

# Self-Organizing Neural Models Integrating Rules and Reinforcement Learning

Teck-Hou Teng, Zhong-Ming Tan, Ah-Hwee Tan, *Senior Member, IEEE*

**Abstract**—Traditional approaches to integrating knowledge into neural network are concerned mainly about supervised learning. This paper presents how a family of self-organizing neural models known as Fusion Architecture for Learning, COgnition and Navigation (FALCON) can incorporate *a priori* knowledge and perform knowledge refinement and expansion through reinforcement learning. Symbolic rules are formulated based on pre-existing know-how and inserted into FALCON as *a priori* knowledge. The availability of knowledge enables FALCON to start performing earlier in the initial learning trials. Through a temporal-difference (TD) learning method, the inserted rules can be refined and expanded according to the evaluative feedback signals received from the environment. Our experimental results based on a minefield navigation task have shown that FALCON is able to learn much faster and attain a higher level of performance earlier when inserted with the appropriate *a priori* knowledge.

## I. INTRODUCTION

Exciting and highly encouraging observations have been made from reinforcement learning systems which start off in a problem domain in complete oblivion and yet achieve impressive performance after repetitive trials and errors [1], [2]. However, it is always a zero-sum game in the sense that sufficient learning cycles are needed for a system to achieve the desired level of performance through repetitive trials. Learning efficiency, besides performance accuracy, is one of the other performance metrics that any sound solution should aim to optimize. A self-organizing neural model that saves precious time spent in learning from the ground up is needed.

A principled way to improve learning efficiency is by inserting *a priori* knowledge into a learning system during initialization. These are prerequisite knowledge about the constraints, limitations and operational characteristics of the problem domain. The field of computational intelligence offers a wide myriad of techniques for the discovery of knowledge from any conceivable perspective of the problem domain. Neural networks, decision trees, and statistical machine learning methods are the major approaches to learning logical rules for knowledge discovery and data understanding [3]. A learning system that is able to leverage on these *a priori* knowledge will be a great asset.

Previous approaches to integrating knowledge into neural network are mainly achieved through the supervised learning. Rules are used to initialize the structure of the neural network and refinement is achieved using the back-propagation algorithm [4], [5] or its variant [6]. Typical of most such

learning approaches, these works assume the rules that are used to initialize the neural networks to be quite complete. Also, the symbolic rules may lose their original meanings after being put through the weight tuning process of the back-propagation algorithm.

Such issues are addressed by Tan with a hybrid system known as Cascade ARTMAP [7] that incorporates symbolic knowledge into ART-based neural models. Cascade ARTMAP is further designed to perform multi-step inferring and pruning of the learnt rules. However, the work is still based on supervised learning and therefore it lacks the exploratory capability of the solution space adopted in the reinforcement learning approach.

Effort that combines *a priori* knowledge and reinforcement learning can be found in Kary's bi-memory model (BIMM) [8] that makes use of pre-existing knowledge to aid in the exploration of state space. The short-term memory of BIMM guides exploration through an action probability that is determined through the SLAP principle while the long-term memory is used for action-value learning. As all pre-existing knowledge is still learned by the neural model, there is no indication that it is able to build on external *a priori* knowledge.

The CLARION cognitive architecture [9] represents explicit knowledge as a rule network at the top level and acquires implicit knowledge using back-propagation neural network through reinforcement learning at the bottom level. While bottom-up transfer of implicit knowledge to the top level is achieved through the Rule-Extraction-Refinement algorithm [10], top-down learning guides the bottom level in the learning of explicit knowledge [11]. The two-level design, motivated by the study of implicit and explicit processes, nevertheless introduces the issues of efficiency and consistency.

This paper presents a family of self-organizing models known as FALCON that integrates symbolic rules and reinforcement learning. As a multi-channel ART-based neural network [12], FALCON is designed to perform online and incremental learning across multi-modal input patterns, involving state, action and reward. As FALCON's network structure is compatible with the symbolic rule-based representation, a class of reactive action rules can be inserted readily into FALCON at any point of the learning process. More importantly, the refinement of such rules and the discovery of new rules can be guided directly through a temporal difference (TD) reinforcement learning method, such as Q-Learning [13] [14]. In contrast to CLARION's two level architecture, FALCON presents an integrated solution

The authors are with the Nanyang Technological University, School of Computer Engineering and Intelligent Systems Centre, Nanyang Avenue, Singapore 639798

for representing both symbolic and learned knowledge.

Specifically in this paper, we present a rule translation and insertion algorithm for incorporating *a priori* knowledge into FALCON networks. Such knowledge is represented as symbolic IF-THEN rules, each of which is associated with a reward factor. Rules with greater reward factors lead to more favorable outcomes while those with negative rewards result in less desirable situations. Inserting rules into a neural model relieves its burden of having to discover these rules on its own. It is also an avenue to include rules that are not captured in the training cases but may yet be useful in the real situation. Rule structures that are not easily learned can also be included.

In most cases, the inserted rules may not be complete or optimal. They have to be constantly evaluated for their viability. Through a temporal difference reinforcement learning method, the reward factor of each rule is constantly moderated by FALCON using the feedback from the environment on the action that the rule recommends. In addition, new rules are generated from the explored actions that result in favorable outcomes. FALCON thus constantly maintains and expand its knowledge using the feedback signals from the environment to achieve optimal performance.

We have conducted extensive experiments using a mine-field navigation domain, wherein a variety of rules are inserted into FALCON before learning. The experiment results show that pre-loading FALCON with *a priori* knowledge leads to significantly better performance and learning efficiency. In addition, it is found that the quality of the rules can also influence the outcome of the experiments. Specifically, FALCON is able to perform better when it is pre-loaded with positive rules recommending do's than when it is pre-loaded with the negative ones recommending don't's.

