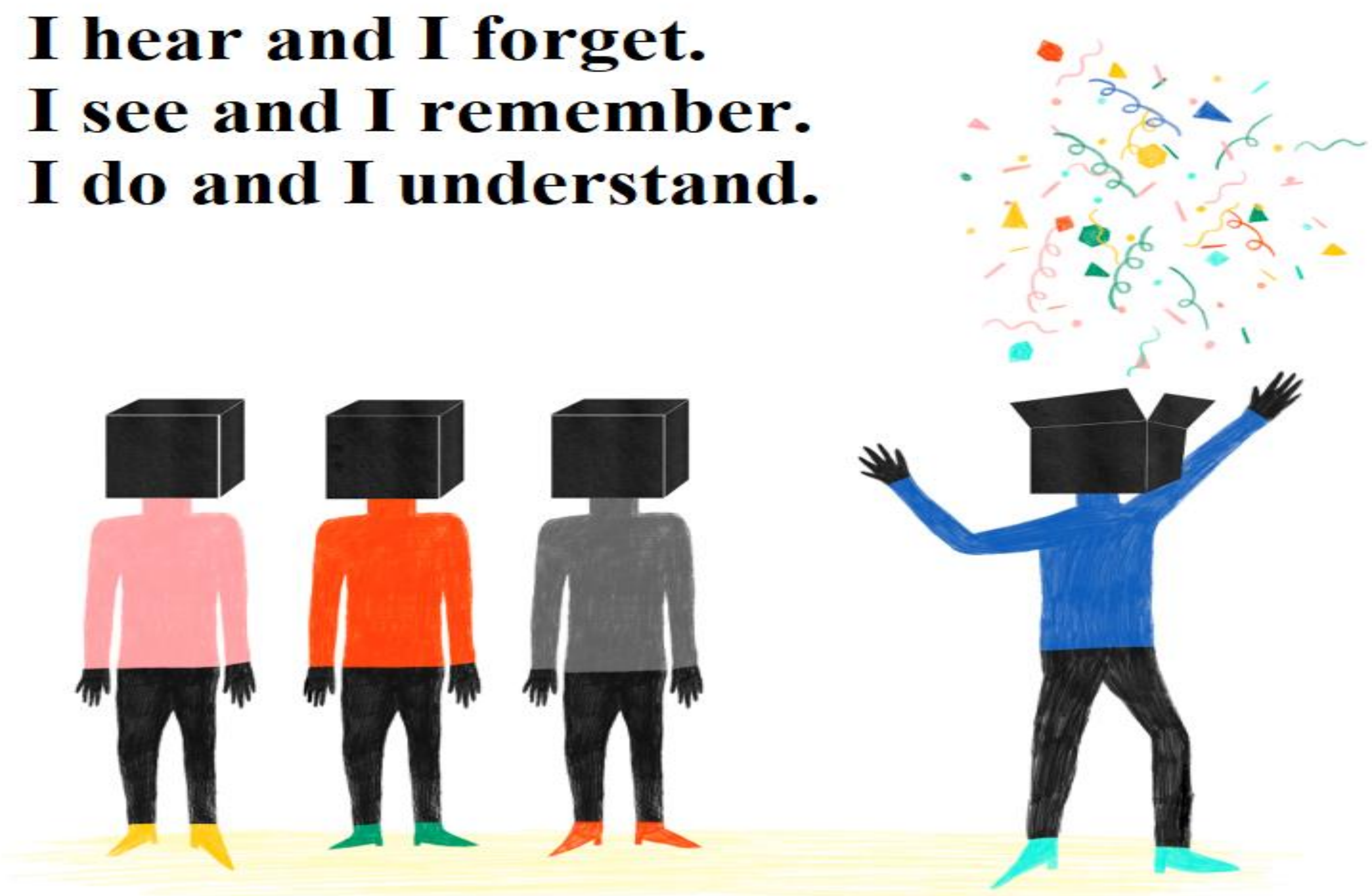
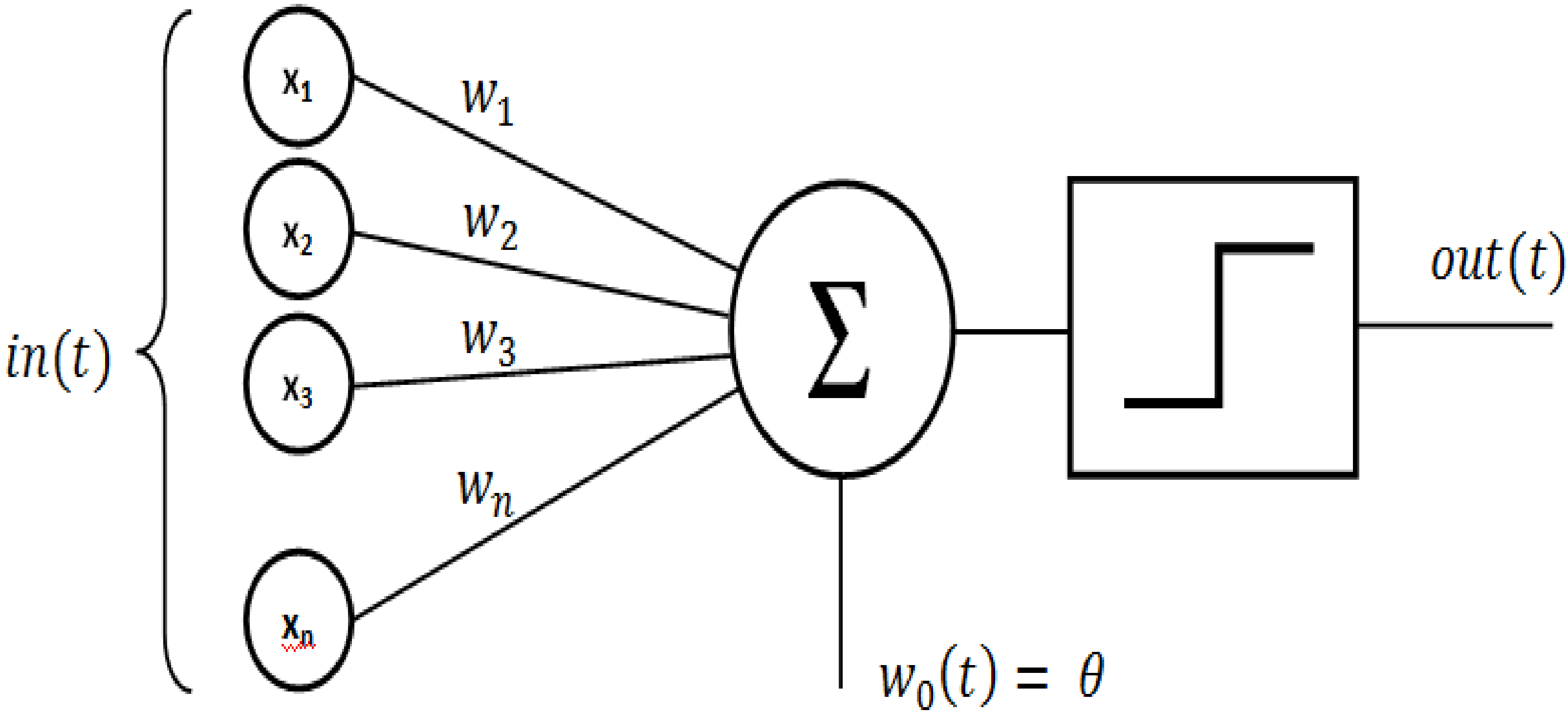


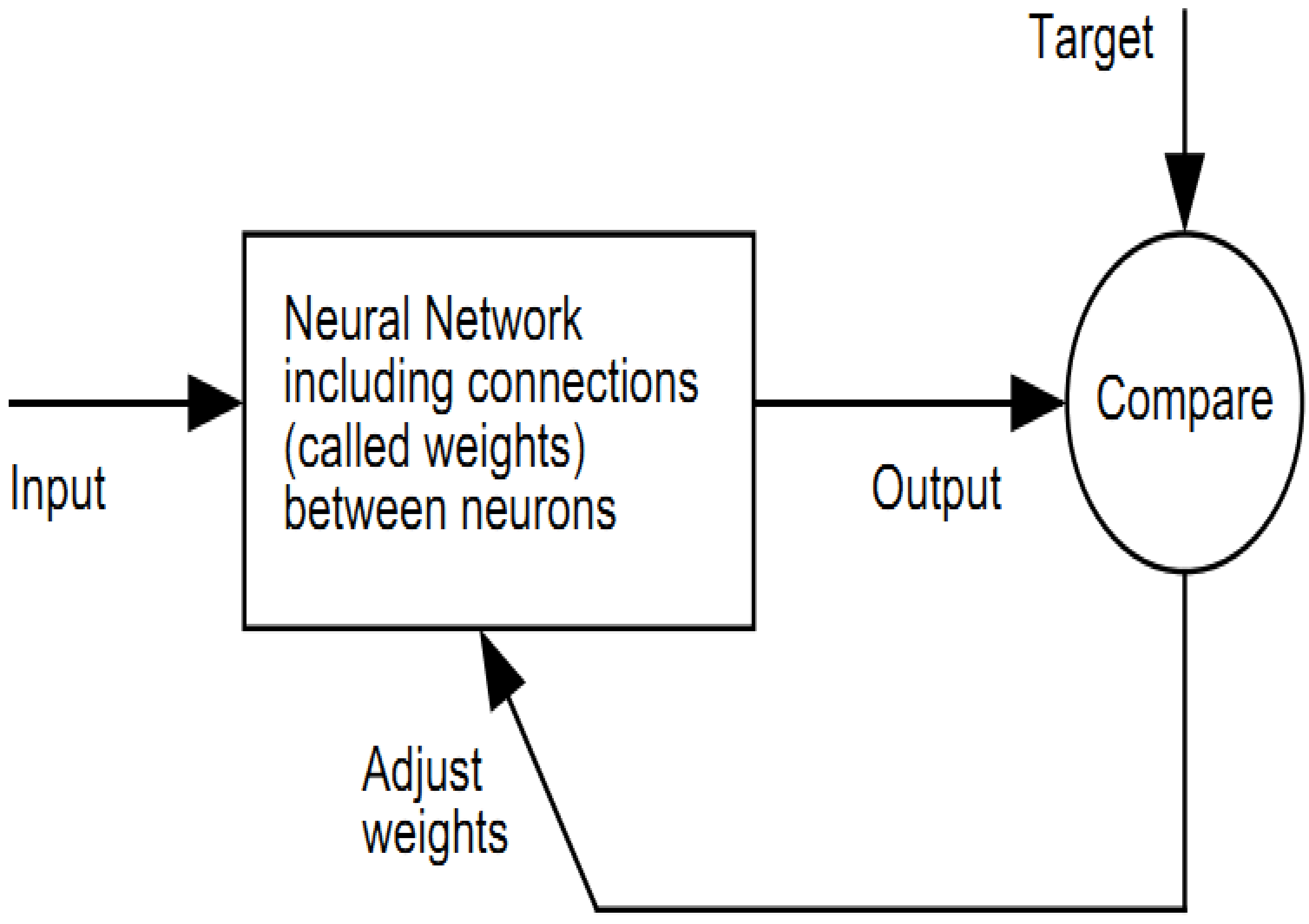
Chandan Verma

Corporate Trainer(Machine Learning,AI,Cloud Computing,IOT)

