Biol Cybern (2015) 109:363–375
DOI 10.1007/s00422-015-0648-4

ORIGINAL PAPER

Biological Cybernetics

Ideomotor feedback control in a recurrent neural network

Mathieu Galtier

Received: 22 February 2014 / Accepted: 7 February 2015 / Published online: 10 March 2015

Abstract The architecture of a neural network controlling an unknown environment is presented. It is based on a randomly connected recurrent neural network from which both perception and action are simultaneously read and fed back. There are two concurrent learning rules implementing a sort of ideomotor control: (i) perception is learned along the principle that the network should predict reliably its incoming stimuli; (ii) action is learned along the principle that the prediction of the network should match a target time series. The coherent behavior of the neural network in its environment is a consequence of the interaction between the two principles. Numerical simulations show a promising performance of the approach, which can be turned into a local and better "biologically plausible" algorithm.

Keywords Recurrent neural network · Echo state network · Feedback control · Recursive least squares

1 Introduction

Animal life is characterized by the emergence of robust control mechanisms used in a large variety of environments. Beyond evolution which selects the life forms that make the most of their environment, it seems that some animals have the ability to quickly learn to interact constructively with new environments. Most probably, it means that the nervous systems of such animals can perform a sort of blind control of their environments: control is achieved without previous knowledge of the environment. It would surely be of great help to uncover such a mechanism, not only from a biological viewpoint, but also to replicate it for engineering tasks.

Control theory has been a vivid field of research for decades, which has provided many useful applications. However, although linear systems are well understood (Kwakernaak and Sivan 1972; Fortmann and Hitz 1977), nonlinear systems still require a significant effort to be dealt with (Slotine and Li 1991; Skogestad and Postlethwaite 2007). In particular, the control system is often assumed to have some explicit knowledge about the system to be controlled. A recent axis of research focuses on design of control schemes when there is a lot of uncertainties about the system to be controlled (Åström 2006). When nothing is known about the system to be controlled, the most widespread method is to use a proportional–integral–derivative controller (Åström and Hägglund 2006), although some sophisticated methods have been proposed, see (Zhong-Sheng 2006) for a review. Indeed, computing the best inverse model (Jordan 1996) from an unknown environment is a very difficult task, which has received no universal answer so far.

Neural networks are bio-inspired mathematical object, which have good learning capacities (Bishop 1995). A large number of control algorithms have used their adaptability to model some aspect of the environment and/or to design adaptive controllers. The challenge is to be able to internally predict the outcomes of a potential movement, so that the best motor commands can be chosen (Kawato et al. 1987; Ge et al. 2008; Yang et al. 2008). Typically, neural networks are used for two purposes: identification of the environment and the actual control, which can be computed once a good estimate of the environment has been designed (Narendra and Parthasarathy 1990; Ge et al. 2010).

M. Galtier (✉)
Minds, Jacobs University Bremen, Bremen, Germany
e-mail: mathieu.galtier@gmx.com

M. Galtier
NeuroMathComp, Inria Sophia, Sophia Antipolis, France

M. Galtier
UNIC, CNRS, Gif-Sur-Yvette, France

A main drawback of feedforward neural networks is their inability to take time dependencies into account. Although the usual work-around is to use tapped delay lines, a more natural approach would be to use recurrent neural networks (RNN) that are more suited to dynamical systems approximation. Based on the two steps of identification and control, various RNNs have been proposed as controllers, e.g., (Chow and Fang 1998; Wang and Hill 2006; Prokhorov 2007). However, RNN learning is notoriously known to be slow and to be subject to problematic bifurcations (Doya 1993; Pearlmutter 1995). To circumvent this problem, recent algorithms (Pan and Wang 2012; Waegeman et al. 2012) have been based on a reservoir computing architecture and more precisely echo state networks (ESN). They are recurrent networks where only the readout form a random reservoir of neurons is learned (Jaeger 2001). In these networks, the slow convergence and the bifurcation issues due to the tuning of the weights are bypassed. However, efficient optimization procedures for the hyper parameters and the overall mathematical understanding of ESNs are still lacking. Nonetheless, ESNs have proved to be very good at handling time dependencies and at predicting time series (Jaeger and Haas 2004). Therefore, they provide a solid basis upon which this paper will design a novel control architecture.

Most neural networks for control used so far have been based on two networks: one for the estimation of the state of the environment (which can be called perception), and another for the design of the control (which can be called action). In this paper, I will introduce a somewhat different architecture, since it will only be made of a single recurrent neural network from which two readouts are drawn and fed back. This is actually a fundamental feature of the approach, which resonates with the field of psychology called ideomotor theory (Greenwald 1970; Shin et al. 2010). This theory argues that perception and action are tightly linked and even represented in a single "domain." More precisely. a fundamental concept is that actions do not aim at changing the world directly, they rather aim at changing the perception of the world. Thus, action and perception are deeply entangled, and one could say that when the neural network "thinks" or predicts a future for its stimuli, then the corresponding action follows (Friston et al. 2010). In a way, the "active inference" perspective presented by Friston et al. (Friston et al. 2010) could be regarded as a modern theory compatible with ideomotor approach, where the underlying imperative for both action and perception is to minimize prediction errors (or variational free energy). Alternatively, the conceptual difference from traditional feedback loop algorithms can be seen in the structure of the controller in Fig. 1: at the level of the controller information flows in both directions, what is usually called the output of the controller is also fed back to the central network.

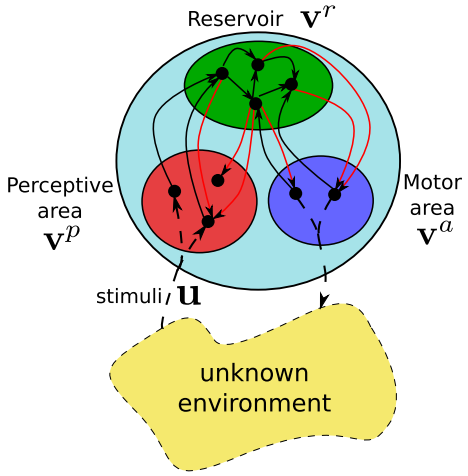

Fig. 1 General structure of the proposed controller interacting with its environment. Not all the connections are represented for clarity. Learning only modifies the connections in *red* (from reservoir to perceptive and motor area) (color figure online)

In this paper, I introduce an ideomotor recurrent neural network (IDRNN) together with a learning procedure in order to control an unknown environment. This paper intends to be a proof of concept that such a neural network can successfully learn to control fairly complicated dynamical systems. In Sect. 2, I introduce the network and the notations. Section 3 will be devoted to explaining the computational principles underlying ideomotor learning, and Sect. 4 describes the corresponding algorithm. Numerical experiments for various environments and a short comparison with the ESN-based method in (Waegeman et al. 2012) will be presented in Sect. 5. Finally, I will discuss the properties of such neural networks, in particular their biological plausibility, in Sect. 6.

2 Model

The model details the dynamics of a recurrent neural networks interacting with an unknown environment. The way the environment interacts with the agent (through sensors and actuators) is also formally unknown. It is the role of the agent to understand this interaction through statistical observations.

2.1 Notations and conventions

Several elements of the neural network and the environment are time-dependent multidimensional functions, which will be written with lowercase bold letters, e.g., $\mathbf{v}$. The value of these functions at time t will be written as an index to the function's notation, e.g., $\mathbf{v}_t$. Most of these functions will be described by recurrence equations, which are assumed to start from $t = 0$ (*for negative time, the functions return zero*). The matrices are written in upper case bold, e.g., $\mathbf{W}$.

We now detail the different parts of the neural network and the environment:

- *Environment*:
  The environment possibly has a complicated nonlinear and/or stochastic dynamics which we know nothing about. From the controller perspective, only some measures of the environment state are known. They are sometimes called observations but we refer to them as *stimuli* not to confuse them with the states of the reservoir which could also be called observations in the framework of online least mean square problems.
  The n_p neurons of the perceptive area receive information from the environment through the input vector $\mathbf{u}_t \in \mathbb{R}^{n_p}$ at time $t \in \mathbb{R}_+$ (where n_p is the dimension of the perceptive area).
- *Network activity*:
  We assume the controller is made of a neural network, which is decomposed in three parts: a *perceptive area*, a *reservoir*, and a *motor area* see, Fig. 1. The variables describing the activity of these subsets of the neural network are, respectively, $\mathbf{v}_t^\mathrm{p} \in \mathbb{R}^{n_\mathrm{p}}$ called *perception*, $\mathbf{v}_t^\mathrm{r} \in \mathbb{R}^{n_\mathrm{r}}$ *called reservoir states*, and $\mathbf{v}_t^\mathrm{a} \in \mathbb{R}^{n_\mathrm{a}}$ called *action*, where $n_p, n_\mathrm{r}, n_\mathrm{a} \in \mathbb{N}^*$ are the number of neurons in the respective area. Note that the letters p, r, a will always stand for perceptive area, reservoir, and motor area, respectively. We also define the variable gathering the entire network activity: $\mathbf{v} = \left(\mathbf{v}^\mathrm{p}\ \mathbf{v}^\mathrm{r}\ \mathbf{v}^\mathrm{a}\right)' \in \mathbb{R}^{n_p+n_\mathrm{r}+n_\mathrm{a}}$.
  The full connectivity matrix of the network is

