Context Discerning Multifunction Networks: Reformulating Fixed Weight Neural Networks

Roberto A. Santiago
NW Computational Intelligence Lab
Systems Science
Portland State University
Portland, OR 97207
E-mail: robes@pdx.edu

***Abstract*— Research in recurrent neural networks has produced a genre of networks referred to as Fixed Weight Neural Networks (FWNNs) which have the ability to adapt without changing explicit weights. FWNNs are unique in that they adapt their processing based on the spatiotemporal characteristics of the incoming signal without need for weight change. As a result, a single FWNN is able to model and control many families of disparate systems without weight changes. FWNNs pose an interesting model for contextual memory in neural systems. The work reported takes a FWNN, decomposes it and analyzes its internal workings. Using new insight, FWNNs are reformulated into a simpler structure, Context Discerning Multifunction Networks (CDMN).**

I. Introduction

Fixed-weight neural networks (FWNNs) refer to recurrent neural networks (RNNs) (a.k.a. dynamic recurrent networks (DRNs)) which, after training, have the ability to adapt without further change of explicit parameters (i.e. weights). The idea seems to originate in 1990 with the work of Cotter and Conwell [1]. In their work, Cotter and Conwell separated the concept of adaptation from the notion of weight adjustment in a neural network. The researchers propose the *Fixed Weight Learning Theorem (FWLT)* which states that a RNN can approximate with arbitrary precision the dynamics of a feedforward neural network being trained with an adaptive weight learning algorithm. The innovative aspect of the FWLT is the RNN can approximate these dynamics without need for weight changes, hence the ”‘fixed-weight”’ label.

This concept would later be picked up again but referred to as *Learning to Learn* and *Metalearning* in [2], [3] and [4]. This work is conceptually different from the earlier work in that it focuses on the concept of RNNs learning to implement learning machines, hence the label metalearning. While the FWLT is built on the notion of calculating the fixed weights of the RNN that would implement a particular adaptive weight learning algorithm, metalearning instead sought to get the same fixed weights through an adaptive learning process. The resulting FWNNs from this line of research were not as general, though. For example, the researchers trained a RNN to learn a family of quadratic functions. After training, the weights of the RNN were fixed thereby making it an FWNN. Then the FWNN was shown input-output (IO) pairs from one member of the family of quadratic functions. The FWNN quickly ”‘learned”’ the quadratic mapping such that upon presentation of a new input, the FWNN produced the correct output as specified by the selected quadratic mapping. After it had learned the selected mapping, a new quadratic mapping was selected and again the FWNN ”‘learned”’ that mapping. The term ”‘learned”’ here is highlighted because it is not clear if this is learning or adaptation. It is probably more reasonable to call this adaptation since further research has shown that FWNNs created in this way while able to generalize their adaptation over the family of quadratic mappings are not able to able to ”‘learn”’ mappings not in the family of quadratic mappings. So in some way, the adaptation happening in these FWNNs is dependent upon storing some meta-level information about the family of quadratic mappings. We will later explore this meta-information as a form of memory. The researchers used specialized neural networks and training algorithms including the more well known Long Short Term Memory (LSTM) approach[5].

Feldkamp et al. [6], [7] were the first to provide useful application of the concepts. In particular, they trained a RNN to identify and control a set of simulated plants but this time using only a generic RNN whose weights were fixed after training. More recent work by Prokhorov et al. [8] has reproduced the quadratic modeling FWNN, but again only using a generic RNN. Most recently, Feldkamp et al. [9] have extended this work to create FWNNs that are able to approximate more than just one family of functions and even family of systems. As stated before, if a FWNN is created through training a RNN over the family of quadratic functions then the resulting FWNN will only adapt to approximate the quadratic functions and no other type of functions. The FWNNs created by Feldkamp and Prokhorov now are able to approximate several families of functions and even more. Their FWNNs also model time evolving dynamical systems, even those with chaotic dynamics. Thus they have extended FWNNs from simply being able to adapt to mappings but also to temporally dynamic systems. Thus for further discussion, the idea of mappings and dynamic systems will simply be referred to as systems. This ability seems very impressive and the work reported here seeks to understand the principles that allow FWNNs to adapt to all of these family of systems.

