# Comparison of Reinforcement Learning Algorithms Based on OpenAI Baselines

Ju Wu (328992), Joachim Honegger (331126)

***Abstract*—In the project of advanced machine learning course, we study the theory and mechanism of two reinforcement learning algorithms, A2C and PPO2 and their application for multiple tasks e.g., Cart pole pendulum and lunar landing task. To do so, we build up a pipeline able to train and manage our models and environments. From the experimental results obtained, we compute multiple reliability metrics to be able to lead a discussion about the performance evaluation of the algorithms.**

***Index Terms*—Reinforcement learning, A2C, PPO2, Performance evaluation, Reliability evaluation, Dispersion, Risk**

## I. Introduction

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. RL is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. And RL differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). RL issues are modeled by the use of the Markov Decision Process (MDP), a framework that permits agents to learn observations, actions, and goals. Learning is performed by maximizing a reward function via trials and errors. For this survey, we are using OpenAI Baselines, an open-source framework implementing some model-free deep RL algorithms. We chose to study and evaluate two of these commonly used algorithm: A2C and PPO2.

## II. Theoretical Background of Reinforcement Learning

### A. Mechanism of Advantage Actor-Critic (A2C)

Actor-Critic learning employs parameterized actor to compute continuous actions without the need for optimization procedures on a value function. It uses the critic to supply the actor with low-variance knowledge of the performance. At once, the lower variance is traded for a larger bias at the beginning of the learning when the critic's estimates are far from accurate and actor-critic methods usually have good convergence properties, in contrast to critic-only methods. Actor-critic algorithms maintain two sets of parameters, one is the critic parameters $\omega$ to approximate action-value function under current policy, and the other is actor (policy) parameters $\theta$. Actor-critic algorithms follow an approximate policy gradient that use critic to update Q-function parameters $\omega$ like in policy evaluation and actor to update policy gradient $\theta$ in direction suggested by critic. We study Advantage Actor Critic(A2C) model as one kind of Actor-Critic learning methods in this project. We study Advantage Actor Critic(A2C) model as one kind of Actor-Critic learning methods in this project. In the critic of A2C, advantage value can significant reduce variance of policy gradient. This function is used as follow:

$$A^{\pi\theta}(s,a) = Q^{\pi\theta}(s,a) - V^{\pi\theta}(s) \tag{1}$$

Intuitively, the advantage value function measures the effectiveness to take a specific action compared to the average, general action at the given state. Thus two function approximators and two parameter vectors $\omega$ and $v$ need to be employed to estimate $V(s)$, $Q$ and the advantage value function:

$$\begin{aligned} \hat{V}^{\pi\theta}(s) &= V^{v}(s) \\ \hat{Q}^{\pi\theta}(s,a) &= Q^{\omega}(s,a) \\ A(s,a) &= Q^{\omega}(s,a) - V^{v}(s) \end{aligned} \tag{2}$$

Both of the value functions can be updated by Temporal Difference (TD) learning.

The A2C model class on Baselines is based on [2]. The policy gradient of A2C is

$$\nabla J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\omega}(s,a)] \tag{3}$$

Considering the expression of $Q(s_t,a_t)$ below.

$$Q(s_t,a_t) = E[r_{t+1} + \gamma V(s_{t+1})] \tag{4}$$

only one neural network for $V(s)$ parameterized by $v$ is needed to represent the value of advantage function as

$$A(s_t,a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t) \tag{5}$$

then equation 3 can be rewritten as

$$\nabla J(\theta) \sim \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s,a)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)) \tag{6}$$

### B. Mechanism of PPO

The Proximal Policy Optimization (PPO) [3] algorithm is based on Policy Gradient Method (PGM). That rely on optimizing a policy directly (online policy) by maximizing the total expected reward. To do so, the algorithm are doing a gradient update from a batch of experience. It starts to compute the policy gradient loss:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[ \log \pi_\theta (a_t \mid s_t) \hat{A}_t \right] \tag{7}$$

$\widehat{A}_t$ is an estimate of the advantage function. It is the relative value of the current action and purpose to know if the action that the agent took was better or not. If $\widehat{A}_t$ is positive it means that the action that the agent took from the sample trajectory is better so it increases these actions probabilities. At the contrary if it is negative it decreases these actions probabilities. By multiplying $\widehat{A}_t$ by the log of the policy actions $\pi_\theta\left(a_t \mid s_t\right)$ (neural network that takes the observed states of the environment and output the actions) and take the expectation (so the policy is computed over batches of trajectories) we get the final objective function that is used for this gradient descent algorithm (vanilla policy gradients).

