Swing Copter Bot - A Reinforcement Learning Task

Chitesh Tewani CTEWANI@IU.EDU
School of Informatics and Computing
Indiana University
Bloomington, IN 47405, USA

Arpit Agarwal ARPIAGAR@IU.EDU
School of Informatics and Computing
Indiana University
Bloomington, IN 47405, USA

Editor: Adam White

Abstract

Swing Copters is an arcade video game where the player controls a character, attempting to fly between rows of pipes attached with swinging hammers without hitting them. We wanted to use reinforcement learning techniques in Arcade Learning Environment since its a challenging task, rich in state space and difficult in policy-selection. The paper discusses application of Reinforcement Learning techniques to the problem. We present an empirical study of effect of different reinforcement learning algorithms, comparing results with various models.

We evaluated our result against random player, human player and a heuristic player and we were able to outperform them. Our results proves that Q-learning demonstrates capability and future in solving tasks in which agent learns from experiences (model-free world).

Keywords: Swing Copter, Reinforcement Learning, Markov Decision Process, Q learning, Expected SARSA

1. Introduction

We decided to implement a easy to play, hard to master game and ended up selecting Swing Copter on the basis of popularity of game and availability of game engine. Swing Copters is the second most popular game created by Dong Nguyen, on the heels of the wildly successful Flappy Bird. Once the copter starts moving in one direction it gains momentum and is hard to get control of its flight path. This makes the gameplay really hard as it is difficult to predict when you should tap to avoid collision. Tapping the screen toggles the direction of the helicopters movement between left and right. The helicopter must avoid collision with the side walls, pass through gaps and dodge swinging hammers. The game is scored based on the number of gaps successfully passed through.

We implemented a Reinforcement Learning agent on an existing JavaScript framework of the game. This Reinforcement Learning task can be treated as the Markov Decision Process (MDP) as it satisfies the Markov property. As each game of Swing Copters can be viewed as an individual episode with arbitrary length, this problem can be considered as an indefinite-horizon, Episodic Markov Decision Process.

We used Q-Learning and Expected SARSAλ to train our agent. We also built a bot taking actions with equal probability, along with an artificially intelligent agent which heuristically determines the actions to perform, given the current state. We present an empirical study on our findings by tuning learning parameters and comparing results between various reinforcement learning models.

2. Background

The Cart-Pole Balancing reinforcement learning task studied by Barto et al. (1983), is similar to our problem. In pole balancing task, the objective is to apply forces to a cart moving along a track to keep a pole hinged to the cart from falling over. Their learning system consisted of two neuron like adaptive elements, single associative search element (ASE) and a single adaptive critic element (ACE). The ASE constructs associations between input and output by searching under the influence of reinforcement feedback and in ACE computation of value function is done by a reinforcement algorithm. The results of this neuron like adaptive system outperformed the entire simulated neural networks systems.

The earliest known methods in reinforcement learning converged slowly even on simple learning tasks. Lin (1992) investigated methods that will speed up reinforcement learning methods. A comparison of adaptive heuristic critic (AHC) Barto et al. (1983), Q-learning Watkins and Dayan (1992) and three extensions of experience replay, learning action models for planning and teaching. An experiment conducted on a moderately complex task-survival in a dynamic environment is related to our problem. In the experiment, agent has multiple goals - to maximize the food collected and avoid being caught by enemies (stochastic environment). A reinforcement learning agent in form of ACH, Q-learning and extensions with coarse coding of environment was used on this test-bed. An empirical results of the experiment are promising for building autonomous learning systems using the formulated extensions.

In the domain of decision making, Stone et al. (2005) describe a multi-agent reinforcement learning application for *keepaway* subtask of RoboCup simulated soccer. They formulated the task using episodic semi-Markov Decision Process version of Sarsa (λ) with linear tile-coding function approximation and replacing eligibility traces. Their work provides empirical results of experiments with range of value of each parameter and demonstrating learning of multiple agent through small number of trials.

