Contents

Task 1

1. Environment

The environment that we present is inspired by the maze arcade game Pac-Man. In our 11 by 11 square tabular environment, there are an agent, given number of ghosts, a power pellet, dots and walls, which are represented in figure 1 and 2 as A, G, P, '.' and # respectively. The walls are installed at specific locations and will not change in any circumstances. The dots are distributed to every space that is not a wall. Among the spaces that are not a wall, in which dots are placed, we randomly choose a space to locate the agent and spaces for the ghosts. Thus, at the beginning of the game, the agent and the ghosts are not at the same location. This procedure of placing the agent and the ghosts is done only when we reset the environment.

Figure 1 - Example of the environment

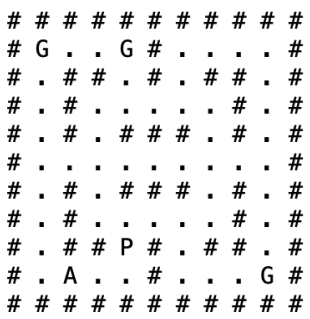
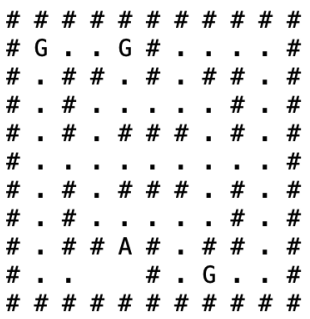


Figure 2 - After the agent moved



The environment follows a finite Markov Decision Process (MDP) for our next task in which we implement reinforcement learning. The terms for MDP are defined below:
• State space S: All possible configurations in the game, including the positions of the agent, positions of the ghosts, positions of the power pellet.
• Action space A: A set of all allowed actions: {left, right, up or down}.

2. Agent and actions

The agent is obviously an agent who explores the environment and is able to move either up, down, left or right by 1 space at each timestep and cannot stay at the same position. However, the agent cannot go to the direction where there is a wall, as it bumps into the wall. For example, in figure 1, the agent can move to either left or right, but cannot move to up or down because there are walls. If the agent is at the same location as the power pellet, it can eat the pellet and gain the ability to kill the ghosts by being on the same location. For example, if the agent on figure 1 moves to the right twice and up once, it reaches to the power pellet and eats it. The power of the pellet lasts only 5 timesteps. When the agent and any dot are at the same location, the agent can eat the dot, which is represented as a white space in figure 1 and 2. The power pellet, the ghosts and the dots disappear once they are eaten.

The ghosts are also able to move in the same way as the agent, however, the ghosts move with the given probability. For example, if the probability of 0.1 is given to the ghosts, the

probability of them moving is 0.1 and they do not move at all with the probability of 0.9. Figure 2 is 3 timesteps after figure 1 with the probability of 0.5 of moving for ghosts and shows that the ghost at the bottom right move only 2 spaces. As the ghosts are not an agent, their moves follow certain rules. Firstly, they cannot move to a direction where they were 1 timestep before. Secondly, they randomly pick a direction that is not prohibited. The ghosts are the enemies of the agent and kill it if they are on the same location.
The power pellet, the dots and the walls will not move in any circumstances.

3. Different dynamics of the environment and rewards

In this game, there are 3 terminate conditions. The game is over when any of these conditions is met. The first terminate condition, which is the goal of the game for the agent, is when all the dots are eaten by the agent. The second condition is when the agent is without the power and is killed by a ghost. Lastly, the game also ends when the time limit that is 121 (size of the environment, 11*11) expires.

The agent can obtain rewards in 3 ways. The first way is to eat a dot, which is worth 5 reward each. The second way is to eat the power pellet, which is worth 20 rewards. The third way is conditional and is possible only when the agent obtains the power by eating the power pellet and it has not expired. If the condition is met, the agent can kill a ghost and it is worth 50 rewards. Besides these 3 ways, there is another way of obtaining rewards, however, the reward is negative. The agent obtains -1 reward at every timestep regardless the move of the agent.

