Collaborative Learning of Human and Computer: Supervised Actor-Critic based Collaboration Scheme

Ashwin Devanga[1] and Koichiro Yamauchi[2]
[1]*Indian Institute of Technology Guwahati, Guwahati, India*
[2]*Centre of Engineering, Chubu University, Kasugai-shi, Aichi, Japan*

Keywords: Actor-Critic Model, Kernel Machine, Learning on a Budget, Super Neural Network, Colbagging, Supervised Learning, Reinforcement Learning, Collaborative Learning Scheme between Human and Learning Machine.

Abstract: Recent large-scale neural networks show a high performance to complex recognition tasks but to get such ability, it needs a huge number of learning samples and iterations to optimize it's internal parameters. However, under unknown environments, learning samples do not exist. In this paper, we aim to overcome this problem and help improve the learning capability of the system by sharing data between multiple systems. To accelerate the optimization speed, the novel system forms a collaboration with human and reinforcement learning neural network and for data sharing between systems to develop a super neural network.

1 INTRODUCTION

During recent years, high performance computers , which we could never have imagined before, have been developed. Recent large scale neural networks and its machine learning methods rely on this computational ability.

One drawback of the machine learning methods for neural networks is that they require a huge number of learning samples, which is usually more than the number of internal parameters used in the learning machine. If the problem domain is unknown new field, we cannot collect such large number of samples in advance. Moreover, the learning method to optimize these parameters usually needs a large number of repeats to reach the optimal parameter values. One solution to solve this problem is using the reinforcement learning. However, the reinforcement learning also wastes huge number of try-and-error testing to get appropriate action (or label) for each situation.

Another possibility to solve this problem is using the crowd sourcing. In the crowd sourcing, many workers in the cyberspace collaborate to solve such open problems and yield many solution candidates(e.g. (Konwar et al., 2015)). Although the crowd sourcing techniques are able to collect many number of solution candidates, the architecture is not designed to get the best function for solving the problems. We have developed a solution to this problem. In the system, there is an online learning machine beside of each worker (Ogiso et al., 2016). The learning machine learns corresponding worker actions to imitate them. The learning machine's outputs are integrated by calculating weighted sum. The weights are determined by means of performances of workers. By using such architecture, we can extract the function of each worker by using the online learning machines. Note that even if each worker is absent, the learning machine substitute the absent workers. Moreover, the integrated solution is fed back to each worker to make them generate better solution. By using this mechanism, workers grow smarter.

The literature (Fernàndez Anta et al., 2015) presents a thorough review of the research related to this field and a mathematical analysis of modern crowd-sourcing methods. Similar to our approach, crowd-sourcing methods try to construct a robust protocol to tackle incorrect solution candidates. The classical models answer the majority of the solution candidates to realize robustness (ANTA and LUIS LOPEZ, 2010) (Konwar et al., 2015). Although these approaches are similar to ours, these systems do not provide feedback about the integrated solution to each worker. On the other hand, crowd-sourcing systems based on the game theoretic approach partially provide feedback about the output of some of the workers together with their inputs to each crowd worker (Golle and Mironov, 2001) (Forges, 1986). These approaches provide feedback regarding the situation of each worker and the evaluation results, which is a reward, to each worker. These approaches are similar to our new proposed method that improves its ability by using a reinforcement learning manner. However, these models do not provide feedback about the integrated solution to each worker. In contrast, our proposed new model provides feedback as to the integrated output to each worker to guide how to revise his/her answer.

This study improves the previous study (Ogiso et al., 2016) by replacing each learning machine with a supervised actor-critic method (Rosenstein and Barto, 2012). This means that each learning machine also has the ability to explore new solutions by itself without the help of each worker.

By using this architecture, each worker do not need to manage the system full time because the new model realizes semi-automatic learning. In other words, the new method also explores better solutions without our operations.

The learning machine suitable in this situation is a light weight method. We found that supervised actor-critic model using kernel method for learning is a very light weight learning machine which can handle one pass learning. Using this algorithm with kernel method makes the calculations simpler for a computer to calculate thus increasing efficiency.

