7-2010

Self-organizing neural networks for behavior modeling in games

Shu FENG

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

Self-Organizing Neural Networks for Behavior Modeling in Games

Shu Feng *Student Member* and Ah-Hwee Tan *Senior Member, IEEE*

***Abstract*— This paper proposes self-organizing neural networks for modeling behavior of non-player characters (NPC) in first person shooting games. Specifically, two classes of self-organizing neural models, namely Self-Generating Neural Networks (SGNN) and Fusion Architecture for Learning and COgnition (FALCON) are used to learn non-player characters' behavior rules according to recorded patterns. Behavior learning abilities of these two models are investigated by learning specific sample Bots in the Unreal Tournament game in a supervised manner. Our empirical experiments demonstrate that both SGNN and FALCON are able to recognize important behavior patterns and learn the necessary knowledge to operate in the Unreal environment. Comparing with SGNN, FALCON is more effective in behavior learning, in terms of lower complexity and higher fighting competency.**

I. Introduction

Modeling of Non-player characters (NPC) is crucial for the success of commercial games as it improves the playability of games and the satisfaction level of the players. Especially, in first person shooting games (FPS), autonomous NPC modeled by machine learning techniques make games more challenging and enjoyable [21]. Learning from behavior patterns is a new and promising approach to the modeling of NPC behavior as the knowledge acquired by learning directly builds the embedded knowledge of their behavior mechanism.

Learning defines the ability of obtaining knowledge automatically. There are many forms of learning, including unsupervised learning, supervised learning and reinforcement learning. Among the various learning paradigms, supervised learning is probably the most effective, due to its use of explicit teaching signals. In this paper, we adopt a supervised learning approach to building the behavior mechanism of non-player characters, by mimicking the behavior patterns of other players.

Self-organizing neural networks are a special class of neural networks that learn without explicit teaching signals. Recent development in self-organizing neural networks has extended them for supervised learning tasks. Compared with gradient descent based neural networks, they offer fast and real-time learning as well as self-scaling architectures that grow in response to signals received from their environment.

This paper studies two specific classes of self-organizing neural networks namely, Self-Generating Neural Networks (SGNN) [25], [9] and Fusion Architecture for Learning and COgnition (FALCON) [17], [26]. SGNN learns behavior rules through a hierarchical tree architecture. Compared with traditional neural networks, SGNN does not require a designer to determine the structure of the network according to the particular application in hand. However, the computational time of SGNN increases dramatically due to the continual creation of neural nodes. To overcome this problem, we propose a pruning method to optimize the SGNN network while maintaining the learning performance. TD-FALCON is a three-channel fusion Adaptive Resonance Theory (ART) network [19] that incorporates temporal difference methods [15], [23] into Adaptive Resonance Theory (ART) models [3], [2] for reinforcement learning. By inheriting the ART code stabilizing and dynamic network expansion mechanism, TD-FALCON is capable of learning cognitive nodes encoding multi-dimensional mappings across multi-modal input patterns, involving states, actions, and rewards, in an online and incremental manner. It has displayed superior learning capabilities, compared with gradient descent based reinforcement learning systems in various benchmark experiments [20], [26].

This paper investigates how these two classes of self-organizing models can be used to build autonomous players by learning the behaviour patterns of sample Bots in a first person shooting game, known as Unreal Tournament 2004 (UT2004). We conduct benchmark experiments to compare the duo in various aspects, including generalization capability, learning efficiency, and computational cost, under the same set of learning conditions. Our benchmark experiments show that, comparing with SGNN, FALCON learns faster and produces a higher level of generalization capability with a much smaller set of nodes. Online testing of NPC in the Death Match scenario also confirms that FALCON Bot produces a similar level of fighting competency to the Hunter Bot, which is not matched by Bots based on SGNN.

The rest of this paper is organized as follows. Section II reviews the related work in building NPC through machine learning techniques. Section III introduces the SGNN architecture with the network generating and pruning methods. Section IV introduces the FALCON architecture with the learning and action selection algorithms. Section V describes the Unreal Tournament 2004 domain and the behavior learning task. Section VI reports the experimental results. The final section concludes and discusses future work.

