Deep Reinforcement Learning on Hopper and Bipedal Walker

Udesh Habaraduwa

1 Part I - Hopper

1.1 Q1 : Choosing a model

1.1.1 Model selection

The first step is to decide on an algorithm for training. For deciding on which algorithm to use, the following factors were considered.

1. The action and observation spaces

   The *Hopper-V4* environment provided in Gymnasium [1] is defined by a continuous action and observsation space. Actions ranging (-1,1) representing the torque that can be applied to the thigh, leg and foot rotors [2]. The observation space ranging (-inf,inf) is similary continuous for 11 different dimensions (angles and velocities joints and body parts). Thus the model should be able to take 11 continous values and return 3 continuous values (a torque vector for each part).

2. Sample efficiency

   The agents will be trained on a GPU4EDU remote compute cluster ((Nvida A40 GPU, Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz)) with no (direct) training / simulation cost but the model should not require too many samples to train, to keep the training periods managable while providing good performance. Given the timeframe and access to a resonably poweful (for the task) computing infrastructure, a less sample efficient (e.g., model-free) but more performant model, that may take is feasible.

3. Model generalization

   The task requires agent evaluation over serveral different environments thus generalizability is an important consideration. Here, two considerations were made, depending on what is considered 'the environment'. If we consider the environment the physics dynamics and not the hopper itself, perhaps we can learn these dynamics which would apply regardless of the mass of the torso. Otherwise, we consider the entire thing (the running environment and the hopper itself) as the environment in which case a model-based strategy would not generalize well.

4. Options available in stable-baselines

Given these considerations, I opt for the model-free TRPO. For one, learning the dynamics using a neural network based method maybe expensive, even with access to a cluster. Though TRPO is less sample efficient than model-based techniques, given the simulation environment (where collecting samples is cheap) this is an acceptable cost for the possiblity of better generalization across environments. It is also less sensitive to hyperparameters [3].

1.2 Training

1.2.1 Setup and validation

Prior to training the model in the training environment (described in 1.1 with packages as in appendix table 7) the model training setup was validated in a Google Colab [4] environment (relevant package versions in appendix table 7) due to its convenient integration with Tensoboard [5]. The doucmentation provided by [6] were used extensively in setting up the training and monitoring settings. It was found that performance gains stabilized at approximately 1 million timesteps which was subsequently chosen for the training runs for all 3 models.

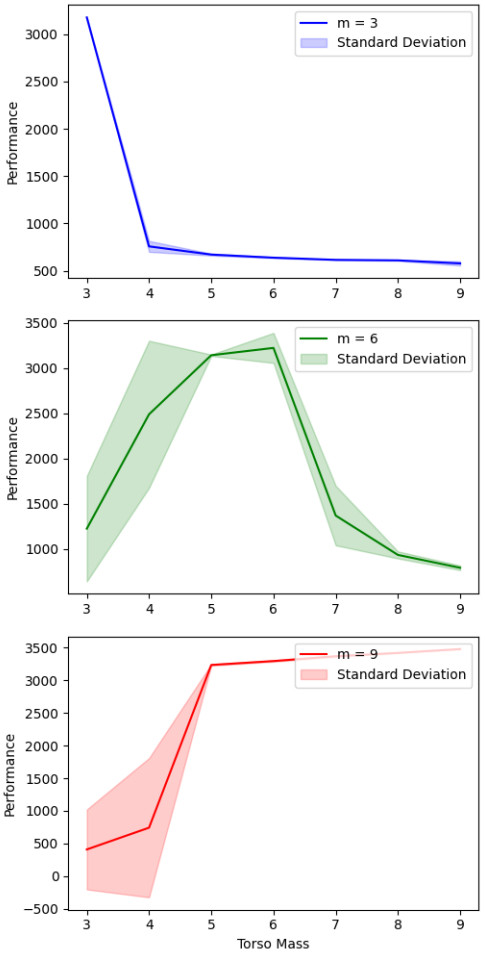


Figure 1: Evaluation results of the 3 models

1.2.2 Hyperparameter tuning

The large search space of hyperparameters and finding the optimal balance is a significant challenge in reinforcement learning [7, Chapters 3, 5, and 6] and requires considerable imperical testing. In the interest of time, the hyperparameters specified by [8] are chosen where options were provided in stable_baselines, with the caveat that these parameters are based on the Hopper-v2 enviornment [9]. The settings are as follows:

1. Multi-layer perceptron policy for actor and critic of size [64,64], relu activation (from [8] with default activation function).
2. Discount factor (γ) = 0.99
3. Generalized advantage estimator = 0.95
4. KL divergence between updates = 0.13
5. learning rate for value function = 0.0002
6. Congugate gradient maximum steps = 10
7. congugate gradient damping = 0.1

1.3 Results

The models were evaluated on a Apple MacBook Air (M1 chip, 2020 with packages as in appendix table 7).

The results are shown in figure 1 1. As expected, the performance of the model degrades as the mass of the torso moves away (in either direction) from the weight on which it was trained.

2 Part 2 : Bipedal Walker

2.1 Q1 : Model selection and training

For the task of training an agent on this environment, I opted for the on-policy PPO [10] and off-policy DDPG [11]. Both of these methods have been applied to a variety of continous action space tasks and have been benchmarked on the most common mujoco [9] environments [10, 11]. Though the benchmarks have not used the Bipedal Walker environment, presumably that these models would work adequately here as well. PPO was chosen over TRPO from task 1 for variety but also because it appears to perform better than or comparably to TRPO in a variety of enviornment [10]. The other considerations from task 1 apply equally in this task (except perhaps sample efficiency caveat in the case of PPO).

2.2 Training

2.2.1 Setup and validation

Validation and training environments (including hardware) were the same as used for task 1 (refer table 7). Testing was conducted in a Colab environment due to package installation issues in the evalutation environment used in part 1. Validation of the training parameters and other experiments were conducted in a Google Colab envrionment as in task 1 [4], adapting code from stable-baselines3 documentation[6] and [12]. Performance was monitored using the Tensorboard interface [5].

During training, both models ran for a maximum runtime of ten million environment steps in the worst case or until a reward threshold of 300 is reached. Both agents were trained three times on three different random seeds of the environment in parallel.

2.2.2 Hyperparameter tuning

Hyperparameters were tuned with the Optuna (v3.6.1) Hyperparameter optimization framework [13]. Given a sample space for multiple hyperparameters(e.g., learning rate and batch size), Optuna efficiently searches for performant hyperparameters by minimizing or maximizing an objective. For the present task, the objective was set to maximize reward. The runtime settings used for DDPG and PPO hyperparameter tuning are shown in table 3.

- **PPO**: The sample space used for PPO and 3 respectively with the values set following [12] where possible. The selected hyperparameters (returned from Optuna) is shown in table 2.
- **DDPG** : The sample space used for DDPG and the selected values are shown in table 4 and 5 respectively following [12] where possible.

3 Results

The mean reward earned by the models when evaluated over 100 episodes is shown in table 6. Note that seed seed 47 for DDPG is empty. This is because it was found that for both PPO and DDPG, training from this initial condition increased runtime when compared to other seeds (figure 2 and table 6). For DDPG, the training did not complete after over tweleve hours and was thus terminated. To test this, the code used to trian the model with seed 47 was copied with only the value changed to 35 and the model was retrained. The model trained successfully and completed ten million environment steps (spread across 4 parall environments) in under 2 hours (comparble to the rest).

The training reward and episode length for both models on the 3 different seeds are shown in figure 3. Figure 2 shows the frame rates for both models during training, an indicator of wall clock time performance.

[2]Intially, kl divergence was set to default (no limit) and thus not tuned.KL divergence is not tuned (presumably relying on clipping alone) in [12] but KL divergence limit appears to be necessary as discussed the stable-baselines3 PPO documentation.

[1]Number of trials and timesteps were 50 and 5000 respectively initially [12], which lead ineffective hyperparameters and thus was increased to presented values.

