Inconsistent Training for Developmental Networks and the Applications in Game Agents

Hao Ye*, Xuanjing Huang*, Juyang Weng†*
*School of Computer Science, Fudan University, Shanghai, 201203 China
†Department of Computer Science and Engineering, Cognitive Science Program, Neuroscience Program,
Michigan State University, East Lansing, MI, 48824 USA
Email: 10110240032@fudan.edu.cn, xjhuang@fudan.edu.cn, weng@cse.msu.edu

***Abstract*—Although a caregiver tries to be consistent while she teaches a baby, it is not guaranteed that she never makes errors. This situation is also true with a digital game, during which the human player needs to teach a non-player character (NPC). In this work, we report how a teacher can successfully train the Developmental Networks (DNs) while she cannot guarantee an error-free sequence of motor-supervised teaching. We establish that, under certain conditions, a DN tolerates a significant number of errors in a teaching sequence as long as the errors do not overwhelm the correct motor supervisions in terms of the Z-normalized frequency. We also provide theoretical arguments why task-nonspecific agents like DNs create a new dimension for the play values of future digital games. The emergent representations in the DN can not only abstract well like a symbolic representation (e.g., Finite Automaton) but also deal with the problem of exponential complexity with the traditional symbolic representations currently prevailing in the artificial intelligence (AI) field and in the digital gaming field. The experimental results showed that the speed of convergence to correct actions depends on the error rates in training.**

I. Introduction

Since the early days of computers, probably first coined by Allan Turing [1], many researchers [2], [3] have strived to build robots that resemble what Turing called "child machines", in the sense that the robot can be taught through its autonomous interactions with humans, like how a human caregiver interacts with a human child. This requires a robot-like machine to directly interact with the physical world, which includes human teachers.

In the field of computer game playing, much progress has been made based on a much simpler interface and a much restricted domain. Many computer games utilize AI techniques to improve the gameplay by NPCs. In a "clean" game environment, some NPCs' level of expertise are well above of many human players. For example, the Deep Blue computer system, consisting of many networked computers [4], has won the arguably the best human player of chess. However, that competition is under well controlled digital interface where the Deep Blue takes only well-specific instructions of logic moves. Among the AI techniques, neural networks and machine learning methods were cited to be the most likely methods to reach human expert levels, and they have achieved impressive successes in computer game playing [5], [6].

However, fundamental challenges persisted. First, many algorithms require handcrafted task-specific representations. The game designers build several types of agents for different stages of a game setting, such as path planning, state classification and skill look-up. The most popular engine is the Finite Automaton (FA), where each state is represented by a different symbol whose meaning is specified by a design document. This results in an increased complexity in game design and a high demand on human resource, computation time, and storage space.

Second, such approaches produce a limited amount of behaviors as well as their sophistication. As Weng [7] analyzed, the number of states in a symbolic representation (e.g., FA) is exponential in the number of concepts (e.g., the number of setting variables and agent actions). If there is a new type of representation that can address this problem without requiring the human designer to consider the exponential number of concept combinations, the playing value of the game can be considerably increased since the number of possible action combinations is too large for the human player to memorize, predict, and get bored.

In this work, we investigate whether DN is capable of addressing these two problems and its limitations. We hope to use DN as a general-purpose "brain-like" engine that is applicable to a variety of games. For each game, the human designer prudently design the game environment which includes the allowed response from the human player and the display to the human player. The customization of the engine to a game includes the design of the symbolic sensory port, the symbolic action port, and the specification of the computational resource for the engine. In this sense, the developmental (learning) program of the DN is sensor specific, effector specific, and "species" (e.g., resource) specific, but not task specific. Equipped by the DN engine, the NPC can learn new skills from the game environment through interactions with the human player (e.g., supervision) but most of time it is the NPC to autonomously generate its actions based on (1) the factory trained skills, and (2) the previously learned skills from the human player.

Mainly inspired by the brain anatomy (e.g., Felleman & Van Essen [8]), DN regards a simplified brain-like network to have three areas, the sensory area X, the internal (brain) area Y and the motor area Z. Different from many artificial neural networks, the internal neurons in Y have two-way connection with both X and Z. In principle, the X area could potentially

model any sensory modality (e.g., vision, audition, and touch), but in a game, it represents symbolic sensory information (e.g., presence of the owner). The motor area Z serves as both input and output. When the environment supervises Z, Z is the input to the network. Otherwise, Z gives an output vector to drive effectors which act on the real world. For a game, the Z vector consists of many concept zones (e.g., move forward or good). Within each concept zone, each neuron represents a particular value of the concept (e.g., a speed). The order of areas from low to high is: X, Y, Z. For example, X provides bottom-up input to Y, but Z gives top-down input to Y.

