Reinforcement Learning for Dynamical Systems

Cesar Santoyo, Angel Yam, Jeremiah Coholich, and Pratik Kunapuli

1. Project Summary

In this project, we survey a variety of reinforcement learning (RL) techniques and study them in the context of dynamical systems; specifically, we address the cart-pole problem. In general, RL algorithms achieve their goal by observing a system's state and executing an action based on the expected reward given by the environment. We perform a review of RL literature highlighting existing approaches. For brevity, we limit our survey to literature relevant to dynamical systems. Furthermore, we implement a series of RL algorithms using the OpenAI's Gym [1] and Pytorch [2]. Specifically, we implement 1) a basic algorithm countering pole motion, 2) policy gradient utilizing a two-layer neural network, 3) deep Q-learning backed by a three-layer neural network, and 4) model predictive control (MPC). To conclude, we compare the individual performance of the the RL techniques against the results of the MPC.

2. Introduction & Motivation

RL is a subset of machine learning concerned with how agents act within an environment in order to maximize a defined reward function [3]. The representation of agents within their environment is formalized by a Markov Decision Process (MDP) which incorporates sensation, action, and an ultimate goal. RL is neither exclusively *supervised learning* or *unsupervised learning* [4, 5, 6]. RL is distinct from supervised learning because it cannot rely solely on labeled data – an RL agent must be able to learn through its own interaction with the environment. [7]. Unlike unsupervised learning, an RL algorithm expects a reward signal which is intentionally crafted to produce desired behavior, rather than trying to find structure in data or the environment. RL is of interest because in many practical applications such as robotics and other dynamical systems [8, 9, 10, 11], computer cluster resource management [12, 13], traffic light control [14, 15], and news recommendations [16], agents utilize sensor data to improve their decision making. This motivates the project to further study the most effective techniques for dynamical systems.

3. Preliminaries

RL problems are formalized by an MDP, in which an agent chooses actions based on its current state within its environment in order to maximize a reward function. This is illustrated in Figure 1.

Definition 1 ([17])**.** *An MDP is a 5-tuple* $(\mathscr{S},\ \mathscr{A},\ T_a,\ R_a,\ \gamma)$ *which follows the Markov property where* $\mathscr{S}$ *is the set of states,* $\mathscr{A}$ *is the set of actions,* $T_a(s,s') : \mathscr{S} \times \mathscr{A} \mapsto [0,1]$ *is a transition function mapping and current state action pairs to the next state,* R_{a_t} *is a reward function given for a particular state, action, and next state tuple, and* γ *is a reward discount factor whose value is in the interval [0, 1].*

An agent at time t chooses an action $a_t \in \mathscr{A}$ from a distribution conditioned off of its current state, $s_t \in \mathscr{S}$. The agent's action selection is called a *policy* whose mapping is represented as $\pi : \mathscr{A} \times \mathscr{S} \to [0,1]$ where $\pi(a,s) = \mathbb{P}(a_t = a \mid s_t = s)$. An agent achieves a total reward $\mathscr{R} = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$ throughout its action period, which is finite in many RL problems.

The cart-pole is a common dynamical system seen in controls literature [18], which consists of a block sliding along a 1-D rail with an unactuated pole atop the cart illustrated in Figure 2. The cart-pole system is defined by the four state variables $\mathbf{x} = [x, \dot{x}, \theta, \dot{\theta}]^T$, which are the position, velocity, pole angle, and pole velocity. The goal is to exert forces on the block to stabilize the pole in the upright position. The 'CartPole-v1' implementation in Open AI only allows two possible actions – an impulsive force of fixed magnitude in either direction. Each episode begins with initial states each sampled from the uniform distribution $[-0.05, 0.05]$ and terminates if the pole angle magnitude dips below $\frac{\pi}{15}$ rad., the cart reaches the edge of the display, or if the agent survives 500 timesteps. A reward of +1 is given for every action.

