CS 285 Final Project Report

Brian Zhu, Tejasvi Kothapalli

Abstract

RL algorithms are generally successful in memorizing tasks very well and performing well on tasks very similar to the trained domain. However, oftentimes RL algorithms fail at new tasks. Open AI released the Gym Retro Contest to challenge people to create RL algorithms that respect the “train, test” split. Contestants are allowed to train on a certain set of training levels but final results are measured on unseen levels. The contest games are from the Sonic The Hedgehog™ series.

Previous Contestants and Open AI benchmarks suggested that using PPO or Rainbow as the baseline RL algorithm would be a good starting point. We wanted to build on top of these RL algorithms with meta-learning and exploration strategies. To our knowledge no previous efforts used Recurrent Policies for meta-learning. Furthermore, we wanted to use Random Distillation Networks as our exploration strategy.

We found that using a LSTM-based recurrent policy improved the performance of Joint PPO with a CNN policy during fine tuning test levels, suggesting that meta-learning does help the agent learn faster in unknown but similar environments. In addition, we also found that using an exploration reward bonus through random network distillation improved performance during fine tuning as well, suggesting that improving state coverage is an important factor in efficiently completing levels in the Sonic benchmark. However, we did not observe additional performance benefits by using both recurrent policies and random network distillation at the same time.

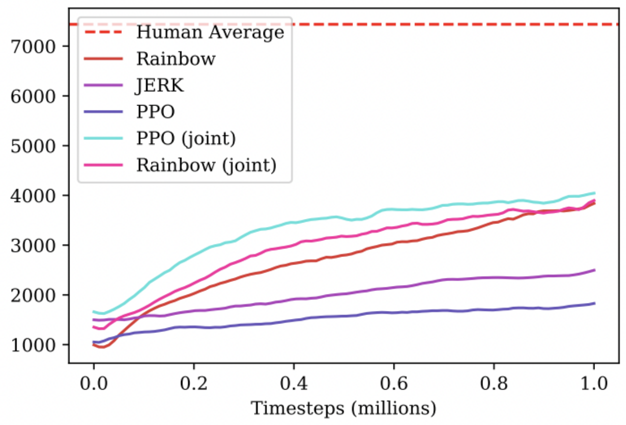
Introduction

Open AI released the Retro Contest [1] which measures a reinforcement learning algorithm's ability to generalize from previous experience. Many reinforcement learning algorithms are able to memorize a certain task very well. However, these same models are not necessarily robust to new tasks. The contest released a set of training levels for the Sonic The Hedgehog™ series and the contest was intended to test the proposed RL algorithms on unseen levels. In this project we approach this contest with two new approaches. We first explore meta-learning with recurrent policies [2]. Second, we explored the use of enhanced exploration with Random Network Distillation [3].

Overview of Sonic Game Contest
The Sonic The Hedgehog™ series is emulated through Open AI's python package Gym Retro. There are three specific Sonic games all with their own set of levels: Sonic The Hedgehog, Sonic The Hedgehog 2, and Sonic 3 & Knuckles. We only experimented with Sonic The Hedgehog. Each game has a set of zones and each zone is further split into acts. Game play can end in three manners: the player successfully finishes the level, player loses a life, or 4500 timesteps have elapsed. The observation used for the model is a 320 by 224 pixel RGB image. The action space is the following: : B, A, MODE, START, UP, DOWN, LEFT, RIGHT, C, Y, X, Z. The reward function is split into two parts. The first part is horizontal displacement from the initial position. The second component is the completion bonus

Previous Work

Retro Baseline Algorithms
Open AI released benchmark algorithms for the contest [6]. The figure below is from the Open AI benchmarks highlighting the reward of various RL algorithms on the test sets. We noticed that PPO joint performed best in the benchmarks so we decided to explore this algorithm as our RL algorithm.



PPO is a policy gradient algorithm. The PPO policy is trained with a joint training algorithm. This is where the policy is concurrently trained on all 188 available training levels. They note that in order to have training converge they had to run hundreds of millions of timesteps. This was simply not possible for us to replicate so we instead chose 8 training set levels and still used joint PPO. Open AI suggests using exploration objectives to improve upon the benchmarks which motivated our use of Random Network Distillation.

Top Contestant Approaches

Next we will discuss successful approaches to the contest [11] and motivate why we wanted to explore meta-learning and exploration strategies for this project. All the top contestants either tuned or modified existing RL algorithms like PPO and Rainbow. In terms of exploration one contestant implemented VAE's to encode the image states [13, 14]. Another contestant tried Noisy Networks for Exploration, #Exploration, and Curiosity-driven Exploration. This contestant noted that their results did not provide conclusive results about the benefits of exploration. Thus we wanted to further explore the prospects of exploration. None of the top contestants tried explicit meta-learning techniques which motivated us to explore Recurrent Policies.

Recurrent Policies

Recurrent policies [2] collect, process, and store transitions over multiple episodes in its hidden state, which is then used to generate a policy. This can be interpreted as using previous transitions to understand the task, and the hidden state that is generated serves as a context to guide the policy. The context in this case could be generated from visual or structural cues in the level to aid the agent's decision-making process.

Random Network Distillation

Random network distillation [3] utilizes the process of distilling a neural network to match another as a way to estimate the density of states visited. In particular, a prediction and target network are initialized with two different random-sampling schemes, and the prediction network is trained to match the output of the target network on next observations. If the prediction network fits to the output of the target network at a specific state well, then it has most likely visited the state. The squared error in prediction can be used as a reward bonus.

Experiment

Overview

The experiment consists of two parts. In the first part, which we will refer to as training, the agent trains on an initial training set of levels. Parameter sharing or other forms of information sharing between levels is permitted and the agent can train for as long as possible. In the second part, which we refer to as fine tuning, the agent continues to train on another set of test levels. The agent's policy and value function are initialized to those from the end of training and sharing information between test levels is not permitted. Agents can only fine tune on each level for 1 million steps. Due to computational restrictions, we reduced the size of the training and test set of levels to the following:

Training Set		
ROM	Zone	Act
Sonic The Hedgehog	SpringYardZone	2
Sonic The Hedgehog	SpringYardZone	3
Sonic The Hedgehog	GreenHillZone	1
Sonic The Hedgehog	GreenHillZone	3
Sonic The Hedgehog	StarLightZone	1
Sonic The Hedgehog	MarbleZone	1
Sonic The Hedgehog	MarbleZone	2
Sonic The Hedgehog	MarbleZone	3

Test Set		
ROM	Zone	Act
Sonic The Hedgehog	SpringYardZone	1
Sonic The Hedgehog	GreenHillZone	2
Sonic The Hedgehog	StarLightZone	3

Table 1: Levels used for training and fine tuning.

The Sonic Gym-Retro Environment

All Sonic levels are loaded using the gym-retro [4] and retro-baselines [5] repositories. The agent takes an action every 4 frames, which will be the length of one timestep. We followed OpenAI's original set of actions as well as the frame-skipping and sticky-action mechanics, please refer to their technical report [6] for more details. Since the goal is to get Sonic from the left side of the map to the right, reward is calculated based on horizontal offset. We use the backtracking mechanic where the agent receives a positive reward if they increase the maximum horizontal progress from the start, and receives 0 reward otherwise. This is to prevent the reward from discouraging the agent from moving backwards, which may be necessary depending on the complexity of the level. Again, please refer to [6] for more information. Reward is normalized to be out of 9000, with an additional 1000 reward bonus for completing the level. The maximum trajectory length is 4500 timesteps.

Algorithm and Policies
We start with a Joint PPO baseline and build off it in two ways: by switching to a recurrent policy (CNN-LSTM) and by using random network distillation [3] to introduce an exploration bonus.

Joint PPO (CNN PPO): An implementation of this can be found in stable-baselines [8], a modern port of OpenAI's original baselines repository [9]. "Joint" refers to the process of running multiple environments in parallel and sharing gradients with a single policy. In our case, PPO [10] uses a single policy to collect rollouts in each level and saves the transitions into a single rollout buffer. The transitions are then sampled from this buffer to run policy gradient.

Joint PPO with recurrent policy (CNN-LSTM PPO): We replaced the CNN based policy with an LSTM on features extracted with a CNN. Using a recurrent policy allows the agent to process previous transitions into an improved policy. In this setting, where we have a training and test set of environments, we can interpret this as the agent meta-learning on the transitions collected during training in hopes for fast fine tuning. The implementation for CNN-LSTM PPO can be found in an experimental fork of stable-baselines [12].

Joint PPO with random network distillation (CNN PPO RND): We use random network distillation [3] as an exploration bonus for Joint PPO to encourage the agent to continue exploring even if the rewards from the environment suggest otherwise. The model used for random network distillation was a CNN feature extractor followed by a two-layer FC network. The weights of the feature extractor are frozen. The original paper suggested creating a separate value head that estimates returns for the exploration bonus (intrinsic reward), and combining the value estimates with the original value net trained on environment rewards (extrinsic reward) when calculating the advantage for policy gradient. We opted to use a simpler method, where we modified the reward by taking a weighted average of the intrinsic and extrinsic reward:

$$r_t = \alpha_i i_t + \alpha_e e_t$$

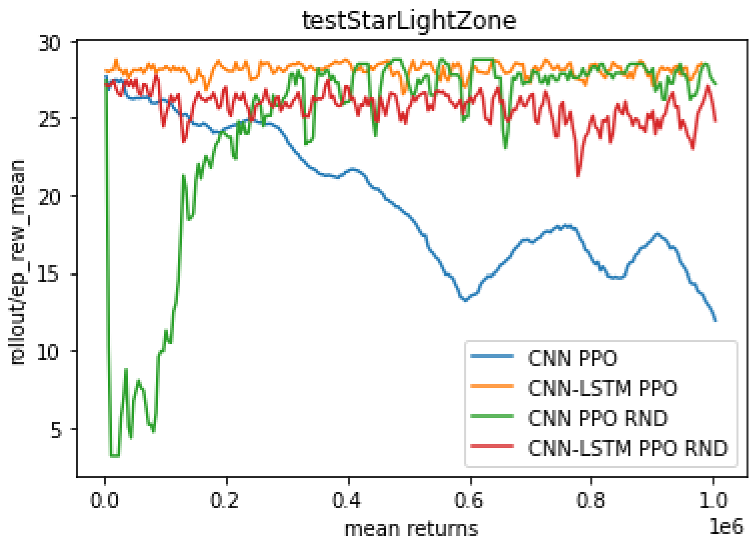
Where both alpha's are predefined. RND hyperparameters are listed in Table 2.

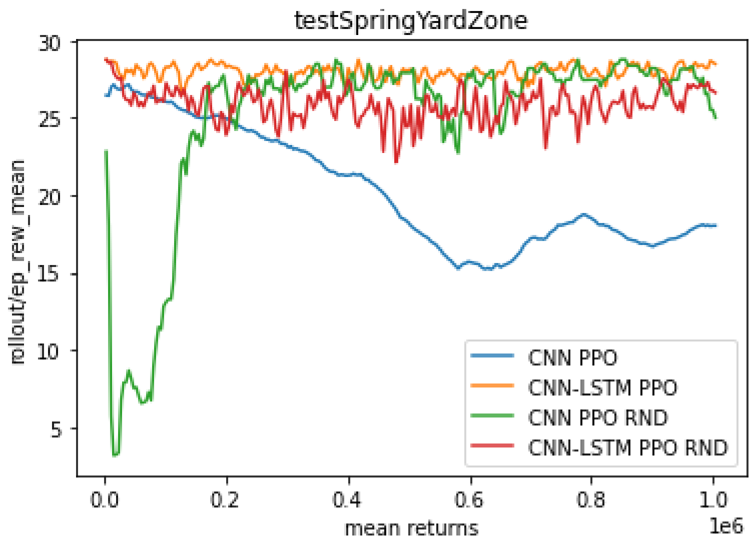
Joint PPO with recurrent policy and RND (CNN-LSTM PPO RND): We finally combined the two changes together to see if there were any additional performance benefits.

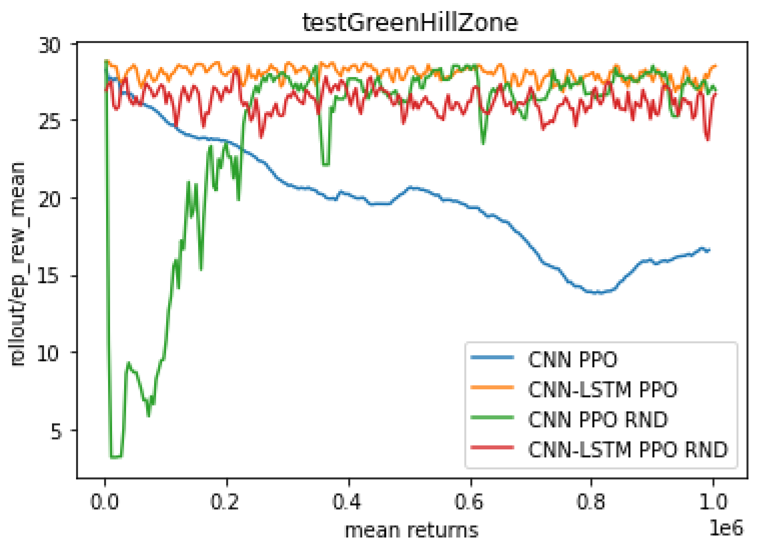
All models are trained with the 8 training levels in parallel over a total of 8 million timesteps.

Results

Note that the figures show a scaled reward. The Sonic environment scales the total reward to be out of 9000, but for the model, the total reward is out of 45.







Discussion

Baseline Results
We see that across all seeds and environments, CNN PPO has poor performance during fine tuning. The policy starts with decent performance but it quickly deteriorates. This could be due to the small training set, where the distribution of level visuals and structure is not diverse enough, and the agent could overfit to the level. However, this does not account for why the agent consistently starts with decent returns, and there needs to be additional investigation into why this behavior occurs

Effect of Using a Recurrent Policy
Across all seeds and environments, CNN-LSTM PPO is able to start with decent returns and continue maintaining decent returns during, which is a significant improvement over the performance of CNN PPO. This shows that using a recurrent policy improves the efficiency of learning, as the agent is able to quickly adjust to the new test environments and achieve similar returns. Perhaps by training on a larger set of levels, a recurrent policy would be able to improve returns over the course of fine tuning.

Effect of Using Random Network Distillation [3]
During fine tuning, CNN PPO RND has a initial dip in performance, but quickly recovers and maintains good returns, which is a significant improvement over the performance of CNN PPO. This suggests that when well-tuned, the exploration signal is beneficial for the policy. This makes sense, as the policy is able to see a larger distribution of states and this robustness carries over when fine tuning.

Effect When Using Both

There does not seem to be additional benefit to using both recurrent policies and random network distillation together when compared to using either one individually. This reflects the same "skill ceiling" seen by other methods, where the average returns does not exceed 30 (about 6000 reward in game), suggesting that there may need to be additional algorithmic improvements to achieve better performance.

Citations

[1] Hesse, Christopher. “Retro Contest.” *OpenAI*, OpenAI, 29 June 2020, https://openai.com/blog/retro-contest/.
[2] Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. RL2: Fast Reinforcement Learning via Slow Reinforcement Learning. 2016.
[3] Burda et al. Exploration by random network distillation. 2018.
[4] https://github.com/openai/retro
[5] https://github.com/openai/retro-baselines
[6] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, & John Schulman. (2018). Gotta Learn Fast: A New Benchmark for Generalization in RL.
[8] https://github.com/DLR-RM/stable-baselines3
[9] https://github.com/openai/baselines
[10] https://arxiv.org/abs/1707.06347
[11] Schulman, John. “Retro Contest: Results.” OpenAI, OpenAI, 2 Sept. 2020, https://openai.com/blog/first-retro-contest-retrospective/
[12] https://github.com/Stable-Baselines-Team/stable-baselines3-contrib
[13] Stadie et al. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models 2015
[14] “World Models Applied to Sonic.” World Models Applied to Sonic | Dylan's Blog, https://dylandjian.github.io/world-models.

Member Contributions

Tejasvi: Gym Retro Setup, Initial Benchmark Running, Evaluation and Visualization.

Brian: Setting up stable-baselines to work with Sonic levels. Setting up forked stable-baselines repo to run recurrent policies. Edited stable-baselines to run random network distillation.