Policy-based Reinforcement Learning on the CartPole Environment: Assignment 3 for Reinforcement Learning 2022

Kutay Nazli [1] **Filip Jatelnicki** [2] **Ivan Horokhovskyi** [2] **Goddess of Probability** [3]

Abstract

In Reinforcement Learning problems that have high-dimensional or continuous action spaces, the value based approach fails to replicate its performance in problems with smaller dimensions. Therefore, instead of learning the Q-value function and having a separate algorithm to pick the optimal next action, it is possible to learn instead the policy. This work considers the simple *CartPole-v1* environment in *Gym* and implements the REINFORCE and actor-critic algorithms to perform gradient-based policy search to solve the environment. Additionally, an ablation study is performed to compare the REINFORCE algorithm to the more sophisticated actor-critic approach, and to investigate the performance impact of bootstrapping and baseline-subtraction in the latter.

1. Introduction

1.1. Deep Reinforcement Learning

The field of Reinforcement Learning (RL) describes the use of the active interactions of an agent with its environment to learn the optimal behavior or achieve a specific goal in that environment. RL problems in general can be formulated as an MDP, a tuple $< S, A, T, R, \gamma >$ where S is the set of states s in the environment, A the set of allowed actions a, $T : S \times A \times S' \to [0, 1]$ is the transition function, $R : S \times A \times S' \to \mathbb{R}$ is the reward function and γ is the discount factor. We seek an optimal policy for the agent to follow, defined as $\pi^* : S \to A$ that maximizes the expected discounted sum of the future rewards $Q(s, a) = \mathbb{E}_\pi[\sum_i \gamma^i r_{t+i+1} | s_t = s, s_t = a]$.

[1]Leiden Observatory, Leiden University, Leiden, The Netherlands [2]LIACS, Leiden University, Leiden, The Netherlands [3]Random State University, MC, Monte Carlo. Correspondence to: <>.

Most simpler Deep RL problems that do not have continuous action spaces can be solved with the previously investigated value-based approach where the neural network(s) (NNs) are trained to learn the value-function that maps between the state-action pairs and the Q-value defined above. However, this approach does not perform in problems where there is a continuous or high-dimensional action space, such as faux or real robotics or electronics problems where it is possible to control various actuators or thrusters with high precision/accuracy. Then, even if the NN learns the Q-value function well, it is a non-trivial problem to then pick an action using this approximate function while still considering the balance of exploration/exploitation and the bias/variance trade-off incurred in the process. Thus, a simpler and better approach in these problems is to "skip a step" if you will and instead learn the policy instead, eliminating the requirement of externally optimizing a policy. This is the approach that is called *policy-based* RL.

1.2. CartPole Environment

The *CartPole-v1* environment in *Gym* consists of a cart that freely moves on a single horizontal axis on a frictionless surface, and a rigid pole attached to the cart by an un-actuated joint. The goal of the agent is to apply force to the car to then attempt to keep the pole upright as long as possible without the pole tipping more than $15°$ from the vertical or the cart getting further than 2.4 units from the center.[1] Every timestep where the pole is kept upright returns a reward of 1, and the environment terminates after 500 successful steps. The solution is considered to be achieved if the average of past 100 runs is above an episode reward of 475. The state space S for this problem consists of the tuple $< x, \dot{x}, \theta, \dot{\theta} >$, the horizontal position and velocity of the cart and the angular position and velocity of the pole respectively. Even though this environment can be easily solved by previously mentioned value-based methods since the action-space is limited, it is still interesting to implement policy-based methods to draw comparisons between the schools in RL.

[1]Definition as provided by OpenAI: https://gym.openai.com/envs/CartPole-v1/

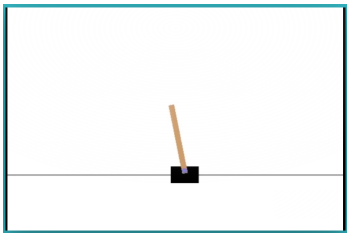

Figure 1. The *CartPole-v1* environment used in this report. Note the aforementioned horizontal track, the cart and the attached pole able to swing from side to side. Source: *Gym*, OpenAI

1.3. System settings

To ensure the experiments' reproducibility and fairness, all experiments were conducted on a single computer. However, the program was tested to work on other operating systems like Linux x64 and macOS. Setup (environment) is as follows:

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- Windows 10 (x64)
- Python 3.9.7
- Conda 4.11.0 (Torch 1.11.0, Numpy 1.22.3, Gym 0.23.1, Matplotlib 3.5.1, WandB (optional))

2. Methodology

2.1. Monte Carlo Policy Gradient (REINFORCE)

The objective of all of RL, but even more so that of policy-based RL is to find the optimal policy weights θ^* that maximizes the expected return of the agent:

$$\theta^* = \text{argmax}_\theta J(\theta) \tag{1}$$

where $J(\theta)$ is the expected return of the start state of the environment under the policy defined by weights θ. Given that the expected return is differentiable, one approach to find the best policy would be to calculate the gradient of the expected return and use the idea that the derivative of a function is equal to 0 at a maximum to find θ^*. In this vein, we can use the following update to perform *gradient ascent*:

$$\theta_{t+1} = \theta_t + \eta \cdot \nabla_\theta J(\theta) \tag{2}$$

where we update the weights of the network using the gradient of the return with some learning rate η. This is the core idea behind a gradient-based policy search.

