Neural Networks 21 (2008) 862–871

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2008 Special Issue

Event detection and localization for small mobile robots using reservoir computing

E.A. Antonelo *, B. Schrauwen, D. Stroobandt

Department of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

ARTICLE INFO

Article history:
Received 16 January 2008
Received in revised form
21 April 2008
Accepted 17 June 2008

Keywords:
Reservoir Computing
Robot localization
Event detection

ABSTRACT

Reservoir Computing (RC) techniques use a fixed (usually randomly created) recurrent neural network, or more generally any dynamic system, which operates at the edge of stability, where only a linear static readout output layer is trained by standard linear regression methods. In this work, RC is used for detecting complex events in autonomous robot navigation. This can be extended to robot localization tasks which are solely based on a few low-range, high-noise sensory data. The robot thus builds an implicit map of the environment (after learning) that is used for efficient localization by simply processing the input stream of distance sensors. These techniques are demonstrated in both a simple simulation environment and in the physically realistic Webots simulation of the commercially available e-puck robot, using several complex and even dynamic environments.

1. Introduction

Autonomous robot navigation systems have been extensively developed in the literature (Antonelo, Baerlvedt, Rognvaldsson, & Figueiredo, 2006; Arkin, 1998; Guivant, Nebot, & Baiker, 2000). Early navigation strategies are either deliberative (generation of robot trajectories based on path planning) or reactive (robot control based on a direct mapping of sensory input to actions). Current state-of-the-art autonomous robot control architectures are *hybrid* (Arkin, 1998): they have an underlying reactive controller which takes care of the real-time basic behaviors such as obstacle avoidance; while an upper deliberative control layer steers this reactive part for performing declarative high level tasks such as planning. Information flow in this architecture is both downwards, from abstract deliberative tasks to concrete physical reactive behaviors, and upwards, from physical data to abstract symbols used for deliberative planning.

This paper investigates two cases of upward information flow: a system for recognizing complex events in particular environments (such as detecting if the robot goes through a door); and a system for determining the current robot location. Both are based solely on low-range, high-noise sensory information, typically found in small and cheap mobile robots. Both tasks are achieved using the same setup.

These tasks have been shown to be difficult (Bailey & Durrant-Whyte, 2006). Traditional algorithms based on the Simultaneous Localization and Mapping (SLAM) concept are expensive to implement due to high computational and memory demands and also hold uncertainties during the calculation of the robot's pose (Bailey & Durrant-Whyte, 2006). They usually need high precision ranging data from, for example, a 2D laser range scanner. These devices are currently still very expensive, consume a lot of power, and cannot be applied in small robots. Cheap, small and lightweight robots that have a high battery autonomy will thus not be able to use a SLAM-based approach. These robot platforms usually only have access to a limited number of ranging sensors which are low range and have high noise.

This work uses an implicit way of forming a representation of the robot's environment that is based on a Recurrent Neural Network (RNN), more specifically using Reservoir Computing (RC). This is a term that groups three similar computing techniques, namely, Echo State Networks (Jaeger, 2001a), Liquid State Machines (Maass, Natschläger, & Markram, 2002), and BackPropagation DeCorrelation (Steil, 2004). All three techniques are characterized by having a fixed (usually random) RNN that is used as a reservoir of rich dynamics and a linear static readout output layer (see Fig. 1). Only the readout layer is trained by supervised learning, while the recurrent part of the network (the so called *reservoir*) has fixed weights, but is scaled so that its dynamic regime is at the edge of stability. Theoretical analysis of reservoir computing methods (Jaeger, 2001b) and a broad range of applications (Verstraeten, Schrauwen, D'Haene, & Stroobandt, 2007) (which sometimes even drastically outperform the current state-of-the-art (Jaeger & Haas, 2004)) show that RC is very powerful and overcomes many of the problems of traditional RNN training such as slow convergence, bifurcations and high computational requirements.

* Corresponding author. Tel.: +32 9 264 3404; fax: +32 9 264 35 94.
E-mail address: eric.antonelo@elis.ugent.be (E.A. Antonelo).
URL: http://www.elis.ugent.be/SNN (E.A. Antonelo).

0893-6080/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.06.010

The short-term memory, present in these networks, is crucial for solving the event detection and localization tasks. It is not only the instantaneous sensory inputs that are needed to solve the tasks, but also the sensory history (Schönherr, Cistelecan, Hertzberg, & Christaller, 2001) and dynamics.

It has already been shown in Hertzberg, Jaeger, and Schönherr (2002) that RC can be used to detect events in an autonomous robot setting. This work extends these results by also considering dynamic environments for event detection, and goes largely beyond that work by using it to construct implicit maps of the environment for robot localization.

The idea of employing a neural network as a localization model for the robot is also inspired by biological systems. Experiments accomplished on rats show that their hippocampus forms activation patterns that are associated with locations visited by the rat. These so called *place cells* encode the spatial location of the animal into its environment. They fire when the animal is in a particular location (O'Keefe & Dostrovsky, 1971). A similar approach is used in this work where distinct outputs are used to encode specific locations in the environment. Other models in literature seek to represent place cells by using: unsupervised growing networks and Hebbian-type learning rules between neural populations (Arleo, Smeraldi, & Gerstner, 2004; Stroesslin, Sheynikhovich, Chavarriaga, & Gerstner, 2005); and a hierarchy of Slow Feature Analysis nodes for self-organized formation of place cells (Franzius, Sprekeler, & Wiskott, 2007) (these models are based on visual (pixel-based) stimuli as external sensory input).

The experiments in this work[1] are performed using both an autonomous robot simulator developed by Antonelo et al. (2006) and the physically realistic Webots (Michel, 2004) simulation of an e-puck robot (e-puck, 2007). The datasets generated by these simulators are used to train a RC system to detect events as well as to predict the robot location in several complex and dynamic environments. The training is done in a supervised fashion, but we plan to develop an autonomous and on-line way of learning novel locations as the robot drives in its environment (resembling the place cells in biological systems).

