Backpropagation-Decorrelation: online recurrent learning with O(N) complexity

Jochen J. Steil

Neuroinformatics Group, Faculty of Technology
University of Bielefeld, Germany
jsteil@techfak.uni-bielefeld.de, www.jochen-steil.de

***Abstract*— We introduce a new learning rule for fully recurrent neural networks which we call *Backpropagation-Decorrelation* rule (BPDC). It combines important principles: one-step backpropagation of errors and the usage of temporal memory in the network dynamics by means of decorrelation of activations. The BPDC rule is derived and theoretically justified from regarding learning as a constraint optimization problem and applies uniformly in discrete and continuous time. It is very easy to implement, and has a minimal complexity of 2N multiplications per time-step in the single output case. Nevertheless we obtain fast tracking and excellent performance in some benchmark problems including the Mackey-Glass time-series.**

I. Introduction

In recent years, recurrent neural networks have become a fundamental tool for trajectory learning, in time-series prediction and generation, speech recognition, adaptive control, or biological modeling. Despite encouraging practical success, one of the main drawbacks for their application is the known high complexity of training algorithms. Thus reduction of training complexity has always been one of the main issues in recurrent learning research (see for a review [1]). In the field of gradient based algorithms, some milestones were the reduction of the $O(N^4)$ real-time recurrent learning [2] to $O(N^3)$ in [3], and the introduction of backpropagation through time (BPTT) in its online version [4], which has $O(N^2)$ but is storage demanding. There is ongoing research to devise other efficient recurrent learning schemes, in particular employing regularization techniques and partially recurrent networks (see the recent review in [5]). But most of the efficient existing algorithms are quite complex and in particular the online techniques typically need proper adjustment of learning rates and time-constants. A technique as simple and easy to use as standard backpropagation for feedforward networks, which could attract a wider audience to the usage of recurrent networks, is still lacking.

Recently two fruitful new ideas have appeared. In [6], Atiya and Parlos have derived a new $O(N^2)$-efficient algorithm, which is based on the idea to differentiate the error function with respect to the states in order to obtain a "virtual teacher" target, with respect to which the weight changes are computed. It has been claimed that this new technique outperforms the BPTT approach significantly [6]. We refer to this approach, which can be applied in continuous and discrete time and for different formulations of the recurrent dynamics, as APRL (Atiya-Parlos recurrent learning). A second, seemingly very different source of ideas has been developed in [7] under the notion "echo state network" and [8] as "liquid state machine". Both approaches use recurrent networks as a kind of dynamic reservoir, which stores information about the temporal behavior of the inputs and allows to learn a linear readout function. There is, however, an interesting connection between these ideas: for the – most common and most important – case of a single output neuron it was shown in [9] that APRL also leads to a functional decomposition of the trained networks into a fast adapting readout layer and a slowly changing dynamic reservoir. In the reservoir weight changes are highly coupled, such that the network develops with much less independent degrees of freedom than usual.

In this contribution, we devise a new and very simple learning rule which combines these two aforementioned ideas with the attempt to optimize information processing by means of decorrelation. While decorrelation learning rules are well known for sparse coding and blind source separation [10], [11] and have also been proposed in biological modeling [12], the combination of decorrelation with backpropagtion of teacher induced errors is not common for recurrent trajectory learning. Our new learning rule uses three important principles: (i) one-step back propagation of errors by means of the virtual teacher forcing like in APRL, (ii) the usage of the temporal memory in the network dynamics which is adapted based on decorrelation of the activations, and (iii) the employment of a non-adaptive reservoir of inner neurons to reduce complexity. The output weights then implement an linear readout function while, however, still give full feedback into the reservoir. We therefore call the new learning scheme *backpropagation-decorrelation* rule (BPDC).

The third principle implies that the BPDC-rule is applied only to the output weights, such that we obtain a recurrent online-learning scheme with *linear complexity* $2N$. It applies uniformly in discrete and continuous time, is simple to implement, and proves to be very robust against parameter variations in simulations. The error measures obtained are excellent and are comparable to the online version of APRL as is shown by simulations of some benchmark problems including the Mackey-Glass time-series.

