Analyzing the weight dynamics of recurrent learning algorithms

Ulf D. Schiller and Jochen J. Steil

Neuroinformatics Group, Faculty of Technology, Bielefeld University

Abstract

We provide insights into the organization and dynamics of recurrent online training algorithms by comparing real time recurrent learning (RTRL) with a new continuous-time online algorithm. The latter is derived in the spirit of a recent approach introduced by Atiya and Parlos [1], which leads to non-gradient search directions. We refer to this approach as Atiya-Parlos learning (APRL) and interpret it with respect to its strategy to minimize the standard quadratic error. Simulations show that the different approaches of RTRL and APRL lead to qualitatively different weight dynamics. A formal analysis of the one-output behavior of APRL further reveals that the weight dynamics favor a functional partition of the network into a fast output layer and a slower dynamical reservoir, whose rates of weight change are closely coupled.

Key words: recurrent network, recurrent learning, weight dynamics, gradient descent

1 Introduction

Recurrent neural networks (RNNs) are attractive tools for tasks of sequential nature like time-series prediction, sequence generation, speech recognition, or adaptive control. Many architectures ranging from fully connected to partially or locally RNNs have been developed. Although their application is successful in practice, the dynamical behavior of the networks leads to high complexity of the algorithms. The credit assignment problem and the potentially rich dynamical properties of the networks make it difficult to devise efficient recurrent

Email addresses: uschille@techfak.uni-bielefeld.de (Ulf D. Schiller), jsteil@techfak.uni-bielefeld.de (Jochen J. Steil).

URLs: www.ulfschiller.de (Ulf D. Schiller), www.jsteil.de (Jochen J. Steil).

learning schemes. Contemporary approaches frequently employ regularization techniques to tackle these problems [2]. The analysis of dynamical properties of recurrent networks and the corresponding learning algorithms nevertheless remains a challenging field of research.

Some encouraging results are available with respect to formulate a common unifying framework for gradient based techniques like real time recurrent learning (RTRL) or backpropagation through time (BPTT). Atiya and Parlos [1] have shown that these can be derived from a constrained optimization problem which combines the quadratic error with constraints reflecting the network dynamics. They also introduced a new strategy to minimize the standard error which employs search directions different from the gradient. It considers the states as control variables and computes weight updates to achieve a targeted change in the states variables. We reference to this strategy as Atiya-Parlos learning (APRL). In [1], an $\mathcal{O}(n^2)$-efficient APRL algorithm was given for discrete networks, while we will introduce below an APRL algorithm for continuous-time networks. We also show that APRL can be interpreted as a truncated "one-step-backward" propagation of the instantaneous error combined with a momentum term providing the necessary dynamic memory.

The existence of such different approaches as RTRL and APRL to minimize the standard quadratic error function along different search directions motivates further investigations on whether and how the resulting weight dynamics differ. This is especially interesting for online trajectory learning, because in this case the input data are highly correlated. In this case the gradient direction becomes a well founded heuristics because it does not implement a stochastic gradient descent any more and rather the combined dynamic of error function and weights determine the learning success. The obvious problem is that direct access to the dynamics of online learning on the high dimensional error surface is not possible for two main reasons: because of the multiple constraints acting through the network dynamics and the complex dependency of the time-varying error surface on the network parameters.

Therefore, we use a comparative approach to investigate the Atiya-Parlos methodology and its weight dynamics as opposed to results for RTRL [3] on two typical tasks. Simulations and theoretical results show that the two algorithms behave quite different and yield evidence that their different ways to minimize the error lead to unrelated weight dynamics. This point of view is further supported by a formal treatment of the one-output case of APRL which reveals a functional division of the network which resembles more the "echo state network" [4,5] and "liquid state machine" approaches [6,7] than classical backpropagation.

The remainder of this work is organized as follows: In section 2 we give a continuous-time online algorithm based on the APRL paradigm [8] and inter-

pret the algorithm with respect to its strategy to minimize the error function. In section 3 we present simulation results from APRL and RTRL and analyze the weight dynamics of the algorithms. In section 4 we turn to the one-output case and prove that APRL leads to a very special weight dynamics that results in a functional partition of the network. In section 5 we summarize the results and discuss the insights gained hereby.

2 Atiya-Parlos Recurrent Learning (APRL)

For further reference, we give a continuous-time algorithm which is derived with the APRL learning approach in a straightforward way and similar to the treatment in [1] for discrete networks [1] . We consider the dynamics

$$\frac{\mathrm{d}\tilde{x}}{\mathrm{d}t} = -\tilde{x} + \mathsf{W} f(\tilde{x}) \tag{1}$$

where $\mathsf{W} = (w_{ij})$ is the weight matrix and a sigmoid $f(\cdot)$ is applied componentwise to its argument $\tilde{x}$ (or x, x^T later). Equation (1) is discretized as

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

where $x(k) = \tilde{x}(k\Delta t)$ to skip the dependence of the time variable on Δt. Let O be the set of output neurons, then recurrent learning aims at minimizing the quadratic error

$$E = \frac{1}{2} \sum_{k=1}^{K} \sum_{i \in O} \left[x_i(k) - d_i(k) \right]^2 . \tag{3}$$

The standard approach to solve this problem is gradient descent in weight space, which means to adapt the weights by a small step $-\eta \nabla_w E$ in the direction of the negative gradient of the error function. However, with APRL we compute a different search direction below.

From a formal point of view the training problem can be formulated as a constrained optimization problem, where the discretized dynamics (2) yields for each $k = 0, \ldots, K-1$ the constraint equation

$$g(k+1) \equiv -x(k+1) + (1 - \Delta t)x(k) + \Delta t \mathsf{W} f(x(k)) = 0. \tag{4}$$

[1] In [1], it has been proposed (but not carried out) to derive a continuous version of APRL based on outputs $y = f(x)$ and $E = \sum_{k=1}^{K} \left[y_i(k) - d_i(k) \right]^2$. This leads to slightly different formulas and is not always suitable, e.g. for identification tasks, where the amplitude of d_i may not fully be known in advance. The basic framework, however, remains identical.

The goal is to minimize E subject to $g(k+1) = 0$, $k = 0, \ldots, K-1$. In the following, we collect the relevant quantities in vectors to write

$$\begin{aligned} \mathbf{x} &\equiv (x^T(1), \ldots, x^T(K))^T, \\ \mathbf{g} &\equiv (g^T(1), \ldots, g^T(K))^T, \\ \mathbf{w} &\equiv (w_1^T, \ldots, w_N^T)^T, \end{aligned} \tag{5}$$

where w_i^T are the rows of W.

