On a hybrid weightless neural system

T.B. Ludermir*

Centre of Informatics,
Federal University of Pernambuco,
Av. Luiz Freire s/n, 50670-901, Recife, PE, Brazil
E-mail: tbl@cin.ufpe.br
*Corresponding author

M.C.P. de Souto

Federal University of Rio Grande do Norte,
Campus Universitario, Natal-RN, 59072-970, Brazil
E-mail: marcilio@dimap.ufrn.br

W.R. de Oliveira

Dept. of Statistics and Informatics,
Rural Federal University of Pernambuco,
Rua Dom Manoel de Medeiros s/n, 52171-030, Recife, PE, Brazil
E-mail: wrdo@deinfo.ufrpe.br

Abstract: A hybrid system using weightless neural networks (WNNs) and finite state automata is described in this paper. With the use of such a system, rules can be inserted and extracted into/from WNNs. The rule insertion and extraction problems are described with a detailed discussion of the advantages and disadvantages of the rule insertion and extraction algorithms proposed. The process of rule insertion and rule extraction in WNNs is often more natural than in other neural network models.

Keywords: weightless neural networks; WNNs; RAM-based neural networks; rule insertion; rule extraction; grammatical inference; automata theory; Boolean expression; bio-inspired computation; hybrid intelligence systems.

Reference to this paper should be made as follows: Ludermir, T.B., de Souto, M.C.P. and de Oliveira, W.R. (2009) 'On a hybrid weightless neural system', *Int. J. Bio-Inspired Computation*, Vol. 1, Nos. 1/2, pp.93–104.

Biographical notes: Teresa B. Ludermir received her PhD in Artificial Neural Networks in 1990 from Imperial College, University of London, UK. From 1991 to 1992, she was a Lecturer at Kings College London. She joined the Federal University of Pernambuco, Brazil in September 1992, where she is currently a Professor and the Head of the Computational Intelligence Group. She has published over 200 articles in scientific journals and conferences, three books. She is one of the editors-in-chief of the *International Journal of Computation Intelligence and Applications*. Her research interests include weightless NN, hybrid neural systems and applications of NNs.

Marcilio C.P. de Souto received his PhD in Neural Networks in 1999 from Imperial College, University of London, UK. From 2000 to 2003, he was a Visiting Lecturer at Federal University of Pernambuco, Brazil. Currently, he is a Lecturer at the Federal University of Rio Grande do Norte, Brazil. His research interests include weightless neural networks, hybrid intelligent systems and bioinformatics.

Wilson de Oliveira received his PhD in Neural Computing Theory in 2004 from the Centre of Informatics (UFPE), Brazil. Since 2000 he is a Lecturer at the Department of Statistics and Informatics (UFRPE) in Recife, currently on Sabbatical leave at the School of Computer Science in the University of Birmingham, UK. His interests lie on theory of computing, quantum computing, neural computing and theoretical physics. He is employing computing theory methods to quantum gravity and neural quantum computing.

1 Introduction

Human learning consists of a variety of complex processes that use information acquired through interactions with the external environment. In fact, there is no one definitive approach that can explain cognition or solve complex problems. Rather, there are a number of tools and models that can be applied under different circumstances. It is the combination of these different types of information processing methods that has enabled humans to succeed in complex, rapidly changing environments. This high complexity and inherent heterogeneity is still one of the major challenges of current artificial intelligence (AI) systems. Due to the need for different problem solving techniques, the interest in hybrid systems is growing rapidly, as shown by the increasing number of publications found in the literature as for example, the *International Journal of Hybrid Intelligent Systems* (Diederich, 2008; Jacobsson, 2005; Duch et al., 2004; Setiono et al., 2002; Kurfess, 2000; Kasabov, 1996).

Hybrid systems are important when considering the varied nature of application domains in AI. Many complex domains have many different component problems, each of which requiring different types of processing. For example, artificial neural networks (ANNs) have been successfully applied to solve low-level cognitive tasks such as perception, motor control and associative information retrieval. However, considerably less experience has been gathered so far in modelling high-level cognitive tasks using ANNs and on how they represent and reason about the acquired knowledge. In contrast, the ability to provide users with higher level explanations of the reasoning process is an important feature of AI systems. Explanation facilities are required both for user acceptance of the solutions generated by the system and for the purpose of understanding whether the reasoning procedure is sound.

Thus, in order for ANNs to achieve a wider degree of user acceptance and to enhance their overall utility as learning and generalisation tools, it is highly desirable, if not essential, that an explanation capability becomes an integral part of the functionality of a trained ANN. Based on this, a hybrid system in which symbolic rules could be inserted/extracted into/from an ANN model is proposed in this paper. More specifically, the neural model used will be the weightless neural networks (WNNs) (Aleksander and Albrow, 1968; Aleksander and Morton, 1995; Ludermir et al., 1999). Automata theory (Hopcroft et al., 2006) is going to be used in the hybrid system to deal with the insertion and extraction of rules. The basic idea of such a hybrid system, called the hybrid weightless neural networks, is to insert a set of symbolic rules into a WNN. Next, the WNN could be refined by using standard WNN learning algorithms and a set of training examples. The refined network can then function as a highly accurate classifier. A final step for this system is the extraction of refined, comprehensible rules from the trained WNN. Hence, the system can be expressed by the following steps:

1 translate the domain theory into a WNN
2 train the network using a learning algorithm and a set of examples
3 use the trained network to classify future examples
4 extract a refined theory system.

Note that any of the steps above can be either omitted or applied alone. For example, a WNN can be trained through examples and afterwards rules can be extracted from the trained network. Step 1 is referred to as rule insertion, whereas Step 4 is referred to as rule extraction. This hybrid system has the advantages of hand-built classifiers, such as expert systems and neural networks.

The remainder of this paper is divided into four sections. The aim of Section 2 is to present an overview on the neural computing models studied in this paper, i.e., the WNNs. In Section 3, two techniques used for knowledge initialisation built upon ANN techniques are described. The methods described map problem-specific domain theories, represented either as weighted regular grammars or as probabilistic automata, into the classes of MLPN and *p*RAM networks (Ludermir et al., 1999). Section 4 shows two methods for extracting rules from trained WNNs. The first one deals with the extraction of symbolic grammatical rules from recurrent WNNs, whereas the second method produces Boolean expression rules from feed-forward WNNs. Finally, the last section presents some final remarks.

