Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2006) 577–591
NEUROCOMPUTING
www.elsevier.com/locate/neucom

PIRANHA: Policy iteration for recurrent artificial neural networks with hidden activities

István Szita, András Lőrincz*

Department of Information Systems, Eötvös Loránd University, Pázmány Péter sétány 1/C, Budapest, Hungary H-1117

Received 19 January 2005; received in revised form 8 September 2005; accepted 16 September 2005
Available online 10 February 2006
Communicated by J. Vandewalle

Abstract

It is an intriguing task to develop efficient connectionist representations for learning long time series. Recurrent neural networks have great promises here. We model the learning task as a minimization problem of a nonlinear least-squares cost function, that takes into account both one-step and multi-step prediction errors. The special structure of the cost function is constructed to build a bridge to reinforcement learning. We exploit this connection and derive a convergent, policy iteration-based algorithm, and show that RNN training can be made to fit the reinforcement learning framework in a natural fashion. The relevance of this connection is discussed. We also present experimental results, which demonstrate the appealing properties of the unique parameter structure prescribed by reinforcement learning. Experiments cover both sequence learning and long-term prediction.

Keywords: Recurrent neural networks; Policy iteration; Sequence learning; Multi-step prediction

1. Introduction

Humans are able to learn long sequences of data (for example, melodies, lyrics etc.) just by observing them a couple of times. This type of learning has interesting characteristics: if a small part of a previously learnt sequence is presented, it is effortless to continue the sequence from that point on. On the other hand, it is much harder to access the data directly, e.g. as the "20th line of a poem" than sequentially. Replay in the reverse direction is anything but effort free. These properties imply that this kind of memory works in an essentially different way than databases or lookup-tables do. Our aim is to provide an algorithm that is able to mimic this type of learning, specifically it should be able to (i) store long data sequences, (ii) recognize portions of the stored sequence, and (iii) continue it sequentially and unidirectionally.

Recurrent neural networks (RNNs) are attractive for our goal: Horne and Hush [16] have shown that, in theory, an RNN with n neurons is capable of storing a sequence of length $O(n^2)$. However, encodings proposed to date are not likely to reach this bound: they either require hand-wiring of weights, like in [2], or apply one-step prediction methods iteratively to obtain multi-step outputs.

Traditional long-term prediction methods either work as iterated 1-step methods (Fig. 1(a)) or by direct learning of the k-step-ahead value (Fig. 1(b)) (for a review on these subjects, see [6]). In the former scheme, during the learning phase, prediction errors are computed relative to the previous sequence values, so errors do not accumulate. However, this does not capture well the behavior in the testing phase, when errors cannot be corrected step-by-step, so they accumulate rather quickly. The latter scheme escapes this trap, but it needs a different estimator for each lookahead time k, which is far from being economic (although implementations exist, see, e.g. [11]). We take a novel approach: perform iterated prediction without correction, and formulate an objective function that directly takes all the prediction errors into account (see Fig. 1(c)).

*Corresponding author. Tel.: +36 209 0555/8473.
E-mail addresses: szityu@eotvos.elte.hu (I. Szita), andras.lorincz@elte.hu (A. Lőrincz).
URL: http://nipg.inf.elte.hu.

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.09.017

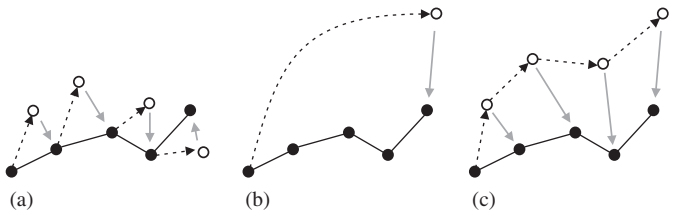

Fig. 1. Comparison of time series learning methods: (a) iterated 1-step prediction; (b) single k-step prediction; (c) single sequence replay prediction. Black dots: the original time series; dotted arrows: predictions; gray arrows: corrections during learning.

The resulting objective function is a least-squares function of highly nonlinear terms, being intractable for traditional minimization techniques. Further, there are many possibilities in combining the different nonlinear terms of the optimization. We proceed by showing that a particular form of the optimization task can be interpreted as a *reinforcement learning* (RL) problem of a hypothetical agent in an abstract environment. This connection relaxes the large number of optimization parameters to the single parameter of discounting of RL and enables the minimization of our objective function using a version of policy iteration.

The outline of the paper is as follows. First, we define the learning task in Section 2. In Section 3 we derive the learning rules of our algorithm. Section 4 provides theoretical analysis of the algorithm and investigate some practical aspects. Computational demonstrations are provided in Section 5, and finally, Section 6 reviews related works and summarizes our results.

2. Definitions and basic concepts

2.1. The network architecture

We consider fully connected RNNs with m input neurons and $n>m$ hidden neurons. We do not use a separate output layer; the states of the first m hidden neurons are considered as outputs. The state of each neuron is a real number in the interval $[-1,+1]$, because neurons admit activation–squashing functions, which maps onto interval $[-1,+1]$. An example is function $\sigma(z)=\tanh(z)$. The input, output and the hidden state at time t are denoted by $v_t \in [-1,+1]^m$, $\hat{x}_{t+1} \in [-1,+1]^m$ and $u_t \in [-1,+1]^n$, respectively. No explicit bias term is applied, instead a constant 1 makes the $(m+1)$st component of the input. Let the recurrent, the input and the output weight matrices be denoted by $F \in \mathbb{R}^{n\times n}$, $G \in \mathbb{R}^{n\times m+1}$, and $H \in \mathbb{R}^{m\times n}$, respectively. Let H be a fixed matrix projecting to the first m components of u_t. The dynamics of the network is as follows:

$$u_{t+1} = \sigma(Fu_t + Gv_t), \tag{1}$$

$$\hat{x}_{t+1} = Hu_{t+1}, \tag{2}$$

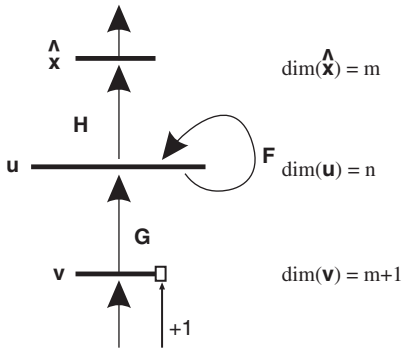

Fig. 2. Recurrent neural network. G: input weight matrix; F: recurrent weight matrix; H: output weight matrix; v: input with threshold (constant $+1$); u: hidden layer; $\hat{x}$: output.

with the initial condition $u_0 = \mathbf{0}$. The network is depicted in Fig. 2.

2.2. Problem description

Suppose that a time series $\{x_t \in \mathbb{R}^m \mid t = 1,2,3,\ldots\}$ and a sequence length T is given. We have to find weight matrices F and G so that the network is able to replay the first T elements of the sequence in correct temporal order, i.e. for any $1<t<T$, having input $v_{t'} := x_{t'}$ for $t' \leqslant t$ and generating $v_{t'} := \hat{x}_{t'}$ for $t<t' \leqslant T$, the total squared error $\sum_{t'=t}^{T} \|\hat{x}_{t'} - x_{t'}\|^2$ is small.

The sum of the least-square errors for all possible t' values characterizes the replay capability of the network with the network weights given. However, this cost function may have pitfalls for ordinary gradient-based methods, because little changes in weights may considerably influence the output many steps ahead. Sensitivity may become crucial under this condition.

3. Construction of the learning algorithm

The traditional approach for solving the problem would be to search for weight matrices F and G that minimize the reconstruction error

$$\sum_{t=0}^{T} \|\hat{x}_t - x_t\|^2 = \sum_{t=0}^{T} \|Hu_t - x_t\|^2 \tag{3}$$

subject to constraints (1)–(2). It is well known that such recurrent network optimization tasks are hard, because the objective function has many local minima and is often ill-conditioned. Let us point out some possible reasons: consider the case when all the predicted output values $\hat{x}_t$ are correct, except for a single time step t_0. The hidden state u_{t_0} for this time step is also bad, and we can modify it only by modifying the weights F and G. However, this modification is likely to compromise all the other errors, resulting in a total increase in the objective value. The reason for this phenomenon is that the hidden states for different time steps are very strongly coupled by Eq. (1).

3.1. Relaxing the constraints

The underlying idea of our approach is that if this coupling is relaxed, the resulting error surface may become smoother. However, it is easy to see that by doing so, we lose all guarantees on the quality of long-term predictions: consider the case when we choose all u_t so that $Hu_t = x_t$ and set F and G arbitrarily. Then the reconstruction error (3) is zero, but multi-step predictions are likely to diverge quickly.

3.2. Hint to the solution: connection to reinforcement learning

Notice that a similar situation occurs in RL problems (for a detailed introduction to RL see [26]), where one aims to minimize the costs of an agent in the long run. However, there may be states with low immediate costs, but every path from them has high cost (just like network states with no reconstruction error, but high prediction error). Although there is no explicit agent in the network learning task, it can be reformulated as a special-case RL problem for a hypothetical agent.

The state of this hypothetical agent is all the state information of the RNN, i.e. the whole sequence $\mathbf{U} = (u_0, u_1, u_2, \ldots)$, an action is a step of the network using some weight matrix (F, G), and a policy is a sequence of such actions. The immediate cost of state $\mathbf{U}$ is the reconstruction error (3), so it is easy to see that the cumulated cost function will be the sum of k-step prediction errors (for details, see Appendix). We shall derive this formally in the following section.

The great advantage of this analogy to RL is that it yields an appropriate cost function to be minimized, and we can also apply standard RL methods for this task. Fortunately, it turns out that one of the methods, *policy iteration*, reduces to a simple gradient descent method.

3.3. Multi-step prediction errors

Suppose that the sequence $\mathbf{U} = (u_0, u_1, u_2, \ldots)$ is an arbitrary state sequence, not necessarily belonging to an RNN. When can we say that matrices F, G and a state u_t in the sequence is a good predictor? The one-step prediction from u_t is

$$\hat{x}_{t\to(t+1)} = H\sigma(Fu_t + Gx_t), \tag{4}$$

for two steps it is

$$\begin{aligned}\hat{x}_{t\to(t+2)} &= H\sigma(F\sigma(Fu_t + Gx_t) + G\hat{x}_{t\to(t+1)}) \\ &= H\sigma((F + GH)\sigma(Fu_t + Gx_t)),\end{aligned} \tag{5}$$

and so on. Let us introduce the notation

$$s^{(1)}_{t,(F,G)}(u) := \sigma(Fu + Gx_t) \tag{6}$$

for the effect of (F, G) on a single RNN state $u \in \mathbb{R}^n$ and

$$s^{(k)}_{t,(F,G)}(u) := \sigma((F + GH)s^{(k-1)}_{t,(F,G)}(u)) \tag{7}$$

to describe the effect of applying (F, G) $k(>1)$ times on state u. For later use we also define $s^{(0)}$ as the identity function.

