CISC 856 - Reinforcement Learning

Group 1 - Space Invaders

Mahmoud Mohamed Abdien
Mohamed Makram Abo Elsood
Osama Ali El Awadia
Sara Ahmed Elfetiany

Jun 7, 2022

Problem Formulation

Space Invaders is a part of the Atari environments, its objective is to destroy the space invaders by shooting its laser cannon and avoiding shooting targets from the invaders. The agent has six actions, and the optimal solution is as mentioned before (i.e., to destroy the invaders). We have found an interesting paper published by Google DeepMind [1], in which they improved the performance of regular DQN (Deep Q-Network) by 164.11%. We also found some promising algorithms that worth a shot, such as Keras-RL and PPO-LSTM included in Stable-Baselines3 community contribution.

Implementation and Results

At first, we have proposed that we would implement Space Invaders using Stable-Baselines, Keras-RL, and Tensorforce. But here we would stick with only Stable-Baselines and Keras-RL as we needed to use both Stable-Baselines2 and Stable-Baselines3 to implement different algorithms as we will see ahead.

The algorithms that are implemented using Stable-Baselines:

- The algorithms we have implemented, and provided by Stable-Baselines3:
  - Vanilla DQN
  - QR-DQN
  - PPO LSTEM
- It was noticed that Stable-Baselines3 framework supports only vanilla DQN (i.e., it doesn't support Dueling DQN) but Stable-Baselines2 does.[2]

- The algorithms we have implemented, and provided by Stable-Baselines2:
  - Vanilla DQN
  - Dueling-DQN
  - PPO2

It was also noticed that Stable-Baselines2 supports TensorFlow version from 1.8.0 to 1.15.0, so we have downgraded the TF version we have. Figure 1 shows Stable-Baselines2 prerequisites.[3]

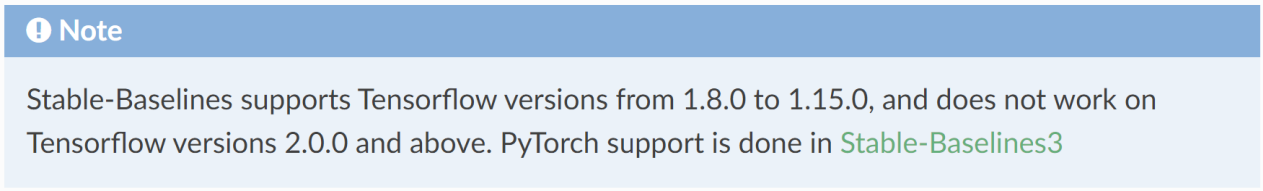

Figure 1: Stable-Baselines2 Prerequisites [3]

Notes about Vanilla DQN, Dueling-DQN, and QR DQN:

- It is a model free approach; meaning the states and rewards are produced by the environment.
- It is an off policy algorithm, because these states and rewards are obtained with a behavior policy independent from the online policy that is being learned.
- It uses memory reply: instead of only using the current experience as prescribed by standard Temporal Difference learning, the network is trained by sampling mini-batches of experiences.
- It proves maximum reward better than that provided by vanilla DQN.
- Figure 2 shows the difference between vanilla DQN and Dueling-DQN regarding the maximum reward; by setting the timesteps to 100k.
- Figure 3 shows the maximum reward by increasing the timesteps to 2M in the vanilla DQN and 1M in the Dueling-DQN.

Figure 2: Vanilla DQN vs Dueling DQN with 100k timesteps

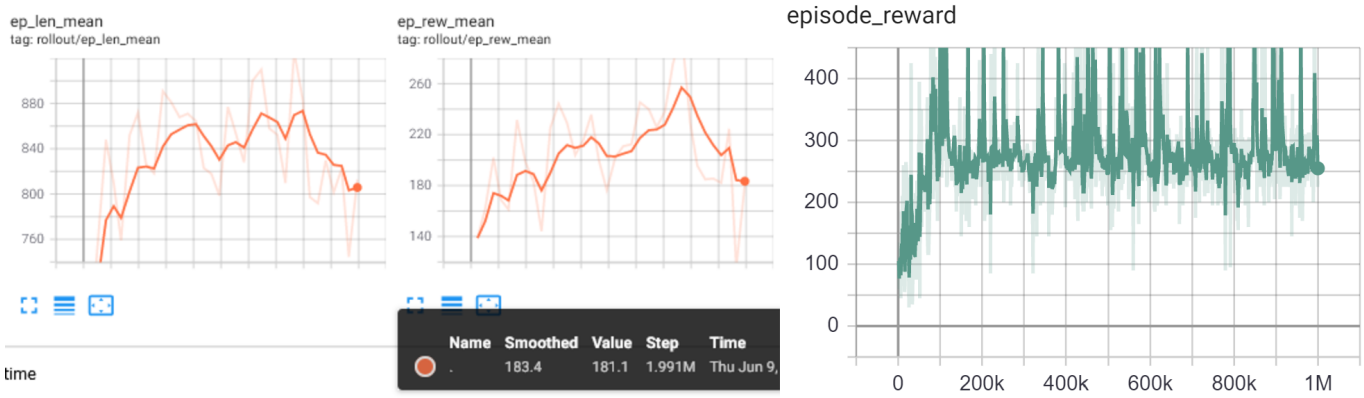

Figure 3: Vanilla DQN with 2M timesteps vs Dueling DQN with 1M timesteps

Notes about Proximal Policy Optimization (PPO):

- It is also a model free algorithm.
- It's an on-policy algorithm
- This is usually an on-line method, where only the most recent transitions taken by the agent is used to update the policy.
- It is a policy gradient method which alternates between sampling data through interaction with the environment, and optimizing an objective function using stochastic gradient ascent. The fundamental idea is to directly update the policy to make the probability of taking actions which provide a larger future reward more likely to occur, it uses trajectories taken by the agent.
- Figure 4 shows reward mean using PPO LSTM with 100k timesteps.

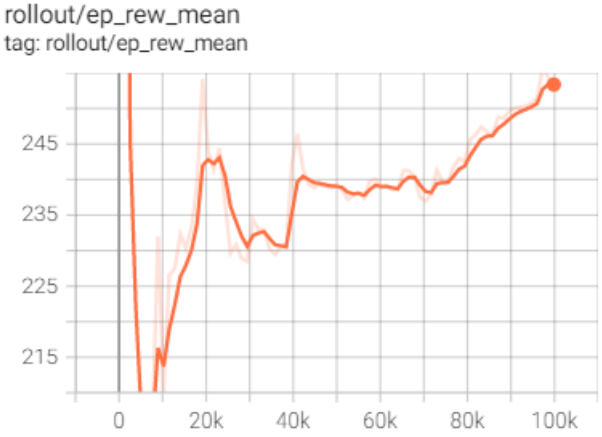

Figure 4: PPO LSTM

- Out of all algorithms we have explored, PPO-LSTM algorithm proved outstanding behaviour in remembering which values to update and which to forget.
- Figure 5 shows the performance of PPO-LSTM vs Vanilla DQN.

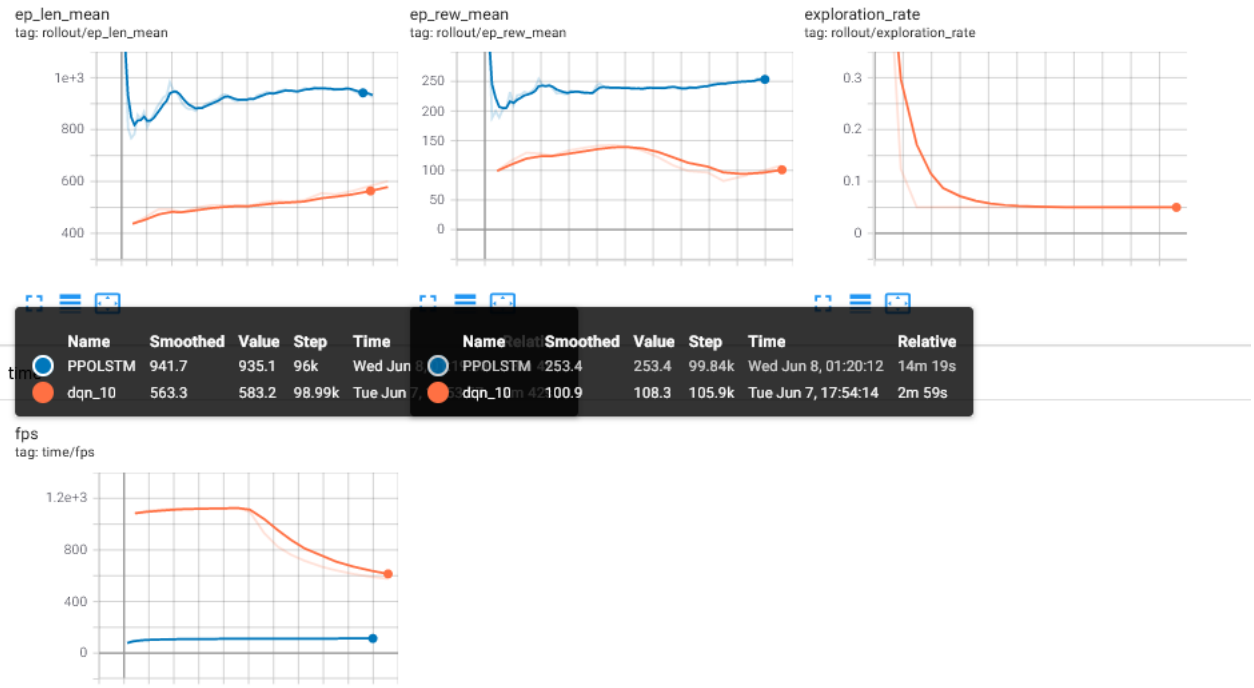

Figure 5: PPO LSTM vs Vanilla DQN

- Table 1 shows the maximum reward reached by different algorithms.

Timesteps	**Maximum Reward**					
	Stable-Baselines3			**Stable-Baselines2**		
	Vanilla DQN	**PPO LSTM**	**QR-DQN**	**Vanilla DQN**	**Dueling-DQN**	**PPO2**
100K	250	300	250	350	400	275 (35K timesteps)
1M	...	280	...	...	500	...
2M	350	...	...	...	...	...

Table 1: Maximum reward for different algorithms

⇒ Videos explain the agent's behaviour (please, click the hyperlinks to see the videos):

- This video shows the agent isn't moving till losing the game, we increased the timesteps to 1 million steps but the agent still in a fixed pose.
- The QR-DQN was suffering from the same issue.
- We tried Keras-RL algorithm, the agent starts to slightly move and evade.
- The PPO-LSTM model shows an outstanding performance out of all models.

Major Obstacles

- Behaviour: The main issue with the agents' behaviour in the DQN models (Vanilla DQN, Dueling DQN, and QR-DQN) was that the model freezed where they are, until they get killed and lose the game
- We needed a model that takes actions more frequently so it can evade, we also needed to have a memory so we tried 2 different methods: PPO & DQN with hindsight reply
- DQN-Hindsight is the current state of the art, recently Stable-Baseline3 merged it with other algorithms and it's no longer a separate model, we tried making a reply buffer with hindsight but there is too many parameters that we still don't understand
- We increased the number of steps and we increased epsilon.
- In Stable-Baselines3 Vanilla DQN, it didn't improve even after one million steps, in 2 million steps the model behaved even worse and couldn't evade shooting target more quickly.
- We updated frequancy from 10 episodes to 20 episodes and there was still no improvement, we followed the paper of Dueling-DQN but we found it is not supported in Stable-Baselines3 so we downgraded to Stable-Baselines2.
- We tried Dueling-DQN and got more stable increase in reward in the first 100K, so we increased the timesteps to one million steps.
- We couldn't train all the models we have for longer than two million steps due to Colab crashes. Community benchmarks suggests that Space Invaders get solved in 4-7 million steps.
- We searched for newer models to solve the game in fewer time steps, we found Stable-Baslines Contrib, which is a collection of community-based algorithms made with the official Stable-Baselines3, one of the promising models that we found is QR-DQN.[4]
- We then switched from DQN to PPO and we noticed a more exploratory game play, and impressive evasions to shooting targets.
- After multiple attempts, we suspected that the model is not evading because he doesn't have time to react, so we implemented our own network with Keras-RL, and we used larger kernel size so the model would get to see the shooting targets in more recent time frames.
- We explored multiple contributed agents other than the official releases (found Recurrent-PPO with LSTM policy most interesting)
- Hyperparameter tunning was crucial, it was hard to fine tune parameters of all 5 variations so we stick with community guidelines from Antonin RAFFIN to avoid Colab crashes.
- For DQN(vanilla, dueling or q) all agents had similar rewards and similar behavior with timesteps lower than one million, the model starts learning after one million steps.
- For PPO-LSTM model, we get promising results as mentioned before. Now, PPO-LSTM is under maintenance and consequently, we can't use it with more timesteps and so on.
- We faced some issues regarding displaying TensorBoard generated plots in Colab but we ran them (i.e., TensorBoard logs) locally and all logs are attached to the submission on onQ.

Conclusion

From the previous observations, we would stick with PPO-LSTM which was the model that gives the most promising results. We would also consider running the algorithms (that gave us the best results) with more than five million timesteps which is more promising to let the agent reach the end of the game as the winner. The usage of parameters provided by community guidelines as mentioned before is beneficial and time saver.

References

[1] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.

[2] R. the Docs, "Stable-baselines3 dqn." https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html.

[3] R. the Docs, "Stable-baselines2 prerequisites." https://stable-baselines.readthedocs.io/en/master/guide/install.html.

[4] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.