For gradient descent in weight space, the weights are considered the control variables. The key idea of the APRL approach is to interchange the role of the weights $\mathbf{w}$ and the states $\mathbf{x}$. Then the states are considered as the control variables and a small step $\eta \Delta \mathbf{x}_{\text{tar}}$ in the direction of the negative gradient of the error with respect to the states is targeted:

$$\Delta \mathbf{x}_{\text{tar}} = -\left(\frac{\partial E}{\partial \mathbf{x}}\right)^T = -\left(e^T(1), \ldots, e^T(K)\right)^T, e_i(k) = \begin{cases} x_i(k) - d_i(k), & i \in O \\ 0, & i \notin O. \end{cases}$$

Because the constraint equation (4) equals zero, we have

$$\frac{\partial g}{\partial \mathbf{w}} \Delta \mathbf{w} + \frac{\partial g}{\partial \mathbf{x}} \Delta \mathbf{x} = 0 \quad \Rightarrow \quad \frac{\partial g}{\partial \mathbf{w}} \Delta \mathbf{w} = -\frac{\partial g}{\partial \mathbf{x}} \Delta \mathbf{x}. \tag{6}$$

and can try to compute weight changes $\Delta \mathbf{w}$ which (approximately) lead to the targeted step $\Delta \mathbf{x}$. We call this strategy of APRL *virtual teacher forcing*, as the desired target states are used to compute the weight changes but never are really fed into the network like in traditional teacher forcing approaches.

A least squares solution of (6) using the pseudo-inverse of $\partial g / \partial \mathbf{w}$ leads to the Atiya-Parlos learning rule for a step $\Delta \mathbf{x} = \eta \Delta \mathbf{x}_{\text{tar}}$:

$$\Delta \mathbf{w}_{\text{batch}} = -\eta \left[\left(\frac{\partial g}{\partial \mathbf{w}}\right)^T \left(\frac{\partial g}{\partial \mathbf{w}}\right) \right]^{-1} \left(\frac{\partial g}{\partial \mathbf{w}}\right)^T \frac{\partial g}{\partial \mathbf{x}} \Delta \mathbf{x}_{\text{tar}} = -\eta \left(\frac{\partial g}{\partial \mathbf{w}}\right)^{\#} \frac{\partial g}{\partial \mathbf{x}} \Delta \mathbf{x}_{\text{tar}}. \tag{7}$$

As pointed out in [1], the batch update defines an overdetermined system and the resulting accumulated weight changes $\Delta \mathbf{w}_{\text{batch}}$ therefore minimize the sum of square error of the residual of the target errors.

The learning rule (7) can be reformulated in the matrix notation of the weights W. In this representation, it yields the batch update $\Delta \mathsf{W}_{\text{batch}} = \Delta \mathsf{W}(K)$ after presenting all data points $(x(k), d(k))$, $k = 1, \ldots, K$. More details are given in Appendix A.

2.1 Online algorithm

To turn (7) into an online algorithm, $\Delta\mathsf{W}(K)$ can be split into the part for the first $K-1$ data points and the online update $\Delta\mathsf{W}_{\text{online}}(K)$:

$$\Delta\mathsf{W}_{\text{batch}}(K) = \Delta\mathsf{W}_{\text{batch}}(K-1) + \Delta\mathsf{W}_{\text{online}}(K) = \sum_{k=1}^{K} \Delta\mathsf{W}_{\text{online}}(k). \quad (8)$$

As shown in Appendix A the matrix $\Delta\mathsf{W}_{\text{online}}(k)$ can be further split into

$$\Delta\mathsf{W}_{\text{online}}(k) = \Delta\mathsf{W}_{\text{inst}}(k) - \Delta\mathsf{W}_{\text{batch}}(k\text{–}1)\left[I - \mathsf{V}(k\text{–}1)\mathsf{V}^{-1}(k)\right], \quad (9)$$

where $\mathsf{V}(k) = \sum_{s=1}^{k} f(x(s))f(x^T(s)) + \epsilon I$, ($\epsilon > 0$ for regularization, I is the identity matrix). The first term $\Delta\mathsf{W}_{\text{inst}}(k)$ is the matrix corresponding to the one-step solution $\Delta\mathbf{w}_{\text{inst}}(k)$ of (7) for the instantaneous error

$$\Delta\mathbf{w}_{\text{inst}}(k) = -\eta \left(\frac{\partial g}{\partial \mathbf{w}}\right)^{\#} \frac{\partial g}{\partial \mathbf{x}} \Delta\mathbf{x}(k), \quad (10)$$

where $\Delta\mathbf{x}(k) = (0, \ldots, 0, e(k)^T, 0, \ldots, 0)^T$ and $e(k') = 0$ for $k' \neq k$. The second is a momentum term which includes the accumulated updates up to step k and decays due to $\mathsf{V}(k-1)\mathsf{V}^{-1}(k) \overset{k\to\infty}{\longrightarrow} I$.

2.2 Interpretation of the online algorithm

In (9), the online update rule is composed of an instantaneous weight update and a momentum term. The first is the solution of (7) for the single step error $\Delta\mathbf{x}(k)$, which in this case is an underdetermined system. The procedure is illustrated schematically in Figure 1, where we regard the error surface at time k dependent of the states as well as on the weights, i.e. consider an error $\mathbf{P}$ in $(\mathbf{x}, \mathbf{w}, E(\mathbf{x}, \mathbf{w}))$-space. The virtual teacher forcing states $\Delta\mathbf{x}$ are computed with respect to the projection of E on the $(E, \mathbf{x})$-plane, then this gradient defines a back projection to the $(E, \mathbf{w})$-plane, where among multiple solutions for $\Delta\mathbf{w}$ which are consistent with the constraints the one with minimal norm is chosen. The weight update $\Delta\mathbf{w}$ may lead to downhill or uphill steps on the original error surface, especially as the number of degrees of freedom for the solution of (7), is in principle [2] n^2 with respect to less than n targeted $\Delta\mathbf{x}(k)$ per time step (often even only one target output is available). The minimum norm solution picked by the pseudo-inverse is not directly sensitive to the error and may not be the best choice and/or the dynamic constraints can enforce quite large steps in the weight space.

[2] In Section 4 we will show that de facto it is only $2n$ in the one-output case.

