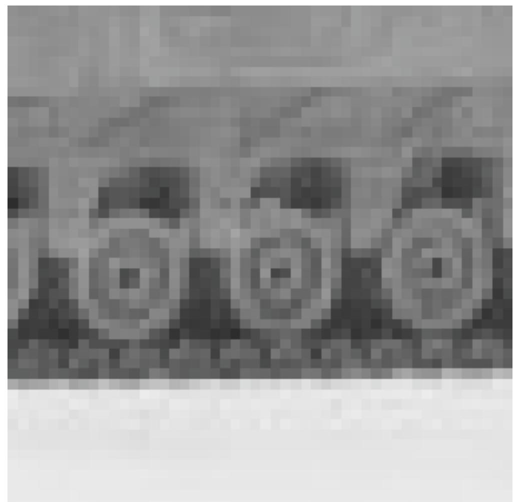
Using Neural Networks for Pattern Classification Problems

Converting an Image

- Camera captures an image
- Image needs to be converted to a form that can be processed by the Neural Network

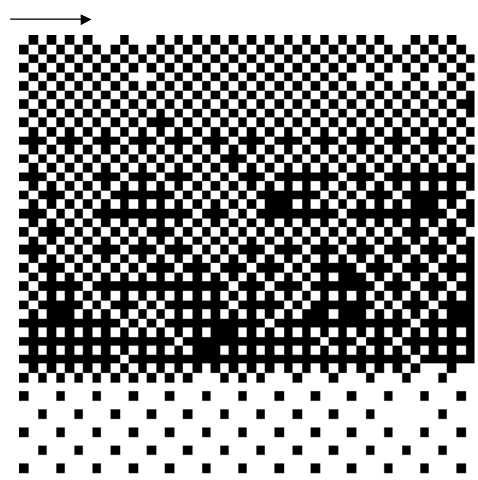
Converting an Image

- Image consists of pixels
- Values can be assigned to color of each pixel
- A vector can represent the pixel values in an image



Converting an Image

- If we let +1 represent black and 0 represent white
- p = [0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 …..



Neural Network Pattern Classification Problem

Tank image = [0 1 0 0 1 1 0 ….]

House image = [1 1 0 0 0 1 0 ….]

Neural Network

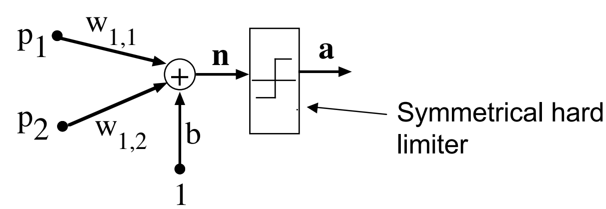
Tank or house ?

Types of Neural Networks

- Perceptron
- Hebbian
- Adeline
- Multilayer with Backpropagation
- Radial Basis Function Network

2-Input Single Neuron Perceptron: Architecture

A single neuron perceptron:

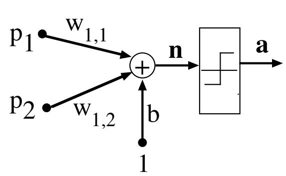


Output:

$$a = \text{hardlims}(\mathbf{Wp} + \mathbf{b}) = \text{hardlims}\left(\begin{bmatrix} w_{1,1} & w_{1,2} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + b \right)$$

$$= \text{hardlims}(w_{1,1}p_1 + w_{1,2}p_2 + b) = \begin{cases} -1, & \text{if } w_{1,1}p_1 + w_{1,2}p_2 + b < 0 \\ +1, & \text{if } w_{1,1}p_1 + w_{1,2}p_2 + b \geq 0 \end{cases}$$

2-Input Single Neuron Perceptron: Example



$$a = \text{hardlims}(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

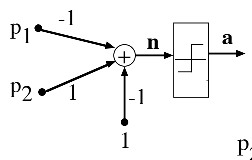
$$= \begin{cases} -1, & w_{1,1}p_1 + w_{1,2}p_2 + b < 0 \\ +1, & w_{1,1}p_1 + w_{1,2}p_2 + b \geq 0 \end{cases}$$

