Taxicab: Duckietown with Soft Actor-Critic

Julian Freedberg*
Department of Computer Science
University of California, Irvine
Irvine, CA 92617
jfreedbe@uci.edu

Bryon Tjanaka*
Department of Computer Science
University of California, Irvine
Irvine, CA 92617
btjanaka@uci.edu

Carleton Zhao*
Department of Computer Science
University of California, Irvine
Irvine, CA 92617
carletoz@uci.edu

Abstract

We use Soft Actor-Critic (SAC) to train a reinforcement learning agent for the Lane Following challenge of the Artificial Intelligence Driving Olympics (AI-DO). Specifically, we modify the existing implementation of SAC in the Spinning Up library to perform progressively more difficult versions of Lane Following.

We succeed in applying SAC to an agent in a default Duckietown environment, though we achieve subpar results. As such, we additionally experiment with simplifying the environment to a short straight track and augmenting the network with convolutional layers. This new agent demonstrates promising results, occasionally completing the simplified track after only 36,000 training steps. Finally, we modify the reward function and train on a longer straight track but fail to see significant improvement. Future work on this project could focus on completing an agent that reliably completes the long straight track and perhaps even a more complex loop track. It could also involve integrating other methods, such as imitation learning and ensembles.

1 Introduction

Deep Reinforcement Learning (RL) has achieved tremendous success in various domains (1), (2). One area that has proven challenging for deep RL is that of self-driving cars. To address this problem, Paull et al. have created Duckietown, an educational platform where small "duckiebots" travel around a miniature city known as a "duckietown" (3).

We propose to further the reputation of deep RL by exploring its feasibility in Duckietown. Specifically, we aim to examine whether deep RL can succeed in the relatively simple Lane Following (LF) task, where the duckiebot traverses a continuous track in the absence of other duckiebots. We begin by using a standard implementation of Soft Actor-Critic (4) to train an agent in a simulated Duckietown environment. As it turns out, training a duckiebot to succeed in the LF task is far from trivial. Our initial attempts yield an agent that spins in place, caught in a local optimum. Thus, we carry out experiments where we modify the environment, model, and reward function. In particular, we explore simpler, straight-track environments, actor networks with convolutional layers, and reward

*Equal contribution. Order determined alphabetically.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

functions that punish turning. We find that our agent performs best when we maintain the default reward function and use convolutional layers in the actor network. [2] [3]

2 Background

2.1 Reinforcement Learning and Soft Actor-Critic

We adopt the standard MDP formulation, in which the MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, the transition function $p(s'|s,a)$ is the probability of transitioning to state s' when taking action a in state s, the reward function $r(s,a)$ gives the reward for taking action a in state s, and γ is the discount factor. In reinforcement learning, the goal of the agent in this MDP is to find a policy $\pi(a|s)$ that maximizes the discounted return over its entire lifetime.

SAC is a state-of-the-art off-policy reinforcement learning algorithm (4). SAC relies on maximum-entropy reinforcement learning, so it balances expected return with the entropy of the policy by including an entropy regularization term in the Q-function, as shown below:

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{\substack{s'|s,a\sim p \\ a'|s'\sim\pi}} \left[r(s,a) + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \tag{1}$$

Here, H is entropy and α is an entropy regularization coefficient. Meanwhile, the policy seeks to maximize the corresponding value function:

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{a|s\sim\pi} \left[Q^{\pi}(s,a) \right] + \alpha H\left(\pi(\cdot|s)\right) \tag{2}$$

In short, SAC trains two Q-networks and a policy network and achieves excellent sample efficiency.

2.2 Duckietown

Duckietown is a platform for evaluating the effectiveness of deep learning and classical algorithms in autonomous driving tasks (3). The AI-DO is a set of challenges for robots ("duckiebots") in Duckietown. It features several tasks, the simplest of which is Lane Following, or driving on the right-hand side of the road for as long as possible. More advanced tasks involve driving while interacting with other vehicles, as well as with intersections. Gym-Duckietown (5) provides a simulated Duckietown as an OpenAI Gym environment. It is commonly used in the AI-DO.

2.3 Libraries

Spinning Up (6) is a collection of implementations of various reinforcement learning algorithms intended for educational use. As such, the implementations, while not designed for efficiency, are easy to read and modify. Most of the algorithms in Spinning Up are implemented with both TensorFlow and PyTorch. We use the PyTorch implementation of SAC.

