Chapter 11
Self Organisation and Modal Learning: Algorithms and Applications

Dominic Palmer-Brown and Chrisina Jayne

Abstract. Modal learning in neural computing [33] refers to the strategic combination of modes of adaptation and learning within a single artificial neural network structure. Modes, in this context, are learning methods that are transferable from one learning architecture to another, such as weight update equations. In modal learning two or more modes may proceed in parallel in different parts of the neural computing structure (layers and neurons), or they occupy the same part of the structure, and there is a mechanism for allowing the neural network to switch between modes.

From a theoretical perspective any individual mode has inherent limitations because it is trying to optimise a particular objective function. Since we cannot in general know a priori the most effective learning method or combination of methods for solving a given problem, we should equip the system (the neural network) with the means to discover the optimal combination of learning modes during the learning process. There is potential to furnish a neural system with numerous modes. Most of the work conducted so far concentrates on the effectiveness of two to four modes. The modal learning approach applies equally to supervised and unsupervised (including self organisational) methods. In this chapter, we focus on modal self organisation.

Examples of modal learning methods include the Snap-Drift Neural Network (SDNN) [5, 25, 28, 33, 32] which toggles its learning between two modes, an adaptive function neural network, in which adaptation applies simultaneously to both the weights and to the shape of the individual neuron activation functions, and the combination of four learning modes, in the form of Snap-drift ADaptive FUnction Neural Network [17, 18, 33]. In this chapter, after reviewing modal learning in general, we present some examples methods of modal self organisation. Self organisation is taken in the broadest context to include unsupervised methods. We review the simple unsupervised modal method called snap-drift [5, 25, 28, 32], which combines Learning Vector Quantization [21, 22, 23, 37] with a 'Min' or Fuzzy AND

Dominic Palmer-Brown · Chrisina Jayne
London Metropolitan University, 166-220 Holloway Road, London, N7 8DB, UK
e-mail: {d.palmer-brown,c.jayne}@londonmet.ac.uk

M. Bianchini et al. (Eds.): *Handbook on Neural Information Processing*, ISRL 49, pp. 379–400.
DOI: 10.1007/978-3-642-36657-4_11

method. Snap-drift is then applied to the Self-Organising Map [34]. The methods are utilised in numerous real-world problems such as grouping learners' responses to multiple choice questions, natural language phrase recognition and pattern classification on well known datasets. Algorithms, dataset descriptions, pseudocode and Matlab code are presented.

1 Introduction

Modal learning is a new approach to neural computing and it refers to the combination of more than one mode of learning within a single neural network. It contrasts with hybrid [42] or multi-modal approaches [6] where the modes of learning occur in different modules and/or at strictly separated times. There are several reasons to explore modal learning. One motivation is to overcome the inherent limitations of any given mode (for example some modes memorise specific features, others average across features, and both approaches may be relevant according to the circumstances); another motivation is from neuroscience, cognitive science and human learning, where multiple modes are required to explain behaviour; and a third reason is non-stationary input data, or time-variant learning objectives, where the required mode is a function of time. Combining modes in one network also presents some efficiency gains, and the possibility of maximising the flexibility (through the parallel application of many forms of plasticity) of the neural network as a learning agent.

Twenty years ago there were already several forms of artificial neural network, each utilising a different form of learning. Along the way, many forms of learning have been hailed as superior forms. However, decades after the introduction of Kohonen learning, SOMs [20], and Backpropagation [38] they are still being used alongside more recent methods such as Bayesian [30] and SVMs [41]. No single method or mode prevails. A wide range of methods are still in use, simply because there are significant problems and datasets for which each method is suitable and effective. In this context, Modal Learning arises from the desire to equip a single neural network or module with the power of several modes of learning, to achieve learning results that a single mode could not achieve, through exploitation of the complementary nature of each mode. A mode in this context is defined as an adaptation method for learning that could be applied in more than one type of architecture or network. It is analogous to a human mode of learning, such as learning by analogy or category learning. Modes of learning map onto learning objectives. Well known modes therefore include the Delta Rule [43], Backpropagation (BP), Learning Vector quantization (LVQ) [22], and Hebbian Learning [14]. In contrast, the Adaptive Resonance Theory (ART) [3] or Bayesian neural networks [30] define architectures and approaches to learning, within which particular modes are used.

In general, the objective of learning may be unknown, changing, or difficult to quantify. Even when the objective of learning is transparent, the optimal mode is not a given. Therefore it is not possible to know a-priori which is the most suitable learning mode. For example, Backpropagation is good for approximation and

transformation, but if the features within the data need to be assimilated (or memorised) directly, then learning vector quantization or SVM [41] would be more appropriate. The learning agent should be able to establish the most effective mode or combination of modes during, and as part of, the learning process.

Each mode is inherently limited because it is tied to a particular objective function. A simple illustration of the potential benefits of a modal learning approach can be seen in the following sequence of class boundaries in a $2D$, 2 class problem (Fig. 1). The example illustrates how it is necessary to increase complexity to an extent that is not well supported by the data in order to find a single mode solution. In contrast a solution with more than one mode allows problem decomposition to occur, and each mode can be relatively simple in structure. A solution can be achieved by combining a straight line (perceptron), a simple curve (multilayer perceptron) and a cluster. This requires 3 modes of learning (see Fig. 2 left). Alternatively combining a curve (multilayer perceptron) and a cluster requires only 2 modes of learning (Fig. 2 right).

