Long Short-Term Memory (LSTM)

A brief introduction

Daniel Renshaw

24th November 2014

Context and notation

- Just to give the LSTM something to do: neural network language modelling
- Vocabulary, size V
- $\mathbf{x}_t \in \mathbb{R}^V$: true word in position t (one-hot)
- $\mathbf{y}_t \in \mathbb{R}^V$: predicted word in position t (distribution)
- Assume all sentences zero padded to length L

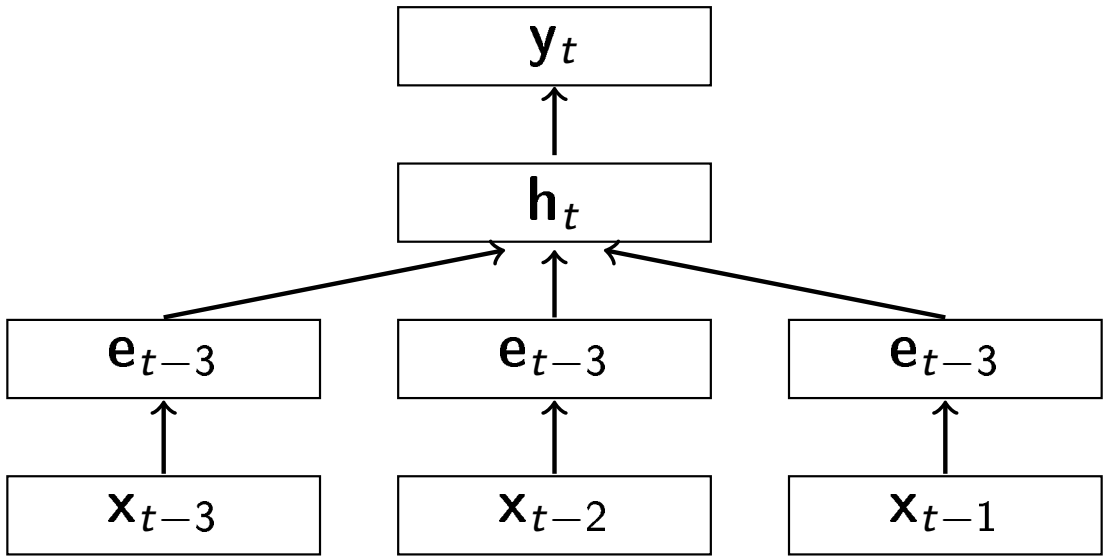
Context and notation

- Model: $\mathbf{y}_{t+1} = p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1\right)$ for $1 \leq t < L$
- Minimize cross-entropy objective:

$$J = \sum_{t=1}^{L-1} H\left(\mathbf{y}_{t+1}, \mathbf{x}_t\right) = \sum_{t=1}^{L-1} \sum_{i}^{V} x_{t,i} log\left(y_{t+1,i}\right)$$

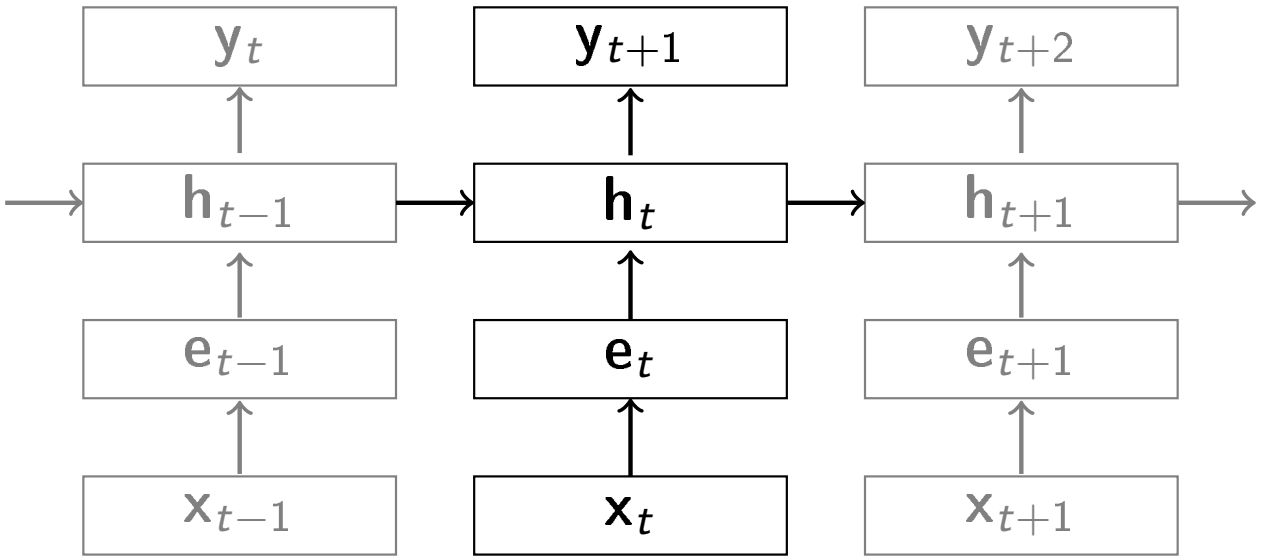
- $\sigma\left(\bullet\right)$ is some sigmoid-like function (e.g. *logistic* or *tanh*)
- $\mathbf{b}^{\bullet}$ is a bias vector, $W^{\bullet}$ is a weight vector

Multi-Layer Perceptron (MLP)



$$
\begin{aligned}
\mathbf{y}_{t+1} &= softmax\left(W^{yh}\mathbf{h}_t\right) \\
\mathbf{h}_t &= \sigma\left(W^{he}\left[\mathbf{e}_{t-1};\mathbf{e}_{t-2};\mathbf{e}_{t-3}\right]+\mathbf{b}^h\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

