

Reinforced Snap-Drift Learning for Proxylet Selection in Active Computer Networks

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

Chris Roadknight
BText Technologies
Martlesham Heath, Ipswich IP5 3RE, United Kingdom.
&
Computational Intelligence Research
School of Computing
Leeds Metropolitan University.

Abstract— A new continuous learning method is applied to the problem of optimizing 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, a hybrid form of learning that employs the complementary modes: fast, minimalist (snap) learning; and slower drift (towards the input patterns) learning, in a non-stationary environment where new patterns are continually introduced. Snap is based on Adaptive Resonance Theory, and drift on Learning Vector Quantization. The new algorithm swaps its learning style between the two modes of self-organisation when declining performance levels are received, but maintains the same learning style during episodes of improved performance. Performance updates occur at the end of each epoch. Reinforcement also occurs by maintaining successful adaptations, since learning is enabled with a probability that increases with declining performance. The method is capable of rapidly re-learning and is used in the design of a modular neural network system, Performance-guided Adaptive Resonance Theory. Simulations demonstrate the learning is stable, effective and able to discover alternative solutions in response to new performance requirements and significant changes in the stream of input patterns.

I. INTRODUCTION

A. The Adaptive Resonance Theory (ART) Network

Adaptive Resonance was introduced by Stephen Grossberg [1, 2]. More recent developments have led to various types of ART networks: ART1 [3] self-organises recognition categories for arbitrary sequences of binary input sequences; ART2, does the same for either binary or analogue inputs [4]. Subsequently, instantiations and developments of the theory, such as ART3 [5] have 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 [6], which is 2 or 3 orders of magnitude faster than ART2. The fuzzy extension of ART, Fuzzy ART [7], incorporated computations from fuzzy set theory into the ART1 architecture. Extensions to ART networks to allow

supervised learning were introduced [8]. ARTMAP [9] and Fuzzy ARTMAP [10] autonomously learn to classify based on predictive success; and there are several other versions of ART network [11 - 13], including supervised multi-layer, self-growing systems [8], [14].

B. 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 even in the presence of new inputs.
- There is no external feedback to improve the performance of the network when it stabilises with poor network performance.

The snap-drift algorithm presented here addresses these shortcomings by shifting the balance between complementary learning modes according to performance feedback.

II. PERFORMANCE-GUIDED ART (P-ART)

A. P-ART Architecture

The P-ART network proposed is a modular, multi-layered architecture as shown in Fig. 1. It 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 are hard-wired onto a specific pre-trained region of the Kohonen Feature map where similar available proxylets are spatially

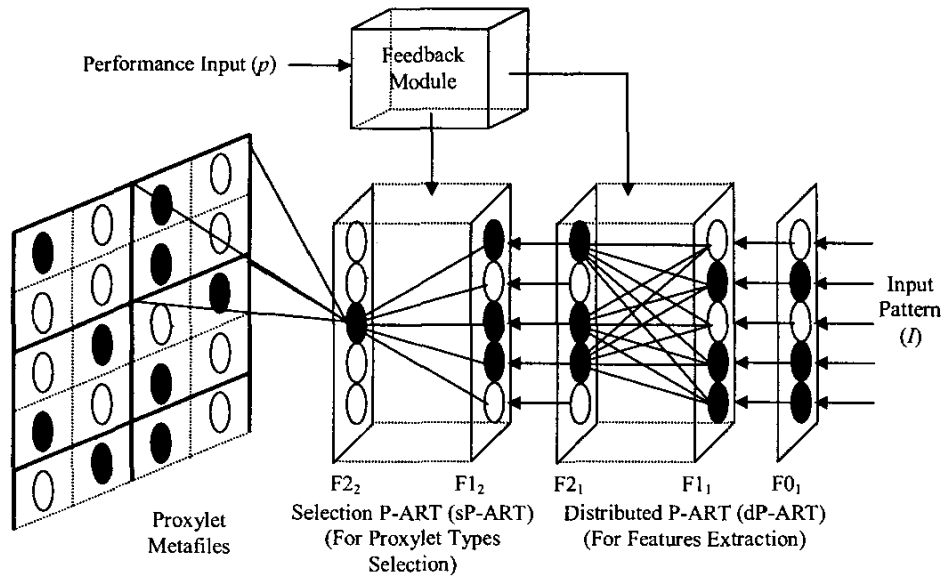


Fig. 1. Architecture of the P-ART network

organised on the 2-D map according to their featural similarity.

