COMP 3004 Designing Intelligent Agents

Name : Ryan Dave
Student ID : 14306556

***Abstract*—In this paper, a robust solution is proposed for OpenAI Gym's implementation of the 'Mountain Car Problem'. In the field of reinforcement learning, the 'Mountain Car Problem' is a commonly known state-action problem in which a weak-powered car is situated between two hills with the aim of reaching the right-most peak by leveraging the opposing hill. The proposed approach uses the deep deterministic policy gradient algorithm to learn in a continuous action setting. Additional environments were implemented as an extension to the problem in order to evaluate training and generalization performance when faced with alternative initial positions.**

I. Introduction

Reinforcement Learning problems are composed of an intelligent agent that is able to perceive an environment with the aim of learning a specified goal based on feedback from interactions. In this paper we will be focusing on the 'Mountain Car Problem', where the environment consists of a car placed between a valley. The agent makes decisions on the acceleration value of the car with the goal of reaching a flag at the top of the right-most hill.

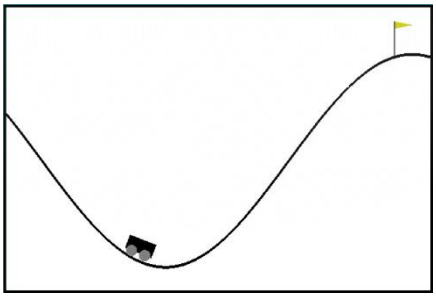

Fig. 1. Snapshot of rendered environment

The difficulty arises due to the power of the engine being unable to clear the hill by solely accelerating forward. The agent must learn that they must utilise potential energy gained by traversing the hill positioned opposite. By leveraging both hills the car can "swing" and reach the goal position. The constraints of the environment used are as follows, where x and y are the observations seen by the agent, and a is the action that can be taken:

$$(x, y) \in Position \times Velocity \begin{cases} Position = [-1.2, 0.6] \\ Velocity = [-0.07, 0.07] \end{cases}$$

$$a = [-1,\ 1]$$

In the domain of Reinforcement Learning, there exists a number of algorithms that have been developed for the agent to efficiently learn the optimal policy to choose actions.

As seen by the constraints, this problem covers a continuous action space and therefore an algorithm to match the problem must be carefully considered. This paper explores the use of the deep deterministic policy gradient algorithm and aims to answer two core questions:

1) Can the agent learn to effectively solve the 'Mountain Car Problem (Continuous)' using DDPG?

2) Does the learning transfer over to a different initial environment state? In other words, will the agent solve the problem and perform effectively when placed at varying initial points in the valley?

II. Literature Review

A. Review of reinforcement algorithms

A policy π gives an action (a), for each state (s) for each time (t), that can be defined as a Markov Decision Process (MDP) [1]. In essence, a policy dictates the action of an agent as a function of its state at time t and represents the 'strategy' that the agent uses.

All reinforcement learning algorithms seek to find the optimal policy for a given problem, meaning the policy that will yield a greater return to the agent than all other policies. In general, the optimal policy will hold an associated optimal state-value function and optimal action-value function.

The optimal policy must also satisfy the Bellman optimality equation, that states for any given state-action pair, (s, a) the expected return is equal to R_{t+1} plus the maximum expected discounted return for any (s', a').

$$\pi_*(s, a) = E[R_{t+1} + \gamma \max_{a'} \pi_*(s', a')]$$

A well-known iterative approach to find the optimal policy known as Q-Learning, Q-Learning uses Temporal Differences (TD) to learn through episodes with no prior knowledge of the environment [2].

This method is sufficient for simplistic environments, however performance of Q-Learning declines as the observation space increases. A significant increase in observation space dramatically increases the time to take to traverse all possible states and iteratively update all Q values.

This has been built upon by improved methods such as Deep Q Networks (DQN) [3] to handle higher dimensional observation spaces for more complex environments by combining deep neural networks with Q-Learning.

Unfortunately, there exists an additional barrier as these methods are unable to be applied a continuous action space. A work around of discretising the action space, suffers greatly from the curse of dimensionality.

An alternative method that builds upon both approaches was proposed, Deep Deterministic Policy Gradient (DDPG) [4]. DDPG combines the actor-critic approach [5] with aspects of DQN [6], namely replay memory and target networks:

i. Actor Critic

In the actor critic model, the actor takes a given state, and returns an action. The critic takes a given state action pair and returns a value (in this case represented by Q).

