Table of Contents

CHAPTER X- TEMPORAL PROCESSING WITH NEURAL NETWORKS 2
1. STATIC VERSUS DYNAMIC SYSTEMS 3
2. EXTRACTING INFORMATION IN TIME 6
3. THE FOCUSED TIME DELAY NEURAL NETWORK (TDNN) 10
4. THE MEMORY PE 18
5. THE MEMORY FILTER 27
6. DESIGN OF THE MEMORY SPACE 30
7. THE GAMMA MEMORY PE 33
8. TIME LAGGED FEEDFORWARD NETWORKS (TLFN) 39
9. FOCUSED TLFNS BUILT FROM RBFS 49
10. PROJECT: ITERATIVE PREDICTION OF CHAOTIC TIME SERIES 52
15. CONCLUSIONS 56
SHORT-TERM VERSUS LONG-TERM MEMORY 60
STATIC VERSUS DYNAMIC MODELING 60
MEMORY DEPTH 61
PROJECTION WITH THE CONTEXT PE 62
NONLINEAR SYSTEM IDENTIFICATION WITH NEURAL NETWORKS 63
MORE VERSATILE MEMORIES 67
TAKENS EMBEDDING THEOREM 70
DYNAMIC SYSTEMS 72
STATIC MAPPERS 72
LONG-TERM 72
SHORT-TERM 72
FOCUSED 73
MEMORY STRUCTURE 73
MEMORY TRACES 73
MEMORY FILTER 73
TDNN 73
CONTEXT PE 73
MEMORY DEPTH 73
DYNAMIC NETWORKS 74
DELAY 74
FEEDBACK 74
KERNEL 74
MEMORY RESOLUTION 74
AWAIBEL 74
WAN 75
GAMMA NET 75
SANDBERG 75
TLFNS 75
SUPPORT 75
MEMORY TRACE 75
EQ.7 76
EQ.13 76
EQ.3 76
STATIONARY 76
DEVRIES AND PRINCIPE 76
PRINCIPE AND HAYKIN 76
GAMMA AS AN EXTENSION OF WIENER FILTERS 77

Chapter X- Temporal Processing with Neural Networks

Version 2.0

This Chapter is Part of:

Neural and Adaptive Systems: Fundamentals Through Simulation© by

Jose C. Principe
Neil R. Euliano
W. Curt Lefebvre

The goal of this chapter is to introduce the concepts of time lagged feedforward networks as a class of dynamic networks specially designed to represent time information, and with a complexity between the full recurrent nets and the simple linear combiner.

- 1. Static versus dynamic systems
- 2. Extracting Information in time
- 3. Time Delay Neural Networks (TDNN)
- 4. The memory PE
- 5. Memory depth and the length of the impulse response
- 6. Definition of memory PE properties
- 7. The memory filter
- 8. Memory traces as a projection of the input signal
- 9. Time lagged feedforward networks (TLFNs)
- 10. How to adapt focused TLFNs
- 11. Design of the memory space
- 12. The gamma memory PE

- 13. Focused TLFNs built from RBFs
- 14. Prediction of chaotic time series
- 15. Conclusions

Go to next section

1. Static versus Dynamic Systems

Chapter IX covered the first type of dynamic systems , i.e. systems where the response evolves in time. Before the adaptive filter chapter, all the learning systems where static mappers . The system response was computed instantaneously from the available input pattern and would not change if the input was kept constant. This is an important subset of learning systems, but as we saw in Chapters VIII and IX there are systems that do not have instantaneous response to inputs. They need some time to stabilize their outputs for a given input excitation.

A linear combiner has memory, i.e. it preserves internally in the tap delay line the past values of the input. When an input is applied, the input samples need to propagate through the filter topology. For a constant input, the response of the FIR will stabilize only after a time equal to the number of delay line taps (the transient response). So static mappers have zero transient time, while the linear combiner has a finite transient response.

Another important class of systems with non-zero transient response is the recurrent topology. *Recurrent topologies have feedback*, i.e. the system output or the output of an intermediate PE is fed back to the input or some internal PE. The feedback brings the influence of the past input (or past output) to the response at the present time so it is also a form of memory. If the system has feedback the response will in general take some time to stabilize. For some values of the system parameters the response may never stabilize, so we see that new system properties (stability) are emerging for recurrent systems. Both these systems (memory based or recurrent) are called dynamical

networks and we will start their study in this chapter. ALL analog systems are dynamic, i.e. their responses take a finite time to reach a steady value (if at all). This naturally creates the concept of time scale and brings new properties when compared to the static mappers.

The weights in a static mapper such as the multilayer perceptron, also represent memory about the data it has been trained with. This type of memory should not be confused with the memory we discussed above in the linear combiner. Therefore we call long-term memory the information contained in the weights, and short-term memory the past information available as variables within the neural network topology. short-term versus long-term memory

In digital simulations we have the power to artificially control time. The sampling frequency links the samples to time, but one can, for most practical applications, think of the data as a sequence of numbers and forget that there is an underlying time scale. Moreover, the availability of short-term memory structures utilized at the system input allows the system to work with many samples at a time, as if the signal was static. In this sense *short-term memory structures have the power to transform dynamic phenomena in multidimensional static patterns*. We saw exactly this in Chapter VIII when the output of a delay line of N taps was shown equivalent to a point (a static pattern) in a space of size N. Finally, the speed of the digital computer enables virtually instantaneous computations, which gives us the power of obtaining the end result in "zero steps".

An important question is: are dynamical models preferable to static models? This is a difficult question to answer. Statistics require static data or at least that the data properties do not change over time (stationary signals). A signal in time is a function, so functional analysis is required for their study, which complicates matters a lot. It is fair to say that our mathematical tools are much more developed for static than for dynamical models, which may explain the generalized use of static modeling.

On the other hand, we see the processing power of biological systems, still more

accurate and robust than most man-made systems, and we have to pause and wonder why. If there is any advantage in time processing, biological systems which evolved embedded in time signals must have learned how to use time for information processing. So, it is certainly important to look at time processing as a promising alternative, albeit more complicated, to the more conventional theory of static machines. When compared with static systems, *the weights of dynamical systems are capable of codifying enhanced information about the input (through filtering in the short-term memory window)*. In fact one can consider a static mapper as a special case of a dynamical system with a window length of one sample. static versus dynamic modeling

NeuroSolutions 1
10.1 Time disambiguates data clusters

Let us create exactly the problem we discussed: a single input datafile containing the amplitudes of 2 alternating sinewaves of different frequencies and equal amplitudes. Let us assume that one frequency is one class and the other frequency belongs to another class. If we try to separate the two segments with a single input one hidden layer MLP, the class separation must be based on the amplitude of each sample. It is easy to convince ourselves that the task is impossible since the two classes have exactly the same amplitude mean and range, i.e. they are indistinguishable from the point of view of amplitude. We will show this with the scatter plot.

Now let us substitute the input axon by the simple first order recurrent lowpass filter we introduced in Chapter VIII. Notice that the number of inputs and system topology is the same, however, when we run the system the separation becomes possible, with some error. Notice that the errors are near the dc value.

What is the new information that the recurrent PE brings to the processing? The recurrent PE adds its past value to the present input, i.e. it brings information from the past samples to the present processing task, unlike the MLP. This shows that the recurrent system is exploring information from the signal time structure, and

that this information is crucial to solve the problem.

If we change the value of the time constant to 0.5 the MLP is unable to differentiate the two classes. So how much information is brought from the past (given by the value of the feedback parameter) is crucial to separate the classes.

Note that if we knew that the signals were sinewaves other and simpler methods could be used to make the separation (a simple zero crossing detector could do the classification). But here we are using the black-box approach, i.e. we assume no knowledge about the patterns to solve the problem.

NeuroSolutions Example

Dynamic neural networks are topologies designed to explicitly include time relationships in the input-output mappings. Time lagged feedforward networks (TLFNs) are a special type of dynamic networks which integrate linear filter structures inside a feedforward neural network, to extend the nonlinear mapping capabilities of the network with time representation. In this context, the delay line of the linear filter will be called a *short-term memory mechanism* . *In TLFNs time representation is brought inside the learning machine, as opposed to windowing the input signal*. TLFNs have *distributed* short-term memory. We will start by studying several linear short-term memory mechanisms, and will combine them with nonlinear PEs in restricted topologies called focused . *Focused TLFNs have memory only at the input layer and can still be adapted with static backpropagation*, so we have already the tools and knowledge to apply focused TLFN for time processing. Next, we will study the function of memory PEs and show that they define the basis for the local projection space. The delay line is compared with the context PE and then an unifying memory PE called the gamma PE will be introduced.

Go to next section

2. Extracting Information in time

As we saw in early chapters, the static classification problem is well understood. It can be

formulated as a special case of an arbitrary mapping between two vector spaces (function approximation) with indicator functions. In static pattern recognition the dimension of the input data vector defines the size of the pattern space (e.g. if the experiment has two independent variables, the size of the input space is 2-D). In static pattern recognition the data clusters are described statistically (mean and variance). The classification problem remains the same when we shuffle the presentation of the data to the learning machine, because we assume that there is no ordering in the data clusters (i.e. we assume no internal sequencing in the data).

Let us now assume that instead of static inputs we seek to recognize *time patterns* from a M dimensional locally stationary signal x(n). We assume that the signal is stationary for the length of observation (locally stationary signal) as in Figure 1.

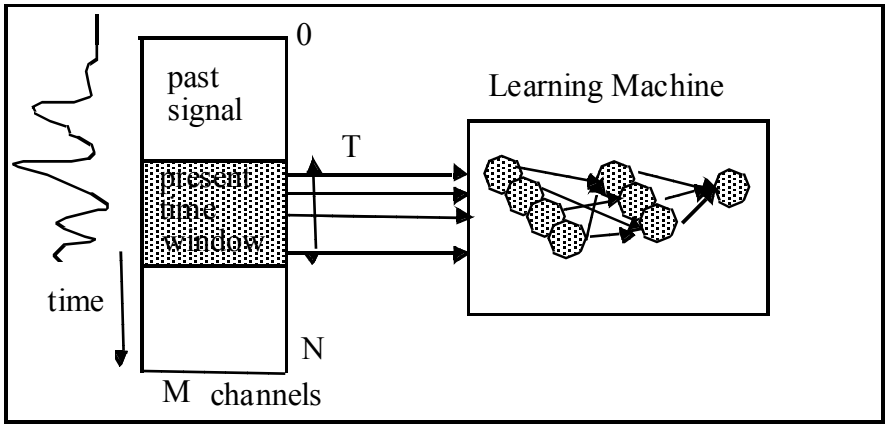


Figure 1. Processing of time varying signals with an external window

In static pattern recognition, the way the data is drawn from the input space is irrelevant. For this problem, the measurements from the world are no longer an independent set of input samples, but *functions of time. A single sensor produces a sequence of measurements* that are linked by an ordered relation, the signal time structure. If we change the order of the samples we are distorting the time signal x(n), i.e. we are changing its frequency content. *So ordering of samples must be preserved in temporal processing*.

An important question is how to *select the length of time* (called the window length) to

collect the measurements to represent the properties of x(n). If the window is too small, the information we are seeking may not be captured, i.e. the time pattern may be truncated. One option is to consider the full length (N samples) of the M dimensional signal as a single vector of dimensionality NxM. Under this condition, *the signal is no longer time dependent, it becomes a point in the space of size NxM, i.e. a static pattern*. However, the brute force mapping from time to space produces an enormous increase in the dimensionality of the data space and therefore of the number of weights in our learning machine. As we saw in Chapter IV large problems make training very slow (or impossible) and very sensitive to noise.

As we saw in Chapter VIII, the signal time structure limits the dimensionality of the manifold where the signal trajectory exists. Hence, we normally can work with smaller dimensionality spaces without loss of information. The problem is that we have to find *the appropriate reconstruction space size* by selecting the length T of the time window. Moreover, we can enhance the difference between the signal we are interested and all the others by projecting onto a single direction where very little is lost about the input signal (i.e. by adapting the network coefficients). This is the basic idea of filtering.

NeuroSolutions 2

10.2 Disambiguate data with the tap-delay line

For the same problem of Example 1, let us modify the input PE to a tap delay line. Enter by hand the values of the filter weights (all ones, which corresponds to a lowpass filter). Start with 3 taps and find out that the system never learns. Then go to 5 taps and verify that the system learns pretty well. Then go to 15 and see that the number of errors decrease. Reduce the number of taps but increase proportionally the delay between taps to find out that what matters is not the number of weights in the system but the length of the time window.

You can then let the weights of the FIR adapt. Now find out if the system found the same solution… It did not. This means that there are other solutions that minimize the error, i.e. filtering is important.

NeuroSolutions Example

In order to explore the signal's time structure for processing, the learning machine must have access to the time dimension. Recall the discussion of the role of the tap-delay line in Chapter IX when we studied the linear combiner, the simplest learning machine. Since physical systems are causal, *the search is restricted to the past of the signal*. Physical structures that store the past of a signal are called short-term memories or simply memory structure . Basically, *a short-term memory structure transforms a sequence of samples into a point in the reconstruction space*.

In Chapter IX the tap delay line was introduced and was used in conjunction with a simple adder to build the linear combiner. In the case of the tap delay line *one can either consider the memory as a window over the data or part of the learning machine.* With the former view the linear combiner is just a regressor which does not offer the right perspective to understand filtering. Moreover, the first order recurrent system also has memory but this memory is not naturally put in terms of ideal delay operators. Therefore a more principled approach must be taken.

Our approach in this chapter is to *incorporate memory structures inside* the learning machine. This means that instead of using a window over the input data, which creates a memory but is external to the learning machine, we will create PEs dedicated to store either the history of the input signal and/or the PE activations when the PE is hidden (Figure 2).

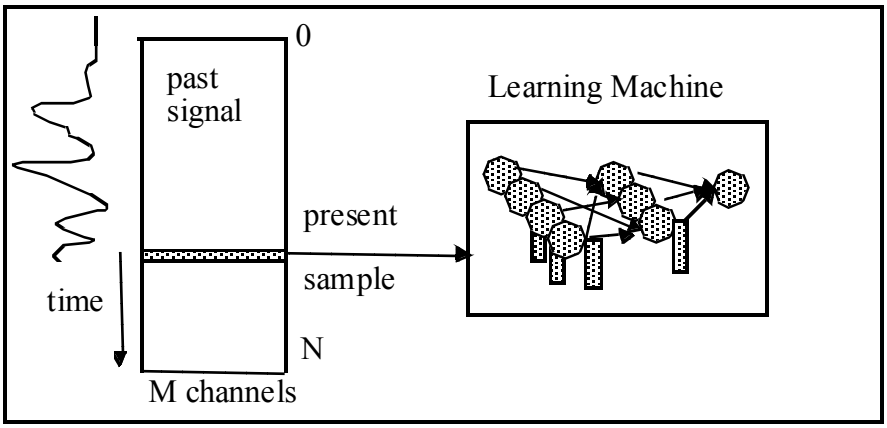


Figure 2. The new paradigm for processing time signals

By bringing the memory inside the learning machine we expect to improve the efficiency of the representation in time in the following aspects:

- Choose the size of the time window that best suites the processing task.
- Choose the weighting of the data samples in the window in order to decrease the output error.
- Moreover, these learning machines will solely receive with the present sample from the external world, just like the biological archetype.

The added advantage of temporal processing is that the system can utilize filtering (frequency domain information) to reach the processing goal. It is also obvious that the learning machine is now required to perform a more complex function, because it has to codify time information. Fully recurrent networks have this ability, but they are too complex. We will study in this Chapter topologies of intermediate complexity between the static networks and the fully recurrent networks to process signals in time.

Go to next section

3. The Focused Time Delay Neural Network (TDNN)

Even without presenting new material we already have the knowledge to create, train and understand our first dynamic neural network by combining the linear combiner with the multilayer perceptron. More specifically we substitute in the linear combiner the adder by several nonlinear PEs in a feedforward arrangement. Or alternatively, we replace the input PEs of a MLP by a tap-delay line. This topology has been called the focused time delay neural network (TDNN). The topology is called *focused* because the memory is at the input layer (Figure 3). TDNNs with short-term memory distributed across the network was first applied by Waibel to speech recognition.

