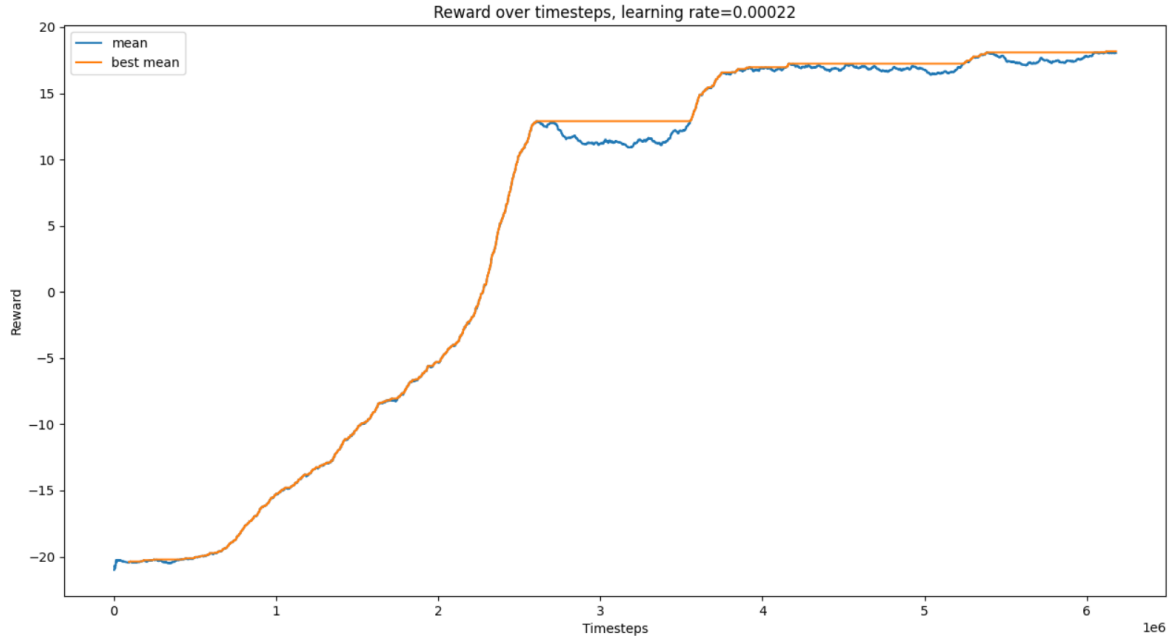
Reinforcement learning project

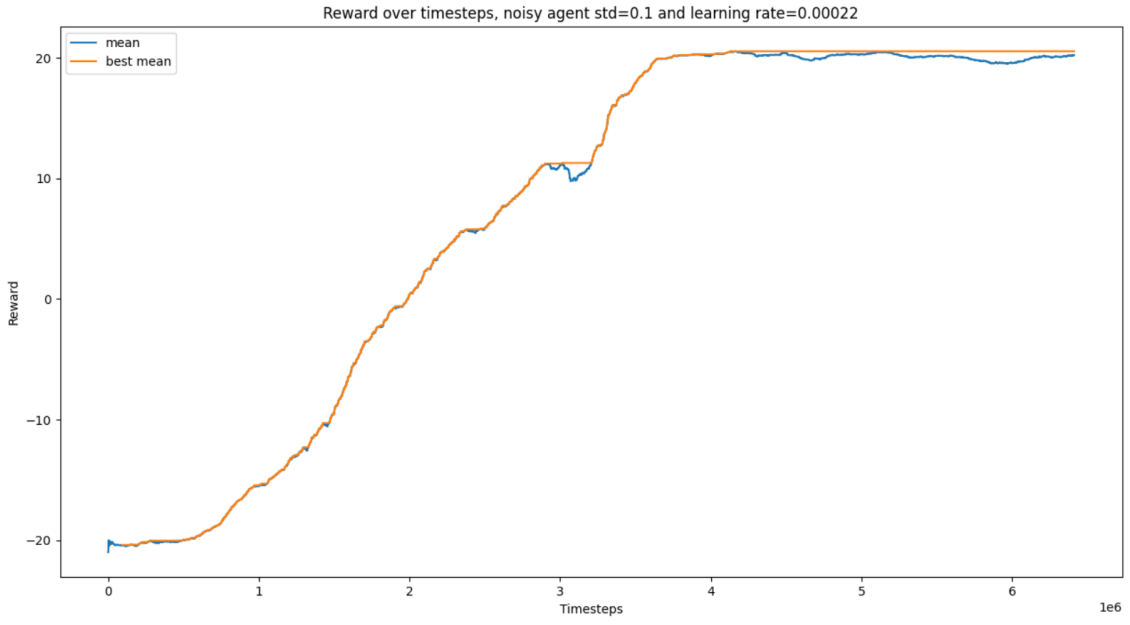
Or Cohen
Gal Amit
Ofek Anixter

Questions:

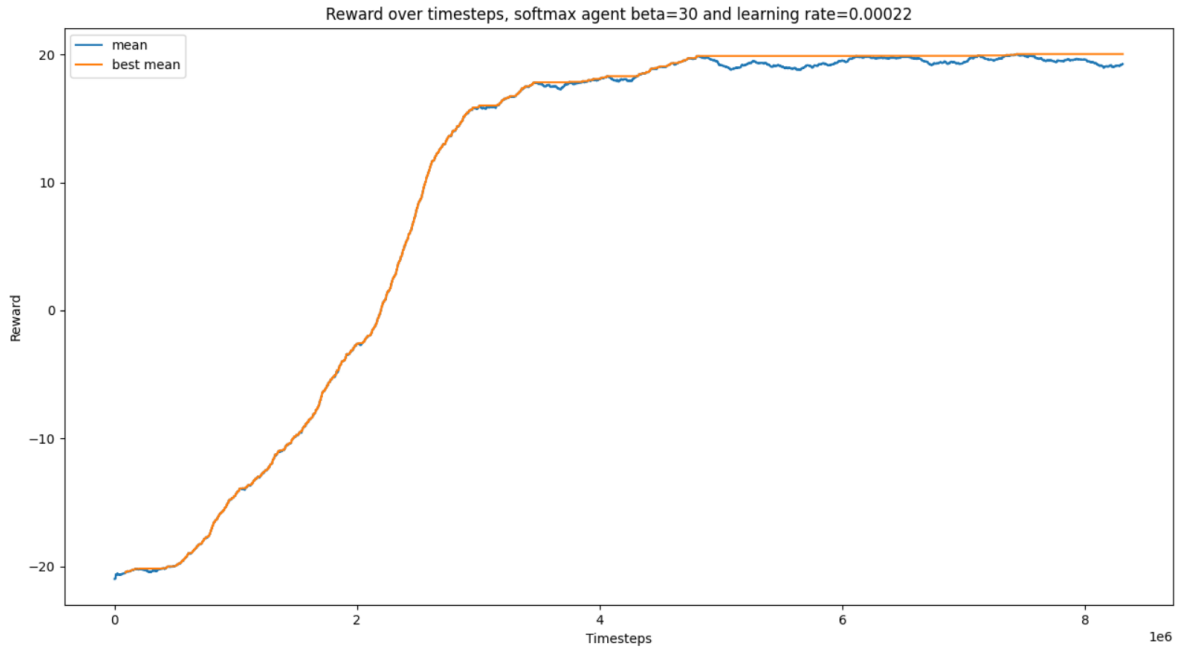
1. After an extensive hyperparameter sweep, we discovered that adjusting the learning rate parameter to 0.00022 and keeping all other hyperparameters with their default values(except for agent type) resulted in the optimal performance for the DQN network when applied to the Atari Pong game gave us a reward of 18.17 and stayed on 18.1 reward values.



However, considering that the agent type is a hyperparameter and changing it together with the learning rate gave us better results as can be seen for noisy agent with a standard deviation of 0.1 gave us a 20.54 reward and stayed on 20 reward values.



