UNIVERSITY OF LIÈGE

MASTER IN ENGINEERING

May 20, 2022

INFO8003-1
Optimal Decision Making for Complex Problems

Inverted Double Pendulum Project

Maxime Amodei - **s171830**
Cédric Siboyabasore - **s175202**

1 Implementation of the domain

The problem tackled in this project is the one of the Double Inverted Pendulum, in which a two-link pendulum is attached to the center of a car which moves along a track. An agent interacts with the system by applying a horizontal force on the cart.
The goal is for the reinforcement learning agent to maintain the two-link pendulum in a vertical upright position.

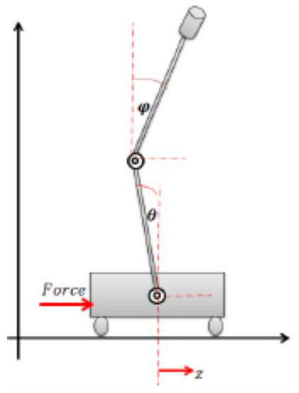


Figure 1: The Double Inverted Pendulum environment

1.1 Formalization of the environment

1.1. (a) State space

The system is characterized by 6 continuous state variables $\mathbf{x} \in \mathbb{R}^6$ defined as

$$\mathbf{x} = \begin{bmatrix} z & \theta & \varphi & \dot{z} & \dot{\theta} & \dot{\varphi} \end{bmatrix}^T$$

where z, θ, φ are the 3 degrees of freedom of the system and respectively correspond to the position of the cart along the horizontal axis, the angle of the first link measured with respect to the vertical axis and the angle of the second link measured with respect to the vertical axis. z takes values between -1 and 1, with the center 0 placed at the center of the track, while θ and γ take values between $-\pi$ and π.
$\dot{z}$ is the horizontal velocity of the cart, while $\dot{\theta}$ and $\dot{\varphi}$ respectively represent the angular velocities of the first and second links.
A terminal state of the environment is reached when the distance between the position p_y of the second link along the vertical axis is above a given thresold of 0.7.

1.1. (b) Action space

The agent can take an action u which has a continuous value in $u \in U = [-1, 1]$. It corresponds to the force the agent applies on the (left side of the) cart.

1.1. (c) Reward signal

The reward is computed as being the sum of 3 terms:

- The "alive" reward, which is state-independent and corresponds to a constant value of 10 in the source code,
- The opposite of the "velocity penality", which is also state-independent and corresponds to a null constant,
- The opposite of the "distance penality", which is given by

  $$0.01p_x^2 + (p_y - 1.7)^2$$

  where p_x and p_y are the positions of the second link along the horizontal axis and along the vertical axis respectively.

The total reward $r(\mathbf{x})$ is thus expressed as

$$r(\mathbf{x}) = 10 - 0.01p_x^2 + (p_y - 1.7)^2$$

1.2 Characterization of the environment

The system is *deterministic*, as its dynamics do not depend on any random disturbance, but only on the system state and on the force F applied on the cart.

The observation $\mathbf{o}(\mathbf{x}) \in \mathbb{R}^9$ of a new state $\mathbf{x}$ of the environment the agent gets is given by

$$\mathbf{o}(\mathbf{x}) = \begin{bmatrix} z & \dot{z} & p_x & \cos\theta & \sin\theta & \dot{\theta} & \cos\varphi & \sin\varphi & \dot{\varphi} \end{bmatrix}^T$$

Therefore, the environment is said to be *partially observable*, as the 9-dimensional observation $\mathbf{o}(\mathbf{x})$ is not equal to the 6-dimensional internal state $\mathbf{x}$. The agent does not directly observe state variables θ and φ.
The system has a *continuous-time* dynamics described by differential equations of the form $\dot{\mathbf{x}} = \mathbf{A}(\mathbf{x})\mathbf{x} + \mathbf{B}(\mathbf{x})u$ derived from the Lagrangian of the system[Bog04] . The environment is *time-invariant*, as the dynamics do not directly depend on time.

The system is in an *infinite-time* ($T \to +\infty$) horizon and *single-state* setting. It is in a *single-agent* framework, with the agent being *single-objective* as the reward $r(\mathbf{x})$ is a scalar.