The rest of the paper is organized as follows. The FALCON architecture with its algorithm is summarized in Section II. The rule representation schemes, the rule insertion algorithm as well as the inferencing framework are presented in Section III. Section IV presents the reinforcement learning algorithm that guides the FALCON's learning activity. The experiments and results are presented in Section V. The final section concludes and provides a brief discussion of future work.

## II. FALCON DYNAMICS

FALCON employs a 3-channel architecture (see Figure 1), comprising a cognitive field  $F_2^c$  and three input fields, namely a sensory field  $F_1^{c1}$  for representing current states, an action field  $F_1^{c2}$  for representing actions, and a reward field  $F_1^{c3}$  for representing reinforcement values. The generic network dynamics of FALCON, based on fuzzy ART operations [15], is described below.

**Input vectors:** Let  $\mathbf{S} = (s_1, s_2, \dots, s_n)$  denotes the state vector, where  $s_i \in [0, 1]$  indicates the sensory input  $i$ . Let  $\mathbf{A} = (a_1, a_2, \dots, a_m)$  denotes the action vector, where  $a_i \in [0, 1]$  indicates a possible action  $i$ . Let  $\mathbf{R} = (r, \bar{r})$  denotes the reward vector, where  $r \in [0, 1]$  is the reward signal

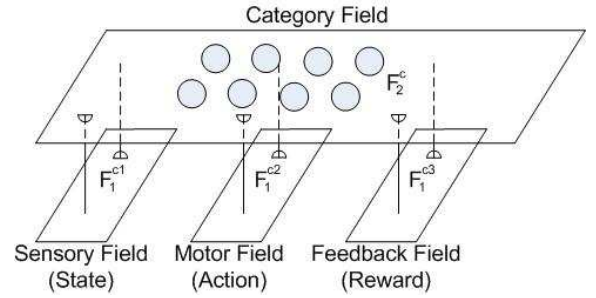


Fig. 1. The FALCON architecture.

value and  $\bar{r}$  (the complement of  $r$ ) is given by  $\bar{r} = 1 - r$ . Complement coding serves to normalize the magnitude of the input vectors and has been found to be effective in ART systems in preventing the code proliferation problem. As all input values of FALCON are assumed to be bounded between 0 and 1, normalization is necessary if the original values are not in the range of  $[0, 1]$ .

**Activity vectors:** Let  $\mathbf{x}^{ck}$  denotes the  $F_1^{ck}$  activity vector for  $k = 1, \dots, 3$ . Let  $\mathbf{y}^c$  denotes the  $F_2^c$  activity vector.

**Weight vectors:** Let  $\mathbf{w}_j^{ck}$  denotes the weight vector associated with the  $j^{\text{th}}$  node in  $F_2^c$  for learning the input patterns in  $F_1^{ck}$  for  $k = 1, \dots, 3$ . Initially,  $F_2^c$  contains only one *uncommitted* node and its weight vectors contain all 1's. When an *uncommitted* node is selected to learn an association, it becomes *committed*.

**Parameters:** FALCON's dynamics is influenced by choice parameters  $\alpha^{ck} > 0$  for  $k = 1, 2, 3$ ; learning rate parameters  $\beta^{ck} \in [0, 1]$  for  $k = 1, 2, 3$ ; contribution parameters  $\gamma^{ck} \in [0, 1]$  for  $k = 1, 2, 3$  where  $\sum_{k=1}^3 \gamma^{ck} = 1$ ; and vigilance parameters  $\rho^{ck} \in [0, 1]$  for  $k = 1, 2, 3$ .

**Code activation:** A bottom-up propagation process first takes place in which the activities (known as choice function values) of the cognitive nodes in the  $F_2^c$  field are computed. Specifically, given the activity vectors  $\mathbf{x}^{c1}$ ,  $\mathbf{x}^{c2}$  and  $\mathbf{x}^{c3}$  (in the input fields  $F_1^{c1}$ ,  $F_1^{c2}$  and  $F_1^{c3}$  respectively), for each  $F_2^c$  node  $j$ , the choice function  $T_j^c$  is computed as follows:

$$T_j^c = \sum_{k=1}^3 \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|} \quad (1)$$

where the fuzzy AND operation  $\wedge$  is defined by  $(\mathbf{p} \wedge \mathbf{q})_i \equiv \min(p_i, q_i)$ , and the norm  $|\cdot|$  is defined by  $|\mathbf{p}| \equiv \sum_i p_i$  for vectors  $\mathbf{p}$  and  $\mathbf{q}$ . In essence, the choice function  $T_j^c$  computes the similarity of the activity vectors with their respective weight vectors of the  $F_2^c$  node  $j$  with respect to the norm of individual weight vectors.

**Code competition:** The  $F_2^c$  node with the highest choice function is identified in a code competition process. The winner is indexed at  $J$  where

$$T_J^c = \max\{T_j^c : \text{for all } F_2^c \text{ node } j\} \quad (2)$$

When a category choice is made at node  $J$ ,  $y_J^c = 1$ ; and  $y_j^c = 0$  for all  $j \neq J$ . This indicates a winner-take-all strategy.

**Template matching:** Before code  $J$  can be used for learning, the template matching process checks the proximity of the weight template of code  $J$  to the corresponding activity pattern  $x^{ck}$ . Resonance occurs if for each channel  $k$ , the match function  $m_J^{ck}$  of the chosen code  $J$  meets its vigilance criterion:

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck} \quad (3)$$

The match function  $m_J^{ck}$  computes the similarity of the activity vector  $x^{ck}$  and weight vector  $w_J^{ck}$  with respect to the norm of the activity vector. Together, the choice and match functions work co-operatively to achieve stable coding and maximize code compression.

