Space Invaders Reinforcement Learning

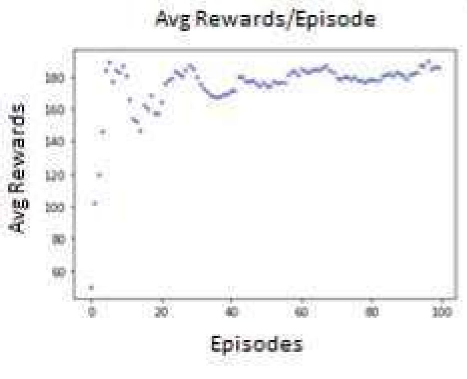
I have chosen to teach my RL agent to play the Space Invaders autonomously from the Atari learning environment which comes from Open AI Gym[v] using the rllib libraries. I selected to learn the policy for the agent to play the game using the DQN algorithm to begin with.

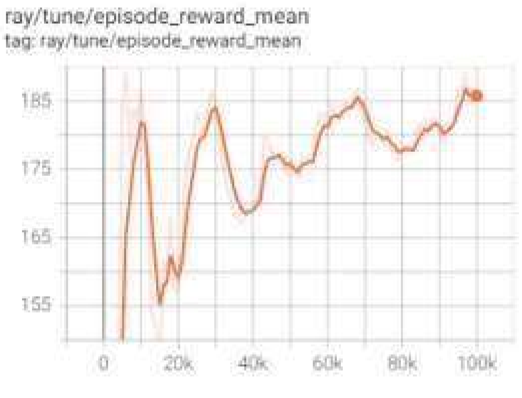
DeepMind founded the Deep Q-Network (DQN) algorithm back in 2015.[vi] This was a big deal in the world of reinforcement learning because this algorithm could figure out a huge spectrum of Atari games, where the input would be numerous images (pixels) of the game and the output is a value function forecasting future rewards. The Q-Value method gives rise to a prohibitively high processing and memory requirement. To improve on this a DQN algorithm is utilised. A DQN algorithm is created by merging Q-Learning with convolutional neural networks (CNN) and a strategy called experience replay. Experience replay enables storage in memory of the episode steps and then random samples are taken at random from this memory. [vii]

The reason I chose to apply DQN is because this works better for complex environments with many states built on the weights. It requires less memory for when the states become bigger whereas a Q-Table would require more as the size of the Q-Table increases. DQN is a well-recognised widely used algorithm. DQN can be applied for discrete actions and Torch models. [viii] Space invaders has 6 discrete action spaces ['NOOP', 'FIRE', 'RIGHT', 'LEFT', 'RIGHTFIRE', 'LEFTFIRE'] and I will also be using a torch framework, so this model seemed relevant to choose. Initially I experimented the interaction with the environment for 5 episodes using a random policy meaning the game will play using the random action spaces and the scores prove just this randomness. First episode gives a high score, then low and the declines to 30 in the 5th episode.

```
Episode: 1 score:260.0
Episode: 2 score:290.0
Episode: 3 score:110.0 Episode: 4 score:45.0
Episode: 5 score:30.0
```

Next, I built the DQN algorithm and used the default configurations. [ix] Some overridden hyperparameters I chose were fully connected networks hidden layers as [64,64] and the activation function as "tanh". After configuration the DQN model is run for 100 episodes and the results are shown below. I also applied TensorBoard for cooler graphs.



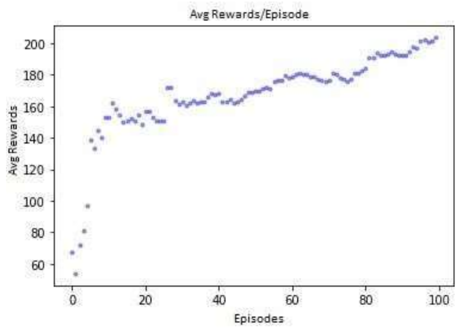


The above graph shows the relationship between average rewards and episodes. As the number of episodes increase, the score generated from playing space invaders increases maxing out at around a score of 185. The upward sloping curve indicates that our agent is learning.
Further tuning of the hyperparameters was then carried out. The three parameters that we are going to control the activation function and learning rate as shown below.

