# Long Short-Term Memory (LSTM)

## A brief introduction

Daniel Renshaw

24th November 2014
(updated 30th November 2015)

# Introduction

- What is Long Short-Term Memory (LSTM)?
  - A type of artificial neural network (ANN) layer
  - More specifically: a type of *recurrent* ANN layer
- Not a whole network architecture, just a layer
- Designed to solve the vanishing/exploding gradient problem
- *Why does it look so complex?*

# Context and notation

- Task: ANN language modelling (example for this presentation)
- Word vocabulary, size $V$
- $\mathbf{x}_t \in \mathbb{R}^V$: true word in position $t$ (one-hot)
- $\mathbf{y}_t \in \Delta^{V-1}$: predicted word in position $t$ (distribution)
- Assume inputs are sentences zero padded to length $L$

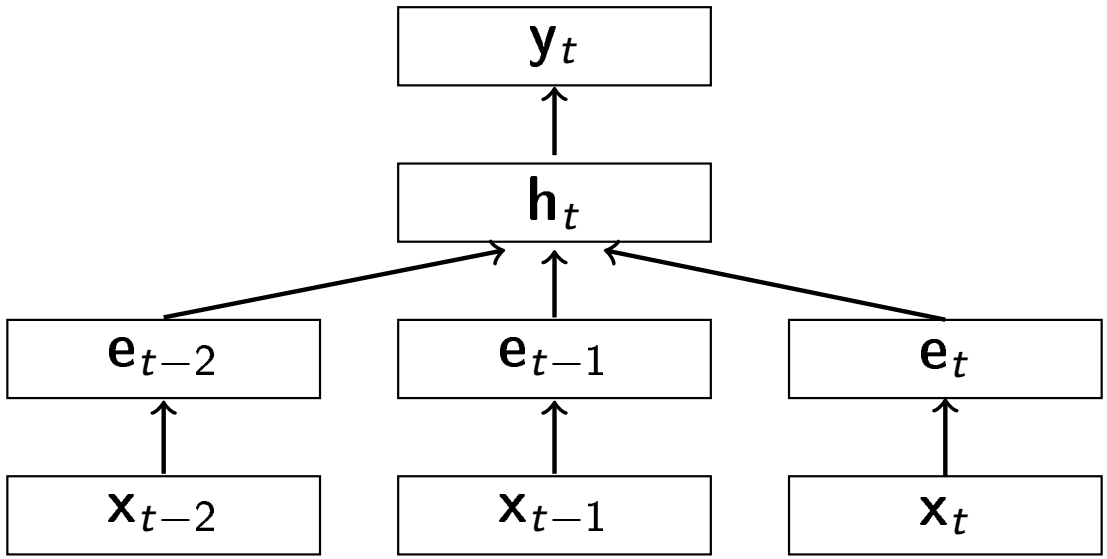
# Context and notation

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

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

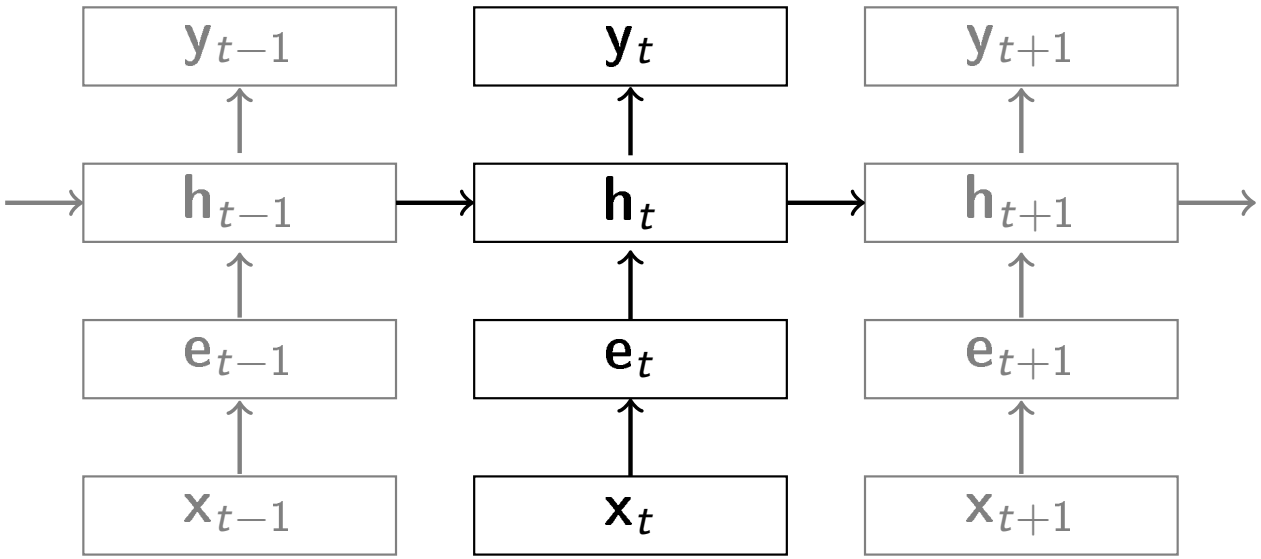
- $\sigma(\bullet)$ is a sigmoid-shaped function (e.g. *logistic* or *tanh*)
- $\mathbf{b}^\bullet$ is a bias vector, $W^\bullet$ is a weight matrix

