EFFECTS OF PRE-TRAINING IN NON-NOISY ENVIRONMENT FOR BEHAVIOR CLONING USING FETCH ROBOT PICK AND PLACE TASK

Batuhan Altundas, Yuuna Hoshi, Reagan Kan and Marcus McGuire
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
(baltundas3, yhoshi3, rkan3, mmcguire31)@gatech.edu

April 30, 2021

ABSTRACT

It is safer, more efficient and cheaper to train robots in simulations before deploying them into a real world environments. In most cases in industrial applications, Reinforcement Learning Algorithms are used in simulation because roll-outs in simulation are cheap, while Learning from Demonstration Algorithms are used in real-world because roll-outs in real-world are expensive. This method however does not allow for non-expert users to train robots to execute custom tasks, limiting accessibility to robots. In this project, we examine the potential use of Behavior Cloning on a simulated environment as pre-training a model before training in a more complex simulated environment again using Behavior Cloning, showing a reduction of the training time and improvement on performance in a complex simulated environment after pre-training in a simple simulated environment with the purpose of applying the same principles of pre-training to deploying robots Real World Environments.

***K*eywords** Pre-training · Behavior Cloning · Learning from Demonstration

1 Introduction

A fundamental challenge in the field of robotics is the access to physical robots. It is often much easier and cheaper to get access to Simulation Environments and testing new applications in virtual environments before calibrating the trained tasks for Real World Environments. The Simulated Environment is often simplified version of the Real World Environment, lacking the noise generated from a more complex world space along with often incalculable minor deviations from the wear and tear on the robot to manufacturing errors that provide error between what is and what is expected.

Current Approach to training robots is to use Reinforcement Learning in the simulation environment, followed by an Learning from Demonstration(LfD) model in the real environment. This is done because it is cheap to train in the Simulated Environment, allowing for Reinforcement Learning to leverage a greater number of trials to reach an optimum policy. The produced policy is than calibrated for Real World through an LfD Algorithm. However, this method requires access to an expert to train the Reinforcement Learning Policy in the Simulated World, making it hard for non-experts to safely add new tasks to the Robots, limiting wide range accessibility of robots that would lead to a rapid growth of the field.

While humans can also learn through experience in a method similar to Reinforcement Learning, they are far more effective in learning how to perform tasks via imitation: they observe others perform a task, and then very quickly infer the appropriate actions to take based on their observations [1]. Learning from Demonstration Algorithms are derived from the way humans learn by observing actions of others. Compared to Reinforcement Learning, LfD Algorithms are easier to use by a broader range of expertise levels. Therefore, using an LfD Algorithm in both stages of simulation and real world would allow any user to train complex tasks in a safe, inexpensive and efficient way. Having access to a

simulation environment to speed up training also increases the opportunities to train new tasks for the robots compared to simply using real world to train them from scratch.

Behavior Cloning (BC) is a simple LfD Algorithm capable of operate both in the absence of demonstrator action information and while requiring no or very few post-demonstration environment interactions [1]. While Behavior Cloning is capable of imitating the demonstrator with little to no additional interactions after providing the demonstration, this also leaves BC to be vulnerable to any suboptimalities in the demonstrations. The need for few post-demonstration interactions allows Behavior Cloning to be used in simple yet repetitive tasks while also being highly dependent on demonstrations has influenced us in choosing Behavior Cloning for our project.

Our project is to explore the effects of pre-training in a non-noisy environment. The algorithm we used is Behavior Cloning, and our task is a Pick and Place task using the Fetch robot.

2 Experiment

We used OpenAI Gym Environment for simulated environments, specifically the FetchPickAndPlace Task. The reason for choosing this specific task was due Fetch Robot is designed for commercial use, therefore accessibility to the robot is greater than any other robot while Pick and Place Tasks were chosen due to being a common task that has sparse reward space that make Reinforcement Learning inefficient. Moreover, OpenAI Fetch Simulation Environment allows for high levels of customizability and documentation needed to created our own more complex environments.

2.1 Environment

Our Experiments were conducted on the Pick and Place Task using the Fetch robot from OpenAI Gym. The Fetch robot uses the MuJoCo Physics Engine to simulate a robot arm with a gripper at the end of the arm, as well a cube that can be manipulated by the gripper.

