10-703 Deep RL and Controls
Homework 3
Spring 2017

April 10, 2017

Due April 25, 2017

Instructions

Refer to gradescope for the exact time due.

You may work in up to groups of 4 people on this assignment. Only one person should submit the writeup and code on gradescope. Make sure you mark your partners as a collaborator on gradescope and that both names are listed in the writeup.

Writeups should be submitted as PDF using LaTeX.

Problem 1

In this set of problems you will implement Linear Quadratic Regulation (LQR) and iterative Linear Quadratic Regulator (iLQR). For extra credit you can reuse your iLQR implementation to write a Model Predictive Controller (MPC).

You will be controlling a simple 2-link planar arm. We have provided a simulator for you. It is based on the OpenAI gym API with a couple of additions you will need to approximate the dynamics. The environment comes with rendering code so that you can visually see what the arm is doing.

We have provided you a coding template. We prefer if you adhere to the API given, but if you need to deviate from it then document in your submission what were your changes and why you made them.

You should not need to use the cluster or a GPU for any of the problems in this section.

Environment

You will be using the two link arm simulator that we have provided. The API is inherited from the gym environment API you are familiar with, but with a few extra attributes and function args to make it easier to run LQR, iLQR and MPC.

The environments themselves define a cost matrix Q and a cost matrix R. Use these when calculating your trajectories. The rewards returned by the environment are computed using the LQR cost function.

The _step function includes an additional argument `dt`. When calculating the finite differences your dt will be much smaller than the dt that the simulator normally steps at when calling `step`. So when executing a command you should use `step`. When you are trying to approximate the dynamics using finite differences you should use `_step` and override the dt argument.

You can also explicitly set the state of this simulator using the `state` attribute. You will need this when doing finite differences. Just set this attribute equal to the q and $\dot{q}$ values you want before calling `_step`.

There are a few different flavors of this environment you will use. The differences are summarized in table 1.

Environment Name	**Description**
`TwoLinkArm-v0`	Two link planar arm with fixed goal. Simulation allows infinite torque values. The reward is setup with a high penalty for being away from the goal and low penalty for controls.
`TwoLinkArm-limited-torque-v0`	Same as `TwoLinkArm-v0` but the torques are limited.
`TwoLinkArm-random-goal-v0`	Same as `TwoLinkArm-v0` but the goal changes on each reset.
`TwoLinkArm-limited-torque-random-goal-v0`	Same as `TwoLinkArm-random-goal-v0` but the torque values are clipped.
`TwoLinkArm-v1`	Same as `TwoLinkArm-v0` but high cost for control input.
`TwoLinkArm-limited-torque-v1`	Same as `TwoLinkArm-limited-torque-v0` buthigh cost for control input.
`TwoLinkArm-random-goal-v1`	Same as `TwoLinkArm-random-goal-v0` but high cost for control input.
`TwoLinkArm-limited-torque-random-goal-v1`	Same as `TwoLinkArm-limited-torque-random-goal-v0` but high cost for control input.

Table 1: Table describing the different environments.

LQR

Implement LQR and then perform the following experiments.

1. **[5pts]** Test your LQR implementation on the `TwoLinkArm-v0` environment. Record the total reward and number of steps to reach the goal. Also plot q, $\dot{q}$, and your control inputs u.

2. **[5pts]** Test your LQR implementation on the `TwoLinkArm-limited-torque-v0` environment. Record the total reward and the number of steps to reach the goal. Also plot q, $\dot{q}$, and your control inputs u. Additionally plot u clipped to the action space of this environment.

3. **[5pts]** Compare the performance of your controller on each of these environments.

4. **[5pts]** Test your LQR implementation on the `TwoLinkArm-v1` environment. Record the total reward and number of steps to reach the goal. Also plot q, $\dot{q}$, and your control inputs u.

5. **[5pts]** Test your LQR implementatoin on the `TwoLinkArm-limited-torque-v1` enivornment. Record the total reward and the number of steps to reach the goal. Also plot q, $\dot{q}$, and your control inputs u. Additionally plot u clipped to the action space of this environment.

6. **[5pts]** Compare the performance on these environments to the v0 versions.

iLQR

Implement iLQR using the provided code templates and then perform the following experiments.

Note: iLQR will take a lot longer to run than LQR, but you should still be able to run it on a regular laptop.

Preliminary: Set number of control steps $tN = 100$. The intermediate cost function of intermediate control steps $1 \sim tN - 1$ is $\|u\|^2$, where u is the control parameters. The final cost function for the final step is $10^4 \times \|x_{tN} - x^*\|^2$, where x_{tN} is the final state generated from your algorithm and x^* is the target state. Set the maximum number of optimization iterations to be 10^5. You can add any heuristic stopping criteria to early stop if the algorithm converges.

Suggestions If you face the numerical issue when you are doing the inverse operation, try to add λI to make the matrix full-rank. You are free to use any $\lambda > 0$. For your reference, I used $\lambda = 1$. Also, please start from some simple tasks for debugging.

1. **[10pts]** Test your iLQR implementation on the `TwoLinkArm-v0` environment. Plot the total cost (intermediate cost + final cost) respect to iterations and record the total reward. Also plot q, $\dot{q}$, your control inputs u.

2. **[10pts]** Test your iLQR implementation on the `TwoLinkArm-v1` environment. Plot the total cost (intermediate cost + final cost) respect to iterations and record the total reward. Also plot q, $\dot{q}$, your control inputs u.

3. **[5pts]** Discuss the comparison between iLQR and LQR algorithm, which one is better and why?

