Policy Gradient Methods
&
Applications to Learned Locomotion

Alexander Krolicki
akrolic@clemson.edu

Sarang Sutavani
ssutava@clemson.edu

Overview

1. Methods
2. Our Problem
3. Results
4. Comparison
5. Summary
6. Future Work

Policy Gradient Methods

- Methods that can learn a parameterized policy without the help of a value or action-value function.
- The methods usually seek to maximize a performance index:

$$J(\theta) = \mathcal{V}_{\pi_\theta} = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{1}$$

- Update rule follows gradient accent:

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t) \tag{2}$$

- All policy gradient methods follow this general scheme.
- Value (or Q) function can still be used for reducing variance (for faster convergence).
- Techniques that try to learn the value function along side the policy are termed actor-critic methods. (Actor - policy; Critic - value function)

Policy Gradient Methods: Advantages

- Value functions are learned to eventually determine a policy, so why use a broker when we can learn the policy directly!
- In certain cases learning a policy can be much more straight forward than learning a value function.
- More effective in dealing with continuous state and/or action spaces.
- Knowledge of the problem can be utilized to guide the policy search.

Limitations of Plain Policy Gradient

- Poor sample efficiency: data is discarded after each update.
  - due to on-policy learning/estimation
  - stable but extremely slow to converge
- Difficulty in deciding and updating proper step size.
  - even small changes in parameter could result in large changes in policy
- Importance Sampling:

$$\nabla_\theta J(\theta) = \underset{\tau \sim \theta'}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \frac{P(\tau_t \mid \theta)}{P(\tau_t \mid \theta')} \gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t) A_\theta(s_t, a_t) \right] \quad (3)$$

$$\frac{P(\tau_t \mid \theta)}{P(\tau_t \mid \theta')} = \prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'} \mid s_{t'})}{\pi_{\theta'}(a_{t'} \mid s_{t'})} \qquad \text{(Importance sampling ratio)} \quad (4)$$

  - data can be used more efficiently (off-policy learning)
  - unstable due to exploding or vanishing gradients (importance sampling ratio)

Comparing 2 Policies

- For any two policies π and π':

$$J\left(\pi'\right)-J(\pi)=\underset{\tau\sim\pi'}{\mathbb{E}}\left[\sum_{t=0}^{\infty}\gamma^{t}A^{\pi}\left(s_t,a_t\right)\right] \tag{5}$$

$$=\frac{1}{1-\gamma}\underset{\substack{s\sim d^{\pi'}\\ a\sim\pi}}{\mathbb{E}}\left[\frac{\pi'(a\mid s)}{\pi(a\mid s)}A^{\pi}(s,a)\right] \tag{6}$$

$$\approx\frac{1}{1-\gamma}\underset{\substack{s\sim d^{\pi}\\ a\sim\pi}}{\mathbb{E}}\left[\frac{\pi'(a\mid s)}{\pi(a\mid s)}A^{\pi}(s,a)\right] \tag{7}$$

$$=\underset{\tau\sim\pi}{\mathbb{E}}\left[\sum_{t=0}^{\infty}\gamma^{t}\frac{\pi'\left(a_t\mid s_t\right)}{\pi\left(a_t\mid s_t\right)}A^{\pi}\left(s_t,a_t\right)\right]\doteq L_{\pi}\left(\pi'\right) \tag{8}$$

discounted future state distribution, $d^{\pi}(s)=(1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}P\left(s_t=s\mid\pi\right)$

- Relative performance in terms of 'Loss function' $L_{\pi}(\pi')$ and KL divergence:

$$\left|J\left(\pi'\right)-\left(J(\pi)+L_{\pi}\left(\pi'\right)\right)\right|\leq C\sqrt{\underset{s\sim d^{\pi}}{\mathbb{E}}\left[D_{KL}\left(\pi'\|\pi\right)[s]\right]} \tag{9}$$

where, $D_{KL}\left(\pi'\|\pi\right)[s]=\sum_{a\in\mathcal{A}}\pi'(a\mid s)\log\frac{\pi'(a\mid s)}{\pi(a\mid s)}$ (10)

Monotonic Improvement in Policy

- Optimize over new function:

$$\max_{\pi'} L_{\pi_k}\left(\pi'\right) - C \max_{s\sim d^{\pi_k}} \left[D_{KL}\left(\pi' \| \pi_k\right)[s]\right] \qquad (11)$$

maximizing π' is an improved policy.

- Surrogate objective used:

$$\begin{aligned} &\arg\max_{\pi'} L_{\pi_k}\left(\pi'\right) \\ \text{s.t.} \quad &\underset{s\sim d^{\pi_k}}{E} \left[D_{KL}\left(\pi' \| \pi_k\right)[s]\right] \leq \delta \end{aligned} \qquad (12)$$

- Some well known policy gradient methods approximating this objective:
    - Natural Policy Gradient
    - Trust Region Policy Optimization
    - Proximal Policy Optimization

Trust Region Policy Optimization

- Linear approximation of objective:

$$L_{\theta_k}(\theta) \approx L_{\theta_k}(\theta_k) + g^T(\theta - \theta_k) \quad g \doteq \nabla_\theta L_{\theta_k}(\theta)\Big|_{\theta_k} \tag{13}$$

- Quadratic approximation of constraint:

$$\bar{D}_{KL}(\theta \| \theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \quad H \doteq \nabla_\theta^2 \bar{D}_{KL}(\theta \| \theta_k)\Big|_{\theta_k} \tag{14}$$

- Optimization problem:

$$\arg\max_\theta g^T(\theta - \theta_k) \tag{15}$$

$$\text{s.t. } \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \leq \delta \tag{16}$$

- Solution to approximated problem:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{17}$$

- $H^{-1}g$: estimated using Conjugate Gradient (CG)
- Line search in direction of the estimated gradient: to make sure $L_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta \| \theta_k) \leq \delta$ (δ defines the trust region). Adjust δ to meet the conditions.

