KU LEUVEN

Machine Learning Project

Report 1

Yannou Ravoet, r0637112

Robrecht Peeters, r0627461

2019 – 2020

1 Introduction

In this report we take our first steps to combining the fields of Reinforcement Learning and Game Theory in a Multi-Agent setting. We look at the matrix games known as the prisoner's dilemma, matching pennies game, battle of the sexes, rock paper scissors. The payoff matrices for each game are visualized in Appendix A. In section 2 we go over the Nash equilibria and Pareto optimal states for each game. Section 3 discusses the implemented independent learning algorithms and population dynamics. Section 4 summarizes the literature that was consulted, and Section 5 draws a conclusion of the report. Section 6 makes an estimate as to which algorithms might be interesting for the next challenge: Kuhn and Leduc Poker.

2 The games

A ***Nash equilibrium*** occurs when both players chose the best response given that they know the action of the other players. In other words, none of the players can react better given that the other players do not change their strategy. We make a distinction between a *pure strategy Nash equilibrium*, wherein each player chooses a single action deterministically, and a *mixed strategy Nash equilibrium* where (at least one of) the players uses a probability distribution over the actions. A state (a selection of actions for each player) is ***Pareto optimal*** if none of the players can receive a better reward without diminishing another player's reward. Intuitively, this corresponds to a state in which a player could convince all other players to change state (since the new state is either better or equally as good for them).

In the prisoner's dilemma there is only 1 Nash equilibrium since both players have a strictly dominating strategy: 'defect'. The only Pareto Optimal state is when both players 'cooperate'. However, since this is not the dominating strategy, this is not a Nash equilibrium. There is thus but 1 Nash equilibrium.

In the matching pennies game, we are dealing with a competitive zero-sum game where one player's increase his reward corresponds to the other player's decrease in reward. There is only a mixed strategy Nash equilibrium where both players choose 'heads' or 'tail' with a 50-50 probability distribution. Each player then loses as much as they win, resulting in a reward of 0. By definition, each state of a zero-sum game is Pareto-optimal since there is no way to increase a player's reward without diminishing another players reward. There is thus but 1 Nash equilibrium.

In the coordination game battle of the sexes, there are 2 pure strategy Nash equilibria: if both players choose 'movies' or both players choose 'ballet'. There is also a mixed Nash equilibrium where each player chooses his/her preferred activity 2/3's of the time and the other activity 1/3 of the time. This leaves each player with an average reward of $(2/3 * 2 * 1/3 + 1/3 * 1 * 2/3) = 2/3$. Both pure strategy Nash equilibria are Pareto-optimal. There are thus 3 Nash equilibria.

In rock paper scissors, there is no pure strategy Nash equilibrium, since none of the state consists of a best response for both players. However, the players can choose each action 1/3 of the time to achieve the mixed Nash equilibrium with a reward of 0. Again, since this is a zero-sum game, each of the 9 states is Pareto-optimal. For the biased version, we chose to set a win by paper = 3, by scissors = 2 and by rock = 1. This moves the mixed strategy equilibrium to $p(rock) = \frac{1}{3}, p(paper) = \frac{1}{6}, p(scissors) = \frac{1}{2}$ with an expected payoff of 0. There is thus but 1 Nash equilibrium.

The fact that we only get odd numbers of Nash equilibria, is due to the *oddness theorem* stating that nearly all finite games have an odd and finite number of equilibria. If not, this is usually due to a weakly dominant strategy (a strategy that is not always better, but never worse).

3 Independent learning algorithms and population dynamics

3.1 Independent learning

Q-Learning is used in single agent Markovian environments[1] to maximize an agent's reward (payoff in game theory). However, in a multi-agent environment the future state is also influenced by the actions of other agents (for which the agent (initially) has no probability distribution). Therefore Q-learning is no longer guaranteed to converge to optimal q-values. The traditional Q-learning update equation updates the q-values of the selected action i, based on its reward and (discounted) future potential. A side effect in one-shot games, is that the *influence of future rewards is eliminated*.

$$\mathrm{Q_i(t\ +\ 1)\ \leftarrow\ Q_i(t)\ +\ \alpha\left(\ r_{i(t)}\ \sout{+\ \gamma\ \mathrm{argmax_j Q_j(t)}}\ -\ Q_i(t)\right)}$$