The Monte Carlo Policy Gradient or REINFORCE algorithm uses this idea and $J(\theta) = \mathbb{E}_{h\sim p_\theta(h)} R(h)$ where now the expected return is defined as the total expected reward over some trajectory in the environment h sampled using the policy p_θ. With this knowledge and using the "log-derivative trick":

$$\nabla_\theta \mathbb{E}_{x\sim g_\theta(x)} f(x) = \mathbb{E}_{x\sim g_\theta(x)}[f(x) \cdot \nabla_\theta \log g_\theta(x)] \tag{3}$$

we can arrive at the following expression for the gradient used for the update of the NN:

$$\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \left[\sum_{t=0}^{n_i} R(h_t^i) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{4}$$

where π_θ is the action-selection policy and the mean of the gradient is calculated over M trajectories to get rid of sum of the inherent variance in generating random trajectories of the environment h^i. Realize that this is a Monte Carlo update since there is no bootstrapping for the reward values and full trajectories are used to calculate the policy gradient.

In this work, the PyTorch framework is used to implement the policy network. Since PyTorch uses loss minimization to update the weights of the NN, the following loss is defined:

$$L(\theta) = -\frac{1}{M} \sum_{i=1}^{M} \left[\sum_{t=0}^{n_i} \gamma^t R(h_t^i) \log \pi_\theta(a_t|s_t) \right] \tag{5}$$

where the minus-sign is necessitated by going from gradient ascent to loss minimization, and the γ is a discount factor applied to the rewards to use the discounted total reward instead of the total reward to reduce the bias that could be incurred by giving equal weight to all future actions.

Finally, to ensure that the trajectories sampled have variety in initial training and the NN still converges as the training is continued, a Gaussian noise parameter $g \sim \mathcal{G}(0, \sigma)$ is added to the predictions of the NN before the action selection occurs. To ensure convergence, σ is annealed by some fraction $\epsilon \in (0, 1]$ at the end of every network update to decrease exploration and increase exploitation to steer the network to convergence.

So in total, this work implements a fully-connected, 1-hidden layer (64-node) network with input and output dimensions are 4 and 2 respectively to match the aforementioned state space S and action space A of the environment. It uses ReLU activation function, however forces softmax on the output to ensure that the probabilities predicted by the net are positive and add up to 1. The loss function is custom-implemented as described above, and Adam is used as the optimization function. The NN uses Algorithm 2.1 to learn.

Algorithm 1 REINFORCE Algorithm

Input: a differentiable policy π_θ, learning rate η, update frequency f, exploration rate σ, discount fraction γ and annealing rate ϵ.
Initialize the policy NN with random weights θ.
Set **loss** to deque with size f.
for *trajectory* **in** *budget* **do**
  Sample trace $h = \{s_0, a_0, r_0, ..., s_{n+1}\}$ with π_θ.
  $R \leftarrow 0$
  $L \leftarrow 0$
  for $t \in n, ..., 1, 0$ **do**
    $R \leftarrow r_t + \gamma \cdot R$
    $L \stackrel{+}{=} R \cdot \log \pi_\theta(a_t|s_t)$
  end for
  Append L to **loss**.
  if *trajectory* $\%f = 0$ **then**
    $L \leftarrow$ mean(**loss**)
    Update the policy NN with L.
    $\sigma \leftarrow \epsilon \cdot \sigma$ ▷ Update the Gaussian noise in π_θ.
  end if
end for

2.2. Actor-Critic

The actor-critic uses the same approach solve the environment, but additionally uses a second critic NN to learn a value function $V_\phi(s)$. In this way, it combines the policy-based approach of REINFORCE with the value-based approaches explored earlier. This comes with the added benefit of using $V_\phi(s)$ to eliminate the aforementioned variance incurred in the Monte Carlo policy gradient algorithm. This is done through bootstrapping and baseline subtraction.

