Team Description of OPU_hana_2D 2007

Tomoharu Nakashima and Yukio Shoji

Department of Computer Science and Intelligent Systems
Graduate School of Engineering
Osaka Prefecture University
Gakuen-cho 1-1, Sakai, Osaka, 599-8531
nakashi@cs.osakafu-u.ac.jp
shoji@ci.cs.osakafu-u.ac.jp

Abstract. This paper describes OPU_hana_2D , our soccer team that has been submitted to the qualification for the competition in the simulation league of RoboCup 2007. The basic skills such as dribble, pass, and shot are improved by modifying the source code of the base team. The main feature of OPU_hana_2D is the introduction of neural networks for dribbling. We call this *neuro-dribble.* In the neuro-dribble, the output of neural network shows the dribble direction. A training data set for the learning of the neural network is obtained from the obserbation of the behaviour of other teams. The coach is also modified from the base source codes so that a heterogeneous player type of each player is determined by a rank-based weighted sum method.

1 Introduction

Team 'hana' has been participating in the RoboCup world competitions since 2002. Our first trial in 2002 ended in losing in all matches without marking any scores. In the second trial in 2003, we survived the first elimination match before losing in the second elimination. We also could proceed to the second round in 2004. As a base team, we used YowAI team in 2002, and have used UvA Trilearn basic [1] since 2002. The team name was changed from 'hana' to 'OPU_hana_2D' in the 2005 competition.

There are three main characteristic features in OPU_hana_2D : Acquiring dribbling skill by neural networks, team strategies obtained by evolutionary computation, and the hetero-selection policy. We will explain each of them in the following subsections.

2 Neuro-Dribble

2.1 Neural Network

In this paper we use neural networks for mimicking the behavior of a target agent. We specifically focus on the dribble skill of the target agent. Thus the task of the neural networks is to learn the sensor-action mapping of the target

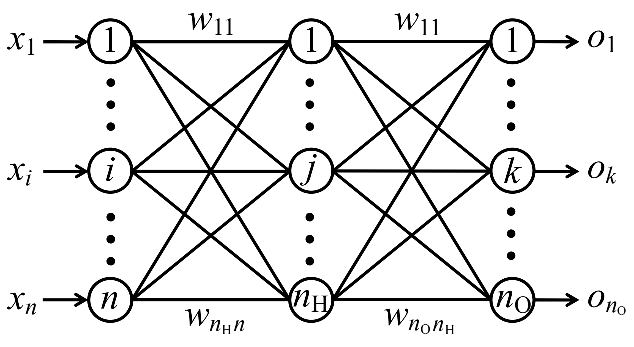

Fig. 1. Feed forward neural network.

agent. We use a standard three-layered feed-forward neural network as shown in Fig. 1.

In Fig. 1, there are n units in the input layer, n_H units in the hidden layer, and n_O units in the output layer. The symbols $n_{n_H n}$ and $n_{n_O n_H}$ show the connection weight between the n_H-th hidden unit and the n-th input unit and between the n_O-th output unit and the n_H-th hidden unit, respectively. It is also assumed that each unit in the hidden layer and the output layer has a threshold value (θ_j for the hidden layer and θ_k for the output layer). The input-output mapping of the neural network with an n-dimensional input vector in Fig. 1 is shown as follows:

Input layer:

$$o_i = x_i, \quad i = 1, 2, \ldots, n, \tag{1}$$

Hidden layer:

$$o_j = f(net_j) = \frac{1}{1 + \exp(net_j)}, \quad j = 1, 2, \ldots, n_H, \tag{2}$$

$$net_j = \sum_i^n w_{ji} o_i + \theta_j, \tag{3}$$

Output layer:

$$o_k = f(net_k) = \frac{1}{1 + \exp(net_k)}, \quad k = 1, 2, \ldots, n_O, \tag{4}$$

$$net_k = \sum_j^{n_H} w_{kj} o_j + \theta_k, \tag{5}$$

Error back-propagation algorithm is used to learn the weights of the neural network. The next subsection describes the extraction of a set of training data for the learning of the neural network.

