CS230

Using Generative Adversarial Imitation Learning for Policy Learning in Agent-based Simulation Environments

Kaushik Kasi
SCPD
Stanford University
kausk@stanford.edu

Abstract

This project attempts to explore Generative Adversarial Reinforcement Learning (GAIL) in the context of simulation environments. This project applies GAIL to learn policies for the Lunar Lander OpenAI gym and Humanoid PyBullet environment, and benchmarks GAIL-learned policies against policies learned from traditional reinforcement learning (RL) algorithms. It finds that in the environments and specifications tested, GAIL actually learns a less optimal policy than traditional RL algorithms such as Trust-Region Policy Optimization and Soft Actor-Critic, and requires more training episodes.

1 Introduction

Generative Adversarial Networks (GANs) have been gaining a lot of traction in recent research efforts, due to their flexibility and effectiveness. The most common applications of GANs have been in the visual space for problems such as image generation and visual style transfer. This project aims to leverage generative networks in order to learn policies for agents in simulation environments - a problem typically studied in the reinforcement learning (RL) space. Generative Adversarial Imitation Learning (GAIL), introduced in [Ermon, Ho], bridges the gap between GANs and the policy training work that has been done in reinforcement learning. Simulation environments such as the OpenAI gym and PyBullet allow researchers to quickly prototype and test policy agents against realistic scenarios. In this project, I will try to use a generative network to train agents in the Lunar Lander OpenAI gym, and the Humanoid PyBullet environment.

2 Related work

At a high level, RL is a set of learning algorithms that help an agent within an environment learn an optimal policy to maximize reward [Szepesvari]. RL algorithms are often formulated as trying to learn a policy, π, by optimizing a function $Q^{\pi}(s, a)$ which is the expected reward of performing action a in state s, given by policy π.

One common RL algorithm is Deep Deterministic Policy Gradient [DDPG], which is also characterized as an "actor-critic" model, and has been used successfully for a variety of simulation benchmarks [Lillicrap]. In this model, an "actor" deep neural network outputs an action based on the state. This proposed action and state is used by another "critic" deep neural network to approximate $Q^{\pi}(s, a)$, and the loss is used to iterate on and develop a policy.

3 Background on GAIL

Another approach to policy learning is having an agent attempt to emulate another agent which has already learned an expert policy. Generative Adversarial Reinforcement Learning, (GAIL), introduced in 2016 by Jonathan Ho and Stefano Ermon, provides an algorithm to do this, by formulating policy learning as a generative modelling problem.

At a high level, GAIL initializes a policy and generates (state, action) pairs, τ_i under that policy in a given environment. A discriminator network is trained to differentiate between τ_E generated under an expert policy vs. τ_i from a generated policy, and the loss from the discriminator is used to update the generated policy using Trust Region Policy Optimization (TRPO). The goal is that over many iterations, the policy and discriminator will both improve simultaneously, until the policy has improved so much that the discriminator cannot distinguish between τ_i and τ_E.

This is the algorithm described in [Ermon, Ho], reproduced from their paper:

Algorithm 1: GAIL algorithm

```
Inputs: Expert trajectories $\tau_E \sim \pi_E$, Discriminator parameters $\theta_0, \omega_0$;
for $i = 0, 1, 2...$ do
    Sample $\tau_E \sim \pi_{\theta_i}$ from the generator policy;
    Update the discriminator parameters from $\omega_i$ to $\omega_{i+1}$ with the gradient
      $\hat{\mathbb{E}}_{\tau_i}\left[\nabla_w \log\left(D_w(s,a)\right)\right] + \hat{E}_{\tau_E}\left[\nabla_w \log\left(1 - D_w(s,a)\right)\right]$;
    Take a policy step from $\theta_i$ to $\theta_{i+1}$ using the Trust-Region Policy Optimization (TRPO) rule,
      with cost function $log(D_{w_{i+1}}(s,a)$;
    $\hat{\mathbb{E}}_{r_i}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q(s,a)\right] - \lambda \nabla_\theta H\left(\pi_\theta\right)$;
    $Q(\bar{s},\bar{a}) = \hat{\mathrm{E}}_{r_i}\left[\log\left(D_{w_{i+1}}(s,a)\right) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$
end
```

4 Approach

For both the Lunar Lander and PyBullet environments, I will identify a pre-trained agent for the purposes of training a GAIL model. I will then generate trajectory data τ_E from the experts, and then use GAIL models to learn policies for the agents in these environments.