$$\mathbf{W} = \begin{pmatrix} 0 & \mathbf{W}^\mathrm{pr} & 0 \\ \mathbf{W}^\mathrm{rp} & \mathbf{W}^\mathrm{rr} & \mathbf{W}^\mathrm{ra} \\ 0 & \mathbf{W}^\mathrm{ar} & 0 \end{pmatrix}$$

  where the second superscript (e.g., b in $\mathbf{W}^\mathrm{ab}$) is the origin of the connection and the first superscript (e.g., a in $\mathbf{W}^\mathrm{ab}$) is the destination, consistently with usual matrix notations. Note that the structure of $\mathbf{W}$ means that there are no connections between and within the perceptive and motor areas.
  We also define $\mathbf{W}^\mathrm{r} = (\mathbf{W}^\mathrm{rp}\ \mathbf{W}^\mathrm{rr}\ \mathbf{W}^\mathrm{ra}) \in \mathbb{R}^{n_\mathrm{r} \times (n_\mathrm{p}+n_\mathrm{r}+n_\mathrm{a})}$ corresponding to all the connections to the reservoir.
  A main idea in echo state networks (Jaeger 2001) from which this work is inspired is that the connections within and to the reservoir are randomly drawn. This means that the components of $\mathbf{W}^\mathrm{rr}$, $\mathbf{W}^\mathrm{rp}$, and $\mathbf{W}^\mathrm{ra}$ are i.i.d. constants along $\mathcal{N}(0, \sigma^2)$, $\mathcal{N}(0, \kappa^2)$, and $\mathcal{N}(0, \gamma^2)$. Albeit surprising, this choice leads to relevant prediction results and cheap algorithms.
  For simplicity, perception and action are considered to be linearly related to the reservoir activity. The reservoir follows a leaky integrator neural network equation (Jaeger et al. 2007), where a time constant τ controls the speed of the dynamics. Note that the contributions from perceptive and motor areas to the reservoir dynamics are linear (which will be important in the following). The perceptive area is stimulated by a weighted mean between the stimuli and the current prediction of the network. With the notations introduced before, this reads

$$\begin{cases} \mathbf{v}_t^\mathrm{r} = (1 - l\tau)\mathbf{v}_{t-1}^\mathrm{r} + \tau \mathbf{W}^\mathrm{r}. \begin{pmatrix} s(\mathbf{v}_{t-1}^\mathrm{r}) \\ \mathbf{v}_{t-1}^\mathrm{p} \\ \mathbf{v}_{t-1}^\mathrm{a} \end{pmatrix} \\ \mathbf{v}_t^\mathrm{p} = (1 - \alpha)\mathbf{W}^\mathrm{pr}.\mathbf{v}_t^\mathrm{r} + \alpha \mathbf{u}_t \\ \mathbf{v}_t^\mathrm{a} = \mathbf{W}^\mathrm{ar}.\mathbf{v}_t^\mathrm{r} \end{cases} \quad (1)$$

  where s is an element-wise sigmoidal function, $l \in \mathbb{R}_+$ is a decay constant, and $\alpha \in [0, 1]$ balances the contribution two terms: the stimuli $\mathbf{u}$ and the term $\mathbf{W}^\mathrm{pr} \cdot \mathbf{v}_t^\mathrm{r}$, which we call *prediction*. Note that if prediction and stimuli are identical, then the perception $\mathbf{v}^\mathrm{p}$ *becomes the same as the stimuli*.
- *Target trajectory*:
  The role of the neural network is to control the environment, so that it follows a *target* trajectory which we write $\mathbf{z}_t \in \mathbb{R}^q$, where $q \leq n_\mathrm{p}$. The target corresponds to (at least) one neuron in the perceptive area: if learning is successful, the stimuli to this subset of the perceptive neurons will be driven along the target trajectory. In the present formalism, the neural network can only control its stimuli (as opposed to an unobserved environment state).
- *Learning* corresponds to tuning the connections in the network. We assume that not all the connections are learnable: only the connections in red in Fig. 1 are learnable. In other words, and in a reservoir computing spirit, the connections to and within the reservoir $\mathbf{W}^\mathrm{r}$ are fixed. We call *perceptive learning* the tuning of the connectivity $\mathbf{W}^\mathrm{pr}$ and *motor learning* the tuning of the connectivity $\mathbf{W}^\mathrm{ar}$.

3 Principle of ideomotor learning

In this paper, ideomotor learning is defined by the combination of two principles: (i) perceptive learning aims at minimizing the distance between internal predictions $\mathbf{W}^\mathrm{pr} \cdot \mathbf{v}^\mathrm{r}$ and stimuli $\mathbf{u}$, and (ii) motor learning aims at minimizing the distance between internal predictions $\mathbf{W}^\mathrm{pr} \cdot \mathbf{v}^\mathrm{r}$ and target $\mathbf{z}$.

A control task is said to be successful when stimuli and target are equal. This can be a difficult task to achieve, in particular when the environment is unknown. The idea behind ideomotor learning is that having a good predictor of the stimuli may help designing an intelligent control. A controller which would be able to faithfully reproduce the stimuli, without needing to "see" them, would also know how to perturb the environment to reach a desired target. This idea is very similar to the good regulator hypothesis (every controller of its environment must possess a model of that environment)

that underpins early work in self-organization and cybernetics (Conant and Ashby 1970). Therefore, one needs to learn a model of the world [principle (i) above] and, at the same time, make sure this model is going in the desired direction [principle (ii) above]. More formally, ideomotor learning can be seen as adding a third variable, the prediction, in the distance between stimuli and target and using the triangular inequality to break the minimization of this distance into two sub-tasks.

The fact that motor learning makes no reference to the stimuli $\mathbf{u}$ has important functional consequences. Indeed, the actions exclusively aims at modifying the reservoir dynamics, so that the perception matches the target. The impact on the actions on the world and the stimuli $\mathbf{u}$ is simply a by-product of the method. In fact, the actions only want to control the internal model of the world that perceptive learning tries to build.

Mathematically, it is possible to formalize the two principles of ideomotor learning as the minimization of prediction errors by perception and action, respectively:

$$\begin{aligned} &\text{(i) perceptive learning} && \underset{\mathbf{W}^{\mathrm{pr}}}{\text{minimize}} && \sum_{s=0}^{t} \|\mathbf{u}_s - \mathbf{W}^{\mathrm{pr}} \cdot \mathbf{v}_s^{\mathrm{r}}\|^2 \\ &\text{(ii) motor learning} && \underset{\mathbf{W}^{\mathrm{ar}}}{\text{minimize}} && \sum_{s=0}^{t} \|\mathbf{z}_s - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_s^{\mathrm{r}}\|^2 \end{aligned} \tag{2}$$

where $(\mathbf{u}, \mathbf{v}^{\mathrm{r}})$ is a solution of system (1) defined for given $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$. The matrix $\mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \in \mathbb{R}^{q \times n_{\mathrm{r}}}$ is the restriction of the perceptive matrix $\mathbf{W}^{\mathrm{pr}}$ to the dimensions that are to be controlled along the trajectory $\mathbf{z}_t$, (i.e., to create $\mathbf{W}_{\mathrm{l}}^{\mathrm{p}}$, some rows of $\mathbf{W}^{\mathrm{pr}}$ have been removed). If learning is perfect, i.e., both sums in (2) are 0 for all t such that $(\mathbf{v}_t, \mathbf{u}_t)$ is on a limit cycle, then it is clear that the task is reached: $\mathbf{u}_t = \mathbf{z}_t$. We restrict our analysis to the case where such a limit cycle exists (which typically excludes non-controllable, non-observable environments).

Note that Eq. (2) effectively computes a path integral or time average of squared prediction error or free energy. As such, Eq. (2) expresses a "principle of least action," where action is the time average of energy.

Designing an online algorithm reaching this limit cycle reveals a fundamental problem: at time step t, the network has to figure out new matrices $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$ based on the potential impact they would have had in the past. But it cannot really know what this impact would have been since it would need to re-experience the past with these new matrices. This is impossible since the network has to update the matrices exclusively based on the available quantities. Another way to see this is to observe that the values of $\mathbf{u}_t$ and $\mathbf{v}_t^{\mathrm{r}}$ depend on $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$, and therefore, learning is not a simple least square problem, where observations $\mathbf{v}_t^{\mathrm{r}}$ and target $\mathbf{u}_t$ are independent weight vector. However, simple greedy algorithms can, in certain cases, converge to a perfect solution. A greedy algorithm roughly ignores the dependencies of $\mathbf{u}_t$ and $\mathbf{v}_t^{\mathrm{r}}$ on $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$, and corresponds to using a traditional least square approach for learning. This is the approach we take in the rest of the paper. This is formally similar to the mean field approximation that is used in variational formulations of active inference, in other words minimizing prediction errors under the assumption of conditional independence between the unknown states (and parameters).