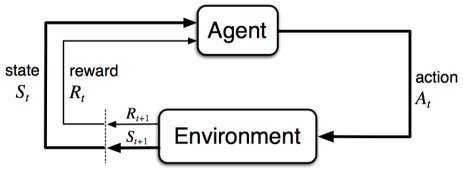

Figure 1: Diagram of a MDP in the agent-environment interaction [3]

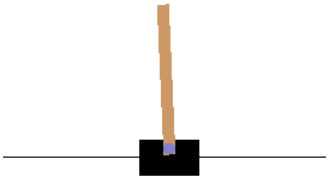

Figure 2: Open AI Gym cart-pole system [1]

3.1. Policy Gradient (PG)

Policy gradient (PG) is one of the basic methods with which to perform reinforcement learning. Here, we limit the RL problem of interest to a finite-time horizon. Setting $\pi_\theta(a,s) := \pi(a,s)$, we can expand to obtain $\theta^* = \text{argmax}\, \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} R_{a_t}(s_t, s_{t+1})\right]$. Setting the function $J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} R_{a_t}(s_t, s_{t+1})\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} R_{a_t}\left(s_t^i, s_{t+1}^i\right)$ where the mean is estimated through sampling. Next, we can compute the gradient $\nabla J(\theta)$. Using the property $\pi_\theta(\tau)\nabla_\theta \log_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$ and the definition of expectation, we arrive at

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_t^i, s_t^i\right)\right)\left(\sum_{t=1}^{T} R_{a_t}\left(s_t^i, s_{t+1}^i\right)\right).$$

To find the parameter θ, we can use gradient descent or similar gradient methods.

3.2. Deep Q-Learning

Q-learning is another form of RL, where if there exists a function $Q^* : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$, then we know the true reward for every state-action pair, and thus an optimal policy could be derived such that $\pi^* = \arg\max_{a_t} Q^*(s_t, a_t)$. The principle idea behind Deep Q-learning is that neural networks are universal function approximators [19]; $Q^*(s_t, a_t)$ is unknown, it can be learned via a neural network. The training update for the model are listed in the equations below. Let B represent a batch of transitions in the form of the collection $\{s_t, a_t, s_{t+1}, R_{a_t}(s_t, s_{t+1})\}$.

The Q-value update rule $Q^\pi(s_t, a_t) = R_{a_t} + \gamma$ is according to the Bellman equation. Equations $\delta = Q(s_t, a_t) - (R_{a_t} + \gamma \max_{a_t} Q(s_{t+1}, a_t))$ and $\mathscr{L} = \frac{1}{|B|}\sum_{\{s_t, a_t, s_{t+1}, R_{a_t}\}\in B} l(\delta)$ show how the loss function for the neural network is derived, and in particular equation $\mathscr{L} = \frac{1}{|B|}\sum_{\{s_t, a_t, s_{t+1}, R_{a_t}\}\in B} l(\delta)$ shows how experience replay is used to sample previous transitions. Equation (1) shows the use of

the Huber loss, which is close to mean square error when δ is small, and close to mean absolute error when δ is large.

$$l(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{1}$$

3.3. Model Predictive Control (MPC)

MPC requires knowledge of system dynamics to calculate control actions [20]. The algorithm does not solve for a general policy, and thus there is no training phase and the optimization is continuously run while the controller is in operation. MPC is not a machine learning method and is included in this paper as a performance benchmark. In the the cart-pole, the system dynamics (2) and (3) are derived directly from the laws of physics using the mass and length parameters found in OpenAI Gym.

$$\ddot{x} = \frac{1}{1 + 0.1\sin^2\theta}\left[u + 0.1\sin\theta\left(0.5\dot{\theta}^2 + 9.8\cos\theta\right)\right] \tag{2}$$

$$\ddot{\theta} = \frac{1}{0.5 + 0.05\sin^2\theta}[-u\cos\theta - 0.05\dot{\theta}^2\cos\theta\sin\theta - 10.78\sin\theta] \tag{3}$$

4. Methodology & Results

We opt to use the OpenAI Gym, which was created for implementing and developing RL algorithms [1]. In the case of policy gradient and Deep Q-Learning, we also use Pytorch [2] to construct the corresponding neural networks. The cart-pole is included in the standard Gym Python package. The objective of each algorithm is to efficiently stabilize each system by maintaining the pole upright within preset thresholds.

4.1. Cart-Pole Model Basic Algorithm

First, we formulate a simple, "naive" policy which chooses an action deterministically based off of its observation of only the angular position of the pole. For example, if the pole is at a positive angle, i.e, leaning rightward, the action would be to push the cart to the right such that the cart can attempt to be directly below the center of mass of the pole. The algorithm serves as a lower-bound of the performance we can expect to achieve, because all other agents will have access to the full state of the system. The results from this implementation are presented in Figure 4, which depicts a histogram of the rewards as well as the mean reward of 42.22.

