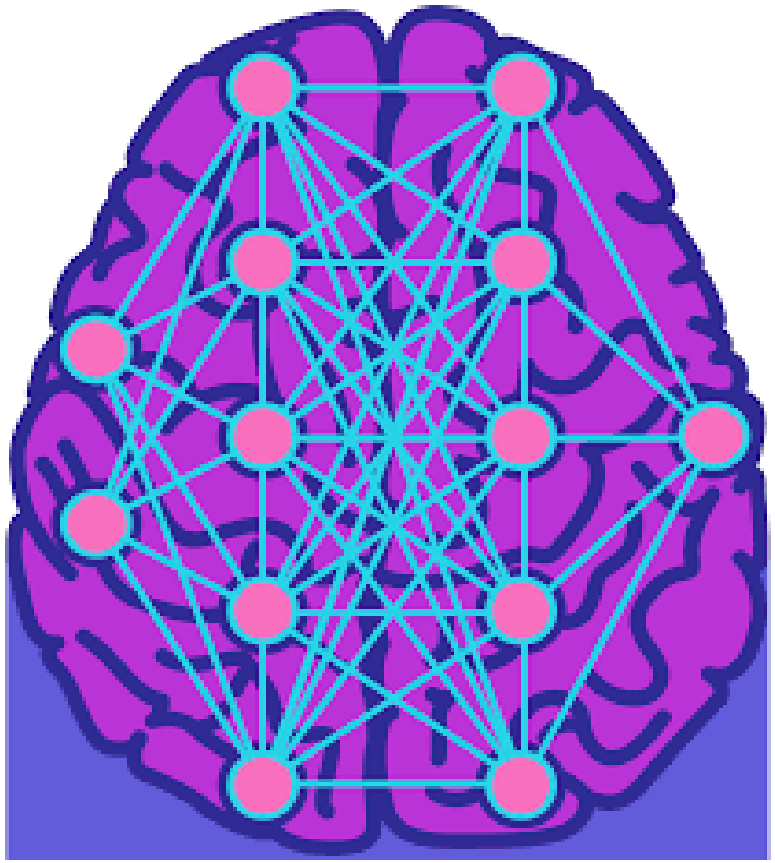
ITF309

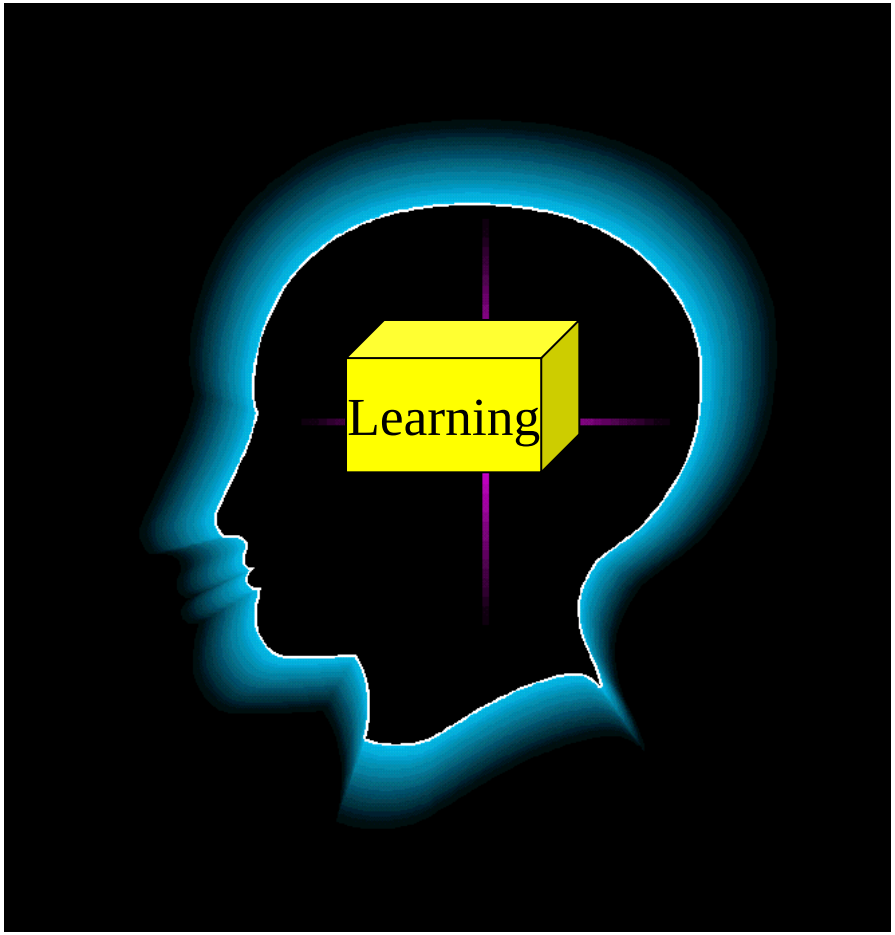
Artificial Neural Networks



Outline

- **Neural models and neural network architectures**
  - Learning Processes
  - Neural model
    - Single-input neuron
      - Transfer functions
    - Multiple-input neurons
  - Network architecture

Learning Processes

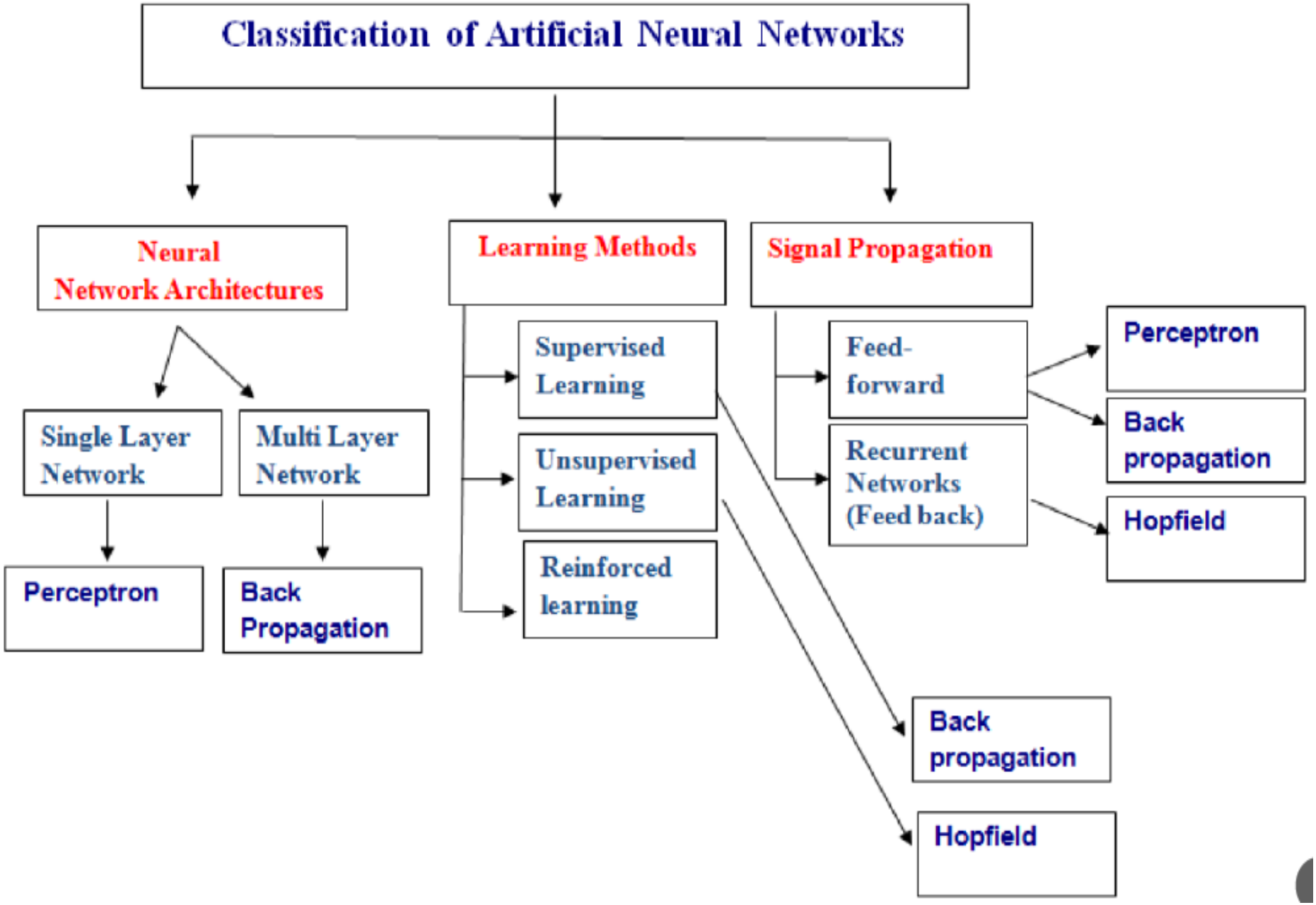


What is Learning?

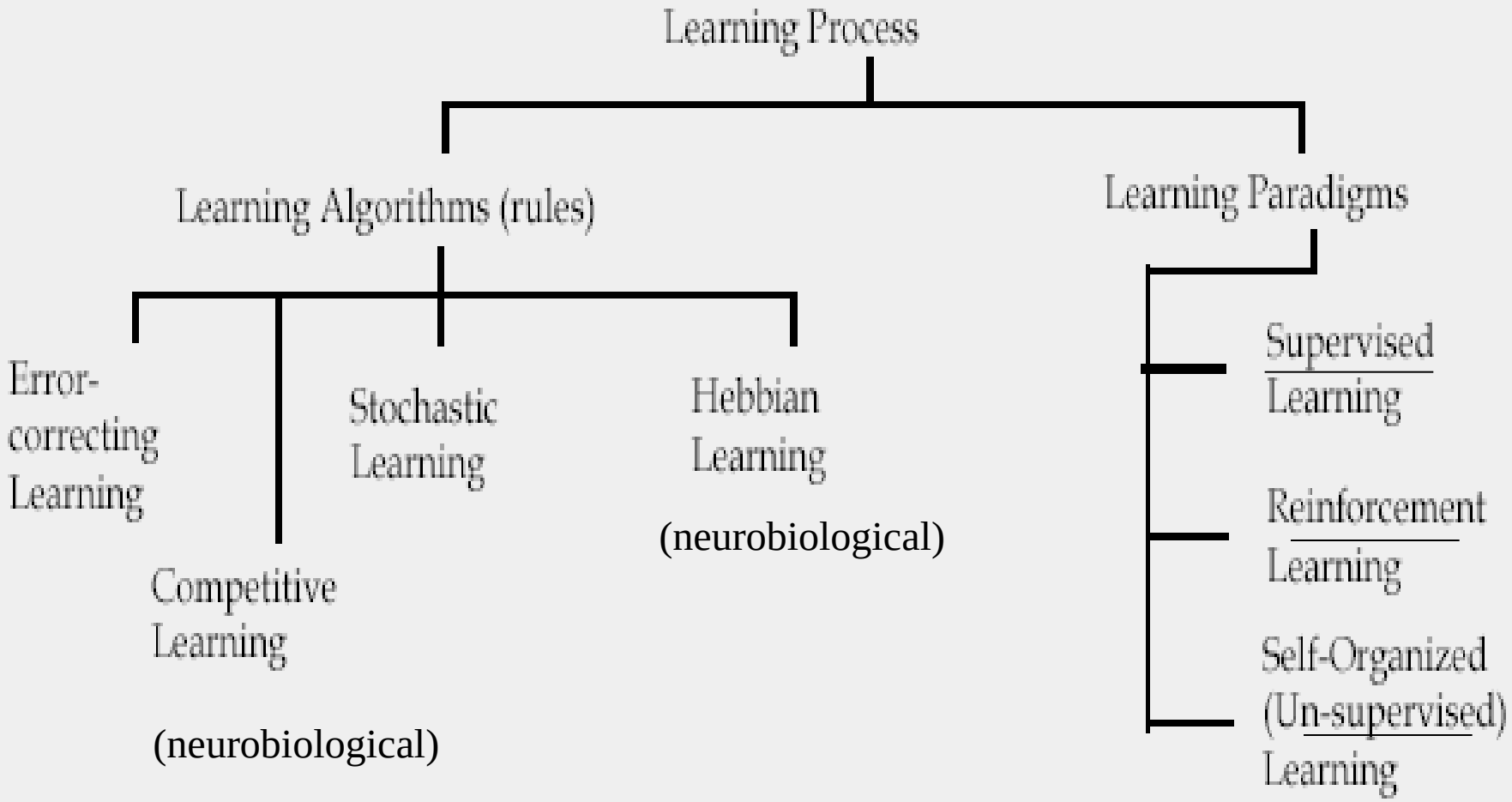
- Learning is a process by which a system modifies its behavior by adjusting its parameters in response to the stimulation by the environment.
- Elements of Learning
  - Stimulation from the environment
  - Parameter adaptation rule
  - Change of behavior of the system