Recurrent Neural Network (RNN)



$$\mathbf{y}_{t+1} = softmax\left(W^{yh}\mathbf{h}_t\right)$$

$$\mathbf{h}_t = \sigma\left(W^{he}\mathbf{e}_t + W^{hh}\mathbf{h}_{t-1} + \mathbf{b}^h\right)$$

$$\mathbf{e}_t = W^{ex}\mathbf{x}_t$$

Vanishing gradients

- Error gradients pass through nonlinearity every step

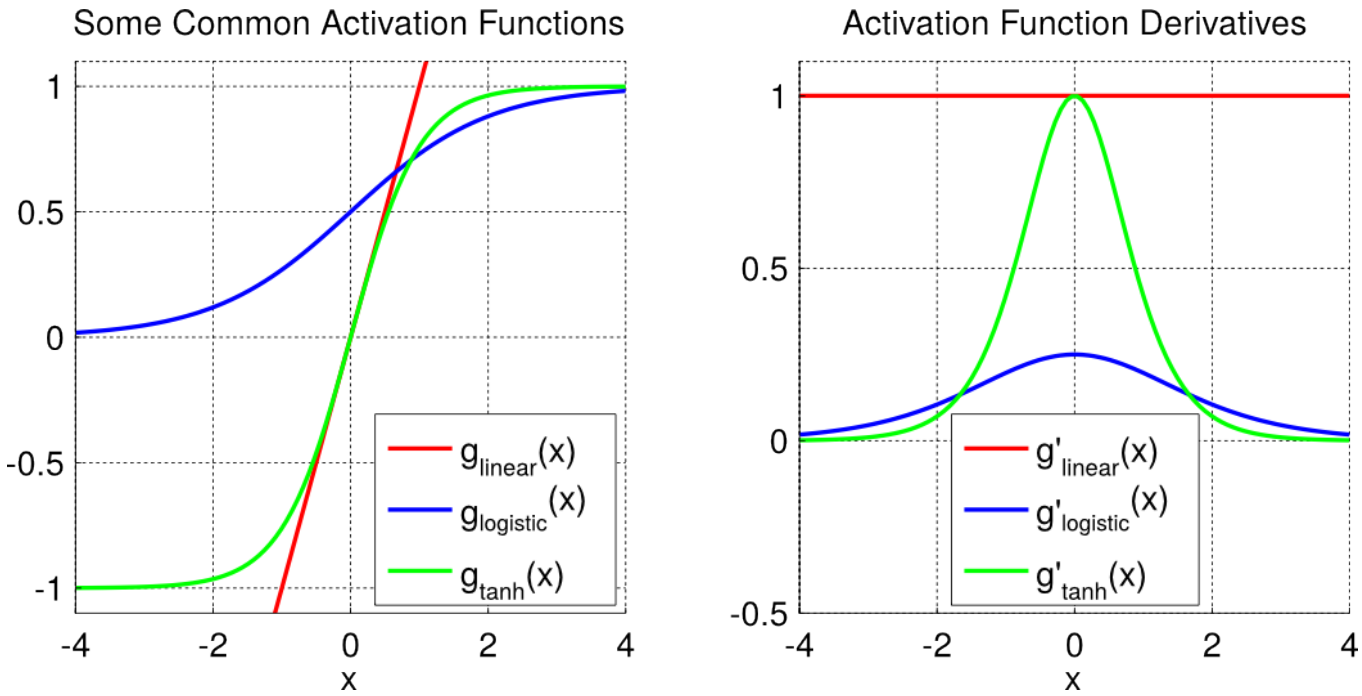


Image from `https://theclevermachine.wordpress.com`

- Unless weights large, error signal will degrade

$$\delta_h = \sigma'(\bullet) W^{(h+1)h} \delta_{h+1}$$

Vanishing gradients

- Gradients may vanish or explode
- Can affect any 'deep' network
  - e.g. fine-tuning a non-recurrent deep neural network