2.2 Generating Training Data

The task of the neural network is to learn the sensor-action mapping of a target agent. We use log files of soccer games that record the position and the velocity of all objects in the soccer field. Soccer games are performed several times using the target agent in order to obtain log files. Since the task in this paper is to mimic the dribble behavior of the target agent, we extract necessary parts from the whole log files so that the target agent is dribbling in the extracted parts of the log files. This process is manually conducted. From the extracted part, we generate a set of training patterns for the learning of the neural network.

2.3 Implementation

In this implementation, we use three neural networks as there are three available actions for a soccer agent: kick, dash, and turn. The parameters of each action are determined by the corresponding neural network. We refer to the three neural network as kick-, dash-, and turn-neural network, respectively. In Fig. 2 we illustrate the neural networks for the implementation.

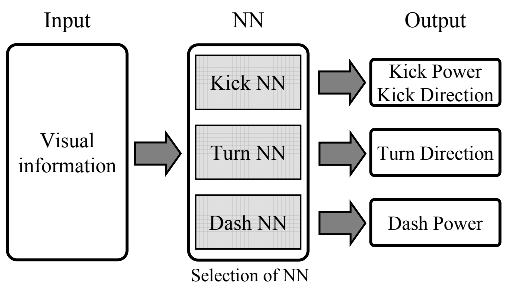

Fig. 2. Implementation.

It is assumed in this implementation that which action is taken according to the agent's situation. We show the procedure of Implementation I as follows:

[Procedure]

Step 1: Stop the ball by applying a negative force to the ball.
Step 2: Determine the dribble direction by the turn-neural network. Send a turn command with the specified direction to the soccer server.
Step 3: If the ball is in front of the ball, determine the dash power by the dash-neural network and send a dash command with the specified power to the soccer server. Otherwise determine the kick power and the kick direction by the kick-neural network and send a kick command with the specified kick power and direction to the soccer server.

Step 4: Stop the dribble procedure if a pre-specified termination condition is satisfied. Otherwise go to Step 3.

2.4 Experimental Settings

This section shows the computational experiments on the mimicking behavior of an agent. In the computational experiments the task of the mimicking agent is to capture the dribbling behavior of a target agent by using neural networks. As a target agent a forward player of STEP is used. STEP won the RoboCup world competition in 2004. The main characteristic feature of STEP is the implementation of their sophisticated dribble skill.

We specify the learning rate of the neural networks as 0.1. We reduced the amount of noise in the soccer field to zero in order not to cause any effect on the performance of the neural networks. In the computational experiments all the objects in the soccer field are observable to the mimicking agent. That is, the mimicking agent has all the necessary information to perform the dribble behavior.

The mimicking agent starts the dribble behavior if the ball is kickable and the consequent action of the applied action rule is to dribble. We also specified five termination conditions for the dribble as follows:

(1) There are any opponent agents that are very close to the mimicking agent,
(2) The ball is taken by an opponent player,
(3) The remaining stamina value of the mimicking agent is less than a prespecified value,
(4) The position of the ball is in pre-specified subareas (e.g., in the penalty area), and
(5) The mimicking agent has been away from the ball for more than 15 time steps.

2.5 Performance Evaluation

In this subsection we show the experimental results of the neuro-dribble. Three neural networks are trained using a set of training patterns extracted from the game logs. We show the learning curve of the kick neural network in Fig. 3. Figure 3 shows the mean squared error of the kick neural network for the training data set. We can see from Fig. 3 that the error of the kick neural network decreases as the number of epochs increases.

Next we show a sequence of the mimicking agent in Fig. 4. From this figure, we can see that the mimicking agent successfully acquire the dribble behavior of the target agent.

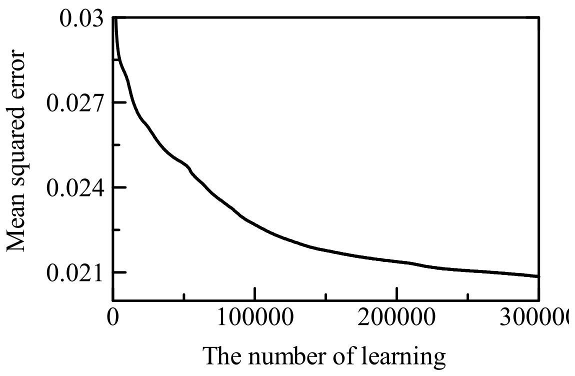

Fig. 3. Learning curve of the neural network.