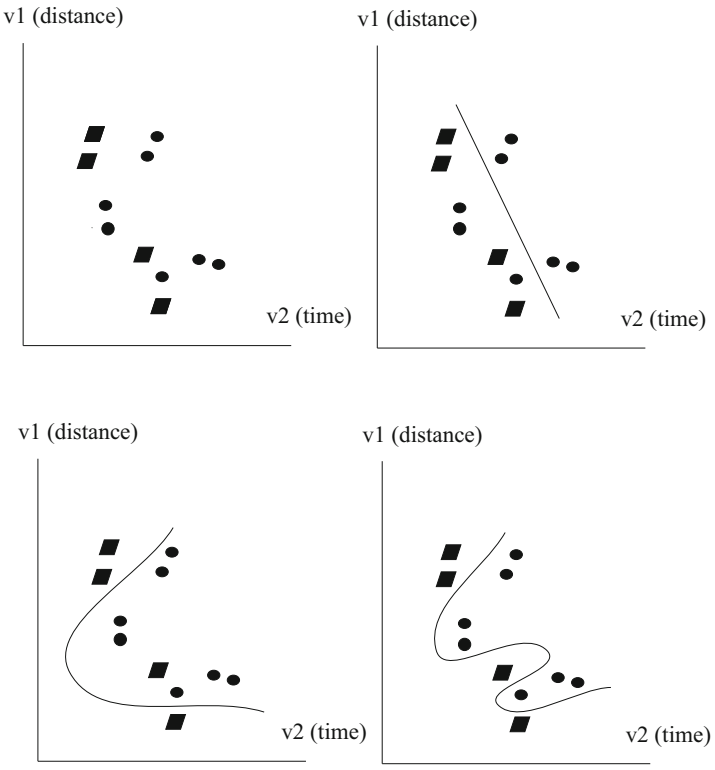

Fig. 1 Increasingly complex solutions to a 2-class problem

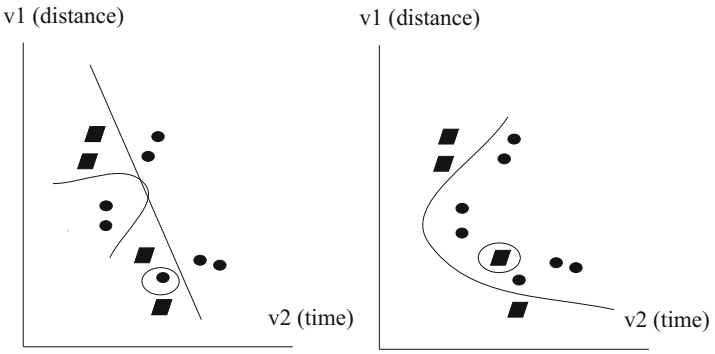

Fig. 2 3 mode solution and 2 mode solution

Rather than trying to solve the whole problem with a single mode, a simpler learnt solution is achievable by combining modes of learning. When we look at human and machine learning in a wider context, there are many reasons and motivations to consider modal learning, as it allows for a range of learning methods to be taken into account, along the spectrum from memorisation to generalisation.

Modal learning in general is achieved by optimising the objective functions for each mode. This involves minimising $\sum_i O_i$, where O_i is the objective function for mode i. For example, if we are combining the Delta Rule and Learning Vector Quantisation, we must minimise $[\sum(squared\ errors) + \sum(Input - Weight)]$.

Snap-Drift Neural Network (SDNN), introduced in [5, 25, 28, 33, 32], is an example of a modal learning method which toggles its weight update equation between two modes: 'Min'('Fuzzy AND') and Learning Vector Quantization [21, 22, 23, 37]. Another example is the Adaptive Function Neural network [17], in which adaptation applies simultaneously to both the weights and to the shape of the individual neuron activation functions [17, 18]. The Snap-drift ADaptive FUnction Neural Network [17, 18, 33] combines four learning modes: two modes for the weights in the snap-drift layer and two modes in the adaptive function layer; one mode adapts weights and the other adapts the function.

In this chapter the SDNN algorithm and its application to self-organisation [34] are reviewed. Section 2 gives a detailed description of the SDNN algorithm, its architecture and pseudocode. In Section 3 the Snap-Drift SOM (SDSOM) which adopts the Kohonen SOM architecture [23] is presented. Section 4 illustrates the applications of SDNN and SDSOM to publically available data. The effect of applying snap-drift is evaluated in respect to classification performance and data visualisation (the shape of the resultant maps). Section 4 shows also the application of SDNN to a real-world problem related to e-learning and more specifically grouping learners responses to multiple choice questions. The Appendix at the end of the chapter includes Matlab code for the SDNN with detailed comments showing how the method can be used in practice.

2 Snap-Drift Neural Network

2.1 Description

The Snap-Drift Neural Network (SDNN) is an unsupervised algorithm able to adapt rapidly, for example in non-stationary environments where new patterns are introduced over time. The standard snap-drift neural network (SDNN) algorithm has been successfully applied for continuous learning in many diverse applications [8, 25, 26, 27, 28]. An example application of the unsupervised snap-drift algorithm is the analysis and interpretation of data representing interactions between trainee computer network managers and a simulated network management system [25], where it helped to identify patterns of the user behaviour. Another application is feature discovery and clustering of speech waveforms recorded from non-stammering and stammering speakers [28]. Phonetic properties of non-stammering and stammering speech were discovered, and rapid automatic classification into stammering and non stammering speech was found to be possible. Most recently, snap-drift has been successfully applied to categorising student responses to multiple choice questions in a virtual learning context [9, 32].

It is essentially a simple modal learning method, which swaps periodically between the two learning modes (snap and drift). Snap is based on the fuzzy AND (Min) of input and weight; and drift is based on learning vector quantization (LVQ) [24]. Snap-Drift harnesses the complementary strengths of the two modes of learning which are dynamically combined in a rapid form of adaptation.

The learning process is unsupervised and unlike error minimisation and maximum likelihood methods in MLPs [38]. Those methods perform optimisation for classification by for example pushing features in the direction that minimizes classification error. In such methods there is no requirement for the learned weight vector to be a significant feature within the input data. In contrast, SDNN swaps its learning mode to find, in an unsupervised fashion, to find a rich set of features in the data and uses them to group the data into categories. The effect of the learning process for a single neuron is illustrated in Fig. 3, for one cluster of data in two dimensions. The weight vectors are normalised and so they are maintained at unit length. The weight vector for the neuron will settle to an angle somewhere between the snap and drift angles. So, each weight vector is bounded by snap and drift: snap gives the angle of the minimum values (on all dimensions) and drifting gives the average angle of the patterns grouped under the neuron. Hence, snap essentially provides an anchor vector pointing at the 'bottom left hand corner' of the pattern group for which the neuron wins. This represents a feature that is common to all the patterns in the group and gives a high probability of rapid (in terms of epochs) convergence (both snap and drift are convergent, but snap is faster). Drift tilts the vector towards the centroid angle of the group and ensures that an average, generalised feature is included in the final vector. The angular range of the pattern-group membership depends on the proximity of neighbouring groups (competition), but can also be controlled by adjusting a threshold on the weighted sum of inputs to the neurons.

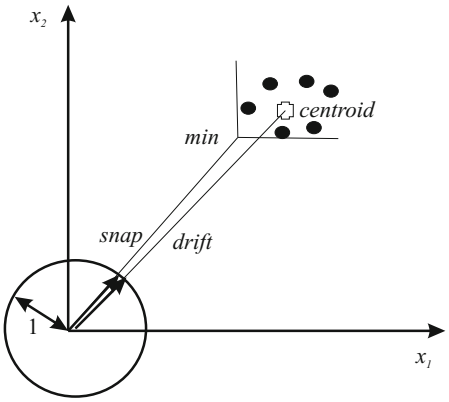

Fig. 3 Snap and drift weight vectors

2.2 Architecture

The architecture of the Snap-Drift Neural network is shown in (Fig. 4). It consists of an input layer, a distributed *d* layer for feature extraction and a selection *s* layer for feature classification [25]. The distributed *d* layer groups the input patterns according to their features using snap-drift. The *D* most activated(winning) *d* nodes whose weight vectors best match the current input pattern are used as the input data to the selection, *s* layer, for the purposes of feature classification. In the *d* layer, the output nodes with the highest net input are accepted as winners.

