GAMING IN 2020: COMPARING REINFORCEMENT LEARNING ALGORITHMS
PPO VS TRULY PPO

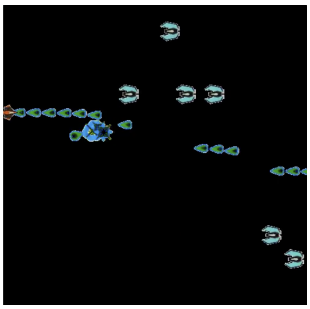
Nicolai Weisbjerg, Kelvin Foster

s174466, s174210

ABSTRACT

This paper investigates and compares the two reinforcement learning algorithms Proximal Policy Optimization (PPO) and an extension of it called Truly PPO on two Procgen atari-like games. With basis in Y. Wang et al.[1], we analyze the properties of Truly PPO, comparing it to the gold standard of PPO. Both theoretically and emperically, we show that Truly PPO is better at bounding the KL divergence between the policy updates, thus in theory adapting a better performance stability. We also show that Truly PPO is a more explorative algorithm, measured by the policy entropy. In terms of generalizability, we do not find conclusive evidence either way.

Index Terms— Reinforcement Learning, PPO, Truly PPO



(a) Starpilot

(b) Chaser

Fig. 1. The two games from the Procgen environment that will be used for the analysis. The first is a classic space-shooter (a), and the second is a Pacman-like game (b).

1. INTRODUCTION

Within the domain of thinking machines, one of the highest goals is the construction of an artificial general intelligence. A hypothesized road towards this goal is reinforcement learning (RL)[2]. In order to test the effectiveness of this approach, video games are regularly used[3], due to their complex decision-making environments where one can often construct a well-defined reward function.
In order to systematize the process of playing games with RL, the OpenAI research company created the Procgen Benchmark environtment, where the chief aim was to *"provide a direct measure of how quickly a reinforcement learning agent learns generalizable skills."*[4]. The environment consists of 16 different games that tests various aspects of playing games. In our paper, the aim is to test two different reinforcement learning algorithms on two qualitatively games from the Procgen environment, namely Starpilot and Chaser (see Fig. 1).

Specifically, we will use the algorithms Proximal Policy Optimization (PPO)[5] and an extension of the algorithm called Truly PPO[1]. The former is often considered the gold standard for many deep RL problems. As the names of the algorithms imply, they are of the Policy Gradient type, which is one of the major branches within RL algorithms[6]. Additionally, Policy Gradient use an on-policy learning approach. In our paper, we will investigate the two algorithms both theoretically and emperically, looking into the connections between these two aspects. Through the analysis, general RL properties such as the explore/exploit trade-off and generalizability will emerge. This will give rise to a discussion of the pros and cons of the two algorithms.

2. RELATED WORK

In this section, an analysis of the two methodoligies will be carried out based on [5] & [1] in order to establish the foundation for the further research. This section will also touch upon the findings of Y. Wang et al. in [1] in terms of the differences in agent behavior between PPO and TrulyPPO. A baseline goal of this paper is to confirm these findings.

PPO was introduced in [5] as a "new family of Policy Gradient methods", which should strike "a favorable balance between sample complexity, simplicity, and wall-time", compared to other online policy gradient methods. The main idea behind desiging PPO was to introduce a more generel and simpler algorithm in terms of implementation than the Trust-region Policy Optimization, TRPO. TRPO is also a policy gradient method, which has some nice properties. TRPO enforces a hard constraint on the policy updates based on the KL-divergence (trust-region) between the old and new policy, which theoretically can gurantee a monotonic performance improvement[1]. However, the hard constraint on the policy updates makes TRPO a second-order optimization method,

which makes it computational inefficient and difficult to scale up. PPO was introduced as an alternative that tried to include the main idea of TRPO, i.e. constraining the policy updates, and thereby also adapting some of the performance properties of TRPO, but without compromising on the first-order optimatization. PPO makes use of a soft constraint (by clipping on the objective function) that triggers whenever the likelihood ratio between the new and old policy on a taken action with positive advantage exceeds a certain level. This somewhat restricts the agent in learning too fast on a single timestep, thus forcing it to explore more and thereby increase performance stability.

