1-1-1991

Recursive neural networks for signal processing and control

Chaouki T. Abdallah

Don Hush

B. Horne

Follow this and additional works at: https://digitalrepository.unm.edu/ece_fsp

Recommended Citation

Abdallah, Chaouki T.; Don Hush; and B. Horne. "Recursive neural networks for signal processing and control." *Proceedings of the 1991 IEEE Workshop Neural Networks for Signal Processing [1991]* (1991): 523-532. doi:10.1109/NNSP.1991.239489.

This Article is brought to you for free and open access by the Engineering Publications at UNM Digital Repository. It has been accepted for inclusion in Electrical & Computer Engineering Faculty Publications by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

RECURSIVE NEURAL NETWORKS FOR SIGNAL PROCESSING AND CONTROL

D. Hush C. Abdallah B. Horne
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131 – USA

Abstract – This paper describes a special type of dynamic neural network called the Recursive Neural Network (RNN). The RNN is a single-input single-output nonlinear dynamical system with a nonrecursive subnet and two recursive subnets arranged in the configuration shown in Figure 1. The purpose of this paper is to describe the architecture of the RNN, present a learning algorithm for the network, and provide some examples of its use.

INTRODUCTION

This paper describes a dynamic neural network structure referred to as the recursive neural network, RNN. This network is particularly well suited for signal processing and control applications. The RNN structure is shown in Figure 1. It consists of three subnets, *A*, *B*, and *C*. All three of the subnets are multi-layer perceptrons with optional high-order connections at the input layer. Subnet *A* is a *nonrecursive* subnet that processes current and delayed values of the input $d(k)$ while subnets *B* and *C* are *recursive* subnets that feedback delayed values from the RNN output, $u(k)$, and the plant output, $y(k)$, respectively. The other components in Figure 1, *M* and *P*, are nonlinear dynamical systems which are to be modeled or controlled by the RNN.

This paper is organized as follows. Section 2 describes the operation of the RNN and introduces the notation that will be used. Section 3 presents a learning algorithm for the RNN. Section 4 presents several examples. These examples illustrate the use of the RNN in a variety of scenarios including system identification, nonlinear digital filtering, inverse modeling, control, and nonlinear prediction. Section 5 contains an analysis and summary of the RNN.

Figure 1: The Recursive Neural Network Structure

THE RECURSIVE NEURAL NETWORK STRUCTURE

An integral part of the recursive neural network structure shown in Figure 1 is the plant, P. In general we allow P to be a nonlinear dynamical system whose input/output behavior is described by

$$y(k) = g(u(k), u(k-1), \dots, u(k-\alpha_n), y(k-1), \dots, y(k-\alpha_r))$$

where α_n and α_r are the *nonrecursive* and *recursive* orders of the plant respectively. The plant may be an unknown system, that is we may not know the exact functional form of $g(\cdot)$.

The output of the RNN at time k is the sum of the outputs of three subnets, a nonrecursive subnet A and two recursive subnets B and C.

$$u(k) = v_{M_A}^A(k) + v_{M_B}^B(k) + v_{M_C}^C(k)$$

Each of these networks is a multi-layer perceptron with second-order terms at the input layer (SOMLP) [1]. The superscripts A, B, and C are used to distinguish between parameters in the nonrecursive and the two recursive subnets respectively. M_A , M_B , and M_C represent the number of layers in subnets A, B, and C respectively.

If we let $v_l^S(k)$ represent the output vector of the l^{th} layer for the S^{th} subnet ($S \in A, B, C$) then signals are propagated through the networks according to

$$v_l^S(k) = f(W_l^S v_{l-1}^S(k) + w_{l,0}^S)$$

W_l^S represents the weight matrix of subnet S that connects the outputs of layer $l-1$ to the nodes in layer l , and $w_{l,0}^S$ are the bias weights for layer l .

$f(\cdot)$ is a function which produces a vector result from a vector argument by applying a nonlinear function, $f(\cdot)$ to each component. One of the common choices for the nonlinear function is $f(z_i) = \tanh(z_i)$.

