Discretized Dynamic Point Mass Control with Reinforcement Learning

Wei Tao Chen

Introduction

This project is an empirical evaluation on discretized dynamic point mass control with reinforcement learning. “Point mass” is a simplification of real objects with infinitesimally small size but non-zero mass. “Dynamic” refers to motion of objects under the influence of forces according to classical mechanics. “Discretized” means the position, velocity, and acceleration of the objects are discrete. Therefore, “discretized dynamic point mass control” is really just controlling the motion of a pixel object with forces on a 2D grid plane.

The reason for choosing this problem is twofold. First, the problem is simple enough for a tabular method, but difficult enough to benefit from approximation method such as deep learning. This allows a direct and fair comparison between tabular method and approximation method. In particular, Q-learning and deep Q-network (DQN) were compared. The second reason is that the problem is versatile. It allows the environment to be changed easily while getting totally different properties. The three environments studied are flappy bird, lunar lander, and racetrack. The same two algorithms were used to solve all three problems. Many real life problems, such as satellite control, can be simplified to point mass control.

Environment

Since there is no suitable environment available, a custom environment was created. Everything in the environment is based on a 2D grid. On the grid, a point mass can move from one cell to another. A cell can also be an obstacle blocking the movement of the point mass. Since the point mass is dynamic, the state is not only its position on the grid but also its velocity. In order to apply tabular method, the velocity is bounded from -5 to 5. This is acceptable because on a small grid, it is difficult to accelerate out of the velocity bound without hitting an obstacle or moving off the grid.

The dynamics of the point mass is represented through the state transitions. The complete equations of motion is shown below.

$$v2 = v1 + a * t$$
$$r2 = r1 + v1 * t + 1/2 * a * t^2$$

Where r is the position vector, v the velocity vector, [r1, v1] the current state, [r2, v2] the next state, t the time step, and a the acceleration. To simplify the calculations, each step takes one time unit. The second

order term is dropped to keep all the state values integers. The resulting state transition with bounded velocity is as follow.

$$v2 = v1 + a$$
$$v2 = clip(v2, -5, 5)$$
$$r2 = r1 + v1$$

Input forces are applied to control the motion of the point mass. To simplify the calculations, point masses have one unit mass. Thus, accelerations can be directly applied. The action space has four actions, one for acceleration in each direction. An exception is the flappy bird problem, where horizontal velocity is fixed. Please refer to the problem description for more detail.

The reward function is different for each problem. In general, the point mass start moving from a starting state, and a negative one reward is given for each step taken. The episode ends when the point mass reaches the goal state or when a maximum of 400 steps has passed. If the point mass hits an obstacle or moves out of bound, it is reset to a starting state. This encourages the agent to reach the goal state as soon as possible to accumulate the least amount of negative rewards.

Related Work

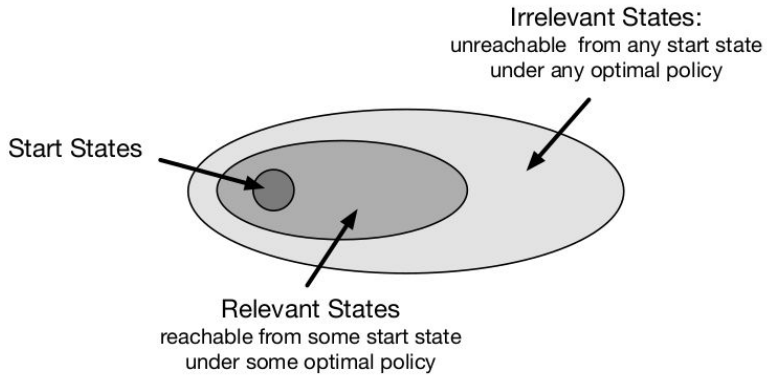
This project is inspired by a racetrack problem in in chapter 5.7 of the textbook “Reinforcement Learning: An Introduction” [1]. The author then compared solving it with dynamic programming and real-time dynamic programming (RTDP) in chapter 8.7. In a paper by Deepmind, DQN surpassed human performance in Atari games [2]. The game of flappy bird was well studied under classical control and machine learning control setting [3]. Specifically, bang-bang controller, SVM, and value iteration with discretized state were compared. Deep reinforcement learning (DRL) techniques, such as actor-critic, can be applied to continuous control [4]. It demonstrated driving a car with continuous inputs in “torcs” simulator. Prioritized sweeping is another technique intended to improve data efficiency [5]. Instead of the usual temporal difference update, prioritized sweeping starts from the goal state and updates the Q-values backwards at the relevant states.