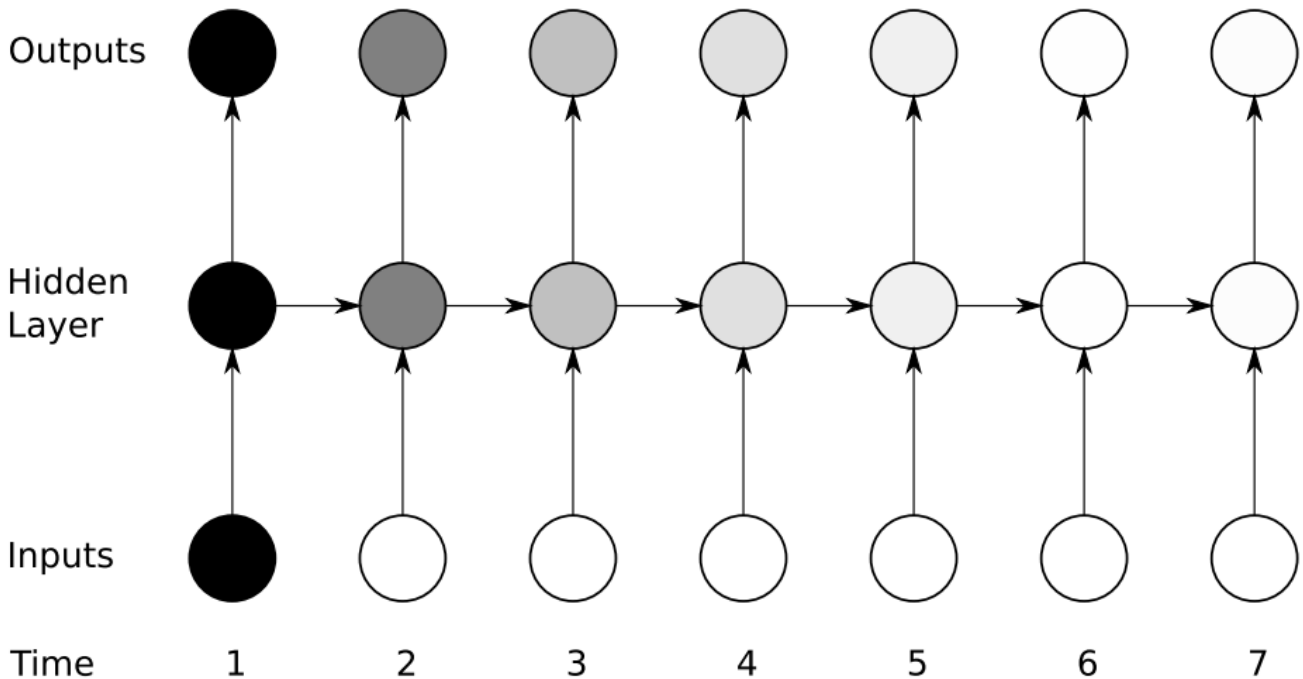
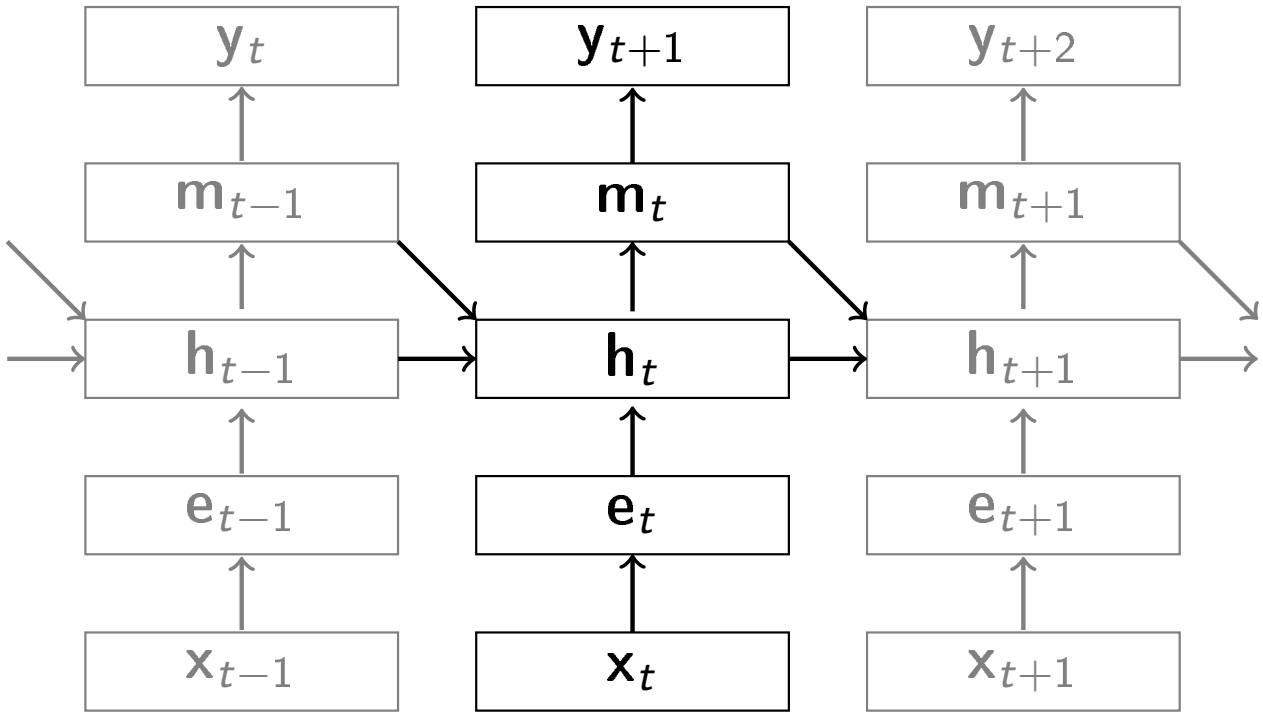


