Convolutional and Recurrent Networks

COURSE: CS40002

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Convolution and Recurrence

- Convolution is useful for learning artifacts that have a small locality of reference
- Recurrence is useful for learning sequences

The Convolution Operation

Suppose we are tracking the location of a spaceship with a laser sensor.

- **Our laser sensor produces a single output $x(t)$, the position of the spaceship at time t**
- **Suppose that our laser sensor is somewhat noisy, and therefore we wish to take the average of multiple measurements.**
- **More recent measurements have more weight, so we need a weighting function $w(a)$, which returns the weight of measurement taken at the past time, a.**

$$s(t) = \int x(a)w(t-a)da = (x * w)(t)$$

This operation is called *convolution.* The first argument, $x(\,)$, is called the *input*, and the second argument, $w(\,)$, is called the *kernel*.

Discrete Convolution

If we assume that x and w are defined only on integer t, we can define discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

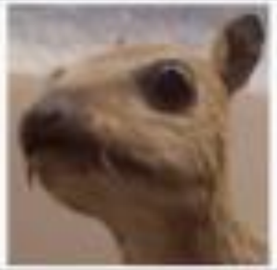
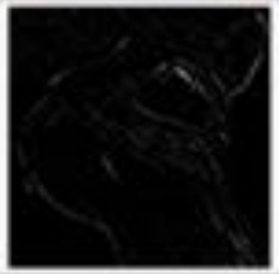
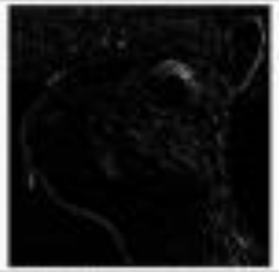
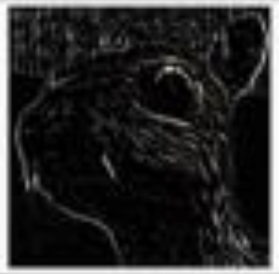
Convolution can also be defined over more than one axis at a time. For example, if we use a two dimensional image I as our input, we may want to use a two dimensional kernel:

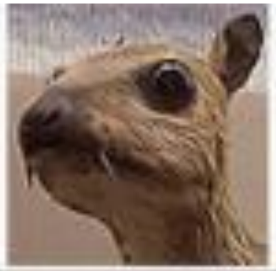
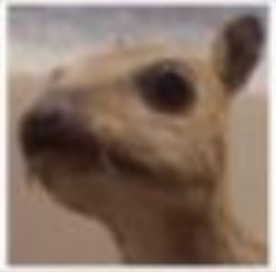
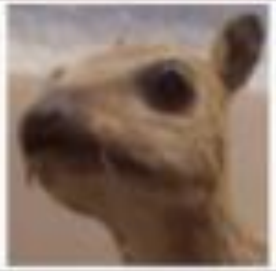
$$s(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m,j-n)$$

Convolution is commutative, that is, we can also write (by replacing m by $i - m$ and n by $j - n$):