2 Policy Search Technique

The policy search technique we use is based on the Proximal Policy Optimization (PPO) method [Sch+17]. Section describes how PPO works, as a remainder, and Section details our original contribution to PPO.

2.1 Proximal Policy Optimization

PPO assumes an actor-critic approach, with a value network NN_ϕ to estimate $V(s_t)$ and a policy network NN_θ to estimate $\pi(u_t|s_t)$. In general these networks could share weights, but following the experiments of the original paper, we will assume they are independent.
The main loop of the algorithm repeats the following instructions :

1. Sample N trajectories of length at most T (to account for terminal states) using NN_θ as a policy. This step can be done in parallel as all actors are independent.

2. Using any technique (such as GAE), estimate the advantage $\hat{A}_t$ and the target value V_t^{targ} for each trajectory (at each time step). This step makes use of NN_ϕ.

3. Do K epoch over the training sets (containing the at-most NT one-step transition, possibly structured in mini-batches) of the following steps

   (a) compute the loss L^{CLIP} using formula (8)

   (b) compute the loss L^{VF} using formula (9)

   (c) update the policy network NN_θ using L^{CLIP}

   (d) update the value network NN_ϕ using L^{VF}

The aforementioned algorithm requires two stopping criteria : the first criterion is the number of epoch (K) to perform after collecting the trajectories, the second is the number of time to repeat the instructions above. In this case, we will take a fixed number of iteration and compare the original PPO with our contribution to evaluate any impact on the learning curve.

The Proximal Policy Optimization method requires having access to $\pi_\theta(u_t|s_t)$, which restricts the types of model that can be used. For a discrete policy, a SoftMax classifier is a suitable tool. For a continuous policy, on the other hand, we cannot use a model that simply predicts the output : this would not give the probability of the action. Instead, we can use a Gaussian Policy and assume a normal distribution on the action space. In these settings, the mean is always parametrized, but the standard deviation can be either parametrized or fixed before training. Looking at the literature, we found [CW18] and [PRB13] which fix a constant standard deviation (the same for all inputs) and [CMS17] which uses a prediction of a model as the standard deviation. We did not find a detailed evaluation of the impact of this choice, as such, we aimed for numerical stability. By fixing $\sigma = 0.5$ for all inputs, the range of $\pi_{\mathcal{N}}(x)$ for $x \in [-1,1]$ is approximately $[0.1, 0.8]$. As such, the ratio $r_t(\theta)$ from (8) will likely be in $[0.01, 8]$. A lower standard deviation would bring more instability (because the ratio could get lower or higher), while a higher deviation would attenuate all variations of $r_t(\theta)$.

2.1. (a) Estimating the Advantage and Value Functions

The original paper cites [Sch+15] for an estimation of the advantage function at time t, here are the key formulas in infinite time domain, as presented by [Sch+15] (r_t, s_t and $V(s_t)$ have the same meaning as presented in the theoretical lectures):

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

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \tag{2}$$

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \quad . \tag{3}$$

Where $0 < \lambda < 1$ from (3) is a parameter that defines a tradeoff between a high bias / lower variance when λ is closer to 0 and a low bias / higher variance when λ is closer to 1.
In, equation (11) from [Sch+17], the advantage is estimated via this formula, which seems erroneous.

$$\hat{A}_t = \delta_t + (\lambda\gamma)\delta_{t+1} + \ldots + (\lambda\gamma)^{T-t+1}\delta_{T-1} + \gamma^{T-1}V(s_T) \tag{4}$$

The term $(\lambda\gamma)^{T-t+1}\delta_{T-1}$ doesn't follow the sequence $(\lambda\gamma)^k\delta_{t+k}$ and we believe the superscript is wrong and should be $(\lambda\gamma)^{T-t-1}\delta_{T-1}$ such that the formula becomes,

$$\hat{A}_t = \gamma^{T-1}V(s_T) + \sum_{k=0}^{T-t-1} (\lambda\gamma)^k \delta_{t+k} \quad . \tag{5}$$

This computation requires evaluating $V(s_t)$ on all points of the trajectory using NN_ϕ to build the δ_t terms and estimate the advantage function.
The target V_t^{targ} on the value network is the reward-to-go :

$$V_t^{\mathrm{targ}} = R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k} \tag{6}$$

