Active Robot Imitation Learning with Autoencoders and Imagined Rollouts

Norman Di Palo
normandipalo@gmail.com

Edward Johns
The Robot Learning Lab
Imperial College London

Abstract

In this work, we propose a novel method for demonstration-efficient imitation learning using an interplay between a policy network, a learned dynamics network, and an uncertainty network. We introduce an active learning algorithm that allows the agent to actively select which particular instances of the task it wants to see solved by the expert. We empirically demonstrate how this method increases task-solving performance by a substantial margin. We also show how the agent is able to autonomously predict failure rapidly using this method, improving safety in real-world experiments.

1 Introduction

To teach robots complex tasks, one promising avenue of research is Imitation Learning (IL), and in particular Behavioural Cloning (BC). In BC, an expert collects a series of demonstrations of how to solve a task, and an agent then learns to imitate the expert when facing new instances of the task through supervised learning. This method has been applied successfully to a variety of tasks (Zhang et al. [2018], Bojarski et al. [2016]). Showing how to solve a task can be, in several cases, easier and faster than hand-crafting a complex reward function. Nevertheless, numerous demonstrations can be necessary in order for the agent to generalize successfully to new, unseen scenarios. This can be time-consuming for the expert. Furthermore, if the agent passively receives a series of demonstrations, it's not immediate to understand at train time the state-space regions in which its uncertainty is high, and that can hence bring to a failure, or worse to a series of dangerous actions for the robot and its surroundings.

In this work, we propose a novel method to tackle these aforementioned problems using an interplay between a policy network, a learned dynamics network, and an uncertainty network: we developed and tested an active learning algorithm that allows the agent to actively select which particular instances of the task it wants to see demonstrated by the expert. By computing a degree of familiarity to the proposed instance of the task, the agent can prevent the expert from solving instances that are familiar and hence less useful to add to the training set, thus being more time and sample efficient, but also to focus on instances that are particularly unfamiliar, and are hence more probable to result in a failure. Our method is based on Denoising Autoencoders for uncertainty estimation (Vincent et al. [2010], Arponen et al. [2017]). We show how these networks can be successfully used to compute an approximation of the uncertainty of the agent in various parts of the state space, as also shown in the literature, and how they can predict the probability of success of the robot on a certain instance of the task in an unsupervised way. Using this information, the agent can then decide which demonstrations are more useful to lower the general uncertainty and increase its chances of success at test time. We empirically demonstrate how this active learning method increases performance by a substantial margin. For our experiments, we use two robot manipulation tasks: bringing a cube to a desired position and stacking two cubes on a desired position, both with the Fetch robot. We use

NeurIPS 2019 Workshop on Robot Learning: Control and Interaction in the Real World, Vancouver, Canada

the MuJoCo physics simulation (Todorov et al. [2012]). The main contributions of this work are the following:

- We propose a novel method for active robot learning, using an interplay between a Policy Network, a learned world model, and Denoising Autoencoders.
- We demonstrate the ability of this method to rapidly predict failures at test time, avoiding possibly dangerous situations that could harm the robot and its environment.
- We demonstrate how these predictions can be used to actively select the most uncertain instances of the task, in order to minimize the number of required demonstrations.

2 Related Work

Learning from Demonstration using deep neural networks applied to robots has shown impressive results in recent years. (Zhang et al. [2018], James et al. [2018]). Lynch et al. [2019], Sermanet et al. [2018], Zhang et al. [2018] showed how virtual-reality can be a straight forward and quick way to provide demonstrations to the agent.

Di Palo and Valpola [2018], Boney et al. [2019], Zhu and Laptev [2017], Arponen et al. [2017] demonstrated how (denoising) autoencoders can be used for detecting out-of-distribution inputs and regularizing their effects on neural networks, with Boney et al. [2019] comparing them to ensemble methods.

In the context of active learning, Ross et al. [2011] proposed DAgger, a popular active imitation learning algorithm. Several variants have been proposed over the years (Menda et al. [2018, 2017]), but these methods require an expert to label a set of states visited by the agent's policy, a method that cannot be applied easily to real-world systems with a human expert. Kelly et al. [2019] extends these methods to give control to a human expert without querying them on single states.

3 Architectures and Method

Inspired by previous works on imitation learning, we use a feed-forward neural network to parametrize our policy, taking as input the current state and desired goal, and computing as output the action. The policy network is trained with Behavioral Cloning. To model the uncertainty of the agent, we use Denoising Autoencoders. As a learned dynamics model, we use a feed-forward network that takes as input the current state and action and predicts the one step difference $\Delta\hat{x}_{t+1}$ such that $\hat{x}_{t+1} = x_t + \Delta\hat{x}_{t+1}$ (Nagabandi et al. [2018]).

