Adaptive Intelligence

Assignment 2 – Reinforcement Learning

COM3240

Question 1:

Before we define the difference between the SARSA and the Q-Learning algorithms we need to outline some concepts; The idea behind reinforcement learning is simple. An agent performs an action from a state, its environment then returns a “reward”. The agent uses this reward to evaluate the action. The agent will decide which action to perform based on a policy. A Q-value function returns a long-term expected return from an action in a state. The policy we use here is an e-greedy policy, which means the algorithm either chooses an action at random, or chooses it greedily (chooses the action with the greatest expected return). Algorithms can be on-policy which means that the way it chooses its action and how it updates its Q-values are based on the same policy. However an off-policy algorithm doesn’t have to follow the policy when updated Q-values. This distinction puts the different between the algorithms in contrast. The Q-Learning algorithm is off-policy which means it *sometimes* uses an e-greedy policy. Whereas the SARSA is on-policy, meaning it always uses e-greedy.

Question 2:

Figure 1 shows the average moves per game and the average reward per game. As the data is quite noisy a moving average was used to smooth the graphs. This was done using a sliding window of 400 games. As we can see the rewards per game has a sharp spike around the 20 000th game indicating the convergence of the weight vectors to an optimal solution. The algorithm continues to learn throughout the loop. The average number of moves per game steadily increases.

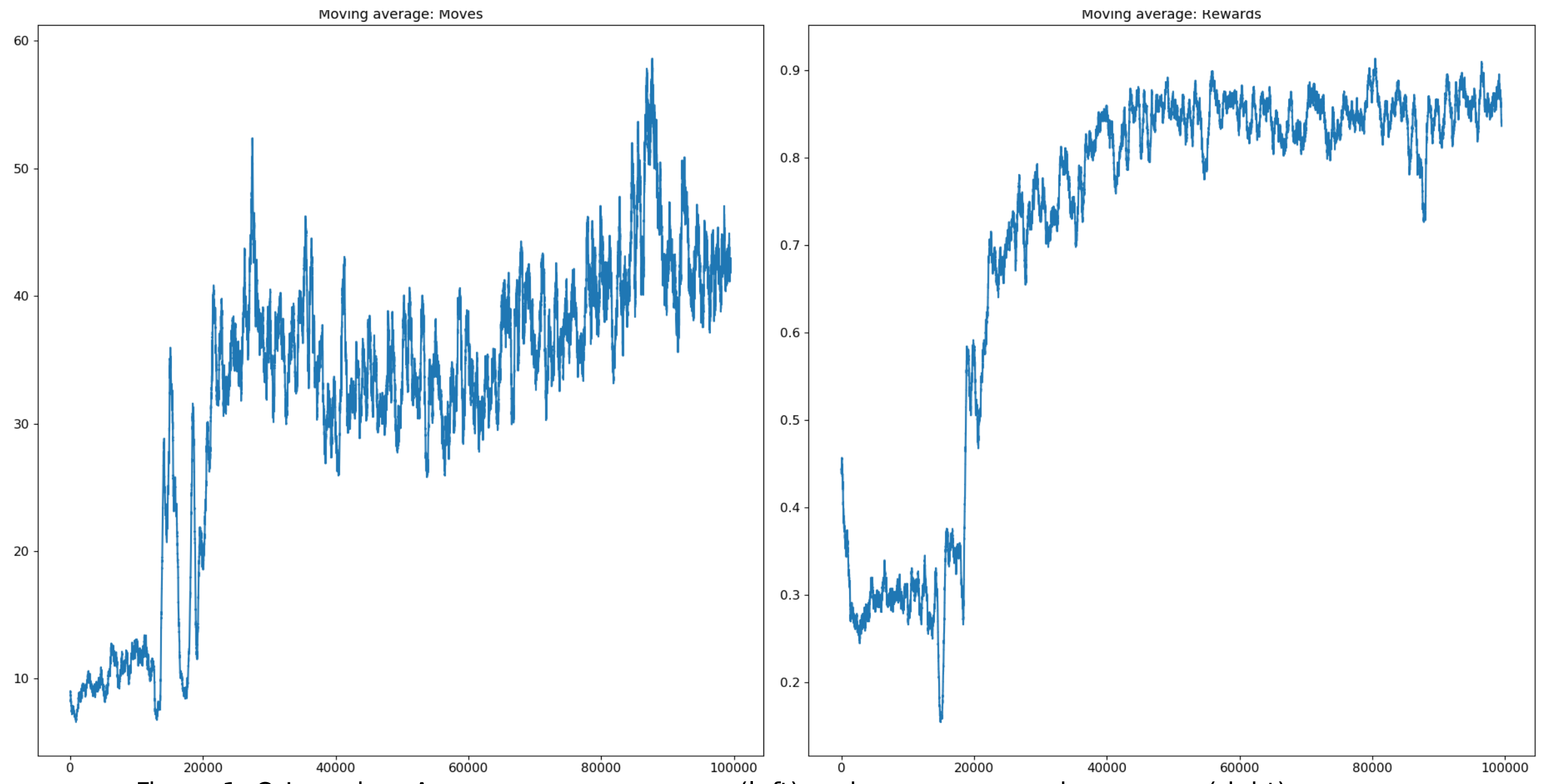

Figure 1: Q-Learning. Average move per game (left) and average reward per game (right).