When resonance occurs, learning ensues. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function  $T_J^c$  is set to 0 for the duration of the input presentation. With a match tracking process, at the beginning of each input presentation, the vigilance parameter  $\rho^{c1}$  equals a baseline vigilance  $\bar{\rho}^{c1}$ . If a mismatch reset occurs,  $\rho^{c1}$  is increased until it is slightly larger than the match function  $m_J^{c1}$ . The search process then selects another  $F_2^c$  node  $J$  under the revised vigilance criterion until a resonance is achieved. This search and test process is guaranteed to end as FALCON will either find a *committed* node that satisfies the vigilance criterion or activate an *uncommitted* node which would definitely satisfy the criterion due to its initial weight values of 1s.

**Template learning:** Once a node  $J$  is selected, for each channel  $k$ , the weight vector  $\mathbf{w}_J^{ck}$  is modified by the following learning rule:

$$\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})}) \quad (4)$$

The learning rule adjusts the weight values towards the fuzzy AND of their original values and the respective weight values. The rationale is to learn by encoding the common attribute values of the input vectors and the weight vectors. For an uncommitted node  $J$ , the learning rates  $\beta^{ck}$  are typically set to 1. For committed nodes,  $\beta^{ck}$  can remain as 1 for fast learning or below 1 for slow learning in a noisy environment. When an uncommitted node is selected for learning, it becomes *committed* and a new uncommitted node is added to the  $F_2^c$  field. FALCON thus expands its network architecture dynamically in response to the input patterns.

### III. RULES REPRESENTATION AND INSERTION

As the knowledge structure of FALCON is compatible with the generalized modus ponens format, knowledge about the desirable and non-desirable actions for a given situation can be formulated as symbolic rules and inserted into a FALCON network. Initializing FALCON with pre-existing rules before learning serves to set up the global solution structure. This helps to improve learning efficiency and prediction accuracy. The subsequent sections contain the details of the symbolic rule handling mechanism of FALCON.

#### A. Rule Representation

Each symbolic rule is associated with a reward factor on the consequent in response to an antecedent. Let  $\mathbf{X}$  denotes the set of attribute-value pairs,  $\mathbf{X}^{rk}$  denotes the set of attribute-value pair  $av_i$  for the antecedents and  $\mathbf{Y}^{rk}$  denotes the set of attribute-value pair  $av_j$  for the consequents of a symbolic rule  $r_k$ . Each symbolic rule  $r_k$  has the following format:

$$\text{IF } \bigwedge_i av_i \text{ THEN } \mathbf{Y}^{rk} (\text{REWARD } p) \quad (5)$$

where  $av_i \in \mathbf{X}^{rk}$ ; and the antecedents  $\mathbf{X}^{rk}$  and the consequents  $\mathbf{Y}^{rk}$  are subsets of  $\mathbf{X}$  with the constraint of  $\mathbf{X}^{rk} \cap \mathbf{Y}^{rk} = 0$ . Each rule has a reward factor  $p$  defined as  $p \in [0, 1]$ . The attribute-value pair  $av_i$  as an element of  $\mathbf{X}$  can be defined as  $\{(x_i, v_{ih}) | v_{ih} \in V^{x_i}, x_i \in \mathbf{X}, i \in 1, \dots, |\mathbf{X}|, h \in 1, \dots, |V^{x_i}|\}$  where  $V^{x_i}$  is the domain of possible values for attribute  $x_i$ . For  $i \geq 2$ , the attribute-value pairs  $av_i$  are conjoined. Disjunctive relationship among the attributes exists among the rules with identical consequents.

#### B. Rule Translation

For insertion into the FALCON model, each symbolic rule has to be translated into an equivalent vector format. Specifically, each attribute-value pair  $av_i$  in a rule is converted into a complement-coded vector  $(v_i, v_i^c)$  such that

$$(v_i, v_i^c) = \begin{cases} (1, 0) & \text{if } x_i = v_{ih} \\ (0, 1) & \text{if } x_i \neq v_{ih} \\ (0, 0) & \text{if } x_i \text{ is not considered} \end{cases} \quad (6)$$

Each discrete-value attribute-value pair  $av_i$  can be translated as a complement-coded vector  $(v_i, v_i^c)$  where  $v_i^c = 1 - v_i$ . Each attribute  $x_i$  is expressed as a concatenation of complement-coded vector  $\{(v_i, v_i^c) | i \in 1, \dots, |V^{x_i}|\}$ . This concatenated vector representation is known as the attribute vector.

The dimension of the attribute vector for discrete-value attribute  $x_i$  is  $(2 \times |V^{x_i}|)$ . The translated antecedent is a concatenation of the attribute vectors. It has the dimension of  $(2 \times |\mathbf{X}| \times |V^{x_i}|)$ . All the key attributes are represented in the translated antecedent regardless of whether they are represented in the relevant rule. In the context of FALCON, the translated antecedent is known as the state vector  $\mathbf{S}$ , the translated consequent is known as the action vector  $\mathbf{A}$  and both of them are associated to reward vector  $\mathbf{R}$ . The consequent vector  $\mathbf{A}$  has the dimension of  $2 \times |\mathbf{Y}^{rk}| \times |V^{x_j}|$ . There is provision for multiple consequents in this solution. The reward vector  $\mathbf{R} = (r, r^c)$  is comprised of the reward factor  $p$  and its complement  $1 - p$ , i.e.,  $(r, r^c) = (p, 1 - p)$ .

