CS533 Intelligent Agents and Decision Making - Homework 3

Distributed Temporal Difference Learning

Austin Lally - November 12, 2021

1 Introduction

This assignment implements both Q-Learning and SARSA to train agents to navigate the FrozenLake environment. It examines the differences between the two algorithms and explores the effects of tuning the learning rate and exploration rate hyperparameters. It then implements a distributed version of the Q-Learning algorithm and compares its runtime and output against the corresponding single-core implementation.

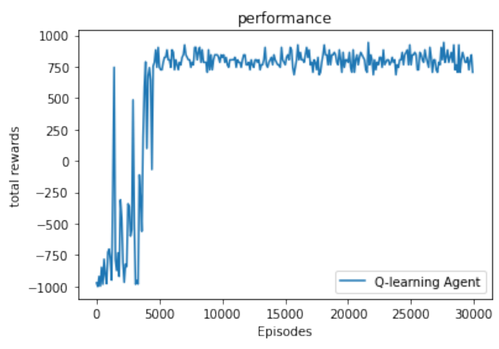
2 Non-Distributed RL Agents

1. *Provide the learning curves for the above experiments. Clearly label the curves by the parameters used.*

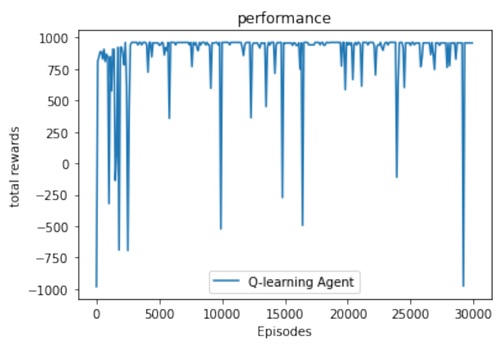
For all experiments, `learning_episodes` and `test_interval` were set to the recommended default values.

The first set of figures compares learning rates $\alpha = 0.001$ and $\alpha = 0.1$ using the default value of $\epsilon = 0.1$. The learning curves for the Dangerous Hallway map are shown in figure 1 and the curves for the 16x16 map are shown in figure 2.

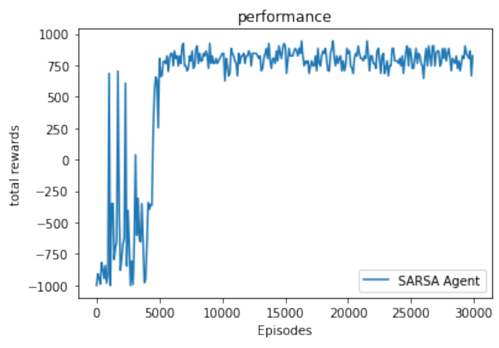
The next set of figures compares exploration rates $\epsilon = 0.3$ and $\epsilon = 0.05$ using the best observed learning rate $\alpha = 0.001$. The learning curves for the Dangerous Hallway map are shown in figure 3 and the curves for the 16x16 map are shown in figure 4.