II. Motivations

The exploration into FWNNs reported here is motivated from several directions. First, FWNNs offer an interesting way of thinking about the relationship between memory and synaptic weights. While concepts like metalearning are interesting, it seems that the simpler explanatory concept of what FWNNs do is more along the lines of storing and recalling memories. While FWNNs by definition are recurrent in nature, to understand their uniqueness one might think first about a simple feedforward network, like a multilayer perceptron (MLP). Theoretical analysis of MLPs state that any function can be approximated to arbitrary precision by an MLP provided enough nodes in the network topology. For a fixed topology, if we hold the weights of an MLP fixed then it is only able to approximate a single function. If the approximated function is treated as a "memory" then an MLP with fixed weights can only hold one memory. Thus for the MLP one set of weights translates into one memory.

But now consider the FWNN that has recurrent connections. The work by Prokhorov et al. [8] and Feldkamp et al. [9] shows that large families of systems can be approximated by a single FWNN. If we now equate the concept of a memory to the the ability to adapt to a specific system then the weights in a FWNN encodes a set of memories. But now one can expand the concept of memory to not just mean the adaptation to a specific system but but to a family of systems (e.g. the quadratic maps, Henon attractors, etc.). Thus a memory is the information needed to model a family of systems.

This now in turn brings up another important aspect of these FWNNs, context. These FWNNs are able to recall this information from memory. All the work cited so far shows an impressive efficiency in being able to recall from memory the information needed to adapt to a system being presented to the FWNN. One cannot help but think that the recall mechanism implemented by FWNNs could shed significant light on the way neural systems can so efficiently resolve context. Put simply, if an FWNN has memory of a particular family of systems, then upon presentation of a member from that family of systems, it is able to place the presented system into the context of the family of systems. Then using information from the memory of the family of systems it adapts to the presented system. There is significant benefit in understanding how FWNNs accomplish this *contextual memory* capability since it seems to be a needed building block for constructing machines that display more general intelligence.

Finally, though, the training of FWNNs is computationally expensive[8] even with the use of more advanced methods for training RNNs, backpropagation through time (BTT) with extended Kalman filtering (EKF) and multistreaming[10]. It is reasonable to believe, though, that understanding the inner workings of a trained FWNN would lead to more efficient methods for creating them.

III. Methods and Results

Prokhorov provided data from a trained FWNN, in particular the one presented in [8]. The network was trained to emulate all mappings as defined by a parameterized quadratic equation. In particular the parameterized equation was

$$\begin{aligned} y(t) &= ax_1(t)^2 + bx_2(t)^2 + cx_1(t)x_2(t) \\ &\quad + dx_1(t) + ex_2(t) + f \end{aligned} \tag{1}$$

where $a, b, c, d, e, f \in [-1, 1]$. The variables are defined in discrete time. Data points were generated by randomly choosing $x_1(t)$ and $x_2(t)$ from a uniform distribution over the interval $[-1, 1]^2$ and using equation 1 to generate the desired output, $y(t)$. The parameters a, b, c, d, e were changed every 1000 steps for training and every 100 steps for testing of FWNN performance. The variables were presented to the network with a time lag on the desired output. For simplicity, the input to the FWNN can be written as $[\ \overline{x}(t) \quad y(t-1)\ ]^T$ where $\overline{x}(t) = [\ x_1(t) \quad x_2(t)\ ]$. Please note that the input is a combination of the new randomly chosen $\overline{x}(t)$ and the desired output from the *previous* time step $y(t-1)$. The FWNN had two recurrent hidden layers, with 30 and 10 nodes each. The network operation can be summarized in the following equations.

$$\begin{aligned} \overline{o}_1(t) &= sig(W_{in}[\overline{x}(t)y(t-1)]^T \\ &\quad + W_{R1}\overline{o}_1(t-1) + \overline{b}_1) \end{aligned} \tag{2}$$

$$\overline{o}_2(t) = sig(W_{hid}\overline{o}_1(t) + W_{R2}\overline{o}_2(t-1) + \overline{b}_2) \tag{3}$$

