Multi-Agent Deep Q-Learning for the Berry Poisoning Game

Xiangyu Zhao
Trinity College
xz398@cam.ac.uk

Abstract

Deep Q-learning (DQN) is a successful algorithm that combines deep learning with reinforcement learning. However, it is of great research interest whether this method can work in a multi-agent environment. In this mini-project, I performed a multi-agent DQN method on the Berry Poisoning Games, and investigated on the agent performance with respect to different game environment parameters, including the bad berry rate, bad/good berry reward ratio, number of agents, and agent visibility range. The results clearly show that my DQN method can successfully train agents to act sensibly in such an environment, within only a few episodes of training. My DQN method also succeeds in transfer learning, training agents that still perform well in other game environment setups, and can be further enhanced through fine-tuning.

1 Introduction

Deep reinforcement learning (DRL) has achieved remarkable successes in games in the past decade, from simple games such as backgammon [1] and Atari [2], to more complex games such as Go [3] and StarCraft II [4], and has developed into a wide variety of methods. Q-learning is a simple and efficient model-free, off-policy temporal-difference (TD) control method that dates back to 1992 in [5], and is combined with deep learning (DL) in 2013 in [2], which achieved human-level performance in the Atari games. However, the Atari games consist of only a single agent, and it is curious to know whether it can also perform well in a multi-agent system. In this mini-project, I performed a multi-agent DQN method on the Berry Poisoning Games, and investigated on the agent performance with respect to different game environment parameters. In this report, I start with providing a detailed definition of the Berry Poisoning Game and its parameters, and the DQN method with relevant changes that I applied for this mini-project. I then describe the experiments I conducted in this mini-project, followed by an analysis of the results in terms of different game parameters, as well as other interesting findings, including agent-agent punishments and the transferability of the method.

2 Game Description

The Berry Poisoning Game is a multi-agent game where agents need to eat berries to survive. However, there exist both good and bad (poisonous) berries, and eating bad berries can have negative ramifications. In my implementation of the Berry Poisoning Game, the agents participate in a finite discrete domain (i.e., a grid world with fixed width W and height H), whose environment is defined as follows:

- There are $\lfloor \beta \times W \times H \rfloor$ berries in total in the world, whose positions are uniformly randomly distributed in the world, where $0 < \beta < 1$ denotes the berry abundance.
- β_b of the total berries are bad berries, where $0 \leq \beta_b \leq 1$ denotes the bad berry rate; this means that there are $\lfloor \beta_b \times \beta \times W \times H \rfloor$ bad berries in the world, and the remaining berries are all good berries.

