Generalising Symbolic Knowledge in Online Classification and Prediction

Richard Dazeley and Byeong-Ho Kang

School of Information Technology and Mathematical Sciences,
University of Ballarat, Ballarat, Victoria 7353, Australia
School of Computing and Information Systems,
University of Tasmania, Hobart, Tasmania, 7001
r.dazeley@ballarat.edu.au, bhkang@utas.edu.au

Abstract. Increasingly, researchers and developers of knowledge based systems (KBS) have been incorporating the notion of context. For instance, Repertory Grids, Formal Concept Analysis (FCA) and Ripple-Down Rules (RDR) all integrate either implicit or explicit contextual information. However, these methodologies treat context as a static entity, neglecting many connectionists' work in learning hidden and dynamic contexts, which aid their ability to generalize. This paper presents a method that models hidden context within a symbolic domain in order to achieve a level of generalisation. The method developed builds on the already established Multiple Classification Ripple-Down Rules (MCRDR) approach and is referred to as Rated MCRDR (RM). RM retains a symbolic core, while using a connection based approach to learn a deeper understanding of the captured knowledge. This method is applied to a number of classification and prediction environments and results indicate that the method can learn the information that experts have difficulty providing.

Keywords: Hidden context, knowledge based systems, knowledge representation, ripple-down rules, situation cognition.

1 Introduction

Traditionally, knowledge based approaches have been based on the physical symbol hypothesis [1] which is built around the idea that knowledge is a substance that exists. However, after numerous failed systems some researchers have revised these concepts of knowledge and moved towards a situation-cognition (SC) based view. The SC view revolves around the premise that knowledge is generated at the time of its use. This implies that the existence of knowledge is based on the context of a given situation [2, 3]. A few methodologies, such as Formal Concept Analysis (FCA) [4], Repertory Grids [5] and Ripple-Down Rules (RDR) [6], have adopted a weak SC position by including contextual information. These approaches either incorporated the context directly in the knowledge itself or in the structure the knowledge was represented. These methods have been reasonably successful, however, they assume that the context is *a priori*, and therefore, deductive [7]. This assumption leads to static representations of contextual based

D. Richards and B.-H. Kang (Eds.): PKAW 2008, LNAI 5465, pp. 91–108, 2009.
© Springer-Verlag Berlin Heidelberg 2009

knowledge. However, context in certain situations could be considered *a posteriori*, and therefore, inductive [7].

The aim of this paper is to present an algorithm that moves away from these contextually static representations and instead heads towards an intermediate SC [8] view by handling hidden and dynamic contexts. This involves incorporation similar behaviour to the traditional strengths of connection based approaches while still being able to acquire and retain knowledge quickly. The result is a system that learns quickly and is still able to generalise effectively. The results in this paper investigates the method's ability to classify cases quickly in an online environment and to predict continuous values. This notion of a symbolic based system capable of finding hidden contextual information through the generalisation of captured knowledge, led to the notion of combining a Knowledge Based System (KBS) with an Artificial Neural Network (ANN). The KBS selected for use in this paper was MCRDR, as this is currently one of the methodologies most capable of modelling multiple contexts [9].

This paper is broken into three main sections. The first section will provide a background on MCRDR. This is followed by a discussion of the algorithm developed. Lastly, extensive results will be given, detailing the systems ability to discover more knowledge than that provided by the expert in both an online environment and in predicting continuous values.

2 Multiple Classification Ripple-Down Rules (MCRDR)

Ripple-Down Rules is a maintenance centred methodology for a KBS based approach using the concept of fault patching [10] and was first proposed by Compton and Jansen in 1988 [6]. It utilises a binary tree as a simple exception structure aimed at partially capturing the context that knowledge is obtained from an expert. It was assumed that the context was the sequence of rules that had evaluated to provide the given conclusion [6, 11-15]. Therefore, if the expert disagrees with a conclusion made by the system they can change it by adding a new rule. However, the new rule will only fire if the same path of rules is evaluated [13].

Ripple-Down Rules has been shown to be a highly effective tool for knowledge acquisition (KA) and knowledge maintenance (KM). However, it lacks the ability to handle tasks with multiple possible conclusions. Multiple Classification Ripple-Down Rules (MCRDR) aim was to redevelop the RDR methodology to provide a general approach to building and maintaining a Knowledge Base (KB) for multiple classification domains, while maintaining all the advantages from RDR. Such a system would be able to add fully validated knowledge in a simple incremental contextually dependant manner without the need of a knowledge engineer [16, 17].

The new methodology developed by [16] is based on the proposed solution by [12, 13]. The primary shift was to switch from the binary tree to an *n-ary* tree representation. The context is still captured within the structure of the KB and explanation can still be derived from the path followed to the concluding node. The main difference between the systems is that RDR has both an *exception* (true) branch and an *if-not* (false) branch, whereas MCRDR only has exception branches. The false branch instead simply cancels a path of evaluation. Like with RDR, MCRDR nodes each contain a rule and a conclusion if the rule is satisfied. Each, however, can have any number of child branches.

Inference occurs by first evaluating the root and then moving down level by level. This continues until either a leaf node is reached or until none of the child rules evaluate to true. Each node tests the given case against its rule. If false it simply returns, X (no classification). However, if this node's rule evaluates to true then it will pass the case to all the child nodes. Each child, if true, will then return a list of classifications. Each list of classifications is collated with those sent back from the other children and returned. However, if none of the children evaluate to true, and thus they all return X, then this node will instead return its classification. Like with RDR the root node's rule always evaluates to true, ensuring that if no other classification is found then a default classification will be returned.

Knowledge is acquired by inserting new rules into the MCRDR tree when a misclassification has occurred. The new rule must allow for the incorrectly classified case, identified by the expert, to be distinguished from the existing stored cases that could reach the new rule [18]. This is accomplished by the user identifying key differences between the current case and each of the rules' cornerstone cases. A cornerstone case is a case that was used to create a rule and was also classified in the parent's node, or one of its child branches, of the new node being created. This is continued for all stored cornerstone cases, until there is a composite rule created that uniquely identifies the current case from all of the previous cases that could reach the new rule. The idea here is that the user will select differences that are representative of the new class they are trying to create [18].