$$\widehat{y}(t) = W_{out}\overline{o}_2(t) \tag{4}$$

where $sig()$ is a bipolar sigmoidal function. The output of the FWNN was $\widehat{y}(t)$ which is the estimate of the desired output $y(t)$. The FWNN was trained using multistreamed extended Kalman filtering details of which can be found in [8] and [10]. By way of qualitative description, the network is able to take a few observations, about ten to twenty, and with those observations reconfigure its internal state to implement the appropriate mapping; that is, it implements the mapping which generated the observed data points. Extending from the previous discussion about memory; it could be said that the trained FWNN simply remembered the mapping or made an approximation from similar memories. Performance of the trained FWNN is shown in Figure 1 where the trained network was shown a new desired mapping every one hundred iterations. Using data from this network an analysis was performed on the input activation of each of the nodes of the network. In particular, the input to the nodes of the first layer is made up of four components which can be seen in equation 2: the new random values for $\overline{x}(t)$, the previous desired output $y(t-1)$, the recurred information $o_1(t-1)$ from the layer and the bias $\overline{b}_1$. Because the input to the network was the combined vector $[\ \overline{x}(t) \quad y(t-1)\ ]$ some work had to be done to isolate the input activation attributable to each part of the input. Conceptually, one could rewrite equation 2 in the following way to make it clear:

$$\overline{x}_{in}(t) = W_{in_x}\overline{x}(t)^T \tag{5}$$

$$\overline{y}_{in}(t) = W_{in_y}y(t-1) \tag{6}$$

$$\overline{r}_1(t) = W_{R1}\overline{o}_1(t-1) \tag{7}$$

$$\overline{o}_1(t) = sig(\overline{x}_{in}(t) + \overline{y}_{in}(t) + \overline{r}_{in}(t) + \overline{b}_1) \tag{8}$$

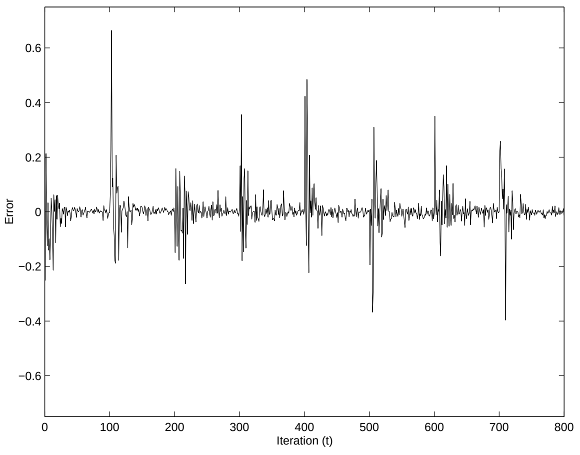

Fig. 1. Performance of trained FWNN as reported in [8]. The desired mapping was changed (i.e. new parameter values for equation 1) every 100 iterations. Notice the rapid increase in error, here $(\widehat{y}(t) - y(t))$, which occurs every hundredth iteration and quickly subsides as the network recognizes which mapping to implement.

where the weight matrix W_{in} from equation 2 is broken into two smaller matrices W_{in_x} and W_{in_y} which isolate those input from $\overline{x}(t)$ and $y(t-1)$, respectively. Similarly equation 3 can be rewritten in the following way to isolate the input activation of each node attributable to output from the first hidden layer and from recurred information:

$$\overline{a}_(t) = W_{hid}\overline{o}_1(t) \quad (9)$$

$$\overline{r}_2(t) = W_{R2}\overline{o}_2(t-1) \quad (10)$$

$$\overline{o}_2(t) = sig(\overline{a}_(t) + \overline{r}_2(t) + \overline{b}_2) \quad (11)$$

Accordingly, data was generated from the trained FWNN and the values of the variables defined in equations 5 through 11. Analysis of these variables revealed interesting features with respect to the internal processing that allowed the FWNN to model the family of quadratic functions. In particular, the weights of the first hidden layer specialized themselves so that they either receptive to input from $x(t)$ or from $y(t-1)$, as can be seen in Figure 2 and 3. For simplicity, those nodes on the first layer which are only receptive to input from $y(t-1)$ are referred to as Type I nodes and the nodes on the first layer which are only receptive to input from $\overline{x}(t)$ are referred to as Type II nodes. While this specialization with respect to input is interesting, the more intriguing aspect of the learned behavior is the contribution made by recurrence on the Type II nodes which can be seen in the upper left panel of Figure 3. The recurrent input shows fixed point equilibrium behavior with the fixed point shifting every one hundred steps, corresponding to the shift in desired mapping. Operationally, this recurrent input on Type II nodes worked to shift the bias on the node, an observation to be further explored soon.

