KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

# Exploring Multi-Agent Learning

Report

Part 2

Evmorfopoulos–Léonard–Maoujoud

Academic Year 2018–2019

H0T25a: Machine Learning: Project

**Angelos Evmorfopoulos**
**Paolo Léonard**
**David Maoujoud**

**Abstract**

**Machine Learning Project: Exploring Multi-Agent Learning**

The present report summarizes tasks 2 and 3 of the machine learning project. The stateless game *Rock-Paper-Scissors* is first analysed in an independent learning setting using Q-learning, the results indicating that the policy trajectories tend to converge to a mixed-strategy equilibrium. The same game is then studied in a joint-action learning setting. When confronting a Q-Learner with a fictitious player, the fictitious player's policy ends up reacting to the Q-Learner's policy changes in an effort to keep the upper hand. Fictitious play and Q-learning are then combined into one to produce the most robust learner overall. Lastly, a stateful game called the *Harvest Game* is considered in an independent learning setting. Due to the complex nature of its state-action space, a convolutional neural network has been trained on features extracted from the agent observations to approximate the optimal agent policy. As the model is trained, the agents playing the game appear to gravitate towards a cooperative strategy instead of trying to minimize other players' scores by firing at them.

Key words: Q-learning, fictitious play, convolutional neural network, game theory, replicator dynamics

**Work distribution**

- *Angelos*: ± 80 hours total. On the first task 1 worked on the comparison of learning models and trajectories and on task 2 on fictitious play. I worked on feature extraction and part of the evaluation on task 3.
- *David*: ± 80 hours total. I worked on the replicator dynamics and independent Q-learning on tasks 1 and 2, and on the neural network structure, parameter tuning, and the model evaluation on task 3.
- *Paolo*: ± 80 hours total. On task 1 I worked on the implementation of the games and the Q-Learner. I developed opponent modeling and its combination with a Q-Learner for task 2. For the last task I mostly worked on the design of the main agent loop and part of the evaluation.

# 1 Introduction

The second part of the *Machine Learning: Project* assignment is divided into two tasks. Section 2 revolves around the notion of *opponent modeling* in the Rock-Paper-Scissors game. Section 3 considers the *Harvest Game*. More specifically, a deep learning model is trained using features that describe the game state in order to circumvent the complexity of the state-action space inherent to games of this size.

# 2 Task 2: Opponent modeling

## 2.1 Problem Statement

The first task addressed the notion of **independent learning** [1], a learning setting in which both players do not communicate with each other, in two-agent, two-action matrix games. Task 2 extends upon this by introducing **joint-action learning (JAL)** [2], defined as the ability of the players to observe and learn from the behavior of other agents. Section 2.2 serves as an introductory step to this task. The Rock-Paper-Scissors game is first implemented in the same **independent learning** setting used in task 1. To this end, Q-learning is utilized for both agents. To achieve JAL, a model of the opponent is built. In particular, Section 2.3 introduces a form of opponent modeling called *fictitious play*. Finally, section 2.4 combines opponent modeling and Q-learning.

## 2.2 Q-learning in the Rock-Paper-Scissors game

**The two-dimensional directional fields have been utilized to visualize two action replicator dynamics. How may replicator dynamics be observed in two-agent, three-action games?**

The matrix form, two-agent replicator dynamics are given by [3]:

$$\frac{dx_i}{dt} = x_i[(Ay)_i - x^T Ay] \tag{1}$$

$$\frac{dy_i}{dt} = y_i[(Bx)_i - y^T Ax] \tag{2}$$

where $A$ and $B$ are the payoff matrices for populations 1 and 2 respectively, $x_i$ is the frequency of individuals of agent 1 playing strategy $i$, $y_i$ is the frequency of individuals of agent 2 playing strategy $i$. The payoff matrices of the Rock-Paper-Scissors game are as follows:

$$A = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \tag{3}$$

The *directional field plots* used for two-action games can no longer be utilized for three-action games. Instead, *ternary plots* make it possible to observe the replicator dynamics of one of the agents. Figure 1 illustrates the replicator dynamics of the Rock-Paper-Scissors game. The replicator dynamics have been plotted using the `egtplot` library [4]. The game has no pure-strategy Nash Equilibria, it does however have a mixed-strategy Nash Equilibrium at $(R, P, S) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

### Q-learning trajectories

The action-value update function $Q : S \times A \rightarrow \mathbb{R}$ is given by [5]:

$$Q^{new}(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \, max_{a' \in A} Q(s', a')) \tag{4}$$

