RNNs: Recurrent Neural Networks

RNNs

- So far, we have seen MLPs and CNNS
- CNNs are suitable for grid data
- RNNs are suitable for processing sequential data
- Central idea: parameter sharing
  - If separate parameters for different time indices:
    - Cannot generalize to sequence lengths not seen during training
  - So, parameters are shared across several time steps
- Major difference from MLP and CNNs: RNNs have **cycles**

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

h: hidden state
theta: (shared) parameters
t: time
x: input

The network maps the whole input $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ to $h^{(t)}$. *e.g.,*

$$h^{(3)} = f(f(f(h^{(0)}, x^{(1)}; \theta), x^{(2)}; \theta), x^{(3)}; \theta)$$

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

h: hidden state
theta: (shared) parameters

t: time
x: input

The network maps the whole input $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$

to $h^{(t)}$. *e.g.*,

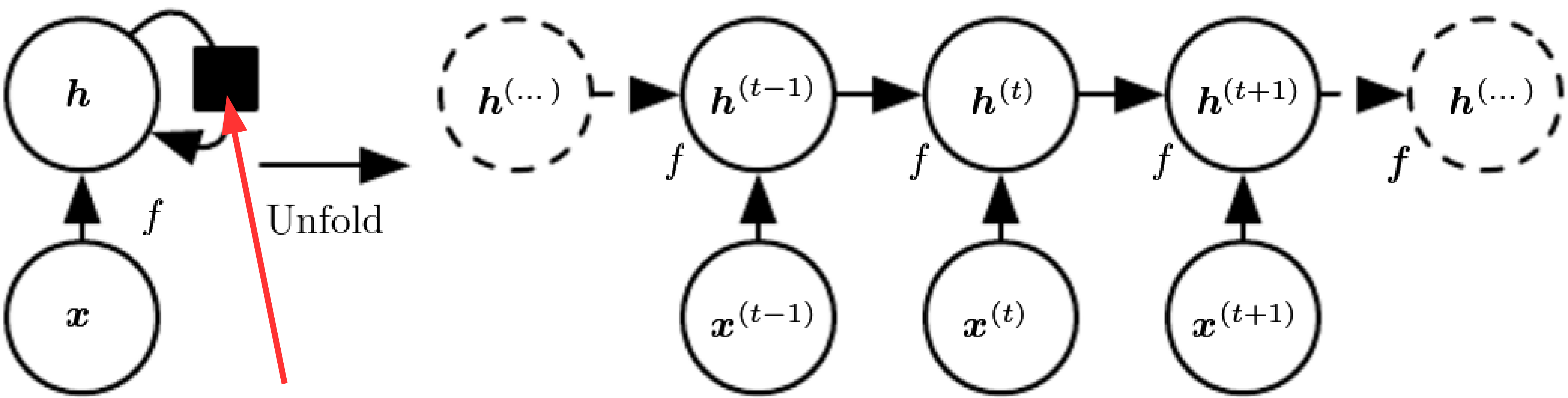
$$h^{(3)} = f(f(f(h^{(0)}, x^{(1)}; \theta), x^{(2)}; \theta), x^{(3)}; \theta)$$

The whole input, $x^{(1)}$ to $x^{(t)}$, is of arbitrary length but $h^{(t)}$ is fixed length. So, $\boldsymbol{h^{(t)}}$ **is a lossy summary of** the task-relevant aspects of $\boldsymbol{x^{(1)}}$ **to** $\boldsymbol{x^{(t)}}$.

Folded representation and unfolding

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

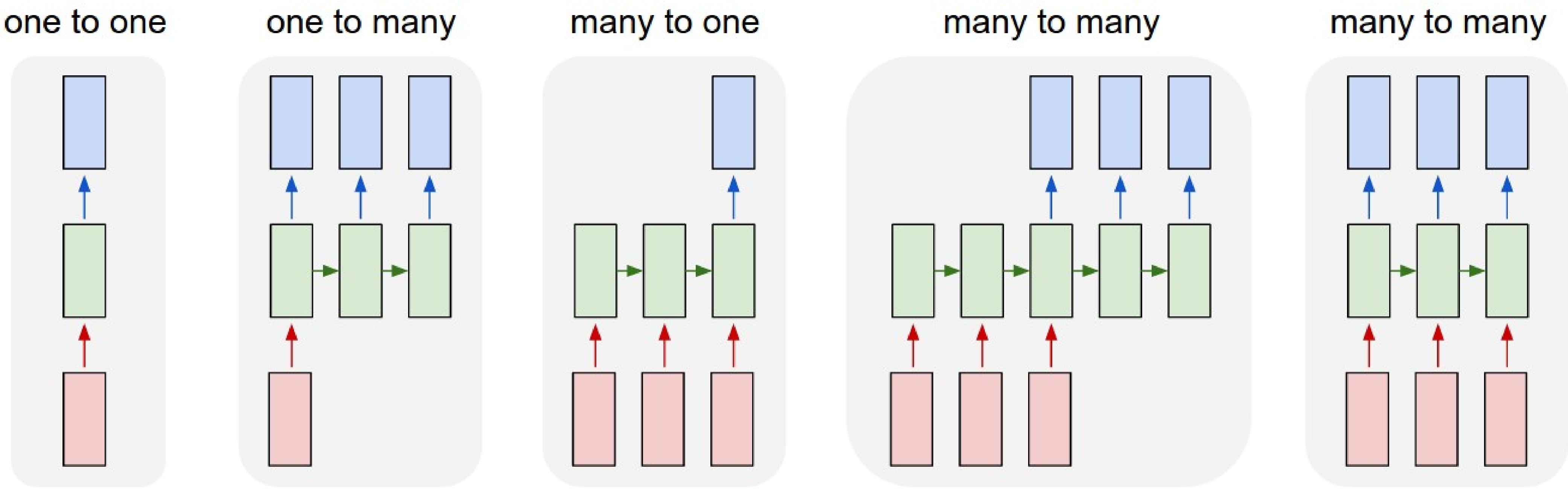
Two different ways of drawing above equation:



This means a single time-step

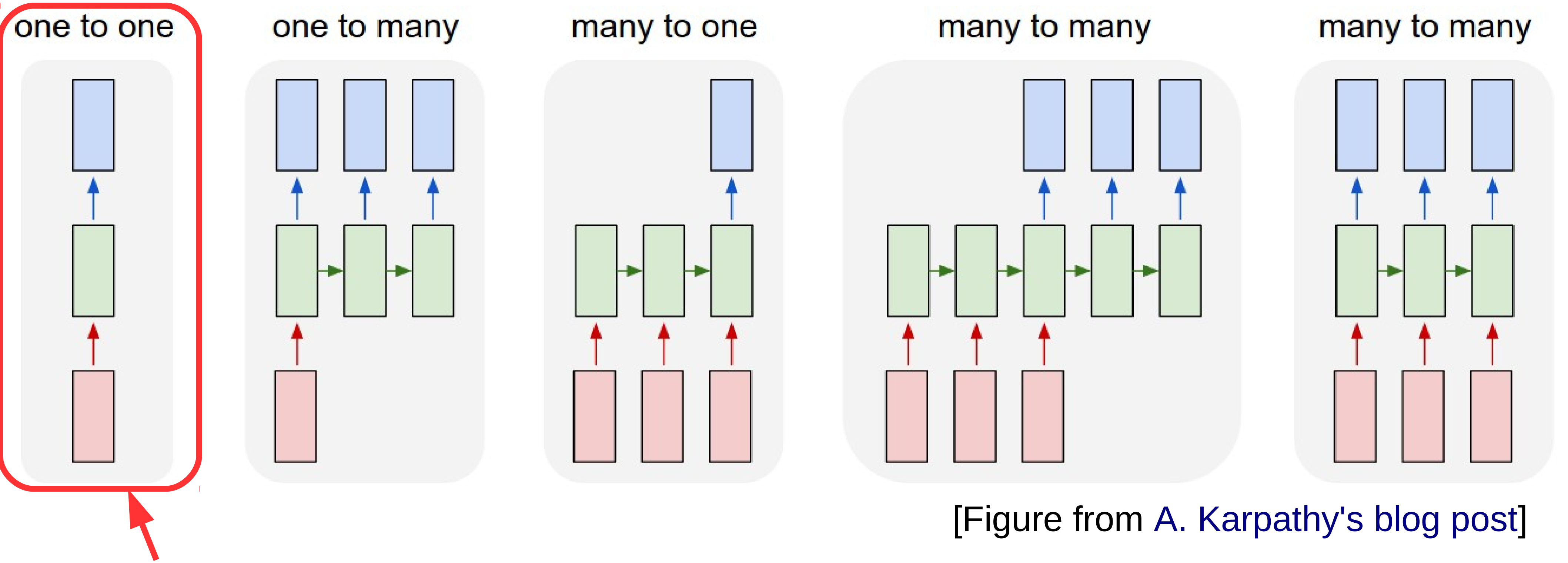
[Fig. 10.2 from Goodfellow et al. (2016)]

A variety of architectures are possible



[Figure from A. Karpathy's blog post]

A variety of architectures are possible

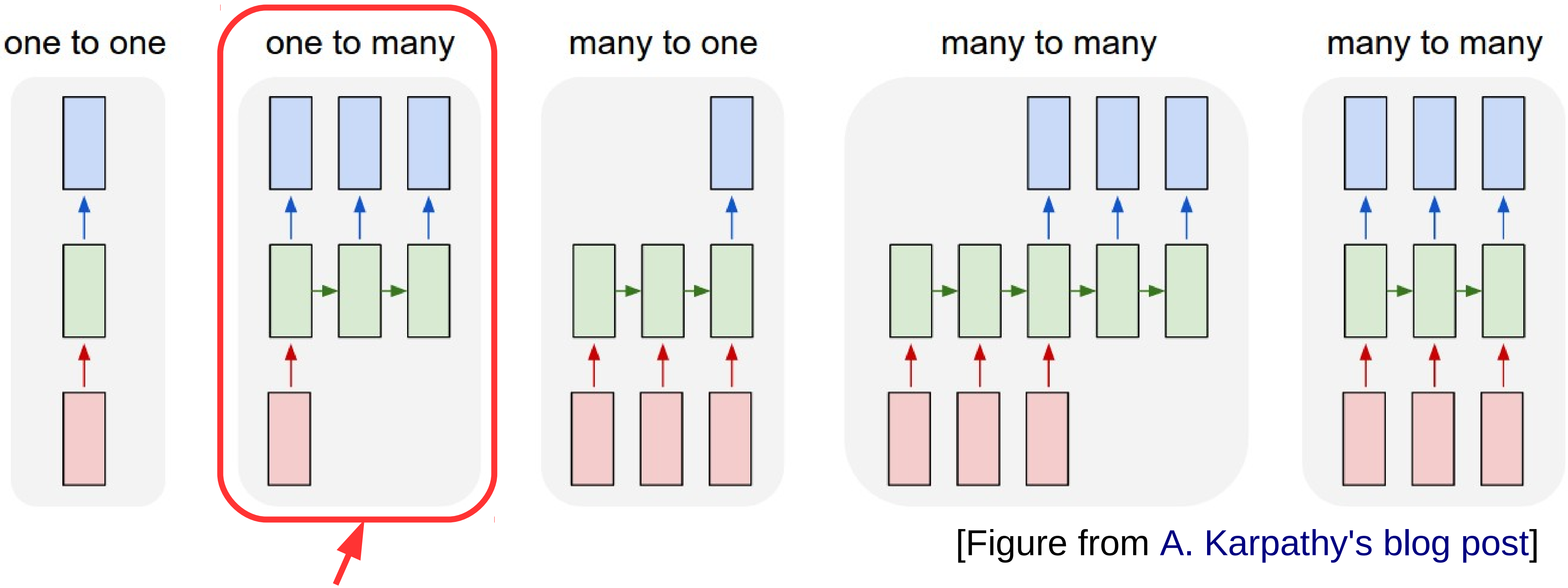


[Figure from A. Karpathy's blog post]

Traditional machine learning: fixed sized input vector in, fixed sized prediction vector out.

