Using Reinforcement Learning to Beat the Best Putter in the PGA

Julia Novakoff, George Pesmazoglou, Teddy Laurita

Abstract

Reinforcement Learning is a type of unsupervised machine learning in which the agent makes decisions upon observations and expected rewards. Reinforcement learning has been useful for robotics skill acquisition and has been applied to a variety of tasks such as autonomous flight for helicopters and soccer. We will use ROS, Gazebo, and the openAI gym to teach a TurtleBot robot in simulation to putt a sphere into a goal area, representing a ball putt into a hole. We will teach the robot to both predict how much power with which it must hit the ball, as well as at which angle it must position itself behind the ball in order to putt at the hole. We will consider those skills learned once our simulated TurtleBot can make its putt a greater percentage of the time than the current best PGA tour putter.

Introduction

Reinforcement learning (RL) is a machine learning method for learning a problem so as to maximize the numerical reward signal. These problems are closed-loop problems, in that the learning system's actions influence its future inputs. Furthermore, the agent is not given any instruction on how to solve the problem, but is instead must discover which actions yield the most reward by trying them. Finally, the third important characteristic of reinforcement learning is that immediate actions affect not only the immediate reward, but also perhaps future situations and all future rewards those situations encounter. The process of reinforcement learning occurs in discrete points in time with the following steps: at each step the agent gets an observation, the agent performs an action, finally the agent may or may not get a reward or change states. This process continues until the system has learned the optimal way to complete the task with regard to getting the maximal amount of reward.

One of the most important aspects of a reinforcement learning problem is the value function, or a function that predicts the value that an agent will get from performing a certain action. The one of the challenges in creating a value function for a reinforcement learning problem is that a in order to obtain a reward a reinforcement learning agent must perform actions it knows will yield a high reward, however to discover which actions produce the highest reward the agent must try new actions. A greedy value function will simply pick the action that nets the greatest reward; a strategy that suffers from the fact that once a method for achieving the goal is discovered, the agent will never check to see whether other sets of actions might achieve the goal with a higher reward. An epsilon greedy function will pick the action that yields the highest reward, while at some small probability epsilon instead picking a random action. The challenge is to utilize an epsilon that will allow the agent to learn the system quickly, while still attempting different methods for doing so to learn the system in the best way possible. It is generally useful to visualize the reinforcement learning problem in order to plan which value function will work best.

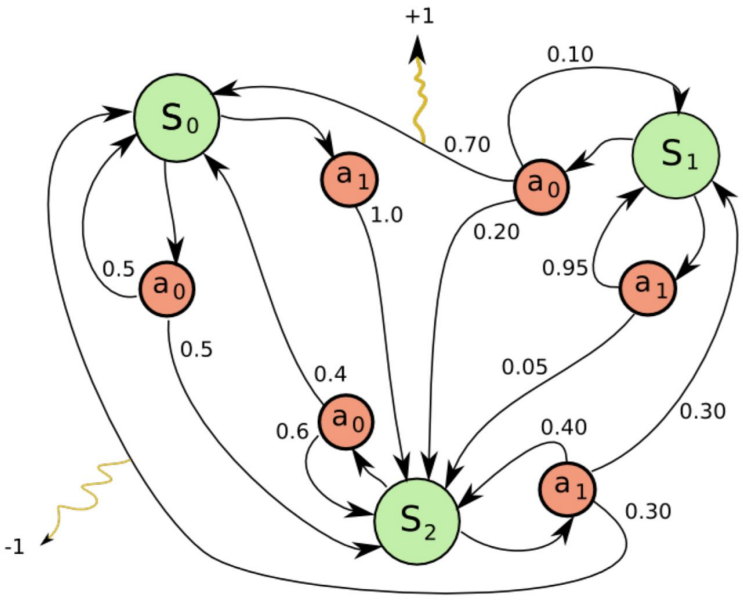


Fig. 1 - A Markov Decision process outlining how actions may or may not cause the agent to change states

A Markov Decision Process (MDP) (fig. 1) is a graph describing the states in which the agent could possibly be at any time period, and the actions the agent might take while in a given state. At each state space, the agent has one or more action available to it, and a probability associated with each action that decides to which next state space that action will send the agent. A Markov Decision Process can only exist if one assumes the Markov Property is true, that is that any decision the agent makes while in any state space is independent of reward garnered or decisions made at a previous time point. In order to create a MDP where the state that the agent is in at time t is dependent on the state that the agent occupied at time t-1, one would create a new MDP in which each state pair from the previous MDP is a state in the new MDP.