In Section 2, we introduce the learning rule, in Section 3 we show that it can be developed and theoretically justified

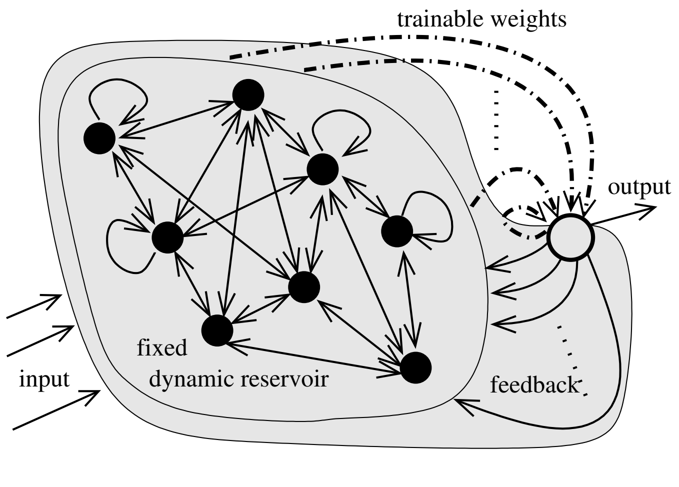

Fig. 1. The BackPropagation-DeCorrelation (BPDC) rule adapts only the output weights of a fully connected recurrent network with non-adaptive internal reservoir. It aims at temporal decorrelation of the network activations with respect to the inputs and one-step backpropagated errors, such that the output neuron can optimally read out from the dynamical memory.

starting from the virtual teacher forcing approach of APRL learning, which it approximates in some loose sense. In Section 4, we give simulation results for some benchmark problems and discuss the results in the final Section 5.

II. Backpropagation-Decorrelation Learning

We consider fully connected recurrent networks

$$x(k+1) = (1-\Delta t)x(k) + \Delta t W f(x(k)), \qquad (1)$$

where $x_i, i=1,\ldots,N$ are the states, $W \in \mathbb{R}^{N\times N}$ is the weight matrix, and $k = \hat{k}\Delta t, \hat{k} \in N_+$ is a discretized time variable such that for small Δt we obtain an approximation of the continuous time dynamics $dx/dt = -x + Wf(x)$ and for $\Delta t = 1$ the standard discrete dynamics. We assume that f is a standard sigmoid differentiable function and is applied component wise to the vector x (or x^T). For simplicity we treat inputs $u_s(k)$, $s = 1,...,N_i$ by lumping them to certain states x_r such that $x_r(k) = u_s(k)$ for all times k and we assume that the inputs are scaled to a reasonable region, such that the mean is approximately zero. We further assume that W is initialized with small random values. Denote by $O \subset \{1,..,N\}$ the set of indices j of N_o output neurons i.e. x_j output $\Rightarrow j \in O$ and let for a single output neuron w.r. $O=\{1\}$ such that x_1 is the respective output as shown in Fig. 1.

How can the architecture shown in Fig. 1 be trained ? Assuming that all but the output weights are fixed, we can regard the inner neurons as dynamical reservoir, which is triggered by the input signal and provides a dynamical memory. Its information processing capacity is maximal, if the states are maximally decorrelated with respect to the given input. Then the output layer can optimally combine these states to read out the desired output. We will show below that a compromise between error propagation and decorrelation is implemented by the *Backpropagation-Decorrelation* rule

$$\Delta w_{ij}(k+1) = \frac{\eta}{\Delta t}\frac{f(x_j(k))}{\sum_s f(x_s(k))^2 + \epsilon}\gamma_i(k+1) \qquad (2)$$

where $\gamma_i(k+1) =$

$$\sum_{s\in 0}\Big((1-\Delta t)\delta_{is} + \Delta t w_{is} f'(x_s(k))\Big)e_s(k) - e_i(k+1). \qquad (3)$$

Here η is the learning rate, ϵ a regularization constant ($\epsilon = 0.002$ throughout), and $e_s(k)$ are the non-zero error components for $s \in O$ at time $k : e_s(k) = x_s(k) - y_s(k)$ with respect to the teaching signal $y_s(k)$. We show below that the term $f(x_j(k))/(\sum_s f(x_s(k)^2 + \epsilon)$ implements an approximative decorrelation rule. The γ_i propagate a mixture of the local errors $e_i(k+1)$, $e_i(k)$ and the errors in the last time step $e_s(k)$ weighted by a typical backpropagation term involving f'. Note that a restriction to adapt only the output weights introduces the kind of inner dynamic reservoir discussed above.

