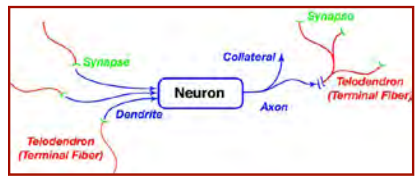
Introduction to Neural Networks

Robert Stengel

Robotics and Intelligent Systems, MAE 345,

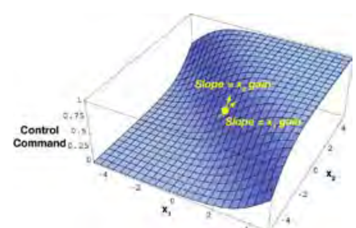
Princeton University, 2017

- 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



http://www.princeton.edu/~stengel/MAE345.html

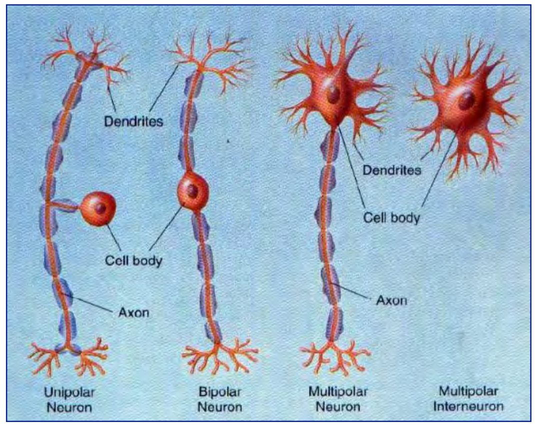
Applications of Computational Neural Networks



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

Neurons

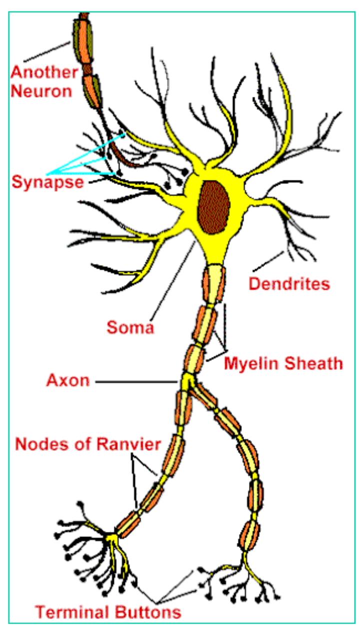
- Biological cells with significant electrochemical activity
- ~10-100 billion neurons in the brain
- Inputs from thousands of other neurons
- Output is scalar, but may have thousands of branches



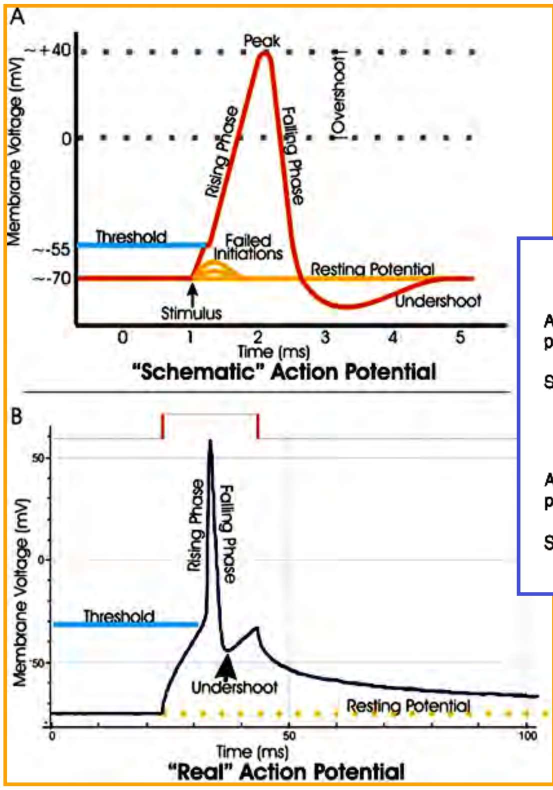
- Afferent (sensor) neurons send signals from organs and the periphery to the central nervous system
- Efferent (motor) neurons issue commands from the CNS to effector (e.g., muscle) cells
- Interneurons send signals between neurons in the central nervous system
- Signals are ionic, i.e., chemical (neurotransmitter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

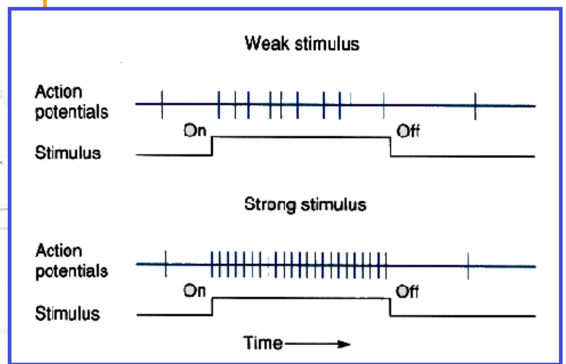
Activation Input to Soma Causes Change in Output Potential

- 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



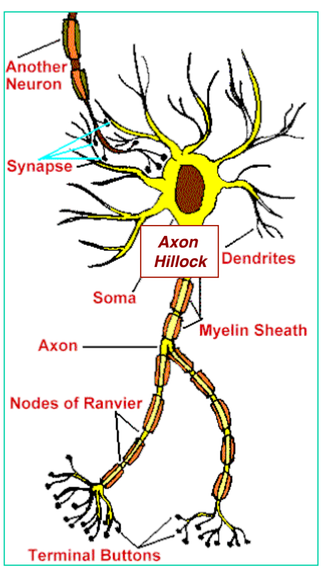
Neural Action Potential

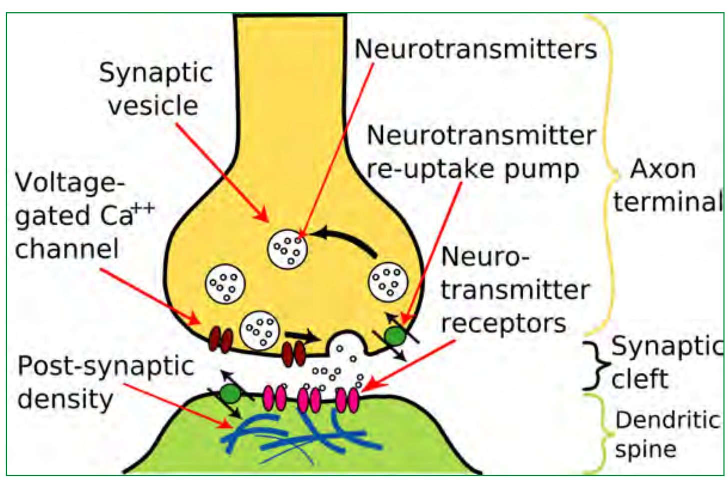




- **Maximum Firing Rate**: 500/sec
- **Refractory Period**: Minimum time increment between action potential firing ~ 1-2 msec

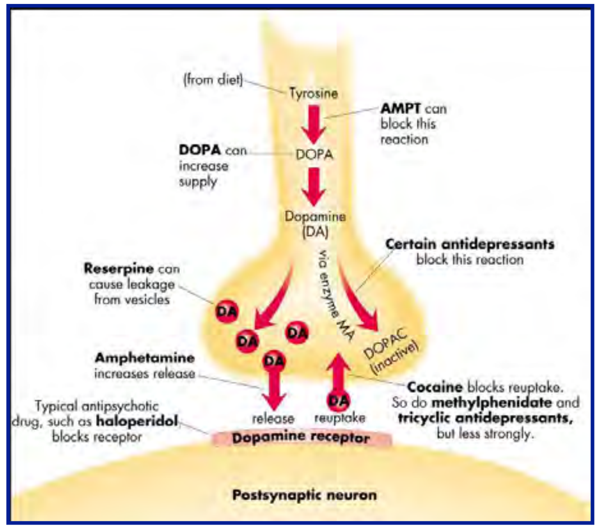
Electrochemical Signaling at Axon Hillock and Synapse



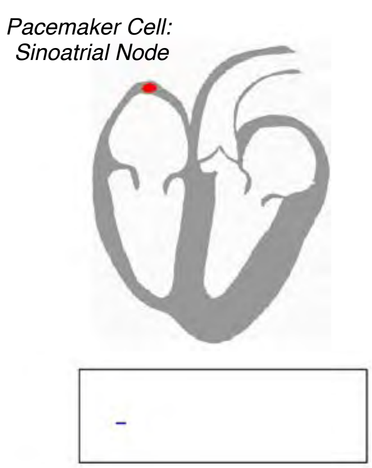


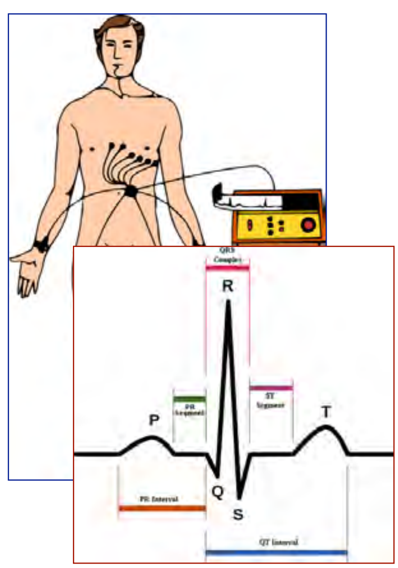
Synaptic Strength Can Be Increased or Decreased by Externalities