In the *s* layer, a quality assurance threshold is applied. If the net input of the most active *s* node is above the threshold, that *s* node is accepted as the winner, and defines the category of the input pattern; otherwise a new uncommitted output node is recruited as the winner and its weights initialized with the current *d* layer pattern

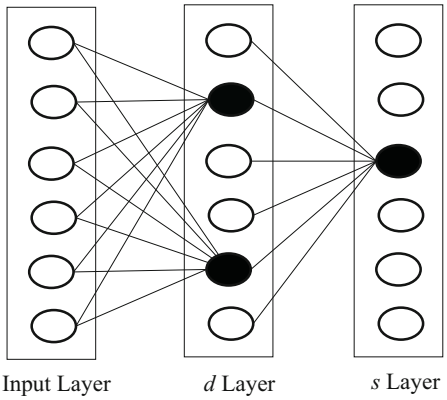

Fig. 4 Snap-drift architecture

(the D winning nodes). The threshold influences the granularity of the categories. If the threshold is zero, relatively few categories will be formed, based on very different combinations of d nodes; but if the threshold is set high, even a small proportion of the winning d nodes not matching the current weights of the winning s node leads to the recruitment a new s node, and thus finer grained categories are formed.

2.3 Algorithm

In essence the snap-drift weight update algorithm can be stated as:

$$Snap-drift = \alpha(Snap) + (1-\alpha)(drift) \tag{1}$$

Snap-drift learning uses a combination of fuzzy AND (or MIN) learning (snap), and Learning Vector Quantisation (drift) [24].

The learning of both of the layers in the neural system follows:

$$w_{ji}^{(new)} = \alpha(I \cap w_{ji}^{(old)}) + (1-\alpha)(w_{ji}^{(old)} + \beta(I - w_{ji}^{(old)})) \,, \tag{2}$$

where $w_{ji} = weights\ vectors$; $I = input\ vectors$, and $\beta = the\ drift\ learning\ rate$. The β learning rate may have different values for the d and s layers. When $\alpha = 1$, fast, minimalist (snap) learning is invoked:

$$w_{ji}^{(new)} = I \cap w_{ji}^{(old)} \tag{3}$$

This works for binary data, otherwise equation (3) becomes the fuzzy AND of the weight with the data, $Min(I, w_{ji}^{(old)})$. Consequently, snap encodes, within the weights, the common elements of all patterns for which the neuron wins. In contrast, when $\alpha = 0$, (2) simplifies to:

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \beta(I - w_{ji}^{(old)}) \tag{4}$$

which implements a simple form of clustering (drift towards the centroid of the pattern group) or LVQ, at a speed determined by β. Finally, after either snap or drift, weights are normalized:

$$w_{ji}^{(new)} = \frac{w_{ji}^{(new)}}{\left| w_{ji}^{(new)} \right|} \tag{5}$$

The snap and drift modes provide complementary features; snap capturing the common elements of the group of patterns as represented by the minimum values on each input dimension, whereas drift captures the average values of the group of patterns. Snap also has the effect of contributing to rapid convergence. The two modes provide different features that are overlaid within a single network and a shared set of weight vectors.

This is a summary of the SDNN algorithm:

1. Initialise parameters:
   n_d number of nodes in the d layer
   n_s number of nodes in the s layer
   D number of winning nodes in the d layer
   t quality assurance threshold
   λ learning rate d layer
   μ learning rate s layer
   Initialise weights $\mathbf{W_d}$ between input and d layer with randmoly selected patterns and normilise these
   Initialise and normlaise weights $\mathbf{W_s}$ between d layer and s (output) layer to small numbers between 0 and 1
   Intialise the output vector $\mathbf{s_o}$ from s layer to 0
   Initialize $\alpha = 1$ or 0, (equation (1))
2. For each epoch (t) and for each input pattern $\mathbf{x}$
   a. Find the D winning nodes at d layer with the largest net input, where $netinput = \mathbf{x}.\mathbf{W_d}$
   b. Adapt the weights $\mathbf{W_d}$ according to the snap or drift learning procedure snap-drift: (equation (2))
   c. Set the the output of d layer $\mathbf{d_o}$ as follows: the elements corresponding to the D winning nodes to 1 and the rest to 0
   d. Calculate the input vector $\mathbf{s_i}$ in the s layer as the product of $\mathbf{d_o}$ and the weights $\mathbf{W_s}$, $\mathbf{s_i} = \mathbf{d_o}.\mathbf{W_s}$
   e. Set $maxVal =$ the activation value of node in s layer with the largest input component from the vector $\mathbf{s_i}$
   f. Test the threshold condition:
      IF $(maxVal > t)$
      THEN
      Weights $\mathbf{W_s}$ are adapted according to the snap or drift learning procedure in equation (2))
      ELSE
      Set the value of the first element in the vector $\mathbf{s_o}$ that equals 0 to 1. (This recruits the first uncomitted node.)
      Adapt weights $\mathbf{W_s}$ according to the equation (2)

3 Snap-Drift Self-Organising Map

3.1 Description

The self-organising feature map algorithm (SOM) developed by Kohonen [23] has been used widely in clustering analysis and visualization of high-dimensional data

[37]. The SOMs can also be used for pattern classification by applying fine tuning of the map with LVQ learning algorithms [21, 23, 24]. The Kohonen feature map was inspired by the idea that self-organising maps resemble the topologically organised maps found in the cortices of the brain [20]. The SOM algorithm is based on unsupervised learning realised by finding the best matching node (the winner) on the map to the input vector and adapting the weights of the winner and the topological neighbourhood nodes. After the training each node on the map identifies particular input vectors and the organisation of the map reflects the organisation of the input data. In this work, SDNN is deployed in a self-organising map, to ascertain whether the advantages of snap-drift over LVQ alone (drift, without snap) transfer into the formation of topological maps. We are interested in classification performance and data visualisation (the shape of the resultant maps).

3.2 Architecture

The SDSOM has the same architecture Fig. 5 as a standard SOM, with a layer of input nodes connecting to the self organising map layer. A shrinking neighbourhood is used during training, as in SOM, with the weight vector of each neighbour of the winning node being adapted according to the input pattern. The difference in SDSOM is the weight update, which consists of either snap (*min* of input and weight) or drift (LVQ, as in SOM) (equation (2)).