The environment contains two parameters that determines the details of it, which are number of ghosts and probability of moving for ghosts. Number of ghosts determines how many ghosts the environment contains and takes 1,2 or 3, thus, the maximum number of ghosts in the environment is 3 and the minimum is 1. Probability of moving for ghosts determines how likely the ghosts move at every timestep and takes an integer that is in the range between 0 and 1. As we adjust these parameters, the environment can be simpler or more complex.

Task 2

1. Algorithm

In this task, we apply Q-learning to our environment that we created in the previous task. Q-learning is a model-free reinforcement learning algorithm that provides agents with the capability of learning to act optimally in Markov Decision Process by experiencing the consequences of actions, without requiring them to build maps of the domains [1]. The learning process of Q-learning is Temporal Difference learning (TD), where an agent tries an action at a particular state and evaluates its consequences in terms of the immediate reward or penalty it receives and its estimate of the value of the state to which it is taken [1].

Equation 1 - Q-learning

$$Q_{new}(s_t, a_t) = Q_{old}(s_t, a_t) + \alpha * (r_t + \gamma * argmaxQ(s_{t+1}, a) - Q_{old}(s_t, a_t))$$

Equation 1 represents the process of Q-learning. Q, s, a and r represents Q value, state, action and reward respectively. New Q value is generated by modifying the old value. The modification part, which is shown after the learning rate α in equation 1, represents TD where we calculate temporal difference target by adding temporal reward to estimated optimal future value discounted by γ and subtract the old value from it.

The policy that the algorithm follows is epsilon greedy policy, which selects the action with the highest estimated reward most of the time [2]. Epsilon parameter decides how likely the algorithm takes random actions to explore possibilities of getting out of local optimum.

2. Case study

We apply the algorithm into the environment that we created in the previous task and investigate the results. Hyperparameter tuning is to be applied by conducting grid search. In grid search, the hyperparameters tuned are gamma, alpha and epsilon. The best hyperparameters are to be used for training models on 2 different environments. The first environment is where there is only 1 ghost who moves with the probability of 0.5, and another one is more complicated where 3 ghosts move with the probability of 1. In both grid search and training, the number of episodes conducted is set to be 100.

3. Evaluation of different hyper-parameters

Table 1, the result of grid search, shows that the algorithm achieved the average rewards of 97.67 with the hyperparameters, gamma 0.9, alpha 0.1 and epsilon 0.05. It is noticeable that an individual hyperparameter does not directly contribute to higher average rewards and it is the combination of each hyperparameter that affect the performance. In other words, there is no perfect value for any hyperparameters, and it is necessary to adjust them by grid search for the environment.

Table 1 - Result of grid search

	average rewards	average length	gamma	alpha	epsilon
0	89.12	47.23	0.80	0.05	0.05
1	69.64	59.21	0.80	0.05	0.10
2	82.03	62.12	0.80	0.05	0.20
3	90.88	52.72	0.80	0.10	0.05
4	80.77	57.33	0.80	0.10	0.10
5	89.02	57.28	0.80	0.10	0.20
6	54.07	54.73	0.80	0.20	0.05
7	63.02	49.83	0.80	0.20	0.10
8	77.05	60.35	0.80	0.20	0.20
9	88.97	55.18	0.90	0.05	0.05
10	69.59	59.21	0.90	0.05	0.10
11	82.13	62.47	0.90	0.05	0.20
12	97.67	48.53	0.90	0.10	0.05
13	68.50	58.60	0.90	0.10	0.10
14	69.67	61.03	0.90	0.10	0.20
15	40.57	69.33	0.90	0.20	0.05
16	33.34	54.11	0.90	0.20	0.10
17	76.05	62.10	0.90	0.20	0.20
18	59.96	57.49	0.99	0.05	0.05
19	87.64	57.96	0.99	0.05	0.10
20	38.12	65.73	0.99	0.05	0.20
21	91.11	48.09	0.99	0.10	0.05
22	33.27	55.18	0.99	0.10	0.10
23	55.35	55.50	0.99	0.10	0.20
24	68.65	50.75	0.99	0.20	0.05
25	66.18	59.02	0.99	0.20	0.10
26	82.96	62.74	0.99	0.20	0.20

4. Performance

Figure 3 shows rewards at each episode and average rewards by the episode. It is shown that average rewards steadily increased thorough the entire process, but it slows down from around episode 20. This indicates that the highest average rewards that the algorithm is capable of achieving is around 100 and it learned the best actions by episode 20. It is also shown in the figure is that rewards at each episode significantly vary, which indicates that the performance of the algorithm strongly depends on how the stochastic environment is generated. For example, the agent would achieve no reward in the environment where starts with the agent surrounded by the ghost and the walls and it moves towards the ghost accidentally.