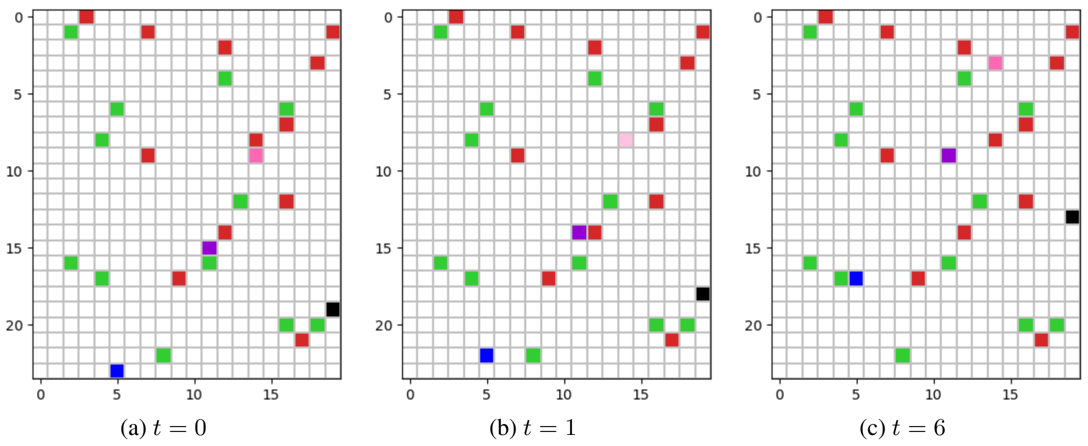


Figure 1: Visualisation of an instance of a 20×24 world in the Berry Poisoning Game with 12 good berries (green), 12 bad berries (red), and 4 agents (blue, violet, pink, red). The agents are manually set to only move up at every time step for the simplicity of visualisation. At (b) $t = 1$, the pink agent moves up 1 square and eats the bad berry at (14, 8), which causes it to be marked (visualised as lightened in colour). 5 time steps later, at (c) $t = 6$, the duration of the bad berry's penalty delay has passed, and the pink agent receives a penalty, while unmarking itself. The bad berry at (14, 8) has also respawned at some time step between $t = 1$ and $t = 6$.

- All good berries look identical, and all bad berries also look identical. The good berries and bad berries look differently, but the agents do not know which appearance corresponds to the good/bad berries at the start of the episode.
- All good berries share the same immediate reward $r_g > 0$.
- All bad berries share the same reward (penalty) $r_b < 0$ with a delay τ.
- Once a berry is eaten, it respawns with a fixed probability p at every time step.
- There are $n \geq 1$ agents in total in the world, whose starting positions are also uniformly randomly assigned.
- Each agent has a visibility range of d, which means it can see all berries and agents in the $(2d + 1) \times (2d + 1)$ grid around itself.

At every time step, each agent can perform one of the following actions, and receives the corresponding rewards:

- Move one square in one of the four directions: up/down/left/right, and eat the berry at the destination (if any). An agent at the boundary of the world can still move across the boundary, in which case it is 'teleported' to the other side of the world. If the agent eats a good berry, it is immediately awarded the good berry reward r_g; if it eats a bad berry, it is visibly 'marked' for the duration τ, and receives a bad berry reward (penalty) r_b after τ time steps. If the agent eats other bad berries during that duration, the effects of all the bad berries eaten are processed separately, and the agent's mark can only be cleared after all effects of the bad berries eaten wear off. This is done by maintaining a queue of bad berries eaten by the agent.
- Stay at its current place, and punish all marked agents nearby in the 3×3 grid around itself, at a cost of $c_p > 0$. For every marked agent it punishes in the 3×3 range, the agent is awarded a reward of $r_p > c_p$, and the punished marked agent receives a penalty of r_p. This automatically gives the agent a penalty for attempting to punish where there is no marked agent nearby.

Figure 1 represents the visualisation of such an environment.

3 Method

3.1 Mathematical background

In the Berry Poisoning Game, for an episode of T time steps, the process of an agent can be formalised as a finite partially observable Markov decision process (MDP), in which it interacts with the environment by observing a state $s_t \in \mathcal{S}$ at time $0 \leq t < T$, takes an action $a_t \in \mathcal{A}$, and receives a reward $r_t \in \mathbb{R}$ for that action plus finding itself in a new state s_{t+1}. The goal of the agent is to select actions in a way that maximises its future rewards. The future rewards at time t can be estimated by the standard definition of the expected discounted return, G_t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} \tag{1}$$

where $0 \leq \gamma \leq 1$ is a hyperparameter representing the discount rate. The optimal action-value function $Q^*(s, a)$ is defined as the maximum expected return of taking action a in state s at time t, under all policies:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[G_t | s_t = s, a_t = a] \tag{2}$$

where π is a policy mapping sequences to actions (or distributions over actions). The goal of the agent then becomes to estimate $Q^*(s, a)$. According to the Bellman optimality equation,

$$Q^*(s,a) = \mathbb{E}_{s_{t+1}} \left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| s_t = s, a_t = a \right] \tag{3}$$

This enables the optimal action-value function to be approximated by using the Bellman optimality equation as an iterative update. The Q-learning algorithm [5] provides a model-free, off-policy TD control method for such an update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{4}$$

where the $r_t + \gamma \max_a Q(s_{t+1}, a)$ part can be set as the target for the TD update, and $r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ serves as the TD error to be minimised. In practice, the policy of the agent derived from $Q(s, a)$ is often constructed using an ϵ-greedy strategy, which follows the greedy strategy $a \leftarrow \arg\max_a' Q(s, a')$ with probability $1 - \epsilon$ and randomly selects an action with probability ϵ. This can make sure that the agent can have adequate exploration of the state space.