# Multi-Layer Perceptron (MLP)




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

# Recurrent Neural Network (RNN)




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

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

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

# Vanishing gradients

- Error gradients pass through nonlinearity every step

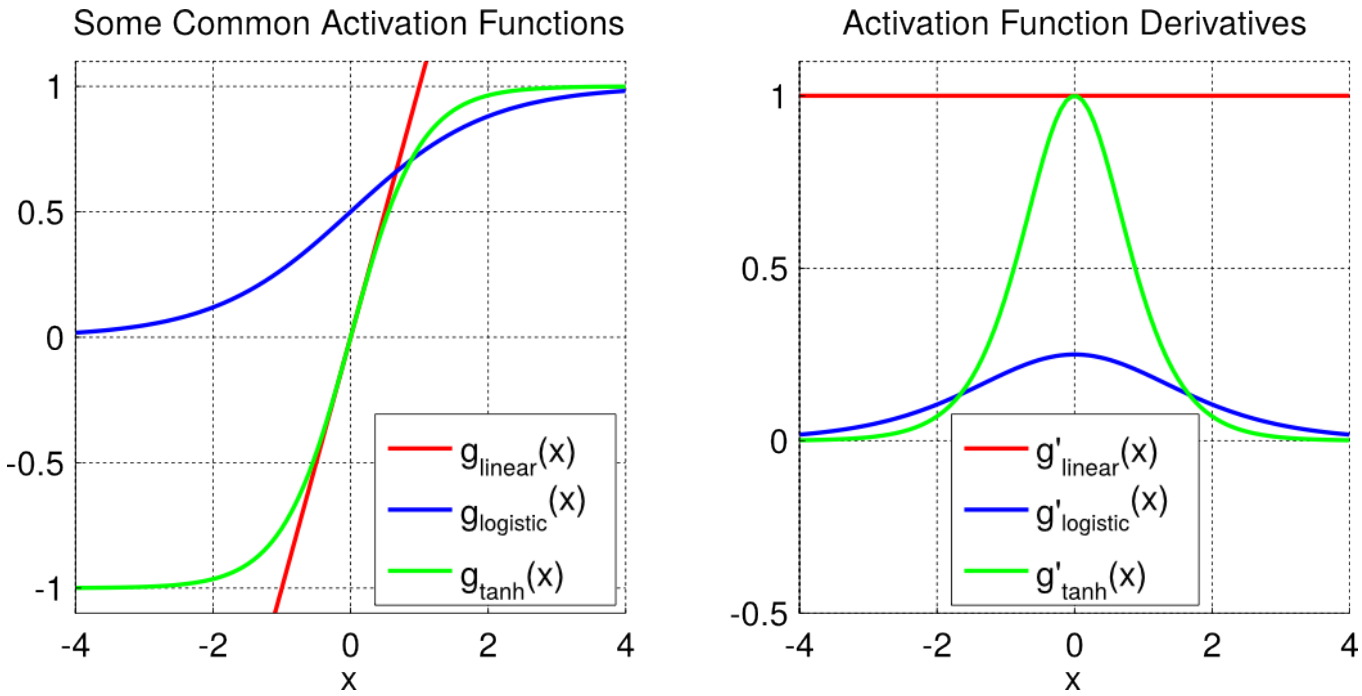



Image from `https://theclevermachine.wordpress.com`

- 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

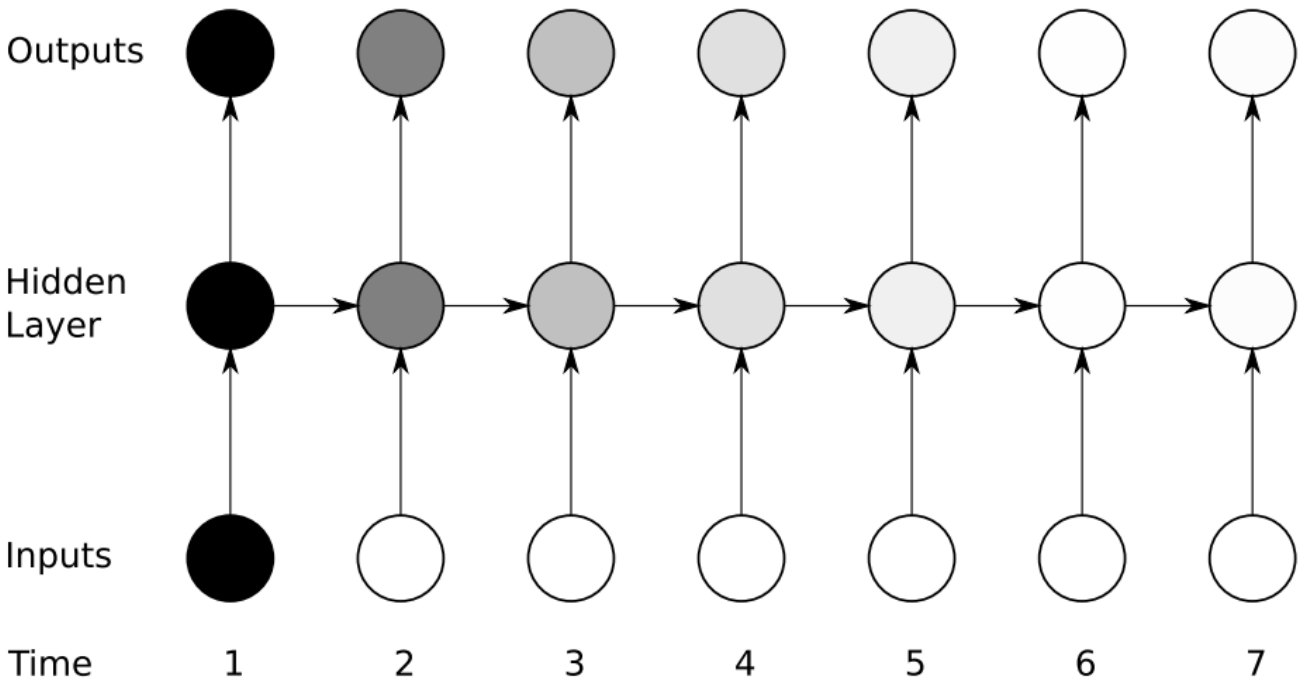
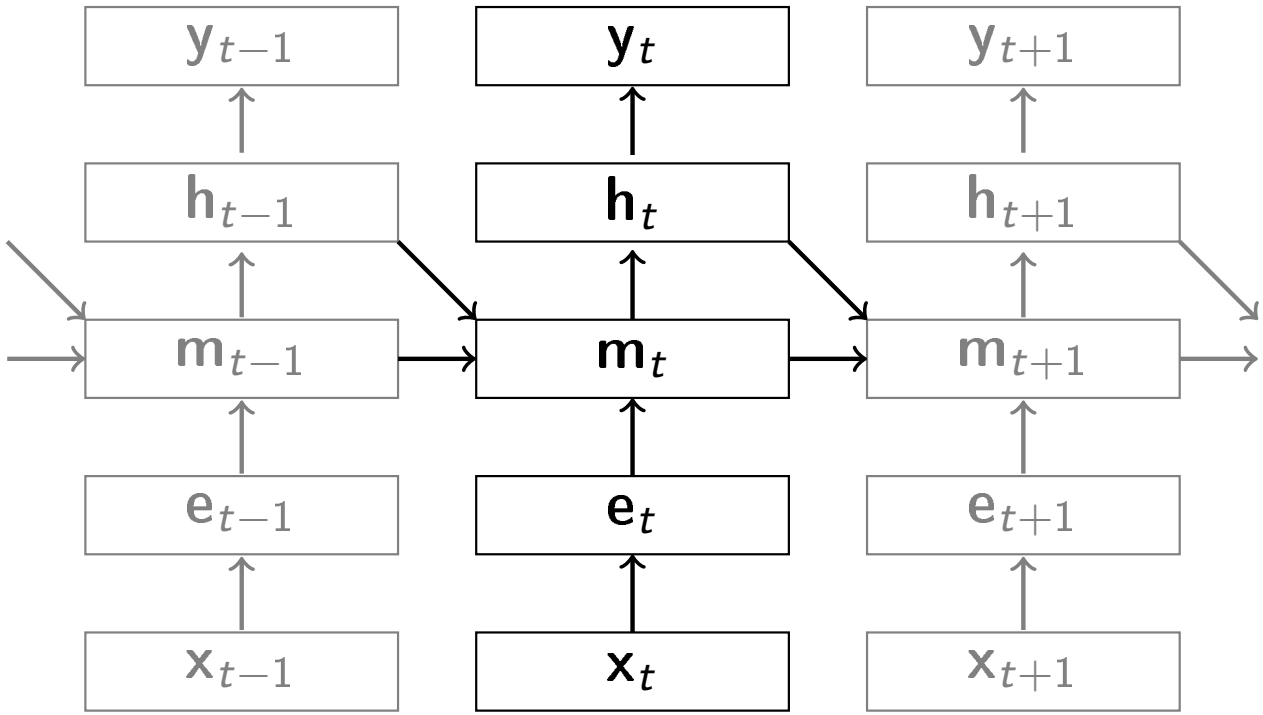