5 Dataset and Features

The Lunar Lander OpenAI gym (LunarLanderContinuous-v2) [Brockman et. al.] is a Reinforcement Learning simulation environment with continuous state space and discrete action space. The state space is an 8-dimensional vector describing velocities and position of a rocket, and the possible actions are push left, push right, push up, and do nothing. Expert data was generated by loading an policy for this environment that was pre-trained using the Soft Actor-Critic algorithm [Haarnoja] from the RL Baselines Zoo [Raffin]. This policy was deployed within the environment and used to generate 10000 trajectories τ_E from the Lunar Lander environment.

The Humanoid simulation (HumanoidPyBulletEnv-v0) is a PyBullet Gym [Ellenberger] that simulates policies on moving the joints of a humanoid agent that is trying to walk. . It is a complex environment to learn with 376-dimensional continuous observation space and a 17-dimensional continuous action space. 10000 expert trajectories were generated using a pre-trained Humanoid model supplied by OpenAI's Roboschool [Brockman et. al.], made avaiable by PyBullet Gym [Ellenberger].

6 Experiments

The rocket agent was trained using the GAIL algorithm within the Lunar Lander environment using the pregenerated data. The training was done for 1 million episodes - this figure was suggested based on experiments done on this environment by previous literature [Li et. al., Ermon, Ho]. For benchmarking purposes, a Trust-Region Policy Optimization (TRPO) policy and a Soft Actor-Critic (SAC) policy were also trained on these environments.

The humanoid agent with the HumanoidPyBulletEnv-v0 environment was trained for 8 million episodes. Previous literature [Ermon, Ho] suggests that 75 million episodes is necessary for this agent to learn a full working policy for this environment, however this wasn't very feasible given time and computational constraints.

The Stable Baselines library [Hill et. al.] which I used to run the GAIL model advised that using the default hyperparameter values should be suitable for most RL problems, therefore I didn't pursue a hyperparameter search. Given more time, I would use a hyperparameter optimization framework such as Optuna [Akiba et. al.] to find the most optimal values.

Hyperparameters used (select values posted here):

Policy	Hidden Layer Size	Episodes per Epoch	GAE factor
MlpPolicy	100	4096	0.97

7 Results

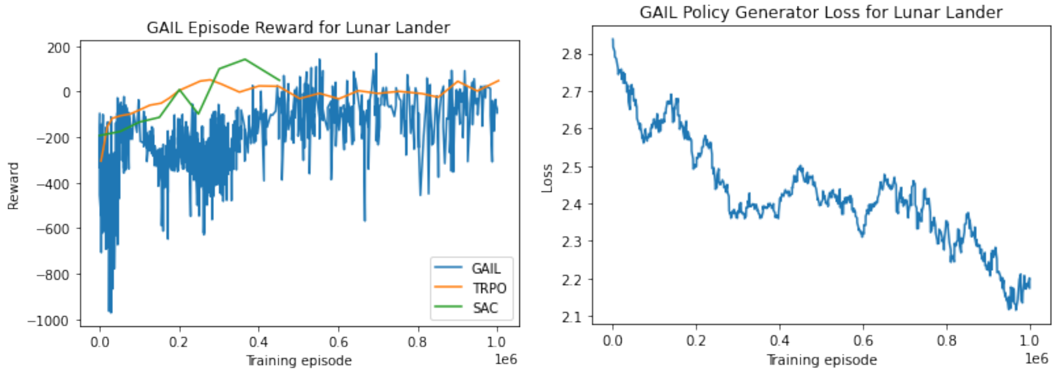

Figure 1: (a) Average reward of policy at each timestep of training (b) Loss of generated policy from discriminator

