Programming Assignment 3

CS6700

Dhruv Laad, AE16B011

29 April 2020

1 Hierarchical Reinforcement Learning

The environment is implemented in OpenAI Gym as `room-world`. Options are implemented as a separate class, `HallwayOption` in `options.py`. Every experiment is run with an ϵ-greedy policy, where $\epsilon = 0.1$. The step size parameter for all learning steps, $\alpha = 0.125$ for all experiments. `HallwayOption` implements a `run()` method that completes execution of the multi-step option till termination. All results are from the average of 100 runs performed. Further, for all experiments, convergence was observed well before 1000 episodes, and so each run consists of 1000 episodes.

The hallway states are of two kinds, $\mathcal{H}_0$ that have walls on the left and right and $\mathcal{H}_1$ that have walls on the top and bottom. For the option whose goal state is $\mathcal{H}_0$, the policy is crafted such that it aligns the agent **first vertically** with the goal hallway state, and then moves the agent vertically toward the goal hallway state. Similarly for hallway states $\mathcal{H}_1$.

1.1 SMDP Q-learning

SMDP Q-learning is performed for both goals $\mathcal{G}_1$ and $\mathcal{G}_2$ and the results are shown in Figs. 1 - 6. From Fig. 1, with both goals, SMDP Q-learning evidently arrives at $\mathcal{Q}^*$ quite rapidly (in ≈ 100 episodes). Reaching goal $\mathcal{G}_1$ takes fewer number of steps on average. Fig. 2 implies that, when options are chosen greedily from $\mathcal{Q}^*$, multi-step options are the optimal choice in (almost) every state.

Finally, in Fig. 6, looking at the $\max\limits_{o\in\mathcal{O}_s}\mathcal{Q}(s,o)$ values for every state, the hallway states have a locally higher value. This could be because as the learning progresses, options are chosen more frequently (as Fig. 2 suggests) and since these terminate (and consequently are initiated) at the hallway states, it is the hallway states whose value is learnt more accurately. This conclusion is further strengthened by the observation that non-hallway states in the vicinity of the goal have higher values than non-hallway states in farther away. This is because here, multi-step options initiated in non-hallway states also have a good probability of terminating at the goal due to the manner in which the multi-step option policies are crafted.

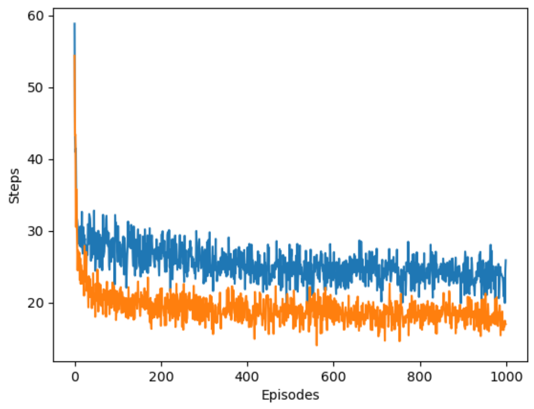
1.2 Fixed initial state

When the initial state is fixed, exploration is vastly reduced as is evident from Fig. 5. Most of the states in the room farthest away from Room 4 still have primitive actions as the optimal option. This conclusion is further strengthened by observing from Fig. 6 that the hallway state connecting room 4 to room 3 has the largest value. In fact, almost all the non-hallway states room other than room 4 have value 0. The multi-step options in Room 4 are clearly used the most. Lastly, as is expected, the number of steps taken to reach the goal is lesser than the case where the initial state is randomly chosen, observed from Fig. 4.

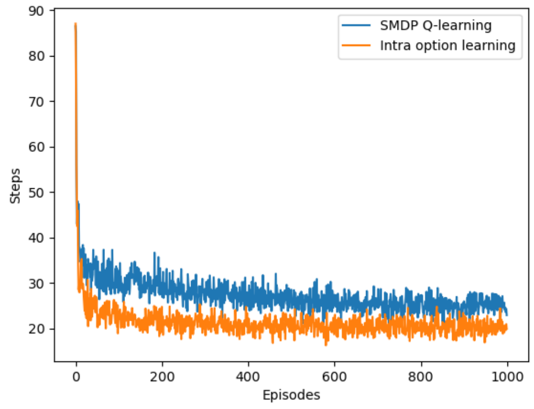
1.3 Intra option learning

From Fig. 1, intra-option learning performs better in terms of steps taken to reach the goal. It thus gives us a more optimal $\mathcal{Q}^*$ than regular SMDP Q-learning. The exploration is the same as SMDP Q-learning, and we see the same trend in terms of preference of multi-step options over primitive actions at every state.