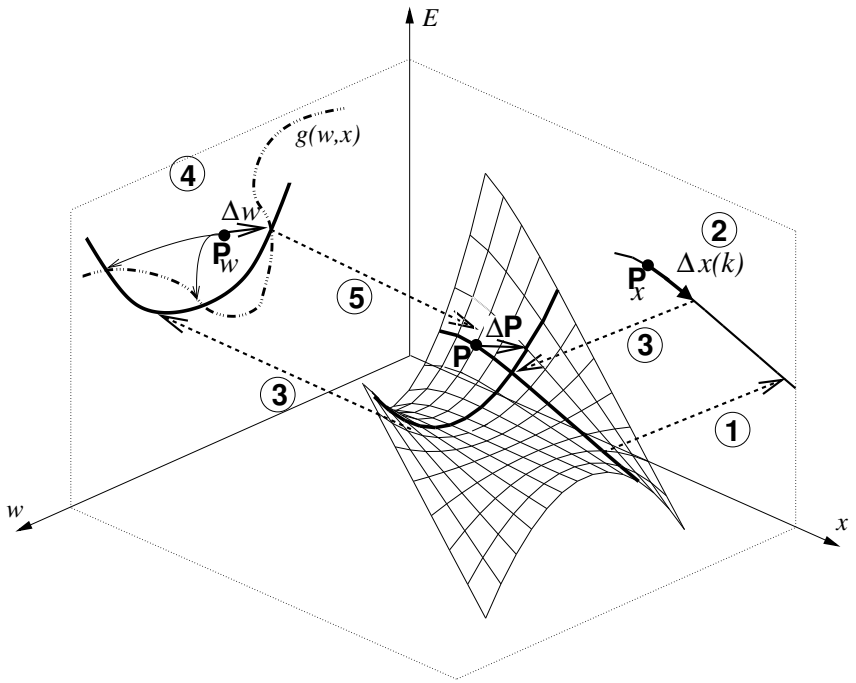

Fig. 1. At time k APRL online learning (9) proceeds starting at a point P on the error surface $E(x, w)$. The E is projected to the (E, x)-plane (1), where gradient decent computes the targeted $\Delta x(k)$ (2). These targets define a back projection to the (E, w)-plane (3), where a solution for Δw is picked which is consistent with the constraints (the recurrent dynamics) and has minimal norm (because the pseudo-inverse is used) (4). Finally, Δw leads to a step ΔP (5). The decaying momentum term modulates this step (not shown).

The momentum term in (9) contributes relative to $\mathsf{V}(k-1)\mathsf{V}^{-1}(k) \stackrel{k\to\infty}{\longrightarrow} I$ and decays in the long run. This results in a potential high sensitivity to initial transients, which we indeed observe in the simulations below. The momentum decay also shows that APRL, though not following the gradient, suffers from fading memory like gradient algorithms from the vanishing gradient (cf. [9]).

3 Simulation results: APRL vs. RTRL

From the previous discussion we expect that simulations may show substantial performance differences between APRL and RTRL, in particular for online learning. Below we consider two tasks to illustrate such differences: the popular Rössler attractor [10], which is mildly chaotic, has a strange attractor limit set, and has been shown to be learnable by relatively small recurrent networks, and secondly the well known Mackey-Glass system.

For the Rössler system we learn the operator which maps coordinate functions of the Rössler dynamics onto each other as described in more detail in [11], a

	steps	min training error		training error after 100 epochs		epoch with min error		generaliz. at min		generaliz. after 100 epochs	
		avg	std	avg	std	avg	std	avg	std	avg	std
Rössler, 10 neurons, time step 0.1											
	1000	0.0650	0.0018	0.0659	0.0018	89.30	11.64	0.0183	0.0029	0.0183	0.0037
RTRL	1019	0.0234	0.0055	0.0278	0.0030	22.50	28.55	0.0099	0.0086	0.0216	0.0068
$\eta = 0.05$	1020	0.0292	0.0014	0.0311	0.0009	42.50	41.60	0.0095	0.0052	0.0201	0.0061
	1250	0.0032	0.0013	0.0035	0.0013	74.60	22.28	0.0017	0.0009	0.0018	0.0009
	1430	0.0186	0.0004	0.0189	0.0005	72.20	29.99	0.0088	0.0020	0.0109	0.0031
	1000	0.0536	0.0243	0.7873	0.2365	19.30	10.45	0.1234	0.3224	0.9184	0.2679
APRL	1019	0.0094	0.0160	0.0887	0.2534	81.80	31.21	0.0098	0.0232	0.1154	0.3394
$\eta = 1$	1020	0.0429	0.0380	0.6425	0.3192	27.20	24.36	0.2048	0.4339	0.8682	0.4401
	1250	0.0016	0.0002	0.0016	0.0002	77.50	34.21	0.0003	0.0003	0.0003	0.0003
	1430	0.0463	0.0453	0.6498	0.3207	26.00	30.58	0.0630	0.0936	0.8604	0.4272
Mackey-Glass, 40 neurons, time step 0.2											
RTRL	200	0.1730	0.0001	0.1730	0.0001	50.00	0.00	0.1410	0.0001	0.1410	0.0001
$\eta = 0.05$	500	0.1417	0.0003	0.1429	0.0001	13.50	2.80	0.2054	0.0048	0.1848	0.0003
APRL	200	0.0592	0.0103	0.0592	0.0103	50.00	0.00	0.1732	0.0445	0.1732	0.0445
$\eta = 1$	500	0.0979	0.0072	0.0979	0.0072	50.00	0.00	0.1550	0.0118	0.1550	0.0118

Table 1
Simulation results. Different numbers of learning steps per epoch (column 1) cause displacements of the trajectory between consecutive epochs and thus different transients. All data are averaged over ten runs of 100 epochs, errors are given as NMSE.

task of medium difficulty. The coupled differential equations are

$$\dot{x} = -y - z, \quad \dot{y} = x + 0.2y, \quad \dot{z} = 0.2 + xz - 5.7z. \tag{11}$$

We obtain the coordinate functions by a fourth order Runge-Kutta integration with initial conditions $(0.495, -0.166, -0.3)$ and use – a relatively small number – of ten steps to integrate from $t \to t + 1$. The coordinates x and z are used as inputs for the network, and y is the target output, all scaled by a factor $\frac{1}{10}$ into a suitable range for the network. Note that the initial conditions are outside the attracting set such that the initial transients play a significant role, see Fig. 2.

The Mackey-Glass equation with parameters for a mildly chaotic system

$$\dot{y}(t) = -0.1y(t) + \frac{0.2y(t-17)}{1 + y(t-17)^{10}} \tag{12}$$

is integrated from $k \to k+1$ using 30 Runge-Kutta 4-th order steps. Network inputs are $y(k), y(k-6), y(k-12), y(k-18)$ while the target is $y(k+84)$.

In all cases the networks are trained in epochs on different numbers of K steps (=data points) by applying at each step the online update for RTRL (see [3]) or APRL from (9), respectively. Then training is restarted at the first step while the trained network is kept such that in training epochs the networks needs transients to recover to the restarted trajectory. For generalization, the initial state is the last point of the previous training epoch, which after training is close to the desired trajectory such that no transients are present and hence

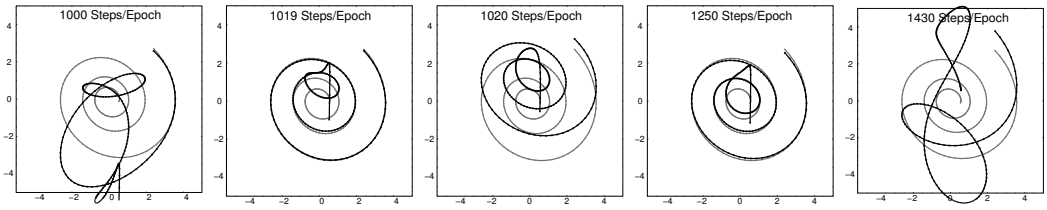

Fig. 2. Transient behavior at the beginning of a training epoch for the Rössler attractor (projected to the xy-space). While the x (and z)-coordinate is given as input, the y-coordinate is output of the network. The darker line is the target trajectory, the brighter line is the network prediction.

generalization errors are sometimes smaller than training errors.

