2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Deep Boltzmann Machine for Evolutionary Agents of Mario AI

Handa Hisashi

Abstract—**Deep Learning has attracted much attention recently since it can extract features taking account into the high-order knowledge. In this paper, we examine the Deep Boltzmann Machines for scene information of the Mario AI Championship. That is, the proposed method is composed of two parts: the DBM and a recurrent neural network. The DBM extracts features behind perceptual scene information, and it learns off-line. On the other hand, the recurrent neural network utilizes features to decide actions of the Mario AI agents, and it learns on-line by using Particle Swarm Optimization. Experimental results show the effectiveness of the proposed method.**

I. Introduction

THIS paper investigates the nature of Deep Boltzmann Machines [1][2] which applies inputs of evolutionary learning, i.e., scenes in the Mario AI Championships [3][4]. Deep Learning including the Deep Boltzmann Machines can extract higher-level features from learning data, by using a number of layers of Neural Networks. Such higher-level features are different with conventional features such as edge detection. The features extracted by the Deep Learning methods have higher-order information taking account into not only the local relationships in an image but also the relationships among low-level features. The DBM are usually used as feature extractors. The DBM show the distinguished performances on benchmark problems of speech recognitions and image recognitions [1][2]. In the case of Mario AI, scene information observed in each time step might have similar property of the images. This is our motivation of applying the DBM to the Mario AI Championships.

In this paper, we introduce two-staged method for constituting Mario AI agents: First, a large number of scenes are collected in advance by using other agents, i.e., heuristic agents, agents evolved in easier difficulties. Collected scenes are used as learning data of the DBM. Off-line learning of the DBM is carried out before the following second stage. The learning at the second stage is for acquiring proper rules of input-actions, meanwhile the one at the first stage is for the dimensionality reduction of scene inputs in video games. That is, in the second-stage learning, game plays are iterated during evolution. At each time step in the game plays, agents observe the scene information, and must decide their actions. In the proposed approach, the agents don't use the scene information directly: The observed scene is transferred into a feature vector by using the DBM learned in the first stage. Therefore, the evolutionary agents decide their action by using the feature vector, instead of the scene information. This decision is carried out a recurrent neural network as known as Elman network [5]. Hence, evolutionary algorithms try to improve the score of plays by changing the weight vector of the recurrent neural networks.

Related works are summarized: "Curse of dimensionality" [6] is also famous in reinforcement learning community so that many researches are examined a sort of the dimensionality reduction techniques [7][8]. On the other hand, in the case of the learning classifier systems [9] and the neruroevolution [10], it seems that to reduce such higher-dimensionality does not attract much attention since the LCS and the neuroevolution are expected to learn not only behavior rules but also the feature extraction if in higher-dimensional case. The conventional evolutionary approaches for Mario AI Championships also constitute the agents without the dimensionality reduction techniques or the feature extraction methods. [11][12]. In our previous study, the Manifold Leaning methods, a sort of non-linear dimensionality reduction techniques, are used for evolutionary learning [3][13].

Organization of this paper is summarized as follows: The next section introduces the Mario AI Championship by referring to game rules, and the preprocessing of scenes. Section 3 reminds you of the Deep Boltzmann Machine, including how to cope with the scene information in Mario AI. Section 4 introduces a proposed framework. Experimental results are shown in Section 5.

II. Mario AI Championship

A. Overview

This paper employs the software library provided by the competition organizers [3][4]. The Mario AI Championship is one of competitions held in IEEE CIS conferences and symposiums. There are a few tracks in the Mario AI Championship: constituting Non Player Characters (NPC), leaning of NPC, level generation, and Turing test track.

In the software library, we can assign two parameters to decide levels to play: the difficulty and the random seed. Hence, we can examine various levels with the almost same difficulty, and we can iterate the identically same level for learning by assigning the same parameters.