Interestingly, observing Fig. 8b, we see that only hallway states have $\mathcal{Q}(s, o) > 0$. Meaning that the agent learns to travel from hallway state to hallway state right from initialisation. Intra-option learning requires fewer steps in both cases, and even seems to converge faster than regular SMDP Q-learning. This is because of the advantage that intra-option learning methods have, where we can learn about the executing option (and other *consistent* options) faster than in SMDP Q-learning, as we do not have to wait for the executing option to terminate.

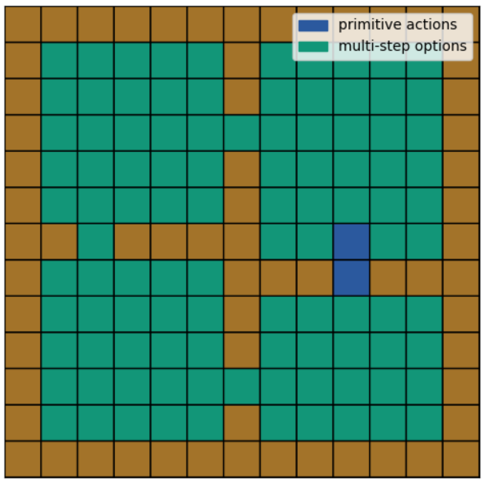


(a) $\mathcal{G}_1$

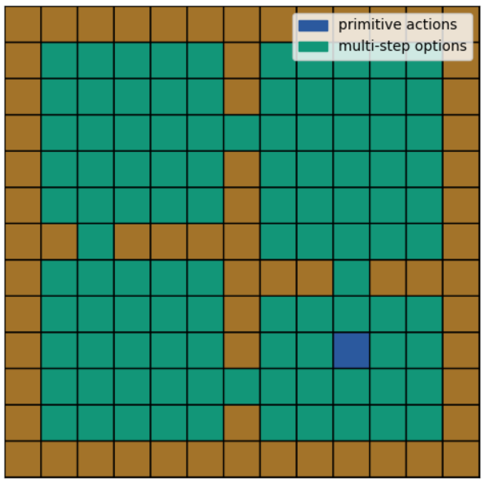


(b) $\mathcal{G}_2$

Figure 1: Learning curves for steps taken till goal is reached for both SMDP Q-learning and intra option learning.

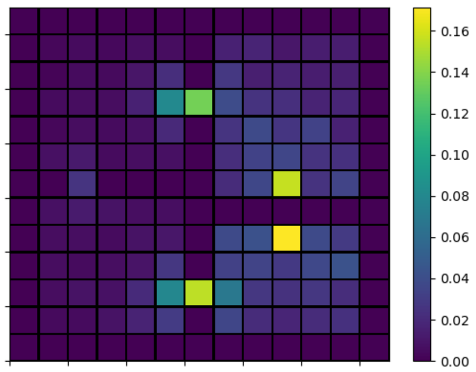


(a) $\mathcal{G}_1$

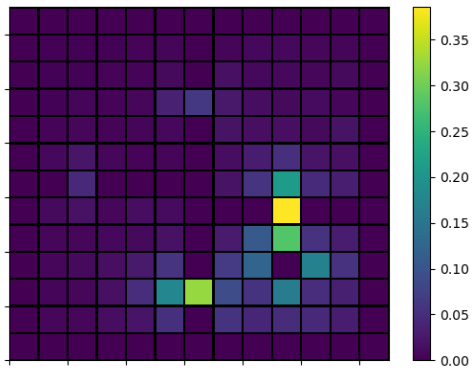


(b) $\mathcal{G}_2$

Figure 2: Learnt state-wise preference of options (multi-step options and primitive actions) for SMDP Q-learning. For each state, the preference between the two kinds of options is shown.