Figure 3 - Rewards and average rewards

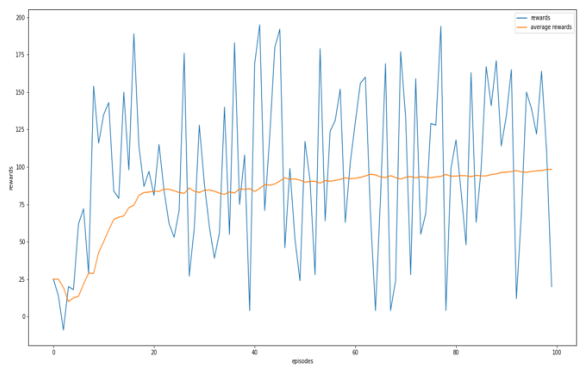


Figure 4 - Length and average length

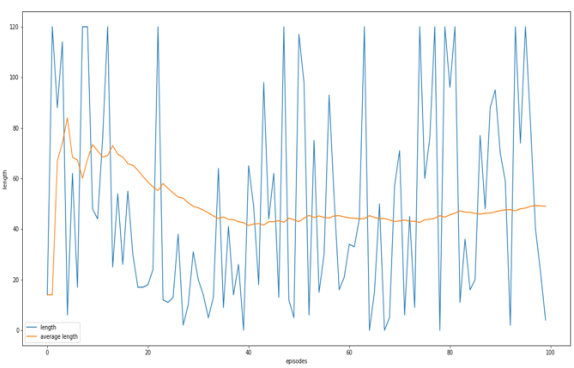


Figure 4 shows total timesteps of each episode and average timesteps by the episode. Compared to figure 3, average timesteps do not necessarily increase as average rewards increase by around episode 40. On the contrary, average timesteps decrease from the beginning to episode 40 while average rewards increase. This suggests that the algorithm finds the local optimum at around episode 20 to 40, and longer timestep does not necessarily increase total rewards in the beginning as the agent does not move as much as it does in later episodes when the algorithm has not learned to earn rewards or even move.

5. More complex environment

We also trained a model in the environment where 3 ghosts move 1 step at every timestep. In figure 5, the same trend as figure 3 is shown, however, the average rewards achieved is around 85 and lower than 100 achieved by the same algorithm in the simpler environment. It is also shown that the range of rewards in figure 5 is narrower than in figure 3, which suggests that the algorithm less likely achieve rewards that are exceptionally higher or lower than the average in more complex environment. In addition to this, it is shown that it took around 50 episodes to find this local optimum in the more complex environment, which indicates that the more the environment is, the longer the algorithm takes to learn.

