Comparison between TD(0) and REINFORCE with baseline in procgen

Riccardo Roveri
Department of Computer Science and Engineering
Alma Mater Studiorum University of Bologna
riccardo.roveri2@studio.unibo.it

Abstract

Reinforcement learning (RL) excels in training agents for dynamic tasks but it often requires lot of training time and data. The procgen benchmark addresses these issues with procedurally generated environments. This paper evaluates RL agent performance within the procgen benchmark, using simplified settings for faster learning. One-Step Actor-Critic (AC) was first chosen for its balance of simplicity and efficiency. Initial training with TD(0) was ineffective, leading to modifications such as negative rewards for game termination and entropy regularization. REINFORCE with baseline outperformed TD(0) in learning the correct policy with the imposed constraints.

1 Introduction

Reinforcement learning (RL) has emerged as a powerful paradigm for training agents to solve complex tasks by interacting with dynamic environments. Traditional benchmarks, while useful, often suffer from limitations such as overfitting to specific environments, lack of diversity, and difficulty in generalizing to novel scenarios. To address these challenges, OpenAI introduced the procgen benchmark, a suite of procedurally generated environments designed to evaluate the robustness and generalization capabilities of reinforcement learning algorithms Cobbe et al. [2019].

The procgen benchmark provides a diverse set of environments that vary significantly from one instance to another, making it an ideal test-bed for assessing the performance of RL agents under conditions of high variability and uncertainty. Each environment within the benchmark is procedurally generated, enabling to create a large number of environments trivial. This characteristic promotes the development of agents capable of generalizing from limited training data to a broad range of unseen situations. This makes it also a very challenging environment to train the agent on in respect to ALE environments that do no change over time Bellemare et al. [2013]. For this reason you can specify parameters that allow to tune the difficulty of the environments such as the level distribution and the in-game asset.

This article will not focus on the generalization aspect of procgen between environments but instead focus on the progression in different level of the same environment to evaluate how the different methods perform. In particular with small training time and little hyper-parameters optimization.

2 Design

2.1 Constraints

Due to the scale of procgen the following simplification are made to accelerate the learning rate, making the overall image more simple for an agent to understand by removing user generated asset

Preprint. Under review.

in favour of monochromatic assets.
This were the parameters given to gym:

- `distribution_mode=easy`
- `use_generated_assets=False`
- `use_backgrounds=False`
- `restrict_themes=True`
- `use_monochrome_assets=True`

To make any significant progress in training and to ensure better repeatability i fixed random seed and changed the number of levels during for different evaluation:

- `rand_seed=42`
- `start_level=0`
- `num_levels=n`
- `use_sequential_levels=False`

The different environment offer the same observation and action space but different reward system due to the different nature of the game. Some game like leaper, coinrun and climber give a positive reward when reaching a coin (climber and coinrun) or ending the level (leaper) and zero otherwise, leading to have the majority of episode ending with zero reward regardless of the agent actions. Other games like chaser or fruitbot are more dynamic so the agent action have a more immediate impact on the score of the agent leading to better visualization of it's performance. In this document i choose to analyze fruitbot and chaser to better show the agent performance in a limited number of levels, they both have also smaller episode that helps with development interations.

2.2 Learning algorithm

In Reinforcement Learning (RL), selecting the right algorithm is crucial for effective learning and performance. Let's evaluate the moste popular ones:

Deep Q-Learning (DQN)
DQN, a value-based method, is great for discrete action spaces but struggles with continuous actions and lacks the sophistication of policy-based methods Mnih et al. [2013].

REINFORCE
REINFORCE is simple but suffers from high variance and slow convergence due to its reliance on episodic returns.Williams [1992].

REINFORCE with Baseline
REINFORCE with baseline improves upon this by incorporating the state-value function compute the actor loss. This provides more accurate values for the error because it has G not an estimate of it like in TD learning at the expense to running the entire episode. In contrast, TD(0) is a value-based method that updates the value function incrementally through bootstrapping, combining immediate rewards and the estimated value of subsequent states. TD(0) generally offers lower variance and faster convergence in value estimation but introduces bias due to reliance on current value estimates.

