Playing Angry Birds with a Neural Network and Tree Search

Yuntian Ma, Yoshina Takano, Enzhi Zhang, Tomohiro Harada, and Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Graduate School of Information Science and Engineering, Ritsumeikan University
Kusatsu, Shiga, Japan
ruck@is.ritsumei.ac.jp

Abstract

In this paper, we introduce a method that combines a deep neural network and tree search for an Angry Birds AI agent. This neural network is trained first by supervised learning from another expert and then by reinforcement learning from self-play. Tree search enhanced by the neural network trained with supervised learning is used to strengthen the agent's game play policy during reinforcement learning. To the authors' knowledge, this is the first time that this approach is used to develop an Angry Birds AI agent. Our agent participates in the 2018 Angry Birds AI Competition and will be made available after the competition. The authors hope other researchers can gain some pieces of useful information from our findings and make deep learning more popular in Angry Birds AI Competition.

1 Introduction

Angry Birds is a popular video game developed by Rovio. It is a game that players use a slingshot to shoot pigs with birds. This mechanism is just like throwing somethings in the real world, so human players can master it very soon from their life experience. AI agents can also play this game well by using strategy or knowledge based methods. However, we want to make an AI agent that thinks more like a human player with deep learning.

The Angry Birds AI Competition [1] is held by Jochen Renz's group at the Australian National University every year from 2012 to now. The task of this competition is to develop agents able to successfully play the game autonomously and without human intervention. Many agents used strategy or knowledge based methods in past competitions [Calimeri *et al.*, 2016; Jutzeler *et al.*, 2013; Paul and Hüllermeier, 2015]. Some agents used machine learning but did not obtain good result [Narayan-Chen *et al.*, 2013; Tziortziotis *et al.*, 2016]. We will be participating in the 2018 competition with our deep reinforcement learning AI agent.

[1] https://aibirds.org/call-for-participation.html

2 Background

2.1 Deep Reinforcement Learning

Deep reinforcement learning (DRL) extends reinforcement learning. In traditional reinforcement learning, an agent learns to take an action that leads to the maximum reward from environment. However, bias from human design of reward functions may influence the agent performance. DRL allows the entire learning process from observation to action to be done with reduced bias. DRL succeeded in many games, such as playing a number of Atari games at a superhuman level [Van Hasselt *et al.*, 2016] and defeating world champions in Go [Silver *et al.*, 2016]. Our agent combines DRL and Monte-Carlo tree search (MCTS) to train a neural network that can decide its actions directly from information on objection positions in the screen.

2.2 Monte-Carlo Tree Search

Many games need players to select an action sequence from a lot of actions. In addition, the game state after an action may be unpredictable because of game randomness or opponent actions (an opponent is usually unpredictable). An AI agent needs both to try many different actions to get a better selection and to try one action several times to increase the credibility of each selection. As a result, such games are multi-armed bandit problems and need players' selection to balance between exploitation and exploration.

Monte-Carlo tree search (MCTS) is a heuristic search algorithm for some kinds of decision processes. It searches solutions based on playout, while balancing between exploitation and exploration. It obtained successful results in many games such as traditional turn-based board game Go and real-time video game Ms. Pac-Man ([Browne *et al.*, 2012] and [Samothrakis *et al.*, 2011]).

2.3 ScienceBirds

ScienceBirds [2] is a Unity 3D open source clone of Angry Birds. Unlike Angry Birds, it is easy to add new levels, using, say, a sample level generator, for testing AI agents. In addition, its game speed can be readily changed by the Unity game engine. The IJCAI/CIG Angry Birds Level Generation Competition uses ScienceBirds as the official platform. One

