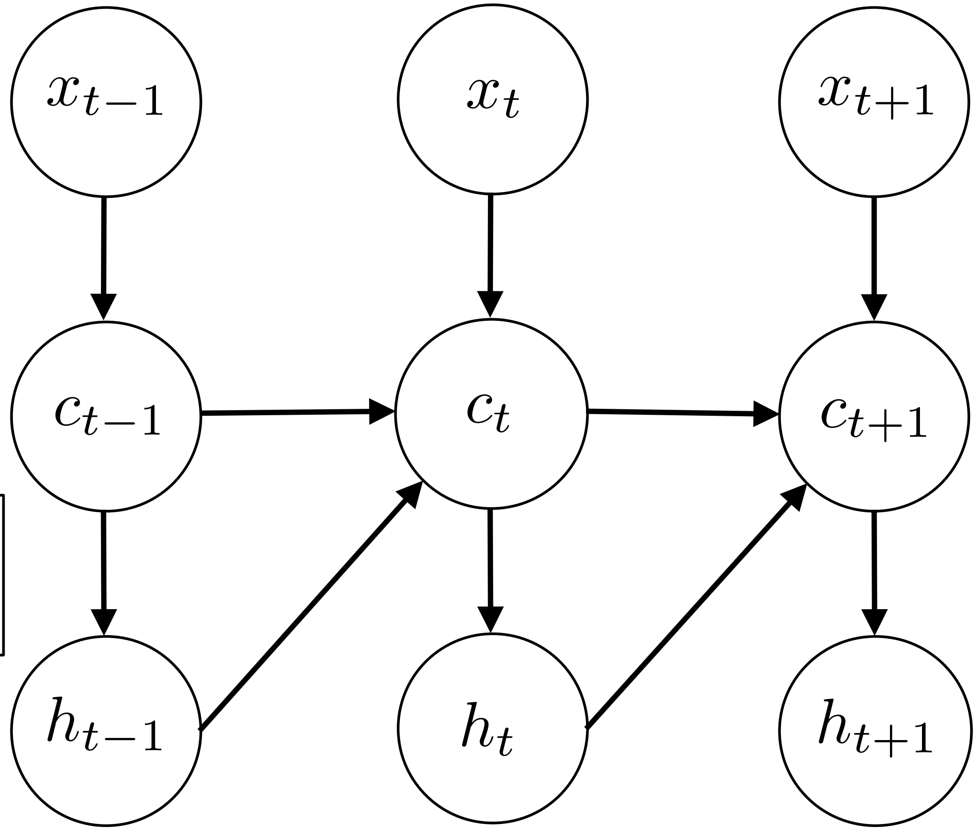
- these diagrams are often useful for helping us understand and communicate neural network architectures
- but they rarely have any sort of formal semantics (unlike graphical models)
- they are more like cartoons

Long Short-Term Memory RNNs (gateless)



Long Short-Term Memory RNNs (gateless)

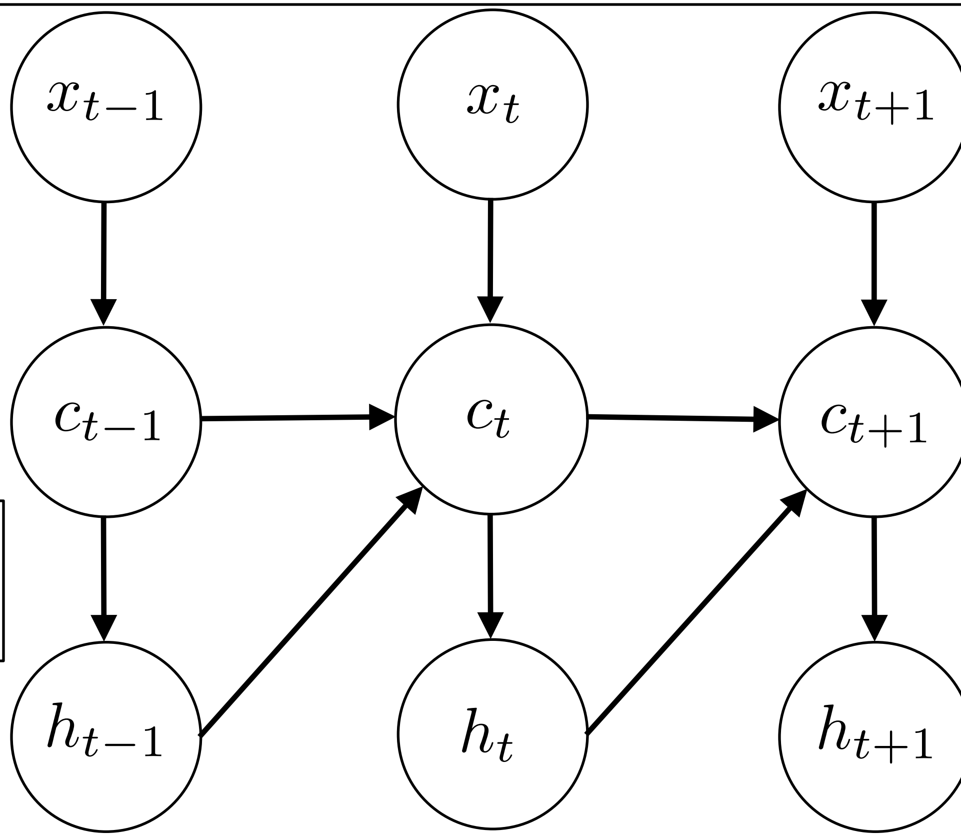
$$h_t = \tanh\left(c_t\right)$$



Long Short-Term Memory RNNs (gateless)

$$\boxed{c_t = c_{t-1} + \tanh\left(W^{(xc)} x_t + W^{(hc)} h_{t-1} + b^{(c)}\right)}$$

$$\boxed{h_t = \tanh\left(c_t\right)}$$



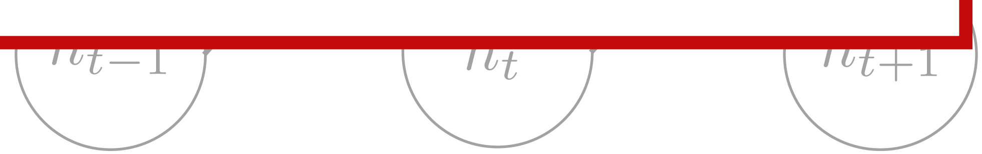
Long Short-Term Memory RNNs (gateless)

$$c_t = c_{t-1} + \tanh\left(W^{(xc)}x_t + W^{(hc)}h_{t-1} + b^{(c)}\right)$$

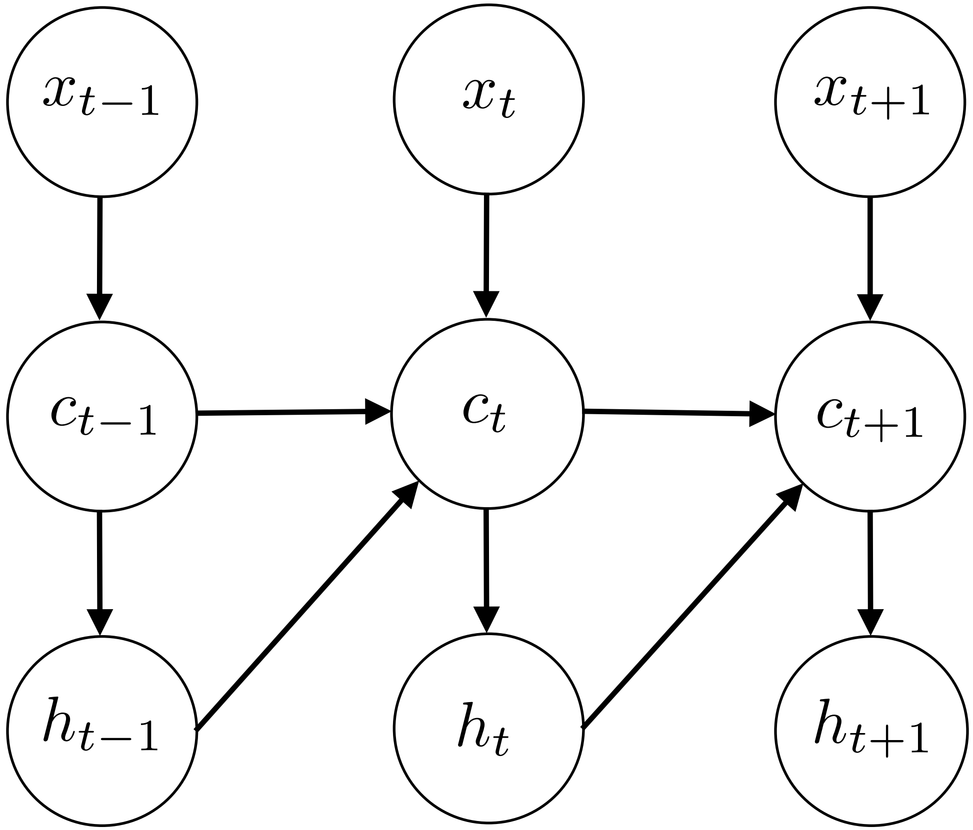
Experiment: text classification

- Stanford Sentiment Treebank
  - binary classification (positive/negative)
- 25-dim word vectors
- 50-dim cell/hidden vectors
- classification layer on **final** hidden vector
- AdaGrad, 10 epochs, mini-batch size 10
- early stopping on dev set