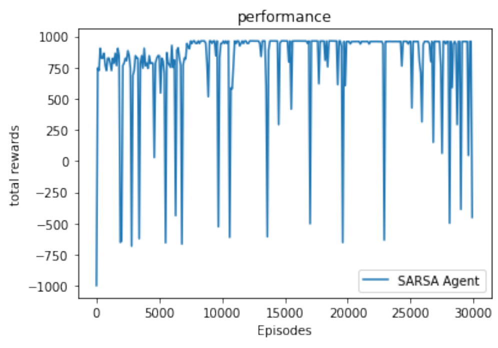
(a) Q-Learning, $\alpha = 0.001$



(b) Q-Learning, $\alpha = 0.1$

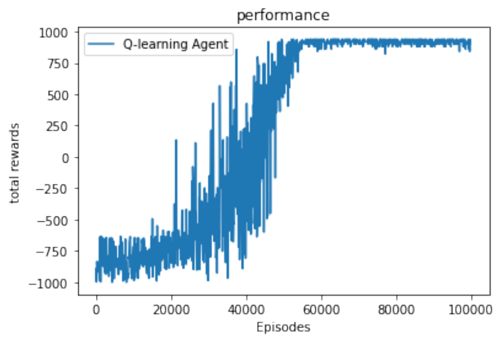


(c) SARSA, $\alpha = 0.001$

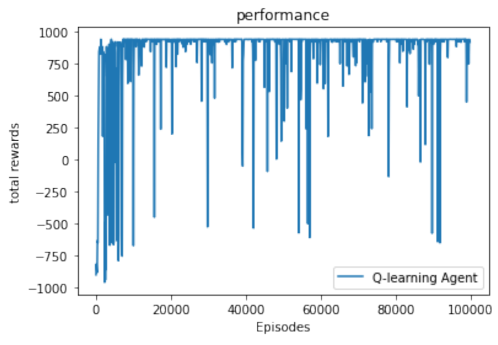


(d) SARSA, $\alpha = 0.1$

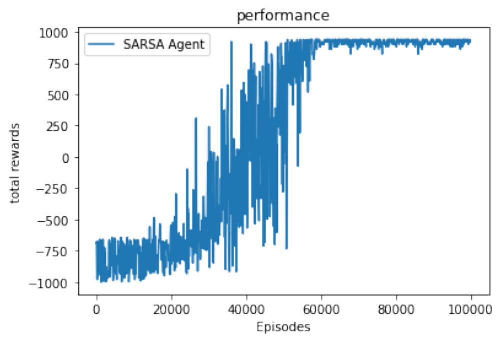
Figure 1: Performance on the Dangerous Hallway map with $\epsilon = 0.1$ and different learning rates α.



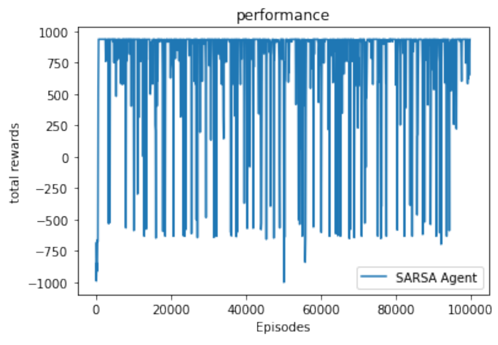
(a) Q-Learning, $\alpha = 0.001$



(b) Q-Learning, $\alpha = 0.1$

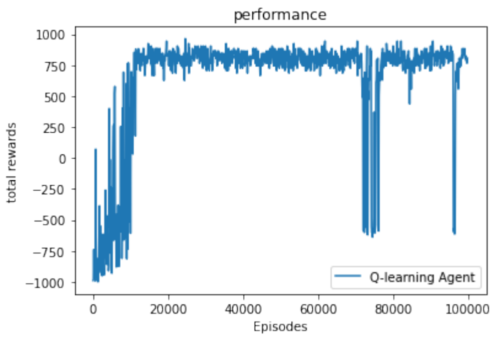


(c) SARSA, $\alpha = 0.001$

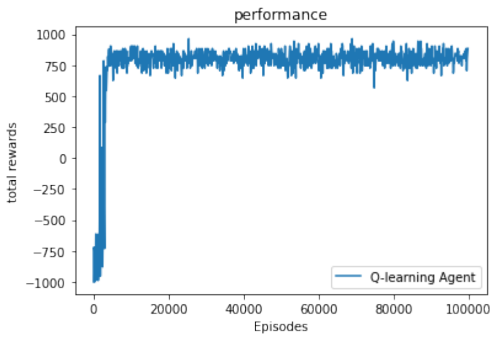


(d) SARSA, $\alpha = 0.1$

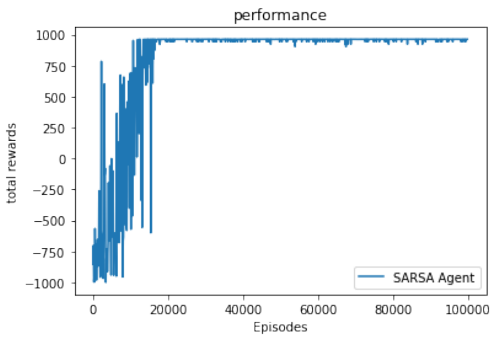
Figure 2: Performance on the 16x16 map with $\epsilon = 0.1$ and different learning rates α.