3 Evolutionary Computation for Team Srategy

The main behavior of players are determined by action rules. The action rules of the following type are used in this paper:

$$\begin{array}{ll} R_j: & \text{If Agent is in Area } A_j \text{ and the nearest opponent is } B_j \\ & \text{then the action is } C_j, \quad j = 1, 2, \ldots, N, \end{array} \tag{6}$$

where R_j is the rule index, A_j is the antecedent integer value, B_j is the antecedent linguistic value, C_j is the consequent action, and N is the number of action rules. Thus the construction of action rules is the crucial part in the development of OPU_hana_2D . In this section, we show an evolutionary approach to the automatic determination of action rules.

3.1 Encoding

As described in the preious section, the action of the agents are specified by the action rules in (6) when they keep the ball. Considering that the soccer field is divided into 48 subfields and the position of the nearest opponent agent (i.e., it is *near* the agent or *not near*) is taken into account in the antecedent part of the action rules, we can see that there are $48 \times 2 = 96$ action rules for each player. In this paper, we apply our evolutionary method to ten soccer agent excluding the goal keeper. Thus, the total number of action rules for a single team is $96 \times 10 = 960$. We use an integer string of length 960 to represent a rule set of action rules for ten players. The task of our proposed evolutionary method is then to evolve the integer strings of length 960 to obtain team strategies with high performance. First 96 integer bits represent the set of action rules for the player 1, the next 96 bits represents the set of action rules for the player 2, and so on. We show in Fig.5 the first 96 bits of an integer string in our evolutionary method. This figure shows an integer string for a single agent. In Fig. 5, the first 48 bits represents the action of an agent when the nearest opponent agent is *near* the agent. On the other hand, the actions of the agent when the nearest opponent agent is *not near* the agent are shown in the other 48 bits. The value of each bit is an integer from the interval $[1, 12]$. This integer value corresponds to the index number of 12 actions that are pre-specified manually. For example, the integers $1 \sim 3$ represents three variations of dribble and the integers $4 \sim 7$ four variations of pass behavior and so on.

3.2 Evaluation of Integer Strings

Generally, the main idea of evolutionary methods is to exploit the information of those individuals whose performance is highly evaluated. In this paper, we evaluate the performance of bit strings through the results of soccer games. Specifically, we use the scores of the soccer games as performance measure in our evolutionary method. We first check the scored goals by the soccer teams that are represented by the bit strings. The more goals a soccer team scores, the higher the performance of the integer string for the soccer team is. When the value of the goals is the same among multiple soccer teams, the lost goals are used as a second performance measure. The soccer teams with lower lost goals are evaluated as better teams. We do not consider the lost goals at all when the goals are different between soccer teams to be evaluated.

3.3 Evolutionary Operation

We use one-point crossover, bit-change mutation, and ES-type selection as evolutionary operations in our evolutionary method. The bit strings are modified through crossover and mutation operations.

In the crossover operation, first we randomly select two integer strings. Then latter part of both strings are exchanged with each other from a randomly selected cut-point. Note that we do not consider any evaluation results when two integer strings for the crossover operation are selected from the current population. In the mutation operation, the value of each integer bit is replaced with an integer value in the interval $[1, 12]$ with a prespecified mutation probability. It is possible that the replaced value is the same as the one before the mutation operation.

Generation update is performed by using our ES-type selection in our method. By iterating the crossover and the mutation operations, we produce the same number of new integer strings as that of the current population. Then the best half integer strings from the merged set of the current and the new strings are chosen as the next generation. The selection is based on the match results as described in Subsection 3.2. This generation update is similar to the $(\mu + \lambda)$-strategy of evolution strategy [2]. Note that the current strings are also evaluated in this selection process. Thus, it is possible that a current integer string with the best performance at the previous generation update is not selected in the next generation update because the performance of the integer string in the next performance evaluation is the worst among the merged strings.

To summarize, our proposed evolutionary method is written as follows:

[Procedure of the proposed evolutionary method]

Step 1: Initialization. A prespecified number of integer strings of length 960 are generated by randomly assigning an integer value from the interval $[1, 12]$ for each bit.

Step 2: Generation of new integer strings. First randomly select two integer strings from the current population. Then the one-point crossover and the bit-change mutation operations are performed to generate new integer strings. This process is iterated until a prespecified number of new bit strings are generated.

