NOTE **Communicated by Ronald Williams**

Relating the Slope of the Activation Function and the Learning Rate Within a Recurrent Neural Network

Danilo P. Mandic
Jonathon A. Chambers
Signal Processing Section, Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, London, U.K.

A relationship between the learning rate η in the learning algorithm, and the slope β in the nonlinear activation function, for a class of recurrent neural networks (RNNs) trained by the real-time recurrent learning algorithm is provided. It is shown that an arbitrary RNN can be obtained via the referent RNN, with some deterministic rules imposed on its weights and the learning rate. Such relationships reduce the number of degrees of freedom when solving the nonlinear optimization task of finding the optimal RNN parameters.

1 Introduction

Selection of the optimal parameters for a recurrent neural network (RNN) is a rather demanding nonlinear optimization problem. However, if a relationship between some of the parameters inherent to the RNN can be established, the number of the degrees of freedom in such a task would be smaller, and therefore the corresponding computation complexity would be reduced. This is particularly important for real–time applications, such as nonlinear prediction.

In 1996, Thimm, Moerland, and Fiesler (1996) provided the relationship between the slope of the logistic activation function,

$$\Phi(x) = \frac{1}{1+e^{-\beta x}}\,, \tag{1.1}$$

and the learning rate for a class of general feedforward neural networks trained by backpropagation. The general weight–adaptation algorithm in adaptive systems is given by

$$\mathbf{W}(n) = \mathbf{W}(n-1) - \eta\nabla_{\mathbf{W}}E(n), \tag{1.2}$$

where $E(n)$ is a cost function, $\mathbf{W}(n)$ is the weight vector at the time instant n, and η is a learning rate.

The gradient $\nabla_{\mathbf{W}}E(n)$ in equation 1.2 comprises the first derivative of the nonlinear activation function, equation 1.1, which is a function of β

(Narendra & Parthasarathy, 1990). As β increases, so too will the step on the error performance surface (Zurada, 1992). It therefore seems advisable to keep β constant, say at unity, and to control the features of the learning process by adjusting the learning rate η, thereby having one degree of freedom less, when all of the parameters in the network are adjustable. Such reduction may be very significant for the nonlinear optimization algorithms, running on a particular RNN.

2 Static and Dynamic Equivalence of Two Topologically Identical RNNs

Because the aim is to eliminate either the slope β or the learning rate η from the paradigm of optimization of the RNN parameters, it is necessary to derive the relationship between a network with arbitrarily chosen parameters β and η, and the referent network, so as to compare results. An obvious choice for the referent network is the network with $\beta = 1$. Let us therefore denote all the entries in the referent network, which are different from those of an arbitrary network, with the superscript R joined to a particular variable, such as $\beta^R = 1$. By static equivalence, we consider the calculation of the output of the network, for a given weight matrix $\mathbf{W}(n)$, and input vector $\mathbf{u}(n)$, whereas by dynamic equivalence, we consider the equality in the sense of adaptation of the weights.

2.1 Static Equivalence of Two Isomorphic RNNs. In order to establish the static equivalence between an arbitrary and referent RNN, the outputs of their neurons must be the same,

$$y_k(n) = y_k^R(n) \Leftrightarrow \Phi\left(\mathbf{w}_k(n)\mathbf{u}_k\right) = \Phi\left(\mathbf{w}_k^R(n)\mathbf{u}_k\right), \tag{2.1}$$

where $\mathbf{w}_k(n)$, and $\mathbf{u}_k(n)$ are, respectively, the set of weights and the set of inputs that belong to the neuron k. For a general nonlinear activation function, we have

$$\Phi\left(\beta, \mathbf{w}_k, \mathbf{u}\right) = \Phi\left(1, \mathbf{w}_k^R, \mathbf{u}\right) \Leftrightarrow \beta\mathbf{w}_k = \mathbf{w}_k^R. \tag{2.2}$$

For the case of the logistic nonlinearity, for instance, we have

$$\frac{1}{1+e^{-\beta\mathbf{w}_k\mathbf{u}}} = \frac{1}{1+e^{-\mathbf{w}_k^R\mathbf{u}}} \Leftrightarrow \beta\mathbf{w}_k = \mathbf{w}_k^R, \tag{2.3}$$

where the time index (n) is neglected, since all the vectors above are constant during the calculation of the output values. As the equality (see equation 2.2) can be provided for any neuron in the RNN, it is therefore valid for the complete weight matrix $\mathbf{W}$ of the RNN.