The update rule is clearly independent of the agent's policy. The two policies we implemented are the epsilon-greedy and Boltzmann exploration scheme. The *epsilon-greedy exploration* scheme selects the best-known action with a percentage $(1-\epsilon)$ and a random action with a percentage ϵ. The *Boltzmann exploration* scheme will instead use a temperature value τ to define a probability distribution over the possible actions (higher τ = more exploration). Thus, actions with a sub-optimal reward are not simply ignored $(1-\epsilon)$ percent of the time, but rather each action is picked with a frequency in line with its expected reward. A side effect of this τ parameter, is that the agent can never quite reach a pure strategy Nash equilibrium, since we cannot set $\tau = 0$ (divide by zero) and there is thus always some randomness left.

$$x_i(\mathrm{Q},\tau) = \frac{e^{\tau^{-1}Q_i}}{\sum_{j\in LegalActions} e^{e^{\tau^{-1}Q_j}}}$$

Note that *mixed strategy Nash equilibria are never achieved* (even with an exploration rate of 0) since agents are constantly changing their q-values (with a probability p or $(1-p)$ the agent will lose/win and adapt their q-values accordingly, leaving the equilibrium) and these equilibria are thus unstable. The only way to reach a mixed strategy Nash equilibrium would be to set the learning rate at 0 the moment both agent arrive in a mixed strategy equilibrium (note that some of these equilibria cannot even be represented by a computer: i.e. $p = 1/3$).

If we train the agents in an *iterated play environment*, there is a higher chance of them hovering around the mixed strategy equilibrium (especially where those equilibria offer a higher reward for a player (such as in matching pennies), since training episodes containing multiple iterations average out the wins and losses (in one-shot learning, a couple of wins in row, could drive the q-values toward a pure strategy Nash equilibrium). Similarly, there is also a higher chance to end up in a Pareto optimal state in a coordination game (for example the 'stag'-'stag' state in the stag hunt game), since when we learn over multiple iteration, the other agents action choices are learned as part of the environment. In this way, iterated play *mimics coordination* between players.

Frequency Adjusted Q-learning (FAQ) tries to compensate for over-selection of a dominant action (due to the action having a higher selection probability) by dividing the learning rate in the update rule by the probability of the chosen action. This mimics a simultaneous update of all actions as would be the case in population dynamics. An extra factor β is also used to ensure we never reach a learning rate > 1, which could lead q-values to become higher than the actual rewards. We can intuitively consider β as a limitation on the range of the policy space on which FAQ will act. As such, a lower β-value ensures convergence over a bigger percentage of the policy space.

$$\mathrm{Q_i(t\ +\ 1) \leftarrow Q_i(t) +\ \min\left(\frac{\beta}{x_i}, 1\right) *\cdot\ \alpha\ \left(r_i(t)\ \sout{+\ \gamma\ \mathrm{argmax_j\ Q_j(t)}}\ -\ Q_i(t)\right)}$$

Lenient Frequency Adjusted Q-learning (LFAQ) tries to introduce a concept of leniency towards other agents. This is useful in coordination-games, where the previous Q-learning algorithms are susceptible to ending in a suboptimal state due to early action choices that do not lead to this Pareto optimal state (miscoordination). The simplest way to implement this leniency towards other agents, is to evaluate actions during κ iteration before training on the iteration that resulted in the best reward (which will often be the Pareto optimal state). For

[1] In a Markovian environment future states are independent of past states given the present state. Thus, if an agent performs action a in state s_t, it will always have the same probability of arriving in state s_{t+1} no matter the states $s_{t_i<t}$.

example, in the stag hunt, there is a high chance that if we evaluate the agent's actions 10 times, a state will occur where both actions choose 'stag'. This is then the best reward and this action is used to update the q-values of both agents. The probability of choosing stag thus rises for both agents. Repeating this procedure leads to the Pareto optimal state of 'stag' - 'stag'. We can intuitively feel that in later training episodes the probability of an optimal choice of action by both players rises, and thus the value κ can decrease to increase training efficiency.

The hyperparameters are valued as follows: a lower learning rate results in slower convergence (and thus $lr = 0.001$ was used for cleaner plots). The initial exploration rate was usually set around 0.6 with an exploration annealing rate of 0.99 or 0.999 and a minimum exploration rate of 0.001. A small β-value (0.001) was usually used to ensure convergence over a large part of the policy state. Finally, κ was usually set between 5 and 10 to ensure the best action combination was often used for training and not have to train over too may iterations. The learning rate for LFAQ was also brought up to around 0.5 since most often, the best action would be chosen. Appendix D goes over the code implementation of each of the algorithms discussed above. Appendix C shows the learning trajectories for each of the learning algorithms on all the 2x2 matrix games.