The environment allows agents to sample an observation at each time step. This observation consists of the location of the midpoint of the gripper in 3D space, the location of the cube in 3D space, the location of the cube relative to the gripper, the distance between each end of the gripper, the rotation of the cube, and the velocities of gripper and box. Each observation also has a section for the achieved goal and the desired goal. In the pick and place task, the achieved goal is the location of the gripper and the desired goal is either the location of the block or the final goal location, depending on the state of the task [3].

When interacting with the simulation, the environment takes in a 4 dimensional action vector as input at each time step. The first three dimensions correspond to the desired 3-dimensional movement in the X,Y,Z axes of the gripper, and the last dimension corresponds to the relative movement the gripper paddles controlling the grasping action. A negative value in the forth dimension represents an action to close the gripper and a positive value represents an action to open the gripper.

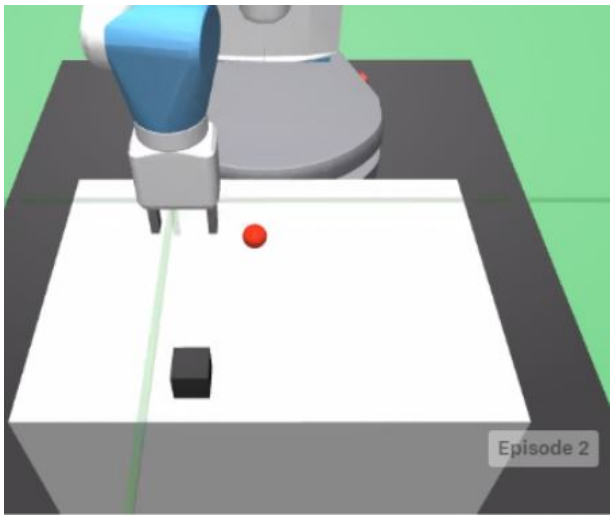


Figure 1: View of the FetchPickAndPlace Simulation Environment [9]

2.2 Task Description and data collection

The FetchPickAndPlace environment randomly places a soft cube and a target during initialization. The cube is always placed on a table in front of the Fetch Robot, and the target is always some location above the table. The Task is to pick up this cube using the gripper located at the end of the Fetch arm, and place it at the target. The nature of this Task allowed us to split it into two smaller tasks that are far more effectively learned through Behavior Cloning: picking up the object, and placing the object at the target.

The tests and training were completed with the initial position of the gripper as constant while the starting position of the object and the target position are initialized randomly between different instances.

Task 1: Picking up the Object This task consists of moving the gripper from the starting position of the gripper to the starting location of the object.

Task 2: Place Object This task is the remaining part of the total task, which is to grip the object, and carry it to the target.

Data Collection In order to collect trajectory data, we combined the OpenAI Gym Environment with a Python library to monitor keyboard input to allow users to control the Fetch robot in the simulated environment with simple keyboard inputs. We collected trajectories from both the simulation environment and in the complex simulation environment.

2.2.1 Interference

Interference layers were added to create a complex, noisy environment that emulates the real world. These interference layers were created to model common interference that happens in a physical robotic environment, but is not often included in simulations. There were three types of interference layers that were created.

- **Gaussian Sensor Noise Layer -** This layer is designed to mimic the imperfections of sensing technology. Real life robots often do not have the ability to perfectly determine their exact location because sensing technology is not precise enough to be noise-free. The Gaussian Sensor Noise layer occurs after the environment is sampled, but before the observation is given to the policy. This layer creates a vector of Gaussian noise with a mean of 0 and a standard deviation of .015 and adds this vector to the observation. This creates an environment observation that that is close to the true observation but off by some small random amount in each dimension.
- **Gaussian Action Noise Layer -** This layer is designed to mimic the imperfection of robotic control technology. Robotic movement is often not consistent and precise enough to be controlled as directly as developers can do in simulations. Instead, there are slight variations to movement that may come from inconsistent power output, mechanical imperfections, or some other source. This is one of two action interference layers that can occur in the complex environment. The action interference layers occur after the policy has determined an action, but before the action is carried out in the environment. Like the Gaussian Sensor Noise Layer, this layer creates a vector of Gaussian noise to serve as interference. However, this layer generates the noise with a mean of 0 and a standard deviation of 0.1. This noise is added to the action vector to create a new, distorted action, that is then sent to the environment to be carried out.
- **Action Speed Noise Layer -** This is the second of the two action interference layers. Like the other action interference layer, this layer takes the output of a policy and creates a distorted version that is used in the next simulation step. This layer also generates Gaussian noise with mean of 0 and standard deviation of 0.1, but this layer only adds the noise to the nonzero dimensions of the action vector. This changes only the speed that the gripper head moves, and thus the distance that is moved in the current simulation time step. This layer is meant to be a counterpart to the Gaussian Action Noise Layer, because that layer creates a distortion that changes the action vector in each dimension. This means that the robot gripper will move drift in directions other than the intended direction. Both types of interference are common in the real world, but the mechanical construction of the robot will determine which happens. The Action Speed type of interference will often happen for robots who have separate motors controlling each dimension's movement, while the Gaussian Action type of interference will often happen for robots that have more complex kinematics.