Hyperparameter	Default	Search Space
Horizon (n_steps)	2048	{8, 16, 32, 64, 128, 256, 512, 1024, 2048}
Learning rate (α)	3×10^{-4}	- log-uniform(1e-5, 1)
Num. epochs	10	{1, 5, 10, 20}
Minibatch size	64	{8, 16, 32, 64, 128, 256, 512}
Discount (γ)	0.99	{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999}
GAE parameter (λ)	0.95	{0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1.0}
Entropy coefficient	-	log-uniform(1e-8, 0.1)
Clip range	-	{0.1, 0.2, 0.3, 0.4}
Max gradient clipping	-	{0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5}
Value function coefficient	-	uniform(0, 1)
Target KL divergence [2]	-	log-uniform(0.001,0.3)
Policy network	[64,64]	-
Value network	[64,64]	-

Table 1: Hyperparameters, Default Values, and Search Spaces for PPO

Hyperparameter	Selected Value
Minibatch size	32
Horizon (n_steps)	512
Discount (γ)	0.99
Learning rate (α)	0.00018451057856134118
GAE parameter (λ)	0.99
Entropy coefficient	6.243275791612454e-06
Clip range	0.2
Num. epochs	5
Max gradient clipping	0.6
Value function coefficient	0.38040715408689363
Target KL divergence	0.1200865249782578
Policy network	[64,64]
Value network	[64,64]

Table 2: Selected Values for PPO Hyperparameters. Reward earned for these values during tuning was 315.19 (best of 50 trails).

Setting	Description	Value
Trials[1]	Maximum number of trials for finding the best hyperparameters	50
Startup trials	Number of startup trials	10
Evaluations	Evaluate every 20th of the maximum budget per iteration	20
Timesteps[1]	Maximum number of timesteps per trial	1000000
Evaluation frequency	Evaluation frequency during training in time steps	250
Evaluation episodes	Evaluate the model during 5 episodes	5
Jobs	Number of parallel jobs	4
Time-out	The maximum allowed time to finish all trials in hours	24

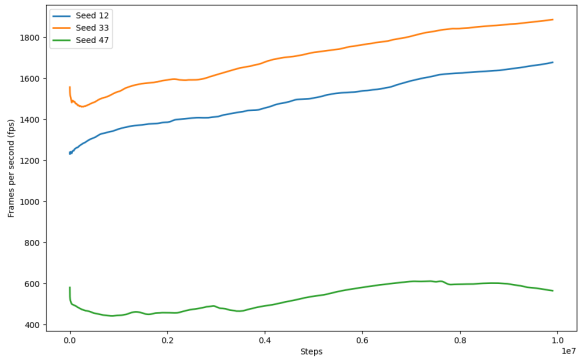
Table 3: Runtime settings for model optimization PPO and DDPG hyperparameter tuning

Hyperparameter	Default	Search Space
Learning rate (α)	-	log-uniform(1e-5, 1)
Minibatch size	-	{16, 32, 64, 100, 128, 256, 512}
Buffer size	-	{1e4, 1e5, 1e6}
Discount (γ)	-	{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999}
τ (target network update weight)	-	{0.001, 0.005, 0.01, 0.02}
Train frequency	-	{1, 16, 128, 256, 1000, 2000}
Noise type	-	{ornstein-uhlenbeck, normal}
Noise standard deviation	-	uniform(0, 1)
Policy network (pi)	[64,64]	-
Value network (vf)	[64,64]	-
Q-function network (qf)	[64,64]	-

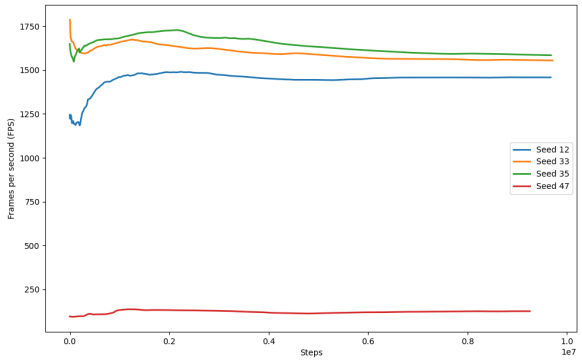
Table 4: Hyperparameters, Default Values, and Search Spaces for DDPG