4 Greedy RLS algorithm

In this paper, a greedy recursive least square (RLS) approach to solve the least square problems is considered. It corresponds to dynamically updating the connections based on a RLS algorithm performing the online minimization of the following problem:

$$\begin{aligned} \underset{\mathbf{W}^{\mathrm{pr}}}{\text{minimize}} \quad & H_t^p = \sum_{s=0}^{t} \lambda_{\mathrm{p}}^{t-s} \|\mathbf{u}_s - \mathbf{W}^{\mathrm{pr}} \cdot \mathbf{v}_s^{\mathrm{r}}\|^2 + \mu_{\mathrm{p}} \|\mathbf{W}^{\mathrm{pr}}\|^2 \\ \underset{\mathbf{W}^{\mathrm{ar}}}{\text{minimize}} \quad & H_t^a = \sum_{s=0}^{t} \lambda_{\mathrm{a}}^{t-s} \|(\mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{ra}})^{\dagger} \cdot (\mathbf{z}_s - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_s^{\mathrm{r}})\|^2 \\ & \quad + \mu_{\mathrm{a}} \|\mathbf{W}^{\mathrm{ar}}\|^2 \end{aligned} \tag{3}$$

where $\lambda_{\mathrm{p}}, \lambda_{\mathrm{a}} \in [0, 1]$ are forgetting factors, and $\mathbf{u}$ and $\mathbf{v}^{\mathrm{r}}$ are the solution of system (1) with time-dependent connection matrices $\mathbf{W}_t^{\mathrm{pr}}$ and $\mathbf{W}_t^{\mathrm{ar}}$. If λ is close to 0, then the sum simply involves very recent observations. The additional terms involving the squared norm of $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$ correspond to the usual Tikhonov regularization, and the numbers $\mu_{\mathrm{p}}, \mu_{\mathrm{a}} > 0$ control the amount of regularization. The notation $(\mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{ra}})^{\dagger}$ is the pseudoinverse of $\mathbf{W}_{\mathrm{l}}^{\mathrm{p}}.\mathbf{W}^{\mathrm{ra}}$.

The slight modification of the motor criterion in (3) due to the inversion of $\mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{ra}}$ simply corresponds to taking a different norm for the minimization. It is necessary to turn motor learning into a classical weighted least square problem. Indeed, one can unravel the dynamics of the reservoir for one time step to let the connection explicitly appear in the criterion. This corresponds to injecting the first row of (1) into $\mathbf{z}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_t^{\mathrm{r}}$, which leads to

$$\mathbf{z}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_t^{\mathrm{r}} = \tau \left(\mathbf{y}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{ra}} \cdot \mathbf{W}^{\mathrm{ar}} \cdot \mathbf{v}_{t-1}^{\mathrm{r}} \right) \tag{4}$$

where $\mathbf{y}_t = \frac{\mathbf{z}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot (1 - l\tau) \mathbf{v}_{t-1}^{\mathrm{r}}}{\tau} - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{rr}} \cdot s(\mathbf{v}_{t-1}^{\mathrm{r}}) - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{rp}} \cdot \mathbf{v}_{t-1}^{\mathrm{p}}$. As a consequence, $\mathbf{z}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_t^{\mathrm{r}}$ appears as a linear function of $\mathbf{W}^{\mathrm{ar}}$, which will make it possible to use classical algorithms for minimization. Note that the particular dynamics of the reservoir in (1) was chosen so that such a linear problem would appear. It also explains why we cannot unravel the dynamics for more time steps: the reformulation into a least square problem would be impossible. From this formulation, it becomes clear that pre-multiplying the factor $\mathbf{z}_t - \mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{v}_t^{\mathrm{r}}$ by $(\mathbf{W}_{\mathrm{l}}^{\mathrm{p}} \cdot \mathbf{W}^{\mathrm{ra}})^{\dagger}$ for the motor learning with the RLS algorithm in Eq. (3) is useful. Indeed, motor learning

becomes the minimization of

$$H_t^a = \sum_{s=0}^{t} \lambda_a^{t-s} \|(\mathbf{W}_{\downarrow}^{p} \cdot \mathbf{W}^{ra})^{\dagger} \cdot \mathbf{y}_s - \mathbf{W}^{ar} \cdot \mathbf{v}_{s-1}^{r}\|^2 + \mu_a \|\mathbf{W}^{ar}\|^2 \quad (5)$$

which takes the form of a classic weighted least square problem, which can directly be solved by a RLS algorithm. Note that matrix $\mathbf{W}_{\downarrow}^{p} \cdot \mathbf{W}^{ra}$ has the size of the number of controlled dimensions (spatial dimension of the target) times the number of dimension for action, which is likely to be small enough to enable a computationally cheap inversion at each time step.

The greedy RLS algorithm consists in implementing (3) with a RLS algorithm, which quickly forgets the past. Indeed, to circumvent the dependency of $\mathbf{u}$ and $\mathbf{W}^{pr} \cdot \mathbf{v}^{r}$ on $\mathbf{W}^{ar}$, the choice of the forgetting factors λ_p, λ_a is crucial. When the network starts from an uninformed state (e.g., the null state), it is important for them to have small values, typically 0.99. A rule of thumb (Haykin 2014) to tune them is that the memory of the algorithm roughly corresponds to $\frac{1}{1-\lambda}$ time steps.

The regularized RLS algo is recalled in this paragraph. It is an algorithm that recursively solves the following generic problem

$$\underset{\mathbf{W}_t}{\text{minimize}} \quad H_t = \sum_{s=0}^{t} \lambda^{t-s} \|\mathbf{x}_s - \mathbf{W}_t.\mathbf{y}_s\|^2 + \mu \|\mathbf{W}\|^2 \quad (6)$$

It can be solved by iterating the following regularized RLS step (Haykin 2014; Gunnarsson 1996) as new targets $\mathbf{x}$ and new observation $\mathbf{y}$ arrive. This algorithm has already been successfully used for echo state networks (Jaeger and Haas 2004; Sussillo and Abbott 2009; Laje and Buonomano 2013). In this paper, we use a version of the algorithm with non-fading regularization (Gunnarsson 1996), which provides good stability properties even when the forgetting factor λ is small. The algorithm corresponding to one step of the considered RLS algorithm is given in algorithm 1. In this algorithm, $\mathbf{P} \in \mathbb{R}^{n \times n}$ is the inverse correlation matrix of the observations, which correspond to the reservoir activity in this paper.

Algorithm 1 Regularized RLS step

procedure RLS_STEP($\mathbf{P}$, $\mathbf{W}$, $\mathbf{x}$, $\mathbf{y}$, μ, λ)
  $\mathbf{P} \leftarrow \mathbf{P} - \mu \mathbf{P}^2$
  $\mathbf{P} \leftarrow \frac{1}{\lambda}\left(\mathbf{P} - \frac{\mathbf{P}.\mathbf{y}.\mathbf{y}'.\mathbf{P}}{\lambda + \mathbf{y}'.\mathbf{P}.\mathbf{y}}\right)$
  $\mathbf{W} \leftarrow \mathbf{W} + (\mathbf{x} - \mathbf{W}.\mathbf{y}).(\mathbf{P}.\mathbf{y})'$
  return $\mathbf{P}$, $\mathbf{W}$
end procedure

Pseudocode Therefore, the algorithm summarizing the update of the greedy RLS neural network is described in algorithm 2.

Algorithm 2 IDRNN

Initialization:
$\mathbf{v}^p, \mathbf{v}^r, \mathbf{v}^a, \mathbf{W}^{ar} \leftarrow 0$
$\mathbf{W}^{pr}, \mathbf{W}_{ij}^{r} \leftarrow \mathcal{N}(0, *)$
$\mathbf{P}^p, \mathbf{P}^a \leftarrow \eta I_d$
# Main loop:
while $\mathbf{u} \leftarrow$ new stimuli **and** $\mathbf{z} \leftarrow$ new target **do**
  # Motor learning:
  $\mathbf{y} \leftarrow (\mathbf{W}_{\downarrow}^{p}.\mathbf{W}^{ra})^{\dagger}.\left(\mathbf{z} - (1 - l\tau)\mathbf{W}_{\downarrow}^{p}.\mathbf{v}^{r} - \tau \mathbf{W}_{\downarrow}^{p}.\mathbf{W}^{rr}.s(\mathbf{v}^{r}) - \tau \mathbf{W}_{\downarrow}^{p}.\mathbf{W}^{rp}.\mathbf{v}^{p}\right)$
  $\mathbf{P}^a, \mathbf{W}^{ar} \leftarrow$ RLS_STEP($\mathbf{P}^a$, $\mathbf{W}^{ar}$, $\mathbf{y}$, $\mathbf{v}^r$, μ_a, λ_a)
  # Reservoir update:
  $\mathbf{v}^{r} \leftarrow (1 - l\tau)\mathbf{v}^{r} + \tau \mathbf{W}^{r}.\left(\mathbf{v}^{p}\ s(\mathbf{v}^{r})\ \mathbf{v}^{a}\right)'$
  # Perceptive learning:
  $\mathbf{P}^p, \mathbf{W}^{pr} \leftarrow$ RLS_STEP($\mathbf{P}^p$, $\mathbf{W}^{pr}$, $\mathbf{u}$, $\mathbf{v}^r$, μ_p, λ_p)
  # Perception and action update:
  $\mathbf{v}^{p} \leftarrow (1 - \alpha)\mathbf{W}^{pr}.\mathbf{v}^{r} + \alpha \mathbf{u}$
  $\mathbf{v}^{a} \leftarrow \mathbf{W}^{ar}.\mathbf{v}^{r}$ is the action that controls the environment.
end while

5 Numerical experiments

In this section, I show on different examples that the proposed architecture can effectively control an unknown environment along a desired trajectory. It is important to realize that the following numerical simulations correspond to a neural network that learns to control the environment: at time $t = 0$, the neural network has no clue about what system it is dealing with. Thus, the results are not to be compared to a classical control setup where the controller was previously tuned to the specific environment.

