CSEP 517: Natural Language Processing
Recurrent Neural Networks
Autumn 2018

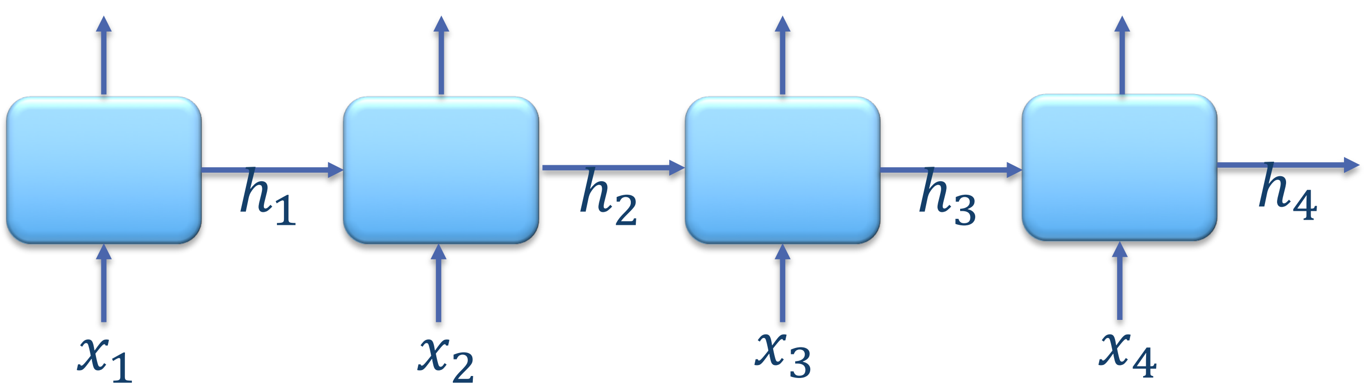
Luke Zettlemoyer
University of Washington

[most slides from Yejin Choi]

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs)

- Each input "word" is a vector $x_t \in R^N$
- Each RNN unit computes a new hidden state using the previous state and a new input $h_t = f(x_t, h_{t-1})$
- Each RNN unit (optionally) makes an output using the current hidden state $y_t = \mathrm{softmax}(V h_t)$
- Hidden states $h_t \in R^D$ are continuous vectors
  - Can represent very rich information, function of entire history
- Parameters are shared (tied) across all RNN units (unlike feedforward NNs)



Softmax

- Turn a vector of real numbers x into a probability distribution

$$softmax(x) = \left[\frac{\exp(x_1)}{\sum_i \exp(x_i)}, \ldots, \frac{\exp(x_n)}{\sum_i \exp(x_i)} \right]$$

- We have seen this trick before!
  - log-linear models…

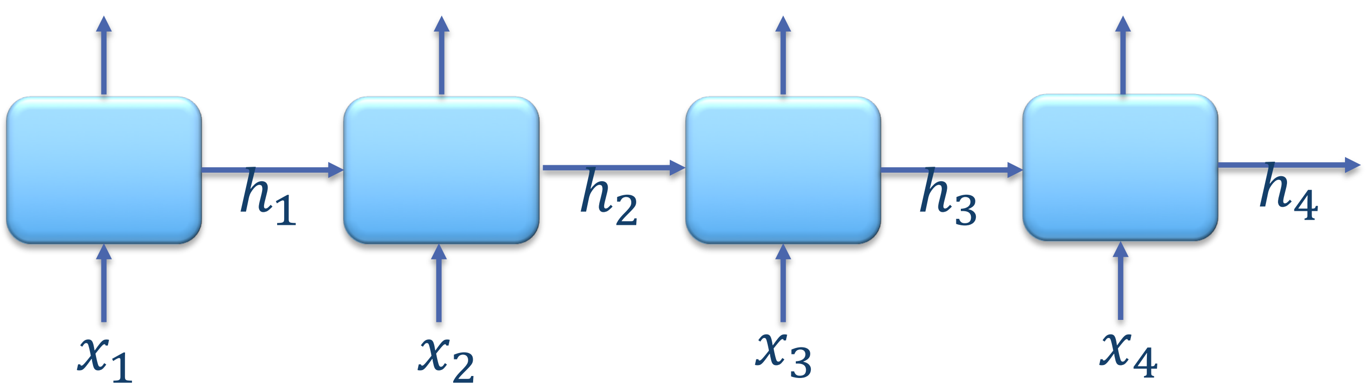
Recurrent Neural Networks (RNNs)

- Generic RNNs:

$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(V h_t)$$

- Vanilla RNN:

$$h_t = \tanh(U x_t + W h_{t-1} + b)$$
$$y_t = \text{softmax}(V h_t)$$

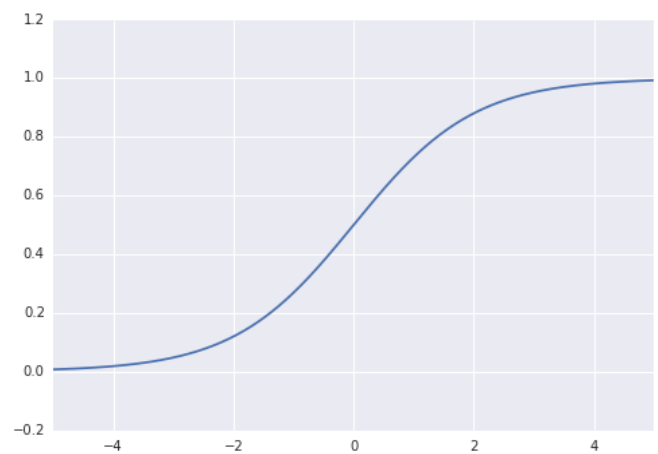


Sigmoid

- Often used for gates
- Pro: neuron-like, differentiable
- Con: gradients saturate to zero almost everywhere except x near zero => vanishing gradients
- Batch normalization helps

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



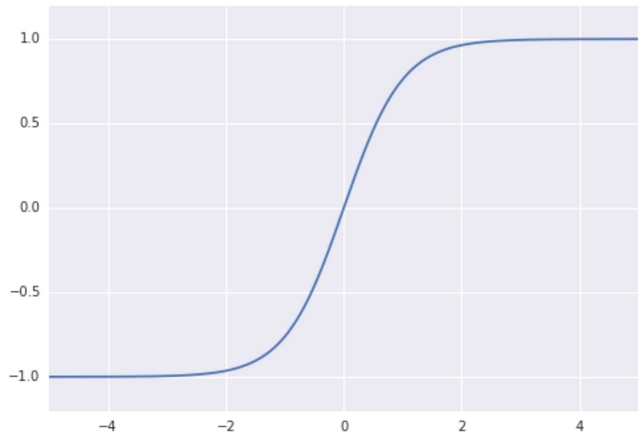
Tanh

- Often used for hidden states & cells in RNNs, LSTMs
- Pro: differentiable, often converges faster than sigmoid
- Con: gradients easily saturate to zero => vanishing gradients