- 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

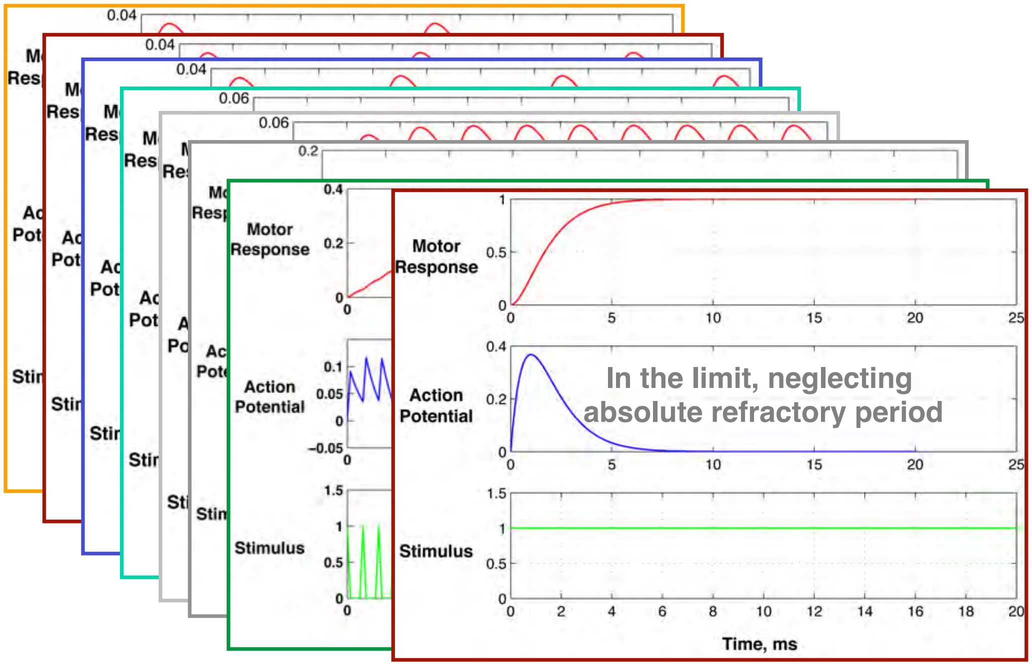


Cardiac Pacemaker and EKG Signals

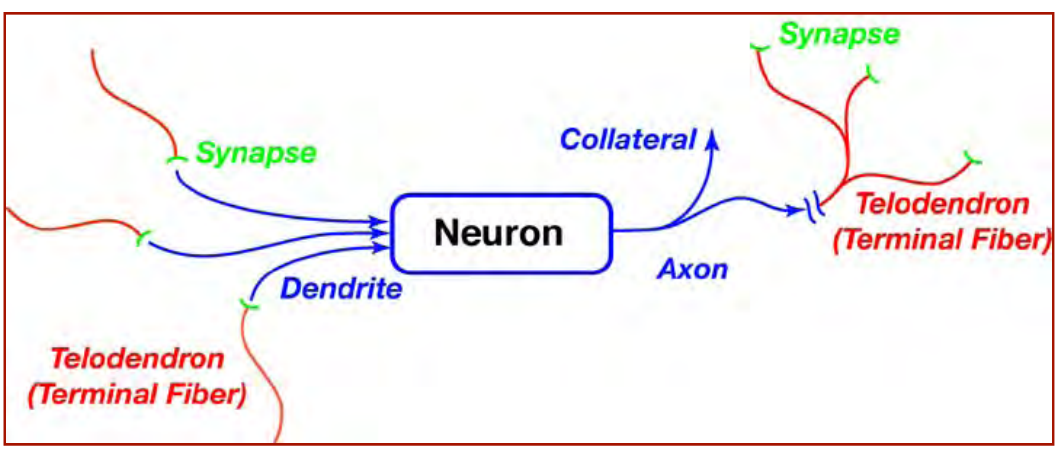




Impulse, Pulse-Train, and Step Response of LTI 2nd-Order Neural Model



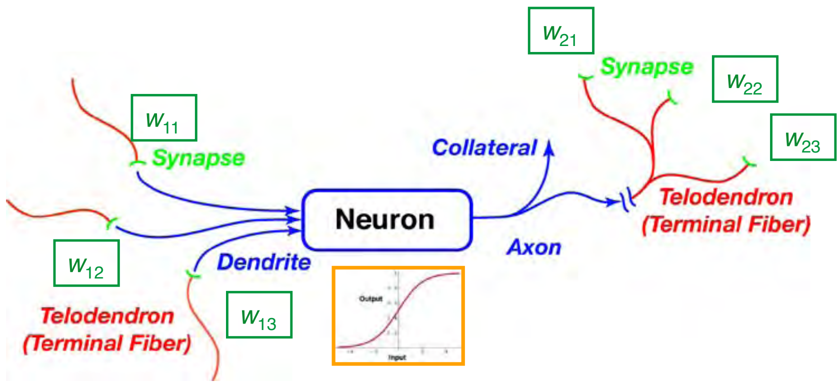
Multipolar Neuron



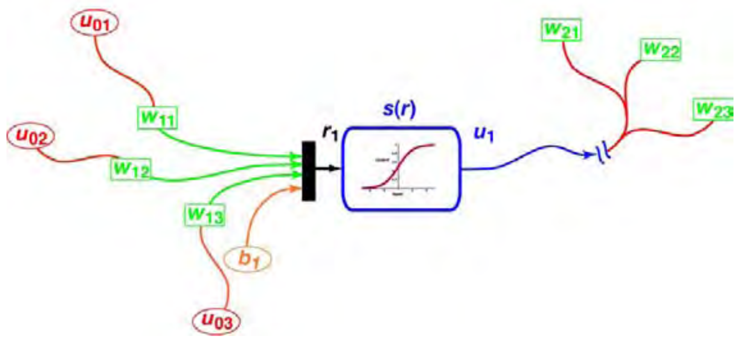
Mathematical Model of Neuron Components

Synapse effects represented by weights (gains or multipliers)

Neuron firing frequency is modeled by linear gain or nonlinear element



The Neuron Function



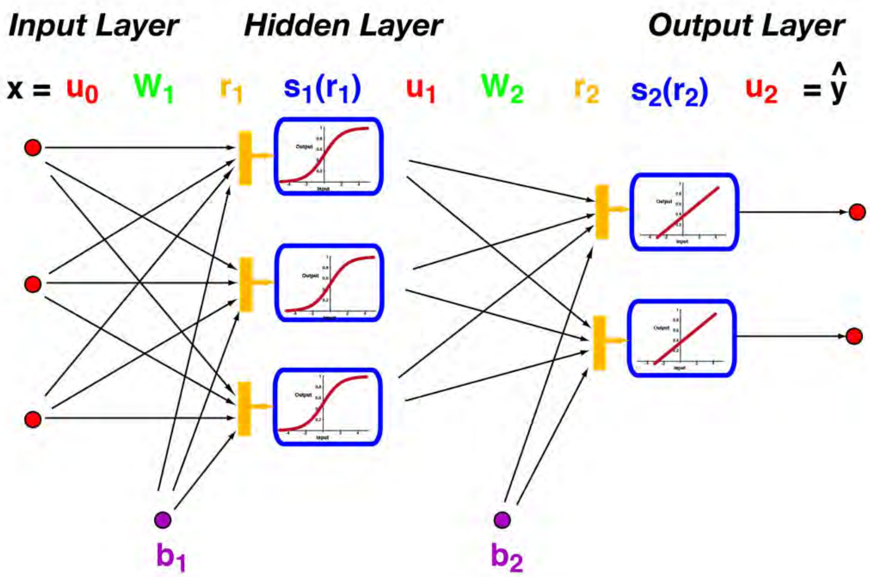
- **Vector input, u, to a single neuron**
  - **Sensory input or output from upstream neurons**
- **Linear operation produces scalar, *r***
- **Add bias, *b*, for zero adjustment**

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

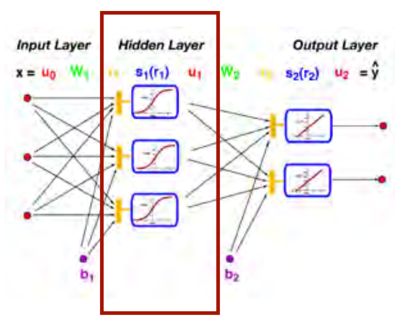
- **Scalar output, *u*, of a single neuron (or node)**
  - **Scalar linear or nonlinear operation, *s*(*r*)**

$$u = s(r)$$

"Shallow" Neural Network



Layered, parallel structure for computation



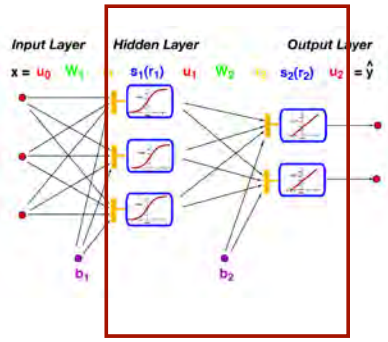
Input-Output Characteristics of a Neural Network Layer

- **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)

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

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

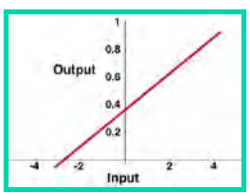
$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \cdots \\ \mathbf{w}_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$



