CS 4104
APPLIED MACHINE LEARNING

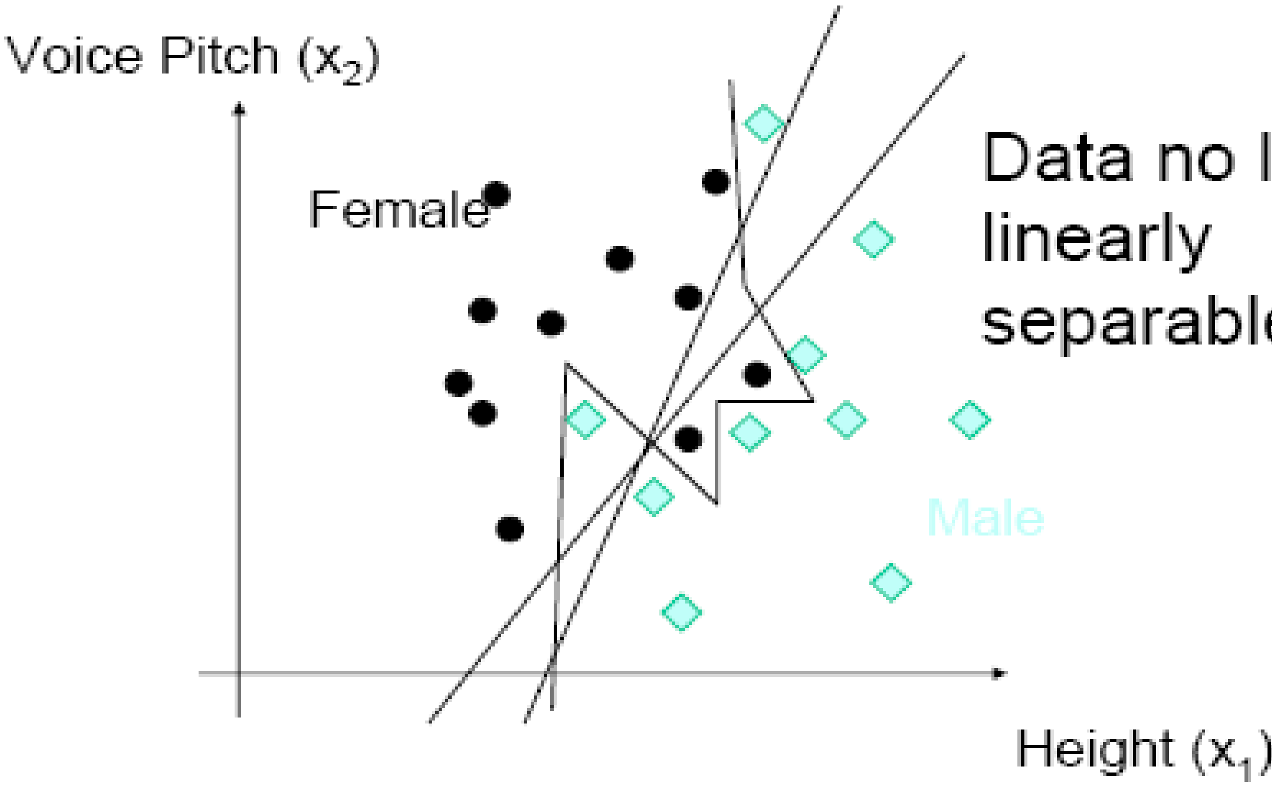
Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

