ENVIRONMENTAL SYSTEMS ANALYSIS

CENG 6652

Chapter 3

Soft computing techniques

Contents

1. Soft computing introduction
2. Data driven modelling
    1. Artificial neural networks
    2. Evolutionary algorithms: Genetic Algorithm
3. Qualitative modelling
    1. Fuzzy Optimization

Computing approaches

- White – Box Models: where the interactions and processes that take place among the various components of the system can scientifically/mathematically be proven/explained.
- Note these mechanistically or process-based models usually contain parameters whose values are determined from observed data during model calibration.
- Black – box models, or statistical models: Such models do not describe physical processes. They attempt to convert observed inputs to observed outputs. They do not really care about the scientific reasoning behind the input-output system.
- Grey – box or hybrid models: models that combine the above two models (in parallel or in series).

1. Soft-computing approaches

- The use of inexact solutions to computationally hard tasks, for which there is no known algorithm that can compute an exact solution in polynomial time is soft computing.
- This chapter introduces some alternative modeling approaches that depend on observed data. These approaches include artificial neural networks and evolutionary model. The chapter ends with some qualitative modeling.
- The data-driven models can serve as substitutes for more process-based models in applications where:
  - Computational speed is critical and/or
  - Where the underlying relationships are poorly understood or too complex to be easily incorporated into calculus-based, linear, nonlinear, or dynamic programming models.

Soft-computing approaches

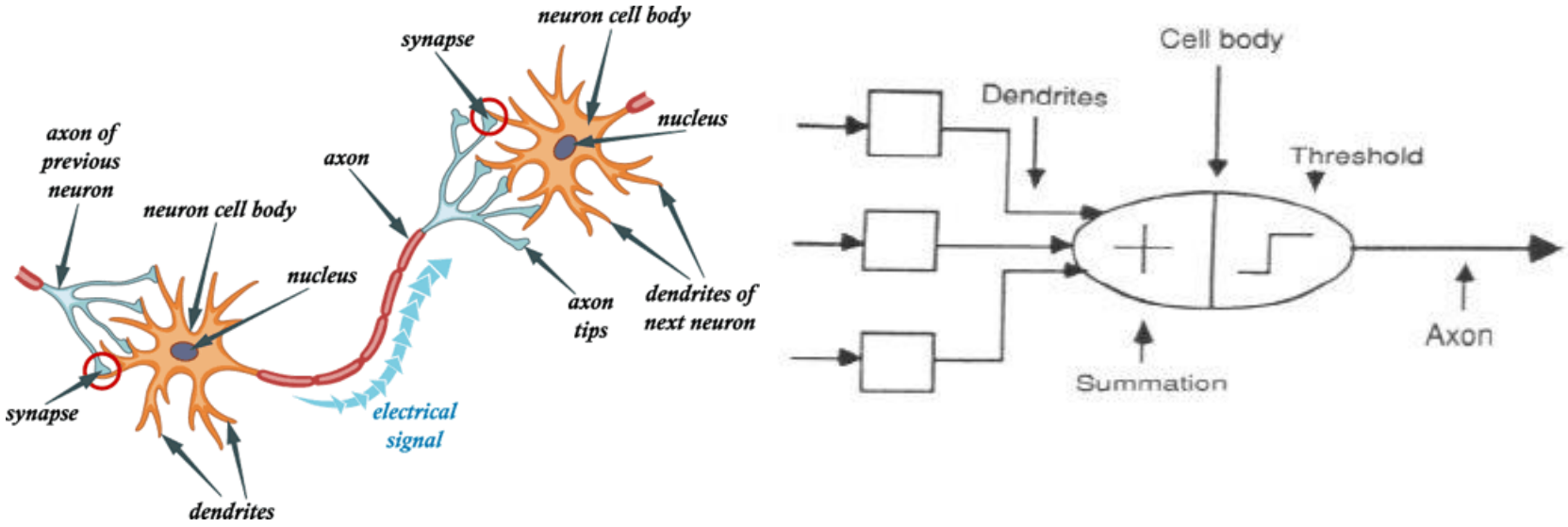
- Evolutionary algorithms involve **random searches** based on evolutionary processes for finding the values of parameters and decision variables that best satisfy system performance criteria.
- Evolutionary algorithms are popular methods for analyzing systems that require complex simulation models to determine values of performance measures.
- Qualitative modeling approaches are useful when performance measures are expressed qualitatively, such as “I want a reliable supply of clean water at a reasonable cost,” where there can be disagreements among different stakeholders and decision makers with respect to specifying just how reliable, how clean, and how affordable.

2. Data driven Models (DDM)

- They depend on observed inputs and observed outputs for the estimation of the values of their parameters and for further refinement of their mathematical structure.
- They lack an explicit, well-defined representation of the processes involved in the transformation of inputs to outputs
- Examples of Black-box models DDMs are: regression analysis, Artificial Neural network, ANFIS etc.
- Other examples of data-driven models are based on Darwinian evolutionary concepts. These are a class of probabilistic search procedures known as evolutionary algorithms (EAs). Such algorithms include genetic algorithms (GAs), genetic or evolutionary programming (GP or EP), and evolutionary strategy (ES).

2.1 Artificial Neural Network (ANN)

- Some computer scientists have been working on creating information processing devices that mimic the human brain. This has been termed neurocomputing.
- ANNs represent simplified models of the brain. In reality, it is just a more complex type of regression or statistical (black-box) model.