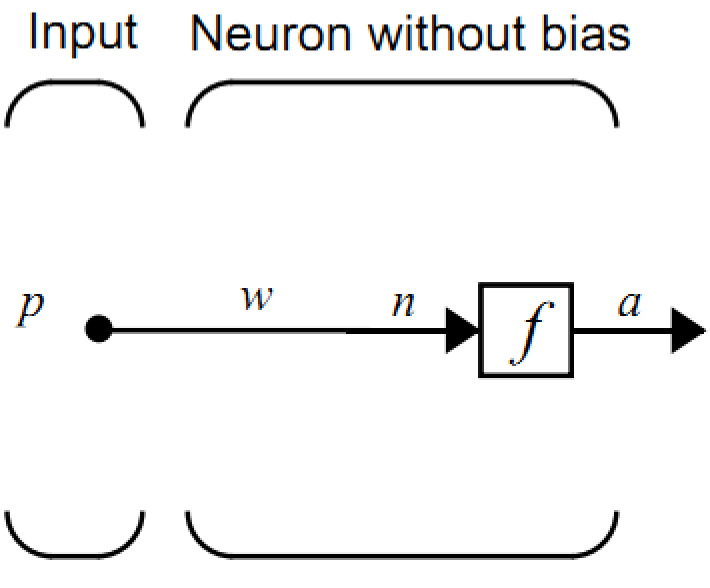
www.facebook.com/verma.chandan.070

Perceptrons

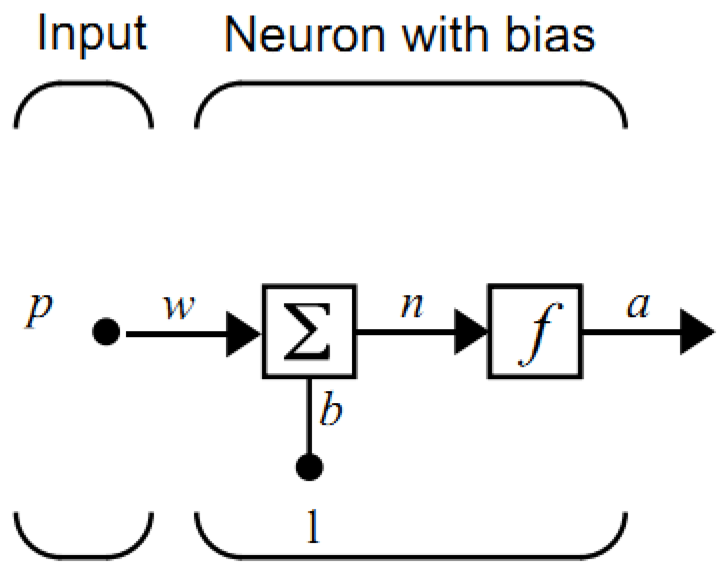



Target
Neural Network
including connections
(called weights)
between neurons
Input
Output
Compare
Adjust
weights

Simple Neuron

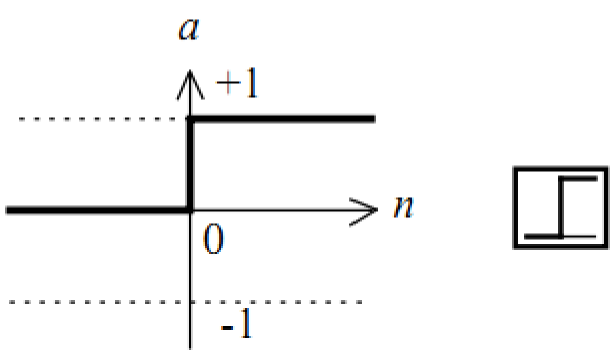


$a = f(wp)$



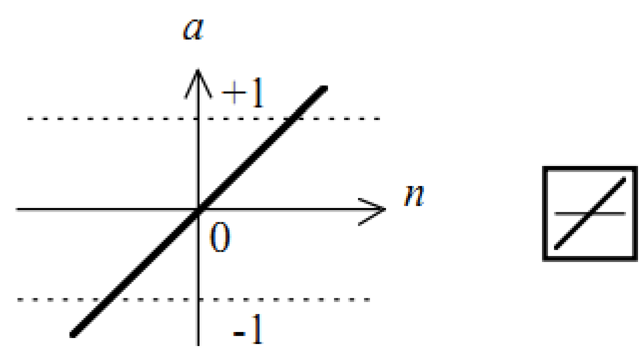
$a = f(wp + b)$

Transfer Functions



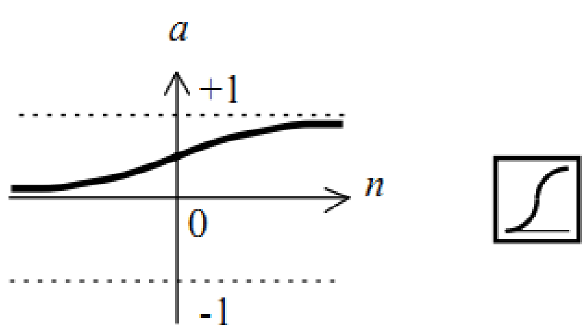
$a = hardlim(n)$

Hard-Limit Transfer Function



$a = purelin(n)$

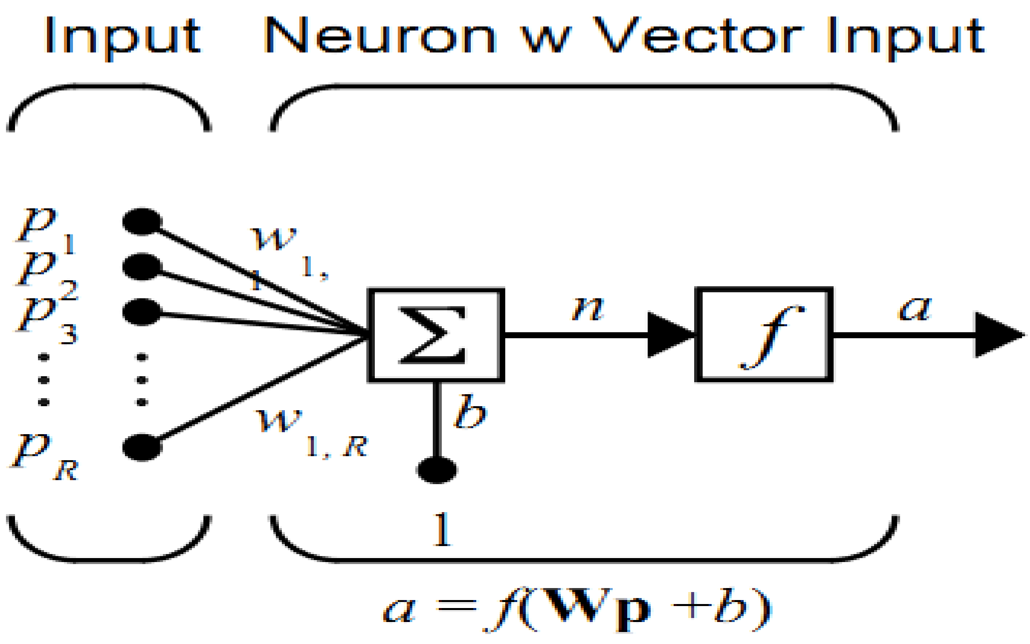
Linear Transfer Function



$a = logsig(n)$

Log-Sigmoid Transfer Function

Perceptrons



$$n = W*p + b$$

hardlim

hardlim is a neural transfer function. Transfer functions calculate a layer’s output from its net input.

```
>> x= -5:0.1:5;
>> y=hardlim(x);
>> plot(x,y)
```

tansig

Transfer functions calculate a layer's output from its net input.
each element of N in is squashed from the interval **[-inf inf]** to the interval [-1 1]with an "S-shaped" function.

```
>> x=-5:.1:5;
>> y=tansig(x);
>> plot(x,y)
```

Example

```
p=[1;-1];
w=[2 0];
b=1;
n=w*p+b;
a=hardlim(n);
disp(a)
```

```
p=[1;-1];
w=[2 0];
b=1;
n=w*p+b;
a=tansig(n);
disp(a)
```

Multiple Layer of Neuron

Input Vector 2 Column

p=[1;-1]

Laybe1 :Node=3

w1=[2 0;1 2; 1 3];

b1=[1;2;3];

Layer2: node=2

w2=[2 0 1;1 2 1];

b2=[0;0]

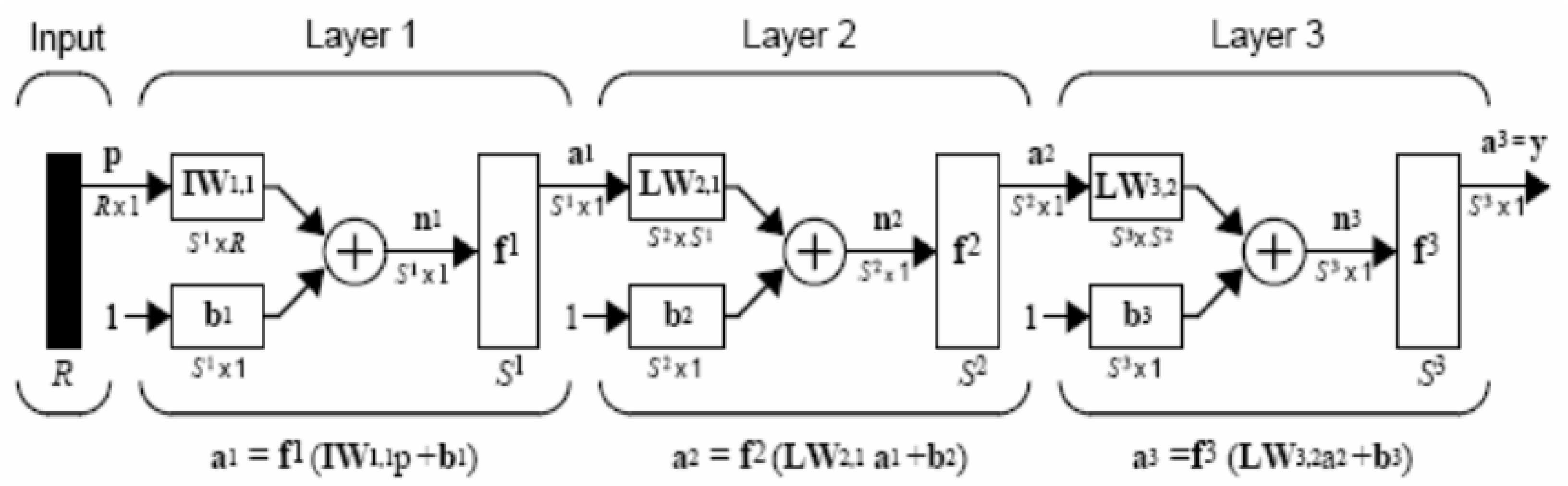
Layer3:node=1

w3=[2 0];

b3=[1]

Output

```
outputnet=
tansig(w3*(tansig(w2*(tansig(w1*p+b1)) +b2)) +b3);
disp(outputnet)
```



Creating a Linear Neuron (newlin)

```
net = newlin(P,S,ID,LR)
```

P input vectors.
S - Number of elements in the output vector.
ID - Input delay vector, default = [0].
LR - Learning rate, default = 0.01;

```
net=newlin([-1 1;-1 1],1);
w=net.IW{1,1}
b=net.b{1}
net.IW{1,1}=[2,3];
w=net.IW{1,1}
net.b{1}=[-4];
```

newp-Create a perceptron

Perceptrons are used to solve simple (i.e. linearly separable) classification problems.

net = newp(pr,s,tf,lf)

PR - Rx2 matrix of min and max values for R input elements.
S - Number of neurons.
TF - Transfer function, default = 'hardlim'.
LF - Learning function, default = 'learnp'.