Example: $w_{1,1} = -1$ $w_{1,2} = 1$ $b = -1$

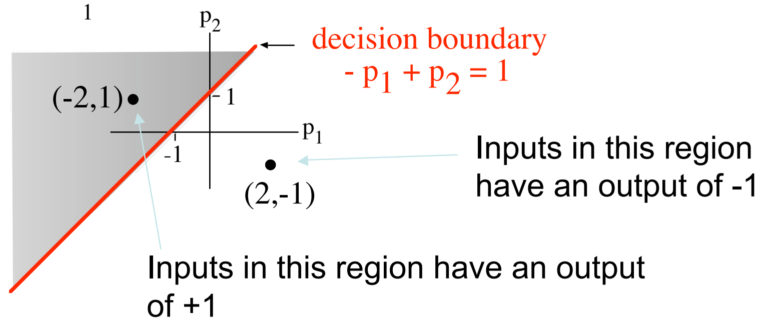
$$a = \begin{cases} -1, & -p_1 + p_2 - 1 < 0 \quad \text{or} \quad -p_1 + p_2 < 1 \\ +1, & -p_1 + p_2 - 1 \geq 0 \quad \text{or} \quad -p_1 + p_2 \geq 1 \end{cases}$$

This separates the inputs $\mathbf{p} = [p_1, p_2]^T$ into two categories separated by the boundary: $-p_1 + p_2 = 1$

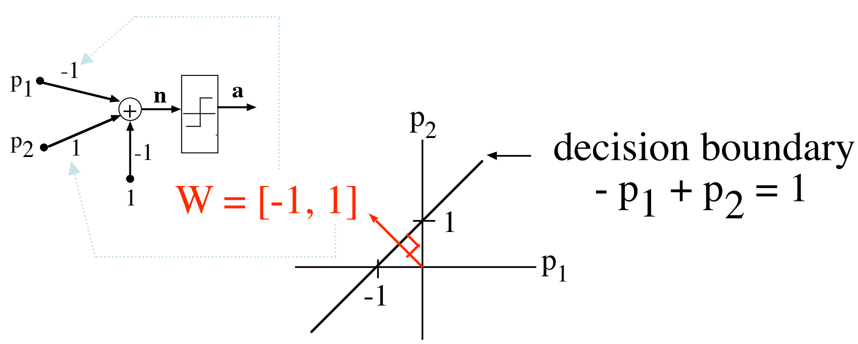
2-Input Single Neuron Perceptron: Decision Boundary



$$a = \begin{cases} -1, & -p_1 + p_2 - 1 < 0 \quad \text{or} \quad -p_1 + p_2 < 1 \\ +1, & -p_1 + p_2 - 1 \geq 0 \quad \text{or} \quad -p_1 + p_2 \geq 1 \end{cases}$$

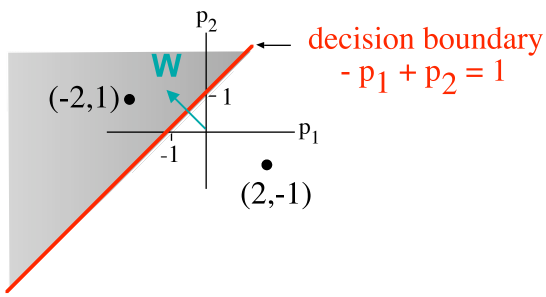


2-Input Single Neuron Perceptron: Weight Vector



- The weight vector, **W**, is orthogonal to the decision boundary

2-Input Single Neuron Perceptron: Weight Vector



- **W** points towards the class with an output of +1

Simple Perceptron Design

- The design of a simple perceptron is based upon:
  - A single neuron divides inputs into two classifications or categories
  - The weight vector, W, is orthogonal to the decision boundary
  - The weight vector, W, points towards the classification corresponding to the “1” output