3.2 Algorithm

The Q-learning algorithm suffers from unscalability as the state space is very large, and weak generalisability since the action-value function is estimated separately for each trajectory. The DQN algorithm [2] provides a solution to these problems, by parametrising $Q(s, a)$ using a neural network (NN), $Q_\theta(s, a)$. In order to use deep learning for Q-learning, the following two challenges need to be tackled:

Firstly, deep learning highly relies on the assumption that the training samples are independently and identically distributed. However, samples from the same trajectory have strong correlations between each other, and training directly using consecutive samples can be very inefficient. DQN resolves this by using the experience replay technique [6], where the agents' experiences at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a replay memory $\mathcal{D} = \{e_1, \cdots, e_N\}$ that is accessed to perform the weight updates. At each training iteration, it randomly samples a mini-batch of experiences from the replay memory $\mathcal{D}$, and use that to perform batched Q-learning updates. This breaks the correlations between the samples, and therefore reduces the variance of the updates. In practice, DQN only stores the last N experience tuples in the replay memory, and samples uniformly at random from $\mathcal{D}$ when performing updates.

Secondly, the naïve TD error for the Q-learning algorithm

$$L(\theta) = \left(r_t + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) \right)^2 \tag{5}$$

causes the target to change after every training iteration, resulting in unstable training. DQN solves this by maintaining a target network $Q_{\theta'}$ whose weights are duplicated from Q_θ and are fixed for a

Algorithm 1 Multi-Agent DQN for the Berry Poisoning Game: individual agent

```
Require: replay memory $\mathcal{D}$ with capacity $N$
Require: action-value network $Q_\theta$ with random weights $\theta$
Require: target network $Q_{\theta'}$ with weights $\theta' = \theta$
  for episode $= 1, \cdots, M$ do
      Get initial state $s_0$
      for $t = 0, \cdots, T-1$ do
          Preprocess $\phi_t \leftarrow \phi(s_t)$
          Select $a_t \begin{cases} \sim \text{Uniform}(\mathcal{A}) & \text{with probability } \epsilon \\ \leftarrow \max_a Q_\theta(\phi_t, a) & \text{otherwise} \end{cases}$
          Take action $a_t$, receive reward $r_t$ and new state $s_{t+1}$
          Preprocess $\phi_{t+1} \leftarrow \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Randomly sample a mini-batch of transitions $(\phi_i, a_i, r_i, \phi_{i+1})$ from $\mathcal{D}$
          Set $G_i \leftarrow r_i + \gamma \max_a Q_{\theta'}(\phi_{i+1}, a)$
          Perform a gradient descent step on $L(G_i, Q_\theta(\phi_i, a_i))$ according to Equation 6
      end for
      Update target network weights $\theta' \leftarrow \theta$
  end for
```

number of iterations. The weights of the target network $Q_{\theta'}$ are periodically updated to match the value network Q_θ. This yields the final DQN update:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \left(r_t + \gamma \max_a Q_{\theta'}(s_{t+1}, a) - Q_{\theta_t}(s_t, a_t) \right) \nabla_\theta Q_{\theta_t}(s_t, a_t) \tag{6}$$

A final modification of the standard Q-learning algorithm by DQN is that instead of storing a single state s_t in each experience tuple, DQN stores a fixed length representation of histories before s_t, produced by a function ϕ. This increases the agents' ability to learn potentially useful information from the previous states.

