CONTINUOUS REINFORCED SNAP-DRIFT LEARNING IN A NEURAL ARCHITECTURE FOR PROXYLET SELECTION IN ACTIVE COMPUTER NETWORKS

Sin Wee Lee
Dominic Palmer-Brown*
Computational Intelligence Research Group
School of Computing
Leeds Metropolitan University
Beckett Park Campus, Leeds LS6 3QS
*Email : D. Palmer-Brown@leedsmet.ac.uk

Christopher Roadknight
BTexact Technologies
BT Adastral Park
Martlesham Heath, Ipswich IP5 3RE
and
Computational Intelligence Research Group
School of Computing
Leeds Metropolitan University

KEYWORDS
Computational Intelligence, Artificial Neural Networks, Category Learning, Reinforcement Learning.

ABSTRACT

A new continuous learning method is used to optimise the selection of services in response to user requests in an active computer network simulation environment. The learning is an enhanced version of the 'snap-drift' algorithm, which employs the complementary concepts of fast, minimalist (snap) learning and slower drift (towards the input patterns) learning, in a non-stationary environment where new patterns arrive continually. Snap is based on Adaptive Resonance Theory, and drift on Learning Vector Quantisation. The new algorithm swaps its learning style between these two self-organisational modes when declining performance is detected, but maintains the same learning mode during episodes of improved performance. Performance updates occur at the end of each epoch. Reinforcement is implemented by enabling learning on any given pattern with a probability that increases linearly with declining performance. This method, which is capable of rapid re-learning, is used in the design of a modular neural network system: Performance-guided Adaptive Resonance Theory (P-ART). Simulations demonstrate the learning is stable, and able to discover alternative solutions in rapid response to new performance requirements and significant changes in the stream of input patterns.

INTRODUCTION

The Adaptive Resonance Theory (ART) Network

Developments (Carpenter and Grossberg 1987a) of the original ART (Grossberg, 1976a; Grossberg 1976b) networks include ART1 that self-organises recognition categories for arbitrary sequences of binary input sequences; and ART2 which does the same for either binary or analogue inputs (Carpenter and Grossberg 1987b). Subsequently, ART3 (Carpenter and Grossberg 1990) has been used to implement parallel searches of compressed or distributed recognition codes (output categories) in a neural network hierarchy. Following the successful implementation of the theory in real-time applications, further development has seen the creation of ART2-A (Carpenter et al. 1991a), which is 2 or 3 orders of magnitude faster than ART2. Fuzzy ART (Carpenter et al. 1991b) the fuzzy extension of ART, incorporated computations from fuzzy set theory. Extensions to ART networks to allow supervised learning were also introduced (Palmer-Brown 1992); and ARTMAP (Carpenter et al. 1991c) and Fuzzy ARTMAP (Carpenter et al. 1992) autonomously learn to classify based on predictive success. Furthermore, there are several other versions of ART network (Tan 1997; Carpenter et al. 1998; Bartfai and White 2000), including supervised multi-layer, self-growing systems (Palmer-Brown 1992).

Limitations

There are limitations of ART networks in non-stationary environments where self-organisation needs to take account of periodic or occasional performance feedback:

- The ART network tends to organize itself into a stable state during fast learning whereby the weights stop changing in the presence of new inputs.
- There is no external feedback to improve the performance of the network when it stabilises with poor performance.