Question 3:

The function of the discount factor (gamma) is to optimise the algorithm for long term or short term rewards. A high value will make the algorithm use long-term strategies whereas a smaller value makes it favour short-term strategies. Figure 2 shows the performance of the algorithm using gamma = 0.45 (0.5 lower than used for figure 1). As we can see in Figure 2 the number of moves per game is much lower than in the last run. The last run has a mean of about 45 in the last few thousand games whereas the mean is closer to 35. This suggests that the algorithm is using short term strategies. On the other hand its reward per game is also lower, so it does not lock onto to an optimal strategy quickly.

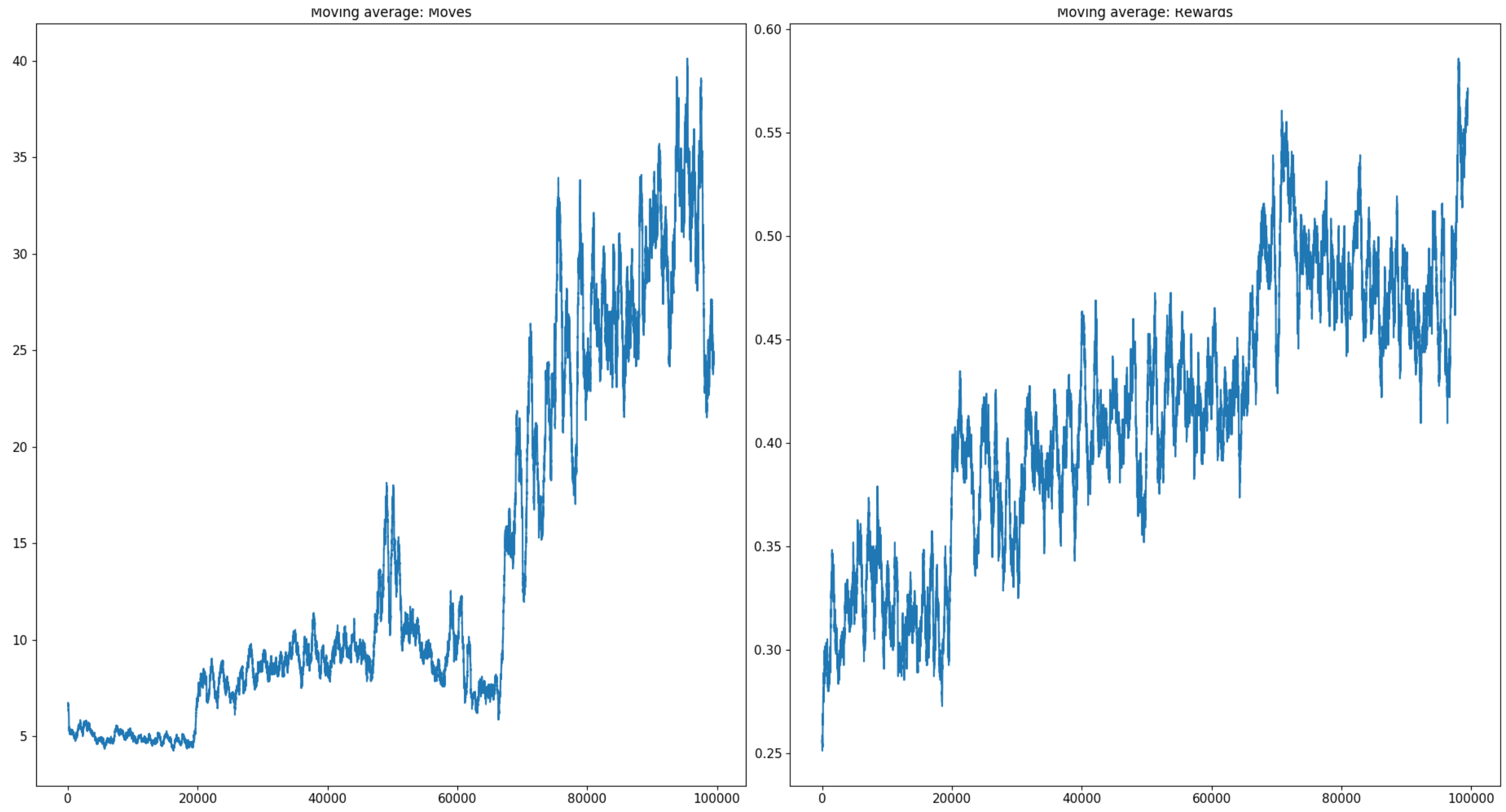

Figure 2: Q-Learning. Gamma 0.45. Average move per game (left) and average reward per game (right).

