RL Agents for Taxi-V3

GROUP MEMBERS

1. Mohamed Anwar Ghanem
2. Sebenele Thwala
3. Stephen Oni
4. Engelbert Tchinde Wamba
5. Awa Ly

Abstract

In this paper, we are going to solve the Taxi-V3 environment using multiple Reinforcement algorithms and discuss the difference among them. In this paper, we use Q-learning, Sarsa, Expected-Sarsa and DQN algorithms to help our agent explore the environment.

ENVIRONMENT

In this project we use the different algorithms discussed in class to train an agent in the taxi-v3 environment. In this environment, the agent is a taxi cab. There are four locations at each of the corners of the grid. A passenger is waiting for the taxi at one location, our agent is supposed to pick up this passenger from this location and drive them to their designated location. Our agent will receive +20 points for a successful dropoff and lose 1 point for every timestep it takes. There is also an additional penalty of 10 points for illegal pick-up and drop-off actions.

In this problem, we have an action space of size 6 where these actions are given as

- **0** = south
- **1** = north
- **2** = east
- **3** = west
- **4** = pickup
- **5** = dropoff

We also have a state space of size 500 which encodes the agent's location, the passenger's location and the destination location. Our agent will then explore the environment and take actions based on the rewards defined in the environment.

ALGORITHMS

We use Q-learning, Sarsa, Expected-Sarsa and DQN algorithms to help our agent explore this environment.

Q-Learning

This is a value-based learning algorithm where our agent attempts to learn an optimal way to explore the environment. It does this by estimating which of the 6 actions available is most likely to lead to completion of the task or maximise the total reward.

We first create a Q-table of the shape (*state* x *action),* which we initialize as zeroes. This table will be updated with the Q-values, calculated using the Q-function:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a) + \gamma \, max_{a'} Q(s',a') - Q_{t-1}(s,a)]$$

As our agent explores the environment, the Q-function gives us improved approximations by updating the Q-values in the Q-table.

The Q-learning algorithm is thus as follows:

1. *Initialize Q-table as zeroes*
2. *Iterate the following until a good Q-table is obtained*
    - *Choose and perform an action*
    - *Measure Reward*
    - *Update Q-table*

Once a good Q-table is obtained, our agent will then start to explore the environment, taking the actions that ensure maximum rewards using the Q-table.

DQN

This is similar to the Q-learning algorithm, the only difference being that a neural network is used to approximate the Q-value function instead of a table. The current state of the agent is given as the input to the network and the Q-value of all possible actions is generated as output.

Therefore what we have is that in Q-learning, a table maps each state-action pair to its corresponding Q-value while in deep Q-learning(DQN), a neural network maps input states to action, Q-value pairs.

The DQN algorithm is thus as follows:

1. *Initialize Main and Target Neural Networks*
2. *Choose an action*

3. *Store state, action and reward in memory*
4. *Update network weights using the memory to generate input and target, and the Q-function*

SARSA

The SARSA algorithm is a slight variant of the above discussed Q-learning algorithm. The difference between the two is that Q-learning is off-policy while SARSA is on-policy. What this means is that in Q-learning the agent learns the value function according to the action derived from another policy, while in SARSA the agent learns the value function according to the current action derived from the policy currently being used.

Because of this difference the Bellman equation that we use to update the Q-value is given by :

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a) + \gamma\, Q(s',a') - Q_{t-1}(s,a)]$$

EXPECTED SARSA

The expected SARSA algorithm is very similar to SARSA and Q-learning and differs only in that action value function it follows. It can also be used as either an off-policy or on-policy technique and hence a more flexible approach compared to the algorithms we have discussed above.

It has the following update rule:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R_{t+1} + \gamma\sum_a \pi(a|s_{t+1})Q\ (s',a) - Q_{t-1}(s,a)$$

As shown in the update rule, EXPECTED SARSA takes the weighted sum of all possible next actions with respect to the probability of taking that action.

RESULTS AND ANALYSIS

We implemented all the above algorithms using the taxi-v3 environment. Full code can be found here. For each agent we ran for about 20,000 episodes. We played around with different values of the parameters and compared how the different agents performed on the environment.

We will first compare and discuss the performances of the Q-learning, SARSA and Expected SAR as we play with the hyperparameters. Finally we will discuss the DQN agent which we found doesn't perform very well for simple environments.

