Interactive Learning
Homework 3

Soleiman BashirGanji
810103077

Question 1
In a Q-learning problem, sparse results slow down the convergence process. Q-learning works based on assigning a value to an action-state based on the estimate of the future rewards (bootstrapping). Sparse rewards means that the agent rarely gets any information about the value of actions, thus it has to explore the environment more extensively, and takes longer to converge to an optimal solution.

There are some ways in order to handle sparse rewards such as,

Reward shaping: Reward shaping means constructing a new kind of reward alongside the actual reward of the environment. If we are able to define meaningful rewards which can guide the agent towards the main reward, the sparsity of the rewards will be reduced.

Exploration strategies: It is good to choose proper exploration strategies which allows the agent to explore the environment enough. The algorithm and its parameters depend on the specific problem, but one must ensure that the exploration strategy take the sparsity of the rewards into account and allows for enough exploration.

Bootstrapping with Model-based Methods: We can train a model of the environment in order to generate more data and use that data to update our own policy.

Intrinsic Motivation: Introducing other rewards which reward curiosity can be helpful.

Except for sparse rewards, one of the biggest problems with a human in the loop is that the human rewards are usually inconsistent, biased and hard to measure or analyze. Human feedback is often imprecise, non-numeric, or qualitative. This means that the rewards are not just rare, but also don't necessarily offer enough guidance.

Ways of handling this can be:
Learning an embedding of the human: We can try and learn an embedding of the human. This embedding is supposed to hold information about the human and its interests. This way we can interpret the rewards while taking into account the embedding of the human, also we can use this embedding for reward shaping which allows the agent to try to move towards the human's interests.

Collaborative filtering - Looking at others: A human in the loop rarely gives us any rewards, but if we have many humans, we can use the data from all of them in order to improve performance.

For example, we can try to identify other humans which are similar together, and use the collaborative generated data by them to guide our agent.

Question 2.
In SARSA, we first take an action and then update the Q-values of that action-state. But in expected SARSA instead of taking the action, we update the Q-value based on an estimate of Q future values of all possible actions in the current state.

SARSA converges more slowly compared to the Expected SARSA, because it is based on taking single actions. But expected SARSA works based on expected values computed off of all possible actions in a state.

SARSA is more consistent with our target policy because it chooses an action based on policy, performs it and then updates the Q-values. But Expected SARSA and deviate from the main action slightly since it takes into account other actions too.

Expected SARSA is computationally costly compared to SARSA, in problems with limited resources, SARSA might be a better choice especially if we want to strictly follow our target policy. On the other hand, Expected SARSA has a higher rate of convergence and if we're ok with the computation cost and want faster convergence, Expected SARSA might be the better choice.

Question 3.
On-policy algorithms (e.g., SARSA) learn only from data generated by their current policy. This can lead to less sample efficiency since they don't use the data that is not generated using the current policy.
Off-policy algorithms (e.g., Q-learning) can learn from data generated by any policy (including previous or random policies), making them more sample-efficient.
On-policy methods focus on balancing exploration and exploitation during training, as the policy is being optimized/learnt but Off-policy methods can rely on separate exploration strategies, like epsilon-greedy, making them more flexible in exploring diverse parts of the environment.
Also, Off-policy algorithms are generally more robust in stochastic or random environments because they optimize for the best action regardless of the current policy.

A good real world example for using Off-policy algorithms is the "trading market". The trading market is complex and changes frequently. An Off-policy algorithm can adapt better in this environment since it can also use the huge amount of data produced by others in the past. Or another example can be self-driving cars for the same reason

Question 4.
The codes and explanations (markdown cells) are provided within a notebook.