3 Problem Statement

We address the problem of training an agent with SAC to complete the LF task in Duckietown. We will experiment with different models, environments, and rewards to accomplish this task.

Our environment is the Gym-Duckietown simulator (5). The action space in Gym-Duckietown consists of two continuous values in the range [-1,1], one for the left wheel and one for the right wheel. Observations from Gym-Duckietown consist of a 3-channel 640x480 image of the world. This image can be provided with fisheye distortion to mimic the real robot camera, but we choose not to do so. Gym-Duckietown provides a variety of built-in city maps. We use both a complex city track and two simpler straight tracks. Finally, it should be noted that Gym-Duckietown provides domain randomization to assist in sim2real transfer, but we disable this feature.

[2] Project Website: `https://taxicab175.github.io`

[3] Code: `https://github.com/btjanaka/taxicab175`

Each episode in our environment terminates after 1000 timesteps, or when the robot leaves the road. During those timesteps, the agent receives reward for following a Bezier curve drawn in the center of the right side of the road. Specifically, reward is calculated according to

$$r(s_t, a_t) = \vec{v}_t \cdot \hat{d} - 10 \min_c |\vec{x_t} - \vec{c}| \tag{3}$$

where $\vec{v}_t$ is the velocity of the robot, $\hat{d}$ is the direction of the Bezier curve, and $\min_c |\vec{x_t} - \vec{c}|$ is the minimum distance from the robot's position x_t to any point c on the curve. In other words, the agent is rewarded for moving in the correct direction, and it is punished for straying from the center of the lane. Note that we assume the action control value is equivalent to the velocity of the robot – in the real world, this is certainly not true. Finally, if the agent goes off the road, it receives -1000 reward.

3.1 Evaluation

We evaluate our success in two ways. Quantitatively, we measure episodic return and look at how it varies as the agent encounters more timesteps. Qualitatively, we look at videos of the robot performing in simulation and see whether it is truly following the lane.

3.2 Milestones

Our milestones are as follows:

1. **Baseline:** Train and evaluate a library implementation of an SAC agent on a complex city in the simulated LF task. This allows us to see how well SAC performs "out-of-the-box." We do not expect the agent to succeed in this milestone.
2. **Simple:** Add convolutional or other layers to the model and train it to complete a short straight track. This simpler task allows us to tune SAC to achieve some form of intelligent motion.
3. **Goal:** Modify the reward signal as needed to create an agent that consistently completes a longer straight track. These modifications are intended to further improve the performance of SAC.
4. **Moonshot:** Train the agent to follow lanes in the complex city, perhaps even on a physical duckiebot.

We discuss our progress on the Baseline, Simple, and Goal milestones in the following sections.

4 Methods

We discuss the algorithm, model, and hyper-parameters for each milestone.

Baseline: We use the PyTorch implementation of SAC found in Spinning Up with almost no modification. This implementation uses an MLP for the policy network and two Q-networks. The input layer to this MLP has 57,600 nodes in order to take in a 3-channel 160x120 image (we downsample from the original 640x480 image). Next, the MLP has 2 hidden fully-connected layers of 128 nodes each, and an output layer of 2 nodes (one output value for each wheel). The networks are optimized with Adam, and the hyper-parameters, including a learning rate of 0.001, are the defaults found in PyTorch [4].

Simple: We begin with the same model and settings as in the Baseline milestone. We augment the policy network by adding 2 convolutional layers before the MLP and changing the observations to single-channel grayscale images. The first convolutional layer has 6 channels and a 5x5 kernel, and the second convolutional layer has 16 channels and another 5x5 kernel. After each convolutional layer, we place a 2x2 max pooling layer with reLU activation. Finally, we reduce the MLP to have only 1 hidden fully-connected layer of 128 nodes. We do not add convolutional layers to our Q-networks, but we do pass them the flattened grayscale images and reduce the number of hidden layers to 1.

[4] `https://pytorch.org/docs/stable/optim.html#torch.optim.Adam`

Goal: We use the same model and settings as in the Simple milestone. Then, we modify the reward function in an attempt to make the agent learn more efficiently (in fewer timesteps). Specifically, we add a "turning penalty" of $-100|v_{\text{left}} - v_{\text{right}}|$, where v_{left} and v_{right} are the control values sent to the left and right wheels (recall that we assume the control value is equivalent to the actual velocity). This term is intended to encourage the agent to go straight. As the difference between v_{left} and v_{right} increases (any difference indicates a turn), the agent is penalized further.