While other interference methods were discussed, such as limitations to the joint angles of the robot, the MuJoCo Physics Environment handles the Inverse Kinematics functions internally where we were unable to access to modify.

The standard deviation parameters for the Gaussian noise generation were set at the highest value that still allowed demonstrations to reliable complete the task. At lower values, there would be less noise and the original pre-training

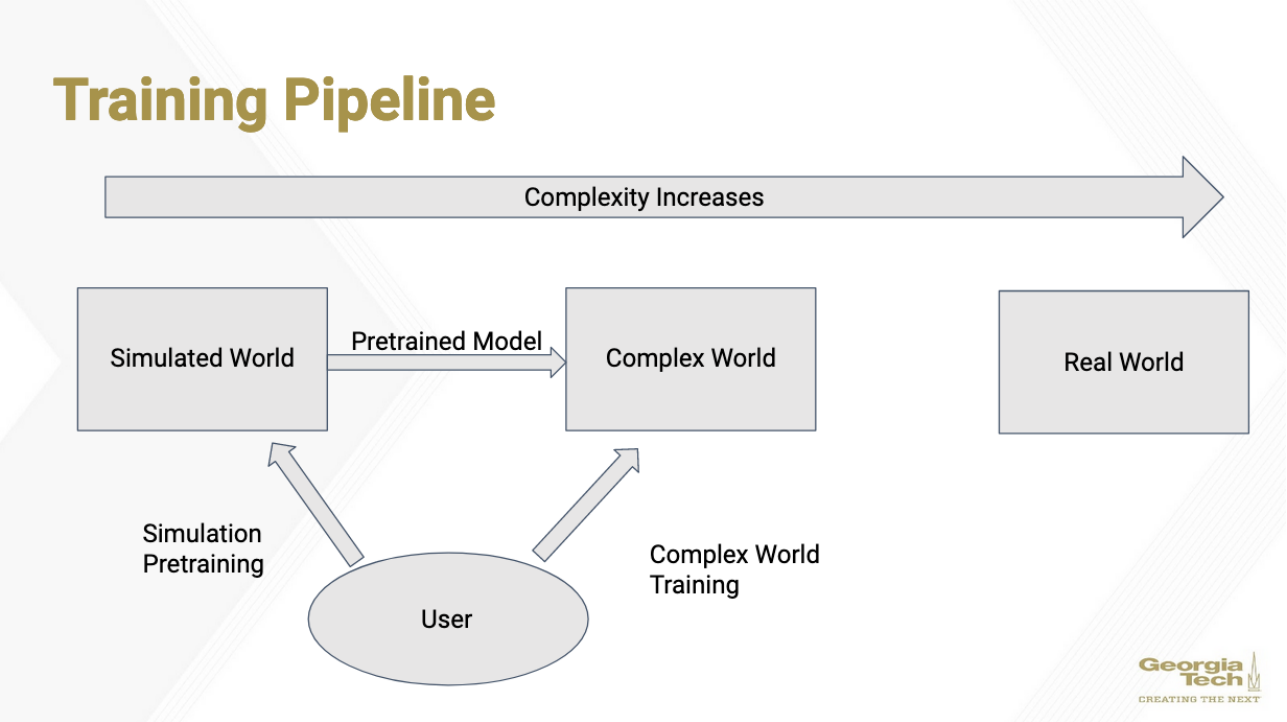
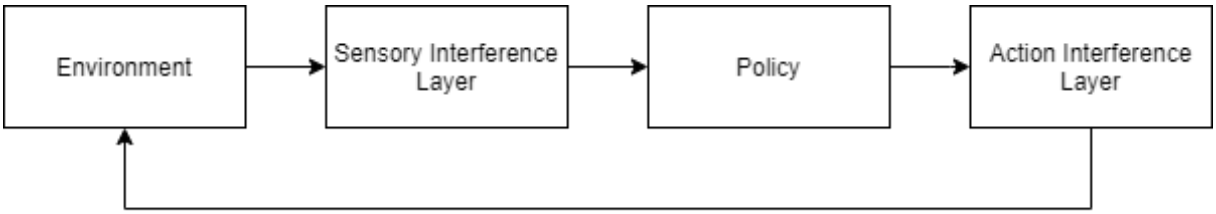


