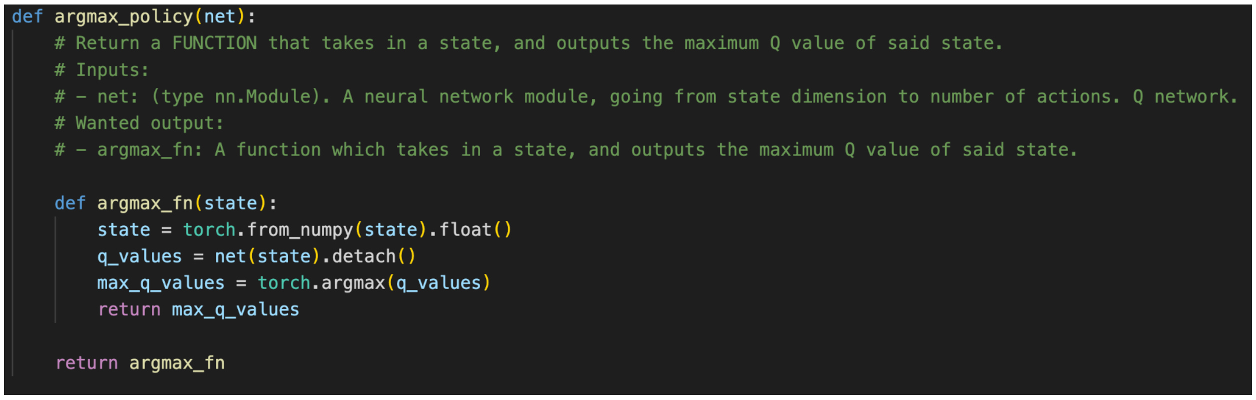
Assignment 2: Imitation Learning on Underactuated Systems
CS 6756, Fall 2022

Stanley Celestin

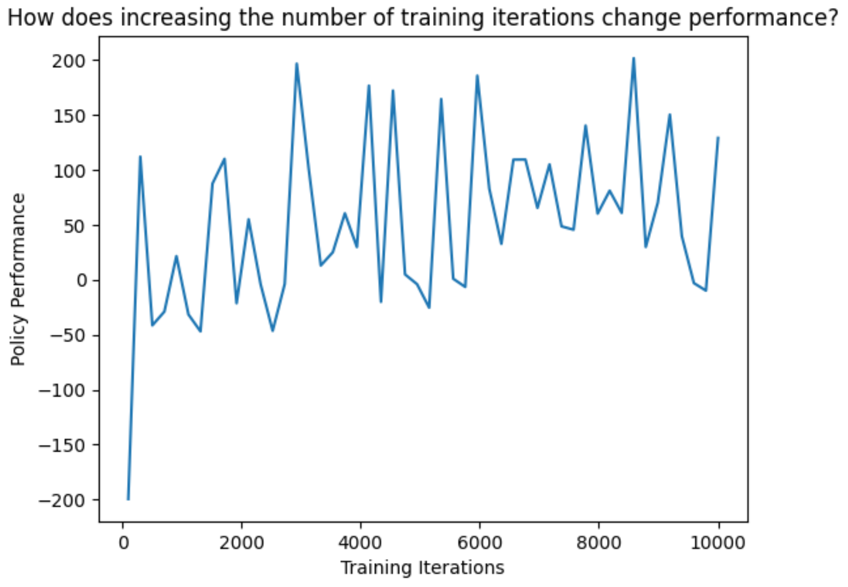
Simple utilities (also in code submission):



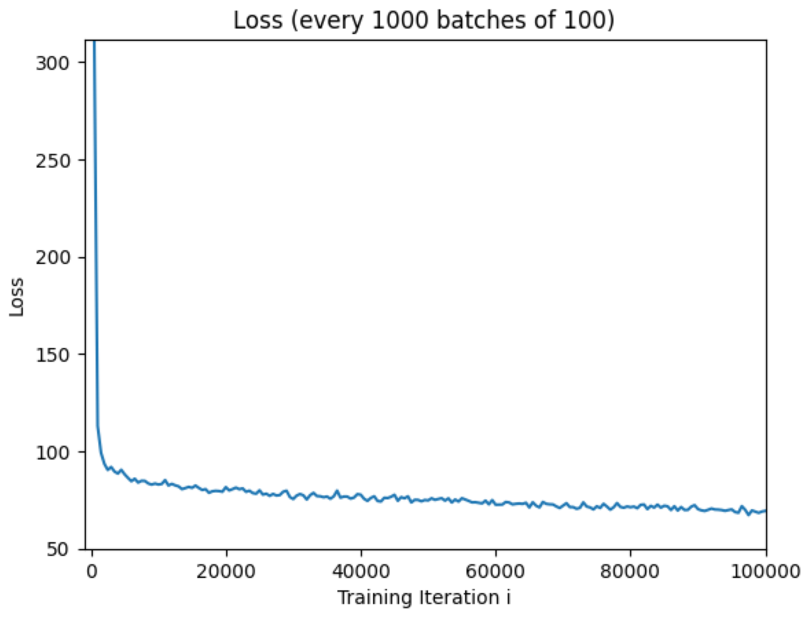
```python
def argmax_policy(net):
    # Return a FUNCTION that takes in a state, and outputs the maximum Q value of said state.
    # Inputs:
    # - net: (type nn.Module). A neural network module, going from state dimension to number of actions. Q network.
    # Wanted output:
    # - argmax_fn: A function which takes in a state, and outputs the maximum Q value of said state.

    def argmax_fn(state):
        state = torch.from_numpy(state).float()
        q_values = net(state).detach()
        max_q_values = torch.argmax(q_values)
        return max_q_values

    return argmax_fn
```