The essence of the above analysis is given in the following lemma, which is independent of the underlying learning algorithm for the RNN, which makes it valid for two isomorphic RNNs of any topology and architecture.

Lemma. *For an RNN with weight matrix* $\mathbf{W}$, *whose slope in the activation function is* β, *to be equivalent in the static sense to the referent network, characterized by* $\mathbf{W}^R$, *and* $\beta^R = 1$, *with the same topology and architecture (isomorphic), as the former RNN, the following condition,*

$$\beta \mathbf{W} = \mathbf{W}^R, \tag{2.4}$$

must hold for every discrete time instant n while the networks are running.

2.2 Dynamic Equivalence of Two Isomorphic RNNs . The equivalence of two RNNs, includes both the static equivalence and dynamic equivalence. As in the learning process in equation 1.2, the learning factor η is multiplied by the gradient of the cost function; we shall investigate the role of β in the gradient of the cost function for the RNN. We are interested in a general class of nonlinear activation functions where

$$\begin{aligned}\frac{\partial \Phi(\beta, x)}{\partial x} &= \frac{\partial \Phi(\beta x)}{\partial(\beta x)} \frac{\partial(\beta x)}{\partial x} \\ &= \Phi'(\beta x)\beta = \beta \frac{\partial \Phi(\beta x)}{\partial(\beta x)} = \beta \frac{\partial \Phi(1, \beta x)}{\partial x}.\end{aligned} \tag{2.5}$$

In our case, it becomes

$$\Phi'(\beta, \mathbf{w}, \mathbf{u}) = \beta \Phi' \left(1, \mathbf{w}^R, \mathbf{u}\right). \tag{2.6}$$

Indeed, for a simple logistic function (see equation 1.1), we have $\Phi'(x) = \frac{\beta e^{-\beta x}}{(1+e^{-\beta x})^2} = \beta \Phi'(x^R)$, where $x^R = \beta x$ denotes the argument of the referent logistic function (with $\beta^R = 1$), so that the network considered is equivalent in the static sense to the referent network. The results, equations 2.5 and 2.6, mean that wherever Φ' occurs in the dynamical equation of the real-time recurrent learning (RTRL)–based learning process, the first derivative (or gradient when applied to all the elements of the weight matrix $\mathbf{W}$) of the referent function equivalent in the static sense to the one considered becomes multiplied by the slope β.

The following theorem provides both the static and dynamic interchangeability of the slope in the activation function β and the learning rate η for the RNNs trained by the RTRL algorithm.

Theorem. *For an RNN with weight matrix* $\mathbf{W}$, *whose slope in the activation function is* β *and learning rate in the RTRL algorithm is* η, *to be equivalent in the* *dynamic sense to the referent network, characterized by* $\mathbf{W}^R$, $\beta^R = 1$, *and* η^R, *with the same topology and architecture (isomorphic), as the former RNN, the following conditions must hold:*

1. *The networks must be equivalent in the static sense, that is,*

$$\mathbf{W}^R(n) = \beta \mathbf{W}(n). \tag{2.7}$$

2. *The learning factor* η *of the network considered and the learning factor* η^R *of the equivalent referent network must be related by*

$$\eta^R = \beta^2 \eta. \tag{2.8}$$

3 Extensions of the Result

It is now straightforward to show that the conditions for static and dynamic equivalence of isomorphic RNNs derived so far are valid for a general RTRL-trained RNN. The only difference in the representation of a general RTRL-trained RNN is that the cost function comprises more squared error terms, that is,

$$E(n) = \sum_{j \in \mathcal{C}} e_j^2(n), \tag{3.1}$$

where $\mathcal{C}$ denotes those neurons whose outputs are included in the cost function.