[2] https://github.com/lucasnfe/Science-Birds

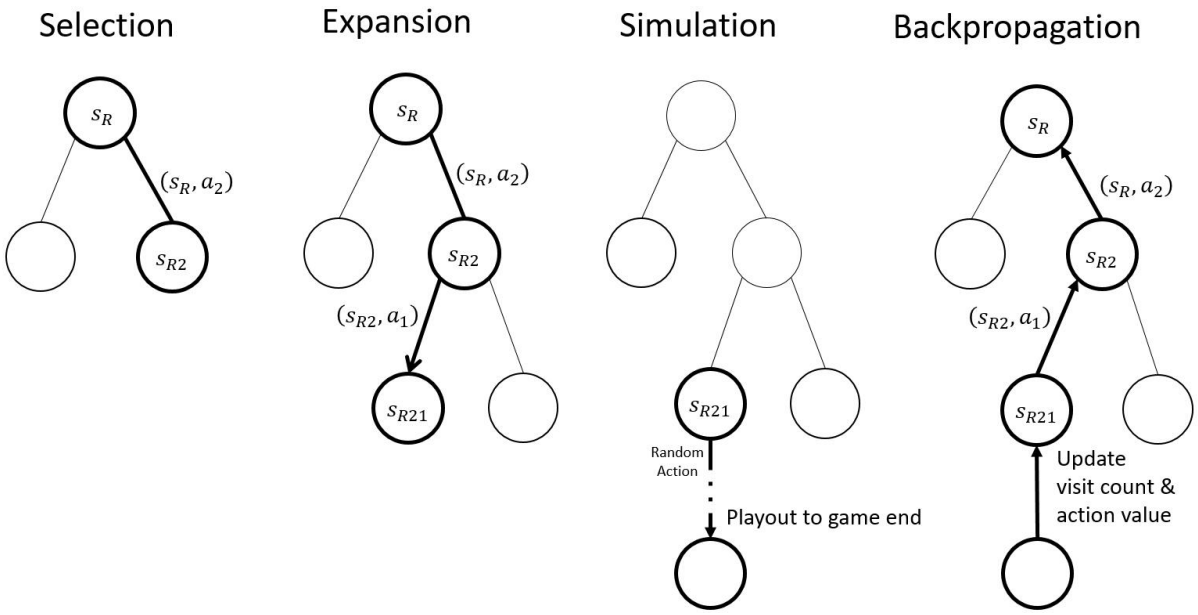

Figure 1: Four Steps of Standard MCTS. Selection: select a path from the root to a leaf. Expansion: expand the selected leaf node when certain conditions are met and select one of the newly expanded nodes as the new leaf of this path. Simulation: run a playout (simulation) from the game state of the leaf node to game end. Backpropagation: update the nodes' information from the leaf to the root with the action value of the playout's result.

can use it to train a deep learning AI agent if a connection between these two games can be established.

ScienceBirds has different graphics from Angry Birds. This will reduce the accuracy of neural networks if screenshots are used as input. To solve this problem, a new input format using Boolean to represent whether game objects exist is used.

$$Matrix(x, y, c) = 1(x, y, c) \tag{1}$$

where $1(x, y, c)$ is one if there is a game object of type c at coordinates (x, y) of the screenshot (game world re-sized to this screenshot's size); otherwise, zero. c is channel index for one type of game objects; there are mainly 11 types of game objects in ScienceBirds, and one independent input channel is used for each object type. In ScienceBirds, all the information can be obtained directly from the Unity game engine. In Angry Birds, all of the above information can be obtained by analyzing game screenshots using a computer vision component provided by the Angry Birds AI Competition's organizers. By this input-formatting method, we can treat ScienceBirds and Angry Birds as the same game.

A simple MCTS AI agent is used for playing the game and generating training samples; in this agent, each shoot, defined by its angle and force magnitude, is associated with a priori probability for the selection step (see the next section for details). New training levels are generated randomly by the baseline level generator from the AIBIRDS Level Generation Competition. Every level is used only once to avoid overfitting of the neural network. All data for training are logged directly by the Unity game engine without any changes.

3 MCTS in ScienceBirds

MCTS' play style and performance vary from game to game. This section focuses on how MCTS operates on ScienceBirds.

3.1 Standard MCTS

Standard MCTS performs well on any games which have limited actions and lengths, including ScienceBirds. Figure 1 shows how standard MCTS operates in ScienceBirds. A node s represents a specific game state and the root node is the starting game state of a level. An edge(s, a) represents a specific action under the game state s. A playout on ScienceBirds can be performed with an increased game speed. Every playout can continue until game end because the game length is not so long.

MCTS in ScienceBirds consists of four steps. An MCTS AI agent repeats them until a limited count or time is reached.