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\text{tanh'(x)} = 1 - \tanh^2(x)$$

$$\tanh(x) = 2\sigma(2x) - 1$$

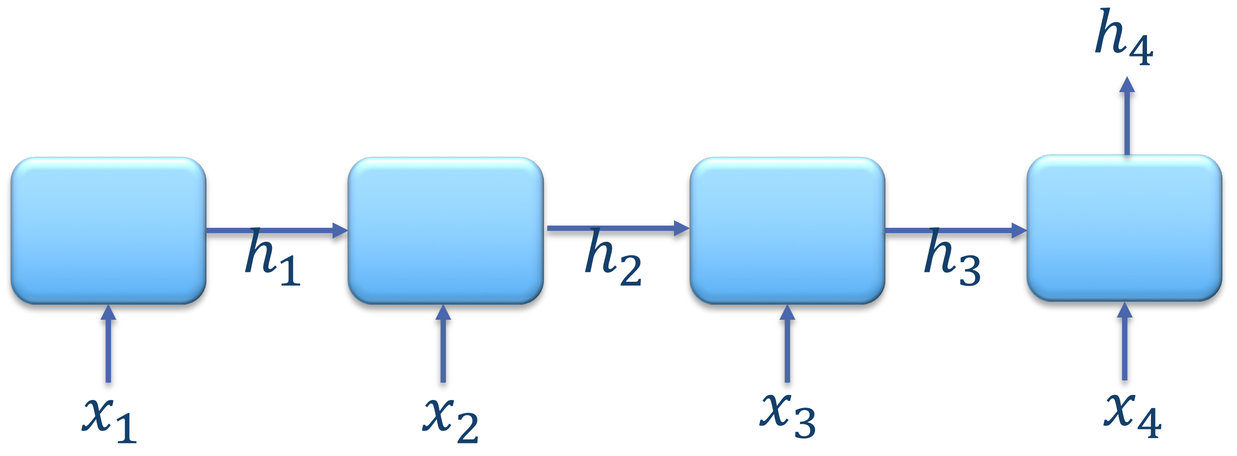


Many uses of RNNs

1. Classification (seq to one)

- Input: a sequence
- Output: one label (classification)
- Example: sentiment classification

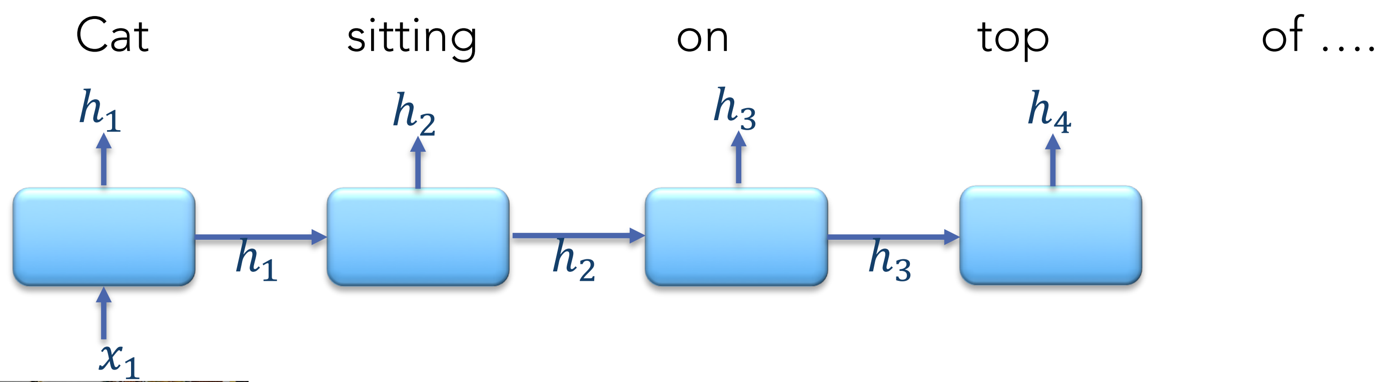
$$h_t = f(x_t, h_{t-1})$$
$$y = \mathrm{softmax}(V h_n)$$



Many uses of RNNs
2. one to seq

- Input: one item
- Output: a sequence
- Example: Image captioning

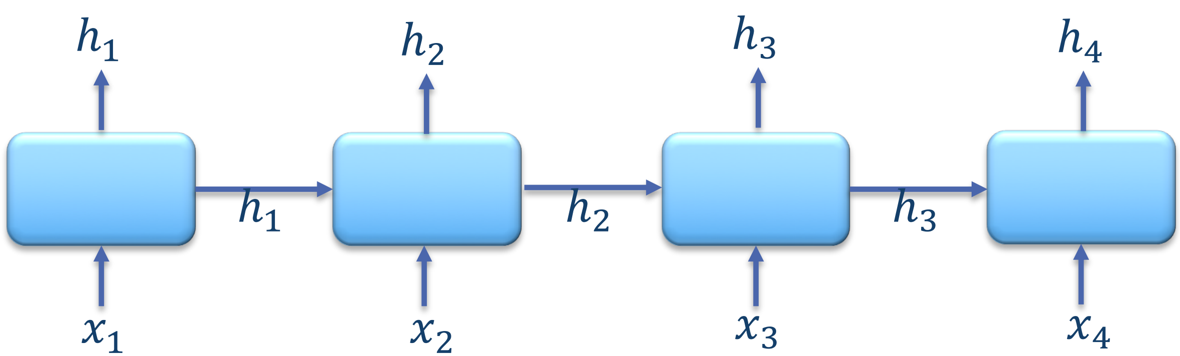
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(V h_t)$$



Many uses of RNNs
3. sequence tagging

- Input: a sequence
- Output: a sequence (of the same length)
- Example: POS tagging, Named Entity Recognition
- How about Language Models?
  - Yes! RNNs can be used as LMs!
  - RNNs make markov assumption: T/F?

