Project for Autonomous and Adaptive Systems

Samuele Bortolato
Department of Computer Science and Engineering - DISI
University of Bologna
Bologna BO 40136
samuele.bortolato@studio.unibo.it

Abstract

The aim of this project is to experiment with continuous state-action spaces. This work mainly focuses on the interaction between PPO, experience replay and hentropy regularization in continuous action spaces.

1 Introduction

While most of the literature in Reinforcement Learning focuses on discrete action spaces, continuous action spaces have the advantage of having the possibility to achieve an infinitely fine-grained control over the action while maintaining a relatively low dimensionality. On the other hand continuous action spaces introduce other challenges, ranging from ensuring convergence with infinitely many different possible states and actions to balancing exploration and exploitation in such a space, requiring careful evaluation.

The goal of this project was to experiment with reinforcement learning techniques in continuous state-action spaces. This work I try different ideas taken from state of the art results in model free policy gradient methods and tests how they can work together, testing the algorithms on a simple environment with continuous low dimensional state-action spaces with sparse rewards. Specifically in this work I started from a Proximal Policy Optimization (PPO) [6] clip style algorithm and in order to encourage exploration and limit early convergence I introduced a new kind of regularization to solve the instability of the entropy regularizer in continuous action spaces. I also experimented with SAC [1] style regularization adding the entropy bonus directly inside the reward. I then tested the viability of adding experience replay to further increase the sample efficiency of the algorithm. I took inspiration by Actor-Critic with Experience Replay (ACER) [10] for the estimators of the gradient and for the network architecture, while substituting the TRPO [5] style update with a PPO clip one, and removing the stochastic part form the Stocastic Dueling Network used for the V and Q value functions. I also tested different architectures and features to make the neural network more suited to deal with coordinates as inputs.

2 System description

2.1 Environment

In order to be able to test, study and compare the algorithms in a continuous state-action space I needed a simple low dimensional environment that would allow me to plot and visualize the the policies and value functions.

The environment I settled with is a simple 2D space simulation where a space ship (controlled by the agent) has to reach a goal while avoiding to being sucked inside the black hole.

The state is represented as the coordinates of the the ship, the goal and the black hole, all of which have coordinates in [-1,1]. The ship is controlled directly in velocity by the agent (there is no inertia)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

and legal actions are also bounded in the range [-1,1]. If the ship tries to exit the designed space the simulation simply ignores the component of the velocity perpendicular to the border and the ship slides along the border. The blackhole pulls the ship with a force that is proportional to the inverse of the distance. For the reward function I chose to only give a +1 reward if the goal is reached and a -1 if the goal is not reached. The goal and the blackhole are considered to be reached if the ship is inside a radius around the coordinates of its center. For the blackhole the radious is equal to the distance at which the force is enough to pull the ship all the way to it's center. A timeout is set to reset the simulations after a given number of steps.

2.2 Agent

Aiming at creating an agent for a continuous action-space, policy gradient methods where the obvious choice. I settled with an actor-critic architecture.

The base network is a quite simple MLP network with relu activations and resnet like skip connections every two layers.

For the actor I chose to output a normal random variable and later squashing the sampled action through a tanh to enforce the action bounds. This required to adjust the probabilities to account for the change of variables (see SAC [1] appendix C). The means for the Gaussian distribution where the simple outputs of the network. To ensure the positive definiteness the covariance matrix V was obtained through the correlation matrix C and the standard deviations σ with the formula

$$V = diag(\sigma) \times C \times diag(\sigma)$$

where C is constructed by making the network output the values for the lower triangle and passing them through a tanh to enforce the range $[-1, 1]$, and making the matrix simmetric and with 1s on the diagonal, while the standard deviations σ just need to be enforced positive with an exponential or a softplus. It's important to ensure that the policy doesn't become completely deterministic to avoid inf and nan, and also that the out of diagonal values for the correlation matrix are actually in the range $[-1 + \epsilon, 1 - \epsilon]$ to ensure the positive definiteness constraint for the sampler (positive semi-definite is not enough).

The critic instead was inspired by Dueling Networks [9] and ACER [10], and aims at predicting both the state value function $V(s_t)$ and the advantage $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$. Since we are in a continuous action space outputting an advantage for every possible value was not possible, so I instead passed the action as input as in ACER, giving the information as second input after the network had already predicted $V(s_t)$ in order to guarantee the independence of that prediction from the action. See Appendix D for a visualization. ACER opted for a stochastic Q network in order to enforce the two estimates to be coherent. I instead chose to remove the stochasticity and simply train them toward the same target.