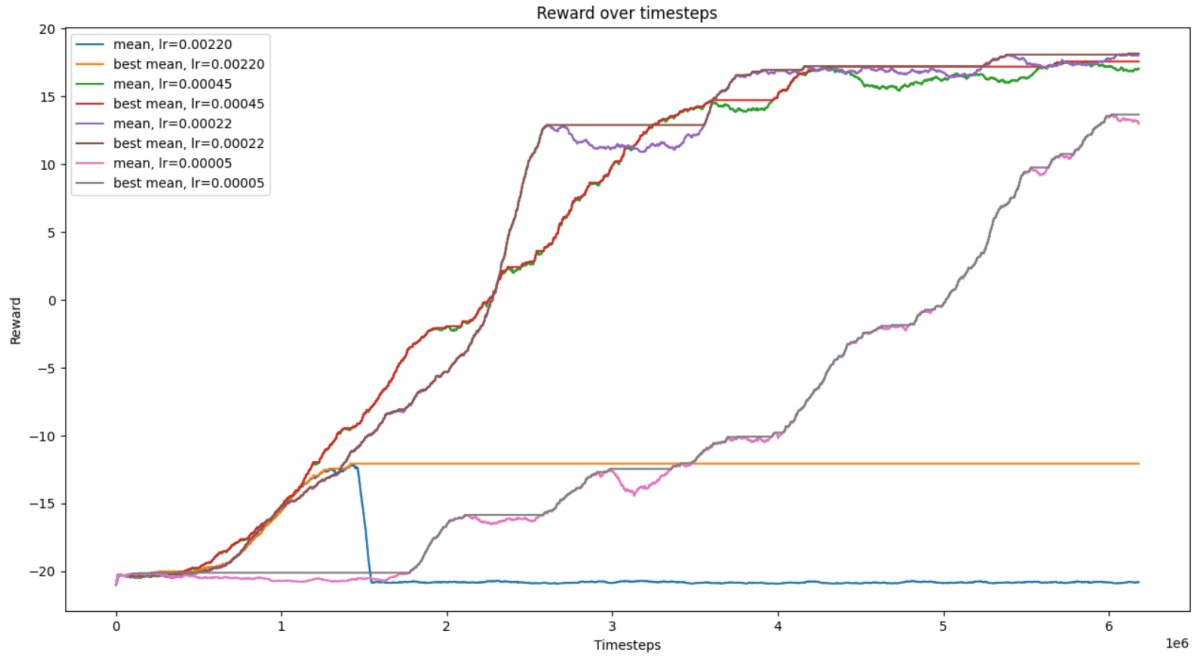
Also for the softmax agent with beta=30, we got a better reward than the epsilon greedy agent of gave us a 20.03 reward and stayed on 19.8 reward values.



2. We will do two experiments changing the agent type and learning rate.

Changing learning rates:

The learning rate is a critical hyperparameter that determines the step size at which the DQN network updates its weights during training. A well-chosen learning rate can significantly influence the network's ability to converge to an optimal solution and improve its performance over time. In our case, we decided to focus on adjusting the learning rate because we knew it played a significant role in the convergence of the solution. Additionally, compared to modifying other aspects of the neural network, such as its architecture, the learning rate is a single value that is easier to analyze and compare across different experiments.



After extensively exploring various learning rate values, we determined that a learning rate of 0.00022 produced the most favorable outcomes compared to the other values we tested. Interestingly, the learning rate curves for 0.00045, 0.00022, and 0.00005 exhibited similar patterns and reached their respective peak rewards at approximately 6,000,000 timestamps. Furthermore, it is worth noting that both the learning rates of 0.00045 and 0.00005 yielded suboptimal results. This observation suggests that the optimal learning rate lies within the range bounded by these values.

In addition, we observed that the learning rate of 0.0005 required a substantial number of steps before demonstrating any improvement in the reward. This behavior aligns with the characteristics of low learning rates. Furthermore, the result of 13.7 can be attributed to the limitations imposed by the low learning rate, which hindered the network's ability to achieve higher rewards.

