Recurrent Neural Networks

CS 6384 Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

Some slides of this lecture are courtesy Stanford CS231n

Single Images

- Convolutional neural networks

Image

CNN

High-level information
- Depth
- Object classes
- Object poses
- Etc.

Sequential Data

- Data depends on time
  - Video

t-1

t

t+1

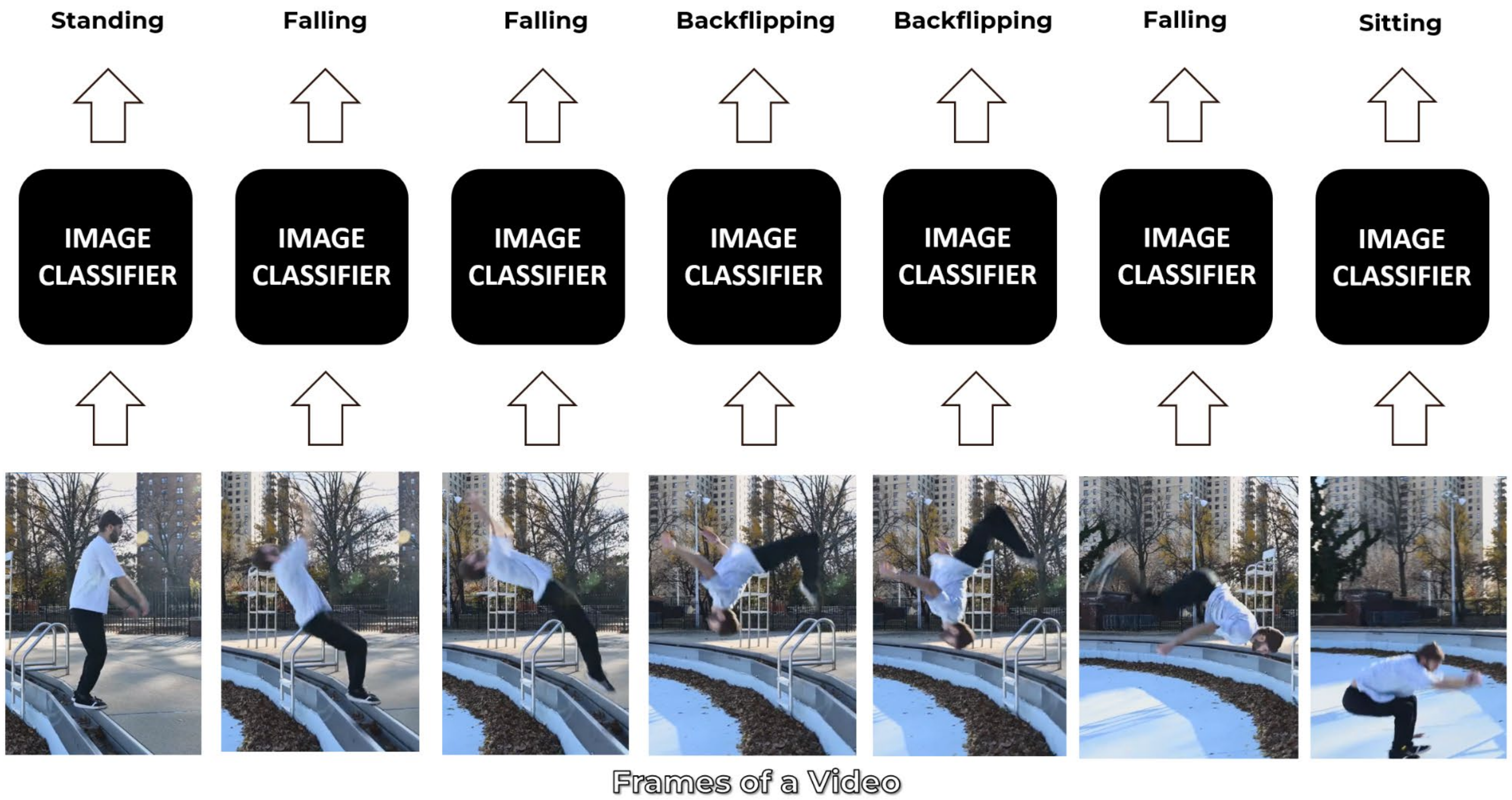
  - Sentence

UT Dallas is a rising public research university in the heart of DFW.

t

Sequential Data Labeling

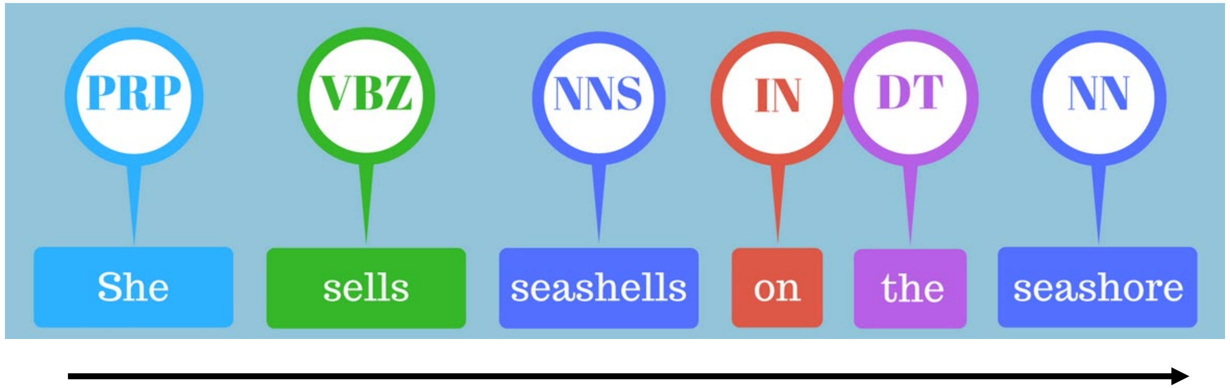
- Video frame labeling



https://bleedai.com/human-activity-recognition-using-tensorflow-cnn-lstm/

Sequential Data Labeling

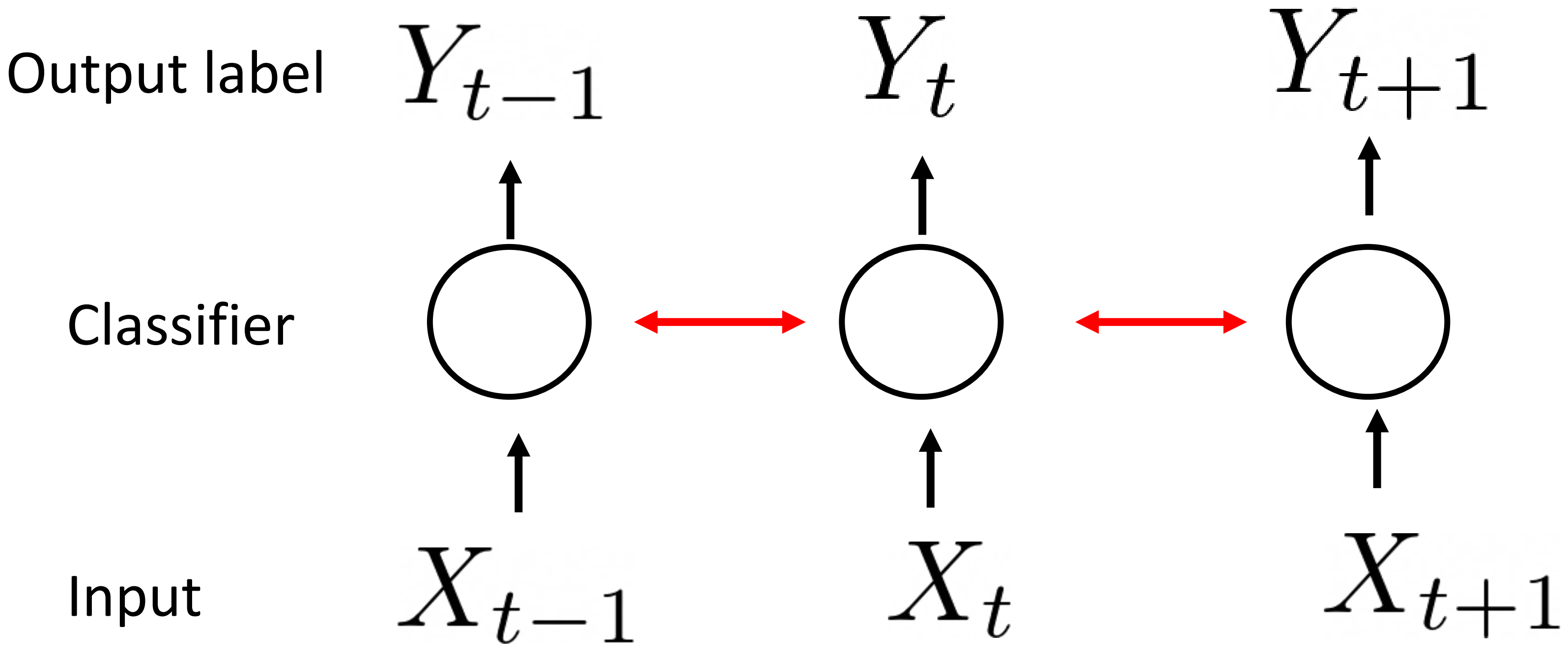
- Part-of-speech tagging (grammatical tagging)