The state-action-reward  $(\mathbf{S}, \mathbf{A}, \mathbf{R})$  triad tuple of the symbolic rule is learnt by FALCON as *a priori* knowledge. Complement coding is employed for the antecedent and consequent as well as the reward factor.

### C. Rule Insertion

After translation, the state  $\mathbf{S}$ , action  $\mathbf{A}$  and reward  $\mathbf{R}$  vectors are inserted into FALCON through the iterative performance of the code activation, code competition, template matching and template learning procedure. The algorithm that is presented in Table I outlines the rule insertion procedure.

TABLE I  
RULE INSERTION ALGORITHM

- 
- 1) Initialize the FALCON network
  - 2) Set  $\rho^{ck}$  to 1
  - 3) For each rule  $r_k$ 
    - a) Translate antecedent  $\mathbf{X}^{rk}$  into state vector  $\mathbf{S}$ ; consequent  $\mathbf{Y}^{rk}$  into action vector  $\mathbf{A}$  and reward  $p$  into reward vector  $\mathbf{R}$
    - b) Present  $\mathbf{S}$ ,  $\mathbf{A}$  and  $\mathbf{R}$  to input fields of FALCON
    - c) Invoke FALCON dynamic (describe in Section II) to insert the translated rule.
  - 4) Repeat from Step 2 to Step 6 for all the rules.
- 

During rule insertion, the vigilance parameters  $\rho^{ck}$  are each set to 1 to ensure that only identical attribute vectors are grouped into one recognition category. Contradictory symbolic rules are detected during rule insertion when identical input attribute vectors are associated with distinct output attribute vectors. The detection is achieved through a *perfect mismatch* phenomenon, in which the system tries to raise sensory field vigilance  $\rho^{c1}$  above 1 in response to a mismatch in the motor field.

In essence, FALCON is taught that by applying action  $\mathbf{A}$  to state  $\mathbf{S}$  will provide it with reward  $\mathbf{R}$ . This accomplishes the basis of a rule. A cognitive node with the  $(\mathbf{S}, \mathbf{A}, \mathbf{R})$  triad tuple establishes such a correlation. Hence, there are as many cognitive nodes as the number of inserted rules. The inserted rules are temporarily assumed to be distinct and viable. Refinement of these *a priori* knowledge is achieved under the guidance of the temporal difference reinforcement learning method which is detailed in the subsequent section.

## IV. REINFORCEMENT LEARNING

Though the rules may be based on sound observations, the heuristic nature of these rules is an indication of their inherent inaccuracies and inconsistencies. Various constraints as well as the vast complexity of the problem domain demand further augmentations of the knowledge base of FALCON. It is a non-trivial task to ensure completeness of a knowledge base for complex problem domains. Autonomous learning capability such as reinforcement learning presents itself as a viable approach in providing an adequate coverage of the solution space.

TD-FALCON is an extension of FALCON to incorporate Temporal Difference (TD) methods to estimate and learn value functions of action-state pairs  $Q(s, a)$  that indicates the goodness for a learning system to take a certain action  $a$  in a given state  $s$ . Such value functions are then used in

the action selection mechanism, also known as the *policy*, to decide on the method in which an action shall be selected. The *policy* decides whether an action shall be identified through exploration of the solution space or exploitation of the knowledge base.

Given the current state  $s$ , it first decides between exploration and exploitation by following an action selection policy. For exploration, a random action is picked. For exploitation, it searches for optimal action through a direct code access procedure. Upon receiving a feedback from the environment after performing the action, a temporal difference formula is used to compute a new estimate of the Q value of performing the chosen action in the current state. The new Q value is then used as the teaching signal for TD-FALCON to learn the association of the current state and the chosen action to the estimated Q value.

### A. Action Selection Policy

The simplest action selection policy is to pick the action with the highest value predicted by the TD-FALCON network. However, a key requirement of autonomous agents is to explore the environment. This is especially important for an agent to function in situations without immediate evaluative feedback. If an agent keeps selecting the optimal action that it believes, it will not be able to explore and discover better alternative actions. There is thus a fundamental tradeoff between *exploitation*, i.e., sticking to the best actions believed, and *exploration*, i.e., trying out other seemingly inferior and less familiar actions.

The  $\epsilon$ -greedy policy selects the action with the highest value with a probability of  $1 - \epsilon$  and takes a random action with probability  $\epsilon$  [16]. With a constant  $\epsilon$  value, the agent will always explore the environment with a fixed level of randomness. In practice, it may be beneficial to have a higher  $\epsilon$  value to encourage the exploration of paths in the initial stage and a lower  $\epsilon$  value to optimize the performance by exploiting familiar paths in the later stage. A decay  $\epsilon$ -greedy policy is thus adopted to gradually reduce the value of  $\epsilon$  over time. The rate of decay is typically inversely proportional to the complexity of the environment as a more complex environment with larger state and action spaces will take a longer time to explore.

### B. Direct Code Access

During the inference process, FALCON attempts to seek out a reasonable response to the current situation. The situation is encoded in the same format as the antecedent vector. It is presented as the state vector  $\mathbf{S}$  to FALCON. Through a direct code access procedure [17], FALCON searches for the cognitive node which matches with the current state and has the maximal reward value. The TD-FALCON algorithm with direct code access is as shown in Table II.