Hyperparameter	Selected Value
Learning rate (lr)	2.7076207562598673e-05
Batch size	512
Buffer size	1000000
Discount (γ)	0.98
τ (target network update coefficient)	0.001
Train frequency	16
Noise type	normal
Noise standard deviation	0.8906602621112266
Noise mean	0
Policy network (pi)	[64,64]
Value network (vf)	[64,64]
Q-function network (qf)	[64,64]

Table 5: Selected Values for DDPG Hyperparameters. Reward earned for these settings during tuning was -3.91 (best of 50 trials)



(a) PPO



(b) DDPG

Figure 2: Comparison of FPS during training for PPO (a) and DDPG (b) models.

Seed	Model	Mean reward	Train runtime	Eval. runtime	Mean steps
47	**PPO**	**308.412 (35.803)**	**4:55:52**	**2.929 (0.416)**	**893.180 (53.486)**
33	PPO	300.070 (74.908)	1:28:15	2.929 (0.416)	744.590 (105.191)
12	PPO	-121.054 (3.599)	1:39:10	2.929 (0.416)	103.85 (10.726)
33	**DDPG**	**156.306 (86.143)**	**1:47:11**	**2.289 (0.645)**	**1272.28 (336.61)**
35	DDPG	-42.779 (10.779)	1:45:15	2.929 (0.410)	1600.000 (0.000)
12	DDPG	-31.334 (68.242)	1:54:18	3.010 (0.488)	1589.350 (85.533)
47	DDPG	-	-	-	-

Table 6: Evaluation results of both models over 100 episodes. All models evaluated with environment seed 1. Best perform model shown in bold. Training time shown (H:MM:SS) to completion of ten million environment steps, spread across 4 environments during training. Evaluation results shown as time taken to complete 1600 environment steps or terminal state is reached (the goal as set in the gym documentation for the environment [14]) (Eval. time) and the mean reward earned. Values shown as 'mean (standard deviation)'.

Figure 4 shows the changes in the PPO policy over time as a function of KL divergence. Figure 5 shows changes in exploratory behavior (entropy loss) and the ability of the value function to predict returns (explained variance). Finally, the loss during training is shown in figure 6.

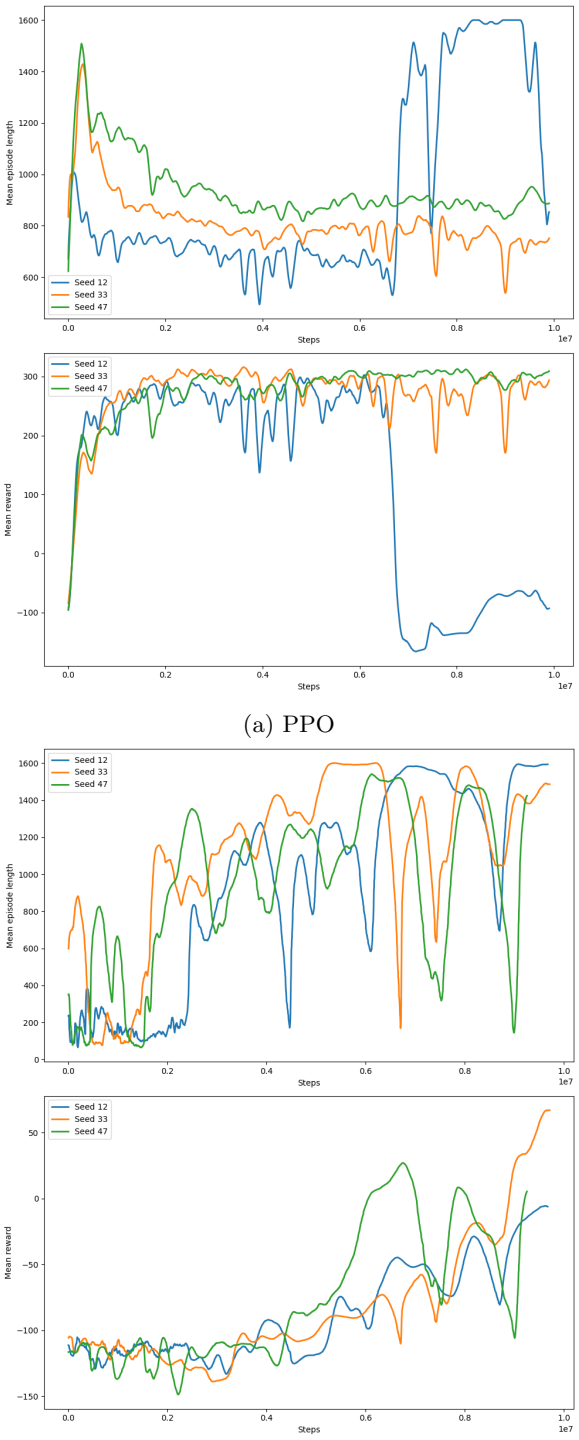
The training reward and episode length for DDPG on the 3 different seeds are shown in figure 3b. The performance of the actor and critic networks during training are shown in figure 7.