e.g. image classification

A variety of architectures are possible

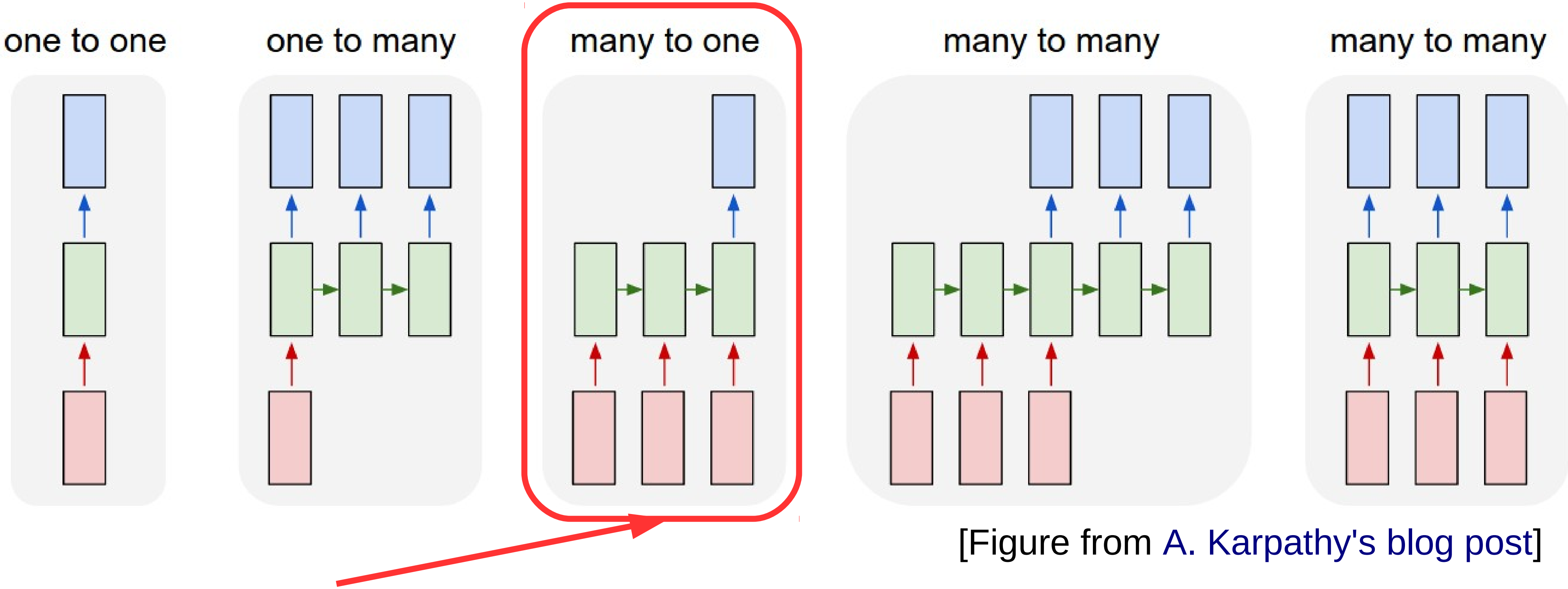


[Figure from A. Karpathy's blog post]

Sequence output

e.g. image captioning (image in, a descriptive sentence out)

A variety of architectures are possible

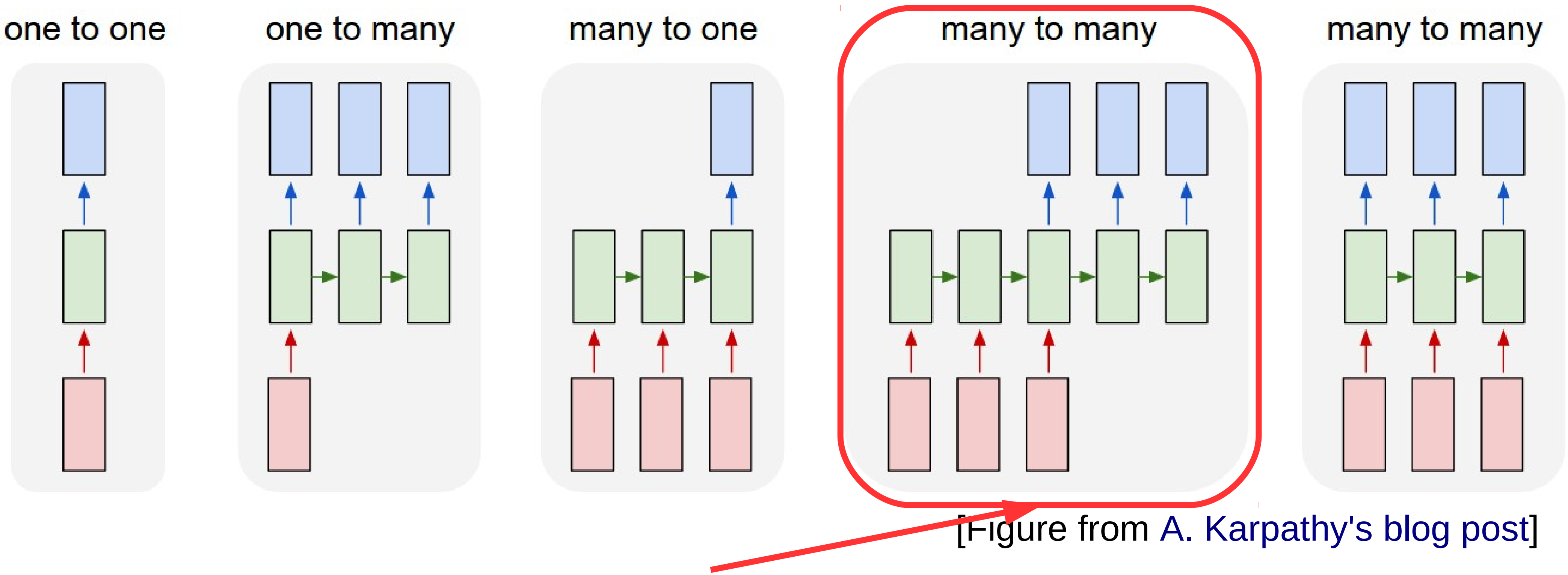


[Figure from A. Karpathy's blog post]

Sequence input

e.g. sentiment analysis (sentence in, sentiment label out)

A variety of architectures are possible

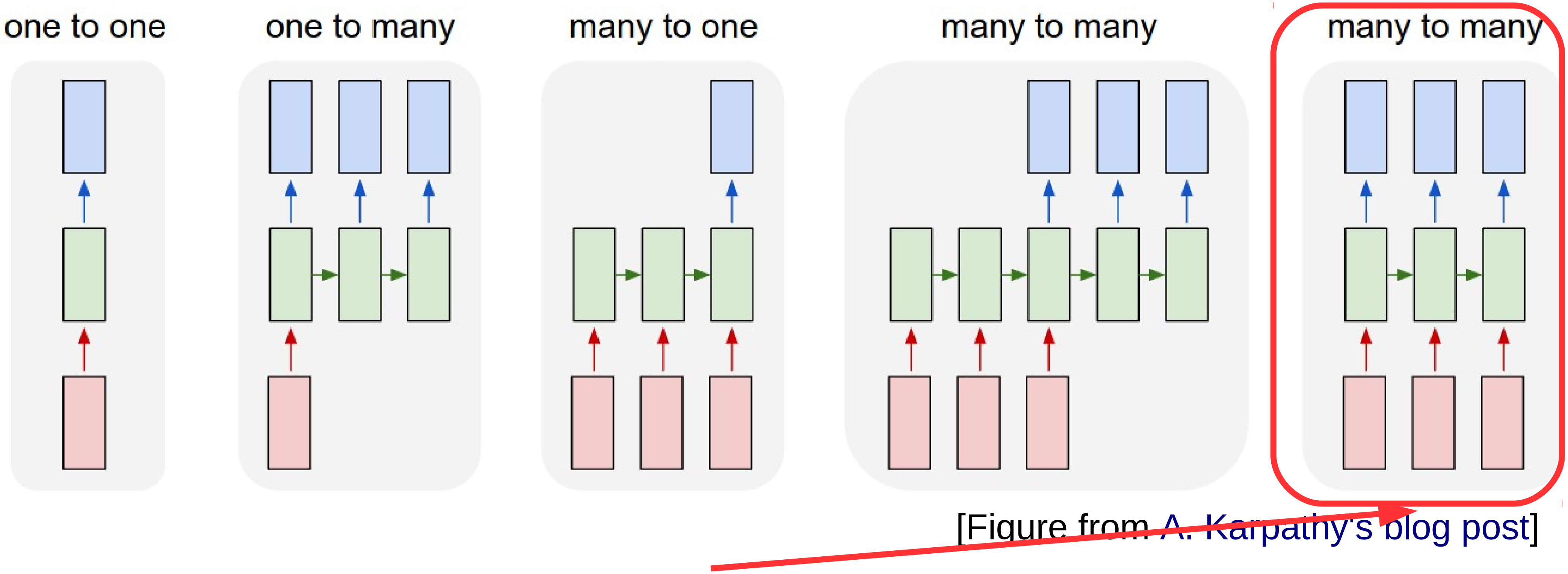


[Figure from A. Karpathy's blog post]

Sequence input, sequence output

e.g. machine translation (Turkish sentence in, English sentence out)

A variety of architectures are possible

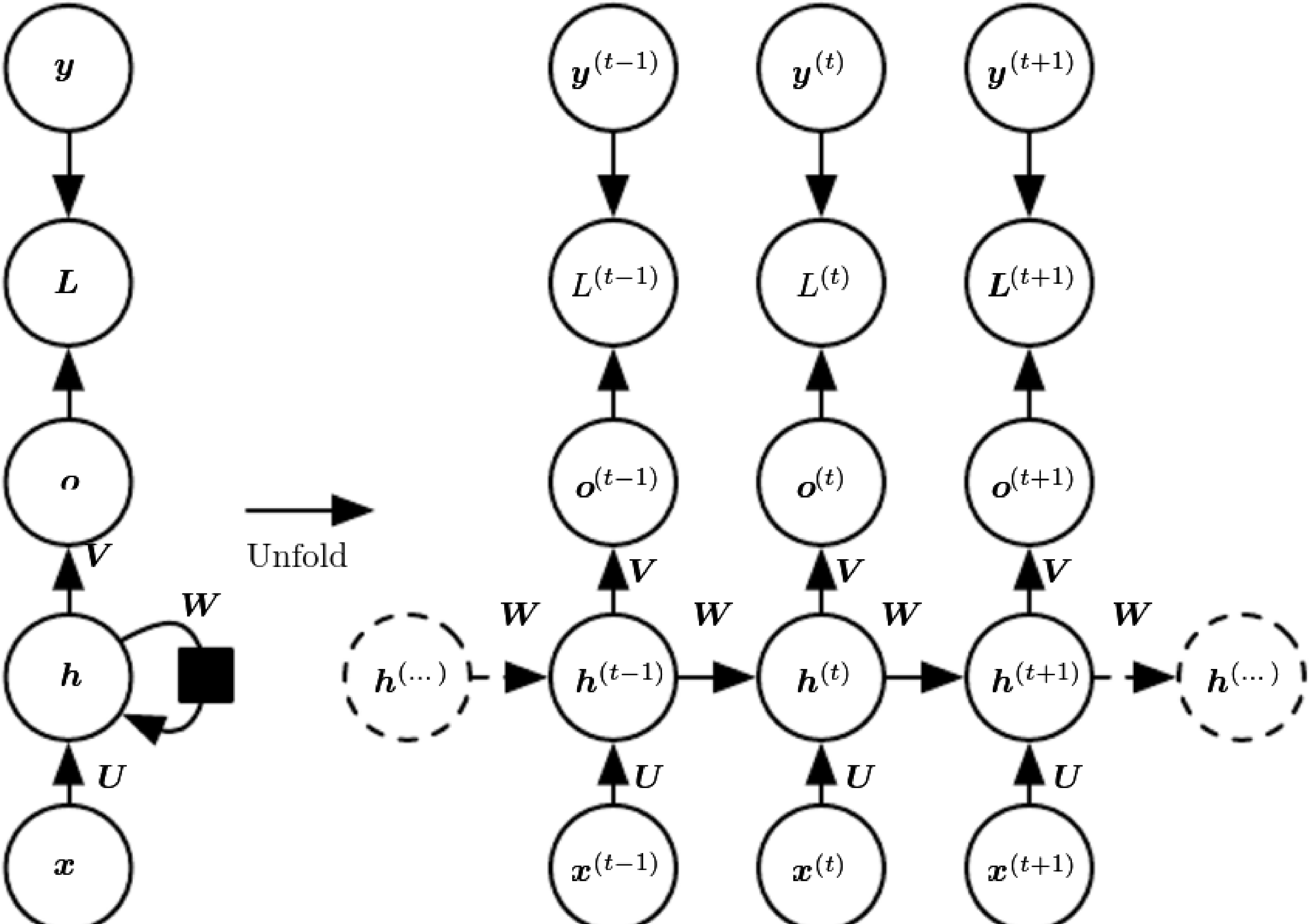


[Figure from A. Karpathy's blog post]

Sequence input, sequence output (there is an output per input)

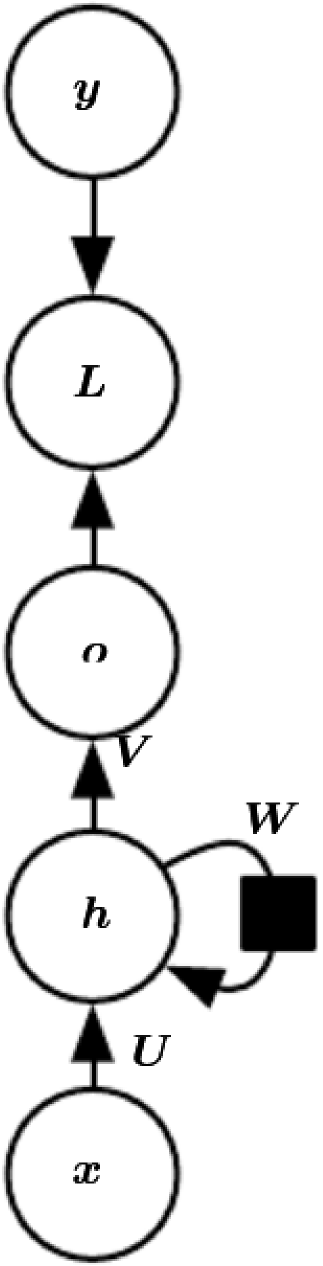
e.g. video frame classification

A recurrent network that maps input sequence ***x*** to output sequence ***o***, using a loss function *L* and label sequence ***y***.



[Fig. 10.3 from Goodfellow et al. (2016)]

A recurrent network that maps input sequence $\boldsymbol{x}$ to output sequence $\boldsymbol{o}$, using a loss function L and label sequence $\boldsymbol{y}$.



