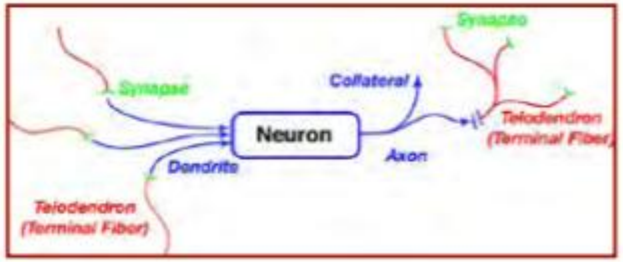
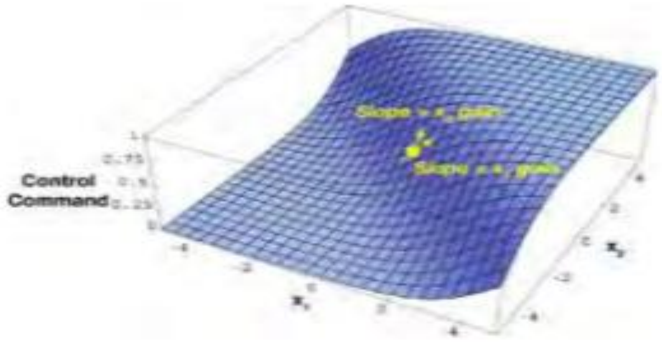
Artificial Neural Network

Introduction to Neural Networks

- Natural and artificial neurons
- Natural and computational neural networks
  - Linear network
  - Perceptron
  - Sigmoid network
  - Radial basis function
- Applications of neural networks
- Supervised training
  - Left pseudoinverse
  - Steepest descent
  - Back-propagation



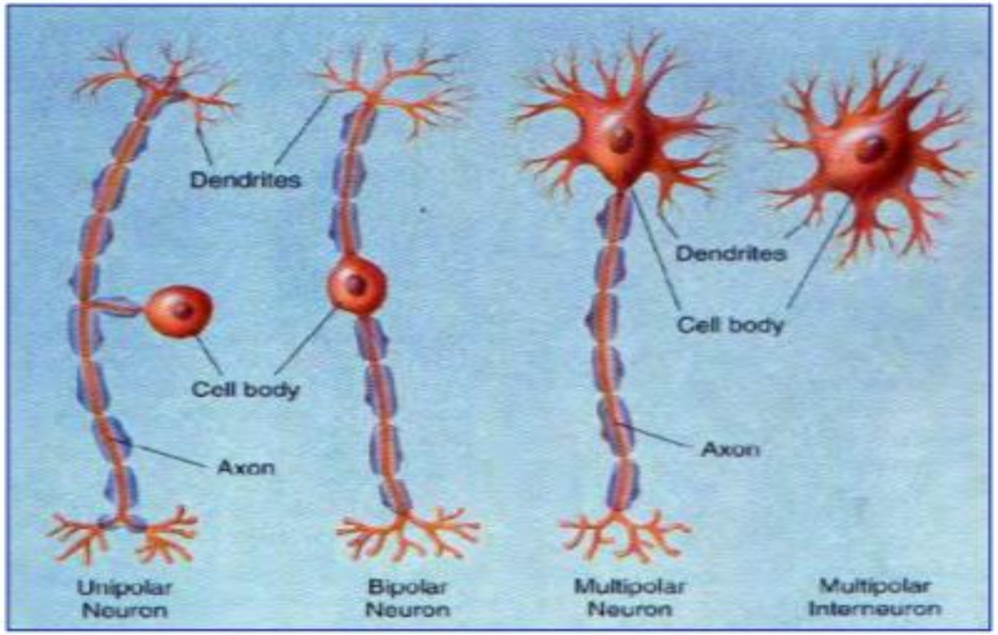
Applications of Computational Neural Networks



- Classification of data sets
- Image processing
- Language interpretation
- Nonlinear function approximation
  - Efficient data storage and retrieval
  - System identification
- Nonlinear and adaptive control systems

Neurons

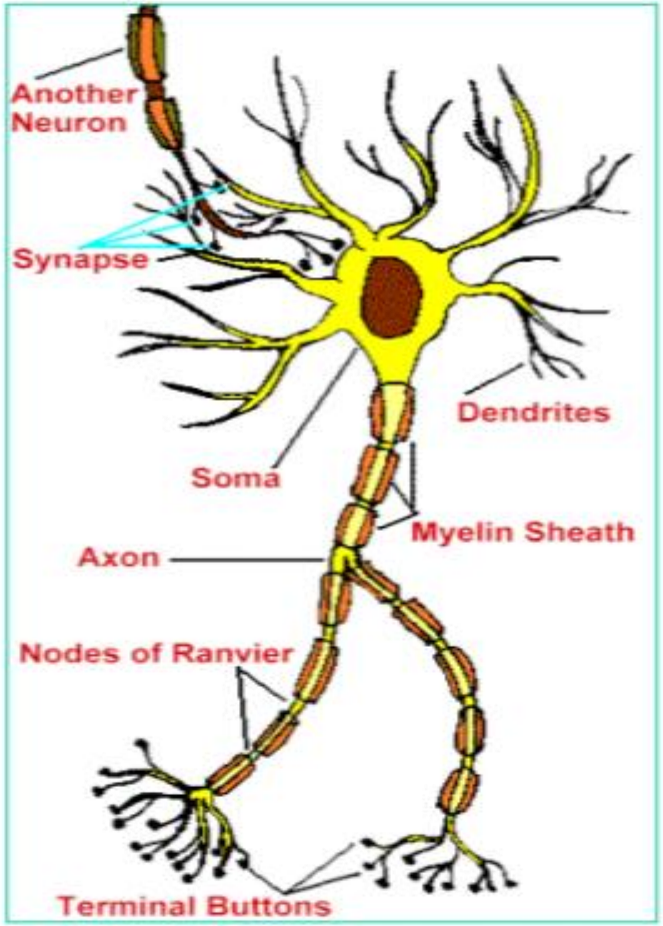
- Biological cells with significant electrochemical activity
- ~10-100 billion neurons in the brain
- Inputs from thousands of other neurons
- Output is scalar, but may have thousands of branches



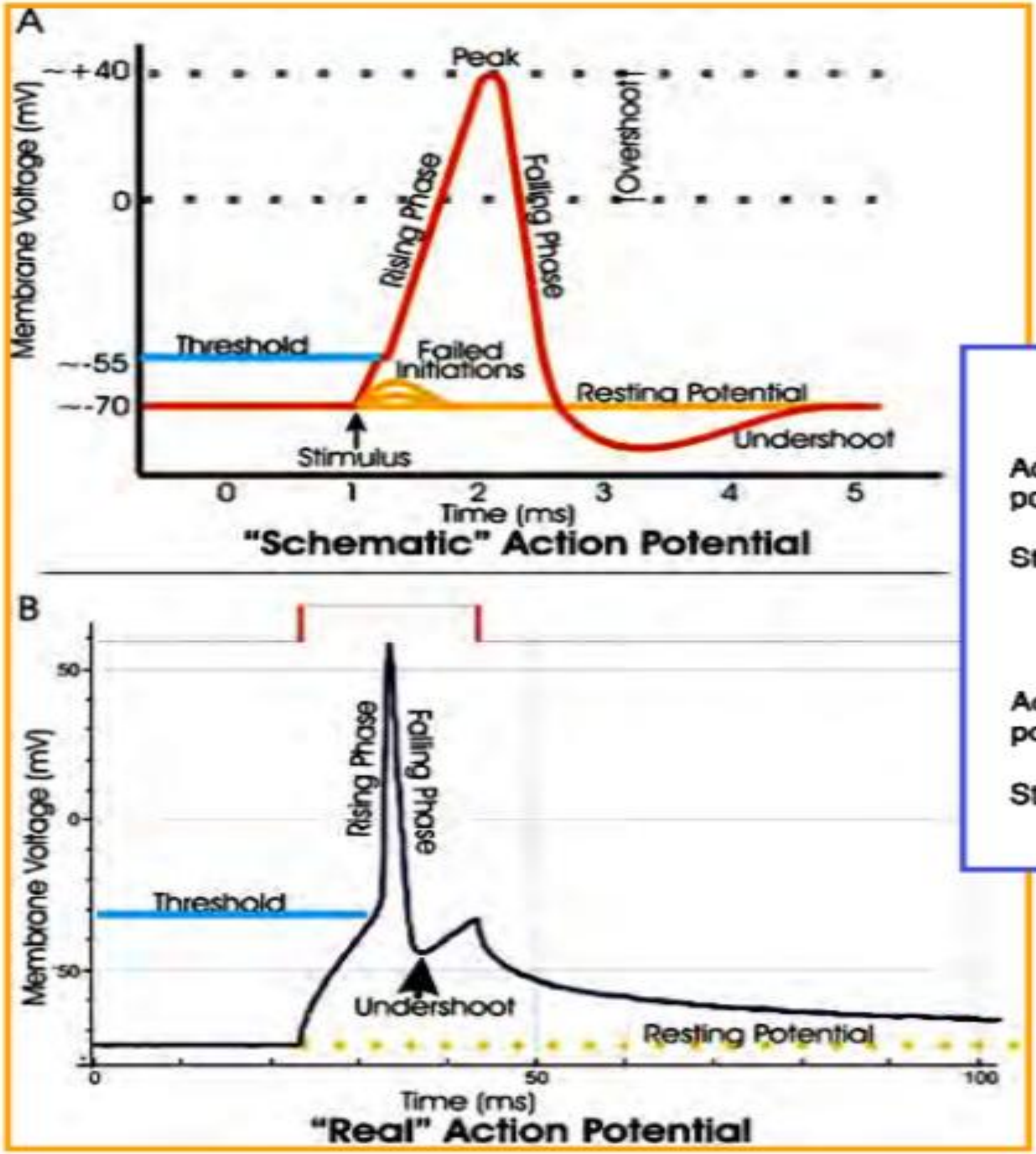
- Afferent (sensor) neurons send signals from organs and the periphery to the central nervous system
- Efferent (motor) neurons issue commands from the CNS to effector (e.g., muscle) cells
- Interneurons send signals between neurons in the central nervous system
- Signals are ionic, i.e., chemical (neurotransmitter atoms and molecules) and electrical (charge)

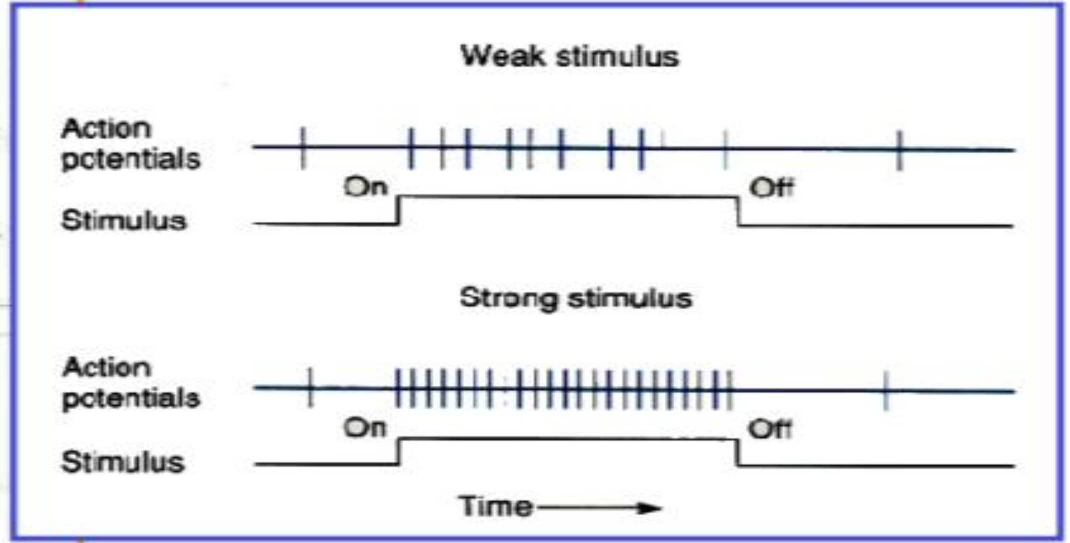
Activation Input to Soma Causes Change in Output Potential

