University of California Berkeley

Computer Science Department

CS 294-112

HW1: Imitation Learning using Behavioral Cloning and DAgger

Submitted by
Cristobal Pais
CS 294-112
Sept 5th
Fall 2018

HW1: Imitation Learning using Behavioral Cloning and DAgger

Cristobal Pais

Sept 5th 2018

Question 2.2: Behavioral cloning performance comparison.

NRolls	Metric	Agent	HalfCheetah	Humanoid
1	AVGReturn	Expert	4147.1	10368.1
		BC	3949.5	226.4
	STD	Expert	0.0	0.0
		BC	0.0	0.0
5	AVGReturn	Expert	4141.5	10446.3
		BC	4172.7	463.6
	STD	Expert	29.3	57.4
		BC	72.1	118.1
10	AVGReturn	Expert	4157.8	10406.4
		BC	4168.1	737.9
	STD	Expert	78.5	60.2
		BC	59.1	378.3
15	AVGReturn	Expert	4158.4	10394.2
		BC	4136.0	812.0
	STD	Expert	72.3	55.1
		BC	84.3	383.8
20	AVGReturn	Expert	4149.5	10402.3
		BC	4156.6	885.0
	STD	Expert	93.2	39.8
		BC	87.4	542.1
25	AVGReturn	Expert	4143.7	10385.3
		BC	4104.3	1021.7
	STD	Expert	76.0	47.2
		BC	98.2	578.0
50	AVGReturn	Expert	4127.2	10373.1
		BC	4110.0	1195.8
	STD	Expert	77.8	124.1
		BC	80.0	874.7
75	AVGReturn	Expert	4137.4	10401.5
		BC	4152.3	1555.0
	STD	Expert	82.4	62.3
		BC	73.9	1046.2
100	AVGReturn	Expert	4145.4	10400.1
		BC	4140.6	1874.5
	STD	Expert	82.0	56.6
		BC	81.2	1265.3

Table 1: Multiple roll-outs with 1000 time-steps comparison between the Expert Policy and the implemented Behavioral Cloning (BC) model for HalfCheetah and Humanoid instances. The BC model consists of a simple full-dense neural network with 3 layers (input, hidden, and output) with a default configuration of 64 and 32 neurons using hyperbolic tangent (tf.nn.tanh) activation functions. A batch-size equal to 64 observations and a total of 1000 training epochs are used with ADAM optimizer for minimizing the MSE (loss function). This initial simple but useful model — great performance among all other tasks as seen in Appendix — is used as a starting point for developing a new model for performance comparisons (question 4.1).

As can be seen from the results, very similar performance to the expert policy is obtained with any number of demonstrations by the BC model when learning from the expert HalfCheetah policy, even reaching better average returns and standard deviation values depending on the number of roll-outs tested (e.g. with 10, 20, and 75 demonstrations). On the other hand, significant poor performance is reached by our BC implementation when learning from the Humanoid expert policy, reaching at most (100 roll-outs) 20% of the expert's average reward. Therefore, the default configuration of our BC algorithm is not able to capture the complexities of this instance.

A full summary of all tested instances is shown in the Appendix section, Table 2.

Question 2.3: Behavioral cloning Hyper-parameter tuning.