Algorithm 1 describes the DQN algorithm for the Berry Poisoning Game in this mini-project. In this algorithm, I update the weights of the target network $Q_{\theta'}$ after every episode. Note that instead of using the mean squared error (MSE) to update the model weights as proposed in the original DQN paper [2], I adopted the Huber loss [7] with $\delta = 1$:

$$L(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{if } |x-y| < 1 \\ |x-y| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{7}$$

This is less sensitive to outliers than the MSE loss, and makes training more stable. Note also that since there is no terminal state in the Berry Poisoning Game, there is no need to distinguish between the terminal and non-terminal cases for the expected return.

4 Experiments

4.1 Game environment setup

In this mini-project, I aimed to experiment on the effects of bad berry rate β_b, bad/good berry reward ratio $\frac{r_b}{r_g}$, number of agents n and agent visibility range d on the agent performance. For all experiments carried out in this mini-project, I let the agents play the Berry Poisoning Game in a 20×24 grid world with a fixed berry abundance $\beta = 0.05$. The respawn rate p of all berries was fixed at 0.33, and the penalty delay τ of the bad berries was fixed at 5. The cost for attempting punishment c_p was fixed at 1, and the reward for successfully punishing a marked agent r_p was fixed at 1.5. In order to control the bad/good berry reward ratio $\frac{r_b}{r_g}$, I fixed the reward for eating a good berry to be $r_g = 1$, and vary the value of the reward (penalty) for eating a bad berry r_b. Details of the values selected for the experimented variables are described in Section 5.

In terms of encoding the states of the Berry Poisoning Games for the NNs, I allocated 8×8 pixels for each square in the grid world, resulting in a 160×192 frame for each state. This is to make sure that the information in each square is not lost through convolutions. The good/bad berries are encoded with values $3/-3$, whose signs are randomly assigned at the start of each episode. This forces the good and bad berries to have various values across different episodes, and prevents the model from memorising a value for the good/bad berries through training for multiple episodes. The agent itself is encoded with values $1/-1$, with positive for unmarked and negative for marked. All other agents are encoded with values $2/-2$, under the same encoding convention for unmarked/marked.

4.2 Model structure and hyperparameters

In this mini-project, I defined the history function ϕ from Algorithm 1 to be stacking the current state and the last state:

$$\phi(s_t) = \{s_{t-1}, s_t\} \tag{8}$$

Therefore, for every state s_t, ϕ produces a $160 \times 192 \times 2$ frame that serves as the input to the NNs.

For the action-value network, I adopted a convolutional neural network (CNN) structure similar to the one used by the DQN paper [2]: my CNN model consisted of a convolutional layer with 16 filters of size 8×8 with stride 4, followed by two convolutional layers with 32 filters of size 4×4 with stride 2, followed by a fully connected layer with 256 hidden nodes. All four hidden layers were followed by a ReLU activation. My model then finally appended an output layer with 5 hidden nodes, each predicting a scalar value of taking one of the five possible actions: move up, right, down, left, and punish. The model was trained using mini-batches of size 32, and used the RMSprop optimiser with a learning rate of 0.01 and a decay factor of 0.99.

For the DQN algorithm, I set the discount rate to be $\gamma = 0.99$, and maintained a replay memory $\mathcal{D}$ with capacity $N = 10000$. I used an ϵ-greedy strategy to select the actions, where ϵ annealed exponentially from 0.95 to 0.05 with mean lifetime 200, within each episode:

$$\epsilon = 0.05 + (0.95 - 0.05) \times e^{-\frac{t}{200}} \tag{9}$$

Last but not least, at every iteration, I clipped the weights of the NN so that they remained in the interval $[-1, 1]$, in order to improve stability.

All agents within a game shared the same model weights, i.e., they used the same model to predict the action value $Q^*(s, a)$, and update the weights of the same model individually using their experiences. This means that I only needed to train one action-value network for each game setup.