Train a neural network

[NET,TR] = train(NET,X,T)

takes a network NET, input data Xand target data T and returns the network after training it, and a training record TR.

```
net=newp([0 1;0 1],1);
p=[0 0 1 1;0 1 0 1];
t=[0 0 0 1];
net.IW{1}=[1,1];
net.b{1}=1;
net.trainParam.nettepochs=2;
net=train(net,p,t);
a=sim(net,p);
disp(a)
```

perceptron

perceptron(hardlimitTF,perceptronLF)

Perceptrons are simple single-layer binary classifiers, which divide the input space with a linear decision boundary.

Suppose you have the following classification problem and would like to solve it with a single vector input, two-element perceptron network.

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1\right\} \left\{\mathbf{p}_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0\right\} \left\{\mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1\right\}$$

$\mathbf{W}(0)=[0 \quad 0] \quad b(0)=0$

Start by calculating the perceptron's output *a* for the first input vector $\mathbf{p}_1$, using the initial weights and bias.

***a*=*hardlim*(W(0)p1+*b*(0))**

=*hardlim*([0 0][2 2]'+0)

=*hardlim*(0)=1

The output *a* does not equal the target value t_1, so use the perceptron rule to find the incremental changes to the weights and biases based on the error.

$e = t1 - \alpha = 0 - 1 = -1$
$\Delta\mathbf{W} = e\mathbf{p}T1 = (-1)[2\ 2] = [-2\ -2]$
$\Delta b = e = (-1) = -1$

new weights and bias using the perceptron update rules.
$\mathbf{W}new = \mathbf{W}old + \mathbf{ep}T$
$= [0\ 0] + [-2\ -2] = [-2\ -2] = \mathbf{W}(1)$
$bnew = bold + e =$
$0 + (-1) = -1 = b(1)$

next input vector, $\mathbf{p}_2$. The output is calculated below.
$\alpha = hardlim(\mathbf{W}(1)\mathbf{p}2 + b(1))$
$= hardlim([-2\ -2][1\ -2]\ -1)$
$= hardlim(1) = 1$

On this occasion, the target is 1, so the error is zero. Thus there are no changes in weights or bias, so $\mathbf{W}(2) = \mathbf{W}(1) = [-2\ -2]$ and $b(2) = b(1) = -1$.

```
>> net=perceptron;
>> p=[2;2];
>> t=[0];
>> net.trainParam.epochs=1;
>> net=train(net,p,t);
>> w=net.IW{1,1}
>> b=net.b{1}
```

```
>> net=perceptron;
>> net.trainParam.epochs=1;
>> p = [[2;2] [1;-2] [-2;2] [-1;1]];
>> t = [0 1 0 1];
>> net = train(net,p,t);
>> w = net.iw{1,1}, b = net.b{1}
>> o=net(p)
```

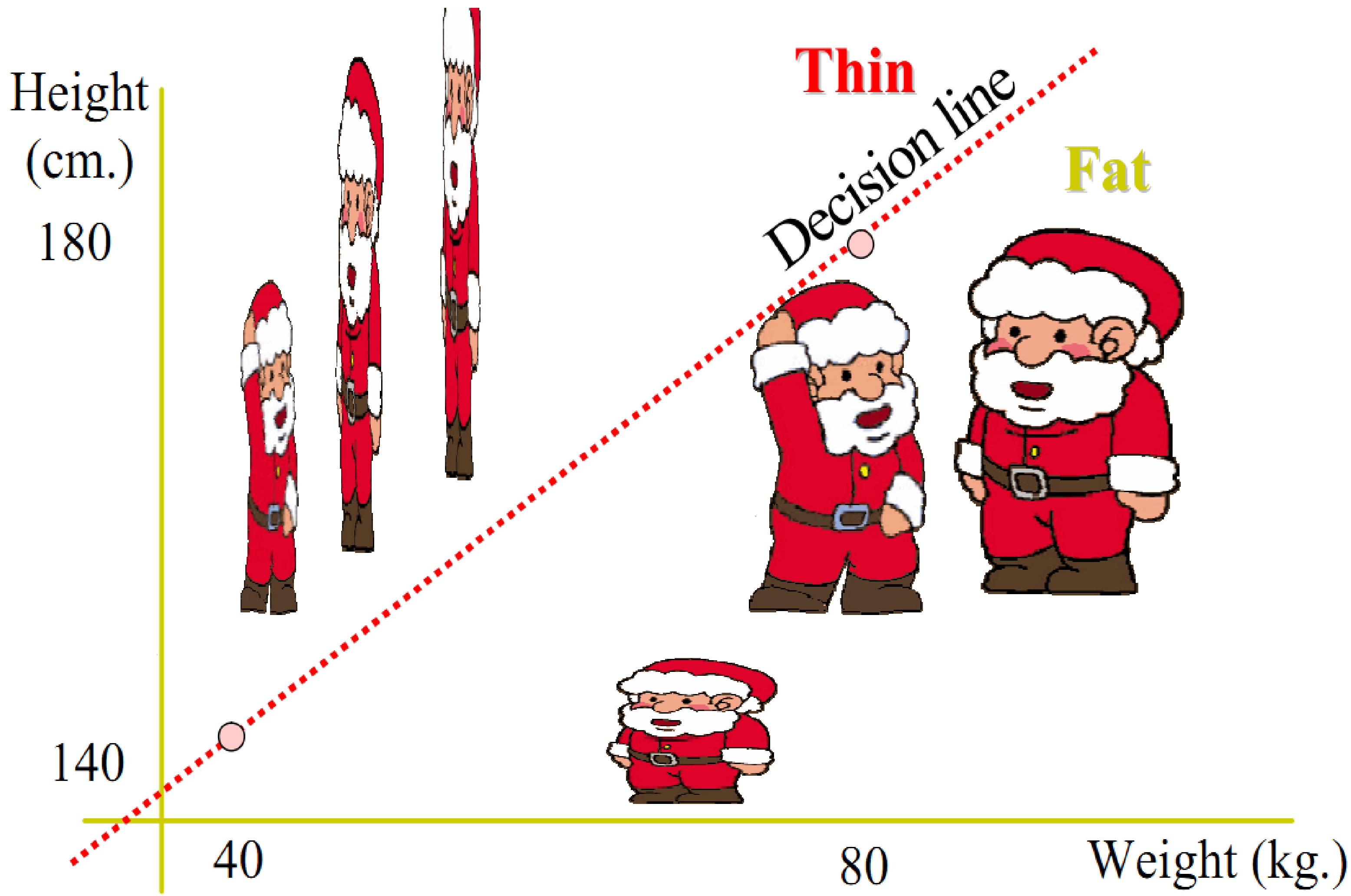
Increasing Epochs

```
>> net.trainParam.epochs = 1000;
>> net = train(net,p,t);
>> w = net.iw{1,1}, b = net.b{1}
>> a = net(p)
>> a-t
```

logical-OR

```
x = [0 0 1 1; 0 1 0 1];
t = [0 1 1 1];
net = perceptron;
net = train(net,x,t);
view(net)
y = net(x)
```

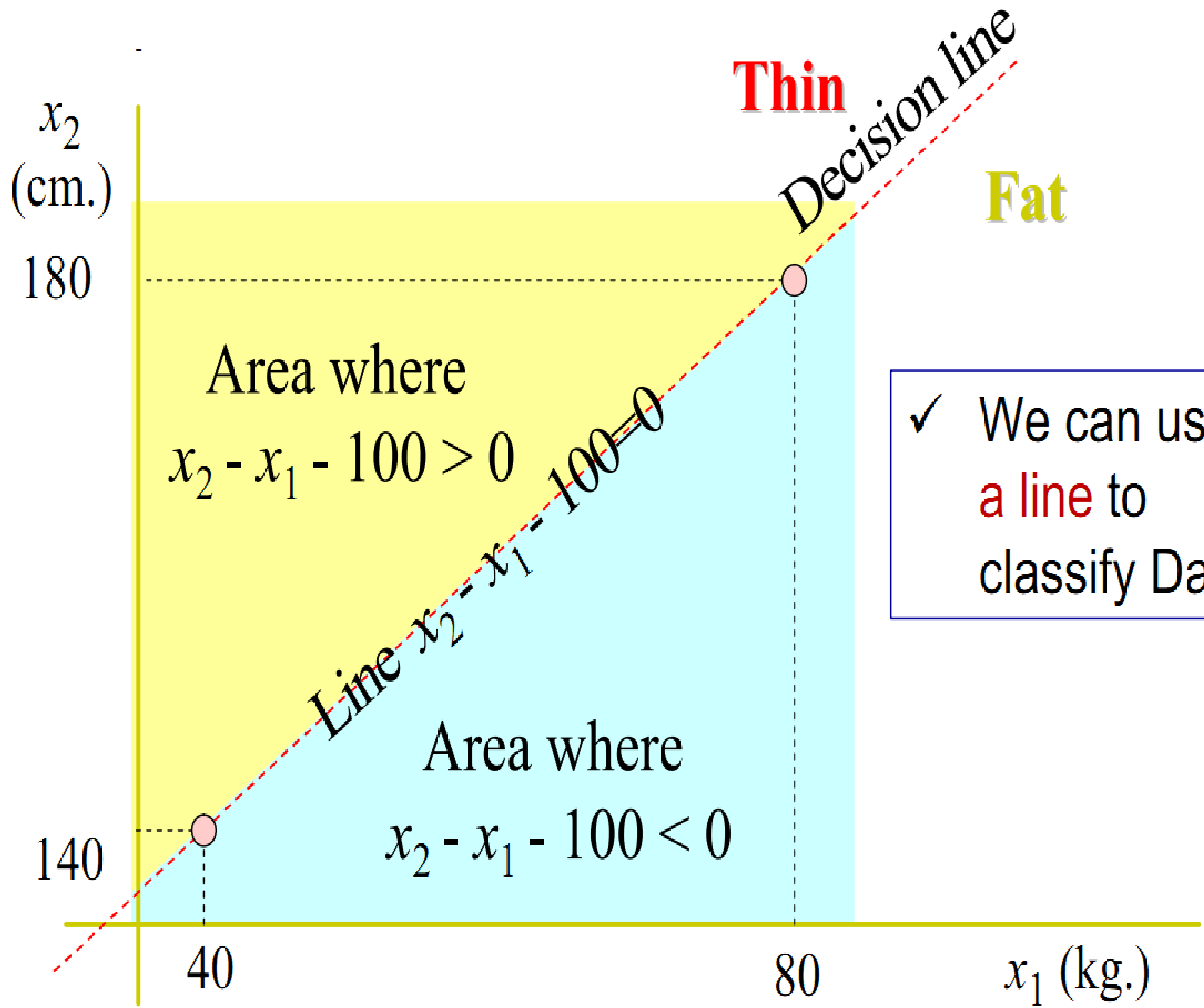
How can we classify fat and thin boys



use a mathematical function

$$y = \begin{cases} 1 \text{ (thin)} & \text{if } \; x_2 - x_1 - 100 \geq 0 \\ 0 \text{ (fat)} & \text{if } \; x_2 - x_1 - 100 < 0 \end{cases}$$