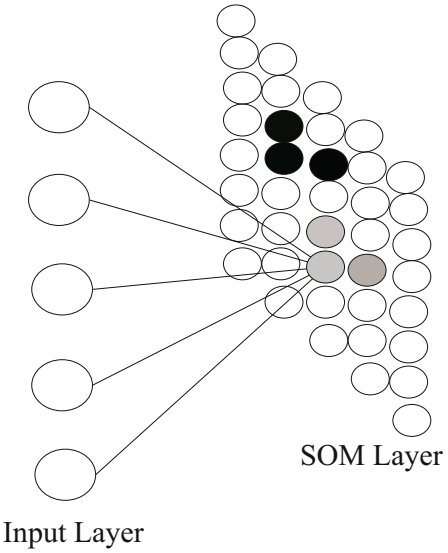

Fig. 5 SOM architecture

3.3 Algorithm

This is a summary of the SDSOM algorithm:

1. Initialize parameters: $\alpha = 1$ or 0, (equation (1))
   Set size of the SOM layer map
   Initialize neighborhood size
   Initialize weights w between input and SOM layer with the values of randomly selected input patterns
   Normalize weights
   Initialize learning rate for drift mode in the range (0,1)
2. For each epoch, swap α between 0 or 1
   a. For each input pattern **x**
      i. Find the winning node in SOM with the largest net input which is the product of the pattern and the weights **x**.**W**
      ii. Update weights of the winning node and its neighbour nodes according to the current learning mode (equation (2))
      iii. Normalize weights (equation (5))
   b. Decrease the neighborhood size by 1
3. Label som layer nodes

The shaded nodes in Fig. 5 represent different classes or labels. Nodes receive the class label of the majority of the patterns for which they win. There is generally a tendency for neighbouring nodes to have the same class, given the nature of SOMs, but this is not forced by the labelling algorithm.

4 Applications

4.1 Applications of SDNN and SDSOM to Publicly Available Data

A range of data sets are chosen to represent a variety of learning challenges. They vary in terms of the number of input variables, the number of classes, and the level of separability of the classes. Since they are all known and freely available they provide useful benchmark comparisons with a number of neural computing and other machine learning techniques.

4.1.1 Description of Data

Animal Data. The Animal data presents a simple classification problem. It is artificial data and consists of 16 animals described by 13 attributes such as size, number of legs etc. [37]. The 16 animals are grouped into three classes (the first one represents bird, the second represents carnivore and the third represents herbivore).

Iris Data. The Iris data set has three classes setosa, versicolor and virginica [10, 7]. The iris data has 150 patterns, each with 4 attributes. The class distribution is 33.3%

for each of 3 classes. One of the classes is linearly separable from the other two, and the two are linearly inseparable from each other.

Wine Data. The Wine data set is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars [11]. The analysis determines the quantities of 13 constituents (input variables) found in each of the three types of wines. There are 178 patterns with the following distribution: class 1:59, class 2:71, class 3:48.

Optical and Pen-Based Recognition of Handwritten Digits (OCR) Data. The OCR data set [1, 19] consists of 3823 training and 1797 testing patterns. Each pattern has 64 attributes which are integer numbers between 0 and 16. There are 10 classes corresponding to the digits 0 to 9. The 64 attributes are extracted from normalised bitmaps of handwritten digits by 43 people.

The experiments with the OCR data set use the already existing division of training/testing patterns 3828 and 1797 respectively, as originally proposed by Kaynak [19]. This facilitates direct performance comparisons between SDSOM and alternative algorithms that have been applied to the same data.

Natural Language Processing Data (NLP). The Lancaster Parsed Corpus [LPC] is a corpus of English sentences excerpted from printed publications of the year 1961, and is a subset of the Lancaster-Oslo/Bergen Corpus [12]. Words are tagged with their syntactic categories and each sentence in the LPC has undergone syntactic analysis. Phrase recognition is a well defined and well known application and a benchmark for testing the performance of neural networks in the field of Natural Language Processing (NLP) [29, 39]. The individual input patterns are encoded in binary according to the structure of the pre-tagged corpus [12]. This is achieved by separating the input layer into several regions for each tag, where each region corresponds to a different symbol type. A total of 45 bits are needed to encode all symbol types. In syntactic terms, there is a variety of terminal and non-terminal symbols tags. The terminal symbol groups are: punctuation (Pu), conjunction (Co), nouns (NP), verbs (VP) and prepositions (PP). The non-terminal symbol groups are Sentences (S), Finite clauses (F), Non-finite clauses (T), major phrase types (V) and minor phrase types (M). Together with a maximum of 4 Look Back symbols and 1 Look Ahead symbol [40], this makes a total of 15 input fields. By using linear binary coding for each symbol type within each input field, the size of an input pattern is 45 x 15 = 675 bits when using a standard time delay neural network input arrangement such as in [27]. By sampling pre-tagged sentences from LPC, we generate 254 input patterns, from all stages of parsing, typically involving mixtures of terminal and non-terminal symbols. Table 1 shows the number of input patterns for each symbol type.

For this problem half of the input set (127 patterns) is used for training. There are 2 types of test data: Natural Test data (ND) and Pure test data (PD) [40]. The ND consists of the remaining 127 patterns, which contains a mixture of some patterns also present in the training set because they happen to occur more than once (naturally occurring syntactic repetition), and new input patterns that have never encountered before. The PD consists entirely of input patterns that have never been

Table 1 Symbol types and number of input sequences

Symbol Type	Symbol and Description	Number of sequences.
Minor Phrase	E = Label used for existential 'there'	2
Major Phrase	J = An adjective phrase	4
	= A noun phrase	83
	Na = A noun phrase marked as subject of the verb	14
	P = A prepositional phrase	16
	Po = A prepositional phrase beginning with preposition 'of'	8
	R = An adverb phrase	12
	Rq = an adverb phrase beginning with a wh-word, e.g. 'How do you feel?' or 'How long'	1
	V = A finite 'verb phrase' i.e. one that exclude objects, complements	49
	Vi = Non-finite verb phrase	5
Sentence	S = Sentence	50
	S& = Compound sentence	2
	S+ = Compound sentence	2
Non-finite and Verbless Clause	Ti = Infinitive clause	4

encountered before (natural repetition removed). For each run the training patterns are selected at random from the entire data set. The average number of patterns for the PD is 102 (st. dev 4).

4.1.2 Experiments and Results

Experiments are carried out with SDNN, SDSOM and SOM. SDNN and SDSOM are typically trained between 200 and 250 epochs while SOM is trained for 500 epochs. This is long enough for the SDNN groups and the SOM maps to be stable in all cases. Summary of the results are presented in Table 2. The SDNN results for the OCR data in Table 2 are based on combining SDNN and the Adaptive Function Neural Network as previously published in [18], and the NLP results are from [27].