Experimenting with the hyperparameters we found the following:

EXPERIMENT ONE

We obtained the following results after using hyperparameter values shown in Figure 1. We found for each agent the best average reward, averaged over 100 consecutive episodes and running it for 20,000 episodes in total, to be:

AGENT	***BEST AVERAGE REWARD***
Q-learning	8.4616
SARSA	-8.744
Expected SARSA	8.2522

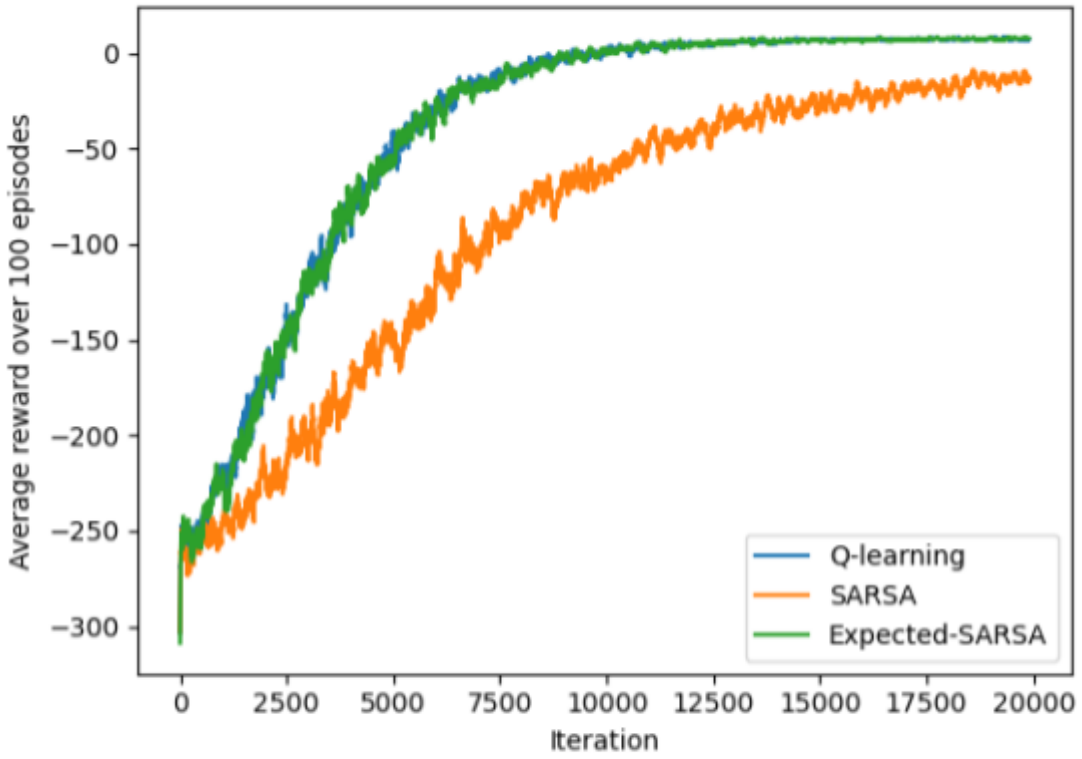

Figure 1. With hyperparameter values gamma = 0.999, alpha = 0.01, epsilon = 1.0 min_epsilon = 0.005

We note that with these values the SARSA agent performs poorly as compared to the others as it obtains a negative best reward and trains slower than the other two. Comparing these results with the top rewards that an agent can get in the taxi environment our results, though not very poor, could improve.

EXPERIMENT TWO

We continued experimenting with the hyperparameters and saw after using the values shown in Figure 2, we saw a considerable improvement in the training time for all three agents and in their performance. After 20,000 episodes the best average results for each agent were as given below

AGENT	***BEST AVERAGE REWARD***
Q-learning	8.731
SARSA	8.512
Expected SARSA	8.614