The input behavior shown in Figures 2 and 3 is representative of the behavior of most nodes on the first hidden layer. Specifically, 16 nodes showed Type I behavior and 10 showed Type II behavior. The remaining four nodes had near zero weights for all inputs, both $\overline{x}(t)$ and $y(t-1)$. Two of these

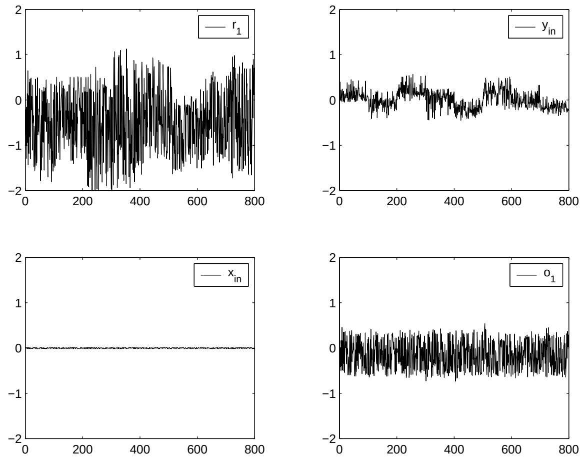

Fig. 2. Type I node behavior from the first layer. The four panels show the input and output behavior from a single node on the first layer showing Type I behavior. Type I behavior is characterized by the elimination of input from $\overline{x}(t)$ which can be seen in the lower left panel. From upper left to lower right the panels show the components of input to the node from $\overline{r}_1(t)$, $\overline{y}_{in}(t)$ and $\overline{x}_{in}(t)$ as well as the output of the node from $\overline{o}_1(t)$.

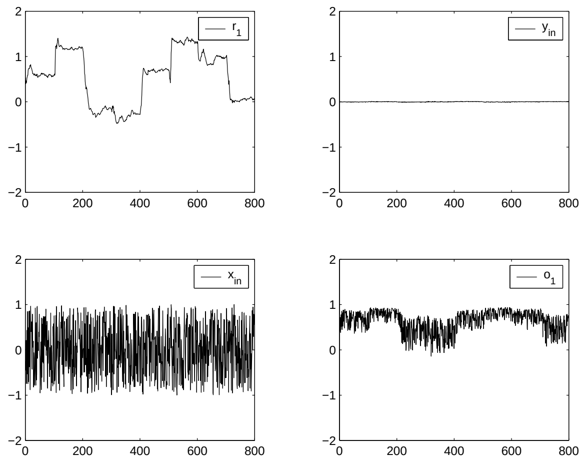

Fig. 3. Type II node behavior from the first layer. The four panels show the input and output behavior from a single node on the first layer showing Type II behavior. Type II behavior is characterized by the elimination of input from $\overline{y}(t)$ which can be seen in the upper right panel as well as the recurrent input which shows fixed point equilibrium behavior which can be seen in the upper left panel. From upper left to lower right the panels show the components of input to the node from $\overline{r}_1(t)$, $\overline{y}_{in}(t)$ and $\overline{x}_{in}(t)$ as well as the output of the node from $\overline{o}_1(t)$.

nodes had output behavior near zero indicating training had led to the functional elimination of these nodes. The other two nodes had fixed point output behavior which looked very similar to the upper left panel of Figure 3.

The second layer also had very interesting behavior. Again, two types of nodes with distinct behaviors were identified and are referred to here as Type III and Type IV nodes. Figure 4 and 5 show the values as defined in equations 9 to 11 for a Type III and a Type IV node, respectively. Type III nodes show the elimination of all output signal from the node as can be seen in the bottom panel of Figure 4. Here the FWNN training has resulted in nodes that are functionally dormant having not operational impact on the network. Type IV nodes show prominent input and output behavior and are responsible for the processing behavior of the second layer. In addition, Type IV nodes do not show any specialized behavior like fixed point equilibrium as can be seen in Type II nodes or shifts in the range of values being used as can be seen in Type I nodes (specifically the upper panels of Figure 2). The ten nodes of the second layer were evenly split between Type III and Type IV behavior.

