Alex Beyer

ENME743

Project Deliverable 2: Baseline Model

This deliverable is written for Standard Project 4: Acrobot Swingup Control.

Problem Formulation

Our problem is to find an optimal swingup control law for the acrobot subject to a reward (regret?) function that provides a reward of -1 for every simulation step where the robot is not swung up, stopping as soon as it does. Optimizing reward means finding the fastest swingup law to terminate the simulation before the cumulative regret gets too large. To do this we can provide a discrete input torque from the set [-1,0,1] to the 2nd joint, enabling inertia based swingups. Importantly, the initial state is slightly randomized between runs, which leads to massive changes in the initial dynamics and makes trying to specify a single optimal law a bad idea.

The problem statement gives a few important details to consider when choosing a model, the easiest to spot being that the slight randomization in the initial conditions make this problem very poorly suited for supervised learning. Supervised learning wants a task that either has a single correct answer or a family of related answers, such that you can extrapolate all of them by providing examples. The chaotic dynamics in this system, when paired with an inconsistent starting configuration means that entirely different control laws made be needed for different start configurations (for example, an arm that starts bent to the right versus bent to the left). Unsupervised learning is also a poor option because we want to optimize over the set of all control laws which will provide a swingup motion; we don't have initial data to give the algorithm. A state space based learning approach would also likely preform poorly here since we aren't trying to estimate a hidden state given a set of observable outputs, we're trying to provide outputs to reach a goal. This leaves reinforcement learning as the best remaining approach, and it also happens to fit the problem well. We have a robot which exists in a current state that we can choose to apply or not apply an action to with the goal of making it transition to a final state in the shortest amount of time, which suits Q-Learning very well.

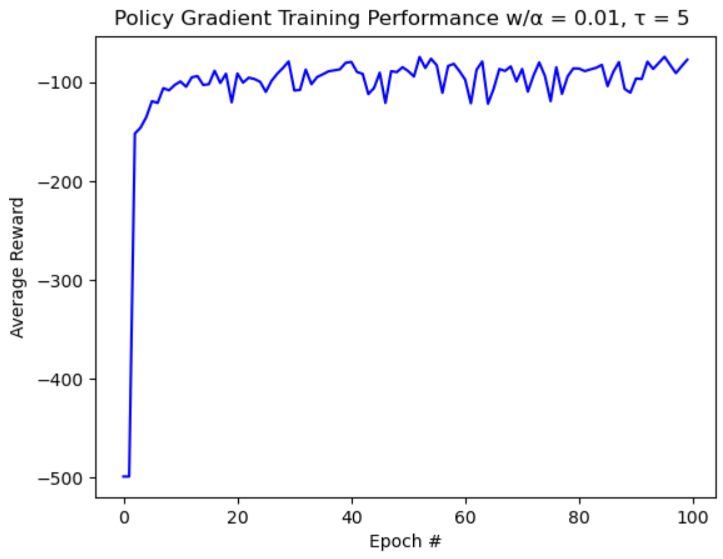
Model Choice

Like was alluded to above, Q-Learning is a good choice of initial model for this problem, it's structured in such a way Q-Learning would be likely to work and, in principle at least, works similarly to certain types of probabilistic control (takes in current state and produces a "best guess" for what the next input should be to get to the desired state). Additionally, to improve model convergence we will add a few extra pieces to our model. We begin with an ε greedy policy which at every step either takes the calculated best option or randomly selects the next action from the set of all possible actions, dependent on our parameter ε. However, because our system is inertia driven we are ultimately relying on a long string of optimal actions to achieve desired performance. This makes the possibility of entirely random actions being added into the chain a problem as they could serve to cancel out some or all of the inertia being used to swing the robot up. To this end we introduce a softmax action selection policy which provides a set of weights for picking the random action, preferring to pick actions more similar to the optimal one for the given state. Not only does this reduce the risk of inertia cancelling actions being taken but it also gives an extra parameter to increase model performance – τ, the 'temperature' for the softmax policy, with higher temperature values corresponding to more equal probability distributions across the action space. This model will not need cross validation because it is a Q-Learning based approach.

In terms of assumptions from the previous deliverable, the main one I could think of was providing some form of dynamics knowledge to the algorithm to stop it from picking control inputs which would disrupt it's own inertia. That has largely been handled by implementing the softmax policy, which allows that to happen occasionally to train the model via exploration but will quickly begin to disincentivize it. In the future I could do more to provide an exponentially decreasing likelihood for exploration as currently the model is as likely to begin exploring during the final episode as it is during the first. I could also try iterating across multiple different solvers and loss functions as well as changing the neurons in my NN to get better results, although that would come down to a lot of trial and error. By the end of the semester I'd expect to have at least a better tuned model, if not one which is able to make more informed decision about when and what to explore.

Test Methodology

For this model success is measured by having the lowest possible cumulative regret (equivalently, the highest possible score). There is a ceiling to this however since it takes time for the model to get in position. Good training results will look like the model starting out with a very high cumulative regret and eventually finding more optimal ways to swing up, with the reward transitioning to a regime of asymptotically approaching a minimum regret (or maximum score). The model should never be able to find a steady state minimum regret because of the randomization in initial parameters so the final regime for the model should consist of episodes whose final rewards oscillate just below the minimum possible regret. A model with well set parameters will not only begin oscillating about the minimum regret faster but should also exhibit smaller oscillations. In the interest of space I won't discuss tuning here but there is a huge amount of interesting detail you can glean about model performance based on how you tune it. For α = .01 (gradient descent rate) and τ = 5 (softmax temperature) this model preforms fairly well.


Policy Gradient Training Performance w/α = 0.01, τ = 5
Average Reward
Epoch #