3 Methodology

The approach developed in this paper is a hybrid methodology, referred to as Rated MCRDR (RM), combining MCRDR with a function fitting technique, namely an artificial neural network (ANN). This hybridisation was performed in such a way that the function fitting algorithm learns patterns of fired rules found during the inferencing process. The position of rules and conclusions in the MCRDR structure represents the context of the knowledge, while the network adjusts its function over time as a means of capturing hidden relationships. It is these relationships that represent the methodology's hidden contexts.

This amalgamation appears simplistic but is by no means trivial. The fundamental difficulty was finding a means for taking the inferenced results from MCRDR and coding an input sequence for the network. The problem is caused by MCRDR's structure constantly expanding. Therefore, the network's input space must also grow to match. However, previous work in the function-fitting literature has not attempted to develop a network capable of increasing its input space. The problem arises from the internal structure of neural networks where, as the input space is altered, the interconnections between neurons and the associated weights are also changed.

Basically, the system discussed in this paper is designed to recognise patterns of rules for particular cases and to attach weightings to these observed patterns. These patterns exist because there is either a conscious or subconscious relationship between these classes in the expert's mind. Therefore, the captured pattern of rules in their static context is effectively a type of hidden or unknown context. This now discovered context can be given a value representing its contribution to a particular task.

1. Pre-process Case
Initialise Case *c*
c ← Identify all useful data elements.

2. Classification
Initialize *list* to store classifications
Loop
If child's rule evaluates Case c to *true*
list ← goto step 2 (generate all classifications in child's branch).
Until no more children
If no children evaluated to true then
list ← Add this nodes classification.
Return *list*.

3. Evaluate Case
$\overline{x}$ ← Generate input vector from *list*.
ANN ← $\overline{x}$
$\overline{v}$ ← *ANN* output value.

4. Return RM evaluation
Return *list* of classifications for case *c* and
Value $\overline{v}$ of case *c*.

Fig. 1. Pseudo code algorithm for RM

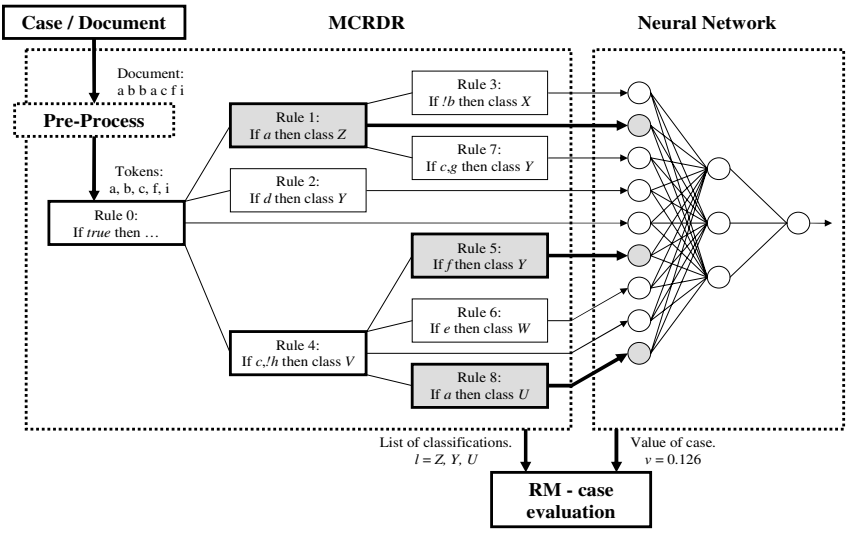

Fig. 2. RM illustrated diagrammatically

The full RM algorithm, given in pseudo-code in Fig 1 and shown diagrammatically in Fig 2, consists of two primary components. Firstly, a case is pre-processed to identify all of the usable data elements, such as stemmed words or a patient's pulse. The data elements are presented to the standard MCRDR engine, which classifies them according to the rules previously provided by the user. Secondly, for each attribute, rule or class identified, an associated input neuron in the neural network will fire. The

network produces a vector of output values, $\bar{v}$, for the case presented. The system, therefore, essentially provides two separate outputs; the case's classifications and the associated set of values for those classifications.

Fig 2 shows a document classification and storage system where documents are also rated to judge their immediate importance. In this example a document with the tokens {a b b a c f i} has been pre-processed to a set of unique tokens {a, b, c, f, i}. The case is then presented to the MCRDR component of the RM system, which ripples the case down the rule tree finding three classifications: Z, Y, and U; from the terminating rules: 1, 5, and 8. In this example, which is using the Terminating Rule Association (TRA) method (section 3.3), the terminating rules then cause the three associated neurons to fire. The input pattern then feeds forward through the neural network producing a single value of 0.126. Thus, this document has been allocated a set of classifications that can be used to store the document appropriately, plus a rating indicating the importance of the document.

3.1 Learning in RM

Learning in RM is achieved in two ways. Firstly, the value for each corresponding value for receives feedback from the environment concerning its accuracy. Thus, a system using RM must provide some means of either directly gathering or indirectly estimating each elements value. For example, in an email application where the system was required to order the documents in the order of importance, the amount of reward given to the network could be based on the order the articles are read by the user or whether the user prints, saves, replies, forwards or deletes the email. How the network actually learns is either using the standard backpropagation approach using a sigmoid thresholding function, or, using the RM specific algorithm described in section 3.4. The MCRDR component still acquires knowledge in the usual way (section 2). Therefore, in the basic RM implementation the expert must still review cases and check classifications are correct.