Orthogonal Vectors

- For any hyperplane of the form:

  $a_1p_1 + a_2p_2 + a_3p_3 + \ . \ . \ . + a_np_n = b$

  the vector c*[a_1, a_2, …, a_n] is orthogonal to the hyperplane (where c is a constant).

$$-p_1 + p_2 = -1 * p_1 + 1 * p_2 = 1$$

$$\mathbf{W} = [\,-1\,,\,1\,]$$

AND Gate: Description

- A perceptron can be used to implement most logic functions
- Example: Logical AND Truth table:

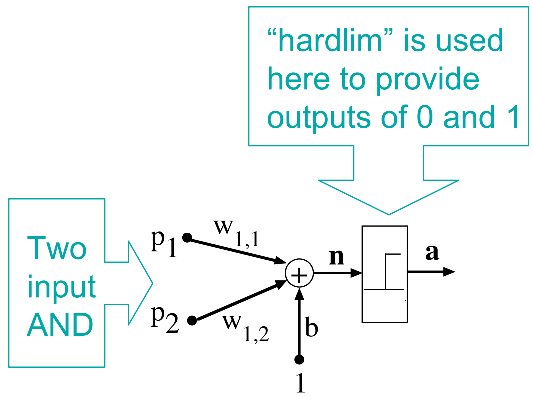
Inputs		Output
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate: Architecture

Input/Target pairs:

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

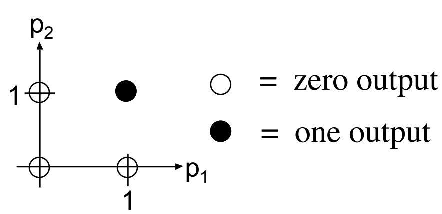
$$\left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 0\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



AND Gate: Graphical Description

- Graphically:

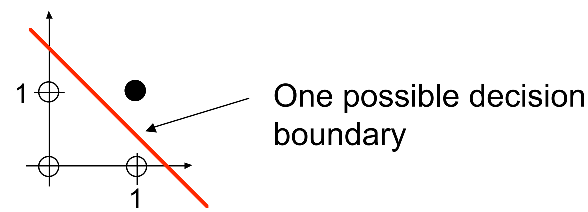
Inputs		Output
0	0	0
0	1	0
1	0	0
1	1	1



- Where do we place the decision boundary?

AND Gate: Decision Boundary

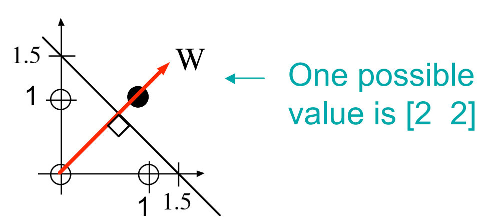
- There are an infinite number of solutions



- What is the corresponding value of **W**?

AND Gate: Weight Vector

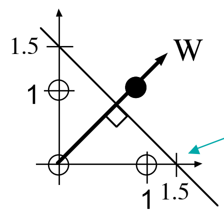
- **W** must be orthogonal to the decision boundary
- **W** must point towards the class with an output of 1



- Output: $\mathbf{a} = \text{hardlim}\left\{\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}p_1\\p_2\end{bmatrix} + b\right\} = \text{hardlim}\{\underbrace{2p_1 + 2p_2 + b}_{\text{Decision boundary}}\}$

AND Gate: Bias

- Decision Boundary: $2p_1 + 2p_2 + b = 0$

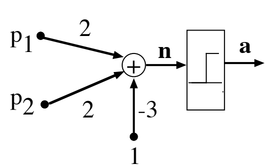


Passes through (1.5, 0)

- At (1.5, 0): 2(1.5) + 2(0) + b = 0 b = -3

AND Gate: Final Design

- Final Design:



$$\mathbf{a} = \mathrm{hardlim}\left\{ \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} - 3 \right\}$$