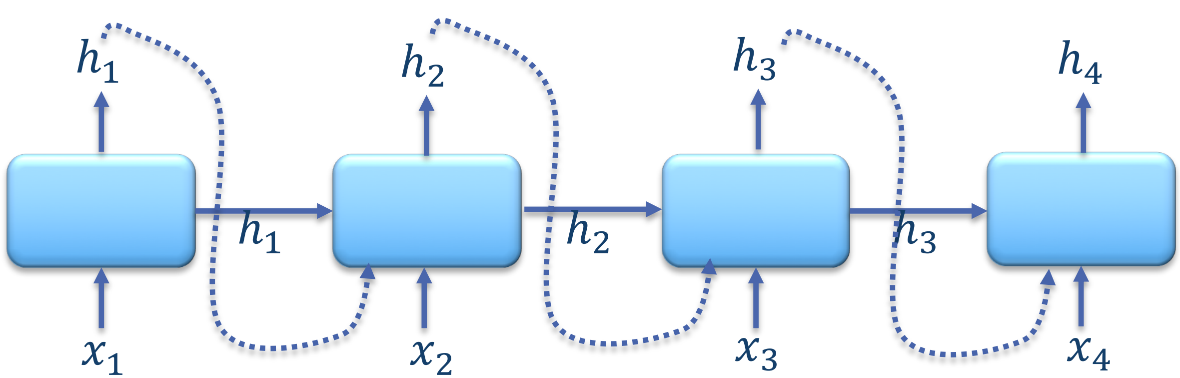
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(V h_t)$$



Many uses of RNNs
4. Language models

$$h_t = f(x_t, h_{t-1})$$
$$y_t = \mathrm{softmax}(V h_t)$$

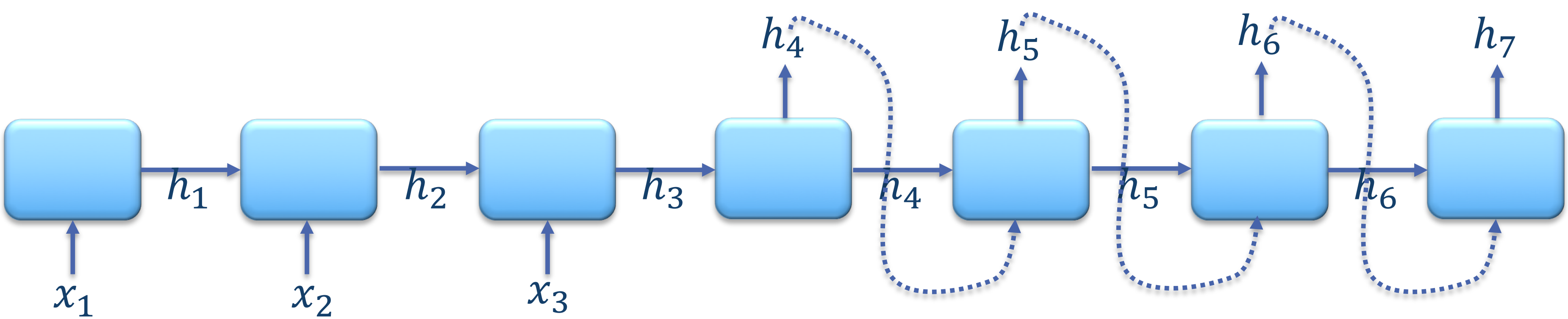
- Input: a sequence of words
- Output: next word
  - (or sequence of next words, if repeated)
- During training, x_t and y_{t-1} are the same word.
- During testing, x_t is sampled from softmax in y_{t-1}.
- Does RNN LMs make Markov assumption?
  - i.e., the next word depends only on the previous N words



Many uses of RNNs
5. seq2seq (aka "encoder-decoder")

- Input: a sequence
- Output: a sequence (of *different* length)
- Examples?

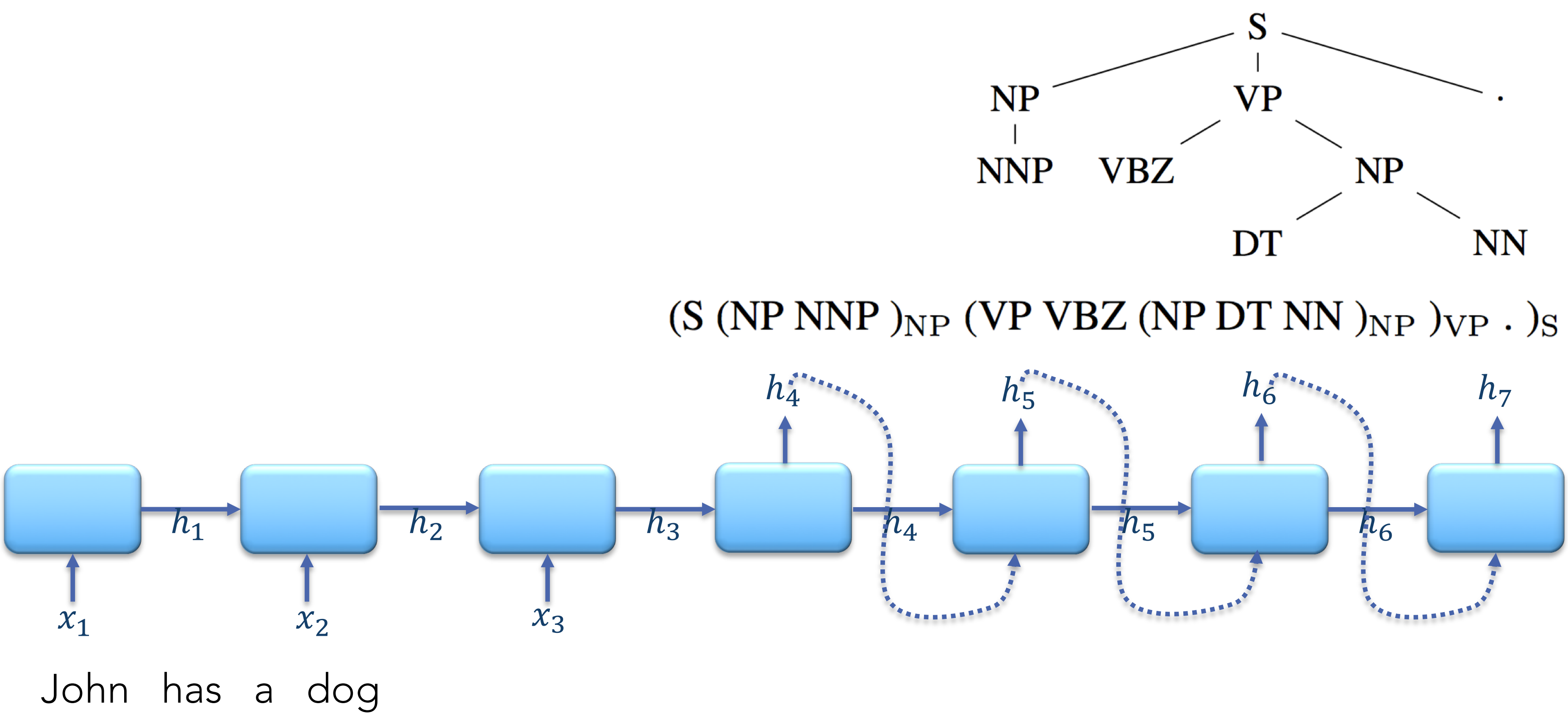
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \mathrm{softmax}(V h_t)$$



Many uses of RNNs
4. seq2seq (aka "encoder-decoder")

Parsing!