Supervised Actor-Critic Model is a state of the art algorithm which runs very light for a reinforcement learning machine. Reinforcement Learning methods are often applied to problems involving sequential dynamics and optimization of a scalar performance objective, with online exploration of the effects of actions. Supervised learning methods, on the other hand, are frequently used for problems involving static input-output mappings and minimization of a vector error signal, with no explicit dependence on how training examples are gathered. The key feature distinguishing Reinforcement Learning and supervised learning is whether training information from the environment serves as an evaluation signal or as an error signal. In this model, both kinds of feedback are available.

Since application of this environment for real world problems would take huge amount of time it was tested on a T-Rex game similar to http://www.trex-game.skipser.com/ which was developed specifically for this project.

In the developed game, the height and width of a part of cactus were modified to make the game player cannot solve them easily without help. Moreover, one another jumping option was added. But these specifications were not announced to the players beforehand. The simulations are explained in section 4.

2 COLLABORATIVE BAGGING SYSTEM USING SUPERVISED ACTOR CRITIC MODEL

Bagging is an old concept of creating a super neural network combining the intelligence of multiple neural networks working on the same problem but different learning and testing scenarios. The idea was that this could save a lot of computational power and time and could also run on simpler hardware such as smart phones or raspberry-pis.

A rough sketch of the ColBagging system is illustrated in Fig 1. The system repeats two phases alternately. The first phase is the training phase, where each worker tries to solve a problem to be solved. Their solutions are emitted by the corresponding online incremental learning machine (MGRNN (Tomandl and Schober, 2001)). At the same time, the performance estimator monitors the solutions from all workers and estimates their quality. This estimation is usually done by the masters or by a pre-determined evaluation function. The performance estimator outputs the results as the weights for the all workers.

This idea was proposed in (Ogiso et al., 2016). It is an improved version of the bagging techniques used before but rather a sophisticated method which calculates weighted averages of the weights of the input neural networks which results in more accurate super neural networks.

In this study, the previous system was improved by introducing one variation of the reinforcement learning method: supervised-actor critic for the learning machine (see Fig 2). By introducing supervised actor-critic method, the solution candidate of each worker will be refined automatically by the exploration done by the reinforcement learning. This means that each worker just need to help the learning machine by teaching action partly. It will not only reduce the work of each worker but also improve the learning speed of each learning machine.

To explain the scheme, the next section explains the supervised actor-critic method used in this system.

3 SUPERVISED ACTOR CRITIC MODEL

Supervised Actor Critic Model (Rosenstein and Barto, 2012) is a variation of reinforcement learning algorithm that introduces human input as the supervised signal. It is well known that the reinforcement learning algorithm can be executed effectively by introducing kernel machines, which add new kernels by

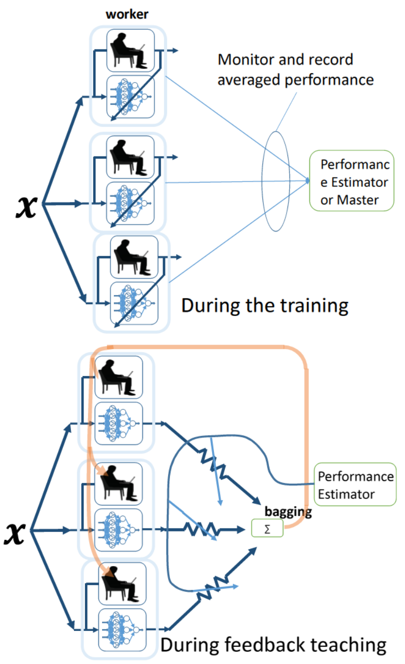

Figure 1: Collaborative Bagging system: The system repeats the training and feedback teaching phases alternately. After the repeats of them several times, the reasoning is done by integrating outputs from the learning machines.