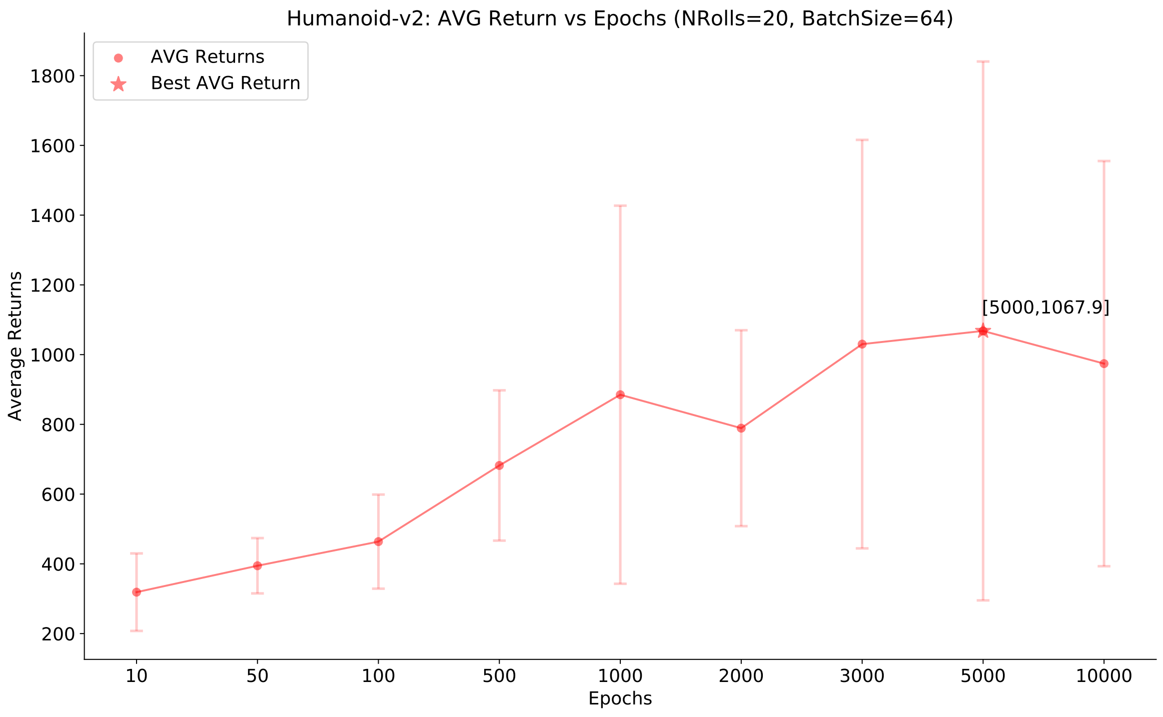


Figure 1: We experimented with the number of training epochs of the Neural Network (NN) model since its impact in the performance of the BC policy is usually significant — with a fixed number of roll-outs and batch-size when training the model — and the obtained results can be clearly compared when experimenting with the Humanoid task. In addition, it is one of the most popular hyper-parameters in Deep Reinforcement Learning models (and Machine Learning in general). From the graph, we can see how increasing the number of training epochs tends to have a positive impact on the average return values of the BC policy: the more time we spent training the model, the better the results of our clone obtaining its best performance with the second largest number of tested epochs (5,000).
However, running times of the whole training of the model are significantly increased with respect to this hyper-parameter, generating an important trade-off between running time versus performance of the trained policy depending on the time constraints of the research project, and thus, its final selection must be carefully taken into account in order to obtain good and suitable results. In addition, the more we train our model, the higher chances of incurring into the over-fitting phenomenon, a situation that can be relevant depending on the environment and context where the policy will be applied.
Analogous results in terms of the increasing performance trend are obtained when using different batches-sizes, with the main difference of reaching different average returns levels (e.g. lower average returns are obtained when using smaller batch sizes as seen in Figure 5.

Question 3.2: DAgger performance comparison.