2 Weightless neural networks: basic definitions

The neuron model used in the great majority of work involving neural networks is related to variations of the McCulloch-Pitts neuron (McCulloch and Pitts, 1943), which will be called the 'weighted-sum-and-threshold neuron', or 'weighted neuron' for short. A typical weighted neuron can be described by equations (1) and (2). These specify a linear weighted sum of the inputs, followed by some non-linear transfer function.

$$h = \sum_{j=1}^{p} W_j u_j \tag{1}$$

$$y = \frac{1}{1+\exp^{(-vh)}} \tag{2}$$

where u_j are the inputs; W_j are the synaptic weights of the neuron; h is the 'activation' of the neuron, v controls the shape of the sigmoid function and y is the output. The weights W_j are the parameters that most training algorithms adjust. The transfer function above could also be replaced by other functions (e.g. the threshold function). However, the use of sigmoid functions is motivated by the need to have error functions which are differentiable with respect to the weights so that gradient-descent algorithms, such as the back propagation algorithm (Rumelhart et al., 1986), could be used for learning. The neural models composed of weighted nodes will be called 'weighted neural networks'.

In this paper, however, the neural computing models studied are based on artificial neurons which often have binary inputs and outputs and 'no weight' between nodes. Neuron functions are stored in look-up tables, which can be implemented using commercially available random access memories (RAMs). Learning on these systems generally consists of changing the contents of look-up table entries, which results in highly flexible and fast learning algorithms. These systems and the nodes that they are composed of will be described respectively as WNNs and weight-less nodes (Aleksander and Morton, 1995; Ludermir et al., 1999). They differ from other models, such as the WNNs, whose training is accomplished by means of adjustments of weights. In the literature, the terms 'RAM-based' and 'N-tuple based' have been used to refer to WNNs (Jorgensen, 1997; Ramanan et al., 1995; Rohwer and Morciniec, 1996). Formally,

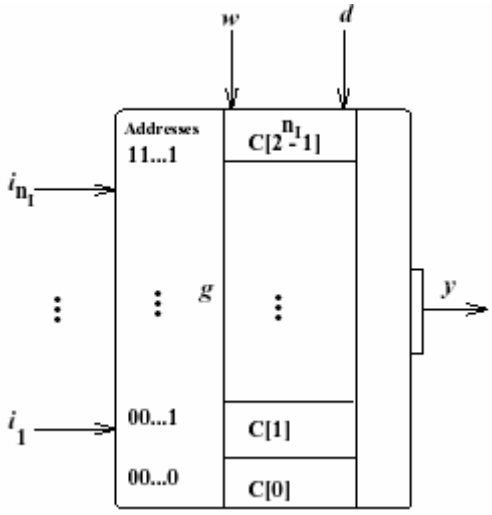
Definition 2.1: A '*q*-RAM neuron' (Figure 1), where $q \in \{1, 2, ...\}$, is an artificial neuron with the following sub-definitions and restrictions:

- $I = \{0,1\}^{n_I}$ is 'the set of inputs for the node', where n_I is 'the number of input terminals'.
- $A = \{0,1\}^{n_I}$ is 'the set of addresses or locations of the node'. For each address $\mathbf{a} \in A$, there is a cell $C[\mathbf{a}]$, which stores the contents or learned information (local memory) in the form of a *q*-bit. A binary signal $\mathbf{i} \in I$ on the input terminals will access only one of these locations, that is, the one for which $\mathbf{a} = \mathbf{i}$. The location $\mathbf{a}$ accessed is called the 'activated location'.
- $y \in S$ is 'the output of the node', where the set S is either [0, 1] or {0, 1};
- $\mathrm{d} \in S$ is 'the teaching terminal', which provides the desired response.
- $w = \{0,1\}$ is 'the operator-mode terminal', which indicates if the neuron is in the learning or recalling phase.
- $g : I \rightarrow S$ is 'the transfer function', which computes y from the *q*-bit stored in the memory location determined by the input terminal, that is, $y = g(C[\mathbf{a} = \mathbf{i}])$.

Definition 2.2: A 'WNN or RAM-based neural network' is a neural network whose neurons are *q*-RAM nodes.

In the next section, the simplest type of weightless node, that is, the RAM node will be presented. For more details about the models introduced in this section or on other types of weightless nodes, such as the general RAM (GRAM) node and the goal-seeking node (GSN), the reader is referred to Aleksander and Morton (1995), Austin (1998) and Ludermir et al. (1999).

Figure 1 A *q*-RAM node



Where:

n_I – connectivity (number of inputs)

i_j – input terminal to receive input j

$C[\mathbf{a}]$ – memory content of address 'a'

y – output

w – operator-mode terminal

d – input terminal to receive the desired response

g – transfer function.

2.1 The RAM model

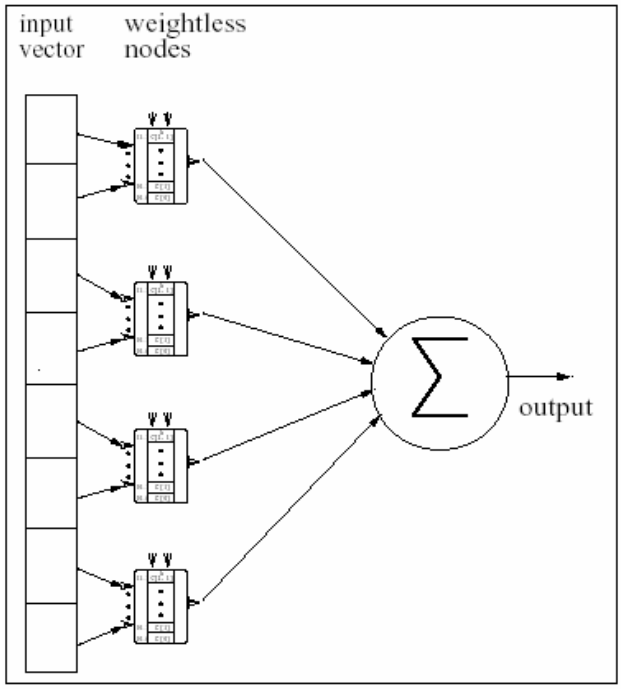
Definition 2.3: A 'RAM neural network' is a WNN in which the neurons are 1-RAM nodes. The 1-RAM node is called the RAM node. The bit, $C[\mathbf{a}]$, stored at the activated memory $\mathbf{a} = \mathbf{i}$ represents the output of a RAM node, that is, $y = C[\mathbf{a}]$. Thus, g is the identity function. In other words, the Boolean function performed by the neuron is determined exactly by the contents of the RAM.