$$s(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m,j-n)K(m,n)$$

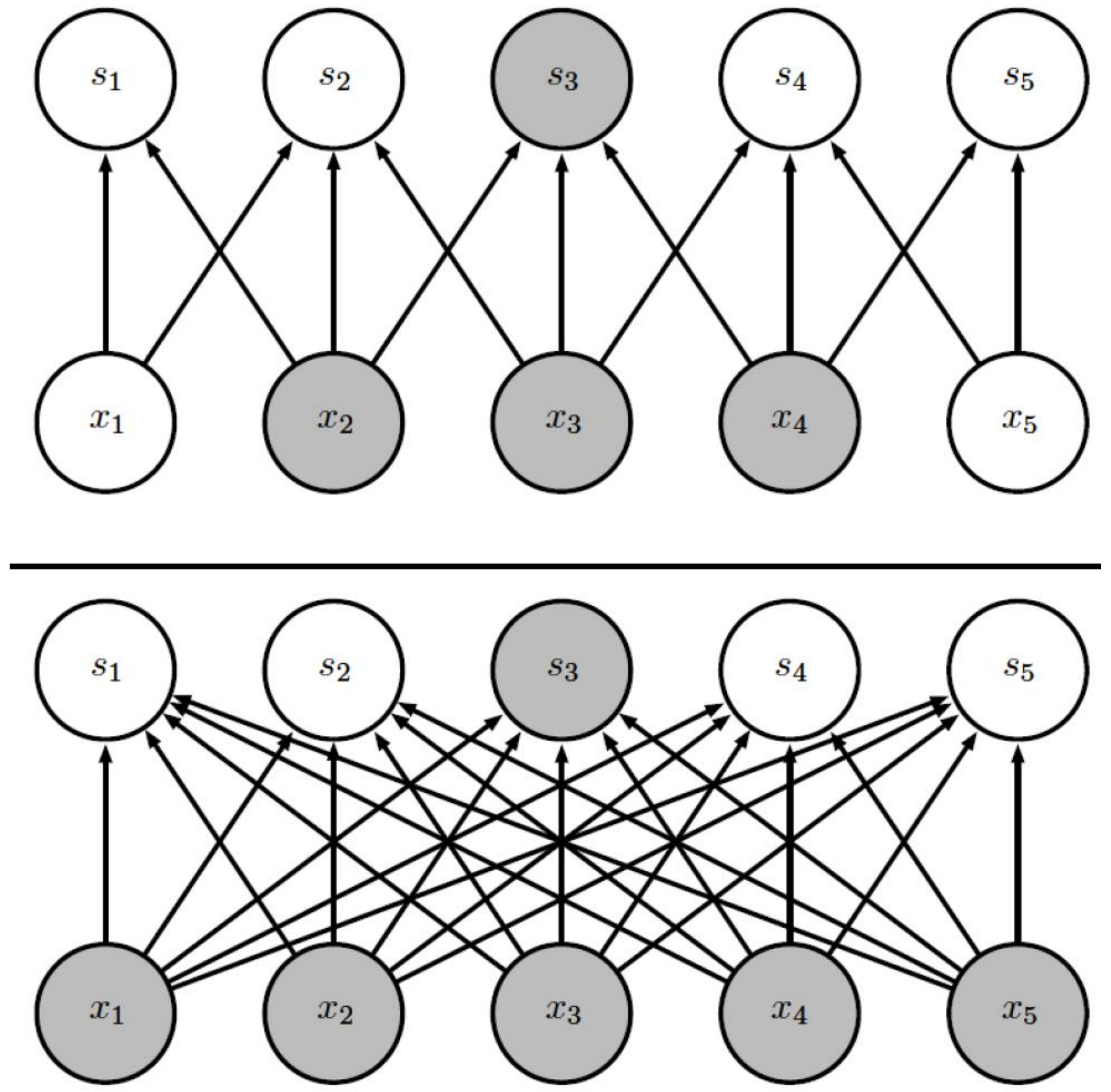
Convolution Networks help us to learn image filters