II. Related Work

Learning from observing, or imitation, has been a promising method for acquiring complex behaviors in various tasks, including recognizing human action sequences [10], training learning robots [12] and creating humanoid virtual characters [14]. Behavior patterns learned through this method provide the seeds or initial knowledge for autonomous agents. So

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore email: feng0027@ntu.edu.sg; asahtan@ntu.edu.sg

978-1-4244-8126-2/10/$26.00

far, many machine learning methods have been applied, including Bayesian method, fuzzy and neural network, Hidden Markov Model, and data mining methods. Gorman *et al.* [6] use the Bayesian method to imitate the behaviors and movement of NPC in a first person shooting game. However, the knowledge is associated with the specific environment learned. Therefore, the obtained behavior patterns have a weak transferability. Noda *et al.* [13] apply Markov Model to model the behaviour of soccer robots and teach the robots by Q-learning algorithm, but robots do not learn complex behaviors of other soccer characters. Lee *et al.* [11] apply data mining methods to sequential databases to find association rules of behavior patterns. That means it can find interrelationship between sequential attributes and actions. However, this method is not suitable to learn behavior rules involving states with multiple attributes.

There have also been extensive works on applying self-organizing neural networks to behavior learning. Barrios *et al.* [1] employ self-organization map (SOM) incorporating fuzzy theory to recognize the pattern groups in behaviors. However, this method requires the number of pattern classes and the architectures to be determined beforehand. Gaussier *et al.* [5] propose a neural architecture for a robot in order to learn how to imitate a sequence of movements performed by another robot. A self-organizing neural network is used to mimic a particular sequence of movement performed by another robot. However, this method focuses only on sequential behaviors such as movement. The learning does not consider external states. Therefore, the resultant robot lacks in adaptation ability and does not handle the changes in the environment. Wanitchaikit *et al.* [22] use self-organizing neural network to imitate the behavior from a teacher's demonstration. Then the learned network could help the robot to decide its action when it approaches the target. However, in this method, only the position of targets is considered as the feedback information. Therefore, it is not suitable for a complex environment.

The self-organizing models described above make use of a fixed architecture. In other words, the structure and the number of nodes in the network have to be determined before training. In addition, SOM performs iterative weight tuning, which is not suitable for real time adaptation. In this paper, we study two extensions of self-organizing neural networks, namely FALCON and SGNN, which have the unique advantages of self-growing architectures and fast incremental learning ability in common. By employing supervised learning to learn behavior patterns of existing players in the games, we aim to create NPC automatically, which have similar behaviour and fighting competency as their teachers.

III. Self-generating neural network

Self-generating neural network (SGNN) is firstly developed by Wen *et al.* [24], [25] based on self-organizing maps (SOM) and implemented as a self-generating neural tree (SGNT) architecture. Later, Inoue *et al.* [9], [8] improve the accuracy of SGNN by applying multiple systems and ensemble method. Self-generating neural network is appealing, as it does not require a designer to specify the structure of network and the class parameters. In addition, it has efficient learning ability and reasonably good adaptive ability. Because of these good attributes, SGNN is a suitable candidate for learning behaviour rules.

A. Architecture

SGNN is self-generating in the sense that there is no need to determine the network structure and the parameters beforehand. In other words, the network structure, including the number of neurons, their interconnections and the weights on the connections, are automatically constructed from a given set of training instances. The generated self-organizing network tree (SGNT) is a tree structure for hierarchical classification. Learning of SGNT is thus defined as a problem of constructing a tree structure from a given set of instances which consist of multiple attributes. As shown in Figure 1, each SGNT is rooted by a root neuron. Each neuron linked directly to the root neuron represents the center of a class. Each leaf node corresponds to an instance in the training data.

To explain the self-generation process, we first define the relevant notations [24] as follows:

Definition 1: Each input example e_i is a vector of real attributes: $e_i =< e_{i1}, ..., e_{im} >$, where e_{ik} represents the k^{th} attribute of e_i.

Definition 2: The j^{th} neuron n_j is expressed as an ordered pair (w_j, c_j), where w_j is the weight vector $w_j = (w_{j1}, w_{j2}, ..., w_{jm})$, and c_j is the number of the child neurons of n_j.

Definition 3: Given an input e_i, the neuron n_k in a neuron set $\{n_j\}$ is called a *winner* for e_i, if $\forall j$, $d(n_k, e_i) \leq d(n_j, e_i)$, where $d(n_j, e_i)$ is the Euclidean distance between neuron n_j and e_i. The *winner* can be the root neuron, a node neuron, or a leaf neuron.

