Training a self driving vehicle with formally verifiable reinforcement learning

Anthony Marchiafava, Jober't Aladwan, Sara Basili

May 16 2023

1 Motivation

Reinforcement learning (RL) is a powerful technique used to solve some world tasks in which an agent learns by interacting with an environment. The agent learns a policy that dictates which action should be taken when in various states. However, the agent's interactions often begin without any knowledge of the environment and must learn an optimal policy by getting rewards from these interactions. If a self-driving car designed to learn to drive on the street instead learns to use off-street paths, it must be trained further to prevent such actions and get closer to the optimal on street driving policy. As this more optimal policy develops, can this policy have some assurance that the off-street behavior would not return? Can this policy incorporate other constraints such as stopping at stoplights? Can anything be said to assure someone that these unacceptable actions will not take place?

2 Challenges

Finding an optimal policy. Reinforcement learning uses rewards to develop an optimal policy. These rewards are designed to encourage actions which lead to more desired states and discourage actions which lead to less desirable states. However, this can have unintended consequences where a less than ideal action is encouraged before a better action is discovered. An agent designed to drive through a racetrack may learn to drive off-course if it leads to fast results and there are no constraints to discourage that behavior. Properly building rewards to only encourage optimal behavior will be paramount in creating this safety-oriented agent.

Modeling the environment. When assessing the value of an algorithm which has learned some actions on its own, the quality of the environment from which the agent gets its state is very important. If the agent is supposed to learn to not drive off a racetrack, there must be some non-racetrack area in which it can be penalized for entering. When learning in an artificial simulation, a more closely related real environment and simulated environment would lead to more realistic policies.

3 Related Work

Different formal verification techniques for Reinforcement Learning have been used in a variety of contexts such as self driving cars [1], human resource management [2], multi-agent systems [3], trajectory generation [4], and could apply to many other problems. These techniques intend to improve the quality of the reinforced learning by incorporating domain knowledge [5], show the ability to reach certain states [3], and show safety [6]. These and other similar problems have been solved using popular techniques such as Neural Fitted Q Iteration [7], Deep Q-Learning [5] and Deep Deterministic Policy Gradient (DDPG) [8]. The process for building Linear Temporal Logic is based on [9].

4 Objectives

Our work will show that some formally defined constraints can be incorporated into a reinforcement learning agent based on Linear Temporal Logic in a manner similar to [5][1]. With some

modifications, a reinforcement learning (RL) algorithm may include domain knowledge which the agent would not otherwise start with and dramatically decrease the need to explore actions whose consequences are already known to be unacceptable. This agent will manage a self-driving wheeled vehicle in a racetrack like environment, and its goal will be to drive through the racetrack while not entering the goal region on a red light and not going off-track.

5 Problem formulation

We can express this problem as a Markov decision process (MDP). MDPs are sequential decision problem for a fully observable, stochastic environment, with a transition model which only requires the current state and an action to get to the next state, and a subsequent reward that gets our agent closer to some goal [10]. The MDP consists of a set of states, a set of actions in each state, a transition model which takes an action and the current state which are used to provide the next state, and a reward function. These are all used to generate a policy which provides an action which should be taken given a state. This policy should provide, for any state encountered, an action which will get the agent closer to our goal.

To win the game the agent must finish the track (by getting into the goal region) without going off the road. The agent will not enter the goal region on a red light. Formally expressed with Linear Temporal Logic our problem is to optimize a policy for an agent constrained by the following:

$$\square(\text{goal} \implies \square\text{goal}) \wedge \square(\text{unsafe} \implies \square\text{unsafe}) \wedge \diamond\text{goal} \tag{1}$$

Our goal, then, is to find some optimal policy $\pi^* : S \to A$, such that the policy will provide an optimal action for each state.

Our problem is broken down into a MDP made up of the state of the environment, modeled with CoppeliaSIM, and an actor in the simulator. This actor interacts with the environment and our actions will be based on fulfilling the logical criteria expressed in (1). This logical criteria is converted to a Limit-Deterministic Büchi Automata (LDBA). The LDBA is then used to label the current state of the program so we are aware of the safe or unsafe states. If the car goes off the boundaries of the track or enters the goal region while the light is red, we consider that unsafe.