4.2. Cart-Pole Model Policy Gradient

We specifically compute the policy parameters θ using the neural network architecture in Figure 3. This architecture is composed of two linear layers as two activation functions. In general, the first linear layer uses $\tanh(\cdot)$ as the activation function. However, the output activation function was a design parameter. Figures 5 and 6 illustrates the reward over all the episodes and the respective mean for the $\text{ReLu}(\cdot)$ function and the identity output activation function, respectively.

4.3. Cart-Pole Model Deep Q-Learning

The neural network used to learn the $Q(s_t, a_t)$ approximation was a three-layer neural network. The inputs to the network were the four observations given by the environment, and the outputs

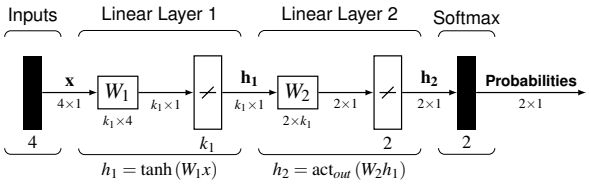

Figure 3: Illustration of Neural Network used for Policy Gradient. Here, a state vector $\mathbf{x}$, i.e., the full cart-pole state is passed through the network and logit probabilities are computed at the end using the softmax function.

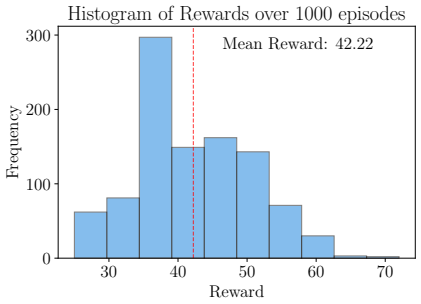

Figure 4: A histogram of the rewards for for 1000 episodes of the basic algorithm.

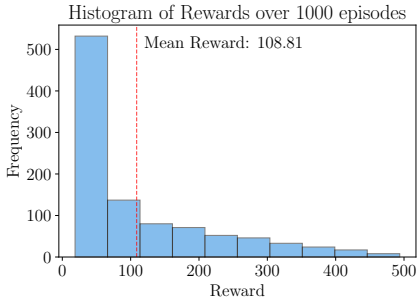

Figure 5: A histogram of the reward of the policy gradient RL algorithm with the ReLu$(\cdot)$ output activation function. Here, we see that the mean reward is 108.81.

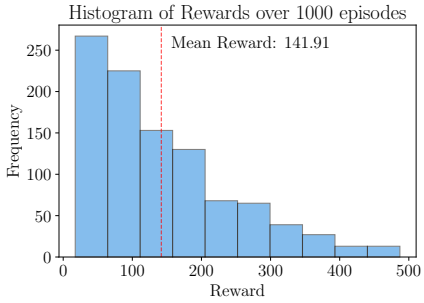

Figure 6: A histogram of the reward of the policy gradient RL algorithm with identity output activation. Here, we see that the mean reward is 141.91.

were the expected Q-value for taking a left and a right action ($Q(s_t, \text{left})$ and $Q(s_t, \text{right})$). The neural network layers had 50, 100, and 50 nodes respectively; each were passed through a 1D batch normalization and then a ReLU activation function was used. Figure 7 presents the rewards achieved over 1000 runs of the learned policy where the average reward achieved was 88.90.

4.4. Cart-Pole Model Predictive Control (MPC)

Many implementations of MPC employ trajectory optimization to find sequence of actions that maximizes the rewards. However, the simplicity of the reward function means the trajectory only needs to avoid episode termination (tipping the pole or moving the cart too far from center). The small, discrete action space precludes the use of gradient methods to optimize the trajectory. Instead, all possible feasible combinations of the two actions are generated for for the planning horizon, which was chosen as 10. A trajectory is chosen based off of a cost function penalizing deviations from state $\mathbf{x} = [0,0,0,0]$. Since this is a simple simulation without external disturbances and unmodeled dynamics, the state transitions occur deterministically and the MPC algorithm's predictions are exact. The algorithm achieved the maximum reward of 500 every time for 100 episodes. The state history for one such episode is depicted in Figure 8. Here, we see only small deviations in pole angle which are well within the limits of the simulation.

