Transfer Learning with Random Network Distillation Theory & Reinforcement Learning

Mustafa Omer Gul - momergul

June 12, 2019

1 Introduction

Deep Reinforcement Learning methods have enjoyed great success on a variety of environments. One reason behind this success is the fact that a majority of the environments these methods have been trained on have dense rewards, allowing the algorithms to quickly learn better policies. When the environment has sparse rewards, however, this success is not replicated. DeepMind's DQN, for instance, achieves 0% of human performance on Montezuma's Revenge, an Atari 2600 game with notoriously sparse rewards (Mnih *et al.*, 2015, p. 3). In situations such as these, we must change the manner in which our agents interact with the environment, such as by using exploration bonuses. One recent success within this domain is OpenAI's Random Network Distillation (RND), a method which has achieved state of the art performance in Montezuma's Revenge, capable of regularly discovering 24 rooms in the first level of the game and exceeding average human performance (Cobbe *et al.*)

RND's success is impressive, regularly achieving good results in hard exploration Atari games (Burda *et al*, 2018, p.8). We do not, however, want our methods to have good performance only on the environment that they have been trained on. A key feature of human intelligence is the ability to transfer insights acquired while solving one problem to more easily solve another. In other words, we want our policies to be generalizable. Cobbe *et al.*, in the OpenAI blog post presenting RND, note that RND's ability to generalize was not yet tested, specifically pointing out "train[ing] a curious agent on many environments without reward and investigat[ing] the transfer to target environments with rewards" as one future direction for research. In this project, my goal is to evaluate, albeit on a small scale, the capability of policies learned by RND to transfer to other tasks.

2 Related Work and Algorithms

2.1 Random Network Distillation

Before beginning to discuss methods, let us first properly introduce what Random Network Distillation is. Random Network Distillation is a method of generating exploration bonuses, rewards that push an agent to explore the environment in the absence of extrinsic rewards that are supplied by the MDP each transition. We can also refer to these bonuses as intrinsic rewards (Burda *et al*, 2018, p.2).

In order to compute intrinsic rewards for each timestep, RND introduces a randomly generated prediction problem. We first randomly initialize a neural network, titled the target network, and fix it. We then randomly initialize another neural network with the same architecture and title it the predictor network. When our agent takes an action, it obtains an observation from the environment. The predictor network aims to predict the output of the target network on the observation. The prediction error, which is the squared error of the outputs of the predictor and target networks, becomes our intrinsic reward. We then train the predictor network to minimize the prediction error (p. 3).

The intuition behind this reward is simple. Firstly, we can expect the squared error to be low in states that we have previously visited due to training, and high in unseen states. As a result, our agent will become "curious," exploring novel states (p. 3). Secondly, unlike forward dynamics methods which aim to compute a similar intrinsic reward by predicting the next state, RND eliminates the issue of stochasticity by making the prediction problem deterministic: the output of the fixed target network (p. 4).

2.2 Proximal Policy Optimization

RND is not a standalone Reinforcement Learning algorithm. It is instead used to extend previous approaches with exploration bonuses. Following Burda *et al.* I use Proximal Policy Optimization (PPO) in this project. PPO is a policy gradient method designed to be robust, easy to train and be data efficient (Schulman *et al*, 2017, p. 1).

Similar to other policy gradient methods, PPO directly estimates a stochastic policy using a neural network and alternates between sampling observations using the policy and optimizing the policy with the observations. The difference with PPO is the optimization process. Let θ be the parameters of the policy network and let $\pi_\theta(a_t|s_t)$ be the probability of taking action a_t at state s_t according to the policy parameterized by θ. The PPO objective then is:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \text{ where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

with θ_{old} being the parameters with which observations were sampled and $\hat{A}_t$ being our estimate of the advantage (p. 3). PPO performs multiple epochs of optimization on this loss rather than the regular one.

The intuition behind this clipped loss is that while we wish to increase the probability of actions that lead to positive advantages and reduce the probability of actions that lead to negative advantages, we do not want our updated policy to stray too far from the old policy, to prevent stability issues. By clipping the ratio and then taking the minimum, we can ensure that the update is as small as possible (p. 3).

Let T be the horizon. Then, Schulman *et al.* estimate advantages using Generalized Advantage Estimation (GAE):

$$\hat{A}_t = \sum_{t'=t}^{T-1} (\gamma\lambda)^{t'-t}\delta_{t'}, \text{ where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

γ is the discount factor, λ is a hyperparameter that determines how much weight to place into later observations and $V(s_t)$ is our learned estimate of the value for s_t, itself represented by a neural network (p. 5).

Similar to Schulman *et al*, I make the policy and the value networks share parameters, allowing us to update them both simultaneously. Let L^{VF} be the squared loss of our value estimate and the returns and let S denote entropy. The complete loss function for PPO then becomes (p. 5):

$$L(\theta) = \hat{E}_t[L^{CLIP}(\theta) - c_1 L^{VF} + c_2 S[\pi_\theta](s_t)]$$

2.3 Combining PPO and RND

Having set PPO up, it is simple to incorporate intrinsic rewards into the algorithm. We represent the extrinsic and intrinsic value of a given state using two separate value heads, which both share parameters with the policy network. The overall value of a particular state is then computed by adding the two value estimates (Burda *et al*, 2018, p. 5).

Since we are using two value heads, we can estimate the values for intrinsic and extrinsic rewards distinctly. The first change that can be made is using different discount factors for each reward (p. 5). The other change made is treating the intrinsic rewards as non-episodic. In a non-episodic setting, we do not set the value of the end state to 0, with the intuition that unlike the extrinsic setting, the value of a particular state is not bound to the episode that it occurs in (p. 4).

3 Methodology and Experiments

3.1 The Sonic Benchmark

For evaluation, I will use the Sonic Benchmark introduced by Nichols *et al.* In the benchmark, levels from the trilogy of Sonic the Hedgehog games on the Sega Genesis are randomly split into training and test sets. The proposed model is trained on the training set for an indefinite period of time. The learned policy is then transferred to each test level independently. In each test level, we finetune the model for 1 million timesteps and average the total reward per training episode. The average score across all test levels then becomes our final score (Nichols *et al.*, 2018, p. 6).

In the original benchmark, Nichols *et al.* have a split of 47 training levels and 11 test levels. Due to computational restrictions I am considering only 8 training levels and 3 test levels. The training levels considered are Angel Island Zone Act 1, Oil Ocean Zone Act 1, Spring Yard Zone Act 1, Ice Cap Zone Act 1, Launch Base Zone Act 1, Hill Top Zone Act 1, Hydrocity Zone Act 2, and Labyrinth Zone Act 2. The test levels considered, on the other hand, are Chemical Plant Zone Act 2, Green Hill Zone Act 3 and Death Egg Zone Act 2.

3.2 The Sonic MDP

In this section, we give a short overview of the MDP Nichols *et al.* used to model the game. Similar to the original DQN paper, every observation is converted to greyscale and resized to an 84×84 image. The four most recent frames observed are stacked, making a given state a $4 \times 84 \times 84$ tensor (Mnih *et al.*, 2015, p. 6). The start state is supplied by the Retro Gym emulator. An episode ends when the player finishes the level, dies, or spends 4500 timesteps (Nichols *et al*, 2018, p. 5).

The actions that Sonic can take are $\{\{\text{LEFT}\}, \{\text{RIGHT}\}, \{\text{LEFT, DOWN}\}, \{\text{RIGHT, DOWN}\}$ $, \{\text{DOWN}\}, \{\text{DOWN, B}\}, \{\text{B}\}\}$ (p. 8). Every action Sonic takes is repeated 4 times, with the action being repeated for a fifth time with probability 0.25 (p. 5).

For each action, Sonic is given his horizontal displacement as a reward, such that moving to the right, where the level ends, is encouraged, and moving to the left is discouraged. The displacements are normalized to sum to 9000. An additional bonus of 1000 points, which linearly decays to 0 over the course of 4500 timesteps, is given for level completion speed (p. 6). For the test levels, I follow Nichols *et al.* by not punishing Sonic for moving to the left and only rewarding him when he goes past the maximum displacement observed for a particular episode, to encourage backtracking. All rewards are scaled by 0.005 to bring to a more preferable range for PPO, making the displacement reward 45 and the time reward 5 (p. 9).