Moreover, because two commonly used learning algorithms for training RNNs, the backpropagation through time (BPTT) (Werbos, 1990) and the recurrent backpropagation algorithms (Pineda, 1987), are derived based on backpropagation and the RTRL algorithm, the above result follows immediately for them.

4 Conclusions

The relationship between the slope β in a general activation function, and the learning rate η in the RTRL-based learning of a general RNN has been derived. Both static and dynamic equivalence of an arbitrary RNN and the referent network with respect to β and η are provided. In that manner, a general RNN can be replaced with the referent isomorphic RNN, with slope $\beta^R = 1$ and modified learning rate $\eta^R = \beta^2\eta$, hence providing one degree of freedom less in a nonlinear optimization paradigm of training the RNNs. The results provided are straightforwardly valid for the BPTT and recurrent backpropagation algorithms.

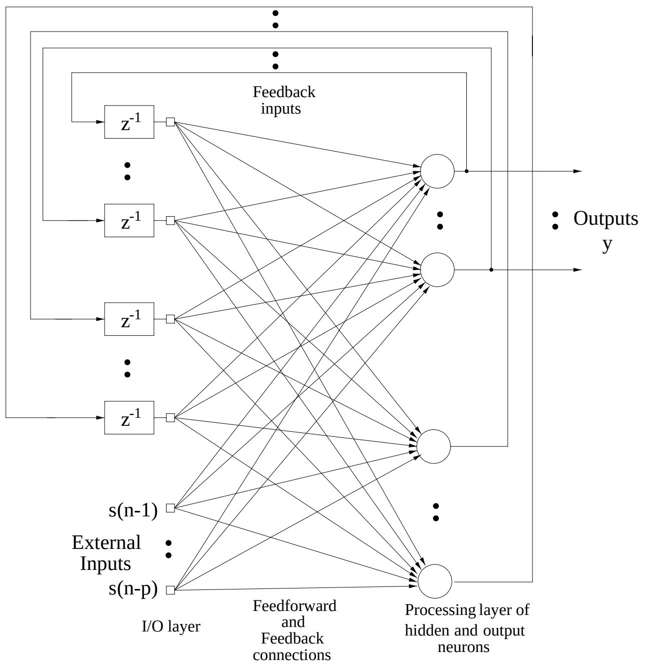
Feedback inputs
z-1
Outputs y
s(n-1)
External Inputs
s(n-p)
I/O layer
Feedforward and Feedback connections
Processing layer of hidden and output neurons

Figure 1: Single recurrent neural network.

Appendix

A.1 RNN and the RTRL Algorithm. The structure of a single RNN is shown in Figure 1. For the kth neuron, its weights form a $(p + F + 1) \times 1$–dimensional weight vector $\mathbf{w}_k^T = [w_{k,1}, \ldots, w_{k,p+F+1}]$, where p is the number of external inputs and F is the number of feedback connections, one remaining element of the weight vector $\mathbf{w}$ being the bias input weight. The feedback connections represent the delayed output signals of the RNN. In the case of the network shown in Figure 1, we have $N = F$. Such a network is called a fully connected recurrent neural network (FCRNN) (Williams & Zipser, 1989). The following equations fully describe the FCRNN:

$$y_k(n) = \Phi(v_k(n)), \quad k = 1, 2, \ldots, N \tag{A.1}$$

$$v_k(n) = \sum_{l=1}^{p+N+1} w_{k,l}(n) u_l(n) \tag{A.2}$$

$$\begin{aligned}\mathbf{u}_i^T(n) = \big[&s(n-1), \ldots, s(n-p), 1,\\ &y_1(n-1), y_2(n-1), \ldots, y_N(n-1)\big],\end{aligned} \tag{A.3}$$

where the $(p + N + 1) \times 1$–dimensional vector $\mathbf{u}$ comprises both the external and feedback inputs to a neuron, with vector $\mathbf{u}$ having "unity" for the constant bias input.

