Hard Atari Games with Deep Reinforcement Learning

Yuan Gao veragao@umich.edu
Xiechen Wang xiechenw@umich.edu
Haoran Zhang zhr@umich.edu
Tian Zhou zhtian@umich.edu

July 29, 2019

Abstract

A grand challenge in Reinforcement Learning (RL) is intelligent exploration, especially when reward is sparse and hard to find. Montezuma's Revenge serves as a benchmark for such hard-exploration domains. On this game, many reinforcement learning algorithms perform poorly. We implement Random Network Distillation (RND) bonus, which is an exploration bonus for Deep RL methods that is easy to implement and adds minimal overhead to the computation performed. This exploration bonus is combined with Proximal Policy Optimization (PPO) algorithm. The reproduced RND model achieved 9500 points on Montezuma's Revenge. Based on RND, we have two improvements. First, we try to filter out meaningless information by downsampling the input images. Second, we add Self-Imitation Learning into our model to exploit good trajectories. The model is tested on four hard Atari games: Montezuma's Revenge, Hero, Venture and Gravitar.

1 Introduction

1.1 Problem Statement

The aim of RL is taking proper actions to maximize rewards in an environment. Thanks to the booming growth of Deep Learning, Deep RL evolved tremendously in recent years. Since the first Deep RL method [1] was proposed, people made lots of effort on this topic and came out many improvements, resulting in huge breakthrough in this realm. For example, Deep RL agents could play many Atari 2600 games and scored higher than human average and even human record [2]. What's more, the success of AlphaGo, a Deep RL computer program to play Go, shocked the world as it defeated the best Go player Lee Sedol in 2016 [3].

Even though Deep RL has conquered many problems, it still performs poorly in many hard Atari games. These games usually share the same feature - sparse rewards. One notorious sample is Montezuma's Revenge. In Montezuma's Revenge, the agent gets points by collecting keys, swords and torches, or slaying enemies and opening gates with swords and keys attained. However, as [2] depicted, Deep-Q Network (DQN), the most common Deep RL model, received 0 points.

Besides working on Montezuma's Revenge, we also compare same methods on three more hard Atari games: Gravitar, Venture and Hero where DQN also under performs.

1.2 Significance

The reason why DQN preformed poorly in Montezuma's Revenge and other hard games is that they are hard-exploration games. In these games, all bonuses are sparsely distributed in the environment so that most RL algorithms like DQN will never find the first reward before game ends. In other words, they failed to learn how to play Montezuma's Revenge and resulted in random play behavior.

In real life, hard-exploration problems are also the most challenging. That is because we usually only have abstract goals and reward functions towards the goal do not provide detailed path on how to reach the goal. What's more, it sometimes creates unintended local optima and get deceived by some sub-optimal goals with rewards.

For example, an artificial robot searching survivors in the ashes is a hard-exploration problem. In such situation, survivors (rewards) will be very few and sparse. What's more, if we also direct the robot to minimize damage to itself, this additional reward may lead it to repel exploring the environment since exploration is initially much more likely to result in damage than finding a survivor.

Thus, we need a strong exploration impulse to solve this problem. To prove our methods' effect, we work on several hard Atari games including Montezuma's Revenge, Gravitar, Venture and Hero. Figure 1 presents the screen captures of them. In these games, long trajectories are inevitable for agents to perform well and score fast and lastingly.

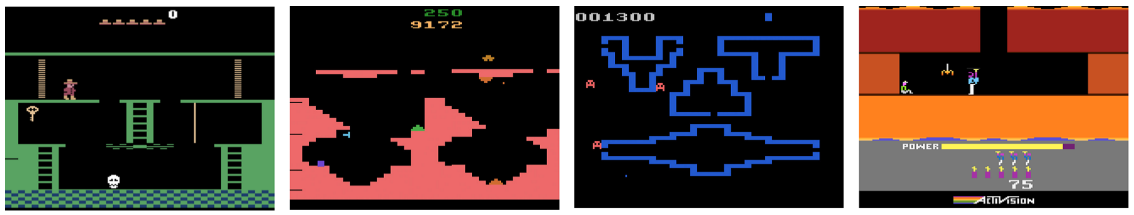

Figure 1: Screen captures of four Atari Games: 1) Montezuma's Revenge 2) Gravitar 3) Venture 4) Hero