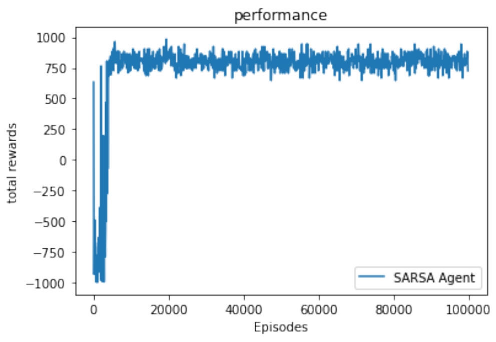
(a) Q-Learning, $\epsilon = 0.3$



(b) Q-Learning, $\epsilon = 0.05$

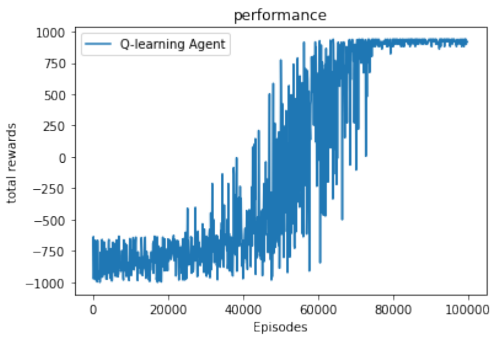


(c) SARSA, $\epsilon = 0.3$

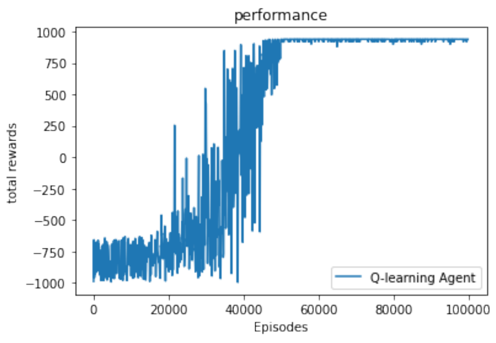


(d) SARSA, $\epsilon = 0.05$

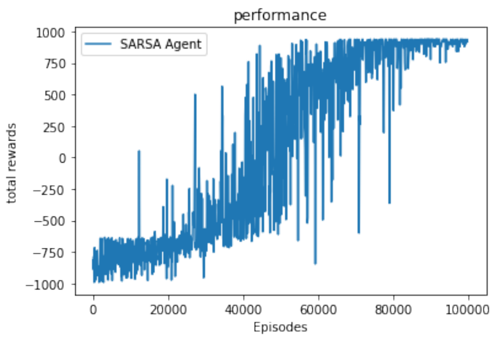
Figure 3: Performance on the Dangerous Hallway map with $\alpha = 0.001$ and different exploration rates ϵ.



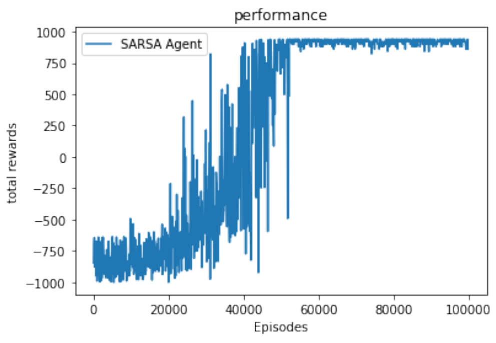
(a) Q-Learning, $\epsilon = 0.3$



(b) Q-Learning, $\epsilon = 0.05$



(c) SARSA, $\epsilon = 0.3$



(d) SARSA, $\epsilon = 0.05$

Figure 4: Performance on the 16x16 map with $\alpha = 0.001$ and different exploration rates ϵ.

2. *Did you observe differences for SARSA when using the two different learning rates? If there were significant differences, what were they and how can you explain them?*

For both maps, using a small learning rate of $\alpha = 0.001$ led to SARSA's total rewards generally increasing over time from close to the minimum of -1000 to something consistently high (around 800 for the Dangerous Hallway map, and around 900 for the 16x16 map). On the other hand, using a larger learning rate of $\alpha = 0.1$ led to a very spiky learning curve characterized by frequent high reward and occasionally very low reward, without any obvious change over time. This behavior can be attributed to overcompensation: the immediate result of an individual decision has too large an effect on the model's understanding of the value of that decision. When this happens, the policy changes too much at each iteration and fails to converge.