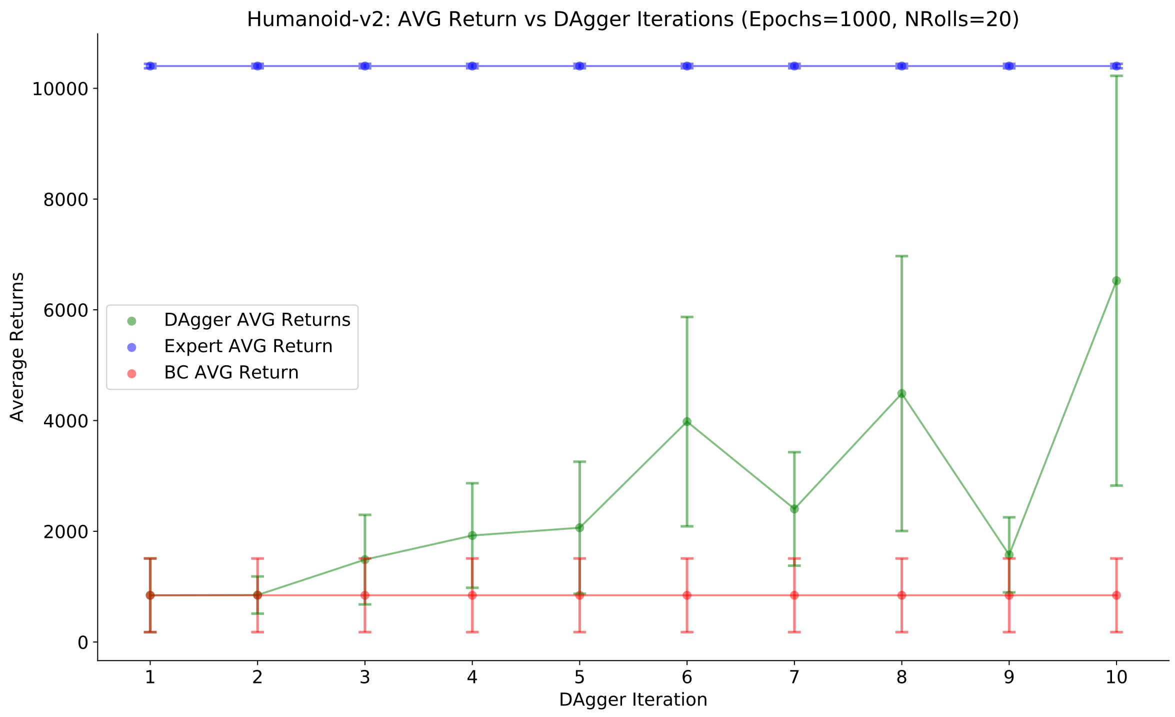
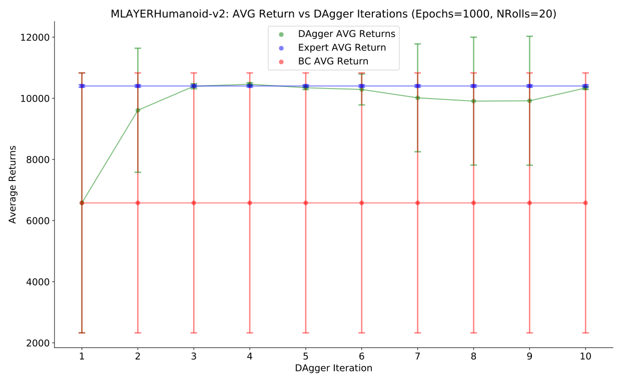
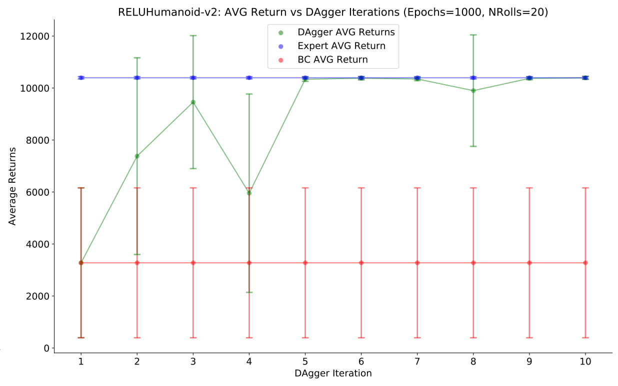


Figure 2: Using DAgger with the Humanoid task, we are able to significantly improve the performance of our clone's policy. As expected, the first iteration of the DAgger algorithm matches the BC policy performance, and then, we can observe from the plot how the average returns are constantly increasing up to iteration number 6 of the algorithm (green line). After this iteration, the performance shows an erratic pattern up to iteration 10 where the best average return using the new policy is obtained (AVG return = 6525.721, STD = 3699.920). However, we can notice how the standard deviation is also very large (larger than half of the mean return value) introducing more risk in our model when generating actions that could lead to poor performance. However, we are able to indicate that the DAgger algorithm has been successfully applied in the current task, leading to significantly improve our trained policy. Potentially, more iterations could lead to better performance as the new model converges towards the Expert's policy average returns.
The comparison is performed using a default configuration with 1000 training epochs, 20 roll-outs, the original 3 layers fully-dense Neural Network model (using the same number of neurons and activation functions as in Question 2.2) and a maximum of 10 DAgger iterations. Important is to remember the fact that every new iteration of the DAgger algorithm contains more data than the previous one ($D_{k+1} \leftarrow D_k \cup D_\pi$, k = the number of iteration) and thus, every new iteration requires a larger training time than the previous ones. A summary of all the DAgger results applied to the different tasks is included in the Appendix, Figure 6.