- Selection. A path starting from the root node to a leaf node will be selected to maximize the UCB1 value.

$$UCB1(s, a) = Q(s, a) + C\sqrt{\frac{ln(N(s))}{N(s, a)}}, \tag{2}$$

  where $Q(s, a)$ is the average action value from the simulation result of edge(s, a), $N(s)$ is the count of all simulations of node s, $N(s, a)$ is the count of simulations for edge(s, a), and C is a parameter for balancing the search length and width.

- Expansion. The leaf node at the selected path is expanded if pre-defined conditions are satisfied. A visit count threshold is used here as a pre-defined condition.

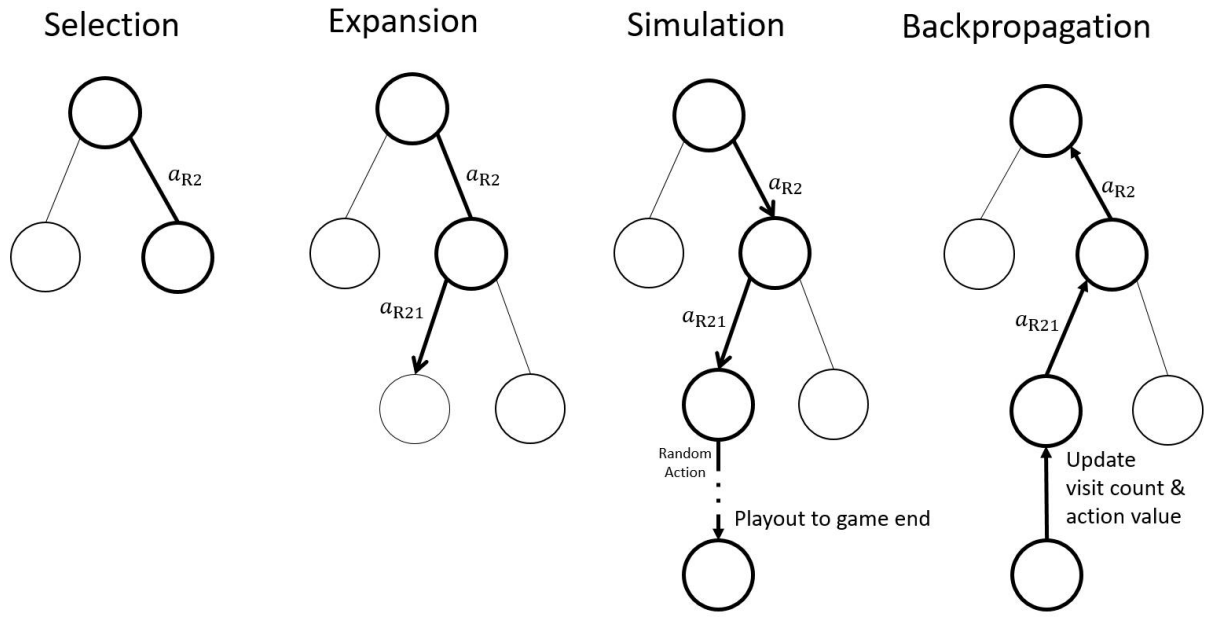

Figure 2: Four Steps of Open Loop MCTS. Different from standard MCTS, Open Loop MCTS does not storage game states in nodes. Only actions are stored in edges. A playout from the selected leaf node must be preceded by a simulation starting from the root node to the selected leaf node using the actions stored in the edges on the path.

- Simulation. A playout is played from the selected leaf node until a terminated length or game end, including clearing or failing the current level.
- Backpropagation. All nodes and edges from the leaf node to the root node are updated with an action value of the simulation result. The action value is calculated by:

$$Q = \lambda(pig_b - pig_a) + (1-\lambda)(block_b - block_a) + bird_a, \tag{3}$$

where pig_b and pig_a are the numbers of pigs before and after the current simulation; $block_b$ and $block_a$ are the numbers of blocks before and after the current simulation; $bird_a$ is the number of birds after the current simulation; λ is set between [0, 1] and should be a large value because the score for killing pigs is much higher than destroying blocks; $bird_a$ is an additional reward if the AI agent can clear a level without using all the birds.

Values stored in nodes and edges are updated by:

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i), \tag{4}$$

$$N(s) = \sum_{i=1}^{n} 1(s,i), \tag{5}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) Q_i, \tag{6}$$

where $1(s,a,i)$ indicates whether edge(s,a) was simulated in the ith simulation, $1(s,i)$ indicates whether node$s$ was simulated in the ith simulation, and Q_i is the action value of the ith simulation.