Terminology

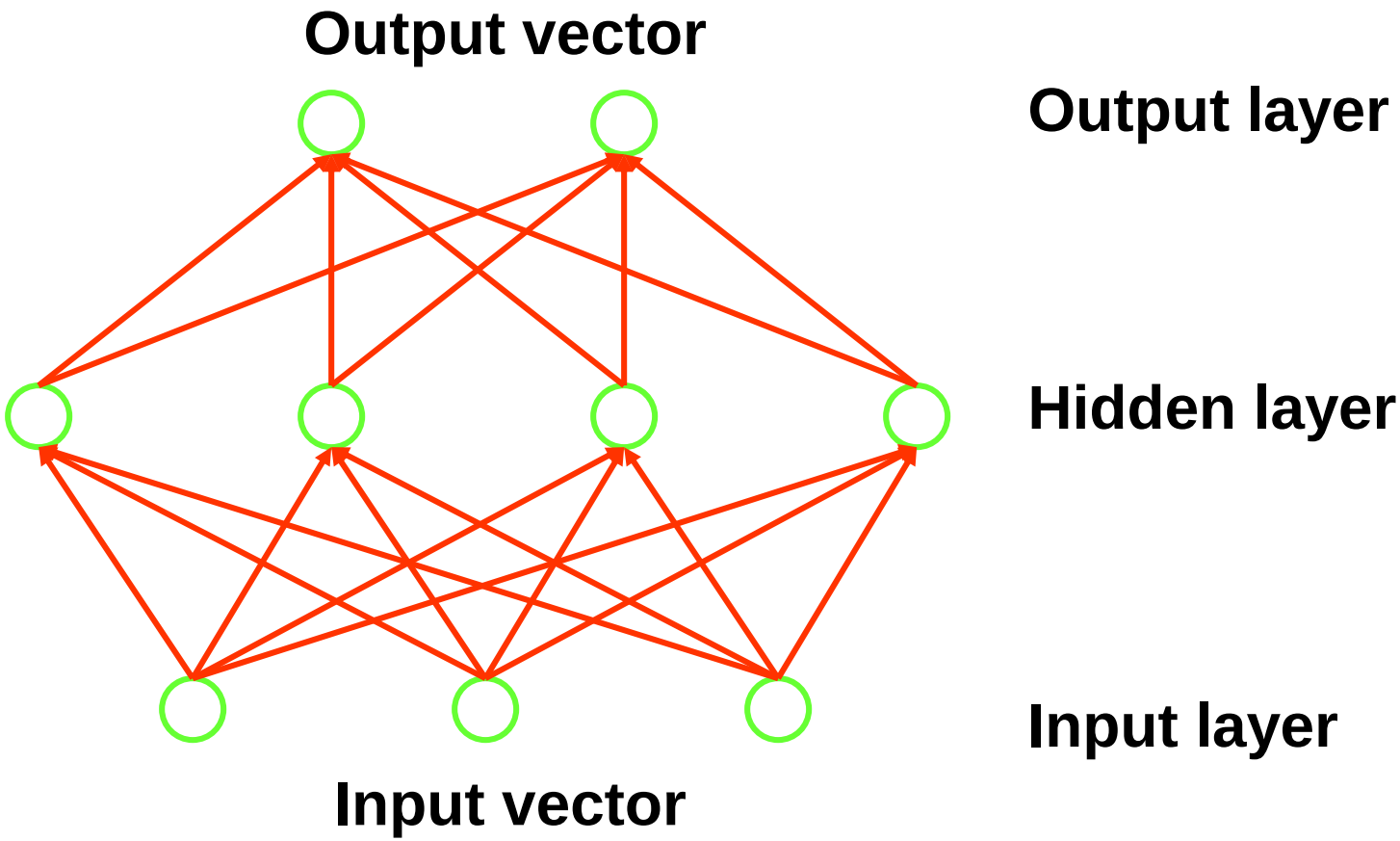
- The “building blocks” of neural networks are the neurons. In ANN, these are referred as units or nodes.
- Basically, each neuron
  - Receives input from many other neurons (through dendrites),
  - Changes its internal state (activation) based on the current input,
  - Sends one output signal to many other neurons, possibly including its input neurons (recurrent network)
- Information is transmitted as a series of electric impulses, so-called spikes.
- The frequency and phase of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as 10,000 other neurons.

Terminology

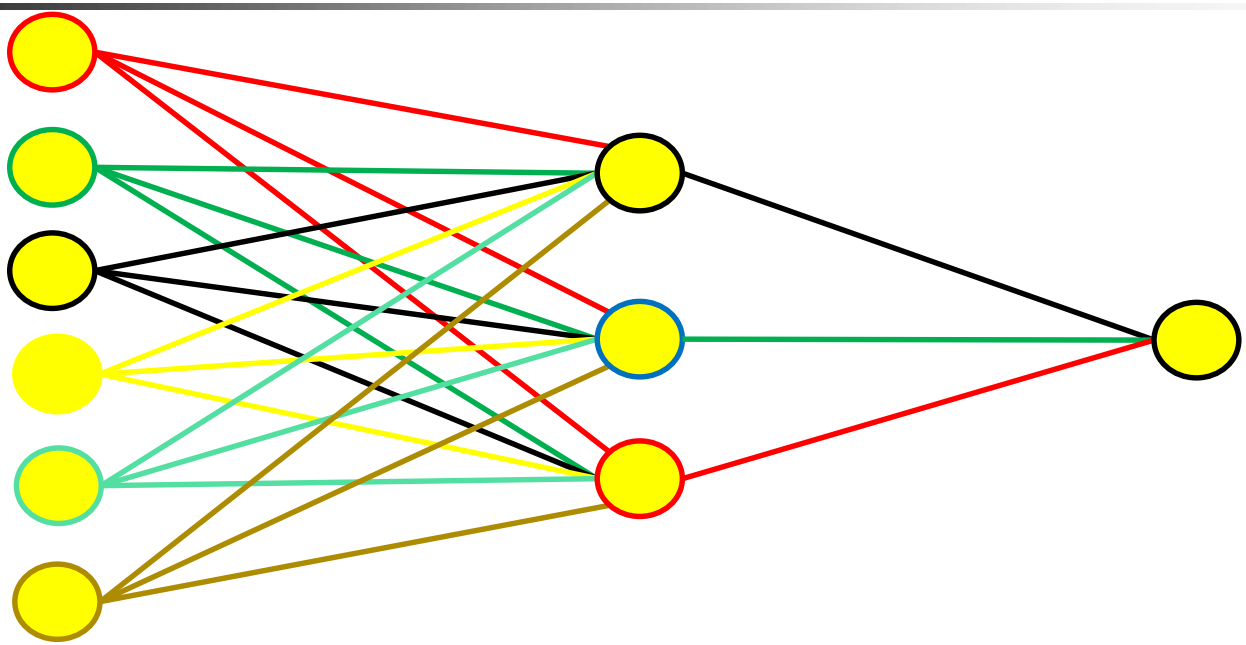
- Usually, we draw neural networks in such a way that the input enters at the bottom/left and the output is generated at the top/right.
- Arrows indicate the direction of data flow.
- The first layer, termed input layer, just contains the input vector and does not perform any computations.
- The second layer, termed hidden layer, receives input from the input layer and sends its output to the next hidden layer or the output layer.
- After applying their activation function, the neurons in the output layer contain the output vector.

Terminology

- Example: Network function



Basic structure of an ANN



Input layer **Hidden layer** **Output layer**

- The number of hidden layers and the number of nodes in each layer are two of the design parameters of any ANN.
- Most applications require networks that contain at least these three types of layers:

Basic structure of ANN

- The input layer consists of nodes that receive an input from the external environment. These nodes do not perform transformations upon the inputs but just send their weighted values to the nodes in the immediately adjacent, usually “hidden,” layer.
- The hidden layer(s) consist(s) of node(s) that typically receive the transferred weighted inputs from the input layer or previous hidden layer, perform their transformations on it, and pass the output to the next adjacent layer, which can be another hidden layer or the output layer.
- The output layer consists of nodes that receive the hidden layer output and send it to the user.
- The ANN shown has links only between nodes in immediately adjacent layers or columns often referred to as a multilayer perceptron (MLP) network, or a feedforward (FF) network.

Basic Structure of ANN

- The number of nodes in the input and output layers are usually predetermined from the problem to be solved.
  - Input nodes = No. of input variables + 1 (Bias Node)
  - Output layer node = No. of output required
- The number of nodes in each hidden layer and the number of hidden layers are calibration parameters.
- The values of the weights and thresholds of each connection, are “learned” during the “training” of the ANN using predefined (or measured) sets of input and output data.
- Determining the best values of all the weights is called training the ANN.
- In supervised learning: the actual output of a neural network is compared to the desired output.