- *"Grammar as Foreign Language"* (Vinyals et al., 2015)



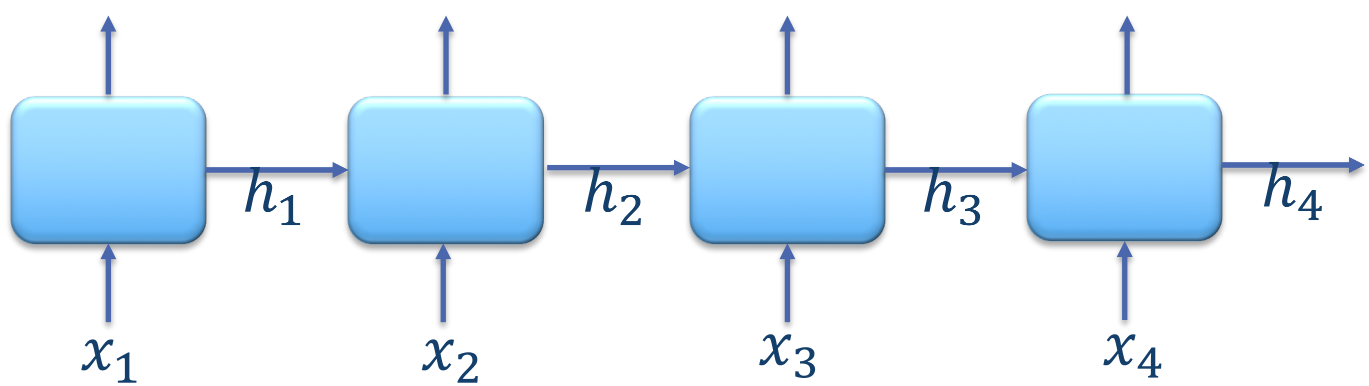
Recurrent Neural Networks (RNNs)

- Generic RNNs:

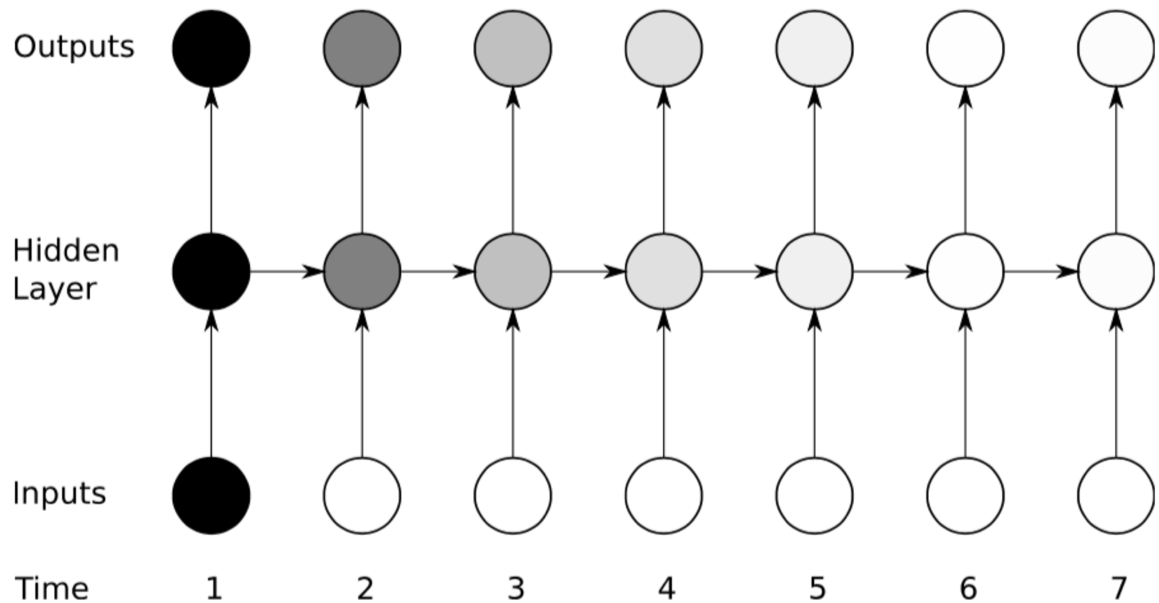
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \mathrm{softmax}(V h_t)$$

- Vanilla RNN:

$$h_t = \tanh(U x_t + W h_{t-1} + b)$$
$$y_t = \mathrm{softmax}(V h_t)$$



vanishing gradient problem for RNNs.



- The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity).
- The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network ‘forgets’ the first inputs.

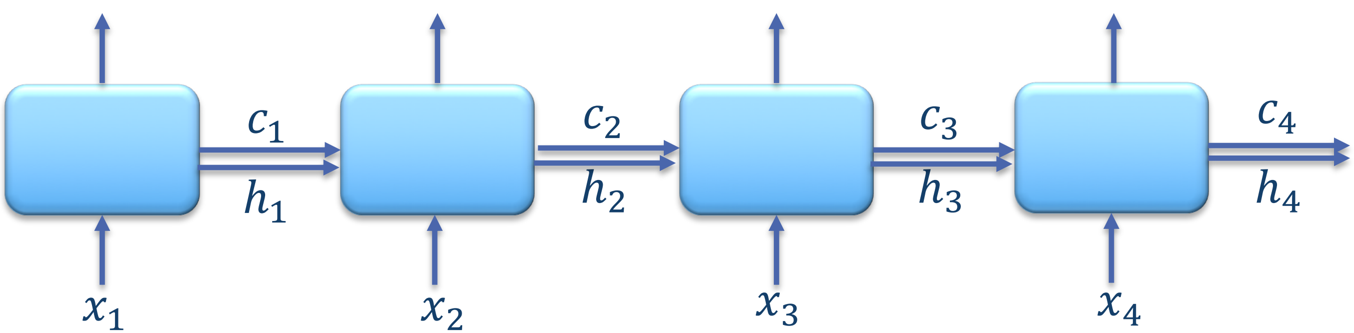
Example from Graves 2012

Recurrent Neural Networks (RNNs)