Proximal Policy Optimization

- **PPO with adaptive KL penalty:** solves unconstrained optimization problem

$$\arg\max_{\theta} L_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta \| \theta_k) \tag{18}$$

β_k is adaptive and optimization is performed over a batch.
$d = \bar{D}_{KL}(.)$, if $d \leq d_{targ}/1.5$, $\beta \leftarrow \beta/2$, if $d \geq d_{targ} * 1.5$, $\beta \leftarrow \beta * 2$.

- **PPO with Clipped Objective:**

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}, \qquad \operatorname{clip}(r, a, b) = \begin{cases} a \text{ if } r < a \\ b \text{ if } r > b \\ r \text{ otherwise} \end{cases} \tag{19}$$

$$L_{\theta_k}^{CLIP}(\theta) = \underset{\tau \sim \pi_k}{\mathbb{E}}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\hat{A}_t^{\pi_k}, \operatorname{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k}\right)\right]\right] \tag{20}$$

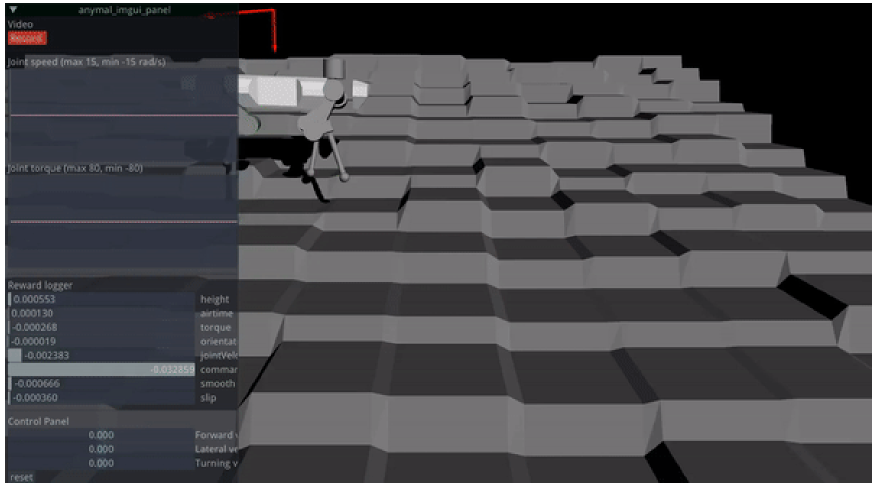
$$\arg\max_{\theta} L_{\theta_k}^{CLIP}(\theta) \qquad \text{(objective without critic)}$$

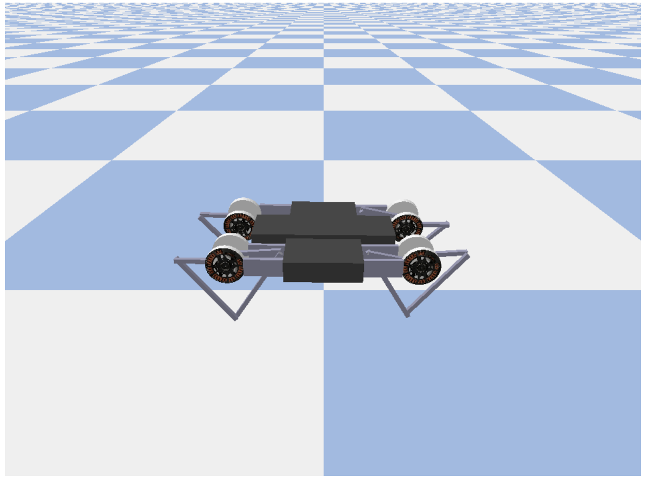
$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{21}$$

$$\arg\max_{\theta} L_{\theta_k}^{CLIP+VF+S}(\theta) \qquad \text{(objective with critic)}$$

Application to Learned Locomotion

- Deep Reinforcement Learning is actively researched and applied to physical robots with impressive results. [1]
- Most methods rely on domain specific knowledge, model based approaches, or even imitation learning. [2]
- Our study aims to evaluate how a quadruped agent can learn a gait motion completely from scratch.

