ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

Attractor-based computation with reservoirs for online learning of inverse kinematics

R. Felix Reinhart[1] and Jochen J. Steil[2]

Research Institute for Cognition and Robotics (CoR-Lab), Bielefeld University
1-freinhar@CoR-Lab.Uni-Bielefeld.de, 2-jsteil@CoR-Lab.Uni-Bielefeld.de

Abstract. We implement completely data driven and efficient online learning from temporally correlated data in a reservoir network setup. We show that attractor states rather than transients are used for computation when learning inverse kinematics for the redundant robot arm PA-10. Our findings shade also light on the role of output feedback.

1 Introduction

In robotics, learning is typically constrained by limited resources such as memory and processing time. Moreover, data is only available sequentially and thus learning has to cope with temporally correlated data. Standard feedforward network architectures fail at online learning from temporally correlated data because of destructive interference. We show that the reservoir computing idea of a fixed encoding device can address both issues by implementing completely data driven and efficient online learning for sequence transduction. The reservoir state encodes the input in a nonlinear and high-dimensional representation. But it is an open question whether transients or attractor states are used for computation. We investigate these questions in the framework of an inverse kinematics learning task. Important is that the presented sequence transduction task is temporal by means of ordered data presentation, although the mapping is instantaneously static. It turns out that in this setting attractor-based computation outperforms computation with transients.

Furthermore, the reservoir model can be extended to a bidirectional and generative setup which is of particular interest in the context of sensory-motor tasks [1, 5]. A single network then learns both relations between input and output, and the reverse mapping that can be used to predict or fill in missing sensory inputs. In this context, feedback of the network output into the reservoir is a key ingredient. Although there are successful applications reported that use reservoir networks with output feedback [1, 5, 6], the effects of output feedback have not been investigated systematically. In principle, we expect output feedback to add information but also to cause additional transients, especially when online learning is applied. Under the hypothesis of attractor-based computation, we expect performance to increase if the network is allowed to converge first to an attractor before read-out is applied. If transients are crucial, it is difficult to explain robustness and generalization because there is no regularizing factor which allows to suppress noise in the transients. To verify convergence in the experimental setting, we use the network state change as criterion and monitor the standard Hopfield energy.

ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

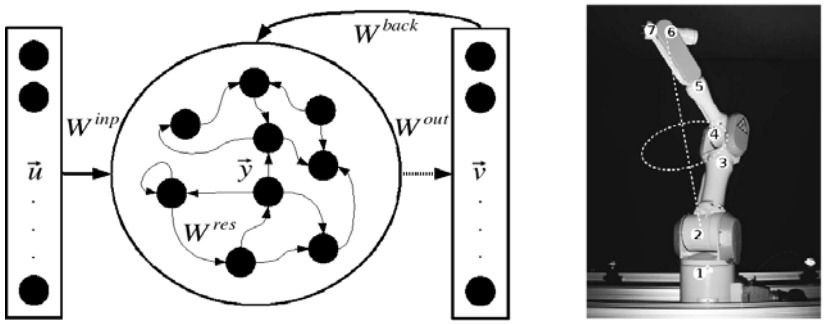

Fig. 1: (left) Reservoir computing network setup. (right) PA-10 robot arm.

2 Reservoir computing: a simple recurrent network model

We focus on a reservoir network setting as depicted in Fig. 1 (left). The network architecture comprises a recurrent reservoir network of nonlinear neurons interconnecting input and output. All but the read-out weights W^{out} are randomly initialized with small weights and stay fixed. W^{back} can add feedback from the output neurons to the reservoir. We consider the recurrent network dynamics

$$\mathbf{y}(k+1) = \mathbf{f}\left(\mathsf{W}^{inp}\mathbf{u}(k) + \mathsf{W}^{res}\mathbf{y}(k) + \mathsf{W}^{back}\mathbf{v}(k)\right) \quad (1)$$

$$\mathbf{v}(k+1) = \mathsf{W}^{out}\mathbf{y}(k). \quad (2)$$

$\mathbf{u}, \mathbf{y}$ and $\mathbf{v}$ are the input, reservoir and output neurons, where $\mathbf{y}$ is obtained by applying parametrized activation functions component-wise. We denote the network state by $\mathbf{z}(k) = (\mathbf{u}(k)^T, \mathbf{y}(k)^T, \mathbf{v}(k)^T)^T$.

