Training a Pac-Man Player with Minimum Domain Knowledge and Basic Rationality

Bo Yuan, Cheng Li and Wei Chen

Intelligent Computing Lab, Division of Informatics, Graduate School at Shenzhen,
Tsinghua University, Shenzhen 518055, P.R. China
yuanb@sz.tsinghua.edu.cn, licheng1492@hotmail.com, china_chenwei@hotmail.com

Abstract. In this paper, a Pac-Man player (agent) is trained based on four neural networks. The motivation is to demonstrate how computational intelligence techniques can be used to simulate the self-learning process of human players with minimum prior knowledge about the game and basic rationality in their behaviors. Experimental results show that, on a simplified version of the original Pac-Man game, the agent can achieve reasonable scores after only a handful of trials. This performance is in contrast to existing work on evolving Pac-Man agents where thousands of trials and huge amount of computational efforts are typically required.

Keywords: Pac-Man, Neural Networks, Games, Computational Intelligence

1 Introduction

Pac-Man is an arcade game developed in 1980s by Namco and became immensely popular worldwide since its first release [9, 12]. In nowadays, Pac-Man and its variants can still be found across various platforms such as PCs and hand-held devices and have been rewritten in different programming languages. It features a real-time predator-prey nature where game strategy and reaction speed are both essential. The player controls Pac-Man to navigate through a maze and scores can be achieved by eating dots (foods) along the path. When all dots are cleared, the game progresses into the next stage. Four ghosts are patrolling the maze and once a ghost catches Pac-Man, a life is lost (three lives in total). There are also four power pills that provide Pac-Man with short-term power to eat ghosts as well as bonus items (usually referred to as fruits) appearing in the centre of the maze from time to time. Fig. 1 shows a screenshot of a typical Pac-Man game [11].

Pac-Man is a conceptually and intuitively simple game (e.g., Pac-Man is only controlled by four directions) running in a highly restricted environment (e.g., the scenario of the game is stationary without any new factors). On the other hand, non-trivial game strategies such as real-time path planning and risk/priority management are crucial to players at advanced levels. Fig. 2 shows the average performance of 10 volunteers (female, aged from 20 to 30) over 20 consecutive attempts who have never played Pac-Man before. Note that to speed up the experiment only one life was allowed in each attempt and the game was restarted once a life was lost. In general, it

took human players some time to understand the features of this game and after 10 trials they started to play some serious games and consistently improved their skills through practicing (the best player achieved more than 10,000 points in a single trial). Since the only information given to all players was how to move Pac-Man around, how quickly they can figure out other advanced game features made a difference.

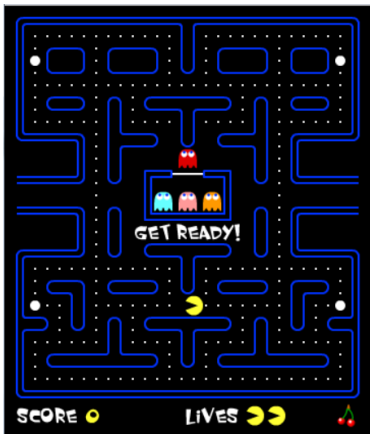

Fig. 1. A screenshot of the starting stage of a typical Pac-Man game with four ghosts and four power pills. Foods are shown as white dots along the path in the maze.

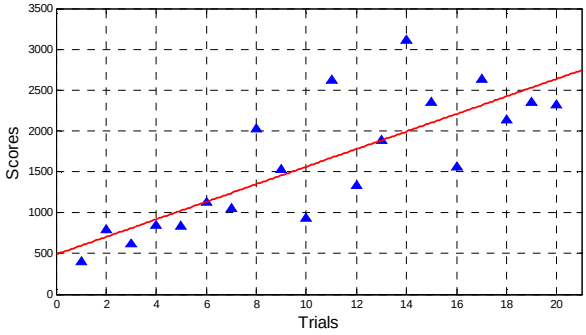

Fig. 2. The average performance of 10 volunteers over 20 consecutive attempts.