Bootstrapping, as investigated earlier in the tabular-methods, refers to the idea of defining a *target depth* n to calculate the discounted rewards, and using the learned value function $V_\phi(s)$ to bootstrap the rest while calculating the rewards-per-step acquired in the trajectories. Hence, the rewards are replaced with:

$$R_n(s_t, a_t) = \gamma^n \cdot V_\phi(s_{t+n}) + \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} \qquad (6)$$

where the first term is the bootstrapping term that uses the value-function learned by the critic NN. This helps eliminate the high amount of variance that the "tail" of the trajectory can have, as the longer the tail is, the more possibilities there are for how to create such a trajectory, and more variance is incurred. By cutting the tail at some point, this problem is somewhat eliminated while still using the information there is available from those state-action pairs.

The second use-case for the value-function $V_\phi(s)$ is baseline subtraction. This need stems from the operational principle of gradient ascent. Since discounted rewards are always positive regardless of the "desirability" or "optimality" of an action, the probabilities of all actions are pushed up. This dilutes the absolute difference between actions, as the difference between probabilities might not increase as fast as the values themselves, making it more difficult to distinguish the better actions from average ones. To this end, $V_\phi(s)$ is used as a baseline to define what an average action would return at a given point, and a comparative measure called *advantage* value $A(s_t, a_t)$ is defined:

$$A_n(s_t, a_t) = R_n(s_t, a_t) - V_\phi(s_t) \qquad (7)$$

Then by this definition, an action better than average would have a positive advantage, and its probability would get pushed up, while a subpar action would have a negative advantage and would get pushed down, overcoming the aforementioned problem.

Therefore, in the actor-critic approach, the actor or the policy part of the algorithms stays unchanged, and inherits the architecture and hyperparameters discussed in Section 2.1. On top of this, a critic network with almost identical structure is adopted as well, the only differences being an output dimension of 1 for the critic values, adopting an MSE loss, and the lack of softmax on the return of the output layer. Additionally, a hyperparameter n is introduced to govern the depth of the bootstrapping updates. All combined, the actor-critic uses Algorithm 2.3 to learn.

2.3. Training and Evaluation

Both REINFORCE and actor-critic NNs were trained using the same methodology. A total training budget of 10000 trajectories per random initialization of weights were chosen with some initial testing to be an adequate upper boundary. Similar to the approach in the previous study on DQNs, it was decided that a stopping criterion be established so that the training of the NN is stopped if this criterion is met and validation is started. In this context, validation is the process of freezing the weights of the NN/s and allowing the NN as trained to "play" the environment to investigate whether the trained weights are actually able to solve the environment reliably. Without a validation algorithm, it is actually not possible to say whether a given set of weights of the NN can actually solve the environment more than only a handful of times, as the weights are adjusted every update frequency f trajectories.

The previous DQN implementation used a 5-episode average of score 475 to consider an NN trained, and the same average score over 100 episodes was required to consider that the trained NN had solved the environment, which is the widely quoted v1 solution condition. Initial testing of both REINFORCE and actor-critic actually showed very

promising results, therefore this work implements a much higher bar for both of these criteria to train networks that perform much better than the v1 solution. For training to be considered complete and validation to be started, an average score of 485 is required over 100 episodes, already overtaking the v1 solution condition. NNs that clear this condition enter validation, where they are allowed to play the environment for an additional 1000 times. The NN is considered validated if an average score of 475 is achieved over these 1000 play sessions. Therefore, all scores quoted for validation in this work are upheld to a condition that we can call a v2 solution that is an order of magnitude more difficult to achieve in terms of generalization than the previous. Even though this is a very simple change to implement, this showcases clearly the robustness of the theory and implementation of this automated training and validation methodology as the performance that will be discussed in Section 2.4 and Section 3 in more detail.

Algorithm 2 Actor-Critic Algorithm

Input: a differentiable policy π_θ, learning rate η, update frequency f, exploration rate σ, discount fraction γ and annealing rate ϵ, bootstrap depth n.
Initialize the actor and critic NNs with random weights θ and ϕ.
Set **actor loss** and **critic loss** to deques with size f.
for *trajectory* **in** *budget* **do**
  Sample trace $h = \{s_0, a_0, r_0, ..., s_{n+1}\}$ with π_θ.
  $R, A \leftarrow 0$
  $L_{\text{actor}} \leftarrow 0$
  for $t \in n, ..., 1, 0$ **do**
    $R \leftarrow \gamma^n \cdot V_\phi(s_{t+n}) + \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k}$
    $A \leftarrow R - V_\phi(s_t)$
    $L_{\text{actor}} \stackrel{+}{=} A \cdot \log \pi_\theta(a_t|s_t)$
  end for
  Append L_{actor} to **actor loss**.
  $L_{\text{critic}} \leftarrow \text{MSE}(R(s_t, a_t), V_\phi(s_t))$
  Append L_{critic} to **critic loss**.
  if *trajectory* $\%f = 0$ **then**
    $L_{\text{actor}} \leftarrow$ mean(**actor loss**)
    Update the actor NN with L_{actor}.
    $L_{\text{critic}} \leftarrow$ mean(**critic loss**)
    Update the critic NN with L_{critic}.
    $\sigma \leftarrow \epsilon \cdot \sigma$ ▷ Update the Gaussian noise in π_θ.
  end if
end for