The building process of SGNT is, in the nutshell, a hierarchical clustering algorithm. Initially, the neural network is empty. The generating algorithm is governed by a set of rules described as follows:

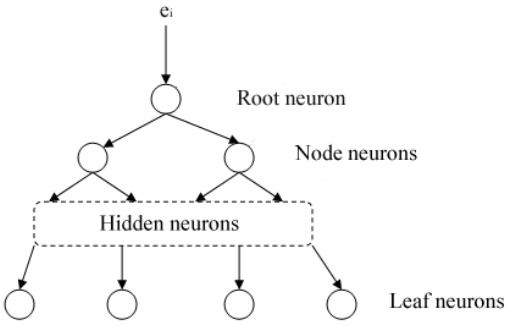

Fig. 1. The structure of Self-generating neural tree.

Node creation rule 1: Given an input example e_i, if $d(n_{winner}, e_i) > \xi$, where ξ is a predefined threshold, a new node n is generated by copying the weights (attributes) from

the current example. Then the new neuron is connected to the *winner* as its child.

Node creation rule 2: If the *winner* node n_{winner} is also a leaf node, another new node n is generated by copying the weights (attributes) from n_{winner}. A neuron can be called a leaf node only if it has no child.

Weight updating rule: The weight vector of neuron n_j is updated by the attribute vector of e_i according to (1):

$$w_{jk} = w_{jk} + \frac{1}{c_j + 1}(e_{ik} - w_{jk}), \tag{1}$$

where w_{jk} is the weight of n_j after learning the first k examples covered by n_j.

B. Pruning

An obvious weakness of self-generating neural network is the continual increase of the number of nodes as the number of samples increases. When the number of samples is very large, the training speed will slow down dramatically when the number of nodes is extremely large. In prior work, researchers also consider this problem [9], [8] and propose pruning algorithm for a multi-classifier system comprising of multiple SGNN. However, the multi-classifier system (MCS) is aimed at improving classification accuracy at the cost of more learning time and it is difficult to apply MCS in real time. Here we introduce a novel pruning method for single SGNN systems, which is aimed at improving learning and classification efficiency.

During the generating period of SGNN, let N_o denote the total number of input training samples and S denote the current number of nodes. In this pruning method, we introduce a threshold S_T, which is defined as:

$$S_T = \eta * N_o \quad (\eta > 0). \tag{2}$$

As the number of input samples increases during the training period, the pruning procedure kicks in when the number of nodes exceeds this threshold ($S > S_T$), which can be set according to a function of the learning speed. When pruning occurs, it checks the connections among the root neuron, the node neurons, and the leaf neurons for redundancies. If there is a hidden node h positioned between the leaf node and its corresponding node neuron, h will be deleted. The leaf node is then connected to the node neuron directly as its child. The weights of node neuron c are then updated according to (3):

$$w_{ci} = \frac{N_c \cdot w_{cj} - w_{hj}}{N_c - 1}, \tag{3}$$

where N_c is the number of examples covered by c. The number of examples covered by c is decreased to $N_c^{'} = N_c - 1$ accordingly.

IV. FALCON

FALCON network learns cognitive codes across multi-channel mappings simultaneously across multi-model input patterns involving sensory input, actions, and rewards. By using competitive coding as the underlying adaptation principle, the network dynamic encompasses a myriad of learning paradigms, including unsupervised learning, supervised learning, as well as reinforcement learning.

Although various models of ART have been widely applied to pattern analysis and recognition tasks, there have been very few attempts to used ART-based networks for building autonomous systems. In this paper, we apply FALCON to learn specific behavior patterns from sample Bots in UT2004 game environment.

A. Architecture

FALCON employs a three-channel architecture (Figure 2) comprising a category field F_2^c 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 the reward values. The dynamics of FALCON based on fuzzy ART operations [4] [16], is described below.

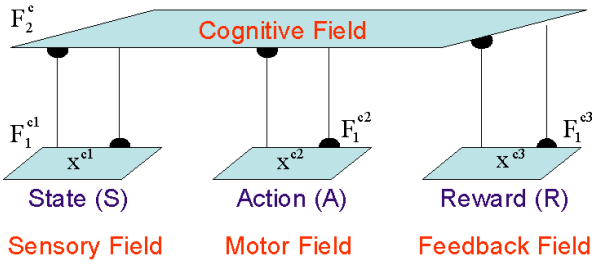

Fig. 2. The FALCON architecture.

