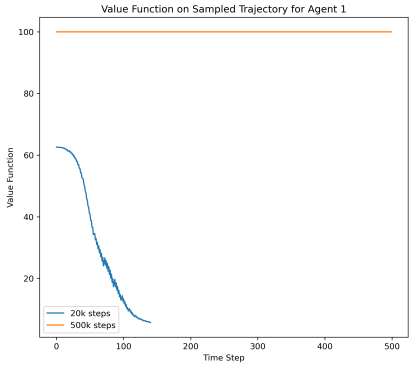
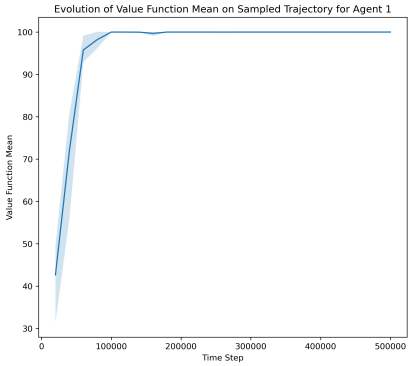
Mini-Project 2: From Discrete to Continuous Advantage Actor-Critic

1 The discrete case: the cart pole environment

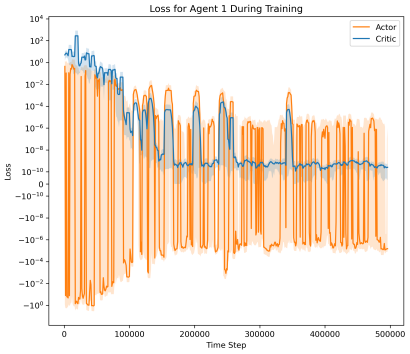
All results for the discrete environment are presented in Figures 1 and 2 below, and these will be referenced throughout this report.



(a) Value functions after 20k and 500k steps on sampled trajectory during evaluation.

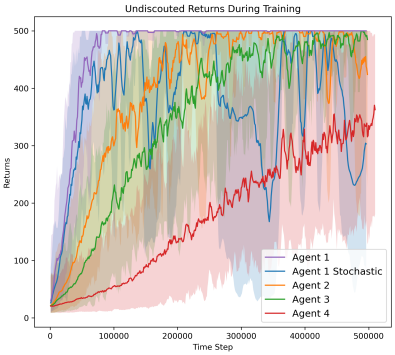


(b) Evolution of the mean of the value function on a sampled trajectory during evaluation.

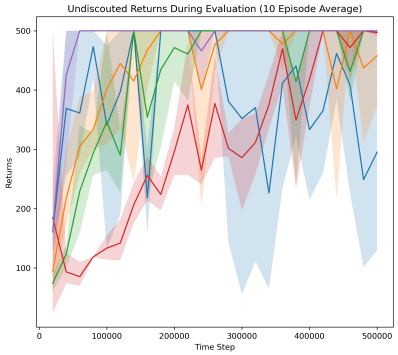


(c) Actor and critic losses (symmetric log scale), smoothed using a moving average with a window of size 5000 steps.

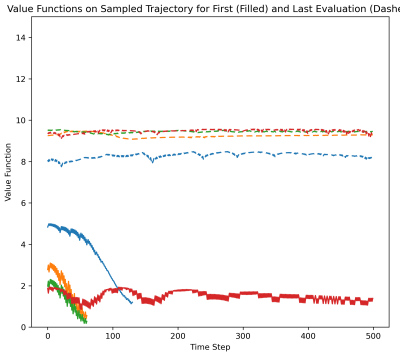
Figure 1: Partial results of agent 1.