The DN learns incrementally while it performs and learns concurrently. The learning mechanism used is the biologically inspired Hebbian learning, which is cell-centered and in-place in the sense that each neuron is responsible for the learning by itself and it does not require other extra-cellular mechanisms to deal with its learning (e.g., no extra-cellular mechanism is allowed to compute the covariance matrix of its input vector). This results in a favorable low degree of computational complexity for each neuron (linear in the number of inputs n for each neuron) and a low degree of computational complexity for each network update: $O(nc)$ where n is the average number of connections for each neuron and c is the number of neurons.

The first theorem in [9] with the proof in [9] has establish the following properties. Give an FA, a DN can learn the FA from the sequential observations of the input σ and its state output q, by updating the DN for one observation at a time. The DN immediately learn all the behavior of FA immediately and error-free. Therefore, for the game application, we expect that DN immediately shows the effect of each instance of learning immediately. A different NPC uses a different customization of DN. Therefore, group behaviors are natural outcome of such NPC, but this work does not address group behaviors.

An important property of DN is that its internal representation emerges from interactions with the external environments via its sensory port X and its effector port Z. This makes DN not only highly flexible and adaptable, but also highly resource saving in the human designer's time. The vectorization of sensory information and action information enables DN to compute the similarity between two states, which leads to the ability of generalization under limited computational resource and learning experience. The vectorization of sensors and motors also enables the DN to handle the problem of combinatorial explosion of states by partitioning the states into different lobe regions [10].

From WWN-1(Where-What Network) to WWN-5, five embodiments of DN, DN has been tested so far for perfect training signals only. However, it is impractical for a human player to only teach an NPC without making any error. In this work, we study how DN deals with inconsistent training experience, the effects on its performance and learning speed. We found that a DN converges to a correct behavior even though we teach them some errors occasionally. Intuitively, when one learns to drive along a trajectory, his steering wheel sometimes turns to left too much and sometimes to right too much, but over time, the brain can autonomously generate more and more accurate muscle signals for the desired heading direction in order to follow a trajectory . This property adds more flexibility for DN in potential game applications. We use our first game software currently under development, EpigenBuddy, as the test bed.

In remainder of this paper, we will introduce some related works on DN and digital games in section II. Section III outlines DN architecture and algorithm. In section IV, we discuss how we customize the DNs for the EpigenBuddy, and present the simulation results with inconsistent training for an game setting represented as an FA. Section V gives some concluding remarks.

II. Related Work

There have been many successful applications of DN before we use it to build NPCs in digital games. DN was used for processing spatial information and obtaining spatial attention [11]. Miyan et al. [12] used DN to process temporal information and obtain temporal attention. Zhang & Weng [13] built the skill transfer model by using developmental architecture. DN also has shown its capability on spoken language [14] and speech recognition [15].

On the other hand, since the early stages of game industry, people recognized the challenges for AI in digital games [16]. Deep Blue beat the world chess champion Kasparov in 1997 was marked as the hallmark of the success of using strategy AI algorithms in board and card games. With the development of different digital game genres such as "First-Person Shooter Games", "Action-adventure Game", "Role-Playing Games", "Strategy Games", and "Simulation Games", modern AI techniques were required to accomplish complex tasks such as agent's perception, navigation and decision making [17]. Finite state machines and neural networks were used in behavior control [5], [18], rule-based systems for goal planning [19], evolutionary algorithms were used for tuning parameters [20]. In [21], three main categories of AI techniques (i.e. neural network, evolutionary approach and reinforcement learning method) were fully discussed in digital game filed. As a matter of fact, neural networks have shown the power in various kinds of learning tasks. Particularly, it appears to be well suited for player modeling, however such an approach requires a comprehensive data samples for training and validation. [22] provided a case study of using supervised neural network in digital game where a robot was trained.

However, neural networks have proved to be hard to train, for it is inefficient to tune parameters and requires a quantity of training examples to cover all possible states that the agents may encounter in the game [23]. If we trained the neural networks off-line, especially when a player-modeling approach was utilized, it is necessary to obtain training examples from a range of players to ensure the generation of useful game agent behaviors [24].

Some alternative training methods for neural networks, especially for online learning were proposed in [25]. In [26] some constraints on current intelligent interface for digital

games were concluded, and some criterions for new kind of intelligent agent were proposed. In [27], the author tried to build human-like agent in digital games, and he tested his agent in his game Social Ultimatum.

III. Developmental Network

A DN is not task-specific as argued by Weng et al [3]. Its concepts are learned dynamically and incrementally through interactions with the environments. In [28], Weng discussed that a DN can simulate any FA. In this section, we will briefly introduce the DN.

A. DN architecture

A basic DN, has three areas, the sensory area X, the internal ("brain") area Y and the motor area Z. An example of DN is shown in Fig. 1. The internal neurons in Y have bi-directional connection with both X and Z.