Another additional feature implemented in this training-validation methodology that was alluded to as a possible future-study in the previous DQN implementation was the addition of retraining of networks after failed validation. The idea behind this comes from imagining a set of weights that achieve for example a validation score of 470 instead of the required 475. Instead of considering these weights inadequate and throwing them out, if they were trained a bit further, it is feasible to imagine that they would clear validation. Therefore with this philosophy in mind, every set of random initializations are given a total of 4 attempts before they are considered unable to clear validation. This number has been also decided on with some initial testing and overall runtime in mind, as every validation adds another 1000 episodes to the total runtime. Additionally, to endure that the weights were different enough between repeated attempts of validation, a minimum threshold of training trajectories that has to cleared before attempting validation again was implemented as a function of the score achieved during the previous attempt, inversely linearly scaling between 10-100 episodes depending on how far the validation score was from the required 475. It was observed during experimentation and studies presented in this work that this has allowed NNs that would have been considered failures to actually train further and clear validation and therefore increase overall success-rate of obtaining solutions to the environment in a more streamlined fashion.

2.4. Hyperparameters and Optimization

Given the nature of NNs, hyperparameter optimization (HPO) has substantial impact on the performance of machine learning deep learning algorithms, more so any DRL NN given the amount of additional hyperparameters that are required to define the training and validation of the RL agent as discussed in the previous sections. This leads to an inherently large search space that is given the scope of this work and the constraints on runtime, is not possible to fully explore. For this problem 3 main HPO methods were considered namely: grid search, random search and Bayesian optimization. Random search is more efficient and explores better than grid search, and theoretically converges to the optimal solution. Bayesian optimization, cannot be parallelized efficiently and has its hyperparameters (such as the acquisition function and kernel of the Gaussian processor) which need to be selected separately. Random search is a good balance in simplicity and efficiency so it was picked for this problem. Therefore, to achieve statistical significance in any HPO implemented, a smaller search space of only 4 deemed-important parameters was performed for this work. The parameters and their search-space are given in Table 1.

From these hyperparameter ranges, a total of 10 different sets were generated, and each set was run for 5 repetitions. These results were used to determine the best possible hyperparameter sets for both REINFORCE and actor-critic. For REINFORCE, out of the 10 sets, 4 sets were able to solve the environment with the v2 criterion. Their parameters and validation scores are presented in Table 2 in the Appendix.

Hyperparameter	Range	Distribution
γ	[0.99; 1]	uniform
σ	[0.03; 0.15]	log
update frequency f	10, 15, 20	singular
depth n	50, 100, 150	singular

Table 1. Hyperparameters selected for optimization and their range distributions.

To pick the best set of hyperparameters from the above-quoted 4, the smoothed training curves and the distributions of their validation rewards were also investigated. With these in mind, since it had the highest average validation rewards in the 5 runs and it had the least fraction of non-500 rewards as presented in the top panel of Figure 2, the hyperparameter set $[0.6571, 0, 9975, 10]$ was selected to be used in the ablation study.

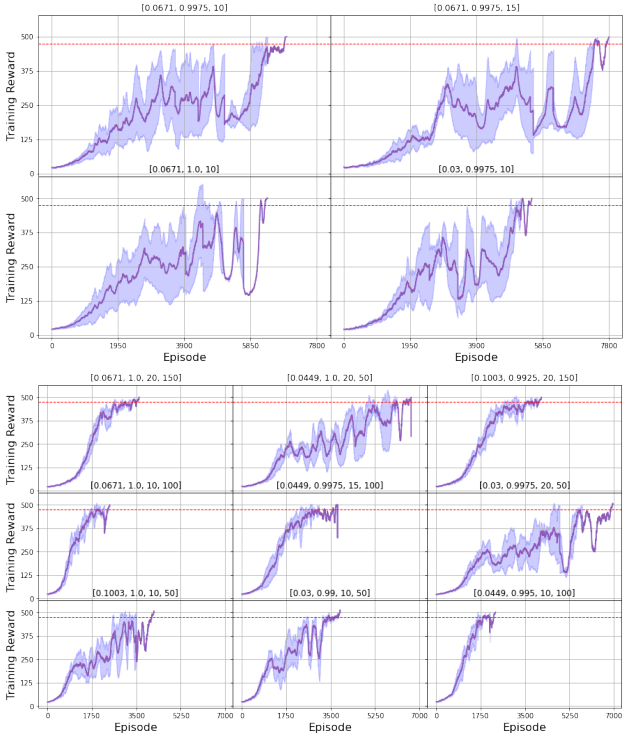

Figure 2. (Top) Smoothed rewards of REINFORCE agent with the various hyperparameter sets with the 1-σ standard deviation of 5 repetitions overplotted. Notice the high variance on the training rewards. (Bottom) Smoothed rewards of actor-critic agent with the various hyperparameter sets with 1-σ standard deviation of 5 repetitions overplotted. Notice the much lower variance on the training rewards, especially for set 9.

