Investigating Generalisation of Reinforcement Learning Algorithms

Aaron Dovey Oliver Simmonds Owen Parsons Benjamin Mardel Max Rooney

***Abstract*—This report explores the generalisability of modern reinforcement learning (RL) algorithms using the OpenAI Gym environments. The core of our methodology consisted of tuning an algorithm in one environment and assessing its performance across domains, utilising an adaptation of the 'inter-algorithm normalisation' method to standardise testing. We adapted Q-learning to ensure compatibility in continuous environments as well as created a new loss function when tuning the Proximal Policy Optimisation (PPO) algorithm to optimise the trade-off between training time and performance. Our findings show that Q-Learning successfully generalised across simple domains, but failed in the more complex environment *LunarLander*. Extending the algorithm via a neural network (NN) to a Deep Q Network (DQN) remediated this failure. In contrast, the PPO algorithm performed well across domains, but tended to over-fit, leading to the DQN being the most generalisable algorithm we investigated.**

I. Introduction

RL has demonstrated breakthrough performances in a variety of domain-specific tasks, including healthcare [1], the automotive industry [2] and gaming (e.g., DeepMind's AlphaGo [3]). However, recent studies have demonstrated that both RL and deep RL agents have a limited ability to generalise across domains, due to agents over-fitting to their training environments [4]. Our project investigates how well RL agents tuned for one environment can generalise, which is crucial to real world applications where agents must often operate in environments that are different from those they encounter during training. There is no universal framework to assess generalisation capabilities and this project builds off existing methods that rely on cross-validation. We compare the performance of Q-Learning, Deep Q Networks (DQN) and Proximal Policy Optimisation (PPO) in OpenAI Gym environments [5] and assess the impact of Bayesian hyper-parameter (HP) tuning. We identify the HPs which generalise best for each algorithm, state the effect the environments had on generalisation and thus conclude which algorithm generalises best according to this framework.

II. Literature Review

A. Generalisation

Farebrother et al. [6] evaluated generalisation using a novel method of training agents in one environment and testing them in a variation that shares key features. Their results found that algorithms such as DQN exhibit an over-fitting trend. As corroborated by Machado et al. [7], an algorithm which learns by memorising a sequence of actions will inherently perform well in a deterministic environment, but fail to generalise across tasks. Furthermore, Machado et al. highlight that while over-fitting is a common problem in RL, under-fitting can also occur in more complex environments where there is a large degree of variability and uncertainty. Advances in deep RL have also shown promising results in addressing these challenges, such as the use of multi-task learning techniques [8].

B. Algorithms

Q-learning is a temporal difference (TD) learning algorithm, first introduced by Prof. Chris Watkins and has laid the framework for modern RL agents. It is based on model-free learning and can operate in a stochastic environment [9], making it a suitable choice for studying generalisation. The natural extension to Q-learning is to introduce multiple neural networks forming a deep RL algorithm, a method largely attributed to the work of Sutton et al. [10]. This is particularly useful when exploring the entire state space becomes computationally infeasible and it is common practice for DQNs to be comprised of one network for estimating the Q values and another for selecting actions to take [11]. In contrast to TD-learning which estimates a value function, policy gradient methods are a family of RL algorithms that directly optimise the policy. OpenAI's PPO is currently the gold standard in policy gradient based methods and incorporates actor and critic agents as well as using a trust region to guide the optimal policy search [12]. This project thus investigates Q-learning, DQN and PPO covering a relatively broad range of modern RL algorithms.

III. Methods

A. The OpenAI Gym

The majority of our investigation takes place in the OpenAI Gym, a widely used toolkit for developing and comparing RL agents. It hosts a variety of different environments to explore, each being fundamentally different from the next with its own set of rules and goals. This ranges from simple environments such as *CartPole*, with just a two dimensional observation space, to more complex environments such as *LunarLander* where there are multiple ways to attain rewards and a higher dimensional state space.

B. Framework

Our system to investigate generalisation used Bayesian HP optimisation to tune RL algorithms in the Gym Classic Control environments: *CartPole*, *MountainCar* and *Acrobot*.