The input vector to subnet A , the nonrecursive subnet, consists of current and delayed values of the input signal, and their second order combinations of these values. Thus, $v_0^A(k)$ takes on the form

$$v_0^A(k) = \begin{bmatrix} x(k), x(k-1), \dots, x(k-o_A), \\ x^2(k), x(k)x(k-1), \dots, x(k)x(k-o_A), \\ x^2(k-1), x(k-1)x(k-2), \dots, x(k-1)x(k-o_A), \\ x^2(k-2), \dots, x^2(k-o_A) \end{bmatrix}$$

Similarly, the input to subnet B consists of previous RNN outputs (which are the plant inputs) and all second order combinations, and the input to subnet C consists of previous plant outputs and all second order combinations. We will use o_A , o_B , and o_C to denote the "order" of subnets A , B , and C respectively (i.e. the number of delayed values that are presented at the inputs to these subnets). Note, all subnets have only one output node so that W_{MA}^A , W_{MB}^B , and W_{MC}^C are row vectors and $w_{MA,0}^A$, $w_{MB,0}^B$, and $w_{MC,0}^C$ are scalars.

WEIGHT UPDATE EQUATIONS

The learning algorithm is a gradient search designed to minimize the total sum-of-squared error over the duration of the training signal,

$$E = \frac{1}{2} \sum_{k=1}^N e^2(k) = \frac{1}{2} \sum_{k=1}^N (d(k) - y(k))^2$$

where $d(k)$ is the output of the model M in Figure 1. The weight updates take on the following form,

$$w_{i,i,j}^S(k+1) = w_{i,i,j}^S(k) + \mu e(k) \beta_{i,i,j}^S(k)$$

where $w_{i,i,j}^S$ is the element of W_i^S that corresponds to the weight connecting node i in layer $l-1$ to node j in layer l , and

$$\beta_{i,i,j}^S(k) = \frac{\partial y(k)}{\partial w_{i,i,j}^S} = q_1^T(k) a_{i,i,j}^S(k) + q_2^T(k-1) b_{i,i,j}^S(k-1) \quad (1)$$

where

$$q_1^T(k) = \left[\frac{\partial g(\cdot)}{\partial u(k)}, \frac{\partial g(\cdot)}{\partial u(k-1)}, \dots, \frac{\partial g(\cdot)}{\partial u(k-o_n)} \right]$$

$$q_2^T(k-1) = \left[\frac{\partial g(\cdot)}{\partial y(k-1)}, \frac{\partial g(\cdot)}{\partial y(k-2)}, \dots, \frac{\partial g(\cdot)}{\partial y(k-o_r)} \right]$$

are elements of the Jacobian of the plant, and

$$\begin{aligned} \mathbf{a}_{i,j}^S(k) &= [\alpha_{i,j}^S(k), \alpha_{i,j}^S(k-1), \dots, \alpha_{i,j}^S(k-\alpha_n)] \\ \mathbf{b}_{i,j}^S(k-1) &= [\beta_{i,j}^S(k-1), \beta_{i,j}^S(k-2), \dots, \beta_{i,j}^S(k-\alpha_r)] \end{aligned}$$

If $g(\cdot)$ is known then $\mathbf{q}_1$ and $\mathbf{q}_2$ can be evaluated directly. If $g(\cdot)$ is unknown then $\mathbf{q}_1$ and $\mathbf{q}_2$ must be estimated.

The other vectors in (1) are determined as follows. First we note that the current value of $\beta_{i,j}^S(k)$ is a recursive function of previous values through the components of the $\mathbf{b}_{i,j}^S$ vector. The components of $\mathbf{a}_{i,j}^S$ are given by [2],

$$\alpha_{i,j}^S(k) = BP_{i,j}^S(k) + FF^B(\mathbf{a}_{i,j}^S(k-1)) + FF^C(\mathbf{b}_{i,j}^S(k-1))$$

where $BP_{i,j}^S(k)$ are the standard backpropagation equations [3] (sometimes called "static backprop") for subnet S , and the $FF^S(\cdot)$ operations are given for any vector $\mathbf{c}$ by

$$FF^S(\mathbf{c}) = h_{M_S}^S(k) \mathbf{W}_{M_S}^S \mathbf{H}_{M_S-1}^S(k) \mathbf{W}_{M_S-1}^S \dots \mathbf{H}_1^S(k) \mathbf{W}_1^S [\nabla \mathbf{r} \mathbf{v}_0^S(k-1)] \mathbf{c}$$

where,

$$\mathbf{H}_l^S(k) = \text{diag}(h_{i,n}^S(k)) \quad n = 1, 2, \dots, N_l$$

$$h_{i,n}^S(k) = \left. \frac{\partial f(z)}{\partial z} \right|_{z_{i,n}^S(k)}$$

and,

$$\begin{aligned} \mathbf{r}^T &= [u(k-1) \ u(k-2) \ \dots \ u(k-o_B)] & \text{for subnet } S = B \\ \mathbf{r}^T &= [y(k-1) \ y(k-2) \ \dots \ y(k-o_C)] & \text{for subnet } S = C \end{aligned}$$

Note that for convenience we have defined $v_{l,0}$, the input to the bias weights, to be 1. Also, N_l represents the number of nodes in layer l .