t

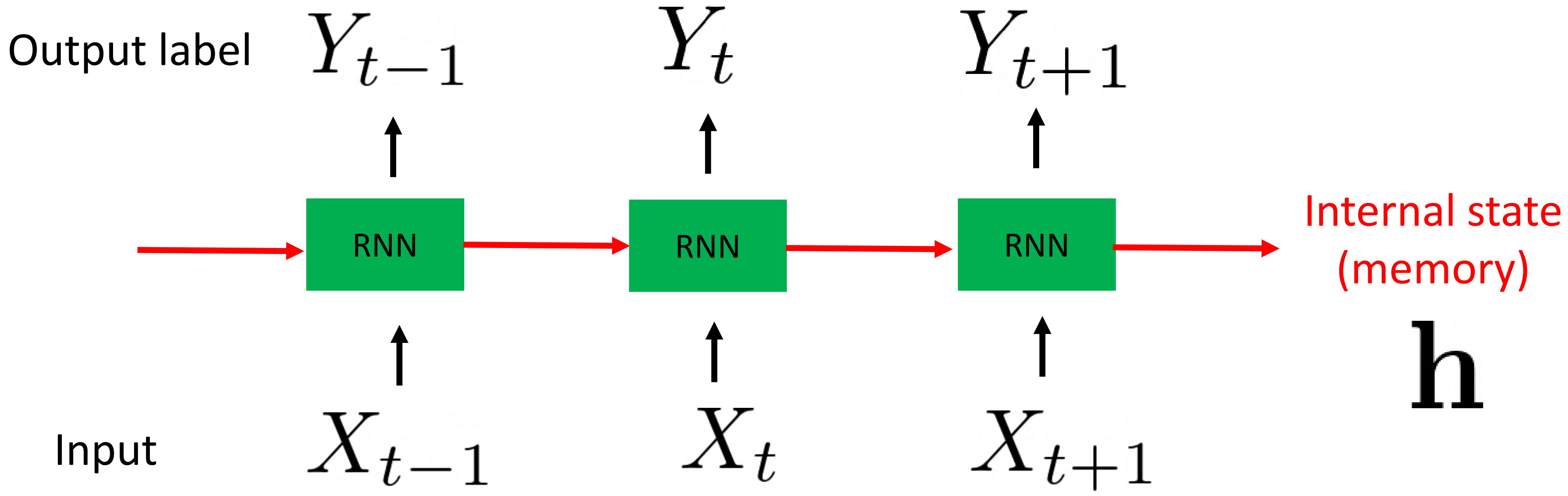
Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

Sequential Data Labeling

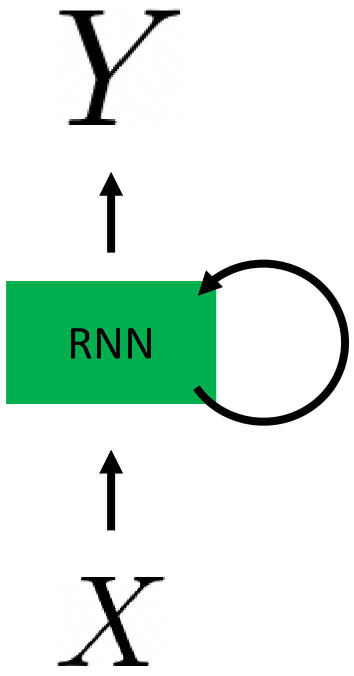


How to capture information across time?

Recurrent Neural Networks



Hidden State Update



Updating function with parameters W

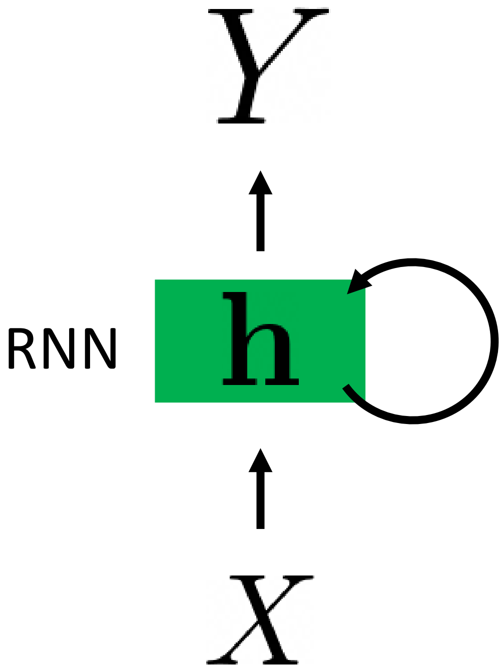
$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

Hidden state at time t

Hidden state at time t-1

Input at time t

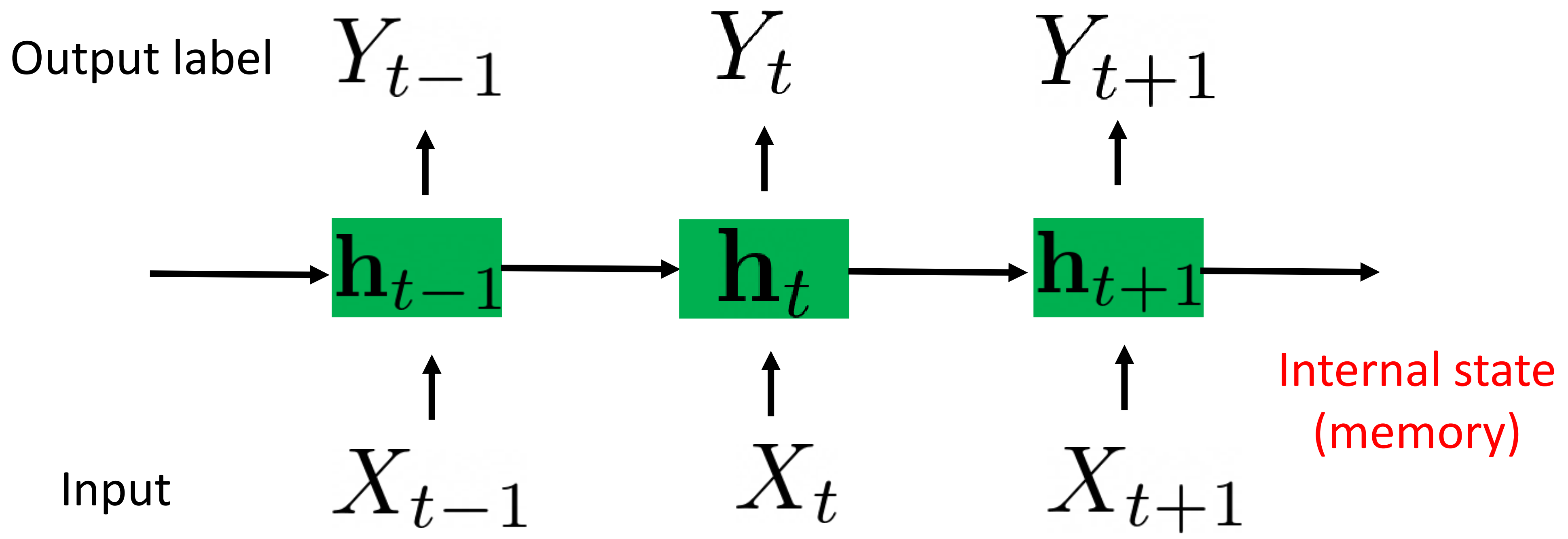
Using the Hidden State



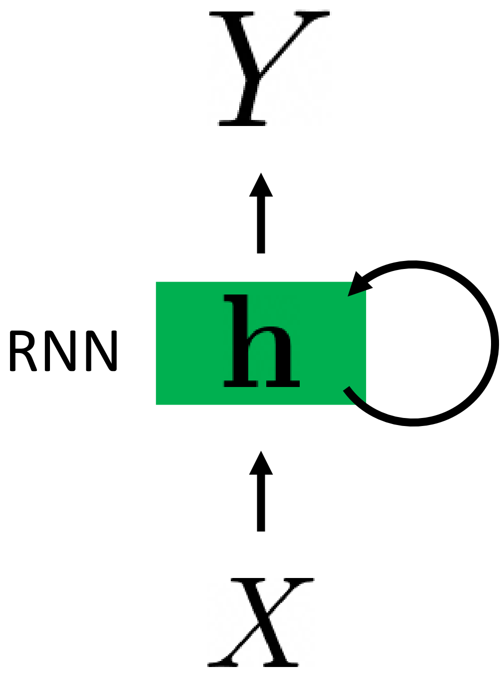
$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{y}_t = f_{W'}(\mathbf{h}_t)$$