Learning in a RAM node takes place simply by writing into the corresponding look-up table entries. This learning process is much simpler than the adjustment of weights. The RAM node, as defined above, can compute all binary functions of its input while the weighted-sum-and-threshold nodes can only compute linearly separable functions. There is no generalisation in the RAM node itself. Nevertheless, there is generalisation in networks composed of RAM nodes (Aleksander and Morton, 1995). Generalisation can be, for example, introduced by considering the hamming distance from training patterns or masks.

In a typical RAM neural network, that is, a feed-forward RAM neural network with a single-layer of nodes, RAM nodes are taught to respond with a 1 for those patterns in the training set and only for those patterns. An unseen pattern is classified in the same class of the training set if all RAM nodes output 1. This simple architecture divides the set of all possible patterns into those that are in the generalisation set and those that are not. If more than two categories are

required, many RAM neural networks are used together, each network trained to respond to one class of pattern. These networks are modified so that instead of having a logic gate to combine the RAM nodes' outputs, the decision is left to the 'maximum response detector'. These modified RAM neural networks, which are used both in SLAMs and WISARD (Aleksander et al., 1984), are called 'discriminators' (Figure 2).

Figure 2 A discriminator



A simple learning algorithm for RAM networks is summarised below:

RAM learning algorithm

1 Present an input pattern to the input terminals.

2 Select the RAM nodes that should learn (α nodes are going to learn, where α is a parameter – learning rate – set before training) and present the desired output to the vector *d*.

3 Set to 1 the operate-mode terminals, w, of the nodes that are going to learn.

4 Repeat by going back to Step 1 for all the N training patterns.

5 The algorithm halts when the error of the solution is acceptable (this parameter is also set before training). Otherwise, repeat the whole procedure by going back to Step 1.

An important aspect about RAM networks is that although there is no ambiguous information concerning RAM locations which contain 1, the same does not hold for those that are left in the 0 state. An unknown vector accessing an entry that has 0 could mean either

1 that such vector is a counter-example of the corresponding class

2 that no information was provided during training for that class feature corresponding to the entry accessed.

It is not easy to define which one is true. Another discriminator holding a value 1 at the same location indicates a counter-example. In the second case, the vector could be an example of the class, but since no information was given, the RAM will output 0 instead of 1. To overcome this problem, Kan and Aleksander (1987) developed the word-addressable logic node called probabilistic logic node (PLN), which will be described in the next section.

2.2 The PLN model

A PLN differs from a RAM node in that a 2-bit number (rather than a single bit) is now stored at the addressed memory location. The content of this location is turned into the probability of firing (i.e., generating 1) at the overall output of the node. In other words, a PLN consists of a RAM node augmented with a probabilistic output generator (Kan and Aleksander, 1987). Thus, like in a RAM node, the n_I binary inputs to a PLN form an address to one of the 2^{nI} addressable locations $a \in A$. Simple RAM nodes then output the stored value directly. In contrast, in a PLN, the content at this address is passed through a transfer function which converts it into a binary node output. Such a content could be either 0's, 1's, or *u*'s. The undefined state *u* implies on the node flipping its output between 0 and 1 with equal probability. The use of a third logic value, *u* (undefined), makes possible the use of an 'unknown' state in the operation of WNNs architectures. This value is stored in all the memory contents before the learning phase, indicating the ignorance of the network before it was trained. Formally,

Definition 2.4: A 'PLN neural network' is a WNN in which the neurons are 2-RAM nodes (PLNs). The output of the PLN is given by:

$$y = \begin{cases} 0 & \text{if } C[\mathbf{a}] = 0 \\ 1 & \text{if } C[\mathbf{a}] = 1 \\ \text{random}(0,\,1) & \text{if } C[\mathbf{a}] = u \end{cases} \tag{3}$$

where $C[\mathbf{a}]$ is the content in the address position associated with the input pattern **i** (i.e. **a** = **i**) and random (0,1) is a random function that generates zeros and ones with the same probability.

Training PLN neural networks becomes a process of replacing *u*'s with 0's and 1's so that the network consistently produces the correct output pattern in response to training pattern inputs. At the beginning of training, all stored values in all nodes are set to *u* and thus the net's behaviour is completely unbiased. A generic gradient descendent learning algorithm, proposed by Myers and Aleksander (1989), uses several presentations of the training

set to teach PLN neural networks by using a reward and a punish phase.

A simple learning algorithm for multi-layer PLN networks is summarised as follows:

PLN learning algorithm

1 Set the contents of all the memory locations to u.

2 Present an input pattern to the input terminals together with the desired response to the output terminals.

3 The contents of the memory locations are propagated forward through the layers of the network until they reach the output node.

4 The response of the network is compared with the desired output.

   a if they are similar, then the contents of the addressed memories will assume their current values (i.e. the addressed memories will be rewarded), or

   b if they are different, then the network is allowed to run again until:

   - The output matches the desired response, then reward.
   - The output mismatches the desired response β times (β is a parameter which should be set before training). In this case, the addressed memories are punished by reverting their contents back to the u state.

5 Repeat by going back to Step 2 for all the N training patterns.

6 The algorithm halts when consistent success (the correct output produced N times consistently) indicates that all patterns have been learned.

The use of three-logic values and PLN neural networks represented a breakthrough in the WNN research. Other WNNs models were soon created incorporating these ideas, such as the MPLN and *p*RAM node. These nodes, which are similar in many ways, were simultaneously and independently developed in Myers and Aleksander (1988, 1989) and Gorse and Taylor (1988, 1991) respectively.

2.3 The MPLN model

The development of the PLN led to the definition of the m-state PLN or MPLN (Myers and Aleksander, 1988, 1989). The main difference between the MPLN and the PLN is that the first allows a wider, but still discrete, range of probabilities to be stored at each memory content unit. One result of extending the PLN to MPLN is that the node locations may now store output probabilities which are more finely graded than in the PLN.

An MPLN, for instance, could output 1 with 15% probability under a certain input. Based on this, learning can allow incremental changes in stored values. This way, one reset does not erase much information – erroneous information is discarded only after a series of errors. Similarly, new information is only acquired after a series of experiences indicates its validity. The reinforcement learning procedure for PLNs was extended to take this into account and was applied to model delay learning in invertebrates (Myers, 1992).