Question 4.1: Modifying the policy architecture



(a) Multilayer model (128, 64x3, and 128 neurons)



(b) RELU activation functions

Figure 3: In order to check the performance of alternative policy structures, two main modifications from the original model are tested with the Humanoid task (the most challenging one): (1) Number of layers/neurons per layer and (2) the activation functions used on each layer. By modifying the main model class file (DRL_BCModel.py) we allow our model-object to get the number of neurons/layers and activation functions as inputs, giving more flexibility and expression to the new policy (see DRL_BCModel_Free.py). We keep the original default configuration of 20 roll-outs, 1000 epochs, and a batch-size equal to 64 observations for all the experiments.

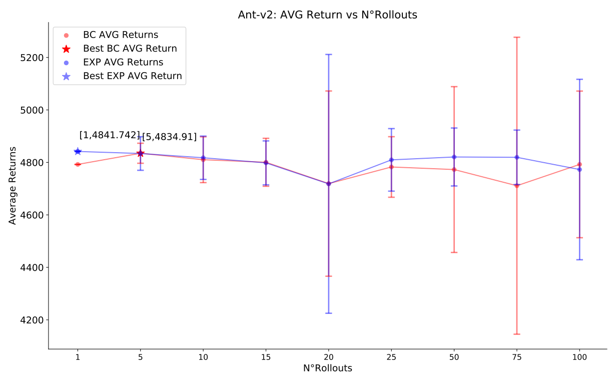
We start by adding more neurons and layers to our original implementation. In Figure (a) we show the results obtained for a fully-dense NN model containing 128, 64, 64, 64, and 128 neurons per layer, all of them using the hyperbolic tangent activation function as in our original model. From the plot, we can see that the new BC and DAgger implementations are clearly superior to our original model: the BC policy (first iteration of DAgger) reaches better average return levels than our previous best result obtained after 10 iterations of DAgger in Question 3.2 and we are able to obtain similar performance to the Expert policy right after the third iteration of the new DAgger algorithm with small standard deviation values (e.g. AVG = 10459.958 and STD = 49.956 on iteration 4). Therefore, we are able to conclude that this new policy architecture is able to capture the complexities of the most challenging task (Humanoid) and thus, obtaining a performance close to the Expert's policy. However, the researcher should be careful since running times are also increased due to the addition of new layers and neurons.

On the other hand, we can see in Figure (b) the results obtained using DAgger algorithm when the original activation functions are replaced by standard RELU functions using the original policy structure (64x32 net). In this case, we can also see an improvement of the model's performance with respect to our original implementation with hyperbolic tangent activation functions, being able to improve the average returns after 10 iterations of the DAgger algorithm. This is mainly due to the better convergence reached when training the model with RELU as the main activation function on the neurons inside our model for the Humanoid task. We can see that, as with the multilayer model in (a), the new policy architecture is able to reach comparable results to the ones obtained by the Expert policy (iteration 5). Other activation functions were tested, obtaining similar performance to our original implementation with some of them (e.g. Sigmoid). Finally, interesting is to mention that all models had also been tested using Keras API within TensorFlow, obtaining analogous results.