3.3 Experiments

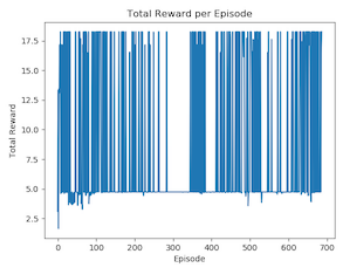
For the project I consider two main algorithms. PPO is my baseline while RND is my proposed method. For both methods, I conduct two sets of experiments. I first evaluate on the test set without any transfer learning to better understand the effect of transferring policies on the performance. I then train an agent on all training levels *jointly.* Environments for all training levels are run in parallel. The observations for all training levels are then combined into a large batch on which optimization is performed.

Joint training for PPO is conducted with 2 million timesteps for each level, resulting in a total 16 million timsesteps (64 million frames) of experience. Joint training for RND is conducted with 1 million timesteps for each level due to time constraints, on the other hand. For all models, a horizon size of 2048 was used with 4 epochs of optimization with minibatches of size 2048. The discount factors for intrinsic and extrinsic rewards were 0.95 and 0.99 respectively. The GAE parameter λ was set to 0.95 and the entropy coefficient was set to 0.005 for the PPO experiments and 0.1 for the RND experiments.

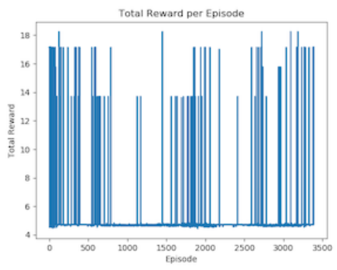
Following Burda *et al*, I base my model architectures after the architecutres used by Mnih *et al.* in their DQN paper (Burda *et al*, 2018, p. 16). The policy networks have the same first four layers as the DQN architecture (Mnih *et al.*, 2015, 6), and then split into multiple streams. The policy stream is a fully connected layer of size 7 with softmax and the value heads are individually fully connected layers of size 1. The target and predictor networks also share the same first four layers as the DQN architecture. These are then followed by a fully connected layer of size 256 with ReLU activation and a fully connected layer of size 32. The code for the project can be found at: https://github.com/omerrose/cs229-project

Algorithm	Chemical Plant Zone Act 2	Green Hill Zone Act 3	Death Egg Zone Act 2	Average
PPO	**8.40345395**	**3.03227703**	3.320360146	**4.918697042**
Joint PPO	5.017792998	1.761001096	**4.05359179**	3.610795295
RND	1.107330284	1.267312163	1.696329927	1.356990791
Joint RND	2.404323031	1.207537965	1.365825208	1.659228735

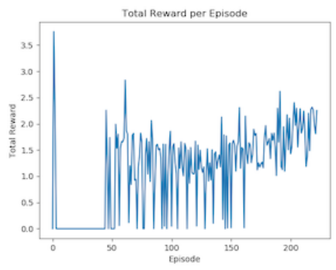
Table 1: Average total reward per episode computed during evaluation on test set



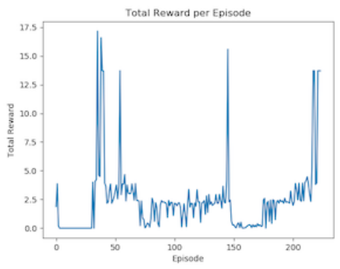
(a) PPO



(b) Joint PPO



(c) RND



(d) Joint RND

Figure 1: Total reward per episode on Chemical Plant Zone Act 2

4 Results and Error Analysis

Due to limitations in space, I have only included plots for Chemical Plant Zone. As such the analysis will primarily be a case study for this level.

In terms of results, there are multiple trends to take note of. The most important of all, however, is the quality of the results obtained. Standard PPO, one of the baselines, outperformed every other method in all categories but the average total reward for Death Egg Zone Act 2 where Joint PPO performs better. Nichols *et al* obtain a significant boost in performance after using Joint PPO (Nichols *et al.*, 2018, p. 10), which calls into question the validity of the results presented in this project.