3.2 Artificial Neural Network Component

The ANN used is based on the backpropagation algorithm and was designed to be plugged on to the end of the MCRDR component. Integration of the MCRDR and ANN components is carried out by codifying the relevant features taken from MCRDR and converting these into a single input array of values, $\bar{x}$, which is to be provided to the ANN for processing. Two methods were used in this paper referred to as the Rule Path and Terminating Rule Association methods. The rule path method provided an input for every rule that fired while the terminating method only fired input nodes associated with the final rule that was reached by the inferencing process.

3.3 Adding Neurons

As the input space grows new input nodes need to be added to the network in such a way that does not damage already learnt information. A number of methods were developed for altering the input space, such as backpropagation and radial basis function networks, as well as non neural network methods such as Kernel based methods. This paper will discuss the most stable and effective method found. This particular

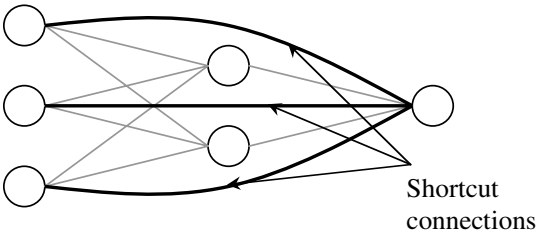

Fig. 3. Network structure of RM showing a single hidden layer network with shortcut connections directly connecting the input nodes to the output nodes

method allows for non-linear relationships while being able to learn quickly when new inputs are added.

RM captures the initial information by directly calculating the required weight to provide us with the correct *weighted-sum* using a new learning rule referred to as the *single-step-Δ-initialisation-rule* (3.4.1). When applying this weight it must be done in so that does not affect any of the already learnt weights. Therefore, the network structure needed to be altered by adding shortcut connections (Fig 3) from any newly created input node directly to each output node and using these connections to carry the entire weight adjustment. When a new input node is added, additional hidden nodes also may be added. Therefore, connections must be added in the following places:

- From the new input node to all of the old hidden nodes.
- From all input nodes, new and old, to each of the new hidden nodes, if any.
- From each of the new hidden nodes, if any, to all of the output nodes.
- The shortcut connections from the new input node to all of the output nodes.

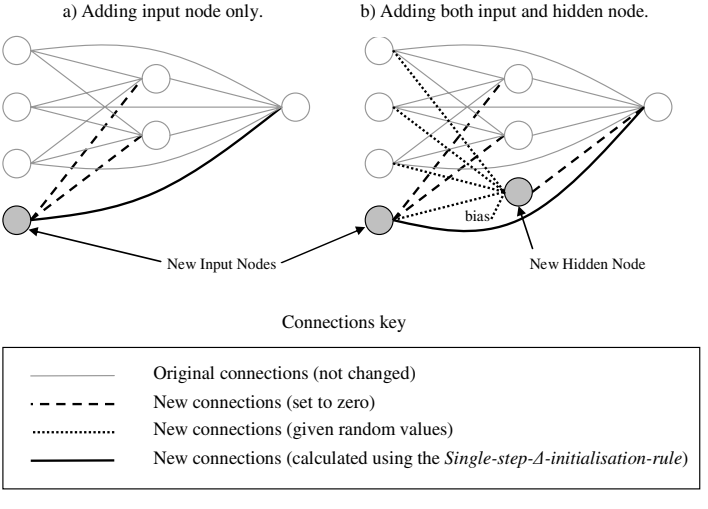

Fig. 4. Process used for adding new input and hidden nodes in RM. (a) shows how inputs are added by themselves. (b) shows how input and hidden nodes are added simultaneously.

The process for adding nodes and connections is illustrated in Fig 4. Each of these different groups of new connections requires particular start up values. First, the new connections from the new input node to all the old hidden nodes should be set to zero so that they have no immediate affect on current relationships. Occasionally, new hidden nodes are also required. These were added at a rate that maintained a number equivalent to half the amount of input nodes. If new hidden nodes were added then the connections from them to the output nodes should also be set to zero for the same reason as with the input nodes. In order for these connections to be trained, the output from the new hidden nodes must be non-zero. Thus, the new connections from all the input nodes to the new hidden nodes, and their biases, are given random values. Finally, the new shortcut connections are given a value calculated using the *single-step-Δ-initialisation-rule*.

3.3.1 The Single-Step-Δ-Initialisation-Rule

The *single-step-Δ-initialisation-rule* directly calculates the required weight for the network to step to the correct solution immediately. This is accomplished by reversing the feedforward process back through the inverse of the symmetric sigmoid. This calculation is performed by finding the weight needed, using equation (1), for the new input connection, w_{no}. This has the requirement that the system does not attempt to set the value of the output outside the range $-0.5 > (f(net) + \delta) > 0.5$ as this will cause an error. The value for *net* for each output node, o, was previously calculated by the network during the feedforward operation where there are $n>1$ input nodes and the n^{th} input node is our new input. Function f is the asymmetric sigmoid and δ is the amount of error. This is then divided by the input at the newly created input node, x_n, which is always *1* in this implementation, where there are $n>0$ input nodes and $o>0$ output nodes. Additionally, it is possible for the expert to add multiple new rules for the one case. In these situations the calculated weight is divided by the number of new features (attribute, rule or class), m. Finally, the equation is multiplied by the step-distance modifier, *Zeta (ζ)*.

$$w_{no} = \zeta \left[\left(\left(\log \left[\frac{f(net)_o + \delta_o + 0.5}{0.5 - (f(net)_o + \delta_o)} \right] \Big/ k \right) - \left[\left(\sum_{i=0}^{n-1} x_i w_{io} \right) + \left(\sum_{h=0}^{q} x_h w_{ho} \right) \right] \right) \Big/ m x_n \right] \quad (1)$$

The *Zeta (ζ)* modifier should always be set in the range $0 \le \zeta \le 1$. It is included to allow adjustments to whether a full step or partial steps should be taken for the new features. For instance, if ζ is 1 then the new weights will provide a full step and any future identical cases will give the exact correct answer. A lesser value for ζ causes new features to only receive a portion of their value. It was found in testing that the inclusion of the ζ modifier allows better performances in some situations.

