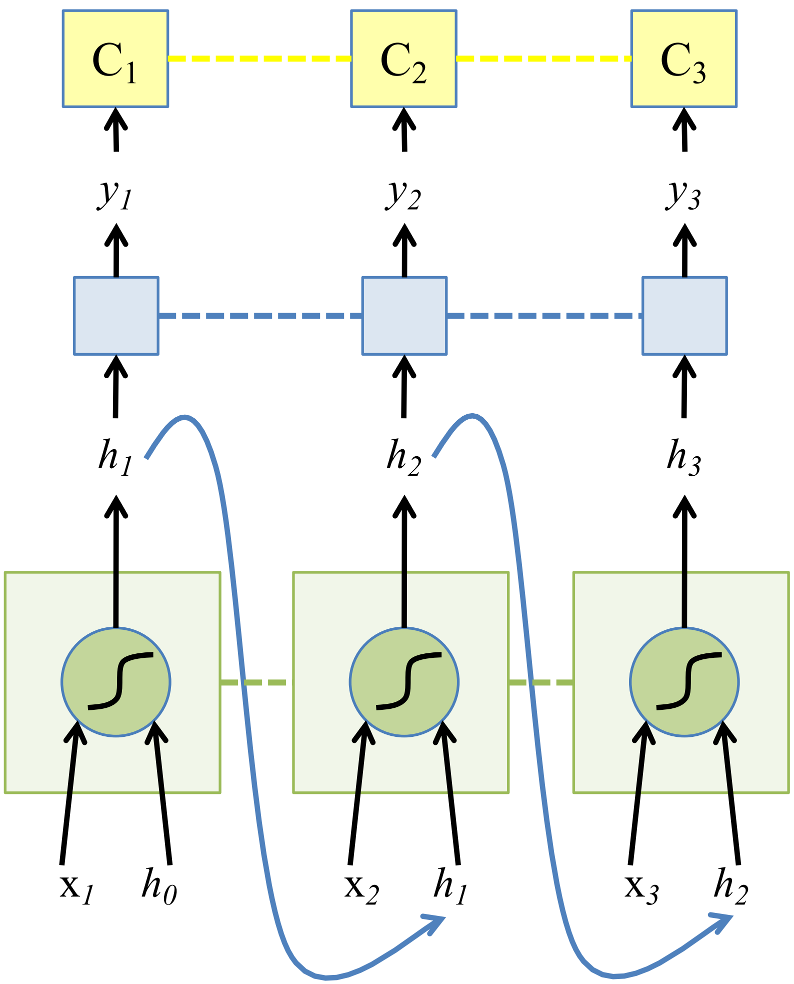
The Vanilla RNN Forward



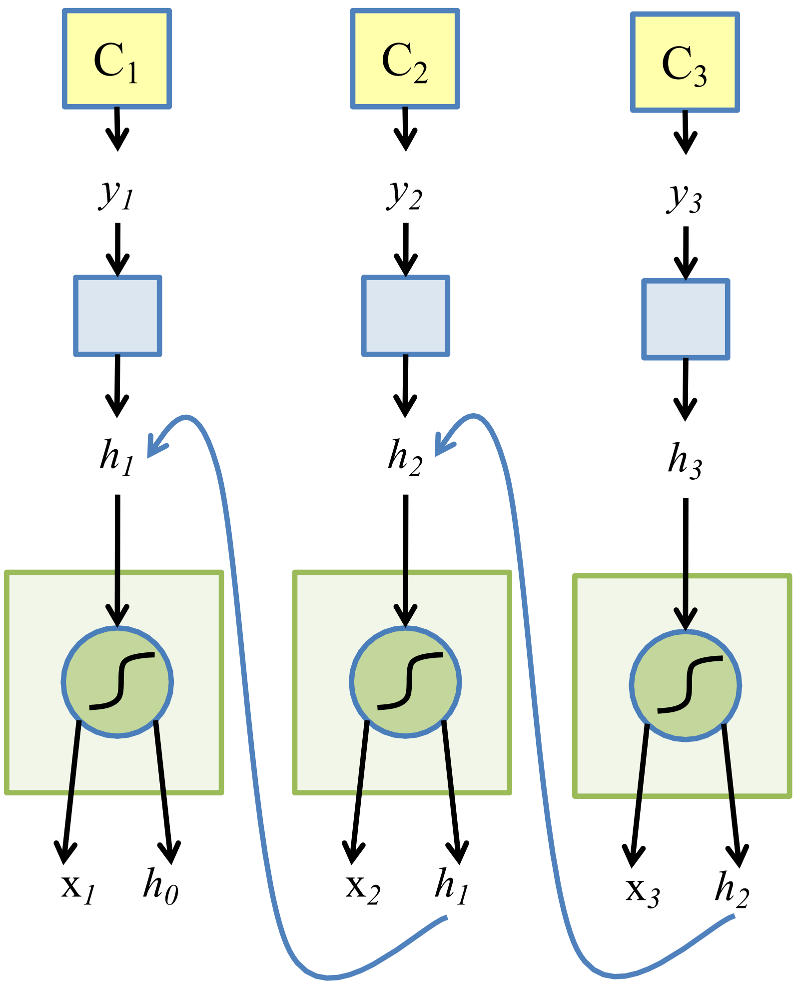
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