Learning of NN

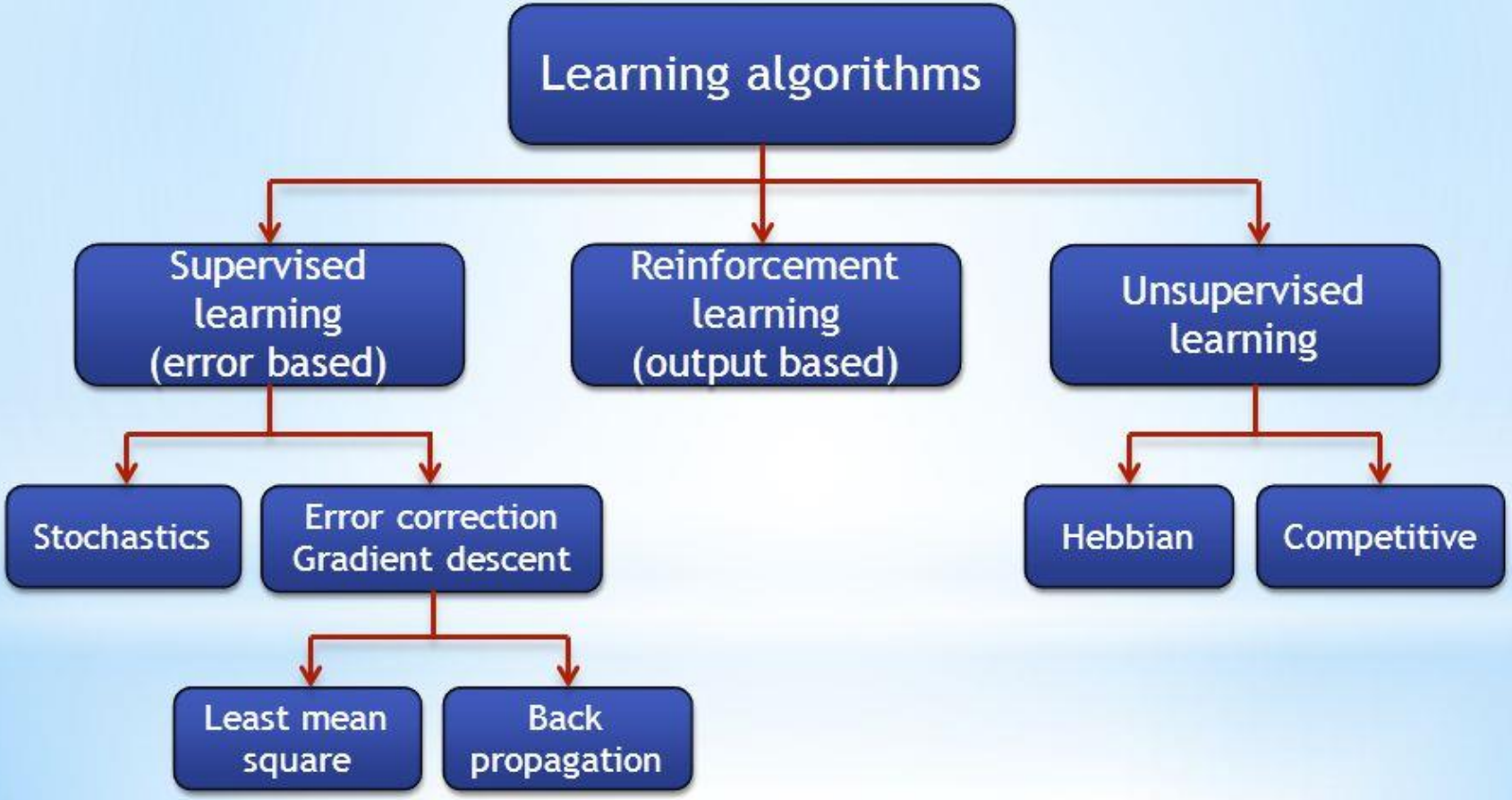
- Learning of NN is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded.
- The **type of the learning** is determined by the manner in which the parameter changes take place. (Mendel & McClaren 1970)


Classification of Artificial Neural Networks
Neural Network Architectures
Learning Methods
Signal Propagation
Single Layer Network
Multi Layer Network
Perceptron
Back Propagation
Supervised Learning
Unsupervised Learning
Reinforced learning
Feed-forward
Recurrent Networks (Feed back)
Perceptron
Back propagation
Hopfield
Back propagation
Hopfield

Taxonomy of Learning



ANN - Training or Learning



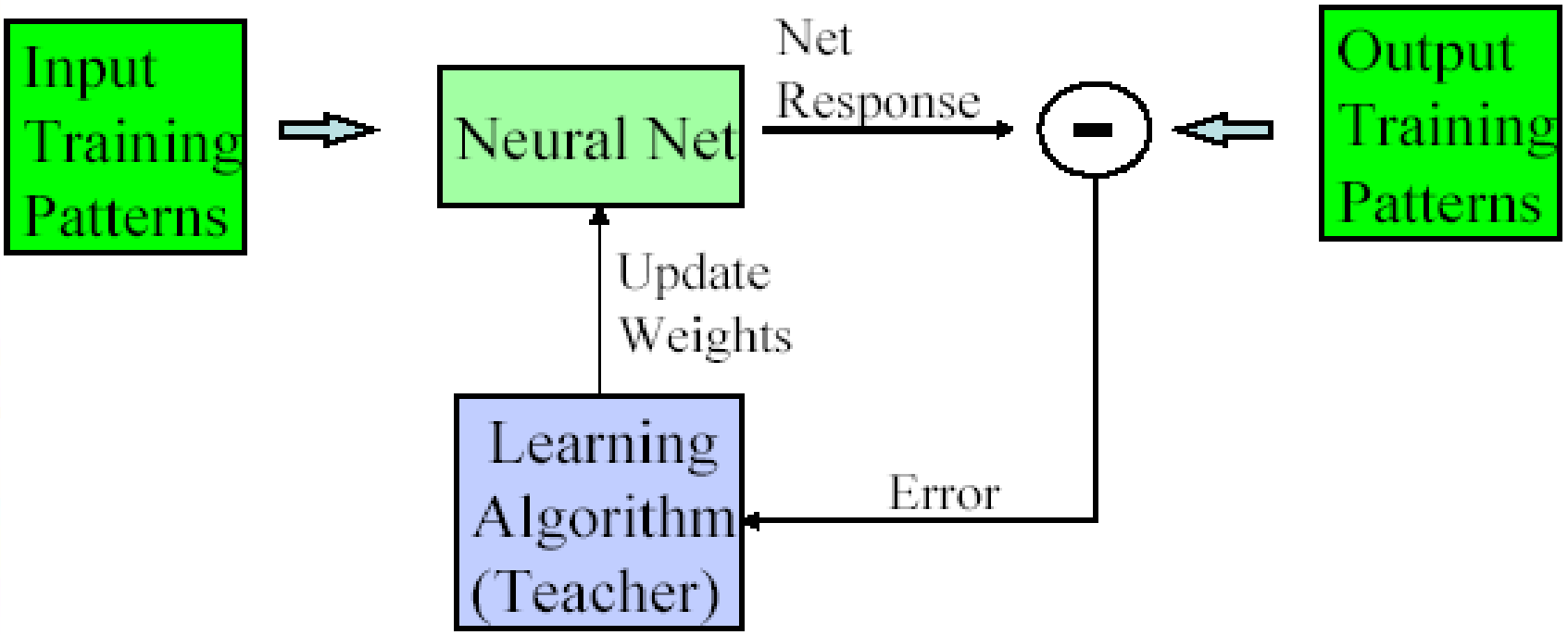
Supervised Learning

- In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs.

- Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights.

- The most common supervised learning method is the **“error correction method”,** using methods such as
  - Least Mean Square (LMS)
  - Back Propagation

NN - Supervised Learning



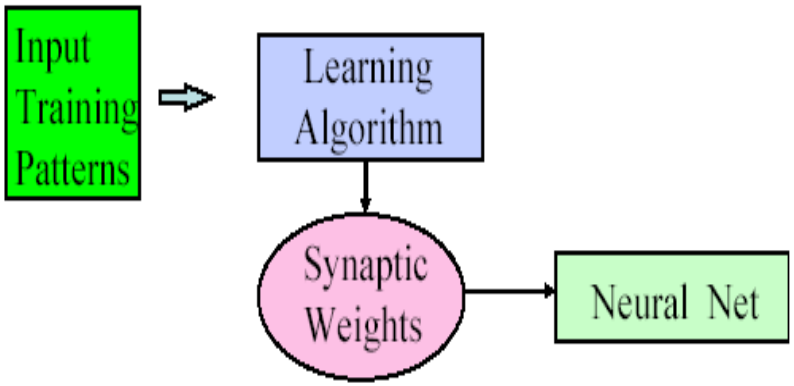
Learning Algorithm trains weights to minimize Error cost function

Unsupervised Learning

- In unsupervised learning, there is no teacher (target signal) from outside and the network adjusts its weights in response to only the input patterns.
- A typical example of unsupervised learning is
-

Unsupervised Hebbian learning

  – Ex. Principal component analysis
- **Unsupervised competitive learning**
  – Ex. Clustering,
  – Data compression



Learning Algorithm trains weights to reach some internal cost function

Unsupervised Competitive Learning

- In *unsupervised competitive learning* the neurons take part in some competition for each input. The winner of the competition and sometimes some other neurons are allowed to change their weights

- In *simple competitive learning* only the winner is allowed to learn (change its weight).
- In ***self-organizing maps*** other neurons in the neighborhood of the winner may also learn.

Learning Tasks

Supervised

Data:
Labeled examples
(input , desired output)

Tasks:
classification
pattern recognition
regression
NN models:
perceptron
adaline
feed-forward NN
radial basis function
support vector machines

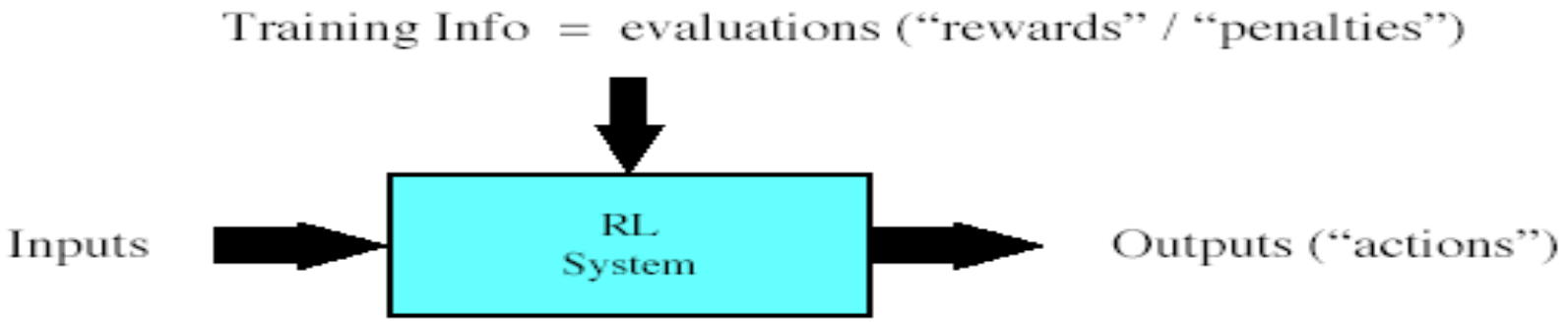
Unsupervised

Data:
Unlabeled examples
(different realizations of the input)

Tasks:
clustering

NN models:
self-organizing maps (SOM)
Hopfield networks

Reinforcement Learning



Objective: get as much reward as possible

Reinforcement learning: Generalization of Supervised Learning;

Reward given later, not necessarily tied to specific action

Learning Laws

- Perceptron Rule
- Hebb's Rule
- Hopfield Law
- Delta Rule (Least Mean Square Rule)
- The Gradient Descent Rule
- Kohonen's Learning Law

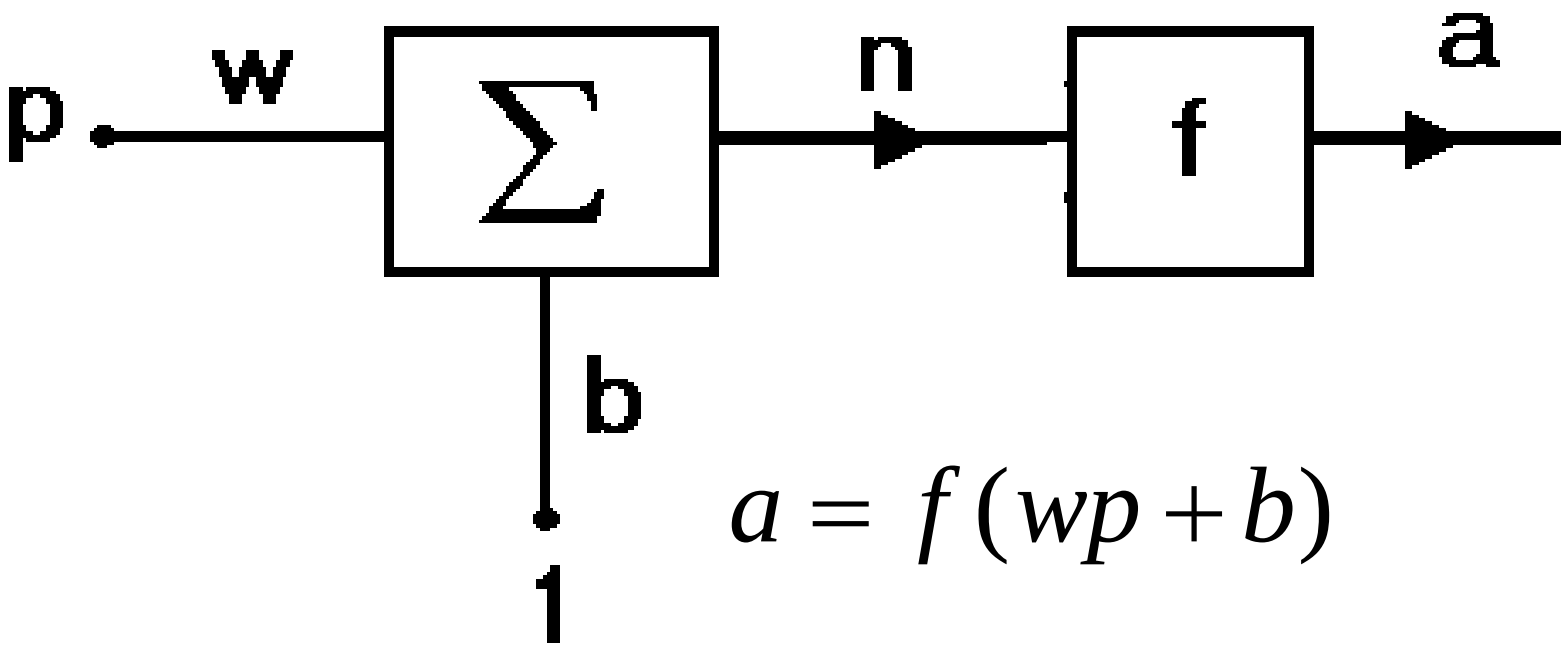
Neuron Models

- McCulloch-Pitts neuron is one example
  - Different functions can be used
  - Different connectivity patterns can be used