5.1 Learning to control random neural networks

As toy models of the environment, I first choose randomly connected neural networks from which a linear readout is to be controlled along a sinus function. It has been argued that this class of systems is very rich since it can approximate any dynamical system (Sontag 1997). I do not claim that the IDRNN can control any such random neural network since they can become very complicated, but it performs well on reasonably complicated instances of such networks. I will consider neural networks made of 5 neurons from which a one-dimensional linear readout is computed. The equation governing these driven random neural networks is

$$\begin{cases} \mathbf{x}_{t+1} = (1 - \bar{l}\bar{\tau})\mathbf{x}_t + \tau \bar{\mathbf{W}} \cdot \tanh(\mathbf{x}_t) + \mathbf{b} \cdot \mathbf{v}_t^{a} \\ \mathbf{u}_t = \mathbf{c}' \cdot \mathbf{x}_t \end{cases}$$

where matrix $\bar{\mathbf{W}} \in \mathbb{R}^{5 \times 5}$ and the vectors $\mathbf{b}, \mathbf{c} \in \mathbb{R}^5$ have components that are drawn i.i.d. $\mathcal{N}(0, \bar{\sigma}^2)$, $\mathcal{N}(0, \bar{\kappa}^2)$, and $\mathcal{N}(0, \bar{\gamma}^2)$, respectively.

With different choices of parameters, one can get very different resulting dynamics as shown in (Figs. 2a, 3a, 4a). Even with identical parameters, the different realizations of the connections can lead to qualitatively different dynamics.

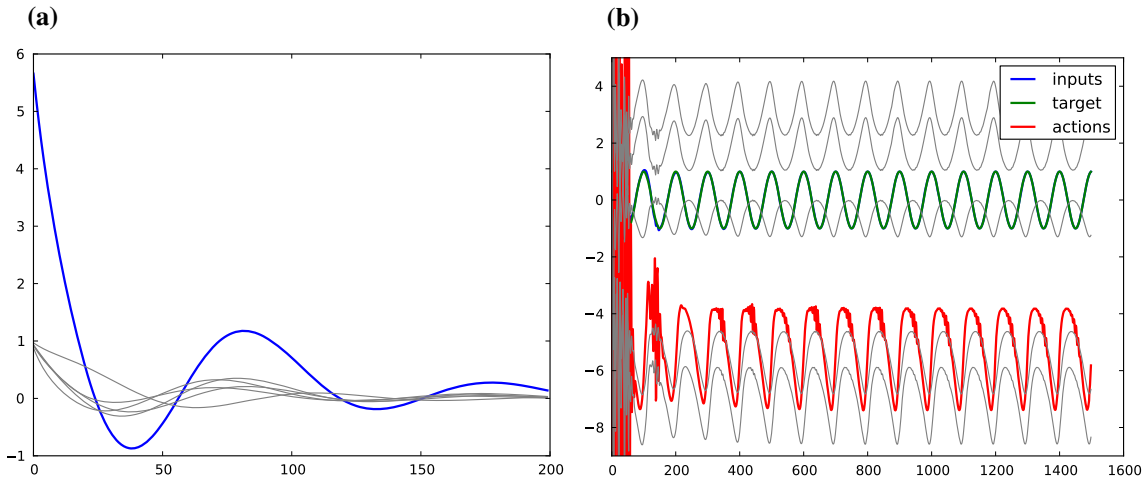

Fig. 2 Learning to control a converging random neural network. In *gray* are represented the five components of the environment states $\mathbf{x}_t$. In *blue* is the readout from the environment $\mathbf{u}_t$, which corresponds to the stimuli or inputs to the IDRRN. In *red* are the actions $\mathbf{v}_t^{\mathrm{a}}$ of the IDRNN on the environment. Environment parameters: $\bar{\sigma} = \bar{\kappa} = \bar{\gamma} = 1$, $\bar{I} = 1$, $\bar{\tau} = 0.1$. **a** Freely run environment, **b** Controlled environment (color figure online)

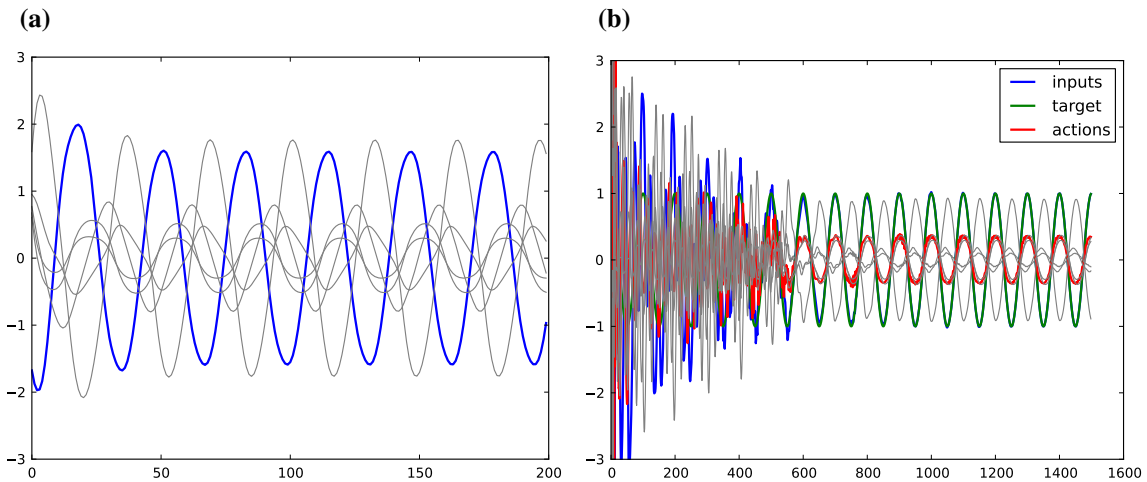

Fig. 3 Learning to control an oscillatory random neural network. See Fig. 2 for the color code. Environment parameters: $\bar{\sigma} = \bar{\kappa} = \bar{\gamma} = 1$, $\bar{I} = 1$, $\bar{\tau} = 0.15$. **a** Freely run environment, **b** controlled environment (color figure online)

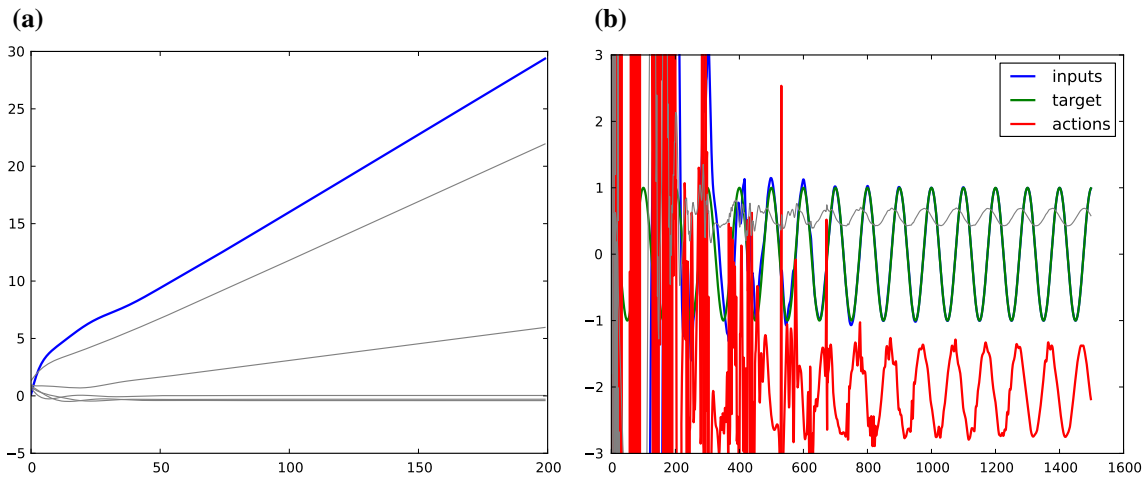

Fig. 4 Learning to control a diverging random neural network. See Fig. 2 for the color code. Environment parameters: $\bar{\sigma} = \bar{\kappa} = \bar{\gamma} = 1$, $\bar{I} = 0$, $\bar{\tau} = 0.1$. **a** Freely run environment, **b** controlled environment (color figure online)

I chose three cases corresponding to converging (Fig. 2), oscillatory (Fig. 3), and diverging (Fig. 4) situations.

For all the simulations, I chose the same parameters for the IDRNN: $n_\mathrm{a} = n_\mathrm{p} = 1$, $n_\mathrm{r} = 100$, $\sigma = 1.5$, $\kappa = \gamma = 0.1$, $l = 1$, $\tau = 0.1$, $\alpha = 0.5$, $\eta = 1$, $s(x) = \tanh(x + 0.01)$, $\lambda_\mathrm{p} = 0.99$, $\lambda_\mathrm{a} = 0.9$, $\mu_\mathrm{p} = 10^{-6}$, $\mu_\mathrm{a} = 10^{-3}$. The motivation for the choice of $\sigma > 1$ and $s(0) \neq 0$ is to have a reservoir with spontaneous activity and no equilibrium point at $\mathbf{v}^\mathrm{r} = 0$ when no stimulation comes in. The parameters of the environment are reported in the figures.

In each situation, the very same neural network has managed to control qualitatively different environments it knew nothing about along the same trajectory $z_t = \sin(2\pi t/100)$. The L_1 error between stimuli and target over the last 100 time steps (corresponding to a period of the target) is, respectively, 0.0006, 0.0046, and 0.0017, for converging, oscillatory, and diverging situations, meaning that the control is successful.