However, as argued by Y. Wang et al. in [1], PPO actually neither constrains the likelihood ratio nor enforces a trust-region bound on the updates[1], and it is therefore still at risk of performance instability. This gave reason to question if there was possibly an even better way of implementing the main idea of TRPO in a first-order derivative optimizer. Y. Wang et al. adressed this question by introducing TrulyPPO. In essence, TrulyPPO is an enhanched version of PPO. The two main differences are the adoption of new trigger for the clipping function as well as a rollback operation in order to restrict differences between the old and new policy even further. The new trigger is based directly on the KL-divergence between the old and new policy as opposed to the likelihood ratio, as this, in theory, should enforce a well-defined trust-region constraint[1]. Y. Wang et al. found that TrulyPPO significantly improved training stability and ability in restricting the policy updates, as expected. Moreover, they found that the agent behaviour for policies based on TrulyPPO did tend to be more explorative and random than that of PPO. In generel they concluded that KL-divergence-based algorithms, such as TrulyPPO, outperformed likelihood-based ones, like PPO.

3. METHODS

In this section, we will delve into the theory and methods used in our analysis.

3.1. Setup

In order to understand both the algorithms and the corresponding results from the agent, it is necessary to get an overview of the setup that they live in (see Fig. 2). As mentioned in Section 1, the algorithms in this paper use online/on-policy learning, which means that generated samples are only used for optimizing the policy once (i.e. as opposed to having a buffer in off-policy learning).

To initialize the loop of getting rewards and creating new, updated policies, the first step is to generate samples using an orthogonally initialized policy. This initialization method has been proven to be efficient for deep networks[7]. The policy is modelled by the Nature CNN encoder, which is the default

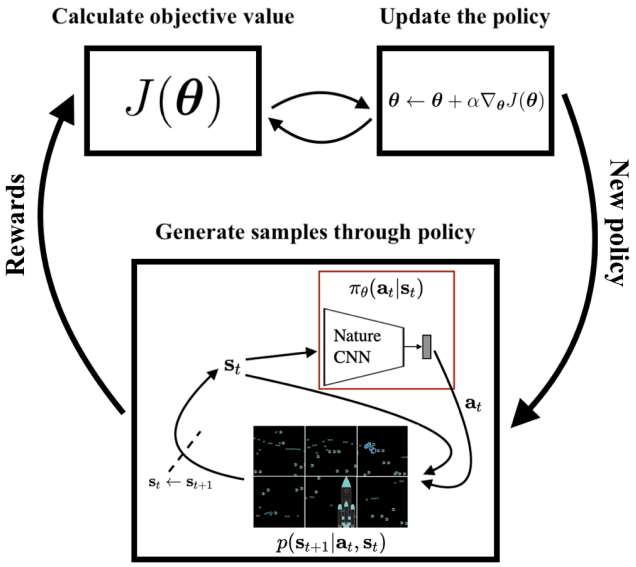


Fig. 2. The RL setup used for the analysis in this paper.

choice at OpenAI[8]. In this setup, the CNN also encodes the input state for the value function.
Samples are then simply generated by letting the agent interact with the enviroment by taking actions based on the policy. From these generated samples, rewards can then be calculated. These rewards are then plugged into the objective function, $J(\boldsymbol{\theta})$, the gradients of which are used to determine the updates of the policy. One then alternates for some number of iterations between calculating $J(\boldsymbol{\theta})$ and updating the policy. This results in a new policy that can be used to generate new samples, and thus the loop continues.

3.2. Objective function

This is the objective function that is approximately maximized in the setup described above in section 3.1:

$$J(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[L_t^{\pi}(\boldsymbol{\theta}) - c_L L_t^{VF}(\boldsymbol{\theta}) + c_H H[\pi_{\boldsymbol{\theta}}](\mathbf{s}_t) \right]. \quad (1)$$

The objective function takes 3 things into account. Firstly, $L_\pi(\boldsymbol{\theta})$ depends on the algorithm used to determine the policy. In the case of PPO, this would correspond to (7). In the case of Truly PPO, it would correspond to (10).

Secondly, (1) takes into account the squared-error loss of the value function, weighted by some user-defined hyperparameter value c_L. Specifically, it is the squared error between the parameterized value of the state and the target value from the roll-out. Thirdly, the objective takes the entropy bonus of the policy into account. The goal of the entropy term is to ensure that the agent explores the environment sufficiently, instead of just immidiately exploiting some perceived advantage. This term is also weighted by a hyperparameter, here c_H.

3.3. Background for Proximal Policy Optimization

In order to understand the PPO algorithm, a bit of background is needed.

