Neural Networks and Deep Learning

Brain inspired way to learn from patterns

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

The ML problem in regression

What is the function $f(.)$?

Solution: *This is where the different ML methods come in*

- Linear model: $f(x) = w^T x$

- Linear basis functions: $f(x) = w^T \phi(x)$
  - Where $\phi(x) = [\phi_0(x) \ \phi_1(x) \ ... \ \phi_L(x)]^T$ and $\phi_l(x)$ is the basis function.
  - Choices for the basis function:
    - Powers of x: $\phi_l(x) = x^l$
    - Gaussian / Sigmoidal / Fourier / ...

- Neural networks

- ...

Classification

Given training data set with:

- Input values: $x_n = [x_1 \; x_2 \; ... \; x_M]^T$ for $n = 1 \, ... \, N$.

- Output class labels, for example:

  - 0/1 or –1/+1 for binary classification problems
  - 1 ... K for multi-class classification problems
  - 1-of-K coding scheme:
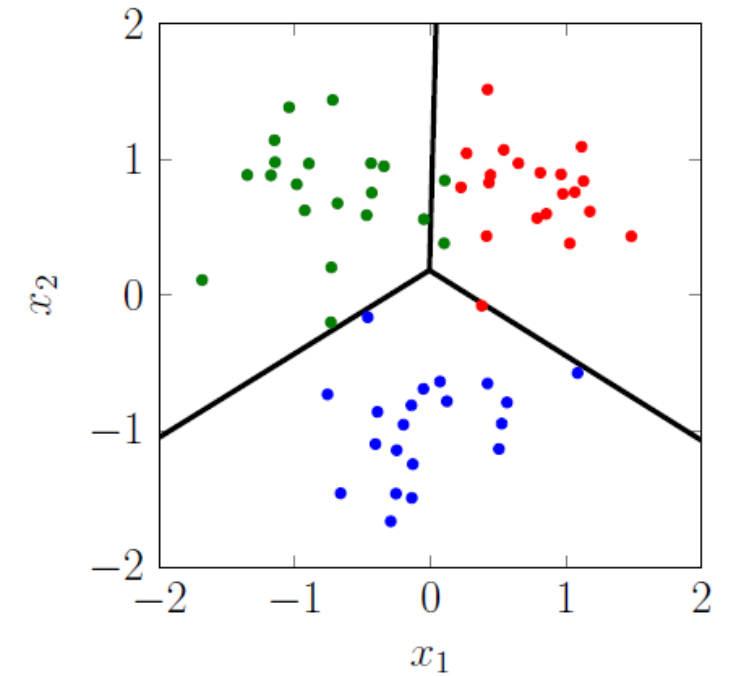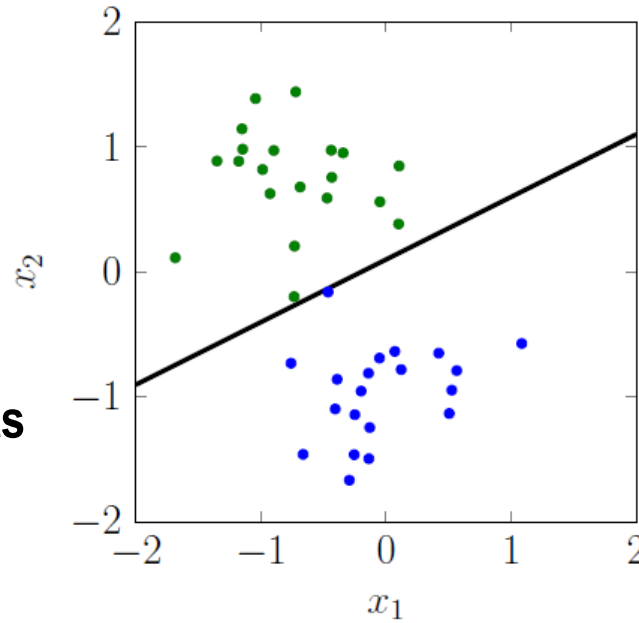
$$y = [0 \; ... \, 0 \; 1 \; 0 \; ... \, 0]^T$$

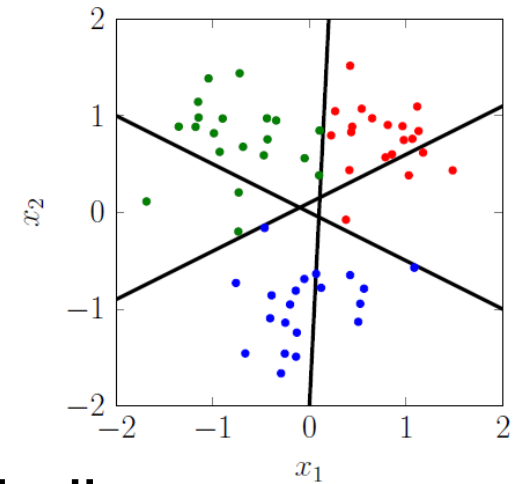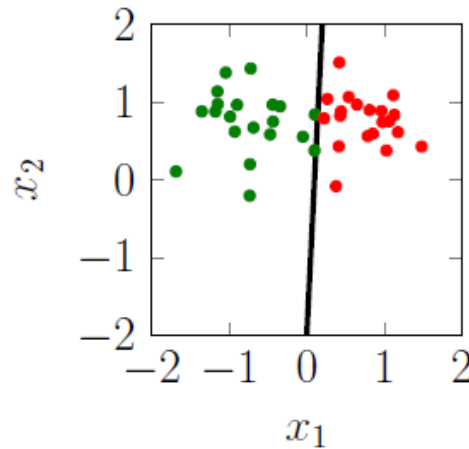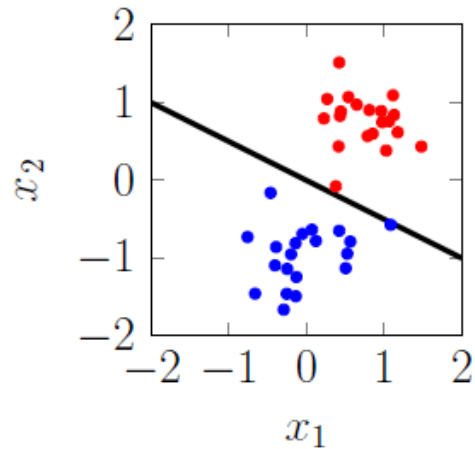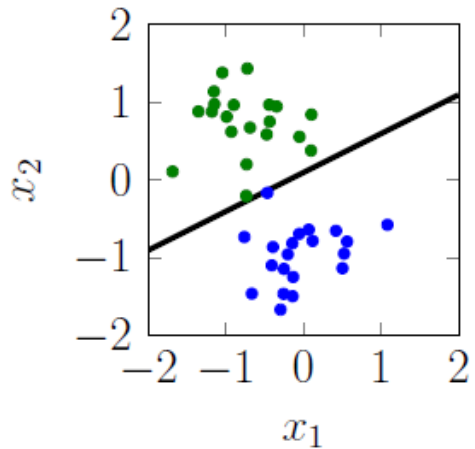 where, if x_n belongs to class k, then the kth bit is 1 and all others are 0.