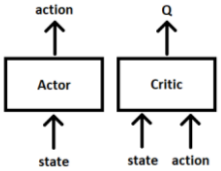

Fig. 2. Representation of an Actor Critic approach

ii. DQN – Replay Memory

The concept of replay memory consists of a fixed size buffer to keep track of previous experiences. This buffer can be randomly sampled to batch train the weights of the respective deep neural network.

iii. DQN – Target Networks

Target networks are used as a secondary network (separate to the online network) to combat unstable learning that occurs due to rapidly changes values for similar actions (as weights are updated every time step). In DQN, the weights are transferred to the target network every n iterations via a hard update. It should be noted that in the paper, this was improved in DDPG by performing a soft copy using a hyper-parameter τ.

B. Review of environment

The mountain car problem [7] is a well-known problem in reinforcement learning and often used as a benchmark for a given algorithm. Therefore, there are instances of published papers that solve this problem, even in a continuous setting [8][9]. Results vary in the two papers found, the first shows that the agent works well to reach the goal between 50-100 epochs with a steady gain (average q value). The second paper shows a worsened result with a less steady gain, however it is not clear to say whether the same hyper-parameters were used in both experiments. In any case, DDPG has been shown to be able to solve the problem when centrally based in the valley.

C. Summary

To answer the core question relating to whether DDPG can effectively learn the problem, we will extend past existing experiments in the reviewed papers by altering the initial starting position. At this time, to my knowledge, both the alteration and experimentation into transfer learning for this problem is unique.

III. Implementation

1) Agent setup

i. agent.py

An agent class was implemented to handle the initialisation, training and updates of the networks required as well as a method to run a specified number of episodes and return a list of rewards.

First, we must initialize our base Actor Critic networks. The actor network remains simplistic, and state passes through two fully connected layers, of dimension 64. The inputs to the critic network differ slightly. State and action are first passed through a layer (dimension of 32) separately before being concatenated and passed through two fully connected layers (dimension of 64).

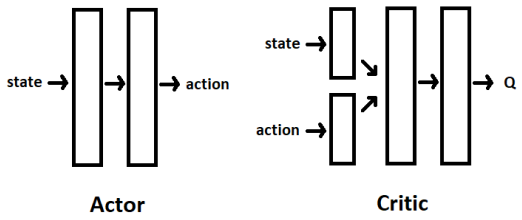

Fig. 3. Representation of network shapes

As mentioned previously, we require a target network and therefore we must initiate an online Actor and Critic, and a target Actor and Critic (totaling to four networks).

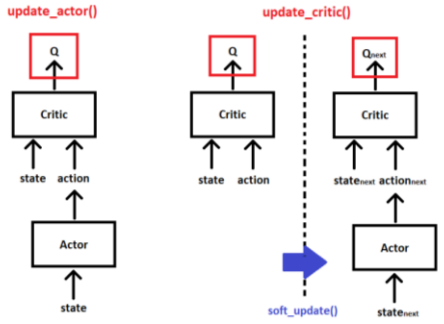

Fig. 4. Representation of network optimisation

Two methods handle the updating of the online networks. The method update_actor(), uses the critic network and real world values to generate a Q value. The weights are optimized to minimise(-Q) (equivalent to maximizing Q) using tf.math.reduce_mean.

Again, the critic remains less trivial, as the method update_critic() utilises the target networks. Here, we gather a Q value from real world values and a Q_{next} value (Q value that relates to the optimal action, for the next observation) and optimise by minimizing the difference between Q and a discounted Q_{next}.

$$| Q - (\text{reward} + \text{discount} * Q_{\text{next}}) |_{min}$$

Here we use both the online and target networks, as explained previously, training would be unstable and is unlikely to converge if confined to a single network. Weights are updated using a soft update approach, shown below, where theta relates to the model variables and tau = 0.01.

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \; with \; \tau \ll 1$$

The pseudocode for an episode run is shown below. The policy method takes a state that is passed through the actor network, introduces an element of noise and returns the action to be taken next. The noise operation will be discussed later.