2 Related Work

Policy Gradient (PG) algorithm [4] is indispensable to Deep RL methods. Compared to traditional value-based algorithms, it works better on large problems with continuous states and actions. However, instability hinders its usage in kaleidoscopic environments. Actor-Critic [5] became an excellent solution, it combines policy and value together and shares advantages from both sides. Its improvements on "actor" part, which are Advantage Actor-Critic (A2C) and Asynchronous A2C (A3C) [6], perform even better. PPO [7] as an enhancement on Actor-Critic's "critic" stage, also contributes a lot. At present, combining PPO and A2C is the most mature and popular method in Deep RL and usually serves as a common baseline.

Since the vanilla Deep Q-Network did not work well [2] due to inefficient exploring, people have raised different ways to activate it. The first method with an acceptable score is to unify count based exploration and intrinsic motivation [8]. Since then, more interesting approaches has emerged, including hierarchical control [9], count-based exploration [10], experience replay with self-imitation [11], studying expert demonstrations by browsing walkthrough videos [12] and a combination of transformed Bellman operator and temporal consistency [13]. Many methods above relied on RAM information or human expert's domain knowledge, so they are not universal and autonomous enough. Conversely, other algorithms cannot beat human beings.

Recently, some new methods based only on image pixels published recently outplayed performance of average human in this game and two of them are highly important. The first one is Random Network Distillation (RND) [14] based on an extra intrinsic reward to encourage agents exploring. The second one is Go-Explore [15], which takes advantage of cell representation and the popular "save and load" game-playing strategy and got 43,763 points - pretty remarkable. However, Go-Explore's excellence are based on several assumptions and it requires massive computational resources and memory.

Thus, we focus on RND method and try to find improvements for better performance in Atari games with sparse rewards.

3 Method

3.1 Background: Policy Optimization

3.1.1 Policy Gradient

PG methods work by approximating a stochastic policy directly using a function estimator [4]. The function can be represented by a neural network whose input is a representation of states, whose output is action selection probabilities and whose weights are the policy parameters.

Policy gradient methods seek policy parameters to maximize the performance measure $J(\theta)$, which is the discounted return in the cases we study. The most commonly used gradient estimator has the form:

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantages function at each timestep t. $\hat{\mathbb{E}}_t$ indicates the empirical average over an infinite batch of samples. The constructed objective function is

$$L^{PG} = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

The policy parameters are found by alternating between sampling and optimization. All samples need to be collected following the current policy, which means all samples need to be discarded after one step of update. While it is appealing to perform multiple steps of optimization using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates [7].

3.1.2 Trust Region Policy Optimization

To stabilize optimization process and use samples collected from the most recent policy as efficiently as possible, Trust Region Policy Optimization (TRPO) [16] exploits the relationships between the performance of two policies and gives the following objective function:

$$\begin{aligned} \max_\theta \quad & \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \\ s.t. \quad & \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_{\theta_{old}}(.|s_t)]] \leq \delta \end{aligned} \tag{3}$$

where $\pi_{\theta_{old}}$ is the policy before the update.

Although TRPO is a very powerful algorithm, it suffers from a significant problem: that bloody constraint, which adds additional overhead to this optimization problem. However, this problem can be efficiently be approximately solved by the conjugate gradient algorithm, instead of adding a constraint separately. The following equation shows incorporating it inside the objective function as a penalty.

$$\max_\theta \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL[\pi_{\theta_{old}}(.|s_t), \pi_{\theta_{old}}(.|s_t)]] \tag{4}$$

A small caveat is that is hard to choose the coefficient β in a way that it works well over the whole course of optimization. Experiments also show that it is not sufficient to simply choose a fixed penalty coefficient β. Additional modifications are required.

3.2 Proximal Policy Optimization

Considering the complexity of penalty, a promising solution clipped surrogate objective is put forward as PPO [7].

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t))] \tag{5}$$

Here, The ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ infers how different the two policies are and ϵ is a hyperparameter. The clipped term removes the incentive for moving $r_t(\theta)$ outside of the interval $[1-\epsilon, 1+\epsilon]$. Finally, we take the minimum of the clipped and unclipped objective, so the final objective is a lower bound on the unclipped objective. Which this scheme, we only ignore the change in probability ratio that would make the objective improve, and include it when it makes the objective worse.

