Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning

Ah-Hwee Tan
Nanyang Technological University
School of Computer Engineering and Intelligent Systems Centre
Nanyang Avenue, Singapore 639798
asahtan@ntu.edu.sg

Abstract

TD-FALCON is a self-organizing neural network that incorporates Temporal Difference (TD) methods for reinforcement learning. Despite the advantages of fast and stable learning, TD-FALCON still relies on an iterative process to evaluate each available action in a decision cycle. To remove this deficiency, this paper presents a direct code access procedure whereby TD-FALCON conducts instantaneous searches for cognitive nodes that match with the current states and at the same time provide maximal reward values. Our comparative experiments show that TD-FALCON with direct code access produces comparable performance with the original TD-FALCON while improving significantly in computation efficiency and network complexity.

1 Introduction

Reinforcement learning [Sutton and Barto, 1998] is an interaction based paradigm wherein an autonomous agent learns to adjust its behaviour according to feedback from the environment. Classical solutions to the reinforcement learning problem generally involve learning one or more of the following mappings, the first linking a given state to a desired action (action policy), and the second associating a pair of state and action to a utility value (value function), using temporal difference methods, such as SARSA [Rummery and Niranjan, 1994] and Q-learning [Watkins and Dayan, 1992]. The problem of the original formulation 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.

Neural networks and reinforcement learning have had an intertwining relationship [Kaelbling *et al.*, 1996]. In particular, multi-layer feed-forward neural networks, also known as multi-layer perceptron (MLP), have been used extensively in many reinforcement learning system and applications [Ackley and Littman, 1990; Sutton, 1984]. Under the recent thread of research in Approximate Dynamic Programming (ADP) [Si *et al.*, 2004], MLP and gradient descent backpropagation (BP) learning algorithms are commonly used to learn an approximation of the value function from the state and action spaces (value policy) and/or an approximation of the action function from the state space (action policy). MLP and BP however are not designed for online incremental learning as they typically require an iterative learning process. In addition, there is an issue of instability in the sense that learning of new patterns may erode the previously learned knowledge. Consequently, the resultant reinforcement learning systems may not be able to learn and operate in real time.

Instead of learning value functions and action policies, self-organizing neural networks, such as Self-Organizing Map (SOM), are typically used for the representation and generalization of continuous state and action spaces [Smith, 2002]. The state and action clusters are then used as the entries in a Q-value table implemented separately. Using a localized representation, SOM has the advantage of more stable learning, compared with backpropagation networks. However, SOM remains an iterative learning system, requiring many rounds to learn the compressed representation of the state and action patterns. In addition, as mentioned in Smith (2002), SOM is expected to scale badly if the dimensions of the state and actions spaces are significantly higher than the dimension of the map. As such, most applications of SOM are limited to low dimensional state and action spaces.

A recent approach to reinforcement learning builds upon Adaptive Resonance Theory (ART) [Carpenter and Grossberg, 1987], also a class of self-organizing neural networks, but with very distinct characteristics from SOM. Ueda *et. al* (2005) use ART models to learn the clusters of state and action patterns. The clusters are in turns used as the compressed states and actions by a Q-learning module. Ninomiya (2002) couples a supervised ART system with a Temporal Difference reinforcement learning module in a hybrid architecture. Whereas the states and actions in the reinforcement module are exported from the supervised ART system, the two learning systems operate independently. The redundancy in representation unfortunately leads to instability and unnecessarily long processing time in action selection as well as learning of value functions. Through a generalization of ART from one input pattern field to multiple pattern channels, Tan (2004) presents a neural architecture called FALCON (Fusion Architecture for Learning, COgnition, and Navigation), that learns multi-channel mappings simultaneously across multi-modal input patterns, involving states, actions, and rewards, in an online and incremental manner. For handling problems with delayed evaluative feedback (reward signal), a variant

of FALCON, known as TD-FALCON [Tan and Xiao, 2005], learns the value functions of the state-action space estimated through Temporal Difference (TD) algorithms. Compared with other ART-based systems described by Ueda *et. al* and Ninomiya, TD-FALCON presents a truly integrated solution in the sense that there is no implementation of a separate reinforcement learning module or Q-value table.

Although TD-FALCON provides a promising approach, its action selection procedure contains an inherent limitation. Specifically, TD-FALCON selects an action by weighting the consequence of performing each and every possible action in a given state. Besides that the action selection process is inefficient with a large number of actions, the numerating step also assumes a finite set of actions, rendering it inapplicable to continuous action space. In view of this deficiency, this paper presents a direct code access procedure by which TD-FALCON can perform instantaneous searches for cognitive nodes that match with the current states and at the same time provide the highest reward values. Besides that the algorithm is much more natural and efficient, TD-FALCON can now operate with both continuous state and action spaces. Our comparative experiments based on a minefield navigation task show that TD-FALCON with direct code access produces comparable performance with the original TD-FALCON system while improving vastly in terms of computation efficiency as well as network complexity.

