Enhancing Deep Reinforcement Learning Agent for Angry Birds *

Yufeng Yuan†, Zitan Chen†, Panhao Wu and Liang Chang‡
College of Information Science and Technology,
Beijing Normal University, China
yufeng_yuan@outlook.com, chenzitan@icloud.com, ekinghao@gmail.com, changliang@bnu.edu.cn

Abstract

Deep reinforcement learning has been successfully used in Atari and robot control. It has great advantages in the learning ability and effective learning strategies. In this work, we use the annotated images generated from key frames in the game as the input of the network. Then, we use several previous successful agent to collect high quality training data and train our model with Deep Deterministic Policy Gradients (DDPG) algorithms to predict the target point. To accelerate the model training, we revised the basic game playing software which enables us to create four Chrome instances to accelerate our training. After training, we find our model, in first 21 levels, outperforms most previous successful agents.

1 Introduction

Angry Birds is a popular physics-based simulation game. In this game, players use a slingshot to launch birds at pigs stationed in or around various structures with the goal of destroying all the pigs on the playing field. As players advance through the game, new types of birds become available, some with special abilities that can be activated by the player at proper time. The goal of the Angry Birds AI Competition is to build an intelligent Angry Birds playing agent that can play new game levels as good or better than the best human players. Humans are very good at predicting consequences of physical actions, but it is difficult for AI.

Various methods based on heuristic search or machine learning have been proposed, which achieved competitive results. In recent years, with the rapid development of GPU and the emergence of large data, deep learning has been used widely in artificial intelligence and computer vision. We base our game engine on deep reinforcement learning in this work and give improvement to enhance the learning ability. We select three key features, bird type, distance to target point and target point state the game state representation. We use two basic approaches to tackle the problem, value iteration and policy iteration. One of well-known value iteration algorithms is Deep Q-Network (DQN) which is capable of human level performance on many Atari video games directly using screen pixels as input. However, DQN can only handle discrete and low-dimensional action spaces since it relies on a finding the action that maximizes the action-value function, which in continuous valued cases the action spaces can be infinite. Unfortunately, our task Angry Birds and many other games and tasks have high-dimensional or continuous valued action spaces. In those problems, Policy Gradient is preferred because it is end-to-end: theres an explicit policy and a principled approach that directly optimizes the expected reward. Google DeepMind has devised a new algorithm named DDPG to tackle the continuous action space problem. It is based on three techniques in reinforcement learning, i.e. Deep Q-Network, Deterministic Policy-Gradient Algorithms and Actor-Critic Methods. We train our model with DDPG to predict the optimal shot angle and tap time based on the original action.

The rest of the paper is organized as follows. In the next section, we review the background and related work for agents playing the Angry Birds game. In Section 3 ,we briefly introduce the method we use in learning to play Angry Birds, including the training steps. In Section 4, we explain the experimental methodology. We present our result and discussion in Section 5 and made the outlook of future work in Section 6. The conclusion was summarized in Sesction 7.

2 Background and Related Work

Since the first competition held in 2012, different approaches are used by participants such as logic programming, qualitative reasoning, advanced simulation, structural analysis, analysis of predicted damage, and different machine learning techniques such as Bayesian ensemble learning, Weight majority algorithm, Heuristic search are applied. But to the best of our knowledge, there hasnt any team who included deep reinforcement learning to solve this problem. and that's what our work was mainly about.

2.1 Existing Agent

Narayan-Chen et al. [Narayan-Chen *et al.*, 2013] train a weighted majority and a Naive Bayes algorithm on a data set consisting of good and bad shots in different states of the

*[This work is supported by National Natural Science Foundation of China (Grant No. 61402040, 61473276)]

†indicates equal contribution

‡corresponding author

game. Nuradiansyah, Ziberi and Tirtarasa [Nuradiansyah *et al.*,] utilize the Analytic Hierarchy Process as their method for decision making. With the goal to hit the best structure, they defined five criteria and evaluate their relative importance, and then determine the best shot considering the importance of these criteria. Tziortziotis and Buche [Tziortziotis *et al.*, 2016] use a tree structure to represent the objects in a game scene, and formulate the problem of selecting an object for shooting as a regression problem. They associate with each pair of object material and bird a Bayesian linear regression model, making decision according to the best prediction of the ensemble model. In [Wałega *et al.*, 2014], the authors assign a numerical score to each reachable object based on its physical properties to reflect the extent of damage it suffers if being hit, and shoots at objects with low stability but high influence on pig or shelters of pigs.