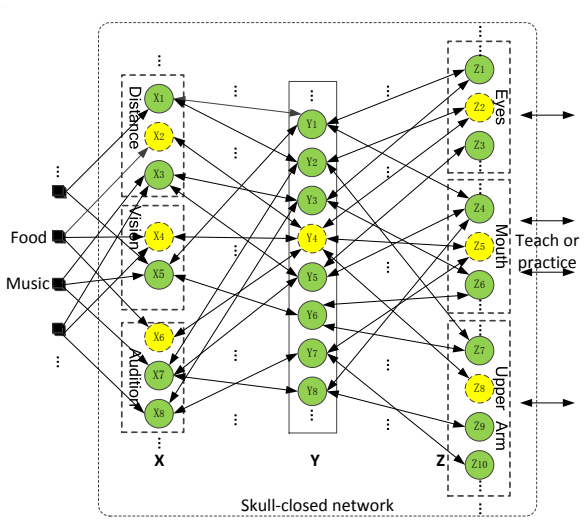

Fig. 1. A DN has three parts: X zone response for the sensor, Z zone response for motor, and Y zone correspond to the inner "brain" which has bi-directional connections with both X and Z. In our game, X does not use Y inputs. Pixel in yellow and circled by dash-line means it is activated. Pixels grouped in dotted box are in the same concept zone.

The DP(Developmental Program) for DNs is not task-specific as suggested for the "brain" in [3] (e.g., not concept-specific or problem specific). In contrast to a static FA, the motor area Z of a DN can be directly observed by the environment (e.g., by the teacher) and thus can be calibrated through interactive teaching from the environment. The environmental concepts are learned incrementally through interactions with the environments. For example, in Fig. 1, the "Food" object makes the pixels 2, 4 and 6 activated and all other green pixels remain normal. However, such an image from the "Food" object is not known during the programming time for the DP.

In principle, the X area can model any sensory modality (e.g., vision, audition, and touch). The motor area Z serves both input and output. When the environment supervises Z, Z is the input to the network. Otherwise, Z gives an output vector to drive effectors (muscles) which act on the real world. The order of areas from low to high is: X, Y, and Z. For example, X provides bottom-up input to Y, but Z gives top-down input to Y.

B. DN algorithm

DN is modeled as an area of the "brain". It has its area Y as a "bridge" for its two banks, X and Z. If Y is meant for modeling the entire "brain", X consists of all receptors and Z consists of all muscles neurons. Y potentially can also model any Brodmann area in the "brain". According to many studies in detailed review by Felleman & Van Essen [8], each area Y connects in both ways with many other areas as its two extensive banks.

The most basic function of an area Y seems to be prediction — predict the signals in its two vast banks X and Y through space and time. The prediction applies when part of a bank is not supervised. The prediction also makes its bank less noisy if the bank can generate its own signals (e.g., X).

A secondary function of Y is to develop bias (like or dislike) to the signals in the two banks, through what is known in neuroscience as neuromodulatory systems.

The DN algorithm is as follows. Input areas: X and Z. Output areas: X and Z. The dimension and representation of X and Y areas are hand designed based on the sensors and effectors of the robotic agent or biologically regulated by the genome. Y is skull-closed inside the "brain", not directly accessible by the external world after the birth.

1) At time $t = 0$, for each area A in $\{X, Y, Z\}$, initialize its adaptive part $N = (V, G)$ and the response vector $\mathbf{r}$, where V contains all the synaptic weight vectors and G stores all the neuronal ages. For example, use the generative DN method discussed below.
2) At time $t = 1, 2, ...$, for each A in $\{X, Y, Z\}$ repeat:
   a) Every area A performs mitosis-equivalent if it is needed, using its bottom-up and top-down inputs $\mathbf{b}$ and $\mathbf{t}$, respectively.
   b) Every area A computes its area function f, described below,
   $$(\mathbf{r}', N') = f(\mathbf{b}, \mathbf{t}, N)$$
   where $\mathbf{r}'$ is its response vector.
   c) For every area A in $\{X, Y, Z\}$, A replaces: $N \leftarrow N'$ and $\mathbf{r} \leftarrow \mathbf{r}'$.

In the remaining discussion, we assume that Y models the entire "brain". If X is a sensory area, $\mathbf{x} \in X$ is always supervised. The $\mathbf{z} \in Z$ is supervised only when the teacher chooses to. Otherwise, $\mathbf{z}$ gives (predicts) effector output.

Put intuitively, like the "brain", the DN repeatedly predicts the output Z for the next moment. When the predicted Z is mismatched, learning proceeds to learn the new information from Z. But, there is no need to check mismatches: learning takes place anyway.

A generative DN (GDN) automatically generates neurons in the Y area. If $(\mathbf{b}, \mathbf{t})$ is observed for the first time ((the pre-action of the top-winner is not 1) by the area Y, Y adds (e.g.,

equivalent to mitosis and cell death, spine growth and death, and neuronal recruitment) a Y neuron whose synaptic weight vector is $(\mathbf{b}, \mathbf{t})$ with its neuronal age initialized to 1. The idea of adding neurons is similar to ART and Growing Neural Gas but they do not take action as input and are not state-based.