For stateless games, this function may be simplified to a function of signature $Q : A \rightarrow \mathbb{R}$ :

$$Q^{new}(a) \leftarrow (1 - \alpha)Q(a) + \alpha(r + \gamma \, max_{a' \in A} Q(a')) \tag{5}$$

Balance between *exploration* and *exploitation*[1] is ensured through an action selection mechanism called the *Boltzmann exploration method*. An action $a_i$ is selected with probability [2] [6]:

$$p(a_i) = \frac{e^{Q(a_i) \cdot \tau^{-1}}}{\sum_j e^{Q(a_j) \cdot \tau^{-1}}} \tag{6}$$

[1] The details of these terms are discussed in section 3.2

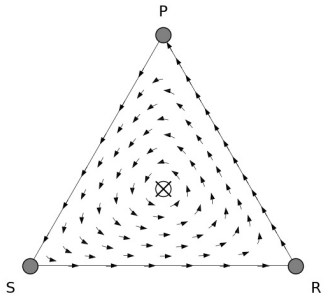
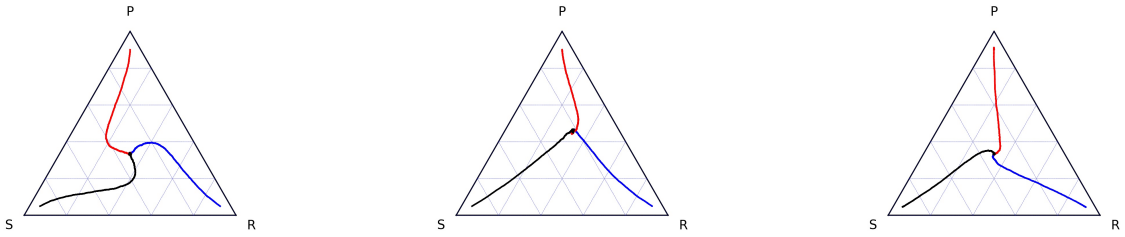



Figure 1: Replicator dynamics of the Rock-Paper-Scissors game

where $\tau$ is the temperature parameter that controls the balance between exploration and exploitation. Figures 2 and 3 depict the learning trajectories of two Q-Learners playing against each other. In particular, figure 2a shows the trajectories of agent 1 when the starting strategy probabilities of agent 2 (figure 3a) are identical to those of agent 1. The learning trajectories of both agents converge to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ Figure 2b shows the trajectories of agent 1 when the starting strategy probabilities of agent 2 (figure 3b) are given by $(R, P, S) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. $\alpha$ has been set to 0 for agent 2, meaning said agent is a *Non-learner*. The learning trajectories of agent 1 end up converging, albeit not exactly to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Figure 2c shows the trajectories of agent 1 when the starting strategy probabilities of agent 2 (figure 3c) are such that agent 1 is more likely to win. More specifically:

$$\begin{cases} (R,P,S)_1 = (0.9, 0.05, 0.05) \longleftrightarrow (R,P,S)_2 = (0.05, 0.05, 0.9) \\ (R,P,S)_1 = (0.05, 0.9, 0.05) \longleftrightarrow (R,P,S)_2 = (0.9, 0.05, 0.05) \\ (R,P,S)_1 = (0.05, 0.05, 0.9) \longleftrightarrow (R,P,S)_2 = (0.05, 0.9, 0.05) \end{cases} \tag{7}$$

The policy trajectories of agent 1 and agent 2 end up converging to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

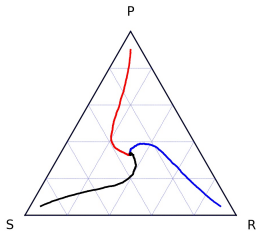



(a) $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$. (b) $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$. (c) $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$.

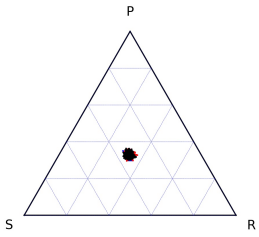
Figure 2: Agent 1: Q-learning policy trajectories of the Rock-Paper-Scissors game using Boltzmann exploration.

## 2.3 Opponent modeling: fictitious play

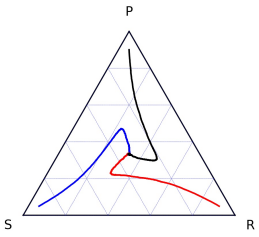
The Q-Learner discussed above improves its behavior based exclusively on its own actions and rewards. Fictitious play is the first step towards a multi-agent system where an agent takes into consideration the agency of other players. By counting the frequencies in which particular actions are chosen by the opponent, we can build a probabilistic model. In the **Rock-Paper-Scissors** game the probabilistic model can be used with payoff matrix (3) to compute the expected utilities of each possible counter [7].




