Recurrent Neural Network Architectures An Overview

Ah Chung Tsoi

Faculty of Informatics
University of Wollongong
Northfields Ave, Wollongong
NSW 2522, Australia
Ph: + 61 2 42 21 3843
Fax: +61 2 42 21 4843
Email: Ah_Chung_Tsoi@uow.edu.au

1 Introduction

The applications of multilayer perceptron (MLP) architecture to many different application areas, ranging from protein modelling [3], branch instruction prediction in software [6], to rolling steel plant modelling [30], have been very successful. It is nowadays commonly used as a mechanism for learning the input-output mapping of an underlying system, from input-output data alone [10]. As long as the underlying input-output mapping is static, and, the training data set is sufficiently large and representative of the normal operation of the system under study, it is commonly believed by practitioners in the field that MLP is good and simple to use. However, there are many problems occurring in nature, science, or engineering, which are more suitably modelled using a dynamic model, i.e., one which takes into account any possible temporal correlation of the data.

Many researchers [35, 19, 22, 9, 2] have considered alternative architectures which are more suitable to temporal signal modelling, and control. However, while there is only one general MLP architecture, there are a number of alternative recurrent neural network (RNN) architectures which have been proposed by various groups [35, 19, 22, 9, 2]. Some of these architectures, at least superficially, do not bear much resemblance to one another. To a newcomer to this area, this may appear to be somewhat confusing, as most architectures, at least superficially, appear to be different. There have been attempts in the past [19, 20, 29, 32] to find unifying themes in various architectures as proposed by researchers.

In this paper, we will give a more elementary view of the subject, emphasizing on the simplicity of the concepts behind the various architectures, and how one may transform from one architecture to another easily. We will then also make some brief remarks on newer architectures, which have been introduced for specific purposes, e.g., for overcoming long term temporal dependency, an issue associated with training algorithms; and for data structure classifications.

The structure of this paper will be as follows: in Section 2, we will first present a number of architectures which have been proposed by various researchers. In Section 3, we will introduce two general subclasses of architectures, viz., fully connected hidden

layer recurrent neural network architecture (FCHLRNN), and dynamic multilayer perceptron respectively. In Section 4, we will indicate how some of the architectures can be transformed into the FCHLRNN architecture, and some of the architectures cannot be transformed into this general form. In Section 5, we will give some brief discussions on some of the more recent architectures which have been introduced for specific purposes, e.g., to overcome long term dependency, and to classify data structures. Some conclusions will be drawn in Section 6.

2 Some Common Recurrent Neural Network Architectures

In this section, we will present a number of commonly used neural network architectures. We will first introduce the multilayer perceptron, as a background. Then, we will introduce some of the most common recurrent neural network architectures which have been proposed by various researchers.

2.1 Multilayer perceptron

Multilayer perceptron is a feedforward multilayered neural network architecture [10]. It consists of an input layer, which is assumed to have linear activation neuron characteristics; an output layer, and one or more hidden layers of neurons, which are not either input or output neurons. The hidden layer neurons are assumed to be nonlinear. Common nonlinearities include: sigmoid function, hyperbolic tangent function, radial basis function [10]. These nonlinearities can be very general, satisfying some very general conditions [11, 24]. The main reason why MLP is popular is that it has been shown, under very general assumptions on the nonlinear activation functions, that this architecture is a universal approximator, for very general static nonlinear input output maps, provided that a sufficient number of hidden layer neurons is being used [11, 24].

In general, a MLP can be described by the following model:

$$y = f(\mathbf{c}^T \mathbf{z} + c_0) \tag{1}$$

$$\mathbf{z} = F_n(B\mathbf{u} + \mathbf{b}_0) \tag{2}$$

where $\mathbf{z}$ is a n dimensional vector, denoting the outputs of the hidden layer neurons. y is a scalar variable, denoting the output of the output neuron. Here, for compatibility with what we will present in later sections, we will only assume one output. The formalism introduced in this paper can easily be carried out for more than one outputs. However, for simplicity sake and for compatibility, we will only assume one output throughout this paper. $\mathbf{u}$ is a m dimensional vector, denoting the inputs to the MLP. Here, in conformance with usual practice [10], we assume that the input layer neurons are linear. c_0 denotes the threshold of the output neuron; and $\mathbf{b}_0$, a n dimensional vector, denotes the threshold of the hidden layer neurons. $\mathbf{c}$ is a n dimensional vector, denoting the hidden layer to output layer weights. Similarly, B is a $n \times m$ matrix, denoting the weights connecting the input layer to hidden layer neurons. $f(\cdot)$ denotes the nonlinear activation of the neurons. For example, $f(\alpha) = \frac{1}{1+e^{-\alpha}}$, the common sigmoidal activation function. $F_n(\cdot)$ is a n nonlinear vector of the following form:

$$F_n(\cdot) = \begin{bmatrix} f(\cdot) \\ f(\cdot) \\ \vdots \\ f(\cdot) \end{bmatrix}$$

The weights B, $\mathbf{c}$, and thresholds c_0, and $\mathbf{b}_0$ are the unknown parameters of the network. These weights govern the behaviour of the network. The total number of adjustable weights is $n \times m + n + n + 1 = n(m+2) + 1$.

Note that it is possible to have direct feed through from the input to output as proposed in [26]. In this case, (1) is modified as follows:

$$y = f(\mathbf{c}^T\mathbf{z} + \mathbf{d}^T\mathbf{u} + c_0) \tag{3}$$

It is shown [26] that the direct feed through can lead to a better performance of the network.

In general, the output is allowed to have a range $-\infty < y(t) < \infty$, hence, it is more reasonable to modify (1) as follows:

$$y(t) = \mathbf{c}^T\mathbf{z}(t) + c_0 \tag{4}$$

i.e., the output neuron has a linear activation function. This will be the assumption that is imposed on all subsequent neural network architectures.

2.2 Algebraic Recurrent Neural Network

Pineda [21] introduced a simple modification of the MLP by allowing feedbacks among the neurons. These feedbacks are connections among the neurons, each connection is armed with an adjustable weight. However, the architecture introduced does not allow for any time delay involved in these feedback paths. Hence, we call them "algebraic" recurrent neural networks to distinguish them from the networks which involve time delayed feedback paths.

2.3 Williams-Zipser architecture

Williams and Zipser [35] introduced a general architecture, whereby the neurons in the network can act either as an input neuron, an output neuron, a hidden neuron, or a combination of input and output. In addition, they allow the connections among the neurons to be of the following type: a time delay with an associated adjustable weight. The time delay can act as a simple memory storage by delaying the incoming signal by one normalized time delay [1]. The weight allows the strength of this time delayed signal to be adjusted. As the time delay introduced in the network contains memory, this

[1] In this paper, we will consider the normalized time delays, rather than the actual absolute time delays. This is in recognition of the fact that the delay element in the architecture may have an absolute time delay of, say, x seconds, or minutes. But for modelling purposes, this is normalized to 1 time unit.

network can be used to model temporal sequences, or time series. Because of its general nature, it is quite difficult to prove any general properties concerning the behaviour of the architecture.

2.4 Time Delay Neural Networks

A simple way in which memory can be introduced into the MLP is by incorporating delays in the output of neurons. Thus, the output signal of the neuron is delayed (stored) and the output together with the stored values of past outputs will be available for further processing. This delayed signal forms a memory device which has been found to be essential in temporal signal processing. For example, if in the MLP, the output of the i th hidden layer neuron is $z_i(t)$, where t denotes the time instant. Then, we can utilize the past information of such an output if we pass the output through a tapped delay line as follows:

$$\begin{aligned} z_{i1}(t) &= z_i(t-1) \\ z_{i2}(t) &= z_{i1}(t-1) \\ &\vdots \\ z_{in_d}(t) &= z_{i,n_d-1}(t-1) \end{aligned} \tag{5}$$

The signals z_{ij}, $j = 1, 2, \ldots, n_d$ are assumed to be available and can be used to feedforward via adjustable weights, to the output layer and form the output accordingly.