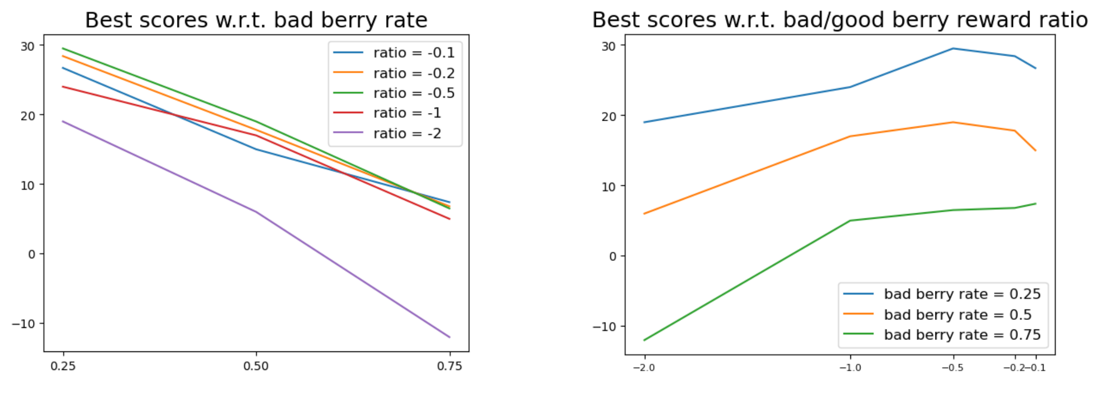
5 Results

In this mini-project, on the berry side, I experimented on the bad berry rate $\beta_b = 0.25, 0.5, 0.75$ and bad/good berry reward ratio $\frac{r_b}{r_g} = -0.1, -0.2, -0.5, -1, -2$ in a grid search manner, in the Berry Poisoning Game with 4 agents, each with visibility range of $d = 5$. Besides, on the agent side, I also experimented on the number of agents $n = 1, 2, 4, 8$ and the agent visibility range $d = 2, 5, 10$ separately, in the standard game setup with bad berry rate $\beta = 0.5$ and bad/good berry reward ratio $\frac{r_b}{r_g} = -1$. For each setup of the game parameters, I trained an action-value network for 50 different episodes, with 500 time steps in each episode. After training, I also performed visual simulations with the trained models, to examine the behaviours of the agents.

My DQN model was trained successfully under all game environment setups, showing a rapid growth in scores at the beginning few episodes and stays steadily ever since, with only minor fluctuations. This not only shows that my DQN model has been successful, but also suggests that my DQN model is quite data-efficient, and can quickly converge to its best ability within only a few episodes.

5.1 Bad berry rate

Figure 2a shows the scores of the best performing agents as a function of the bad berry rate β_b, split by different bad/good berry reward ratios $\frac{r_b}{r_g}$. The results showed a linear downward trend as β_b grows larger, which coincided with the intuition that if the world has a larger proportion of bad berries while the number of total berries remains unchanged, than an agent can only get less reward. There were no notable difference in the best scores when $-1 \leq \frac{r_b}{r_g} < 0$; however, there was a significant, drop on the best score for $\frac{r_b}{r_g} = -2$, which was disproportional to the differences in the



(a) (b)

Figure 2: Scores of the best performing agents with respect to (a) the bad berry rate β_b and (b) the bad/good berry reward ratio $\frac{r_b}{r_g}$, in the Berry Poisoning Game with 4 agents, each with visibility range of $d = 5$.

bad/good berry reward ratio from other scenarios. This is probably due to the difference between my implementation of DQN and the original DQN paper [2]: the original DQN method clips the rewards into the interval $[-1, 1]$, which further improves the training stability, whereas my implementation enabled the possibility that the reward of the agents can get below -1, which makes the gradient descent unstable. In fact, such unstability was enlarged when I performed test-runs for $\frac{r_b}{r_g} < -2$, which decreased the agent performances even further.

The visual simulation also cross-validated my hypothesis. In the $-1 \leq \frac{r_b}{r_g} < 0$ scenarios, at the beginning few time steps, the trained agent approached to both the nearby good and bad berries without preference, which was due to the fact that the agents did not know which were the good berries at the beginning. Then, after a few time steps when the agents have finally learnt to map the berry appearances with their good/bad properties, they grew a preference over the good berries than the bad berries. However, due to insufficient amount of history knowledge, the agents still occasionally made mistakes and ate the bad berries. In contrast, in the $\frac{r_b}{r_g} = -2$ scenario, although the agents could still learn a slight preference over the good berries, their error rates was significantly larger than the $-1 \leq \frac{r_b}{r_g} < 0$ scenarios, showing a relatively inefficient training outcome.