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

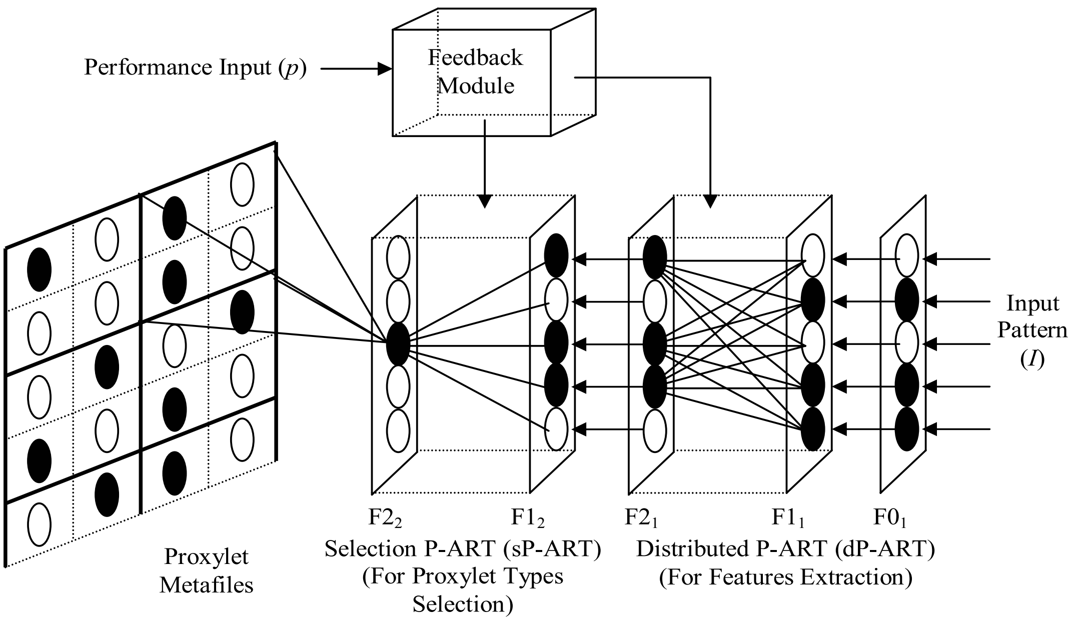
Performance Input (p)
Feedback Module
Input Pattern (I)
$F2_2$
$F1_2$
$F2_1$
$F1_1$
$F0_1$
Proxylet Metafiles
Selection P-ART (sP-ART) (For Proxylet Types Selection)
Distributed P-ART (dP-ART) (For Features Extraction)

Figure 1: Architecture of the P-ART Network

PERFORMANCE-GUIDED ART (P-ART)

P-ART Architecture

The P-ART network proposed is a modular, multi-layered architecture as shown in Figure 1. It is composed of 3 modules, a Distributed P-ART (dP-ART) network, a Selection P-ART (sP-ART) network and a Kohonen Self-Organising Map. The $F1_1 \leftrightarrow F2_1$ connections of the dP-ART network and $F1_2 \leftrightarrow F2_2$ of the sP-ART are interconnected through weighted bottom-up and top-down connections that can be modified during the learning stage. For clarity, only the connections from the F1 layer to the active (winning) F2 node in each P-ART module are shown. The $F0_1 \rightarrow F1_1$ and two P-ART modules connected through $F2_1 \rightarrow F1_2$ are unidirectional, one to one and non-modifiable. Each of the $F2_2$ nodes is hard-wired onto a specific pre-trained region of the Kohonen Feature map where similar available proxylets (the target outputs) are spatially organised on the 2-D map according to their featural similarity.

Overview of the Operation of the System

On presentation of an input pattern at the input layer $F0_1$, the dP-ART will learn to group the input patterns according to their general features using the novel learning principles developed in this work from the snap-drift' algorithm recently developed (Lee et al. 2002; Lee et al. 2003; Lee et al. 2004). The latest version has several improvements over the previous one in terms of the normalization process, and the synchronization of learning between the s and d P-ARTs; but the two key differences are performance guided toggling of learning between snap and drift, and the introduction of a probabilistic aspect to enhance reinforcement and stability. The standard matching and reset mechanism of ART (Carpenter and Grossberg 1987a) is retained: If no existing matching prototype is found, i.e. when the stored pattern prototypes are not a good match for the input, the winning $F2_1$ node is reset and another $F2_1$ node is selected. When no corresponding output category can be found, the network considers the input as novel, and generates a new output category node that learns the current input pattern.