Definition 2.5: An 'MPLN neural networks' is a WNN where the node are q-RAM nodes, for $q > 2$, where the activation function g, $g: I \to \{0,1\}$, is a probabilistic function.

3 Rule insertion in WNN

There are basically two approaches to achieving problem-specific expertise in a computer: hand-built classifiers (e.g. expert systems) and empirical learning (e.g. ANNs). Hand-built classifiers correspond to teaching by giving a person a domain theory without an extensive set of examples. Conversely, empirical learning corresponds to giving a person lots of examples without any explanation on why the examples are members of a particular class.

Hybrid learning methods use theoretical knowledge of a domain and a set of classified examples to develop a method for accurately classifying examples not seen during training. The challenge of hybrid learning systems is to use the information provided by one source of information to offset information missing from the other source. By so doing, a hybrid learning system should learn more effectively than systems that use only one of the information sources.

In this section, two techniques used for knowledge initialisation built upon ANN techniques are presented. The methods described map problem-specific domain theories, represented either as weighted regular grammars or as probabilistic automata, into neural networks. That is, an existing set of rules is converted into a neural network, which then can be trained (in the context of this research with WNN training algorithms) with sample data. The goal is the utilisation of existing knowledge and the adaptation of sets of rule to actual data. The availability of prior information can also be used to improve the training process of neural networks. Since the starting configuration of the network is not randomly chosen, but often reflects important aspects of the learning task, the time to train the network may be substantially reduced, or the resulting network may offer better generalisation capabilities.

3.1 From weighted regular grammars to multi-layer MLPN networks

In Ludermir (1992), an algorithm is presented that allows the insertion of rules, given in terms of weighted regular grammars, into MLPN networks.

Theorem 3.1: (Ludermir, 1992) Let G_w be a weighted regular grammar, then there exists an MPLN neural network that recognises $L(G_w)$ with some cut-point λ.

The way in which the grammar is transformed into the neural network is such that all its properties are preserved. Thus, the grammar can be inferred back from the WNN generated. The networks are constructed using four basic kinds of nodes: 'p-and', 'p-or', 'complement' and 'delay' nodes. The algorithm is based on the complexity of the production rules. More specifically, the procedure is divided into three cases. The first case deals with production rules of the form $S_1 \rightarrow w(p)$ where w is a word in the language, S_1 is a non-terminal symbol of the grammar and p is the probability associated with this production rule. The second case deals with production rules of the form $S_i \rightarrow wS_j(p)$ and the last case deals with production rules of the form $S_i \rightarrow w_1S_j(p_1) \mid S_i \rightarrow w_2S_k(p_2)$ where $\mid$ denotes the possibility of S_i being replaced with $w_1S_j(p_1)$ or $w_2S_k(p_2)$. Each of these cases is divided into sub-cases. Furthermore, for every sub-case, the circuit that implements such a case is designed using the four types of nodes previously mentioned. As regular grammars and RAM nodes are special cases of weighted regular grammars and MPLNs, respectively, the algorithms in Ludermir (1992) work for them as well.

A drawback with the previous procedure is the fact that the MPLN neural networks generated are not well suited to learning. They possess irregular topologies in that:

1 they could have several layers (blocks) with different number of nodes

2 each node within the same layer could have a different fan-in

3 each layer could have feedback connections coming from nodes within the same layer as well as from nodes in any other layer. Nevertheless, as will be shown in the next section, de Souto et al. (1998) proposed an algorithm to transform rules given as probabilistic automata (weighted regular language recognisers) into a given class of single-layer sequential WNNs.

3.2 From probabilistic automata to single-layer recurrent WNN

An important representative of the research on temporal pattern processing in WNNs is the class sequential weightless neural networks (SSWNNs) (de Souto et al., 2005). In de Souto et al. (1998, 2000, 2005), the computability of a class of SSWNNs, called general single-layer sequential weightless neural networks (GSSWNNs), was studied. These networks were assumed to be implemented either with *p*RAM nodes (Gorse and Taylor, 1988) or MPLNs. Indeed, one of the proofs presented provides an algorithm to map any probabilistic automaton into a GSSWNN:

Theorem 3.2: (de Souto et al., 1998) Let $\mathbf{A}p = (\sum, Q, H, q_I, F)$ be a probabilistic automaton with cut-point λ. Then there exists a GSSWNN that implements $\mathbf{A}_P$.

With respect to proof of the previous theorem, a probabilistic automaton $\mathbf{A}_P$ is reduced to a GSSWNN as follows:

1 A mapping $\sum \rightarrow \{0,1\}^{n_U}$, where $n_U = \lceil \log_2 |\sum| \rceil$, is defined. Such a mapping transforms each possible input symbol $\sigma_k \in \Sigma$ into a different input vector $\mathbf{u}_k$ for the network.

2 A mapping $Q \rightarrow \{0,1\}^{n_X}$, where $n_X = \lceil \log_2 |Q| \rceil$, is defined. Thus, each state $q_i \in Q$ is assigned to a vector $\mathbf{x}_i$ for the network, such that if $q_i \neq q_j$, then $\mathbf{x}_i \neq \mathbf{x}_j$

3 The network initial state $\mathbf{x}_0$ is defined as the vector standing for $q_I \in Q$, that is, the automaton initial state.

4 The set *R* of network accepting states will consist of vectors $\mathbf{x}_i$ representing the states $q_I \in F$, that is, the automaton set of final states.

In summary, the algorithm above generates a GSSWNN which has its number of nodes logarithmic in the number of automaton states and each node has a number of memory locations linear in the number of such states. Once the network is generated, it can be used with the recognition algorithm presented in de Souto et al. (1998), which makes such a network behave like a probabilistic automaton.

3.3 Discussion

The two methods presented in the previous sections not only allow the construction of any weighted regular language/probabilistic automaton into MPLN/*p*RAM networks, but also increases the class of functions that can be computed by such networks. For instance, these networks can now deal with some context-free languages (Ludermir, 1992; de Souto et al., 1998).

Also, these methods could be used to create the structure of the network and its initial probabilities from a partially known weighted regular grammar/probabilistic automaton. Then, such a network can be submitted to some kind of learning algorithm. For instance, its training could involve:

- only changes in the probabilities stored in the nodes
- changes in any memory position of the node
- changes in the value of the threshold (cut-point).