Techniques

The two techniques chosen to tackle the problem are Q-learning and DQN. Q-learning is a tabular method, so the Q function is a table spanning the entire state-action space. In our case, the state-action space spans [x-position, y-position, x-velocity, y-velocity, action]. The learning rate is 0.9. Such a high learning rate is desirable because there is little randomness in the environment. Also, due to the large state space, a high learning rate speeds up convergence.

By including the velocity, the size of the state space can grow very fast. Even with a small 10 by 10 grid, the state space is [10 x 10 x 11 x 11]. (Velocity ranges from -5 to 5.) Fortunately, most of the states are irrelevant as shown in the figure below. For example, in the racetrack problem, it is impossible to reach a

high velocity in a tight turn. Even though a exact description of the environment is given and dynamic programming could be used, trajectory sampling methods such as Q-learning are much more efficient, since it does not waste time exploring irrelevant states. [1]



DQN is a approximation method based on Q-learning. In particular, it uses an artificial neural network to approximate the Q-function. [2] The architecture of the neural network contains two fully connected layers of 256 neurons, and a third layer connects to each action. The learning rate is 0.001.

For both Q-learning and DQN, action selection is performed with epsilon-greedy with a constant epsilon of 0.1. A decay of epsilon could potentially improve convergence, but a constant epsilon ease the comparison. Both techniques has a discount factor of 0.99.

Although it is possible to solve these problem with rule-based approach, reinforcement learning is able to solve all three problems with a single algorithm. Beside having the advantage of trajectory sampling, Q-learning and DQN are similar to each other algorithmically. This allows the comparison to focus on the representation of the Q function and show the advantage and disadvantage of deep learning approximation over traditional tabular-based approach.

Evaluation

For evaluation, the three problems studied are flappy bird, lunar lander, and racetrack.