In each case, the first few hundred time steps display an extremely variable behavior. The network has difficulties choosing relevant actions since its perception is not fully developed yet. Although the variations look completely unstructured, the network still manages to calm down and converges to the desired trajectory. Unfortunately, this is not always the case: the search period can bring the environment to an undesired location in the state space leading to a failure of the control. For instance, controlling a quickly diverging environment (not shown) can fail when the environment diverges faster than the network can learn. The two important parameters to control the behavior of the network in the beginning are the initial values of $\mathbf{W}^\mathrm{pr}$ and $\mathbf{P}_p$, $\mathbf{P}_a$. If the value of $\mathbf{W}^\mathrm{pr}_0$ is too small, then the pseudoinversion of $\mathbf{W}^\mathrm{p}_\mathrm{i}.\mathbf{W}^\mathrm{ra}$ in the algorithm can lead to extremely large control values in the very beginning of the simulation. Similarly, choosing large initial values for $\mathbf{P}_p$ or $\mathbf{P}_a$ induces a large variability at the beginning of the simulation (Haykin 2014), which can lead to poor results.

Observe in Fig. 4b that the hidden variables in the environment $\mathbf{x}$ are not displayed. Indeed, although the network is controlling properly the environment readout $\mathbf{u}$, it does not necessarily prevent these hidden variables from diverging. This leads to a numerical overflow in the simulation after a certain time and an increased variability due to numerical round-offs. To prevent this, one would need to ask the IDRNN to control also the states $\mathbf{x}$.

5.2 Learning to control delayed systems

Actually, the IDRNN in its current form fails at controlling delayed systems or even some instantaneous systems, which for which the impact of an action may take some time steps to reveal itself. This is not a surprise since the network only tries to control what it happening from one time step to the other.

A simple extension of the algorithm 2 makes the IDRNN able of controlling such delayed systems. The idea is to learn to predict the future of the stimuli at $t + \delta$ time steps with $\delta > 1$. This corresponds to adding a third readout matrix $\mathbf{W}^\mathrm{fr}$ to the network in Fig. 1. The predicted value $\mathbf{W}^\mathrm{fr}.\mathbf{v}^\mathrm{r}_t$ should be a approximation of $\mathbf{u}_{t+\delta}$ and is not fed back to the network. In practice, learning such a future prediction matrix can be done by applying an RLS algorithm to approximate $\mathbf{u}_t$ based on the observation $\mathbf{v}^\mathrm{r}_{t-\delta}$. This implies to store the history of the reservoir during δ time steps. Once this prediction is set up, it suffices to replace $\mathbf{W}^\mathrm{p}_\mathrm{i}$ by $\mathbf{W}^\mathrm{fr}$ in algorithm 2. This leads to a delayed version of the IDRNN detailed in algorithm 3, which does have a time window similar to (Waegeman et al. 2012).

Algorithm 3 Delayed IDRNN

```
# Initialization:
v^p, v^r, v^a, W^pr, W^ar ← 0
W^fr, W^r_ij ← N(0, *)
P^p, P^a, P_f ← η I_d
# Main loop:
while u ← new stimuli and z ← new target do
    # Motor learning:
    y ← (W^fr.W^ra)^†. (z − (1 − lτ)W^fr.v^r − τW^fr.W^rr.s(v^r)
        −τW^fr.W^rp.v^p)
    P^a, W^ar ← RLS_STEP(P^a, W^ar, y, v^r, μ_a, λ_a)
    # Reservoir update:
    v^r ← (1 − lτ)v^r + τW^r. (v^p  s(v^r)  v^a)'
    v^r_list ← [v^r, v^r_list]
    # Perceptive learning:
    P^p, W^pr ← RLS_STEP(P^p, W^pr, u, v^r, μ_p, λ_p)
    P^f, W^fr ← RLS_STEP(P^f, W^fr, u, v^r_list(δ), μ_f, λ_f)
    # Perception, future and action update:
    v^p ← (1 − α)W^pr.v^r + αu
    v^f ← W^fr.v^f is a proxy of the future inputs.
    v^a ← W^ar.v^r is the action which controls the environment.
end while
```

With this new algorithm, it is possible to control systems for which the instantaneous IDRNN failed. Here, I consider two examples of such systems: first, a linear oscillatory system with a single control variable, and second, a random neural network with delayed action.

In a first time, we consider the following environment to control

$$\begin{pmatrix}\mathbf{u}_{t+1}\\ \mathbf{x}_{t+1}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_t\\ \mathbf{x}_t\end{pmatrix} + 0.1\begin{pmatrix}-0.05 & -1\\ 1 & -0.05\end{pmatrix}\cdot\begin{pmatrix}\mathbf{u}_t\\ \mathbf{x}_t\end{pmatrix} + 0.1\mathbf{v}^\mathrm{a}_t\begin{pmatrix}1\\ 0.5\end{pmatrix}$$

This corresponds to a linear oscillatory system as shown in Fig. 5a somewhat similar to the first system studied in (Waegeman et al. 2012). The negative terms on the diagonal compensate for the tendency of the Euler algorithm to diverge when simulating linear systems and can be disregarded for the interpretation. Actually, the difficulty to control the system comes from the control vector (1 0.5), which is positive for both components. To understand intuitively the problem,

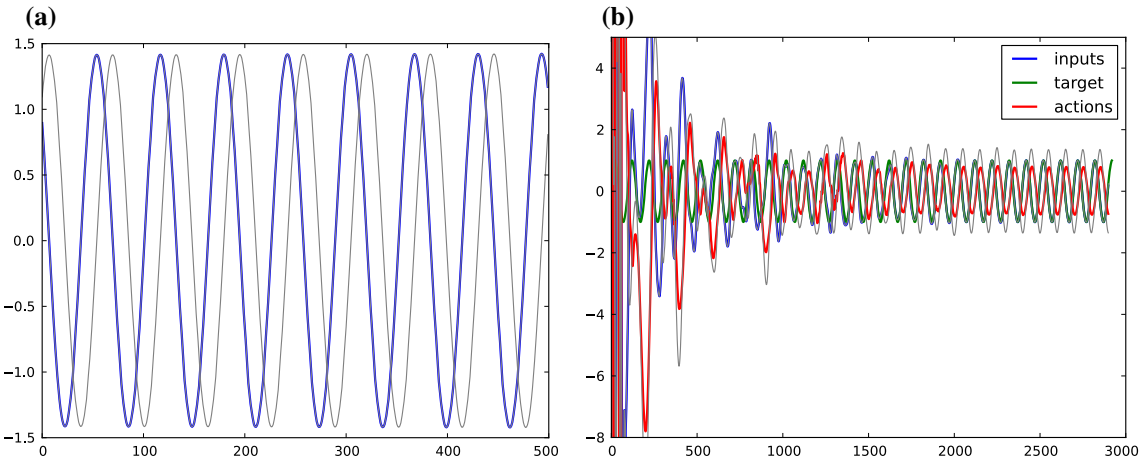

Fig. 5 Learning to control a linear oscillatory system with a single positive control variable. See Fig. 2 for the color code. **a** Freely run environment, **b** controlled environment (color figure online)

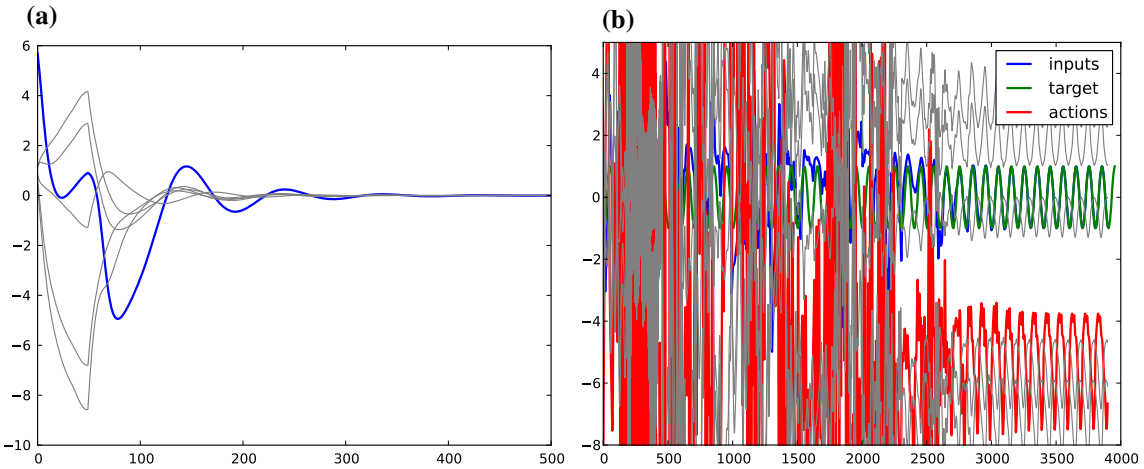

Fig. 6 Learning to control of a random neural network with a delayed control variable. See Fig. 2 for the color code. Environment parameter: $\bar{\sigma} = \bar{\kappa} = \bar{\gamma} = 1$, $\bar{l} = 1$, $\bar{\tau} = 0.1$. **a** Freely run environment, **b** controlled environment (color figure online)

it is useful to see that one dimension is excitatory while the second is inhibitory. When the control variable only makes it possible to positively stimulate both dimensions, it is not clear what will be the situation in a few time steps: although the excitatory dimension is directly stimulated by the inputs, the inhibitory dimension will suppress this excitation (even more since it is stimulated by the excitatory dimension). In the end, one needs to see a bit more in the future to understand the impact of the action. This is precisely why adding the time window δ to the prediction makes the task much more simple. Indeed, the IDRNN can effectively control the system as shown in Fig. 5b. The parameters used are the same as previously with the addition of $\delta = 20$, $\lambda_f = 0.995$, and $\eta = 10^{-4}$.