Due to the abovementioned features, Pac-Man is a relatively ideal test bed for the application of computational intelligence techniques in computer games. In recent years, a moderate amount of work has been conducted in this area from different angles. Koza [5] investigates the effectiveness of Genetic Programming for task prioritization in the domain of Pac-Man. Lucas [6] uses neural networks as move evaluators to assign scores to each possible movement and Evolution Strategies are used to evolve the connection weights in the networks. The input vector contains the shortest distances from Pac-Man to each ghost, power pill and maze junction etc. Robles and Lucas [7] propose a route-tree approach based on possible moves that the agent can take to depth 40 and use hand-coded heuristics to evaluate each path. Gallagher and Ryan [4] use decision rules to govern the movement of Pac-Man based

on the turn type at its current location and PBIL is used to evolve the weight parameters of the rules. Gallagher and Ledwich [3] demonstrate how neural networks evolved by Evolution Strategies with raw screen data (including three matrices representing walls, dots and ghosts together with four additional inputs to provide dot information beyond the current window) as the representation of the game state can potentially be used to produce a good Pac-Man playing agent. Szita and Lõrincz [8] apply a simple rule-based policy where rules are organized into modules. The policy of the agent is represented as a list of if-then rules with priorities and is optimized by the cross entropy method. Wirth and Gallagher [10] develop a Pac-Man playing agent based on an influence map model with three main parameters. Their results show that these parameters interact in a rather simple way and can be optimized without difficulty. It should be emphasized that different Pac-Man simulators were used in the above work and a direct head-to-head comparison of the performance is not practical.

The work presented in this paper, as well as some of the existing studies, reflect similar philosophy as the work of Chellapilla and Fogel in the Blondie24 checker agent [1, 2] where a set of neural networks is evolved as board evaluators through co-evolution. One of the key highlights is that only minimum game knowledge is given to each neural network and they are largely required to learn from scratch through trial and error. In other words, the purpose is not to *design* a competent agent by explicitly incorporating hand-coded human knowledge and strategies. Instead, an agent is *evolved* gradually from a novice (in fact a crazy player making random decisions) to an expert. By doing so, this process can be arguably claimed to mimic at least certain aspects of human learning or intelligence.

However, in general, the computational time required is cumbersome. Typically, a population containing dozens of individuals is maintained and several game trials are required to evaluate the fitness of each individual due to the randomness of the game. Furthermore, in order to achieve modest performance, thousands of generations are usually needed, which may take several days of computing time even on modern PCs. By contrast, as shown in Fig. 2, human players require much less number of trials to develop basic game strategies to be able to play at a reasonable level.

We argue that the main challenge lies in the fact that only final scores are used to evaluate each candidate agent, which provide little if any direct guidance on the decision making activity in each time step (each game play is made of a long sequence of decisions). Undoubtedly, human players can actively acquire much richer feedback information during game play and use this information to improve their skills and adapt their strategies effectively. We are not talking about any advanced intelligent capabilities of human in this scenario. Instead, a human player can benefit greatly from his/her very basic and general rationality almost irrelevant to the specific game under consideration. For example, a human player can realize immediately that a wrong or invalid operation has been performed if something unexpected happens or a fatal decision has been made if a life is lost subsequently.

Note that assuming this type of rationality does not compromise our underlying philosophy of "learning without teaching" as no additional information or strategy specific to the game is provided to the agent. Also, it does not mean to abandon the idea of evolving and evaluating agents based on their overall performance. Instead, the learning process of agents can be expected to be more effective with the help of rationality and better mimic the behaviors of human players.

2 Game Modeling

A Matlab implementation of Pac-Man was developed in our experiments, which followed most of the details of the original game and facilitated the algorithm design and testing task thanks to the comprehensive toolboxes coming with Matlab. Certain factors have been removed from this simulator to create a more restricted test environment. There was only one ghost instead of four ghosts in the original game and there were no power pills or fruits either but otherwise this simulator can be regarded as a replicate of the Pac-Man game shown in Fig. 1.