C. Unified DN area function

It is desirable that each "brain" area uses the same area function f, which can develop area specific representation and generate area specific responses. Each area A has a weight vector $\mathbf{v} = (\mathbf{v}_b, \mathbf{v}_t)$. Its pre-response value is:

$$r(\mathbf{v}_b, \mathbf{b}, \mathbf{v}_t, \mathbf{t}) = \dot{\mathbf{v}} \cdot \dot{\mathbf{p}} \tag{1}$$

where $\dot{\mathbf{v}}$ is the unit vector of the normalized synaptic vector $\mathbf{v} = (\dot{\mathbf{v}}_b, \dot{\mathbf{v}}_t)$, and $\dot{\mathbf{p}}$ is the unit vector of the normalized input vector $\mathbf{p} = (\dot{\mathbf{b}}, \dot{\mathbf{t}})$. The inner product measures the degree of match between these two directions $\dot{\mathbf{v}}$ and $\dot{\mathbf{p}}$, because $r(\mathbf{v}_b, \mathbf{b}, \mathbf{v}_t, \mathbf{t}) = \cos(\theta)$ where θ is the angle between two unit vectors $\dot{\mathbf{v}}$ and $\dot{\mathbf{p}}$. This enables a match between two vectors of different magnitudes (e.g., a weight vector from an object viewed indoor to match the same object when it is viewed outdoor). The pre-action value ranges in $[-1, 1]$.

This pre-response is inspired by how each neuron takes many lines of input from bottom-up and top-down sources. It generalizes across contrast (i.e., the length of vectors). It uses inner-product $\dot{\mathbf{v}} \cdot \dot{\mathbf{p}}$ to generalize across many different vectors that are otherwise simply different as with symbols in an FA. The normalization of the first and second terms above is for both the bottom-up source and top-down source to be taken into account, regardless the dimension and magnitude of each source.

To simulate lateral inhibitions (winner-take-all) within each area A, top k winners fire. Considering $k = 1$, the winner neuron j is identified by:

$$j = \arg \max_{1 \le i \le c} r(\mathbf{v}_{bi}, \mathbf{b}, \mathbf{v}_{ti}, \mathbf{t}). \tag{2}$$

The area dynamically scale top-k winners so that the top-k respond with values in $(0, 1]$. For $k = 1$, only the single winner fires with response value $y_j = 1$ (a pike) and all other neurons in A do not fire. The response value y_j approximates the probability for $\dot{\mathbf{p}}$ to fall into the Voronoi region of its $\dot{\mathbf{v}}_j$ where the "nearness" is $r(\mathbf{v}_b, \mathbf{b}, \mathbf{v}_t, \mathbf{t})$.

D. DN learning: Hebbian

All the connections in a DN are learned incrementally based on Hebbian learning — cofiring of the pre-synaptic activity $\dot{\mathbf{p}}$ and the post-synaptic activity y of the firing neuron. If the pre-synaptic end and the post-synaptic end fire together, the synaptic vector of the neuron has a synapse gain $y\dot{p}$. Other non-firing neurons do not modify their memory. When a neuron j fires, its firing age is incremented $n_j \leftarrow n_j + 1$ and then its synapse vector is updated by a Hebbian-like mechanism:

$$\mathbf{v}_j \leftarrow w_1(n_j)\mathbf{v}_j + w_2(n_j) y_j \dot{\mathbf{p}} \tag{3}$$

where $w_2(n_j)$ is the learning rate depending on the firing age (counts) n_j of the neuron j and $w_1(n_j)$ is the retention rate with $w_1(n_j) + w_2(n_j) \equiv 1$. The simplest version of $w_2(n_j)$ is $w_2(n_j) = 1/n_j$ which corresponds to:

$$\mathbf{v}_j^{(i)} = \frac{i-1}{i}\mathbf{v}_j^{(i-1)} + \frac{1}{i} 1 \dot{\mathbf{p}}(t_i), i = 1, 2, ..., n_j,$$

where t_i is the firing time of the post-synaptic neuron j. The above is the recursive way of computing the batch average:

$$\mathbf{v}_j^{(n_j)} = \frac{1}{n_j} \sum_{i=1}^{n_j} \dot{\mathbf{p}}(t_i)$$

where is important for the proof of the optimality of DN in [29].

The initial condition is as follows. The smallest n_j in Eq. (3) is 1 since $n_j = 0$ after initialization. When $n_j = 1$, $\mathbf{v}_j$ on the right side is used for pre-response competition but does not affect $\mathbf{v}_j$ on the left side since $w_1(1) = 1 - 1 = 0$.

A component in the gain vector $y_j \dot{\mathbf{p}}$ is zero if the corresponding component in $\dot{\mathbf{p}}$ is zero. Each component in $\mathbf{v}_j$ so incrementally computed is the estimated probability for the pre-synaptic neuron to fire under the condition that the post-synaptic neuron fires.

E. GDN area functions

For simplicity, let us consider $k = 1$ for top-k competition.