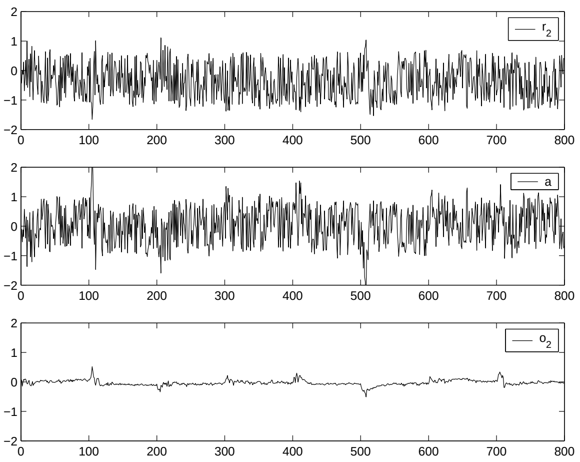

Fig. 4. Type III node behavior from the second layer. The three panels show input and output behavior for a single node from the second layer showing Type III behavior. The upper two panels show the input contributions from $\overline{a}_(t)$ and $\overline{r}_(2)$. The bottom panel shows the output from the node. Type III nodes are characterized by the elimination of output from the node.

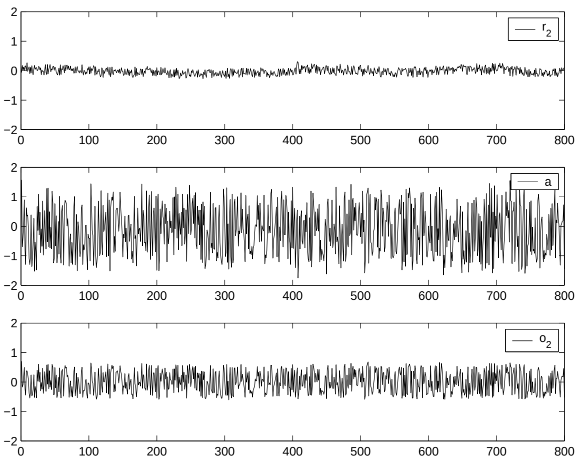

Fig. 5. Type IV node behavior from the second layer. The three panels show input and output behavior for a single node from the second layer showing Type IV behavior. The upper two panels show the input contributions from $\overline{a}_(t)$ and $\overline{r}_(2)$. The bottom panel shows the output from the node. Type IV are responsible for processing on the second layer of the network.

In analyzing the internal working of the FWNN it is apparent that much of the training resulted in both functional specialization of nodes on the first and second layer as well as functional elimination of several nodes. This suggests an overly complex network architecture was employed to solve the problem. As a result, it was hypothesized that a much simpler approach to FWNN adaptation could be synthesized. The strongest clue for this came from the analysis of Prokhorov et al. Prokhorov et al. in [8] suggests a strong connection to the work of Back and Chen in [11]. In particular they state: ”‘...it appears possible to extend the results of theoretical analysis in [Back and Chen 1998], which treats the ability of a single network with output-to-input recurrence to approximate multiple systems to the case of RMLP.”’
In 1998, Back and Chen used a hybrid system framework to show that recurrent networks could be used to implement fixed weight systems which modeled multiple systems. The work provided only hints as to the general construction process for such networks. Among those hints were included a construction process which focused on shifting the biases of a standard multi-layer perceptron (MLP). Put simply, the authors observe that for two different sets of biases, holding all other weights the same, the MLP will approximate two distinct models. Subsequently, the authors indicate bias-shifting as a method for getting a fixed weight neural network to approximate several systems [1]. The question of exactly how to adjust those biases in response to changes in desired model is not directly addressed.

As observed before, the input from the recurrent connections for Type II nodes, seen in the upper left panel of Figure 3, seems to perform this bias shifting. It seems almost too convenient that the Type II nodes also eliminate any input from $y(t-1)$ so that they are specialized to transforming $\overline{x}(t)$ into $\widehat{y}(t)$. Thus it also seems logical to hypothesize that the Type I nodes carry out the function of observing the sequence of inputs and internal states to identify shifts in desired mapping and, most importantly, to assign a set of fixed point outputs which shift the biases of Type II nodes thereby changing the mapping being approximated by the network.