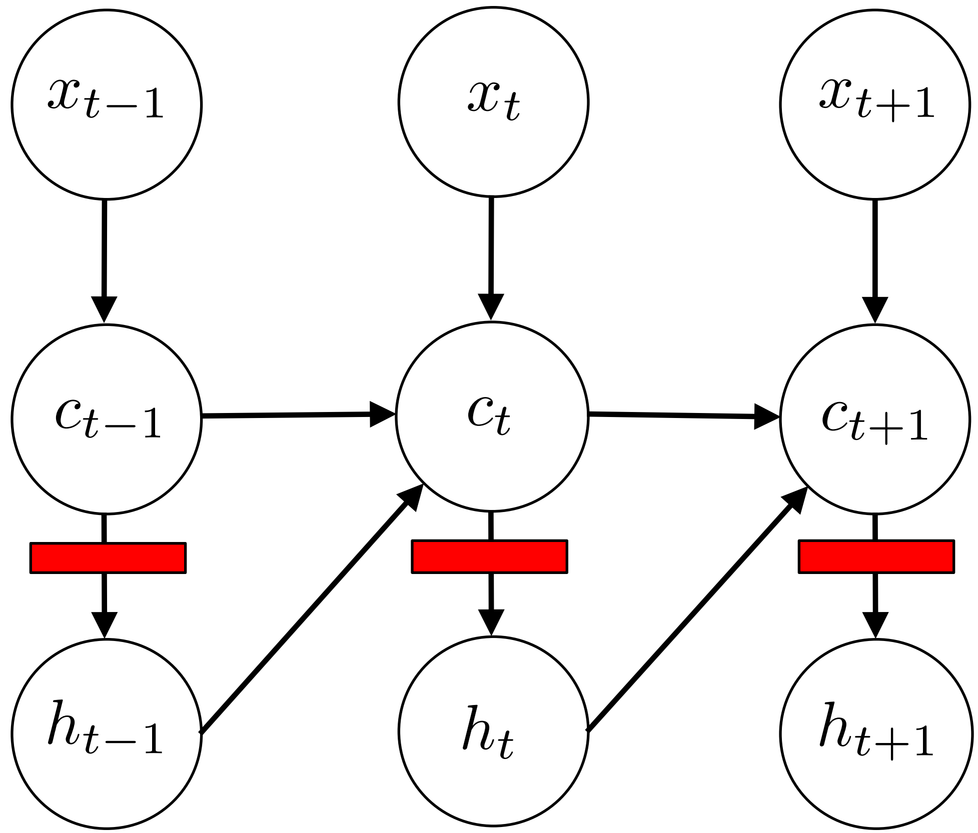
accuracy
80.6



Output Gates



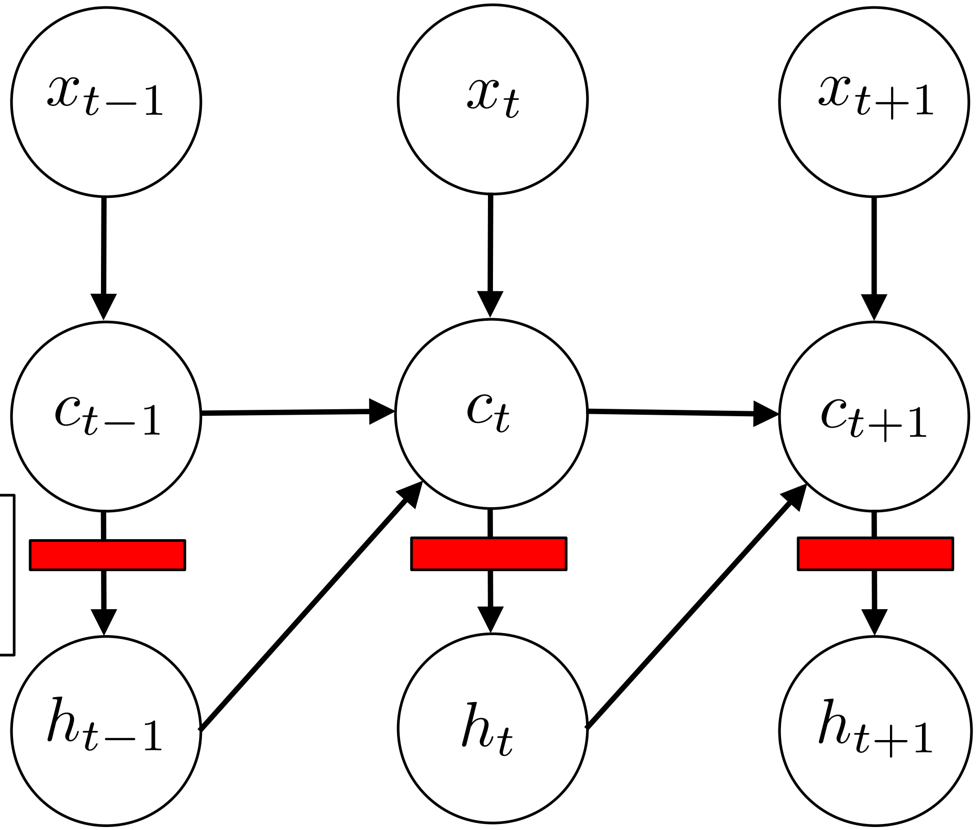
Output Gates



Output Gates

$$h_t = \tanh\left(c_t\right)$$

$$h_t = o_t \tanh(c_t)$$



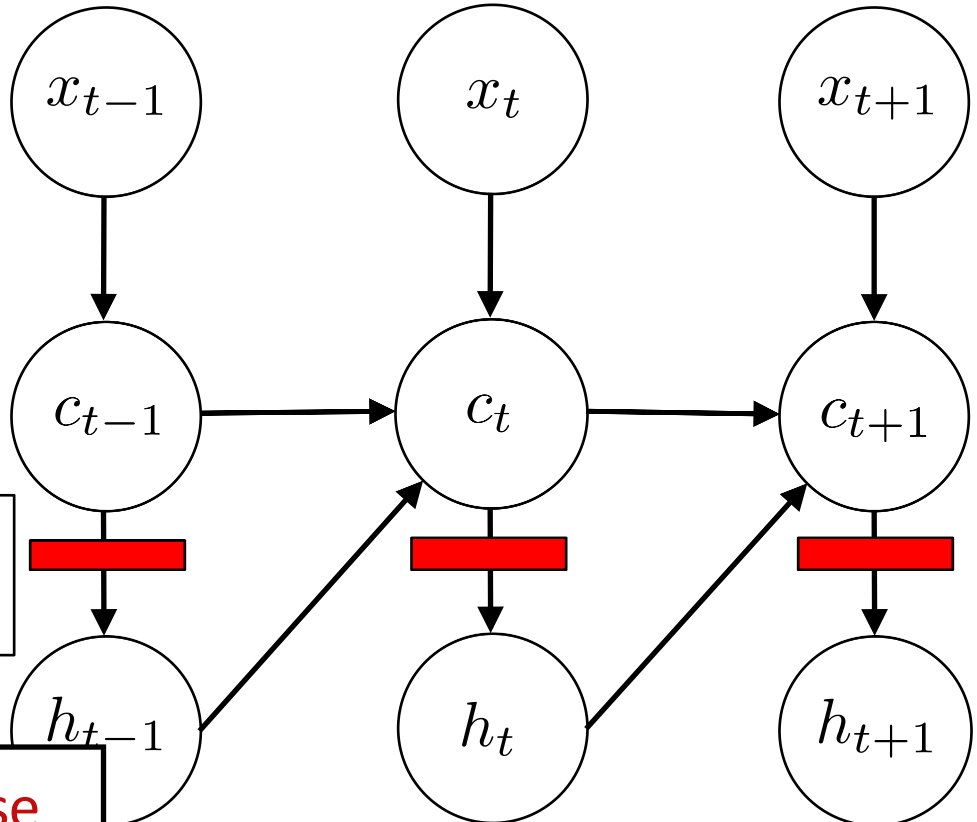
Output Gates

$$h_t = \tanh\left(c_t\right)$$

$$h_t = o_t \tanh(c_t)$$

this is pointwise multiplication!

o_t is a vector



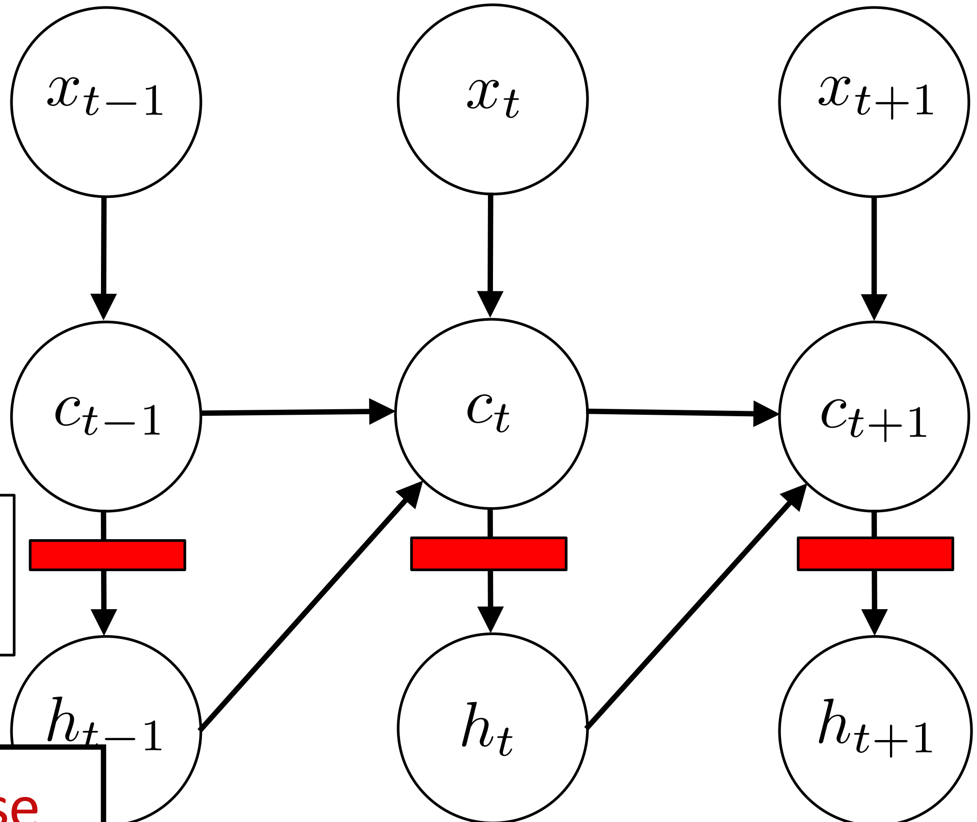
Output Gates

$$o_t = \sigma\left(W^{(xo)}x_t + W^{(ho)}h_{t-1} + W^{(co)}c_t + b^{(o)}\right)$$