Using these notations, the k-step prediction error of state u_t is

$$(Hs^{(k)}_{t,(F,G)}(u_t) - x_{t+k}). \tag{8}$$

(For the sake of notational simplicity, we assume that the input sequence x_t is defined for $t > T$ as well.) The overall cost of state u_t should be the sum of the k-step prediction error norms, for all time instants and for all ks. To ensure convergence, we use a geometrically decaying weight sequence with decay rate γ. So the total cost of the state sequence $\mathbf{U}$ is

$$J(\mathbf{U}, F, G) = \sum_{k=0}^{\infty}\sum_{t=1}^{T} \gamma^k \|H(s^{(k)}_{t,(F,G)}(u_t)) - x_{t+k}\|^2. \tag{9}$$

Note that if a set of weights (F, G) minimizes Eq. (3) subject to Eq. (1), it also minimizes Eq. (9), and technical assumptions can ensure that the converse also holds (for a rigorous statement of this claim, see Appendix). Note also that cost function in Eq. (9) has a large number of partial sequences and so, it has an enormous number of adjustable quantities. This gives one great freedom in accomplishing the minimization. However, there is a price to pay: minimization is ill-posed, because the number of parameters is much larger than the number of data points. We restrict the choices to the parameters of the original problem, but in a way which is different from Eq. (3), and which reflects the structure of the sequence replay problem better.

3.4. Digression: policy iteration

Let us return to the RL analogue of the problem. A standard way to find a policy (action set) that minimizes the cumulated cost function (in our case, (9)) is *policy iteration*. Here we give only a very brief and informal description. More details are provided in the Appendix. Policy iteration starts with an arbitrary policy, then iterates an evaluation step and an improvement step. In the evaluation step, the cost of the current policy is calculated in all relevant states, which is done usually by following the policy for many steps, and approximating the state costs. Knowing the previously approximated costs, an improved policy can be easily computed, and for this, we need only *one-step* lookahead (for details, see Appendix). This is done in the policy improvement step. Subsequently, we can evaluate this new policy, then improve it, and so forth.

3.5. The sketch of the algorithm

In this part, we formalize the policy iteration approach to obtain an algorithm for minimizing (9). To this end, note that if a particular state u_t is a good predictor at time step t using weights (F, G), then $\sigma(Fu_t + Gx_t)$ is a good predictor

at time step $t+1$, following from the structure of the cost function. Let

$$S_{(F,G)}\mathbf{U} := (u'_0, u'_1, \ldots, u'_T) \quad \text{where}$$
$$u'_0 = \mathbf{0},$$
$$u'_{t+1} = \sigma(Fu_t + Gx_t) \quad \text{for } 0 \leqslant t < T. \tag{10}$$

Using this notation the above statement says that if $\mathbf{U}$ is a good predictor with (F,G), then $S_{(F,G)}\mathbf{U}$ is also a good predictor. This argument can be applied iteratively, yielding the result that $S^k_{(F,G)}\mathbf{U}$ is a good predictor for $k>1$. However, it is easy to see that for $k>T$ $S^k_{(F,G)}\mathbf{U}$ is equal to the state sequence generated by (1), and will be denoted by $\mathbf{U}_{(F,G)}$:

$$\mathbf{U}_{(F,G)} := (u_0, u_1, \ldots, u_T) \quad \text{where}$$
$$u_0 = \mathbf{0},$$
$$u_{t+1} = \sigma(Fu_t + Gx_t) \quad \text{for } 0 \leqslant t < T. \tag{11}$$

This justifies the following improvement scheme:

- fix $\mathbf{U}$, F_0, G_0;
- find (F,G) for which $J(S_{(F,G)}\mathbf{U}, F_0, G_0)$ is small;
- continue with $\mathbf{U} := \mathbf{U}_{(F,G)}$, $F_0 := F$ and $G_0 = G$.

In Section 4 we prove the correctness of the method, but before doing so, we elaborate on some of the details.

First of all, note that the number of adjustable parameters is the same as in the original method, but the parameters F and G play a different role: they are chosen so that they minimize the multi-step prediction errors, and relation (1) is part of the full optimization problem.[1]

3.6. Computing the gradient

In this part we work out the details of the above scheme. Let us denote the state sequence at iteration i by $\mathbf{U}_i$, and the weight matrices by F_i and G_i, respectively. We would like to find F and G so that

$$\begin{aligned} J'(F,G) &:= J(S_{(F,G)}\mathbf{U}_i, F_i, G_i) \\ &= \sum_{t=1}^{T}\sum_{k=1}^{\infty} \gamma^k \| H(s^{(k-1)}_{t+1,(F_i,G_i)} s^{(1)}_{t,(F,G)}(u_t)) - x_{t+k} \|^2 \\ &:= \sum_{t=1}^{T}\sum_{k=1}^{\infty} \gamma^k \| e_{t,k} \|^2 \end{aligned} \tag{12}$$

is minimized, that is, we are minimizing the cost by propagating all states u_t by (F,G) once, and then by propagating with (F_i, G_i) further.

In order to do this, we have to compute the gradient of the cost function with respect to the weights. For any $1 \leqslant a,b,c \leqslant n$ and $1 \leqslant d \leqslant m+1$, the gradients are given by

$$\left(\frac{\partial J'}{\partial F_{ab}}\right)(F_i, G_i) = \sum_{t=1}^{T}\sum_{k=1}^{\infty} \gamma^k (e_{t,k})^\top H[m(k)]_a [u_t]_b \tag{13}$$

and

$$\left(\frac{\partial J'}{\partial G_{cd}}\right)(F_i, G_i) = \sum_{t=1}^{T}\sum_{k=1}^{\infty} \gamma^k (e_{t,k})^\top H[m(k)]_c \begin{bmatrix} x_{t+k} \\ 1 \end{bmatrix}_d, \tag{14}$$

where

$$m(k) := \prod_{j=1}^{k} \sigma'([s^{(j-1)}_{t,(F_i,G_i)} u_t]) F_i, \tag{15}$$

$(.)^\top$ denotes transposition and $[v]_b$ denotes the bth component of vector v.

If we can ensure that the terms $m(k)$ remain bounded, then the above sums are convergent, because all terms can be bounded by $C \cdot \gamma^k$ (for details see the proof of Lemma 3 in the Appendix). This means that if K is a sufficiently large positive number, then the gradients of

$$\hat{J}'(F,G) := \sum_{t=1}^{T}\sum_{k=1}^{K} \gamma^k \| e_{t,k} \|^2 \tag{16}$$

will be arbitrarily close to (13) and (14), so we can use it as an approximation.

Using the above formulae, we can obtain weight matrices better than (F_i, G_i) by taking a gradient descent step. The description of the algorithm is therefore complete. We have summarized it in Fig. 1, which will be called *Policy Iteration for Recurrent Artificial Neural Networks with Hidden Activities*, or *PIRANHA* for short.

4. Analysis

The convergence analysis of the algorithm described in Table 1 is easy, because it is a gradient descent method, therefore, as it is well known, it converges to a (local) minimum if the step sizes α_i are sufficiently small. The only remaining question is whether gradients (13) and (14) are convergent. In this case, for sufficiently large K, the gradients of (12) will be approximated well by (16).

It turns out that this condition can be ensured for sufficiently small discount factor γ. The proof can be found in the Appendix.

4.1. Practical considerations

4.1.1. The natural gradient

Although we have just proved that PIRANHA is convergent, convergence can be quite slow for nontrivial learning tasks. This is a common problem of gradient-based methods over nonlinear neural networks operating on squashing functions that can have very small slopes. This problem is quite severe in the case of PIRANHA, because we have to differentiate expressions with many

[1] Several numerical simulations showed the step $\mathbf{U} := \mathbf{U}_{(F,G)}$ could be substituted by $\mathbf{U} := S_{(F,G)}\mathbf{U}$ or $\mathbf{U} := S^k_{(F,G)}\mathbf{U}$ for any k and under such conditions, constraint (1) does not appear at all. However, theoretical analysis is easier for the definition, which includes Eq. (1) and this definition will be used here.

Table 1
Pseudo-code of the PIRANHA algorithm

```
input: α, K, T, (x_1, ..., x_T);
initialize F_0, G_0; i:=0;
for i:=1 to max_iter
  U_i:=U_(F_i,G_i);
  for t:=0 to T
    m_t(1):=F_i;
    for k = 1 ... (K − 1),
      m_t(k+1):=m_t(k) · σ'([s^(k)_(t,(F_i,G_i)) u_t]) F_i;
    e_{t,k}:=H(s^(k−1)_(t+1,(F_i,G_i)) s^(1)_(t,(F,G))(u_t)) − x_{t+k};
  [ΔF]_ab:=Σ_{t=1}^{T} Σ_{k=1}^{K} γ^k (e_{t,k})^⊤ H [m_t(k)]_a [u_t]_b;
  [ΔG]_cd:=Σ_{t=1}^{T} Σ_{k=1}^{K} γ^k (e_{t,k})^⊤ H [m_t(k)]_c [x_{t+k}; 1]_d;
  F_{i+1} = F_i − α_i · ΔF;
  G_{i+1} = G_i − α_i · ΔG;
  i:=i + 1;
end
```

nested sigmoid functions, so the gradient may be vanishingly small.