This updating method is best understood by seeing what is occurring diagrammatically. Fig 6 shows an input pattern that had a weighted sum of 3.0 at the output node. This was passed through the symmetric sigmoid function, finding the output value 0.47. The correct output after a rule was added is -0.358. The correct weight for the new input node is calculated by feeding this target back through the inverse of the sigmoid function finding the value -1.8. Therefore, the new node's weight is the difference between these two values, giving -4.8.

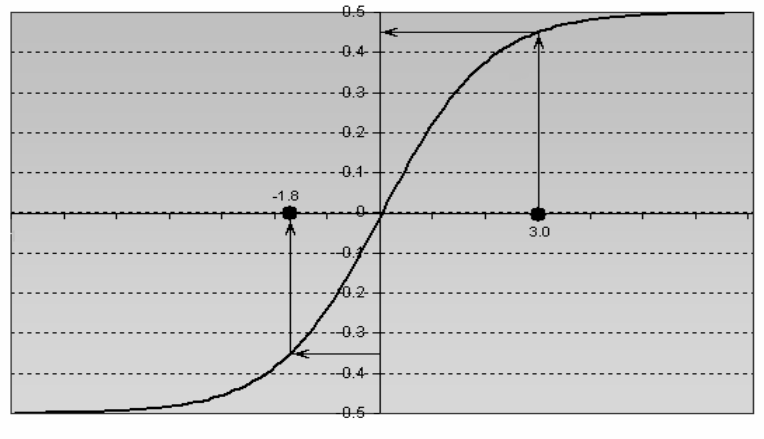

Fig. 5. Example of the *single-step-Δ-initialisation-rule* shown diagrammatically

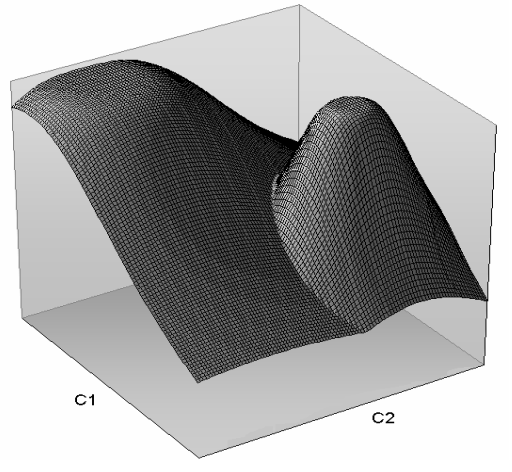

Fig. 6. Example of a randomly generated energy pattern used in the MCP simulated expert

4 Experiments and Results

This section's results are in two parts. First RM compares against the two underlying methodologies, MCRDR and a backpropagation neural network, in an online classification task. The second collection of results illustrates how RM compares against a backpropagation neural network in both on and off-line prediction. The aim of these results is to show how RM learns as fast as MCRDR, yet maintains the generalisation of a neural network. This section also contains a discussion of the simulated experts used for the experiments, along with the datasets used.

4.1 Experimental Method

In the first group of experiments the aim is to test RMs classification ability in an online environment compared to the underlying approaches. Therefore, the output from RMs ANN will consist of a vector, $\bar{v}$, of outputs. Each output will relate to one possible classification. There will be an equal number of outputs to the number of class types in the dataset being tested. If the output is positive then it will be regarded as providing that classification. The same output method is used for the backpropagation method being compared against. In the first collection of results presented in this paper each test used 10 different randomisations of the relevant datasets. The test investigates how the methods can correctly classify cases over time. In this test the entire dataset is broken up into small blocks, each $1/50^{th}$ of the original dataset, and passed through the system one group at a time. The system's performance is recorded after each group, showing how fast the system learns for each new batch of cases.

In the prediction domain RM and the ANN must output a single value, which must be as close to the expected value as possible. In the second collection of results presented in this paper each test used 10 different randomisations of the dataset. The first, *generalisation* test, divides each dataset into ten equal sized groups. Results are presented where $9/10^{ths}$ of the dataset are used for training and $1/10^{th}$ for testing. The size of the training set is then reduced incrementally in steps of $1/10^{th}$, down to $1/10^{th}$. The same $1/10^{th}$ set is always used as the test set. The *online* prediction test investigates how the methods can correctly predict values over time. Similar to the online classification test, the entire dataset is broken up into smaller blocks, each $1/50^{th}$ of the original dataset, and passed through the system one group at a time. The system's performance is recorded after each group. The value returned is then compared to the simulated expert's correct value. The absolute difference between these two values (*error*) is then averaged over all the cases in the data segment and logged.

4.2 Simulated Expertise

One of the greatest difficulties in KA and KBSs research is how to evaluate the methodologies developed [19]. The method used by the majority of RDR based research has been to build a simulated expert, from which knowledge can be acquired [19]. It is this approach that has been taken in this paper. This section will discuss the three simulated experts created for the tests performed in this paper.

4.2.1 C4.5 Simulated Expert

The only purpose of the simulated expert is to select which differences in a difference list are the primary ones. It uses its own KB to select the symbols that will make up the new KB. C4.5 [20] is used to generate the simulated expert's knowledge base. The resulting tree then classifies each case presented, just like our KB under development. If the KB being constructed, incorrectly classifies a case then the simulated expert's decision tree is used to find attributes within rules that led to the correct classification. This is similar to the *'smartest'* expert created by Compton et al [21].

4.2.2 Non-linear Multi-class Simulated Expert

The fundamental problem with the above simulated expert is that it requires an induction system, such as C4.5, to generate a complete KB prior to its use. This is a problem because no suitable system is available that can create such a tree for a multiple classification domain. However, the system being developed in this paper is primarily targeting domains with multiple classification domains. Therefore, a second simulated expert was created specifically designed for handling a particular multiple classification based dataset (4.3).

This heuristic based simulated expert has two stages in calculating classifications based on a cases attributes. The first stage uses a randomly generated table of values, representing the level that each attribute, $a \in A$, contributes to each class, $c \in C$. An example of an expert's attribute table used is shown in Table 1.