Step 3: Performance evaluation. The performance of both the current integer strings and new integer strings generated by Step 2 is evaluated through the results of soccer games. Note that the performance of current integer strings is evaluated every generation because the game results are not constant but different game by game.

Step 4: Generation update. From the merged set of the current integer strings and new ones, select best integer strings according to the performance evaluation in Step 3. The selected bit strings form the next generation.

Step 5: Termination of the procedure. If a prespecified termination conditions are satisfied, stop the procedure. Otherwise go to Step 2.

4 Evolutionary Computation for Heterogeneous Players

4.1 Rank-Based Weighted Sum Method

For determining the type of heterogeneous players, we propose a rank-based weighted sum method. Let us suppose that each player has a weight vector for nine parameters that are changeable by selecting different heterogeneous types. We first transform the values of each parameter into ranks so that the largest value among all the heterogeneous types has the highest rank and the lowest rank is assigned to the smallest value. Each agent evaluates the value of each heterogeneous type by a weighted sum of the rank as follows:

$$V_i(j) = \sum_{k=1}^{9} W_{ik} \times R_{jk}, \quad i = 1, \ldots, 10, \quad j = 0, \ldots, 6, \tag{7}$$

where i is the index of agent, j is the index of the heterogeneous type, $V_i(j)$ is the value of the j-th heterogeneous type for the i-th agent, W_{ik} is the weight of the i-th player for the k-th parameter, and R_{jk} is the rank of the k-th parameter of the j-th heterogeneous type.

The task of the evolutionary computation in this section is to optimize the weight W_{ik} for maximizing the competitive ability of the team.

4.2 Genetic Coding

We represent the weight W_{ik} as a bit string in our evolutionary computation. For simplicity, we only use integer values in the interval $[-3, 3]$ as a value in the bit strings. Since nine bits are necessary for representing the weights for nine heterogeneous parameters for each player, the total length of the bit string for ten players (excluding the goal keeper) is 90.

Generation update is performed in the same way as in the evolutionary computation for team strategy in Section 3.

5 Conclusions

In this team description paper, we explained the three main characteristic features of OPU_hana_2D . One is Neuro-dribble where the task is to mimic the dribble behavior of other teams by using neural networks. The second one is the use of evolutionary computation for obtaining team strategies. The last one is the use of evolutionary computation for acquring the coach strategy of hetero-type selection.

References

1. UvA Trilearn, URL at *http://staff.science.uva.nl/˜jellekok/robocup/index_en.html*
2. Bäck, T.: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY (1996)

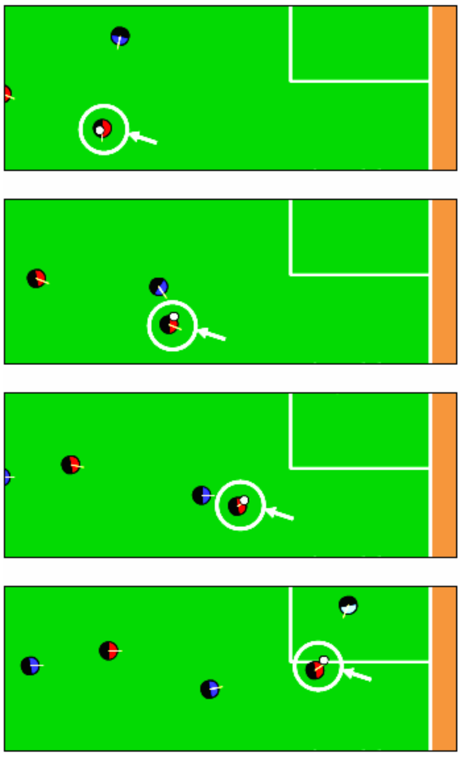
Fig. 4. Action sequence of the mimicking agent.

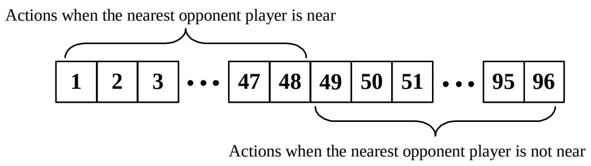
Actions when the nearest opponent player is near
1
2
3
• • •
47
48
49
50
51
• • •
95
96
Actions when the nearest opponent player is not near

Fig. 5. Integer string for a single agent.