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

Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector for $k = 1, ..., 3$. 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 representation in F_1^{ck} for $k = 1, ..., 3$. 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 $k = 1, ..., 3$; learning rate parameters $\beta^{ck} \in [0, 1]$ for $k = 1, ..., 3$; contribution parameters $\gamma^{ck} \in [0, 1]$ for $k = 1, ..., 3$ where $\sum_{k=1}^{K} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0, 1]$ for $k = 1, ..., 3$.

B. Supervised Learning

In supervised learning mode, FALCON learns an action policy which maps directly from states to desired actions. Given the state vector $\mathbf{S}$ and an action vector $\mathbf{A}$, the activity vectors are set as $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{R} = (1, 0)$. FALCON then performs code activation to select a category

node J in the F_2^c field to learn the association between $\mathbf{S}$ and $\mathbf{A}$. The detailed algorithm is presented as follows.

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

$$T_j^c = \sum_{k=1}^{K} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}, \tag{4}$$

where the fuzzy AND operation $\wedge$ is defined by $(p \wedge q)_i \equiv \min(p_i, q_i)$ and the norm $|\cdot|$ is defined by $|p| \equiv \sum_i p_i$ for vectors p and 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

$$T_J^c = \max\{T_j^c : \text{for all} \quad F_2^c \quad \text{node} \quad j\}. \tag{5}$$

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 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 activity patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen node J meets its vigilance criterion

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \tag{6}$$

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 all $1s$.

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:

$$\mathbf{w}_J^{ck(new)} = (1 - \beta^{ck})\mathbf{w}_J^{ck(old)} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(old)}). \tag{7}$$

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.

Code creation: Our implementation of FALCON maintains ONE uncommitted node the F_2^c field at any one time. 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.

C. Action Selection

Given a state vector $\mathbf{S}$, FALCON selects a category node J in the F_2^c field which determines the action. For action selection, the activity vectors $\mathbf{x^{c1}}$, $\mathbf{x^{c2}}$, and $\mathbf{x^{c3}}$ are initialized by $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = (1, ..., 1)$, and $\mathbf{x}^{c3} = (1, 0)$. Through a direct code access procedure [18], FALCON searches for the cognitive node which matches with the current state using the same code activation and code competition processes according to equations (4) and (5).

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

$$\mathbf{x}^{c2(new)} = \mathbf{x}^{c2(old)} \wedge \mathbf{w}_J^{c2}. \tag{8}$$

FALCON then examines the output activities of the action vector $\mathbf{x}^{c2}$ and selects an action a_I, which has the highest activation value

$$x_I^{c2} = \max\{x_i^{c2(new)} : \text{for all} \quad F_1^{c2} \quad \text{node} \quad i\}. \tag{9}$$

V. Learning Behavior Patterns in Unreal 2004

A. UT2004 Environment

Unreal Tournament 2004 (UT2004) is a first person shooting game featuring close combat fighting between robots and human. Figure 3 provides a snapshot of the game environment taken from the view of a human player. The armed soldiers running and shooting in the environment are non-player characters, called Bots. The gun shown at the lower right hand corner is controlled by the human player. In our experiments, we use a "Deathmatch" mode, in which every Bot must fight with any other player in order to survive and win.

UT2004 does not merely offer an environment for gaming. More importantly, it also provides a platform for building and evaluating autonomous agents. Specifically, an Integrated Development Environment (IDE), called Pogamut [7], is available to developers for building agents for the UT environment. This means the developers can implement their own agents (or Bots) using any specific algorithms and run them in UT. Running as a plug-in for the NetBeans Java development environment, Pogamut communicates with the UT2004 game through Gamebots 2004 (GB2004), which is a built-in server inside UT2004 for exporting information from the game to the agent and vice versa. Pogamut also

TABLE I

THE EIGHT BEHAVIORS OF THE HUNTER BOT.

No.	*Behaviors*	*Description*
A_1	$ChangeToBetterWeapon$	Switch to a better weapon.
A_2	$Engage$	Shooting the enemy.
A_3	$StopShooting$	Stop shooting.
A_4	$ResponseToHit$	Turn around, try to find the enemy.
A_5	$Pursue$	Pursue the enemy spotted.
A_6	$Walking$	Walk and check walking path.
A_7	$GrabItem$	Grab the most suitable item.
A_8	$GetMedicalKit$	Pick up medical kit.

TABLE II

THE TEN STATE ATTRIBUTES OF THE HUNTER BOT.