In order to perform a labelling of nodes for the purposes of classification the number of patterns for which the node wins is accumulated for each class and for each node. The majority class, with the highest number of patterns, becomes the class label of that node. The training classification score is the percentage of patterns categorised by nodes of the correct class. The training class labels are retained for use in testing. Nodes in the *s* (SDNN) layer or SOM map that by the end of training have no associated patterns for which they win are not labelled. During testing,

Table 2 Mean % correct classification for training and test sets based on 10 runs. Standard deviation given in the brackets.

Method/Data set	Animal	Iris	Wine	OCR	NLP - ND	NLP - PD
SDNN train	100(0)	98(0.82)	95.8(0.8)	99.53	93.47	93.47
SDNN test	100(0)	100(0)	95.6(0.9)	94.99	89.65	86.98
SDSOM train	100(0)	100(0)	100(0)	99.7(0)	100(0)	100(0)
SDSOM test	100(0)	98.8(1.5)	91.7(2.2)	97.3(0)	81.8(1.7)	77.6(2.4)
SOM train	100(0)	100 (0)	100 (0)	99.6(0)	100(0)	100(0)
SOM test	100(0)	95.7 (3.1)	85(3.8)	97.3(0)	79.8(3.1)	75.9(5.1)

if a winning node is unlabelled (which is rare) then the most active labelled node provides the class (correct or incorrect).

The Animal data presents a relatively easy classification task because each pattern differs quite significantly, therefore it is a simple challenge for any method to separate or classify them individually without the need for generalised rules. All methods perform well. There is however an important qualitative difference between the SDSOM and SOM results. SDSOM has projected the classes onto the map in a linearly separable fashion; two straight lines can separate the three animal classes on the map. This is not possible in the SOM, which mixes the herbivores and carnivores to a greater extent. The snap mode finds some common elements that are specific to herbivores that are not based on the overall similarity of herbivores across all dimensions, which is the limitation of LVQ, or any method that calculates overall similarity. This is a characteristic of modal learning. By superposition in the weight vectors of features from both modes, the network assimilates a combination of overall similarity of pattern groups and specific within-pattern features.

The Iris maps differ substantially between SOM and SDSOM, see Fig. 6. The SOM map presents a widely dispersed set of points (Fig. 6 right). They are nonetheless in clear regions associated with the three classes. However, the lines between classes in the map are curved with several changes of direction and there is no margin between the classes, even in the case of the linearly separable classes. In the SDSOM map, the margin between setosa and the other two classes is significant, and the linearly inseparable virginica is more tightly grouped than in SOM (Fig. 6 left). These factors give a classification advantage to SDSOM of 98.8% as opposed to 95.7%, and the Studentś t test indicates a 99.5% probability of the higher SDSOM rate being statistically significant. The SDNN (without a map but with d and s layers) performs well on this data set giving 100% accuracy on the test set.

The average separation on the Wine data map of the classes is larger in SDSOM (see Fig. 7), and the classification is 91.7% as opposed to 85% with SOM. This increase is 99.99% likely to be statistically significant. The additional layer (d layer) of SDNN allows it to perform very well on this data.

The OCR data maps for SDSOM and SOM are shown in Fig. 8 left and right respectively. The accuracy of classification for both methods for the test set is 97.3%. In common with the Iris data set the SOM map (Fig. 8 right) presents clearly the

different classes but fills the entire space with no margin between the classes, while in the SDSOM map (Fig. 8 left) the classes are more tightly grouped and with larger margins in between them.

The results for the NLP data show that the average correct classifications for the ND and PD using SDSOM are slightly higher than the ones obtained with SOM, although these differences are not statistically significant. The lower performance of both SDSOM and SOM is due to the number of input patterns for each of the symbols, which varies greatly and is insufficient for effective training in some cases. For example, symbols E, S& and S+ only have 2 input patterns corresponding to them. Chance dictates that all of the input patterns may happen to be used exclusively as either testing or training data. When these inputs are selected for testing, e.g. S+ and S&, unsurprisingly they tend to be recognized as S. The SDNN performs better on this task [27]. It uses the performance guided version of snap-drift, whereby the mode is swapped only when performance on an *s* node declines.

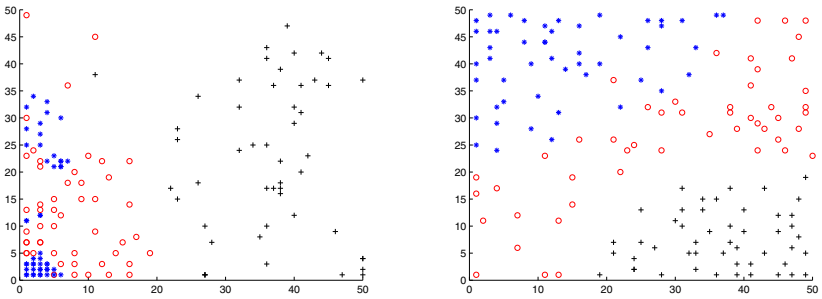

Fig. 6 SDSOM and SOM 50x50 applied to Iris Data set o (verisicolor) + (setosa) * (virginica)

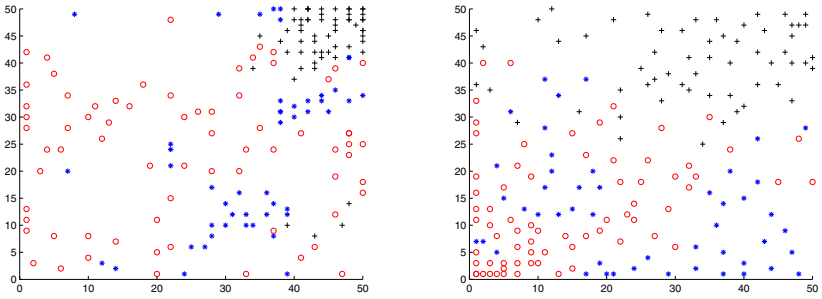

Fig. 7 SDSOM and SOM 50x50 applied to Wine Data set + (class 1) o (class 2) * (class 3)

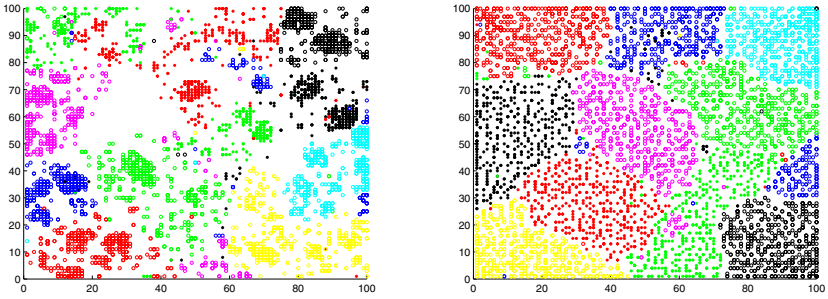

Fig. 8 SDSOM and SOM 100x100 applied to OCR Data set. The different colours represent the 10 different classes (digits 0 to 9)

4.1.3 Application of SDNN to E-Learning

Snap-Drift has been applied to online formative assessments of multiple choice questions with the purpose of providing feedback based on the common features in groups of answers [32], [9]. When students attempt on-line formative assessments they generate data that is invaluable for understanding their learning. That data is generally lost. However it can be captured, analysed and used as the basis for providing immediate feedback to the students as well as providing lecturers and tutors with a detailed picture of the learning of their students.

