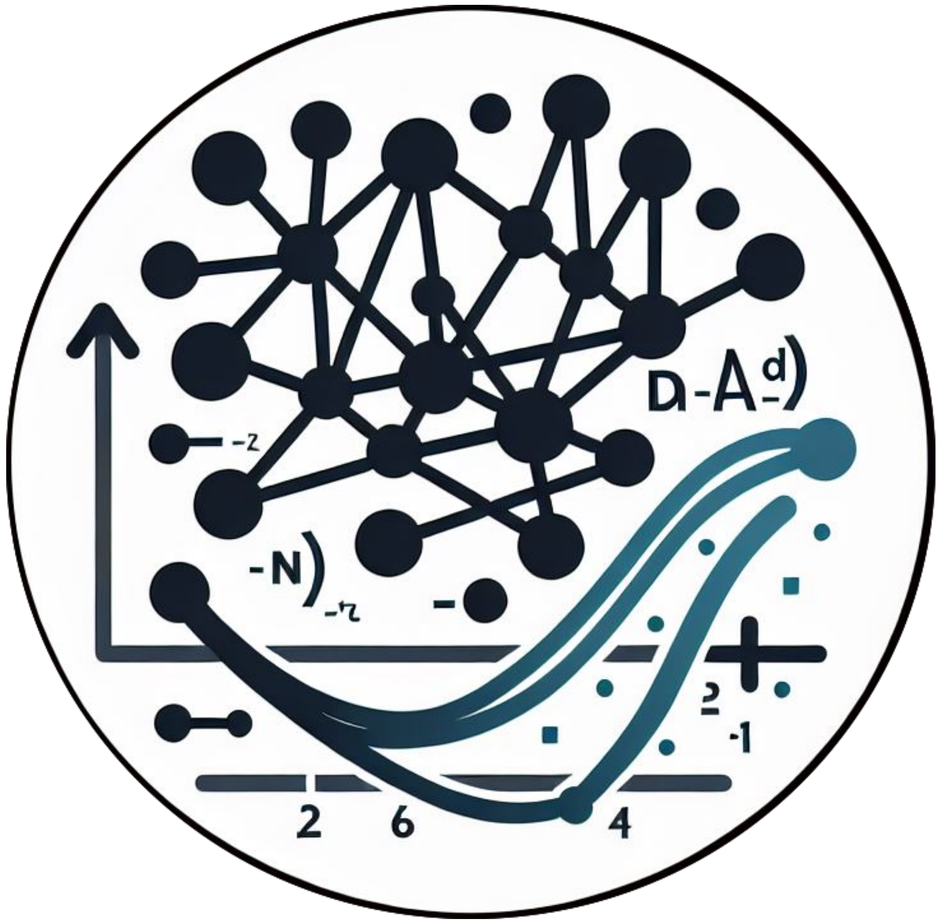
Recurrent Neural Nets



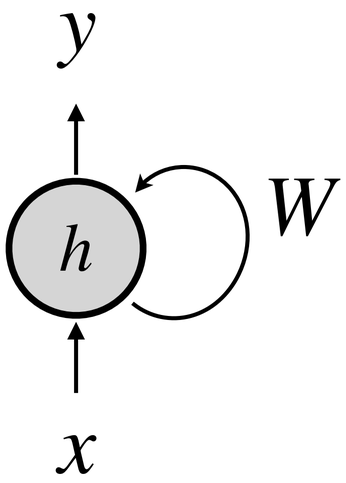
Deep Learning for Engineers
Andrew Ning
aning@byu.edu

When to use a RNN (and its variants) over a Neural ODE

RNN might be preferable if:

- uniform sequence, but no underlying physics
- sequence is long and need faster inference (e.g., real time)
- dynamics hard to capture with ODEs (e.g., discontinuities or long-range dependencies)

Recurrence



hidden state: "memory"

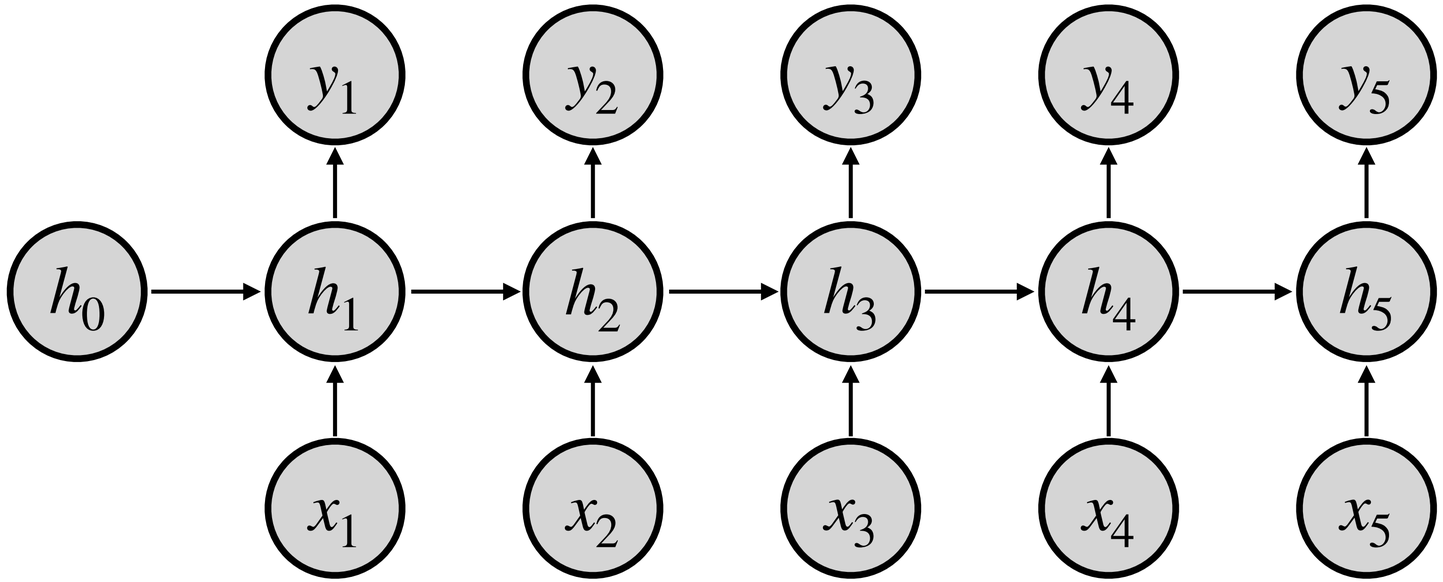
summarizes everything seen so far

Vanilla RNN

outputs:

hidden:

inputs:

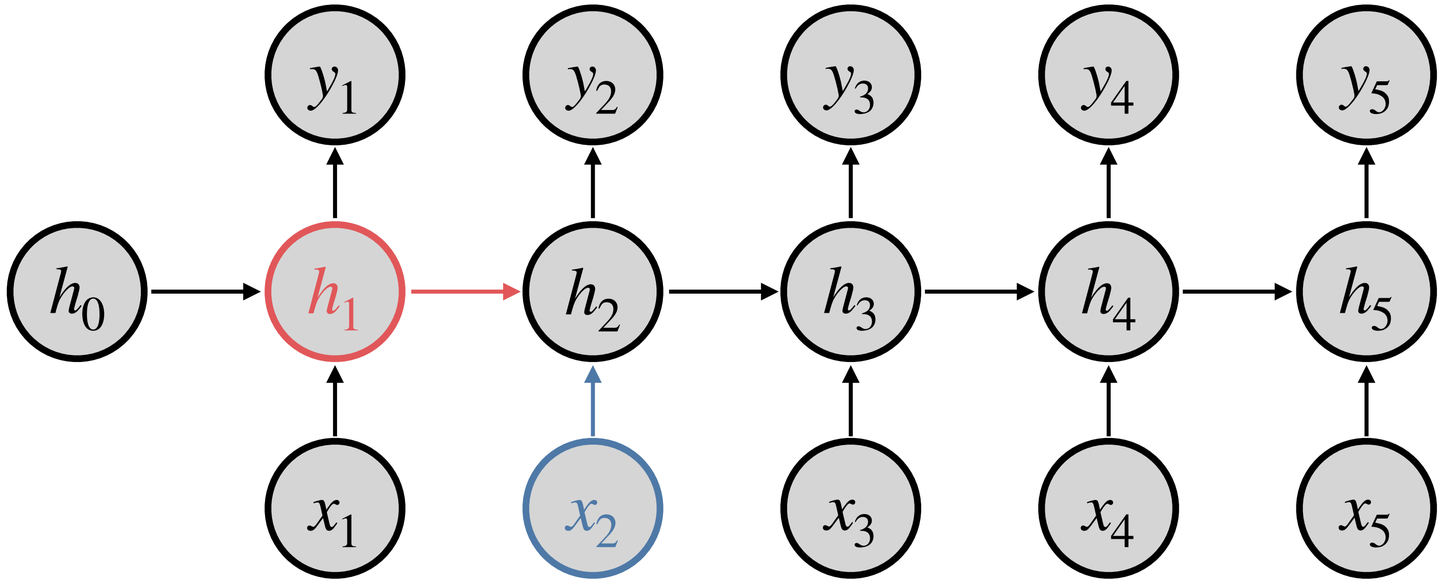


Vanilla RNN

outputs:

hidden:

inputs:



$$h^{(t)} = \tanh\left(W_{hh}h^{(t-1)} + W_{hx}x^{(t)} + b_h\right)$$

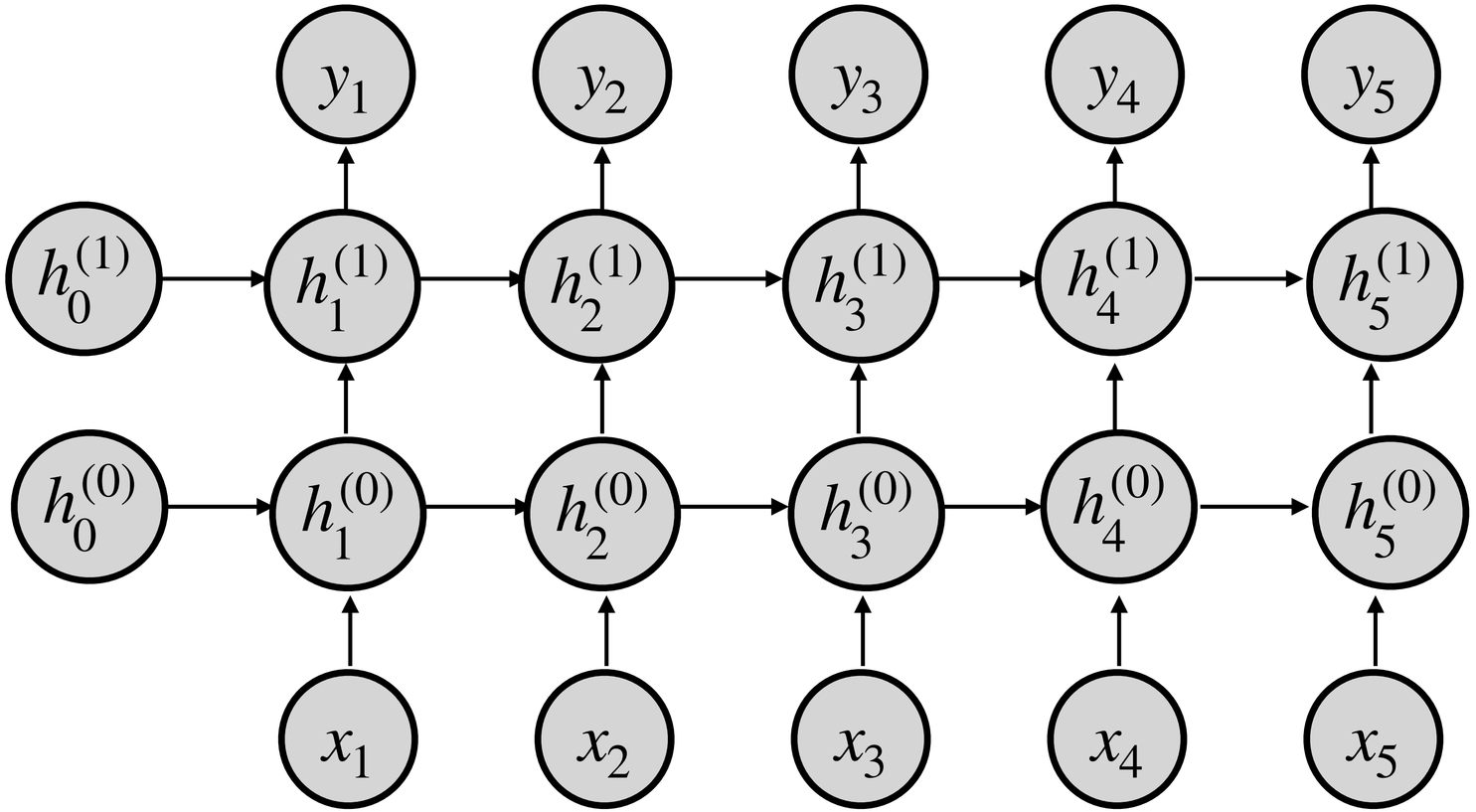
(weights shared across time)