Objective: Predict the output class for new, unknown inputs $\widehat{x}_m$.

Classification strategies

Linear discriminants (2-class classifiers)

K-class discriminant

Combining 2-class classifiers to obtain multi-class classifiers is a bad idea !!
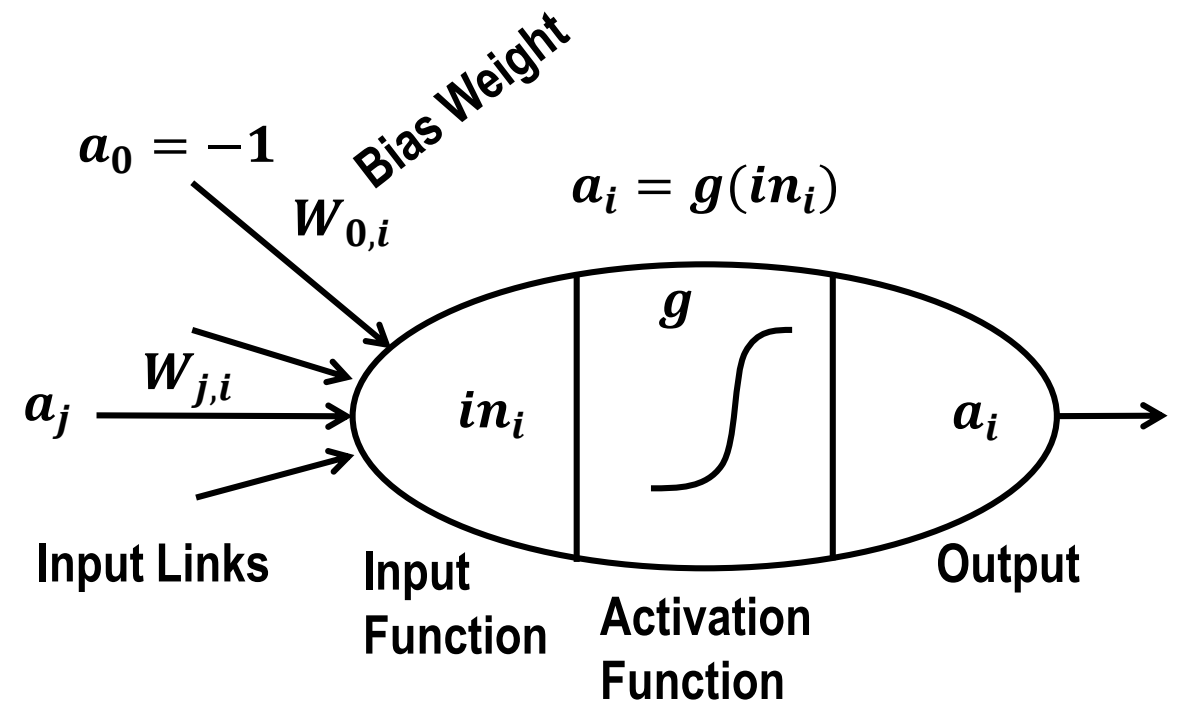
Neural Networks

A neural network consists of a set of nodes (neurons/units) connected by links

- **Each link has a numeric weight**

Each unit has:

- **a set of input links from other units,**
- **a set of output links to other units,**
- **a current activation level, and**
- **an activation function to compute the activation level in the next time step.**

$a_0 = -1$ **Bias Weight**

$a_i = g(in_i)$

$W_{0,i}$

g

$a_j \xrightarrow{W_{j,i}}$ in_i a_i

Input Links   Input Function   Activation Function   Output

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j \qquad a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j \right)$$

Types of Neural Networks

Activation Functions

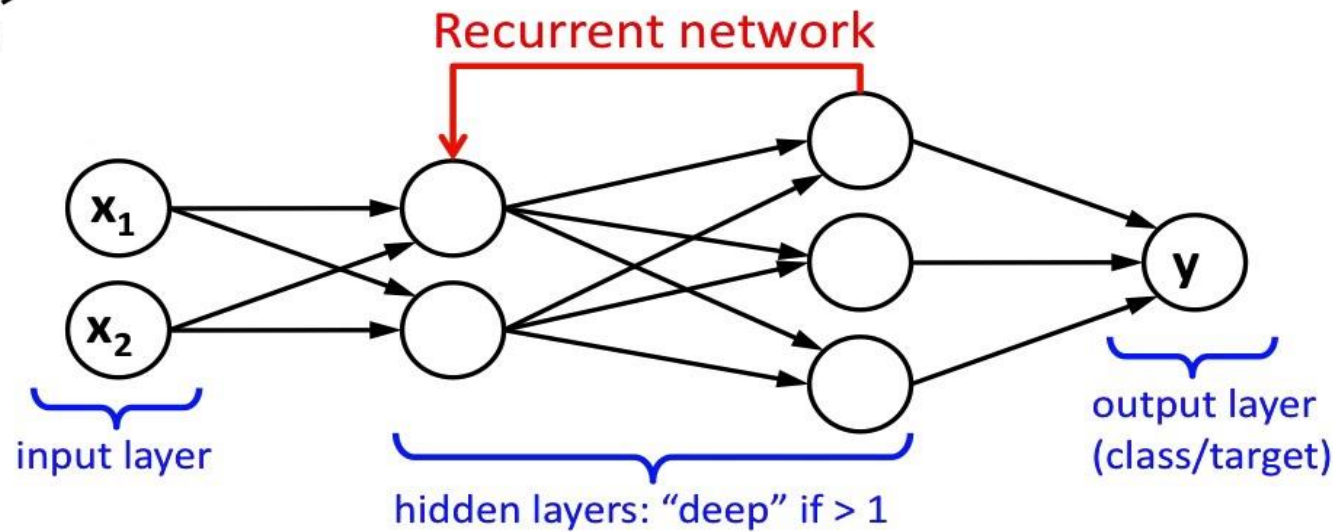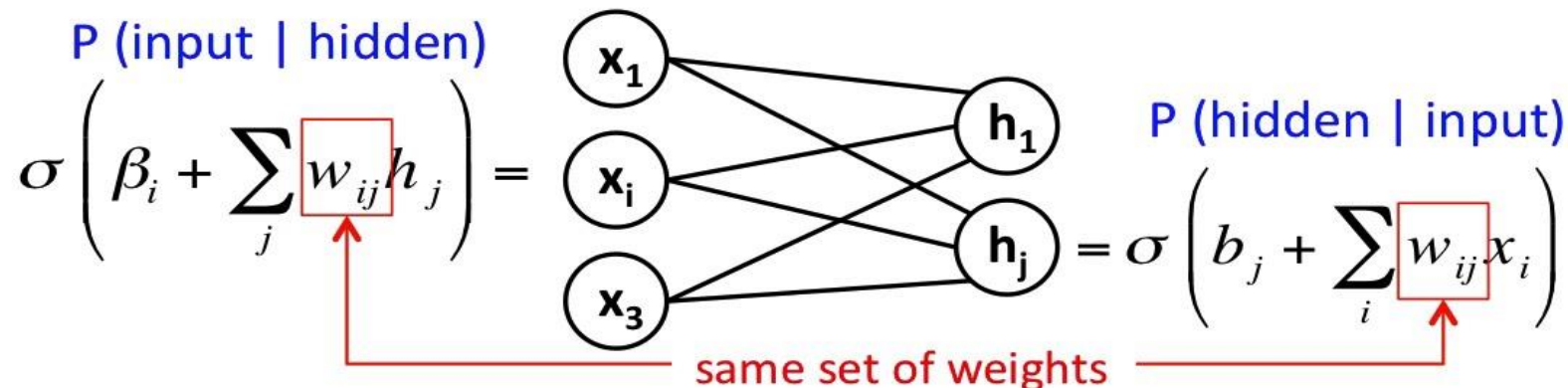| Name	Formula
Identity	$A(x) = x$
Sigmoid	$A(x) = \dfrac{1}{1 + e^{-x}}$
Tanh	$A(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
Step	$A(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