Combining the clipped loss with Actor-Critic algorithm [5], we must use a loss function that combines the policy loss and value loss. The objective can also be augmented by adding an entropy bonus to ensure sufficient exploration.

$$L(\theta) = L_t^{CLIP}(\theta) - c_1 L_t^V(\theta) + c_2 Entropy \tag{6}$$

where $L_t^V(\theta)$ is the MSE between estimated value and target value. c_1, c_2 are coefficients.

3.3 Random Network Distillation

Burda and Edwards et al. proposed Random Network Distillation (RND) as an exploration bonus for deep RL [14]. This approach is particular simple to implement, works well with high-dimensional observations and can be used with any policy optimization algorithm.

Exploration bonus encourages an agent to explore even when the environment's reward e_t is sparse. They do so by replacing e_t with $r_t = e_t + i_t$, where i_t is the intrinsic reward (exploration bonus) associated with the transition at time t. The intrinsic reward of RND is based on the observation that neural networks tend to have a significantly lower prediction errors on examples similar to those on which they have been trained. This motivates the use of prediction errors of networks trained on the agent's past to quantify the novelty of new experience.

RND involves two neural networks: a fixed (untrainable) and randomly initialized *target* network which set the prediction problem, and a *predictor* network trained on data collected by agent.

- The target network takes an observation to an embedding $f : \mathcal{O} \to \mathbb{R}^k$.
- The predictor neural network $\hat{f} : \mathcal{O} \to \mathbb{R}^k$ is trained to minimize the MSE $||\hat{f}(x;\theta) - f(x)||^2$.

This process distills a randomly initialized neural network into a trained one. And the prediction error is used as an intrinsic reward.

RND can be used with any RL algorithm by adding the exploration bonus. We combine RND with PPO [7] as we illustrated in previous sections. Algorithm 1 gives the overall picture of the RND method [14].

Corresponding to intrinsic reward and extrinsic reward, the critic network gives both intrinsic value and extrinsic value. So the value loss $L_t^V(\theta)$ term in loss function (6) is given by

$$L_t^V(\theta) = L_{int}^V(\theta) + L_{ext}^V(\theta) \tag{7}$$

The network architecture is shown in Figure 2.