The complexity of this rule (counting only multiplications) is of order $N^2 + 2N + NN_o$: N for the factor in the denominator and N for multiplication of all $f(x_j)$ with that factor. Computation of all γ_i need NN_o multiplications, finally there remain N^2 for multiplying each $f_j\gamma_i$, $i,j \in \{1,...N\}$.

The discussion below and the simulation results will show that it is in most cases reasonable to restrict adaptation to the output weights. Then for the most common case of a single output neuron we obtain

$$\Delta w_{1j}(k+1) = \frac{\eta}{\Delta t}\frac{f(x_j(k))}{||f(x(k))||^2 + \epsilon} \times\Big[\big[(1-\Delta t) + \Delta t\, w_{11} f'(x_1(k))\big]e_1(k) - e_1(k+1)\Big] \qquad (4)$$

This update can be computed with complexity $2N$ only. The method as presented here is a fast online algorithm and needs to store only the N activations of the last time step. Its derivation in the next section shows that it coarsely approximates batch and online algorithms introduced in [6], which in turn can also be interpreted as backpropagation-decorrelation rules.

III. Derivation of the learning rule

A. Recurrent learning as constraint optimization

Let x_1 be the output neuron, then we have to solve the constraint optimization problem

$$\text{minimize} \quad E \quad \text{with respect to} \quad g \equiv 0. \qquad (5)$$

where the error function E with respect to the target output y for K time-steps is given by

$$\mathsf{E} = \frac{1}{2}\sum_{\hat{k}=1}^{K}\sum_{s\in O}\big[x_s(\hat{k}\Delta t) - y_s(\hat{k}\Delta t)\big]^2 \qquad (6)$$

and the constraint equation obtained from the original recurrent network dynamics (1) is

$$\mathsf{g}(k+1) \equiv -x(k+1) + (1-\Delta t)x(k) + \Delta t W f(x(k)) = 0. \qquad (7)$$

It has been shown in [6] that many of the common training algorithms for recurrent networks (including RTRL and BPTT) can be derived from this starting point. Additionally a number of approaches using classical quadratic optimization methods to solve (6), (7) have been introduced, for instance conjugate gradients [13] and Newton methods, mostly under the term second order learning (see [14] and the references therein).

B. Virtual Teacher Forcing

To minimize (5), we follow a new approach introduced in [6]. The idea is to use the constraint equation to compute weight changes to approach a virtual target state, which is obtained by differentiating the error E with respect to the state (instead of the weights as in the usual gradient methods like RTRL or BTTP). To get a compact notation, we collect the relevant quantities in vectors (w_i^T are the rows of W)

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

We obtain a targets

$$\Delta\mathbf{x} = -\left(\frac{\partial \mathsf{E}}{\partial \mathbf{x}}\right)^T = -(e^T(1), \dots, e^T(K))^T,$$
$$\text{where} \quad e_s(k) = \begin{cases} x_s(k) - y_s(k), & s \in O, \\ 0, & s \neq O \end{cases}$$

and compute weight updates $\Delta\mathbf{w}$ to drive the network towards $\eta\Delta\mathbf{x}$ by using the constraint (7):

$$\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\Delta\mathbf{w} \approx -\eta \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Delta\mathbf{x}. \tag{8}$$

We call this approach *virtual teacher forcing* because the targeted teacher states $\mathbf{x} - \Delta\mathbf{x}$ are never actually fed into the network but nevertheless enforce the weight changes. We refer to the approach to solve the equation (8) using a pseudo-inverse for $\partial\mathbf{g}/\partial\mathbf{w}$ as Atiya-Parlos recurrent learning (APRL). It yields the training rule

$$\Delta\mathbf{w}_{batch} = -\eta \left[\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)\right]^{-1} \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Delta\mathbf{x}. \tag{9}$$

It is worth noting that this update direction $\Delta\mathbf{w}$ does not follow the conventional gradient direction as for real time recurrent learning [6], [15] and therefore leads to different weight dynamics, which show a larger sensitivity to transient behavior for online-learning [15]. The batch update (9) can be computed for one output and ($K \gg N$) with $3N^2 + 3N$ multiplications per time-step [6]. It is straightforward to derive the respective online algorithm [6], [9], [15] by recursively computing the pseudo-inverse in (9).