Image from Alex Graves' textbook

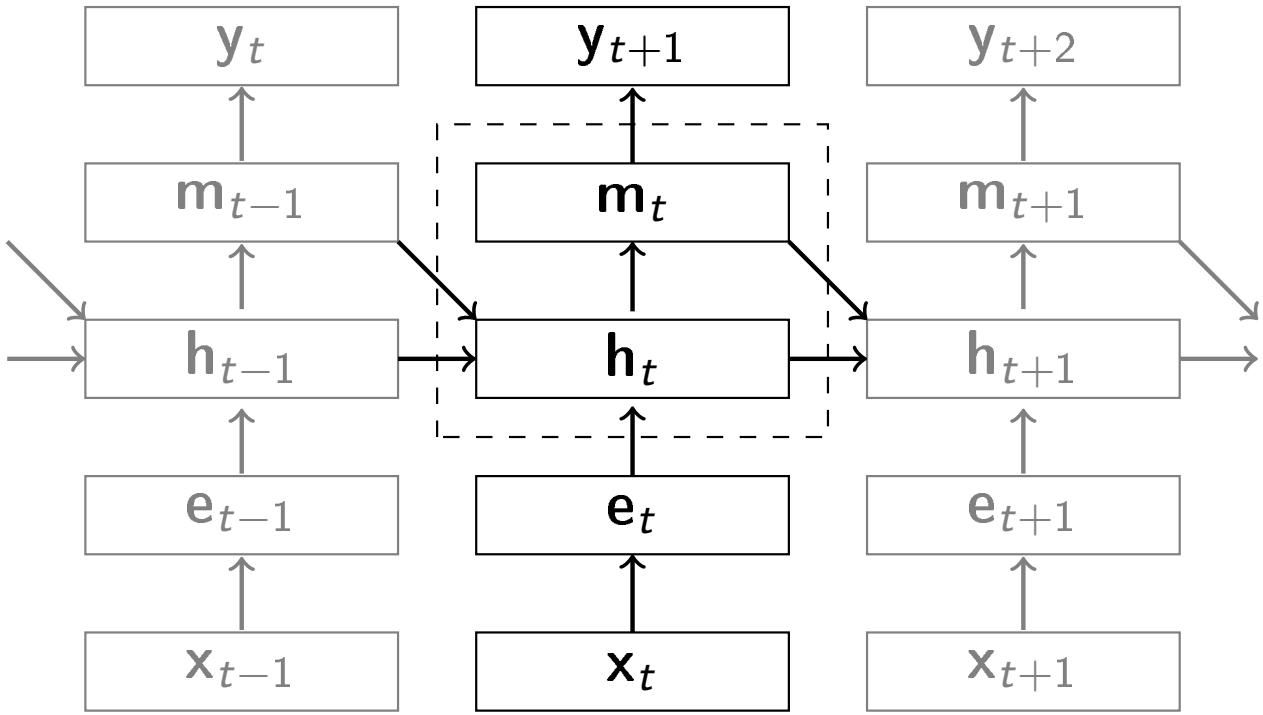
Constant Error Carousel

- Allow the network to propagate errors without modification
- No nonlinearity in recursion



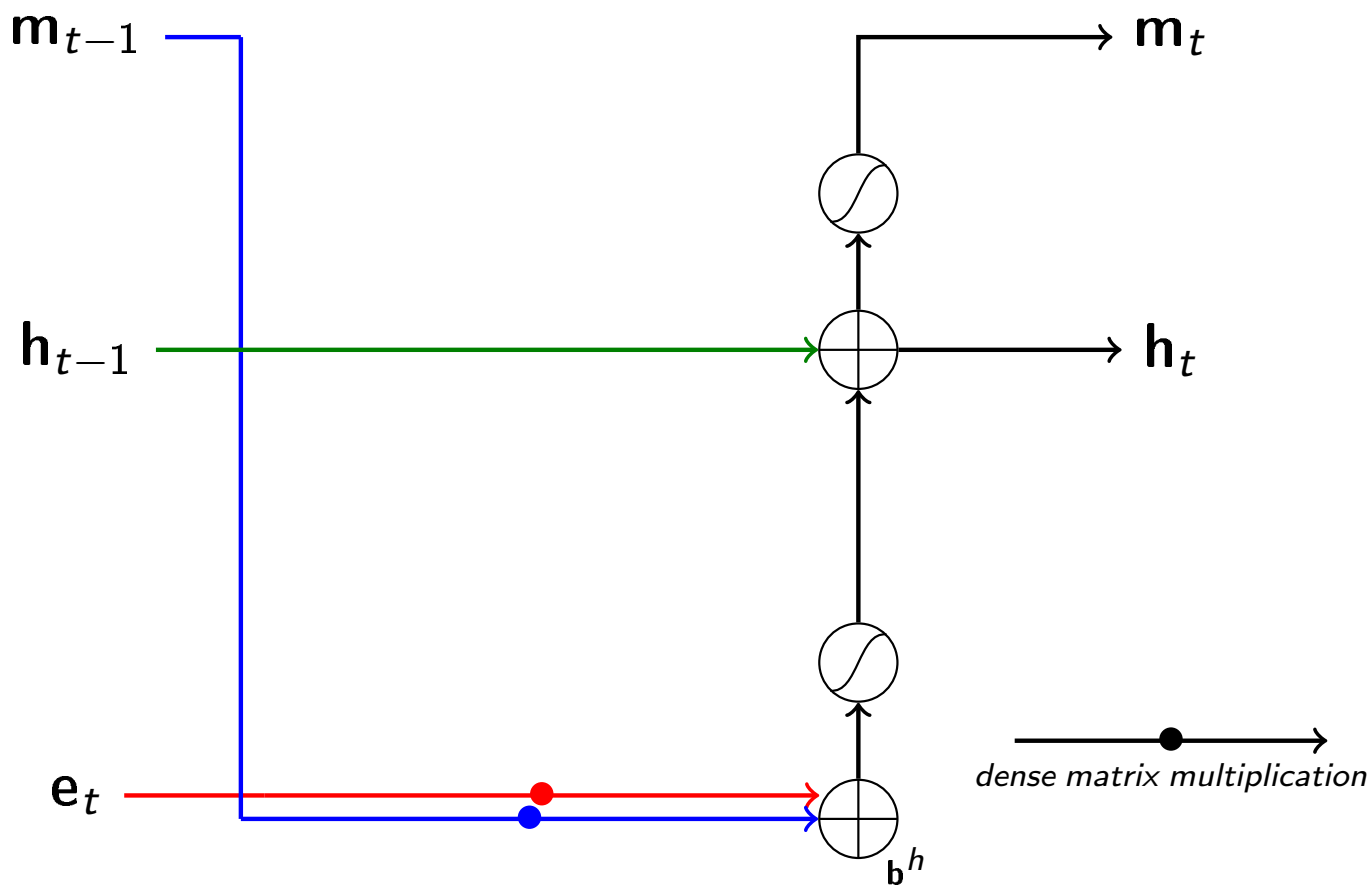
Constant Error Carousel

- Allow the network to propagate errors without modification
- No nonlinearity in recursion



Constant Error Carousel

- Allow the network to propagate errors without modification
- No nonlinearity in recursion