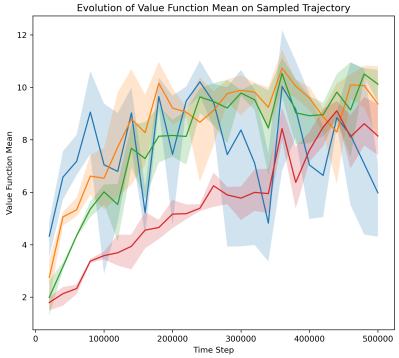
(a) Undiscounted returns during training, smoothed using a moving average with a window of size 5000 steps.



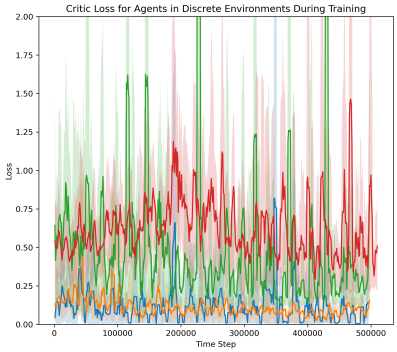
(b) Average of undiscounted returns during evaluation for 10 episodes.



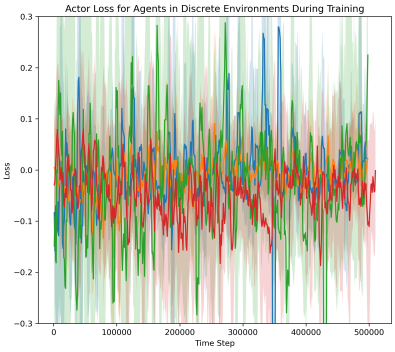
(c) Value functions sampled after 20k (filled) and 500k (dashed) steps during evaluation.



(d) Evolution of the mean of the value functions on sampled trajectories during evaluation.



(e) Critic loss, smoothed using a moving average with a window of size 5000 steps.



(f) Actor loss, smoothed using a moving average with a window of size 5000 steps.

Figure 2: Results of the agents in the discrete environments. Note: Agent 1 is present in a) and b).

1.1 Agent 1: Basic A2C version in cart pole

Q: *What values does your value function take after training has stabilized around an optimal policy (value loss less than 1e-4) using correct bootstrapping? What happens if you do not bootstrap correctly? Explain your findings with a theoretical argument.*
A: As can be observed in figure 1a, the value function converges to 100, when an optimal policy has been achieved. This is expected since the critic network approximates the value function defined as

$$V_\gamma^\infty(\pi, s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| S_0 = s\right] = \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}\left[r_t \middle| S_0 = s\right] \tag{1}$$

for an infinite horizon. Here we used the linearity of expectation. If an optimal policy is found, then hopefully the pole never falls down, and $r_t = 1 \ \ \forall t \geq 1$, according to the reward schema of the cart pole environment. This makes it independent of $S_0 = s$ and deterministic, implying $\mathbb{E}\left[r_t \middle| S_0 = s\right] = 1$. This yields the geometric series

$$V_\gamma^\infty(\pi, s) = 1 + \gamma + \gamma^2 + ... = \frac{1}{1-\gamma} = \frac{1}{1-0.99} = 100,$$

with $\gamma = 0.99$. Regarding bootstrapping incorrectly, there is likely several ways one could do that, however the most obvious mistake is incorrect handling of *truncation*. When stepping in the environment one receives the variables `terminated`, and `truncated`. In this context termination is when the pole falls down, and truncation is when 500 steps have been taken without falling. A common mistake is to zero out the critic of the next state (bootstrapping) if one reaches `terminated or truncated`. Indeed one does want to do this if `terminated` is reached since the stick has fallen down and one should not account for future (discounted) rewards. However, in the case of truncation, the episode ends due to an externally defined condition, and in essence the stick is still upright, and thus one should account for future (discounted) rewards. Thus one should zero out the critic if and only if `terminated == True`. In practice, zeroing out the critic for truncation as well will give a value function that falls down towards the end or a value function that converges to a lower value than 100, thus not resulting in the same result of $1/(1-\gamma)$ over the whole trajectory.