Algorithm 1: Run episode

```
while True:
    action ← policy(state, noise)
    env_data ← env(action)
    buffer ← env_data
    if training:
        sample buffer
        update networks
    if episode has timed out, or completed
        break
save episode rewards
```

Fig. 5. Psuedo-code for Agent learning method

ii. buffer.py

The buffer class holds the implementation for the replay memory. A buffer object can store, and randomly sample observations. The size is fixed, meaning the buffer will begin to overwrite old observations once full.

iii. noise.py

The noise class implements Ornstein-Uhlenbeck noise (identical to original paper). Noise is required in DDPG due to its deterministic nature, meaning that for a given state, it will return the same action. It is necessary for the exploration of the environment, and the parameters set are balanced well for the exploration-exploitation tradeoff.

B. *Environment setup*

The task environment was chosen from OpenAI gym as the framework holds a working version of the problem that is easily extendable.

Three additional Gym environments were created from the original:'Continuous_MountainCarEnv_v1','Continuous_MountainCarEnv_v2' and 'Continuous_MountainCarEnv_v3'. These were created to alter the initial position of the car, which is core to answering the initial project questions. Using multiple environments allows a parameter to be passed to carry out the individual experiments. In addition, the render method was altered to give the user a visual indication of the initial point. During the discussions, the environments will be shortened to 'Env_v1' (representing the middle position), 'Env_v2' (representing the right-most position) and 'Env_v3' (representing the left-most position).

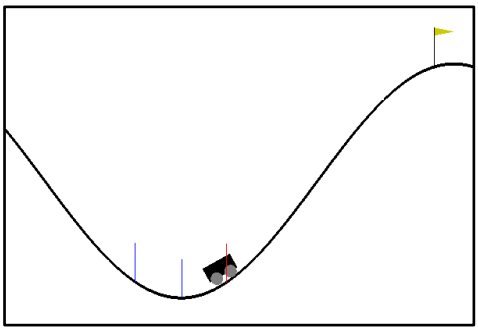

Fig. 6. Snapshot of 'Continuous_MountainCarEnv_v2

C. *Experiment setup*

The experiments were designed as follows. An experiment consists of a number of trials, with each trial performing a number of episodes. A control experiment is performed for every environment, returning the baseline reward values without training the agent. A training experiment is performed for every environment, returning the reward progression as the agent is trained on a given environment. Finally, a test experiment is performed on each environment for every model that was trained. This means that for a given environment x, three agents are tested, one that was trained on environment x and two other agents that were trained on the remaining environments.

To perform these experiments, an automated suite was created to run any given experiment, or all, by providing the runner with the number of trials and the environments to be included in the experiments. This allows a simplified way to perform the experiments and an easily extendable solution if more environments were to be added in the future. The framework allows the control of the parameters of the agents, including the saving/loading operations. The absolute metrics are automatically saved in .txt files structured in a folder for later statistical analysis, while plots are also automated for ease.

IV. DISCUSSION

In order to successfully interpret the results, it should be noted that the reward value is based on two factors. The first factor is a large reward (100) given to the agent if the primary goal is achieved. The second, is a negative reward based on the acceleration input of the agent. By using two metrics we can assess the agent's performance based on if they successfully reach the top of the hill and how 'efficient' the solution is (taking advantage of gravity to use less energy).

1) *Can the agent learn to effectively solve the 'Mountain Car Problem (Continuous)' using DDPG?*

i. Visual Analysis

To address this question using visual analysis, we will first focus on the training summary plot for the original environment (figure 1 in Appendix A). The training summary plot shows the average reward for the first 50 episodes, for 5 distinct trials.

3 of the 5 trials are shown to have a steady rate of learning as the average reward increases as the number of episodes increases. In these instances, the agent was able to reach the flag.

Interestingly, the remaining 2 trials are shown to remain steadily around the 0 mark, with one trial even seeing a slight decrease in rewards over the episodes. In these instances, the agent was not able to reach the flag and therefore learned to conserve its energy and remain at the base.

As an extension we can look at the training performance of all initial starting positions (figure 4 in Appendix A). Env_v2 and Env_v3 show that the agent can successfully learn when placed either side of the base of the valley. In fact, the agent is seen to successfully learn the problem 100% of the time in both environments, reaching the flag with a minimal deduction. Both learning rates are steady and with a similar gradient, however Env_v2 is shown to have a higher variance (than Env_v3) and some trials show the agent to be slower at learning (reach goal after a larger number of episodes).

ii. Descriptive Statistics

To address this question using descriptive statistics, we will focus on the comparisons of average rewards between the training environment and control values (no training performed).