This gave us three tuned models for Q-learning and PPO. We then cross-validated performance by testing the algorithm tuned for one environment on the other two environments. We selected the *BayesOpt* [13] package when tuning HPs since it has been shown that Bayesian optimisation techniques can outperform random search and grid search, by tracking past evaluations to inform its future choices [14]. We found that within `BayesOpt`, a good proportion of random iterations (`init_point`) to Bayesian iterations (`n_iter`) was 1:10. In order to assess the effectiveness of the found HP combination, we conducted experiments where the algorithm was trained using those HPs and its performance was evaluated over 100 runs across the three environments.

To provide a benchmark for evaluating the performance of our algorithms, we compare our results to Stable Baselines [15] - a Python library that offers high-quality implementations of state-of-the-art RL algorithms built on top of OpenAI Gym. It has an easy-to-use interface supporting various algorithms such as PPO and DQN [1] and allowed us to efficiently compare our models against a good standard. To maintain consistency when comparing our results to Stable Baselines', we used their approach of capping the attainable maximum and minimum rewards of the environments, e.g. capping *CartPole* at 500. However, we found that PPO was unable to train in the environments with the minimum caps of -500, as used for Q-learning and DQN. Instead PPO required a minimum of -4000 in *MountainCar* and -1000 for *Acrobot*, and hence more time during each episode before it learned to take a sequence of actions which yielded reward. This can be explained by PPO's trust region optimisation process, which ensures the policy update is not too large. This can result in a small change in policy, requiring a larger amount of experience to accumulate before updating the policy significantly. Conversely, DQN updated the Q-values after each action, making it more efficient in terms of sample size compared to PPO. Thus, if the environment requires a large number of steps to achieve a positive reward (as is the case in *MountainCar*), PPO may require a larger number of steps per episode to accumulate enough experience to update the policy significantly.

To further investigate the generalising capabilities of our strongest models, we concluded the analysis by testing them on *LunarLander*. This is a more complex environment due to its larger observation space and more complicated reward system.

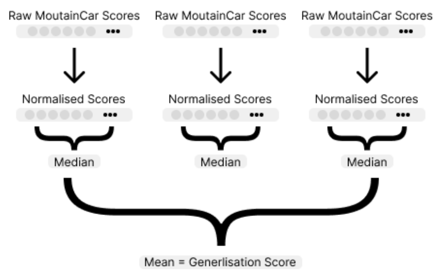
Each environment varies in difficulty and its rewards can be positive or negative, creating a problem when comparing scores across games. We found it appropriate to normalise rewards in order to compare the performance across environments on a standardised scale. Our project differs from the experimentation in Bellemare et al. [16] by quantifying the effect the tuning environment had on how well an algorithm generalises. To do this we modified their 'Inter-Algorithm Normalisation' method to 'Inter-Environment Normalisation'.

[1]Note: Stable Baselines does not support Q-learning on the Classic Control environments.

We normalised the reward from each environment using the following formula:

$$s_{norm} = \frac{(s - r_{min})}{r_{max} - r_{min}}$$

where s_{norm} is the new normalised reward, s is the raw reward from our test data and $[r_{min}, r_{max}]$, is the minimum and maximum rewards achieved in that environment. Having normalised every reward, we then calculated the median reward achieved in the environment. The final score for each of the tuned algorithms is calculated as the mean of the median scores. We call this the 'Generalisation Score', a measure of how well a tuned algorithm generalises. Figure III-B illustrates the steps involved in this calculation. Note that a high generalisation score is indicative of good generalisation.



'

C. Modifying the algorithms

1) Q-Learning: Unlike a typical Q-learning agent, our model required a `discretise_state` function, since Q-learning works with a discrete table and the Gym environments investigated have a continuous observation space. The function maps the continuous space onto a discrete space by partitioning each dimension of the observation space. The number of partitions was chosen via Bayesian optimisation and was held fixed for each game when executing the cross-validation.