3. *Repeat (2) for Q-Learning.*

The Q-Learning algorithm behaved much the same as SARSA for different learning rates. Its performance increased over time with a small learning rate $\alpha = 0.001$, but its performance was spiky with the larger rate $\alpha = 0.1$. The only visible difference was that the Q-Learning approach may be slightly more robust to the larger learning rate. The frequency of negative value spikes decreased somewhat after the first few thousand episodes, and the values not coinciding with negative spikes did seem to increase a little during the same span. It fails to converge, though, for the same reasons as SARSA: the large learning rate leads to oscillations that "jump past" the correct Q values.

4. *Did you observe differences for SARSA when using different values of ϵ? If there were significant differences, what were they and how do you explain them?*

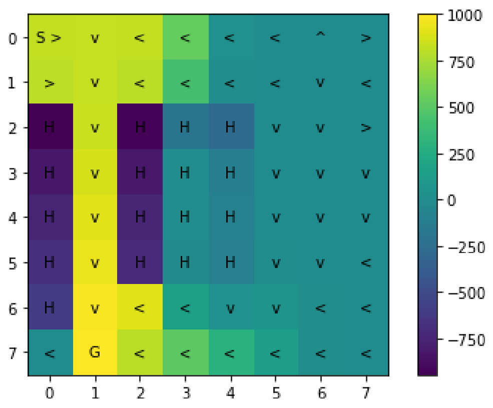
With a lower exploration rate of $\epsilon = 0.05$, SARSA converged more quickly (about 5000 episodes vs. 20,000 on the Dangerous Hallway map, and about 55,000 episodes vs. at least 90,000 on the 16x16 map). However, it converged to a noticeably higher and more consistent total reward when using the higher exploration rate of $\epsilon = 0.3$ on the Dangerous Hallway map. Slower convergence with a higher exploration rate makes sense because there is more randomness overall. The algorithm will spend more time exploring a wider variety of paths and less time sticking to the greedy path. The fact that SARSA converges to a higher overall reward value with the higher exploration rate is due to the Dangerous Hallway phenomenon (explained further in question (6) below). With $\epsilon = 0.3$, it learns to avoid the hallway. This costs an additional -10 reward for the extra travel distance, but avoids the $-2000 * (1/2) * (1 - 0.95^4) = -185$ reward associated with a 5% chance of acting randomly in the hallway and possibly stepping in a hole.

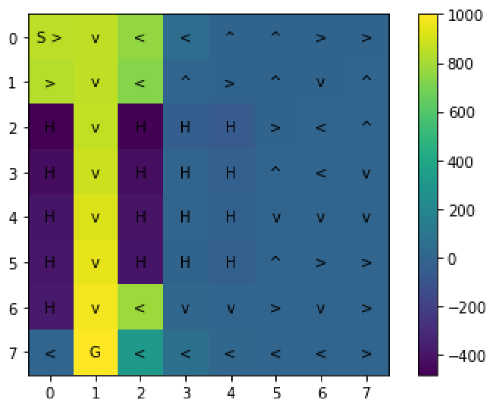
5. *Repeat (4) for Q-Learning.*

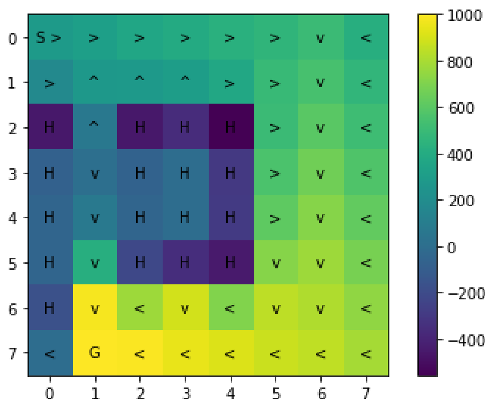
The Q-Learning algorithm exhibits similar behavior to SARSA in that it appears to converge more quickly with the lower exploration rate (about 4000 vs. 10,000 episodes on the Dangerous Hallway map, and about 50,000 vs. 75,000 on the 16x16 map). The difference here is that Q-Learning does not converge to a higher value with a higher exploration rate. Instead, the converged total reward is about the same between the two exploration rates. Also, the higher exploration rate led to some low-reward spikes around the 75,000- and 95,000- episode marks on the Dangerous Hallway map. The slower convergence rate for higher exploration rates is explained in the same way as for SARSA above: more random behavior spends more time exploring areas that turn out to be suboptimal. In this case the ultimate converged total reward value is the same between the two ϵ values because the learned policy is the same.