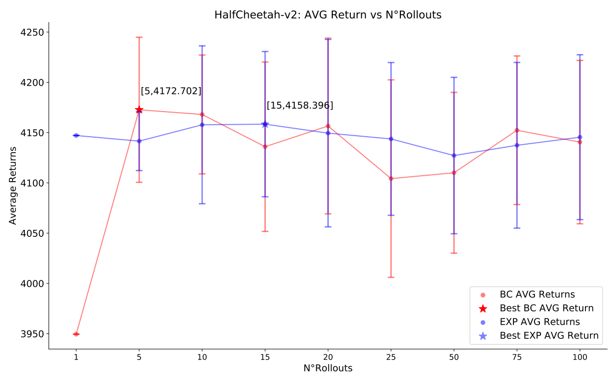
Appendix

NRolls	Metric	Agent	Ant	Half-Cheetah	Hopper	Humanoid	Reacher	Walker2d
1	AVGReturn	Expert	4841.7	4147.1	3778.8	10368.1	-3.7	5567.5
		BC	4792.4	3949.5	696.0	226.4	-55.3	2275.4
	STD	Expert	0.0	0.0	0.0	0.0	0.0	0.0
		BC	0.0	0.0	0.0	0.0	0.0	0.0
5	AVGReturn	Expert	4834.0	4141.5	3776.7	10446.3	-2.4	5552.3
		BC	4834.9	4172.7	1222.3	463.6	-8.9	5515.9
	STD	Expert	64.0	29.3	3.2	57.4	1.1	26.8
		BC	38.2	72.1	223.2	118.1	4.4	48.4
10	AVGReturn	Expert	4817.7	4157.8	3775.1	10406.4	-4.0	5519.0
		BC	4810.3	4168.1	3059.3	737.9	-12.2	5520.1
	STD	Expert	82.5	78.5	3.5	60.2	0.5	48.6
		BC	87.3	59.1	768.0	378.3	2.0	40.3
15	AVGReturn	Expert	4798.2	4158.4	3777.8	10394.2	-4.3	5535.1
		BC	4800.5	4136.0	2408.2	812.0	-9.0	5497.6
	STD	Expert	83.7	72.3	2.7	55.1	2.2	53.4
		BC	91.4	84.3	827.9	383.8	3.4	43.3
20	AVGReturn	Expert	4718.2	4149.5	3778.7	10402.3	-3.5	5530.8
		BC	4719.2	4156.6	3676.0	885.0	-6.4	5487.4
	STD	Expert	493.2	93.2	5.1	39.8	1.4	48.8
		BC	353.0	87.4	444.2	542.1	2.8	104.5
25	AVGReturn	Expert	4809.7	4143.7	3778.7	10385.3	-4.2	5546.4
		BC	4782.6	4104.3	3775.2	1021.7	-5.0	5485.2
	STD	Expert	119.2	76.0	2.5	47.2	1.5	31.3
		BC	115.4	98.2	3.6	578.0	2.0	59.8
50	AVGReturn	Expert	4820.7	4127.2	3777.0	10373.1	-4.1	5519.6
		BC	4772.8	4110.0	3732.9	1195.8	-4.1	5508.3
	STD	Expert	110.5	77.8	3.8	124.1	1.9	63.0
		BC	316.0	80.0	187.6	874.7	1.4	47.3
75	AVGReturn	Expert	4819.4	4137.4	3777.4	10401.5	-4.4	5498.5
		BC	4711.1	4152.3	3579.5	1555.0	-4.0	5502.5
	STD	Expert	103.8	82.4	3.9	62.3	1.9	169.1
		BC	565.9	73.9	242.6	1046.2	1.5	76.4
100	AVGReturn	Expert	4772.9	4145.4	3777.5	10400.1	-3.7	5511.1
		BC	4792.4	4140.6	3172.6	1874.5	-3.9	5047.5
	STD	Expert	343.9	82.0	3.6	56.6	1.7	74.5
		BC	279.4	81.2	770.1	1265.3	1.7	967.6