The ghost had a hand-coded decision function that preferred the shortest path towards Pac-Man. For simplicity, the distance measure in use was Manhattan distance instead of the true distance based on the specific maze structure. The wrap-up effect was also considered so that the ghost could take advantage of the tunnels when necessary. To put in some randomness in the game play, the ghost was allowed to choose a random direction at each time step with a small probability.

The game information made available to the agent was chosen carefully to provide nothing more than what is readily available to a human player. A 3-by-3 matrix (actually the minimum window size) was used to represent Pac-Man's view of the maze (path vs. wall). A 4D vector was employed to indicate whether there was food in each of its four immediate neighbors. No global view of foods was provided due to the definition of rationality to be given later on. The location of ghost was represented by a 2D vector (row & column), indicating its relative position from Pac-Man. Since even human players cannot calculate the distance between Pac-Man and ghost precisely in real time when the ghost is far away, the location values were truncated to be within in the range of [-5, +5]. As a result, the complete game information consisted of 14 parameters in total in which 8 parameters specified the local maze structure and 4 parameters indicated the locations of foods and 2 parameters gave the distance of the ghost from Pac-Man (Fig. 3).

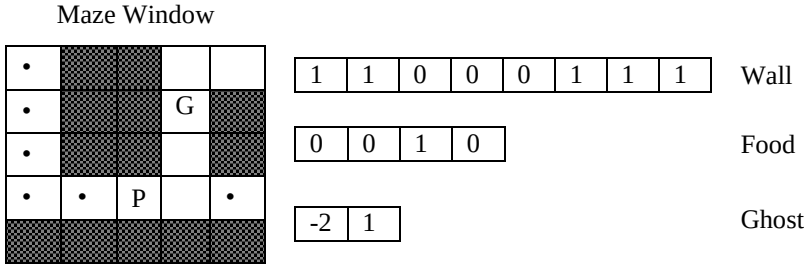

Fig. 3. An example of the game modeling scheme ("P" for Pac-Man and "G" for ghost). Walls are specified from left to right and from top to bottom not including Pac-Man's current position ("1" for wall and "0" for path). Foods are shown as black dots and are counted in the sequence of up, down, left and right. The ghost's location is represented by [-2, 1] as it is two rows above and one column to the right of Pac-Man. Special care should be taken when Pac-man is at the end of the tunnels so that information on the other side can be correctly represented.

The rationality of the agent was implemented in a very intuitive way depending on the feedback that it received from the game play as shown in Table 1. When Pac-Man did not move in the direction as expected (crashing into walls), a value of 0 was

assigned as the agent should realize that the current direction was invalid, although it may not know which direction was correct. When a life was lost (caught by the ghost), a value of 0 (actually worse than simply getting stuck) was assigned to indicate this fatal error. In the meantime, when a food was eaten, the agent should notice the increase of its score and a reward (a value of 1) should be given to its decision. For all other situations (e.g., walking along an empty path), a neutral value of 0.5 was given.

Table 1. Definition of rationality of the Pac-Man agent

Feedback	Score
Losing a life	0
Getting stuck	0
Gaining scores	1
Others	0.5

According to the above specification, at each time step in a game play, the agent can record a vector of input about the current game information, as well as the direction it choose to move together with a score for that particular decision based on its rationality (instinct). This set of information can be used to tell the agent whether the chosen move is favorable or not in a certain scenario. In other words, there is a set of training samples created from the game playing experience of the agent. Note that this representation scheme is not meant to be optimal. In fact, it is not difficult to see that the vector for walls can be made more compact by removing the 4 parameters related to the four corners as Pac-Man cannot move diagonally.

It should be emphasized that the training set generated in this way only indicates whether a certain move is good or bad but does not show the optimal move that the agent should follow. As a result, four neural networks are required with each one trained on a subset of the training samples grouped according to the type of move.