(a) $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$.




(b) $\alpha = 0$.




(c) $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$.

Figure 3: Agent 2: Q-learning policy trajectories of the Rock-Paper-Scissors game using Boltzmann exploration.

**Expected utilities**

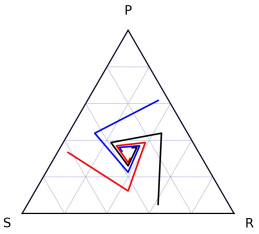
For each agent in the system, the expected utilities are given by:

$$\begin{cases} E(R) = P_{opp}(R) \cdot PO(R,R) + P_{opp}(P) \cdot PO(R,P) + P_{opp}(S) \cdot PO(R,S) \\ E(P) = P_{opp}(R) \cdot PO(P,R) + P_{opp}(P) \cdot PO(P,P) + P_{opp}(S) \cdot PO(P,S) \\ E(S) = P_{opp}(R) \cdot PO(S,R) + P_{opp}(P) \cdot PO(S,P) + P_{opp}(S) \cdot PO(S,S) \end{cases} \tag{8}$$

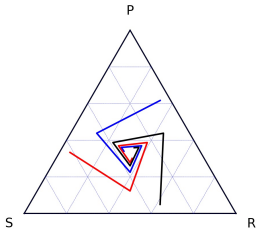
where $E(R)$ is the expected utility of playing **Rock**, $P_{opp}(R)$ is the probability of the opponent playing **Rock**, and $PO(R,R)$ is the payoff matrix value corresponding to the agent playing **Rock** and the opponent playing **Rock**. The agent's actions are determined by the value of the expected utilities. Each turn it selects the action producing the highest expected utility. The Nash equilibrium is reached for $P_{opp}(R) = P_{opp}(P) = P_{opp}(S) = \frac{1}{3}$

**Learning trajectories**

In order to make out the fictitious player's behavior, two scenarios have been tested. Two fictitious players are first paired with one another. A fictitious player is then paired with a Q-Learner. The trajectories obtained when facing two fictitious players against each other are depicted in figure 4. Prior to the first game played by the *blue* agents, both agents are informed that their respective opponent is currently using the following strategy layout: $(R, P, S) = (0.9, 0.05, 0.05)$. Both agents end up reacting the same way and quickly converge to the Nash equilibrium $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. A fictitious player may furthermore be paired with a Q-Learner. The




(a) Fictitious player a.



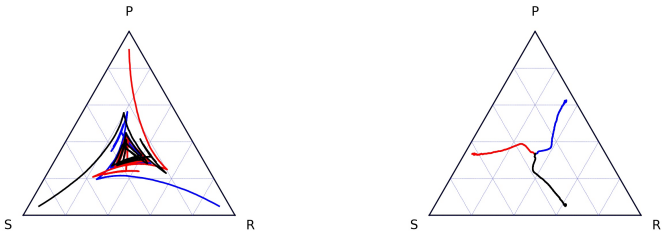

(b) Fictitious player b.

Figure 4: Trajectories of the Rock-Paper-Scissors game played by two Fictitious learners

corresponding trajectories are presented in figure 5. Similarly to the situation presented in section 2.2, the Q-Learner has different starting probabilities.
Given the fictitious learner's properties, its trajectories *react* to the Q-Learner's trajectories by tending to opt for the better strategy. As an example, when the Q-learner initially picks **Rock**, the Fictitious learner will tend to play **Paper** or **Rock** (see blue trajectory).

## 2.4 Combining Q-learning and opponent modeling

This section considers a combination of Q-learning and opponent modeling. Instead of purely reacting to the opponent's strategy as could be observed by the fictitious player when paired with a Q-Learner in section 2.3,




(a) Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using explore with Boltzmann exploration.