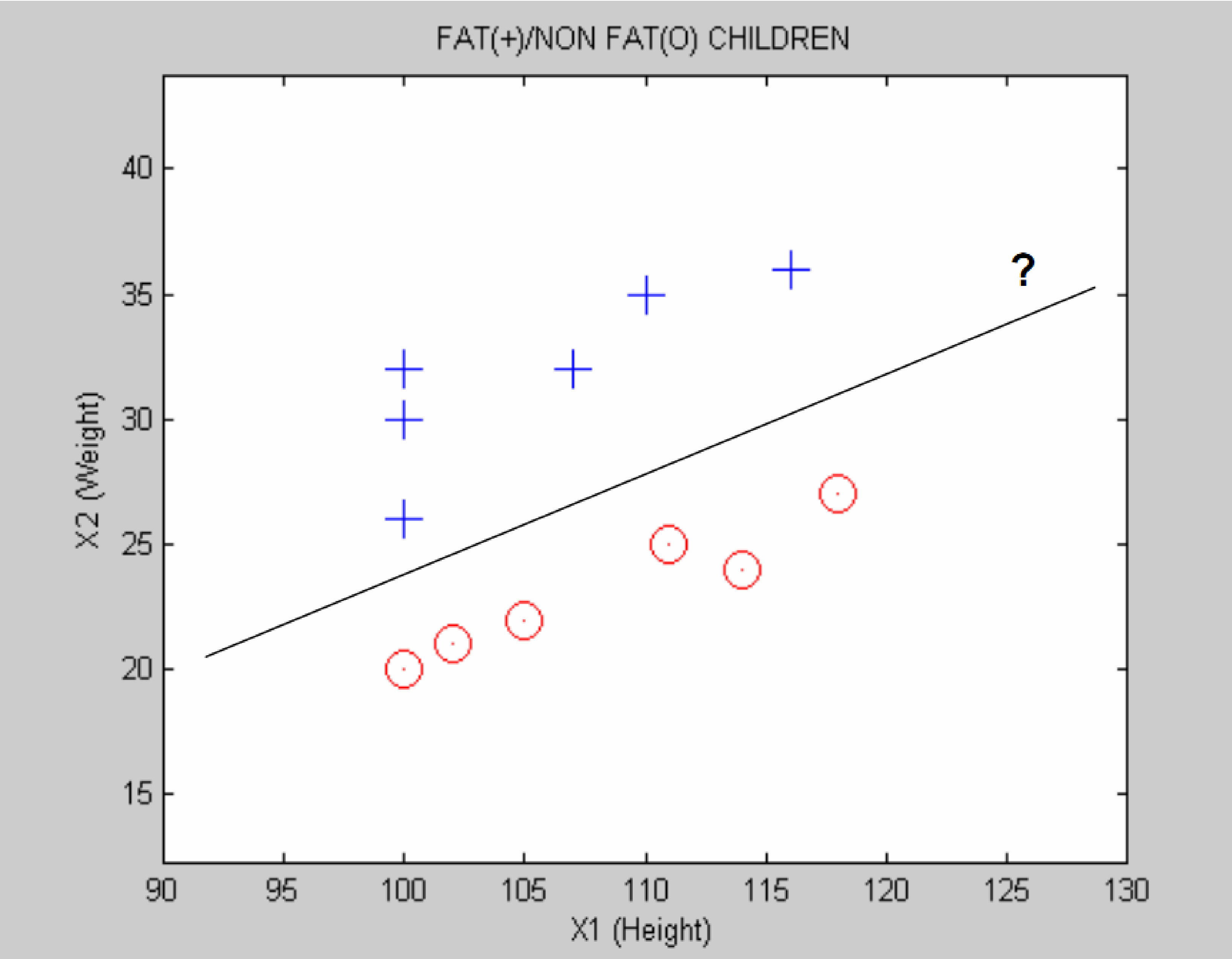
$$\begin{aligned} y &= g(w_1 x_1 + w_2 x_2 - \mu) \\ &= g(-x_1 + x_2 - 100) \end{aligned}$$



✓ We can use a line to classify Data.

Training Samples

No.	Height (Cm.)	Weight (Kg.)	Fat/Not Fat (+1/-1)
1.	100	20	-1
2.	100	26	+1
3.	100	30	+1
4.	100	32	+1
5.	102	21	-1
6.	105	22	-1
7.	107	32	+1
8.	110	35	+1
9.	111	25	-1
10.	114	24	-1
11.	116	36	+1
12.	118	27	-1


FAT(+)/NON FAT(O) CHILDREN
X2 (Weight)
X1 (Height)
?
40
35
30
25
20
15
90
95
100
105
110
115
120
125
130

```
net = newp([90 130; 10 50],1);
P = [100 20; 100 26; 100 30; 100 32; 102 21; 105 22; 107
32; 110 35; 111 25;114 24; 116 36; 118 27]';
T = [0; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0]';
net.trainParam.epochs = 40;
net = train(net,P,T);
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```

```
NP = [120 32]';
hold on;
plot(NP(1),NP(2),'*r');
Y = sim(net,NP)
```

Classification of linearly separable data with a perceptron

1. Define input and output data
2. Create and train perceptron
3. Plot decision boundary

plotpv
Plot perceptron input/target vectors

Define input and output data

```
close all,clear all,clc
N=20;
offset=5;
x=[randn(2,N) randn(2,N)+offset];
y=[zeros(1,N) ones(1,N)];
```

```
figure(1)
plotpv(x,y)
```

Plotting input/output

Plotpv

plotpv Plot perceptron input/target vectors.

```
x = [0 0 1 1; 0 1 0 1];
t = [0 0 0 1];
plotpv(x,t)
```

Create and train perceptron

```
net =perceptron;
net=train(net,x,y);
view(net)
```

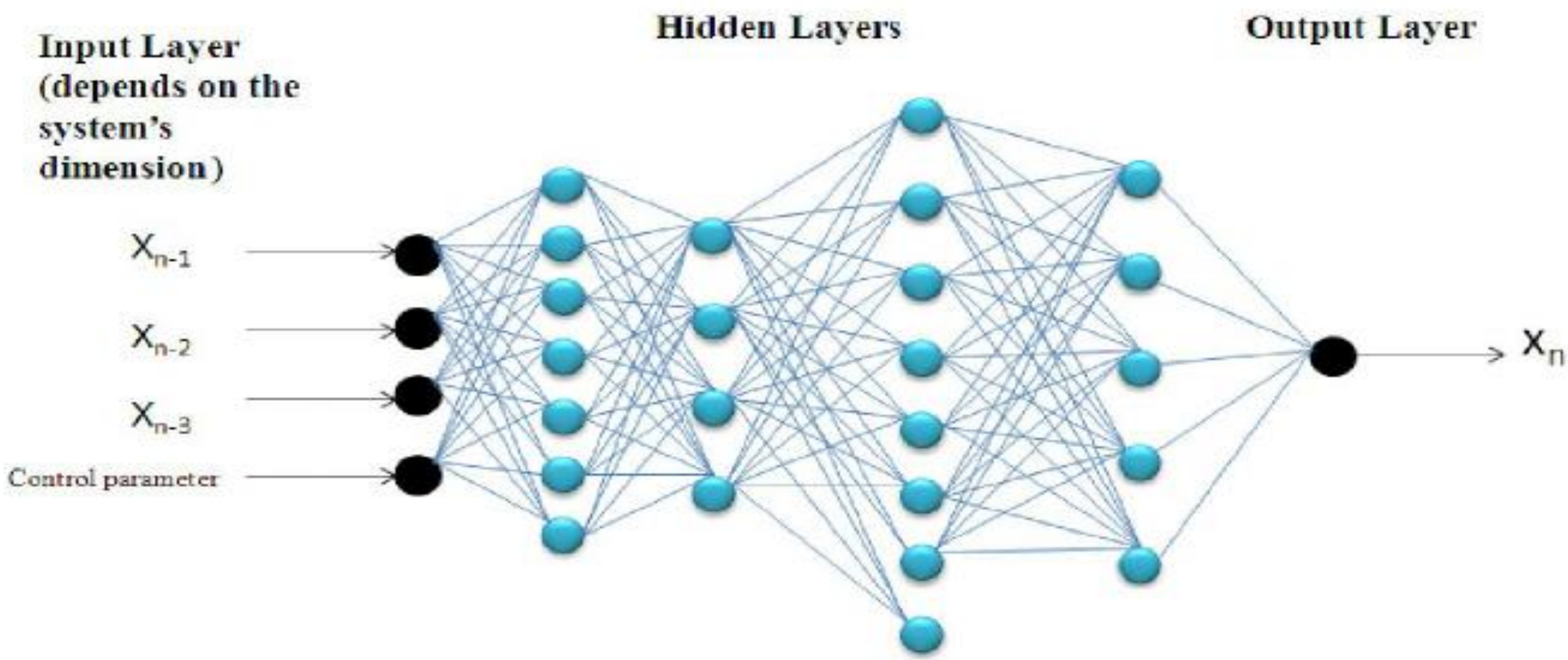
```
figure(1)
plotpc(net.IW{1},net.b{1});
```

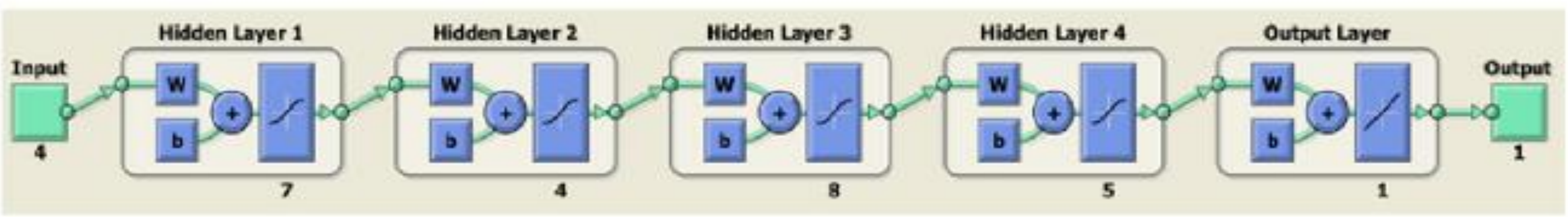
Plot decision boundary

Plot a classification line on a perceptron vector plot.

