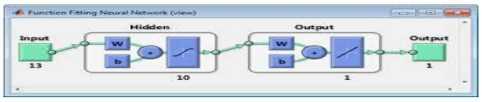


Neural Networks and Fuzzy Logic (630514)

Lecture 5

Multi-Layer Feedforward Neural Networks using matlab
Part 1

- With Matlab toolbox you can design, train, visualize, and simulate neural networks.
- The Neural Network Toolbox is designed to allow for many kinds of networks.

Workflow for Neural Network Design

To implement a Neural Network (design process), 7 steps must be followed:

1. Collect data (Load data source).
2. Neural Network creation.
3. Configure the network (selection of network architecture).
4. Initialize the weights and biases.
5. Train the network.
6. Validate the network (Testing and Performance evaluation).
7. Use the network.

Multilayer perceptron networks procedure steps using matlab:

- The structure of the network is first defined, activation functions are chosen and weights and biases are initialized.
- The training algorithm' parameters like error goal, maximum number of epochs (iterations), etc., are defined.
- Run the training algorithm.
- Simulate the output of the neural network with the measured input data. This is compared with the measured outputs.
- Final validation must be carried out with independent data.
  - ♣ **For demo programs type nnd in matlab command**
  - ♣ ***Graphical Interface Function*: *nntool* Neural Network Tool - GUI.**

The MATLAB commands used in the procedure are **newff, train,** and **sim**

1) **newff** create a feed-forward backpropagation network **object** and It also automatically initializes the network.

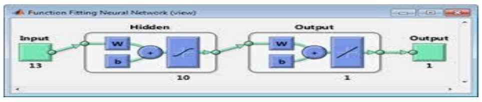
Syntax:

- ♣ net = newff (P,T,S)
- ♣ net = newff (PR, [S1 S2 …SNl], {TF1, TF2, …, TFNl}, BTF ,BLF,PF)
- ♣ net = newff (P,T,S,TF,BTF,BLF,PF,IPF,OPF, DDF)

Description:

The function takes the following parameters

- ✓ **P** - RxQ1 matrix of Q1 representative R-element input vectors.
- ✓ **T** - SNxQ2 matrix of Q2 representative SN-element target vectors.
- ✓ **S** - Sizes of N-1 hidden layers, S1 to S(N-1), default = [].
- ✓ **PR** - = Rx2 matrix of **min** and **max** values for R input elements.



- ✓ **Si** - Number of neurons (**size**) in the ith layer, i = 1,…, Nl.
- ✓ **Nl** - Number of layers.
- ✓ **TFi** - Transfer function of ith layer. Default is 'tansig' for hidden layers, and 'purelin' for output layer. The transfer functions TF{i} can be any differentiable transfer function such as TANSIG, LOGSIG, or PURELIN.
- ✓ **BTF** - Backpropagation training function, default = 'traingdx'.
- ✓ **BLF** - Backpropagation learning function, default = 'learngdm'.
- ✓ **PF** - Performance function, default = 'mse'.

And returns an N layer feed-forward backpropagation Network.
newff uses random number generator in creating the initial values for the network weights.

If neurons should have different transfer functions then they have to be arranged in different layers.

Example:
net = newff (minmax(p), [5, 2], {'tansig','logsig'}, 'traingdm', 'learngdm', 'mse');
2) train: is used to train the network whenever train is called.
Syntax:

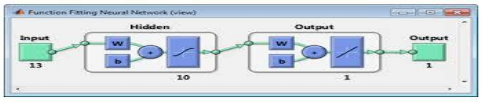
net1 = train (net, P, T)

Description:
The function takes the following parameters
- ✓ **net** - the initial MLP network generated by newff.
- ✓ **P** – Network' measured input vector.
- ✓ **T** - Network targets (measured output vector), default = **zeros**.

And returns
- ✓ **net1** - New network object.

The network's training parameters (***net.trainParam***) are set to contain the parameters:
- ***trainParam :***This property defines the parameters and values of the current training function.
- ***net.trainParam:*** The fields of this property depend on the current training function.
- The most used of these parameters (components of ***trainParam***).
  - ***net.trainParam.epochs*** which tells the algorithm the maximum number of epochs to train.
  - ***net.trainParam.show*** that tells the algorithm how many epochs there should be between each presentation of the performance.

- Training occurs according to **trainlm** training parameters, shown here with their default values:
  - ✓ *net.trainParam.epochs 100* Maximum number of epochs to train
  - ✓ *net.trainParam.show 25* Epochs between showing progress
  - ✓ *net.trainParam.goal 0* Performance goal
  - ✓ *net.trainParam.time inf* Maximum time to train in seconds
  - ✓ *net.trainParam.min_grad 1e-6* Minimum performance gradient
  - ✓ *net.trainParam.max_fail 5* Maximum validation failures



- Typically **one epoch** of training is defined as a **single presentation of all input vectors** to the network. The network is then updated according to the results of all those presentations.
- Each weight and bias updates according to its learning function after each epoch (one pass through the entire set of input vectors).
- *trainFcn*: This property defines the Backpropagation training function.

net.trainFcn = 'trainlm';

- *performFcn*: This property defines the function used to **measure** the network's **performance**. The performance function is the function that determines how well the ANN is doing its task.

net.performFcn

  - *Performance Functions*

mae *Mean absolute error-performance function*.
mse *Mean squared error-performance function.*
msereg *Mean squared error w/reg performance function.*
sse *Sum squared error-performance function.*

    - To prepare a custom network to be trained with **mae**, set

      net.performFcn = 'mae';
    - To prepare a custom network to be trained with **mse**, set

      net.performFcn = 'mse'.
    - For a perceptron it is the mean absolute error performance function **mae**. For linear regression usually the mean squared error performance function **mse** is used.
- Training stops when any of these conditions are met:
  - *The maximum number of epochs (repetitions) is reached.*
  - *Performance has been minimized to the goal.*
  - *The maximum amount of time has been exceeded.*
  - *Validation performance has increase more than max_fail times since the last time it decreased (when using validation).*
- *train* calls the function indicated by **net.trainFcn**, using the training parameter values indicated by **net.trainParam**.

3) sim is used to simulate the network when **sim** is called.

Syntax:

a = sim (net1, P)

Description:

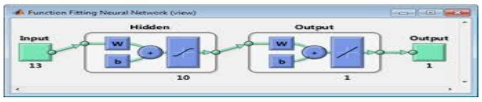
The function takes the following parameters

- ✓ **net1** - final MLP object.
- ✓ **P** - input vector

And returns

- ✓ **a** - measured output.

- To test how well the resulting MLP **net1** approximates the data, **sim** Command is applied. The measured output is **a (**simulated output of MLP network**).**



- **Error difference** **(e = T – a)** at each measured point. The final validation must be done with **independent** data.

4) **Init:** is used to initialize the network whenever **init** is called.

net = init (net)

- The **initFcn** is the function that initialized the weights and biases of the network.

5) **adapt** :allows a neural network to adapt (change weights and biases on each presentation of an input).

- The **trainFcn** and **adaptFcn** are used for the two different learning types:
    - Batch learning.
    - Incremental or on-line learning.

6) **display** the name and properties of a neural network's variables.

display (net)

7) **view :** View network structure.

view (net);

8) Type **net** to see the network:

>> net