EXAMPLES

In this section we illustrate the use of the RNN in a variety of applications. These include system identification, nonlinear digital filtering, inverse modeling, control, and nonlinear prediction.

The RNN in System Identification

In this application subnet C is omitted, $P=1$, and M represents the unknown nonlinear system to be identified (refer to Figure 1). The two subnets A and B allow the RNN to model both the recursive and nonrecursive behavior of M .

Figure 2: Example of System Modeling Using MLPs

Figure 3: Example of System Modeling Using SOMLPs

One can obtain a linear IIR filter from the RNN by reducing both subnets to a single linear node. If, in addition, subnet B is omitted one obtains a linear FIR filter. Procedures for adapting the weights of both FIR and IIR filters are well known. The gradient learning algorithm which is used to adapt the weights of the RNN reduces to the well known Recursive LMS algorithm when subnets A and B are linear [4]. It is worth mentioning that with the weights of the structure fixed and bounded the RNN is BIBO stable. In contrast, this property does not hold in general for the linear IIR filter.

As an example we have trained the RNN to model the input/output behavior of the nonlinear system described by the following difference equation [5]

$$y(k) = x(k) + \frac{y(k-1)y(k-2)(y(k-1) + 2.5)}{1 + y^2(k-1) + y^2(k-2)} \quad (2)$$

For proper generalisation the network should be trained with white noise.

Figure 4: The Mapping (a) of Equation (3), (b) Learned by the RNN.

In our first simulation we used only linear terms in the input layers to the MLPs. The training signal consisted of uniformly distributed white noise over the range $[-2,2]$. The nonrecursive subnet had two hidden layer nodes and the recursive subnet had thirty. The inputs to the subnets were both of order 2, that is $o_A = o_B = 2$. This yielded a network with a total of 121 weights. Figure 2 compares the output of the RNN (after training) to the output of the actual system when presented with a new white noise.

In our second simulation we used second order terms at the input to the recursive subnet (i.e. a SOMLP is used for subnet B). The input signal to the nonlinear system and the RNN in this case was uniform white noise over the interval $[-3,3]$. Here we found that the RNN was capable of producing a model with the same accuracy as in the first example with only 1 layer (with 1 node) in the nonrecursive subnet, and 2 layers with 6 hidden layer nodes in the recursive subnet. This implementation contains a total of 46 weights. An example of the operation of this network (after training) is shown in Figure 3.

A closer examination of the manner in which the RNN actually forms the model in this second case suggests that the nonrecursive subnet simply passes the input $x(k)$ with a gain of 1, and that the recursive subnet models the recursive part of the system in (2). That is, the recursive subnet attempts to produce a mapping of the form

$$q_3 = \frac{q_1 q_2 (q_1 + 2.5)}{1 + q_1^2 + q_2^2} \quad (3)$$

Actually the recursive subnet is not asked to produce q_3 for all possible pairs $[q_1, q_2]$, only those that correspond to valid state trajectories in the nonlinear system. The mapping in (3) is shown in Figure 4 along with the mapping produced by the recursive subnet.

Figure 5: Example of nonlinear filtering.

The RNN in Nonlinear Filtering

When the RNN is used for nonlinear filtering the network configuration is the same as in the previous section. That is, subnet C is omitted and $P=1$. The difference in this case is that M represents a known nonlinear filtering operation that we wish to implement with the RNN. In the general case both subnets A and B will be needed. Many popular nonlinear filtering techniques are nonrecursive however, and can be implemented using only subnet A . As an example we trained the RNN (subnet A only with linear input terms) to act as a 1-D median filter with a window size of 3 samples (i.e. $o_A = 2$). The subnet had two layers with 12 nodes in the hidden layer. These results are illustrated in Figure 5 which shows the input to the filter, the median filter output, and the RNN output.

The RNN in Inverse Modeling

In this application the configuration in Figure 1 is such that $M=1$ and P represents the system that we wish to invert. As in the previous applications subnet C is omitted. Here the RNN is trained to form an inverse model of P so that the transformation from the input of the RNN to the output of P is 1 (possibly with a fixed delay, that is M may be equal to $z^{-\Delta}$). Obviously P must be invertible before the RNN can be applied successfully in this application.