An issue in RL is that the data distributions of our observations and rewards are constantly changing during training and lead to instability. To make sure the updated policy does not go too far from the current policy, PPO trains the policy in the same region over the batches. Indeed, this method used Trust Region Policy Optimisation (TRPO) [4]. TRPO algorithm uses KL constraint to the optimisation objective. However, this constraint can lead to undesirable training behaviour and complex tuning. To avoid this constraint, PPO replaced it by the Clipped "Surrogate" Objective:

$$L^{CLIP}(\theta)=\hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{8}$$

where $r_t(\theta)$ is a probability ratio to estimate if the action is more likely now than the old version (before the last gradient step) and $\epsilon$ a hyperparameter.

When $A>0$, the action had a better than expected return on the outcome. The loss function (8) flatten out when $r_t(\theta)$ gets too high because the action is a lot more likely in this current policy than before. However, we don't want to overdue the policy update (not change the policy too much) so we clipped it to a maximal value. It is the same but reverse when $A<0$.

The final loss function that is used to train an agent is:

$$L_t^{CLIP+VF+S}(\theta)=\hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta)-c_1 L_t^{VF}(\theta)+c_2 S\left[\pi_\theta\right]\left(s_t\right)\right] \tag{9}$$

PPO added to the objective function an error term on value estimation and an entropy term for explorations. $c_1$ and $c_2$ are hyperparameters to weight the contribution of these two last terms. Finally, as A2C, this loss function is implemented in an Actor-Critic Style algorithm.

In conclusion, PPO is one of the most popular RL algorithms because it combines easy tuning, high performance (stability, reliability) and outperforms most of TRPO algorithms. This state-of-the-art method is used for a wide range of RL tasks. We will use PPO2, a full implemented Tensorflow version of OpenAI made for GPU on baselines library.

## III. Pipeline implementation

### A. Implementation of Algorithms

In this section, we make detailed description of the procedure for implementation of the chosen reinforcement learning algorithms. The training and testing platform for the implementations is Google Colab. With the help of the examples given by the stable-baseline documentation, we build up a complete pipeline to train, monitor in live the training, load the trained agent, evaluate, and simulate. To use the pipeline, you need to set up the algorithm and the environment you want to use. As Mninst or Imagenet did for classification, we are using OpenAI environments. These environments provided all the necessaries to benchmark the different algorithms available on stable-baselines [7], notably by the fact that the Reward function is already adjust. You can choose between 12 RL algorithms and 143 environments [6] (Algorithms, Atari, Box2D, Classic control and Toy text) using our pipeline on Google collab. Finally, you can set-up some hyperparameters regarding the training (learning rate, total time steps), the model (model policy). For the RL algorithm parameters, we chose to keep most of the default values of PPO2 since the are consistent with the used environments. Then, we can visualize the training via Tensorboard and monitor it (decide when to save a trained model) via callback functions. We implemented a callback function to save a checkpoint every 5000 steps (CheckpointCallback) and a callback function to evaluate the model periodically and save the best one based on the best training reward (EvalCallback). We can then load and evaluate the best model based on its mean reward. Finally, the pipeline includes a simulation where you can see a subsets of trained agent behaviours in the environment.

### B. Selection of Datasets and Environments

In this section, we first discuss the principle of selection of datasets and environments and then make choice of the experimental situations employed to evaluate the reinforcement learning algorithms.

RL involves two fundamental strategies: exploration, where we gather more information that might lead us to better decisions in the future and exploitation, where we make the best decision given current information. In the RL setting, no one gives us some batch of data like in supervised learning. We're gathering data as we go, and the actions that we take affects the data that we see, and so sometimes it's worth to take different actions to get new data. the environments should simulate the tasks in real world as much as possible to facilitate the RL algorithms trade off exploration and exploitation.

In this project, we use two physical tasks to evaluate the selected algorithms. The first one is the CartPole-v1 environment figure 1.