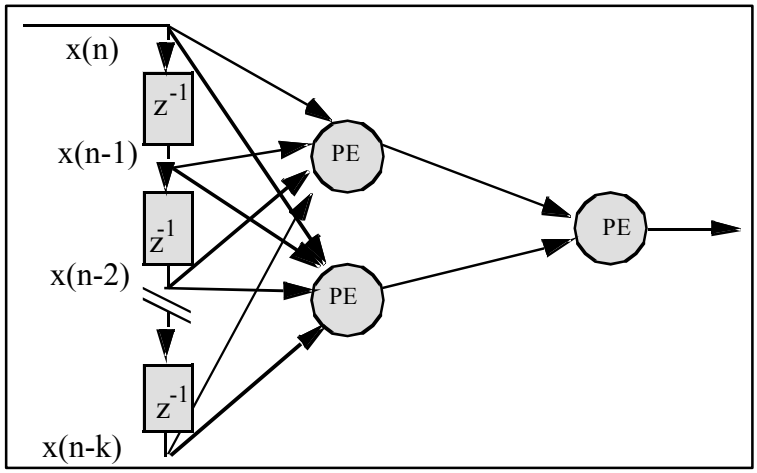


Figure 3. A focused TDNN with one hidden layer and a tap delay line with k+1 taps.

The delay line of the focused TDNN stores the past samples up to its size (the number of taps minus one). The combination of the tap delay line and the weights that connect the taps to the PEs of the first hidden layer are nothing but linear combiners followed by a static nonlinearity. So the first layer of the focused TDNN is a filtering layer, with as many adaptive filters as PEs in the first hidden layer. The outputs of the linear combiners are passed through a nonlinearity (of the hidden layer PE) and are then further processed by the subsequent layers of the MLP to achieve one of the following tasks:

- In classification the goal is to find weights that separate the signal trajectories which correspond to different time patterns.
- In system identification the goal is to find the weights that best match the present output of the system by combining the information of the present and a predefined number of past samples (given by the size of the tap-delay line).
- In prediction the goal is to approximate the next sample as a nonlinear combination of past input samples (given once again by the size of the tap delay line).

For all practical purposes we can think of the focused TDNN as a MLP that uses not only the current sample of the time series but also a certain number of past samples. So we could design a MLP equivalent to the focused TDNN if we use multiple samples from the time series instead of using a single input and the tap delay line. Effectively we are place a rectangular window over the time series and choose an MLP with as many inputs as samples in the window. Note however that in this case the memory becomes external to the network.

The problems of designing TDNN topologies are all the ones we discussed for the MLP with the addition of choosing the size of the tap delay line (also called the memory layer). We will discuss later a more principled way to address this issue. Here it suffices to say that the size of the memory layer is dependent upon the number of past samples that are needed to describe the input characteristics in time. This number depends upon the characteristics of the input and the task, so it has to be determined in a case by case basis. Before we are able to apply the focused TDNNs to solve problems we have to address how to train them.

3.1. Training the focused TDNN

One of the appeals of the focused TDNN of Figure 3 is that it *can still be trained with static backpropagation*, provided a desired signal is available at each time step. The reason is that the tap delay line at the input does not have any free parameters, so the only adaptive parameters are in the static feedforward path, which are trainable with the backpropagation algorithm of Chapter III without any modifications. We already utilized static backpropagation to train the network weights for examples 1 and 2 with successful results. We will discuss further the training of dynamic neural networks in the next chapter.

3.2. Applications of the focused TDNN

The focused TDNN topology has been successfully utilized in nonlinear system identification, time series prediction, and temporal pattern recognition. We have demonstrated this topology with temporal pattern recognition for the simple case of discriminating between two sinewave segments. We will now create a more challenging task of phoneme classification.

NeuroSolutions 3
10.3 Temporal pattern recognition with the focused TDNN

In this example we will exemplify the power of focused TDNN to classify phonemes. A phoneme is the fundamental building block for the sounds that make

up our spoken language. The phoneme is the sound equivalent of the letter in our written alphabet. The problem with phonemes is that they exist in time, i.e. they are time phenomena. The phones are produced by puffs of air coming out of the lungs that make our vocal cords vibrate (sometimes) and are shaped by the resonances of our vocal track. So when we collect sounds with a microphone we translate the sound in electrical voltages and we can see them in a oscilloscope. Each phoneme corresponds to a different waveform, so if we want a TDNN to differentiate among phonemes we have to do time processing.

Researchers in speech found out that a good way to characterize phonemes is to model the time series with an AR model. In chapter IX we saw that through prediction we can adapt the linear combiner to predict the next time series sample. The coefficients of the predictor are called LPC (linear predictive coding) coefficients and are going to be used here as the translators of each segment of the speech waveform. Every 10 msec (roughly 128 samples) we read the coefficients of the linear combiner, obtaining a set of 11 LPC coefficients. So the speech time series is translated into a sequence of LPC vectors (of size 11) which will be input to the focused TDNN.

Now the problem of training the TDNN is relatively easy to explain. When the sound we want to classify appears in the time series, we give a desired response of 1 to the TDNN. When the phoneme is absent the desired response should be zero. So we have to create different desired signals in time to train the TDNN to recognize phonemes. Here the desired response will be a square wave of 0 (phoneme absent) and 1 (phoneme present). This is one of the differences between temporal and static pattern recognition, the desired response is also a time signal. So we have to create a TDNN with the number of outputs equal to the number of phonemes. Here we will only classify 3 phonemes.

The other big difference between static and temporal pattern recognition is that

here the pattern has a given (but unknown and eventually variable) duration. So we have to set the size of the tap delay line such that it looks far back in time to recognize the time structure of the phoneme which is translated here in a sequence of LPC vectors. This is not a trivial problem and as we discussed is one of the sources of difficulty for temporal pattern recognition. If the window is too small not enough vectors are present in the window and performance suffers. On the other hand if the window is too long the system may be confused and there are many more weights to train. So the size of the window is a design parameter. In the TDNN we have to physically modify the network topology to adjust the length of the window in time. We selected it here at 3 LPC vectors (a memory depth of 3).

In the TDNN we still can use static backpropagation to train the system. It is amazing that we can, but let us observe the topology in detail to understand the reason why. When the data is clocked in the delay line and a desired response is provided, the weights can be trained at each time tick. There is no recurrent connections, i.e. the system is fully static, so it looks like a very large static net, and backpropagation can train the weights. This is in fact one of the advantages of this topology, the weight gradients are not dependent upon time because the net is static. We can think that training in time for these nets is just like batch learning (except that there is a physical meaning to the sequence of patterns given by the time structure of the signal). Let us train the system and see how it works.

As with the examples of previous chapters, you the reader should modify the parameters of the system to see how well it works, and when it fails. Understanding why a system fails is normally much more enlightening than when everything goes right.

In particular you should observe the difference between dynamic and static pattern recognition. Just set the tap delay line with a single tap, i.e. the classification will be done only with one LPC vector. Try to train the system. You will see that the

performance drops, i.e. by including time information the classifier can be improved.

NeuroSolutions Example

Next, the focused TDNN is applied to nonlinear system identification extending the linear combiner to nonlinear models.

NeuroSolutions 4
10.4 Nonlinear system identification with the focused TDNN

This problem is a system identification problem. To illustrate the power of the TDNN we are going to identify a nonlinear plant, and compare the results of the TDNN with those of the linear combiner with the same number of delays. The plant is given by the following equations

$$x_1(k+1) = \left(\frac{x_1(k)}{1+x_1^{\,2}(k)} + 1\right)\sin(x_2(k))$$

$$x_2(k+1) = x_2(k)\cos\{x_2(k)\} + x_1(k)\exp\left(-\frac{x_1^2(k)+x_2^2(k)}{8}\right) + \frac{u^3(k)}{1+u^2(k)+0.5\cos\{x_1(k)+x_2(k)\}}$$

$$y(k) = \frac{x_1(k)}{1+0.5\sin(x_2(k))} + \frac{x_2(k)}{1+0.5\sin(x_1(k))}$$

The input is

$$u(k) = \sin\frac{2\pi k}{10} + \sin\frac{2\pi k}{25}$$

The block diagram for system identification is still the same as in the linear case. A common input is given to the TDNN and the plant and the desired response for the TDNN is obtained from the plant output. In our case we have stored in files both the input and the plant response to make it compatible with NeuroSolutions. In this case the input is a sum of two sinusoids. We will put scopes at the input and at both the model and plant outputs to see how the learning progresses.

We start with the linear combiner of order 10. Run the simulator and observe that the error stalls immediately at a very large value. Increasing the size of the linear

combiner does not help. Try 20 taps and higher. This can be understood by the nonlinear nature of the plant. Put a FFT at the input and output and observe that new frequencies are created by the plant, hence a linear system is hopeless in this case.

Let us now see how the focused TDNN does. It is amazing, but a TDNN with 7 PEs in the hidden layer does almost a perfect job, and the training is very fast. Modify the size of the hidden layer and of the input layer to understand better their role in the solution. For instance, is there a size of the tap delay line below which the system does not train, irrespective of how many PEs we use in the hidden layer? Conversely, is there a minimum number of hidden PEs below which no matter how large the time window is the system never trains?

NeuroSolutions Example

Finally, we will also apply the focused TDNN for chaotic time series prediction.

NeuroSolutions 5

10.5 Nonlinear prediction with the focused TDNN

This example deals with the other major application of focused TDNN, nonlinear prediction. In fact as was explained in Chapter IX, prediction is a form of system identification where both the input and desired signal come from the same source but at different time intervals (the input is delayed, normally by one sample). With this introduction we can see that if the system that created our time series is not linear, the linear combiner will not be able to perform as well as the TDNN.

The example that we prepared is exactly a time series that is produced by a nonlinear dynamical system, the Mackey-Glass system given by

$$\frac{dx}{dt} = -0.1x(t) + \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)}$$

As we can immediately see the system is nonlinear (notice the power 10 in the

denominator), and with a delayed feedback (the delay τ). So we can expect some type of oscillation with a very complex shape. Notice that the equations are written in continuous time, so we have to discretize the system. Here we will be using a 4th order Runge-Kutta integration, downsample it by 6, normalize to [-1,1] and use a delay of 30 (this is called the MG30 time series). MG30 happens to be a chaotic time series, i.e. the waveform is always different and unpredictable in the long run.

Once we have the time series we just need to delay the input to the TDNN by one (segment the desired file starting at 1) and run the breadboard. We will also compare the performance of the TDNN with that of the linear combiner of the same memory size. As we will see the TDNN is superior in capturing the fine detail of the waveform, but the linear combiner does an excellent job in approximating the major features of the waveform, which clearly shows the reason linear systems have been in use for so long and still satisfy most of the requirements.

You should experiment with the size of the delay line and the size of the hidden layer to understand better the breadboard. You can also delay the input by more of one sample and see how much more difficult the task becomes. The linear system breaks down quickly while the TDNN still works reasonably well.

If the waveform is chaotic, i.e. unpredictable in the long run, does it make sense to try to predict it? As we can see there is a deterministic equation that relates the samples, so locally there is a structure that can be captured by the predictor. The interest of doing prediction in chaotic signals is related to the difficulty of the task which provides an excellent ground to test predictors. Lots of signals from the real world which we usually model as noise may in fact be chaotic, and we may be able to model them.

NeuroSolutions Example

The focused TDNN is a great compromise simplicity/processing power, but it is not the only dynamic neural network topology to extract information from time patterns as we

saw in Example 1 with the first order lowpass filter. We are interested in formulating a theory for the processing of time signals with neural networks, so we will develop a new neural component explicitly designed to remember the past, which we call the *memory PE.* We will see that the ideas of filtering explained in Chapter VIII are crucial to understand the memory PEs.

Go to next section

4. The memory PE

Figure 4 shows a block diagram of the new PE called a *memory PE*, where g(.) is a delay function.

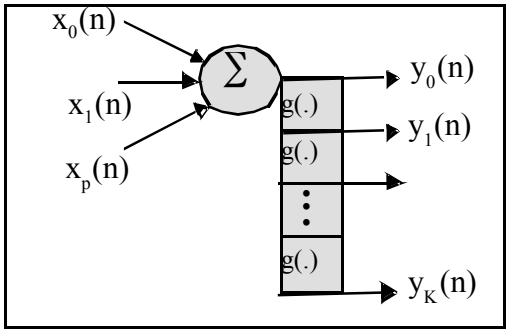


Figure 4. The memory PE

The memory PE is *linear for linear delay functions g(.),* receives in general many inputs $x_i(n)$ and produces multiple outputs $\mathbf{y} = [y_0(n),\ldots,y_k(n)]^T$ which are delayed versions of $y_0(n)$ the combined input, i.e.

$$\mathbf{y}_i(n) = \left[y_{i0}(n),\ldots,y_{iK}(n)\right]^T \qquad y_{i0}(n) = \sum_{j=0}^{p} x_j(n) \qquad y_{ik}(n) = g\left(y_{ik-1}(n)\right)$$

Equation 1

Although we have so far restricted the location of memory at the input layer, in principle memory PEs can be included anywhere in the neural network to function as time representation modules. By analyzing Eq. 1 we conclude that these short-term memory

structures can be studied by linear adaptive filter theory if g(.) is a linear operator. It is important to emphasize that the memory PE is a *short-term memory* mechanism, to make clear the distinction with the network weights which represent the *long term memory* of the network.

The memory PE has a biological interpretation (Figure 5). When the biological neuron receives in its dendritic tree multiple connections from the same axon at different levels in the tree, the signal propagation towards the soma produces different time delays.

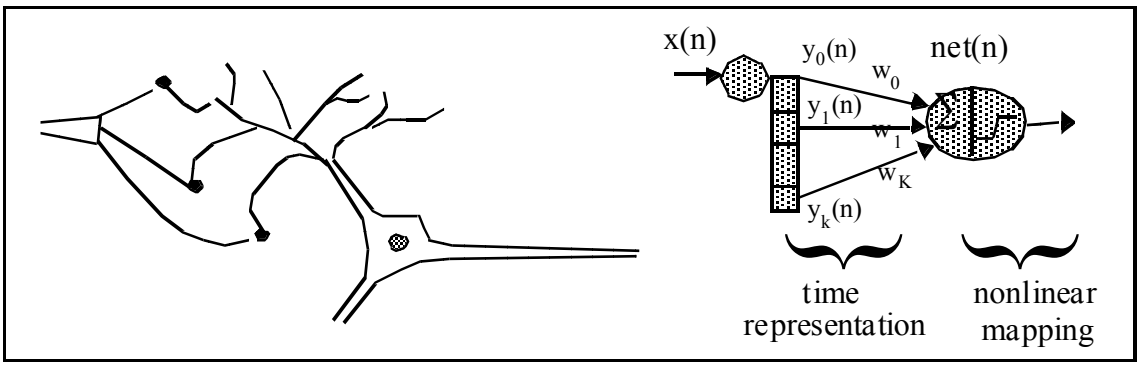


Figure 5. Similarity between biological neuron and the memory PE feeding a M-P PE.

The cell responds to the average field, and so the output will be a weighted sum of delayed versions of the original excitation. Although these aspects are not fully understood, the dendritic tree works as a *signal representation stage*. If we represent the cell by a McCulloch and Pitts PE, and the dendritic tree by a single input memory PE, the combination of the two PEs by means of a set of weights creates a processing block incorporating the same basic principles of signal representation and nonlinear response. Figure 5 shows the similarity of the biological neuron with the memory PE feeding a M-P PE.

4.1. The delay line PE

When the *memory PE is built from a delay line*, we will call it a *delay line PE*, and it implements memory by delay , i.e. by simply holding past samples of the input signal. The delay line PE is the memory structure utilized in the TDNN.

The delay line PE is a multidimensional single-input, multiple-output linear system where the tap outputs are delayed by one sample from the previous tap (Figure 6)

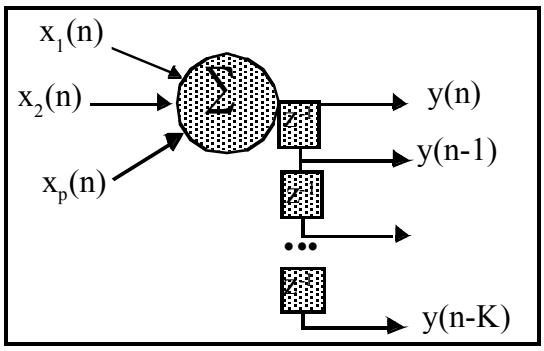


Figure 6. The delay line PE.

In terms of an equation, the delay line PE with K+1 outputs can be represented by

$$\mathbf{y}_i(n) = \left[y_i(n),\ldots,y_i(n-K)\right]^T \qquad y_i(n) = \sum_{j=1}^{p} x_j(n)$$

Equation 2

where $y_i(n)$ is the combined input to the PE. Comparing Eq. 2 with Eq. 1 we verify that g(.) is the delta function operator $\delta(n-1)$. Note that a delay line PE with K+1 outputs has K delays. At sample time n, the output of the delay line PE of size K+1 only stores samples that occurred after (and including) time (n-K), i.e. K samples into the past. Any input event occurring before time (n-K) is not represented at the output. *Effectively the delay line PE with K delays implements a time window of length K+1 samples positioned on the current sample*.

