Engineering Applications of Artificial Intelligence 74 (2018) 312–321

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Application of new training methods for neural model reference control

Amir H. Jafari [a], Martin T. Hagan [b,*]

[a] George Washington University, Washington, DC, United States
[b] Oklahoma State University, Stillwater, OK, United States

ARTICLE INFO

Keywords:
Recurrent neural network
Model reference control
SOM
Novelty sampling
Magnetic levitation

ABSTRACT

In this paper, we introduce new, more efficient, methods for training recurrent neural networks (RNNs) for system identification and Model Reference Control (MRC). These methods are based on a new understanding of the error surfaces of RNNs that has been developed in recent years. These error surfaces contain spurious valleys that disrupt the search for global minima. The spurious valleys are caused by instabilities in the networks, which become more pronounced with increased prediction horizons. The new methods described in this paper increase the prediction horizons in a principled way that enables the search algorithms to avoid the spurious valleys.

The work also presents a novelty sampling method for collecting new data wisely. A clustering method determines when an RNN is extrapolating, which occurs when the RNN operates outside the region spanned by the training set, where adequate performance cannot be guaranteed. The new method presented in this paper uses a clustering method for extrapolation detection, and then the novel data is added to the original training set. The network performance is improved when additional training is performed with the augmented data set.

The new techniques are applied to the model reference control of a magnetic levitation system. The techniques are tested on both simulated and experimental versions of the system.

1. Introduction

RNNs are good candidates to represent nonlinear dynamic systems, as demonstrated by their application in many areas, such as system identification and control (Hagan and Demuth, 1999), long term predictions of chemical processes (Su et al., 1992), financial analysis of multiple stock markets (Roman and Jameel, 1996) and phasor detection (Kamwa et al., 1996). However, it is well known that RNNs are difficult to train (Atiya and Parlos, 2000; Gori et al., 2010). (It should be noted that we are using RNN to designate any discrete time neural network with one or more feedback connections that contain one or more delays.)

Two of the proposed reasons for the difficulties in RNN training are the problems of vanishing and exploding gradients (Bengio et al., 1994; Pascanu et al., 2012). When a recurrent network is stable (or, more precisely, a given trajectory is stable), effects of inputs to the network diminish as they move forward in time, and, consequently, the gradients of performance with respect to inputs and weights diminish as they are propagated backward in time. This is referred to as the vanishing gradient problem, which makes it difficult to learn long-term dependencies between inputs and outputs, if the initial weights of the network produce a stable response. This is generally not as important an issue in nonlinear system identification as it is, for example, in natural language processing, where the meaning of a word might be more accurately identified by context in the previous paragraph.

The exploding gradient problem is caused by the complement of the vanishing gradient problem. If the network is unstable (a given trajectory is unstable), the effects of inputs to the network grow as they propagate forward in time. The gradients will therefore grow as they move backward in time. This is connected to the existence of spurious valleys in the error surfaces of recurrent networks (Jesus et al., 2001; Horn et al., 2009; Phan and Hagan, 2013a). These valleys are not associated with the true minimum of the surface, or to the problem the RNN is trying to solve. They are strongly dependent on the input sequence in the training data. (If the input sequence changes, even though the system being modeled stays the same, the valleys will move significantly.) Any batch search algorithm is very likely to be trapped in these spurious valleys. These valleys occur in regions of instability, as was shown in Horn et al. (2009) and Phan and Hagan (2013a).

Alternate training methods have been developed to mitigate the effects of these spurious valleys (Jesus et al., 2001; Phan and Hagan, 2013b). Because the spurious valleys depend so strongly on the input sequence, one alternate method is to divide the data into multiple subsequences, or minibatches. The subsequences can be alternated during training, which will move the valleys and prevent the algorithm

* Corresponding author.
E-mail address: mhagan@okstate.edu (M.T. Hagan).

https://doi.org/10.1016/j.engappai.2018.07.005
Received 8 August 2017; Received in revised form 16 June 2018; Accepted 14 July 2018

from becoming trapped (Jesus et al., 2001). Recently, Phan and Hagan (2013b) demonstrated a modified procedure, in which the error gradient associated with each subsequence is monitored during training. Large gradient magnitudes indicate that the training algorithm is located within a spurious valley for those subsequences, and so those subsequences can be removed temporarily from the training process.

Another technique that was introduced in Phan and Hagan (2013b) was to increase the prediction horizon gradually during the training process. The initial training segment used a one-step-ahead prediction. This was increased at each training segment, until the prediction horizon during the final training segment covered the full length of the original sequences. This process can require long training times, if the prediction horizon is increased too slowly, but will fail to converge if the prediction horizon is increased too quickly. In this paper, we are introducing a method that searches for an optimal horizon step at each training segment (Jafari and Hagan, 2015). We demonstrate the process on a practical system identification problem.

Even after a recurrent network has been successfully trained, satisfactory performance can only be ensured if the network inputs are similar to those in the training set. This is also true for feedforward networks, but extrapolation is a more urgent problem for recurrent networks, where, because of feedback connections, responses can become unstable when network inputs (including feedback signals) fall outside the training set. The process of detecting network inputs that are outside the training set is called novelty detection (Pimentel et al., 2014). In this paper, we are proposing a type of novelty detection based on clustering. We demonstrate that the proposed technique is able to detect incipient network failures and instabilities well before they occur.

