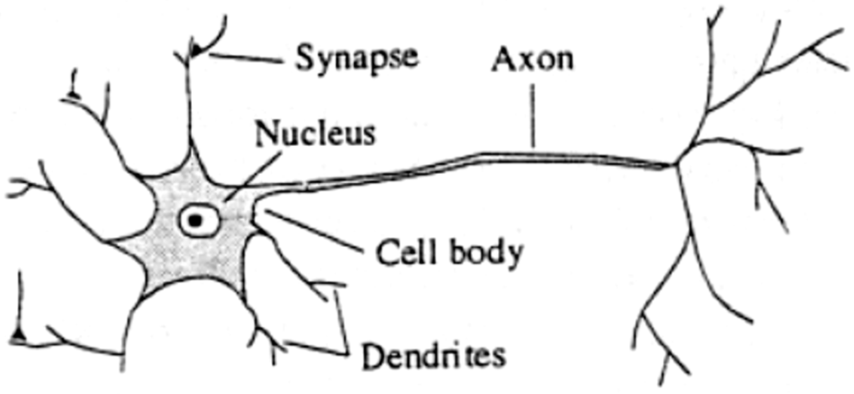
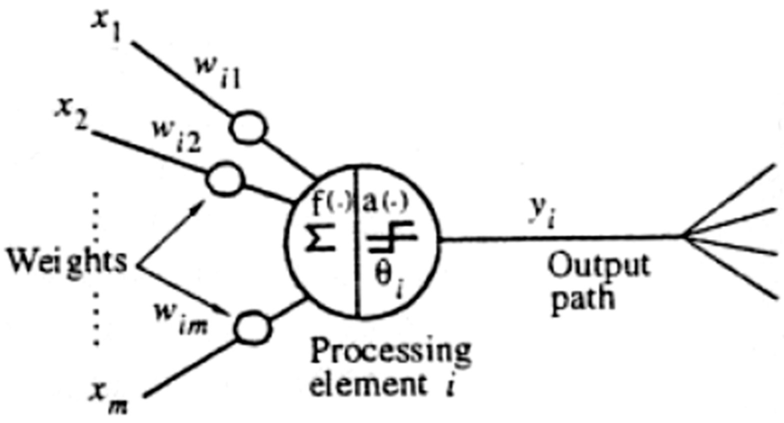
Neural networks

Preface and Neurons

Biological and MP neuron





(a) Biological neuron (b) artificial McCulloch and Pitts neuron [1943]

$$y_i(t+1) = a\left(\sum_{j=1}^{m} w_{ij}x_j(t) - \theta_i\right),$$

$$a(f) = \begin{cases} 1 & \text{if } f \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The weight w_{ij} represents the strength of the synapse (called the *connection* or *link),* connecting neuron j (source) to neuron i (destination).

ANN definition

- **In summary, an ANN is a parallel distributed information processing structure with the following characteristics:**

1. It is a neurally inspired mathematical model.
2. It consists of a large number of highly interconnected processing elements.
3. Its connections (weights) hold the knowledge.
4. A processing element can dynamically respond to its input stimulus, and the response completely depends on its local information; that is, the input signals arrive at the processing element via impinging connections and connection weights.
5. It has the ability to learn, recall, and generalize from training data by assigning or adjusting the connection weights.
6. Its collective behavior demonstrates the computational power, and no single neuron carries specific information *(distributed representation* property).

Processing Elements

- Net input
  - Linear function

$$f_i \triangleq \text{net}_i = \sum_{j=1}^{m} w_{ij} x_j - \theta_i,$$

  - Quadratic function

$$f_i = \sum_{j=1}^{m} w_{ij} x_j^2 - \theta_i,$$

  - Spherical function

$$f_i = \rho^{-2} \sum_{j=1}^{m} (x_j - w_{ij})^2 - \theta_i,$$

  - Polynominal function

$$f_i = \sum_{j=1}^{m} \sum_{k=1}^{m} w_{ijk} x_j x_k + x_j^{\alpha_j} + x_k^{\alpha_k} - \theta_i,$$

Processing Elements

- Activation function (transfer function) 1/2
  - Step function

$$a(f) = \begin{cases} 1 & \text{if } f \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

  - Hard limiter

$$a(f) = \operatorname{sgn}(f) = \begin{cases} 1 & \text{if } f \geq 0 \\ -1 & \text{if } f < 0, \end{cases}$$

  - Ramp function

$$a(f) = \begin{cases} 1 & \text{if } f > 1 \\ f & \text{if } 0 \leq f \leq 1 \\ 0 & \text{if } f < 0, \end{cases}$$