MULTILAYER NETWORKS

Multilayer Networks… Example



Data no longer linearly separable

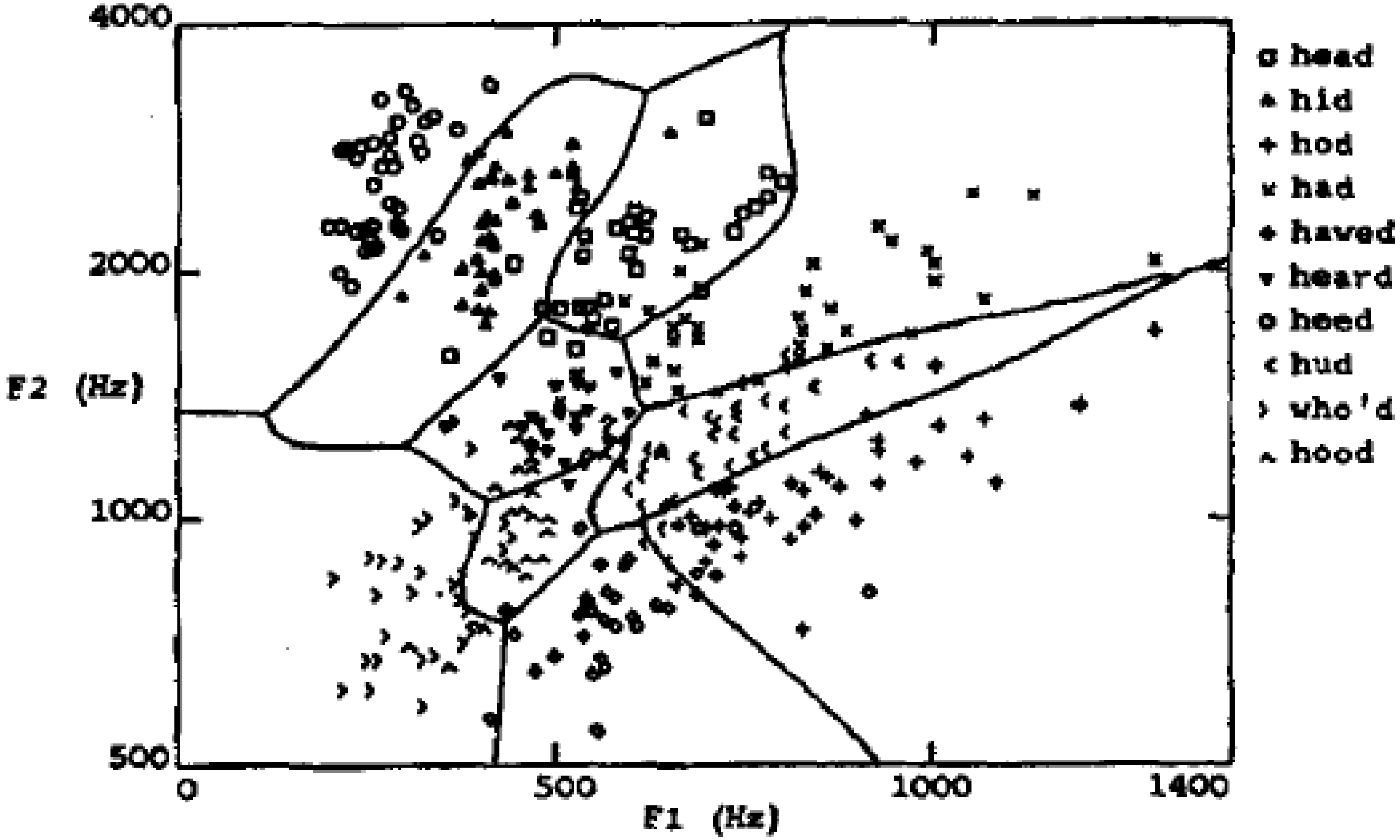
What is a good decision boundary ?