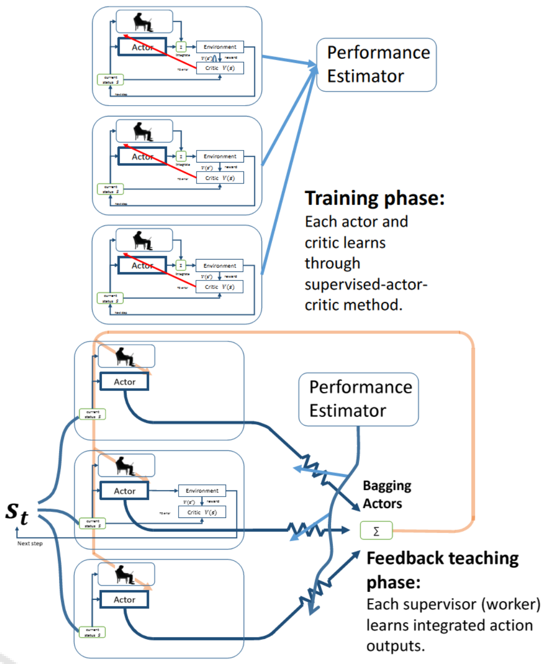

Figure 2: Improved collaborative Bagging system: The previous system was improved by replacing the MGRNN with the supervised actor-critic model.

itself (Xu et al., 2007)[1].

We found that when this model is coupled with Kernel Regression Neural Networks, we get a very efficient learning machine. The supervised nature of the algorithm handles human input and learns from it. This reduces learning time as human input is more filtered than raw data usually used by other reinforcement learning algorithms. The algorithm also uses two Neural Networks, one in Actor and one in Critic. There is an evaluation of results calculated by the actor by the critic. Also since the learning works on the reward system, we get more specific learning. In other words we get a weighted learning system. The added advantage is that there is more data which can be provided to the supervisor for making a choice. Data from both Actor and Critic Neural Network can be supplied to the the user before he makes a call.

3.1 Calculation of TD Error

Referring to Figure 1 and Figure 2, we can understand the entire model. Environment in this case is the T-Rex game in the test case scenario. The state variables of the game are supplied to the actor model and the critic model calculates TD error also known as Temporal-Difference error. Refer to the following equation:

$$\delta = r + \gamma V(s') - V(s) \tag{1}$$

This equation is the definition of TD error. The reward from the action is given by 'r' and '$\gamma \in [0,1]$' is a constant value. The function $V(x)$ is a value function which takes in the state variables as input. In the equation, 'δ' is the current TD error and 's' and 's'' are the current state variables and the future state variables respectively.

3.2 Actor Neural Network

The actor neural network has an output of values only between 0 and 1. This is because the actor is supposed to play the role of the supervisor and the supervisor has to output only whether the T-Rex should jump or not. The ground state is set as 0. The jump state is set as 1. The value output by the Actor is the probability of jump requirement at that position. The input of the neural network is the current state variables, along with the current jump state. The current jump state is used to train the neural net.

The actor uses a kernel perceptron based network.

[1]We do not use the approximated linear dependency based projection method for the kernel manipulation. Instead, the proposed system simply employs the least recently and frequently used estimator for pruning with replacement of kernels as described later. This is for restricting the number of kernels to reduce the computational complexity.

Algorithm 1: Supervised Actor-Critic.

```
Require:
  {critic value function} $V(s)$,
  {Actor policy} $\pi^A(s)$,
  {Exploration size} $\sigma$,
  {actor step size} $\alpha$,
  {critic step size} $\beta$,
  {discount factor} $\gamma \in [0,1]$,
  {eligibility trace decay factor,} $\lambda$
  {Ratio of supervised signal,} $k \in \{1,0\}$
  Initialize $\theta$, $w$ arbitrarily
  for all trials do
    $e = 0$ (clear the eligibility trace)
    $s \leftarrow initialState$
    repeat
      $a^A = \pi^a(s)$
      $a^E = a^A + n$, where $n \sim \mathcal{N}(0,\sigma^2)$
      $a = ka^E + (1-k)a^A$, where $k$ is determined by the supervisor
      TAKE action $a$, observe reward, $r$, and next state $s^{'}$.
      $\delta = r + \gamma V(s^{'}) - V(s)$
      Modify Actor parameters by Algorithm 2.
      Modify Critic parameters by Algorithm 3.
      $s = s^{'}$
    until $s$ is terminal
  end for
```