$$h_t = \tanh\left(c_t\right)$$

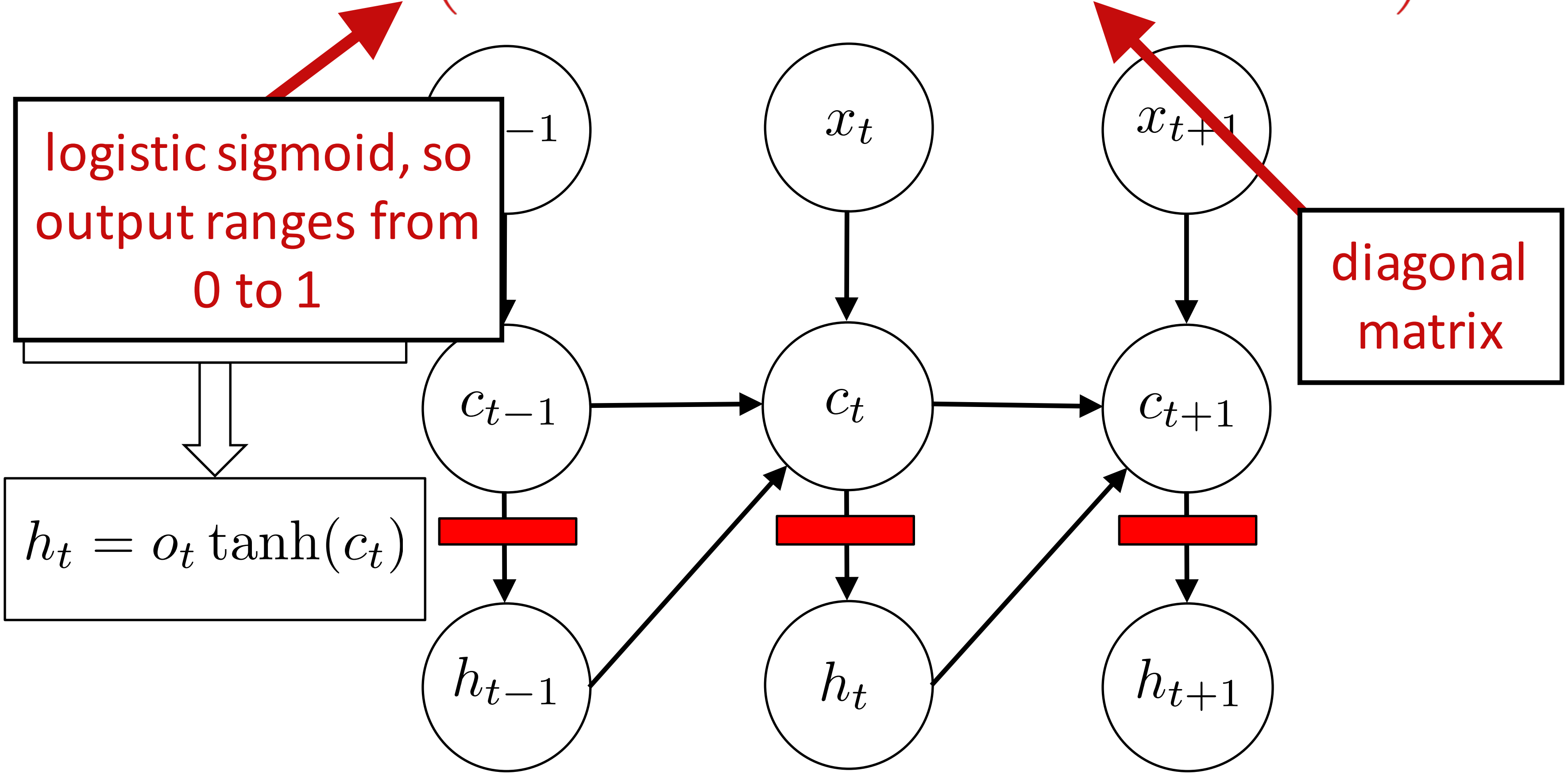
$$h_t = o_t \tanh(c_t)$$

this is pointwise multiplication!
o_t is a vector



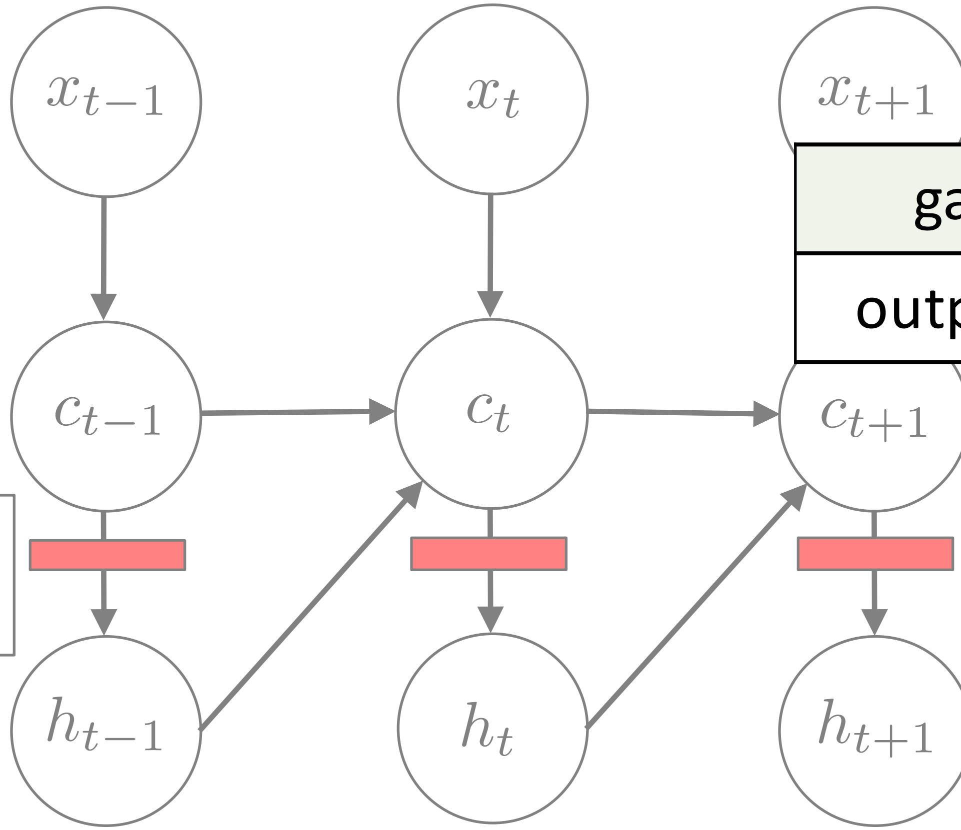
Output Gates

$$o_t = \sigma\left(W^{(xo)}x_t + W^{(ho)}h_{t-1} + W^{(co)}c_t + b^{(o)}\right)$$



Output Gates

$$o_t = \sigma\left(W^{(xo)}x_t + W^{(ho)}h_{t-1} + W^{(co)}c_t + b^{(o)}\right)$$

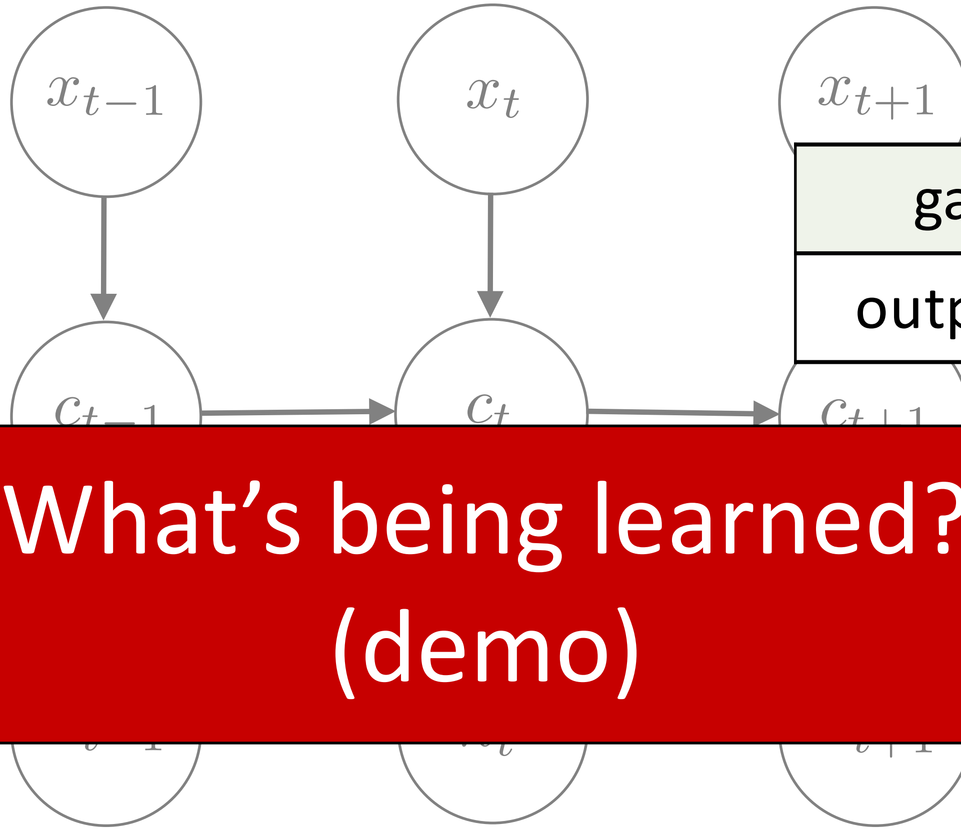


	acc.
gateless	80.6
output gates	81.9

$$h_t = o_t \tanh(c_t)$$

Output Gates

$$o_t = \sigma\left(W^{(xo)}x_t + W^{(ho)}h_{t-1} + W^{(co)}c_t + b^{(o)}\right)$$