------ indicates shared weights

The Vanilla RNN Backward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

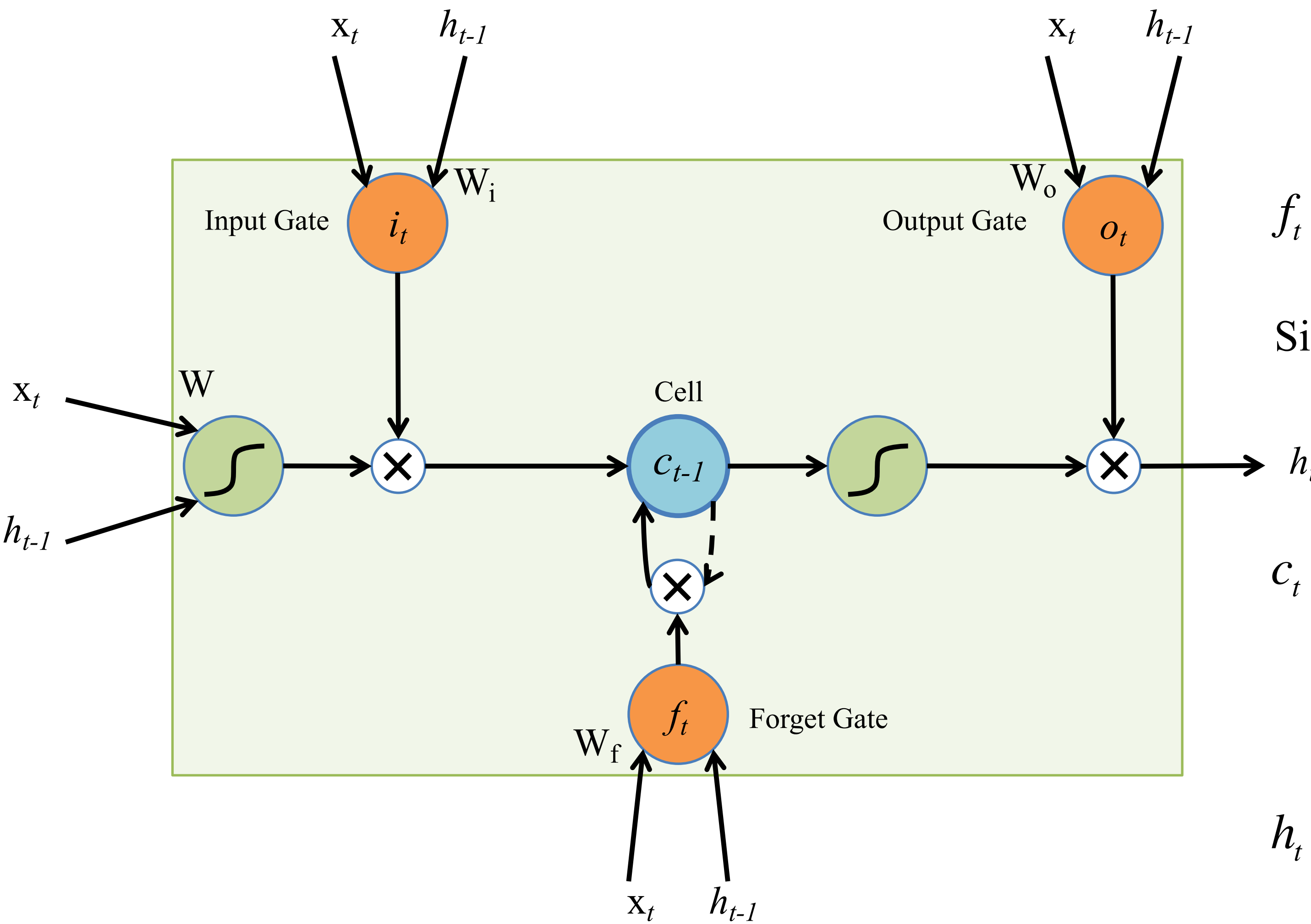
$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