3.3.1. Policy Gradients

As mentioned in section 1, the Proximal Policy Optimization algorithm is of the Policy Gradient type. For this type, the aim is to directly determine some stochastic policy for the agent by estimating the gradients of the policy. In this framework, the objective of interest to be maximized is:

$$L^{PG}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\log \pi_{\boldsymbol{\theta}} \left(a_t \mid \mathbf{s}_t \right) \hat{A}_t \right]. \tag{2}$$

Where $\pi_{\boldsymbol{\theta}}$ is the policy; a_t is the action; $\mathbf{s}_t$ is the state; $\hat{A}_t$ is the advantage estimate. The advantage is calculated by subtracting some baseline estimate (i.e. what we would expect happens) from our discounted rewards (i.e. what actually happened when we used our policy). As such, the advantage is positive when we perform better than expected (and vice versa). The baseline estimate is calculated by using the value function. The factor $\log \pi_{\boldsymbol{\theta}} \left(a_t \mid \mathbf{s}_t \right)$ is the log-probabilities of the action given some environment state. Since we want to maximize the probability of actions that result in a positive advantage, it makes sense that this is an maximization objective. The gradients are then estimated in order to improve/update the policy:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a_t \mid s_t \right) \hat{A}_t \right]. \tag{3}$$

However, using this pure policy gradient approach may risk overfitting to the current batch, which will generally not be representative of the overall environment. To circumvent this problem, some conservative force is often applied to the updates of the policy, one such example being Trust Region methods. In order to understand this set of methods, it is necessary to have an understanding of the Kullback–Leibler divergence.

3.3.2. Kullback–Leibler Divergence

The Kullback–Leibler Divergence (KL divergence) is a measure of the relative entropy[9] between two probability distributions P and Q. In more intuitive terms, it measures how different two distributions are. If the measure is high, they are different (and vice versa). In the discrete case, it is expressed as:

$$D_{\mathrm{KL}}(P \| Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \tag{4}$$

3.3.3. Trust Region Policy Optimization

The first core idea in the Trust Region Policy Optimization algorithm (TRPO) is to use a likelihood ratio between the new and old policy. This idea was first proposed in [10]. The other core idea in TRPO is to directly constrain the size of the policy update by measuring the KL divergence (once again) between the old and new policy. Thus, the second-order optimization problem of TRPO can be expressed as:

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \\ \text{s.t.} \quad & \mathbb{E}_t \left[D_{\mathrm{KL}} \left[\pi_{\boldsymbol{\theta}_{\text{old}}} \left(\cdot \mid s_t \right), \pi_{\boldsymbol{\theta}} \left(\cdot \mid s_t \right) \right] \right] \leq \delta. \end{aligned} \tag{5}$$

Both core ideas can be framed in relatively intuitive terms. If the likelihood ratio $\frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)} > 0$, then the given action, a_t, is more likely under the new policy than the old one (given some state, s_t). Thus, if the ratio is very large, then the policy has moved (too) far away from the old one. The constraint tell us the KL divergence must be lower than or equal some bound δ. This hard-constrained version of policy gradients is, however, a second order optimization problem, which is computationally expensive. This, finally, leads us to the first algorithm that will be used in this paper: Proximal Policy Optimization.

3.4. Proximal Policy Optimization

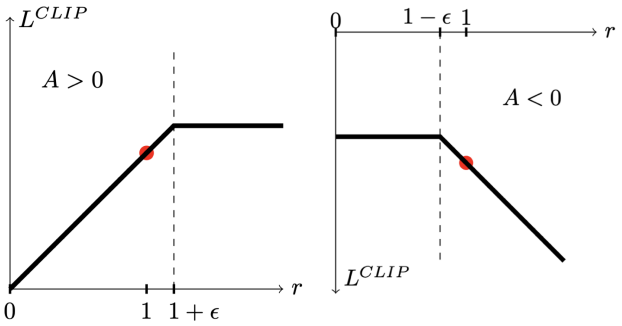
The starting point for the PPO algorithm is to consider the unconstrained version of TRPO:

$$\hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}} \left(a_t \mid \boldsymbol{s}_t \right)}{\pi_{\boldsymbol{\theta}_{\text{old}}} \left(a_t \mid \boldsymbol{s}_t \right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\pi_{\boldsymbol{\theta}}) \hat{A}_t \right] \tag{6}$$