(b) Fictitious player

Figure 5: Trajectories of the Rock-Paper-Scissors game played by a Q-Learner and a Fictitious learner

or not considering the opponent at all (Q-learning), one may consider a combination of both learning settings in an effort to produce a more robust algorithm. This section explores a learning setting called *Joint-Action Learning (JAL)*. As with fictitious play, the player picks an action based on the expected utilities of each action. The expected value of an action is given by the sum of joint Q-values, weighed by the estimated probability of the associated complementary joint action profiles [8]. In order to balance exploration and exploitation, the Boltzmann exploration mechanism is used again here, the Q-values being replaced by the Expected utilities. An action is selected with probability:

$$p(a_i) = \frac{e^{E(a_i)\cdot\tau^{-1}}}{\sum_j e^{E(a_j)\cdot\tau^{-1}}} \tag{9}$$

**Expected utilities**

The expected utilities of an action are given by [8]:

$$E(a_i) = \sum_{a_{-i}\in A_{-i}} Q(a_i \cup a_{-i}) \cdot P_{opp}(a_{-i}) \tag{10}$$

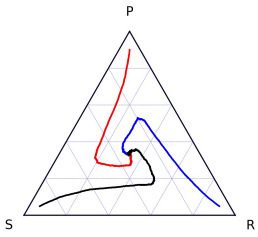
where $a_i$ is the action of agent $i$, $a_{-i}$ is the complementary joint action profile, and $P_{opp}(a_{-i})$ is the probability of the opponent playing action $a_{-i}$. The Q-value is updated using an equation similar to equation 4. In this case, the Q-value does not only depend on the value of the agent's action but on the value of the opponent's action as well.

**Learning trajectories**

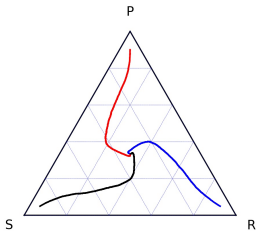
A fictitious Q-Learner is paired with a Q-Learner, a fictitious learner and another fictitious Q-Learner in figures 6, 7 and 8 respectively. Figure 6 shows both learners converging to the Nash equilibrium with the fictitious Q-Learner converging faster. Figure 7 shows the trajectories of the fictitious Q-Learner matched against a fictitious player. The fictitious learner quickly adapts to the fictitious Q-learner by learning a strategy that counters its opponent's strategy best. Both learners end up converging to the Nash equilibrium. Finally, figures 8a and 8b show the trajectories of two fictitious Q-Learners playing against each other when their starting probabilities are the same. Figure 8c and figure 8d show the trajectories of agent 1 when the starting strategy probabilities of agent 2 are such that agent 1 is more likely to win, see (7).

## 2.5 Concluding thoughts and commonalities between trajectories

Different learners have been matched against one another, and the combinations produce various results. In particular, the ability of the fictitious Q-learner to *read* the opponent while performing Q-learning appears to allow for faster convergence to the Nash Equilibrium $(R, P, S) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, making it the better performer overall. Despite the differences between learners, they do possess one commonality, the fact that they all end up opting for a mixed strategy.

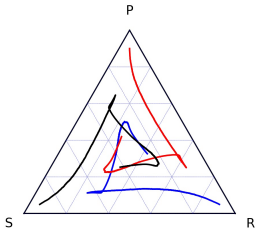



(a) Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.

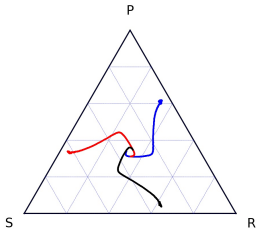



(b) Fictitious Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.

Figure 6: Trajectories of the Rock-Paper-Scissors game played by a Q-Learner and a Fictitious Q-Learner.

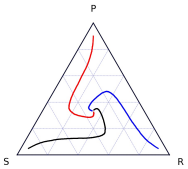



(a) Q Fictitious Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.

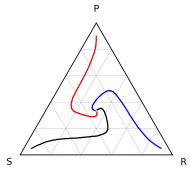



(b) Fictitious learner.

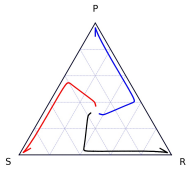
Figure 7: Trajectories of the Rock-Paper-Scissors game played by a Fictitious Q-Learner and a Fictitious learner.



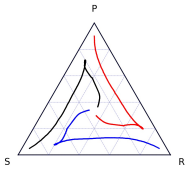

(a) Fictitious Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.




(b) Fictitious Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.




(c) Fictitious Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.




(d) Fictitious Q-Learner with $\alpha = 5 \cdot 10^{-5}$ and $\gamma = 0.9$ and using Boltzmann exploration.

Figure 8: Trajectories of the Rock-Paper-Scissors game played by two Fictitious Q-Learners.