The software library provides us the simulated environments. At each time step, the learning agents should perceive inputs, and take actions: The learning agents can perceive their states at surrounding cells such that the size of cells is 22x22. Moreover, the positions of enemies, MarioOnGround, MarioAbleToJump, and so on, can be observed. For actions, there are five buttons: LEFT, RIGHT,

H. Handa is with Kindai University, Kowakae 3-4-1, Higashi-Osaka 577-8502, JAPAN (corresponding author to provide phone: +81 6 6721 2332; fax: +81 6 6727 2024; e-mail: handa@ info.kindai.ac.jp).

This work was partially supported by the Grant-in-Aid for Young Scientists (B) of MEXT, Japan (21700254, 23700267), and by Grant-in-Aid for Scientific Research (B) of MEXT, Japan (26330291).

DOWN, JUMP, and SPEED. The LEFT and RIGHT buttons are used for deciding the direction that the agents should go. The agents crouch if the DOWN button is pressed. The agents can jump if the JUMP button is pressed and the flag MarioOnGround is true. The height of the jump is decided by the duration time of which the JUMP button is pressed and the flag MarioAbleToJump is true. This means, the agents cannot jump without any limitation. The SPEED button can change the velocity of the agents. The distance of jumps is affected by the velocity of the agents, i.e., faster agents can jump over bigger holes and obstacles.

The goal of the agents is located at the most right area in the levels. The game is over if the agents are caught by enemies, the agents run out time for play, and the agents fall into holes. In this paper, the settings of the simulator are changed such that the agents are of invulnerable mode such that the agents are not killed even if any enemies catch the agents. The reason of the settings is that the main purpose of this study is to investigate the effectiveness of the Deep Boltzmann Machines for scene information.

B. Scene Information and its Preprocessing

Fig. 1. (a) depicts the original perceptual inputs of the agents: As mentioned in the previous subsection, the size of the inputs is 22x22. The center cell filled in the light-blue color indicates the position of the agents. In the orginal perceptual inputs, the agents located only in the center. That is, other cells are moved if the agents are moved. The cells with green color and yellow ocher mean the obstacles, and the ground in the level, respectively. The blank cells below the ground denote areas which are not shown in the game screen. The second, third, and fourth cells from the right, there is no ground (cells with yellow ocher). They represents there is a hole.

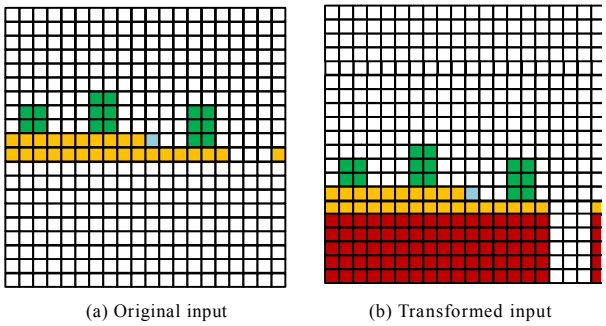
As preprocessing for the original perceptual input, the scene information is transformed: (1) the cell information, and (2) the absolute coordination systems. We have several kinds of grounds, obstacles, and blocks. All the grounds and obstacles are transformed a unified type, e.g., a ground cell. Similarly, all the blocks are also transformed a unified type: a block cell. Therefore, any cell can take either of three values, ground, block, or blank.

The second transformation is of the coordination systems: As depicted in Fig. 1. (b), the vertical position of the bottom of the game screen is fixed: the 6th cell from the bottom cell of the inputs. The blank cells from the 1st cell to the 5th cell are changed into the cell at the 6th cell. Therefore, in the case of holes which should have the blank cell at the 6th cell, all the cells from the 1st cell to the 5th cell are set to be blank. The brown cells in Fig. 1. (b) are newly changed from the blank cell to ground cell. Note that the cells in the hole remain as the blank cell.

III. Deep Boltzmann Machines

A. Restricted Boltzmann Machines

Restricted Boltzmann Machine (RBM) is a two-layered network as shown in Fig.2: the input layer v and the hidden layer h. Each neuron in the input layer is connected with all the neurons in the hidden layer, vice versa, each neuron in the hidden layer is connected with all the neuon in the input layer. However, there is no connection between neurons in the same layer.