x_1
x_2
x_3

$$y = \sigma\left(b + \sum_i w_{ij} x_i\right)$$
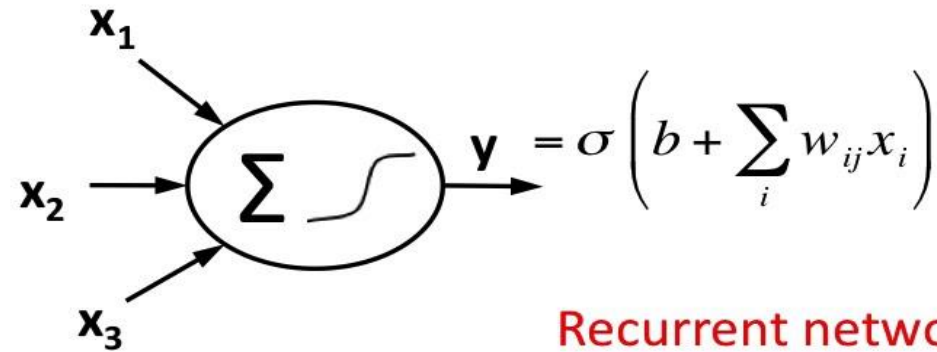
Single neuron: perceptron,

linear / logistic regression

Recurrent network

x_1
x_2

input layer

hidden layers: "deep" if > 1

y

output layer
(class/target)

Feed-forward network
(no cycles) -- non-linear
classification & regression

P (input | hidden)

$$\sigma\left(\beta_i + \sum_j \boxed{w_{ij}} h_j\right) =$$

x_1
x_i
x_3

h_1
h_j

P (hidden | input)

$$= \sigma\left(b_j + \sum_i \boxed{w_{ij}} x_i\right)$$

same set of weights

Symmetric (RBM)
unsupervised, trained
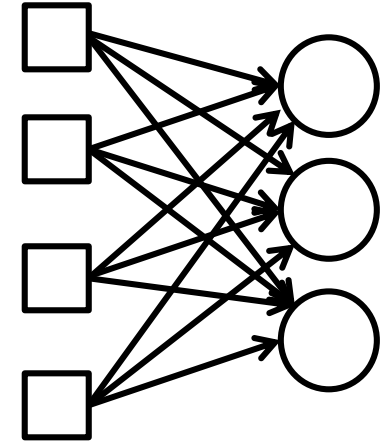to maximize likelihood
of input data

a mixture model

Learning in Single Layered Networks

Idea: Optimize the weights so as to minimize error function:

$$E = \frac{1}{2}Err^2 = \frac{1}{2}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right)^2$$

We can use gradient descent to reduce the squared error by calculating the partial derivative of E with respect to each weight.

$$\frac{\partial E}{\partial W_j}$$

$$= Err \times \frac{\partial Err}{\partial W_j}$$

$$= Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right)$$

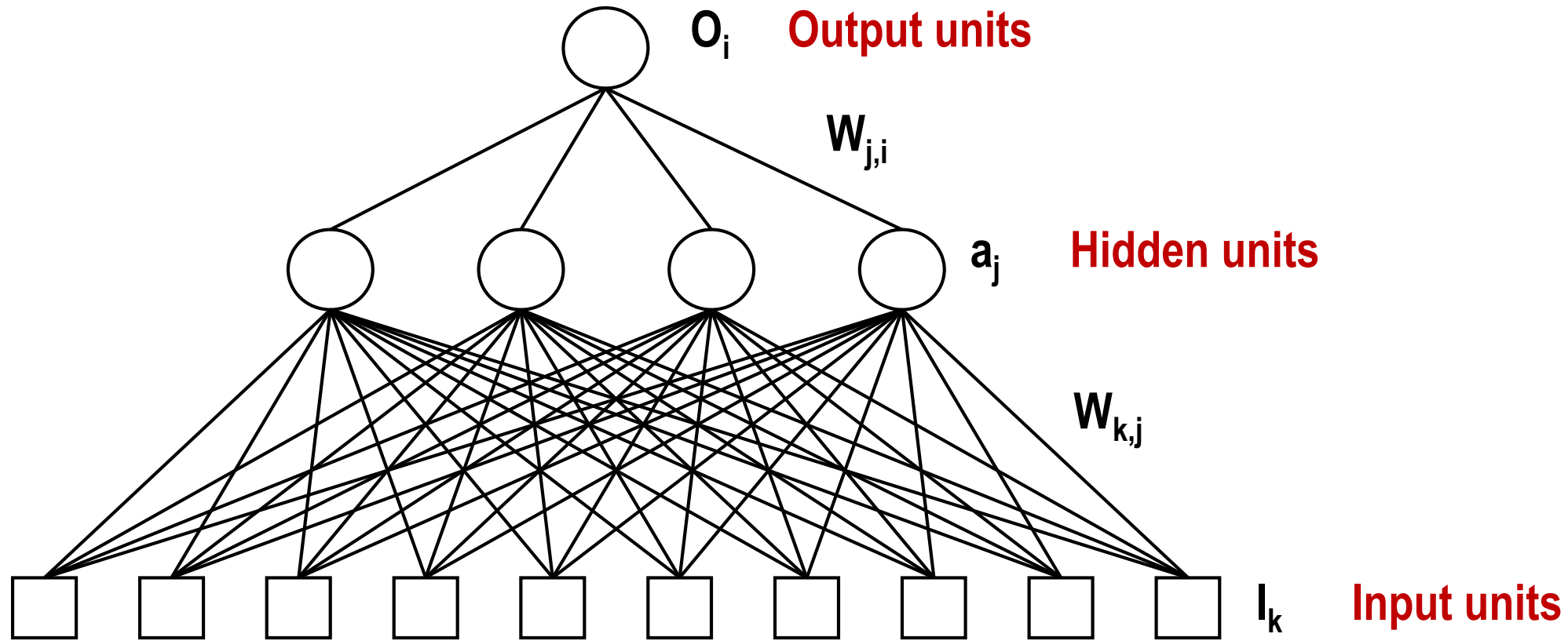$$= -Err \times g'(in) \times x_j$$

Weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

where α is the learning rate

Multi-Layer Feed-Forward Network

Weight updation rule at the output layer: $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$ (same as single layer)

However in multilayer networks, the hidden layers also contribute to the error at the output.