Table 1. Example of a randomly generated table used by the *non-linear multi-class* simulated expert. Attributes a - l are identified across the top, and the classes C1 – C6 down the left.

	A	b	c	d	e	f	g	h	i	j	k	l
C1	0	0	-1	3	0	0	0	0	0	0	-1	3
C2	0	0	0	-2	2	0	0	-2	0	0	1	0
C3	0	-2	1	0	0	0	0	0	0	1	0	-1
C4	-1	3	0	0	0	0	1	0	-1	0	0	0
C5	0	0	0	0	-2	2	-2	0	2	0	0	0
C6	2	0	0	0	0	-2	0	1	0	-2	0	0

To make the task sufficiently difficult for the systems to learn a second stage of the expert's classification process is to provide a non-linearity. A non-linear expert needs the attributes' contribution to classifications to vary according to what other attributes were in the case. This was achieved by selecting an even number of attributes and pairing them together for each of the classes. Once paired, they were given an increasing absolute value. Additionally, alternate pairs had their sign changed. This can best be understood by investigating the example shown in Table 2. Here it can be seen that for the class *C1*, the attribute pairs *{b, j}*, *{f, h}* and *{d, j}* have a positive influence, while *{a, l}*, *{h, i}* and *{a, k}* have a negative influence.

Table 2. Example of a randomly generated table of attribute pairs. The top numbers represent the positive or negative values for the pairs. Each class in this example has six pairs.

	1		-1		2		-2		3		-3	
C1	b	j	a	l	f	h	h	i	d	j	a	k
C2	g	b	c	f	e	h	a	b	k	d	g	k
C3	i	d	e	b	g	i	k	l	j	a	c	f
C4	l	a	c	i	j	a	i	l	f	h	j	a
C5	k	g	b	f	d	g	j	f	b	c	a	e
C6	c	l	h	j	a	c	j	b	g	k	d	e

Table 3. Two example cases being evaluated by the non-linear multi-class simulated expert

Case A = *{a, b, c, d}*							Case B = *{a, c, e, g}*						
Attributes	Classifications						Attributes	Classifications					
	1	2	3	4	5	6		1	2	3	4	5	6
a	0	0	0	-1	0	2	*a*	0	0	0	-1	0	2
b	0	0	-2	3	0	0	*c*	-1	0	1	0	0	0
c	-1	0	1	0	0	0	*e*	0	2	0	0	-2	0
d	3	-2	0	0	0	0	*g*	0	0	0	1	-2	0
{a, c}						2	*{a, c}*						2
{a, b}		-2					*{e, a}*					-3	
{b, c}					3								
Total	2	-4	-1	2	3	4	Total	-1	2	1	0	-7	4
Classified	✓	×	×	✓	✓	✓	Classified	×	✓	✓	×	×	✓

When a case is presented to the expert the class it belongs to is calculated by adding all the attribute values and there attribute pairs. The expert will then classify the case according to which classes provided a positive, > 0, total. When creating a new rule, the expert selects the attribute or attribute-pair from the difference list that distinguishes the new case from the cornerstone case to the greatest degree. Table 3, gives two example cases where each case has 4 attributes.

4.2.3 Multi-class-Prediction Simulated Expert

Testing RM using simulation has an added difficulty in the prediction domain. This is because available datasets do not give both symbolic knowledge and a target value instead of a classification. This could be partially resolved by assigning each classification a value. However, fundamentally this would still be a classification type problem. The approach taken in this paper was to develop a third simulated expert, which has two stages in calculating a value for a case based on a set of randomly generated attributes. The first stage uses a randomly generated table of values, in the same way as the first stage of the non-linear simulated expert described above. This classification stage is merely an intermediate step to finding a rating for the case. It is also used during knowledge acquisition for identifying relevant attributes in the difference lists. When creating a new rule, the expert selects the attribute from the difference list that distinguishes the new case from the cornerstone case to the greatest degree. This was achieved by locating the most significant attribute, either positively or negatively, that appeared in the difference list (see example in Table 1).

To fully push the system's abilities, the rating calculated by the simulated expert needs to generate a non-linear value across the possible classifications. The implementation used for prediction generates an energy space across the level of class activations, giving an energy dimensionality the same as the number of classes possible. Each case is then plotted on to the energy space in order to retrieve the case's value. First, the strength of each classification found is calculated. As previously discussed a case was regarded as being a member of a class if its attribute value was greater than *0*. However, no consideration was made to what was the degree of membership. In this expert the degree of the case's membership is calculated as a percentage, *p*, of membership using Equation 2.

$$p = t^{a} / \mathrm{t}^{\mathrm{m}} \tag{2}$$

This is simply the actual calculated total, t^a, divided by the maximum possible total, t^m, for that particular class. Extending the example from Table 3 for case A, classification C1, the total *2* is divided by the best possible degree of membership *6,* max value from row C1 in table 1, thereby, giving a percentage, p, membership of *33%*. This calculation is performed for each class. Each class then has a randomly selected point of highest value, or centre, c, which is subtracted from the percentage and squared, Equation 3. This provides a value which can be regarded as a distance measure, d, from the centre. This distance measure can be *stretched* or *squeezed*, widening or contracting the energy patterns around a centre, by the inclusion of a width modifier, w.

$$\mathrm{d} = w\,(\,p - c\,)^{2} \tag{3}$$

The classes' centres are combined to represent the point of highest activation for the expert, referred to as a *peak*. Therefore, if the square root of the sum of distances is taken then the distance from this combined centre can be found. This distance can then be used to calculate a lesser value for the case's actual rating. Therefore, as a case moves away from a *peak* its *value* decreases. Any function can be used to calculate the degree of reduction in relation to distance. In this paper a Gaussian function was used. Equation 4 gives the combined function for calculating a value for each possible peak, v^p, where n is the number of classes in the dataset.

$$v^{p} = \frac{1}{1 + e^{-0.5\left(1 \Big/ \sqrt{\sum_{j=0}^{n}\left(\left(\frac{t_j^a}{t_j^m}\right) w_j - c_j\right)^2}\right)}} - 0.5 \tag{4}$$

Finally, it is possible to have multiple peaks in the energy space. In such a situation each class has a centre for each peak. Each peak is then calculated in the same fashion as above, resulting in a number of values, one for each peak. The expert then simply selects the highest value as the case's actual rating. This rating method is best understood by looking at a three dimensional representation shown in Fig 6.