- Generic RNNs: $h_t = f(x_t, h_{t-1})$
- Vanilla RNNs: $h_t = \tanh(Ux_t + Wh_{t-1} + b)$
- LSTMs (Long Short-term Memory Networks):

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$
$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$
$$o_t = \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)})$$
$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

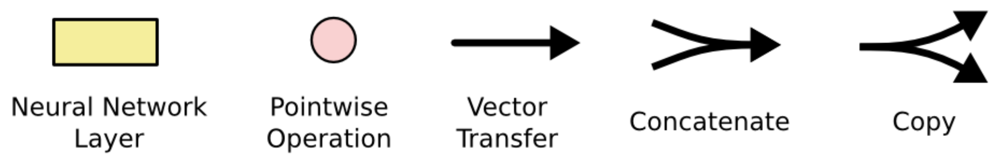
There are many known variations to this set of equations!



c_t : cell state

h_t : hidden state

LSTMS (LONG SHORT-TERM MEMORY NETWORKS



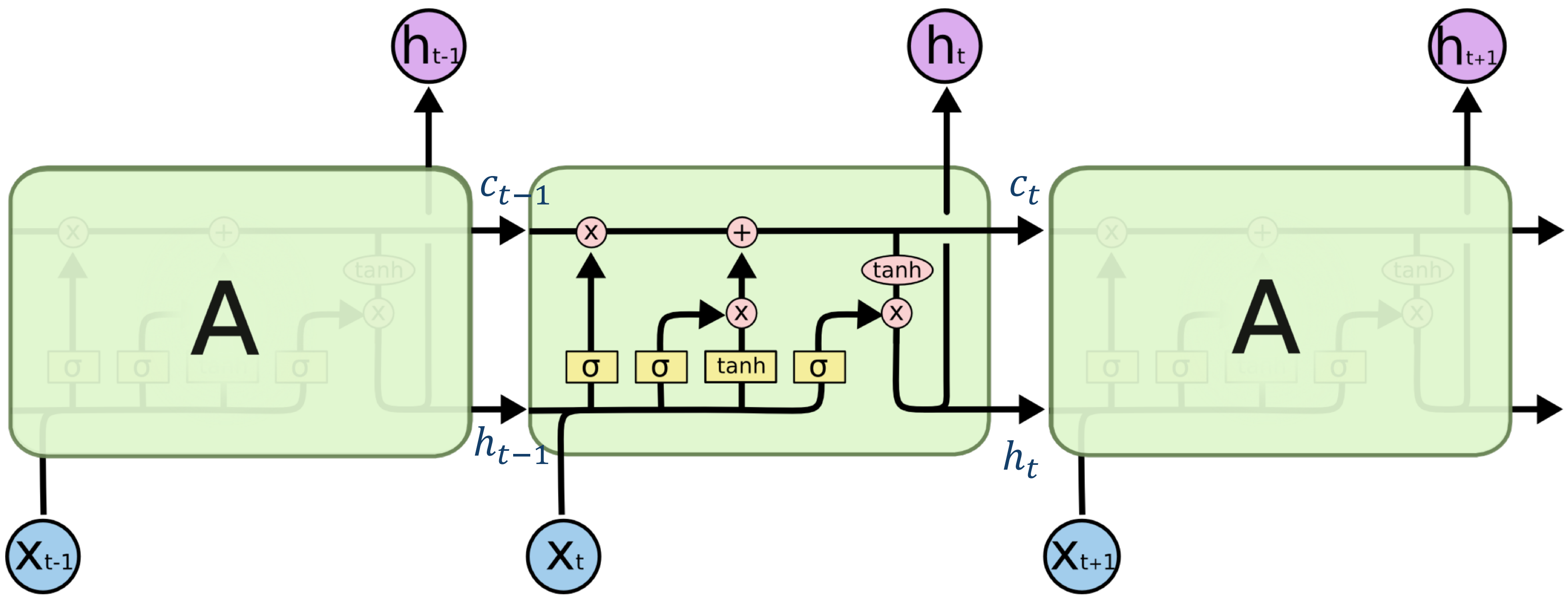
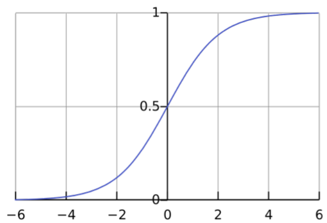


Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid:
[0,1]



Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)} x_t + W^{(f)} h_{t-1} + b^{(f)})$$

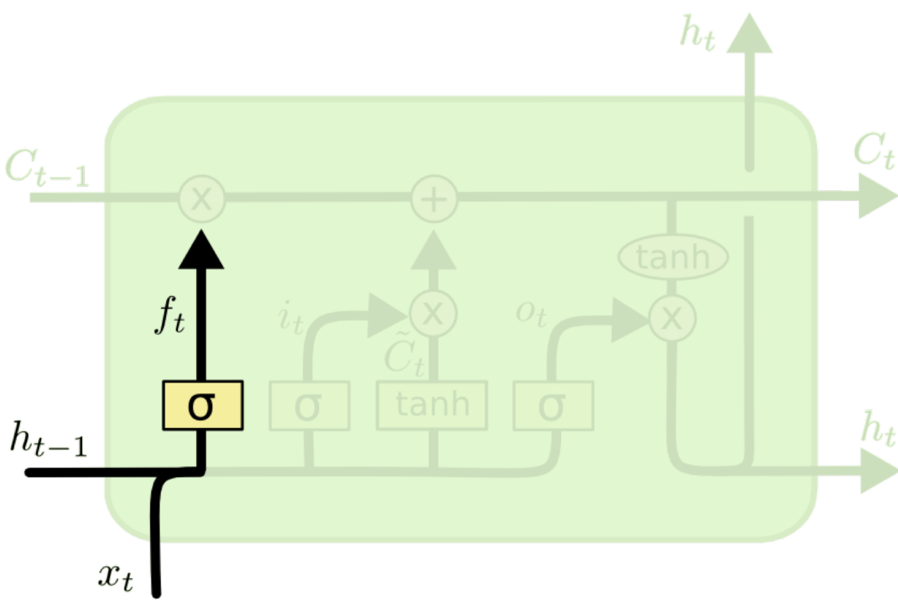
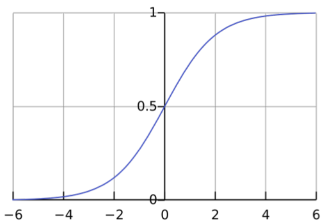