C. Atiya-Parlos recurrent learning revisited

We now directly compute the terms in (9) to further interpret the APRL method. Denote the vector of activations at time-step k by $\mathsf{f}_k = (f(x_1(k)), \dots, f(x_N(k)))^T$, then

$$\frac{\partial \mathbf{g}(k)}{\partial \mathbf{w}} = \begin{pmatrix} [\ \mathsf{f}_k\ ]^T & & \dots \\ \dots & [\ \mathsf{f}_k\ ]^T & \dots \\ & & \ddots & \\ \dots & \dots & & [\ \mathsf{f}_k\ ]^T \end{pmatrix} \in \mathbb{R}^{N \times N^2}.$$

Further

$$\frac{\partial \mathbf{g}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathbf{g}(1)}{\partial \mathbf{w}} \\ \vdots \\ \frac{\partial \mathbf{g}(K)}{\partial \mathbf{w}} \end{bmatrix}, \quad \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T = \begin{bmatrix} \frac{\partial \mathbf{g}(1)}{\partial \mathbf{w}}^T & \dots & \frac{\partial \mathbf{g}(K)}{\partial \mathbf{w}}^T \end{bmatrix}$$

and it is easy to see that

$$\left[\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)\right]^{-1} = \mathrm{diag}\left\{\left(\sum_{k=0}^{K-1} \mathsf{f}_k \mathsf{f}_k^T\right)^{-1}\right\} = \mathrm{diag}\left\{C_{K-1}^{-1}\right\}$$

where C_{K-1} is the auto-correlation matrix of the network activities. On the other hand we have $(\partial\mathbf{g}/\partial\mathbf{x})\,\Delta\mathbf{x} = \boldsymbol{\gamma}$, where

$$\boldsymbol{\gamma} = (\gamma(1)^T, \dots, \gamma(K)^T)^T, \quad \gamma(k) = (\gamma_1(k), \dots, \gamma_N(k))^T$$

is given in (3). Then

$$\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^T \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Delta\mathbf{x} = \begin{bmatrix} \sum_{k=0}^{K-1} \mathsf{f}_k \gamma_1(k+1) \\ \vdots \\ \sum_{k=0}^{K-1} \mathsf{f}_k \gamma_N(k+1) \end{bmatrix} \in \mathbb{R}^{N^2 \times 1}$$

$$\Rightarrow \quad \Delta\mathbf{w}_{batch} = \begin{bmatrix} \mathsf{C}_K^{-1} \sum_{k=0}^{K-1} \mathsf{f}_k \gamma_1(k+1) \\ \vdots \\ \mathsf{C}_K^{-1} \sum_{k=0}^{K-1} \mathsf{f}_k \gamma_N(k+1) \end{bmatrix} \in \mathbb{R}^{N^2 \times 1}$$

By splitting the batch update into the update up to step $(k-1)$ and the increment for $(k-1) \to k$ we finally obtain the online rule

$$\begin{aligned}\Delta w_{ij}(k+1) = &\frac{\eta}{\Delta t}\left[C_k^{-1}\mathsf{f}_k\right]_j \gamma_i(k+1) \\ &+ \frac{\eta}{\Delta t}(C_k^{-1} - C_{k-1}^{-1}) \sum_{r=0}^{k-1} [\mathsf{f}_r]_j \gamma_i(r+1) \end{aligned} \tag{10}$$
$$\text{where} \quad C_k^{ij} = \epsilon\,\delta_{ij} + \sum_{r=0}^{k-1} f(x_i(r)) f(x_j(r)).$$

If we assume that all activations $f(x_i)$ are approximately centered at zero, for instance that the activation function $f = \tanh$[1], then C_k is the correlation matrix of activations regularized by adding ϵI. Thus the term $[C_k^{-1}\mathsf{f}_k]_j$ decorrelates the vector of activations which then is multiplied by the one-step error backpropagation of the virtual teacher forcing rule. The second term in (10) collects all previous errors, which are multiplied by the increment of the decorrelation matrix, such that over time these terms exactly sum up to

[1]Note that any system with a common sigmoid activation function can be represented by a dynamically equivalent system with $\tanh$ as activation and appropriately scaled inputs [16].