where $r_t(\pi_{\boldsymbol{\theta}}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)}$ is introduced for readability. In order to make the optimization problem first-order, the constraint of TRPO is discarded and the following, altered objective function is proposed by J. Schulman et al.[5]:

$$L_{\pi}^{PPO}(\boldsymbol{\theta}) = \hat{\mathrm{E}}_t \left[\min \left(r_t \left(\pi_{\boldsymbol{\theta}} \right) \hat{\mathrm{A}}_t, \mathrm{clip} \left(r_t \left(\pi_{\boldsymbol{\theta}} \right), 1 - \epsilon, 1 + \epsilon \right) \hat{\mathrm{A}}_t \right) \right] \tag{7}$$

where L_{π}^{PPO} is the same as L^{CLIP} in Fig. 3. ϵ is a hyperparameter. To understand what this objective does, it is necessary to first look at what the two terms in the min-operater mean. The first term, $r_t \left(\pi_{\boldsymbol{\theta}} \right) \hat{\mathrm{A}}_t$, is the same as the objective from TRPO (see (5)). The second term, $\mathrm{clip} \left(r_t \left(\pi_{\boldsymbol{\theta}} \right), 1 - \epsilon, 1 + \epsilon \right) \hat{\mathrm{A}}_t$, is a clipped version of that term. The two cases of the clipping function can be seen in Fig 3.



(a) $L_\pi^{PPO}(\boldsymbol{\theta})$ when $A > 0$ (b) $L_\pi^{PPO}(\boldsymbol{\theta})$ when $A < 0$

Fig. 3. Source: Schulman J., et al.: Proximal Policy Optimization Algorithms (2017)

Part (a) of Fig. 3 tells us how the objective is clipped when the advantage is positive. It tells us that if the ratio is too positive (measured by ϵ), meaning that the new policy has moved too far away from the old one, we will not get any added benifit, since the line simply flattens. If we make too big changes to our policy based on just one batch, then we risk overfitting since the batch will most likely not be representative of the overall environment.
On the other hand, when the advantage is negative, we should not make the given action too much more unlikely, again since we are only looking at one batch. When $A < 0$ there is a case of particular interest, namely when $r > 1$. In this region, the new policy makes an action that results in a negative advantage more likely, which is obviously not desired. This is remidied by the fact that the gradient is negative here and so the optimization will want to move the objective in the other direction, thus making said action less likely.

3.5. Problems with Proximal Policy Optimization

As shown in [5], the trigger limit constant ϵ of PPO does determine the level of restriction on the updates of the policy (also shown emperically in Section 4). However, as it has been shown, PPO has two theoretical issues. First of all, PPO does not even manage to bound the likehood ratio at the updates ([1], Theorem 2). Secondly, it does not enforce a trust-region bound on the updates, which was the major point of mimicing TRPO ([1], Theorem 3).

Problem 1: Intuitvely ([1], Theorem 2) says, that the clipping function only removes the incentives for driving $r_t(\pi_{\boldsymbol{\theta}})$ even further beyond the clipping range. However, that does not mean, that the remainding terms of the overall objective function (i.e. (1)) has no incentives for doing so. Since there is no difference between exceeding the clipping by a little, or by a lot, it is in risk of pushing $r(\pi_{\boldsymbol{\theta}})$ far beyond the clipping range.

Problem 2: By proof by example, it can easily be proven that it is neither trust-region bound. It can be shown that:

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & D_{\mathrm{KL}}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}, \pi_{\boldsymbol{\theta}}\right] = \infty \\ \text{s.t.} \quad & |1 - r_t(\pi_{\boldsymbol{\theta}})| < \epsilon, \end{aligned} \tag{8}$$

where $\pi_{\boldsymbol{\theta}} = \pi_{\boldsymbol{\theta}}(\cdot \mid s_t)$ and $r_t(\pi_{\boldsymbol{\theta}}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)}$.