No.	*State attributes*	*Type*	*Description*
Att_1	$SeeAnyEnemy$	Boolean	See enemy?
Att_2	$HasBetterWeapon$	Boolean	Have a better weapon?
Att_3	$HasAnyLoadedWeapon$	Boolean	Have weapon loaded?
Att_4	$IsShooting$	Boolean	Is shooting?
Att_5	$IsBeingDamaged$	Boolean	Is being shot?
Att_6	$LastEnemy$	Boolean	Have enemy target to pursue?
Att_7	$IsColliding$	Boolean	Colliding with wall?
Att_8	$SeeAnyReachableItemAndWantIt$	Boolean	See any wanted item?
Att_9	$AgentHealth$	$[0, 1]$	Agent's health level
Att_{10}	$CanRunAlongMedKit$	Boolean	Medical kit can be obtained?

TABLE III

THE HARD-CODED RULES OF THE HUNTER BOT IN UT2004.

No.	*IF (Condition)*	*THEN (Behavior)*
1	see the enemy, has better weapons, and is being shot by others	$ChangeToBetterWeapon$
2	see the enemy, has weapon loaded, and is being shot	$Engage$
3	has weapon loaded, is shooting and being shot, and is pursuing enemy	$StopShooting$
4	is pursuing enemy, has weapon loaded, and gets damaged	$ResponseToHit$
5	has weapon loaded	$Pursue$
6	has weapon loaded, see some item he want, but colliding on the path	$Walking$
7	spots some item and want it	$GrabItem$
8	has weapon loaded, and health level is weak	$GetMedicalKit$

Fig. 3. Unreal Tournament 2004 game environment.

has a built-in parser module, which is used for translating messages into Java objects and vice versa.

B. The Behavior Learning Task

We focus on the task of learning from the behaviour patterns from a sample Bot called Hunter provided in UT2004. *Hunter* is a rule-based Bot, which exhibits a full range of combat competency, including fighting with enemies and making use of resources such as weapons and medical kits. Hunter has eight types of behaviors (shown in Table I) which he switches from one to the other based on ten state attributes (shown in Table II). With the exception of the health attribute, all attributes are boolean. There are in total eight main rules captured in the Hunter's behavior mechanism based on these state attributes, which are summarized in Table III.

When playing the UT2004 game, the internal states, ex-

ternal states, and behavior patterns of Hunter are recorded as training data. Each training example consists of a vector of the state attribute values as well as the behaviour (action chosen). The collected data are then used to train the self-organizing neural models using the supervised learning paradigm. After learning, the behavior pattern rules can be utilized as the embedded knowledge of a new bot. By assimilating the behaviour of the sample Bot, the new Bot is expected to exhibit similar behavior patterns in the same environment and produce comparable fight competency to the sample Bot.

VI. Experiments

To evaluate the effectiveness of SGNN and FALCON in learning NPC, we first conduct benchmark experiments based on off-line learning to compare their performance in terms of learning time, generalization capability and computational cost. Online testing of Bots are subsequently conducted, wherein we investigate the competency of the new Bots when fighting against the sample Bot which they learn from.

A. Off-line Testing

We first conduct empirical experiments to evaluate the performance of SGNN and FALCON in off-line learning. The data set consists of a total of 8000 training samples and 8000 test samples generated by the Hunter Bot. By training on a varying number of training examples, we test the generalization capability of SGNN and FALCON on an equivalent number of test samples. We also measure their efficiency in terms of the number of internal nodes/rules created and the time taken for learning.

In our experiments, SGNN and FALCON use a standard set of parameter values. For SGNN, we adopt $\xi = 0$ and $\eta = 1.5$. For FALCON, we adopt the parameter setting as follows: choice parameter α=(0.1, 0.1, 0.1); learning rate parameter β=(1, 1, 1); contribution parameter γ=(1, 0, 0); and vigilance parameter ρ=(1, 1, 0).

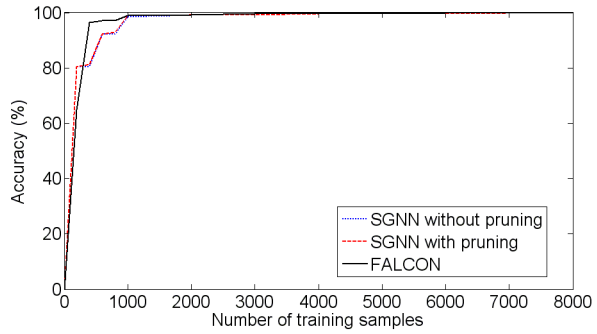

Fig. 4. The accuracy of baseline SGNN, SGNN with pruning, and FALCON in classifying the test samples.