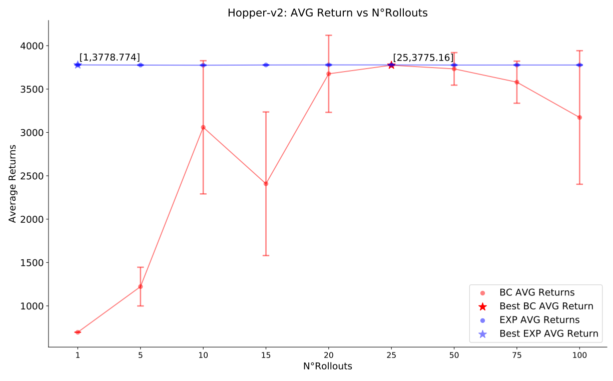
Table 2: Average returns and standard deviation results comparison for all tested instances with the Expert policy and the Behavioral Cloning (BC) initial algorithm using 1000 epochs for training the model, 1000 time-steps per roll-out, and a batch size equal to 64. As described in Table 1, the worst performance of the BC implementation is obtained in the Humanoid instance, mainly due to its inherent complexity — complex interaction between its components — and the simplicity of our original Neural Network (3 layers) implementation, not able to capture the relevant patterns of this task. High-performance results can be seen for all the rest of instances, thus, the BC policy is able to capture the behavior of the expert in the majority of the tasks.



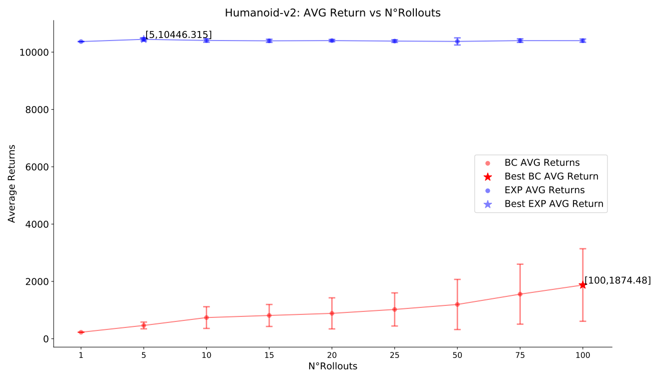
(a) Ant-v2 BC vs Expert.



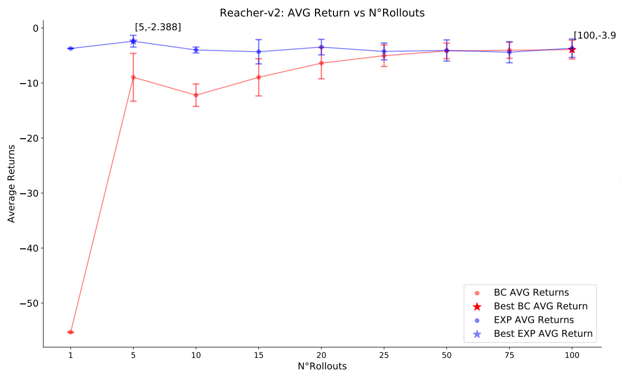
(b) HalfCheetah-v2 BC vs Expert



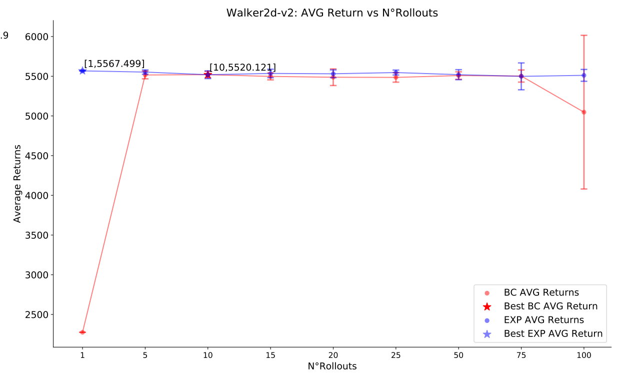
(c) Hopper-v2 BC vs Expert.



(d) Humanoid-v2 BC vs Expert



(e) Reacher-v2 BC vs Expert.



(f) Walker-v2 BC vs Expert

Figure 4: Comparison plots for the BC model performance versus the Expert policy average reward when using different roll-out values (1000 training epochs, 1000 time-steps, original 64x32 fully-dense NN model). The standard deviation of the results is represented by vertical lines.