For actor-critic, the same range in Table 1 was used to generate 10 sets of hyperparameters, and these were also tested over 5 repetitions. Remarkaby, 9 out 10 of these sets were able to pass the v2 validation criterion 5 out of 5 times, so these were investigated further to pick the best set. Their parameters and validation rewards are presented in Table 3 in the Appendix.

In this case, the decision to select the best set of hyperparameters was slightly more difficult, given that a clear standout set did not exist. Even though set $[0.03, 0.99, 10, 50]$ had the highest average score, it can be seen in the bottom panel of Figure 2 that set $[0.0449, 0.995, 10, 100]$ had the most consistent and quickest convergence off all tested sets and was also the third-highest in average by a small margin. Given this consistency, this set was selected as the best to be used in the ablation study.

3. Ablation Study

Once the best parameter sets were selected as described in the previous section, more statistically significant random runs of these hyperparameter sets were performed with 20 repetitions each. With these, an ablation study was performed to better understand and quantify the performance impact of bootstrapping and baseline subtraction on solutions based on actor-critic algorithm. In this section, the results of these 20 repetitions will be discussed for the following 4 models: REINFORCE, actor-critic with only baseline subtraction, actor-critic with only bootstrapping and the full actor-critic network with both functionalities.

3.1. REINFORCE

The training progress and the smoothed average rewards over the validation attempts for the REINFORCE algorithm with best hyperparameters is shown in Figure 3. Out of the 20 random runs conducted, 18 has managed to complete training (aka. reach an average reward of 485 over 100 episodes) and subsequently cleared the v2 criterion that was set. 17 of these 18 runs managed to achieve this level of training within only ~ 6500 episodes of training, with the single outlier visible in the figure. Even though the runtimes and overall validation performances for these is remarkable, there still is a large variation of achieved rewards during a given episode in the training process, especially noticeable in the 3000-6000 episode mark, meaning that some runs are just able to train and validate much faster than the average and vice versa. This again points back to the aforementioned high-variance in the gradient update of the REINFORCE algorithm, even with the best hyperparameter set and more runs to attempt to decrease variation. But regardless, these 18 runs have an average validation reward of 486.16, easily clearing the v2 by more than 10 points over 1000 episodes.

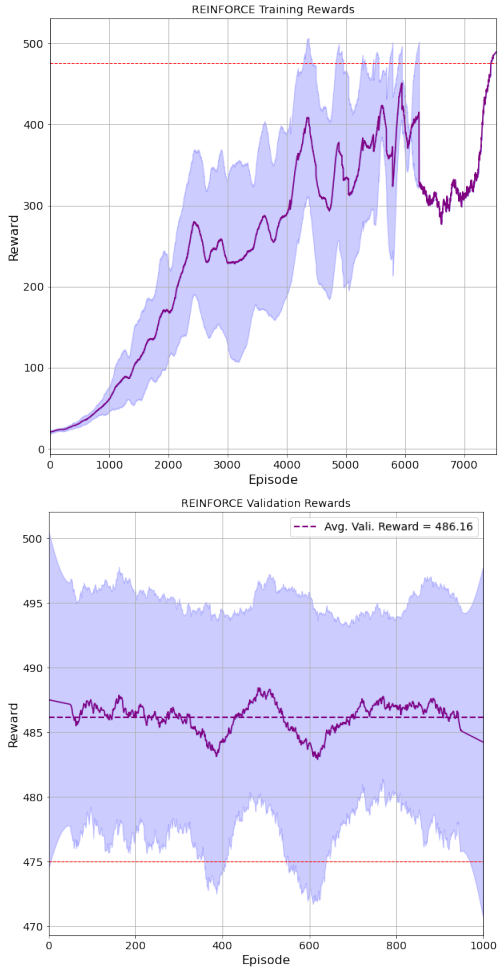

Figure 3. Smoothed training and validation rewards of REINFORCE agent with the best achieved hyperparameter set, with the 1-σ standard deviation of 18 repetitions overplotted. During training after 2500 episodes, variance in achieved rewards increases significantly. Worst performing run solved the environment in more than 7000 episodes. During validation phase, variance is much lower and reward averages at around 486.