Numerous methods have been proposed to ease this problem, e.g. the momentum method, or various adaptive learning rate methods, but we chose the natural gradient method of Amari et al. [1], because of its relative simplicity and high efficiency (Amari has reported a problem where it accelerated learning by a factor of 1000 over a traditional multi-layer perceptron). One of the basic observations underlying the natural gradient is that the convergence of gradient descent is preserved if the gradient is preconditioned by a positive definite matrix, i.e. the update rule $F_{i+1}:=F_i-\alpha\cdot\Delta F$ has the same convergence properties as $F_{i+1}:=F_i-\alpha\cdot M\cdot\Delta F$ for *any* positive definite matrix M. Amari found the inverse of the transformation matrix from the Riemannian space of perceptron parameters to Euclidean space particularly effective. According to our calculations, this matrix can be approximated by a matrix $\bar{Q}$, which has entries

$$\bar{Q}_{(ab),(cd)}=\frac{1}{I}\sum_{i=1}^{T}\sum_{t=0}^{T}(q_{ab}^{(i,t)})(q_{cd}^{(i,t)}), \tag{17}$$

at iteration I, where $q^{(i,t)}$ is an n^2 vector with the abth component being the tth term of (13) at iteration i:

$$q_{ab}^{(i,t)}:=\sum_{k=1}^{\infty}\gamma^k(e_{t,k})^{\top}H[m(k)]_a[u_t]_b \tag{18}$$

and $M:=\bar{Q}^{-1}$. According to Amari, M can be updated adaptively. The preconditioning matrix for G can be derived in a similar fashion.

4.1.2. Regularization

We have found that—similarly to other tuning techniques—the weights F and G tend to grow indefinitely, resulting in unstable behavior. A standard way of weight regularization was applied to avoid this obstacle: we added a quadratic regularization term $(c_1/2)\sum_{a,b}(F_{ab})^2+(c_1/2)\sum_{c,d}(G_{cd})^2$ to J'. Weight regularization can be justified theoretically either from a statistical point of view (Gaussian prior over the weights) or from an analytical point of view: using weight regularization, one can efficiently control ill-conditioned weight matrices by bounding its eigenvalues both from beneath and above. This topic is investigated in great details in [22].

Preconditioning by the natural gradient introduces another source of instability: as Amari has pointed out, at long plateaus of the cost function, $\bar{Q}$ tends to be near-singular, and consequently, M grows indefinitely. The standard procedure in this case is to add a regularization term $c_2\cdot I$ to $\bar{Q}$, where I is the identity matrix. As this step preserves the positive definiteness of M, convergence is preserved. This update rule can be performed adaptively as well.

5. Experiments

5.1. Chaotic time series: the Mackey–Glass series

We chose the Mackey–Glass chaotic time series [19] for demonstration purposes, which is a commonly used benchmark problem. The standard parameter set $\tau=17$, $a=0.2$, $b=0.1$, $c=10$, and a sampling rate of 6 s was applied. Data were scaled to interval $[-1;+1]$.

Discount rate γ was set to 0.9, and learning rate decayed as $\alpha_i=1/15i$. We have applied preconditioning by the natural gradient with regularization and also weight regularization with coefficients $c_1=c_2=0.01$. The input of the network was only the actual entry of the time series, no past data were given. Networks were trained on a short series with $T=200$.

After training each network, they were evaluated by setting $u_1=\mathbf{0}$, and presenting N_1 consecutive inputs from the time series starting from some randomly chosen t_0. Upon observation, the networks had to reproduce the following N_2 values of the time series without external input. The expected average squared replay error

$$RE_{N_1,N_2}=E_{t_0}\left(\frac{1}{N_2}\sum_{t=t_0+N_1+1}^{t_0+N_1+N_2}|\hat{x}_t-x_t|^2\right) \tag{19}$$

was approximated by averaging the errors for 100 consecutive samples with starting points chosen randomly. N_1 and N_2 were fixed to 10.

5.1.1. The effect of the lookahead parameter

In our first experiment we examined how the lookahead length parameter affects performance. $T=200$, $n=8$ were set, and K (the upper limit of the summation over the lookahead parameter) varied from 1 to 10. For each value of K, we performed five test runs with random initial weights, and measured the replay errors of the resulting networks. The results are summarized in Fig. 3 and show that PIRANHA performs well for small values of K already ($K\geqslant 3$).

5.1.2. The effect of the number of neurons

In another test the effect of the number of neurons on the replay capability of the network was investigated. We varied the number of neurons from 2 to 12 by increments of 2 and used fixed $K = 5$ with five runs for each n. The results (Fig. 4(a)) show that networks with moderate number of neurons ($n \geqslant 8$) can already be trained by PIRANHA to reproduce the data with good accuracy.

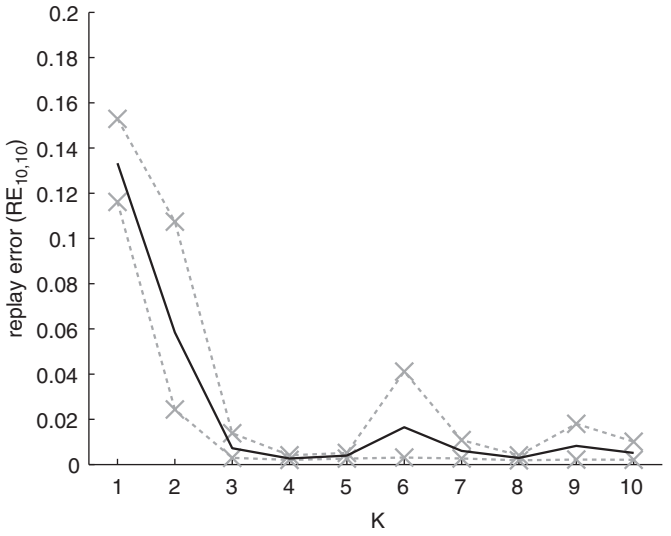

Fig. 3. Replay error vs. lookahead parameter K. The average replay error $RE_{10,10}$ of eight-neuron networks, after 200 epochs of training. Black line: average of five test runs; gray dashed lines: best and worst runs.

Although PIRANHA was designed to train RNNs to reproduce a fixed data sequence, the architecture is capable for multi-step predictions. In the evaluation phase, previously unseen sequences (instead of portions of the training data) were fed into the network and the outputs of the network can be considered as predictions on subsequent inputs. To test the prediction capabilities of our method, we evaluated the networks obtained in the previous training process with unseen data. The method of evaluation was the same (after observing 10 samples, the next 10 samples had to be predicted), but the test sequences were selected randomly from interval 201–10,000. The results are shown in Fig. 4(b), which looks almost identical to Fig. 4(a). These results seem to indicate that despite its chaotic nature, the Mackey–Glass-17 time series has a relatively simple underlying structure, and small networks trained by PIRANHA managed to approximate this dynamics and are able to provide reliable long-term predictions. This hypothesis is justified by plotting the predictions of specific networks: Fig. 5 shows that after an observed short period of the dynamics and tuning for this short time period, the network is able to imitate the MG17 sequence very closely.

The situation is slightly different for Mackey–Glass-30 time series, which is more chaotic. The performance of PIRANHA deteriorates considerably (Fig. 4(c)). However,

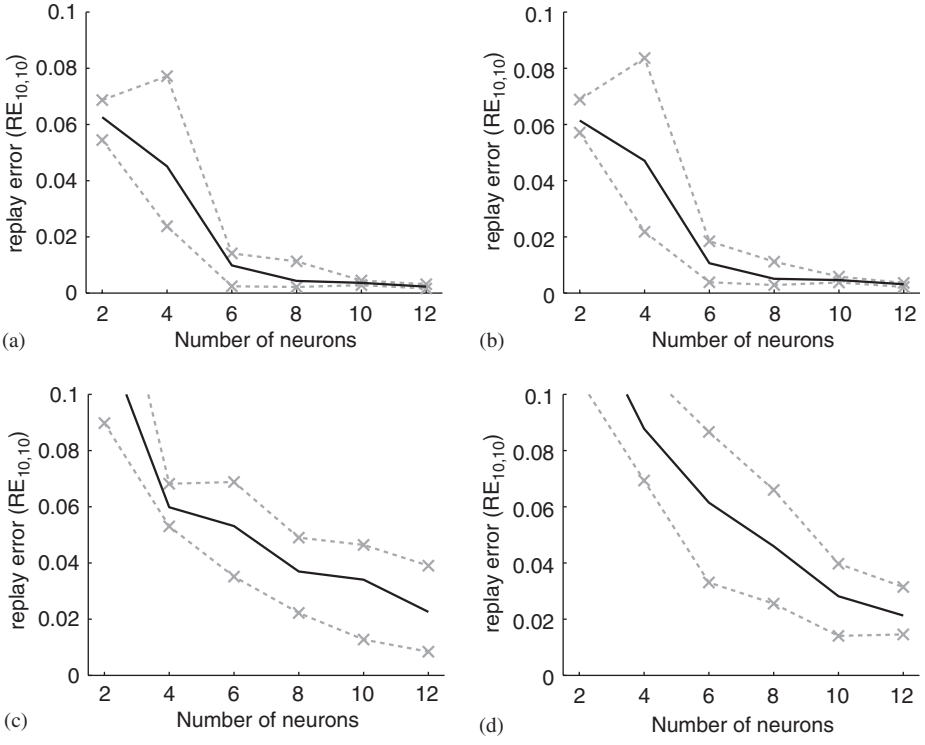

Fig. 4. Replay error vs. number of neurons. (a) MG17: replay error on training data vs. number of neurons; (b) MG17: prediction error on previously unseen data vs. number of neurons; (c) MG30: replay error on training data vs. number of neurons; (d) MG30: prediction error on previously unseen data vs. number of neurons. The average replay error $RE_{10,10}$ of eight-neuron networks, after 200 epochs of training. Black line: average of five test runs; gray dashed lines: best and worst runs.

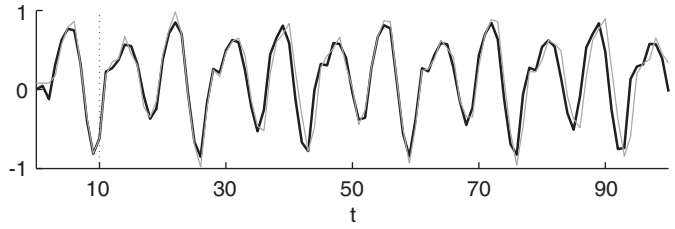

Fig. 5. Predictions of a trained RNN. One of the trained networks with eight neurons and $K = 5$ was trained on 10 time step inputs (placed before the vertical dashed line). Prediction was made without further observations. Thin gray line: the MG17 time series; thick black line: predictions of the RNN trained by PIRANHA.

as it can be seen in Fig. 4(d), the generalization capability to unseen portions of the time series remains similar to the case we experienced on MG17.

