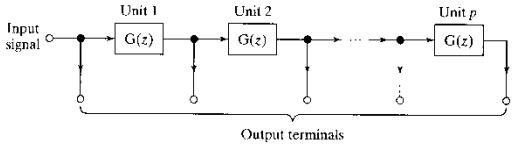
TEMPORAL PROCESSING USING FEEDFORWARD NETWORKS

Source: Haykin, 2009

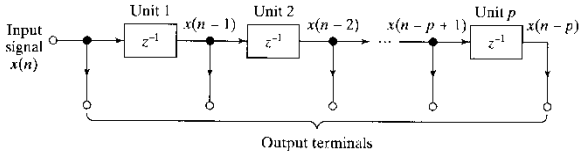
- Time is an ordered entity that is basic to many of the cognitive tasks
- Incorporation of time into a network's operation can enable the network to follow statistical variations in *nonstationary processes* (such as speech signals, radar signals, degradation signals, and fluctuations in stock market prices)
- How do we build time into the operation of a neural network?
  - ➢ *Implicit Representation*
    - For example, the input signal can be uniformly sampled, and the sequence of synaptic weights of each neuron connected to the input layer of the network is *convolved* with the sequence
  - ➢ *Explicit Representation*
    - Time is given its own particular representation
    - For example, time can be explicitly introduced into the network by presenting it as an additional input variable
- Here, we are concerned with the implicit representation of time, whereby a "static" neural network (such as an MLP) is provided with *dynamic* properties
- For a neural network to be dynamic, it must be given *memory*
- Memory may be divided into "short-term" and "long-term" memory, depending on the retention time
- Long-term memory is built into a network through supervised learning, whereby the information content of the training data set is stored in the synaptic weights
- It is the short-term memory that makes the network dynamic
- One simple way of building short-term memory into the structure of a neural network is through the use of *time delays*
  - Can be implemented at the synaptic level inside the network (Distributed TLFN)
  - Can be implemented at the input layer of the network (Focussed TLFN)
- The use of time delays in neural networks is neurobiologically motivated (signal delays are omnipresent in the brain and play an important role in neurobiological information processing [Miller, 1987])



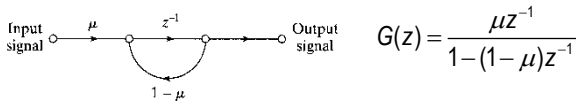
Generalized Tapped Delay Line Memory of Order *p*.

SHORT-TERM MEMORY STRUCTURES

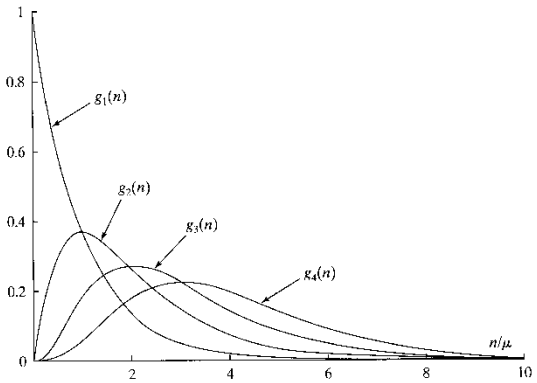
- By embedding memory into the structure of a static network, the output of the network becomes a function of time
  - The static network accounts for nonlinearity
  - The short-term memory accounts for time
- Two approaches to introducing short-term memory:
  - ➢ *Ordinary Tapped Delay Line Memory of Order p*



  - ➢ *Gamma Memory*



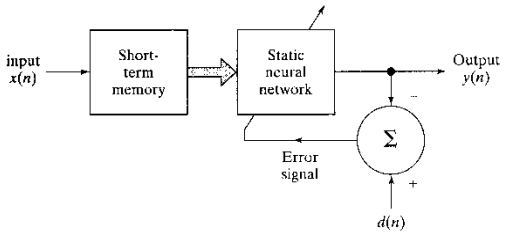
Signal-flow Graph for One Section of Gamma Memory.