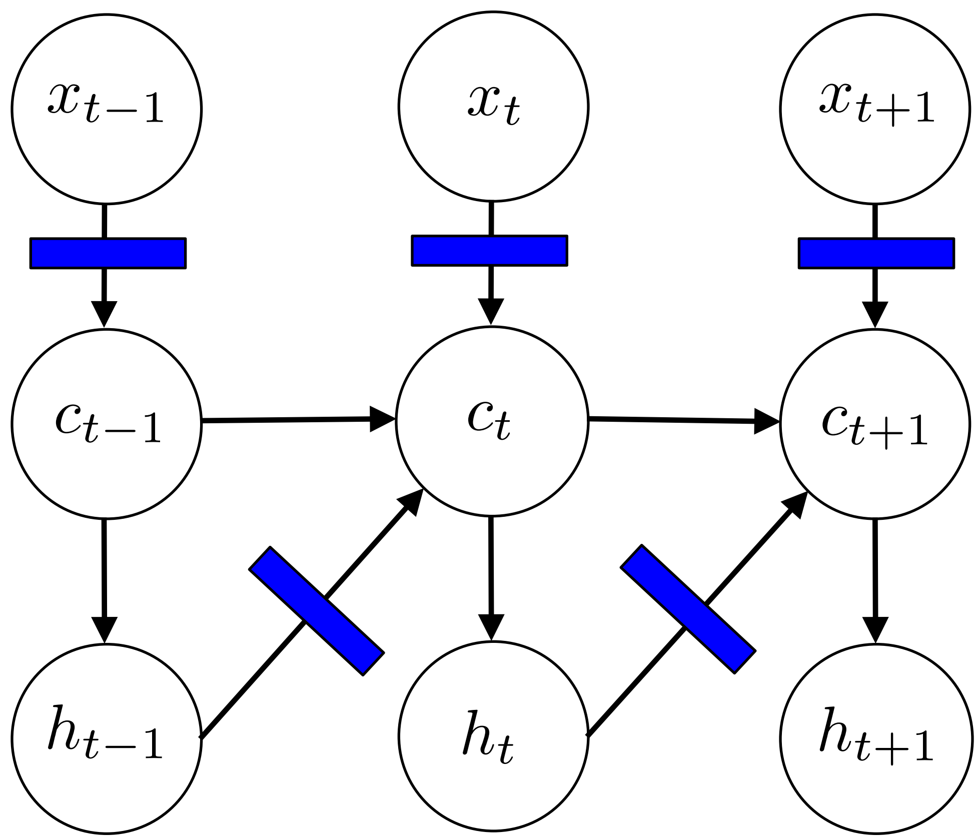


	acc.
gateless	80.6
output gates	81.9

$h_t = o_t \tanh(c$

What's being learned?
(demo)

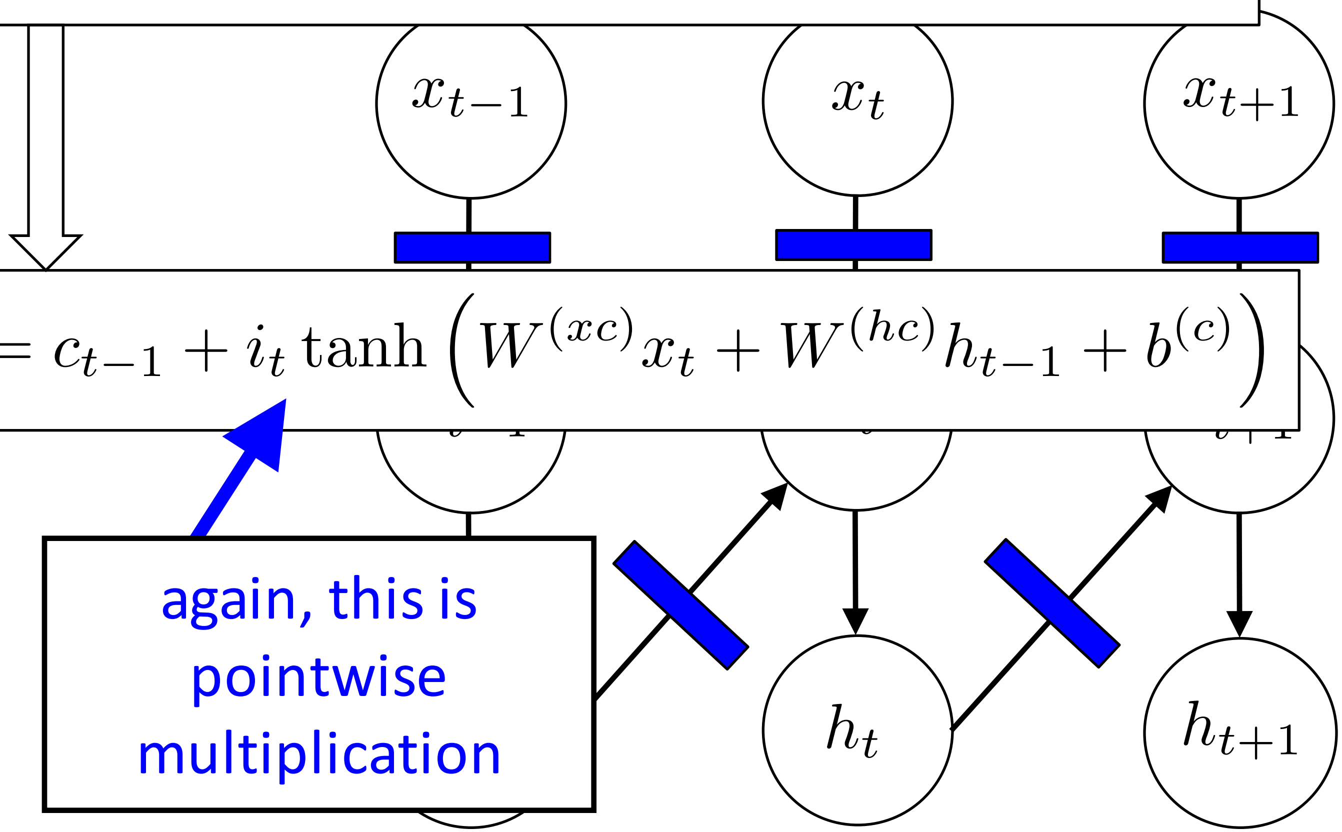
Input Gates



Input Gates

$$c_t = c_{t-1} + \tanh\left(W^{(xc)}x_t + W^{(hc)}h_{t-1} + b^{(c)}\right)$$

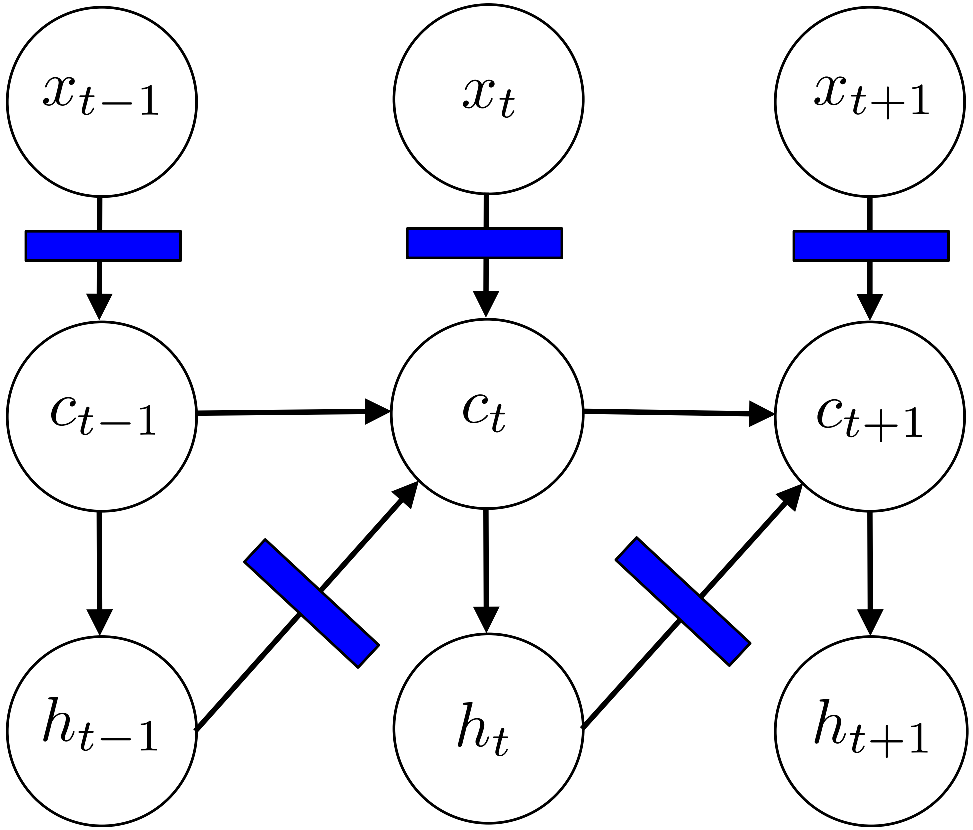
$$c_t = c_{t-1} + i_t \tanh\left(W^{(xc)}x_t + W^{(hc)}h_{t-1} + b^{(c)}\right)$$



again, this is pointwise multiplication

Input Gates

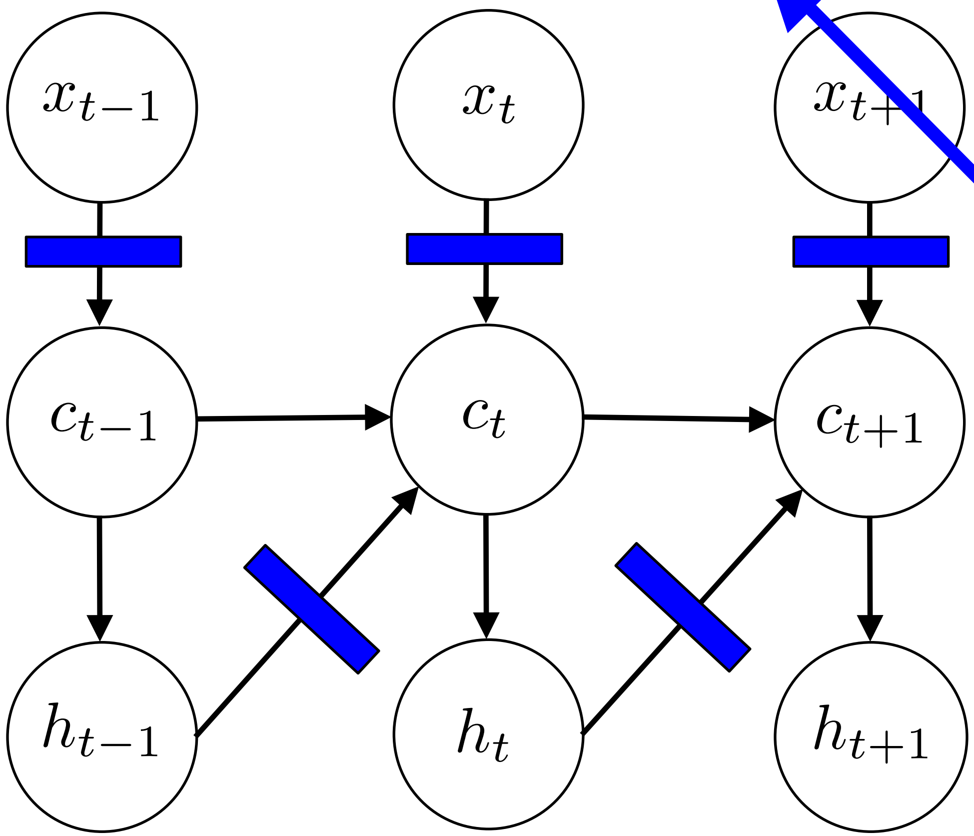
$$\boxed{c_t = c_{t-1} + i_t \tanh\left(W^{(xc)} x_t + W^{(hc)} h_{t-1} + b^{(c)}\right)}$$