4.2. The context PE

The first order recurrent system of Figure 11 in Chapter VIII (Eq. 13) can be used to implement a linear context PE for general use in neural networks. This type of memory will be called memory by feedback and the memory PE will be called a context PE . It is only necessary to equip it with multiple inputs and eventually a bias

$$y_j(n) = (1-\mu)y_j(n-1) + \mu(\sum_{i=1}^{p} x_i(n)) + b_j \qquad i \neq j$$

Equation 3

which is depicted in Figure 7. Note that the combined input is multiplied by μ for normalization purposes.

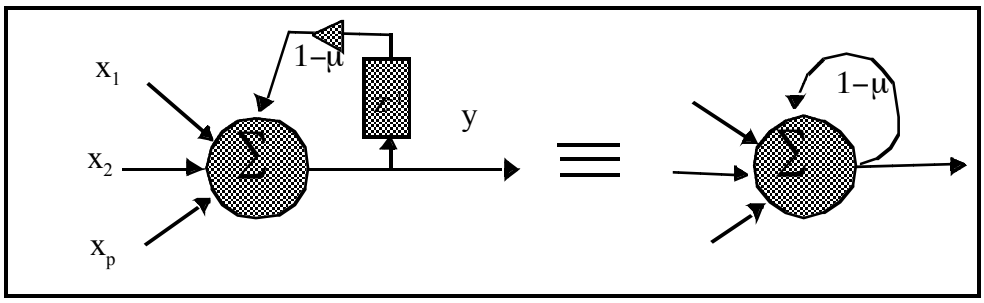
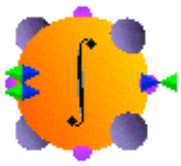


Figure 7. A linear context PE and its representation

We normally represent this PE as in the right diagram of Figure 7, where the *delay is not apparent*, but remember that *context PEs always have a delay of one sample* in the feedback loop.

NeuroSolutions 6
10.6 Context PEs revisited

Let us revisited the first example of this chapter, and analyze the reason why the system with a context PE at the input was able to solve reasonably well the discrimination task. Note that now the recurrent PE has been substituted by the context PE (they differ only in the normalization factor)



Attach a scope to the output of the recurrent PE to see why the network is capable of classifying the two signals. Although the two sinewaves at the input have the same amplitude, at the output of the recurrent PE they have DIFFERENT amplitudes because the recurrent PE is a lowpass filter and it filters more (attenuates the amplitude) the high frequency sinewave. These two different amplitudes is enough for the MLP to create a decision surface between the two amplitudes and distinguish the two regimes. Notice that with this explanation we

also understand the errors clearly…. when the low frequency sine passes close to zero the MLP is fooled and assigns those samples to the high frequency sinewave.

NeuroSolutions Example

4.3 Memory depth and the length of the impulse response

The signal processing function of these two types of memory is to hold information from the past. The question that we have to ask is how long and how accurate is the information kept in the memory structure, which naturally leads to the concept of *memory depth* . In the case of linear memories, the memory depth is related to the length of the impulse response of the memory PE. The two types of memory, memory by feedback and memory by delays, hold the information from the past differently.

The delay line PE represents the past of the signal y(n) in a discontinuous way. For delays within the memory size K, *the past is represented exactly*, but beyond this limit it is totally forgotten. The only way to represent events further back in time is to extend the size of the delay line, i.e. *to modify the topology of the PE*. Figure 8 shows an impulse at n=0 and the progression of time by the position of the window advancing along the time line. Abruptly at n=K+1 the response of the system changes from 1 to 0. Note that K is the length of the impulse response, which is also called the region of support of the response.

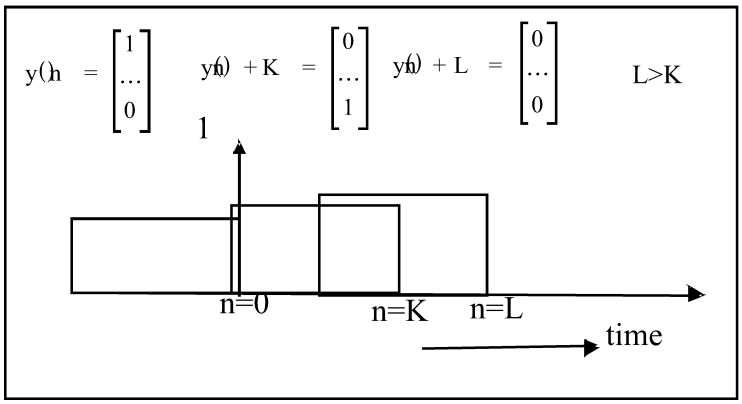


Figure 8. Response of the delay line PE.

K is exactly the number of samples the output of the delay line PE "remembers" the impulse at n=0. Let us explain this terminology. When the output of the PE is different from zero, there is evidence at the output that an event (in this case an impulse) occurred somewhere in the recent past, more precisely within K samples. So a K+1 tap PE remembers K samples in the past or has a memory depth of K samples.

NeuroSolutions 7
10.7 Memory depth of tap-delay line

Let us construct a delay line PE in NeuroSolutions and show that an impulse will disappear after a number of samples given by the number of taps. This explains why in the examples the TDNN "starts" to work abruptly when the size of the memory in increased. The tap delay line has a "all-or nothing" type of memory.

NeuroSolutions Example

Let us now study the context PE. Again we suppose that an impulse is applied at time n=0 to two context PEs H_1 and H_2, given by Eq. 3 with $\mu_2<\mu_1$ ($0<\mu_2,\mu_1<1$). In Figure 9 the impulse response appears *reversed in time* as dictated by the convolution operation. We measure the system response y(n) at time $n_t>0$, and the response is practically zero for H_1 and different from zero for H_2. You can visualize how this system responds at n_t to a delta function at n=0 by sliding the impulse response and looking at its y intercept.

When the response is non-zero at n_t there is evidence that something happened in the input channel at some time instant prior to n_t. On the other hand, if the response is zero (or vanishing small) at n_t, then we can not tell from the system output if any event occurred before n=0. *So by means of the impulse response, the information of the event occurring at n=0 was brought forward in time to n=n_t.* Or alternatively, the system "remembers" at n_t that an event occurred at n=0. In the first case there is a trace of a response (a memory trace), while in the second case the system simply "forgot" what happened at n=0.

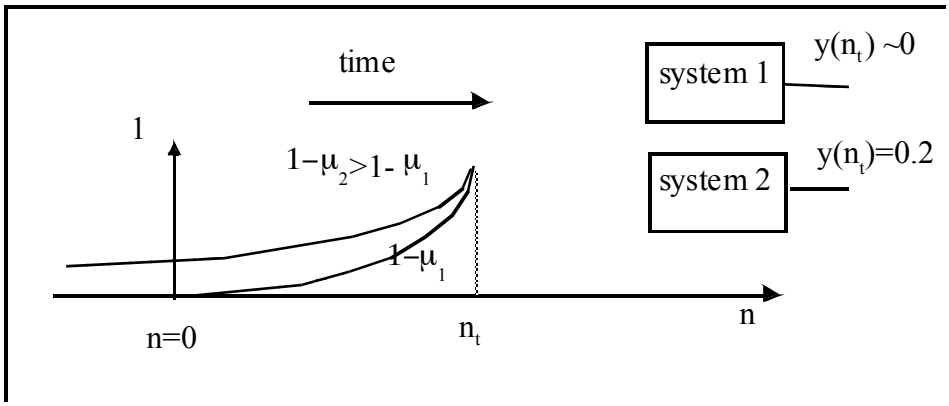


Figure 9. Response of two different lowpass filters to an impulse.

NeuroSolutions 8

10.8 Memory depth of the context PE

Let us show the memory trace in NeuroSolutions. When an impulse is presented to the memory the output decays exponentially in time. Note that the region of support of the impulse response is in fact infinite, but for all practical purposes (noise) it is finite since the amplitude becomes negligible after a finite time. Note that when you change the feedback parameter the region of support of the memory changes. This is the beauty of these memories for neurocomputing: once the system can adapt the feedback parameter it can choose the value that minimizes the output mean square error and work with the best possible memory WITHOUT changing the topology. In the TDNN to change the region of support a change in the topology is required.

NeuroSolutions Example

In the context PE the memory depth can be increased without any topological modifications by decreasing the feedback parameter μ. This is unlike the case of the delay line PE, where the only way to increase the memory depth is by *a change in the topology*, i. e. an increase in the number of tap delay.

The recurrent system of Figure 7, with a single delay unit displays an impulse response that decays towards zero with a time constant τ

$$e^{-1/\tau} = 1 - \mu \quad \rightarrow \quad \tau \sim \frac{1}{\mu}$$

Equation 4

So in reality the region of support of the impulse response is infinite. But after some time (taken normally as 4 time constants) the response becomes so small that any noise can corrupt the system output. The equivalent power under this decaying exponential and the rectangular window of N samples is obtained with a *number of samples equal to half time constant*, i.e. n = ½μ.

Notice that the past samples are not preserved exactly as in the case of the delay line PE. In the context PE the output is an addition of the present input added to a weighted version of the past output (see Eq. 3), so the information from the past is progressively distorted. This is the reason we sometimes talk about a memory trace, i.e. a modified version of the input.

The *memory trace in a context PE gracefully degrades in time*, unlike the response of the delay line PE where the transition from memory to forgetting is abrupt. Moreover, the memory depth of the context PE is *controlled by the system parameter* μ. In an adaptive system framework, *recurrent systems are at an advantage*, since the system can *control its own memory depth* by changing the parameter instead of the topology. But notice that the flexibility of this memory PE is very *limited since the system can only control the rate of decay of the memory trace.*

4.4. Memory PE properties

With this explanation we are ready to distill these concepts in two definitions that characterize linear memory PEs, i.e. PEs in Figure 4 where g(.) the transformation kernel becomes the impulse response from tap to tap in the linear case. We define a *linear memory* PE as an (eventually multidimensional) single-input-multiple-output system whose transformation kernel g(n) is causal and normalized, i.e.

$$\sum_{n=0}^{\infty}\left|g(n)\right| = 1 \qquad g(n) = 0 \quad for \quad n < 0$$

Equation 5

We define memory depth D as the modified center of mass (first moment in time) of the impulse response $g_K(n)$ of the last memory tap K. That is

$$D = \sum_{n=0}^{\infty} n g_K(n)$$

Equation 6

where $g_k(n) = g(n) \bullet g_{k-1}(n)$ and $\bullet$ is the convolution operation.

Similarly, we define the memory resolution R as the number of taps per sample (or unit time). With this definition the memory depth and the resolution are coupled i.e.

$$RD = K$$

This has interesting implications since for a given number of taps K, an increase in R (resolution) implies a decrease in D (depth). Or in other words, the only way to increase depth and resolution at the same time is to increase the number of taps of the memory.

Applying these definitions to the tap-delay line shows that D=K and that R=1. For the context PE, we have to first normalize the input by μ to formally call the PE a short term memory. In these conditions, the memory depth is $1/\mu$, and the resolution is μ. Memory depth

So we can now quantify the difference between the feedforward and the recursive memories. In the tap delay line the only way that memory depth is changed is by topologically adding or removing delay units. These memories display the maximal resolution.

On the other hand, in the recursive memories, the feedback parameter controls the properties (memory depth and resolution) of the memory. The memory depth for the context PE is $1/\mu$ and the memory resolution is μ. In fact it trades one for the other, i.e. to increase the memory depth the memory resolution is decreased. But ultimately, *the system has the ability to choose through adaptation the best compromise*, i.e. the best

value of the feedback parameter to minimize the output MSE. So this is a very important observation for connectionist memory structures. The shortcoming of the context PE is that it is a single input-single output device.

Go to next section

5. The memory filter

Let us examine the signal processing operation produced by attaching a single input memory PE to a sigmoidal PE through a set of weights (Figure 10).

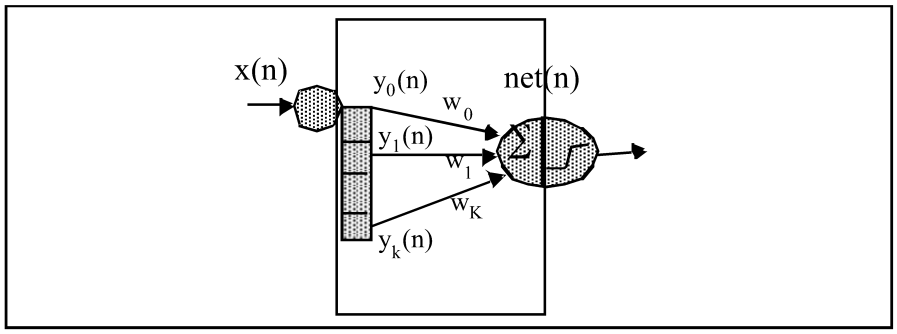


Figure 10. Block combining the memory PE with the next sigmoidal PE.

If we restrict our analysis to the subsystem created by the memory PE and the output of the linear part (*net(n)*) of the nonlinear PE, we can recognize a topology similar to the linear combiner. The delayed versions of the input signal x(n) created by the memory PE are either called the *tap signals or the memory traces* . These signals are then multiplied by weights w_k and summed to obtain net(n). Hence, we will call this arrangement a *memory filter* , and its function is to project the input signal x(n) into a linear manifold defined by the memory traces $y_i(n)$. The signal net(n) is the result of the projection of the input x(n) on the weigth vector **w**. Wan called this special configuration of the memory filter a FIR *synapse*.

The fundamental conclusion from this observation is that besides the qualitative properties of memory depth and resolution one *can use the theory and tools of filtering explained in Chapter VIII to study the effects of the memory PE in the mappings.*

10.9 Interpretation of focused TDNNs as nonlinear memory filters

Go back to the topology of Example 2 and let us interpret what is happening in the input layer. Start with a 3 tap delay line and 2 hidden PEs. We will place a 3D probe at one of the channels of the input such that we can see the two signals in signal space. We will place also a scope and a scatter plot at the pre-activity of the hidden layer such that we can see the outputs of the two memory filters. The scope shows the projections produced by the memory filters. We will start with fixed coefficients: one of the filters will be a lowpass (all ones) and the other an highpass (1,-2,1).

The scatter plot shows us exactly the set of points that the MLP is trying to separate, *so this scatter plot shows the decision space*. We have configured the display such that it first shows one class and then the other class. The goal of the MLP training is to place discriminant functions for classification in the space created by the output of the memory filters.

Start with a τ =1. The two signal trajectories are narrow ellipses which makes them difficult to separate visually in 3D space. The projections in the decision space are on top of each other, so the MLP has difficulty in placing the 2 discriminant functions on this space. The classification results are poor.

Now increase τ = 3. Notice that the signal trajectories open up in 3D space (now the ellipses use more of the signal space, making the differences in trajectories clearer). Notice now that the projections with the lowpass and highpass filters that form the decision space are further apart, so it should be easier for the MLP to do the placement of its discriminant functions. The errors are exactly at the point that the two projections cross each other. Notice that the linear projections (implemented with the linear memory filters) do not have the power to move these trajectories to different parts of the decision space. They are both anchored at the origin. This is one of the shortcomings of the TDNN topology.

Finally we can let the system learn all of the parameters including the weights of the memory filters. Do this with the same 3 tap memory and triple delay. The system found a good solution as seen by the MSE and the output. Let us now look at the weights of the memory filters. Amazingly enough, the optimal weights do not correspond to our solution of lowpass and highpass filters.

We can see clearly from the output of the filters in the scope that the solution found is to zero one of the waveforms while letting the other through (If we want to visualize the frequency response of each filter just copy the weights to a piece of paper and go to example 16 of Chapter 7 to plot the frequency response). This can be achieved if the filters are such that they place one of its zeros at the frequency of one the sinewave. This is a particular solution that works for this case of a single frequency input, so the solution found actually is pretty smart and minimal. One just needs one filter to do this, i.e. one hidden layer PE, which means that a perceptron can solve this problem. Adaptive systems tend to find the most obvious solution all the time….

Substitute the input sinewaves by triangular waves of the same frequencies, and redo the problem to see what is the solution found.