Vanilla RNN

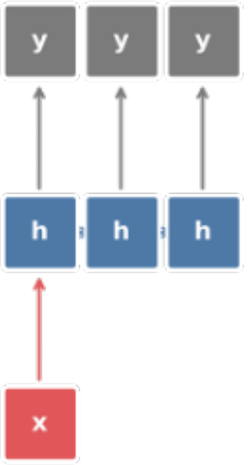
outputs:

hidden:

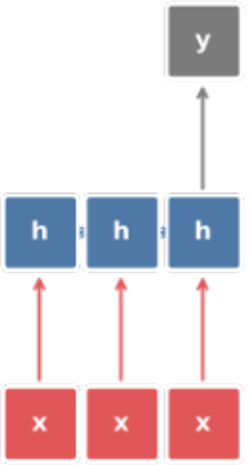
inputs:



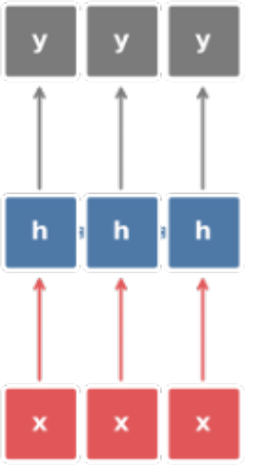
one-to-one



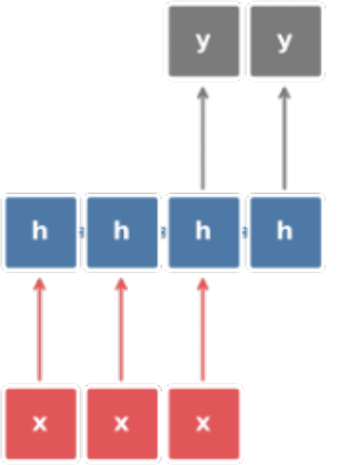
one-to-many



many-to-one

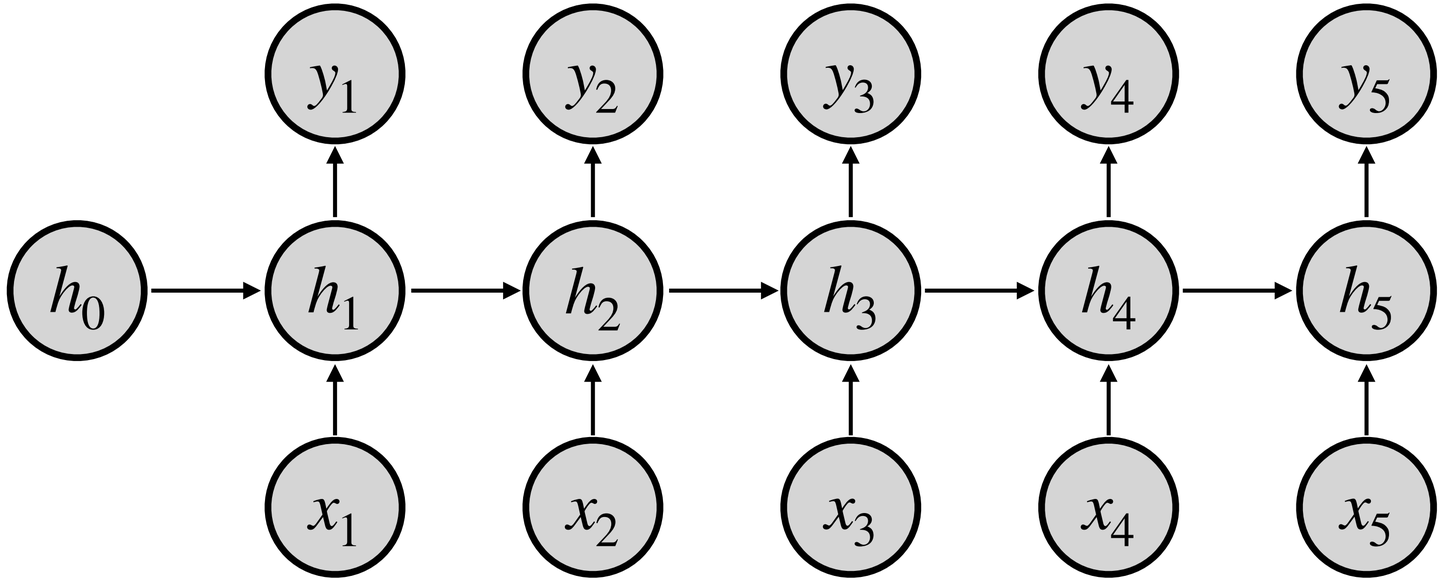


many-to-many



many-to-many

Vanishing Gradients

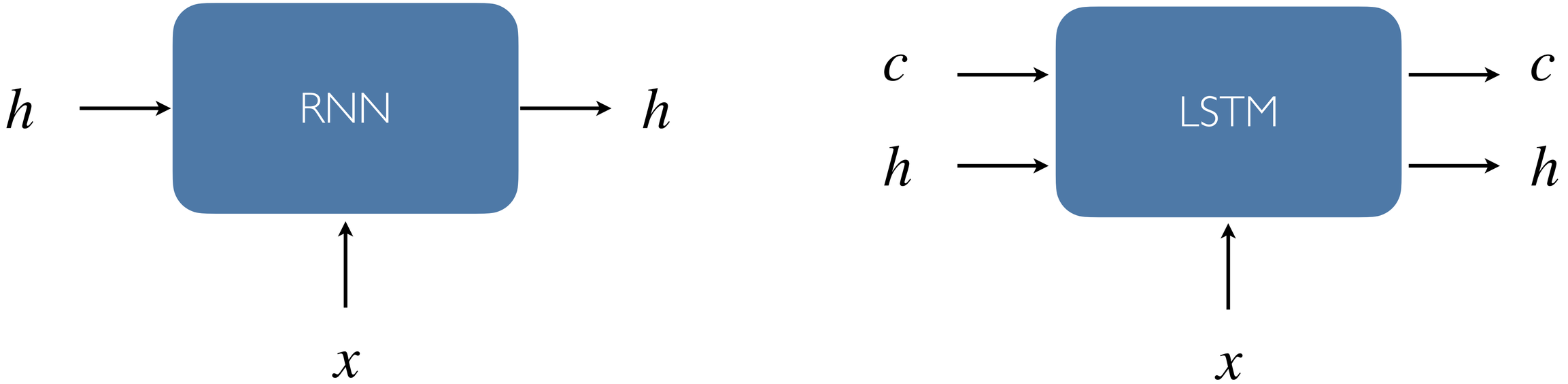


$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_T} \prod_{i=1}^{T-1} W_{hh} \cdot \text{diag}\left(\tanh'(\cdot)\right)$$

Long Short-Term Memory (LSTM)