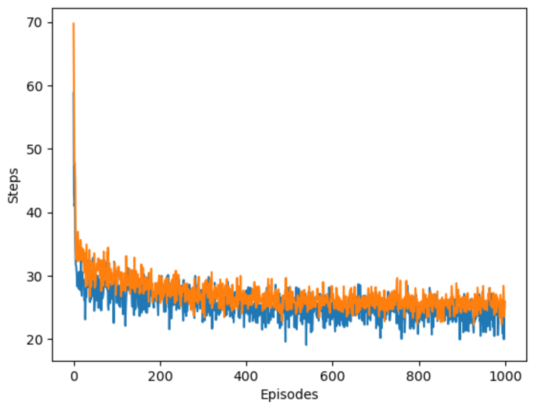


(a) $\mathcal{G}_1$

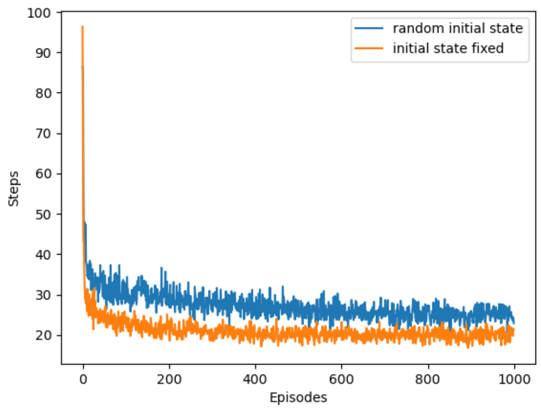


(b) $\mathcal{G}_2$

Figure 3: Values, learnt by SMDP Q-learning, of $\max_{o \in \mathcal{O}_s} \mathcal{Q}(s, o)$ for each state s in the room world.

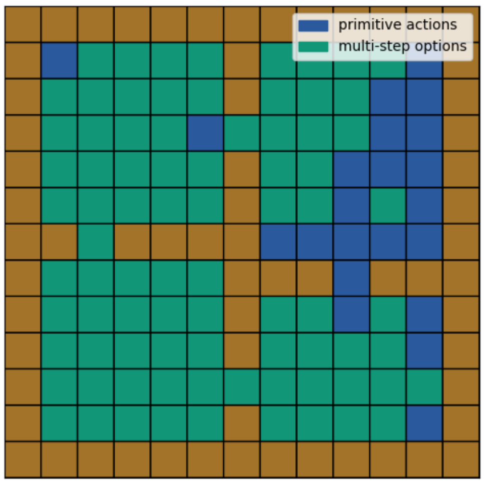


(a) $\mathcal{G}_1$

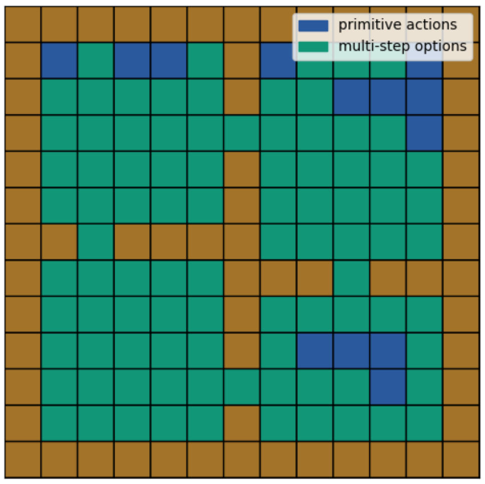


(b) $\mathcal{G}_2$

Figure 4: Learning curves for steps taken till goal is reached for SMDP Q-learning with random and fixed state initialisation.