(a) Original input (b) Transformed input

Fig. 1. Preprocess of scenes.

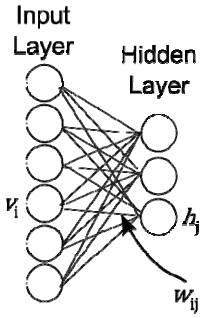


Fig. 2. A Depiction of Restricted Boltzmann Machine.

Energy function for this RBM can be written as follows:

$$\mathrm{E}(\mathrm{v},\mathrm{h}) = -\sum_{\mathrm{i}=1}^{\mathrm{n}}\sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{ij}}\mathrm{h}_{\mathrm{i}}\mathrm{v}_{\mathrm{j}} - \sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{b}_{\mathrm{j}}\mathrm{v}_{\mathrm{j}} - \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{c}_{\mathrm{i}}\,\mathrm{h}_{\mathrm{i}},$$

where v_j and h_i denote the j^{th} neuron in the input layer and the i^{th} neuron in the hidden layer, respectively. w_{ij}, b_j, and c_i indicate a weight value, a bias term for the input layer, and a bias term for the hidden layer, repectively. m and n mean the nymber of neurons in the input layer and the hidden layer, respectively. By using this energy function, the joint probability $\mathrm{p}(\boldsymbol{v},\boldsymbol{h})$ can be represented by the following equation:

$$\mathrm{p}(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{z}\exp\big(-\mathrm{E}(\boldsymbol{v},\boldsymbol{h})\big),$$

where z denotes a normalization term represented by

$$z = \sum_{\boldsymbol{h}} \exp\,(-\mathrm{E}(\boldsymbol{v},\boldsymbol{h})).$$

The parameters in $\mathrm{p}(\boldsymbol{v},\boldsymbol{h})$ is iteratively modified such that the following probability distribution $\mathrm{p}(\boldsymbol{v})$ is closed to observation distribution:

$$\mathrm{p}(\boldsymbol{v}) = \sum_{\boldsymbol{h}} \mathrm{p}(\boldsymbol{v},\boldsymbol{h}).$$

This maximum-likelihood estimation is called Constrastive Divergence Learning.

For input $\boldsymbol{v}$, the activation value of the neuron h_i in the hidden layer is calculated as follows:

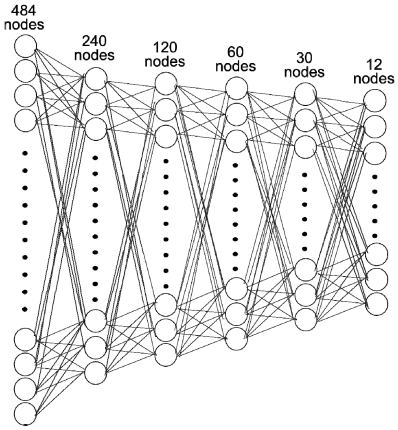


Fig. 3. A Deep Boltzmann Machine used in this paper.

$$\mathrm{p}(h_i = 1|\boldsymbol{v}) = \sigma\left(c_i + \sum_{j=1}^{m} w_{ij} v_j\right), \qquad (1)$$

where σ stands for Sigmoid function.

B. Deep Boltzmann Machines

Deep Boltzmann Machines (DBM) have many layers as delineated in Fig. 3. The DBM in this figure is composed of five RBM. The parameter learning of the DBM is carried out by iterating the learning of the RBM: Suppose that learning data is represented by $\boldsymbol{v}^{(d)}(d = 1, \ldots, D)$, where D represents the number of the learning data. The first RBM in the DBM, i.e., the input layer $\boldsymbol{v}^1$and the hidden layer $\boldsymbol{h}^1$is learned by using learning method in the previous subsection. The learning data $\boldsymbol{v}^{(d)}$ is presented to $\boldsymbol{v}^1$. After learning, the learning data for the succeeding RBM is generated by using equation (1). This procedure is iterated by layer to layer.