The time steps and number of neurons are given in Table 1, the learning rates are $\eta = 0.05$ for RTRL and $\eta = 1$ for APRL. The APRL regularization parameter is $\epsilon = 1$ for the Rössler dynamics and $\epsilon = 0.002$ for the Mackey-Glass system. Initial weights are generated randomly and uniformly distributed in $[-0.2, 0.2]$. Errors are given as NMSE $= \langle (x(k) - d(k))^2 \rangle / \sigma^2$, where σ^2 is the standard deviation of the reference signal $d(k)$ and $x(k)$ is the network output.

For RTRL in the majority of all cases the training error gradually improves as training proceeds though overfitting occurs in some cases. The differences for the varying number of learning steps per epoch show that initial transients influence the errors, but mainly appear as a bias while the overall learning performance is not perturbed. RTRL turns out to be robust and yields good training and generalization errors.

The results in Table 1 show that APRL achieves training errors comparable to RTRL, but the minima are reached faster. Single trials reach the minimal training errors often after less than 15 epochs. This confirms the results of Atiya and Parlos [1] and shows that APRL with high learning rates is an efficient alternative to RTRL and leads to good training errors and generalization, respectively.

Influence of transients and overshoot

The results for APRL on the Rössler system show the influence of the initial transients. Because we start a new training epoch at the end of the last epoch, the starting points result in quite different transients even for the small displacement between 1019 and 1020 steps as shown in Fig. 2. Especially interesting is also the average epoch in which the minimal error is reached for 1019 and 1020 in Table 1. While for 1019 steps typically the error gradually decreases and the best performance is achieved after many epochs (avg ≈ 82), for 1020 steps the minimum is reached earlier and in later epochs an error over-

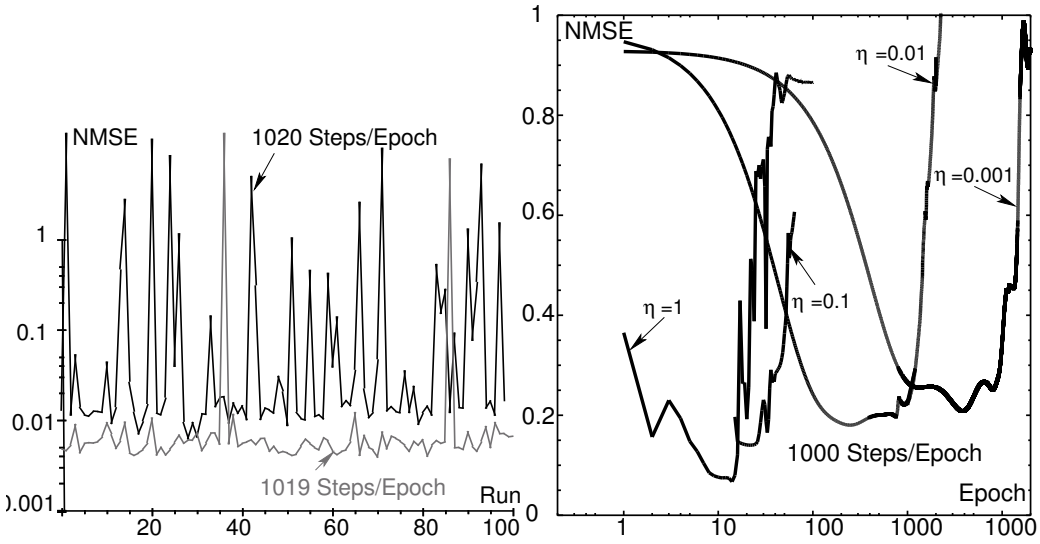

Fig. 3. Left: Final errors for 100 runs of 100 epochs for training 1019 or 1020 points of the Rössler system. Right: Error curves for APRL with reduced learning rates. The respective learning rates are indicated at the curves.

shoot occurs, which often finally drives the network out of the working region. After 1019 steps, the first error signal in the new epoch in most cases induces a $\Delta x(1)$ in the correct positive y-direction and the transient is fairly short, after 1020 steps, the first error signal $\Delta x(1)$ points in the negative y-direction and the transients are more erratic. Though in both cases the network comes back to the correct trajectory and small errors, the influence of the transient persists and causes the significant difference in the long term performance. In Fig. 3 the final errors for full 100 runs of 100 epochs are shown in both cases.

In order to investigate whether error overshoot is due to large learning rates, we reduced the rate to $\eta = 0.1$, $\eta = 0.01$ and $\eta = 0.001$ (1/1000 of the standard rate for APRL) from the beginning of the training (here for 1000 steps). Results are shown in Fig. 3. An improvement of the overall training error is not observed, however, the minimum is reached later, for $\eta = 0.001$ after 3829 epochs.

The sensitivity to transients of APRL is plausible from the characteristic momentum term in (9). Therefore we conclude that the error overshoot is a characteristic feature of APRL which is caused by the transients and therefore problem specific. We did not observe such problems for the Mackey-Glass system, where APRL yields excellent results and performed very stably. Our experience with error overshoot is in line with [1], where it was already noticed that after the training minimum the error increases again and learning has to be stopped at the minimum.

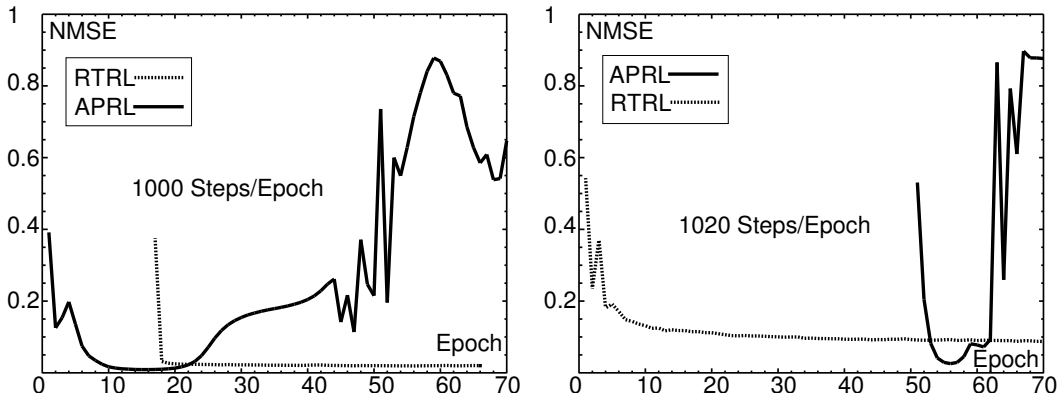

Fig. 4. Error curves for switching the algorithms while learning the Rössler attractor. Left: RTRL applied after APRL. Right: APRL applied after RTRL.