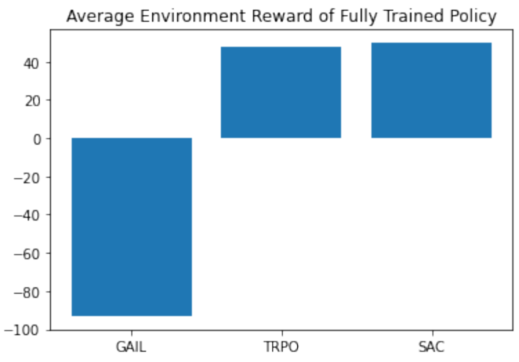

Figure 2: (a) Average reward of final policy taken over 40 simulation episodes

8 Discussion

During the experiment for Lunar Lander, 3 metrics were observed: the learning curve, the generated policy's loss against the discriminator, and the final reward yielded by the trained policy. As evidenced

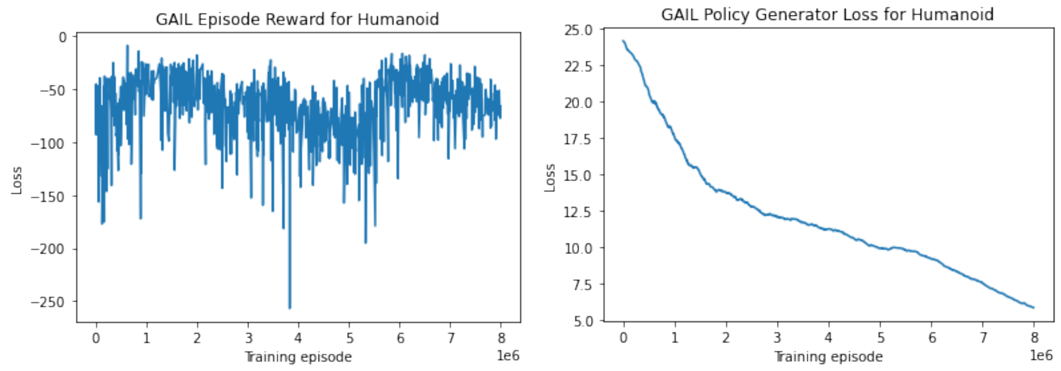

Figure 3: (a) Average reward of policy at each timestep of training (b) Loss of generated policy from discriminator

