Sequence Processing with Recurrent Neural Networks

Chun-Cheng Peng
University of London, UK

George D. Magoulas
University of London, UK

INTRODUCTION

Sequence processing involves several tasks such as clustering, classification, prediction, and transduction of sequential data which can be symbolic, non-symbolic or mixed. Examples of symbolic data patterns occur in modelling natural (human) language, while the prediction of water level of River Thames is an example of processing non-symbolic data. If the content of a sequence will be varying through different time steps, the sequence is called *temporal* or *time-series*. In general, a temporal sequence consists of nominal symbols from a particular alphabet, while a time-series sequence deals with continuous, real-valued elements (Antunes & Oliverira, 2001). Processing both these sequences mainly consists of applying the current known patterns to produce or predict the future ones, while a major difficulty is that the range of data dependencies is usually unknown. Therefore, an intelligent system with memorising capability is crucial for effective sequence processing and modelling.

A recurrent neural network (RNN) is an artificial neural network in which self-loop and backward connections between nodes are allowed (Lin & Lee 1996; Schalkoff, 1997). Comparing to feedforward neural networks, RNNs are well-known for their power to memorise time dependencies and model nonlinear systems. RNNs can be trained from examples to map input sequences to output sequences and in principle they can implement any kind of sequential behaviour. They are biologically more plausible and computationally more powerful than other modelling approaches, such as Hidden Markov Models (HMMs), which have non-continuous internal states, feedforward neural networks and Support Vector Machines (SVMs), which do not have internal states at all.

In this article, we review RNN architectures and we discuss the challenges involved in training RNNs for sequence processing. We provide a review of learning algorithms for RNNs and discuss future trends in this area.

BACKGROUND

One of the first RNNs was the *avalanche network* developed by Grossberg (1969) for learning and processing an arbitrary spatiotemporal pattern. Jordan's *sequential network* (Jordan, 1986) and Elman's *simple recurrent network* (Elman, 1990) were proposed later.

The first RNNs did not work very well in practical applications, and their operation was poorly understood. However, several variants of these models were developed for real-world applications, such as robotics, speech recognition, music composition, vision, and their potential for solving real-world problems has motivated a lot of research in the area of RNNs.

Current research in RNNs has overcome some of the major drawbacks of the first models. This progress has come in the form of new architectures and learning algorithms, and has led in a better understanding of the RNNs' behaviour.

ARCHITECTURES OF RECURRENT NETWORKS

In the literature, several classification schemes have been proposed to organise RNN architectures starting from different principles for the classification, i.e. some consider the loops of nodes in the hidden layers, while others take the types of output into account. For example, they can be organised into *canonical* RNNs and *dynamic MLPs* (Tsoi, 1998a); *autonomous converging* and *non-autonomous non-converging* (Bengio et al., 1993); *locally* (receiving feedback(s) from the same or directly connected layer), *output feedback*, and *fully connected* (i.e. all nodes are capable to receive and transfer feedback signals to the other nodes, even within different layers) RNNs (dos Santos & Zuben, 2000); *binary* and *analog* RNNs (Orponen, 2000).

From mathematical point of view (Kremer, 2001), assuming that *y* and *z* are respectively the response of the output layer and the output of the hidden layer, a static feedforward neural network can be formulated as follows:

$$y=\phi\left(\mathbf{W}^{\mathrm{II}}z+b^{\mathrm{II}}\right) \tag{1}$$

$$z=\phi\left(\mathbf{W}^{\mathrm{I}}x+b^{\mathrm{I}}\right), \tag{2}$$

where $f(\bullet)$ denotes nonlinear activation function, $\mathbf{W}^{\mathrm{I}}$ and $\mathbf{W}^{\mathrm{II}}$ the weights of the hidden layer and the output layer, *x* the input vector, and *b* the biases. This general form could be easily transformed to describe a Feed-Forward Time-Delayed (FFTD) RNN by substituting the following delayed equations with time index *t*,

$$y(t)=\phi\left(\mathbf{W}^{\mathrm{II}}z(t)+b^{\mathrm{II}}\right) \tag{3}$$

$$z(t)=\phi\left(\mathbf{W}^{\mathrm{I}}s(t)+b^{\mathrm{I}}\right) \tag{4}$$

$$s(t)=\left\{x(t)\oplus x(t-1)\oplus\cdots\oplus x(t-d)\right\}, \tag{5}$$

where *s(t)* denotes the state vector at time *t*, $\oplus$ the Cartesian product, *d* the number of delays. By adding a feedback connection from the hidden layer to the delay unit then Eq. (4) can be stated as

$$z(t)=\phi\left(\Lambda z(t-1)+\mathbf{W}^{\mathrm{I}}x(t)+b^{\mathrm{I}}\right), \tag{6}$$

where Λ is a diagonal matrix, which describes an Elman-type RNN.

For the Nonlinear Autoregressive Network with Exogenous Inputs (NARX) the state is described as

$$s(t)=\left\{x(t)\oplus x(t-1)\oplus\cdots\oplus x(t-d+1)\right\}\oplus$$
$$\left\{y(t-1)\oplus y(t-2)\oplus\cdots\oplus y(t-m)\right\}, \tag{7}$$

where *m* is the number of output feedbacks. The formulations of a fully RNN can also be derived by combining Eqs. (3) and (7) with the following one:

$$z(t)=\phi\left(\Lambda z(t-1)+\mathbf{W}^{\mathrm{I}}s(t)+\mathbf{W}^{\mathrm{I}}x(t)+b^{\mathrm{I}}\right). \tag{8}$$

Table 1 provides an overview of the various architectures and of the relevant literature.

LEARNING ALGORITHMS FOR RECURRENT NETWORKS

With regards to training RNNs and storing information in their internal representations, Gradient Descent-based learning algorithms (GD) are the most commonly applied methods, even though it has been claimed that GD has some drawbacks (Bengio et al., 1994). Firstly, when the delays or recursive connections are very deep, i.e. when long-term memory is required, the backpropagation error may be vanished and the training process could become inefficiently. Secondly, the most common way to apply GD algorithms into RNN is to unfold the recursive layers and train the whole network as a feedforward network. Another drawback is that the generalisation is highly affected by the samples in the training dataset. In temporal processing it is difficult to extract or prepare negative samples from a given

Table 1. Classification summary of RNNs

Recurrence	Globally	Locally	Fully	Partially
Reference	Brouwer (2005) Kremer & Kolen (2000) Puskorius & Feldkamp (1994)	Assaad et al. (2005) Boné & Cardot (2005) Sperduti & Starita (1997) Temurtasm et al. (2004) Tiňo & Mills (2005)	Pedersen (1997)	All, except Pedersen (1997)
Equations	(3),(4),(7)	(3),(6)	(3),(7),(8)	Globally/Locally