Y area function:

1) Every neuron computes pre-response using Eq. (1).
2) Find the winner neuron j using Eq. (2).
3) If the winner pre-response is not 2, generate a Y neuron using the input $\dot{\mathbf{p}}$ as the weight with age 0. The new Y neuron as it is the winner for sure.
4) The winner neuron j increment its age: $n_j \leftarrow n_j + 1$, fire with $y_j = 1$, and updates its synaptic vector, using Eq. (3).
5) All other neurons do not fire, $y_i = 0$, for all $i \neq j$, and do not advance their ages.

Z Area function: This version has $k = 1$ for top-k competition within each concept zone.

1) If the dimension of Y has not been incremented, do:
   a) Every neuron computes pre-response using Eq. (1).
   b) Find the winner neuron j using Eq. (2).

   Otherwise, do the following:
   a) Supervise the pre-response of every neuron to be 1 or 0 as desired.
   b) Add a dimension for the weight vector of every neuron, initialized to be 0, which may be immediately updated below.
2) Each winner or supervised-to-fire neuron j increment its age: $n_j \leftarrow n_j + 1$, fire with $z_j = 1$, and updates its synaptic vector, using Eq. (3).
3) All other neurons do not fire, $z_i = 0$, for all $i \neq j$, and do not advance their ages.

The Y area function and the Z functions are basically the same. Z can be supervised but Y cannot since it is inside the

closed "skull". During the simple mode of learning discussed here, neurons responding for backgrounds are suppressed (not attending), so that no neurons learn the background.

IV. Developmental Agents

We use our digital game, EpigenBuddy, which is a real time online game as an example to discuss how a Developmental Agent (DA) is designed and trained. In EpigenBuddy, the story occurs in a farm, in which several animals are raised as non-player clients (NPCs, which equal to DAs in EpigenBuddy).

A. Concept Zone Vectorization

In real time online games, the statuses of the game are unpredictable for the exponential growth of states. DN consider projecting all these states into the state space through vectorizing the sensory and motor areas for the following motivations: 1) It dramatically increases the representation power of sensory and motor areas. The sensory and motor areas of DN are two input resources of the network, which can be represented as two vectors X and Z. Suppose that there are n concept zones and each concept zone has k neurons each representing a distinct value of the zone. Then, there are a total of k^n states in the vector. When $k = 4$ and $n = 30$, $k^{30} > 16^{15}$, more than the number of neurons in the human brain. 2) It allows the Y area to develop only neurons (feature clusters) for the typically much lower dimensional manifolds of data actually observed in the $X \times Z$ space. This advantage only greatly reduces the memory space of the engine, but also relieves the human programmer from exhaustively predicting and handcrafting the representations of all states in such a high dimensional input space $X \times Z$. According to LCA(Lobe Component Analysis) [10], the self-organization of the neuronal representation in the $X \times Z$ is dually optimal, optimal in the space (best represent the signal manifolds) and optimal in time (best representation given the limited neuronal resource and the limited amount of training samples).

According to our analysis, the sensory area X can simulate any sensory modality, motor area Z can simulate any actions if only the sensors and motors are sophisticated enough. But in EpigenBuddy, we don't have to simulate every details because the game is not grounded. From the viewpoint of emergent networks [7], each state is composed of several concepts, each concept is a functional activity of the sensors or motors. Therefore, we can represent the inputs or outputs with combination of concepts. Similar functional concepts are grouped together to form a concept zone, the most convenient zone partition method is grouping by part of the body. For the X area which simulates the NPC's sensory, 10 concept zones have been created. They are orientation, distance, vision, audition, touch, role, time, friend or foe, object recognition. Each concept zone has 2 to 5 concepts and the total concepts add up to 30. Take the audition concept zone for example, it includes 3 concepts: nothing, sweet sound, and noisy sound. The concepts in the same concept zone are encoded by the neuron activations. We use canonical encoding in our game, although we can use much less neurons, theoretically, to encode the concepts in one concept zone. This means that each neuron represents a concept in each concept zone. In our digital game, these concepts are rich enough to build a vivid virtual game world. For example, when the NPC finds the food, every concept zone will emerge one concept which represented by an activated neuron to constitute this sensory input. A 30 bits binary array will be took into the algorithm as bottom up input, 1 indicates the neuron fired and the concept emerged. TABLE I lists 5 sensory concept zones.

TABLE I
Partial Concepts of the Sensory Area X

Concept Zone	Concept	
Orientation	Front	Left
	Right	Back
Distance	Near	Middle
	Far	
Vision	Detect object	Detect nothing
Audition	Hear nothing	Hear sweet sound
	Hear noisy sound	
Touch	Touch nothing	Comfortable
	Pain	

It is also impractical and unnecessary to model all the motors in the game for advanced animals have hundreds of muscles, which are responsible for controlling various actions. We divide NPC's body into 11 parts, they are eyes, forehead, mouse, nose, face, left & right ear, left & right upper arm, body with legs and tail. A concept zone is established for each part of the body and the total number of concepts adds up to 106. As well as sensory area, each state of the motor area consists several concepts. A 106 bits binary array is generated by the algorithm at each epoch, where each bit represents a motor concept, 1 indicates to activate the corresponding concept. This array also acts as the top-down input in the following epoch. TABLE II outlines some motor concept zones.