ANN Topologies

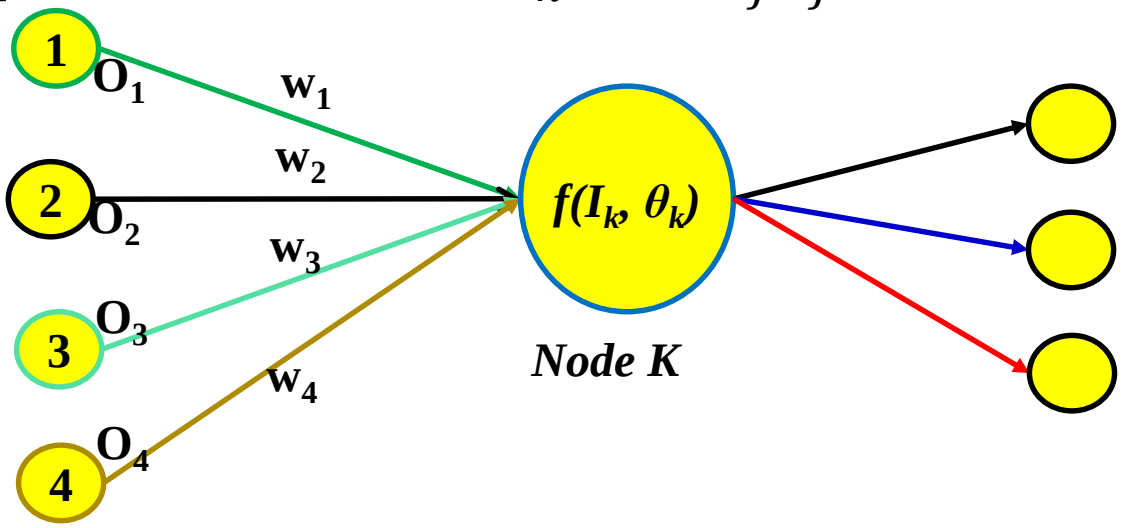
- There are two major connection topologies that define how data flows between the input, hidden, and output nodes.
- ***Feedforward networks:*** in which the data flow in one direction from the input layer to the output layer through the hidden layer(s).
- Here the nodes of one layer are fully connected to the nodes in the next layer; however, this is not a requirement.
- ***Recurrent or feedback networks***: in which, as their name suggests, the data flow not only in one direction but in the opposite direction as well for either a limited or a complete part of the network.
- The recurrent types of artificial neural networks are used when the answer is based on current data as well as on prior inputs.

How do NNs and ANNs work?

- Neurons of similar functionality are usually organized in layers.
- Often, there is a hierarchy of interconnected layers with the lowest layer receiving sensory input and neurons in higher layers computing more complex functions.
- NNs are able to learn by adapting their connectivity patterns so that the organism improves its behavior in terms of reaching certain (evolutionary) goals.
- The strength of a connection, or whether it is excitatory or inhibitory, depends on the state of a receiving neuron's synapses.
- The NN achieves learning by appropriately adapting the states of its synapses.

Artificial Neuron

- Essentially, the strength (or weight) of the connection between adjacent nodes is a design parameter of the ANN.
- The output values O_j that leave a node j on each of its outgoing links are multiplied by a weight, w_j.
- The input I_k to each node k in each middle and output layer is the sum of each of its weighted inputs, $w_j O_j$, from all nodes j providing inputs (linked) to node k.
  - Input value to node k: $I_k = \sum w_j O_j$



Artificial neuron

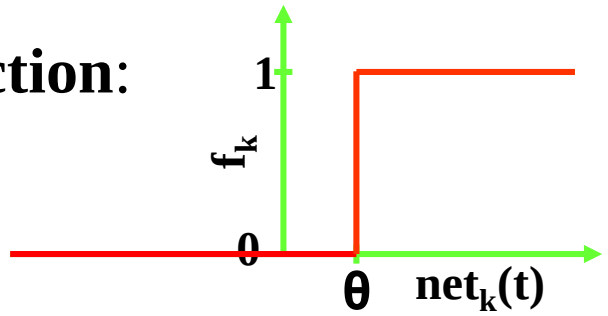
- At each node K of hidden and output layers, the input I_k is an argument to a linear or nonlinear function $f_k(I_k + \theta_k)$, which converts the input I_k to output O_k.
- The variable θ_k represents a bias or threshold term that influences the horizontal offset of the function.
- This transformation can take on a variety of forms. A commonly used transformation is a sigmoid or logistic function as defined:

$$O_k = \frac{1}{1+exp\{-(I_k+\theta_k)\}}$$

- The same process also happens at each output layer node.

The net input signal

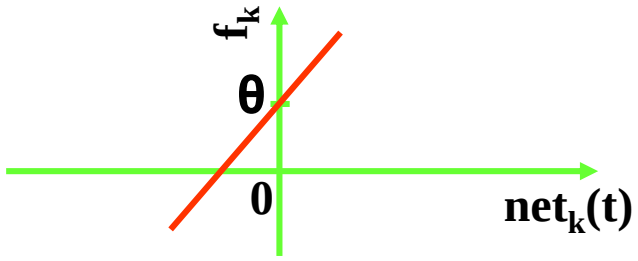
- The net input signal is the sum of all n-inputs after passing the synapses: $\text{net}_k(t) = I_k = \sum_{j=1}^{n} w_{kj} O_j(t)$
- In most ANNs, the activation of a neuron is simply defined to equal its net input signal: $a_k(t) = \text{net}_k(i)$
- Then, the neuron's activation function (or output function) f_i is applied directly to $net_k(t)$: $O_k(t) = f_k\big(net_k(t)\big)$
- What do such functions f_i look like?
- One possible choice is a **threshold function**:
  - $f_k\big(net_k(t)\big) = 1, \quad \text{if } net_k(t) \geq \theta$
  - $f_k\big(net_k(t)\big) = 0, \quad otherwise$



- Obviously, the fact that threshold units can only output the values 0 and 1 restricts their applicability to certain problems.

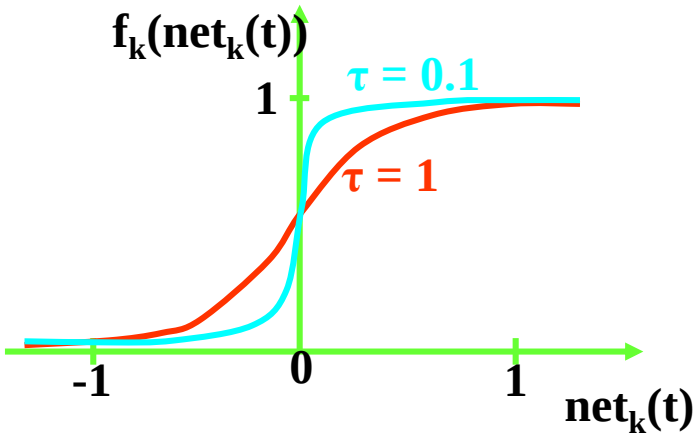
Linear Neurons

- We can overcome this limitation by eliminating the threshold and simply turning f_k into the identity function so that we get:
  - $O_k(t) = f_k(t) = a \times net_k(t) + \theta$
- With this neuron, we can build networks with m input and n output neurons that compute a function f: $R^m \rightarrow R^n$.
- Limitation: Each neuron computes a linear function, and therefore the overall network function f: $R^m \rightarrow R^n$ is also linear.
- This means that if an input vector *x* results in an output vector *y*, then for any factor ϕ the input $\phi \cdot x$ will result in the output $\phi \cdot y$.
- Obviously, many interesting functions cannot be realized by networks of linear neurons.