There are many studies investigating the role of different types of feedback in web-based assessments that report positive results from the use of Multiple Choice Questions (MCQs) in online tests for formative assessments (e.g. [4], [15], [36]). In these studies it is assumed that all the possible errors for a question can be predicted and a generic and focused feedback can be written for that question. However, this kind of feedback relates to a specific question rather than a combination of questions.

The diagnostic feedback developed here differs in that it does not reveal which questions were wrong; instead, the students are encouraged by the feedback to reflect on misunderstood concepts (that relate to their combination of errors on all the questions), and then to attempt the test again. Predicting all possible mistakes and writing generic and focused feedback for a combination of questions would be a daunting task and would not be feasible for large test banks (2 questions with 5 possible answers creates 25 possible answer combinations; 5 questions creates 3125 combinations, and so on). As the question bank grows the number of possible answer combinations increases exponentially, so that automation is essential for at least part of the process.

The neural network and in particular the Snap-Drift approach can address these problems by providing an efficient means of discovering a relatively small numbers of groups of similar answers so that responses can be targeted to the answers given by a very wide range of students with different states of knowledge. First the Snap-Drift neural network (SDNN) is trained in order to learn the different categories

of student answers. The data used for training can be collected from students' responses for questions on a particular topic in a subject from the previous cohorts of students. In order to pre-process the data each response from the students is encoded into binary form, in preparation to be presented as input patterns for the SDNN. Here is an example of a possible format for 5 questions

$$A = 00001;\ B = 00010;\ C = 00100;\ D = 01000;\ E = 10000$$

and, a response such as $[D,D,C,B,A]$ will be encoded as

$$[0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1].$$

During training, on presentation of each input pattern, the SDNN will learn to group the input patterns according to their general features. The groups are recorded, and represent different states of knowledge in relation to a given topic, in the sense that responses within that group share a particular combination of answers to certain questions. In other words, they contain the same mistake(s) and/or answer the same question(s) correctly. For example, for one group every response might have in common answer B to question 3 and answer A to question 4. The other answers to the other questions will vary within the group, but the group is formed by the neural network based on the commonality in some question answers (two of them in that case). From one group to another the precise number of common responses varies between 2 and the number of questions.

The training relies upon having representative training data. The number of responses required to train the system so that it can generate reasonable groups, varies from one domain to another. When new responses are still creating new groups, more training data is required. Once new responses are not creating new groups, it is because those new responses are similar to previous responses, and enough responses to train the system reliably are already available.

Snap-drift is suitable because it is an unsupervised, easy-to-apply, quick and effective means of discovering groupings, and is capable of discovering both clearly separable clusters (drift) and groups that are characterized by precise features that may represent only a fraction of the structure of patterns (snap). After training the SDNN provides grouped of answers and their common features. Suitable feedback is written for each group of answers. A particular feedback derives from the responses to several questions and is therefore not tied to any particular question, and so the learner is encouraged to retake the same test, receiving different feedbacks depending on their evolving state of knowledge.

SDNN is integrated with an on-line system of multiple choice questions tests. Students login into the system with their student id numbers. The student responses, time and student id are recorded in the database after each student's submission of answers. The students are prompted to select a module and a topic and this leads to the screen with a specific set of multiple choice questions. On submission of the answers the system converts these into a binary vector which is fed into the SDNN. The SDNN produces a group number and the system retrieves the corresponding feedback for this group from the feedback file and sends it to the student's browser.

They are then prompted to go back and try the same questions again or to select a different topic. Responses, recorded in the database, can also be used for monitoring student progress and for identifying misunderstood concepts that can be addressed in subsequent face-to-face sessions. The collected data facilitates analysis of how the feedback influences the learning of individual students and it can be used for retraining the neural network. Subsequently the content of the feedback can be improved. Once designed, MCQs and feedbacks can be reused for subsequent cohorts of students.

5 Conclusions and Future Work

Combining two modes of unsupervised learning and self-organisation in one neural network produces results that are not achievable with single modes. Modal learning with up to four modes, combining snap-drift, the delta rule, and an adaptive function method has demonstrated the potential for combining several modes [18]. Future work needs to explore mechanisms for controlling and optimising the selection of modes in real time. Performance guided mode switching has proved effective [25], [35] where a periodic or occasional system performance measure is available. Toggling between modes every epoch works well for two unsupervised modes as illustrated in this chapter. However, with multiple unsupervised modes toggling would not be an option. Borrowing an idea from real-time systems, a queue of modes could be maintained with an abandoned mode passing to the back of the queue. Alternatively, performance measures associated with each mode could be used as the priority values to create a queue with the most recently effective modes at the front of the queue.

Appendix

Matlab Code Snap-Drift

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sd =
   SnapDrift(input,dlayer,slayer,dnode,threshold,learning_rateD,learning_rateS)
%SnapDrift class constructor
%training patterns matrix
sd.input = input;
%the number of nodes in the d layer
sd.dlayer = dlayer;
%number of nodes in the s layer
sd.slayer = slayer;
%dnode is D, the number of winning nodes in the d layer
sd.dnode = dnode;
%threshold is the quality assurance threshold
sd.threshold = threshold;
sd.learning_rateD = learning_rateD;
sd.learning_rateS = learning_rateS;
[sd.no_pattern, sd.ilayer] = size(sd.input);
```

```
%Initialise and normalise weights between input
%and d layer.
for j = 1:sd.dlayer
    r = floor(rand(1)*sd.no_pattern)+1;
    %initialise weights with randmoly selected patterns
    sd.weights_ilayer_dlayer(:,j) = sd.input(r,:);
end
for j = 1:sd.dlayer
    norm_weight = norm(sd.weights_ilayer_dlayer(:,j));
    sd.weights_ilayer_dlayer(:,j) =
      sd.weights_ilayer_dlayer(:,j)/norm_weight;
end

%Initialise and normlaise weights between d layer
%and s  (output) layer
sd.weights_dlayer_slayer = ones(sd.dlayer,sd.slayer);
for j = 1:sd.slayer
    norm_weight = norm(sd.weights_dlayer_slayer(:,j));
    sd.weights_dlayer_slayer(:,j) =
     sd.weights_dlayer_slayer(:,j)/norm_weight;
end
%Initilalise output layer s (for large D, initialise some weights
%to 1's, with probability D/d)
sd.output = zeros(sd.slayer,1);
end %end function sd