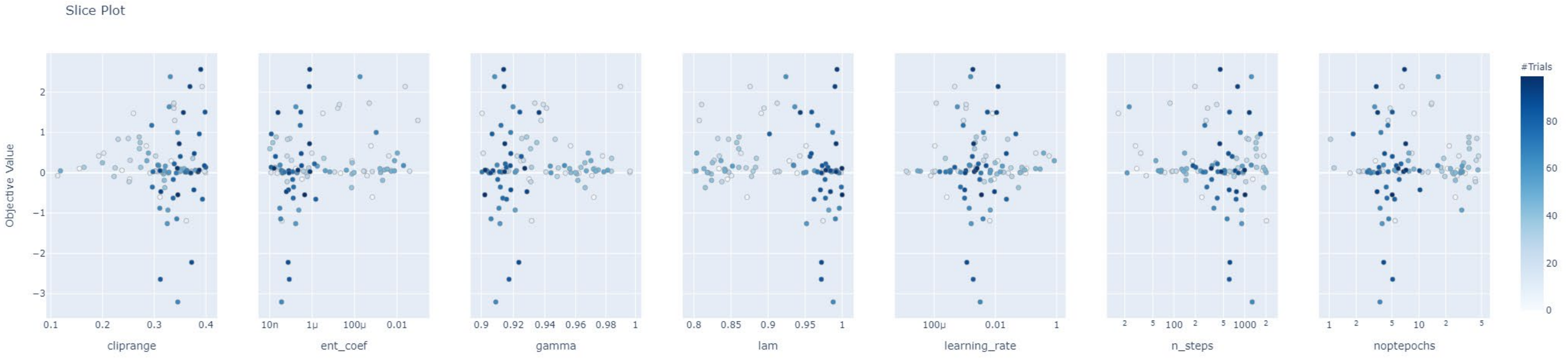




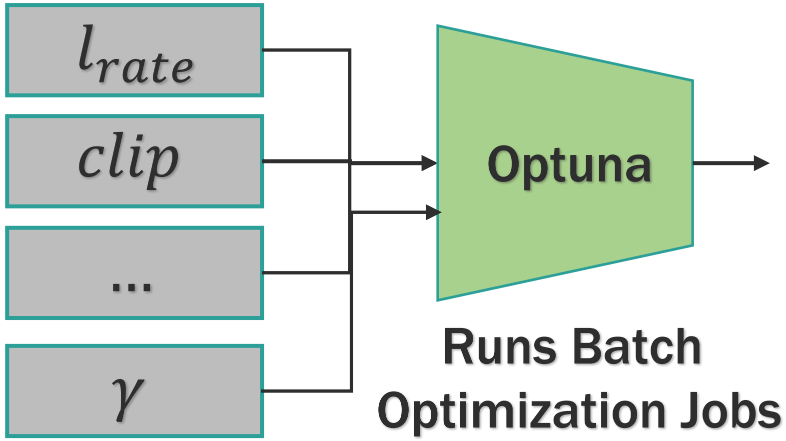
Environment: Spaces, Rewards, Episodes

- State Space $S \in \mathbb{R}^{28}$
  - Motor Angles $S \in \boldsymbol{q} = \{q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, q_{42}\}$
  - Motor Velocities $S \in \dot{\boldsymbol{q}} = \{\dot{q}_{11}, \dot{q}_{12}, \dot{q}_{21}, \dot{q}_{22}, \dot{q}_{31}, \dot{q}_{32}, \dot{q}_{41}, \dot{q}_{42}\}$
  - Motor Torques $S \in \boldsymbol{T} = \{T_{11}, T_{12}, T_{21}, T_{22}, T_{31}, T_{32}, T_{41}, T_{42}\}$
  - Base Pose $S \in \boldsymbol{p} = \{\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}, \vec{\boldsymbol{z}}, \vec{\boldsymbol{w}}\}$
- Action Space $S \in \mathbb{R}^{8}$
  - Motor Angles $A \in \boldsymbol{q}_{\boldsymbol{desired}} = \{q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, q_{42}\}$
- Reward Function
  - Penalizes not moving forward and actuator effort
  - $R = (\boldsymbol{p}_n - \boldsymbol{p}_{n-1})\overrightarrow{\boldsymbol{x}} - w\Delta t|\boldsymbol{T}_n \cdot \dot{\boldsymbol{q}}|$
- Terminal State
  - Center of mass is <0.13 meters to the ground
  - Simulation reaches 1000 steps forward in time