McCulloch and Pitts Model

- McCulloch and Pitts Neuron
  - It is a simple model as a binary threshold unit
  - The neuron first computes a weighted sum of its inputs
  - It outputs one if the weighted sum is above a threshold and zero otherwise

A Single-Input Neuron



$$a = f(wp + b)$$

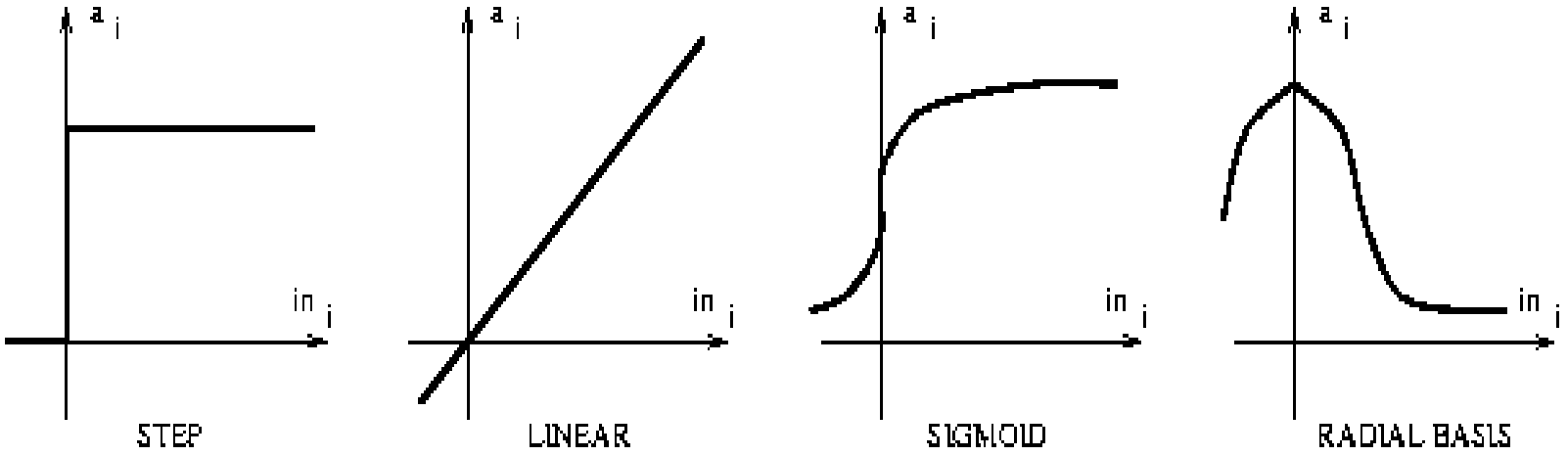
Note that w and b are adjustable parameters of the neurons: *w* is called weight and *b* is called bias

A Single-Input Neuron – cont.

- Transfer functions
  - They can be linear or nonlinear
    - What is a linear function?
    - What is a nonlinear function?
  - Commonly used transfer functions
    - Hard limit transfer function
    - Linear transfer function
    - Sigmoid function

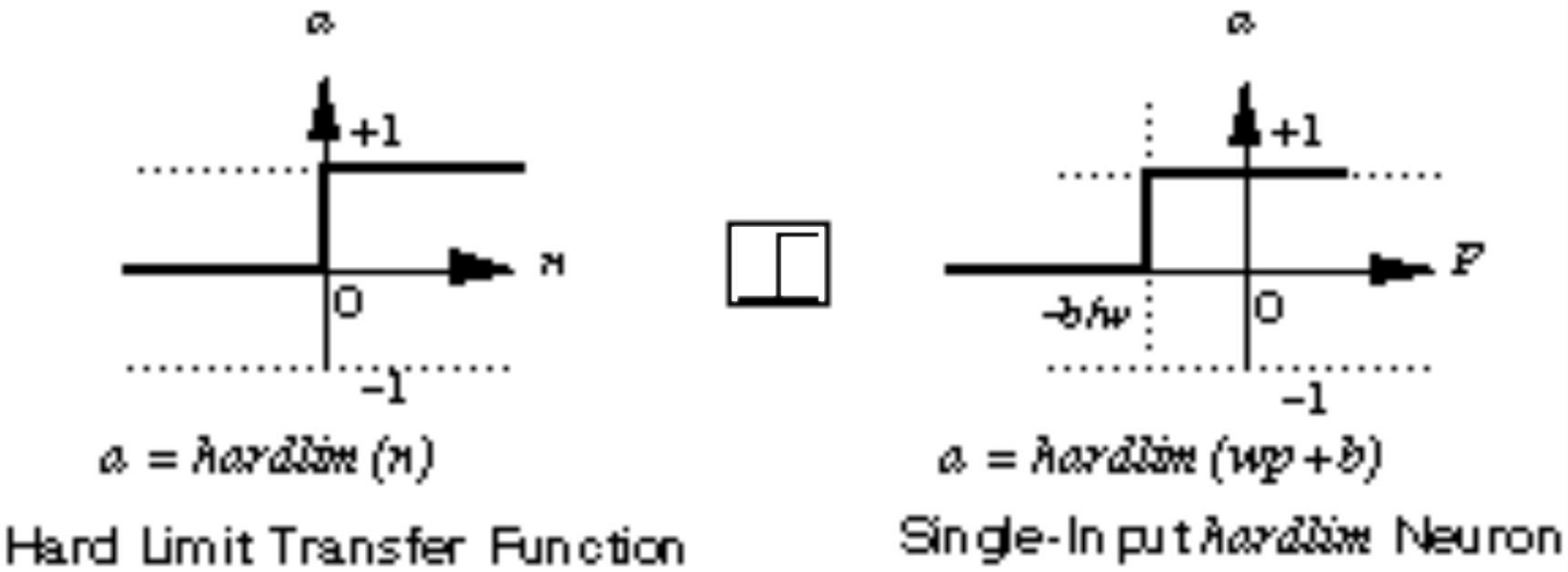
A Single-Input Neuron – cont.

- Transfer functions
  - a = f(n)



A Single-Input Neuron – cont.

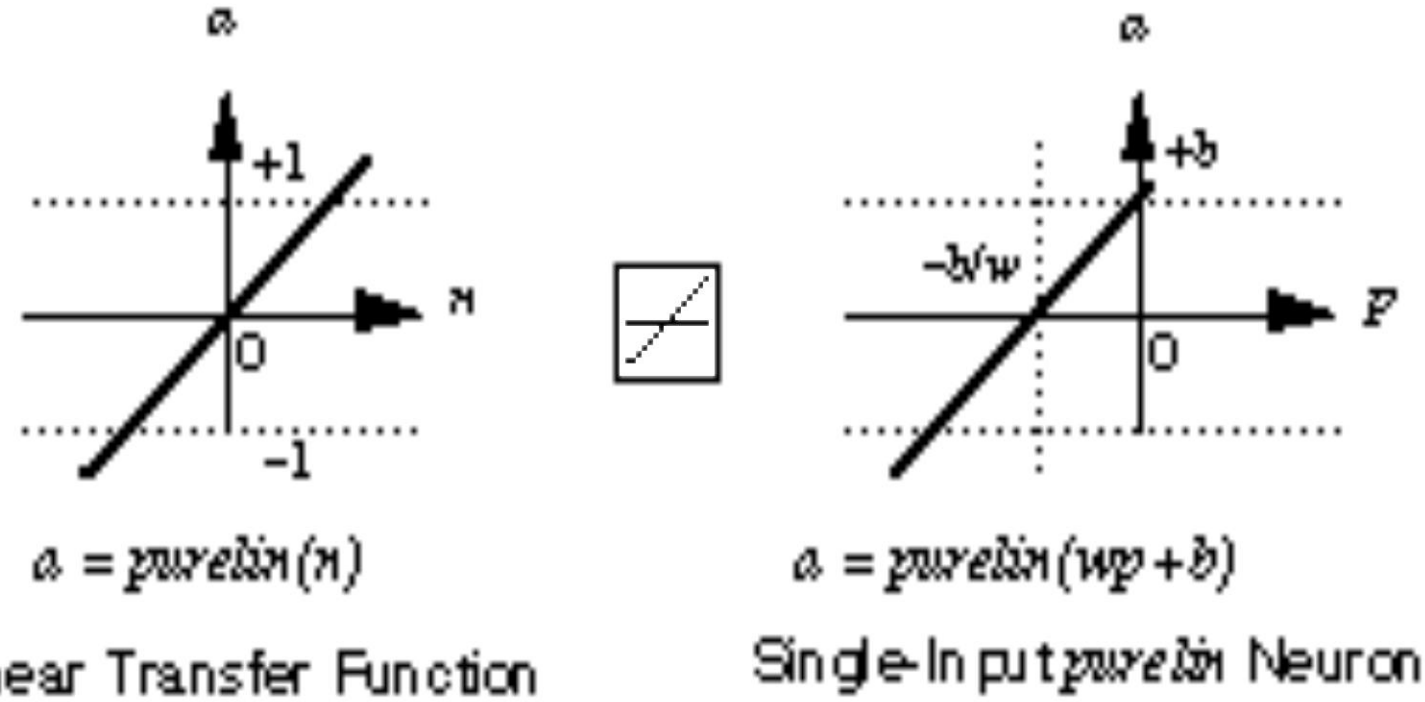
- Hard limit transfer function
  - Also known as the step function



  - Symmetrical hard limit transfer function

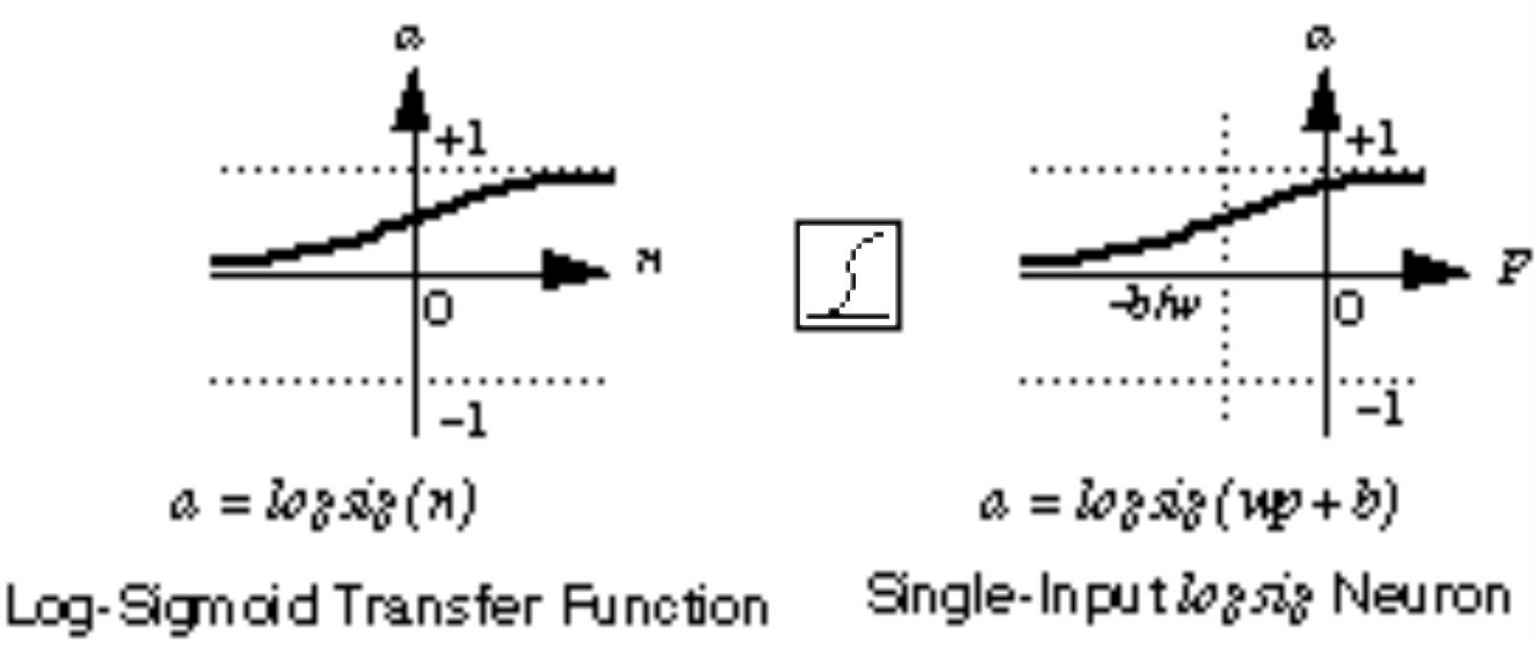
A Single-Input Neuron – cont.

- Linear transfer function



A Single-Input Neuron – cont.