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is

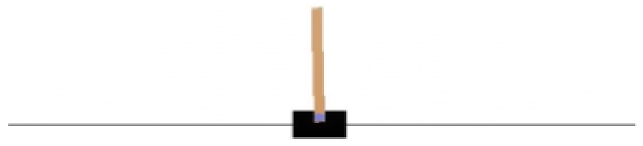

Fig. 1: render of the CartPole-v1 task

controlled by applying a discrete force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

The second environment we employ is LunarLander-v2 2

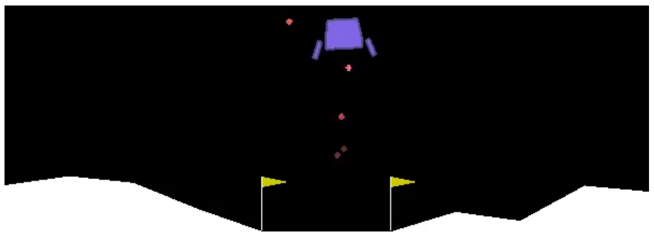

Fig. 2: render of the LunarLander-v2 task

The landing pad is always at coordinates (0,0) and coordinates are the first two numbers in state vector. In this task, the reward for moving from the top of the screen to landing pad with zero speed is about 100 to 140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10 points. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions are available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

## IV. Experiments and Evaluation

In this section, we demonstrate the experimental results obtained after the implementation procedure of Section. III.

### A. Experimental Procedures

We started to do some training to tune our hyperparameters. By looking the losses over epoch, we observed that the learning rate was too important and the model was not able to converge. We thus added a Schedules function. Schedules are used as hyperparameter to change value of the learning rate over time.

The learning rate now changes according to a linear function as we can observe Figure 3. This guaranty us better and faster convergences.

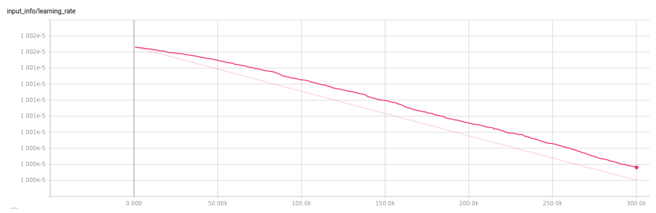

Fig. 3: learning rate of PPO2 per step for the LunarLander-v2 task

We record below the chosen hyperparameters we used to do the different runs for each algorithms.

TABLE I: Table of A2C hyper-parameters

| ***A2C*** | **Lunar-lander-v2** | **Cartpole** |
|---|---|---|
| number of runs | 3 | 3 |
| Policy | MLP | MLP |
| batch size | 5 | 5 |
| learning rate | linear function starting at 0.01 | idem |
| Value function term c1 | 0.25 | 0.25 |
| entropy term c2 | 0.01 | 0.01 |
| Discount factor | 0.99 | 0.99 |
| gradient clip max value | 0.5 | 0.5 |
| total timesteps | 300000 | 300000 |
| train env seed | 42 | 42 |
| model seed | random | random |

TABLE II: Table of PPO2 hyper-parameters

| ***PPO2*** | **Lunar-lander-v2** | **Cartpole** |
|---|---|---|
| number of runs | 3 | 3 |
| Policy | MLP | MLP |
| batch size | 128 | 128 |
| learning rate | linear function from 5e-3 to 1e-5 | idem |
| Value function term c1 | 0.5 | 0.5 |
| entropy term c2 | 0.01 | 0.01 |
| Discount factor | 0.99 | 0.99 |
| gradient clip max value | 0.5 | 0.5 |
| total timesteps | 300000 | 300000 |
| train env seed | 42 | 42 |
| model seed | random | random |

### B. Experimental Results

In this section, we list some primary experimental results of the selected RL algorithms for different task.

In figure 4, we can observe at primary results that better results were obtained with PPO2 algorithm for the CartPole but in figure 5 better ones have been achieved with A2C for the lunarLander.

We launched several runs to establish a good number of time-steps (not too much for training time reason but enough to observe a loss convergence of the policy). We set the time

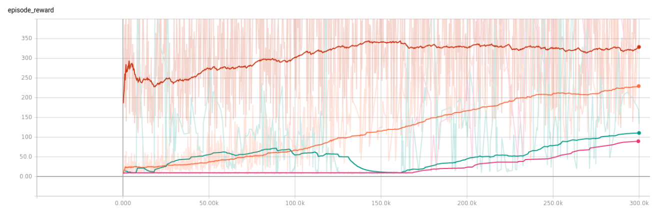
Fig. 4: smoothed curves of CartPole-v1 training (episode Reward per step) - PPO2 (red and orange) - A2C (green and pink)

step at 300k steps which is enough to obtain a good reward as we can observe in the PPO paper [3] where they trained some environments for one million time steps and as we can see in the figure 5 bellow.

