IMPLEMENTING CURIOSITY AS REWARD VARIANCE TO HELP ESCAPE LOCAL OPTIMUM IN REINFORCEMENT LEARNING

Jounaid Beaufils
Maastricht University
j.beaufils@student.maastrichtuniversity.nl

ABSTRACT

We Investigate the use of Curiosity in avoiding sub-optimal convergence in Reinforcement Learning. This Curiosity is modelled as reward variance awareness both estimated and directly calculated. The reward variance awareness is implemented as modifications to Proximal Policy Optimisation. These models were then trained on an environment susceptible to sub-optimal convergence. The implementation details are compared with other implementations of curiosity in Reinforcement Learning. Finally, unexpected changes in convergence behaviour were noted but we outline further research areas needed to reach a conclusion on the effectiveness of the models as a way to avoid sub-optimal convergence. A personal reflection by the author on the experiences during their Bachelor Research Thesis is also presented.

1 Introduction

Reinforcement Learning (RL) is a distinctive field within machine learning where agents, such as a robotic arm, self-driving car or the paddle in a game of pong (see Figure 2a), learn from interacting with their environment [1]. The goal is for these agents to maximize a reward by acquiring complex or unknown behaviours that are often too intricate or unfeasible to implement manually. As the complexity of these environments increases (pong is a straightforward environment compared to the robotic arm or self-driving car) the chances for a sub-optimal convergence increase. In a sub-optimal convergence, the agent stagnates and is incapable of improving while there exists a better solution or behaviour. For a diagram of the RL cycle see Figure 1

In this paper we investigate a novel approach to escape sub-optimal convergence in Proximal Policy Optimisation (PPO)[2], a popular RL algorithm, by attempting to mimic human curiosity. The interest in mimicking curiosity is its contribution to human learning[3], as humans tend to investigate for the sake of discovery rather than simply because a task is rewarding in of itself. There are various ways of implementing curiosity, including asking the agent to predict the next state not judge its understanding of the environment. More of these methods will be discussed.

We modify an existing PPO implementation [4] to value reward variance (how varied the expected reward is from a given point forward) as a model for curiosity. We hypothesised the model will explore actions resulting in drastic changes rather than being limited to exploring actions with positive rewards. This would push the model out of a local optimum where all changes in behaviour negatively impact performance, into neighbouring slopes towards a better optimum. This is expected to yield better results than exploration done without awareness of results, as is the case with entropy.

Entropy in policy models instils uncertainty in the distribution of predicted action probabilities, driving the model to diminish disparities between these probabilities and thus fostering exploration. To illustrate, an action probability set like 0.65-0.15-0.2, with higher entropy than 0.85-0.1-0.05, promotes the exploration of different actions, even those not yielding immediate positive outcomes. This exploration-before-exploitation approach is fundamental for efficient learning, even though the role of entropy lessens as the agent undergoes more training.

While there has been some research into using reward variance [5, 6]for RL, to the best of our knowledge, the various methods we use to calculate reward variance have not been reported. Our methods calculate reward variance while training the agent rather than sampling the environment prior to training, this ensures that the environment we train in

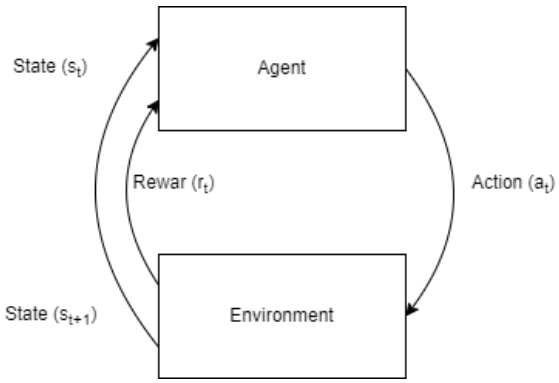
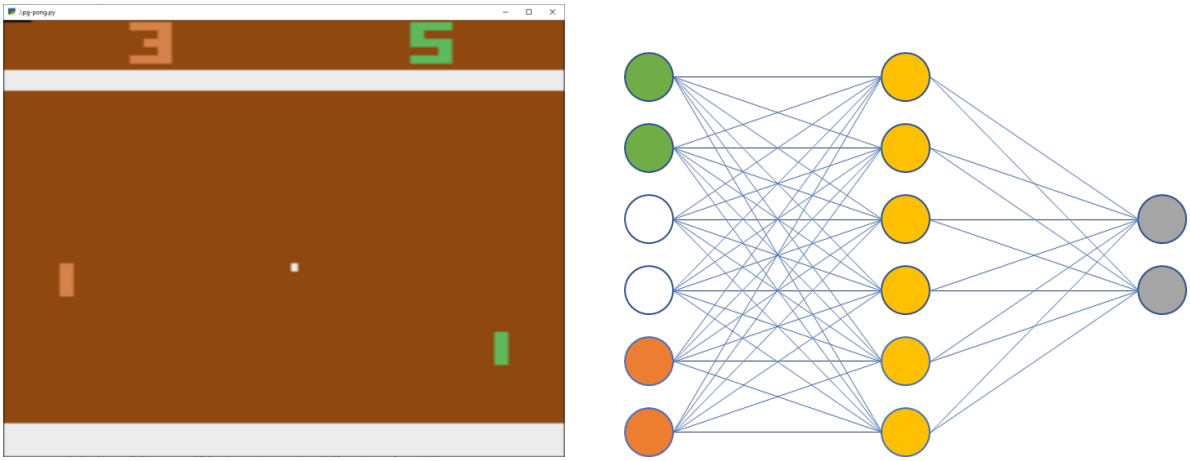


Figure 1: The Reinforcement Learning cycle, The agent acts (a_t) on a state at time t (s_t) and recieves the reward r_t and the next state (s_{t+1}). The cycle repeats, and each state-action-reward triple is stored. After a number of steps in the environment, the agent trains on the collected data and then interacts with the environment again.

can increase in complexity. Using reward variance as a way to model curiosity is also something we have not seen reported on and is different to the curiosity implementations discussed in Section 2.

Our research investigates whether reward variance has the potential to improve the convergence of RL algorithms in complex spaces.



(a) Pong environment from the Gym library

(b) Basic neural network structure

Figure 2: (a) In this game of Pong the policy might take every pixel, or just the (x,y) position of the two players and the ball as a representation of the state. This state representation is then used to predict the action to take. (b) Every line in the graph represents an adjustable weight while every node has a bias. Each yellow and grey node's value is the sum of the previous layers nodes multiplied by their weights w and that node's bias b or $a_{r,c} = a_{0,c}w_{0,c} + a_{1,c}w_{1,c} + ... + a_{r_{max},c}w_{r_{max},c} - b_{r,c}$ where $r = row, c = collumn$. As such this simple network contains 62 parameters.

2 Background

This is achieved by finding a (behavioural) policy implemented as, e.g., an artificial neural network (ANN; see Figure 2b). The learning occurs as the agent interacts with the environment, makes decisions based on its policy, and then adapts the policy based on the outcomes of those decisions. However, the outcome is not always clear, in the game of pong (Figure 2a) we don't necessarily know at each step if the right action was to move up or down; the reward for winning (+1) or losing (-1) the game is only discovered later. As a result, the state-action pair before a player scored will receive the full reward, and the second to last step will receive a fraction of the reward as it contributed less to the score. This reward discount is applied for every subsequent step. This cumulative discounted reward is called *return*.

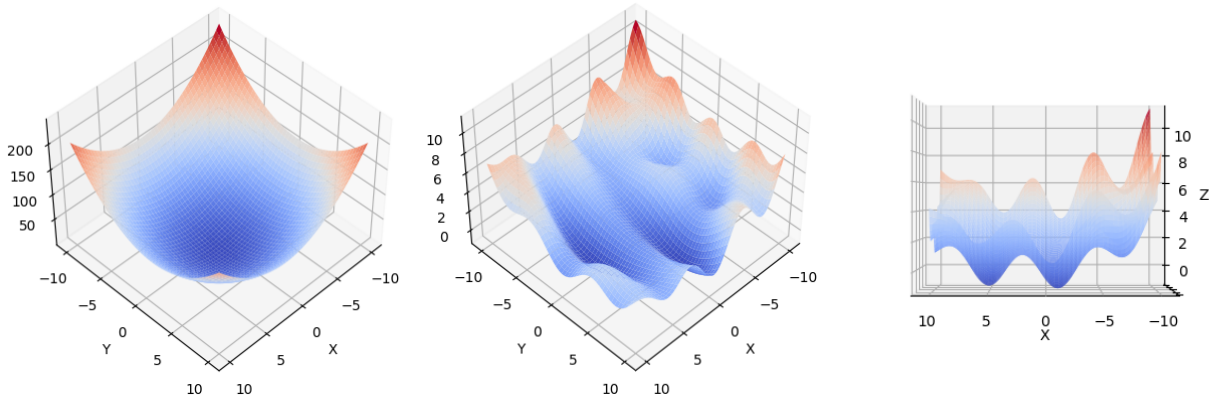
To train these learning models, a metric typically known as the "loss" is needed. The computation of this loss can vary depending on the specifics of the task. In many RL scenarios, the loss is not a single factor but a combination of multiple elements. One component might be a measure of how successfully the agent performed the task, while another component might be the extent of exploration the agent has conducted in its environment. Such exploration is rewarded because it can lead to the discovery of new and potentially more efficient solutions.

2.1 Backpropagation

The loss is backpropagated through the network to adjust its behaviour. The derivative of the loss with respect to the parameters of the ANN is calculated and the parameters are adjusted based on the gradient. This is possible because a neural network is a function with every weight and bias being a parameter (see Figure 2b). It can therefore be treated as a function with hundreds to billions of parameters. In order to maximise performance, an optimum is found by the derivative in the direction that increases the loss.

The ANN starts with its weights randomly initialised (many noise functions exist to initialise these weights, each with their own uses [7]). State-action-return sets collected, the loss and an approximation of the local gradient are calculated. The weights and biases are then adjusted accordingly so that model becomes more likely to predict an action that results in a low loss. This method is called *gradient descent*, repeating this cycle is how the RL models learn.

2.2 Non-Convex Optimisation



(a) 3D Convex Curve (b) 3D Non-convex Curve (c) 3D Non-convex Curve (Side View)

Figure 3: If one pictures releasing a bag of marbles on the curves (b) and (c) there would be multiple groups of marbles where as (a) would have a single group. This is the intuitive difference between a convex and a non-convex curve. Looking at (c) shows that if the optimisation follows the local gradient it can converge at any number of saddle or equilibrium points (black dots where the derivative is zero) the lower the convergence (on the Z axis) the better. There is no method that can guarantee the convergence find the lowest point but avoiding as many of the higher ones as possible results in increased performance from the network

Sub-optimal convergence can arise when the space we optimise in is non-convex, neither concave nor convex [8]. Within the context of ANNs and calculus a curve is non-convex if it has multiple points at which the derivative is zero, see Figure 3.

Multiple mathematical solutions exist to escape local minimum or maximum [8], including projecting the function into convex space or calculating Hessian matrices. However, these methods are impractical and computationally expensive for larger and more complex environments [8, 9]. Projecting the function into convex space changes the function in ways that make the solution inapplicable in most cases. Hessian matrices, on the other hand, require matrix inversion, which does not scale well with the number of parameters. Lastly, Zeroth-order methods aim to approximate the function's gradient without explicitly deriving it [9]. These are becoming popular in research but are not easy to access and integrate with machine learning libraries available. Overall, non-convex optimisation remains a challenging area in machine learning with many different solutions being researched.

2.3 Policy Gradient

Policy gradient methods are a class of algorithms that optimise the policy directly by backpropagating the reward generated by the environment directly through the network [2]. To stabilise the way the network's weights are updated the return is compared to a baseline, often the approximate average value of the state. The difference between return and baseline is termed *advantage*. Suppose the return for a certain action at a specific state is 100. Initially, this may seem favourable, but when considering that the average return from this state is 110, the computed advantage of -10 provides a more reliable metric for evaluating the agent's action.

In this work, the policy is an ANN which when given a state (such as the coordinates of objects in the simulation or all the pixels from the visual representation of the simulation) will output a probability distribution from which the agent will choose an action (see Figure 2).

2.4 Proximal Policy Optimisation

Proximal Policy Optimization (PPO) is a policy gradient algorithm that ensures stability by controlling how much the policy can change when optimising [2]. A number of agents use a policy to act in an environment T time steps to collect data. This data is then optimised on n times, concluding a cycle. The next cycle repeats this procedure with the updated policy.

Advantage Estimation in PPO

The advantage in PPO is estimated as follows:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{1}$$

$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2}$$

where t is an integer denoting the time step (or frame), the agent receives the state s_t and responds with the action a_t finally collecting a reward r_t and state s_{t+1}. In PPO the advantage $\hat{A}_t$ is never precisely calculated because the true state value is unknown. It is only estimated with the help of $V(s_t)$ predicting the value of s_t and implemented by another ANN.

δ_t is a measure of the gain made in s_t, as it sums the reward at a time t with the difference in value between time t and $t+1$ in a series of steps. This difference is equivalent to how much the agent expects to gain in the current state. Because δ_t requires multiple time steps $\hat{A}_t$ is only calculated after a number of steps T are taken. Lastly, λ_r is the discount applied to r and γ_r is a discount applied to $V(s_{t+1})$, these discounts are set between 0.95 and 0.99.

Surrogate Objective

The surrogate objective L^{CLIP} is the contribution to the loss controlling how the policy changes based on performance and differences between this policy and the previous one. Where $r_t(\theta)$ is the ratio of the action probability given the same state under the current policy and the previous policy used when collecting data.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \tag{3}$$

$$L^{CLIP} = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{4}$$

In Equation 4, the $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ term ensures that the value of $r_t(\theta)$ stays within ϵ of 1. The result is that large changes to the policy are not rewarded in the loss. The $\min$ results in a pessimistic bound on the objective as it can never be bigger than $(1+\epsilon)\hat{A}_t$ but there is no minimum. $\hat{E}_t$ is the expected value over observed steps.

Combined Loss

The loss used by PPO is as follows:

$$L^{CLIP+VF+S}(\theta) = \hat{E}_t[L^{CLIP} - c_1 L^{VF} + c_2 S[\pi_\theta](s_t)] \tag{5}$$

L^{VF} is the mean squared error between the predicted value and the returns for each s_t. $S[\pi_\theta](s_t)$ is the entropy of the action probability distribution. c_1 and c_2 are coefficients used to control how much of L^{VF} and $S[\pi_\theta](s_t)$ we want in $L^{CLIP+VF+S}(\theta)$

2.5 Curiosity in Reinforcement learning

One of the main drivers for human learning is curiosity [3]. In RL, this can be imitated by designing various metrics that mimic human curiosity, such as rewarding an agent for exploration independently of whether that exploration accomplishes the task. However, it has yet to be implemented widely in machine learning, much less fully understood. Over the last five years, more algorithms have been developed with various successful attempts to implement curiosity [10].

There are many ways to implement a curiosity metric. For instance, with simple count-based rewards [11], the agent is rewarded less for exploring previously visited states, because after all a curious agent is one that explores and so staying in a similar state should not be rewarded.

In Random Network Distillation (RND) [12], the algorithm trains an ANN to predict how the current state (a single frame of the environment) will be distorted when fed through a random ANN. The more familiar the image is, the better the prediction result. In this case, curiosity is the prediction error. Curiosity as predicting what the next state will be or how an ANN might distort the current state is implemented in the vast majority of curiosity-based models currently available [2, 10, 13]. The states that are familiar are easier to predict accurately. If the model is rewarded for prediction error while simultaneously becoming better at predicting states it has previously seen that it will be encouraged to explore new states, similar to human curiosity and its search for novel things.

Moreover, the SelMo model [14] first explores the environment in search of prediction error without any incentive to achieve a specific behaviour, similar to a child's uneducated and undirected curiosity. Snapshots of its policy are taken at regular intervals, and a Regularized Hierarchical Policy Optimization (RHPO) taken from Wulfmeier et al.'s work [15] then learns when to use which snapshot. Intuitively, snapshots are learned skills combined by the second ANN to complete the main task, much like humans combine skills acquired from past experiences to solve new tasks. Many other algorithms for curiosity exist and are reviewed in [10].

2.6 The Noisy TV Problem

The Noisy TV problem occurs when stochastic noise in the environment is interpreted as a prediction error worthy of curios exploration. An agent exploring a maze will stop at a wall displaying random images because there is more novelty there than exploring the maze[12].RND appeared to remove this issue by instead predicting how an additional ANN distorts a state rather than predicting the next state[12]. This prediction mechanism is more reliable, as the state distortion is not affecting my random noise in the environment such as animations with no intrinsic meaning.

However, Mavor-Parker et al. [16] show that RND and other similar models are still susceptible to noise when it is dependent on the agent's action. Meaning that if the next state is dependent on the agent's action but still contains elements of unpredictability the agent will return to the state and perform the same action in search of novelty. An intuitive example is if the TV had a remote which allowed the agent to change the displayed image. It would still be stuck.

As a solution, they predict the next state and its variance. They also distinguish between epistemic (predictable) and aleatoric (unpredictable) uncertainty. Predictable uncertainty is things that the agent can come to predict such as what happens to the position of an object as it travels through the environment. While the flicker of flame cannot be predicted.

They subtract unpredictable uncertainty from the total observed uncertainty to approximate predictable uncertainty, presented as the state's variance, which the agent learns to predict, is not rewarded for the variance that is deemed unpredictable. Kendal and Gal [17] detail the mathematical equations used by Mavor-Parker et al. [16]. These methods of calculating variance do not scale well as the state space increases. The method requires sampling the state space prior to training the agent, this limits the scale and complexity of the state space that can be trained on. However, the underlying information theory hints at the potential value of using variance-aware optimisation when training a neural network.

BYOL-Hindsight (Build Your Own Latent) is another method to separate predictable features from unpredictable noise [13]. The method encodes the environment into a set of vectors and then establishes which features in an environment are stochastic and thus cannot be predicted whether action-dependent or not. These features are removed from the error to avoid rewarding the agent for pursuing stochastic noise. The method is referred to as 'Hindsight' because it establishes after the fact which features it is failing to predict and should exclude in the next prediction.

Focusing only on the variance of future reward, Tamar et al. [5, 6] use several methods, including temporal differences and linear projections, to approximate the variance in the reward-to-go (reward available until the end of the episode). The terminal state assumption requires there to be only one success criterion. However, this method also requires matrix

inversion as well as sampling the state space of the environment to calculate the variance-to-go. These two requirements restrict their use in large models.

3 Algorithm

We have implemented multiple variants of our model to test different ways of using reward variance. Equation 5 is modified as denoted below, $\hat{\sigma}_t$ is the variance term, which is calculated differently in each model variant.

$$L^{CLIP+VF+Var}(\theta) = \hat{E}_t[L^{CLIP} - c_1 L^{VF} + c_2 S[\pi_\theta](s_t) + c_3\hat{\sigma}] \tag{6}$$

3.1 Absolute Value and Noise Baselines

$$\hat{\sigma}_t = \mid r_t \mid \tag{7}$$

$$\hat{\sigma}_t \sim U(0,1) \tag{8}$$

Our *Absolute Value* and *Noise Baseline* models are used as benchmarks. Absolute is the simplest implementation of a variance-like metric, it implements $\hat{\sigma}_t$ as $\mid r_t \mid$. By removing the sign of the reward we only account for the magnitude, removing all negative feedback. However the magnitude and variance of consecutive steps are simply ignored, removing any 'long-term awareness' in favour of immediate drastic action in every frame. The Noise model replaces $\hat{\sigma}_t$ with a random value between 0 and 1 sampled from a uniform distribution.

3.2 True Variance

The True Variance model attempts to assess the reward variance of a state by naively calculating the reward for the states that follow it. For every s_t the variance of all rewards collected thereafter is calculated. A major flaw in this model is the decreasing reliability of the data's variance as t approaches T. With s_T having a variance of 0, when in fact, the average reward variance in this state is likely higher than 0. This would have been calculated if more steps were taken in the environment. or if the number of steps taken corresponded with a terminal state such as winning or losing.

$$\hat{\sigma}_t = Var(r_{t:T}) \tag{9}$$

For every s_t, $\hat{\sigma}_t$ is the variance of reward up to s_T, hence $Var(r_{t:T})$.

3.3 Estimated Variance

The Estimated Variance model mirrors the advantage calculation in Section 2.4, we take the calculated reward variance for a state and add the expected future variance as predicted by an ANN. Because of variances dependence on sample size, the pooled variance (weighted mean) of the True Variance for the current state, the predicted variance for the current state and the predicted variance of the next state. The approach should increase the reliability of the variance estimation by including both a prediction and the True variance. An additional benefit of the Estimated Variance is the inclusion of a curiosity metric spanning multiple time-step. Curiosity metrics that include a long-term horizon have shown increased stability and performance compared to their counterparts without long-term curiosity [18].

$$\hat{\sigma} = \sigma^2_{pool_t} + (\kappa\lambda)\sigma^2_{pool_{t+1}} + ... + ... + (\kappa\lambda)^{T-t+1}\sigma^2_{pool_{T-1}} \tag{10}$$

$$\text{where } \sigma^2_{pool_t} = \frac{(n_t - 1)Var(r_{t:T}) + (N(s_t) - 1)\hat{\sigma}^2(s_t) + (N(s_t) - 1)\hat{\sigma}^2(s_{t+1})}{n_t + N(s_t) + N(s_{t+1}) - 3} \tag{11}$$

In Equation 2, δ_t is reduced if the $V(s_{t+1}) > V(s_t)$, similarly if $\hat{\sigma}^2(s_{t+1}) > \hat{\sigma}^2(s_t)$ and $N(s_t) = N(s_{t+1})$ then $\sigma^2_{pool_t}$ is reduced. κ is the variance discount that can be set separately from γ the reward discount. It should be noted that $\hat{\sigma}^2(s_t)$ and $N(s_t)$ are the variance and sample size for s_t predicted by the same ANN.

3.4 Division by Advantage

For each model, we have the option of dividing the variance term by the advantage. Such that the Loss equation looks like this:

$$L^{CLIP+VF+Var}(\theta) = \hat{E}_t[L^{CLIP} - c_1 L^{VF} + c_2 S[\pi_\theta](s_t) + c_3 \frac{\hat{\sigma}}{adv}] \text{where } adv \neq 0 \tag{12}$$

The original intent was to divide by the magnitude of the advantage $\mid adv \mid$, however an implementation error results in the models dividing by the advantage instead. The rationale for this addition is as follows. The reward variance

awareness is intended to help the model escape the local optimum by inducing drastic behaviour. This behaviour is likely not necessary or helpful when the model is converging a solution as it could push the agent off that slope. When the model has converged on a solution (which may be sub optima) we expect the magnitude of the advantage to approach, this increases the impact of the variance term. Unfortunately due to the implementation error, the model will now be discouraged from exploring reward spaces with high variance when the advantage is negative.

4 Experiments & Results

The Experiments are divided into two parts. Hyperparameter optimisation ensures the model is tuned and performing as well as possible, while the Model Comparison assesses how each model performed on the same two tasks, the PandaPushDense-v2 and PandaReachDense-v2 environments from panda-gym[19] (see Figure 4). While the case can be made for optimising the hyperparameters of each separate model on a large number of environments to ensure the model is generalisable, the main deciding factor was the lack of computational resources. We optimise the hyperparameters of the most advanced model, EVAD. before testing all other models on those hyperparameters.

PandaReachDense-v2 requires the agent to move a robotic arm to a target location whereas PandPushDense-v2 the agent to use the arm to push a cube to a target location. PandaPushDense-v2 is harder than PandaReachDense-v2 because of the potential for 'mistakes', if the robotic arm pushes the cube away from the target it is negatively rewarded. Such interaction may then discourage the agent from interacting with the cube and limit the reward it can achieve. This is an environment in which the AngoraPy implementation struggles to perform well at.

4.1 Hyperparameter Optimization

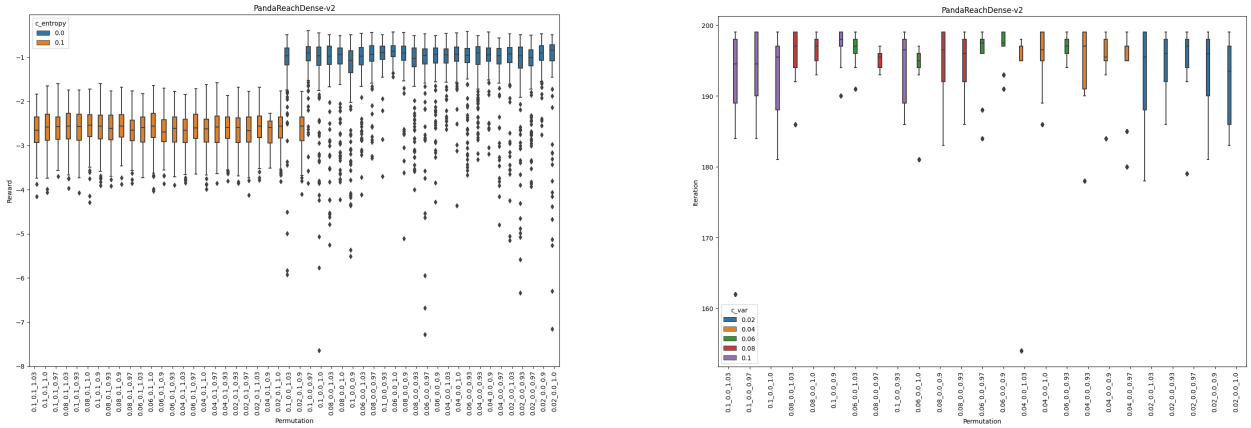
(a) PandaReachDense-v2 (b) PandaPushDense-v2

Figure 4: In (a) the robotic arm must place its clipper on the green target. In (b) the robot must push the green cube towards the green target. Images were taken from the panda-gym repository [19]

As seen in the various equation the models contain a number of coefficients and discounts that govern how much of an impact each part of the model has, these are referred to as hyperparameters and need to be tuned to ensure the model works as intended. This can only be done experimentally.

During hyperparameter optimization, we investigated different values for c_2 the coefficient of entropy in Equations 6 and 5, c_3 the coefficient of variance in Equation 6 and κ the variance discount in Equation 10 (see Figure 5). We train 10 Estimated Variance Models for every hyperparameter permutation (a total of 50) on PandaPushDense-v2 and PandaReachDense-v2 environments.

Using entropy achieved fewer rewards than without (see Figure 5a) so c_2 was set to 0. Of the models without entropy, no permutation scored a significantly higher reward than any other (see Figure 5b). κ was set to 0.99, this is the default value for λ in the AngoraPy implementation. The coefficient of variance was set to 0.06 as the models with this value showed the highest convergence stability interpreted as the shortest interquartile range for the number of training cycles before the best policy was found. A cycle is n optimisations done on the same data, in these training runs, $n = 3$.



(a) Entropy Coefficient highlight

(b) Variance Coefficient highlight

Figure 5: Averages for Reward (a) and Convergence Cycle# (b) for each permutation.

4.2 Model Comparison

Environment	Absolute	Noise	True Variance (Advantage Division)	True Variance	Estimated Variance (Advantage Division)	Estimated Variance
Panda Reach	-3.5*	2.1*	-9.3	-2.6*	-8.3	-1.7*
Panda Push	-89.6	-29.0	-35.9	-70.5	-131.3	-98.5

Table 1: Mean* difference between the Original model without entropy and the Variance Models in convergence cycle#. Negative values show the corresponding Variance model converged faster than the Original model, values with an "*" are statistically insignificant. The average cycle# was 474.0 and 230.9 for PandaReachDense-v2 and PandaPushDense-v2 respectively. Complete ANOVA and Tukey HSD in the Appendices.

Using the hyperparameters that were explored above, every model was trained 20 times without Division by Advantage. An additional 20 Estimated Variance and True Variance models were trained with Division by Advantage, while the original model was also trained 20 times with and without entropy for reference. This setup was done for both the PandaPushDense-v2 and PandaReachDense-v2 environments. No model showed a statistically significant ($p < 0.05$) difference in achieved rewards on PandaPushDense-v2 (see Figure 6). The Original model with entropy achieved lower rewards than all other models in PandaReachDense-v2 (see Figure 6).

However, statistically significant differences were noted for the number of cycles required to converge. In the PandaReachDense-v2 environment, the Variance based model all converged to the best solution faster than the original without entropy model by a significant margin (see Table 1). The fastest convergence rate is the Estimated Variance with Advantage Division model, the mean convergence cycle# 45.8% lower than the Original without entropy Model.

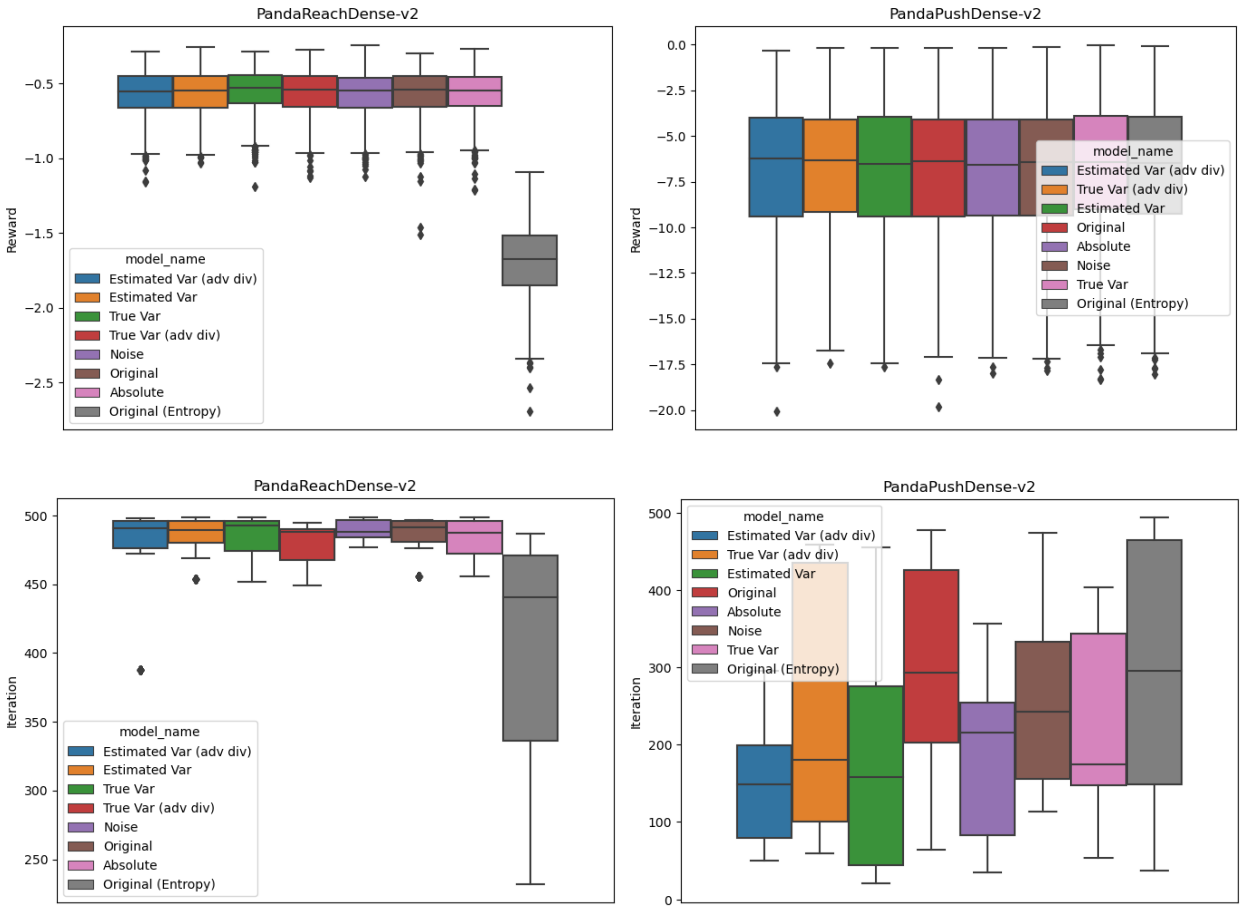


Figure 6: Reward and Convergence Cycle# by model.

5 Discussion

Reward and Convergence Time

Our models did not manage to improve the reward achieved by the original PPO model. However, the number of cycles taken to achieve similar rewards. One explanation is that the models achieve the opposite of their intended effect design while another is that the achieve a similar score with different behaviour. An Analysis of the learned behaviour might clarify the situation.

Visually inspecting the different models while they interact with the environment would take a considerable amount of time. An alternative would be to compare the actions probability distribution for a model where given the same state or even compile a list of the states encountered by each model. This would allow us to investigate the existence of different behaviours.

Additionally, having only two environments to asses the models limits the opportunities to assess the types of behaviours generated by these models. PandaReachDense-v2 is optimised by every model except the Original with Entropy. The models have no trouble learning to place the clipper on the cube as fast as possible. On the other hand, no model seems to overcome the difficulty of PandaPushDense-v2. A possibility is that the task is out of reach for both the original and variance-based models regardless of which would perform best on a task of 'intermediate' difficulty. The use of only two environments was primarily due to a lack of available computing resources and more environments paint a more reliable picture of the models' capacities.

However, unless adding variance awareness results increases the rewards achieved by the model, the use of variance as curiosity would yield no benefit compared to the increased performance and wider number of tasks models a model

can complete when augmented with other forms of curiosity [10, 13]. Whether or not this is the case is currently not obvious.

Advantage Division

A major inconvenience to this work's results is the unintended implementations of Advantage Division which was used in True Variance with Advantage Division (TVAD) and Estimated Variance with Advantage Division (EVAD). Our capacity to investigate the consequences of this difference are quite limited due to time constraint and so this section of the discussion remains hypothetical and assesses possible future work on this specific aspect of the research.

The intended behaviour for TVAD and EVAD was to divide the $varianceterm$ by $\mid adv \mid$ rather than adv, the most glaring difference is that while the size of the $varianceterm$ will increase as the model advantage function approaches zero, the direction of the approach changes the sign. If the advantage is negative and the agent is collecting lower (negative advantage) rewards then in the prior cycle then reward variance is discouraged whereas if the agent is collecting higher rewards (positive advantage) reward variance is encouraged.

Adding complexity, TVAD and EVAD behave differently compared to their respective parent algorithms True Variance (TV) and Estimated Variance (EV). EVAD converges only marginally faster better than EV in PandaPushDense-v2 (10%, p = 0.0) but importantly has a much more stable performance with an interquartile range about half that of EV (see Figure 6). Whilst the opposite is true of TVAD and TV. TVAD is less stable than TV and converges marginally later (14%, p = 0.0).

Various methods of trajectory analysis might help clarify the behaviour. Such as Principal component analysis (PCA) [20] or Potential of Heat diffusion for Affinity-based Transition Embedding [21]. Both can be used to project the value of loss functions on onto 2D graphs by dimension reduction methods to better understand the behaviour of a policy. While EVAD and TVAD were incorrectly implemented their behaviour is nonetheless worth further investigating. Comparing them to the intended implementation may also clarify the behaviour and test the original hypothesis for Advantage Division.

Absolute Value and Noise Baseline

No reward differences between any model were found further the Absolute Baseline model did not have a clear convergence difference. The Noise Baseline model converged faster than the Original (13%, p = 0.0) and Original with Entropy (17%, p = 0.0) while converging slower than other variance models. The only perceived difference between the original models and the variance-based ones is that the variance base ones converge faster, To that end they converge faster that the addition of random noise. Suggesting that while the effects may not achieve much in the way of better rewards they do distinctly reduce convergence time.

6 Conclusion

The use of reward variance as curiosity isn't doesn't seem to offer the desired improved reward, however, the change in convergence time suggests a different underlying behaviour. A correction of the Advantage Division and training on a greater number of environments is required to confirm the lacklustre rewards and potentially more stable conversion. Dimensional reduction analysis on the various loss elements and action probabilities may also help discern different behaviours between models.

7 Personal Reflection

Suggestions for another student

This section details various elements that were of note during my time working on the thesis that could be taken into account by another student working in a similar environment or by CNN Maastricht with their next Bachelor Research Thesis students. The main challenge was the code familiarisation and understanding of the technicalities of PPO implementations. I only started working on the code base after the proposal was submitted, it took some time to set up the AngoraPy on Windows and enable CUDA development tools this task was made harder by the unfamiliar node. Additionally, some concepts behind the PPO algorithm are easier to grasp when the implementation is available. Earlier exposure to the code base could make a thesis similar to this one easier.

My thesis Supervisor was the only one with access to the High-Performance Cluster used to train the algorithm. While there may be some restrictions on access to third-party hardware, the workflow would have been greatly enhanced if I

could run the training myself, catch mistakes that inevitably arose and restart the training as soon as possible rather than after analysing the data after a 24-hour training loop.

Personal Notes

The thesis was spread over 4 months during which the only time restriction was 4 deadlines and weekly meetings, time management became a challenge as there was always more work that could be done at any time of the day. Putting extra hours of work into a problem I could ask for help on come Monday's meeting was a regular occurrence. I would then take time off during a weekday, this lead to social isolation as most people are busy during the week and socialising on weekends. I eventually got a handle on the situation by setting my own schedule to stick to and switching to biweekly check-ins in the morning with my supervisor. This ensured the rest of the day was available for work while working later into the evening was a possibility. You don't usually know how long it'll take to fix a coding bug so deadlines and check-ins first thing in the morning is something I'll keep doing/ask for.

Code quality is another topic that was interesting, every project has its own metrics. Should the code be easy to modify and abide by every code convention and design pattern (operationally) or is performance the focus and readability second? In this case, I was making changes to an existing code base and trying to add my modifications without breaking anything. This was quite hard and sometimes tedious work, exacerbated by my unfamiliarity with the code base. In hindsight, refactoring the parts of the code that I intended to modify to make changes easier while maintaining existing functions would have been worth in invested time. However, doing so requires an understanding and familiarity with code that I did not have at the time. Looking for such possibilities in the early days of a future project will now be a priority.

A major challenge towards the end of the thesis when the models were trained was having to repeat the training multiple times, due to 'silent errors' where the code had mistakes that did not raise any errors. These were usually mistakes such as forgetting to take the magnitude of a value or normalising data. While catching onto these mistakes is something I will definitely improve on with experience, they can never be fully eradicated and are sometimes mentioned even in research papers. While these errors are more apparent when data is needed before a specific deadline and less of an issue when software updates are available, I am now more aware of the risk they pose.

This thesis has greatly improved my ability to write quality code while working with others and was a valuable full-time endeavour much closer to professional work than any other work required of me throughout my bachelor's education.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Mass: A Bradford Book, Mar. 1998. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[3] A. G. Barto, "Intrinsic Motivation and Reinforcement Learning," in *Intrinsically Motivated Learning in Natural and Artificial Systems*, G. Baldassarre and M. Mirolli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 17–47. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32375-1_2

[4] T. Weidler and M. Senden, "AngoraPy - Anthropomorphic Goal-Oriented Robotic Control for Neuroscientific Modeling," Mar. 2020, original-date: 2019-09-03T13:03:05Z. [Online]. Available: https://github.com/ccnmaastricht/angorapy

[5] A. Tamar, D. D. Castro, and S. Mannor, "Temporal Difference Methods for the Variance of the Reward To Go," in *Proceedings of the 30th International Conference on Machine Learning*. PMLR, May 2013, pp. 495–503, iSSN: 1938-7228. [Online]. Available: https://proceedings.mlr.press/v28/tamar13.html

[6] ——, "Learning the Variance of the Reward-To-Go," *Journal of Machine Learning Research*, vol. 17, no. 13, pp. 1–36, 2016. [Online]. Available: http://jmlr.org/papers/v17/14-335.html

[7] S. Ruder, "An overview of gradient descent optimization algorithms," Jun. 2017, arXiv:1609.04747 [cs]. [Online]. Available: http://arxiv.org/abs/1609.04747

[8] P. Jain and P. Kar, "Non-convex Optimization for Machine Learning," *Foundations and Trends® in Machine Learning*, vol. 10, no. 3-4, pp. 142–336, 2017, arXiv:1712.07897 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/1712.07897

[9] M. Danilova, P. Dvurechensky, A. Gasnikov, E. Gorbunov, S. Guminov, D. Kamzolov, and I. Shibaev, "Recent theoretical advances in non-convex optimization," 2021. [Online]. Available: https://arxiv.org/abs/2012.06188

[10] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-Scale Study of Curiosity-Driven Learning," Aug. 2018, arXiv:1808.04355 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1808.04355

[11] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3a20f62a0af1aa152670bab3c602feed-Abstract.html

[12] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by Random Network Distillation," Oct. 2018, arXiv:1810.12894 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.12894

[13] Z. D. Guo, S. Thakoor, M. Pîslar, B. A. Pires, F. Altché, C. Tallec, A. Saade, D. Calandriello, J.-B. Grill, Y. Tang, M. Valko, R. Munos, M. G. Azar, and B. Piot, "BYOL-Explore: Exploration by Bootstrapped Prediction," Jun. 2022, arXiv:2206.08332 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2206.08332

[14] O. Groth, M. Wulfmeier, G. Vezzani, V. Dasagi, T. Hertweck, R. Hafner, N. Heess, and M. Riedmiller, "Is Curiosity All You Need? On the Utility of Emergent Behaviours from Curious Exploration," Sep. 2021, arXiv:2109.08603 [cs]. [Online]. Available: http://arxiv.org/abs/2109.08603

[15] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller, "Compositional Transfer in Hierarchical Reinforcement Learning," May 2020, arXiv:1906.11228 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1906.11228

[16] A. Mavor-Parker, K. Young, C. Barry, and L. Griffin, "How to Stay Curious while avoiding Noisy TVs using Aleatoric Uncertainty Estimation," in *Proceedings of the 39th International Conference on Machine Learning*. PMLR, Jun. 2022, pp. 15 220–15 240, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v162/mavor-parker22a.html

[17] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://papers.nips.cc/paper/2017/hash/2650d6089a6d640c5e85b2b88265dc2b-Abstract.html

[18] N. Bougie and R. Ichise, "Fast and slow curiosity for high-level exploration in reinforcement learning," *Applied Intelligence*, vol. 51, no. 2, pp. 1086–1107, Feb. 2021. [Online]. Available: https://doi.org/10.1007/s10489-020-01849-3

[19] Q. Gallouédec, "panda-gym," Mar. 2023, original-date: 2020-09-13T18:13:39Z. [Online]. Available: https://github.com/qgallouedec/panda-gym

[20] A. Mohtashami, M. Jaggi, and S. Stich, “Special properties of gradient descent with large learning rates,” 2023. [Online]. Available: https://arxiv.org/abs/2205.15142

[21] S. Horoi, J. Huang, B. Rieck, G. Lajoie, G. Wolf, and S. Krishnaswamy, “Exploring the geometry and topology of neural network loss landscapes,” 2022. [Online]. Available: https://arxiv.org/abs/2102.00485

A ANOVA Test Output

A.1 Best Reward

```
ANOVA for 'best_reward' Results:
Environment: PandaReachDense-v2
F-value: 6988.760519843
p-value: 0.0

Environment: PandaPushDense-v2
F-value: 0.1867790723319655
p-value: 0.9882320431688237
```

A.2 Best Iteration

```
ANOVA for 'best_iteration' Results:
Environment: PandaReachDense-v2
F-value: 807.7503181074554
p-value: 0.0

Environment: PandaPushDense-v2
F-value: 187.37201765247957
p-value: 2.7136539694255204e-261
```

B Tukey HSD Output

group1	group2	meandiff	p-adj	lower	upper	reject
Absolute	Estimated Var	-0.0035	0.9996	-0.0240	0.0170	False
Absolute	Estimated Var (adv div)	-0.0054	0.9934	-0.0258	0.0151	False
Absolute	Noise	-0.0110	0.7292	-0.0315	0.0094	False
Absolute	Original	0.0006	1.0000	-0.0198	0.0211	False
Absolute	Original (Entropy)	-1.1305	0.0000	-1.1510	-1.1101	True
Absolute	True Var	0.0094	0.8596	-0.0110	0.0299	False
Absolute	True Var (adv div)	0.0000	1.0000	-0.0205	0.0205	False
Estimated Var	Estimated Var (adv div)	-0.0019	1.0000	-0.0223	0.0186	False
Estimated Var	Noise	-0.0075	0.9540	-0.0280	0.0130	False
Estimated Var	Original	0.0041	0.9987	-0.0163	0.0246	False
Estimated Var	Original (Entropy)	-1.1270	0.0000	-1.1475	-1.1065	True
Estimated Var	True Var	0.0129	0.5393	-0.0075	0.0334	False
Estimated Var	True Var (adv div)	0.0035	0.9996	-0.0170	0.0240	False
Estimated Var (adv div)	Noise	-0.0057	0.9909	-0.0261	0.0148	False
Estimated Var (adv div)	Original	0.0060	0.9871	-0.0145	0.0265	False
Estimated Var (adv div)	Original (Entropy)	-1.1252	0.0000	-1.1456	-1.1047	True
Estimated Var (adv div)	True Var	0.0148	0.3569	-0.0057	0.0353	False
Estimated Var (adv div)	True Var (adv div)	0.0054	0.9933	-0.0151	0.0259	False
Noise	Original	0.0117	0.6690	-0.0088	0.0321	False
Noise	Original (Entropy)	-1.1195	0.0000	-1.1400	-1.0990	True
Noise	True Var	0.0205	0.0502	-0.0000	0.0409	False
Noise	True Var (adv div)	0.0110	0.7287	-0.0094	0.0315	False
Original	Original (Entropy)	-1.1312	0.0000	-1.1516	-1.1107	True
Original	True Var	0.0088	0.8982	-0.0117	0.0293	False
Original	True Var (adv div)	-0.0006	1.0000	-0.0211	0.0199	False
Original (Entropy)	True Var	1.1400	0.0000	1.1195	1.1604	True
Original (Entropy)	True Var (adv div)	1.1305	0.0000	1.1101	1.1510	True
True Var	True Var (adv div)	-0.0094	0.8599	-0.0299	0.0111	False