TABLE II
Partial Concept of the Motor Area Z

Concept Zone	Concept		
Eyes	Normal	Smiling	...
	Scared	Depressed	
Mouth	Normal	Chewing	...
	Vomit	Eating	...
	Bite		...
Left & Right Upper arm	Normal	Wave hand	...
	Grab hand	Clapping	...
Body & Legs	Walk	Run	...
	Jump	Rotate	...
Intention	Hungry	Thirsty	...
	Uncertainty	Missing	...

Notice that the "Eyes" concept zone here is totally different from the visual concept zone defined in sensory area. In fact, it

Fig. 2. Game UI for training the NPC. In this figure, the NPC is unhappy and we are teaching it to be friendly. The list on the right side of the scene is all concepts of the Eye concept zone.

is a concept zone of motor area to control the facial expression. We explicitly define intention here as one concept zone for better game experience. For instance, a bubble containing food popping on the head of NPC means the NPC is hungry.

In summary, there are two conditions for the zoning of concepts. (1) Concept (values) are mutually exclusive within the same concept zone. This means only one concept is activated at a time, which implies there is only one neuron can fire at a time so that top-1 operation is applied in each concept zone to determine the single winner. This condition is useful to re-conditioning the numeric motor vector. (2) Concepts from different zones are allowed to fire simultaneously. This allows the learning for complex (composite) actions.

B. Agent Training

Teaching their NPCs various actions is one main task of the players. From the viewpoint of FA, the training process is the process of building the state transition, where the states emerged dynamically. In EpigenBuddy, the player enters into the training mode when he wants to train one of his NPCs. In the training mode, the NPC's corresponding concepts are listed besides it, the player chooses the concepts he feels appropriate. Fig. 2 shows the game UI of training the NPC to be friendly when the player finds the NPC unhappy.

Having been taught various actions, NPC can automatically finds out how to deal with the situations automatically. Fig. 3 shows 4 different actions of the NPC when interacts with player, which are normal, unhappy, depressed and exciting responses under different conditions.

For convenience, we will train the NPCs some default skills before the release of the game, such as patrolling for guard, dancing to music, and playing for fun.

C. Inconsistent Training

In [29], Weng has proved that in each epoch of the training phase, the synaptic weights of DN are maximum likelihood estimated and error free under the assumption that the training examples are consistent. When we train the NPCs off-line, we don't need to worry much about the consistence of the training examples, they can be checked beforehand. But in EpigenBuddy, chances are high that the players may make mistakes occasionally, they may teach NPC several contradictive reactions under the same condition.

Fortunately, benefitted from the Hebbian learning rule, we find that the DAs have the property of converging to the behaviors which were trained frequently. We give a theoretical proof of this property here.

For convenience, some notations and terminologies are defined first:

1) y_j is the j-th Y neuron, z_k is the k-th Z neuron.
2) $\mathbf{v}_k = (p_1, p_2, \ldots, p_{c(Y)})$ is the weight vector of the z_k.
3) $p_j = \frac{f_{jk}}{n_k}$ is the amnesic average computed from Hebbian learning expression Eq. (3). It is the synaptic weight from y_j to z_k which indicates the frequency at which the y_j triggered z_k when z_k fires.
4) We call $\frac{f_{jk}}{n_k||\mathbf{v}_k||}$ as Z-normalized frequency, where $||\mathbf{v}_k||$ indicates how diverse the Y neurons that trigger the z_k are.

According to the criterions for the concept zone vectorizing, without loss of generality, we assume that the sensory or motor area of our DN has only one concept zone, then we have the following conclusion.

Lemma 1 (Property of Inconsistent Training): Suppose a DN trained by two inconsistent state transitions $s_0 \xrightarrow{\sigma} s_i$, $i = a$ or b, s_0 will transit to the state which has the higher Z-normalized frequency at any time when it receives σ.

Proof: According to the criterions of vectorization and our assumption, only one Z neuron which has larger pre-response value is allowed to fire at any time. By Eq. (1) the pre-response of z_i is:

$$r(\mathbf{v}_i, y) = \dot{\mathbf{v}}_i \cdot \dot{\mathbf{y}} = \frac{p_j}{\|\mathbf{v}_i\|} \cdot 1 = \frac{f_{ji}/n_i}{\|\mathbf{v}_i\|} = \frac{f_{ji}}{n_i\|\mathbf{v}_i\|} \quad (4)$$

which is the Z-normalized frequency of z_i. $\|\mathbf{v}_i\|$ may affect which Z neuron to fire since the larger $\|\mathbf{v}_i\|$, the smaller the pre-response value while other values are the same.