Reservoir optimization: Optimizing the randomly initialized reservoir is significant for learning success. Several approaches have been proposed [2, 4, 6]. We present our results for online reservoir optimization by intrinsic plasticity and beforehand weight-scaling such that the spectral radius is near unity.

Scaling the spectral radius (SR) of the weight matrix is commonly used in echo state networks [2, 4]. We use hyperbolic tangent activation functions $y_i = f(x_i) = tanh(x_i)$ in combination with SR scaling. Usually, a spectral radius around unity is preferred, but the concrete choice depends on the task at hand.

Intrinsic plasticity (IP) was introduced to reservoir computing in [6]. It is an online and unsupervised self-adaptation rule that optimizes a neuron's excitability by adapting the parameters a_i, b_i of the activation functions $y_i = f_i(x_i, a_i, b_i) = (1 + \exp(-a_i x_i - b_i))^{-1}$ used in the network equation (1).

Read-out learning: Only the read-out weights W^{out} are trained by either backpropagation-decorrelation (BPDC), a supervised online training scheme [6], or ridge regression. In the proposed setup, where no recurrent connections between the output neurons exist, the BPDC learning rule simplifies to

$$\Delta w_{ij}^{out}(k) = \frac{\eta}{\|\mathbf{y}(k-1)\|^2 + \varepsilon}\, y_j(k-1)\,(d_i(k) - v_i(k)). \quad (3)$$

ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

Algorithm 1 Convergence algorithm

Require: get external input $\mathbf{u}(k)$
Require: set $t=0$, $\Delta z=\infty$, $\delta=10^{-6}$ and $t_{max}=100$
1: **while** $\Delta z > \delta$ and $t < t_{max}$ **do**
2: inject external input $\mathbf{u}(k)$ into network
3: execute network iteration (1), (2)
4: compute state change $\Delta z = \|\mathbf{z}(t) - \mathbf{z}(t-1)\|^2$
5: $t=t+1$
6: **end while**
7: return t

$d_i(k)$ is the desired target value of output i at time step k. For each input pattern $\mathbf{u}(k)$, the network dynamics (1), (2) are iterated first, before the readout weights W^{out} are adapted according to (3). For more details about batch learning with reservoir networks see [2].

3 Transient or attractor-based computation?

We investigate whether reservoir networks of analog neurons compute with transient dynamics (temporal elusive but reproducible traces in the network state trajectory) or (fixed point) attractor states in an exemplary sequence transduction scenario. The idea is to eliminate transients and measure the error, which we expect to increase if computation is based on transients. Transients are removed by the convergence procedure outlined in Algorithm 1: it iterates the network dynamics (1), (2) with clamped input pattern $\mathbf{u}(k)$ until the network state change Δz falls below a small constant δ. We show later on that the network state actually converges.

Test setting: The network model is applied to learn observed motion in joint and task space of the redundant Mitsubishi PA-10 robot arm (shown in Fig. 1 (right)), a task introduced in [5]. Training data is generated by projecting a task trajectory specified in Cartesian endeffector coordinates into the joint space of the PA-10 robot arm by means of the analytically calculated inverse kinematics. We compute for each task space input $(u_1(k), u_2(k), u_3(k))^T$ (with fixed orientation pointing upwards along the z-axis) the 7-dim target vector $(d_1(k), \ldots, d_7(k))^T$. Thus the network in Fig. 1 (left) is set up with 3 input and 7 output neurons. For more details see appendix A.

Eliminating transients: harm or bless? Fig. 2 and Table 1 show the mean square errors for online and batch learning of four model variants (SR/IP and no/with output feedback) operating in two conditions: (i) presentation of the input patterns $\mathbf{u}(k)$ without convergence (standard operation mode) and (ii) presentation of each pattern $\mathbf{u}(k)$ until the network state converges and then calculating the error. Note that Algorithm 1 is not used for training.