As seen in figure 3 of Appendix A, 3 out of 5 trained agents in Env_v1 and all trained agents in Env_v2 and Env_v3 produced an average reward close to 100, showing that the problem was solved. Again, 2 of the 5 agents in Env_v1 is shown to have a significantly lower reward due to not reaching the goal position. The negative reward shown in is due to the noise that is added to the model output value.

Inspecting the reward values closer, Env_v3 is shown to have reached the goal with a lower cost. It makes sense to put this difference down to the starting position giving initial momentum.

iii. Inferential Statistics

To address this question using inferential statistics, we will focus on the results of t-tests performed on the data shown in figure 3 of Appendix A. P-values were calculated as 0.0406, <0.0001 and <0.0001 for the environments Env_v1, Env_v2, Env_v3 respectively. Using these p-values we can say with 95% confidence for all environments that there is a statistical increase as a result of training. We can say with greater confidence (>99.9999%) that Env_v2 and Env_v3 agents improve the average reward.

iv. Conclusion

By using a combination of all three methods we can safely conclude that DDPG does indeed effectively learn to solve the problem, not only at the initial starting position (as seen in the literature) but also when placed in a position favouring forward movement and backwards movement. It should be noted that Env_v1 did not solve the problem 100% of trials and an increased sample size would provide a more accurate representation of the true success rate.

2) Does the learning transfer over to a different initial environment state? In other words, will the agent solve the problem and perform effectively when placed at varying initial points in the valley?

i. Visual Analysis

To address this question using visual analysis, we will focus on the testing summary plots (Appendix B figures 1 and 2). To maintain a structure during the discussion, we will inspect each agent type (i.e. trained on either Env_v1, Env_v2 or Env_v3) separately. The data for these plots were produced by running the trained model for a number of episodes with no further training.

If we first inspect the line graphs, we can see in the second column all of the test plots for the agents trained on Env_v1. As discussed in the previous section, when trained in its own environment 3 of the 5 models reached the goal and the remaining failed to reach the flag, so whilst discussing transfer learning, this must be taken into account. When tested on Env_v2 and Env_v3, those that were successful on the training environment were all able to solve the problem successfully. Surprisingly, in some instances, agents that were unsuccessful in reaching the goal originally, managed to reach the goal on some episodes.

These results are backed up by the box plot (red elements) for the successful agents. Out of the unsuccessful agents, most of the occasional successes are seen as outliers, except for trial 4 on Env_v3. In this instance, the number of successful episodes are seen as significant and included in the plot.

In the same way, if we inspect the third column we can see all of the test plots for the agents trained on Env_v2. In this case, the agent reaches the goal in all instances, in all environments. These results are again backed up by the box plots (purple elements).

Finally, if we inspect the fourth column of line graphs, we can see all plots for agents trained on Env_v3. When tested on its own training environment all episodes result in a high reward. When tested on the other environments however, two trials represented by the green and yellow lines show a sporadic series of episodes, occasionally solving the problem. This is also seen by inspecting the blue elements in the box plots, where trials 2 and 3 (in Env_v1 and Env_v2) have low values for their respective quartiles.

ii. Descriptive Statistics

To address this question using descriptive statistics, we will inspect the tables shown in figure 3 of Appendix B.

For the agent trained on Env_v1 (represented in table as v1), the mean and standard deviation is consistent when tested on Env_v2, and even results in a slight increase in mean (50.0336 to 64.3825) and decrease in standard deviation (59.2531 to 44.5563) when tested on Env_v3.

For the agent trained on Env_v2, we can observe that this agent results in the best performance of any agent across all environments (greatest mean, lowest standard deviation and lowest standard error), solving the problem in every instance.

For the agent trained on Env_v3, we can see that it performs well on its original environment, however fails to perform as well across other environments. This is shown by the numerous trials that are unable to reach the goal and low mean values.

iii. Inferential Statistics

To address this question using inferential statistics, we will focus on the results of the one-way ANOVA test performed on the data shown in figure 3 of Appendix B. P-values were calculated as 0.2729, 0.1872 and 0.1275 for the test environments Env_v1, Env_v2, Env_v3 respectively. This means that with a threshold of 0.05, we fail to reject the null hypothesis and are unable to say whether there is a statistical significance between the means of the three groups without additional evidence.

iv. Conclusion

Despite the non-conclusive results of the statistical test, the visual and descriptive analysis provides us with a lot of information to answer if ‘the agent will solve the problem and perform effectively when placed at varying initial points’. When placed at the base of the valley or placed to the right of this position, learning seems to transfer over to other positions in the environment with a performance at least as good as the original trained model. When placed to the left of this position, agents are able to sometimes achieve a similar performance, however some cases show that this is not

always the case. This may be due to weights in the trained models favouring a forward movement first, whereas the position favours movement to the left due to gravity. Unfortunately, we cannot say with statistical confidence, however evidence strongly backs this conclusion. In this paper, more trials were not possible under time constraints (as the length of training time is substantial). This would be the area of improvement if research was to continue.