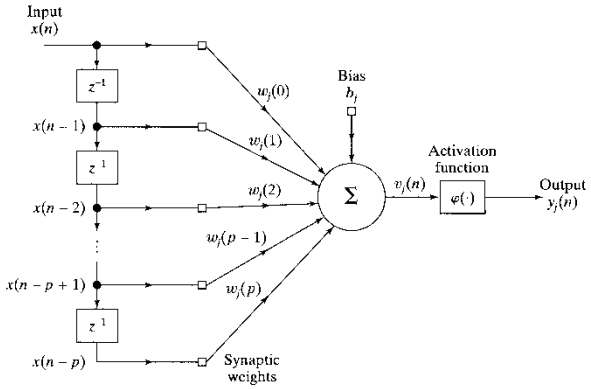
Family of Impulse Responses for the Gamma Memory for Order *p* = 1, 2, 3, 4 and μ =0.7.

Note: Ordinary tapped delay is nothing but Gamma Memory with μ =1.

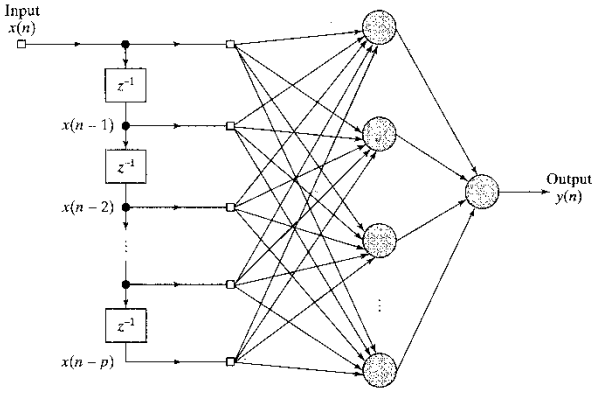
FOCUSSED TIME LAGGED FEEDFORWARD NETWORKS (TLFN)



Nonlinear Filter Built on a Static Neural Network.



Focussed Neuronal Filter.



Focussed TLFN with Omitted Bias Levels.

COMPUTER EXPERIMENT - FOCUSSED TLFN

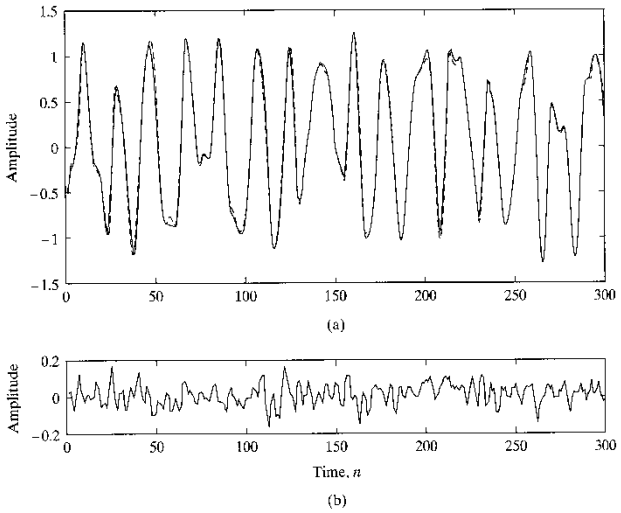
One-step ahead time series forecasting of the following frequency modulated signal:

$$x(n) = \sin(n + \sin(n^2)), \qquad n = 0, 1, 2, \ldots$$

Parameters of the Focussed TLFN:

Order of tapped delay line memory, p:	20
Hidden layer, m_1:	10 neurons
Activation function of hidden neurons:	logistic
Output layer:	1 neuron
Activation function of output neuron:	linear
Learning-rate parameter (both layers):	0.01
Momentum constant:	none

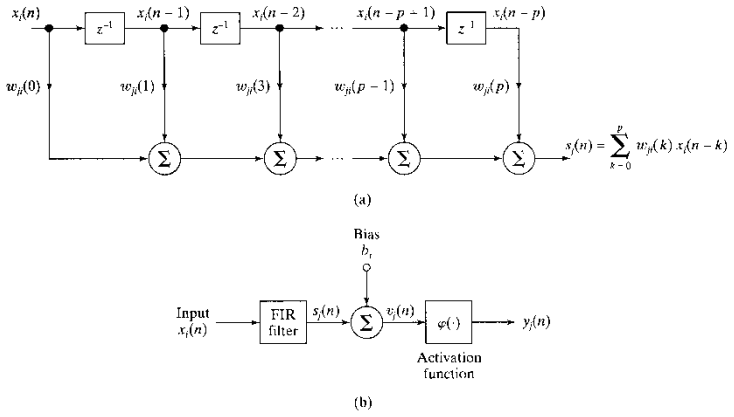
Results:



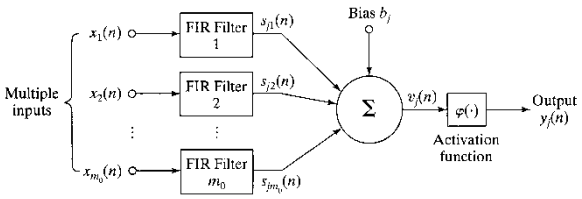
(a) Superposition of Actual (continuous) and Predicted (dashed) waveforms.
(b) Waveform of Prediction Error.

DISTRIBUTED TIME LAGGED FEEDFORWARD NETWORKS (TLFN)

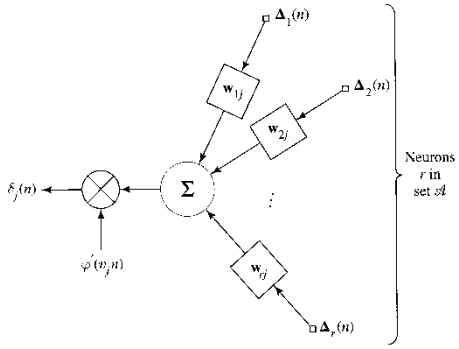
- The universal myopic mapping algorithm, which provides the mathematical justification for focused TLFNs, is limited to maps that are shift invariant
- The implication of shift invariance is that the use of focused TLFNs is only suitable for use in stationary (i.e., time-invariant) environments
- One can overcome this limitation by using a distributed TLFN, distributed in the sense that the implicit influence of time is distributed throughout the network
- Construction of distributed TLFNs is normally based on the finite-duration impulse response (FIR) filter as the spatio-temporal model of a neuron



(a) Finite-duration Impulse Response (FIR) Filter.
(b) Interpretation of Neuronal Filter as a Nonlinear FIR Filter.



Multiple Input Neuronal Filter.



Back-Propagation of Local Gradients Through a Distributed TLFN

1. Propagate the input signal through the network in the forward direction, layer by layer. Determine the error signal $e_j(n)$ for neuron j in the output layer by subtracting its actual output from the corresponding desired response. Also record the state vector for each synapse in the network.
2. For neuron j in the output layer compute

$$\delta_j(n) = e_j(n)\varphi_j'(n)$$

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta\delta_j(n)\mathbf{x}_i(n)$$

where $\mathbf{x}_i(n)$ is the state of synapse i of a hidden neuron connected to output neuron j.

3. For neuron j in a hidden layer, compute

$$\delta_j(n - lp) = \varphi'(v_j(n - lp)) \sum_{r \in \mathcal{A}} \boldsymbol{\Delta}_r^T(n - lp)\mathbf{w}_{rj}$$

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta\delta_j(n - lp)\mathbf{x}_i(n - lp)$$

where p is the order of each synaptic FIR filter, and the index l identifies the hidden layer in question. Specifically, for networks with multiple hidden layers, $l = 1$ corresponds to one layer back from the output layer, $l = 2$ corresponds to two layers back from the output layer, and so on.