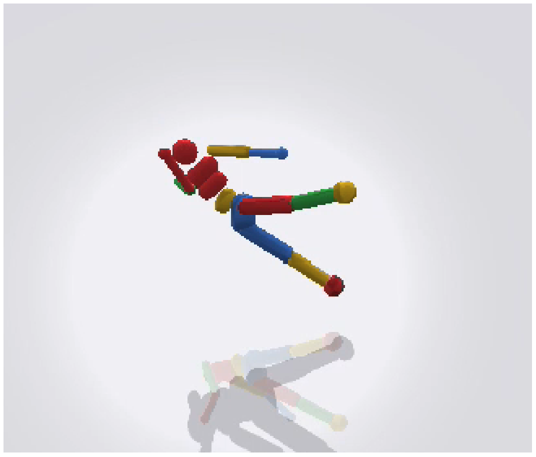

Figure 4: (a) Screenshot of humanoid from PyBullet

by the learning curve in Figure 1(a), GAIL generally required more training to learn optimal policies compared to TRPO and SAC. However, it is corroborated by the increasing reward and decreasing discriminator loss that policy iteration is indeed occurring by GAIL (Figures 1(a), 1(b)). The final GAIL policy at the end of 1 million training episodes yielded an average reward of -93, compared to the average reward of SAC and TRPO which were in the 48-50 range (Figure 2(a)).
Of note - the reward of the SAC policy peaked around 400k training episodes and actually declines after that (SAC was only trained for 480k iterations). Similarly, TRPO training saw steady improvements until 250k training episodes and only moderate improvements afterwards. This may be a sign of the algorithm overfitting during training.
Preliminary metrics were collected for the humanoid agent, which was trained for 8 million episodes. Though there wasn't a noticeable improvement in the agent's policy in this timeframe (Figure 3(a)), the discriminator loss decreases over time for this agent, which indicates that policy learning is occuring (Figure 3(b)). Qualitatively, observing the video of the humanoid shows that the agent has learned a "jumping" behavior, but is not quite able to walk yet. The Github link for this project will include videos of this.

9 Conclusion/Future Work

While this project was not able to show that GAIL is superior to existing policy-learning algorithms, it opens up interesting follow-up inquiries. Given more time, human-hours, and computational resources, I would focus improving this project along these criteria:

1. Exploring and tuning hyperparameters according to best practices in reinforcement learning.

2. Training the GAIL agents for more iterations to see when they eventually learn optimal policies comparable to trained experts.
3. Training agents in more dynamic environments, such as Humanoid Flag Runner, which require more complex policy learning.
4. Performing more robust error analysis on the learned models.
5. Visualizing these simulations in more interesting ways.

10 Contributions

Apart from the gracious help of the TAs and cited reference materials, I was the only contributor to this project. This work was only used for CS230 and not for any other class or purpose. I worked on everything mentioned here, including identifying the problem, setting up the simulation environments, generating the data, training the models, and surfacing/interpreting the results.

11 Code and Supplemental Videos

`https://github.com/kausk/cs230-gail-finalproject`

References

Benjamin Ellenberger. (2018–2019). PyBullet Gymperium. https://github.com/benelot/pybullet-gym.

Deep Deterministic Policy Gradient¶. (n.d.). Retrieved November 17, 2020, from https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba. (2016). OpenAI Gym.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.. (2018). Stable Baselines. https://github.com/hill-a/stable-baselines.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel. (2017). Trust Region Policy Optimization.

Jonathan Ho, Stefano Ermon. (2016). Generative Adversarial Imitation Learning.

Mart Abadi and Ashish Agarwal and Paul Barham and Eugene Brevdo and Zhifeng Chen and Craig Citro and Greg S. Corrado and Andy Davis and Jeffrey Dean and Matthieu Devin and Sanjay Ghemawat and Ian Goodfellow and Andrew Harp and Geoffrey Irving and Michael Isard and Yangqing Jia and Rafal Jozefowicz and Lukasz Kaiser and Manjunath Kudlur and Josh Levenberg and Dandelion Mané and Rajat Monga and Sherry Moore and Derek Murray and Chris Olah and Mike Schuster and Jonathon Shlens and Benoit Steiner and Ilya Sutskever and Kunal Talwar and Paul Tucker and Vincent Vanhoucke and Vijay Vasudevan and Fernanda Viégas and Oriol Vinyals and Pete Warden and Martin Wattenberg and Martin Wicke and Yuan Yu and Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.

Oliphant, T. (2006). A guide to NumPy. (Vol. 1) Trelgol Publishing USA.

Paul Barde, Julien Roy, Wonseok Jeon, Joelle Pineau, Christopher Pal, Derek Nowrouzezahrai. (2020). Adversarial Soft Advantage Fitting: Imitation Learning without Policy Optimization.

Peng, X., Abbeel, P., Levine, S., Panne, M. (2018). DeepMimicACM Transactions on Graphics, 37(4), 1–14.

Raffin, A.. (2018). RL Baselines Zoo. https://github.com/araffin/rl-baselines-zoo.

Szepesvari, C. (2010). Algorithms for Reinforcement Learning. Morgan and Claypool Publishers.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. (2019). Continuous control with deep reinforcement learning.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.

Yunzhu Li, Jiaming Song, Stefano Ermon. (2017). InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations.