Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems | School of Computing and Information Systems

2-2008

Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Ning LU

Dan XIAO

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Computer Engineering Commons, Databases and Information Systems Commons, and the OS and Networks Commons

Citation

TAN, Ah-hwee; LU, Ning; and XIAO, Dan. Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. (2008). *IEEE Transactions on Neural Networks*. 9, (2), 230-244.
Available at: https://ink.library.smu.edu.sg/sis_research/5237

This Journal Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning With Delayed Evaluative Feedback

Ah-Hwee Tan, *Senior Member, IEEE*, Ning Lu, and Dan Xiao

***Abstract*—This paper presents a neural architecture for learning category nodes encoding mappings across multimodal patterns involving sensory inputs, actions, and rewards. By integrating adaptive resonance theory (ART) and temporal difference (TD) methods, the proposed neural model, called TD fusion architecture for learning, cognition, and navigation (TD-FALCON), enables an autonomous agent to adapt and function in a dynamic environment with immediate as well as delayed evaluative feedback (reinforcement) signals. TD-FALCON learns the value functions of the state–action space estimated through on-policy and off-policy TD learning methods, specifically state–action–reward–state–action (SARSA) and Q-learning. The learned value functions are then used to determine the optimal actions based on an action selection policy. We have developed TD-FALCON systems using various TD learning strategies and compared their performance in terms of task completion, learning speed, as well as time and space efficiency. Experiments based on a minefield navigation task have shown that TD-FALCON systems are able to learn effectively with both immediate and delayed reinforcement and achieve a stable performance in a pace much faster than those of standard gradient–descent-based reinforcement learning systems.**

***Index Terms*—Reinforcement learning, self-organizing neural networks (NNs), temporal difference (TD) methods.**

I. Introduction

REINFORCEMENT learning [1] is an interaction-based paradigm wherein an autonomous agent learns to adjust its behavior according to feedback received from the environment. The learning paradigm is consistent with the notion of embodied cognition that intelligence is a process deeply rooted in the body's interaction with the world [2]. Often formalized as a Markov decision process (MDP) [1], an autonomous agent performs reinforcement learning through a sense, act, and learn cycle. First, the agent obtains sensory input from the environment representing the current state ($\mathbf{S}$). Depending on the current state and its knowledge and goals, the system selects and performs the most appropriate action ($\mathbf{A}$). Upon receiving feedback in terms of rewards ($\mathbf{R}$) from the environment, the agent learns to adjust its behavior in the motivation of receiving positive rewards in the future. It is important to note that reward signals may not always be available in a real-world environment. When immediate evaluative feedback is absent, the system will have to internally compute an estimated payoff value for the purpose of learning.

Classical approaches to the reinforcement learning problem generally involve learning one or both of the following functions, namely, *policy function* which maps each state to a desired action and *value function* which associates each pair of state and action to a utility value. The learning problem is closely related to the problem of determining optimal policies in discrete-time dynamic systems, of which dynamic programming (DP) provides a principled solution. The problem of the DP approach is that mappings must be learned for each and every possible state or each and every possible pair of state and action. This causes a scalability issue for continuous and/or very large state and action spaces.

This paper describes a natural extension of a family of self-organizing neural networks (NNs), known as adaptive resonance theory (ART) [3], for developing an integrated reinforcement learner. Whereas predictive ART performs supervised learning through the pairing of teaching signals and the input patterns [4], [5], the proposed neural architecture, known as fusion architecture for learning, cognition, and navigation (FALCON), learns multichannel mappings simultaneously across multimodal input patterns, involving states, actions, and rewards, in an online and incremental manner. Using competitive coding as the underlying adaptation principle, the network dynamics encompasses a myriad of learning paradigms, including unsupervised learning, supervised learning, as well as reinforcement learning.

The first FALCON system developed is a reactive model, known as R-FALCON, that learns a policy directly by creating category nodes, each associating a current state to a desirable action [6]. A positive feedback reinforces the selected action, whereas a negative experience results in a reset, following which the system seeks alternative actions. The strategy is to associate a state with an action that will lead to a desirable outcome. As the reactive model relies on the availability of immediate feedback signals, it is not applicable to problems in which the merit of an action is only known several steps after the action is performed.

To overcome this inadequacy, this paper presents a family of deliberative models that learns the value functions of the state–action space estimated through temporal difference (TD) algorithms. Whereas a *reactive* model learns to match a given state directly to an optimal action, a *deliberative* model learns to weigh the consequences of performing all possible actions

Manuscript received November 18, 2005; revised May 30, 2006 and January 22, 2007; accepted May 25, 2007.

The authors are with the School of Computer Engineering and Intelligent Systems Centre, Nanyang Technological University, Singapore 639798, Singapore (e-mail: asahtan@ntu.edu.sg; xiao0002@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2007.905839

1045-9227/$25.00 © 2007 IEEE

in a given state before selecting an action. We develop various types of TD-FALCON systems using TD methods, specifically, Q-learning [7], [8] and state–action–reward–state–action (SARSA) [9]. The learned value functions are then used to determine the optimal actions based on an action selection policy. To achieve a balance between exploration and exploitation, we adopt a hybrid action selection policy that favors exploration initially and gradually leans towards exploitation.

Experiments on TD-FALCON have been conducted based on a case study on minefield navigation. The task involves an autonomous vehicle (AV) learning to navigate through obstacles to reach a stationary target (goal) within a specified number of steps. Experimental results have shown that using the proposed TD-FALCON models, the AV adapts in real time and learns to perform the task rapidly in an online manner. Benchmark experiments have also been conducted to compare TD-FALCON with two gradient–descent-based reinforcement learning systems. The first system, called BP-Q learner, employs the standard Q-learning rule with a multilayer feedforward NN trained by the backpropagation (BP) learning algorithm as the function approximator [7], [10], [11]. The second system is direct neural dynamic programming (NDP) [12], belonging to a class of adaptive critic designs (ACDs), known as action-dependent heuristic dynamic programming (ADHDP). The results indicate that TD-FALCON learns significantly faster than the two gradient–descent-based reinforcement learners, at the expense of creating larger networks.

The rest of this paper is organized as follows. Section II provides a review on related work. Section III introduces the FALCON architecture and the associated learning and prediction algorithms. Section IV provides a summary of the reactive FALCON model. Section V presents the TD-FALCON algorithm, specifically, the action selection policy and the value function estimation mechanism. Section VI describes the minefield navigation experiments and presents the simulation results. Section VII analyzes the time and space complexity of TD-FALCON, comparing with BP-Q and direct NDP. The final section concludes and discusses limitations and future work.

II. Related Work

Over the years, many approaches and designs have been proposed and used in different disciplines to deal with the scalability problem of reinforcement learning. A family of approximate dynamic programming (ADP) systems [13], [14], most notably based on ACDs, has been steadily developed, which employs function approximators to learn both policy and value functions by iterating between policy optimization and value estimation. A typical ACD system consists of an actor for learning the action policy and a critic for learning the value or cost function. Most ADP systems do not constrain the use of function approximators. Applicable to function approximation are many statistical and supervised learning techniques, including gradient-based multilayer feedforward NNs [also known as multilayer perceptron (MLP)] [10], [15], [16], generalized adalines [17], decision tree [18], fuzzy logic [19], cerebellar model arithmetic computer (CMAC, also known as tile coding) [20], radial basis function (RBF) [1], [21], and extreme learning machines (ELMs) [22], [23].

Among these methods, multilayer perceptron (MLP) with the gradient–descent-based BP learning algorithm has been used widely in many reinforcement learning systems and applications, including complementary reinforcement backpropagation algorithm (CRBP) [15], Q-AHC [24], backgammon [25], connectionist learning with adaptive rule induction online (CLARION) [11], and ACDs [12], [26]. The BP learning algorithm, however, makes small error correction steps and typically requires an iterative learning process. In addition, there is an issue of instability as learning of new patterns may erode the previously learned knowledge. Consequently, the resultant systems may not be able to learn and operate in real time. Compared with the gradient–descent approach, linear function approximators such as CMAC and RBF often learn faster but at the expense of using more internal nodes or basis functions. A variant of RBF networks called resource allocation networks (RAN) [27] further adds locally tuned Gaussian units to the existing network structure dynamically as and when necessary. This idea of dynamic resource allocation has been adopted in a Q-learning system with a restarting strategy for reinforcement learning [28]. More recently, reinforcement learning systems with dynamic allocation and elimination of basis functions have also been proposed [29].

Instead of using supervised learning to approximate the value functions directly, *unsupervised* learning NNs, such as self-organizing map (SOM), can be used for the representation and generalization of continuous state and action spaces [30], [31]. The state and action clusters are then used as the entries in a traditional Q-value table implemented separately. Using a localized representation, SOM has the advantage of more stable learning, compared with gradient–descent NNs based on distributed representation. However, SOM remains an iterative learning system, requiring many rounds to converge. In addition, SOM is expected to scale badly if the dimensions of the state and action spaces are significantly higher than the dimension of the map [30].