Image from Alex Graves' textbook

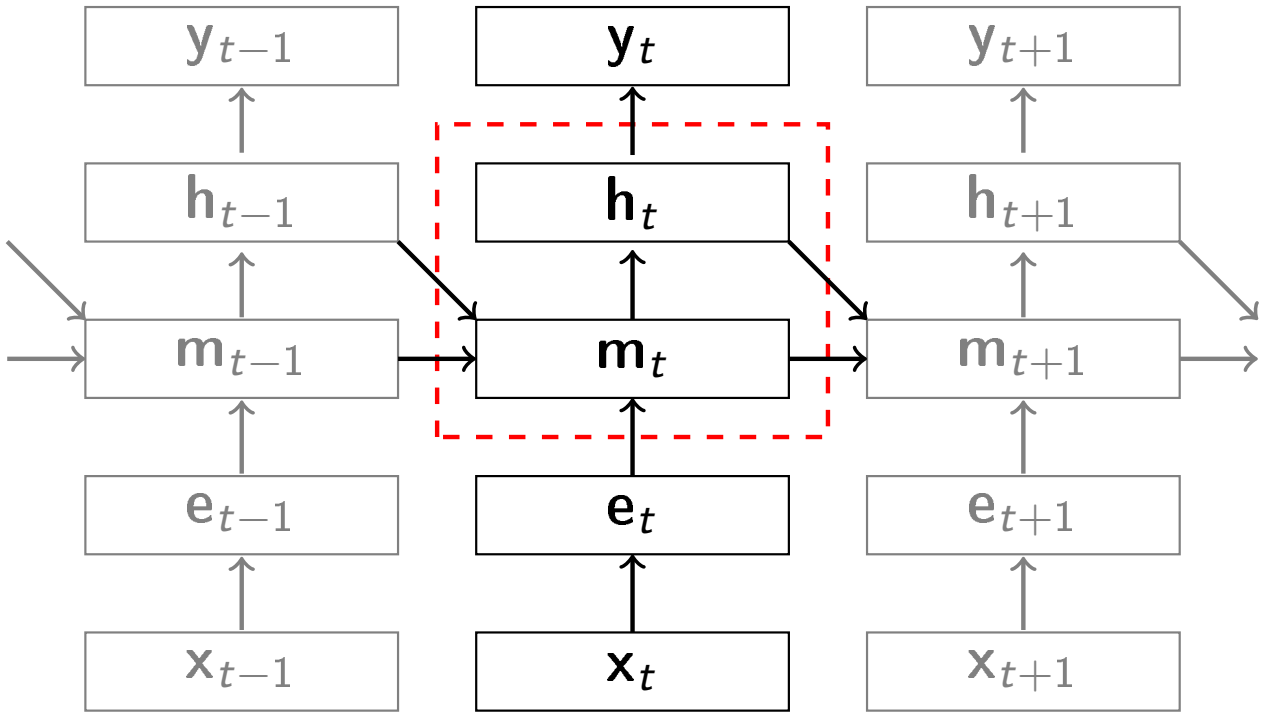
# Constant Error Carousel

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



# Constant Error Carousel

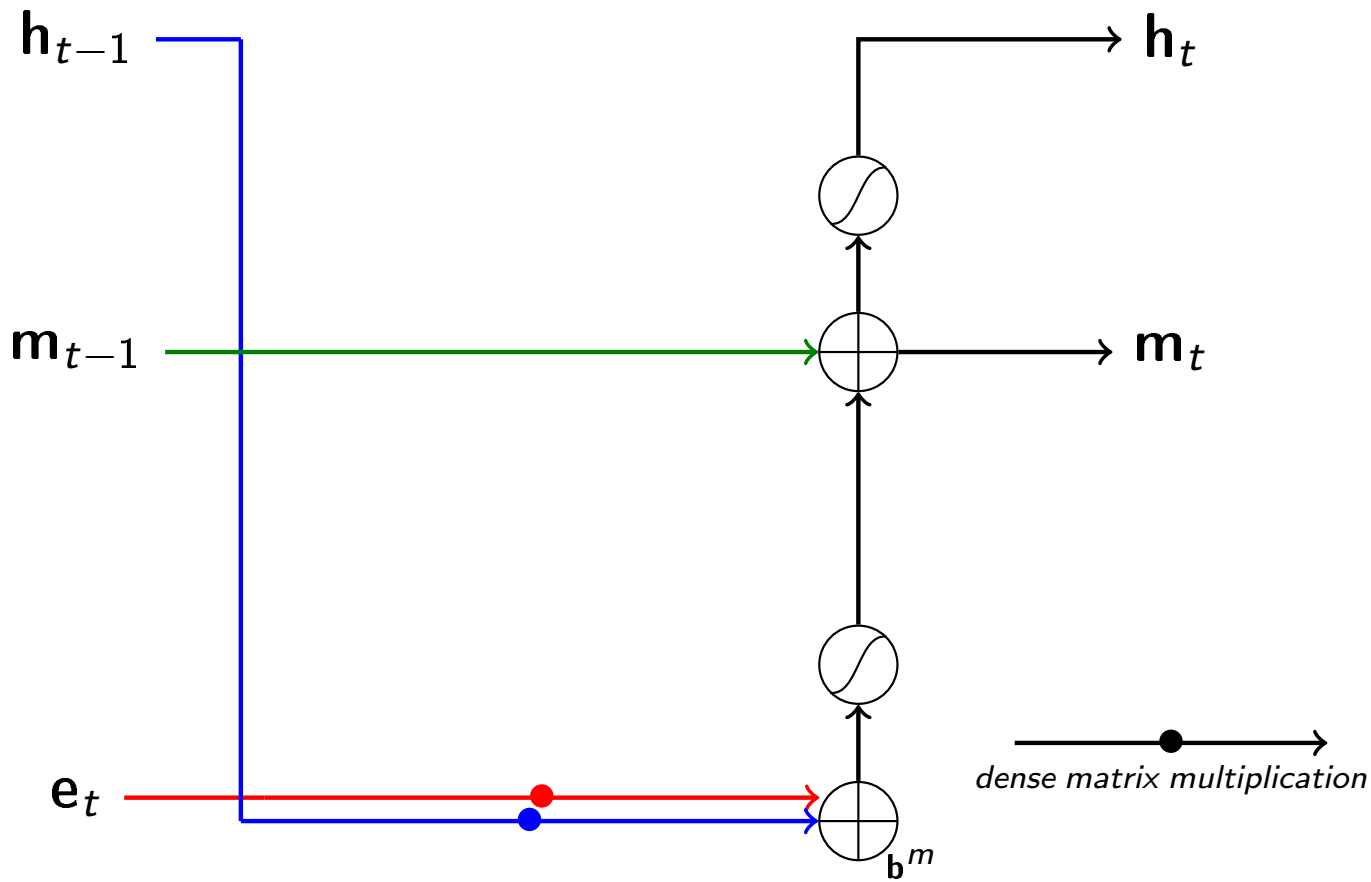
- Allow the network to propagate errors without modification
- No nonlinearity in recursion