One of the most recent and successful case study related to our work is playing Atari 2600 games using Deep Q-Networks(DQN) Mnih et al. (2013). It is a challenging reinforcement learning application with high dimensional visual input (210 x 160 pixels raw Atari frames with 128 color palette) with varied tasks and problems. It implements convolutional neural networks with a variant of Q-learning algorithm along with stochastic gradient updates. In this the agents experiences is stored at each time-step, $s_t, a_t, r_t, s_{t+1}, a_{t+1}$ in a data-set D = $e_t, ..., e_N$ collected over episodes. They draw random samples of experience from memory and apply Q-learning updates. After the updates the agent selects and executes an action based on ϵ-greedy policy. The experience of fixed length is given as an input to neural networks.

The Deep Q-Networks algorithm is efficient as every experience updates multiple weights. Randomising the samples break correlation between consecutive samples and reduces the

variance updates. However, the algorithm can only store the last few experiences and important experiences are over-written by recent lesser important experiences prove to be the limitation of the algorithm. Deep Q-Networks (DQN) outperformed all the previous reinforcement learning algorithms on multiple Atari Games.

The limitation of Mnih et al. (2013) is overcome by use of prioritize experience replay in Deep Q-Networks (DQN) Schaul et al. (2016). The algorithm uses Stochastic sampling method which interpolates between pure greedy prioritization and uniform random sampling. Using stochastic prioritization and importance sampling instead of uniform random sampling with Deep Q-Networks, outperformed on 42 out of 57 games, with an increase in the median normalised performance across the 57 games from 67% to 97%.

The paper Guo et al. (2014) talks about developing an online real-time playability of a model-free reinforcement learning agent. This method retains the deep learning advantage of not feeding hand crafted features. For training the UCT-agent they used 300 as maximum-depth and 10000 as number of trajectories. One of methods that produced best result is UCTtoClassification-Interleaved. It collects data from 200 agents run and train the CNN via multinomial classification. This trained CNN is now used to decide action choices for next 200 runs. This process is repeated until there a total of 800 runs worth of data. This agent uses the final CNN-classifier learned by this training procedure to select actions during testing. The only drawback of this method is that it takes several days to generate the data to train the UCT-agent; however, it outperforms Deep Q-Networks significantly.

3. Environment Dynamics

The Game engine simulates the game playing Swing Copter on the browser with real time graphics.

It provides the current position of the copter, swinging hammers and platform with respect to the walls on the side, along with the direction in which swing copter is moving.

3.1 State Space

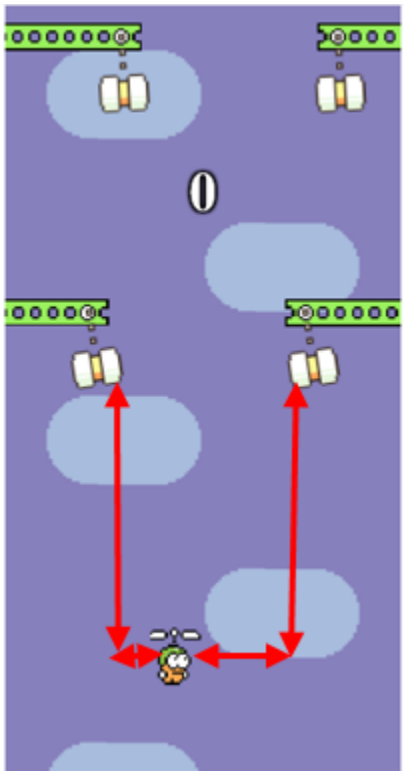
We considered the following state space: Horizontal distance from the left hammer, Horizontal distance from the right hammer, Vertical distance from the nearest platform and Moving direction of the copter. Manhattan distance metric is used.

The distances are in a continuous domain, we round the number to closest integer. This helps to discretize the state and also reduce the state space.

However, we are not choosing the distance from the walls on the side, as combination of negative and positive distances from the hammer helps the agent to determine the distance from the walls.

