ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

On the weight dynamics of recurrent learning

U. D. Schiller and J. J. Steil

Neuroinformatics Group, Bielefeld University, Germany
{uschille,jsteil}@techfak.uni-bielefeld.de, www.jsteil.de

Abstract. We derive continuous-time batch and online versions of the recently introduced efficient $O(N^2)$ training algorithm of Atiya and Parlos [2000] for fully recurrent networks. A mathematical analysis of the respective weight dynamics yields that efficient learning is achieved although relative rates of weight change remain constant due to the way errors are backpropagated. The result is a highly structured network where an unspecific internal dynamical reservoir can be distinguished from the output layer, which learns faster and changes at much higher rates. We discuss this result with respect to the recently introduced "echo state" and "liquid state" networks, which have similar structure.

1 Introduction

Many recurrent neural architectures ranging from fully connected to partially or locally RNNs have been developed for various applications in time-series prediction and generation, speech recognition, adaptive control, or biological modeling. Despite their practical success, one of the main drawbacks is the known high complexity of learning algorithms. Besides the credit assignment problem, these also have to cope with the potential for dynamically rich behavior reaching from globally multistable networks for content addressable memory up to oscillating pattern generators or even provable chaotic networks. Thus there is ongoing research to devise efficient recurrent learning schemes, among which are a large number of approaches to use regularization techniques (Hammer and Steil [2002]).

Some insight in recurrent learning has been gained recently by showing that many of the well known algorithms like RTRL, BPTT can be unified and understood as implementing different ways to minimize the standard quadratic error functional (Atiya and Parlos [2000]). In this context there has also been introduced a new discrete time $O(N^2)$-efficient algorithm, which we will reference as Atiya-Parlos recurrent learning (APRL for short). Seemingly very different approaches independently developed by Jaeger [2001, 2002] under the notion "echo state network" and Natschläger et al. [2002] as "liquid state machine" follow the idea to use a recurrent network as a kind of dynamic reservoir, which can store information about the time development of the inputs and then allows to learn efficiently a linear readout function.

In this contribution we will show that in fact APRL leads to a network structure which is very similar to the idea of using a dynamic reservoir. In Section 2, we derive continuous-time batch and online versions of the algorithm. In Section 3, we prove that the kind of error propagation in Atiya and Parlos [2000] leads to constant rates of weight change. In Section 4, we discuss and compare this result with respect to the "echo state network" and "liquid state machine" concept.

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

2 The Atiya-Parlos continuous time algorithms

For further reference we give a continuous-time version of APRL[1] for the dynamics:

$$\frac{dx}{dt} = -x + Wf(x), \tag{1}$$

where f is applied componentwise to the vector x (or x^T later). It is discretized as

$$x((\hat{k}+1)\Delta t) = (1-\Delta t)x(\hat{k}\Delta t) + \Delta t W f(x(\hat{k}\Delta t)). \tag{2}$$

From this, we get the constraint equation

$$g((\hat{k}+1)\Delta t) \equiv -x((\hat{k}+1)\Delta t) + (1-\Delta t)x(\hat{k}\Delta t) + \Delta t W f(x(\hat{k}\Delta t)) = 0. \tag{3}$$

Let O be the set of output neurons, then the error for target outputs d_j is given by

$$E = \frac{1}{2}\sum_{\hat{k}=1}^{K}\sum_{j\in O}[x_j(\hat{k}\Delta t) - d_j(\hat{k}\Delta t)]^2. \tag{4}$$

In the following we use as time variable $k = \hat{k}\Delta t$ to skip the dependence on Δt and collect the relevant quantities in vectors to write (w_i^T are the rows of W)

$$\mathbf{x} \equiv (x^T(1), \ldots, x^T(K))^T, \mathbf{g} \equiv (g^T(1), \ldots, g^T(K))^T, \mathbf{w} \equiv (w_1^T, \ldots, w_N^T)^T.$$

The key idea of APRL is to interchange the usual role of the states $\mathbf{x}$ and the weights $\mathbf{w}$ by first computing the gradient of E with respect to $\mathbf{x}$:

$$\Delta\mathbf{x} = \frac{\partial E}{\partial \mathbf{x}} = (e(1), \ldots, e(K)), \text{ where } [e(k)]_i = \begin{cases} x_i(k) - d_i(k) & i \in O \\ 0 & \text{otherwise} \end{cases}.$$

Note that **x**, **g** and **w** defined above are column vectors, while $\Delta\mathbf{x}$ is a row vector. In a second step, the constraint equation (3) is used to compute weight changes $\Delta\mathbf{w}$ which generate a small step along $-\Delta\mathbf{x}$:

$$\frac{\partial g}{\partial \mathbf{w}}\Delta\mathbf{w} \approx \eta\frac{\partial g}{\partial \mathbf{x}}\Delta\mathbf{x}^T. \tag{5}$$

This equation is solved using a pseudo-inverse for $\partial g/\partial \mathbf{w}$ and yields the training rule

$$\Delta\mathbf{w}_{batch} = \eta\left[\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)\right]^{-1}\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Delta\mathbf{x}^T. \tag{6}$$

In line with Atiya and Parlos [2000], equation (6) can be reformulated as

$$\Delta W_{batch}(K) = \frac{\eta}{\Delta t}B(K)V^{-1}(K), \tag{7}$$

[1] Though straightforward to derive (Atiya and Parlos [2000]), a continuous-time version of APRL has to the best of our knowledge not yet been published elsewhere.