C. Application of DBM to the Scene Information of Mario AI

The learning data for the DBM is a large number of scenes which are collected in advance by using other agents. Hence, the other agents examine a large number of levels with the same difficulty with different random seeds. These levels are not used for the evolutionary learning which describes subsequent sections. The number D of the learning data is set to be 10,000. These learning data is used for learning the DBM. In evolutionary learning, the DBM is not learned at all even if unknown scenes are observed. The DBM is only used for generating feature vectors in the Evolutionary Learning phase.

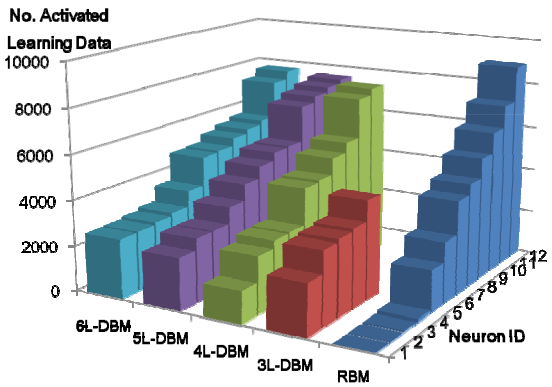


Fig. 4. The number of activated learning data for each neuron: The order of neuron is sorted by the number of activated learning data.

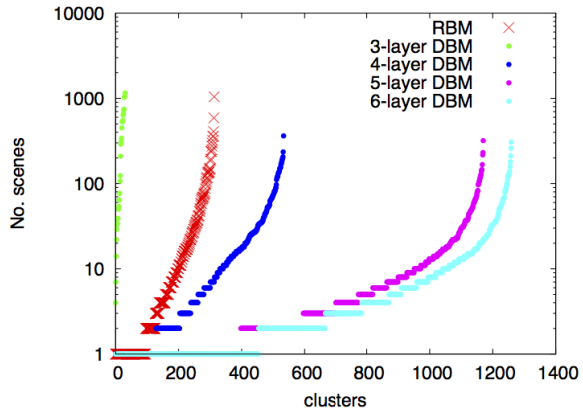


Fig. 5. The number of scenes in clusters, where cluster denotes scenes (learning data) with the same output.

We examine one RBM (484 inputs -12 hidden neurons) and 4 DBM, including Fig. 3. 4 DBM consist of 6-layer DBM (Fig.3), 5-layer, 4-layer, and 3-layer DBM. The number of neurons for each DBM is summarized as follows:

- 6-layer DBM: 484-240-120-60-30-12
- 5-layer DBM: 484-240-120-60-12
- 4-layer DBM: 484-240-120-12
- 3-layer DBM: 484-240-12

10,000 learning data is given to each of the RBM and the four DBM.

Fig. 4. shows the number of activated learning data for each neuron, where the activated learning data means that corresponding learning data activates a certain neuron $\{h_i = 1|\boldsymbol{v}\}$ as in equation (1). The order of neurons is sorted by the number of activated learning data. The neuron ID in Fig. 4. is assigned by this order. There are only 5 bars and 10 bars for 3L-DBM and 4L-DBM, respectively. It means that missing neurons activate all the learning data or none of the learning data. The result of the RBM show there are more and less activated neurons. The results of 6L-DBM and 5L-DBM are similar but the variance of the number of activations of the 6L-DBM is small than the one of the 5L-DBM.

After learning, the value of each neuron for learning data converges to 0 or 1. We regard them either of 0 or 1 by threshold value 0.5. Therefore, each learning data is encoded into a binary vector whose size is 12. Fig. 5. plots the number of learning data per encoded binary vector (cluster). That is, the learning data encoded into the same binary vector are regarded as members in the same cluster. As mentioned in the beginning of this subsection, we have 10,000 learning data. Each algorithms have 12 neurons ($m = 12$). The possible number of clusters is $2^m = 2^{12} = 4096$. The horizontal axis in Fig. 5. denotes clusters' ID. These ID are assigned by the ascending order of clusters' size, where the size means the number of learning data with the same activated binary vector. In the case of the RBM, and the 3L-DBM, there are several big clusters. Meanwhile, 5L-DBM and 6L-DBM have a large number of small clusters. It may imply the classification resolution of the DBM is fine.