The languages recognised by a neural network when only the threshold is changed are related to each other. The larger the value of the threshold, the fewer the elements of the language that will be recognised by the network. That is, $L_1 \supset L_2 \supset ... \supset L_n$ when $\lambda_1 < \lambda_2 < ... < \lambda_n$. If the threshold is small, there will be more restrictions in the path followed in the network: every time a new symbol, X_i, is submitted to the network, the value of the probability of the pattern $p(\tilde{X})$ will decrease or will not change, but it will never increase.

When changes occur only in the probabilities stored in the nodes, the training algorithm can be very simple. Given a pattern $\tilde{X}$, if $\tilde{X} \in L$, then reward the network (increase the probabilities of the transitions that the network has undergone when fed with $\tilde{X}$), otherwise punish the network (decrease the probabilities of the transitions that the network has undergone when fed with $\tilde{X}$).

As these WNNs, generated by the previous procedures, are similar to probabilistic automata (Ludermir, 1992; de Souto et al., 1998), one should take into account their behaviour in a noisy environment. For instance, a slight perturbation in the probability of the state transition of a probabilistic automaton could lead the automaton to recognise a different language (Rabin, 1963). In fact, there are some sufficient conditions for stability in probabilistic automata, as well as there are cases in which stability is not possible.

The general problem of stability is still unsolved, which means that if the changes generated by the training algorithm in the state transition of the network are small there is no guarantee that the training will change the language recognised by the network. Thus, if changes are allowed in the contents of the nodes, a completely different structure might arise. However, practical shortcomings of these results need to be investigated further.

4 The rule extraction problem

In this section, the rule extraction problem is described with a detailed discussion of the advantages and disadvantages of the rule extraction algorithms proposed. The approaches of rule extraction are described followed by the two methods proposed by the authors. At the end of the section, a comparative study of the algorithms is presented.

Given a trained ANN and the examples used in the training process, produce a concise and precious symbolic description of the ANN. This is the formulation of the rule extraction problem originally made in Craven and Shavlik (1994).

In essence, the task of extracting rules from a trained ANN is one of interpreting in a comprehensive fashion the collective effect of: the network architecture, an activation function associated with each unit of the ANN and a set of numerical values stored/associated in/with each unit.

A correlated problem to that of rule extraction from a trained ANN is that of using the ANN for the refinement of existing rules within symbolic knowledge bases. The idea is to program the network with a knowledge base (a prior knowledge) and train the network to produce a set of better rules (a posteriori knowledge).

The main benefits of ANN rule extraction, as mentioned in Andrews et al. (1995) are:

- Provision of a user explanation capability – while provision of an explanation capability is a significant innovation in the ongoing development of ANNs, of equal importance is the quality of the explanation given. For example, experience has shown that an explanation based on a rule traced from a poorly organised rule-base with perhaps hundreds of premises per rule is not regarded as being transparent. ANNs has no declarative knowledge structures and hence, is limited in providing an explanation component.
- Extension of ANN systems to safety-critical problem domains – the internal states of the system to be both accessible and able to be interpreted unambiguously. Satisfaction of such requirements would make a significant contribution to the task of identifying and, if possible, excluding those ANN-based solutions that have the potential to give erroneous results without any accompanying indication as to when and why a result is sub-optimal.
- Software verification and debugging of ANN components in software systems – if ANNs are to be integrated within larger software systems that need to be verified, then this requirement clearly must be met by the ANN as well.
- Improving the generalisation of ANN solutions – by being able to express the knowledge embedded within the trained ANN as a set of symbolic rules, the rule extraction process may provide an experienced system user with the capability to anticipate or predict a set of circumstances under which generalisation failure can occur.
- Data exploration and the induction of scientific theories – ANNs has proven to be extremely powerful tool for discovering previously unknown dependencies and relationships in data sets. As Craven and Shavlik (1994) observe, a (learning) system may discover salient features in the input date whose importance was not previously recognised. But even if a trained ANN has learned interesting and possibly non-linear relationships, these relationships are encoded incomprehensibly as weight vectors within the trained ANN and hence cannot easily serve the generation of scientific theories.
- Knowledge acquisition for symbolic AI systems – the most difficult, time consuming and expensive task in building an expert system is constructing and debugging its knowledge base. The notion of using trained ANNs to assist in the knowledge acquisition task has existed for some time. An extension of these ideas is to use trained ANNs as a vehicle for synthesising the knowledge that is crucial for the success of knowledge-based systems. Alternatively, domain knowledge that is acquired by a knowledge engineering process may be used to constrain the size of the space searched during the learning phase and hence contribute to improving the learning performance.

4.1 Approaches to rule extraction

One approach to testing rules for a multi-layer network is to treat the network as a collection of perceptrons and to extract rules for each hidden and output unit separately (Maclin and Shavlik, 1993). In this approach, the rules for each unit are expressed in terms of the units that feed into it. An advantage of this approach is that it produces 'intermediate terms' which may result in simpler descriptions. A disadvantage of this method, however, is that it requires that the hidden units of the networks be approximated by threshold units and, thus, the extracted rules may not provide an accurate representation of the network. This approach is referred to as 'decompositional' or 'local'. The distinguishing characteristic of the decompositional approach is that the focus is on extracting rules at the level of individual (hidden and output) units within the trained ANN. Examples of algorithms in this class are SUBSET (Towell and Shavlik, 1993), M of N (Towell and Shavlik, 1993) and Rulex (Andrews et al., 1995). Other works can be found in Andrews et al. (1995), Towell and Shavlik (1993), Craven and Shavlik (1994), Thrun (1991) and Jacobsson (2005).

The other approach is called 'global method'. This approach describes the behaviour of hidden and output units in function of the input units alone. The extracted knowledge can only provide a crude approximation of the network because these methods treat the trained ANN as a black box. The main algorithms in this approach are the validity interval analysis (VIA) (Thrun, 1991) and the Trepan (Craven and Shavlik, 1996).

Most rule extraction methods for ANN suffer from both a lack of generality and lack of stability. Some methods are limited in their applicability because they impose restrictions on the network architecture (Thrun, 1991; Craven and Shavlik, 1994) or because they require hidden units to use sigmoidal transfer functions (Maclin and Shavlik, 1993).