The rest of the paper is organized as follows. For completeness, section 2 presents a summary of the FALCON architecture and the associated learning and prediction algorithms. Section 3 presents the new TD-FALCON algorithm with the direct code access procedure. Section 4 introduces the minefield navigation simulation task and presents the experimental results. The final section concludes and provides a brief discussion of future work.

2 FALCON Dynamics

FALCON employs a 3-channel architecture (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 [Carpenter *et al.*, 1991], is described below.

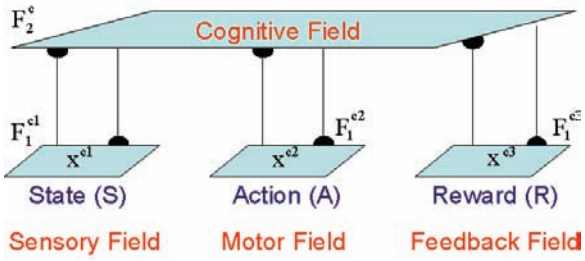

Figure 1: The FALCON architecture.

Input vectors: Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where $s_i \in [0, 1]$ indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where $a_i \in [0, 1]$ indicates 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 range of $[0, 1]$.

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

Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2^c for learning the input patterns in F_1^{ck} for $k = 1, \ldots, 3$. 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: The FALCON's dynamics is determined by choice parameters $\alpha^{ck} > 0$ for $k = 1, \ldots, 3$; learning rate parameters $\beta^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$; contribution parameters $\gamma^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$ where $\sum_{k=1}^{3} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$.

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^{cs} and F_1^{c3} respectively), for each F_2^c node j, the choice function T_j^c is computed as follows:

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

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$. In essence, the choice function T_j 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: A code competition process follows under which the F_2^c node with the highest choice function value is identified. The winner is indexed at J where

$$T_J^c = \max\{T_j^c : \text{for all } F_2^c \text{ node } j\}. \tag{2}$$

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, a template matching process checks that the weight templates of code J are sufficiently close to their respective activity patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen code J meets its vigilance criterion:

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \tag{3}$$

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

When resonance occurs, learning ensues, as defined below. 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 ρ^{c1} equals a baseline vigilance $\bar{\rho}^{c1}$. If a mismatch reset occurs, ρ^{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 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. When an uncommitted node is selecting 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.

3 TD-FALCON

TD-FALCON incorporates 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 select an action with the maximal payoff. The original TD-FALCON algorithm proposed by Tan and Xiao (2005) selects an action with the maximal Q-value in a state s by enumerating and evaluating each available action a by presenting the corresponding state and action vectors $\mathbf{S}$ and $\mathbf{A}$ to FALCON. The TD-FALCON presented in this paper replaces the action enumeration step with a direct code access procedure, as shown in Table 1. Given the current state s, TD-FALCON first decides between exploration and exploitation by following an action selection policy. For exploration, a random action is picked. For exploitation, TD-FALCON searches for optimal action through a direct code access procedure. Upon receiving a feedback from the environment after performing the action, a TD 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. The details of the action selection policy, the direct code access procedure, and the Temporal Difference equation are elaborated below.

3.1 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 ϵ-greedy policy selects the action with the highest value with a probability of $1-\epsilon$ and takes a random action with probability ϵ [Pérez-Uribe, 2002]. With a constant ϵ value, the agent will always explore the environment with a fixed level of randomness. In practice, it may be beneficial to have a higher ϵ value to encourage the exploration of paths in the initial stage and a lower ϵ value to optimize the performance by exploiting familiar paths 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 larger state and action spaces will take a longer time to explore.

3.2 Direct Code Access

In an exploiting mode, an agent, to the best of its knowledge, selects an action with the maximal reward value given the current situation (state). Through a direct code access procedure, TD-FALCON searches for the cognitive node which matches with the current state and has the maximal reward value. For direct code access, the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$ are initialized by $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = (1,\ldots,1)$, and $\mathbf{x}^{c3} = (1,0)$. TD-FALCON then performs code activation and code competition according to equations (1) and (2) to select a cognitive node. The following lemma shows that if there is at least one cognitive node(s) encoding the current state, TD-FALCON with the given activity vectors will enable the selection of the cognitive node with the maximum reward value.