Figure 5 - Rewards and average rewards

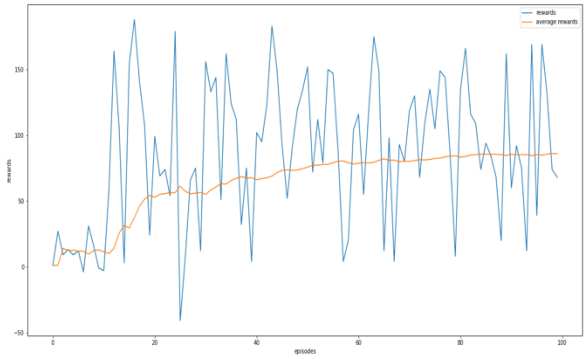


Figure 6 - Length and average length

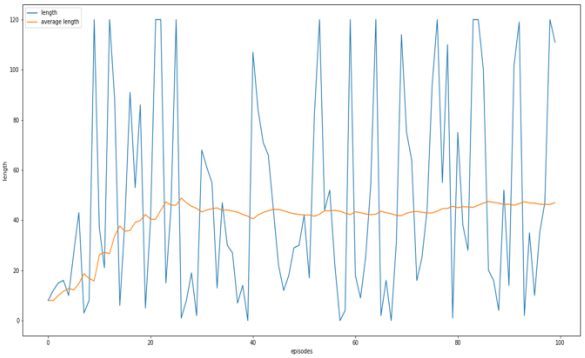


Figure 6 shows a different trend from figure 4, that average timesteps increase since the biggening. It is suggested that the risk of being killed by the ghosts is much higher in more complex environment and the agent does not survive for a long time even when the agent has not learned and does not move a lot in the earlier episodes.

Task 3

1. Algorithm

In this task, we apply Soft Actor Critic (SAC), which is an off-policy maximum entropy actor-critic algorithm that provides for both sample-efficient learning and stability [3]. SAC can be derived starting from a maximum entropy variant of the policy iteration method [4].

Equation 2 - Objective function

$$J_\pi(\phi) = \mathbb{E}_{St\sim D}[\mathbb{E}_{at\sim\pi\phi}[\alpha \log\left(\pi_\phi(a_t|s_t)\right) - Q_\theta(s_t, s_t)]]$$

Equation 2 is the objective function of SAC consisting of both a reward term and an entropy term weighted by α. In order to optimise this function, SAC uses three networks, a state value function, a soft Q-function and a policy function.

Equation 3 - Value function

$$V(s_t) = \mathbb{E}_{at\sim\pi}[Q_\theta(s_t, a_t) - \alpha\, log\, \pi(a_t|s_t)]$$

The value function used is shown as equation 3. We minimise the difference between the predicted value and the expected prediction weighted by the entropy of the policy function.

Equation 4 - Soft Q-function

$$J_Q(\theta) = \mathbb{E}_{(st,at)\sim D}[\frac{1}{2}(Q_\theta(s_t, s_t) - (r(s_t, s_t) + \gamma\mathbb{E}_{st+1\sim p}[V_{\bar{\theta}}(s_{t+1})]))^2]$$

Equation 4 is the soft Q-function used in SAC. We train the Q network by minimising the error of this function, which is the squared difference between the prediction of the value function and the sum of the immediate reward and the discounted expected value of the next state.

Equation 5 - Policy function

$$\pi_{new} = \arg min\, D_{KL}\left(\pi(\cdot, s_t) \parallel \frac{\exp\left(\frac{1}{\alpha} Q^{\pi\, old}(s_t, \cdot)\right)}{Z^{\pi\, old}(s_t)}\right)$$

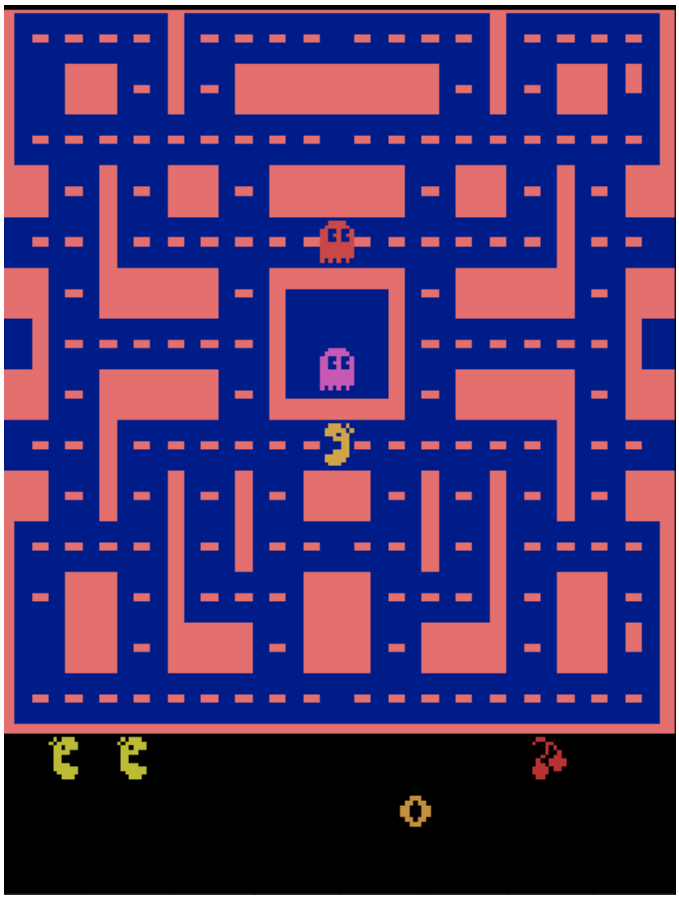
In the policy improvement step, we update the policy, which is shown as equation 5, towards the exponential of the new soft Q-function [4]. The D_{KL} function used in the equation is the Kullback-Leibler Divergence [5].