V. Conclusion

In conclusion, the work performed in this research successfully answers the two core questions that were originally outlined. We can see that the agent can effectively learn to solve the problem under differing initial placements. Learning can be seen to transfer to other positions in the environment, however the agent has been seen to underperform in some cases. An experiment framework was implemented to automate the running of the agent and allows extendibility for future work. Future work could explore similar OpenAI Gym environments (as the framework remains interchangeable for such environments) and aim to explore the effectiveness of DDPG for RL problems and transfer learning further. No major limitations were identified, however results may rely heavily on hyper-parameters and the choice of noise object. The time constraint for the project could be seen as a limitation, as episodic runtime is significant. Given more time, more trials and more environments could be evaluated to consolidate these findings.

References

[1] Ahiska, S.S., Appaji, S.R., King, R.E. and Warsing, D.P. (2013). A Markov decision process-based policy characterization approach for a stochastic inventory control problem with unreliable sourcing. *International Journal of Production Economics*, 144(2), pp.485–496.

[2] W. -K. Yun and S. -J. Yoo. (2021) "Q-Learning-Based Data-Aggregation-Aware Energy-Efficient Routing Protocol for Wireless Sensor Networks," in *IEEE Access*, vol. 9, pp. 10737-10750

[3] Bradtke S.J., Duff. M.O. (1994). "Reinforcement Learning Methods for Continuous-Time Markov Decision Problems,"

[4] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N.M., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning.

[5] I. Grondman, L. Busoniu, G. A. D. Lopes and R. Babuska, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1291-1307, Nov. 2012.

[6] Roderick, M., MacGlashan, J., & Tellex, S. (2017). "Implementing the Deep Q-Network."

[7] Gatti, C. (2015). "The Mountain Car Problem. In: Design of Experiments for Reinforcement Learning." Springer Theses. Springer, Cham.

[8] J. Wu, R. Wang, R. Li, H. Zhang and X. Hu, "Multi-critic DDPG Method and Double Experience Replay," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, pp. 165-171.

[9] Hollenstein, J.J., Renaudo, E., Saveriano, M., & Piater, J.H. (2020). "Improving the Exploration of Deep Reinforcement Learning in Continuous Domains using Planning for Policy Search."

Appendix A

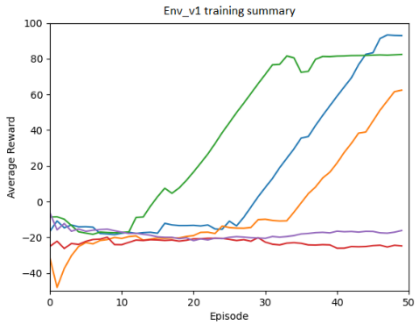

Fig.1. Training summary plot for Env_v1

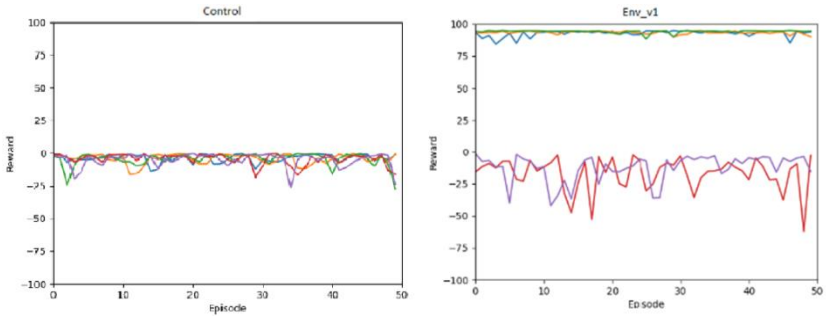

Fig.2. Testing summary plots for Env_v1 Agent

Env-v1		
Trial	Control	v1
1	-13.3635	92.62698
2	-15.9912	93.22382
3	-15.5153	94.01178
4	-12.22	-17.1865
5	-16.125	-12.5083