Processing Elements

- Activation function (transfer function) 2/2
  - Unipolar sigmoid function

$$a(f) = \frac{1}{1+e^{-\lambda f}},$$

  - Bipolar sigmoid function

$$a(f) = \frac{2}{1+e^{-\lambda f}} - 1,$$

Activation functions

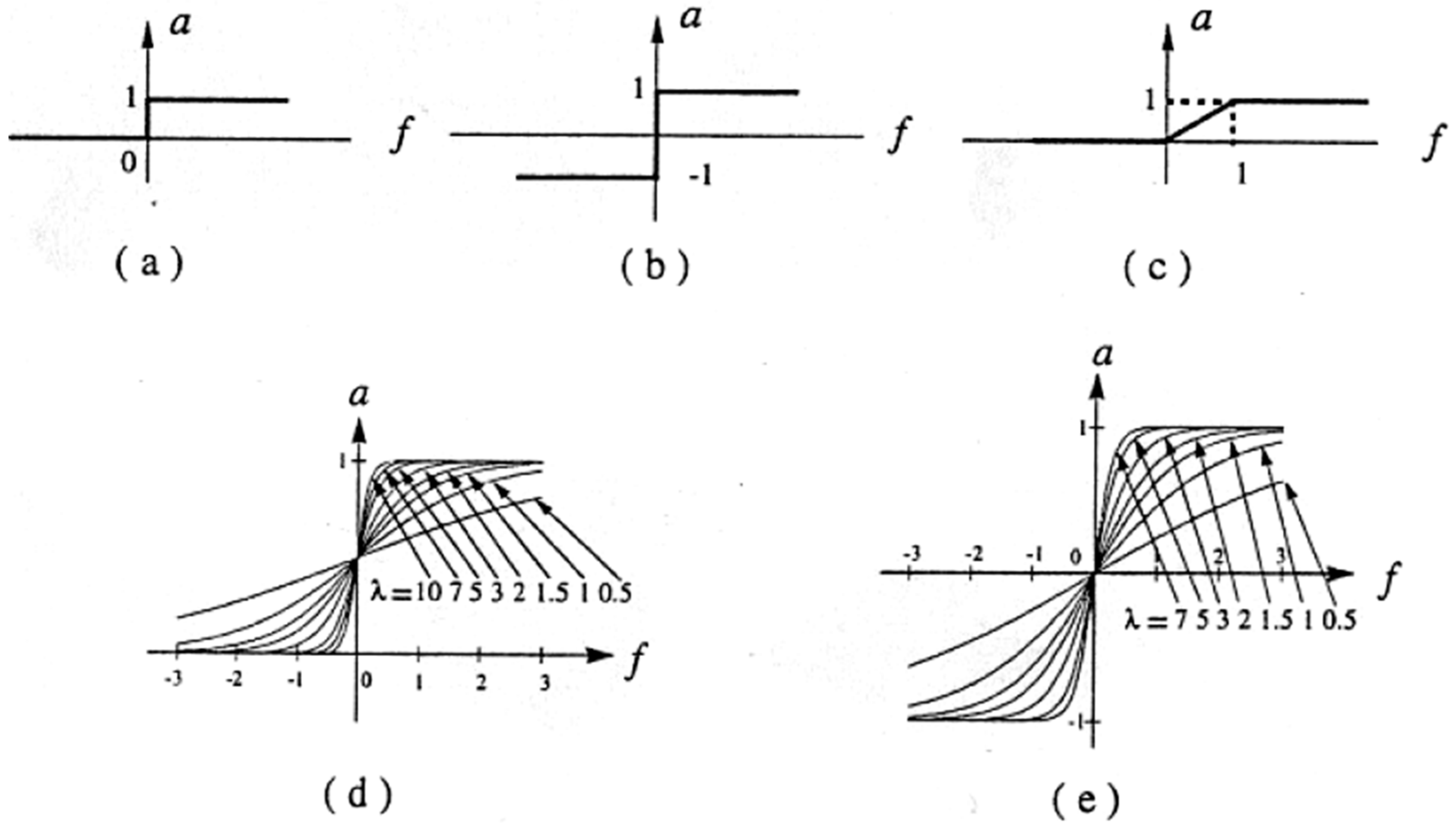
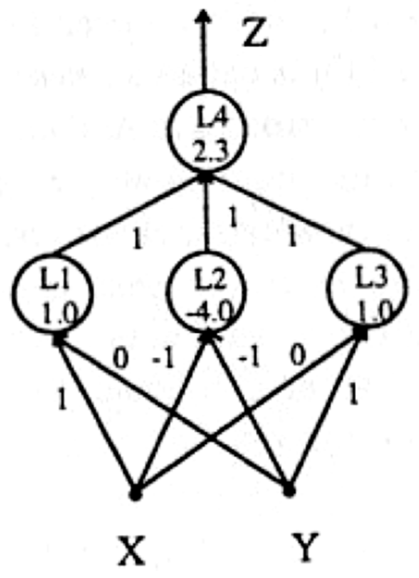
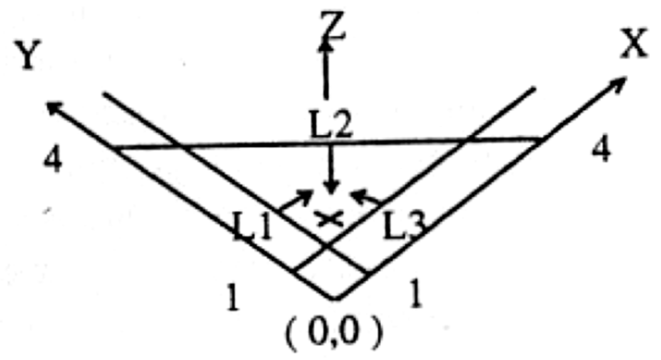


Figure 9.2 Sample activation (transfer) functions. (a) Step function. (b) Hard limiter. (c) Ramp function. (d) Unipolar sigmoid function. (e) Bipolar sigmoid function.

Example

Assume first that each neuron is a LTU (linear treshold unit). The neurons Ll, L2, and L3 correspond to the lines, and the weights and threshold of each neuron define the corresponding line. Each neuron decides on which side of its line the input point lies. When a point in the region interior to the triangle formed by the lines L1, L2, and L3 is inputed, neurons L1, L2, and L3 turn on. Then, since the combined input to neuron L4 is 3.0 and it exceeds its threshold value of 2.3, neuron L4 also turns on. Hence, neuron L4 acts like an AND gate.