The second environment is governed by a random neural network as previously but with a delayed command

$$\begin{cases} \mathbf{x}_{t+1} = (1 - \bar{l}\bar{\tau})\mathbf{x}_t + \tau\bar{\mathbf{W}} \cdot \tanh(\mathbf{x}_t) + \mathbf{b} \cdot \mathbf{v}^{\mathrm{a}}_{t-50} \\ \mathbf{u}_t = \mathbf{c}'.\mathbf{x}_t \end{cases}$$

The parameters of the IDRNN are identical to before except that $\delta = 55$, λ_a and $\eta = 10^4$, which explains the great variability at the beginning of the simulation.

The control performed by the IDRNN is displayed in Fig. 6. Although the control takes more time (4000 time steps instead of 1500), it finally manages to bring the system along the desired trajectory.

5.3 Short comparison with the double reservoir architecture

Although an extensive comparison between the IDRNN and the double reservoir architecture (DRA) (Waegeman et al. 2012) is beyond the scope of this paper, I show here that they perform similarly on the task of controlling a heating tank.

The system to control, a heating tank, is the same as the second example in (Waegeman et al. 2012), and the interested reader should refer to this paper for the implementation details (I took the same parameters). Briefly, the system

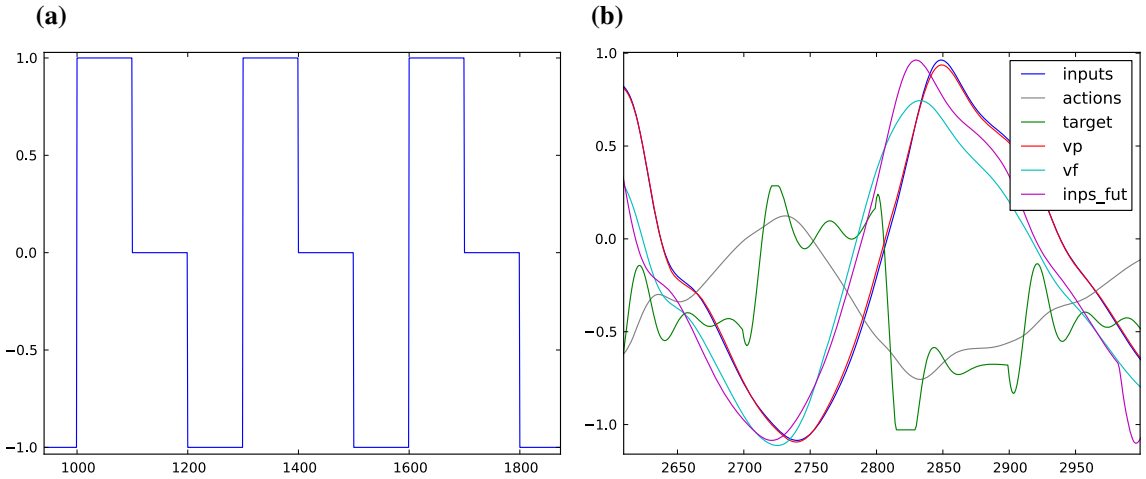

Fig. 7 (*Left*) Target trajectory used for the heating tank experiment. (*right*) Result of the initialization of the IDRNN. The actual implementation of (algorithm 3) started *right* after. In *blue* are the inputs $\mathbf{u}_t$, and in *pink* are the future inputs $\mathbf{u}_{t+\delta}$. In *gray* are the actions $\mathbf{v}_t^{\mathrm{a}}$. In *green* is the target for the initialization phase (different from $\mathbf{z}_t$). Note that it is a target for the action and not the perception. In *red* is the perception $\mathbf{v}^{\mathrm{p}}$, and in *cyan* in the prediction of the future $\mathbf{v}^{\mathrm{f}}$. **a** Target trajectory $\mathbf{z}$, **b** initialization of IDRNN (color figure online)

is a nonlinear state-delayed system where the delay depends on the action. It corresponds to controlling a heating tank whose input is a stream of cold water. The water is heated in the tank and is then channeled through a tube to a destination where the output temperature is measured. When the water throughput (corresponding to the action $\mathbf{v}^{\mathrm{a}}$) is increased, the output temperature decreases, but it takes less time to get out of the tube. It is shown in (Waegeman et al. 2012) that the DRA outperforms another algorithm, called NEPSAC (Gálvez-Carrillo et al. 2009), on this task.

In this paper, there are two differences from the heating tank used in (Waegeman et al. 2012): first, the stimuli to the network are scaled to meet the networks dynamics. I take as input $\frac{T-30}{5}$ where T is the output of the heating tank simulation. Second, the target trajectory is different, see Fig. 7a. It is much faster, so that the results of the control algorithm do not look as good as in the original paper presenting the DRA. I have not tried to tune the parameters to get the best results, since I only intended to show that both algorithms would perform similarly on this task with naive parameter tuning.

Actually, the IDRNN failed to control the system when it is randomly initialized. Interestingly, the DRA could control the system, although I did not use any babbling initialization as discussed in (Waegeman et al. 2012). To get good results with the IDRNN, I had to properly initialize the network. Indeed, a useful property of the IDRNN is its ability to be initialized along any preexisting control scheme. The initialization procedure consists in recording both perceptions and actions (more commonly named inputs and outputs) of another control scheme (here I took the DRA) during a long time (here I took 3000 time steps). Then, a simple ESN procedure can be used to reproduce the recorded trajectories by

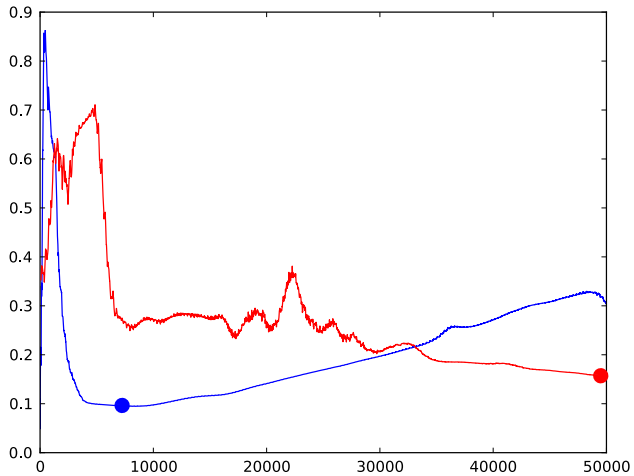

Fig. 8 Control performance of IDRNN (*red*) and DRA (*blue*). The L_1 distance between the target $t \mapsto \mathbf{z}_t$ and $t \mapsto \mathbf{u}_{t+\delta}$ over a siding window of size 900 is used for this performance index. The *two dots* correspond to Fig. 9 (color figure online)

learning the appropriate feedback connections $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$. After learning, when the very same environment is presented again to the neural network, it will reproduce both predictions and actions of the learned control scheme. However, if the learning time was too small or the environment variable enough, then there may be some discrepancies between recorded and reproduced trajectories, which is the case in this application: at the end of initialization, the system had a behavior, which was very different from the DRA, see Fig. 7b. Nonetheless, it was sufficient for the coupled system to be in an attractor *basin* and the control did work.

I simulated the IDRNN and the DRA control for 50,000 time steps and obtained a performance displayed in (Fig. 8). It appears that the DRA converges faster to a good con-

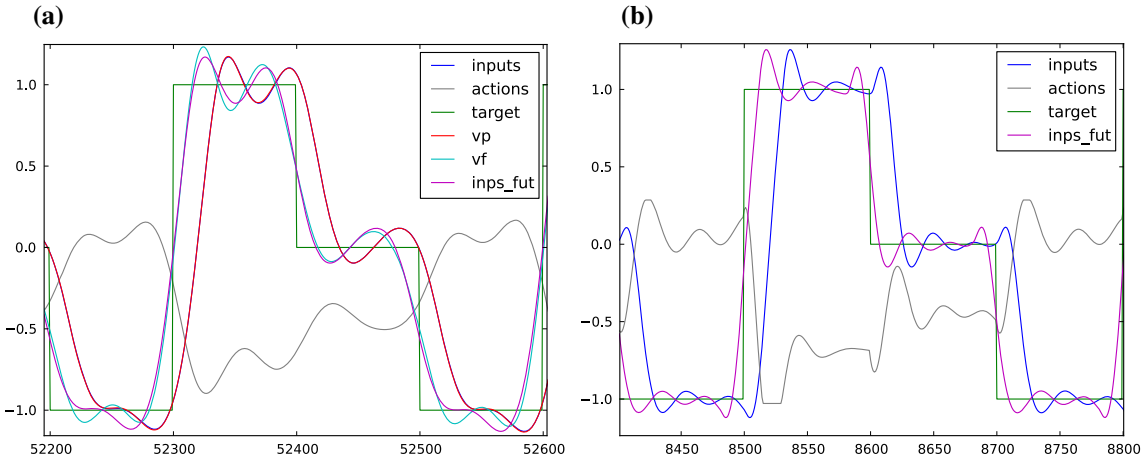

Fig. 9 Results of the control of the heating tank for the IDRNN (*left*) and DRA (*right*). The IDRNN control corresponds to the *red dot* in Fig. 8, and the DRA control corresponds to the *blue dot* in Fig. 8. In *blue* are the inputs $\mathbf{u}_t$, and in *pink* are the future inputs $\mathbf{u}_{t+\delta}$. In *gray* are the actions $\mathbf{v}_t^{\mathrm{a}}$. In *green* is the target $\mathbf{z}_t$. In *red* is the perception $\mathbf{v}^{\mathrm{p}}$, and in *cyan* in the prediction of the future $\mathbf{v}^{\mathrm{f}}$. **a** IDRNN control, **b** DRA control (color figure online)