Recurrent Neural Networks



Vanilla RNN



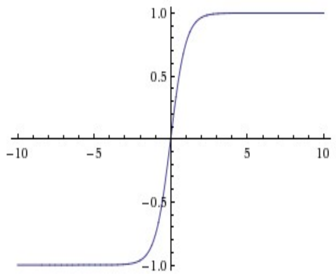
Hidden state updating rule

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$$

$m \times 1$ $\quad m \times m$ $\quad m \times 1$ $\quad m \times n$ $\quad n \times 1$

tanh tanh(x)

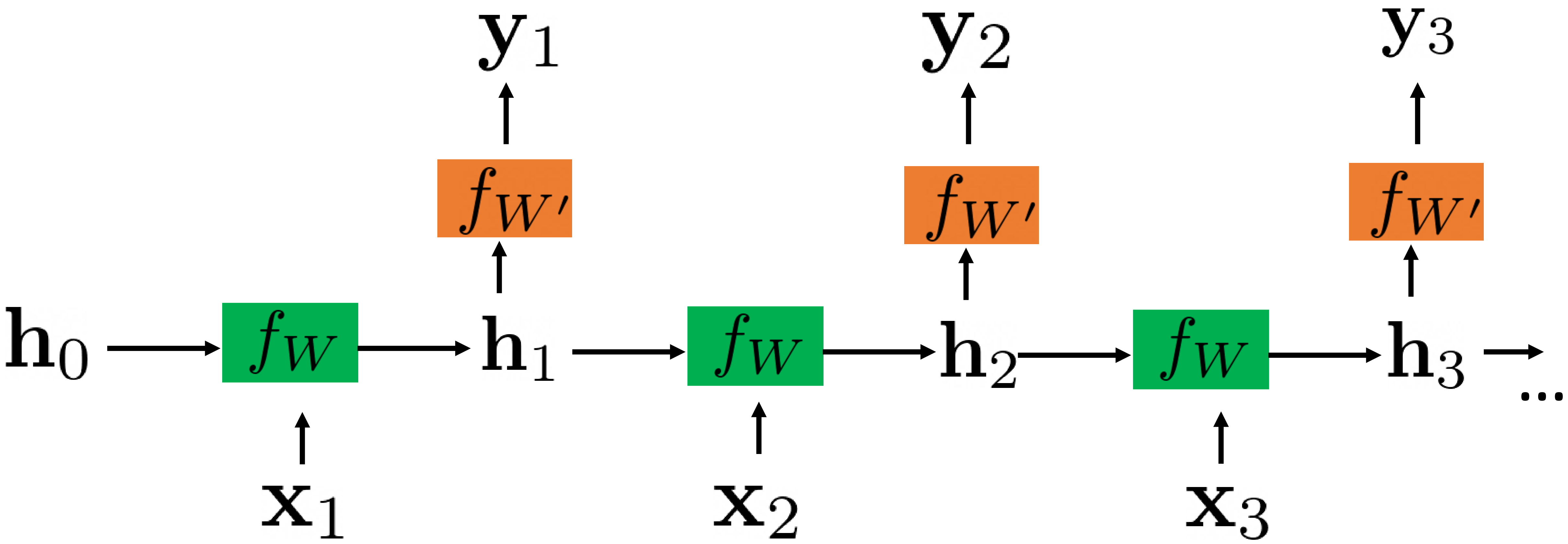
$$\frac{e^{2x} - 1}{e^{2x} + 1}$$



$$\mathbf{y}_t = W_{hy}\mathbf{h}_t$$

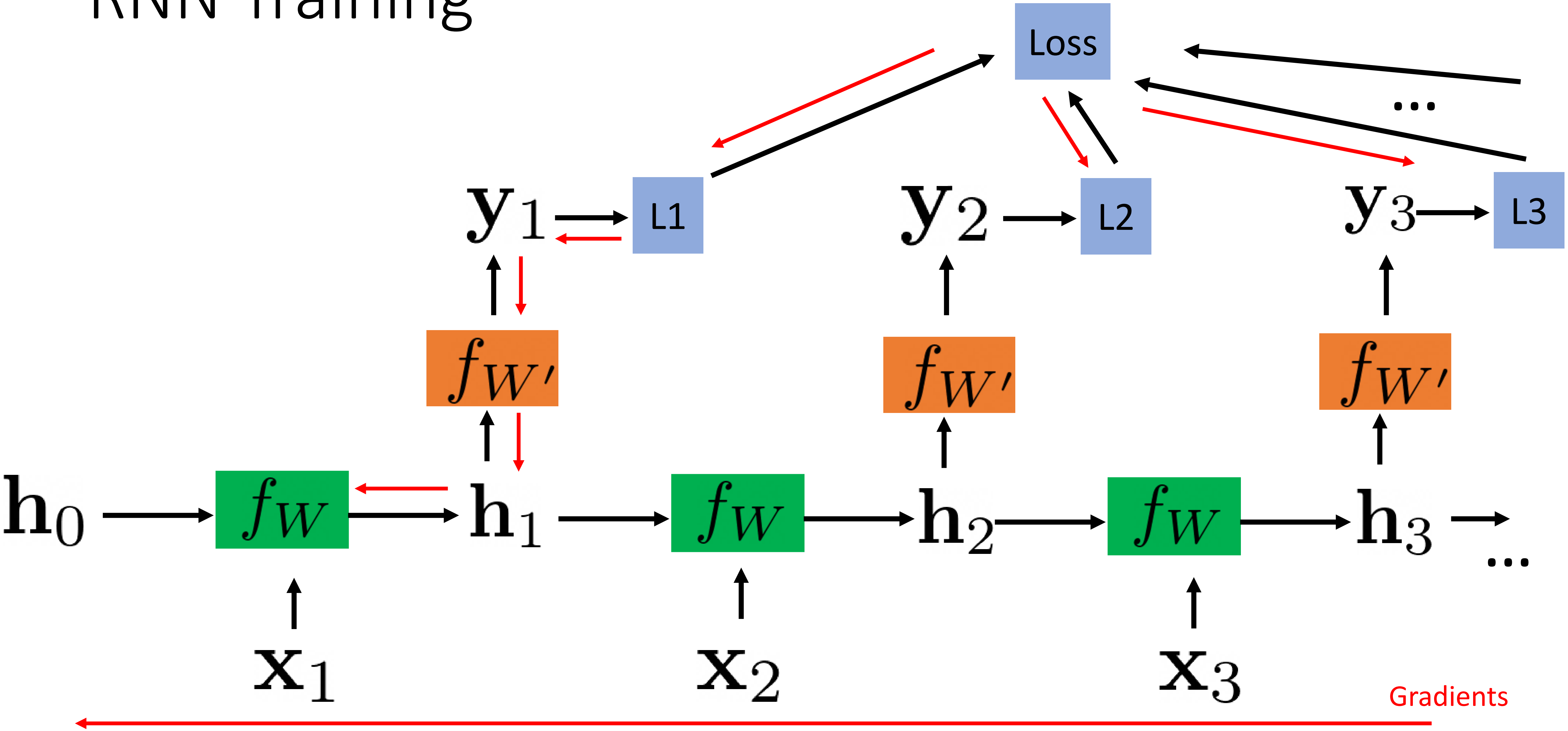
$l \times 1$ $\quad m \times 1$

RNN Computation Graph

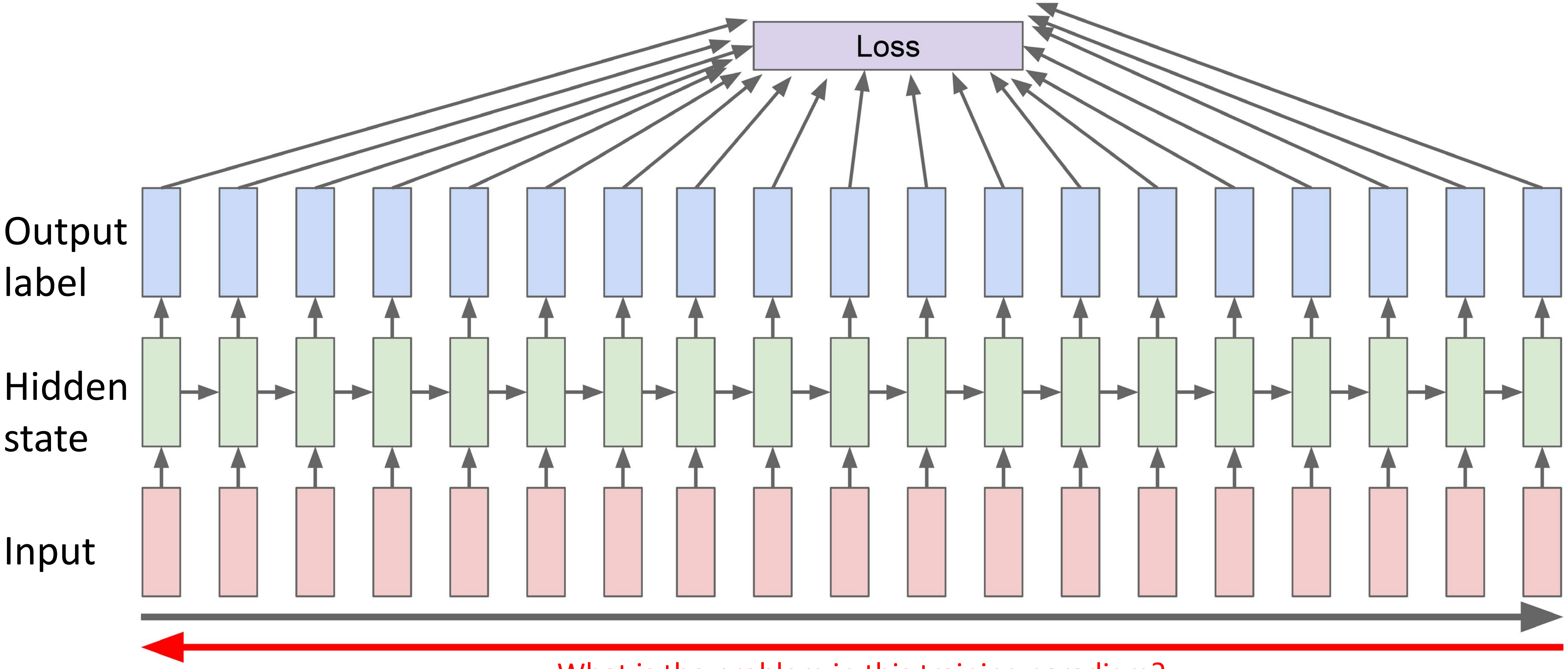


The same set of weights for different time steps f_W $f_{W'}$