4 Discussion

First, the models differ in the time required to reach ten million environment steps across seeds. While this maybe expected due to the stochastic nature of the environment, there was a drastic difference between seed 47 and the rest. For this seed, run times were significantly lower for both models (figure 2). None the less, the PPO model appears to be able to recover and complete the task, though at a much longer time compared to the other seeds. It's possible that this initial condition generates something very different in the environment but it's not clear what that might be. Additionally, the run time (in FPS) is shown to improve over environment steps (atleast for seed 33 and 35), presumably as training is halted once the reward target is reached. The pattern appears to hold for seed 47 as well but curiously appears to drop again towards the end (again, not clear as to why). For the successful seeds, the run times appear similar sans the improving trend for PPO.

In terms of reward earned, PPO models reaches the target reward much faster, approximately between three and four million environment steps compared to the DDPG models that have not even reached 100 at ten million. However, the model on seed 12 suffers a collapose at around seven million environment steps, from which it seems unable to recover (may do so if more time was permitted). The DDPG models show an upward trend in reward and it's possible that training for much longer (than was not feasible at present) may result in the models finally reaching the target. It may be the case that faster learning (increasing the learning rate) would amelorate the slow convergence of the DDPG models. None the less, the target networks updates used in DDPG may slow down learning but with the benefit of increased stability [11].

Exploration and exploitation is managed differently between the models. For DDPG, a noise paramter is used (in this case Gaussian noise with mean 0) for the actions. For PPO, an entropy bonus is added to the objective. Here, entropy is calculated over the probability distribution of actions. Higher entropy indicates that more actions are equally likely, thus encouraging exploration. This effect can be seen in figure 5. As it is given a entropy bonus, the model is incentivized to strike a balance between increasing the entropy of the model and maximizing expected reward. In the present experiments, it can be seen that the model,



(a) PPO

(b) DDPG

Figure 3: Training performance of PPO (a) and DDPG (b) models across 3 random seeds. Values were reported every 2048 steps and smoothed with a window size of 50 for plotting.

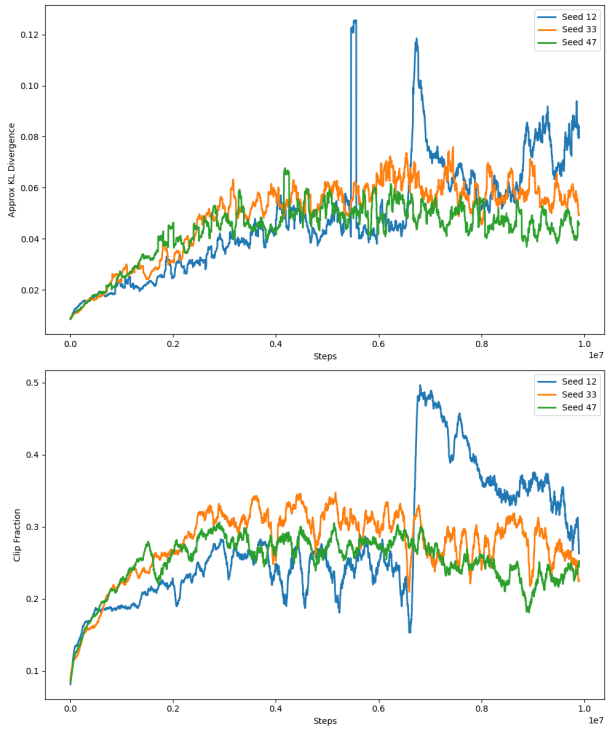


Figure 4: Approximate KL divergence (top) and clip fraction (bottom) of the three models. Values were reported every 2048 steps and smoothed with a window size of 50 for plotting.