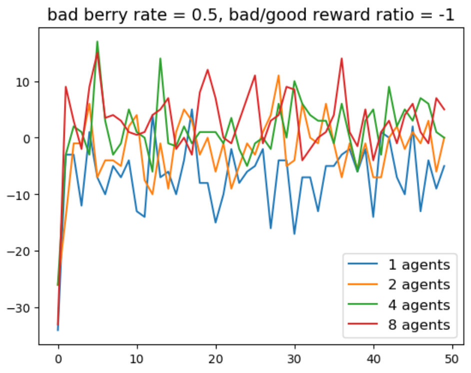
5.2 Bad/good berry reward ratio

Figure 2b shows the scores of the best performing agents as a function of the bad/good berry reward ratios $\frac{r_b}{r_g}$, split by different bad berry rate β_b. The results were consistent with the previous subsection: all 3 scores experienced a significant increase from $\frac{r_b}{r_g} = -2$ to $\frac{r_b}{r_g} = -1$, and the overall changes between $-1 \leq \frac{r_b}{r_g} < 0$ were not significant. The gaps between the three score curves were even, which cross-validates the linear downward trend of the scores with respect to the bad berry rate in Section 5.1.

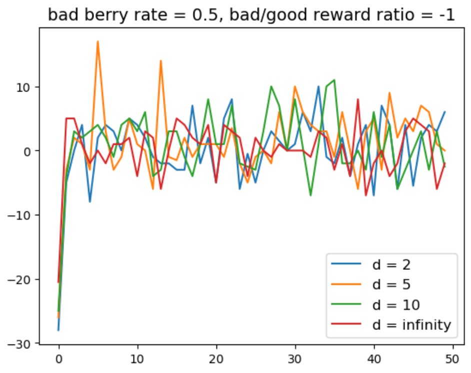
It is also worth noticing in both figures that the gaps between the $\frac{r_b}{r_g} = -2$ score and the $-1 \leq \frac{r_b}{r_g} < 0$ scores when $\beta_b = 0.25$ is smaller than the gaps elsewhere; this is likely to be because the reward is less likely to exceed -1 when β_b is smaller.

5.3 Number of agents

Figure 3a shows the learning curves of the final best performing agents with respect to the number of agents n, in the standard Berry Poisoning Game with bad berry rate $\beta_b = 0.5$ and bad/good berry reward ratio $\frac{r_b}{r_g} = -1$. The curves show that when the number of agents in a game is larger, the performance of the best performing agent can get better. This is likely to be due to the following two reasons:



(a)



(b)

Figure 3: Learning curves of the final best performing agents with respect to (a) the number of agents n and (b) the agent visibility range d, in the Berry Poisoning Game with bad berry rate $\beta_b = 0.5$ and bad/good berry reward ratio $\frac{r_b}{r_g} = -1$.

- Firstly, since the initial locations of the agents are uniformly allocated at random, having more agents can help explore the world more thoroughly, resulting in a better cumulative contributions toward the shared action-value network.
- Secondly, with more agents in a game, more experiences can be produces within the same number of time steps, which can then fill the replay memory $\mathcal{D}$ with the latest experiences more quickly, which are likely to be more important for training.

5.4 Agent visibility range

Figure 3a shows the learning curves of the final best performing agents with respect to the agent visibility range d, in the standard Berry Poisoning Game with bad berry rate $\beta_b = 0.5$ and bad/good berry reward ratio $\frac{r_b}{r_g} = -1$. The results showed no notable difference in terms of the agent performance when d grows larger. This is slightly counter-intuitive, but is likely to be due to the fact that the agents can move at most 1 square at a time, and having a visibility range greater than necessary does not contribute much to the decision making. However, I still cannot overlook the possibility that the DQN method might work poorly in this case; it is worth more detailed investigation in the future.