For the nonlinear time-series prediction paradigm, there is only one output neuron of the RNN. RTRL-based training of the RNN is based on minimizing the instantaneous squared error at the output of the first neuron of the RNN (Williams & Zipser, 1989; Haykin, 1994), which can be expressed as

$$\min(e^2(n)) = \min([s(n) - y_1(n)]^2), \tag{A.4}$$

where $e(n)$ denotes the error at the output of the RNN, and $s(n)$ is the teaching signal. Hence, the correction for the lth weight of neuron k at the time instant n can be derived as follows:

$$\begin{aligned}\Delta w_{k,l}(n) &= -\eta \frac{\partial}{\partial w_{k,l}(n)} e^2(n) \\ &= -2\eta e(n) \frac{\partial e(n)}{\partial w_{k,l}(n)}.\end{aligned} \tag{A.5}$$

Since the external signal vector $\mathbf{s}$ does not depend on the elements of $\mathbf{W}$, the error gradient becomes

$$\frac{\partial e(n)}{\partial w_{k,l}(n)} = -\frac{\partial y_1(n)}{\partial w_{k,l}(n)}. \tag{A.6}$$

Using the chain rule, this can be rewritten as,

$$\begin{aligned}\frac{\partial y_1(n)}{\partial w_{k,l}(n)} &= \Phi'(v_1(n)) \frac{\partial v_1(n)}{\partial w_{k,l}(n)} \\ &= \Phi'(v_1(n)) \left(\sum_{\alpha=1}^{N} \frac{\partial y_\alpha(n-1)}{\partial w_{k,l}(n)} w_{1,\alpha+p+1}(n) + \delta_{kl} u_l(n) \right),\end{aligned} \tag{A.7}$$

where

$$\delta_{kl} = \begin{cases} 1, & k = l \\ 0, & k \neq l \end{cases}. \tag{A.8}$$

Under the assumption, also used in the RTRL algorithm (Robinson & Fallside, 1987; Williams & Zipser, 1989; Narendra & Parthasarathy, 1990), that when the learning rate η is sufficiently small, we have

$$\frac{\partial y_\alpha(n-1)}{\partial w_{k,l}(n)} \approx \frac{\partial y_\alpha(n-1)}{\partial w_{k,l}(n-1)}. \tag{A.9}$$

A triply indexed set of variables $\{\pi^j_{k,l}(n)\}$ can be introduced to characterize the RTRL algorithm for the RNN, as

$$\pi^j_{k,l} = \frac{\partial y_j(n)}{\partial w_{k,l}} \quad 1 \le j, k \le N, \quad 1 \le l \le p + 1 + N, \tag{A.10}$$

which is used to compute recursively the values of $\pi^j_{k,l}$ for every time step n and all appropriate j, k, and l as follows,

$$\pi^j_{k,l}(n+1) = \Phi'(v_j)\left[\sum_{m=1}^{N} w_{j,m}(n)\pi^m_{k,l}(n) + \delta_{kj}u_l(n)\right], \tag{A.11}$$

with the values for j, k, and l as in equation A.11 and the initial conditions

$$\pi^j_{k,l}(0) = 0. \tag{A.12}$$

A.2 Proof of the Theorem. From the equivalence in the static sense, the weight update equation for the referent network, can be written as

$$\mathbf{W}^R(n) = \mathbf{W}^R(n-1) + \beta\Delta\mathbf{W}(n), \tag{A.13}$$

which gives

$$\Delta\mathbf{W}^R(n) = \beta\Delta\mathbf{W}(n) = \beta\left(2\eta e(n)\frac{\partial y_1(n)}{\partial \mathbf{W}(n)}\right) = 2\eta\beta e(n)\Pi_1(n), \tag{A.14}$$

where $\Pi_1(n)$ is the matrix of $\pi^1_{k,l}(n)$.

In order to derive the conditions of dynamical equivalence between an arbitrary and the referent RNN, the relationship between the appropriate matrices $\Pi_1(n)$ and $\Pi^R_1(n)$ must be established. That implies that for all the neurons in the RNN, the matrix $\Pi(n)$, which comprises all the terms $\frac{\partial y_j}{\partial w_{k,l}}$, $\forall w_{k,l} \in \mathbf{W}$, $j = 1, 2, \ldots, N$ must be interrelated to the appropriate matrix $\Pi^R(n)$, which represents the referent network.

