**1. Behavior Cloning**

The purpose of behavior cloning is to build an agent as mapping from observation space to action space M: O -> S. In the computation, an expert policy serves as the true mapping from O to S, and the agent replicates the behavior of the expert policy by treating the mapping as a supervised learning problem, where the observations are the input and the actions are the output.

In the imitating, one of the underlying assumption is the Markov property: the action of the expert agent taken at time t is only a function of its observation at time t, and has nothing to do with the observations before time t. This assumption can be fixed by using more complicated neural networks, such as convolutional neural network or recurrent neural network.

The other problem of the behavior cloning is that the expert policy only provides data along an optimum path. Therefore, the cloning agent will not be able know what is the appropriate action when it makes a small mistake and deviates from the optimum path. Then the cloning agent will make bigger mistake since it has less clear idea of what to do at this new position, and such errors accumulates into large errors. In one word, the system is unstable if it is simply trying to clone the path from the optimum expert agent.

The system is not stable, and this can be regarded as the result that the system has no feedback. The system is an open-loop system, and any open-loop system is intrinsically unstable.

The instability of the system can be attributed to the mismatch of data distribution: The training data have a distribution around the optimum trajectory, but the testing data have a distribution that can be arbitrarily scattered in the state space.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Environment | Mean Return (Expert) | Mean Return (Clone) | Std Return (Expert) | Std Return  (Clone) | Comment |
| Humanoid | 10419 | 3882 | 43 | 3287 | When the humanoid falls off, it cannot stand up since this case is never met from the data of expert policy. The change of success depends critically on the initial stage of walking. The score also depends critically on whether the simulation is terminated only after maximum steps or after the simulator determines the humanoid will fall. |
| Reacher | -3.94 | -4.13 | 1.73 | 1.82 | The score of clone is very close to the score of expert, because the system itself is relatively independent from early history. |
| Hopper | 3778 | 2889 | 4 | 664 | The problem is the same as the humanoid: when the hopper deviates from common stances, it does not know how to recover. |
| Walker2d | 5523 | 5499 | 63 | 41 | Surprisingly, the clone performs as well as the expert policy. |

2. DAGGER (Dataset Aggregation)

The DAGGER algorithm solves the problem of data distribution mismatch by collecting training data around the possible trajectory from the clone policy instead from the possible trajectory from expert policy: Assume the expert policy is a mapping from any of the state space observation to the action. If we let the expert policy runs and use the trajectory data of the expert policy to train the clone, the clone will not have any training data on its own imperfect trajectory. Instead, we let the clone gets its trajectory, and let the expert policy converts the state space observation from the trajectory of clone into the training data by labeling the corresponding observation with the correct action. In this way, we get (observation, action) pair along the trajectory of the clone instead of the expert.

After the model was trained in DAGGER, the humanoid model can always perform a good job until the number of steps reaches the maximum. In contrast, the humanoid model trained simply in the expert policy data can only run about 30% of maximum step on average. The model trained in DAGGER has average reward of 9956 and standard deviation of 888, which is comparable to the expert policy.