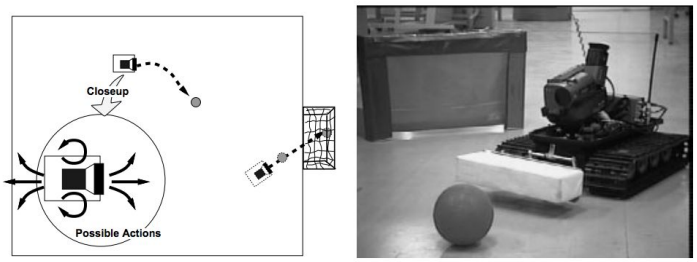
One classic method for implementing a reinforcement learning problem is Q-learning. Q-learning relies on using a Q-table to store the reward associated with each possible action at each possible state in which the agent could be. The table gets continually updated as the agent learns about the actual reward garnered with each certain action at each state space, though it takes into account not only the reward gained from performing each action, but also the maximum potential reward the agent might get in the next state space if it performs any given action in the current. Q-learning takes into account the maximum potential reward to be gained at a future state space, but it discounts that reward to account for the fact that it is potential and not actually the reward gained in the current state space. Thusly Q-learning can create a path from start to finish that maximizes the reward at each individual step, as the Q-table accounts for which action will produce the greatest reward in the future. The greatest advantage to using Q-learning methods for learning a RL problem is that it theoretically guarantees convergence to the true Q-function, were the problem to be attempted an infinite number of times. That being said, Q-learning is still useful for getting a good estimate for the ideal Q-function, albeit at the

expense of potentially needing to store enormous Q-tables and being completely unable to estimate the reward values for actions that the agent has not taken yet. One of the important aspects that affects how well a Q-learning problem will learn is the reward system formulation.

The reward system in a reinforcement learning problem is the system by which the agent is rewarded for performing certain actions whilst in certain state. The reward system one uses for an RL problem is extremely important for how well the agent learns, as a poor reward system that only rewards the agent for completing the problem will cause the agent to take many iterations to learn which actions garner the most rewards. One way around this problem is to utilized reward shaping, in which the experimenter awards a smaller reward for taking actions that lead the agent closer to the goal, but don't actually complete the goal itself, or awarding a negative reward for actions that take the agent further away from the goal. In this way the agent can learn which actions are relatively better than others and incrementally learn how to gain the greatest reward, instead of fumbling in the dark until it randomly completes the goal, and then has to reverse engineer how to get from the goal back to the start.

Related Work

Early work has shown that reinforcement learning is a powerful method for skill acquisition in robots. Gullapalli, Franklin & Benbrahim (1994) used a stochastic real-valued reinforcement learning algorithm to successfully teach a robot how to insert a peg into a hole and balance a ball. Asada, Noda, Tawaratsumida & Hosoda (1996) was also able to use reinforcement learning to teach a robot to shoot a ball into a soccer goal (fig. 2). Later on, Hester, Quinlan & Stone (2010) solved a similar penalty kick problem using reinforcement learning. They argue that in order for reinforcement learning to work optimally it must be *sample-efficient*, meaning that it can make an accurate prediction after relatively few trials. To do this, they used decision trees to generalize the outcomes of current states and actions, so less time would be exhausted searching across all possible states.



(a) The task is to shoot a ball into a goal. (b) A picture of the radio-controlled vehichle.

Fig. 2 - Asada et al. (1996) - diagram of penalty kick task and photograph of actual robot

Policy selection refers to how the reward for a given action is chosen, and has proven to be a challenging area of work. Some interesting methods for policy selection have been researched thus far. Kim et al. (2004) used reinforcement learning to create an autonomous helicopter that could hover and fly simple maneuvers on its own. They used the PEGASUS

algorithm, outlined by Ng & Jordan (2000), to select a policy for their very large MDP. The PEGASUS algorithm searches over estimates made by transforming the original MDP into a similar one with only deterministic transitions. If the estimate is a good approximation, optimizing it will select the best policy for the given MDP.

Q-learning has proven to be a successful method for implementing reinforcement learning. Minh et al. (2015) found that a deep Q-network agent was able to learn how to play 49 different Atari 2600 games at a level on par with human game testers. However, Van Hasselt, Guez & Silver (2016) showed that Q-learning often results in overestimations of action values, and deep Q-learning can augment these overestimations. They proposed an algorithm for double deep Q-learning they showed to be successful in reducing the overestimations.

Leffler, Littman & Edmunds (2007) formulated a different algorithm for policy selection, which they called RAM-Rmax. They used a relocatable action model, which maps a type and action to a set of probabilities of different outcomes. RAM-Rmax assumes that the type and next state functions are known, and only the outcomes of actions must be estimated, making the decision process more efficient.

In an attempt to accelerate reinforcement learning, Price & Boutilier (2003) proposed that introducing a form of *implicit imitation* could help a robotic agent gather information about its own abilities through observing a mentor. Through this, the agent gains some knowledge about previously unexplored components of the state space, and can thus be more efficient in selecting a policy.

OpenAI is a company that whose mission is to build a safe AGI. They have released open source code to the community, and their results are very impressive, as they were able to beat the best DOTA players in the world after 6 months of unsupervised learning. We will be using their libraries to implement the reinforcement learning algorithms.