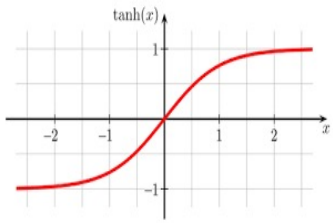
Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid:
[0,1]



tanh:
[-1,1]



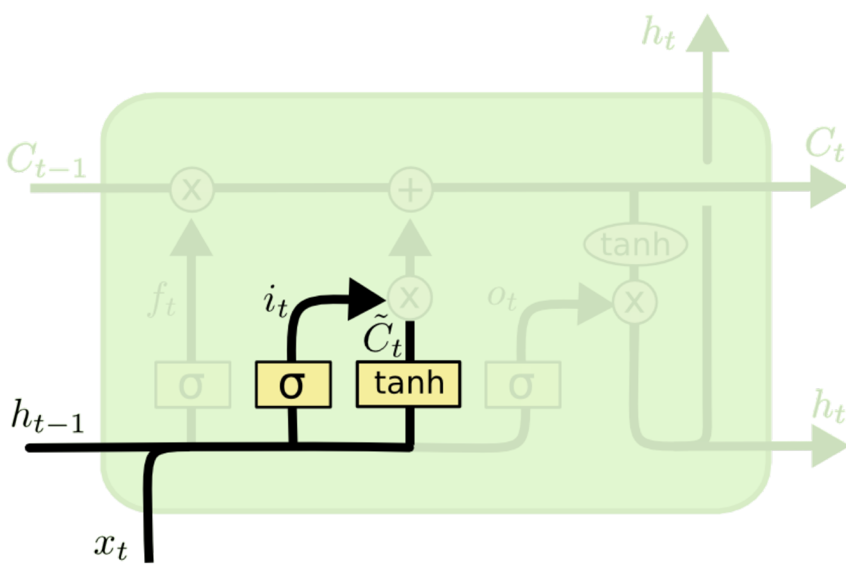


Figure by Christopher Olah (colah.github.io)

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

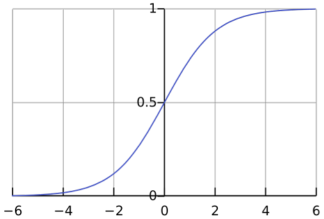
$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

New cell content (temp):

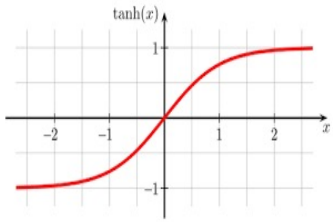
$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid:
[0,1]



tanh:
[-1,1]



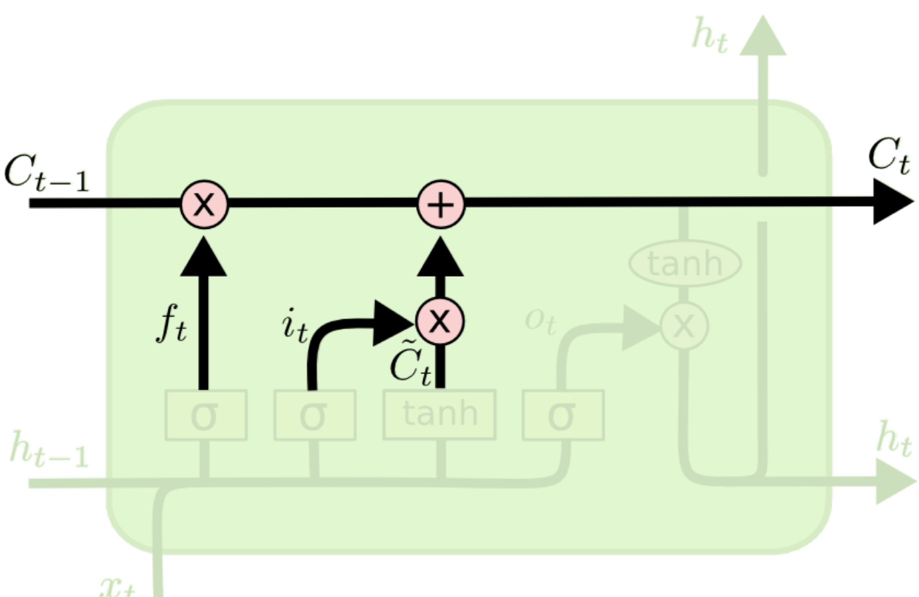


Figure by Christopher Olah (colah.github.io)

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

New cell content (temp):

$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

New cell content:

- mix old cell with the new temp cell

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

Output gate: output from the new cell or not

$$o_t = \sigma(U^{(o)} x_t + W^{(o)} h_{t-1} + b^{(o)})$$

Hidden state:

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

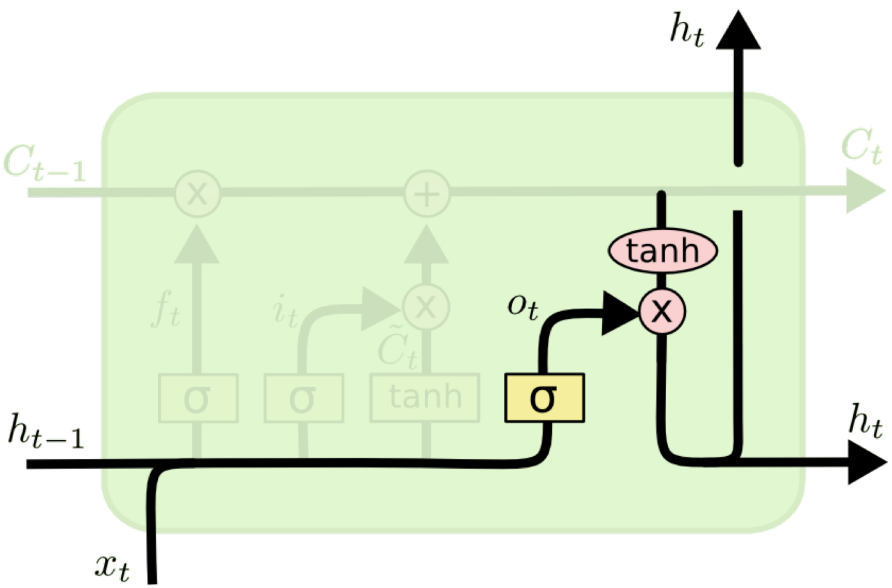


Figure by Christopher Olah (colah.github.io)