NeuroSolutions Example

5.1. Memory traces as a projection of the input signal

The delay line PE and the weights that connect it to the next layer create a memory filter which is a linear combiner. We saw in Chapter IX that the output of the linear combiner can be interpreted as the projection of the input onto the weight vector. The linear context PE is also a projection operator. In fact, the output of the context PE is the inner product (convolution) of the input with the impulse response of the PE as shown in Figure 11.

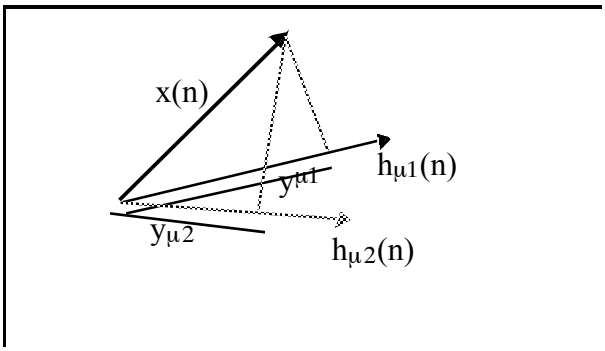


Figure 11. Dependence of the output on the parameter μ

The projection is on the impulse response vector, which is equivalent to the previous statement for the linear combiner (since the weight vector has the same values as the impulse response). The value of the PE output y(n) becomes here a function of the feedback parameter μ, since this parameter changes the impulse response as we saw above. When μ varies, the value of the output changes, which means that the relative position between the impulse response vector and the input vector changes with μ. So we conclude that the *feedback parameter has the ability to control how much of the past is retained at the context PE output.*

Conceptually this is similar to what we studied in Chapter IX with the linear combiner. When a desired response is available to the context PE, the value of μ can be adapted to minimize the cost, exactly as was done for the weights of the linear combiner. We will do this in the next chapter. But note that the context PE is restricted to a lowpass filter, while the delay line PE with the weights to the next layer may implement a FIR filter of arbitrary type. There are some more subtle differences. *Selecting μ effectively changes the size of the projection space and orientation of the projection*. projection with the context PE

Go to next section

6. Design of the memory space

As we have seen in the section that discussed memory PEs there are basically two types of memory mechanisms*: memory by delay and memory by feedback*. We seek to find the

most general linear delay operator (special case of the ARMA model) where the memory traces $y_k(n)$ would be *recursively computed* from the previous memory trace $y_{k-1}(n)$. This memory PE is the *generalized feedforward memory* PE (Figure 12). One can show that the defining relation for generalized feedforward memory PE is

$$g_k(n) = g(n) \bullet g_{k-1}(n) \qquad k \geq 1 \qquad \textbf{Equation 7}$$

where • is the convolution operation, g(n) is a causal time function, and k is the stage index. Since this is a recursive equation we have to provide independently a value for *$g_0(n)$*. In the Z-domain Eq. 7 reads

$$G_k(z) = G(z)G_{k-1}(z) = G_0(z)\prod_{i=0}^{K} G_i(z) \qquad \textbf{Equation 8}$$

where K is the number of delay stages in the cascade and we are denoting transforms by capital letters. This relation means that the next memory trace is constructed from the previous memory trace by convolution with the same function g(n), the memory kernel yet unspecified. Different choices of g(n) will provide different choices for the projection space axes.

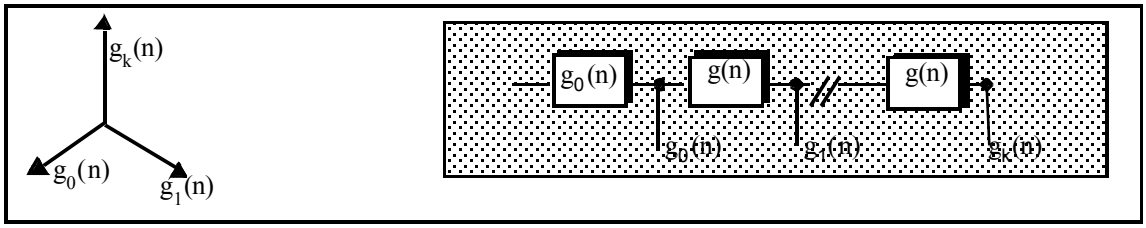


Figure 12. The generalized feedforward structure

When we apply the input x(n) to the generalized feedforward memory PE, the tap signals yk(n) become

$$y_k(n) = g(n) \bullet y_{k-1}(n) \qquad k \geq 2 \qquad \textbf{Equation 9}$$

the convolution of $y_{k-1}(n)$ with the memory kernel. For k=1 we have

$$y_0(n) = g_0(n) \bullet x(n) \qquad \textbf{Equation 10}$$

where g0(n) may be specified separately. The projection $\bar{x}(n)$ of the input signal is obtained by linearly weighting the tap signals according to

$$\bar{x}(n) = \sum_{k=0}^{K} w_k y_k(n)$$

Equation 11

and shown in Figure 13.

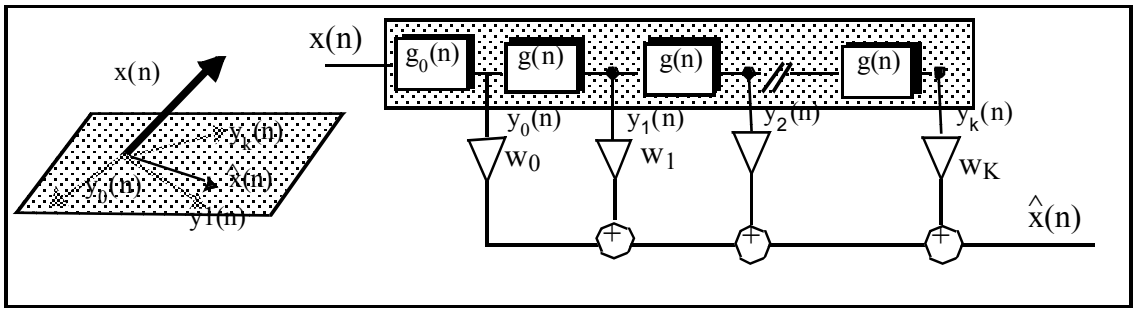


Figure 13. The memory filter created by the generalized feedforward memory PE.

The most obvious choice for the basis is to use directly the past samples of the input signal x(n), i.e. the k^{th} tap signal becomes $y_k(n)=x(n-k)$. From Eq. 9 it is easy to show that this choice corresponds to

$$g(n) = \delta(n-1)$$

Equation 12

and we are back to the delta function operator utilized in the tap delay line (in this case g0(n) is also a delta function $\delta(n)$). In this case the memory depth is strictly controlled by K, i.e. the memory traces store the past K samples of the input. TDNN uses exactly this choice of basis.

With *recurrent memory structures*, such as the linear context unit, the projection space basis becomes a processed version of the input. The information about the past of the input is contained in the memory traces, instead of simple storage of past input samples. The context PE equation in our notation is

$$y_0(n) = (1-\mu)y_0(n-1) + \mu x(n)$$

Equation 13

where 1-μ is the feedback parameter. It is easy to show that we can write

$$y_0(n) = \mu(1-\mu)^n \bullet x(n) \qquad \textbf{Equation 14}$$

When compared with Eq. 9 the context PE becomes a special case of a single tap (k=1) generalized feedforward structure with

$$g_0(n) = \mu(1-\mu)^n \qquad \textbf{Equation 15}$$

and $y_0(n)$, the memory trace, becomes a processed version through *convolution* of the input samples.

Go to next section

7. The gamma memory PE

The gamma memory PE is a special case of the generalized feedforward memory PE where

$$g(n) = \mu(1-\mu)^n \qquad n \geq 1 \qquad \textbf{Equation 16}$$

and $g_0(n)=\delta(n)$. Figure 14 shows the block diagram for the gamma memory PE. The gamma memory is basically a cascade of lowpass filters with the same time constant 1-μ. As can be seen from Figure 14, when the number of stages K=1, the gamma memory defaults to the context unit, and when the feedback parameter μ=1 the gamma memory becomes the tap delay line deVries . As a result, the gamma memory contains as special cases the context unit, and the tap delay line used in TDNN or in the FIR synapses. Hence, the gamma memory appears as a *unifying structure for the most common connectionist memories.*

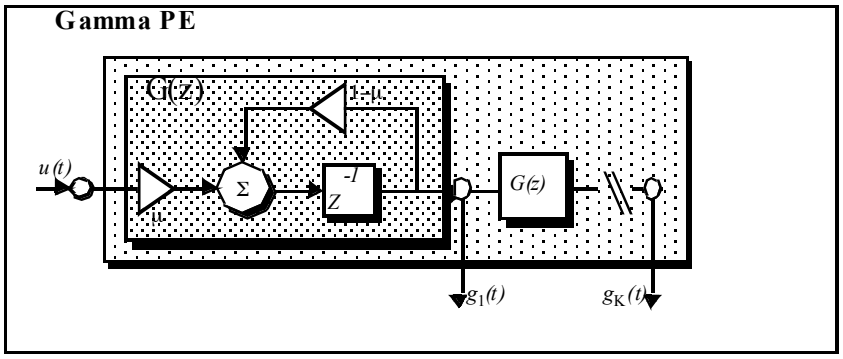


Figure 14. The gamma memory PE.

Figure 15 shows the family of impulse responses from the input to tap k in the gamma memory PE which is mathematically given by

$$g_k(n) = \binom{n-1}{k-1} \mu^k (1-\mu)^{n-k} \qquad n \geq k, k \geq 1$$

Equation 17

These functions are discrete versions of the integrands of the gamma function, hence the name given to the memory PE. They form a *complete basis in L_2 space* (i.e. one can approximate a finite energy signal arbitrarily closely as a weighted sum of these functions). In the context of memory structures, they can be interpreted as impulse responses of the memory PE from the input to the different taps. They peak at k/μ (k>1).

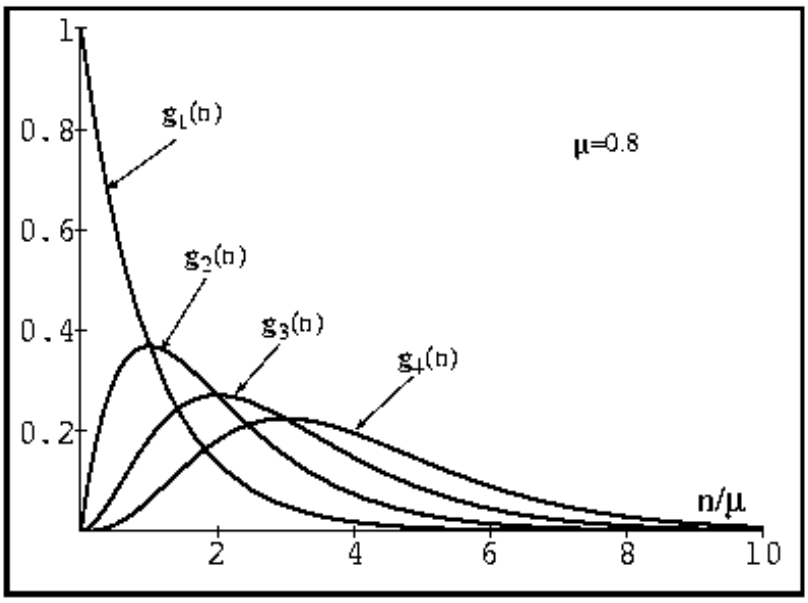


Figure 15. Impulse response of the gamma kernel from k=1 to k=4 for μ=0.7

In the frequency domain Eq. 17 can be alternatively written

$$G_k(z,\mu) = \prod_{i=1}^{k} \frac{\mu z^{-1}}{1-(1-\mu)z^{-1}}$$

Equation 18

So alternatively, the tap signals of the gamma memory can be interpreted as lowpass filtered versions (memory traces) of the input. The filters are all equal, with a pole at $z_p = 1-\mu$. So the transfer function from the input to the kth tap will have a pole of multiplicity k. This means that the memory traces for deeper taps correspond to progressively filtered versions of the input. So signals are progressively more delayed but also more filtered.

An interesting property of this family is that the time axis is scaled by the parameter μ, which means that there is a change in time scale from the input to the memory traces (uniform time warping). When this parameter is adapted with the information of the output mean square error, *the neural system can choose the best time scale to represent the input signal information*. The memory depth and resolution for the gamma memory are

respectively

$$D = \frac{K}{\mu} \quad and \quad R = \mu$$

Equation 19

The gamma memory has the great property of decoupling the memory depth D from the order of the memory K. This is a very important property. Just think that a given application requires a memory of 100 samples, but three parameters are sufficient to model the memory traces. A tap delay line will need 100 taps, and equivalently 100 parameters. This is a waste because we know only three are sufficient. During adaptation in a perfect world, the system must find out this and set to zero 97 weights. In reality noise may make all the parameters different from zero which will produce poor performance.

The context PE is also able to decouple the memory depth from the order, but unfortunately it can model just one of the modes of the input since it has a single parameter (projection to a one dimensional space). The gamma PE produces a K dimensional space to represent the input, which is better than a single line. The gamma PE is more versatile than the tap delay line PE because there is an extra parameter in the representation that controls the time axis scale. This means that the memory PE *can represent in K taps D samples into the past* (D>K), i.e. the dimensionality of the input space is reduced when the gamma kernel is used to build the memory filter. Hence the gamma PE normally leads to much more parsimonious representation of time information.

It is still easy to ensure that the gamma memory PE is stable by limiting the value of the feedback parameter μ to values $0<\mu<2$.

7.1. The gamma filter

The linear topology created by adding (weighted average) the taps of the gamma memory creates a very interesting filter called the gamma filter (Figure 16). The gamma filter has a feedforward topology but it is an IIR filter with a multiple pole at the location z

=1-μ. Hence the gamma filter is very similar to the linear combiner, in the sense that it creates a projection space of size equal to the dimensionality of the filter, and the output is the linear projection on the weight vector (which exists in the projection space). However, the gamma filter has several important advantages.

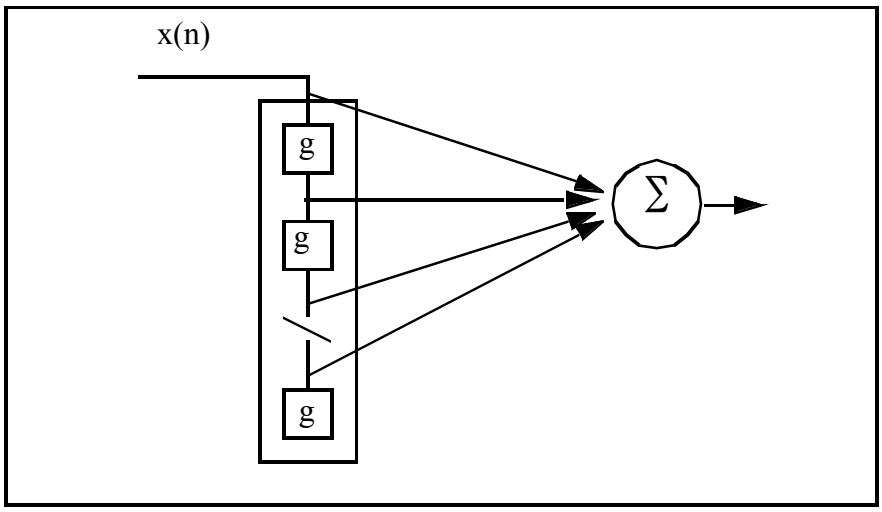


Figure 16. The gamma filter

The bases are no longer the input signal and its delayed versions, but are the convolution of the input with the gamma bases. They decouple the memory depth from memory resolution unlike the linear combiner. This has potential importance in system identification of real world system which tend to have very long (exponential decaying) impulse responses. In such cases, the gamma filter is able to provide a smaller fitting error than the linear combiner for a given filter size.

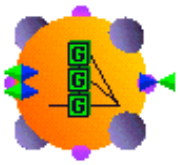
Moreover, the gamma filter has an extra parameter μ that controls the compromise depth-resolution given the selected number of taps. The output MSE becomes a function of the μ parameter, i.e. under the framework of adaptation, the gamma filter has the ability to choose the best compromise depth-resolution for the task at hand. It does this without any topological modification (as would be required with the tap delay line), simply by changing the feedback parameter. Unfortunately, the performance surface is non-convex so there is the potential that the search is caught in local minima. The adaptation

of the gamma filter feedback parameter can also be done with gradient descent learning, but this will be done in the next Chapter. gamma as an extension of Wiener filters

NeuroSolutions 10

10.10 Gamma filter for system identification

This example will present the system identification problem solved in Chapter IX with the linear combiner. As we have shown the delay line is a special case of the gamma filter when μ=1. So if we start with μ=1 we are mimicking the linear combiner. This is an important case because we can see what we gain by using the gamma memory instead of the more conventional delay line. The gamma memory PE is shown in the figure, and remember that now it has one parameter which is the feedback (μ) parameter.