4.2 Extracting rules from WNNs

In the next sections, two methods for extracting rules from trained WNNs are presented. The first one deals with the extraction of symbolic grammatical rules from recurrent WNNs, whereas the second method produces Boolean expression rules from feed-forward WNNs.

4.2.1 Extracting symbolic grammatical rules from WNNs

Next, a generic description for extracting rules from trained recurrent WNNs is given. It is shown that for any MPLN network, it is possible to generate a set of symbolic rules that describes the set of patterns that the network recognises. This result is expressed as the following theorem:

Theorem 4.1: If a set of patterns L is recognised by an MPLN neural network, then this set can be generated by a weighted regular grammar G_w.

Proof: Let N be the MPLN network that recognises only the set of patterns L and let $G_W = (V_N, V_T, P_W, q_0)$ be the grammar that generates only and all patterns of L. Suppose there is an initial state q_0 of N from which the feed of all patterns of L will start and suppose further that q_0 is not a final state. Then, there is a production rule $S_i \rightarrow aS_j(p)$ whenever the feed of the symbol a to the network in state S_i causes the network to enter state S_j with probability p and also $S_i \rightarrow a(p)$ whenever the feeding of the symbol a to the network in state S_i takes the network to a final state with probability p.

In the same way, there is a set of production rules such that $S_i \rightarrow wS_j(p)$, whenever the feed of the patterns w to the network in state S_i causes the network to enter state S_j with probability p. If w is accepted by N then, S_i is q_0 and S_j is a final state. Hence $L(N) = L(G_W)$.

Now let q_0 be in the set of final states, then the empty word, $\in$, is in L. Note that the grammar defined above is $L = L - \in$. G_W can be modified by adding a new start symbol S with productions $S \rightarrow q_0(p_1)$ and $S \rightarrow \in (p_2)$.

4.2.2 Extracting rules from feed-forward WNNs

An algorithm to extract rules from feed-forward WNNs, in terms of Boolean expressions, is given below:

Rule extraction algorithm

1 For each RAM node in the network do:
   - a find all 1's positions in the memory locations
   - b generate the expressions for all memory contents with 1's.
   - c build an *OR* function with all expressions generated in Step (b) above.

2 Apply the method of Veitch-Karnaugh for expression simplification to the expression generated by 1.

This algorithm works for any kind of feed-forward WNN. As a result of the application of the algorithm to a trained WNN, a set of simple rules is obtained. It is also possible to insert this set of rules back into a WNN.

In what follows, two examples of rule extraction are given. The first example uses a standard example of WNN found in the literature (Beale and Jackson, 1991) while the second is a classical real world classification problem, the Winsconsin breast cancer database (Merz and Murphy, 1996).

The first example is the network of Figure 3. The training set for this network is shown in Figure 4. The RAM nodes are taught to respond with 1 for those patterns in the training set and only those patterns. Thus, any pattern that makes all three nodes to output 1s would be classified in the same way as the training set – in this case, the generalisation set (the extra patterns recognised by the network) is shown in Figure 5.

Figure 3 RAM network with three nodes

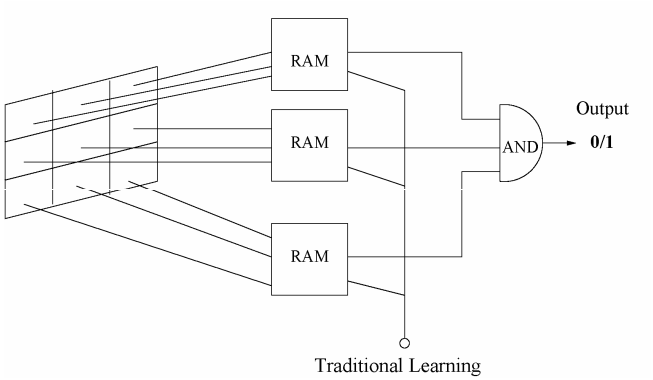


Figure 4 Training set

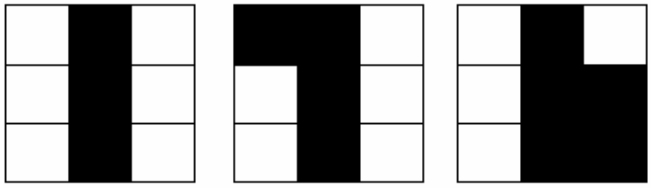
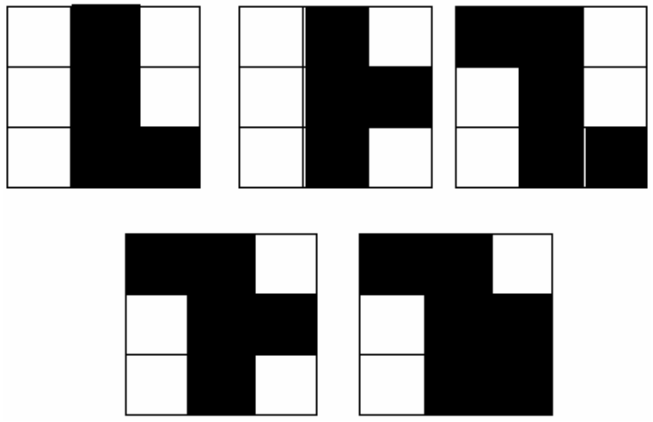


Figure 5 The generalisation set: extra patterns that the network recognises



Consider the input pattern as a matrix where the elements of the first line are a_{11}, a_{12} and a_{13}; the elements of the second line are a_{21}, a_{22} and a_{23}; and the elements of the third line are a_{31}, a_{32} and a_{33}. The rule generated by Step 1 of the algorithm is:

(((not a_{11}) and a_{12} and (not a_{13})) or (a_{11} and a_{12} and (not a_{13}))) **and**

(((not a_{21} and a_{22} and (not a_{23})) or ((not a_{21}) and a_{22} and a_{23})) **and**

(((not a_{31}) and a_{32} and (not a_{33}) or ((not a_{31}) and a_{32} and a_{33}))

The (minimal) rule generated by Step 2 of the algorithm is:

(a_{12} and (not a_{13})) **and**

((not a_{21}) and a_{22}) **and**

((not a_{31}) and a_{32})

The **and**s in bold face in the rules above is because of the **and** gate, which connects the RAM nodes in the network of Figure 3. In general, the minimal rule has less than half the size of the rule generated by Step 1.