2.1. (b) Losses in PPO

In Trust Region Policy Optimization, we optimize a surrogate loss

$$L^{\mathrm{CPI}} = \mathbb{E}_t \left\{ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)} \hat{A}_t \right\} = \mathbb{E}_t \left\{ r_t(\theta)\hat{A}_t \right\} \tag{7}$$

where $r_t(\theta)$ is the ratio of probabilities and should not be mistaken for the reward signal.
PPO suggests a new loss

$$\begin{aligned} L^{\mathrm{CLIP}} &= \min\left(r_t(\theta)\hat{A}_t, g(\varepsilon, \hat{A}_t)\right) \\ \text{with} \quad g(\varepsilon,\hat{A}_t) &= \begin{cases} (1+\varepsilon)\hat{A}_t & \hat{A}_t \geq 0 \\ (1-\varepsilon)\hat{A}_t & \hat{A}_t < 0 \end{cases} \end{aligned} \tag{8}$$

It is also possible to add a penalty term to the loss using the KL divergence between $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{\mathrm{old}}}(a_t|s_t)$ but the authors didn't add it in their experiment, so we won't either. Lastly, an entropy bonus term could also be added to encourage exploration, but again, the authors didn't use it, and we won't either.

For the value function, the authors suggest an MSE loss based on a target for the value estimation. We can thus write

$$L_t^{VF} = \left(V_\phi(s_t) - V_t^{\text{targ}}\right)^2 \tag{9}$$

using V_t^{targ} as defined in (6).

2.2 Original Contribution to PPO

The core idea of our contribution is to reorganize the update iteration of the PPO algorithm. The first instruction of the iteration is to estimate the advantage at all time-steps, using the value network from the last iteration. Then update the policy and value networks to fit the training set built using the estimation.
Since estimating $\hat{A}_t$ requires NN_ϕ , we could first update the value network, then estimate the advantage, then update the policy network using information based on a more up-to-date version of the policy network.
More formally, here is our version of the algorithm :

1. Sample N trajectories of length at most T (to account for terminal states) using NN_θ as a policy. This step can be done in parallel as all actors are independent.
2. Collect the rewards-to-go and estimate V_t^{targ} for all time-steps of all trajectories.
3. Do K epoch over the training sets (containing the at-most NT one-step transition with values for V_t^{targ}, possibly structured in mini-batches) of the following steps
   (a) compute the loss L^{VF} using formula (9)
   (b) update the value network NN_ϕ using L^{VF}
4. Using GAE, estimate the advantage $\hat{A}_t$ for each trajectory (at each time step). This step makes use of the updated NN_ϕ.
5. Do K epoch over the training sets (containing the at-most NT one-step transition, possibly structured in mini-batches) of the following steps
   (a) compute the loss L^{CLIP} using formula (8)
   (b) update the policy network NN_θ using L^{clip}

All losses are kept the same, more details are provided in later sections regarding the implementation and experiments.
We hope that by learning the policy on a more "up-to-date" version of the advantage, the model will be faster to reach the same level of performance : the training of the value function is itself not altered.

2.3 Fitted Q-Iteration

The classical algorithm we chose to compare our policy search technique with is the Fitted-Q-iteration, with the Extremely Randomized Trees Regressors as supervised learning algorithm, as it was the algorithm that led to our best results in Assigment 2.

3 Implementation

The PPO algorithm is implemented in `ppo.py` while our version is in `ppo2.py`. As a comparison, `fqi.py` contains a Fitted Q-Iteration implementation. `models.py` contains the definition for all our models and `utils.py` defines various method such as trajectory sampling, advantage estimation, etc. The runs are described in the Jupyter notebook (for the PPO and PPO2 algorithm only).
We did not implement any entropy bonus or KL divergence penalty on the objective function, as the authors of PPO showed no improvement.