$$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

The Popular LSTM Cell



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

Similarly for i_t, o_t

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

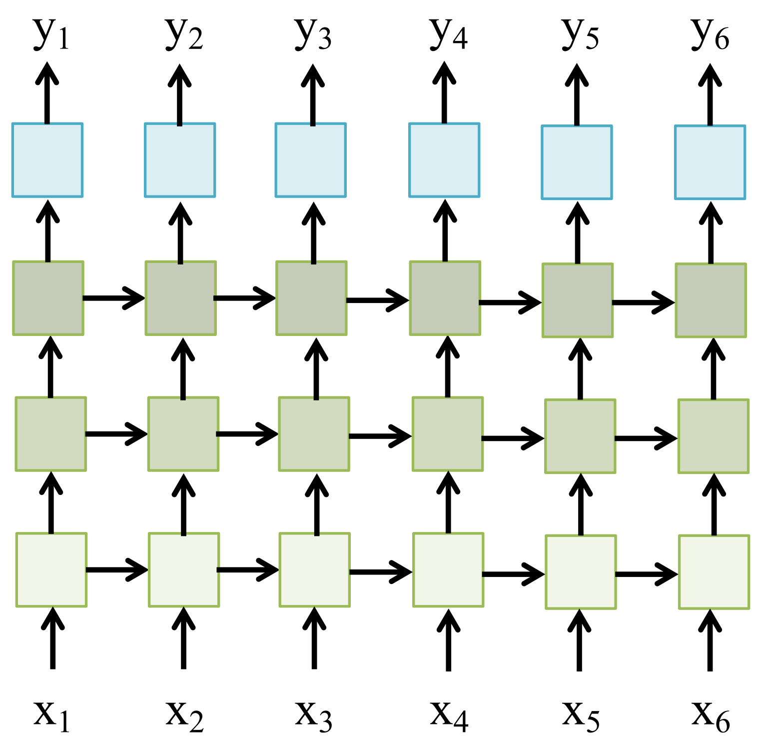
* Dashed line indicates time-lag

LSTM – Forward/Backward

Go To: http://arunmallya.github.io/writeups/nn/lstm/index.html#/

Multi-layer RNNs

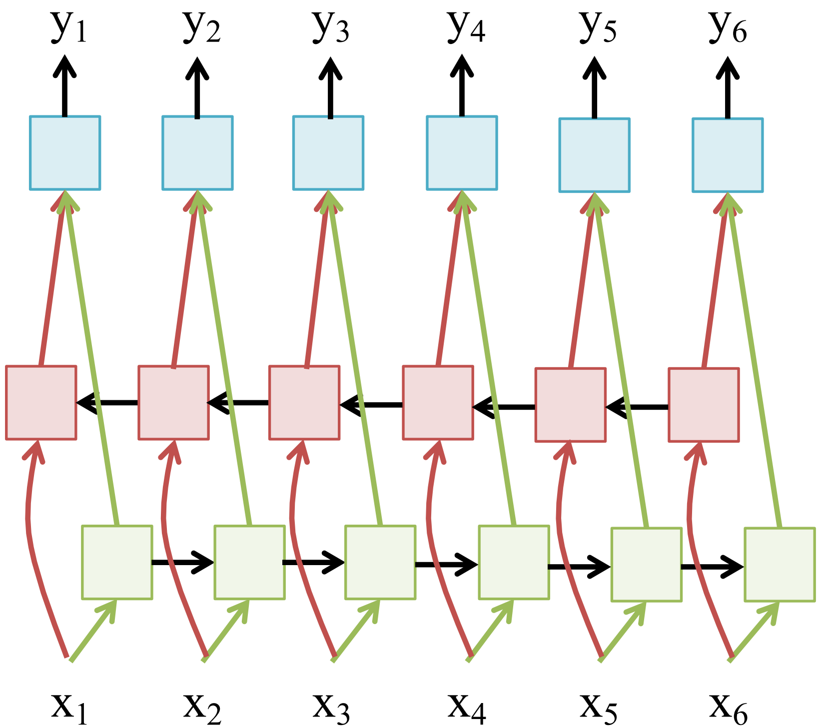
- We can of course design RNNs with multiple hidden layers