2) Deep Q Network (DQN): Regular DQN's remember past experiences through their weights and biases, however we found that this alone was producing a very unstable network. As suggested by Zhang et al. [17], storing previous experiences in the form of a replay buffer, for the network to access during training, proved helpful in minimising variations in test results. During Bayesian HP tuning we encountered challenges with the time costs, which limited our ability to obtain rigorous results. We instead manually fine-tuned the parameters to identify specific values that were effective, as suggest by Kiran et al. [18].

3) Proximal Policy Optimisation (PPO): As with Q-learning we added a modification to the PPO algorithm. This was because PPO performed very well on *CartPole* before HP tuning, receiving the maximum reward of 1000 (an artificial cap) after only a few training epochs. Only testing performance could not usefully distinguish between sets of HPs that resulted in agents reaching maximum scores. Consequently we included a penalty term to encourage the agent to train faster whilst maintaining high performance. Our tailored loss function for Bayesian optimisation was:

$$-1000^{*}\texttt{E} + \texttt{sum\_5\_tests},$$

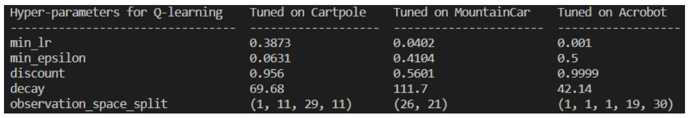
where `E` is the first training epoch to produce a reward of over 1000 and `sum_5_tests` is the sum of the rewards of five test runs during epoch `E`. Consequently, a larger `E` term was penalised whereas the `sum_5_tests` encouraged continued high performance after the first time the model reaches 1000 reward. The '1000' term in the loss function was selected after trial and error to see what value would best balance performing highly against shorter training times. The expensive time costs of HP tuning called for carefully refining the space we optimised over, making our choices in line with OpenAI's original PPO paper [12] and the Stable Baselines 3 documentation [19]. The resulting Bayesian optimisation used `init_points = 15` and `n_iter = 150`. We noted that OpenAI's paper found that variations which used a clip ratio were more successful than those using a KL penalty when tested on robotic tasks in the MuJoCo Gym environment. As a result, we decided to use a clipping variation of PPO throughout as well as encompassing the range of clip ratios that OpenAI tested in our own HP tuning.

IV. Experimentation and Analysis

A. *Q-Learning*

Overall we found that, despite producing some unexpected results, Q-learning performs adequately on simple games, see Figure 2 and 3, with all tuned agents scoring fairly well on both *MountainCar* and *Acrobot*. However, not all of our Q-learning agents generalised well. In particular, the Q-learning agent tuned for *MountainCar* performed badly in *CartPole*, receiving an average score of just 26. The Q-learning agent tuned on *CartPole* received the worst scores when tested on the other environments. This is likely due to *CartPole* having the objective of staying alive, the opposite to *MountainCar* and *Acrobot*. Furthermore, the agent tuned on Acrobot outperformed the agent tuned on *MountainCar* in the *MountainCar* environment, beating it at its own game. We believe this is due to the decay rate found by tuning *Acrobot* being higher than that found in *MountainCar*, see Figure 1. We hypothesise that making a random action in *Acrobot* is more detrimental, due to it being a more sensitive game to sub-optimal actions, causing the agent to act more carefully than the *MountainCar* agent in its own environment. Another reason as to why the agent tuned in *Acrobot* beat *MountainCar* could simply be that Bayesian HP tuning is not guaranteed to find the optimal HPs.