3.2. Actor-Critic with Baseline Subtraction

In a similar vein, the training progress and the smoothed average rewards over the validation attempts of the actor-critic algorithm with only baseline subtraction is given in Figure 4. All actor-critic runs for this and following subsections managed to clear the v2 validation criterion 20 out of 20 times. Notice that the algorithm manages to learn and train much faster than the REINFORCE algorithm presented before, with all finishing training in less than 3000 training episodes. Also note the much lower variance in both rewards and times of convergence for this algorithm. On top of this, the learning is much smoother, with little to no "over-learning" or "forgetting" present, meaning the rewards steadily increase towards training criterion, unlike REINFORCE which is marked with many peaks and troughs. This shows that allowing for baseline subtraction already decreases the variance of the gradient update, and allows for only the correct actions' weights to be pushed up without the same amount of trial-error necessary. In general, this algorithm can be said to reach a similar overall reward after validation as REINFORCE, but much quicker and more consistently.

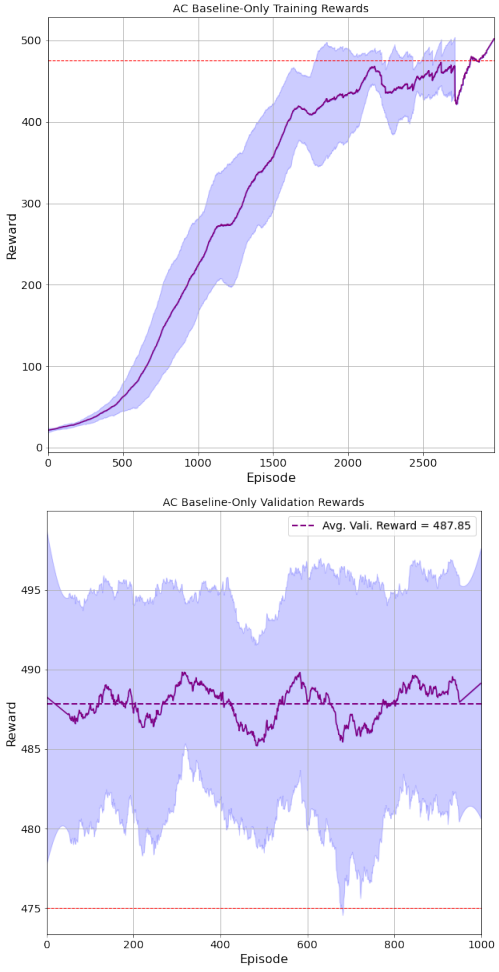

Figure 4. Smoothed training and validation rewards of Actor-Critic agent with only baseline subtraction, with the best achieved hyperparameter set, with the 1-σ standard deviation of 20 repetitions overplotted. Training and validation variance is significantly lower comparing to REINFORCE agent. Actor-Critic solves the environment much faster than REINFORCE agent, and average reward during validation phase is slightly better.

3.3. Actor-Critic with Bootstrapping

The training progress and the smoothed average rewards over the validation attempts of the actor-critic algorithm with only bootstrapping is given in Figure 5. Most of what

has been said about the actor-critic-specific improvements in Subsection 3.2 can also be said here, however 2 small things are notable. The overall convergence time of this version of actor-critic is even less than the previous by ~ 500 episodes, mostly due to the removing of the variance that is present in the tails of the long trajectories. Additionally, the validation rewards have increased to 489.18 averaged over 1000 episodes and 20 runs, which is showing promising performance.

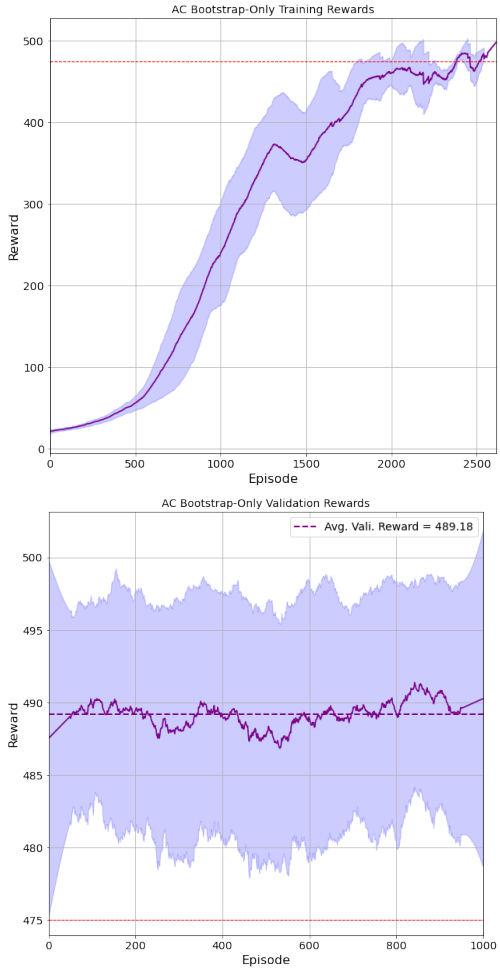

Figure 5. Smoothed training and validation rewards of Actor-Critic agent with only bootstrapping, with the best achieved hyperparameter set, with the 1-σ standard deviation of 20 repetitions overplotted. Performs faster and yields better results compared to the actor-critic algorithm with only baseline-subtraction.