- Think exotic: Skip connections across layers, across time, …

Bi-directional RNNs

- RNNs can process the input sequence in forward and in the reverse direction



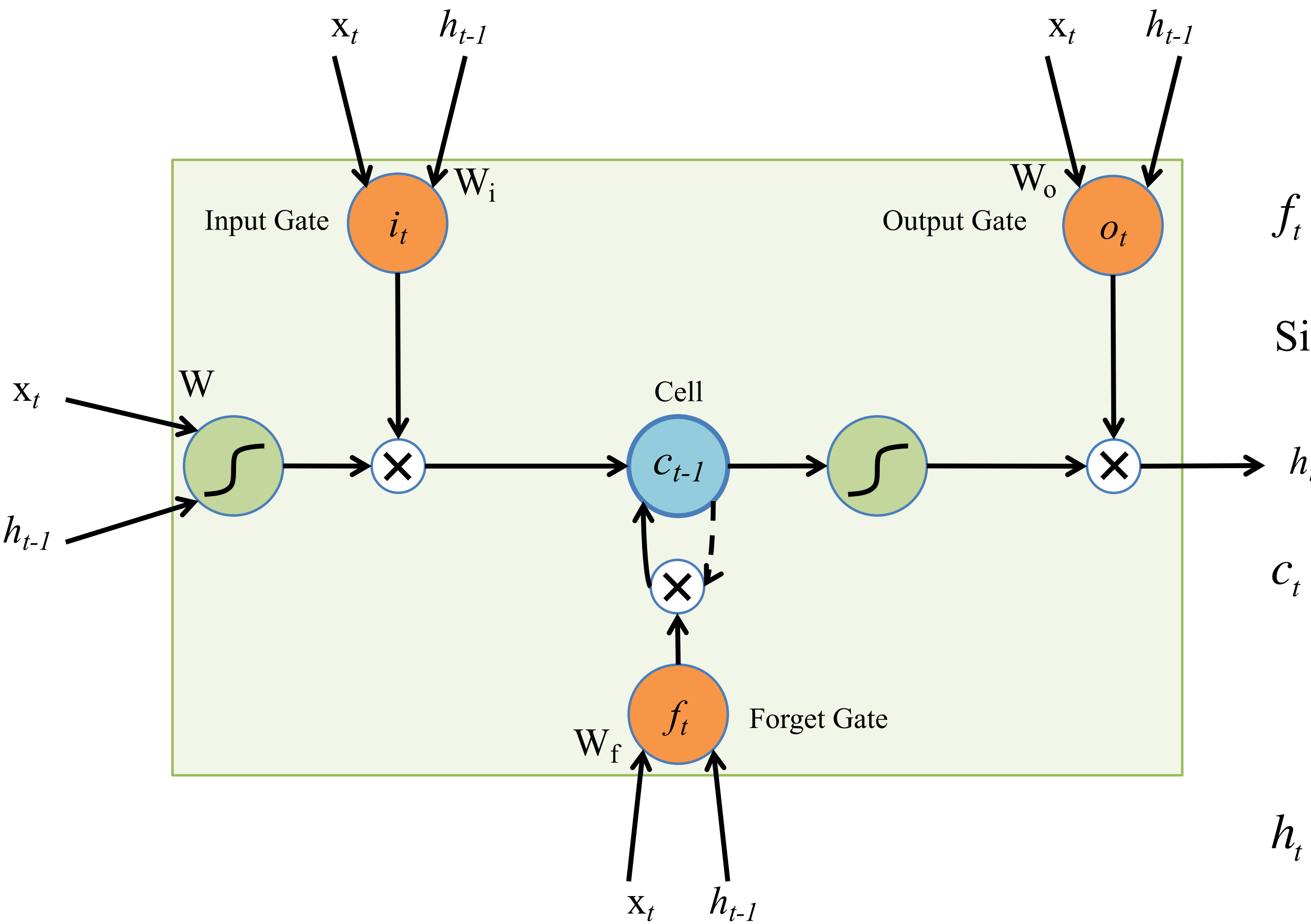
- Popular in speech recognition

Recap

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various Input-Output scenarios are possible (Single/Multiple)
- RNNs can be stacked, or bi-directional

- Vanilla RNNs are improved upon by LSTMs which address the vanishing gradient problem through the CEC
- Exploding gradients are handled by gradient clipping

The Popular LSTM Cell



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

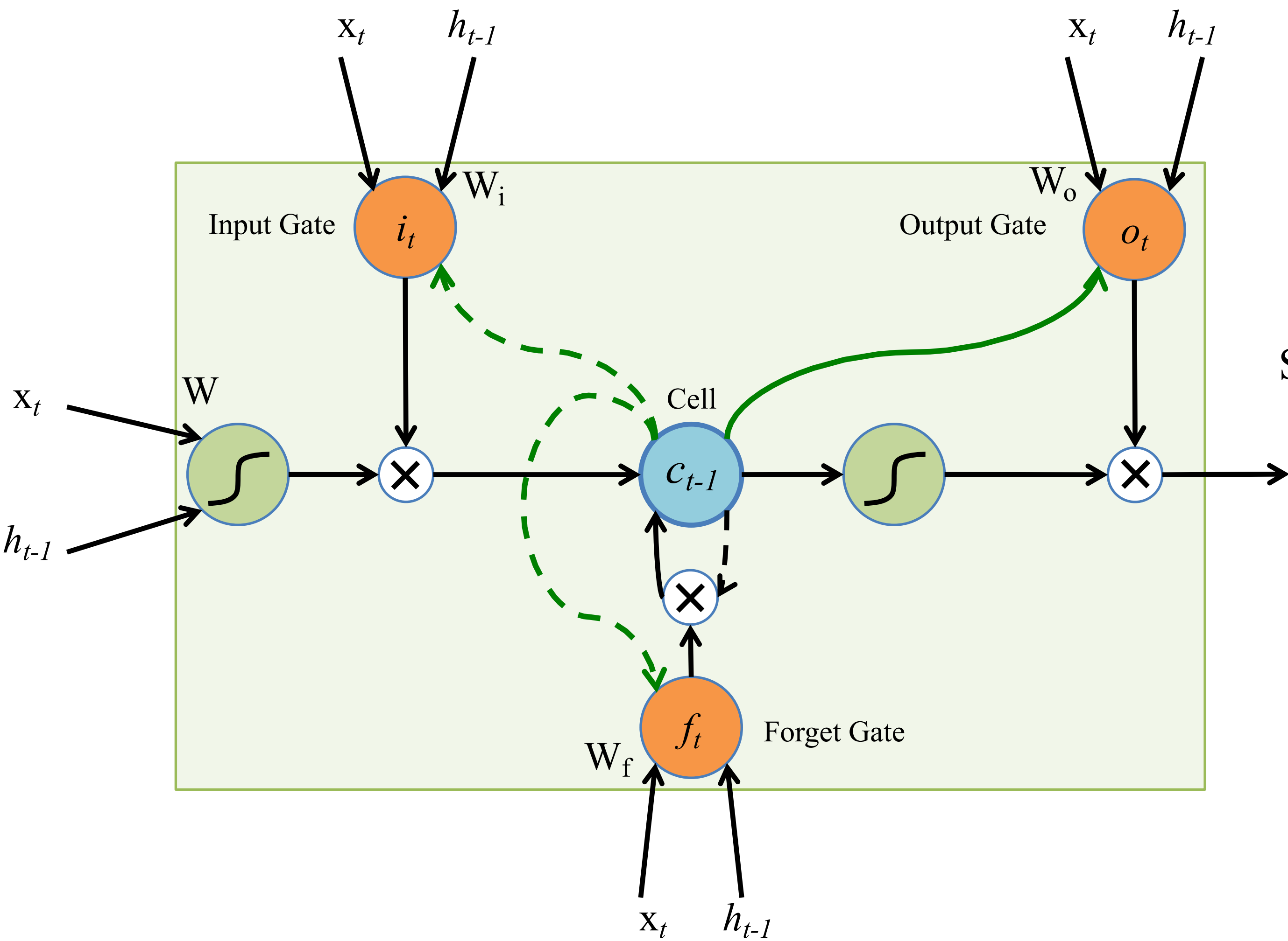
Similarly for i_t, o_t

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

* Dashed line indicates time-lag

Extension I: Peephole LSTM



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \\ c_{t-1} \end{pmatrix} + b_f\right)$$

Similarly for i_t, o_t (uses c_t)

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

* Dashed line indicates time-lag

Peephole LSTM

- Gates can only see the output from the previous time step, which is close to 0 if the output gate is closed. However, these gates control the CEC cell.