We found that tuning times were not consistent across environments due to the episode lengths varying game to game. In games such as *CartPole*, where the objective is to stay alive, the episode lengths naturally increase as the agent learns. For the majority of training, the agent exhibits poor performance, finishing quickly. This is the opposite of what happens in *MountainCar* and *Acrobot* where the objective is to finish quickly and poor episodes take longer to complete. Since we tuned the agent over a fixed number of episodes, the environments which have the longest episodes on aggregate take the longest time to train on, explaining why *CartPole* tuned the quickest. Furthermore, *Acrobot* was the slowest to tune as it had the largest state/observation space and thus the largest Q-table. From these results, the longer a game took to tune the better it was at generalising, see Figure 4. This could be down to the fact that the two games which took long to tune were more similar to each other than to *CartPole*. Q-learning trained faster than other algorithms which allowed us to do a large number of `BayesOpt` iterations, with `init_points = 50` and `n_iter = 500`. See Figure 1 for the HPs found.



```
Hyper-parameters for Q-learning    Tuned on Cartpole    Tuned on MountainCar    Tuned on Acrobot
-------------------------------    -----------------    --------------------    -----------------
min_lr                             0.3873               0.0402                  0.001
min_epsilon                        0.0631               0.4104                  0.5
discount                           0.956                0.5601                  0.9999
decay                              69.68                111.7                   42.14
observation_space_split            (1, 11, 29, 11)      (26, 21)                (1, 1, 1, 19, 30)
```

Fig. 1: Hyper-parameters found through Bayesian Optimisation for the Q-learning algorithm.

We conclude that the agent tuned on *Acrobot* was the best at generalising for Q-learning, receiving the highest generalisation score of 0.688, see Figure 4. Hence we tested this model's performance on a much harder OpenAI environment, *LunarLander*.

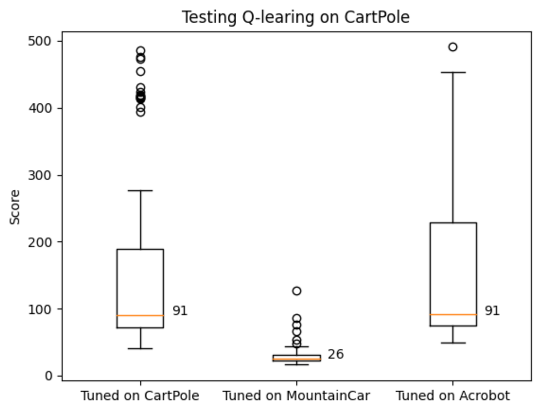


Fig. 2: Testing scores achieved by our Q-Learning agent tuned on *CartPole*.

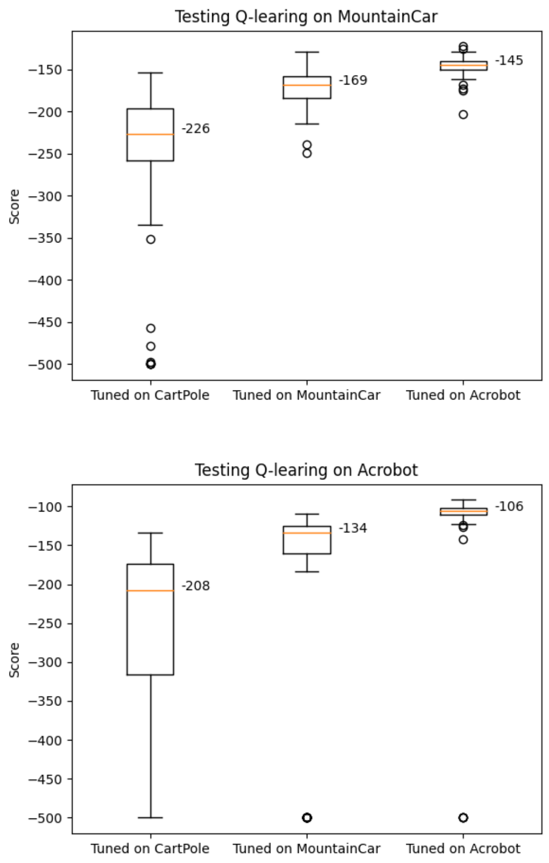


Fig. 3: Testing scores achieved by our Q-Learning agent tuned on *MountainCar* and *Acrobot*.