4 Methodology

To evaluate the performance of our algorithm, we took the original PPO algorithm as well as the Fitted Q-Iteration algorithm (on Extra-Trees) as comparisons.
The performance criterion is the Expected Cumulative Return (ECR) which is estimated by taking the mean cumulative return of 50 trajectories, limiting the length to 200 steps. This limit may be seen as too restrictive (the environment allow up to 1000 steps), and we intended to raise it, but no trajectory ever reached past 100 steps. Thus, this limit stayed the same. The metric is measured at every epoch (i.e. each that the complete dataset is used) of every iteration (i.e. each time new trajectories are sampled).
For each iteration, we compute the maximum length of the sampled trajectories. This value is less indicative than ECR, thus it is not reported in the results.
We report the loss L^{CLIP} for the policy network and L^{VF} for the value function network. It is worth remembering that the former should be maximized and the latter minimized during training. These metrics are measured at every epoch (i.e. each that the complete dataset is used) of every iteration (i.e. each time new trajectories are sampled).
To compare the algorithms in different settings, we first build a continuous policy that outputs a single scalar (*-Continuous* suffix in the graphs), then a first discrete policy that chooses its action from the set $\{-1.0, -0.5, -0.1, 0.0, 0.1, 0.5, 1.0\}$ (*-Discrete* suffix in the graphs), and finally a discrete policy that chooses its action from the set $\{-1.0, 1.0\}$ (*-Discrete-Bad* suffix in the graphs). We are unsure whether we should expect the second model to perform as well as the first one : the action spaces are different, thus we are comparing different problems without any insight. However, we expect the last model to perform at most as good as the second one, and likely worse : this problem seems complicated and restraining the action space to brutal inputs is not a good idea. We include it to assess how PPO and our modification work in these settings. Note the values 0.1 and -0.1 in the second model : we suppose that tiny nudges may help to stabilize the network, but do not bring any proof for this.
All models (1 continuous policy, 2 discrete policies and 1 value function) are neural networks with 5 hidden layers of 128 nodes each. We looked at a few implementations of Policy Gradient methods, most of them use at least 3 layers with 64 to 512 nodes. The different networks do no share any weight, and they all have the same input shape (9 nodes). The standard deviation of the continuous policy is fixed at $e^{-0.69} \approx 0.5$. We also tried using a learnable parameter, but this parameter converges toward 0, making the loss less stable and thus worsening the training.
We tried different structures for the network (all models were using the same number of hidden layers and nodes) : ranging from 64 to 512, with 3 to 8 hidden layers. While some configuration performed worse than the results shown below, most were very similar to the presented results. Thus, these results are not reported.

As will be shown later, the networks progress well for 1-2 iterations, then the improvements stop. We tried using different scheduler techniques (plateau, regions, exponential) but none could solve our issue. The reported metrics do not use anything fancy : the optimizer type is Adam and the learning rate $5 \cdot 10^{-5}$.
We also tried to deepen the network while training (adding new layers every N step) but this did not solve our "plateau" issues.
All training were done using 50 actors (each sampling independent trajectories), for 50 iterations (number of times the trajectories were sampled), each iteration consisting of 10 epochs.
The length is limited to 200 steps, but this is not too restrictive, as no model trained will ever do better than 150 steps. Finally, the ε parameter of the clipping loss is set to 0.2 (following the original paper) and the minibatch size is set to 64.

5 Results

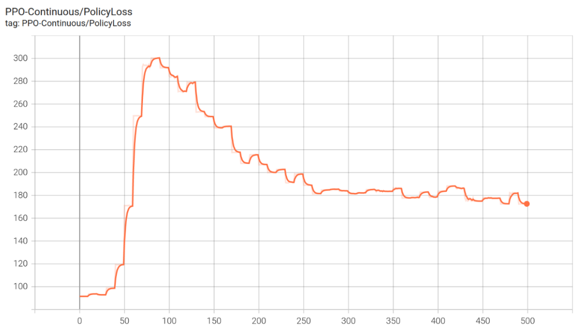
Our results for the original PPO algorithm are shown in Figures 2 to 4 : while some of them show initial progress, all graphs ends training with a plateau, with a model with underwhelming performances. The x axis can be interpreted as the consecutive epochs : the number of time each dataset was seen. In this case we had 50 iterations of 10 epochs so after 10 consecutive epochs, new trajectories are sampled (with the updated policy) and the epochs 11 to 20 use these trajectories as training set.