3.2 Open Loop MCTS

There exists some randomness in ScienceBirds that sometimes makes game results different even the player takes the same actions. Hence, using a node to store a game state and re-use it is not practical. Open loop MCTS success in some video games [Perez Liebana *et al.*, 2015] and can be used in ScienceBirds for better performance. In open loop MCTS, game states are not stored and will be simulated every time. Figure 2 shows how open loop MCTS operates in ScienceBirds.

3.3 MCTS with priori probabilities

In UCB1, all legal actions are initially seen having the same weight. However, in many situations, useless actions can be distinguished easily. As a result, a selection policy with prior probabilities will help improve the search performance.

Google AlphaGo uses a selection function consisting of P and Q shown in eqn. (7) and obtained a big success in game Go, a type of game which has a very large amount of exploration space. Angry Birds also have a large amount of exploration space, so we expect that this kind of selection function may also work well. In this work, we use open loop MCTS because open loop is more suitable for ScienceBirds.

- Selection.
  An action is selected to maximize the following value:

$$SEL(a) = Q(a) + C\frac{P(a)}{1+N(a)}, \tag{7}$$

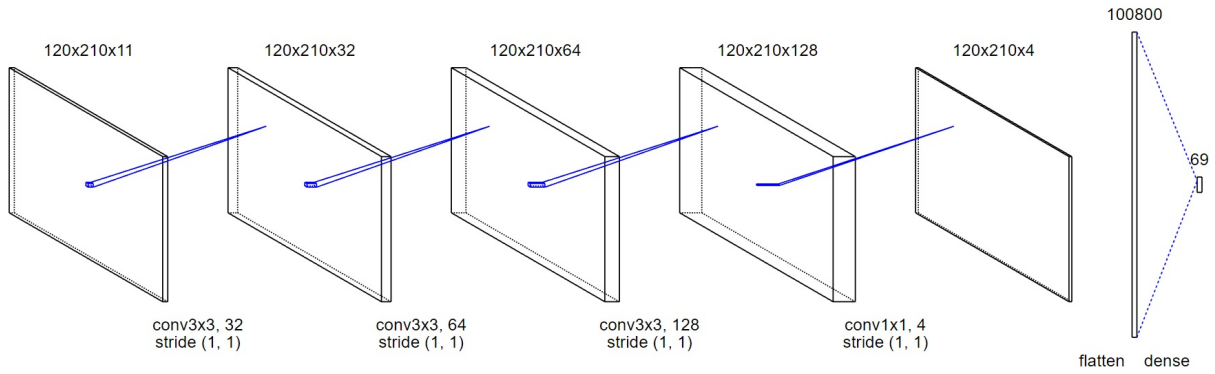

Figure 3: Deep Neural Network Structure. The 1st to 3rd convolutional layers use 32, 64, 128 filters with size 3×3. The 4th convolutional layer uses 4 filters with size 1×1 for compression. The 5th fully connected layer uses 69 units to output the probability of each action.

where $Q(a)$ is the average action value of edge a from the simulation results, $N(a)$ is the visit count of edge a, and $P(a)$ is the prior probability of edge a with its weight becoming less as $N(a)$ increases. There are many methods to decide P, and we use a simple one as follows:

$$P(a) = force(a), \tag{8}$$

where $force(a)$ is the magnitude of shooting force, scaled to [0, 1], of the action in edge a, and C is a balance weight between P and Q. This method is used because larger force has more destructive power and has more chance to clear levels.

At the start of an action plan, Q and N of all edges are the same. As such, selection is only based on P or in other words by a priori experience. Once an edge is visited, it will have N and Q from the simulation result; then the influence of prior probabilities will be divided by N for encouraging more exploration. Large Q will promote exploitation.

- Expansion and Simulation. Same as open-loop MCTS with UCB1.
- Backpropagation.

  All the edges from the leaf node to the root node are updated by the simulation result using the same action value as eqn. (3) by:

$$N(a) = \sum_{i=1}^{n} 1(a,i), and \tag{9}$$

$$Q(a) = \frac{1}{N(a)} \sum_{i=1}^{n} 1(a,i) Q_i, \tag{10}$$

  where $1(a,i)$ indicates whether edge a is visited in the ith simulation.