5.1.3. Noise

In another experiment the robustness of PIRANHA against noise was tested. The MG17 time series was perturbed by fixed-variance Gaussian noise, and we measured the replay error $RE_{10,10}$ on the original, noiseless time series. The number of neurons was fixed to $n = 8$, the lookahead parameter to $K = 5$, the other parameters were left unchanged, and the noise variance was gradually increased from 0 to 0.3 by steps of 0.01. The results are summarized in Fig. 6. As it can be seen, the algorithm is efficient in filtering the noise.

5.1.4. Multi-step prediction

Although PIRANHA was designed to learn sequence replay, it can also be used for multi-step-ahead prediction (MSP), as the two tasks are similar: the former requires a low total error on the first N_2 predictions, while the latter requires a low error only on the N_2th predictions. A quite commonly used benchmark for multi-step predictors is the normalized root mean square error of the $(t + 14)$th prediction on the MG17 time series. Running PIRANHA with $n = 8$, $k = 5$ and $\alpha_i = 1/15i$ on a 200-element part of the time series, we measured an NRMSE of 0.0832. This places PIRANHA among the best known predictors for this problem (a more detailed comparison to other methods is given in the Related work section).

5.2. Chaotic time series: sunspot numbers

In the next experiment we trained RNNs to predict sunspot numbers. Wolf's annual sunspot numbers are recorded since 1770, and constitute one of the most often used benchmark dataset for time series processing algorithms.

We trained 30-neuron networks using PIRANHA with $K = 5$ and $\gamma = 0.95$. The networks were trained on data from 1770 to 1979, and tested both on the interval 1980–1988 and 1980–2002. The mean squared error of the predictions were compared to several other methods. The results are summarized in Table 2.

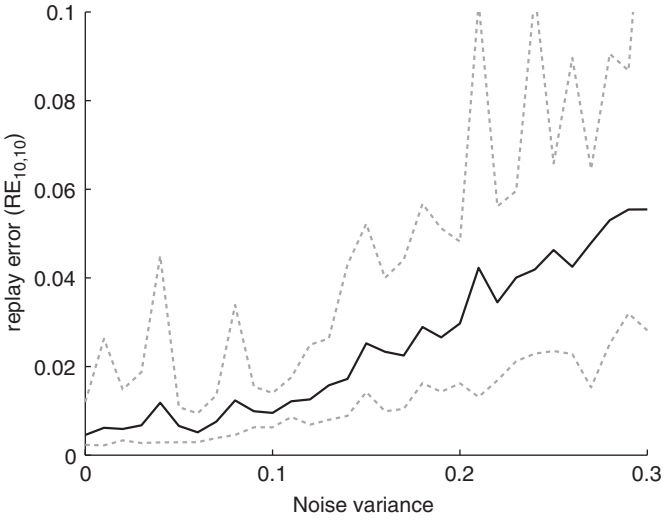

Fig. 6. The effect of noise. Replay error ($RE_{10,10}$) on the noiseless MG17 time series. Training was executed 20 times on noisy MG17 series with Gaussian noise, and with $n = 8$, $K = 5$, $T = 200$. Black line: average of 20 test runs; gray dashed lines: best and worst runs.

Table 2
MSE of various algorithms on Wolf's annual sunspot numbers. All results (except for PIRANHA) are taken from Small and Tse [25]

Method	MSE	
	1980–1988	1980–2002
AR(9) [28]	334	416
SETAR [14]	413	1728
Reduced AR [28]	214	291
Radial basis [18]	306	489
MDL neural network [25]	625	356
PIRANHA	211	298

5.3. Chaotic time series: the Lorenz attractor

The Lorenz system is governed by the simultaneous differential equations:

$$\dot{x} = -\sigma x + \sigma y, \tag{20}$$

$$\dot{y} = -xz + rx - y, \tag{21}$$

$$\dot{z} = xy - bz, \tag{22}$$

which is the earliest known chaotic system, and as such, it is often used to demonstrate the capabilities of time series processing algorithms, see e.g. [8,9,20,30,17].

Unfortunately, probably due to the widespread usage of the Lorenz system, there is no standard parameter set in the literature: most researchers agree on using $\sigma = 10$, $r = 28$ and $b = \frac{8}{3}$, but there is no consensus on other important characteristics, like the sampling interval, the prediction

horizon, the error measure, or the number of predicted state variables (most approaches either predict x or x, y pairs). This makes comparison difficult, so in this subsection we resort to demonstrating the performance of PIRANHA on the Lorenz dataset without comparison to other algorithms.

We used a sampling interval of $\tau = 0.1$ for predicting the x coordinate of the system. 2000 data points were used for training the networks, and the next 2000 points were used for computing the normed root mean squared errors (NRMSE) of k-step predictions for $k = 1, 2, \ldots$.

The networks consisted of 20 neurons, and we used PIRANHA parameters $K = 7$ and $\gamma = 0.95$. We trained five randomly initialized networks for 200 epochs. The mean prediction errors with respect to the prediction horizon are shown in Fig. 7.

Fig. 8 shows a typical prediction sequence. As it can be seen, the network learned the characteristic dynamics considerably well: it produces Lorenz-like output in spite of the large differences between the predicted and the real signals. The dynamics of the attractor is thus well represented by the RNN.

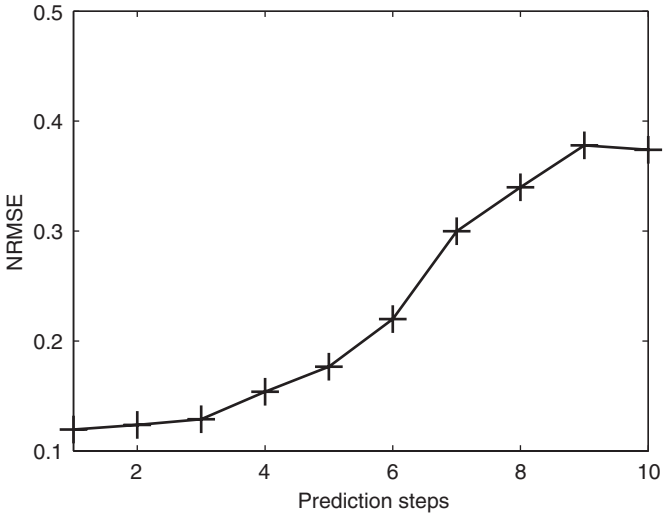

Fig. 7. Multi-step prediction errors on the Lorenz series. The average NRMSE of 20-neuron networks, after 200 epochs of training. Black line: average of five test runs.

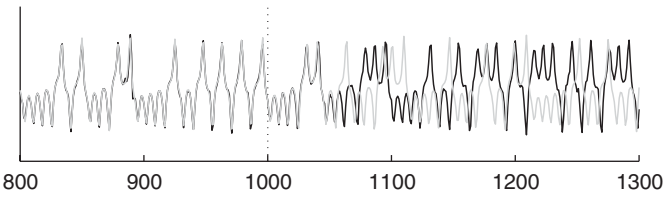

Fig. 8. A typical prediction of the Lorenz series. A typical output of a PIRANHA-trained 20 neuron network ($K = 7$). Free run starts after the dotted line. Gray line: the original Lorenz time series; black line: predictions of the RNN trained by PIRANHA.

5.4. Text memorization

Capabilities of PIRANHA were tested on a problem of different nature. Experiments were performed on memorizing text data: networks were trained on a given text, then small portions of the text were presented to the network, which had to recognize it and continue from then on. The alphabet was fixed to the 26 lowercase letters plus a whitespace and a full stop character, and each symbol was encoded as a five-element vector with ± 1 entries. The five-element output of the networks were decoded by taking the sign of the output vector and then taking the symbol corresponding to this $\{-1, +1\}^5$ vector (or '*', if no such symbol exists).

We used $c_1 = c_2 = 0.01$ and $\gamma = 0.9$ here as well. $K = 5$ and $\alpha = 0.01$ were used. The texts we selected were three short English tongue-twisters:

“ `we surely shall see the sun shine soon` ” ,

“ `a big bug bit the little beetle` ” ,

“ `sam's shop stocks short spotted socks` ” .

Note that the characteristics of these tongue-twisters is that they contain multiple similar substrings like “big”, “bug” and “bit”, and also relatively long identical substrings (like the five-character portion “s sho” in the last tongue-twister), for which the sequel is ambiguous without a memory of the previous characters. For the sake of comparison, we have also made test runs on a 40-character long randomly generated string.

Each trained network was evaluated by using the following method: the network was allowed to observe the first N_1 characters of the text, then had to complete the remainder. The number of mistakes (wrong symbol predicted) was added for all observation lengths in the interval $5 \leqslant N_1 \leqslant T - 1$, and divided by the total number of predictions, thus giving the ratio of total mistakes. This evaluation method is somewhat arbitrary, but takes into account every position in the text as a starting position for prediction with equal weights. In turn, it is a suitable qualitative measure to gauge the replay performance of networks.

We varied the number of neurons from 10 to 28 by steps of two, training each one for 100 steps. After training we evaluated them by the method described above to get the ratio of mistakes for each of them, as shown in Fig. 9. Small networks were unable to learn the strings, predicting only up to 30% of the characters correctly. It seems that these networks did not have enough resources to resolve the ambiguous situations. Surprisingly, the decrease of error ratio is *quite sudden* for each problem, and networks with more neurons than some threshold value are able to reproduce the text from any starting point. This indicates that the character sequences are stored in the network in a “holistic” way, i.e. either the whole sequence can be stored or none of it.