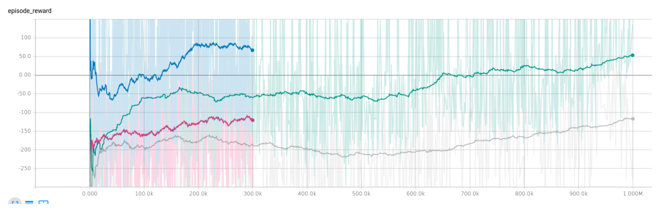
Fig. 5: smoothed curves of LunarLander-v2 training (episode Reward per step) - PPO2 (pink(300k) and grey(1000k)) - A2C (blue(300k) and green(1000k))

We used the same seed of environment and parameters for all the training in figures 4-7. However, the training plots are different. Indeed, the seed of the pseudo-random generator changes so the results vary from one run to another. Notice that with a fixed seed of the model, we were able to reproduce almost the same results.

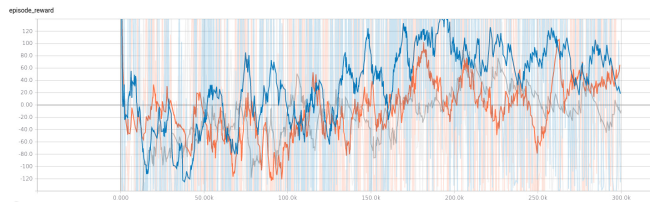
Fig. 6: Curves of LunarLander-v2 training (episode Reward per step) for 3 runs of A2C

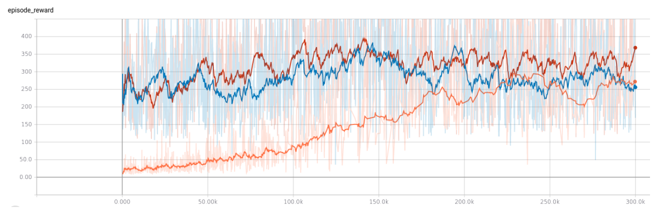
Fig. 7: Curves of Cartpole training (episode Reward per step) for 3 runs of PPO2

In RL, the data used to train the agent is collected through interactions with the environment by the agent itself. For this reason, it was necessary to perform different runs as we can see Figure 6 and 7. Therefore, we were able to obtain quantitative results to evaluate the performance of an algorithm. For example, if one agent collects low quality data it will not improve and continue to have a bad reward.

To evaluate our models, we compute the mean reward of 10 episodes in another seed environment. As example for the CartPole task for the PPO2 run above (red plot), we obtained a mean reward before training of 32.80 +/- 17.63 and 450.8 +/- 27.60 for the trained agent. In the same way for the A2C for the same task, we obtained a mean reward of 9.80 +/- 0.60 and 405.8 +/- 18.42 after.

### C. *Performance Evaluation*

In this section, we evaluate the performance of the two algorithms for the obtained experimental results. Lack of reliability is a well-known issue for RL algorithms. To facilitate the comparison of the performance of two RL algorithms under two environments, the reliability of RL algorithms represented by various metrics across different dimensions is measured via analyzing the training curves collected trough the experiments. The standardized measures of reliability can benefit the selection of algorithms for specific environments and tasks via identifying particular strengths and weaknesses of algorithms across various aspects.

We have two measures of variability as follows [5]:

- Dispersion: it indicates the width of the distribution of the reward curves. The robust statistics like the *inter-quartile range*(IQR) more appropriate for asymmetric distributions than *median absolute deviation from the median*(MAD) is employed in our cases.
- Risk: we are concerned about the worst-case scenarios in many cases. Therefore, as a complement of measures of dispersion like IQR, which cuts off the tails of the distribution, we employed the measure of risk like *Conditional Value at Risk*(CVaR) known as “expected short-fall”. CVaR measures the expected loss in the worst-case scenarios, defined by some quantile-$\alpha$. It is computed as the expected value in the left-most tail of a distribution. For instance, we use the following definition for the CVaR of a random variable $X$ with a given quantile-$\alpha$:

$$CVaR_{\alpha}(X) = E[X|X \leq VaR_{\alpha}(X)] \qquad (10)$$

  where $\alpha$ belongs to $(0, 1)$ and $VaR_{\alpha}$(Value at Risk) is the quantile-$\alpha$ of the distribution.