- Test: $\mathbf{a} = \mathrm{hardlim}\left\{ \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 3 \right\} = 0$

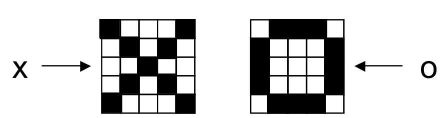
$\mathbf{a} = \mathrm{hardlim}\left\{ \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 3 \right\} = 0$ $\mathbf{a} = \mathrm{hardlim}\left\{ \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - 3 \right\} = 0$ $\mathbf{a} = \mathrm{hardlim}\left\{ \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 3 \right\} = 1$

Perceptron Learning Rule

- Most real problems involve input vectors, p, that have length greater than three
- Images are described by vectors with 1000s of elements
- Graphical approach is not feasible in dimensions higher than three
- An iterative approach known as the Perceptron Learning Rule is used

Character Recognition Problem

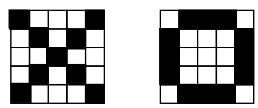
- Given: A network has two possible inputs, “x” and “o”. These two characters are described by the 25 pixel (5 x 5) patterns shown below.



- Problem: Design a neural network using the perceptron learning rule to correctly identify these input characters.

Character Recognition Problem: Input Description

- The inputs must be described as column vectors
- Pixel representation: 0 = white
  1 = black



The “x” is represented as: $[\,1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\,]^T$

The “o” is represented as: $[\,0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\,]^T$

Character Recognition Problem: Output Description

- The output will indicate that either an “x” or “o” was received
- Let: 0 = “o” received
  1 = “x” received

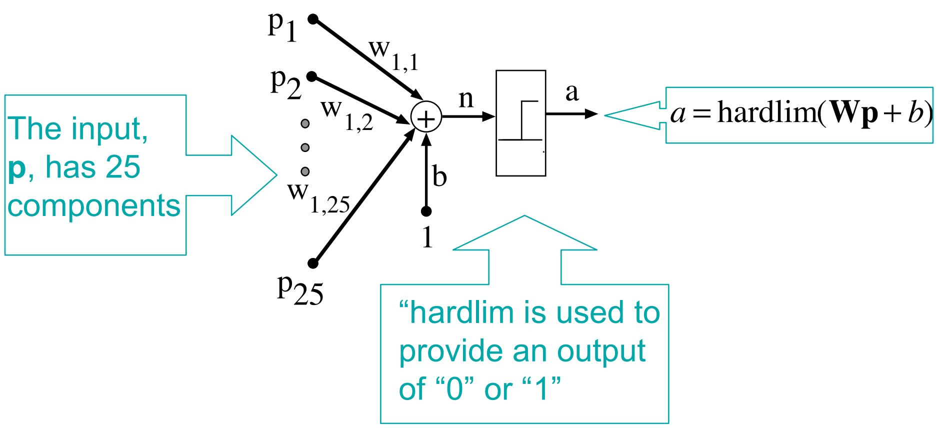
A hard limiter will be used

- The inputs are divided into two classes requiring a single neuron
- Training set:

$\mathbf{p_1} = [\,1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\,]^T$, $t_1 = 1$

$\mathbf{p_2} = [\,0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\,]^T$, $t_2 = 0$

Character Recognition Problem: Network Architecture



Perceptron Learning Rule: Summary

- Step 1: Initialize **W** and **b** (if non zero) to small random numbers.
- Step 2: Apply the first input vector to the network and find the output, a.
- Step 3: Update **W** and **b** based on:

$$\mathbf{W}_{new} = \mathbf{W}_{old} + (\mathbf{t}-\mathbf{a})\mathbf{p}^T$$

$$\mathbf{b}_{new} = \mathbf{b}_{old} + (\mathbf{t}-\mathbf{a})$$

- Repeat steps 2 and 3 for all input vectors repeatedly until the targets are achieved for all inputs