Sigmoidal (Logistic) Neurons

- Sigmoidal neurons accept any vectors of real numbers as input, and they output a real number between 0 and 1.
- Sigmoidal neurons are the most common type of artificial neuron, especially in learning networks.
- A network of sigmoidal units with *m* input and *n* output neurons realizes a network function f: $R^m \rightarrow (0,1)^n$
- $f_k(\text{net}_k(t)) = \frac{1}{1+e^{\left(-(\text{net}_k(t)+\theta)/\tau\right)}}$
- The parameter τ controls the slope of the sigmoid function, while the parameter θ controls the horizontal offset of the function in a way similar to the threshold neurons.

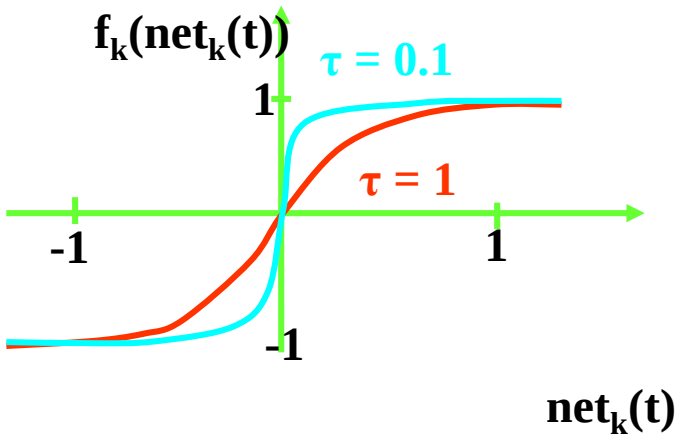


Tanh (hyperbolic tangent) Neurons

- Log-Sigmoidal neurons accept any vectors of real numbers as input, and they output a real number between -1 and 1.
- A network of Log-sigmoidal units with *m* input and *n* output neurons realizes a network function f: $R^m \rightarrow (-1,1)^n$

$$f_k(\text{net}_k(t)) = Tanh(\text{net}_k(t)) = \frac{2}{1+e^{-2(\text{net}_k(t)+\theta)/\tau}} - 1$$

- The parameter τ controls the slope of the tan hyperbolic function, while the parameter θ controls the horizontal offset of the function in a way similar to the threshold neurons.

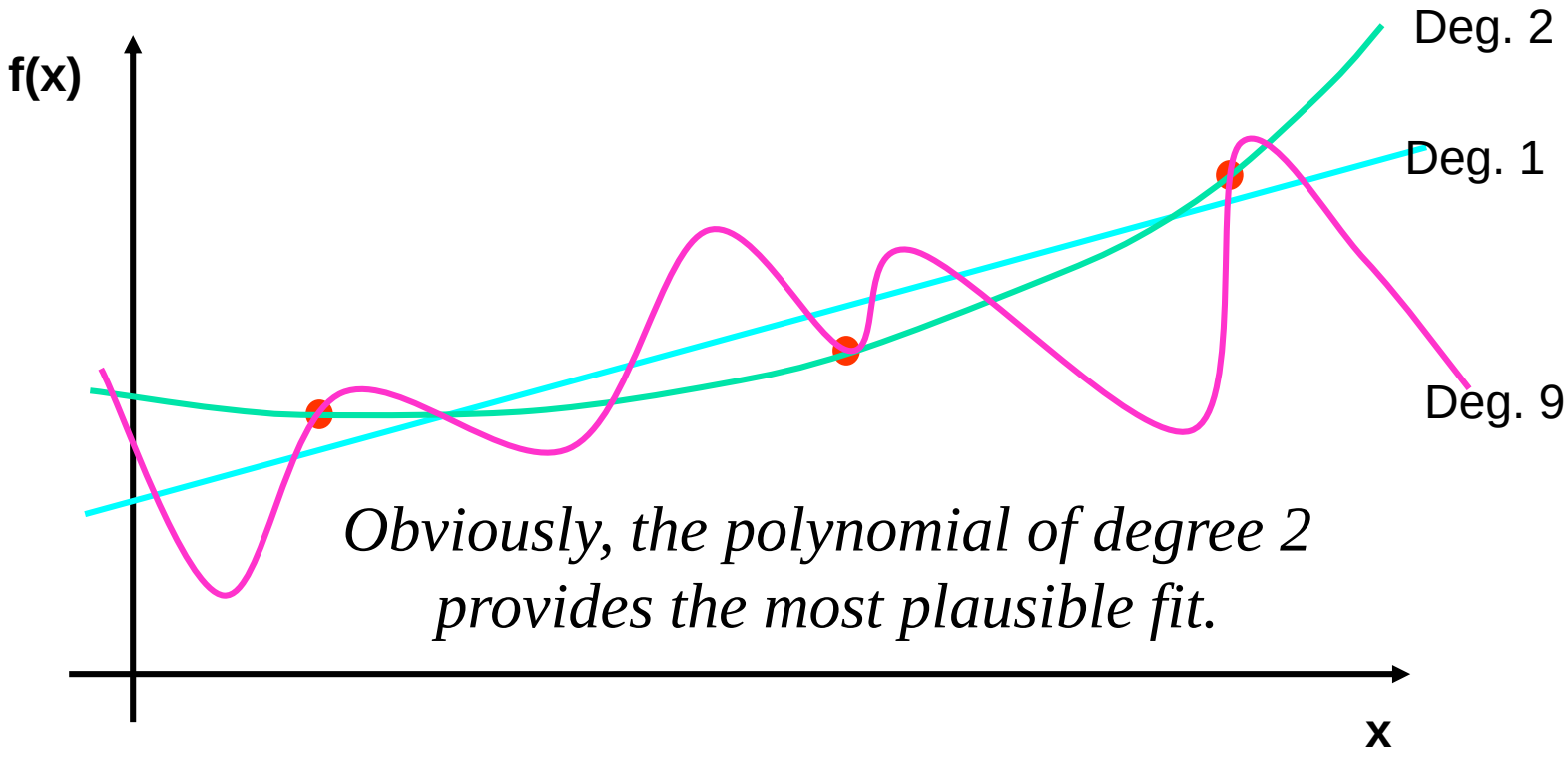


Learning in ANNs

- In supervised learning, we train an ANN with a set of vector pairs, so-called exemplars. Each pair (x, y) consists of an input vector x and a corresponding output vector y.
- Whenever the network receives input x, we would like it to provide output y. The exemplars thus describe the function that we want to "teach" our network.
- Besides learning the exemplars, we would like our network to generalize, that is, give plausible output for inputs that the network had not been trained with.
- There is a tradeoff between a network's ability to precisely learn the given exemplars and its ability to generalize (i.e., interpolate and extrapolate).
- This problem is similar to fitting a function to a given set of data points.

Learning in ANNs

- Let us assume that you want to find a fitting function f:R→R for a set of three data points.
- You try to do this with polynomials of degree one (a straight line), two, and nine. Which one is best?



Obviously, the polynomial of degree 2 provides the most plausible fit.

Learning in ANNs

- The same principle applies to ANNs:
  - If an ANN has too few neurons, it may not have enough degrees of freedom to precisely approximate the desired function.
  - If an ANN has too many neurons, it will learn the exemplars perfectly, but its additional degrees of freedom may cause it to show implausible behavior for untrained inputs; it then presents poor ability of generalization.
- Unfortunately, there are no known equations that could tell you the optimal size of your network for a given application; you always have to experiment.