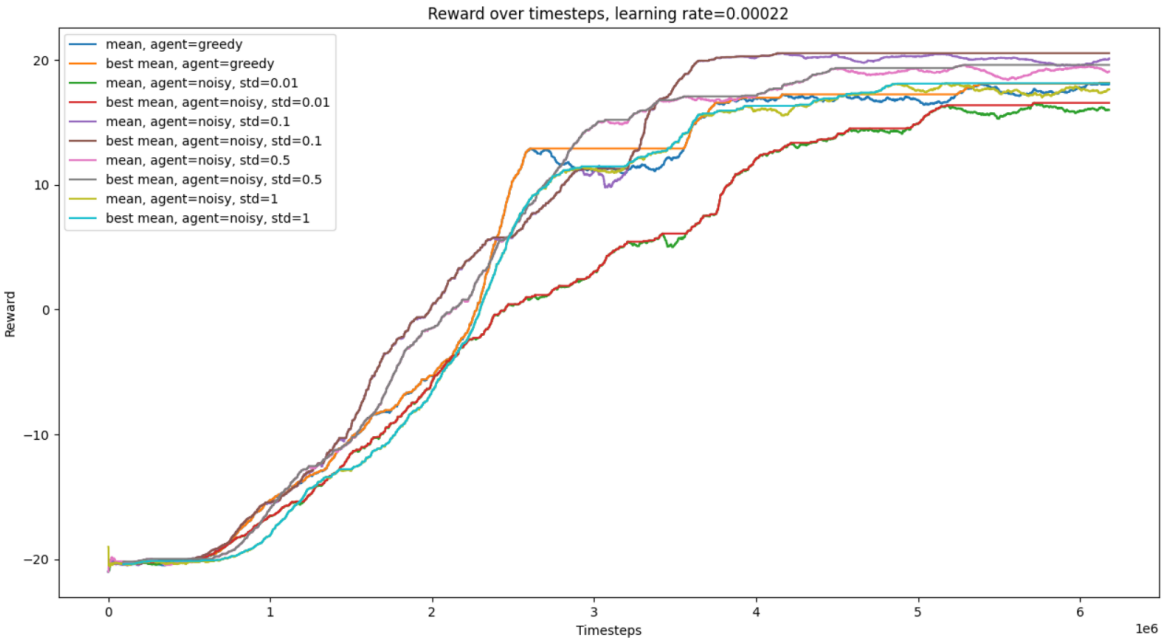
The learning rate of 0.0045 could be too high, as it caused the network to overshoot the optimal solution. Consequently, the network's performance was suboptimal, with a score of 17.58. However, it is worth noting that this score is very close to the maximum result achieved with a learning rate of 0.00022.

The learning rate of 0.00220 was considered too high for our purposes. High learning rates can lead to undesirable divergent behavior in the loss function, which was indeed observed in our case. The network experienced overshooting, resulting in worse performance compared to all other tested learning rates.

Alternatives for epsilon greedy:

Noise for exploration [3]

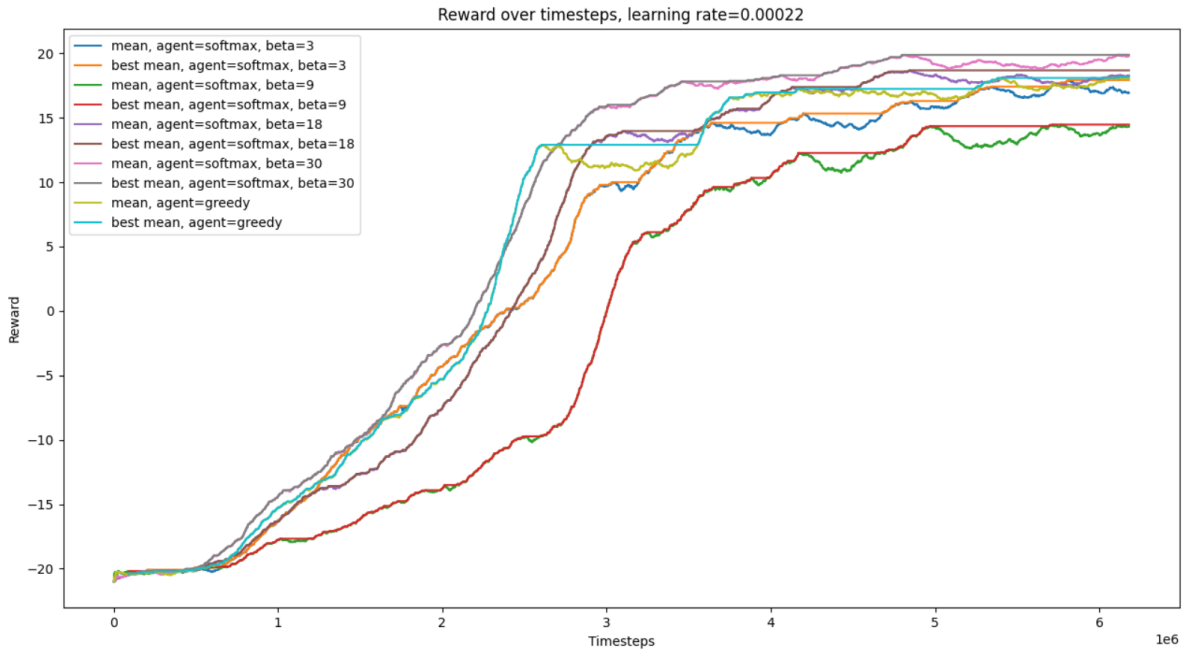
We aim to investigate the potential benefits of incorporating noise into the DQN (Deep Q-Network). Introducing noise to the network's weights can enhance exploration in a more efficient manner compared to traditional exploration techniques like epsilon-greedy. By employing NoisyNet, which incorporates parametric noise, we can achieve a broader range of exploration and potentially achieve significantly higher scores across various Atari games. This exploration technique has shown promise in improving performance, including the possibility of surpassing human-level performance. We will examine whether integrating noise into our network aligns with the aforementioned claims and enhances our network's performance. We will evaluate and compare the results obtained using the default parameters with those achieved by solely modifying the learning rate to the optimal value of 0.00022 for the noisy agent we used different standard deviation values.