Input Gates

$$i_t = \sigma\left(W^{(xi)}x_t + W^{(hi)}h_{t-1} + W^{(ci)}c_{t-1} + b^{(i)}\right)$$

diagonal matrix



Input Gates

$$i_t = \sigma\left(W^{(xi)}x_t + W^{(hi)}h_{t-1} + W^{(ci)}c_{t-1} + b^{(i)}\right)$$

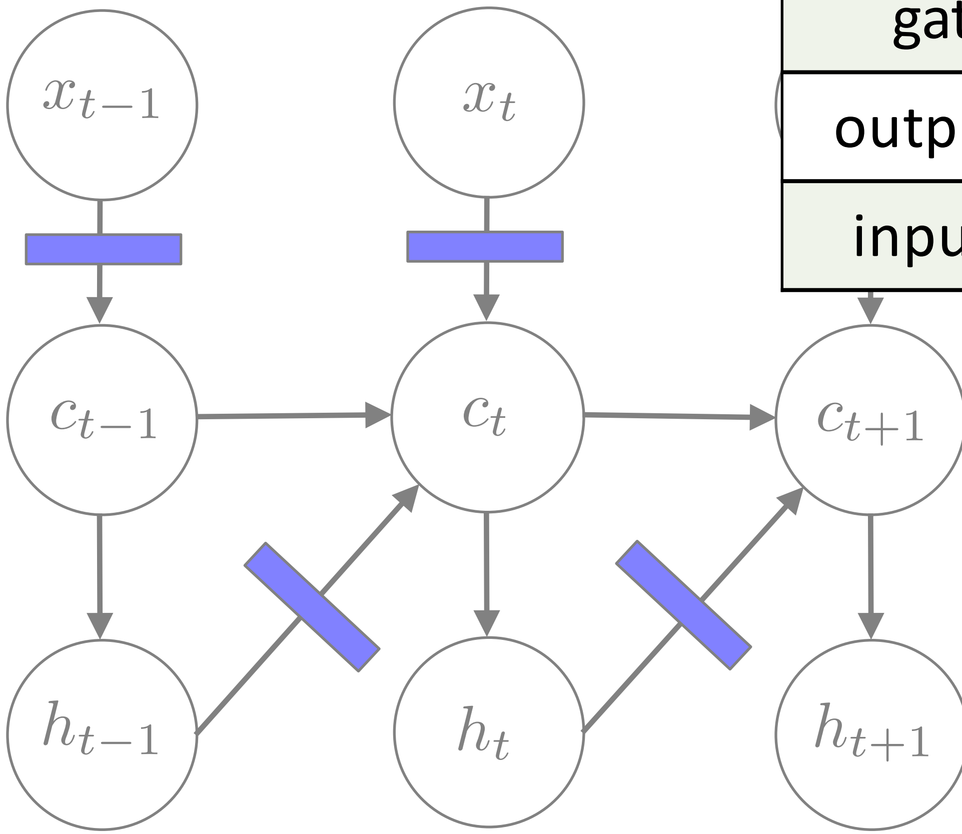
difference

Output Gates

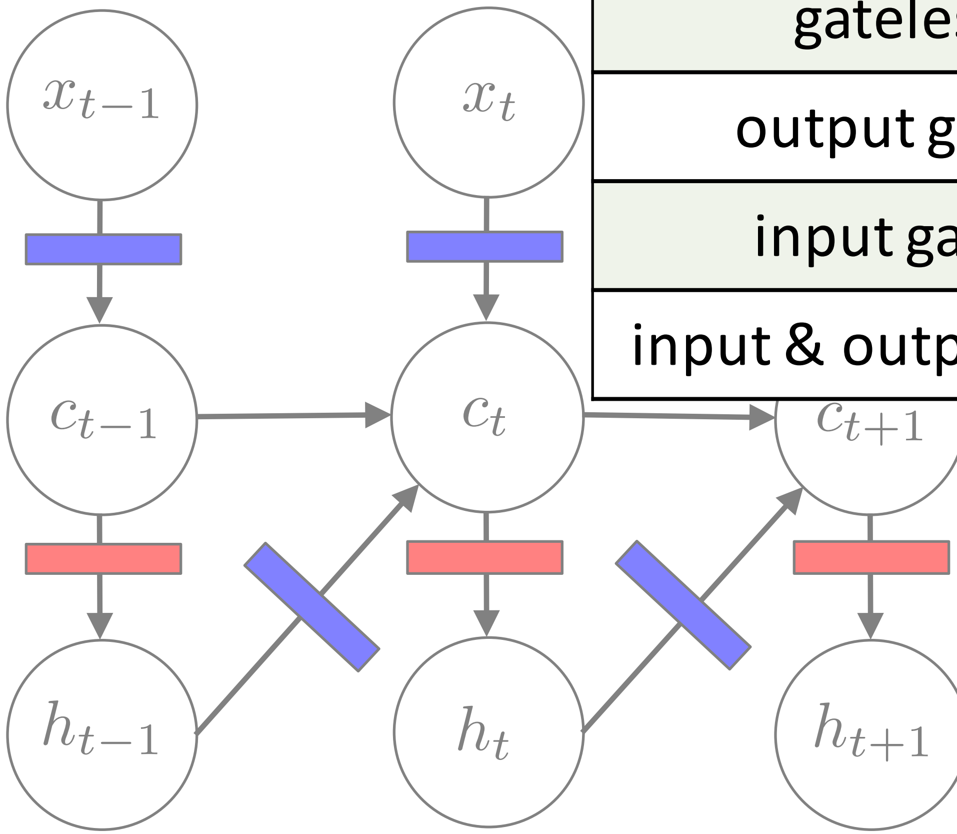
$$o_t = \sigma\left(W^{(xo)}x_t + W^{(ho)}h_{t-1} + W^{(co)}c_t + b^{(o)}\right)$$