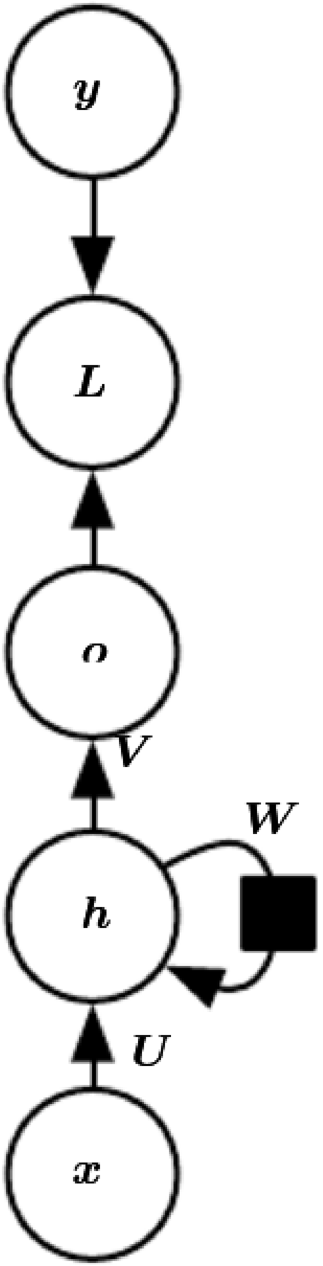
Update equations:

$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}$$

Note: *tanh( ), softmax( )* are just example choices.

[Fig. 10.3 from Goodfellow et al. (2016)]

A recurrent network that maps input sequence ***x*** to output sequence ***o***, using a loss function *L* and label sequence ***y***.



Total loss:

$$L\left(\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}\},\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(\tau)}\}\right)$$
$$=\sum_t L^{(t)}$$
$$=-\sum_t \log p_{\text{model}}\left(y^{(t)}\mid\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)}\}\right),$$

Negative log-likelihood loss, just as an example

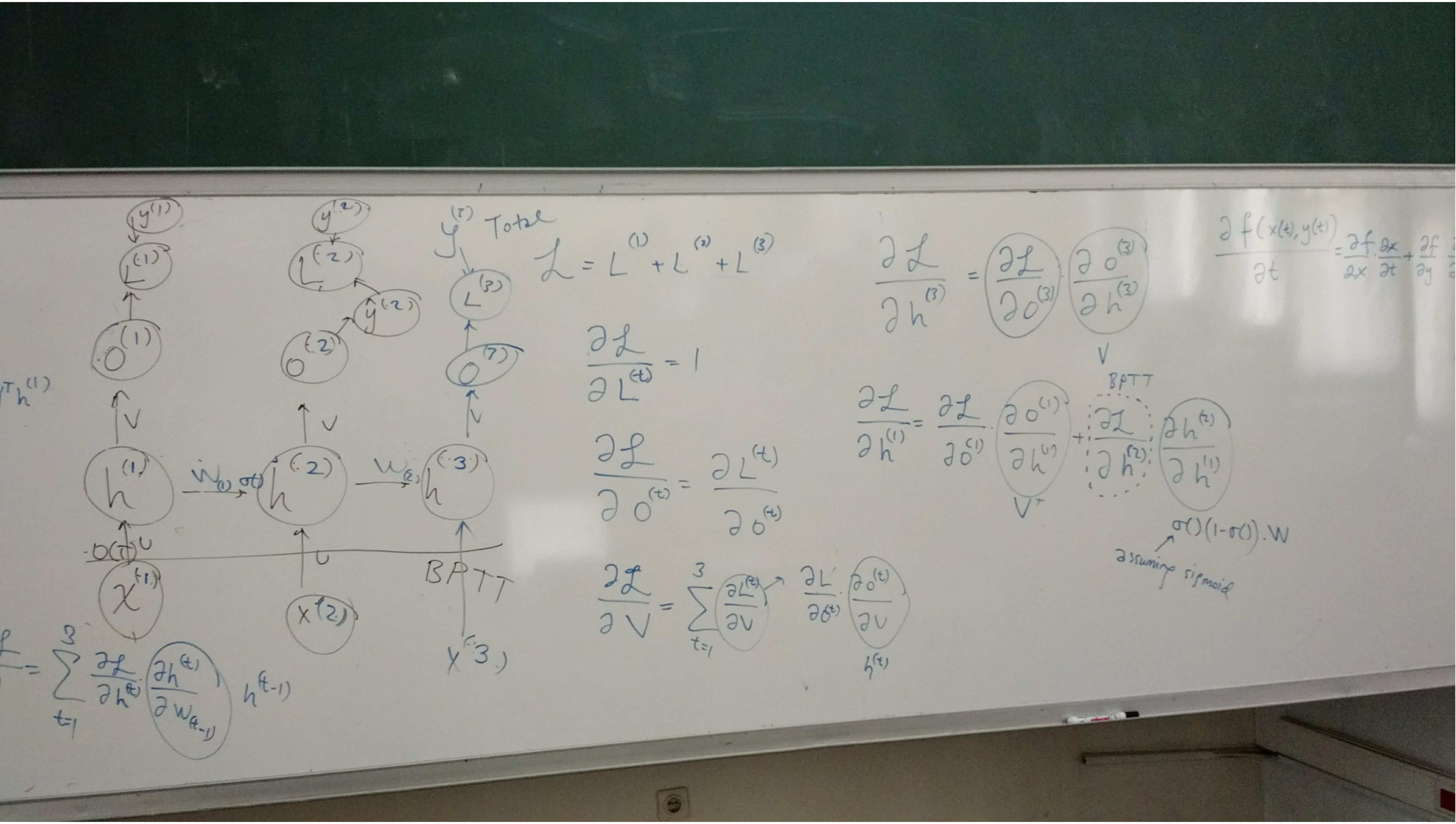
[Fig. 10.3 from Goodfellow et al. (2016)]

Back-propagation in RNNs:

Back-propagation through time (BPTH)

(Nothing new or special but a good exercise)

[Derivation on board]


$y^{(1)}$
$y^{(2)}$
$y^{(3)}$ Total
$L^{(1)}$
$L^{(2)}$
$L^{(3)}$
$\hat{y}^{(2)}$
$o^{(1)}$
$o^{(2)}$
$o^{(3)}$
V
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$W_{(1)}$ $\sigma(t)$
$W_{(2)}$
U
$\sigma(t)$
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
BPTT
$\mathcal{L} = L^{(1)} + L^{(2)} + L^{(3)}$
$\frac{\partial \mathcal{L}}{\partial L^{(t)}} = 1$
$\frac{\partial \mathcal{L}}{\partial o^{(t)}} = \frac{\partial L^{(t)}}{\partial o^{(t)}}$
$\frac{\partial \mathcal{L}}{\partial V} = \sum_{t=1}^{3} \frac{\partial L^{(t)}}{\partial V} \rightarrow \frac{\partial L}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$
$h^{(t)}$
$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{3} \frac{\partial \mathcal{L}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial W_{(t-1)}}$
$h^{(t-1)}$
$\frac{\partial \mathcal{L}}{\partial h^{(3)}} = \frac{\partial \mathcal{L}}{\partial o^{(3)}} \cdot \frac{\partial o^{(3)}}{\partial h^{(3)}}$
V
BPTT
$\frac{\partial \mathcal{L}}{\partial h^{(1)}} = \frac{\partial \mathcal{L}}{\partial o^{(1)}} \cdot \frac{\partial o^{(1)}}{\partial h^{(1)}} + \frac{\partial \mathcal{L}}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}}$
V^T
$\sigma()(1-\sigma()) \cdot W$
assuming sigmoid
$\frac{\partial f(x(t), y(t))}{\partial t} = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}$

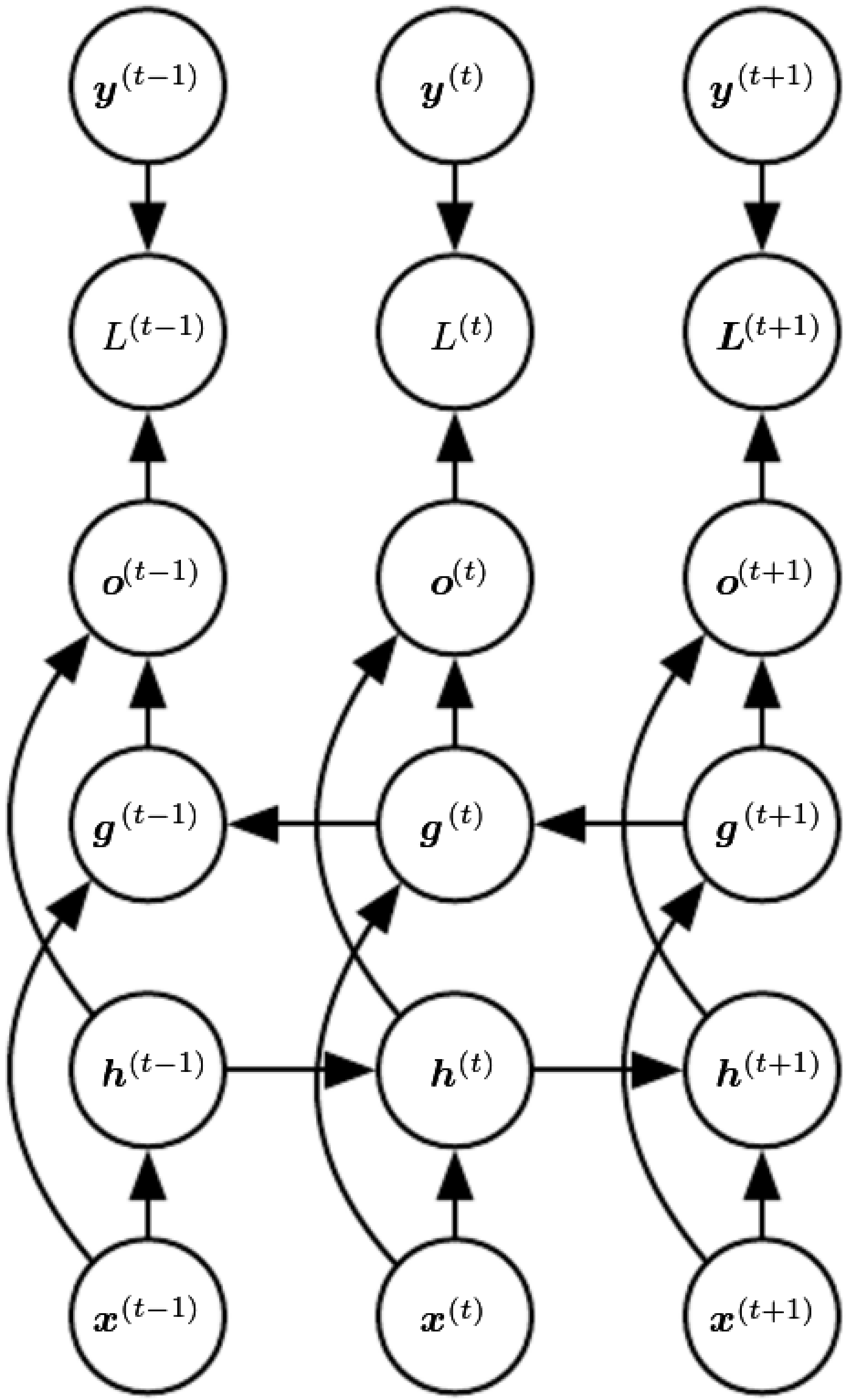
Bi-directional RNNs

So far, we have considered ***causal*** RNNs, i.e.

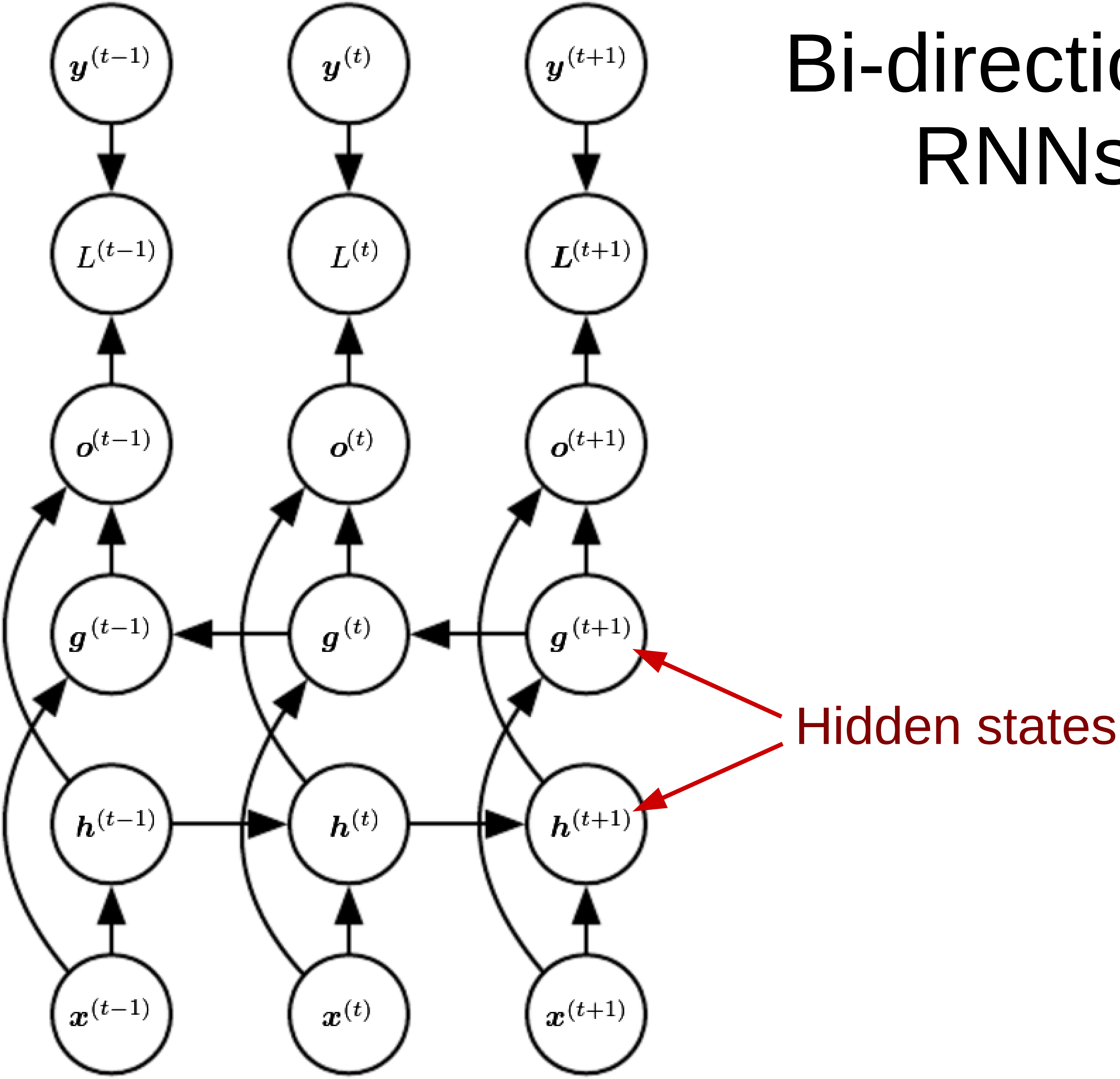
state at time t captures information from the past, i.e. from $x^{(k)}$ where $k<t$

What if we want $\boldsymbol{o}^{(t)}$ to depend on the whole input sequence?