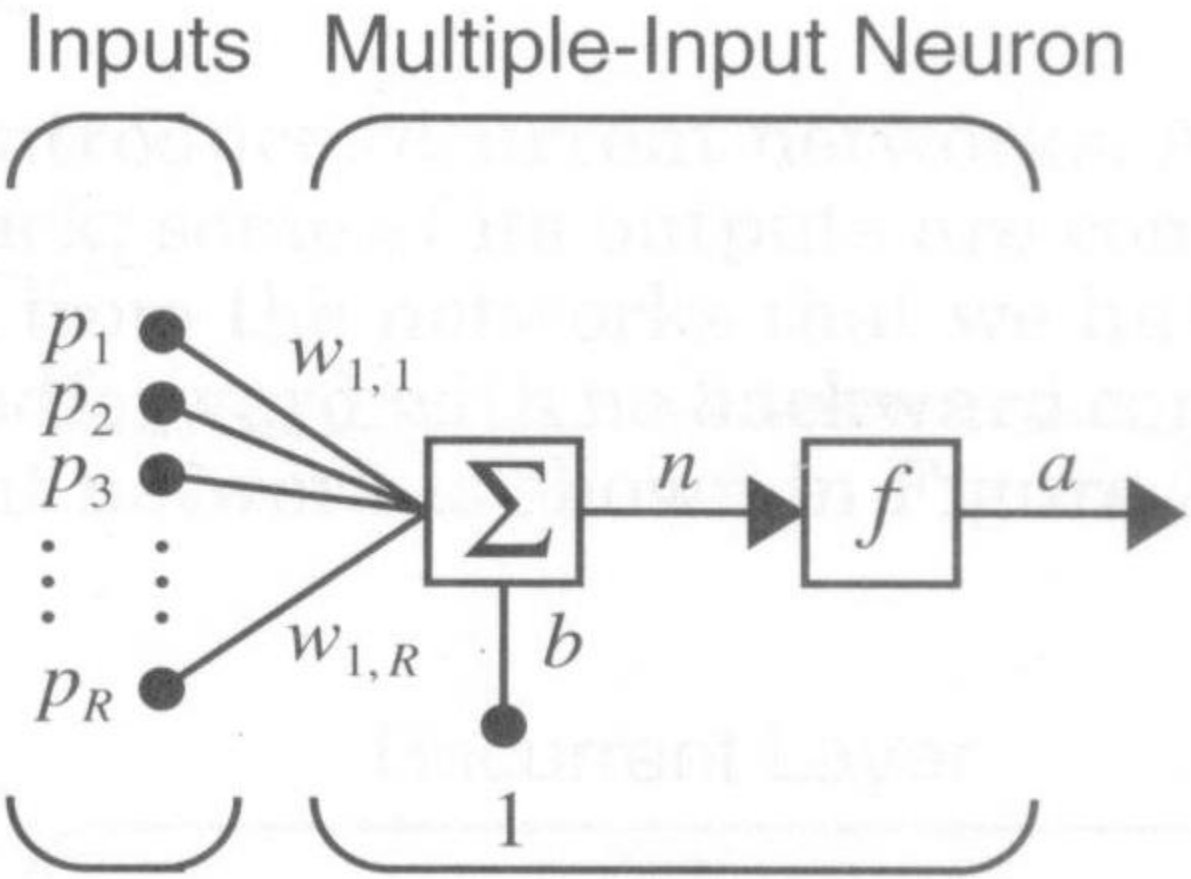
- Log-sigmoid transfer function



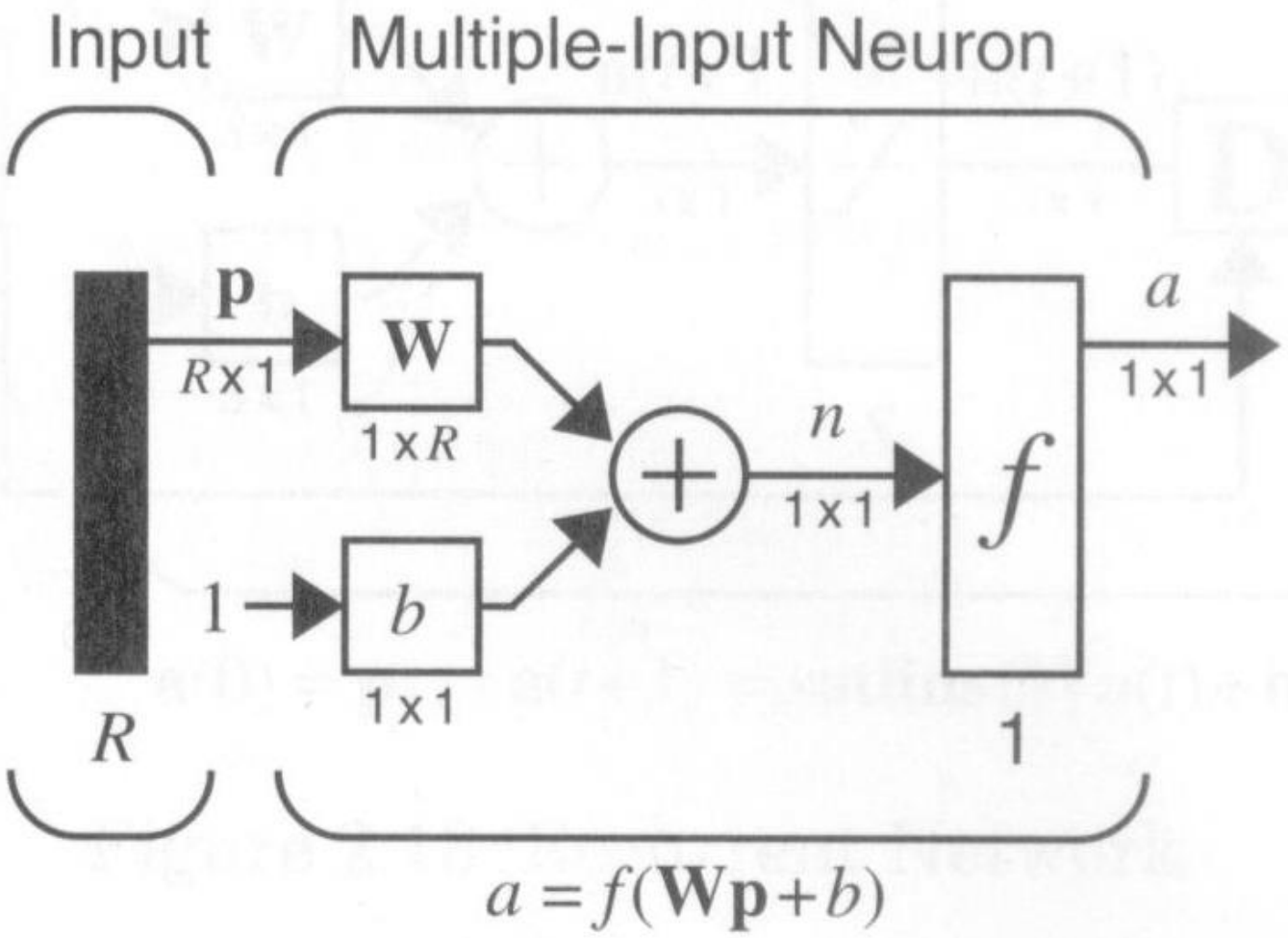
A Single-Input Neuron – cont.

- Other transfer functions
  - See Table 2.1 on p. 2-6
- How to choose transfer functions
  - Transfer functions are determined by the input and desired output range

Multiple-Input Neuron



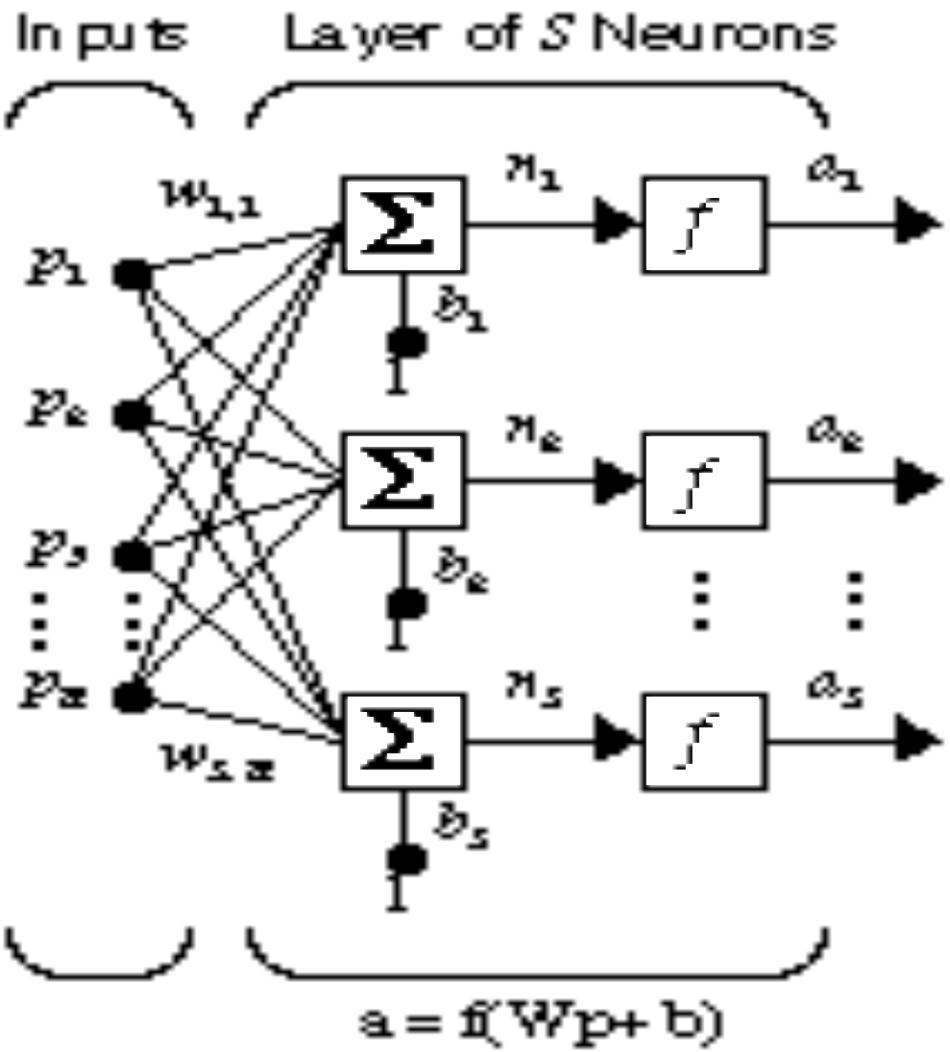
Multiple-Input Neuron – cont.



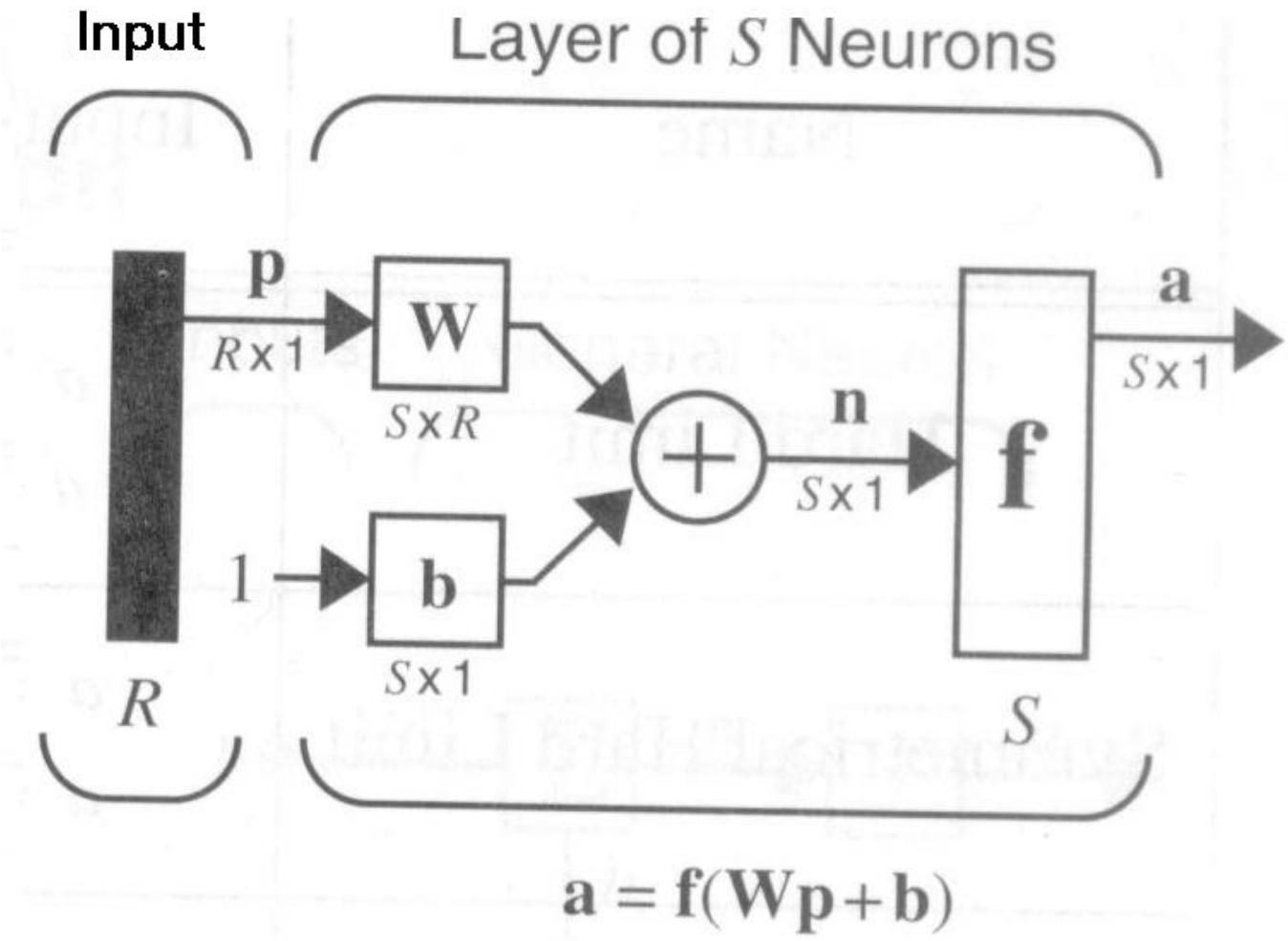
Neural Network Architecture

- A layer of neurons
  - Consists of a set of neurons
  - The inputs are connected to each of the neurons