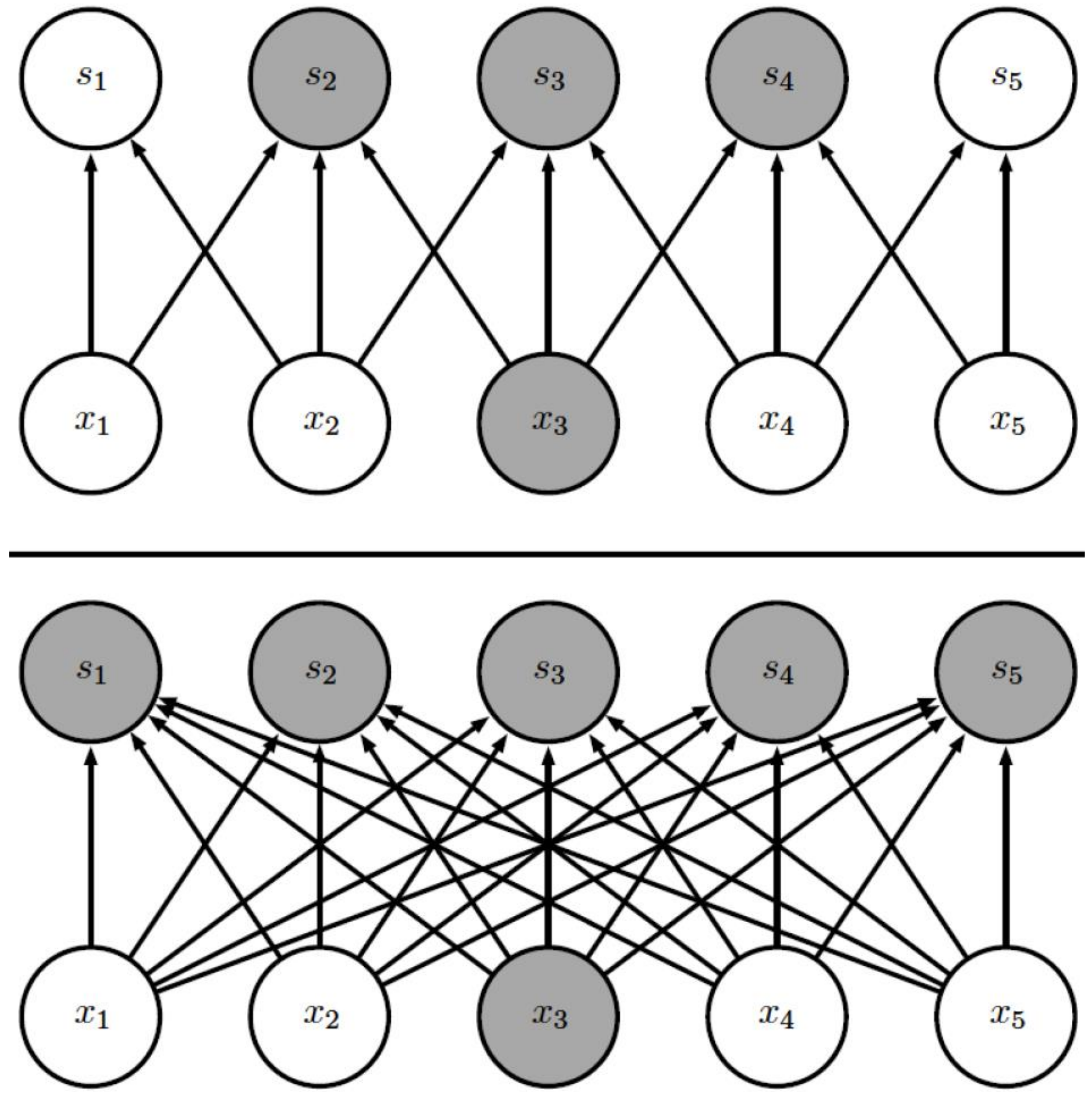
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Machine learning can be used to learn these filters.
- The weights of a convolutional network are learned
- How does the network look like?