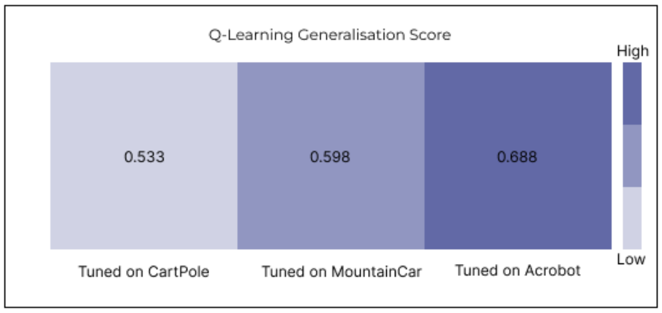


Fig. 4: Gradient map of the 'Generalisation Scores' for for the three tuned Q-learning agents. Darker tones are indicative of higher scores

B. Deep Q Network (DQN)

Both our and Baselines' DQN agents achieved only the maximum score of 500 in *CartPole*, see Figure 5. Testing in *MountainCar*, our agent outperformed Baselines' but exhibited more outliers and had a significantly narrower inter-quartile range. In terms of generalisation, our DQN agent performed well across all three domains learning the various spaces and optimal policies within just one thousand episodes. This is all considering the agent had the same parameters for all three problems indicating that they were well chosen. To be confident that this was not due to the simple nature of the three environments we again later transition onto *LunarLander*.

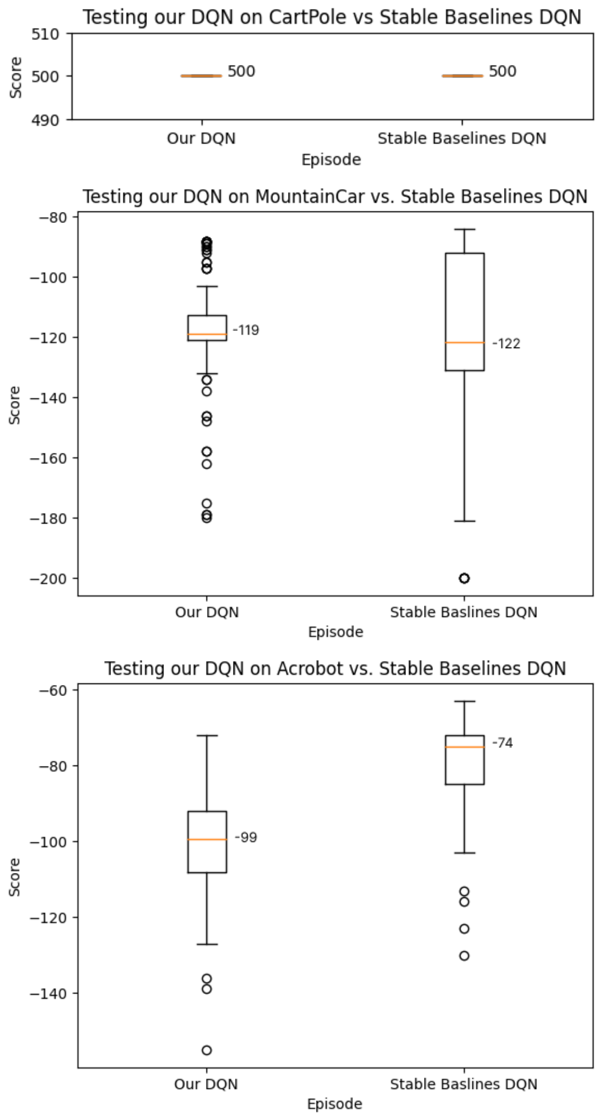


Fig. 5: Testing scores achieved by our DQN and Stable Baselines' DQN in each of the three environments.

C. Proximal Policy Optimisation

Figure 6[2] shows an example of training on *CartPole* for 35 epochs for each tuned agent after implementing the loss function mentioned in the method section. This plot demonstrates the strength of PPO, with all models completing the game within 250 training episodes (averaging over 195). Moreover, our tailored loss function seems relatively successful, achieving maximum reward with

[2]Note: Training each set of HPs runs for a different number of episodes. This is because we trained for 35000 steps (as opposed to episodes), so the algorithms which averages a high number of steps per episode runs for less episodes during training.