Training ANN

- Weights, which are usually randomly set to begin with, are then adjusted so that the next iteration will produce a closer match between the desired and the actual output.
- Various learning methods for weight adjustments try to minimize the differences or errors between observed and computed output data.
- It is considered complete when the ANN reaches a user-defined performance level. After which the resulting weights are typically fixed for the application.
- Once a supervised network performs well on the training data, it is important to test it (what it can do with data it has not seen before).
- If the ANN did not give a reasonable output for this test set, the training period should continue.

Backpropagation Preparation

- **Training Set:** A collection of input-output patterns that are used to train the network
- **Testing Set:** A collection of input-output patterns that are used to assess network performance
- **Learning Rate-η:** A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments
- **Network Error**
  - Total-Sum-Squared-Error (TSSE)
    - $TSSE = \frac{1}{2}\sum_{patterns}\sum_{outputs}(Desired - Actual)^2$
  - Root-Mean-Squared-Error (RMSE)
    - $RMSE = \sqrt{\frac{2*TSSE}{\#patterns*\#outputs}}$

Training ANN: the backpropagation Algorithm

- Error is the mean square of differences in output layer
- $E(x) = \frac{1}{2}\sum_{k=1}^{K}(T_k(x) - O_k(x))^2$, where O – computed output and T – target output, K number of training data pairs.
- Error of training epoch is the average of all errors.
- Update weights and thresholds using
  - Weights: $w_{jk} = w_{jk} + (-\eta)\frac{\partial E(x)}{\partial w_{jk}}$
  - Bias: $\theta_k = \theta_k + (-\eta)\frac{\partial E(x)}{\partial \theta_k}$
  - η is a possibly time-dependent factor that should prevent overcorrection called **learning rate**

A Pseudo-Code BP Algorithm

- Randomly choose the initial weights
- While error is too large
  - For each training pattern (presented in random order)
    - Apply the inputs to the network
    - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
    - Calculate the error at the outputs
    - Use the output error to compute error signals for pre-output layers
    - Use the error signals to compute weight adjustments
    - Apply the weight adjustments
  - Periodically evaluate the network performance

Steps in ANN development

1. Design a network.
2. Divide the data set into training, validation and testing subsets.
3. Train the network on the training data set.
4. Periodically stop the training and measure the error on the validation data set.
5. Save the weights of the network.
6. Repeat Steps 2, 3, and 4 until the error on the validation data set starts increasing. This is the moment where the overfitting has started.
7. Go back to the weights that produced the lowest error on the validation data set, and use these weights for the trained ANN.
8. Test the trained ANN using the testing data set. If it shows good performance, use it. If not, redesign the network and repeat entire procedure from Step 3.

Example

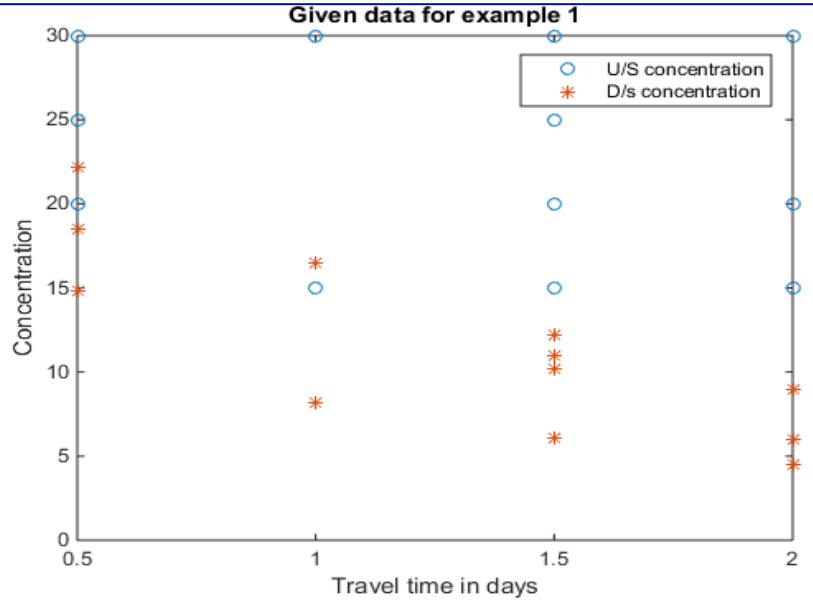
- Consider predicting a downstream pollutant concentration based on an upstream concentration and the streamflow.
- Twelve measurements of the streamflow quantity, velocity, and pollutant concentrations at two sites are available. The travel times between the two measurement sites (an upstream and a downstream site) have been computed and these, plus the pollutant concentrations, are shown in Table below.

Travel Time (Days)	2	2	1.5	1.5	0.5	1	0.5	1.5	1.5	2	1	0.5
Concentration U/S	20	15	30	20	20	15	30	25	15	30	30	25
Concentration D/S	6	4.5	12.2	11	14.8	8.2	22.2	10.2	6.1	9	16.5	18.5

Plotting the data

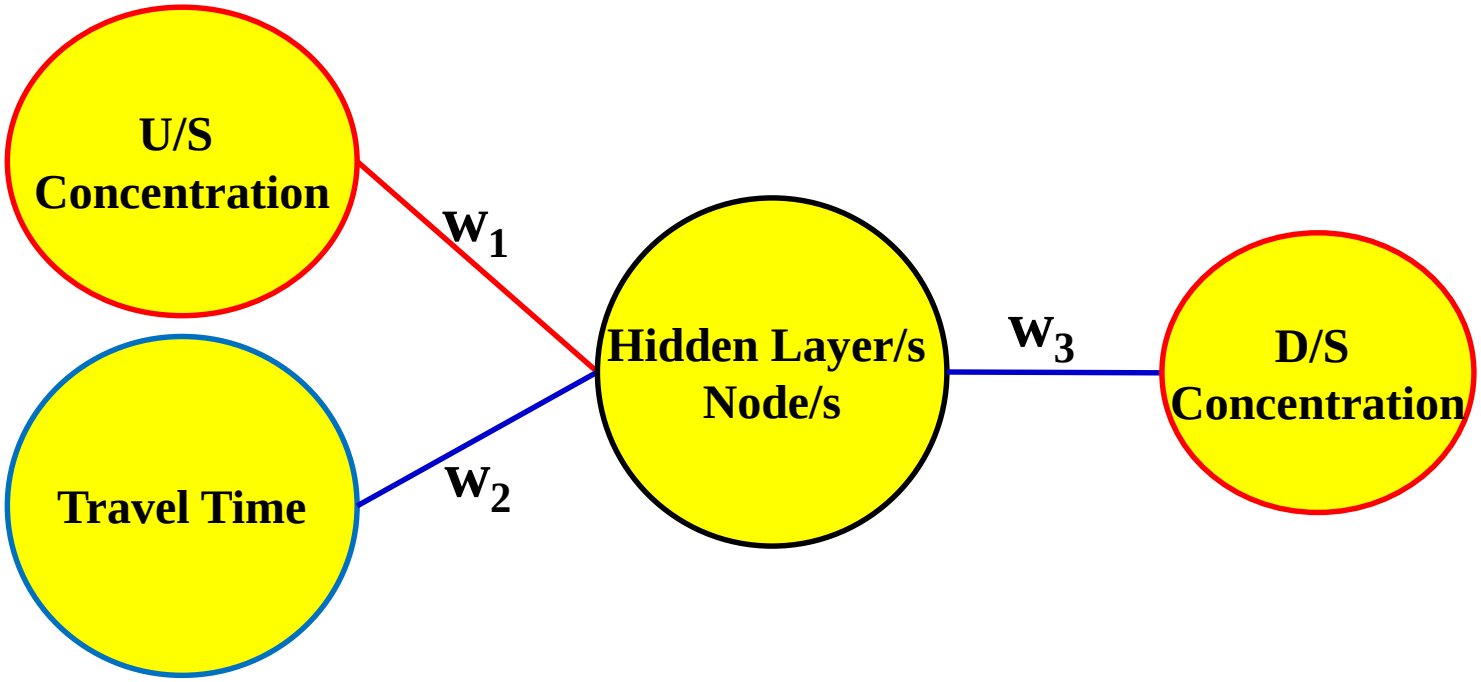
```
%load the input data
inputdata=load('input1.dat');     %data available arranged with inputs in the
prior columns and the last column will represent the output
plot(inputdata(:,1),inputdata(:,2),'o',inputdata(:,1),inputdata(:,3),'*');
title('Given data for example 1');xlabel('Travel time in days');
ylabel('Concentration'); legend('U/S concentration','D/s concentration');
```

```
[2.0   20.0   6.0;
 2.0   15.0   4.5;
 1.5   30.0   12.2;
 1.5   20.0   11.0;
 0.5   20.0   14.8;
 1.0   15.0   8.2;
 0.5   30.0   22.2;
 1.5   25.0   10.2;
 1.5   15.0   6.1;
 2.0   30.0   9.0;
 1.0   30.0   16.5;
 0.5   25.0   18.5]
```




```
[ndata, nc]=size(inputdata); % ndata is the number of data and nc is the
number of column provided in the data
ni=2; %ni is the number of input variables
no=nc-ni;   %no is the number of output expected from the model
inputs1=inputdata(:,1)';  inputs2=inputdata(:,2)';   inputs=[inputs1;inputs2];
outputs=inputdata(:,nc)';
```

