Final Project - Q Learning To Balance Inverted Pendulum

Geoffrey Clark
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: gmclark1@asu.edu

Venkatavaradhan Lakshminarayanan
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: vvlakshm@asu.edu

Shubham Sonawani
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, AZ
Email: sdsonawa@asu.edu

***Abstract*—This paper works to develop an on-line reinforcement learning framework. This on-line learning framework learns via the model free reinforcement learning method: Q-learning. The on-line learning policy improves overtime by finding the optimal action policy for each output. The learned policy will be judged against comparable policies in literature in order to analyze the utility of our method, with regard to learning speed and learning success rate. The learning policy will be tested on a single-unit cart pole balancing simulator.**

***Index Terms*—Q-learning, Cart-pole, Inverted Pendulum, On-Line Learning, Open AI gym**

I. Introduction

Online reinforcement learning [1][2] has held great appeal for the learning of dynamic control policies. The model free algorithm Q-learning has especially been shown to be quite adept at solving these types of problems. In this work we analyze the use of q-learning in [3] to create a control policy for the cart-pole problem. Q learning is a type of reinforcement learning that learns the system behavior by trial and error. Over time rewards are propagated through the algorithm and developed into a heuristic policy.

Our main objective is to design a controller for the balancing of an inverted pendulum (cart-pole problem) using reinforcement learning algorithm Q-learning. Initially, we started of by simulating the inverted pendulum dynamics in Openai-gym. Existing Q-learing algorithms provided inspiration for our solution, which was solved using the dynamics engine from openAI gym. However, the desired dynamics were different from the dynamics provide in the cart-pole model in Openai-gym. Thus, we have modified the source code of cart-pole model which contains all the equations and the necessary parameters according to Barto and Sutton's simulation [3].

Our proposed solution is very close to a traditional Q-learning approach except that we modify the learning and exploration rates before every trial. This modification to the traditional Epsilon greedy type policy help the policy to converge to an optimal solution faster than constant learning and exploration rates. Both the learning and exploration rates are very high at the beginning but as the algorithm learns they are moved lower and lower until the policy is only exploiting instead of exploring the policy. In the next few sections we talk further about the algorithm and how it performed in real world tests with differing levels of both actuator and sensor noise. Finally we will discuss the pros and cons to this approach.

II. Q-Learning

Q-learning is a model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. Additionally, Q-learning can handle problems with stochastic transitions and rewards, without requiring any adaptations. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable.

1) Algorithm: The learning algorithm consists of an agent capable of interacting with the environment, with states S and a set of actions per state A. By performing an action a $\in$ A, the agent can move from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score). The goal of the agent is to maximize its total reward. It does this by learning which action is optimal for each state. The action that is optimal for each state is the action that has the highest long-term reward. This reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state, where the weight for a step from a state $\triangle$t steps into the future is calculated

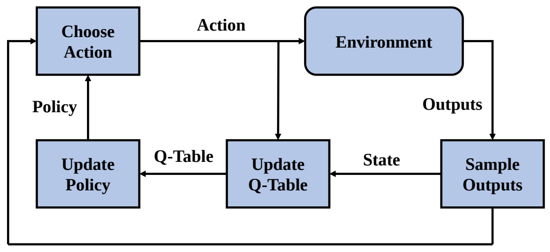

Fig. 1. Q-Learning system model

as $\gamma^{(\triangle t)}$. Here, γ is a number between 0 to 1($0 \leq \gamma \leq 1$) called the discount factor and trades off the importance of sooner versus later rewards. γ may also be interpreted as the likelihood to succeed or survive at every step $\triangle$t.

The algorithm therefore has a function that calculates the Quality of a state-action combination : $-\ Q : S \times A \rightarrow \mathbb{R}$ Before learning has started, Q returns an (arbitrary) fixed value, chosen by the designer. Then, at each time t the agent selects an action a_t and observes the reward r_t and a new state s_{t+1} that may depend on both the previous state s_t and the selected action, Q is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and new information :

$$Q(s_t,a_t) \leftarrow (1\text{-}\alpha) * Q(s_t,a_t) + \alpha * (r_t + \gamma * max_a\ Q(s_{t+1}))$$

where r_t is the reward observed for the current state s_t, and α is the learning rate ($0 < \alpha \leq 1$)

It should also be noted that this cart-pole balancing problem does not have finite states. Therefore, we had to modify the agent's observations. As the states grow, the probability to traverse that state again starts to approach 0. This can be eliminated by two methods; 1. Reward shaping 2. Function approximation or Discretization. We followed the second method and discretized the observations. The number of discrete states for each observations $(x, \dot{x}, \theta, \dot{\theta})$ are 1, 1, 8, 3 respectively. This solved the problem well and the agent was able to converge to optimal policy sooner.