Run the example with μ=1 and write down the error. Now enter different values of μ=0.8, 0.6, 0.4, 0.2 and see how the MSE changes. You will see that for this problem μ=0.4 provides the smallest error. The beauty of the gamma memory is that it allows the same topology to provide different projection spans, which is impossible to do with the tap delay line. In the case of the gamma memory, the bases are a convolution of the input with kernel g(n), while in the tap delay line they are simply the delayed versions of the input. Now we have the power of adapted weights to deal directly with the span of the time basis. Notice that for this problem the difference in terms of MSE is appreciable.

How can we, with the linear combiner decrease the error? The solution is to increase the tap delay line (the filter order). So we could alternatively try to solve the same problem with different filter orders until we find the best order. This is

very time consuming, and with the gamma filter we can do basically the same with the same number of taps by just changing the μ parameter. For this problem the required memory depth is (Eq. 18) 3/ 0.4 ~ 7 samples. So let us try an 7th order FIR and see the results. As we can expect this linear combiner works better, since it has more coefficients (which is not always better under noisy conditions as we recall from Chapter V). But the issue is that we have to guess a priori the size of the memory. Try a 6th order FIR and see that the results are much worse than the 7th order FIR.

Next chapter we will see how to adapt the μ parameter with the output MSE.

NeuroSolutions Example

Go to next section

8. Time Lagged Feedforward Networks (TLFN)

A time lagged feedfoward network (TLFN) is a feedforward neural network built from memory PEs and nonlinear PEs. *The short-term memory in TLFNs can be of any type and distributed in any layer*. The advantage of TLFNs is that they share some of the nice properties of feedforward neural networks (such as trivial stability), but they are *no longer static mappers*, i.e. they can capture the information present in the input time signals. In this section we will study a special case of TLFNs where the memory layer is constrained to be the input layer. These TLFNs are called *focused TLFNs* as we saw in the focused TDNN.

8.1. Focused TLFNs

In the focused TLFN the memory PEs are restricted to the input layer (Figure 17). The overall input-output mapping created by a focused TLFN neural network will be decomposed in two stages: a local time representation stage that is linear - the memory PE layer, and a nonlinear mapper between the representation layer and the decision (output) space which *will be static but nonlinear*.

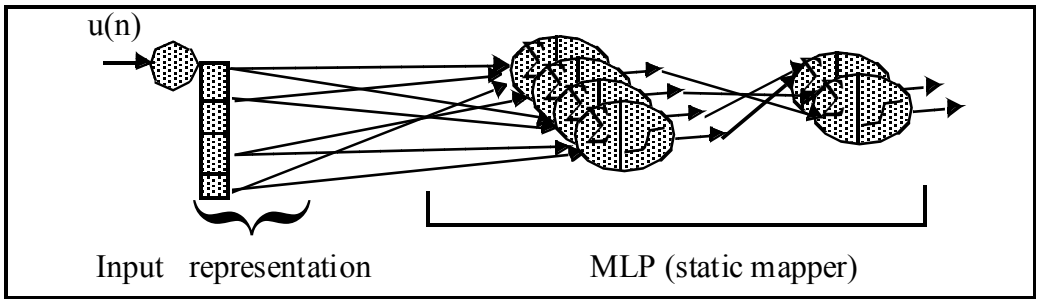


Figure 17. Single input focused TLFN

Let us analyze the function of this network when all the elements are trained to minimize the output mean square error (Figure 18).

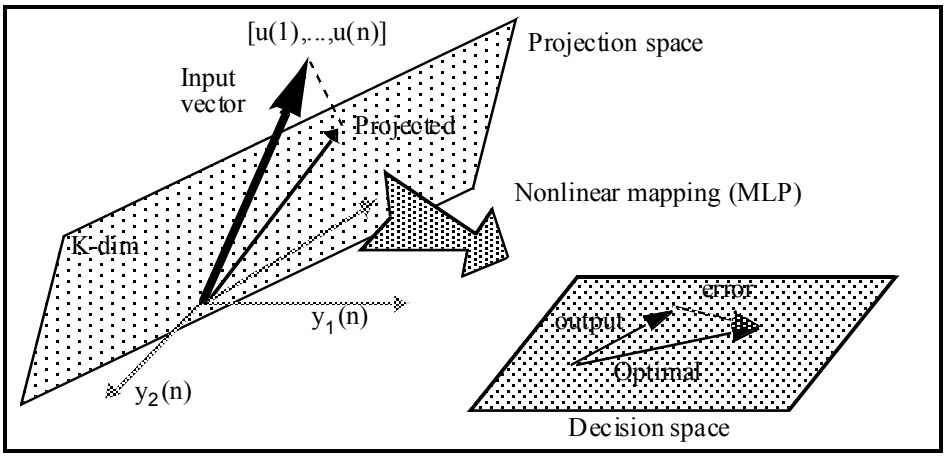


Figure 18. Projection of the input u(n) and the nonlinear mapping produced by the MLP.

We consider two stages in the network: the representation stage built by the memory filters, and the mapping stage built by the static nonlinear system.

The representation stage finds the best projection of the input signal into an intermediate *projection space* according to the goal of the application which is given implicitly by the error backpropagated from the output (in the context PE case the *size* of the projection space is also defined by the feedback parameter). Note that the backpropagated error is equivalent to *an implicit desired signal at the output of the representation stage*. So we can utilize the Kolmogorov interpretation to say that with the optimal weights, the output of the representation stage is the orthogonal projection of the input signal onto the

projection space. *The representation stage is exploring in an optimal way (through linear filtering) the spectral differences embedded in the time series patterns.* This projection space becomes the input space for the nonlinear mapper, i.e. the space where the discriminant functions are placed for optimal decisions.

In the TLFN architectures, the mapping stage is a feedforward neural network such as the MLP (or RBF). The MLP inputs are the outputs of the memory filters. *Hence, the space where the nonlinear mapping takes place has bases created by the hidden PE outputs which can be approximately interpreted as the outputs of the memory filters*. The number of bases is given by the number of memory filters.

The MLP will place its discriminant functions in the projection space to achieve the desired processing goal. Normally the goal is the minimization of an error in the decision (output) space. Note however that the weights of both the representation and mapping layers are trained together, so the user has no direct control on the projection space.

In this topology the dimension of the projection space (the number of memory filters) and the number of input PEs of the MLP are normally coupled together. There is no reason to couple the size of the space to the number of hidden PEs as we saw in Chapter III. So we can de-couple the number of axes from the number of hidden PEs by including explicitly in the topology the output of the memory filters as an extra layer of linear PEs. The input to the MLP will be a linear combination (fixed or adaptive weights) of these signals.

NeuroSolutions 11
10.11 Increasing versatility for TLFN architectures

We can separate the number of FIRs from the number of hidden PEs by including explicitly a layer of linear Axons and connect them to the MLP with a synapse (with fixed weights of 1).

With this arrangement we can show and control the effect of each part (representation or decision) in the mapping. The size of the decision space is

controlled by the number of memory filters, while the number of discriminant functions is controlled by the number of PEs in the hidden layer.

Remember that the scatter plot shows us the decision space. This is where the MLP is going to draw its decision surface. But the difficulty of the task is that the projections are also changing since the weights of the memory filters are being adapted. Overall the system will try to minimize the number of errors in the classification and move the projections such that they are easy to classify with linear discriminant functions. Notice however that all the "action" occurs around the origin. So the order of the filter counts as well as the number of PEs.

NeuroSolutions Example

8.2. Focused TLFNs as function approximators

Since Chapter V that we developed the idea that neural networks are function approximators. At that time we mentioned only the case of the linear regressor and the classifier as special cases. In Chapter IX we presented this same interpretation for the adaptive linear combiner. *Effectively the linear combiner is doing function approximation in the signal space,* and in this regard it is equivalent to the linear regressor.

Here we will revisit the same topic for the focused TLFNs. We submit that the previous perspective of dividing the function of focused TLFNs in signal representation and nonlinear mapping is very general and very productive. Applying this interpretation to the focused TDNN, we can immediately conclude that *the focused TDNN is an universal function approximator in the signal space*. In fact, the delay line produces a representation-preserving mapping onto the signal space (an embedding provided the size of the delay line is large enough - see Takens Embedding below), and the MLP is able to approximate arbitrary complex functions in the newly created signal space. Sandberg proved mathematically this statement. The demonstration basically states that the focused TDNN is all that is needed to approximate arbitrary time mappings that have

a finite region of support. Other TLFNs share the same property of the TDNN as we will discuss below.

The arbitrary mapping capability of the focused TDNN is very important because it makes this neural topology *highly suited for function approximation involving time signals*, extending to time the universal approximation results of the MLP and RBFs in vector spaces. As we discussed in Chapter IX function approximation is exactly what engineers call system identification. Therefore we can understand the reason why the focused TDNN outperformed the linear combiner for system identification and time series prediction. nonlinear system identification with neural networks

8.3. Focused TLFNs build from context PEs

When the memory of focused TLFNs is a delay line we obtain such an important case that it has been named the focused TDNN. We already discussed the focused TDNN and we have seen the power of such networks. But we saw that the tap-delay line was not the only type of memory PE. Let us examine the topology of Figure 19.

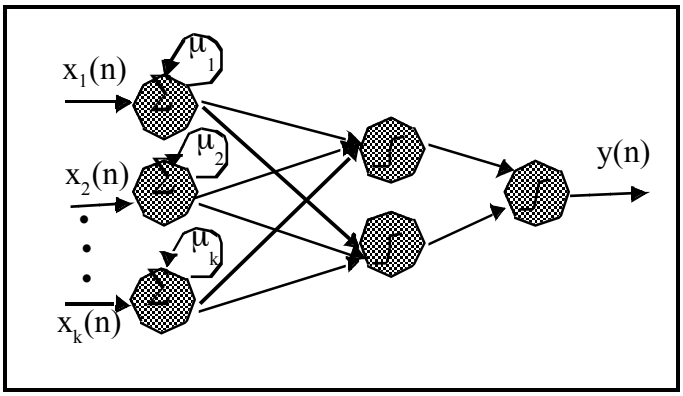


Figure 19. MLP with an input layer of context PEs (focused architecture)

The topology has an input layer of context PEs followed by a MLP. The first layer of input PEs can be also thought as a *representational layer for time information*. The context PEs remember the past values of the input indirectly because their output is a weighted average of the present and past inputs. The hidden layer PEs will nonlinearly combine this information to produce the desired output. This topology is also a *focused topology*

and implements a *static nonlinear function* of the present and past of the input.

The focused topology of Figure 19 is a recurrent neural network, but notice that the *recurrency is local to the PE.* One of the advantages of locally recurrent neural networks is that we can judge the stability of the system by constraining the value of the local feedback parameters such that the local PE is stable. *If local stability is enforced, the global system will be stable*.

The goal of the following examples is to solve with the focused TLFN built from context PEs the same three classes of problems that were presented for the focused TDNN. We will show that the TLFN produces acceptable results, although the context memory structure is less versatile than the linear combiner.

NeuroSolutions 12
10.12 Phoneme classification with focused TLFNs (context PEs)

This example solves the phoneme classification problem with a focused TLFN built from context PEs. Run the network and verify the performance. Change the context PE parameter and observe its effect on performance.

NeuroSolutions Example

NeuroSolutions 13
10.13 Nonlinear system identifiction with focused TLFNs (Context PEs)

This example is the system identification problem solved above with the TDNN. Here we will apply the focused TLFN to solve the problem. If we try the straight architecture with the default time constant of 0.9, we will find out that it fails. The reason is that for this system identification we have a single input, and so the context PE is limited to lowpass this input with a single time constant. The MLP is unable to grab the necessary information to find the mapping.

We can help the system if we provide several views of the input with different time constants, which can be easily done with a layer of context PEs with different feedback parameters. To implement this modification go to the Matrix Editor on

top of the context PE and enter values between 0 and 1 for each PE. Observe that in fact the focused TLFN now works as well as the TDNN. The lesson is that creating a space where signals differ in their frequency content is very useful for function approximation. We have seen in Chapter V that having more basis help the reconstruction. This is the same thing here. A single context PE gives you a single view of the trajectory. Many context PEs provide a multispectral view of the same trajectory, which captures much more information. This reasoning will be expanded later to design the memory space.

NeuroSolutions Example

NeuroSolutions 14
10.14 Nonlinear prediction with focused TLFNs (context PEs)

The next example will be the prediction of the Mackey-Glass signal using a focused TLFN built from context PEs. As in the previous example we have a single input, so we need to create a projection space to represent signals in time. We can do this by creating a layer of context PEs with different time constants.

Run the example and verify that for this problem the TDNN works slightly better. The context PEs are lowpass filters, so they smooth out the sharp discontinuities of the input signal. So the approximation suffers exactly at the sharp transitions (peaks and troughs). We can expect this from the discussion on memory depth and memory resolution. The context PE increases one at the expense of the other. So it is a compromise, but it is better than the delay line PE where there is no compromise (the delay line always uses the highest resolution).

NeuroSolutions Example

8.4. Focused TLFNs with gamma memories

The gamma memory can be used as the first layer of the focused architecture yielding another type of focused TLFNs which has been called the focused gamma neural network . Sandberg also showed that the focused gamma network is a universal function

approximator.

This network can be utilized with advantages over the focused TDNN. Experience has shown that for a focused TDNN with the same number of parameters the gamma network is able to provide a smaller output error in identification problems when the system response has a long region of support (as is the case of most real plants).

Another very interesting characteristic of the gamma neural network is that it has the capability to produce an uniform warping of the time axis. This means that the network is able to control the delay of its response with fixed weights by changing the recurrent parameter.

The following examples yield a one to one comparison of the TDNN with the gamma TLFN since the gamma memory for μ=1 defaults to a tap delay line. So we can immediately see in the same breadboard the change in performance between the two systems.

NeuroSolutions 15

10.15 Phoneme classification with a focused TLFN (gamma memory)

The first example deals with the phoneme classification. We have substituted the tap delay line by a gamma memory. Remember that for μ=1 defaults to the tap delay line, so we can obtain the performance of the TDNN when μ=1. Start with this value. Run the network which is set with a μ=0.5 for the value of the gamma memory. Notice that it runs very well. This is a great example to see how the network is assigning internal resources, and when one output is correct the others will modify themselves to "grab" its own desired response.

Modify the gamma parameter to 1 and see that now the system is more difficulty solving the task, and trains to a larger error. This tells us that longer memory depth is important to classify phonemes and it is more important than high resolution (which is provided by μ=1).

Just to see how well the network can do in this problem when optimized, let us substitute the output PE by a softmax nonlinearity. You have to decrease the stepsize to 0.05 and the momentum to 0.5. Run the network and see how perfect the classification is (remember that this is the training set, so this fact tells little about the overall accuracy in the task). The softmax cleans the outputs because it guarantees (competition) that the sum of outputs has to be 1.

In classification applications, the softmax should be the component of choice.

NeuroSolutions Example

NeuroSolutions 16
10.16 Nonlinear system identification with a focused TLFN (gamma memory)

The system identification example is also a very good test for the gamma TLFN. Most of the real world plants have long impulse responses since (if linear) they are IIR. We know that the TDNN has a memory that is FIR (the tap delay line), so for adequate modeling long memories are necessary. But long tap delay lines mean lots of parameters, which make the TDNN over parameterized and affect generalization. So the answer is to use generalized feedforward memories as the gamma. The gamma, due to the fact that it is globally FIR but has an IIR kernel, it can with the selection of a single parameter choose the appropriate memory depth. For the gamma D=K/μ, so for a fixed number of taps (K) we can increase D the memory depth by decreasing μ. Let us see how this helps for the nonlinear plant.

Start with μ=1 and register the final MSE. Then enter μ=0.7 and let the system adapt. Note that the error now is smaller and the waveform is smoother. Experiment with longer taps and find out that the improvement as a function of μ is not as apparent.

NeuroSolutions Example

NeuroSolutions 17
10.17 Nonlinear prediction with a focused TLFN (gamma memory)

Finally, we are going to show an example where the gamma memory does worse than the TDNN. Let us do nonlinear prediction on the Mackey-Glass time series. The waveform if you recall has lots of detail (high frequencies) mainly at the extrema (peaks and troughs). The gamma basis with small μ will emphasize depth at the expense of resolution. The MG30 does not require a long depth, but it needs the detail to follow the high frequencies riding on top of the waveform.