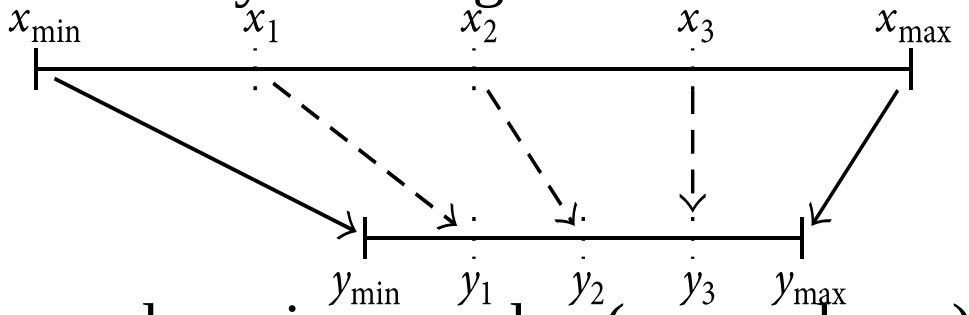
Design of the network



- The two inputs and the output do have different units which results in different magnitude. Besides the activation function to be used in the hidden layer node (here Logistic sigmoid will result in values between zero and one. Thus ANN requires us to bring every input and output with equivalent input magnitude and outputs within the activation range of output values.

Normalization

- Two types of normalization methods, min.-max. method and normal distribution method
- The min.-max. method linearly maps the original values to the new interval determined by the assigned min.-max. values (see Figure below).

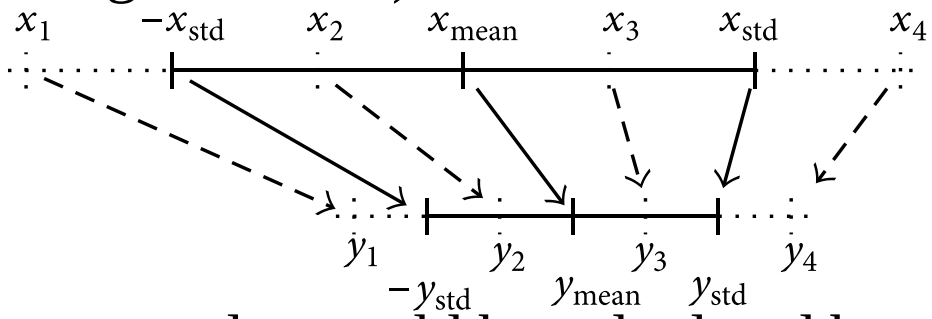


- The original minimum and maximum value (x_{min} and x_{max}) can be achieved from the statistical information of the raw data, and the new minimum and maximum value (y_{min} and y_{max}) are assigned on the basis of the activation function the new value y from the original value x is estimated by

$$y = \frac{(x - x_{min}) \times (y_{max} - y_{min})}{x_{max} - x_{min}} + y_{min}$$

Normalization

- The normal distribution method maps the original values to the new interval according to the new mean and standard deviation (see Figure below).



- Similarly, the new value could be calculated by

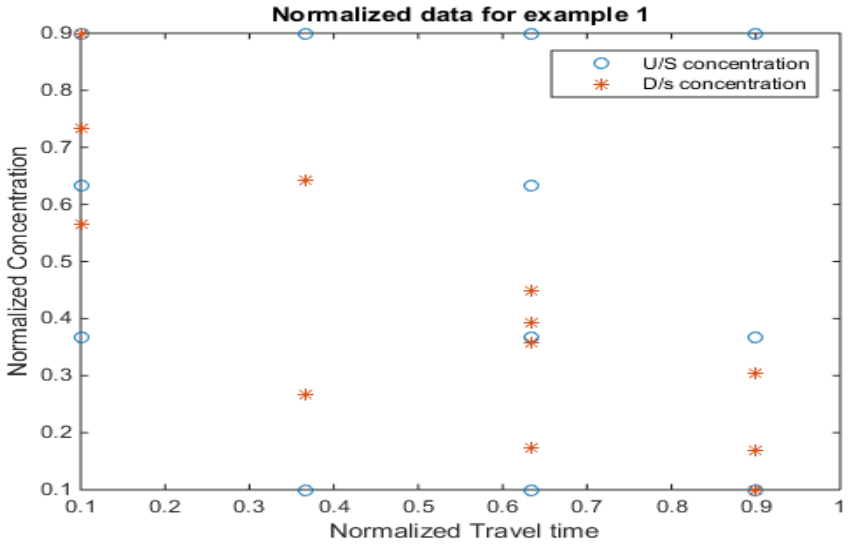
$$y = \frac{(x - x_{min}) \times y_{std}}{x_{std}} + y_{mean}$$

- NOTE the mean and deviation in place of the min.-max. values in the min max method