A recent approach to reinforcement learning builds upon ART [3], also a class of self-organizing NNs, but with very distinct characteristics from SOM. Through a unique code stabilizing and dynamic network expansion mechanism, ART models are capable of learning multidimensional mappings of input patterns in an online and incremental manner. Whereas various models of ART and their predictive (supervised learning) versions have been widely applied to pattern analysis and recognition tasks [4], [5], there have been few attempts to use ART-based networks for reinforcement learning. Ueda *et al.* [32] adopt an approach similar to that of SOM using unsupervised ART models to learn the clusters of state and action patterns. The clusters are then used as the compressed states and actions by a separate Q-learning module. Another line of work by Ninomiya [33] couples a supervised ART system with a TD reinforcement learning module in a hybrid architecture. While the states and actions in the reinforcement module are exported from the supervised ART system, the two learning systems operate independently. This redundancy in representation unfortunately leads to instability and an unnecessarily long processing time in action selection and learning of value functions.

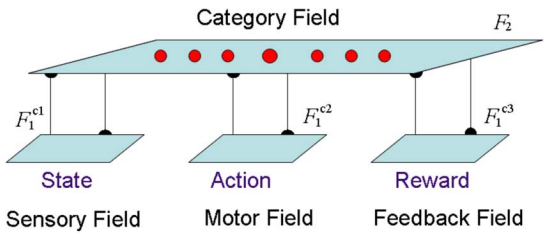


Fig. 1. FALCON architecture.

Compared with these ART-based systems [32], [33], our proposed FALCON model presents a truly integrated solution in the sense that there is no implementation of a separate reinforcement learning module or Q-value table. Comparing with RBF-based systems, the category nodes of FALCON are similar to the basis functions. Also, the inherent capability of ART in creating category nodes dynamically in response to incoming patterns is also found in dynamically allocated RBF networks. However, the output of RBF is based on a linear combination of RBFs whereas FALCON uses a winner-take-all strategy for selecting ONE category node at a time so as to achieve fast and stable incremental learning.

III. FALCON ARCHITECTURE

FALCON employs a three-channel architecture (Fig. 1), comprising a category field F_2 and three input fields, namely, a sensory field F_1^{c1} for representing current states, a motor field F_1^{c2} for representing actions, and a feedback field F_1^{c3} for representing reward values. The generic network dynamics of FALCON, based on fuzzy ART operations [34], is described as follows.

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

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

Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2 for learning the input patterns in F_1^{ck} for $k = 1, \ldots, 3$. Initially, F_2 contains only one *uncommitted* node. An *uncommitted* node is one which has not been used to encode any pattern and its weight vectors contain all 1s. When an *uncommitted* node is selected to learn an association, its weight vectors are modified to encode the patterns and the node becomes *committed*.

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

To emulate the activities of sense, act, and learn, FALCON network operates in one of the two modes, namely, predicting and learning. The detailed algorithm is presented in the following.

A. Predicting

In a predicting mode, FALCON receives input patterns from one or more input fields and predicts the patterns in the remaining fields. Upon input presentation, the input fields receiving values are initialized to their respective input vectors. Input fields not receiving values are initialized to $\mathbf{N}$, where $N_i = 1$ for all i.

Prediction in FALCON proceeds in three key steps, namely, code activation, code competition, and activity readout, described as follows.

Code activation: A bottom-up propagation process first takes place in which the activities (known as choice function values) of the category nodes in the F_2 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 node j, the choice function T_j is computed as follows:

$$T_j = \sum_{k=1}^{3} \gamma^{ck} \frac{\left|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}\right|}{\alpha^{ck} + \left|\mathbf{w}_j^{ck}\right|} \tag{1}$$

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

Code competition: A code competition process follows under which the F_2 node with the highest choice function value is identified. The system is said to make a choice when at most one F_2 node can become active after the code competition process. The winner is indexed at J where $T_J = \max\{T_j : \text{ for all } F_2 \text{ node } j\}$.

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

Activity readout: The chosen F_2 node J performs a readout of its weight vectors into the input fields F_1^{ck} such that

$$\mathbf{x}^{ck(\text{new})} = \mathbf{x}^{ck(\text{old})} \wedge \mathbf{w}_J^{ck}. \tag{2}$$

The resultant F_1^{ck} activity vectors are thus the fuzzy AND of their original values and their corresponding weight vectors.

B. Learning

In a learning mode, FALCON performs code activation and code competition (as described in Section III-A) to select a winner J based on the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$. To complete the learning process, template matching and template learning are performed as described in the following.

Template matching: Before node J can be used for learning, a template matching process checks that the

weight templates of node J are sufficiently close to their respective input patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen node J meets its vigilance criterion

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

Whereas the choice function computes the similarity between the input and weight vectors with respect to the norm of the weight vectors, the match function computes the similarity with respect to the norm of the input vectors. The choice and match functions work cooperatively to achieve stable coding and maximize code compression.

When resonance occurs, learning then ensues, as outlined in the following. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T_J is set to 0 for the duration of the input presentation. With a *match tracking* process in the sensory field, at the beginning of each input presentation, the vigilance parameter ρ^{c1} equals a baseline vigilance $\bar{\rho}^{c1}$. If a mismatch reset occurs in the motor and/or feedback field, ρ^{c1} is increased until it is slightly larger than the match function m_J^{c1}. The search process then selects another F_2 node J under the revised vigilance criterion until a resonance is achieved. This search and test process is guaranteed to terminate 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 firing, for each channel k, the weight vector $\mathbf{w}_J^{ck}$ is modified by the following learning rule:

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

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 and the weight vectors. For an uncommitted node J, the learning rates β^{ck} are typically set to 1. For committed nodes, β^{ck} can remain as 1 for fast learning or below 1 for slow learning in a noisy environment.

Node Creation: Our implementation of FALCON maintains ONE uncommitted node in the F_2 field at any one time. When the uncommitted node is selected for learning, it becomes *committed* and a new uncommitted node is added to the F_2 field. FALCON thus expands its network architecture dynamically in response to the incoming patterns. The FALCON network dynamics described previously can be used to support a myriad of learning operations. We present the various FALCON models, namely, R-FALCON and TD-FALCON, in Sections IV–VII.

IV. Reactive Falcon

The reactive FALCON model (R-FALCON) acquires an action policy directly by learning the mapping from the current states to the corresponding desirable actions. A summary of the R-FALCON dynamics based on the generic FALCON predicting and learning algorithms is provided in the following. Interested readers may refer to [6] for the detailed algorithm.

A. From Sensory to Action

During prediction, the activity vectors are initialized as $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0,1]$ indicates the value of sensory input i, $\mathbf{x}^{c2} = \mathbf{N} = (1, 1, \ldots, 1)$, and $\mathbf{x}^{c3} = (1, 0)$. Setting the reward vector to $(1, 0)$ favors the selection of a category node with the maximum reward value for a given state. With the activity vector values, R-FALCON performs code activation and code competition as described in Section III-A. Upon selecting a winning F_2 node, the chosen node J performs a readout of its weight vector into the motor field F_1^{c2} such that

$$\mathbf{x}^{c2(\text{new})} = \mathbf{x}^{c2(\text{old})} \wedge \mathbf{w}_J^{c2}. \quad (5)$$

R-FALCON then examines the output activities of the action vector $\mathbf{x}^{c2}$ and selects an action a_I such that $x_I^{c2} = \max\{x_i^{c2} : \text{ for all } F_1^{c2} \text{ node } i\}$.

B. From Feedback to Learning

Upon receiving a feedback from its environment after performing the action a_I, R-FALCON adjusts its internal representation using the following strategies. If a reward (positive feedback) is received, R-FALCON learns that the chosen action executed in a given state results in a favorable outcome. Therefore, R-FALCON learns to associate the state vector $\mathbf{S}$, the action vector $\mathbf{A}$, and the reward vector $\mathbf{R}$. During input presentation, $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0,1]$ indicates the value of sensory input i, $\mathbf{x}^{c2} = \mathbf{A} = (a_1, a_2, \ldots, a_m)$ where $a_i \in [0,1]$ indicates the preference of an action i, and $\mathbf{x}^{c3} = \mathbf{R} = (r, \bar{r})$ where $r \in [0,1]$ is the reward signal value and $\bar{r}$ is given by $\bar{r} = 1 - r$.

Conversely, if a penalty is received, there is a reset of action and R-FALCON learns the mapping among the state vector $\mathbf{S}$, the complement of action vector $(\bar{\mathbf{A}})$, and the complement of reward vector $(\bar{\mathbf{R}})$. During input presentation, $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ and $\mathbf{x}^{c2} = \bar{\mathbf{A}} = (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_m)$ where $\bar{a}_i = 1 - a_i$ for all i, and $\mathbf{x}^{c3} = \bar{\mathbf{R}} = (\bar{r}, r)$.

R-FALCON then proceeds to learn the association among the activity vectors of the three input fields using the learning algorithm as described in Section III-B.