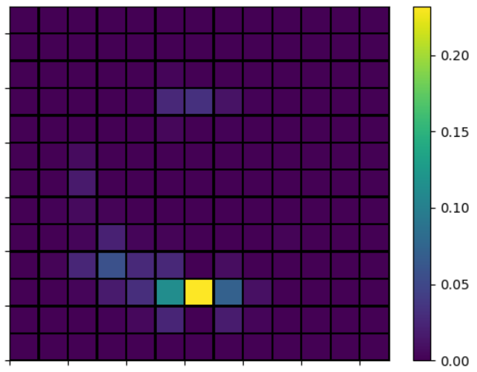


(a) $\mathcal{G}_1$

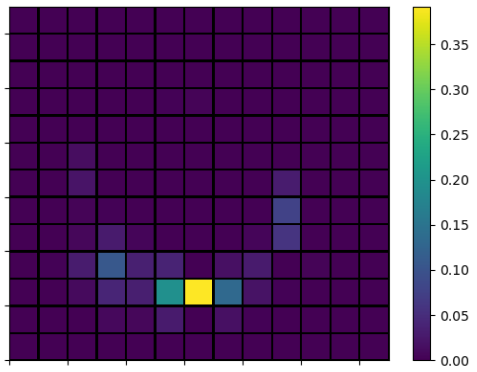


(b) $\mathcal{G}_2$

Figure 5: Learnt state-wise preference of options (multi-step options and primitive actions) for SMDP Q-learning with random and fixed state initialisation.

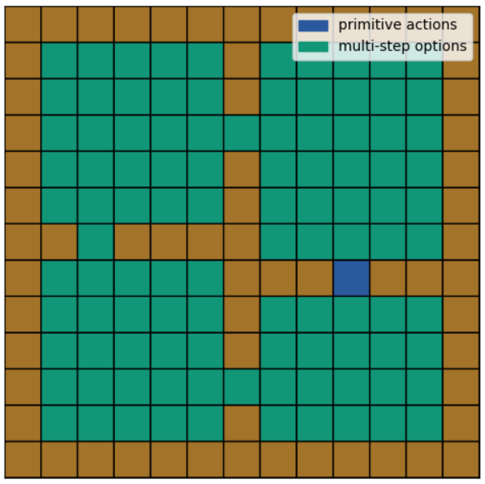


(a) $\mathcal{G}_1$

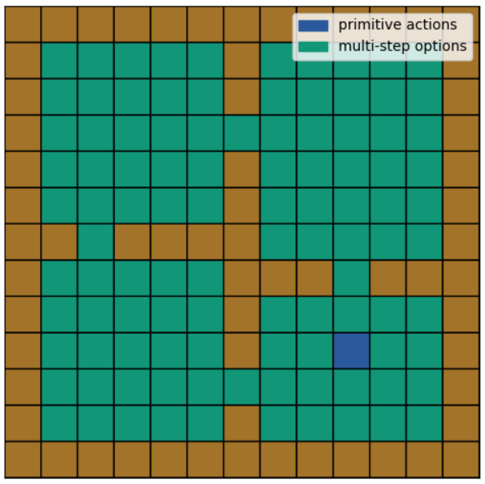


(b) $\mathcal{G}_2$

Figure 6: Values, learnt by SMDP Q-learning, of $\max_{o \in \mathcal{O}_s} \mathcal{Q}(s, o)$ for each state s with random and fixed state initialisation

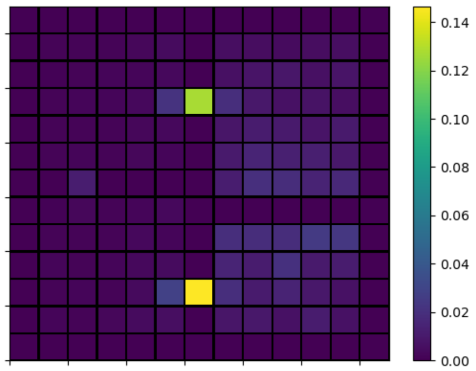


(a) $\mathcal{G}_1$

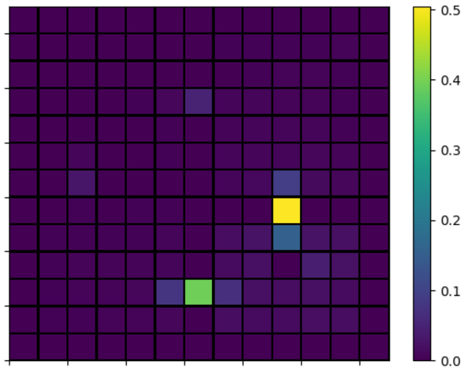


(b) $\mathcal{G}_2$

Figure 7: Learnt state-wise preference of options (multi-step options and primitive actions) for one step intra-option Q-learning.



(a) $\mathcal{G}_1$



(b) $\mathcal{G}_2$

Figure 8: Values, learnt by one step intra-option Q-learning, of $\max_{o \in \mathcal{O}_s} \mathcal{Q}(s, o)$ for each state s.