The epsilon value in our algorithm is used to set a threshold between choosing greedily and exploring. Essentially it introduces some probability that the algorithm chooses a random course of action, this helps ensure that it does not lock onto a sub-optimal solution. The higher the value of epsilon the more stochasticity we introduce into the algorithm. In our algorithm we have implemented a reduction per turn for the epsilon value. This should bring the algorithm towards a more greedy behaviour. In Figure 3 we can see the algorithm run with higher reduction value (by a factor of 10). The reward per turn does not converge as quickly because the algorithm is encouraged to explore more often. This results in lower rewards per game as in the strategy used for figure 1. On the other hand we can be more confident that the algorithm converges to the best solution in time. The number of moves per game seems to have an anomaly but trends around 40-50 moves.

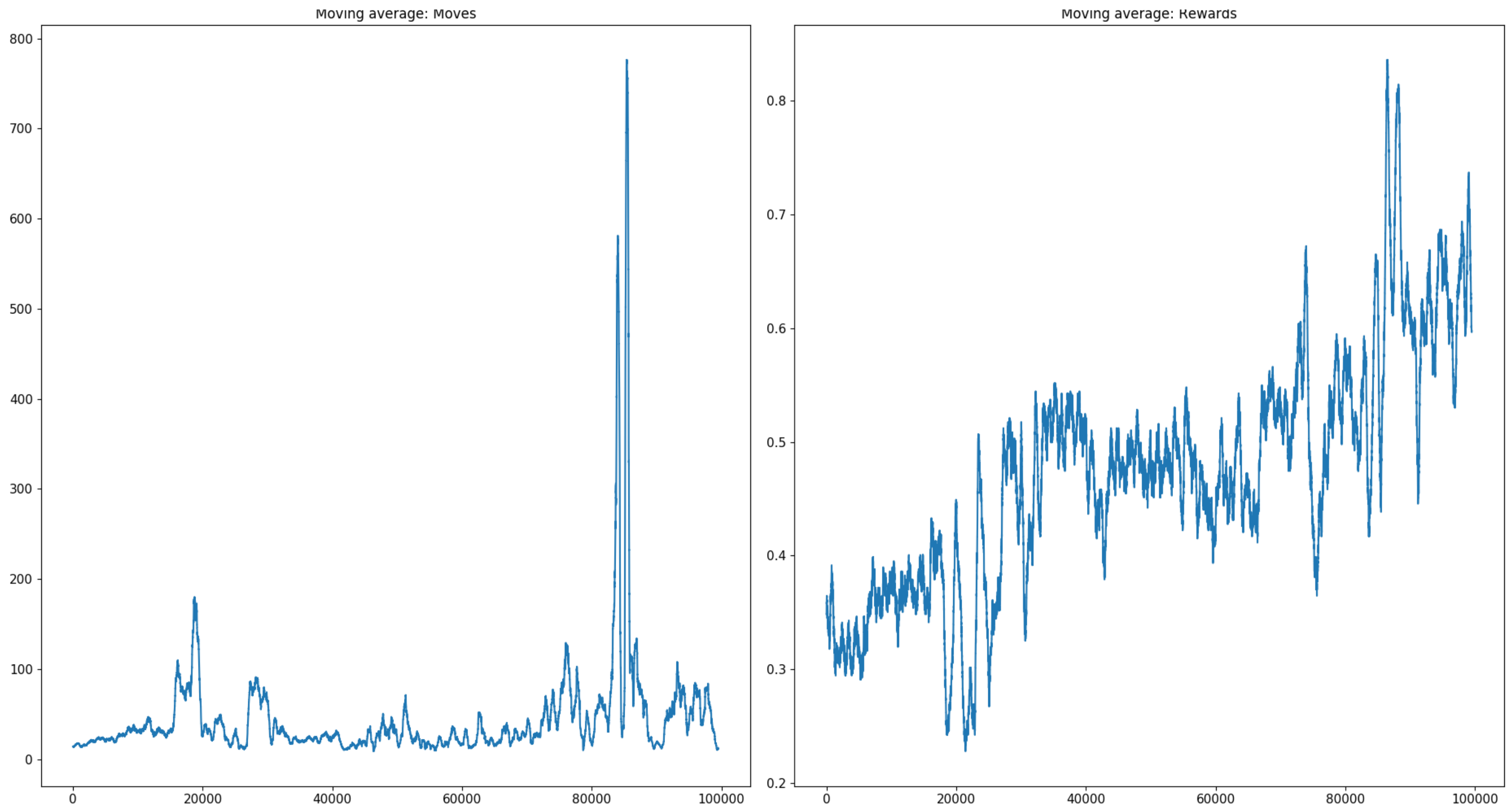

Figure 3: Q-Learning. Beta 0.0005. Average move per game (left) and average reward per game (right).

Question 4:

As explained above the SARSA version of the algorithm is on-policy. That means that it updates its Q values based on the same strategy it chooses its action with. So as opposed to the Q-learning algorithm it doesn’t always update its weights based on the greedy policy it also updates them based on random moves. Figure 4 shows the implementation of the SARSA algorithm plotted with Q learning. It is immediately obvious that the algorithm takes longer to learn, this suggests that it is exploring before converging to an optimal solution. In figure 5 we can see the Q-learning algorithm plot with the SARSA.

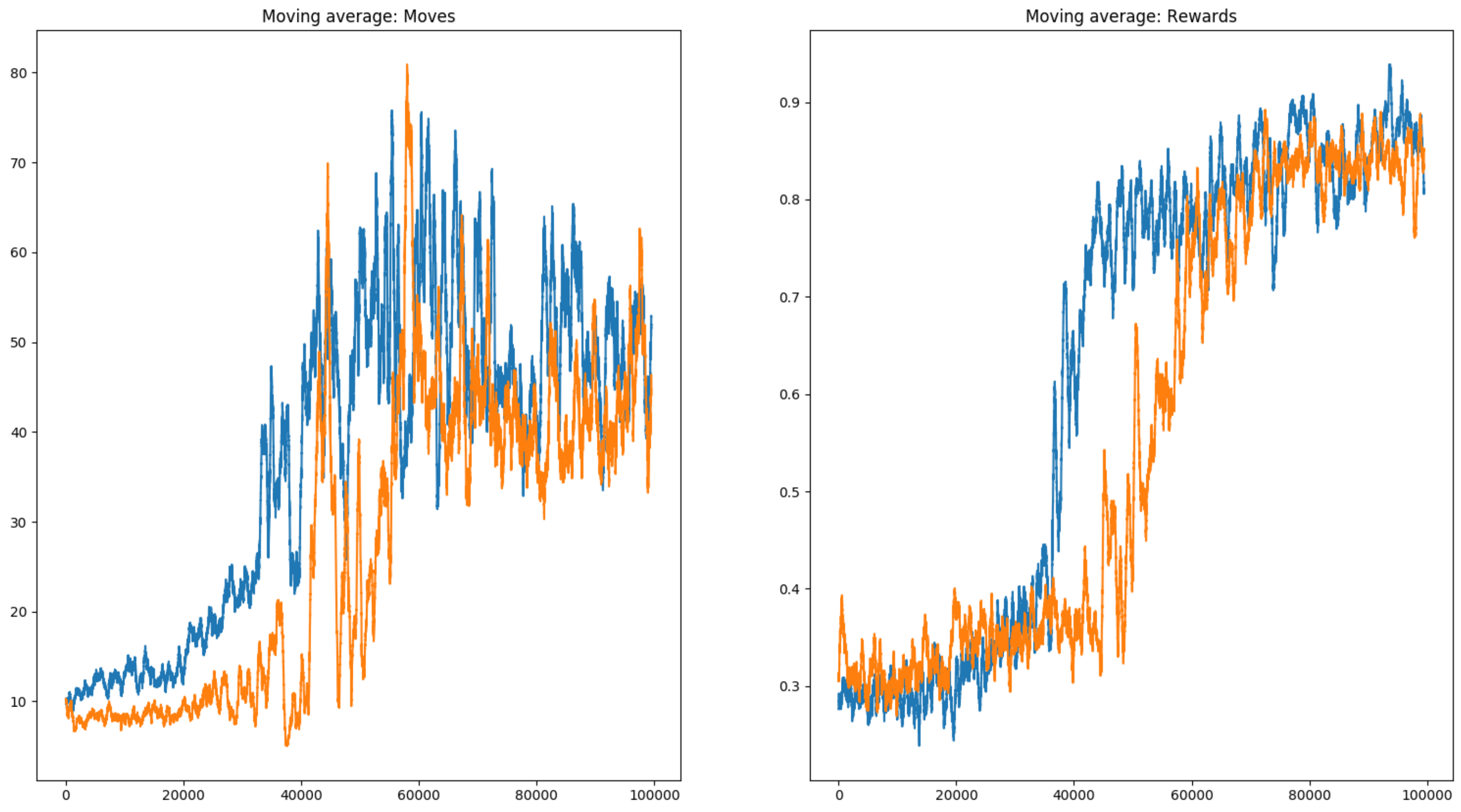

Figure 4: SARSA plotted with Q-learning.

Note: All of these runs are implemented using suggested parameters. Except from changes mentioned in the captions.