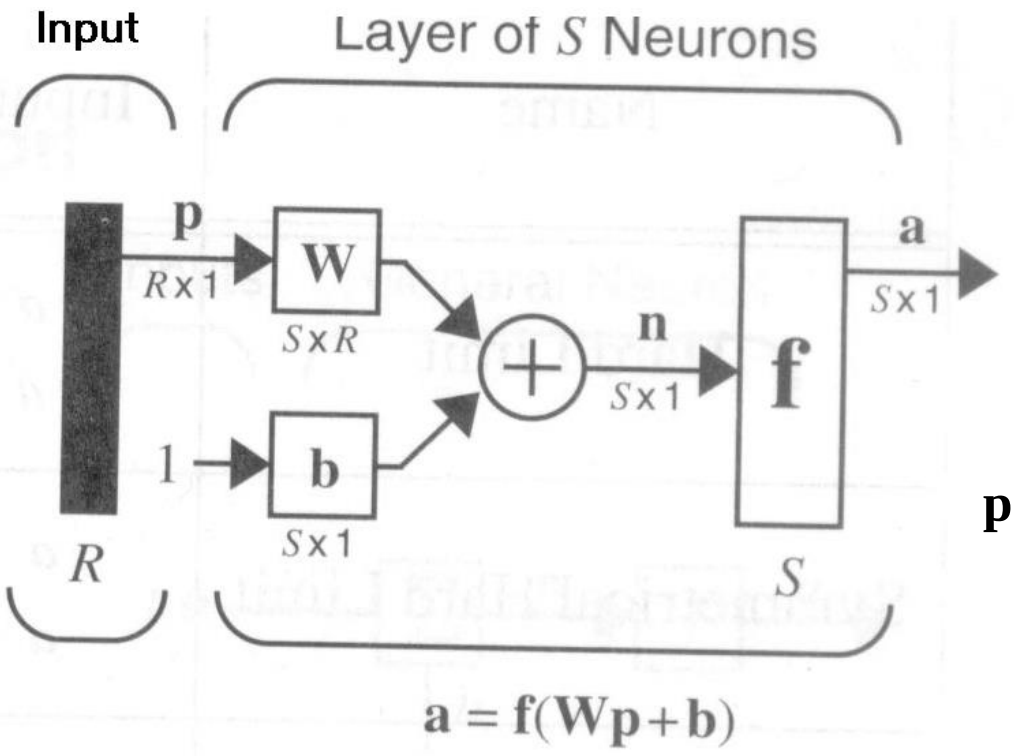
A Layer of Neurons



A Layer of Neurons



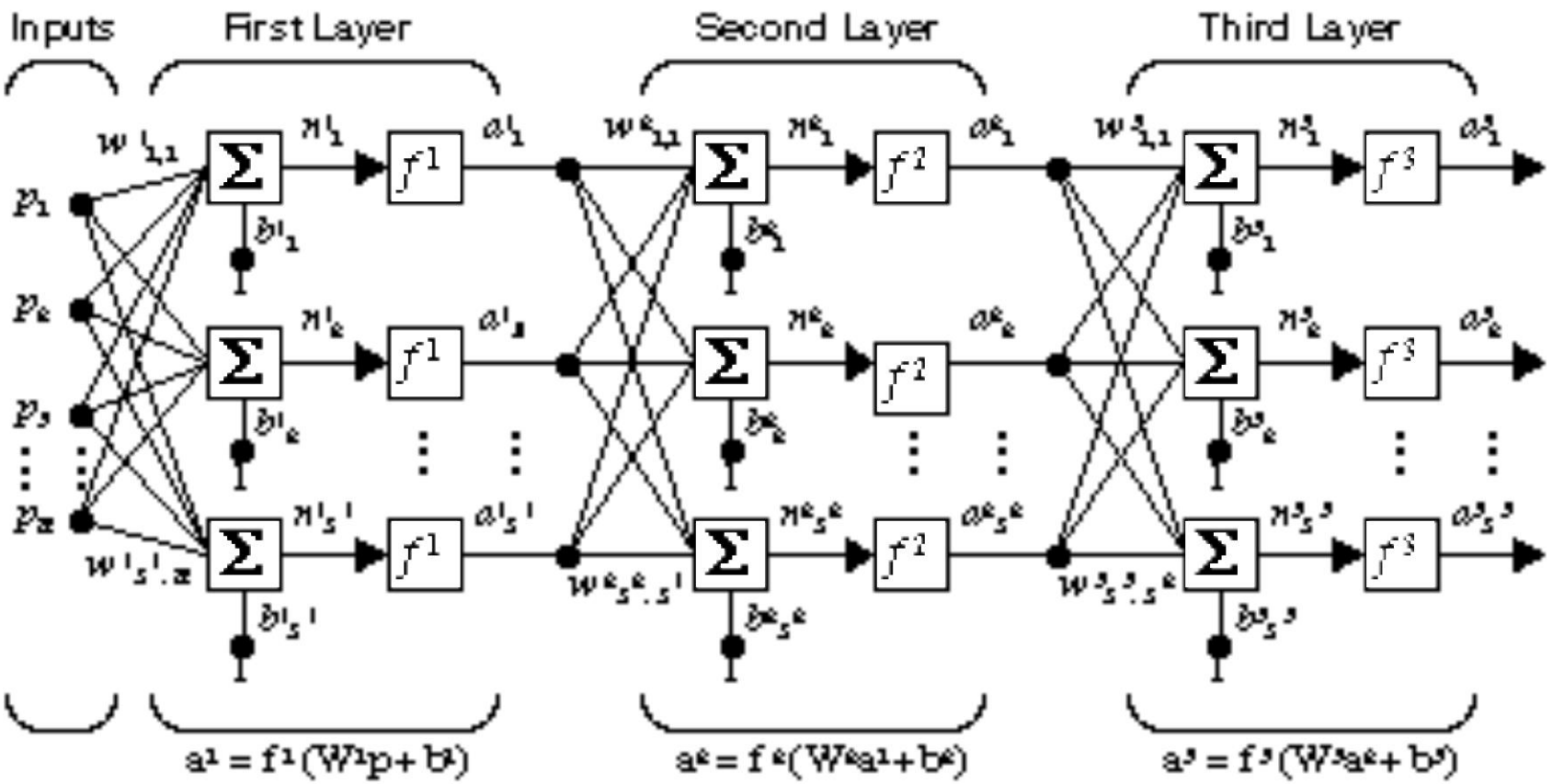
A Layer of Neurons



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

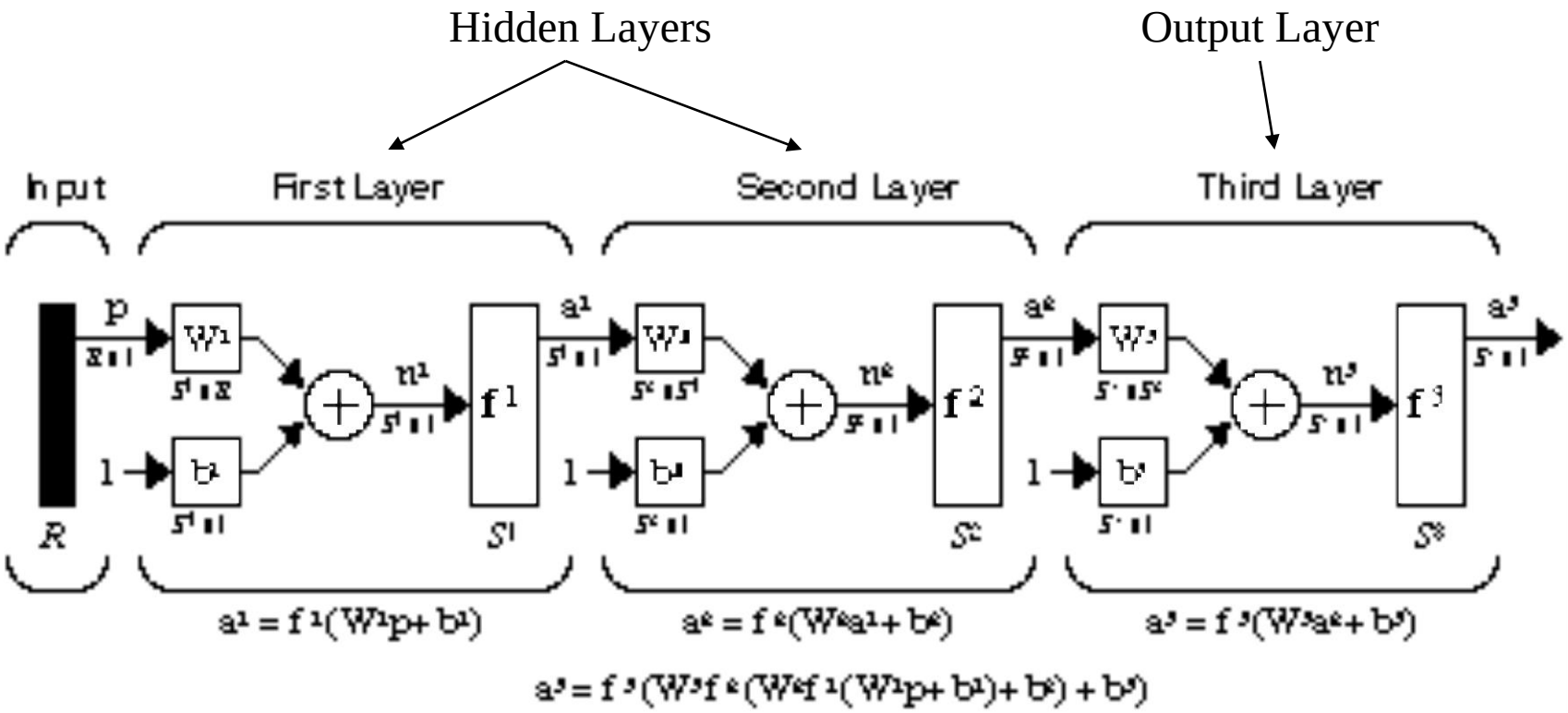
$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Multiple Layers of Neurons



$$a^3 = f^3(W^3 f^2(W^2 f^1(W^1 p + b^1) + b^2) + b^3)$$

Multiple Layers of Neurons

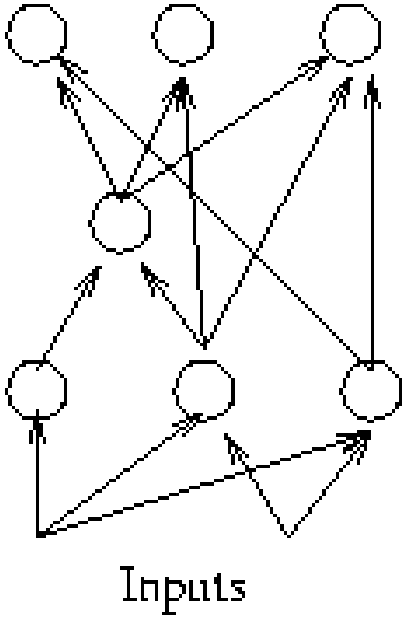


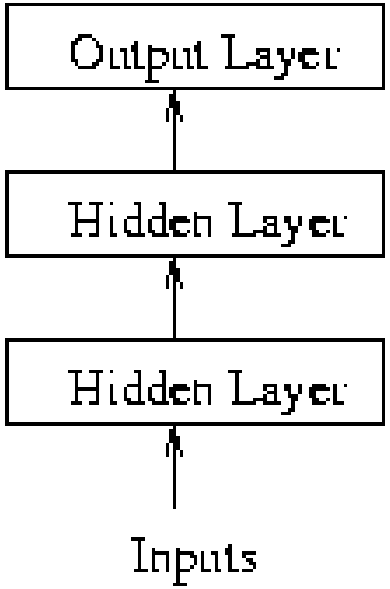
Multiple Layers of Neurons – cont.

- Multiple layer neural networks
  - A network with several layers
  - Multi-layer networks are more powerful than single-layer networks with nonlinear transfer functions

Neural Network Architecture – cont.