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

using $D(k) = diag(f'(x_i(k)))$ and the definitions

$$V(K) = \sum_{k=0}^{K-1} f(x(k)) f(x^T(k)), \tag{8}$$

$$B(K) = \sum_{k=1}^{K} \gamma(k) f(x^T(k-1)), \tag{9}$$

$$\gamma(k) = -e^T(k) + [(1-\Delta t)I + \Delta t W D(k-1)]\, e^T(k-1). \tag{10}$$

Here $\gamma(k) = (\nabla_{\mathbf{x}} g(k))\Delta \mathbf{x}^T$. Applying the same manipulations and using a numerical approximation for the inversion of $V(k)$ as in Atiya and Parlos [2000], we get the on-line algorithm:

1. initialize x(0) to random values
2. $\mathbf{k=1}$**:** compute $x(1)$ according to (2), $\gamma(1) = -e^T(1)$, $B(1) = \gamma(1) f(x^T(0))$,

$$V^{-1}(1) = I/\epsilon - f(x(0)) f(x^T(0))/[\epsilon^2 + \epsilon f(x^T(0)) f(x(0))], \quad 1 >> \epsilon > 0,$$
$$W(1) = W(0) + \Delta W(1) = W(0) + \frac{\eta}{\Delta t} B(1) V^{-1}(1).$$

3. $\mathbf{k=k+1}$**:** compute $x(k)$ according to (2),

$$\gamma(k) = -e^T(k) + [(1-\Delta t)I + \Delta t W D(k-1)]\, e^T(k-1)$$
$$\Delta W(k) = \frac{\eta}{\Delta t} \frac{\left(\gamma(k) - B(k-1)V^{-1}(k-1) f(x(k-1))\right)[V^{-1}(k-1) f(x(k-1))]^T}{1 + f(x^T(k-1))V^{-1}(k-1) f(x(k-1))}$$
$$W(k) = W(k-1) + \Delta W(k), \quad B(k) = B(k-1) + \gamma(k) f(x^T(k-1))$$
$$V^{-1}(k) = V^{-1}(k-1) - \frac{V^{-1}(k-1) f(x(k-1))[V^{-1}(k-1) f(x(k-1))]^T}{1 + f(x^T(k-1))V^{-1}(k-1) f(x(k-1))}$$

4. Repeat 3.) until end of data.

3 Analysis of the weight dynamics

In the following, we consider the case of only one output neuron, say x_1. It turns out that the weight updates in the non-output part of the weight matrix scale equally and with constant rate in every column. The scaling factors are proportional to the weights in the first column. Formally, this is stated as

$$\forall i > 1, j > 1 : \Delta W_{ik}(K) = \frac{W_{i1}(0)}{W_{j1}(0)} \Delta W_{jk}(K). \tag{11}$$

Lemma: Let $i > 1, j > 1$. Then $\forall K$

$$\gamma_i(K) = \frac{W_{i1}(0)}{W_{j1}(0)} \gamma_j(K), \tag{12}$$

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

$$B_{ik}(K) = \frac{W_{i1}(0)}{W_{j1}(0)} B_{jk}(K), \tag{13}$$

$$W_{i1}(K) = \frac{W_{i1}(0)}{W_{j1}(0)} W_{j1}(K). \tag{14}$$

Note that $W_{j1}(0) \neq 0$ can be assured by the initialization of the weights.

Proof: For one output neuron, only the first component of the error vector $e(K)$ is non-zero:

$$\forall i > 1 \forall K : e_i(K) = 0. \tag{15}$$

We show 11 to 14 by induction on the time step K.
K=1: Let $i > 1, j > 1$. Then

$$\gamma_i(1) = -e_i^T(1) = 0 = -e_j^T(1) = \gamma_j(1)$$

$$B_{ik}(1) = \gamma_i(1) f(x_k^T(0)) = 0 = \gamma_j(1) f(x_k^T(0)) = B_{jk}(1)$$

$$\Delta W_{ik}(1) = \frac{\eta}{\Delta t} \sum_h B_{ih}(1) V_{hk}^{-1}(1) = 0 = \frac{\eta}{\Delta t} \sum_h B_{jh}(1) V_{hk}^{-1}(1) = \Delta W_{jk}(1)$$

$$\begin{aligned} W_{i1}(1) = W_{i1}(0) + \Delta W_{i1}(1) = W_{i1}(0) &= \frac{W_{i1}(0)}{W_{j1}(0)} W_{j1}(0) \\ &= \frac{W_{i1}(0)}{W_{j1}(0)} (W_{j1}(0) + \Delta W_{j1}(1)) = \frac{W_{i1}(0)}{W_{j1}(0)} W_{j1}(1) \end{aligned}$$