shorter training times than the *Acrobot* tuned model. However, the *MountainCar* tuned model reached 1000 reward in the smallest training time. This result is unexpected, similar to how *Acrobot* beat *MountainCar* at its own game in the Q-learning section, which may again be due to tuning not being guaranteed to find optimal HPs. As shown by Figure 8 and 9 , our PPO performed well on each environment it was tuned for and on the whole matched the achievements of Stable Baselines' PPO. However, some agents did not generalise well, as shown by the models tuned in *CartPole* and *Acrobot* performing extremely poorly in the *MountainCar* environment. This suggests that HP tuning on these environments caused PPO to over-fit. On the other hand, as confirmed by the generalisation score in Figure 10, the agent tuned on *MountainCar* performed relatively well on all environments, suggesting this PPO agent is strongest at generalising. This is contrary to what was found with Q-learning, where *Acrobot* was found to be the best environment for generalisation. See Figure 7 for the optimised PPO HPs found through Bayesian optimisation.

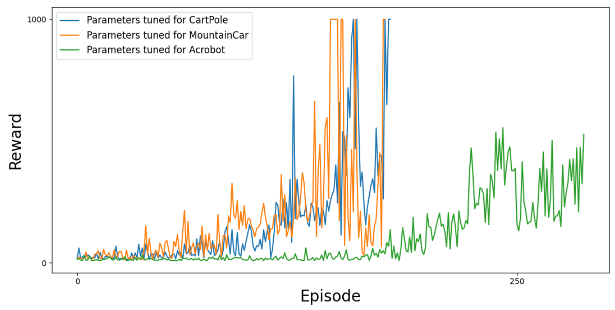
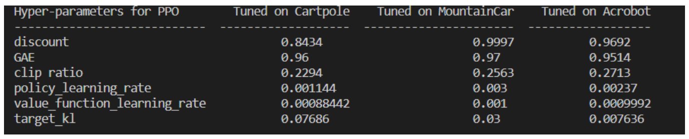


Fig. 6: Testing scores achieved in CartPole by all three tuned PPO agents.



```
Hyper-parameters for PPO        Tuned on Cartpole    Tuned on MountainCar    Tuned on Acrobot
----------------------------  -------------------  ----------------------  ------------------
discount                                   0.8434                  0.9997           0.9692
GAE                                        0.96                    0.97             0.9514
clip ratio                                 0.2294                  0.2563           0.2713
policy_learning_rate                       0.001144                0.003            0.00237
value_function_learning_rate               0.00088442              0.001            0.0009992
target_kl                                  0.07686                 0.03             0.007636
```

Fig. 7: Hyper-parameters found through Bayesian Optimisation for our PPO algorithm.

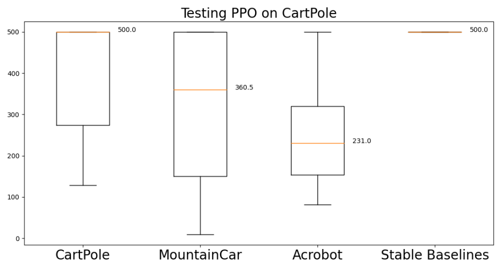


Fig. 8: Testing scores achieved by our PPO agent tuned on *CartPole*. Stable Baselines' PPO scores with default HPs are also given.

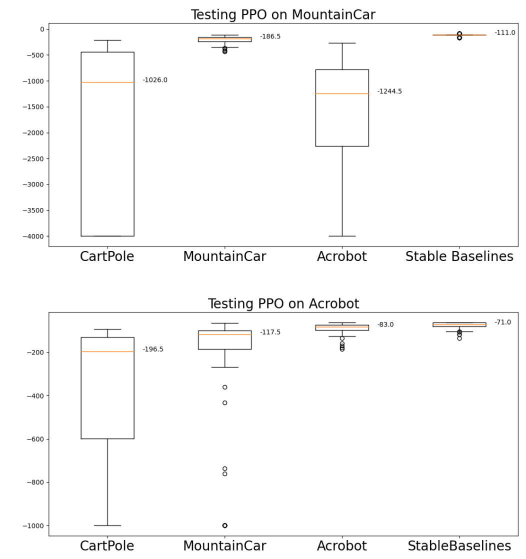


Fig. 9: Testing scores achieved by our PPO agent tuned on *MountainCar* and *Acrobot*. Stable Baselines' PPO scores with default HPs are also given.

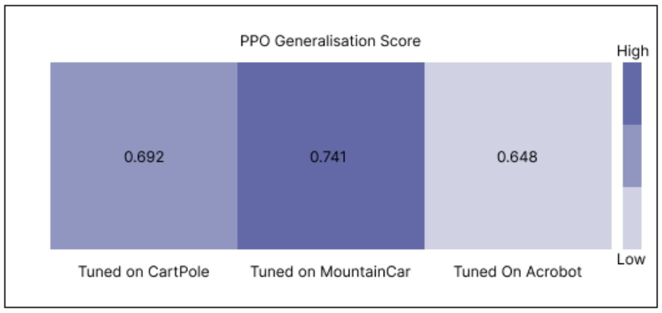


Fig. 10: Gradient map of the generalisation scores for the three tuned PPO agents. Darker tones are indicative of higher scores.

D. *LunarLander*

After ranking our models based on their generalisation score, we tested our strongest models on *LunarLander*, namely Q-learning using *Acrobot*'s HPs, PPO using *MountainCar*'s HPs, and the manually tuned DQN. Although Q-learning received higher rewards than an agent taking random actions, it never achieved a positive score, see Figure 11. When rendering the testing episodes, we saw this algorithm used a strategy of crashing into the landing pad instead of trying to land the ship upright. This is in contrast to our observations of the DQN agent which received negative rewards, primarily as a result of spending thrust on landing in the correct zone rather than crashing. However, our DQN was highly successful and as demonstrated in Figure 11, actually surpassed Stable Baselines' DQN in median rewards and had far fewer outliers. Similarly to previous sections, our PPO struggled to generalise, only outperforming our Q-Learning agent according to both the median reward and the inter-quartile range seen in Figure 11.

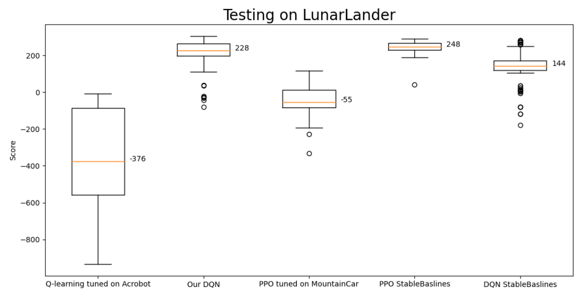


Fig. 11: Performance of our Q-learning and PPO models with highest generalisation score, our DQN and Stable Baselines' on LunarLander.

V. Conclusion

In this project, we investigated the generalisability of RL agents across different OpenAI Gym environments through HP tuning and cross-validation. Our findings suggest that Q-learning performed well in simpler environments but struggled with *LunarLander*. In contrast, DQN demonstrated good generalisation performance across all environments. However, PPO did not generalise well and over-fitted, heavily relying on being tuned to a specific environment in order to do well in that environment. This is evidenced by the most general PPO model tuned for *Acrobot* receiving subpar rewards in *LunarLander*. These results highlight the importance of selecting appropriate HPs for specific tasks to achieve optimal performance. Moreover, we show the tuning environment tends to have a significant impact on how well the tuned model generalises. We finally claim that DQN was our most general RL algorithm.

VI. Future Work

We now suggest future research directions to build upon our findings. One flaw in our methodology was to only test our agents on three games. Although all environments were distinct, two of the games being more similar than the other naturally gives a skewed judgment of the true generalisability of our models. We have understood that some models may appear to generalise well on a test set but do not perform well in unseen environments, as was seen with the *Acrobot* tuned Q-learning agent's failure in *LunarLander*. Zhang et al. [20] and Cobe et al. [21] both reached the conclusion that increasing the number of training environments improved generalisation and hence a natural progression to our work would be to introduce many more environments into our experimentation. In practice this would require us developing a convolution layer for our algorithms so that we could investigate more complex environments.