The results demonstrate that employing a noisy agent in the network yields significant improvements over the regular DQN network with a greedy agent. With a standard deviation of 0.1, the noisy agent achieved a reward of 20.54, surpassing the reward of 18.17 obtained by the greedy agent. Similarly, for a standard deviation of 0.5, the results improved, although the reward remained suboptimal at 19.6. These findings suggest that incorporating noise with mediocre standard deviation values in the network enhances performance by enabling effective exploration. However, it is worth noting that using a small standard deviation of 0.01 does not provide sufficient exploration, leading to worse results compared to the greedy agent and other standard deviation values. Conversely, employing large standard deviations, such as 1, introduces excessive exploration, preventing the network from converging to the optimal solution. Nonetheless, even with a standard deviation of 1, the noisy agent still achieves results comparable to the greedy agent. Furthermore, it is notable that the network with a noisy agent exhibits faster convergence towards its own maximum score compared to the DQN network with a greedy agent. This suggests that the noisy agent promotes efficient learning and quicker improvement in performance. In summary, the experiments indicate that the noisy agent outperforms the greedy epsilon agent, providing better results, faster convergence, and a favorable trade-off between exploration and exploitation.

Softmax for exploration

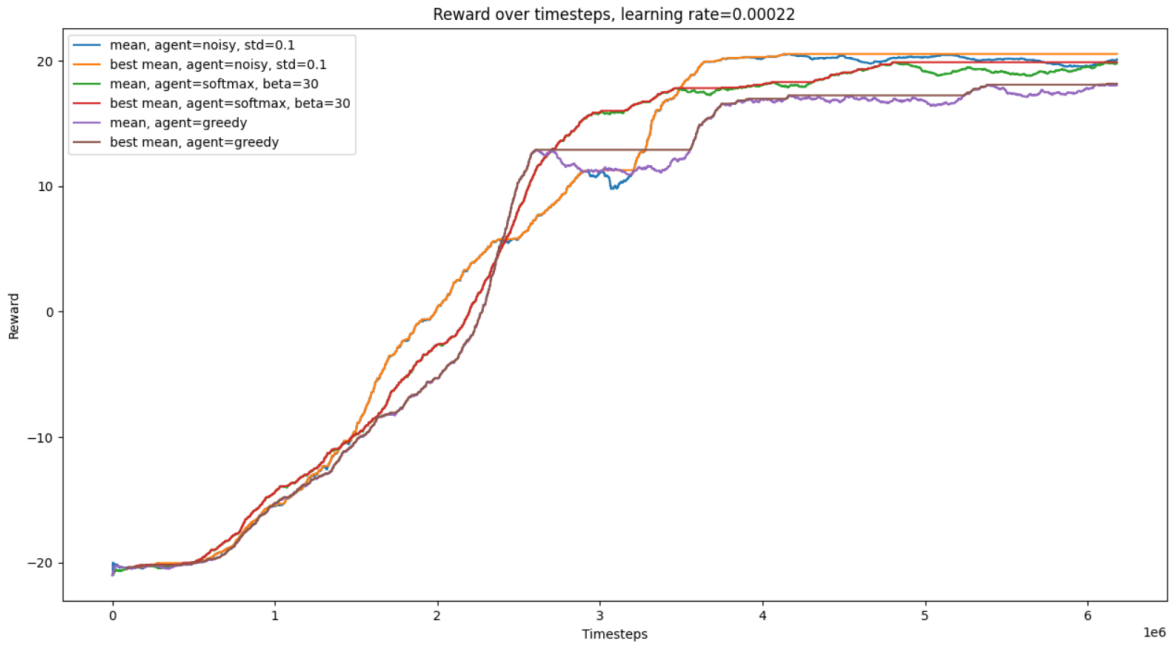
A softmax agent offers advantages over a purely greedy agent in several aspects. Firstly, the softmax agent promotes exploration by assigning non-zero probabilities to all actions, allowing it to systematically explore the environment and potentially discover better action choices. In contrast, a greedy agent always selects the action with the highest estimated value, which can lead to suboptimal decisions in unexplored areas. Secondly, the softmax function provides flexibility through its temperature parameter. By adjusting the temperature, we can control the trade-off between exploration and exploitation. Higher temperatures encourage more exploration, while lower temperatures prioritize actions with higher values. This adaptability is beneficial in scenarios where different levels of exploration are required at various stages of learning. Additionally, the softmax agent handles uncertainty inherent in estimated action values. The probabilities assigned by the softmax function reflect the agent's confidence or uncertainty in those values. This capability is especially valuable when there is noise in the estimation process or when the estimated values are imprecise. In contrast, a greedy agent does not consider uncertainty and may make overconfident decisions solely based on current estimated values. While the choice between softmax and greedy agents depends on the specific problem, softmax offers a more robust and flexible exploration-exploitation strategy in complex and uncertain environments. By using a softmax agent, we aim to assess if it can provide better results compared to the greedy agent in our reinforcement learning task.



For small beta values, such as 3 and 9, the agent tends to prioritize exploitation over exploration. As a result, it may focus on actions with higher estimated values and exploit the currently known knowledge. However, this approach may limit the agent's ability to explore new areas of the environment and discover potentially better actions. Consequently, we observed lower results compared to the greedy agent. As we increased the value of beta, we observed a shift towards more exploration. The agent assigned higher probabilities to actions with lower estimated values, allowing for a more thorough exploration of the environment. This increased exploration led to improved results. For example, with a beta of 18, we achieved a slightly better reward of 18.68 compared to the greedy agent. Furthermore, as we continued to increase beta, encouraging even more exploration, we observed a further improvement in performance. For instance, with a larger beta value of 30, the agent achieved a reward of 20. This suggests that the increased exploration facilitated by higher beta values allowed the agent to discover more effective actions and ultimately achieve better results.

Summary

In conclusion, it appears that utilizing agent types such as noisy or softmax with a higher beta value of 30 yields better results compared to the epsilon greedy agent. These alternative agent types promote exploration and provide a better balance between exploration and exploitation, leading to improved performance in reinforcement learning tasks.



Bonus

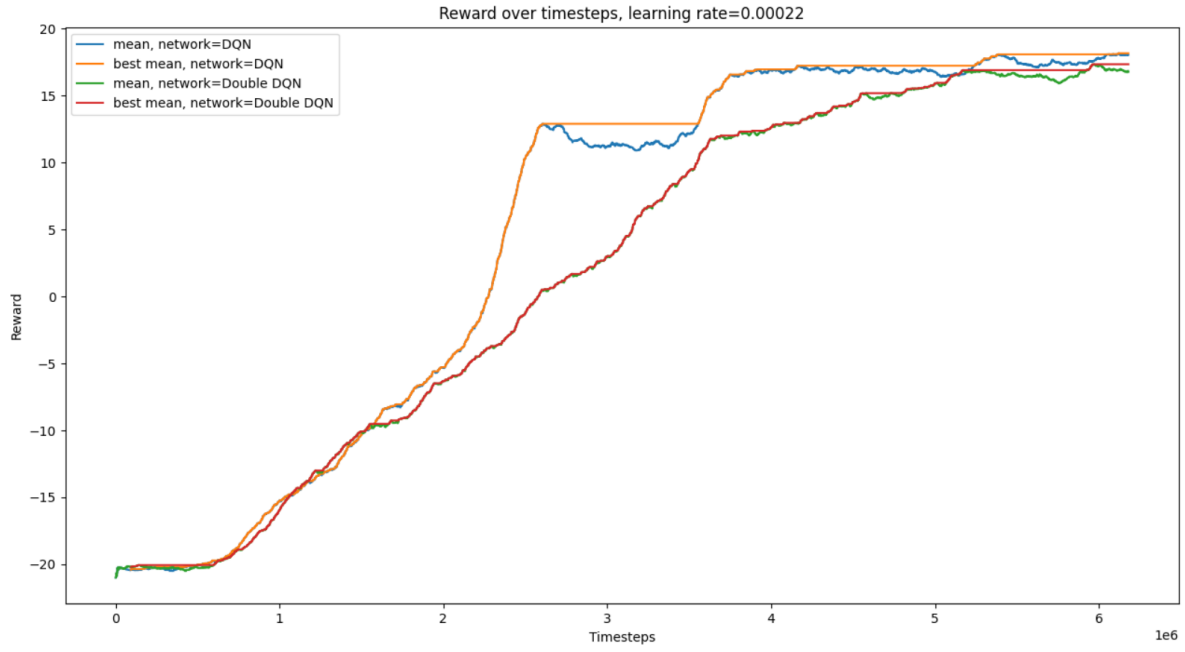
1. Double DQN [1]

Double DQN (DDQN) is a variant of DQN that addresses the overfitting problem by using two neural networks: a "main" network and a "target" network. The main network is used to select actions, while the target network is used to estimate Q-values. The target network is updated less frequently than the main network, which helps to prevent the target network from becoming too optimistic.
We would like to check if DDQN will outperform the DQN on the Atari Pong for this we decided to build the DDQN class in the dqn_modle.py

DDQN has been shown to outperform DQN on a variety of tasks. Here are some of the reasons why DDQN is better than DQN:

- DDQN is less likely to overfit, which can lead to improved performance.
- DDQN can learn more quickly than DQN, which can lead to faster convergence.
- DDQN is more robust to noise in the environment, which can lead to improved performance in real-world applications.

We will evaluate and compare the results obtained using the default parameters with those achieved by solely modifying the learning rate to the optimal value of 0.00022.

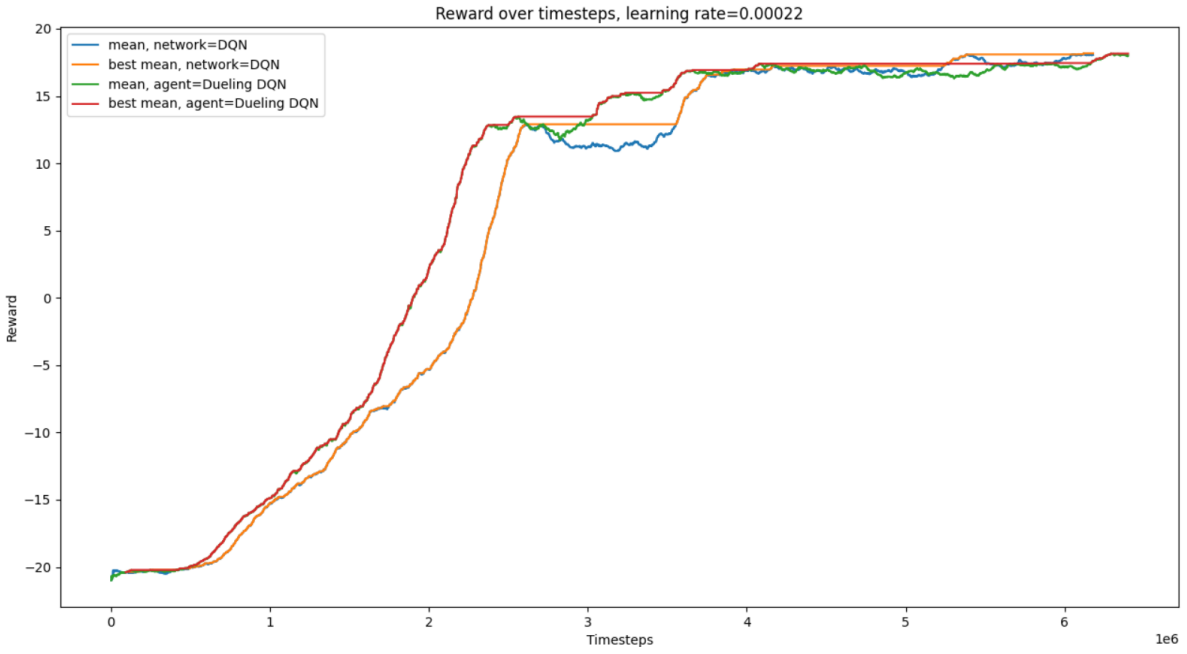


The results we obtained indicate that the DQN network (with a reward of 18.17) performed better than the DDQN network (with a reward of 17.35), which was unexpected. Several factors may have contributed to this outcome:
1. The performance of the Double DQN algorithm is sensitive to hyperparameter settings, such as the learning rate, discount factor, and exploration strategy. It is possible that the hyperparameters used in your experiments were not optimal for the

Double DQN network, leading to suboptimal results compared to the standard DQN.
2. Training duration: Reinforcement learning algorithms, including Double DQN, often require extensive training to achieve optimal performance. Insufficient training time or a limited number of training episodes could prevent the Double DQN network from fully converging and reaching its true potential, which can be seen that indeed the curve of the DDQN network is more moderate.

2. Dueling DQN [2]

We aim to evaluate the performance of Dueling DQN compared to the standard DQN algorithm in the Atari Pong game. To achieve this, we have implemented the DuelDQN class in the dqn_model.py file. The Dueling DQN architecture separates the estimation of the state value function and the advantage function, allowing for more efficient learning and improved action selection accuracy. By explicitly modeling the state value and advantage separately, Dueling DQN can better handle state-value approximation and reduce bias. This architectural modification has shown promising results in various reinforcement learning tasks by promoting effective exploration and enhancing learning efficiency. In order to determine whether Dueling DQN surpasses the performance of DQN on the Atari Pong game, We will evaluate and compare the results obtained using the default parameters with those achieved by solely modifying the learning rate to the optimal value of 0.00022.



Both the DQN network and the Dueling DQN network achieved similar rewards, with scores of 18.17 and 18.15, respectively. However, there were some noticeable differences in their convergence behavior. Initially, the Dueling DQN network exhibited a faster convergence pace compared to the DQN network. However, it took slightly longer for the Dueling DQN network to reach the maximum reward compared to the DQN network. These results were somewhat unexpected, as we initially anticipated that the Dueling DQN network would outperform the regular DQN network. The reasons for this outcome could be attributed to various factors such as the specific characteristics of the problem, the network architecture, or the interaction between the chosen hyperparameters and the learning dynamics. Further analysis and experimentation may be necessary to gain a deeper understanding of why the Dueling DQN network did not demonstrate a clear advantage over the regular DQN network in this particular scenario.

References:

[1] Deep Reinforcement Learning with Double Q-learning - Hado van Hasselt and Arthur Guez and David Silver: https://arxiv.org/pdf/1509.06461.pdf

[2] Dueling Network Architectures for Deep Reinforcement Learning - Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas: https://arxiv.org/abs/1511.06581

[3] Noisy networks for exploration - Meire Fortunato∗ Mohammad Gheshlaghi Azar∗ Bilal Piot ∗ Jacob Menick Matteo Hessel Ian Osband Alex Graves Volodymyr Mnih Remi Munos Demis Hassabis Olivier Pietquin Charles Blundell Shane Legg: https://arxiv.org/pdf/1706.10295.pdf