The third dimension, shown by the height, illustrates the value at any particular point in the energy space. This figure shows a dataset with only two possible classes, C1 and C2, and two peaks. A three class dataset cannot be represented pictorially. The advantage of this approach is that it generates an energy pattern that is nonlinear. At no location can a straight line be drawn where values are all identical.

4.3 Datasets

The method was tested using six datasets. The first five are standard datasets retrieved from the University of California Irvine Data Repository [22]. These five datasets were tested using the *C4.5* based simulated expert. The sixth dataset is a purpose designed randomised set and is used with the *non-linear multi-class* and *multi-class-prediction* simulated experts. Below is a list describing each of the five dataset used from the University of California Irvine Data Repository [22].

- **Chess –** has 36 attributes with a binary classification over 3196 cases. In each 1/50th group there are 63 cases.
- **Tic-Tac-Toe (TTT)** –has 9 attributes with a binary classification over 958 cases. In each 1/50th group there are 19 cases.
- **Nursery Database** – has 8 nominal attributes with 5 classifications over 12960 cases. In each 1/50th group there are 259 cases.
- **Audiology** – has 70 nominal-valued attributes with 17 classifications over only 200 cases. In each 1/50th group there are 4 cases.
- **Car Evaluation** – has 6 attributes with 4 classifications over 1728 cases. In each 1/50th group there are 34 cases.

The multi-class dataset builds cases by randomly selecting attributes from the environment. For instance, an environment setup for the example simulated experts used in section 4.2.2 would allow for 12 possible attributes. For the tests in this paper each case selected 6 attributes, giving a possible 924 different cases. There were also 6 possible conclusions. Therefore, in each 1/10th group used in the offline prediction there are 92 cases and 18 cases in each 1/50th group.

4.4 Online Classification

One of the main features RM was aiming to achieve from the use of the ANN was the ability to learn quickly and generalise well in an online environment. The results in this section investigate how RM compares with its two underlying methodologies in the online environment. Fig 7 (a) - (f) shows how RM, MCRDR and the ANN perform on the six datasets. Each point on the charts is an average of the previous 10 data segments (except 2 data segments for the Audiology dataset) which are then further averaged over the ten randomised runs. Each segment contains a random selection of cases, each 1/50th the size of the whole dataset. Error bars have been omitted to allow for greater readability.

These comparisons are powerful indicators of the advantages of RM over a standard backpropagation neural network when being applied in an online environment. On the chess, TTT and audiology datasets it can be seen that RM has learned as fast or nearly as fast as MCRDR. On the nursery and car evaluation datasets it was only between 3% and 6% below MCRDRs performance and overtime was narrowing this gap. This meets our original goal of gaining the speed of MCRDR's instantaneous learning as soon as a rule is added. In the multi-class results this same result can be observed, except it can also be seen how RM continued to learn after MCRDR had accrued all its possible knowledge.

MCRDR's failure to continue to learn after its initial gains was a point of concern in the multi-class test. However, after investigation, it was found to be caused by two main factors. Firstly, for a case to be correctly classified it must get all six classes correct. Therefore, MCRDR's performance was not as poor as it first appears. Secondly, there is one unusual problem in the MCRDR rule creation and validation phase. It is possible that when an expert attempts to create a rule there may be no suitable attributes available. This generally only occurs on the later difference lists generated when there are multiple cornerstone cases.