# 3 Task 3: Harvest Game Agent

## 3.1 Problem Statement

The final task concerns itself with the *Harvest Game*. At an abstract level, the main difference between this game and the ones explored during tasks 1 and 2 resides in its **stateful** nature. Due to the complexity of its *state-action* space, a result of multiple actors (agents and apples) involved in it, learning the corresponding Q-function on a state-action table would require a lot of time and computing power. An efficient way of circumventing this issue is made possible through the use of a *deep neural network* trained to approximate the aforementioned Q-function.

## 3.2 Deep Q-Networks

**Working principles: How can deep learning be used in a multi-agent reinforcement learning environment to approximate the optimal action value function Q?**

Deep Q-learning combines reinforcement learning, in particular Q-learning, and deep learning [9]. As schematized in figure 9, 2 major components characterize reinforcement learning: the *agent* and the *environment* [6]. The agent *acts* upon the environment given a *state*, which results in the environment returning a new state as well as an associated reward. The function that models this agent-environment interaction is given by (4). The complexity of the state-action space in the *Harvest Game* makes learning this Q-function tiresome. The regular Q-Learner may however be replaced by a deep neural network that is trained to approximate said Q-function.

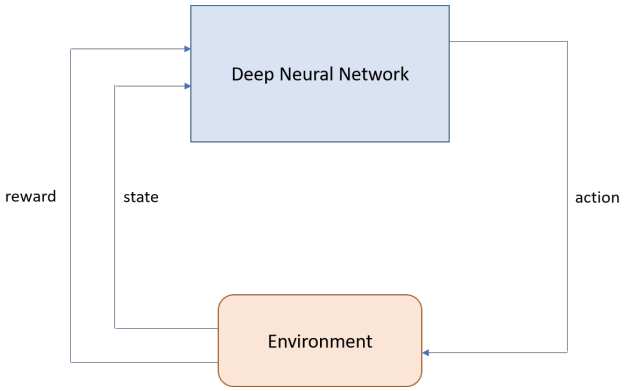



Figure 9: Reinforcement learning, the agent is represented by a deep neural network

Several changes to the aforementioned Q-function are necessary in order to accommodate the particular deep learning multi-agent setting of the *Harvest Game*. In fact, at any given state, each agent's *view* is limited to a $15 \times 15$ window centered on their current location. Let $\mathcal{O}_i$ be the observation space of agent $i$ [10]. It represents the space that for each actual game state contains the corresponding observation for agent $i$. $\Theta$ denotes the deep learner parameter space. The $i$-th agent's deep network function has the signature $Q_i : \mathcal{O}_i \times A_i \times \Theta \rightarrow \mathbb{R}$ and is given by:

$$Q_i(o_i, a; \theta^{new}) \leftarrow (1-\alpha)Q_i(o_i, a; \theta) + \alpha(r_i + \gamma \, max_{a' \in A_i} Q_i(o_i', a'; \theta)) \tag{11}$$

where $Q_i(o_i, a, \theta)$ is an approximator of the optimal action value function $Q_i^*(o_i, a)$ [11]. The key functions of the Q-Learner `save_experience(state, action, reward, next_state, game_over)`, `choose_action(state)` and `learn()` can be described as follows:

- `save_experience(state, action, reward, next_state, game_over)`: The deep Q-Learner maintains a `memory` of *experiences*, which are tuples of the form $< o_i, a, r_i, o_i', game\_over >$. These experiences are sampled during the *replay* phase (see `learn()`) by the Q-Learner for it to be trained.
- `choose_action(state)`: [10]

$$\pi_i(o_i, \theta) = \begin{cases} argmax_{a \in A_i} Q_i(o_i, a; \theta), & \text{with probability } 1-\epsilon \\ random(A_i), & \text{with probability } \epsilon \end{cases}$$

  where $\epsilon$ is the $\epsilon$-exploration parameter.
- `learn()`: The `memory` is first sampled, producing a `batch` of experiences. The Q-Learner then updates its policy according to equation (11) by looping through each $< o_i, a, r_i, o_i', game\_over >$ batch-experience.

It should be noted that this particular algorithm considers agents to be **independent** from one another ([10], section 3.1). In fact, given an initial observation $o_i$, agent $i$ may act upon the environment, resulting in observation $o_i'$. This new observation $o_i'$ does not correspond to the 'state' directly following the agent's action, but instead regroups each agent's action following agent $i$'s up to agent $i$'s turn. In other words, each agent makes no reasoning about the other agents' learning, and simply considers them as being part of the environment.