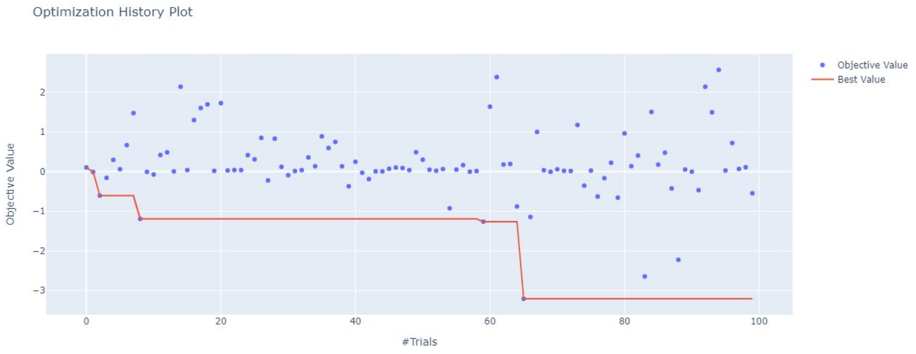
Hyperparameter Tuning



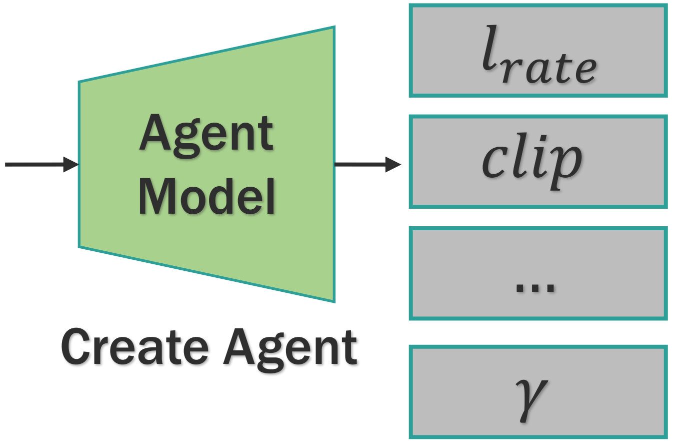
Hyperparameter Distributions



Runs Batch Optimization Jobs



Load Best Hyperparameters

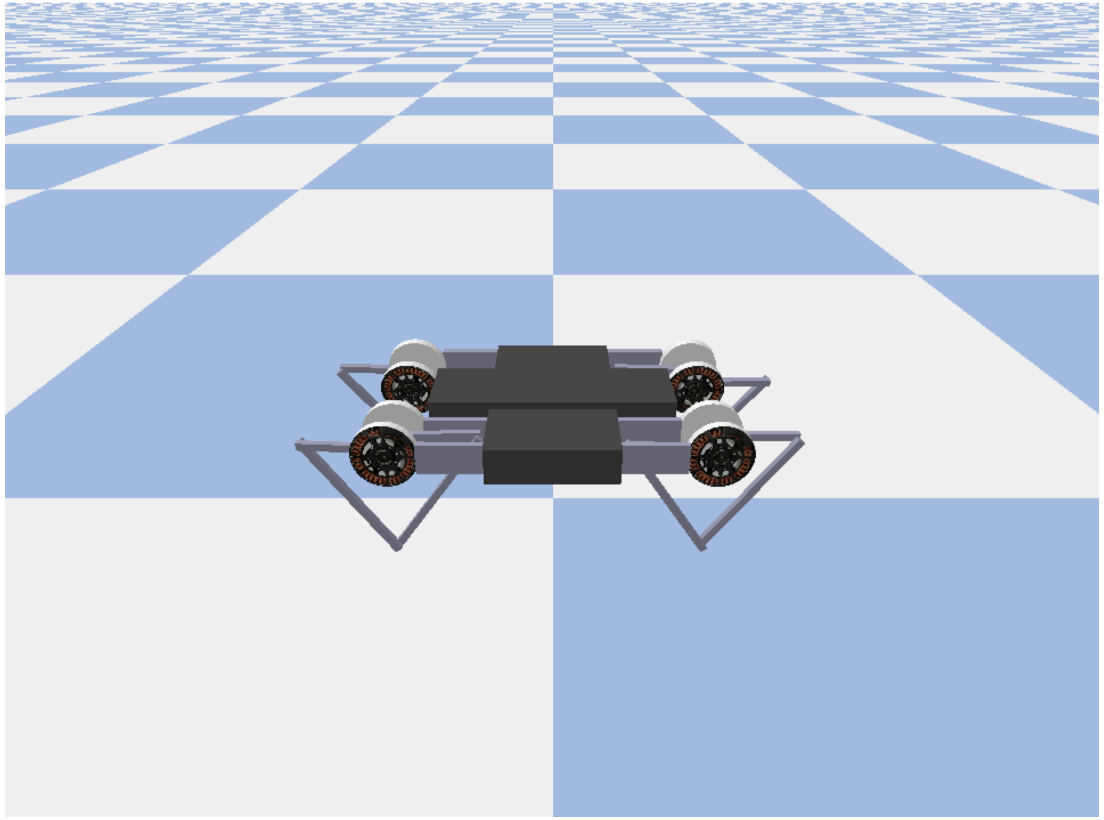


Create Agent

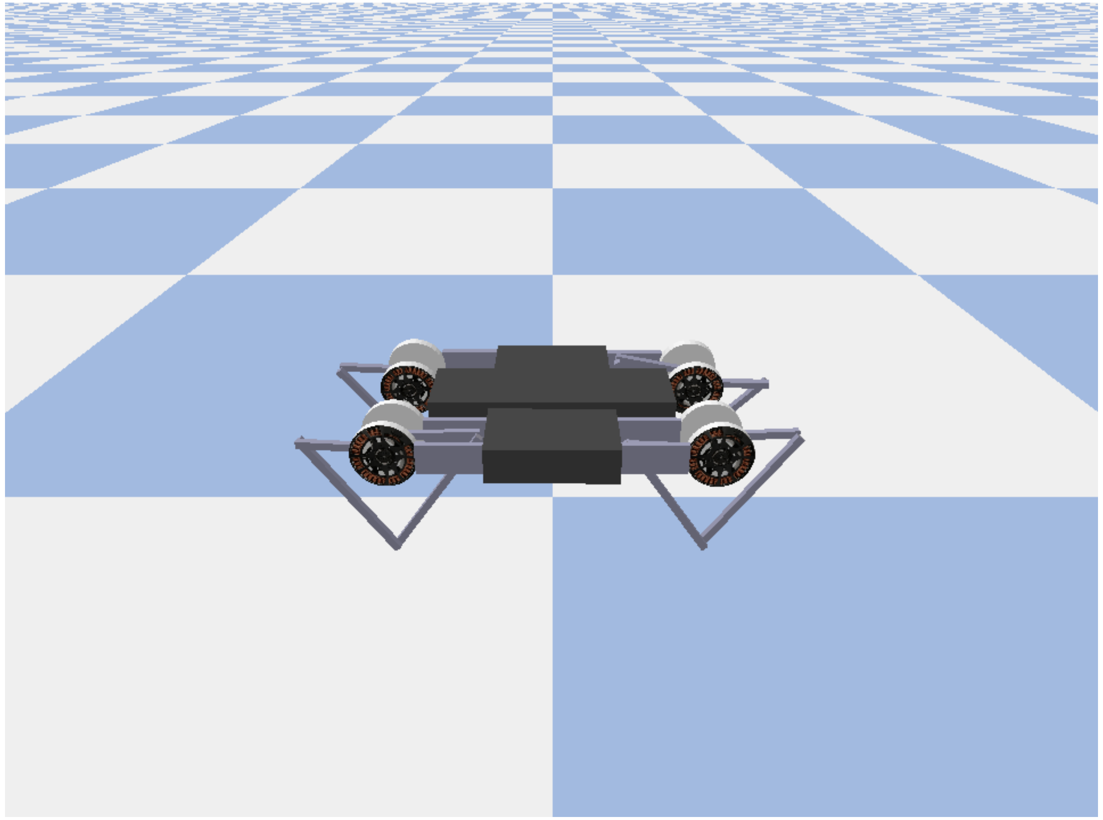
PPO2 Tuning
100 permutations
100,000 Training Episodes/permutation
~24 Hours to complete

TRPO Tuning
100 permutations
200,000 Training Episodes/permutation
~24 Hours to complete