There are multiple reasons why this may have occurred. The first, and most simple, is that there exists an implementation error. The lower scores obtained for Green Hill Zone and Death Egg Zone are partly due to Sonic repeatedly dying to enemies or spikes blocking the path. The persistence of this behaviour leads me to believe that there may be issues in how episode boundaries are treated by my code. One possibility is that Generalized Advantage Estimation is implemented poorly, leading to the wrong computation of values and advantages for the end of an episode. Another possibility has to do with the observations that OpenAI Gym emits at the end of an episode. My current implementation assumes that the first frame of the next episode is emitted when an episode ends and uses this observation as the first frame of the next episode. Assume, however, that the last frame emitted is that of Sonic dying. If I have a successful episode with a high return, the value of Sonic dying will be large since I use it as the first frame. As values are estimated with neural networks, this large value will generalize to the states that lead to Sonic dying and prevent him from avoiding the sequence of actions that lead to his demise.

We can also provide explanations specific to the algorithms used. Although PPO is meant to be an algorithm that is data efficient, it still remains a policy gradient method. The objective of PPO is computed with a Monte Carlo estimate consisting of a sample of size 1 while evaluating the models on the test set. PPO is, however, often run with multiple environments in parallel to collect more data and to perform less variable Monte Carlo estimates. The instability of the results, as seen in Figure 1, could therefore be explained by the high variance estimations.

Another notable trend is the worse performance of the RND methods compared to PPO, with PPO consistently and more frequently achieving higher scores. Although there yet again is the possibility of implementation errors, I believe that the issue has to do more with the algorithms themselves. Notice that the total reward plateaus to 0 between iterations 0 and 50 for both RND experiments. In the standard RND run, this is due to Sonic repeatedly running left while in the Joint RND run this is due to Sonic consistently crouching, performing no rightward motion that would result in rewards. We can, in fact, explain this in terms of searching for novel states. Unlike Montezuma's Revenge, a given frame of a Sonic game is highly dynamic, with the timer incrementing each second, clouds moving at the top of the screen, neon lights flashing on and off. Even when crouching and remaining motionless, the model could receive intrinsic rewards due to this dynamism. When the target network can adequately predict the output of the predictor network on this state, we see the algorithm beginning exploration yet again.

It is this conflict between intrinsic and extrinsic rewards that lead to worse performance compared to PPO. With only a single environment, as opposed to, for instance, 128 parallel environments as Burda *et al* run for their Montezuma's Revenge experiments (Burda *et al*, 2018, p. 5), our agent cannot learn which actions lead to novel states quickly, leading to subpar intrinsic value estimates. Since the overall advantage is computed by combining the intrinsic and extrinsic advantage, our model performance will suffer due to the low quality of the intrinsic advantages.

There is the final observation that joint training does not result in a significant boost for either PPO or RND. This is likely due to the fact that joint training was done with 8 environments, as opposed to 128 or more parallel environements, for a relatively short period of time. Any statement made regarding joint training will not be valid as a result.

5 Conclusion

In this project I aimed to determine how well policies trained using only intrinsic rewards computed with RND translated to other environments using both intrinsic and extrinsic rewards. Because the results obtained were highly subpar, I cannot make any conclusion regarding whether or not RND generalizes well.

To reach a state where I can confidently make such judgements, I must first review my code and compare with existing implementations to see if there are any fatal differences leading to lower performance. After this, I would need better computational power, as a single GPU machine cannot manage too many parallel environments of Sonic The Hedgehog, both in terms of memory and speed. Finally, I would need to perform hyperparameter tuning on my models to ensure that my run of RND is suitable for the Sonic games. It is only after this that I can ensure that my results are valid and that I can reach correct conclusions about

References

Burda, Y., Edwards, H., Storkey, A., Klimov, O. (2018). Exploration by random network distillation. ArXiv:1810.12894 [Cs.LG], , 1-17.

Cobbe, K., et al. (2018). Reinforcement learning with prediction-based rewards. Retrieved 11/29, 2018, from https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/

Details. (2018). Retrieved 10/23, 2018, from https://contest.openai.com/2018-1/details/

Hesse, C., Schulman, J., Pfau, V., Nichol, A., Klimov, O. Schiavo, L. (2018). Retro contest. Retrieved 10/23, 2018, from https://blog.openai.com/retro-contest/

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529-533.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., Schulman, J. (2018). Gotta learn fast: A new benchmark for generalization in RL. ArXiv:1804.03720v2 [Cs.LG],

openai. (2018). Gym retro. Retrieved 10/23, 2018, from https://github.com/openai/retro

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. ArXiv:1707.06347v2 [Cs.LG], , 1-12.