In order to assess the performance of the method proposed in this section in a real world pattern classification problem, the breast cancer database from the UCI repository will be used (Merz and Murphy, 1996). This database, consisting of 699 cases – 458 presenting benign cancer and 241 of the malign type, represent here the knowledge of the specialist of domain and will be used to test and train the RAM network. All these cases reflect the results from clinical diagnosis obtained from the Hospital of Wisconsin University (Madison-USA), between January 1989 and November 1991. Each register of this database corresponds to a case of breast cancer and has 11 attributes. The first attribute corresponds to the identification number of the case, the next nine attributes refer to the symptoms (attribute of microscope analyses made in the tumours removed from the patients which is denoted by x_1 to x_9 in this paper), necessary to reach a classification and the last attribute is the class or type of cancer associated to the symptoms mentioned in the nine preceding attributes. For 16 registers, one attribute is missing, so they have been removed from the dataset used in this work. The remaining 683 have been subdivided into training (372 patterns) and test set (311 patterns). For the nine attributes of the symptoms, the values are integer number in the range 1 to 10. The integer numbers are converted in binary numbers to be used as the input of the RAM networks. The input pattern is represented as a matrix nine by four, the lines are the nine input features of the problem. As in the previous example, Figure 3, a feed-forward RAM network with a single-layer of adjustable nodes is used where each line of the input matrix is the input of one RAM node. The outputs of the nine RAM nodes are combined by an and gate. The rule below, extracted from the trained network with this problem, is expressed with integer numbers (instead of binary numbers) to better expressiveness, so each line i of the matrix is denoted by x_i.

The rule generated by the algorithm is:

$x_1 <= 8$ and $x_4 <= 7$ and $x_5 <= 5$ and $x_6 <= 2$ and $x_7 <= 9$ and $x_8 <= 8$ and $x_9 <= 3$ or

$x_1 <= 8$ and $x_3 <= 2$ and $x_4 <= 7$ and $x_5 <= 5$ and $x_6 <= 3$ and $x_8 <= 8$ or

$x_3 = 1$ and $x_4 <= 9$ and $x_8 <= 8$ or

$x_1 <= 8$ and $x_4 <= 7$ and $x_5 <= 5$ and $x_6 <= 4$ and $x_7 <= 2$ and $x_8 <= 8$ and $x_9 <= 3$

The direct application of this rule in the test set gives an error of 14 patterns misclassified, a generalisation error of about 2%.

In a previous work, a rule extraction mechanism were used to extract IF/THEN rules, from an MLP trained network with back propagation using the breast cancer

database. The generalisation error obtained was 17, 10% with a more complicated set of rules (Campos et al., 2004).

4.3 Comparative study

A classification scheme for rule extraction techniques is proposed in Andrews et al. (1995). The method of classification is in terms of:

1 the expressive power of the extracted rules:
   a the output as a set of rules expressed using conventional logic
   b rules using the concept of membership functions (fuzzy logic)
   c rules represented in first-order logic form, i.e., rules with quantifiers and variables
2 the translucency of the view taken within the rule extraction technique of the underlying ANN unit – the relationship between the extracted rules and the internal architecture:
   a decompositional – at the level of individual units
   b pedagogical – treats the trained ANN as a 'black box'
   c eclectic – which combines elements of the two basic categories above
3 the extent to which the underlying ANN incorporates specialised training regimes provides some measure of the portability of the rule extraction technique across various ANN architectures
4 the quality of the extracted rules:
   a accuracy – if it can correctly classify previously unseen examples
   b fidelity – if it can mimic the behaviour of the ANN from which it was extracted by capturing all the information embodied in the ANN
   c consistency – if, under differing training sessions, the ANN generates rule sets which produce the same classifications as unseen examples
   d comprehensibility – is measured by the size of the rule set (in terms of the number of rules) and the number of antecedents per rule
5 the algorithmic complexity of the rule extraction technique – the algorithms to be as efficient as possible.

The first method described in this paper is as follows:

1 has high expressive power
2 uses a decompositional technique, that is, the rule extraction is made at the level of individuals nodes
3 is valid for every type of WNN
4 extract rules of high quality, which means that:
   a accuracy – it can correctly classify previously unseen examples
   b fidelity – it mimics the behaviour of the ANN from which it was extracted by capturing all the information embodied in the ANN
   c consistency – the method does not have this property because the training algorithms used do not have it
   d comprehensibility – the size of rule is not large
5 the complexity of the algorithm is low.

The second method described in this paper is as follows:

1 has the output expressed as a set of rules using conventional symbolic logic
2 treat the trained ANN as a black-box
3 is valid for all every type of WNN
4 extract rules of high quality, which means:
   a accuracy – it can correctly classify previously unseen examples
   b fidelity – it mimics the behaviour of the ANN from which it was extracted by capturing all the information embodied in the ANN
   c consistency – the method does not have this property because the training algorithms used do not have it
   d comprehensibility – the size of rule is very small
5 the complexity of the algorithm is low.

With both methods, rules hold regardless of the values that unmentioned variables take on. The rules are maximally general in the sense that if any of the conditions are removed, then the rules are no longer valid.

The overall process must preserve genuine knowledge/rules and must correct prior incorrect information rules.

Clearly, an assessment of the quality of the rules produced by a given rule extraction technique is potentially of significant value to a prospective user.

5 Conclusions

This paper describes a hybrid system for WNNs. This is a hybrid system that uses both domain knowledge and labelled examples. For rule insertion, the system uses weighted regular grammars and probabilistic automata. After the rule insertion phase, the systems can use a WNN learning algorithm to improve performance. These two steps used together could lead to a classifier with better generalisation and efficiency (in terms of the number of training examples required) than systems that make use of only training examples. After the first step, the network topology and node contents are set in such a way that the network initially reproduces the set of inserted rules. The

network is then trained using a set of labelled examples and standard WNN algorithms. A hybrid system like this has the advantages of both hand-built classifiers and neural networks.

Acknowledgements

The authors would like to thank CNPq, FACEPE and CAPES (Brazilian research agencies) for their financial support.

References

Aleksander, I. and Albrow, R.C. (1968) 'Adaptive logic circuits', *Computer Journal*, Vol. 11, p.65.

Aleksander, I. and Morton, H. (1995) *An Introduction to Neural Computing*, 2nd ed., Intl Thomson Computer Pr, London, UK.

Aleksander, I., Thomas, W.V. and Bowden, P.A. (1984) 'WISARD: a radical step forward in image recognition', *Sensor Review*, Vol. 4, No. 3, pp.120–124.

