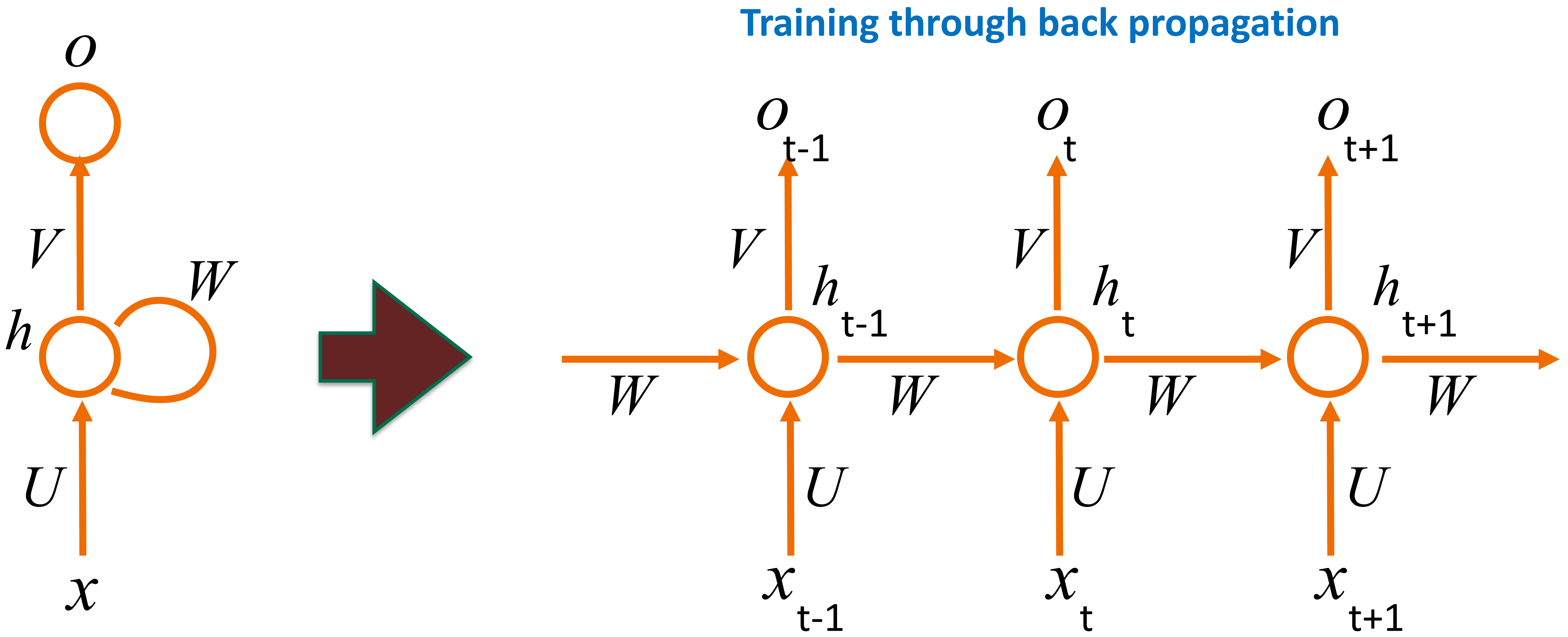
Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- RNN and Learning

RNN and Learning

1

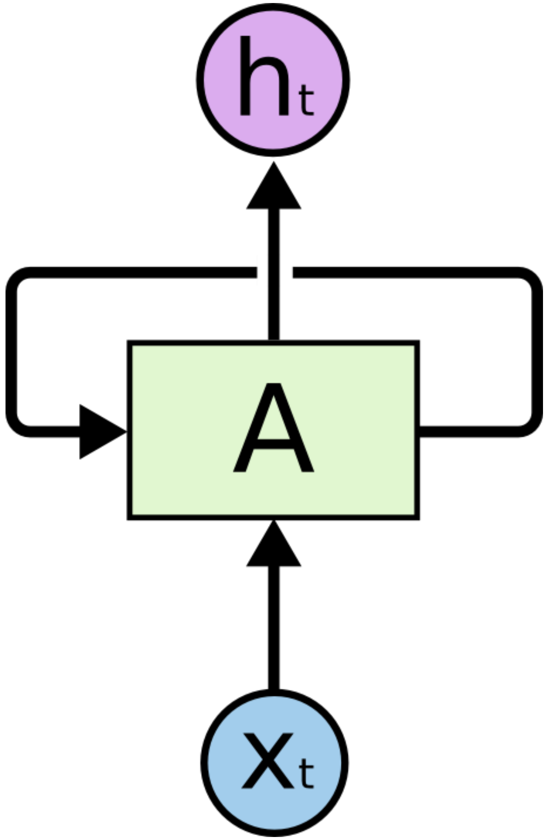
RNN: Recap



$$h_t = f(Ux_t + Wh_{t-1}) \qquad o_t = \text{softmax}(Vh_t)$$

RNN: Recap

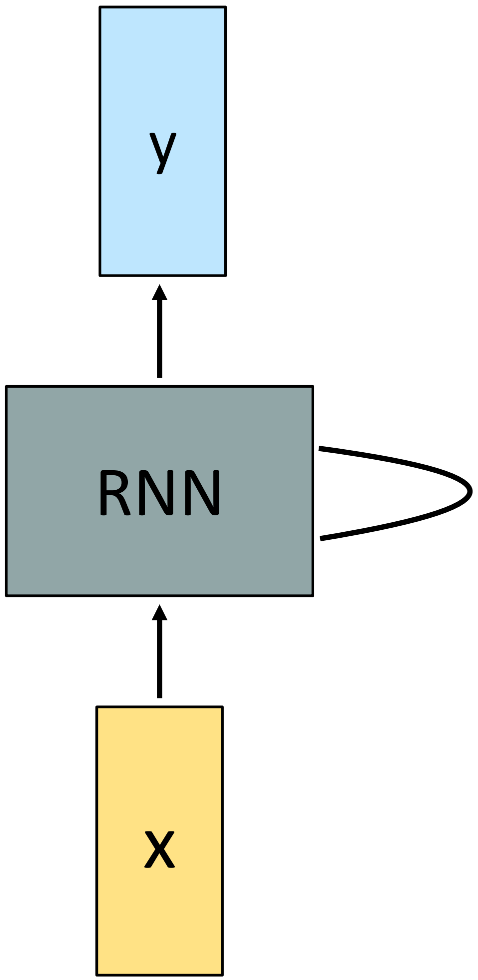
- RNNs



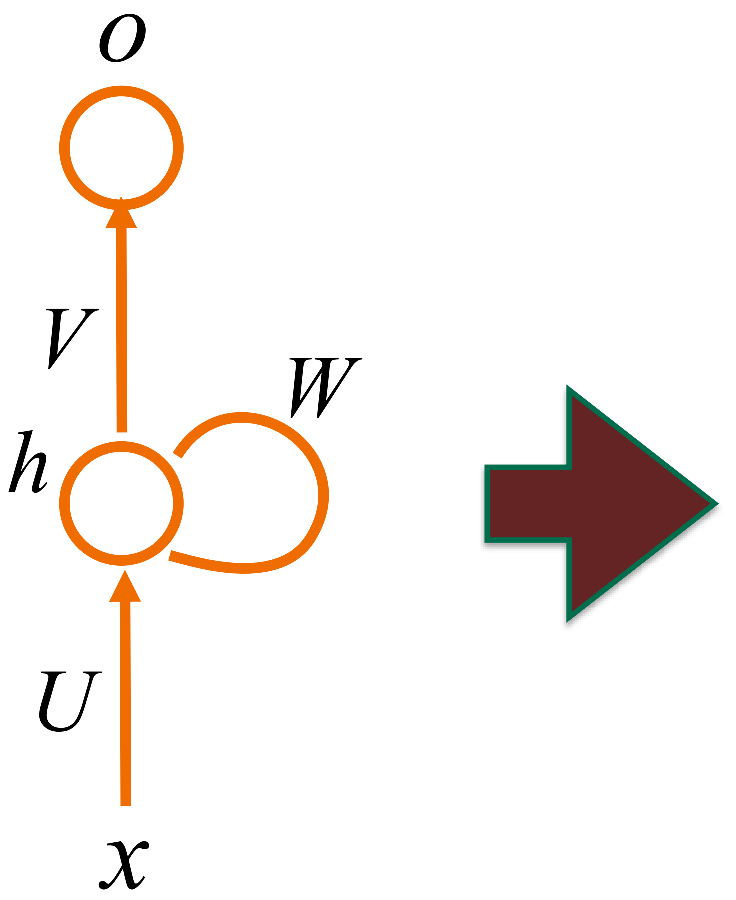

```
*
* Increment the size file of the new incorrect UI_FILTER
* of the size generatively.
*/
tatic int indicate_policy(void)

 int error;
 if (fd == MARN_EPT) {
   /*
    * The kernel blank will coeld it to userspace.
    */
   if (ss->segment < mem_total)
     unblock_graph_and_set_blocked();
   else
     ret = 1;
   goto bail;
 }
 segaddr = in_SB(in.addr);
 selector = seg / 16;
 setup_works = true;
 for (i = 0; i < blocks; i++) {
   seq = buf[i++];
   bpf = bd->bd.next + i * search;
   if (fd) {
     current = blocked;
```