V. TD-FALCON

It is significant to note that the learning algorithm of R-FALCON relies on the feedback obtained after performing each action. In a realistic environment, it may take a long sequence of actions before a reward or penalty is finally given. This is known as a temporal credit assignment problem in which we need to estimate the credit of an action based on what it will lead to eventually.

In contrast to R-FALCON that learns a function mapping states to actions directly, TD-FALCON incorporates TD methods to estimate and learn value functions, specifically, functions of state–action pairs $Q(s,a)$ that indicate the goodness for a learning system to take a certain action a in a given state s. Such value functions are used in the action selection

TABLE I
GENERIC FLOW OF THE TD-FALCON ALGORITHM

1.	Initialize the FALCON network.
2.	Given the current state s, for each available action a in the action set $\mathcal{A}$, predict the value of the action Q(s,a) by presenting the corresponding state and action vectors **S** and **A** to FALCON.
3.	Based on the value functions computed, select an action a from $\mathcal{A}$ following an action selection policy.
4.	Perform the action a, observe the next state s', and receive a reward r (if any) from the environment.
5.	Estimate the value function $Q(s,a)$ following a temporal difference formula given by $\Delta Q(s,a) = \alpha \mathrm{TD}_{err}$.
6.	Present the corresponding state, action, and reward (Q-value) vectors, namely **S**, **A**, and **R**, to FALCON for learning.
7.	Update the current state by s=s'.
8.	Repeat from Step 2 until s is a terminal state.

mechanism, *the policy*, that strives to achieve a balance between exploration and exploitation so as to maximize the total reward over time. A key advantage of using TD methods is that they can be used for multiple-step prediction problems, in which the merit of an action can only be known after several steps into the future.

The general sense–act–learn algorithm of TD-FALCON is summarized in Table I. Given the current state s, the FALCON network is used to predict the value of performing each available action a in the action set $\mathcal{A}$ based on the corresponding state vector $\mathbf{S}$ and action vector $\mathbf{A}$. The value functions are then processed by an action selection strategy (also known as policy) to select an action. Upon receiving a feedback (if any) from the environment after performing the action, a TD formula is used to compute a new estimate of the Q-value for performing the chosen action in the current state. The new Q-value is then used as the teaching signal (represented as reward vector $\mathbf{R}$) for FALCON to learn the association of the current state and the chosen action to the estimated value. The four key steps of the TD-FALCON algorithm, namely, value prediction, action selection, value estimation, and value learning, are elaborated in Sections V-A–V-D.

A. Value Prediction

Given the current state s and an available action a in the action set $\mathcal{A}$, the FALCON network is used to predict the value of performing the action a in state s based on the corresponding state vector $\mathbf{S}$ and action vector $\mathbf{A}$. Upon input presentation, the activity vectors are initialized as $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0,1]$ indicates the value of sensory input i, $\mathbf{x}^{c2} = \mathbf{A} = (a_1, a_2, \ldots, a_n)$, where $a_I = 1$ if a_I corresponds to the action a, $a_i = 0$ for $i \neq I$, and $\mathbf{x}^{c3} = (1,1)$.

With the activity vector values, FALCON performs code activation and code competition as described in Section III-A. Upon selecting a winning F_2 node, the chosen node J performs a readout of its weight vector into the reward field F_1^{c3} such that

$$\mathbf{x}^{c3(\mathrm{new})} = \mathbf{x}^{c3(\mathrm{old})} \wedge \mathbf{w}_J^{c3}. \tag{6}$$

The Q-value of performing the action a in the state s is then given by

$$Q(s,a) = \frac{x_1^{c3}}{\sum_i x_i^{c3}}. \tag{7}$$

If node J is uncommitted, $\mathbf{x}^{c3} = (1,1)$ and thus the predicted Q-value is 0.5.

B. Action Selection Policy

Action selection policy refers to the strategy for selecting an action from the set of actions available for an agent to take in a prescribed state. The simplest action selection policy is to pick the action with the highest value predicted by the FALCON network. However, a key requirement of autonomous agents is to explore the environment. If an agent keeps selecting the optimal action that it believes in, it may 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. Two policies designed to achieve a balance between exploration and exploitation are presented in the following.

1) The ϵ-greedy Policy: This policy selects the action with the highest value with a probability of $1-\epsilon$ and takes a random action with a probability of ϵ [35]. In other words, the policy will pick the action with the highest value with a total probability of $1 - \epsilon + (\epsilon/|\mathcal{A}(s)|)$ and any other action with a probability of $(\epsilon/|\mathcal{A}(s)|)$, where $\mathcal{A}(s)$ denotes the set of the available actions in a state s.

With a constant ϵ value, the agent always explores the environment with a fixed level of randomness. In practice, it may be beneficial to have a higher ϵ value to encourage exploration in the initial stage and a lower ϵ value to optimize the performance by exploiting known actions in the later stage. A decay ϵ-greedy policy is thus adopted to gradually reduce the value of ϵ over time. The rate of decay is typically inversely proportional to the complexity of the environment as a more complex environment with a larger state and action space will take a longer time to explore.

2) Softmax Policy: Under this policy, the probability $p(s,a_i)$ of choosing an action a_i in state s is given by the following:

$$p(s,a_i) = \frac{e^{Q(s,a_i)/\tau}}{\sum_{j=1}^{n} e^{Q(s,a_j)/\tau}} \tag{8}$$

where τ is a positive parameter called *temperature* and $Q(s,a_i)$ is the estimated Q-value of action a_i. At a high temperature, all actions are equally likely to be taken, whereas at a low temperature, the probability of taking a specific action is more dependent on the value estimate of the action.

C. Value Function Estimation

One key component of the TD-FALCON (Step 5) is the iterative estimation of value function $Q(s,a)$ using a TD equation

$$\Delta Q(s,a) = \alpha \mathrm{TD}_{\mathrm{err}} \tag{9}$$

where $\alpha \in [0,1]$ is the learning parameter and $\mathrm{TD}_{\mathrm{err}}$ is a function of the current Q-value predicted by FALCON and the Q-value newly computed by the TD formula. Two distinct Q-value updating rules, namely, Q-learning and SARSA, are described as follows.

1) Q-Learning: Using the Q-learning rule, the temporal error term is computed by

$$\mathrm{TD}_{\mathrm{err}} = r + \gamma \mathrm{max}_{a'} Q(s',a') - Q(s,a) \tag{10}$$

where r is the immediate reward value, $\gamma \in [0,1]$ is the discount parameter, and $\max_{a'} Q(s',a')$ is 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 FALCON network and not by a separate reinforcement learning system. The Q-learning update rule is applied to all the states that the agent traverses. With value iteration, the value function $Q(s,a)$ is expected to converge to $r + \gamma \mathrm{max}_{a'} Q(s',a')$ over time.

a) Threshold Q-learning: Whereas many reinforcement learning systems have no restriction on the value of the immediate reward r and thus the value function $Q(s,a)$, TD-FALCON and ART systems typically assume that the input values are bounded between 0 and 1. A simple solution to this problem is to apply a linear threshold function to the Q-values computed such that

$$Q(s,a) = \begin{cases} 1, & \text{if } Q(s,a) > 1 \\ 0, & \text{if } Q(s,a) < 0 \\ Q(s,a), & \text{otherwise} \end{cases}. \tag{11}$$

The threshold function, though simple, provides a reasonably good solution if the reward value r is bounded within a range, say between 0 and 1.

b) Bounded Q-learning: Instead of using the threshold function, Q-values can be normalized by incorporating appropriate scaling terms into the Q-learning updating equation directly. The bounded Q-Learning rule is given by

$$\Delta Q(s,a) = \alpha \mathrm{TD}_{\mathrm{err}} \left(1 - Q\left(s,a\right)\right). \tag{12}$$

With the scaling term, the adjustment of Q-values becomes 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. This property is formalized in the following lemma.

Lemma—Bounded Q-Learning Rule: Given that $0 \le r \le 1$, $0 \le \alpha \le (1/2)$, $\gamma \le 1$, and initially $0 \le Q(s,a) \le 1$, the bounded Q-learning rule

$$\Delta Q(s,a) = \alpha \mathrm{TD}_{\mathrm{err}} (1 - Q(s,a)) \tag{13}$$

where

$$\mathrm{TD}_{\mathrm{err}} = r + \gamma \mathrm{max}_{a'} Q(s',a') - Q(s,a) \tag{14}$$

ensures that the Q-values are bounded between 0 and 1, i.e., $0 \le Q(s,a) \le 1$, and that when learning ceases, the Q-values equal either $r + \gamma \mathrm{max}_{a'} Q(s',a')$ if $r + \gamma \mathrm{max}_{a'} Q(s',a') < 1$, or 1, otherwise.

Proof: The proof of the lemma consists of three parts as follows.