Therefore, the output from the actor $\pi^a(s)$ is

$$\pi^a(s) = f\left[\sum_{j\in S^A_{t-1}} w^a_j k(s,u_j)\right], \qquad (2)$$

where S^A_t denotes the set of actor kernels after time $t-1$ and $k(s,u_j)$ denotes a Gaussian kernel function:

$$k(s,u_j) \equiv \exp\left(-\frac{\|s-u_j\|^2}{2\sigma^2}\right) \qquad (3)$$

Since the output is just a value between 0 to 1 in (2), we use the sigmoid function: $f[\cdot] \equiv 1/(1+e^{-x})$.

Actor outputs a^E is then calculated as follows:

$$a^E = \pi^A(s) + n, \qquad (4)$$

where n denotes the normal random value $n \sim \mathcal{N}(0,\sigma^2)$. However, if the supervisor yield gives the action a^S, the actor integrates them as follows:

$$a^A = ka^S + (1-k)a^E, \qquad (5)$$

where k denotes the ratio of introducing the supervised signal a^S and we set $k \in \{0,1\}$. Therefore, the parameter k is occasionally switched 1 and 0 by the game player and the mode is changed. The final action is a^A.

During the learning, if there is no activated kernel, the actor allocates a new kernel. Therefore, if $k(s,u_j) < \varepsilon$ for all $j \in S^A_t$, the actor allocates a new kernel $k(\cdot, u_{|S^A_{t-1}|+1})$, where $u_{|S^A_{t-1}|+1} = s$. Note that the set of kernels is updated as $S^A_t = S^A_{t-1} \cup \{t\}$.

However, if such allocation process is continued for ever, the computational complexity is also increased endlessly. Under such situation, the Game system described later slow downed and it is hard to proceed the game. To overcome this difficulty, the actor restricts the size of the actor network. To this end, if $|S^A_{t-1}|$ reaches an upper bound, the actor replaces the most ineffective kernel with the new kernel.

Algorithm 2: Actor learning.

```
Require:
  {Current Status}, $s$
  {TDerror}, $\delta_t$ (1).
  {Set of Kernels(support set)}, $S^A_{t-1}$
  {Distance threshold}, $min_d$
  {parameter vector of $\pi^A(s)$}, $w^A$
  {parameter for LRFU estimation}, $C_j$ where $j \in S_{t-1}$

  $i^* = \arg\min_j \|s-u_j\|^2$.
  $MinDistance = \|s-u_{i^*}\|^2$
  Update $C_j$ for all $j \in S^A_{t-1}$ by (6).
  if $|S^A_{t-1}| < B$ and $MinDistance \geq min_d$ then
    $S^A_t = S^A_{t-1} \cup \{t\}$, where $u_{|S_t|} = s$
    $w^a_{|S_t|} = f^{-1}[a^A]$
  else
    if $MinDistance \geq min_d$ then
      $j^* = \arg\min C_j$ { $j^*$ : Most Ineffective Kernel}
      $u_{j^*} = s$, $w^A_{j^*} = f^{-1}[a^A]$
    else
      Update $w^a$ by using (see (8))
    end if
  end if
  $t = t+1$.
```

Note that the most ineffective kernel is determined by using a least recently and frequently used (LRFU) estimator(Lee et al., 2001), which is a page-replacement algorithm of the virtual memory in operating systems. For the LRFU estimator, each Gaussian kernel has an additional parameter C_j, whose initial value is 1. C_j is updated at each learning step as follows. Therefore,

$$C_j = \begin{cases} C_j + 1 & j = \arg\min_k \|s-u_k\|^2 \\ (1-\varepsilon)C_j & \text{otherwise} \end{cases} \qquad (6)$$

Then, the most ineffective kernel is determined as

$$j^* = \arg\min C_j \qquad (7)$$

Otherwise, the actor modifies existing parameter w^A by (8). The summarized actor learning is shown in Algorithm 2.

$$w^a = w^a + \alpha\left[k\delta(a^E - a^A) + (1-k)(a^S - a^A)\right]\nabla_{w^a}\pi^A(s) \quad (8)$$

3.3 Critic Neural Network

As mentioned in section 3.1, to calculate TD error i.e 'δ' we need to calculate $V(x)$. This value function is being calculated by the critic neural network. It can output a range of values so there is no need of a sigmoid function. We sill directly use the kernel function to calculate the value function.