Connections

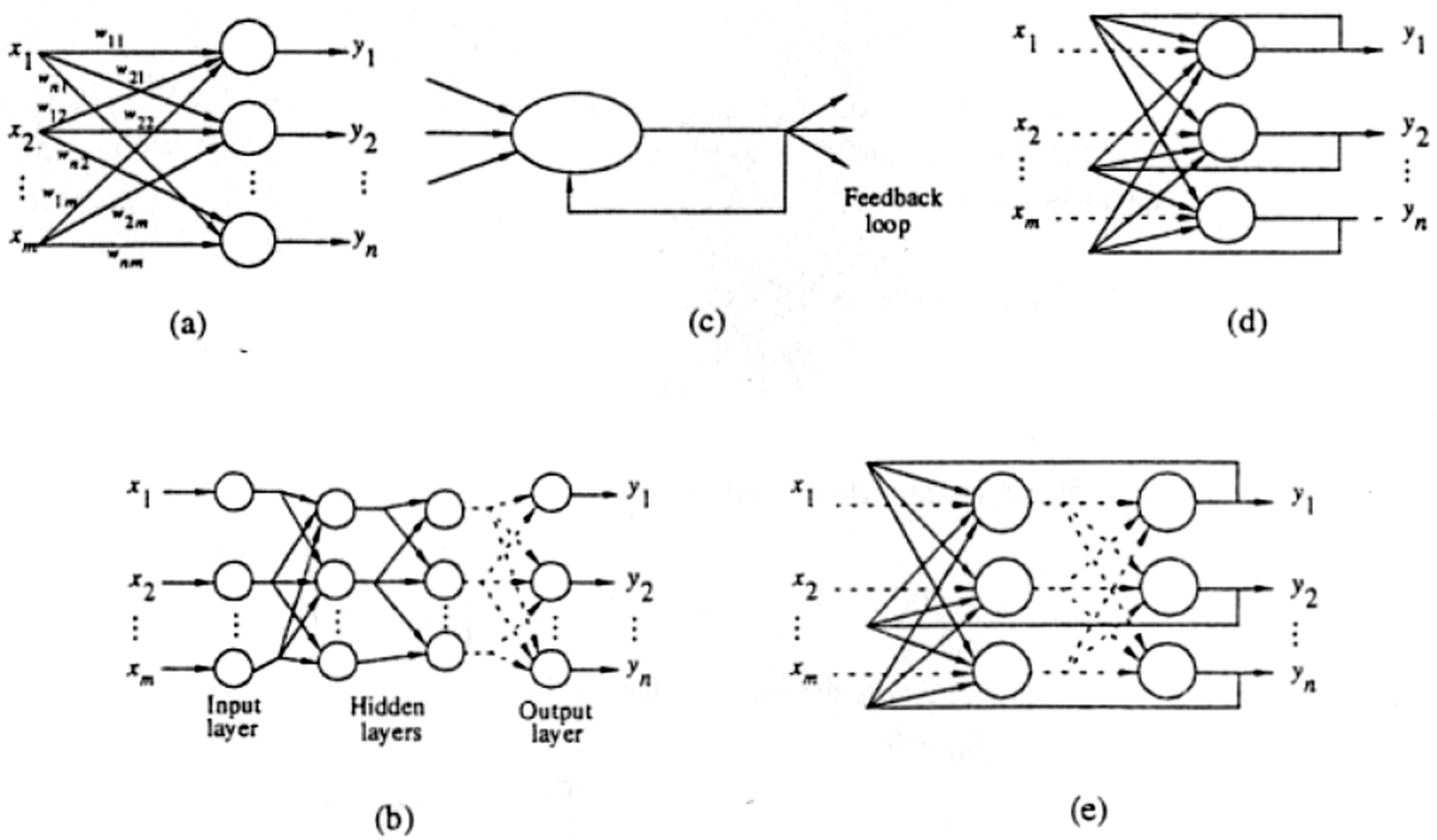


Figure 9.4 Five basic network connection geometries. (a) Single-layer feedforward network. (b) Multilayer feedforward network. (c) Single node with feedback to itself. (d) Single-layer recurrent network. (e) Multilayer recurrent network.

Learning rules

- Parameter learning
- Structure learning
- These two kinds of learning can be performed simultaneously or separately
- Most of the existing learning rules are parameter learning type

Categories of learning

- Supervised learning (learning with a teacher)
- Reinforcement learning (learning with a critic)
- Unsupervised learning

Categories of learning

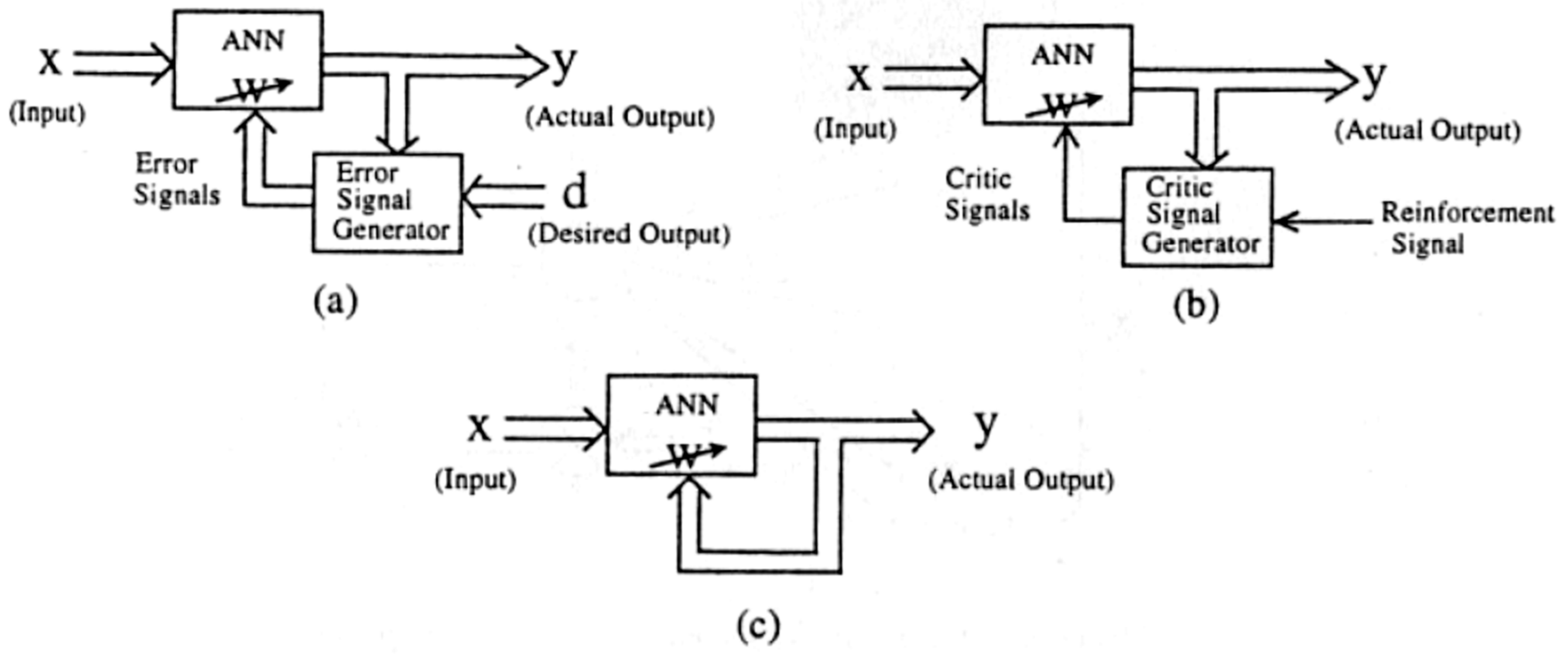
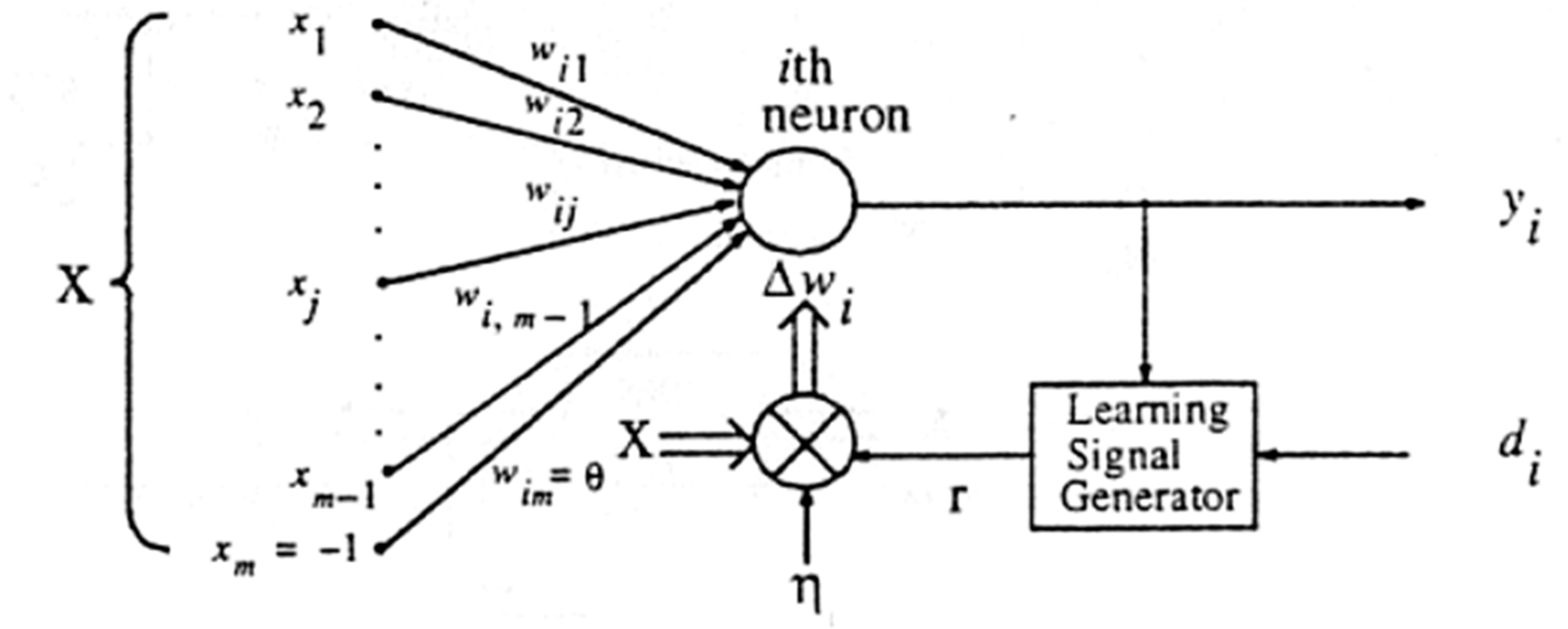