B. Overview of the Operation of the System

On presentation of an input pattern at the input layer F0₁, 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 [17 – 19]. 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 [3] 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 F2₁ node is reset and another F2₁ 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 F2₁ 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) [20, 21], 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 Section III, The Learning. The following is a summary of the steps that occur in P-ART:

Step 1: Initialise parameters: ($\alpha = 1, \sigma = 0$)

Step 2: For each epoch, t
 Measure or calculate performance 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 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 $(PI \leq 0)$ THEN

Weights of d-PART adapted according to the alternate learning procedure: (α, σ) becomes Inverse(α and σ) in equation (8) below

ELSE

Weights of d-PART adapted according to the same procedure as in the last epoch: (α, σ) unchanged.

Step 4: Process the output pattern of $F2_1$ as input pattern of $F1_2$

Find winning node (just one) in $F2_2$ by same procedure as in d-PART.

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

III. THE LEARNING

A. The 'Snap-Drift' Algorithm

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) [22]. 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, as indicated in Overview in Section B. In general terms, the snap-drift algorithm can be stated as:

$$w = \alpha(\text{Fast_Learning_ART}) + \sigma(\text{LVQ}) \quad (1)$$

where α and σ are determined by performance feedback. In previous simulations, α and σ were real values [17 - 19]. 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.

B. Input Encoding

A form of coarse coding [23] 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 I shows the realistic range for each of the request properties. The coding of the user request is performed as illustrated in Table II, across a different range

TABLE I
VALUE RANGES OF USER REQUEST PROPERTIES

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

TABLE II
EXAMPLE CODING OF B/W IN THE USER REQUESTS

Range	User Request
200 - 400	10000
800-1000	01100
1800-2000	00001

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.

C. Weight 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.

$$w_{ij}(0) = [0,1] \quad (2)$$

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_{ji} are assigned initial values corresponding to the initial values of the top-down weights w_{ij} . This is accomplished by equation (3):

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

D. The Distributed P-ART (dP-ART) Learning

On presentation of input pattern, the bottom-up activation is calculated using (3). Then the $D F2_1$ nodes with the highest bottom-up activation, using (4), are selected.

$$T_J = \max\{T_J \mid J = 1,2,\dots, M\} \quad (4)$$

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 (5):

$$w_{ji}^{(new)} = \alpha (I \cap w_{ji}^{(old)}) + \sigma (w_{ji}^{(old)} + \beta (I - w_{ji}^{(old)})) \quad (5)$$

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

When $\alpha = 1$, (5) can be simplified to:

$$w_{ji}^{(new)} = (I \cap w_{ji}^{(old)}) \quad (6)$$

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

$$w_j \rightarrow I \cap w_j^{(old)} \quad (7)$$

In contrast, when $\sigma = 1$, (5) simplifies to:

$$w_{ji}^{(new)} = (w_{ji}^{(old)} + \beta (I - w_{ji}^{(old)})) \quad (8)$$

This causes a simple form of clustering or LVQ at a speed determined by β . As describe in the pseudo code in section II, the 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:

$$w_{ji}^{(new)} = \frac{w_{ij}^{(new)}}{|w_{ij}^{(new)}|} \quad (9)$$

where $w_{ij}^{(new)}$ = 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.

E. 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 II A: *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 correct SOM regions and thus selecting 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.

F. The Performance Feedback

The external performance feedback into the PART reflects the performance requirement in different circumstances. Various performance feedback 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 [17 - 19]. 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.

IV. BRITISH TELECOM (BT) APPLICATION

A. 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 [24], 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' [24] 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 [28] 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.

B. Automated Active Network Management using Distributed Genetic Algorithm (GA)

The original ALAN proposal, the management system supports conventional management agent interfaces [29], [30] 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 [31]. 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 patterns, which contain the main attribute values of user proxylet requests, performs intelligent proxylet type selection.