```
plotpc(W,B)
```

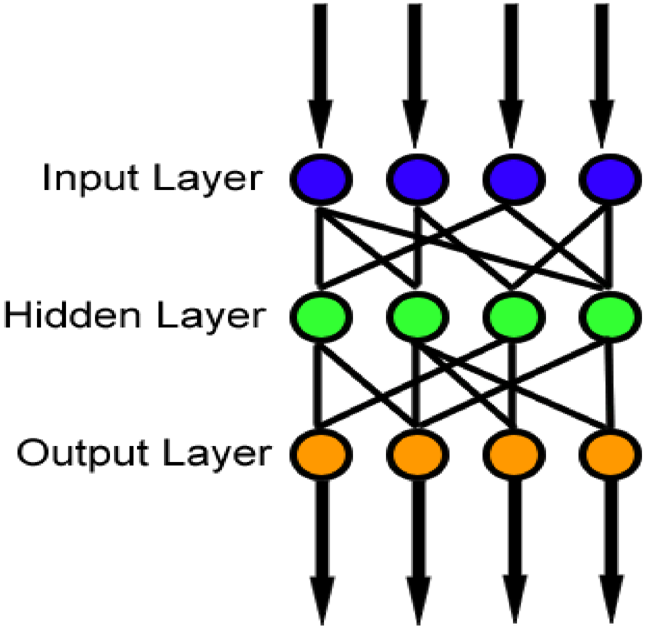
feedforwardnet





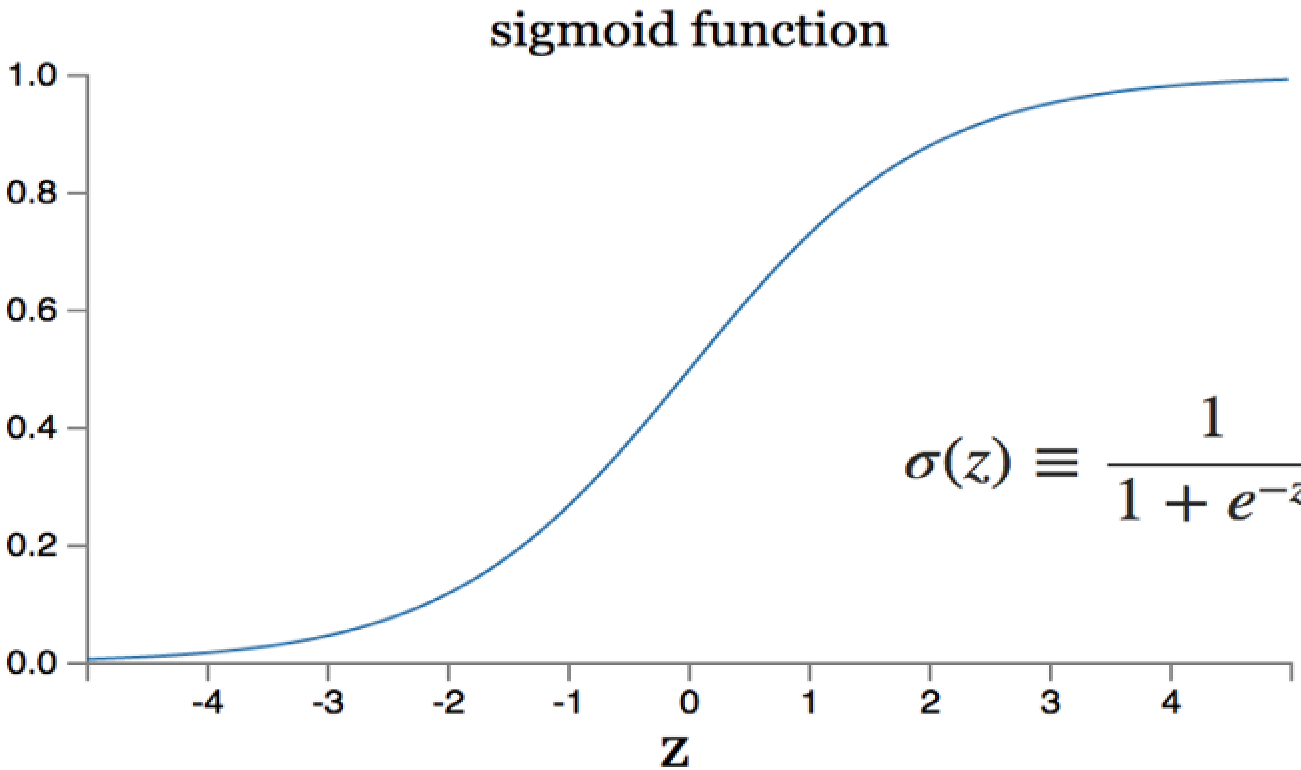
The feedforward neural network is the first and simplest type of artificial neural network.

In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network



$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

sigmoid function.



$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

feedforwardnet