For direct code access, the activity vectors  $\mathbf{x}^{c1}$ ,  $\mathbf{x}^{c2}$ , and  $\mathbf{x}^{c3}$  are initialized by  $\mathbf{x}^{c1} = \mathbf{S}$ ,  $\mathbf{x}^{c2} = (1, \dots, 1)$ , and  $\mathbf{x}^{c3} = (1, 0)$ . FALCON then performs code activation and

TABLE II  
TD-FALCON WITH DIRECT CODE ACCESS ALGORITHM

- 1) Initialize the TD-FALCON network with *a priori* knowledge.
- 2) Sense the environment and formulate a state representation  $s$ .
- 3) Follow an action selection policy to make a choice between exploration and exploitation.
  - a) If exploration, take a random action.
  - b) If exploitation, identify the encoded action  $a$  with the maximal  $Q(s,a)$  value by presenting the state vector  $\mathbf{S}$ , the action vector  $\mathbf{A} = (1, \dots, 1)$ , and the reward vector  $\mathbf{R} = (1, 0)$  to TD-FALCON.
- 4) Perform the action  $a$ , observe the next state  $s$  and receive a reward  $r$  (if any) from the environment.
- 5) Estimate the revised value function  $Q(s,a)$  following a Temporal Difference formula such as  $\delta Q(s,a) = \alpha TD_{err}$ .
- 6) Present the corresponding state  $\mathbf{S}$ , action  $\mathbf{A}$  and reward(Q-value)  $\mathbf{R}$  vectors to TD-FALCON for learning.
- 7) Update the current state by  $s = s'$ .
- 8) Repeat from Step 2 until  $s$  is a terminal state.

code competition according to equations (1) and (2) to select a cognitive node.

Upon selecting a winning  $F_2^c$  node  $J$ , the chosen node  $J$  performs a readout of its weight vector to the action field  $F_1^{c2}$  such that

$$\mathbf{x}^{c2(\text{new})} = \mathbf{x}^{c2(\text{old})} \wedge \mathbf{w}_J^{c2} \quad (7)$$

An action  $a_I$  is then chosen, which has the highest activation value

$$x_I^{c2} = \max\{x_i^{c2(\text{new})} : \text{for all } F_1^{c2} \text{ node } i\} \quad (8)$$

The state vector  $\hat{S}_i$  is the only input to FALCON. In exploitation mode, it seeks out a cognitive node  $j$  whose state vector  $\mathbf{S}_j$  has the best match with state vector  $\mathbf{S}_i$  shall provide its action vector  $\mathbf{A}_j$  as the output. A best match is determined through the vigilance criterion during *template matching*. This requires the vigilance parameter  $\rho^{pk}$  to have a value between 0 and 1, i.e.,  $\rho^{pk} \in (0, 1)$ . The closer  $\rho^{pk}$  is to 1, the stronger the match is required to be and vice versa. The action vector of the chosen code  $J$  which satisfies the vigilance criterion is decoded to obtain the symbol equivalent to be acted upon by the agent on the environment.

### C. Value Function Learning

A typical Temporal Difference equation for iterative estimation of value functions  $Q(s,a)$  is given by

$$\Delta Q(s,a) = \alpha TD_{err} \quad (9)$$

where  $\alpha \in [0, 1]$  is the learning parameter and  $TD_{err}$  is a function of the current Q-value predicted by TD-FALCON and the Q-value newly computed by the temporal difference formula.

TD-FALCON employs a Bounded Q-learning rule, wherein the temporal error term is computed by

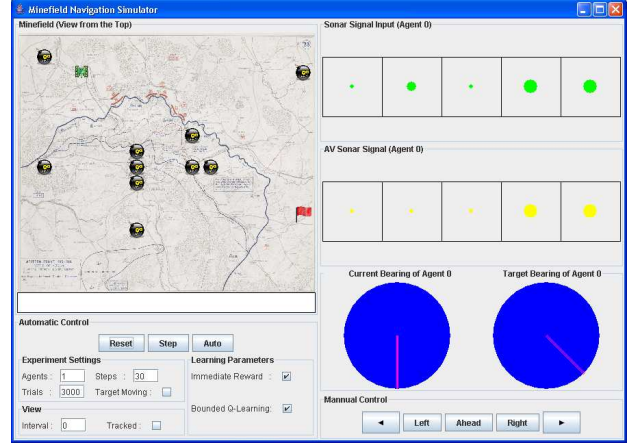


Fig. 2. The minefield navigation simulator.

$$\Delta Q(s,a) = \alpha TD_{err} (1 - Q(s,a)) \quad (10)$$

where  $TD_{err} = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ , of which  $r$  is the immediate reward value,  $\gamma \in [0, 1]$  is the discount parameter, and  $\max_{a'} Q(s', a')$  denotes the maximum estimated value of the next state  $s'$ . It is important to note that the Q values involved in estimating  $\max_{a'} Q(s', a')$  are computed by the same TD-FALCON network itself and not by a separate reinforcement learning system. The Q-learning update rule is applied to all states that the agent traverses. With value iteration, the value function  $Q(s, a)$  is expected to converge to  $r + \gamma \max_{a'} Q(s', a')$  over time. By incorporating the scaling term  $1 - Q(s, a)$ , the adjustment of Q values will be self-scaling so that they will not be increased beyond 1. The learning rule thus provides a smooth normalization of the Q values. If the reward value  $r$  is constrained between 0 and 1, we can guarantee that the Q values will remain to be bounded between 0 and 1.

## V. EXPERIMENTS

In the minefield navigation domain, an autonomous vehicle (AV) is tasked to navigate a minefield (see Fig. 2) to a randomly placed target destination within a specified time frame. The mines are randomly distributed throughout the minefield. The task fails when the AV hits a mine or when it is unable to reach the target destination using the allocated number of steps. The task is successful when the AV reaches the target destination within the allocated number of steps and without moving into any mine along the way. The AV starts to move from a random vacate location in the minefield. It acts out its existence through a repetitive cycle of sense, act and learn. All the experiments are based on a 16 by 16 minefield consisting of ten mines. In addition, the target destination and the mines remain stationary during the trial.