Let us start with a tap delay line of 5 and a μ=0.7. Register the error, and repeat the experiment with μ=1. You will see that the error is smaller for this case.

This does not necessarily mean that the gamma memory is worse for this case. Suppose that we reduce the memory to K=3 taps. Now with μ=1 the prediction is poor, since there is not enough depth. Now if we select μ=0.5, the error improves. So even for this application depth is important to discover the correct mapping.

One of the issues that you have noticed is that we have to enter manually the value of μ, and there is no guarantee that it is the best value that minimizes the MSE. We will see in the next chapter a way to adapt μ based on gradient descent using the output MSE. In such cases the gamma memory always provides the best compromise depth resolution.

NeuroSolutions Example

There are many other delay operators that can be defined to create TLFNs, each exploring differently the information contained in the input signal spectrum. Effectively each delay operator is establishing a different basis to project the time signal. more versatile memories

8.5. How to adapt the focused TLFN

How can the focused TLFNs be trained? We saw that the focused TDNNs can be trained with static backpropagation provided a desired signal exists at each time step. But notice that the TDNN does not have any parameter in the delay line, while other focused TLFNs

as the one in Figure 14 may have one parameter in the memory structures. *If we pre-select the value of the feedback parameter μ, static backpropagation* can still be utilized all the way through because the adaptive part of the network is static. Even if the feedback parameter is changed manually or automatically during operation, this effect is equivalent to a change at the input, so the gradient computation for the feedforward network weights is still static. This is in fact one of the advantages of the focused topologies and the reason for the name.

However, *the feedback parameter can NOT be adapted exactly by the backpropagation algorithm* discussed in Chapter III. We will address this issue in the next chapter.

A word of caution is in order at this moment. The focused TLFN is a very special topology. If the recurrent connections would occur in the *hidden layers even if the recurrent parameter was kept constant, static backpropagation could not be utilized to train the weights between the input and the location of the recurrent PEs.* The gradients will become time dependent, and the backpropagation algorithm covered in Chapter III is not prepared to handle them. The reason we can use static backpropagation to train focused TLFNs is because the recurrency exists at the input layer, i.e. there are no layers preceding it.

Go to the next section

9. Focused TLFNs built from RBFs

Radial Basis Functions (RBFs) are the other alternative to create universal mapping networks. RBFs where explained in Chapter V, and they can substitute the MLPs in the focused TLFNs. So this type of focused TLFN is built from a memory representation layer followed by a RBF, as shown in Figure 20.

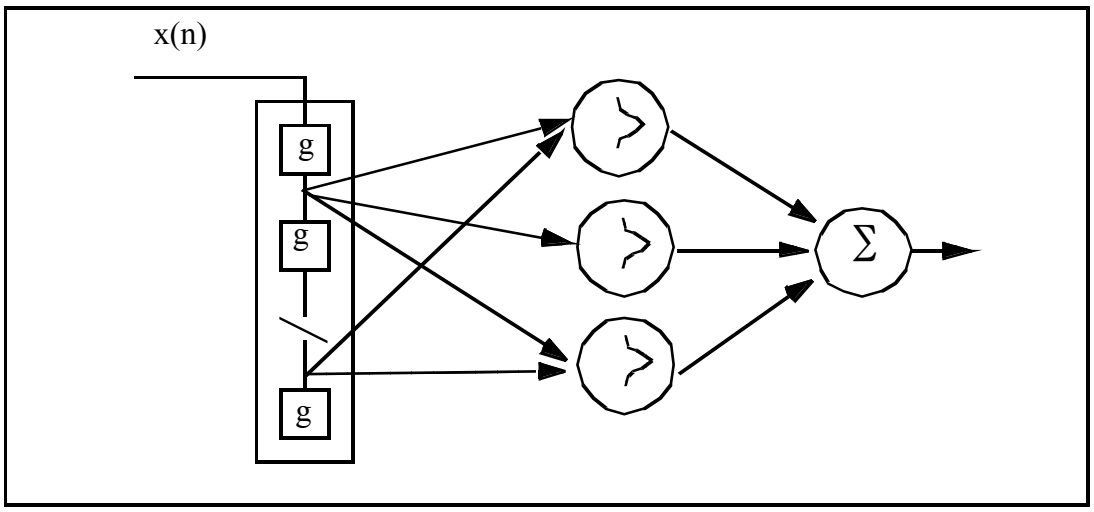


Figure 20. A focused TLFN built from a memory layer and a RBF network.

The idea of the topology is the same as for the TLFNs built with MLPs. The representation layer is producing the embedding from the time series to the signal space. In the signal space the RBFs will find the mapping to the desired response. Note however that the RBFs work directly in the signal space, without any projections, since there are no weights linking the memory taps and the RBF inputs. The RBF centers and variances should be adapted as discussed in Chapter V.

One potential advantage of the RBFs is that once the centers and variances are established the *adaptation is linear in the parameters*, so they normally train much faster than MLPs. However, one potential disadvantage is that one may need many RBFs when the signal space is large. For system identification many researchers do not adapt the centers and variances. Since the signal trajectory in general visits the full space (if the embedding dimension is correctly specified) placing the centers randomly or in a grid is commonly preferred.

We are going to exemplify the performance of TLFNs built from RBFs in the same three problems we encountered before.

NeuroSolutions 18
10.18 Phoneme recognition with a focused RBF-TLFN (gamma memory)

The first problem is temporal pattern recognition. We have the memory structure feeding now a RBF layer. Notice that the weights of this layer are solely utilized to establish the centers of the RBF network. Hence they are going to be trained in the unsupervised mode using competitive learning. Once the centers are established and the variances estimated, then the second phase of training is to find the weights that best meet the class specifications.

RBFs train very fast since the weights on the top layer receive directly the output error. The centers train also fast because the rule is local and unsupervised (no need to send the error through the structure).

Train the network and see that it does a good job of classifying the three phonemes. Modify the number of RBFs , and the number of delays to see the impact on the accuracy. RBFs must cover well the representation space, otherwise performance can not be good. Unlike the sigmoids, the Gaussians in RBF are local, so they just "see" a bit of the signal space. Hence we need many centers (here 100) and it is important to place then in areas that have data. Notice that with a delay line of 5 taps for 11 LPC coefficients we have a 55 dimensional space which is huge!!!…

This is the reason why the gamma memory may be very useful for these networks since it can cover a given memory depth with much fewer taps, decreasing the size of the signal space. Include a gamma memory instead of the delay line and see that the performance is normally better. .

NeuroSolutions Example

NeuroSolutions 19
10.19 Nonlinear system identification with a focused RBF-TLFN (gamma memory)

In this example we are going to solve the system identification problem for the nonlinear system of the TDNN example. But we are going to start with the gamma memory instead of the tap delay line to decrease the size of the signal space.

The network is basically the same as before, except that now it is a single input/single output system. We have also here the two phases of learning, first unsupervised to place the centers of the RBFs and to estimate the variances, followed by the adaptation of the output weights. Run the network and find out that it works as well or better than the TDNN case, with a much faster training. Now you understand why this topology is very attractive for real time applications. You can compare the performance of the memory kernel (gamma versus tap delay line) by selecting μ=1 (tap delay line).

NeuroSolutions Example

NeuroSolutions 20
10.20 Nonlinear prediction with a focused RBF-TLFN (gamma memory)

This example will implement a nonlinear predictor for the MG30 chaotic time series. We are also going to use here the gamma memory since it includes the tap delay line as a special case. The network is basically the same, here with 25 RBFs. We will train the network in two phases as before. Notice that the time series is well modeled even with few RBFs, but notice that the extrema of the waves lack detail. This is the price we pay by using the gamma memory, since we are trading in detail (resolution) by depth. In this case the fine detail of the time series requires more resolution than depth.

NeuroSolutions Example

Go to next section

10. Project: Iterative Prediction of chaotic time series

Neural networks are well suited for the prediction of time series produced by nonlinear models, which we will call here nonlinear time series. The idea is to extend the prediction model discussed in Chapter IX. The block diagram of the linear combiner is substituted

by a focused TLFN, trained with backpropagation in a single step prediction framework.

One important consideration is to choose appropriately the size of the projection space. In the case of prediction of nonlinear time series produced by deterministic systems there is an important theorem that can be used to help in this definition, the Takens Embedding Theorem Takens Embedding Theorem . Basically the theorem states that we have to find the size of a reconstruction space where the trajectories do not cross. We can estimate the dimension of this space, but the algorithms are difficult to work with.

As an alternative we can experiment with the size of the delay line and the value of the delay (i.e. multiple delays) using the simulator, as we experiment with the size of the MLP topology in Chapter III. The results of nonlinear predictors frequently outperform the prediction with linear models, but both yield reasonable approximations.

Prediction can also be thought as the first step towards dynamic modeling, where the overall goal is exactly to identify the dynamical system that created the time series. Once the predictior is trained, a dynamical model can be obtained by feeding back the output into the input (delayed by one sample). This creates an autonomous system as Figure 21 shows. Can the above prediction methodology create a good dynamical model? Our model is considered good if it can create in autonomous mode a time series with the same characteristics as the original time series (see Principe).

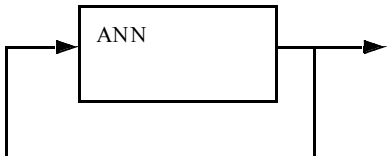


Figure 21. Iterative prediction for dynamic modeling

When we attempt to use the linear system in autonomous mode, the results are very disappointing, since the output quickly approaches zero (or oscillates). With the focused TDNN, the response has many other options, but hardly ever resembles the original time series. So how can we design improved dynamical models?

A very important modification to the overall methodology of signal prediction is to

introduce the concept of iterative prediction. Notice that until now we are always using the time series as the input to the neural network. However, when we want to generated the time series from our trained model we are using the most current output of the system. We can mimic this during training by feeding back the output of the predictor back to its input (and disconnect the input time series). This arrangement is called iterative prediction. We first seed the predictor with one (or more) point in state space, compute the output and feed it back to the input for a number of samples. At the same time we know the desired response, so we can create an instantaneous error and adapt the model parameters. After a certain number of iterations we should reset the process, since the model output and the time series will differ more and more (at least in the beginning of training).

Implementing this scheme normally produces much more accurate dynamic models. Try this in NeuroSolutions and experiment with the parameters to see the effect in the training MSE and in the quality of the generated time series.

NeuroSolutions 21
10.21 Iterative prediction with focused TLFNs with feedback

In this example we will detail a little more the problem of modeling a chaotic time series. The reason we think this is important is that neural networks are posed to play an important role in this area. What we are going to demonstrate is that the TLFNs not only can predict the chaotic signal as we saw in the previous examples, they can learn the dynamics from the time series and produce a system that can generate the SAME signal. In a sense they have identified the parameters of the system that produced the time series and they can substitute the original dynamical system.

Signal generators are important and the most widely used in engineering are the oscillators. But a linear system can not generate autonomously waveforms more complex than a sinusoid. This is due to the fact that the transient dynamics of a linear system either go to a point attractor or a limit cycle. The limit cycle produces

exactly the sinusoid.

Let us take now the MG30. This is a very complex waveform. How can we build a system that produces such a waveform? There is a trivial way: input white noise into a linear system and keep on adapting the parameters. This will provide a waveform very similar to the MG30, but notice that the source of complexity is in the WHITE NOISE input. White noise contains potentially all of the wavforms in the world. So by filtering we can obtain the one we want.

The second way to generate the MG30 (the difficult one) is to use a system WITHOUT an external input. Such a system is called autonomous. Just feed the output of the system to itself and we have a signal generator. Can we design an autonomous system that produces the MG30? We can immediately rule out linear systems, since they are able to generate limit cycles or the output decays to zero. Hence we must have a nonlinear dynamical system.

It turns out that dynamic neural networks can be taught to generate complex (even chaotic) time series such as the MG30. This is what we would like to demonstrate in this example.

We will start with the liner combiner. We can train it to a very small prediction error, but when we fix the weights and feedback the output to the input the output decays to zero (or to a limit cycle). The autonomous dynamic regimes of a linear system are not very exciting…..

Now let us do the same thing for the TDNN. We train it to a very small prediction error. Then we fix the weights and feed the output back to the input. Voila! The output resembles the MG30. In fact the error (compared with the signal it has been trained with) is very small up to sample 50 and then the error increases (and depending upon the training may go back down). But visually speaking, the two waveforms are very similar. Let us take a FFT of each to see that their spectra almost exactly match. So we have created a system that behaves just like the

MG30.

The big attractive of the method is that we never used the information about the equation. We just picked a time series and used it to train the TDNN. So in principle this technique can be used to create synthetic models from real world signals…. This application is called dynamic modeling.

Experiment with the size of the TDNN and see when it fails to produce a waveform similar to the MG30.

NeuroSolutions Example

Go to next section

15. Conclusions

This chapter brought together the two fundamental concepts of nonlinear topologies and time which were independently developed in Chapter III and Chapter IX respectively. The MLP was a very powerful mapper but it only worked with static patterns. On the other hand, the linear combiner was able to work with time signals but its mapping ability is reduced since it can only perform linear approximations. Combining a memory structure with the MLP gave rise to the time lagged feedforward network (TLFN) class of dynamic networks.

TLFNs are very powerful mappers, and have the advantage that can be trained with static backpropagation provided that the memory layer is placed at the input, a topology that is called focused. However, static backpropagation can NOT train the eventual feedback connection in focused TLFNs, so it has to be manually selected. Next Chapter will deal with the training of TLFNs and fully recurrent systems.

This chapter introduced the concept of the memory PE as a special PE configured to process time signals. In a sense the memory PE brings the representation of time to the inside of the neural topology, with the advantage of an internal (and hopefully better)

control on time representations. We quantified the properties of memory PEs and gave the interpretation of the memory PE and its connection weights to the other network PEs as a filter.

We also presented the two fundamental connectionist memory mechanisms (memory by delay and memory by feedback), and showed that the gamma memory is an unifying principle in this perspective. However, the gamma memory can still be extended to other memory mechanisms that extract other information from the time series (the gamma II memories). So Chapter X is a very important Chapter in the book because it opens up the possibility to use nonlinear systems for time processing. The major hurdle that we still need to conquer is how to train arbitrary topologies built from memory PEs and nonlinear PEs.

