Performance-guided Neural Network for Self-Organising Network Management

Sin Wee Lee, Dominic Palmer-Brown, Jonathan Tepper and Christopher Roadknight↕

↑Computational Intelligence Research Group, Leeds Metropolitan University, ‡The Nottingham Trent University, ↕BTexact Research Laboratories

Abstract: A neural network architecture is introduced for real-time learning of input sequences using external performance feedback. Some aspects of Adaptive Resonance Theory (ART) networks [1] are applied because they are able to function in a fast real-time adaptive active network environment where user requests and new proxylets (services) are constantly being introduced over time [2,3]. The architecture learns, self-organises and self-stabilises in response to user requests, mapping the requests according to the types of proxylets available. However, in order make the neural networks respond to performance feedback, we introduce a modification to the original ART1 network in the form of the 'snap-drift' algorithm, that uses fast convergent, minimalist learning (snap) when the overall network performance is poor, and slow learning (drift towards user request input pattern) when the performance is good. Preliminary simulations evaluate the two-tiered architecture using a simple operating environment consisting of simulated training and test data.

1. Introduction

An Application Layer Active Network (ALAN) [3] was first introduced to enable the users to supply JAVA based active service code known as proxylets. It runs on an edge system (Execution Environment for Proxylets – EEP) provided by the network operator. The purpose of the architecture is to enhance the communication between servers and clients using the EEPs that are located at optimal points of the end-to-end path between the server and the clients without the requirements in dealing with the current system architecture and equipments. This approach relies on the redirecting of selected request packets into the EEP, where the appropriate proxylets can be executed to modify the packets contents without impacting on the router's performance and thus no any additional standardisations are required.

However, since ALAN provides active services that are unbounded in both scale and function, and with an enormous range of services being developed and evolved at an unprecedented rate, it is necessary to combine the active services with a highly automated and adaptive management [4,5,6] and control solution. In this paper, we describe an artificial neural network (ANN) based approach for the adaptive management of ALAN. The approach involves modifying the properties of ART networks [1], in order to develop a novel ANN-based solution to adaptively select the appropriate proxylets available to the local EEPs, in response to continually changing input requests. A novel reinforcement-based learning algorithm is developed to improve the performance of the network under different commercial circumstances.

The remainder of this paper is structured as follows: Section 2 describes the architecture and the learning principles of the proposed architecture. Experiments that we carried out on the novel learning algorithm on the Distributed ART1 module (see Figure 1) are presented in Section 3, and conclusions and future work are discussed in Section 4.

2. The Proposed Architecture

The tiered-ART network we propose is a modular, multi-layered architecture that can be used to select available proxylets (services) in the networks, in response to continually changing input requests while trying to improve the overall performance of the network. It is composed of 3 modules, a Distributed ART1 network, an ART1 network and a Kohonen Feature Map. The $F1_1 \leftrightarrow F2_1$ and $F1_2 \leftrightarrow F2_2$ connections of the Distributed ART1 network are weighted bottom-up and top-down connections that can be modified during the learning stage. The $F0_1 \rightarrow F1_1$ connections and the connections between the two ART modules connected through $F2_1 \rightarrow F1_2$ are unidirectional, one to one and non-modifiable. Each of the $F2_2$ nodes are hard-wired onto a specific pre-trained region of the Kohonen Feature map where similar available proxylets are spatially organised on the 2-D map according to their featural similarity (similar requests will appear in neighbouring map regions). The working of the network can be summarised as follows:

Upon the presentation of an input pattern at the input layer $F0_1$, the Distributed ART will learn to group the input patterns according to their general features using the novel learning principles, known as 'snap-drift' proposed in the next section. If no existing matching prototype is found, i.e. when the stored pattern templates is not a good match for the input, then the winning $F2_1$ node is reset and another $F2_1$ node is selected, whose pattern template will be matched against the input, and so on. When no corresponding output category can be found, the network considers the input as novel, and generates a new output category that learns the current input pattern.

The top three $F2_1$ nodes are used as the input for the ART1 module where an appropriate proxylet type is selected accordingly. For the purpose of selecting the required proxylet, the proxylet type information indicated by the ART1 acts as a reference to pre-trained locations on the Kohonen-Feature Map, which represent specific proxylets. If the proxylet is unavailable, one of its neighbours is selected (the most similar alternative available).

In order to guide the network's learning, there is an external indicator of performance of the system as a whole. A general performance measure is used because there are no specific performance measures (or external feedback) in response to each *individual* network decision. This measure is used to enable the reselection of a proxylet types to occur in order to improve system performance.