The tapped delay line device can be applied to the outputs of the input neurons, the outputs of the hidden layer neurons, and/or the outputs of the output neurons. This results in what is commonly called a time delay neural network architecture [33]. This architecture has been used in speech processing [33].

2.5 Canonical form networks

Another method of modifying the MLP to incorporate delays is by using the following model (see Figure 1 which consists of two sub-models:

1. Feedforward sub-model. This is just the classic MLP (4), (2).
2. Feedback sub-model. In this feedback sub-model, it includes only d delays, i.e., if the output of the feedforward path is $y(t)$, then the feedback sub-model can be represented as follows:

$$\begin{aligned} v_0(t) &= y(t) \\ v_1(t) &= v_0(t-1) \\ v_2(t) &= v_1(t-1) \\ &\vdots \\ v_d(t) &= v_{d-1}(t-1) \end{aligned} \tag{6}$$

These d feedback signals $v_i(t)$, $i = 1, 2, \ldots, d$ can be used as additional inputs to augment the inputs of the MLP.

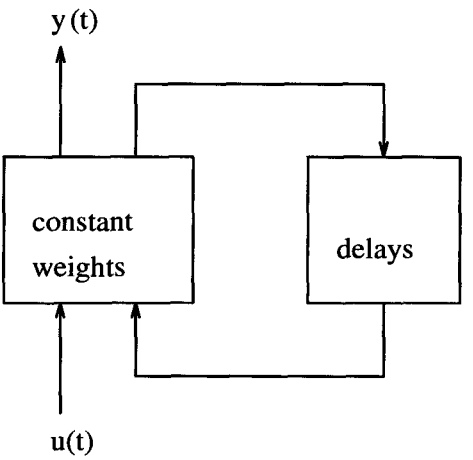

Fig. 1. The canonical form as introduced by Nerrand et al

Thus, the architecture consists of two parts:

1. a feedforward part which can be described by (4), and (2). Equation (2) is modified as follows:

$$\mathbf{z}(t) = F_n\left(\begin{bmatrix} B_1 & B_2 \end{bmatrix}\begin{bmatrix} \mathbf{u}(t) \\ \mathbf{v}(t) \end{bmatrix} + \mathbf{b}_0\right) \tag{7}$$

   where $\mathbf{v}(t)$ is a d dimensional vector with elements $v_i(t)$, $i = 1, 2, \ldots, d$ as given in (6). B_1 and B_2 are respectively constant matrices of dimensions $n \times m$ and $n \times d$.
2. A feedback part which is described by the vector $\mathbf{v}(t)$.

Note that even though the architecture has an explicit time delay of d normalized time steps, the memory storage in the architecture is dependent on the weights B_1, B_2, $\mathbf{c}$, and thresholds $\mathbf{b}_0$ and c_0. This memory storage may be much longer than d normalized time units. In other words, even though the architecture has an explicit time delay of d units, the influence of the past output values on the current output can be much longer than d time units.

This model was originally proposed in [16]. It was later shown by [20] that such a model (which they called a canonical model [2]) is a general model, in which all recurrent neural networks can be transformed. This will be the topic of Section 4.

[2] The canonical model proposed in [20] is in fact more general than the one used in this paper. They propose to decompose the network into a feedforward part, which includes only constant weights, and static nonlinear elements, and a feedback part, which involves only delay elements. Hence, in their model, the feedforward part does not need to have any particular structure. However, since the MLP is a universal approximator [11], without loss of generality, it is reasonable to assume that the feedforward part is a MLP model.

2.6 Nonlinear autoregressive moving average model with exogenous inputs

A popular model in linear time series analysis is the autoregressive moving average (ARMA) model [5]. This model can be represented as follows:

$$y(t) = \sum_{i=1}^{n_a} a_i y(t-i) + \sum_{i=1}^{n_b} b_i \epsilon(t-i) + \epsilon(t) \tag{8}$$

where $\epsilon(t)$ denotes the past imperfection of the model to represent the output $y(t)$. n_a, and n_b are constants. a_i, $i = 1, 2, \ldots, n_a$; and b_i, $i = 1, 2, \ldots, n_b$ are constants. The unknowns in this model are a_i, $i = 1, 2, \ldots, n_a$, and b_1, $i = 1, 2, \ldots, n_b$ which can be estimated from input-output data.

If b_i, $i = 1, 2, \ldots, n_b$ are zero, then the model is known as an autoregressive (AR) model. Conversely, if a_i, $i = 1, 2, \ldots, n_a$ are zero, then it is called a moving average (MA) model.

This model has been very popular with linear time series analysis [5]. It has also been a popular model in control [14]. It is possible to include an exogenous input to this model to give an ARMA model with exogenous input as follows:

$$y(t) = \sum_{i=1}^{n_a} a_i y(t-i) + \sum_{i=1}^{n_b} b_i \epsilon(t-i) + \epsilon(t) + \sum_{i=0}^{n_d} d_i u(t-i) \tag{9}$$

where $u(t)$ is an external input. d_i, $i = 1, 2, \ldots, n_d$ are constants. If $d_0 = 0$, this means there is no direct feed through from the input to the output. On the other hand, if $d_0 \neq 0$, then there is a direct feed through from the input to the output.

It is quite easy to observe that one may obtain the nonlinear version of these models by modifying the equations as follows:

$$y(t) = f\left(\sum_{i=1}^{n_a} a_i y(t-i) + \sum_{i=1}^{n_b} b_i \epsilon(t-i) + \epsilon(t) + \sum_{i=0}^{n_d} d_i u(t-i)\right) \tag{10}$$

where $f(\cdot)$ is a nonlinear function. This model is commonly known as the nonlinear ARMAX model, or NARMAX model for short [7]. In a similar manner to the AR and MA nomenclature, if b_i, $i = 1, 2, \ldots, n_b$ are zero, then it is called a NARX model; if a_i, $i = 1, 2, \ldots, n_a$ are zero, it is called a NMAX model.

Note that in these models, we have assumed just a single input and a single output. It is quite easy to extend such models to multi-input, and multi-output situation. It has been shown that the NARX model is Turing equivalent [25], i.e., it is at least as powerful as the Turing machines. This does not imply that we do not need to know other RNN architectures, as Turing equivalent is a theoretical concept. It is possible that other architectures have advantages which the NARX model does not have, e.g., universal approximator property.

2.7 Narendra-Parthasarathy architectures

Narendra and Parthasarathy [19] proposed four different categories of architectures which use neural networks for identification purposes. For convenience, we will denote them as NP-1, NP-2, NP-3, and NP-4 respectively. These are very general classification of RNNs which can be used for identification of nonlinear dynamical systems. These models are presented below only for the single-input single-output case. It is quite easy to extend the representation to multi-input multi-output cases.

NP-1 Model This model is characterized by the fact that it contains only an input nonlinearity, i.e., the input is passed through a nonlinear function before it is input to the dynamical part of the model. This can be represented as follows:

$$y(t) = \sum_{i=1}^{n} \alpha_i y(t-i) + v(t) \tag{11}$$

where $u(t)$ and $y(t)$ are respectively the input and output variables. $v(t) = g(u(t-1), \ldots, u(t-m))$, $g(\cdot)$ is a nonlinear operator. This is the nonlinear element which combines the past inputs in a nonlinear fashion. α_i, $i = 1, 2, \ldots, n$ are constants. Note that in this case, there is no direct feedthrough from the input to the output. If there is a direct feed through term from the input to the output, then we have $v(t) = g(u(t), u(t-1), \ldots, u(t-m))$ instead.

It is noted that (11) is essentially a linear dynamical system, except that the input is a nonlinear combination of the past input values.

NP-2 Model This model is the dual of the NP-1 model. It consists of a linear combination of the past input values, but the past outputs are combined in a nonlinear fashion. This can be represented as follows:

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n)) + \sum_{i=1}^{m} \beta_i u(t-i) \tag{12}$$

where $f(\cdot)$ is a nonlinear operator. It combines the past n output values in a nonlinear manner. β_i, $i = 1, 2, \ldots, m$ are constants.