5.1 PPO

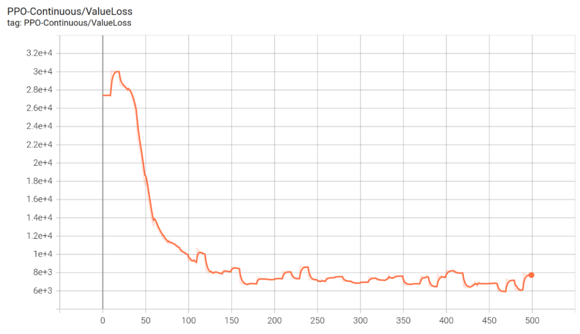
What is interesting is all analogous graphs show the same behavior : for the policy loss (Figs. 2a to 4a), the loss increases at first before reaching a maximum at 50-150 consecutive epochs and finally decrease to reach a plateau. Both the continuous and first discrete policy have their plateau at around 180 (indicating similar performances) while the last discrete (PPO-Discrete-Bad) reaches a plateau at 120.
A similar story can be told about the value loss (Figs. 2b to 4b) : both continuous and discrete policies start at around, $3 \cdot 10^4$ then decrease until the 50th-100th consecutive epoch to reach a plateau at around 10^4. This is deceiving, since even the value function doesn't learn well : we would expect the critic part of the PPO algorithm to perform better, as it is critical to build the loss for the policy. Instead, we see that the models don't learn all details of the problem : which hiders the performance of both algorithm. Interestingly, the value loss for the bad-discrete model starts lower but follow roughly the same shape.
Finally, the expected cumulative return of the policy doesn't change a lot : Figures 2c to 4c) show that if there is an increase at the beginning, it is very small. This shows that even if the policy loss initially increase (and the value loss initially decrease), there is no significant improvement in the performance of the model. We cannot interpret these losses as performance metrics for the model. We can however note that the ECR of the bad-discrete policy is indeed much lower than the ECR of the other policies ($130 < 280$), confirming our beliefs that this model is complicated and needs precision in the action space of the policy.

5.2 PPO with our contribution

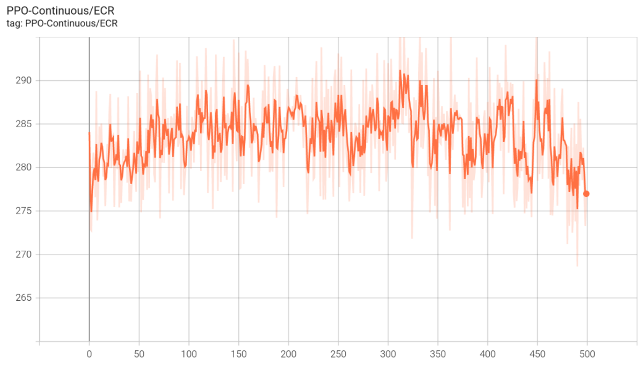
After seeing that the value function has trouble reaching good performances, we should not expect our contribution to be any better : the hope of this contribution was to help the policy network learn



(a) L^{CLIP}



(b) L^{VF}



(c) Expected Cumulative Reward

Figure 2: Losses and Expected Cumulative Reward for the Continuous model on the original PPO algorithm.

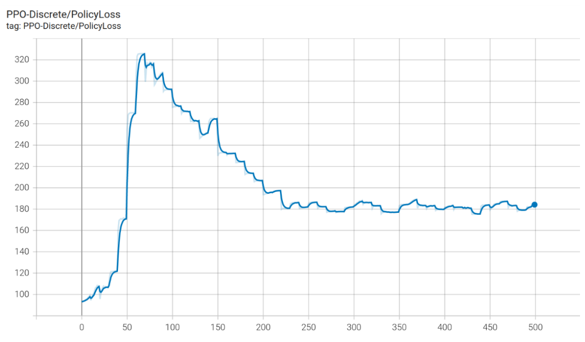
with a better advantage estimation. Since the estimation is bad, there is no reason to believe that the policy would reach good performance either.
Indeed, Figures 5a-5c are almost identical to Figures 2a-2c, showing no benefit in our method. Again, the curve show some progress until 50-150 epochs where the curves start to plateau to similar values. Figure 6c may seem to show a good learning potential at first, but the highest value reached (168) is lower than what the original PPO achieved (288). While this run shows potential,
The last discrete model (with over constrained action space) shows similar performances to the PPO algorithm.