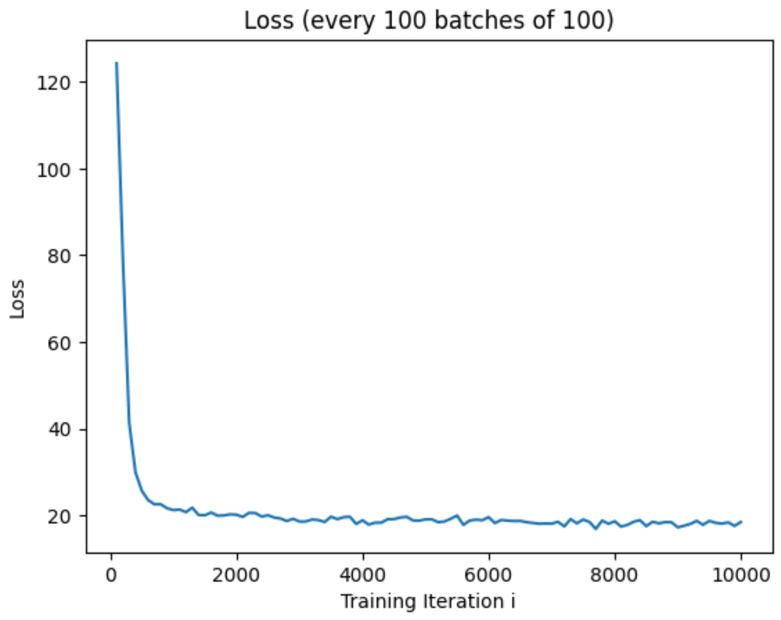
1. <u>How well does the BC **argmax_policy** perform when rolled out (total rewards)? As you increase the number of training iterations of BC, what happens to train / validation loss / total reward?</u>

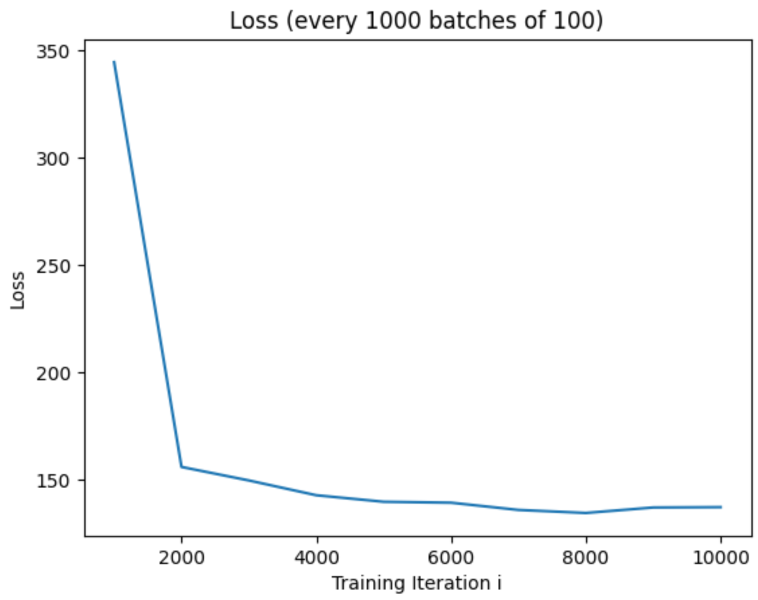
In order to answer this question, I trained different BC argmax policies that each have a different number of training iterations and tested their performance using **eval_policy()**. The performance of BC does not necessarily increase because it still suffers from the distribution shift.



I also tracked the loss during training to see how it changed through the training iterations. From the three plots below we can see that after a certain point rate of change of the loss function is loss, suggesting that increasing the training iteration any more will not have any significant effect on the loss.