The clustering method we use for novelty detection is the Self-Organizing Map (SOM) (Kohonen, 1990). This is a topology preserving network, in that neurons within the network have neighbor relationships that are preserved by the training process. The idea will be to train the SOM on composite vectors that contain the inputs to the network augmented with the target network output. The SOM will divide the training data into clusters, so that each input/target pair will be near one cluster center. When a new data point appears that is not near any cluster center, extrapolation will be identified.

We are also going to use the SOM to collect additional data in order to improve the training procedure. It is unlikely that the original data set will effectively cover the full range of conditions where the network will be used. The RNN will extrapolate when network inputs fall outside the space spanned by the training data set. We are going to collect additional training data when the SOM indicates extrapolation. Then, we will retrain the RNN network with the new data combined with the initial training data set. This procedure is known as *novelty sampling* (Raff et al., 2005). This will be done in phases until no novel conditions are detected after many additional tests.

Some of these ideas were first presented in Jafari and Hagan (2015), but they will be expanded in three ways in this paper. First, we introduce the use of the SOM for novelty sampling, which is used to select new data for the training process. Secondly, in addition to the modeling of dynamic systems, we also apply novelty sampling, and the other new recurrent network training methods, to the training of a model reference control (MRC) system, represented as a larger RNN that contains both the plant model and a neurocontroller. Finally, all of the new procedures will be tested on a magnetic levitation system — both simulated and experimental versions.

This paper is organized as follows. In Section 2, we explain the modified training algorithm, in which subsequences are removed from the training set when their gradient becomes large. Next, in Section 3, we introduce the new method for determining the optimal prediction horizon. In Section 4, we describe the recurrent network modeling of a magnetic levitation system. In Section 5 we enhance the models by adding new data by novelty detection. Next, in Section 6, we describe how the new procedures can be used to develop model reference controllers. In Section 7, we perform experimental verification of the methods on an experimental prototype maglev system.

2. Modified training for recurrent neural networks

In order to train an RNN to approximate a dynamic system, we need appropriate data. Unlike static networks, where each input/target pair stands on its own, RNN data must consist of ordered sequences of inputs and target outputs. When training recurrent networks, the length of the sequence determines the *prediction horizon*. If the length of a sequence increases by one, then the prediction horizon increases by one. For example, if the training sequences have a length of 5 time steps, and the maximum number of delays in the network is 2, then training the closed-loop network means that we are doing 3-step-ahead predictions.

For the method introduced in Phan and Hagan (2013b), training begins with short prediction horizons, and then the prediction horizons increase as training proceeds. Short prediction horizons require short sequences, so the original training sequences are divided into multiple *subsequences*. The network is trained for multiple iterations at a given prediction horizon. We call this a *training segment*. After the completion of a training segment, the prediction horizon is increased by the *horizon step*, which requires that the original training sequences be subdivided again.

The key concept introduced in Phan and Hagan (2013b) was that, because the spurious valleys are caused by the input sequence, each training subsequence will have a different set of valleys. If training becomes trapped in a valley, the sequence that owns that valley could then be removed from the training set. In order to determine which sequence to remove, the individual training gradients for each sequence are computed, and the sequence with the largest gradient is removed, since gradients will be highest inside the spurious valleys. To determine when the training algorithm entered a valley, Phan and Hagan (2013b) proposed using a feature of the Levenberg–Marquardt (LM) training algorithm (Hagan and Menhaj, 1994), as described below.

The LM update rule for weights $\mathbf{x}_k$ at the kth iteration is

$$\Delta\mathbf{x}_k = -\left[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k) \tag{1}$$

where $\mathbf{e}$ is the network error and $\mathbf{J}$ is the Jacobian matrix of the network errors with respect to the weights. $\mathbf{J}$ can be computed using backpropagation. For RNNs, we need to use dynamic backpropagation. Jacobian calculations for a general dynamic network can be found in Jesus and Hagan (2007).

The important feature of the LM algorithm is that as μ_k becomes large it reverts to steepest descent with a small learning rate, which guarantees that the performance function $\mathbf{F}(\mathbf{x})$ (typically mean square error) must decrease if μ is made large enough. The algorithm starts with a small μ_k, and if $\mathbf{F}(\mathbf{x})$ does not decrease at any iteration, the algorithm increases μ_k by factor of 10. If $\mathbf{F}(\mathbf{x})$ decreases, μ_k is reduced by a factor of 10, because the algorithm converges faster in the Gauss–Newton mode (μ_k small). The algorithm is stopped, if μ_k becomes too large. This indicates that $\mathbf{F}(\mathbf{x})$ does not decrease, even when a very small step in the steepest descent direction is taken. This indicates that the algorithm is stuck in one of the spurious valleys, which tend to be steep and narrow.

The other approach suggested in Phan and Hagan (2013b) was to start the first training segment with one-step-ahead predictions and then to increase the prediction horizon gradually for each successive training segment.

The training procedure from Phan and Hagan (2013b) can be summarized as follows (*Method 1*):

1. In the first training segment, use open-loop training (one-step-ahead predictions). All training segments involve *maxit* iterations of the training algorithm.
2. Closed-loop training with increasing prediction horizon: Do k-step-ahead prediction ($k \geq 2$). This includes segmentation of the original long sequences into smaller subsequences.
3. At each iteration of the LM algorithm, if μ reaches μ_{max}, remove the subsequence with largest gradient. If $\mathbf{F}(\mathbf{x})$ does not decrease, keep removing the subsequence with next largest gradient until