Figure 9.6 Three categories of learning. (a) Supervised learning. (b) Reinforcement learning. (c) Unsupervised learning.

General weight learning rule



The general weight learning rule (d_i is not provided for the unsupervised learning mode)

General weight learning rule

$$\Delta \mathbf{w}_i(t) = \eta r \mathbf{x}(t),$$

- r – learning signal
- η – learning constant

$$r = f_r(\mathbf{w}_i, \mathbf{x}, d_i).$$

$$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} + \eta f_r\left(\mathbf{w}_i^{(t)}, \mathbf{x}^{(t)}, d_i^{(t)}\right)\mathbf{x}^{(t)},$$

Hebb's learning law

- Hebb [1949] hypothesized that when an axonal input from neuron A to neuron B causes neuron B to immediately emit a pulse (fire) and this situation happens repeatedly or persistently, then the efficacy of that axonal input, in terms of its ability to help neuron B to fire in the future, is **somehow increased**.
- Hence, he suggested that synaptic strengths in the brain change proportionally to the correlation between the firing of the pre- and postsynaptic neurons

Hebb's learning law

- The weights are adjusted according to the pre- and postcorrelations, the learning signal r in the **general weight learning rule** is set as:

$$r = a\left(\mathbf{w}_i^T \mathbf{x}\right) = y_i$$

- Where $a(.)$ is activation function of PE. Hence, in the Hebbian learning rule, the learning signal r is simply set as the PE's current output.

$$\Delta \mathbf{w}_i = \eta a\left(\mathbf{w}_i^T \mathbf{x}\right)\mathbf{x} = \eta y_i \mathbf{x}$$

$$\Delta w_{ij} = \eta a\left(\mathbf{w}_i^T \mathbf{x}\right)x_j = \eta y_i\, x_j \quad i = 1,2,\ldots,n; j = 1,2,\ldots m$$

Hebb's learning law

- The Hebbian learning rule is an **unsupervised learning rule** for a feedforward network since it uses only the product of inputs and actual outputs to modify the weights
- No desired outputs are given to generate the learning signal to update the weights
- This learning rule requires **weight initialization** at small random values around zero before learning.
- Equation indicates that if the input-output correlation term $y_i x_j$ is positive, the weight w_{ij} will increase; otherwise the weight w_{ij} decrease. Furthermore, since the output is strengthened in turn for each input presented, frequently occurring input patterns will have the most influence on the weights and eventually produce the largest output

Example

- Consider the Hebbian learning rule for an ANN with a single PE which is a LTU. There are four inputs, x_1, x_2, x_3 and x_4 to this PE. The corresponding weight vector is $\boldsymbol{w} = (w_1, w_2, w_3, w_4)^T$

$$\boldsymbol{x}^{(1)} = \begin{pmatrix} 1 \\ 1.5 \\ 0.5 \\ 0 \end{pmatrix}, \quad \boldsymbol{x}^{(2)} = \begin{pmatrix} -0.5 \\ 1 \\ 0 \\ 1.5 \end{pmatrix}, \quad \boldsymbol{x}^{(3)} = \begin{pmatrix} -1 \\ 0 \\ -1 \\ -0.5 \end{pmatrix}$$

$$\boldsymbol{w}^{(1)} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

Example

- Step1

$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + sgn\left(\left(\boldsymbol{w}^{(1)}\right)^T \boldsymbol{x}^{(1)}\right)\boldsymbol{x}^{(1)} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix} + sgn(0.5)\begin{pmatrix} 1 \\ 1.5 \\ 0.5 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1.5 \\ -0.5 \\ 0 \end{pmatrix}$$

- Step2

$$\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(2)} + sgn\left(\left(\boldsymbol{w}^{(2)}\right)^T \boldsymbol{x}^{(2)}\right)\boldsymbol{x}^{(2)} = \begin{pmatrix} 2 \\ 1.5 \\ -0.5 \\ 0 \end{pmatrix} + sgn(0.5)\begin{pmatrix} -0.5 \\ 1 \\ 0 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 2.5 \\ -0.5 \\ 1.5 \end{pmatrix}$$