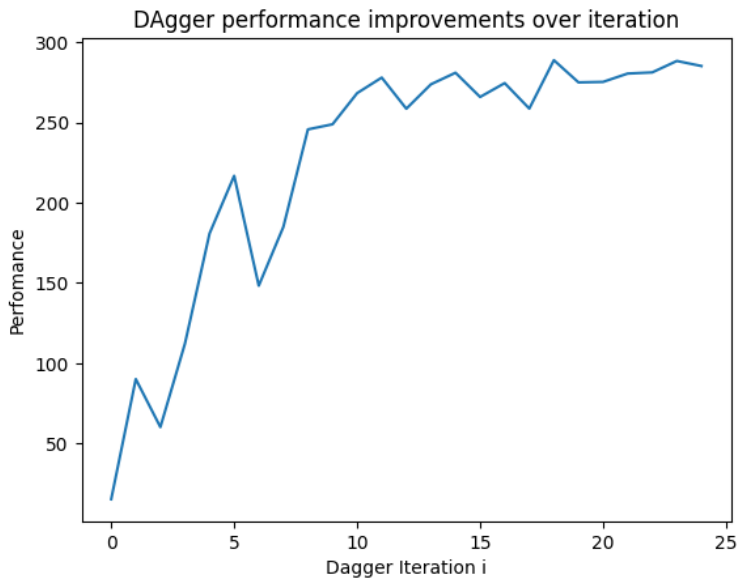






2. <u>How well does the DAgger **argmax_policy** policy perform when rolled out (total rewards)? How does the loss and total reward vary over DAgger iterations?</u>

DAgger argmax_policy performs just about as well as the expert with just N=5 iterations. This is because on each iteration of improvement using expert actions, it is able to decrease the distribution shift. As shown below as the number of iterations increases, DAgger approaches the expert performance.

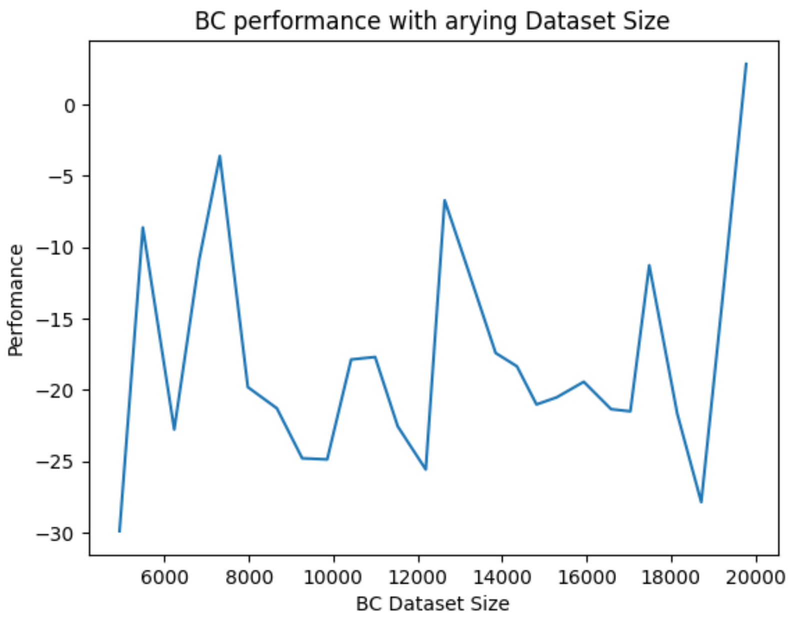


3. <u>If performance is significantly better or worse for any agent over another, what do you think the reason is?</u>

The performance of DAgger is clearly better than behavior cloning because BC cloning has no way of improving from covariate shift. In the DAgger algorithm, by incorporating what the expert would do in each state that was traversed using the learners policy, a new policy is able to to learn by augmenting previous data with this new information. Essentially, the learner is able to see more situations where it made bad judgment calls in the previous iteration. I really like this algorithm because it mimics how humans learn in certain situations. It's analogous to film study by athletes where they review their previous game and get feedback on what they should have done in each situation.

4. <u>Some easy extensions that we can check is to see how the size of the dataset plays a role in BC performance. Ideally, the larger the dataset, the more signal BC will have to train on, so hopefully we can see better performance!</u>

As seen below as the size of the dataset increases, we don't observe any trend that would indicate performance improvement. I think this is due to the fact the expert behaves optimally so the dataset does not have a lot of examples of the expert "doing the wrong actions" and recovering from it while it is possible for the learner to take the wrong actions and not recover. This is where DAgger proved to be better as discussed in the previous question.


BC performance with arying Dataset Size
Perfomance
BC Dataset Size