The sonar readings and the relative target bearings are taken into consideration in the symbolic rules. Sonar readings of the mines are available from five positions - left, left diagonal, front, right diagonal and right. For each direction

$i$ , the sonar signal is measured by  $s_i = \frac{1}{d_i}$ , where  $d_i$  is the distance to an obstacle (that can be a mine or the boundary of the minefield) in the  $i$  direction. When the sonar reading of a particular direction is 1, this indicates that the mine is adjacent to the AV. The other input attribute is the relative bearing of the target from the current position. There are eight target bearings - front, diagonal right, right, diagonal back right, back, diagonal back left, left and diagonal left. Table IV contains  $P1$  rules that recommend the best possible actions. Table V contains the  $P2$  rules which are the supplementary rules that recommend the second best actions. Table III contains the  $N$  rules which are the negative rules. Whereas a positive rule recommends action(s) leading to a positive outcome, a negative rule highlights actions leading to undesirable situations.

TABLE III  
N RULES

NR1:	IF	LeftSonar = 1
	THEN	Move Left (REWARD = 0)
NR2:	IF	DiagonalLeftSonar = 1
	THEN	Move DiagonalLeft (REWARD = 0)
NR3:	IF	FrontSonar = 1
	THEN	Move Front (REWARD = 0)
NR4:	IF	DiagonalRightSonar = 1
	THEN	Move DiagonalRight (REWARD = 0)
NR5:	IF	RightSonar = 1
	THEN	Move Right (REWARD = 0)

Each symbolic rule leads to a decision on the movement direction. The AV can choose from one out of the five possible actions, namely move left, move diagonally left, move straight ahead, move diagonally right, and move right. At the end of a trial, a reward of 1 is given when the AV reaches the target. A reward of 0 is given when the AV fails to reach the target destination or when it moves into a mine. For the delayed feedback scheme, no immediate feedback is given at each step of the trial.

A TD-FALCON network containing 18 nodes in the sensory field (representing  $5 \times 2$  complement-coded sonar signals and 8 target bearing values), five nodes in the action field, and two nodes in the reward field (representing the complement-coded function value) is included as the brain of the AV. TD-FALCON employed a set of parameter values obtained through empirical experiments: choice parameters  $\alpha^{c1} = 0.1, \alpha^{c2} = 0.001, \alpha^{c3} = 0.001$ ; learning rate  $\beta^{ck} = 1.0$  for  $k = 1, 2, 3$  for fast learning; contribution parameters  $\gamma^{c1} = \gamma^{c2} = \gamma^{c3} = \frac{1}{3}$ ; baseline vigilance parameters  $\bar{\rho}^{c1} = 0.25$  and  $\bar{\rho}^{c2} = 0.2$  for a marginal level of match requirement in the state and action spaces, and  $\bar{\rho}^{c3} = 0.5$  for a stricter match criterion on reward values. For Temporal Difference learning, the learning rate  $\alpha$  was fixed at 0.5 and the discount factor  $\gamma$  was set to 0.9. The initial Q value was set to 0.5. For action selection policy,  $\epsilon$  was initialized to 0.5 and decayed at a rate of 0.0005 until it dropped to 0.005.

TD-FALCON is evaluated for its ability to response ap-

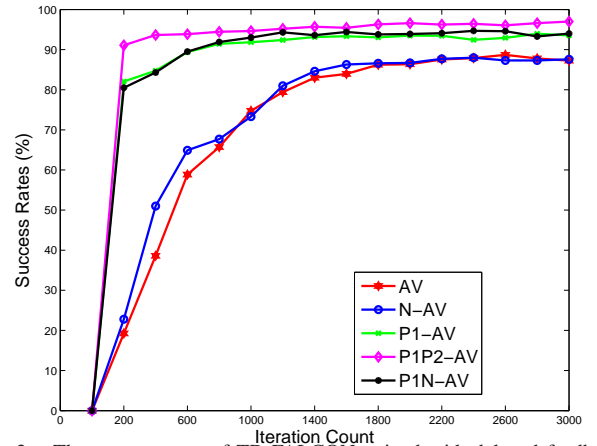


Fig. 3. The success rates of TD-FALCON trained with delayed feedback.

propriately to same task configuration under five different initialization schemes. TD-FALCON that is trained up to 3000 iterations using Q-Learning with delayed feedback is used for all the schemes. Each of these initialization schemes differs in the rule set used to initialize TD-FALCON for the navigation task in the Minefield Simulation Platform. From the legend of the plots, AV is a TD-FALCON that has not been initialized with any *a priori* knowledge. This forms the baseline for the performance comparison;  $P1$ -AV is a TD-FALCON that is pre-loaded with a set of  $P1$  rules (see Table IV);  $P1P2$ -AV is a TD-FALCON that is pre-loaded with the combination of  $P1$  rules and  $P2$  rules, i.e.,  $P1+P2$  rules (see Table V for  $P2$  rules);  $N$ -AV is a TD-FALCON that is pre-loaded with a set of  $N$  rules (see Table III) and  $P1N$ -AV is a TD-FALCON that is pre-loaded with the combination of  $P1$  rule and  $N$  rules.

The success rates of the AVs using various rule initialization schemes are shown in Figure 3. We can see that AV has a much more modest level of success rates over the increasing number of training iterations as compared to the other AVs. It only achieve a success rate of 19.3% after 200 iterations. The success rates begins to plateau near the 87% mark after around 2200 iterations. Its success rates peak at around 88.7% after 2600 iterations before retreating to 87.3% level at 3000 iterations. All these observations indicate that TD-FALCON has reached a local saturation point. It is not able to improve its performance beyond the 90% level.  $P1$ -AV, on the other hand, exhibits a marked improvement in performance. The success rates of  $P1$ -AV shoots right up to the 82.1% level only after 200 iteration. After some minor corrections, the success rates begin to stabilize after only 1000 iterations at a higher level of around 93%. Notably, its peak success rates performance is 6% more than that of the AV without rule insertion. This clearly shows that inserting rules into TD-FALCON has lifted its performance.