Address vanishing gradient problem by learning what to remember

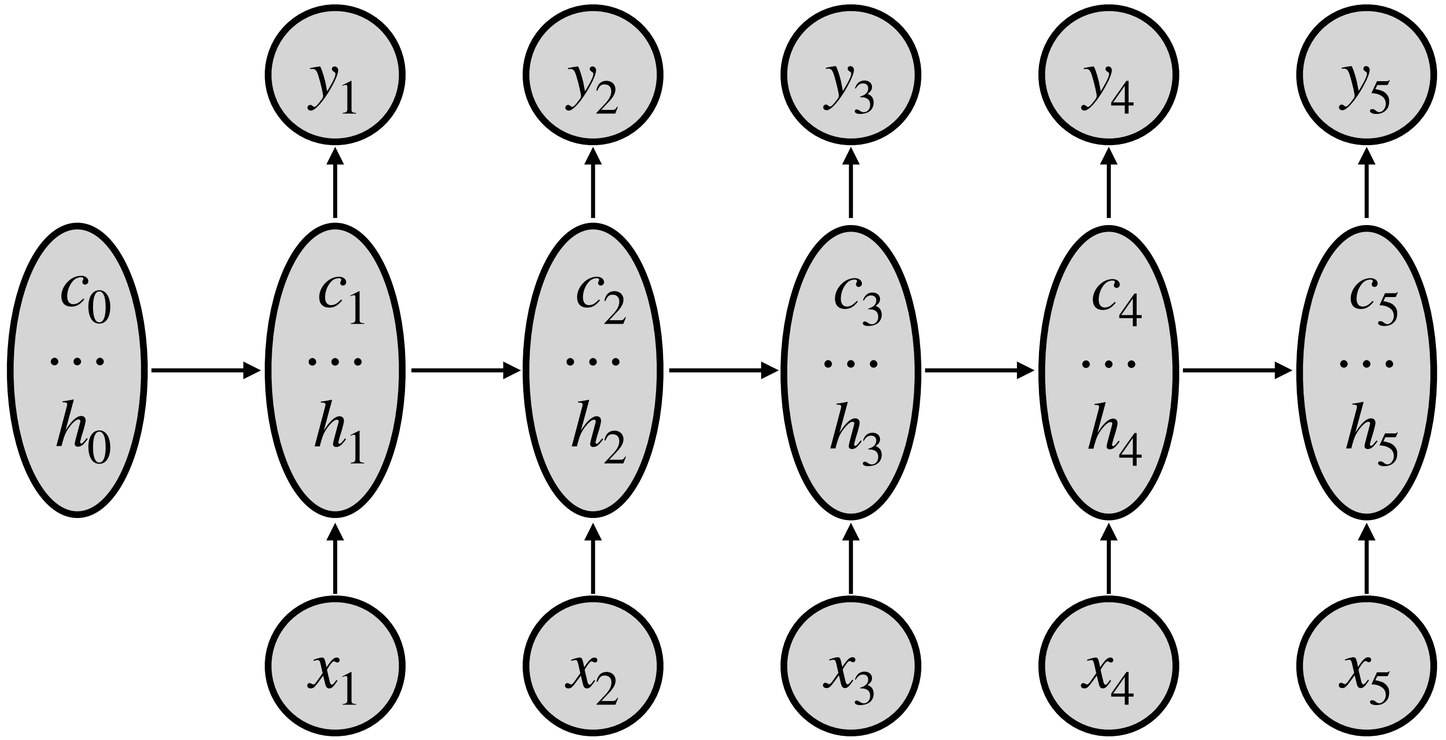
Two Information Pathways



h: working memory

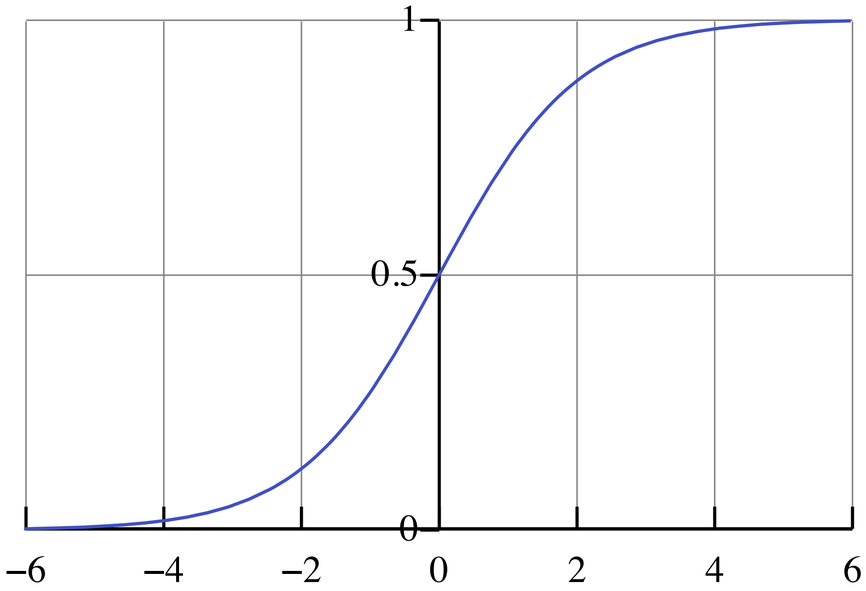
c: long-term storage

Long Short-Term Memory Networks



Gates

sigmoid function

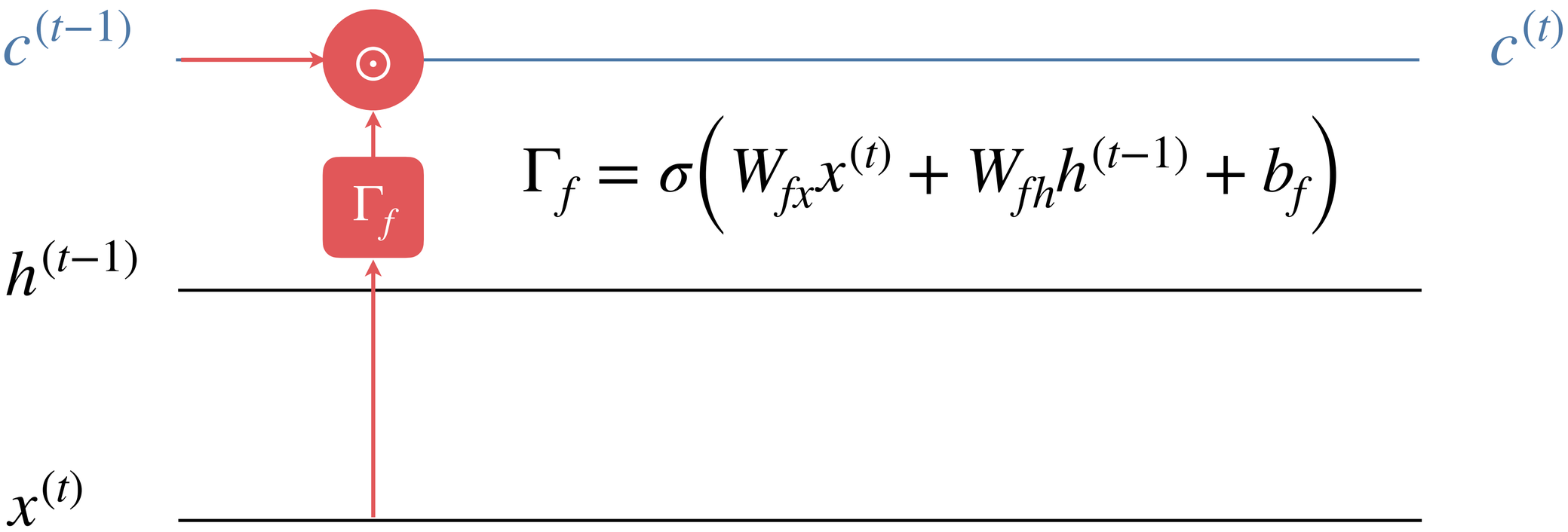


$$\Gamma = \sigma\left(W_x x^{(t)} + W_h h^{(t-1)} + b\right)$$

$\Gamma \odot c$ element-wise multiply