Taking inspiration from the DreamerV3 [2] I also tested the effect of discretizing the output of the value value functions through two-hot encoding and training it as a classification problem. Using symlog activations in this environment doesn't have much effect since we are not varying the scale of the rewards, which are fixed at +1 and -1, but for completeness I added it when testing the two-hot encoding.

Finally, testing the ability of the agent in learning the reward function I realized that the critic was struggling to learn it properly, mainly due to the fact that the chosen network was intrinsically a piece-wise linear function and the reward was a function of the distance between the coordinates. In the literature the most common solution to this problem is creating new features tailored for the task at hand. Among the possible solutions the one that worked the best was the products between the coordinates as input, making the neural network effectively a piece-wise second order polynomial. Since the square of the L2 norm is a quadratic function with respect to the coordinates, with the new features distinguishing distances becomes a linear function. I further testing passing directly also the distances between the points, which should be superfluous given the previous features, but could potentially speed up the training. Using higher order polynomials can be unstable if the inputs are not constrained between -1 and 1, and is often a good idea to apply L2 regularization or weight decay.

2.3 PPO

I started this project by reproducing the PPO algorithm. Introduced by Schulman et al. [6] Proximal Policy Optimization is popular policy gradient algorithm designed to optimize policies in a safe and efficient manner.

PPO is an on policy algorithm that allows for multiple training epochs for each batch of data by constraining the policy not to vary too much from the generating policy, similarly in what is done in TRPO [5]. The clip version, also known as PPO-clip, introduces a surrogate training objective

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

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio. The first term inside the min is

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$$

is the conservative policy iteration objective [3] that TRPO maximizes. The second term modifies the objective clipping the value around the identity, removing the incentive for moving $\pi_\theta(a_t|s_t)$ too far away from $\pi_{\theta_{old}}(a_t|s_t)$. Taking the minimum of the clipped and unclipped objective ensures that L^{CLIP} is a lower bound for L^{CPI}, effectively ignoring the probability changes only when they would improve the objective, and considering them when they make it worse. This allows to recover from an update that worsened the objective.

PPO is a quite simple yet effective algorithm, and a solid starting point.

There were multiple choices for computing the advantages $\hat{A}_t$, I settled with Generalized Advantage Estimation [7]

$$GAE(\gamma, \lambda)) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the TD residual of V with discount γ [8], from which is also straightforward to compute the $TD(\lambda)$ for the targets of the value functions

$$TD_t(\lambda) = GAE_t(\gamma, \lambda)) + V(s_t)$$

For simplicity I ran n simulation in parallel for t steps and later compute the targets for the segment and train for some epochs. Running the simulations asynchronously with respect to the training would have been more efficient at the cost of not being perfectly on policy, forcing me to account for the off-policyness at this early stage.

In the original paper they also used an entropy bonus for the discrete action spaces to incentivize exploration

$$actor\_loss = -L^{CLIP} - H(\pi(\cdot|s_t))$$

In a continuous action space on the other hand it's uncommon to have a close form formula for the entropy, and often has to be approximated by sampling

$$H(\pi(\cdot|s_t)) = \mathbb{E}_{a\sim\pi}[-log(\pi(a|s_t))]$$

The problem with adding this approximation as a regularization to the loss is that eventual actions with low probability cause extremely big gradients that make the training unstable. Furthermore, while the entropy of a discrete random variable is positive since the probability of the single actions is bounded between 0 and 1, in the continuous domain the probability density function is not bounded, causing the approximation to have a relatively small gradient for actions with very high probability.

A possible solution is to use as regularizer the expected action probability, which is characterized by a constant derivative and does not present instability

$$actor\_loss = -L^{CLIP} + \mathbb{E}_{a\sim\pi}[\pi(a|s_t)]$$

In my experiments training without any sort of regularizer caused early convergence to a policy too deterministic, resulting in inf and nan during training. On the other hand training with the entropy approximation as a regularizer in the continuous case was still extremely unstable, and most of the time didn't finish the training.