- Stimulus from
  - Receptors
  - Other neurons
  - Muscle cells
  - Pacemakers (c.g. cardiac sino-atrial node)
- When membrane potential of neuronal cell exceeds a threshold
  - Cell is polarized
  - Action potential pulse is transmitted from the cell
  - Activity measured by amplitude and firing frequency of pulses
  - Saltatory conduction along axon
    - Myelin Schwann cells insulate axon
    - Signal boosted at Nodes of Ranvier
- Cell depolarizes and potential returns to rest



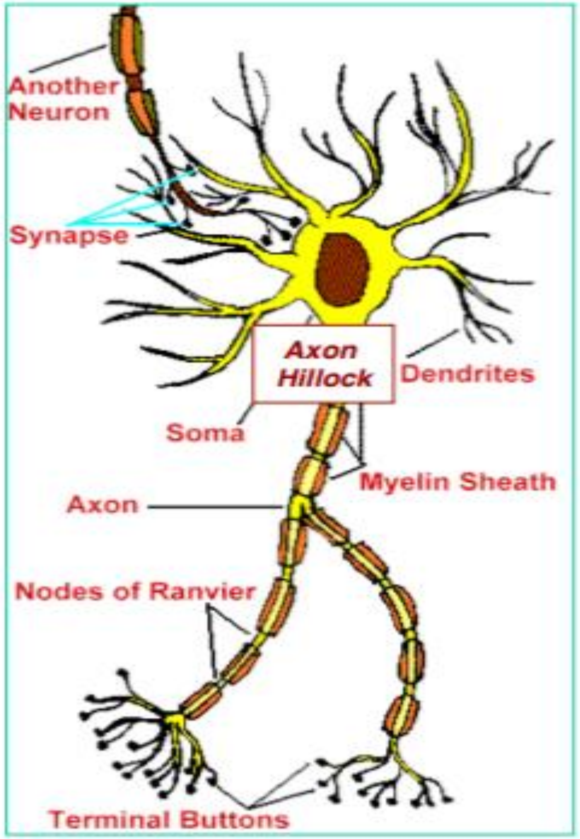
Neural Action Potential

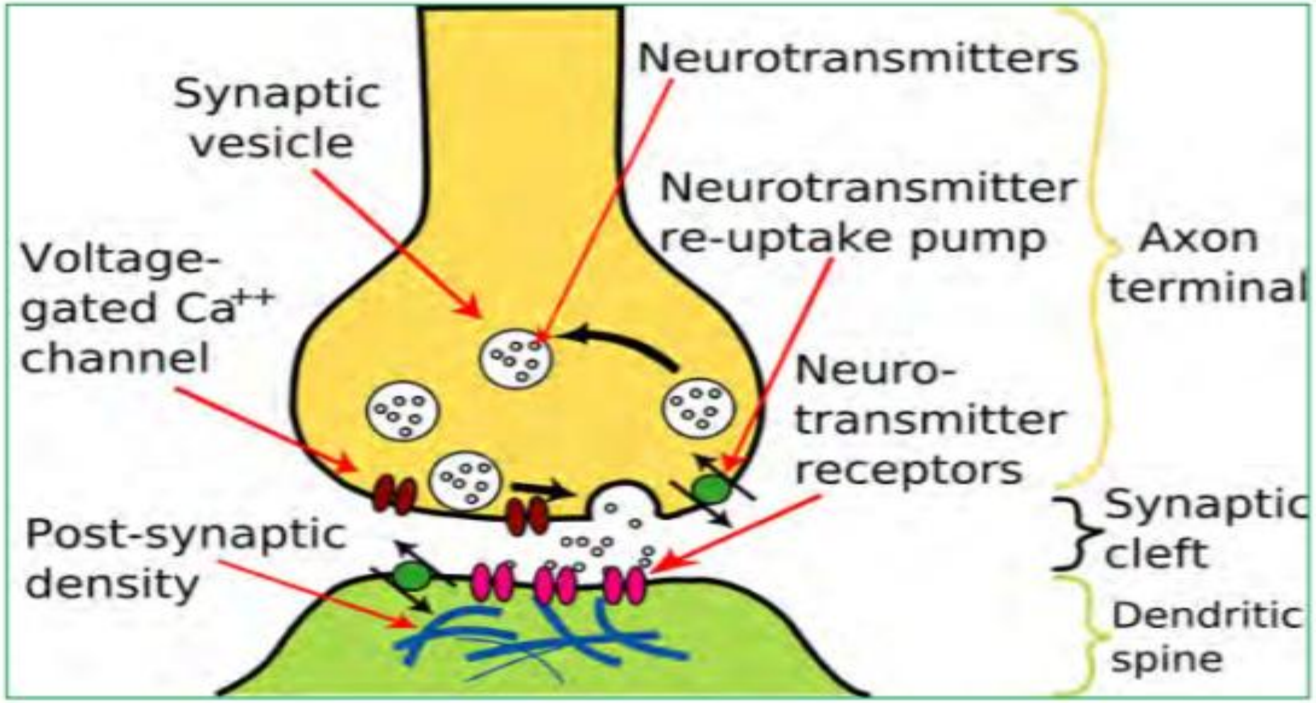




- **Maximum Firing Rate**: 500/sec
- **Refractory Period**: Minimum time increment between action potential firing ~ 1-2 msec

Electrochemical Signaling at Axon Hillock and Synapse

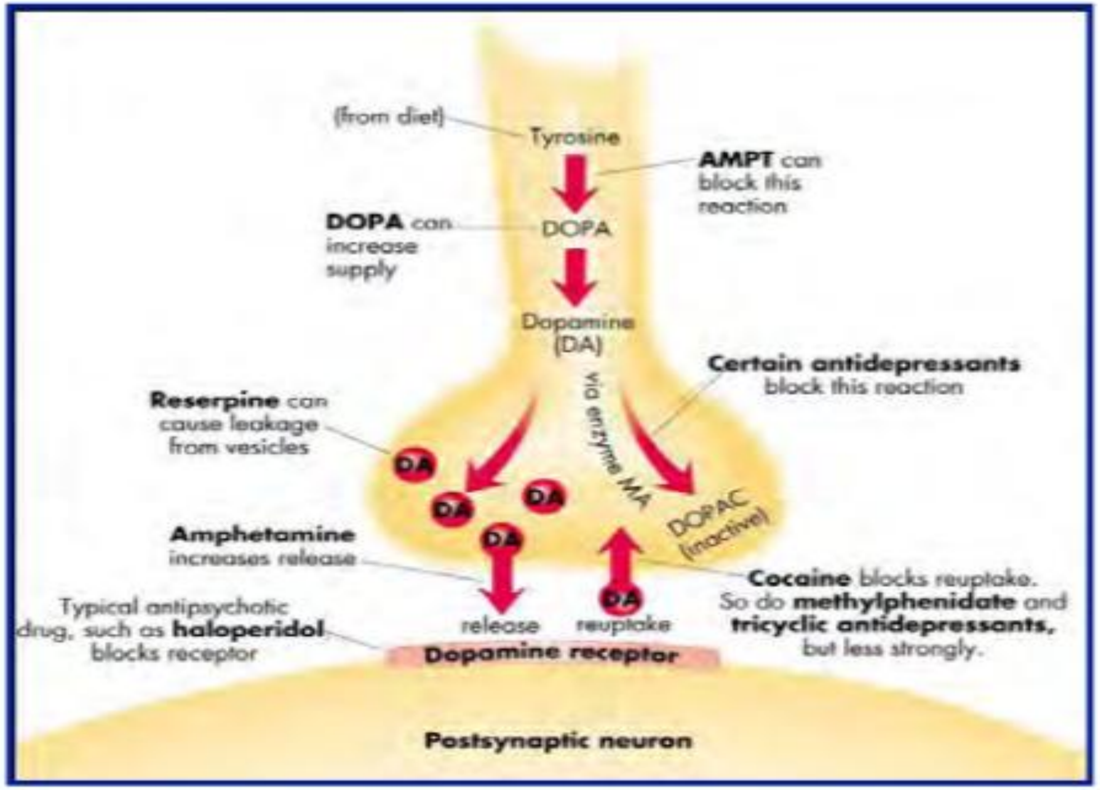




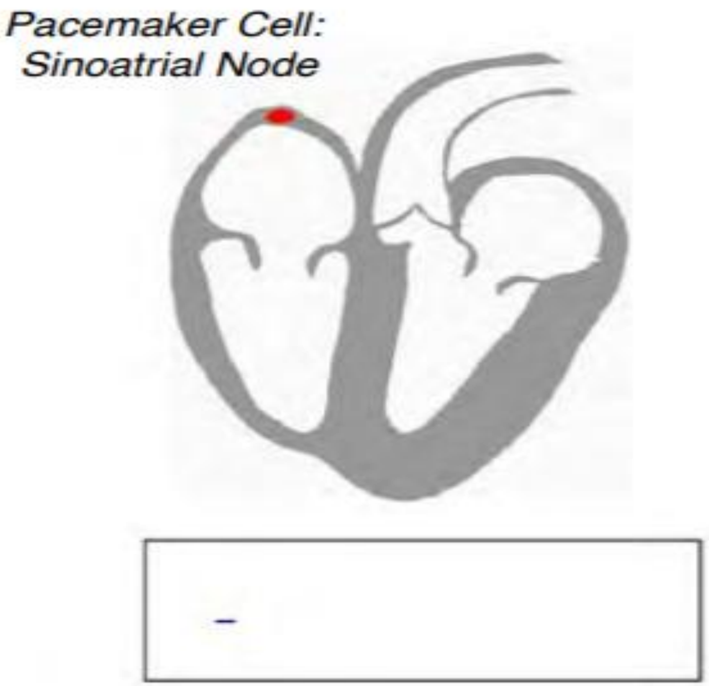
Synaptic Strength Can Be Increased or Decreased by Externalities

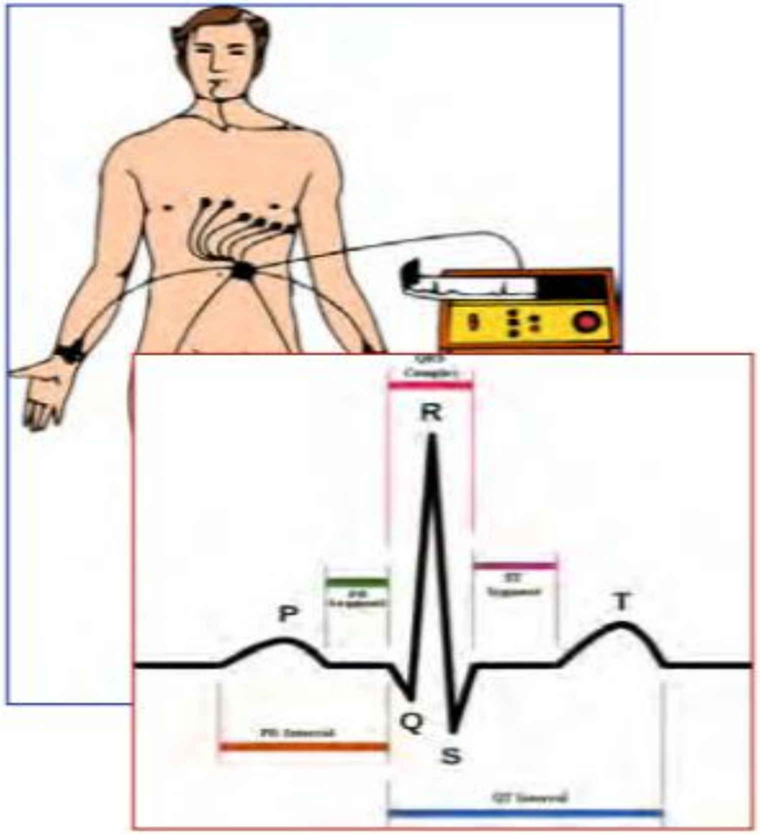
- **Synapses: learning elements of the nervous system**
  - Action potentials enhanced or inhibited
  - Chemicals can modify signal transfer
  - Potentiation of pre- and post-synaptic cells