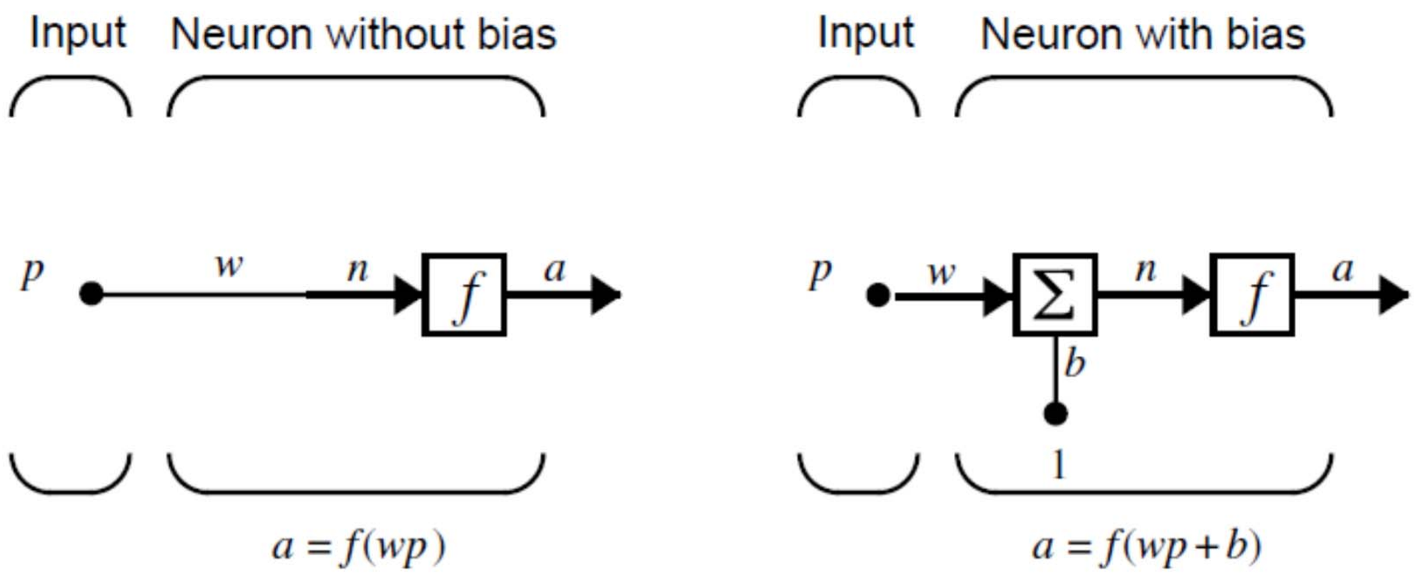
Example

- Step3

$$\boldsymbol{w}^{(4)} = \boldsymbol{w}^{(3)} + sgn\left(\left(\boldsymbol{w}^{(3)}\right)^T \boldsymbol{x}^{(3)}\right)\boldsymbol{x}^{(3)} = \begin{pmatrix} 1.5 \\ 2.5 \\ -0.5 \\ 1.5 \end{pmatrix} + sgn(-1.75)\begin{pmatrix} -1 \\ 0 \\ -1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 2.5 \\ 0.5 \\ 2 \end{pmatrix}$$

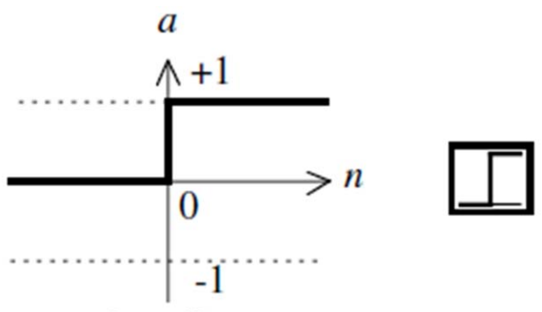
- It can be verified that $sgn\left(\left(\boldsymbol{w}^{(4)}\right)^T \boldsymbol{x}^{(1)}\right) = sgn\left(\left(\boldsymbol{w}^{(4)}\right)^T \boldsymbol{x}^{(2)}\right)$=1 and $sgn\left(\left(\boldsymbol{w}^{(4)}\right)^T \boldsymbol{x}^{(3)}\right) = -1$. This means that the inputs $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ that caused the PE to fire previously will cause it to fire again in the future with the final learned weights. This is also true for the input $\boldsymbol{x}^{(3)}$ which inhibits the firing of the PE. Thus, the final weights have captured the coincidence relationship of input-output training pairs.

Simple neuron



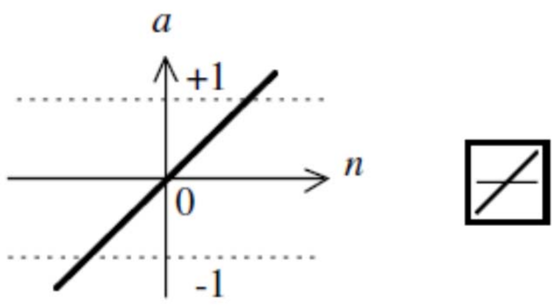
- *p* – input
- *w* – weight
- *b* – bias
- *a* – output
- *f* – transfer function