One-Step Actor-Critic (AC)
One-Step AC balances the simplicity of REINFORCE and the complexity of PPO. It uses a single-step temporal difference (TD) error for efficient learning without complex updates. It also does not require a replay buffer which helps with the implementation because is fully online. This algorithms suffer from poor performance when running on a single environment and on CPU because it has to run an update for each episode. There are other variant of TD Learning like Forward and Backward TD(λ) learning but are not used for reason similar to PPO. Another popular variant is Advantage Actor critic (A2C) and Asynchronous Advantage Actor Critic (A3C) that try to mitigate some shortcomings about actor critic Mnih et al. [2016].

Proximal Policy Optimization (PPO)
PPO is robust and handles large action spaces well, but its complexity and need for extensive hyperparameter tuning make it less desirable for simpler applications Schulman et al. [2017].

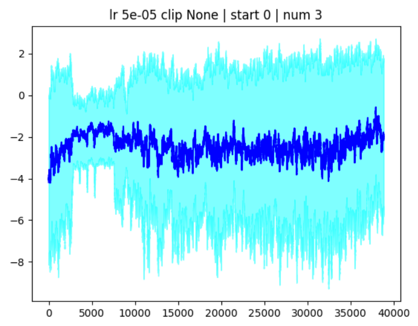

Figure 1: Environment fruitbot using TD(0) with Nature CNN

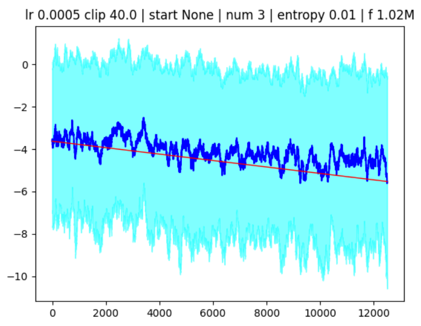

Figure 2: Environment fruitbot using A2C with optimization

2.3 Neural Network

The simplifications made to the procgen environment make is similar to Atari games than the default settings that have generated asset, in this case all the element are identified by monochromatic rectangles, but maintain an increased level complexity and variety.

At first i tried using the neural network architecture chosen to solve Atari Breakout (Nature CNN), that provides a small network as that is able to process images Mnih et al. [2013]Mnih et al. [2015]. The adaptation for using for doing actor critic was just to change the output layer for providing the probability distribution and critic value.

When trying to use the model with a different range of learning rate (1e-3 to 1e-5) and different environments (chaser, fruitbot, climber) and it seemed to not learn. This is expected and said in chapter 2.3 of the original paper Cobbe et al. [2019] but i expected that with a simpler environment (no background,no generated asset, monochromatic asset) that would not be the case.

To exclude any limitation due to network size and complexity i chose to use a simplified version of the IMPALA CNN Espeholt et al. [2018] that used only the network part and not the training optimization like v-traces or the distributed training. Initially i implemented the network removing the second iteration of the logic block consisting of conv2d,max,residual,residual to try to decrease training time but not found significant gain so i left as the paper suggested.

I also evaluated the option to use AlexNET Krizhevsky et al. [2012] but is larger than my IMPALA implementation and does not have Residual block He et al. [2015] that generally improve accuracy and mitigates the vanishing gradient problem, so i didn't use it.

3 Learning

3.1 TD(0)

When initially trying to train an agent on procgen, on paper i thought that TD(0) could offer the best performance as a starting point for doing further exploration. It seems that on both environment with sparse and dense return the agent did not make any significant progress. I expected the performance to be quite poor in the context of sparse return like coinrun or climber because it struggle to back-propagate a reward in all the previous state. In environment more dense like fruitbot or even more like chaser i expected to see a different behaviour but the reward graph remains quite flat.

