DSAI - HW4: Classic control in reinforcement learning

Student ID: P76065013
Student Name: Leung Yin Chung 梁彥聰

Github and Submission Details

The source code of the experiment is saved as “main.py” in the github dexterleung92/dsai_hw_4 . The report is saved as “report.docx” and “report.pdf” with both Microsoft Office and PDF format available.

Analysis of Algorithm

Experiment Setup

The experiments are written in Python 3.6, working on the “MountainCar-v0” case in OpenAI gym, with Tensorflow for deep-learning related libraries.

Below 5 experiment model parameters setup: (Ref: main.py Ln481-485)

Train Name	**Timestep per Epoch**	**Total Epochs**	**Reward Function ID**	**Initial Learning Rate**	**Batch Size**
DQN-1	400	250	1	0.001	256
DQN-2	400	250	1	0.0005	256
DQN-3	400	250	1	0.001	128
DQN-4	800	400	0	0.001	256
DQN-5	800	400	2	0.001	256

Definitions and setup of the model parameters:

1. **Timestep per Epoch**
   The number of frame to proceed with a full game play. Noted the game may have not finished within limited timesteps in case the mountain car cannot climb to the goal position.
2. **Total Epochs**
   The number of full game play that the experiment will go over, including success and failed gameplay, and the final epoch is a test gameplay, which would not contribute to learning.
3. **Reward Function ID**
   0: Use default reward from the environment, i.e. constant -1 no matter at any states
   1: The position from the lowest boundary of the position domain
   2: Similar to 0, but with a 10-times of the timestep taken in case of a goal-achieved state
   Customized rewards were used because it could not learn well using constant rewards in Q-learning based on preliminary experiments and attempts from other developers. Possible explanation is that the machine has no information on whether to success solely from the given rewards, and it also cannot infer the goal from the observation, unlike taking screenshots of the game.
4. **Initial Learning Rate**
   The learning rate of the optimizer for deep learning
5. **Batch Size**
   The number of random timestep transitions sampled from a continuous series of states

6. **Optimizer**
   Adam optimizer was used
7. **Repeated Trials**
   5 runs were taken for each model to understand any variance if needed

Algorithm Overview

`dqnMountainCar()` is a standalone function for controlling a complete training. It initiates an Open AI environment, handles gameplay lifetime and prepares the repeated trials. Customized rewards as discussed above is also handled here.

`DeepQLearning()` is a class for handling a DQN training. It builds the Tensorflow graph and provide methods for training. On repeated trials, it has to be restarted where the Tensorflow graph and sessions will be reset. A target network and an evaluation network are both created.

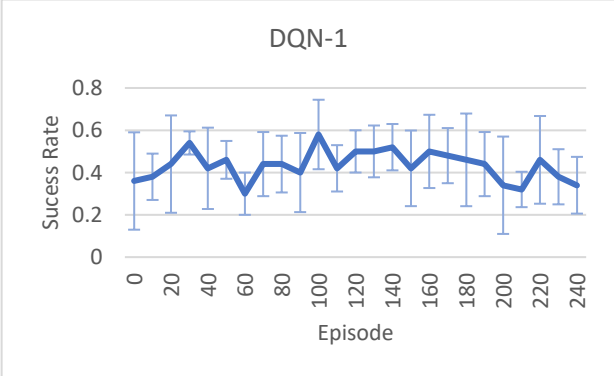
The evaluation network may provide estimation of Q-values for finding an optimized action from a given state in `getAction()` method. Noted epsilon soft policy is taken where it may not take greedy approach initially, while epsilon would go down for training and it will become greedier as times goes on.