We used two methods for learning an optimal policy, Neural Fitted Q-Learning and Deep Deterministic Policy Gradient. These both rely on the idea of a Q-function and a policy. The Q-function is an optimal action-value function. Provided some state and action, our Q-function provides us the value of doing that action in that state [10].

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{2}$$

Where $Q(s,a)$ is the Q-function of some state s and some action a, $R(s)$ is the reward function which specifies the reward for a given state (s), γ is a discount factor to discount the value future rewards, and we would sum up the transition model $(P(s'|s,a))$ which shows the probability of reaching the next state s' from state s doing action a which is chosen, and $Q(s',a')$ shows the utility value of action (a') from state (s') which is the expected reward. This would require a transition model, but it is possible to learn without such a model. For Q-learning we could instead use [7]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3}$$

So if we keep a table of our utility values from our state action pairs and update this table until we sufficiently explore the search space. Then we can use the Q table to discover which action to take for an optimal policy without needing the transition model. This can work for modeling discrete actions and discrete states, but proves challenging with problems with continuous action spaces and continuous state spaces.

6 Methodology

For our task of testing logically constrained learning, we created a working version of a Markov Decision Process contained within CoppeliaSim to act as the environment and adopted the framework as seen in [11] to create our logical constrained learning simulations.The knowledge of the environment each agent will have will be composed of its X and Y velocity, the status of the traffic light, and how far the agent is in from the goal object. Our environment would be converted to

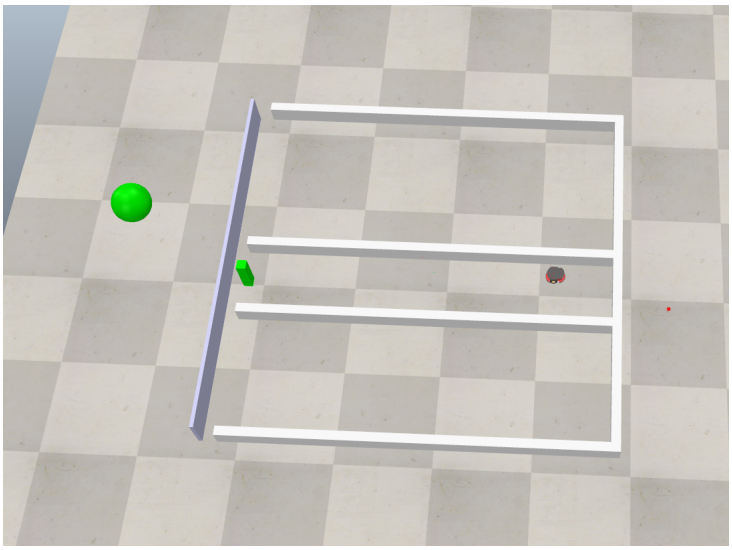

Figure 1: Simulation Track in CoppeliaSim

a discrete state space for the agents which are configured to only work with a discrete space, and left continuous for the agents that were capable. Our agent could choose to go north, east, south, west, northeast, southeast, northwest, southwest, or remain stationary. Our agent would have its goal region marked on the street, and a light which would change color. The main job would be to get into the goal region while the light was not marked red, while not hitting any of the sides of the street. Entering the goal region while the light was red or hitting the sides of the street were specified as unsafe states, being in the goal region while the light was green was the accepting goal state. We chose three methods of generating a policy to explore: Basic Q-Learning, Neural Fitted Q-Learning (NFQ), and Deep Deterministic Policy Gradient (DDPG).

Neural Fitted Q-Learning This algorithm allows offline learning of Q-values with a densely connected neural network. Instead of storing a complete Q-table of every state and action from that state, we instead could use a neural network to represent the value function. We would generate a large table of experiences made up of the current state s, the action taken a, and the next state s' from having an agent interact with the simulated environment. Then we would use a multi-layer perceptron (MLP) with two layers to train on the entire sample set, so that it could approximate Q values.

Deep Deterministic Policy Gradient This algorithm is based on the actor-critic approach to allow online learning of the optimal policy instead of just learning a Q function. The DPG algorithm has both a parameterized actor function $\mu(s|\theta^{\mu})$ for the policy, and $Q(s, a)$ for the critic. DDPG is an extension of DPG, utilizing a deep neural network function approximators to learn large state and action spaces online. It also uses a replay buffer which stores transitions (s_t, a_t, r_t, s_{t+1}) where s_t is a state at time t, a_t is the action at time t, r_t is the reward at time t, and s_{t+1} reflects the next state. To make our discrete action space and state space into a continuous state and action space we took our predicted move output and converted it back into an integer.