TABLE I

COMPLEXITY OF BPDC ALGORITHMS COUNTING MULTIPLICATIONS FOR ONE OUTPUT NEURON

algorithm	update all weights	output weights only
APRL batch	$3N^2+2N$	$2N^2+3N$
APRL online	$7N^2+4N$	$6N^2+5N$
BPDC online	N^2+3N	2N

the full decorrelation matrix. Here the term *backpropagation-decorrelation* learning originates. We find in the experiments that indeed APRL as well as the new BPDC rule tend to center activations at zero (except the output neuron) and to decorrelate them. The update complexity depends on the recursive computation of $C_k \to C_{k+1}$ and adds to a total of $7N^2+4N$ [17]. We refer to this algorithm as APRL-online.

We can rewrite (10) also as

$$\begin{aligned}\Delta w_{ij}(k{+}1) =& \frac{\eta}{\Delta t}\left[C_k^{-1}\mathsf{f}_k\right]_j \gamma_i(k+1)\\ &+\frac{\eta}{\Delta t}(C_k^{-1}C_{k-1}C_{k-1}^{-1}-C_{k-1}^{-1})\sum_{r=0}^{k-1}\left[\mathsf{f}_r\right]_j\gamma_i(r+1)\\ =& \frac{\eta}{\Delta t}\left[C_k^{-1}\mathsf{f}_k\right]_j\gamma_i(k+1)+\frac{\eta}{\Delta t}\left(C_k^{-1}C_{k-1}-I\right)\Delta w_{ij}^{batch}(k),\end{aligned}$$

which shows that APRL in fact employs a mixture of the instantaneous error and a momentum term, which decays to zero. Thus, though not following the gradient, APRL suffers from fading memory like gradient algorithms from the vanishing gradient (cf. [18]) and cannot preserve information for long times like for instance Long-Short term memory networks [19]. For more discussion of the APRL strategy see [15].

D. Backpropagation-decorrelation learning

The considerations above motivate an approximation which does not try to accumulate a full correlation matrix and consequently also skips the accumulation of previous errors. We rather use only the instantaneous correlation at time step k: $C(k)=\epsilon I+\mathsf{f}_k\,\mathsf{f}_k^T$ to get

$$\Delta w_{ij}(k+1)=\eta[C(k)^{-1}\mathsf{f}_k]_j\gamma_i(k+1).$$

We compute $C(k)^{-1}\mathsf{f}_k$ using the small rank adjustment matrix inversion lemma

$$\begin{aligned}C(k)^{-1}\mathsf{f}_k &= \left[\frac{1}{\epsilon}I-\frac{\frac{1}{\epsilon}[I\mathsf{f}_k][\frac{1}{\epsilon}I\mathsf{f}_k]^T}{1+\mathsf{f}_k^T\frac{1}{\epsilon}I\mathsf{f}_k}\right]\mathsf{f}_k\\ &=\mathsf{f}_k\left[\frac{1}{\epsilon}-\frac{1}{\epsilon^2}\frac{\mathsf{f}_k^T\mathsf{f}_k}{1+\frac{1}{\epsilon}||\mathsf{f}_k||^2}\right]=\mathsf{f}_k\frac{1}{\epsilon+||\mathsf{f}_k||^2}.\end{aligned}$$

The denominator can be computed with N multiplications and therefore the vector $C(k)^{-1}\mathsf{f}_k$ with $2N$. We obtain

$$\begin{aligned}\Delta w_{ij}(k{+}1) &= \frac{\eta}{\Delta t}\frac{[\mathsf{f}_k]_j}{||\mathsf{f}_k||^2+\epsilon}\\ &\times\begin{cases}\left[(1{-}\Delta t)+\Delta t\,w_{11}f'(x_1(k))\right]e_1(k){-}e_1(k{+}1),\\ \left[(1{-}\Delta t)+\Delta t\,w_{i1}\,f'(x_1(k))\right]e_1(k), \qquad i>1.\end{cases}\end{aligned} \tag{11}$$

This update needs just N^2+3N multiplications per data point k and is BPDC learning applied to all weights.

E. Adaptation of the output weights only

The final step is to apply (11) only to the output weights to obtain (4). It is motivated by the fact that these change at much higher rates than the internal weights because the instantaneous error $e_1(k+1)$ is backpropagated only to the output weights. Further motivation provides that in the one neuron case for APRL the rates of change of the internal weights are coupled by constant scaling factors which are fully determined by the initialization [9]. We therefore consider only $i=1$ in (11) and obtain the BPDC rule (4) introduced above. Table I summarizes the complexity of the different algorithms introduced. It turns out in the simulation results that fixing the reservoir sometime even gains better approximation results.