5 Experiments

We describe the experiment associated with each milestone. All training is performed on a single NVIDIA GeForce 940MX GPU. As mentioned previously, default PyTorch hyperparameters are used across all experiments. Every 4,000 steps, we evaluate the policy by rolling it out for 10 episodes and calculating the average non-discounted return across these episodes.

Baseline: We train our Baseline implementation of SAC in the `udem1` map in Gym-Duckietown. This is a relatively complex map, with buildings and intersections in addition to a main track with multiple turns. We train for 200,000 training steps.

Simple: We train our modified implementation of SAC on a straight track that is 4 tiles long. We train for 36,000 timesteps.

Goal: We train the same implementation of SAC as in the Simple experiment but with a modified reward function as described earlier. We use a straight track that is 36 tiles long, and we train for 80,000 timesteps.

6 Results

We describe the results we obtained for each milestone.

Baseline: The agent merely spun in circles.[5] This is not surprising, as this experiment's model treats the input as a flattened vector rather than an image, and the `udem1` environment is very complex. Figure 1a shows the evaluation return in this experiment (refer to Section 5 for how evaluation return is calculated).

Simple: The agent made some progress on the short straight track. In the evaluation video, we see that the agent moves forward a bit before beginning to oscillate between left and right turns. [6] While training, however, the agent was able to make it all the way to the end of the track on some occasions. See Figure 1b for the evaluation return from this experiment. Clearly, the policy has not converged yet, and we believe that if we were to train it for more timesteps, we would see the agent consistently reach the end of the track.

Goal: The agent made no progress on the longer straight track. In fact, it seemed to want to *leave* the track. [7] Looking at the evaluation return in Figure 1c, this behavior makes sense, as the agent has almost no return after the first 10000 timesteps. This behavior may occur because the turning penalty is too high, thus preventing the agent from aligning itself properly before driving straight. It is possible that if we train the agent for more timesteps, it would learn that it needs to overcome this penalty in order to go straight.

6.1 Additional

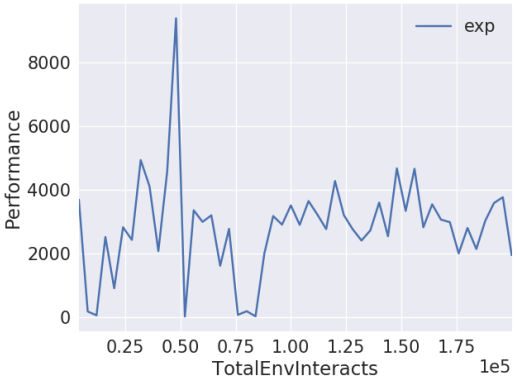
In addition to the experiments related to the milestones, we ran several additional experiments. First, we made the robot simply drive straight at full speed on the long straight track. [8] The return from doing this was approximately 6,800, much higher than the final evaluation returns in any of the experiments (we are somewhat comfortable comparing the numbers directly because the function for calculating evaluation return stays the same across environments).

[5] Video of an evaluation episode for the Baseline experiment: `https://youtu.be/dgNZG6V5d9A`

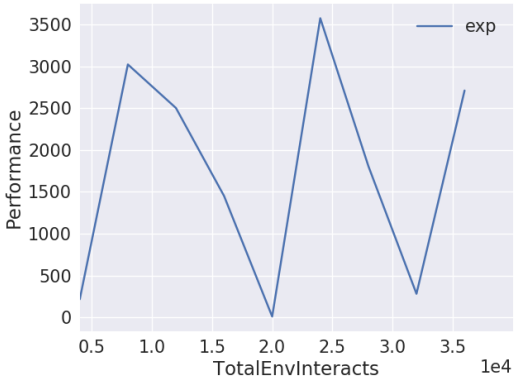
[6] Video of an evaluation episode for the Simple experiment: `https://youtu.be/lu4tbP2URtg`

[7] Video of an evaluation episode for the Goal experiment: `https://youtu.be/ynOGQZgWOKc`

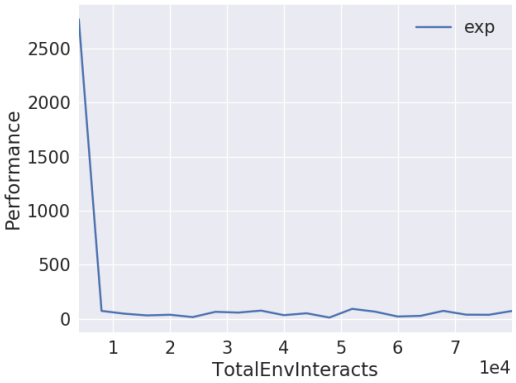
[8] Video of driving forward: `https://youtu.be/idnVgvIC61E`