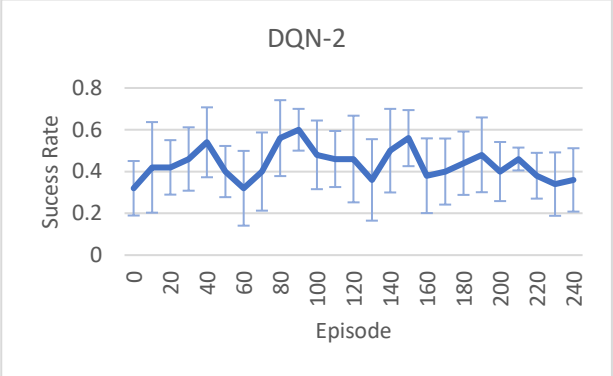
Experience replay is adopted, so the object has to remember a list of state transitions for later random sampling use. Every state transition is updated through the `addTransition()` method. There were a few transition recording implementations (main.py Ln148-181), where the 5 experiments have adopted the basic first in first out method. However, a variant of prioritized memory is added, where the best 5 episodes will be always kept as the pool of transition recap.

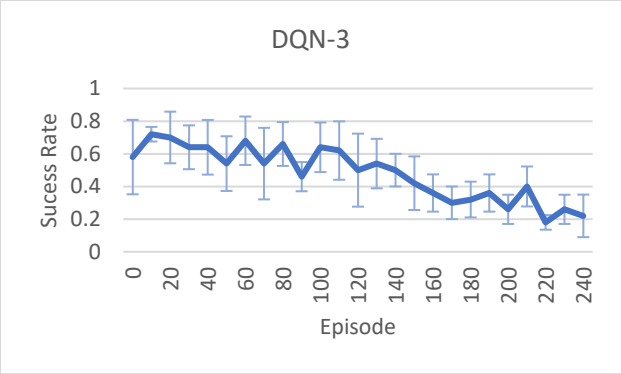
Experiment Analysis

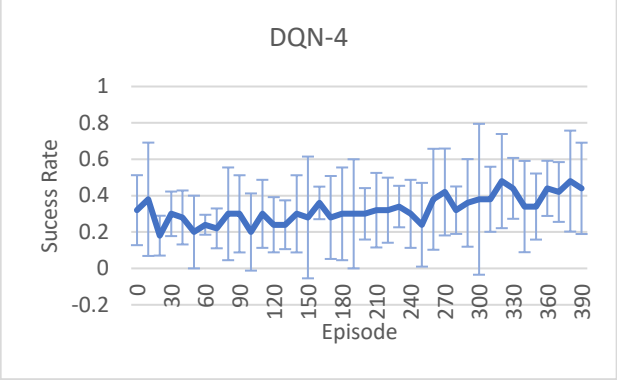
Results will be analyzed in the next few sections, where the data are logged as CSV files. Presentations below are modified using Microsoft Excel as found in the 2 Excel files just under the train name folders.