The most visited action will be selected after all searches end.

4 Training

This section focuses on how to train a deep neural network on ScienceBirds.

4.1 Supervised Learning

At the first part of training, our AI agent is trained under supervised learning. The main purpose of this part is to learn an expert's action policy. The input is the Object Matrix input format which represents the current game state. The output layer is the probability distribution of all legal actions. This neural network aims at maximizing the likelihood of the expert's action a under game state s. The expert in use is open loop MCTS with prior probabilities (sect. 3.3).

A 5-layer convolutional neural network is used in this part and shown in Fig. 3. All layers except the output layer use ReLU as the activation function. The output layer uses softmax as the activation function, and the number of neurons equals the number of legal actions. Mini-batch gradient descent is used for training this neural network. Softmax cross entropy of the network output and that of the expert action are used for the loss function.

All training samples in this part are generated by the aforementioned open loop MCTS with prior probabilities. The MCTS AI, an expert in our case, runs on ScienceBirds and obtains information directly from the game engine. Game states from the game engine are transformed into the Object Matrix format as the network's input. At a given state, the count of each action selected at the root node during the simulation step of MCTS is recorded and transformed into a softmax form as the label of this state. Training of this part continue until the AI agent can clear 90% of normal-difficulty levels.

4.2 Reinforcement Learning

At the second part of training, the neural network trained in first part is improved by reinforcement learning and MCTS. The main purpose of this part is to learn and gain experience from the AI agent's self-play. Google Deepmind proposed an excellent method which is used in their AI agent of Go ([Silver *et al.*, 2016] and [Silver *et al.*, 2017]). Figure 4 shows how our AI agent uses reinforcement learning and

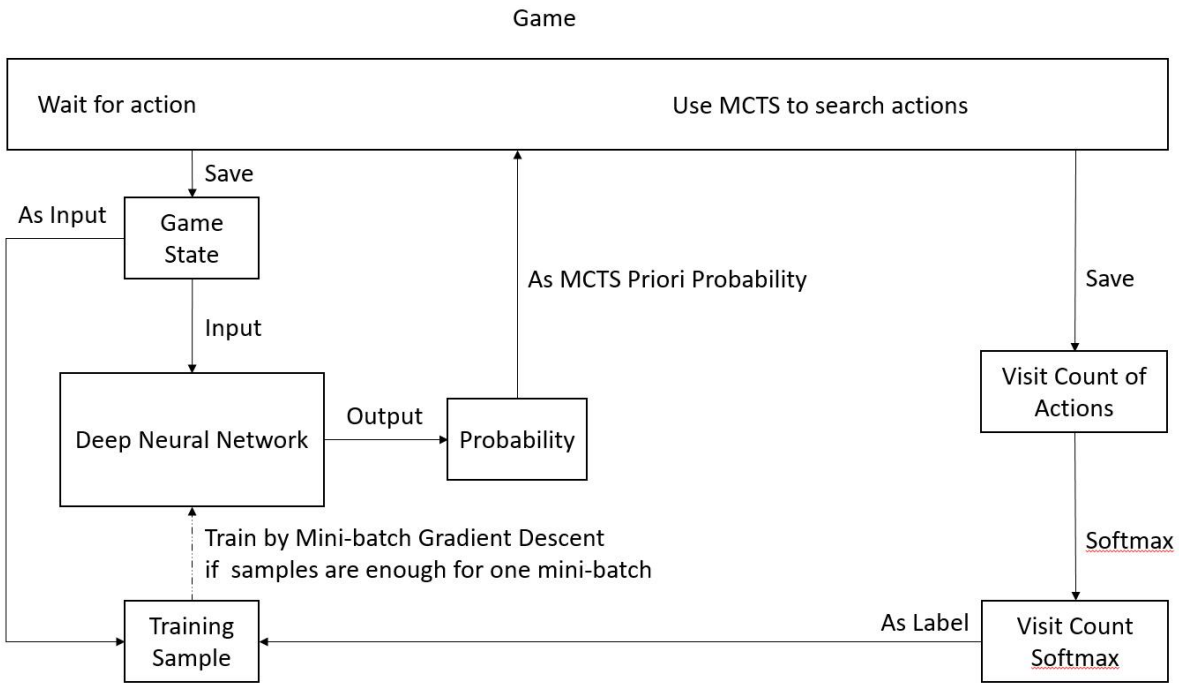

Figure 4: Reinforcement Learning Flow chart

MCTS to improve itself in ScienceBirds. The MCTS AI with priori probabilities is also used here for generating new training samples, but this time the prior probabilities of its actions come from the output of the latest version of the neural network at the current game state s. This neural network aims at maximizing the likelihood of the improved AI agent action a under game state s.