The three winning $F2_1$ nodes, whose prototypes are best match to the current input pattern, are used as the input data to the P-ART module for selecting an appropriate output type (called a proxylet in the target application). For the purpose of selecting the required proxylet, the proxylet type information indicated by the P-ART references pre-trained locations on the Kohonen Self-Organising Map (SOM) (Kohonen 1982; Kohonen 1990a), which represent specific proxylets. If the proxylet is unavailable, one of its neighbours is selected (the most similar alternative available).

A non-specific performance measure is used because, as in many applications, there are no specific performance measures (or external feedback) in response to each individual output decision. This measure is used to encourage or discourage reselection of outputs (proxylet types) to occur in order to improve the performance of the neural system. The continuous learning method is the snap-drift algorithm. It involves toggling between snap and drift modes depending on performance changes. Snap and drift are alternative forms of adaptation, and they are described in the next section,

THE LEARNING. The following is a summary of the steps that occurs in P-ART:

```
Step 1: Initialise parameters: ($\alpha$ = 1, $\sigma$ = 0)
Step 2: For each epoch, t
        Measure or calculate performance in the range {0,1} over the last epoch, P(t).
        Performance improvement, PI = P(t) – P(t-1)
        Set probability of learning, PL = 1 – P(t)
Step 3: For each new input pattern
        Find the D winning nodes with the largest input (or create new nodes for mismatches)

        Set learn (adapt) true with probability PL.

        If learn is true test learning strategy condition:
        IF (PI <= 0) THEN
          Weights of d-PART adapted according to the alternate learning procedure: ($\alpha$, $\sigma$) becomes Inverse ($\alpha$ and $\sigma$) in equation (8) below
        ELSE
          Weights of d-PART adapted according to the same procedure as in the last epoch: ($\alpha$, $\sigma$) unchanged.
Step 4: Process the output pattern of $F2_1$ as input pattern of $F1_2$

          Find winning node (just one) in $F2_2$.
```

Weights of s-PART adapted according to the same learning probability and strategy conditions as above, except that the in first half of the learning epoch, both dP-ART and sP-ART learn, whereas in the second half of the epoch, only sP-ART learns. This allows relearning of the mapping from features to selections without the moving target problem of those features changing simultaneously.

THE LEARNING

Snap-Drift

In an environment where new patterns are introduced over time, the learning utilises a novel snap-drift algorithm based on fast, convergent, minimalist learning (snap) and cautious learning (drift) when the performance is good. Snap is based on a modified form of ART; and drift is based on Learning Vector Quantization (LVQ) (Kohonen 1990b). The two forms are combined within a semi-supervised learning system that shifts its learning style whenever it receives a drop in the performance feedback. So, in general terms, the snap-drift algorithm can be stated as:

$$w = \alpha(Fast\_Learning\_ART) + \sigma(LVQ) \qquad (1)$$

where α and σ are determined by performance feedback. In previous simulations, α and σ were real values (Lee et al. 2002; Lee et al. 2003; Lee et al. 2004). In this paper, α and σ are set to (0, 1) or (1, 0) depending on changes in performance, and the learning is then enabled probabilistically.

Input Encoding

A form of coarse coding (Eurich 1997) is used to represent proportional differences between numeric data encoded within the input patterns, e.g. the representation of the value 15 must be closer in input space to the representation of value 20 than that of say 30. The input pattern is arranged in a 25 bit vector. Each property, such as bandwidth, time, file size, loss and completion guarantee, occupies 5 bits of the overall pattern. Table 1 shows the realistic range for each of the request properties. The coding of the user request is performed as illustrated in Table 2, across a different range for the 5 bits in the case of each property. The input patterns are generated by maintaining the coding of each field in turn and randomly generating the codes for rest of the fields for every 20 patterns, giving 1000 patterns in all.