5.3 FQI

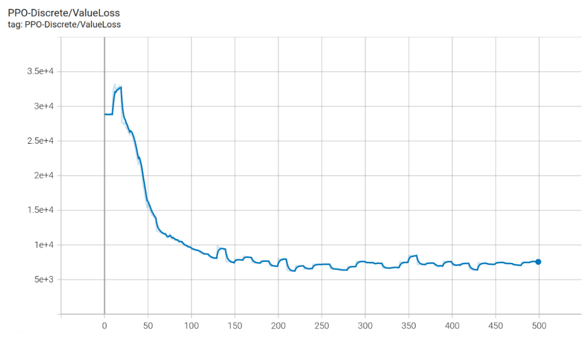
Despite running for $N = \frac{\log\left\{b\frac{(1-\gamma)^2}{2B_r}\right\}}{\log\gamma} \approx 225$ where b is a bound we set to 0.1 and discount factor γ was set to 0.95 (to have a number of iterations not too large), we show the ECR for the first few iterations, as the result does not seem to be better than with our PPO.

Iteration 1	228.4658
Iteration 2	274.4154
Iteration 3	255.0989
Iteration 4	224.6316
Iteration 5	313.5742
Iteration 6	256.1562

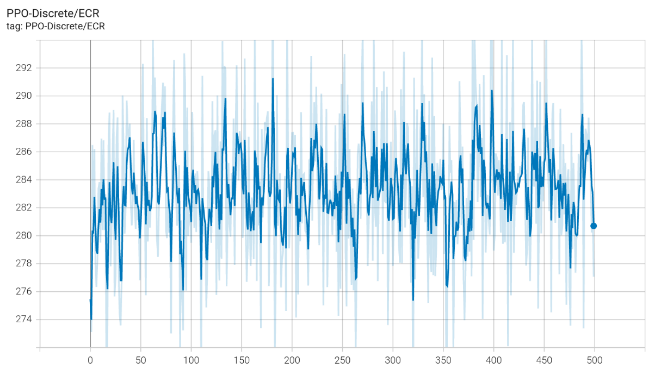
Table 1: Expected Cumulative Return with Fitted-Q-Iteration



(a) L^{CLIP}



(b) L^{VF}



(c) Expected Cumulative Reward

Figure 3: Losses and Expected Cumulative Reward for the Discrete model with a fine range of actions ($[-1,-0.5,-0.1,0.0,0.1,0.5,1.0]$) on the original PPO algorithm.

References

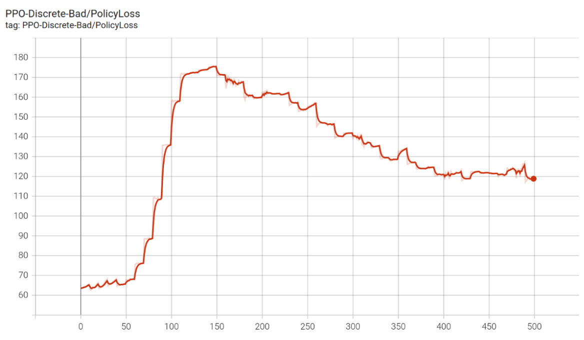
[Bog04] Alexander Y. Bogdanovr. *Optimal Control of a Double Inverted Pendulum on a Cart*. 2004.

[PRB13] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. "Adaptive Step-Size for Policy Gradient Methods". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: `https://proceedings.neurips.cc/paper/2013/file/f64eac11f2cd8f0efa196f8ad173178e-Paper.pdf`.

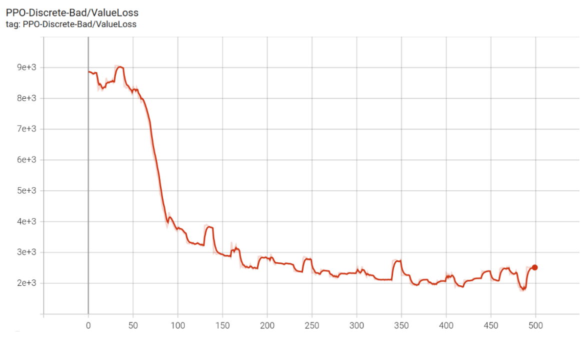
[Sch+15] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: `10.48550/ARXIV.1506.02438`. URL: `https://arxiv.org/abs/1506.02438`.

[CMS17] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. "Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 834–843. URL: `https://proceedings.mlr.press/v70/chou17a.html`.