```
feedforwardnet(hiddenSizes,trainFcn)
```

```
[x,t] = simplefit_dataset;
net = feedforwardnet(10);
net = train(net,x,t);
view(net)
y = net(x);
perf = perform(net,y,t)
```

Working With Iris Data

https://archive.ics.uci.edu/ml/datasets/iris

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Iris Data Set

Download: Data Folder, Data Set Description

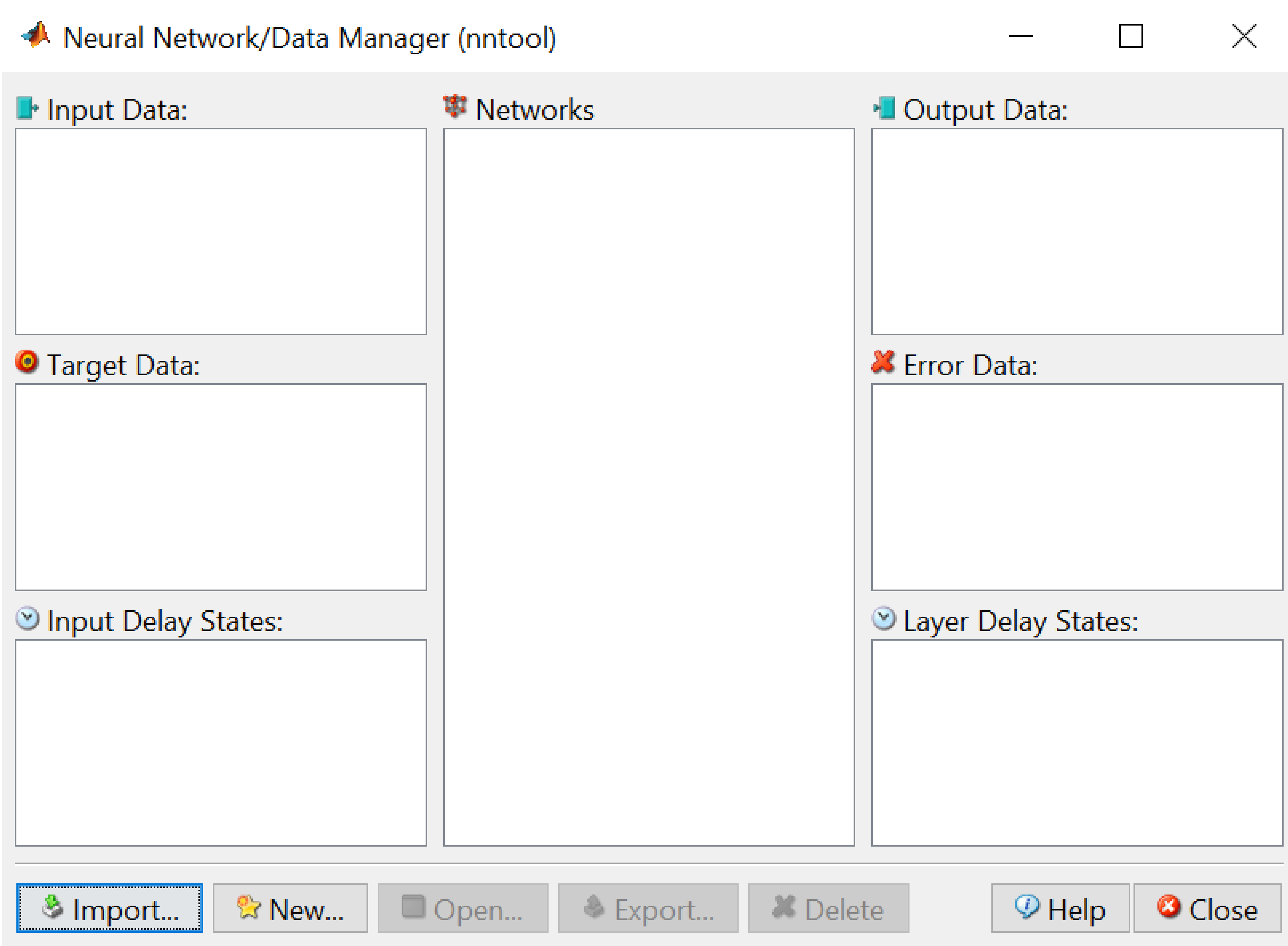
Abstract: Famous database; from Fisher, 1936

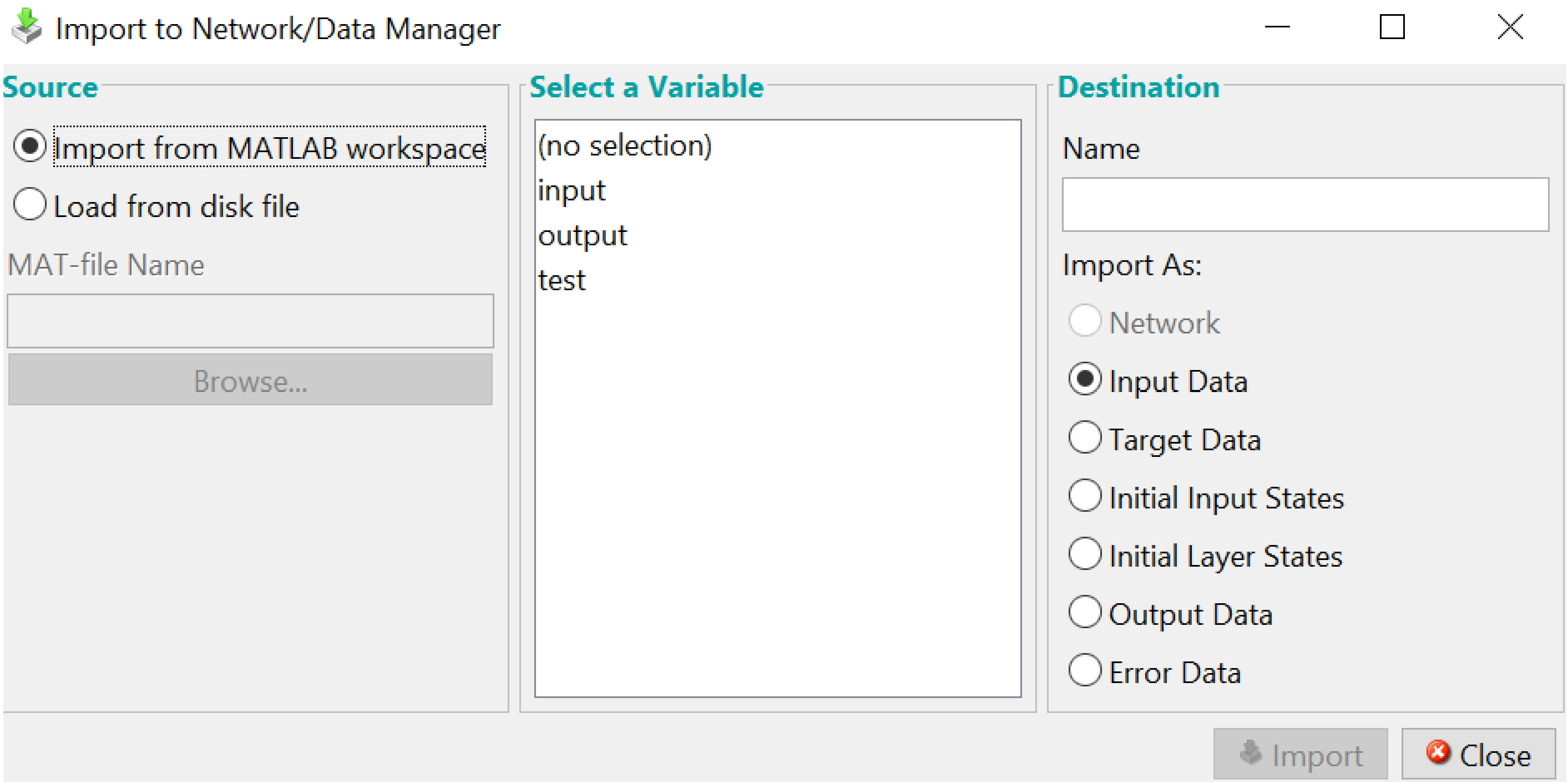
Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1817398

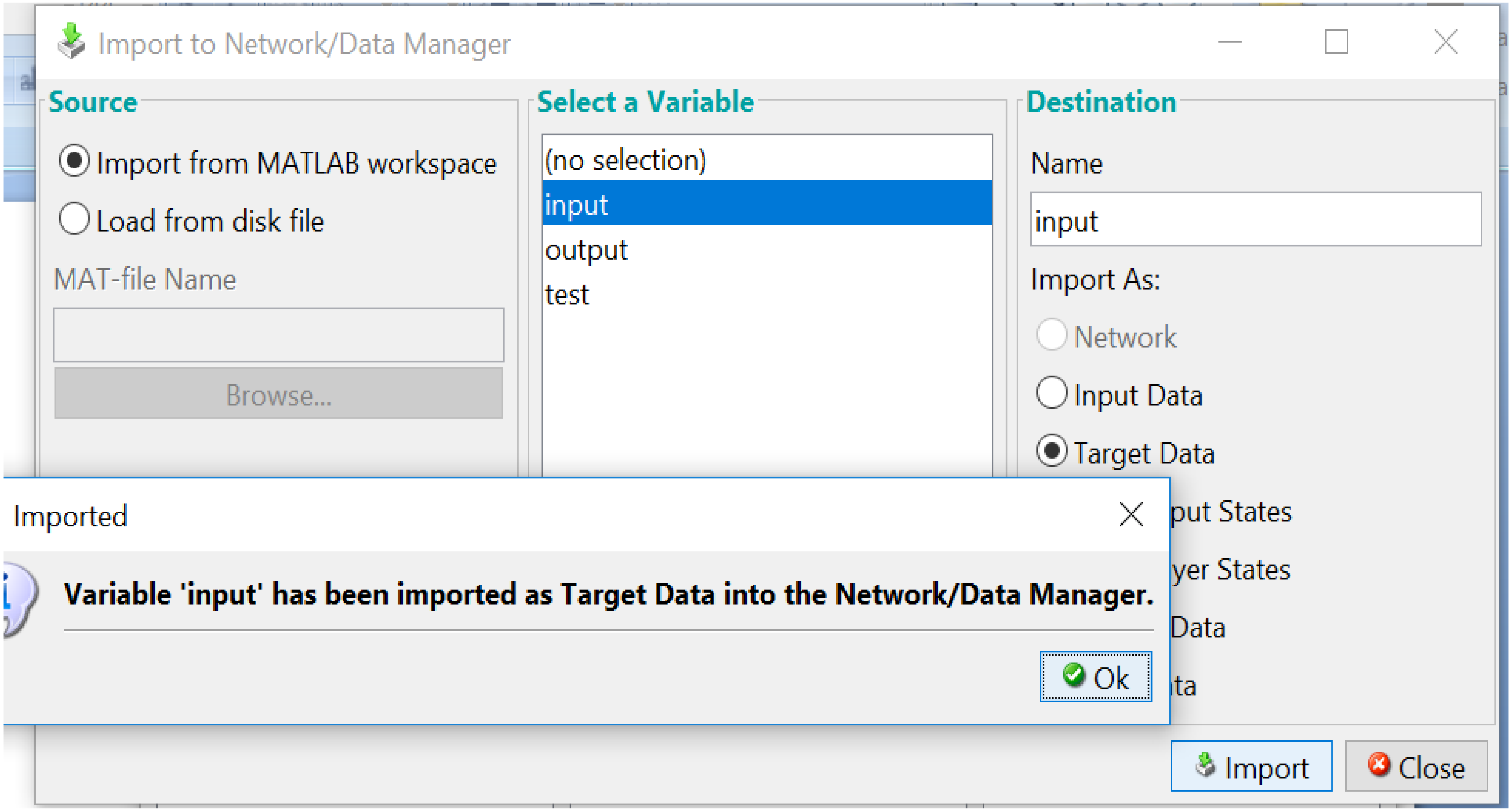
1	SEPAL LENGTH	SEPAL WIDTH	PETAL LENGTH	PETAL WIDTH	FLOWER
2	5.1	3.5	1.4	0.2	Iris-setosa
3	4.9	3	1.4	0.2	Iris-setosa
4	4.7	3.2	1.3	0.2	Iris-setosa
5	4.6	3.1	1.5	0.2	Iris-setosa
6	5	3.6	1.4	0.2	Iris-setosa
7	5.4	3.9	1.7	0.4	Iris-setosa
8	4.6	3.4	1.4	0.3	Iris-setosa
9	5	3.4	1.5	0.2	Iris-setosa
10	4.4	2.9	1.4	0.2	Iris-setosa
11	4.9	3.1	1.5	0.1	Iris-setosa
12	5.4	3.7	1.5	0.2	Iris-setosa
13	4.8	3.4	1.6	0.2	Iris-setosa
14	4.8	3	1.4	0.1	Iris-setosa
15	4.3	3	1.1	0.1	Iris-setosa
16	5.8	4	1.2	0.2	Iris-setosa
17	5.7	4.4	1.5	0.4	Iris-setosa
18	5.4	3.9	1.3	0.4	Iris-setosa
19	5.1	3.5	1.4	0.3	Iris-setosa

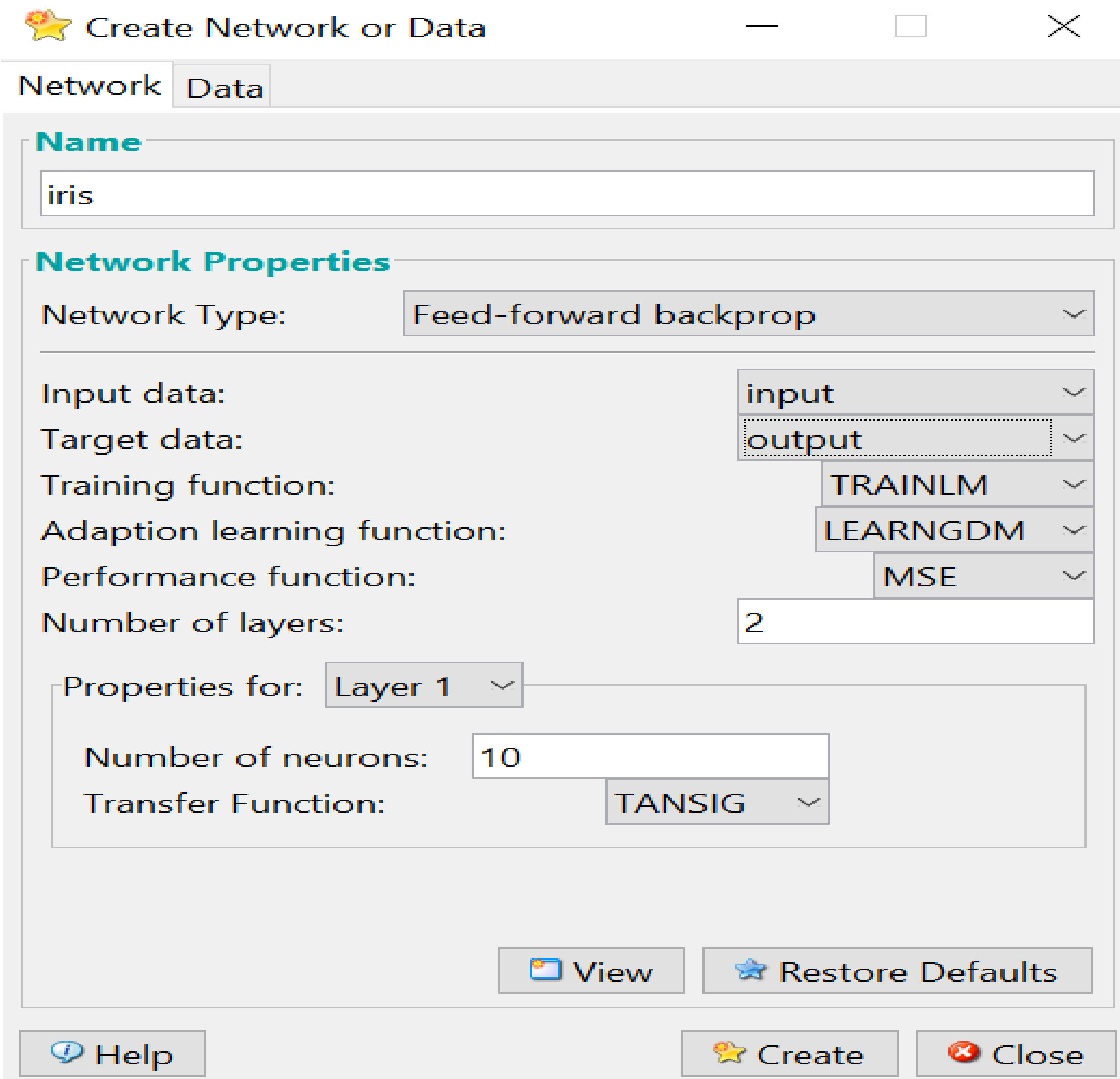
Data Collection

