Neural Networks 80 (2016) 67–78

Contents lists available at ScienceDirect

Neural Networks

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

Synthesis of recurrent neural networks for dynamical system simulation

Adam P. Trischler[a,*], Gabriele M.T. D'Eleuterio[b]

[a] Maluuba Research, 2000 Peel Street, Montreal, Canada

[b] University of Toronto Institute for Aerospace Studies, 4925 Dufferin Street, Toronto, Canada

ARTICLE INFO

Article history:
Received 10 August 2015
Revised and accepted 5 April 2016
Available online 20 April 2016

Keywords:
Recurrent neural network
Dynamical system
Approximation
Attractor
Chaos
Nonautonomous system

ABSTRACT

We review several of the most widely used techniques for training recurrent neural networks to approximate dynamical systems, then describe a novel algorithm for this task. The algorithm is based on an earlier theoretical result that guarantees the quality of the network approximation. We show that a feedforward neural network can be trained on the vector-field representation of a given dynamical system using backpropagation, then recast it as a recurrent network that replicates the original system's dynamics. After detailing this algorithm and its relation to earlier approaches, we present numerical examples that demonstrate its capabilities. One of the distinguishing features of our approach is that both the original dynamical systems and the recurrent networks that simulate them operate in continuous time.

1. Introduction and survey

Recurrent neural networks (RNNs) represent a large class of computational models designed in analogy to the brain. What distinguishes them from the better known feedforward neural networks is the existence of closed cycles in the connection topology; as a consequence of these cycles, RNNs may exhibit self-sustained dynamics in the absence of any input (Lukoševičius & Jaeger, 2009). Mathematically, RNNs are dynamical systems.

Not only is the brain characterized by massively recurrent connectivity, but dynamical systems themselves are now a mainstay of computational neuroscience. Persistent activity in biological neural networks is posited to result from dynamical attractors in neural state space (Amit, 1992), and dynamical computation underlies a variety of models for information processing and memory function in the brain (Afraimovich, Rabinovich, & Varona, 2004; Eliasmith & Anderson, 2004; Kaneko & Tsuda, 2003).

1.1. RNN training techniques

Motivated by these facts and other interests, various researchers have developed techniques by which recurrent networks can be trained to approximate dynamical systems.

One set of such techniques takes a discrete-time approach: dynamical systems, whether continuous or discrete in time, are approximated with discrete-time RNNs. This set includes backpropagation through time (Werbos, 1990), real-time recurrent learning (Williams & Zipser, 1989), the extended Kalman filter (Feldkamp, Prokhorov, Eagen, & Yuan, 1998), reservoir computing (Jaeger & Haas, 2004), and phase-space learning (Tsung & Cottrell, 1995).

Backpropagation through time (BPTT) adapts the standard backpropagation algorithm (Rumelhart, Hinton, & Williams, 1988) for feedforward networks to recurrent networks. BPTT works by "unfolding" a network in time: identical copies of the RNN are stacked in layers, and connections within the network are redirected to obtain connections between subsequent copies. Each layer represents the same network at a different step in time. The result of this unfolding is a feedforward network amenable to standard backpropagation. BPTT is probably the most widely used method for RNN training and can be made to perform very well with various modifications (*e.g.*, stochastic sample selection (Bottou, 2010) and the addition of measures for computational efficiency and stability like momentum (Sutskever, Martens, Dahl, & Hinton, 2013)). One common issue is that error gradients shrink or expand exponentially over time because they are multiplied repeatedly by copies of the same weight matrix. This is referred to as the problem of vanishing and exploding gradients. As a consequence, long-term memory effects are quite difficult to train. This issue can be overcome with second-order optimization techniques that use curvature information, such

* Corresponding author.
E-mail addresses: adam.trischler@maluuba.com (A.P. Trischler), gabriele.deleuterio@utoronto.ca (G.M.T. D'Eleuterio).

http://dx.doi.org/10.1016/j.neunet.2016.04.001
0893-6080/© 2016 Elsevier Ltd. All rights reserved.

as the popular Hessian-free (Martens & Sutskever, 2011), or by modifying the network architecture. The Long Short-Term Memory (LSTM) network, for example, features multiplicative gates and linear units whose gradients are unity so that their values can be remembered over many timesteps (Hochreiter & Schmidhuber, 1997).

Real-time recurrent learning (RTRL) computes a RNN's exact error gradient at every discrete timestep (Williams & Zipser, 1989). Differentiating the network equations by the weights yields a discrete-time linear dynamical system with time-varying coefficients, where the resulting partial derivatives of the network state are the dynamical variables. Iterating through time for the error gradients simultaneously with the network dynamics provides the required weight adjustment. Although RTRL is mathematically transparent, the computational cost for each update step is $O(n^4)$, where n is the number of neurons. This renders the scheme practicable only for very small networks on the order of about ten neurons (Lukoševičius & Jaeger, 2009).