Input Gates

	acc.
gateless	80.6
output gates	81.9
input gates	84.4

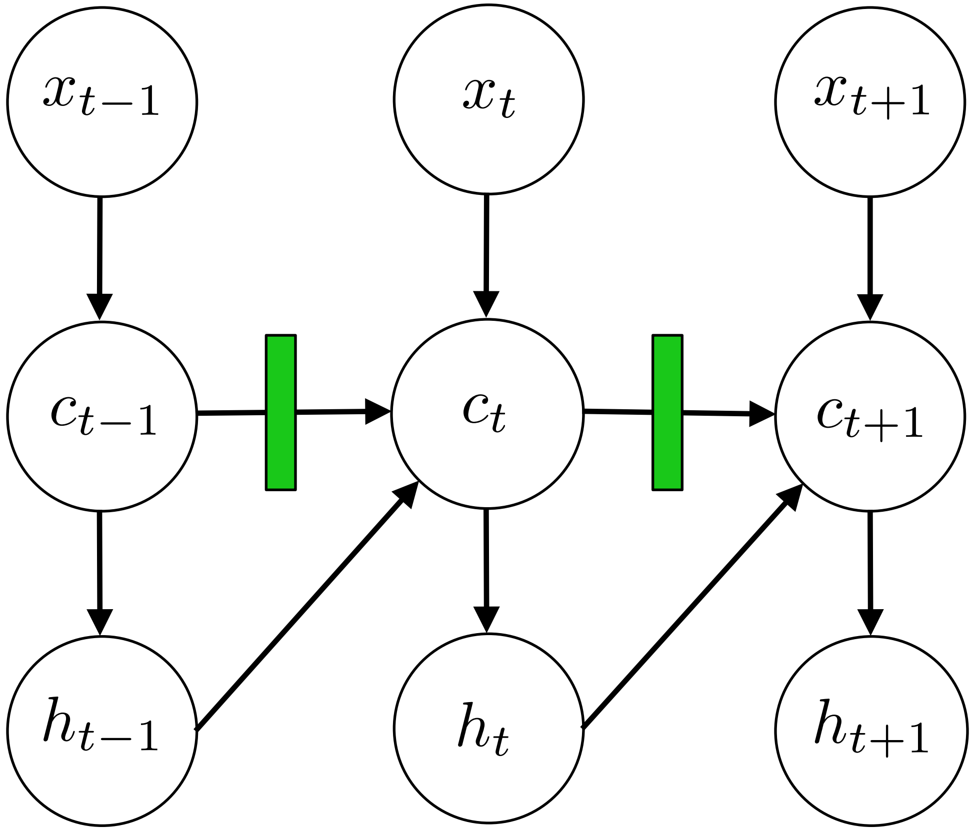


Input and Output Gates



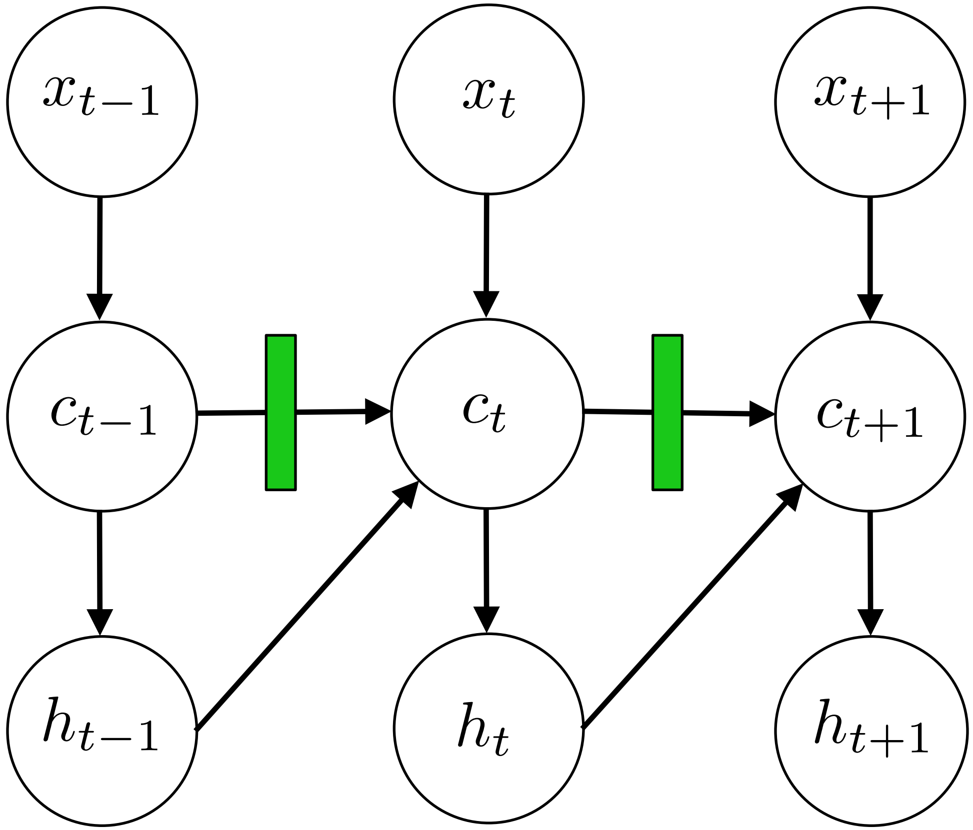
	acc.
gateless	80.6
output gates	81.9
input gates	84.4
input & output gates	84.6

Forget Gates



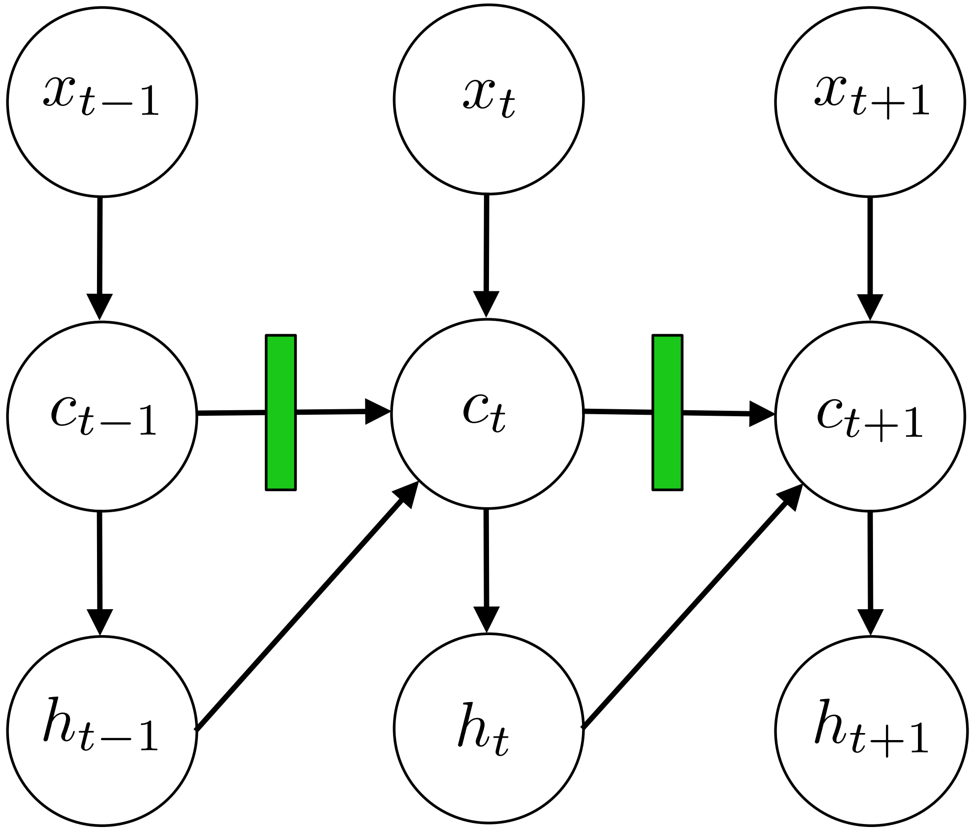
Forget Gates

$$\boxed{c_t = f_t c_{t-1} + \tanh\left(W^{(xc)} x_t + W^{(hc)} h_{t-1} + b^{(c)}\right)}$$



Forget Gates

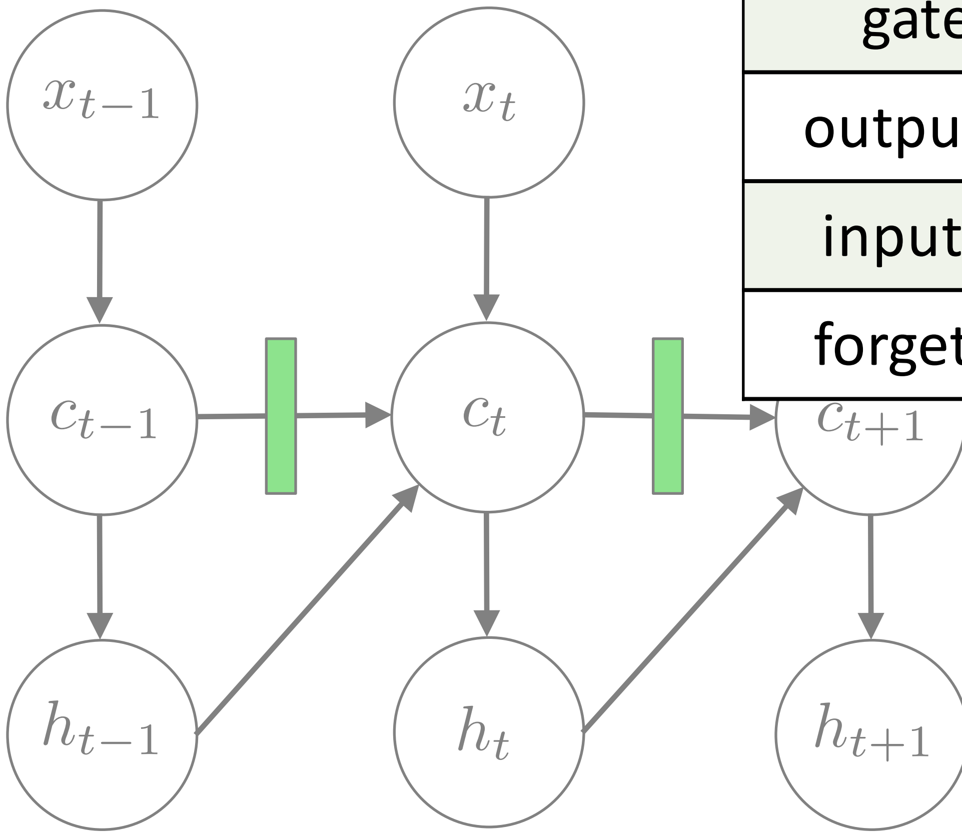
$$f_t = \sigma\left(W^{(xf)}x_t + W^{(hf)}h_{t-1} + W^{(cf)}c_{t-1} + b^{(f)}\right)$$



Forget Gates

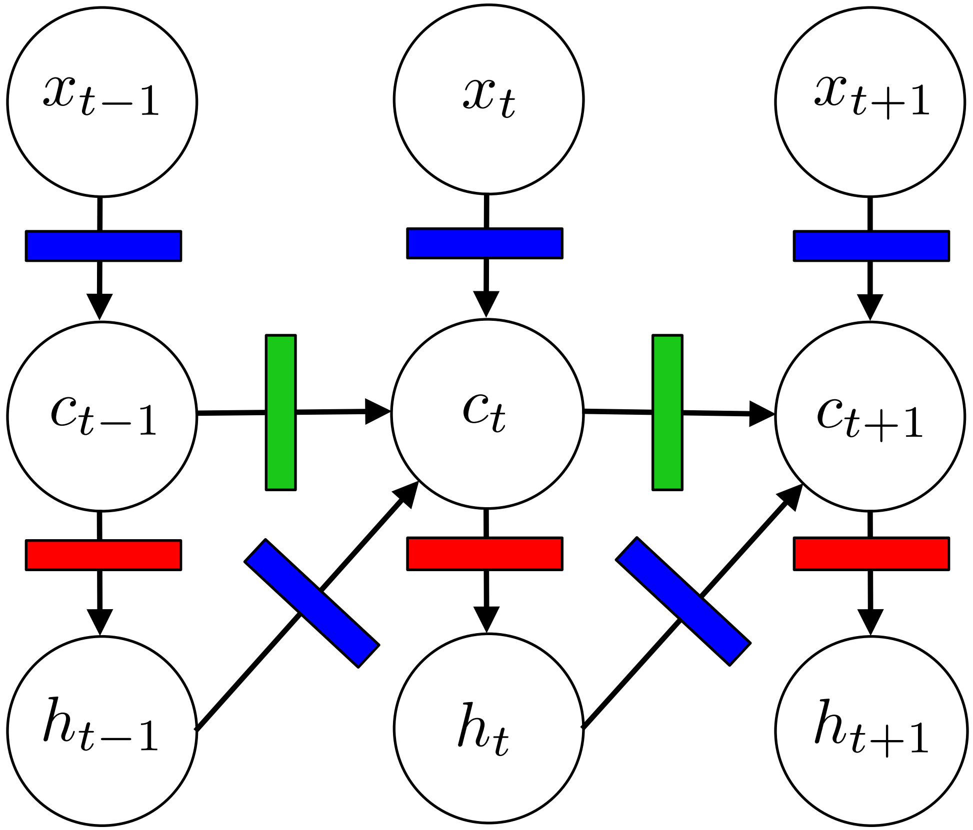
$$f_t = \sigma\left(W^{(xf)}x_t + W^{(hf)}h_{t-1} + W^{(cf)}c_{t-1} + b^{(f)}\right)$$