NP-3 Model This model is a combination of both NP-1 and NP-2 models. It can be represented as follows:

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n)) + g(u(t-1), u(t-2), \ldots, u(t-m)) \tag{13}$$

Note that (13) can be considered as a summation of two signals $r(t) = f(y(t-1), \ldots, y(t-n))$, and $v(t) = g(u(t-1), \ldots, u(t-m))$ as follows: $y(t) = r(t) + v(t)$. Thus this model can be considered as the addition of two nonlinear signal components, $r(t)$, and $v(t)$ respectively.

NP-4 Model This a a more general model than the NP-3 model in that the output $y(t)$ is formed by a nonlinear combination of both past input and output values. This model can be represented as follows:

$$y(t) = f(y(t-1), \ldots, y(t-n), u(t-1), \ldots, u(t-m)) \tag{14}$$

where $f(\cdot)$ is a nonlinear function.

It is observed that the NP-4 model is the most general model. It includes the NP-1 model, the NP-2 model and the NP-3 model as special cases. Hence, in the rest of this paper, we will only consider the NP-4 model.

The relationship between the NP-4 model, and the NARMAX model will be explored in Section 4.

2.8 Simple recurrent neural network architecture

A simple modification to the MLP was proposed in [9]. They proposed to incorporate a simple local feedback loop around each hidden layer neuron in an otherwise classic MLP (4), and (2). In this case, the model can be described by modifying (2) as follows:

$$\mathbf{z}(t) = F_n(\Lambda \mathbf{z}(t-1) + B\mathbf{u}(t) + \mathbf{b}_0) \tag{15}$$

where Λ is a diagonal matrix of the following form:

$$\Lambda = \begin{bmatrix} w_1 & 0 & 0 \ldots 0 \\ 0 & w_2 & 0 \ldots 0 \\ \vdots & & \\ 0 & 0 & 0 \ldots w_n \end{bmatrix}$$

w_i, $i = 1, 2, \ldots, n$ are adjustable weights. Note that this model is a very simple modification of the MLP architecture. It incorporates memory by incorporating delay around the hidden layer neurons.

2.9 Fully connected hidden layer recurrent neural network architecture

It is a simple observation that (15) can be modified to incorporate a hidden layer in which all neurons are fully connected, i.e., each hidden layer neuron is connected via a normalized delay, together with an adjustable weight to each every other hidden layer neuron. In this case, the model can be represented by modifying (2) as follows:

$$\mathbf{z}(t) = F_n(A\mathbf{z}(t-1) + B\mathbf{u}(t) + \mathbf{b}_0) \tag{16}$$

where A is a $n \times n$ matrix.

This model has been introduced by a number of researchers independently in the past [8, 23, 36]. Various researchers have called it by a different name, e.g., error propagation network [23], stack machines [36].

Note that the simple recurrent neural network is a special case of this more developed model. Note that A may also take on other configurations. For example, the hidden layer neurons can be connected in a ring structure which results in an A matrix as follows:

$$A = \begin{bmatrix} 0 & a_{12} & 0 & 0 & 0 \ldots & 0 & a_{1n} \\ a_{21} & 0 & a_{23} & 0 & 0 \ldots & 0 & 0 \\ 0 & a_{32} & 0 & a_{34} & 0 \ldots & 0 & 0 \\ \vdots & & & & & & \\ a_{n1} & 0 & 0 & 0 & 0 \ldots & a_{n,n-1} & 0 \end{bmatrix}$$

where a_{ij} are constants.

The main point is that the FCHLRNN architecture is a general architecture, which enjoys a universal approximator property. On the other hand, the simple RNN architecture, or the ring structured hidden layer RNN while having a simpler structure than the FCHLRNN, they do not have universal approximator properties.

The normalized delay of 1 in the hidden layer neurons can be extended to incorporate d normalized delays as follows:

$$\mathbf{z}(t) = F_n\left(\sum_{i=0}^{d} A_i \mathbf{z}(t-i) + B\mathbf{u}(t) + \mathbf{b}_0\right) \tag{17}$$

where $A_i, i = 0, 1, \ldots, d$ are $n \times n$ matrices, denoting the connection among the hidden layer neurons.

Note that (17) can be written in a form reminiscent of the form shown in (16). Let us define a number of intermediate variables as follows:

$$\begin{aligned} \mathbf{v}_1(t) &= \mathbf{z}(t) \\ \mathbf{v}_2(t) &= \mathbf{z}(t-1) \\ &\vdots \\ \mathbf{v}_d(t) &= \mathbf{z}(t-d+1) \end{aligned} \tag{18}$$

Armed with these definitions, it is possible to write (17) as follows:

$$\begin{bmatrix} \mathbf{v}_1(t) \\ \mathbf{v}_2(t) \\ \vdots \\ \mathbf{v}_d(t) \end{bmatrix} = \begin{bmatrix} F_n\left(\sum_{i=1}^{d} A_i \mathbf{v}_i(t) + B\mathbf{u}(t) + \mathbf{b}_0\right) \\ \mathbf{v}_1(t-1) \\ \vdots \\ \mathbf{v}_{d-1}(t-1) \end{bmatrix}$$

$$= F_{nd}\left(\begin{bmatrix} A_1 & A_2 & A_3 & \ldots & A_{d-2} & A_{d-1} & A_d \\ I & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & I & 0 & \ldots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \ldots & 0 & I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1(t-1) \\ \mathbf{v}_2(t-1) \\ \vdots \\ \mathbf{v}_d(t-1) \end{bmatrix} + \begin{bmatrix} B \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mathbf{u}(t) + \begin{bmatrix} \mathbf{b}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right)$$

where $F_{nd}(\cdot)$ is a nd dimensional nonlinear function of the following form:

$$F_{nd}(\mathbf{v}(t)) = \begin{bmatrix} F_n(\mathbf{v}_1(t)) \\ \mathbf{v}_2(t) \\ \vdots \\ \mathbf{v}_d(t) \end{bmatrix}$$

$\mathbf{v}$ is a nd dimensional vector, with elements $\mathbf{v}_i$, $i = 1, 2, \ldots, d$. F_{nd} contains only n nonlinear elements, and the rest are linear elements.

From this observation, it is simple to infer that one can either express an architecture in the form of (16), or in the form of (17).

It is almost trivial to note that we can also incorporate a direct feed through from the input to the output directly [23]. In this case, the output equation (4) can be modified as follows:

$$y(t) = \mathbf{c}^T\mathbf{z}(t) + \mathbf{d}^T\mathbf{u}(t) \tag{19}$$

where $\mathbf{d}$ is a m dimensional vector.

2.10 Jordan architectures

Jordan [13] suggested two variants of a MLP architecture, whereby memory could be incorporated.

1. Feedback from the output layer to the hidden layer neurons. In this case, the model can be represented, by modifying (2) as follows:

$$\mathbf{z}(t) = F_n(\mathbf{a}y(t-1) + B\mathbf{u}(t) + \mathbf{b}_0) \tag{20}$$

   where $\mathbf{a}$ is a n dimensional vector, denoting the feedback weights connecting the output to the hidden layer neurons.
2. Feedback from the output to the input. This is the same as the architecture indicated in Section 2.5.

2.11 Finite Impulse Response MLP

This architecture was proposed independently by [34], and [2]. Basically, the idea is to replace the constant synaptic weights of the classic MLP architecture, by a finite impulse response (FIR) filter. A FIR filter [18] can be represented as follows:

$$y(t) = \sum_{i=1}^{n_b} b_i u(t-i) \tag{21}$$

where $u(t)$ is the input to the filter, and $y(t)$ is the output of the filter.

Each constant weight in the MLP architecture can be replaced by a different FIR filter which may have a different order n_b. But in order to represent the architecture more compactly, it is advisable to assume a common order for all FIR filters, n_b. This can be the maximum of the order of all the FIR filters in the architecture. In this case, each FIR synapse will have different coefficients, b_i, but has a common order, n_b. Note that in this case, the overall architecture is still a feedforward one. We will call this architecture the FIR MLP.