IV. SIMULATIONS AND RESULTS

The new learning rule has been tested on a number of problems in discrete as well as in continuous time and some of the results are given below. In all cases we use the normalized mean square error NMSE defined as $E[(x_1(k)-y(k))^2]/\sigma^2$, where σ^2 is the variance of the teacher signal. For comparison with APRL we use the same network parameters $\epsilon=0.002$ and $\eta=0.2$ as in [6]. All results are averages over twenty runs. We omit further comparisons to other online-schemes and refer to [6] for a comparison of APRL with BPTT.

Example 1 (Roessler dynamics): Consider the task to implement a mapping between the coordinate functions $z_1(t), z_2(t), z_3(t)$ of the chaotic Roessler dynamics

$$\dot{z}_1=-z_2-z_3,\quad \dot{z}_2=z_1{+}0.2z_2,\quad \dot{z}_3=0.2{+}z_1z_3-5.7z_3,$$

where $z_1(t), z_3(t)$ are inputs and $z_2(t)$ reference output. This is a mildly complex trajectory learning task in continuous time with no closed form for its solution [20]. Training proceeds on 1200 time steps with $\Delta t=0.1$ integrated by a 4-th order Runge-Kutta scheme starting from initial conditions $[0.495, -0.116., -0.3]$. The first 200 steps are disregarded for the error. Generalization is measured for the next 1000 steps, starting at the end of training, such that there occur no transients.

Example 2 (second order system): The following second order example is taken from [6], it is given in discrete time as predicting the next output for

$$y(k+1)=\frac{y(k)y(k-1)(y(k)+0.25)}{1-y(k)^2+y(k-1)^2}+u(k).$$

The network receives input $u(k)$ and the teacher signal $y(k{+}1)$.

Example 3 (Mackey-Glass): As higher order benchmark we use the well known Mackey-Glass system with standard parameters

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

Inputs are $y(k), y(k-6), y(k-12), y(k-18)$ and the target $y(k+84)$, where we integrate from $k\to k{+}1$ using 30 Runge-Kutta 4-th order steps.

TABLE II

AVERAGE TEST/TRAINING NMSE FOR 20 RUNS OF 10 EPOCHS, THE NUMBER OF NEURONS, AND THE TIME STEP ARE GIVEN IN THE FIRST LINE. ALL WEIGHTS ARE INITIALISED RANDOMLY IN $[-0.2, 0.2]$.

algorithm	Roessler/20/.1	2nd Order/30/1	Mackey-G./40/.2	10-th Order/40/1
APRL all	.092/.068	.017/.017	.101/.175	.340/.362
APRL out	.354/.440	.062/.062	.101/.182	.194/.194
BPDC all	.123/0.467	.058/.099	.094/.318	.152/.681
BPDC out	.006/.094	.050/.085	.034/.408	.142/.540

Example 4 (Tenth-order system) The following very hard problem in discrete time has also been considered in [6]

$$y(k+1) = 0.3y(k) + 0.05y(k)\left[\sum_{i=0}^{9} y(k-i)\right] + 1.5u(k-9)u(k) + 0.1.$$

Here we supply $u(k), u(k-9)$ as input to the network to predict the next output $y(k+1)$.

In examples 2 and 4 we supply 500 points for relaxation of the system, followed by 500 points for training, and further 500 for generalization. For the Mackey-Glass system it turns out that 500 points for training are already too many, such that overfitting occurs. We therefore reduce the training sequence to 200 points. The comparative results in Table 2 show that it is better to adapt only the output weights sometimes even for APRL, for which we were able to roughly reproduce the results from [6]. The proposed BPDC rule shows very good performance at extremely low computational costs and generalizes very well for the simpler tasks and reasonable for Mackey-Glass. The results for the tenth-order system are less satisfactory and we suspect that the rigid cut off of the error backpropagation is too strict in this case.

In Table 3 training/test NMSE results of BPDC for the Mackey-Glass system are shown together with the best performance for 200/500 points for training and 500 test points are given. All results are avarages over 20 run. In lines 3/4 the learning rate is *increased* every epoch, lines 5/6 refer to a single "one shot" training. Network size, number of epoches, learning rates, and increments are shown as (20/5/0.2/0.25) in the headlines. For results marked with $()^{\star}$ maximal 2 outliers were disregarded. The results show typical overfitting: for decreasing training error the test error increases, regardless to whether we supply more training data or use more neurons. In Fig. 2 a further interesting feature of BPDC is highlited: it is able to adapt extremely fast to the main characteristics of the task in a sort of one-shot online-learning. The results shown were obtained with a single epoch of training and a relatively small number of 500 training points.