Character Recognition Problem: Perceptron Learning Rule

- Step 1: Initialize **W** and **b** (if non zero) to small random numbers.
  - Assume **W** = [0 0 . . . 0] (length 25) and b = 0
- Step 2: Apply the first input vector to the network
  - $\mathbf{p_1}$ = [1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1]^T, $t_1 = 1$
  - $a = \text{hardlim}(\mathbf{W(0)p_1} + b(0)) = \text{hardlim}(0) = 1$
- Step 3: Update W and b based on:

$$\mathbf{W_{new} = W_{old}} + (t\text{-}a)\mathbf{p_1}^{\mathbf{T}} \mathbf{= W_{old}} + (1\text{-}1)\mathbf{p_1}^{\mathbf{T}}$$

$$\mathbf{= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]}$$

$$b_{new} = b_{old} + (t\text{-}a) = b_{old} + (1\text{-}1) = 0$$

Character Recognition Problem: Perceptron Learning Rule

- Step 2 (repeated): Apply the second input vector to the network
  - $\mathbf{p_2}$ = [0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0]^T, $t_2 = 0$
  - $a = \text{hardlim}(\mathbf{W}(1)\mathbf{p}_2 + b(1)) = 1$
- Step 3 (repeated): Update W and b based on

$$\mathbf{W_{new} = W_{old}} + (t\text{-}a)\mathbf{p_1}^{\mathbf{T}} \mathbf{= W_{old}} + (0\text{-}1)\mathbf{p_2}^{\mathbf{T}}$$

$$\mathbf{=} [0\ {-1}\ {-1}\ {-1}\ 0\ {-1}\ 0\ 0\ 0\ {-1}\ {-1}\ 0\ 0\ 0\ {-1}\ {-1}\ 0\ 0\ 0\ {-1}\ 0\ {-1}\ {-1}\ {-1}\ 0]$$

$$b_{new} = b_{old} + (t\text{-}a) = b_{old} + (0\text{-}1) = -1$$

Character Recognition Problem: Perceptron Learning Rule

W	b	p	t	a	e
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0	p_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 0 0 0 0]	0	p_2	0	1	-1
[0 -1 -1 -1 0 -1 0 0 0 -1 -1 0 0 0 -1 -1 0 0 0 -1 0 -1 -1 -1 0]	-1	p_1	1	0	1
[1 -1 -1 -1 1 -1 1 0 1 -1 -1 0 1 0 -1 -1 1 0 1 -1 1 -1 -1 -1 1]	0	p_2	0	0	0
[1 -1 -1 -1 1 -1 1 0 1 -1 -1 0 1 0 -1 -1 1 0 1 -1 1 -1 -1 -1 1]	0	p_1	1	1	0

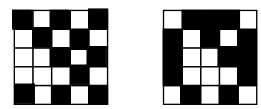
Character Recognition Problem: Results

- After three epochs, **W** and b converge to:
  - **W** = [1 -1 -1 -1 1 -1 1 0 1 -1 -1 0 1 0 -1 -1 1 0 1 -1 1 -1 -1 -1 1]
  - b = 0
- One possible solution based on the initial condition selected. Other solutions are obtained when the initial values of W and b are changed.
- Check the solution: a = hardlim(W*p + b) both both inputs

Character Recognition Problem: Results

- How does this network perform in the presence of noise?

x and o with three
pixel errors in each



- For the “x” with noise:

  a = hardlim{W*[1 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1] + 0} = 1
- For the “o” with noise:

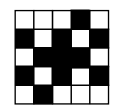
  a = hardlim{W*[0 1 1 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0] + 0} = 0
- The network recognizes both the noisy x and o.

Character Recognition Problem: Simulation

- Use MATLAB to perform the following simulation:
  – Apply noisy inputs to the network with pixel errors ranging from 1 to 25 per character and find the network output
  – Each type of error (number of pixels) was repeated 1000 times for each character with the incorrect pixels being selected at random
  – The network output was compared to the target in each case.
  – The number of detection errors was tabulated.