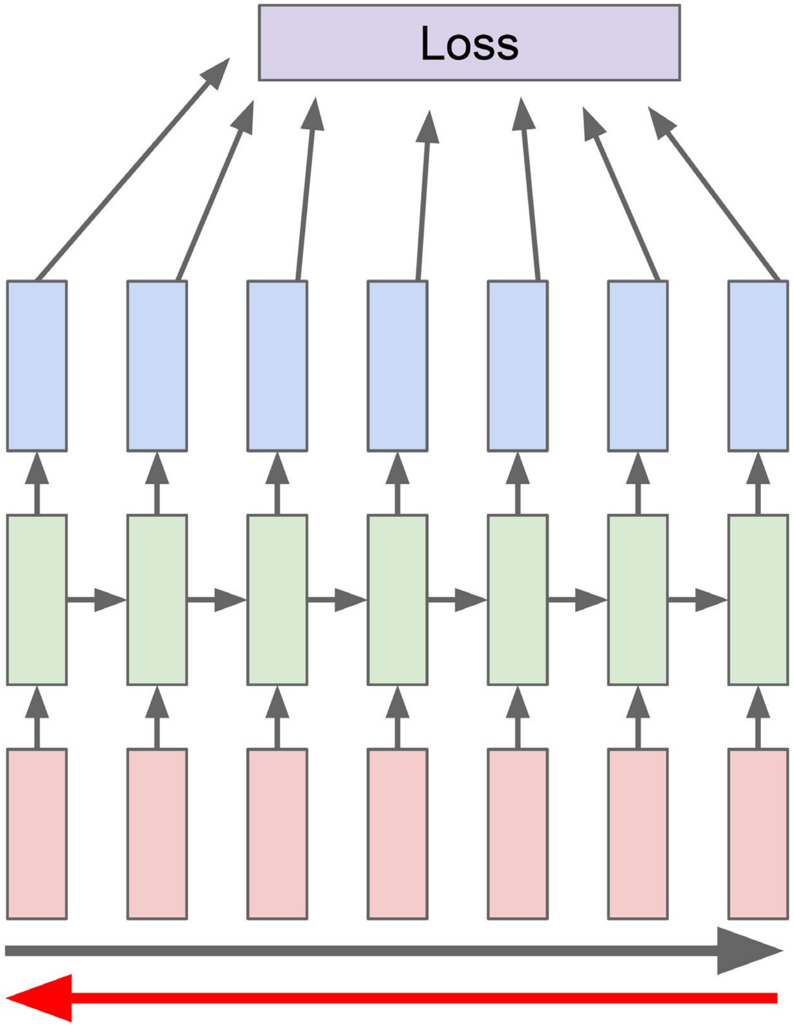
RNN Training



Backpropagation through Time

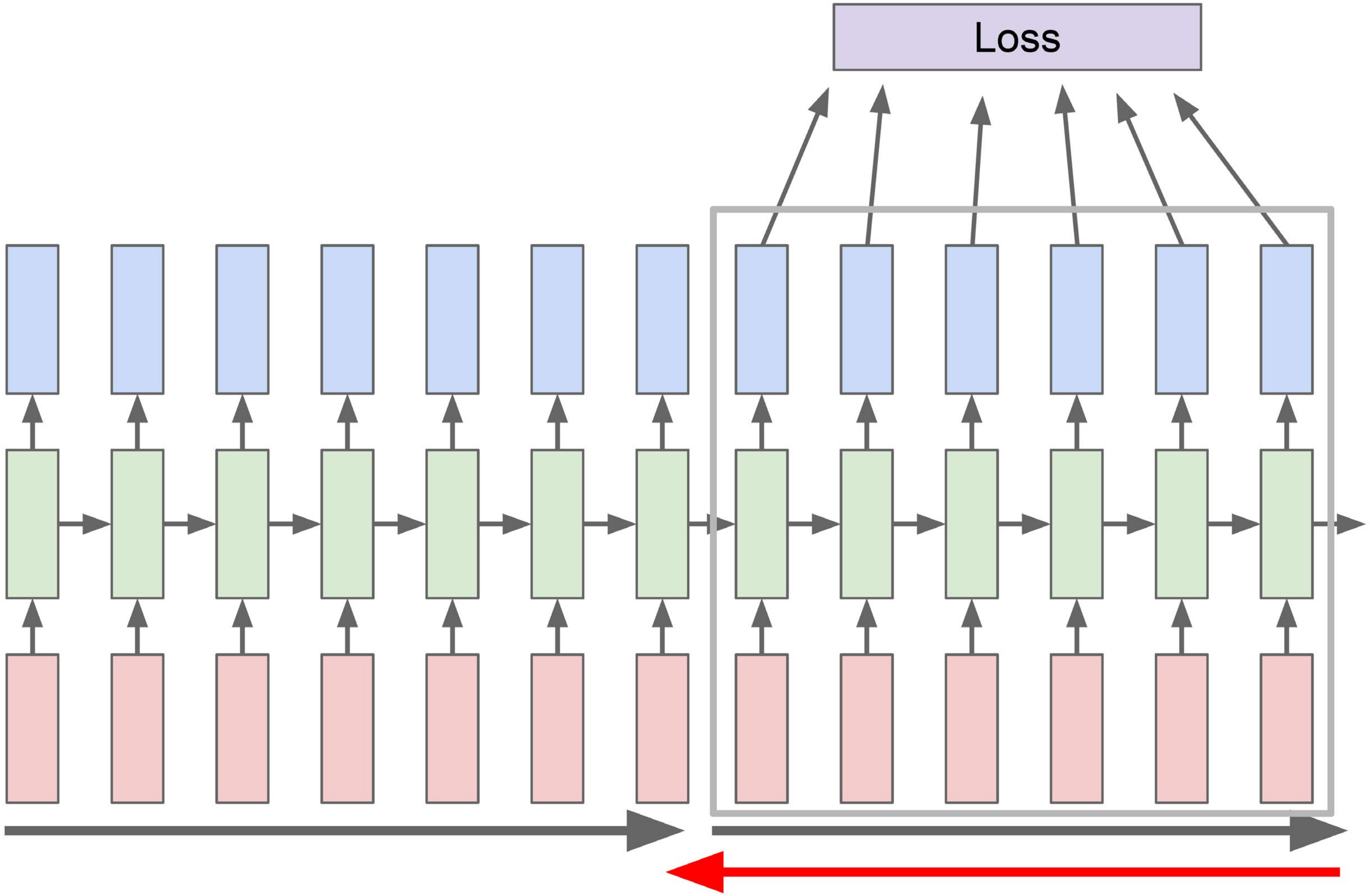


Truncated Backpropagation through Time



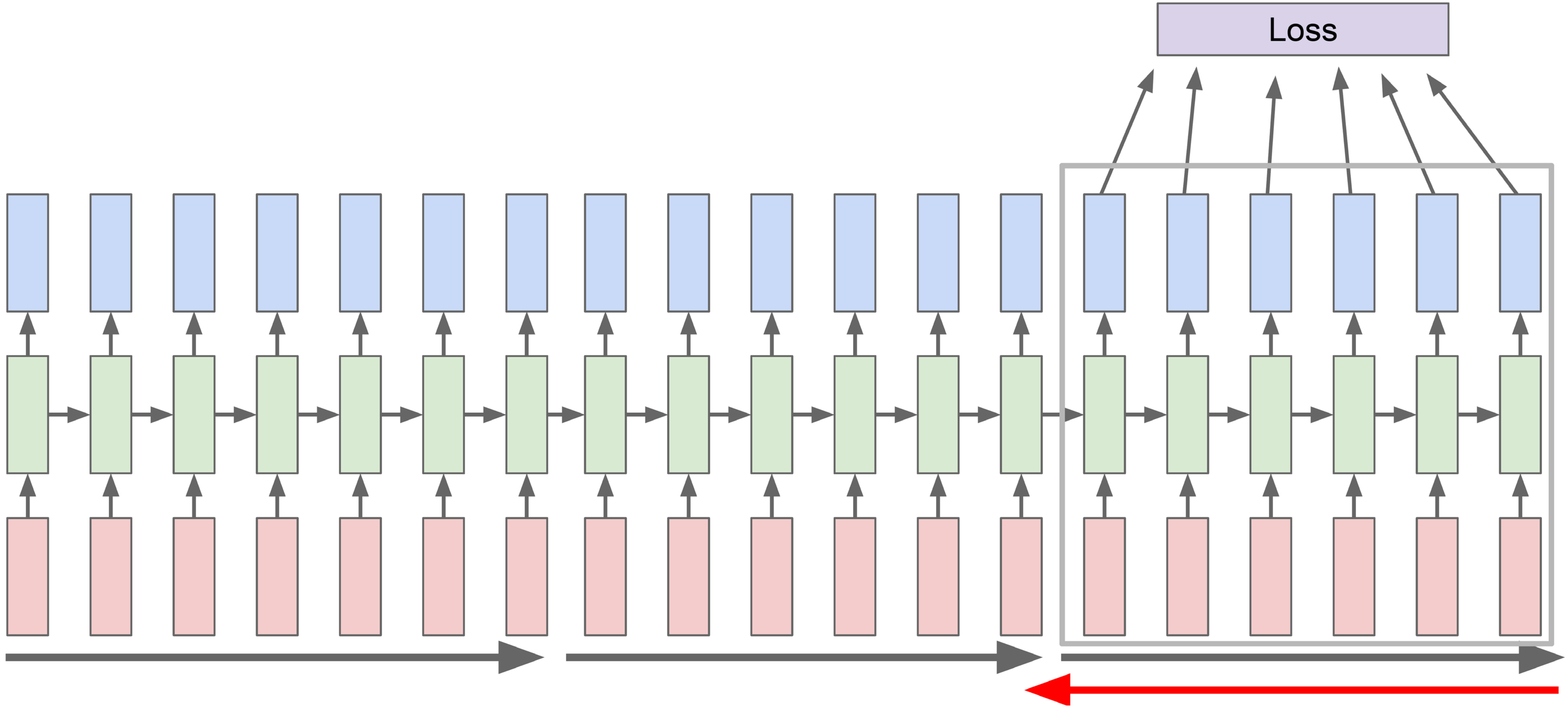
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through Time