Direct Access Principle: During direct code access, given the activity vectors $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = (1,\ldots,1)$, and $\mathbf{x}^{c3} = (1,0)$, the TD-FALCON code activation and competition process will select the cognitive node J with the weight vectors $\mathbf{w}_J^{c1}$ closest to $\mathbf{S}$ and $\mathbf{w}_J^{c3}$ representing the maximal reward value, if one exists.

Proof (by Contradiction): Suppose TD-FALCON selects a F_2^c node J and there exists another F_2^c node K of which the weight vector $\mathbf{w}_K^{c1}$ is more similar to $\mathbf{S}$ than $\mathbf{w}_J^{c1}$ and the weight vector $\mathbf{w}_K^{c3}$ encodes a higher reward value than $\mathbf{w}_J^{c3}$. As $\mathbf{w}_K^{c1}$ is more similar to $\mathbf{S}$ than $\mathbf{w}_J^{c1}$, we derive that

$$\frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_K^{c1}|}{\alpha^{c1} + |\mathbf{w}_K^{c1}|} > \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_J^{c1}|}{\alpha^{c1} + |\mathbf{w}_J^{c1}|}. \quad (5)$$

Likewise, as $\mathbf{w}_K^{c3}$ encodes a higher reward than $\mathbf{w}_J^{c3}$, we have

$$\frac{|\mathbf{x}^{c3} \wedge \mathbf{w}_K^{c3}|}{\alpha^{c3} + |\mathbf{w}_K^{c3}|} > \frac{|\mathbf{x}^{c3} \wedge \mathbf{w}_J^{c3}|}{\alpha^{c3} + |\mathbf{w}_J^{c3}|}. \quad (6)$$

Table 1: TD−FALCON algorithm with direct code access.

1. Initialize the TD-FALCON network.
2. Sense the environment and formulate a state representation s.
3. Following an action selection policy, first make a choice between exploration and exploitation.
   If exploring, take a random action.
   If exploiting, identify the action a with the maximal Q(s,a) value by presenting the state vector **S**, the action vector **A**=(1,...1), and the reward vector **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 $\Delta Q(s,a) = \alpha \text{TD}_{err}$.
6. Present the corresponding state, action, and reward (Q-value) vectors (**S**, **A**, and **R**) to TD-FALCON for learning.
7. Update the current state by s=s'.
8. Repeat from Step 2 until s is a terminal state.

By equation (1), the above conditions imply that $T_K^c > T_J^c$, which means node K should be selected by TD-FALCON instead of node J (Contradiction).

[End of Proof]

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

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

3.3 Learning Value Function

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 TD formula.

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

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

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 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 $1 - Q\left(s,a\right)$, 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.

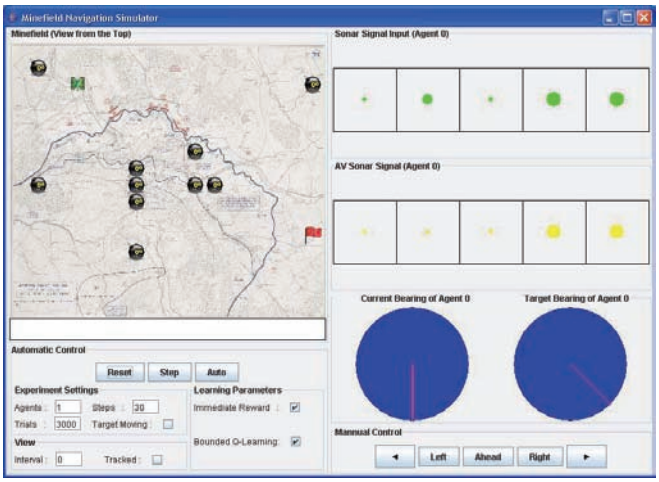

Figure 2: The minefield navigation simulator.

4 The Minefield Navigation Task

The objective of the given task is to teach an autonomous vehicle (AV) to navigate through a minefield to a randomly selected target position in a specified time frame without hitting a mine. In each trial, the AV starts from a randomly chosen position in the field, and repeats the cycles of sense, act, and learn. A trial ends when the AV 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. All results reported in this paper are based on a 16 by 16 minefield containing 10 mines as illustrated in Figure 2.

The AV has a rather coarse sensory capability with a 180 degree forward view based on five sonar sensors. 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. Other input attributes of the sensory (state) vector include the bearing of the target from the current position. In each step, the AV can choose 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 hits a mine. For the delayed reward scheme, no immediate reward is given at each step of the trial. A reward of 0 is given when the system runs out of time.