In general we encountered no stability problems, neither of the learning process, nor did divergence of the resulting network occur. It is worth noting that the learning rate has to increase with the size of the network and for performance gain. Though not reflected in the averages in Tables 2 and 3, we sometimes observed large fluctuations in the results, and for all parameter choices there were networks with excellent performance (with respect to Table 3, line 1, a best performance of 0.024/.161 was reached at least once for all configurations).

As the initialization fixes with the reservoir a large portion of the system it can be expected to be crucial for the success. However, the very small variance obtained for the results in Table 2 ($< 10^{-3}$ in all cases) disprove this claim and the performance is not very sensitive to the initialization interval. The strong robustness of the results against initialization changes or different learning rates can be explained from the self-stablelizing effect of the resulting increase/decrement of the denominator in the decorrelation factor.

V. CONCLUSION

We have introduced a new effective and simple learning rule for online adaptation of recurrent networks, which combines backpropagation, virtual teacher forcing, and decorrelation. Applied only to the output weights, it yields a minimal $O(N)$ complexity at very good performance. The resulting network structure resembles the "echo state networks" [7], which also use a dynamical reservoir and optimize a linear readout function, but need a prescaling of the weight matrix to obtain a suitable spectral radius and rely on a special local random connectivity. In some control experiments, we found no dependence of the performance of our method on the spectral radius of the randomly initialized weight matrices. The online version of echo state further uses recursive least squares algorithms, which themselves introduce new parameters and

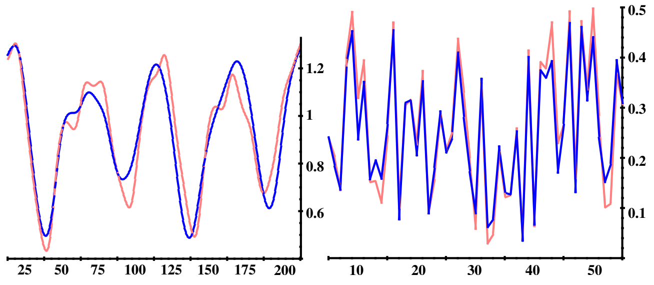

Fig. 2. 200 points of generalization for the Mackey-Glass and 60 points of generalization for the 2cnd-oder system both after a one-shot training with 500 points, the network is shown dark.

TABLE III

AVERAGED TRAINING/TEST NMSE OF BPDC FOR THE MACKEY-GLASS AND BEST PERFORMANCE FOR 200/500 POINTS TRAINING AND 500 TEST POINTS. SEE TEXT FOR MORE DETAILS.

line	data	20/5/.2/-	50/6/.2/-	75/6/.5/-	100/10/.5/-	150/10/.5/-
	200	.075/.180	.044/.376	.038/.220	.028/.221	.032/.432
2	500	.085/.357	.089/.352	.032/.265	.024/.212	.027/.397
		20/5/.2/+.25	50/6/.2/+.2	75/6/+.2	100/10/+.1	150/10/+.1
3	200	.013/1.84	.016/1.32	.016/.826	.009/.581*	.008/.477*
4	500	.017/–	.011/–	.006/.732	.005/.520*	.007/.384*
		20/1 .2/-	50/1/.2/-	75/1/.5/-	100/1/.5/-	150/1/.5/-
5	500	.033/.382	0.047/.598	.071/1.231	.069/.826	.071/.725
6	1000	.082/.366	0.079/1.100	.067/1.372	.061/.526	.070/.687

computational complexity though the results reported in [7] yield a better generalization.

In comparison, our approach yields very good performance at maximal simplicity and minimal complexity. It can be well interpreted and be derived from the standard error function and a constraint optimization approach. It relies on a rigid error cutoff and therefore may have more difficulties with long term dependencies. However, the extremely fast adaptation allows to use and train large networks up to several hundred neurons. The encouraging results, which indicate that a fast online one-shot learning is possible, make the networks feasible in particular for identification and adaptive control tasks.