Andrews, R., Diederich, J. and Tickle, A. (1995) 'A survey and critique of techniques for extracting rules from trained artificial neural networks', *Knowledge Based Systems*, Vol. 8, pp.373–389.

Austin, J. (1998) 'RAM-based neural networks: a short history', in Austin, J. (Ed.): *RAM-based Neural Networks*, World Scientific, UK.

Beale, R. and Jackson, T. (1991) *Neural Computing: An Introduction*, Adam Hilger, Bristol.

Campos, P.G., Oliveira, E.M.J. and Ludermir, T.B. (2004) 'Mlp networks for classification and prediction with rule extraction mechanism', in *IJCNN*, pp.1387–1392.

Craven, M. and Shavlik, J. (1994) 'Using sampling and queries to extract rules from trained neural networks', in *Proceedings of the Eleventh International Conference of Machine Learning*, San Francisco, pp.152–169.

Craven, M.W. and Shavlik, J.W. (1996) 'Extracting tree-structured representations of trained networks', *Advances in Neural Information Processing Systems*, Vol. 8, p.2430.

de Souto, M.C.P., Ludermir, T.B. and Campos, M.A. (2000) 'Encoding of probabilistic automata in RAM-based networks', in *Proc. of the International Joint Conference on Neural Networks*, IEEE, pp.439–444.

de Souto, M.C.P., Ludermir, T.B. and de Oliveira, W.R. (1998) 'Synthesis of probabilistic automata in pRAM neural networks', in Niklasson, L., Boden, M. and Ziemke, T. (Eds.): *Proc. of the International Conference on Artificial Neural Networks (ICANN98)*, Vol. 2 of *Perspectives in Neural Computing*, pp.603–608, Springer-Verlag, Skovde, Sweden.

de Souto, M.C.P., Ludermir, T.B. and de Oliveira, W.R. (2005) 'Equivalence between ram-based neural networks and probabilistic automata', *IEEE Trans. Neural Networks*, pp.996–999.

Diederich, J. (2008) *Rule Extraction from Support Vector Machines*, Springer, USA.

Duch, W., Setiono, R. and Zurada, J. (2004) 'Computational intelligence methods for rule-based data understanding', *Proceedings of the IEEE*, Vol. 92, pp.771–805.

Gorse, D. and Taylor, J.G. (1988) 'On the equivalence and properties of noisy neural networks and probabilistic RAM nets', *Physics Letters A*, Vol. 131, No. 6, pp.326–332.

Gorse, D. and Taylor, J.G. (1991) 'A continuous input RAM-based stochastic neural model', *Neural Networks*, Vol. 4, pp.657–665.

Hopcroft, J., Motwani, R. and Ullman, J. (2006) *Introduction to Automata Theory, Languages, and Computation*, 3rd ed., Addison Wesley, USA.

Jacobsson, H. (2005) 'Rule extraction from recurrent neural networks: a taxonomy and review', *Neural Computation*, Vol. 17, pp.1223–1263.

Jorgensen, T.M. (1997) 'Classification of handwritten digits using a RAM neural net architecture', *International Journal of Neural Systems*, Vol. 8, No. 1, pp.17–25.

Kan, W.K. and Aleksander, I. (1987) 'A probabilistic logic neuron network for associative learning', in *Proc. of the IEEE International Conference on Neural Networks*, San Diego, California, Vol. 2, pp.541–548.

Kasabov, N.K. (1996) *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, MIT Press, USA.

Kurfess, F.J. (2000) 'Neural networks and structured knowledge: rule extraction and applications', *Applied Intelligence*, Vol. 12, pp.7–13.

Ludermir, T.B. (1992) 'Computability of logical neural networks', *Journal of Intelligent Systems*, Vol. 2, pp.261–289.

Ludermir, T.B., de Carvalho, A., Braga, A.P. and de Souto, M.C.P. (1999) 'Weightless neural models: a review of current and past works', *Neural Computing Surveys*, Vol. 2, pp.41–61.

Maclin, R. and Shavlik, J.W. (1993) 'Using knowledge-based neural networks to improve algorithms: refining the Chou-Fasman algorithm for protein folding', *Machine Learning*, Vol. 11, pp.195–215.

McCulloch, W.S. and Pitts, W. (1943) 'A logical calculus of the ideas immanent in nervous activity', *Bulletin of Mathematical Biophysics*, Vol. 5, pp.115–137.

Merz, C.J. and Murphy, P.M. (1996) 'UCI repository of machine learning databases', technical report, University of California, Irvine, CA.

Myers, C. (1992) *Delay Learning in Artificial Neural Networks*', Chapman & Hall.

Myers, C. and Aleksander, I. (1988) 'Learning algorithms for probabilistic logic nodes', in *Abstracts of I Annual INNS Meeting*, Boston, p.205.

Myers, C. and Aleksander, I. (1989) 'Output functions for probabilistic logic nodes', in *Proc. IEE International Conference on Artificial Neural Networks*, UK, pp.310–314.

Rabin, M.O. (1963) 'Probabilistic automata', *Information and Control*, Vol. 6, pp.230–245.

Ramanan, S., Petersen, R.S., Clarkson, T.G. and Taylor, J.G. (1995) 'pRAM nets for detection of small targets in sequence of infrared images', *Neural Networks*, Vol. 8, Nos. 7–8, pp.1227–1237.

Rohwer, R. and Morciniec, M. (1996) 'A theoretical and experimental account of n-tuple classifier performance', *Neural Computation*, Vol. 8, No. 3, pp.629–642.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) 'Learning internal representations by error propagation', in Rumelhart, D.E., McClelland, J.L. and the PDP Research Group (Eds.): *Parallel Distributed Processing*, MIT Press, Cambridge, MA, Vol. 1, pp.318–362.

Setiono, R., Leow, W. and Zurada, J. (2002) 'Extraction of rules from artificial neural networks for nonlinear regression', *IEEE Transactions on Neural Networks*, Vol. 13, No. 3, pp.564–577.

Thrun, S. (1991) 'The monk's problems: a performance comparison of different learning algorithms', Cmu-cs-91-197, Carnegie Mellon University, USA.

Towell, G.G. and Shavlik, J.W. (1993) The extraction of refined rules from knowledge-based neural networks', *Machine Learning*, Vol. 13, No. 1, pp.71–101.