For learning the minefield task, we use a TD-FALCON network containing 18 nodes in the sensory field (representing 5x2 complement-coded sonar signals and 8 target bearing values), 5 nodes in the action field, and 2 nodes in the reward field (representing the complement-coded function value). TD-FALCON with direct code access 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 α was fixed at 0.5 and the discount factor γ was set to 0.9. The initial Q value was set to 0.5. For action selection policy, ϵ was initialized to 0.5 and decayed at a rate of 0.0005 until it dropped to 0.005.

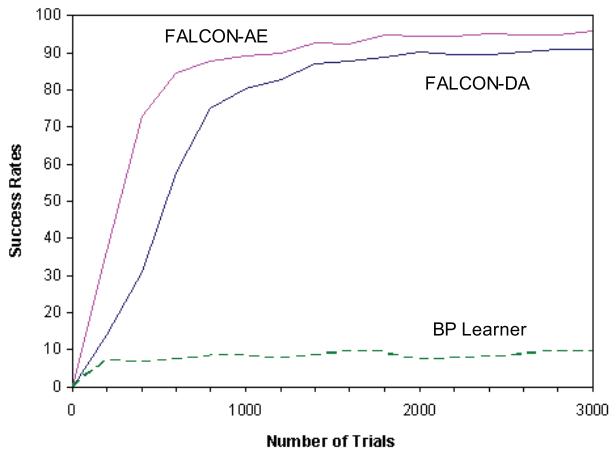

Figure 3: The success rates of TD-FALCON using action enumeration (AE) and direct code access (DA) compared with those of BP reinforcement learner.

To put the performance of FALCON in perspective, we further conducted experiments to evaluate the performance of an alternative reinforcement learning system, using the standard Q-learning rule and a multi-layer perceptron network, trained with a gradient descent based backpropagation algorithm, as the function approximator. We have chosen the backpropagation (BP) algorithm as the reference for comparison as gradient descent is by far one of the most widely used universal function approximation techniques and has been applied in many contexts, including Q-learning [Sun *et al.*, 2001] and adaptive critic [Werbos, 2004]. The BP learner employed a standard three-layer feedforward architecture to learn the value function with a learning rate of 0.25 and a momentum term of 0.5. The input layer consisted of 18 nodes representing the 5 sonar signal values, 8 possible target bearings, and 5 selectable actions. The output layer consisted of only one node representing the value of performing an action in a particular state. A key issue in using a BP network is the determination of the number of hidden nodes. We experimented with a varying number of nodes empirically and obtained the best results with 36 nodes. To enable a fair comparison, the BP learner also made use of the same action selection module based on the decay ϵ-greedy policy.

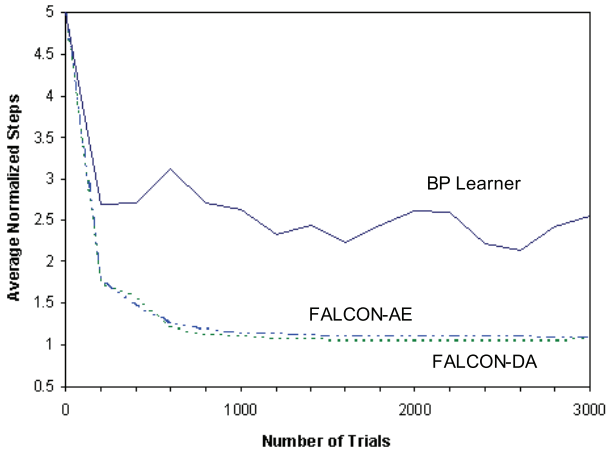

Figure 4: The average normalized steps taken by TD-FALCON using action enumeration (AE) and direct code access (DA) compared with those of BP reinforcement learner.

Figure 3 summarizes the performance of TD-FALCON with direct code access (FALCON-DA), the original TD-FALCON with action enumeration (FALCON-AE), and the BP reinforcement learner in terms of success rates averaged at 200-trial intervals over 3000 trials across 10 sets of experiments. We observed that the success rates of both FALCON-DA and FALCON-AE increased steadily right from the beginning. Beyond 1000 trials, both systems can achieve over 90 percent success rates. The backpropagation (BP) based reinforcement learner on the other hand required a much longer exploration phase. In fact, the BP learner only managed to reach around 90% success rates at 50,000 trials with a lower ϵ decay rate of 0.00001. Therefore, in the first 3000 trials, its success rates remained under 10%.

To evaluate how well the AV traverses from its starting position to the target, 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. As depicted in Figure 4, after 1000 trials, both FALCON-AE and FALCON-DA were able to reach the target in optimal or close to optimal paths in a great majority of the cases. The BP learner as expected produced unsatisfactory performance in terms of path optimality.