2 (Not so) Deep RL

All the main code is present in the file `dqn.py`. The code consists of three classes: `QNetwork` (PyTorch model to define the neural network), `ReplayMemory` (to handle operations of the experience replay buffer) and `DQN` (the complete DQN algorithm). Before training begins, the experience replay buffer is filled with experiences (using `DQN.gather_experience()`) so that these steps are not counted in the learning curve.

2.1 Learning curve

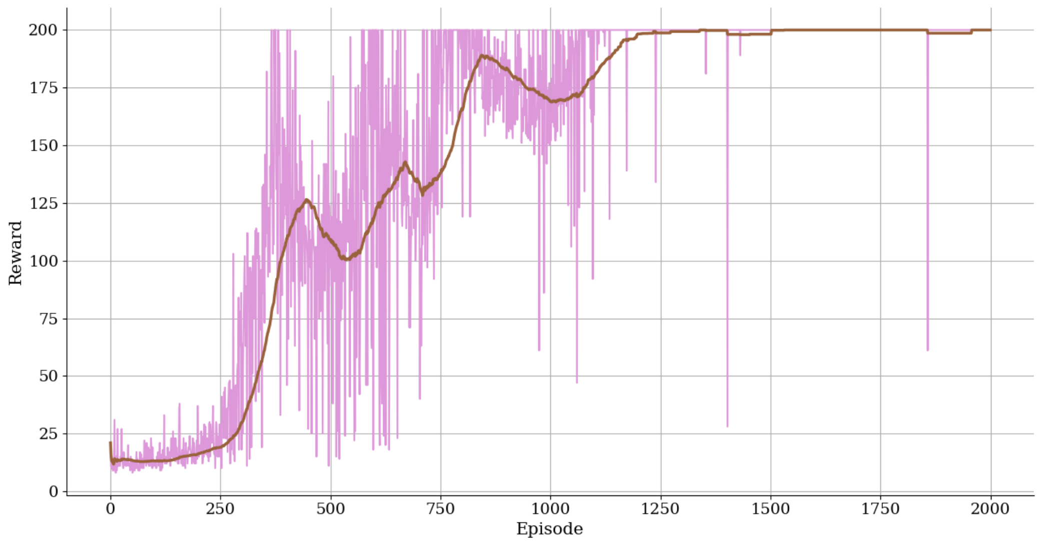
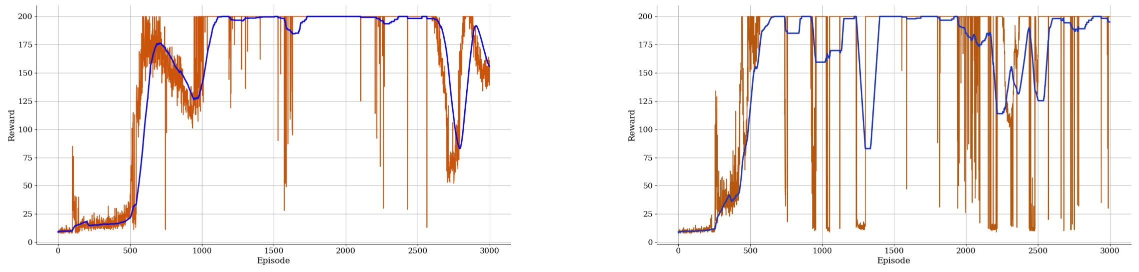


Figure 9: Learning curve for DQN with soft target network update for `CartPole-v0` environment and hyperparameters given in Table 1

2.2 Final set of hyperparameters

Parameter	Description	Value
`REPLAY_MEMORY_SIZE`	Size of the experience replay buffer	10^6
`BATCH_SIZE`	Size of the mini-batch used during Q-learning	1024
`NUM_EPISODES`	Number of episodes during training	2000
`MAX_STEPS`	Maximum number of steps during each episode	200
`EPSILON_MAX`	Maximum rate of exploration in ε-greedy policy	0.5
`EPSILON_MIN`	Minimum rate of exploration in ε-greedy policy	0.01
`EPSILON_DECAY`	Decay rate of exploration in ε-greedy policy	0.9985
`LEARNING_RATE`	Learning rate for the optimizer	10^{-4}
`TAU`	Soft update rate for target network	0.999
`H1_SIZE`	Size of first fully-connected hidden layer in the Q-network	64
`H2_SIZE`	Size of second fully-connected hidden layer in the Q-network	64