Problem Formulation and Technical Approach

Our study involves a setup very similar to putting in golf. The robot will bump into a round ball and record the what happens, that is whether the ball moved, how far, and in what direction. We will be using reinforced learning to make the robot push the ball to a specific location, which will represent the hole in golf.

We will be using the Gazebo simulation with a simulated TurtleBot to perform this experiment. The main reason for using it is that reinforcement learning requires a very large number of iterations of the same task. Performing reinforcement learning in a physical robot is more difficult because it is slower, and real world imperfections (from the environment or the robot's hardware) can cause errors. The simulation will contain the following objects: the simulated TurtleBot, a simulated ball, and a clearly marked zone on the floor which designates the location of the hole. The zone is simply a circular shape on the ground, with an easily visible color. The robot changes its speed and approach to the object to perform a successful putt. When the putt is successful, the robot is given a reward, whereas when there is a miss, it is given a reward with negative value based on the distance from the target location. Based on previous research, this approach should, after many iterations, lead to the robot learning how to accurately push the ball to the target. After having trained the robot to consistently putt

successfully, we will move on to the next phase, in which the target location will be variable, and the robot will have to take that location into account before performing an action.

We will be using three pieces of software, ROS, Gazebo and OpenAI gym. The reinforced learning algorithm is contained in the OpenAI libraries, and Gazebo is used for the simulation. ROS is talking to both OpenAI and gazebo (fig. 3).

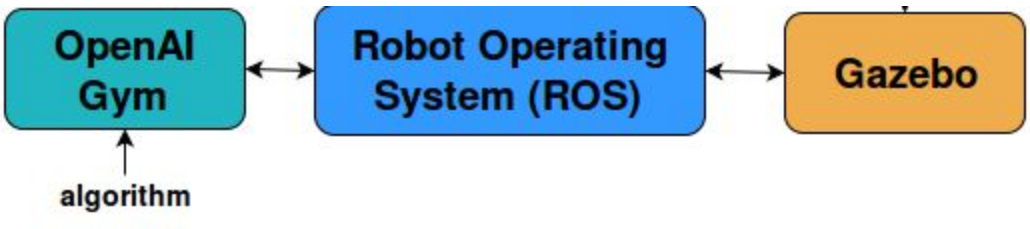


Fig. 3 - We will integrate the OpenAI library with ROS, which in turn makes use of Gazebo physics engine.

We will try two algorithms, Q-Learning and Sarsa. Q-Learning was thoroughly discussed in the introduction. Sarsa is an On-Policy algorithm for Temporal Difference Learning. It is different from Q-Learning in that the maximum reward for the next state is not always used for updating Q-values. New actions are decided the same way, the original action was decided.

Expected Results and Experimental Validation

We expect to be able to teach the TurtleBot to push (putt) a spherical object in simulation into a goal area (hole). Not only will we teach the robot to understand at which velocity it must contact the ball in order to push it a certain distance, furthermore we will teach it to line itself up at the correct angle between the ball and the hole so that when the robot contacts the ball it will shoot at the hole.

We have decided that the robot will have successfully learned how to putt a ball into a hole when it can successfully make the putt more than 83.33% of the time, as this is the best putting percentage on the PGA tour this week for putts between five to ten feet (Scott Piercy week of 10/22/17).

Timeline

- Nov 3: Gazebo, Ros and openAI gym environment setup, working tutorials
- Nov 10: Q-table implemented, ability to train robot
- Nov 13: Project report finished
- Nov 14: Progress presentation in class
- Nov 22: Robot has been trained for base case, begin trials where the target destination is at a different location, at every iteration
- Nov 28: Robot can putt the ball based on the location of the “goal” - class presentation
- Dec 5: Final Presentation

References

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning*, *23(2),* 279-303.

Degris, T., Sigaud, O., & Wuillemin, P. H. (2006). Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd international conference on Machine learning, June*, 257-264.

Gullapalli, V., Franklin, J. A., & Benbrahim, H. (1994). Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, *14(1),* 13-24.

Hester, T., Quinlan, M., & Stone, P. (2010). Generalized model learning for reinforcement learning on a humanoid robot. *2010 IEEE International Conference on Robotics and Automation (ICRA), May,* 2369-2374.

Kim, H. J., Jordan, M. I., Sastry, S., & Ng, A. Y. (2004). Autonomous helicopter flight via reinforcement learning. *Advances in neural information processing systems*, 799-806.

Leffler, B. R., Littman, M. L., & Edmunds, T. (2007). Efficient reinforcement learning with relocatable action models. *AAAI, 7,* 572-577.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.

Ng, A. Y., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, *June,* 406-415.

Price, B., & Boutilier, C. (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, *19*, 569-629.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *AAAI*, *February,* 2094-2100.

Zamora, I., Lopez, N. G., Vilches, V. M., & Cordero, A. H. (2016). Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. *arXiv preprint arXiv:1608.05742*.