Table 2. An example of the training samples available to the four neural networks

Move	***Walls***								***Ghost***		***Food***				***Score***
2	0	1	1	0	0	1	1	1	-5	0	0	0	1	1	**0**
4	0	1	1	0	0	1	1	1	-5	-1	0	0	1	1	**1**
1	1	1	0	0	0	1	1	1	-5	-2	0	0	0	1	**0**
3	0	1	1	0	0	0	1	1	0	-1	0	0	0	0	**0**

Four training samples are presented in Table 2. For instance, the first one indicates that there are walls both above and below the agent and foods on both left and right sides while the ghost is on the far north. However, the move chosen by the agent is "2" (going down) and it gets stuck there immediately. As a result, the score for this decision in this particular situation is 0. The second one shows an almost identical game scenario but the move chosen by the agent is "4" (going right). Since there is a food on the right, a score of 1 is given to this training sample. In the last one, the ghost is on the left of the agent and is only one unit away. Unfortunately, the move chosen by the agent is "3" (going left) and a life is lost consequently while a score of 0 is assigned to this training sample.

After training, at each time step, the neural network with the highest output value decided the move of the agent. A possible undesirable phenomenon is that the agent may sometimes demonstrate a kind of oscillating behavior, typically in the corners. As a result, a rule can be imposed so that the agent will not make U turns as long as making a U turn is not significantly better than other possible moves.

3 Experiments

The purpose of the experimental studies was to investigate whether the agent could learn anything about playing Pac-Man solely based on the primitive information described in the last section. No special efforts were devoted to tuning the representation or algorithm parameters in the hope to achieve better performance as the focus was on scientific studies instead of comparative studies.

In the first series of experiments, the agent behaved in a completely random manner (each decision function was in fact a random number generator) simulating a novice trying all kinds of moves blindly. Unsurprisingly, the performance of the agent was very poor. Averaged over 10 trials, the agent was killed after 54.9 steps and only 10.3 foods were eaten, corresponding to a mean score of 103 (the maximum score was 2450 in our simulator). However, apart from the poor final scores, the agent did collect some useful information from the game plays. A training set with 549 samples was obtained, which contained each of its move and the corresponding feedback.

In the second series of experiments, the original training set was divided into four subsets based on the type of move. The number of samples in each subset was: 143 for "moving up"; 130 for "moving down"; 139 for "moving left"; 137 for "moving right". Four feedforward neural networks with 14 inputs, 10 hidden units and 1 output were trained using the Neural Network Toolbox in Matlab. The only parameter manually set was the number of hidden units.

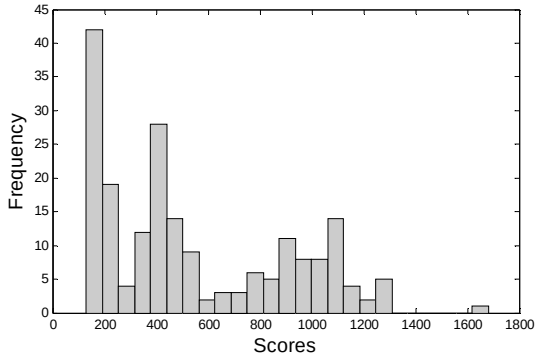

Fig. 4. The distribution of the scores of the agent over 200 trials (one life per trial).

After training, the four neural networks were loaded into the agent's decision function, evaluating the four possible moves at each step. At this stage, the agent was expected to behave more sensibly, with the help of its previous experience (regardless of good or bad experience). Although the outputs of the neural networks were

deterministic, the built-in function in ghost had some level of randomness. As a result, each trial may be more or less different and 200 trials were conducted. Fig. 4 shows the distribution of scores from which it is clear that the performance of the agent did vary a lot. The average score was 558.9 points, five times as high as its predecessor and the top score was 1680 points. A closer check of the games showed that Pac-Man had already learned how to walk around in the maze without crashing into walls. Note that this performance was obtained after only 10 game plays.

4 Discussion

One of the unique features of our approach is that four neural networks were employed with each one dedicated to evaluating a specific type of move. The reason is that there was no external teacher to show which move was the best in each situation that the agent encountered, other than the feedback that it perceived for its chosen move. These four neural networks, trained on different training sets, can be regarded as an ensemble of decision makers with different expertises.