So the important question is: *How do we revise the hidden layers?*

Back-Propagation Learning

- To update the connections between the input units and the hidden units, we need to define a quantity analogous to the error term for output nodes

- We do an error back-propagation, defining error as $\Delta_i = Err_i \times g'(in_i)$

- The idea is that a hidden node *j* is *responsible* for some fraction of the error in each of the output nodes to which it connects

- Thus the Δ_i values are divided according to the strength of the connection between the hidden node and the output node and are propagated back to provide the Δ_j values for the hidden layer.

- The propagation rule for the Δ values is the following:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

- The update rule for the hidden layers is: $W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$

The mathematics behind the updation rule

The squared error on a single example is defined as:
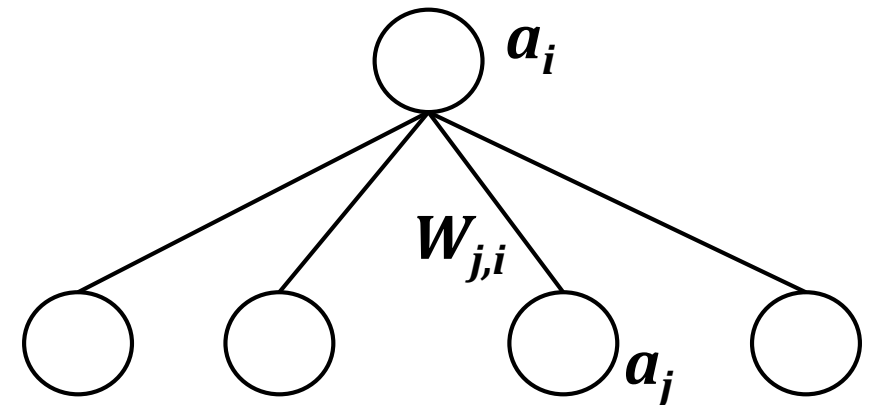
$$E = \frac{1}{2}\Sigma_i(y_i - a_i)^2$$

where the sum is over the nodes in the output layer. To obtain the gradient with respect to a specific weight $W_{j,i}$ in the output layer, we need only expand out the activation a_i as all other terms in the summation are unaffected by $W_{j,i}$

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}}$$

$$= -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$
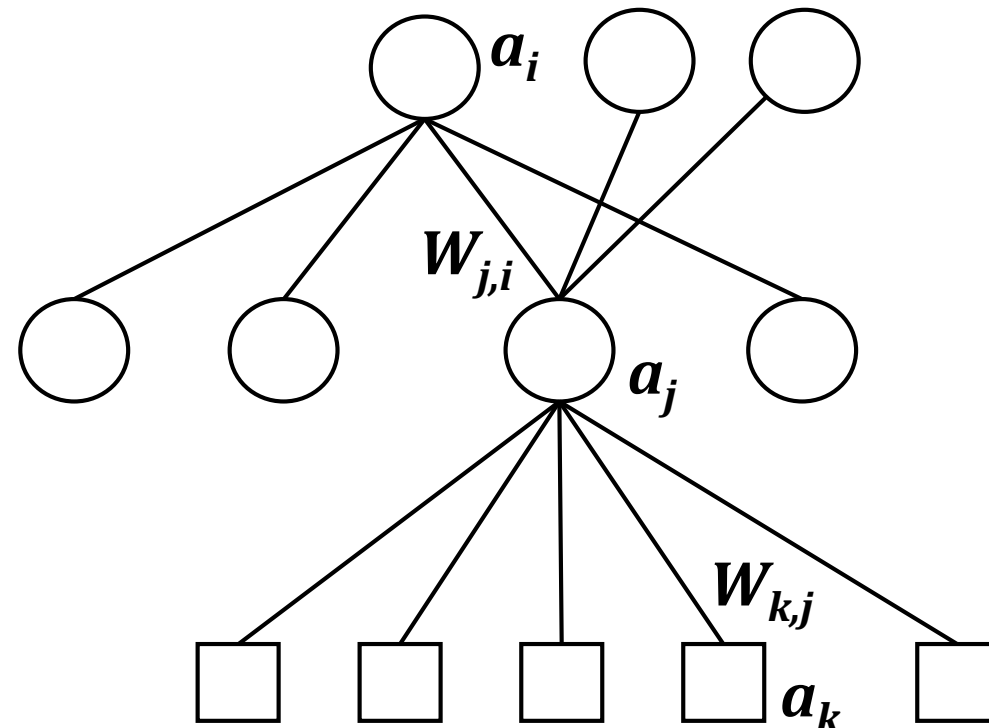
$$= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i$$

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

The mathematics contd.

$$\frac{\partial E}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} = -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right) = -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial in_j}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j}a_k\right)$$

$$= -\sum_i \Delta_i W_{j,i}g'(in_j)a_k = -a_k\Delta_j$$

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

Gradient Descent

- The weight updation rules define a single step of gradient descent

- Each training sample is presented and weights are updated

- This continues, until the training error converges to a (possibly local) minima

- At the end of the learning phase, the network is ready for use (generalization)

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Boltzmann Machines

A Boltzmann machine is a network of units with an *energy* defined for the overall network. Its units produce binary results. The global energy, E, is:

$$E = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

where:

- w_{ij} is the connection strength between unit j and unit i.

- s_i is the state, $s_i \in \{0,1\}$, of unit i

- θ_i is the bias of unit i in the global energy function. ($-\theta_i$ is the activation threshold for the unit)

$$\Delta E_i = \sum_{j>i} w_{ij} s_j + \sum_{j<i} w_{ji} s_j + \theta_i$$

- From this we obtain (the scalar T is called the *temperature*):

$$p_{i=On} = \frac{1}{1 + exp\left(-\frac{\Delta E_i}{T}\right)}$$

Convolutional and Recurrent Neural Networks

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

- **Recurrence is useful for learning sequences**

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

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*.