- **Adaptation/Learning (potentiation)**
  - Short-term
  - Long-term

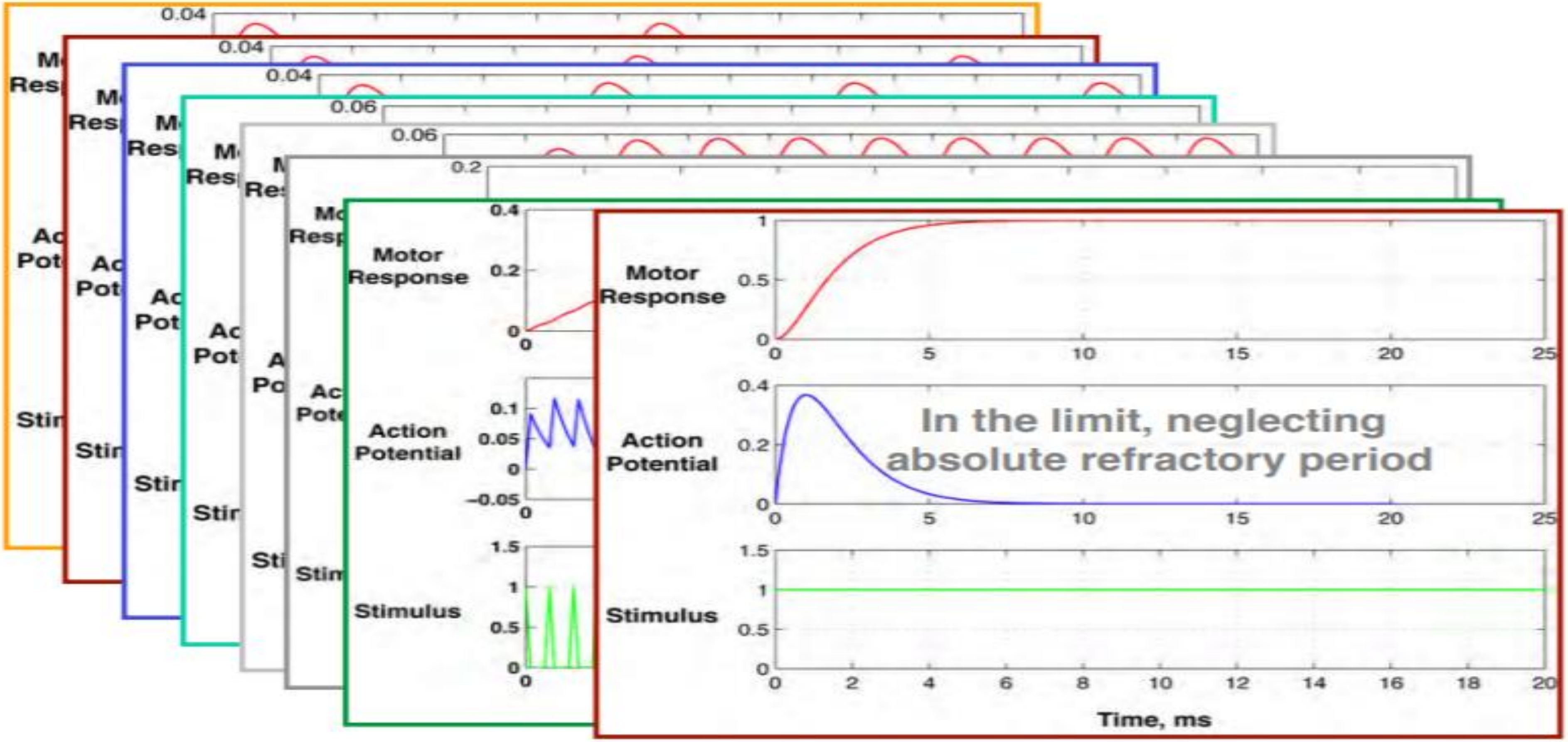


Cardiac Pacemaker and EKG Signals

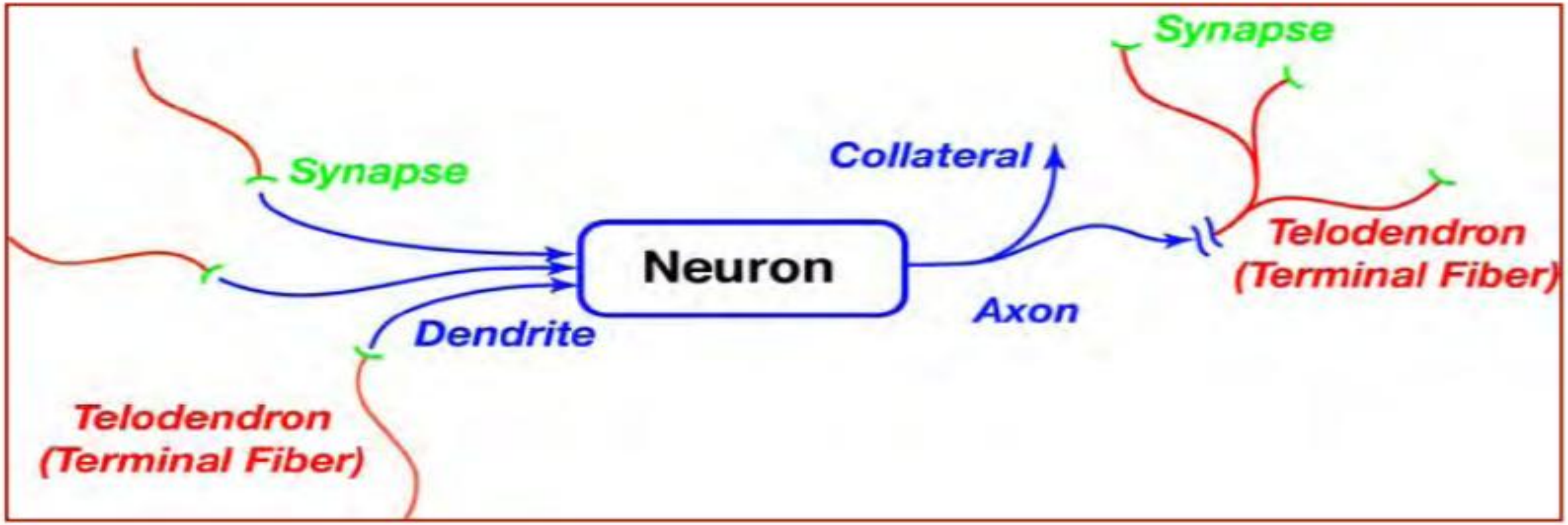




Impulse, Pulse-Train, and Step Response of LTI 2nd-Order Neural Model



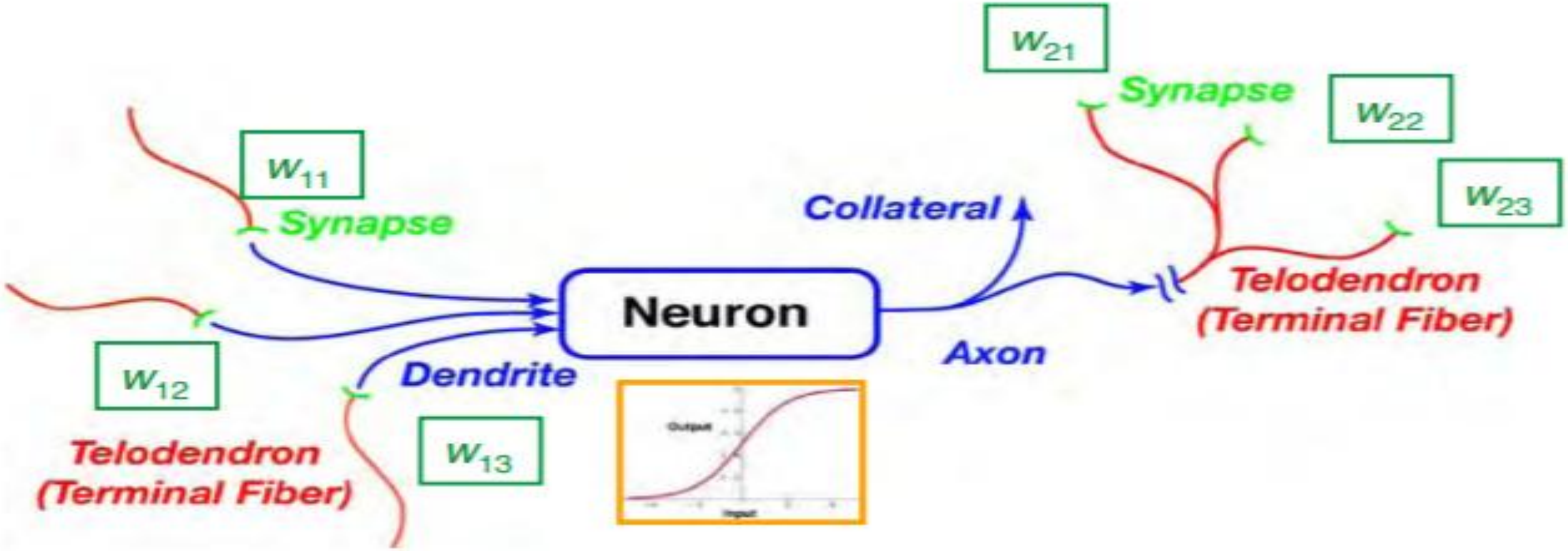
Multipolar Neuron



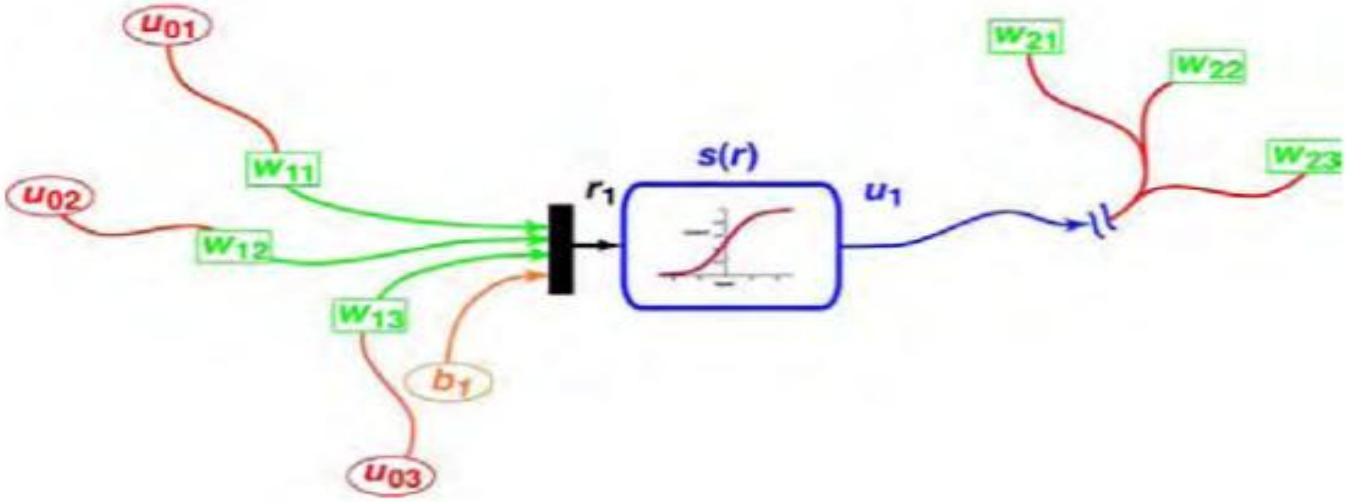
Mathematical Model of Neuron Components

Synapse effects represented by weights (gains or multipliers)