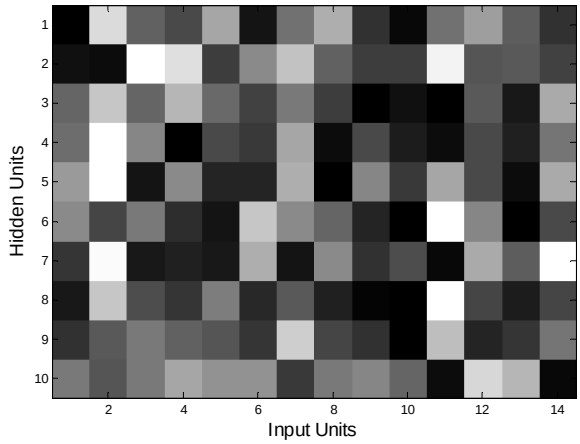

Fig. 5. Visualization of the weight matrix between input units and hidden units in the first neural network ("moving up").

Fig. 5 shows the weight matrix between 14 input units and 10 hidden units in the neural network evaluating the decision of "moving up" (the lighter a cell, the higher the absolute value of the corresponding weight). Although in general it is difficult to find out some comprehensible patterns, the second column related to the position directly above the agent has relatively large values as it determines whether moving up is feasible or not. In the meantime, the 11th column corresponding to whether there is food above the agent also has larger values compared to other three directions (the 12th, 13th and 14th columns) as it determines whether moving up is beneficial. Finally, since the impact of the ghost was only visible once in a trial when it caught Pac-Man, it did not play a significant role in the neural network (the 9th and 10th columns contain relatively small weights).

5 Conclusion

This paper demonstrated an alternative approach to the fascinating challenge of training a Pac-Man agent with minimum prior knowledge about the game. Different from previous studies solely relying on the overall scores and using evolutionary techniques to evolve agents based on neural networks or rule sets, we proposed to look into the details of each game play and collect additional information about the agent's behavior without violating the philosophy of "learning without teaching". Similar to a rational human player, the agent can also recognize and distinguish between positive and negative feedbacks by observing the progress of the game.

Experimental results show that, even with a highly restricted view of the surrounding environments and very basic computational capability, the agent can still master many essential skills for playing the Pac-Man game within significantly less number of trials compared to existing approaches. As to future work, it would be interesting to investigate how the proposed mechanism can help training more powerful Pac-Man agents that can effectively navigate through the maze. In the meantime, the combination of classical evolutionary approach and the rationality based approach may also produce a number of interesting research topics.

Acknowledgement

This work is supported by the Scientific Research Foundation for Returned Overseas Scholars, Ministry of Education, P.R. China and National Natural Science Foundation of China (No. 60905030). The authors are also grateful to the volunteers for their help.

References

1. Chellapilla, K., Fogel, D.: Evolving an Expert Checkers Playing Program without Using Human Expertise. IEEE Transactions on Evolutionary Computation, 5(4), 422-428 (2001)
2. Fogel, D.: Blondi24: Playing at the Edge of AI. Morgan Kaufmann, San Francisco (2002)
3. Gallagher, M., Ledwich, M.: Evolving Pac-Man Players: Can We Learn from Raw Input? In: IEEE Symposium on Computational Intelligence and Games, pp. 282-287 (2007)
4. Gallagher, M., Ryan, A.: Learning to Play Pac-Man: An Evolutionary, Rule-Based Approach. In: Congress on Evolutionary Computation, pp. 2462-2469 (2003)
5. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
6. Lucas, S.: Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In: IEEE Symposium on Computational Intelligence and Games, pp. 203-210 (2005)
7. Robles, D., Lucas, S.: A Simple Tree Search Method for Playing Ms. Pac-Man. In: IEEE Symposium on Computational Intelligence and Games, pp. 249-255 (2009)
8. Szita, I., Lõrincz, A.: Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man. Journal of Artificial Intelligence Research, 30(1), 659-684 (2007)
9. Uston, K.: Mastering Pac-Man. Signet (1981)
10. Wirth, N., Gallagher, M.: An Influence Map Model for Playing Ms. Pac-Man. In: IEEE Symposium on Computational Intelligence and Games, pp. 228-233 (2008)
11. Online Pac-Man Game, http://www.neave.com/games/pacman
12. Pac-Man, http://en.wikipedia.org/wiki/Pac-Man