Paul and Hllermeier [Paul and Hüllermeier, 2015] build a case-based angry birds agent. They use a preference-based approach which compares different solutions for a given problem and maintains the better one to build the case base, and then try to find the most similar game scene from its case base and adopts its shot when playing a new game. Calimeri, Fink and Germano [Calimeri *et al.*, 2016] extend Answer Set Programming with external sources and other extensions based on declarative knowledge bases. They rank each shootable target considering their likelihood to destroy other objects for a single shoot.

Polceanu and Buche [Polceanu and Buche, 2014] build their decision making based on the theory of mental simulation. Their agent observes the effects of performing multiple simulations of different shots in a given game scene and selects the optimal solution based on these results.

2.2 Deep Q-Network

A representative work of deep learning is on playing Atari with Deep Reinforcement Learning [Mnih *et al.*, 2013]. The reinforcement learning algorithm is connected to a deep neural network which operates directly on RGB images. The training data is processed by using stochastic gradient method.

A Q-network denotes a neural network which approximates the reinforcement learning value-action function with weights θ. At each iteration i, we train the Q-network by minimizing the loss functions $L_i(\theta_i)$, which is formulated as $L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$. Here, $y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target and $\rho(s,a)$ is a probability distribution over sequences s and actions a. And $\rho(s,a)$ is the behaviour distribution.

In the work[Mnih *et al.*, 2013], they utilize a technique known as experience replay. The agents experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored in a data-set $\mathcal{D} = \{e_1, ..., e_N\}$, pooled over many episodes into a replay memory. In the inner loop of the algorithm, they apply Q-learning updates to samples of experience drawn at random from the pool of stored samples. The agent chooses and executes an action using an ϵ-greedy policy after performing experience replay. The algorithm is called deep Q-learning and is presented below.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for t = 1, T **do**
    With probability ϵ select action a_t
    otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = r_j$ for terminal ϕ_{j+1}
    Set $y_j = r_j + \gamma max_{a'} Q((\phi_{j+1}, a^{'}); \theta)$ for non-terminal ϕ_{j+1}
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

2.3 Deterministic Policy-Gradient

Policy evaluation methods estimate the action-value function $Q^\pi(s,a)$ or $Q^\mu(s,a)$ by Monte-Carlo evaluation or temporal-difference learning[Sutton and Barto, 1998]. We use action-value function to update the policy. The general approach is a greedy maximization of the action-value function. In continuous action spaces, a simple approach is to move the policy in the direction of the gradient of Q. The policy parameters θ^{k+1} are updated in proportion to the gradient $\nabla Q^{\mu^k}(s, \mu_\theta(s))$ for each state s.

Taking an expectation with respect to the state distribution $\rho^\mu(s)$, the policy parameters are updated as:

$$\theta_{k+1} = \theta_k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}[\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))]$$

Using the chain rule, we derive the following equation, i.e., the policy improvement is represented by the the gradient of the action-value with respect to actions multiplies the gradient of the policy with respect to the policy parameters.

$$\theta_{k+1} = \theta_k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s,a)|_{a=\mu_\theta(s)}]$$

Here, $\nabla_\theta \mu_\theta(s)$ is a Jacobian matrix. Each column of $\nabla_\theta \mu_\theta(s)$ is the gradient $\nabla_\theta [\mu_\theta(s)]_d$ of the dth action dimension of the policy with respect to θ.

2.4 Actor-Critic Model

The Actor-Critic Model combines the policy gradient method and the value function method. We call the policy function the actor and the value function the critic. In the model, the actor makes an action in the current state, while the critic criticizes the actions made by the actor. In our task Angry Birds, we use the continuous Q-learning (SARSA) as our critic model and use policy gradient method as our actor model.

2.5 Deep Deterministic Policy Gradient

Google DeepMind has devised a new algorithm to tackle the continuous action space problem called Deep Deterministic Policy Gradients (DDPG). It is based on the above three techniques in reinforcement learning: Deep Q-Network, Deterministic Policy-Gradient Algorithms and Actor-Critic Methods. The following is the main algorithm we use in our reinforcement learning agent.