Part I) To prove that $Q(s,a) \le 1$, we show that the new Q-values computed by the updating rule will not be greater than 1

$$\begin{aligned} Q^{(\mathrm{new})}(s,a) &= Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \\ &\quad \times (1 - Q(s,a)) \\ &\le Q(s,a) + 0.5(1 + 1 \times 1 - 0)(1 - Q(s,a)) \\ &\le 1. \end{aligned}$$

Part II To prove that $Q(s,a) \ge 0$, we show that the new Q-values computed by the updating rule will not be smaller than 0

$$\begin{aligned} Q^{(\mathrm{new})}(s,a) &= Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \\ &\quad \times (1 - Q(s,a)) \\ &\ge Q(s,a) + (0 + 0 \times 0 - Q(s,a))(1 - Q(s,a)) \\ &\ge \alpha Q^2(s,a) + (1 - \alpha) Q(s,a) \\ &\ge 0. \end{aligned}$$

Part III) When learning ceases, we have $\Delta Q(s,a) = 0$. This implies that either

$$r + \gamma \mathrm{max}_{a'} Q(s',a') - Q(s,a) = 0 \tag{15}$$

or

$$1 - Q(s,a) = 0. \tag{16}$$

As Q-values are estimates of the discounted sums of future rewards in a given state, our requirement for Q-values to be bounded within the range of 0–1 imposes certain restriction on the types of problems TD-FALCON can handle directly. In cases where the discounted sums of future rewards fall significantly outside $[0,1]$, TD-FALCON may lack the sensitivity to learn the Q-values accurately.

2) SARSA: Whereas Q-learning estimates future reward as a function of the discounted maximum possible reward of taking

an action a' from the next state s', the SARSA rule simply estimates the future reward using its behavior policy with a discounted factor given by $\gamma Q(s', a')$. Using the SARSA rule, the temporal error term is computed by

$$\mathrm{TD}_{\mathrm{err}} = r + \gamma Q(s', a') - Q(s, a) \tag{17}$$

where r is the immediate reward signal, $\gamma \in [0, 1]$ is the discount parameter, and $Q(s', a')$ is the estimated value of the next state s'. Unlike Q-learning, SARSA does not have a separate estimation policy. Consequently, SARSA is said to be an on-policy as it estimates value functions based on the actions it takes. With value iteration, the value function $Q(s, a)$ is expected to converge to $r + \gamma Q(s', a')$.

As the value range of $\mathrm{TD}_{\mathrm{err}}$ for SARSA is the same as that for Q-learning, the normalization techniques derived for Q-learning (described in Section V-C1) are applicable to SARSA. Following the bounded Q-learning rule, the bounded SARSA learning rule is given by

$$\Delta Q(s, a) = \alpha \left(r + \gamma Q\left(s', a'\right) - Q\left(s, a\right)\right)\left(1 - Q\left(s, a\right)\right). \tag{18}$$

D. Value Function Learning

Upon estimating a new Q-value, FALCON learns to associate the current state s and the action a with the Q-value. During input presentation, $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0, 1]$ indicates the value of sensory input i, $\mathbf{x}^{c2} = \mathbf{A} = (a_1, a_2, \ldots, a_n)$ where $a_I = 1$ if a_I corresponds to the action a and $a_i = 0$ for $i \neq I$, and $\mathbf{x}^{c3} = (r, \bar{r})$ where $r = Q(s, a)$ and $\bar{r} = 1 - r$. FALCON then performs code activation, code competition, template matching, and template learning as described in Sections III-A and III-B to encode the association.

VI. Experimental Results

A. Minefield Navigation Task

The minefield simulation task studied in this paper is similar to the underwater navigation and mine avoidance domain developed by U.S. Naval Research Laboratory (NRL) [18]. The objective is to navigate through a minefield to a randomly selected target position in a specified time frame without hitting a mine. To tackle the minefield navigation task, Gordan and Subramanian [18] build two cognitive models, one for predicting the next sonar and bearing configuration based on the current sonar and bearing configuration and the chosen action, and the other for estimating the desirability of a given sonar and bearing configuration. Sun *et al.* [11] employ a three-layer feedforward NN trained by error BP to learn the Q-values and an additional layer to perform stochastic decision making based on the Q-values.

For experimentation, we develop a software simulator for the minefield navigation task. The simulator allows a user to specify the size of the minefield as well as the number of mines in the field. Our experiments so far have been based on a 16×16 minefield containing ten mines. In each trial, the AV starts at a randomly chosen position in the field and repeats the cycles of sense–act–learn. A trial ends when the system reaches the target (success), hits a mine (failure), or exceeds 30 sense–act–learn cycles (out of time). The target and the mines remain stationary during the trial.

TABLE II
TD-FALCON Parameters for Learning With Immediate Rewards

FALCON Parameters	
Choice parameters $(\alpha^{c1}, \alpha^{c2}, \alpha^{c3})$	$0.1, 0.1, 0.1$
Learning rates $(\beta^{c1}, \beta^{c2}, \beta^{c3})$	$1.0, 1.0, 1.0$
Contribution parameters $(\gamma^{c1}, \gamma^{c2}, \gamma^{c3})$	$0.5, 0.5, 0.0$
Baseline vigilance parameters $(\bar{\rho}^{c1}, \bar{\rho}^{c2}, \bar{\rho}^{c3})$	$0.2, 0.2, 0.5$
Temporal Difference Learning Parameters	
TD learning rate α	0.5
Discount factor γ	0.1
Initial Q-value	0.5
ϵ-greedy Action Policy Parameters	
Initial ϵ value	0.5
ϵ decay rate	0.0005

Minefield navigation and mine avoidance are nontrivial tasks. As the configuration of the minefield is generated randomly and changes over trials, the system needs to learn strategies that can be carried over across experiments. In addition, the system has a rather coarse sensory capability with a 180° forward view based on five sonar sensors. For each direction i, the sonar signal is measured by $s_i = (1/d_i)$, where d_i is the distance to an obstacle (that can be a mine or the boundary of the minefield) in the i direction. Other input attributes of the sensory (state) vector include the bearing of the target from the current position. In each step, the system can choose one of the five possible actions, namely, move left, move diagonally left, move straight ahead, move diagonally right, and move right.

B. Learning With Immediate Reinforcement

We first consider the problem of learning the minefield navigation task with immediate evaluative feedback. The reward scheme is described as follows: 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 hits a mine. At each step of the trial, an immediate reward is estimated by computing a utility function

$$\text{utility} = \frac{1}{1 + rd} \tag{19}$$

where rd is the remaining distance between the current position and the target position. When the AV runs out of time, the reward is computed using the utility function based on the remaining distance to the target.

We experiment with R-FALCON that learns the state–action policy directly and four types of TD-FALCON models, namely, Q-FALCON and BQ-FALCON based on threshold Q-learning and bounded Q-learning, respectively, as well as S-FALCON and BS-FALCON based on threshold SARSA and bounded SARSA, respectively. Each FALCON system consists of 18 nodes in the sensory fields (representing 5×2 complement-coded sonar signals and eight target bearing values), five nodes in the action field, and two nodes in the reward field (representing the complement-coded function value).

All FALCON systems use a standard set of parameter values as shown in Table II. The choice parameters are used in the choice function (1) in selecting category nodes. Using a larger choice value generally improves the predictive performance of the system but increases the number of category nodes created.

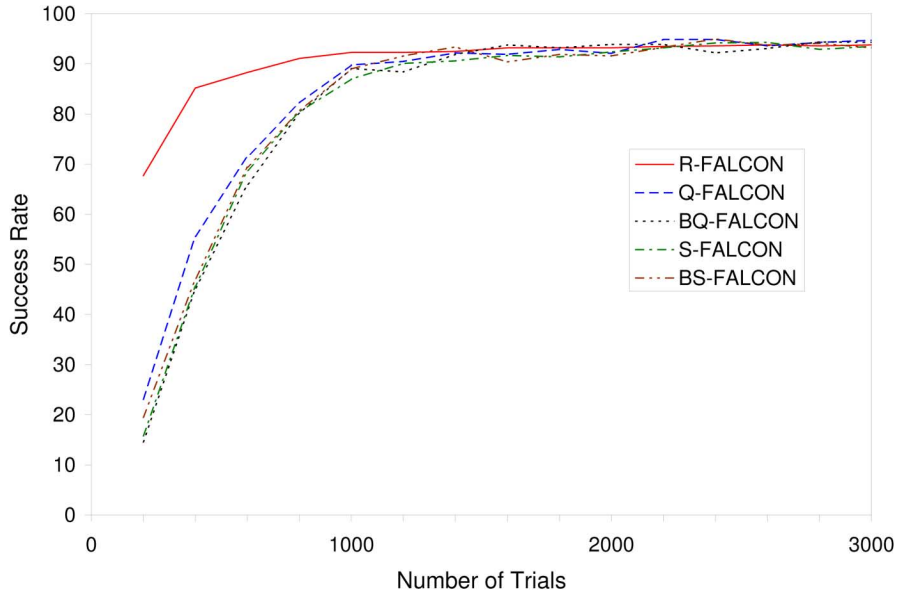


Fig. 2. Success rates of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement over 3000 trials across ten experiments.