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

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 $j-n$):

$$s(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n)$$
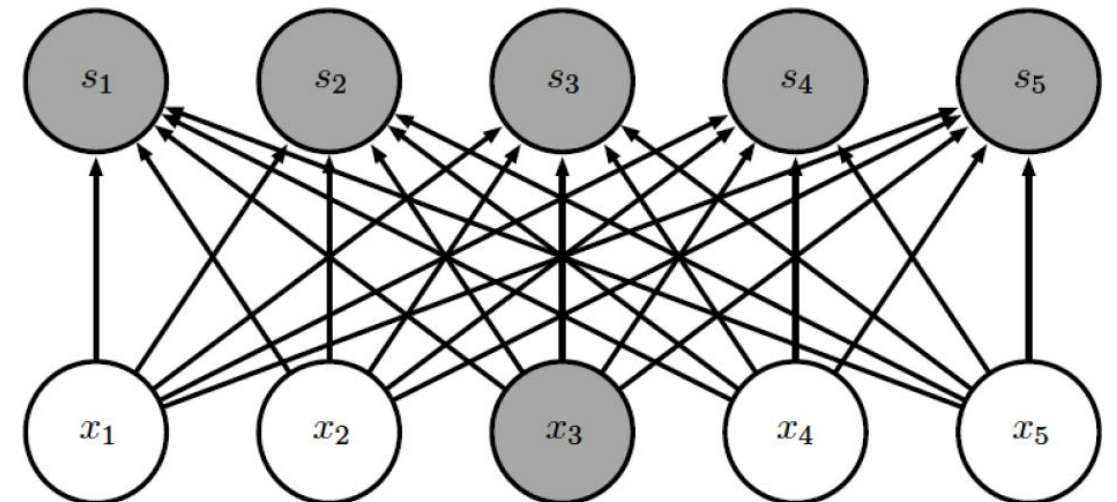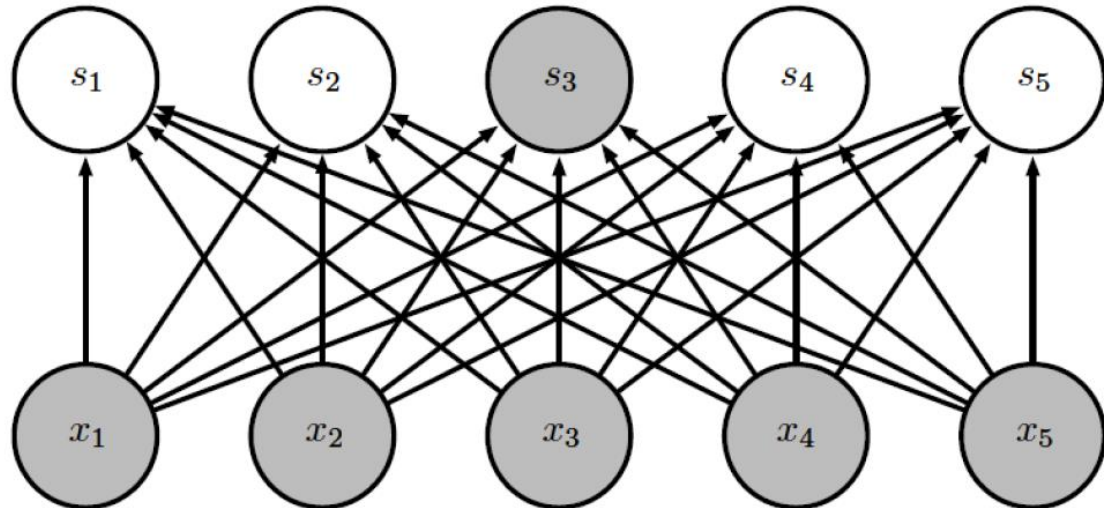
Convolution Networks help us to learn image filters

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

Operation	Filter	Convolved Image
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?**

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

17

If kernel width is small, the network will be sparse

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
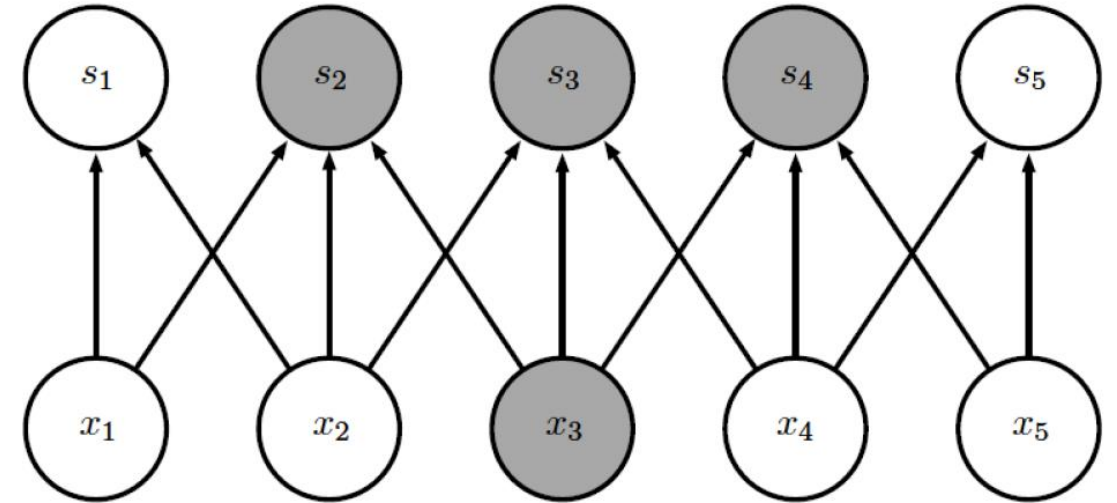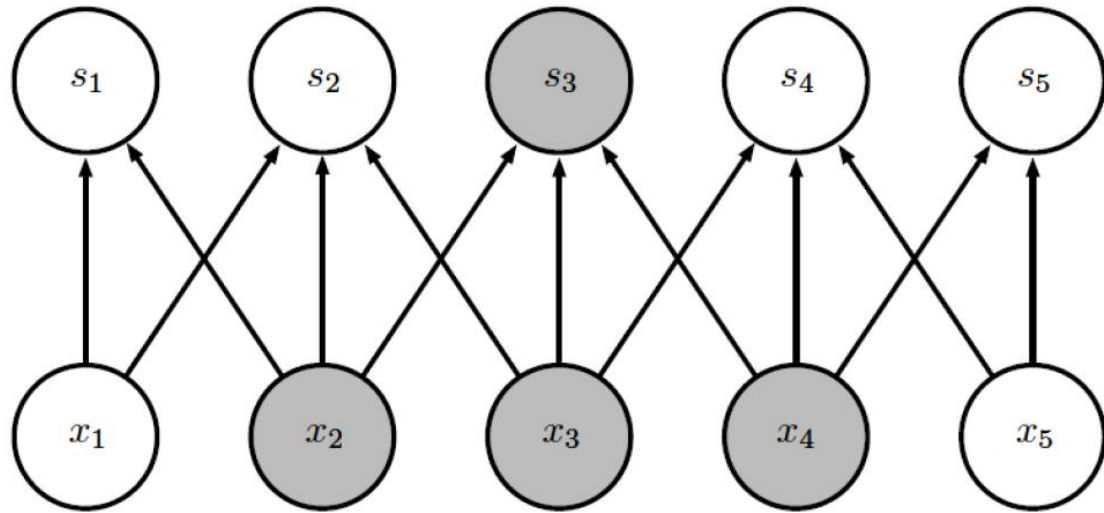
Convolution and Pooling