In the setting of evaluation during training, one desirable property for an RL algorithm is to be stable ”across time” within each training run. In general, smooth monotonic improvement is preferable to noisy fluctuations around a positive trend, or unpredictable swings in performance. This type of stability is important for several reasons. First, during learning, especially when deployed for real applications, it can be costly or even dangerous for an algorithm to have unpredictable levels of performance even in cases where plenty of poor performances do not directly cause harm. For instance, if the model is trained in simulation, high instability implies that algorithms have to be check-pointed and evaluated more frequently in order to catch the peak performance of the algorithm but can be expensive. Furthermore, while training, it can be a waste of computational resources to train an unstable algorithm that tends to forget previously learned behaviors.

During the training, RL algorithms should have easily and consistently reproducible performances across multiple training runs. Depending on the components that we allow to vary across training runs, this variability can encapsulate the algorithm's sensitivity to a variety of factors, such as random seed and initialization of the optimization, random seed and initialization of the environment, implementation details, and hyper-parameter settings. Depending on the goals of the analysis, these factors can be held constant or allowed to vary, in order to disentangle the contribution of each factor to variability in training performance. High variability on any of these dimensions leads to unpredictable performance, and also requires a large search in order to find a model with good performance. Therefore the following criteria for reliability of RL algorithm performance are employed:

- Dispersion across Time (DT) i.e., IQR across Time: it isolates higher-frequency variability, rather than capturing longer-term trends and avoid metrics being influenced by positive trends of improvement during training, detrending is employed before computing dispersion metrics. The final measure consisted of inter-quartile range (IQR) within a sliding window along the detrended training curve.
- Short-term Risk across Time i.e., (SRT)CVaR on Differences: the most extreme short-term drop over time is measured over time. CVaR is carried out to the changes in performance from one evaluation point to the next. Then, the expected value of the distribution below the $\alpha$-quantile is computed, which gives us the worst-case expected drop in performance from one point of evaluation to the next during training.
- Long-term Risk across Time (LRT) CVaR on Drawdown: it captures whether an algorithm has the potential to lose a lot of performance relative to its peak, even on a longer timescale such as over an accumulation of small drops. CVaR is applied to the Drawdown, and the Drawdown at time $T$ is defined as $Drawdown_T = R_T - max_{t\leq} R_t$. Thus LRT can capture unusually large drops in performance that occur within short-term and over longer timescales.
- Dispersion across Runs (DR) IQR across Runs: robust statistics like IQR is measured across training runs at a set of evaluation points.
- Risk across Runs (RR) CVaR across Runs: CVaR is applied to the final performance of all the training runs, which gives a measure of the expected performance of the worst runs.

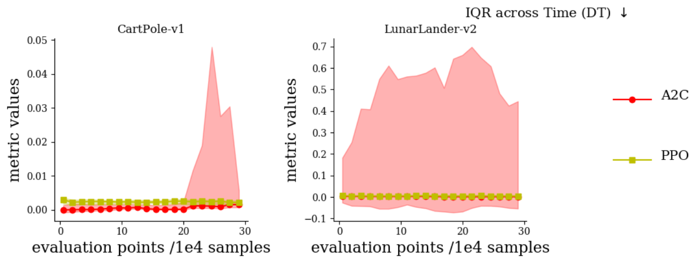

Fig. 8: IqrWithinRuns for per task at evaluation points

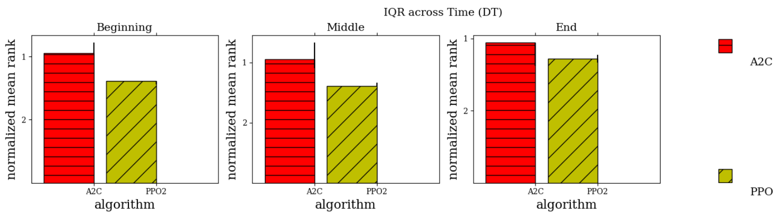

Fig. 9: Mean rankings of IqrWithinRuns for two algorithms under three training phases

We analyze a subset of the OpenAI stable baselines: 2 algorithms (A2C, PPO2) and 2 tasks (CartPole-v1 and LunarLander-v2) and 3 training runs per combination(algo&task) with the configuration and hyper-parameters shown in Table. I-II. The results of reliability evaluation are demonstrated in Fig. 10-19.

Some customized parameters that can affect the results of reliability evaluation have been clearly specified, i.e., window size (for Dispersion across Time): 5000, threshold frequency for low-pass and high-pass filtering (Dispersion across Time, Dispersion across Runs): 0.01Hz, interval of evaluated points: 15000, $\alpha$: 0.05, and held constant across runs of each combination (algo&task) for meaningful comparisons. Because different environments have different ranges and distributions of reward, we must be careful when aggregating across environments or comparing between environments. Thus, if the analysis involves more than one environment, the per-environment median results for the algorithms are first converted to rankings, by ranking all algorithms within each task. To summarize the performance of a single algorithm across multiple tasks, we compute the mean ranking across tasks.