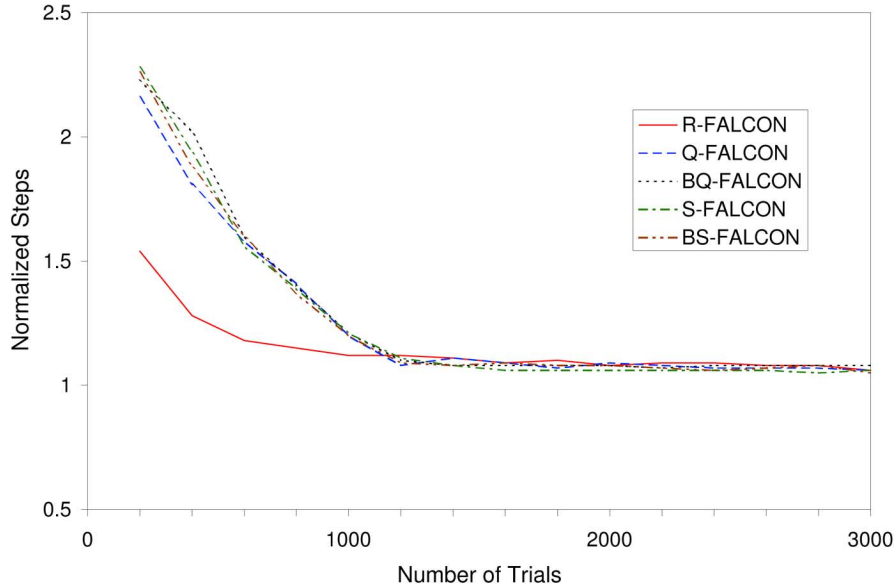


Fig. 3. Average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement to reach the target over 3000 trials across ten experiments.

The learning rate parameters β^{ck} for $k = 1, 2, 3$ are set to 1.0 for fast learning. Decreasing the learning rates slows down the learning process, but may produce a smaller set of better quality category nodes and thus lead to a slightly better predictive performance. The contribution parameters γ^{c1} and γ^{c2} are set to 0.5 as TD-FALCON selects a category node based on the input activities in the state and action fields. The baseline vigilance parameters $\bar{\rho}^{c1}$ and $\bar{\rho}^{c2}$ are set to 0.2 for a marginal level of match criterion on the state and action spaces so as to encourage generalization. The vigilance of the reward field $\bar{\rho}^{c3}$ is fixed at 0.5 for a stricter match criterion. Increasing the vigilance values generally increases the predictive performance with the cost of creating more category nodes. For the TD learning rules, the learning rate α is fixed at 0.5 to allow a modest pace of learning while retaining stability. The discount factor γ is set to 0.1 to favor the direct reward signals available. The initial Q-value, used when TD-FALCON selects an uncommitted node during prediction, is set to 0.5, corresponding to a weight vector of (1,1). For action selection policy, the decay ϵ-greedy policy is used with ϵ initialized to 0.5 and decayed at a rate of 0.0005 per trial, until ϵ drops to 0.005. This implies that the system will have a low chance to explore new moves after around 1000 trials.

Fig. 2 summarizes the performance of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON in terms of success rates averaged at 200-trial intervals over 3000 trials across ten sets of experiments. We can see that the success rates of all systems increase steadily right from the beginning.

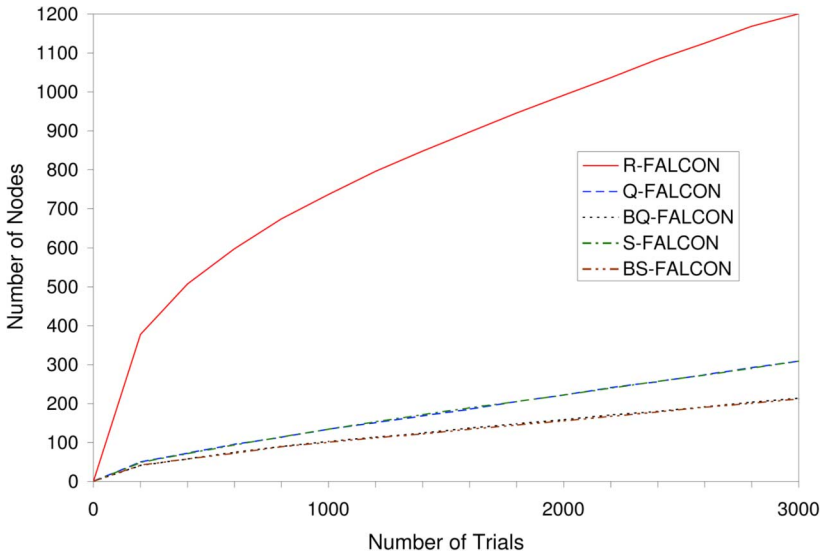


Fig. 4. Average numbers of category nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement over 3000 trials across ten experiments.

Among all, R-FALCON is the fastest, achieving 90% at 600 trials. Nevertheless, beyond 1000 trials, all TD-FALCON systems can achieve over 90% success rates. In the long run, R-FALCON and all four TD-FALCON systems achieve roughly the same level of performance.

To evaluate in quantitative terms how well a system traverses from a starting position to the target, we define a measure called *normalized step* given by $\text{step}_n = (\text{step}/sd)$, where "step" is the number of sense–act–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 that the system has taken the optimal path to the target.

Fig. 3 depicts the average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON to reach the target over 3000 trials across the ten sets of experiments. We see that all systems are able to reach the targets via near-optimal paths after 1200 trials, although R-FALCON achieves that in 600 trials. In the long run, all systems produce a stable performance in terms of the quality of the paths taken.

Fig. 4 depicts the average numbers of category nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FLACON, and BS-FALCON over 3000 trials across the ten sets of experiments. Among the five systems, R-FALCON creates the most number of codes, significantly more than those created by the TD-FALCON systems. While we observe no significant performance difference among the four TD-FALCON systems in other aspects, BQ-FALCON and BS-FALCON demonstrate the advantage of the bounded learning rule by producing a more compact set of category nodes than Q-FALCON and S-FALCON.

C. Learning With Delayed Reinforcement

In this set of experiments, the AV does not receive immediate evaluative feedback for each action it performs. This is a more realistic scenario, because in the real world, the targets may be blocked or invisible. The reward scheme is described as follows: A reward of 1 is given when the AV reaches the target. A reward of 0 is given when the AV hits a mine. Different from the previous experiments with immediate rewards, a reward of 0 is given when the system runs out of time. In accordance with the bounded Q-learning lemma, negative reinforcement values are not used in our reward scheme to ensure the Q-values are always bounded within the desired range of 0–1.

All systems use the same set of parameter values as shown in Table II, except that the TD discount factor γ is set to 0.9 due to the absence of immediate reward signals. Fig. 5 summaries the performance of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON in terms of success rates averaged at 200-trial intervals over 3000 trials across ten sets of experiments. We see that R-FALCON produces a miserable near-zero success rate throughout the trials. This is not surprising as it only undergoes learning when it hits the target or a mine. The TD-FALCON systems, on the other hand, maintain the same level of learning efficiency as those obtained in the experiments with immediate reinforcement. At the end of 1000 trials, all four TD-FALCON systems can achieve success rates of more than 90%. In the long run, there is no significant difference in the success rates of the four systems.

Fig. 6 shows the average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON to reach the targets over 3000 trials across ten experiments. Without immediate rewards, R-FALCON as expected performs very poorly. All four TD-FALCON systems, on the other hand, maintain the quality by always taking near-optimal paths after 1000 trials.

Fig. 7 shows the numbers of category nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON over the 3000 trials. Without immediate reward, the quality of the estimated value functions declines. As a result, all systems create a significantly larger number of category nodes comparing with those created in the experiments with immediate reinforcement. Nevertheless, TD-FALCON systems with bounded learning rule (i.e., BQ-FALCON and

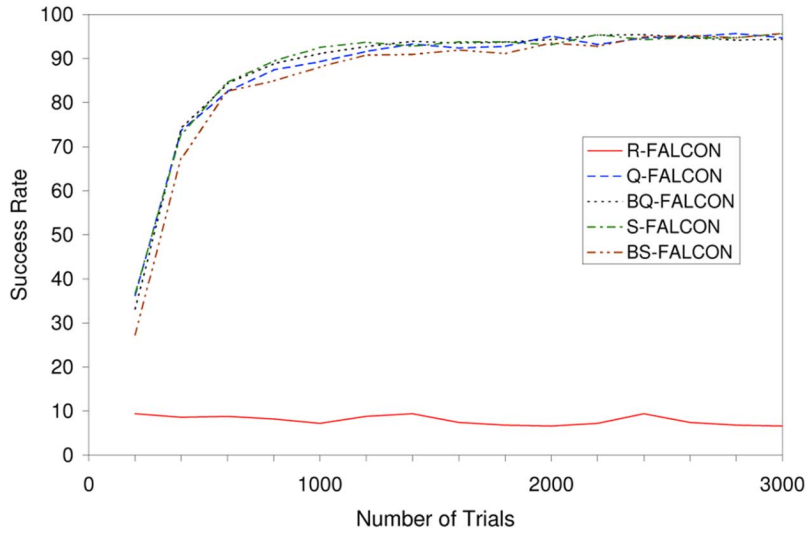


Fig. 5. Success rates of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement over 3000 trials across ten experiments.