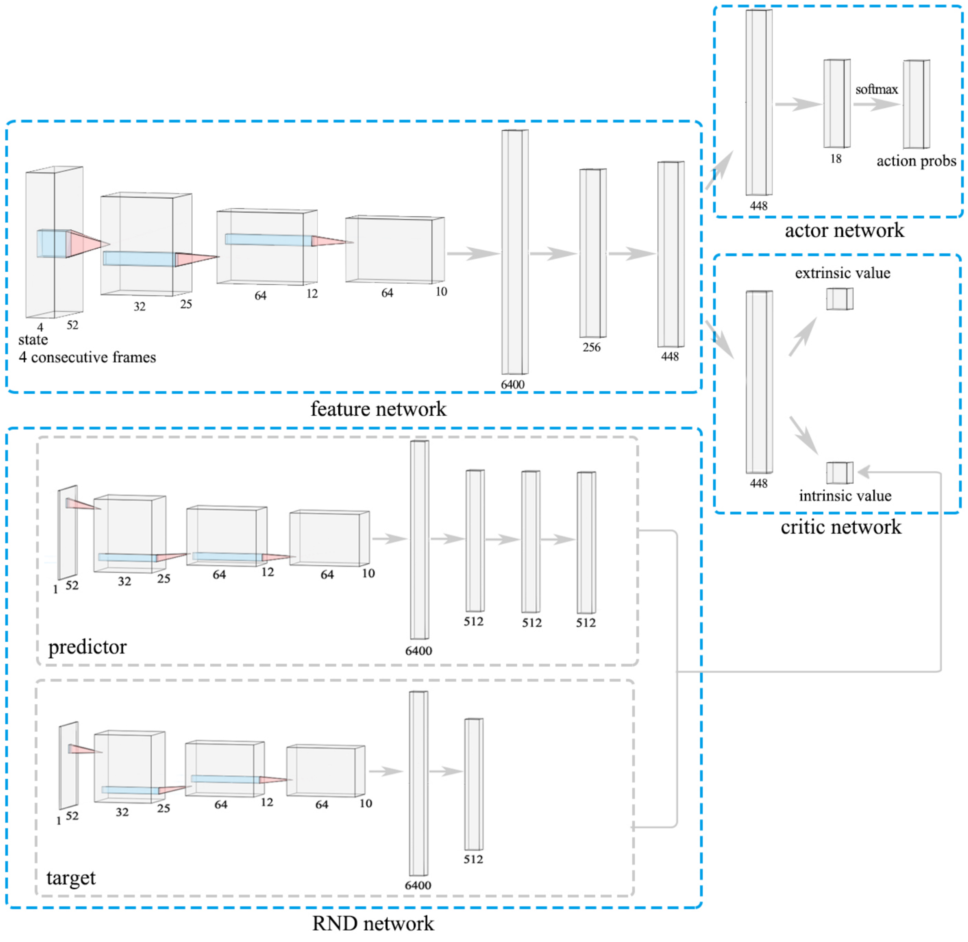

Figure 2: The overall network architecture including PPO and RND

Algorithm 1 RND pseudo-code

```
$N \leftarrow$ number of rollouts
$N_{opt} \leftarrow$ number of optimization steps
$K \leftarrow$ length of rollout
$M \leftarrow$ number of initial steps for initializing observation normalization
$t = 0$
Sample state $s_0 \sim p_0(s_0)$
for $m = 1$ to $M$ do
    sample $a_t \sim$ Uniform($a_t$)
    sample $s_{t+1} \sim p(s_{t+1} s_t, a_t)$
    Update observation normalization parameters using $s_{t+1}$
    $t+ = 1$
end for
for $i = 1$ to $N$ do
    for $j = 1$ to $K$ do
        sample $a_t \sim \pi(a_t|s_t)$
        sample $s_{t+1}, e_t \sim p(s_{t+1}, e_t|s_t, a_t)$
        calculate intrinsic reward $i_t = ||\hat{f}(s_{t+1}) - f(s_{t+1})||^2$
        add $s_t, s_{t+1}, a_t, e_t, i_t$ to optimization batch $B_i$
        Update reward normalization parameters using $i_t$
        $t+ = 1$
    end for
    Normalize the intrinsic rewards contained in $B_i$
    Calculate returns $R_{I,i}$ and advantages $A_{I,i}$ for intrinsic reward
    Calculate returns $R_{E,i}$ and advantages $A_{E,i}$ for extrinsic reward
    Calculate combined advantages $A_i = A_{I,i} + A_{E,i}$
    Update observation normalization parameters using$B_i$
    for $j = 1$ to $N_{opt}$ do
        optimize $\theta_\pi$ wrt PPO loss on batch $B_i, R_i, A_i$ using Adam
        optimize $\theta_{\hat{f}}$ wrt distillation loss on $B_i$ using Adam
    end for
end for
```

3.4 Self-Imitation Learning

The goal of Self-Imitation Learning (SIL) [11] is to imitate the agent's past good experience. Store past episodes with cumulative rewards in a replay buffer and exploit good state-action pairs a few more times to strengthen the learning effect. Junhyuk and Yijie proposed a off-policy Actor-Critic loss [11], which allows the agent to learn only when the return in the past is greater than the agent's estimate $R > V_\theta$. This encourages the agent to imitate its good decisions in the past. Moreover, in order to get good state-action pairs that satisfy $R > V_\theta$, they also proposed to use the prioritized experience replay [17], and the sampling probability is porpotional to $(R - V_\theta(s))_+$.

In our implementation, to be more compatible with the whole network, we use a PPO based loss function instead of the off-policy Actor-Critic loss proposed by authors.

$$L^{sil}_{policy} = \hat{\mathbb{E}}_t[\min(r_t(\theta)(\hat{A}_t)_+,\ clip(r_t(\theta), 1-\epsilon, 1+\epsilon)(\hat{A}_t)_+)] \tag{8}$$

$$L^{sil}_{value_{int}} = \frac{1}{2}||(R_{int} - V_{int}(s))_+||^2 \tag{9}$$

$$L^{sil}_{value_{ext}} = \frac{1}{2}||(R_{ext} - V_{ext}(s))_+||^2 \tag{10}$$

Because of the limitation of CPU memory, we only select state-action pairs sampled from current policy without building a large prioritized replay buffer.

In our model, SIL helps the agent exploit every encountered reward, and RND encourages the agent explore more unseen states. Trade-off between exploration and exploitation is one of the fundamental challenges in reinforcement learning. In hard-exploration problems, we may want to make full use of the rare rewards. However, exploiting too much will sacrifice the exploring ability provided by RND, or even destroy the whole learning process. So when trying to combine SIL and RND, we need to be very cautious about balancing exploration and exploitation. In our model, the SIL loss is multiplied by 0.5 to prevent SIL from taking very large step.

3.5 Resize

The most popular way to pre-process image-based Atari games is resizing, as known as image scaling. This is to say, shrinking the image size for better computing efficiency and smaller storage space. Also, it helps eliminate redundant information. Usually in Atari games, people implement nearest neighbor downsampling from 210×160 to 84×84. However, size 84×84 is just an empirical result and we desire more. Thus, we propose bilinear interpolation method for a better resizing quality.

Bilinear interpolation considers the closest 2×2 neighborhood of known pixels surrounding the unknown pixel. Then it takes a weighted average of these 4 pixels as the final value. Interpolated images are much smoother than nearest neighbor methods. Besides, it even keeps some key information that nearest neighbor's 84×84 image neglects. As Figure 3 shown, the yellow suspended line is lost after traditional nearest neighbors method whereas bilinear interpolation still keeps it even the images are smaller. Therefore, such an output is both computing effective and omniscient.

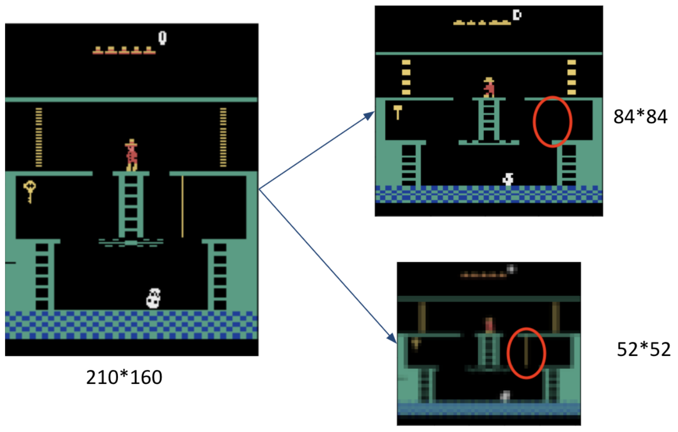

Figure 3: Nearest neighbor image scaling and bilinear interpolation

4 Experiments

In Atari games, one natural performance measure is the final score. Better agent achieves higher final scores. Besides, training speed of models could be another important consideration.

We use OpenAI Atari 2600 environment to test our method and compare output scores with other state-of-the-art methods as well as human-level. Due to restriction of computing resources, we used 32 parallel environments to collect experience for Gravitar, Venture and Hero and 128 environments for Montezuma's Revenge. Mean episodic return is the running average of 40 consecutive episodes. The results are shown in Figure 4.

For Montezuma's Revenge, vanilla RND performs best in terms of the final score. However, it does not tell the full story because Montezuma's Revenge has multiple paths to reach high scores, then it is not enough to decide the method effects simply from final scores. In other words, due to RND's magnificent uncertainties, agent would go to left or right pathway randomly after getting the second key and achieved different scores with same ability. Therefore, comparing the quality of different methods should also be judged by the time of completing difficult tasks, including getting the first key, opening door and getting sword in Room 5. These tasks are on the same fixed path and result in a total of 500 points.

We also want to check whether resize would affect continuing exploring ability of RND itself like SIL. As shown in Figure 4, resize would not affect this significant exploration superiority. 7100 is the best score that can be achieved on the left path under vanilla RND. Our resized model also achieve the same score. While its best score 9500 is achieved on the right path, this totally results from path choice of the agent.