Algorithm 2 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

3 Learning to Play Angry Birds

To train our reinforcement learning agent, we need to collect a lot of training data. It's obvious that high-quality data, which means data with high episode score can make our training more effective and less likely to diverge. However, due to the inability of the Naive Agent to win game consistently and produce high quality training data, we uses several previous successful agent to collect our training data, including BamBird, DataLab bird which are open source agent listed in the competition forum.

3.1 Game State Feature Representation

In a game episode of Angry Birds, we use the anotated image generated by our agent for training, which keep the relevant features while eliminate the unnecessary features. Using the provided game-playing software, we recognize birds, blocks and etc. We use different colors to represent objects separately. The technique for hole filling is designed. An original sceenshot picture in the game and its corresponding annotated image are shown below in Figure 2.

Since the game scences are so complicated, different objects, the terrain and sky are recognized. Then annotated images are generated based on the extracted features.

Figure 1: The Original Image and the Annotated Image

- Birds: Birds represented in their original colors and at their locations.
- Blocks and TNT: Wood colored in orange, glass in light blue, stones in grey and TNT in light red.
- Sky, air, ground and mountains: Sky and air colored in black. Ground and mountains colored in brown.

3.2 Shoot The Target Point

In the game, our agent needs to predict the target point for shoot action. We use the annotated image as the state, and the target point as the action. The reinforcement learning agent would execute the actions with some noise in the real game. After training, it would be able to figure out whether the current action is optimal and how much it should be changed.

3.3 Training Steps

The whole Agent consists of two parts: a Java controller and a RL model implemented in Python. After the game starts, specific level of the game will first be loaded by the controller and then the RL model begins to work.

Figure 2: The Process to Improve the Target Point

The base agent will generate a target point each time when a bird is waiting to shoot.

1. Before the shoot action is taken, a sequence of information is sent from the controller to the model, including the current level number, bird type of the shot and distance from the release point to target point.
2. Reinforcement learning model receives the information and calculate the offset distance from the original target point to activate the bird's special ability.
3. The controller executes shooting action generated by the RL model.
4. The controller collects the information about this shot and status, i.e., the current level number, bird type of the shot, distance from the release point to target point generated by base agent, target point before this shot, offset to original target point, score of this shot, target point after this shot and whether this episode is finished. The information will be stored for agent training.
5. For each iteration of training, we randomly sample a mini-batch of training data from the replay buffer before applying stochastic gradient descent to our model.

4 Experimental Methodology

We train our model from Levels 1 to 42 in Poached Eggs theme in Angry Birds. In the experiments, we implement the training with these levels repeatedly until the model converges. To accelerate the convergence, we start 4 Chrome instances with all Angry Birds episode(1-42) unlocked. In the experiment, we use 4 Java controller instances, each communicating with a Chrome instance which enables us to collect 4 training data during the time of one game episode.

We implemented our model in Python with Keras on top of TensorFlow and the controller in Java based on the original Angry Birds software. Controller processes communicate with the model process by POSIX named pipeline in synchronous mode.

5 Result and Discussion

We test our agent at different levels. Let the agent play every single level for a sufficient number of iterations. In each iteration, it has a score that the agent actually achieves. The score at each level is the sum of the iteration scores. The following table shows the maximum score(Max-Score) attained for some levels, each level is played for 8 times.

Level	Level 1	Level 5	Level 10	Level 15
Max-Score	30930	63540	61970	54860

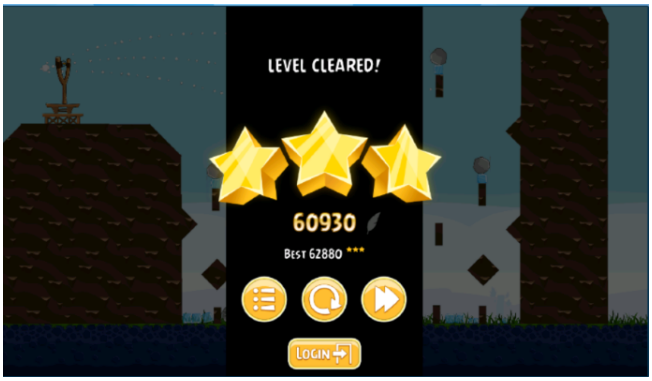

Figure 4: Results for Level 3-3 (Level 45)

The results are promising even with short training time. It can be seen from Figure 4, our agent performs well even in harder tasks in Level 3-3. Right now, the online training time of one level is 2 hours. Thus, we believe with collecting more data and training for long enough time, the performance of our agent should be further enhanced.