With the insertion of the rule sets  $P1$  and  $P2$ ,  $P1P2$ -AV shows an even higher level of performance. Specifically, the success rates achieve 91.1% only after 200 iterations. Its performance which stabilizes after just 600 iterations has the highest success rate of 97% among all the AVs. This

TABLE IV  
P1 RULES

P1R1:	IF	FrontSonar $\neq$ 1 AND TargetBearing = front
	THEN	Move Front (REWARD = 1)
P1R2:	IF	DiagonalRightSonar $\neq$ 1 AND TargetBearing = DiagonalRight
	THEN	Move DiagonalRight (REWARD = 1)
P1R3:	IF	RightSonar $\neq$ 1 AND TargetBearing = Right
	THEN	Move Right (REWARD = 1)
P1R4:	IF	RightSonar $\neq$ 1 AND TargetBearing = DiagonalBackRight
	THEN	Move Right (REWARD = 1)
P1R5:	IF	LeftSonar $\neq$ 1 AND TargetBearing = Back
	THEN	Move Left (REWARD = 1)
P1R6:	IF	LeftSonar $\neq$ 1 AND TargetBearing = DiagonalBackLeft
	THEN	Move Left (REWARD = 1)
P1R7:	IF	LeftSonar $\neq$ 1 AND TargetBearing = Left
	THEN	Move Left (REWARD = 1)
P1R8:	IF	DiagonalLeftSonar $\neq$ 1 AND TargetBearing = DiagonalLeft
	THEN	Move DiagonalLeft (REWARD = 1)

TABLE V  
P2 RULES

P2R1:	IF	DiagonalLeftSonar $\neq$ 1 AND FrontSonar = 1 AND TargetBearing = front
	THEN	Move DiagonalRight (REWARD = 1)
P2R2:	IF	DiagonalRightSonar = 1 AND RightSonar $\neq$ 1 AND TargetBearing = back
	THEN	Move Right (REWARD = 1)
P2R3:	IF	DiagonalRightSonar $\neq$ 1 AND RightSonar = 1 AND TargetBearing = Right
	THEN	Move DiagonalRight (REWARD = 1)
P2R4:	IF	DiagonalRightSonar $\neq$ 1 AND RightSonar = 1 AND TargetBearing = DiagonalBackRight
	THEN	Move DiagonalRight (REWARD = 1)
P2R5:	IF	RightSonar $\neq$ 1 AND TargetBearing = Back
	THEN	Move Right (REWARD = 1)
P2R6:	IF	LeftSonar = 1 AND DiagonalLeftSonar $\neq$ 1 AND TargetBearing = DiagonalBackLeft
	THEN	Move DiagonalLeft (REWARD = 1)
P2R7:	IF	LeftSonar = 1 AND DiagonalLeftSonar $\neq$ 1 AND TargetBearing = Left
	THEN	Move DiagonalLeft (REWARD = 1)
P2R8:	IF	LeftSonar $\neq$ 1 AND DiagonalLeftSonar = 1 AND TargetBearing = DiagonalLeft
	THEN	Move Left (REWARD = 1)

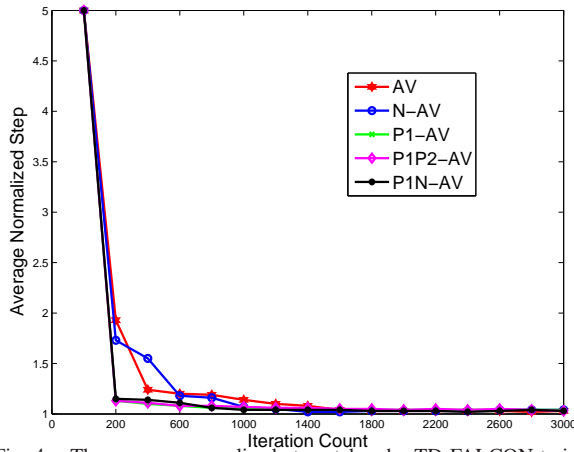


Fig. 4. The average normalized steps taken by TD-FALCON trained with delayed feedback.

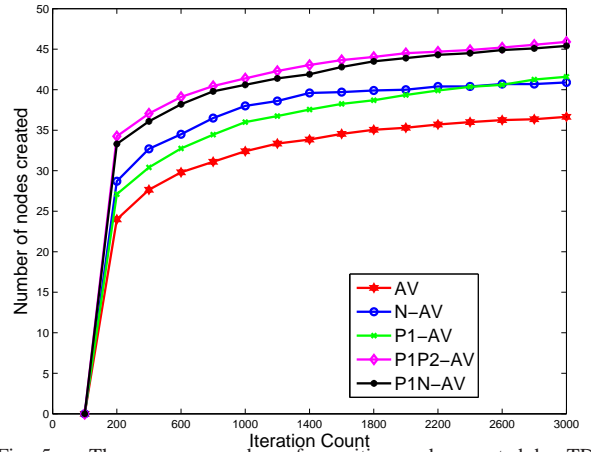


Fig. 5. The average number of cognitive nodes created by TD-FALCON trained with delayed feedback.

has clearly highlighted the merit of inserting positive rules into TD-FALCON. With the negative rules inserted into TD-FALCON, *N-AV* reveals a rather modest improvement in performance. Its performance stabilizes earlier than *AV* at around 1800 iterations and its improvement is marginal and is only observable in the initial learning trials. Its success rates peak at 88% after 2400 iterations. *P1N-AV* has a slightly lackluster performance than that of *P1P2-AV* but it is still better than *P1-AV*. Though the insertion of negative rules may be beneficial, its benefits is diminished in TD-FALCON that has been inserted with positive rules.