As it is shown in the equations, SAC the rewards and also the entropy of the policy simultaneously, that is, to succeed at the task while acting as randomly as possible [4].

2. Environment

As we apply more complex and sophisticated algorithm, we use a different environment from task 1 and 2. The environment chosen is the Atari 2600 game MsPacman [6], and the example is shown as figure 7. The basic rules of the game are same as the previous environment, which are that there is an agent, ghosts, walls and power pellets. However, the size of the environment is bigger and there are 4 ghosts and 4 power pellets. Also, the ghosts move as fast as the agent at the agent and will move faster as the agent proceeds to next stages. The agent has 3 lives, and the game is over when the agent gets killed by the ghosts 3 times. Another difference is that there are 4 warp tunnels to the sides in this environment, which allow the agent and the ghosts to move to the opposite side of the map. While the agent is entering the warp tunnels, the ghosts move slower than usual, which might help the agent to escape from them.

Figure 7 - Example of the environment



In this environment, instead of an RGB image of the screen, the observation is the RAM of the Atari machine consisting of 128 bytes, and each action is repeatedly performed for a duration of k*k* frames, where k*k* is uniformly sampled from {2,3,4} [6].

3. Case study

We conduct grid search to investigate how different hyperparameters in SAC affect its performance and then apply SAC with the default hyperparameters into the environment and analyse the results. In gird search, we have 2 hyperparameters, alpha and beta, and 2 different values for each parameter, and only conduct 10 episodes for each grid as it is very computationally expensive to apply grid search with several different parameters for a large number of episodes. We also compare our results with the previous empirical studies [7] [8] and see how good the performance of our model is.

In our model, the network used is Convolutional Neural Network, which consists of 3 convolutional layers, 3 max pooling layers, 2 fully connected layers and an output layer. In each convolutional layer, input and output channels are set to be 256 with kernel size of 1. Strides are set to be 4, 2 and 1 for the first, second and third layers respectively. Each max pooling layer follows each convolutional layer. After these layers, we move to the 2 fully connected layers which are applied ReLU activate function, and then the output layer.

4. Evaluation of different hyper-parameters

We conducted grid search in our model for hyperparameters, alpha and beta. Alpha represents the learning rate in the policy function, and beta represents the learning rates in the value function and the soft Q-function, which are both represented as α in equation 3,4 and 5. The grid values for alpha and beta are 0.01 and 0.03, and the number of episodes for each grid pair is 10.

Table 2 is the result of grid search, which shows that the algorithm achieved the highest average scores 588 with the hyperparameters, alpha 0.01 and beta 0.01, and achieved the lowest average scores 240 with the default hyperparameters, alpha 0.03 and beta 0.03. This indicates that learning rate 0.03 is too high for the algorithm to effectively learn in the environment. Our model achieved the average scores 348 with alpha 0.01 and beta 0.03, which is higher than the one with alpha 0.03 and beta 0.01. This suggests that the optimum learning rate for the policy function is higher than the one for the value function and the soft Q-function.