Therefore, the output from the critic $V(s)$ is

$$V(s) \equiv \sum_{j\in S^C_{t-1}}^{B} w^c_j k(s,u_j) \quad (9)$$

The critic network also learns its output so as to reduce the TD error. If current status s is far from the closest kernel center of the critic, the critic allocates a new kernel, whose parameter is $w^C_{|S^C_t|} = \delta$. However, if current status is closed to the nearest kernel center of the critic, the critic modifies its parameter by the gradient descent method. Therefore,

$$w^C = w^C + \alpha\delta\nabla_{w^C}V(s), \quad (10)$$

where δ is calculated by (1).

Note that in the case of critic, the size of kernel set (support set) S^C_t is not restricted, because $|S^C_t|$ does not become so large.

The summarized critic learning algorithm is shown in Algorithm 3.

Algorithm 3: Critic learning.

```
Require:
  {Current status}, $s$,
  {Next status}, $s'$,
  {Current Action}, $a$,
  {TD error}, $\delta$,
  $j^* = \arg\min_{j\in S^C_{t-1}} \|s - u^c_j\|^2$
  $minDistance = \|X - u^c_{j*}\|^2$
  if $minDistance \geq \varepsilon$ then
    $S^C_t = S^C_{t-1} \cup \{t\}$
    $w^C_{|S^C_t|} = \delta$.
  else
    Update $w^C$ by (10).
  end if
```

3.4 Weight Detection for ColBagging

The training of all supervised actor-critic modules are repeated for several training sessions. Each training session consists of several trials. Each training session is continued until the number of trials reaches a maximum number. After each training session, a weighted majority of all actors is calculated and the integrated actor output is evaluated.

The weight w_a for the a-th actor is detected by

$$w_a = \frac{\exp(\gamma\hat{S}_a)}{\sum_i \exp(\gamma\hat{S}_i)}, \quad (11)$$

where $\hat{S}_a$ denotes the score of the a-th actor of the last training session. $\gamma > 1$ is a coefficient to determine the sensitivity to the score. The final integrated output is

$$O_{integrated} = \sum_a w_a \pi^a(s) + n, \quad (12)$$

where n denotes a noise and $n \sim \mathcal{N}(0,\sigma)$. This noise is added to make a condition the same as the actor-critic learning during $k = 0$ in (5). In the experiment, the score was determined by the maximum number of cactus, which the Dino jumped over successfully. In many cases, w_a is nearly 1 when the a-th agent shows the highest score, otherwise w_a is closed to 0.

4 SIMULATIONS

The application of this learning machine is for electrical bidding but for simulation purposes we had to choose an environment where there were fast changes and quicker data input from the environment. For this purpose we chose the T-Rex game also known as Dino game. It is a small, well known game found in the Google chrome browser. We built a simpler version of this game and used it as an environment for the testing of our learning machine.

The entire learning machine and environment was simulated on JAVA platform. We used Eclipse as the IDE for development. All testing has been conducted on Windows 10 and Ubuntu 16.04.3.

4.1 Game Environment

The learning machine was built interlinked with the environment to handle data transfer between the two easily. Data being transferred between the environment involved just the state variables and reward. The game is very simple. There are just three type of cactus which we must try to avoid by jumping. The speed of the character is also exponentially increased with the following equation:

$$speed = (1+\alpha)\ speed \tag{13}$$

Where α is a constant which is a positive number but very small and in the order of 10^{-4} which keeps the game in a playable situation. If the value is increased the character ends up accelerating too fast which makes it impossible to play.

The initial speed has to be set to a certain value. We chose this value to be 4 as it seemed to be the most practical value for the start of the game. Also the maximum speed of the game was set at 450 as it is practically impossible to play the game at such a high speed. All speeds referred to till now is the speed in the 'X' direction. We also have to maintain the speed in the 'Y' direction. This aspect of the game comes in during jumping. There was gravity implemented into the game as well. To increase the difficulty of the game, we also modified a part of cactus sizes. Therefore, the height of a part of cactus was prolonged to be about two times higher than that of original one. Similarly, the width of a part of cactus was also prolonged as well.