From the above observations, it can be concluded that TD-FALCON that is not initialized with any prior knowledge is not able to achieve the same level of success over the same learning trials as those that have been pre-loaded with some form of *a priori* knowledge. The merits of having pre-existing knowledge is immediately apparent from the comparison among the success rates plots against the baseline.

To evaluate how well the *AV* is able to traverse from its initial position to its target destination, we define a measure called *normalized step* given by  $step_n = \frac{step}{sd}$ , where *step* is the number of sense-move-learn cycles taken to reach the target and *sd* is the shortest distance between the starting and target positions. A normalized step of 1 means the system has taken the optimal (shortest) path to the target destination. Figure 4 shows that *P1-AV*, *P1P2-AV* and *P1N-AV* all take a close to optimal path to the target destination after only 200 iterations. *AV* and *N-AV* require more iterations before it acquires the capability to move along an approximately optimal path to the target destination. This clearly illustrates the beneficial effect of inserting positive logic rules over negative logic rules and not inserting any rules at all.

Inserted rules are represented as cognitive nodes in TD-FALCON. Each of these nodes maps a state vector **S** to its action vector **A** and the reward vector **R**. The network complexity plot in Figure 5 shows the node count at a 200-iteration interval. It shows *AVs* pre-loaded with rules have a higher number of node counts at the earlier iterations. With reference back to Figure 3, *AVs* with higher node count have higher success rate. However, a noteworthy observation is made between the node counts of *P1N-AV* and *P1-AV*. At 200 iterations, *P1-AV* has around 27 nodes and manages to achieve 82.1% success rate while *P1N-AV* manages only 80.5% success rate with around 33 nodes. It has 6 more nodes yet its success rate is 1.55% lower. *P1N-AV* has more nodes than *P1-AV* because it is pre-loaded with *P1* and *N* rules. However these extra nodes do not translate into better success rates. This is an indication that the quality of the nodes has an important role in the performance of the TD-FALCON. Therefore it is necessary to refine the rules as represented by these cognitive nodes through the reinforcement learning approach to ensure optimal performance.

## VI. CONCLUSION

We have shown that inserting FALCON with *a priori* knowledge greatly enhances its performance for the naviga-

tion task towards the target destination in the minefield simulation platform. FALCON is known to be able to perform well with zero knowledge. With *a priori* knowledge, it is able to perform even better. Thus, it is concluded that it is always better to bootstrap FALCON with sound *a priori* knowledge about the problem domain. In the experiments, FALCON has also been inserted with different types of knowledge. It is shown that positive knowledge is able to provide much greater improvement to the performance of FALCON while negative knowledge is able to lift the performance of FALCON slightly. It is also observed that the quality rather than the quantity of the inserted rules has a greater influence on the performance of FALCON. Subsequent extension to this effort may include further investigation on various learning paradigms, the use of novel techniques for obtaining better quality rules as well as to expand on the capability of FALCON to handle more expressive rules.

## REFERENCES

- [1] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998.
- [2] A.H. Tan, "FALCON: A fusion architecture for learning, cognition, and navigation," in *Proceedings of the International Joint Conference on Neural Networks*, 2004, pp. 3297–3302.
- [3] J. M. Žurada W. Duch, R. Setiono, "Computational intelligence methods for rule-based data understanding," in *Proceedings of the IEEE*. IEEE, May 2004, vol. 92, pp. 771–805.
- [4] L. M. Fu and L. C. Fu, "Mapping rule-based knowledge into neural architecture," *Knowledge-Based Systems*, vol. 3, pp. 48–56, 1990.
- [5] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of approximately correct domain theories by knowledge-based neural networks," in *Proceedings, 8th National Conference on AI, Boston, MA*, 1990, pp. 861–866.
- [6] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, "Backpropagation learning in expert networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 62–72, 1992.
- [7] A.H. Tan, "Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 237–250, March 1997.
- [8] Kary Främling, "Guiding exploration by pre-existing knowledge without modifying reward," *Neural Networks*, vol. 20, pp. 736–747, January 2007.
- [9] Ron Sun, *Cognition and Multi-Agent Interaction*, chapter The CLARION Cognitive Architecture: Extending Cognitive Modeling to Social Simulation, Cambridge University Press, 2006.
- [10] Ron Sun, Edward Merrill, and Todd Peterson, "From implicit skills to explicit knowledge: a bottom-up model of skill learning," *Cognitive Science*, pp. 203–244, 2001.
- [11] Ron Sun, "Robust reasoning: integrating rule-based and similarity-based reasoning," *Artificial Intelligence*, vol. 75, pp. 241–295, 1995.
- [12] G.A. Carpenter and S. Grossberg, "Adaptive Resonance Theory," in *The Handbook of Brain Theory and Neural Networks*, M.A. Arbib, Ed., pp. 87–90. Cambridge, MA: MIT Press, 2003.
- [13] C.J.C.H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.
- [14] A.H. Tan, N. Lu, and D. Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 230–244, February 2008.
- [15] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.
- [16] A. Pérez-Urbe, *Structure-Adaptable Digital Neural Networks*, Ph.D. thesis, Swiss Federal Institute of Technology-Lausanne, 2002.
- [17] A.H. Tan, "Direct code access in self-organizing neural networks for reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Hyderabad, India, January 2007, pp. 1071–1076.