6. *For the map "Dangerous Hallway" did you observe differences in the policies learned by SARSA and Q-Learning for the two values of epsilon (there should be differences between Q-learning and SARSA for at least one value)? If you observed a difference, give your best explanation for why Q-learning and SARSA found different solutions.*

Figure 5 shows the policies learned by the two algorithms for exploration rates $\epsilon \in \{0.3, 0.05\}$. With a high exploration rate $\epsilon = 0.3$, SARSA learned to avoid the dangerous hallway. This makes sense because SARSA learns from the *behavior policy* while Q-Learning learns from the *greedy policy*. Because of the high exploration rate, when SARSA enters the dangerous hallway there is a high probability that it will take an exploratory step into a hole. This reduces the value of the hallway states. On the other hand, when Q-Learning steps into the hallway, there is a relatively low probability that it will find a hole. Since it will never randomly explore a step toward a hole, the only probability of hitting one comes from the comparatively low chance of slipping on the ice.


(a) Q-Learning, $\epsilon = 0.3$


(b) Q-Learning, $\epsilon = 0.05$


(c) SARSA, $\epsilon = 0.3$

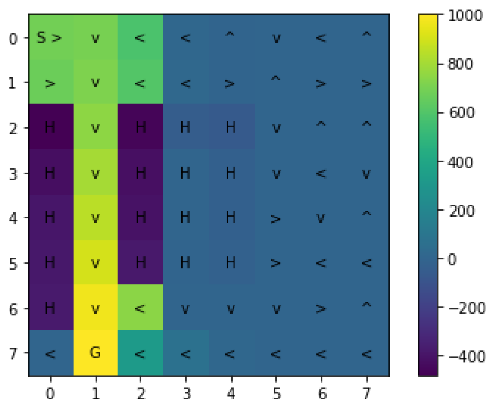

(d) SARSA, $\epsilon = 0.05$

Figure 5: Policies learned on the Dangerous Hallway map with $\alpha = 0.001$ and different exploration rates ϵ.

To clarify, assume the agent is at state $(1, 1)$ above the top of the hallway and performs an update step. Assume the current policy is to go down from that state, and assume that the agent doesn't slip on the ice. For both algorithms, the update is the same except for one term which we call $\mathbf{Q}^{next}$:

$$\mathbf{Q}[(1,1),\downarrow] \longleftarrow \mathbf{Q}[(1,1),\downarrow] + \alpha(-1 + \beta(\mathbf{Q}^{next} - \mathbf{Q}[(1,1),\downarrow])$$

For the Q-Learning algorithm, $\mathbf{Q}^{next}$ is defined as the value of following the greedy policy:

$$\mathbf{Q}^{next} = \mathbf{Q}[(2,1), \downarrow].$$

For SARSA, though, $\mathbf{Q}^{next}$ is defined as the *actual* value of the next action taken, which might turn out to be exploratory. With $\epsilon = 0.3$,

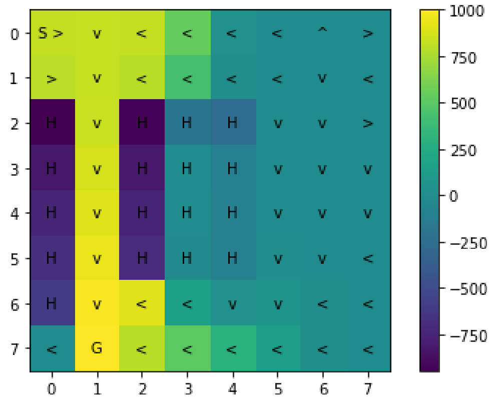
$$\begin{aligned}\mathbf{Q}^{next} = 0.7 \cdot \mathbf{Q}[(2,1), \downarrow] + 0.3\Big(&0.25 \cdot \mathbf{Q}[(2,1), \uparrow] \\ &+ 0.25 \cdot \mathbf{Q}[(2,1), \leftarrow] \\ &+ 0.25 \cdot \mathbf{Q}[(2,1), \downarrow] \\ &+ 0.25 \cdot \mathbf{Q}[(2,1), \rightarrow]\Big).\end{aligned}$$