Visualizing Final Learned Policies



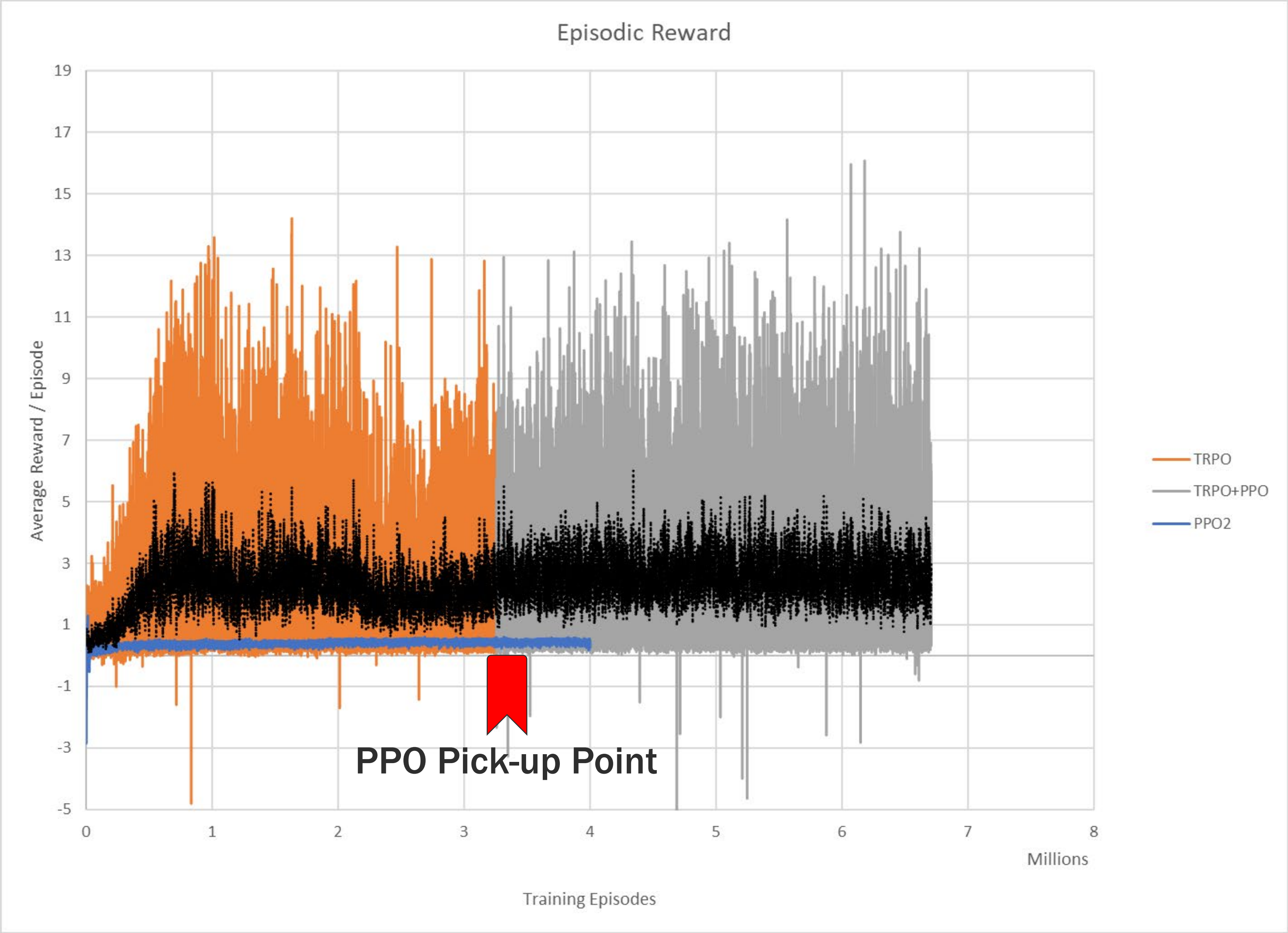
PPO2 Agent After 4 Million Episodes
Avg. Reward = 0.5 $\pm$0.05
Training Time = 5 Hours



TRPO Agent After 4 Million Episodes
Avg. Reward = 2.5 $\pm$1.5
Training Time = 6 Hours

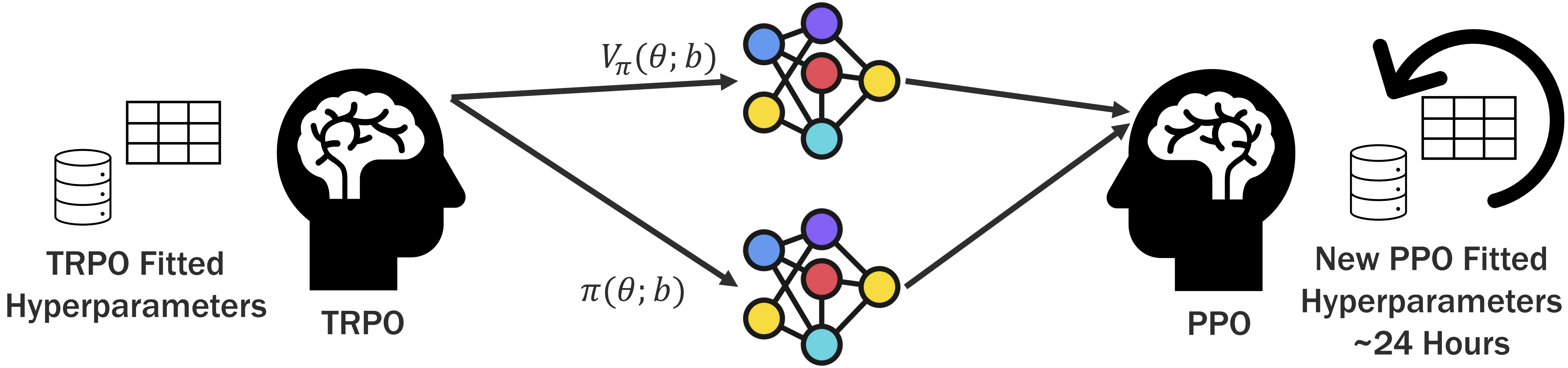
Training Scheme

- PPO and TRPO were trained separately with their own optimal parameters.
- Benchmarks of the learned models were saved every 200k episodes
- Since TRPO performed well in exploration, the policy and value function networks were extracted and placed into a PPO agent to continue training.
- The hypothesis was that the reward signal noise would reduce since PPO showed little variance during learning.