Table 1: Value Ranges of User Request Properties

Properties	Ranges
Bandwidth	10Kb/s → 2000Kb/s
Time	1ms → 1000ms
Loss	20% → 60%
Cost	0.1p → 100p
Completion Guarantee	40% → 100%

Table 2: Example Coding of Bandwidth in User Requests

Ranges (Kb/s)	User Request
200 → 400	10000
800 → 1000	01100
1800 → 2000	00001

Weights Initialisation

The weights are calculated as floating point and are initialised at the beginning of the simulations. Top-down weights are set randomly to either 0 or 1.

$$w_{ji}(0) = [0,1] \qquad (2)$$

Thus, 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 F_2 from the start. The bottom-up weights w_{ij} are assigned initial values corresponding to the initial values of the top-down weights w_{ji}. This is accomplished by equation (3):

$$w_{ij}(0) = \frac{w_{ji}(0)}{|w_{ji}(0)|} \quad (3)$$

The Distributed P-ART (dP-ART) Learning

On presentation of input pattern, the bottom-up activation is calculated using (4). Then the D number of $F2_1$ nodes with the highest bottom-up activation, using (5), are selected.

$$T_J = \sum |w_{ij} \cap I| \quad (4)$$

$$T_J = max\{ T_J \mid J = 1,2,\ldots\ldots, M \} \quad (5)$$

D is set to 3 in this application. If the distributed output categories are found with the required matching level, the three $F2_1$ nodes will enter into resonant state and learn using (6):

$$w_{Ji}^{(new)} = \alpha(I \cap w_{Ji}^{(old)}) + \sigma(w_{Ji}^{(old)} + \beta(I - w_{Ji}^{(old)})) \ (6)$$

where w_{ji} = top-down weights vectors; I = binary input vectors, and β = the drift speed constant = 0.5.

When α =1, (6) can be simplifies to:

$$w_{Ji}^{(new)} = (I \cap w_{Ji}^{(old)}) \quad (7)$$

This invokes fast minimalist learning, causing the top-down weights to reach their new asymptote on each input presentation:

$$w_J \rightarrow I \cap w_J^{(old)} \quad (8)$$

In contrast, when σ = 1, (6) simplifies to

$$w_{Ji}^{(new)} = (w_{Ji}^{(old)} + \beta(I - w_{Ji}^{(old)})) \quad (9)$$

This causes a simple form of clustering or LVQ at a speed determined by β. As describe in the pseudo code presented in **Operation of the System** above, learning is a combination of the two forms of adaptation, because the mode is toggled between snap and drift whenever performance has deteriorated during the previous epoch. In addition, whether adaptation occurs or not on a given pattern is a probabilistic decision, whereby the probability of the snap or drift occurring is proportional to declining performance. The novel bottom-up learning of the P-ART is a normalised version of the top-down learning:

$$w_{iJ}^{(new)} = \frac{w_{Ji}^{(new)}}{|w_{Ji}^{(new)}|} \quad (10)$$

where $w_{Ji}^{(new)}$ = top-down weights of the network after learning. Poor performance can occur when the final selection of proxylet type is wrong, ***even if*** the general feature extracted by dP-ART is valid. To cope with this, the dP-ART learning is toggled on-off every half-epoch so that sP-ART can readjust its learning of selections without modification of the general features in dP-ART, thus resolving a moving target problem.