Two-Layer Network

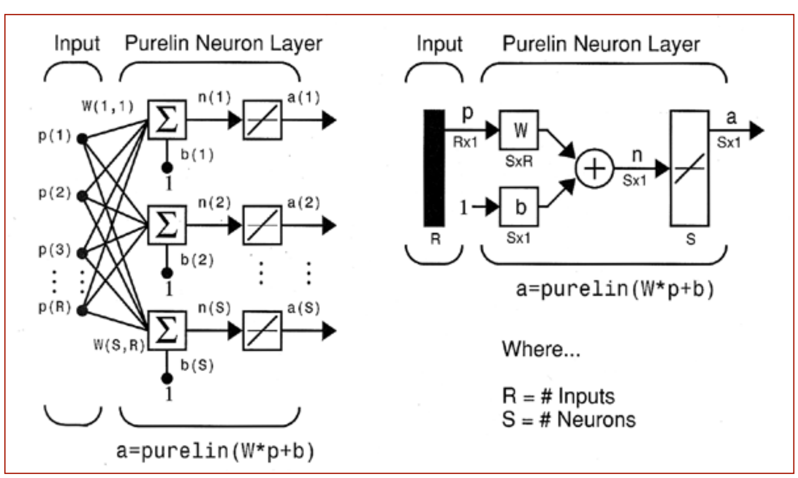
- Two layers
  - Node functions may be different, e.g.,
    - Sigmoid hidden layer
    - Linear output layer
  - Number of nodes in each layer need not be the same
- Input sometimes labeled as layer

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



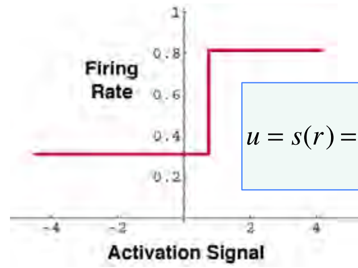
Linear Neural Network

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



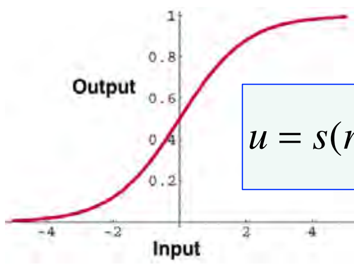
Idealizations of Nonlinear Neuron Input-Output Characteristic

Step function ("Perceptron")



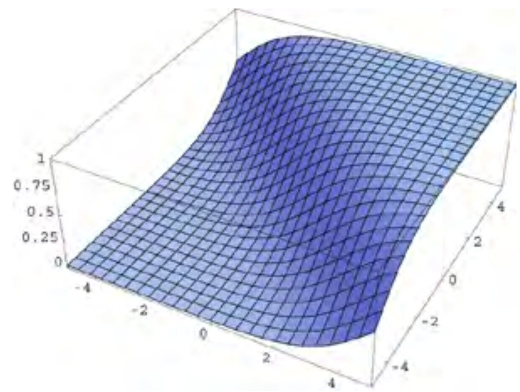
$$u = s(r) = \begin{cases} 1, & r > 0 \\ 0, & r \le 0 \end{cases}$$

Logistic sigmoid function



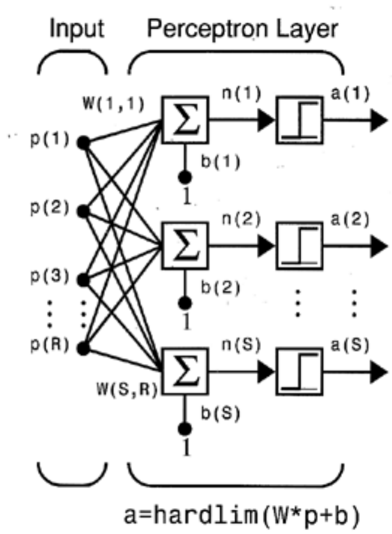
$$u = s(r) = \frac{1}{1 + e^{-r}}$$

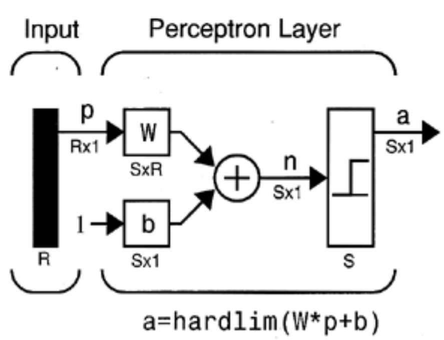
Sigmoid with two inputs, one output



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

Perceptron Neural Network





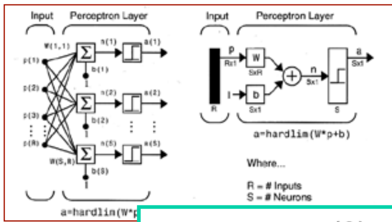
Where...

R = # Inputs
S = # Neurons

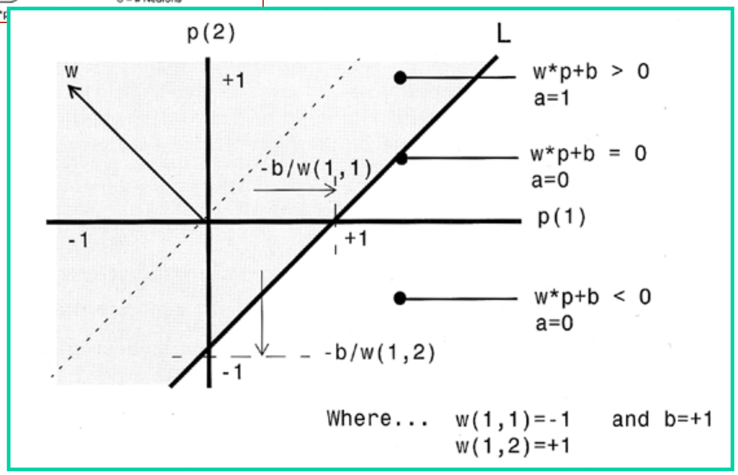
Each node is a step function

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

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

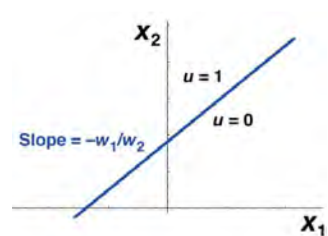
Two inputs, single step function

Discriminant

$$0 = w_1 x_1 + w_2 x_2 + b$$

$$x_2 = \frac{-1}{w_2}(w_1 x_1 + b)$$

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



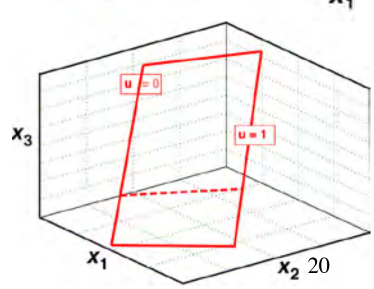
Three inputs, single step function

Discriminant

$$0 = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$x_3 = \frac{-1}{w_3}(w_1 x_1 + w_2 x_2 + b)$$

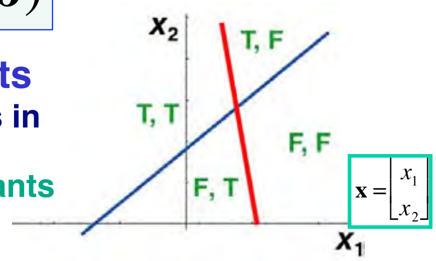
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

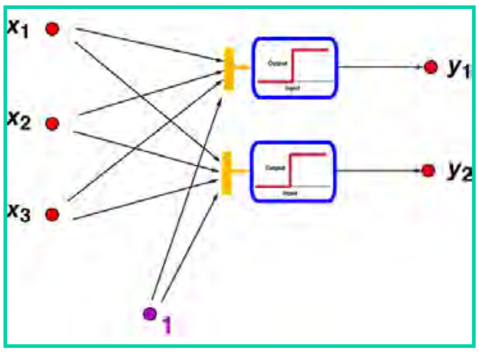


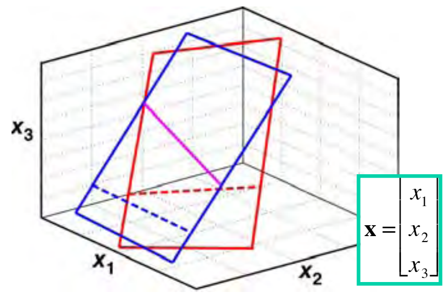
Single-Layer, Multi-Node Perceptron Discriminants

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

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





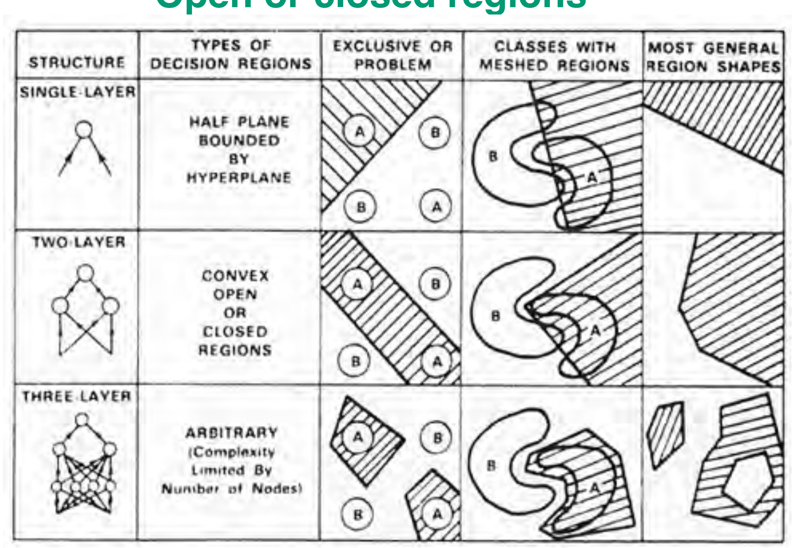


Multi-Layer Perceptrons Can Classify With Boundaries or Clusters

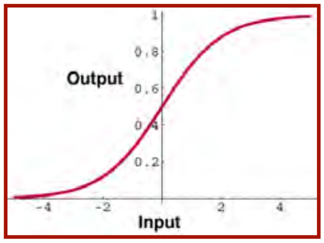
Classification capability of multi-layer perceptrons