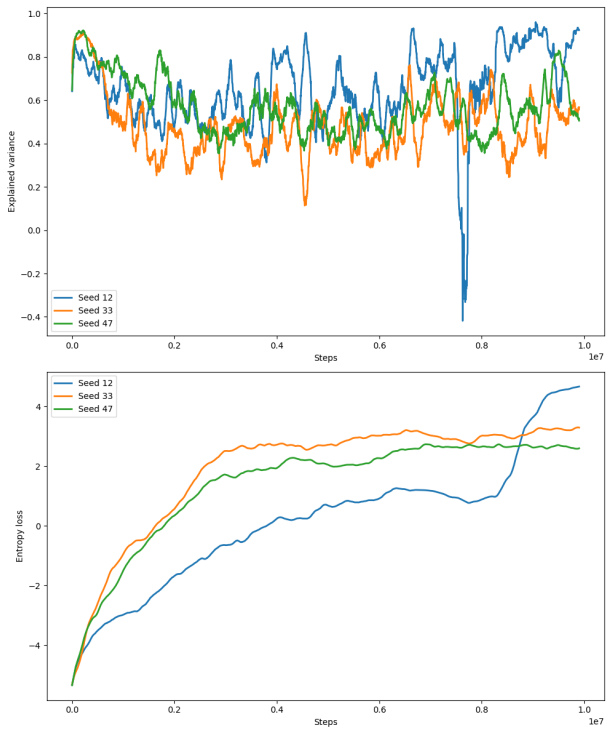


Figure 5: Change explained variance (top) and entropy loss (bottom). Values were repported every 2048 steps and smoothed with a window of size 50 for plotting.

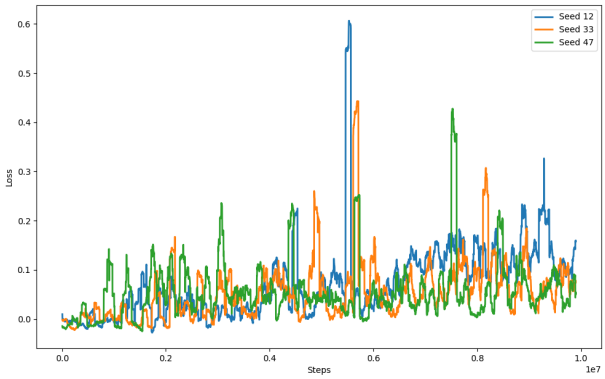


Figure 6: Training loss PPO. Values were repported every 2048 steps and smoothed with a window of size 50 for plotting.

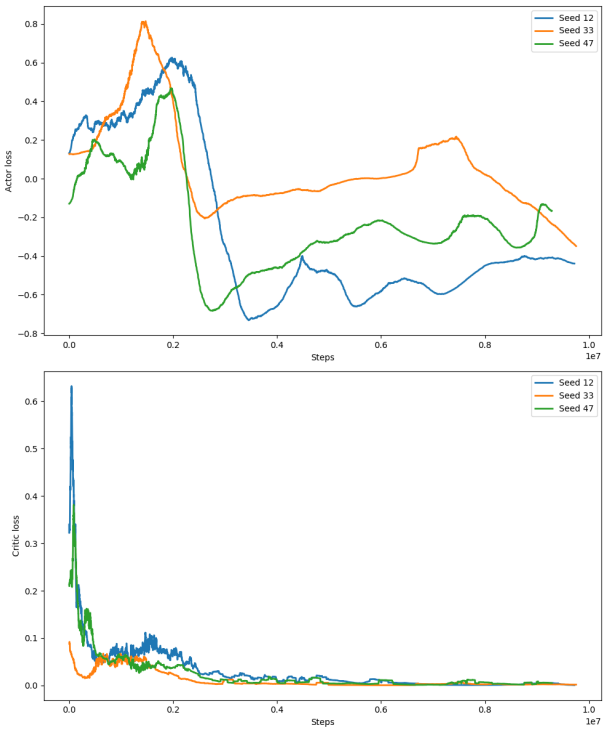


Figure 7: Training loss of the actor (top) and critic (bottom) networks of the DDPG model. Values were reported every 2048 steps and smoothed with a window size of 50 for plotting