The structure of the neural network is the same as the supervised neural network, and its weights are initialized with the trained weights of the supervised one. The training algorithm is as follows:

- Start a random level and launch the game run until it awaits an action;
- Save the game state and input it to the neural network;
- Get the output probability distribution and use it as prior probabilities in MCTS;
- Use the MCTS AI with prior probabilities to play the game;
- Record the visit counts of all nodes in depth 1 in the simulation step of MCTS;
- Repeat until the level is cleared or failed;
- Save the game states and the visit counts of the nodes in depth 1 as training samples;
- Start a training step when the number of training samples is enough for one mini-batch.

Training of this part continues until the AI agent is strong enough.

5 Playing

This section focuses on how to use the proposed deep neural network to play Angry Birds.

5.1 Action Adjustment

In ScienceBirds, the AI agent combines the neural network and MCTS when playing. The method for playing Angry Birds is similar to the method used in reinforcement learning, but randomness is removed for the sake of performance. Unlike ScienceBirds, due to the unavailability of the game's forward model, we cannot use MCTS in the Angry Birds chrome version that is used in the competition. Ideally, the AI agent should always select the action with the highest probability in the neural network output. However, it cannot be guaranteed that the resulting neural network is perfect, so the AI agent needs another policy for adjusting its actions to obtain a better result when replaying a level.

In the competition, an AI agent can play and replay any level until a given time limit. Although there is some randomness in Angry Birds, but its effect to results is small. In most of the cases, game results will not change that much if the AI agent does not change its actions. To avoid this risk of repeating meaningless replays, our AI agent adjusts its actions, according to which the game will enter a new sequence of game states, and different results can be expected.

In particular, the proposed AI agent adjusts the most useless action. Sometimes Angry Birds levels can be very complex, so it is difficult to judge an action's importance by only pig count. When pigs are all hidden by blocks and no pig can be killed after only one action, a good action should destroy many blocks as much as possible. When pigs can be reached, a good action should try to kill them. As a result, the success

rate of one action can be calculated by combining the pig kill count and the block destroy count:

$$rate = \lambda \frac{pig_b - pig_a}{pig_b} + (1 - \lambda) \frac{block_b - block_a}{block_b}, \quad (11)$$

where pig_b and pig_a are the numbers of pigs before and after the action, $block_b$ and $block_a$ are the number of blocks before and after the action, λ is set between 0 and 1 for taking the balance of killing pigs and destroying blocks with the higher λ the greedier in killing pigs. In our AI agent, λ is set to 0.9 because killing pigs is much more important than destroying blocks. The action having the highest probability in the neural network output but the lowest success rate will be replaced with the action with the next highest probability.

5.2 Bird Skills

We do not consider bird skills in training. As a result, we use a simple method when playing Angry Birds. Our strategy is to ensure that the change of the bird's trajectory is as little as possible. Blue bird skill is randomly used at 65% - 85% of its path to attack more objects. Yellow bird and white bird skill are randomly used at last for more attack power. Black bird skill is used automatically after hit.

6 Results

This section describes the results in ScienceBirds training and Angry Birds real play.

6.1 Training part

In the supervised learning part, we used the softmax cross entropy loss between the neural network output and the action selection of the MCTS AI to represent the accuracy in predicting the actions of the MCTS AI agent. Figure 5 shows the result. In this part, the loss continues to drop until about the 290th training batch. This shows that the output of the neural network gets closer to action selection of the MCTS AI.