A. Karpathy http://karpathy.github.io/2015/05/21/rnn-effectiveness/

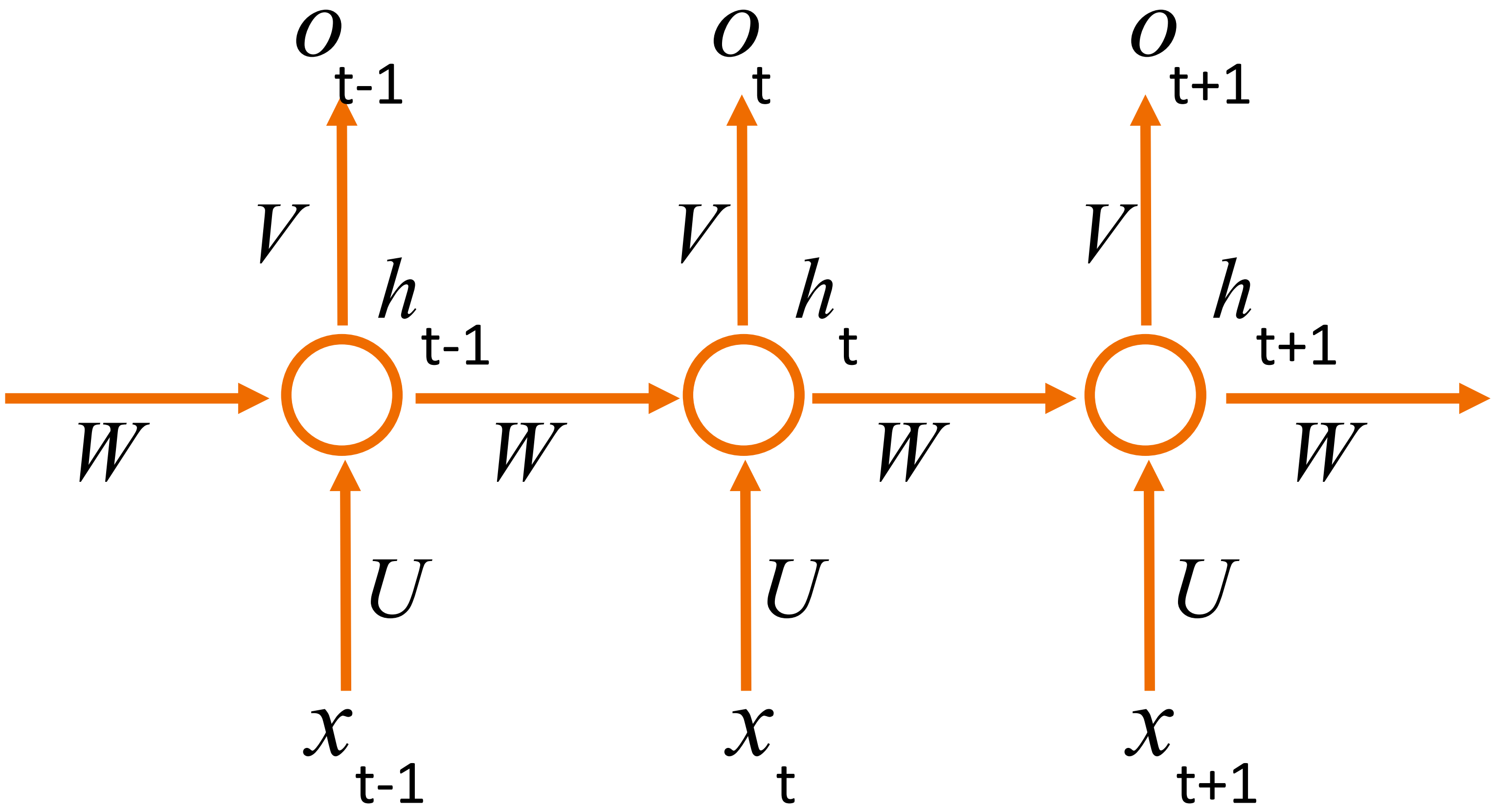
RNN basic architecture



RNN basic architecture

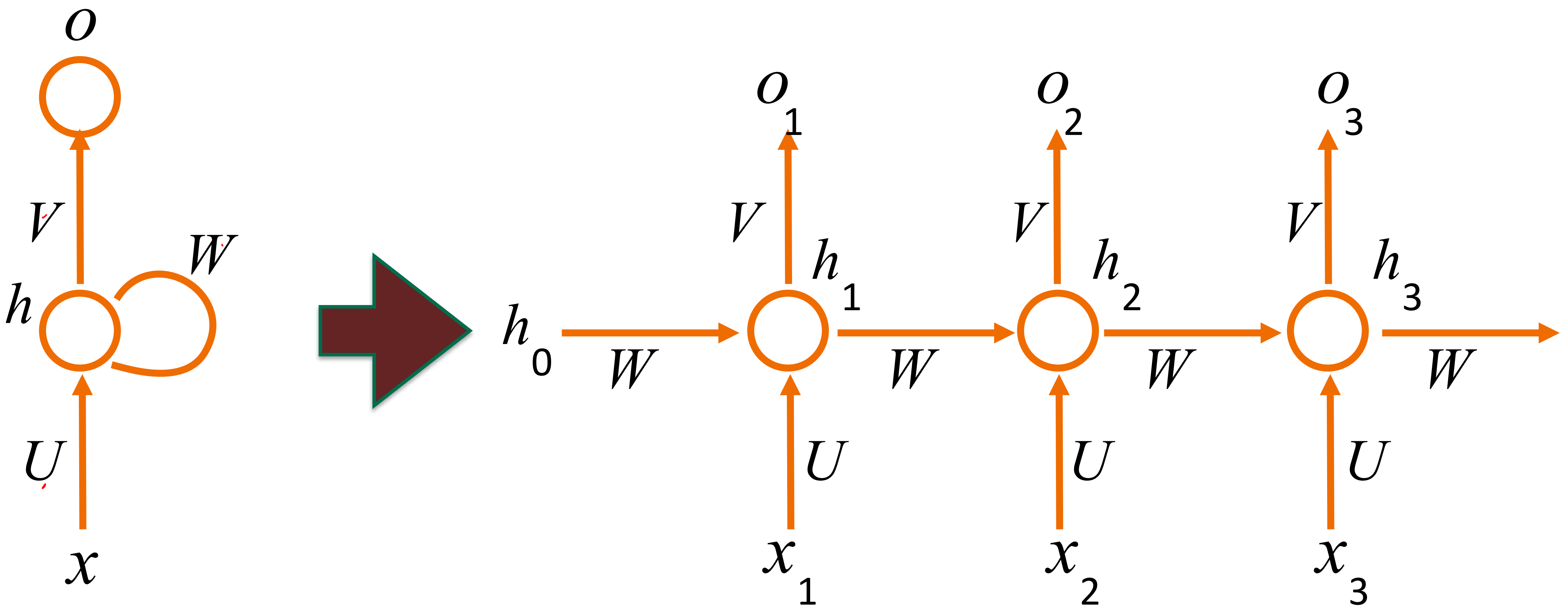


Training through back propagation



$$h_t = f(Ux_t + Wh_{t-1}) \qquad o_t = \mathrm{softmax}(Vh_t)$$

RNN basic architecture

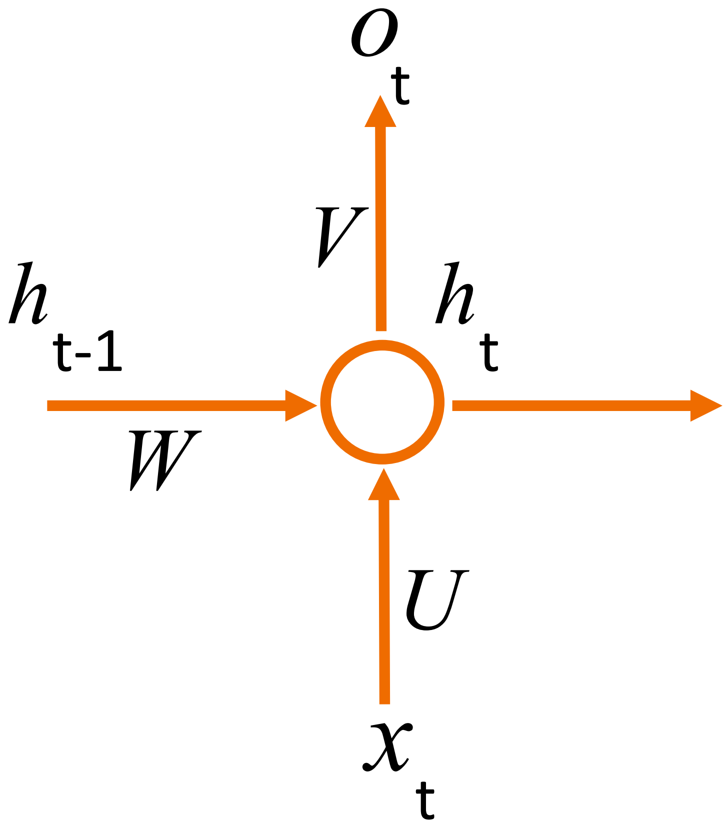


$$h_t = f(Ux_t + Wh_{t-1}) \qquad o_t = \text{softmax}(Vh_t)$$

RNN basic architecture

- x_t - input at time step t
- h_t - hidden state at time step t (memory of the network)
- o_t - output at time step t
- *U,V,W* are parameters (shared across all layers)

RNN basic architecture



$$\boldsymbol{h_t = f(Ux_t + Wh_{t-1})}$$

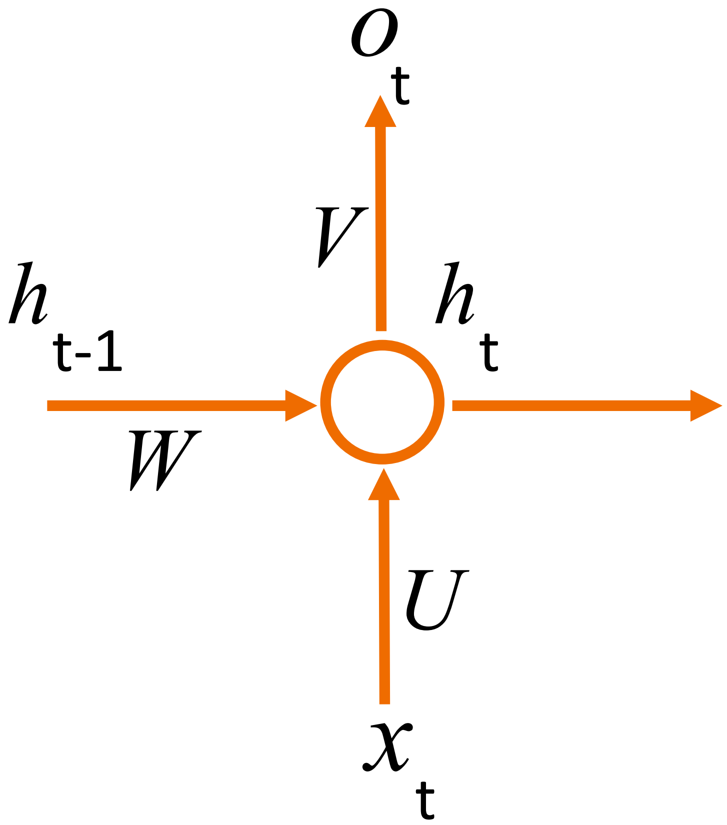
$$U = \begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}$$