Table 1: Final parameters for DQN network training with `CartPole-v0` environment



(a) Catastrophic forgetting in deep Q-learning

(b) Effect of a large learning rate

Figure 10: Effects of specific parameters on the training progress of the DQN algorithm

2.3 Observations

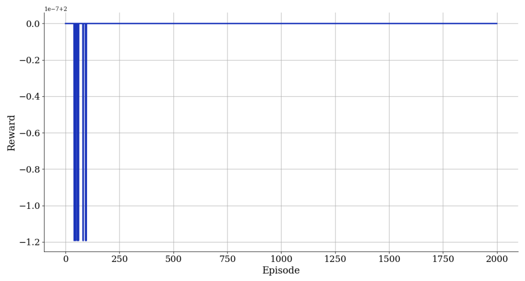
- The most recurring observation is that even after an average reward greater than 195 is attained over an extended period of time, the performance suddenly degrades quite rapidly. This could be possibly due to a small experience replay buffer size e.g. Fig. 10a shows the learning progress for the same set of hyperparameters as listed in Table 1, except `REPLAY_MEMORY_SIZE` is set to 10^4 and `NUM_EPISODES` is set to 3000. After achieving the almost identical progress as in Fig. 9 the performance of the agent drastically declines after ≈ 2500 episodes. Therefore to ensure best results, the experience replay buffer size was increased and training was stopped early.

- When `LEARNING_RATE` is set too high, the characteristic non-monotonic learning behaviour is observed. Fig. 10b shows the unstable learning behaviour of the agent when the learning rate is too high (in this case it was 10^{-2}. The average reward reaches 195 relatively fast (within 700 episodes as compared to over 1200 in Fig. 9) but there is no stability in as learning progresses and the average reward oscillates a lot.

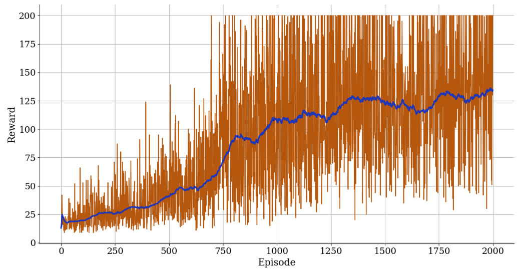
- The original algorithm proposes the use of a "hard" update for the target network. This has a destabilizing effect on the training. A "soft" update law of the form, $\boldsymbol{\theta}^- = \tau\boldsymbol{\theta}^- + (1-\tau)\boldsymbol{\theta}$ solves a lot of the problems faced with certain hyperparameters. Note that the soft update is carried out at every learning step. Further, even with a good hyperparameter setting, the hard update causes rapid degradation in performance. It was observed that if the target update frequency with the hard update rule is set increased, the performance is improved however no stability is observed.

- Too small a mini-batch size causes high variance in the learning process. While this does not always cause instability in the learning, the hard update rule for the target network specified by the original paper causes degradation in the learning process. Using the soft update rule gives significantly improved performance, however learning is slow, with the average reward crossing 195 only after ≈ 1500 episodes, and not converging suitably even after 2000 episodes.

- Initially only a single hidden layer of 128 units was used, but this failed to achieve the target reward. The performance drastically improved when two hidden layers were used instead of one. While this is not surprising, what is interesting is that the even with a small number of units (8 and 4 in the first and second hidden layer respectively), the performance improved significantly.



(a) DQN without experience replay



(b) DQN without target network

Figure 11: Learning curves for when experience replay is turned off and target network is not used in the DQN algorithm

2.4 Importance of experience replay and target network

The results of not using experience replay and target network to generate the bootstrapping targets are shown in Fig. 11. Not including experience replay has drastic effects as the agent is not able to make any progress, as is evident from Fig. 11a. The agent fails within two steps (from Fig. 11a we can see that the reward accumulated is 2 for almost all episodes) and since the training samples are generated one after another, it does not have enough “different” data to learn effectively from.

If we use the same Q-network to generate the bootstrapped targets, we must include them in the gradient calculation. This actually shows some progress during training, seen from Fig. 11b, however the training is much slower than in Fig. 9 and also has very high variance. This is expected as the targets used for the training itself are not stable.