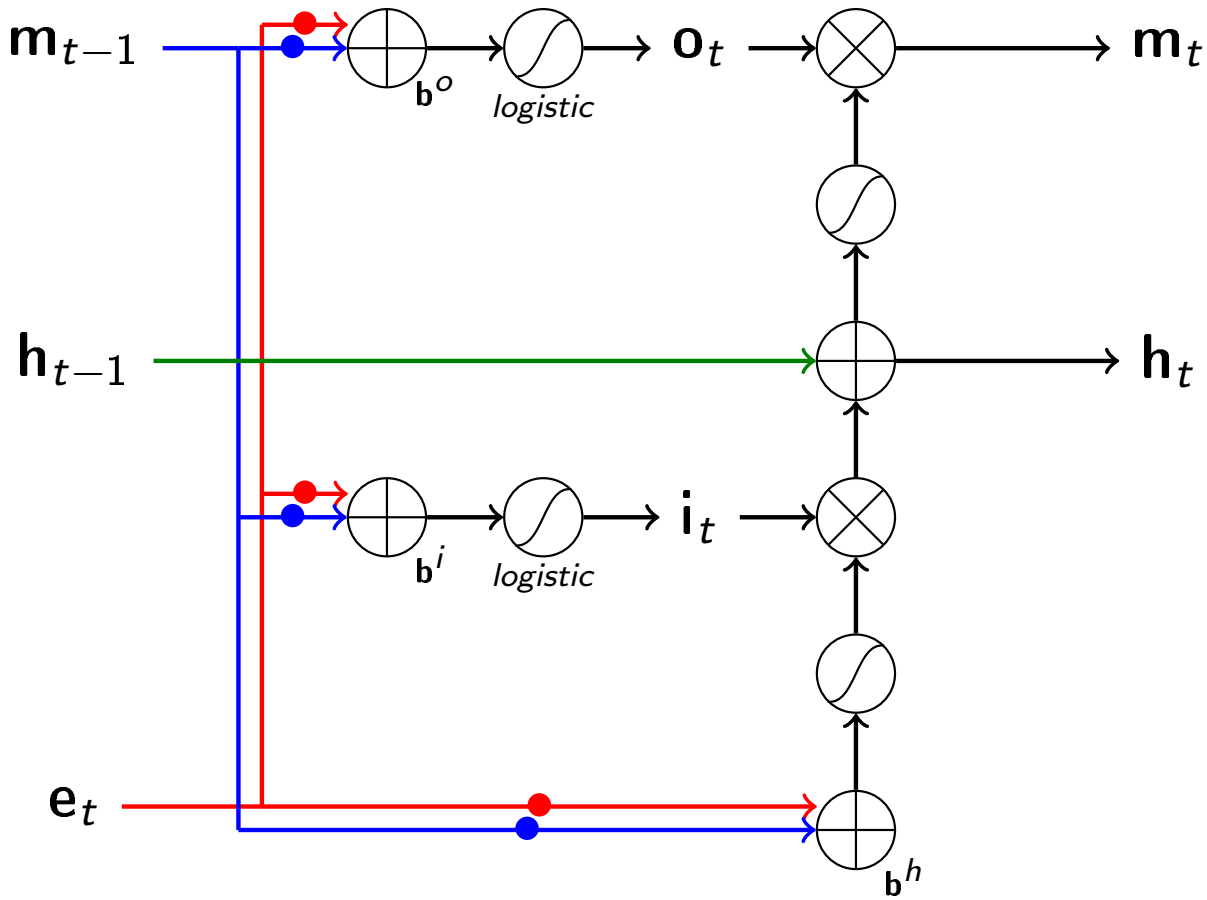
Constant Error Carousel

- Allow the network to propagate errors without modification
- No nonlinearity in recursion

$$
\begin{aligned}
\mathbf{y}_{t+1} &= \mathit{softmax}\left(W^{ym}\mathbf{m}_t\right) \\
\mathbf{m}_t &= \sigma\left(\mathbf{h}_t\right) \\
\mathbf{h}_t &= \mathbf{h}_{t-1} + \sigma\left(W^{he}\mathbf{e}_t + W^{hm}\mathbf{m}_{t-1} + \mathbf{b}^h\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