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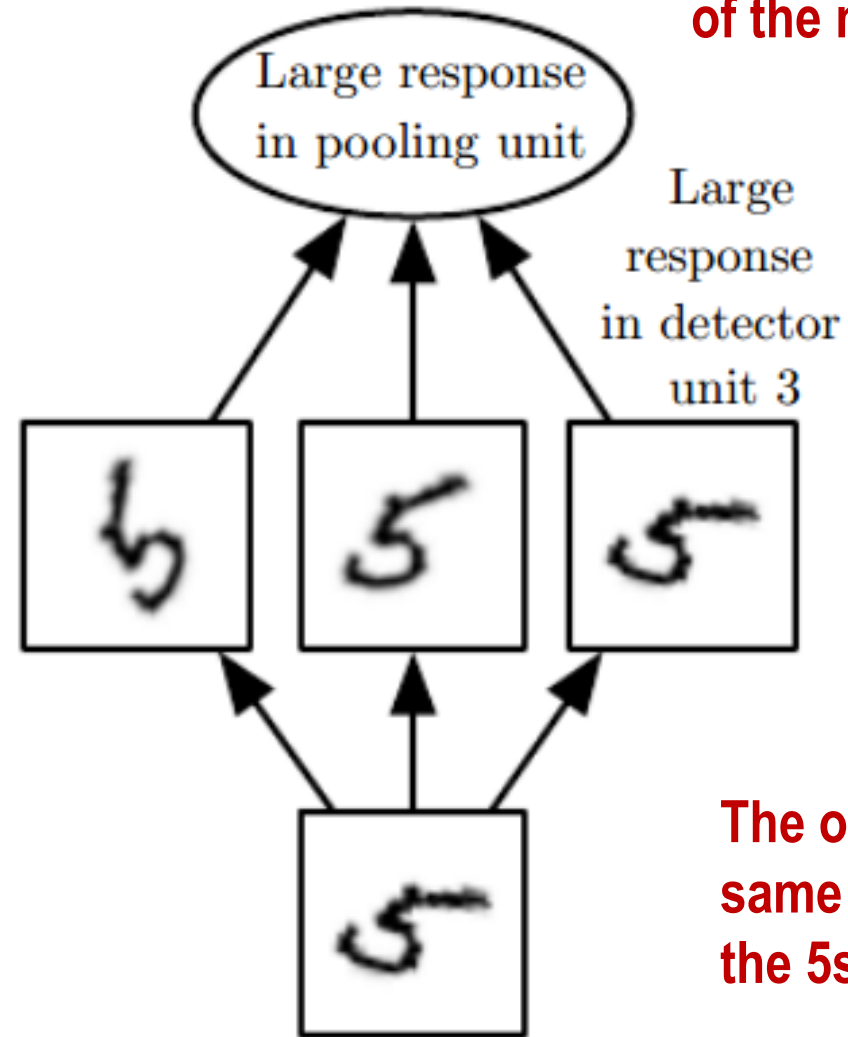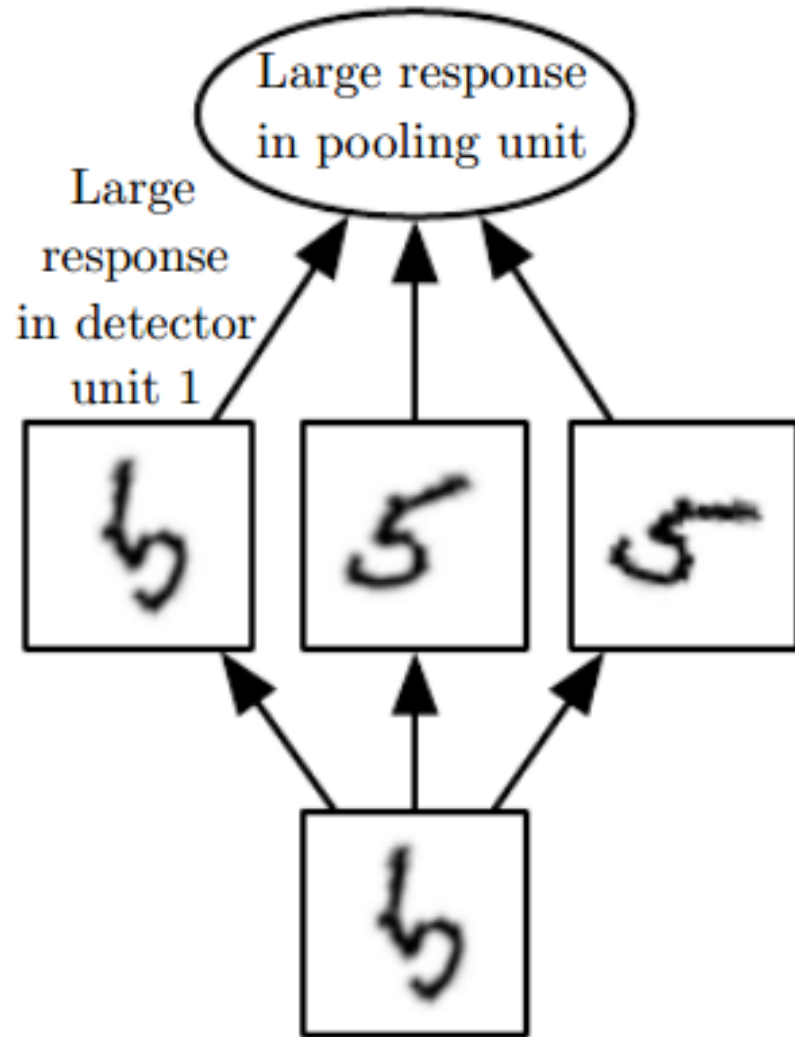
Set of three learned filters

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**

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Unfolding Computation

Consider a dynamical system:

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

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

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

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

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

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

Unfolding computation and Recurrent Network
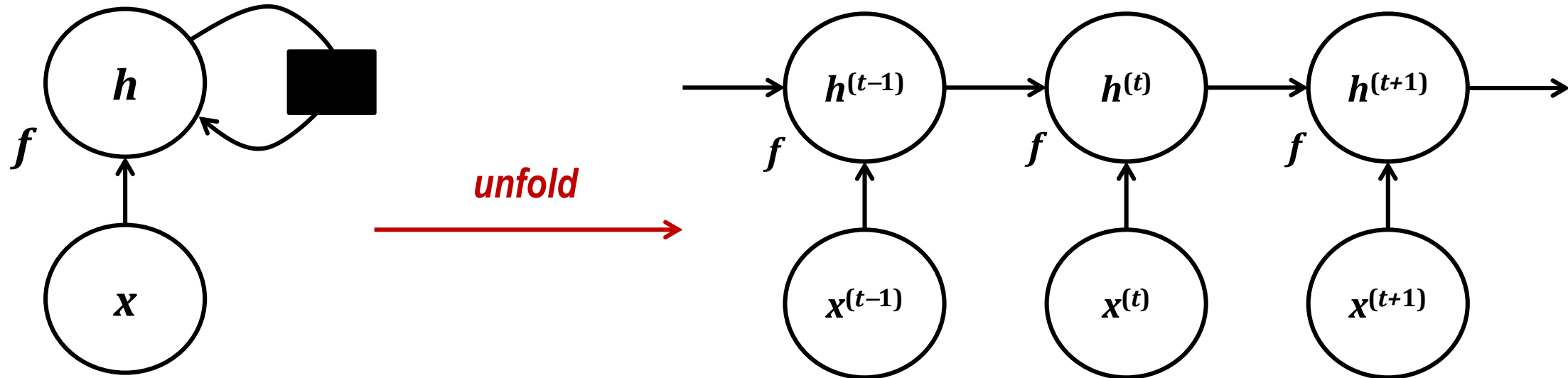
$$h^{(t)} = f\big(h^{(t-1)}, x^{(t)}; \theta\big)$$