Next steps in the investigation of this algorithm are a theoretical analysis of the network stability with the methods proposed in [20], [21]. Of further interest also is a characterization of the generalization ability with respect to properties of the reservoir or the information transmission rate of the reservoir. We believe that the decorrelation of network states is optimal for generalization, however, a theoretic and quantitative account could provide further insights, also with respect to an optimized initialization. Though these and other issues certainly need to be investigated, we believe that already in the present form the BPDC learning rule, – especially because of its efficiency, robustness, and simplicity – is a step towards easier and more widespread application of recurrent learning.

ACKNOWLEDGMENT

Special thanks to H. Ritter and U. Schiller for very valuable discussions and to U. Schiller for the APRL online-learning implementation according to [9], [17].

REFERENCES

[1] R. J. Williams and D. Zipser, “Gradient-based learning algorithms for recurrent networks and their computational complexity,” in *Back-propagation: Theory, Architectures, and Applications*, Y. Chauvin and D. E. Rumelhart, Eds. Lawrence Erlbaum Publ., 1995, pp. 433–486.

[2] B. A. Pearlmutter, “Gradient calculations for dynamic recurrent neural networks: A survey,” *IEEE Tansactions on Neural Networks*, vol. 6, no. 5, pp. 1212–1228, 1995.

[3] J. Schmidhuber, “A fixed size storage $O(N^3)$ time complexity learning algorithm for fully recurrent continually running networks,” *Neural Computation*, vol. 4, no. 2, pp. 243–248, 1992.

[4] R. J. Williams and J. Peng, “An efficient gradient–based algorithm for on–line training of recurrent network trajectories,” *Neural Computation*, vol. 2, no. 4, pp. 490–501, 1990.

[5] B. Hammer and J. J. Steil, “Tutorial: Perspectives on learning with recurrent neural networks,” in *Proc. of ESANN*, 2002, pp. 357–368.

[6] A. B. Atiya and A. G. Parlos, “New results on recurrent network training: Unifying the algorithms and accelerating convergence,” *IEEE Trans. Neural Networks*, vol. 11, no. 9, pp. 697–709, 2000.

[7] H. Jaeger, “Adaptive nonlinear system identification with echo state networks,” in *NIPS*, 2002.

[8] T. Natschläger, W. Maass, and H. Markram, “The ”liquid computer”: A novel strategy for real-time computing on time series,” *TELEMATIK*, vol. 8, no. 1, pp. 39–43, 2002.

[9] U. D. Schiller and J. J. Steil, “On the weight dynamcis of recurrent learning,” in *Proc. ESANN*, 2003, pp. 73–78.

[10] S. Choi, S. Amari, and A. Cichocki, “Natural gradient learning for spatio-temporal decorrelation: Recurrent network,” *IEICE Trans. Fundamentals*, vol. E83-A, no. 12, 2000.

[11] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. Wiley, 2001.

[12] K. P. Körding and P. König, “A learning rule for dynamic recruitment and decorrelation,” *Neural Networks*, vol. 13, pp. 1–9, 2000.

[13] W. F. Chang and M. W. Mak, “A conjugate gradient learning algorithm for recurrent neural networks,” *Neurocomputing*, vol. 24, no. 1-3, pp. 173–189, 1999.

[14] E. P. dos Santos and F. J. V. Zuben, *Recurrent Neural Networks: Design and Applications*. CRC Press, 1999, ch. Efficient Second-Order Learning Algorithms for Discrete-Time Recurrent Neural Networks.

[15] U. D. Schiller and J. J. Steil, “Analyzing the weight dynamics of recurrent learning algorithms,” *Neurocomputing*, 2004, in press.

[16] P. Tiňo, B. G. Horne, and C. L. Giles, “Attractive periodic sets in descrete-time recurrent networks (with emphasis on fixed-point stability and bifurcations in two-neuron networks),” *Neural Computation*, vol. 13, pp. 1379–1414, 2001.

[17] U. D. Schiller, “Analysis and comparison of algorithms for training recurrent neural networks,” Master’s thesis, Faculty of Technology, University of Bielefeld, 2003. [Online]. Available: http://www.ulfschiller.de/publications/diploma.pdf

[18] S. Hochreiter, “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions,” *Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[19] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[20] 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*. IEE, 1999, pp. 761–766.

[21] J. J. Steil, “Local structural stability of recurrent networks with time-varying weights,” *Neurocomputing*, vol. 48, no. 1-4, pp. 39–51, 2002.