Table 2: Pairwise Comparison of the mean reward in PandaReachDense-v2, the mean across all groups is -0.7051748520858095

group1	group2	meandiff	p-adj	lower	upper	reject
Absolute	Estimated Var	1.8	0.8898	-2.3197	5.9197	False
Absolute	Estimated Var (adv div)	-4.8	0.0098	-8.9197	-0.6803	True
Absolute	Noise	5.6	0.0010	1.4803	9.7197	True
Absolute	Original	3.5	0.1649	-0.6197	7.6197	False
Absolute	Original (Entropy)	-76.3	0.0000	-80.4197	-72.1803	True
Absolute	True Var	0.9	0.9979	-3.2197	5.0197	False
Absolute	True Var (adv div)	-5.8	0.0005	-9.9197	-1.6803	True
Estimated Var	Estimated Var (adv div)	-6.6	0.0000	-10.7197	-2.4803	True
Estimated Var	Noise	3.8	0.0960	-0.3197	7.9197	False
Estimated Var	Original	1.7	0.9164	-2.4197	5.8197	False
Estimated Var	Original (Entropy)	-78.1	0.0000	-82.2197	-73.9803	True
Estimated Var	True Var	-0.9	0.9979	-5.0197	3.2197	False
Estimated Var	True Var (adv div)	-7.6	0.0000	-11.7197	-3.4803	True
Estimated Var (adv div)	Noise	10.4	0.0000	6.2803	14.5197	True
Estimated Var (adv div)	Original	8.3	0.0000	4.1803	12.4197	True
Estimated Var (adv div)	Original (Entropy)	-71.5	0.0000	-75.6197	-67.3803	True
Estimated Var (adv div)	True Var	5.7	0.0007	1.5803	9.8197	True
Estimated Var (adv div)	True Var (adv div)	-1.0	0.9959	-5.1197	3.1197	False
Noise	Original	-2.1	0.7827	-6.2197	2.0197	False
Noise	Original (Entropy)	-81.9	0.0000	-86.0197	-77.7803	True
Noise	True Var	-4.7	0.0127	-8.8197	-0.5803	True
Noise	True Var (adv div)	-11.4	0.0000	-15.5197	-7.2803	True
Original	Original (Entropy)	-79.8	0.0000	-83.9197	-75.6803	True
Original	True Var	-2.6	0.5417	-6.7197	1.5197	False
Original	True Var (adv div)	-9.3	0.0000	-13.4197	-5.1803	True
Original (Entropy)	True Var	77.2	0.0000	73.0803	81.3197	True
Original (Entropy)	True Var (adv div)	70.5	0.0000	66.3803	74.6197	True
True Var	True Var (adv div)	-6.7	0.0000	-10.8197	-2.5803	True

Table 3: Pairwise Comparison of the mean convergence cycle# in PandaReachDense-v2, mean across all groups is 474.0125

group1	group2	meandiff	p-adj	lower	upper	reject
Absolute	Estimated Var	-8.9	0.6737	-24.5802	6.7802	False
Absolute	Estimated Var (adv div)	-41.7	0.0000	-57.3802	-26.0198	True
Absolute	Noise	60.6	0.0000	44.9198	76.2802	True
Absolute	Original	89.6	0.0000	73.9198	105.2802	True
Absolute	Original (Entropy)	100.4	0.0000	84.7198	116.0802	True
Absolute	True Var	19.1	0.0055	3.4198	34.7802	True
Absolute	True Var (adv div)	53.7	0.0000	38.0198	69.3802	True
Estimated Var	Estimated Var (adv div)	-32.8	0.0000	-48.4802	-17.1198	True
Estimated Var	Noise	69.5	0.0000	53.8198	85.1802	True
Estimated Var	Original	98.5	0.0000	82.8198	114.1802	True
Estimated Var	Original (Entropy)	109.3	0.0000	93.6198	124.9802	True
Estimated Var	True Var	28.0	0.0000	12.3198	43.6802	True
Estimated Var	True Var (adv div)	62.6	0.0000	46.9198	78.2802	True
Estimated Var (adv div)	Noise	102.3	0.0000	86.6198	117.9802	True
Estimated Var (adv div)	Original	131.3	0.0000	115.6198	146.9802	True
Estimated Var (adv div)	Original (Entropy)	142.1	0.0000	126.4198	157.7802	True
Estimated Var (adv div)	True Var	60.8	0.0000	45.1198	76.4802	True
Estimated Var (adv div)	True Var (adv div)	95.4	0.0000	79.7198	111.0802	True
Noise	Original	29.0	0.0000	13.3198	44.6802	True
Noise	Original (Entropy)	39.8	0.0000	24.1198	55.4802	True
Noise	True Var	-41.5	0.0000	-57.1802	-25.8198	True
Noise	True Var (adv div)	-6.9	0.8860	-22.5802	8.7802	False
Original	Original (Entropy)	10.8	0.4227	-4.8802	26.4802	False
Original	True Var	-70.5	0.0000	-86.1802	-54.8198	True
Original	True Var (adv div)	-35.9	0.0000	-51.5802	-20.2198	True
Original (Entropy)	True Var	-81.3	0.0000	-96.9802	-65.6198	True
Original (Entropy)	True Var (adv div)	-46.7	0.0000	-62.3802	-31.0198	True
True Var	True Var (adv div)	34.6	0.0000	18.9198	50.2802	True

Table 4: Pairwise Comparison of the mean convergence cycle# in PandaPushDense-v2, mean across all groups is 230.9