It also seemed convenient that the Type III nodes on the second layer had no output and therefore were not directly involved in generating $\widehat{y}(t)$ or even in providing recurrent information to the Type IV nodes. The exact operation of the Type IV nodes still needs further analysis but it was hypothesized that the recurrence in the second layer worked to counterbalance information coming from the first layer such that the Type IV nodes of the second layer operate as standard feedforward nodes or bias-shifted feedforward nodes. This hypothesis has not yet been verified but it will simply be assumed for the reformulation of fixed weight networks to be discussed next.

From the analysis so far it seems clear that there are essentially two components of the trained FWNN producing the adaptation in the fixed weight network. The first component can be thought of as a context discernment network (CDN). This seems to be the most significant contribution of the recurrent connections, especially within the first layer. In particular, the Type I nodes seem to perform this context discernment function. By context discernment it is meant

[1]Further theoretical analysis has also been developed by Back and Chen in the series of articles [12], [13] and [14] which uses the concept of multiple nonlinear operators instead of hybrid systems.

that the component is able to identify spatial and temporal characteristics (hereafter referred to as spatiotemporal characteristics) and in response to those characteristics produce a set of fixed parameters. Those fixed parameters can be thought of as context parameters. In turn, those context parameters can be used in the second component, the multifunction network (MFN) to adjust biases as suggested by Back and Chen and apparently occurring in Type II nodes (again, refer to the upper left panel of Figure 3). Together these two subnetworks form a Context Discerning Multifunction Network (CDMN), a reformulation of the FWNN. For clarity, Figure 6 provides a diagrammatic description of the concept of CDMNs.

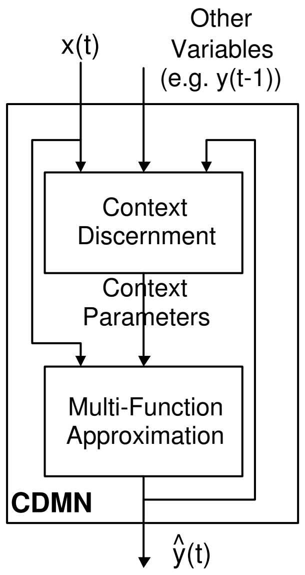

Fig. 6. Context Discerning Multifunction Network(CDMN). CDMNs have two subnetworks, a context discernment network and a multifunction approximation network. The context discernment network looks at the spatiotemporal characteristics of the incoming signals and produces a fixed value set of parameters, called context parameters. These context parameters are used by the multifunction network to adapt processing of the incoming signals.

For purposes of illustration a simple example of CDMN functionality was assembled. A CDMN was constructed which approximated a similar family of quadratic functions as already described before in equation 1. The CDN component was a simple linear regression algorithm. The algorithm took as input the last five $\overline{x}, y$ pairs (i.e. $(\overline{x}(t-1), y(t-1))$ through $(\overline{x}(t-5), y(t-5))$). These were assembled into a linear equations. For simplicity let

$$m(x_1, x_2) = [\ x_1^2 \quad x_2^2 \quad x_1x_2 \quad x_1 \quad x_1\ ] \tag{12}$$

The linear equation was as follows:

$$\begin{bmatrix} m(x_1(t-1), x_2(t-1)) \\ m(x_1(t-2), x_2(t-2)) \\ m(x_1(t-3), x_2(t-3)) \\ m(x_1(t-4), x_2(t-4)) \\ m(x_1(t-5), x_2(t-5)) \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} y(t-1) \\ y(t-2) \\ y(t-3) \\ y(t-4) \\ y(t-5) \end{bmatrix} \tag{13}$$

Simple linear regression was used to solve for a, b, c, d, e, the parameters for the system. This was done for every five data points so the values of these were constantly updated. Output from this component can be seen in Figure 7. These values were the output of the context discernment component and used as input to the MFN. The MFN was a MLP which took these parameters as inputs as well as $\overline{x}(t)$ and output $\widehat{y}(t)$. The MFN was trained to minimize squared error using the backpropagation algorithm. Training took less than three minutes on a Pentium III 450MHz computer. Figure 8 shows the raw error of the trained CDMN. Results compare favorably with those of Prokhorov's, shown in Figure 1.