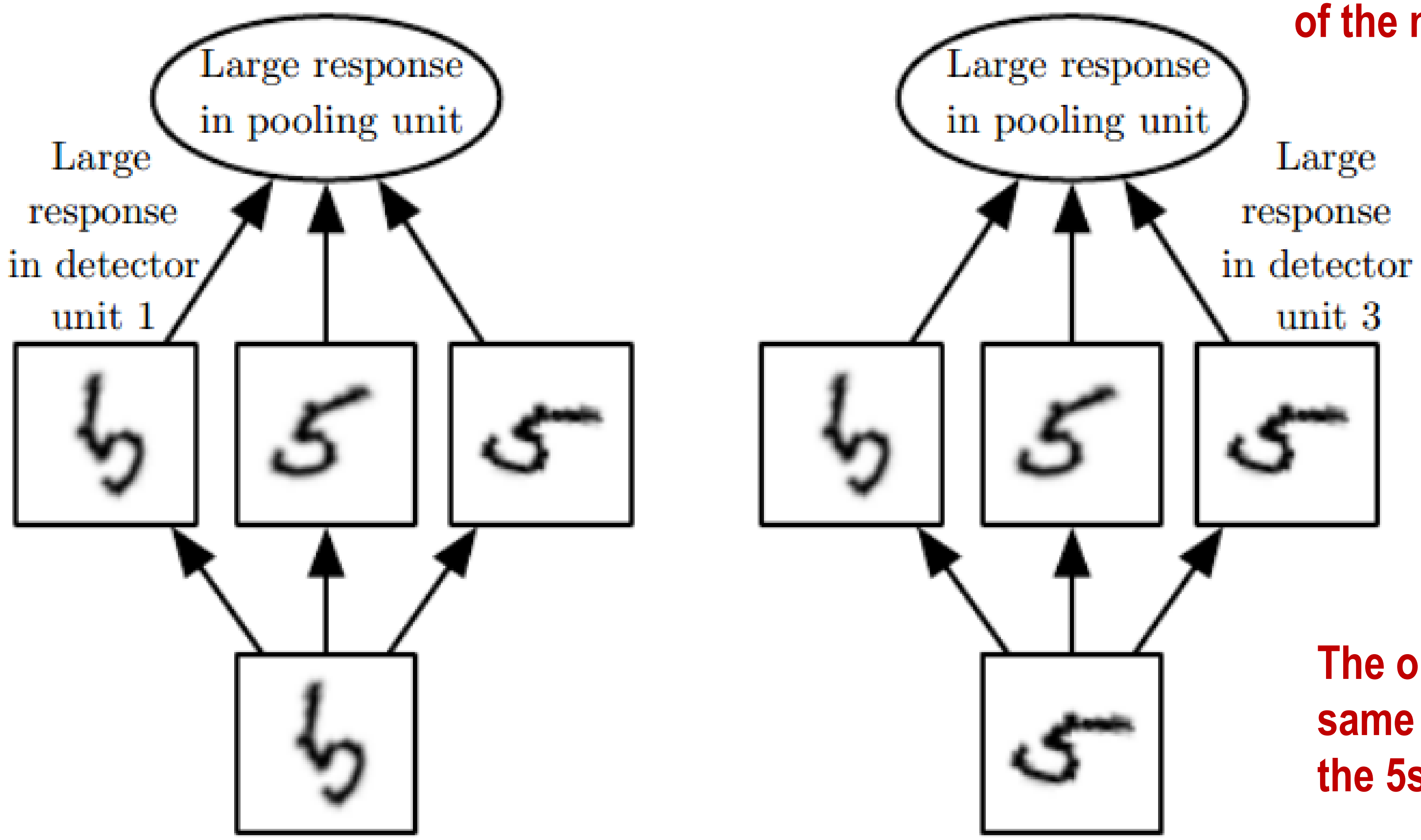
If kernel width is small, the network will be sparse





Convolution and Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs


Large response in pooling unit
Large response in detector unit 1
Large response in pooling unit
Large response in detector unit 3

Set of three learned filters

The output of pooling unit is the same in both cases. Hence both the 5s are recognized.