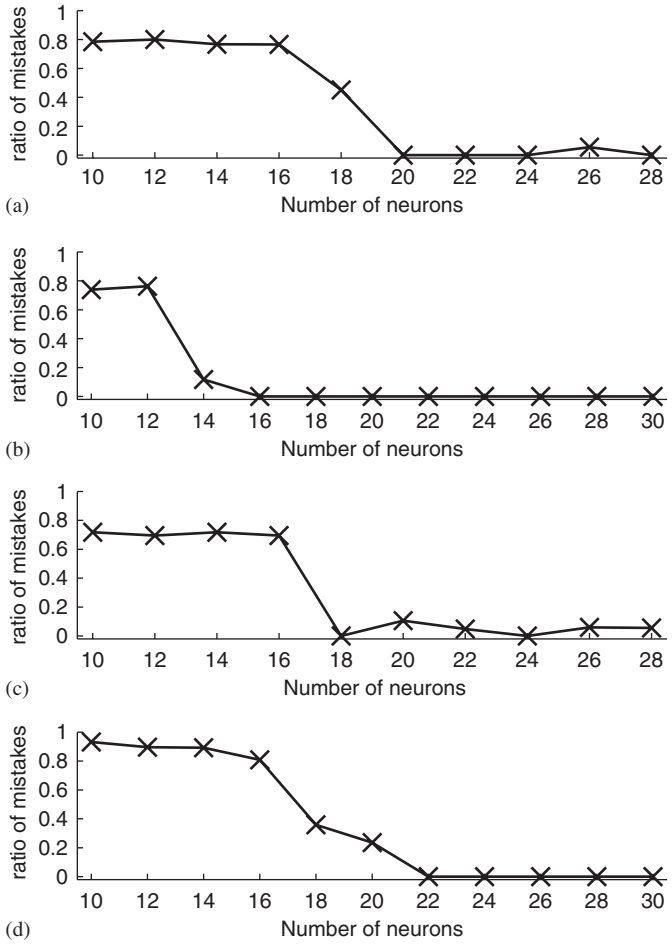

Fig. 9. Ratio of mistakes for various tongue-twister strings. (a) We surely shall see the sun shine soon; (b) a big bug bit the little beetle; (c) sam's shop stocks short spotted socks; (d) 40-character long random string. See text for details.

6. Discussion

6.1. Related work

There exists a large variety of algorithms for training RNNs. Recurrent backpropagation and backpropagation through time (BPTT) [31,24,23], real-time recurrent learning (RTRL) [33,15], or extended Kalman-filter (EKF) based training [32] are such examples. (For a review on these algorithms and for a unifying framework, see [21]). However, they are not particularly well-suited for the sequence replay problem, because they all concentrate on predicting a single output value rather than a whole sequence of them.

The amount of publications on learning, representing and retrieving complete data sequences in neural systems is surprisingly small. One attempt was made by Barreto and Araújo [3,2]. They train a neural network to replay sequences exactly as they appeared originally. However, in their approach temporal ordering is essentially hand-wired, and the utility of local context neurons to resolve ambiguities also seems limited. Another approach is the fractal prediction machine of Tiňo and Dorffner [27], which is able to learn the long-term characteristics of a time series. Their solution is based on iterated function systems, which is in close relationship with neural networks, but the connectionist algorithm remains to be developed.

6.1.1. Multi-step prediction

An approach similar to sequence learning is MSP. In fact, RNNs trained by PIRANHA can be used for MSP as well. An excellent overview on state-of-the art MSP methods can be found in [13]. To compare PIRANHA to them, we made test runs on the commonly used benchmark problem: predict the (t_0+14)th element of the Mackey–Glass-17 series and measure the normalized root mean square error (NRMSE). The best results to date were produced by *dynamic cell structure* [10] and *self-organizing map* algorithms [29] (NRMSE $= 0.022$ and 0.03, respectively). There are numerous RNN training methods that have been tested on this problem, e.g. the LSTM of Gers and Schmidhuber [13] or the genetic algorithm approach of De Falco et al. [12] (NRMSE $= 0.267$ and 0.291). Applying PIRANHA on a eight-neuron RNN with $K = 5$ and a training set of size 200, with 200 iterations produces an error of 0.0832. We consider this remarkable because the training set was tiny compared to that of used by the other methods (3000–20,000).

6.1.2. Relationship to BPTT

Although the motivation and derivation of BPTT and PIRANHA are completely different, the learning formulae of the two algorithms may look similar. However, there is an important difference between these methods. When long-term predictions are considered, PIRANHA uses uncorrected prediction (scheme 1(c)), while BPTT uses the correct input values at each step (scheme 1(a)), thus losing the chance of ever correcting false long-term trends. This is demonstrated clearly by experiments: according to [5], on the MG17 dataset (and several others) BPTT produces fairly good 1-step predictions, but completely fails on longer range (on the 14-step-ahead prediction problem BPTT produced a NRMSE of 0.58, which could be improved by weight pruning down to 0.44. In contrast, using PIRANHA, we obtained an NRMSE of 0.0832.

A method using uncorrected prediction errors similar to PIRANHA can also be found in the online supporting material of [17] as an ad hoc improvement method, and is applied only for several steps. The policy iteration idea provides a theoretically sound framework for the scheme of Fig. 1(c).

6.1.3. Improving echo state networks

The big question of RNN training is how to train them both quickly and accurately. Echo state networks (ESN) were recently proposed by Jaeger and Haas [17], as extremely fast and accurately learning recurrent networks. ESNs train only the output weights, while keeping the randomly chosen recurrent weights fixed. As a consequence,

they are unbeatable in speed, but their accuracy can be improved using PIRANHA, by augmenting the 1-step prediction ESN training with the multi-step prediction training of PIRANHA, as it is shown by our experiments.

Moderate sized networks were studied. Good performance was reached quickly by Jaeger's training method, which was then slowly improved by adjusting the recurrent weights using PIRANHA. A set of tests are summarized in Fig. 10. The performance improvement was about 30% for the size range studied (10–60 neurons).

6.1.4. The role of reinforcement learning

The final form of the PIRANHA algorithm is a gradient descent on a cost function, which is a weighted sum of multi-step prediction errors. As such, it can be derived without any reference to RL. However, RL still has an important role, as it motivates the choice of this particular cost function and provides insight into the working of the algorithm. We believe that similar applications of RL can be fruitful in other tasks neural networks as well.

6.2. The route suggested by PIRANHA

In many RL problems, the state space is prohibitively large or continuous, so storing all the state values is infeasible. In most cases this problem is resolved by applying some kind of function approximation (FAPP). It is well known that neural networks have excellent function approximation capabilities, so it is no wonder that they have been widely applied in various RL problems. These neural networks are typically feed-forward networks, whose choice is satisfactory for estimating the value function of Markovian problems, as there is no need to keep past memories. Traditional RL approaches with or without FAPPs have the following peculiar features: they neglect the problem of the recognition of states; they make use of a list of the states and optimize the policy over this list; or alternatively, they apply a function approximator to compress this enormous list. Thus, traditional RL characterizes states by an index or as a set of features *occurring at a given instant*.

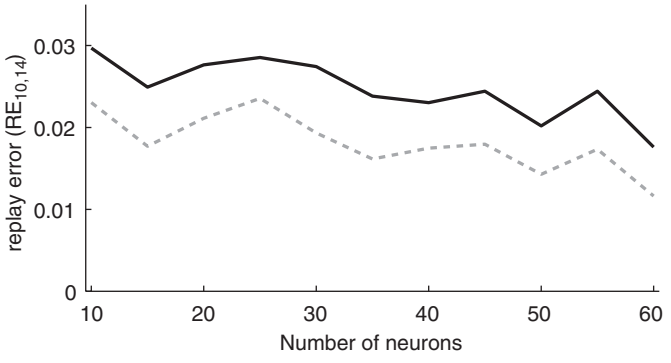

Fig. 10. Applying PIRANHA after ESN training. Replay error on the MG17 dataset. Solid black: ESN; dashed gray: PIRANHA on ESN. All points are the average of five runs. Parameters of PIRANHA are as above. ESN parameters are as in [17].

In contrast, we see RNNs as devices, which are input filters and can identify *spatio-temporal* regions of the state space. The unification of RNN learning and the RL framework is then important because of the following reasons:

(1) RNNs can identify deterministic spatio-temporal regions as the states for RL. That is, recurrent networks provide a natural and adaptive discretization over time, a not-yet-resolved problem of RL.
(2) RNNs have the reconstruction capability and can signal whether a given state is 'over' or not.
(3) RNNs have noise reduction capability and thus can significantly lower the number of RNN states, which need to be listed, or learned.
(4) Moreover, RNNs learn the dynamics of the environment and can represent the spatio-temporal process over a longer time scale than the experienced time region. In turn, RNNs are ideal for establishing the temporal memory of the dynamics and may provide enormous help in avoiding the combinatorial explosion related to the representation of temporal depth as required for MDPs.
(5) In our approach, the learning process of RNNs is naturally weighted by the behavioral relevance, i.e. the long-term cumulated reward of the RL process. We believe that *seamless integration* of these learning techniques is possible, because time scale can be discounted at the same rate at both levels, i.e. at the RNN level, which defines the states by partitioning time and at the decision making level that optimizes actions belonging to these states.

This means that RNNs are promising tools for representing the 'states' of non-Markovian, partially observable dynamical systems within the RL framework: the internal states of a network have much in common with 'belief states'.

Moreover, RNNs with PIRANHA could form a hierarchical RL scheme in a natural fashion: the raw input corresponds to a state of the lower. It can be continuous and non-Markovian. RL at the lower-level uses the reconstruction/prediction error as reward signal and is able to identify spatio-temporal regions of the input space, providing indexed high-level state description in the form of a set of RNNs. Thus, all high-level states have reconstruction ability through the RNNs belonging to the states. These indexed higher level states are used by a traditional RL agent at the upper level. Value prediction errors at the higher level can indicate that the lower level RNN representation needs to be improved. These, or similar ideas could be used in simulations. However, proof of convergence is missing and may be demanding. More work is needed to explore such extensions of RL.

6.3. Summary

In this article, a solution to the sequence replay problem was proposed. The question was to represent time series

data so that it can be reproduced sequentially from seeing only a small portion of it. We formalized the task as the minimization of the cost function of a recurrent neural network. An analogy to reinforcement learning enabled us to adapt policy iteration method to our problem, yielding a novel algorithm that we called PIRANHA. We showed that PIRANHA is convergent under appropriate conditions. PIRANHA was successfully applied for learning chaotic one-dimensional time series and text data. Also, PIRANHA could improve the performance of echo state networks considerably. It was found that PIRANHA is advantageous also for multi-step predictions.