- Helped the LSTM learn better timing for the problems tested – Spike timing and Counting spike time delays

Recurrent nets that time and count, Gers *et al*., 2000

Other minor variants

- Coupled Input and Forget Gate $\quad f_t = 1 - i_t$

- Full Gate Recurrence $\quad f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \\ c_{t-1} \\ i_{t-1} \\ f_{t-1} \\ o_{t-1} \end{pmatrix} + b_f \right)$

LSTM: A Search Space Odyssey

- Tested the following variants, using Peephole LSTM as standard:
  1. No Input Gate (NIG)
  2. No Forget Gate (NFG)
  3. No Output Gate (NOG)
  4. No Input Activation Function (NIAF)
  5. No Output Activation Function (NOAF)
  6. No Peepholes (NP)
  7. Coupled Input and Forget Gate (CIFG)
  8. Full Gate Recurrence (FGR)
- On the tasks of:
  - Timit Speech Recognition: Audio frame to 1 of 61 phonemes
  - IAM Online Handwriting Recognition: Sketch to characters
  - JSB Chorales: Next-step music frame prediction

LSTM: A Search Space Odyssey, Greff *et al*., 2015

LSTM: A Search Space Odyssey

- The standard LSTM performed reasonably well on multiple datasets and none of the modifications significantly improved the performance
- Coupling gates and removing peephole connections simplified the LSTM without hurting performance much
- The forget gate and output activation are crucial

- Found interaction between learning rate and network size to be minimal – indicates calibration can be done using a small network first

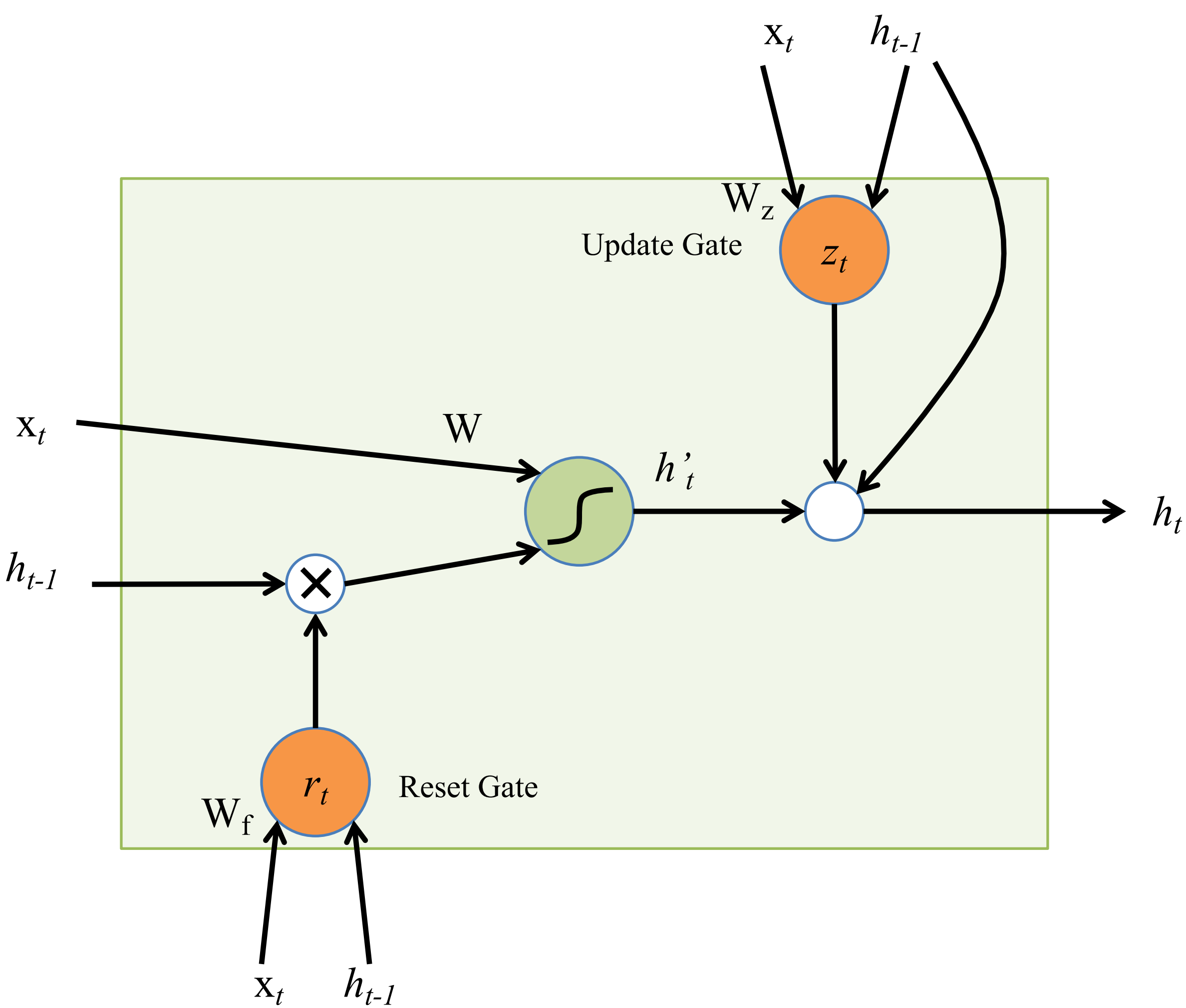
LSTM: A Search Space Odyssey, Greff *et al*., 2015