4.2 Learning Machine

The theoretical working of the learning machine was explained in section 3. We implemented the learning module as a thread together with the other Game environment threads. Because we needed to run both the learning machine and game at the same time and for this purpose we used threads. We had two major threads, one for the neural network and the other for the game.

4.3 Approximated Behavior of the Proposed System

The structure of the proposed system is similar to the structure of the particle swarm optimization (Venter and Sobieszczanski-Sobieski, 2003). That is, if the knowledge of the learning device of ColBagging corresponds to the position of the particle, it can be considered that the behavior of each particle in the PSO corresponds to the transition of the knowledge acquired by the learning device of Colbagging. For this reason, first, we approximately simulate how the knowledge of the learner changes, using PSO. Using the PSO, we can visualize the whole status of the learning machines easily.

The PSO has an objective function to be maximized and the particle aims to move toward the position where the objective function is maximized. Normally, each particle moves randomly, but close to its personal best and the global best solutions. In the ColBagging, each worker revises the corresponding particles moving course, according to the workers prior knowledge. We assume that the worker can only recognize the gradient of current position and knows how to move from the position according to the gradient information.

Let $v_i(t)$ and $P_i(t)$ be the velocity and position vectors of the i-th particle at time t, respectively. These two vectors are updated at each step as follows:

$$P_i(t) = P_i(t-1) + v_i(t-1). \tag{14}$$

$$\begin{aligned} v_i(t) &= v_i(t-1) + c_1 r_1 (P_{pbest}(i,t) - P_i(t)) \\ &+ \gamma c_2 r_2 (P_{gbest}(t) - P_i(t)) + \gamma S_i(t), \end{aligned} \tag{15}$$

where $P_{pbest}(i,t)$, $P_{gbest}(t)$ and $S_i(t)$ are the i-the particle's personal-best, the global-best positions and the supervised signal respectively. c_1, c_2 and γ are the fixed positive coefficients. r_1, r_2 are the importance weights, whose values are valid randomly at every time step. Now, let $O(position)$ be the objective function of this PSO. Then, the global best particle is

$$gbest = \arg\max_j O(P_i(t)). \tag{16}$$

The supervised vector is

$$S_i(t) = \nabla_s O(s)|_{s=position_i(t)}\, o(s). \tag{17}$$

We conducted the experiments using this PSO by using following two objective functions. These two functions are defined on a two dimensional space $x = [x_1 x_2]^T$

$$O_1(X) = \frac{\sin(\|X\|)}{\|X\|} \tag{18}$$

$$O_2(X) = \|X\| + 3\cos(\|X\|) + 5 \tag{19}$$

In each objective function, the experiments was done by changing the size of particles:10 and 500. In each experiments, the parameter γ was also valid as 0.1, 0.5 and 1. Note that $\gamma > 0$ means that the PSO is assisted by the supervised signal generated from the gradient signals. The performances were compared with that of the original PSO, where $\gamma = 0$. The performance were measured by the averaged number of particles reached to the optimal point over five trials. In the cases of using (18), the number of reached particles are the number of particles whose value of the objective function is larger than 0.9. On the other hand, in the cases of using (19) , the performance is the averaged number of particles whose value of (19) is less than 8. The results are shown in Figures (5) and (6). From these figure, we can see that, in almost all cases, the particles converged to the optimal solution faster than that of original one when $\gamma > 0$. However, the convergence speed is valid depending

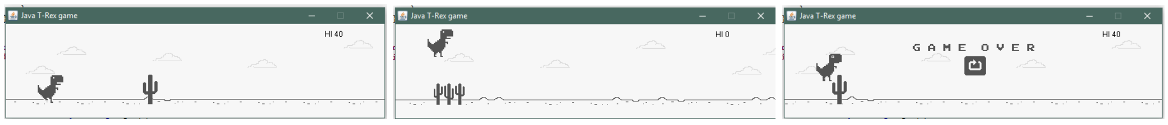

Figure 3: The game in action.