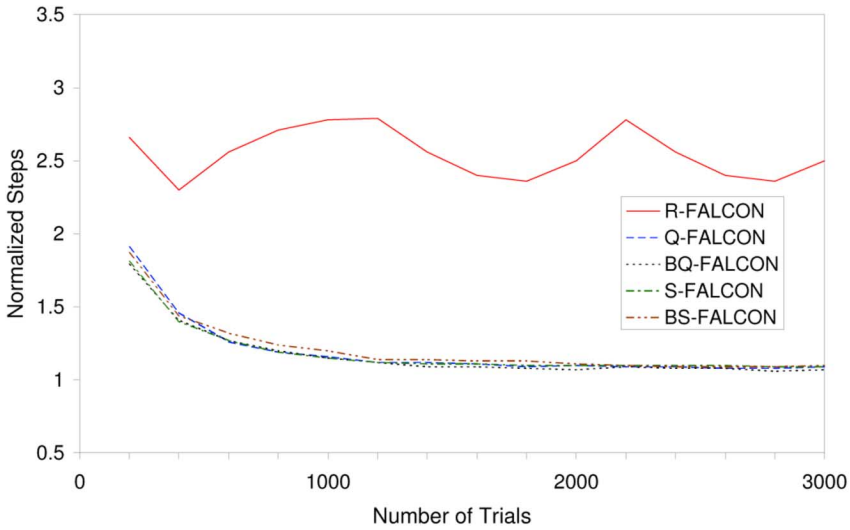


Fig. 6. Average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement to reach the target over 3000 trials across ten experiments.

BS-FALCON) as before cope better with a smaller number of nodes.

D. Comparing With Gradient–Descent-Based Q-Learning

To put the performance of TD-FALCON in perspective, we further conduct experiments to evaluate the performance of a reinforcement learning system (hereafter referred to as BP-Q learner), using the standard Q-learning rule and a gradient–descent-based multilayer feedforward NN as the function approximator. Although we start off by incorporating TD learning into the original (reactive) FALCON system for the purpose of handling delayed rewards, FALCON effectively serves as a function approximator for learning the Q-value function. It thus makes sense to compare FALCON with another function approximator in the same context of Q-learning. Among the various universal function approximation techniques, we have chosen the gradient-descent BP algorithm as the reference point for comparison as it is by far one of the most widely used and has been applied in many different systems, including Q-learning [10], [11] as well as ACD [12], [13], [26]. The specific configuration of combining Q-learning and multilayer feedforward NN with error BP has been used by Sun *et al.* [11] in a similar underwater minefield navigation domain.

The BP-Q learner employs a standard three-layer (consisting of one input layer, one hidden layer, and one output layer) feedforward architecture to learn the value function. The input layer consists of 18 nodes representing the five sonar signal values, eight possible target bearings, and five selectable actions. The input attributes are exactly the same as those used in the TD-FALCON, except that the sonar signals are not complement coded. The output layer consists of only one node representing the value of performing an action in a particular

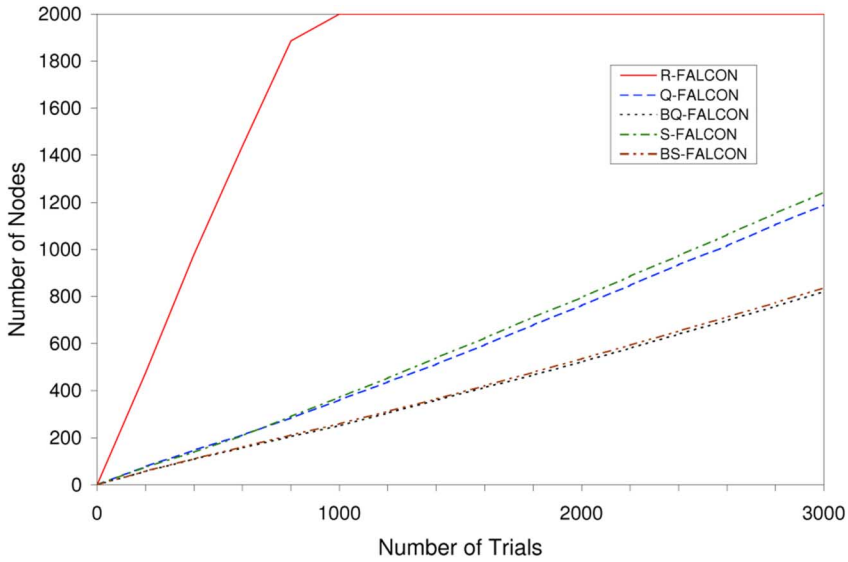


Fig. 7. Average numbers of category nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement over 3000 trials across ten experiments.

state. All hidden and output nodes employ a symmetrical sigmoid function. For a fair comparison, the BP-Q learner also makes use of the same decay ϵ-greedy action selection policy.

Using a learning rate of 0.25 and a momentum term of 0.5 for the hidden and output layers, we first experiment with a varying number of hidden nodes and obtain the best results with 36 nodes. Using a smaller number of, say 24, nodes produces a slightly lower success rate with a larger variance in performance. Increasing the number of nodes to 48 leads to a poorer result as well. We then experiment with different learning rates, from 0.1 to 0.3, for the hidden and output layers and obtain the best results with learning rates of 0.3 for the two layers. Increasing the learning rates to 0.4 and 0.5 produces slightly inferior results. We further experiment with different decay schedules for the ϵ-greedy action policy. We find that BP-Q requires a much longer exploration phase with an ϵ decay rate of 0.00001. Attempts with a higher ϵ decay rate meet with significantly poorer results. The best results obtained by the BP-Q learner across ten sets of experiments in terms of success rates are reported in Fig. 8. The performance figures, obtained with initial random weight values between -0.5 and 0.5, are significantly better than our previous results obtained using initial weight values between -0.25 and 0.25.

Although there has been no guarantee of convergence by using a function approximator, such as MLP with error BP, for Q-learning [7], the performance and the stability of BQ-P are actually quite good. For both experiments involving immediate and delayed rewards, the BP-Q learner can achieve very high success rates consistently, although it generally takes a large number of trials (around 40 000 trials) to cross the 90% mark. In contrast, TD-FALCON achieves the same level of performance (90%) within the first 1000 trials. This indicates that TD-FALCON is around 40 times (more than an order of magnitude) faster than the BP-Q learner in terms of learning efficiency.

Considering network complexity, the BP-Q learner has the advantage of a highly compact network architecture. When trained properly, a BP network consisting of 36 hidden nodes can produce performance equivalent to that of a TD-FALCON model with around 200 category nodes. In terms of adaptation speed, however, TD-FALCON is clearly a faster learner by consistently mastering the task in a much smaller number of trials.

E. Comparing With Direct NDP

We have also attempted an ACD model [26], specifically direct NDP [12], belonging to the class of ADHDP, on the minefield navigation problem. Direct NDP consists of a critic and action networks, wherein the output of the action network feeds directly into the input layer of the critic network. Our Java implementation of the direct NDP is modified from the Matlab code. As in typical action-dependent (AD) versions of ACD, training of the critic network is based on optimizing a cost or reward-to-go function by balancing the Bellman's equation [36], whereas training of the action network relies on the error signals backpropagated from the critic network.

We first experiment with the original direct NDP and find several extensions needed for the minefield problem. The key changes include modifying the output layer of the action network and the input layer of the critic network from a single action node to multiple action nodes (one for each of the five movement directions) and restricting the choice of actions to those valid ones only. We also make use of the next total discounted reward-to-go $(J(t+1))$ in calculating the error term of the critic network [26] instead of the previous total discounted reward-to-go $(J(t-1))$ as used in the original direct NDP code. This modification is necessary as the minefield navigation task does not run indefinitely (as in tasks such as pole balancing) and using the next J enables us to "ground" the J values at the terminal states. Specifically, when an action of the AV leads to the target, we assign 1 to $J(t+1)$ instead of using the critic network

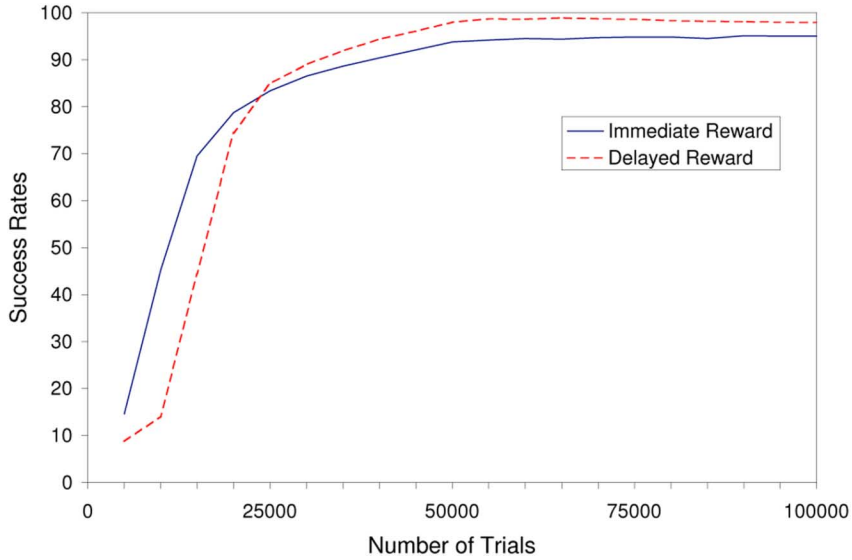


Fig. 8. Success rates of BP-Q learning over 100 000 trials across ten experiments.