Classifications of classifications

Open or closed regions



STRUCTURE	TYPES OF DECISION REGIONS	EXCLUSIVE OR PROBLEM	CLASSES WITH MESHED REGIONS	MOST GENERAL REGION SHAPES
SINGLE-LAYER	HALF PLANE BOUNDED BY HYPERPLANE	A B B A	B A	
TWO-LAYER	CONVEX OPEN OR CLOSED REGIONS	A B B A	B A	
THREE-LAYER	ARBITRARY (Complexity Limited By Number of Nodes)	A B B A	B A	



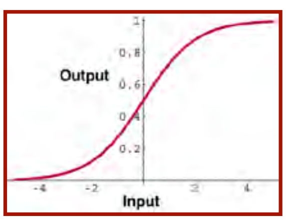
Sigmoid Activation Functions

- **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**

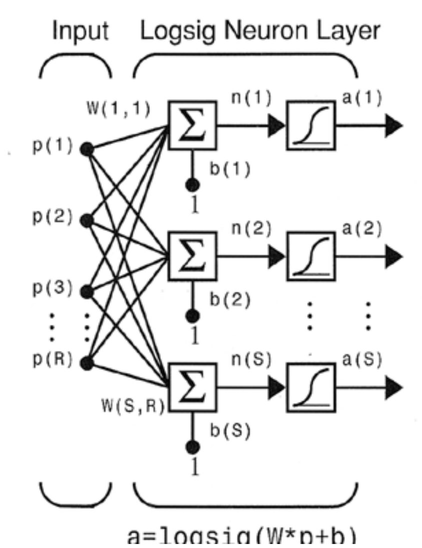
$$u = s(r) = \frac{1}{1+e^{-r}}$$

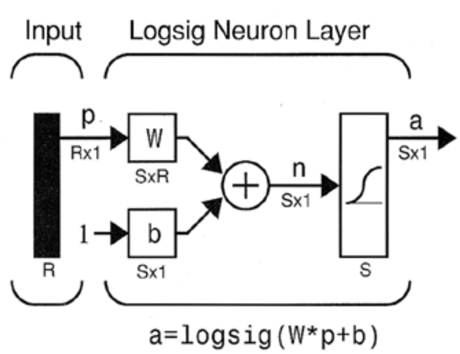
$$u = s(r) = \tanh r = \frac{1-e^{-2r}}{1+e^{-2r}}$$

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



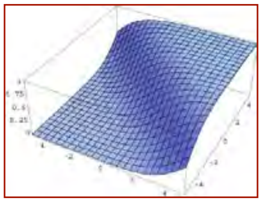
Single-Layer Sigmoid Neural Network





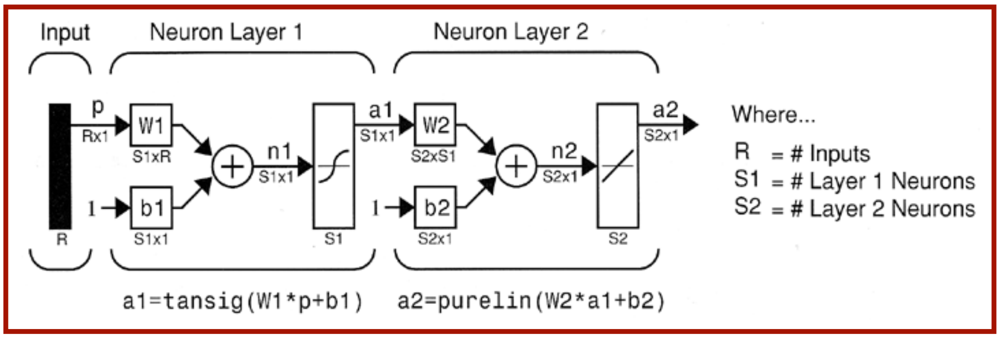
Where...

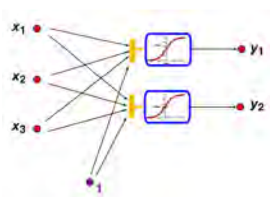
R = # Inputs
S = # Neurons



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)**

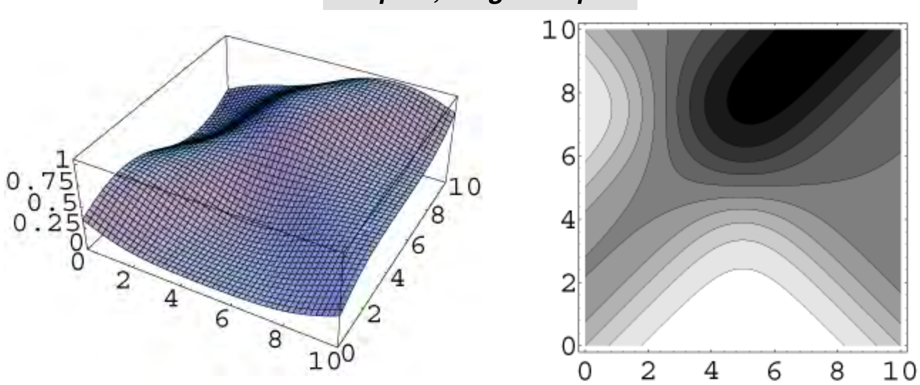




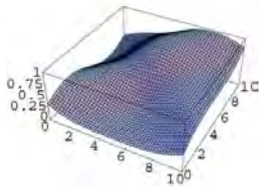
Typical Output for Two-Sigmoid Network

Classification is not limited to linear discriminants

2 Inputs, Single Output

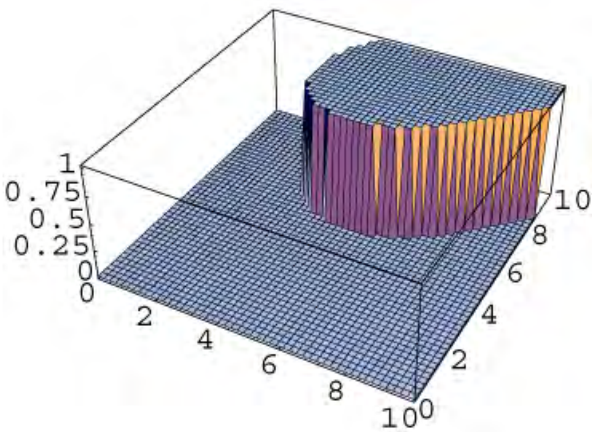


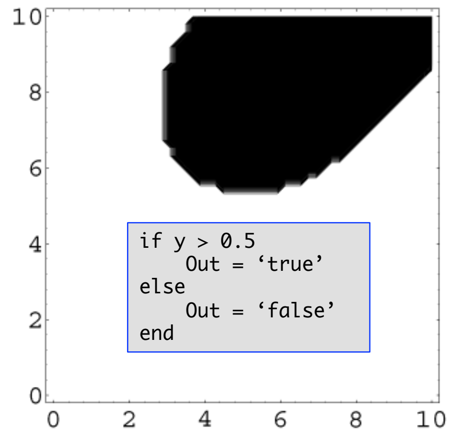
Sigmoid network can approximate a continuous nonlinear function to arbitrary accuracy with a *single hidden layer*



Thresholded Neural Network Output

Threshold gives “yes/no” output

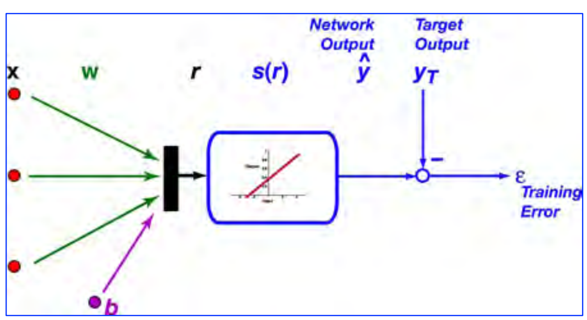




Least-Squares Training Example: Single Linear Neuron

- **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**

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

- **Training error**

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

- **Quadratic error cost**

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

*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)

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

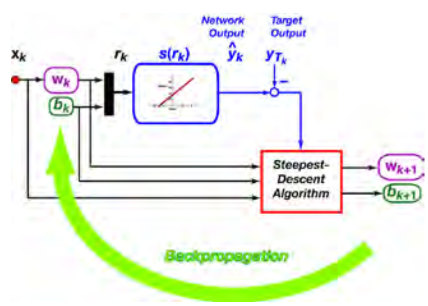
- **Optimality condition** $$\frac{\partial J}{\partial \mathbf{p}} = \mathbf{0}$$

- **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

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

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



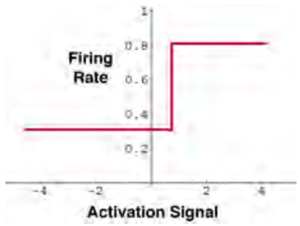
Gradient

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

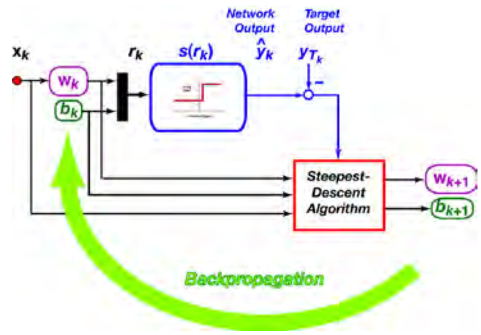
Steepest-descent algorithm

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

$\eta =$ learning rate
$k =$ iteration index(epoch)