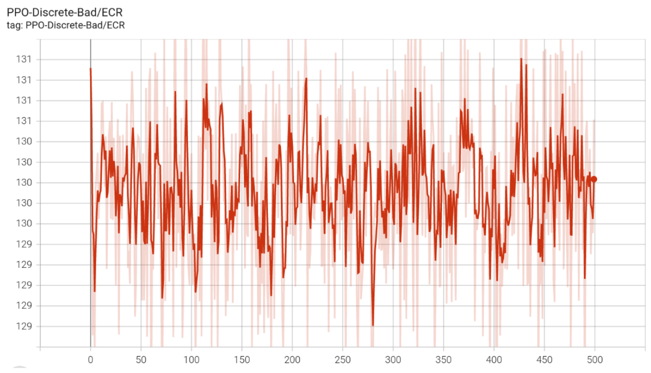
[Sch+17] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: `10.48550/ARXIV.1707.06347`. URL: `https://arxiv.org/abs/1707.06347`.



(a) L^{CLIP}



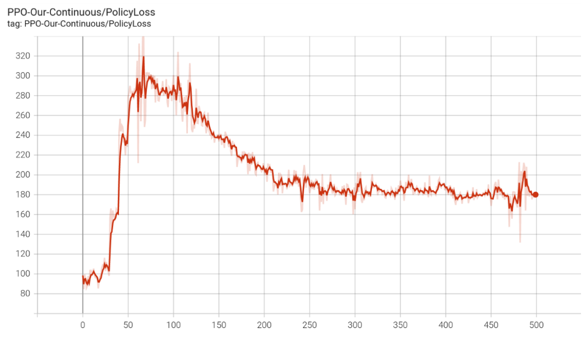
(b) L^{VF}



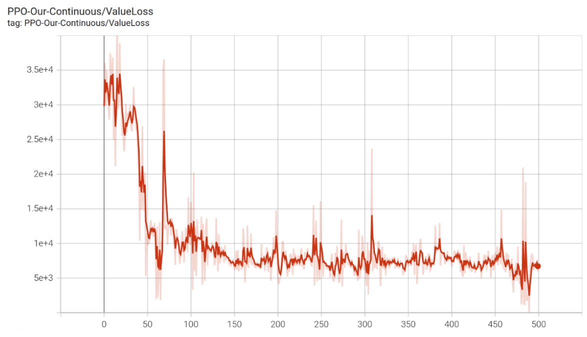
(c) Expected Cumulative Reward

Figure 4: Losses and Expected Cumulative Reward for the Discrete model with only full throttle actions ($[-1, 1.0]$) on the original PPO algorithm.

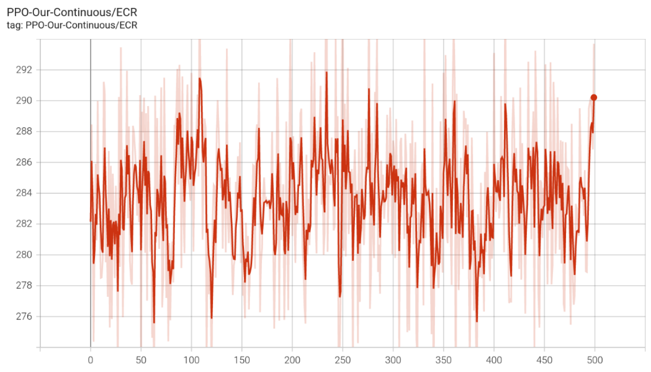
[CW18] Kamil Ciosek and Shimon Whiteson. "Expected Policy Gradients". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). URL: `https://ojs.aaai.org/index.php/AAAI/article/view/11607`.