Table 2 - Result of grid search

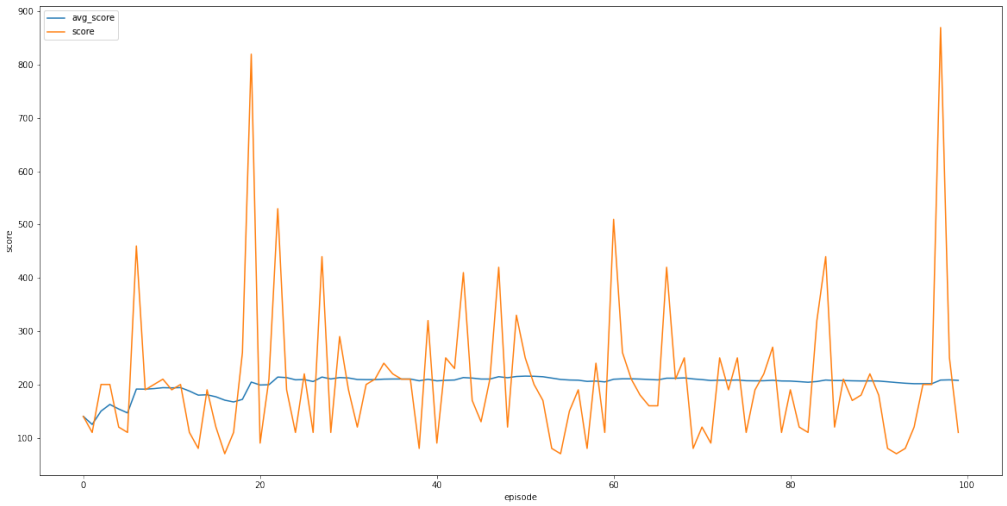
	average scores	alpha	beta
0	588	0.01	0.01
1	348	0.01	0.03
2	424	0.03	0.01
3	240	0.03	0.03

However, our grid search was conducted for only 10 episodes in 4 grid pairs, and further hyperparameter optimisation is necessary. For example, one might conduct grid search with more grids for the 2 hyperparameters for more episodes, e.g., 100, which allows us to see the effect of different hyperparameters in a longer period.

5. Performance

We applied our SAC model into the MsPacman environment, and the result is shown as figure 7. The average scores go up until around 200 by around episode 20, however, any change is not seen after episode 20. This suggests that the algorithm is not capable of achieving average scores that are higher than 200. On the other hands, scores of each episode significantly diverse compared to the average scores, which indicate that the stochasticity of the environment play a considerable size of a role in the algorithm and the agent cannot handle an environment where its stochasticity goes against the agent.

Figure 7 - Scores and average scores



In the previous empirical study [7], they applied SAC with networks consisting of 3 convolutional layers and 2 fully connected layers, which is the same as our model, despite the minor differences in the channels of the convolutional layers and the kernel sizes. They do not provide the information about how much average scores their SAC model achieved, however, they claim that their model achieved the maximum score 690.9 and the minimum score 141.8 after running 100,000 timesteps in the environment. Compared to their results, our model achieved the highest score 870 and the lowest score that was under 100, which is also a wide range. This suggests that our hypothesis that the performance heavily depends on the stochasticity of the environment is true.

Our experiment showed that our SAC model learned to play the Atari Pacman game better through the learning process. However, the performance was limited, and it is necessary to conduct further hyperparameter tuning and implement more sophisticated network architecture in order to achieve better performance. Thus, for further work, hyperparameter tuning on a machine with a good GPU is recommended. In addition to this, it is also recommended to compare our results with different algorithms. For instance, applying other off-policy algorithm such as Q-learning would allow us to compare the performances between off-policy algorithms in the environment.

Task 4

1. Algorithm

In this task, we apply Deep Q-Network (DQN) into the same environment as task 3. DQN is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks and to develop a wide range of competencies on a varied range of challenging tasks [8].

Equation 6 - Target Q-function

$$Q(s_t, a) = Q(s_t, a) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, p) - Q(s_t, a)]$$

Equation 6 represents the target Q-function of DQN, which replaces parameters with the latest network every steps.

The network architecture used is CNN, which is same as task 3, however, some parameters for the network are different. For the first, second and the third convolutional layers, the input channels are 3, 16, 32 and the output channels are 16, 32, 32 respectively. The kernel size is set to be 5 and the stride is 2 for each layer. Another difference is that this network contains only 1 fully connected layer as an output layer. ReLU activate function is to be applied for each layer.