Env-v2		
Trial	Control	v2
1	-14.5755	94.65293
2	-15.0882	94.6002
3	-15.6519	94.53074
4	-14.6119	95.21931
5	-13.3061	94.15264

Env-v3		
Trial	Control	v3
1	-8.91994	95.61583
2	-10.1013	96.26328
3	3.545134	95.95878
4	-8.84714	96.19583
5	-2.90653	96.23012

Fig.3. Testing summary tables for all Agent types

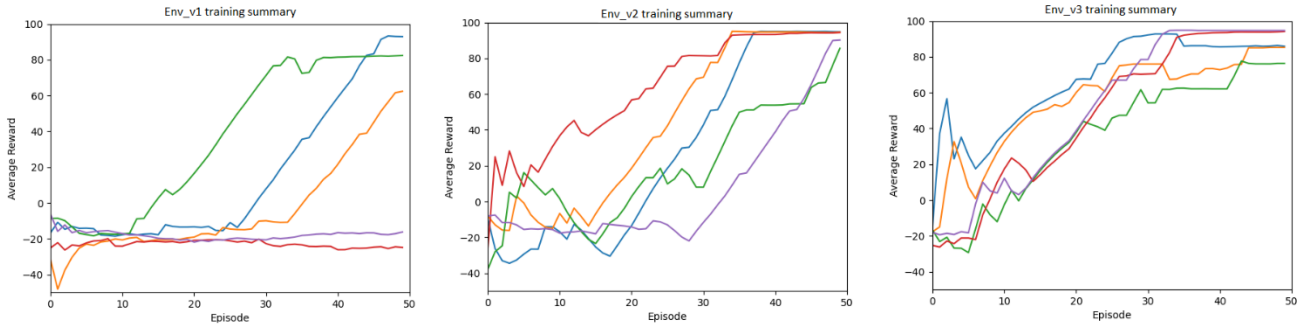

Fig.4. Training summary plots for all Agent types

APPENDIX B

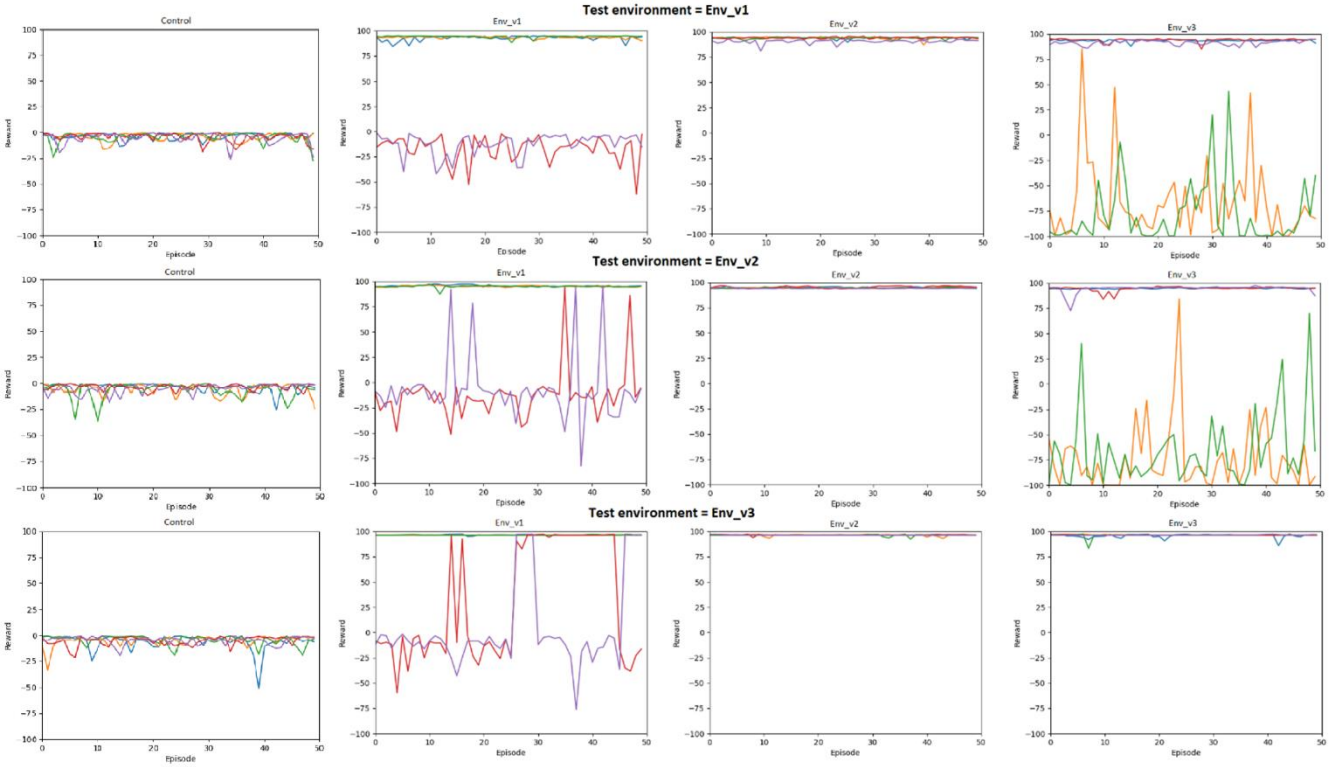

Fig.1. Testing summary plots for all Agent types in all environments

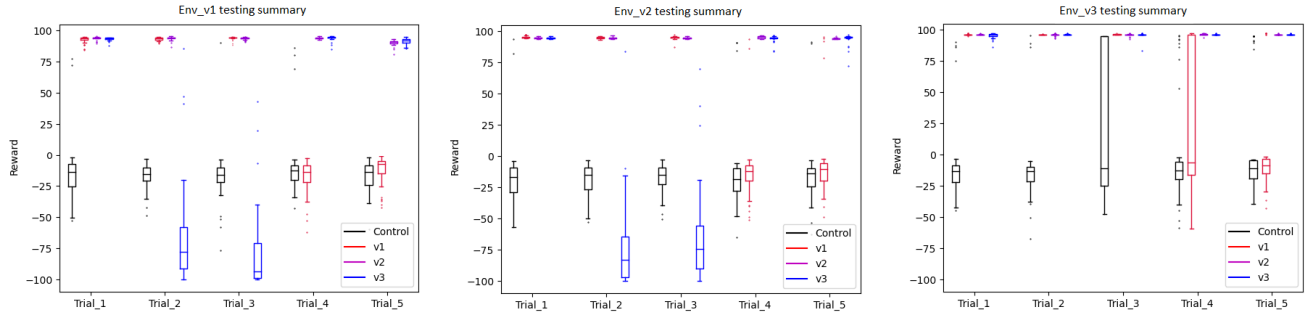

Fig.2. Testing summary plots for all environments

Env-v1

Trial	Control	v1	v2	v3
1	-13.3635	92.62698	93.9207	93.41459
2	-15.9912	93.22382	93.83917	-66.4504
3	-15.5153	94.01178	93.66574	-78.1438
4	-12.22	-17.1865	93.99188	94.01333
5	-16.125	-12.5083	90.40983	91.36019
Mean	-14.643	50.0336	93.1655	26.8388
SD	1.7524	59.2531	1.5452	90.598
SE	0.7837	26.4988	0.6911	40.5167

Env-v2

Trial	Control	v1	v2	v3
1	-14.5755	95.42666	94.65293	94.57904
2	-15.0882	94.93613	94.6002	-73.8755
3	-15.6519	94.87491	94.53074	96.27587
4	-14.6119	-12.3565	95.21931	29.50771
5	-13.3061	-7.19404	94.15264	3.706951
Mean	-14.6467	53.1374	94.6312	30.0388
SD	0.8673	57.4606	0.3829	70.7895
SE	0.3879	25.6972	0.1712	31.658

Env-v3

Trial	Control	v1	v2	v3
1	-8.91994	96.19836	96.20784	95.61583
2	-10.1013	96.2236	96.0852	96.26328
3	3.545134	96.27587	96.09423	95.95878
4	-8.84714	29.50771	96.37998	96.19583
5	-2.90653	3.706951	96.17461	96.23012
Mean	-5.446	64.3825	96.1884	96.0528
SD	5.7575	44.5563	0.1192	0.272
SE	2.5749	19.9262	0.0533	0.1217

Fig.3. Testing summary tables for all
Agent types for all environments