2.4 SAC

Another recent development aiming at dealing with the early convergence of the policy network is Soft Actor Critic, Haarnoja et al. [1]. SAC is based on the maximum entropy framework, where the policy maximizes both the expected return and the expected entropy of the policy.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{[s_t,a_t]}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))]$$

In their work they propose a soft policy iteration where they include the entropy term inside the value function by defining

$$T^{\pi}Q(s_t,a_t) = r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}}[V(s_{t+1})]$$
$$V(s_{t+1}) = \mathbb{E}_{a\sim\pi}[Q(s_t,a_t) - log(\pi(a_t|s_t))]$$

I tested the viability of adding such a regularization inside PPO. PPO requires to be able to estimate the advantage of the taken action, the formulation used in SAC is not directly appicable due to the fact that $V(s_{t+1})$ is not the expected value of $Q(s_t,a_t)$ over the possible actions. For the same reason training the dueling network would be more complex to define the targets and maintain the estimates coherent. So instead I chose to move add the Information of an action to the reward of the Q network, making the $V(s_{t+1})$ the expected value of the $Q(s_t,a_t)$

$$T^{\pi}Q(s_t,a_t) = r(s_t,a_t) - log(\pi(a_t|s_t)) + \gamma \mathbb{E}_{s_{t+1}}[V(s_{t+1})]$$
$$V(s_{t+1}) = \mathbb{E}_{a\sim\pi}[Q(s_t,a_t)]$$

where it's triviall to show that the estimate for the value functions remained unchanged with respect to SAC

$$V(s_{t+1}) = \mathbb{E}_{a\sim\pi}[Q(s_t,a_t)] = \mathbb{E}_{a\sim\pi}[r(s_t,a_t) - log(\pi(a_t|s_t)) + \gamma \mathbb{E}_{s_{t+1}}[V(s_{t+1})]]$$

By defining the information augmented reward as

$$r_{\pi}(s_t,a_t) = r(s_t,a_t) - log(\pi(a_t|s_t)) = r(s_t,a_t) + I_{\pi}(a_t|s_t))$$

and have the same convergence results of SAC (see appendix B.1 in SAC [1]).

This new formulation can be inserted in other estimators (like GAE and $TD(\lambda)$ used in PPO) by simply substituting the reward with the information augmented reward.

2.5 ACER

Reducing as much as possible the number of simulation steps required to train an agent is of vital importance to make Reinforcement Learning viable in many domains, and it's necessary to learn as much as possible from the experience we already have. PPO is designed to allow multiple gradient steps for a single batch of experience. At the same time the states are highly correlated and multiple parallel simulations are needed to ensure a good coverage of the state space.

Another important technique to reduce the sample complexity is Experience Replay, where the agent is trained both on the online experience and on replayed experiences sampled from a buffer. Can PPO be combined with Experience Replay?

The main challenge with Experience Replay is that training the network on old experiences is basically an off policy training, because the old experience was obtained with a policy that is no longer exactly the same as the current one.

The common solution of using importance sampling to account for the off-policy training suffers form the fact that the variance of such estimator is infinite, making it quite unstable to train. One approach to solve this issue is to clip the importance sampling ratio to prevent it from exploding, in Actor Critic with Experience Replay, Wang et al. [10] they further corrected the bias in the estimate due to the clipping. Taking inspiration from them I computed the estimates for the Q values using $Retrace(\lambda)$ [4] and used their bias corrected estimate for the policy gradient

$$\hat{g}_t^{acer} = \overline{\rho}_t \nabla_{\pi_\theta} log(\pi_\theta(a_t,s_t))[Q_{ret}(s_t,a_t) - V_\theta(s_t)]$$
$$+\mathbb{E}_{a\sim\pi}\left(\left[1 - \frac{c}{\rho_t}\right]_+ \nabla_{\pi_\theta} log(\pi_\theta(a,s_t))[Q_\theta(s_t,a) - V_\theta(s_t)]\right)$$