2.12 Gamma networks

Principe et al [22] proposed to replace the input to hidden layer weights in the classic MLP by gamma filters. A gamma filter [22] can be described as follows:

$$y(t) = \left(\frac{\gamma q^{-1}}{1-\gamma q^{-1}}\right)^{i} u(t) \tag{22}$$

where $q^{-1}y(t) \triangleq y(t-1)$. It is commonly called the shift operator. γ is a constant. i denotes the order of the gamma filter.

The gamma filter can be considered as a cascade of i sections of a first order gamma filter module $\frac{\gamma q^{-1}}{1-\gamma q^{-1}}$ together. For example, (22) may be written in the following form:

$$\begin{aligned}
y_1(t) &= \left(\frac{\gamma q^{-1}}{1-\gamma q^{-1}}\right) u(t) \\
y_2(t) &= \left(\frac{\gamma q^{-1}}{1-\gamma q^{-1}}\right) y_1(t) \\
y_3(t) &= \left(\frac{\gamma q^{-1}}{1-\gamma q^{-1}}\right) y_2(t) \\
&\vdots \qquad (23) \\
y(t) &= \left(\frac{\gamma q^{-1}}{1-\gamma q^{-1}}\right) y_{i-1}(t) \qquad (24)
\end{aligned}$$

It is assumed that the variables $y_j, j = 1, 2, \ldots, i$ are available and can be used for the construction of signals in the next layer. For example, for synapse k, one of the synapses from a particular input neuron to the hidden layer neurons, then, it is assumed that all these intermediate signals $y_j(t), j = 1, 2, \ldots, i$ are available to construct, together with other intermediate signals from synapses connecting other input neurons to hidden layer neurons, to form the input to the hidden layer neurons. For convenience we will assume that all the synapses have the same order i. Thus each synapse differs from the other synapses by having a different coefficient γ.

It is not too difficult to see that we can substitute the hidden layer to output layer weights by gamma filters as well. In this case, we will call the architecture a gamma MLP [15].

Note that the gamma MLP can be considered similar in spirit to the time delay neural network, except in this case, instead of incorporating pure time delays, we use a gamma filter module instead.

2.13 Infinite Impulse Response MLP

It is possible to substitute the constant weights in the MLP by dynamic synapses of the infinite impulse response (IIR) filter type [2]. An IIR filter [18] can be represented as follows:

$$y(t) = \left(\frac{\sum_{i=1}^{n_b} b_i q^{-i}}{1 + \sum_{i=1}^{n_a} a_i q^{-i}} \right) u(t) \tag{25}$$

where a_i. $i = 1, 2, \ldots, n_a$; b_i, $i = 1, 2, \ldots, n_b$ are constants. It is observed that the IIR filters are linear filters. These filters can modify the content of the input signals before they are modified by the neurons (nonlinear functions). Once again, we will assume that all the synapses in the architecture have the same order, i.e., n_a and n_b are the same for all synapses. Each synapse differs from the other synapses by having a different set of coefficients, a_i, $i = 1, 2, \ldots, n_a$, and b_i, $i = 1, 2, \ldots, n_b$.

It is trivial to observe that the gamma filter is a special case of the IIR filters. It is also simple to observe that the FIR filter is a special case of the IIR filter.

We will refer to this class of architectures as an IIR MLP.

3 Two Classes of General Architectures

As indicated in Section 2, there are a number of recurrent neural network (RNN) architectures. At least superficially, some of the architectures appear quite different to the others. On the other hand, as it was observed previously, some of the architectures are closely related.

There are a number of ways in which these architectures can be categorized. For example, the four models proposed by Narendra-Parthasarathy [19] are very general architectures for categorizing recurrent neural networks. The NARMAX model appears to be quite similar to NP-4 model. Hence, one would expect that the NARMAX model and the NP-4 model to be related. Another group of models which appear quite similar to one another is the FCHLRNN model, the simple recurrent neural network, the Jordan networks. Yet another group of architectures which exhibit similarity is the one based on substitution of the constant weights in the classic MLP, by dynamic synapses. Hence, the question arises: what are the relationships among all the different groups of architectures? Are they related to one another? If so, can we transform one architecture into another?

In this section, we will propose two general subclasses of recurrent neural network architectures, and then in the next section, we will deal with the issue how one architecture is related to the other.

3.1 Canonical form

This is the canonical form introduced in [20]. For ease of reference, we will repeat the description of the model here.

$$y(t) = \mathbf{c}^T \mathbf{z}(t) + c_0 \tag{26}$$

$$\mathbf{z}(t) = F_n(B\mathbf{v}(t) + \mathbf{b}_0) \tag{27}$$

where $y(t)$ and $\mathbf{v}(t)$ are respectively the scalar output and $m + d$ dimensional input vector to the multilayer perceptron. $\mathbf{z}(t)$ denotes the concatenation of the outputs of the n hidden layer neurons into one vector. B and $\mathbf{c}$ are respectively matrix and vector

of appropriate dimensions. $\mathbf{b}_0$ and c_0 denote respectively the thresholds of the hidden layer neurons and the output neuron. $\mathbf{v}(t)$ is a concatenation of the m input $\mathbf{u}(t)$ and the feedback signals from a feedback block of the following form:

$$\zeta(t) = \begin{bmatrix} y(t-1) \\ y(t-2) \\ \vdots \\ y(t-d) \end{bmatrix}$$

i.e., $\mathbf{v}(t) = \begin{bmatrix} \mathbf{u}(t) \\ \zeta(t) \end{bmatrix}$

3.2 Dynamic multilayer perceptrons

There are two ways in which this model can be represented. We will indicate both representations, as both appear to be informative, and interesting.

3.3 Representation 1

In this section, we will give a representation of a general form of dynamic MLPs. We will assume that there is one hidden layer only. Then, a dynamic MLP can be considered as a cascade connection of three subsections:

1. Input dynamical subsection. This is the subsection whereby the inputs pass through a linear filter. This subsection can be modelled [14] as follows:

$$\mathbf{y}_1(t) = C_1\mathbf{x}_1(t) + D_1\mathbf{u}(t) \tag{28}$$
$$\mathbf{x}_1(t+1) = A_1\mathbf{x}_1(t) + B_1\mathbf{u}(t) \tag{29}$$

   where $\mathbf{y}_1(t)$ is a n dimensional vector, denoting the outputs of the linear subsection. $\mathbf{u}(t)$ is a m dimensional vector, denoting the input to the dynamic MLP. $\mathbf{x}_1(t)$ is a n_1 state vector, denoting the internal state of the linear filter. A_1, B_1, C_1 and D_1 are matrices of appropriate dimensions.
2. Nonlinear section. This is the hidden layer neuron section. We will assume that there are n hidden layer neurons. They will accept the output of the input dynamical subsection $\mathbf{y}_1(t)$, and convert them through a nonlinear bank of neurons, resulting in a n dimensional output vector $\mathbf{y}_2(t)$, i.e.,

$$\mathbf{y}_2(t) = F_n(\mathbf{y}_1(t)) \tag{30}$$

   where $F_n(\cdot)$ is a n dimensional vector, denoting the nonlinearity of the hidden layer neurons.

3. Output dynamic subsection. This subsection will take the outputs from the nonlinear subsection $\mathbf{y}_2(t)$ and then transform them via a linear dynamical block to obtain the scalar output $y(t)$, i.e.,

$$y(t) = \mathbf{c}_2^T \mathbf{x}_2(t) + \mathbf{d}_2^T \mathbf{y}_2(t) \tag{31}$$

$$\mathbf{x}_2(t+1) = A_2 \mathbf{x}_2(t) + B_2 \mathbf{y}_2(t) \tag{32}$$

Here in conformity with previous sections, we have assumed that there is only one output variable of the dynamic MLP architecture. $\mathbf{x}_2(t)$ is a n_2 dimensional state vector. The input to this subsection is the n dimensional output vector $\mathbf{y}_2(t)$ from the nonlinear subsection. A_2, B_2, $\mathbf{c}_2$ and $\mathbf{d}_2$ are respectively matrices and vectors of appropriate dimensions.