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)} x_t + W^{(f)} h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)} x_t + W^{(i)} h_{t-1} + b^{(i)})$$

New cell content (temp):

$$\tilde{c}_t = \tanh(U^{(c)} x_t + W^{(c)} h_{t-1} + b^{(c)})$$

New cell content:

- mix old cell with the new temp cell

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)} x_t + W^{(f)} h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)} x_t + W^{(i)} h_{t-1} + b^{(i)})$$

Output gate: output from the new cell or not

$$o_t = \sigma(U^{(o)} x_t + W^{(o)} h_{t-1} + b^{(o)})$$

New cell content (temp):

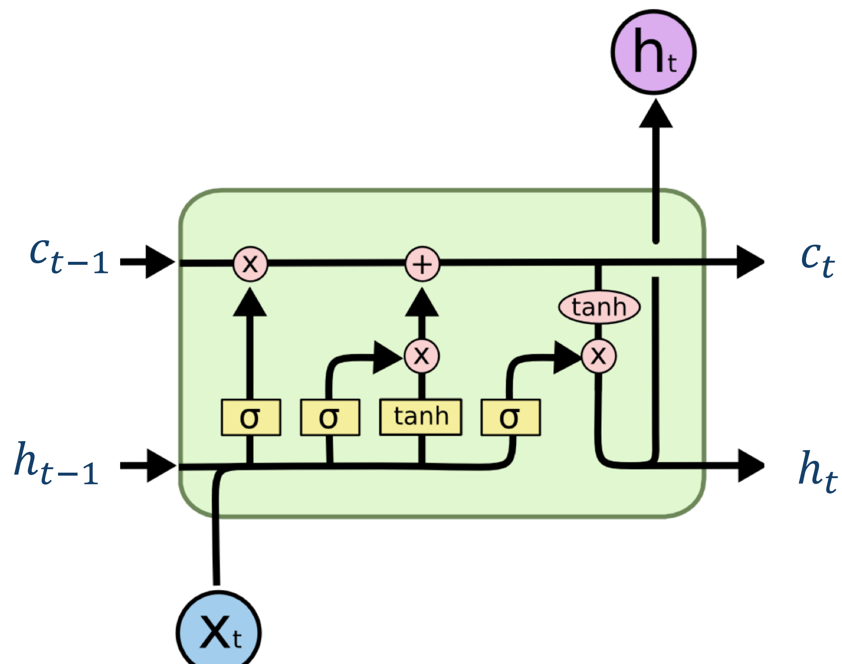
$$\tilde{c}_t = \tanh(U^{(c)} x_t + W^{(c)} h_{t-1} + b^{(c)})$$

New cell content:
- mix old cell with the new temp cell

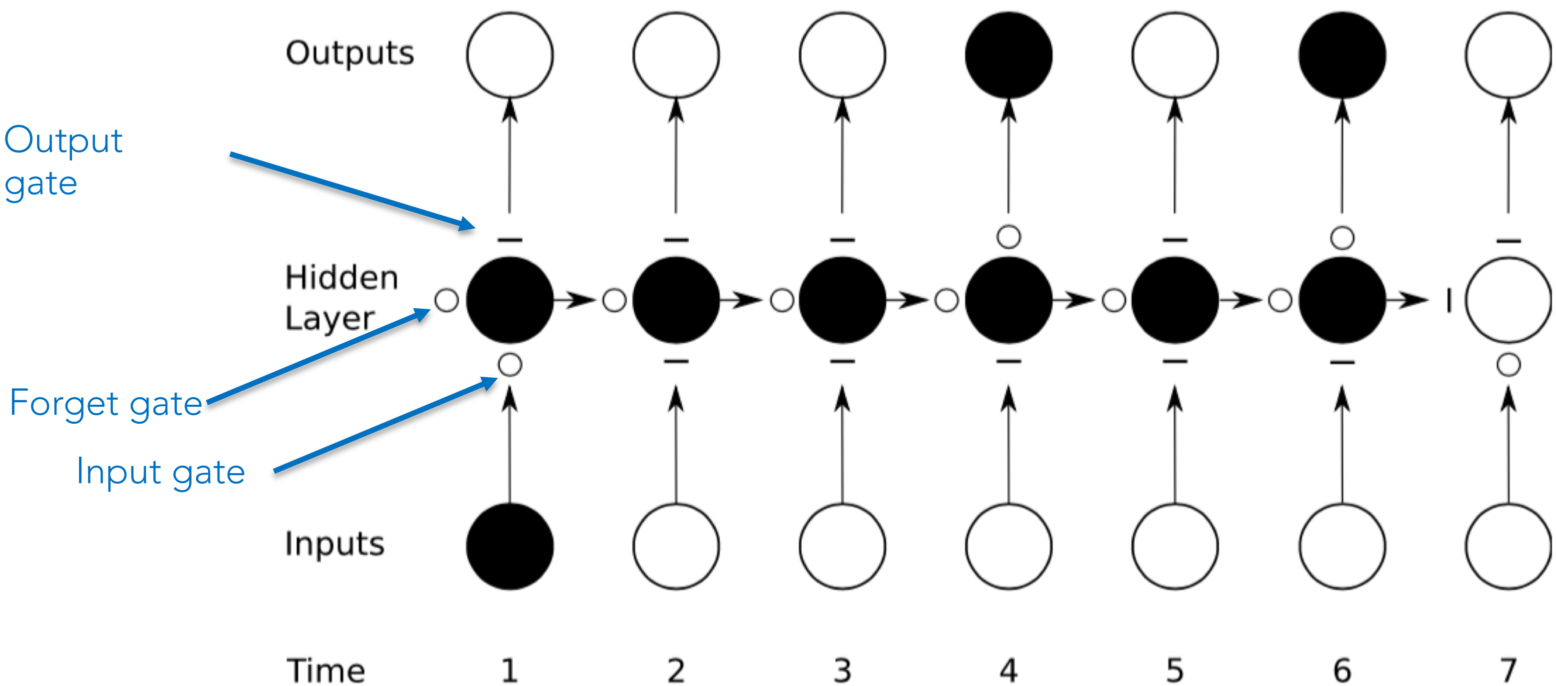
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Hidden state:

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



Preservation of gradient information by LSTM



- For simplicity, all gates are either entirely open ('O') or closed ('—').
- The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed.
- The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