LSTM v1: input and output gates

- Attenuate input and output signals



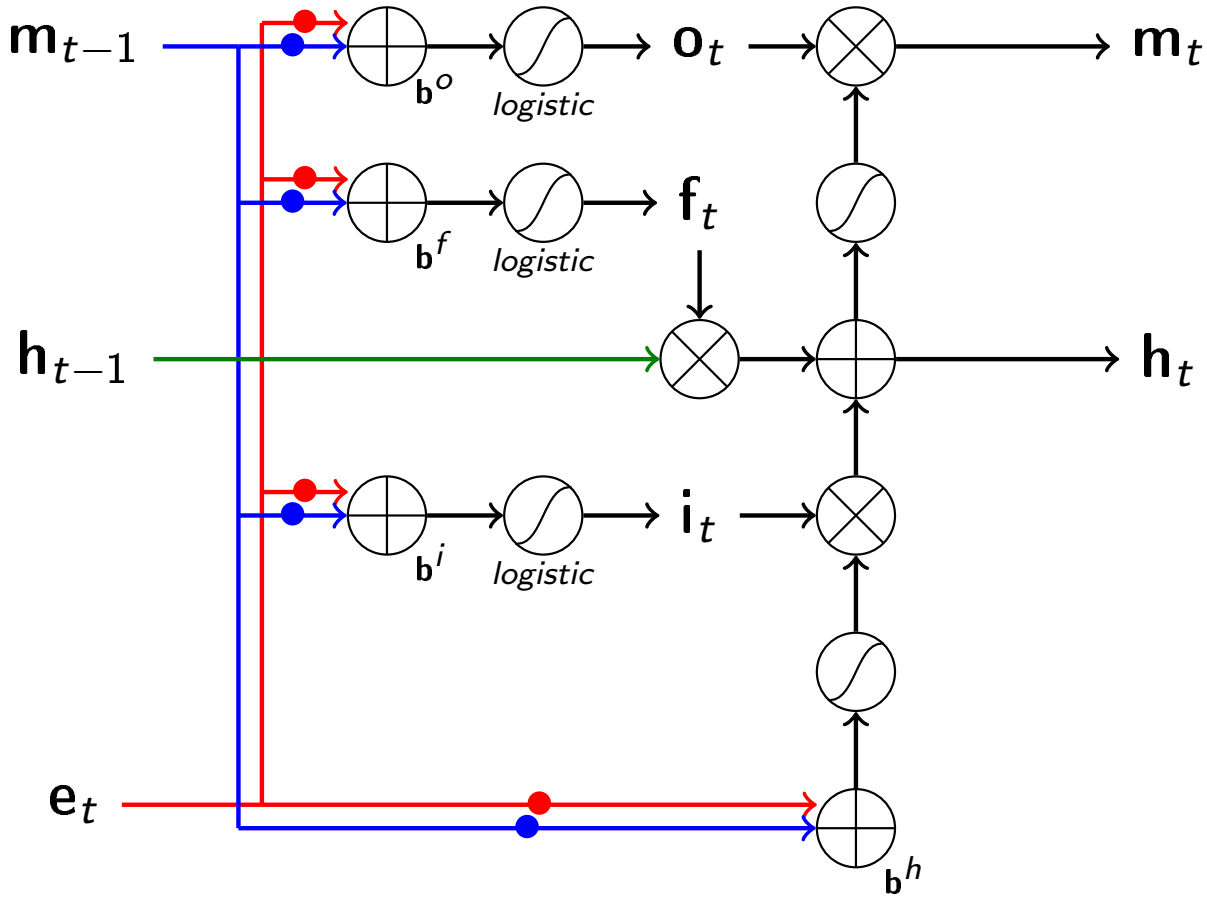
LSTM v1: input and output gates

- Attenuate input and output signals

$$
\begin{aligned}
\mathbf{y}_{t+1} &= \mathit{softmax}\left(W^{ym}\mathbf{m}_t\right) \\
\mathbf{m}_t &= \mathbf{o}_t \odot \sigma\left(\mathbf{h}_t\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{om}\mathbf{m}_{t-1} + \mathbf{b}^o\right) \\
\mathbf{h}_t &= \mathbf{h}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{he}\mathbf{e}_t + W^{hm}\mathbf{m}_{t-1} + \mathbf{b}^h\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{im}\mathbf{m}_{t-1} + \mathbf{b}^i\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

LSTM v2: forget (remember) gate

- Model controls when memory, $\mathbf{h}_t$, is reduced
- Forget gate should be called remember gate



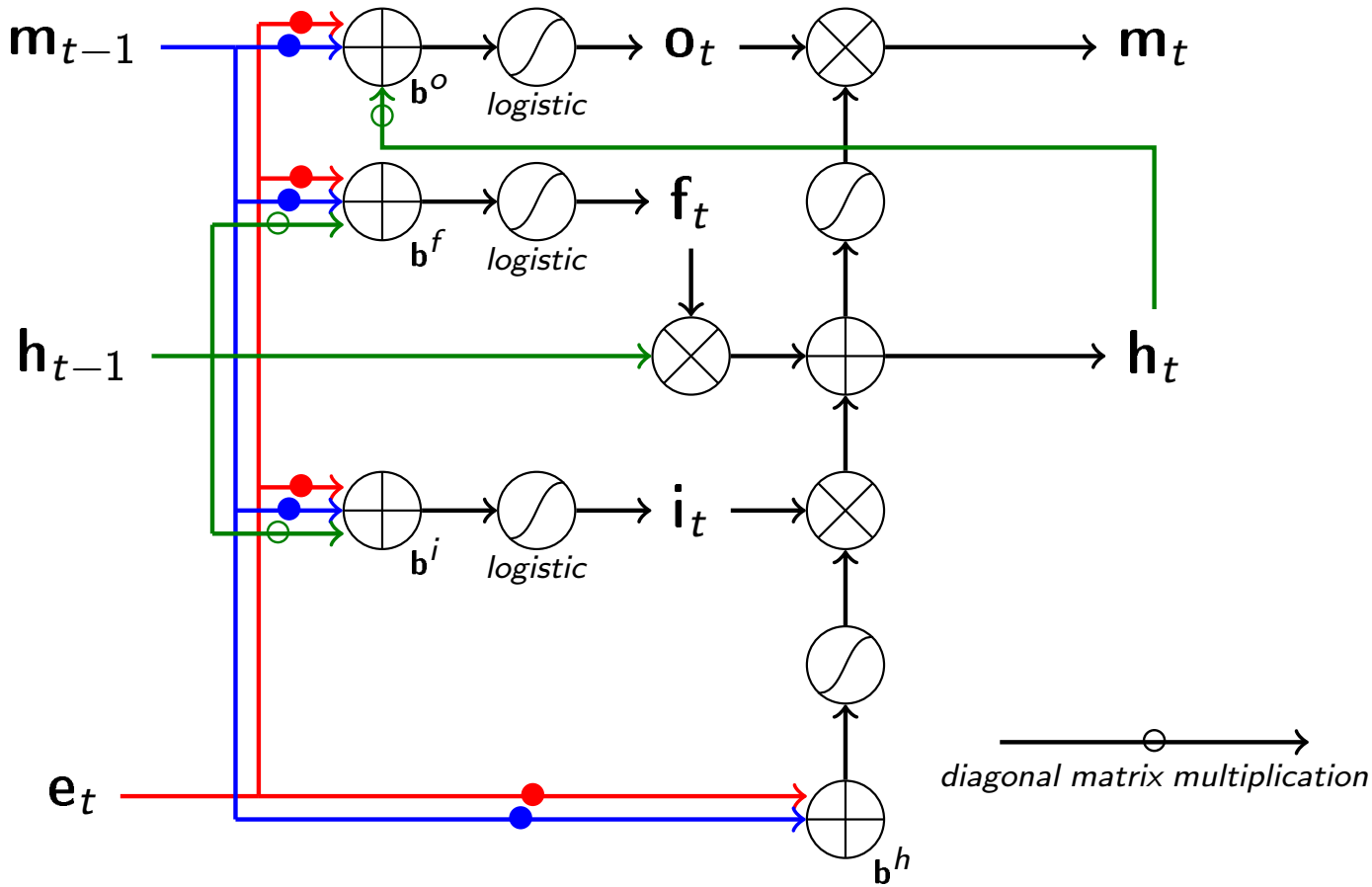
LSTM v2: forget (remember) gate

- Model controls when memory, $\mathbf{h}_t$, is reduced
- Forget gate should be called remember gate