Results of dispersion across time for per task are shown in Fig. 8-9. Better reliability is indicated by less positive values and the x-axes indicates the number of environment steps. A2C has better IqrWithinRuns than PPO2 under CartPole-v1 task, the two algorithms have the similar IqrWithinRuns under LunarLander-v2 task, and A2C has higher mean rank across tasks than PPO2 through all of three training phases.

Results of dispersion across runs are shown in Fig. 10-11. Better reliability is indicated by less positive values. PPO2

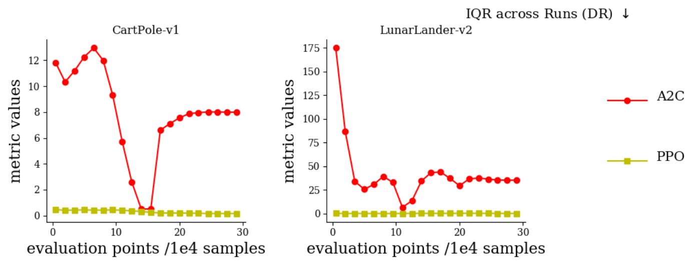

Fig. 10: IqrAcrossRuns for per task at evaluation points

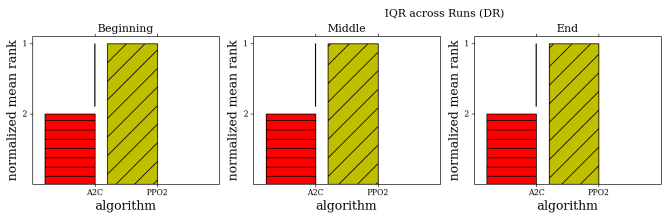

Fig. 11: Mean rankings of IqrAcrossRuns for two algorithms under three training phases

has better IqrAcrossRuns than A2C under both tasks and higher rank of IqrAcrossRuns than A2C under all three training phases.

Results of short-term risk across time are shown in Fig. 12-13. Better reliability is indicated by more positive values. A2C is slightly better than PPO2 for short-term risk under CartPole-v1 task while PPO2 is slightly better than A2C for short-term risk under LunarLander-v2 task so that they have nearly the same average rank across tasks. But PPO2 is more stable to have less variance of short-term risk across runs.

Results of risk across runs are shown in Fig. 14-15. Better reliability is indicated by more positive values. PPO2 has better risk across runs than A2C for both tasks during all of three training phases.

Results of long-term risk across time are shown in Fig. 16-17. Better reliability is indicated by less positive values. A2C has slightly better long-term risk than PPO2