**How is a balance between Exploration and Exploitation obtained using $\epsilon$-decay?**

Exploration is defined as the phase where the agent gathers information that may help it make better decisions at a later stage. This is achieved by having the agent pick random actions. Exploitation is defined as the phase where the agent picks the best action given its current knowledge about the game. The amount of exploration is determined by the $\epsilon$ variable. In order to allow the agent to quickly get accustomed to the environment it interacts with, it will initially mostly explore said environment. As its knowledge base increases and the neural network has had a fair amount of training, we would like the agent to increasingly exploit the environment, for us to be able to gauge its performance. The $\epsilon$-decay variable is used for this purpose specifically. A high-level overview of the agent's main loop is given in listing 1 [9].

Listing 1: Agent: main loop

```
while True:
    while current game is not over:
    do
        state, reward = environment.get_response()
        save_experience(previous_state, previous_action, reward, state)
        action = choose_action(state)
        previous_state = state
        previous_action = action
    learn()
    epsilon *= epsilon_decay if epsilon has not reached its min value
```

**Agent layout: What characteristics is a Convolutional Neural Network used to learn the *Harvest Game* expected to display?**

Several deep learning layouts may be used to model the agents. The current version of the Q-Learner makes use of a *Convolutional Neural Network (CNN)* implementation. The summary of the model is given in listing 2. The last layer has 4 neurons, one for each possible action (*move, left, right, fire*). Layers `dense_4` and `dense_5` both use the *rectified linear unit (ReLU)* activation function. The input size of the network is equal to the size of the feature vectors that describe the game observations.