7 Simulation Results

Q-Learning We began training our agent using Q-learning with hyperparameters set to 20000 episodes, 2000 iterations, discount rate set to 0.95, and a learning rate of 0.05. Q-learning was first trained on a modified version of the track that featured a much wider straight-away to estimate the difficulty of the track. In this wide track, the Q-learning algorithm was successful in finding a reliable route to the goal multiple times within 2000 episodes and achieved roughly 70% success rate in reaching the goal without violating any of our constraints. However, when switching to the intended narrow track, Q-learning failed to reach the goal a single time over 20,000 episodes. Upon inspection of the environment during training, the agent was hardly able to even make it halfway up the track without running into the side of the wall. We hypothesize this shortcoming to be because as the agent is filling out the Q-Table, due to the sparse reward, it has no bearing on whether it is

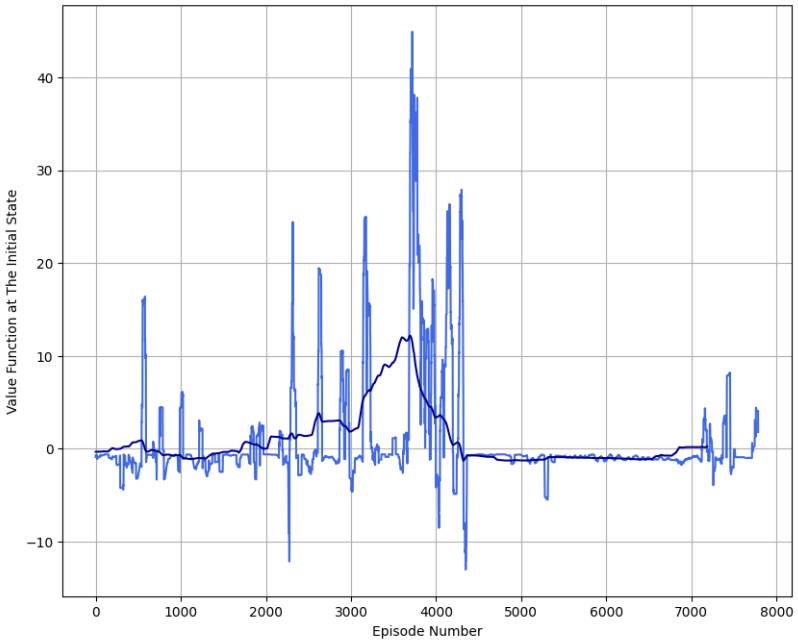

Figure 2: Value Function of DDPG over Episode Count

reaching the goal and is essentially performing random actions until it does. The abstraction process of the environment to a discrete state space also poses trouble for the agent's learning ability as some states may not be represented as well as they should be when discretized.

Neural Fitted Q-Learning NFQ was trained using similar hyperparameters as Q-learning, with 20,000 episodes and 2000 iterations and the same discount and learning rate. A memory buffer of 100,000 was set to preserve as much of the Q-Table as possible in the hopes the agent will train faster. The results of testing our agent on the straight-line track using NFQ to learn the environment were unsatisfactory. After roughly 18 hours of training and 20,000 episodes, the agent failed to reach the goal a single time. When observing the agent train, we noticed a common failure that Q-learning performed, which was frequently driving into walls and taking nonsensical pathways to fill out the Q-table. Clearly, the abstraction process of a continuous world such as in CoppeliaSim and reality is a significant hamstring to both Q-learning models. Although we were not able to test NFQ in a wider track to see if it replicated the success Q-Learning had, we have a strong suspicion it would learn similarly, if not faster due to the neural fitting modification to the algorithm.