- Focus on region in red dashed box from now on

# Constant Error Carousel

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



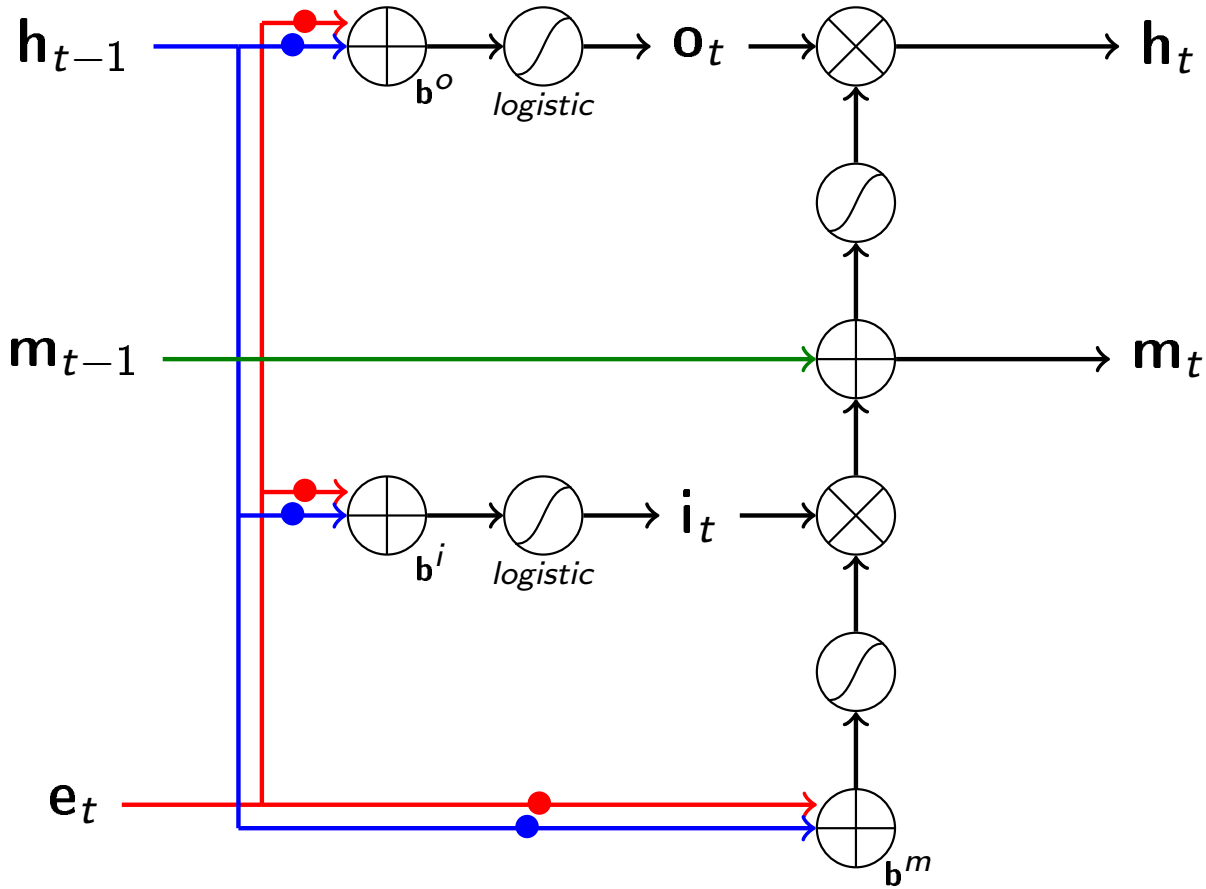
# Constant Error Carousel

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

$$
\begin{aligned}
\mathbf{h}_t &= \sigma\left(\mathbf{m}_t\right) \\
\mathbf{m}_t &= \mathbf{m}_{t-1} + \sigma\left(W^{me}\mathbf{e}_t + W^{mh}\mathbf{h}_{t-1} + \mathbf{b}^m\right)
\end{aligned}
$$