TABLE III
TIME COMPLEXITIES OF R-FALCON, TD-FALCON, BP-Q, AND DIRECT NDP PER SENSE–ACT–LEARN CYCLE S AND A DENOTE THE DIMENSIONS OF THE SENSORY AND ACTION FIELDS, RESPECTIVELY. N INDICATES THE NUMBER OF CATEGORY NODES FOR TD-FALCON AND THE NUMBER OF HIDDEN NODES IN THE CONTEXT OF BP-Q AND DIRECT NDP

	R-FALCON	Q-FALCON BQ-FALCON	S-FALCON BS-FALCON	BP-Q	direct NDP
Predicting	$O((S+A)N)$	$O((S+A)NA)$	$O((S+A)NA)$	$O((S+A)NA)$	$O((S+A)N)$
Learning	$O((S+A)N)$	$O((S+A)NA)$	$O((S+A)N)$	$O((S+A)NA)$	$O((S+A)N)$

to compute $J(t+1)$. Similarly, when an action results in hitting a mine, we assign 0 to $J(t+1)$. We also experiment with other enhancement, such as incorporating bias nodes in the input and hidden layers of the action and critic networks, and adding in an exploration mechanism as used by Q-learning, but find that they are not necessary in the context of direct NDP.

Our experiments of direct NDP so far do not always result in convergence. Whereas training the critic network is generally problem-free, convergence of the action network is much more challenging. Despite experimenting with various learning and decay rates, the output (action vector) values of the AN could still become saturated (at 1 or -1) and this prevents further reduction of the action network's error function value. In some experiments, direct NDP does converge successfully. In a typical successful run, direct NDP is able to cross 90% success rate in 30 000 trials and achieve around 95% after 50 000 trials. Although the stability and performance of direct NDP should improve as we gain more experience of the system, we reckon it is unlikely to match the learning speed displayed by TD-FALCON in the minefield domain.

VII. COMPLEXITY ANALYSIS

A. Space Complexity

The space complexity of FALCON is determined by the number of weight values or conditional links in the FALCON network. Specifically, the space complexity is given by $O((S+A+R)N)$, where S, A, and R are the dimensions of the sensory, action, and reward fields, respectively, and N is the number of category nodes in the category field. With a fixed number of hidden nodes, the space complexity of the BP-Q learner as well as that of direct NDP is in the order of $O(S+A+R)$. BP-Q and direct NDP are thus typically more compact than a FALCON network.

Without function approximation, a table lookup reinforcement learning system would associate a value for each state or for each state–action pair. The space complexity for learning state–action mapping is thus $O(D^S)$, where S is the number of the sensory inputs and D is the largest number of discretized values across the S attributes. On the other hand, the space complexity for learning the state–action-value mapping is $O(D^S A)$, where A is the number of available actions. It can be seen that whereas the space complexities of TD-FALCON, BP-Q, and direct NDP are in the order of polynomial, the space complexity of a traditional table lookup system is exponential.

B. Time Complexity

Table III summarizes the computational complexity of various FALCON systems compared with BP-Q and direct NDP, in terms of action selection and learning. For simplicity, we have omitted the dimension of reward field (R), which is fixed at 2. As TD-FALCON and BP-Q both compute the Q-values of all possible actions before selecting one, they have a higher time complexity than R-FALCON and direct NDP, which select an action based on the current state input directly. In terms of learning, Q-FALCON, BQ-FALCON, and BP-Q are more time consuming as they need to evaluate the maximum Q-value of the next state. As TD-FALCON creates category nodes dynamically whereas BP-Q and direct NDP use a fixed number of

TABLE IV
COMPUTING TIME TAKEN BY R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, AND BS-FALCON FOR LEARNING MINEFIELD NAVIGATION WITH IMMEDIATE REINFORCEMENT

Per experiment (3000 trials)	R-FALCON	Q-FALCON	BQ-FALCON	S-FALCON	BS-FALCON
Computing Time (seconds)	47.5	80.1	65.2	86.0	62.6
Average no. of Steps	25209	29224	30562	29416	28491
Computing Time/Step (ms)	1.9	2.7	2.1	2.9	2.2

TABLE V
COMPUTING TIME TAKEN BY R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, AND BS-FALCON FOR LEARNING MINEFIELD NAVIGATION WITH DELAYED REINFORCEMENT

Per experiment (3000 trials)	R-FALCON	Q-FALCON	BQ-FALCON	S-FALCON	BS-FALCON
Computing Time (seconds)	1741.6	241.4	166.2	255.5	172.9
Average no. of Steps	57892	29766	28646	27256	27969
Computing Time/Step (ms)	30.0	8.1	5.8	9.4	6.2

hidden nodes, the latter two are deemed to have a lower time complexity. Based on the time complexity analysis, we conclude that the time complexity of direct NDP per reaction cycle is the lowest, followed by R-FALCON and BP-Q. Among the various TD-FALCON systems, the time complexities are basically equivalent with a small action set. The overall relations can be summarized as

$$\begin{aligned} \mathrm{T}(\text{direct NDP}) &< \mathrm{T}(\text{R-FALCON}) \\ &< \mathrm{T}(\text{BP-Q}) \\ &< \mathrm{T}(\text{S-FALCON}) \\ &\equiv \mathrm{T}(\text{BS-FALCON}) \\ &\equiv \mathrm{T}(\text{Q-FALCON}) \\ &\equiv \mathrm{T}(\text{BQ-FALCON}) \end{aligned}$$

where $T(\cdot)$ refers to the time complexity of the individual system, "$<$" means "is lower than" and "$\equiv$" means "is equivalent to."

C. Run Time Comparison

Tables IV and V show the computation time taken by the various systems per step (i.e., sense–act–learn cycle) in the minefield experiments with immediate and delayed reinforcement, respectively. The figures are based on our experiments conducted on a notebook computer using a 1.6-GHz Pentium M processor with 512-MB memory. For experiments with immediate reinforcement, R-FALCON is the fastest by learning the action policy directly. BQ-FALCON and BS-FALCON are slower than R-FALCON, but are faster than S-FALCON and Q-FALCON. For experiments with delayed reinforcement, BQ-FALCON and BS-FALCON are also faster than Q-FALCON and S-FALCON. As the time complexities of the four TD-FALCON systems are in the same order of the magnitude, the variations in reaction time among the four TD-FALCON systems are largely due to the different numbers of category nodes created by the various systems over the 3000 trials. On the whole, the reaction time per step for all systems are in the range of a few milliseconds. This shows that TD-FALCON systems are able to learn and function in real time with both immediate and delayed reinforcement.

TABLE VI
COMPUTING TIMES TAKEN BY BP-Q AND DIRECT NDP FOR LEARNING MINEFIELD NAVIGATION. THERE IS NO NOTICEABLE DIFFERENCE BETWEEN EXPERIMENTS WITH IMMEDIATE AND DELAYED REINFORCEMENT

Per experiment (100,000 Trials)	BP-Q	dNDP
Computing Time (seconds)	253.7	1351.1
Average no. of Steps	975882	1073184
Computing Time/Step (ms)	0.3	1.3

Referring to Table VI, the computing time of BP-Q and direct NDP presents an interesting picture. BP-Q and direct NDP tend to be more computationally expensive in the initial learning stage. However, once the networks are fully trained, a minimal amount of time is spent in learning and the reaction time per cycle is extremely short. Averaged over 100 000 trials, the reaction times of BP-Q and direct NDP are 0.3 millisecond and 1.3 ms, respectively, even lower than those of TD-FALCON systems. However, both BP-Q and direct NDP require a much larger number of trials to achieve the same level of performance as TD-FALCON. The computing time required on the whole is in fact longer.

VIII. CONCLUSION

We have presented a fusion architecture, known as TD-FALCON, for learning multimodal mappings across states, actions, and rewards. The proposed model provides a basic building block for developing autonomous agents capable of functioning and adapting in a dynamic environment with both immediate and delayed reinforcement signals. Among all, BQ-FALCON and BS-FALCON are the best performers in terms of task completion, learning speed, and efficiency.