Multilayer Networks… Example

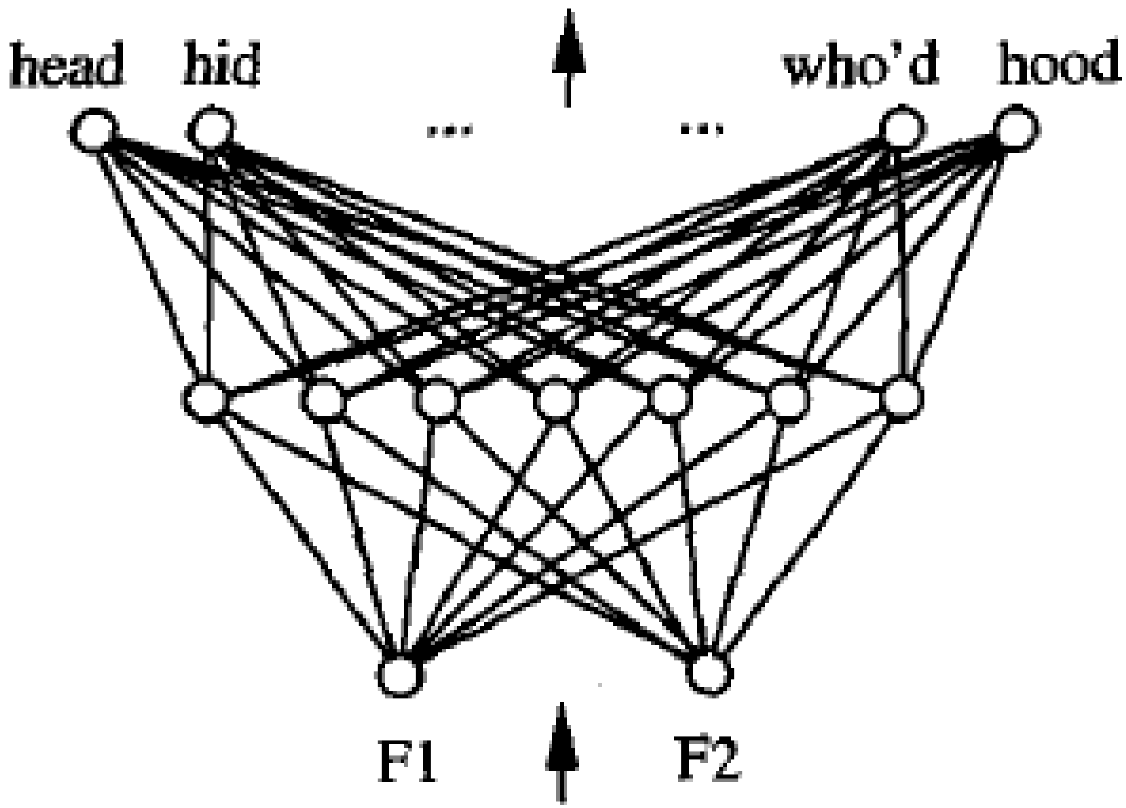
Example:

- The speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h-d" (i.e., "hid," "had," "head," "hood," etc.).