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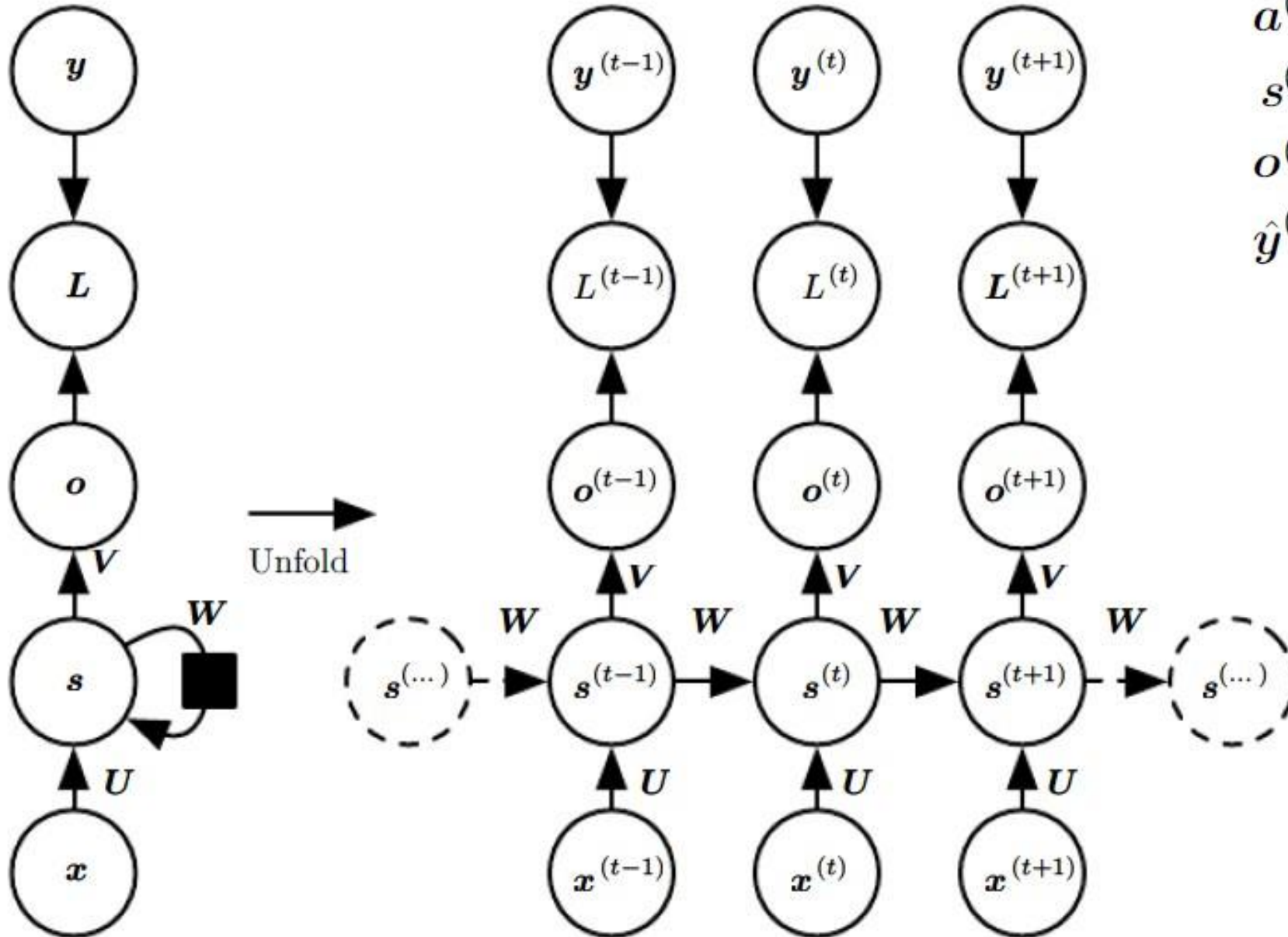
$$
\begin{aligned}
a^{(t)} &= b + W s^{(t-1)} + U x^{(t)} \\
s^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + V s^{(t)} \\
\hat{y}^{(t)} &= \text{softmax}(o^{(t)})
\end{aligned}
$$