Changing the hyper-parameter did not seam to help, generally having an high learning rate should make the training more unstable but this was not the case. In fruitbot for example after about 3M time-step the agent had the same performance as it first started as a random agent. Changing the learning rate from 6e-4 to 6e-5 did not change the reward graph so i believe that the problem is fundamental rather than on the hyper-parameter selection

3.2 End of episode

When looking at the behaviour of the agent it seems that in fruitbot it just ignore the presence of walls. The reward provided by procgen is zero as it counts as just end of episode but from a game point of view the agent performed poorly because it missed possible future rewards. I believe that with enough time the agent would learn this policy but to speed up training i made so that if is the end of the game i assume the agent died and gave him a negative reward. The amplitude of this negative reward can be considered as an hyper-parameter and for and can be tweaked for optimality, in any case setting it negative has an impact and id has to vary from game to game in line with the reward structure. In fruitbot for example i ranged from -0.1 to -1.0 not to deviate too much from the original.

In environment like coinrun or climber this optimization does not make sense because the end of episode occurs when finishing the episode or running out of time. A possible approach would be to punish the agent for every time-step passed to reward more for reaching the reward earlier.

3.3 Layer initialization

In the early stage of development i tried to initialize the last dense layer to zero for both the actor and the critic to start from a policy that is not in favour of any particular action. This did not improve in meaningful way training so i did not explore further this idea. When inspecting the agent behaviour the resulting policy was to stay still because most of the action are null, due to the shared action space between all the games, and the ones that are not counteract each other (like going left and right).

3.4 Entropy

Looking at the agent behaviour it seemed to get stuck doing certain actions, for example in fruitbot i saw many times the agent just staying still until it died. This is not a terrible policy as it seems because with the level number of 5 most of the levels at the start allow the agent to survive staying in a straight line. In chaser for example i noticed the agent getting stuck in corners or in climber just hugging the wall and jumping.

While punishing the agent for dying early did help i needed a better solution to discourage having a deterministic policy this early in the training process. For the actor loss i added a term that encourages exploration rather than deterministic policy as described in Mnih et al. [2016]. The addition of entropy in the training loop achieved what i hoped, the agent did never learn a deterministic policy like before and instead showed a more stochastic behaviour. In all the training i used the same entropy weight used in the original paper of 0.01 but can be tweaked as any other hyper-parameter of the network.

This is not surprising because TD learning is fully online and on-policy which means that exploration is not considered in the training loop. There are more complex variant that in fact try to solve this exact issue like soft actor critic Haarnoja et al. [2018] that takes into account entropy by optimizing a surrogate function and is in fact considered an off-policy method.

3.5 Performance

With this optimization the agent did not learn any policy that performs any better than a random agent. I believe that having just a single frame and a single future reward is too little for this kind of environments. I did not try . To improve the first issue the frame stacking could help in training as described in the original DQN paper Mnih et al. [2015]. For the second problem could be solved using TD(n) or TD(λ), in both cases when at the upper limit is equivalent to REINFORCE with baseline that calculates the critic loss on the true discounted reward not the critic estimate.

3.6 A2C

The idea behind Advantage Actor Critic is to provide a better loss for the actor by changing the critic function to a action-value Q function like in DQN instead of a state-value function. This improves the estimation of the error and should improve the training. In the limited testing i did that was not the case and the learning seems the same as in vanilla actor-critic so i chose to better evaluate another method as described in the next section.

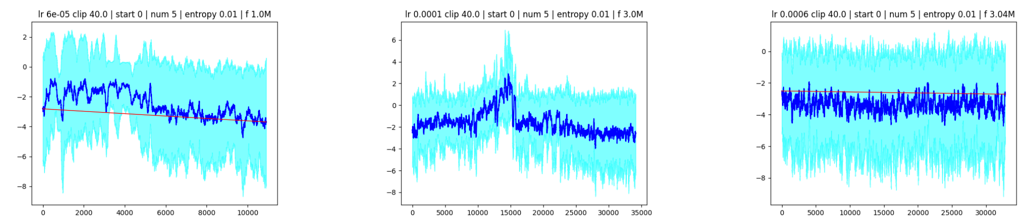

Figure 3: Environment fruitbot using TD(0) with optimizations with different hyper-parameters

3.7 REINFORCE with baseline

I chose to evaluate how REINFORCE with baseline would perform in comparison to TD(0) in procgen with the current setup. I applied the same optimization as before, so adjusting the final reward and using entropy to encourage exploration.