3.4 Representation 2

An alternative representation of the dynamic MLP can be considered as consisting of a feedback connection of the following blocks:

1. Input delay and weight subsection. This is obtained by noting that in the input dynamical subsection, there are n_1 delay elements which are arranged in the feedforward direction. In addition, the constant matrices A_1, B_1, C_1 and D_1 denoting the connecting weights can be considered to be external to the delay elements. In other words we can consider the input dynamical subsection as consisting of two blocks, one block, the feedforward block consisting of all n_1 delay elements, together with the weights as represented by matrices B_1, C_1 and D_1; and another block, a feedback block, consisting of weights A_1. This block can be connected through a bank of linear neurons, i.e., neurons with linear activations, and be shown as outputs of the dynamic MLP architecture. Thus, by making the signals associated with the A_1 matrix available globally, it can be considered as a feedback signal from the output to the input of the dynamic MLP.
2. Output delay subsection. In a similar manner, we can consider the output dynamical subsection to be consisting of a bank of n_2 delays, together with the weights represented by B_2, $\mathbf{c}_2$ and $\mathbf{d}_2$; and a feedback block consisting of the constant matrices, A_2. The weights can be considered to be feedback via a bank of linear activation neurons. Thus, once again, by making the signals associated with the matrix A_2 available globally through the deployment of linear neurons, it is possible to interpret these as global feedbacks from the output of the dynamic MLP to the input of the dynamic MLP.
3. Nonlinear and linear subsection. This is a concatenation of the n nonlinear hidden layer neurons, the n_1 linear activation neurons, and n_2 linear activation neurons.

Thus, we have an overall architecture, which consists of a feedforward part, consisting of delays, nonlinear elements, and constant weights; and a feedback part, consisting of only constant weights.

To some readers, this type of manipulations may appear to be rather contrived. However, the main reason why we consider it here is that we can obtain a "dual" [3] of the canonical form [20] considered in the previous section. In this case, we have a feedforward subsection which consists of three subblocks: an input bank of delay elements, together with some feedforward weights, connected with a bank of $n+n_1+n_2$ neurons, $n_1 + n_2$ are linear and n are nonlinear, and an output bank of delay elements, together with some feedforward weights; and a feedback block which consists of only constant weights. Thus, while we cannot achieve the complete separability of delay elements and weights in this case, we have achieved a partial separation, in terms of a feedback block consisting of only constant weights, and a feedforward block consisting of delays, constant weights, linear and nonlinear elements.

It is quite obvious from this representation that the canonical form [20] and representation 2 presented here are different. It would be difficult to find a transformation from one into another. Hence, they are distinct subclasses of the general recurrent neural network architectures.

4 Transformation of one architecture to another

In this section, we will consider how the architectures indicated in Section 2 can be transformed into either of the two subclasses of neural network architectures, viz., the canonical form, and the dynamic MLP respectively.

4.1 NARMAX model and Narendra-Parthasarathy models

These two architectures are quite similar to one another. Indeed, it can be observed that in the NP-4 model (14), the feedback path is a nonlinear combination of the past input and output values, i.e.,

$$y(t) = f(y(t-1), \ldots, y(t-n), u(t-1), \ldots, u(t-m)) \tag{33}$$

In the NARMAX model, we have,

$$y(t) = f\left(\sum_{i=1}^{n_a} a_i y(t-i) + \sum_{i=1}^{n_b} b_i \epsilon(t-i) + \epsilon(t) + \sum_{i=1}^{n_d} d_i u(t-i)\right) \tag{34}$$

It is easily observed that (34) is a special case of (33) if we assume that $\epsilon(t)$ is zero, and that the nonlinearity in (33) takes on a special form of being a linear combination of the past input and output values respectively. This implies that (34) is a stochastic version of (33), or in other words, the NARMAX model is a stochastic model, as it includes the past inaccuracies in the model and the NP-4 model is a deterministic model. Indeed, without the $\epsilon(t)$ terms in (34), we have the NARX model.

[3] This is not strictly in the mathematical sense of "duality", as in this case, the nonlinear/linear neuron sections has more elements than those considered in the canonical form [20]. We use throughout in this paper, the term "dual" quite loosely. The task of establishing "duality" in the strict mathematical sense will be left as topics of future research.

Combining these two concepts, we have a more general model:

$$y(t) = f(y(t-1), \ldots, y(t-n), u(t-1), \ldots, u(t-m), \epsilon(t-1), \ldots, \epsilon(t-\ell)) + \epsilon(t) \tag{35}$$

of which both the NARMAX model and the NP-4 model are subclasses of this more general model. Now since [25] has shown that the NARX model is Turing equivalent, hence from a Turing machine point of view, there is no need to consider the more general NP-4 model.

4.2 Relationship of the NARMAX model and the canonical form

In this section, we wish to explore the relationship, if there is any, between the NARX model, and the canonical form.

Consider the deterministic NARX model as follows:

$$y(t) = f\left(\sum_{i=1}^{n_a} a_i y(t-i) + \sum_{i=1}^{n_d} d_i u(t-i)\right) \tag{36}$$

We can define a number of intermediate variables as follows:

$$\begin{aligned} z_1(t) &= y(t) \\ z_2(t) &= y(t-1) \\ &\vdots \\ z_{n_a}(t) &= y(t-n_a+1) \end{aligned} \tag{37}$$

Using these intermediate variables, we can re-write (36) as follows:

$$\begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_{n_a}(t) \end{bmatrix} = \begin{bmatrix} f\left(\sum_{i=1}^{n_a} a_i z_i(t-1) + \sum_{i=1}^{n_d} d_i u(t-i)\right) \\ z_1(t-1) \\ \vdots \\ z_{n_a-1}(t-1) \end{bmatrix} \tag{38}$$

If we define a n_a dimensional vector $\mathbf{z}(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_{n_a}(t) \end{bmatrix}$ then (38) can be written in a more compact form as follows:

$$\mathbf{z}(t) = F_{n_a}\left(A\mathbf{z}(t-1) + B\mathbf{u}(t)\right) \tag{39}$$

where A is a $n_a \times n_a$ matrix of the form

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_{n_a-2} & a_{n_a-1} & a_{n_a} \\ 1 & 0 & 0 & \dots 0 & 0 & 0 \\ 0 & 1 & 0 & \dots 0 & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots 0 & 1 & 0 \end{bmatrix}$$

B is a $n_a \times n_d$ matrix of the following form:

$$B = \begin{bmatrix} b_1 \dots b_{n_d} \\ 0 \dots 0 \\ \vdots \\ 0 \dots 0 \end{bmatrix}$$

$\mathbf{u}(t)$ is a n_d vector, of the following form:

$$\mathbf{u}(t) = \begin{bmatrix} u(t-1) \\ u(t-2) \\ \vdots \\ u(t-n_d) \end{bmatrix}$$

and $F_{n_a}(\cdot)$ is a n_a vector function which has the following form:

$$F_{n_a}(\mathbf{z}(t)) = \begin{bmatrix} f(z_1(t)) \\ z_2(t) \\ \vdots \\ z_{n_a}(t) \end{bmatrix}$$

i.e., F_{n_a} is a special type of nonlinear function, in which the first element is nonlinear, and the other elements are linear.

It is trivial to note that (39) is a special case of the fully connected hidden layer RNN architecture, in that the nonlinear function F_n is of a special form as indicated. Note that in this formulation, we do not have the threshold vector in (39) as in the FCHLRNN architecture. This can be added to the nonlinear term in (39) quite easily.

An interesting question arises here: can we transform any fully connected hidden layer RNN architecture into a NARX form? It turns out that this question has been answered in the negative [1]. Basically, in order for a general FCHLRNN architecture to be transformed to a NARX form, one needs to find a similarity transformation [14] which transforms the full matrix A in the FCHLRNN architecture to the more sparsely populated A matrix in (39). It turns out that the only similarity transformations that exist are the input permutations, and the sign changes [1]. There does not exist any other type of similarity transformations which can transform the full matrix A in the FCHLRNN architecture to a simpler form.

Indeed from our simple transformation indicated above, it is observed that the A matrix obtained in (39) is a very special form. In addition, the nonlinearity in (39) is

also a very special case of the more general nonlinear vector used in the FCHLRNN architecture. Hence, it is no surprise that, in general, we cannot transform a FCHLRNN architecture into a NARX model.