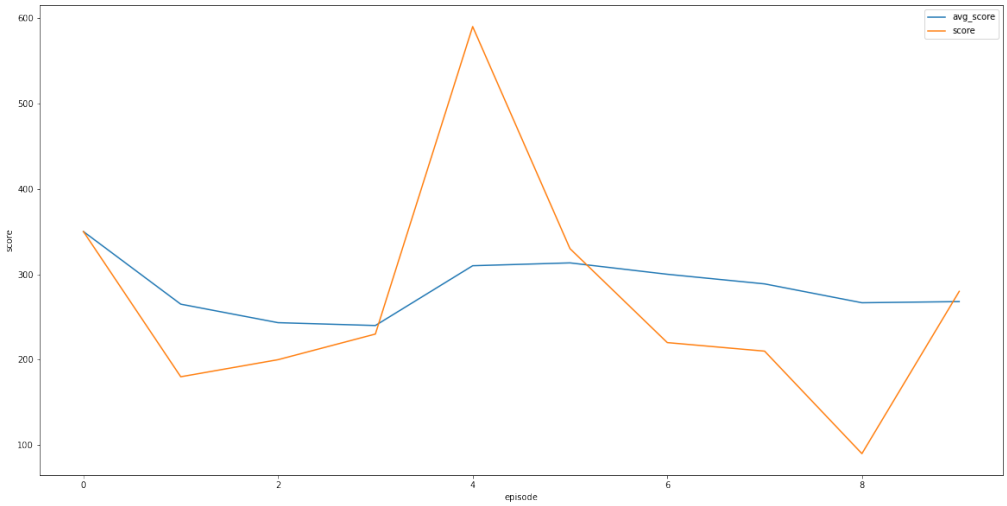
2. Case study

We apply DQN into the same environment as the one used in task 3 and compare the results. Number of episodes to run is only 10 for this case study as the procedure of training the DQN model is computationally very expensive.

3. Performance

As shown in figure 8, the average scores that the DQN algorithm achieved is constantly higher than SAC. Even at the early episodes, DQN easily achieved the average scores that are higher than 200, which was the maximum capacity of the SAC model. Moreover, DQN seems to have achieved these scores more steadily and learned faster, considering the algorithm was run for only 10 episodes. However, we cannot see any improvement in the scores and the algorithm even achieve the lowest score at a later episode. In order to see the actual capability of the algorithm, it is necessary to run it for at least 100 episodes, which allow us to fairly compare the result with SAC. In addition, to this, hyperparameter tuning is also required for further improvement.

Figure 8 - Scores and average scores



In conclusion, our experiment showed that DQN outperforms SAC in the Atari Pacman game, which is consistent with the previous empirical studies [7] [8]. This suggests that DQN performs better than SAC when the environment has complex rules and uncertainty.

Reference:

[1] C. Watkins and P. Dayan, 'Technical Note: Q-Learning', *Machine Learning*, vol. 8, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[2] M. Wunder, M. Littman, and M. Babes, 'Classes of Multiagent Q-learning Dynamics with -greedy Exploration', p. 8.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, 'Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor', p. 10.

[4] T. Haarnoja *et al.*, 'Soft Actor-Critic Algorithms and Applications', *arXiv:1812.05905 [cs, stat]*, Jan. 2019, Accessed: Apr. 08, 2021. [Online]. Available: http://arxiv.org/abs/1812.05905.

[5] J. R. Hershey and P. A. Olsen, "Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models," 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, Honolulu, HI, USA, 2007, pp. IV-317-IV-320, doi: 10.1109/ICASSP.2007.366913.

[6] OpenAI, 'Gym: A toolkit for developing and comparing reinforcement learning algorithms'. https://gym.openai.com (accessed Apr. 08, 2021).

[7] P. Christodoulou, 'Soft Actor-Critic for Discrete Action Settings', *arXiv:1910.07207 [cs, stat]*, Oct. 2019, Accessed: Apr. 08, 2021. [Online]. Available: http://arxiv.org/abs/1910.07207.

[8] V. Mnih *et al.*, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.