Neuron firing frequency is modeled by linear gain or nonlinear element



The Neuron Function



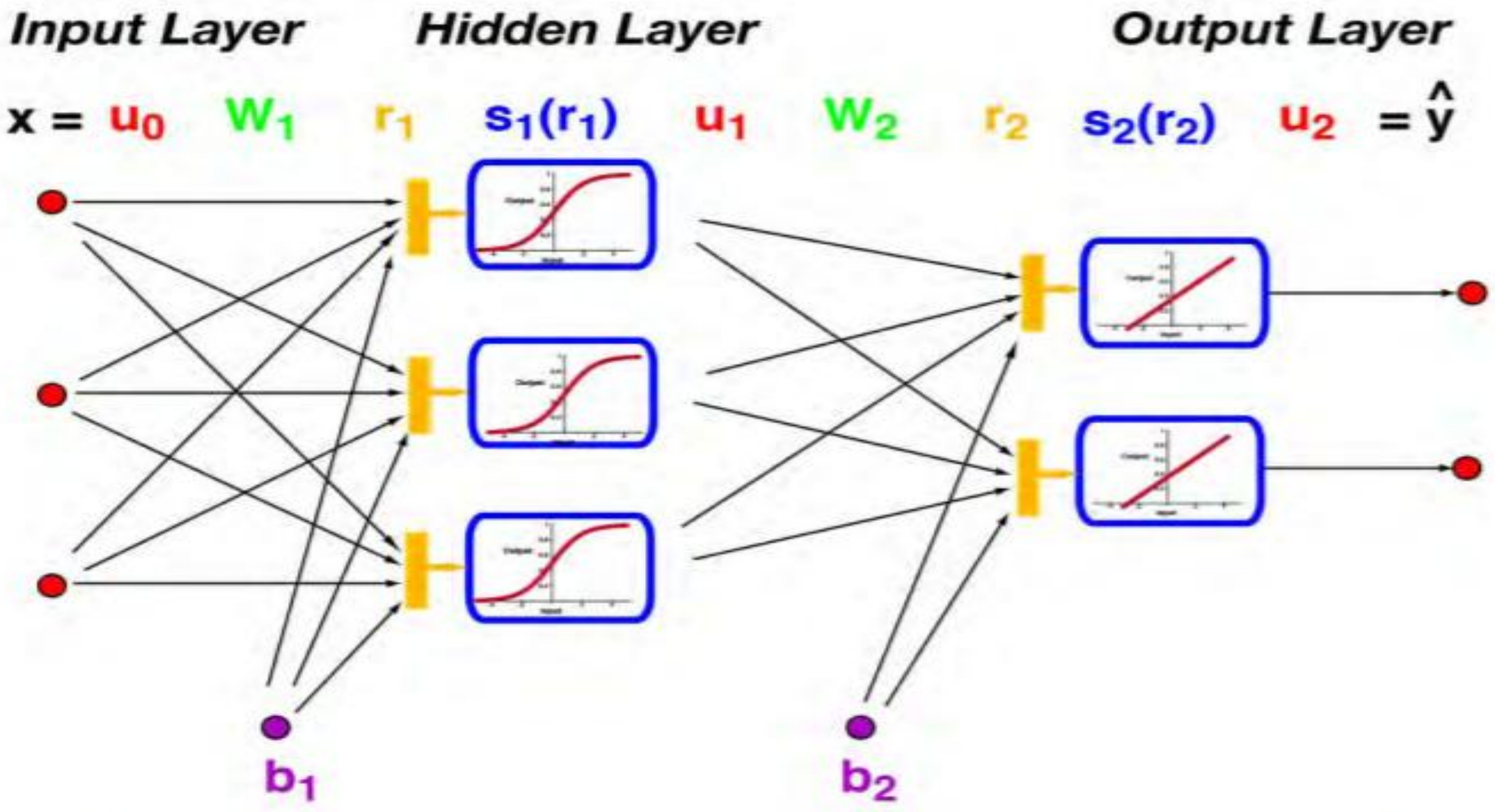
- Vector input, **u**, to a single neuron
  - Sensory input or output from upstream neurons
- Linear operation produces scalar, *r*
- Add bias, *b*, for zero adjustment

$$r = \mathbf{w}^T \mathbf{u} + b$$

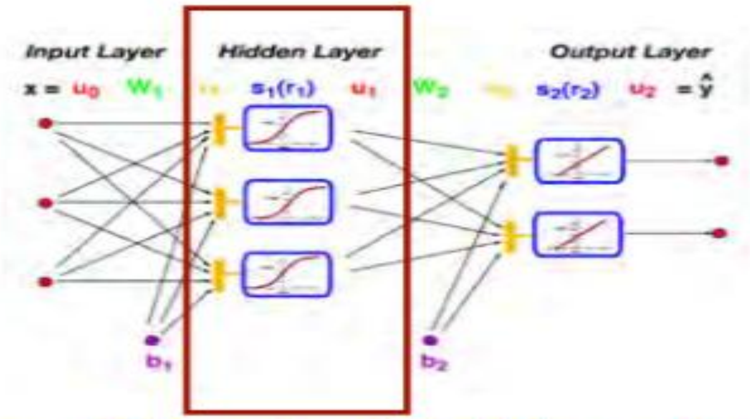
- Scalar output, *u*, of a single neuron (or node)
  - Scalar linear or nonlinear operation, *s*(*r*)

$$u = s(r)$$

"Shallow" Neural Network



Layered, parallel structure for computation



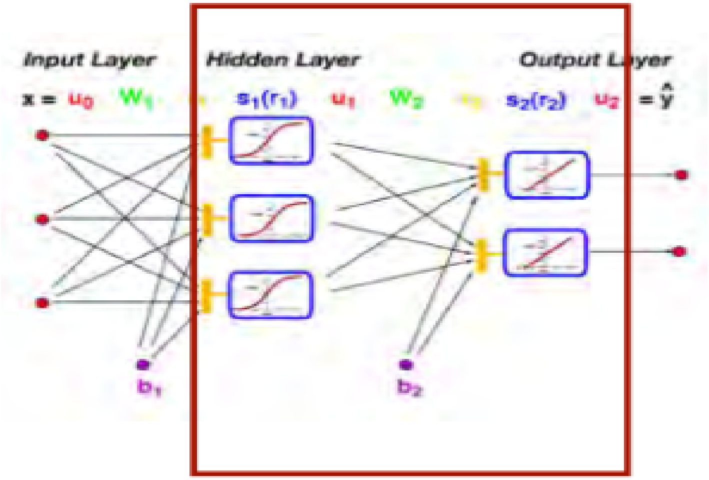
Input-Output Characteristics of a Neural Network Layer

- **Single hidden layer**
  - Number of inputs = *n*
    - dim(u) = (*n* x 1)
  - Number of nodes = *m*
    - dim(r) = dim(b) = dim(s) = (*m* x 1)

$$\mathbf{r} = \mathbf{W}\mathbf{u} + \mathbf{b}$$

$$\mathbf{u} = \mathbf{s}(\mathbf{r})$$

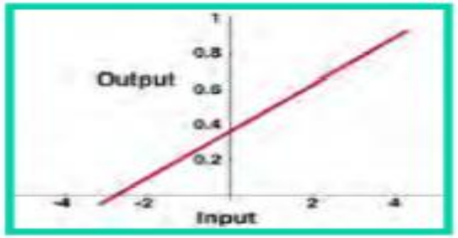
$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1{}^T \\ \mathbf{w}_2{}^T \\ \cdots \\ \mathbf{w}_n{}^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$



Two-Layer Network

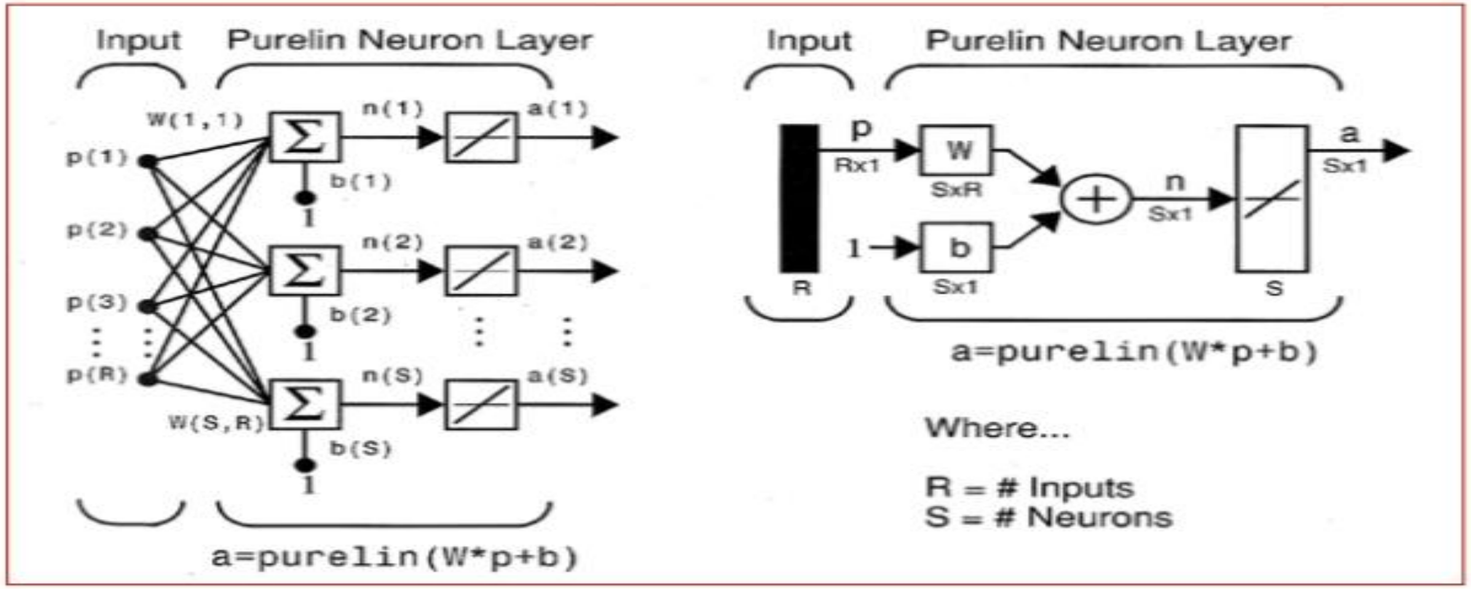
- Two layers
  - Node functions may be different, e.g.,
    - Sigmoid hidden layer
    - Linear output layer
  - Number of nodes in each layer need not be the same
- Input sometimes labeled as layer

$$\begin{aligned}
\mathbf{y} &= \mathbf{u}_2 \\
&= \mathbf{s}_2(\mathbf{r}_2) = \mathbf{s}_2(\mathbf{W}_2\mathbf{u}_1 + \mathbf{b}_2) \\
&= \mathbf{s}_2\left[\mathbf{W}_2\, \mathbf{s}_1(\mathbf{r}_1) + \mathbf{b}_2\right] \\
&= \mathbf{s}_2\left[\mathbf{W}_2\, \mathbf{s}_1(\mathbf{W}_1\mathbf{u}_0 + \mathbf{b}_1) + \mathbf{b}_2\right] \\
&= \mathbf{s}_2\left[\mathbf{W}_2\, \mathbf{s}_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\right]
\end{aligned}$$



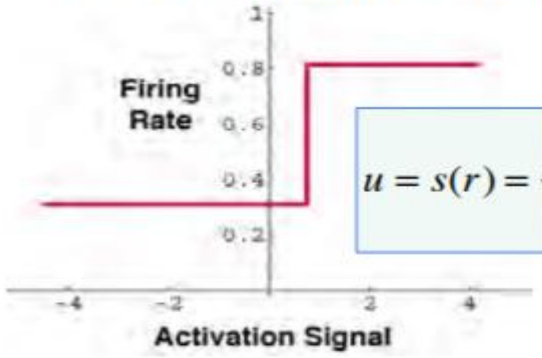
Linear Neural Network

- Outputs provide linear scaling of inputs
- Equivalent to matrix transformation of a vector, $y = Wx + b$
- Easy to train (left pseudoinverse, *TBD*)
- MATLAB symbology



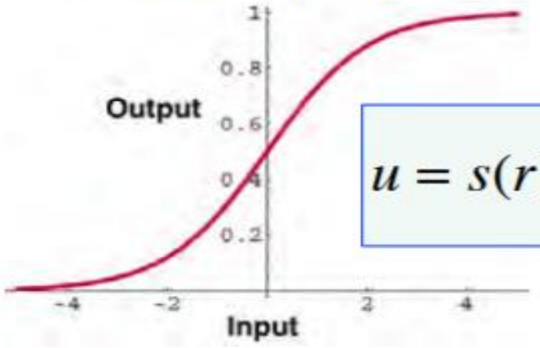
Idealizations of Nonlinear Neuron Input-Output Characteristic

Step function ("Perceptron")