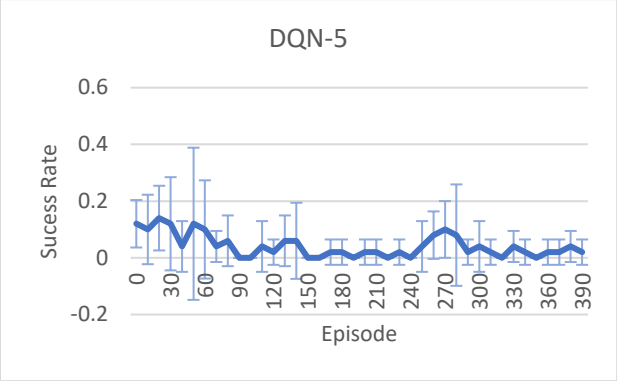
Analysis: Success or Not









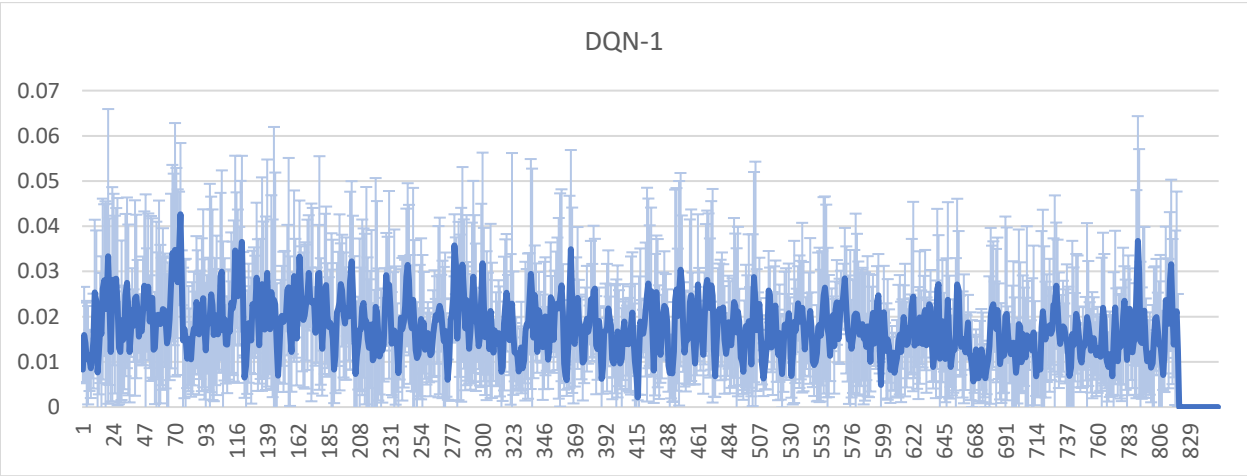


The graphs above are consolidated line charts on the 5 trainings, with y-axis as the success rate of climbing up on the goal location within every 10 episode of gameplays, with 1 up/bottom standard deviation plotted on each 10-episode interval across the 5 repeated trials.

It is noted DQN-1 and DQN-2 behaves more or less the same. DQN-3 has a relatively higher performance at the beginning, but a significant poorer performance trend throughout the learning. DQN-4 and DQN-5 performs worse, possibly because of the constant reward issue.

Based on the graphs, it looks the models have learnt nothing with no improving trend. Perhaps a more episodes should be done for further investigations.

Analysis: Loss during Training



To have better understanding the learning, the loss during training is observed for DQN-1. It is observed that the loss has not been converged to a low level, indicating the model has not learnt much within the recorded episodes. Noted the ending records are extremely low because of varied recording lengths across 5 trials. Some trials may have lower training times as they approach goals faster within an episode.

Algorithm Analysis Conclusions

Adjusting the rewards rather than using the default constant rewards may allow the programme to record successful episode in a faster way. However, there have not been observation with a significant learning progress based on DQN model.

Further studies may need to be taken on analyzing more episodes with successful records. In addition, since the problem cannot be regarded as a direct route to the goal, the car may need several up and down movement to accumulate inertia, the observation actually provides a very implicit hint to the goal. Unlike training games using CNN based on visual inputs, the given observation cannot understand how far away the goal is. The model may need multiple trials and errors to understand on which direction of the mountain to climb on and how high will the goal is. Perhaps with accumulated successful trials, the model may understand the rule behind the game.

In addition, accumulating non-successful episode memories may dilute the experience of successful memories. A better prioritized experience replay may need to be implemented to allow the model to learn the actual trajectory rather than learning the blindly testing on different routes.

Three Requested Questions

What kind of RL algorithms did you use? value-based, policy-based, model-based? why?

Deep-Q Learning has been used, which is a value-based RL algorithm. This was chosen in an attempt to allow self-learning for archiving a reward for long-run experience of the game play.

This algorithms is off-policy or on-policy? why?

It is off-policy, because it's not using the current policy to make decision. It's trying to update the Q-values using the next state and action. Therefore, the policy given to generate the data is based on the maximum returns from the next updates.

How does your algorithm solve the correlation problem in the same MDP?

The correlation problem is solved through experience replay, where a batch of random sampled observation transitions are taken from a continuous series of gameplay.