K $\Rightarrow$ K+1: Let $i > 1, j > 1$.

$$\begin{aligned} \frac{\gamma_i(K+1)}{\gamma_j(K+1)} &= \frac{-e_i^T(K+1) + \sum_h [(1-\Delta t) I + \Delta t W(K) D(K)]_{ih} e_h^T(K)}{-e_j^T(K+1) + \sum_h [(1-\Delta t) I + \Delta t W(K) D(K)]_{jh} e_h^T(K)} \\ &= \frac{\Delta t W_{i1}(K) f'(x_1(K)) e_1^T(K)}{\Delta t W_{j1}(K) f'(x_1(K)) e_1^T(K)} = \frac{W_{i1}(K)}{W_{j1}(K)} = \frac{W_{i1}(0)}{W_{j1}(0)} \end{aligned}$$

If $e_1(K) = 0$, then $\gamma_i(K+1) = \gamma_j(K+1) = 0$ and 12 trivially holds.

$$\begin{aligned} \frac{B_{ik}(K+1)}{B_{jk}(K+1)} &= \frac{B_{ik}(K) + \gamma_i(K+1) f(x_k^T(K))}{B_{jk}(K) + \gamma_j(K+1) f(x_k^T(K))} \\ &= \frac{\frac{W_{i1}(0)}{W_{j1}(0)} (B_{jk}(K) + \gamma_j(K+1) f(x_k^T(K)))}{B_{jk}(K) + \gamma_j(K+1) f(x_k^T(K))} = \frac{W_{i1}(0)}{W_{j1}(0)} \end{aligned}$$

If $B_{jk}(K+1) = 0$ then also $B_{ik}(K+1) = 0$ and 13 holds.

$$\begin{aligned} \frac{\Delta W_{ik}(K+1)}{\Delta W_{jk}(K+1)} &= \frac{(\gamma_i(K+1) - \sum_h B_{ih}(K) [V^{-1}(K) f(x(K))]_h) [V^{-1}(K) f(x(K))]_k^T}{(\gamma_j(K+1) - \sum_h B_{jh}(K) [V^{-1}(K) f(x(K))]_h) [V^{-1}(K) f(x(K))]_k^T} \\ &= \frac{\frac{W_{i1}(0)}{W_{j1}(0)} (\gamma_j(K+1) - \sum_h B_{jh}(K) [V^{-1}(K) f(x(K))]_h)}{\gamma_j(K+1) - \sum_h B_{jh}(K) [V^{-1}(K) f(x(K))]_h} = \frac{W_{i1}(0)}{W_{j1}(0)} \end{aligned}$$

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

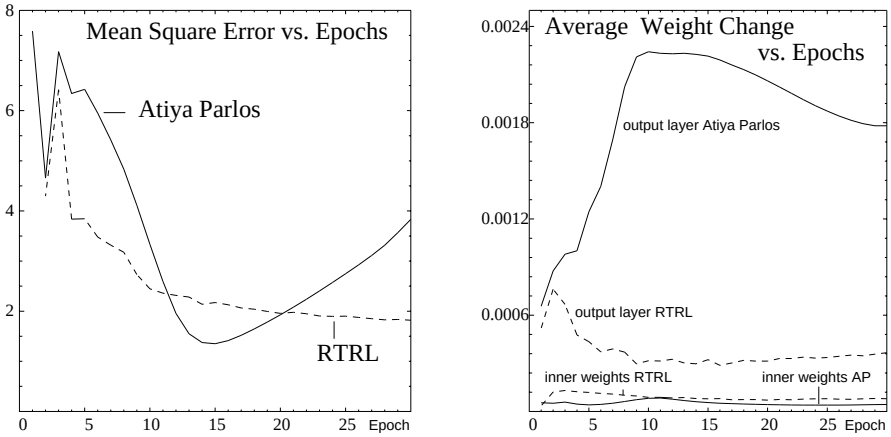

Figure 1: Left: The respective errors show a clear minimum after 15 epochs for APRL, whereas RTRL continuously improves. Right: Rates of change of the output vs. internal weights for RTRL (dashed) and the Atiya-Parlos algorithm avaraged over respective epochs. The learning rate was 0.05 for RTRL and 1.0 for APRL.