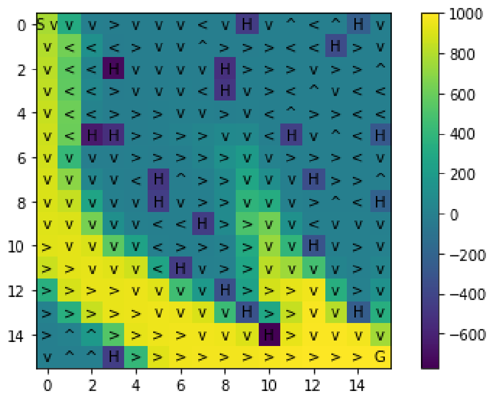
Because $\mathbf{Q}[(2,1), \leftarrow]$ and $\mathbf{Q}[(2,1), \rightarrow]$ usually lead to holes, their values are very low and they pull down the overall value of $\mathbf{Q}^{next}$ for the SARSA algorithm. Therefore, SARSA will assign a lower updated value to $\mathbf{Q}[(1,1), \downarrow]$ and will tend to go away from the hallway in that state instead.

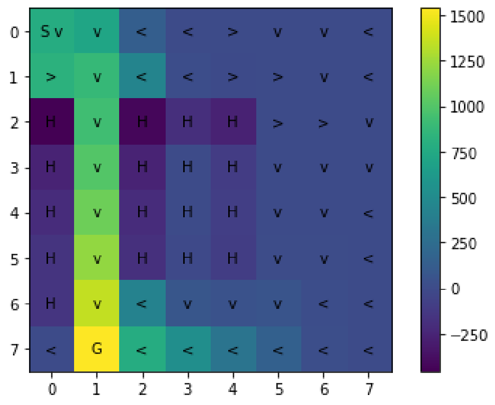
3 Distributed Q-learning Agent

7. *Show the value functions learned by the distributed methods for the best policies learned with ϵ equal to 0.3 and compare to those of the single-core method. Run the algorithm for the recommended number of episodes for each of the maps. Did the approaches produce similar results?*

Figure 6 compares the policies learned by the distributed Q-Learning approach for each map against the corresponding single-core policies. The results are very similar, up to some scaling factor. The policies are nearly identical, differing in just a few individual states, with the same overall strategy and highest-value path. The main difference is that the distributed implementations assigned higher values overall to all states (notice the difference in the color scales).


(a) Single-core, Dangerous Hallway map


(b) Single-core, 16x16 map


(c) Distributed, Dangerous Hallway map

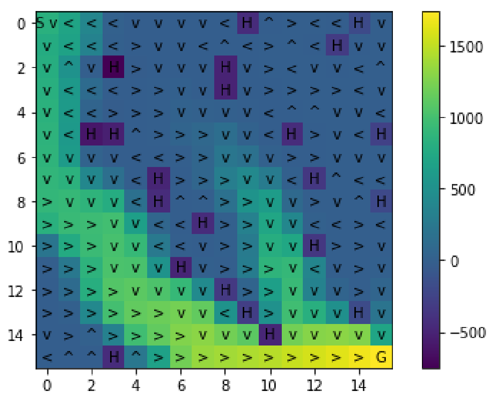
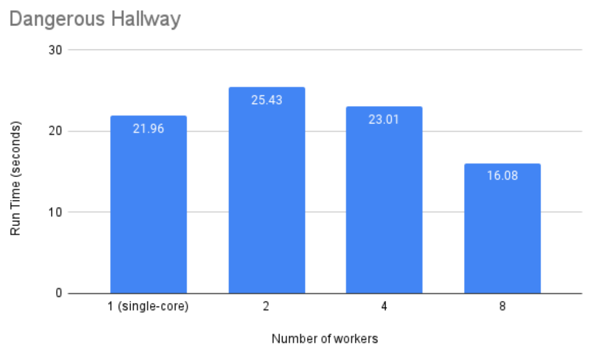

(d) Distributed, 16x16 map

Figure 6: Comparison of policies learned by distributed and non-distributed Q-Learning approaches. All experiments used $\epsilon = 0.3$ and the recommended number of episodes for each map. The distributed approaches used 8 collection workers. Note that the color scale differs between the single-core and distributed approaches.