Figure 4 summaries the performance of the baseline SGNN (without pruning), SGNN with pruning, and FALCON, in terms of the classification accuracy on the test set. As the size of the training data increases, the accuracies of all three models converge to 100%. In general, SGNN with pruning achieves roughly the same accuracy level as SGNN without pruning. Comparing with SGNN, FALCON shows a faster rate of convergence by obtaining a higher accuracy with small data sets.

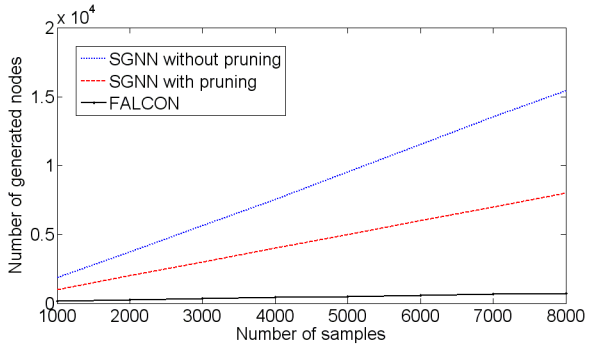

Fig. 5. The number of nodes generated by baseline SGNN, SGNN with pruning, and FALCON.

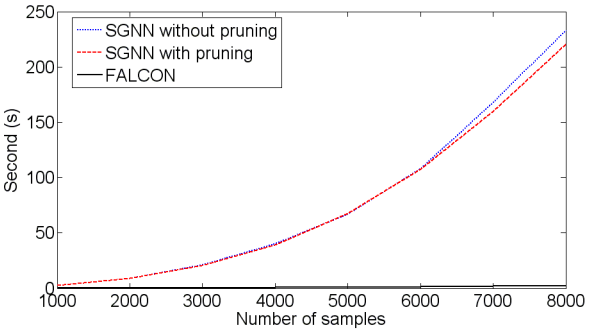

Fig. 6. The learning time of baseline SGNN, SGNN with pruning, and FALCON.

Figure 5 depicts the performance of the baseline SGNN (without pruning), SGNN with pruning, and FALCON, in terms of the average number of neurons/nodes created during learning. We see that the pruning method greatly reduces the number of neurons in SGNN. However, the number of nodes created by FALCON is significantly less than those of SGNN.

Figure 6 shows the learning time taken by baseline SGNN without pruning, SGNN with pruning, and FALCON. For the two SGNNs, the learning time is about the same. Nevertheless, considering all aspects, the pruning method is still proved to be effective for SGNN, as it effectively reduces the number of neurons, while maintaining the learning accuracy. Partly because the number of nodes generated by FALCON is the least among these three systems, the required learning time is also significantly less than those of SGNN. This suggests that FALCON is clearly a more suitable candidate for real time learning and performance.

B. Online Testing of SGNN Bots

In this section, experiments are conducted in the UT2004 game environment to check if the Bots created based on the two SGNN models could learn the behavior patterns and contend against the Hunter Bot. In this set of the experiments,

all SGNN Bots are trained using 8000 training sample data recorded from the Hunter Bot.

Under the Deathmatch scenario, each of the learning Bots enters into a series of one-on-one battles with the Hunter Bot. When a Bot kills its opponent, one point is awarded. The battle repeats until any one of the Bots reaches a maximum score of 25. During the battles, the scores, updated in intervals of 25 seconds, indicate the fighting competency of the Bots. For benchmark purpose, we run the game for ten times and record the average scores obtained.

1) Experiment 1: Battle between Hunter and SGNN Bot: This experiment examines the competency of the Bot created using the baseline SGNN (without pruning). As shown in Figure 7, the SGNN Bot can achieve a respectable level of performance but its scores are always 2 to 3 points lower than those of Hunter.

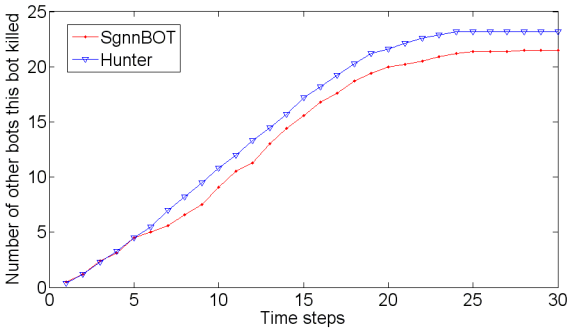

Fig. 7. Performance of SGNN Bot fighting against Hunter.