From this observation, it can be concluded that NARX models and FCHLRNN architectures are not equivalent to one another. The NARX model is a special case of the FCHLRNN architecture.

4.3 Equivalence of FCHL RNN architecture and the Canonical form

In this section, we will show that the FCHLRNN architecture is equivalent to the canonical form proposed in [20]. It is important to note here, that the output neuron must be linear. Otherwise the transformation methods used in this section will not be valid.

Consider the FCHLRNN architecture as follows:

$$y(t) = \mathbf{c}^T \mathbf{z}(t) + c_0 \tag{40}$$

$$\mathbf{z}(t) = F_n(A\mathbf{z}(t-1) + B\mathbf{u}(t) + \mathbf{b}_0) \tag{41}$$

where the variables are as defined in Section 2. Now it is possible to define an intermediate variable $\mathbf{v}(t) = \mathbf{z}(t-1)$. With this intermediate variable, (41) may be written as follows:

$$\mathbf{z}(t) = F_n\left(\begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{b}_0\right) \tag{42}$$

It is observed that (40) and (42) are in the same form as the classic MLP. The intermediate variable can be interpreted as a feedback signal from the output of the hidden layer neurons, delayed by 1 normalized time step, and then feedback into the input of the MLP. This is exactly the canonical form which is proposed in [20].

In a similar manner, the demonstration that a canonical form can be transformed into a FCHLRNN architecture can also be easily shown. Consider the following representation of the canonical form:

$$y(t) = \mathbf{c}^T \mathbf{z}(t) + c_0 \tag{43}$$

$$\mathbf{z}(t) = F_n(B\mathbf{v}(t) + \mathbf{b}_0) \tag{44}$$

where the variables are as defined in Section 2. B is a $n \times (m + d)$ matrix, and,

$$\mathbf{v}(t) = \begin{bmatrix} \mathbf{u}(t) \\ y(t-1) \\ y(t-2) \\ \vdots \\ y(t-d) \end{bmatrix}.$$

Let us define an intermediate variable $\zeta(t) = \begin{bmatrix} y(t-1) \\ y(t-2) \\ \vdots \\ y(t-d) \end{bmatrix}$. Let us also decompose the matrix B as follows: $B = \begin{bmatrix} B_1 & B_2 \end{bmatrix}$, where B_1 and B_2 are respectively matrices of dimension $n \times m$ and $n \times d$. Armed with these definitions, it is quite easy to note that (44) can be decomposed in the following form:

$$\begin{aligned} \mathbf{z}(t) &= F_n \left(\begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} \mathbf{u}(t) \\ \zeta(t) \end{bmatrix} + \mathbf{b}_0 \right) \\ &= F_n \left(B_1 \mathbf{u}(t) + B_2 \zeta(t) + \mathbf{b}_0 \right) \end{aligned}$$

But it is noted that $y(t) = \mathbf{c}^T \mathbf{z}(t) + c_0$, hence, $B_2\zeta(t)$ can be written in the following form:

$$\begin{aligned} B_2\zeta(t) &= B_2 \begin{bmatrix} \mathbf{c}^T \mathbf{z}(t-1) + c_0 \\ \mathbf{c}^T \mathbf{z}(t-2) + c_0 \\ \vdots \\ \mathbf{c}^T \mathbf{z}(t-d) + c_0 \end{bmatrix} \\ &= G \begin{bmatrix} \mathbf{z}(t-1) \\ \mathbf{z}(t-2) \\ \vdots \\ \mathbf{z}(t-d) \end{bmatrix} \qquad (45) \\ &= \sum_{i=1}^{d} G_i \mathbf{z}(t-i) \qquad (46) \end{aligned}$$

where G is a $n \times nd$ matrix. G can be written also equivalently in the following form: $G = \begin{bmatrix} G_1 & G_2 \ldots G_d \end{bmatrix}$, where G_i is a $n \times n$ matrix. G, or equivalently, G_i are functions of B_2 and $\mathbf{c}$. Thus, it is shown that the canonical form can be transformed into a FCHLRNN architecture.

It is rather simple to show that the Jordan network can also be expressed either in the canonical form, or the FCHL RNN architecture.

4.4 The dynamic MLP architecture

It is relatively trivial to observe that the IIR MLP is a special case of the general form of dynamic MLP. Indeed, in the IIR MLP, the input to the hidden layer neuron synapses are a special case of the more complicated dynamic MLP case. That this is the case can be easiest illustrated by an example.

Example: Consider a simple two input and two output situation. This can be considered as the input to hidden layer section, which consists of two inputs and two hidden layer neurons. Assume that each synapse is a simple first order IIR filter of the form:

$$y_i = \frac{b_i q^{-1}}{1 + a_i q^{-1}} u_j(t) \tag{47}$$

where $j = 1, 2$. The index i is used to denote the ith synapse. Thus we have the following model describing the input-output relationships:

$$\begin{aligned}
y_{11} &= \frac{b_{11} q^{-1}}{1 - a_{11} q^{-1}} u_1(t) \\
y_{12} &= \frac{b_{12} q^{-1}}{1 - a_{12} q^{-1}} u_2(t) \\
y_{21} &= \frac{b_{21} q^{-1}}{1 - a_{21} q^{-1}} u_1(t) \\
y_{22} &= \frac{b_{22} q^{-1}}{1 - a_{22} q^{-1}} u_2(t)
\end{aligned}$$

where $u_i(t)$, $i = 1, 2$ are the inputs; y_{ij}, $i = 1, 2$ and $j = 1, 2$ denote the synapses from the jth input to the ith hidden layer neuron. b_{ij}, and a_{ij} are constants.

These equations can be expressed more easily in a state space formulation as follows:

$$\begin{bmatrix} x_{11}(t) \\ x_{12}(t) \\ x_{21}(t) \\ x_{22}(t) \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{12} & 0 & 0 \\ 0 & 0 & a_{21} & 0 \\ 0 & 0 & 0 & a_{22} \end{bmatrix} \begin{bmatrix} x_{11}(t-1) \\ x_{12}(t-1) \\ x_{21}(t-1) \\ x_{22}(t-1) \end{bmatrix} + \begin{bmatrix} b_{11} & 0 \\ 0 & b_{12} \\ b_{21} & 0 \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} \tag{48}$$

and

$$\begin{bmatrix} y_{11}(t) \\ y_{12}(t) \\ y_{21}(t) \\ y_{22}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11}(t) \\ x_{12}(t) \\ x_{21}(t) \\ x_{22}(t) \end{bmatrix} \tag{49}$$

It is observed that these equations do not have full A, B or C matrices. On the other hand, the matrices in the general dynamic MLP model can be any general matrices. Hence, it is observed that the IIR MLP case is a very special case of the dynamic MLP model.

It is a simple observation to observe that the gamma MLP is also a special case of the dynamic MLP model.

It is known that the FIR filter is a simpler model than the IIR filter [18]. Indeed, in the state space realization of a FIR filter, it can be shown [18] that the A matrix is of a very simple form. It does not contain any parameters at all. It contains all zero elements, except elements 1 in some locations (see e.g., some of the examples shown in [31] in this volume). Thus, it is trivial to observe that the FIR MLP is a special case of the dynamic MLP.

Since the time delay neural network model is simply the same as the FIR MLP [34], hence it is trivial to observe that the time delay neural network is also a special case of the dynamic MLP architecture.

4.5 Zipser-Williams architecture

As was discussed in Section 2, the Zipser-Williams architecture is a very general model, in that the neurons in the network can act either as an input neuron, an output neuron, or both. Hence, it is not easy to write the Zipser-Williams architecture into either the dynamic MLP architecture or the FCHLRNN architecture. But it may be easily deduced that if there is no feedforward delay [4] involved in the network, then the Zipser-Williams architecture can be expressed in the canonical form. On the other hand, if there are feedforward delays in the network, then it may be possible to express it in the form of a dynamic MLP.

5 Some Advanced Architectures for Specific Purposes

In this section, we will consider some more recent architectures which have been developed for special purposes. In particular, we will consider two such recent classes of architectures: one class designed specifically to overcome long term temporal dependency, and the other one designed to classify data structures.

5.1 Long term dependency

In training recurrent neural networks, it has been observed that if there are a large number of stages, e.g., in the Backprop Through Time (BPTT) type algorithm [10], the backprop error could become very small when it is back propagated through from the last stage to the earlier stages. This phenomenon could cause considerable difficulties in the training process. For example, if the backprop error is too small, then the parameters of the architecture do not get updated.

A main cause of this phenomenon is that the back propagation error is required to pass through the derivative of a nonlinear function which is the nonlinear activation of the neuron. For example, if the neuron has a sigmoidal activation: $y = f(\alpha) = \frac{1}{1+e^{-\alpha}}$, then the derivative of this nonlinear function can be expressed as $y(1-y)$. Now, the value of y is between 0 and 1. Hence, multiplying by the derivative of $f(\alpha)$ would result in a positive value always smaller than 1. If there are a large number of stages, then the back propagation error will get progressively smaller and smaller.

There have been a number of architectures [4, 17, 12] which have been proposed to specifically deal with this problem. In this section, we will consider some of the more popular ones.

A simple architecture for overcoming the long term dependency issue was proposed in [17]. This model is particularly simple in that it advocates that lengthening the delay in the FCHLRNN architecture, is sufficient to reduce the problem of long term temporal dependency. Thus, in this case, we have

$$y(t) = \mathbf{c}^T \mathbf{z}(t) + c_0 \tag{50}$$

[4] Here feedforward delay can be understood in the same way as Representation 2 of the dynamic MLP architecture shown in Section 3.

$$\mathbf{z}(t) = F_n\left(\sum_{i=1}^{d} A_i \mathbf{z}(t-i) + B\mathbf{u}(t) + \mathbf{b}_0\right) \tag{51}$$

The reason why this architecture works can be reasoned as follows: since the back propagation error is the one which gets reduced as it passes through the derivative of nonlinear element, it makes sense to "circulate" as long as possible the signal in the architecture, before it is used to back propagate to the earlier sections. The effectiveness of this method depends on d, the number of delays in the fully connected hidden layer, and also the weights.

Hochreiter and Schmidhuber [12] proposed a new architecture which is specifically designed to overcome the long term temporal dependency problem. Essentially, the method allows a constant signal strength to be maintained irrespective of where it is located.

While their model is quite complicated in appearance, a simplified model may be used to gain an understanding of why the method would work. At the core of the architecture, they have the following model:

$$x = \sum_{i=1}^{m} w_i u_i \tag{52}$$

$$y = f(x) + kx \tag{53}$$

where u_i, $i = 1, 2, \ldots, m$ are the input variables. w_i, $i = 1, 2, \ldots, m$ are constants. k is a constant. Now, because of the form of (53), if we take the derivative of y with respect to some parameter (as is common in gradient based learning algorithms), from the chain rule of differentiation [10], we need to evaluate correspondingly the partial derivative of y with respect to x. Hence, we have

$$\frac{\partial y}{\partial x} = \frac{\partial f(x)}{\partial x} + k \tag{54}$$

Thus, independent of the location of the error in the architecture, the derivative $\frac{\partial y}{\partial x}$ has a constant term k. Hence, if $k > 0$, the error is guaranteed not to fall to zero.

In their full architecture [12], they have included some input and output gating devices, designed to modify the input or the output in accordance with the input or output signal strength respectively. They have shown [12] that this architecture can handle very long term temporal dependency.

5.2 Data structure

Another very interesting recent development in using recurrent neural network architectures is the application to data structure classifications. There are many problems [28] in science and engineering, e.g., molecular biology and chemistry, software verification, automated reasoning, which are more conveniently modelled in structured domains, e.g., lists, trees, graphs, of variable sizes, and complexity. Traditionally, with variable data length, the idea is to augment the data length so that they are all of the same length,

and then one may apply multilayer perceptron for its classifications [27]. However, this is rather cumbersome; and in some cases very difficult to apply [27]. [27] has proposed using an autoassociative architecture to perform this task. More recently, [28] has considered more general problems of classification of structures. It was shown [28] that recurrent neural networks can be used to learn from such structures from a set of positive and negative learning examples, and then be used to classify unseen examples.

We need to introduce some notions in graph theory before we can describe the ideas behind using recurrent neural networks to classify structures.

Graph theory concepts The materials in this subsection are presented in [28]. We will consider the set of finite directed vertex labelled graphs without multiple edges. For a set of labels Σ, a graph X (over Σ is specified by a finite set V_X of vertices, a set E_X of ordered couples of $V_X \times V_X$, and a function ϕ_X from V_X to Σ. E_X is called the set of edges, and ϕ_X is called the labelling function. The graphs may have loops, and the labels can take multiple values, i.e., not necessarily binary values.

A graph X' is called a subgraph of X if there exists a labelling function $\phi_{X'}$ such that $\phi_{X'} = \phi_X$, $V_{X'} \subset V_X$, and $E_{X'} \subset E_X$. For a finite set V, $\#V$ denotes its cardinality. Given a graph X and any vector $x \in V_X$, the function $\mathcal{O}(x)$ returns the number of edges leaving from x. In other words, $\mathcal{O}(x) = \#\{(x,z)|(x,z) \in E_X, \text{and} z \in V_X\}$. In a similar manner, we can define a function called $\mathcal{I}(x)$ which returns the number of edges entering x, i.e., $\mathcal{I}(x) = \#\{(x,z)|(x,z) \in E_X \text{and} z \in V_X\}$. The maximum number of $\mathcal{O}(x)$ over all the nodes of X will be called the valence. We assume that there is a supersource for the graph, i.e., every other node can be reached from this node. The root of a tree is always the unique supersource of the tree.

Encoding of graphs Now the graph can be encoded [28] for the purpose of classification of structured patterns. We are given a structured domain $\mathcal{D}$ over Σ, and a training set $\mathcal{T} = (X, \zeta(X))$, where ζ is the target function, defined as any function $\zeta : \mathcal{D} \rightarrow \mathcal{R}^k$, k is the output dimension. [28] introduced the notion of a generalized recursive neuron for encoding this type of graph problems as follows:

$$o^{(g)}(x) = f\left(\sum_{i=1}^{N_L} w_i l_i + \sum_{j=1}^{\mathcal{O}(x)} \hat{w}_j o^{(g)}(out_X(x,j))\right) \tag{55}$$

where $f(\cdot)$ is a sigmoidal function, N_L is the number of units encoding the label $\mathbf{I} = \phi_X(x)$ attached to the current input x, w_i are the weights associated with the inputs, and $\hat{w}_j$ are the weights on the recursive connection. $o^{(g)}(x)$ denotes the output of the generalized recursive neuron. $o^{(g)}(out_X(x,j))$ denotes the output of other generalized recursive neurons which are connected to $o^{(g)}(x)$, i.e., those vertices which point to it. The connections are governed by the graph topology.

It is best to illustrate these concepts using an example.

Example: Consider the simple example as shown in Figure 2. In this example, we have 3 nodes, A, B, and C. We will use 1 in 3 encoding scheme to encode the inputs. Thus for convenience, we will represent $A = [1\,0\,0]$, $B = [0\,1\,0]$, and $C = [0\,0\,1]$. In

this case, the input encoding is of the form: $[u_1\ u_2\ u_3]$, where u_i can take on the values of 0 or 1 dependent on the input symbol.