We would also recommend the Procgen Benchmark [22]. Procgen (also OpenAI) offers similar features as Gym however the key difference is that the environments are procedurally generated and all seem to require a convolutional layer to interpret the state space. Following this route would take our work one step closer to emulating how the real world works with every episode differing slightly from the last. If one took the route of using more environments a natural progression would be to use higher than 1-fold cross validation. Since we used a relatively small number of training environments, using a higher number of folds would be sub optimal. However, using Procgen could allow one to perform k-fold cross validation leading to more accurate estimates of the model's performances.

References

[1] A. Esteva et al., "A guide to deep learning in healthcare," Nat Med, vol. 25, no. 1, pp. 24–29, Jan. 2019, doi: 10.1038/s41591-018-0316-z. [Online]. Available: https://www.nature.com/articles/s41591-018-0316-z.

[2] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," Robotics and Autonomous Systems, vol. 114, pp. 1–18, Apr. 2019, doi: 10.1016/j.robot.2019.01.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889018302021.

[3] M. Lapan, Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Packt Publishing Ltd, 2018.

[4] C. Zhao, O. Sigaud, F. Stulp, and T. M. Hospedales, "Investigating Generalisation in Continuous Deep Reinforcement Learning." arXiv, Feb. 20, 2019 [Online]. Available: http://arxiv.org/abs/1902.07015.

[5] "Gym Documentation." [Online]. Available: https://www.gymlibrary.dev/.

[6] J. Farebrother, M. C. Machado, and M. Bowling, 'Generalization and Regularization in DQN'. arXiv, Jan. 17, 2020. doi: 10.48550/arXiv.1810.00123.

[7] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, 'Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents'. arXiv, Nov. 30, 2017. doi: 10.48550/arXiv.1709.06009.

[8] Hessel, Matteo, et al. "Multi-task deep reinforcement learning with popart." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.

[9] "Q Learning." [Online]. Available: https://cml.rhul.ac.uk/qlearning.html.

[10] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction. Cambridge, Mass: MIT Press, 1998.

[11] V. Mnih et al., 'Human-level control through deep reinforcement learning', Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236. [Online]. Available: https://www.nature.com/articles/nature14236.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy optimisation Algorithms." arXiv, Aug. 28, 2017 [Online]. Available: http://arxiv.org/abs/1707.06347.

[13] F. Nogueira, "bayesian-optimisation: Bayesian optimisation package." [Online]. Available: https://github.com/fmfn/Bayesianoptimisation.

[14] A. H. Victoria and G. Maragatham, "Automatic tuning of hyperparameters using Bayesian optimisation," Evolving Systems, vol. 12, no. 1, pp. 217–223, Mar. 2021, doi: 10.1007/s12530-020-09345-2. [Online]. Available: https://link.springer.com/10.1007/s12530-020-09345-2.

[15] 'Welcome to Stable Baselines docs! - RL Baselines Made Easy — Stable Baselines 2.10.3a0 documentation'. [Online]. Available: https://stable-baselines.readthedocs.io/en/master/

[16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, 'The Arcade Learning Environment: An Evaluation Platform for General Agents', jair, vol. 47, pp. 253–279, Jun. 2013, doi: 10.1613/jair.3912.

[17] S. Zhang and R. S. Sutton, "A Deeper Look at Experience Replay." arXiv, Apr. 30, 2018 [Online]. Available: http://arxiv.org/abs/1712.01275.

[18] M. Kiran and M. Ozyildirim, "Hyperparameter Tuning for Deep Reinforcement Learning Applications." arXiv, Jan. 26, 2022 [Online]. Available: http://arxiv.org/abs/2201.11182.

[19] "PPO — Stable Baselines3 1.8.0a7 documentation." [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html.

[20] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, 'A Study on Overfitting in Deep Reinforcement Learning'. arXiv, Apr. 20, 2018. [Online]. Available: http://arxiv.org/abs/1804.06893

[21] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, 'Quantifying Generalization in Reinforcement Learning'. arXiv, Jul. 14, 2019. doi: 10.48550/arXiv.1812.02341.

[22] 'Procgen Benchmark'. [Online]. Available: https://openai.com/research/procgen-benchmark.