We shall prove this relationship by induction. For convenience, let us denote $net = \mathbf{w}(n)\mathbf{u}(n)$, and $net^R = \mathbf{w}^R(n)\mathbf{u}(n)$.

Given: $\mathbf{W}^R(n) = \beta\mathbf{W}(n)$ (static equivalence)

$\Phi'(net^R) = \frac{1}{\beta}\Phi'(net)$ (activation function derivative)

$y^R_j(n) = \Phi(net^R) = \Phi(net) = y_j(n), \quad j = 1, \ldots, N$ (activation).

Induction base: The recursion (see equation A.11) starts as

$$(\pi^j_{k,l}(n=1))^R = \Phi'(net^R)\left[\sum_{m=1}^{N} w^R_{j,m}(n=0)\pi^m_{k,l}(n=0)+\delta_{kj}u_l(n=0)\right]$$
$$= \frac{1}{\beta}\Phi'(net)\delta_{kj}u_l(n=0) = \frac{1}{\beta}\pi^j_{k,l}(n=1),$$

which gives $\Pi^R(n=1) = \frac{1}{\beta}\Pi(n=1)$.

Induction step: $(\pi^j_{k,l}(n))^R = \frac{1}{\beta}\pi^j_{k,l}(n)$, and $\Pi^R(n) = \frac{1}{\beta}\Pi(n)$ (assumption)
Now, for the $(n+1)$st step we have:

$$\begin{aligned}(\pi^j_{k,l}(n+1))^R &= \Phi'(net^R)\left[\sum_{m=1}^{N} w^R_{j,m}(n)\pi^m_{k,l}(n) + \delta_{kj}u_l(n)\right]\\ &= \frac{1}{\beta}\Phi'(net)\left[\sum_{m=1}^{N} \beta w_{j,m}(n)\frac{1}{\beta}\pi^m_{k,l}(n) + \delta_{kj}u_l(n)\right]\\ &= \frac{1}{\beta}\pi^j_{k,l}(n+1),\end{aligned}$$

which means that,

$$\Pi^R(n+1) = \frac{1}{\beta}\Pi(n+1).$$

Based on the established relationship and equation A.14, the learning process for the referent RNN can be expressed as

$$\begin{aligned}\Delta\mathbf{W}^R(n) = \beta\Delta\mathbf{W}(n) &= 2\beta\eta e(n)\Pi_1(n)\\ &= 2\beta^2\eta e(n)\Pi^R_1(n) = 2\eta^R e(n)\Pi^R_1(n).\end{aligned} \tag{A.15}$$

Hence, the referent network with the learning rate $\eta^R = \beta^2\eta$ and slope $\beta^R = 1$ is equivalent in the dynamic sense, with respect to the RTRL algorithm, to an arbitrary RNN with slope β, and learning rate η.

Acknowledgments

We acknowledge the contributions of the anonymous reviewers in improving the clarity of the presentation of this work.

References

Haykin, S. (1994). *Neural networks—A comprehensive foundation*. Englewood Cliffs, NJ: Prentice Hall.

Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transaction on Neural Networks, 1*(1), 4–27.

Pineda, F. (1987). Generalization of backpropagation to recurrent neural networks. *Physical Review Letters, 59*, 2229–2232.

Robinson, A. J., & Fallside, F. (1987). *The utility driven dynamic error propagation network* (Tech. Rep. CUED / F–INFENG / TR.1). Cambridge: Cambridge University Engineering Department.

Thimm, G., Moerland, P., & Fiesler, E. (1996). The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation, 8*, 451–460.

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE, 78*(10), 1550–1560.

Williams, R., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation, 1*, 270–280.

Zurada, J. (1992). *Introduction to artificial neural systems*. St. Paul, MN: West Publishing Company.

Received March 11, 1998; accepted October 29, 1998.