3.1 Switching between algorithms

Our simulations and the results in [1] show that APRL not only has a lower computational complexity than RTRL but also reaches the minimum of the training error fast. RTRL in turn is less sensitive to transients and achieves better generalization errors. To combine the speed of APRL with the robustness of RTRL, we tried to use the weight matrices with the minimal training error obtained with APRL as initial weight matrices for further training with the RTRL algorithm. To generate some learning memory and to initialize the sensitivity values $\partial x_l / \partial w_{ij}$, which are needed to apply RTRL, we first run the network 200 steps without actually applying updates.

Surprisingly, we observe a jump of the error in all cases, as well for the Rössler task (cf. Fig. 4 left) as for the – otherwise very well behaved – Mackey-Glass system, even for very small learning rates of RTRL. We also observed cases where RTRL diverges after training initialized with the APRL-optimized configuration. Fig. 5 shows one of the runs, where RTRL achieves to come back to the desired Mackey-Glass trajectory, however, the performance is much worse than starting with a usual initialization. Fig. 5 gives an explanation of this fact in terms of the maximum norm of the weight matrices $\|W\|_m = \max_{ij} |w|_{ij}$. While RTRL keeps the weights small, APRL optimized weight matrices have large entries, which despite their actual good performance are a bad initialization for RTRL and can be pushed out of the working region by a few number of learning steps, which are caused by the small but always existing residual errors.

These results show that switching from APRL to RTRL is not well behaved and cannot easily be used to exploit the properties of both algorithms. We also carried out switching in the other direction from RTRL to APRL and the results show that the above observations apply vice versa (cf. Fig. 4 right).

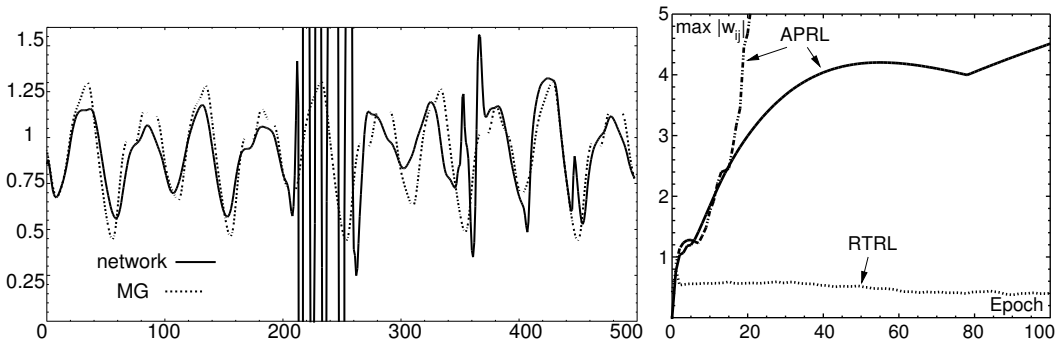

Fig. 5. Left: A trajectory for switching between algorithms while learning the Mackey-Glass system. After 200 iterations of the network initialized with an APRL-optimized weight matrix, RTRL starts and generates large errors quickly. Right: Matrix maximum norms for typical runs of 100 epochs of training 1019 steps of the Rössler attractor. The plot shows diverging (dash-dotted curve) and stable (solid curve) APRL runs, and an RTRL run (dotted curve).

That is, the minima obtained by RTRL are not minima of APRL, at least for the online algorithms. We believe that this sheds some light on the often ignored role played by the interaction between the time-variance of error surface, which is caused by the online learning algorithm, and the errors, which are derived from that very surface and drive the learning algorithm. Our results suggest that this interaction drives the weight dynamics of APRL and RTRL to follow qualitatively different paths, such that it is difficult to exploit both the speed of APRL and the robustness of RTRL.

4 The one-output behavior of APRL

In the following, we consider the case of only one output neuron, say x_1 (note that all simulations presented use only one output). 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

Proposition 1.

$$\forall k\ \forall i > 1\ \forall j > 1\ \forall h : \Delta w_{ih}(k) = \frac{w_{i1}(0)}{w_{j1}(0)} \Delta w_{jh}(k). \tag{13}$$

The proof for (13) is given in appendix B.

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

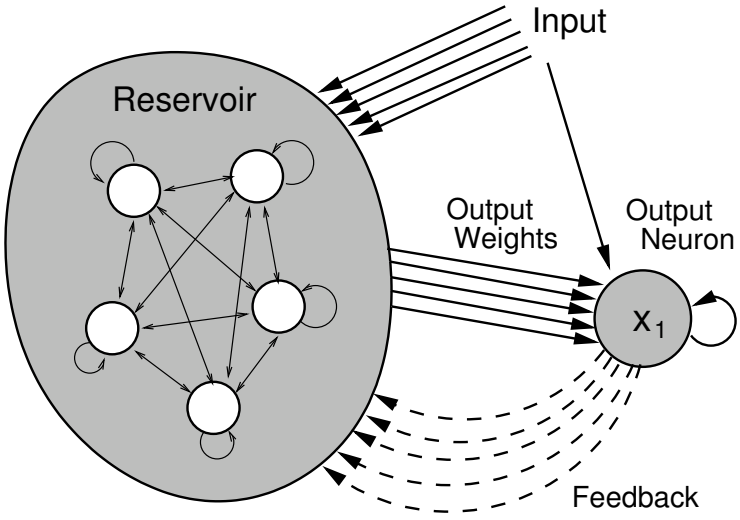

Fig. 6. Functional structure for one-output APRL: The network is partitioned into a dynamical reservoir, whose weights change slowly and are coupled, and an output layer that learns fast and uncoupled.

change and the majority of weights interconnecting the inner neurons, whose rates of change are systematically coupled and determined beforehand by the initialization. The result holds for the case of a single output neuron and does not easily generalize to more than one output.

To clarify this result further, let $s_{ij} = \frac{w_{i1}(0)}{w_{j1}(0)}$ denote the constant scaling factors. Then the update $\Delta \mathsf{W}$ of the weight matrix reads

$$\Delta \mathsf{W} = \left(\begin{array}{cccc} \Delta w_{11} & \Delta w_{12} & \cdots & \Delta w_{1n} \\ \hline \Delta w_{21} & \Delta w_{22} & \cdots & \Delta w_{2n} \\ s_{32}\Delta w_{21} & s_{32}\Delta w_{22} & \cdots & s_{32}\Delta w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n2}\Delta w_{21} & s_{n2}\Delta w_{22} & \cdots & s_{n2}\Delta w_{2n} \end{array} \right). \tag{14}$$

From equation (14), we obtain a functional division of the network structure. The inner weights form a reservoir whose dynamical behavior is used by a readout layer made up of the output weights. After T update steps we obtain

$$w_{ij}(T) = w_{ij}(0) + \Delta w_{ij}(T) = w_{ij}(0) + \frac{w_{i1}(0)}{w_{21}(0)} \sum_{k=1}^{T} \Delta w_{2j}(k), j > 2. \tag{15}$$

The functional division of the network structure is depicted in Figure 6. The partitioning into modules resembles seemingly very different approaches independently developed by Jäger [4,5] under the notion "echo state network"

and Natschläger et al. [6,7] as "liquid state machine". These approaches follow the idea to use a recurrent network as a dynamic reservoir that provides sufficient dynamics to store information about the time development of the inputs. The desired output function can then be learned efficiently by a linear readout layer. It can be shown that under special assumptions the echo state approach leads to a learning rule similar to the Atiya-Parlos learning rule [12]. However, in the general case there are two main differences: while the structure of echo state networks is an a-priori assumption and neurons are only locally connected, the functional division of APRL is a consequence of the error propagation based on the standard error function. Further, echo state networks do not have a feedback from the output layer and no online scaling of the reservoir takes place. Therefore the APRL one output case can be regarded a sort of intermediate model between the fully variable networks trained with RTRL and the specially structured echo state network.