$$
\begin{aligned}
\mathbf{y}_{t+1} &= \mathit{softmax}\left(W^{ym}\mathbf{m}_t\right) \\
\mathbf{m}_t &= \mathbf{o}_t \odot \sigma\left(\mathbf{h}_t\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{om}\mathbf{m}_{t-1} + \mathbf{b}^o\right) \\
\mathbf{h}_t &= \mathbf{f}_i \odot \mathbf{h}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{he}\mathbf{e}_t + W^{hm}\mathbf{m}_{t-1} + \mathbf{b}^h\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{im}\mathbf{m}_{t-1} + \mathbf{b}^i\right) \\
\mathbf{f}_i &= \mathit{logistic}\left(W^{fe}\mathbf{e}_t + W^{fm}\mathbf{m}_{t-1} + \mathbf{b}^f\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

LSTM v3: peepholes

- Allow the gates to additionally see the internal memory state
- Diagonal matrices only (all others dense)



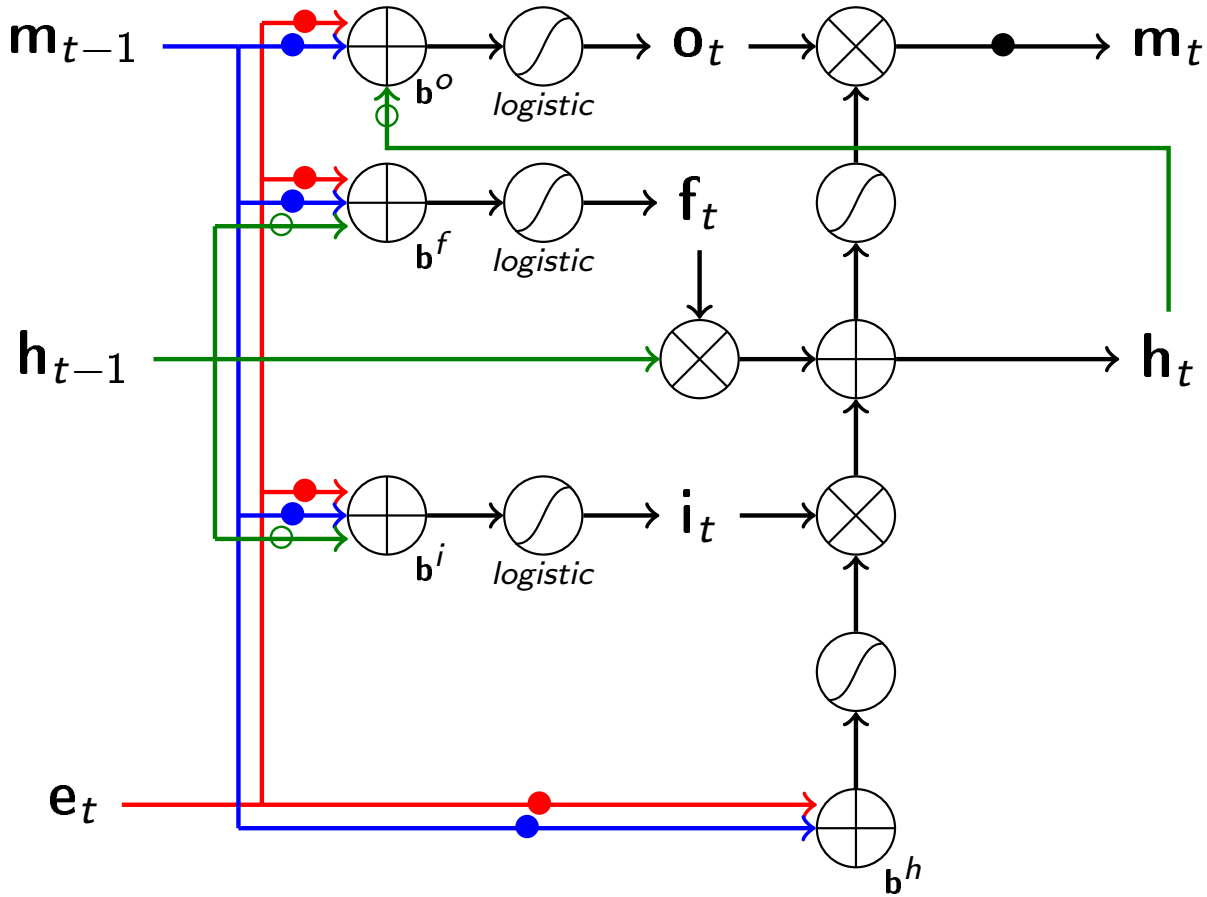
LSTM v3: peepholes

- Allow the gates to additionally see the internal memory state
- Diagonal matrices only (all others dense)

$$
\begin{aligned}
\mathbf{y}_{t+1} &= \mathit{softmax}\left(W^{ym}\mathbf{m}_t\right) \\
\mathbf{m}_t &= \mathbf{o}_t \odot \sigma\left(\mathbf{h}_t\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{om}\mathbf{m}_{t-1} + W^{oh}\mathbf{h}_t + \mathbf{b}^o\right) \\
\mathbf{h}_t &= \mathbf{f}_i \odot \mathbf{h}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{he}\mathbf{e}_t + W^{hm}\mathbf{m}_{t-1} + \mathbf{b}^h\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{im}\mathbf{m}_{t-1} + W^{ih}\mathbf{h}_{t-1} + \mathbf{b}^i\right) \\
\mathbf{f}_i &= \mathit{logistic}\left(W^{fe}\mathbf{e}_t + W^{fm}\mathbf{m}_{t-1} + W^{fh}\mathbf{h}_{t-1} + \mathbf{b}^f\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

