CS182 Project Report: Intrinsic Curiosity Module

Matthew Finlayson[*] `matthewfinlayson`
Nicolas Lepore `nicolaslepore`
Andrea Zhang `andreazhang`

December 7, 2020

1 Introduction

Much of reinforcement learning is dependent on defining extrinsic reward functions for different states and actions, which can often be arbitrary or difficult to define. [Burda et al., 2019] recently showed that it was possible to get great results with the introduction of an “intrinsic” reward function based on the prediction error of the learning agent. As an agent gets higher rewards for seeing a different state than predicted, it is motivated to explore new environments. We aim to compare reinforcement learning algorithms with intrinsic and extrinsic reward functions on a number of tasks including raw performance and generalizability while playing MountainCar.

Our code is based off of Seungeun Rho’s basic implementation of RL algorithms which can be found at `https://github.com/seungeunrho/minimalRL`. Specifically, we compared a PPO (Proximal Policy Optimization) algorithm to our ICM (Intrinsic Curiosity Module) implementation from scratch.

2 Background and Related Work

The goal of our empirical investigation was to better understand how intrinsic curiosity can be used to improve results by incorporating it into a PPO agent. Our implementation followed closely the architecture specified in [Pathak et al., 2017] and incorporates it in to the minimal PPO implementation by Rho.

In the context of this course, we saw this technique as a generalization of Markov decision processes where the reward function cannot be directly observed. Instead, we supplemented extrinsic reward with intrinsic reward to attempt to approximate a new reward function by incentivizing surprising results. This strategy is interesting because it follows a pattern seen in nature, where intelligent animals seek out new experiences out of curiosity as part of their strategy. This can have benefits as well as problems. For instance, people are easily distracted

[*]All authors contributed equally, listed in alphabetical order by name.

by screens which capture their attention through novel images and sensory input. Similarly [Burda et al., 2019] found that agents were easily distracted when screens flashing random images were placed in their environment.

3 Problem Specification

We wanted to train an agent to the play the game MountainCar using both PPO (Proximal Policy Optimization) and ICM (Intrinsic Curiosity Module) to compare the two performances.

Note: the choice of MountainCarContinuous is unimportant, since this algorithmic framework can apply to many tasks and environments!

4 Approach

We opted to focus on the AI algorithm rather than the environment. Though there are definitely ways to fine-tune our model to perform better on MountainCar, we decided to prioritize generalizability, as that was a large advantage of ICM.

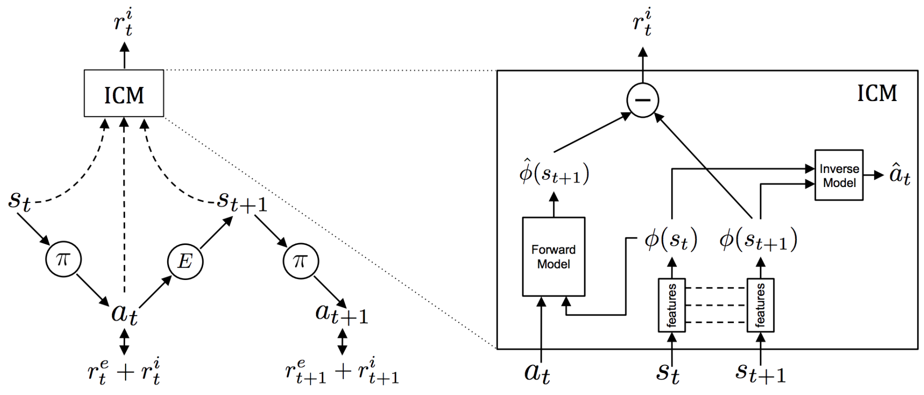
First, we built upon Rho's simple PyTorch PPO implementation and got it working. PPO balances simple implementation, sample complexity, and ease of tuning. At every step, it computes the update that will minimize the cost function while keeping deviation from the previous policy relatively small. We used this formula from [Schulman et al., 2017] to calculate each step:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\ ]$$

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

The ICM has a number of components. At each timestep, given the transition s, a, s', the module calculates $\phi(s)$ and $\phi(s')$ where ϕ is the "feature" encoding. This encoding is meant to capture only parts of the state relevant to the actions taken by the agent. Next, the inverse model g is used to predict $\hat{a} = g(\phi(s), \phi(s'))$. The loss $L_I(\hat{a}, a) = \hat{a} - a$ is minimized by training the parameters of both the inverse model and the feature encoding, so that feature encoding will learn only encodings relevant to the agent's action. Next, the forward model f is used to predict $\hat{\phi}(s') = f(\phi(s), a)$, the next state, and the loss $L_F(\phi(s'), \hat{\phi}(s')) =$

$\frac{1}{2}|\phi(s') - \hat{\phi}(s')|_2^2$ is minimized by training the forward model to predict the next state. The L_F loss is also used as part of the reward for the agent, which tries to maximize it by adjusting the policy π to generate states that the agent has yet to see.