The Selection P-ART (sP-ART) Learning

The outputs produced by the dP-ART act as input to the sP-ART. The behaviour of sP-ART is the same as that described in section **P-ART Architecture***,* with one exception; only the F2 node with the highest activation is adapted. Each output node of the sP-ART points to a set of available application-specific groupings (in this case proxylet types). The proxylet type data, containing attributes of the types, is used as off-line training data for the SOM so that it forms a map with similar proxylets placed on adjacent nodes. This allows each output node of the sP-ART to be 'hardwired' onto regions of the SOM. The task of the sP-ART is therefore to learn to associate the correct group of input patterns with an output node that is hardwired to a region of the SOM. The effect of learning and relearning within the sP-ART module is that specific output nodes will relate different groups of input patterns to different regions of the SOM until the performance feedback indicates that it is indexing the SOM regions that select the most appropriate proxylets. In that event, the learning probability is low, so that even if the snap-drift has not yet converged, further adjustment is slow.

The Performance Feedback

The external performance feedback into the PART reflects the performance requirement in different circumstances. Various performance feedbacks profiles in the range {0, 1} are fed into the network to evaluate the dynamic stability and effectiveness of the learning. Initially, some very basic tests with performances of 1 or 0 were evaluated in a simplified system (Lee et al. 2002; Lee et al. 2003; Lee et al. 2004). Below, the simulations involve computing the performance based on a parameter associated with the winning output neuron. Ultimately, a realistic commercial external performance feedback criteria will be established, which will be obtained from BT, to evaluate the improvement in performance of the network learning under realistic external performance feedback. Factors which affect performance include latencies for request with differing time to live, dropping rate for requests with differing time to live, and different charging level with related Quality of Service.

BRITISH TELECOM (BT) APPLICATION

Application Layer Active Network (ALAN)

British Telecom (BT) is the main data network provider in the UK. At present, most applications are run on edge devices (which send and receive data, but do not route third party data), such as servers, PCs and WAP enabled devices. There are strong arguments for moving as many of these applications as possible into the network (Tennenhouse and Wetherall 1996), thereby ensuring optimal placement of applications with respect to performance, version synchronicity (so that more users have the same version), and increased security. 'Active Networking' (Tennenhouse and Wetherall 1996) aims to achieve this application migration into the network by running code within the network on specialised routers. It gives users the ability to load software components onto network devices dynamically without explicit references to any third party. The ALAN architecture (Fry and Ghosh 1999) enabled the user to supply JAVA based active-service codes known as proxylets that run on a network device. Each networked server runs the 'Execution Environments for Proxylets' (EEPs) that contains the user supplied software. The purpose of the architecture is to locate the software at optimal points of the end-to-end path between the server and the clients.

Automated Active Network Management using Distributed Genetic Algorithm (GA)

The original ALAN proposal, the management system supports conventional management agent interfaces (Marshall 1999; Marshall et al. 2000) that respond to instructions from the system operators. Each application is individually placed in the network. However, since ALAN with the potential for an enormous range of services, it is necessary to combine the active services with an automated and adaptive management solution. Recently, a novel adaptive approach, a Distributed Genetic Algorithm (GA) solution was introduced by BT Research Laboratories (Marshall and Roadknight 2001). It performs proxylet placement. Here, P-ART provides a means of finding a set of conditions that produce optimum proxylet selection in an EEP containing the frequently requested proxylets that have been placed. Continuous performance guided adaptation of the mapping of input patters, which contain the main attribute values of user proxylet requests, performs intelligent proxylet type selection.

THE P-ART SIMULATION

P-ART is used for learning and mapping user requests onto appropriate proxylets. The test patterns consist of 1000 input vectors. Each test pattern characterizes the properties of a network request, such as bandwidth, time, file size, loss and completion guarantee. These test patterns are presented in random order with 10 patterns per epoch for 100 epochs where the performance, p, is calculated according to the average bandwidth of selections. This on-line continuous random presentation of test patterns simulates the real world scenario in which pattern presentation order is random, so that a given network request might be repeatedly encountered while others are not used at all.