Steepest-Descent Algorithm for a Single-Step Perceptron



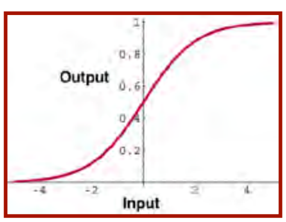
Neuron output is discontinuous

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

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

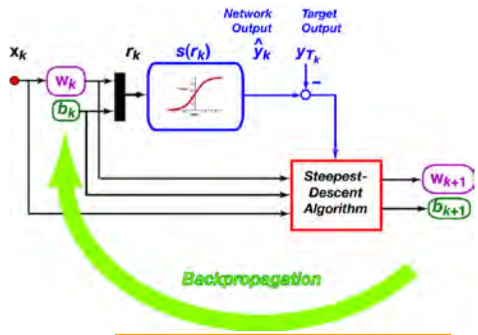
$$\varepsilon_j = \hat{y}_j - y_T$$

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

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

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

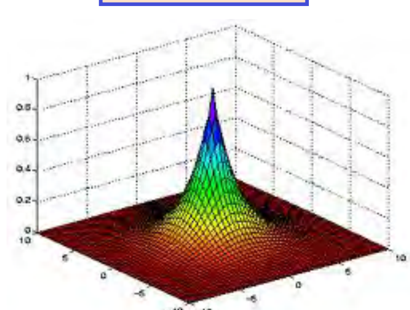
See Supplemental Material for training multiple sigmoids

Radial Basis Function

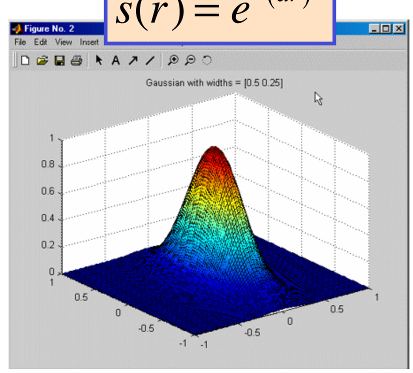
Unimodal, axially symmetric function, e.g., exponential

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

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



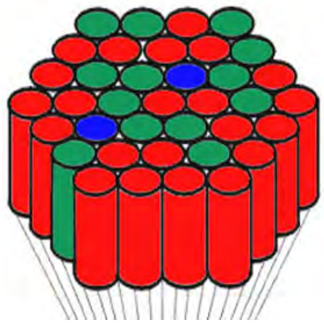
$$s(r) = e^{-(ar)^2}$$

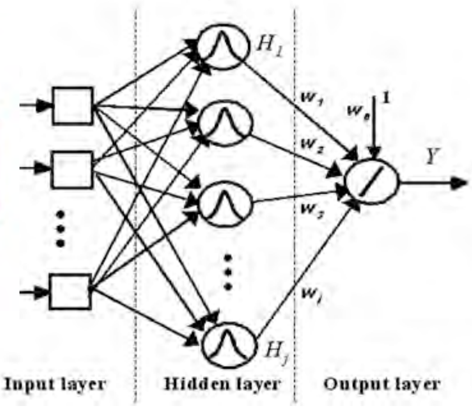


Network mimics stimulus field of a neuron receptor, e.g., retina

Radial Basis Function Network

Array of RBFs typically centered on a fixed grid





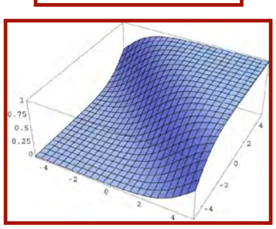
http://en.wikipedia.org/wiki/Radial_basis_function_network

Sigmoid vs. Radial Basis Function Node

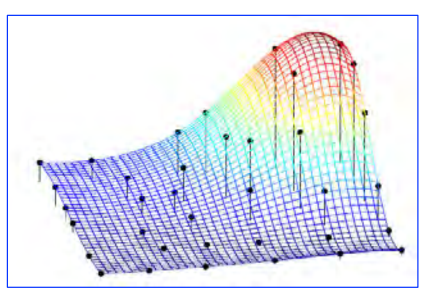
- **Considerations for selecting the basis function**
  - **Prior knowledge of surface to be approximated**
  - **Global vs. compact support**
  - **Number of neurons required**
  - **Training and untraining issues**

Sigmoid function

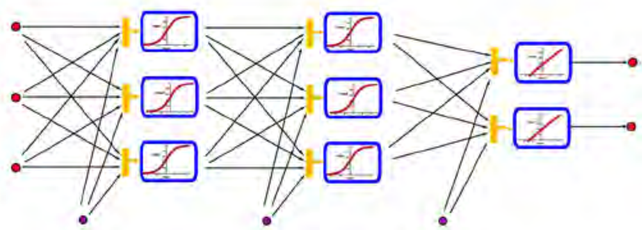
$$s(r) = \frac{1}{1+e^{-r}}$$



Radial basis functions



"Deep" Sigmoid Network



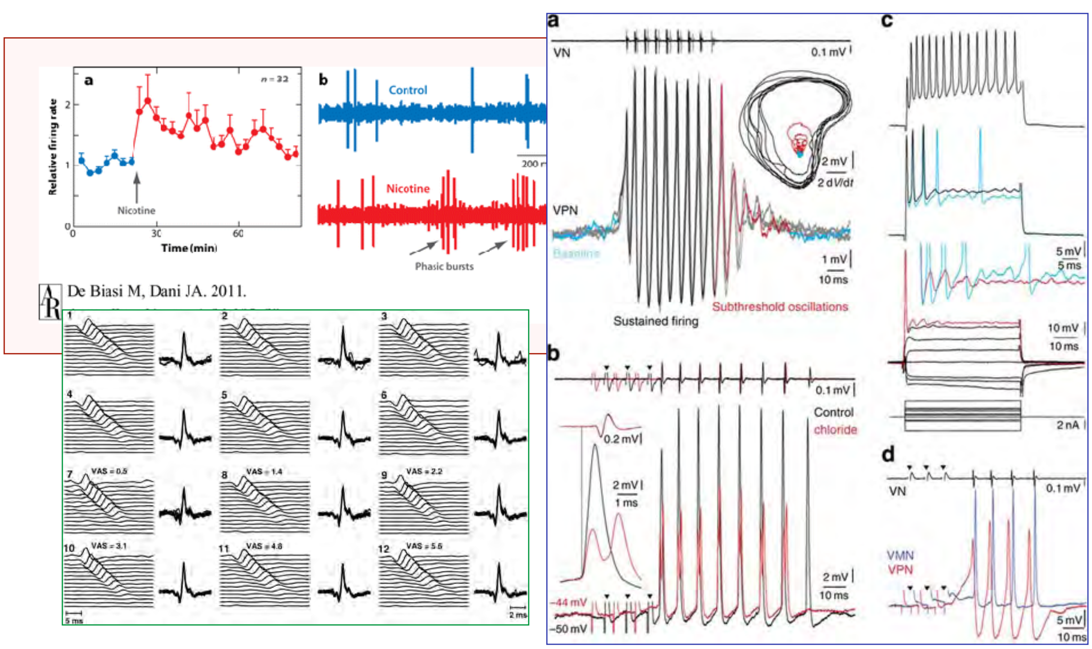
- **Multiple hidden and "visible" layers can improve accuracy in image processing and language translation**
- **Problem of the "vanishing gradient" in training**
- **<u>One solution</u>: *Convolutional neural network* of neuron input/output by incremental training**
  - **Pooling or clustering signals between layers *(TBD)***
  - **Limited receptive fields for filter (or kernel) nodes**
  - **Node is activated only when input is within pre-determined bounds (see *CMAC*, Lecture 19)**

Next Time: More on Neural Networks

Supplementary Material

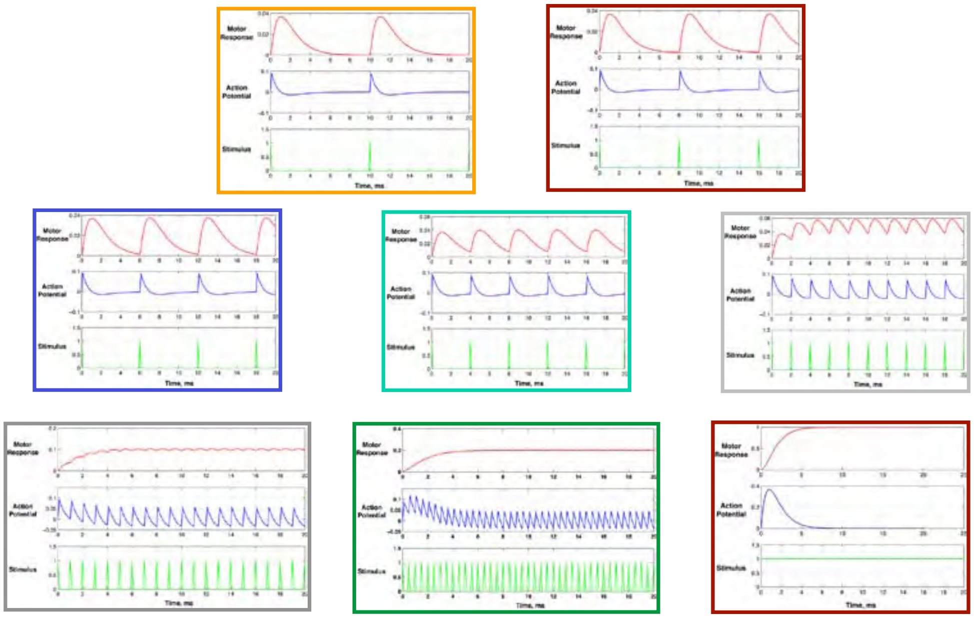
39

Some Recorded Action Potential Pulse Trains

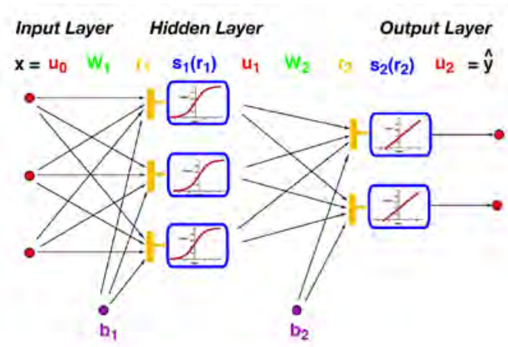


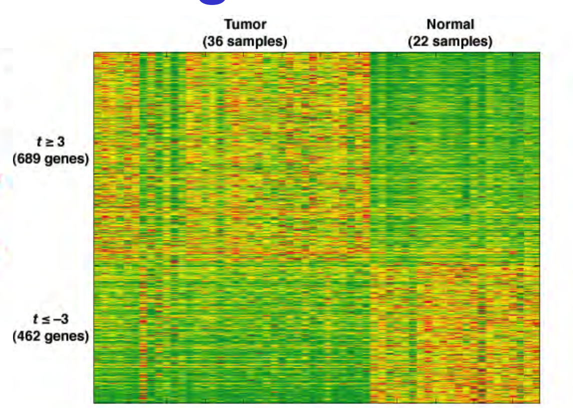
40

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



Microarray Training Set



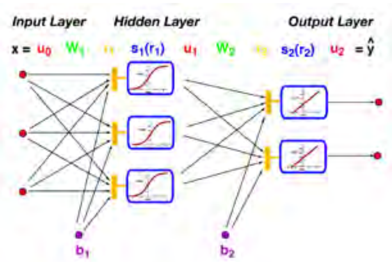


$$\begin{bmatrix} Identifier \\ \mathbf{y}_T \\ \mathbf{X} \end{bmatrix} = \begin{bmatrix} Sample\ 1 & Sample\ 2 & Sample\ 3 & \ldots & Sample\ n-1 & Sample\ n \\ Tumor & Tumor & Tumor & \ldots & Normal & Normal \\ Gene\ A\ Level & Gene\ A\ Level & Gene\ A\ Level & \ldots & Gene\ A\ Level & Gene\ A\ Level \\ Gene\ B\ Level & Gene\ B\ Level & Gene\ B\ Level & \ldots & Gene\ B\ Level & Gene\ B\ Level \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ Gene\ m\ Level & Gene\ m\ Level & Gene\ m\ Level & \ldots & Gene\ m\ Level & Gene\ m\ Level \end{bmatrix}$$

Microarray Training Data

- **First row: Target classification**
- **2nd-5th rows: Up-regulated genes**
- **6th-10th rows: Down-regulated genes**

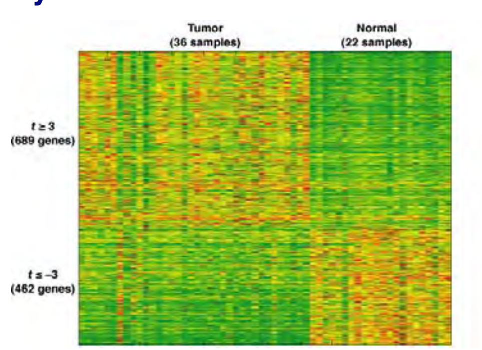
```
Lab Analysis of Tissue Samples
    Tumor   =[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
              1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
              0 0 0 0 0 0 0 0];

Normalized Data: Up-Regulated in Tumor
    U22055  =   [138  68   93   62   30   81   121  7    82   24   -2   -48  38   ...
                 82   118  55   103  102  87   62   69   14   101  25   47   48   75   ...
                 59   62   116  54   96   90   130  70   75   74   35   149  97   21   ...
                 14   -51  -3   -81  57   -4   16   28   -73  -4   45   -28  -9   -13  ...
                 25   25   19   -21  3    19   34];
.........
Normalized Data: Up-Regulated in Normal
    M96839  =   [3    -23  3    12   -22  0    4    29   -73  32   5    -13  -16  14   ...
                 2    24   18   19   9    -13  -20  -3   -22  6    -5   -12  9    28   ...
                 20   -9   30   -15  18   1    -16  12   -9   3    -35  23   3    5    ...
                 33   29   47   19   32   34   20   55   49   20   10   36   70   50   ...
                 15   45   56   41   31   40];
```

Neural Network Classification Example

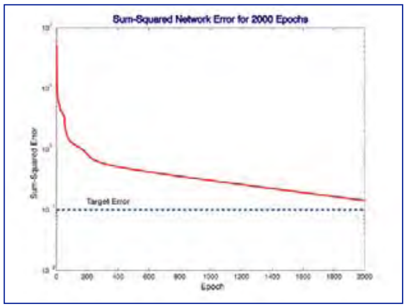
- **~7000 genes expressed in 62 microarray samples**
  - **Tumor = 1**
  - **Normal = 0**

- **8 genes in strong feature set**
  - **4 with Mean Tumor/Normal > 20:1**
  - **4 with Mean Normal/Tumor > 20:1**
  - **and minimum variance within up-regulated set**



Dukes Stages: A -> B -> C -> D

Neural Network Training Results:
Tumor/Normal Classification, 8 Genes, 4 Nodes



- **Training begins with a random set of weights**
- **Adjustable parameters**
  - **Learning rate**
  - **Target error**
  - **Maximum # of epochs**
- **Non-unique sets of trained weights**

Binary network output (0,1) after rounding

Zero classification errors

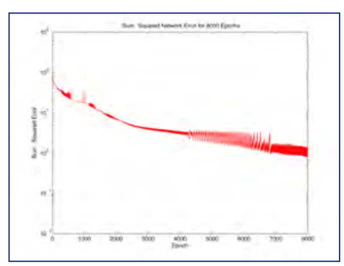
Classification =

Columns 1 through 13
1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 14 through 26
1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 27 through 39
1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 40 through 52
1 0 0 0 0 0 0 0 0 0 0 0 0
Columns 53 through 62
0 0 0 0 0 0 0 0 0 0

Neural Network Training Results:
Tumor Stage/Normal Classification
8 Genes, 16 Nodes

- **Colon cancer classification**
  - **0 = Normal**
  - **1 = Adenoma**
  - **2 = A Tumor**
  - **3 = B Tumor**
  - **4 = C Tumor**
  - **5 = D Tumor**

Scalar network output with varying magnitude

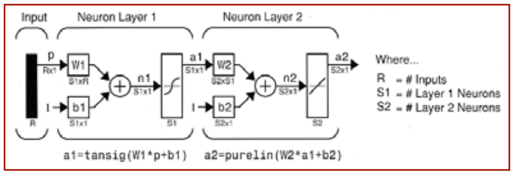


Target =
[2 1 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4 5 5 5
5 5 5 5 5 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0]

One classification error

Classification =

Columns 1 through 13
2 1 3 3 3 3 3 3 3 3 3 3 3
Columns 14 through 26
3 3 3 3 3 3 3 4 4 $\boxed{5}$ 4 4 4
Columns 27 through 39
4 4 4 5 5 5 5 5 5 5 5 1 0
Columns 40 through 52
0 0 0 0 0 0 0 0 0 0 0 0 0
Columns 53 through 60
0 0 0 0 0 0 0 0



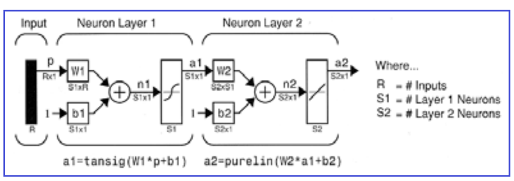
Training a Sigmoid Network

Two parameter vectors for 2-layer network

$$\mathbf{p}_{1,2} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}_{1,2} = \begin{bmatrix} p_1 \\ p_2 \\ \ldots \\ p_{n+1} \end{bmatrix}_{1,2}$$

Output vector

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



Training a Sigmoid Network

$$\mathbf{p}_{1,2k} = \mathbf{p}_{1,2k} - \eta\left(\frac{\partial J}{\partial \mathbf{p}_{1,2}}\right)_k^T$$

where

$$\frac{\partial J}{\partial \mathbf{p}_{1,2}} = (\hat{\mathbf{y}} - \mathbf{y}_T)\frac{\partial \mathbf{y}}{\partial \mathbf{p}_{1,2}} = (\hat{\mathbf{y}} - \mathbf{y}_T)\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{r}_{1,2}}\frac{\partial \mathbf{r}_{1,2}}{\partial \mathbf{p}_{1,2}}$$

$$\mathbf{r}_{1,2} = \mathbf{W}_{1,2}\mathbf{u}_{0,1} + \mathbf{b}_{1,2}$$

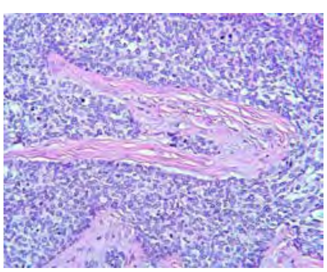
$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{r}_2} = \mathbf{I}; \quad \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{r}_1} = \begin{bmatrix} (1-\hat{y}_1)\hat{y}_1 & 0 & \ldots & 0 \\ 0 & (1-\hat{y}_2)\hat{y}_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & 0 \\ 0 & 0 & \ldots & (1-\hat{y}_n)\hat{y}_n \end{bmatrix}$$

$$\frac{\partial \mathbf{r}_1}{\partial \mathbf{p}_1} = \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix}; \quad \frac{\partial \mathbf{r}_2}{\partial \mathbf{p}_2} = \begin{bmatrix} \mathbf{u}_1^T & 1 \end{bmatrix}$$

MATLAB Neural Network Toolbox Training Algorithms

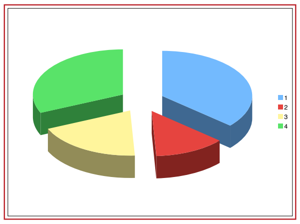
```
Backpropagation training functions that use Jacobian derivatives

  These algorithms can be faster but require more memory than gradient
  backpropation.  They are also not supported on GPU hardware.

  trainlm    - Levenberg-Marquardt backpropagation.
  trainbr    - Bayesian Regulation backpropagation.

Backpropagation training functions that use gradient derivatives

  These algorithms may not be as fast as Jacobian backpropagation.
  They are supported on GPU hardware with the Parallel Computing Toolbox.

  trainbfg   - BFGS quasi-Newton backpropagation.
  traincgb   - Conjugate gradient backpropagation with Powell-Beale restarts.
  traincgf   - Conjugate gradient backpropagation with Fletcher-Reeves updates.
  traincgp   - Conjugate gradient backpropagation with Polak-Ribiere updates.
  traingd    - Gradient descent backpropagation.
  traingda   - Gradient descent with adaptive lr backpropagation.
  traingdm   - Gradient descent with momentum.
  traingdx   - Gradient descent w/momentum & adaptive lr backpropagation.
  trainoss   - One step secant backpropagation.
  trainrp    - RPROP backpropagation.
  trainscg   - Scaled conjugate gradient backpropagation.

Supervised weight/bias training functions

  trainb     - Batch training with weight & bias learning rules.
  trainc     - Cyclical order weight/bias training.
  trainr     - Random order weight/bias training.
  trains     - Sequential order weight/bias training.

Unsupervised weight/bias training functions

  trainbu    - Unsupervised batch training with weight & bias learning rules.
  trainru    - Unsupervised random order weight/bias training.
```


Small, Round Blue-Cell Tumor Classification Example



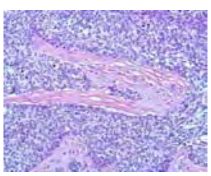
Desmoplastic small, round blue-cell tumors

- Childhood cancers, including
  - Ewing's sarcoma (EWS)
  - Burkitt's Lymphoma (BL)
  - Neuroblastoma (NB)
  - Rhabdomyosarcoma (RMS)
- cDNA microarray analysis presented by **J. Khan, *et al.*, *Nature Medicine*, 2001, 673-679.**
  - 96 transcripts chosen from 2,308 probes for training
  - 63 EWS, BL, NB, and RMS training samples
- **Source of data for my analysis**



Overview of Present SRBCT Analysis

- **Transcript selection by *t* test**
  - **96 transcripts, 12 highest and lowest *t* values for each class**
  - **Overlap with Khan set: 32 transcripts**
- **Ensemble averaging of genes with highest and lowest *t* values in each class**
- **Cross-plot of ensemble averages**
- **Classification by sigmoidal neural network**
- **Validation of neural network**
  - **Novel set simulation**
  - **Leave-one-out simulation**



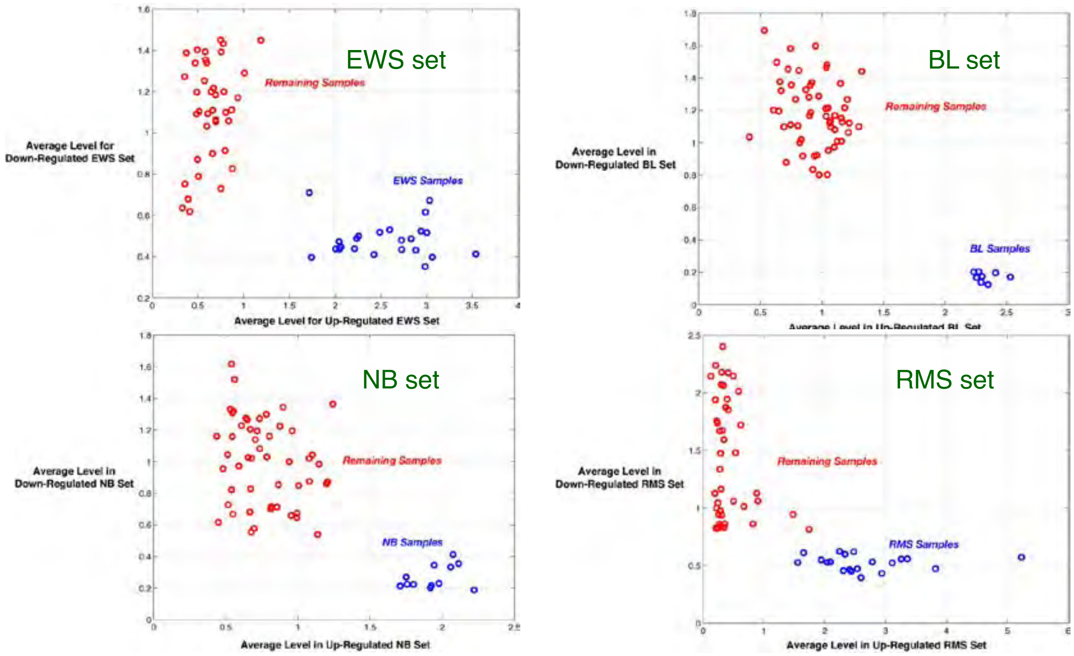
Ranking by EWS *t* Values (Top and Bottom 12)

- **24 transcripts selected from 12 highest and lowest *t* values for EWS vs. remainder**

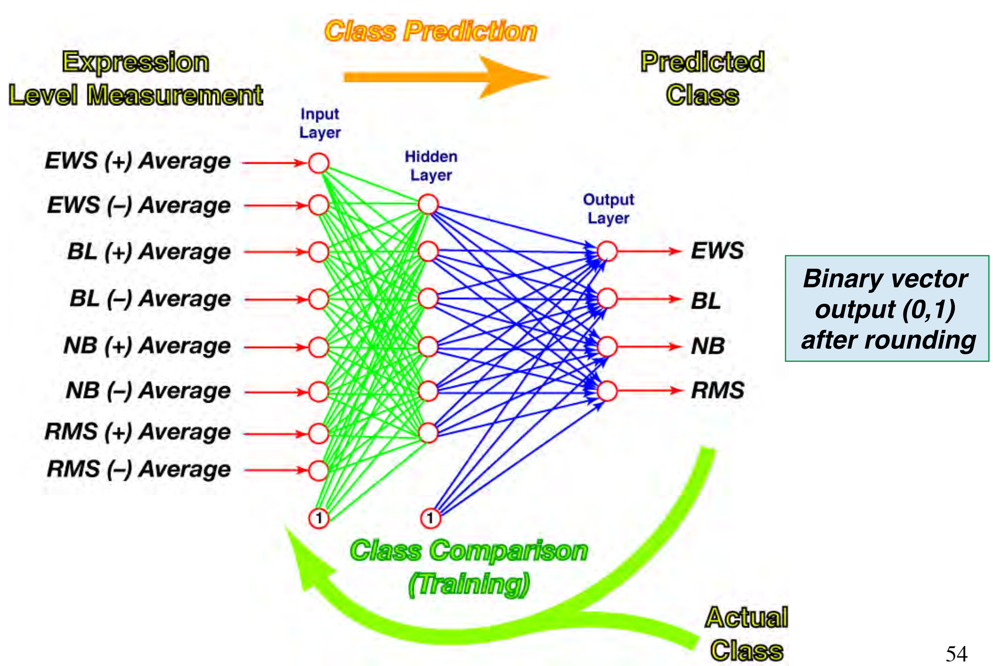
Image ID	Sort by EWS t Value Transcript Description	EWS t Value	BL t Value	NB t Value	RMS t Value
770394	Fc fragment of IgG, receptor, transporter, alpha	12.04	-6.67	-6.17	-4.79
1435862	antigen identified by monoclonal antibodies 12E7, F21 and O13	9.09	-6.75	-5.01	-4.03
377461	caveolin 1, caveolae protein, 22kD	8.82	-5.97	-4.91	-4.78
814260	follicular lymphoma variant translocation 1	8.17	-4.31	-4.70	-5.48
491565	Cbp/p300-interacting transactivator, with Glu/Asp-rich carboxy-terminal domain	7.60	-5.82	-2.62	-3.68
841641	cyclin D1 (PRAD1: parathyroid adenomatosis 1)	6.84	-9.93	0.56	-4.30
1471841	ATPase, Na+/K+ transporting, alpha 1 polypeptide	6.65	-3.56	-2.72	-4.69
866702	protein tyrosine phosphatase, non-receptor type 13	6.54	-4.99	-4.07	-4.84
713922	glutathione S-transferase M1	6.17	-5.61	-5.16	-1.97
308497	KIAA0467 protein	5.99	-6.69	-6.63	-1.11
770868	NGFI-A binding protein 2 (ERG1 binding protein 2)	5.93	-6.74	-3.88	-1.21
345232	lymphotoxin alpha (TNF superfamily, member 1)	5.61	-8.05	-2.49	-1.19
786084	chromobox homolog 1 (Drosophila HP1 beta)	-5.04	-1.05	9.65	-0.62
796258	sarcoglycan, alpha (50kD dystrophin-associated glycoprotein)	-5.04	-3.31	-3.86	6.83
431397		-5.04	2.64	2.19	0.64
825411	N-acetylglucosamine receptor 1 (thyroid)	-5.06	-1.45	5.79	0.76
859359	quinone oxidoreductase homolog	-5.23	-7.27	0.78	5.40
75254	cysteine and glycine-rich protein 2 (LIM domain only, smooth muscle)	-5.30	-4.11	2.20	3.68
448386		-5.38	-0.42	3.76	0.14
68950	cyclin E1	-5.80	0.03	-1.58	5.10
774502	protein tyrosine phosphatase, non-receptor type 12	-5.80	-5.56	3.76	3.66
842820	inducible poly(A)-binding protein	-6.14	0.60	0.66	3.80
214572	ESTs	-6.39	-0.08	-0.22	4.56
295985	ESTs	-9.26	-0.13	3.24	2.95

Repeated for BL vs. remainder, NB vs. remainder, and RMS vs. remainder

Clustering of SRBCT Ensemble Averages



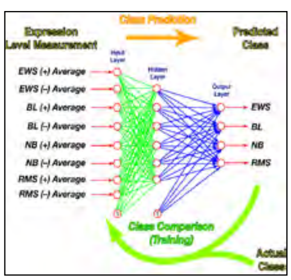
SRBCT Neural Network



Neural Network Training Set

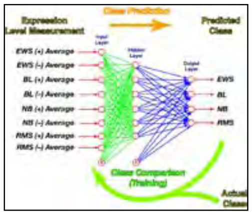
Each input row is an ensemble average for a transcript set, normalized in (–1,+1)

Identifier	*Sample* 1	*Sample* 2	*Sample* 3	...	*Sample* 62	*Sample* 63
Target Output	*EWS*	*EWS*	*EWS*	...	*RMS*	*RMS*
Transcript Training Set	*EWS*(+)*Average*	*EWS*(+)*Average*	*EWS*(+)*Average*	...	*EWS*(+)*Average*	*EWS*(+)*Average*
	EWS(–)*Average*	*EWS*(–)*Average*	*EWS*(–)*Average*	...	*EWS*(–)*Average*	*EWS*(–)*Average*
	BL(+)*Average*	*BL*(+)*Average*	*BL*(+)*Average*	...	*BL*(+)*Average*	*BL*(+)*Average*
	BL(–)*Average*	*BL*(–)*Average*	*BL*(–)*Average*	...	*BL*(–)*Average*	*BL*(–)*Average*
	NB(+)*Average*	*NB*(+)*Average*	*NB*(+)*Average*	...	*NB*(+)*Average*	*NB*(+)*Average*
	NB(–)*Average*	*NB*(–)*Average*	*NB*(–)*Average*	...	*NB*(–)*Average*	*NB*(–)*Average*
	RMS(+)*Average*	*RMS*(+)*Average*	*RMS*(+)*Average*	...	*RMS*(+)*Average*	*RMS*(+)*Average*
	RMS(–)*Average*	*RMS*(–)*Average*	*RMS*(–)*Average*	...	*RMS*(–)*Average*	*RMS*(–)*Average*



SRBCT Neural Network Training

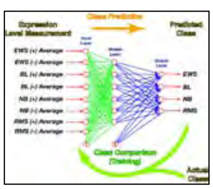
- **Neural network**
  - 8 ensemble-average inputs
  - various # of sigmoidal neurons
  - 4 linear output neurons
  - 4 outputs
- **Training accuracy**
  - Train on all 63 samples
  - Test on all 63 samples
- **100% accuracy**



Leave-One-Out Validation of SRBCT Neural Network

- Remove a single sample
- Train on remaining samples (125 times)
- Evaluate class of the removed sample
- Repeat for each of 63 samples

- 6 sigmoids: 99.96% accuracy (3 errors in 7,875 trials)
- 12 sigmoids: 99.99% accuracy (1 error in 7,875 trials)



Novel-Set Validation of SRBCT Neural Network

- Network always chooses one of four classes (i.e., “unknown” is not an option)
- Test on 25 novel samples (400 times each)
  - 5 EWS
  - 5 BL
  - 5 NB
  - 5 RMS
  - 5 samples of unknown class

- 99.96% accuracy on first 20 novel samples
  (3 errors in 8,000 trials)
- 0% accuracy on unknown classes

Observations of SRBCT Classification using Ensemble Averages

- *t* test identified strong features for classification in this data set
- Neural networks easily classified the four data types
- Caveat: Small, round blue-cell tumors occur in different tissue types
  - Ewing's sarcoma: Bone tissue
  - Burkitt's Lymphoma: Lymph nodes
  - Neuroblastoma: Nerve tissue
  - Rhabdomyosarcoma: Soft tissue

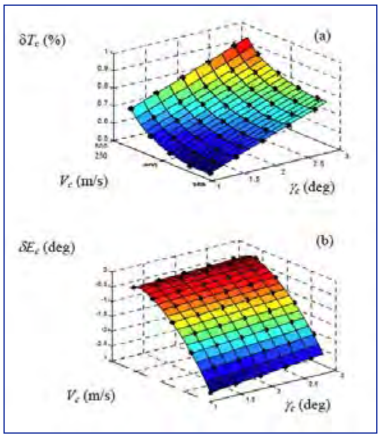
Gene expression (i.e., mRNA) level is linked to tissue difference as well as tumor genetics

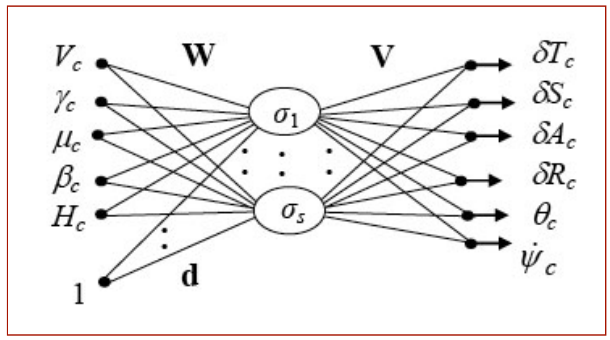
Algebraic Training of a Neural Network

Ferrari, S. and Stengel, R.,
Smooth Function Approximation Using Neural Networks (pdf), *IEEE Trans. Neural Networks*, Vol. 16, No. 1, Jan 2005, pp. 24-38 (with S. Ferrari).

Algebraic Training for Exact Fit to a Smooth Function

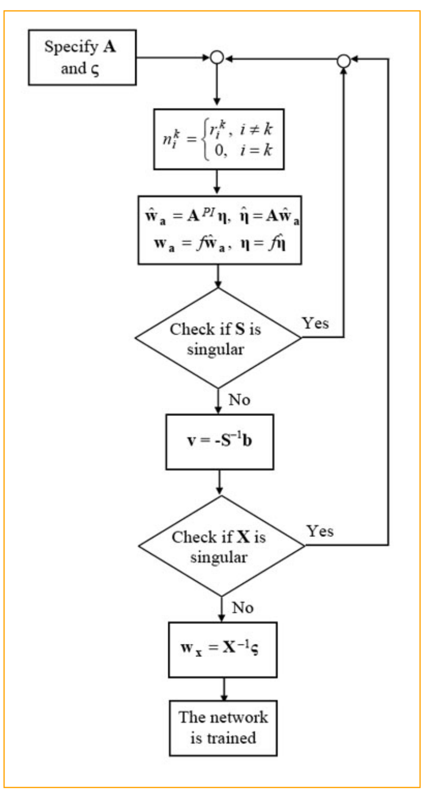
- **Smooth functions define equilibrium control settings at many operating points**
- **Neural network required to fit the functions**

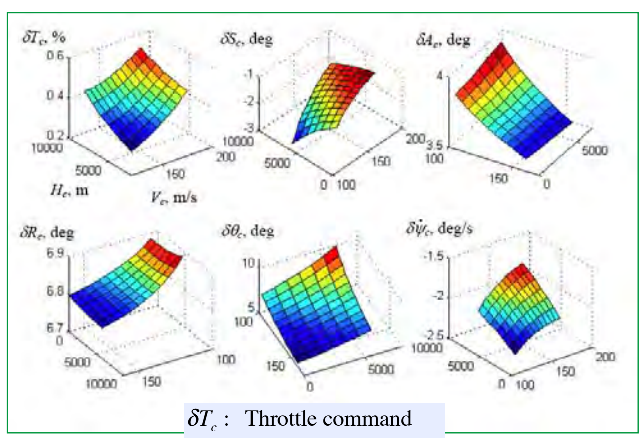




Ferrari and Stengel

Algorithm for Network Training





δT_c : Throttle command
δS_c : Spoiler command
δA_c : Aileron command
δR_c : Rudder command
$\delta \theta_c$: Pitch angle command
$\delta \psi_c$: Yaw rate command

Results for Network Training

45-node example

Algorithm is considerably faster than search methods

Algorithm:	Time (Scaled):	Flops:	Lines of code (MATLAB®):	Epochs:	Final error:
Algebraic	1	2×10^5	8	1	0
Levenberg-Marquardt	50	5×10^7	150	6	10^{-26}
Resilient Backprop.	150	1×10^7	100	150	0.006