Tranfer functions



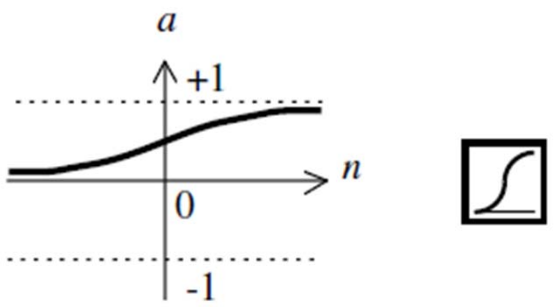
$a = hardlim(n)$

Hard-Limit Transfer Function



$a = purelin(n)$

Linear Transfer Function

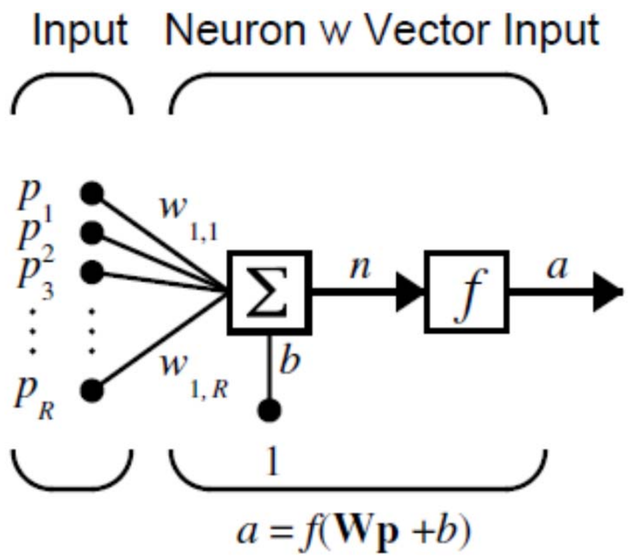


$a = logsig(n)$

Log-Sigmoid Transfer Function

```
n = -5:0.1:5;
plot(n,hardlim(n),'c+:');
```

Neuron with Vector limit



Where

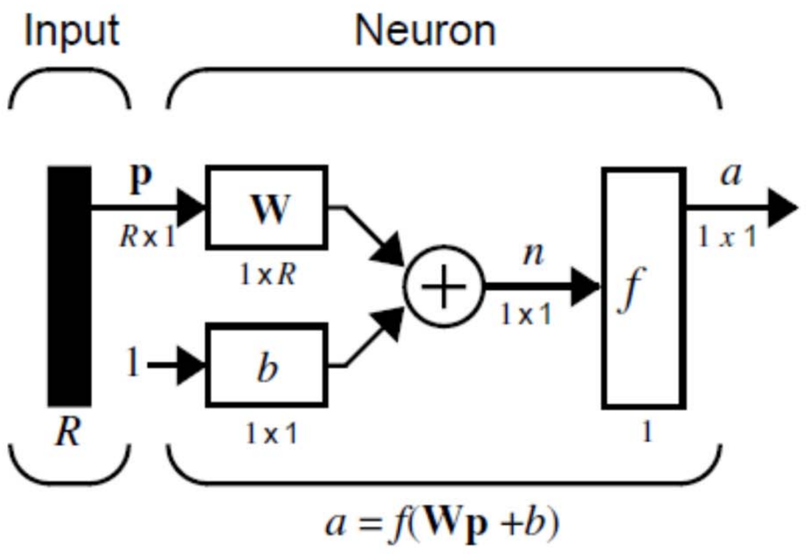
R = number of elements in input vector

- p_1, p_R – inputs
- $w_{1,1}$, $w_{1,R}$ – weights

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

```
n = W*p + b
```

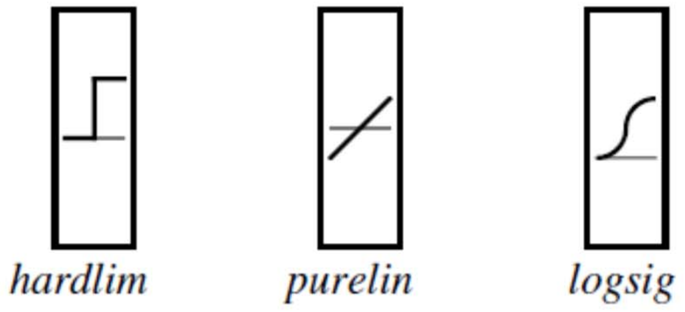
Neuron with Vector limit



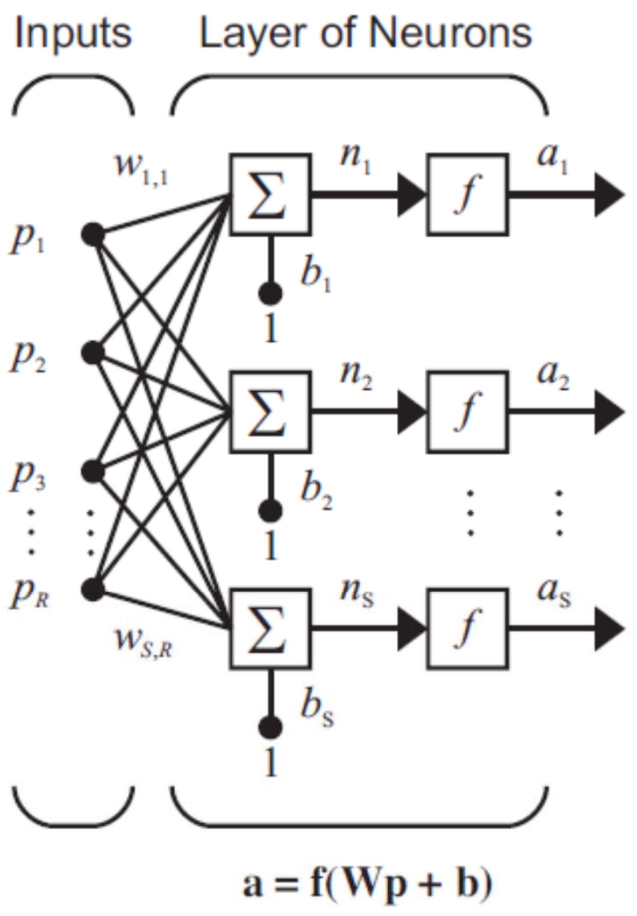
Where...

R = number of elements in input vector

Transfer functions



A Layer of neurons



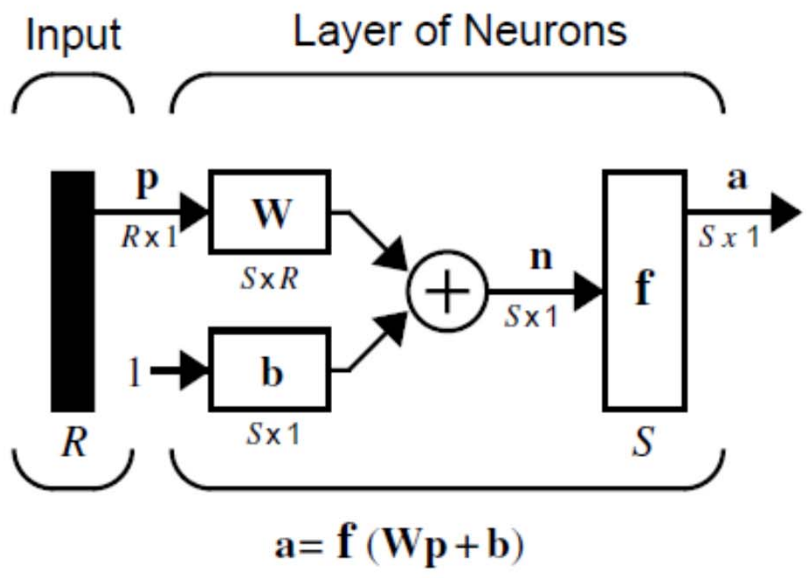
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ & & & \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Where