Character Recognition Problem: Performance Results

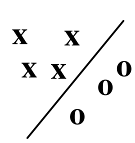
No. of Pixel Errors	No. of Character Errors		Probability of Error	
	x	o	x	o
1 - 9	0	0	0	0
10	96	0	.10	0
11	399	0	.40	0
12	759	58	.76	.06
13	948	276	.95	.28
14	1000	616	1	.62
15	1000	885	1	.89
16 - 25	1000	1000	1	1



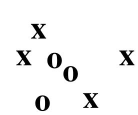
An "o" with 11 pixel errors

Perceptrons: Limitations

- Perceptrons only work for inputs that are linearly separable



Linearly separable



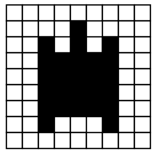
Not Linearly separable

Other Neural Networks

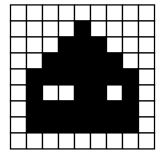
- How do the other types of neural networks differ from the perceptron?
  - Topology
  - Function
  - Learning Rule

Perceptron Problem: Part 1

- Design a neural network that can identify a tank and a house.
  - Find W and b by hand as illustrated with the x-o example.
  - Use the Neural Network Toolbox to find W and b



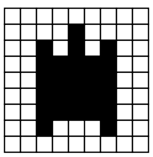
Tank
(t = 1)



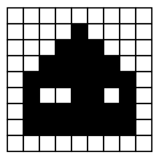
House
(t = 0)

Perceptron Problem: Part 2

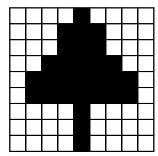
- Design a neural network that can find a tank among houses and trees.
  - Repeat the previous problem but now with a tree included.
  - Both the house and tree have targets of zero.



Tank
(t = 1)



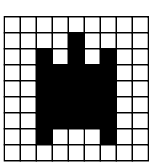
House
(t = 0)



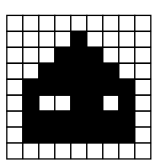
Tree
(t = 0)

Perceptron Problem: Part 3

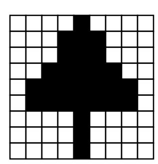
- Design a neural network that can find a tank among houses, trees and other items.
  - Create other images on the 9 x 9 grid.
  - Everything other than a tank will have a target of zero.
  - How many items can you introduce before the perceptron learning rule no longer converges?



Tank
(t = 1)



House
(t = 0)

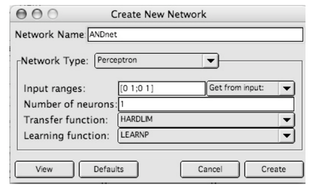


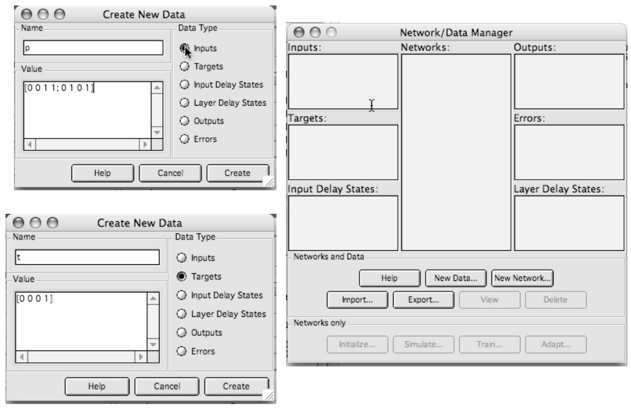
Tree
(t = 0)

+ ????

MATLAB: Neural Network Toolbox

- >> nntool





MATLAB: Neural Networks Toolbox

- Go to MATLAB Help and review the documentation on the Neural Networks Toolbox
- Use the GUI interface (>> nntool) to reproduce the results you obtained for the perceptron (tank vs. house, tree, etc.)
- Data can be imported/exported from the workspace to the NN Tool.