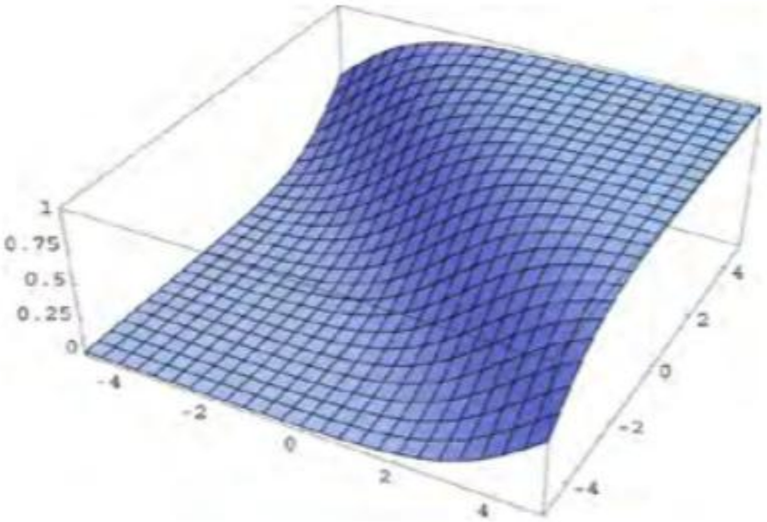
$$u = s(r) = \begin{cases} 1, & r > 0 \\ 0, & r \le 0 \end{cases}$$

Logistic sigmoid function



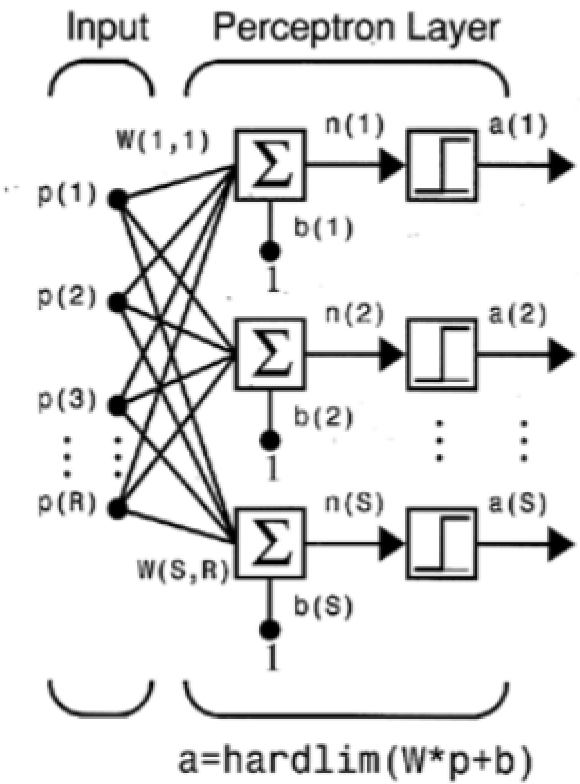
$$u = s(r) = \frac{1}{1+e^{-r}}$$

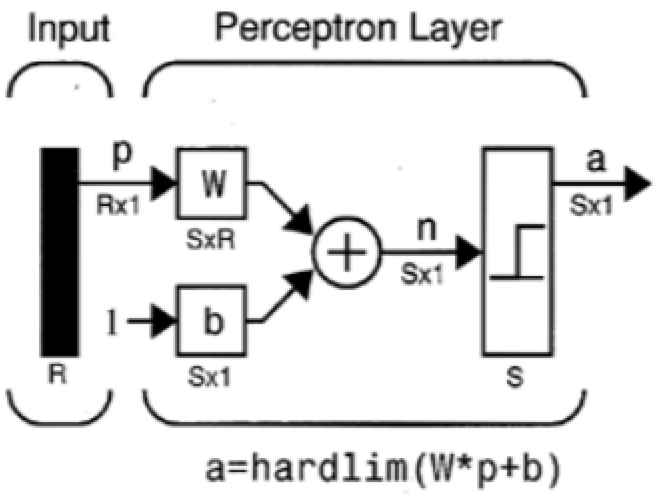
Sigmoid with two inputs, one output



$$u = s(r) = \frac{1}{1+e^{-(w_1 r_1 + w_2 r_2 + b)}}$$

Perceptron Neural Network





Where...

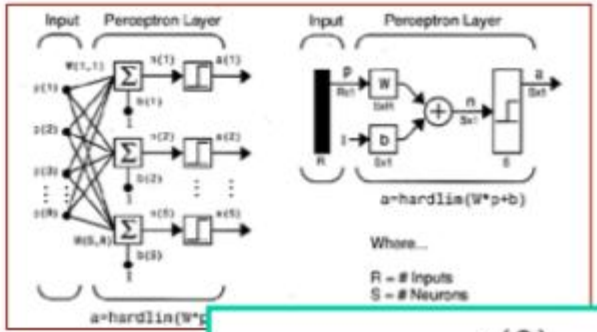
R = # Inputs
S = # Neurons

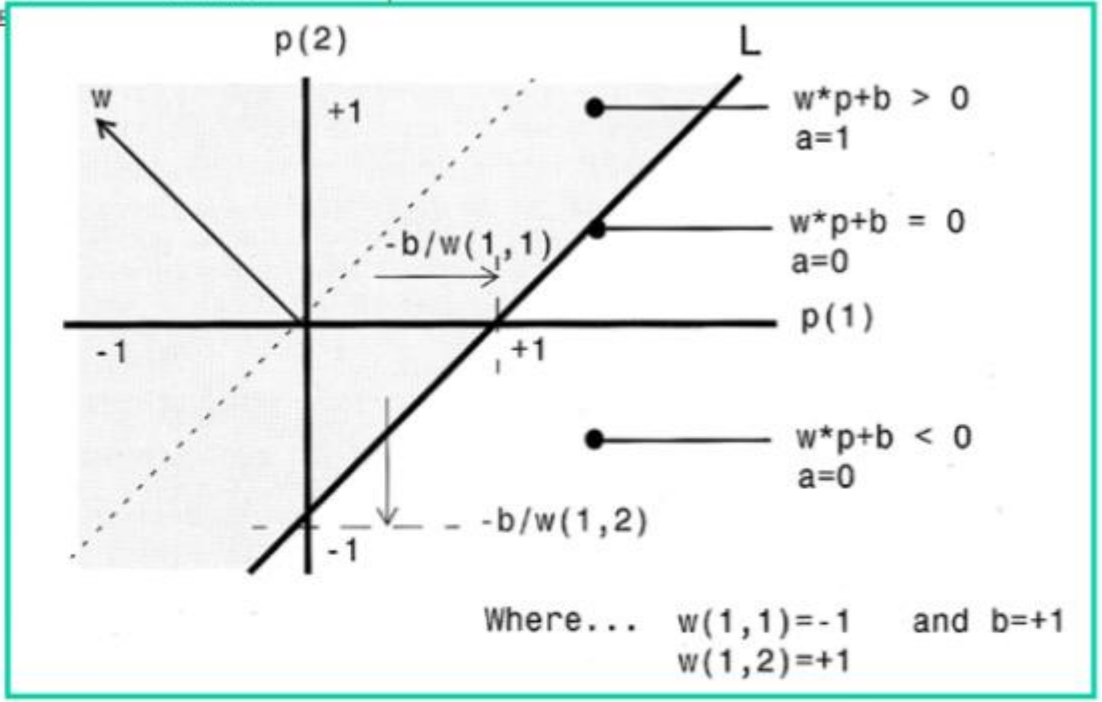
Each node is a step function

Weighted sum of features is fed to each node

Each node produces a linear classification of the input space

Perceptron Neural Network





Weights adjust slopes
Biases adjust zero crossing points

Single-Layer, Single-Node Perceptron Discriminants

Perceptron Function

$$u = s(\mathbf{w}^T\mathbf{x}+b) = \begin{cases} 1, & (\mathbf{w}^T\mathbf{x}+b) > 0 \\ 0, & (\mathbf{w}^T\mathbf{x}+b) \le 0 \end{cases}$$

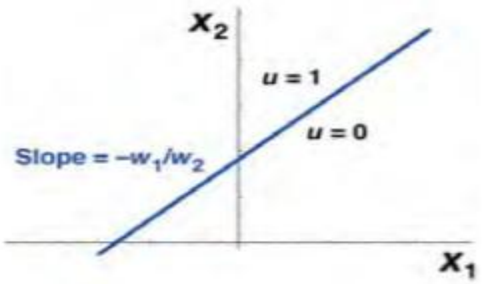
Two inputs, single step function

Discriminant

$$0 = w_1x_1 + w_2x_2 + b$$

$$x_2 = \frac{-1}{w_2}\left(w_1x_1 + b\right)$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



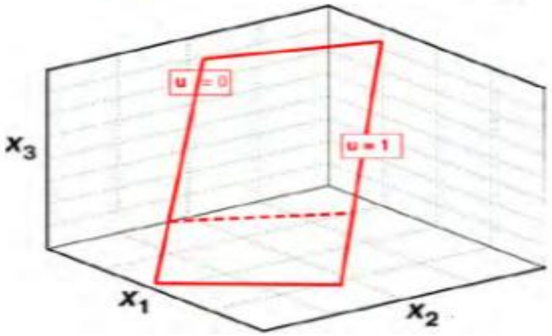
Three inputs, single step function

Discriminant

$$0 = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$x_3 = \frac{-1}{w_3}\left(w_1x_1 + w_2x_2 + b\right)$$

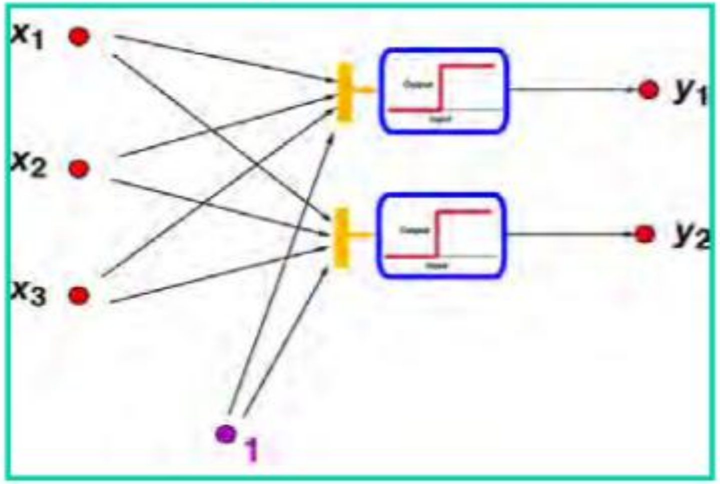
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

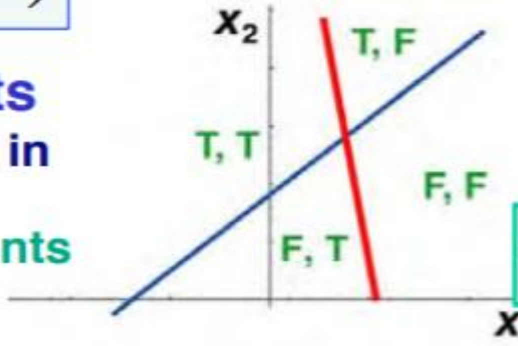


Single-Layer, Multi-Node Perceptron Discriminants

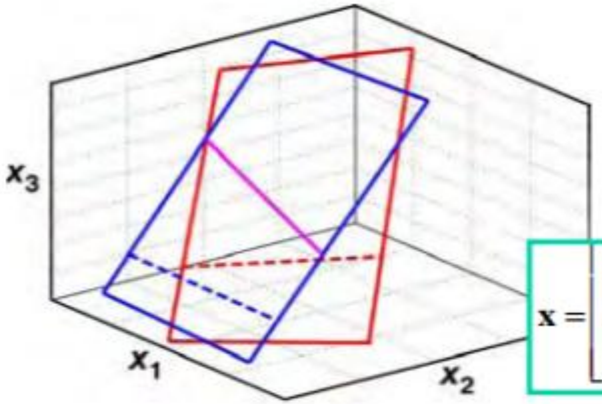
$$\mathbf{u} = \mathbf{s}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Multiple inputs, nodes, and outputs
  - More inputs lead to more dimensions in discriminants
  - More outputs lead to more discriminants