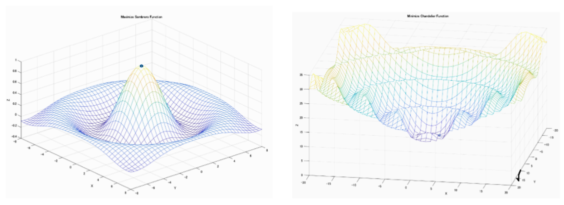

Figure 4: Objective functions left: Sombrero (18) and right: Chandelier (19)) used for PSO experiments.

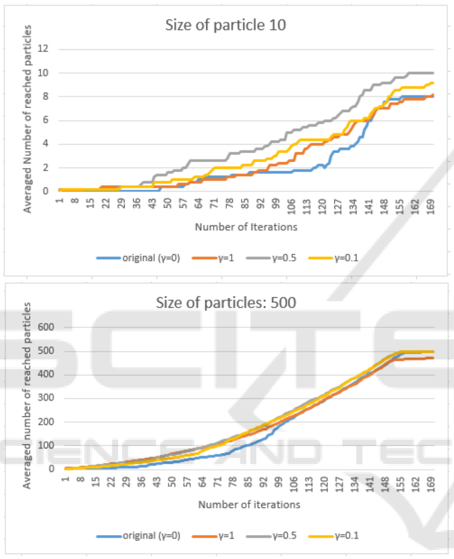

Figure 5: PSO using 10 and 500 particles for the objective function (18).

on the objective function. This is because the supervised signals: the gradient of the objective functions are also valid depending on the shape of the surface of the objective functions.

5 RESULTS

We conducted the test on 16 game players, 10 normal game players and 6 controlled players. The 10 normal players played with the proposed new colbagging system, and the control played on the colbagging without the feedback-teaching process. In the first session, each person played the Dino game normally for 40 rounds. Note that each round is ended when the Dino hits a cactus. In each session, each player can switch between two game modes: the supervised or

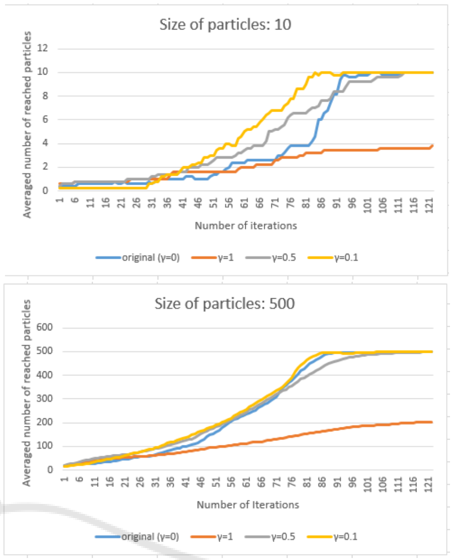

Figure 6: PSO using 10 and 500 particles for the objective function (19).

semi-automatic modes. In the supervised mode, the Dino learns how to play by imitating the playing behaviors of the player. In the semi-automatic mode, the Dino jumps according to the previous learning results. In this mode, player's controls were still available to control the Dino behaviors. After the first session, the performances of each actor were evaluated. The score of each actor and player was determined 20× number of jumped cactus.

Before starting the second session, the 10 normal players watched a stream image of automatic-playing scene of the integrated actors. The colbagging system calculates the weighted sum of all actor's outputs, where the weight for each actor was determined by (11). On the other hand, the controlled player did not watch the integrated actor playing scene before going to the second session. After finishing the second session, each actor's performance was evaluated. The performance was measured by the maximum number of successful jumps over the cactus. The results are plotted in Figure 7 and 8. In each figures, the scores during the learning and test are plotted. The test scores refers the score of actor.