1.2 Adding complexity: stochastic rewards

Q: *Which value does your value function take after convergence to an optimal policy, using correct boot- strapping? Explain and interpret. Compare learning in your deterministic and stochastic environment: Does the value loss you observe after convergence differ from the deterministic to the stochastic environment? Explain your findings. How stable is the learning in each environment? To what can you attribute the presence/absence of difference in learning stability? Hint: think of the policy loss and how you are estimating it: How large is your number of samples and how does that affect your estimator? What about the role of the value function in the estimator?*
A: We do indeed observe in figure 2c that the value function converges to a significantly lower value, approximately 8. However we are a bit unlucky in the end since the agent forgets, and is not actually in a state of optimal policy. In theory it should converge to $\approx 0.1 \cdot 100 = 10$, if an optimal policy is found. It can be seen in figure 2a when the stochastic agent reaches a maximum reward of 500, the value function in figure 2d actually is converged around 10. Since we zero out 90% of the rewards we find

$$V_\gamma^\infty(\pi, s) = \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}\left[0.1 r_t \middle| S_0 = s\right] = \sum_{t=1}^{\infty} 0.1\gamma^{t-1} = 0.1\frac{1}{1-\gamma} = 10,$$

using the results from equation 1. However we do notice that this convergence is not as stable as with the deterministic environment. Although we do still find an optimal policy, the agent forgets and jumps back. Clearly, from figure 2a, the learning is not as stable with the deterministic environment, both in terms of loss of the networks, and undiscounted returns. For $K = n = 1$ we update the actor with $\nabla A_t \log \pi_\theta(a_t, s_t)$, where $A_t = R_t - V_\phi(s_t)$ and $R_t = r_t + \gamma V_\phi(s_{t+1})$. Having 90% of the rewards zeroed out will make the policy gradient estimator much noisier, since it depends on the advantage which in turn depends on the returns. This could potentially be mitigated by having multiple samples (multiple workers), making the estimator more robust. In this case $K = n = 1$ you effectively only get a single sample at each time step to estimate the policy gradient providing a noisy learning signal. The value function V_ϕ helps reduce variance in the policy gradient estimator by acting as a baseline that A_t is estimated relative to. Because rewards are so sparse, it will take longer for the value function to learn an accurate estimate, since on most steps it is only learning from its own predictions rather than real rewards. This could slow

down or destabilize learning. Furthermore, since the loss calculation depends on R_t, the introduced stochasticity prevents the value loss from fully converging to zero. The variability in R_t means that there will always be some level of fluctuation in the value estimates, which translates to a non-zero value loss. Even as the agent learns an optimal policy, the inherent randomness in the environment ensures that some degree of error remains in the value function approximation.

1.3 Agent 2: K-workers

Q: *Is the learning slower or faster than with $K = 1$? Contrast the speed in terms of number of environment interactions vs. wall-clock time. Is the learning more or less stable than with $K = 1$? To what can you attribute the difference?*
A: As seen in Figures 2a and 2b, the agent manages to reach an undiscounted return of 500, indicating it finds the optimal policy. Furthermore, learning with $K = 6$ is clearly more stable than with $K = 1$ (Agent 1 Stochastic). Although the value function does not exactly converge to 10 (figure 2c) as one might expect in theory, it performs significantly better in this regard compared to the stochastic Agent 1. This increased stability is expected with $K = 6$ because we are using multiple samples for our loss estimators, resulting in more robust estimations. However, because we update the network parameters $K = 6$ times less often, this agent might take longer to reach an optimal policy in terms of environment interactions compared to Agent 1. On the other hand the number of environment interactions per second increases significantly, for us, from approximately 300 interactions/s to 800 interactions/s. In practice, implementing multiple K-workers proves to be worthwhile.