Gated Recurrent Unit (GRU)

- A very simplified version of the LSTM
  - Merges forget and input gate into a single ‘update’ gate
  - Merges cell and hidden state
- Has fewer parameters than an LSTM and has been shown to outperform LSTM on some tasks

[Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, Cho *et al*., 2014]

GRU



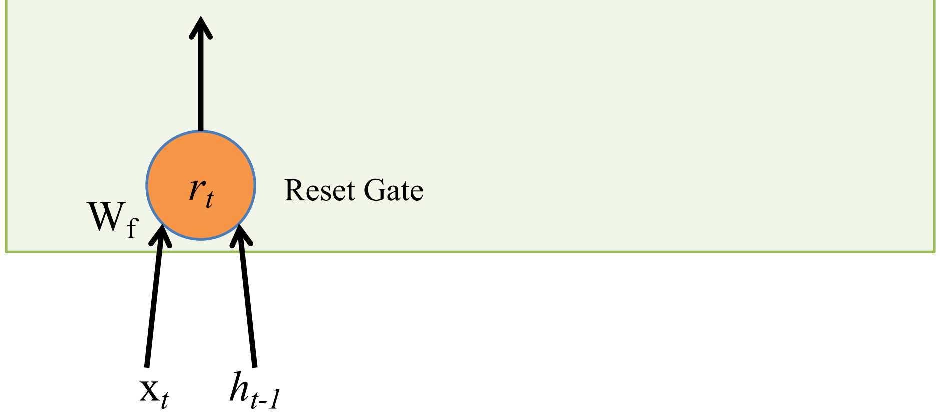
$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

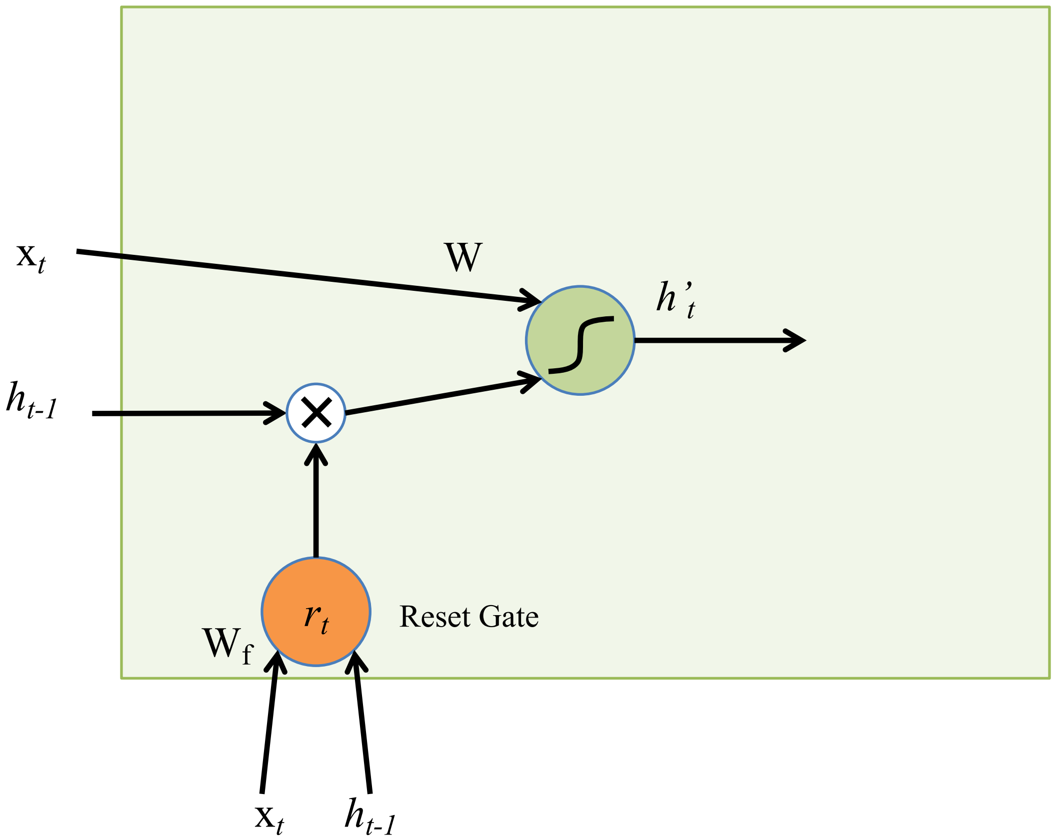
$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