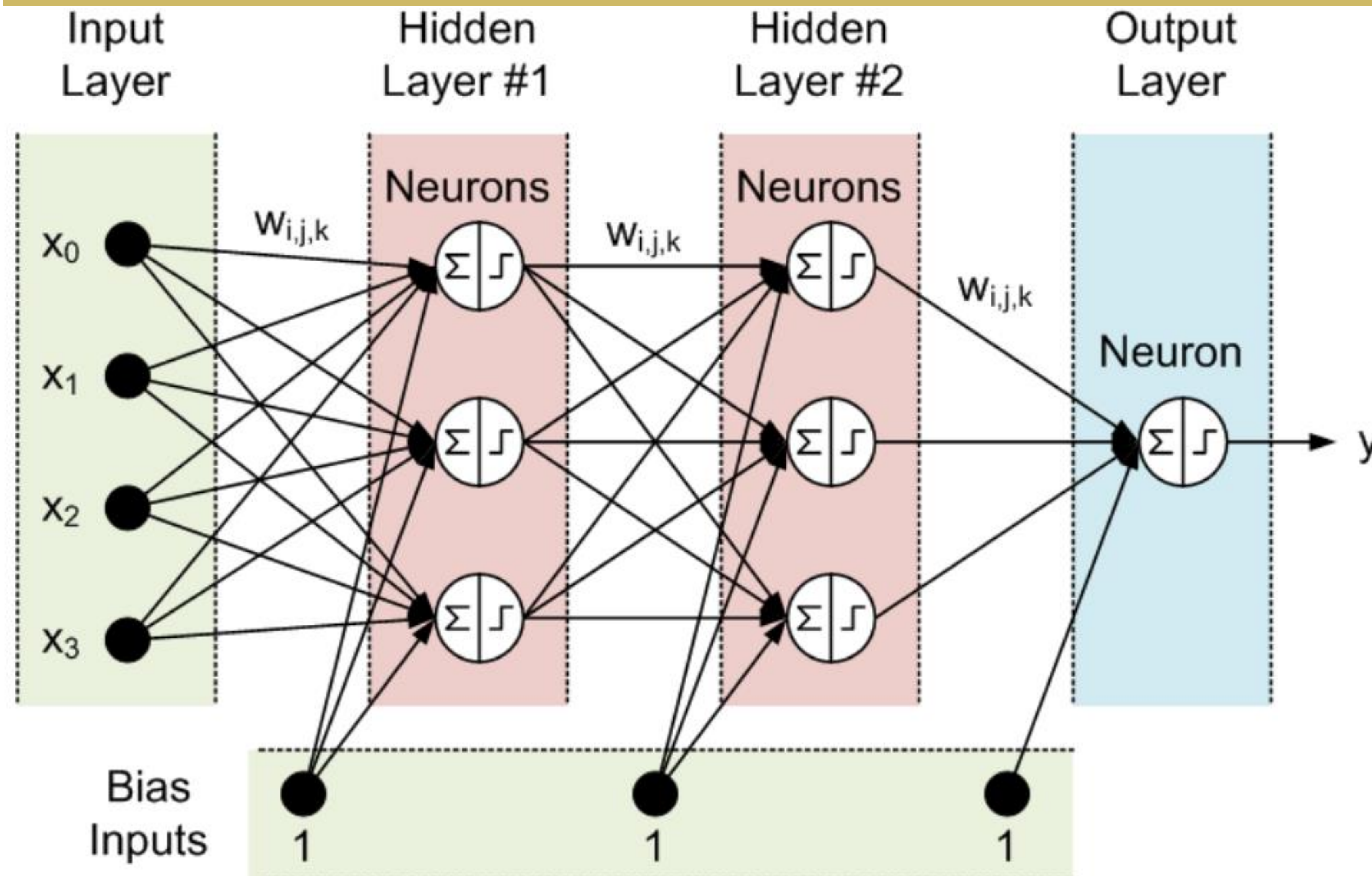
Multilayer Networks… Example



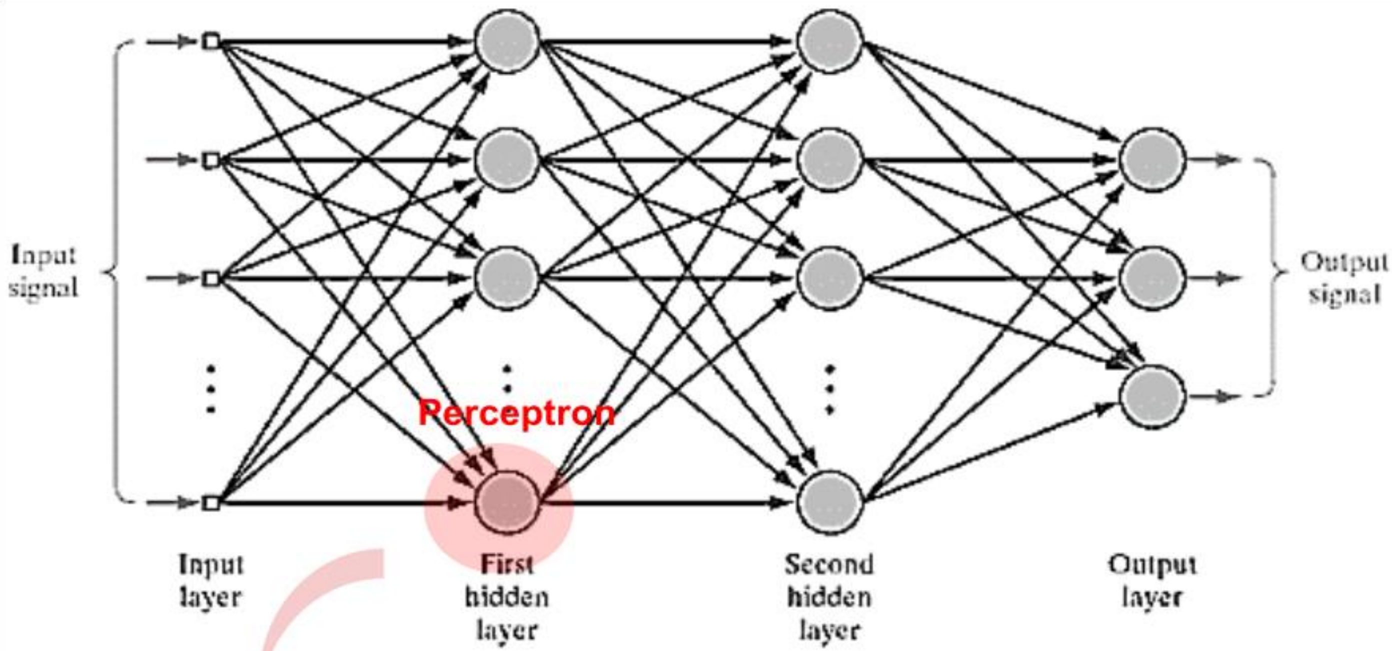
Multilayer Networks… Example

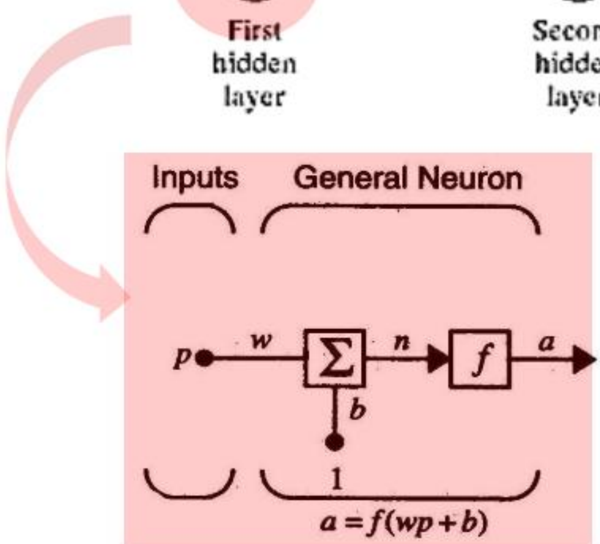


Multilayer Perceptron Architecture



Multilayer Network Architecture





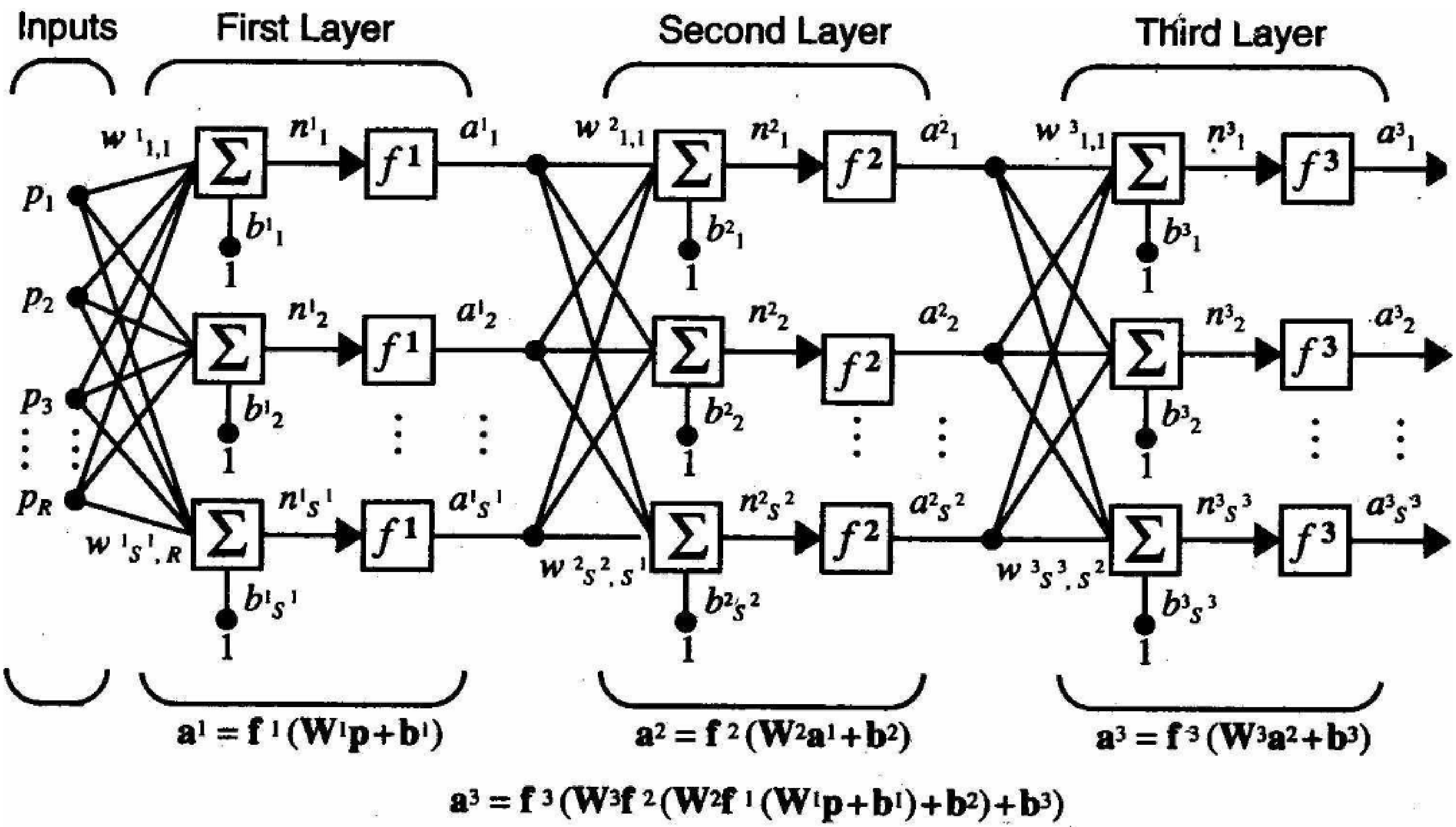
The neuron output is calculated as

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

If, for instance, $w = 3$, $p = 2$ and $b = -1.5$, then

$$a = f(3(2) - 1.5) = f(4.5)$$

Multilayer Network Architecture



MLP – A static composite (nested) function

Multilayer Networks

- The single perceptron can only express **linear decision surfaces**.