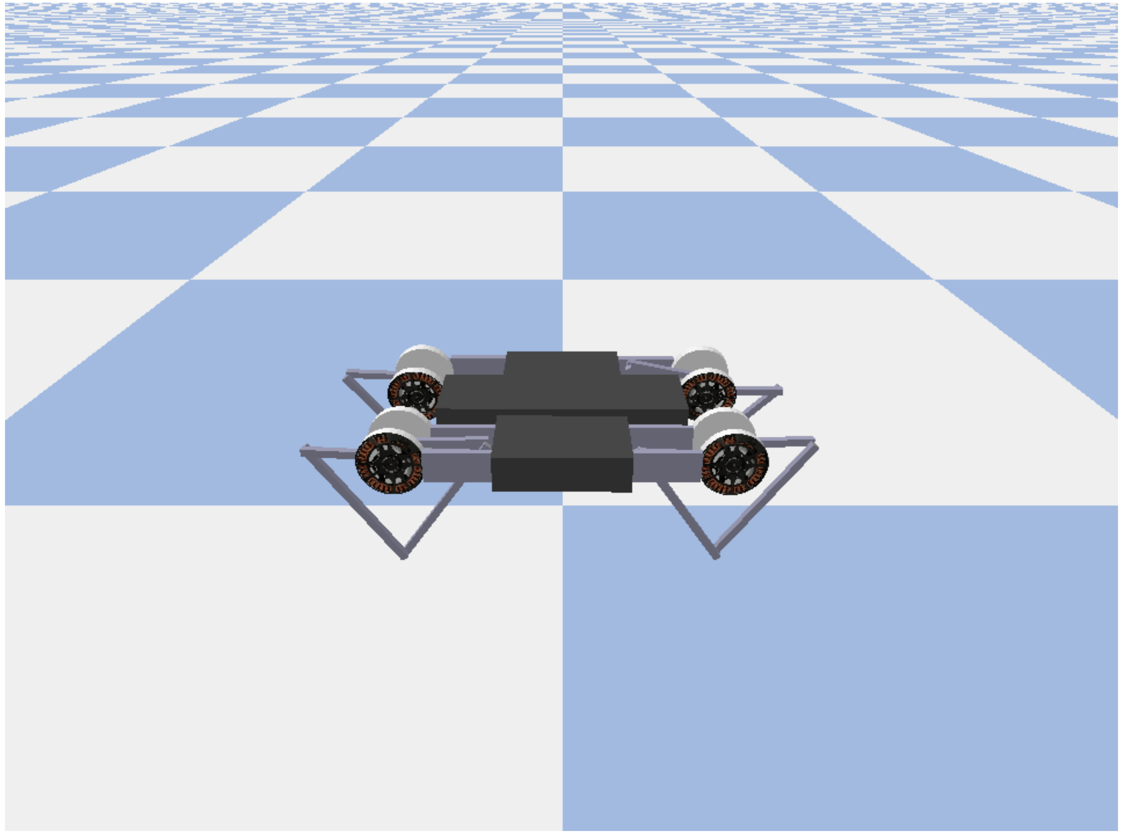


Best of both worlds?

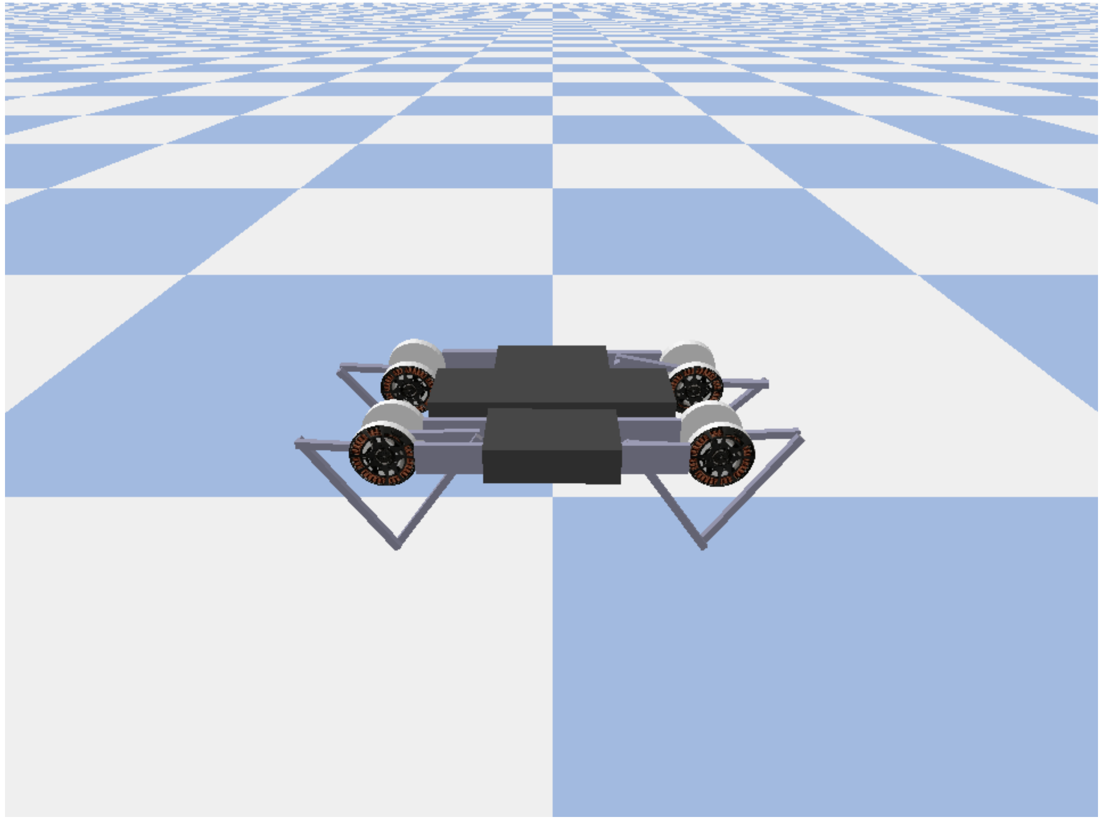
- Using TRPO to find a high return policy and value function network, we then transfer this network to a PPO agent and run optimization starting 4 Million episodes.
- The PPO agents best hyperparameters were then trained for an additional 4 Million episodes to attempt to improve the quality of the policy and value function networks.