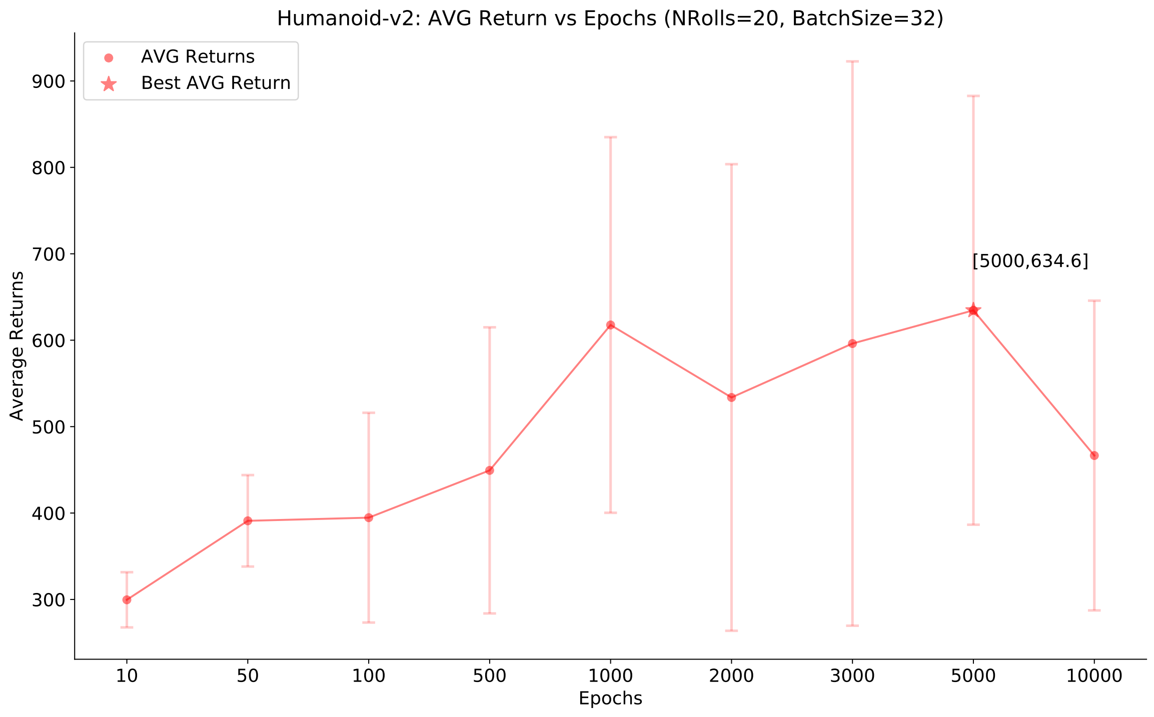
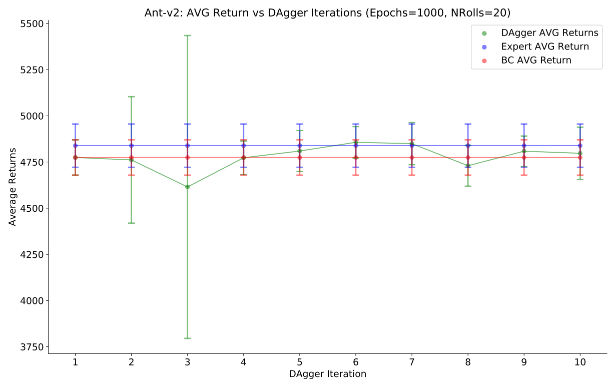
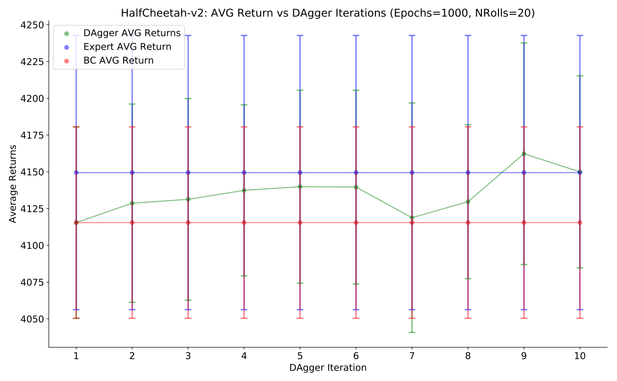