# LSTM v1: input and output gates

- Attenuate input and output signals
- Gates applied elementwise



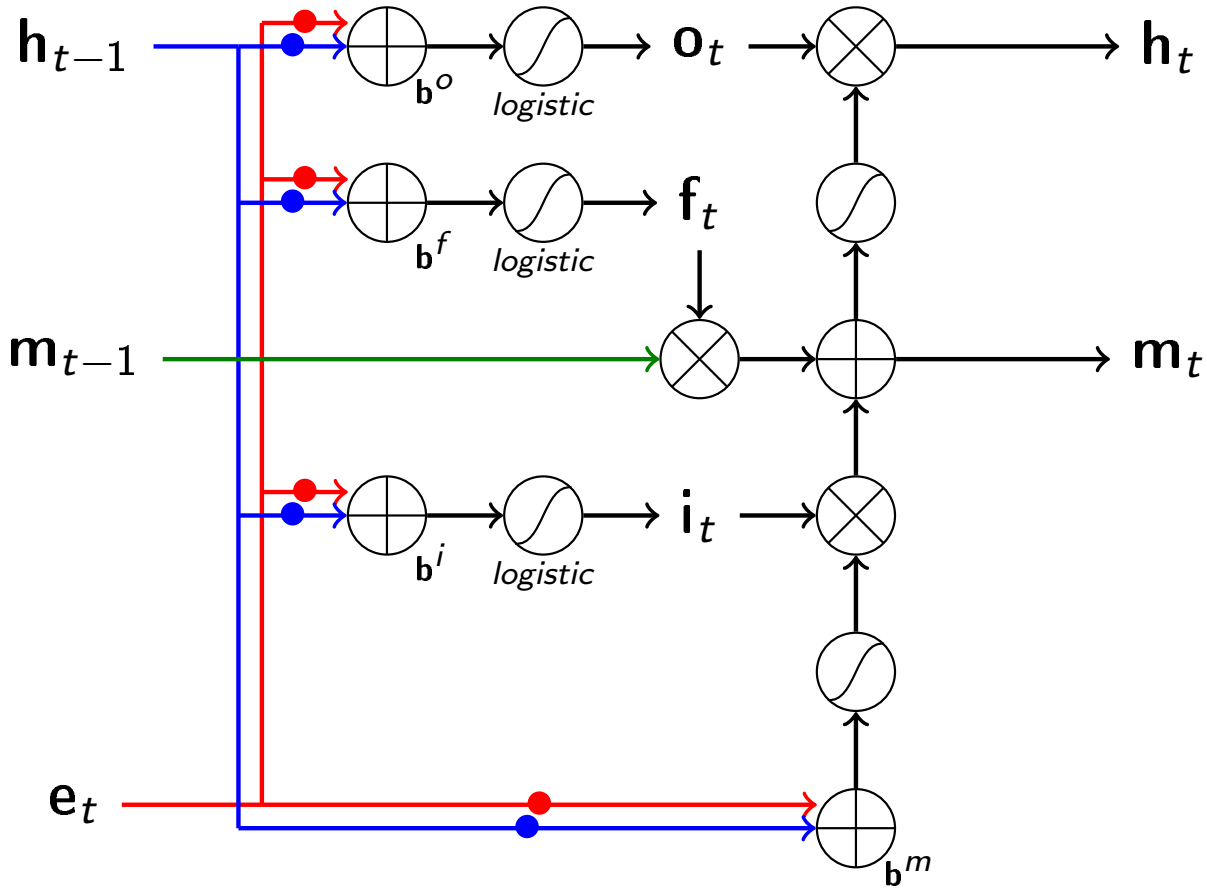
# LSTM v1: input and output gates

- Attenuate input and output signals
- Gates applied elementwise

$$
\begin{aligned}
\mathbf{h}_t &= \mathbf{o}_t \odot \sigma\left(\mathbf{m}_t\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{oh}\mathbf{h}_{t-1} + \mathbf{b}^o\right) \\
\mathbf{m}_t &= \mathbf{m}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{me}\mathbf{e}_t + W^{mh}\mathbf{h}_{t-1} + \mathbf{b}^m\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{ih}\mathbf{h}_{t-1} + \mathbf{b}^i\right)
\end{aligned}
$$

# LSTM v2: forget (remember) gate

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



