Model Based Learning - CartPole Swing Up

Submit by October 29 2018

October 19 2018

1 Problem

This problem is designed to give you an idea of some of the work our group does. This assignment is designed such that the solution cannot be found on Google or in a textbook. It is designed to help you think through the problem at hand from first principles and arrive at a reasonable solution. It is hard but we hope it will extremely rewarding. Email required plots (read on) and link to your repo (read on) to (arbaazk@seas.upenn.edu)
Have fun.

Consider a pendulum with a some unknown mass and unknown dynamics as your system (see Fig 1). Your system takes in as input a torque (clockwise or counter clockwise). Your goal is to compute a sequence of actions that ensure the pendulum can stay in an upright position. This is a classic model based problem and can be broken down to 2 pieces :

1. Learn the model of the system.
2. Using the learned model, use a planner to compute the sequence of states required to achieve the desired position.

Part a) Implement a system that learns the model of the pendulum.

The state of the pendulum at any time t is given as $s(t) = [cos(\theta), sin(theta), \theta, \dot{\theta}$. Your output action space is a continuous action space corresponding to the torque τ applied at the hinge $\tau \in (-2, 2)$. To get started with this task :

1. Install the openai-gym package and run through the tutorial on how to get set up with a pendulum environment. (google this)
2. Once you have the simulation set up, learn the dynamics model of the system with any parametric function. For example, if attempting to use a neural network to learn a model, the input to the model is the current state s_t and the current action a_t (the current action is a real valued number representing torque) and the model predicts the next state $\hat{s}_{t+1}$. The neural network attempts to minimize the loss between the predicted next

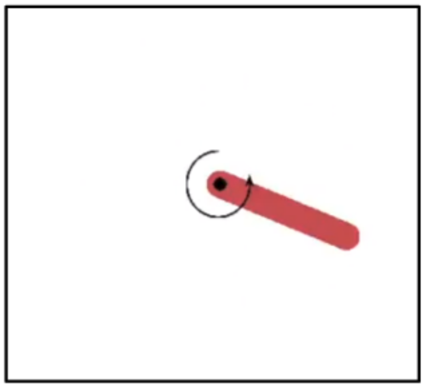

Figure 1: Pendulum

state $\hat{s}_{t+1}$ and the true next state s_{t+1} which can be obtained from the simulator. (Randomly sample actions and states as inputs. Think if this is a good strategy - if you have time google DAgger (dataset aggregation) to understand why this is not a good strategy). To implement a model go through any deep learning tutorial and adapt it such that the input is $[s_t, a_t]$ and output is s_{t+1}

Plot a training curve for your model. Report training error.

Part b Using the learned model from Part a) and implement a graph based planner that can compute the most optimal solution. The easiest graph based planner can be A^* (reads as A-star). For example, A-star takes in as input the current state, evaluates reachable states from the current state by computing a heuristic and then picks the best possible next state according to this heuristic. Thus, to do this subpart :

1. Read up A-star and understand its workings in and out.
2. You first need to discretize your state space. Binning your state space is a good strategy ($-1 \leq \theta \leq 1$ and $-8 \leq \dot{\theta} \leq 8$).
3. Sweep through all your states and for every state find the best possible state by evaluating a heuristic. (a good choice for a heuristic $h = -(\theta^2 + \dot{\theta}^2 + 0.001 * {a_t}^2)$ (Think why)

If you have gotten so far, you are almost there. To recap :

You have a pendulum with some initial state s_0. You wish to output a sequence of actions $a_1, a_2, \ldots a_n$ such that you reach a desired s_{n+1}.

In the first step, you learned what the next state is going to be if fed in information about the current state and an action. Thus, you know all reachable states from your current state. Say you start with s_0, actions possible in s_0 are

a_1 and a_2. a_1 takes you to state s_1 and action a_2 takes you to state s_2. Now using your planner you can evaluate a cost for s_1 and s_2 and evaluate which state has higher value (say it is s_2). Thus, the best action to take in s_0 is a_2 and end up in s_2. Now repeat this process till we reach desired state s_n. Thus, you successfully learned how to control an unknown object.

Zip your plots and email it to arbaazk@seas.upenn.edu. Upload your code to github and send the link of the repo to arbaazk@seas.upenn.edu