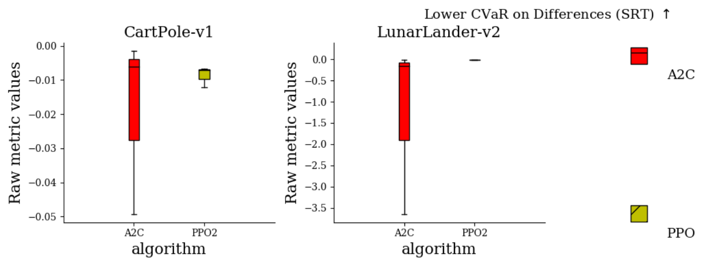

Fig. 12: Short-term risk across runs per task for two algorithms

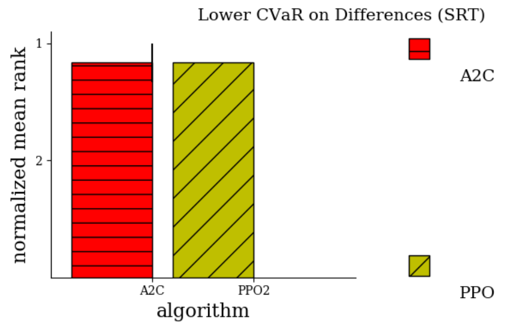

Fig. 13: Mean rankings of short-term risk across runs for two algorithms

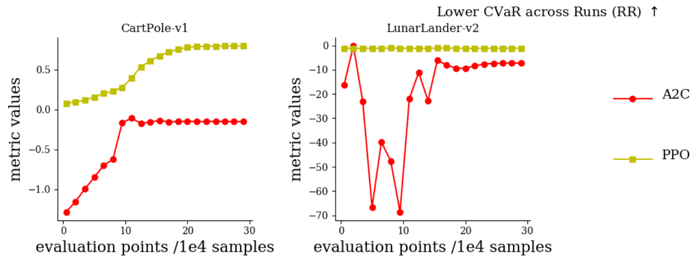

Fig. 14: Risk across runs for per task at evaluation points

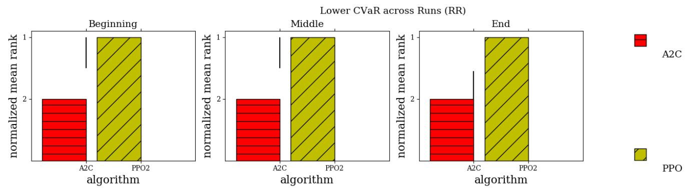

Fig. 15: Mean rankings of risk across runs for two algorithms under three training phases

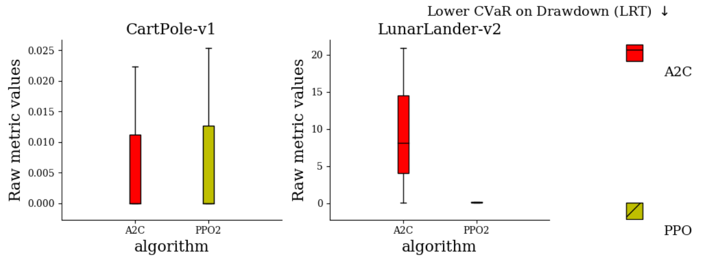

Fig. 16: Risk of drawdown per task for two algorithms

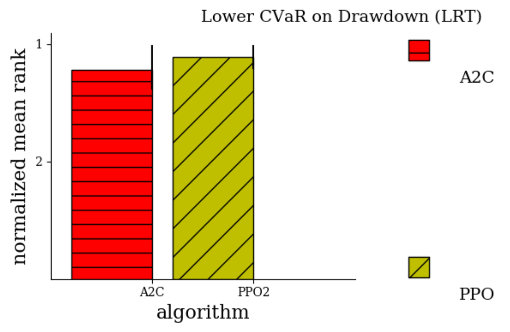

Fig. 17: Mean rankings of risk of drawdown across runs for two algorithms

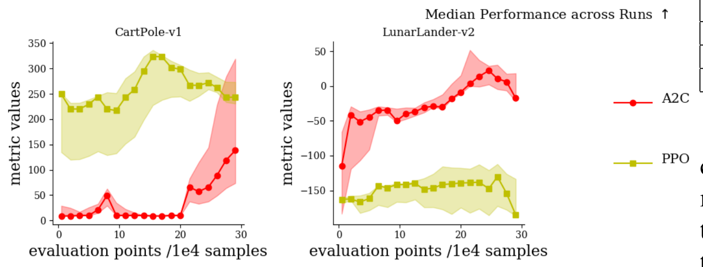

Fig. 18: Median performance across run per task at evaluation points

under CartPole-v1 task, while PPO2 has better and more stable long-term risk than A2C under LunarLander-v2 task. Therefore, PPO2 has slightly higher normalized mean rank across tasks than A2C.

Results of median performance during training are shown in Fig. 18-19. Better performance is indicated by more positive values. PPO2 has better median performance than A2C under CartPole-v1 task while A2C has better median performance than PPO2 under LunarLander-v2 task and they have nearly the same mean rank of median performance across tasks during all of three training phases.

To compare algorithms on per-run metrics, first per-run metrics are evaluated for each run then they are normalized and ranked within each task, finally the mean ranks and corresponding confidence(95%) intervals (upper and lower bounds) of each algorithm across tasks are computed. To

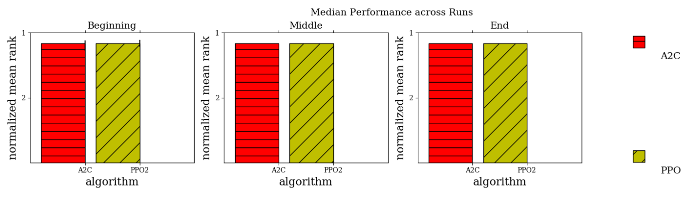

Fig. 19: Median rankings of median performance across runs under three training phases