1.4 Agent 3: n-step returns

Q: *Bootstrapping after 1 step as done so far allows the agent to learn faster but may be unstable. Can you explain why? Which value does your value function converge to? Is the learning slower or faster than with $n = 1$? Is the learning more or less stable than with $n = 1$? Explain your results.*
A: Bootstrapping after one step allows the agent to update its value estimates after every interaction with the environment. This frequent updating leads to faster incorporation of new information, speeding up the learning process. However, 1-step bootstrapping can be sensitive to noise and can lead to instability, especially in stochastic environments. When bootstrapping after one step, the updates rely heavily on the estimated value of the next state, which may be inaccurate or highly variable, especially in the early stages of training. Bootstrapping after n-steps allows the agent to consider a longer-term view of the rewards and state transitions, providing a more accurate estimate of the expected future returns by reducing variance. However, this means that the agent needs to wait for n steps before making an update, which can slow down the learning process compared to 1-step bootstrapping. This is the common dilemma of finding a balance between exploration and exploitation in RL.

As with agent 2 and seen in figure 2c and 2d, the Value function seems to converge close to 10. We also see that the learning is more stable, but takes longer, in terms of the number of environment interactions. This coincides with our reasoning above. Furthermore, as we saw before, the policy loss is guided by the advantage A_t which is in turn connected to R_t. If R_t is noisy or inaccurate, the computed advantage will also be inaccurate, leading to suboptimal or unstable policy updates. By taking n-steps and averaging the R_t we get also a good estimate for the Advantage and therefore the policy loss. We provide the algorithm for $K \times n$ batch learning in algorithm 1.

When the value of n becomes very large (e.g., $n > 500$), the algorithm starts to resemble the Monte Carlo (MC) method. In the MC method, the agent updates its value estimates based on the complete returns obtained from each episode, without relying on bootstrapping. With a large value of n, the agent effectively waits until the end of the episode to make updates, similar to the MC method. Using a very large value of n can lead to slower learning and may require more samples to converge, similar to the MC method.

1.5 Agent 4: $K \times n$ batch learning

Q: *Is the learning slower or faster than with $n = 1$ and $K = 1$? Contrast the speed in terms of number of environment interactions vs wall-clock time. Is the learning more or less stable than with $n = 6$ or $K = 6$? What are be the effects of combining both? Explain and interpret your results.*
A: As seen in figure 2a, Agent 4 does not quite find the optimal policy during training, on average, in 500k steps. The best seed does reach an optimal policy in the end, and in terms of evaluation where we use a greedy action policy, we do receive the maximum amount of rewards after 500k steps. The plots in figure 2a, 2b and 2d, indicates

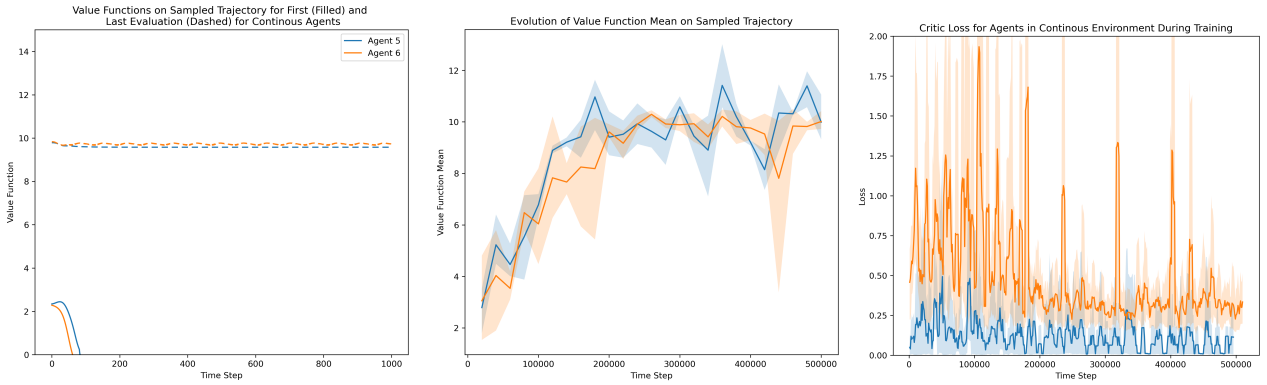
a slower, but a very stable learning, even more so than with Agent 2 and 3. Furthermore we get around 1200 environment interactions/s, a significant improvement to 800 when we had $K = 6$, $n = 1$. We should mention that we did not implement parallel environments, since this complicates bootstrapping (still doable), but this could also increase wall-clock time significantly. Combining both multiple workers and multiple steps synergistically improves stability. The diversity of experiences from multiple workers and the depth of learning from multiple steps create a robust learning process. However the downside is that one needs to take more environment steps in order to find the optimal policy. Below, in algorithm 1, we provide the algorithm similar to what we implemented, skipping a lot of important implementation details, such as when to bootstrap or handling that episodes may end at different time steps. For exact implementation we refer to our code.