Surprisingly, all networks perform better in the convergence condition. Without output feedback the errors are almost equal in both conditions, but conver-

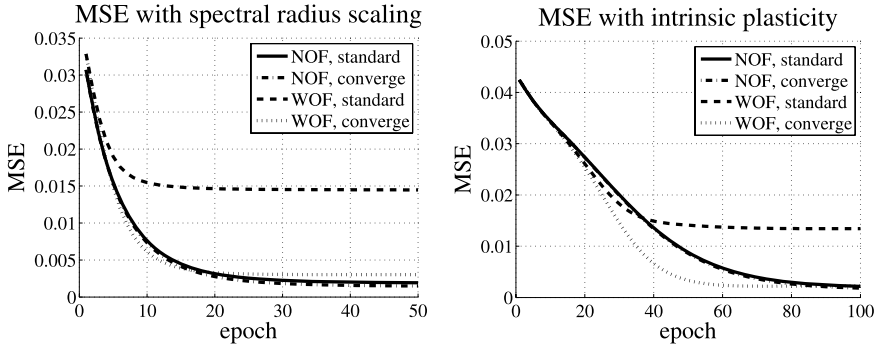

Fig. 2: Mean square errors for online learning averaged over 100 network initializations per setup variant operating in standard or converge mode: (left) spectral radius scaling, (right) intrinsic plasticity learning, no/with output feedback (NOF/WOF). Inter-trial variance of errors (not shown) is marginal.

gence improves the performance significantly if the output is fed back into the reservoir. We also observe this phenomenon for batch learning with ridge regression (see Table 1). We suggest the reason for this effect is that feedback of defective outputs causes additional transients. Note that the network output is teacher-forced during training meaning injection of exact feedback $\mathbf{d}(k)$ into the network. Learning introduces errors when testing and thus defective feedback which perturbs the network state. Hence, a different, transient network state is read out explaining the increasing error. Obviously, convergence recovers the network state apparent during training by balancing external input, output feedback and internal reservoir dynamics. Although performance is best without feedback, the errors are competitive when Algorithm 1 is applied.
In general, contraction to stable states improves the performance indicating an attractor-based computation in this setting. Transients are perturbations and are induced by the network dynamics (1), (2) during transition between two attractor states. Remarkable are the similar results for batch and online learning. Online learning copes with sequential data presentation in the reservoir setting.
Convergence behavior and network energy: The results above indicate an attractor-based encoding of the input in the reservoir network. But do the networks actually converge to stable states?

res. opt.	**feedback**	**standard**	**converge**
SR	no	$0.0016 \pm 4.1 \cdot 10^{-10}$	$0.0012 \pm 4.9 \cdot 10^{-10}$
	yes	$0.0142 \pm 5.5 \cdot 10^{-6}$	$0.0014 \pm 4.2 \cdot 10^{-8}$
IP	no	$0.0016 \pm 1.7 \cdot 10^{-9}$	$0.0013 \pm 1.8 \cdot 10^{-9}$
	yes	$0.0133 \pm 4.0 \cdot 10^{-6}$	$0.0015 \pm 5.9 \cdot 10^{-8}$

Table 1: Mean square errors with variances for batch learning by ridge regression. Results are averaged over 100 network initializations per condition.

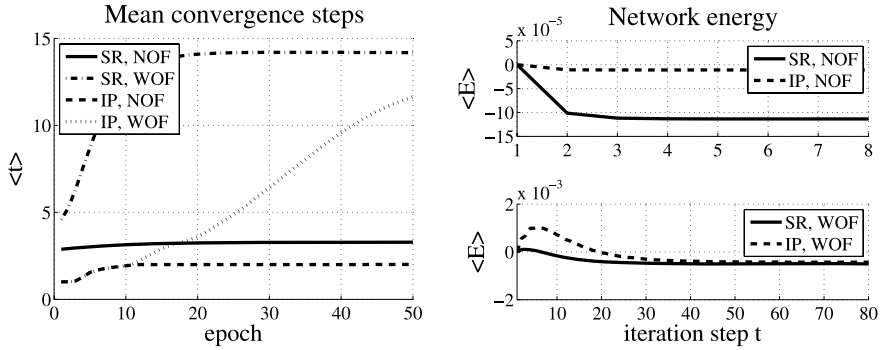

Fig. 3: Network properties: (left) Average number of steps until network state converges. (right) Relative network energy during convergence (in last epoch).