Data normalization : if needed

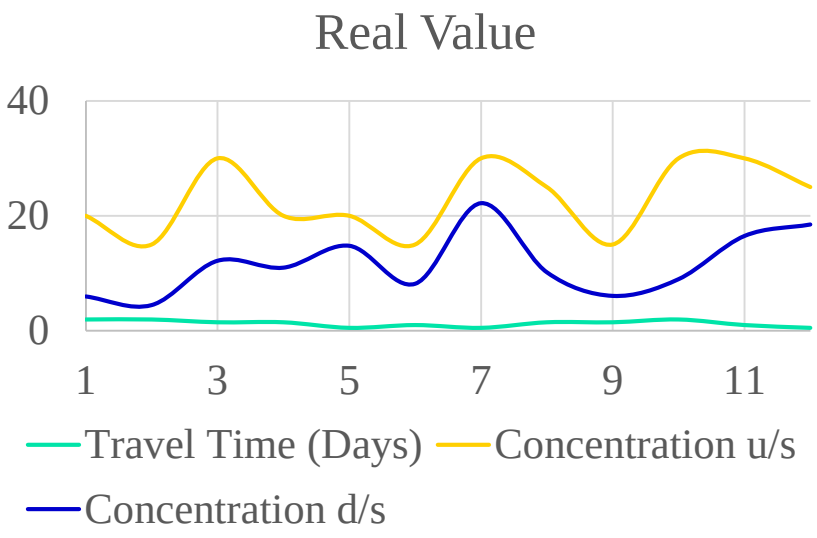
```
inmax=max(inputdata);   % the maximum values among the entire data
inmin=min(inputdata);   % the minimum values among the entire data
lfmax=0.9;  % the maximum value to be assigned for the logistic activation
function
lfmin=0.1;  % the minimum value to be assigned for the logistic activation
function
% the following code will create the normalized values
indnorm=zeros(ndata,nc);
for i=1:ndata
    for j=1:nc
        indnorm(i,j)=lfmin+(lfmax-lfmin)*(inputdata(i,j)-inmin(j))/…
(inmax(j)-inmin(j));
    end
end
```

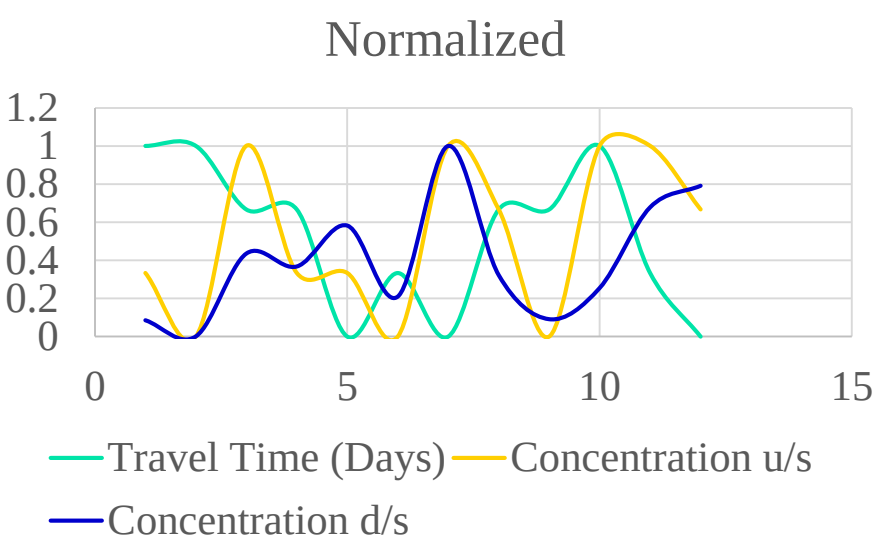
0.9000	0.3667	0.1678
0.9000	0.1000	0.1000
0.6333	0.9000	0.4480
0.6333	0.3667	0.3938
0.1000	0.3667	0.5655
0.3667	0.1000	0.2672
0.1000	0.9000	0.9000
0.6333	0.6333	0.3576
0.6333	0.1000	0.1723
0.9000	0.9000	0.3034
0.3667	0.9000	0.6424
0.1000	0.6333	0.7328




```
inputs1=indnorm(:,1)'; inputs2=indnorm(:,2)';    inputs={inputs1;inputs2};
outputs=indnorm(:,nc)';
```

Normalization results for the example

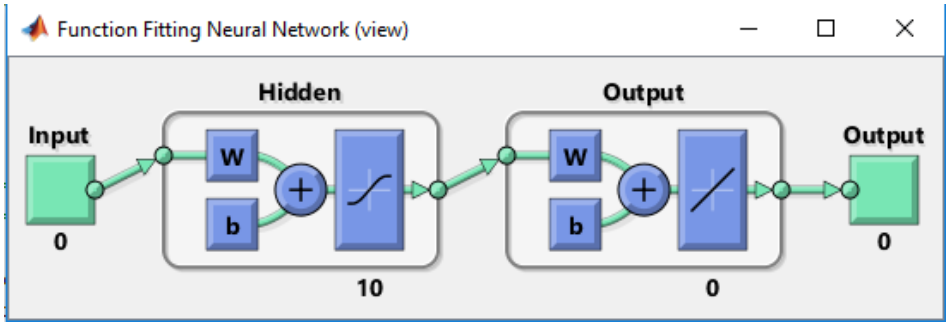




Travel Time (Days)	2	2	1.5	1.5	0.5	1	0.5	1.5	1.5	2	1	0.5
Concentration u/s	20	15	30	20	20	15	30	25	15	30	30	25
Concentration d/s	6	4.5	12.2	11	14.8	8.2	22.2	10.2	6.1	9	16.5	18.5
Travel Time (Days) N	1.00	1.00	0.67	0.67	0.00	0.33	0.00	0.67	0.67	1.00	0.33	0.00
Concentration u/s N	0.33	0.00	1.00	0.33	0.33	0.00	1.00	0.67	0.00	1.00	1.00	0.67
Concentration d/s N	0.08	0.00	0.44	0.37	0.58	0.21	1.00	0.32	0.09	0.25	0.68	0.79

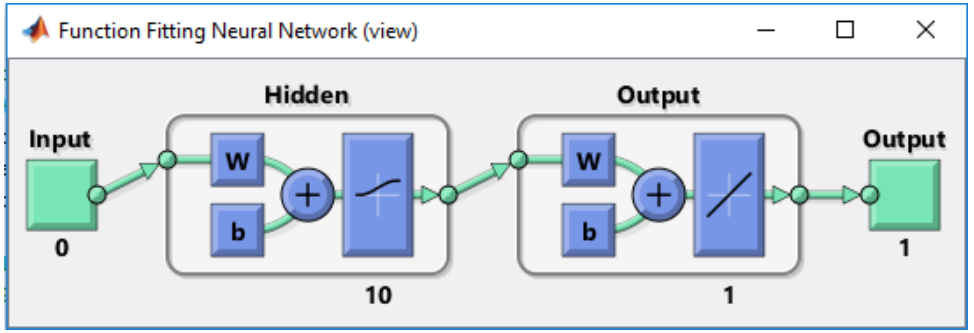
Network design

```
% Two-layer (i.e. one-hidden-layer) feed forward neural networks can fit
% any input-output relationship given enough neurons in the hidden layer.
%Layers which are not output layers are called hidden layers.
% We will try a single hidden layer of 10 neurons for this example.
% In general, more difficult problems require more neurons, and perhaps
%more layers. Simpler problems require fewer neurons.
inputs=inputdata(:,1:2); inputs = inputs'; outputs=inputdata(:,3);
outputs = outputs';
numNodesLayers=10;
net = fitnet(numNodesLayers);
view(net)
```




```
% Define topology and transfer function
net.layers{1}.transferFcn = 'logsig';% hidden layer 1 transfer function
net.layers{2}.transferFcn = 'purelin';% hidden layer 2 transfer function
view(net);
```