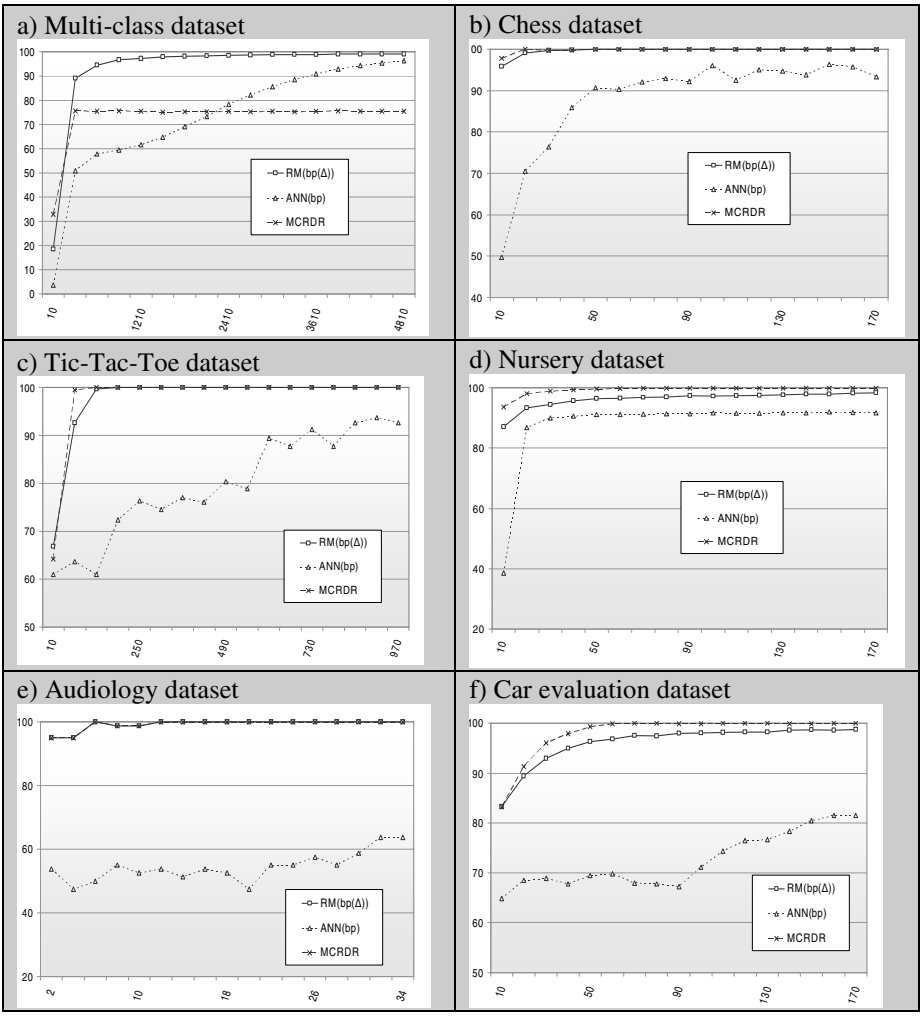

Fig. 7. (a) – (f) shows charts comparing the performance of RM, an ANN and MCRDR using different datasets. The x-axis shows the amount of $1/50^{th}$ data segments that have been seen. The y-axis shows the average accuracy over the last 10 data segments.

The complexity of the multi-class dataset, especially the use of attribute pairs highlights this problem. This caused the simulated expert to be unable to create required rules on some occasions. Therefore, for the purposes of this test, failed rules were not added to the knowledge base. However, these lost rules can now be treated as a form of hidden context. Therefore, RM's ability to significantly outperform the MCRDR's performance shows its ability to capture that hidden information even when it is unavailable to the knowledge base. The performance of RM appears to essentially learn exactly like any standard learning curve but rather than start from scratch it began from where MCRDR had finished learning.

4.5 Prediction Generalisation

Traditionally, MCRDR and other KBSs can usually only be applied to classification problems. Even when used for prediction they usually still use the same basic classification style but each classification gives a predictive value instead. One advantage found with RM is that it can also be applied in a prediction environment. This can be achieved by the network being setup to output just a single value, representing the system's prediction for the task at hand.

The ability of a method to generalise is measured by how well it can correctly rate cases during testing that it did not see during training. The value returned by RM and the ANN is then compared to the simulated expert's correct value. The absolute difference between these two values (error) is then averaged over all the cases in the data segment and logged. The results shown in Fig 8 show they each performed. Each point on the charts is the average error for the test data segment averaged over ten randomised runs of the experiment, for each of the nine tests. To reduce the complexity of the charts shown, error bars have been omitted.

These results show that the RM hybrid system has done exceptionally well both initially as well as after training is complete when generalising. Additionally, it can be observed that the neural network was unable to significantly improve with more training data. This problem is caused by the network having consistently fallen into

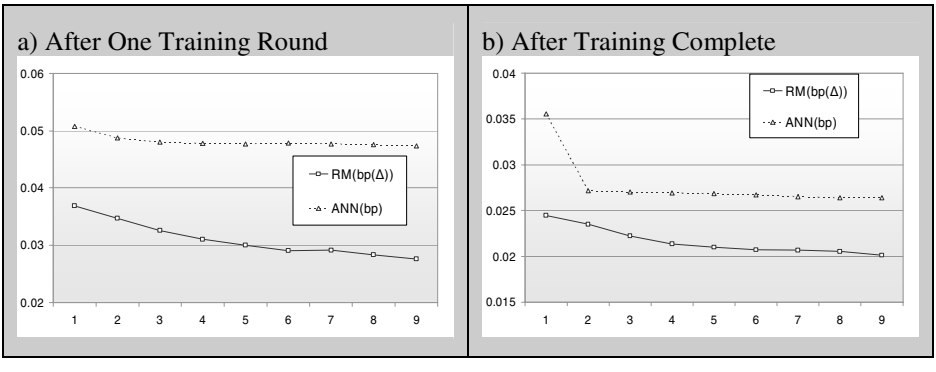

Fig. 8. (a) – (b) Two charts comparing how RM and ANN. Chart a) shows how the methods compare after only one viewing of the training set. Chart b) shows how the methods compare after training was completed. The x-axis shows how many tenths of the dataset were used for training. All results used the last tenth for testing. The y-axis shows the average error.

local minimum, a problem common to neural networks especially in prediction domains. RM is less likely to encounter this learning problem as the knowledge base provides an extra boost, similar to a momentum factor, which propels it over any local minima and closer to the true solution. Therefore, not only does RM introduce KBSs into potential applications in the prediction domain, as well as, allow for greater generalisation similar to an ANN, but it also helps solve the local minima problem.

4.6 Prediction Online

The process of RM being able to predict an accurate value in an online environment could potentially allow the use of RM in a number of environments that have previously been problematic. For instance, KBSs in *information filtering* (IF) have difficulties due to their problems in prediction, while neural networks are far too slow. RM allows for the inclusion of expert knowledge with the associated speed but also provides a means of value prediction. Fig 9 shows a comparison between RM and an ANN in an online environment. Here it can once again be observed that RM has performed outstandingly well from the outset and was able to maintain this performance. This fast initial learning can be vital in many applications as it is what users usually expect.