We can see that the performance of each actors were not always reflect the score during the learning. During the training, four normal player's scores after the Second session were larger than those of the first

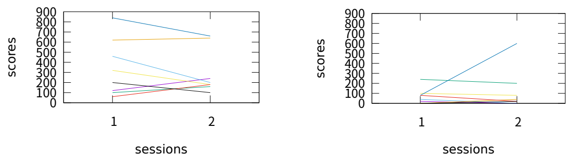

Figure 7: Score transition between the two sessions of 10 normal players: left:During the training, right: Test (Actor's score).

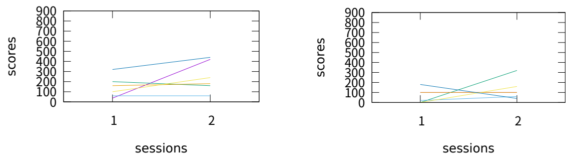

Figure 8: Score transition between the two sessions of 6 controlled players: left: During the training, right: Test (Actor's score).

session. But, five controlled player's scores after the second session were larger than those of the first session.

The performances of the controlled players after the first and second sessions were not changed largely. On the other hand, five normal players increased their score. We predict that this is due to the long waiting time (about two-three hours) between the first and second sessions. Therefore, such long waiting time makes player forget subtle operating patterns. So far, the waiting time is needed to manually create the streaming image of integrated actor.

Moreover, we also found that there were no guarantees that the actor's performance is improved like their players. We also predict that this is due to the actors adjust to the later steps of the game and forgets the early steps of the game environment. We believe that if we reduce the waiting time and modify the hyper parameters for the actor to reduce the interference, the normal-player's performance will be superior to the control players.

The data so far is limited, to provide reliable results. We plan to conduct more tests in future to get enough data.

6 CONCLUSIONS

We proposed our improved collaborative learning scheme between human and machine learning. The system is a variation of crowd sourcing system accelerated by machine learning. The system behavior was roughly simulated by the particle swam optimization accelerated by prior knowledge of each particle. The results suggest that the prior knowledge accelerate the convergence speed. We have also conducted the evaluation of the effectiveness of the collaboration between real human and machine learning by using the Dino game.

ACKNOWLEDGEMENTS

This research has been supported by Grant-in-Aid for Scientic Research(c) 15K00321.

REFERENCES

ANTA, A. F. and LUIS LOPEZ, A. S. (2010). Reliable internet-based master-worker computing in the presence of malicious workers. *Parallel Processing Letters*, 22(1):1250002(17.

Fernàndez Anta, A., Georgiou, C., Mosteiro, M. A., and Pareja, D. (2015). Algorithmic mechanisms for reliable crowdsourcing computation under collusion. *PLoS ONE*, 10(3):1–22.

Forges, F. (1986). An approach to communication equilibria. *ECONOMETRICA*, 54(6):1375–1385.

Golle, P. and Mironov, I. (2001). Uncheatable distributed computations. In *Topics in Cryptology– CT-RSA 2001 Volume 2020 of the series Lecture Notes in Computer Science*, pages 425–440. Springer-Verlag.

Konwar, K. M., Rajasekaran, S., and Shvartsman, A. A. (2015). Robust network supercomputing with unreliable workers. *Journal of Parallel and Distributed Computing*, 75:81–92.

Lee, D., Noh, S., Min, S., Choi, J., Kim, J., Cho, Y., and Sang, K. C. (2001). Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transaction on Computers*, 50(12):1352–1361.

Ogiso, T., Yamauchi, K., Ishii, N., and Suzuki, Y. (2016). Co-learning system for humans and machines using a weighted majority-based method. *International Journal of Hybrid Intelligent Systems*, 13(1):63–76.

Rosenstein, M. T. and Barto, A. G. (2012). *Supervised Actor-Critic Reinforcement Learning*, chapter 14, pages 359–380. Wiley-Blackwell.

Tomandl, D. and Schober, A. (2001). A modified general regression neural network (mgrnn) with a new efficient training algorithm as a robust 'black-box'-tool for data analysis. *Neural Networks*, 14:1023–1034.

Venter, G. and Sobieszczanski-Sobieski, J. (2003). Particle swam optimization. *AIAA*, 41(8):1583–1589.

Xu, X., Hu, D., and Lu, X. (2007). Kernel-based least squares policy iteration for reinforcement learning. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 18(4):973–992.