Summary of Temporal Back-Propagation Algorithm

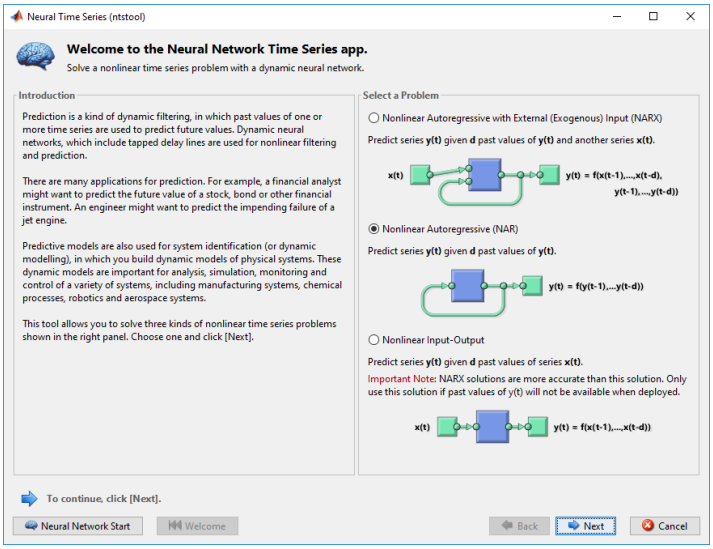
IMPLEMENTING TIME-SERIES FORECASTING IN MATLAB

MODELING AND PREDICTION WITH NARX AND TIME-DELAY NETWORKS[1]

Solve time-series problems using dynamic neural networks, including networks with feedback

Apps

Neural Net Time Series	Solve a nonlinear time series prob



Functions

nnstart	Neural network getting started GUI
view	View neural network
timedelaynet	Time delay neural network
narxnet	Nonlinear autoregressive neural network with external input
narnet	Nonlinear autoregressive neural network
layrecnet	Layer recurrent neural network
distdelaynet	Distributed delay network
train	Train neural network
gensim	Generate Simulink block for neural network simulation
adddelay	Add delay to neural network response
removedelay	Remove delay to neural network's response
closeloop	Convert neural network open-loop feedback to closed loop
openloop	Convert neural network closed-loop feedback to open loop
ploterrhist	Plot error histogram
plotinerrcorr	Plot input to error time-series cross-correlation
plotregression	Plot linear regression
plotresponse	Plot dynamic network time series response
ploterrcorr	Plot autocorrelation of error time series
genFunction	Generate MATLAB function for simulating neural network

Examples and How To

Basic Design

Shallow Neural Network Time-Series Prediction and Modeling
Make a time-series prediction using the Neural Network Time Series App and command-line functions.

Design Time Series Time-Delay Neural Networks
Learn to design focused time-delay neural network (FTDNN) for time-series prediction.

Multistep Neural Network Prediction
Learn multistep neural network prediction.

Design Time Series NARX Feedback Neural Networks
Create and train a nonlinear autoregressive network with exogenous inputs (NARX).

Design Layer-Recurrent Neural Networks
Create and train a dynamic network that is a Layer-Recurrent Network (LRN).

Deploy Trained Neural Network Functions
Simulate and deploy trained neural networks using MATLAB® tools.

Deploy Training of Neural Networks
Learn how to deploy training of a network.

Training Scalability and Efficiency

Neural Networks with Parallel and GPU Computing
Use parallel and distributed computing to speed up neural network training and simulation and handle large data.

Automatically Save Checkpoints During Neural Network Training
Save intermediate results to protect the value of long training runs.

Optimize Neural Network Training Speed and Memory
Make neural network training more efficient.

Optimal Solutions

Representing Unknown or Don't-Care Targets

Choose Neural Network Input-Output Processing Functions
Preprocess inputs and targets for more efficient training.

Configure Neural Network Inputs and Outputs
Learn how to manually configure the network before training using the `configure` function.

Divide Data for Optimal Neural Network Training
Use functions to divide the data into training, validation, and test sets.

Choose a Multilayer Neural Network Training Function
Comparison of training algorithms on different problem types.

Improve Neural Network Generalization and Avoid Overfitting
Learn methods to improve generalization and prevent overfitting.

Train Neural Networks with Error Weights
Learn how to use error weighting when training neural networks.

Normalize Errors of Multiple Outputs
Learn how to fit output elements with different ranges of values.

Concepts

How Dynamic Neural Networks Work
Learn how feedforward and recurrent networks work.

Multiple Sequences with Dynamic Neural Networks
Manage time-series data that is available in several short sequences.

Neural Network Time-Series Utilities
Learn how to use utility functions to manipulate neural network data.

Neural Network Object Properties
Learn properties that define the basic features of a network.