Visualizing Final Learned Policies (Best of the Rest)



TRPO to PPO2 Agent After ~7 Million Episodes
Avg. Reward = 3 $\pm$ 1.5
Training Time = 11 Hours



TRPO Agent After 4 Million Episodes
Avg. Reward = 2.5 $\pm$1.5
Training Time = 6 Hours

Comparison

- Qualitative Assessment
  - PPO: Took too literally the reward function and converged to a local minimum solution.
  - TRPO: Achieved a fast novel gait motion but with high speed comes greater risk of losing or gaining a significant amount of rewards.
- Value Function Approximation
  - Clearly TRPO achieves a higher return, but suffers from high variance in the policy update without clipping.
  - PPO starting with a high return policy still was not able to stabilize the policy and value function networks.
  - Increasing the # of layers and hidden units may be necessary for this problem
- Optimization Results
  - Running both agents for a longer number of episodes during optimization would benefit the in the long-term stability of the learned policy and value function networks.

Summary

- A total time of 72 Hours tuning hyperparameters and 24 Hours raining
- TRPO outperformed PPO in terms of exploration
- PPO was not able to stabilize the optimal policy generated by TRPO
- Hyperparameter tuning is extremely important when experimenting with different RL environments
- Without a reference gait trajectory, the learned policy depends on a well defined reward function
- Variations to the quadrupeds mass, leg lengths, joint friction, sensor noise, ect… May improve the robustness of the learned policy

- Developed in a Google Colab Notebook for ease of access

Future Work

- Investigate Multi-agent reinforcement learning methods
- Provide a reference trajectory for stable policy
- Deploy methods onto physical systems

References

Schulman, John, et al. “Proximal Policy Optimization Algorithms.” *ArXiv.org*, 28 Aug. 2017, arxiv.org/abs/1707.06347 .

Schulman, John, et al. “Trust Region Policy Optimization.” *ArXiv.org*, 20 Apr. 2017, arxiv.org/abs/1502.05477 .

Tan, Jie, et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots.” *ArXiv.org*, 16 May 2018, arxiv.org/abs/1804.10332 .

Erwin Coumans and Yunfei Bai PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2016—2021, http://pybullet.org

Tsounis, Vassilios, et al. “DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning.” ArXiv.org, 31 Jan. 2020, arxiv.org/abs/1909.08399.

Kakade, S., et al. (2002, July). "Approximately optimal approximate reinforcement learning." In ICML (Vol. 2, pp. 267-274).

Kakade, S. M. (2002). A natural policy gradient. In Advances in neural information processing systems (pp. 1531-1538).

https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html

https://stable-baselines.readthedocs.io/en/master/modules/trpo.html

Appendix

Policy Gradient Theorem

- Policy Gradient Theorem:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta})$$

- New update rule:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}\left(S_t, a, \mathbf{w}\right) \nabla \pi\left(a \mid S_t, \boldsymbol{\theta}\right)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \left(G_t - b\left(S_t\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \qquad \textit{(REINFORCE with baseline)}$$

- For advantage estimate:

$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \underset{\tau \sim \pi_\theta}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta \left(a_t \mid s_t\right) A_{\pi_\theta}\left(s_t, a_t\right) \right]$$

Environment: Overcoming Implementation Challenges

- Physics/Kinematics
  - Bullet Physics engine handles the model environment interaction.
  - All components have mass and inertia matrices.
  - Joints have friction and dampening.
- Actuator Models
  - Accurate models of the motors operating characteristics are used to generate actions
- Simulated Latency
  - Observations are back logged and sent with a delay to simulate the latency a real control system would exhibit (0.001 – 0.002s)
  - Gaussian noise is injected into state signals
- Parallelizable Agents
  - Its possible to spin up several agents in headless mode, which can help expedite training if the algorithm exploits multithreading or tensor cores.

PPO Tuning Results

Hyperparameters	Values
Number of State Steps until Terminal State n_{steps}	1276
Discount Factor γ	0.909
Learning Rate l_{rate}	0.00317
Entropy Coefficient e	3.59e-8
Clipping parameter controlling policy update rate ϵ	0.345
Clipping parameter controlling value function update rate	0
# of epochs when optimizing the surrogate objective function K	4
Generalized Advantage Estimator factor λ	0.988
Policy Network (DNN) 2 layers with 64 hidden units each	-
Value Function Network (DNN) 2 layers with 64 hidden units each	-