Considering network complexity, FALCON-DA demonstrated a great improvement over FALCON-AE by creating

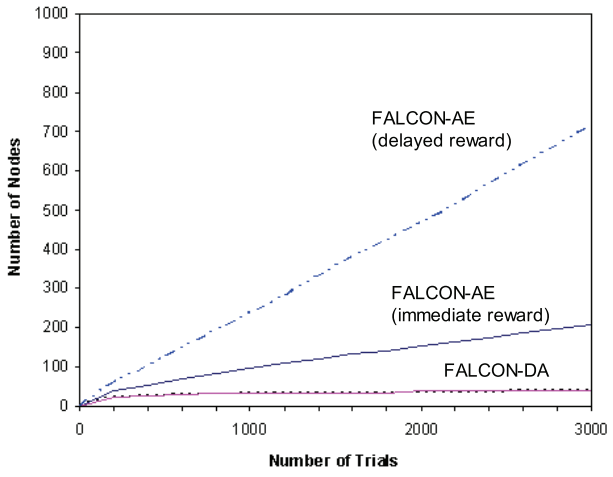

Figure 5: The average number of cognitive nodes created by TD-FALCON using action enumeration (AE) and direct code access (DA).

a much smaller set of cognitive nodes. In fact, the FALCON-DA networks consisting of around 38 cognitive nodes were almost the same size as a BP network with 36 hidden nodes. With a much more compact network structure, the efficiency of FALCON-DA in terms of computation time was also much better than FALCON-AE. The average reaction time taken to complete a sense-act-learn cycle clocked 0.12 millisecond, in contrast to 4 to 5 milliseconds as required by FALCON-AE.

5 Conclusion

We have presented an enhanced TD-FALCON model with a direct code access mechanism for selecting optimal actions in reinforcement learning. Besides that the network dynamics is more direct and natural, our comparative experiments show that TD-FALCON with direct code access is much more efficient than its predecessor in terms of computation time as well as network complexity. With a highly efficient learning architecture, our future work will involve applying FALCON to more complex and challenging domains. Another interesting extension is to interpret the learned cognitive nodes for explanation of the system's behaviour in performing specific tasks.

References

[Ackley and Littman, 1990] D.H. Ackley and M.L. Littman. Generalization and scaling in reinforcement learning. In *Advances in Neural Information Processing Systems 2*, pages 550–557. San Mateo, CA. Morgan Kaufmann, 1990.

[Carpenter and Grossberg, 1987] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[Carpenter *et al.*, 1991] 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*, 4:759–771, 1991.

[Kaelbling *et al.*, 1996] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Ninomiya, 2002] S. Ninomiya. *A hybrid Learning Approach Integrating Adaptive Resonance Theory and Reinforcement Learning for Computer Generated Agents*. Ph.D. Thesis, University of Central Florida, 2002.

[Pérez-Uribe, 2002] A. Pérez-Uribe. *Structure-Adaptable Digital Neural Networks*. PhD thesis, Swiss Federal Institute of Technology-Lausanne, 2002.

[Rummery and Niranjan, 1994] G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166, Cambridge University, 1994.

[Si *et al.*, 2004] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, editors. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, August 2004.

[Smith, 2002] A. J. Smith. Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15(8-9):1107–1124, 2002.

[Sun *et al.*, 2001] R. Sun, E. Merrill, and T. Peterson. From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, 25(2):203–244, 2001.

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

[Sutton, 1984] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. Thesis, University of Massachusetts, Amherst, MA, 1984.

[Tan and Xiao, 2005] A.-H. Tan and D. Xiao. Self-organizing cognitive agents and reinforcement learning in multi-agent environment. In *Proceedings, IEEE/WIC/ACM International Conference on Intelligent Agent Technologies*, pages 351–357. IEEE Computer Society press, 2005.

[Tan, 2004] A.H. Tan. FALCON: A fusion architecture for learning, cognition, and navigation. In *Proceedings, International Joint Conference on Neural Networks*, pages 3297–3302, 2004.

[Ueda *et al.*, 2005] H. Ueda, N. Hanada, H. Kimoto, and T. Naraki. Fuzzy Q-learning with the modified fuzzy ART neural network. In *Proceedings, IEEE/WIC/ACM International Conference on Intelligent Agent Technologies*, pages 308–315. IEEE Computer Society press, 2005.

[Watkins and Dayan, 1992] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[Werbos, 2004] P. Werbos. ADP: Goals, opportunities and principles. In Jennie Si, Andrew G. Barto, Warren Buckler Powell, and Don Wunsch, editors, *Handbook of Learning and Approximate Dynamic Programming*, pages 3–44. Wiley-IEEE Press, August 2004.