Deep Deterministic Policy Gradient DDPG was trained using the same hyperparameters as the previous two algorithms, along with a similar memory buffer of 100,000 which was left default. However, due to time constraints, the agent's training was restricted to about 8000 episodes. The results of training a DDPG agent on the narrow track were extremely successful. Although a convergence of the value function did not occur which would signify optimal training as seen in 2, the tested agent achieved a goal success rate of 76%, meaning the agent is able to reach the goal without violating any of our constraints 76% of the time. We suspect this success is tied to two factors: the algorithm's ability to handle continuous state spaces and not relying on a Q-Table that must be filled out to find a result. Due to DDPG's specialty in handling continuous state spaces, we were able to remove our rounding actions that discarded information that may have been useful to the Q-Learning algorithms discussed prior. Through visual observation of the agent, we noticed a few intriguing developments of how the agent learns its strategy. At first, the agent would careen into the walls early on the track, but as more episodes occurred, the agent learned to drive straight to put off its failures. However, this strategy is not optimal as the agent must refrain from entering the goal when the traffic light is in the red state. The agent, over time, learned to perform wait actions in between its movements so as to delay its arrival to the goal before the light turned red.

We believe given much more training time to allow for value function convergence this agent will master the environment with an extremely high success rate.

8 Conclusion

The problems reinforcement learning is well suited for problems which are difficult to solve analytically. Analytical assurances are a major goal of any safety critical system. Logically constrained reinforcement learning can provide some assurances of adherence to some specified logical criteria. However, it is no panacea.

In our experiences here we can see a clear trade off between the assurances LCRL provides and the sparse reward structure that it entails. Our experiments here show that while our agent did not learn and converge on some less than ideal policy, instead it sometimes did not converge on an ideal policy at all. There is a trade off that is made. Since our best policy did not get higher than 76% we cannot be assured that our policy will be safe. This also means we know to what degree our policy does not adhere to the logical controls we put forward. More work in reward shaping, increasing training time, and further altering input and state space parameters and algorithm hyperparameters may improve our agent in the future. LCRL is no universal solution, it does not provide a simple answer to all of reinforcement learning's problems, but it does provide some assurance that if a particular policy is found we can know within some statistical certainty that it fulfils all the criteria we require.

References

[1] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, "Reinforcement learning with probabilistic guarantees for autonomous driving," 2019. DOI: 10.48550/ARXIV.1904.07189. [Online]. Available: https://arxiv.org/abs/1904.07189.

[2] Z. Boudi, A. A. Wakrime, M. Toub, and M. Haloua, "A deep reinforcement learning framework with formal verification," *Form. Asp. Comput.*, Dec. 2022, Just Accepted, ISSN: 0934-5043. DOI: 10.1145/3577204. [Online]. Available: https://doi.org/10.1145/3577204.

[3] X. Wang, J. Peng, S. Li, and B. Li, "Formal reachability analysis for multi-agent reinforcement learning systems," *IEEE Access*, vol. 9, pp. 45 812–45 821, 2021. DOI: 10.1109/ACCESS.2021.3060156.

[4] D. Corsi, E. Marchesini, A. Farinelli, and P. Fiorini, "Formal verification for safe deep reinforcement learning in trajectory generation," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, 2020, pp. 352–359. DOI: 10.1109/IRC.2020.00062.

[5] M. Hasanbeig, A. Abate, and D. Kroening, *Logically-constrained reinforcement learning*, 2018. DOI: 10.48550/ARXIV.1801.08099. [Online]. Available: https://arxiv.org/abs/1801.08099.

[6] P. S. N. Mindom, A. Nikanjam, F. Khomh, and J. Mullins, "On assessing the safety of reinforcement learning algorithms using formal methods," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, 2021, pp. 260–269. DOI: 10.1109/QRS54544.2021.00037.

[7] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Proceedings of the 16th European Conference on Machine Learning*, ser. ECML'05, Porto, Portugal: Springer-Verlag, 2005, pp. 317–328, ISBN: 3540292438. DOI: 10.1007/11564096_32. [Online]. Available: https://doi.org/10.1007/11564096_32.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, 2015. DOI: 10.48550/ARXIV.1509.02971. [Online]. Available: https://arxiv.org/abs/1509.02971.

[9] S. Sickert, J. Esparza, S. Jaax, and J. Křetínský, "Limit-deterministic büchi automata for linear temporal logic," in *Computer Aided Verification*, S. Chaudhuri and A. Farzan, Eds., Cham: Springer International Publishing, 2016, pp. 312–332, ISBN: 978-3-319-41540-6.

[10] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[11] H. Hasanbeig, D. Kroening, and A. Abate, *Lcrl: Certified policy synthesis via logically-constrained reinforcement learning*, 2022. arXiv: 2209.10341 [cs.LG].