Extra Credit: Speed up iLQR [5pts]

iLQR is not as fast as we want. Try to improve the convergence in any way you can figure out. Potential directions include changing cost function and use better optimization procedures.

Extra Credit: MPC [10pts]

Using your iLQR code implement MPC and then perform the same iLQR experiments but with your MPC code. Compare your results with the LQR and iLQR algorithms.

Problem 2

In this problem, you will implement behavior cloning using supervised imitation learning from an expert policy. We have provided you with an expert for the CartPole-v0 environment. You will use this expert to generate training data, and then train a cloned model and evaluate it.

Note: This problem requires keras/tensorflow, but it should train in a couple of seconds even on a CPU. You should not need the cluster or a GPU for this problem.

Imitation Module

We have provided you with some function templates that you should implement. If you need to modify the function signatures, you may do so, but specify in your report what you changed and why.

All of the provided code is in the `deeprl_hw3.imitation` module.

You can load the expert model using the `load_model` function. We have provided you with a model config file called `CartPole-v0_config.yaml` and a set of expert weights called `CartPole-v0_weights.hf5`. You can load them using the following code snippet:

```
expert = deeprl_hw3.imitation.load_model('CartPole-v0_config.yaml',
                                         'CartPole-v0_weights.h5f')
```

`expert` is a Kears Model instance. Use this expert to generate a training dataset that you can use with the Keras `fit` method. You will need to collect states and a one-hot encoding of the expert selected actions.

We have provided you with a `test_cloned_policy` method that you can use to compare the performance of the expert and cloned model.

The default initialization of the CartPole-v0 environment is too easy for the behavior cloning model. You shouldn't be able to tell a difference between the expert and cloned model with the default initial states. So we have provided you a wrapper function thta will initialize the environment in hard to recover from states. Use the `wrap_cartpole` function on a CartPole-v0 instance to get this harder version of the environment. Using this version you should see that your cloned model does worse than an expert on the same domain.

Here is a code snippet demonstrating how to use the wrapper:

```
env = gym.make('CartPole-v0')
env = deeprl_hw3.imitation.wrap_cartpole(env)
test_cloned_policy(env, cloned_policy)
```

Behavior Cloning

Start by implementing the `generate_expert_training_data` function. Then generate training datasets consisting of 1, 10, 50, and 100 expert episodes.

When cloning the behavior we recommend using the Keras `fit` method. You should compile the model with Adam as the optimizer, binary_crossentropy as the objective function, and include 'accuracy' as a metric.

1. [**10pts**] Use each of your datasets to train a cloned behavior using the dataset as a supervised learning problem. Record the final loss and accuracy of your model after training for at least 50 epochs. Make sure you include any hyperparameters in your report.

2. [**10pts**] Evaluate each of your cloned models on the `CartPole-v0` model using the `test_cloned_policy` method. How does the amount of training data affect the cloned policy?

3. [**10pts**] Evaluate each of your cloned models on the `CartPole-v0` domain wrapped with the `wrap_cartpole` function. Also evaluate the expert policy. How does your cloned behavior compare with respect to the expert policies and each other.

Extra Credit: DAGGER

In the previous problem you saw that when the cloned agent is in states far from normal expert demonstration states, it does a worse job of controlling the cart-pole than the expert. In this problem you can implement the DAGGER algorithm [3] for extra credit.

Using your DAGGER implementation answer the following questions.

1. [**10pts**] Clone the expert behavior. Plot learning curves of the cloned behavior. In otherwords, every k episodes, freeze the current cloned policy and run 100 test episodes. Average the total reward and track the min and max reward. Plot the total reward on the y-axis with min/max values as error-bars vs the number of training episodes.

2. [**10pts**] Evaluate your cloned policy on the wrapped cartpole like in part 3 of the Behavior Cloning section. Compare your cloned policies average performance to the expert and your best cloned policy from the Behavior Cloning section.

Problem 3

In this section you will implement REINFORCE and test on the CartPole-v0 domain. REINFORCE is a policy gradient method that was discussed in class. We recommend you also take a look at page 270 of Sutton and Barto's book [1]

[1]`http://incompleteideas.net/sutton/book/bookdraft2016sep.pdf`

You can use the same model configuration from the Behavior Cloning section. For REINFORCE you need a stochastic policy. CartPole-v0 is a discrete action space, so rather than training to match Q-values on the network, we will train the network to output the correct action probabilities. The provided model config already has a softmax output so you shouldn't have to modify the model.

Looking at the pseudocde for REINFORCE you'll see that you need to take the gradient of the log of the policy. This is quite easy to do with Tensorflow. Simply take the output tensor of the Keras model, call `tf.log` on that tensor and then use the `tf.gradients` method to get the gradients of the network parameters with respect to this log. You will also have to scale by the rewards from your sampled policy runs (i.e. scale by G).

Note: While you are using Keras and Tensorflow for this problem, the models are small enough that yo ucan train on a laptop CPU. You should not need the cluster or a GPU for this assignment.

1. [**10pts**] Train REINFORCE on CartPole-v0 until convergence. Show your learning curves for the agent. In otherwords, every k episodes, freeze the current cloned policy and run 100 test episodes. Average the total reward and track the min and max reward. Plot the total reward on the y-axis with min/max values as error-bars vs the number of training episodes.

2. [**10pts**] How does this policy compare to our provided expert and your cloned models?

References

[1] J Andrew Bagnell. An invitation to imitation. Technical report, DTIC Document, 2015.

[2] Stephane Ross. Interactive learning for sequential decisions and predictions. 2013.

[3] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.