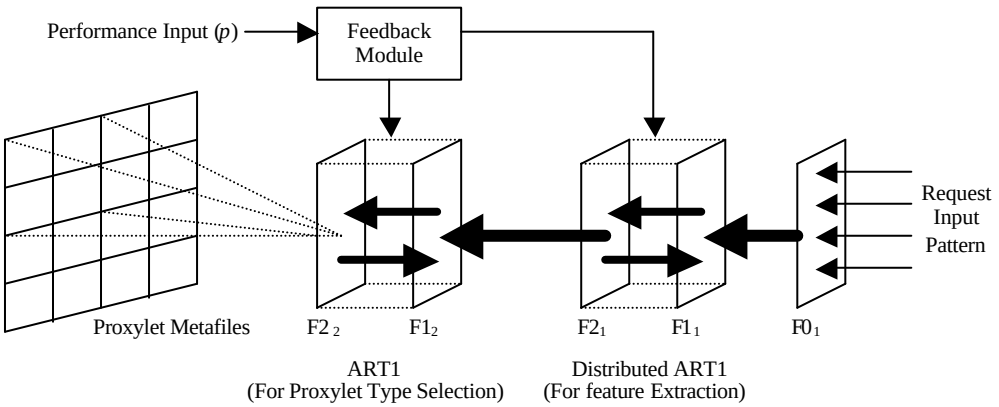

Figure 1: The Proposed Architecture

3. Learning Principles

The learning process of the Tiered-ART network is described as follows. Note that the novel learning principles described here are applied to the both ART modules in Figure 1.

3.1 Top-down Learning

The top-down learning of the network can be illustrated using the following equation:

$$w_{iJ}^{(new)} = (1-p)\,(I \cap w_{iJ}^{(old)}) + p\,(w_{iJ}^{(old)} + \beta\,(I - w_{iJ}^{(old)})) \qquad (1)$$

where

$w_{ij}^{(old)}$ = the top-down weight vectors at the start of the input presentation
p = performance parameter
I = binary input vectors
β = 'drift' constant

Initially, all the w_{ij} are set randomly in the range (0.99,1)

$$w_{ij}(0) = (0.99, 1) \qquad (2)$$

Thus with the learning, a simple distributed affect will be generated at the output layer of the network, with different patterns tending to give rise to different activations across F2.

By substituting p in equation (1) with 0 for very poor performance, equation (1) simplifies to:

$$w_{iJ}^{(new)} = (I \cap w_{iJ}^{(old)}) \qquad (3)$$

Thus fast learning is invoked, causing the top-down weights to reach their new asymptote on each input presentation

$$w_J \rightarrow I \cap w_J^{(old)} \qquad (4)$$

In contrast, for excellent performance where $p = 1$, equation (1) can be simplifies to:

$$w_{iJ}^{(new)} = (w_{iJ}^{(old)} + \beta(I - w_{ij}^{(old)})) \quad (5)$$

Thus, a simple form of clustering occurs at a speed determined by β.

It is assumed that there is a considerable interval between updates of *p* during which time new previously unseen requests are likely to appear. Equation (5), or indeed equation (1) whenever performance is not perfect, enables the top-down weights to drift towards the input patterns. New category node selection may occur for one of two reasons: as a result of the drift itself, or as a result of the drift enabling a further snap to occur (since drift has moved away from convergence) if performance *p* goes down. Essentially, the principle is that drift, by itself, will only results in slow (depending on β) reselection over time, thus keeping the network up-to-date without a radical set of reselections for exiting patterns. By contrast, snapping results in rapid reselection of a proportion of patterns to quickly respond to a significantly changed situation, in terms of the input vectors (requests) and/or of the environment which may require the same requests to be treated differently. For example, bandwidth and/or completion time may become critical at certain times.

In this simulation, β is set to 0.5. This will provide the control for the drifting of the weights vector where eventually

$$w_J \rightarrow I \quad (6)$$

With alternate episodes of $p = 0$ and $p = 1$, the characteristics of the learning of the network will be the joint effects of the equation (3) and (5). This joint effect can enable the network to learn using fast and convergent, snap learning when the performance is poor, yet be able to drift towards the input patterns when the performance is good.

3.2 Bottom-up Learning

In the simulations, the bottom-up weights w_{ji} are assigned initial values corresponding to the initial values of the top-down weights w_{ij}. This is accomplished by equation (7):

$$w_{ji}(0) = \frac{w_{ij}(0)}{1 + N} \quad (7)$$

where N = number of input nodes.

By selecting this small initial value of w_{ji}, the network is more likely to select a previously learned category node that to some extent matches the input vector rather than an uncommitted node.
For learning,

$$w_{jI}^{(new)} = \frac{I \cap w_{jI}^{(old)}}{| I \cap w_{jI}^{(old)} |} \quad (8)$$

This is fast, convergent learning.