As suggested by the literature, rather than opting for a purely intrinsic reward function, we used intrinsic curiosity reward with an external reward. This is because pure intrinsic rewards can be very useful for level progression games, such as Mario, where the environment changing unpredictably likely indicates success. However, for games like MountainCar, there are issues. For instance, the agent would not know that using an action such as acceleration has a reward cost, and thus it would not factor into its optimization. Further, the agent would not be rewarded for reaching the flag, besides seeing a restart of the game, which does not provide enough incentive. Thus, ICM can be far more useful in this case as a supplement to the external reward, rather than as the sole reward function.

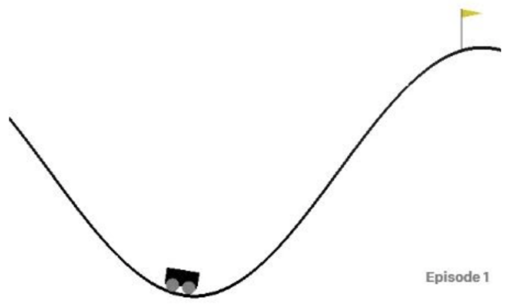
5 Experiments

To build our intrinsic curiosity module, we had to make some design choices about the architectures of each model. For simplicity we used feed-forward neural networks with two hidden layers and ReLU activations for the feature, inverse, and forward models. We briefly explored other architectures such as convolutional neural networks as were used in [Pathak et al., 2017], but since our use case was for very small state vectors, feed-forward nets seemed to already contain more than enough expressiveness. Looking back, it may have been worth it to simplify these architectures further by making the networks shallower and feature encodings smaller (we used an 8 unit encoding when 2-4 probably would have been sufficient).

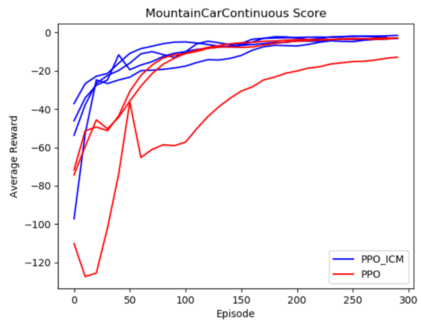
We set up our agent to be able to switch on and off use of the intrinsic curiosity module so that we could compare the two modes. We trained each agent on the MountainCar problem from OpenAI Gym for 300 episodes and plotted the results to compare them. The results are summarized in the plots included in section 5.1.

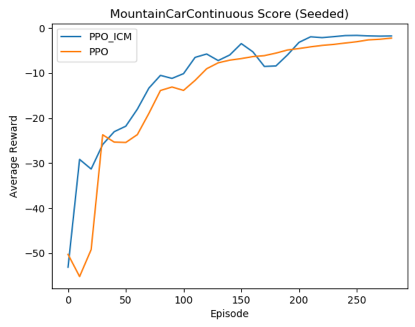
5.1 Results

We used the MountainCarContinuous-v0 environment from the OpenAI Gym with Classic Control. Each state is a [position, velocity] vector, and each action is $a \in [-1.0, 1.0]$. The reward is 100 if the car reaches the flag, and $-a^2 \cdot 0.1$ if it does not reach the flag. There were two local optima for MountainCar: either the agent would learn to not accelerate at all, causing the reward to asymptote at 0, or it would learn to reach the flag, causing the reward to approach approximately 90.