Appendix A. Near-equivalence of the two cost functions

Let us first define the "per-step normalized" version of the two cost functions:

$$J_1(\mathbf{U}, F, G) := \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \sum_{t=0}^{T} \gamma^k \| H(s^{(k)}_{t,(F,G)}(u_t)) - x_{t+k} \|^2 \tag{A.1}$$

and

$$J_2(F, G) := \sum_{t=0}^{T} \| Hu_t - x_t \|^2 \tag{A.2}$$

$$\text{s.t.} u_{t+1} = \sigma(Fu_t + Gv_t), \tag{A.3}$$

$$\hat{x}_{t+1} = Hu_{t+1}. \tag{A.4}$$

Proposition 1. *For* $\mathbf{U} = \mathbf{U}_{(F,G)}$, *the two cost functions are almost the same, more exactly*,

$$|J_1(\mathbf{U}_{(F,G)}, F, G) - J_2(F, G)| = O\left(\frac{1}{T}\right). \tag{A.5}$$

Proof.

$$J_1(\mathbf{U}_{(F,G)}, F, G)$$

$$= \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{T} \| H(s^{(k)}_{t,(F,G)}(u_t)) - x_{t+k} \|^2 \tag{A.6}$$

$$= \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{T} \| H(u_{t+k}) - x_{t+k} \|^2 \tag{A.7}$$

$$= \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \sum_{t'=k}^{T+k} \| Hu_{t'} - x_{t'} \|^2 \tag{A.8}$$

$$= \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \sum_{t'=0}^{T} \| Hu_{t'} - x_{t'} \|^2 + \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \left(\sum_{t'=0}^{k-1} \| Hu_{t'} - x_{t'} \|^2 - \sum_{t'=T+1}^{T+k} \| Hu_{t'} - x_{t'} \|^2 \right) \tag{A.9}$$

$$= J_2(F, G) + \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \left(\sum_{t'=0}^{k-1} \| Hu_{t'} - x_{t'} \|^2 - \sum_{t'=T+1}^{T+k} \| Hu_{t'} - x_{t'} \|^2 \right).$$

However, elements of u_t and x_t fall into the interval $[-1, 1]$, and H is a projection matrix, so

$$\| Hu_{t'} - x_{t'} \|^2 \leqslant 4m, \tag{A.10}$$

where m is the dimension of the input vector. Therefore

$$\left| \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k \left(\sum_{t'=0}^{k-1} \| Hu_{t'} - x_{t'} \|^2 - \sum_{t'=T+1}^{T+k} \| Hu_{t'} - x_{t'} \|^2 \right) \right| \tag{A.11}$$

$$\leqslant \frac{1-\gamma}{T} \sum_{k=0}^{\infty} \gamma^k (k \cdot 4m + k \cdot 4m) \tag{A.12}$$

$$= \frac{1}{T}(1-\gamma) 8m \left(\sum_{k=0}^{\infty} k \cdot \gamma^k \right) = O\left(\frac{1}{T}\right), \tag{A.13}$$

proving our statement. □

Remark 2. Note that although the cost function (A.1) has significantly more parameters, PIRANHA will optimize it subject to the constraint $\mathbf{U} = \mathbf{U}_{(F,G)}$. According to Proposition 1, in this case the two cost functions are nearly equivalent.

Appendix B. Interpreting PIRANHA as policy iteration

B.1. Markov decision processes and policy iteration

Let us denote the state space and the actions of the agent by $\mathcal{S}$ and $\mathcal{A}$, respectively. The dynamics of the environment is characterized by the transition probability function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and the immediate cost function $c : \mathcal{S} \to \mathbb{R}$. A deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ of the agent is a mapping from states to actions. Future costs to be cumulated may be discounted by some factor $0 \leqslant \gamma \leqslant 1$, so the expected cost of a state s_0 is

$$J^{\pi}(s_0) = E(c(s_0) + \gamma c(s_1) + \gamma^2 c(s_2) + \cdots), \tag{B.1}$$

where $E(.)$ denotes expectation, and for each $t \geqslant 0$, $a_t = \pi(s_t)$ and the system arrives at s_{t+1} with probability $P(s_t, a_t, s_{t+1})$. The task is to find an optimal policy π^*, for which $J^{\pi^*}(s) \leqslant J^{\pi}(s)$ for any state s and any policy π.

Policy iteration approaches the optimal policy by a two-phase iteration procedure: at each iteration i, the current policy π_i is first *evaluated*, i.e. $J^{\pi_i}(s)$ is computed (or approximated), usually by letting the agent to take many steps using π_i and processing the experienced costs. The other phase is called *policy improvement*. In this phase the policy is modified by using the inequality

$$J^{\pi_i}(s) = c(s) + \gamma \sum_{s'} P(s, \pi(s), s') J^{\pi_i}(s') \tag{B.2}$$

$$\geqslant \min_a \left(c(s) + \gamma \sum_{s'} P(s, a, s') J^{\pi_i}(s') \right), \tag{B.3}$$

which implies that

$$\pi_{i+1}(s) := \arg\min_a \left(c(s) + \gamma \sum_{s'} P(s,a,s') J^{\pi_i}(s') \right) \tag{B.4}$$

is an improvement over π_i. The improvement can be gradual: there is no need to find the minimum, but it is sufficient if a partial and/or approximate step is made in that direction.

If transition probabilities, policies and cost function values are stored in a lookup-table, with separate entries for each different argument, policy iteration converges to an optimal policy under appropriate conditions. This is true even if approximations are used in either step or P and c is not known in advance but have to be estimated from the agent–environment interaction. For details about the conditions of the various convergence results, see [4,26].

In some cases, the construction of the lookup-table is not feasible, e.g. when the state and/or the action spaces are continuous. Then one needs to revert to function approximation methods. It has been shown, however, that policy iteration for function approximators can be divergent even for the simplest cases [4,7] and, in turn, the proof of convergence becomes a central issue.

B.2. Fitting PIRANHA into the RL framework

B.2.1. The agent and its environment

Let us define a hypothetical agent that observes the state sequence $\mathbf{U}$, and based on that, chooses a set of weights (F, G), which is used for prediction. Formally this corresponds to $\mathscr{S} = \mathbb{R}^{n\times T}$, so that a state $\mathbf{U} \in \mathscr{S}$ of the agent is the whole sequence of hidden states of the RNN: $\mathbf{U} = (u_1, u_2, \ldots, u_T)$, while the action space will be $\mathscr{A} := \mathbb{R}^{n\cdot n + n\cdot(m+1)}$. The environment of this hypothetical agent is quite simple: if the agent makes an action (F, G) in state $\mathbf{U}$, then the environment transfers it deterministically to the successor state $S_{(F,G)}\mathbf{U}$.

In a general-case RL problem, a policy of the agent is a mapping $\pi : \mathscr{S} \to \mathscr{A}$. However, in our problem, time independent weight matrices (F and G) are searched for, therefore we restrict the mapping to constant policies. Such policies $\pi_{(F,G)}$ execute the fixed action (F, G) regardless of the current state. For a policy $\pi \equiv (F, G)$ we will also use the notation $S_\pi(.)$ instead of $S_{(F,G)}(.)$.

B.2.2. Costs

Let $\mathbf{U} = (u_1, u_2, \ldots, u_T)$ be an arbitrary state. Its immediate cost $c(\mathbf{U})$ is defined as the error of reconstructing the input series x_t:

$$c(\mathbf{U}) := \sum_{t=1}^{T} \|Hu_t - x_t\|^2. \tag{B.5}$$

Thus, by (B.1), the total discounted cost of $\mathbf{U}$ using policy $\pi \equiv (F, G)$ and discount factor $0 \leqslant \gamma < 1$ is

$$J^\pi(\mathbf{U}) := c(\mathbf{U}) + \gamma c(S_\pi \mathbf{U}) + \gamma^2 c(S_\pi^2 \mathbf{U}) + \cdots \tag{B.6}$$

$$= \sum_{k=0}^{\infty} \sum_{t=1}^{T} \gamma^k \|H(s^{(k)}_{t,(F,G)}(u_t)) - x_{t+k}\|^2, \tag{B.7}$$

which is the same cost function as (9). This cost function reverts to the cost function of Eq. (3) (=Eq. (B.5)) for $\gamma = 0$, but then policy iteration reduces to the iterative 1-step method.

B.2.3. Policy evaluation

At iteration $i > 0$, we have a policy $\pi_i = \pi_{(F_i,G_i)}$ to be evaluated. The evaluation starts by taking many steps using π_i, which (by (11)) guarantees that the agent is in state $\mathbf{U}_{(F_i,G_i)}$. Therefore, we have to evaluate the cost function in a single state $\mathbf{U}_{(F_i,G_i)}$ only, which—in contrast to traditional policy iteration—can be done directly.

B.2.4. Policy improvement

Policy improvement will be accomplished only for the single state $\mathbf{U}_{(F_i,G_i)}$, because this is the only state where the full cost is known. As a further deviation from the basic policy improvement strategy (B.4), we take only a small step towards

$$\arg\min_{(F,G)} (c(\mathbf{U}_{(F_i,G_i)}) + \gamma \hat{J}^{\pi_i}(S_{(F,G)}\mathbf{U}_{(F_i,G_i)})) \tag{B.8}$$

$$:= \arg\min_{(F,G)} \hat{J}'(F, G), \tag{B.9}$$

and improve the policy gradually:

$$\pi_{i+1}(\mathbf{U}_{(F_i,G_i)}) = (F_{i+1}, G_{i+1}) := (F_i, G_i) - \alpha_i \nabla \hat{J}'(F_i, G_i), \tag{B.10}$$

where α_i is the learning rate for iteration i, and ∇ denotes the gradient with respect to components of (F, G).

Iterating these two steps completes policy iteration, and as it can be easily seen, it is identical to the PIRANHA algorithm described in Fig. 1.

B.2.5. Differences from standard policy iteration

There are several important differences between standard policy iteration algorithm and PIRANHA. Firstly, in the policy improvement step we modified the policy so that only the cost of a *single state*, $\mathbf{U}_i$ is considered. However, changing the policy in $\mathbf{U}_i$ changes the costs of every other state as well, and we have no guarantee that the change is really an improvement for all states (in fact, it can be shown that there will be states for which the cost increases). Thus, we cannot apply any of the convergence results of (approximate) policy iteration algorithms, because they all require improvement over the whole state space. Fortunately, $J^{\pi_i}(\mathbf{U}_i) \geqslant J^{\pi_{i+1}}(\mathbf{U}_{i+1})$ still holds, but we need to prove this by taking into account the specific structure of the cost function.

B.3. The finite-K approximation

The gradients of (12) are infinite sums, which are approximated by finite sums up to the Kth term. The first question we have to answer is whether this approximation is feasible. The following lemma shows that for sufficiently small discount factor γ, gradients (13) and (14) are

convergent, therefore for sufficiently large K, the gradients of (12) will be approximated well by (16).