Neural Network Subobject Properties
Learn properties that define network details such as inputs, layers, outputs, targets, biases, and weights.

[1] Source: https://www.mathworks.com/help/nnet/modeling-and-prediction-with-narx-and-time-delay-networks.html

IMPLEMENTING FOCUSED AND DISTRIBUTED TLFNS IN MATLAB

TIMEDELAYNET

Time delay neural network

Syntax

```
timedelaynet(inputDelays,hiddenSizes,trainFcn)
```

Description

Time delay networks are similar to feedforward networks, except that the input weight has a tap delay line associated with it. This allows the network to have a finite dynamic response to time series input data. This network is also similar to the distributed delay neural network (`distdelaynet`), which has delays on the layer weights in addition to the input weight.

`timedelaynet(inputDelays,hiddenSizes,trainFcn)` takes these arguments,

`inputDelays`	Row vector of increasing 0 or positive delays (default = 1:2)
`hiddenSizes`	Row vector of one or more hidden layer sizes (default = 10)
`trainFcn`	Training function (default = `'trainlm'`)

and returns a time delay neural network.

Examples

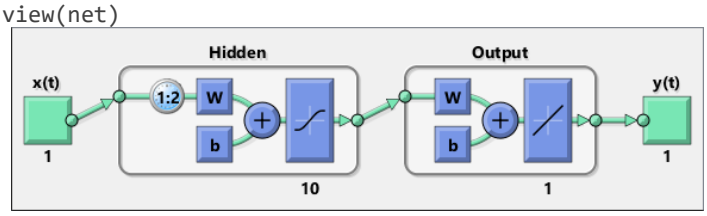
Train Time Delay Network and Predict on New Data

Partition the training set. Use Xnew to do prediction in closed loop mode later.

```
[X,T] = simpleseries_dataset;
Xnew = X(81:100);
X = X(1:80);
T = T(1:80);
```

Train a time delay network, and simulate it on the first 80 observations.

```
net = timedelaynet(1:2,10);
[Xs,Xi,Ai,Ts] = preparets(net,X,T);
net = train(net,Xs,Ts,Xi,Ai);
view(net)
```



Calculate the network performance.

```
[Y,Xf,Af] = net(Xs,Xi,Ai);
perf = perform(net,Ts,Y);
```

Run the prediction for 20 timesteps ahead in closed loop mode.

```
[netc,Xic,Aic] = closeloop(net,Xf,Af);
y2 = netc(Xnew,Xic,Aic);
```

DISTDELAYNET

Distributed delay network

Syntax

```
distdelaynet(delays,hiddenSizes,trainFcn)
```

Description

Distributed delay networks are similar to feedforward networks, except that each input and layer weights has a tap delay line associated with it. This allows the network to have a finite dynamic response to time series input data. This network is also similar to the time delay neural network (`timedelaynet`), which only has delays on the input weight.

`distdelaynet(delays,hiddenSizes,trainFcn)` takes these arguments,

`delays`	Row vector of increasing 0 or positive delays (default = 1:2)
`hiddenSizes`	Row vector of one or more hidden layer sizes (default = 10)
`trainFcn`	Training function (default = `'trainlm'`)

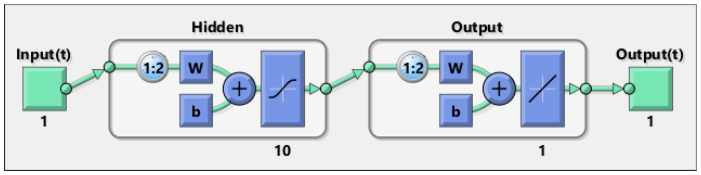
and returns a distributed delay neural network.

Examples

Distributed Delay Network

Here a distributed delay neural network is used to solve a simple time series problem.

```
[X,T] = simpleseries_dataset;
net = distdelaynet({1:2,1:2},10);
[Xs,Xi,Ai,Ts] = preparets(net,X,T);
net = train(net,Xs,Ts,Xi,Ai);
view(net)
```




```
Y = net(Xs,Xi,Ai);
perf = perform(net,Y,Ts)
perf =

    0.0323
```