As we are discretizing from a continuous domain, the size of state space is really large. It scales to $180 * 180 * 80 * 2 \approx 5184000$, where 180 is the farthest distance from hammer; 80, the farthest distance from the platform and 2 movements. Considering the state and action pairs, twice of state space value is used for computation.

We have reduced the state space in two ways. First, the copter takes an action in multiple steps, which gives a more human-like playing and also saves computation time. Second, we merged the nearby pixels, which further reduces the state space.



3.2 Reward

Reward is based on number step taken. The environment gives a reward of +1 if the copter is still flying and -1000 if the copter collided.

3.3 Action

Game has two types of actions - click or dont click. With a "click", the direction of the copter is flipped.

4. Methodology

Our empirical study involves the use of multiple agents.

4.1 Random and Heuristic Agent

Random agent, picks an action at every time step with equal probability.

Another agent attempts to survive using a simple heuristic of deciding the action based on horizontal distances from the hammer and his current direction.

We consider the random

4.2 One-step Q-Learning

We implemented a reinforcement learning agent using off-policy TD control algorithm of Q-learning Watkins and Dayan (1992). It solves the reinforcement learning task by using

experiences from the environment instead of creating a model of environment. The main idea behind this algorithm is to learn optimal action-value function (q*) by updating action-value function $Q(s, a)$ using the following equation -

$$Q(S_t, A_t) \ = \ Q(S_t, A_t) \ + \ \alpha[R_{t+1} \ + \ \gamma * \max_{\alpha} Q_{S_{t+1},a}] \ - \ Q_{S_{t+1},A_t}$$

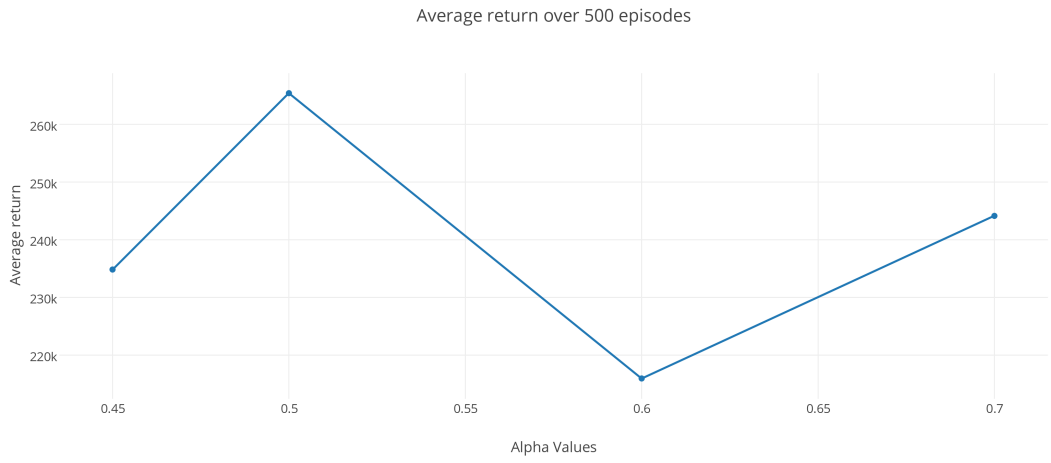
Q-learning is an off-policy algorithm and chooses best action that maximizes the return.

4.3 Expected SARSA(λ)

We implemented another reinforcement agent using Expected SARSA(λ) using eligibility traces. The algorithm used create a model of environment and leads to convergence earlier than Q-learning however Q-Learning converges at much lower values.