For the estimate of the value function instead I substituted the $V_\theta(s_t)$ with $Q_\theta(s_t)$. This is due to the fact that I didn't use the stochastic dueling networks they proposed and I had to ensure the two networks gave coherent estimates. Finally I substituted the TRPO [5] update used in ACER with a PPO [6] update, maximizing the following objective

$$L^{ACER-PPO} = \mathbb{E}_\mu \left[min(\frac{\pi'(a_t|s_t)}{\mu(a_t|s_t)}, c) \cdot L^{CLIP} \left(\frac{\pi(a_t|s_t)}{\pi'(a_t|s_t)},\ [Q_{ret}(s_t, a_t) - V_\theta(s_t)],\ \epsilon \right) \right]$$
$$+ \mathbb{E}_\pi \left[\left[1 - \frac{c \cdot \mu(a|s)}{\pi'(a|s)} \right]_+ \cdot L^{CLIP} \left(\frac{\pi(a|s_t)}{\pi'(a|s_t)},\ [Q_\theta(s_t, a) - V_\theta(s_t)],\ \epsilon \right) \right]$$

where

$$L^{CLIP}(r_t, A_t, \epsilon) = min(r_t \cdot A_t, clip(r_t, 1-\epsilon, 1+\epsilon) \cdot A_t)$$

and where π' is the exponential moving average of π and μ is the policy used to produce the experience. Derivation of the objective can be found at Appendix A.

Since the Dueling Network doesn't have the stochastic part as in ACER, the target for the value function becomes instead

$$V^{target}(x_t) = \mathbb{E}_\mu \left[min \left\{ c, \frac{\pi'(a_t|s_t)}{\mu(a_t|s_t)} \right\} (Q_{ret}(s_t, a_t) - Q_\theta(s_t, a_t)) \right] + \mathbb{E}_{a \sim \pi'}[Q_\theta(s_t, a)]$$

to maintain the consistency between Q and V estimates.

3 Experimental setup and results

For my experiments I ran 128 simulations in parallel for $2^{17} = 131,072$ step for a total of $16,777,216$ simulation steps. I batched the experience and ran the optimization in blocks of 32 steps. I slightly discounted the rewards with a $\gamma = 0.98$, and set $\lambda = 0.8$ for $GAE(\lambda)$, $TD(\lambda)$ and $Retrace(\lambda)$. The network was an MLP with 5 layers of 512 hidden neurons, with skip connections every two, with on top an head depending on the network: 5 outputs for the actor and 1 output for the value networks (unless I was testing the two-hot encoding). The actor and the V network take as input the coordinates and the additional features, while the Q network takes in input the last hidden layer of the V network and the action. For the optimizer I found that training the actor with a lower learning rate increased the stability, so I used AdamW with lr=3e-4 for the value networks and lr=3e-5 for the actor. I also added a small L2 regularization (factor 1e-5) to the output of the actor to avoid problems when sampling from the Gaussian distribution. The replay buffer store a maximum of 1024 training steps (32,768 environment steps) and are not prioritized. All the experiments where run with random seeds 0,1 and 2, and in the following graphs are shown the averages over the runs. The plots of the reward are smoothed taking the average of 100 training steps.

In the first experiment I compared the results with PPO with a discrete action space using the entropy regularization proposed in the original paper against the same with the probability regularization I proposed. For the continuous case there are only the results with the latter because the approximation of the entropy was too unstable and would throw errors due to infs and nans at early stages. From the graphs (fig. 1) we can see that in the discrete case both the normalization using the entropy and using the probability are quite stable and the training behaves similarly reaching the same final level of performance, with the normalization through the entropy training a bit faster. On the other hand in the discrete case the normalization of with the prob still trains smoothly but the final performance of the agent is lower compared to the discrete case. This could be either due to the continuous version of the network being less flexible (it's trying to learn a multivariate Gaussian) or due to the regularization being too strong in the continuous case and it's stopping the network from becoming too deterministic.

In the second experiment I focused on the networks and tested using the two-hot encoding compared to a direct regression and using the base input (only the coordinates) against augmenting it with the squares of the coordinates and with the final complete features where I also added the distances

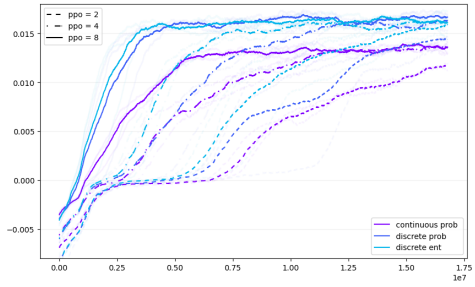


Figure 1: Reward training with either entropy or probability regularization, discrete or continuous action space

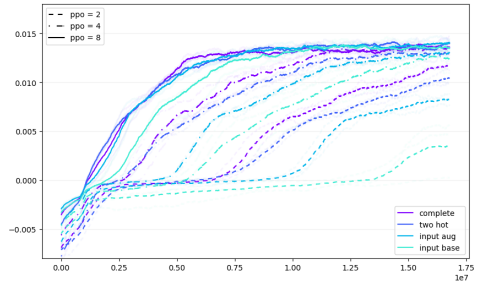
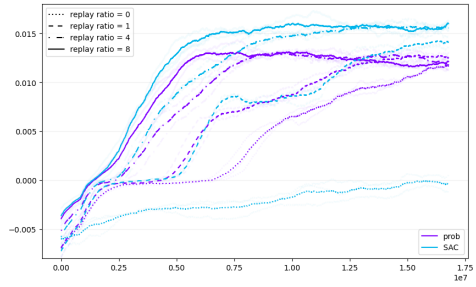
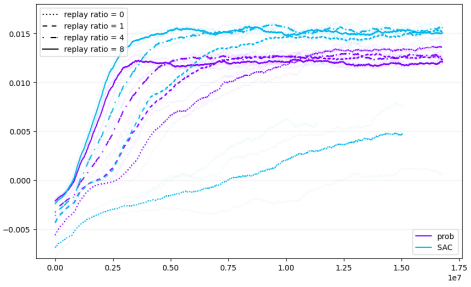


Figure 2: Reward training with different networks

among the points. From the results (fig. 2) can be seen that using the two-hot formulation for the regression doesn't really change the results, rather it slightly slows down the convergence, so I didn't include it in the following tests. For the choice of the feature set instead can be seen that using a more suitable feature set helps the agent learn faster, nevertheless also the agent with the base feature set (only the raw coordinates of the objects) managed to reach the same level of performance with more training. The difference is more negligible with more PPO epochs of training.



PPO=2



PPO=4

Figure 3: Reward training with different replay rations using 2 and 4 PPO epochs. Tested both probability regularization and SAC regularization

Finally I tested including the regularization directly inside the value function as in SAC and tested the proposed experience replay with different number of PPO updates and replay ratios (fig. 3). As expected adding experience replay increased the sample efficiency of the algorithm increasing the performance monotonically with the replay ratio. Since the computational complexity of the algorithm is proportional both to the PPO epoch and the replay ratio (+1) the most efficient implementation would have to find a balance between the two tuning based on the problem. Interestingly the usage of the proposed SAC regularization outperformed the previous probability regularization when combined with the experience replay, being also able to reach an higher final level of performance comparable with the discrete network. On the other hand in the base PPO settings it performed way worse than the prob regularizer, training slower and being more unstable to train. The cause of this behavior is still unclear.

4 Conclusion

In this work I implemented form scratch PPO-clip and tested possible ways to regularize it's training and combine it with experience replay, testing it on a simple low dimensional continuous environment with sparse rewards created for this project. The two proposed regularizers obtained both stable training, with the SAC based one obtaining the best results when paired with experience replay. Further investigation would be needed to explain the slight instability when training the SAC regularizer without experience replay. The proposed experience replay is quite stable and monotonically increases the sample efficiency when increasing the replay ratio.

5 Code

The code for this project can be found at https://github.com/samuele-bortolato/RL-project

References

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

[2] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

[3] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning, 2002.

[4] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning, 2016.

[5] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL `http://arxiv.org/abs/1502.05477`.

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

[7] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[9] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL `http://arxiv.org/abs/1511.06581`.

[10] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL `http://arxiv.org/abs/1611.01224`.

A ACER-PPO Objective

A.1 Off-policy PPO

In normal off policy learning we correct the gradient through importance sampling

$$\nabla_\theta \mathbb{E}_\pi \left[log(\pi(a_t|s_t)) A_t(s_t,a_t) \right] = \nabla_\theta \mathbb{E}_\mu \left[\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} log(\pi(a_t|s_t)) A_t(s_t,a_t) \right]$$

which if we don't derive throught the advantage becomes

$$= \mathbb{E}_\mu \left[\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\nabla_\theta \pi(a_t|s_t)}{\pi(a_t|s_t)} A_t(s_t,a_t) \right] = \mathbb{E}_\mu \left[\frac{\nabla_\theta \pi(a_t|s_t)}{\mu(a_t|s_t)} A_t(s_t,a_t) \right]$$

With PPO-clip we want to instead substitute the

$$\nabla log(\pi(a_t|s_t)) = \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)}$$

with the ratio between the actual policy and the reference policy, in order to later clip it to stop the gradient

$$\frac{\nabla \pi(a_t|s_t)}{\pi'(a_t|s_t)}$$

The off policy PPO would then maximize the objective

$$\mathbb{E}_\mu\left[\frac{\pi'(a_t|s_t)}{\mu(a_t|s_t)}\cdot min(\frac{\nabla\pi(a_t|s_t)}{\pi'(a_t|s_t)}\cdot A_t,\ clip(\frac{\nabla\pi(a_t|s_t)}{\pi'(a_t|s_t)},1-\epsilon,1+\epsilon)\cdot A_t)\right]=\mathbb{E}_\mu\left[\frac{\pi'(a_t|s_t)}{\mu(a_t|s_t)}\cdot L^{CLIP}\right]$$

Notice that the π' inside the importance sampling simplifies with the one inside the PPO ration in case it's not clipped, obtaining as before

$$\mathbb{E}_\mu\left[\frac{\nabla_\theta\pi(a_t|s_t)}{\mu(a_t|s_t)}A_t(s_t,a_t)\right]$$

A.2 Truncation and Bias Correction

$$\begin{aligned}\mathbb{E}_\mu\left[\frac{\pi'(a|s)}{\mu(a|s)}\cdot L^{CLIP}\right]&=\mathbb{E}_\mu\left[\left(min(\frac{\pi'(a|s)}{\mu(a|s)},c)+\left[\frac{\pi'(a|s)}{\mu(a|s)}-min(\frac{\pi'(a|s)}{\mu(a|s)},c)\right]\right)\cdot L^{CLIP}\right]\\&\mathbb{E}_\mu\left[\left(min(\frac{\pi'(a|s)}{\mu(a|s)},c)+\left[\frac{\pi'(a|s)}{\mu(a|s)}-c\right]_+\right)\cdot L^{PPO}\right]\\&=\mathbb{E}_\mu\left[min(\frac{\pi'(a|s)}{\mu(a|s)},c)\cdot L^{CLIP}\right]+\mathbb{E}_\mu\left[\left[\frac{\pi'(a|s)}{\mu(a|s)}-c\right]_+\cdot L^{CLIP}\right]\\&=\mathbb{E}_\mu\left[min(\frac{\pi'(a|s)}{\mu(a|s)},c)\cdot L^{CLIP}\right]+\mathbb{E}_\pi\left[\frac{\mu(a|s)}{\pi'(a|s)}\left[\frac{\pi'(a|s)}{\mu(a|s)}-c\right]_+\cdot L^{CLIP}\right]\\&=\mathbb{E}_\mu\left[min(\frac{\pi'(a|s)}{\mu(a|s)},c)\cdot L^{CLIP}\right]+\mathbb{E}_\pi\left[\left[1-\frac{c\cdot\mu(a|s)}{\pi'(a|s)}\right]_+\cdot L^{CLIP}\right]\end{aligned}$$

following ACER we then compute the L^{PPO} in the first expectation based on the advantages computed with retrace and the one on the correction based on the current Q estimates.

$$\begin{aligned}L^{ACER-PPO}=\mathbb{E}_\mu&\left[min(\frac{\pi'(a_t|s_t)}{\mu(a_t|s_t)},c)\cdot L^{CLIP}\left(\frac{\pi(a_t|s_t)}{\pi'(a_t|s_t)},\ [Q_{ret}(s_t,a_t)-V_\theta(s_t)],\ \epsilon\right)\right]\\&+\mathbb{E}_\pi\left[\left[1-\frac{c\cdot\mu(a|s)}{\pi'(a|s)}\right]_+\cdot L^{CLIP}\left(\frac{\pi(a|s_t)}{\pi'(a|s_t)},\ [Q_\theta(s_t,a)-V_\theta(s_t)],\ \epsilon\right)\right]\end{aligned}$$

with

$$L^{CLIP}(r_t,A_t,\epsilon)=min(r_t\cdot A_t,clip(r_t,1-\epsilon,1+\epsilon)\cdot A_t)$$