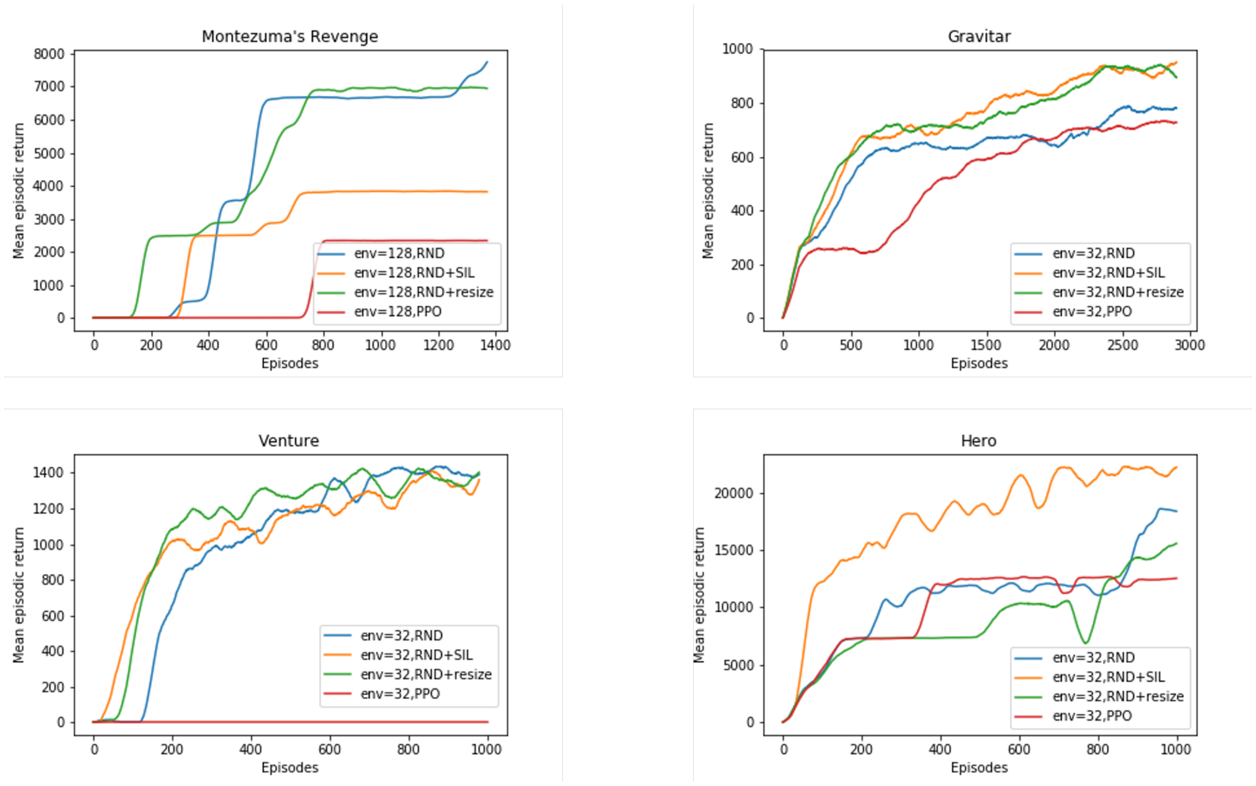

Figure 4: Mean episodic returns of different methods on different games

To better evaluate the resize method, we run multiple times and record the time agent get key and open reward in room 1. As shown in Figure 6, RND+resize model got the 500 points at around 110 episodes, while vanilla RND used around 250 episodes. Results show that resize method get these rewards ahead of original way at least 140 episodes. Reducing useless information disturbance and keeping track of key information may bring this improvement. Meanwhile, the relative small input size to neural network can accelerate the model's calculation time and save the buffer space. Especially, training time is always a big issue in RL and need a large buffer storage to process data.