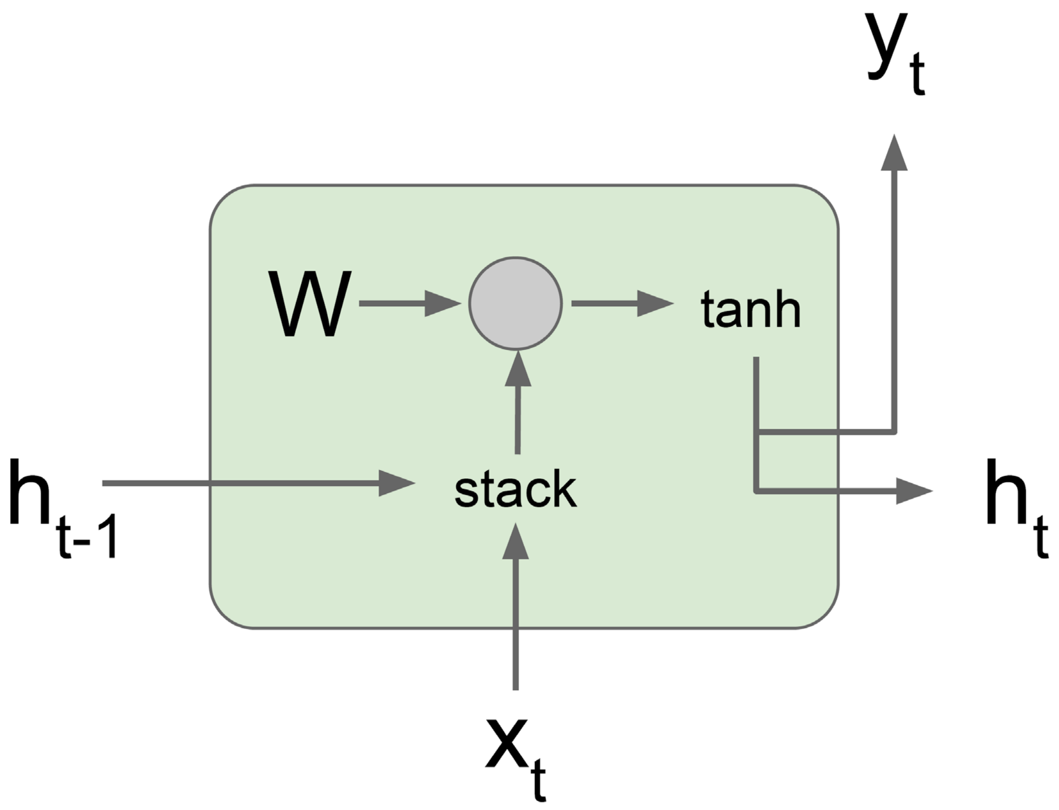


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through Time



Vanilla RNN Gradient Flow

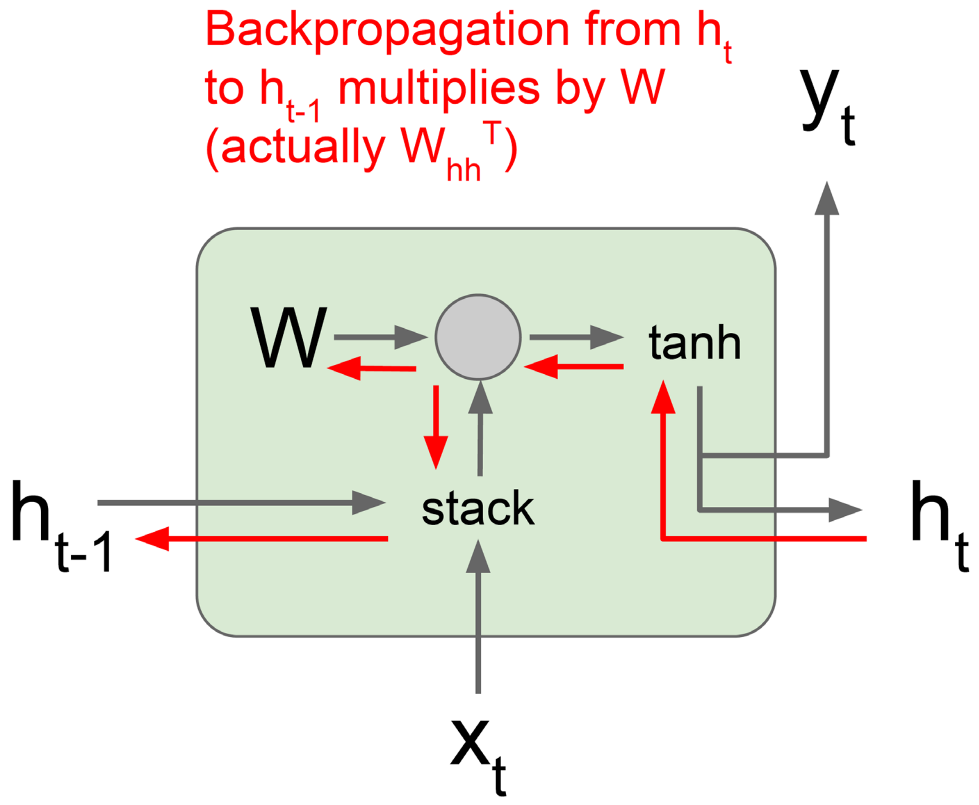


$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$$

$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

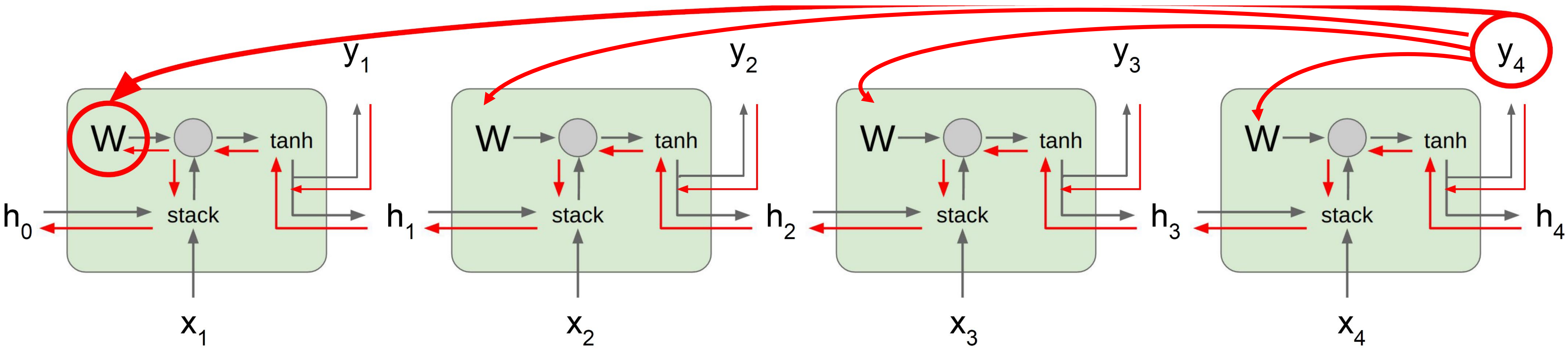
Vanilla RNN Gradient Flow



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}
$$

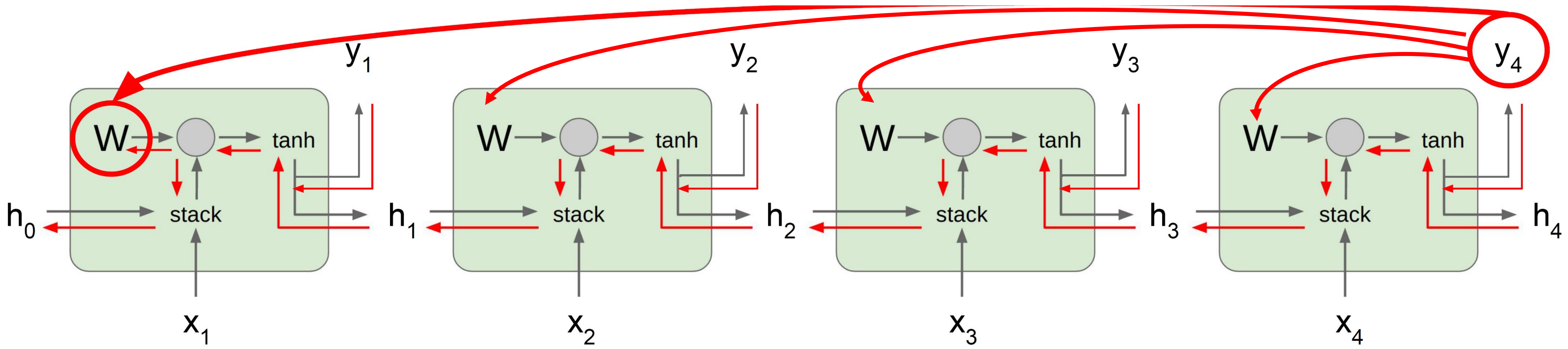
Vanilla RNN Gradient Flow



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow



$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

- Vanishing gradients $\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\|_2 < 1$
- Exploding gradients $\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\|_2 > 1$

https://en.wikipedia.org/wiki/Matrix_norm

Vanilla RNN Gradient Flow

- Exploding gradients $\|\frac{\partial h_t}{\partial h_{t-1}}\|_2 > 1$
  - Gradient clipping

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

- Vanishing gradients $\|\frac{\partial h_t}{\partial h_{t-1}}\|_2 < 1$
  - Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

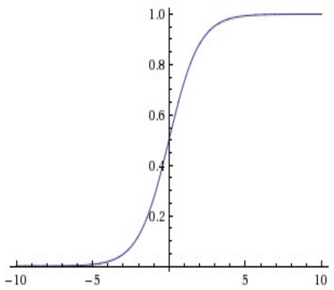
$$h_t = \tanh\left(W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}\right)$$

LSTM

$$\begin{array}{l} \text{Input gate} \\ \text{forget gate} \\ \text{output gate} \\ \text{gate gate} \end{array} \begin{pmatrix}i\\f\\o\\g\end{pmatrix} = \begin{pmatrix}\sigma\\\sigma\\\sigma\\\tanh\end{pmatrix} W \begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}$$

Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

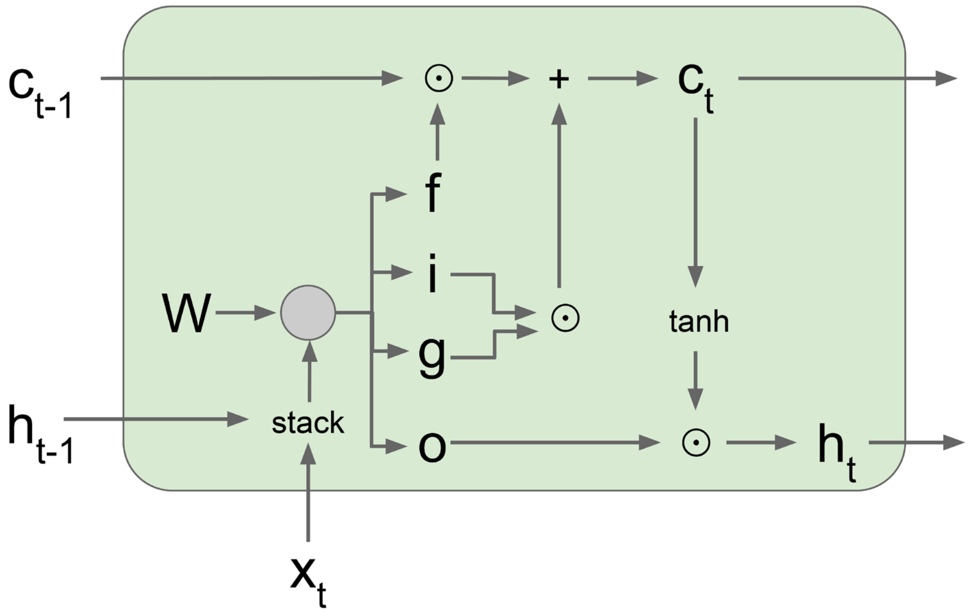


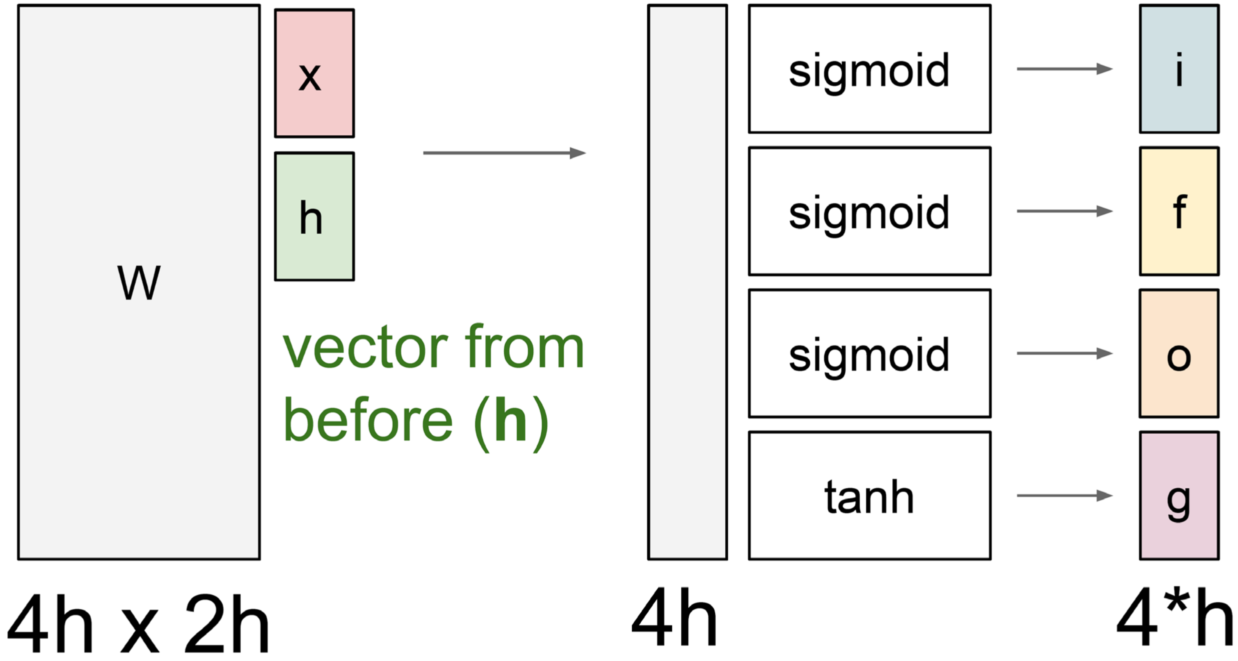
Cell $c_t = f \odot c_{t-1} + i \odot g$

Hidden state $h_t = o \odot \tanh(c_t)$

Store Cell and hidden states

Long Short Term Memory (LSTM)





$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

- **g**: Gate gate, how much to write to cell
- **i**: Input gate, whether to write to cell
- **f**: Forget gate, whether to erase cell
- **o**: Output gate, how much to reveal cell

Long Short Term Memory (LSTM)

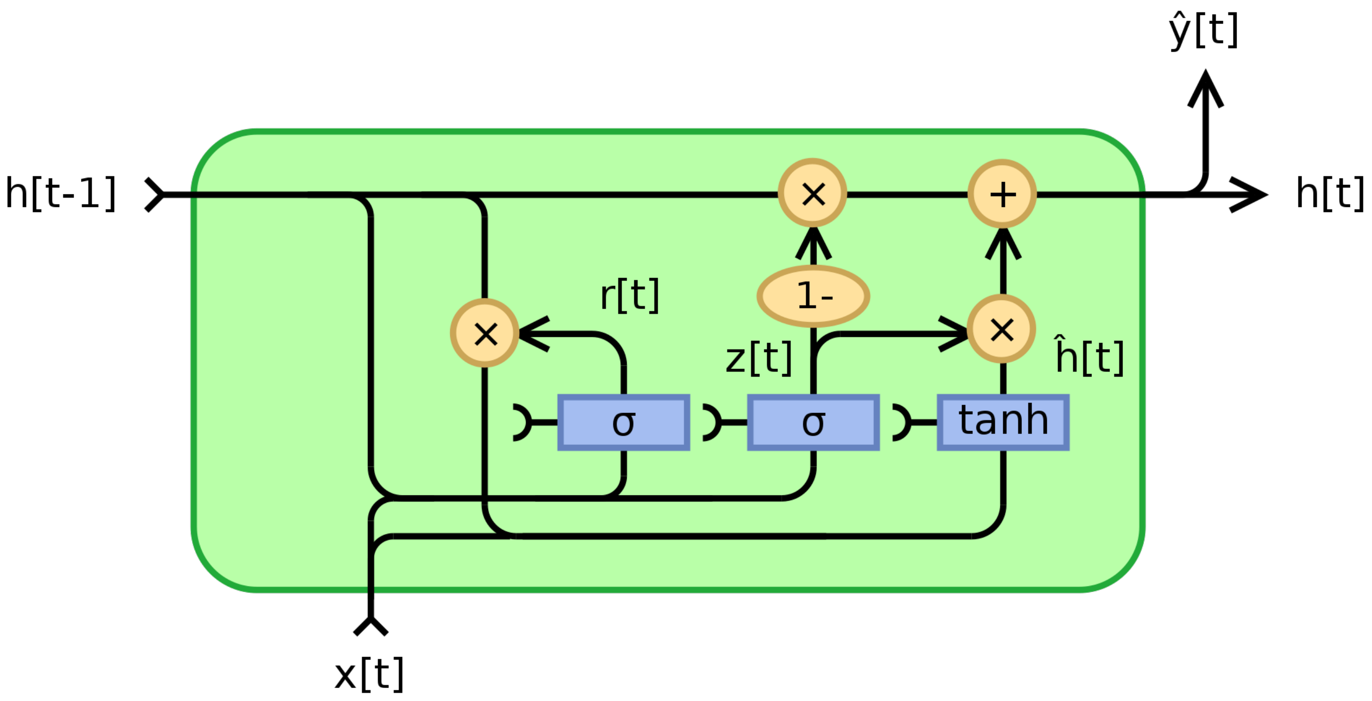
- Make the RNN easier to preserve information over many steps
  - E.g., f = 1 and i = 0
  - This is difficult for vanilla RNN
- LSTM does not guarantee that there is no vanishing or exploding gradient
- It provides an easier way to learn long-distance dependencies

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Gated Recurrent Unit (GRU)



https://en.wikipedia.org/wiki/Gated_recurrent_unit

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

- x_t: input vector
- h_t: output vector
- $\hat{h}_t$: candidate activation vector
- z_t: update gate vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

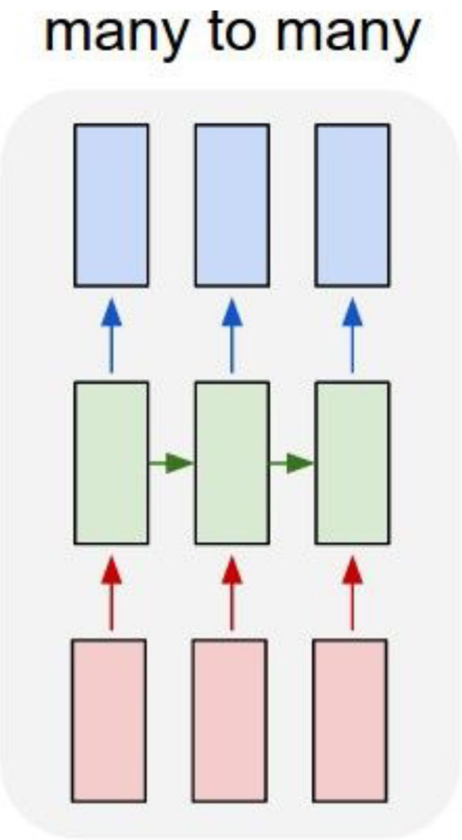
GRUs vs. LSTMs

- Both have a forget gate

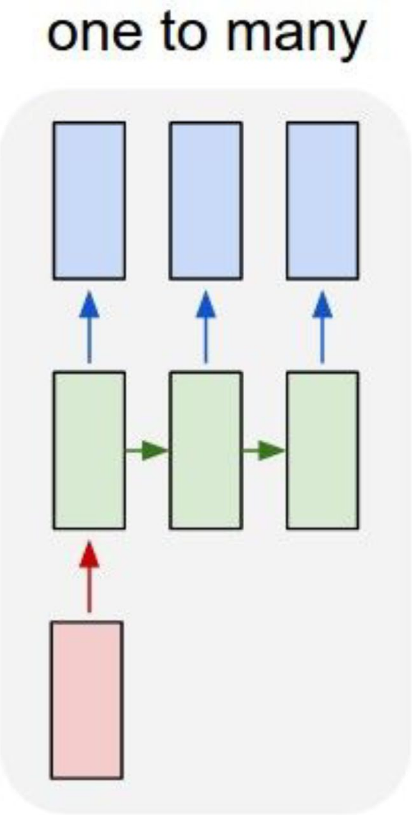
- GRU has fewer parameters, no output gate