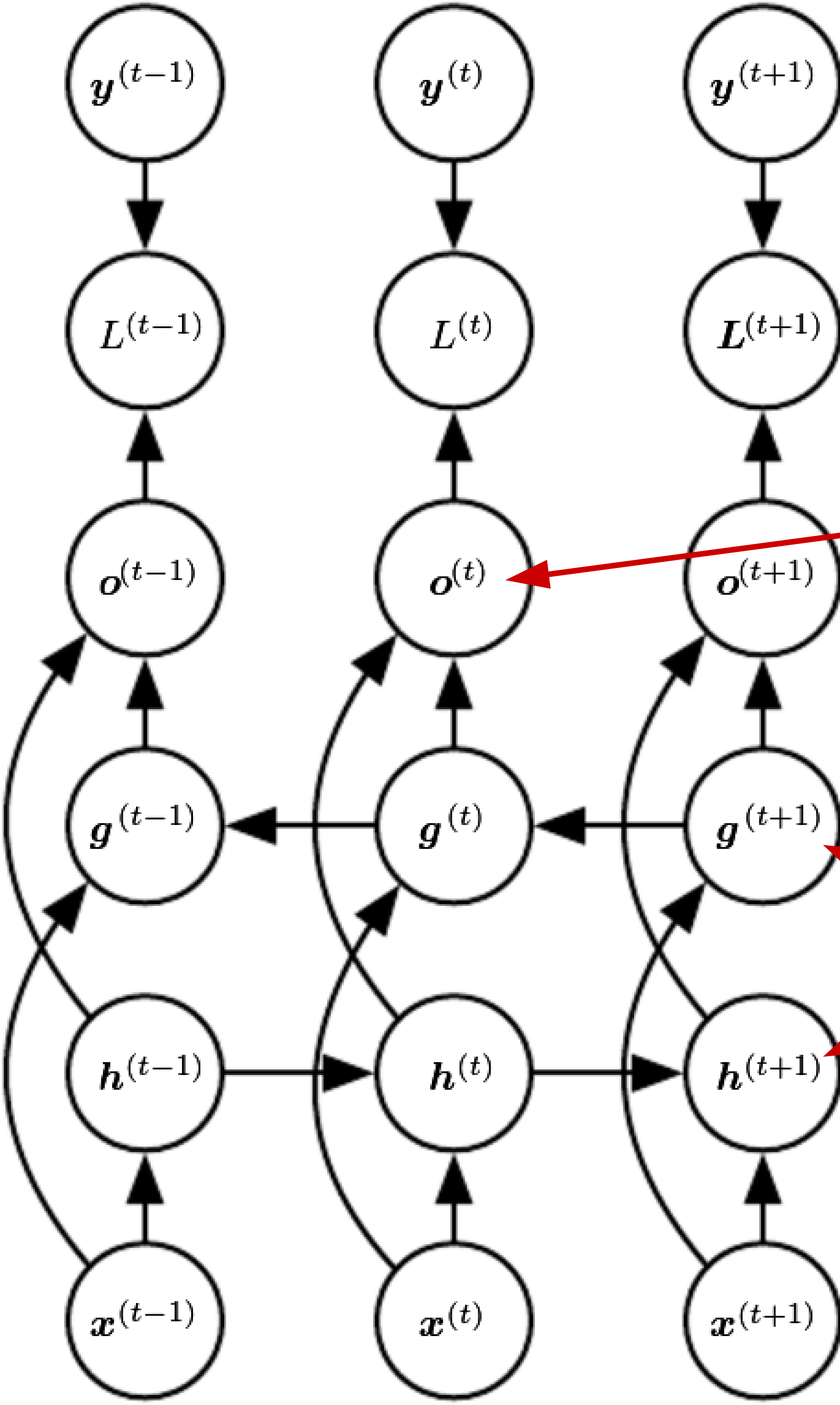
Bi-directional RNNs



Bi-directional RNNs



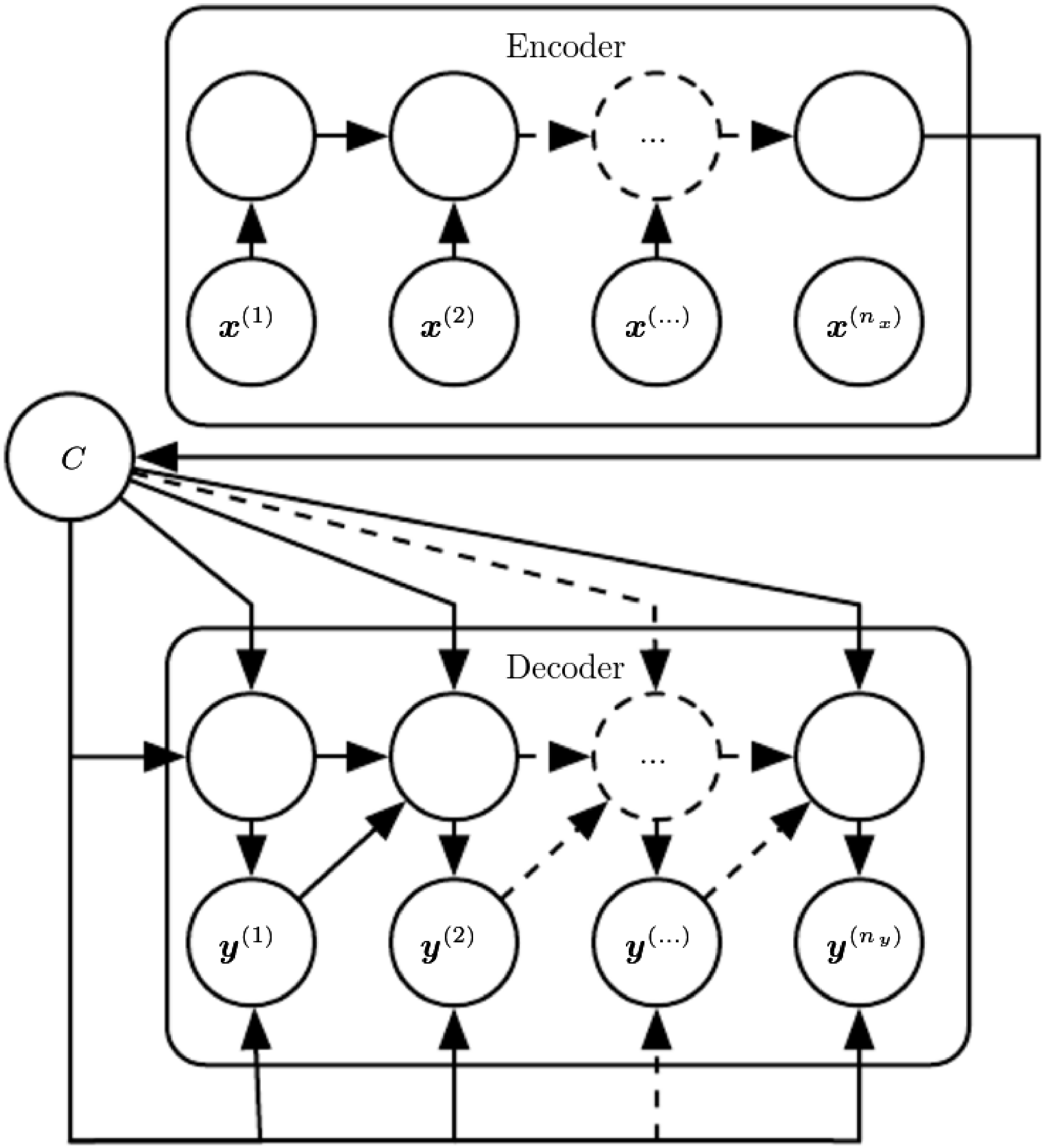
Bi-directional RNNs



Output at t depends on both the *past* and the *future*

Hidden states

Encoder-decoder sequence-to-sequence architectures



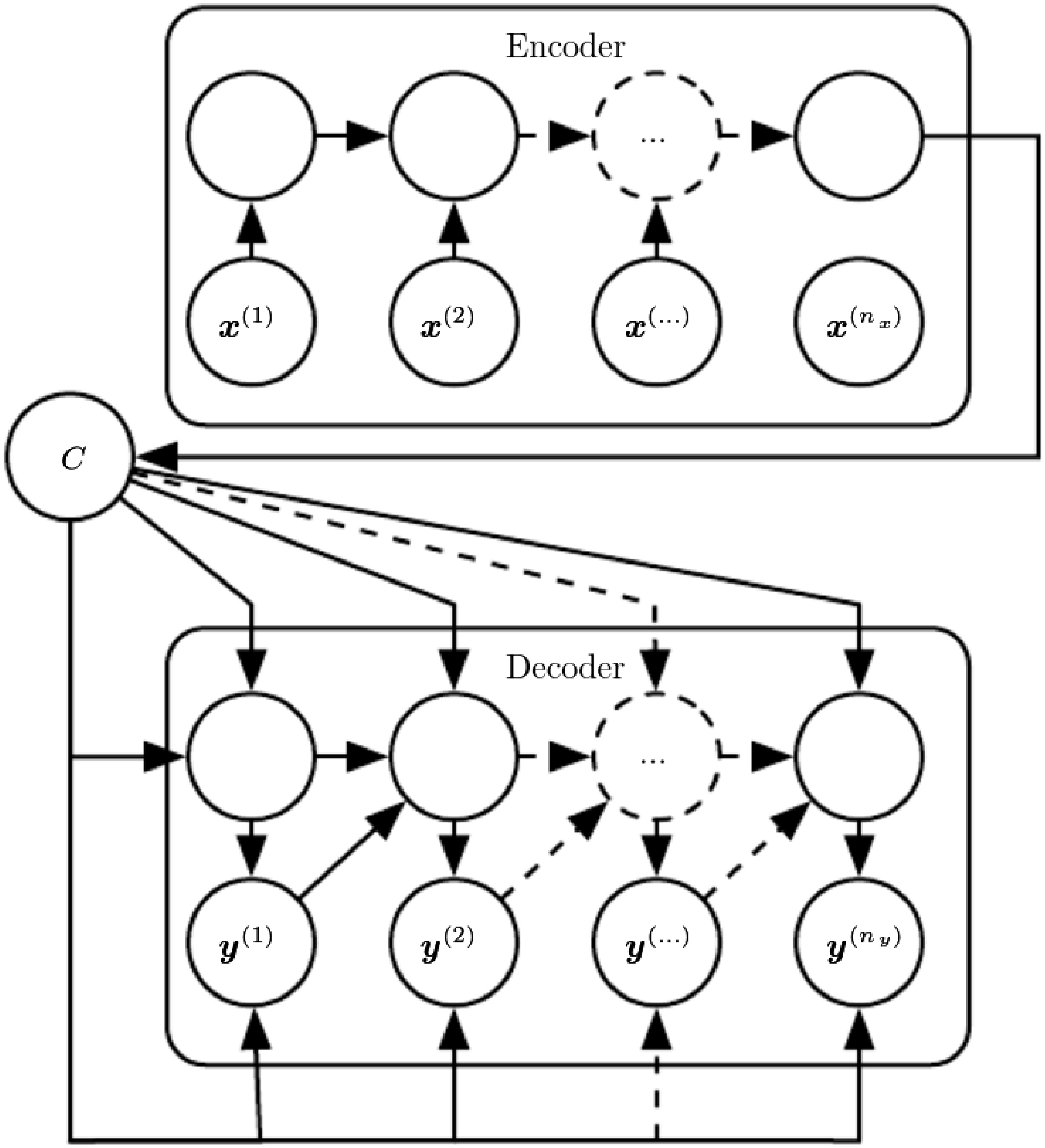
[Fig. 10.12 from Goodfellow et al. (2016)]

Composed of two RNNs

Can map an arbitrary length input sequence to an arbitrary length output sequence (notice n_x and n_y). e.g. machine translation, speech recognition.