- The kind of **multilayer networks** learned by the **back propagation** algorithm are capable of expressing a rich variety of **nonlinear decision surfaces**.

Multilayer Networks

- **What type of unit shall we use as the basis for constructing multilayer networks?**

- Can we use the delta/gradient descent learning rule?
  - **multi-layers of linear units**… multiple layers of cascaded linear units still produce only linear functions, and we prefer networks capable of representing highly nonlinear functions.

- The perceptron unit is another possible choice, is it?
  - its discontinuous threshold makes it undifferentiable and hence unsuitable for gradient descent.

Multilayer Networks

Solution:

- One solution is the **sigmoid unit:**
  - a unit very much like a perceptron, but based on a **smoothed, differentiable threshold function.**
- Like the perceptron, **the sigmoid unit**,
  - first computes a linear combination of its inputs,
  - then applies a threshold to the result. However, the threshold output is a continuous function of its input.

Multilayer Networks

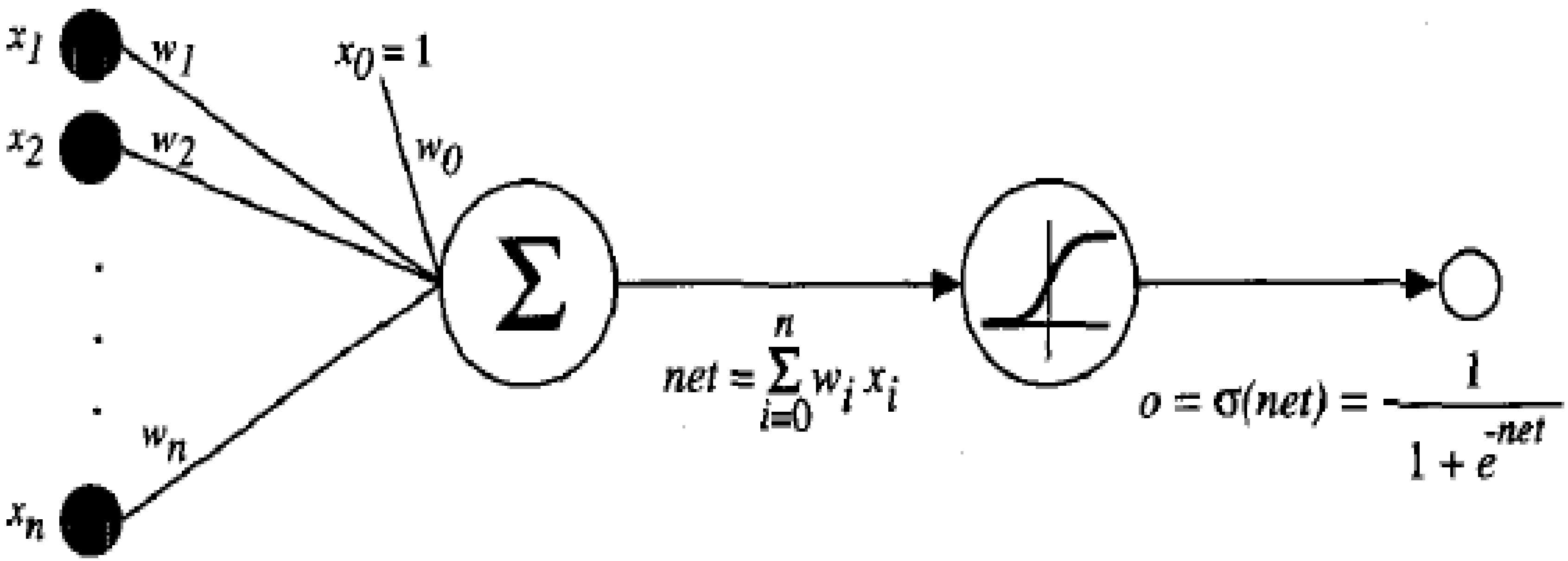
- In case of **sigmoid unit**, however, the **threshold output is a continuous function** of its input.
- More precisely, the sigmoid unit computes its output ***o*** as,

$$o = \sigma(\vec{w} \cdot \vec{x})$$

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

- σ is often called the sigmoid function or, alternatively, the logistic function.