- GRUs have similar performance compared to LSTMs, have shown better performance on certain datasets

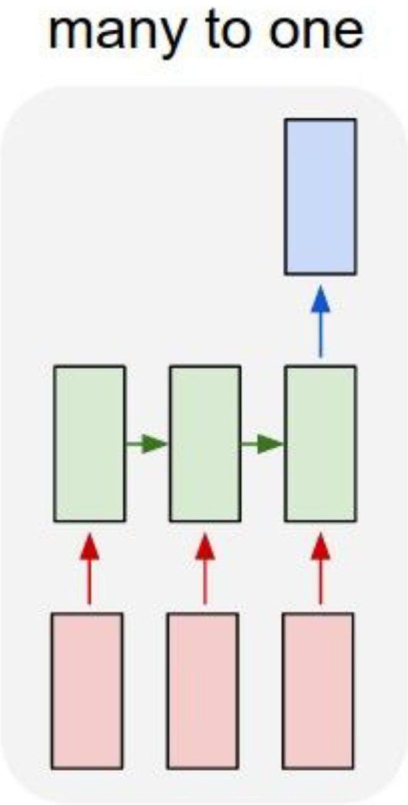
Recurrent Neural Networks



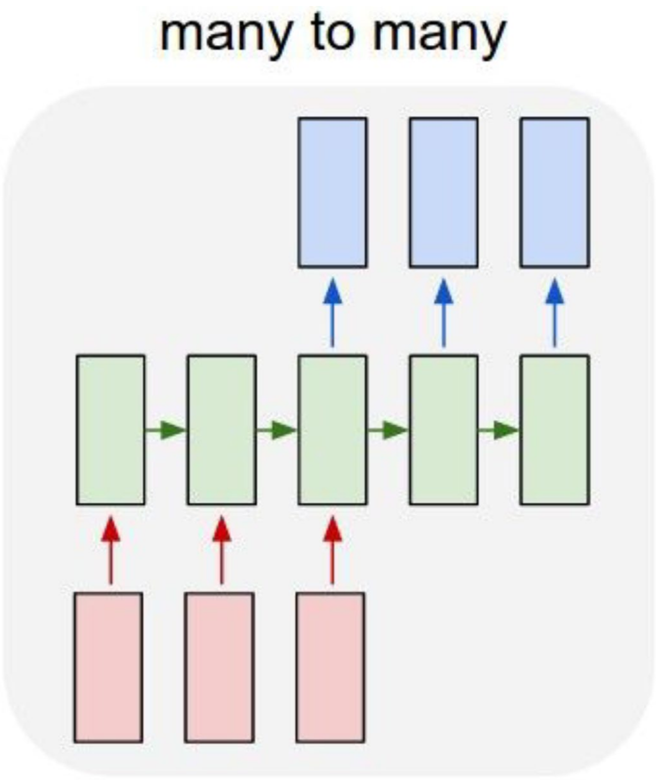
E.g., action recognition on video frames



E.g., image captioning, image -> sequences of words

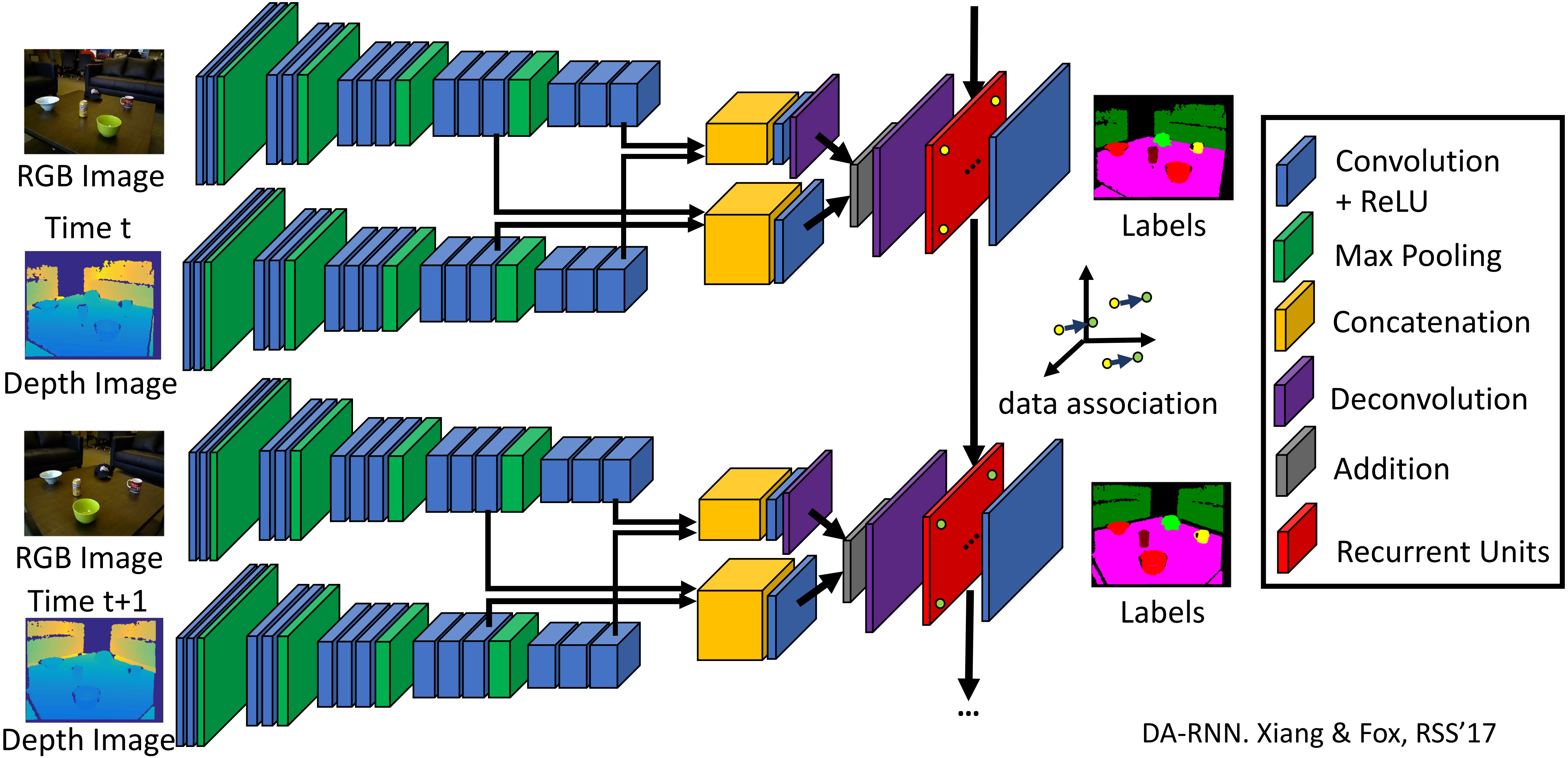


E.g., action prediction, sequences of frames -> action class



E.g., Video Captioning Sequence of video frames -> caption

Recurrent Units on CNN Features



DA-RNN. Xiang & Fox, RSS'17

Summary

- RNNs can be used for sequential data to capture dependencies in time

- LSTMs and GRUs are better then vanilla RNNs

- It is difficult to capture long-term dependencies in RNNs

- Use transformers (next lecture)

Further Reading

- Stanford CS231n, lecture 10, Recurrent Neural Networks
  http://cs231n.stanford.edu/
- Long Short Term Memory
  https://www.researchgate.net/publication/13853244_Long_Short-term_Memory
- Gated Recurrent Units https://arxiv.org/pdf/1412.3555.pdf