Figure 2: Training Pipeline

would be more effective. At higher values, demonstrators would not be able to complete the task reliably and no behavioral cloning method would lead to a successful policy.



2.3 Policy

As previously mentioned, we use a Behavior Cloning policy, which is represented by a neural network with ReLU activation functions and a single hidden layer of 16 nodes. The policies are trained to minimize Cross Entropy Loss using an Adam optimizer with 0.01 learning rate.

We train a separate policy for Task 1 and Task 2. During inference, the robot follows Policy 1 until the gripper grasps the object when it switches to Policy 2.

2.4 Experiments

We perform 3 experiments, one for each of the interference layers. Before the experiments, we train behavioral cloning policies in the non-noisy environment for tasks 1 and 2 using 50 demonstrations. Because Task 2 is simple and the policy can easily recover from failure, the original task 2 policy works in all experiment setups and is not retrained.

In each experiment, the corresponding interference layer was added to the environment. We collected 50 demonstrations in the noisy environment. Afterwards, we trained 3 behavioral cloning models from scratch with a random initialization using 50, 25, and 10 demonstrations. We also trained another set of 3 behavioral cloning models with 50, 25, and 10 complex environment demonstrations, but those models started off using the weights from the models trained in the simple environment model.

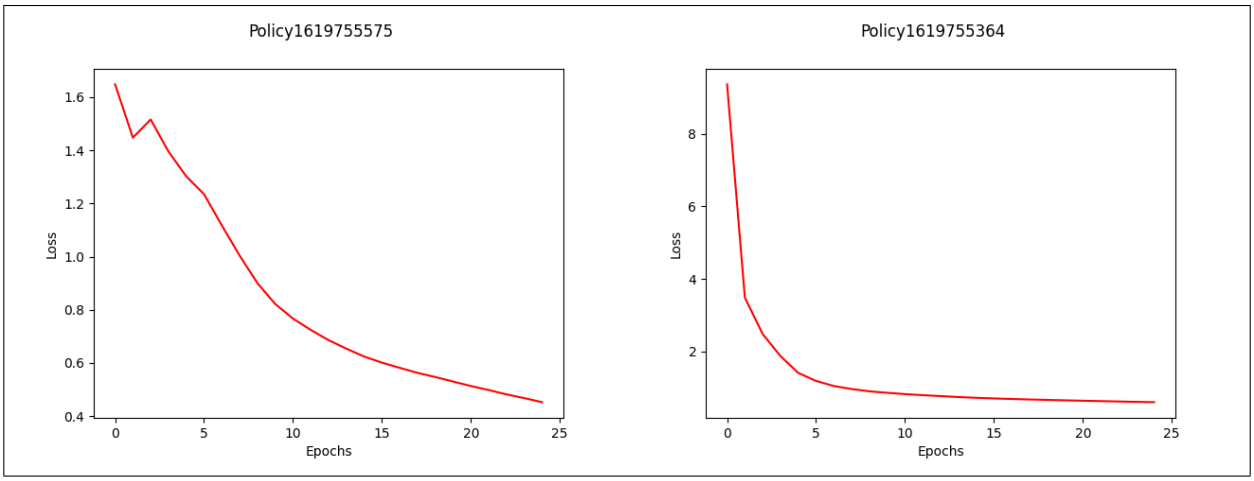


Figure 3: Sensor Noise Interference 10 Demonstrations in the Noisy Sensor Environment.