Sequence Modeling: Recurrent and Recursive Networks

- Recurrent Neural Networks (RNNs) are a family of neural networks for processing sequential data
- Recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialization
  - Most recurrent networks can also process sequences of variable length
- The key idea behind RNNs is *parameter sharing*
  - For example, in a dynamical system, the parameters of the transfer function do not change with time
  - Therefore we can use the same part of the neural network over and over again

Unfolding Computation

Consider a dynamical system:

$$\boldsymbol{s}^{(t)} = \boldsymbol{f}\left(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}\right)$$

where $\boldsymbol{s}^{(t)}$ is the state at time $\boldsymbol{t}$ and $\boldsymbol{\theta}$ is the set of parameters of $\boldsymbol{f}$

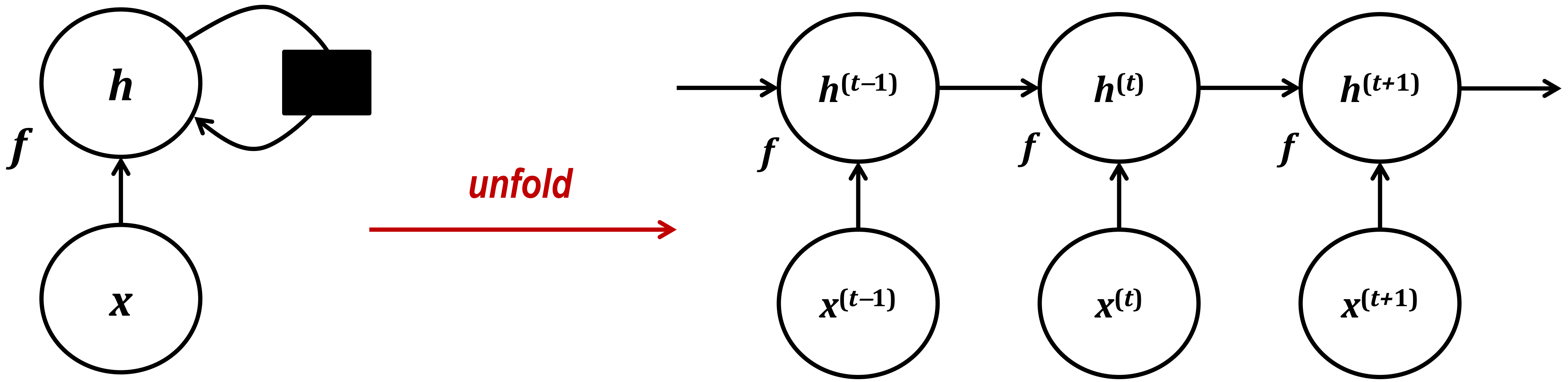
- The state after a finite number of steps can be obtained by applying the definition recursively. For example, after 3 steps:

$$\boldsymbol{s}^{(3)} = \boldsymbol{f}\left(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}\right) = \boldsymbol{f}\left(\boldsymbol{f}\left(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}\right); \boldsymbol{\theta}\right)$$

- For a dynamical system driven by an external input signal $\boldsymbol{x}^{(t)}$:

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

Unfolding computation and Recurrent Network



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

- Regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states
- It is possible to use the *same* transition function *f* with the same parameters at each step