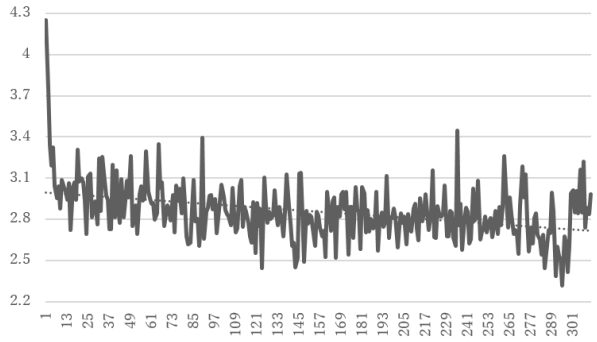

Figure 5: Loss transition in supervised learning

In the reinforcement learning part, the softmax cross entropy loss between the neural network output and the action selection of the MCTS AI enhanced by this output is used to represent the performance of the proposed DRL AI agent. Figure 6 shows the result.

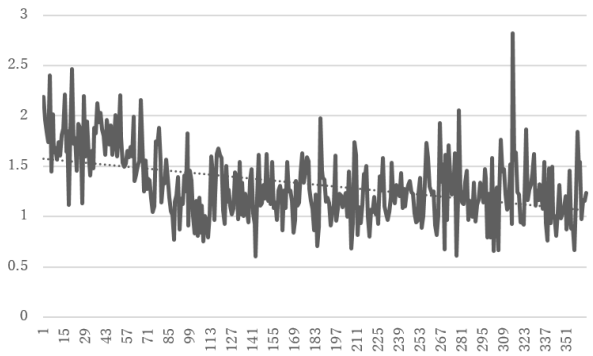

Figure 6: Loss transition in reinforcement learning

In this part, the performance of the MCTS AI will grow up by imitating MCTS AI enhanced by the output of the neural network. The loss can also show the growth of AI agent. Ideally the loss should drop to somewhere close to 0 for the output to be very close to the real selection. However, it is very difficult in this case because there are too many types of game states to handle.

6.2 Real play performance

We tested our AI agent in all past competition Quarter Final levels in from the 2013 to 2017 competitions. The detail results of other AI agents can be found in aibird.org. Our results are shown in Table 1.

Table 1: Results of Past Competition Levels

Year	2013	2014	2015	2016	2017
Score	76770	203800	143260	203430	124110

7 Conclusions and Future Work

This paper presented a new method that can train a deep learning Angry Birds AI agent by tree search and self-play reinforcement learning. For achieving tree search which is almost impossible in Angry Birds, an open source clone game named ScienceBirds was used for training the neural network. Although these two games are different in details, it can be seen from the given results that the proposed AI agent is promising.

For future work, more details about Angry Birds need to be handled by the neural network such as bird skill. Adjusting the release point of a shoot is totally different between the two games, and more work about the game control is necessary. In addition, development of an Angry Birds simulation for being used directly in the training step is also a promising direction for more effective neural networks.

Acknowledgments

Our thanks go to the organizers of aibirds.org for holding the Angry Birds AI Competition and to Lucas N. Ferreira for making ScienceBirds and publishing its open source for research.

References

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Calimeri *et al.*, 2016] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-hex: an artificial player for angry birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):128–139, 2016.

[Jutzeler *et al.*, 2013] Arnaud Jutzeler, Mirko Katanic, and Jason Jingshi Li. Managing luck: A multi-armed bandits meta-agent for the angry birds competition. *AI Birds Competition*, 2013.

[Narayan-Chen *et al.*, 2013] Anjali Narayan-Chen, Liqi Xu, and Jude Shavlik. An empirical evaluation of machine learning approaches for angry birds. In *International Joint Conference on Artificial Intelligence*, 2013.

[Paul and Hüllermeier, 2015] Adil Paul and Eyke Hüllermeier. A cbr approach to the angry birds game. In *ICCBR (Workshops)*, pages 68–77, 2015.

[Perez Liebana *et al.*, 2015] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas. Open loop search for general video game playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 337–344. ACM, 2015.

[Samothrakis *et al.*, 2011] Spyridon Samothrakis, David Robles, and Simon Lucas. Fast approximate max-n monte carlo tree search for ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154, 2011.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Tziortziotis *et al.*, 2016] Nikolaos Tziortziotis, Georgios Papagiannis, and Konstantinos Blekas. A bayesian ensemble regression framework on the angry birds game. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):104–115, 2016.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.