GRU



$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_f\right)$$

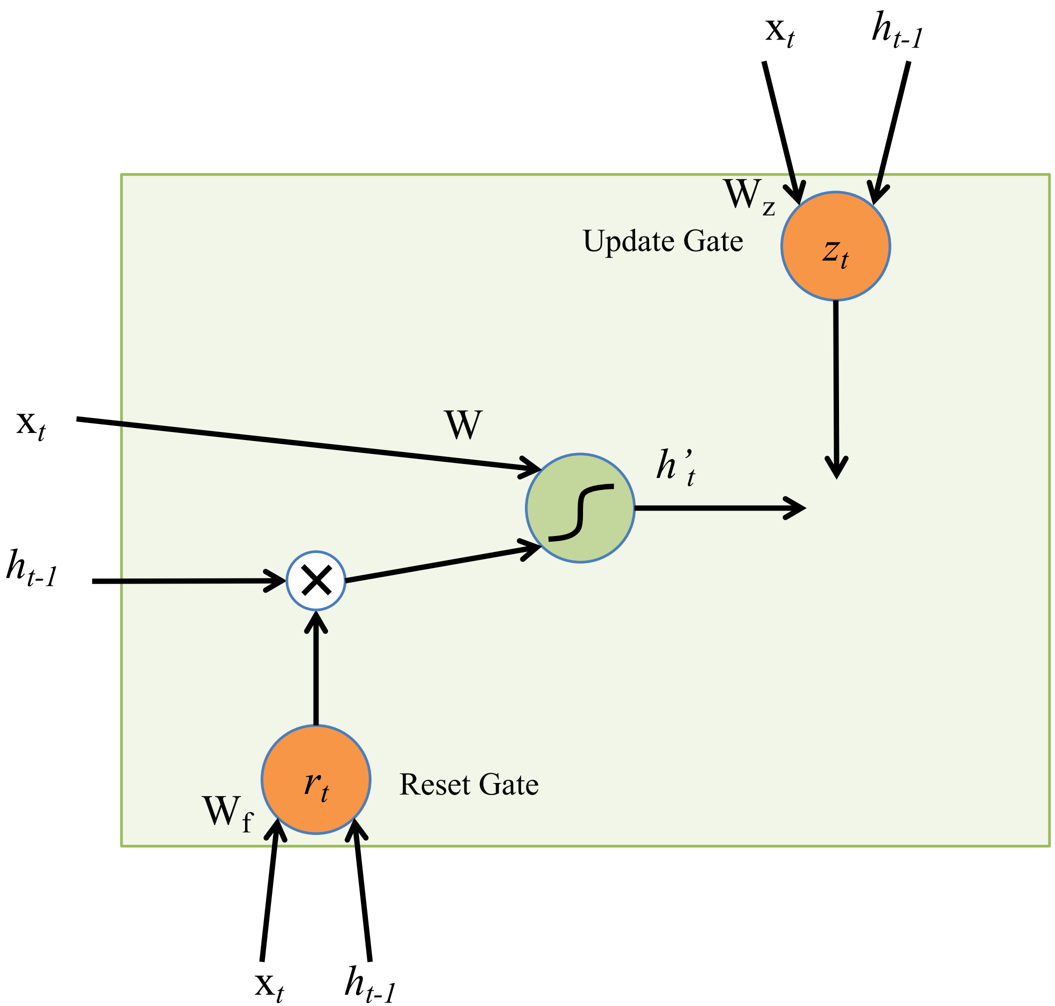
GRU



$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

GRU

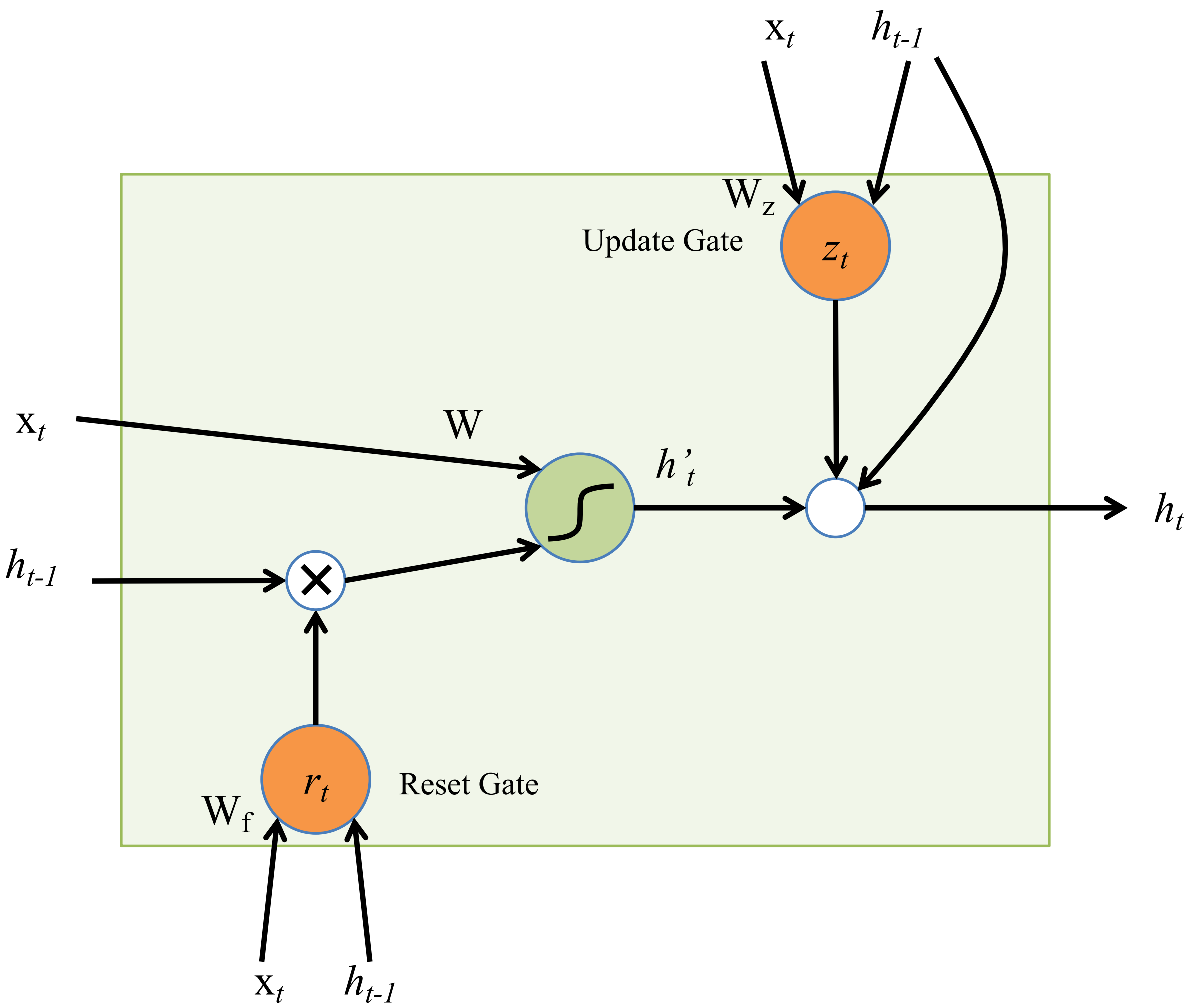


$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

GRU



$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

An Empirical Exploration of Recurrent Network Architectures

- Given the rather ad-hoc design of the LSTM, the authors try to determine if the architecture of the LSTM is optimal

- They use an evolutionary search for better architectures

An Empirical Exploration of Recurrent Network Architectures, Jozefowicz *et al.*, 2015

Evolutionary Architecture Search

- A list of top-100 architectures so far is maintained, initialized with the LSTM and the GRU
- The GRU is considered as the baseline to beat
- New architectures are proposed, and retained based on performance ratio with GRU

- All architectures are evaluated on 3 problems
  - Arithmetic: Compute digits of sum or difference of two numbers provided as inputs. Inputs have distractors to increase difficulty
    3e**36**d**9-**h**1**h**39**f**94**eeh**43**keg**3**c = 3369 – 13994433 = -13991064
  - XML Modeling: Predict next character in valid XML modeling
  - Penn Tree-Bank Language Modeling: Predict distributions over words

[An Empirical Exploration of Recurrent Network Architectures, Jozefowicz *et al*., 2015]

Evolutionary Architecture Search

- At each step
  - Select 1 architecture at random, evaluate on 20 randomly chosen hyperparameter settings.
  - Alternatively, propose a new architecture by mutating an existing one. Choose probability *p* from [0,1] uniformly and apply a transformation to each node with probability *p*
    - If node is a non-linearity, replace with {tanh(x), sigmoid(x), ReLU(x), Linear(0, x), Linear(1, x), Linear(0.9, x), Linear(1.1, x)}
    - If node is an elementwise op, replace with {multiplication, addition, subtraction}
    - Insert random activation function between node and one of its parents
    - Replace node with one of its ancestors (remove node)