4.1 Simulation results: Weight change

To illustrate the impact of the partitioning of the network into different functional groups of weights, we analyze the rates of change of the weights in the output layer and in the reservoir. We compute the average absolute change

$$\overline{\Delta w_{ij}} = \frac{1}{L} \sum_{k=1}^{L} |\Delta w_{ij}(k)| \tag{16}$$

of each weight, where L is the number of steps per epoch. Then the mean is taken for the complete weight matrix, for the output weights and for the inner weights, respectively, to match the functional modules.

$$\langle \Delta W \rangle_{\text{all weights}} = \frac{1}{n^2} \sum_i \sum_j \overline{\Delta w_{ij}}, \tag{17}$$

$$\langle \Delta W \rangle_{\text{output}} = \frac{1}{n} \sum_j \overline{\Delta w_{1j}}, \tag{18}$$

$$\langle \Delta W \rangle_{\text{reservoir}} = \frac{1}{n^2 - n} \sum_{i \neq 1} \sum_j \overline{\Delta w_{ij}}. \tag{19}$$

Figure 7 shows plots of these values for a trial for the Rössler system with 1250 learning steps per epoch for RTRL and APRL, where APRL behaves very regular and stable. As expected for APRL, the output weights change much more rapidly than the weights connecting the inner neurons. This behavior is also present for RTRL, yet much less pronounced. For the latter, the differences in the rates of change are a consequence of the vanishing gradient [9].

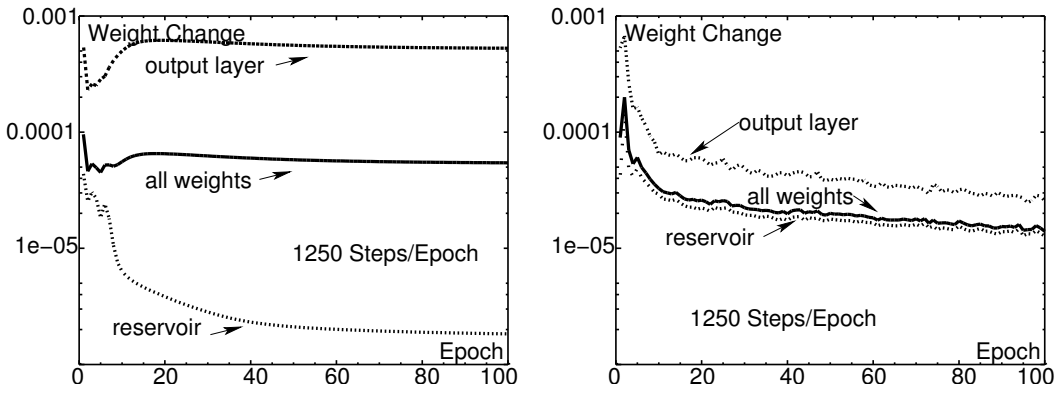

Fig. 7. Weight change of APRL (left) and RTRL (left). The curves depict the weight change of the output layer (dashed), the reservoir (dotted) and of all weights (solid). The difference between the reservoir and the output layer is explicit for APRL.

5 Summary and discussion

Our investigations on the weight dynamics of the continuous-time online-learning algorithm derived with the recent APRL approach [1,8,12] in comparison with RTRL show that the different strategies to minimize the error function lead to significantly different weight and error surface dynamics. The simulations confirm results from [1] which have also been obtained for different tasks and indicate that APRL can be used with higher learning rates such that the minimal training error is reached very quickly. Comparison with RTRL reveals that APRL is more sensitive to transients and has in some cases a characteristic overshoot of the error after reaching the minimum. This overshoot seems due to the initial transients and can not be avoided by decreasing the learning rate. This view is supported by long run simulations and a formal analysis, which shows that online APRL uses a momentum term, which is largest in the first learning steps and then gradually decays.

There are two striking results supporting the conjecture of unrelated weight dynamics of RTRL and APRL. The first is the lack of success in switching between the algorithms, which indicates that an optimal weight configuration for one of them is not useful for the other. Inspection of the corresponding weight matrices and computing norms show that APRL optimal configurations typically have much larger weights than RTRL matrices and therefore are unsuitable initial conditions for further RTRL training. The second interesting fact is provided by the analysis of the one-output case of APRL. In this case the relative rates of change in the non-output part of the weight matrix remain constant for every column. The result is a highly structured network which is partitioned into a fast output layer and a slower reservoir, whose inner degrees of freedom to adapt to the task are drastically reduced in comparison with RTRL. It seems that nevertheless at least in the case of

randomly initialized weight matrices there are always enough good working points for the reservoir to allow the fast changing output layer to implement a suitable readout function. However, the smaller number of degrees of freedom seems to require larger weights and consequently in some cases the algorithm does not seem to be able to fully control the process of scaling the reservoir such that at later stages the performance decreases and the typical overshoot can be observed.

The subset of all possible weight configurations accessible by APRL is determined beforehand by the initialization and consequently the network performance is sensitive to this initialization. We performed control experiments where the variability of the scaling factors was restricted systematically (only positive weights, particularly scaled weights, or large weights). In all such cases, where the degrees of freedom are additionally reduced by the initialization, the learning performance significantly decreases.

Further investigations should be made how far a similar behavior occurs in the multi-output case. Possible improvements could concern task-specific or otherwise optimized pre-shaped reservoirs and related further simplifications of APRL. We believe that these and further investigations can give insights into the ways recurrent learning works and can hereby lead to even more efficient algorithms.

Acknowledgments

We would like to thank three anonymous reviewers for detailed and valuable comments, which helped very much to improve the manuscript.

A Matrix notation of APRL

In this appendix section, we give the matrix representation of the Atiya-Parlos learning rule (following a similar notation as in [1]).

