Robot Learning
Weekly Exercise 3

Marc Toussaint & Wolfgang Hönig
Learning & Intelligent Systems Lab, Intelligent Multi-Robot Coordination Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Summer 2024

1 Literature: DAgger

The following paper introduces DAgger (short for "Dataset Aggregation"):

S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, 2011-03-16. URL: http://arxiv.org/abs/1011.0686, arXiv:1011.0686, doi:10.48550/arXiv.1011.0686

a) First have a look at Section 5 (Experiments), and if you like, the youtube video https://www.youtube.com/watch?v=V00npNnWzSU. Two basic questions about what is mentioned in 5.1:
   - The method uses a regression technique to train the policy $\pi : y \mapsto u$ (y are observations). Which technique is used?
   - Fig. 2 mentions β_i, which is a parameter of the method that changes with iteration i. How exactly is it chosen?

Footnote 4 on page 6: Ridge regression

In the experiment of Fig. 2 $\beta_1 = 1$ in the first iteration (following exactly the expert π^*), and $\beta_i = 0$ for later iterations (generating only data by own policy $\widehat{\pi}_i$). In other experiments (page 7, top right) β_i is an exponential schedule.

b) Now look at the pseudo code Alg. 3.1 on page 4. The introduction of Sec. 3 explains the pseudo code. The lines 4 and 5 ("Let π_i...", and "Sample T-step...") are perhaps the hardest to really understand. Your exercise: Write explicit pseudo code of how you generate such a "T-step trajectory using π_i", where this pseudo code can only call the dynamics function $x_t = f(x_{t\text{-}1}, u_{t\text{-}1})$, the expert policy $u_t = \pi^*(x_t)$, the trained policy $u_t = \widehat{\pi}_i(x_t)$, and a state initialization method $x_0 \sim p(x_0)$.

   Note: Line 4 defines π_i to be a probabilistic mixing of policies π^* and $\widehat{\pi}_i$, with coefficients β_i and $1-\beta_i$ respectively. This notation is typically used when π are stochastic policies, but "implicitly clear" also when they are deterministic.

(Using notation s, a, as in the paper, and `rand()` returns number uniform in $[0, 1]$.

1: init $D_i = \emptyset$, and startnstate $s \sim p(s_0 = s)$
2: **for** $t = 1 : T$ **do**
3: $\quad a^* \leftarrow \pi^*(s)$, $\widehat{a} \leftarrow \widehat{\pi}_i(s)$
4: $\quad$ if `rand()`$\leq \beta_i$ then $a \leftarrow a^*$ else $a \leftarrow \widehat{a}$
5: $\quad D \leftarrow D \cup \{(s, a^*)\}$
6: $\quad s \leftarrow f(s, a)$
7: **end for**

2 Trajectory Distributions, GMMs, ProMPs

Imitation learning can also be formulated as learning the distribution of demonstrated trajectories (rather than directly the policy), and thereafter use control theory to derive controllers that imitate this distribution. The following paper is a typical representative for using Gaussian Mixture Models (GMMs) to learn the distribution of demonstrated trajectories:

S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, pages 255–262. ACM, 2007-03-10. URL: https://dl.acm.org/doi/10.1145/1228716.1228751, doi:10.1145/1228716.1228751

Only have a look at Figures 3 and 6 – they should clarify what it means to use Gaussians to "cover" the distribution of demonstrated trajectories, and thereby learn the distribution. To enable this, a trajectory $x_t \in \mathbb{R}^n$ for $t = 1, .., T$ is embedded in $n + 1$-dimensional space (t, x_t), and then standard density estimation using GMMs applied.

Consider a dataset $D = \{x_t^i\}_{t=1,..,T}^{i=1,2}$ with two 1-dimensional trajectories of length T, namely these two:

- First demonstrated trajectory $x_t^1 = \cos(t/3)$ for $t = 1, .., 20$
- Second demonstrated trajectory $x_t^2 = \cos(t/3 - 1)$ for $t = 1, .., 20$

a) Plot both of these demonstrations

```
plot [0:20] cos(x/3), cos(x/3-1)
```

b) Assume you would fit a Gaussian Mixture Model with 2 components (2 Gaussians) to this data (using a time-embedding as above), how might it look like? (Sketch on paper. Where might be their centers and the ellipse illustrating their covariance matrices?) Conditioning this distribution on a particular t, e.g. $t = 11$, what would be the conditional variance over x? (Just argue in terms of your sketching.)

Consider using 'equal scale' in the plot... (We're mixing time and state scales in the embedding - should GMM modelling really depend on the time scale?)

The 2 modes might capture the higher variance parts.

Conditioning is interesting! Whereever you cut through a Gaussian, the conditional variance is the same along a particular direction! Therefore, conditional variances are not small where you might naively expect.

c) Consider a fully different approach: Treat each x^i as a vector with 20 entries x_t^i. The two vectors x^1 and x^2 form our tiny data set $D = \{x^i\}_{i=1,2}$. From this data we can estimate the element-wise mean μ_t and standard deviation σ_t for each t. Sketch these analogous to the above.

[Note: The latter approach is called ProMP (Paraschos et al, NeurIPS'13).]

Here we really have low (conditional) variances, e.g. at t=2, 11, 20. Compare to Gaussian Processes!

3 Mountain Car Imitation Learning

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)

We use the same mountain car example as in Exercise 2, so look for more detailed instructions there, if you haven't set it up, yet.

The following "policy" was written by an expert to solve the control problem:

```
def expert(t):
    if t < 50:
        return np.array([-1.0])
    elif t < 100:
        return np.array([1.0])
    return np.array([0.0])
```

Note that this uses the time step t and not the state as input, which is why we put "policy" in quotes.

a) Collect a sufficient amount of data and learn a real policy, i.e., a function that maps from the current state to the action. Report your achieved loss.

You may still use any ML technique, including linear regression. However, this might also be a good starting point to use pyTorch, so that you have some experience with more complicated exercises later. You can follow the official tutorial at https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html.

Hints: You can convert data using `torch.from_numpy(data_input).float()`. A useful function is `torch.utils.data.random_split`. From the tutorial, make sure you adjust the neural network and loss function to match our target domain.

See separate example code.

b) Validate your learned policy in the gym environment. What happens if you start from a starting state that was not part of your training data (e.g., use `env.reset(options='low': 0.1, 'high': 0.4)`)?

See separate example code. Generally, the policy works as long as a suitable activation function (e.g., tanh) was used. Note that we do not check the reward here and that the resulting solution does not have the best possible reward.

c) Can DAgger help here to collect a better dataset? Explain why or why not.

No, because DAgger requires an expert policy that maps states to actions.

References

[1] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, pages 255–262. ACM, 2007-03-10. URL: https://dl.acm.org/doi/10.1145/1228716.1228751, doi:10.1145/1228716.1228751.

[2] S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, 2011-03-16. URL: http://arxiv.org/abs/1011.0686, arXiv:1011.0686, doi:10.48550/arXiv.1011.0686.