3.4. Actor-Critic with Bootstrapping and Baseline subtraction

The training progress and the smoothed average rewards over the validation attempts of the actor-critic algorithm with both bootstrapping and baseline subtraction is given in Figure 6. This full version of the actor-critic performs very similarly to the previous actor-critic agent with only bootstrapping, but can be said to yield overall better results, with a validation average of 489.75, which is the combined effect of the baseline subtraction and bootstrapping. However, this comparison allows us to see, even though the results are close, most of the additional performance benefits in speed and rewards obtained, the bootstrap seems to be more influential than baseline subtraction. This is probably due to the variance of the tail being much more influential than the variance of the probabilities. Since a softmax is applied to the probabilities of the actor network output, even though the network is less likely to distinguish what is above or below average, the action selection still represents the correct behavior.

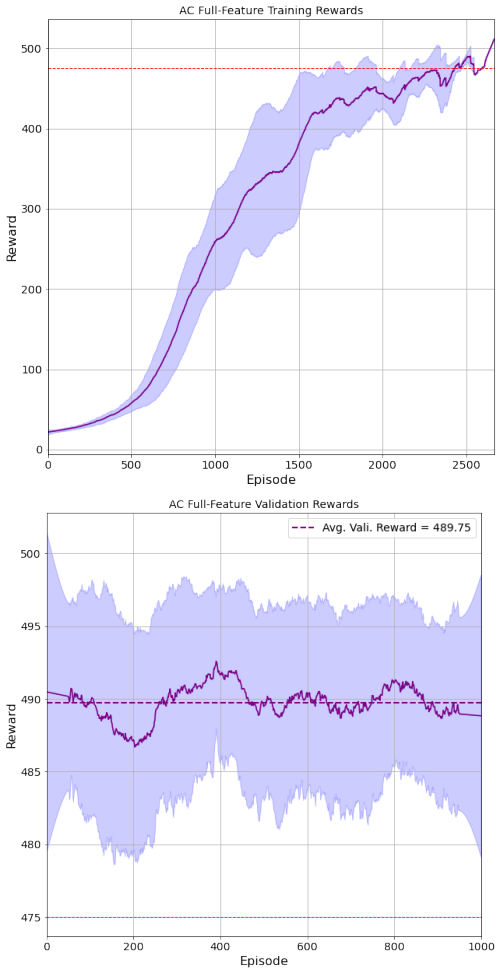

Figure 6. Smoothed training and validation rewards of Actor-Critic agent with both functionalities, with the best achieved hyperparameter set, with the 1-σ standard deviation of 20 repetitions overplotted. Converges at a similar speed with the actor-critic with only bootstrapping, however has a minuscule performance improvement in validation.

4. Conclusions

4.1. Results

As the result of this project an automated parallelizable framework for solving reinforcement learning problems with HPO was built along with a validation script, automated results and artifact (model) logging, analytical notebook to provide various statistics and plots.

This project refines and builds up on the previously presented automated and parallelizable framework for solving DRL problems, with a built in script for HPO, validation, and logging of results and model weights and models, together with notebooks to analyze and plot the results. Improvements were made to the HPO and validation architectures, and as mentioned in the future-work for the previous work, multiple stopping criteria were tested before coming up with the implemented v2 criterion. Additionally, the retraining of models with failed validations was also implemented with a mathematical function to assess the closeness to training to allow for a dynamic amount of additional training episodes to circumvent overtraining models, which improved the convergence of models in general.

Previously, this process of training and validating random network weights was not highly successful, and relied on the relatively low runtime to achieve solutions. However, with the refined architecture and better DRL algorithms, this success rate of getting solutions is $\geq 90\%$ for both REINFORCE and actor-critic agents for the v2 criterion. It is safe to say that all of the randomly initialized networks have cleared the v1 criterion since that is a subset of the training criterion we seek to start validation. So if comparisons are to be made to the widely-defined v1 criterion, this work presents an architecture to reach solutions under 15 minutes for REINFORCE and under 10 minutes for actor-critic from the repeated testing done during HPO and ablation study.