In consistent training, the Z-normalized frequency of z_b is zero since there is no chance to be error if z_a is correct. When we trained the DN with two inconsistent actions $s_0 \xrightarrow{\sigma} s_a$ and $s_0 \xrightarrow{\sigma} s_b$, Z-normalized frequency of z_a and z_b can be computed respectively. Therefore, the Z neuron with the higher Z-normalized frequency fires. This concludes the proof that under inconsistent Z-supervised training, the Z neuron that has the highest Z-normalized frequency fires.

■

Lemma 1 proves that DN will converge to the state with higher Z-normalized frequency. This property guarantees that DN has the capability of rectifying its behaviors in an inconsistent environment which is more close to the reality.

Converge speed is another big issue of our algorithm. In order to check how fast the algorithm to converge, we manually designed a state transition diagram, which contains

Fig. 3. Four different actions when the player interacts with the NPC. (a)Normal: Nothing special. (b)Unhappy: Hungry for long time. (c)Depressed: Punished by the player. (d) Exciting: The Player rewards the NPC food.

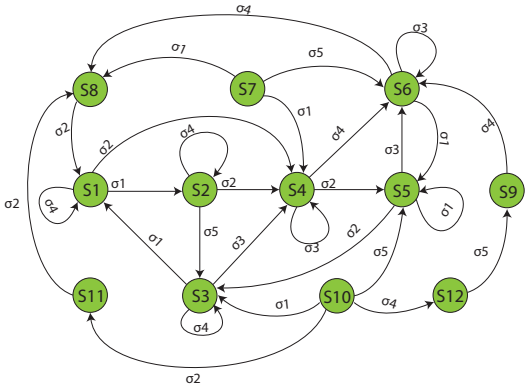

Fig. 4. A state transition diagram representing the correct transition rules. There are 12 states and 5 inputs in the diagram.

12 meaningless states and 5 inputs. Fig. 4 shows this diagram. During the training phase, we were not observing the rules of the transition diagram, and we taught the agent some errors deliberately with a certain probability. For example, when the agent is in state q_1 and receives the input σ_1, it is supposed to transit to q_2, but we teach the agent to transit to q_3 with a probability of 10% on purpose. After training for several epochs, we let the agent run by itself to exam its accuracy. We found out that although the agent was trained inconsistently, it converged to the right behavior finally. Fig. 5 shows the convergence speed under different correct teaching frequency. The X axis represents the epoch number of average training times, Y axis represents the precision. The trend in Fig. 5 is obvious that under 4 different correct teaching frequencies, all the precisions are converged to 100%. And it is also obvious that the speed of convergence depends on the correct teaching frequency, the higher the faster.

V. Conclusion and Future Work

Using DN, we build a kind of general purpose, flexible and fault-tolerable agent in our game EpigenBuddy. DAs increase the efficiency of game design, save the resources, and reduce the costs due to their task non-specific nature. The vector representation addresses the exponential number of problem by allowing the Y vectors to interpolate among

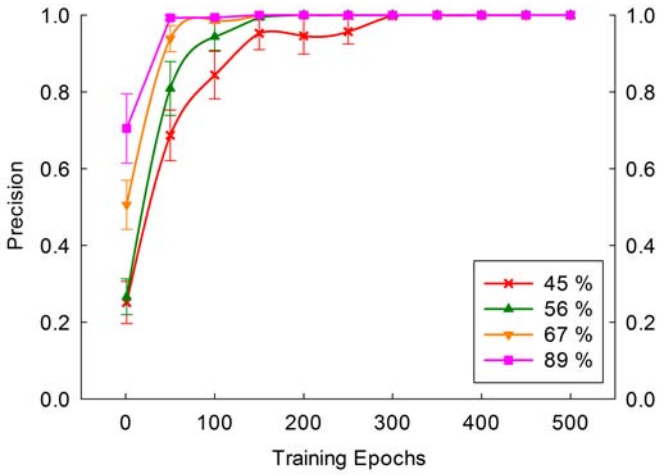

Fig. 5. Convergence speed under different correct teaching frequency.

a very large, potentially unbounded, number of samples in the $X \times Z$ space. We also established that DAs can learn well from inconsistent training but they require more training experience if the training is inconsistent. This property is important especially in real time online games where players may make mistakes.

In our next work for EpigenBuddy, we plan to implement the modulatory system for the DN in our game agents, to allow the agent to autonomously learn (in addition to motor supervised learning) via punishments, rewards, and novelty. The modulatory system is a biological term that accounts for what is called reinforcement learning and instrumental conditioning in psychology and machine learning, but it is more general in the sense that it is also responsible for novelty, uncertainty, and higher emotion.