Proof: Let $p^{(d)}$ denote the probability of an action d given the state s for the new policy, and respectively $p_{\text{old}}^{(d)}$ for the old policy, where $d \in \mathcal{A}$ for the discrete action-space of the agent where, $|\mathcal{A}| > 1$. Now let $a_t \in \mathcal{A}$ be the chosen action by the new policy such that:

$$1 - \epsilon < \frac{p^{(a_t)}}{p_{\text{old}}^{(a_t)}} < 1 + \epsilon,$$

and let $p^{(z)} = 0$ and $p_{\text{old}}^{(z)} > 0$, where $z \in \mathcal{A}$, $z \neq a_t$, we see that:

$$D_{\mathrm{KL}}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}, \pi_{\boldsymbol{\theta}}\right] = \sum_{d \in \mathcal{A}} p_{\text{old}}^{(d)} \log\left(\frac{p_{\text{old}}^{(d)}}{p^{(d)}}\right) = \infty. \tag{9}$$

This problem has been visualized in Figure 4. Please be aware of the difference in notation, $\boldsymbol{\theta} \sim \pi_{\boldsymbol{\theta}}$.

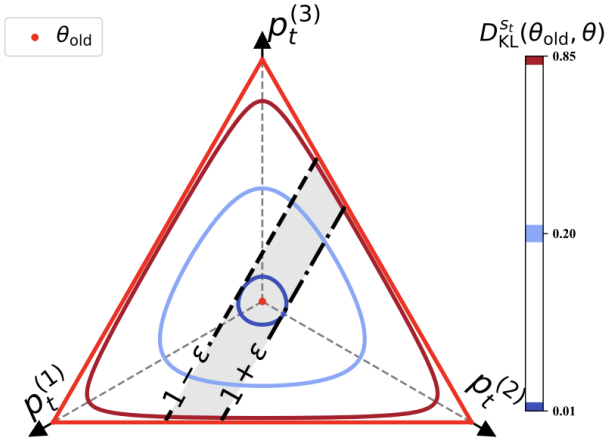


Fig. 4. Source: [1]. The red dot represents the action distribution of the old policy with actions-space $|\mathcal{A}| = 3$ for a given state $\mathbf{s}_t$, drawn on the hyperplane $\sum_{d \in \mathcal{A}} p_t^{(d)} = 1$. The grey area represents the likelihood-based clipping area for the case where action 2 was chosen by the new policy. The solid lines represent levels of KL divergence between the old and new policy $\{\pi_{\boldsymbol{\theta}} | D_{\mathrm{KL}}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}, \pi_{\boldsymbol{\theta}}\right] = \delta\}$.

3.6. Truly Proximal Policy Optimization

Y. Wang et al. adressed the above problems with PPO by introducing TrulyPPO. To address the fact that PPO does not constrain the updates on its likelihood-based clipping criteria,

Y. Wang et al. propossed the inclusion of a roll-back operation to the objective function ([1], Theorem 4). The roll-back operation should ensure that negivative incentives was enforced on policy updates beyond the clipping range, thus bounding the updates on its clipping criteria even more.
To ensure a more strictly constrained trust-region bound, Y. Wang et al. furthermore propossed to directly use the KL-divergence as the clipping metric, rather than the likelihood ratio, just like for TRPO. The final objective function can be seen below

$$L_{\pi}^{\text{Truly}}(\boldsymbol{\theta}) = r_t(\pi_{\boldsymbol{\theta}})\hat{\mathrm{A}}_t - \begin{cases} \alpha D_{\mathrm{KL}}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}, \pi_{\boldsymbol{\theta}}\right] & D_{\mathrm{KL}}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}, \pi_{\boldsymbol{\theta}}\right] \geq \delta \text{ and} \\ & r_t(\pi_{\boldsymbol{\theta}})\hat{\mathrm{A}}_t \geq r_t(\pi_{\boldsymbol{\theta}_{\text{old}}})\hat{\mathrm{A}}_t \\ \delta & \text{otherwise,} \end{cases} \tag{10}$$

where α is a constant that determines the level of roll-back. This in itself does not make a constraint on the trust-region. However, it is possible to achieve a hard constraint by tweaking the objective function slighty ([1], Theorem 5). Y. Wang et al. has shown emperically that the above proposed objective function does manage to bound the KL-divergence of the updates much more than PPO, thereby, emprically, adapting the proporties of performance stability from TRPO to a larger degree.

4. EXPERIMENTS AND RESULTS

In this section, we want to experimentally verify and analyze the differences between the two algorithms explained in section 3. In this instance we expect to see that TrulyPPO is better at establishing a bound on the KL-divergence thereby having a more stable performance increase. We also want to verify that TrulyPPO is more explorative than PPO, which will be measured by the entropy. We will also look into generalizability.

4.1. Comparison of PPO and TrulyPPO During Training

In all our experiments, all other parameters than those indicated on the legend are held equal across algorithms. For a full listing of all parameters used in the tests, please refer to our GitHub repository.