RESULTS & CONCLUSIONS

Results are presented in Figure 2, 3 and Table 3 and 4. They are representative (typical) of many simulations that have been run. Performance feedback is updated at the end of each epoch of 10 patterns. Much longer epochs are less effective. The best results are for the shortest epochs for which the performance estimate remains a reasonable estimate of overall performance, which of course it would not be for a very small number of patterns. In this application, although there are 1000 patterns, there are only 100 general types, and hence 10 is approximately the smallest reasonable sample for which updates in performance may be trusted to increase or decrease with true overall performance. This will clearly be different for each application. A key observation behind the results presented in the tables and graphs here is that learning is actually over by epoch 64, after which no new selections of proxylets occur until the criteria change to low bandwidth. After 64 (640 pattern presentations), all the performance variation between epochs (the jitter in the performance curve) is due to the epochs being short (in other words, samples of 10 give approximately 70% accurate performance values), and hence the performance over 1000 patterns is actually constant at about the average of the values of the table values from 70-100, which is just under 70%.

In Table 4 and Figure 3 the performance criterion is swapped from high to low bandwidth after 100 epochs, and we see relearning and re-stabilisation occur, with similar results to above once convergence has occurred.

In conclusion, learning stabilizes reliably and is able to map the inputs onto appropriate proxylets. The simulations have shown that the snap-drift algorithm is able to provide continuous probabilistic real-time learning in order to improve the performance, based on the external performance feedback.

Further work is applying the same methods in other areas, but we are also working on attributing performance to each output neuron separately, so that the approach can be appied to classification tasks.

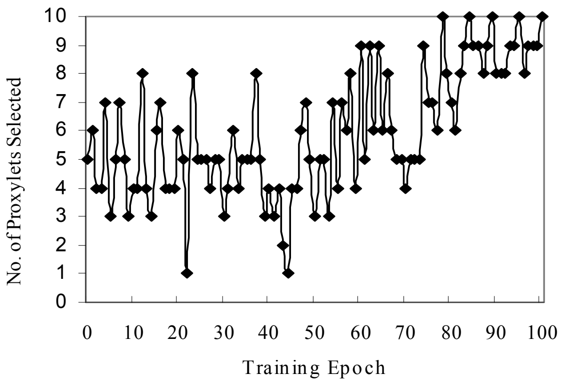

Figure 2: The Selection Frequency of the Proxylet Type. E.g. Bandwidth Bands: Low Bandwidth Proxylet: 0 → 1000 Kb/S and High Bandwidth Proxylet Type: 1001 → 2000 Kb/S

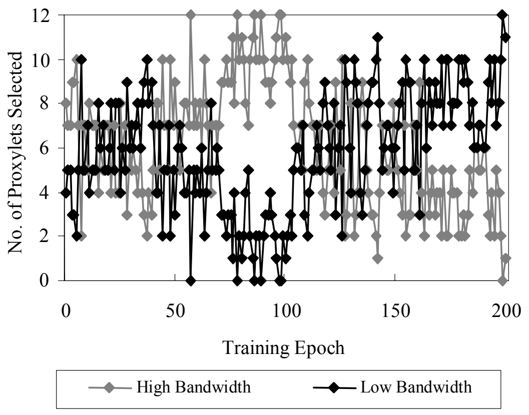

Figure 3: The Selection Frequency of Proxylet Type.

Table 3: Performance of P-ART

Epoch	Average No. of High Bandwidth Proxylet Selected (/10)	Performance (%)
0 – 9	4.9	49
30 – 39	4.8	48
40 – 49	4.0	40
50 – 59	5.2	52
70 – 79	6.6	66
80 – 89	**8.5**	**85**
90 – 99	**8.7**	**87**