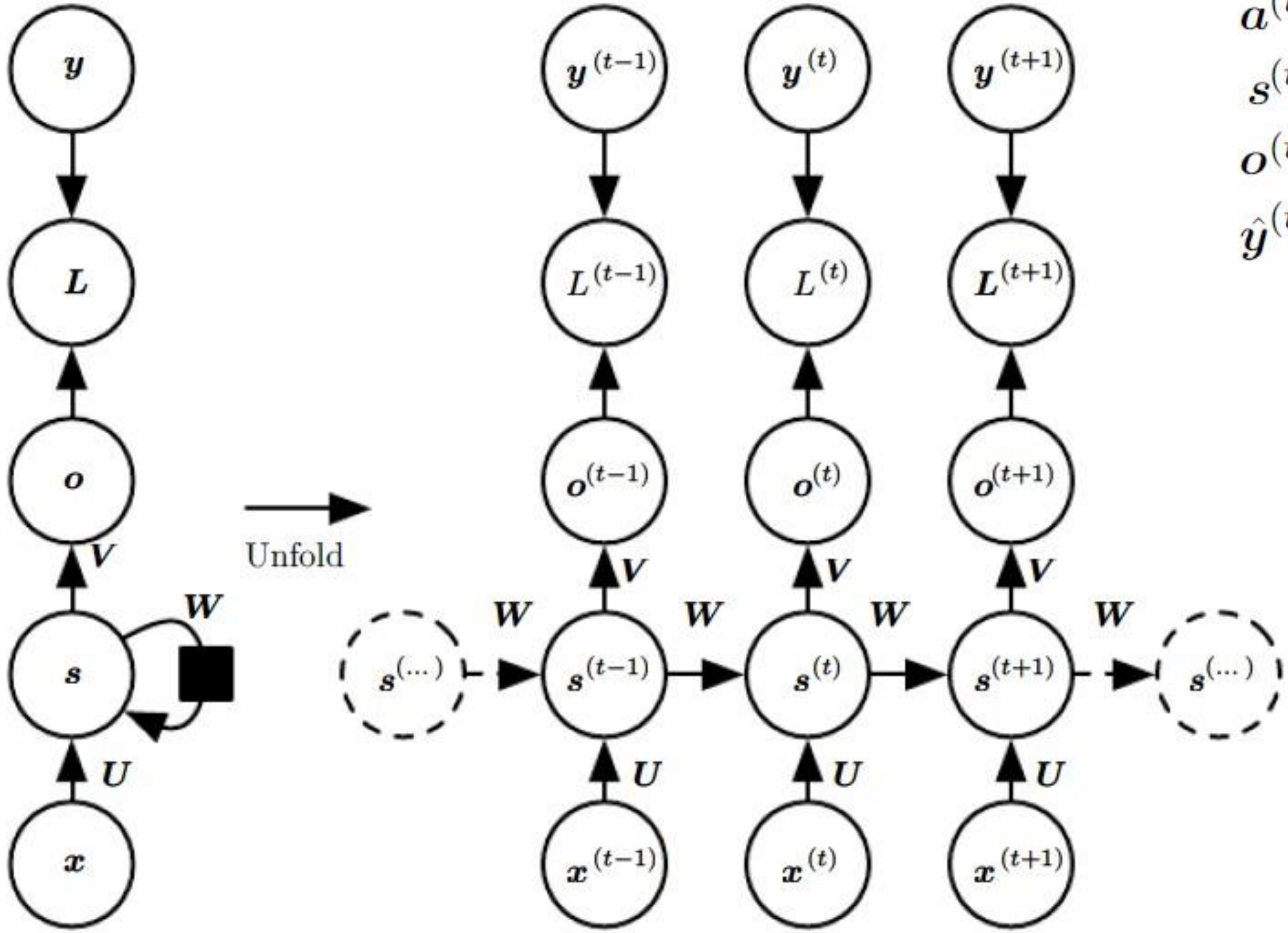
Useful topologies of RNNs

- RNNs that produce an output at each time step and have recurrent connections between hidden units

- RNNs that produce an output at each time step and have recurrent only from the output at one time step to the hidden units at the next time step

- RNNs with recurrent connections between hidden units, that read an entire sequence and then produce a single output