The extended Kalman filter (EKF) is a state estimation technique for nonlinear systems derived by linearizing the original Kalman filter about the current state estimate. It is a second-order gradient-descent algorithm that uses curvature information from the squared error surface. The pioneering work in adapting the EKF to RNN training was done by Feldkamp et al. (1998) within the domain of system identification. Here, the unknown network weights are interpreted as the state of a dynamical system, and the desired dynamical trajectory as a measurement of that state.

In reservoir computing (RC), a randomly recurrently connected sea of neuron units, referred to as the *reservoir*, feeds forward to a set of output units and may be driven by an input signal. The reservoir functions as a dynamical system and must exhibit what is called the *echo state property*. This relates asymptotic properties of the reservoir dynamics to the driving signal. Intuitively speaking, the echo state property means that the reservoir asymptotically eliminates any information from initial conditions (Lukoševičius & Jaeger, 2009). The output units in RC may be made to approximate prescribed dynamical trajectories by training only the output weights that feed them (typically by a linear regression process); the random recurrent connections within the reservoir remain fixed. An issue with RC is that the random reservoir is, to date, poorly understood from a theoretical standpoint (Lukoševičius & Jaeger, 2009). Networks generated by our procedure in fact share a structural relation to RC systems, which we shall discuss.

The phase-space learning method (PSL) of Tsung and Cottrell (1995) takes a vector-field approach to RNN training that is similar to our algorithm from a high-level perspective. PSL consists of (1) embedding a dynamical trajectory to recover its phase-space structure (this is an application of Takens (1981) theorem); (2) generating local approximations of the underlying vector field about the given trajectory; and (3) approximating the vector field with a feedforward network. This method transforms the recurrent network problem into a feedforward one, as does ours. However, the networks generated by our approach and that of Tsung and Cottrell (1995) differ significantly. PSL networks are discrete in time, and they also remain essentially feedforward even after training. They do not contain a recurrently connected reservoir of hidden neurons; recurrence only arises in piping network outputs back to the input neurons.

As noted, these techniques train discrete-time RNNs.[1,2] Discrete-time networks are simpler conceptually and easier to train, and of course numerical simulations are carried out on discrete-time digital computers. However, discrete-time systems are of less interest to neuroscience than their continuous-time counterparts since the brain is inherently continuous.[3] It is also well known that finite-difference equations can behave very distinctly from ODEs in some respects: *e.g.*, chaos can occur in one-dimensional finite-difference systems, while three dimensions are required for chaos in continuous systems. This, along with the desire to apply tools of functional analysis, motivated us to set our RNNs within a continuous-time formalism. In our work, both the original dynamical system and its recurrent-network approximation are modeled with ordinary differential equations (ODEs).

To our knowledge, the most widely used technique for training continuous-time recurrent networks is the Neural Engineering Framework (NEF) of Eliasmith (2005) and Eliasmith and Anderson (2004). In this framework, the activities of a population of neurons encode some input vector. Given this encoding, the original stimulus vector can be approximately recovered by decoding the population activities; this is accomplished by taking the inner product of the activities with a set of decoding vectors. Decoding vectors are determined by a least-squares method. Similarly, approximate transformations of the original stimulus vectors can be defined using a modified set of decoding vectors, referred to as transformational decoders.

An interesting aspect of the NEF that bears some relation to the RC approach is that the linear encoder weights are set randomly; they are not trained. Contrary to typical backpropagation, error is evaluated only at the output, based on the linear decoders; it is not propagated back to assign credit or blame to the linear encoders. In some ways this is advantageous: because errors need not propagate through the neuron activation function, this function need not be differentiable. In the NEF, both the activation function and the transfer functions of its neurons are generic. For example, the activation function can be continuous (sigmoidal) or spiking (leaky integrate-and-fire).

There exist strong analogies between the NEF and our RNN training technique, although the two methods spring from different formalisms. We take as our starting point a theorem from the continuous-time RNN literature that is not employed by the NEF authors. Through it, we arrive at a procedure and a class of networks which can be viewed as a special case of the NEF. However, by starting from this theorem and tracing a different route to the end procedure, we show that the special case of networks we utilize is governed by theoretical bounds on its performance. To our knowledge, this theoretical support for (a special case of) the NEF was not known previously. We will characterize the relation to the NEF mathematically in Section 4, after detailing our procedure in Section 2.

1.2. The proposed RNN training procedure

Our algorithm for training RNNs to approximate prescribed dynamical systems is based on a theoretical result of Funahashi and Nakamura (1993). Theorem 1 therein states that any dynamical system can be "approximated to arbitrary accuracy" by a recurrent neural network. This theorem is an existence result; here, we will present a constructive algorithm for obtaining the approximating networks that Funahashi and Nakamura (1993) theorize.

In our approach, a feedforward neural network is first trained on the vector-field representation of a given dynamical system using standard backpropagation techniques. Then the trained

[1] A continuous-time EKF exists, but its application to the RNN training problem requires continuous-time derivatives for the Jacobian.

[2] Hermans and Schrauwen (2010) adapted reservoir computing to continuous time.

[3] Although the brain operates on spike trains, the spikes themselves are best modeled by ODEs, such as the Hodgkin–Huxley model.