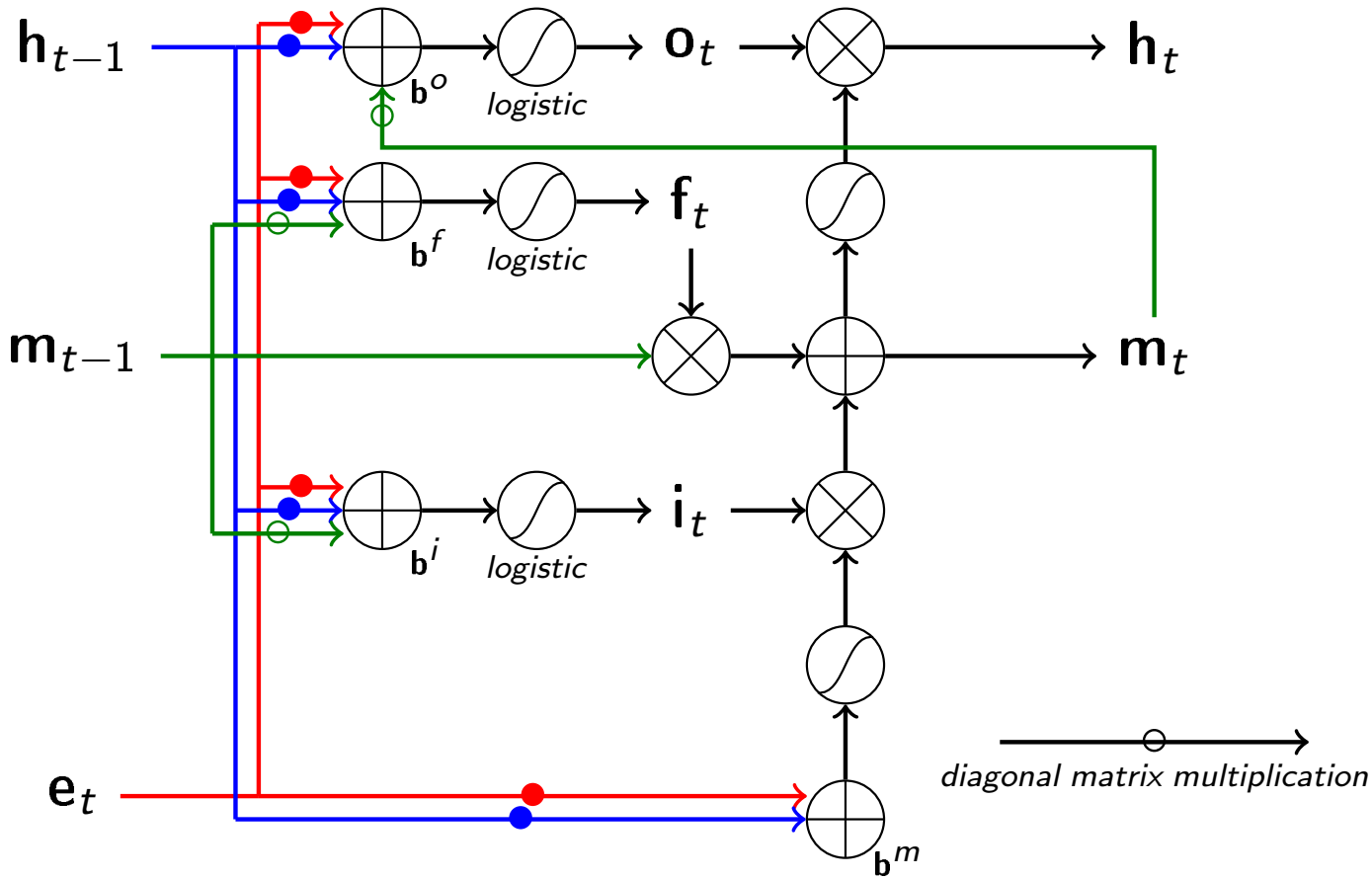
# LSTM v2: forget (remember) gate

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

$$
\begin{aligned}
\mathbf{h}_t &= \mathbf{o}_t \odot \sigma\left(\mathbf{m}_t\right) \\
\mathbf{o}_t &= \mathit{logistic}\left(W^{oe}\mathbf{e}_t + W^{oh}\mathbf{h}_{t-1} + \mathbf{b}^o\right) \\
\mathbf{m}_t &= \mathbf{f}_i \odot \mathbf{m}_{t-1} + \mathbf{i}_t \odot \sigma\left(W^{me}\mathbf{e}_t + W^{mh}\mathbf{h}_{t-1} + \mathbf{b}^m\right) \\
\mathbf{i}_t &= \mathit{logistic}\left(W^{ie}\mathbf{e}_t + W^{ih}\mathbf{h}_{t-1} + \mathbf{b}^i\right) \\
\mathbf{f}_i &= \mathit{logistic}\left(W^{fe}\mathbf{e}_t + W^{fh}\mathbf{h}_{t-1} + \mathbf{b}^f\right)
\end{aligned}
$$