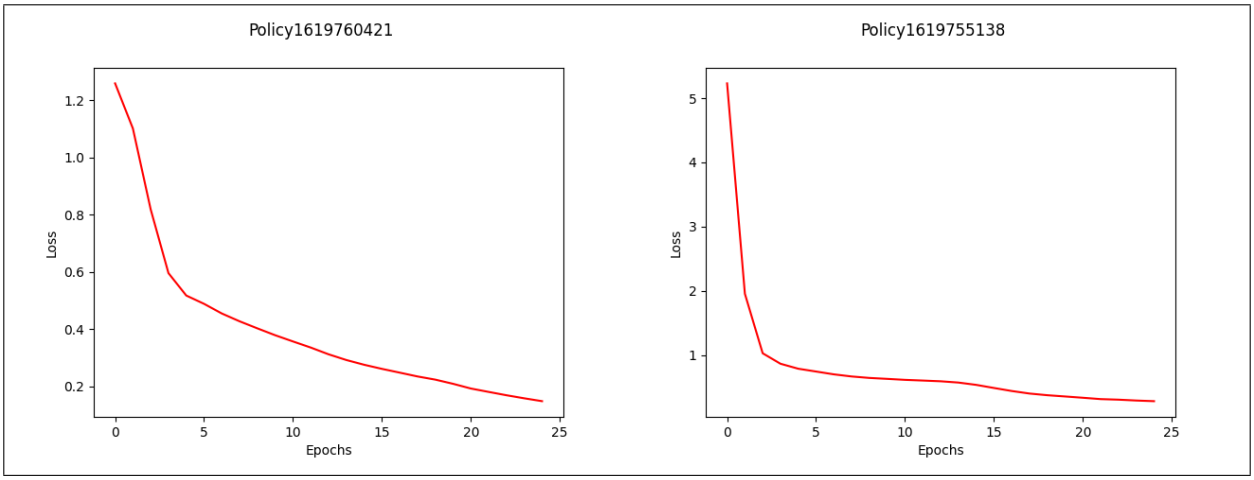


Figure 4: Sensor Noise Interference 25 Demonstrations in the Noisy Sensor Environment. No pretraining (Left), Pretraining (Right).

2.5 Results

Overall, demonstrations in a Simulated Environment are easier to produce than in a Noisy Environment. Training in Simulated Environment followed by training in the Noisy Environment provides better results than training from scratch in the Noisy Environment. Learning is far more sensitive to Sensor Noise than Actuator Noise.

To evaluate our results, we define a successful policy as a policy that is able to complete the task 80 percent of the time. Because of the nature of the task, there are few policies that are able to complete the task between 1% and 80% of the time. Most policies either fail at the task almost every time, or succeed almost every time.

Sensor Noise In the sensor noise experiment, we found that neither the pre-trained nor the from-scratch model were able to produce a successful policy with 10 demonstrations. After 25 and 50 demonstrations, both were able to produce a successful policy. We also find that the pre-trained model converges faster than the from-scratch model, but ends with a higher loss on average than the from-scratch model.

Actuator Noise In the experiment with the Gaussian Action Noise Layer, we found that the pre-trained model was able to produce a successful policy after only 10 demonstrations in the complex environment. The from-scratch model did not produce a successful policy until it was trained with 25 demonstrations. The pre-trained model converges in few epochs than the from-scratch model, on average, and ends with a lower total loss.