The following definitions will be used:

$$\mathsf{D}(k) = \operatorname{diag}\left(f'(x(k))\right), \tag{A.1}$$
$$\gamma(k) = -e(k) + \left[(1-\Delta t)I + \Delta t \mathsf{W}\mathsf{D}(k-1)\right] e(k-1), \tag{A.2}$$
$$\mathsf{B}(K) = \sum_{k=1}^{K} \gamma(k) f(x^T(k-1)) = \left(b_{ij}(K)\right), \tag{A.3}$$
$$\mathsf{V}(K) = \sum_{k=0}^{K-1} f(x(k)) f(x^T(k)) + \epsilon I. \tag{A.4}$$

where ϵ is a small constant which ensures invertibility of $\mathsf{V}(K)$. The learning rule (7) then can be written as

$$\Delta \mathsf{W}_{\text{batch}}(K) = \frac{\eta}{\Delta t} \mathsf{B}(K) \mathsf{V}^{-1}(K). \tag{A.5}$$

The matrices $\mathsf{B}(k)$ and $\mathsf{V}(k)$ can be obtained recursively as

$$\mathsf{B}(k) = \mathsf{B}(k-1) + \gamma(k) f(x^T(k-1)), \tag{A.6}$$
$$\mathsf{V}^{-1}(k) = \mathsf{V}^{-1}(k-1) - \frac{\mathsf{V}^{-1}(k-1) f(x(k-1)) [\mathsf{V}^{-1}(k-1) f(x(k-1))]^T}{1 + f(x^T(k-1)) \mathsf{V}^{-1}(k-1) f(x(k-1))} \tag{A.7}$$

where we use the small rank matrix inversion lemma in (A.7). The online update applies at each time step the increment

$$\begin{aligned}
\Delta \mathsf{W}(k) &= \Delta \mathsf{W}_{\text{batch}}(k) - \Delta \mathsf{W}_{\text{batch}}(k-1) \\
&= \frac{\eta}{\Delta t} \left[\mathsf{B}(k) \mathsf{V}^{-1}(k) - \mathsf{B}(k-1) \mathsf{V}^{-1}(k-1) \right] \\
&= \frac{\eta}{\Delta t} \left[\gamma(k) f(x^T(k-1)) + \mathsf{B}(k-1) \right] \mathsf{V}^{-1}(k) - \frac{\eta}{\Delta t} \mathsf{B}(k-1) \mathsf{V}^{-1}(k-1) \\
&= \frac{\eta}{\Delta t} \left[\gamma(k) f(x^T(k-1)) + \mathsf{B}(k-1) \mathsf{V}^{-1}(k-1) \mathsf{V}(k-1) \right] \mathsf{V}^{-1}(k) \\
&\qquad - \frac{\eta}{\Delta t} \mathsf{B}(k-1) \mathsf{V}^{-1}(k-1) \\
&= \frac{\eta}{\Delta t} \gamma(k) f(x^T(k-1)) \mathsf{V}^{-1}(k) + \Delta \mathsf{W}_{\text{batch}}(k-1) \left[\mathsf{V}(k-1) \mathsf{V}^{-1}(k) - I \right]
\end{aligned} \tag{A.8}$$

Observing that $\frac{\eta}{\Delta t} \gamma(k) f(x^T(k-1)) \mathsf{V}^{-1}(k) = \Delta \mathsf{W}_{\text{inst}}(k)$ is the least squares solution of (7) for $e(k') = 0, k' \neq k$ we obtain (9) discussed in Section 2.1.

The recursions in (A.6) and (A.7) lead to the continuous time online algorithm:

Algorithm 1. (1) Initialize $x(0)$ and $\mathsf{W}(0)$.

(2) $\mathbf{k}=1$:

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

(3) $\mathbf{k}=\mathbf{k}+\mathbf{1}$:

$$
\begin{aligned}
x(k) &= (1-\Delta t)x(k-1) + \Delta t \mathsf{W}(k-1) f(x(k-1)),\\
\gamma(k) &= -e(k) + [(1-\Delta t) I + \Delta t \mathsf{W}(k-1) \mathsf{D}(k-1)]\, e(k-1),\\
\Delta \mathsf{W}(k) &= \frac{\eta}{\Delta t} \frac{[\gamma(k) - \mathsf{B}(k-1)\mathsf{V}^{-1}(k-1) f(x(k-1))]\,[\mathsf{V}^{-1}(k-1) f(x(k-1))]^T}{1 + f(x^T(k-1)) \mathsf{V}^{-1}(k-1) f(x(k-1))},\\
\mathsf{W}(k) &= \mathsf{W}(k-1) + \Delta \mathsf{W}(k),\\
\text{update} \quad & \mathsf{B}(k), \mathsf{V}^{-1}(k-1) \quad \text{according to} \quad (A.6), (A.7),
\end{aligned}
$$

(4) Repeat step 3 for all data points.

By counting only multiplications, we have $n(1+3n_O)$ operations for the computation of $\gamma(k)$. For $\Delta \mathsf{W}(k)$ we need to compute $\mathsf{B}(k-1)\mathsf{V}^{-1}(k-1) f(x(k-1))$, $\mathsf{V}^{-1}(k-1) f(x(k-1))$ and the outer product in the numerator, yielding a total of $4n^2$ operations. The update of $\mathsf{B}(k)$ needs n^2 operations and $\mathsf{V}^{-1}(k)$ can be computed in $2n^2$ operations. Altogether we have $7n^2 + 3n_O n + n$ operations and the complexity of continuous-time online APRL is $\mathcal{O}(n^2)$.

B Proof of Proposition 1

In this section, we proof Proposition 1. For convenience, we repeat equation (13) and use $s_{ij} = \frac{w_{i1}(0)}{w_{j1}(0)}$ for the constant scaling factors.

$$
\forall k \; \forall i > 1 \; \forall j > 1 \; \forall h : \Delta w_{ih}(k) = \frac{w_{i1}(0)}{w_{j1}(0)} \Delta w_{jh}(k) = s_{ij} \Delta w_{jh}(k) \qquad \text{(B.1)}
$$

In the following Lemma similar formulas can be derived for $\gamma(k)$, $\mathsf{B}(k)$ and the first column of the weight matrix $\mathsf{W}(k)$.

Lemma 2. *Let $i > 1$, $j > 1$. Then $\forall k$*

$$\gamma_i(k) = \frac{w_{i1}(0)}{w_{j1}(0)}\gamma_j(k) = s_{ij}\gamma_j(k), \tag{B.2}$$

$$b_{ih}(k) = \frac{w_{i1}(0)}{w_{j1}(0)}b_{jh}(k) = s_{ij}b_{jh}(k), \tag{B.3}$$

$$w_{i1}(k) = \frac{w_{i1}(0)}{w_{j1}(0)}w_{j1}(k) = s_{ij}w_{j1}(k). \tag{B.4}$$

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

PROOF. Note that for one output neuron only the first component of the error vector $e(k)$ is non-zero, $\forall k\ , \forall i > 1 : e_i(k) = 0$. We show (B.1) and (B.2) to (B.4) by induction on the time step k.