2) Experiment 2: Battle between Hunter and Pruned SGNN Bot: This experiment examines the competency of the Bot created using SGNN with pruning. As shown in Figure 8, after applying SGNN pruning, the new Bot produces a lower level of performance, widening the gap between the competency of SGNN Bot and Hunter.

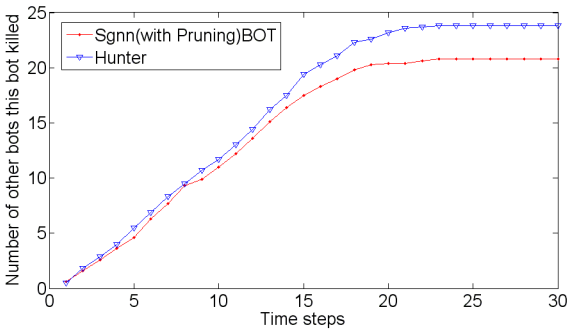

Fig. 8. Performance of SGNN Bot (with pruning) fighting against Hunter.

C. Online Testing of FALCON Bot

In this section, a series of experiments are conducted in UT2004 game environment to check if the Bots created based on FALCON could learn the behavior patterns and contend against the Hunter Bot. The FALCON Bot, trained using the same 8000 training samples data recorded from the Hunter Bot, consists of 647 behavior rules.

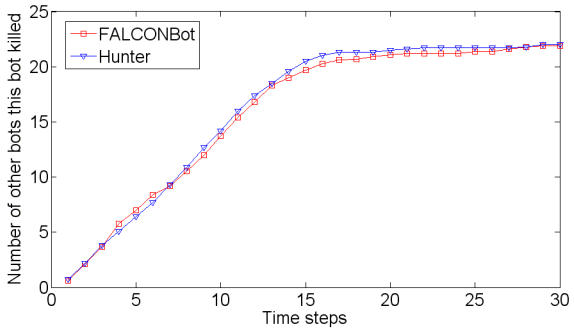

Fig. 9. Performance of FALCON Bot fighting against Hunter.

Figure 9 shows the scores of the FALCON Bot fighting against Hunter averaged across ten games. We see that the fighting competency of FALCON Bot is almost identical to that of Hunter. This shows that FALCON has learned most, if not all, of the Hunter's knowledge perfectly. Comparing with Bots created based on SGNN, FALCON Bot is thus obviously a much better learner in assimilating the behavior patterns from the Hunter Bot.

Table IV shows a set of sample rules learned by FALCON. Their translated symbolic form, as exemplified in Table V, shows that FALCON rules are close to the original rules of Hunter and are easy to interpret.

VII. Conclusion

Learning from behavior patterns is becoming a promising approach to modeling non-player characters (NPC) in computer games. This paper has successfully shown that two classes of self-organizing neural networks, namely self-generating neural network (SGNN) and Fusion Architecture for Learning, and COgnition (FALCON), can be used to learn behaviour patterns of sample characters and produce new NPCs with similar behaviour and a comparable level of performance. Our empirical experiments based on the Unreal Tournament game also show that, compared with SGNN, FALCON is able to achieve a higher level of performance with a much more compact network structure and a much shorter learning time.

Moving forward, we aim to create more versatile NPCs which are able to further learn and adapt during game play in real time. As FALCON is designed to support a myriad of learning paradigms, including unsupervised learning, supervised learning and reinforcement learning [19], it is our natural choice for modeling autonomous NPCs in games.

References

[1] D. Barrios-Aranibar and P.J. Alsina. In *Hybrid Intelligent Systems, 2005. HIS '05. Fifth International Conference on*, page 6, 2005.

[2] G. A. Carpenter and S. Grossberg. ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, July 1987.

[3] 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, June 1987.