RNN with hidden-hidden feedback

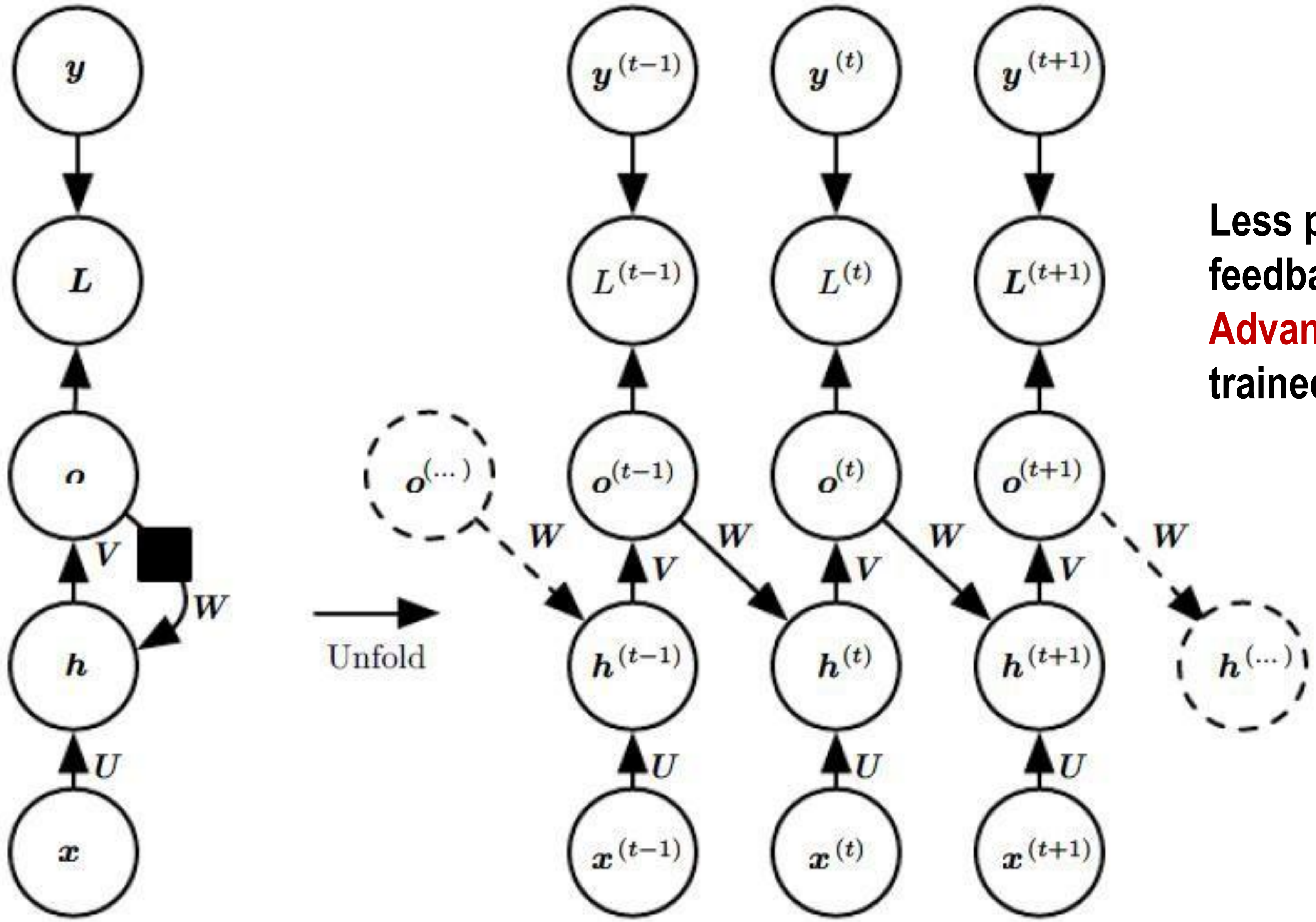


$$
\begin{aligned}
a^{(t)} &= b + W s^{(t-1)} + U x^{(t)} \\
s^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + V s^{(t)} \\
\hat{y}^{(t)} &= \text{softmax}(o^{(t)})
\end{aligned}
$$

RNN with hidden-hidden feedback is *universal*. Any function computable by a Turing machine can be computed by such a RNN of finite size (weights can have infinite precision).

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

RNN with output-hidden feedback



Less powerful than the hidden-hidden feedback model.
Advantage: Each time step can be trained in isolation (why?)