k=1: Let $i > 1$, $j > 1$. Then

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

$$b_{ih}(1) = \gamma_i(1)f(x_h^T(0)) = 0 = \gamma_j(1)f(x_h^T(0)) = b_{jh}(1)$$

$$\Delta w_{ih}(1) = \frac{\eta}{\Delta t}\sum_l b_{il}(1)[\mathsf{V}^{-1}(1)]_{lh} = 0 = \frac{\eta}{\Delta t}\sum_l b_{jl}(1)[\mathsf{V}^{-1}(1)]_{lh} = \Delta w_{jh}(1)$$

Since these quantities are equal to zero, (B.2), (B.3) and (B.1) hold. (B.4) follows immediately:

$$\begin{aligned} w_{i1}(1) &= w_{i1}(0) + \Delta w_{i1}(1) = w_{i1}(0) \\ &= s_{ij}w_{j1}(0) = s_{ij}(w_{j1}(0) + \Delta w_{j1}(1)) = s_{ij}w_{j1}(1). \end{aligned}$$

k $\Rightarrow$ k+1: Let $i > 1$, $j > 1$ and suppose that (B.1) to (B.4) hold for all $k' \leq k$. From the formula for $\gamma(k+1)$ we get

$$\begin{aligned} \frac{\gamma_i(k+1)}{\gamma_j(k+1)} &= \frac{-e_i(k+1) + \sum_l \left[(1-\Delta t)I + \Delta t\mathsf{W}(k)\mathsf{D}(k)\right]_{il} e_l(k)}{-e_j(k+1) + \sum_l \left[(1-\Delta t)I + \Delta t\mathsf{W}(k)\mathsf{D}(k)\right]_{jl} e_l(k)} \\ &= \frac{\Delta t\ w_{i1}(k)f'(x_1(k))e_1(k)}{\Delta t\ w_{j1}(k)f'(x_1(k))e_1(k)} = \frac{w_{i1}(k)}{w_{j1}(k)} = \frac{w_{i1}(0)}{w_{j1}(0)} = s_{ij}. \end{aligned} \tag{B.5}$$

We have used that $e_i(k+1)$ and $e_j(k+1)$ are zero for $i, j > 1$. The sum contributes only the term for $l = 1$ and since $i \neq 1$ the identity in the square brackets vanishes. The last equality follows from the induction hypothesis.

The case $\gamma_j(k+1) = 0$ can occur only for $e_1(k) = 0$ as in (B.5) $f'(x_1) > 0$ and $w_{ij} \neq 0$, but then $\gamma_i(k+1) = 0 = \gamma_j(k+1)$ and (B.2) trivially holds.

For $B(k+1)$ we get

$$\begin{aligned}\frac{b_{ih}(k+1)}{b_{jh}(k+1)} &= \frac{b_{ih}(k)+\gamma_i(k+1)f(x_h^T(k))}{b_{jh}(k)+\gamma_j(k+1)f(x_h^T(k))} \\ &= \frac{s_{ij}\left(b_{jh}(k)+\gamma_j(k+1)f(x_h^T(k))\right)}{b_{jh}(k)+\gamma_j(k+1)f(x_h^T(k))} = s_{ij}.\end{aligned}$$

We have used (B.2) and the induction hypothesis to substitute $\gamma_i(k+1)$ and $b_{ih}(k)$. If $b_{jh}(k+1)=0$, then also $b_{ih}(k+1)=0$ and thus (B.3) holds. Now we turn to $\Delta\mathsf{W}(k+1)$. We cancel $\frac{\eta}{\Delta t}$ and the denominator and get

$$\begin{aligned}\frac{\Delta w_{ih}(k+1)}{\Delta w_{jh}(k+1)} &= \frac{\left(\gamma_i(k+1)-\sum_l b_{il}(k)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_l\right)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_h^T}{\left(\gamma_j(k+1)-\sum_l b_{jl}(k)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_l\right)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_h^T} \\ &= \frac{s_{ij}\left(\gamma_j(k+1)-\sum_l b_{jl}(k)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_l\right)}{\gamma_j(k+1)-\sum_l b_{jl}(k)\left[\mathsf{V}^{-1}(k)f(x(k))\right]_l} = s_{ij}.\end{aligned}$$

The terms $\left[\mathsf{V}^{-1}(k)f(x(k))\right]_h^T$ cancel. We used (B.2) and (B.3) to substitute $\gamma_i(k+1)$ and $b_{il}(k)$. If $\Delta w_{jh}(k+1)=0$, then also $\Delta w_{ih}(k+1)=0$ and hence (B.1) holds. Finally,

$$\begin{aligned}\frac{w_{i1}(k+1)}{w_{j1}(k+1)} &= \frac{w_{i1}(k)+\Delta w_{i1}(k+1)}{w_{j1}(k)+\Delta w_{j1}(k+1)} \\ &= \frac{s_{ij}\left(w_{j1}(k)+\Delta w_{j1}(k+1)\right)}{w_{j1}(k)+\Delta w_{j1}(k+1)} = s_{ij},\end{aligned}$$

where we have used (B.1) and the induction hypothesis to substitute $w_{i1}(k)$ and $\Delta w_{i1}(k+1)$. If $w_{j1}(k+1)=0$, then also $w_{i1}(k+1)=0$ and consequently (B.4) holds. Altogether it follows that Proposition 1 and Lemma 2 hold.

References

[1] A. F. Atiya, A. G. Parlos, New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence, IEEE Trans. Neural Networks 11 (3) (2000) 697–709.

[2] B. Hammer, J. J. Steil, Tutorial: Perspectives on Learning with RNNs, in: Proc. ESANN, 2002, pp. 357–368.

[3] B. A. Pearlmutter, Gradient Calculations for Dynamic Recurrent Neural Networks: A Survey, IEEE Trans. Neural Networks 6 (5) (1995) 1212–1228.

[4] H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks, Tech. Rep. 148, GMD (2001).

[5] H. Jaeger, Adaptive nonlinear system identification with echo state networks, in: S. T. S. Becker, K. Obermayer (Eds.), Advances in Neural Information Processing Systems 15, MIT Press, Cambridge, MA, 2003, pp. 593–600.

[6] W. Maass, T. Natschläger, H. Markram, Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations, Neural Computation 14 (11) (2002) 2531–2560.

[7] T. Natschläger, W. Maass, H. Markram, The "liquid computer": A Novel Strategy for Real-Time Computing on Time Series, TELEMATIK 8 (1) (2002) 39–43.

[8] U. D. Schiller, J. J. Steil, On the weight dynamics of recurrent learning, in: Proc. ESANN, 2003, pp. 73–78.

[9] S. Hochreiter, The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions, Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems 6 (2) (1998) 107–116.

[10] O. E. Rössler, An equation for continuous chaos, Phys. Letters 57A (5) (1976) 397–398.

[11] J. J. Steil, Input-Output Stability of Recurrent Neural Networks, Cuvillier Verlag, Göttingen, 1999, (also: Phd Thesis, Faculty of Technology, Bielefeld University, 1999).

[12] U. D. Schiller, Analysis and comparison of algorithms for training recurrent neural networks, Master's Thesis, Bielefeld University, , http://www.ulfschiller.de/publications/diploma.pdf (April 2003).