Looking at the results of the ablation study, it is clear to see that the best performing algorithm is the actor-critic agent with both bootstrapping and baseline subtraction. Considering that the same NN architecture is used for policy network in all 4 cases presented, REINFORCE can be taken as the baseline, and the bootstrapping and baseline subtraction can be seen as features implemented on top of the REINFORCE agent. In this sense, both bootstrapping and baseline subtraction aim to solve the various problems with high variance in the REINFORCE gradient update one with the tails of the trajectories, and the latter with the runoff weights and understanding of above and below average actions due to all-positive rewards, and hence improve the performance. Of the two, bootstrapping seems to be the dominant factor in this implementation.

4.2. Future Work

Given the satisfactory results of this work, nothing major is obviously present as improvements. However, work in two areas can make results even better. The first is a more comprehensive HPO. Since HPO is very costly in time and computational power if statistically significant results are sought, the HPO of this work was built to be generalizable to multiple search techniques and increasingly more complex search-spaces, but the HPO performed was kept simple with concerns regarding the runtime. Improving this HPO with either a larger search space, going from a coarser to a finer search, or using a different strategy such as Bayesian optimization. The second could be to push the training and validation criterion even higher to see, especially for the actor-critic agent, if a stricter criterion for training than 485 average over 100 episodes can be enforced to get even better networks in the allowed but under-utilised 10000 training episode budget. This could lead to networks that could, given more training time, clear validation conditions that are ~ 495 over 1000 validation episodes, which is well above-and-beyond the widely-documented solutions.

References

Moerland, T. *Lecture Notes: Continuous Markov Decision Process and Policy Search*, May 2022.

Plaat, A. *Course book: Deep Reinforcement Learning*, January 2022.

A. Appendix

#	σ	γ	f	$n.$	R
1	0.6571	0.9975	10	-	492.44
2	0.0671	0.9975	15	-	486.58
3	0.0671	1.0	10	-	487.73
4	0.03	0.9975	10	-	487.71

Table 2. The hyperparameter sets that validated the environment 5 times in 5 training runs for REINFORCE. Further experimentation and results are based on these sets of parameters. Table also includes average reward from these repetitions of validation phase.

#	σ	γ	f	$n.$	R
1	0.0671	1.0	20	150	483.25
2	0.0449	1.0	20	50	486.29
3	0.1003	0.9925	20	150	484.83
4	0.0671	1.0	10	100	484.25
5	0.0449	0.9975	15	100	486.42
6	0.03	0.9975	20	50	484.82
7	0.1003	1.0	10	50	485.65
8	0.03	0.99	10	50	488.88
9	0.0449	0.995	10	100	486.40

Table 3. The hyperparameter sets that validated the environment 5 times in 5 training runs for actor-critic. Further experimentation and results are based on these sets of parameters. Table also includes average reward from these repetitions of validation phase.

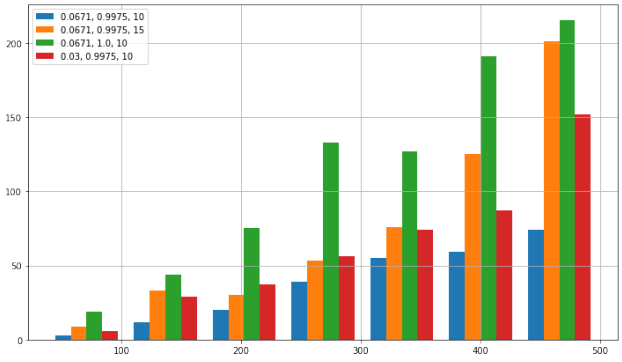

Figure 7. The histogram of non-500 rewards acquired by the different hyperparameter sets of the REINFORCE agent over 5 repetitions of 1000-episode long validation runs. Notice the significantly smaller amount of such sub-optimal rewards of the first set that is given in blue.

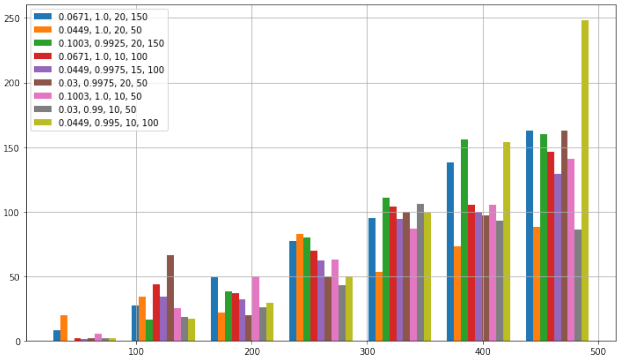

Figure 8. The histogram of non-500 rewards acquired by the different hyperparameter sets of the actor-critic agent over 5 repetitions of 1000-episode long validation runs. Even though the last set given in lime-green seems to have high fraction of sub-optimal rewards, most of these are in the 450-500 range, and therefore lead to less deviation in rewards achieved.