learn what to block and what to pass through

Forget Gate

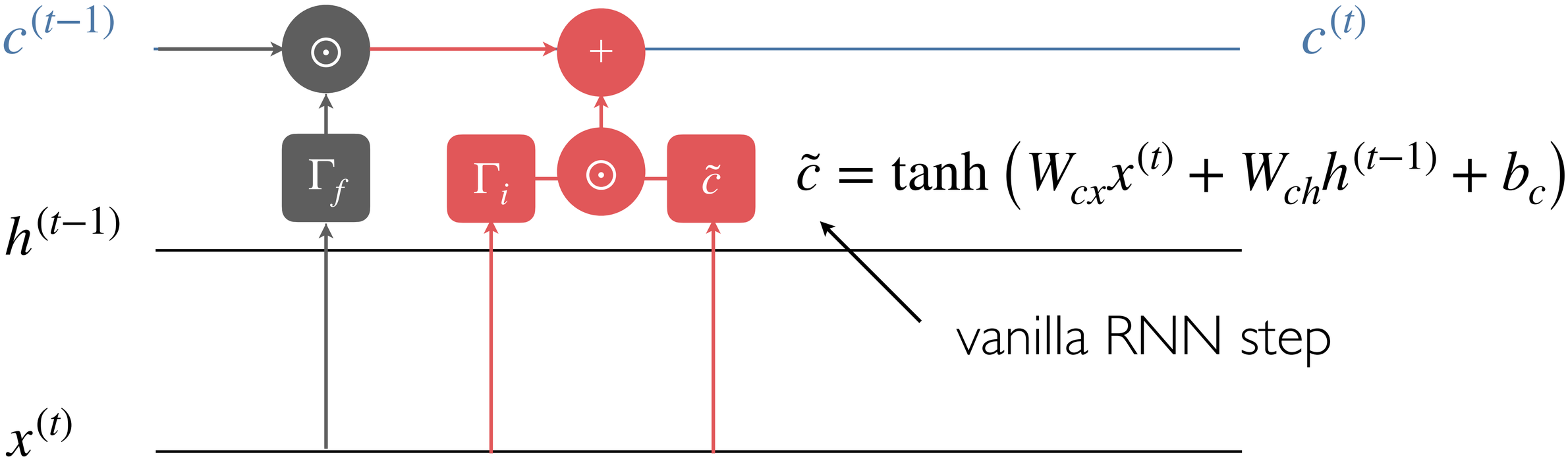


$$\Gamma_f = \sigma\left(W_{fx} x^{(t)} + W_{fh} h^{(t-1)} + b_f \right)$$

learn how much of prior memory to retain

Input Gate

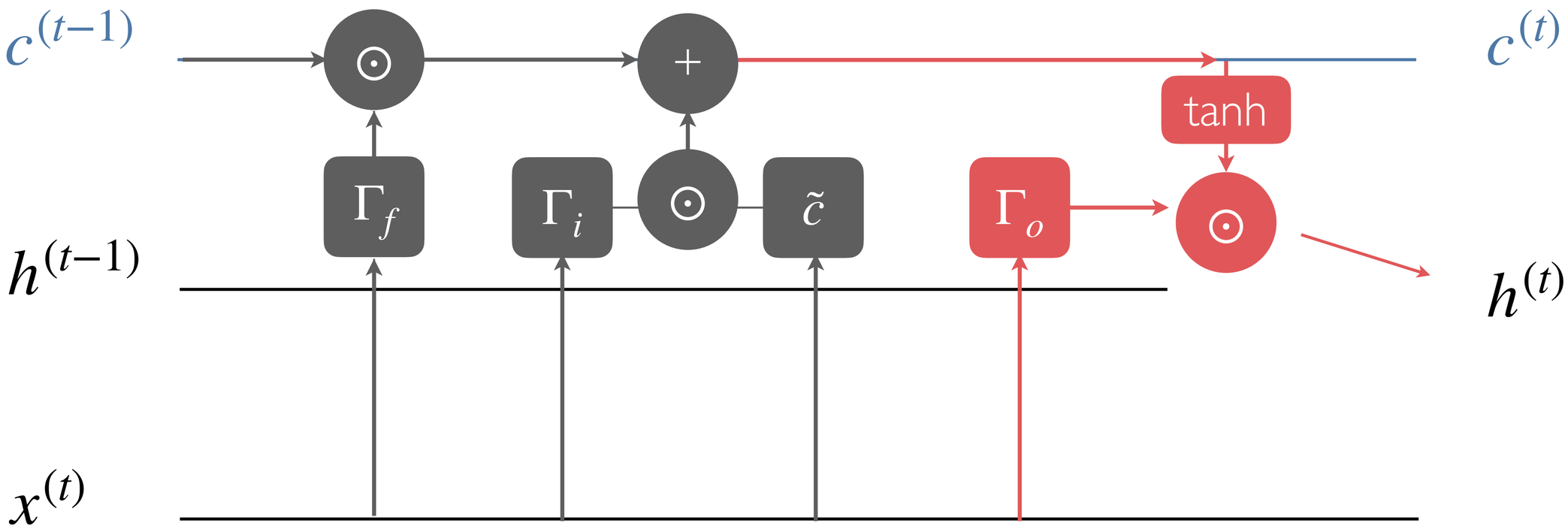
$$\Gamma_i = \sigma\left(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i\right)$$



learn how much to store in long-term

Output Gate

$$\Gamma_o = \sigma\left(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o\right)$$



learn what to pass on to short-term

LSTM

gates:

$$\Gamma_f = \sigma\left(W_{fx} x^{(t)} + W_{fh} h^{(t-1)} + b_f \right)$$

$$\Gamma_i = \sigma\left(W_{ix} x^{(t)} + W_{ih} h^{(t-1)} + b_i \right)$$

$$\Gamma_o = \sigma\left(W_{ox} x^{(t)} + W_{oh} h^{(t-1)} + b_o \right)$$

cell state:

$$\tilde{c}^{(t)} = \tanh(W_{cx} x^{(t)} + W_{ch} h^{(t-1)} + b_c)$$

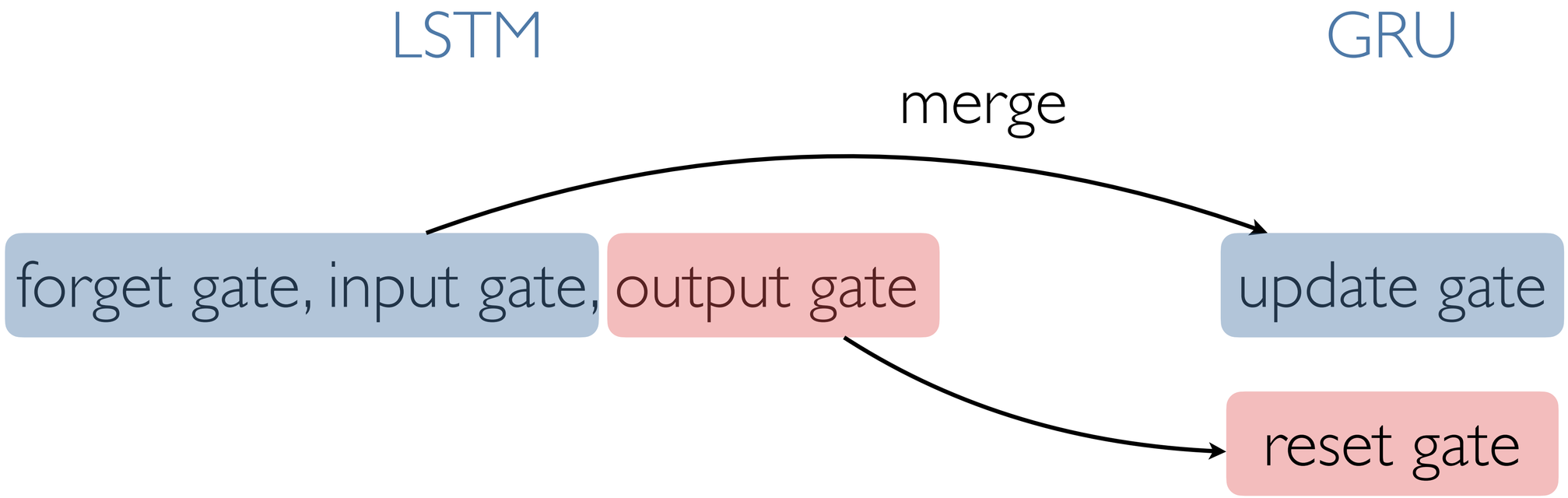
$$c^{(t)} = \Gamma_i \odot \tilde{c}^{(t)} + \Gamma_f \odot c^{(t-1)}$$

hidden state:

$$h^{(t)} = \Gamma_o \odot \tanh(c^{(t)})$$

Gated Recurrent Unit (GRU)

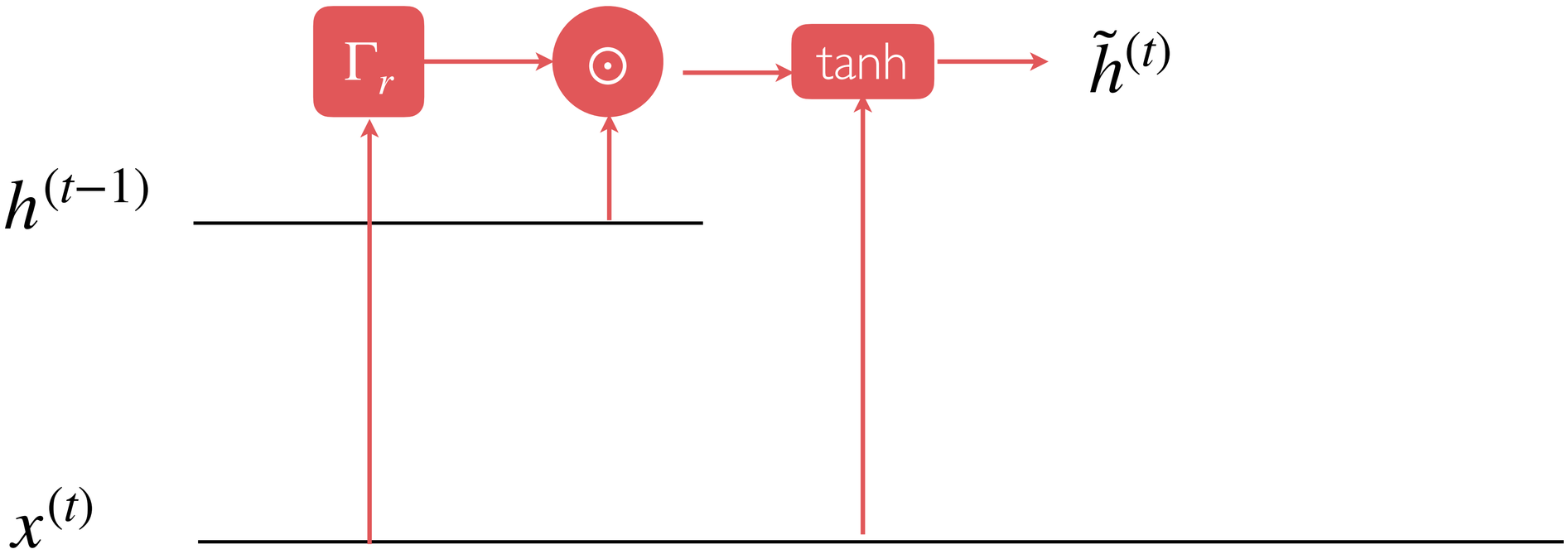
Use gating concept, simplify architecture



cell state, hidden state ⟶ hidden state

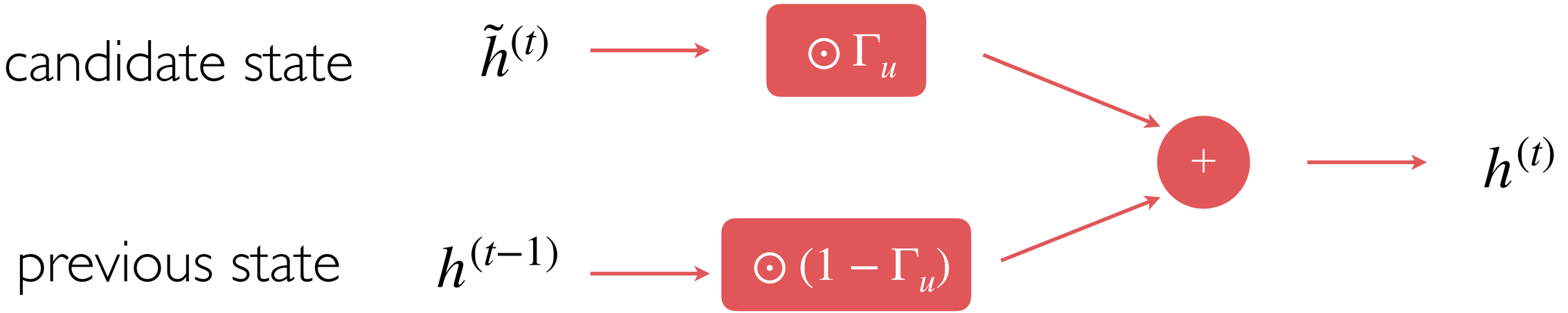
Reset gate

$$\tilde{h}^{(t)} = \tanh(W_{hx}x^{(t)} + W_{hh}\left(\Gamma_r \odot (h^{(t-1)})\right) + b_h)$$