Sigmoid Threshold Unit



Sigmoid Function

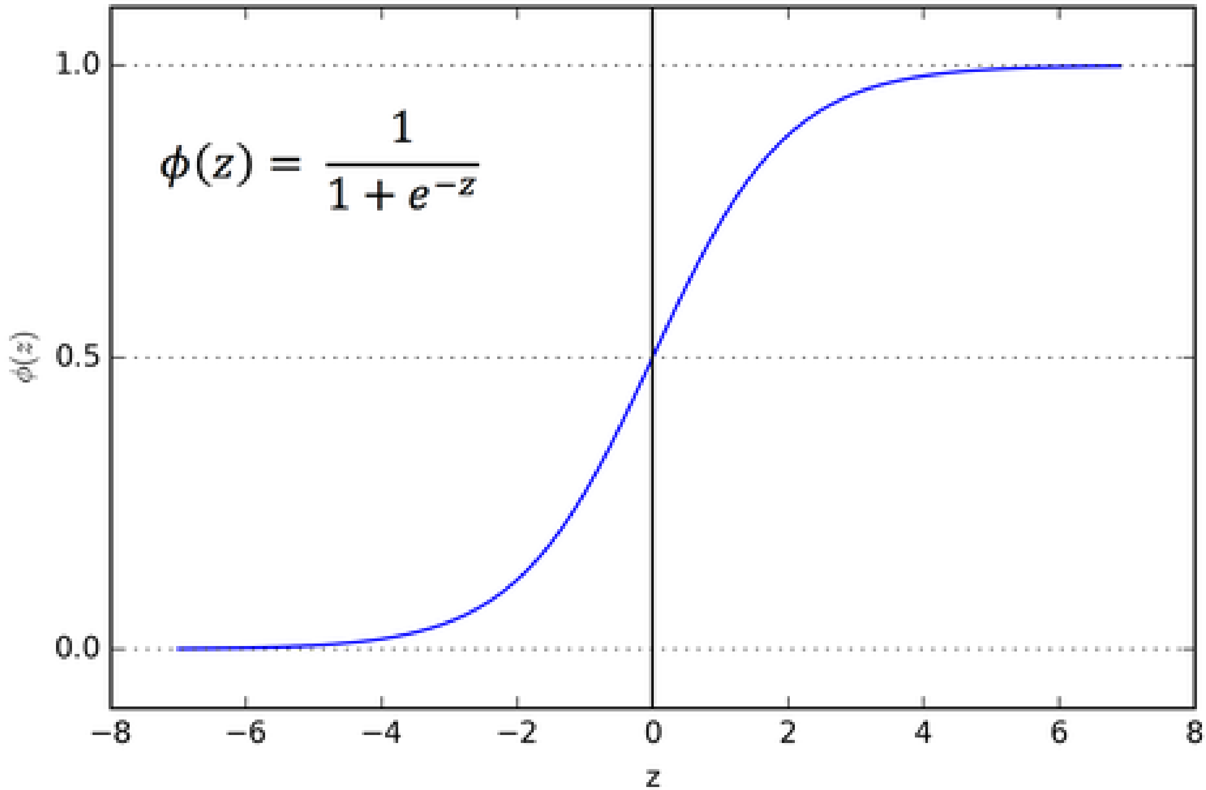
- Sigmoid function maps a very large input domain to a small range of outputs, it is often referred to as the ***squashing function*** of the unit.
- The sigmoid function has the **useful property** that **its derivative is easily expressed in terms of its output**.

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))]$$

Sigmoid Function

- Sigmoid function exists between 0 and 1.