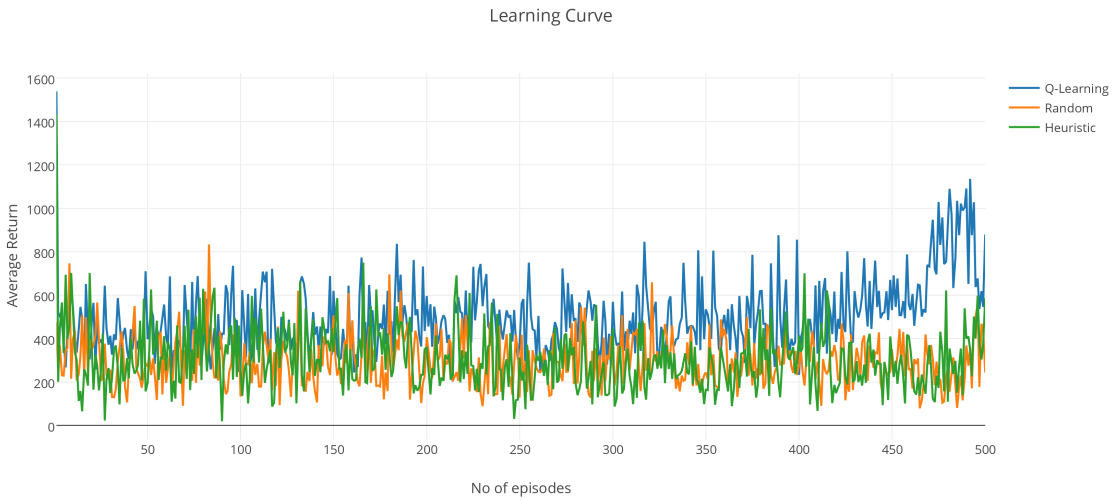
5. Empirical Result

Initially, we sampled different values of α with $\epsilon = 0.1$ and $discount factor(\gamma) = 1$. We performed experiments for 500 episodes over 10 runs.



Using $\alpha = (0.45, 0.5, 0.6, 0.7)$, we averaged return in each episode for 500 episodes. For $\alpha = 0.5$, we comparatively got better return and we used this value of α in future experiments.

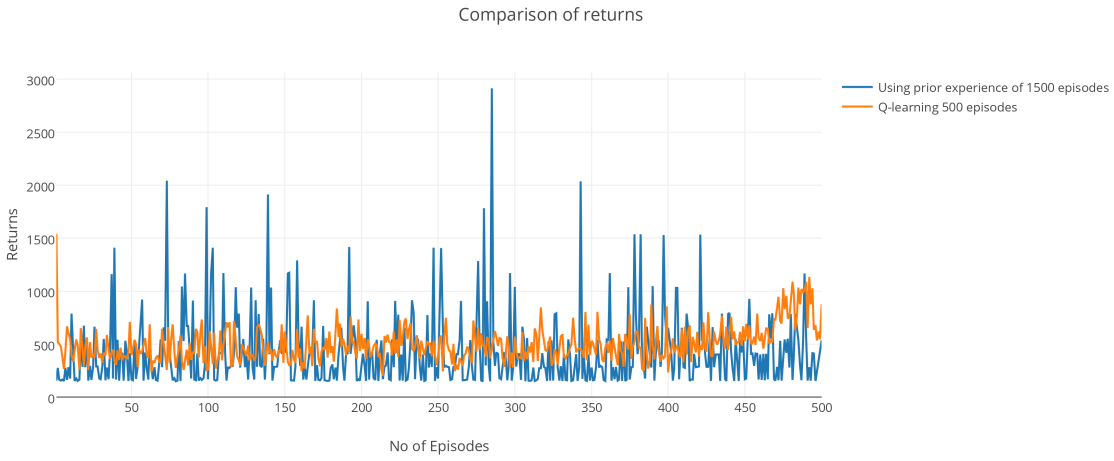
We performed experiments on the random agent, heuristic agent and off-policy Q-learning agent with $\alpha = 0.5, \gamma = 1 and \epsilon = 0.1$. It can be seen Q-learning agent outperformed both the agents, as it receives more average return per episodes relatively.



Most of the time, heuristic agent performs better than random agent, but in some cases the random agent simply stays in middle of the screen and due to stochastic behavior of platform creation, it survives for a longer time.