$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



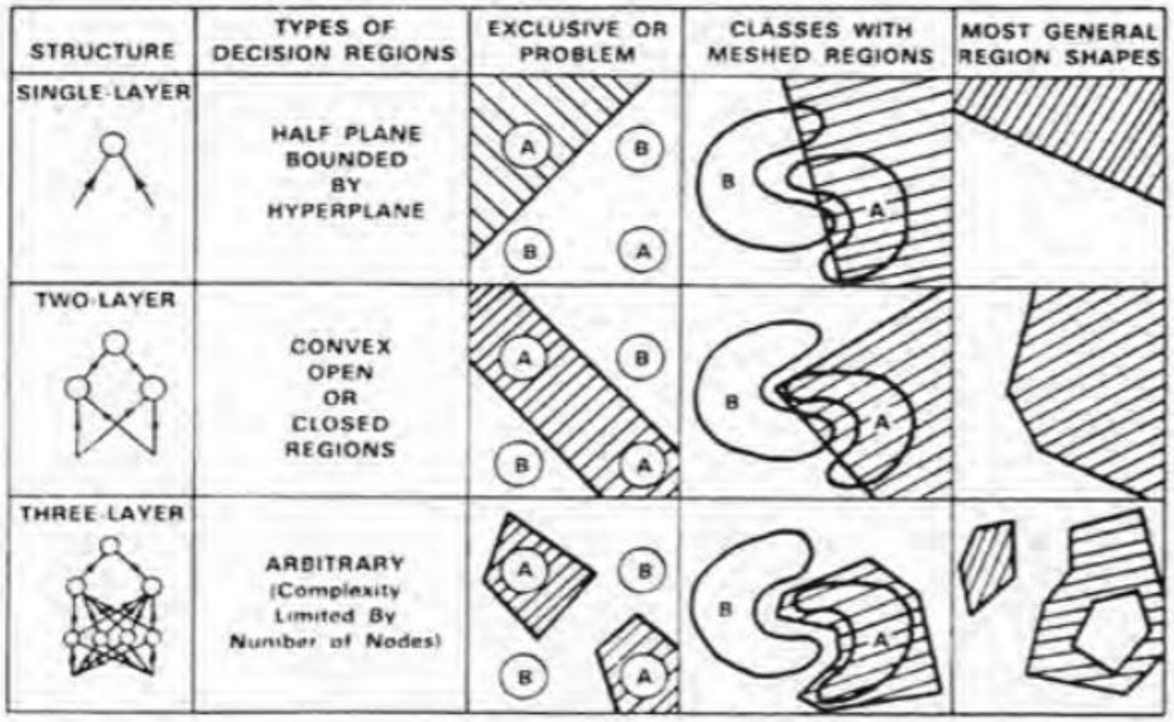
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

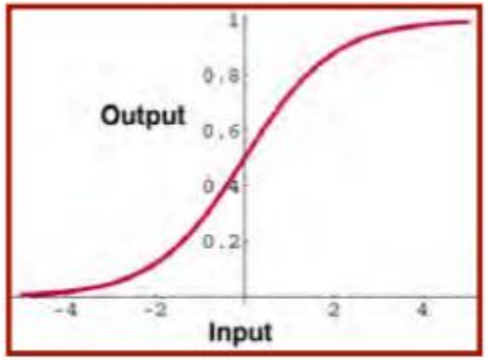
Multi-Layer Perceptrons Can Classify With Boundaries or Clusters

Classification capability of multi-layer perceptrons

Classifications of classifications

Open or closed regions





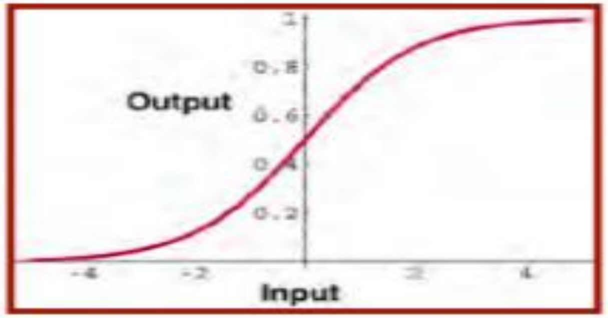
Sigmoid Activation Functions

- **Alternative sigmoid functions**
  - Logistic function: 0 to 1
  - Hyperbolic tangent: –1 to 1
  - Augmented ratio of squares: 0 to 1
- **Smooth nonlinear functions that limit extreme values in output**

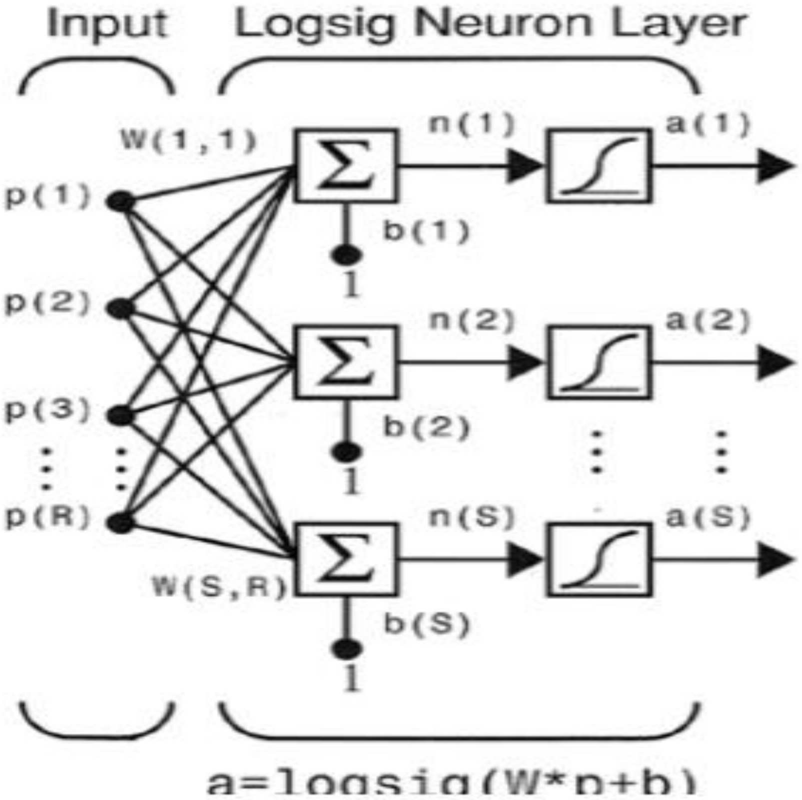
$$u = s(r) = \frac{1}{1+e^{-r}}$$

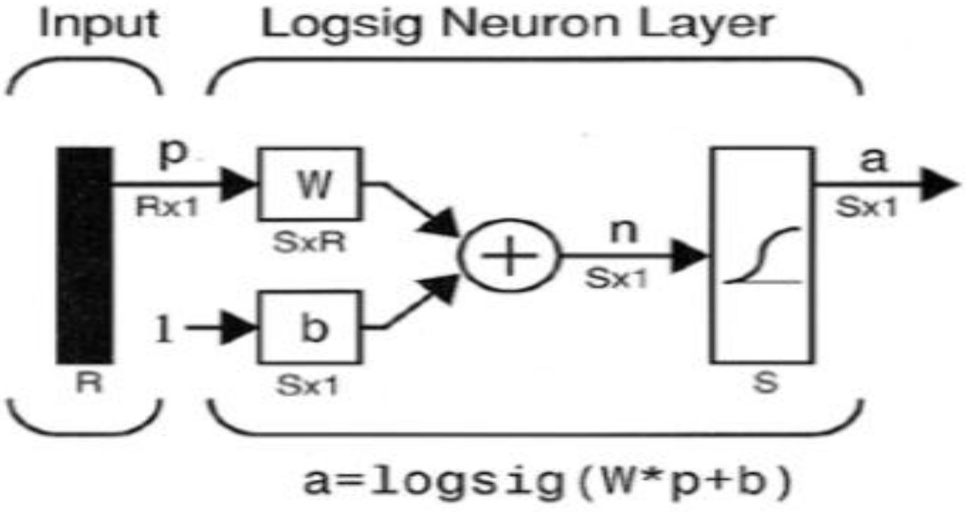
$$u = s(r) = \tanh r = \frac{1-e^{-2r}}{1+e^{-2r}}$$

$$u = s(r) = \frac{r^2}{1+r^2}$$



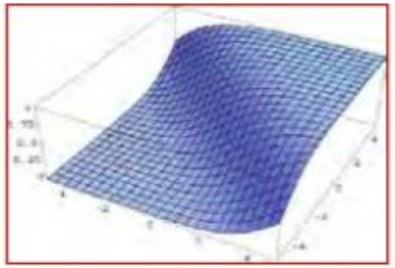
Single-Layer Sigmoid Neural Network





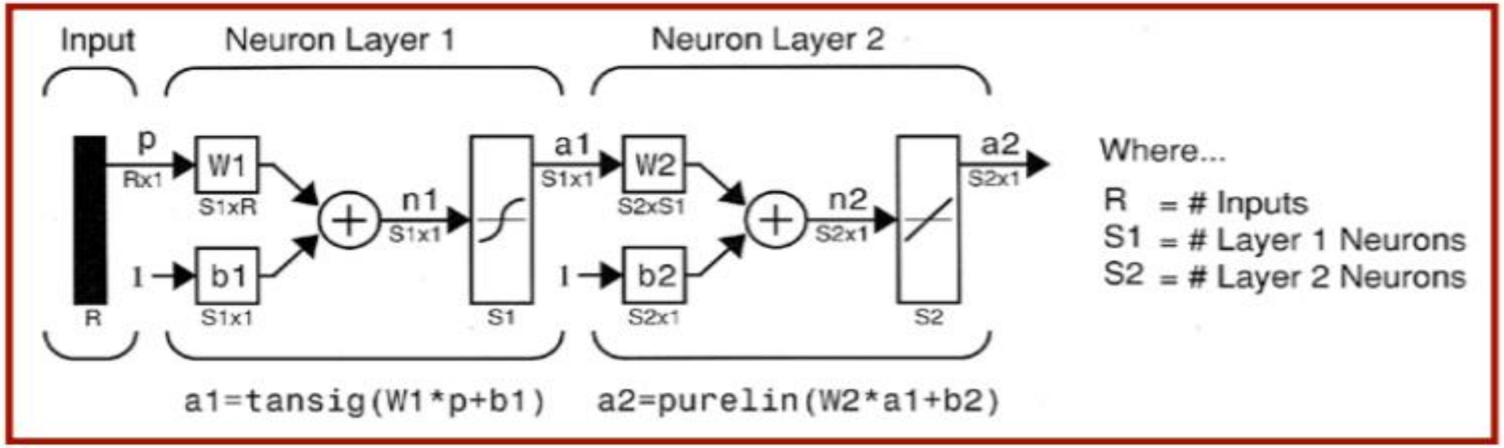
Where...

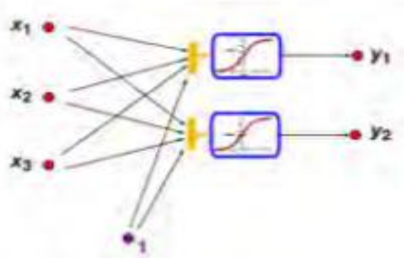
R = # Inputs
S = # Neurons



Fully Connected Two-Layer (Single-Hidden-Layer) Sigmoid Layer

- Sufficient to approximate any continuous function
- All nodes of one layer connected to all nodes of adjacent layers
- Typical sigmoid network contains
  - Single sigmoid hidden layer (nonlinear fit)
  - Single linear output layer (scaling)

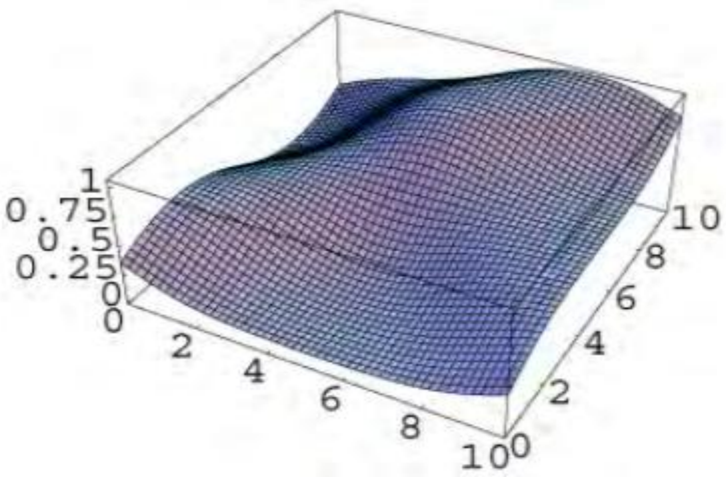


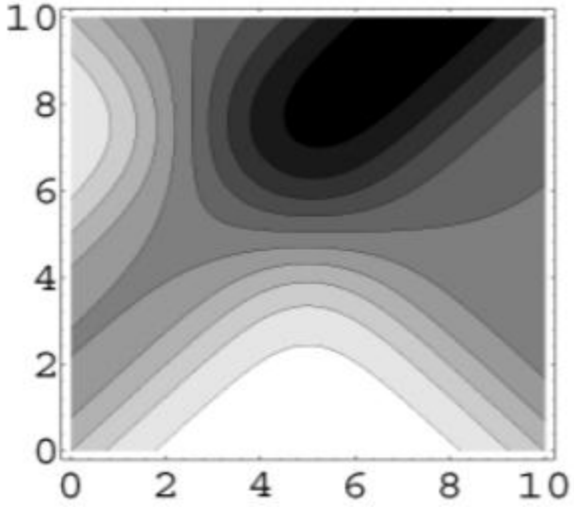


Typical Output for Two-Sigmoid Network

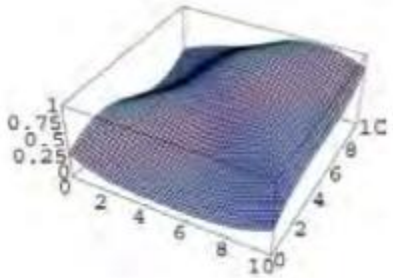
Classification is not limited to linear discriminants