This work is organized as follows. In Section 2 we give an overview of Reservoir Computing as well as the RC model used for the following robotic experiments. Section 3 presents the two different robot models and simulators used in the experiments. The problems of event detection and robot localization (and their respective experimental results) are presented in Sections 4 and 5, respectively. Conclusions and future research directions are given in Section 6.

2. Reservoir computing

In this work, we use the Echo State Network approach as learning model for performing event detection as well as robot localization.

An ESN is composed of a discrete hyperbolic-tangent RNN (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output. The general state update equation for the nodes in the reservoir and the readout output equation are as follows:

$$\mathbf{x}(t+1) = f\left(\mathbf{W}^{\text{res}}_{\text{res}}\mathbf{x}(t) + \mathbf{W}^{\text{res}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{res}}_{\text{out}}\mathbf{y}(t) + \mathbf{W}^{\text{res}}_{\text{bias}}\right) \quad (1)$$

$$\mathbf{y}(t+1) = \mathbf{W}^{\text{out}}_{\text{res}}\mathbf{x}(t+1) + \mathbf{W}^{\text{out}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{out}}_{\text{out}}\mathbf{y}(t) + \mathbf{W}^{\text{out}}_{\text{bias}} \quad (2)$$

where: $\mathbf{u}(t)$ denotes the input at time t; $\mathbf{x}(t)$ represents the reservoir state; $\mathbf{y}(t)$ is the output; and $f() = \tanh()$ is the hyperbolic tangent activation function (most common type of activation function used for ESNs). The initial state is set to $\mathbf{x}(0) = \mathbf{0}$. All weight matrices to the reservoir (denoted as $\mathbf{W}^{\text{res}}_{\cdot}$) are initialized randomly (represented by solid arrows in Fig. 1), while all connections to the output (denoted as $\mathbf{W}^{\text{out}}_{\cdot}$) are trained (represented by dashed arrows in Fig. 1).

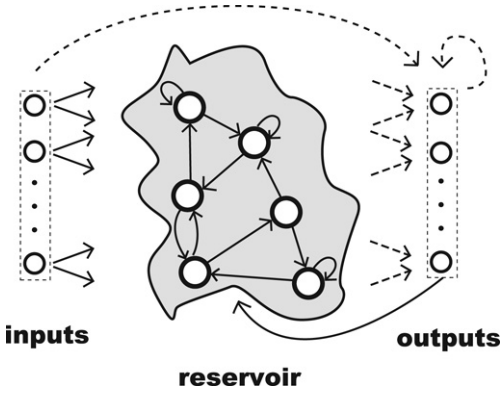

Fig. 1. Reservoir Computing network. The reservoir is a dynamical system of recurrent nodes. Solid lines represent connections which are fixed. Dashed lines are the connections which can be trained.

However, we discard the readout's output feedback to the reservoir because the problems in this work do not require a very long-term memory. We also add a leak rate α as in Schrauwen, Defour, Verstraeten, and Van (2007) to the state update equation in order to make the reservoir timescale more flexible in matching the timescale of the input. The changes can be seen in (3).

$$\mathbf{x}(t+1) = f\left((1-\alpha)\mathbf{x}(t) + \alpha(\mathbf{W}^{\text{res}}_{\text{res}}\mathbf{x}(t) + \mathbf{W}^{\text{res}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{res}}_{\text{bias}})\right). \quad (3)$$

The output calculation gets simpler once we do not use the direct connections from input to output neither the connections from output to output:

$$\mathbf{y}(t+1) = \mathbf{W}^{\text{out}}_{\text{res}}\mathbf{x}(t+1) + \mathbf{W}^{\text{out}}_{\text{bias}}. \quad (4)$$

The leak rate α can effectively tune the dynamics of the reservoir. If the leak rate is chosen correctly, the reservoir dynamics can be adjusted to match the timescale of the input flow, making it possible to achieve enhanced performance (this can also be achieved by resampling the input (Antonelo et al., 2007b; Jaeger, Lukosevicius, & Popovici, 2007)). The leak rate can be chosen empirically or alternatively by a parameter search over a set of leak rates (parameter optimization). In this work, some experiments use 3 pools of neurons in the reservoir with distinct leak rates to achieve better performance. The method used for choosing the leak rate(s) is presented in the following sections depending on the considered task. Further investigation about timescales in reservoirs and leaky integrator neurons can be found in Jaeger et al. (2007); Schrauwen et al. (2007).

Each element of the connection matrix $\mathbf{W}^{\text{res}}_{\text{res}}$ is drawn from a normal distribution with mean 0 and variance 1. The randomly created $\mathbf{W}^{\text{res}}_{\text{res}}$ matrix is rescaled such that the system is stable and the reservoir has the echo state property (i.e., it has a fading memory (Jaeger, 2001b)). This can be accomplished by rescaling the matrix so that the spectral radius $|\lambda_{\max}|$ (the largest absolute eigenvalue) of the linearized system is smaller than one (Jaeger, 2001b). Standard settings of $|\lambda_{\max}|$ lie in a range between 0.7 and 0.98 (Jaeger, 2002). In this work we scale all reservoirs to a spectral radius of $|\lambda_{\max}| = 0.9$ which is an arbitrarily chosen value (the optimization of the spectral radius for each experiment was not necessary because the changes in performance resulting from using distinct spectral radius in the range [0.7, 0.98] are not clearly visible).

[1] This paper is an extended version of Antonelo, Schrauwen, Dutoit, Stroobandt, and Nuttin (2007b) which was presented at ICANN 2007.