Whereas Q-learning implemented with table lookup has been proven to converge under specific conditions [8], the proof of convergence for TD learning with the use of function approximators, in general, is still an open problem. Nevertheless, ART-based systems appear to provide a better incremental learning and convergence behavior compared with standard gradient–descent-based methods in our past and present experiments.

The minefield navigation task has supported the validity of our approach and algorithms. However, the problem is relatively small in scale. Our future work will involve applying TD-FALCON to more complex and challenging domains and

comparing with key alternative systems. As TD-FALCON assumes that the input values are bounded between 0 and 1, our requirement for Q-values to be bounded thus imposes some constraints on the choice of reward function (r) and the TD parameter values (α and γ). These, in turn, may restrict the types of problems TD-FALCON can handle directly. In addition, our study so far has assumed the use of a discrete action set. For tasks that involve actions with continuous values, we would need to extend the learning algorithms to handle both continuous state and action spaces.

Our experiments have also shown that TD-FALCON may create too many category nodes during learning resulting in a drop in efficiency. As such, we will explore algorithms for generating a more compact TD-FALCON network structure. Another solution is to incorporate a real-time node evaluation and pruning mechanism [6], [37] as part of the TD-FALCON learning dynamics in order to reduce network complexity and improve computational efficiency.

While the comparisons between TD-FALCON and the standard gradient–descent-based methods have shown an advantage of TD-FALCON, additional comparisons remain to be performed with more sophisticated gradient–descent approaches, such as least squares policy iteration (LSPI) [38], and dynamic resource allocating methods, such as ones based on Platt's resource-allocating network (RAN) [27]. Considering that TD-FALCON employs an augmented learning network embedding the Q-learning algorithm, it will also be interesting to see if other reinforcement learning methods, such as NDP, can be integrated into the FALCON network to produce a more robust and efficient learning system.

ACKNOWLEDGMENT

The authors would like to thank the three anonymous reviewers for providing many valuable comments and suggestions to the various versions of this paper. They would like to thank J. Si for the discussion on applying direct NDP to the minefield navigation problem, J. Jin for contributing to the development of the minefield navigation simulator, and C. A. Bastion for help in editing this manuscript.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] M. L. Anderson, "Embodied cognition: A field guide," *Artif. Intell.*, vol. 149, pp. 91–130, 2003.

[3] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vis. Graph. Image Process.*, vol. 37, pp. 54–115, Jun. 1987.

[4] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 698–713, Sep. 1992.

[5] A. H. Tan, "Adaptive resonance associative map," *Neural Netw.*, vol. 8, no. 3, pp. 437–446, 1995.

[6] A. H. Tan, "FALCON: A fusion architecture for learning, cognition, and navigation," in *Proc. Int. Joint Conf. Neural Netw.*, 2004, pp. 3297–3302.

[7] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Dept. Comput. Sci., King's College, Cambridge, U.K., 1989.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.

[9] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge Univ., Cambridge, U.K., Tech. Rep. CUED/F-INFENG/TR166, 1994.

[10] L. J. Lin, "Programming robots using reinforcement learning and teaching," in *Proc. 9th Nat. Conf. Artif. Intell.*, 1991, pp. 781–786.

[11] R. Sun, E. Merrill, and T. Peterson, "From implicit skills to explicit knowledge: A bottom-up model of skill learning," *Cogn. Sci.*, vol. 25, no. 2, pp. 203–244, 2001.

[12] J. Si, L. Yang, and D. Liu, "Direct neural dynamic programming," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, 2004, pp. 125–151.

[13] P. Werbos, "ADP: Goals, opportunities and principles," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, 2004, pp. 3–44.

[14] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds., *Handbook of Learning and Approximate Dynamic Programming*. New York: Wiley-IEEE Press, 2004.

[15] D. H. Ackley and M. L. Littman, "Generalization and scaling in reinforcement learning," in *Advances in Neural Information Processing Systems 2*. Cambridge, MA: MIT Press, 1990, pp. 550–557.

[16] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Massachusetts, Amherst, MA, 1984.

[17] M. Wu, J. Lin Z.-H, and P.-H. Hsu, "Function approximation using generalized adalines," *IEEE Trans. Neural Netw.*, vol. 17, no. 3, pp. 541–558, May 2006.

[18] D. Gordan and D. Subramanian, "A cognitive model of learning to navigate," in *Proc. 19th Annu. Conf. Cogn. Sci. Soc.*, 1997, pp. 271–276.

[19] T. T. Shannon and G. Lendaris, "Adaptive critic based design of a fuzzy motor speed controller," in *Proc. Int. Symp. Intell. Control (ISIC)*, Mexico City, 2001, pp. 359–363.

[20] J. C. Santamaria, R. S. Sutton, and A. Ram, "Experiments with reinforcement leaning in problems with continuous state and action spaces," *Adapt. Behavior*, vol. 6, pp. 163–217, 1997.

[21] D. P. Bertsekas and J. N. Tsitsiklis, *Neural Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[22] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Real-time learning capability of neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 863–878, Jul. 2006.

[23] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul. 2006.

[24] G. A. Rummery, "Problem solving with reinforcement learning," Ph.D. dissertation, Eng. Dept., Cambridge Univ., Cambridge, U.K., 1995.

[25] G. J. Tesauro, "TD-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, no. 2, pp. 215–219, 1994.

[26] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997.

[27] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.

[28] C. W. Anderson, "Q-learning with hidden-unit restarting," in *Advances in Neural Information Processing Systems 5*. Cambridge, MA: MIT Press, 1993, pp. 81–88.

[29] S. Iida, K. Kuwayama, M. Kanoh, S. Kato, and H. Itoh, "A dynamic allocation method of basic functions in reinforcement learning," in *Lecture Notes in Computer Science*, ser. 3339. Berlin, Germany: Springer-Verlag, 2004, pp. 272–283.

[30] A. J. Smith, "Applications of the self-organizing map to reinforcement learning," *Neural Netw.*, vol. 15, no. 8-9, pp. 1107–1124, 2002.

[31] J. Provost, B. J. Kuipers, and R. Miikkulainen, "Self-organizing perceptual and temporal abstraction for robotic reinforcement learning," presented at the AAAI Workshop Learn. Plan. Markov Processes, 2004.

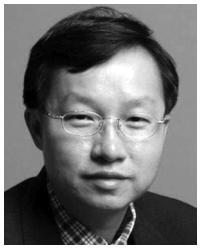
[32] H. Ueda, N. Hanada, H. Kimoto, and T. Naraki, "Fuzzy Q-learning with the modified fuzzy ART neural network," in *Proc. IEEE/WIC/ACM Int. Conf. Intell. Agent Technol.*, 2005, pp. 308–315.

[33] S. Ninomiya, "A hybrid learning approach integrating adaptive resonance theory and reinforcement learning for computer generated agents," Ph.D. dissertation, Dept. Inf. Systems, Univ. Central Florida, Orlando, FL, 2002.

[34] 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 Netw.*, vol. 4, pp. 759–771, 1991.

[35] A. Pérez-Uribe, "Structure-adaptable digital neural networks," Ph.D. dissertation, Comp. Sci. Dept., Swiss Fed. Inst. Technol., Lausanne, Switzerland, 2002.

[36] R. Bellman, Ed., *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[37] G. A. Carpenter and A. H. Tan, "Rule extraction: From neural architecture to symbolic representation," *Connection Sci.*, vol. 7, no. 1, pp. 3–27, 1995.
[38] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, 2003.



Ah-Hwee Tan (SM'04) received the B.Sc. (first class honors) and M.Sc. degrees in computer science from the National University of Singapore, Singapore, in 1989 and 1991, respectively, and the Ph.D. degree in cognitive and neural systems from Boston University, Boston, MA, in 1994.

Currently, he is an Associate Professor and the Director of Emerging Research Laboratory, School of Computer Engineering, Nanyang Technological University, Singapore. He is also a Faculty Associate of A*STAR Institute for Infocomm Research, where he was formally the Manager of the Text Mining and Intelligent Cyber Agents groups. He holds several patents and has successfully commercialized a suite of document analysis and text mining technologies. His current research areas include cognitive and neural systems, intelligent agents, machine learning, media fusion, and information mining.

Dr. Tan is a member of Association for Computing Machinery (ACM) and an editorial board member of *Applied Intelligence*.

Ning Lu received the B.Eng. degree from the School of Computer Engineering, Nanyang Technological University, Singapore. He contributed to the reported work while he was doing his final year project.

Dan Xiao received the B.S. degree from the Department of Computer Science, Beijing University, Beijing, China, in 1992 and the M.S. degree in applied science from the School of Applied Science at Nanyang Technological University, Singapore, in 2000, where currently, he is working towards the Ph.D. degree at the School of Computer Engineering.

His research areas include cluster-based systems and multiagent learning.