R = number of elements in input vector

S = number of neurons in layer

A Layer of neurons

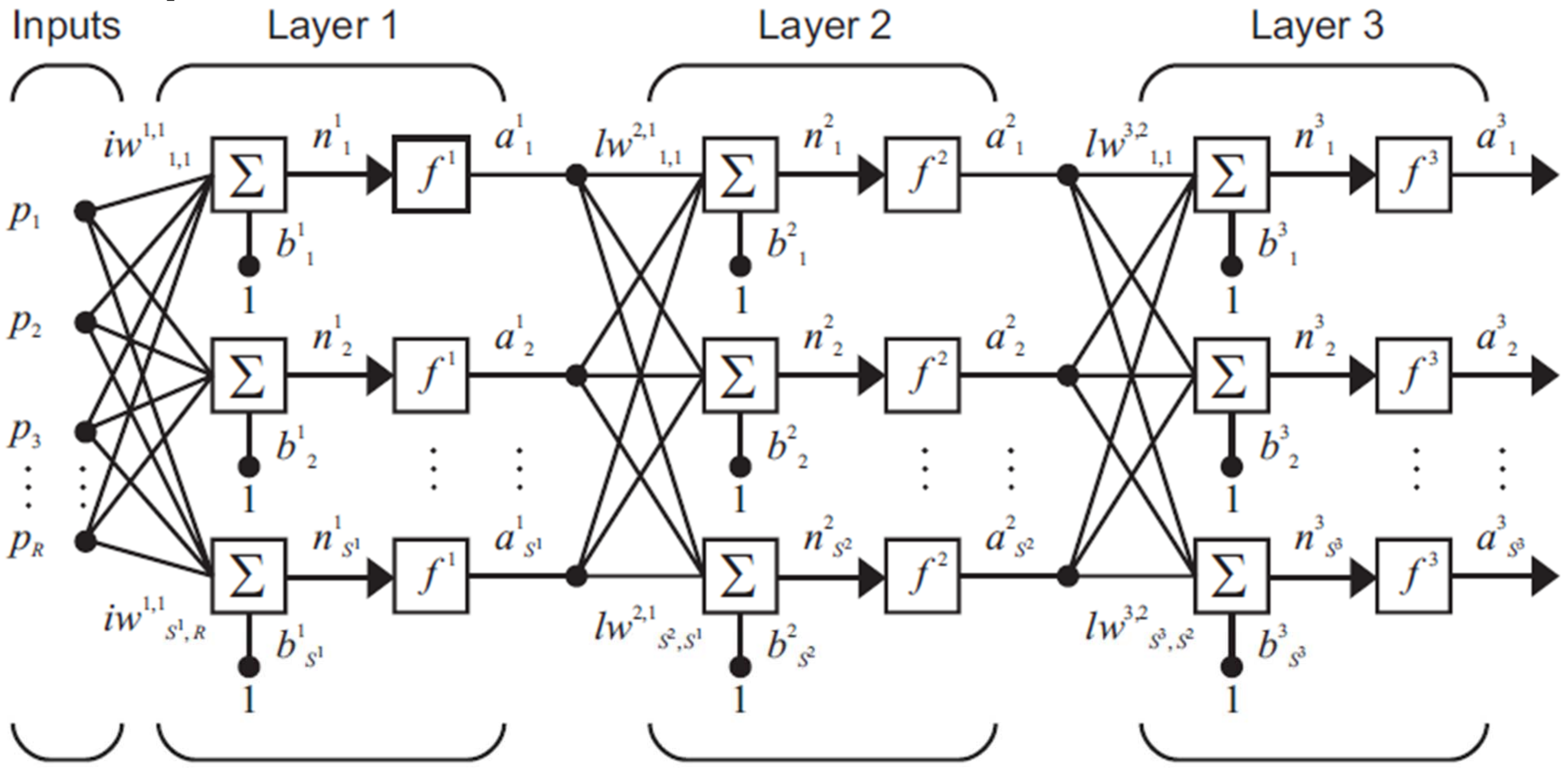


Where...

R = number of elements in input vector

S = number of neurons in layer 1

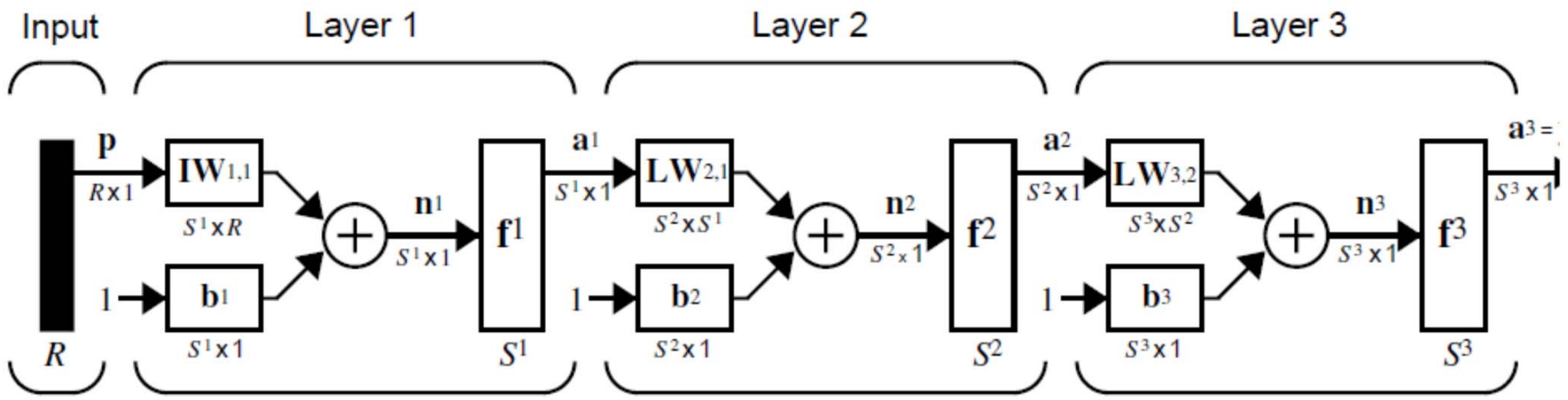
Multiple layer of neurons



$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1)$ $\quad$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1+\mathbf{b}^2)$ $\quad$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2+\mathbf{b}^3)$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