$$W = \begin{pmatrix} 0.4 & 0.5 \\ 0.3 & 0.5 \end{pmatrix} \quad x_t = \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix} \quad h_{t-1} = \begin{pmatrix} 0.3 \\ 0.8 \end{pmatrix}$$

$$h_t = \tanh\left(\overset{U}{\begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}} \overset{x_t}{\begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix}} + \overset{W}{\begin{pmatrix} 0.4 & 0.5 \\ 0.3 & 0.5 \end{pmatrix}} \overset{h_{t-1}}{\begin{pmatrix} 0.3 \\ 0.8 \end{pmatrix}} \right)$$

$$h_t = \tanh\left(\begin{pmatrix} .28 \\ .28 \end{pmatrix} + \begin{pmatrix} .52 \\ .29 \end{pmatrix} \right) = \tanh\begin{pmatrix} .88 \\ .77 \end{pmatrix} = \begin{pmatrix} .66 \\ .64 \end{pmatrix}$$

RNN basic architecture



$$h_t = f(Ux_t + Wh_{t-1})$$

$$o_t = softmax(Vh_t)$$

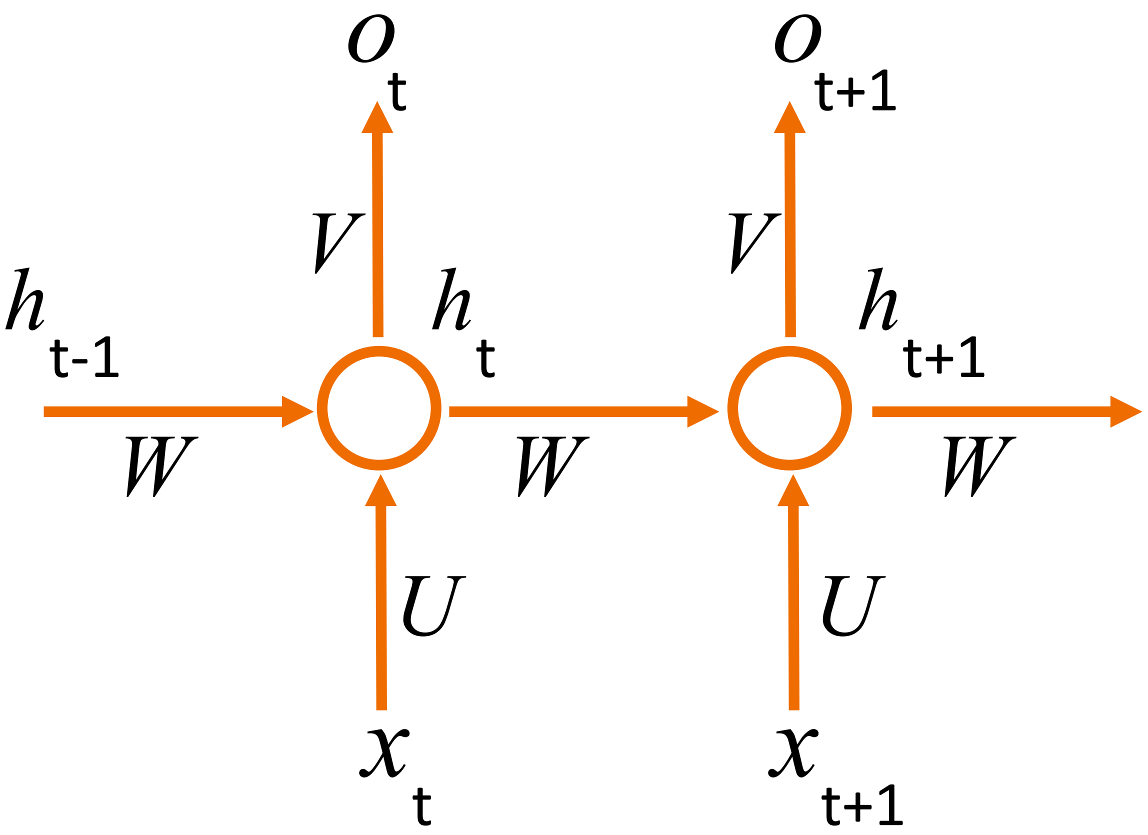
$$U = \begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}$$

$$W = \begin{pmatrix} 0.4 & 0.5 \\ 0.3 & 0.5 \end{pmatrix} \quad x_t = \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix} \quad h_{t-1} = \begin{pmatrix} 0.3 \\ 0.8 \end{pmatrix}$$

$$h_t = \begin{pmatrix} .66 \\ .64 \end{pmatrix} \quad V = \begin{pmatrix} 0.4 & 0.7 \\ 0.3 & 0.1 \end{pmatrix}$$

$$O_t = softmax\left(\underset{V}{\begin{pmatrix} 0.4 & 0.7 \\ 0.3 & 0.1 \end{pmatrix}} \underset{h_t}{\begin{pmatrix} .66 \\ .64 \end{pmatrix}}\right) = \begin{pmatrix} .61 \\ .39 \end{pmatrix}$$

RNN basic architecture



$$h_{t+1} = f(Ux_{t+1} + Wh_t)$$

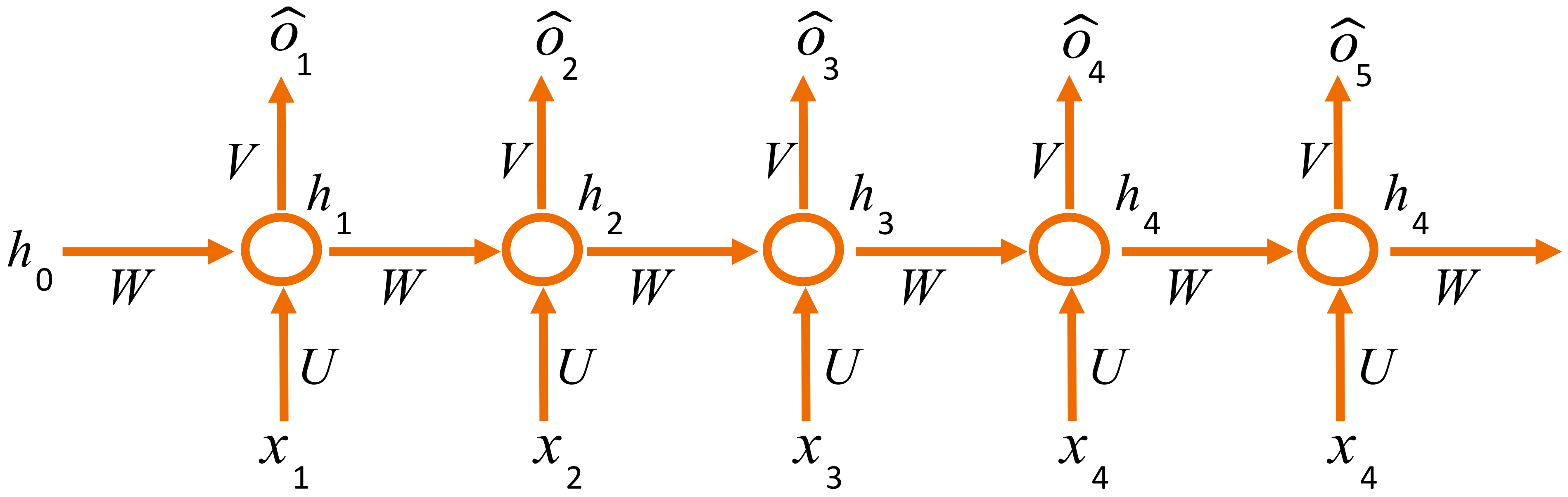
$$U = \begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}$$

$$W = \begin{pmatrix} 0.4 & 0.5 \\ 0.3 & 0.5 \end{pmatrix} \quad x_{t+1} = \begin{pmatrix} 0.5 \\ 0.4 \end{pmatrix} \quad h_t = \begin{pmatrix} .66 \\ .64 \end{pmatrix}$$

$$h_{t+1} = \tanh\left(\underbrace{\begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}}_{U}\underbrace{\begin{pmatrix} 0.5 \\ 0.4 \end{pmatrix}}_{x_{t+1}} + \underbrace{\begin{pmatrix} 0.4 & 0.5 \\ 0.3 & 0.5 \end{pmatrix}}_{W}\underbrace{\begin{pmatrix} .66 \\ .64 \end{pmatrix}}_{h_t}\right)$$

Blank Slide

Forward Pass, Loss



$$h_t = f(Ux_t + Wh_{t-1})$$

$$\hat{o}_t = \text{softmax}(Vh_t)$$

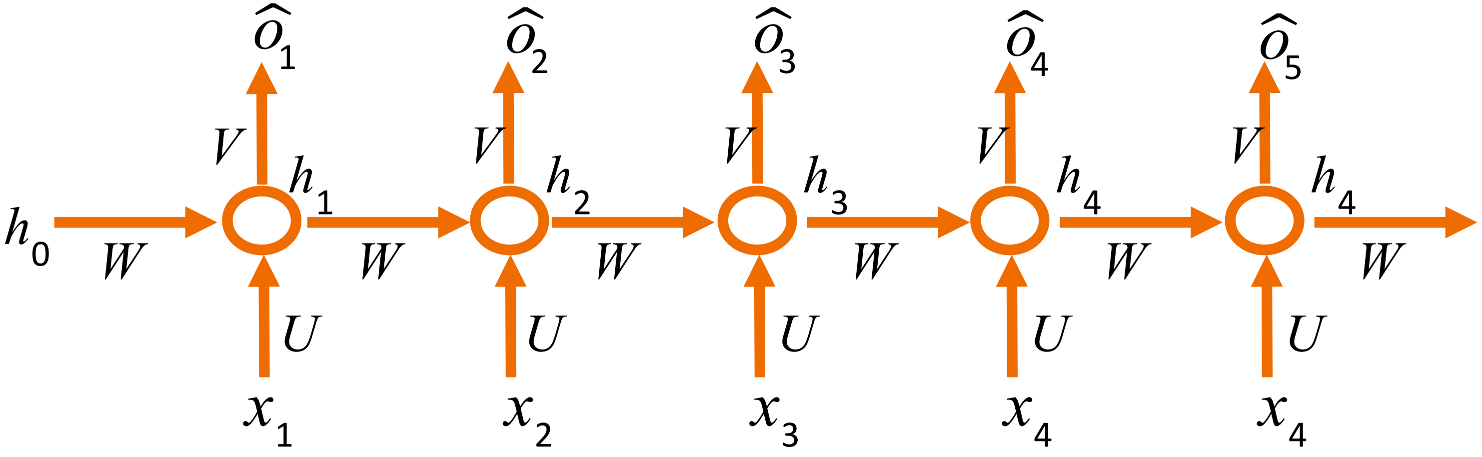
$$E(o, \hat{o}) = \sum_t E_t(o_t, \hat{o}_t)$$

Back Propagation Through Time (BPTT)

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_4}{\partial W} = \frac{\partial E_4}{\partial \hat{o}_4} \frac{\partial \hat{o}_4}{\partial h_4} \frac{\partial h_4}{\partial W}$$

But, $h_4 = f(Ux_3 + Wh_3)$



i.e., h_4 depends on W and h_3;
h_3 depends on W and h_2 and so on..

$$\frac{\partial E_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial E_4}{\partial \hat{o}_4} \frac{\partial \hat{o}_4}{\partial h_4} \frac{\partial h_4}{\partial h_k} \frac{\partial h_k}{\partial W} \Rightarrow \frac{\partial E_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial E_4}{\partial \hat{o}_4} \frac{\partial \hat{o}_4}{\partial h_4} \left(\prod_{j=k+1}^{4} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

This is our good-old chain rule.
[Do not worry about the exact equation]

Vanishing Gradients

$$\frac{\partial E_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial E_4}{\partial \hat{o}_4} \frac{\partial \hat{o}_4}{\partial h_4} \frac{\partial h_4}{\partial h_k} \frac{\partial h_k}{\partial W}$$

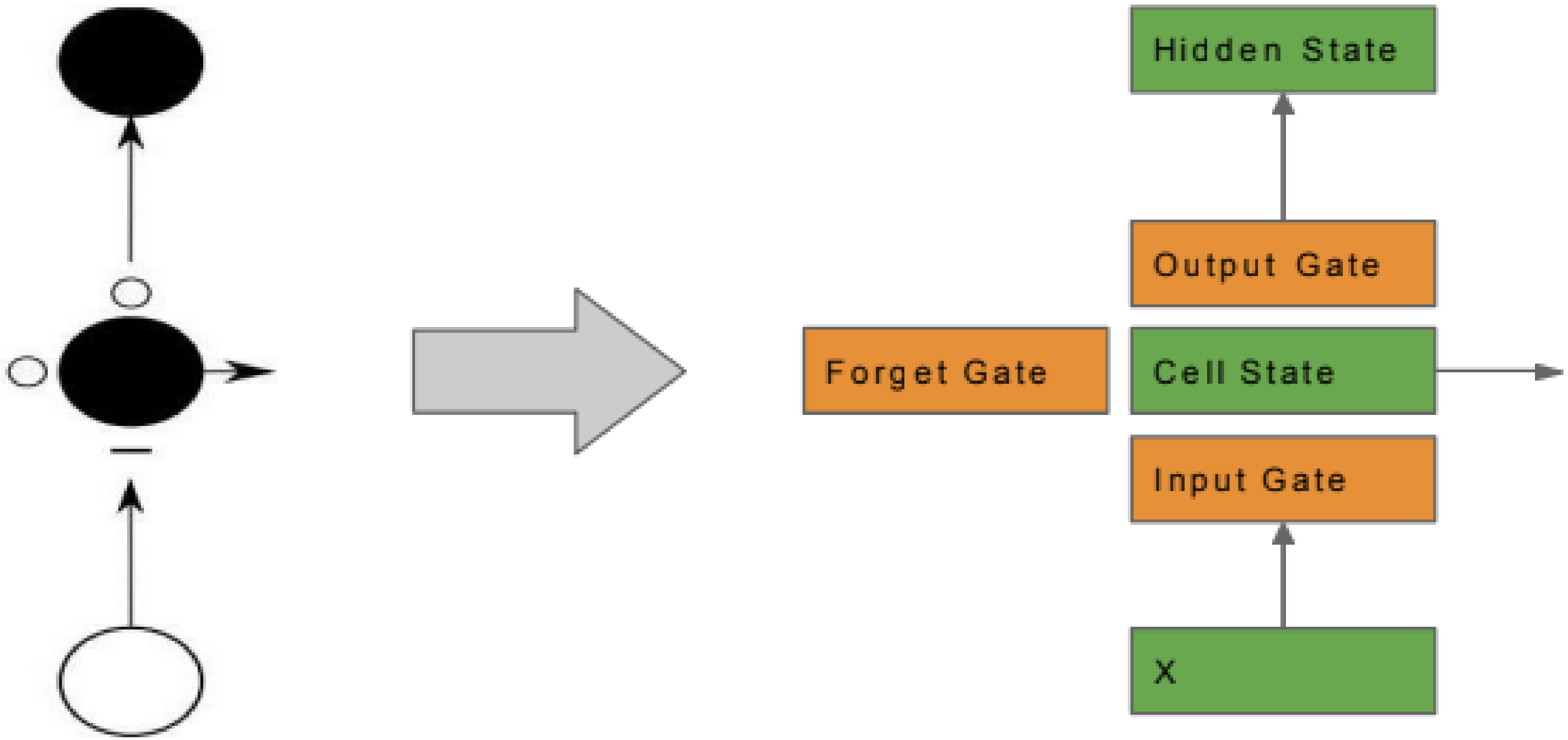
$$\frac{\partial E_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial E_4}{\partial \hat{o}_4} \frac{\partial \hat{o}_4}{\partial h_4} \left(\prod_{j=k+1}^{4} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

- If a quantity is less than 1.0, then its continued multiplication can lead to vanishing
- If a quantity is greater than 1.0, then its continued multiplication can lead to explosion
- This is true for matrices as well.

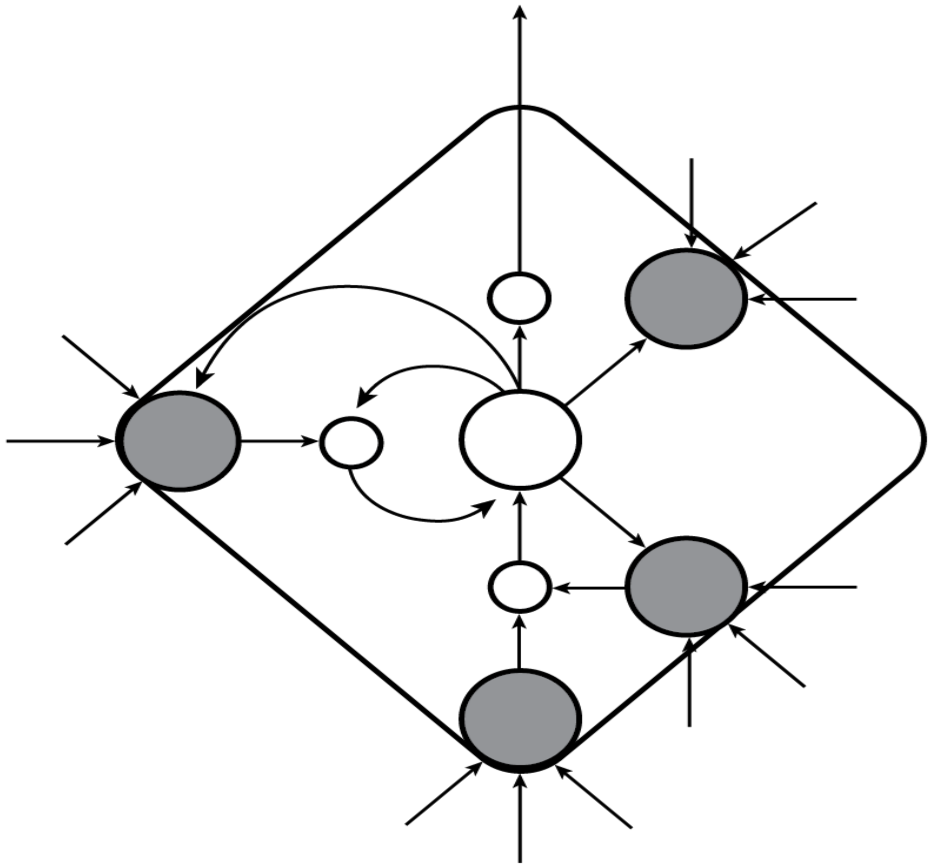
Blank Slide

Questions?

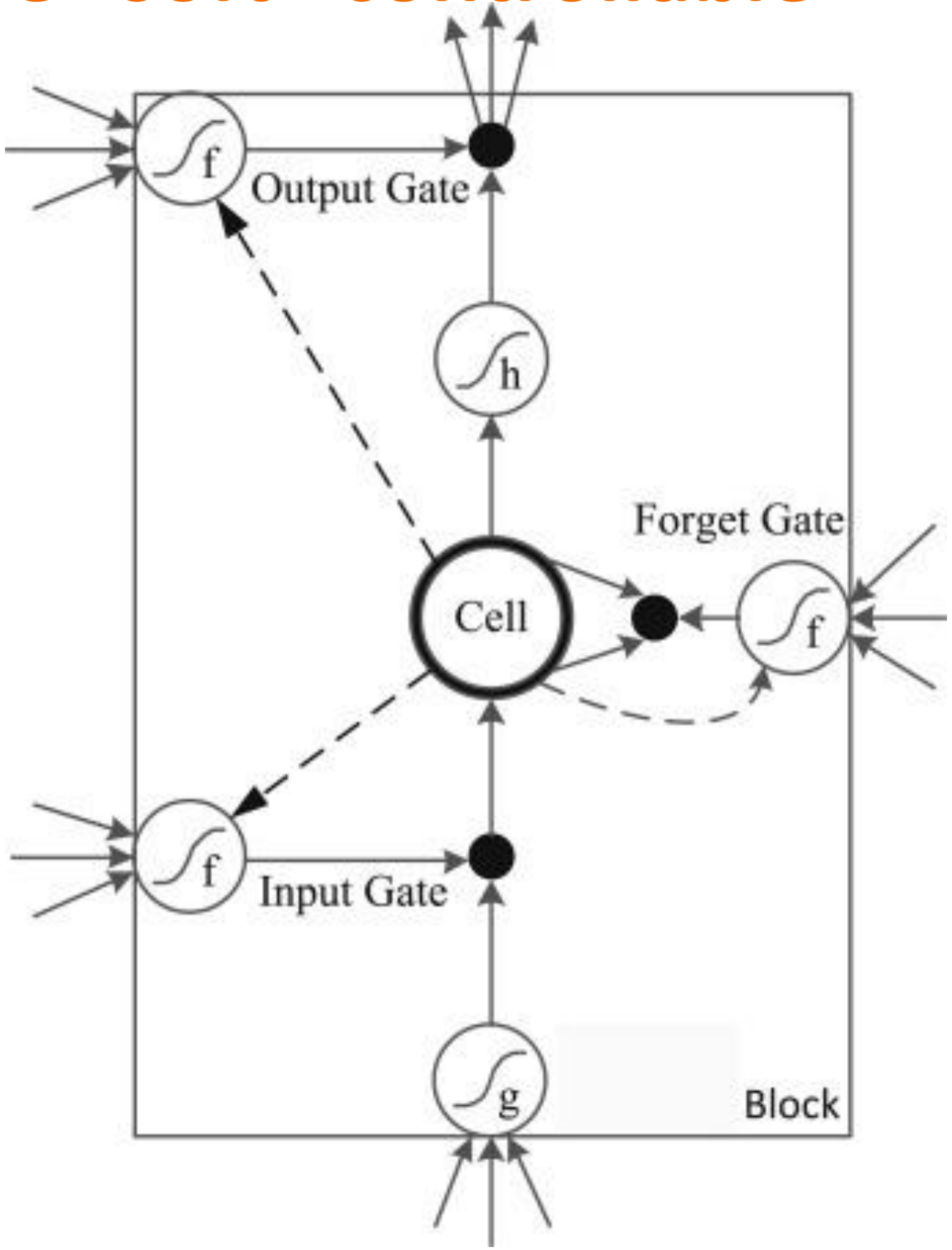
LSTM Node


Hidden State
Output Gate
Forget Gate
Cell State
Input Gate
X

Gates/Switches are “Controllable”



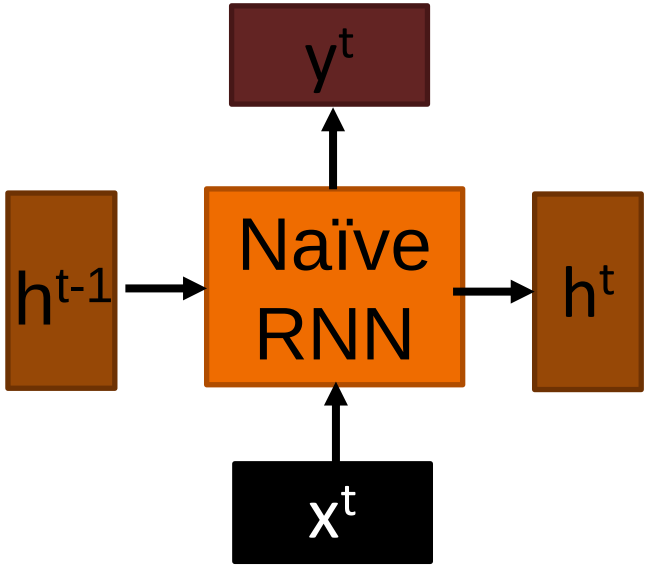
LSTM : Gates are “soft” controllable

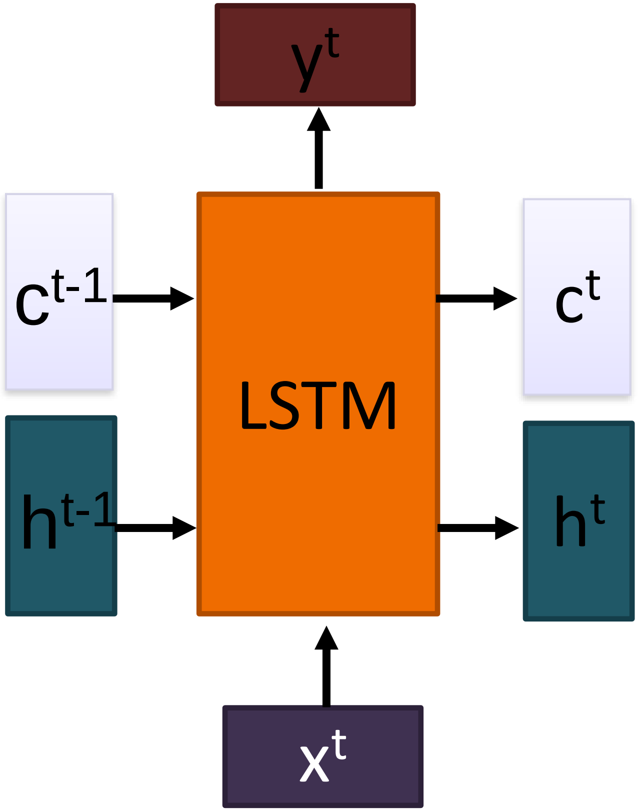


Specific Equations are Avoided

Blank Slide

Naïve RNN vs LSTM

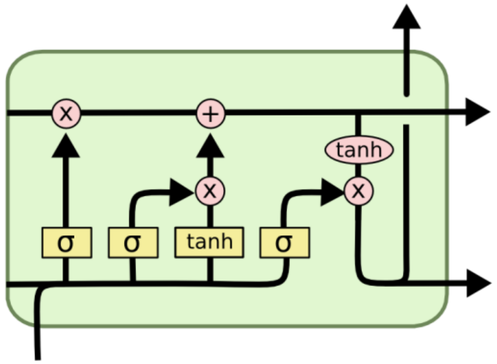




c changes slowly ➡ c^t is c^{t-1} added by something

h changes faster ➡ h^t and h^{t-1} can be very different

c^{t-1}



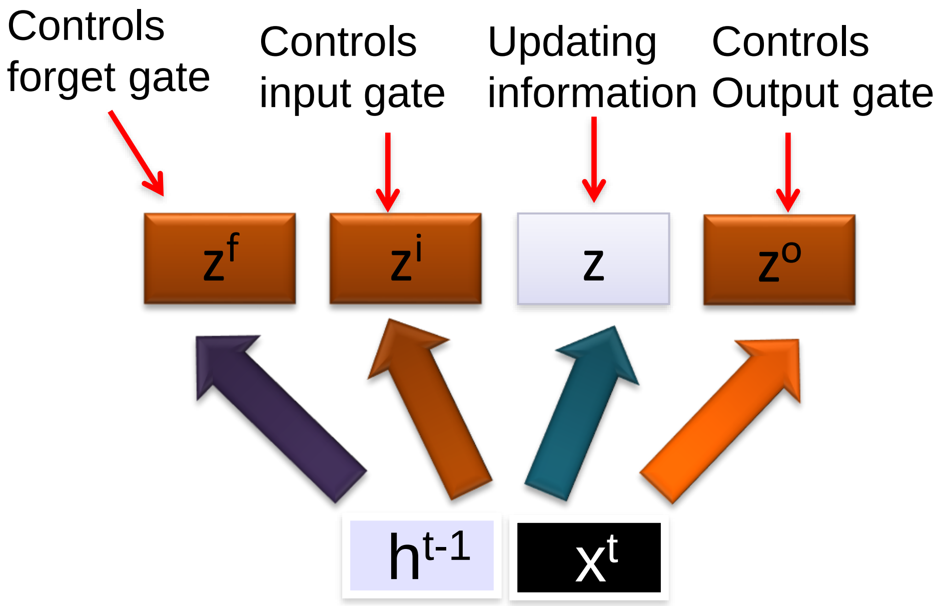
These 4 matrix computation should be done concurrently.