	acc.
gateless	80.6
output gates	81.9
input gates	84.4
forget gates	82.1



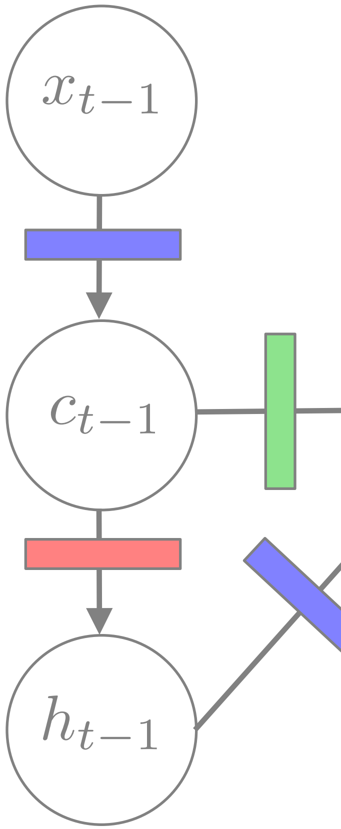
All Gates

$$c_t = f_t c_{t-1} + i_t \tanh\left(W^{(xc)} x_t + W^{(hc)} h_{t-1} + b^{(c)}\right)$$



$$h_t = o_t \tanh(c_t)$$

All Gates

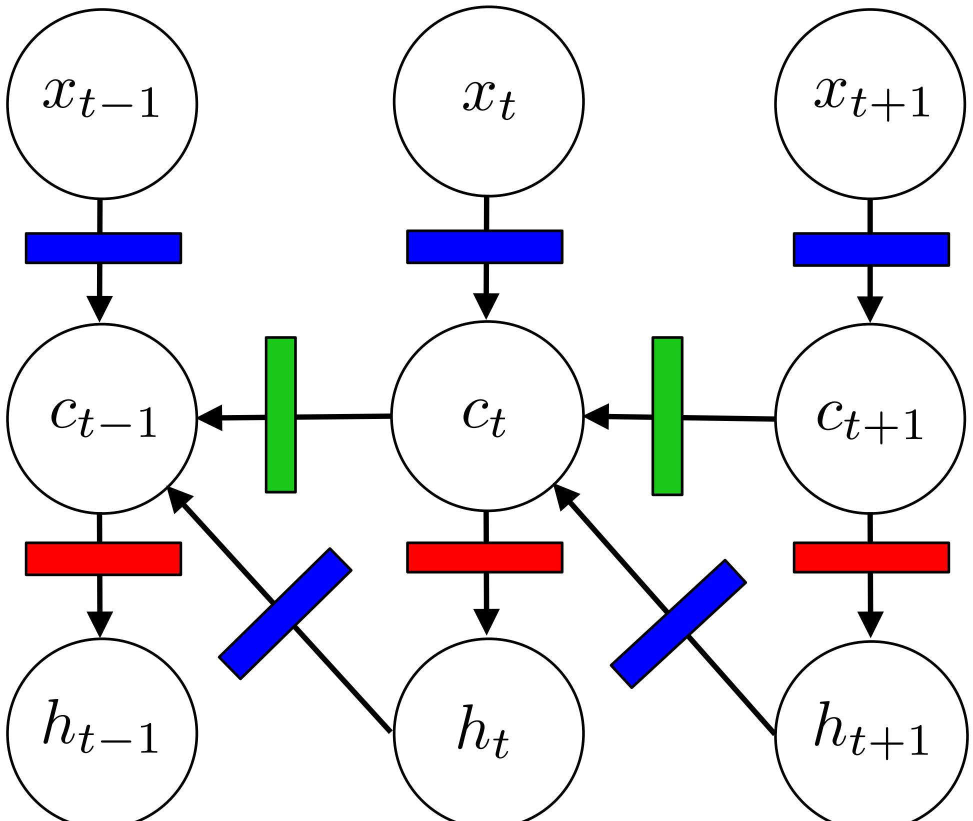


	acc.
gateless	80.6
output gates	81.9
input gates	84.4
input & output gates	84.6
forget gates	82.1
input & forget gates	84.1
forget & output gates	82.6
input, forget, output gates	**85.3**

Backward & Bidirectional LSTMs

bidirectional:

if shallow, just use forward and backward LSTMs in parallel, concatenate final two hidden vectors, feed to softmax

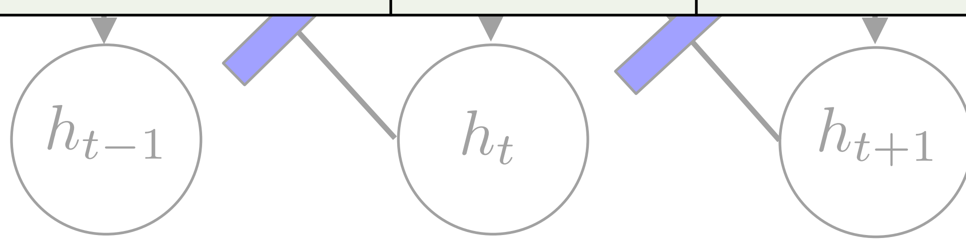


Backward & Bidirectional LSTMs

bidirectional:

- if shallow, just use forward and backward LSTMs in parallel, concatenate final two hidden vectors, feed to softmax

	forward	backward
gateless	80.6	80.3
output gates	81.9	83.7
input gates	84.4	82.9
forget gates	82.1	83.4
input, forget, output gates	85.3	85.9

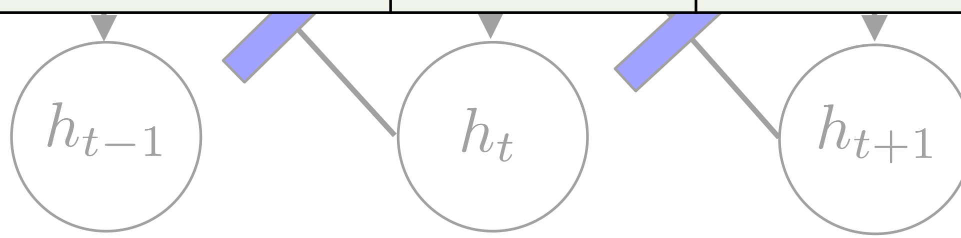


Backward & Bidirectional LSTMs

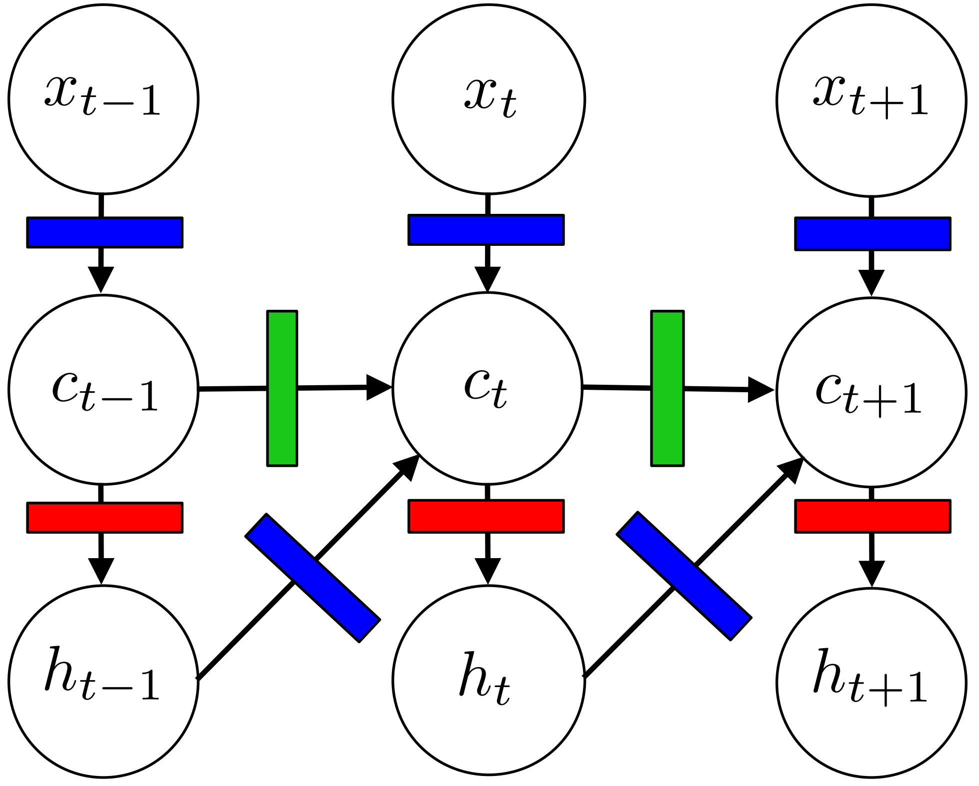
bidirectional:

if shallow, just use forward and backward LSTMs in parallel, concatenate final two hidden vectors, feed to softmax