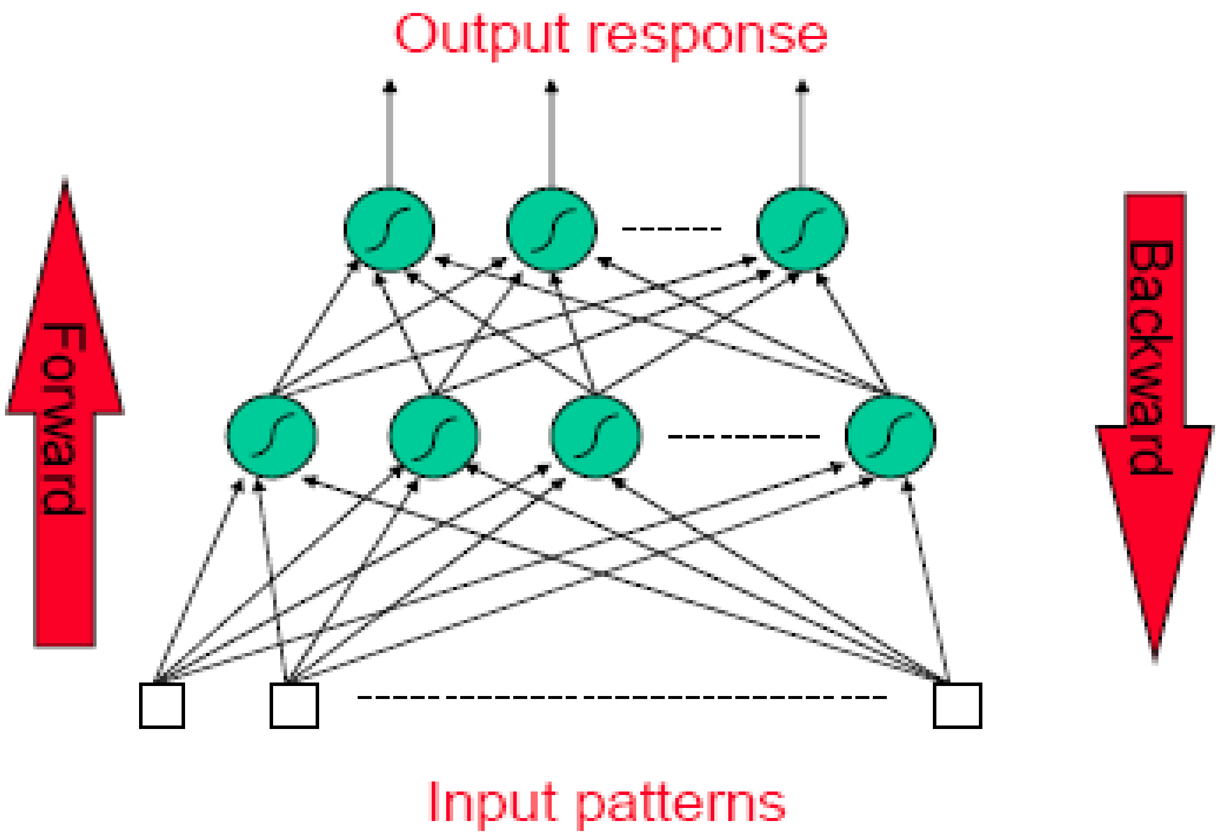
BACK PROPAGATION ALGORITHM

The Back Propagation Algorithm

The Back Propagation algorithm has two phases:

- **Forward pass phase:** computes 'functional signal', feed forward propagation of input pattern signals through network
- **Backward pass phase:** computes 'error signal', *propagates* the error *backwards* through network starting at output units
  - (where the error is the difference between actual and desired output values)

The Back Propagation Algorithm



Conceptually: Forward Activity - Backward Error

The Back Propagation Algorithm

- The back propagation algorithm learns the weights for a multilayer network,
  - given a network with a fixed set of units and interconnections.

- It **employs gradient descent** to attempt *to minimize the squared error* between the network output values and the target values for these outputs.

- As we are considering networks with multiple output units, we begin by redefining **E** to sum the errors over all of the network output units.

The Back Propagation Algorithm

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

- where ***outputs*** is the *set of output units* in the network, and t_{kd} and O_{kd} are the target and output values associated with the *k*th output unit and training example **d**.

The Back Propagation Algorithm

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$*, where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i into unit j is denoted x_{ji}*, and the weight from unit i to unit j is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between −.05 and .05).
- Until the termination condition is met, Do

The Back Propagation Algorithm

For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

Reading Material

- **Artificial Intelligence,** A Modern Approach

  Stuart J. Russell and Peter Norvig

  - **Chapter 18.**
- **Machine Learning**

  Tom M. Mitchell

  - **Chapter 4.**