4. Network Experiments

This section presents some conclusions drawn from the results for a number of simulations performed on the Distributed ART1 (see Figure 1) to emulate possible environmental conditions and thus evaluate the behaviour and performance of the system. Three simulations were performed: (i) long periods without external performance feedback; (ii) long periods where there is a constant supply of performance feedback, and (iii) long periods of feedback being available on alternate time steps.

These simulations illustrate the influence of performances feedback toward the on-line learning of the network. The test patterns consist of 100 input vectors. Each test pattern characterizes the features/properties of a realistic network request, such as bandwidth, time, file size, loss and completion guarantee. These test patterns were presented in random order for 10 epochs where the performance, *p*, is set to the different possible environmental conditions (long period of poor performance, long period of good performance and alternate performance feedback). This on-line random presentation of test patterns simulates the possible real world scenario where the order of patterns presented is random so that a given network request might be repeatedly encountered while others are not used at all. Furthermore, no input pattern is removed from the list of possible requests after it is encountered.

The first two simulations show the plausibility of the 'snap-drift' algorithm proposed in this paper. In these two simulations, the network has been put to test with 100 input patterns. In the first simulation, fast learning has been invoked due to the absence of performance feedback or long period of poor performance. The network learned to classify the input patterns according to their general features rapidly; it stabilised within 4 epochs of the training simulations. Standard ART 1 would stabilise in 3 epochs.

In the second simulation, the network learned to classify the test input vectors more gradually. This is because during learning, the weights are drifting towards the input vectors, in comparison with the snap effect in the first simulation. This simulation provides a dynamic view of the network where it is continuously learning throughout the process with performance p alternating between good and bad. The overall picture is of an ongoing partial reselection of proxylets. At the beginning of the simulation, the performance p is initialised to 0 to invoke rapid system learning. The performance of the network is assumed to be measured after every simulation epoch. Since the network is fed with the dummy performance values of 0 and 1 alternately, the weights started to drift toward the input vector at the start of the second epoch (p = 1). But re-selection of the output node in attempt to improve the network performance started when the network went from good performance to the bad performance phase in the third epoch. In the third epoch (p = 0), some of the input vectors select different output nodes, either previously committed or uncommitted nodes, depending on the drifting of the top-down weights they learned on the previous epoch, and on the new fast learning. Since more then one output node is selected for learning at the beginning of the simulation, partial reselection is possible where only one output node out of the 3 output nodes previously selected is re-learned.

5. Conclusions and Future Work

In this paper, we have introduced a modular network architecture based on ART modules and the Kohonen Feature Map. It is capable of stable learning of the network input request patterns in real-time and is able to map them onto the appropriate proxylets available to the system. We have shown the plausibility of the 'snap-drift' algorithm, which is able to provide continuous real-time learning in order to improve the network performance, based on the external performance feedback. These system properties have been confirmed by the conclusion obtained from the experiments performed using the Distributed ART1 module, which was evaluated using performance feedback scenarios.

The full architecture will be evaluated to explore its full potential in providing adaptive network management. There are also several ways to further explore the architecture and algorithm:

- Investigate bottom-up learning with performance feedback to improve the efficiency of the reselection process.
- Modify or substituting the original ART mismatch process to improve the robustness of the network [7,8]
- Explore different ways of using the performance feedback.

References:

[1] G.A. Carpenter, S. Grossberg, "A Massively Parallel Architecture for a Self-Organising Neural Pattern Recognition machine", Computer Vision, Graphics and Image Processing, 37, pp 54-115, 1987

[2] D. Tennehouse, D. Wetherall, "Towards an Active Network Architecture", Computer Communication Reviews, Vol. 26, No. 2, 1996

[3] M. Fry, A. Ghosh, "Application Layer Active Network", Computer Networks, Vol. 31, No.7, 1999

[4] I.W. Marshall, J. Hardwicke, H. Gharib, M. Fisher, P. Mckee, "Active Management of multiservice Networks", Proceeding of the IEEE NOMS2000, pp 981-983

[5] I.W. Marshall, C.M. Roadknight, "Adaptive Management of an Active Services Network", British Telecom Technology Journal, Vol.18, No. 4, 2000

[6] I.W. Marshall, C.M. Roadknight, "Differentiated Quality of Service in Application Layer active Network", Proceedings of International Working Conference on Active Networks, Springer-Verlag, pp 358-370, 16-18 Oct. 2000

[7] D. Palmer-Brown, "High Speed Learning in a Supervised, Self Growing Net", In: I. Aleksander and J. Taylor, eds., Proceeding of ICANN 92, Vol 2, pp 1159 - 1162, ISBN 0 444 894888, 4-7 Sept 1992, Brighton

[8] S. Barker, H. Powell, D. Palmer-Brown, "Size Invariant Attention Focusing (with ANNs)", Proceeding of International Symposium on Multi-Technology Information Processing. 1996.