# LSTM v3: peepholes

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



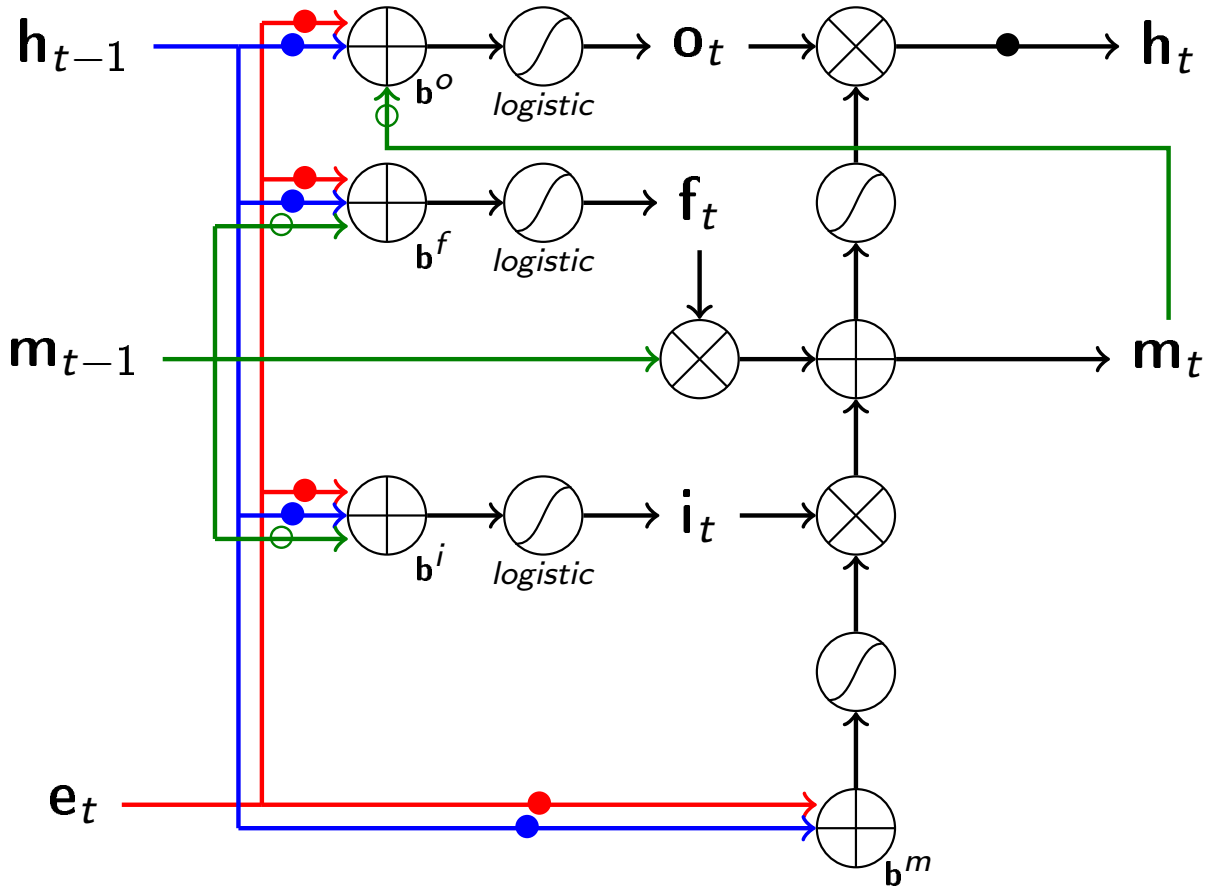
# LSTM v3: peepholes

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

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

# 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

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

# Gradients no longer vanish

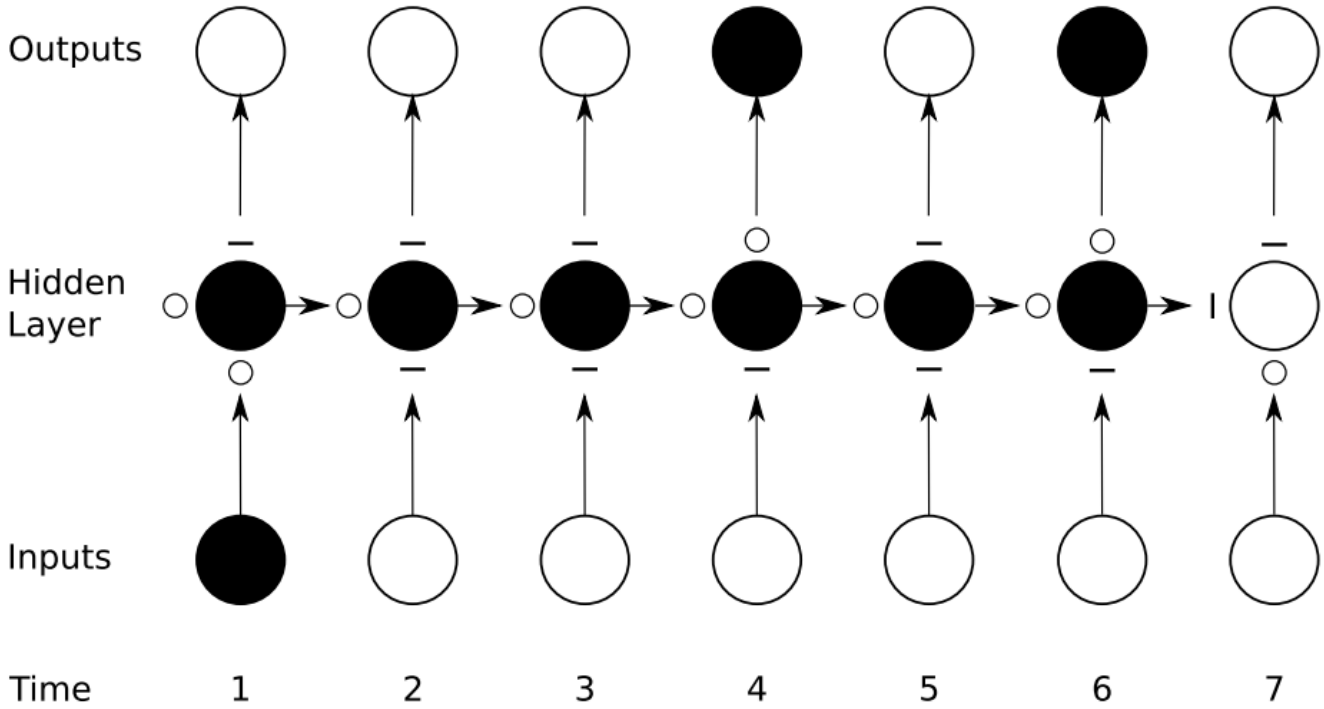



Image from Alex Graves' textbook

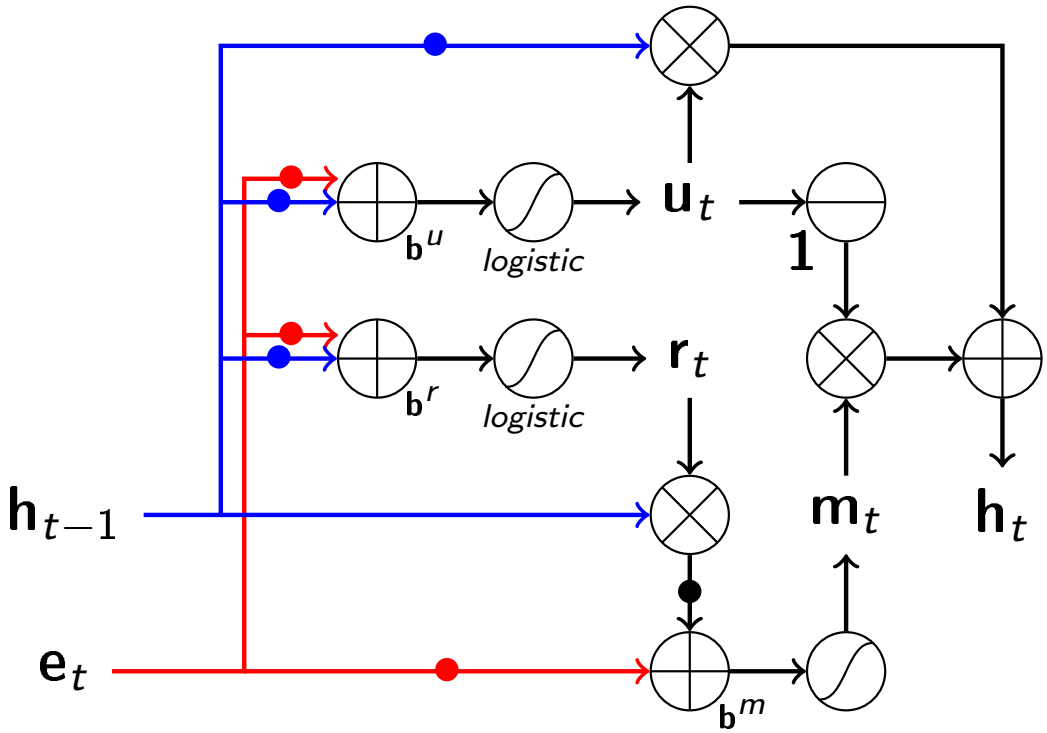
# LSTM implementations

- RNNLIB (Alex Graves) `http://sourceforge.net/p/rnnl/`
- Built-in: Lasagne, Blocks, Keras
- Built-out: Caffe (to be built-in later), Torch, Theano, e.g.

```
def lstm_step(x_t, m_tm1, h_tm1, w_xe, ..., b_y):
 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) + b_y)
 return m_t, h_t, y_t
```

# Gated Recurrent Unit

- Introduced with RNN Encoder-Decoder (Cho et al. 2014)
- Complexity midway between RNN and LSTM



# Gated Recurrent Unit

- Introduced with RNN Encoder-Decoder (Cho et al. 2014)
- Complexity midway between RNN and LSTM

$$
\begin{aligned}
\mathbf{h}_t &= \mathbf{u}_t \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{u}_t)\, \mathbf{m}_t \\
\mathbf{u}_t &= \mathit{logistic}\left(W^{ue}\mathbf{e}_t + W^{uh}\mathbf{h}_{t-1} + \mathbf{b}^u\right) \\
\mathbf{m}_t &= \sigma\left(W^{me}\mathbf{e}_t + W^{mh}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{b}^m\right) \\
\mathbf{r}_t &= \mathit{logistic}\left(W^{re}\mathbf{e}_t + W^{rh}\mathbf{h}_{t-1} + \mathbf{b}^r\right)
\end{aligned}
$$