Universiteit van Amsterdam
Faculty of Science

# Reproducible Research Assignment

*Reinforcement Learning*

October 22, 2021

Students:
Willemijn de Hoop (11436433)
Siem Teusink (11433175)
Casper Wortmann (11913029)
Ivo Verhoeven (13013319)

TA:
Niklas Höpner
Matthew Macfarlane

# 1 Introduction and hypothesis

Within the field of reinforcement learning, Policy Gradient (PG) methods have become widely used for determining policies in continuous control tasks [5]. PGs use the gradient of the expected future reward to find the optimal from a set of policies. Both the direction and magnitude of the update is of great importance for the improvement of the policy in each step of the algorithm. Multiple approaches for choosing the appropriate update step can be distinguished.

Vanilla PG updates along the direction which improves the parameters most. To be precise, they find the direction of most improvement per unit $L2$-distance in the parameter space. However, taking a step of some arbitrary distance can have a drastic effect on the policy's output distribution. For example, consider a Gaussian policy parameterized by a mean and variance. While changing the mean leaves the distribution largely intact, a similar change in the variance alters the shape significantly. This phenomenon leads to slower and less stable learning. It can, however, be circumvented by utilizing a covariant update step. This implies the update is no longer dependent on the parameterization used. Instead, the probability space's geometry is captured and used to transform the regular PG update.

Two methods that use this principle are the Natural Policy Gradient (NPG) [6] and the Trust Region Policy Optimization (TRPO) [7] algorithms. Whereas NPG uses a fixed learning-rate at each step, TRPO derives this learning rate from a divergence based penalty term. As such, in NPG we set the learning rate, while in TRPO we set a KL-constraint (the penalty term). This allows the learning rate to be dynamically adapted while learning in TRPO.

A large amount of research has been conducted into the performance of TRPO, and many extensions have been proposed [4, 3]. Unfortunately, however, to our knowledge no thorough research has been conducted into the evolution of the learning rate in TRPO during training, nor the co-dependence of the learning rate to other variables closely connected to the update step. Therefore, in this paper, we investigate the behavior of the learning rate in TRPO. Specifically, we apply TRPO to several OpenAI Gym and Mujoco [2, 9] environments and observe how the learning rate changes at each stage of the algorithm. We investigate how the learning rate differs between environments and types of policies (discrete or continuous). In addition, we want to verify TRPO's reported ability to safely take large steps in parameter updates [7].

Concretely, we try to provide an answer to the following research questions: Can we confirm the widely accepted assumption that TRPO safely takes large steps in parameter space (Q1)? How does the learning rate of TRPO change as learning progresses (Q2)? Is the behavior of the change in the learning rate consistent over different environments, or is it environment-specific (Q3)? Answers to these questions could provide insight into the working of TRPO, which could aid to development of novel PG methods.

We hypothesize that TRPO takes steps in parameter space safely, i.e. it takes no steps that hinder the learning process while maintaining a large step-size (H1). Additionally, we expect that the algorithm produces large updates due to large learning rates early in the learning process, which decrease when approaching convergence (H2). Finally, as more exploration is needed to find a solution for more difficult problems, we expect the learning rate decrease to be slower or occur at a later stage, compared to simpler tasks (H3).

# 2 Method

## 2.1 Environments

Experiments were performed on five different environments: three continuous, and two discrete. The continuous environments are from the MuJoCo simulator [9]: Swimmer, Hopper, and Walker, in increasing order of difficulty [3]. The discrete environments, both far simpler than any of the continuous tasks, were OpenAI Gym's cart-pole balancing problem [1], and the Acrobot arm swing-

ing task [8]. All of these environments are also used in the benchmarking paper [3], which allows us to use their optimal hyperparameter set.

## 2.2 Algorithm

TRPO is a second-order gradient ascent algorithm, with numerous approximations for efficiency. The TRPO update rule is given as,

$$\Delta\theta = \eta F\left(\theta_{\text{old}}\right)^{-1} \nabla_\theta J_{\theta_{\text{old}}}\left(\theta\right) \tag{1}$$

with $\nabla_\theta J_{\theta_{\text{old}}}$ denoting the first-order policy gradient, $F$ the Fisher-information matrix, and $\eta$ the learning rate. For the Fisher information matrix, one ought to take the Hessian of the sample-average KL-divergence. Since getting the inverse matrix $F^{-1}$ is nigh intractable, TRPO uses the conjugate gradient algorithm to solve for $Fx = \nabla J$ for $x = F^{-1}\nabla J$, requiring only a matrix-vector product. For a more detailed description, we refer to the original paper by Schulman et al. [7].

To accommodate errors from approximation, a 1D line search learning rates guarantees a KL-divergence post-update below some *a priori* defined limit, $\delta_{KL}$. Specifically,

$$\eta = \alpha^i \sqrt{\frac{2\delta_{KL}}{\nabla_\theta J_{\theta_{\text{old}}}\left(\theta\right)^T F\left(\theta_{\text{old}}\right)^{-1} \nabla_\theta J_{\theta_{\text{old}}}\left(\theta\right)}} \tag{2}$$

with $\alpha^i$ being the line-search coefficient which ensures the actual KL-divergence falls within $\delta_{KL}$.

We used two implementations of TRPO. For discrete actors, we used a heavily modified version of Eluxmixor's minimal TRPO gist. For our continuous actors, tested within the MuJoCo environment, we use the publicly available RL Korea TRPO implementation, with only a few modifications. Our implementations, including plotting scripts and logging output, can be found here[1].

## 2.3 Metrics & Hyperparameters

To test our hypotheses, we run each environment in 10 different random seeds. Where possible, we use the original hyperparameters of [7]. For hyperparameters that were not specified, we used values from a benchmark paper [3], or the values specified in the source code of RL Korea (see 2.2). All hyperparameters and their source are given in Table A.1.

The most important metrics for answering our research question come from the TRPO update rule in Eq. 1:

- `Norm Grad` refers to the magnitude of the first-order gradient $\nabla_\theta J_{\theta_{old}}(\theta)$
- `Norm SearchDir` denotes the norm of the direction of the parameter update, and is defined by the $\|F(\theta_{old})^{-1}\nabla_\theta J_{\theta_{old}}(\theta)\|$
- `Effective Learning Rate` refers to the size of the parameter update, depicted as $\eta$ in Eq. 2. This metric will show the variability of the learning rate over time. We min-max scale the learning rate to allow comparison across environments.
- `Norm Update` is the total size of the update, the norm of the model update as a function of the number of steps. It is the multiplication of the `Effective Learning Rate` and the `Norm SearchDir`

Beyond these, we track a myriad of other metrics that indicate TRPO's learning behavior. For a complete overview we refer to Sec. A.

[1] https://github.com/IvoOVerhoeven/UvA_RL_2021_Reproducible_Lab

# 3 Results

The aforementioned metrics are depicted as a function of the number of training iterations in Fig. 4.1, succinctly summarizing our main results.

For all environments but the CartPole problem, we observe that the effective learning rate (Fig. 4.1c) initially increases for a few steps, which is immediately followed by a downward trend. After this downward trend, the learning rate seems to stabilize. Note, however, that CartPole converges within a fraction of the steps required for the other problems (see Fig. B.1). When looking at the norm of the model update, (Fig. 4.1d) we observe a spike in the beginning followed by a decreasing slope until the end of training for all environments. As for the norm of the gradient (Fig 4.1a), we see that this metric moves about a constant for the discrete environments, while it has an increasing slope for the entire learning process of the continuous problems. Finally, as displayed in Fig 4.1b, the norm of the search direction shows contrasting behavior between the discrete and continuous setup. In the former, it shows a curve that decreases quickly after initialization, whereas in the latter, the curve quickly increases in the first iterations. After this, both the discrete and continuous environments plateau.

When comparing all metrics in the continuous setup, it can be noted that Swimmer and Hopper show a similar trend for all metrics. Walker, the most difficult problem, behaves slightly differently, with higher values for the search direction and a curve that decreases at a later stage than the others. Furthermore, while decreasing, it does so at a faster pace. The discrete problems behave differently in the first iterations, when compared to continuous. Both show more similar curves as training continues.

For all environments, we observed that the KL-divergence between policies never exceeds the set limit, and that the mean reward consistently increases over 500 iterations. The complete mean reward graph can be found in B.

Finally, we observe an inverse relation between the search direction and the learning rate. This can be seen in the plots in Figure 4.1b and c, where, as the learning rate goes down, the search direction goes up. This relation is further highlighted by the scatter plots of Fig. B.5.

# 4 Discussion & Conclusion

While plenty of research has been conducted into the performance of TRPO and its variants, to our knowledge, no empirical investigation into the evolution of the actual step size during training has been published. In light of this, we have analyzed the gradients and learning rates during various phases of learning, across several distinct environments and policy parameterizations.

TRPO's reported ability to safely take large steps can be validated. We enforce our algorithm to take the biggest possible step in the search direction, under the constraint that the KL-divergence never exceeds the provided limit. Despite this, for all environments, we observe fast and stable improvement of the mean reward. The stability of learning is further corroborated by the consistent trend of `Norm Update`. Thus, TRPO balances fast progress with keeping the policy robust under incorporation of new updates.

Concerning the question of how the learning rate of TRPO changes as learning progresses, we draw several conclusions. The change of learning rate over time follows a similar trend for four of the five tested environments: the learning rate decreases as the number of iterations increases. CartPole converges so quickly, that comparison proves difficult. Thus, our hypothesis is confirmed: the algorithm typically produces large learning-rates early in the learning process, and these decrease as the policy tends towards optimal.

Beyond a relationship as iterations progress, we further observe an inverse relation between the learning rate and search direction. As the search direction norm increases, the learning rate, in turn, decreases. The result of this is that the actual update is clamped within a smaller region than either the gradient or search direction would suggest. Thus, we interpret the learning rate's

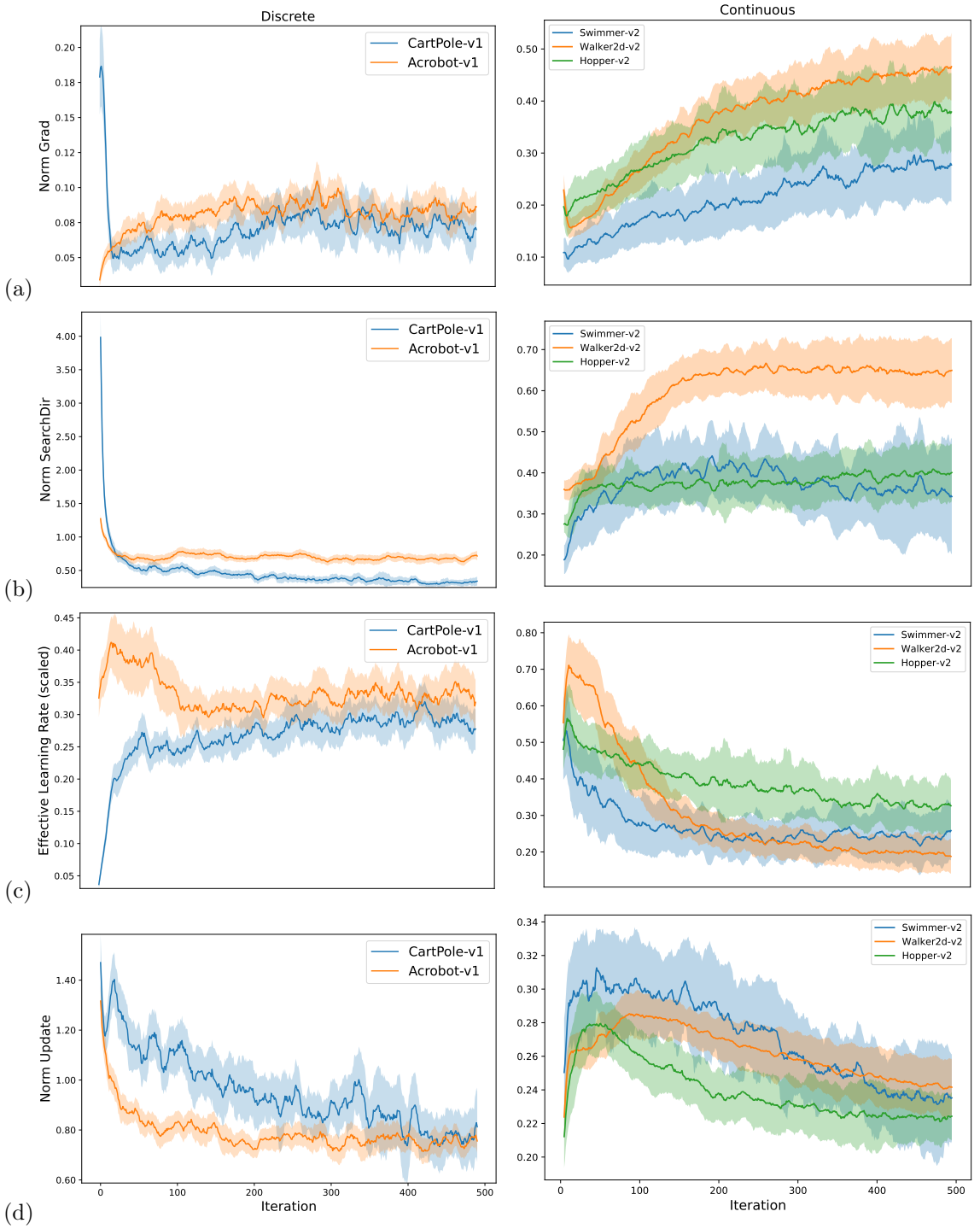


Figure 4.1: Performance metrics as defined in 2.3. (a) shows the `Norm Grad`, the norm of the first-order gradient, (b) the `Norm SearchDir`, the direction of the gradient after multiplication with the stepsize, (c) the `Effective Learning Rate`, and (d) `Norm Update`, the norm of the model update as a function of the number of steps. Dark lines give the median over all runs, mean-smoothed over 10 iterations. The shaded area gives ± 1 standard error. Where applicable, the y-axis is min-max scaled to allow comparison across environments. The left column gives the discrete actor and the right column the Gaussian actor.

function to be to dampen the search direction's suggested step size, especially as the algorithms start making incremental gains in reward. It compensates for small gradients early on in training,

and moderates the updates in later stages.

Regarding the third research question, we conclude that the behavior of the learning rate is largely dependent on the training environment. While most metrics exhibit similar trends across tasks, changes in the curves occur at different points or with different degree. For tasks whose mean reward increase plateaus quickly, e.g. CartPole, Acrobot and Swimmer, the effective learning and update norm also flatten. Contrasted to these, a harder environment like Walker shows a slower decreasing update step size. Yet, even in these simpler environments, the differences within metrics can be striking.

All in all, while patterns between metrics and environments can be seen, these are more dependent on the environment and proximity to convergence than anything else.

It should be noted that our experiments have been conducted under a fixed set of hyperparameters that we have taken from several sources in an attempt for the algorithm to run optimally. However, to make our conclusion more robust, more experiments should be conducted in the future for a variety of combinations of hyperparameters.

A further limitation is using two different implementations. While a necessity due to time and Mujoco interfacing constraints, this could invalidate comparison between the discrete and continuous environments. However, due to strong performance in either situation, we believe this to have been mitigated.

# References


[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13 (1983), pp. 834–846.

[2] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[3] Yan Duan et al. *Benchmarking Deep Reinforcement Learning for Continuous Control*. 2016. arXiv: 1604.06778 [cs.LG].

[4] Peter Henderson et al. "Deep reinforcement learning that matters". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[5] Riashat Islam et al. "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control". In: (Aug. 2017).

[6] Sham M Kakade. "A Natural Policy Gradient". In: *Advances in Neural Information Processing Systems* 14 (Nov. 2017). DOI: https://repository.upenn.edu/statistics_papers/471.

[7] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 1889–1897.

[8] Richard S Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems* (1996), pp. 1038–1044.

[9] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

# A Metrics & Hyperparameters

## A.1 Hyperparameters

We have taken the hyperparameters from the papers of the respective techniques [7], a benchmark paper [3], and filled up the missing parameters with the values in the source code of our adopted implementation. This creates a fair comparison between the performance of both techniques because we try to run them both in an optimal fashion. The hyperparameters are displayed in Table A.1

Table A.1: Overview of hyperparameters used.

| Parameter | CartPole-v1 | Acrobot-v1 | Swimmer-v2 | Walker-v2 | Hopper-v2 |
|---|---|---|---|---|---|
| KL constraint ($\delta_{ql}$) | $1 \cdot 10^{-2}$ | $1 \cdot 10^{-2}$ | $1 \cdot 10^{-2}$ | $1 \cdot 10^{-2}$ | $1 \cdot 10^{-2}$ |
| Hidden-layer size (actor) | 32 | 32 | 30 | 50 | 50 |
| Hidden-layer size (critic) | 32 | 32 | 30 | 50 | 50 |
| Learning rate (critic) | $5 \cdot 10^{-3}$ | $5 \cdot 10^{-3}$ | $1 \cdot 10^{-3}$ | $1 \cdot 10^{-3}$ | $1 \cdot 10^{-3}$ |
| Line search factor ($\alpha$)* | 0.9 | 0.9 | 0.5 | 0.5 | 0.5 |
| Line search steps* | 10 | 10 | 10 | 10 | 10 |
| Min. Samples per Episode* | 32 | 32 | 2048 | 2048 | 2048 |
| Max. Steps per Sample* | 200 | 200 | 10000 | 10000 | 10000 |
| Batch size | 32 | 32 | 64 | 64 | 64 |
| Episodes | 500 | 500 | 500 | 500 | 500 |
| Discount factor $\gamma$ | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| GAE $\lambda$ | 1.00 | 1.00 | 0.98 | 0.98 | 0.98 |

## A.2 Metrics

**Performance** To understand the performance of the algorithm on the provided task, we track the mean reward per batch. Each environment has a different reward range, (e.g. [-500,0] for Acrobot-v1, and [0,$\infty$) for a continuous task like Walker-v2). As such, we minmax scale the rewards to [0,1], based on the maximum and minimum achieved over all runs. This allows comparing across different environments, with the speed of reaching the maximum reward being of greater importance than the actual achieved value (in this instance).

**Step-size** Likely most relevant for our research is the update step size, defined as $\eta$ in Eq. 2. To get a well-rounded insight into the behavior of the learning rate, we report four different metrics; i) the maximum step size, i.e. Eq. 2 with $i = 0$, ii) the line search coefficient, $\alpha^i$, with $i$ being the number of line search steps and $\alpha$ the attenuation factor, iii) the effective learning rate, 2, and iv) the actual KL-divergence per iteration after update.

**Gradients & Convergence** To track how close an algorithm is to convergence during learning, we use three metrics. The first two measure the size of gradients; the Frobenius vector-norm[2] of the i) first-order gradient of the expected return, $\nabla J_{\theta_{\text{old}}}$, and the ii) the full TRPO update, $\Delta\theta$. Both *should* decline in magnitude as the algorithm approaches convergence.

The third metric of relevance, applicable only to the discrete actors, is the average entropy of the produced policies. As the policies approach optimality, they will become more deterministic for the majority of states. This would correspond to a gradually decreasing entropy. We do not report this for the continuous policies, as these are parameterized as Gaussians with constant variance (thus, the entropy remains constant).

[2]We flatten the matrices to long vectors

# B Additional Figures

## B.1 Metrics by Training Iterations

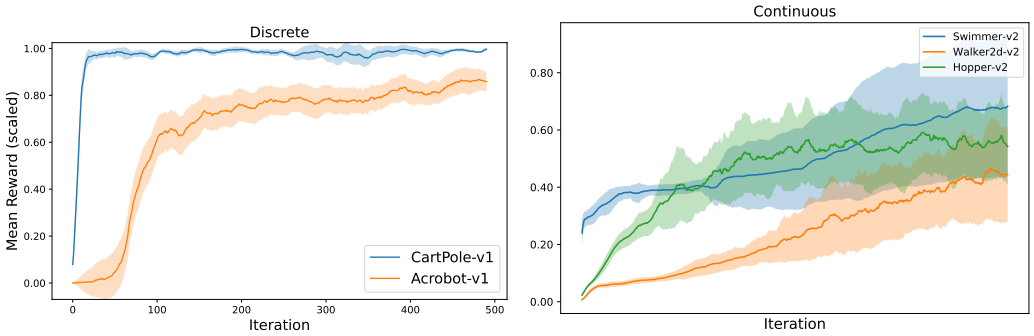


Figure B.1: The mean reward per iteration. Averaged over 10 independent runs and smoothed by taking the average over 10 iterations. Shaded area indicates ± one standard error. To allow for comparison across environments, the units are min/max scaled to range $[0, 1]$.

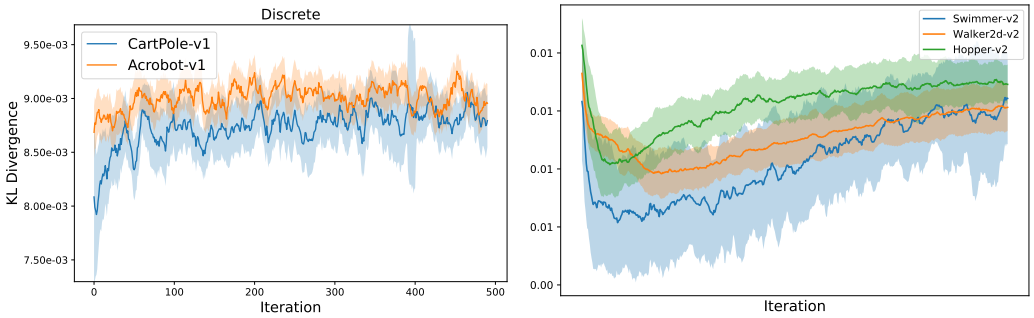


Figure B.2: The KL-divergence per iteration. Averaged over 10 independent runs and smoothed by taking the average over 10 iterations. Shaded area indicates ± one standard error.

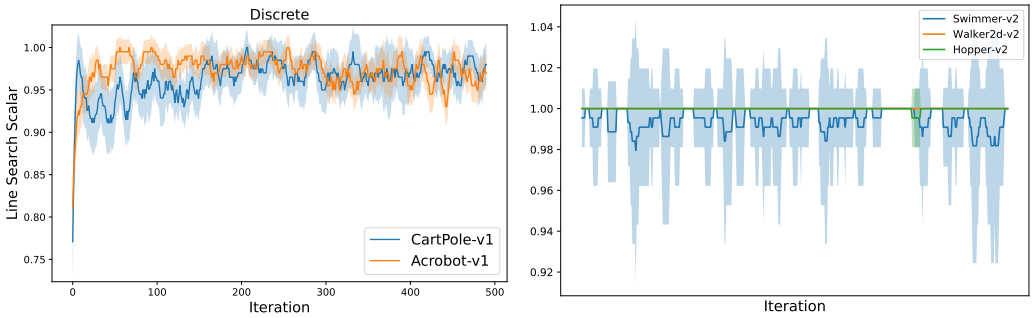


Figure B.3: The linear search scalar, $\alpha^i$, per iteration. Averaged over 10 independent runs and smoothed by taking the average over 10 iterations.

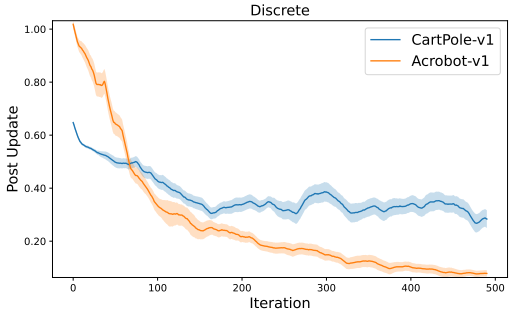


Figure B.4: The mean batch entropy (nats) of the discrete policies post-update, per iteration. Averaged over 10 independent runs and smoothed by taking the average over 10 iterations.

## B.2 Pairplot

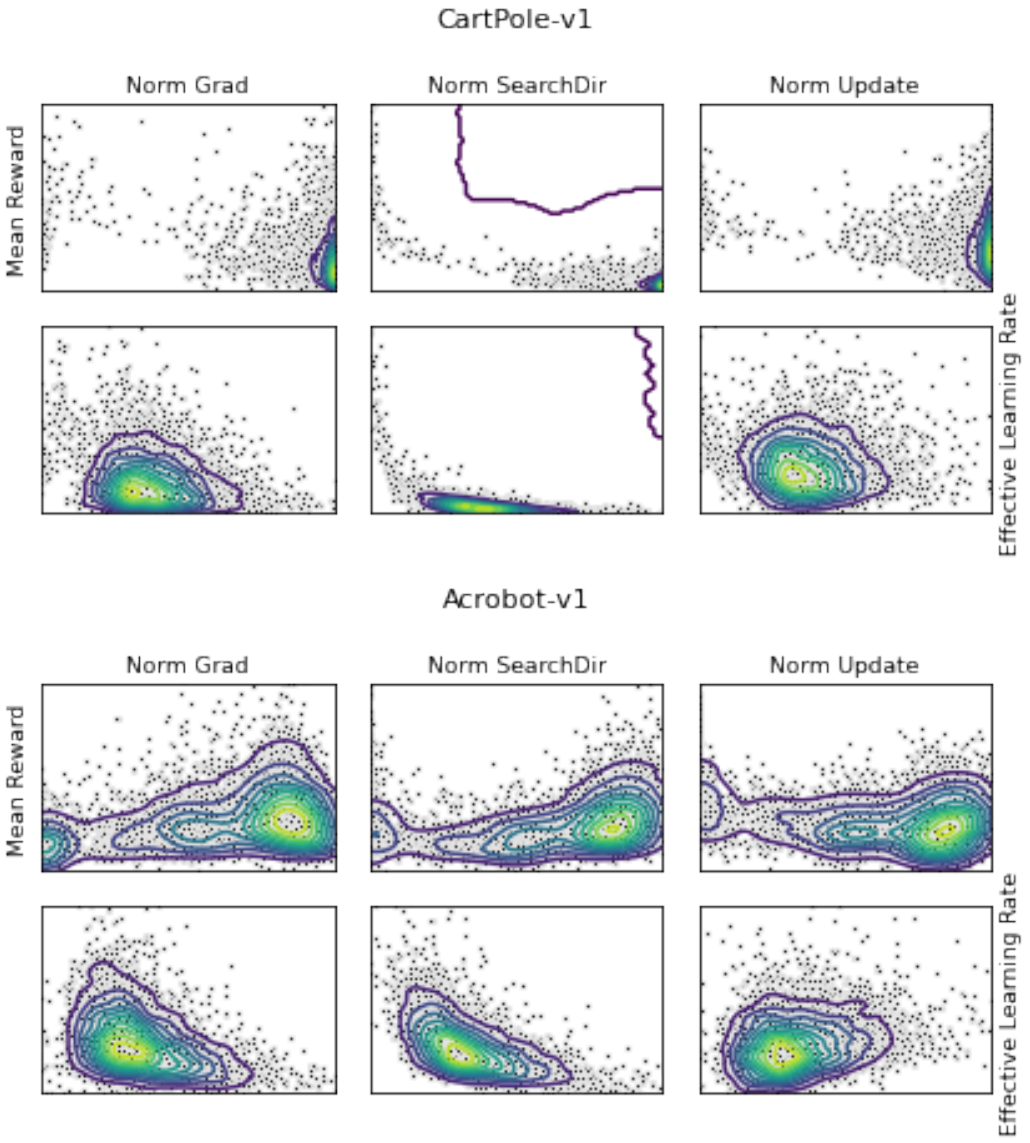


Figure B.5: Pairplots of interesting quantities. Top gives the situation for CartPole-v1, bottom for Acrobot-v1. Lines denote density, with brighter colors indicating higher density and vice versa. All units are min-max scaled.