[Cho et al. (2014)]
[Sutskever et al. (2014)]

Encoder-decoder sequence-to-sequence architectures

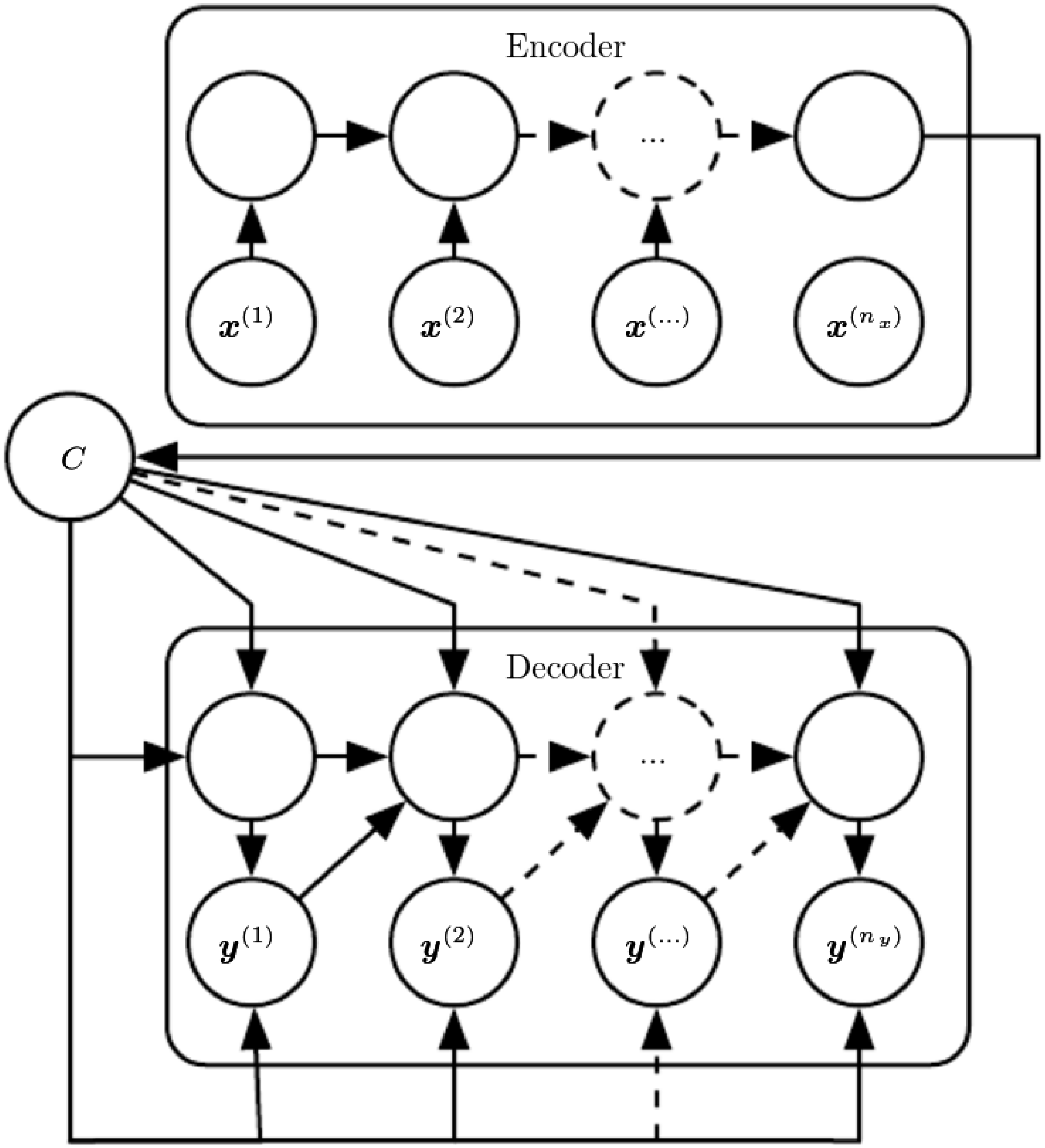


[Fig. 10.12 from Goodfellow et al. (2016)]

Steps

1) Encoder or reader RNN reads processes the input sequence
2) Encoder emits the learned context C (a simple function of its learned hidden states)
3) Decoder or writer RNN which is conditioned on C, produces the output sequence.

Encoder-decoder sequence-to-sequence architectures



[Fig. 10.12 from Goodfellow et al. (2016)]

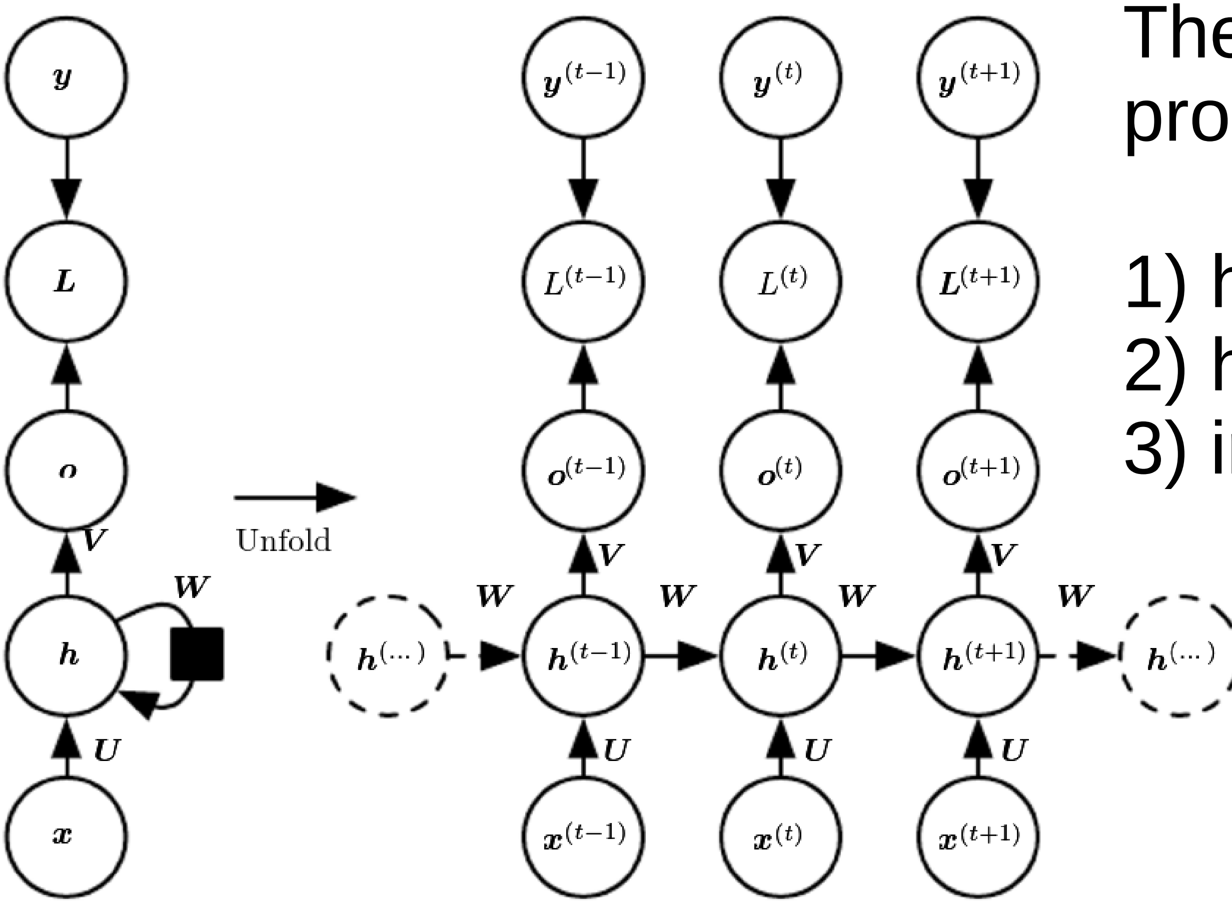
The two RNNs are trained jointly to maximize the average

$$P(\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(n_y)} \mid \boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n_x)})$$

over all (x,y) pairs in the training set.

Typically, $C = h_{n_x}$

How to make RNNs deeper?

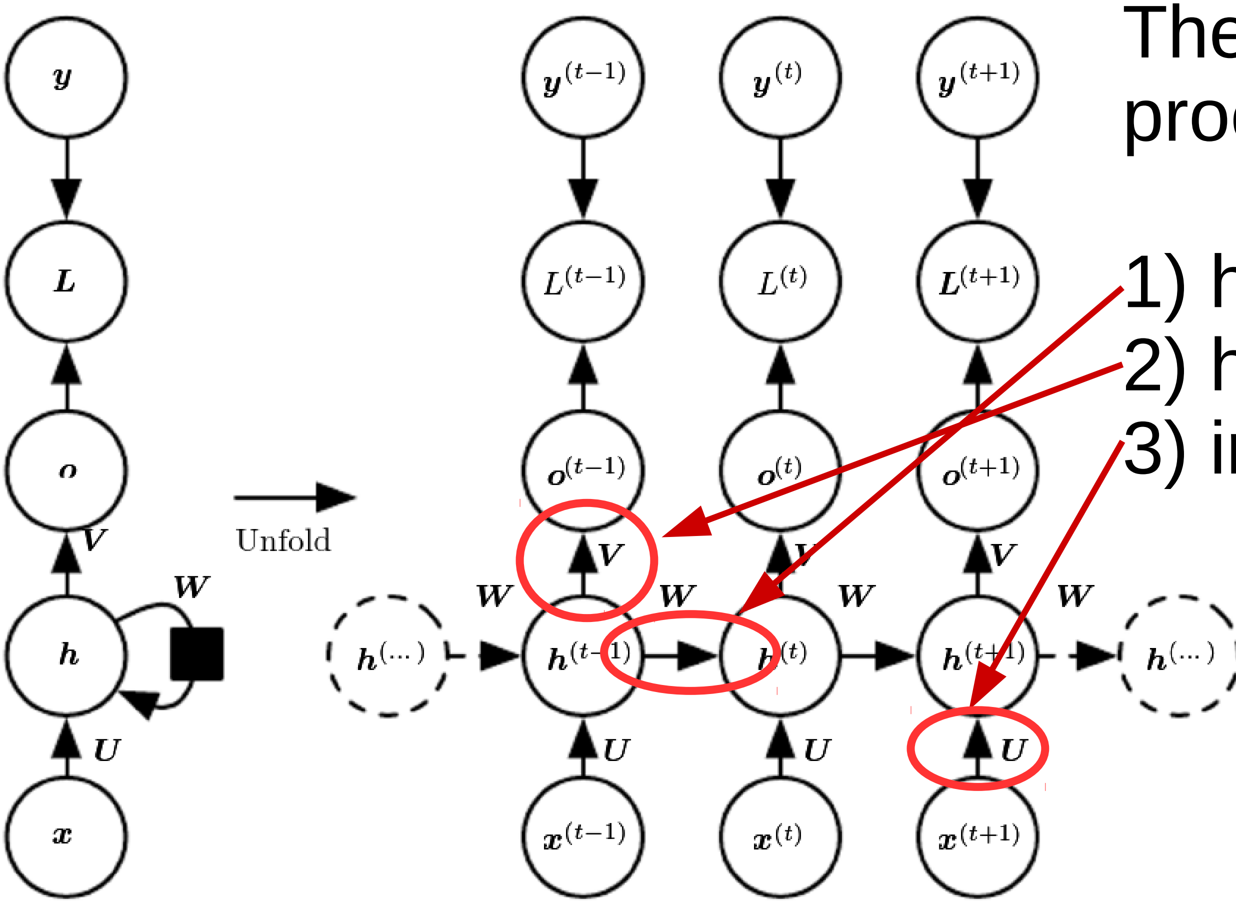


[Fig. 10.3 from Goodfellow et al. (2016)]

There are three processing blocks:

1) hidden to hidden
2) hidden to output
3) input to hidden

How to make RNNs deeper

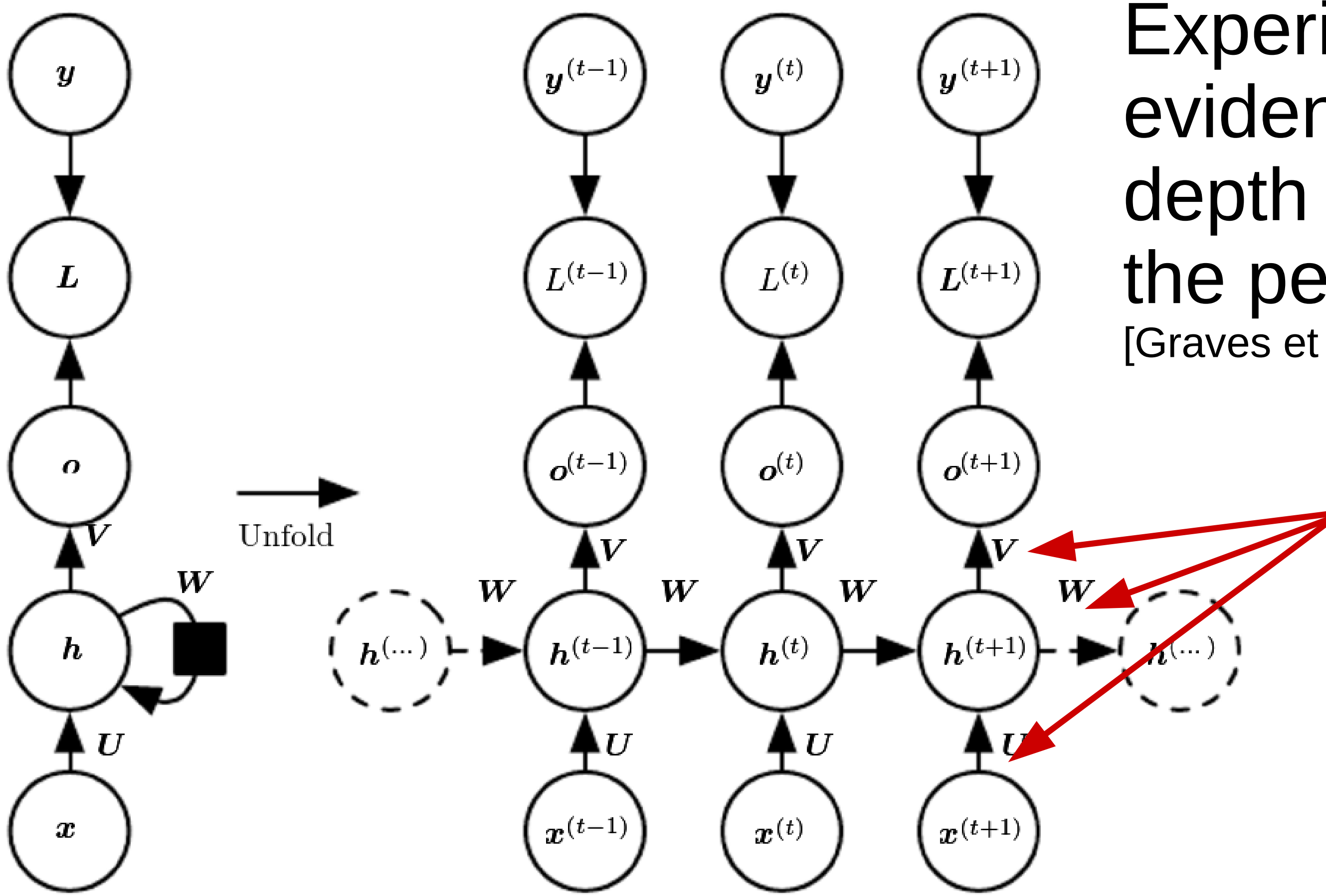


[Fig. 10.3 from Goodfellow et al. (2016)]

There are three processing blocks:

1) hidden to hidden
2) hidden to output
3) input to hidden

How to make RNNs deeper

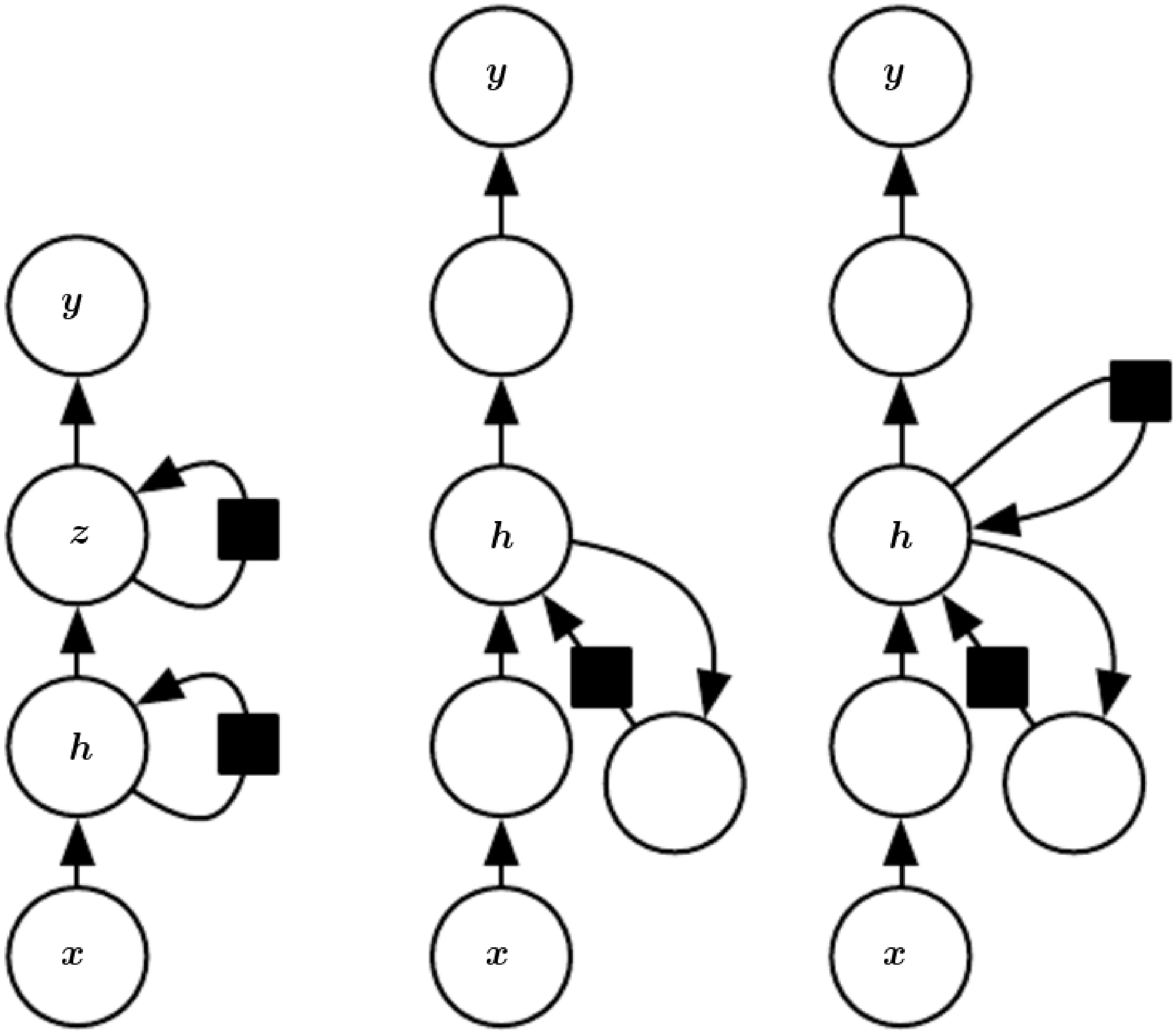


[Fig. 10.3 from Goodfellow et al. (2016)]

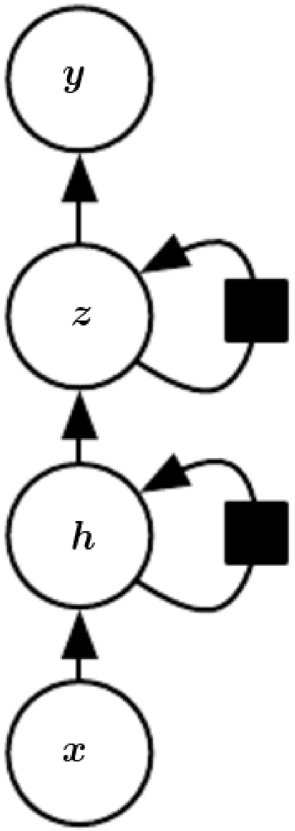
Experimental evidence suggest that depth helps improve the performance.
[Graves et al. (2013)]

Shallow transformations

Three ways of adding depth

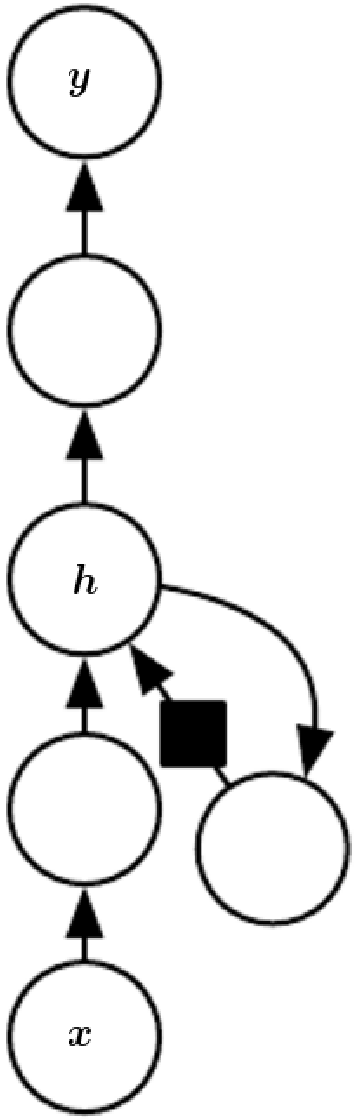


Ways of adding depth



Adding depth to hidden states

Ways of adding depth

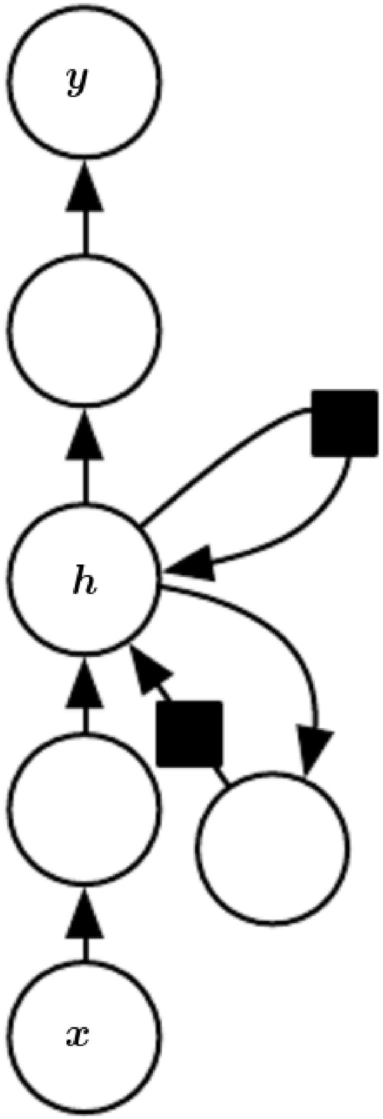


Making each processing block a MLP

Increased capacity

But training becomes harder (optimization is more difficult)

Ways of adding depth



To mitigate the difficult optimization problem, skip connections can be added.

[Pascanu et al. (2014)]

The challenge of long-term dependencies

- More depth → more “vanishing or exploding gradient” problem
- Why?

The challenge of long-term dependencies

- More depth → more "vanishing or exploding gradient" problem
- Why?
- Consider repeated matrix multiplication:

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^\top \boldsymbol{h}^{(t-1)}$$

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^t\right)^\top \boldsymbol{h}^{(0)},$$

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \quad \rightarrow \quad \boldsymbol{h}^{(t)} = \boldsymbol{Q}^\top \boldsymbol{\Lambda}^t \boldsymbol{Q}\boldsymbol{h}^{(0)}$$

Values here will either vanish or explode!

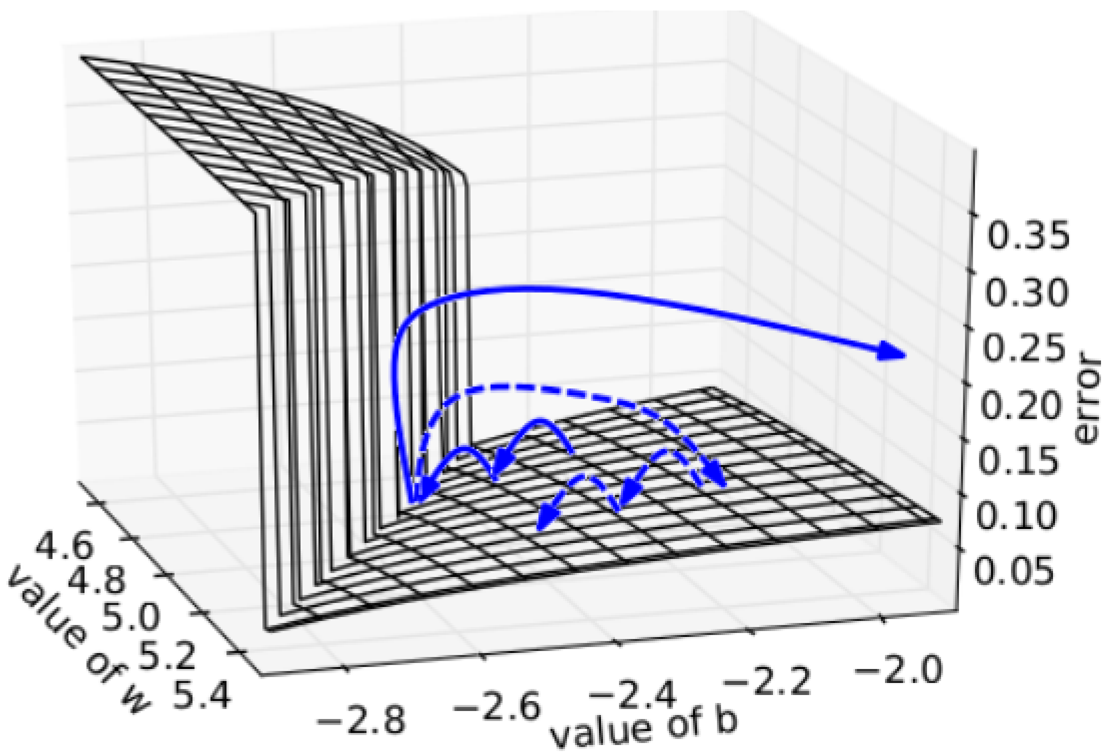
Solution to exploding gradients: gradient clipping

Clip the magnitude.

by Pascanu et al. (2013)

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if



Error surface for a single hidden unit RNN. Solid lines depict trajectories of the regular gradient, dashed lines clipped gradient.
[From Figure 6 in Pascanu et al. (2013)]

Solution to vanishing gradients: regularize the gradient

$$\Omega = \sum_k \Omega_k = \sum_k \left(\frac{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\|}{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \right\|} - 1 \right)^2$$

The regularizer prefers solutions for which **the error preserves norm** as it travels back in time.