Flappy Bird

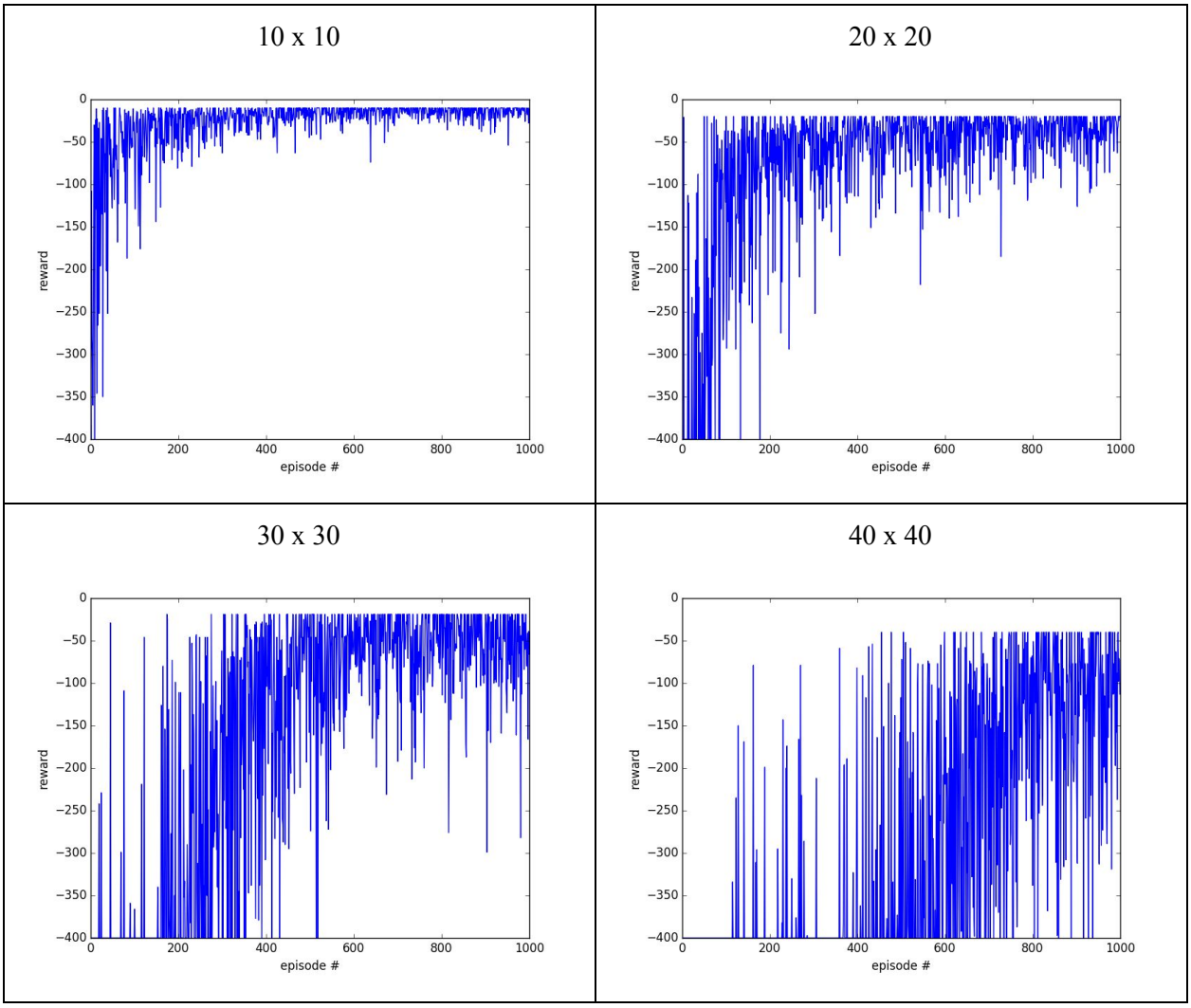
The first problem is inspired by the popular mobile game “Flappy Bird”. The goal of the game is to maneuver a bird through a series of narrow tunnels. The control is simple and only involves two actions. One is to flap its wings and gain a upward acceleration; Two is do nothing and the bird will drop due to gravity.

This game was discretized onto a grid map. The example plays illustrates the game setup. A column of obstacles with a narrow opening is located near the right side. A point mass representing the bird starts on the leftmost column with [1, 0] velocity pointing to the right. The horizontal velocity will not change and thus the bird moves right one column in each time step. The first action applies [0, 4] acceleration upwards, while the second action does nothing. A gravity vector of [0, -1] pointing downwards is added to the acceleration in each time step. A negative one reward is applied until the bird safely go through the opening and reach the rightmost column. If the bird hit an obstacle or move out of bound, it is reset to a random position on the leftmost column.