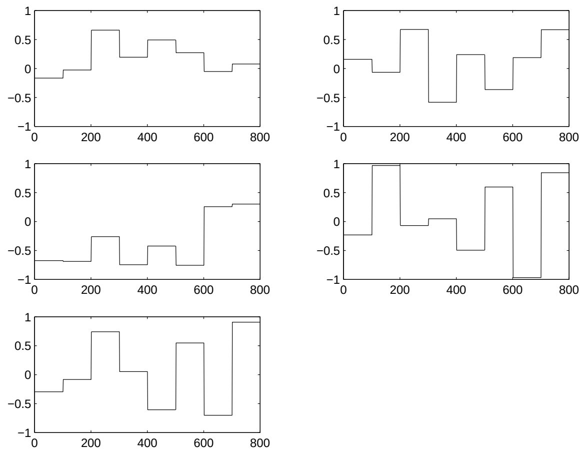

Fig. 7. Context Parameters. These context parameters are generated from using a simple regression method. While not a network, the method provides a functional example of estimated parameters uniquely identifying the spatiotemporal characteristics of the input signal. These were then used by a multifunction network which was a standard MLP.

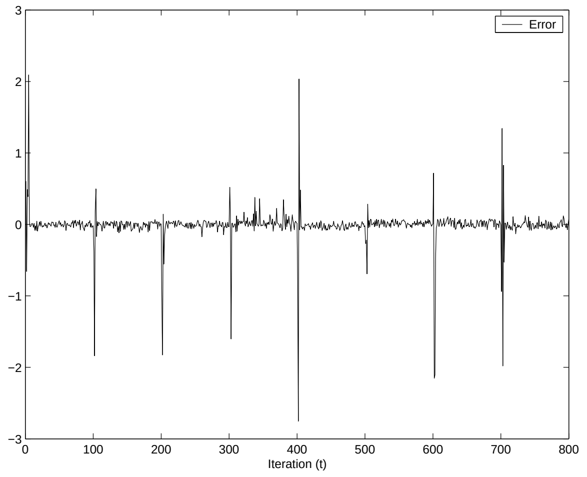

Fig. 8. Result from the simple CDMN example on the quadratic function approximation problem. The desired quadratic function is changed every 100 steps. Results are comparable to those of Prokhorov seen in Figure 1.

The use of linear regression over a polynomial basis for the CDN component was done for illustrative purposes with respect to the intended functionality of that component. Indeed, the CDMN approach highlights the challenge of future research, namely, to understand how to easily create RNNs that are able to capture spatiotemporal features of input signal and represent them parametrically so that they may be used for bias shifting of a multifunction network. This is similar to the insight that Back and Chen provide in [11]. Moreover, the FWNN analyzed only modeled a set of quadratic functions and not dynamical systems as another FWNN reported in [9] can. It would be interesting to analyze one of those networks. It is hypothesized that the CDMN formulation would still be appropriate with the exception that the context parameters might switch over the course of modeling a single system or the MFN would have to be stated as a RNN.

IV. Conclusions

Decomposition and analysis of a FWNN discussed in [8] was performed and reported. Analysis revealed functional specialization of nodes classified into four types. Further analysis revealed that several nodes of the trained FWNN have no functional impact on the processing of the network. More importantly, the functional specialization reveals a subcomponent of the network which functions as a context discernment unit. This context discernment unit observes the spatiotemporal characteristics of the input signal and assigns a set of fixed parameters to each unique source system. These fixed parameters are used to shift the biases of the remaining network thereby changing the model being approximated by the rest of the network. Using this insight, and insight from Back and Chen [11], FWNNs are now reformulated into two separate components, a context discernment network (CDN) and a multifunction network (MFN), together referred to as a Context Discerning Multifunction Network (CDMN). A very simple functional example is provided to illustrate the concept. It is concluded from this analysis that the CDMN approach is a much more efficient and intuitive method for synthesizing the desired behavior from FWNNs.