Update gate

$$h^{(t)} = \Gamma_u \odot \tilde{h}^{(t)} + (1 - \Gamma_u) \odot h^{(t-1)}$$



GRU

gates:

$$\Gamma_u = \sigma\left(W_{ux}x^{(t)} + W_{uh}h^{(t-1)} + b_u\right)$$

$$\Gamma_r = \sigma\left(W_{rx}x^{(t)} + W_{rh}h^{(t-1)} + b_r\right)$$

hidden state:

$$\tilde{h}^{(t)} = \tanh(W_{hx}x^{(t)} + W_{hh}\left(\Gamma_r \odot (h^{(t-1)})\right) + b_h)$$

$$h^{(t)} = \Gamma_u \odot \tilde{h}^{(t)} + (1 - \Gamma_u) \odot h^{(t-1)}$$

RNN Training Tips

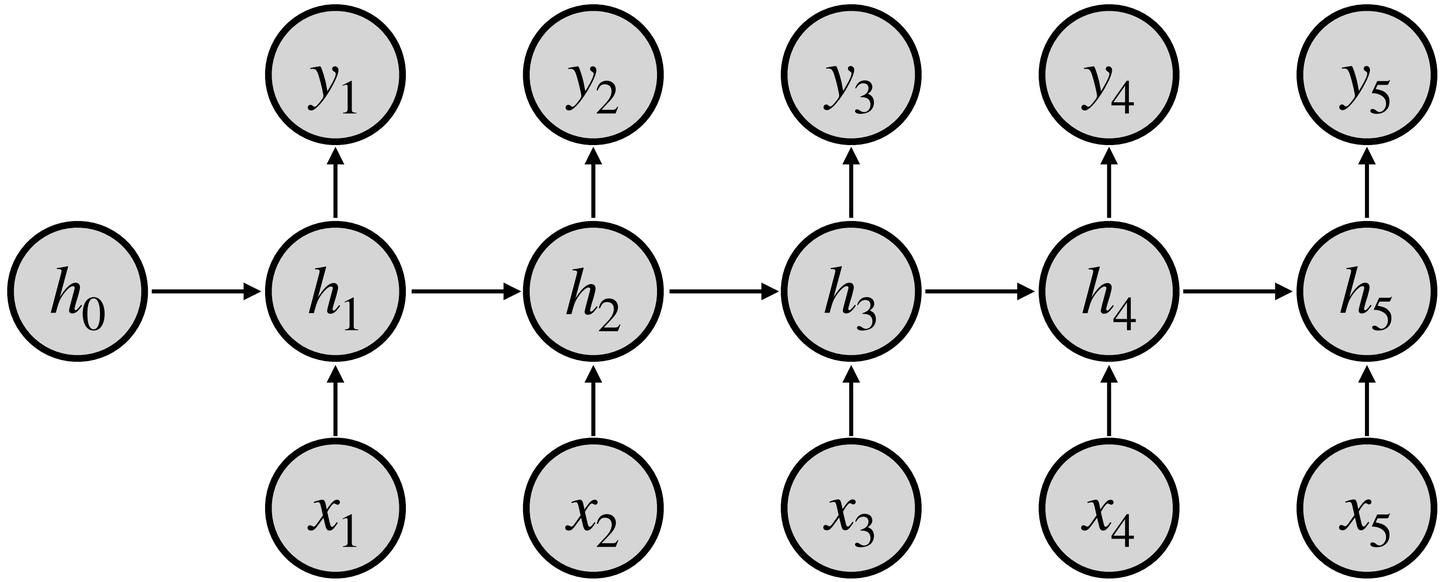
may need gradient clipping

```
torch.nn.utils.clip_grad_norm_(
    model.parameters(), max_norm)

torch.nn.utils.clip_grad_value_(
    model.parameters(), max_value)
```

RNN Training Tips

might need to truncate backpropagation
(only backprop through last k steps)

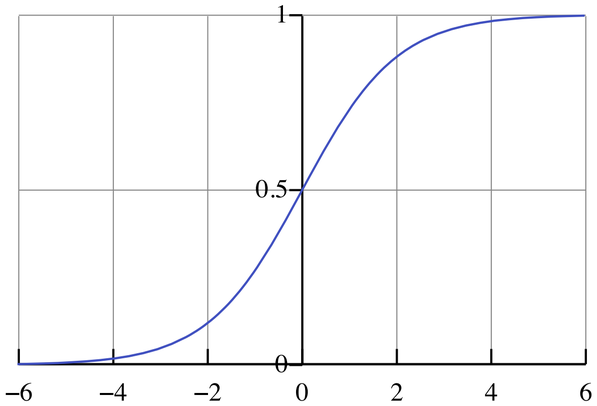


RNN Training Tips

might need better initialization

initialize bias of forget gate to something like +1 or +2
otherwise, you forget a lot of info right off the bat.

$$\Gamma_f = \sigma\left(W_{fx} x^{(t)} + W_{fh} h^{(t-1)} + b_f \right)$$



zero mean at init

zero at init

usually initialized to zero

Demo