NeuroSolutions Examples

10.1 Time disambiguates data clusters

10.2 Disambiguate data with the tap-delay line

10.3 Temporal pattern recognition with the focused TDNN

10.4 Nonlinear system identification with the focused TDNN

10.5 Nonlinear prediction with the focused TDNN

10.6 Context PEs revisited

10.7 Memory depth of tap-delay line

10.8 Memory depth of the context PE

10.9 Interpretation of focused TDNNs as nonlinear memory filters

10.10 Gamma filter for system identification

10.11 Increasing versatility for TLFN architectures

10.12 Phoneme classification with focused TLFNs (context PEs)

10.13 Nonlinear system identification with focused TLFNs (context PEs)

10.14 Nonlinear prediction with focused TLFNs (context PEs)

10.15 Phoneme classification with a focused TLFN (gamma memory)

10.16 Nonlinear system identification with a focused TLFN (gamma memory)

10.17 Nonlinear prediction with a focused TLFN (gamma memory)

10.18 Phoneme recognition with a focused RBF-TLFN (gamma memory)

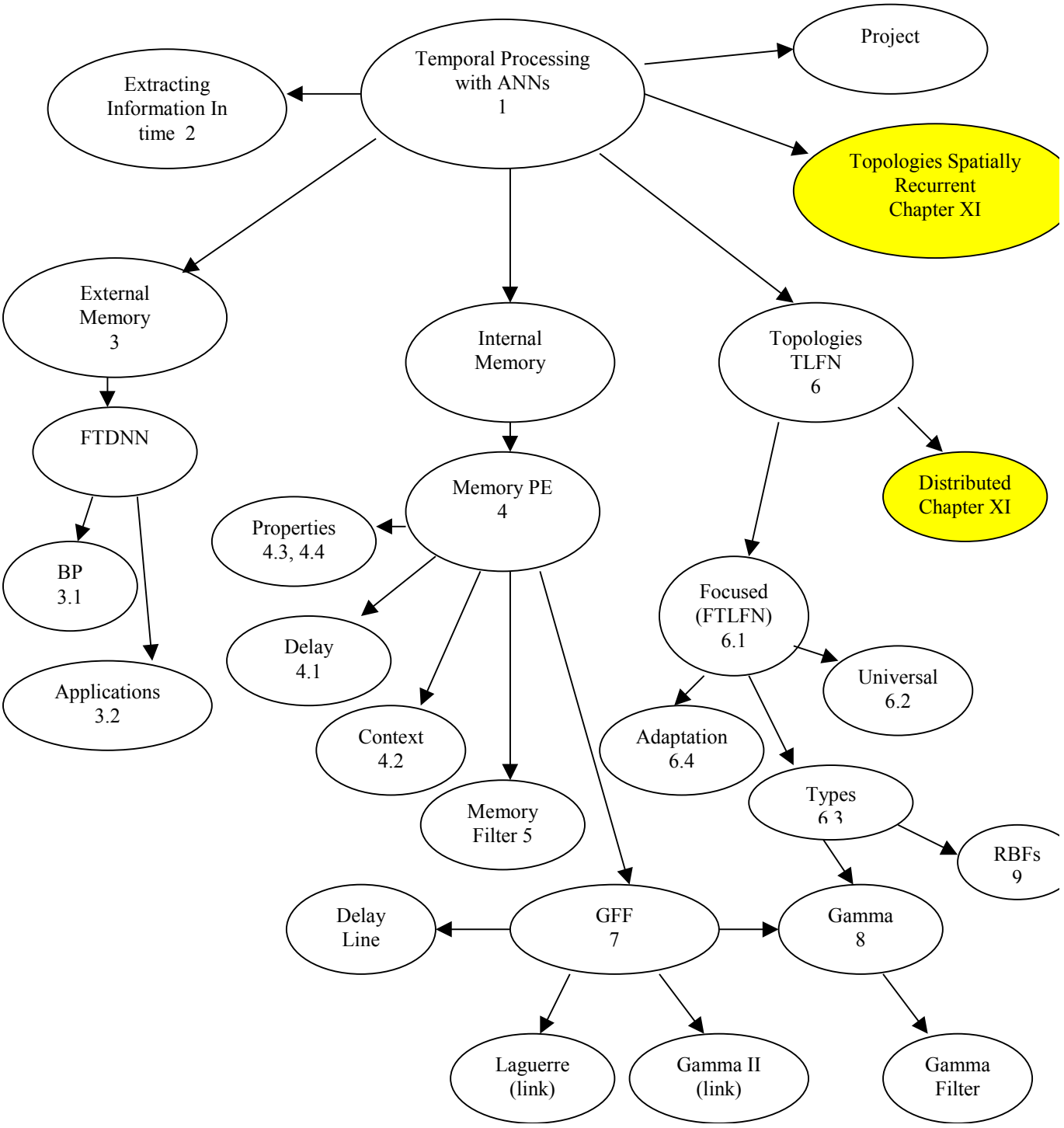
10.19 Nonlinear system identification with a focused RBF-TLFN (gamma memory)

10.20 Nonlinear prediction with a focused RBF-TLFN (gamma memory)

10.21 Iterative prediction with focused TLFNs with feedback

Concept Map for Chapter X

Chapter X


Temporal Processing with ANNs 1
Project
Extracting Information In time 2
Topologies Spatially Recurrent Chapter XI
External Memory 3
Internal Memory
Topologies TLFN 6
FTDNN
Memory PE 4
Distributed Chapter XI
Properties 4.3, 4.4
BP 3.1
Delay 4.1
Focused (FTLFN) 6.1
Universal 6.2
Applications 3.2
Context 4.2
Adaptation 6.4
Types 6.3
Memory Filter 5
RBFs 9
Delay Line
GFF 7
Gamma 8
Laguerre (link)
Gamma II (link)
Gamma Filter

Go to Next Chapter
Go to Table of Contents

short-term versus long-term memory

What is the computational difference between short and long term memory? This is a critical and difficult question that has not been properly studied. Static mappers as the MLP have long term memory, since the information utilized during system training is converted into weight values using the learning rules. So static mappers contain a repository of past information which we associate with memory. However, they are unable to differentiate time relationships because the information collected through time is collapsed in the weight values.

Dynamic systems are different. Due to the fact that they have short-term memory structures or recurrent connections they are sensitive to the sequence of presentation of information. They also have weights so they also have long-term memory. But unlike static mappers these weights code differences within the time window of observation.

Return to Text

static versus dynamic modeling

It is important to understand the source of the difference between the two types of systems. We will give an example. Suppose that we have two clusters of points with the same mean and variance but which belong to two different classes. Training an MLP with MSE to distinguish these two classes is virtually impossible. However, let us assume that the two classes have different time histories, i.e. that they are created by two sinusoidal signal sources of different frequencies. It is obvious that we may separate the two

classes with a linear combiner, but we will never be able to separate them with an MLP. This is an extreme case to illustrate the difference between static versus dynamic modeling. In static modeling we are using statistical properties of the data clusters to distinguish them. In time phenomena, time imposes a structure in the input space (see previous Chapter) that may be used to separate data clusters with overlapping statistics, i.e. the sequence in which the points are visited is different, and this difference can be used for separation.

In many practical cases one does not know if there is a time structure underlying our data. And so, it is not clear if dynamic modeling will help to improve performance. The trouble (as we will see next) is that the window in time where we seek the information is crucial to improve the results.

Return to Text

Memory depth

The impulse response of the context PE is

$$h(n) = \mu(1-\mu)^n$$

Its Z transform as we saw in Chapter VIII is

$$H(z) = \frac{\mu}{z-(1-\mu)}$$

Now the definition of the memory depth in the time domain can be converted to the frequency domain as the derivative with respect to z of the system transfer function evaluated at z = 1, i.e.

$$D = \sum nh(n) = -z\frac{dH(z)}{dz}\bigg|_{z=1} = \frac{1}{\mu}$$

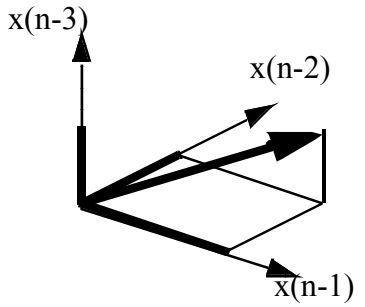
Return to text

projection with the context PE

The projection operated by the context PE can not be analyzed in finite dimension vector spaces since the impulse response is infinite. However, we can approximately study it taking into consideration that there is an “effective” finite memory depth given by $1/\mu$. Using this approximation the context PE can be interpreted as a projection on a linear manifold of size int($1/\mu$).

The *number of degrees of freedom* of this projection is nevertheless very different from the delay line. In the delay line the basis are the past values of the input. The output is constructed by weighting the values of each bases with the filter coefficients to obtain a direction. One *has as many coefficients as bases* to choose the direction of the projection.

In the case of the context PE, a similar model can be used to analyze the projection operated by the system. But the details are different. The bases can still be considered the previous samples but now the *weighting is not independent from axis to axis*, it is fixed by the value of the feedback coefficient. The most recent past sample is weighted by $1-\mu$, two samples in the past by $(1-\mu)^2$, etc. So, there is no flexibility to choose independently the values of the filter coefficients. This means that the projection vector is always biased towards the first basis, where the weights are larger. Obviously this limits the projection directions that can be created with the context PE. The context PE is able to create projections that are only lowpass filters of the input.



The projection vector always exists in the first quadrant. The direction approaches the

bisector of the first quadrant when μ approaches 0, and the projection space dimension is very large (since the effective dimension is $1/\mu$). So the context PE is practically equivalent to the linear combiner 1,1,……. which we identified before as a lowpass filter.

In the other extreme of μs close to 1, the projection approaches the first projection axis (since the largest weight is associated with the sample n-1, and the geometric ratio is close to zero), which is able to represent the input without error in large dimensional spaces, but the size of the projection space is very small since it is 1-D (the present sample). Hence, there is a clear trade-off between the size of the projection space and the resolution of the representation.

The conclusion of this analysis is the following. With the context PE we do not need to consider the projection space, since we control the location of the projection vector directly (i.e. this system is single-input single-output). However internally we still can analyze the projection as in the case of the linear combiner, but now there is only one free parameter that controls at the same time the direction of the projection and the effective size of the memory space. The context PE is only able to create lowpass projections. Another interesting characteristic is that the size of the projection space is controlled by the free parameter.

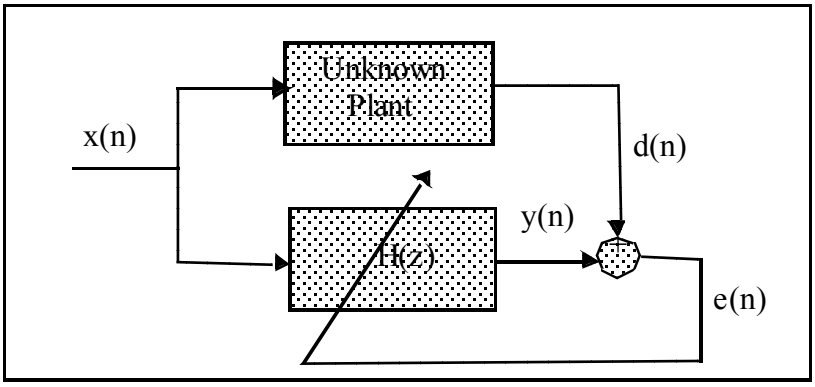
Return to Text

nonlinear system identification with neural networks

System identification is a very important application in engineering. Very often one is interested in modeling the input-output relationship of an unknown plant. We saw in Chapter IX that this problem is a practical application of function approximation. In fact, we can consider the output of the system d(n) as a fixed but unknown function of x(n), i.e. d(n)=f(x(n)). The role of the model is exactly to approximate the function f(.) by using a

set of basis that are defined by the topology of the model system.

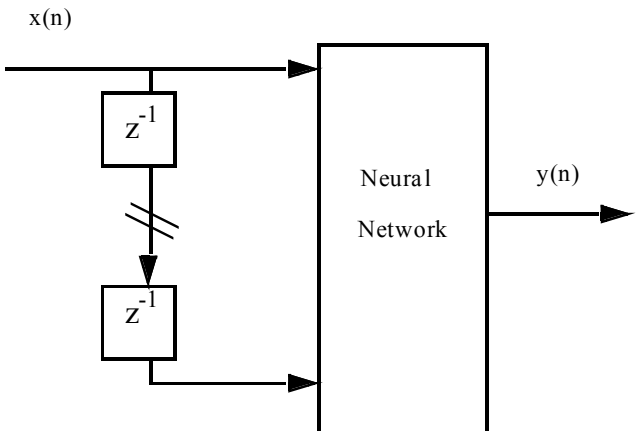
The linear combiner implements a set of linear bases, so we are effectively doing function approximation with a linear system that is MA (moving average). We covered other possibilities in Chapter IX but we restricted our attention to linear bases. In this chapter we are going to relax this constraint and work with nonlinear systems, of the TLFN class. With TLFNs we can basically utilize the same block diagram (Figure 24 of Chapter IX is copied below for reference) but develop nonlinear models of the unknown plant.



In nonlinear moving average modeling, the output of the model is a nonlinear function of its input, i.e.

$$y(n+1) = f[x(n), x(n-1), \ldots, x(n-k+1)] \qquad \textbf{Equation 20}$$

This equation basically states that the model is feedforward. So a TDNN is effectively a NMA model of the plant when it is used as the model in the Figure above. The TDNN is a nonlinear version of the linear combiner, and an alternative to Volterra models (which are also NMA).



In nonlinear autoregressive models (NAR) the output of the model is given by

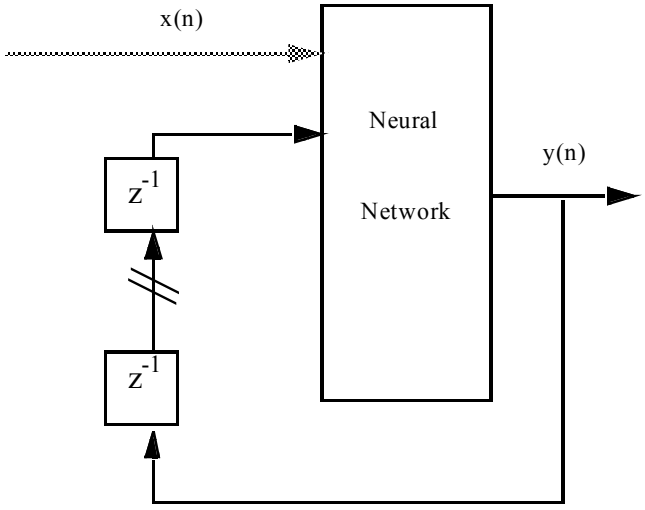
$$y(n+1) = f[y(n), y(n-1), \ldots, y(n-k+1)] \qquad \textbf{Equation 21}$$

In this equation the next output is simply a function of the past values of the output. This type of models is effectively used in prediction where we train a NMA model but due to the block diagram form of prediction we are fitting a NAR model to the data.

In general we use a NARX (nonlinear autoregressive with external input) model (also called an ARMA model with single input) for system identification, i.e.

$$y(n+1) = f[y(n), y(n-1), \ldots, y(n-k+1), x(n)] \qquad \textbf{Equation 22}$$

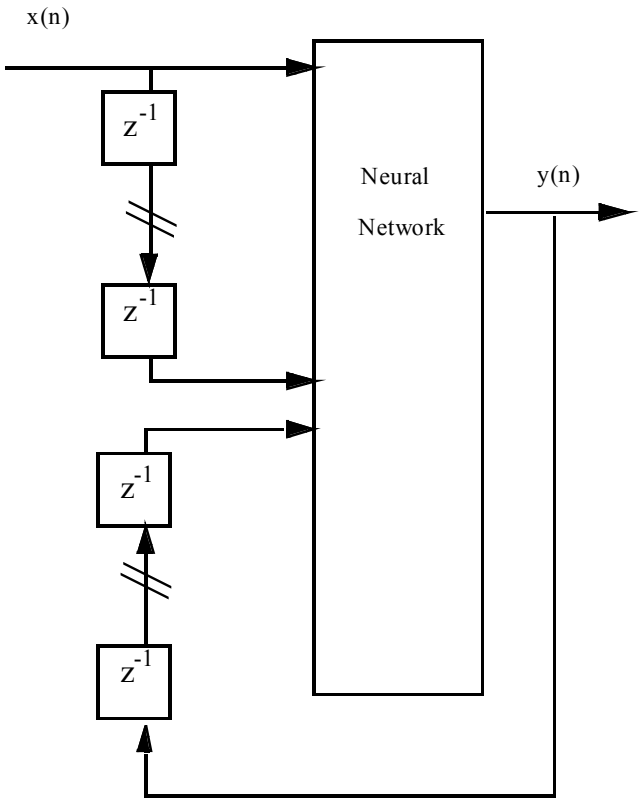
i.e. we drive our TLFN with a signal (also going to the plant) and we use its output back at the input (global feedback).



The nonlinear autoregressive moving average (NARMA) also called the NARX model is the most general class of nonlinear models and it is a blend of the two previous types, i.e.

$$y(n+1) = f[y(n), y(n-1), \ldots, y(n-k+1), x(n), x(n-1), \ldots x(n-j)] \quad \textbf{Equation 23}$$

i.e. we drive the TLFN with its past outputs (global feedback) and the input and its delayed versions.



A fundamental aspect of system identification is to understand when a given model should be utilized depending upon the knowledge that we have about the plant. In terms of the function approximation paradigm what this means is how can we chose appropriately the basis. A related problem is how are we going to parameterize our nonlinear models. Once again the NARMA (Eq. 23) seems a very general parameterization, but in some cases simpler but easier to train parameterizations may exist.

Return to text

more versatile memories

There are a many linear operators fitting our definition of generalized feedforward memory PE that can be used as short term memory mechanisms for neural networks. We will review here two useful cases, but the choice of the most appropriate memory PE for a given application remains an open question.