$$z = tanh\left(W \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \right)$$

$$z^i = \sigma\left(W^i \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \right)$$

$$z^f = \sigma\left(W^f \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \right)$$

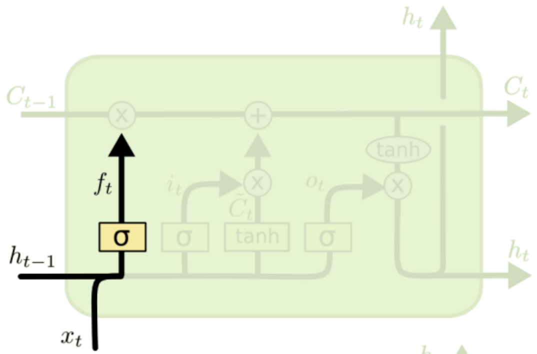
$$z^o = \sigma\left(W^o \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \right)$$



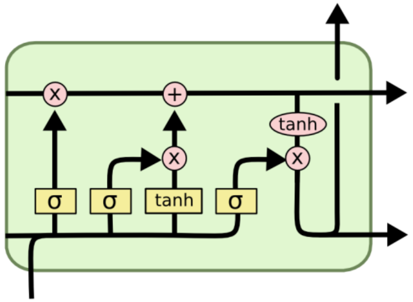
Information flow of LSTM

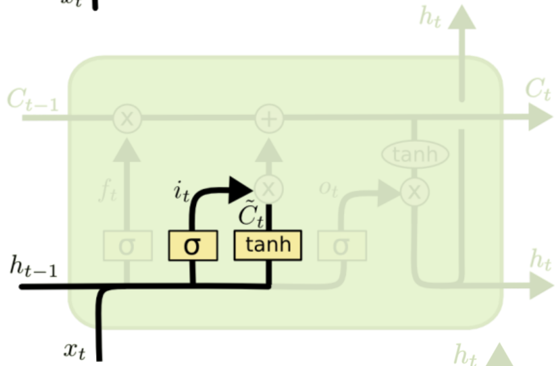
Summary of Steps

- Step 1: Decide How Much Past Data It Should Remember
- Step 2: Decide How Much This Unit Adds to the Current State
- Step 3: Decide What Part of the Current Cell State Makes It to the Output



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

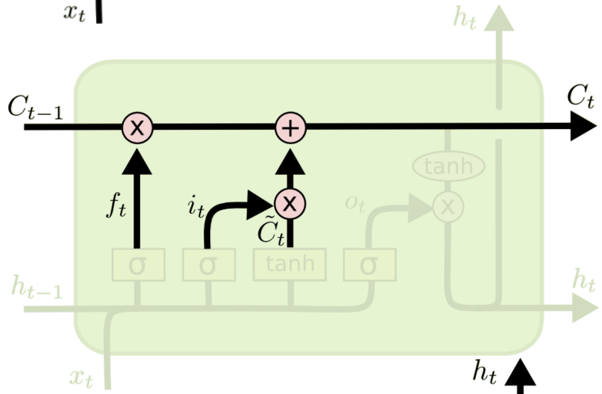




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

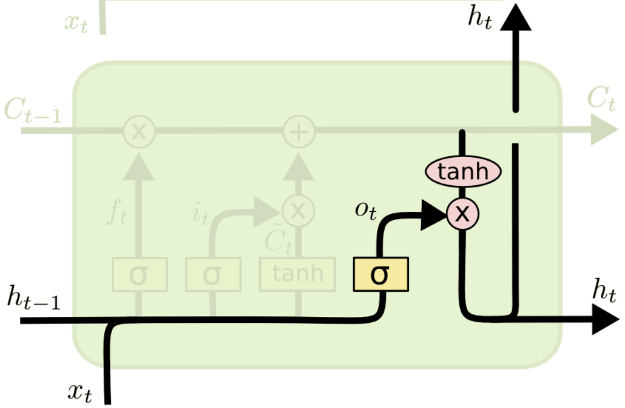
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

i_t decides what component is to be updated.
C'$_t$ provides change contents



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updating the cell state



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Decide what part of the cell state to output

Summary of Steps

Step 1: Decide How Much Past Data It Should Remember

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$

f_t = forget gate
Decides which information to delete that is not important from previous time step

Step 2: Decide How Much This unit adds to the current state

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

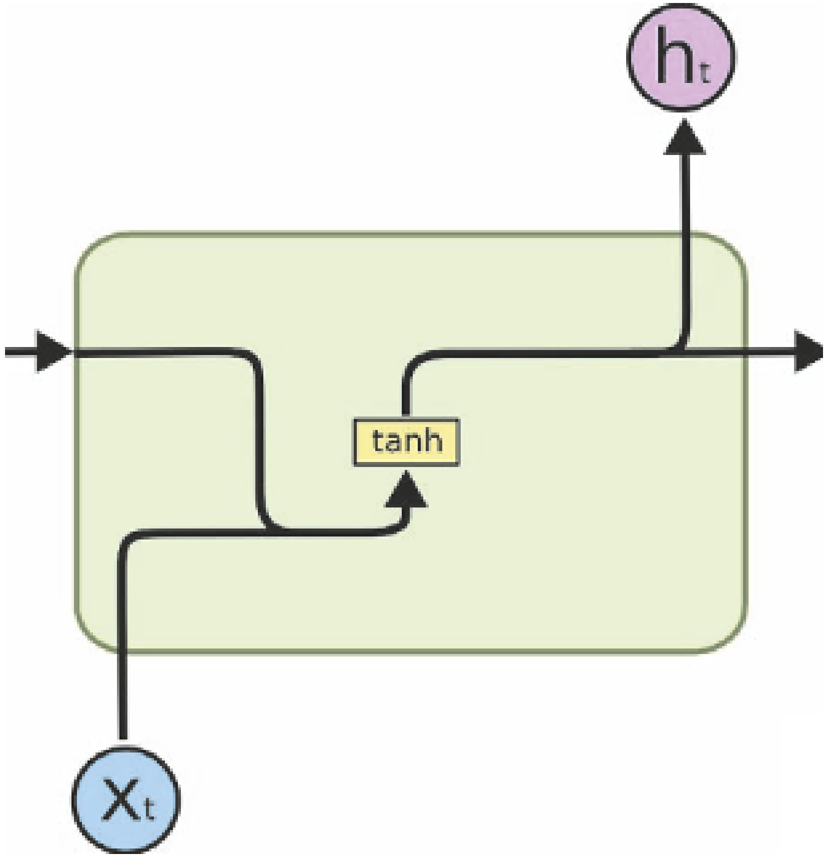
i_t = input gate
Determines which information to let through based on its significance in the current time step

Step 3: Decide What part of the current state makes to the output.