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sd = train(sd,epoch)
for q = 1:epoch
   for counter = 1:sd.no_pattern
      dlayer_input = sd.input(counter,:)*sd.weights_ilayer_dlayer;
      %input from dlayer to slayer
      sInput = zeros(sd.dlayer,1);
      %learning between d layer and input layer on D most active nodes
      for s = 1:sd.dnode
          [maxVal, ind] = max(dlayer_input);
          dInput = sd.input(counter,:)';
      sd.weights_ilayer_dlayer =
           snap_drift_learn(sd.learning_rateD,sd.weights_ilayer_dlayer,q,
           ind,dInput);
          dlayer_input(ind) = 0;
          sInput(ind) = 1;
      end %for sd.dnode
      %learning between d layer and s layer
      slayer_input = (sInput)'*sd.weights_dlayer_slayer;
      [maxVal, ind] = max(slayer_input);
      if (maxVal > sd.threshold)
          sd.weights_dlayer_slayer =
            snap_drift_learn(sd.learning_rateS,sd.weights_dlayer_slayer,
            q,ind,sInput);
          sd.output(ind) = 1;
      else
         j = 1; %recruit a new, as yet untrained, s node
         while (j<=sd.slayer & sd.output(j)>0)
            j = j + 1;
         end %end while
         if (j<=sd.slayer)
             sd = snap_drift_ds_layer(sd,q,j,sInput);
             sd.output(j) = 1;
         end
     end %if threshold
 end %for counter
end %for epoch
end %train
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function weights = snap_drift_learn(lr,weights,q,ind,input)
if (mod(q,2) == 0)
   %adapt weights of the winning node using snap
   weights(:,ind) = min(weights(:,ind),input);
else
   %adapt weights of the winning node using drift
   weights(:,ind) = weights(:,ind) + lr*(input - weights(:,ind));
end %end if
norm_weight = norm(weights(:,ind));
if (norm_weight)>0
   weights(:,ind) = weights(:,ind)/norm_weight;
end
end %snap_drift_learn

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function label(sd,train_class,input_test,test_class,clNum)
%label s nodes based on response to training data data
%Set class frequency matrix of s nodes to 0
committed_class_counter = zeros(sd.slayer,clNum);
%For each pattern increment its class counter for the winning s node
for counter = 1:sd.no_pattern
    dlayer_input_train = sd.input(counter,:)*sd.weights_ilayer_dlayer;
    %input from dlayer to slayer
    sInput = zeros(sd.dlayer,1);
    for s = 1:sd.dnode
        [maxVal, ind] = max(dlayer_input_train);
        dlayer_input_train(ind) = 0;
        sInput(ind) = 1;
    end %for sd.dnode
    slayer_input = (sInput)'*sd.weights_dlayer_slayer;
    [maxVal, ind] = max(slayer_input);
    committed_class_counter(ind,train_class(counter)+1) =
             committed_class_counter(ind,train_class(counter)+1)+1;
end %for counter
error_train = 0;
output_map = zeros(sd.slayer,1);
%Find the maximum class counter for each s node and assign that as its
%class in the output_map array
for s = 1:sd.slayer
    [maxVal,ind] = max(committed_class_counter(s,:));
    if (committed_class_counter(s,ind)>0)
        output_map(s) = ind;
    end
    for j=1:clNum
        if (j ~= ind)
            error_train = error_train + committed_class_counter(s,j);
        end
    end
end
% % train error
error_train = error_train/sd.no_pattern;

%label test data
[m,n] = size(input_test);
test_error = 0;
output_test_map = zeros(sd.slayer,1);
committed_class_counter = zeros(sd.slayer,clNum);
%For each pattern in the test set
%find the most active s node with a class identity in the output_map
for counter = 1:m
    dlayer_input_test = input_test(counter,:)*sd.weights_ilayer_dlayer;
    %input from dlayer to slayer
    sInput = zeros(sd.dlayer,1);
    for s = 1:sd.dnode
        [maxVal, ind] = max(dlayer_input_test);
        dlayer_input_test(ind) = 0;
```

```
        sInput(ind) = 1;
    end %for sd.dnode
    slayer_input = (sInput)'*sd.weights_dlayer_slayer;
    [maxVal, ind] = max(slayer_input);
    while (output_map(ind) == 0)
        slayer_input(ind) = 0;
        [maxVal, ind] = max(slayer_input);
    end
    if (output_map(ind) - (test_class(counter)+1))>0
        test_error =  test_error + 1;
    end
    output_test_map(ind) = output_map(ind);
end %for counter
% % test error
test_error = test_error/m;
end %label function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Sample use of the Snap Drift Matlab code
%Save all functions in separate files named as the function names
%
%load the data (e.g. iris data)
input_train = load('irisTrain.txt');
input_test = load('irisTest.txt');
train_class = load('irisTrainClass.txt');
test_class = load('irisTrestClass.txt');

%Instantiate the SnapDrift object with suitable parameter values
%Set learning constants in the range 0,1
%D (dnode) is the number of winning dlayer nodes, and therefore
%the number of features required for categorisation or
%classification in the s layer
%The size of the d (dlayer) is normally much greater than D
%Threshold: because weights are normalised, thresholds of greater than 1
%tend to force additional s categories.
%It does not matter if the max number of s nodes is greater
%than required, snap-drift recruits new s nodes only when required.

sd = SnapDrift(input_train,50,10,10,2.5,0.1,0.2);