    - Randomly select a node (node A). Replace the current node with either the sum, product, or difference of a random ancestor of the current node and a random ancestor of A.
  - Add architecture to list based on minimum relative accuracy wrt GRU on 3 different tasks

An Empirical Exploration of Recurrent Network Architectures, Jozefowicz *et al.*, 2015

Evolutionary Architecture Search

- 3 novel architectures are presented in the paper
- Very similar to GRU, but slightly outperform it

- LSTM initialized with a large positive forget gate bias outperformed both the basic LSTM and the GRU!

An Empirical Exploration of Recurrent Network Architectures, Jozefowicz *et al.*, 2015

LSTM initialized with large positive forget gate bias?

- Recall

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$\delta c_{t-1} = \delta c_t \otimes f_t$$

- Gradients will vanish if f is close to 0. Using a large positive bias ensures that f has values close to 1, especially when training begins
- Helps learn long-range dependencies
- Originally stated in Learning to forget: Continual prediction with LSTM, Gers *et al.*, 2000, but forgotten over time

An Empirical Exploration of Recurrent Network Architectures, Jozefowicz *et al.*, 2015

Summary

- LSTMs can be modified with Peephole Connections, Full Gate Recurrence, etc. based on the specific task at hand

- Architectures like the GRU have fewer parameters than the LSTM and might perform better

- An LSTM with large positive forget gate bias works best!

Other Useful Resources / References

- http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
- http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

- R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013
- S. Hochreiter, and J. Schmidhuber, Long short-term memory, Neural computation, 1997 9(8), pp.1735-1780
- F.A. Gers, and J. Schmidhuber, Recurrent nets that time and count, IJCNN 2000
- K. Greff , R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems, 2016
- K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014
- R. Jozefowicz, W. Zaremba, and I. Sutskever, An empirical exploration of recurrent network architectures, JMLR 2015