If $\Delta W_{jk}(K+1) = 0$ then also $\Delta W_{ik}(K+1) = 0$ and 11 holds. Finally

$$\frac{W_{i1}(K+1)}{W_{j1}(K+1)} = \frac{W_{i1}(K) + \Delta W_{i1}(K+1)}{W_{j1}(K) + \Delta W_{j1}(K+1)}$$
$$= \frac{\frac{W_{i1}(0)}{W_{j1}(0)}(W_{j1}(K) + \Delta W_{j1}(K+1))}{W_{j1}(K) + \Delta W_{j1}(K+1)} = \frac{W_{i1}(0)}{W_{j1}(0)}$$

If $W_{j1}(K+1) = 0$ then also $W_{i1}(K+1) = 0$ and hence 14 holds. Hence we have shown by induction, that 11 to 14 hold.

In summary, the above result shows that there are two different groups of weights: those who connect arbitrary neurons to the output neuron and which may arbitrarily change and the majority of weights interconnecting the inner neurons (reservoir neurons), whose rates of change are systematically coupled and determined beforehand by the initialization. This result holds for the case of a single output neuron and does not easily generalize to more than one output.

An explanation for this sort of linear learning (scaling) in the reservoir can be found in equation (10), which shows the special way how errors $e(k)$ are propagated. As APRL relies on evaluating the gradients of the constraint equation (3), the errors enter in (10) only partially, once to influence the change of the output rates by means of $-e^T(k)+(1-\Delta t)Ie^T(k-1)$ and as part of a scaling factor $\Delta t W D(k-1)e^T(k-1)$ for the reservoir. This factor can be interpreted easily, the magnitude of the reservoir scaling in the i–th column is proportional to the error in the last time-step times $[D(k-1)]_{11} = f'(x_1(k-1))$, which is small for activations x_1 close to saturation. This keeps a balance between changes and the need to keep activations in a working area around zero. In further time-steps $k+s, s>1$ the error $e(k)$ is not relevant.

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 73-78

4 Discussion

We compared the online variant of APRL with the online variant of RTRL from Pearlmutter [1995]. The task was to learn an operator implicitly given by the Roessler dynamics and is described in more detail in Steil and Ritter [1999]. The parameters of the respective algorithms where chosen such that a comparable behavior in the mean square error was achieved (Fig. 1 (left)). As in the discrete time case Atiya and Parlos [2000], the learning rate for APRL can be chosen much higher than that for RTRL. As expected, an analysis of the rates of change reveals that the output weights change much more rapidly than the weights connecting the inner neurons (Fig. 1 (right)).

The partitioning of the weights in fast and uncoupled learning output weights and coupled inner connections shows that the Atiya-Parlos approach is based on a functional division between a (linear) readout layer and a larger dynamical reservoir of inner neurons. As discussed above, the output errors are not directly propagated to the reservoir and rather determine only the magnitude of the scaling factors, which are predefined by the initialization. From this observation we expect that APRL in its current form should perform much worse in tasks with long term dependencies only, at least as long as the reservoir is relatively small. Further, APRL should be more sensitive to local deviations and indeed our experiments showed problems, when there were long transients to the attractor. We also found that the initialization is a relevant performance factor and, though always good minima were found, not always the best error achieved was comparable to RTRL. On the other hand, we believe that the fast learning at low complexity $O(n^2)$ motivates further investigations on this algorithm.

Obviously the data driven and coupled scaling of that initially randomly connected reservoir provides enough memory to allow successful function approximation. This exactly resembles the functional modules of the “echo state” or “liquid state” networks. However, there are two major differences which may provide also more insights to these approaches: (i) there is no need for a “big” network (≈ 100 neurons) to provide the necessary reservoir, (ii) the functional specialization is generated by the error backpropagation of the Atiya-Parlos algorithm itself based on the standard error function and the constraint equation. Future investigations may be conducted with respect to whether the “echo state” and “liquid state” networks may as well be interpreted as an efficient minimization with respect to a standard error function.

References

A. B. Atiya and A. G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Trans. Neural Networks*, 11(9):697–709, 2000.

B. Hammer and J. J. Steil. Tutorial: Perspectives on learning with recurrent neural networks. In *Proc. of ESANN 2002*, pages 357–368, 2002.

H. Jaeger. The ”echo state” approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, 2001.

H. Jaeger. Adaptive nonlinear system identification with echo state networks. In *NIPS*, 2002.

T. Natschläger, W. Maass, and H. Markram. The ”liquid computer”: A novel strategy for real-time computing on time series. *TELEMATIK*, 8(1):39–43, 2002.

B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Tansactions on Neural Networks*, 6(5):1212–1228, 1995.

J. J. Steil and H. Ritter. Recurrent learning of input-output stable behaviour in function space: A case study with the Roessler attractor. In *Proc. ICANN 99*, pages 761–766. IEE, 1999.