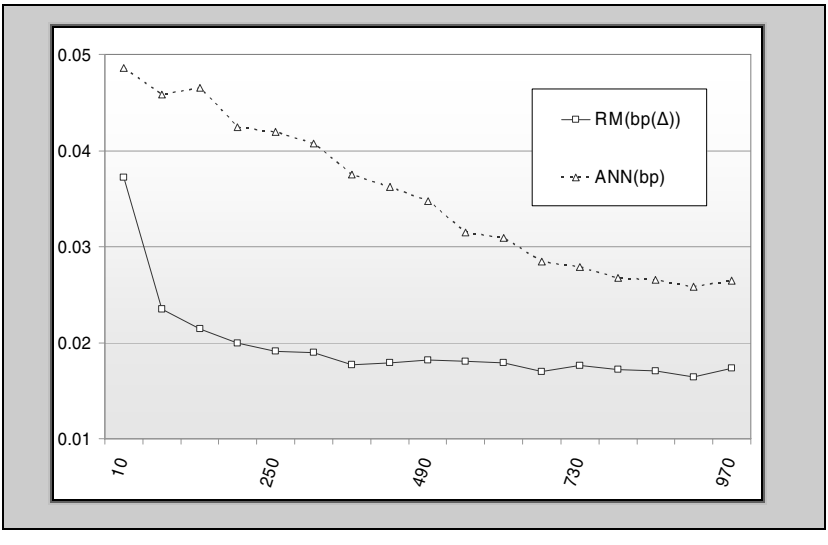

Fig. 9. This chart compares how RM and an ANN, perform in an online environment. The x-axis shows the amount of $1/50^{th}$ data segments that have been seen. The y-axis shows the average error over the last 10 data segments, also averaged over 10 trials.

5 Conclusion

This paper presented an algorithm that detects and models hidden contexts within a symbolic domain. The method developed builds on the already established Multiple Classification Ripple-Down Rules (MCRDR) approach and was referred to as Rated MCRDR (RM). RM retains a symbolic core that acts as a contextually static memory, while using a connection based approach to learn a deeper understanding of the knowledge captured.

A number of results were presented in this paper, which have shown how RM is able to acquire knowledge and learn. RM's ability to perform well can be put down to two features of the system. First, is that the flattening out of the dimensionality of the problem domain by the MCRDR component allows the system to learn a problem that

is mostly linear even if the original problem domain was non-linear. This allows the network component to learn significantly faster. Second, the network gets an additional boost through the *single-step-Δ-initialisation rule*, allowing the network to start closer to the correct solution when knowledge is added.

Acknowledgements

The majority of this paper is based on research carried out while affiliated with the Smart Internet Technology Cooperative Research Centre (SITCRC) Bay 8, Suite 9/G12 Australian Technology Park Eveleigh NSW 1430 and the School of Computing, University of Tasmania, Locked Bag 100, Hobart, Tasmania.

References

1. Newell, A., Simon, H.A.: Computer Science as empirical Inquiry: Symbols and Search. Communications of the ACM 19(3), 113–126 (1976)
2. Menzies, T.: Assessing Responses to Situated Cognition. In: Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Catelonia, Spain (1996)
3. Menzies, T.: Towards Situated Knowledge Acquisition. International Journal of Human-Computer Studies 49, 867–893 (1998)
4. Wille, R.: Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In: Rival, I. (ed.) Ordered Sets: Proceedings of the NATO Advanced Study Institute held at Banff, Canada, pp. 445–472. D. Reidel Publishing, Dordrecht (1981)
5. Kelly, G.A.: The Psychology of Personal Constructs, Norton, New York (1955)
6. Compton, P., Jansen, R.: Knowledge in Context: a strategy for expert system maintenance. In: Second Australian Joint Artificial Intelligence Conference (AI 1988), vol. 1, pp. 292–306 (1988)
7. Brezillon, P.: Context in Artificial Intelligence: II. Key elements of contexts. Computer and Artificial Intelligence 18(5), 425–446 (1999)
8. Dazeley, R., Kang, B.: Epistemological Approach to the Process of Practice. Journal of Mind and Machine, Springer Science+Business Media B.V. 18, 547–567 (2008)
9. Gaines, B.: Knowledge Science and Technology: Operationalizing the Enlightenment. In: Proceedings of the 6th Pacific Knowledge Acquisition Workshop, Sydney, Australia, pp. 97–124 (2000)
10. Menzies, T., Debenham, J.: Expert System Maintenance. In: Kent, A., Williams, J.G. (eds.) Encyclopaedia of Computer Science and Technology, vol. 42, pp. 35–54. Marcell Dekker Inc., New York (2000)
11. Beydoun, G.: Incremental Acquisition of Search Control Heuristics, Ph.D thesis (2000)
12. Compton, P., Edwards, G., Kang, B.: Ripple Down Rules: Possibilities and Limitations. In: 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW 1991), vol. 1, pp. 6.1–6.18. SRDG publications, Canada (1991)
13. Compton, P., Kang, B., Preston, P.: Knowledge Acquisition Without Knowledge Analysis. In: European Knowledge Acquisition Workshop (EKAW), vol. 1, pp. 277–299. Springer, Heidelberg (1993)
14. Preston, P., Edwards, G., Compton, P.: A 1600 Rule Expert System Without Knowledge Engineers. In: Moving Towards Expert Systems Globally in the 21st Century (Proceedings of the Second World Congress on Expert Systems 1993), New York, pp. 220–228 (1993)

15. Preston, P., Edwards, G., Compton, P.: A 2000 Rule Expert System Without a Knowledge Engineer. In: Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, pp. 17.1-17.10 (1994)
16. Kang, B.: Validating Knowledge Acquisition: Multiple Classification Ripple Down Rules, Ph.D thesis (1996)
17. Kang, B.H., Compton, P., Preston, P.: Multiple Classification Ripple Down Rules: Evaluation and Possibilities. In: The 9th Knowledge Acquisition for Knowledge Based Systems Workshop. SRDG Publications, Department of Computer Science, University of Calgary, Banff, Canada (1995)
18. Preston, P., Compton, P., Edwards, G.: An Implementation of Multiple Classification Ripple Down Rules. In: Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Department of Computer Science, University of Calgary. SRDG Publications, Calgary (1996)
19. Compton, P.: Simulating Expertise. In: Proceedings of the 6th Pacific Knowledge Acquisition Workshop, Sydney, Australia, pp. 51–70 (2000)
20. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
21. Compton, P., Preston, P., Kang, B.: The Use of Simulated Experts in Evaluating Knowledge Acquisition. In: 9th AAAI-sponsored Banff Knowledge Acquisition for Knowledge Base System Workshop (KAW 1995), vol. 1, pp. 12.1--12.18. SRDG publications, Canada (1995)
22. Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences (1998)