```
>> input=[]
>> output=[]
>> test=[]
>> input=input';
>> output=output';
>> test=test';
```

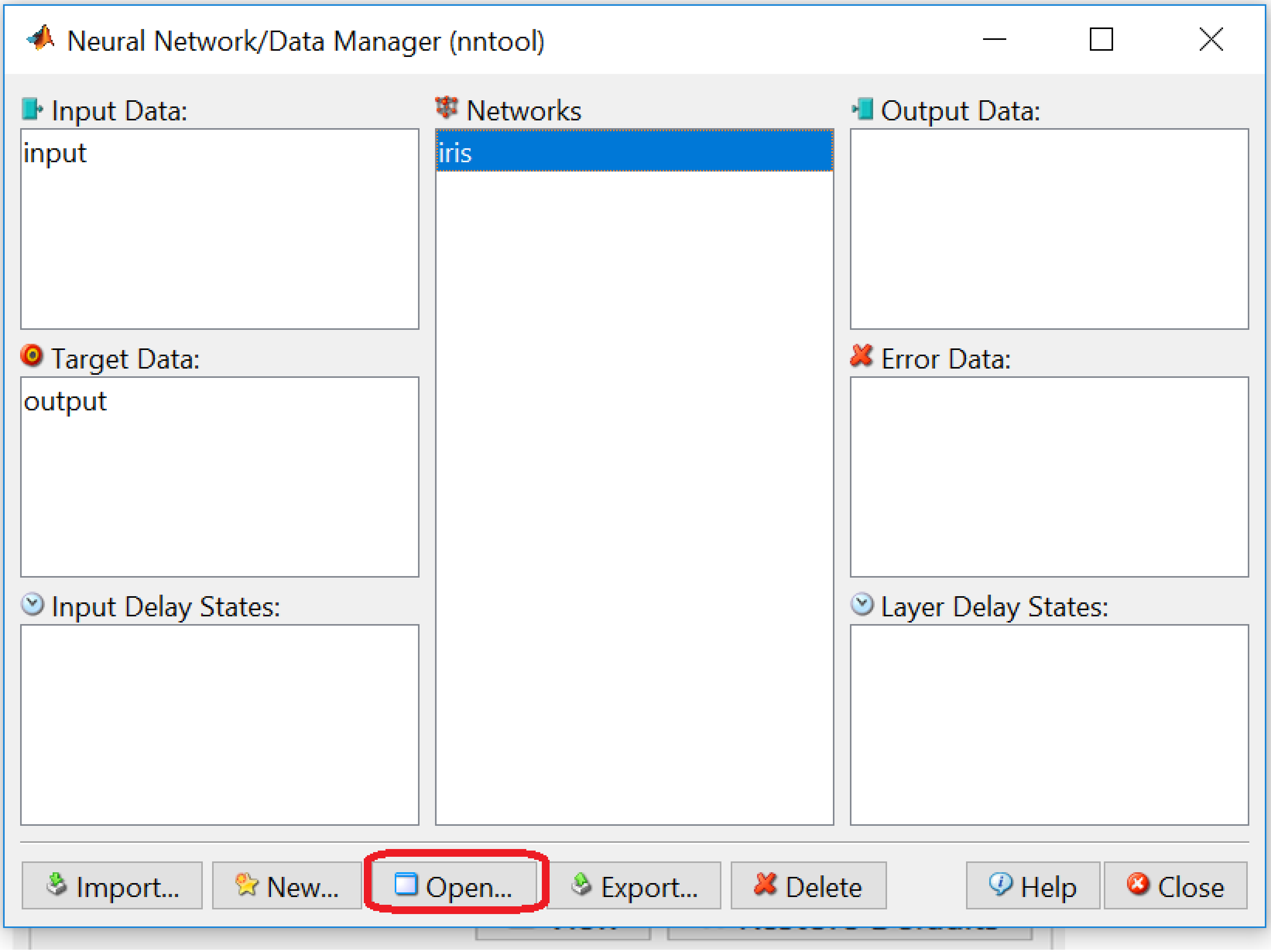

Neural Network/Data Manager (nntool)
Input Data:
Networks
Output Data:
Target Data:
Error Data:
Input Delay States:
Layer Delay States:
Import...
New...
Open...
Export...
Delete
Help
Close

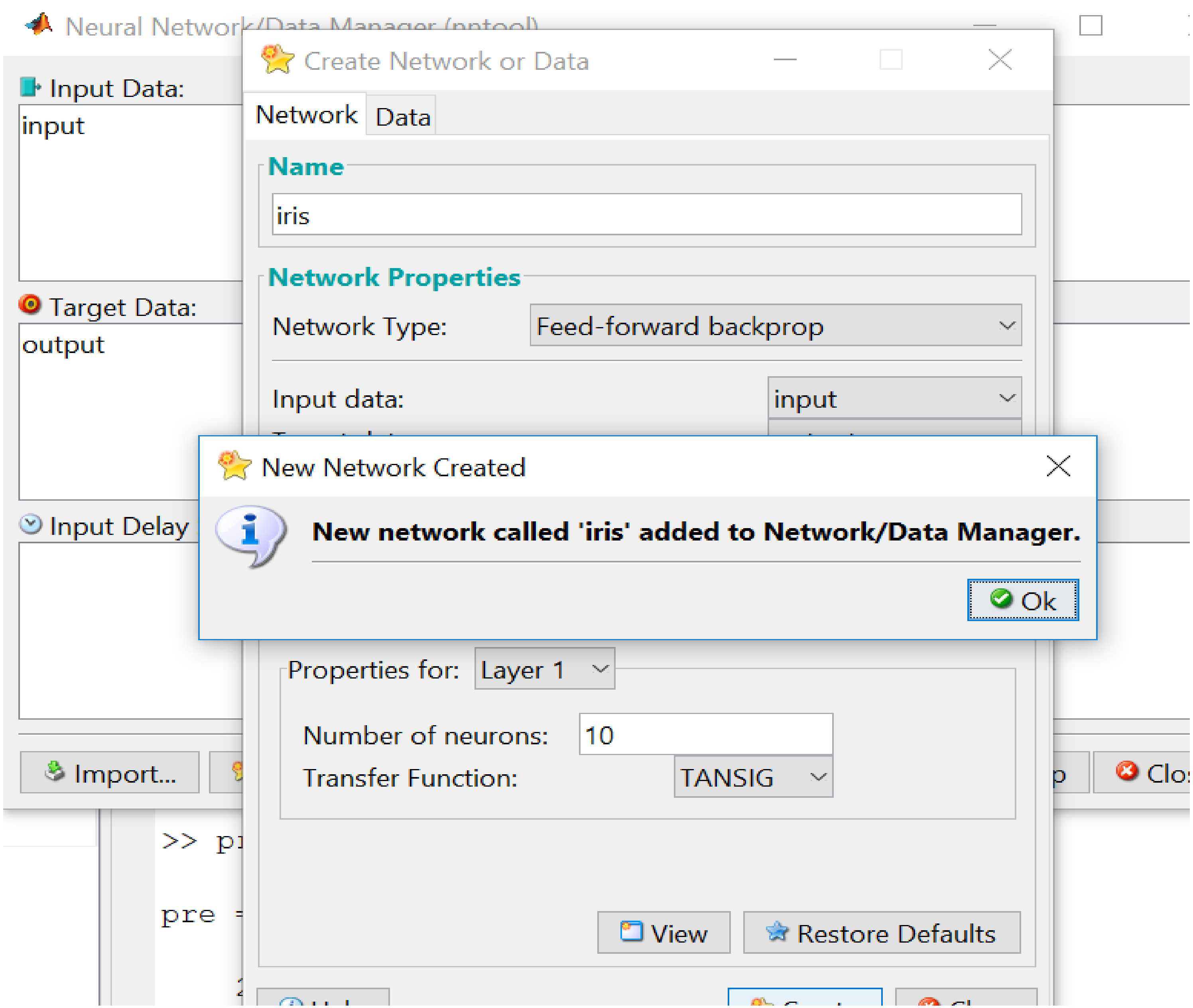

Import to Network/Data Manager
Source
Import from MATLAB workspace
Load from disk file
MAT-file Name
Browse...
Select a Variable
(no selection)
input
output
test
Destination
Name
Import As:
Network
Input Data
Target Data
Initial Input States
Initial Layer States
Output Data
Error Data
Import
Close


Import to Network/Data Manager
Source
Import from MATLAB workspace
Load from disk file
MAT-file Name
Browse...
Select a Variable
(no selection)
input
output
test
Destination
Name
input
Import As:
Network
Input Data
Target Data
Imported
Variable 'input' has been imported as Target Data into the Network/Data Manager.
Ok
Import
Close

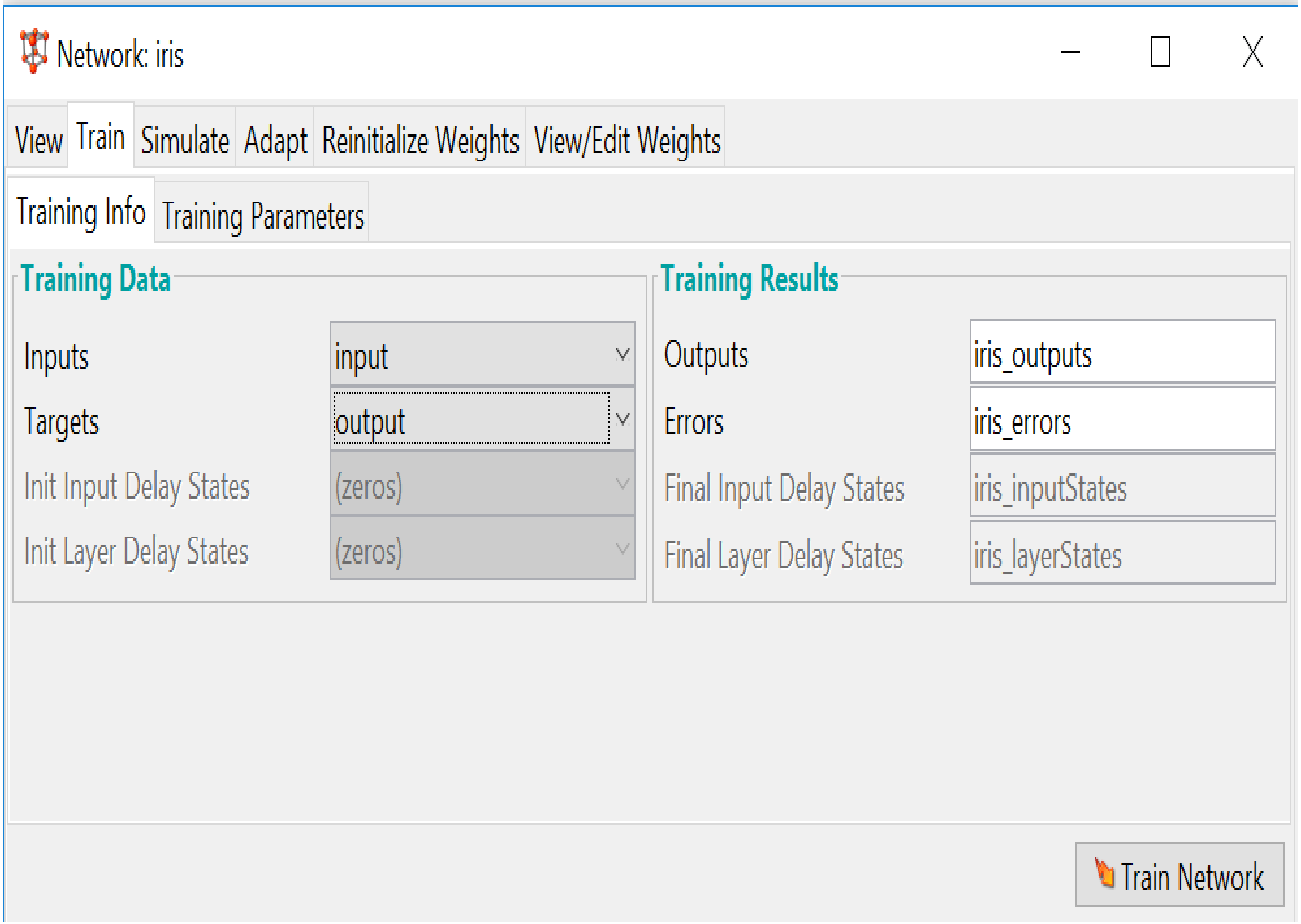


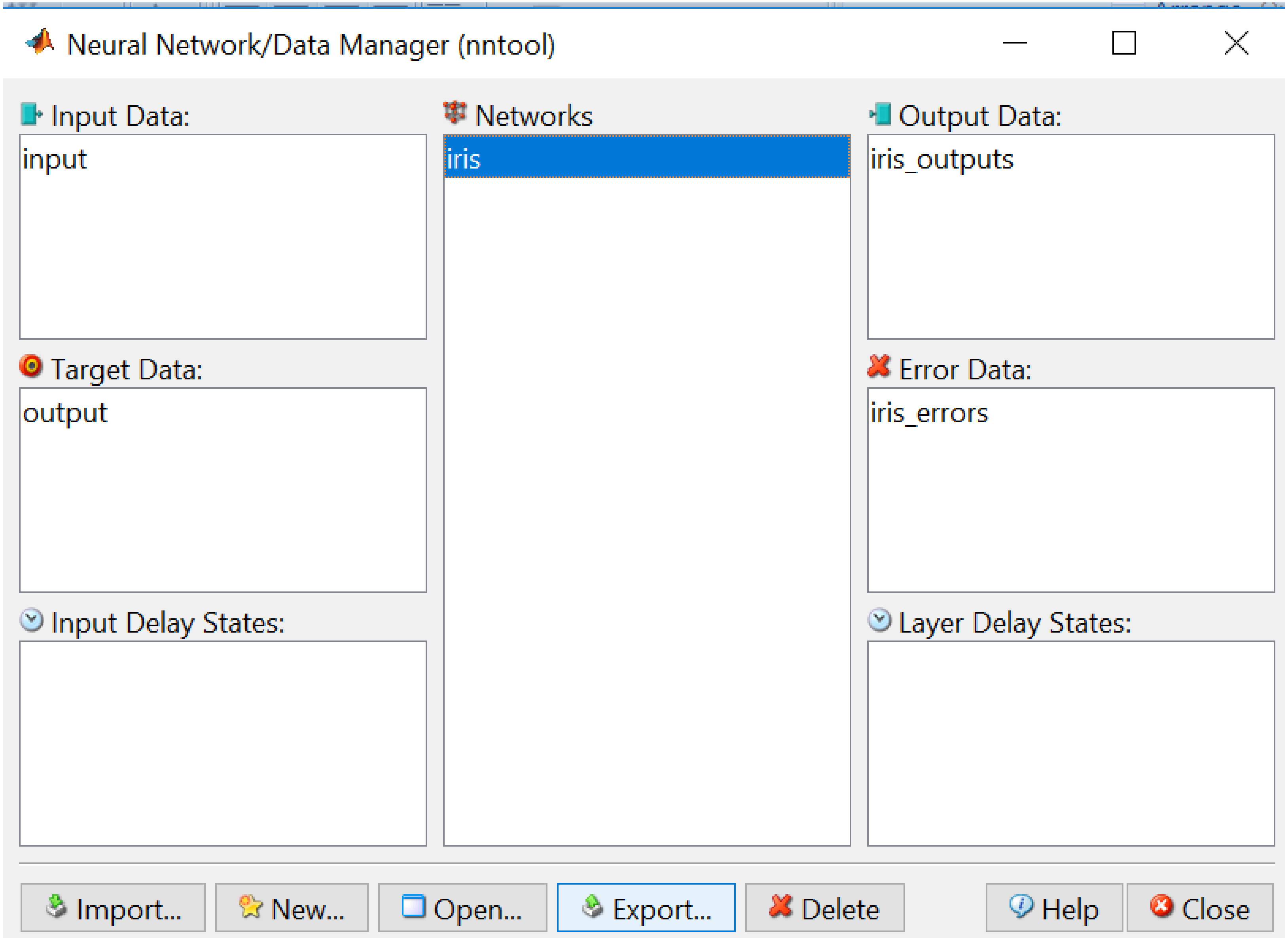
Create Network or Data
Network
Data
Name
iris
Network Properties
Network Type:
Feed-forward backprop
Input data:
input
Target data:
output
Training function:
TRAINLM
Adaption learning function:
LEARNGDM
Performance function:
MSE
Number of layers:
2
Properties for:
Layer 1
Number of neurons:
10
Transfer Function:
TANSIG
View
Restore Defaults
Help
Create
Close

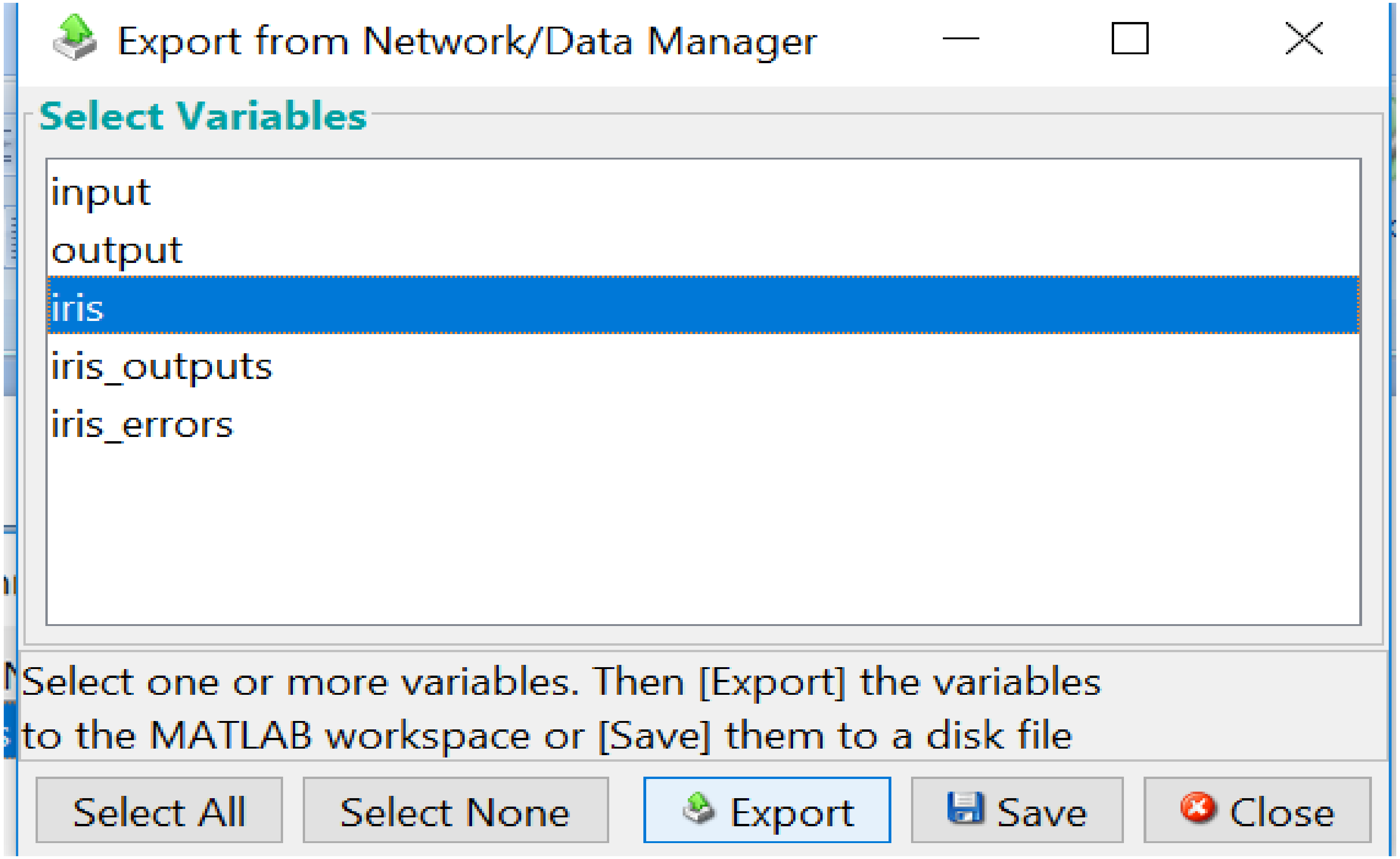

Neural Network/Data Manager (nntool)
Input Data:
input
Networks
iris
Output Data:
Target Data:
output
Error Data:
Input Delay States:
Layer Delay States:
Import...
New...
Open...
Export...
Delete
Help
Close



Neural Network/Data Manager (nntool)
Input Data:
input
Target Data:
output
Input Delay
Import...
Create Network or Data
Network
Data
Name
iris
Network Properties
Network Type:
Feed-forward backprop
Input data:
input
New Network Created
New network called 'iris' added to Network/Data Manager.
Ok
Properties for:
Layer 1
Number of neurons:
10
Transfer Function:
TANSIG
View
Restore Defaults
>> pr
pre =


Network: iris
View
Train
Simulate
Adapt
Reinitialize Weights
View/Edit Weights
Training Info
Training Parameters
Training Data
Inputs
input
Targets
output
Init Input Delay States
(zeros)
Init Layer Delay States
(zeros)
Training Results
Outputs
iris_outputs
Errors
iris_errors
Final Input Delay States
iris_inputStates
Final Layer Delay States
iris_layerStates
Train Network


Neural Network/Data Manager (nntool)
Input Data:
input
Networks
iris
Output Data:
iris_outputs
Target Data:
output
Error Data:
iris_errors
Input Delay States:
Layer Delay States:
Import...
New...
Open...
Export...
Delete
Help
Close




```
>> pre=sim(iris,test)
```