[Pascanu et al. (2013)]

Another solution to vanishing gradients is LSTM.

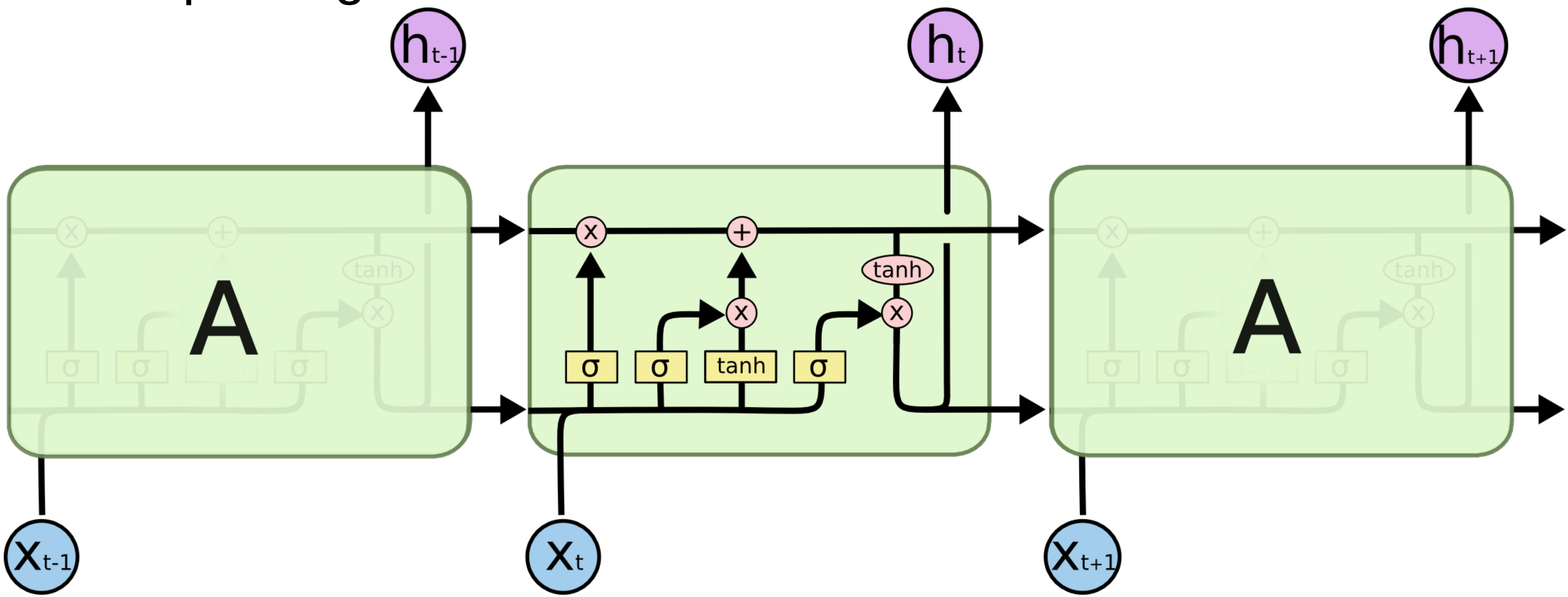
[Hochreiter & Schmidhuber (1997)]

LSTM (Long short-term memory)

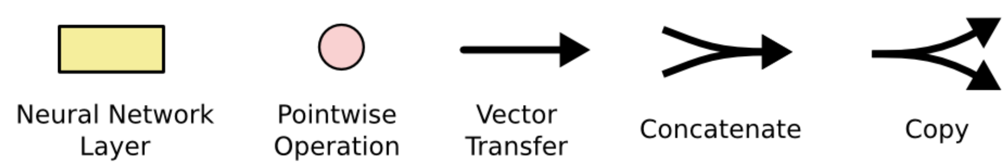
Long short-term memory (LSTM)

[Hochreiter & Schmidhuber (1997)]

A repeating module in a LSTM:



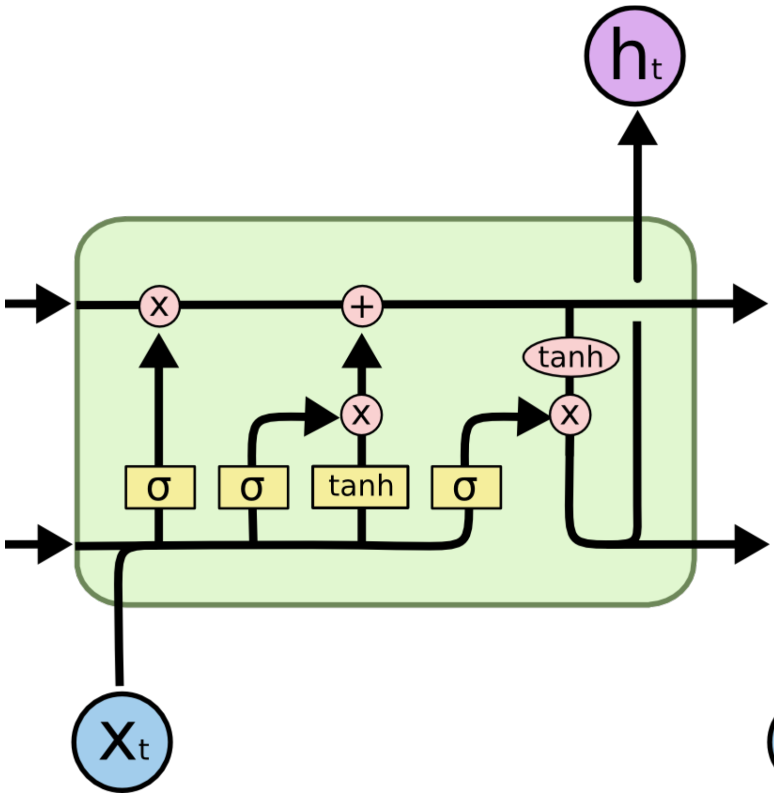
[Figures from C. Olah's blog post.]



Long short-term memory (LSTM)

[Hochreiter & Schmidhuber (1997)]

A repeating module in a LSTM:



[Figure from C. Olah's blog post.]

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

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

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

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

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

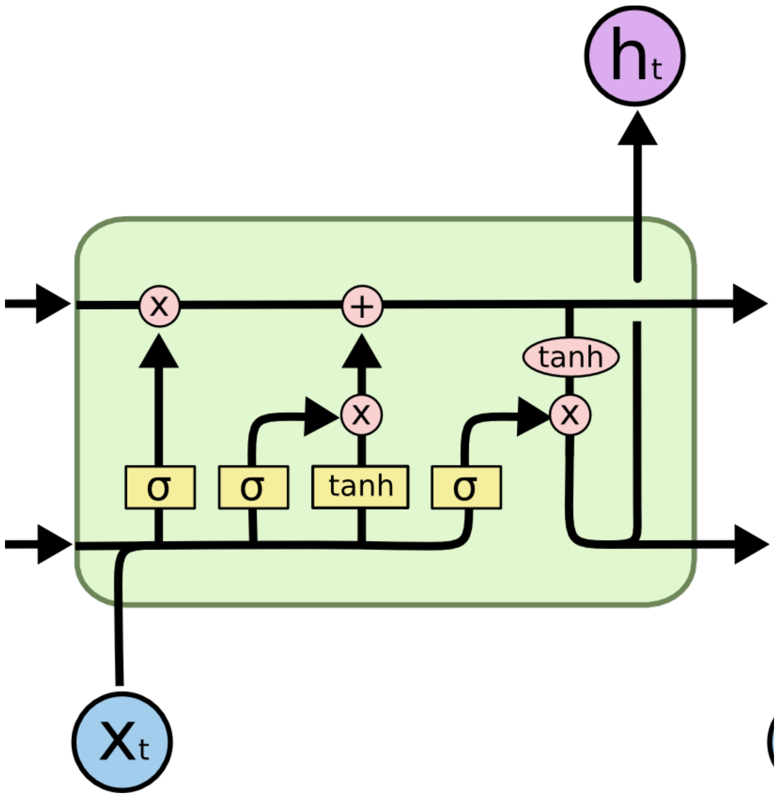
$$h_t = o_t * \tanh\left(C_t\right)$$

[Explanation of these equations on board]

Long short-term memory (LSTM)

[Hochreiter & Schmidhuber (1997)]

Summary



[Figure from C. Olah's blog post.]

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

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

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

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

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

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

Output gate

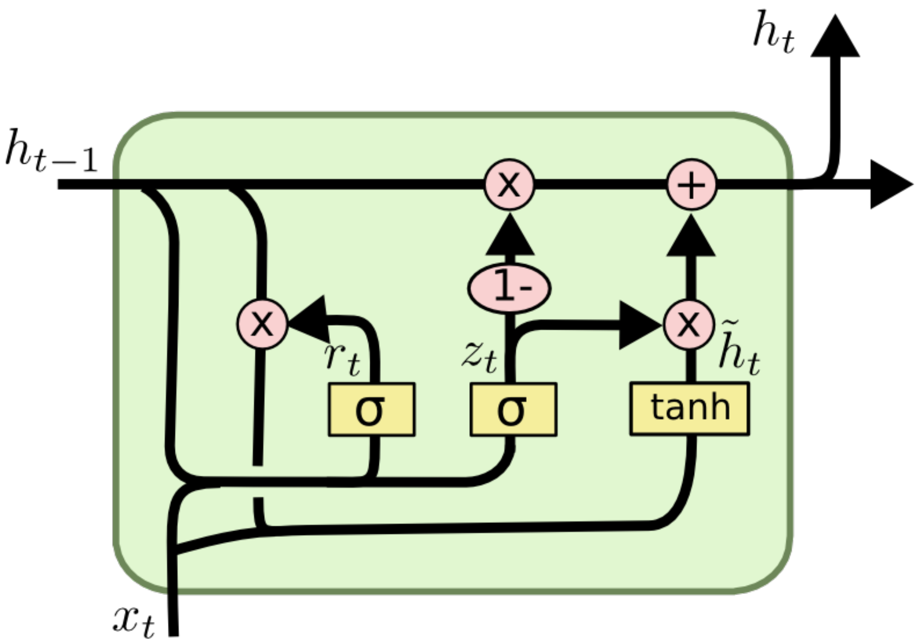
Input gate

Forget gate

The key idea behind LSTM: cells can implement the identity transform. i.e. $C_t = C_{t-1}$ is possible with appropriate gate values.

There are many variants of LSTM

Gated Recurrent Unit (GRU) Cho et al. (2014)



[Figure from C. Olah's blog post.]

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Combines the forget and input gates into a single "update gate." Merges the cell state and hidden state (no C_t), and makes some other changes. The resulting model is simpler than standard LSTM.

An example application of CNN and RNN being used together:

Image captioning

Image captioning

Images from
NeuralTalk Demo

Demo video

[Karpathy and Fei-Fei (2015)]
[Vinyals et al. (2015)]

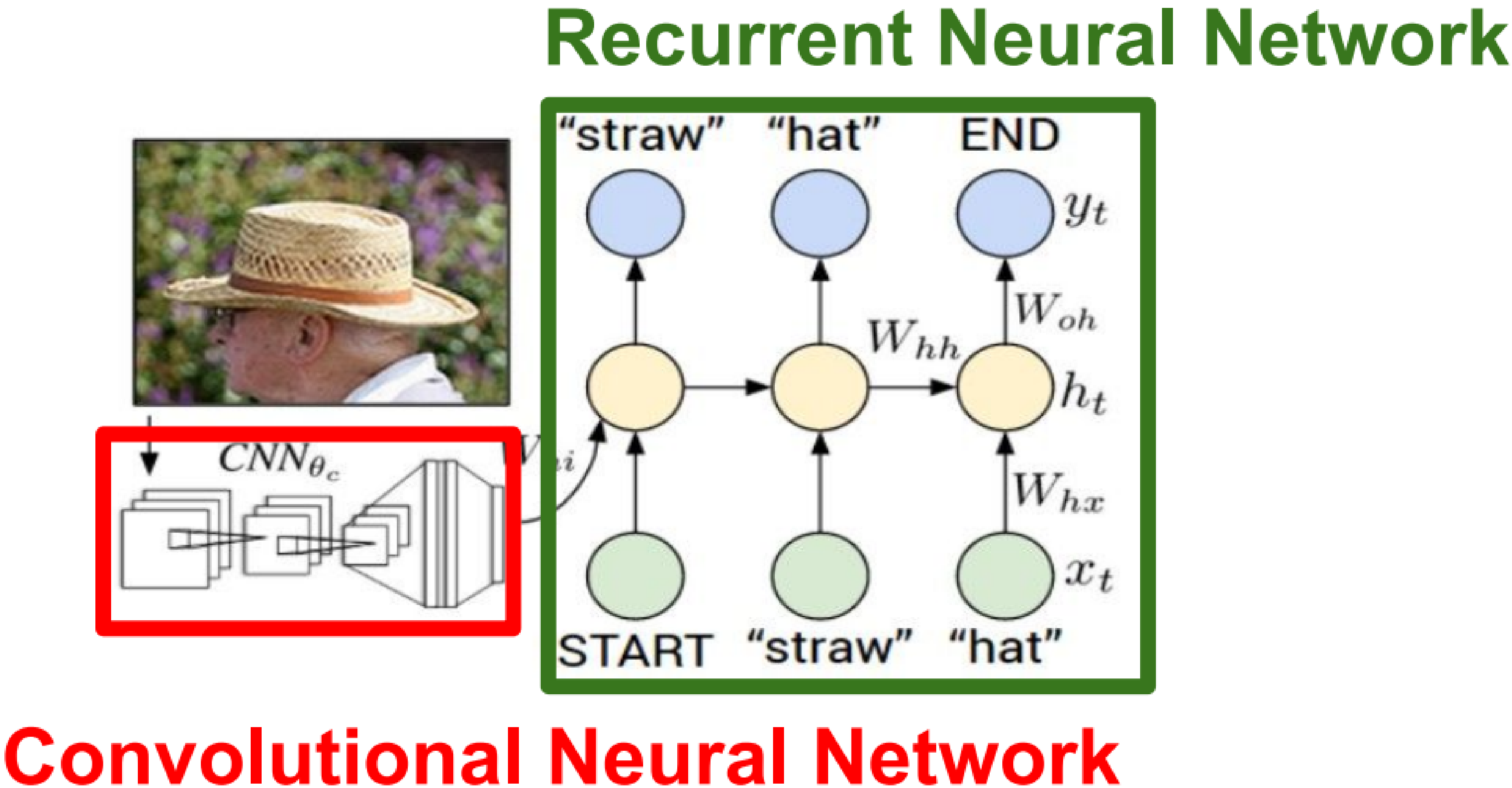
a street sign on a pole in front of a building