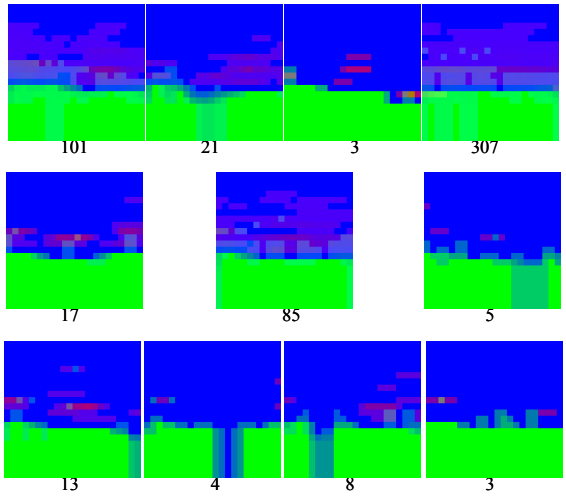


Fig. 6. Averaged images: base image (MIDDLE CENTER); the outputs of DBM for surrounding images are 1-bit different with the one of the base image. Hence, surrounding images have 2-bit different outputs of the DBM with each other.

Fig. 6. shows the various averaged image of the learning data. As mentioned in Section 2, each cell of the learning data can take one of three values (blank, ground, block). For each cell, the ratio of values is calculated, e.g., (0.8, 0.2, 0). Then, the ratio is multiplied by 255: (204, 52, 0). These numbers are regarded as the strength of color (Blue, Green, Red).

The center image in Fig. 6. indicates an averaged image of the learning data which have the same binary output vector. The surrounding images are averaged images of the learning data with a 1-bit different output vector against the center image. Hence, the surrounding images are 2-bit different with each other. Numbers below images mean the number of the learning data to be used for generating corresponding averaged image.

These averaged images have similar tendencies: there are two ground cells from the bottom of screen so that totally seven cells are appeared in the averaged images. The holes are captured in the averaged image with few learning data.

IV. Proposed Method

A. Overview

Fig. 7. depicts the framework of the proposed method. The proposed method consists of two parts: the DBM and recurrent neural networks. The learning of the DBM is carried out in advance. That is, evolutionary algorithms are applied to improve the weights of the recurrent neural networks. The task of the DBM is to extract features from perceptual inputs, i.e., 484 cells' information.

The inputs of the recurrent neural network are composed of the output of the DBM, MarioAbleToJump, MarioOnGround (cf. section 2), and the distance from the ground.

Each agent, i.e., the weight vector of the recurrent neural network, is examined one game play. Fitness function predefined in the software library is used for evaluation of the agent. This fitness function takes account into how close to the goal, how many coins are collected, how much time remains, and so on.

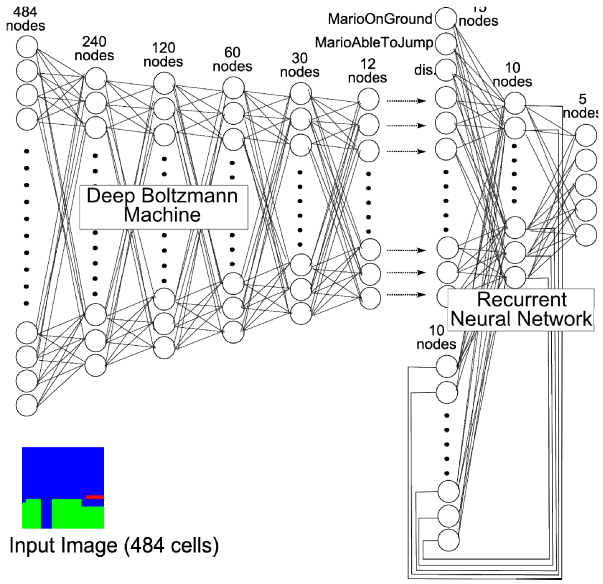


Fig. 7. A depiction of the proposed method.