A set of bases intimately related to the gamma functions is built with the Laguerre functions. The Laguerre functions are an orthogonal span of the gamma space, which means that the information provided by both memories is the same. The z- transform of the Laguerre functions is given by

$$L_i(z,\mu) = \sqrt{1-(1-\mu)^2}\,\frac{\left(z^{-1}-(1-\mu)\right)^{i-1}}{\left(1-(1-\mu)z^{-1}\right)^{i}} \qquad i = 1,2,\ldots$$

Equation 24

One can show that the Laguerre functions can be recursively computed, following the definition of the generalized feedforward structures given in Eq. 7 . In fact the Laguerre PE is a cascade of a lowpass filter with impulse response

$$g_0(n) = \sqrt{1-(1-\mu)^2}\,(1-\mu)^n$$

Equation 25

followed by a cascade of all-pass functions with a kernel given by

$$g(n) = \mu(1-\mu)^n + \mu(1-\mu)^{n+1}$$

Equation 26

It is easier to write the equations in the Z domain, as

$$L_i(z,\mu) = G_{i-1}(z)\,\frac{\sqrt{1-(1-\mu)^2}}{1-(1-\mu)z^{-1}} \qquad G_{i-1}(z) = \prod_{j=1}^{i-1}\frac{z^{-1}-(1-\mu)}{1-(1-\mu)z^{-1}}$$

Equation 27

It is interesting to analyze the difference between the Laguerre and gamma memory traces: the gamma memory PE will attenuate the signals at each tap because it is a

cascade of leaky integrators with the same time constant. However, the Laguerre memory PE has a frontend lowpass filter, but the next stages place a zero in the mirror location (w.r.t. the unit circle) of the pole of the previous stage, and includes a pole of its own in the same pole location. Effectively this topology is creating a basis using a *Gram-Schmidt orthogonalization of the tap signals* after the lowpass filter, which we know produces an orthogonal basis when the input is an impulse (or white noise). So the Laguerre PE will not attenuate the tap signals in amplitude but will delay the signal components. This leads to a set of basis functions that are oscillating. Figure 21 compares the response of the Laguerre and the Gamma PE to an impulse, which produces the set of Laguerre and gamma basis respectively.

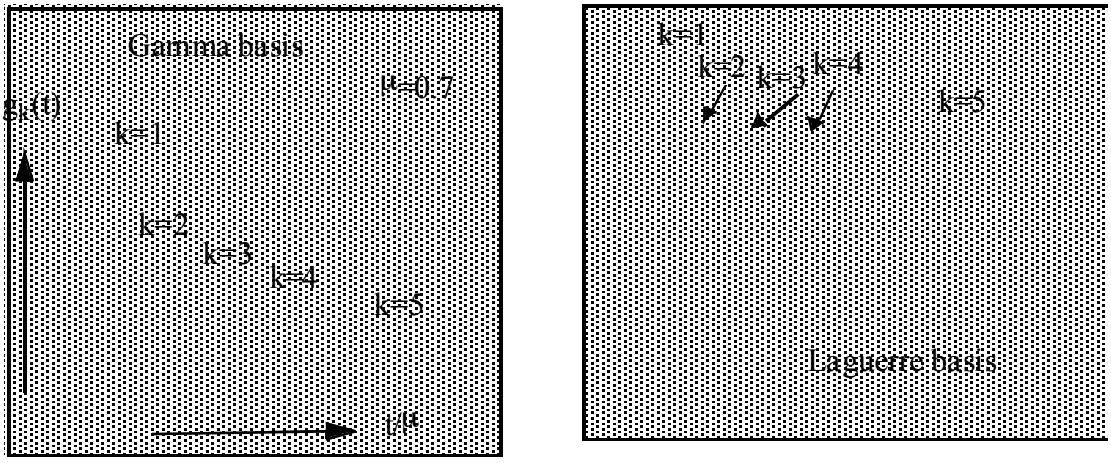


FIGURE 21. **Comparison of the Laguerre and the gamma basis**

The advantage of the Laguerre is that the tap signals (the bases) are obtained by convolving the lowpass filtered input with an orthogonal set of function (the allpass functions), so the basis is less correlated and the adaptation speed becomes faster specially when μ is close to 0 or 2. The Laguerre memory is still very easy to compute and has only one free parameter μ as the gamma memory. The allpass function is the closest realizable analog circuit implementation to the ideal delay operator.

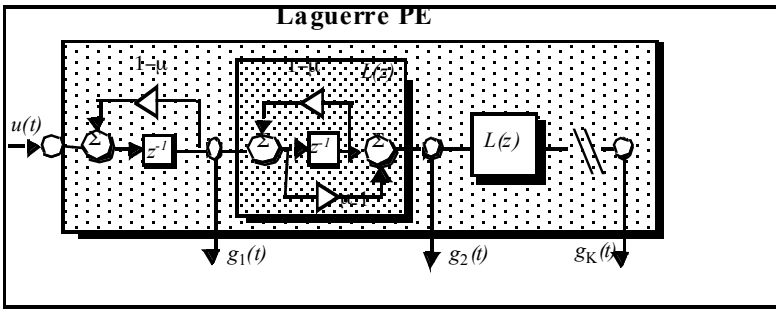


FIGURE 22. **Block diagram of the Laguerre PE.**

The gamma memory PE has a multiple pole that can be adaptively moved along the real Z domain axis, i.e. the gamma memory can only implement lowpass ($0< \mu <1$) or highpass ($1 <\mu<2$) transfer functions. The highpass creates an extra ability to match the prediction by alternating the signs of the samples in the gamma PE (the impulse response for $1< \mu <2$ has alternating signs). But with a single real parameter the adaptation is unable to move the poles to complex values. Two conditions come to mind that require a memory structure with complex poles. First, the information relevant for the signal processing task appears in periodic bursts; and second, the input signal is corrupted by periodic noise. A memory structure with adaptive complex poles can successfully cope with these two conditions by selecting in time the intervals where the information is concentrated (or the time intervals where the noise is concentrated). Figure 23 shows one possible implementation for the Gamma II kernel.

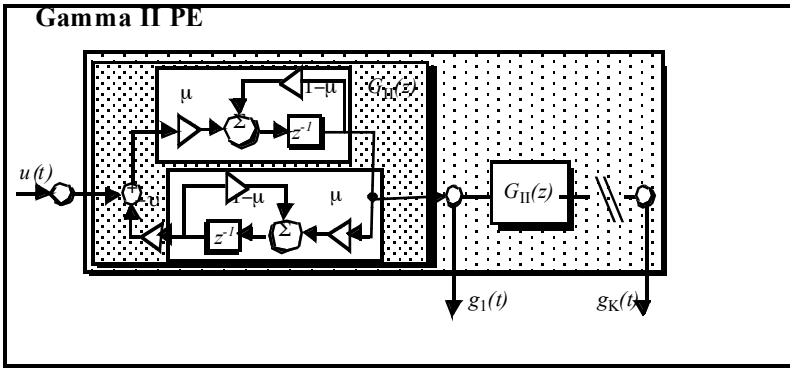


FIGURE 23. **Implementation of Gamma II**

In the Z domain the transfer function of the Gamma II kernel is

$$G_{II}(z) = \frac{\mu\left[z - (1-\mu)\right]}{\left[z - (1-\mu)\right]^2 + \vartheta\mu^2}$$

Equation 28

which corresponds in the time domain to a kernel given by

$$g(n) = r^n \cos(w_0 n) \qquad r = \sqrt{(1-\mu)^2 + \upsilon\mu^2} \qquad w_0 = \tan^{-1}\frac{\mu\sqrt{\upsilon}}{1-\mu}$$

Equation 29

Notice that for stability, the parameters μ,υ must obey the condition $0 < \mu(1+\upsilon) < 2$.

Complex poles are obtained for $\upsilon > 0$.

In terms of versatility, the Gamma II has a pair of free complex poles, the Gamma I has a pole restricted to the real line in the Z domain, and the tap delay line has the pole set at the origin of the Z domain (z=0). A multilayer perceptron equipped with an input memory layer with the Gamma II memory structure implements a nonlinear mapping on an ARMA model of the input signal. Table 1 shows a comparison of the characteristics of the memory PEs.

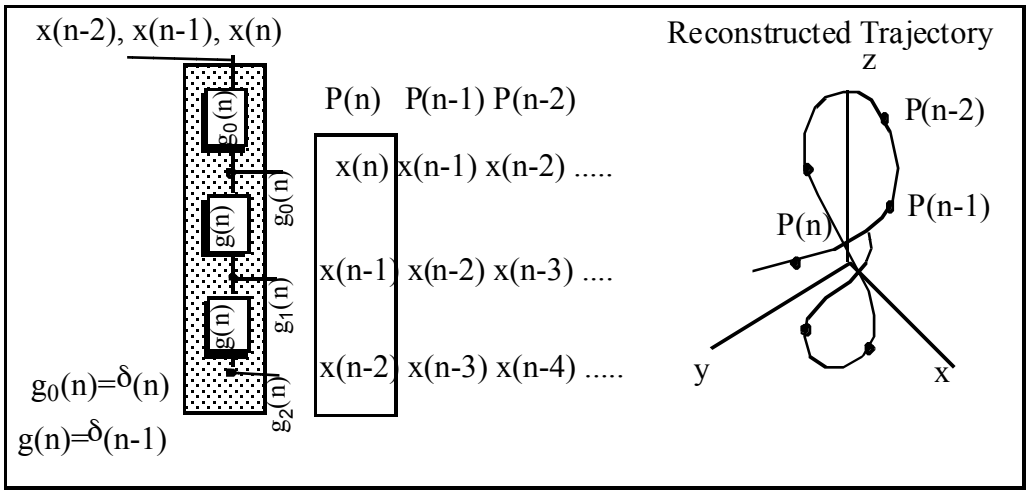
	tap delay	gamma	Laguerre	gamma II
depth (k taps)	k	k/μ		
resolution	1	μ		
stability	always	0<μ<2	0<μ<2	0<μ(1+υ)<2
filter	allpass	lowpass	low/allpass	bandpass
pole	0	1-μ	1-μ	r,w0

Return to Text

Takens Embedding Theorem

Takens proved that some properties (dynamic invariants) of the system that produced the time series can be preserved if we transform the time series into a sufficiently large

reconstruction space (N, the size of the space, should be at least 2M+1 where M is the number of degrees of freedom of the dynamical system). He proposed that the point coordinates of the reconstructed trajectory be read as N-tuples of the times series. For instance, for a reconstruction in a 3D space, consecutive 3-tuples of the time series should be read at a time, with the first time series sample being the x coordinate of the first point in the reconstruction space, the 2nd sample the y coordinate, the 3rd sample the z coordinate, the 4th sample being the x coordinate of the second point etc. (see Figure below). When the points of the reconstructed space are connected, a trajectory is found from which properties of the original dynamical system can be estimated (such as dimension and Lyapunov exponents). This means that some of the important dynamical properties of the original signal are preserved in this reconstruction.



What is interesting is that the Takens embedding is naturally implemented by generalized feedforward structures. In fact a TDNN with 3 taps provides x(n), x(n-1) and x(n-2), which are exactly the coordinates needed to reconstruct the points of the trajectory in the reconstruction space.

This alternative view of memory structures is very enlightening, because we can see that the PE that receives the output of the memory filter is effectively working with the trajectory in state space, i.e. it is accessing the dynamical information. So at least as a first approximation this methodology helps us set the size N of the memory filter as twice the size of the dimension of the dynamical system that produced the time series.

The dimension of the dynamical system can be estimated from the time series using for instance the *correlation dimension algorithm*. But unfortunately, there is still a free delay parameter τ, in Takens embedding theorem that is unspecified. Its function is basically to uncorrelate as much as possible the samples in time. If t is not set right the training can be very slow. One can experiment with the selection of τ by changing the number of delays between the tap outputs.

Note that the Takens embedding theorem is a more mathematical formulation of the ideas of signal space and reconstruction space that were presented in Chapter IX. It shows that the signal trajectory can preserve information about the dynamics of the system that produced the time series.

Return to Text

dynamic systems

systems with outputs that depend upon more than one time instant

static mappers

systems where the output is dependent upon a single time instant (instantaneous response).

long-term

information contained in the network weights.

short-term

past information available as variables (signals) within the topology

focused

is a terminology originally utilized to describe the fact that the gradients of such networks are time invariant.

memory structure

is a multidimensional single-input-multiple-output device that takes an ordered sequence of points and produces a static pattern (i.e. a point in the multidimensional space).

memory traces

are the signals at the output of the memory PE.

memory filter

is a linear structure used to store the past of the input

TDNN

stands for time delay neural network and it is a dynamic neural network where the memory structure is a tap delay line, the same memory utilized in the linear combiner of Chapter IX.

context PE

is a memory PE formed by a single first order recurrent system.

memory depth

We define the *memory depth* as the temporal extent of the system response after the application of an impulse at n=0, i.e. the number of time units where the response of the

system is different from zero.

dynamic networks

neural networks with short term memory or with feedback connections which have outputs that depend upon more than one time instant.

delay

is a type of memory where the memory mechanism is a pure delay

feedback

is another memory mechanism that is based on a recurrent connection of the output to the input.

kernel

or also called operator, is a function that implements the delay mechanism

memory resolution

number of taps per samples

AWaibel

Alex Waibel, please consult “phoneme recognition using time delay neural networks”, IEEE Proc. ASSP-37, 328-339, 1989.

Wan

Eric Wan defined the FIR synapse. Please see “Time series prediction using a a network with internal delay lines”, in Time series prediction:Forecasting the future and understanding the past, 195-217, Addison-Wesley, 1994.

gamma net

see Bert DeVries and Principe, “The gamma model: a new neural model for temporal processing”, Neural Networks 5, 565-576, 1992.

Sandberg

Erwin Sandberg, please see “Uniform approximation and gamma neural networks”, Neural Networks, vol 10, 781-784, 1977.

TLFNs

stand for time lagged feedforward networks and they are dynamic neural networks of intermediate complexity between fully recurrent networks and static networks. They are build from feedforward combinations of nonlinear PEs and linear filters.

support

the region of support is the length of the impulse response, or the number of samples the output of the delay line “remembers” the input. This will later be called the memory depth of the memory

memory trace

is a processed version of the past of the input. Note that the delay line memory produces

exactly the past inputs up to a given number of delays. So the memory is exact. But in the context PE the output is never exactly equal to the input decaying over time with a single time constant. So it is a memory trace.

Eq.7

$$g_k(n) = g(n) \bullet g_{k-1}(n) \qquad k \geq 1$$

Eq.13

$$y(n+1) = (1-\mu)y(n) + x(n)$$

Eq.3

$$y_j(n) = (1-\mu)y_j(n-1) + \mu(\sum_{i=1}^{p} x_i(n)) + b_j \qquad i \neq j$$

stationary

a signal is said to be stationary if its statistical properties do not change over time.

deVries and Principe

The gamma model: a new neural network model for temporal processing, Neural Networks 5, 565-576, 1992.

Principe and Haykin

Haykin S., Principe J., "Dynamic modeling with neural networks", in IEEE Signal Processing Magazine, vol 15, #3, 66, 1998.

gamma as an extension of Wiener filters

The gamma filter can be considered as an extension to the famous Wiener filter. Let us define the cost as $J = E\{e^2(n)\}$, where E{.} is the expectation operator. Let us call the K+1 tap vector $X(n) = [x_0(n),...,x_K(n)]^T$ and the weight vector $W = [w_0,...,w_K]^T$. So

$$J = E\{d^2(n)\} + W^T R W - 2P^T W$$

where R and P are respectively the autocorrelation function of the tap signals and the crosscorrelation vector between the tap signals and the desired response. Note that the input in the linear combiner is effectively substituted here by the tap signals. The goal of adaptation is to find the minimum of J in the space of the K+1 weights and μ. Taking the partial derivatives with respect to the weights and μ we obtain

$$\begin{cases} RW = P \\ W^T[R_\mu W - 2P_\mu] = 0 \end{cases} \quad \textbf{Equation 30}$$

where $R_\mu = \frac{\partial R}{\partial \mu} = 2E\{X(n)\frac{\partial X^T(n)}{\partial \mu}\}$ and $P_\mu = \frac{\partial P}{\partial \mu} = E\{d(n)\frac{\partial X(n)}{\partial \mu}\}$

Note that the first equation in Eq. 30 is the same as the Wiener-Hopf for the linear combiner. The difference once again is that the autocorrelation is not of the input but of the tap signals. The extra scalar condition is a result of the extra parameter in the gamma filter, and as we discussed it represents the optimal memory depth for the application. Although the gamma kernels have infinite extent they can be computed exactly in the frequency domain (provided x(n) has a computable Fourier transform). Eq. 30 has a form that applies to all the generalized feedforward filters described by Eq. 7.

Return to Text

Index

1

1. Introduction 3
13. Design of the memory space 20
15. The gamma memory PE 22
17. Prediction of nonlinear time series 34

2

2. Extracting Information in time 5

3

3. The memory PE 12
3. The Time Delay Neural Network (TDNN) 7

4

4. The memory filter 18

9

9. Time lagged recurrent networks (TLRN) 26

C

Chapter VIII- Temporal Processing with Neural Networks 2
Conclusions 36

F

Focused TLRNs built from RBFs 32

G

gamma as an extension of Wiener filters 52

M

Memory depth 41
more versatile memories 45

N

nonlinear system identification with neural networks 42

P

projection with the context PE 41

S

short-term versus long-term memory 40
static versus dynamic modelling 40

T

Takens Embedding Theorem 48