V. 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.

VI. RESULTS & CONCLUSION

Results are presented in Figs. 2, 3 and Tables III and IV. 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

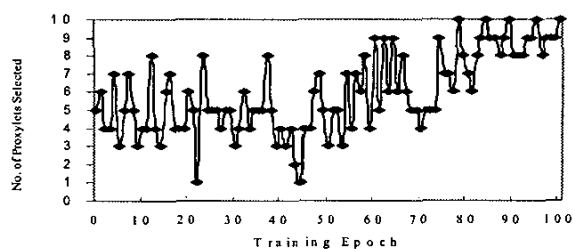


Fig. 2. The selection frequency of the proxylet type. In this case, we have the following bandwidth bands: Low bandwidth proxylet: 0 → 1000 Kb/s and High bandwidth proxylet type: 1001 → 2000 Kb/s

TABLE III

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

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 IV and Fig. 3 the performance criterion is swapped from high to low bandwidth after 100 epochs, and we see relearning and restabilisation, with similar results being achieved once convergence has occurred. In conclusion, learning stabilizes – and restabilises - reliably and is able to map the inputs onto appropriate proxylets. Since the application is a novel one, there are no meaningful comparisons to be made in terms of effectiveness with alternative methods of optimal proxylet selection. There is therefore a clear need to apply snap-drift in other areas to establish its true worth. Currently, we are using snap-drift in a version of P-ART to categorise speech waveforms and detect the key features that characterize aspects of stammering and normal speech, as well as modifying the methods for classification problems in natural language to compare results with those achieved by MLPs on the same tasks.

TABLE IV

Epoch	Average No. of High bandwidth Proxylet Selected (/12)	Average No. of Low bandwidth Proxylet Selected (/12)	High Bandwidth Proxylet Selection Performance (%)	Low Bandwidth Proxylet Selection Performance (%)
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 - 179	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

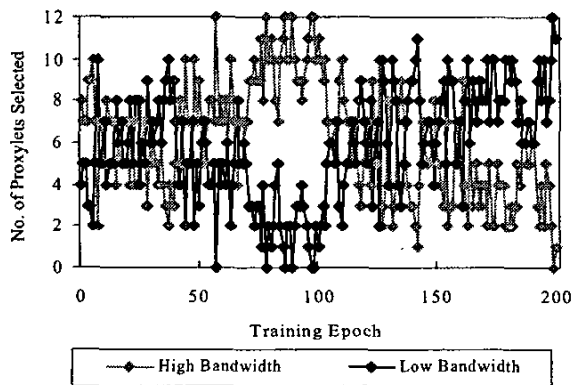


Fig. 3. The Selection Frequency of the Proxylet Type.