ACKNOWLEDGMENTS

This project was supported in part by the Changjiang research fund provided by the Ministry of Education of China and Fudan University. The authors would like to thank Matthew Luciw at Michigan State University for providing the source program of WWN-3 to be adapted to the game engine here. We are grateful to Yin Liu for her participation in the design of game scripts, sensory port, the effector port, art resource, and to Zhengzhong Yu for his user interface design and 3D-engine programing. Special thanks to Hao Dong for his management and support for the game group in the Generic Network Inc.

REFERENCES

[1] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433–460, October 1950.

[2] R. A. Brooks, *Cambrian Intelligence: The Early History of the New AI*. Cambridge, Massachusetts: MIT Press, 1999.

[3] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, "Autonomous mental development by robots and animals," *Science*, vol. 291, no. 5504, pp. 599–600, 2001.

[4] F. H. Hsu, "IBM's deep blue chess grandmaster chips," *IEEE Micro*, vol. 19, no. 2, pp. 70–81, 1999.

[5] D. Charles and S. McGlinchey, "The past, present and future of artificial neural networks in digital games," in *Proc. of the 5th international conference on computer games: artificial intelligence, design and education*, 2004, pp. 163–169.

[6] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 123–161, Aug. 2009.

[7] J. Weng, "Symbolic models and emergent models: A review," *IEEE Trans. Autonomous Mental Development*, vol. 3, pp. +1–26, 2012, accepted, available online, and to appear.

[8] D. J. Felleman and D. C. Van Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cerebral Cortex*, vol. 1, pp. 1–47, 1991.

[9] J. Weng, "Three theorems about developmental networks and the proofs," Department of Computer Science, Michigan State University, East Lansing, Michigan, Tech. Rep. MSU-CSE-11-9, May,12 2011.

[10] J. Weng and M. Luciw, "Dually optimal neuronal layers: Lobe component analysis," *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 68–85, 2009.

[11] Z. Ji, M. Luciw, J. Weng, and S. Zeng, "Incremental online object learning in a vehicular radar-vision fusion framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 402–411, 2011.

[12] K. Miyan and J. Weng, "WWN-Text: Cortex-like language acquisition with What and Where," in *Proc. IEEE 9th Int'l Conference on Development and Learning*, Ann Arbor, August 18-21 2010, pp. 280–285.

[13] Y. Zhang and J. Weng, "Task transfer by a developmental robot," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 226–248, 2007.

[14] ——, "Conjunctive visual and auditory development via real-time dialogue," in *Proc. 3rd Int'l Workshop on Epigenetic Robotics (EPIROB'03)*, Boston, Massachusetts, August 4-5 2003, pp. 974 – 980.

[15] Y. Zhang, J. Weng, and W. Hwang, "Auditory learning: A developmental method," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 601–616, 2005.

[16] D. Charles, "Enhancing gameplay: Challenges for artificial intelligence in digital games," in *Proc. of Digital Games Research Conference*, 2003.

[17] S. Woodcock, "Game ai: The state of the industry 2000-2001: Its not just art, its engineering," *Game Developer magazine*, 2001.

[18] L. Galway, D. Charles, and M. Black, "Improvement in game agent control using state-action value scaling," in *Proceedings of 16th European Symposium on Artificial Neural Networks*, 2008, pp. 155–160.

[19] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game ai," in *Proc. AAAI AIIDE'05*, 2005.

[20] S. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 10–18, 2006.

[21] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 123–161, 2008.

[22] D. Charles, C. Fyfe, D. Livingstone, and S. McGlinchey, *Biologically Inspired Artificial Intelligence for Computer Games*. Medical Information Science Reference, 2008.

[23] B. MacNamee and P. Cunningham, "Creating socially interactive non-player characters: The μ-siv system," *International Journal of Intelligent Games and Simulation*, vol. 2, no. 1, pp. 28–35, 2003.

[24] S. McGlinchey, "Learning of ai players from game observation data," in *Proc. of the 4th international conference on intelligent games and simulation*, 2003, pp. 106–110.

[25] G. Yannakakis, J. Levine, J. Hallam, and M. Papageorgiou, "Performance, robustness and effort cost comparison of machine learning mechanisms in flatland," in *Proc. of the 11th Mediterranean Conference on Control and Automation*, 2003.

[26] D. Livingstone and D. Charles, "Intelligent interfaces for digital games," in *Proc. of the AAAI-04 Workshop on Challenges in Game Artificial Intelligence*, 2004, pp. 6–10.

[27] Y. Chang, R. Maheswaran, T. Levinboim, and V. Rajan, "Learning and evaluating human-like npc behaviors in dynamic games," in *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[28] J. Weng, "A 5-chunk developmental brain-mind network model for multiple events in complex backgrounds," in *Proc. Int'l Joint Conf. Neural Networks*, Barcelona, Spain, July 18-23 2010, pp. 1–8.

[29] ——, "Three theorems: Brain-like networks logically reason and optimally generalize," in *Proc. Int'l Joint Conference on Neural Networks*, San Jose, CA, July 31 - August 5 2011, pp. 2983–2990.