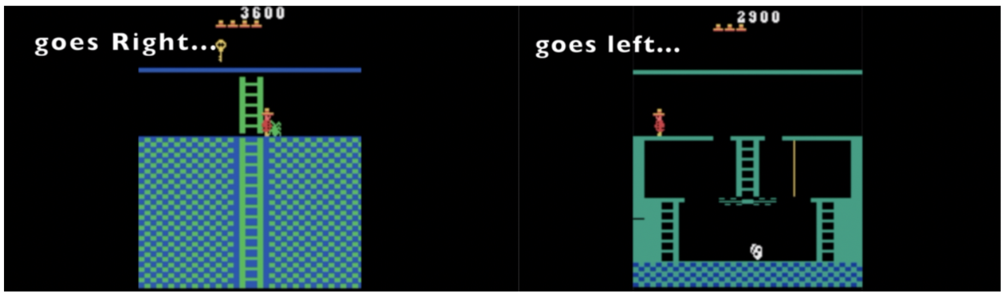

Figure 5: Score depends on the first gate opened

We also test these methods on other Atari games and regard PPO as the baseline. For Venture, both RND+SIL and RND+resize get rewards faster than RND and both methods perform well in the end, but PPO can not get any rewards. RND+SIL prevails among all algorithms on Gravitar and Hero, which proves our former discussion again, when rewards become denser, the RND+SIL perform better. the performance of PPO is similar with RND on these two games. However, these three games' score are quite high, which mainly results from low parallel environments. We expect huge improvement with 128 parallel environments, but it still can prove RND+SIL and RND+resize improve on these games.

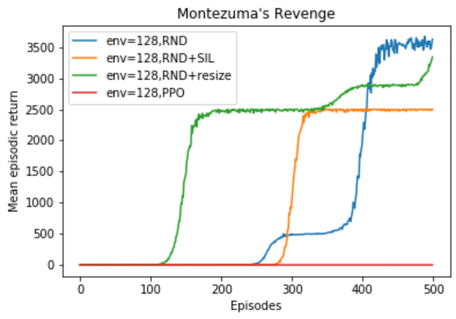

Figure 6: Mean episodic return of different methods on Montezuma's Revenge

5 Conclusion

Resizing eliminates much redundant information to help train more effectively, whereas too much shrinking might lead to key information loss. Choosing a proper downsampling algorithm and a suitable size would result in a better performance. In our experiments, bilinear interpolating to 52×52 shows promising results. SIL is conducive to exploit and study on visited good trajectories whereas sometimes it sacrifices the strong exploration ability of RND. This conflict is particularly obvious in Montezuma's revenge. We should emphasizing the SIL effect when rewards are not extremely sparse just like Hero. Both enhanced RND methods are proved to be useful and further improvement is needed.

Also, since the training is time-consuming, we only run each experiment once or twice. More experiments are preferred for better accuracy. Even though Go-Explore [15] relies on some assumptions and vast computing and storage resources hinders our deployment, it is overwhelming results supposes that its ideas might help our methods perform better.

6 Statement of Contributions

Tian Zhou and Xiechen Wang designed the project.

Tian Zhou, Xiechen Wang, Haoran Zhang and Yuan Gao performed the experiments.

Haoran Zhang and Yuan Gao analyzed the results.

Tian Zhou developed the theoretical framework.

Xiechen Wang, Haoran Zhang and Yuan Gao designed the poster for presentation.

Tian Zhou, Xiechen Wang, Haoran Zhang and Yuan Gao wrote the article.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[4] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[5] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.

[9] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3540–3549.

[10] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2721–2730.

[11] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," *arXiv preprint arXiv:1806.05635*, 2018.

[12] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching youtube," in *Advances in Neural Information Processing Systems*, 2018, pp. 2935–2945.

[13] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Večerík *et al.*, "Observe and look further: Achieving consistent performance on atari," *arXiv preprint arXiv:1805.11593*, 2018.

[14] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[15] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[16] J. Hui. (2018) Rl-trust region policy optimization (trpo) explained. [Online]. Available: https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *ICLR*, 2016.

A Demo of Trained Models

We uploaded a demo for the model of four Atari games. `https://www.youtube.com/watch?v=XqJfA4Oozr0`

B Performance on Atari Games

Table 1: Mean final scores of last 40 episodes

	Gravitar	Venture	Hero	Montezuma's Revenge
PPO	719	0	12,525	2,497
RND	765	1,392	18,356	**7,826**
RND+SIL	**967**	1,367	**22,193**	3,900
RND+resize	895	**1,406**	15,556	6,947
Avg. Human	3,351	1,188	25,763	4,367

Table 1 shows the average reward of last 40 episodes under different methods and environments instead of the highest scores. For Montezuma's Revenge, the highest scores of RND method and RND+resize method are 9,500 and 7,100. It will take some time to make the average points approach the highest ones. Because of the time limitation, we stopped training right after some 9,500-point episodes and the scores will be stable if we have more time.