5.5 Punishment

The visual simulation shows that in all scenarios, the agents have successfully learnt to punish nearby marked agents, and only attempt punishment when there are marked agents nearby. However, the agents neither showed the tendency to approach other marked agent, nor tried to escape from other agents when they were marked – They simply performed punishments when they are coincidentally near a marked agent. This shows that the agents were trained to focus on maximising their overall scores through eating more good berries and less bad berries, rather than competing against each other.

5.6 Transferability

I also tested the transferability of my trained model by transferring the model trained for the task with berry abundance $\beta = 0.05$, bad berry rate $\beta_b = 0.5$, bad/good berry reward ratio $\frac{r_b}{r_g} = -1$, number of agents $n = 8$, and agent visibility range $d = 5$, to a different task where $\beta = 0.05$, $\beta_b = 0.25$, $\frac{r_b}{r_g} = -0.5$, $n = 4$ and $d = 8$. I first evaluated the performance of the original model on the new task as the baseline, and the performance of the model trained directly for the latter task as the target. I then trained a copy of the original model in the new task's environment for a further 30 episodes, and tested its performance on the new task to assess its transferability. Since this fine-tuning can be

Model	Mean best score	Max. best score
Direct transfer evaluation of the model ($\beta = 0.05$, $\beta_b = 0.5$, $\frac{r_b}{r_g} = -1$, $n = 8$, $d = 5$)	22.6	33.5
Transfer learning from the model ($\beta = 0.05$, $\beta_b = 0.5$, $\frac{r_b}{r_g} = -1$, $n = 8$, $d = 5$)	26.0	36.5
Original model trained for the task ($\beta = 0.05$, $\beta_b = 0.25$, $\frac{r_b}{r_g} = -0.5$, $n = 4$, $d = 5$)	23.6	44.0

Table 1: Results of the transfer learning experiments

regarded as a continuation of DQN training, I set the initial ϵ to be the final value of ϵ in the training for the original task (i.e., the annealed ϵ value after 50×500 steps).

The results of the experiments are reported in Table 1, showing that on the new task, even without fine-tuning, the model can achieve a mean best score very close to the model directly trained on that task, and the model after fine-tuning can even exceeds the mean best score by the model directly trained on the new task. In terms of the maximum best scores, the model with fine-tuning performed better on the transferred task than the original model without fine-tuning, but still scored less than the directly trained model. The phenomenon that the fine-tuned model achieves a higher mean best score but a lower maximum best score than the directly trained model can be caused by a more stable action distribution (i.e., for any given state, the dominant action for that state has a greater probability and is more likely to be selected) learnt through an overall longer training. The success in transferability also suggests that the agents have learnt similar playing styles across different game environment setups through my DQN method.

6 Conclusion

In this mini-project, I performed a multi-agent DQN method on the Berry Poisoning Games, and investigated on the agent performance with respect to different game environment parameters, including the bad berry rate, bad/good berry reward ratio, number of agents, and agent visibility range. The results clearly show that my DQN method can successfully train agents to act sensibly in such an environment, within only a few episodes of training. My DQN method also succeeds in transfer learning, training agents that still perform well in other game environment setups, and can be further enhanced through fine-tuning. However, due to insufficient amount of history knowledge, there is still rooms for improvements.

Further research directions for this mini-project can include adding more history knowledge by stacking more previous states for the input, or try a CNN + LSTM structure that has a better potential in learning sequential data. It is also of great interest to investigate on the communications and collaborations between different agents.

References

[1] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *The 27th Conference on Neural Information Processing Systems (NIPS 2013) Deep Learning Workshop*, 2013.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[4] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[5] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[6] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, 1992.

[7] Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

A Training results

The curves show the scores of the best performing agents during DQN training. All experiments were conducted with berry abundance $\beta = 0.05$, number of agents $n = 4$ and agent visibility range $d = 5$.