Listing 2: Deep Q-learning CNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 64)                768
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_6 (Dense)              (None, 4)                 260
=================================================================
Total params: 5,188
Trainable params: 5,188
Non-trainable params: 0
_________________________________________________________________
```

It should be noted that the Q-function can be parameterized in two ways:

$$o_i \xrightarrow{DQN} (Q_{o_i,a_1}, Q_{o_i,a_2}, Q_{o_i,a_3}, Q_{o_i,a_4}) \tag{12}$$

or:

$$(o_i, a_j) \xrightarrow{DQN} (Q_{o_i,a_j}) \quad \forall j \in \{1,2,3,4\} \tag{13}$$

Approach (12) was chosen in the framework of this project. In fact, as suggested by *DeepMind* in [9], the advantage of this architecture is the fact that the Q-values can be computed for all actions in a single forward pass, as opposed to four, thereby reducing the computation cost.

**Extracting features from the environment: Which features are interesting in the *Harvest Game* context and how may they be represented in a meaningful way based on their relative importance?**

Communication between agent and environment happens by means of *websockets*, opened by the agent on a given port. An action is sent to the game application in a specific format for it to be understood by said application. Similarly, the application sends back messages to the agent informing it of the current game state as seen by said agent, i.e. within the constraints of its $15 \times 15$ window. The raw messages, that contain information such as the whereabouts and orientation of all players as well as the location of apples that lie within the $15 \times 15$ window, sent to the agent may now be transformed into feature vectors, that can then be fed to the Deep Neural Network. Ideally the feature vector should contain as much information as possible while simultaneously being as small as possible. This allows uninteresting information to be avoided while designing the features. The following information is present in the current version of the feature vectors:

1. Distance to the closest apple in front
   Distance to the closest apple to the left
   Distance to the closest apple to the right
2. Number of enemies currently within the $15 \times 15$ frame
3. Whether or not the agent is currently winning by score
4. Whether an enemy is right in front of the agent
   Whether an enemy is to the left of the agent
   Whether an enemy to the right of the agent
   Whether an enemy is behind the agent

   In the event that an agent is hit by an opponent, it is punished with a score of -50. Information about the enemies surrounding the agent should help said agent plan its next move accordingly.
5. Number of apples within the $15 \times 15$ frame
6. Whether or not there is an apple right in front of the agent
   Whether or not there is an apple to the right of the agent
   Whether or not there is an apple to the left of the agent

   Information about the apples directly surrounding the agent should help said agent plan its next move accordingly in order to maximize its rewards.

The values are normalized to values $\in [0, 1]$, before weights $w_i$ are individually applied to some of the components to increase or reduce their priority over other components. As such, component **3.** is given a weight of 2, while components **6.** are given a weight of 3.5. Components **1.** are given a weight of 0.5.

## 3.3 Experiments

**Training the DQN: Parameter settings**

The *replay memory* was set to a size of 50000. After each iteration of the game, a sample of 5000 experiences $< o_i, a, r_i, o'_i, game\_over >$ is taken from said replay memory and fed to the DQN. The exploration parameter $\epsilon$ is initially set to 0.9, the $\epsilon$-decay parameter is set to 0.9, and $min(\epsilon) = 0.01$.
The model has been trained with a learning rate $\alpha$ of $5 \cdot 10^{-6}$, and a discount factor $\gamma$ of 0.85.
The number of agents has been set to 4 for each iteration of the game, the apple growth rate has been set to 3. The evaluation of the DQN following 35 training iterations is done using different values for the number of agents and the apple growth rate.
Every player in the game uses the same agent, i.e. the memory keeps track of the experiences of all players , and as a consequence experiences from all players are sampled each iteration to train a single model.

**Training the DQN: How can the metrics tracked by `keras` be used to measure training progress, and how effective are they?**

To make sure a deep neural network is being trained properly, the `keras` library allows to track several metrics during the training process. In particular, the *mean square error (MSE)* is the training metric used by the

DQN during training to update its weights. More specifically, the learning algorithm implemented uses the *one-step Q-learning* approach [11], in which a sequence of loss functions is iteratively minimized during any given training cycle, i.e. the loss function of agent $i$ during the replay of experience $j$ is defined as follows [9]:

$$L_i(\theta_j) = (\underbrace{r_i + \gamma \, max_{a' \in A_i} Q_i(o_i', a'; \theta_{j-1})}_{\text{Target value } (\alpha = 1)} - \underbrace{Q_i(o_i, a; \theta_j)}_{\text{DQN prediction}})^2 \tag{14}$$

The loss values of any given training cycle are then averaged, and the average loss as a function of the number of training iterations is plotted in figure 10. Although there seems to be a slight downward trend when training the model for 35 iterations, the loss function appears to be a poor indicator of model performance by itself. This may be correlated with the random nature of the samples drawn from the memory. A more stable metric of model performance, the final score of all players, is discussed hereafter.

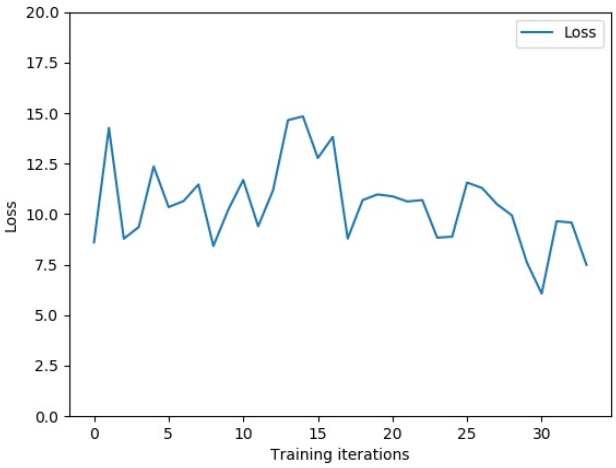



Figure 10: Average loss as a function of the number of training iterations