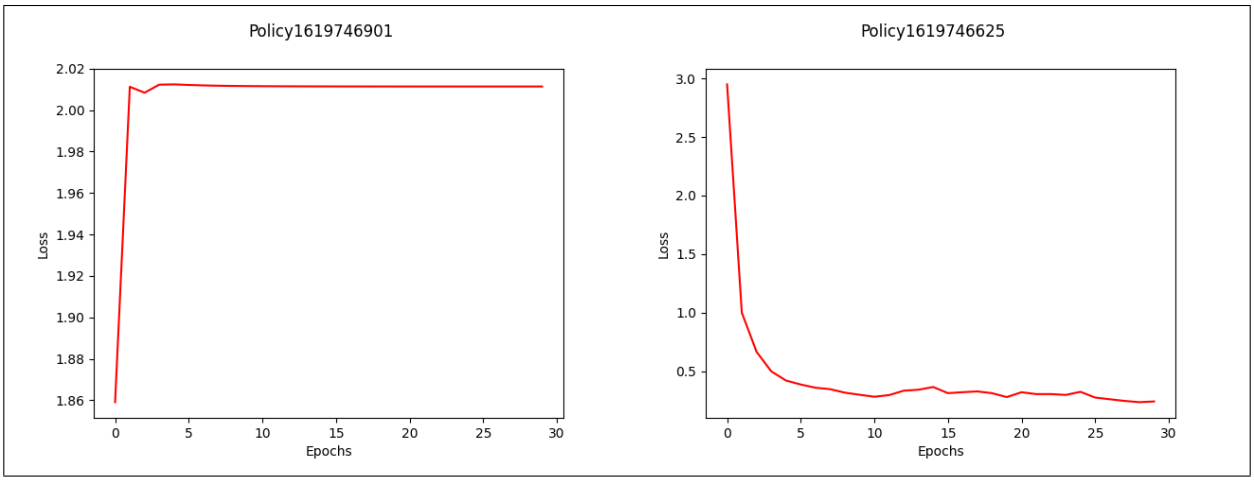


Figure 5: Task 1 Loss Without Pre-training vs With Pre-training

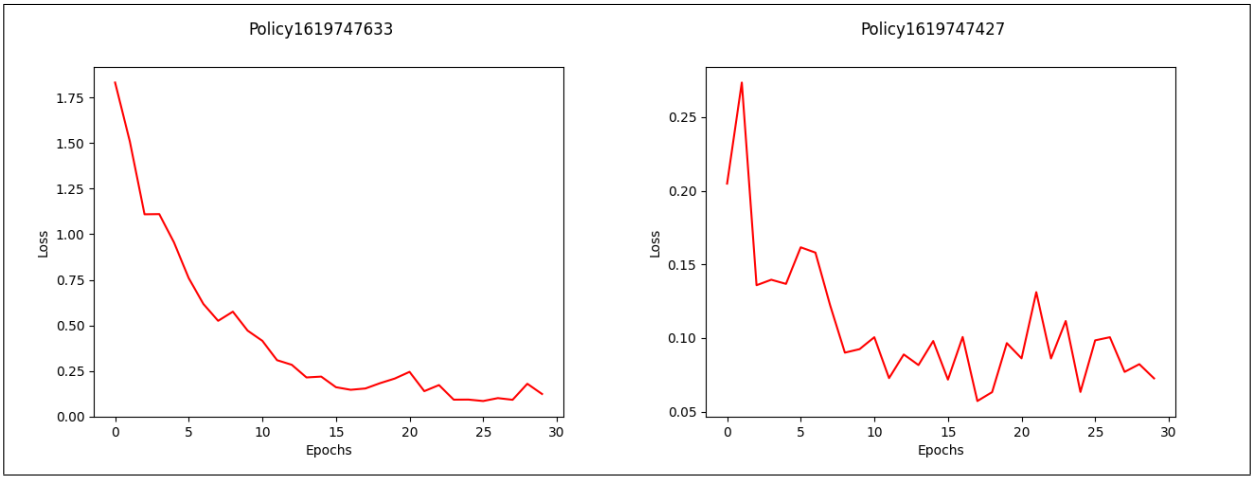


Figure 6: Task 2 Loss Without Pre-training vs With Pre-training

Speed Interference In the Speed Interference experiment, we found that the pre-trained model was able to produce a successful policy after 10 demonstrations in the complex environment, while the from-scratch model required 25 demonstrations. In the expiriment setup, the pre-trained model converged in fewer epochs than the from-scratch model, and ended with a lower total loss.

3 Discussion

According to our results, as see in the graphs in Figures 3, 4, 5, 6, 7 and 8, as well as in Table 1, pre-training in a non-noisy environment produces performance that is significantly improved compared to models generated without

	10 Demos	25 Demos	50 Demos
Pretrained Policy with Gaussian Action Interference	Successful	Successful	Successful
From-Scratch Policy with Gaussian Action Interference	Unsuccessful	Unsuccessful	Successful
Pretrained Policy with Gaussian Sensor Interference	Unsuccessful	Successful	Successful
From-Scratch Policy with Gaussian Sensor Interference	Unsuccessful	Successful	Successful
Pretrained Policy with Action Speed Interference	Successful	Successful	Successful
From-Scratch Policy with Action Speed Interference	Unsuccessful	Successful	Successful

Table 1: Summary of Demonstrations Required to Produce a Successful Policy in Each Experiment.