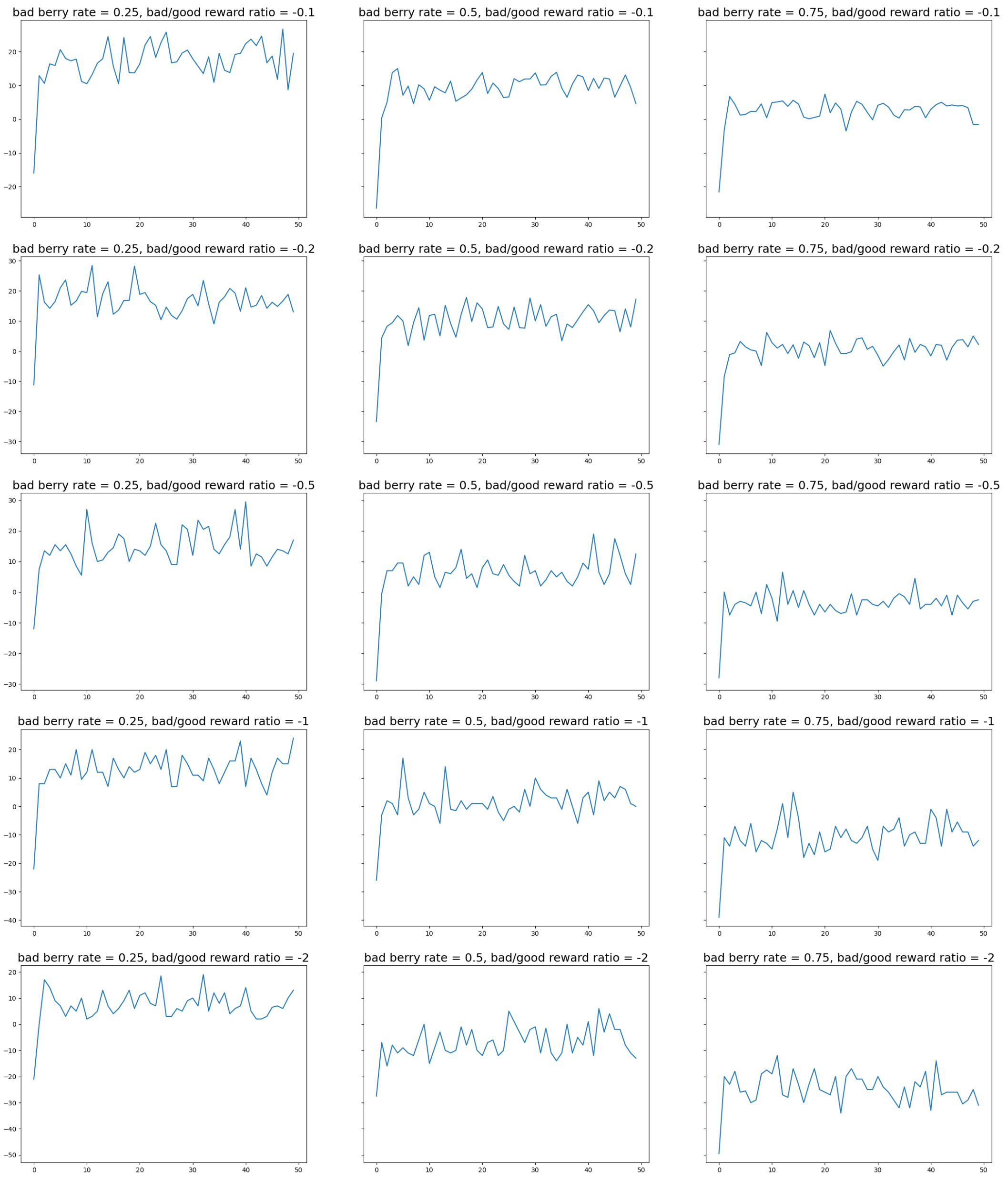


Figure 4: Learning curves of the experiments with $\beta = 0.05$, $n = 4$ and $d = 5$.