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

RNN with output only at the end

Can be used to summarize a sequence and produce a fixed-size representation to be used as an input for further processing

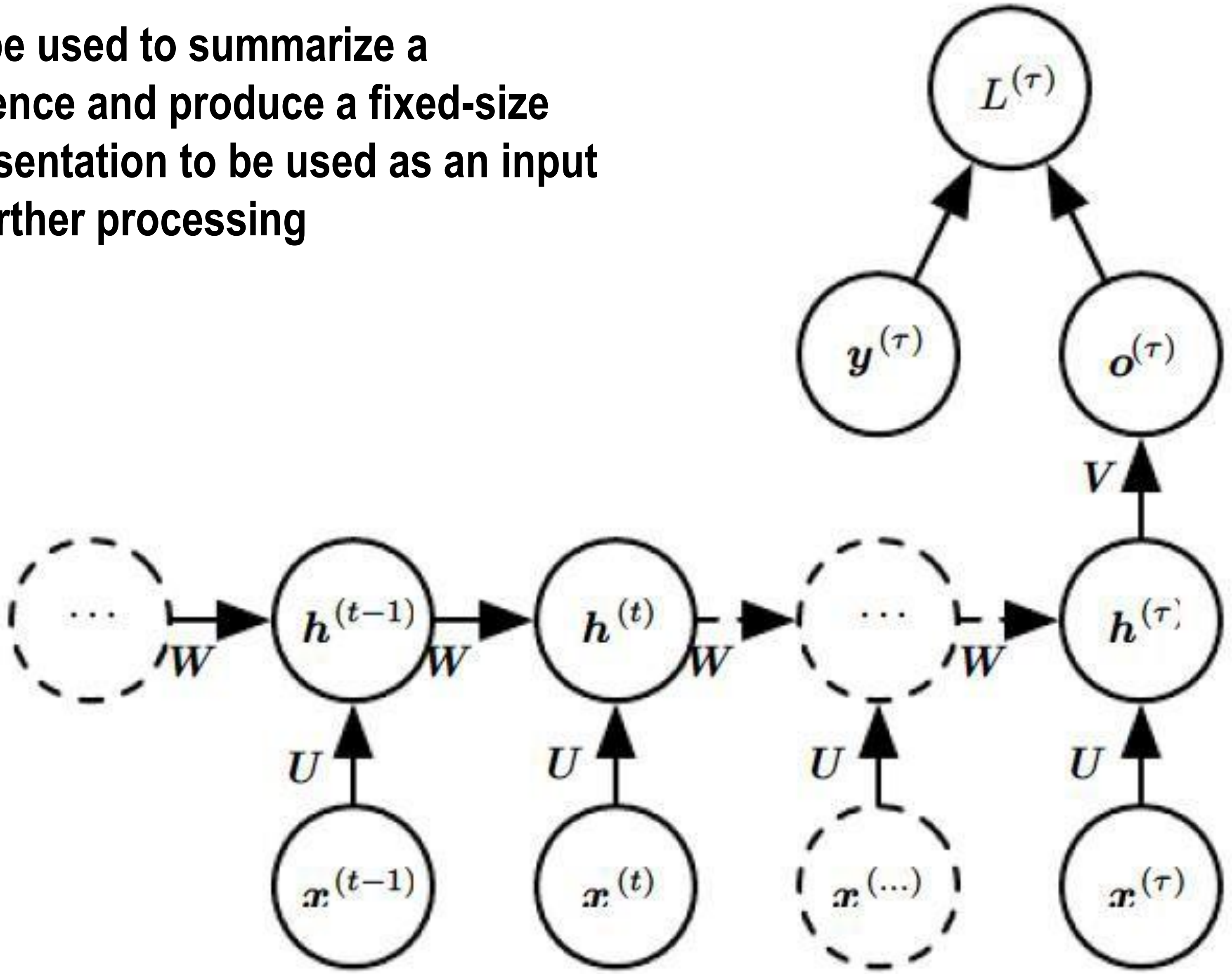


Figure from *Deep Learning*, Goodfellow, Bengio and Courville