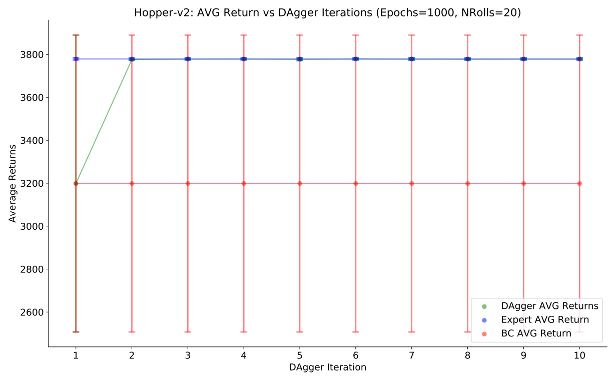
Figure 5: Average return values versus the number of training epochs for the Humanoid task using a batch-size equal to 32 observations and 20 roll-outs. The same increasing trend in performance is observed as when using larger batch-size values but lower average return values are obtained when comparing it with the batch-size equal to 64 experiment.



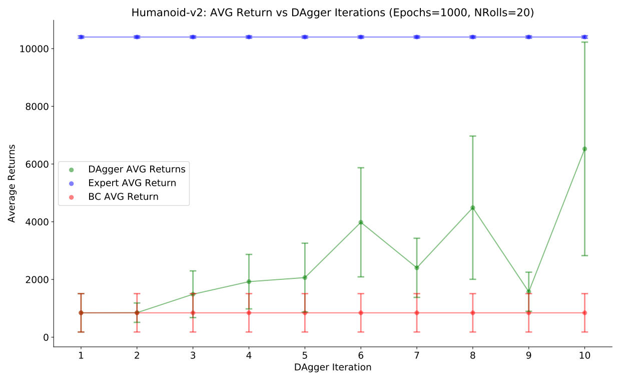
(a) Ant-v2 DAgger



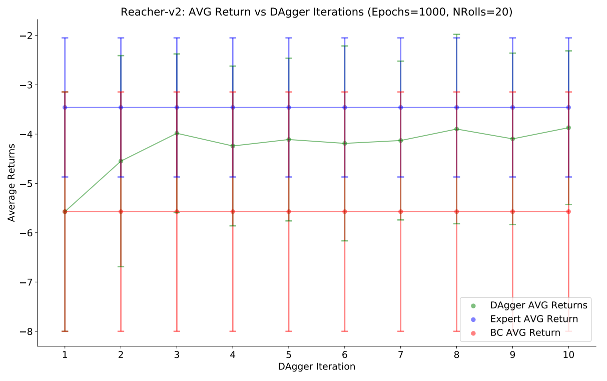
(b) Cheetah-v2 DAgger



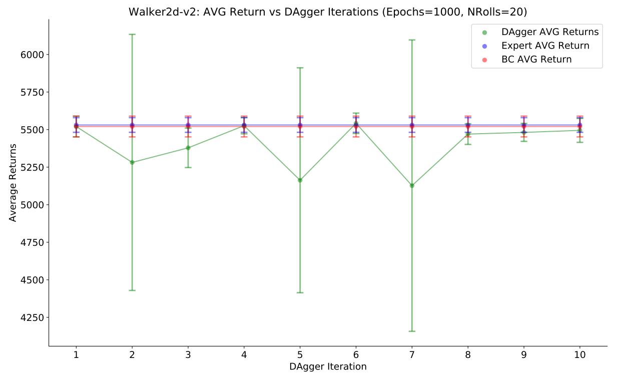
(c) Hopper-v2 DAgger



(d) Humanoid-v2 DAgger



(e) Reacher-v2 DAgger



(f) Walker-v2 DAgger

Figure 6: Comparison plots for the DAgger algorithm with respect to the BC model and the Expert policy average reward performance for different number of iterations of the DAgger algorithm (1000 training epochs, 1000 time-steps, same NN/policy model 64x32 layers). The standard deviation of the results is represented by vertical lines.