Further work is needed in formalizing the concepts of context and memory in CDMNs. It is desirable that the definition of these terms with respect to CDMNs bear strong relationship to they way they are used in neuroscience and cognitive science. To be useful in further research in machine intelligence as well as cognitive science and neuroscience more efficient methods for creating FWNNs is still needed. This paper has shown that a much of the computational effort results in specialization of the network topology and the functional elimination of nodes. It is reasonable to believe that DRNs will still be necessary for context discernment. Alternatively, a new method known as Echo State Networks [15], [16], [17] developed by Herbet Jaeger seems a promising method for developing context discernment networks computationally efficiently. Also, the method known as adaptive critics (a.k.a. approximate dynamic programming) may also play an important role. Experimentation is already underway with both of these approaches.

Acknowledgment

The author is much indebted to Danil Prokhorov for generously allowing use of data from the trained FWNN reported in [8] as well as George G. Lendaris and Tad Shannon for helpful discussion and direction. This work was supported in part by the National Science Foundation under grant ECS-0301022.

References

[1] N. Cotter and P. Conwell, “Fixed-weight networks can learn,” in *Proceedings of the International Conference on Neural Networks*, vol. 2. San Diego, CA: IEEE Press, 1990, pp. 553–559.

[2] J. Schmidhuber, “Steps towards ‘self-referential’ learning,” Department of Computer Science, University of Colorado at Boulder, Tech. Rep. CU-CS-627-92, November 1992 1992.

[3] S. Hochreiter, A. Younger, and P. Conwell, “Learning to learn using gradient descent,” in *Proceedings of the International Conference on Artificial Neural Networks (ICANN2001)*. Heidelberg: Springer, 2001, pp. 87–94.

[4] A. Younger, S. Hochreiter, and P. Conwell, “Meta-learning with backpropagation,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN2001)*. Washington, DC: IEEE Press, 2001, pp. 2001–2006.

[5] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: citeseer.nj.nec.com/hochreiter96long.html

[6] L. Feldkamp, G. Puskorius, and P. Moore, “Adaptation from fixed weight dynamic networks,” in *Proceedings of the IEEE International Conference on Neural Networks (ICNN96)*. Washinton, DC: IEEE Press, 1996, pp. 155–160.

[7] ——, “Adaptive behaviour from fixed weight dynamic networks,” *Information Sciences*, no. 98, pp. 217–235, 1997.

[8] D. Prokhorov, L. Feldkamp, and I. Tyukin, “Adaptive behavior with fixed weights in recurrent neural networks: An overview,” in *Proceedings of the International Joint Conference on Neural Networks 2002*. Honolulu, HI: IEEE Press, 2002.

[9] L. Feldkamp, D. Prokhorov, and T. Feldkamp, “Simple and conditioned adaptive behavior from a fixed neural network,” *Neural Networks*, vol. 16, no. 7, pp. 683–689, 2003.

[10] D. Prokhorov, G. Puskorius, and L. Feldkamp, “Dynamical neural networks for control,” in *A Field Guide to Dynamical Recurrent Networks*, J. Kolen and S. Kremer, Eds. IEEE Press, 2001.

[11] A. Back and T. Chen, “Approximation of hybrid systems by neural networks,” in *International Conference on Neural Information Processing (ICONIPS97)*, vol. 1. Singapore: Springer-Verlag, 1998, pp. 326–329.

[12] T. Chen and H. Chen, “Approximation of continuous functionals by neural networks with application to dynamic systems,” *IEEE Transactions on Neural Networks*, vol. 4, no. 6, pp. 910–918, 1993.

[13] ——, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,” *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[14] A. Back and T. Chen, “Universal approximation of multiple nonlinear operators by neural networks,” *Neural Computation*, no. 14, pp. 2561–2566, 2002.

[15] H. Jaeger, “The ”echo state” approach to analysing and training recurrent neural networks,” German National Research Center for Information Technology, Tech. Rep. GMD Report 148, 2001.

[16] ——, “Short term memory in echo state networks,” German National Research Center for Information Technology, Tech. Rep. GMD Report 152, 2001.

[17] ——, “Adaptive nonlinear system identification with echo state networks,” in *Neural Information Processing Systems 2002*, S. Becker, S. Thrun, and K. Obermayer, Eds., vol. 15. Vancouver, BC: MIT Press, 2002.