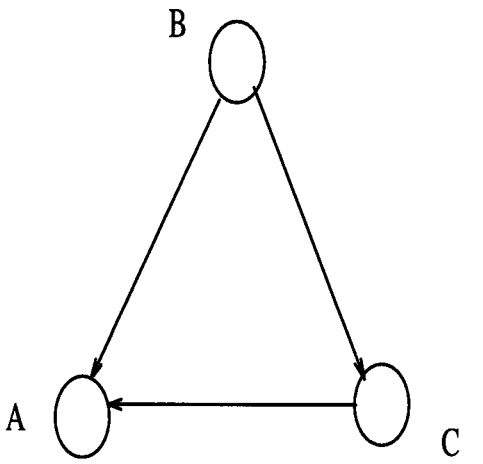

Fig. 2. A simple graph to illustrate the concepts

Then the equation describing the graph can be written as follows:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = F_3\left(\begin{bmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} + \begin{bmatrix} 0 & \hat{w}_1 & \hat{w}_2 \\ 0 & 0 & 0 \\ 0 & \hat{w}_3 & 0 \end{bmatrix}\right) \tag{56}$$

where z_i, are respectively the outputs of the nodes. b_i, $i = 1, 2, 3$ are the weights associated with the inputs u_i. Note that because we are encoding a graph structure, as a result, the weights connecting the inputs to the nodes are the same. The weights $\hat{w}_i$ are the weights which are dictated by the graph topology. F_3 is a vector of dimension 3, and is defined a follows:

$$F_3(\cdot) = \begin{bmatrix} f(\cdot) \\ f(\cdot) \\ f(\cdot) \end{bmatrix}$$

It is noted that the equation governing the encoding of the graph is a recurrent neural network, though, it is in a restricted form; due to the graph topology, and the way in which the inputs are encoded.

6 Conclusions

In this paper, we have first considered a number of popular recurrent neural network architectures. Then, two subclasses of general recurrent neural network architectures

are introduced. It is shown that all these popular recurrent neural network architectures can be grouped under either of these two subclasses of general recurrent neural network architectures. It is also inferred that these two subclasses of recurrent neural network architectures are distinct, in that it is not possible to transform from one form to the other. Two recently introduced recurrent neural network architectures specifically designed for special purposes, viz., for overcoming long term temporal dependency, and for data structure classifications are also considered.

Once the architectural aspects of the class of networks are settled, then one could consider the training aspects. This will be considered in a companion paper [31].

References

1. Albertini, F., Sontag, E. "For neural networks, function determines form". *Neural Networks.* Vol 6, pp 975 - 990, 1993.
2. Back, A.D., Tsoi, A.C. "FIR and IIR synapses, a new neural network architecture for time series modelling". *Neural Computation.* Vol. 3, No. 3, pp 375 - 385, 1991.
3. Baldi, P., Chauvin, Y. "Hybrid modelling, HMM/NN architectures, and protein modelling". *Neural Computation.* Vol 8, No. 7, pp 1541 - 1565, 1996.
4. Bengio, Y., Simard, P., Frasconi, P. "Learning Long term dependencies with gradient descent is difficult". *IEEE Trans Neural Networks.* Vol. 5, pp 157 - 166, 1994.
5. Box, G. E. P., Jenkins, G. *Time Series Analysis.* Holden Day, 1967.
6. Calder, B., Grunwald, D., Jones, M., Lindsay, D., Martin, J., Mozer, M., Zorn, B. "Evidence-based static branch prediction using machine learning". *ACM Transaction on on Programming Languages and Systems,* Vol. 19, pp 188 - 222, 1997.
7. Chen, S., Billings, S. Grant, P. "Nonlinear system identification using neural networks". *International Journal of Control.* Vol. 51, No. 6, pp. 1191 - 1214, 1990.
8. Elman, J. "Finding structure in time". *Cognitive Science.* Vol. 14, pp 179 - 211, 1990.
9. Frasconi, P., Gori, M., Soda, G. "Local feedback multilayered networks". *Neural Computation.* Vol. 4, pp 120 - 130, 1992.
10. Haykin, S. *Neural Networks, A comprehensive foundation.* MacMillan College Pub Co. 1994.
11. Hornik, K. "Approximation capabilities of multilayer feedforward neural networks". *Neural Networks.* Vol. 4, pp 251 - 257, 1990.
12. Hochreiter, S., Schmidhuber, J. "Long short-term memory". *Neural Computation.* Vol 9, pp 1735 - 1780, 1997.
13. Jordan, M. "Supervised learning and systems with excess degree of freedom". Massachusetts Institute of Technology, COINS Technical Report 88 - 27, May, 1988.
14. Kailath, T. *Linear Systems.* Prentice Hall, Englewood Cliffs, N.J., 1980.
15. Lawrence, S., Giles, L., Back, A., Tsoi, A. C. "The gamma MLP - multiple temporal resolutions, the curse of dimensionality, and gradient descent learning". *Neural Computation* To appear.
16. Lapedes, A., Farber, R. "Nonlinear signal processing using neural networks prediction and system modelling". Los Alamos Natioanl Laboratory, Los Alamos, LA-UR-262, 1987.
17. Lin, T., Horne, B.G., Giles, L. "How embedding memory in recurrent neural network architecture helps learning long term temporal dependencies". *Technical Report, University of Maryland.* Report Number UMIACS-TR-96-76, and CS-TR-3706, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, 1996.
18. Marple, S.L. *Digital spectral analysis and applications.* Englewood, N.J.: Prentice Hall, 1987.

19. Narendra, K.P., Parthasarathy, K. "Identification and Control of Dynamical Systems using Neural Networks". *IEEE Trans Neural Networks.* Vol 1, pp 4 - 27, 1990.
20. Nerrand, O., Roussel-Ragot, P., Personnaz, L., Dreyfus, G., Marcos, S. "Neural Networks and nonlinear adaptive filtering: Unifying concepts and new algorithms". *Neural Computation.* Vol 5, pp 165 - 197, 1993.
21. Pindea, F. "Dynamics and architecture for neural computation in recurrent neural networks". *Journal of Complexity.* Vol 4., pp 216 - 245, 1988.
22. Principe, J., de Vries, B., Oliveira, P. "The gamma filter – a new class of adaptive IIR filters with restricted feedback". *IEEE Trans Signal Processing.* Vol. 41, pp 649 - 656, 1993.
23. Robinson, A.J. *Dynamic error propagation networks.* PhD thesis, University of Cambridge, Cambridge, U.K., 1989.
24. Scarselli, F. Tsoi, A.C. "Universal approximation using feedforward neural networks: A survey of some existing methods, and some results". *Neural Networks.* To appear.
25. Siegelmann, H., Horne, B., Giles, L. "Computational capabilites of recurrent NARX neural networks". *IEEE Trans System, Man and Cybernetics.* Part B, Vol 27, pp 208 - 218, 1997.
26. Sontag, E. "Neural networks for control". In *Essay on Control: Perspectives in the Theory and its applications.* H. L. Trentelman, J. C. Willems, Ed. Boston:Birkhauser, pp. 339 - 380, 1993.
27. Sperduti, A. "Labelling RAAM". *Connection Science.* Vol. 6, No. 4, pp 429 - 459, 1994.
28. Sperduti, A., Starita, A. "Supervised neural networks for the classification of structures". *IEEE Trans Neural Networks.* Vol 8, pp 714 - 735, 1997.
29. Tsoi, A.C., Back, A.D. "Locally recurrent globally feedforward networks: a critical review of architectures". *IEEE Trans on Neural Networks.* Vol. 5, No. 2, pp 229 - 239, 1994.
30. A C Tsoi, "Application of neural network methodology to the modelling of the yield strength in a steel rolling plate mill", *Advances in Neural Information Processing Systems,* Vol 4. Ed. Moody, J, Hansen, S, Lippmann, R, Morgan Kaufmann Publishers, 1992.
31. Tsoi, A.C. "Gradient based learning methods". This volume.
32. Tsoi, A.D., Back, A.D. "Discrete time recurrent neural network architectures: a unifying review". *Neurocomputing.* Vol. 15, pp 183 - 224, 1997.
33. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, L. "Phonemic recognition using time delay neural networks" *IEEE Trans Acoustic Speech and Signal Processing.* Vol. 37, No. 3, pp 328 - 339, 1989.
34. Wan, E. "Temporal backpropagation for FIR neural networks". *Proc Int Joint Conf Neural Networks.* San Diego, June, 1990, pp 575 - 580, 1990.
35. Williams, R., Zipser, D. "A learning algorithm for continually running fully recurrent neural networks". *Neural Computation.* Vol. 1, pp 270 - 280, 1989.
36. Zomaya, A., Mills, P.M., Tade, M.O. *Neuron-adaptive process control, a practical approach.* Wiley, 1996.