LSTM v4: output projection layer

- Reduces dimensionality of recursive messages
- Can speed up training without affecting results quality



LSTM v4: output projection layer

- Reduces dimensionality of recursive messages
- Can speed up training without affecting results quality

$$
\begin{aligned}
\mathbf{y}_{t+1} &= \mathit{softmax}\left(W^{ym}\mathbf{m}_t\right) \\
\mathbf{m}_t &= W^{mm}\left(\mathbf{o}_t \odot \sigma\left(\mathbf{h}_t\right)\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{om}\mathbf{m}_{t-1} + W^{oh}\mathbf{h}_t + \mathbf{b}^o\right) \\
\mathbf{h}_t &= \mathbf{f}_i \odot \mathbf{h}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{he}\mathbf{e}_t + W^{hm}\mathbf{m}_{t-1} + \mathbf{b}^h\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{im}\mathbf{m}_{t-1} + W^{ih}\mathbf{h}_{t-1} + \mathbf{b}^i\right) \\
\mathbf{f}_i &= \mathit{logistic}\left(W^{fe}\mathbf{e}_t + W^{fm}\mathbf{m}_{t-1} + W^{fh}\mathbf{h}_{t-1} + \mathbf{b}^f\right) \\
\mathbf{e}_t &= W^{ex}\mathbf{x}_t
\end{aligned}
$$

Gradients no longer vanish

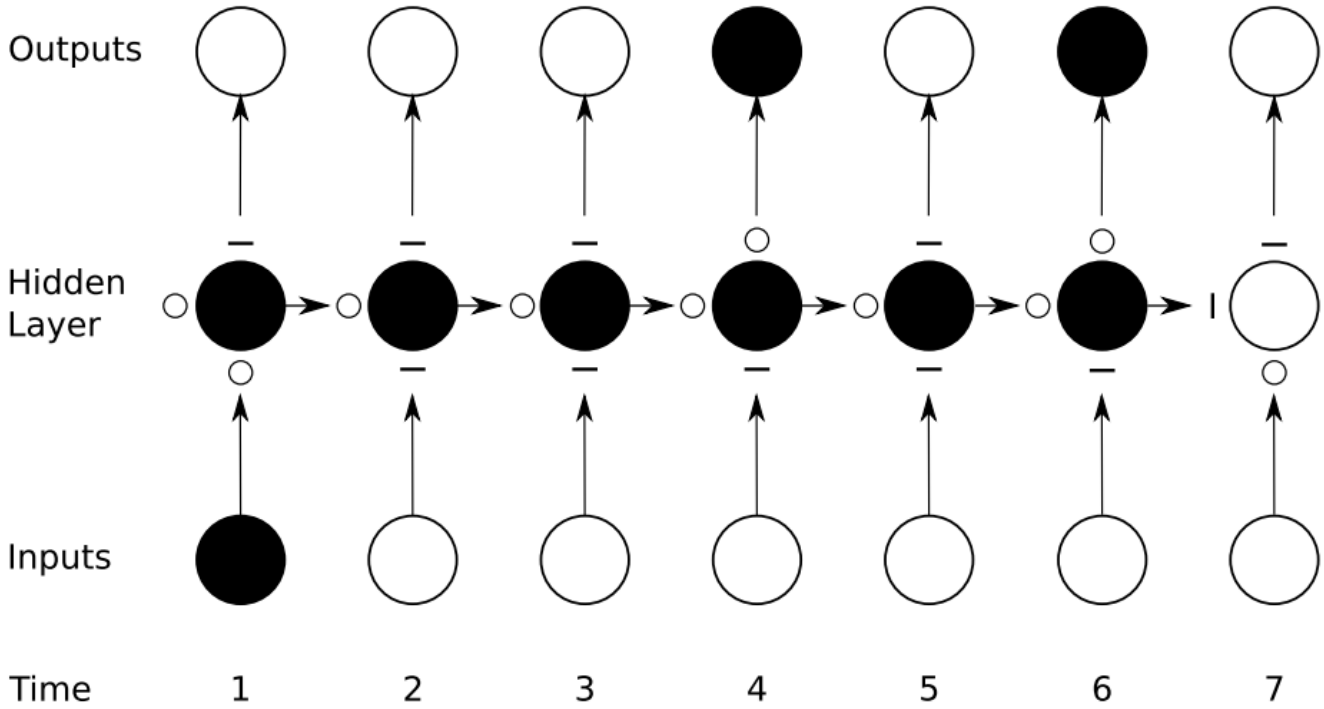


Image from Alex Graves' textbook

LSTM implementations

- RNNLIB (Alex Graves) `http://sourceforge.net/p/rnnl/`
- PyLearn2 (experimental code, in sandbox/rnn/models/rnn.py)
- Theano, e.g.

```
def lstm_step(x_t, m_tm1, h_tm1, w_xe, ..., b_o):
 e_t = dot(x_t, w_xe)
 i_t = sigmoid(dot(e_t, w_ei) + dot(m_tm1, w_mi) + c_tm1 * w_ci + b_i)
 f_t = sigmoid(dot(e_t, w_ef) + dot(m_tm1, w_mf) + c_tm1 * w_cf + b_f)
 h_t = f_t * h_tm1 + i_t * tanh(dot(e_t, w_eh) + dot(m_tm1, w_mh) + b_h)
 o_t = sigmoid(dot(e_t, w_eo) + dot(m_tm1, w_mo) + c_t * w_co + b_o)
 m_t = dot(o_t * tanh(h_t), w_mm)
 y_t = softmax(dot(m_t, w_my))
 return m_t, c_t, y_t
```

Further thoughts

- Sequences vs. hierarchies vs. plain 'deep'
- Other solutions to vanishing gradients
  - Clockwork RNN
  - Different training algorithms (e.g. Hessian Free optimization)
  - Rectified linear units (ReLU)? $\sigma(x) = max(0, x)$; constant gradient when active