The Autoencoder is a feed-forward network that takes as input the current state and desired goal, that during training time is corrupted with Gaussian noise, $\tilde{x}$, and is trained to give as output the denoised input, $e = \sum_i (x_i - g_\theta(\tilde{x})_i)^2$, where e is the denoising error that we wish to minimize with gradient descent. The denoising error will be a proxy of the familiarity of the agent with the state and goal at hand: if those were part of the previous training samples, the errors will be lower, while being higher on novel states or goals (Di Palo and Valpola [2018], Boney et al. [2019]). In the following we will also refer to the Autoencoder as the Uncertainty Network.

To compute the uncertainty of a particular state, we use an interplay of the three previously mentioned networks, as we show in Figure 1. From a state s_t, we use the Policy Network and the Dynamics Network to predict the future states that the agent will encounter following its current policy, $s_{t+1}, ..., s_{t+T}$. We use the Autoencoder to compute the aforementioned error on each of those states, and then average the result. This value is used as a proxy for the uncertainty of the current state. We show empirically how using the future uncertainty allows the agent to predict failures more quickly, adding safety to the experiments.

We designed the following experiments to compare the final performance of the agent using Passive Learning and Active Learning, reducing all possible influences of external factors. We sample a set of M instances of the task that will be used as a test set, and will be unseen during training. We sample a set N of instances of the task we wish to solve, providing N demonstrations to the agent in the form of trajectories of states and actions $(s_0, a_0, ..., s_T, a_T)$.

For Passive Learning, we sample N additional instances of the task, providing demonstrations. Finally, we train the Policy Network on these $2N$ trajectories.

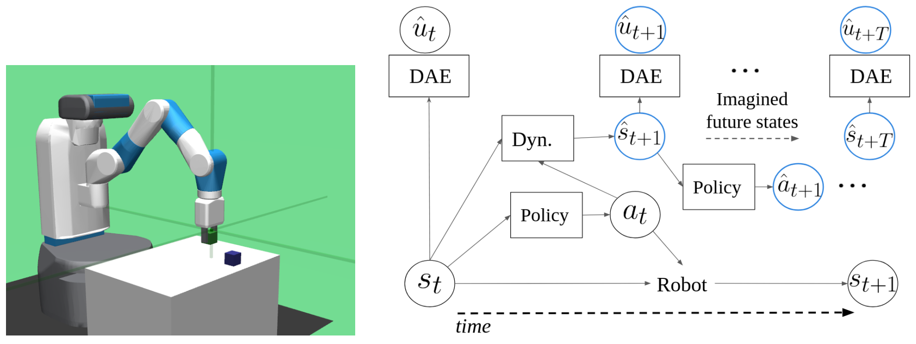

Figure 1: Left: picture of the 2 Cubes task with Fetch Robot. Right: Diagram describing the overall interplay between the networks. The Policy Network and Dynamics Network predict an imagined rollout of T steps. The DAE predicts uncertainty on this rollout, then averages all the outputs $\hat{u}_i$. Blue color denotes values in imagined rollout. A detailed description can be found in the Appendix.

During the Active Learning part we follow these steps: we train the Policy Network, the Dynamics Network and the Uncertainty Network on the N demonstrations. Then, we initialize an instance of the task and let the agent try to solve it and measure its uncertainty online at each step. If the uncertainty becomes greater than a threshold, the agent autonomously stops and asks for a demonstration on how to solve the task starting from where it stopped. This method is useful to let the agent autonomously find areas of the state-space that were unexplored, stopping before a possible failure. The general algorithm follows these steps: we gather m demonstrations before retraining the networks, then repeat until we obtain additional N demonstrations. Hence, in Passive Learning and Active Learning we use the same number of demonstrations, $2N$. We then test the performance of the two Policy Networks on the M test configurations of the task. This method also got very interesting results without using the Dynamics Network, measuring uncertainty only on the current state. Algorithm 1 in the Appendix is a generalization of what described here, were we assumed γ, the ratio of active demonstrations, to be equal to $1/2$ for simplicity.

4 Experiments

In this section we describe the experiments we designed to benchmark the performance of our method on different manipulation tasks[1]. We furthermore empirically measure the ability of the method to predict failure before it happens.