across the 3 seeds, is able to strike this balance during training as indicated by a leveling out of the entropy loss. For DDPG, this is harder to evaluate as this is not a directly included term in the optimization objective but a tuned hyperparameter. However, looking at the training reward, it appears that DDPG is able to avoid local optimums (general increasing trend in the reward).

Finally, considering model stability during learning (as a function of mean reward earned during training) both DDPG and PPO appear relatively stable (sans the collapse seen for the PPO model in the present experiments). For PPO, this can be evaluated more clearly than the DDPG models as a function of approx KL divergence and clip fraction in figure 4. These plots can be interpreted as showing how much the action policy changes between upates. The KL divergence appears to settle at as training progresses, conceiably as a function of how the entropy bonus affects training, encouring a exploration into new actions.

To conclude, in the present experients, under the current hyperparameters, PPO appears to be desirable due to its relatively faster convergence, stability, and simplicity. DDPG while shown to be improving, was found to take much longer to train (owing perhaps to the multiple neural networks that need to be updated) though similar in terms of stability, perhaps reaching the target with more environment steps. The structure of PPO also provides insightful metrics (e.g., KL divergence and entropy loss) to better understand the model behavior (e.g., exploratory behavior). None the less, it may still be the case that under a different hyperparameter space and selected values, the DDPG performance may be improved. Next steps would be to investigate how variations in of a given hyperparameter effects the model performance while holding the others constant (which was not done here in consideration of the already lengthy training times) for DDPG. Finally, an investigation as to what occurs in the environment in seed 47 may also be insightful.

All relevant code for both environements can be found at `https://github.com/onedeeper/deepRL`

5 Technology & genrative tools statement

GPT-4 and GPT-4o by OpenAI was used for the following tasks:

1. Code for plotting functions in matplotlib.
2. Debugging error messages during development.
3. Preparing tables for the report
4. Help understanding Optuna
5. Help understanding the output of tensorboard logs
6. Code for reading and plotting from stored tensorlogs
7. Code for stable-baselines3 callbacks

A Appendix

Table 7: Software Versions Across Environments

Software	Val	Train	Eval
Python	3.10.12	3.12.2	3.11.9
gym	0.29.1	0.29.1	0.29.1
sb3	2.3.2	2.3.0	2.3.2
sb3_contrib	2.3.0	2.3.0	2.3.0
torch	2.2.1+cu121	2.2.2+cu121	2.3.0

References

[1] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[2] *Gymnasium Documentation — gymnasium.farama.org*. https://gymnasium.farama.org/environments/mujoco/hopper/. [Accessed 20-04-2024].

[3] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[4] Ekaba Bisong and Ekaba Bisong. "Google colaboratory". In: *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* (2019), pp. 59–64.

[5] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[6] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[8] Logan Engstrom et al. "Implementation matters in deep policy gradients: A case study on ppo and trpo". In: *arXiv preprint arXiv:2005.12729* (2020).

[9] Costa. *MuJoCo v2 vs v4 environments — wandb.ai*. https://wandb.ai/costa-huang/cleanRL/reports/MuJoCo-v2-vs-v4-environments--VmlldzoxNjM1OTAx. [Accessed 09-05-2024].

[10] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[11] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[12] Antonin Raffin, Jens Kober, and Freek Stulp. "Smooth exploration for robotic reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1634–1644.

[13] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[14] Oleg Klimov. *Bipedal Walker Environment Documentation*. Gym Library. Available from: `https://www.gymlibrary.dev/environments/box2d/bipedal_walker/`. 2024.