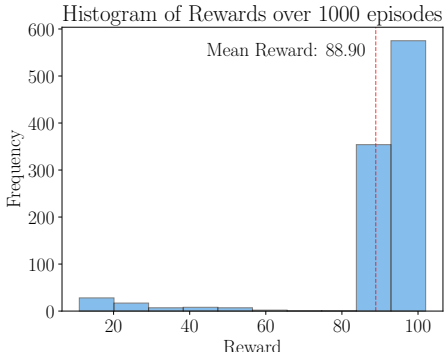

Figure 7: A histogram of the reward of the Q-learning RL algorithm. Here, we see that the mean reward is 88.90.

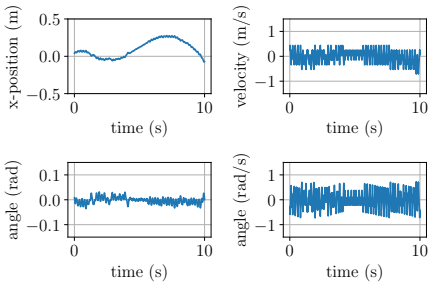

Figure 8: Results from stabilizing the cart-pole system using MPC.

5. Discussion

The relative performance of each algorithm can be explained in terms of their fundamental differences. Of the RL algorithms tested, the PG algorithm learned a policy which performed the best. For PG, we tested two output activation function architectures, ReLu$(\cdot)$ and tanh$(\cdot)$, where the average rewards are 108.81 and 141.91, respectively. ReLu performs worse than tanh$(\cdot)$, which can be attributed to the fact the neurons can "die" as a result of using ReLu [21]. DQN averages 89.90 over 1000 episodes, making it perform below both PG architectures. Lastly, the basic algorithm, which served as a baseline model comparison, achieves an average reward of 42.22 over 1000 episodes. The performance of PG over DQN can be attributed to the fact that the PG algorithm is on-policy as opposed to DQN which is off-policy. This means that the PG algorithm can learn and tweak the policy during the course of any episode of optimization since it is testing and learning on the policy at the same time. This is not the case for the DQN, where the off-policy nature means that for any single episode of optimization, only one update to the policy network will be made. Furthermore, the optimizations used for DQN, such as experience replay and dual function learning, perhaps hindered the ability of a complex network (three layers) to learn the policy given a simple simulation such as cartpole. Finally, one fundamental difference between the two algorithms that could serve to explain the differences in performance is that the DQN is attempting to approximate the Q-value for either action in any state, while the PG is performing parameter estimation and using the long-term expected reward to optimize. This difference in objective functions of the algorithms could point to why the DQN learns a policy that performs worse than the PG does, because the DQN may not have explored every state-action pair and learned the proper action to take according to the estimated Q-value. This leads to the takeaway that ***there exist trade-offs between the complexity of the learning algorithm and the problem itself, i.e., sometimes more complex algorithms are not better***. Overall, MPC is the most effective as it does not need to cycle through a series of failures to ultimately reach the desired goal; however, RL methods may be much better when dealing with either more complex or stochastic system dynamics.

6. Conclusions

In this project, we performed a survey of reinforcement learning techniques and implemented various methods relevant to dynamical systems. Here, we benchmark the results of RL algorithms against the results of a classical controller. Among the RL methods, PG proved to be the most effective with an identity output activation; however, MPC is better suited to address this problem since the system dynamics are easily understood without needing trial-and-error learning. Future work could involve testing these algorithms on more complex dynamical systems and implementing them on real-world hardware.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[5] D. P. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[6] L. Graesser and W. L. Keng, *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley Professional, 2019.

[7] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, pp. 34–38, 2013.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[9] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[10] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[11] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of International Conference on Robotics and Automation*, vol. 4. IEEE, 1997, pp. 3557–3564.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[13] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[14] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[15] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[16] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 167–176.

[17] C. C. White III and D. J. White, "Markov decision processes," *European Journal of Operational Research*, vol. 39, no. 1, pp. 1–16, 1989.

[18] 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*, no. 5, pp. 834–846, 1983.

[19] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[20] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[21] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.