4.1 Active Learning Performance on Test

We evaluate our Active Learning algorithm against the classic passive learning approach, as described on Section 3. The two manipulation tasks are 1 Cube pick-and-place and 2 Cubes stacking. All experiments are repeated several times with different random seeds. For each task, we measure how many test instances are solved by each method, repeat the training and testing process with different seeds, and in Table 1 we show on how many seeds one method outperforms the other. We show in Table 1 the results of the Active Learning (AL) algorithm against the Passive Learning (PL) one. In Autonomous AL, the robot tries to complete the task but stops if the uncertainty grows over a threshold mid-execution, then stops and the expert provides a demonstration from there (as described in Algorithm 1 in the Appendix and in section 3). Autonomous AL has been tested both with and without imagined rollouts, showing similar results. In Figure 2 we show how increasing the ratio of AL demonstrations increases performance, as expected. The greatest impact of the Dynamics Network and imagined rollouts can be seen in section 4.2. We also tried Manual AL, where the expert can move the initial configuration and desired goal until the computed uncertainty surpasses a threshold. We tested this approach only on the 1 Cube Pick-and-Place task, measuring uncertainty on current state only, with no imagined rollouts, surpassing Passive Learning **33 times out of 50**.

[1]Code can be found at `https://sites.google.com/view/robotactivelearning/`.

Task name	**Aut. AL-PL - No rollouts**	**Aut. AL-PL - 5 steps**
Fetch 1-Cube pick and place	**22**-3	**21**-4
Fetch 2-Cubes Stack	**10**-2	**10**-2

Table 1: Experimental results of Active Learning (AL) against Passive Learning (PL). We show how many times AL obtained a better performance on the test set, and how many times PL did. AL outperforms PL in both methods and tasks by a significant margin.

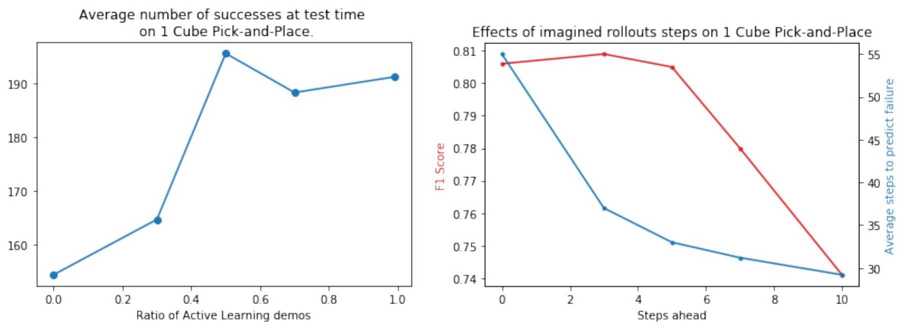

Figure 2: Left: How test-time performance varies as we modify the ratio of Active Learning examples over Passive Learning ones, γ. The plot represents the average number of successes over 13 different experiments. Right: How the averages of failure prediction performance and number of real-world steps needed to predict failures change when modifying the number of look ahead steps in imagined rollouts. Notice how up to 5 steps performance doesn't degrade, while the number of steps is significantly lower.

4.2 Unsupervised Failure Prediction

We experimentally demonstrate the ability of the proposed method to predict failures from the current state and desired goal, trying to minimize the steps needed. We sample N instances of the task and train a Policy Network, Dynamics Network and Uncertainty Network on the collected demonstrations. Then, we sample M test instances of the task. We compute the agent's uncertainty online and predict a failure (and stop the execution) if it surpasses a threshold. We then let the robot complete the task, and record if the task would've been a success or a failure. We finally compute the true/false positives/negatives and the F1 score. We compared the performance on failure prediction of the proposed Denoising Autoencoder with respect to dropout-based computation of uncertainty as proposed in Menda et al. [2018], Kelly et al. [2019]. In Figure 2 we show how changing the number of look-ahead rollout steps using the Dynamics Network affects the failure prediction performance and the real-world steps after which the agent is able to predict failure, demonstrating the potential positive impact on safety of the imagined rollouts of our method. On the 1-Cube Pick-and-Place task, our method achieves **0.80 F1 score** at predicting failures online, on par with the performance of a dropout-based method. On the 2-Cubes Stacking task, our method, without using future rollouts, achieves **0.56 F1 score**, compared to **0.50** of a dropout-based method. Hence, in our experiments the proposed method had comparable or slightly superior performance, measured as F1 score, while being considerably faster given the need for only a single feed-forward pass.

5 Conclusion

We proposed a new method for Active Learning in robotic manipulation tasks, based on a Denoising Autoencoder for uncertainty prediction, and an interplay between policy networks, dynamics models and uncertainty networks. We demonstrated how this method significantly improves task-solving performance. We also demonstrated how the use of imagined future rollouts helps the agent predict failures in fewer steps, hence avoiding coming close to dangerous configurations. These experiments overall showed very interesting results, and as future work we will investigate the method further, extending it to more scenarios, and to real robots.