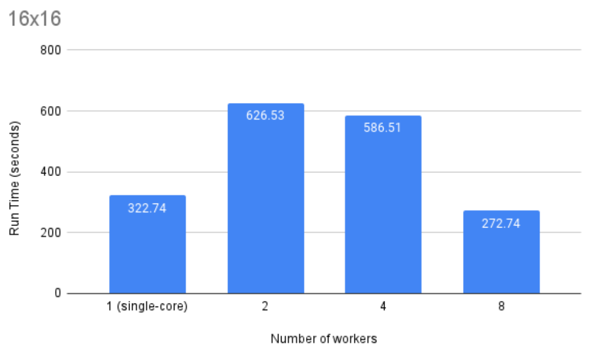
8. *Provide and compare the timing results for the single-core and distributed experiments, including the time to do the evaluations during learning. Describe the trends you observe as the number of workers increases.*

Figure 7 compares the running time of single-core Q-Learning and distributed Q-Learning with various numbers of workers for each of the two maps. For this experiment, all hyperparameters were set to their default values (notably, `test_interval` was set to 100 and `do_test` was `True`).

Among distributed methods, run time decreased as a function of worker count as expected. However, the single-core method remained faster than the distributed method when only two or four workers were available. The additional overhead of inter-process communication in these cases outweighed the benefit of parallelized computation. The other noteworthy point is that the speedup gained by going up to eight workers was more pronounced for the 16x16 map. The larger state space benefits more from increased parallel throughput.



(a) Dangerous Hallway map

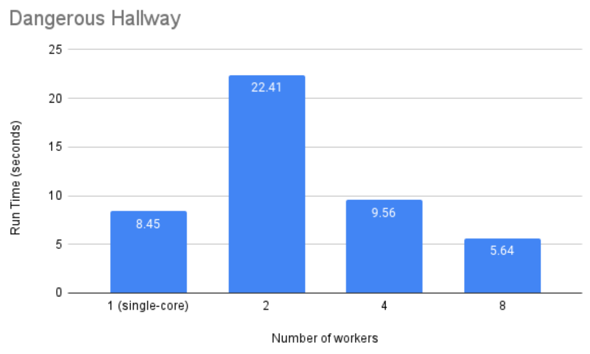


(b) 16x16 map

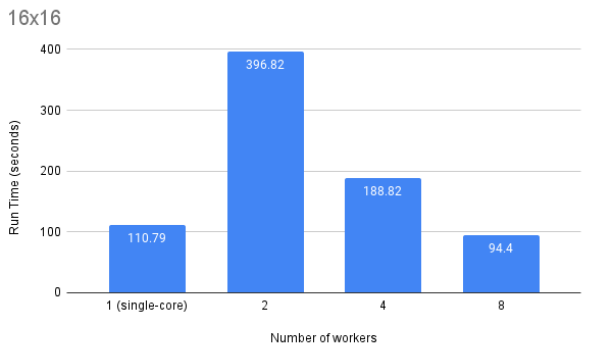
Figure 7: Comparison of training time (with interleaved evaluation) for Q-Learning method with increasing number of workers.

9. *Provide and compare the timing results for the single-core and distributed experiments with the evaluation procedure turned off. That is, here you only want to provide timing results for the learning process without any interleaved evaluation. Compare the results with (8).*

Figure 8 compares the running time of single-core Q-Learning and distributed Q-Learning with various numbers of workers for each of the two maps, but with NO interleaved evaluation. For this experiment, all hyperparameters were set to their default values except that `do_test` was set to `False`.



(a) Dangerous Hallway map



(b) 16x16 map

Figure 8: Comparison of training time (with NO interleaved evaluation) for Q-Learning method with increasing number of workers.

The results are very similar to those in (8) above. The single-core approach still wins out over the distributed approach with two or four workers, and the distributed approach pulls ahead when given eight workers. This time, the largest speedup comes between four and eight workers.

When comparing to the time required to run with interleaved evaluation (fig. 7), this approach is much faster. When looking at the fastest configurations (single-core and 8-worker distributed), omitting the periodic evaluation reduced the running time by about two thirds.