6 Future Work

From our knowledge, the experiment presented in this paper is the first attempt to apply deep reinforcement learning in

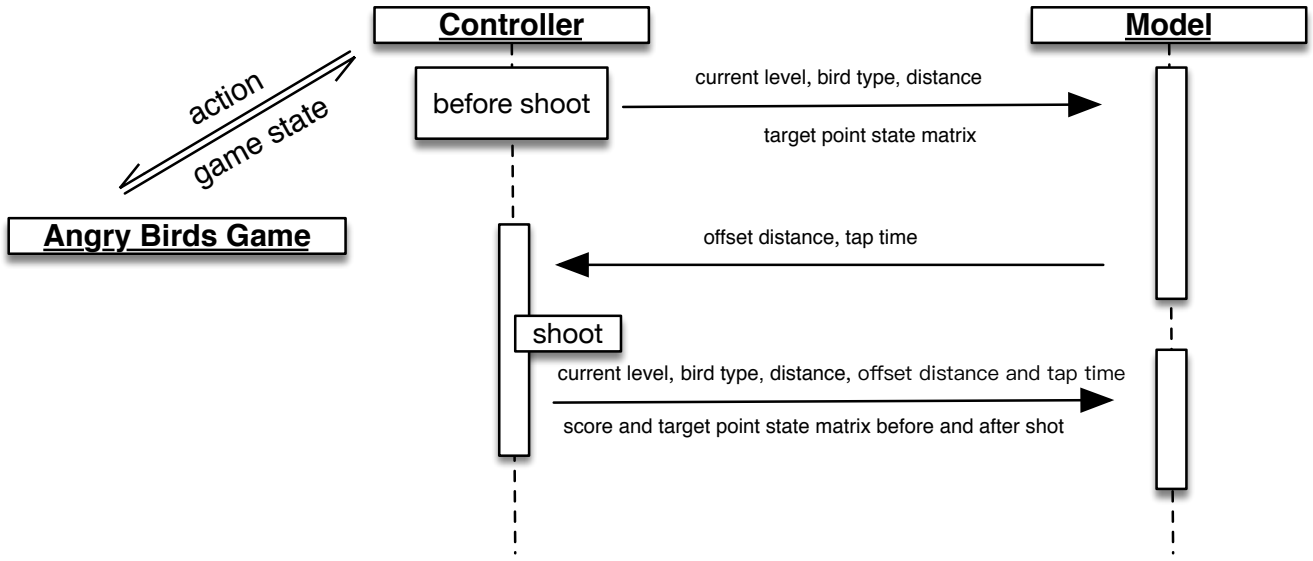

Figure 3: Model Training Architecture

Angry Birds AI Competition, which uncover the great potential in the application of deep reinforcement learning in similar games.

In this paper, we use several previous successful agent to collect high quality training data and train our model with DDPG algorithm to predict the target point. Further investigation is needed to determine whether this approach works well if we replace global features with local features.

Another challenge is to provide the RL model with more learning dimensions, e.g., the prediction of the damage of objects acting on each other. With more training data and better algorithms, We expect the AI to play more intelligently.

7 Conclusion

Playing Angry Birds intelligently was a challenging problem for machine learning. We experimented a new approach to play Angry Birds by applying deep reinforcement learning. While our agent outperforms several previous successful agents, there is much room for improvement considering many opportunities in future work. Integrating with superior algorithms and effectively collecting data play a key role on the achievements of our experimental methodology, and our experimental results could be a baseline for deep reinforcement learning in Angry Birds.

References

[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.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 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.

[Nuradiansyah *et al.*,] Adrian Nuradiansyah, Emira Ziberi, Satyadharma Tirtarasa, and Lukas Schweizer. Seabirds: An ahp approach to solve the angry birds ai challenge.

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

[Polceanu and Buche, 2014] Mihai Polceanu and Cédric Buche. Towards a theory-of-mind-inspired generic decision-making framework. *arXiv preprint arXiv:1405.5048*, 2014.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[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.

[Wałega *et al.*, 2014] Przemysław Wałega, Tomasz Lechowski, and Michał Zawidzki. Qualitative physics in angry birds: first results. 2014.