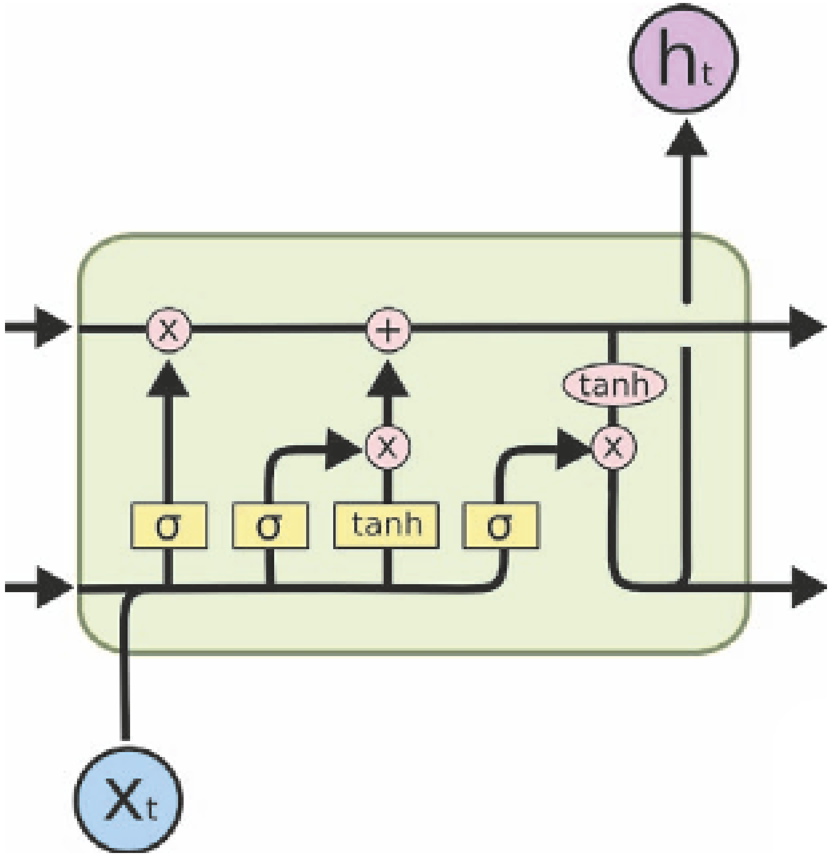
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \;+\; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

o_t = output gate
Allows the passed in information to impact the output in the current time step

RNN vs LSTM

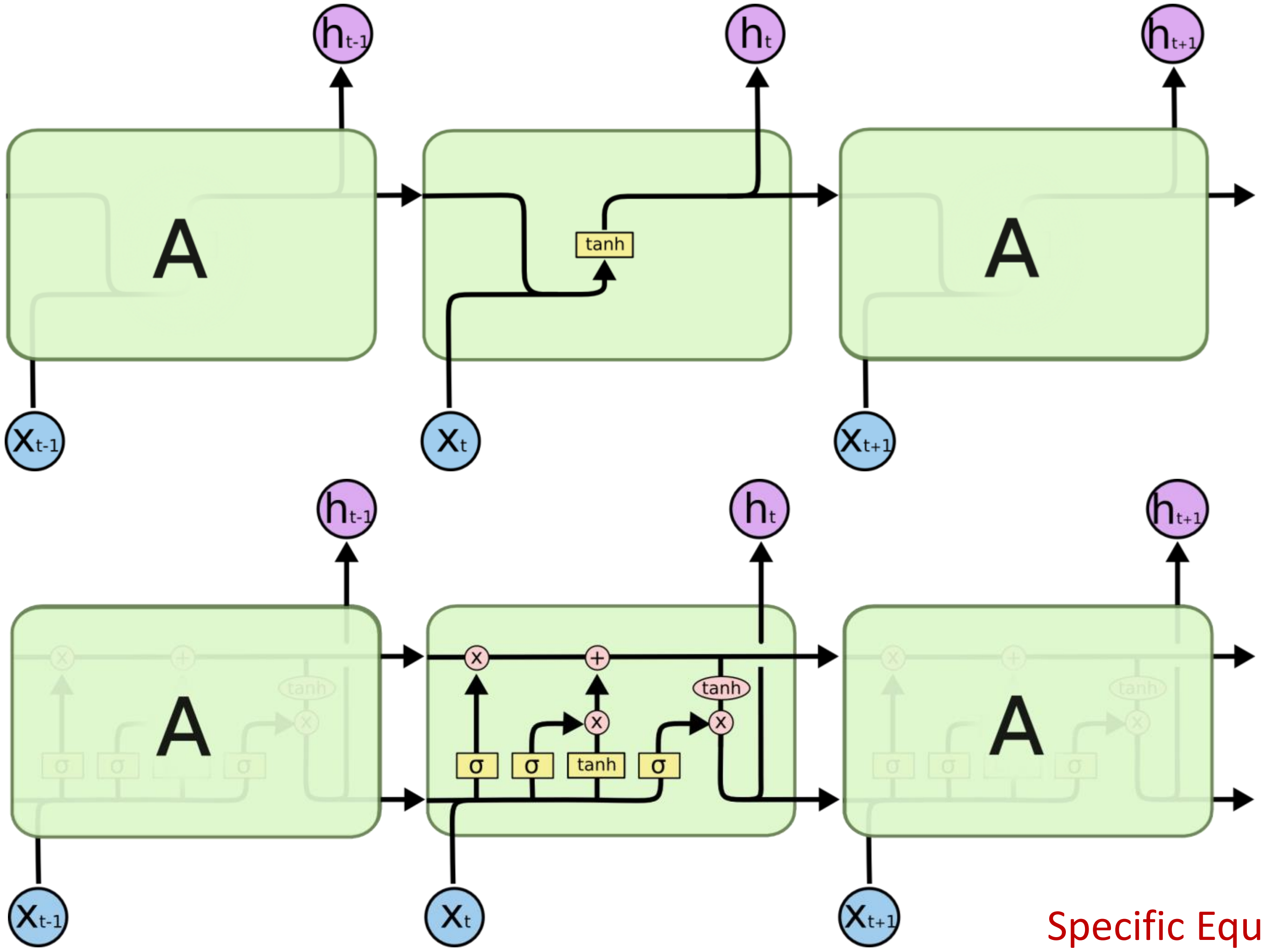


(a) RNN



(b) LSTM

RNN vs LSTM's

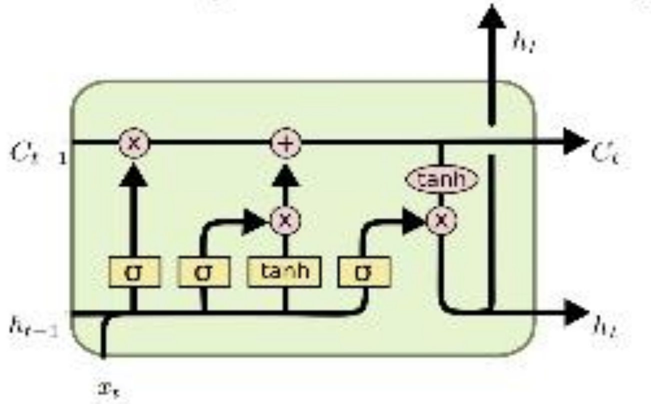


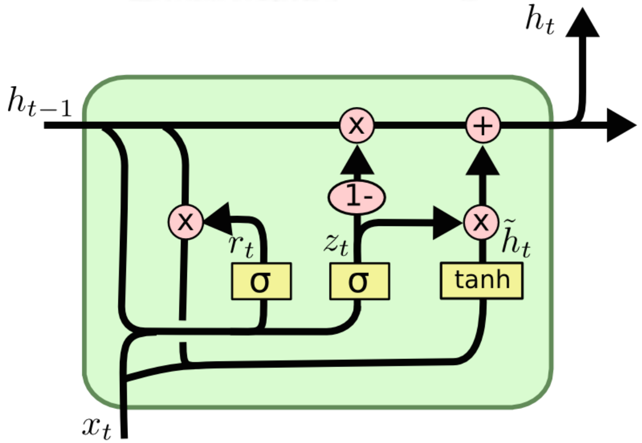
Specific Equations are Avoided

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM and GRU

- LSTM [Hochreiter&Schmidhuber97]





GRUs also takes x_t and h_{t-1} as inputs. They perform some calculations and then pass along h_t. What makes them different from LSTMs is that GRUs don't need the cell layer to pass values along. The calculations within each iteration ensure that the h_t values being passed along either retain a high amount of old information or are jump-started with a high amount of new information.

Specific Equations are Avoided

RNNs

- A very useful and powerful category of NNs
- With newer implementations (LSTM, GRU), it is ``practical’’
- Deep RNNs with stacked layers (but not as deep as CNNs)
- Less computer memory (compared to CNNs)
- Recurrence also leads to “delay’’
- Interpretation is not very simple. (unlike CNNs)

Thanks!!

Questions?