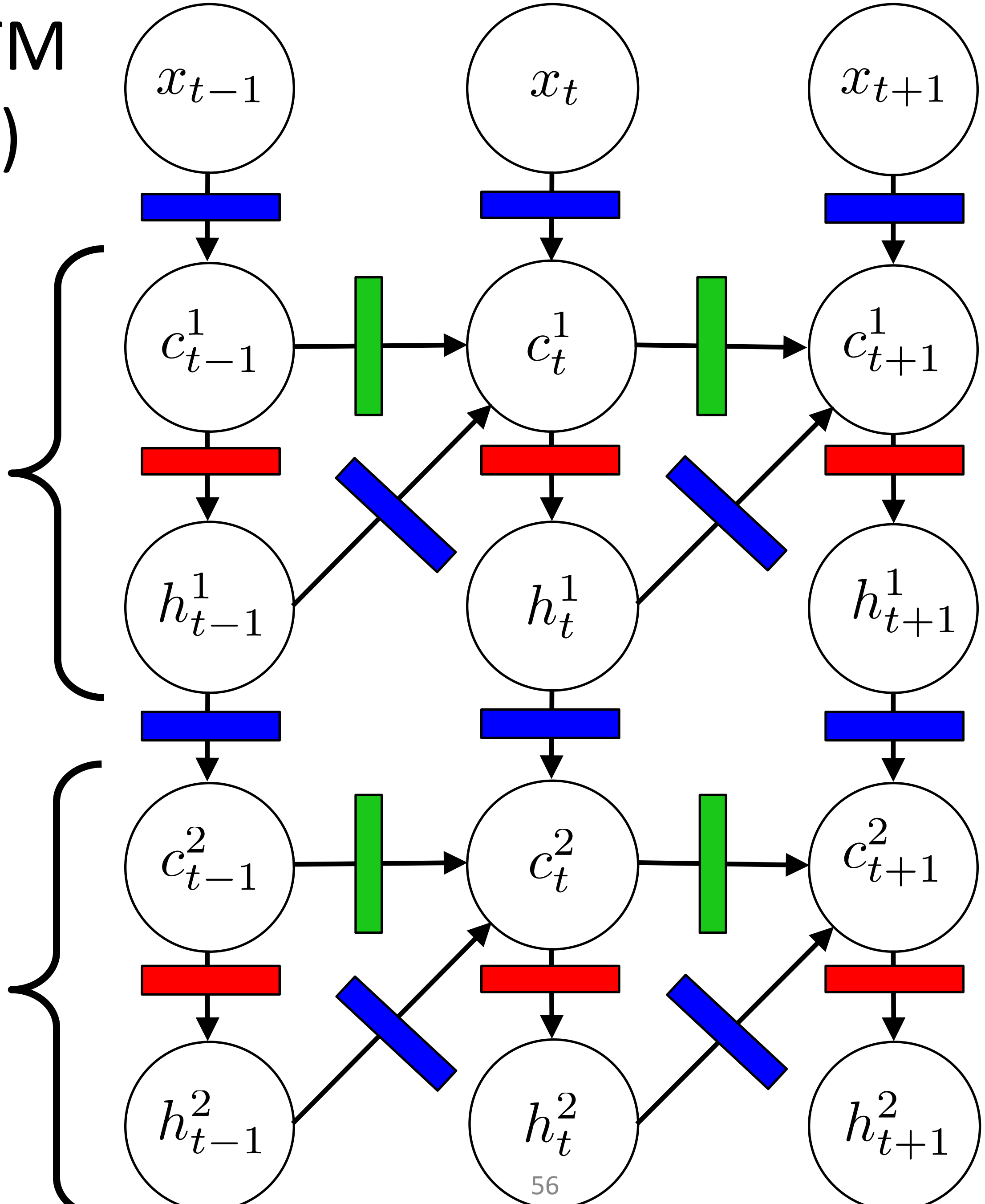
	forward	backward	bidirectional
gateless	80.6	80.3	81.5
output gates	81.9	83.7	82.6
input gates	84.4	82.9	83.9
forget gates	82.1	83.4	83.1
input, forget, output gates	85.3	85.9	85.1



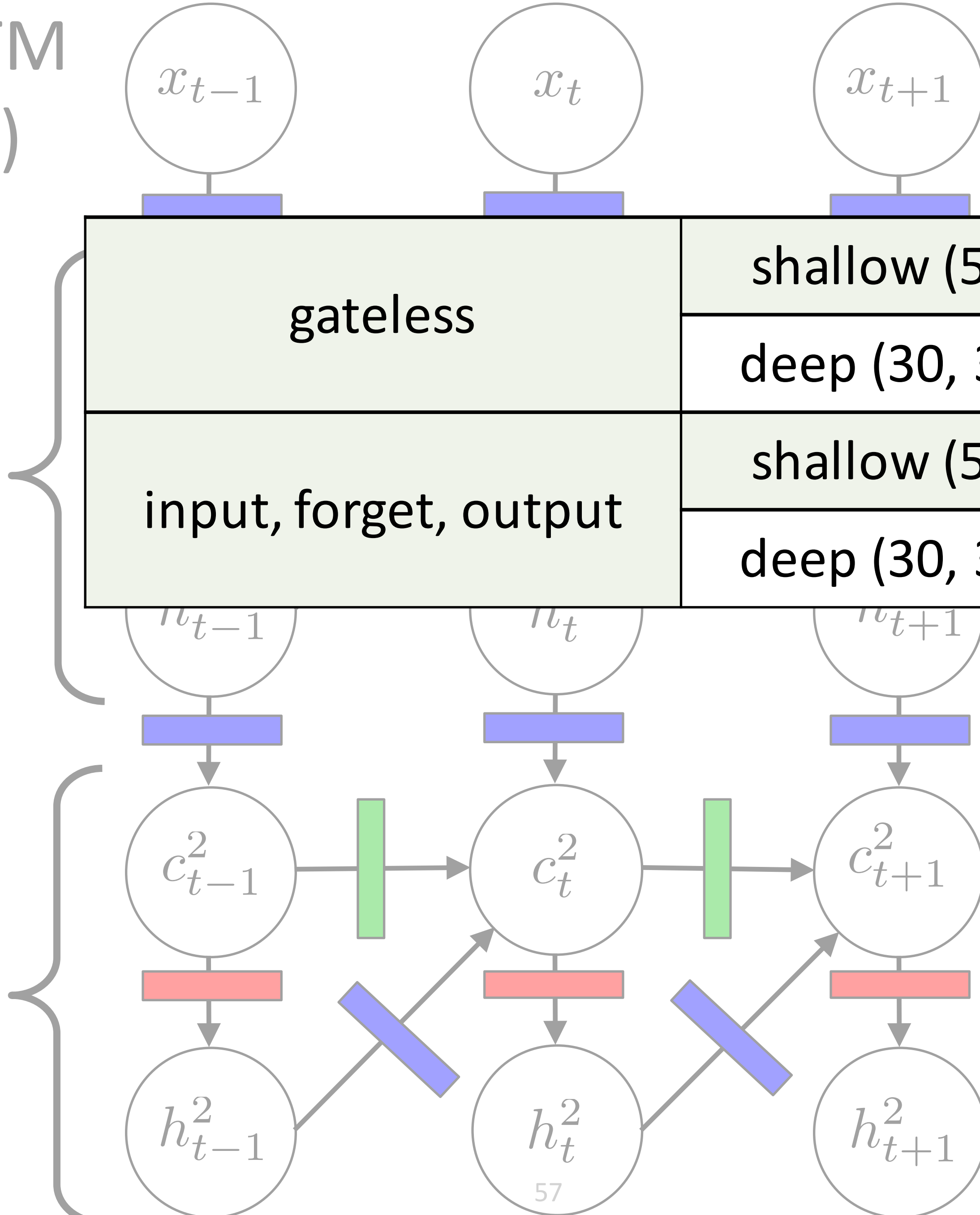
LSTM



Deep LSTM (2-layer)



Deep LSTM (2-layer)



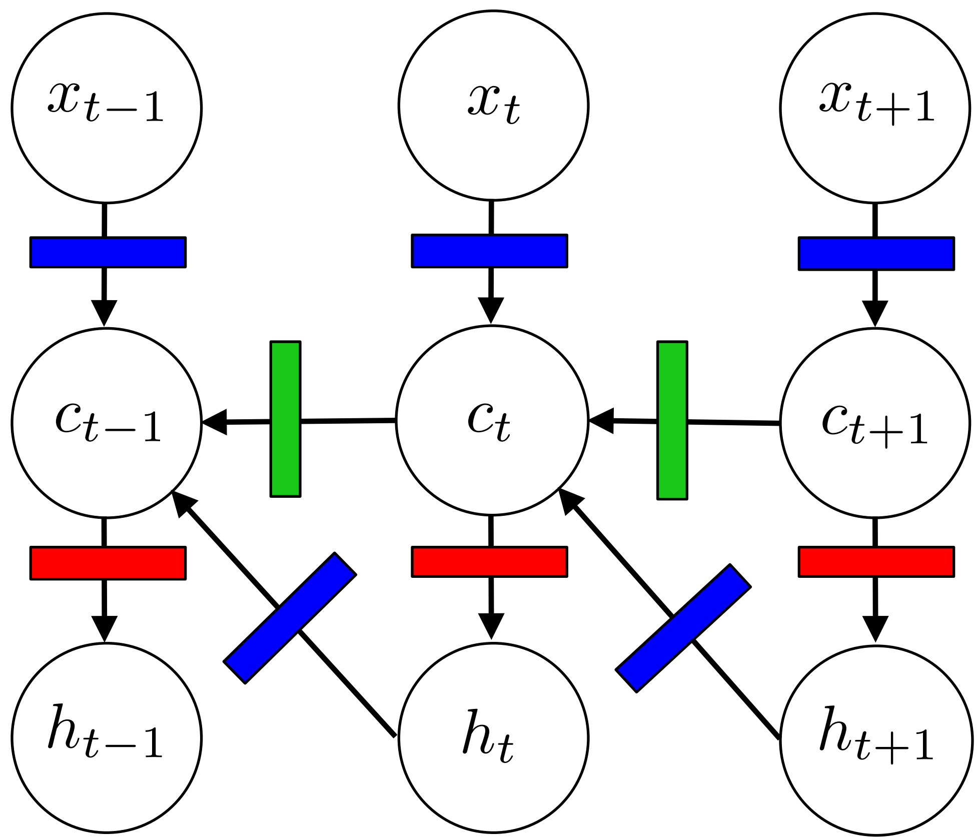
layer 1

layer 2

		acc.
gateless	shallow (50)	80.6
	deep (30, 30)	80.8
input, forget, output	shallow (50)	85.3
	deep (30, 30)	~85

Deep Bidirectional LSTMs

concatenate hidden vectors of forward & backward LSTMs, connect each entry to forward and backward hidden vectors in next layer



(logistic) sigmoid: $$y = \frac{1}{1 + \exp\{-x\}}$$