Algorithm 1 n-step A2C with K workers

Initialize neural networks π_θ and V_ϕ.
Set counter $t \leftarrow 0$, observe s_0.
repeat
  for all workers $k = 1, \ldots, K$ **do**
    for $i = 0, \ldots, n-1$ **do**
      Take action $a_{t+i}^{(k)}$ and observe reward $r_{t+i}^{(k)}$ and next state $s_{t+i+1}^{(k)}$
    end for
    Define $R_{t+j:t+n}^{(k)} = \sum_{i=j}^{n-1} \gamma^{i-j} r_{t+i}^{(k)} + \gamma^{n-j} V_\phi \left(s_{t+n}^{(k)} \right)$
    for $i = 0, \ldots, n-1$ **do**
      Compute $(n-i)$-step advantage $A_{t+i}^{(k)} = R_{t+i:t+n}^{(k)} - V_\phi \left(s_{t+i}^{(k)} \right)$
    end for
  end for
  Update θ with gradient of $\frac{1}{Kn} \sum_{k=1}^{K} \sum_{i=0}^{n-1} A_{t+i}^{(k)} \log \pi_\theta \left(a_{t+i}^{(k)} \mid s_{t+i}^{(k)} \right)$
  Update ϕ with gradient of $\frac{1}{Kn} \sum_{k=1}^{K} \sum_{i=0}^{n-1} \left(R_{t+i:t+n}^{(k)} - V_\phi \left(s_{t+i}^{(k)} \right) \right)^2$
  Increment $t \leftarrow t + Kn$
until some termination criterion is met.
return π_θ and V_ϕ

2 The continuous case: the InvertedPendulum environment

All results for the discrete environment are presented in figure 3 on the next page, and these will be referenced throughout this part of the report.

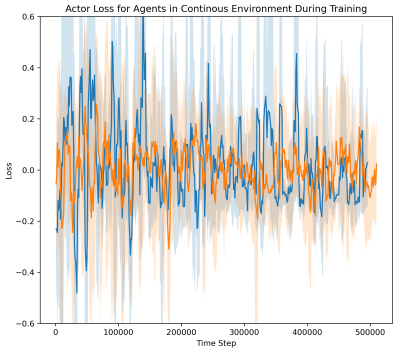


(a) Value functions sampled trajectory after 20k (filled) and 500k (dashed) steps during evaluation.

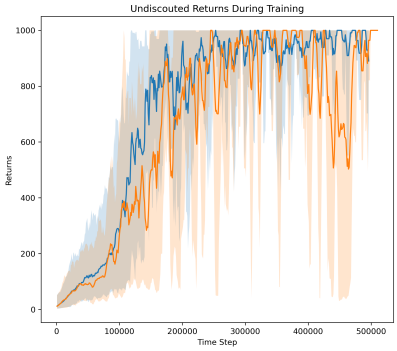
(b) Evolution of the mean of the value functions on sampled trajectories during evaluation.