Table 4: Performance of P-ART with Switching of Performance Criterion

Epoch	Average No. of High B/W Proxylet Selected (/12)	Average No. of Low B/W Proxylet Selected (/12)	High B/W Proxylet Selection (%)	Low B/W Proxylet Selection (%)
0 - 9	6.08	3.92	50.69	32.64
10 - 19	5.25	4.75	43.75	39.58
30 - 39	3.50	6.50	29.17	54.17
40 - 49	6.00	4.00	50.00	33.33
50 - 59	6.33	3.67	52.78	30.56
60 - 69	5.83	4.17	48.61	34.72
70 - 79	7.83	2.17	65.28	18.06
80 - 89	8.42	1.58	**70.14**	13.19
90 - 99	8.33	1.67	**69.44**	13.89
100 -109	6.67	3.33	55.56	27.78
110 - 119	5.17	4.83	43.06	40.28
130 - 139	4.42	5.58	36.81	46.53
150 - 159	3.75	6.25	31.25	52.08
160 - 169	2.83	7.17	23.61	59.72
170 -1 79	3.42	6.58	28.47	54.86
180 - 189	2.83	7.17	23.61	59.72
190 - 199	0.08	9.92	0.69	**82.64**

REFERENCES

Bartfai, G. and R. White. 2000. “Incremental Learning and Optimization of Hierarchical Clustering with ART-based Modular Networks.” In *Innovations in ART Neural Networks*, L. C. Jain, B. Lazzerini and U. Halici (Eds.). Physica-Verlag, 87 – 132.

Carpenter, G. A. and S. Grossberg. 1987a. “A Massively Parallel Architecture for a Self-Organising Neural Pattern Recognition Machine,” *Computer Vision, Graphics and Image Processing*, Vol. 37, 54-115.

Carpenter, G. A. and S. Grossberg. 1987b. “ART2: Self-Organization of Stable Category Recognition Codes for Analogue Pattern.” *Applied Optics*, Vol. 26, 4919 - 4930.

Carpenter, G. A. and S. Grossberg. 1990. “ART 3: Hierarchical Search Using Chemical Transmitter in Self-Organizing Pattern Recognition Architectures.” Neural Network, Vol. 3, No. 4, 129 – 152.

Carpenter, G. A.; S. Grossberg and D.B. Rosen. 1991a. “ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition.” *Neural Networks*, Vol. 4, 493 - 504.

Carpenter, G. A.; S. Grossberg and D.B. Rosen. 1991b. “Fuzzy ART: Fast Stable Learning and Categorization of Analogue Pattern by an Adaptive Resonance System.” *Neural Networks*, Vol. 4, 759 - 771.

Carpenter, G. A.; S. Grossberg and J. H. Reynold. 1991c. “ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Networks.” *Neural Networks*, Vol. 4, 565 – 588.

Carpenter, G. A.; S. Grossberg; A. Markuzon; J. H. Reynold and D. B. Rosen. 1992. “Fuzzy ARTMAP: A Neural Network Architecture for Incremental

Supervised Learning of Analogue Multidimensional Maps." *IEEE Transactions in Neural Networks*, Vol. 3, No. 5, 698 – 713.

Carpenter, G. A.; B. Milenova and B. Noeske. 1998. "dARTMAP: A Neural Network for Fast Distributed Supervised Learning." *Neural Networks*, Vol. 11, 793 – 813.

Eurich, C. W.; H. Schwegler and R. Woesler. 1997. "Coarse Coding: Applications to the Visual System of Salamenders." *Biological Cybernetics*, Vol. 77, 41 – 47.

Fry, M. and A. Ghosh. 1999. "Application Layer Active Network." *Computer Network*, Vol. 31, No. 7, 655 – 667.

Grossberg, S. 1976a. "Adaptive Pattern Classification and Universal Recoding. I. Parallel Development and Coding of Neural Feature Detectors." *Biological Cybernetics*, Vol. 23, 121 – 134.

Grossberg, S. 1976b. "Adaptive Pattern Classification and Universal Recoding. II. Feedback, Expectation, Olfaction, and Illusions." *Biological Cybernetics*, Vol. 23, 187 – 202.

Kohonen, T. 1982. "Self-Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics*, Vol. 43, 53 – 69.