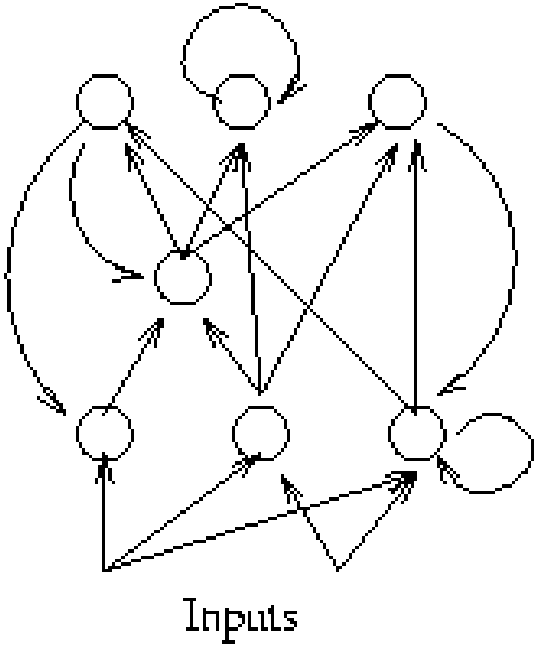
- Feed-forward networks

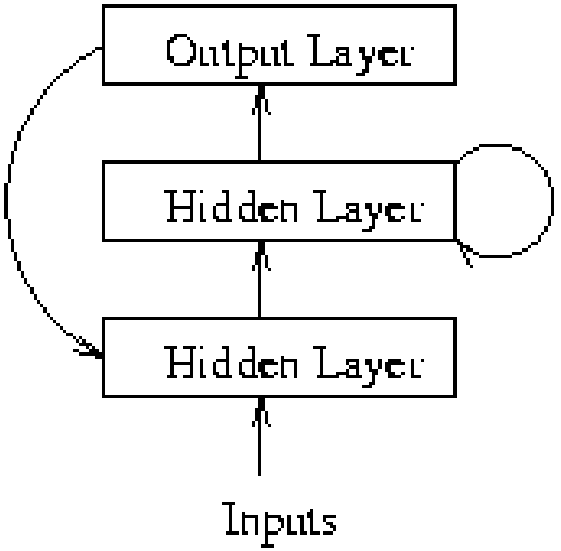




Neural Network Architecture – cont.

- Recurrent neural networks

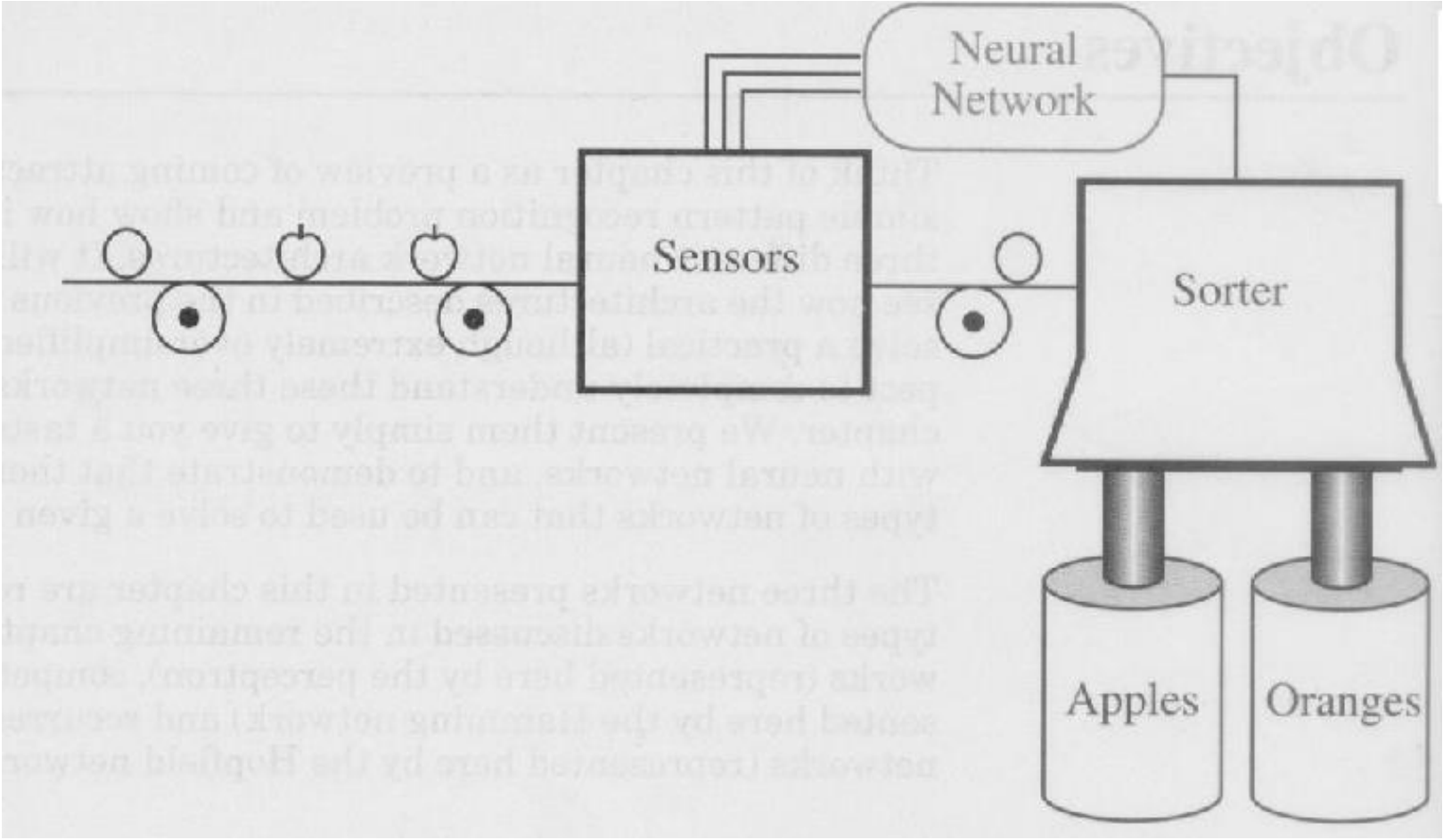




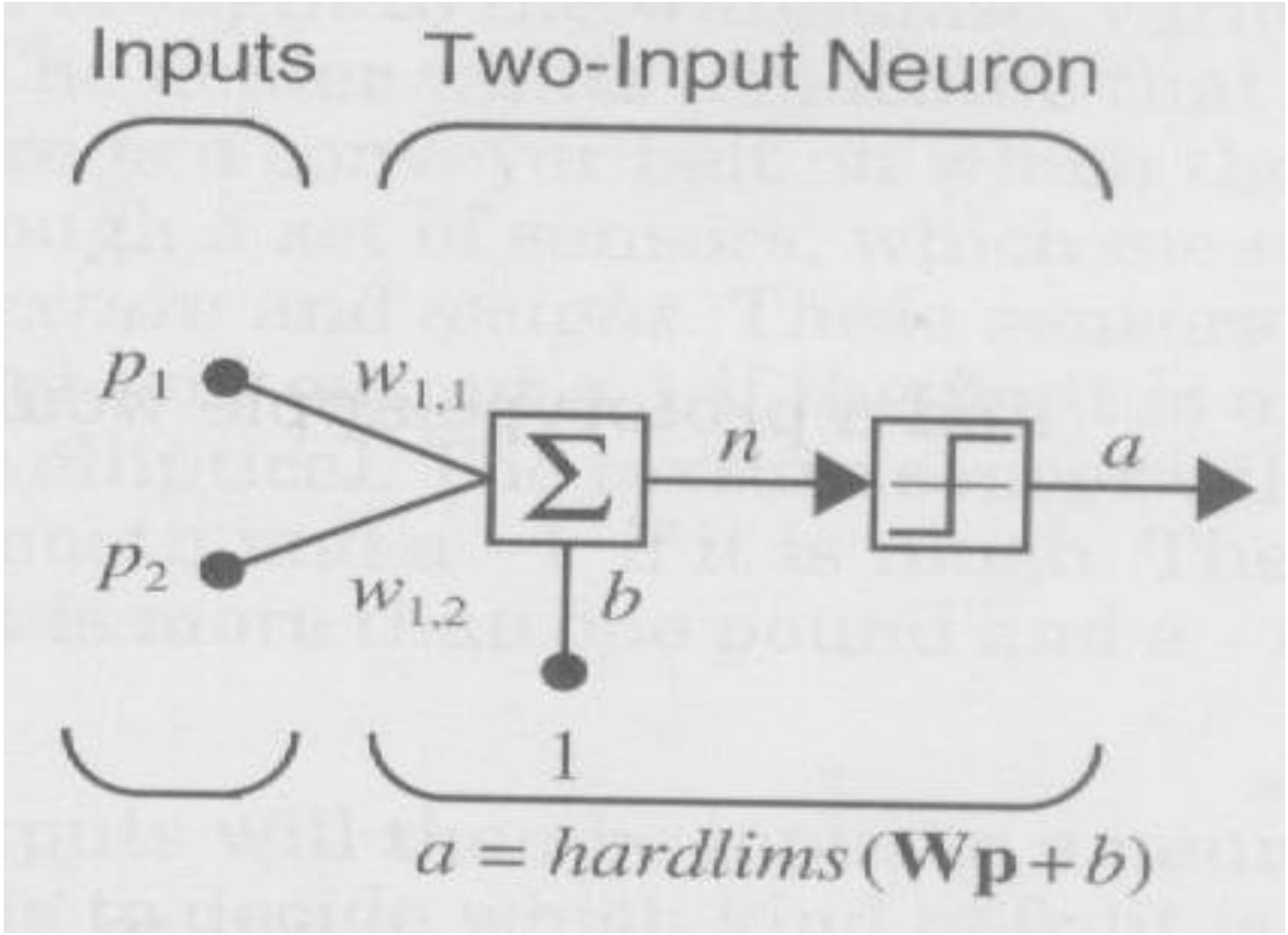
Network Architecture – cont.

- How to determine a network architecture
  - Number of network inputs is given by the number of problem inputs
  - Number of neurons in output layer is given by the number of problem outputs
  - The transfer function is partly determined by the output specifications

An Illustrative Example



Perceptron


Inputs
Two-Input Neuron
p_1
p_2
$w_{1,1}$
$w_{1,2}$
Σ
n
a
b
1
$a = hardlims(\mathbf{W}\mathbf{p}+b)$

Outline

- Perceptron Neural Network
- Perceptron Learning Rule
  - Perceptron network
  - Decision boundary

Using Neural Networks

- Choose a neural network architecture
  - Feed-forward network
  - Recurrent network
  - Each network has its own characteristics
    - Simple networks may not be able to solve a complex problem
    - For example, perceptron can only solve linearly separable problems

Using Neural Networks – cont.

- Specify the network architecture
  - How many layers
  - How many neurons at each layer
  - A specific connection pattern
- Choose a learning algorithm
  - Specify the parameters of the learning algorithm
  - Decide how to train the network

Using Neural Networks – cont.

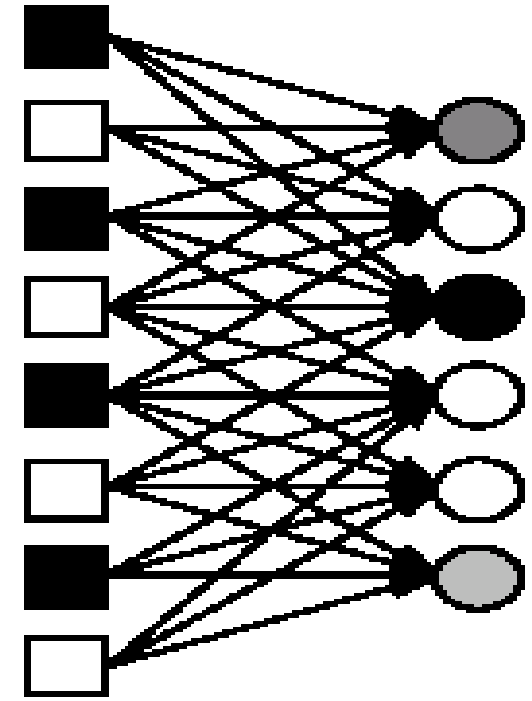
- Testing
  - A trained network will be normally tested on data that are not used during training
  - A network's ability to process outside training is called generalization
    - Generalization is critical for practical applications

The Perceptron

by Frank Rosenblatt (1958, 1962)

- Two-layers
- binary nodes (McCulloch-Pitts nodes) that take values 0 or 1
- continuous weights, initially chosen randomly
- Today, used as a synonym for a single-layer, feed-forward network

Perceptrons

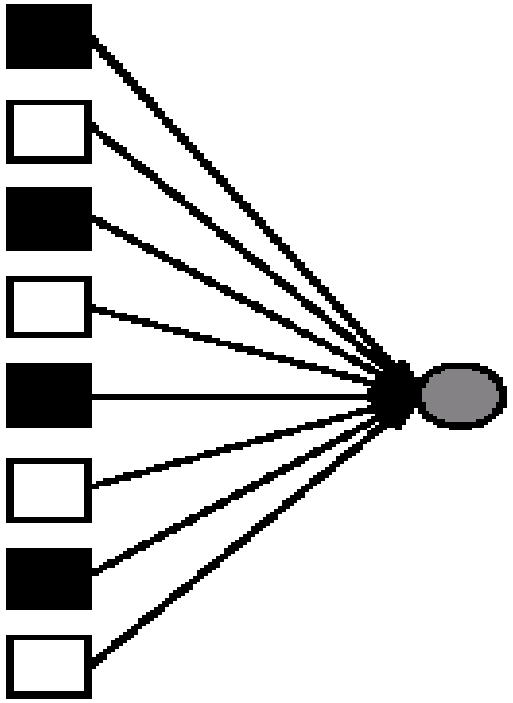


I_j $W_{j,i}$ O_i

Input Units

Output Units

Perceptron Network



I_j W_j O

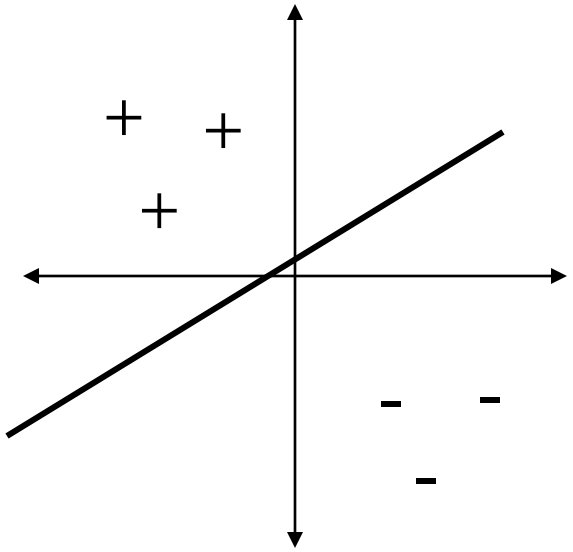
Input Units

Output Unit

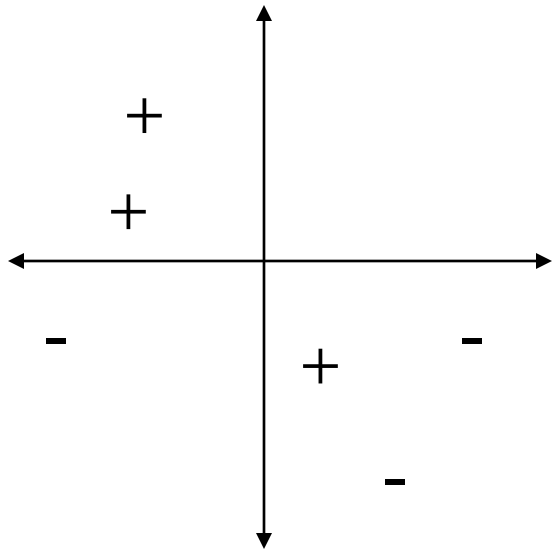
Single Perceptron

Hyperplane, again

A perceptron can learn only examples that are called "linearly separable". These are examples that can be perfec separated by a hyperplane.