The adaptation is not big, i just added a buffer to store the episode and a function to calculate the trajectory of all the cumulative rewards. We lose the advantage of having a fully online learning process but due to the, relatively, small episode size is not a problem with memory. In my case the training was faster, i believe because applying all the gradients in one go is faster than having to do it each time-step. I wanted to compare them with the same hyper-parameter to have

The result are positive and the agent seem to learn a policy looking at the reward graph that is quite different from TD learning. I tried with different hyper-parameters and number of level and seems to work for environments with dense returns.

fruitbot
Looking and how it plays fruitbot it shows that avoids the non-fruit items and tries to get fruit correctly, it seems that still struggles with the walls but because of the optimizations made i believe that with time or decreasing the done reward it can overcome this.

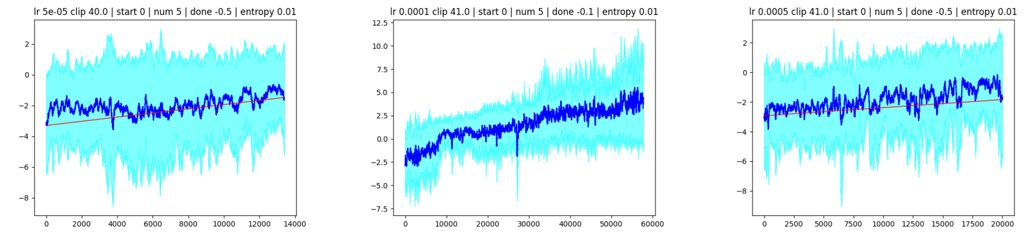

Figure 4: Environment fruitbot using REINFORCE with baseline and optimizations with different hyper-parameters

chaser
In chaser is noticed an increased difficulty for the agent to learn a policy, semantically i believe chaser is more difficult than fruitbot so this is expected. In this case i wanted to show how the learning with a different number of generated levels to see how it effects the learning. As expected with fewer level the agent can learn faster and with better performance, but the training and the network allows it to learn also more complex policy that can accommodate a larger number of level.

3.8 Hyper-parameters

Parameter	Value
Optimizer	ADAM
Gamma	0.999
Entropy Weight	0.01
Clipnorm	40
Random Seed	42

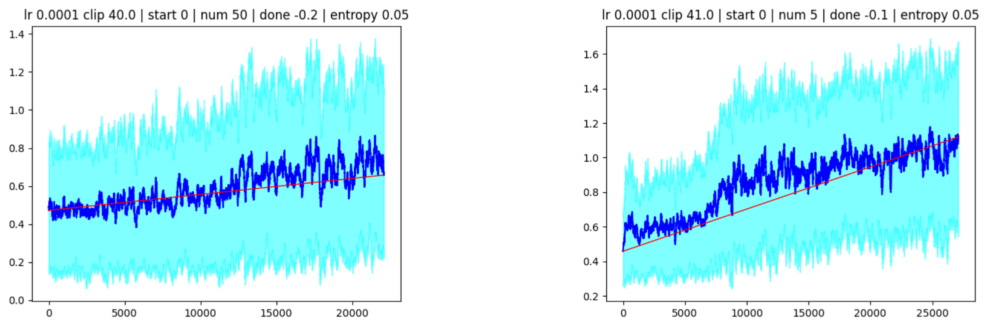

Figure 5: Environment chaser using REINFORCE with baseline and optimizations comparing different level numbers

4 Conclusion

The result are not significant in general because of the limited environment tested on and the relatively limited number of training step, but show the difference between the algorithms in the first stages of training. There are lots of hyper-parameter that greatly influence the resulting training, starting from the environment itself so further testing should be done on that. The training done was with just a single environment at a time, still testing generalization due to the different level but less in respect to the original paper..

The next step is to try frame-stacking to evaluate how it performs on both TD and REINFORCE learning in absolute term and in relation to each other. It would also be interesting to see the performance difference if using the original assets from the game or a background.

References

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL `http://dx.doi.org/10.1613/jair.3912`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL `https://doi.org/10.1038/nature14236`.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL `http://arxiv.org/abs/1802.01561`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.