Lemma 3. *Suppose that* (a) *For all* i, $\|F_i\|_\infty \leqslant 1/\sqrt{\gamma}$, (b) *the slope of the sigmoid function is at most* 1, *i.e.* $\sup_z \sigma'(z) \leqslant 1$. *Then for any* $\varepsilon_0 > 0$ *there exists a* K *so that if* (16) *is used for calculating the gradient, the difference is less than* ε_0.

Proof. The difference of derivatives with respect to some component F_{ab} is

$$\left| \left(\frac{\partial J'}{\partial F_{ab}} \right)(F_i, G_i) - \left(\frac{\partial \hat{J}'}{\partial F_{ab}} \right)(F_i, G_i) \right| \tag{B.11}$$

$$= \left| \sum_{t=1}^{T} \sum_{k=K+1}^{\infty} \gamma^k (e_{t,k})^\top H [m(k)]_a [u_t]_b \right| \tag{B.12}$$

$$\leqslant \sum_{k=K+1}^{\infty} \sum_{t=1}^{T} \gamma^k \|e_{t,k}\|_\infty \|H\|_\infty \|m(k)\|_\infty \|u_t\|_\infty. \tag{B.13}$$

But u_t is an inner state of a neural network, so $\|u_t\|_\infty \leqslant 1$, and similarly, $\|e_{t,k}\|_\infty \leqslant 2$, and $\|H\|_\infty = 1$ because it is a projection matrix. Using (15) and Assumption (b), the norm of $m(k)$ can be bounded from above by

$$\|m(k)\|_\infty \leqslant \|F_i\|_\infty^k \leqslant \gamma^{-k/2}, \tag{B.14}$$

so the difference of the exact and approximate derivatives can be bounded as

$$\sum_{k=K+1}^{\infty} \sum_{t=1}^{T} \gamma^k \cdot 2 \cdot 1 \cdot \gamma^{-k/2} \cdot 1 = \sum_{k=K+1}^{\infty} 2T\gamma^{k/2} \tag{B.15}$$

$$= \frac{2T\gamma^{(K+1)/2}}{1-\gamma^{1/2}} := C_0 \gamma^{K/2}, \tag{B.16}$$

which can be made less than ε_0 for sufficiently large K. One may use (14) to show that the same bound applies for the differences of derivatives with respect to elements of G. □

B.3.1. Convergence proof

Proving the convergence of the algorithm described in Table 1 is easy, because it is a gradient descent method. Therefore, as it is well known, it converges to a (local) minimum if the step sizes α_i are sufficiently small. However, below we give an alternative proof, which gains its relevance by building the bridge to policy iteration. This formulation can serve as the basis of future generalization to connect RNNs to RL and reflects the basic idea of our algorithm better.

B.4. The proof of the convergence theorem

First, suppose that J and J' can be computed exactly. We proceed by a series of lemmas:

Let $\pi_i = (F_i, G_i)$ be the policy at iteration i, $\mathbf{U}_{\pi_i} = (u_0, u_1, \ldots, u_T)$, furthermore, let the gradient of J' be denoted by $\Delta\pi = (\Delta F, \Delta G)$ as above.

Lemma 4. *For* $\Delta\pi \not\equiv 0$ *there exists a number* $\alpha_0 > 0$ *such that for all* $0 < \alpha \leqslant \alpha_0$,

$$J^{\pi_i + \alpha\Delta\pi}(\mathbf{U}_{\pi_i}) < J^{\pi_i}(\mathbf{U}_{\pi_i}). \tag{B.17}$$

Proof. By definition, $\Delta\pi$ is the steepest descent direction of $c(\mathbf{U}_{\pi_i}) + J^{\pi_i}(S_\pi \mathbf{U}_{\pi_i})$, and it is nonzero, therefore for sufficiently small β_0, for all $0 < \beta \leqslant \beta_0$,

$$c(\mathbf{U}_{\pi_i}) + \gamma J^{\pi_i}(S_{\pi_i + \beta\Delta\pi} \mathbf{U}_{\pi_i}) \tag{B.18}$$

$$< c(\mathbf{U}_{\pi_i}) + \gamma J^{\pi_i}(S_{\pi_i} \mathbf{U}_{\pi_i}) = J^{\pi_i}(\mathbf{U}_{\pi_i}). \tag{B.19}$$

Let

$$\mathscr{U} := \{\mathbf{U} \in \mathbb{R}^{n \times (T+1)} \mid c(\mathbf{U}) + \gamma J^{\pi_i}(S_{\pi_i + \beta\Delta\pi}\mathbf{U}) < J^{\pi_i}(\mathbf{U}) \text{ for all } \beta \leqslant \beta_0\}$$

be the set of states for which $\pi_i + \beta_0 \Delta\pi$ is an improvement over π_i. This is an open set, because $c(\mathbf{U}) + \gamma J^{\pi_i}(S_{\pi_i + \beta_0 \Delta\pi}\mathbf{U}) - J^{\pi_i}(\mathbf{U})$ is a continuous function of $\mathbf{U}$. Furthermore, $\mathbf{U}_{\pi_i} \in \mathscr{U}$, so it also contains a ball with some positive radius r centered on $\mathbf{U}_{\pi_i}$.

Note that $\mathbf{U}_{\pi_i} = S_{\pi_i} \mathbf{U}_{\pi_i}$ and the operator $S_{\pi_i + \beta\Delta\pi}$ is continuous in β, so there exists a β_1 so that $\|\mathbf{U}_{\pi_i} - S_{\pi_i + \beta_1 \Delta\pi}\mathbf{U}_{\pi_i}\| \leqslant r$, which means that $S_{\pi_i + \beta_1\Delta\pi}\mathbf{U}_{\pi_i} \in \mathscr{U}$, and in general, for all k there exists a β_k so that $S^k_{\pi_i + \beta_k \Delta\pi}\mathbf{U}_{\pi_i} \in \mathscr{U}$. Let

$$\alpha_0 := \min(\beta_0, \beta_1, \ldots, \beta_{T-1}). \tag{B.20}$$

Then for all $0 \leqslant k < T$ and $0 < \alpha < \alpha_0$

$$c(S^k_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) + \gamma J^{\pi_i}(S^{k+1}_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) < J^{\pi_i}(S^k_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) \tag{B.21}$$

by the definition of $\mathscr{U}$. Adding these inequalities with weights γ^k gives

$$c(\mathbf{U}_{\pi_i}) + \gamma c(S_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) + \cdots + \gamma^{T-1} c(S^{T-1}_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) \tag{B.22}$$

$$+ \gamma^T J^{\pi_i}(S^T_{\pi_i + \alpha\Delta\pi}\mathbf{U}_{\pi_i}) < J^{\pi_i}(\mathbf{U}_{\pi_i}). \tag{B.23}$$

The LHS is exactly the cost of taking T steps using $\pi_i + \alpha\Delta\pi$, and then using π_i. Noting that this is the same as following $\pi_i + \alpha\Delta\pi$ all the way, so we get

$$J^{\pi_i + \alpha\Delta\pi}(\mathbf{U}_{\pi_i}) < J^{\pi_i}(\mathbf{U}_{\pi_i}), \tag{B.24}$$

which was to be shown. □

Lemma 5. *For* $\Delta\pi \not\equiv 0$ *there exists a number* $\alpha_1 > 0$ *such that for all* $0 < \alpha \leqslant \alpha_1$,

$$J^{\pi_i + \alpha\Delta\pi}(\mathbf{U}_{\pi_i + \alpha\Delta\pi}) < J^{\pi_i}(\mathbf{U}_{\pi_i}). \tag{B.25}$$

Proof. The proof proceeds similarly as the proof of Lemma 4: let

$$\mathscr{U}' := \{\mathbf{U} \in \mathbb{R}^{n \times (T+1)} \mid J^{\pi_i + \alpha\Delta\pi}(\mathbf{U}) < J^{\pi_i}(\mathbf{U}_{\pi_i}) \text{ for all } \alpha \leqslant \alpha_0\}.$$

This set is open as well because of the continuity of J, and by Lemma 4, it contains $\mathbf{U}_{\pi_i}$, and thus contains also a ball centered around $\mathbf{U}_{\pi_i}$. This means that for sufficiently small β_0, $S_{\pi_i + \beta\Delta\pi}\mathbf{U}_{\pi_i} \in \mathscr{U}'$ for all $0 < \beta < \beta_0$.

Setting $\alpha_1 := \min(\alpha_0, \beta_0)$ proves the statement of the lemma. □

These lemmas enable us to prove convergence for the exact cost function case:

Theorem 6. *If the learning rates α_i are determined by Lemma 5, PIRANHA converges to a policy $\bar{\pi}$ that is a (local) minimum of $J^{\pi}(\mathbf{U}_{\pi})$.*

Proof. As Lemma 5 states, $J^{\pi}(\mathbf{U}_{\pi})$ is monotonously decreasing, thus it is necessarily convergent, which also means that the policy series π_i converges to some $\bar{\pi} = \lim_{i\to\infty} \pi_i$. By taking the limit of $\pi_{i+1} = \pi_i + \alpha_i \Delta\pi$ we get by applying Lemma 5 to $\bar{\pi}$ that $\bar{\alpha}\Delta\bar{\pi} = 0$. This means that either $\Delta\bar{\pi} = 0$ or $\bar{\alpha} = 0$. However, if $\Delta\bar{\pi} \neq 0$, Lemmas 4 and 5 guarantee the existence of a positive learning rate α, so the gradient necessarily vanishes. □

Now suppose that we do not compute the exact J' for computing the gradient, but use the approximation

$$\bar{J}'(F, G) = c(\mathbf{U}_i) + \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma^k \|e_{t,k}\|^2 \tag{B.26}$$

instead with some fixed lookahead parameter K. In the following lemma we rephrase Lemma 3, and shows that we can get arbitrarily close to the minimum if K is sufficiently large and the norm of recurrent weights does not grow much beyond 1.

To this end, let us first define the norm of a policy π: recall that π is the concatenation of a $n \times n$ and an $n \times (m+1)$ real-valued matrix. If we regard policies as $n(n+m+1)$ dimensional vectors, then we can define the scalar product $\langle \pi_1, \pi_2 \rangle$ of two policies as normal scalar products in Euclidean space, and the norm of policy π as $\|\pi\| = \langle \pi, \pi \rangle^{1/2}$. The norm of matrices and vectors is defined as usual; $\|.\|_{\infty}$ denotes the maximum-norm.