(c) Critic loss, smoothed using a moving average with a window of size 5000 steps.

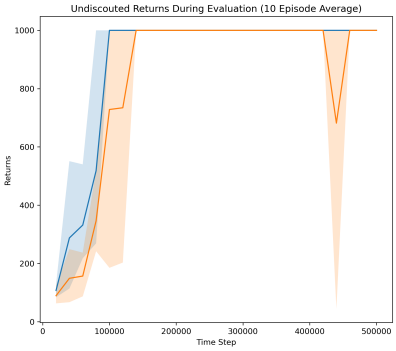
Figure 3: Results of agents for continuous environment.



(d) Actor loss, smoothed using a moving average with a window of size 5000 steps.



(e) Undiscounted returns during training, smoothed using a moving average with a window of size 5000 steps.



(f) Average of undiscounted returns during evaluation for 10 episodes.

Figure 3: (Continued) Results of agents for continuous environment.

2.1 Agent 5: Evaluating the cart pole agent in the new environment

Q: *Is this continuous version of the environment harder than the discrete one? Can you precisely describe why? Hint: This is a characteristic of RL that doesn't exist in supervised learning, for example).*
A: From the sub-figures in Figure 3, we conclude that this agent does indeed find an optimal policy, and that the value function converges to approximately 10. Although the training here went quite well, in general, a continuous environment is harder to tackle than a discrete environment. In the discrete case, we only have binary actions (move left or right), whereas in the continuous case, we must decide not only the direction to move but also the precise force, which can take any value within the infinite range $[-3, 3]$. In the discrete action space, the agent can systematically explore each action in different states, making the exploration process more straightforward. In contrast, the continuous action space presents an infinite number of possibilities, making exhaustive exploration impractical. To address this, we fit a Gaussian distribution and sample from it to determine the action. However, there is no guarantee that this distribution is the optimal choice. Essentially, a continuous action space makes the exploration aspect of the exploration vs. exploitation dilemma in reinforcement learning significantly harder. Despite these challenges, our agent performs quite well in the continuous environment. This success can be attributed to the fact that actor-critic methods naturally extend to continuous action spaces. This is typically the case with continuous environments, where specialized methods such as A2C (Advantage Actor-Critic) or DDPG (Deep Deterministic Policy Gradient) are required to effectively handle the increased complexity.

2.2 Agent 6: A2C agent

Q: *Why should one expect that when increasing K and n, one can also increase the actor's learning rate to speed up the performance of the agent while keeping it stable? What happens with performance and stability if you increase the actor's learning rate while keeping $K = 1$ and $n = 1$?*
A: Our policy does indeed find an optimal policy quickly in terms of wall-clock time, though not in 10 seconds, but rather a few minutes. We get a speed of about 1400 interactions/s. This is likely due to not implementing parallel workers. Our implementation could be extended using vectorized environments provided by Gymnasium, but as mentioned earlier, this requires special care for correct bootstrapping.

As seen in Figures 3e and 3f, the number of interactions needed to reach the optimal policy is comparable to Agent 5, thanks to the increased learning rate, even though we update the parameters $K \times n$ times less often. Of course, we have some added instability due to the 13-fold increased learning rate. Combined with a significantly higher number of interactions per second, this makes our method very efficient. Using $K \times n$ batch learning, as opposed to stochastic/online learning, enhances the robustness of our estimators, allowing for more confident gradient calculations and a higher learning rate, which speeds up convergence. The larger batch of experiences gathered from K environments helps average out noise and variance in the updates. With n step returns, the agent considers longer trajectories, capturing more information about the environment's dynamics. When we increased the actor learning rate to $3 \cdot 10^{-4}$ with $K = n = 1$, the agent reached an undiscounted return of 1000 after only 20k steps. However, this approach was as expected very unstable, with returns often dropping below 100, coinciding with our analysis.