Solving Cartpole Using Reinforcement Learning and Policy Gradients

Machine Learning Engineer Nanodegree
Capstone Proposal

Richard Decal

May 16, 2017

Domain Background

Reinforcement learners are a class of algorithms which learn to perform a task given a way to evaluate its own performance. These algorithms can amazingly become proficient at the task– in some tasks outperforming humans– without ever being given explicit directions, rules, explanations, or any domain knowledge whatsoever. Given enough time to test various strategies, a good reinforcement learning algorithm iteratively biases its decision policy until it converges to the optimal performace.

Problem Statement

In this work, I want to train an agent to play "CartPole", a simple 2D game which simulates an upright pole conected by a hinge to a movable cart.[1] Each trial is initated with the cart at the center of the environemtn and the pole upright. At each timestep, the agent has to choose between pushing the cart to left or to the right. The trial ends if the pole tilts 15° away from vertical, if the cart has translated to the edge of the screen, or if the trial reaches its time limit.

Datasets and Inputs

For this task, my learner will be totally naive and will have no previous data to inform the policy. At each timestep, the game provides the agent the current environment state as well as the reward for the action taken at the previous timestep. The sensory input is the cart's position x, the cart's velocity $\dot{x}$, the pole's angle θ (measured in radians), and its rate of change $\dot{\theta}$.[2] I propose to reduce the dimensionality of the inputs by half by considering solely x and θ, since I predict these are the most salient features (discussed in the "evaluation metrics" section).

Given the state and the chosen action, the agent is given immediate feedback from the game in the form of a reward. These (state, action, reward) pairs are crucial for the reinforcement algorithm to improve its policy from its experiences (discussed in the project design section).

Solution Statement

For the purposes of this project, I will consider the game"solved" when the pole is maintaned upright for 90% of the maximum possible gameplay time for 20 consecutive trials.

Benchmark Model

To benchmark our models performance, I will create models to function as baselines. The first is a random agent, which takes a random sample of the allowable actions at each timestep. The second policy is to push the cart in the same direction that the pole is tilting. In other words, if the pole is leaning to the right, the agent will always push to the cart to the right, restoring balance by positioning the cart directly beneath the pole (and vice versa).

Evaluation Metrics

We can expect an optimal agent to keep the pole balanced upright (i.e. close to 0°). This can be evaluated by charting the average observed angle θ. In addition, we want the agent to bias the cart to move away from the edges to minimize the risk that it prematurely ends a trial for moving out-of-bounds. We can check whether our policy is biased towards the center by plotting a histogram of the cart positions. The optimal agent will be able to maintain the pole upright for long periods. We can quantify this by plotting the average length of the trials in each episode. Finally, we can plot the average reward per trial for each episode to get a sense of how the agent judges its own performance.

Project Design

First, I will implement the simple baseline models mentioned in the benchmarking section. These are very simple and easy to implement. I will also create plots which visualize the statistics mentioned in the evaluation metrics section above. The combination of these two will provide a reference for the rest of the project.

Given the relatively low-dimensional state space, Q-learning is a possible learning algorithm to use. However, I would like to implement an algorithm which is more broadly applicable to tasks with a very large state space. Generally, these require on-policy learners, since an off-policy learner would explore the state space more and would be too computationally intensive.

The algorithm I have chosen is policy gradients. Policy gradients do not require any prior understanding of the reward function.[3] The learning algorithm is online, meaning that the policy is being updated at every timestep. This also reduces variance while keeping the bias low.[3]

According to reference [3], it is best to optimize over individual timesteps rather than over whole trajectories. The problem with the whole-trajectory method is that it has very high variance, as the algorithm confounds the rewards from all the actions. The gradient boosts the probability of all the actions taken in the trajectory *equally*, reguardless if any single one was a mistake. This is a low bias, high variance approach, which means we need many samples before the empiracle estimator becomes good. Computing the gradient for each reward reduces the variance while maintaing low bias.

In addition, we want an estimator that only increases the probability of actions which has above-average rewards. To achieve this, we add a bias term, which increases the log-probability of action a_{t_i} proportionally to how much reward is better than expected.

I propose to implement my algorithm using Tensorflow. This will help me visualize computational graph of the model.

References

[1] OpenAI. *Cartpole Documentation.* URL: https://gym.openai.com/envs/CartPole-v0.

[2] *gym/cartpole.py line 58.* URL: https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py#L58.

[3] John Schulman. *Deep Reinforcement Learning (lecture 2).* 2016. URL: https://www.youtube.com/watch?v=oPGVsoBonLM.