a plate with a sandwich and a salad

an elephant standing in a grassy field with trees in the background

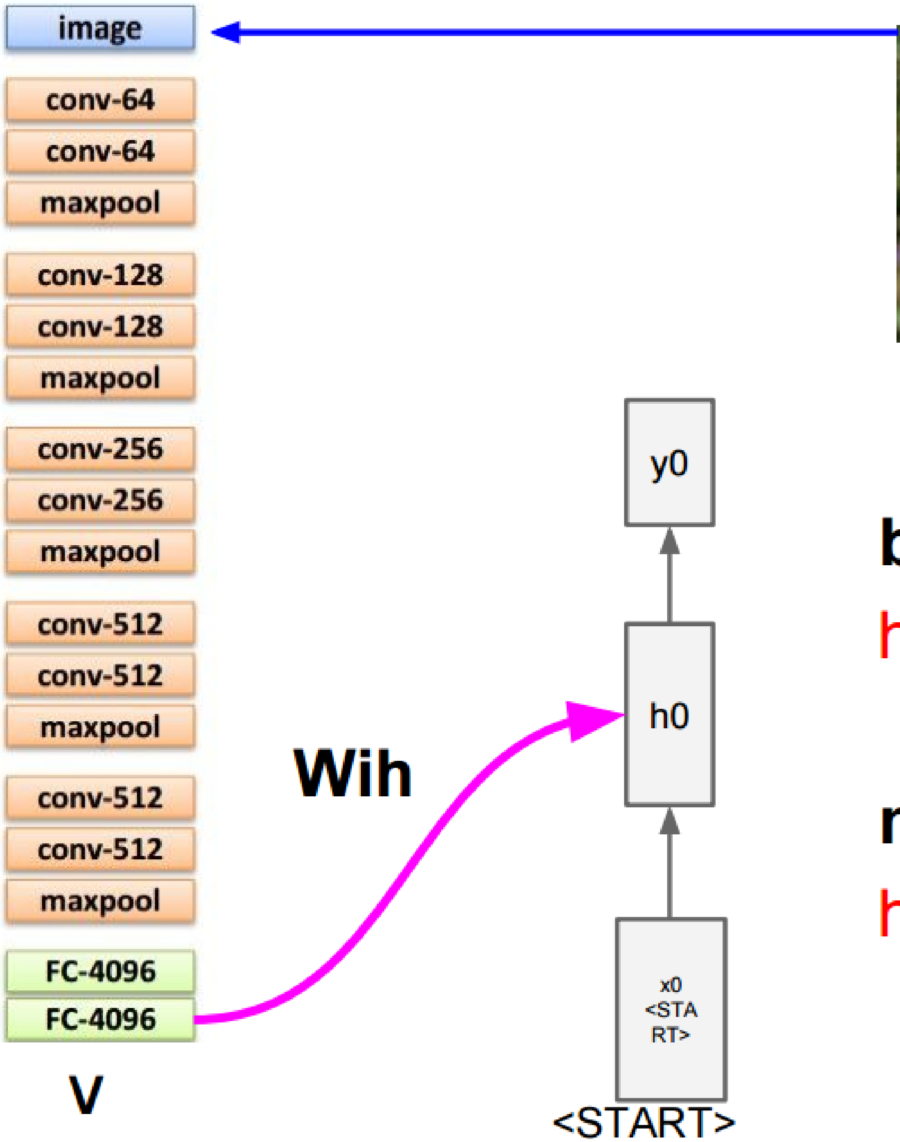
a man is throwing a frisbee in a park

Image captioning



Slide by A. Karpathy http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Image captioning



test image

before:

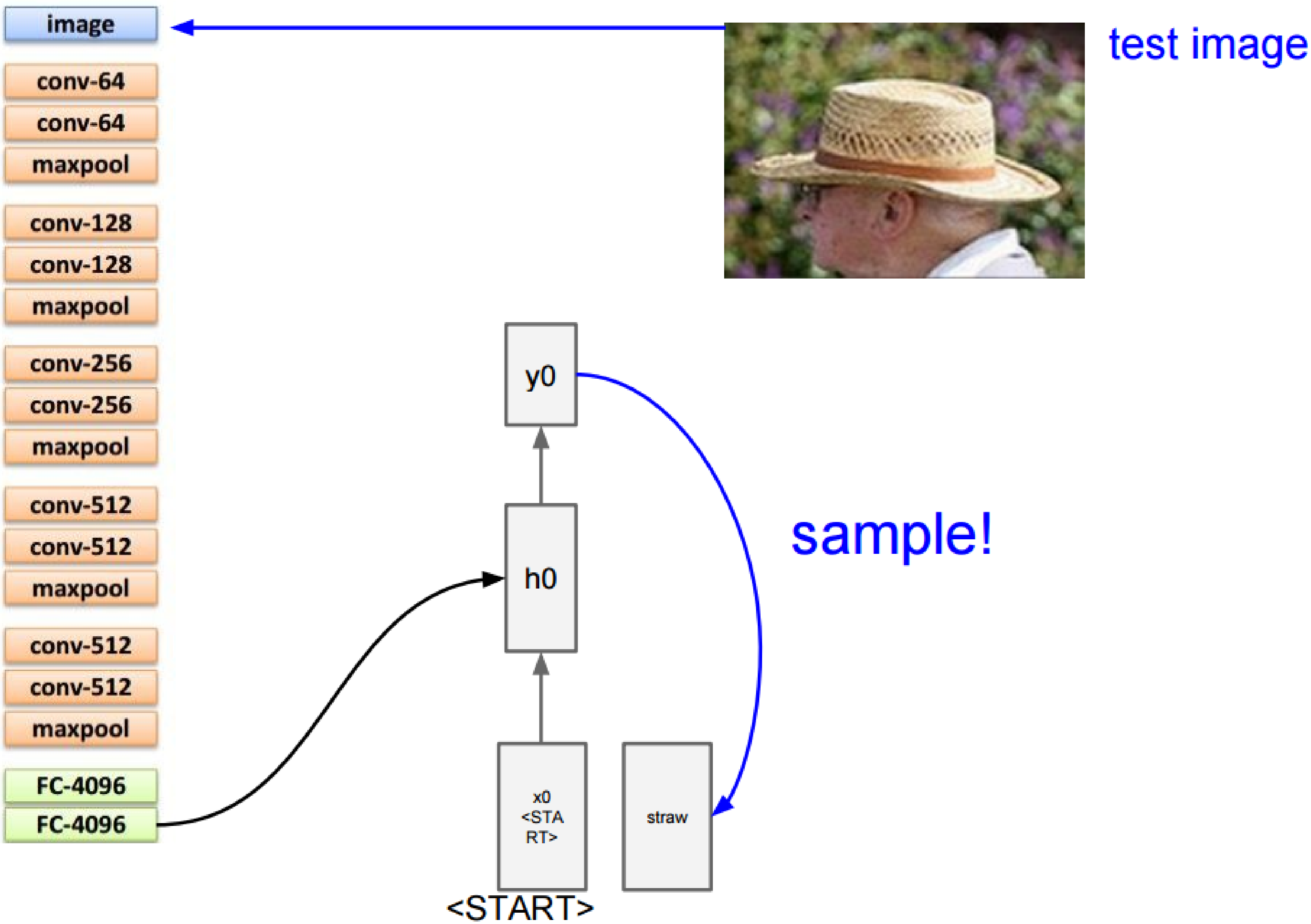
h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

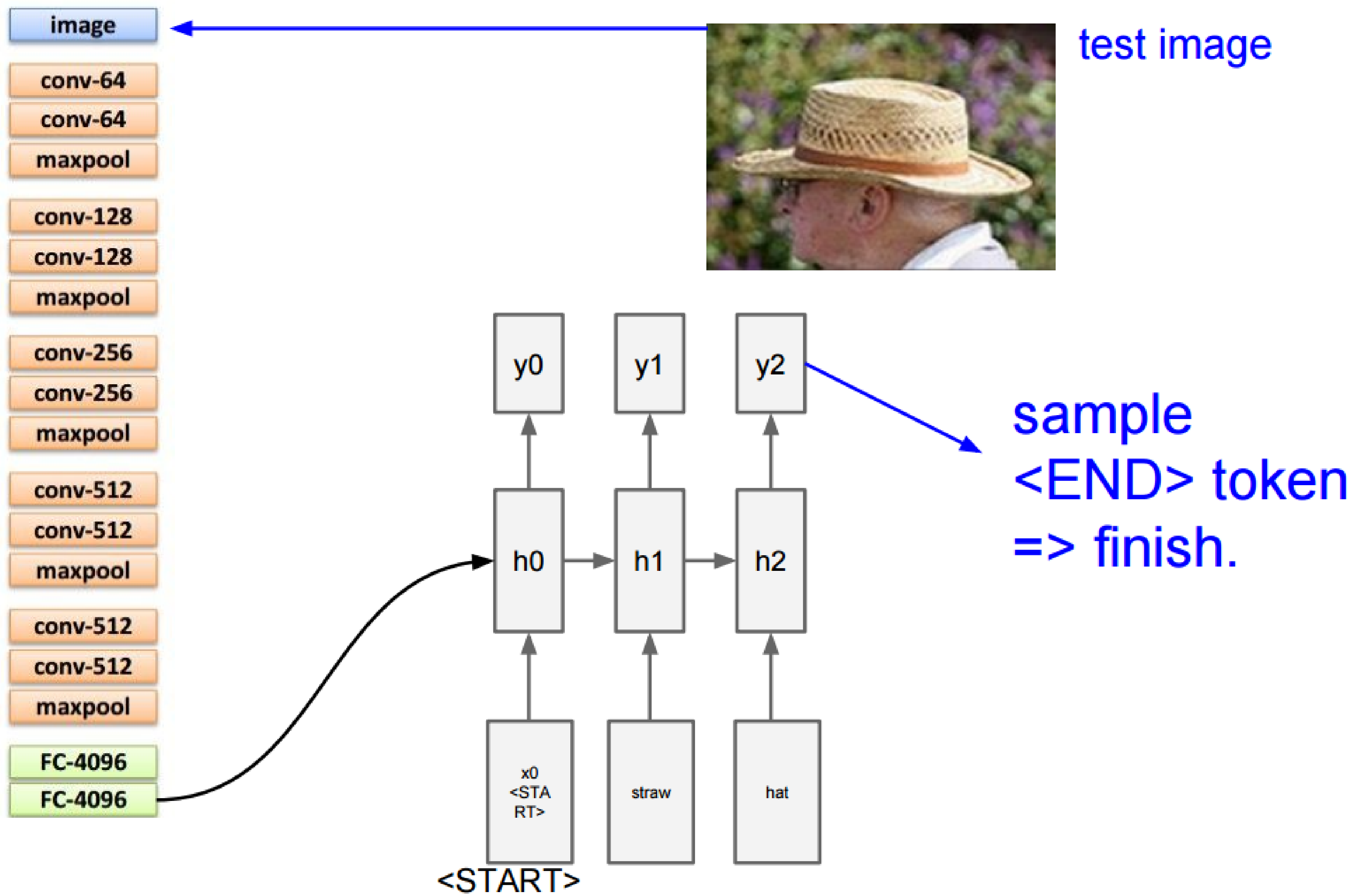
Slide by A. Karpathy http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Image captioning



Slide by A. Karpathy http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Image captioning



Slide by A. Karpathy http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

References

- Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014).
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- Graves, A. (2013). Generating sequences with recurrent neural networks. Technical report, arXiv:1308.0850.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation,9(8), 1735–1780.
- Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). How to construct deeprecurrent neural networks. In ICLR'2014 .
- Sussillo, D. (2014). Random walks: Training very deep nonlinear feed-forward networkswith smart initialization.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning withneural networks. In NIPS'2014, arXiv:1409.3215