Training

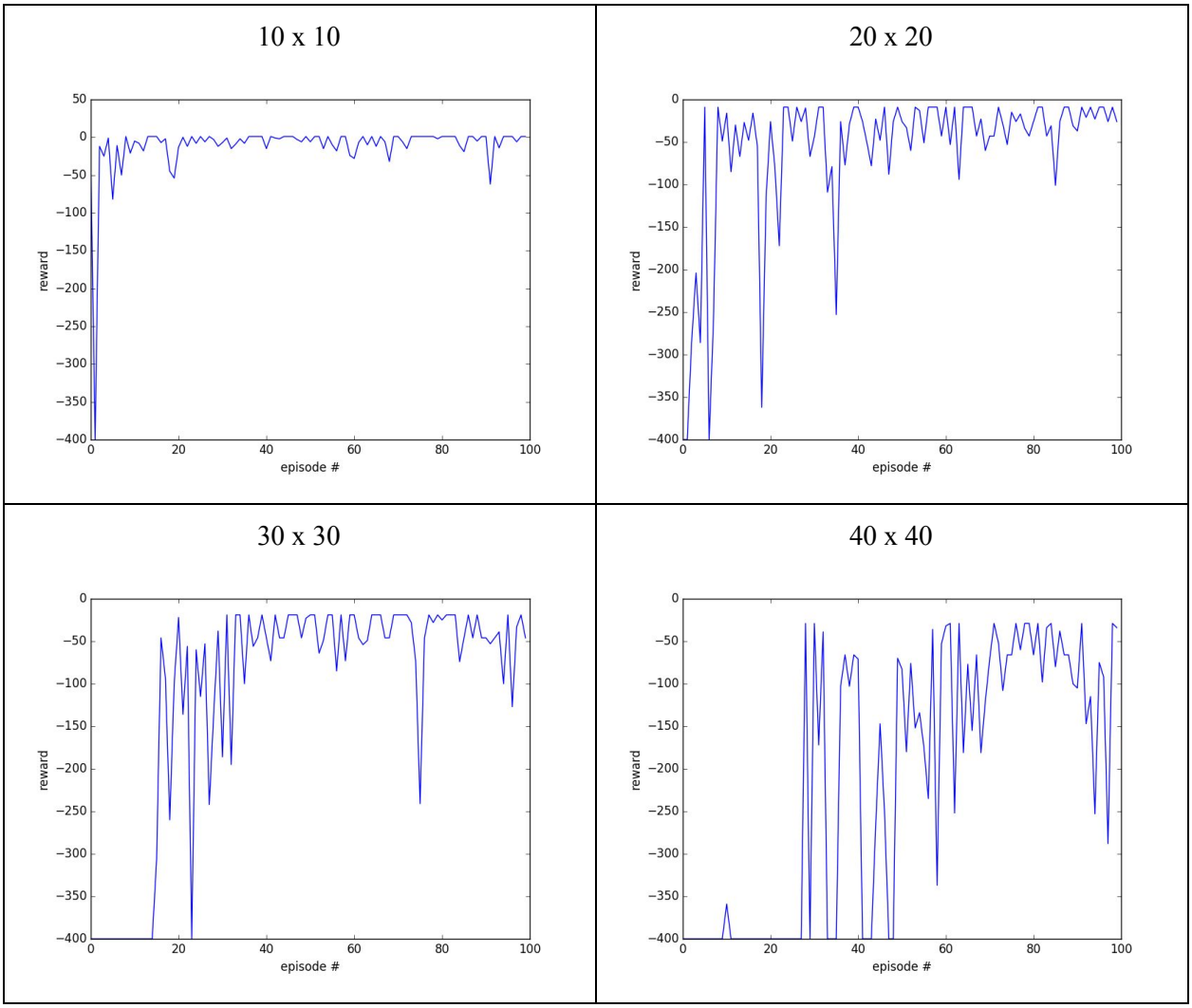
Four experiments were run, each with increasing grid size from 10 by 10 to 40 by 40. The following are the plots of accumulated rewards in each episode with Q-learning. The maximum reward in one episode is the negative length of the grid, representing a successful run through the grid. Anything less than that means the the bird hit an obstacle or moved off bound and was reset to starting state.