TABLE IV

FALCON RULE EXAMPLES OF LEARNING "HUNTER" IN UT2004

No.	Att_1	Att_2	Att_3	Att_4	Att_5	Att_6	Att_7	Att_8	Att_9	Att_{10}	*Action*
R_1	1	1	1	0	0	0	1	0	0.481	1	A_1
R_2	1	0	1	0	1	0	0	1	0.489	1	A_2
R_3	0	0	1	1	1	1	0	0	0.874	1	A_3
R_4	0	0	1	0	1	1	0	0	0.474	1	A_4
R_5	0	0	1	0	0	1	0	0	0.953	1	A_5
R_6	0	0	1	0	0	0	1	1	0.384	1	A_6
R_7	0	0	1	0	0	0	0	1	0.216	1	A_7
R_8	0	0	1	0	0	0	0	0	0.205	1	A_8

TABLE V

FALCON RULES IN SYMBOLIC FORM.

No.	*IF (Condition)*	*THEN (Behavior)*
R_5	weapon loaded, 95.3% health, enemy spotted, and medical kits around	$Persue$
R_7	weapon loaded, 21.6% health, see some needed item, and it can be obtained	$GetMedicalKit$

[4] 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.

[5] P. Gaussier, S. Moga, J.P. Banquet, and M. Quoy. From perception-action loops to imitation precesses: A bottom-up approach of learning by imitation. *Applied Artificial Intelligence*, 7-8(12):701–727, 1998.

[6] B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys. Bayesian imitation of human behavior in interactive computer games. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 1244–1247, 2006.

[7] Pogamut homepage is available online. http://artemis.ms.mff.cuni.ca/pogamut/.

[8] H. Inoue and H. Narihisa. Effective online pruning method for ensemble self-generating neural networks. In *Midwest Symposium on Circuits and Systems*, volume 3, pages III85 – III88, Hiroshima, Japan, 2004.

[9] H. Inoue and H. Narihisa. Self-organizing neural grove and its applications. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1205 – 1210, Montreal, QC, Canada, 2005.

[10] Y. Kuniyoshi and H. Inoue. Qualitative recognition of ongoing human action sequences. In *International Joint Conference on Artificial Intelligence (IJCAI93)*, volume 13, pages 1600–1609, 1993.

[11] S. C. Lee, E. Lee, W. Choi, and U. M. Kim. Extracting temporal behavior patterns of mobile user. In *Fourth International Conference on Networked Computing and Advanced Information Management*, pages 455–462, Sept. 2008.

[12] H. Miyamoto and M. Kawato. A tennis serve and upswing learning robot based on bi-directional theory. *Neural Networks*, 11(7/8):1331–1344, 1998.

[13] I. Noda. Hierarchical hidden markov modeling for teamplay in multiple agents. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 38–45, Oct 2003.

[14] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

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

[16] A.-H. Tan. Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transaction on Neural Networks*, 8(2):237–250, 1997.

[17] A.-H. Tan. FALCON: A fusion architecture for learning, cognition, and navigation. In *2004 IEEE International Joint Conference on Neural Networks*, volume vol.4, pages 3297 – 3302, Piscataway, NJ, USA, 2004.

[18] A.-H. Tan. Direct code access in self-organizing neural architectures for reinforcement learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1071–1076, 2007.

[19] A.-H. Tan, G.A. Carpenter, and S. Grossberg. Intelligence Through Interaction: Towards A Unified Theory for Learning. In *Proceedings of the 4th International Symposium on Neural Networks: Advances in Neural Networks, LNCS 4491*, pages 1094–1107, 2007.

[20] A.-H. Tan, N. Lu, and D. Xiao. Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. *IEEE Transactions on Neural Networks*, 9(2):230–244, 2008.

[21] D. Wang, B. Subagdja, A.-H. Tan, and G. W. Ng. Creating human-like autonomous players in real-time first person shooter computer games. In *Proceedings of Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence (IAAI'09)*, pages 14–16, Pasadena,California, July 2009.

[22] S. Wanitchaikit, P. Tangamchit, and T. Maneewarn. Self-organizing approach for robot's behavior imitation. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3350–3355, May 2006.

[23] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[24] W.X. Wen, H. Liu, and A. Jennings. Self-generating neural networks. In *Neural Networks, 1992. IJCNN., International Joint Conferencen*, volume 4, pages 850 – 855, June 1992.

[25] W.X. Wen, V. Pang, and A. Jennings. Self-generating vs. self-organizing, what's different? In *1993 IEEE International Conference on Neural Networks*, pages 1469 – 1473, New York, NY, USA, 1993.

[26] D. Xiao and A.-H. Tan. Self-organizing neural architectures and cooperative learning in multi-agent environment. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 37(6):1567–1580, 2007.