The graphs on Fig. 5 are based on the average across five runs with differents seeds. The shaded areas represent the associated standard deviation. The graph concerning the score is the average, non-discounted training return.

Score: First we see that PPO outperforms TrulyPPO based on the training score only. In the PPO runs with $\epsilon = 0.1$ we see that there for some of them were a huge decrease in training score after around 0.5×10^7 timesteps. This could be an indicator of the issues with performance stability for PPO as described in [1]. It is also worth to acknowledge that a more thorough exploration of tuning the hyperparameters

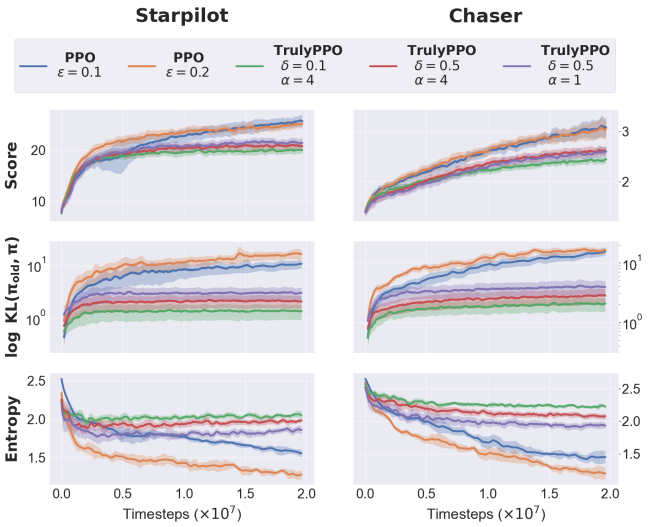


Fig. 5. Comparing the two algorithms during training measured on training score, the log of the max KL divergence for a given batch, and the Entropy for the two Procgen games Starpilot and Chaser. These graphs are constructed from a moving average using the last 500 observations.

for TrulyPPO could have resulted in a better score.

KL-divergence: In terms of the KL divergence, it is worth stressing that it is the logged values that are plotted, and that it is the maximum KL-divergence observed in each batch. With this in mind, it is quite clear that TrulyPPO is much better at constraining the KL divergence than PPO. We also see that altering the hyperparameters of TrulyPPO has a direct, tangible effect on the constraint level. Especially for Chaser, we see that the KL divergence seems to be steadily growing for PPO, whereas the graph flattens for TrulyPPO.

Entropy: The plots for the entropy show that TrulyPPO ensures a higher level of entropy for the entire duration of training, which in a sense means that the agent is more explorative. As time goes on, we see that the entropy is steadily declining for PPO, whereas TrulyPPO maintains a higher level. One could argue that it would be a good idea to have an iteration-dependent, controlled way of adjusting the entropy for TrulyPPO such that it becomes more exploitive towards the end of the training.

4.2. Comparison of PPO and TrulyPPO During Evaluation

In this section we will test the apparent hypothesis of Y. Wang et al. that TrulyPPO outperforms PPO. The graphs on Fig. 4.2 are only based on one run each, however with both training score and test score extracted from the same run. The test score is based on a rollout of the policy on a sequence of levels, that is has not been trained on. Furthermore, we chose to only include one PPO configuration and

one TrulyPPO configuration, as the others all virtually show the same behavior. For the chaser game it is especially easy

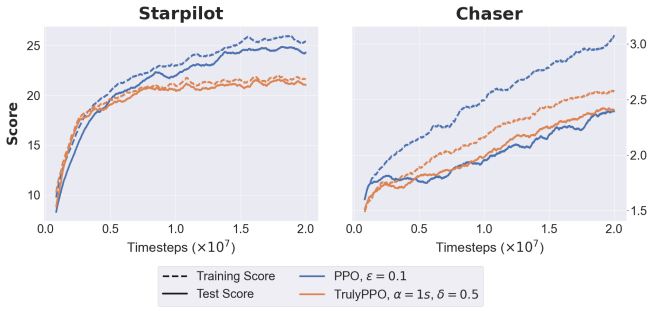


Fig. 6. Comparison of the two algorithms based on the training score and test score. These graphs are constructed from a moving average using the last 100 observations.

to see that the TrulyPPO policy tends to overfit much less on the training levels than that of PPO. Even though the training score is significantly larger for the PPO run, the test score is more or less the same for the algorithms - if any a bit larger for TrulyPPO.