Theorem 7. *Suppose that* (a) *for all i, $\|F_i\|_{\infty} \leqslant 1/\sqrt{\gamma}$,* (b) *the slope of the sigmoid function is at most* 1, *i.e.* $\sup_z \sigma'(z) \leqslant 1$. *Then for any $\varepsilon_0 > 0$ there exists a K so that if* (B.26) *is used for calculating the gradient $\hat{\pi}_i$, then* $\limsup_{i\to\infty} \|\Delta\hat{\pi}_i\| \leqslant \varepsilon_0$.

Proof. For the proof we will exploit the fact that gradient descent is convergent with approximating the direction of the steepest descent as long as actual direction of descent and the direction of steepest descent enclose an acute angle, i.e. as long as the scalar product is positive. In our case this means that we have to prove that at each iteration, $\langle \Delta\pi, \Delta\hat{\pi} \rangle > 0$. We begin by bounding $\|\Delta\pi - \Delta\hat{\pi}\|$.

Lemma 3 states that the difference of the exact and approximate derivatives can be bounded by

$$\left| \left(\frac{\partial J'}{\partial F_{ab}} \right)(F_i, G_i) - \left(\frac{\partial \hat{J}'}{\partial F_{ab}} \right)(F_i, G_i) \right| \tag{B.27}$$

$$\leqslant \frac{2T\gamma^{(K+1)/2}}{1-\gamma^{1/2}} \coloneqq C_0 \gamma^{K/2}, \tag{B.28}$$

where C_0 is a constant independent of K, and the same holds for the differences of derivatives with respect to elements of G.

Therefore the norm of the gradient error $\Delta\pi - \Delta\hat{\pi}$ can be bounded from above as

$$\|\Delta\pi - \Delta\hat{\pi}\| \leqslant \sqrt{n(n+m+1)} C_0 \gamma^{K/2} \coloneqq C_1 \gamma^{K/2}. \tag{B.29}$$

This can be made smaller than any fixed $\varepsilon_0 > 0$ if $K > 2\log(\varepsilon_0/C_1)/\log\gamma$.

Applying this result to the scalar product yields

$$\langle \Delta\pi, \Delta\hat{\pi} \rangle = \langle \Delta\pi, \Delta\pi \rangle + \langle \Delta\pi, \Delta\hat{\pi} - \Delta\pi \rangle \tag{B.30}$$

$$\geqslant \|\Delta\pi\|^2 - \|\Delta\pi\| \|\Delta\pi - \Delta\hat{\pi}\| \tag{B.31}$$

$$\geqslant \|\Delta\pi\| (\|\Delta\pi\| - \varepsilon_0), \tag{B.32}$$

which is greater than zero if $\|\Delta\pi\| > \varepsilon_0$, but this holds by the assumption of the theorem. □

Remark. Assumption (a) requires that the recurrent weights remain relatively small (the norm of F does not grow much beyond 1), thus avoiding chaotic behavior. Although this constraint cannot be verified a priori, it can be enforced by either choosing small γ values or by applying weight regularization technique.

References

[1] S. Amari, H. Park, K. Fukumizu, Adaptive method of realizing natural gradient learning for multilayer perceptrons, Neural Comput. 12 (6) (2000) 1399–1409.

[2] A. Araújo, G. Barreto, A self-organizing context-based approach to tracking of multiple robot trajectories, Int. J. Appl. Intell. 17 (1) (2002) 101–119.

[3] G. Barreto, A. Araújo, Storage and recall of complex temporal sequences through a contextually guided self-organizing neural network, in: Proceedings of the IEEE–INNS–ENNS International Joint Conference on Neural Networks, (IJCNN'00), vol. 3, Como, Italy, 2000, pp. 207–212.

[4] D. Bertsekas, J. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, 1996.

[5] R. Boné, M. Crucianu, An evaluation of constructive algorithms for recurrent networks on multi-step-ahead prediction, in: International Conference on Neural Information Processing, vol. 2, Singapore, 2002, pp. 547–551.

[6] R. Boné, M. Crucianu, Multi-step-ahead prediction with neural networks: a review, in: 9emes Rencontres Internationales: Approches Connexionnistes en Sciences, vol. 2, Boulogne sur Mer, France, 2002, pp. 97–106.

[7] J.A. Boyan, A.W. Moore, Generalization in reinforcement learning: safely approximating the value function, in: G. Tesauro, D.S. Touretzky, T.K. Leen (Eds.), Advances in Neural Information Processing Systems, vol. 7, MIT Press, Cambridge, MA, 1995, pp. 369–376.

[8] R. Castro, T. Sauer, Reconstructing chaotic dynamics through spike filters, Phys. Rev. E 59 (3) (1999) 2911.

[9] D. Chakrabarti, C. Faloutsos, F4: large-scale automated forecasting using fractals, in: Proceedings of International Conference on Information and Knowledge Management, 2002, pp. 2–9.

[10] L. Chudy, I. Farkas, Prediction of chaotic time-series using dynamic cell structures and local linear models, Neural Network World 8 (5) (1998) 481–489.

[11] M. Duhoux, J. Suykens, B.D. Moor, J. Vandewalle, Improved long-term temperature prediction by chaining of neural networks, Int. J. Neural Syst. 11 (1) (2001) 1–10.

[12] I.D. Falco, A. Iazzetta, P. Natale, E. Tarantino, Evolutionary neural networks for nonlinear dynamics modeling, Lecture Notes in Computer Science, vol. 1498, Springer, Berlin, 1998, pp. 593–602.
[13] F. Gers, D. Eck, J. Schmidhuber, Applying LSTM to time series predictable through time-window approaches, Lecture Notes in Computer Science, vol. 2130, Springer, Berlin, 2001, pp. 669–677.
[14] D.K. Ghaddar, H. Tong, Data transformation and self-exciting threshold autoregression, Appl. Stat. 30 (1981) 238–248.
[15] M. Gherrity, A learning algorithm for analog fully recurrent neural networks, in: IJCNN 89, vol. 1, San Diego, 1989, pp. 643–644.
[16] B. Horne, D. Hush, Bounds on the complexity of recurrent neural network implementations of finite state machines, in: J. Cowan, G. Tesauro, J. Alspector (Eds.), Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann Publishers, Los Altos, CA, 1994, pp. 359–366.
[17] H. Jaeger, H. Haas, Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication (with online supporting material), Science (2004) 78–80.
[18] K. Judd, A. Mees, On selecting models for nonlinear time series, Physica D 82 (1995) 426–444.
[19] M. Mackey, L. Glass, Oscillation and chaos in physiological control systems, Science 197 (1977) 287–289.
[20] J. McNames, Innovations in local modeling for time series prediction, Ph.D. Thesis, Department of Electric Engineering, Stanford University, 1999.
[21] B. Pearlmutter, Gradient calculations for dynamic recurrent neural networks: a survey, IEEE Trans. Neural Networks 6 (5) (1995) 1212–1228.
[22] M. Pedersen, Optimization of recurrent neural networks for time series modeling, Ph.D. Thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 1997.
[23] F. Pineda, Generalization of back-propagation to recurrent neural networks, Phys. Rev. Lett. 19 (59) (1987) 2229–2232.
[24] D. Rumelhart, G. Hinton, R. Williams, Learning internal representations by error propagation, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[25] M. Small, C.K. Tse, Minimum description length neural networks for time series prediction, Phys. Rev. E 66 (2002) 066706.
[26] R. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, 1998.
[27] P. Tino, G. Dorffner, Predicting the future of discrete sequences from fractal representations of the past, Mach. Learning 45 (2) (2001) 187–217.
[28] H. Tong, Non-linear Time Series: A Dynamical Systems Approach, Oxford University Press, New York, 1995.
[29] J. Vesanto, Using the SOM and local models in time-series prediction, in: Proceedings of Workshop on Self-Organizing Maps (WSOM'97), Espoo, Finland, 1997, pp. 209–214.
[30] E. Wan, Modeling nonlinear dynamics with neural networks: examples in time series prediction, in: The Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defense (WNN93/FNN93), San Francisco, CA, 1993, pp. 327–332.
[31] P. Werbos, Beyond regression: new tools for prediction and analysis in the behavioral sciences, Ph.D. Thesis, Harvard University, Cambridge, MA, 1974.
[32] R. Williams, Training recurrent networks using the extended Kalman filter, in: International Joint Conference on Neural Networks, vol. IV, Baltimore, 1992, pp. 241–250.
[33] R. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural Comput. 1 (1989) 270–280.

István Szita graduated in mathematics from the Eötvös Loránd University, Budapest in 2002. Currently he is working on his Ph.D. Dissertation at the Ph.D. School of Information Science of the same university. In 2003, he received the prestigeous Pro Scientia gold medal for his work as a young researcher from the Republic of Hungary. He is the first author or the coauthor of 10 peer reviewed communications. His research interest includes recurrent neural networks and reinforcement learning in complex domains.

András Lőrincz as a professor, senior researcher has been teaching at the Faculty of Informatics at Eötvös University, Budapest since 1998. His research focuses on distributed intelligent systems and their applications in neurobiological and cognitive modeling, as well as medicine. He has founded the Neural Information Processing Group of Eötvös University and he directs a multidisciplinary team of mathematicians, programmers, computer scientists and physicists. He has acted as the PI of several successful international projects in collaboration with Panasonic, Honda Future Technology Research and the Information Directorate of the US Air Force in the fields of hardware-software co-synthesis, image processing and human-computer collaboration. He graduated in Physics from Eötvös Loránd University in 1975 where he received his Ph.D. in 1978 and his C.Sc. in 1986 in Experimental and Theoretical Solid-State Physics and Chemical Physics, respectively. He conducted research and taught quantum control, photoacoustics and artificial intelligence at the Hungarian Academy of Sciences, University of Chicago, Brown University, Princeton University, the Illinois Institute of Technology and ETH Zurich. He authored about 200 peer reviewed scientific publications. During 1997–1998 he was the scientific director of the Hungarian subsidiary of US-based Associative Computing Ltd. He has received the Széchenyi Professor Award, Master Professor Award and the Széchenyi István Award in 2000, 2001, and 2004, respectively. Four of his students won the prestigious Pro Scientia Gold Medal in the field of information science over the last 4 years. In 2004, he was awarded the Kalmár Prize of the John von Neumann Computer Society of Hungary.