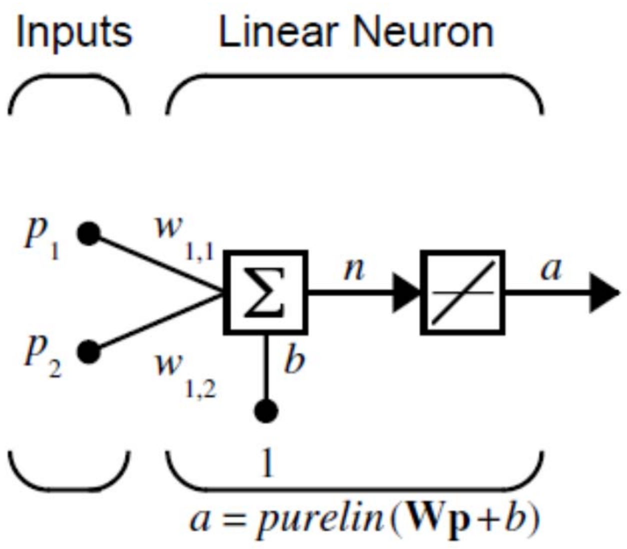
Multiple layer of neurons



$a^1 = f^1(IW^{1,1}p + b^1)$ $a^2 = f^2(LW^{2,1} a^1 + b^2)$ $a^3 = f^3(LW^{3,2} a^2 + b^3)$

$$a^3 = f^3(LW^{3,2} f^2(LW^{2,1} f^1(IW^{1,1} p + b^1) + b^2) + b^3 = y$$

Simulation with Concurrent Inputs in a Static Network



- NEWLIN(P,S,ID,LR) takes these arguments,
  - P - RxQ matrix of Q representative input vectors.
  - S - Number of elements in the output vector.
  - ID - Input delay vector, default=[0]
  - LR - Learning rate, default = 0.01;
- and returns a new linear layer.

```
o P=[-2 -1 0 1 2 3; -3 -2 0 2 3 5];
o Net=newlin(P,1)

o net.inputs{1}.inputs - 2 inputs
o net.inputs{1}.range
    o First input range(-2,3)
    o Second input range(-3,5)
```

Simulation with Concurrent Inputs in a Static Network

$\mathbf{W} = \begin{bmatrix} 1 & 2 \end{bmatrix}$ and $b = \begin{bmatrix} 0 \end{bmatrix}$

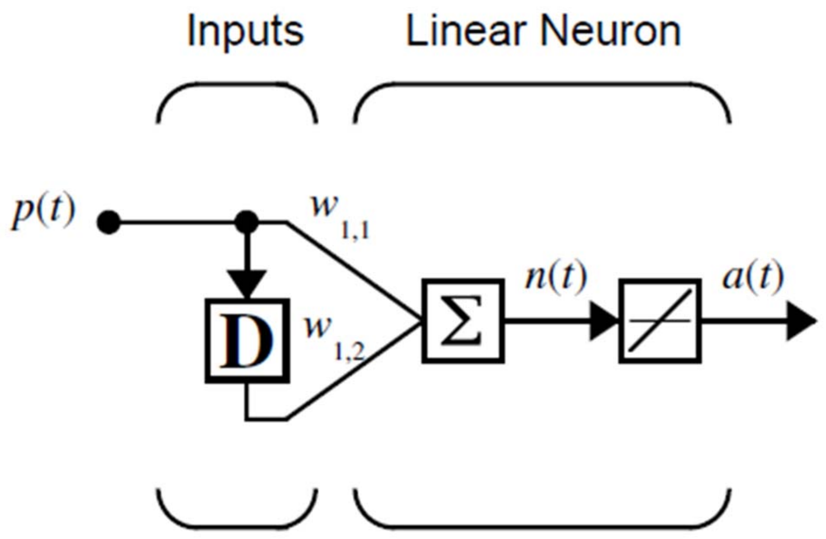
```
net.IW{1,1} = [1 2];
net.b{1} = 0;
```

$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{p}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \mathbf{p}_3 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \mathbf{p}_4 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

```
P = [1 2 2 3; 2 1 3 1];
```

```
A = sim(net,P)
A =
     5     4     8     5
```

Simulation with Sequential Inputs in a Dynamic Network



$$a(t) = w_{1,1}p(t) + w_{1,2}p(t-1)$$

```
net = newlin([-1 1],1,[0 1]);
net.biasConnect = 0;
```

Simulation with Sequential Inputs in a Dynamic Network

$\mathbf{W} = \begin{bmatrix} 1 & 2 \end{bmatrix}$

```
net.IW{1,1} = [1 2];
```

Suppose that the input sequence is

$\mathbf{p}1 = \begin{bmatrix} 1 \end{bmatrix}, \mathbf{p}2 = \begin{bmatrix} 2 \end{bmatrix}, \mathbf{p}3 = \begin{bmatrix} 3 \end{bmatrix}, \mathbf{p}4 = \begin{bmatrix} 4 \end{bmatrix}$

```
P = {1 2 3 4};
```

```
A = sim(net,P)
A =
    [1]     [4]     [7]     [10]
```

Simulation with Concurrent Inputs in a Dynamic Network

- Concurrent set of inputs

$$\mathbf{p}_1 = \begin{bmatrix}1\end{bmatrix}, \quad \mathbf{p}_2 = \begin{bmatrix}2\end{bmatrix}, \quad \mathbf{p}_3 = \begin{bmatrix}3\end{bmatrix}, \quad \mathbf{p}_4 = \begin{bmatrix}4\end{bmatrix}$$

```
P = [1 2 3 4];
```

```
A = sim(net,P)
A =
       1        2        3        4
```

- Two input sequences

$$\mathbf{p}_1(1) = \begin{bmatrix}1\end{bmatrix}, \mathbf{p}_1(2) = \begin{bmatrix}2\end{bmatrix}, \mathbf{p}_1(3) = \begin{bmatrix}3\end{bmatrix}, \mathbf{p}_1(4) = \begin{bmatrix}4\end{bmatrix}$$

$$\mathbf{p}_2(1) = \begin{bmatrix}4\end{bmatrix}, \mathbf{p}_2(2) = \begin{bmatrix}3\end{bmatrix}, \mathbf{p}_2(3) = \begin{bmatrix}2\end{bmatrix}, \mathbf{p}_2(4) = \begin{bmatrix}1\end{bmatrix}$$

```
P = {[1 4] [2 3] [3 2] [4 1]};
```

```
A = sim(net,P);

A = {[1 4] [4 11] [7 8] [10 5]}
```