One can observe from the learning curve that, towards the end of 500 episodes, Q-learning agent performs much better to gain higher returns.

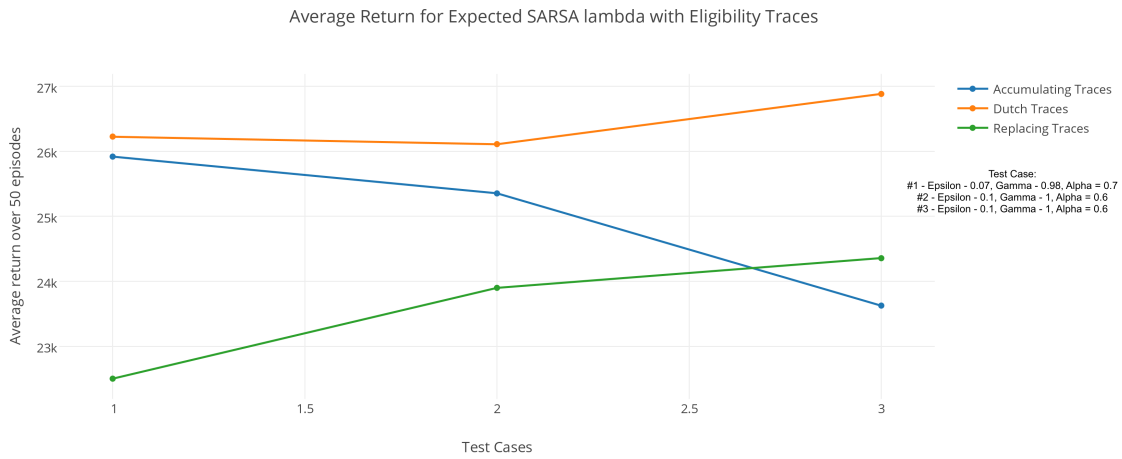
In another set of experiment, we let our Q-learning agent learn for 1000 episodes and then turned-off learning for next 500 episodes, during which it relied on the policy that it as developed.



The average returns from our learned Q-learning model outperformed the previous model. It has many higher peaks of survival time through return and scored as high as 32 levels.

We experimented different Eligibility Traces on our Expected-SARSA(λ) agent. As our state space is large, we were never not able to run over 50 steps, as it takes a lot of time to perform updates.

We performed 3 test cases on our Expected SARSA λ using different combinations of $\alpha, \epsilon and \epsilon_\mu$, with $\lambda = 0.01$.



The results showed that Dutch Traces outperformed Replacing Traces and Accumulating Traces in all the three test cases.

6. Limitations

One of the limitations is that we are evaluating action on every step however, in real time scenario or human play, one can only take action after certain steps.

We are not getting direction of swing of hammers from the environment, this makes it really difficult to learn.

Since our Game framework is browser based and hence our state space differ with resolution of screen.

7. Future Work

Use of linear function like Tile Coding Sherstov and Stone and non-linear function Function approximation algorithm will help us to approximate our action-value function.

We would also like to implement Deep Q-Learning Mnih et al. (2013) to this domain as this was the main motivation for our work.

Culminating our findings, we would like to publish our work to The Journal of Machine Learning Research (JMLR) which is an international forum for the electronic and paper publication of high-quality scholarly articles in all areas of machine learning.

8. Conclusion

From the experiments on reinforcement learning agents like Q-Learning and Expected SARSA (λ), we analyzed the advantages and short-comings of both the method. Though, the environment was a continuous domain, discretizing it helped to implement the algorithms.

Both our agents could managed to defeat the human average of in the game of Swing Copters.

References

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):835–846, 1983.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag.

Peter Stone, R. S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *International Society for Adaptive Behavior*, 13(3):165–188, 2005.

Christopher J. C. H. Watkins and Peter Dayan. Technical note: q-learning. *Mach. Learn.*, 8(3-4):279–292, May 1992. ISSN 0885-6125.