Kohonen, T. 1990a. "The Self-Organizing Maps." In *Proceeding of IEEE*, Vol. 78, No. 9, 1464 – 1480.

Kohonen, T. 1990b. "Improved Versions of Learning Vector Quantization." In *Proceeding of the International Joint Conference on Neural Networks*, Vol. 1, 545 – 550.

Lee, S. W.; D. Palmer-Brown; J. Tepper and C. M. Roadknight. 2002. " Performance-guided Neural Network for Rapidly Self-Organising Active Network Management." In *Soft Computing Systems: Design, Management and Applications*, A. Abraham J. Ruiz-del-Solar and M. Köppen (Eds.). IOS Press, Amsterdam, 21 – 31.

Lee, S. W.; D. Palmer-Brown; J. Tepper and C. M. Roadknight. 2003. "Snap-Drift: Real-Time Performance-guided Learning." In *Proceeding of International Joint Conference on Neural Networks*, Vol. 2, 1412 – 1416.

Lee, S. W.; D. Palmer-Brown and C. M. Roadknight. 2004. "Performance-guided Neural Network for Rapidly Self-Organising Active Network Management." *Accepted for Neurocomputing*.

Marshall, I. W. 1999. "Application Layer Programmable Internetwork Environment." *British Telecom Technology Journal*, Vol. 17. No. 2, 82 – 94.

Marshall, I. W.; J. Hardwicke; H. Gharid; M. Fisher and P. Mckee. 2000. "Active Management of Multi-Service Networks." In *Proceeding of the IEEE Network Operations and Management Symposium (*Hawaii). 981 – 983.

Marshall, I. W. and C. M. Roadknight. 2001. "Provision of Quality of Service for Active Services." *Computer Networks*, Vol. 36, No. 1, 75 – 85.

Palmer-Brown, D. 1992. "High Speed Learning in a Supervised, Self Growing Net." In *Proceeding of the International Conference of Artificial Neural Network*, Vol.2, 1159 – 1162.

Tan, A-H. 1997. " Cascade ARTMAP: Integrating Neural Computation and Symbolic Knowledge Processing." *IEEE Transactions in Neural Networks*, Vol. 8, No. 2, 237 – 250.

Tennenhouse, D. and D. Wetherall. 1996. "Towards An Active Network Architecture." *Computer Communication Reviews*, Vol. 26, No. 2, 5 – 18.

AUTHOR BIOGRAPHIES

SIN WEE LEE was born in Melaka, Malaysia, in 1976. He graduated with first class honours in electronics and computing engineering from the Nottingham Trent University, United Kingdom, in 1999. His PhD's thesis focuses on the development of performance-guided neural network for active network management. From 2000 to 2001, he was a systems engineer at Malaysia Multimedia University in Malaysia. In December 2001, he joined the School of Computing, Leeds Metropolitan University, Leeds, United Kingdom, with a research scholarship from EPSRC/BT Research Laboratories in Neural Networks. As a research assistant, he works with Dominic Palmer-Brown, on the improvement and development of connectionist language processing, improvements to the snap-drift algorithm and its applications.

DOMINIC PALMER-BROWN is Professor of Neurocomputing and the leader of the Computational Intelligence Research Group, in the School of Computing at Leeds Metropolitan University, UK. The Group have active research links with several organisations, including British Telecom Research Labs, The Centre for Ecology and Hydrology, and with other universities. A key focus of our research is Neurocomputing and related methods of adaptation and learning in cognitive science, intelligent data analysis, and pattern recognition. Dominic has published over 40 international conference and journal papers and supervised 10 PhDs since 1993, having completed his own PhD on an adaptive resonance classifier in 1991. His interests have principally concerned supervised and performance-guided ART, enhanced MLPs for intelligent data analysis, and architectures incorporating MLPs and SRNs for thematic knowledge extraction and natural language processing. He was editor of the review journal Trends in Cognitive Sciences during 2000-2 before rejoining Leeds Met.