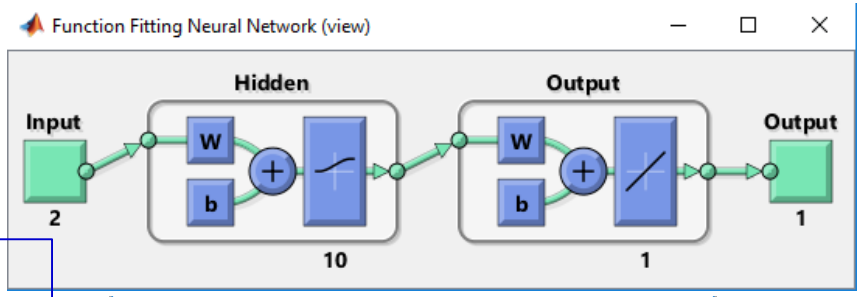
Activation functions in mat lab
'purelin'
'hardlim'
'tansig'



Network configuration and training

```
% Configure network
net = configure(net, inputs, outputs);
view(net);
```

```
% network training
net.trainFcn = 'trainlm';
net.performFcn = 'mse';
[net,tr] = train(net,inputs,outputs);
nntraintool
```

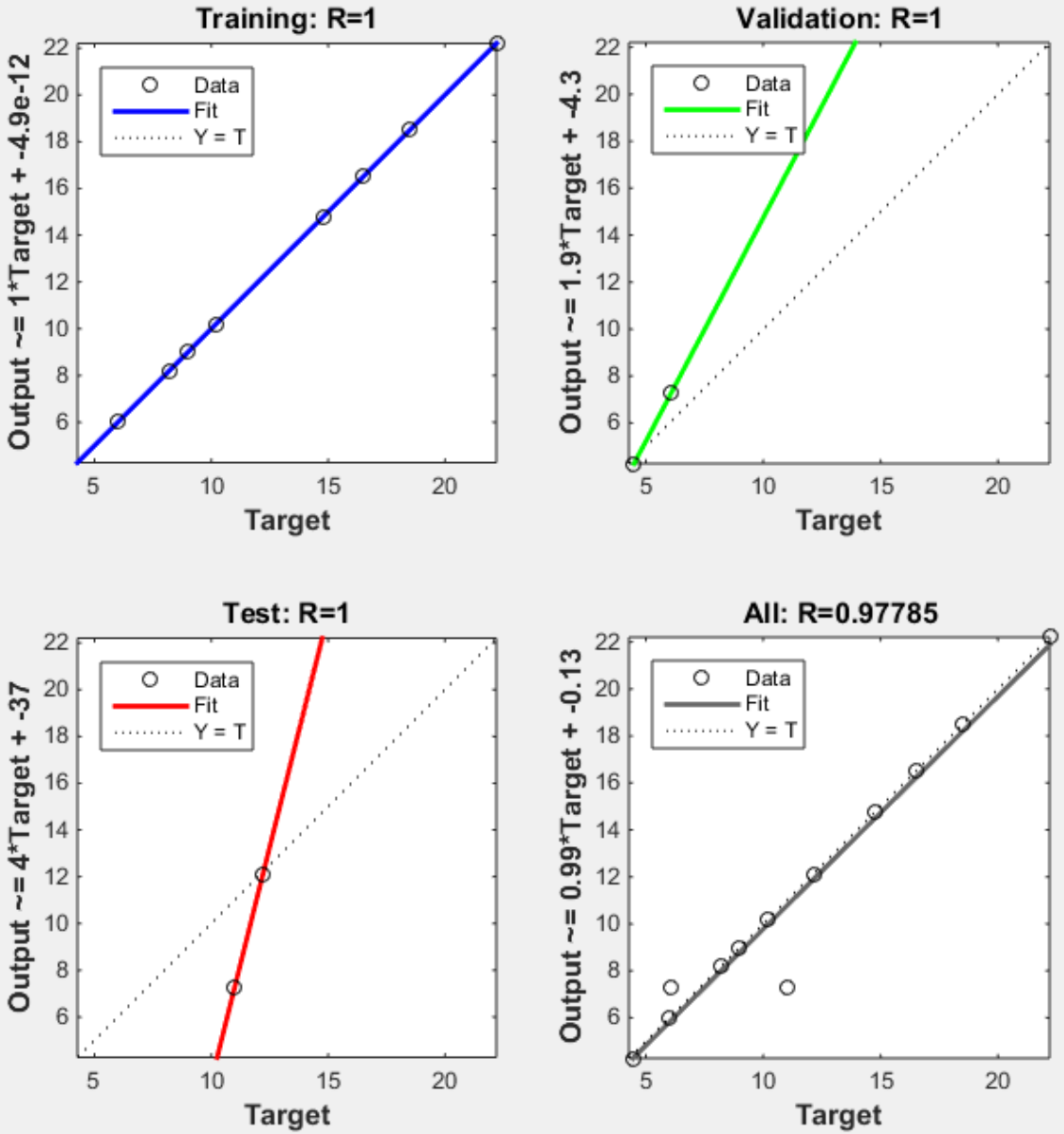


Network training functions
trainscg - Scaled conjugate gradient backpropagation
traingdx - Gradient descent with momentum and adaptive learning rate backpropagation
traingdm - Gradient descent with momentum backpropagation
trainlm - Levenberg-Marquardt backpropagation

Network performance evaluation functions
mse - Mean squared normalized error performance function
sse - Sum squared error performance function
sae - Sum absolute error performance function
mae - Mean absolute error performance function

Neural Network Training Regression (plotregression), Epoch 4, Minimum gradient reached.

File Edit View Insert Tools Desktop Window Help



Weight values visualization

```
%weights values visualization
IW1=cell2mat(net.IW); %this is the weights connecting input layer to the
first hiddel layer. numLayers-by-numInputs cell array of input weight
values
LW1=cell2mat(net.LW); %this is the weights connecting first hidden layer
to the next hiddel layer. numLayers-by-numlayers cell array of input
weight values
b1=cell2mat(net.b); % numLayers-by-1 cell array of bias values
```

```
IW1 =                    b1 =
    7.9764   -3.3507        -9.0363
    8.0493   -3.6607        -6.8680
    8.7922   -1.0225        -4.8964
    0.7938    8.7609        -3.1232
   -5.1062   -7.1132         1.3503
   -2.8551    8.3126        -1.0320
   -8.5812    0.9957        -3.3931
   -6.9597    5.3381        -5.0446
    8.4508   -2.8375         6.8096
   -8.3113   -1.9682        -9.0913
                             0.2162

 LW1 =
   -0.2579    0.2427   -0.3880    0.3428   -0.1397    0.0299    0.4041    0.0824   -0.7280   -0.2644
```

- The same can be done by:
  - nnstart

Exercise

- Develop an artificial neural network for flow routing given the following two sets of upstream and downstream flows..
- Develop the simplest artificial neural network you can that does an adequate job of prediction.

Period	U/S flow	D/S flow	Period	U/S flow	D/S flow
1	450	366	1	550	439
2	685	593	2	255	304
3	830	755	3	830	678
4	580	636	4	680	679
5	200	325	5	470	534