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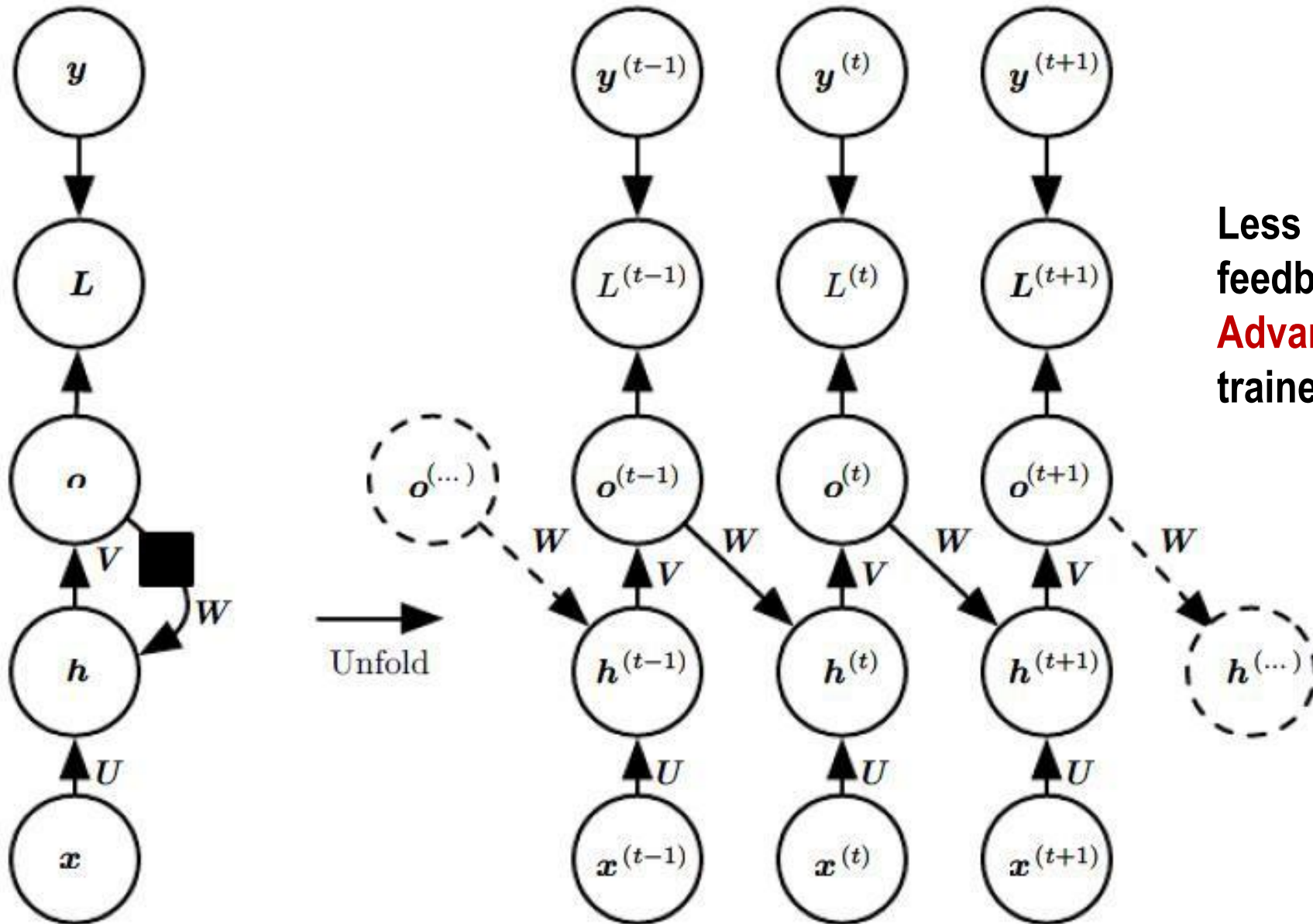
Less powerful than the hidden-hidden feedback model.
Advantage: Each time step can be trained in isolation (why?)

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

Source: DARPA

Training Data → New Learning Process → Explainable Model → Explanation Interface

This is a cat:
- It has fur, whiskers, and claws.
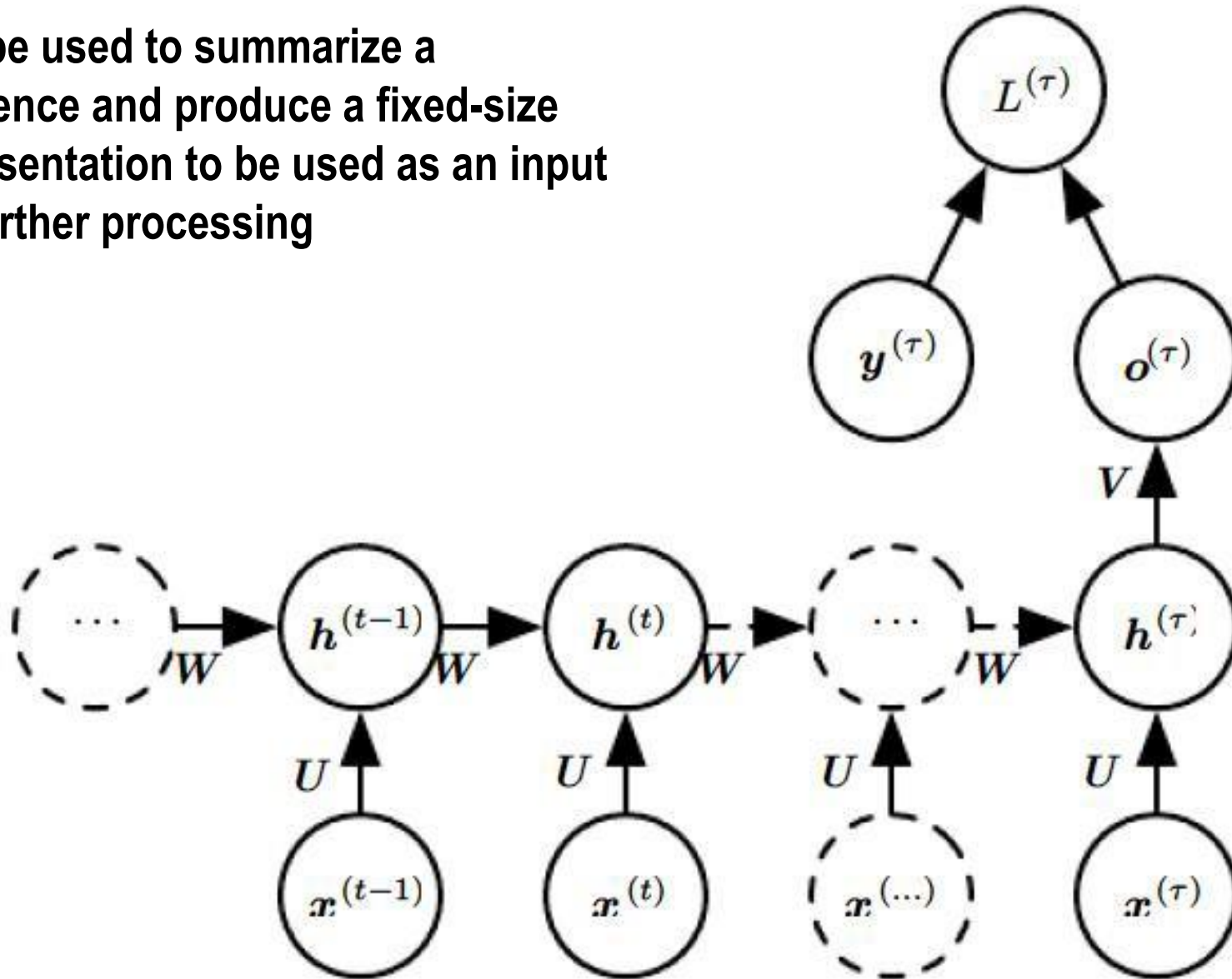- It has this feature:

- I understand why
- I understand why not
- I know when you'll succeed
- I know when you'll fail
- I know when to trust you
- I know why you erred

Deep Explanation	Interpretable Models	Model Induction	HCI	Psychology
Learning Semantic Associations H. Sawhney (SRI Sarnoff)	**Stochastic And-Or-Graphs (AOG)** Song-Chun Zhu (UCLA)	**Local Interpretable Model-agnostic Explanations (LIME)** C. Guestrin (UW)	**Prototype Explanation Interface** T. Kulesza (OSU/MSR)	**Principles of Explanatory Machine Learning** M. Burnett (OSU)
Learning to Generate Explanations T. Darrell, P. Abeel (UCB)	**Bayesian Program Learning** J. Tenenbaum (MIT)	**Bayesian Rule Lists** C. Rudin (MIT)	**UX Design, Language Dialog, Visualization** ENGINEERING PRACTICE	**Psychological Theories of Explanation** T. Lombrozo (UCB)