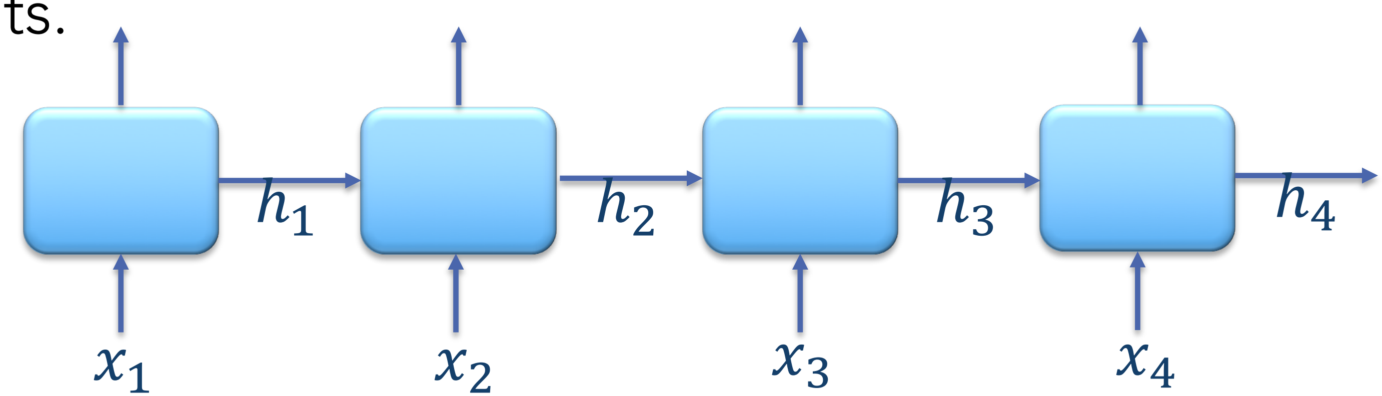
Example from Graves 2012

Gates

- Gates contextually control information flow
- Open/close with sigmoid
- In LSTMs, they are used to (contextually) maintain longer term history

RNN Learning: Backprop Through Time (BPTT)

- Similar to backprop with non-recurrent NNs
- But unlike feedforward (non-recurrent) NNs, each unit in the computation graph repeats the exact same parameters…
- Backprop gradients of the parameters of each unit as if they are different parameters
- When updating the parameters using the gradients, use the average gradients throughout the entire chain of units.



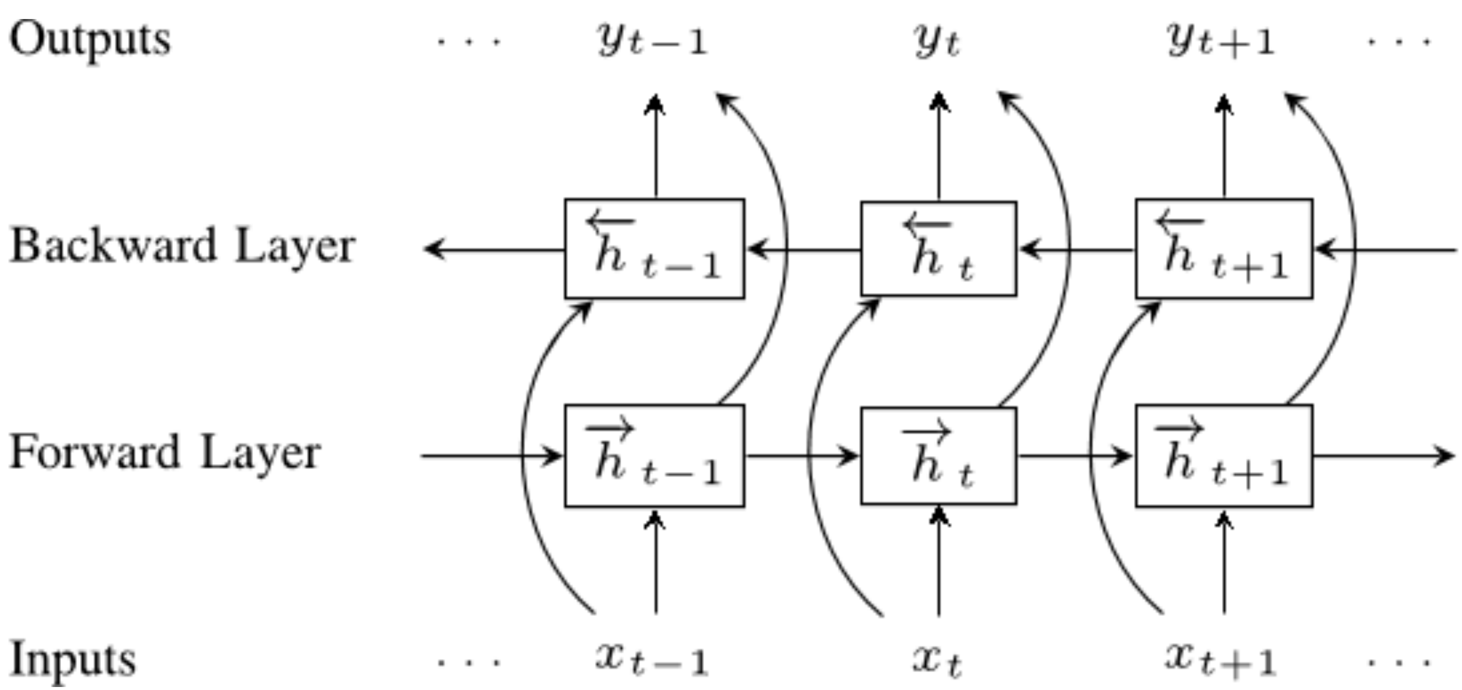
Vanishing / exploding Gradients

- Deep networks are hard to train
- Gradients go through multiple layers
- The multiplicative effect tends to lead to *exploding* or *vanishing* gradients
- Practical solutions w.r.t.
  – network architecture
  – numerical operations

Vanishing / exploding Gradients

- Practical solutions w.r.t. numerical operations
  - Gradient Clipping: bound gradients by a max value
  - Gradient Normalization: renormalize gradients when they are above a fixed norm
  - Careful initialization, smaller learning rates
  - Avoid saturating nonlinearities (like tanh, sigmoid)
    - ReLU or hard-tanh instead
  - Batch Normalization: add intermediate input normalization layers

Sneak peak: Bi-directional RNNs



- Can incorporate context from both directions
- Generally improves over uni-directional RNNs

RNNs make great LMs!

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Table 2. Comparison on 1B word in perplexity (lower the better). Note that Jozefowicz et al., uses 32 GPUs for training. We only use 1 GPU.

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/