Reward vs Episode # with Q-learning



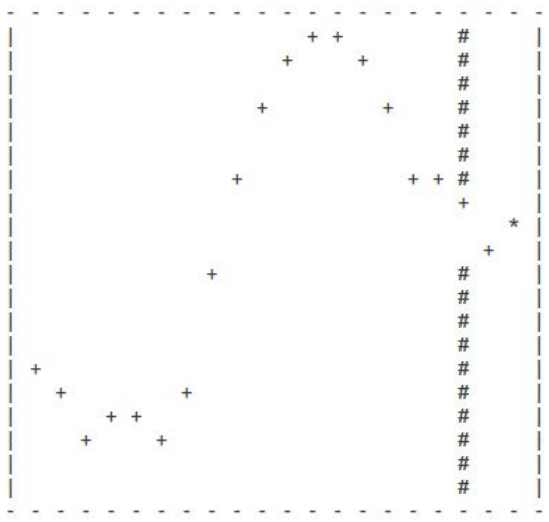
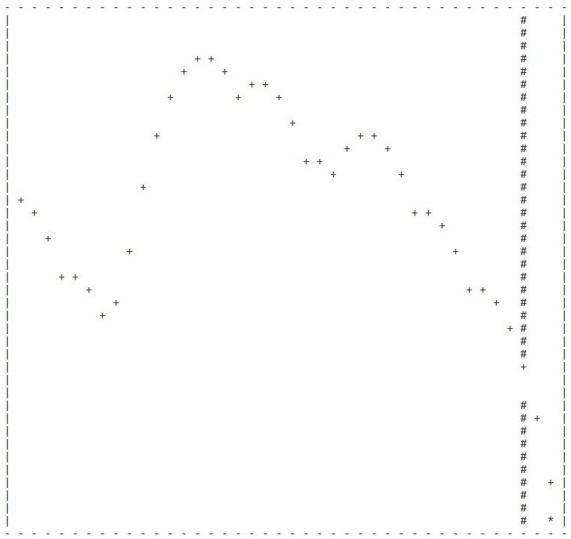
As shown by the plots, the performance deteriorates quickly as the size of the map increase. Following are the results with DQN. Only 100 episodes were run compare to 1000 episodes with Q-learning. DQN is able to learn at least 10 times faster than Q-learning, and the performance does not deteriorate much as the map size increases. Overall, DQN dominates Q-learning in this flappy bird problem.

Reward vs Episode # with DQN



Example Plays

Both techniques learned to play the game eventually. Here are some examples of successful episodes. The "#" sign represents an obstacle, and the "+" sign represents a past position.

20 x 20	40 x 40
	

Example Policies

Here are the policies of Q-learning and DQN on the same 20 by 20 board with a velocity of [1, -2]. 1 represents flapping and 0 doing nothing. Intuitively, since the bird is dropping, one should flap when the altitude is low. Q-learning has a very sparse policy matrix, while DQN has smooth boundary between 1's and 0's. Although both policies are able to play the game, the policy produced by DQN is more intuitive. The policy of Q-learning places 1's at crucial places but does not generalise to regions not well explored. DQN is able to generalize in unexplored regions and take advantage of high correlation of Q-values between neighbouring states.

Policy on 20 by 20 Board with Velocity of [1, -2]

Q-learning

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

DQN

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1]
 [1 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0]]
```

Cheating behaviour

An interesting observation is that on larger maps, Q-learning exhibit behaviour where it “commits suicide” early in an episode by intentionally moving off boundary. This gives it a chance to restart an episode quickly and begin in a new random starting position hoping it would be easier.

Lunar Lander

The second problem is inspired by one of OpenAI gym environment. It has an empty map with no obstacle. The lander spawns at a random cell in the top row with zero initial velocity. The goal is to land at a cell on the bottom row with zero velocity. The four action correspond to plus one acceleration in each direction. This game is potentially harder as the chance of randomly stumbling on the goal position with zero velocity is very low. The reward plots and example plays are shown in the appendix.

Overall, DQN performed better than Q-learning. Similar to flappy bird, the correlation between Q-value of neighbouring states is high, and DQN is able to generalize to regions not well explored. Also, It was observed that giving a small +10 reward after reaching the goal improved the performance of both techniques.

Racetrack

The last problem involves maneuvering a car on racetrack. The car starts at a random position on the left column and needs to navigate through a narrow track with tight turns. Moving off bound at the right border is considered as reaching the goal. Similar to lunar lander, The four action correspond to plus one acceleration in each direction. The reward plots and example plays are shown in the appendix.

Surprisingly, DQN did not learn at all with the current hyperparameters. It seems to be stuck in a tight corner and oscillate back and forth. The reason is that, due to the narrow track and tight turns, the Q-values are highly nonlinear and have low correlation with the neighbouring states. DQN is over-generalizing. On the other hand, Q-learning is able to learn quickly after stumbling on the goal. Despite the large state space, the reachable state space is much smaller as most of the map is covered with obstacles. Tabular approach such as Q-learning benefits from the reduced reachable state space and high Q-value nonlinearity.

Conclusion

From the empirical evaluations, there is no best technique. In settings with large state space, such as flappy bird and lunar lander, DQN generally perform better than Q-learning as it is able to generalize Q-value to unexplored regions. However, it is important to distinguish total state space with reachable state space. In settings with small reachable state space and high Q-value nonlinearity, such as racetrack, Q-learning can still outperform DQN. In addition, DQN takes much more computing power to train and

require more tuning, while Q-learning is easier to use. Thus, caution is advised when blindly applying deep learning approximation methods.

As the experiments have shown, reinforcement learning techniques such as q-learning and DQN are successful at discretized dynamic point mass control. The three problems investigated are simple enough to be considered solved. Although, if the rewards are sparse and the chance of randomly stumbling on the goal state is low, it might take a long time before any improvement is made. Techniques such as reward shaping should be considered.

For future research, DQN on the racetrack problem can be tuned to encourage more exploration. More scenarios can be explored. Other approximation techniques such as linear methods can be compared. The environment can be easily extended to be three dimensional or continuous. For any rigid body control problem, point mass control is a good starting point.

References

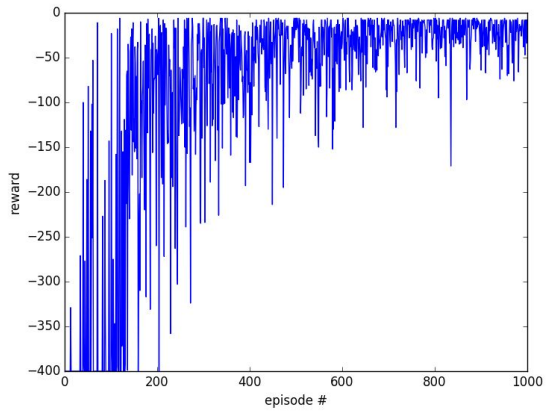
1. Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.
2. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
3. Shu, Yi, et al. "Obstacles Avoidance with Machine Learning Control Methods in Flappy Birds Setting." *Univ. of Stanford, CS229 Machine Learning Final Projects Stanford University*(2014).
4. Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971*(2015).
5. Moore, Andrew W., and Christopher G. Atkeson. "Prioritized sweeping: Reinforcement learning with less data and less time." *Machine learning* 13.1 (1993): 103-130.

Appendix

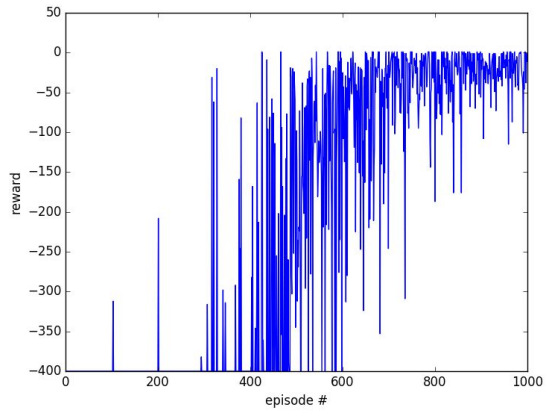
Lunar Lander

Reward vs Episode # with Q-learning

10 x 10

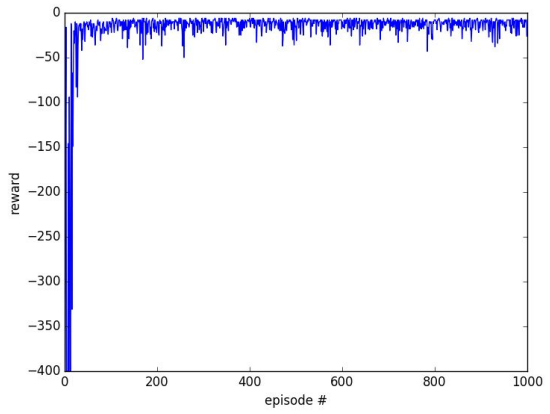


20 x 20

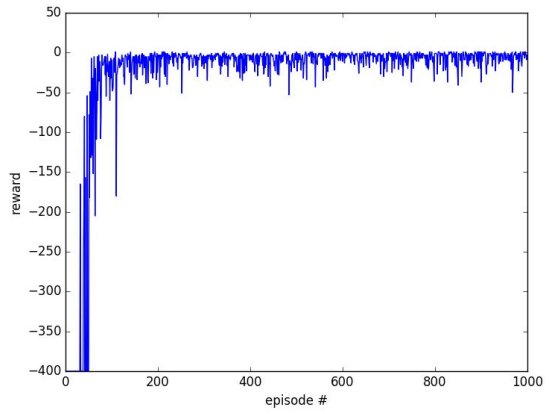


Reward vs Episode # with DQN

10 x 10

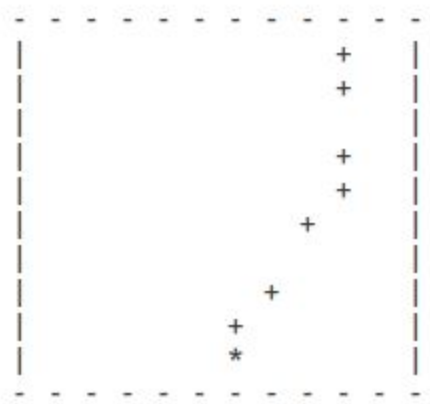


20 x 20

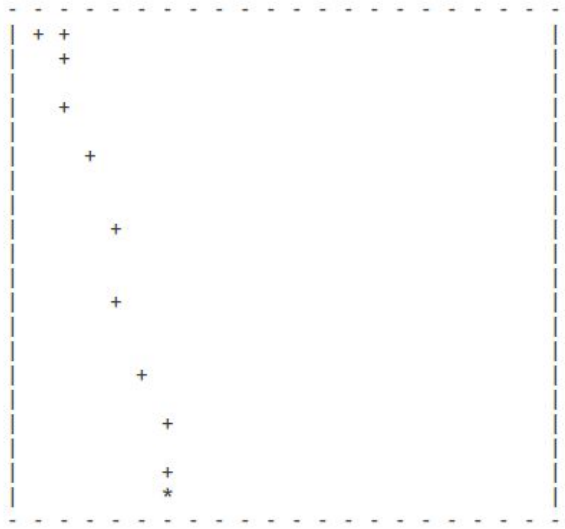


Example Plays

10 x 10

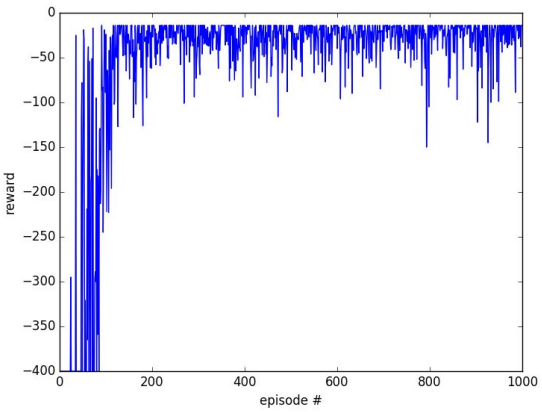
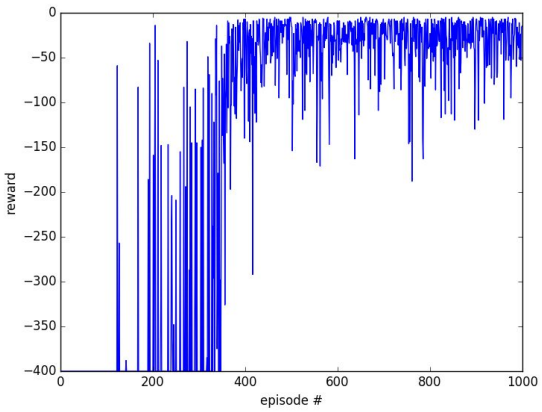


20 x 20



Racetrack

Reward vs Episode # with Q-learning

10 x 10	20 x 20
	

Example Plays

10 x 10	20 x 20