In Fig. 3 (left) the average number of iterations $\langle t \rangle$ needed for convergence are plotted over the learning epochs. Important is that the networks actually converge, i.e. $\Delta z \leq \delta$, within the maximal allowed amount of iterations t_{max} (see Algorithm 1). Convergence takes several iterations with clamped input meaning transients are actually apparent and the results presented in Fig. 3 confirm that feedback increases duration of these transients.
To complement the results so far, we monitor the network energy

$$E(t) = -\sum_{i,j} z_i(t)\, w_{ij}\, z_j(t) + \sum_i b_i\, z_i(t)$$

for every iteration step t during convergence. We account for different absolute network energies for each input pattern $\mathbf{u}(k)$ and network initialization by presenting relative energies. The network energy anneals to a fixed value below zero relative to the energy $E(0)$ when starting convergence, as shown in Fig. 3 (right). With output feedback, we also observe a slower convergence of the network energy preceded by an overshoot. However, the stable states belong to valleys in the energy landscape and hold the property $\Delta z \approx 0$, i.e. they are attractors.

4 Conclusion and discussion

We conclude that reservoirs with small weights stay stable despite defective output feedback and cope with online learning from time-series. The presented convergence algorithm provides a technique to fully exploit the network capabilities. The results are consistent for all experimental variations we considered: online or batch learning with reservoir optimization by IP or SR scaling using Fermi- or *tanh*-activation functions. Note that convergence erases the network's memory, which is crucial for prediction tasks with branches depending on the history. In this context, it is of special interest how transients with appropriate properties can be obtained [3]. But this is not critical for the presented task. We use the reservoir model as associative memory with a hidden layer to learn a

static mapping, where the fixed feature generation by the reservoir prevents destructive interference during online read-out adaptation. The system's attractor is a highly nonlinear and smooth representation of the current input, explaining excellent generalization [5], whereas the network dynamics limit the speed of convergence to the attractor. The empirical results presented here demand for a rigorous mathematical analysis of attractor-based input encoding by reservoirs.

A Appendix

Network parameter: All experiments are conducted with reservoirs of 100 neurons. The reservoir neurons are fully connected with input and output. The input weights W^{inp} are randomly initialized in range $[-0.2, 0.2]$. Connections between reservoir nodes are sparse (only 20% of W^{res} occupied) and initialized with values ranging from -0.05 to 0.05. In case of output feedback, W^{back} is set to random values out of $[-0.1, 0.1]$, otherwise $\mathsf{W}^{back} \equiv \mathbf{0}$. We present results for reservoir optimization by IP (desired mean activity $\mu = 0.2$ and learning rate $\eta_{IP} = 0.0002$) and without IP. In the latter case, we scaled the spectral radius of the weight matrix to approximately 0.95.
Training: We use a figure eight-like motion as training data. In task space, endeffector positions having fixed orientation and are defined by

$$u_1(k) = 0.15\ cos(\bar{\omega}k), \quad u_2(k) = 0.15\ sin(2\bar{\omega}k), \quad u_3(k) = 0.9.$$

For $\bar{\omega} = 2\pi/100$ we obtain patterns with a period of 100 samples. Training and test phases comprise 1000 samples each and were preceded by a washout phase lasting for a single pattern period. During training, the network is teacher-forced with the desired target values. Parameters for online BPDC learning (3): learning rate $\eta = 0.05$, weight-decay $\lambda = 10^{-5}$ and a small regularization constant $\varepsilon = 0.002$. For batch learning we use ridge regression with Tikhonov factor $\alpha = 1$, where in case of IP the network is pretrained for 100 epochs.

References

[1] E. Antonelo, B. Schrauwen and J. V. Campenhout. Generative Modeling of Autonomous Robots and their Environments using Reservoir Computing. NPL, 26(3):233–249, 2007.

[2] H. Jäger. Adaptive nonlinear system identification with echo state networks. In *NIPS*, pages 593–600, 2002.

[3] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[4] M. C. Ozturk, D. Xu, and J. C. Principe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.

[5] R. F. Reinhart and J. J. Steil. Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. *ECSIS Symp. on LAB-RS*, pages 35–40, 2008.

[6] J. J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.