B. Recurrent Neural Networks

The recurrent neural network used in this paper is shown in the right hand of Fig. 7., which is known as Elman network. There are three layers in the recurrent neural network: Suppose that the neurons in the input layer, in the hidden layer, and the output layer, are represented by $\boldsymbol{I} = (I_1, \ldots, I_{n^I})$, $\boldsymbol{H} = (H_1, \ldots, H_{n^H})$, and $\boldsymbol{O} = (O_1, \ldots, O_{n^O})$, respectively. The neurons at the previous time step in the hidden layer is denoted by $\boldsymbol{H'} = (H'_1, \ldots, H'_{n^H})$. The weights between the input layer and the hidden layer, the hidden layer at the previous time step and the hidden layer, and the hidden layer and the output layer, are represented by w_{ij}^{IH}, $w_{ij}^{H'H}$, and w_{ij}^{HO}, respectively. n^I, n^H, and n^O stand for the number of neurons in the input layer, in the hidden layer, and the output layer, respectively.

The output of the recurrent neural networks is calculated as follows'

$$H_i = \sigma\left(T_i^H + \sum_{j=1}^{n^I} w_{ij}^{IH} I_j + \sum_{j=1}^{n^H} w_{ij}^{H'H} H'_j\right),$$

$$O_i = \sigma\left(T_i^O + \sum_{j=1}^{n^H} w_{ij}^{HO} H_j\right),$$

where T_i^H and T_i^O denote the threshold value for corresponding neuron in the hidden layer, and the output layer. σ is a sigmoid function.

The number of neuron in the output layer n^O is set to be 5. The i^{th} button is pressed if $O_i > 0$. Otherwise, the button is released.

C. Particle Swarm Optimization

As an evolutionary algorithm, we employ Particle Swarm Optimization algorithms (PSO). All the parameters of the recurrent neural networks w_{ij}^{IH}, $w_{ij}^{H'H}$, w_{ij}^{HO}, T_i^H, and T_i^O in the previous subsection are regarded as a vector $\boldsymbol{x}$, i.e., an

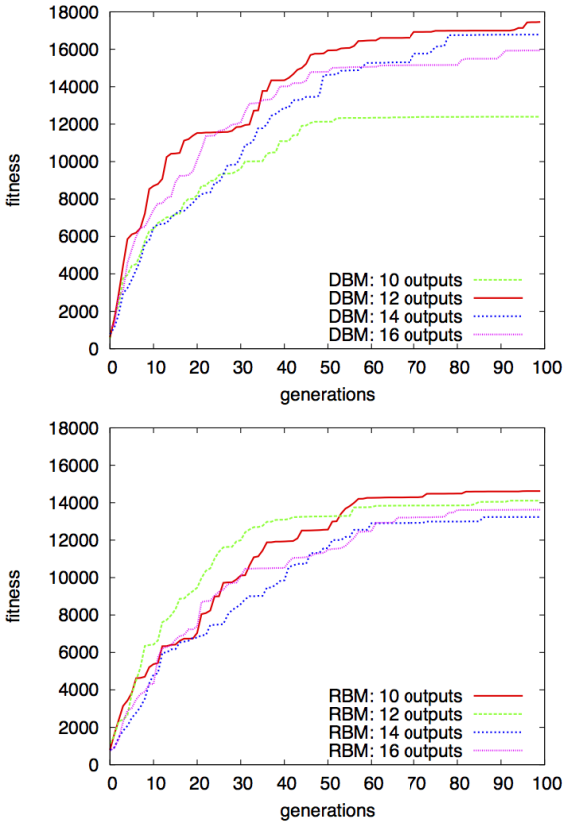


Fig. 8. Temporal changes of best fitness: averaged results over 20 runs; the results of DBM (UPPER) and RBM (LOWER)

individual. This paper employ Clerc's constriction factor method [13][14]:

1. Initialize individuals $\boldsymbol{x}_i$, and their velocity $\boldsymbol{V}_i$
2. Evaluate individuals by playing a game per individual
3. Set the best individual during search, called gbest, and the personal best for each individual, called pbest_i
4. Update the velocities and the individuals:

$$\boldsymbol{V}_i \leftarrow \mathrm{K} * [\boldsymbol{V}_i + \mathrm{c}_1 * \mathrm{rand}() * (\text{pbest}_i - \boldsymbol{x}_i) + \mathrm{c}_2 * \mathrm{rand}() * (\text{gbest} - \boldsymbol{x}_i)]$$

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{V}_i$$

5. Go back to Step 2. until terminal criteria hold.

The parameters c_1, and c_2 are set to be 2.05. The parameter K iscalculated by the following equation:

$$\mathrm{K} = \frac{2}{\left| 2 - \phi - \sqrt{\phi^2 - 4\phi} \right|} = 0.72$$

$$\phi = \mathrm{c}_1 + \mathrm{c}_2$$

V. Experiments

A. Experimental Settings

This subsection describes experimental settings for the following subsections. The population size of the PSO is set to be 100. The number of generations is set to be 100. Hence, totally, 10,000 fitness evaluations are examined. This is in accordance with the setting of the competition.

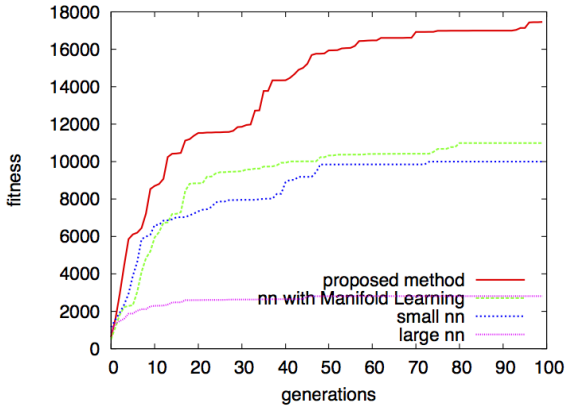


Fig. 9. Temporal changes of best fitness: averaged results over 20 runs; comparison with NN with Manifold Learning, raw input NN (small and large)

The horizontal and the vertical axes of the graphs shown in the following subsections denote the number of generations and the best fitness value averaged over 20 runs. The difficulty of games is set to be 1. It means there are holes in the level. Therefore, agents must learn how to jump over the holes.

B. Effectiveness of "Deep" Learning

This subsection examines the 6-layers DBM and the RBM introduced in Section 3. The subsequent part of the proposed method, i.e., is the recurrent neural networks presented in Section 4. The number of neurons in the input layer for the DBM and the RBM are the same: 484. The number of hidden neurons in the hidden layer for the last layer of the DBM and the RBM are set to be either of 10, 12, 14, and 16. In accordance with the number of the hidden neurons, the number of input neurons of the recurrent neural networks is varied. Fig. 8. shows the experimental results of the proposed method with the DBM (UPPER) and with the RBM (LOWER).

The DBM with 12 neurons outperforms other settings. The difference between the RBM and the DBM is only features extracted if the number of neurons is the same. To introduce "deepness" might bring in good effects for the evolutionary learning of the recurrent neural networks.

Fig. 5. told up there are more 1,200 clusters in the DBM with 12 neurons. It is greater than the capacity of the DBM with 10 neurons (at most 1,024 clusters). It would be a reason why the DBM with 10 neurons did not show better performance.

C. Comparison with other methods

This section compares the proposed method with evolutionary learning with the Manifold learning (nn with Manifold Learning) [4], and two neural networks with raw inputs [11]. Two neural networks are composed of small NN and large NN. The small NN has the input of the state 9 cells surrounding the agents. On the other hand, the large NN has 49 cells surrounding the agents. In the case of nn with Manifold Learning, Isomap is used as a Manifold Learning

method [16]. As in the proposed method, Elman Network is used as neural networks in these algorithms.