Figure 1 shows the learning trajectory of independent learner overlaid on their corresponding population dynamics for the battle of the sexes and matching pennies game. We can see that the independent learner follows the same general trend as the population dynamics. FAQ learning makes the independent learner follow the population dynamics more closely. The LFAQ learner results in a mixed strategy Nash equilibrium for the Battle of the Sexes game and more directed trajectory towards the resulting end state.

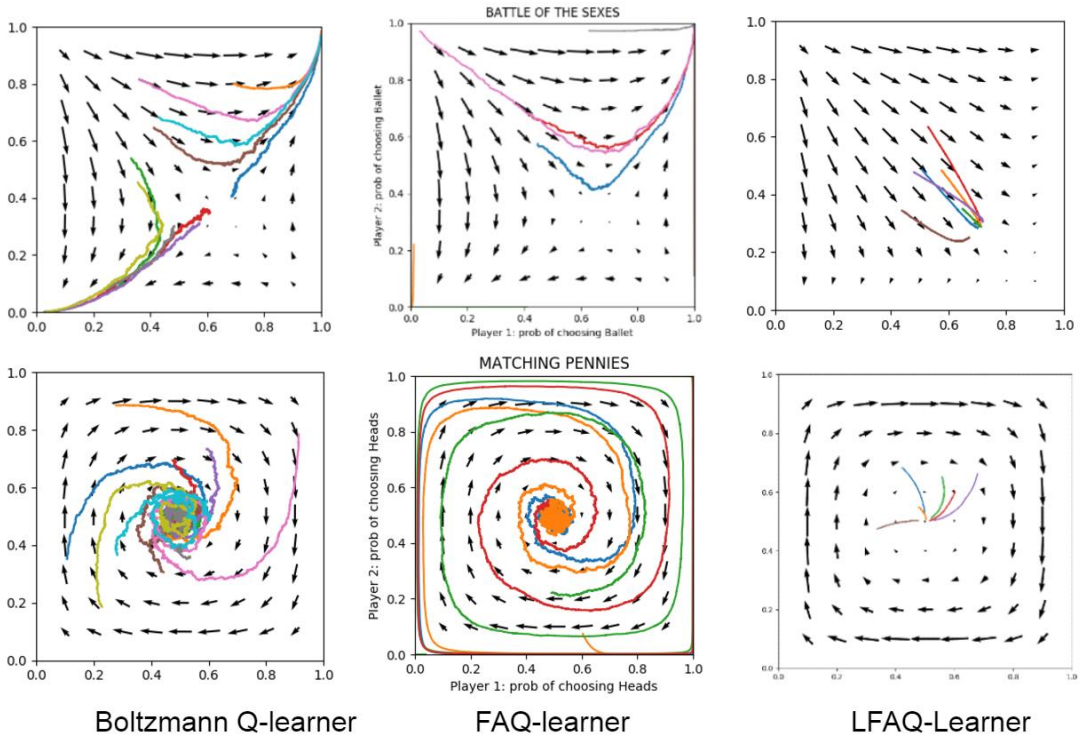

Figure 1: *trajectory plot on top of a phase plot for each independent learning implementation and its corresponding dynamics for the battle of the sexes and matching pennies games.*

3.1 Population dynamics

Dynamics are used to model a change in policy in populations through time. The intuition behind ***Replicator Dynamics*** *is* that policies that are more present (a bigger population) or are simply better (a higher reward) have a higher chance of surviving. In a game an action i will thus increase more if the probability x_i of choosing action i rises or if the reward (or fitness) f_i of action i rises relative to the average reward $\bar{f}$ of all possible actions. For matrix games with payoff matrices A and B for player 1 and 2 respectively, this can be expressed as follows:

$$\dot{x}_i = x_i[(Ay)_i - x^T Ay] \quad \text{and} \quad \dot{y}_i = y_i[(x^T B)_i - x^T By]$$

Boltzmann Q-learning dynamics describe the change in policy of Q-learning agents with a Boltzmann exploration scheme and an infinitesimally small learning rate. The dynamics can be divided in an exploitation and exploration part, where the exploration part can be further subdivided in the exploration of a single strategy and the exploration of the entire policy. In the equation below: i is the chosen action and k is any legal action.

$$\dot{x}_i = \frac{\alpha x_i}{\tau} \underbrace{\left[(\mathbf{Ay})_i - \mathbf{x}^\top \mathbf{Ay}\right]}_{\text{exploitation}} - \alpha x_i \underbrace{\left[\log x_i - \textstyle\sum_k x_k \log x_k\right]}_{\text{exploration}}$$

FAQ-learning dynamics are derived from the FAQ-learning update rule and can be seen as a weighted average between cross learning (replicator dynamics) and exploration.

$$\dot{x}_i = \frac{\alpha x_i}{\tau} \underbrace{\left[(\mathbf{Ay})_i - \mathbf{x}^T \mathbf{Ay}\right]}_{\text{exploitation}} - \alpha x_i \underbrace{\left[\sum_k x_k \ln(\frac{x_k}{x_i})\right]}_{\text{exploration}}$$

In ***LFAQ-learning dynamics*** the fitness function of a population changes (due to playing κ iterations before training). The resulting fitness of a strategy i is expressed as the expected maximum payoff of that strategy. The dynamics update equation is the same as for the FAQ-learning dynamics with a replacement of the fitness function $(\boldsymbol{Ay})_i$ by u_i.

$$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k:A_{ik} \leq A_{ij}} y_k\right)^\kappa - \left(\sum_{k:A_{ik} < A_{ij}} y_k\right)^\kappa\right]}{\sum_{k:A_{ik}=A_{ij}} y_k}$$

Appendix F shows LFAQ plots for each game with different κ-values. Appendix E goes over the code implementation for the different learning dynamics.

4 Literature review

To implement the OpenSpiel library functions, we mostly looked at the examples and algorithm code in the library. The basic concepts of game theory such as payoff matrices, mixed and pure strategy Nash equilibria, Pareto-optimality, strong/weak/payoff/risk dominant strategies and replicator dynamics were mostly found in the introduction sections of [1]–[3] as well as in online resources such as the YouTube series 'Game Theory 101' [4], [5]. The different independent learning algorithms and their corresponding dynamics can be found in the FAQ and LFAQ papers [2], [3], [6]. The most intuitive explanation of the β parameter of FAQ and LFAQ can be found in [7]. For the research on the algorithms we want to implement to solve Kuhn and Leduc Poker we mostly used the 'related work' and 'Poker – Current Methods' sections of [8] and the papers of the discussed algorithms [9], [10].

5 Conclusion

We saw in Figure 1 and Appendix C that although independent learning algorithm are not proven to converge in multi-agent environments (since the Markovian property does not hold), they do often result in the agent learning a Nash equilibrium. We also saw different dynamic equations that simulate a population of independent learning agents. Finally, we noticed that a lenient version of an algorithm can help agents reach Pareto optimal states consistently in coordination games and thus we can consider leniency in an algorithm as a form of communication between the agents, similar to having the agents perform in an iterated play environment.

6 Future work

Kuhn and Leduc poker are incomplete information game and thus we are looking for an algorithm we can run on an extensive-form game, where we can use chance events. Since we need to output a lookup table to compete against the other agents of the course, we decide that the self-play Counterfactual Regret Minimization (CFR) algorithm would be interesting. If a more efficient implementation is necessary (although certainly not likely for Kuhn Poker), we can look at the Monte Carlo variant (MCCFR). Although the strategies from other groups will be static, we cannot train on them, and thus Fictitious Play is unlikely to be a good contender.

APPENDIX A: Payoff matrices for all games

Battle Of The Sexes

	Ballet	*Movies*
Ballet	$(2,1)$	$(0,0)$
Movies	$(0,0)$	$(1,2)$

Staghunt

	Stag	*Hare*
Stag	$(2,2)$	$(0,1)$
Hare	$(1,0)$	$(1,1)$

Prisonner's Dilemma

	Cooperate	*Defect*
Cooperate	$(5,5)$	$(0,10)$
Defect	$(10,0)$	$(1,1)$

Matching Pennies

	Heads	*Tails*
Heads	$(1,-1)$	$(-1,1)$
Tails	$(-1,1)$	$(1,-1)$

Biased Rock Paper Scissors

	Rock	*Paper*	*Scissors*
Rock	$(0,0)$	$(-3,3)$	$(1,-1)$
Paper	$(3,-3)$	$(0,0)$	$(-2,2)$
Scissors	$(-1,1)$	$(2,-2)$	$(0,0)$