2 Inputs, Single Output



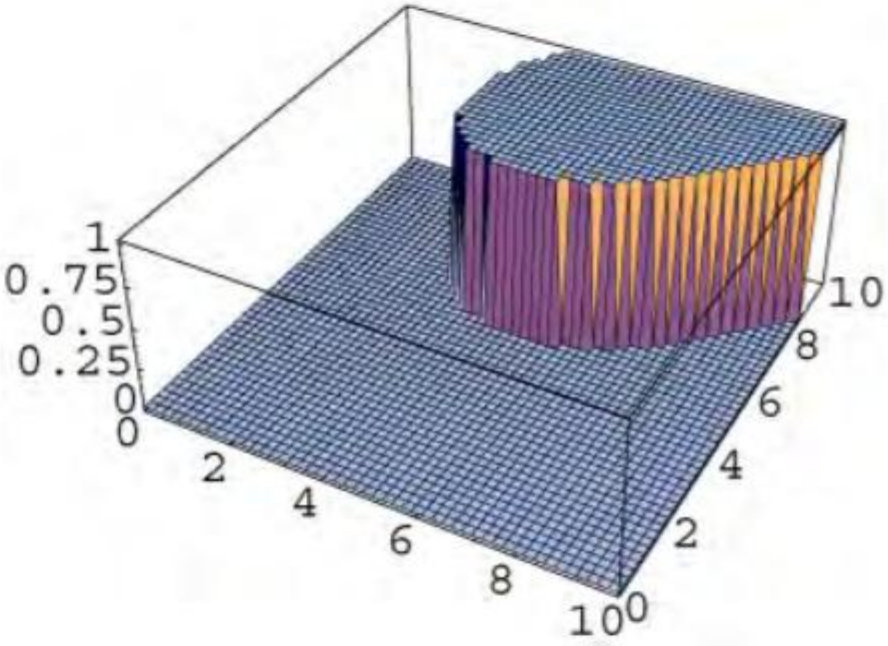


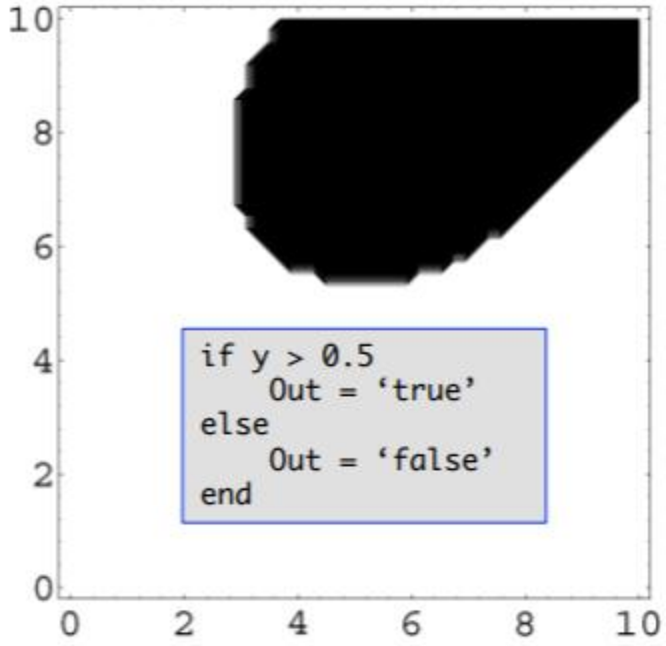
Sigmoid network can approximate a continuous nonlinear function to arbitrary accuracy with a *single hidden layer*



Thresholded Neural Network Output

Threshold gives "yes/no" output

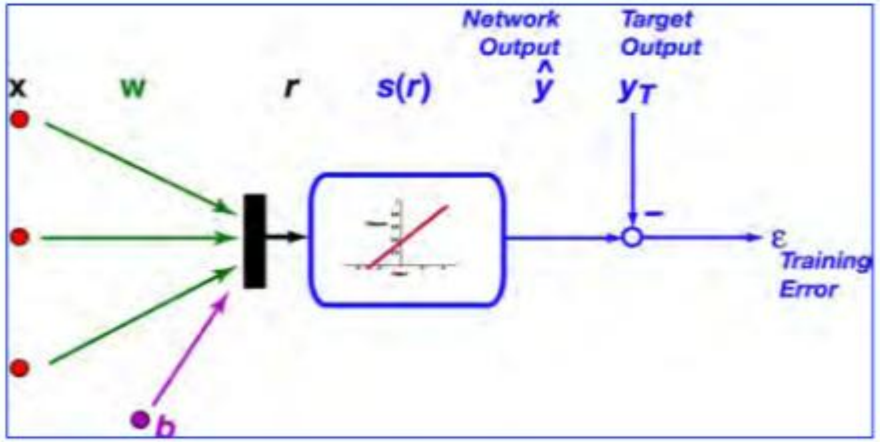




Least-Squares Training Example: Single Linear Neuron

- **Training set (*n* members)**
  - Target outputs, $\mathbf{y}_T$ (1 x *n*)
  - *m* Features (inputs), **X** (*m* x *n*)

$$\begin{bmatrix} \mathbf{y}_T \\ \mathbf{X} \end{bmatrix} = \begin{bmatrix} y_{T_1} & y_{T_2} & \cdots & y_{T_N} \\ \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}_1 & \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}_2 & \cdots & \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}_n \end{bmatrix}$$



- **Network output, single input**

$$\hat{y}_j = r_j = \hat{\mathbf{w}}^T \mathbf{x}_j + \hat{b}$$

- **Training error**

$$\varepsilon_j = \hat{y}_j - y_T$$

- **Quadratic error cost**

$$J = \frac{1}{2}\sum_{j=1}^{n} \varepsilon_j^{\,2} = \frac{1}{2}\sum_{j=1}^{n}\left(\hat{y}_j - y_T\right)^2 = \frac{1}{2}\sum_{j=1}^{n}\left(\hat{y}_j^{\,2} - 2\hat{y}_j\, y_T + y_T^{\,2}\right)$$

*Note: This is an introduction to least-squares **back-propagation training**. Training of a linear neuron more readily accomplished using left pseudoinverse (Lec. 21).*

Linear Neuron Gradient

$$\hat{y}_j = r_j = \mathbf{w}^T \mathbf{x}_j + b$$

$$\frac{d\hat{y}_j}{dr_j} = 1$$

$$\varepsilon_j = \hat{y}_j - y_T$$

$$J = \frac{1}{2}\sum_{j=1}^{n} \varepsilon_j^{\,2} = \frac{1}{2}\sum_{j=1}^{n} \left(\hat{y}_j - y_T\right)^2 = \frac{1}{2}\sum_{j=1}^{n} \left(\hat{y}_j^{\,2} - 2\hat{y}_j y_T + y_T^{\,2}\right)$$

- Training (control) parameter, **p**
  - Input weights, **w** (n x 1)
  - Bias, b (1 x 1)

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_{n+1} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

- Optimality condition $\dfrac{\partial J}{\partial \mathbf{p}} = \mathbf{0}$

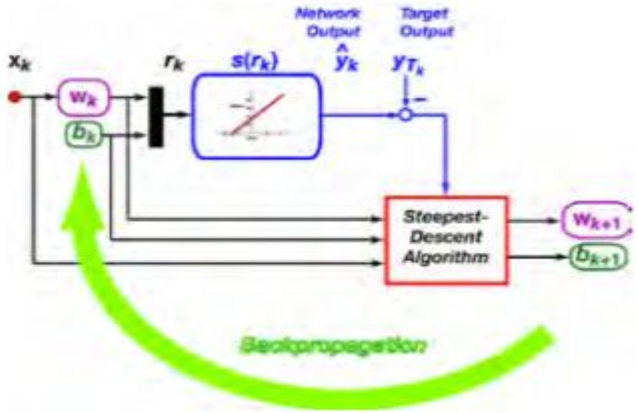
- Gradient

$$\frac{\partial J}{\partial \mathbf{p}} = \frac{1}{2}\sum_{j=1}^{n} \left(\hat{y}_j - y_T\right)\frac{\partial y_j}{\partial \mathbf{p}} = \frac{1}{2}\sum_{j=1}^{n} \left(\hat{y}_j - y_T\right)\frac{\partial y_j}{\partial r_j}\frac{\partial r_j}{\partial \mathbf{p}}$$

where

$$\frac{\partial r_j}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial r_j}{\partial p_1} & \frac{\partial r_j}{\partial p_2} & \dots & \frac{\partial r_j}{\partial p_{n+1}} \end{bmatrix} = \frac{\partial\left(\mathbf{w}^T \mathbf{x}_j + b\right)}{\partial \mathbf{p}} = \begin{bmatrix} \mathbf{x}_j^{\,T} & 1 \end{bmatrix}$$

Steepest-Descent (Back-propagation) Learning for a Single Linear Neuron



Gradient

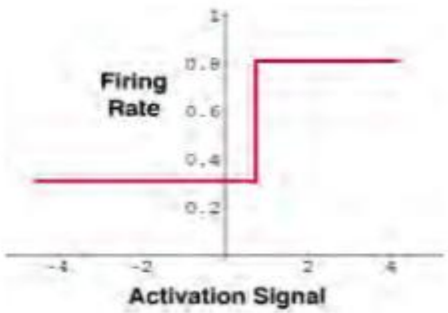
$$\frac{\partial J}{\partial \mathbf{p}} = \frac{1}{2}\sum_{j=1}^{n}\left(\hat{y}_j - y_T\right)\begin{bmatrix} \mathbf{x}_j^{\,T} & 1 \end{bmatrix} = \frac{1}{2}\sum_{j=1}^{n}\left[\left(\mathbf{w}^T\mathbf{x}_j + b\right) - y_T\right]\begin{bmatrix} \mathbf{x}_j^{\,T} & 1 \end{bmatrix}$$

Steepest-descent algorithm

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \eta\left(\frac{\partial J}{\partial \mathbf{p}}\right)_k^T$$

$\eta =$ learning rate

$k =$ iteration index(epoch)



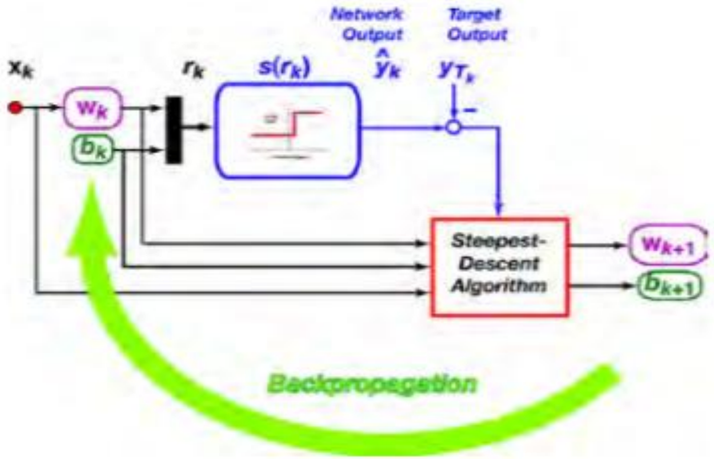
Steepest-Descent Algorithm for a Single-Step Perceptron