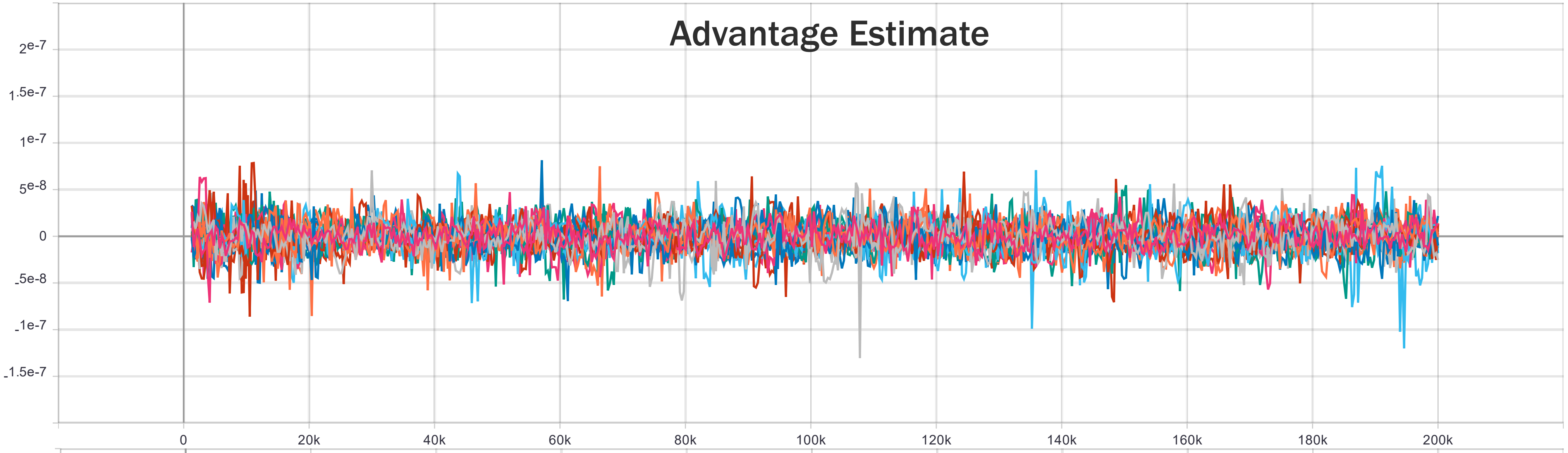
TRPO Tuning Results

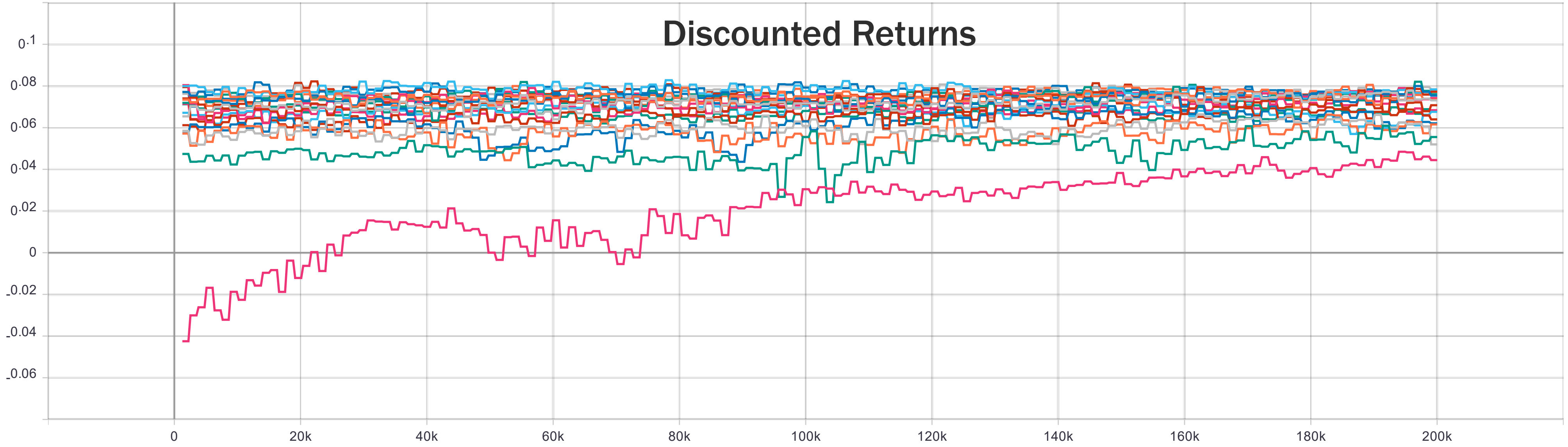
Hyperparameters	Values
Time steps per batch t_{batch}	293
Discount Factor γ	0.974
Kullback-Leibler loss threshold	0.0503
Weight for the entropy loss	5.03e-3
The compute gradient dampening factor	0.0135
Value Function Step Size	3.2e-3
Value Function # of iterations for learning	3
Generalized Advantage Estimator factor λ	0.988
Policy Network (DNN) 2 layers with 64 hidden units each	-
Value Function Network (DNN) 2 layers with 64 hidden units each	-

PPO+TRPO Tuning Results

Hyperparameters	New Values	Old Values
Number of State Steps until Terminal State n_{steps}	1277	1276
Discount Factor γ	0.913	0.909
Learning Rate l_{rate}	1.83e-5	0.00317
Entropy Coefficient e	5e-4	3.59e-8
Clipping parameter controlling policy update rate ϵ	0.379	0.345
Clipping parameter controlling value function update rate	0	0
# of epochs when optimizing the surrogate objective function K	1	4
Generalized Advantage Estimator factor λ	0.861	0.988
Policy Network (DNN) 2 layers with 64 hidden units each	-	-
Value Function Network (DNN) 2 layers with 64 hidden units each	-	-

PPO Results


Advantage Estimate
2e-7
1.5e-7
1e-7
5e-8
0
-5e-8
-1e-7
-1.5e-7
0
20k
40k
60k
80k
100k
120k
140k
160k
180k
200k


Discounted Returns
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
20k
40k
60k
80k
100k
120k
140k
160k
180k
200k

- On-Policy method which aims to learn iteratively through a surrogate objective function, which learns new policies for a specified number of epochs.
- After these epochs have passed, the policy update is performed carefully by choice of a clipping hyperparameter which ensures policy update steps are not too large.