trol behavior, see (9b) but its performance slowly deteriorates with time. On the contrary, the IDRNN converged more slowly but its performance improves with time. Eventually, the system behavior (see 9a) is similar to the best case for the DRA, although a bit worse. The parameters used are the same as in the previous section with the addition of $\delta = 20$, $\lambda_{\mathrm{p}} = \lambda_{\mathrm{a}} = \lambda_{\mathrm{f}} = 0.9999$, $\mu_{\mathrm{p}} = \mu_{\mathrm{f}} = \mu_{\mathrm{a}} = 10^{-6}$. The reason why I take the forgetting parameters $\lambda_{\mathrm{p}} = \lambda_{\mathrm{a}} = \lambda_{\mathrm{f}}$ to be so large is because an initializing method is used, and I want the network not to forget immediately the initialization. I also reset the matrix $\mathbf{P}_a$ to $0.002 I_d$ at the end of the initialization to slow learning down to stay close to initialization at first. The parameters for the DRA are identical to that of the IDRNN.

To summarize, the two methods seem to roughly give similar results at their optimum. However, they seem to differ in the evolution of the performance with time.

6 Discussion

Beyond the proof of concept supported by the previous numerical experiments, IDRNN has some notable properties that are discussed in this section.

6.1 Reproducing other control schemes

An important characteristic of IDRNN (and also DRA) is its universality, in the sense that it can embed any existing control schemes. Indeed, as detailed in the previous section, it is possible to initialize these networks, using traditional ESN algorithms. The procedure is, first, to record the inputs (perception) and (output) of a preexisting control scheme for a given environment, and second, to use traditional ESN algorithms on the IDRNN architecture in order to reproduce these trajectories when exposed to a similar environment.

Sontag has proven that, in theory, recurrent neural networks could reproduce any dynamical systems (Sontag 1997). Thus, there exists an IDRNN, possibly with a very large number of neurons, which can emulate a given control scheme arbitrarily accurately. Naturally in applications, the number of neurons is limited, and the reproduction is not necessarily accurate, even more if the recorded trajectories were not long enough (see previous section). Nonetheless, this method can be used as a precious initialization procedure in order to design incremental control schemes.

6.2 Extension to reinforcement learning

Although the present version of IDRNN is explicitly designed for a supervised learning framework, it is possible to extend the approach to a reinforcement learning framework. The first step is to include a reward in the present model (which the neural network will try to maximize). This can be easily done by adding a stimulus neuron exclusively excited when a reward in presented. Thus, implementing a reinforcement learning approach would simply correspond to maximizing the activity of this reward neuron instead of minimizing the distance between such a neuron and a target trajectory. In the mathematical formalism, this can be immediately implemented by replacing the target $\mathbf{z}_t$ by 0 in (2) and asking the motor learning to maximize (rather than minimize) its criterion. Thus, motor learning aims at increasing the current and future rewards.

Besides, the network can also handle a variety of hybrid approaches corresponding for instance to a mix of supervised/reinforcement learning. More precisely, an interesting situation is to design an agent with a lot of stimuli (including a reward) all of which perceptive learning aims at predicting, while motor learning exclusively aims at increasing the reward. This would correspond to a common reinforcement learning situation where there is no target, while making sure the network can behave coherently and in context with its environment.

The IDRNN approach has, in principle, no problem handling rare or intermittent rewards. Indeed, it still produces behavior when the rewards are very sparse. The IDRNN network activity is always running and not directly influenced by the presence of a reward. Only learning is directly boosted when a reward is presented. In my opinion, this provides a notable advantage of IDRNN over DRA, which cannot be extended to reinforcement learning so easily because the activity of the network is directly influenced by the target/reward. Thus, with rare rewards, i.e., sparse target, the DRA approach would not work.

The proper handling of distal rewards (Jordan and Rumelhart 1992) by the IDRNN is still an open question, but the machinery introduced here for the prediction of the future in several time steps seems to be an appropriate way to treat this difficult problem. Nonetheless, this stands as a perspective to be investigated.

6.3 Biological plausibility

Of course, I do not claim the full biological plausibility for the IDRNN approach because such simple networks can never represent the incredible complexity found in neuroscience. However, I would argue that this approach is biologically more plausible than the DRA. Indeed, the latter approach uses two reservoirs with exactly the same recurrent and output weights. This clearly breaks the locality requirement of biologically plausible algorithms (Gerstner and Kistler 2002) to modify the connection between neurons i and j, learning rules should only include information available at neurons i and j. On the other hand, the IDRNN approach is based on a single reservoir, and there is no sharing of the weights. Thus, contrary to DRA, it is not biologically implausible by construction.

Although this paper is based on RLS implementation of the ideomotor principles (for efficiency reason), it is also possible to design for LMS implementation, which is more biologically plausible. Indeed, RLS is not local since it involves computing the inverse of the global correlation matrix. LMS corresponds to a simple stochastic gradient descent of the ideomotor principles in (2). It leads to slower convergence times, but more robustness, than the RLS algorithm (Haykin 2014), but both aim at solving the same problem. The ideomotor LMS algorithm can be derived from the modified ideomotor principles (3) under the same greedy assumption that $\mathbf{u}$ and $\mathbf{v}^{\mathrm{r}}$ do not depend on $\mathbf{W}^{\mathrm{pr}}$ and $\mathbf{W}^{\mathrm{ar}}$. It comes as the gradient descent of H^p and H^a. Assuming that we consider the reinforcement learning case detailed above, where $\mathbf{z} = 0$ and motor learning is a maximization, it can be written as

$$\begin{aligned} \mathbf{W}^{\mathrm{pr}}_{t+1} &= \mathbf{W}^{\mathrm{pr}}_{t} + \epsilon_p(\mathbf{u}_t - \mathbf{W}^{\mathrm{pr}}_{t} \cdot \mathbf{v}^{\mathrm{r}}_{t}) \cdot \mathbf{v}^{\mathrm{r}\prime}_{t} \\ \mathbf{W}^{\mathrm{ar}}_{t+1} &= \mathbf{W}^{\mathrm{ar}}_{t} + \epsilon_a \mathbf{W}^{\mathrm{p}}_{\mid} \cdot \mathbf{v}^{\mathrm{r}}_{t} \cdot \mathbf{v}^{\mathrm{r}\prime}_{t-1} \end{aligned} \tag{7}$$

where ϵ_p and ϵ_a have been re-parametrized to absorb constants.

In this form, perceptive learning is local. Indeed, the modification of the connection $\{\mathbf{W}^{\mathrm{pr}}\}_{ij}$ only depends on the stimulus $\{\mathbf{u}\}_i$, the prediction $\{\mathbf{W}^{\mathrm{pr}}_t \cdot \mathbf{v}^{\mathrm{r}}\}_i$, and the reservoir state $\{\mathbf{v}^{\mathrm{r}}\}_j$, which are locally available quantities between reservoir and perceptive area.

Motor learning can also be said to be plausible if we slightly nuance the locality requirement by the experimental fact that a few modulatory synapses can bring some additional information to distant connections as shown in Fig. 10. Indeed, the modification of the connection $\{\mathbf{W}^{\mathrm{ar}}\}_{ij}$ depends on the reservoir state $\mathbf{v}^{\mathrm{r}}{}_j$, but also on the $\mathbf{W}^{\mathrm{p}}_{\mid} \cdot \mathbf{v}^{\mathrm{r}}$, which is not available between reservoir and motor area. Thus, there is a need for a modulatory synapse, involved mainly in learning, from the perceptive area to the motor connections to biologically implement this algorithm.

Interestingly, LMS and RLS can be shown to be equivalent if the reservoir has a temporal correlation matrix proportional to the identity (Farhang-Boroujeny 1998). This regime has been often observed in biology, where in vivo cortical tissues are said to be in an asynchronous irregular state (Ecker et al. 2010; Renart et al. 2010). This would support the fact

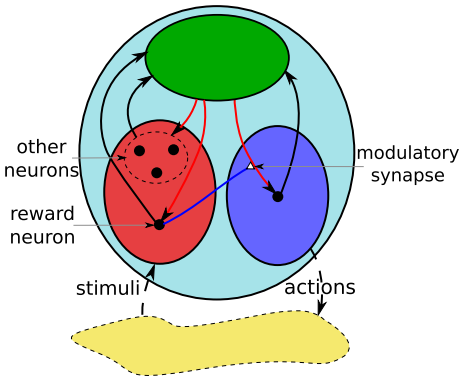

Fig. 10 Diagram showing the necessary wiring of the network if learning has to be local, in the illustrative case of a single reward neuron and a single motor neuron. The perceptive area is made of a reward neuron and other perceptive neurons, all of which receive stimuli from the environment (possibly some other part of the brain). The reward neuron has to send a modulatory synapse to the motor connections. The information from the motor neuron is only used for learning and does not modify directly the dynamics of the motor neuron

that such a biological LMS implementation of the ideomotor principle can be efficient, although showing this rigorously stands as a perspective.