References

Heikki Arponen, Matti Herranen, and Harri Valpola. On the exact relationship between the denoising function and the data distribution. *arXiv preprint arXiv:1709.02797*, 2017.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Rinu Boney, Norman Di Palo, Mathias Berglund, Alexander Ilin, Juho Kannala, Antti Rasmus, and Harri Valpola. Regularizing trajectory optimization with denoising autoencoders. *arXiv preprint arXiv:1903.11981*, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Norman Di Palo and Harri Valpola. Improving model-based control and active exploration with reconstruction uncertainty optimization. *arXiv preprint arXiv:1812.03955*, 2018.

Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. *arXiv preprint arXiv:1810.03237*, 2018.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *arXiv preprint arXiv:1903.01973*, 2019.

Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Dropoutdagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1709.06166*, 2017.

Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Ensembledagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1807.08364*, 2018.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.

Appendix

Algorithms

In this section of the Appendix we describe in more detail the proposed algorithms. Algorithm 1 presents the steps taken in the Active Learning procedure. We define *unc_rollout* as the function that computes the uncertainty using the Dynamics Network for predicting future states and their predicted uncertainty, describing it in Algorithm 2.

Algorithm 1 Active Learning with Uncertainty Networks - Autonomous Variant

```
Initialize demonstrations set $D = \{\}$, Policy, Dynamics and Uncertainty networks $f_\theta, d_\gamma, g_\phi$,
total desired demos $N$, active learning demos ratio $\gamma$, steps-to-retrain $\mu$.
# Start collecting a fraction of the total desired demos in passive learning to train the networks.
for i in $N(1-\gamma)$ do
    Sample task instance.
    Collect demo in the form $d_i = (s_0, a_0, ..., s_T, a_T)$.
    $D \leftarrow D \cup d_i$
end for
Train networks on $D$.
# Collect demos with Active Learning. Retrain every $\mu$ steps, stop at $N$ total demos.
for i in $\gamma N/\mu$ do
    for j in $\mu$ do
        Sample task instance.
        while Task is not finished/failed do
            Measure uncertainty $u_t = unc\_rollout(s_t, f_\theta, d_\gamma, g_\phi, steps)$, compute next action $a_t =$
            $f_\theta(s_t)$.
            if $u_t > u_{thr}$ then
                Stop execution, collect expert demo from $s_t$ as $d_i = s_t, a_t, ..., s_T, a_T$.
                $D \leftarrow D \cup d_i$
                Break While.
            end if
            Execute action $a_t$, obtain observation $s_{t+1}$.
        end while
    end for
    Train $f_\theta, d_\gamma, g_\phi$ on $D$.
end for
```

Algorithm 2 Function *unc_rollout()*

```
INPUTS: current state $s_i$, Policy Network $f_\theta$, Dynamics Network $d_\gamma$, Uncertainty Network $g_\phi$,
number of future steps $steps$.
Initialize $u_{tot} \leftarrow 0$.
for i in $steps$ do
    Predict uncertainty and add to total. $u_{tot} \leftarrow u_{tot} + g_\phi(s_i)$
    Predict action $a_i \leftarrow f_\theta(s_i)$.
    Predict next state $s_{i+1} \leftarrow s_i + d_\gamma(s_i, a_i)$
    $s_i \leftarrow s_{i+1}$
end for
Return $u_{tot}/steps$
```

Hyperparameters and details of experiments

In this section we describe the hyperparameters and architectures we used in our experiments. We used the OpenAI Gym Fetch environment (Brockman et al. [2016]) for our experiments, that was modified to add the 2 Cubes scenario. The 1 Cube scenario has an observation space of 31, describing positions, velocities and orientations of end-effector and objects and desired goal position, and an action space of 4, describing cartesian velocities and gripper position. The 2 Cubes scenario has an observation space of 49 and same action space.

Table 2: Architectures of Policy Network (PN), Denoising Autoencoder (DAE) and Dynamics Network (DN). We describe the hidden nodes' sizes (HS), number of hidden layers (HL).

Task name	**PN HS**	**PN HL**	**DAE HS**	**DAE HL**	**DN HS**	**DN HL**
1 Cube Pick-and-Place	128	2	8	2	128	4
2 Cubes Stacking	128	2	32	2	128	4

Table 3: Hyperparameters of Algorithm 1 on the various experiments reported in Table 1.

Experiment name	N **demos**	γ	μ	u_{thr}
1 Cube Aut. AL-PL - No rollouts	60	0.5	5	1.5 error on train set
1 Cube Aut. AL-PL - 5 steps rollouts	60	0.5	5	6 error on train set
2 Cubes Aut. AL-PL - No rollouts	300	0.5	25	1.1 error on train set
2 Cubes Aut. AL-PL - 5 steps rollouts	200	0.5	5	1.1 error on train set