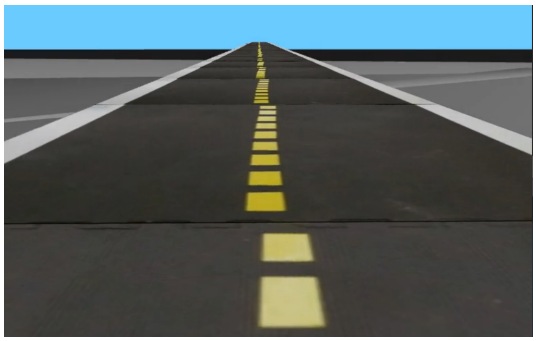
(a) Evaluation return for the Baseline experiment



(b) Evaluation return for the Simple experiment



(c) Evaluation return for the Goal experiment. Note that the values in this plot do not include the additional reward term that we added for this milestone.



(d) A view of the long straight track used in the Goal experiment

Figure 1: Plots and images related to the results.

We also ran the policy from the Simple milestone on the longer straight track and found that it performed surprisingly well, going straight for almost the entire episode episode before stopping and oscillating. [9] The return for doing this was 3,153, which is much higher than the Goal policy that was directly trained on the long straight track. We believe the Simple policy does well on this track because the short and long straight tracks are not very different.

7 Conclusion

We present several attempts to make an SAC agent learn to navigate in the Duckietown environment from nothing but images. We find that the agent we train for our Simple milestone performs best, as it is able to go straight for some distance on the straight track environments. This agent uses convolutional layers in its policy network, as well as the built-in reward function, which rewards following the center of the right lane of the road.

In the future, we would improve several aspects of this project. First, we would use another reinforcement learning library, such as rllib, which supports distributed training. We would then set up infrastructure to train the agent in the cloud, such as via AWS or UCI's HPC. With this training infrastructure established, we would train the current policies further and see whether they can reliably complete courses such as the long straight track and even a circular loop. Meanwhile, on the algorithms side, we would integrate methods such as imitation learning (7) and ensembles (8).

[9] Video of evaluating Simple policy on longer straight track: `https://youtu.be/HeuEx0bq5ls`

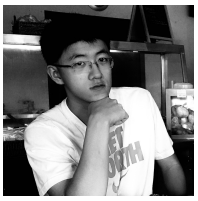
Acknowledgments

This research was performed as part of the Winter 2020 offering of CS 175: Project in Artificial Intelligence taught by Roy Fox at the University of California, Irvine. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the instructor.

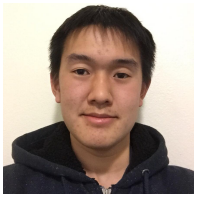
In addition to Professor Fox, the authors would like to thank the members of the Duckietown Slack for providing technical help throughout the course of the project.

Contributions

Julian Freedberg modified the baseline SAC agent to include convolutional layers for better performance and experimented with different reward functions to promote proper driving behavior.



Bryon Tjanaka brainstormed algorithms, set up project infrastructure, trained agents, took meeting notes, maintained the website (`https://taxicab175.github.io`), and wrote the proposal and reports.



Carleton Zhao made sure the inputs and output sizes for the convolutional layers would work with the MLP that the SAC implementation originally came with and wrote parts of the progress report.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017.

[3] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497–1504, May 2017.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 1861–1870, PMLR, 10–15 Jul 2018.

[5] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for openai gym." `https://github.com/duckietown/gym-duckietown`, 2018.

[6] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[7] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," *CoRR*, vol. abs/1802.05313, 2018.

[8] S. Faußer and F. Schwenker, "Neural network ensembles in reinforcement learning," *Neural Processing Letters*, vol. 41, pp. 55–69, Feb 2015.

[9] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," 2018.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] J. Zilly, J. Tani, B. Considine, B. Mehta, A. F. Daniele, M. Diaz, G. Bernasconi, C. Ruch, J. Hakenberg, F. Golemo, A. K. Bowser, M. R. Walter, R. Hristov, S. Mallya, E. Frazzoli, A. Censi, and L. Paull, "The ai driving olympics at neurips 2018," *arXiv preprint arXiv:1903.02503*, 2019.