(a) L^{CLIP}

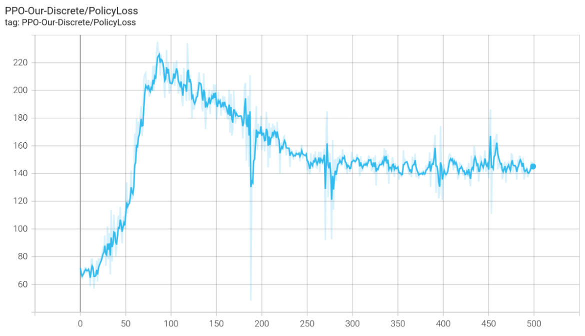


(b) L^{VF}

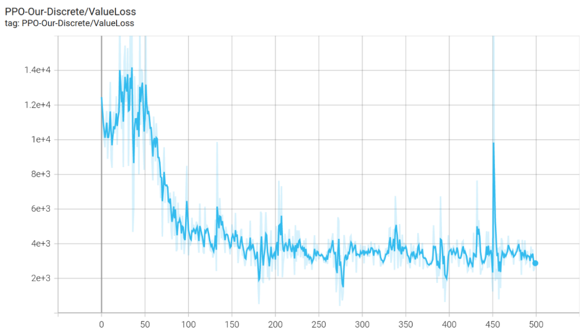


(c) Expected Cumulative Reward

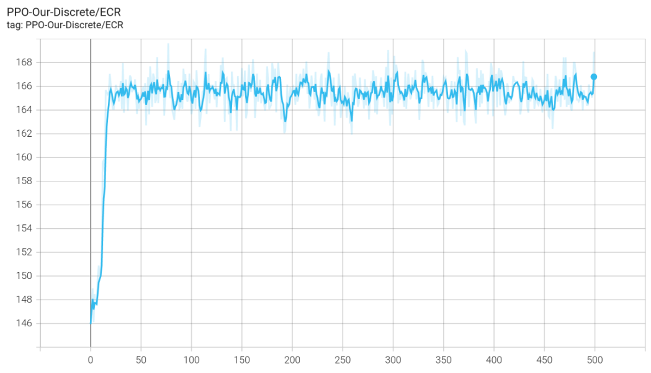
Figure 5: Losses and Expected Cumulative Reward for the Continuous model on the modified PPO algorithm (PPO2).



(a) L^{CLIP}

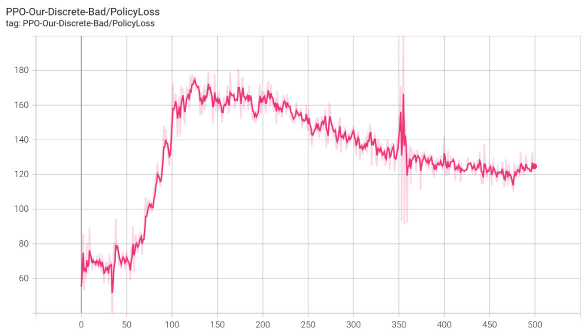


(b) L^{VF}

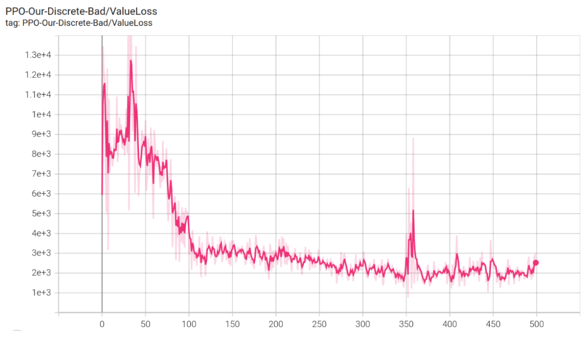


(c) Expected Cumulative Reward

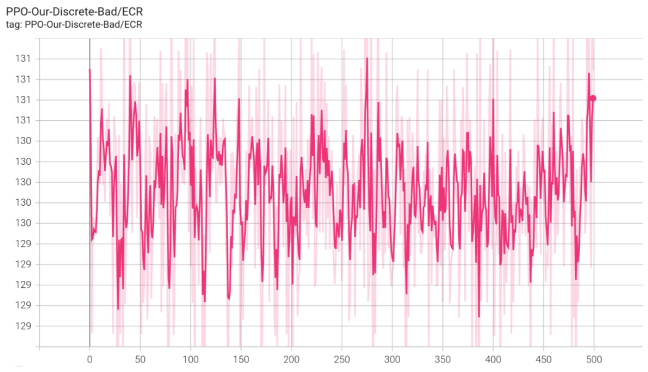
Figure 6: Losses and Expected Cumulative Reward for the Discrete model with a fine range of actions ($[-1, -0.5, -0.1, 0.0, 0.1, 0.5, 1.0]$) on the modified PPO algorithm (PPO2).



(a) L^{CLIP}



(b) L^{VF}



(c) Expected Cumulative Reward

Figure 7: Losses and Expected Cumulative Reward for the Discrete model with only full throttle actions ($[-1, 1.0]$) on the modified PPO algorithm (PPO2).