As an example, the RNN was trained to form an inverse model of the plant described by the difference equation in (2). To insure proper generalization the network was trained with uniform white noise in the range $[-3,3]$. Both subnets in the RNN had two layers, the nonrecursive subnet had 6 hidden layer nodes, and the recursive subnet had 2. Second-order terms were computed at the input to both subnets and $o_A = o_B = 2$. This resulted in an overall network structure with 82 weights. The plots in Figure 6 show the

Figure 6: Example of Inverse Modeling

Figure 7: Example of Model Following

output of the plant (after training) with the RNN subjected to a different white noise input.

The RNN in Control

In this application all three subnets may be used. M is a model that we would like P to follow. Obviously the plant must be controllable before the RNN can be applied successfully in this application. Fortunately, in many control applications, the plant is required to follow the model for specific inputs only. This often makes the training task easier since generalization is not as much an issue.

As an example the RNN was trained to make a plant described by the difference equation in (2) follow a linear model described by [5],

$$d(k) = 0.32d(k-1) + 0.64d(k-2) - 0.5d(k-3) + x(k-1)$$

Figure 8: Example of Nonlinear Prediction

The input in this example was a sinusoid of the form $x(k) = \sin(2\pi k/25)$. Subnet B was disabled so that only subnets A and C were used. Subnet A contained a single linear node with $o_A = 2$. Subnet C was a two layer network with $o_C = 2$, linear input terms, and 8 hidden layer nodes for a total of 37 weights. Figure 7 compares the output of the model to the output of the plant after training.

The RNN in Nonlinear Prediction

When the RNN is used for prediction the structure in Figure 1 is such that $P=1$, $M=1$, and subnet C is omitted. This leaves only the RNN with subnets A and B . For Δ -step prediction the input to subnet A is delayed by Δ . Thus, the RNN is asked to use past inputs $x(k-\Delta)$, $x(k-\Delta-1)$, etc. to predict the current input, $x(k)$. As an example the RNN was trained to perform 2-step prediction of the chaotic sequence $x(k) = 4.0x(k-1)[1.0 - x(k-1)]$. In this example only subnet A was used. It had three layers with $o_A = 2$, second order input terms, 8 nodes in the first hidden layer, and 4 nodes in the second hidden layer for a total of 89 weights. The results are illustrated in Figure 8 which shows the actual and predicted sequences after training.

SUMMARY

We have introduced a nonlinear dynamical system called the recursive neural network and presented a learning algorithm for this network based on a gradient search. The network has been shown to be useful in a variety of applications including system modeling, nonlinear filtering, inverse modeling, nonlinear prediction, and control.

The recursive neural network was motivated by problems and concepts from nonlinear filtering and control. It closely resembles the architectures

proposed in [5], and can be viewed as a complement to that work.

The RNN is also similar in capability to the recurrent neural networks discussed in [6], in that both networks are nonlinear dynamical structures that contain adjustable weights. In the RNN the states are created by feeding the input and output of the net through an observer composed of the α_n and α_r delays. In recurrent neural networks every node in the network has access to its own state and assumes the ability of directly measuring the state of a system.

It should be cautioned that this paper and the simulated examples are not meant to illustrate the universality or the ease of using the RNN. Indeed, we can point out that the behavior of the closed-loop system is sensitive to many design parameters and modeling assumptions. The fact remains however, that after extensive simulation runs, the RNN presents itself as a useful alternative when analytical methods are lacking.

REFERENCES

- [1] C. Giles and T. Maxwell, "Learning, invariance, and generalisation in high-order neural networks," *Applied Optics*, vol. 26, no. 23, pp. 4972-4978, 1987.
- [2] D. Hush, C. Abdallah, and B. Horne, "The recursive neural network," Tech. Rep. EECE 91-002, Department of Electrical and Computer Engineering, University of New Mexico, 1991.
- [3] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. Rumelhart and J. McClelland, eds.), pp. 318-362, Cambridge, MA: MIT Press, 1986.
- [4] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985.
- [5] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 4-27, March 1990.
- [6] F. Pineda, "Generalization of backpropagation to recurrent and higher order neural networks," in *Neural Information Processing Systems* (D. Anderson, ed.), pp. 602-611, American Institute of Physics, 1988.