B Target for V

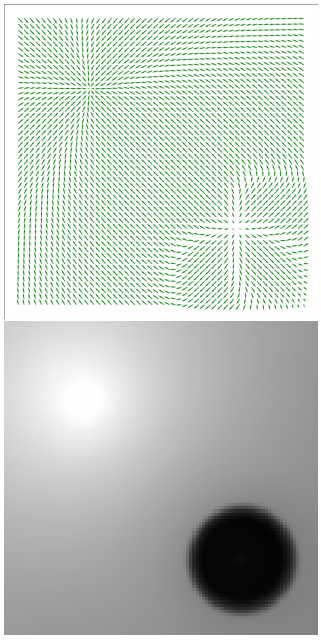
From ACER appendix D we obtain the bias corrected estimate

$$V^{target}(x_t)=\mathbb{E}_\mu\left[min\left\{c,\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}\right\}(Q_{ret}(s_t,a_t)-Q_\theta(s_t,a_t))\right]+\ \mathbb{E}_{a\sim\pi}[Q_\theta(s_t,a)]$$

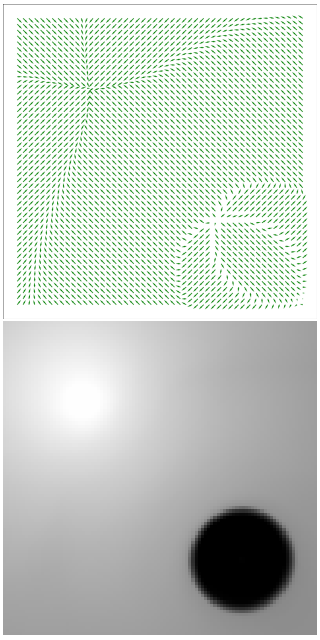
where they can substitute $\mathbb{E}_{a\sim\pi}[Q_\theta(s_t,a)]$ with $V_\theta(s_t)$ because they have already enforced the coherence between the Q and V estimates throught the stocastic part of the dueling network. Since I chose to remove the stochasticity I have to keep the expected value, using instead the Q value computed with the action sampled from π.

C Plots

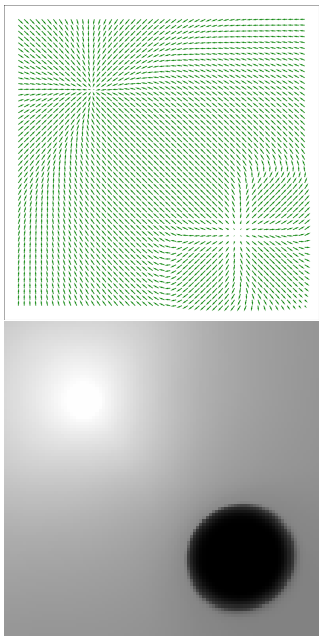
Plots of the policies and V function on a validation plane (with a goal on position (0.25,0.25) and a blackhole in position (0.75,0.75)) at the last step of training with PPO = 8.



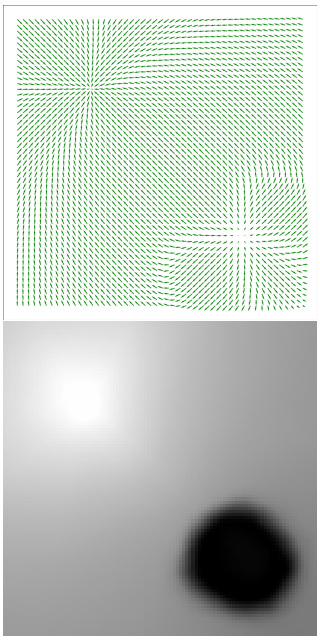
Complete continuous



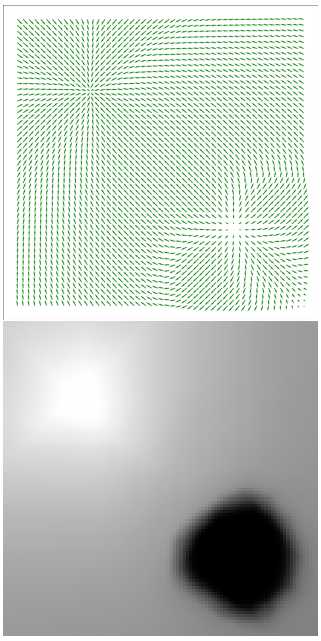
discrete



2 hot



input augmented



input base

D Dueling Network