%call the train method on the sd object
sd = train(sd,50);
%call the label method if classification required
%and classes of training data are available
classNum = 3; % 3 classes in this case
label(sd,train_class,input_test,test_class,classNum);
```

References

1. Alpaydin, E., Kaynak, C.: Cascading Classifiers. Kybernetika 34, 369–374 (1998)
2. Burge, P., Shawe-Taylor, J.: An Unsupervised Neural Network Approach to Profiling the Behavior of Mobile Phone Users for Use in Fraud Detection. Journal of Parallel and Distributed Computing 61(7), 915–925 (2001)
3. Carpenter, G.A., Grossberg, S.: Adaptive resonance theory. In: Arbib, M.A. (ed.) The Handbook of Brain Theory and Neural Networks, Second Edition, 2nd edn., pp. 87–90. MIT Press, Cambridge (2003)
4. Dafoulas, G.A.: The role of feedback in online learning communities. In: Fifth IEEE International Conference on Advanced Learning Technologies, pp. 827–831 (2005)

5. Donelan, H., Pattinson, C., Palmer-Brown, D.: The Analysis of User Behaviour of a Network Management Training Tool using a Neural Network. Journal of Systemics, Cybernetics and Informatics 3(5) (2006)
6. Dotan, Y., Intrator, N.: Multimodality exploration by an unsupervised projection pursuit neural network. IEEE Transactions on Neural Networks 9(3), 464–472 (1998)
7. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis, p. 218. John Wiley & Sons, New York (1973)
8. Ekpenyong, F., Palmer-Brown, D., Brimicombe, A.: Extracting road information from recorded GPS data using snap-drift neural network. Neurocomputing 73, 24–36 (2009)
9. Fernandez Aleman, J.L., Palmer-Brown, D., Jayne, C.: Effects of Response Driven Feedback in Computer Science Learning. IEEE Transactions on Education 99 (2010), doi:10.1109/TE.2010.2087761
10. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annual Eugenics 7, Part II, 179–188 (1936); also in Contributions to Mathematical Statistics. John Wiley, NY (1950)
11. Forina, M., Lanteri, S., Armanino, C., et al.: PARVUS–an extendible package for data exploration, classification and correlation. Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy (1991)
12. Garside, R., Leech, G., Varadi, T.: Manual of Information to Accompany the Lancaster Parsed Corpus. University of Oslo (1987)
13. Gupta, L., McAvoy, M.: Investigating the prediction capabilities of the simple recurrent neural network on real temporal sequences. Pattern Recognition 33(i12), 2075–2081 (2000)
14. Hebb, D.O.: The organization of behavior. Wiley & Sons, New York (1949)
15. Higgins, E., Tatham, L.: Exploring the potential of Multiple Choice Questions in Assessment. Learning & Teaching in Action 2(1) (2003)
16. Horton, P., Nakai, K.: A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins. Intelligent Systems in Molecular Biology, 109–115 (1996)
17. Kang, M., Palmer-Brown, D.: A Multilayer ADaptive FUnction Neural Network (MADFUNN) for Analytical Function Recognition. IJCNN (2006); part of the IEEE World Congress on Computational Intelligence, WCCI 2006, Vancouver, BC, Canada, pp. 1784–1789 (2006)
18. Kang, M., Palmer-Brown, D.: A Modal Learning Adaptive Function Neural Network Applied to Handwritten Digit Recognition. Information Sciences 178(20), 3802–3812 (2008)
19. Kaynak, C.: Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition. MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University (1995)
20. Kohonen, T.: Self-organised formation of topologically correct feature maps. Biological Cybernetics 43 (1982)
21. Kohonen, T.: Learning Vector Quantisation. Helsinki University of Technology, Laboratory of Computer and Information Science, Report TKK-F-A-601 (1986)
22. Kohonen, T.: Learning Vector Quantisation. Neural Networks 1, 303 (1988)
23. Kohonen, T.: Self-Organisation and Asssociative Memory, 3rd edn. Springer, Heilderberg (1989)
24. Kohonen, T.: Improved Versions of Learning Vector Quantization. In: Proc. of IJCNN 1990, Washington, DC, vol. 1, pp. 545–550 (1990)
25. Lee, S.W., Palmer-Brown, D., Roadknight, C.M.: Performance guided Neural Network for Rapidly Self Organising Active Network Management. Neurocomputing 61C, 5–20 (2004a)

26. Lee, S.W., Palmer-Brown, D., Roadknight, C.M.: Reinforced Snap Drift Learning for Proxylet Selection in Active Computer Networks. In: Proc. of IJCNN 2004, Budapest, Hungary, vol. 2, pp. 1545–1550 (2004b)
27. Lee, S.W., Palmer-Brown, D.: Snap-drift learning for phrase recognition. In: Proc. IEEE IJCNN 2005, Montréal, Québec, Canada, vol. 1, pp. 588–592 (2005)
28. Lee, S.W., Palmer-Brown, D.: Phonetic Feature Discovery in Speech Using *Snap-Drift* Learning. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006, Part II. LNCS, vol. 4132, pp. 952–962. Springer, Heidelberg (2006)
29. Mayberry III, M.R., Miikkulainen, R.: SARDSRN: A Neural Network ShiftReduce Parser. In: Proc. of the 16th IJCAI, Stockholm, Sweden, pp. 820–825 (1999)
30. MacKay, D.J.C.: Bayesian methods for supervised neural networks. In: Arbib, M.A. (ed.) The Handbook of Brain Theory and Neural Networks, pp. 144–149. MIT Press, Cambridge (1998)
31. Palmer-Brown, D., Tepper, J., Powell, H.: Connectionist Natural Language Parsing. Trends in Cognitive Sciences 6(10), 437–442 (2002)
32. Palmer-Brown, D., Draganova, C., Lee, S.W.: Snap-Drift Neural Network for Selecting Student Feedback. In: Proc. IJCNN 2009, Atlanta, USA, pp. 391–398 (2009)
33. Palmer-Brown, D., Lee, S.W., Draganova, C., Kang, M.: Modal Learning Neural Networks. WSEAS Transactions on Computers 8(2), 222–236 (2009)
34. Palmer-Brown, D., Draganova, C.: Snap-Drift Self Organising Map. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) ICANN 2010, Part II. LNCS, vol. 6353, pp. 402–409. Springer, Heidelberg (2010)
35. Palmer-Brown, D., Draganova, C.: Recurrent Snap Drift Neural Network for Phrase Recognition. In: WCCI 2010 IEEE World Congress on Computational Intelligence, IJCNN 2010, Barcelona, Spain, pp. 3445–3449 (2010)
36. Payne, A., Brinkman, W.-P., Wilson, F.: Towards Effective Feedback in e-Learning Packages: The Design of a Package to Support Literature Searching, Referencing and Avoiding Plagiarism. In: Proceedings of HCI 2007 Workshop: Design, Use and Experience of e-Learning Systems, pp. 71–75 (2007)
37. Ritter, H., Kohonen, T.: Self-Organizing Semantic Maps. Biological Cybernetics 61, 241–254 (1989)
38. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagation errors. Nature 323, 533–536 (1986)
39. Rushton, J.N.: Natural Language Parsing using Simple Neural Networks. In: Proc. of MLMTA, Las Vegas, Nevada, pp. 137–141 (2003)
40. Tepper, J., Powell, H.M., Palmer-Brown, D.: A Corpus based Connectionist Architecture for Large scale Natural Language Parsing. Connection Science 14(2), 93–114 (2002)
41. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
42. Webb, A.R., Lowe, D.: A hybrid optimisation strategy for adaptive feed-forward layered newtorks. RSRE Memorandum 4193, Royal Signals and Radar Establishemnt, St Andrews Road, Malvern, UK (1988)
43. Widrow, B., Hoff, M.E.: Institute of Radio Engineers WESCON Convention Record, Adaptive switching circuits, pp. 96–104. Institute of Radio Engineers, New York (1960)