REFERENCES

- [1] S. Grossberg, "Adaptive Pattern Classification and Universal Recoding. I. Parallel Development and Coding of Neural Feature Detectors," *Biol. Cybern.*, Vol. 23, 1976, pp. 121 - 134.
- [2] S. Grossberg, "Adaptive Pattern Classification and Universal Recoding. II. Feedback, Expectation, Olfaction, and Illusions," *Biol. Cybern.*, Vol. 23, 1976, pp. 187 - 202.
- [3] G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing*, Vol. 37, 1987, pp. 54-115.
- [4] G. A. Carpenter and S. Grossberg, "ART2: Self-Organization of Stable Category Recognition Codes for Analogue Pattern," *Applied Optics*, Vol. 26, 1987, pp. 4919 - 4930.
- [5] G. A. Carpenter and S. Grossberg, "ART 3: Hierarchical Search Using Chemical Transmitter in Self-Organizing Pattern Recognition Architectures," *Neural Networks*, Vol. 3, No. 4, 1990, pp. 129 - 152.
- [6] G. A. Carpenter, S. Grossberg and D.B. Rosen, "ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition," *Neural Networks*, Vol. 4, 1991, pp. 493 - 504.
- [7] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast Stable Learning and Categorization of Analogue Pattern by an Adaptive Resonance System," *Neural Networks*, Vol. 4, 1991, pp. 759 - 771.
- [8] D. Palmer-Brown, "High Speed Learning in a Supervised, Self Growing Net," in *1992 Proc. ICANN*, Vol. 2, pp. 1159-1162.
- [9] G. A. Carpenter, S. Grossberg, and J. H. Reynold, "ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Networks," *Neural Networks*, Vol. 4, 1991, pp. 565 - 588.
- [10] G. A. Carpenter, S. Grossberg, A. Markuzon, J. H. Reynold, and D. B. Rosen, "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analogue Multidimensional Maps," *IEEE Trans. Neural Network*, Vol. 3, No. 5, 1992, pp. 698 - 713.
- [11] A-H Tan, "Cascade ARTMAP: Integrating Neural Computation and Symbolic Knowledge Processing," *IEEE Trans. Neural Networks*, Vol. 8, No. 2, 1997, pp. 237 - 250.
- [12] G. A. Carpenter, B. Milenova, and B. Noeske, "dARTMAP: A Neural Network for Fast Distributed Supervised Learning," *Neural Networks*, Vol. 11, 1998, pp. 793 - 813.
- [13] G. Bartfai, and R. White, "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. Haldi, Eds. Physica-Verlag, 2000, pp. 87 - 132.
- [14] S. Barker, H. Powell, and D. Palmer-Brown, "Size Invariant Attention Focusing (with ANNs)," in *1996 Proc. Int. Symp. Multi-Technology Information Processing*.
- [15] G. A. Carpenter, and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Networks," *IEEE Computer*, Vol. 21, No. 3, 1988, pp. 77 - 88.
- [16] G. A. Carpenter, and S. Grossberg, "Search Mechanism for Adaptive Resonance Theory (ART) Architecture," in *1989 Proc. IJCNN*, Vol. 1, pp. 201 - 205.
- [17] S. W. Lee, D. Palmer-Brown, J. Tepper, and C. M. Roadknight, "Performance-guided Neural Network for Rapidly Self-Organising Active Network Management," in *Soft Computing Systems: Design, Management and Applications*, A. Abraham et. al., eds, IOS Press, 2002, pp. 21 - 31.
- [18] S. W. Lee, D. Palmer-Brown, J. Tepper, and C. M. Roadknight, "Snap-Drift: Real-time Performance-guided Learning," in *2003 Proc. IJCNN*, Vol. 2, pp. 1412 - 1416.
- [19] S.W. Lee, D. Palmer-Brown, C. Roadknight. Performance-guided neural network for rapidly self-organising active network management. Accepted for Neurocomputing.
- [20] T. Kohonen, "Self-Organised Formation of Topologically Correct Feature Maps," *Biol. Cybern.*, Vol. 43, 1982, pp. 53 - 69.
- [21] T. Kohonen, "The Self-Organizing Maps," *Proc. IEEE*, Vol. 78, No. 9, 1990, pp. 1464 - 1480.
- [22] T. Kohonen, "Improved Versions of Learning Vector Quantization," in *1990 Proc. IJCNN*, Vol. 1, pp. 545 - 550.
- [23] C. W. Eurich, H. Schwegler, and R. Woessler, "Coarse Coding: Applications to the Visual System of Salamanders," *Biol. Cybern.*, Vol. 77, 1997, pp. 41-47.
- [24] D. Tennenhouse, and D. Wetherall, "Towards An Active Network Architecture," *Comp. Comm. Reviews*, Vol. 26, No. 2, 1996, pp. 5 - 18.
- [25] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open Signalling for ATM, Internet and Mobile network," *Comp. Comm. Reviews*, Vol. 29, No. 1, Jan 1999.
- [26] D. S. Alexander, "Active Bridging," *Comp. Comm. Reviews*, Vol. 26, No. 2, 1996, pp. 101 - 111.
- [27] I. W. Marshall, "Active Network - Making the Next Generation Internet Flexible," *BT Eng.*, Vol. 18, 1999, pp. 2-8.
- [28] M. Fry, and A. Ghosh, "Application Layer Active Network," *Computer Networks*, Vol. 31, No. 7, 1999, pp. 655-667.
- [29] I. W. Marshall, "Application Layer Programmable Internetwork Environment," *British Tech. Journal*, Vol. 17, No. 2, 1999, pp. 82 - 94.
- [30] I. W. Marshall, J. Hardwicke, H. Gharid, M. Fisher, and P. Mckee, "Active Management of Multi-Service Networks," in *2000 IEEE Proc. NOMS*.
- [31] I. W. Marshall, and C. M. Roadknight, "Provision of Quality of Service for Active Services," *Computer Networks*, Vol. 36, No. 1, 2001, pp. 75 - 85.