**Training the DQN: Can the final agent score per training iteration be used to track training progress in a stable way?**

Following an evaluation approach similar to [9], the final score of player all players is plotted with respect to the number of training iterations, as illustrated in figure 11. Several pieces of information may be extracted from these results:

- The agents appear to favor *cooperating* with the other players, by trying to maximize their own score instead of firing at the other players. This is likely due to several factors:
  - A player is punished with a score of -1 each time it fires.
  - The information present in the feature vector, in particular *whether or not the agent is currently winning* (see section 3.2), may not be tempting enough for a player to want to tag another player.
  - The growth rate was set to 4, and thus rapidly filled the entire map with apples. There was likely no incentive for a player to reduce the other players' scores given the abundance of apples.
- The player scores appear to level out at about 800 after approximately 20 training iterations, each game lasting 1000 steps. Some factors that contribute to the approximately 200 steps without rewards are:
  - The $15 \times 15$ window centered on each player makes it difficult for said players to notice the apples right away, especially during the first steps of the game.
  - The presence of multiple agents on the map: a player might 'steal' apples from another player.

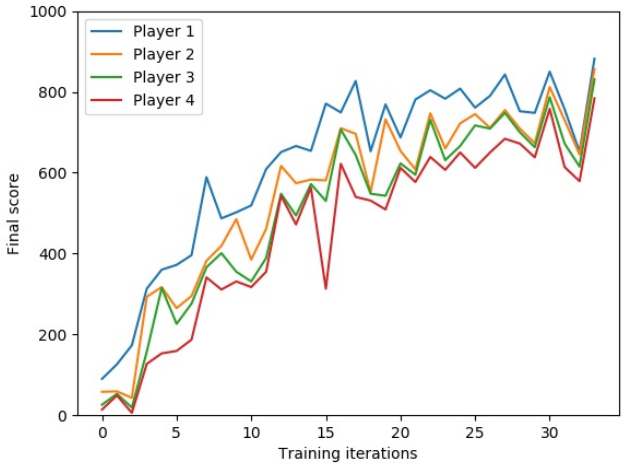



Figure 11: Final score of each player as a function of the number of training iterations

**Model behavior following 35 training iterations**

Several observations and observations can be drawn from testing the trained model under various conditions:

- Different growth rates and numbers of players have not affected said players' *aggressiveness* during the test runs, i.e. players do not resort to the *fire* command. This is in line with the reasoning made in the previous subsection. A game similar to the *Harvest Game*, the *Gathering Game* is evaluated in [10]. In this paper, the aggressiveness of players, i.e. their frequency of firing, is found out to be a function of the re-spawn time apples and the re-spawn time of enemies after tagging them. The second argument differs considerably from the way the *Harvest Game* is designed. Players are not temporarily removed from the game by getting tagged in the *Harvest Game*. Furthermore, unlike in the *Gathering Game*, they are punished when firing. This is likely reason enough for the agents to refrain from firing. By considering the *Harvest Game* to be a social dilemma as was done in [10], and using the frequency of firing as a tool to determine whether the agents are following a *defecting* or *cooperative* policy, the agents would fall in the cooperative category.

- The agents appear to be shortsighted, in that they regularly miss apples that are within their field of view, but are not right next to them. The main contributing factor is likely to be the weight of 0.5 given to components **1.** as discussed in section 3.2.

- The agents have a tendency of moving straight ahead until they sense the presence of nearby apples. They appear to occasionally move out of the line of fire of enemy players. This is likely a consequence of the early experiences drawn from the memory and replayed, as the higher $\epsilon$ values forced the players to occasionally fire at their opponents.

- The agents tend to perform well once they reach a *pile* of apples, or more specifically, a sequence of apples that can be traveled without any major *holes* in between.

# 4 Conclusion

Several factors influence and need to be considered in multi-agent reinforcement learning: the game of choice may be stateless or stateful, a player can take other players' agency into account (JAL) or consider them to be part of the environment (independent learning). Evolutionary game theory may help in understanding and predicting the dynamics of learning in 2 player, 2/3 strategy games. More complex and stateful games such as the *Harvest Game* require the learning models to generalize in order to make the learning process productive. Various machine learning techniques, e.g. deep learning, may assist in approximating state-action functions.

# Bibliography


[1] Daan Bloembergen. Analyzing reinforcement learning algorithms using evolutionary game theory, 06 2010. http://michaelkaisers.com/publications/2010_MT_Bloembergen.pdf.

[2] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38:156 – 172, 04 2008. http://busoniu.net/files/papers/smcc08.pdf.

[3] Karl Tuyls, Julien Pérolat, Marc Lanctot, Georg Ostrovski, Rahul Savani, Joel Z. Leibo, Toby Ord, Thore Graepel, and Shane Legg. Symmetric decomposition of asymmetric games. *CoRR*, abs/1711.05074, 2017. https://arxiv.org/pdf/1711.05074.pdf.

[4] Egtplot: A python package for 3-strategy evolutionary games. https://github.com/mirzaevinom/egtplot.

[5] Hendrik Blockeel. Machine learning and inductive inference, course notes. KU Leuven, 2018.

[6] Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. Evolutionary dynamics of multi-agent learning: A survey. *Journal of Artificial Intelligence Research*, 53:659–697, 08 2015. https://jair.org/index.php/jair/article/view/10952/26090.

[7] Mike Goodrich. Fictitious play. https://students.cs.byu.edu/~cs670ta/FictitiousPlay.html, 09 2004. Accessed: 2019-04-30.

[8] Gerard Vreeswijk. Mutli-agent learning, course notes. Utrecht University, 2010. http://www.cs.uu.nl/docs/vakken/maa/current/slides_old/MAA_slides_Marl.pdf.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. https://arxiv.org/pdf/1312.5602.pdf.

[10] Joel Z. Leibo, Vinícius Flores Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, abs/1702.03037, 2017. https://storage.googleapis.com/deepmind-media/papers/multi-agent-rl-in-ssd.pdf.

[11] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. https://arxiv.org/pdf/1602.01783.pdf.