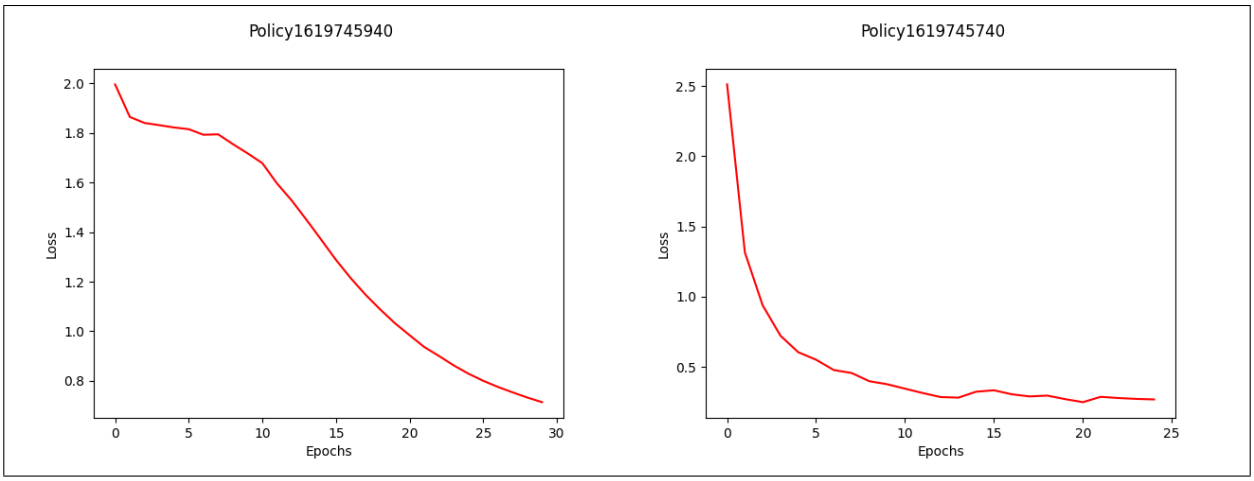


Figure 7: Actuator Noise Interference 10 Demonstrations in the Noisy Sensor Environment. No pretraining (Left), Pretraining (Right).

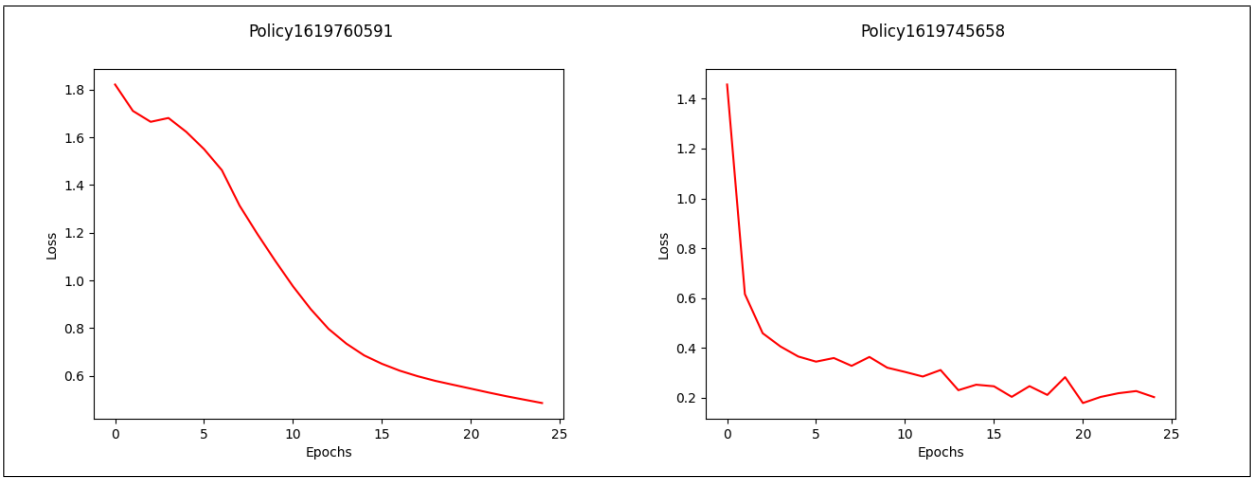


Figure 8: Actuator Noise Interference 25 Demonstrations in the Noisy Sensor Environment. No pretraining (Left), Pretraining (Right).

pre-training. We have observed that performance of pre-trained models that use 10 demonstrations in the complex environment still outperform models that are not pre-trained but are trained with 25 Demonstrations in the complex environment.