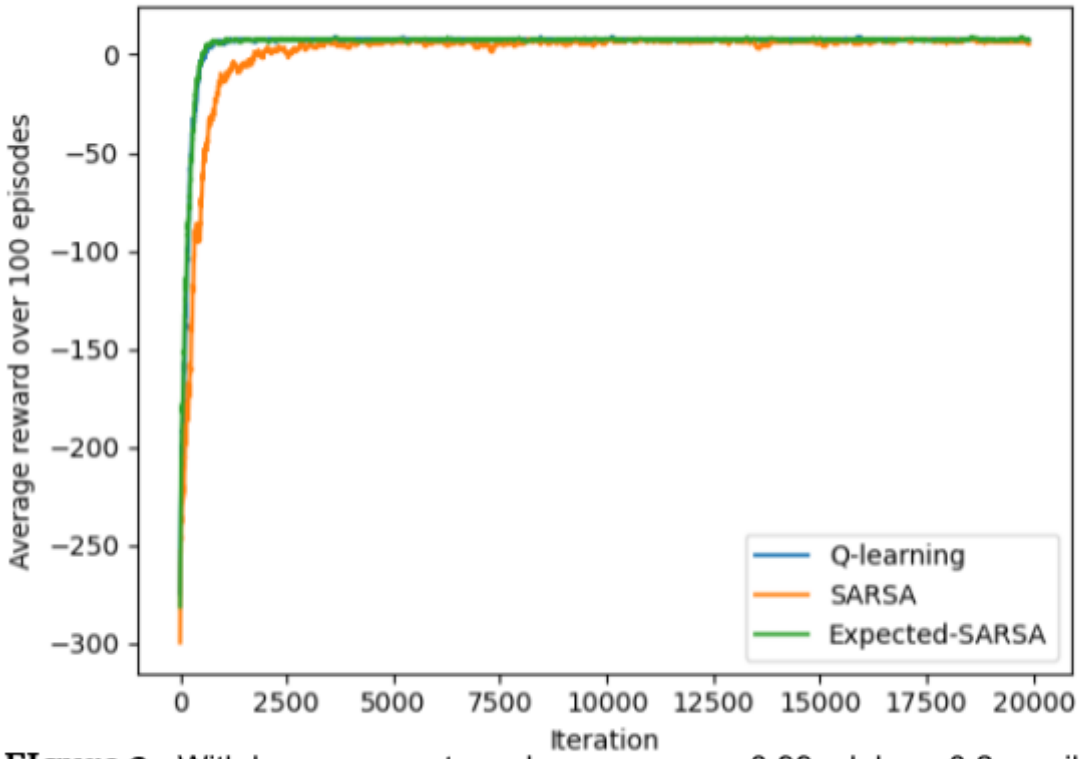

FIgure 2. With hyperparameter values: gamma = 0.99, alpha = 0.2, epsilon = 1.0 min_epsilon = 0.005

This was a considerable improvement to the results we obtained previously. We note the SARSA agent, though improved, still trains slower than the other agents and also obtains a slightly lower reward.

EXPERIMENT THREE

We continued playing around with the hyperparameters and after many different values the best results we obtained were we as shown below:

AGENT	*BEST AVERAGE REWARD*
Q-learning	8.788
SARSA	8.613
Expected SARSA	8.929

We note that all 3 performed considerably well, with the Expected SARSA performing the best.