Linearly separable



Non-linearly separable

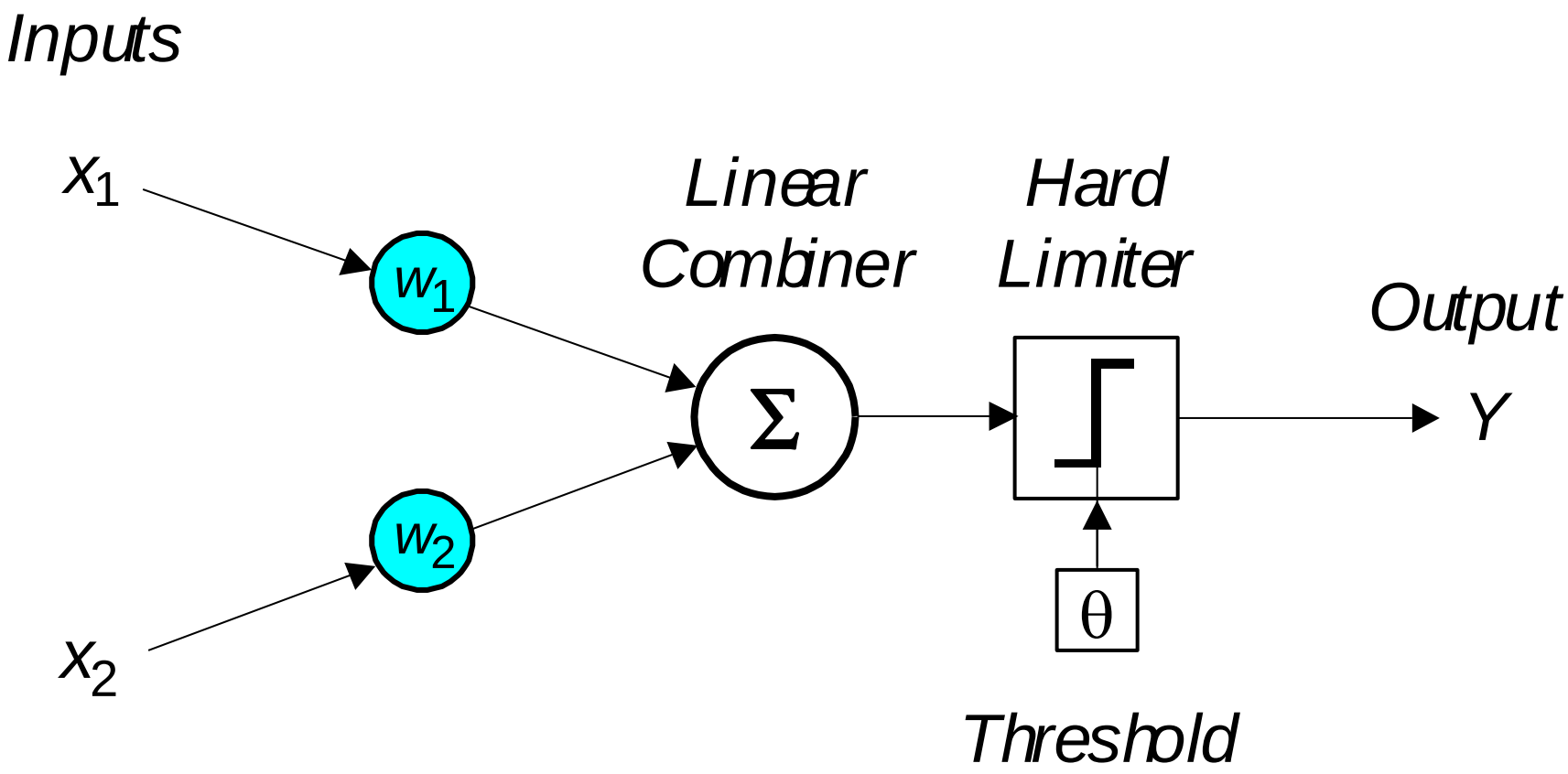
How does the perceptron learn its classification tasks?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [–0.5, 0.5], and then updated to obtain the output consistent with the training examples.

Can a single neuron learn a task?

- In 1958, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure for training a simple ANN: a **perceptron**.
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a *hard limiter*.

Single-layer two-input perceptron



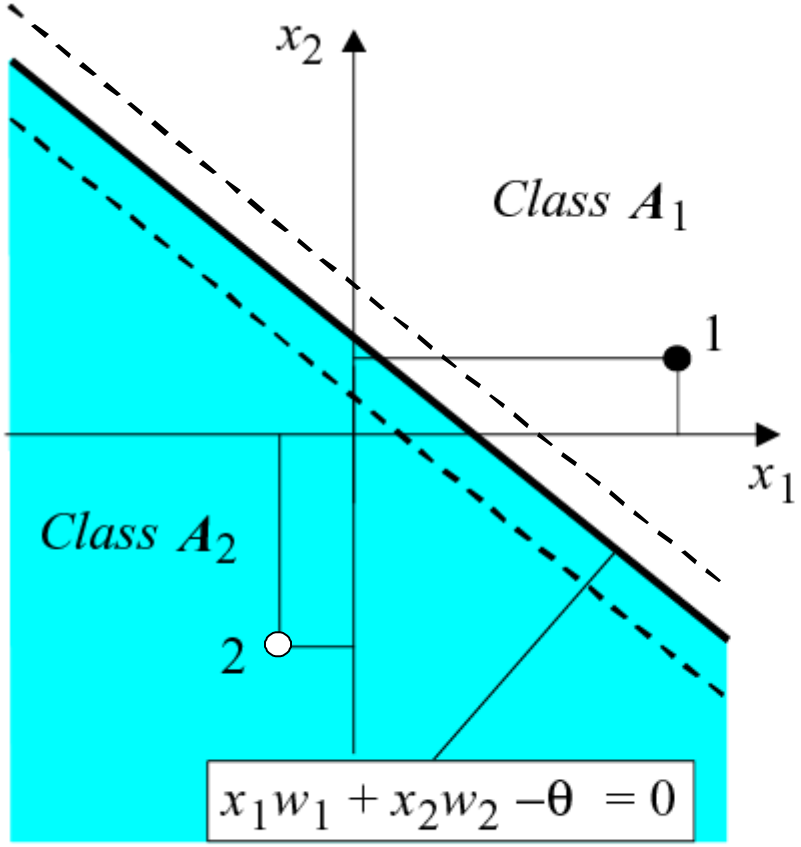
The Perceptron

- The operation of Rosenblatt's perceptron is based on the **McCulloch and Pitts neuron model**. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and –1 if it is negative.

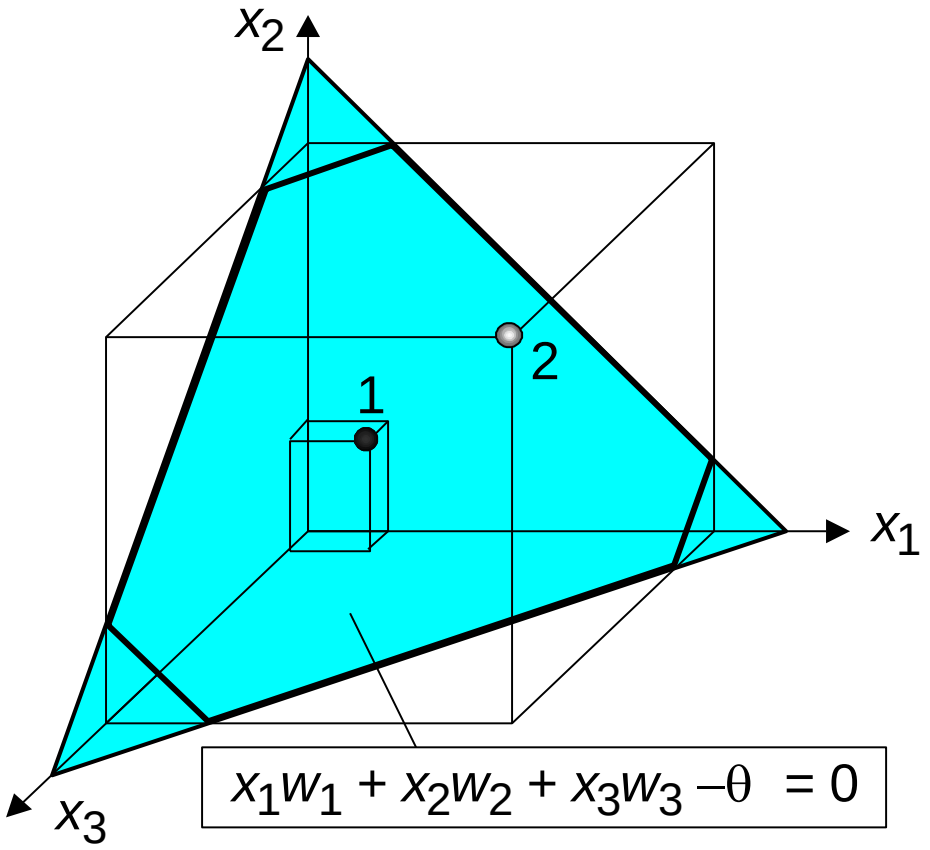
- The aim of the perceptron is to classify inputs, $x_1, x_2, \ldots, x_n$, into one of two classes, say A_1 and A_2.
- In the case of an elementary perceptron, the n-dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the ***linearly sep***

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

Linear separability in the perceptrons



(*a*) Two-input perceptron.



(*b*) Three-input perceptron.

How does the perceptron learn its classification tasks?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [–0.5, 0.5], and then updated to obtain the output consistent with the training examples.

- If at iteration p, the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by:

$$e(p) = Y_d(p) - Y(p) \quad \text{where } p = 1, 2, 3, \ldots$$

  Iteration p here refers to the pth training example presented to the perceptron.

- If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

The perceptron learning rule

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where $p = 1, 2, 3, \ldots$

α is the **learning rate**, a positive constant less than unity.

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

Perceptron's training algorithm

Step 1: Initialisation

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range [–0.5, 0.5].

If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

Perceptron's training algorithm (continued)

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$.

Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

Perceptron's training algorithm (continued)

Step 3: Weight training

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

Update the weights of the perceptron

where Dwi(p) is the weight correction at iteration p.

The weight correction is computed by the delta rule:

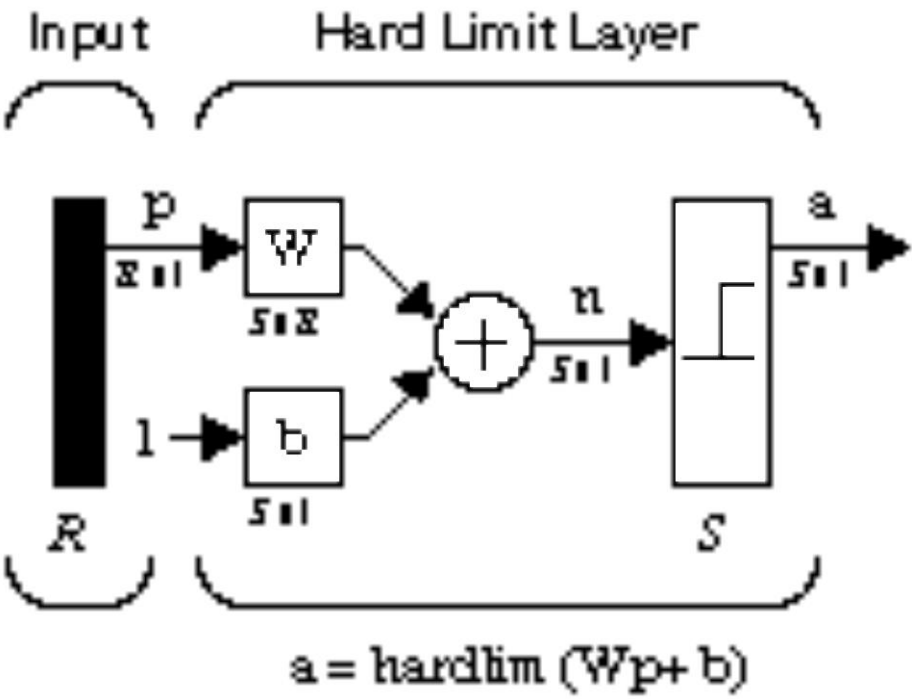
$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Step 4: Iteration Increase iteration p by one, go back to Step 2 and repeat the process until convergence.