Besides, one might ask how the current scheme differs from active inference that uses explicit forward or generative models of predicted stimuli. They key difference is the simplicity of the current scheme that just involves optimizing connection weights from the reservoir to action and perception states. Heuristically, this can be regarded as equipping an agent with a vast repertoire of generative models and then optimizing the weights to select the model with the greatest evidence (least free energy or prediction error). The connection between the current scheme and active inference may be important from the point of view of biological implementation: there is now a literature on biological schemes for minimizing prediction error using hierarchical predictive coding and Bayesian filtering schemes. Of particular interest here is the role of reflexes in mediating action. The current scheme can be regarded as selecting a forward model of sensations. In active inference, these forward models also predict kinesthetic or proprioceptive sensations. This means that the prediction errors minimized by action can be resolved very simply, through peripheral reflex arcs (Adams et al. 2013).

On the whole, this theory-driven approach may contribute to the debate about the computational role of several parts of the brain. A rigorous link with the brain clearly stands as a long-term perspective. Yet, I think it is interesting to note the ingredients this architecture needs in order to design what could considered as a simplistic embodied agent. The first ingredient is the combination of a central recurrent network and two types of readout, as can be observed in the spinal chord (Butler and Hodos 2005) with the dorsal root for perception and the ventral root for action. The second ingredient is the modulation of motor connections by reward neurons, which seems to be handled in the brain by a variety of neurotransmitters (Seamans and Durstewitz 2008). I believe that the study of basic vertebrate nervous system may, in the long term, benefit from such constructive approaches.

7 Conclusion

This paper defines a recurrent neural network, which blindly learns to control an unknown environment. Based on a randomly connected reservoir, it learns on the fly two readouts that correspond to perception and action. These readouts are learned according to two principles: perceptive learning corresponds to maintaining good predictions of the incoming stimuli; motor learning tries to change the dynamics of the reservoir, so that the stimuli predictions match a target trajectory. Actually, the control of the environment is just a by-product of the behavior of the neural network, which only cares about its own predictions. This algorithm is closely related to the ideomotor theory and active inference, providing an efficient computational implementation of these concepts. An implementation of the proposed method to robotics may highlight the similarities with more classical approaches in this field (Tani 1996).

Several numerical simulations have established a proof of concept for this neural network. It manages to control fairly complicated dynamical systems and properly handles nonlinearities and delays. The robustness of the approach with respect to most parameters is supported by the fact that a single set of parameters (with little tuning) was used to control all environments in this paper.

Several challenges can be foreseen for the future development of such algorithm. First, a extensive benchmarking of IDRNN, DRA, and competitors on real-world environments is needed. Second, the development of a mathematical theory explaining the power of such random networks would be useful. Third, extending the ideomotor approach presented here to a fully connected neural network (e.g., with connections from perceptive to motor area) can be done straightforwardly and may prove more efficient in certain cases. Finally, building architectures with building blocks such as the network presented here could prove interesting, not only for designing even more intelligent agents, but also to shed light on possible information hierarchies that might be implemented by the brain.

Acknowledgments I thank Herbert Jaeger, Michael Thon, Jochen Steil, Felix Reinhart, and Benjamin Schrauwen for helpful discussions. I was funded by the European project AMARSI.

References

Adams RA, Shipp S, Friston KJ (2013) Predictions not commands: active inference in the motor system. Brain Struct Funct 218(3):611–643

Åström KJ (2006) Introduction to stochastic control theory. Courier Dover Publications, New York

Åström KJ, Hägglund T (2006) Advanced PID control. ISA-The Instrumentation, Systems, and Automation Society, Research Triangle Park

Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press, Oxford

Butler AB, Hodos W (2005) Comparative vertebrate neuroanatomy: evolution and adaptation. Wiley, New York

Chow TW, Fang Y (1998) A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics. IEEE Trans Ind Electron 45(1):151–161

Conant RC, Ashby W (1970) Every good regulator of a system must be a model of that system. Int J Syst Sci 1(2):89–97

Doya K (1993) Bifurcations of recurrent neural networks in gradient descent learning. IEEE Trans Neural Netw 1:75–80

Ecker AS, Berens P, Keliris GA, Bethge M, Logothetis NK, Tolias AS (2010) Decorrelated neuronal firing in cortical microcircuits. Science 327(5965):584–587

Farhang-Boroujeny B (1998) Adaptive filters: theory and applications. Wiley, New York

Fortmann TE, Hitz KL (1977) An introduction to linear control systems. CRC Press, Boca Raton
Friston KJ, Daunizeau J, Kilner J, Kiebel SJ (2010) Action and behavior: a free-energy formulation. Biol Cybern 102(3):227–260
Gálvez-Carrillo M, De Keyser R, Ionescu C (2009) Nonlinear predictive control with dead-time compensator: application to a solar power plant. Solar Energy 83(5):743–752
Ge S, Hang CC, Lee TH, Zhang T (2010) Stable adaptive neural network control. Springer, New York
Ge SS, Yang C, Lee TH (2008) Adaptive predictive control using neural network for a class of pure-feedback systems in discrete time. IEEE Trans Neural Netw 19(9):1599–1614
Gerstner W, Kistler WM (2002) Mathematical formulations of hebbian learning. Biol Cybern 87(5–6):404–415
Greenwald AG (1970) Sensory feedback mechanisms in performance control: with special reference to the ideo-motor mechanism. Psychol Rev 77(2):73
Gunnarsson S (1996) Combining tracking and regularization in recursive least squares identification. In: IEEE Conference on Decision and Control, vol 3, pp 2551–2552. Citeseer
Haykin SO (2014) Adaptive filter theory, 5th edn. Pearson Education. http://www.pearsonhighered.com/educator/product/Adaptive-Filter-Theory/9780132671453.page
Jaeger H (2001) The "echo state"approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report 148:34
Jaeger H, Haas H (2004) Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science 304(5667):78–80
Jaeger H, Lukosevicius M, Popovici D, Siewert U (2007) Optimization and applications of Echo State Networks with leaky-integrator neurons. Neural Netw 20(3):335–352
Jordan MI (1996) Computational aspects of motor control and motor learning. Handb Percept Action Motor Skills 2:71–118
Jordan MI, Rumelhart DE (1992) Forward models: supervised learning with a distal teacher. Cogn Sci 16(3):307–354
Kawato M, Furukawa K, Suzuki R (1987) A hierarchical neural-network model for control and learning of voluntary movement. Biol Cybern 57(3):169–185
Kwakernaak H, Sivan R (1972) Linear optimal control systems, vol 1. Wiley, New York
Laje R, Buonomano DV (2013) Robust timing and motor patterns by taming chaos in recurrent neural networks. Nat Neurosci 16(7):925–933
Narendra KS, Parthasarathy K (1990) Identification and control of dynamical systems using neural networks. IEEE Trans Neural Netw 1(1):4–27
Pan Y, Wang J (2012) Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks. IEEE Trans Ind Electron 59(8):3089–3101
Pearlmutter BA (1995) Gradient calculations for dynamic recurrent neural networks: a survey. IEEE Trans Neural Netw 6(5):1212–1228
Prokhorov DV (2007) Training recurrent neurocontrollers for real-time applications. IEEE Trans Neural Netw 18(4):1003–1015
Renart A, de la Rocha J, Bartho P, Hollender L, Parga N, Reyes A, Harris KD (2010) The asynchronous state in cortical circuits. Science 327(5965):587–590
Seamans J, Durstewitz D (2008) Dopamine modulation. Scholarpedia 3(4):2711
Shin YK, Proctor RW, Capaldi E (2010) A review of contemporary ideomotor theory. Psychol Bull 136(6):943
Skogestad S, Postlethwaite I (2007) Multivariable feedback control: analysis and design, vol 2. Wiley, New York
Slotine J-JE, Li W et al (1991) Applied nonlinear control, vol 199. Prentice-Hall, Englewood Cliffs
Sontag ED (1997) Recurrent neural networks: Some systems-theoretic aspects. In: Karny M, Warwick K, Kurkova V (eds) Dealing with complexity: a neural network approach. Springer, London, pp 1–12
Sussillo D, Abbott LF (2009) Generating coherent patterns of activity from chaotic neural networks. Neuron 63(4):544–557
Tani J (1996) Model-based learning for mobile robot navigation from the dynamical systems perspective. IEEE Trans Syst Man Cybern Part B Cybern 26(3):421–436
Waegeman T, Wyffels F, Schrauwen B (2012) Feedback control by online learning an inverse model. IEEE Trans Neural Netw Learn Syst 23(10):1637–1648
Wang C, Hill DJ (2006) Learning from neural control. IEEE Trans Neural Netw 17(1):130–146
Yang C, Ge SS, Xiang C, Chai T, Lee TH (2008) Output feedback nn control for two classes of discrete-time systems with unknown control directions in a unified approach. IEEE Trans Neural Netw 19(11):1873–1886
Zhong-Sheng H (2006) On model-free adaptive control: the state of the art and perspective. Control Theory Appl 4:018