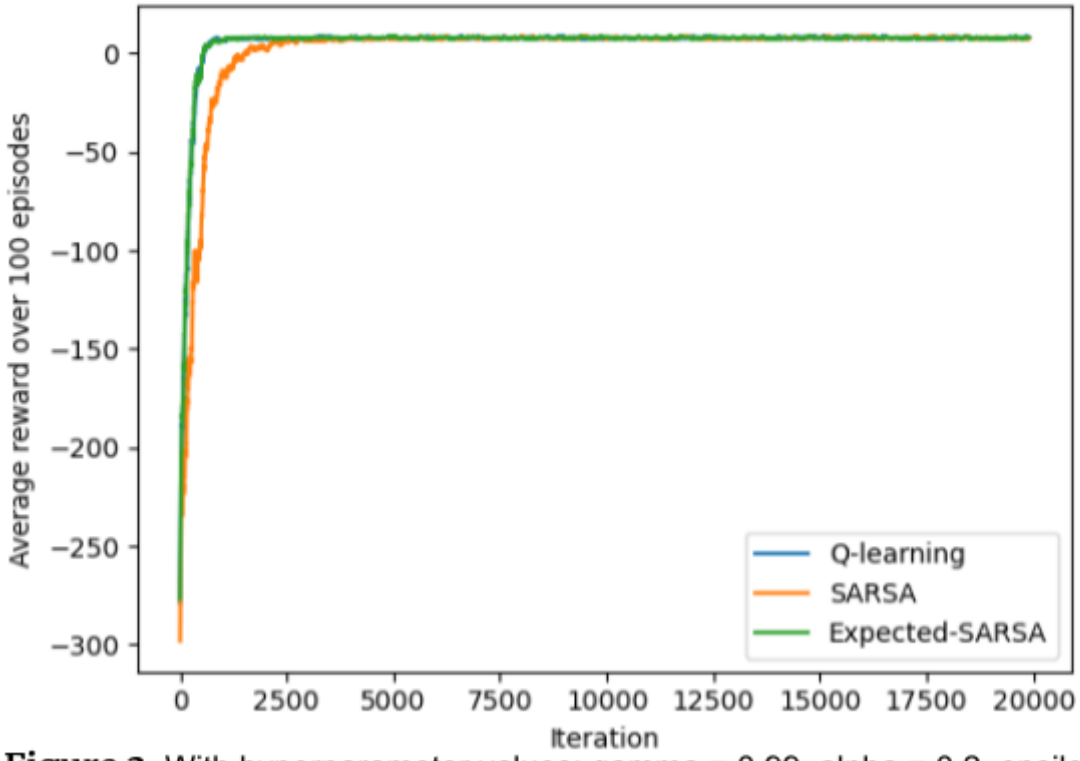

Figure 3. With hyperparameter values: gamma = 0.99, alpha = 0.2, epsilon = 1.0 min_epsilon = 0.005

EXPERIMENT FOUR

We then proceeded to train and test the DQN agent. This proved difficult as the results were unreliable. In addition, due to the nature of the neural network, it was computationally expensive to train our agent for a large number of episodes.

The agent did not converge as shown by Figure 4.

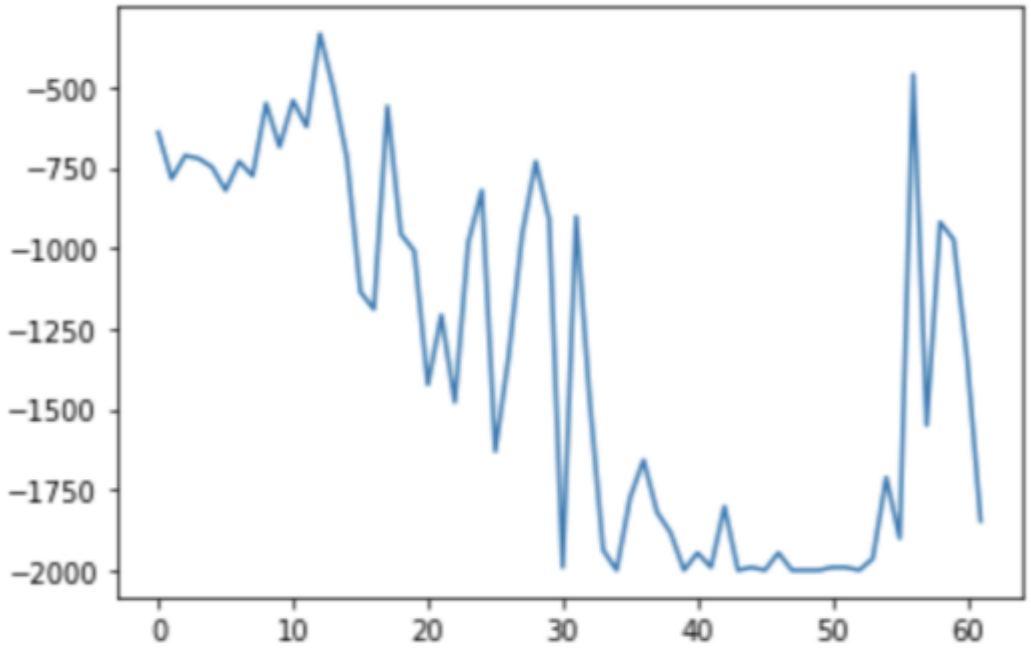

Figure 4. Performance of the DQN agent after 60 episodes

Discussion and Conclusion

From running multiple experiments with different hyperparameters for the four agents we discussed above we found that the DQN algorithm does not perform well for simple environments like the one we considered, hence we got inconsistent results and the agent never converged.

For the other 3 agents, we found that the SARSA agent, though performing well, trains longer and converges to a lower reward than the other two. However all agents perform relatively well in this environment as the best rewards that an agent can get in this environment is around 10.

We cannot conclusively say which agent between Q-learning and Expected SARSA is the best performance as they seem to converge to the same reward values as shown by Figures 1, 2 and 3.

Resources:
Code for the project on github: [github repository]