Perceptron Network

- A perceptron network is a single-layer neural network
  - With hardlim as the transfer function
    - Or hardlims, which does not affect the capabilities of the network
  - It in general can have more than one neuron
    - We can, however, determine the weights and bias of each neuron individually

Perceptron Architecture



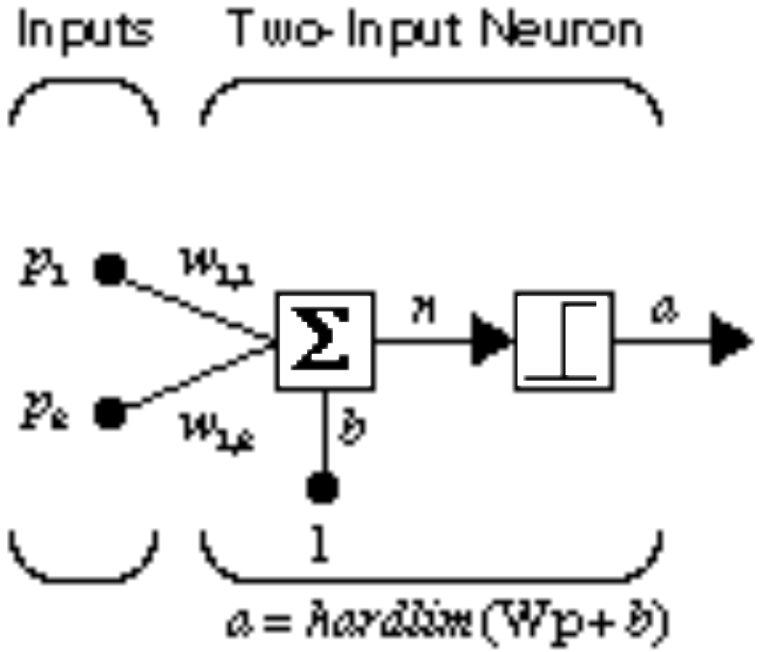
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}$$

$$a_i = hardlim(n_i) = hardlim({}_i\mathbf{w}^T\mathbf{p} + b_i)$$

Single-Neuron Perceptron

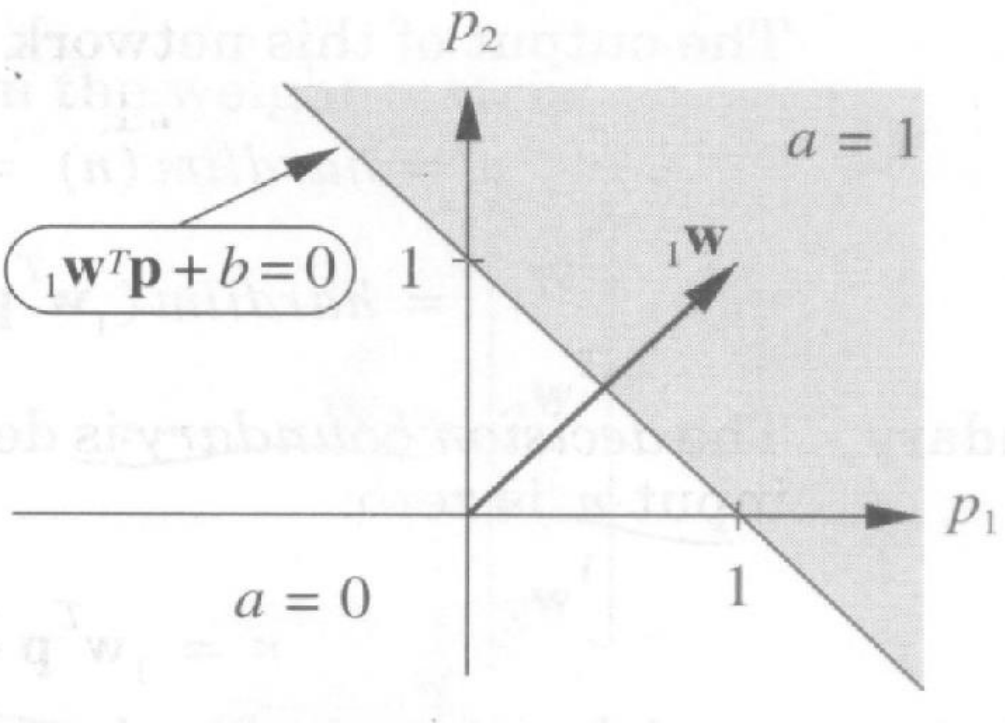
- The output of a single-neuron perceptron is given by $a = \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b)$



Single-Neuron Perceptron – cont.

- Decision boundary

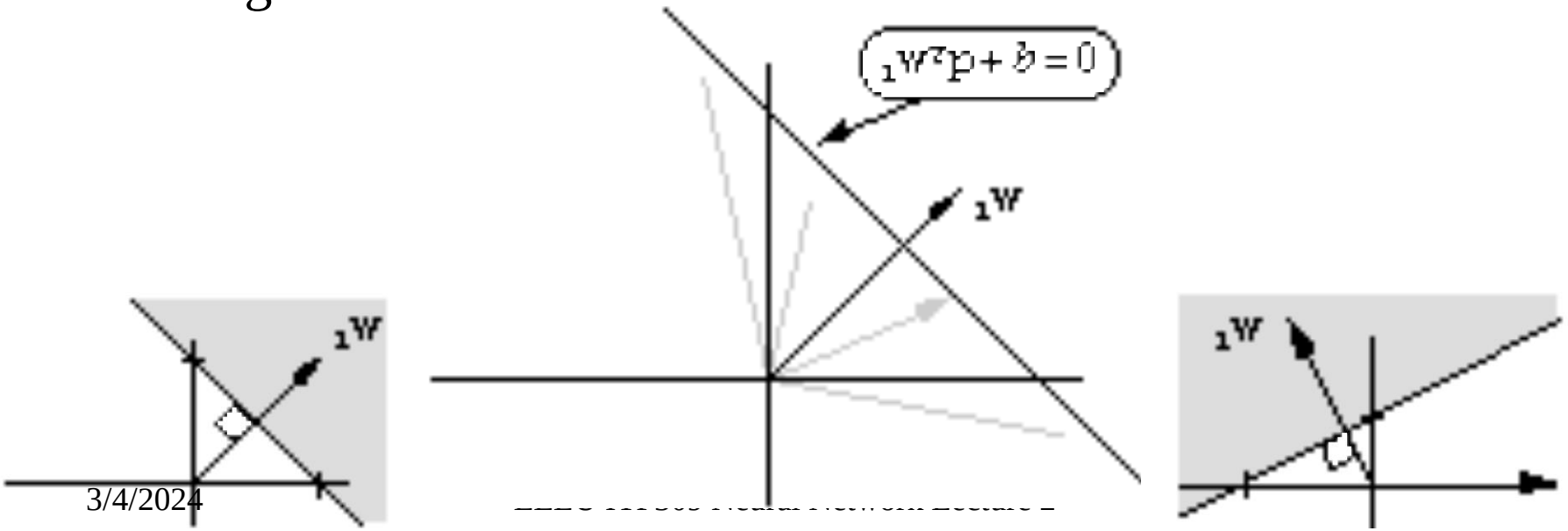
$$n = w_{1,1}p_1 + w_{1,2}p_2 + b = 0$$



Decision Boundary

$${}_1\mathbf{w}^T\mathbf{p} + b = 0 \qquad {}_1\mathbf{w}^T\mathbf{p} = -b$$

- All points on the decision boundary have the same inner product with the weight vector.
- Therefore they have the same projection onto the weight vector, and they must lie on a line orthogonal to the weight vector

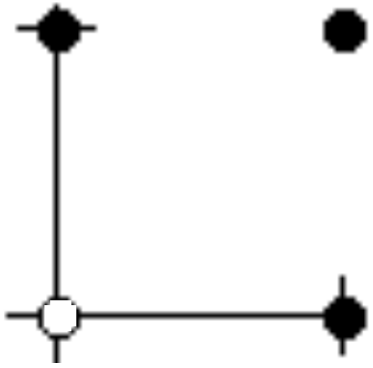


Single-Neuron Perceptron – cont.

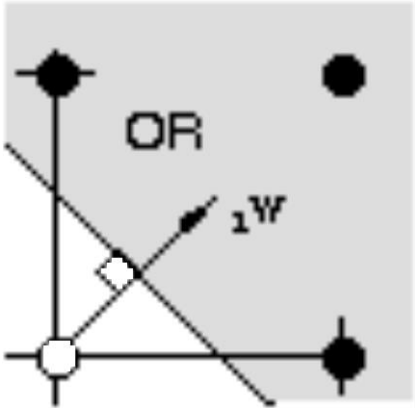
- Design a perceptron graphically
  - Select a decision boundary
  - Choose a weight vector that is orthogonal to the decision boundary
  - Find the bias b
  - When there are multiple decision boundaries, which one is the “best”?

Example - OR

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



OR Solution



Weight vector should be orthogonal to the decision boundary.

$$ {}_1\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} $$

Pick a point on the decision boundary to find the bias.

$$ {}_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + \mathrm{b} = 0 \quad \Rightarrow \quad b = -0.25 $$

Multiple-Neuron Perceptron

- A decision boundary for each neuron
  - For neuron *i* the boundary is defined as

$$_i w^T p + b_i = 0$$

  - A multiple-neuron perceptron can classify into many categories

Learning Rules

- A learning rule is a procedure for modifying the weights and biases of a neural network
  - It is often called a training algorithm
  - The purpose of the learning rule is to train the network to perform some task

Learning Rules – cont.

- Three categories of learning algorithms
  - Supervised learning
    - Training set $\{p_1, t_1\}, \{p_2, t_2\}, \ldots\ldots, \{p_Q, t_Q\}$
    - Here t_q is called the target for input p_q
  - Reinforcement learning
    - No correct output is provided for each network input but a measure of the performance is provided
    - It is not very common
  - Unsupervised learning
    - There are no target outputs available

Perceptron Learning Rule

- Perceptron learning rule

$$W^{new} = W^{old} + ep^T$$

$$b^{new} = b^{old} + e$$

Perceptron Learning Rule – cont.

- Proof of convergence
  - We need to show the learning rule will always converge to weights that accomplish the desired classification assuming that such weights exist
  - We need to show that we only need to update the weights finite number of times

Multiple-Neuron Perceptrons

To update the ith row of the weight matrix:

$$ {}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p} $$

$$ b_i^{new} = b_i^{old} + e_i $$

Matrix form:

$$ \mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T $$

$$ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e} $$

Apple/Banana Example

Training Set

$$\left\{\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \end{bmatrix}\right\}$$

Initial Weights

$$\mathbf{W} = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \qquad b = 0.5$$

First Iteration

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(-0.5) = 0 \qquad e = t_1 - a = 1 - 0 = 1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (1)\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$

Second Iteration

$$a = hardlim\,(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} + (1.5)\right)$$

$$a = hardlim\,(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix} + (-1)\begin{bmatrix}1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Check

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(1.5) = 1 = t_1$$

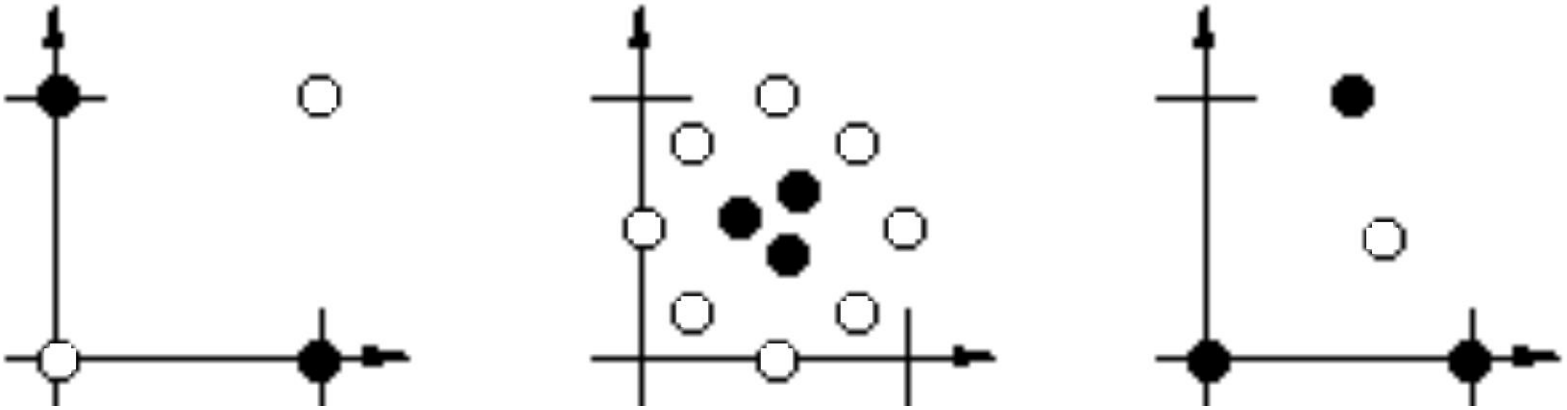
$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(-1.5) = 0 = t_2$$

Limitations of Perceptrons

- Linear separability
  - Perceptron networks can only solve problems that are linearly separable, which means the decision boundary is linear
- Many problems that are not linear separable
  - XOR is a well-know example

Linearly Inseparable Problems



Limitations of Perceptrons – cont.

- Possible solutions
  - Have a more complex network architecture
    - Have several layers instead of one-layer
    - How to train the network then?
  - Project the data into a high dimensional feature space
    - Called kernel representation
    - The resulting network is called support vector machines, which are widely used for real-world applications