3.1 Limitations

While our study examined three distinct types of interference that modeled noise in the real world, all types assumed a Gaussian noise distribution. This is certainly not true for all situations.

It is also important to mention that this project was conducted entirely in the Fetch PickAndPlace environment. The results found in the project may not generalize to all other environments. In particular, the fetch environment is not dependent on time or past actions. For environments where this temporal independence assumption is not true, the results of this project likely do not apply.

3.2 Future Work

While gathering human demonstrations, using keyboard input provided some challenge and had a learning curve, as the user would have to memorize which keys mapped to which movement. For future work, a more intuitive approach such as a 3D mouse may prove to be more efficient.

Due to only having access to the simulated environment and the lack of access to a Fetch Robot to test our work on Real World Environment, following research will involve testing our work on a Fetch Robot in real life. This will likely give more insights on differences between a real life robot and the OpenAI simulation, giving opportunities to improve our noise models.

Another direction to explore is to apply the reverse of the principles explored in this work, by adding a filter to a Noisy Environment to create a Simplified Environment that could then be used for pre-training. A successful noise filter would remove unrelated noise from the Noisy Environment while keeping the important characteristics, and therefore provide a better pre-training environment compared to a complete environment.

4 Conclusion

We identified a potential limitation for the accessibility of robots and the training of robots for new tasks. We showed that in practice, pre-training in a non-noisy simulation environment improves the performance of a new learned task in the noisy environment, requiring less demonstrations in the complex environment to achieve equal and greater performance to a model trained in the complex environment with more demonstrations.

Source Code for this paper can be found in `https://github.gatech.edu/baltundas3/cs7648_project.git`

Acknowledgements

This work is done as part of CS7648 Interactive Robot Learning Course in Georgia Institute of Technology. We would like to thank Matthew Gombolay, Andrew Silva, Letian Chen and Zheyuan Wang for their help.

References

[1] Faraz Torabi and Garrett Warnell and Peter Stone, Behavioral Cloning from Observation, *CoRR, abs/1805.01954*, 2018.

[2] Sun, W., Venkatraman, A. and Gordon, G. J. and Boots, B. and Bagnell, J. A. Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. In International Conference on Machine Learning (pp. 3309-3318), 2017.

[3] Raghav Nagpal and Achyuthan Unni Krishnan and Hanshen Yu, Reward Engineering for Object Pick and Place Training, *arXiv:2001.03792*, 2020.

[4] Marcin Andrychowicz and Filip Wolski and Alex Ray and Jonas Schneider and Rachel Fong and Peter Welinder and Bob McGrew and Josh Tobin and Pieter Abbeel and Wojciech Zaremba, Hindsight Experience Replay, *arXiv:1707.01495*, 2018.

[5] Matthias Plappert and Marcin Andrychowicz and Alex Ray and Bob McGrew and Bowen Baker and Glenn Powell and Jonas Schneider and Josh Tobin and Maciek Chociej and Peter Welinder and Vikash Kumar and Wojciech Zaremba, Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research, *arXiv:1802.09464*, 2018.

[6] Brian D. Ziebart and Andrew Maas and J. Andrew Bagnell and Anind K. Dey, Maximum Entropy Inverse Reinforcement Learning, Proc. AAAI p.1433–1438, 2008

[7] Schrum, M.L. and Gombolay, M.C. When Your Robot Breaks: Active Learning During Plant Failure, IEEE Robotics and Automation Letters, pp.438-445, *doi:10.1109/LRA.2019.2961598*, 2020.

[8] Matthew Alger. Inverse Reinforcement Learning. *doi:10.5281/zenodo.555999*, 2016.

[9] OpenAI, Fetch PickAndPlace, *https://gym.openai.com/envs/FetchPickAndPlace-v0/*, 2020.