Neuron output is discontinuous

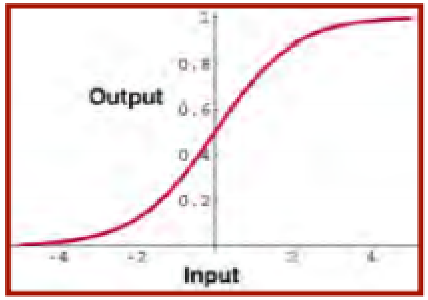
$$\hat{y} = s(r) = \begin{cases} 1, & r > 0 \\ 0, & r \le 0 \end{cases}$$

Binary target output
y_T = 0 or 1, for classification

$$\left(\hat{y}_{jk} - y_{T_k}\right) = \begin{cases} 1, & \hat{y}_{jk} = 1, \quad y_{T_k} = 0 \\ 0, & \hat{y}_{jk} = y_{T_k} \\ -1, & \hat{y}_{jk} = 0, \quad y_{T_k} = 1 \end{cases}$$



$$\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}_{k+1} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}_k - \eta \sum_{j=1}^{N} \left[\hat{y}_{jk} - y_{T_k}\right] \begin{bmatrix} \mathbf{x}_j \\ 1 \end{bmatrix}_k$$



Training Variables for a Single Sigmoid Neuron

Neuron output is continuous

$$\hat{y} = s(r) = \frac{1}{1+e^{-r}}$$
$$= s\left(\mathbf{w}^T\mathbf{x}+b\right) = \frac{1}{1+e^{-\left(\mathbf{w}^T\mathbf{x}+b\right)}}$$

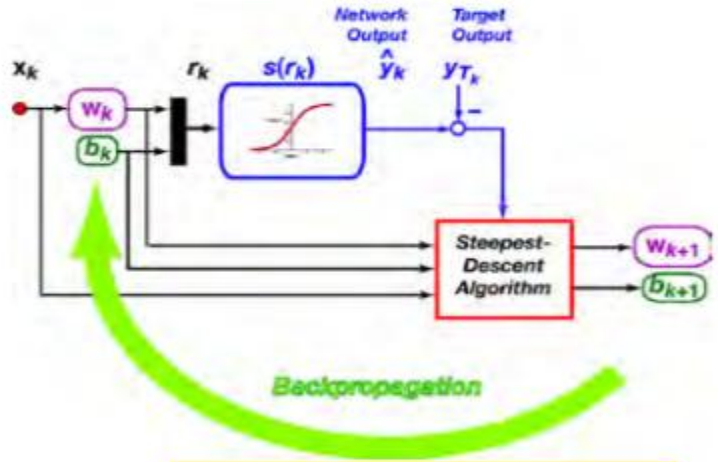
Training error and quadratic error cost

$$\varepsilon_j = \hat{y}_j - y_T$$
$$J = \frac{1}{2}\sum_{j=1}^{n}\varepsilon_j^{\,2} = \frac{1}{2}\sum_{j=1}^{n}\left(\hat{y}_j - y_T\right)^2 = \frac{1}{2}\sum_{j=1}^{n}\left(\hat{y}_j^{\,2} - 2\hat{y}_j y_T + y_T^{\,2}\right)$$

Neuron output sensitivity to input

$$\frac{d\hat{y}}{dr} = \frac{ds(r)}{dr} = \frac{e^{-r}}{\left(1+e^{-r}\right)^2} = e^{-r}s^2(r)$$
$$= \left[\left(1+e^{-r}\right)-1\right]s^2(r) = \left[\frac{1-s(r)}{s(r)}\right]s^2(r)$$

$$\frac{d\hat{y}}{dr} = \left[1-s(r)\right]s(r) = \left(1-\hat{y}\right)\hat{y}$$



Back-Propagation Training of a Single Sigmoid Neuron

$$\frac{\partial J}{\partial \mathbf{p}} = \frac{1}{2}\sum_{j=1}^{N}\left(\hat{y}_j - y_T\right)\frac{\partial \hat{y}_j}{\partial r}\frac{\partial r}{\partial \mathbf{p}}$$

where

$$r = \mathbf{w}^T\mathbf{x} + b$$

$$\frac{d\hat{y}}{dr} = (1-\hat{y})\hat{y}$$

$$\frac{\partial r}{\partial \mathbf{p}} = \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \eta\left(\frac{\partial J}{\partial \mathbf{p}}\right)_k^T$$

or

$$\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}_{k+1} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}_k - \eta\sum_{j=1}^{N}\left\{\left[\hat{y}_{jk} - y_{T_k}\right](1-\hat{y}_k)\hat{y}_k\begin{bmatrix} \mathbf{x}_j \\ 1 \end{bmatrix}\right\}_k$$

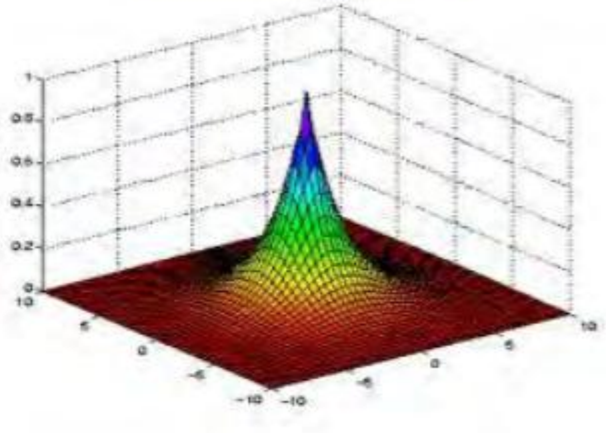
See Supplemental Material for training multiple sigmoids

Radial Basis Function

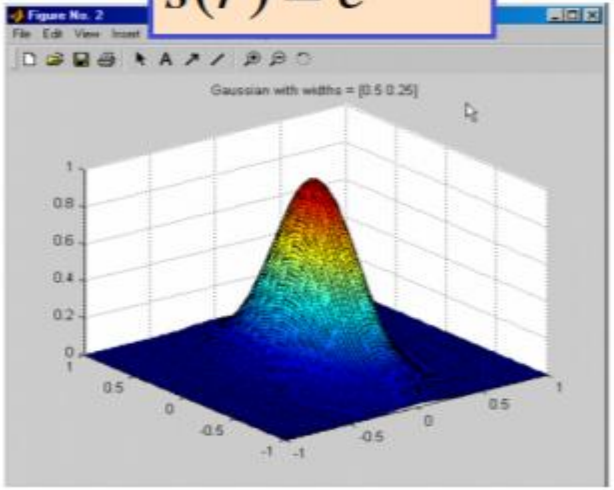
Unimodal, axially symmetric function, e.g., exponential

$$s(r) = e^{-|ar|^n}, \quad r = \sqrt{(\mathbf{x} - \mathbf{x}_{center})^T (\mathbf{x} - \mathbf{x}_{center})}$$

$$s(r) = e^{-|ar|}$$



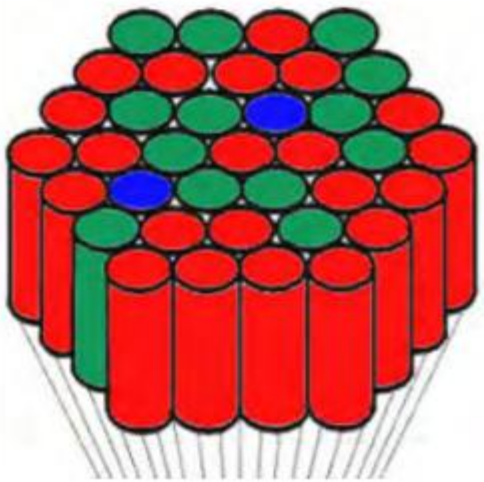
$$s(r) = e^{-(ar)^2}$$

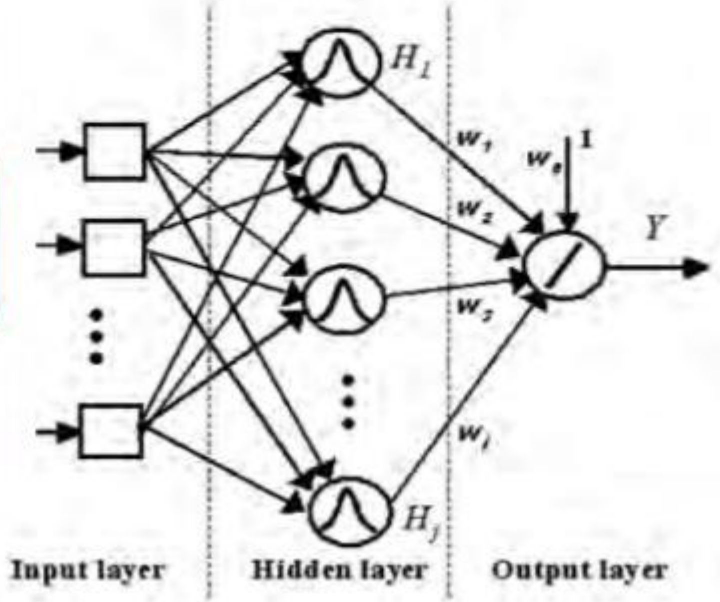


Network mimics stimulus field of a neuron receptor, e.g., retina

Radial Basis Function Network

Array of RBFs typically centered on a fixed grid





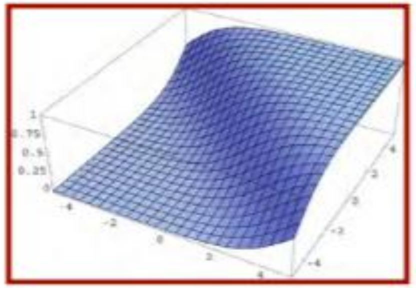
http://en.wikipedia.org/wiki/Radial_basis_function_network

Sigmoid vs. Radial Basis Function Node

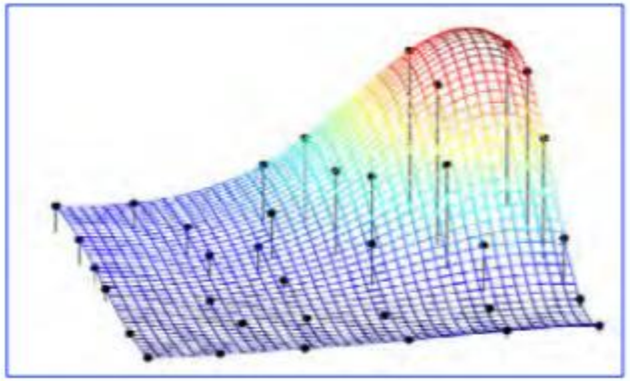
- **Considerations for selecting the basis function**
  - Prior knowledge of surface to be approximated
  - Global vs. compact support
  - Number of neurons required
  - Training and untraining issues

Sigmoid function

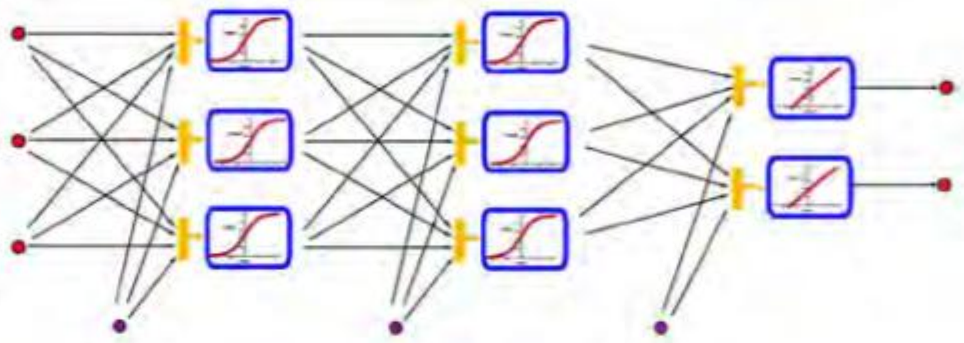
$$s(r) = \frac{1}{1+e^{-r}}$$



Radial basis functions



“Deep” Sigmoid Network



- **Multiple hidden and “visible” layers can improve accuracy in image processing and language translation**
- **Problem of the “vanishing gradient” in training**
- **One solution: *Convolutional neural network* of neuron input/output by incremental training**
  - **Pooling or clustering signals between layers *(TBD)***
  - **Limited receptive fields for filter (or kernel) nodes**
  - **Node is activated only when input is within pre-determined bounds**