For starpilot, the problem of overfitting for PPO is not as significant. For this game, we see that TrulyPPO outperforms PPO in the beginning but it flattens out very fast. This might be an indicator that our set of hyperparameters are too restrictive. It seems like the trade-off between exploring and exploiting should be tweaked in order to increase performance.

The two points above show that there are qualitative differences between the games that could be taken into account when choosing which algorithm is the best. One could argue that since chaser is played in a more schematic setting (i.e. the playable field is restricted to certain hallways), it is much more difficult for the agent to generalize on few levels than for starpilot, since it can quickly learn level-specific paths around the field. Thus, we intuitively think a few levels of starpilot is better at representing the general environment, meaning that learning from each level carries more value in terms of generalizability.

Table 1 shows the final scores obtained during evaluation. The numbers are based on averaging 10 rollouts from different seeds. We see that PPO generally outperforms TrulyPPO in terms of final evalutation score for Starpilot. For Chaser, TrulyPPO marginally outperforms PPO.

4.3. Final Agent in Action!

To see how the two algorithms perform in action for the chaser, please refer to VideoOfAgents. It is worth to notice the difference in exploratio/exploitation for the two policies. The one trained using PPO seems to be much more exploitive, whereas the one trained using TrulyPPO is more explorative.

	Starpilot	Chaser
PPO $\epsilon = 0.1$	21.40	2.79
PPO $\epsilon = 0.2$	21.14	2.73
Truly PPO $\delta = 0.1, \alpha = 4$	17.03	2.55
Truly PPO $\delta = 0.5, \alpha = 4$	19.13	2.87
Truly PPO $\delta = 0.5, \alpha = 1$	17.13	2.82

Table 1. Evaluation results

This is what we expected to see, both from theory and emperical results in 4.1

5. CONCLUSION

Y. Wang et al. concluded that TrulyPPO in general outformed PPO[1]. With our current level of hyperparameter optimization, we do not reach the same conclusion. In terms of both training and test score, PPO in general outperforms TrulyPPO. However, we do see that TrulyPPO is better at constraining and controlling the KL divergence directly. This might be strongly favourable in some cases, for which we feel it would be logical to look into TrulyPPO. We also see that TrulyPPO can ensure a higher entropy throughout the training duration. This results in a more random policy which favors exploration over exploitation. This might be a great trait in the begining of the training, since we do not want the agent to make too rash decisions in what is good or bad, before has gotten a better understanding of the underlying complexity. However, as training progresses we also see that the entropy stays relatively high for TrulyPPO, resulting in a policy that still really has not determined a well-defined way to play the game.

6. FUTURE WORK

First step is to look more into hyperparameter tuning of TrulyPPO, as we expect that this is the limiting factor for our results for this algorithm. There has been done little work for TrulyPPO so there are no well-established baselines.

Another avenue is to play additional games in the Procgen environment. This will build intuition for how PPO and TrulyPPO compare on a qualitative basis, which will make it easier to determine when/if TrulyPPO should be used instead of PPO.

One could also look into adapting the TrulyPPO algorithm to an actor-critic framework. Additionally, it would also make sense to try the Impala CNN as encoder, as it has been shown in [4] that it performs significantly better.

7. REFERENCES

[1] Y. Wang, H. He, and X. Tan, "Truly proximal policy optimization," *CoRR*, vol. abs/1903.07940, 2019.

[2] I. Arel, "Deep reinforcement learning as foundation for artificial general intelligence," *Atlantis Thinking Machines*, vol. 4, 2012.

[3] I. Szita, "Reinforcement learning in games," *Adaptation, Learning, and Optimization*, vol. 12, 2012.

[4] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," *CoRR*, vol. abs/1912.01588, 2019.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[6] S. Levine and V. Koltun, "Guided policy search," *International Conference on Machine Learning*, vol. 12, 2013.

[7] W. Hu, L. Xiao, and J. Pennington, "Provable benefit of orthogonal initialization in optimizing deep linear networks," *ICLR 2020 Conference*, 2020.

[8] Aurelian Tactics, *Custom Models with Baselines: IMPALA CNN, CNNs with Features, and Contra 3 Hard Mode*, 2019 (acessed 16/12-2020), `tinyurl.com/yazg4ze8`.

[9] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, 1951.

[10] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," *ICML 2002*, 2002.