Fig. 9. shows the experimental results. Note that the performance of the small NN is better than the one of the large NN. This implies further addition of input cells may not work. In paper [4], the difficulty of levels is set to be 0 while it is set to be 1 in this paper. Main difference of the difficulty 0 and 1 is the existence of holes. Hence, the agent by nn with Manifold Learning could not cope with holes. However, the agent with the proposed method jumps over such holes.

VI. Conclusion

In this paper, the Deep Boltzmann Machine is applied into the scene information in Mario AI Championship. By using features extracted by the Deep Boltzmann Machine, subsequent neuroevolution worked well. That is, by using the Deep Boltzmann Machine all the scene information, i.e., 22x22 cells can be utilized. Our previous method, i.e., nn with Manifold Learning could not apply games such that the difficulty of levels is 1, where holes exist. Meanwhile, the proposed method, i.e., Neural Networks with the Deep Boltzmann Machine worked well. This is one of contributions of this paper.

In section 3, the sizes of clusters by the Restricted Boltzmann Machine and the Deep Boltzmann Machine are compared. "Deeper" Boltzmann Machine had more clusters. From Fig. 7, similar learning data are classified into the same clusters. Some exceptional learning data are also clustered even if cluster's size is small.

Future work is summarized as follows. Recently, there are a large number of Deep Learning methods are proposed so that we would like to examine them for the proposed method. Neuroevolution for Deep Learning might be promising research area for evolutionary computation community.

References

[1] R. Salakhutdinov, and G. E. Hinton, "Deep Boltzmann machines", Proc. JMLR Workshop and Conference Proceedings: AISTATS 2009, vol. 5, 2009, pp. 448–455.

[2] A. Fischer, and C. Igel, "An Introduction to Restricted Boltzmann Machines," Proc. CIARP 2012, LNCS 7441, 2012, pp. 14–36.

[3] S. Karakovsky, and J. Togelius, "Mario AI Benchmark and Competitions", IEEE Trans. Computational Intelligence and AI in Games, Vol.4, No.1, pp. 55–67 (2012)

[4] H. Handa, "Neuroevolution with manifold learning for playing Mario", International Journal of Bio-Inspired Computation, Vol. 4, No.1, 2012, pp. 14–26.

[5] J. L. Elman, "Finding structure in time", Cognitive Science, Vol. 14, 1990, pp. 179–211.

[6] R. Bellman, "Dynamic Programming", Dover Publications, reprint edition, 2003.

[7] T. Tateyama, S. Kawata, and T. Oguchi, "A teaching method using a self-organizing map for reinforcement learning", Artificial Life and Robotics, Vol. 7, No. 4, 2006, pp. 193–197.

[8] H. Handa, A. Ninomiya, T. Horiuchi, T. Konishi, and M. Baba, "Adaptive State Construction for Reinforcement Learning and its Application to Robot Navigation Problems", Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference, 2001, pp.1436–1441.

[9] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[10] X. Yao, "Evolving artificial neural networks",

[11] J. Togelius, S. Karakaovskiy, J., Koutnik, and J. Schmidhuber, "Super mario evolution", Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG), 2009, pp. 156–161.

[12] S. Bojarski, and C. B. Congdon, "REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario", Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010), 2010, pp.83–90.

[13] H. Handa, "Experimental Analysis of the Effect of Dimensionality Reduction on Instance-Based Policy Optimization", PRICAI 2010: Trends in Artificial Intelligence, 11th Pacific Rim International Conference on Artificial Intelligence, LNCS6230, 2010, pp.433–444.

[14] R. C. Eberhart, and Y. Shi, "Particle swarm optimization: developments, applications and resources", Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Vol.1, 2001, pp.81–86.

[15] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization", Proceedings of the 1999 IEEE Congress on Evolutionary Computation, Vol.3, 1999, pp.1951–1957.

[16] J. B. Tenenbaum, V. de Sliva, and J. C. Lagford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction", Science, Vol. 290, 2000, pp.2319–2323