TABLE III: table of normalized rank of each reliability metrics for A2C

| Algorithms | A2C | | |
|---|---|---|---|
| Rank metrics | Upper bound | Lower bound | Mean across tasks |
| Dispersion across time | 2.33, 2, 3.2 | 2.83, 3.28, 4.13 | 1, 1, 1.1 |
| Dispersion across runs | 1, 1, 1 | 1.9, 1.9, 1.9 | 2, 2, 2 |
| Short-term risk | 3 | 4 | 1.2 |
| Long-term risk | 3.02 | 4.17 | 1.2 |
| Risk across runs | 1, 1, 1.55 | 1.5, 1.5, 2 | 2, 2, 2 |
| Median performance | 3.35, 3.5, 3.5 | 3.83, 3.5, 3.5 | 1.2, 1.2, 1.2 |

TABLE IV: table of normalized rank of each reliability metrics for PPO2

| Algorithms | PPO2 | | |
|---|---|---|---|
| Rank metrics | Upper bound | Lower bound | Mean across tasks |
| Dispersion across time | 4.17, 4, 3.68 | 4.32, 4.17, 4 | 1.4, 1.4, 1.3 |
| Dispersion across runs | 1, 1, 1 | 1, 1, 1 | 1, 1, 1 |
| Short-term risk | 3.5 | 3.5 | 1.2 |
| Long-term risk | 3.03 | 3.63 | 1.13 |
| Risk across runs | 1, 1, 1 | 1, 1, 1 | 1, 1, 1 |
| Median performance | 3.33, 3.5, 3.5 | 3.65, 3.5, 3.5 | 1.2, 1.2, 1.2 |

compare algorithms on across-run metrics, first the across-run metrics are evaluated for each combination (task, algorithm), then they are normalized and ranked within each task, finally the mean ranks and corresponding confidence(95%) intervals (upper and lower bounds) of each algorithm across tasks are computed. The statistics of the above metrics for A2C and PPO2 are listed in Table. III-IV and if the number of values in a cell is 3, it means that the metrics have been tested during three training phases in virtue of sampled evaluation points.

## *D. Discussion*

In this part, we integrate the experimental results we obtained and analyse the performance of RL algorithms we tested with various settling and under different tasks in virtue of evaluation above.

Although it is explicit that A2C and PPO2 have different training performance on the selected two tasks, they are on par with each other with respect to some of the reliability metrics. We can observe that A2C and PPO2 have the similar ranks of short-term and long-term risks across time but risks of PPO2 have less variance than that of A2C, which means that they have the similar average performance for the worst-case scenarios on different tasks. However, PPO2 is more stable than A2C in this case across different tasks and runs with various settings. We can also see that PPO2 has better reliability performance than A2C with respect to dispersion across time and risk across runs, which infers that A2C has the larger average variance of performances and the worse average final performance across the training runs across training runs under different tasks. But A2C has better dispersion across time than PPO2, which means A2C has the smaller distribution for differential of training curves.

The differing patterns in these metrics demonstrates that reliability is a separate dimension that needs to be inspected separately from mean or median performance i.e., two algorithms may have similar mean and median performance but may nonetheless significantly differ in reliability, as with A2C and PPO2 above. Additionally, these results demonstrate that reliability along one axis does not necessarily correlate with that on other axes so that we can select different dimensions of reliability to compare based on the requirements of the problem at hand.

## V. Conclusion

We have seen the mechanisms of A2C and PPO2 and their implementations of OpenAI baselines, the selection of tasks and realized experiments on them. We have presented a number of metrics, designed to measure different aspects of reliability of RL algorithms.

Additionally, we presented examples of applying these metrics to three runs with slightly different settings for each combination(algorithm&task), and showed that these metrics can reveal strengths and weaknesses of an algorithm that are obscured when we only inspect mean or median performance.

## Acknowledgment

Thanks to Professor, TAs, OpenAI stable baseline, Google Colab and Github and so on.

## References

[1] Eslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research, Volume 4, 1996.

[2] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). DIO: 1602.01783.

[3] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. DOI: 1707.06347.

[4] SCHULMAN, John, LEVINE, Sergey, ABBEEL, Pieter, et al. Trust region policy optimization. In : International conference on machine learning. PMLR, 2015. p. 1889-1897. DOI: 1502.05477

[5] Stephanie CY Chan, Sam Fishman, John Canny, Anoop Korattikara et al. Measuring the Reliability of Reinforcement Learning Algorithms. In : International Conference on Learning Representations, Addis Ababa, Ethiopia, 2020

[6] OpenAI environments: https://gym.openai.com/envs

[7] Stable-baselines documentation: https://stable-baselines.readthedocs.io/