Hidden Layers	{[128, 128]]}
Activation Function	{"relu", "swish"}
Learning Rate	{[3e-4, 7e-4, 1e-3]}
Buffer Size	3000

The above gridsearch produced the best parameters below:
{'env': 'SpaceInvaders-v0', 'framework': 'torch', 'model':
{'fcnet_hiddens': [128, 128], 'fcnet_activation': 'relu'}, 'lr': 0.0005,
'buffer_size': 3000}

These parameters were then applied to our original DQN model to see if any improvement has been observed.

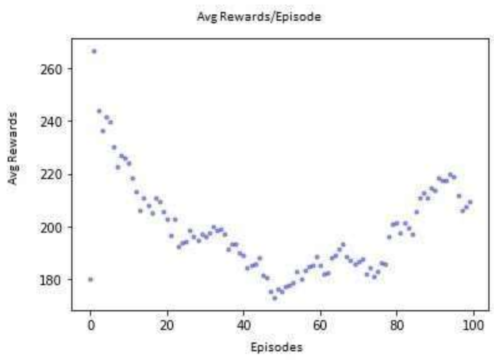


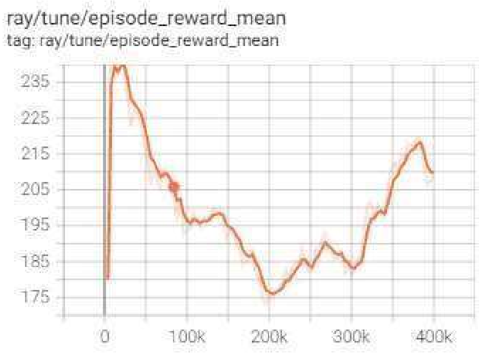
Notice that with a bit of hyperparameter tuning, we have managed to achieve a higher average reward/score of around 200 after running for 100 episodes. This took a long time to run (especially the hyperparameters). With an extended larger grid search run over a higher number of episodes can potentially give better results (converge faster) or we can apply improvements to the DQN algorithm in the future.

The Proximal Policy Optimisation (PPO) algorithm was chosen to also apply to the same Space Invaders environment. PPO is another algorithm that can be applied to reinforcement learning problems. PPO adjusts its existing policy at each phase in order to improve on certain parameters and is referred to as a policy gradient optimisation algorithm. It guarantees that the new policy is consistent with the previous policy, therefore significant updates are not accepted. PPO employs the actor-critic technique, which entails the use of two deep neural networks. The actor is responsibility for the action, while its critic is in control of the rewards.[x]

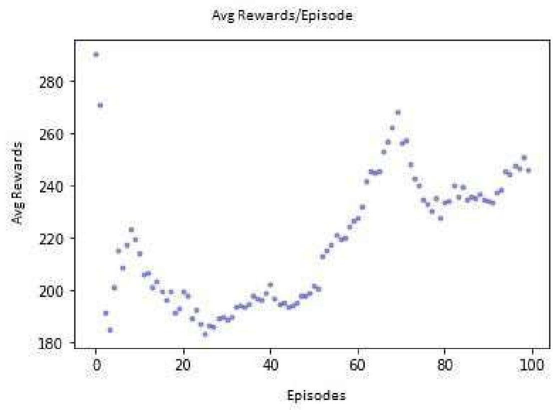
PPO was chosen as my second algorithm as it is one of the most widely used policy optimisation algorithms in reinforcement learning. Since our environment is discrete, PPO can be applied to discrete action spaces. We follow the same initial steps as the DQN model in terms of experimenting with a random episode and thus receiving random scores over 5 episodes.

The PPO algorithm was built, and the default configurations used from the ray documents. Here I chose hidden layers as [25,25] for my initial run of the model. The model was run over 100 episodes and the results are shown below. It seems the PPO has achieved high rewards initially but then had a performance drop before slowly rising again towards the 100 episodes mark. One of the disadvantages in policy gradient algorithms is that they are very sensitive to parameter changes. My motivation to apply a PPO was to make the new policy consistent with the old policy but I saw massive drop in performance as shown below. We could run improvements by using Value Function rescaling or a hyperparameter search.





Although in the code we have specified a wider hyperparameter search that we can do. I ran a smaller hyperparameter search of just the activation function of [“tanh”, ”swish”]. Using the best hyperparameters back in our PPO model, the graph looks slightly better of 100 episodes however still has a sudden performance drops as we can observe two instances in the below graph. Possibly over running higher number of episodes this graph would be more smoothed.



References

1. Sutton, R. S., & Barto, A. G. (2018). A Beginner's Guide to Q-Learning. *Towards Data Science.* [Online]. Available at: [https://towardsdatascience.com/a-beginners-guide-to-q-learningc3e2a30a653c](https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c) (Accessed:
[Access Date]).

2. Sahoo, S. (2020). Bellman Equation and Dynamic Programming. *Medium.* [Online]. Available at: [https://medium.com/analytics-vidhya/bellman-equation-and-dynamic-programming-773ce67fc6a7](https://medium.com/analytics-vidhya/bellman-equation-and-dynamic-programming773ce67fc6a7) (Accessed: [Access Date]).

3. Verma, S. (2021). Solving Lunar Lander (OpenAI Gym) - Reinforcement Learning. *Medium.* [Online]. Available at: https://shiva-verma.medium.com/solving-lunar-lander-openaigym-reinforcement-learning785675066197 (Accessed: [Access Date]).

4. OpenAI. (n.d.). Lunar Lander - OpenAI Gym Environment. *OpenAI Gym.* [Online]. Available at: https://gym.openai.com/envs/LunarLander-v2/ (Accessed: [Access Date]).

5. OpenAI. (n.d.). Space Invaders - OpenAI Gym Environment. *OpenAI Gym.* [Online]. Available at: https://gym.openai.com/envs/SpaceInvaders-v0/ (Accessed: [Access Date]).

6. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529-533. [Online]. Available at: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf (Accessed: [Access Date]).

7. DeepMind. (n.d.). Deep Q-Networks (DQN). *Papers with Code.* [Online]. Available at: https://paperswithcode.com/method/dqn#:~:text=A%20DQN%2C%20or%20Deep%20Q,each%20action%20as%20an%20output (Accessed: [Access Date]).

8. Ray Team. (n.d.). RLlib Algorithms Overview. *Ray Documentation.* [Online]. Available at: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html (Accessed: [Access Date]).

9. Ray Team. (n.d.). DQN - Ray RLlib Algorithm Documentation. *Ray Documentation.* [Online]. Available at: [https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#dqn](https://docs.ray.io/en/latest/rllib/rllibalgorithms.html#dqn) (Accessed: [Access Date]).

10. Dharmendra, B. (n.d.). A Brief Introduction to Proximal Policy Optimization. *GeeksforGeeks.* [Online]. Available at: [https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policyoptimization/](https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policy-optimization/) (Accessed:
[Access Date]).

Additional References:

11. PyTorch. (n.d.). Reinforcement Q-Learning Tutorial. *PyTorch Tutorials.* [Online]. Available at: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html (Accessed: [Access Date]).

12. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602.* [Online]. Available at: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf (Accessed: [Access Date]).

13. Kaptan, V., Pechenizkiy, M., & Puuronen, S. (2020). Multi-step deep Q-networks with experience replay. *Neural Networks, 129,* 273-286. [Online]. Available at: https://www.sciencedirect.com/science/article/pii/S0896627320308990 (Accessed: [Access Date]).

14. California State University, San Bernardino. (n.d.). Space Invaders. *CSUSBDT GitHub.* [Online]. Available at: [https://github.com/csusbdt/441-2015/wiki/Space-Invaders](https://github.com/csusbdt/4412015/wiki/Space-Invaders) (Accessed: [Access Date]).

15. Ray Team. (n.d.). RLlib Colab Notebook. *Google Colab.* [Online]. Available at: https://colab.research.google.com/github/rayproject/tutorial/blob/master/rllib_exercises/rllib_colab.ipynb#scrollTo=i3FVvEJRNlhy (Accessed: [Access Date]).

16. deeplizard. (Year). Exploration vs. Exploitation - Learning the Optimal Reinforcement Learning Policy. *YouTube.* [Online]. Available at: https://www.youtube.com/watch?v=hCeJeq8U0lo (Accessed: [Access Date]).

17. Torres, J. (Year). The Bellman Equation. V-function and Q-function Explained. *Towards Data Science.* [Online]. Available at: [https://towardsdatascience.com/the-complete-reinforcement-learning-dictionarye162

30b7d24e#:~:text=Greedy%20Policy%2C%20%CE%B5%2DGreedy%20Policy,Agent%20to%20explore%20at
%20all](https://towardsdatascience.com/the-complete-reinforcement-learning-
dictionarye16230b7d24e#:~:text=Greedy%20Policy%2C%20%CE%B5%2DGreedy%20Policy,Agent%20to%2
0explore%20at%20all) (Accessed: [Access Date]).