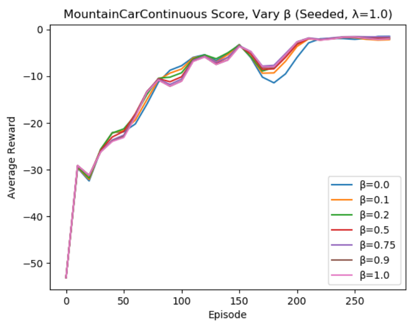


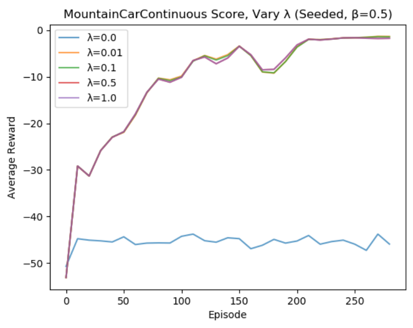
From this first graph, we can see that although the PPO alone and PPO/ICM implementations would approach 0, the PPO/ICM version approaches it more quickly and more consistently.





Then, we experimented with varying parameters. For fixed $\lambda = 1.0$, we varied β from 0.0 to 1.0, where β is the parameter associated with how much inverse and forward loss to include in the optimization problem. A lower β has more inverse loss, while a higher β has more forward loss. We found that a more even split of $\beta = 0.5$ or $\beta = 0.2$ had the best performance, which was consistent with expectations, since we want to incorporate forward and inverse network losses roughly evenly.





We then tried fixing $\beta = 0.5$ while varying λ between 0.0 and 1.0, where λ is the parameter associated with how to scale PPO policy loss, which we derived from previous research. We found that as long as λ was positive, it had a similar effect. This is most likely because adding ICM loss is monotonic in the sense that it keeps approximately the same maxima. However, when $\lambda = 0.0$, only the ICM loss remains, so it continues to reward prediction error. This shows the issue with only using intrinsic reward, as mentioned earlier. The model is always updated with the newest observations, so it is constantly "surprised" as it can never predict its environment.

When hypertuning the PPO with ICM model to MountainCarContinuous-v0 by adding in state normalization, reward normalization, generalized reward estimation, and additional intrinsic reward integration and entropy calculations from tuning suggestions taken from `https://github.com/adik993/ppo-pytorch`, we can solve the game, as it locates the global maximum of reaching the flag rather than staying in place! This is significant, as the global optimum can be found by a fine-tuned version of ICM, whereas it cannot be found by PPO.

6 Discussion

PPO is a versatile and effective algorithm, and can be successfully augmented by other reinforcement learning techniques. In particular, ICM has the potential to be incredibly useful for building general agents that can learn environments they were never explicitly tuned to learn on. As we discover here, however, curiosity is not necessarily well equipped to deal on its own with all environments, and is only a heuristic for approximating reward functions. In problems like MountainCar, where winning and losing is difficult to tell apart simply by looking at the state and no novel experiences necessarily pop up in the form of new levels, ICM can be a helpful addition to give a model an edge, but is not sufficient on its own.

In our experiments using varying amounts of ICM loss, as well as varying how heavily to weight the environmental vs. intrinsic reward, we were able to explore and improve PPO. As seen in other implementations of ICM found online, intrinsic curiosity can make the difference between solving a problem and getting stuck at a local optimum. In future work we would be interested in experimenting more with the architecture of the ICM, such as using convolutional neural nets that look at the render of the environment as is done in some papers working on ICM for environments like Mario. We would also like to explore modifying the extrinsic reward function and environment to see if we could get the agent to become less dependent on the extrinsic reward.

A System Description

Our code can be found at `https://github.com/nlepore33/cs182curiosity`. To use our code and replicate our results, run `python run.py` (assuming you have the basic Gym dependencies correctly installed) to train the agent. The agent's settings will be saved to a file which you can then generate plots for by running `python plots.py`.

B Group Makeup

Our work was split up as follows.

Matt identified the relevant papers on intrinsic curiosity and proximal policy optimization and found Rho's minimal implementation of PPO on GitHub. He designed the intrinsic curiosity module architecture and worked with Nick to integrate it with Rho's minimal proximal policy optimization agent. After the presentation, Matt got the project report into shape for final submission.

Andrea worked hard throughout the project on debugging the code and spearheaded the presentation preparation and execution. She also put together the first draft of the project report.

Nick drove the latter half of the implementation phase, designed the experiments, ran the experiments, aggregated and visualized the results, and worked closely with Andrea to prepare the presentation.

All team members worked closely together and helped each other to make each part of this project work.

References

[Burda et al., 2019] Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2019). Large-scale study of curiosity-driven learning. In *ICLR*.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *ICML*.

[Schulman et al., 2017] Schulman, J., Klimov, O., Wolski, F., Dhariiwal, P., and Radford, A. (2017). Proximal policy optimization. In *OpenAI*.