An episode of the algorithm ends when state s_{t+1} is a final state (or, "absorbing state"). However, Q-learning can also learn in non-episodic tasks. If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops.

The code provided is a modification of Matthew Chan's algorithm [5] for "Cart-Pole balancing with Q-learning". While we found numerous errors in his code-base it was a useful first step to understanding the problem.

2) Search Policy: The agent when deployed in an environment may take several episodes to converge to optimal policy to attain maximum reward. Determining a good search policy determines the convergence rate. In our work we used a epsilon-greedy approach which takes care of the exploration-exploitation trade off. On a high level, we let the agent to explore the environment and do not care about obtaining maximum reward at every time step until it reaches some threshold value. Once it reaches the minimum exploration value, the agent is no longer allowed to explore and it has to start exploiting the environment (i.e) look for maximum rewards in each time step. Using this approach we choose the action and the policy as formulated below, where t is the time step, α is the explore rate and β is the minimum explore rate,

$$\alpha = max\left\{\beta, min\left\{1, 1 - \frac{log(t + 0.2))}{log(300)}\right\}\right\} \tag{1}$$

$$action = \left\{\begin{array}{cc} 0 \text{ or } 1 & \text{if } x \leq \alpha \\ max(q[state]) & \text{elsewhere} \end{array}\right\} \tag{2}$$

III. PERFORMANCE EVALUATION

The Q-learning algorithm that is proposed above has been implemented in the OpenAI gym CartPole simulation environment [5]. The cart pole problem consists of a simple inverted pendulum attached to a cart which has to balance itself. To solve the problem the agent must apply forces (either to the left or to the right), of a fixed magnitude, to the cart such that the pendulum is stabilized and the rate of change of theta is minimized. While both linear and non-linear pure controls solutions exist to this problem, we tried to incorporate a reinforcement learning algorithm which is completely model free and contains no prior information about the simulation environment or the physical system for which it is trying to find the optimal policy. The algorithm will receive reinforcement at every time step and use this to update its state action policy.

The simulation is a modified version of the OpenAI gym's CartPole-v0. This simulation encompass the mathematical representation in python of the nonlinear system described above, with the addition of friction. This simulation should match a corresponding physical system reasonably well. The state space will be sampled from the simulator to use in the algorithm, at every time step, in much the same way it would be done with physical sensors.

1) Cart - Pole Balancing: The system dynamics used in our simulation is the same as the one used in [1]

$$\frac{d^2\theta}{dt^2} = \frac{g\sin\theta + \cos\theta[-F - ml\dot{\theta}^2\sin\theta + \mu_x \mathrm{sgn}(\dot{x})] - \frac{\mu_p\dot{\theta}}{ml}}{l(\frac{4}{3} - \frac{m\cos^2\theta}{m_c+m})} \tag{3}$$

$$\frac{d^2x}{dt^2} = \frac{F + ml[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta] - \mu_c \mathrm{sgn}(\dot{x})}{m_c + m} \tag{4}$$

where

g	$9.8m/s^2$,acceleration due to gravity;
m_c	$1.0kg$, mass of cart;
m	$1.0kg$,mass of pole;
1	$0.5m$,half-pole length;
μ_c	0.0005,coefficient of friction of cart on track;
μ_p	0.000002,coefficient of friction of pole on cart;
F	± 10 Newtons,force applied to cart's center of mass;

$$\mathrm{sgn}(x) = \left\{ \begin{array}{cl} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ \text{-1} & \text{if } x < 0 \end{array} \right\}$$

The simulation environment for the OpenAI gym solves the differential equations, and provides sensor readings for,

$x(t)$ - acceleration due to gravity;
$\theta(t)$ - position of cart with respect to the center of the track;
$\dot{x}(t)$ - angle of pole with ;
$\dot{\theta}(t)$ - half-pole length;

For the purposes of our experiments a run will consist of up to 10000 episodes, and will be considered successful when a single trial lasts more than 60000 time steps(20 mins in real time), this will end the run. If the algorithm is unable to balance any trial within the run for for this number of time steps than the run will be considered unsuccessful. Our simulation environment employs a time step of 0.02s. A single trial will start around the zero position(unstable equilibrium) and will end when either the pole exceeds the range of $\pm 12°$ and/or the exceeds the range of $\pm 2.4m$ from the center of the track.

Additionally both sensor and actuator noise have been added to the simulation. Sensor and actuator noise is expresses as either uniform or gaussian noise as a percentage of the total range. Specifically the actuator noise is describes as $u(t) = u(t) = \rho$, where ρ is a uniformly distributed random variable. Sensor noise is described as either zero mean gaussian noise where the standard deviation is a percentage of the total sensor range, or uniform random noise of the type $u(t) = u(t) = \rho$, where ρ is a uniformly distributed random variable.

TABLE I
PERFORMANCE EVALUATION

Noise type	success rate	# of Episodes
Noise free	90%	268
Uniform 5% actuator	100%	268
Uniform 10% actuator	85%	265
Uniform 5% sensor	90%	272
Uniform 10% actuator	95%	274
0.1 Gaussian sensor	90%	289
0.2 Gaussian sensor	75%	507

2) Simulation Results: Inorder to test the algorithm, we utilized a Google cloud compute engine's virtual machine which was powered by an 8 core Intel Haswell CPU with 50 gigabytes of RAM. Ubuntu 16.04 LTS was running on top of this hardware with a 10 gigabytes of persistent storage. Several experiments were conducted to analyze the performance of our algorithm. The results of these experiments are show in Table II below. The results shown in Table II were created from and average of 20 runs. To create this Table II, we are using two outputs as metrics for how well and quickly the algorithm learned. These metrics are success rate across total runs, and the average number of trials needed to reach 600,000 time steps in a single trial. This first metric is used as an indicator of how well the algorithm is able to learn a complete policy under random conditions. The second metric is used to show within each run how quickly the algorithm converges to an optimal policy. The number of trials that determine a successfulness run is therefore very important here. While a low value for the number of time steps necessity to consider the trial a success will be quite easy to reach and will suffice for most real life applications, we used a considerably higher number of 600,000 time steps. Our choice representing a total run time of approximately 3.3 hours, will force our algorithm to come up with a very robust solution. A good solution is one that has a high success rate and a low number of trials needed to reach the time step goal.

Fig. 2, below presents the number of time steps for each trial against the number of total trials necessary to achieve the time step goal in a single trial. Fig. 3, below shows what the pendulum angle(theta) looks like over 1500 time steps for a successful trial. This particular trial was not subject to any noise and lasted the required 600,000 time steps.

IV. RESULTS AND DISCUSSION

This paper focuses on the application of Q-learning on the cart-pole problem. When first run with constant or linearly decreasing learning rate and exploration values we found it nearly impossible for the algorithm to learn. Q-learning turned out to be far from an optimal algorithm, but our method shows that with a small amount of hyper-parameter tuning it is possible to learn an optimal policy.

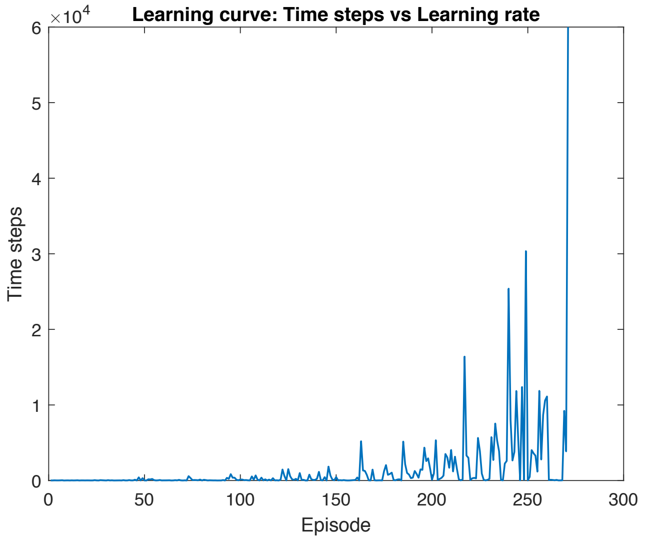

Fig. 2. Learning curve across multiple episodes

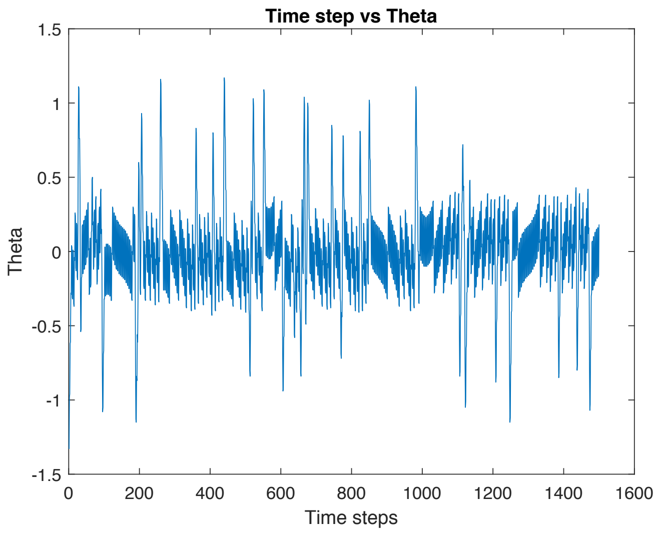

Fig. 3. Observed theta across multiple time steps

The Q-learning agent solved this problem better than expected, with no noise the algorithm was able to learn within 268 trials and most noise situations took less than 300 trials to learn the optimal policy. This shows that the learning method we propose here is very resilient against both actuator and sensor noise and perturbations. From Fig. 2 you can see that it takes many trials for the policy to start to learn but once it does the algorithm learn very quickly. This is because it takes many trials to propagate the rewards across the entire q-table(across all states). This propagation time shows that this learning method is not an optimal solution for this problem due to the number to states required. More optimal method in shown in [4] where an actor critic network is used to learn much more quickly than is possible with q-learning.

While testing the agent, we found that we could achieve better results when we decrease the minimum explore rate to 0.001. This is because, initially we let the agent to explore the environment as much as possible and then use the explored knowledge to exploit when the minimum rate was achieved. We also found that when actual noise is present the discretization states for cart displacement and cart velocity must be greater than 1 to combat the noise in the actuator.

Further, shaping the reward might have been a good option, but there might be a great loss where the agent might settle for intermediate reward and not look for the ultimate reward. Also, to improve this approach we tried to learn again from the observations and the actions by storing the data from those successful episodes by training a support vector machine to predict the actions and not guess them initially (i.e) we tried to help the agent make an educated guess. Since the time steps are really fast, training the agent on this didn't help instead it broke it. This is because building a classifier over 60000 data points takes considerable time. And on an average the agent beats its maximum time steps over a run, 10-12 times, which means for every run, the model will train on the trained data 10-12 times and also we need to keep in mind that the data set could be as large as 60000.

V. CONCLUSION

Given the problem statement, We have implemented Q-learning for controller design on the Cart-pole Problem. Our results showcases a 90% success rate in balancing the cart-pole with noise free environment on average of 268 number of trials over 20 runs. As we can see in figure-3 that learning curve of agent reached 60000 time step from 10000 time within last 60 episodes (trials). Thus, we can see Q-learning gives optimized results after much 200 trials and hence successfully designed a Q-learning agent to balancing the inverted pendulum or cart-pole problem.

REFERENCES

[1] David Silver1*, Julian Schrittwieser1*,and et al. "Mastering the game of Go without human knowledge,"2017 Macmillan Publishers Limited, part of Springer Nature.

[2] D. George,* W. Lehrach, K. Kansky and et al. "A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs," Science 10.1126/science.aag2612 (2017).

[3] Andrew G. Barto , Richard S. Sutton , Charles W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems ," IEEE Systems, Man, and Cybernetics Society, Volume: SMC-13, Issue: 5, Sept.-Oct. 1983

[4] Jennie Si, Senior Member, IEEE, and Yu-Tsung Wang, Member, IEEE,"On-Line Learning Control by Association and Reinforcement,"IEEE transaction on neural networks , Vol. 12, No. 2, March 2001

[5] Matthew Chan, " Cart-pole balancing using Q-learning," Online source: https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947