Figure 2: *Payoff matrices for all games discussed in the report*

APPENDIX B: Replicator Dynamics phase plots for all games

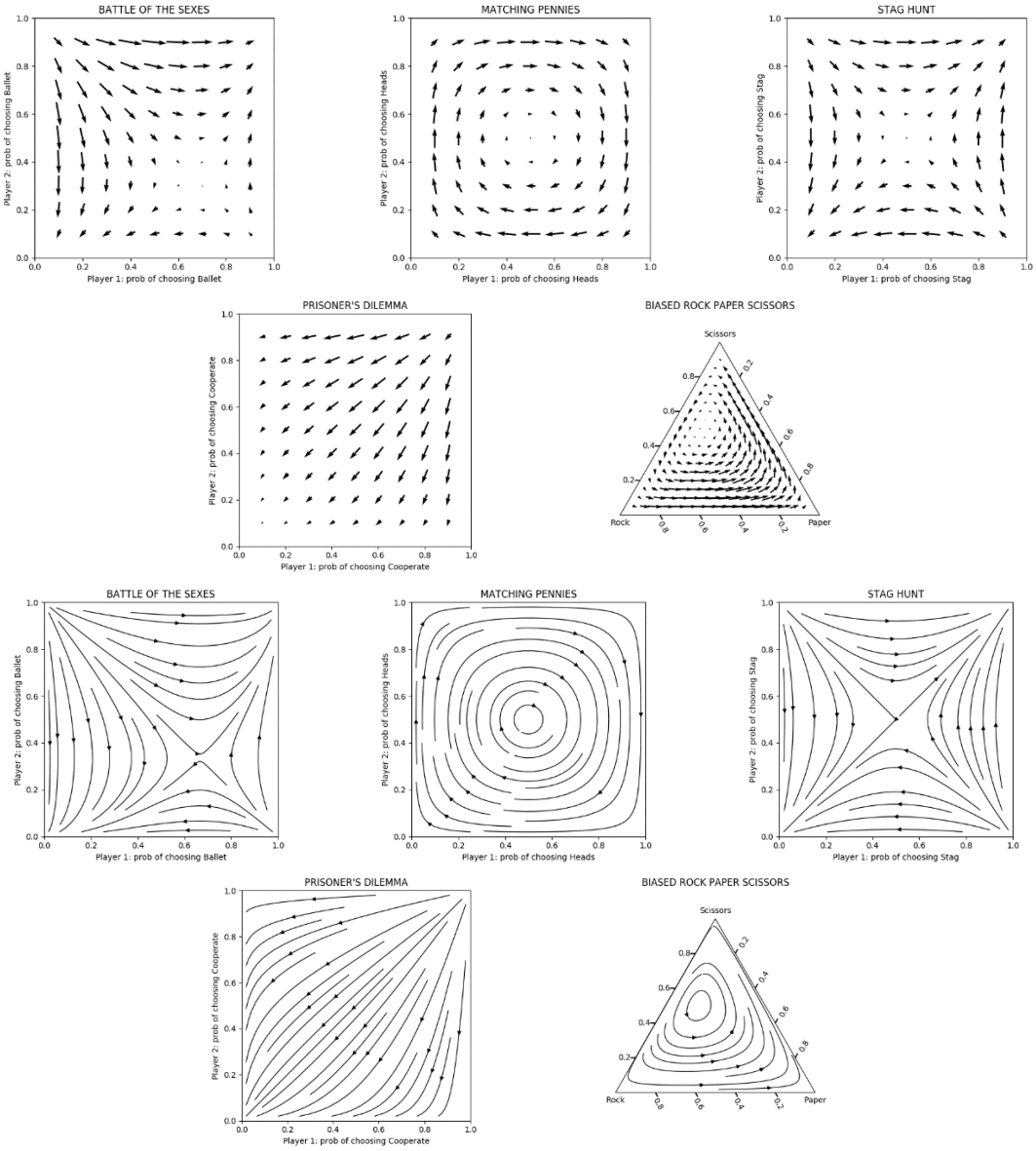

Figure 3: *Phase plot of each game showing the strategy evolution with replicator dynamics.*

APPENDIX C: Trajectory plots of all independent learning algorithms on the corresponding phase plots for the 2x2 matrix games.

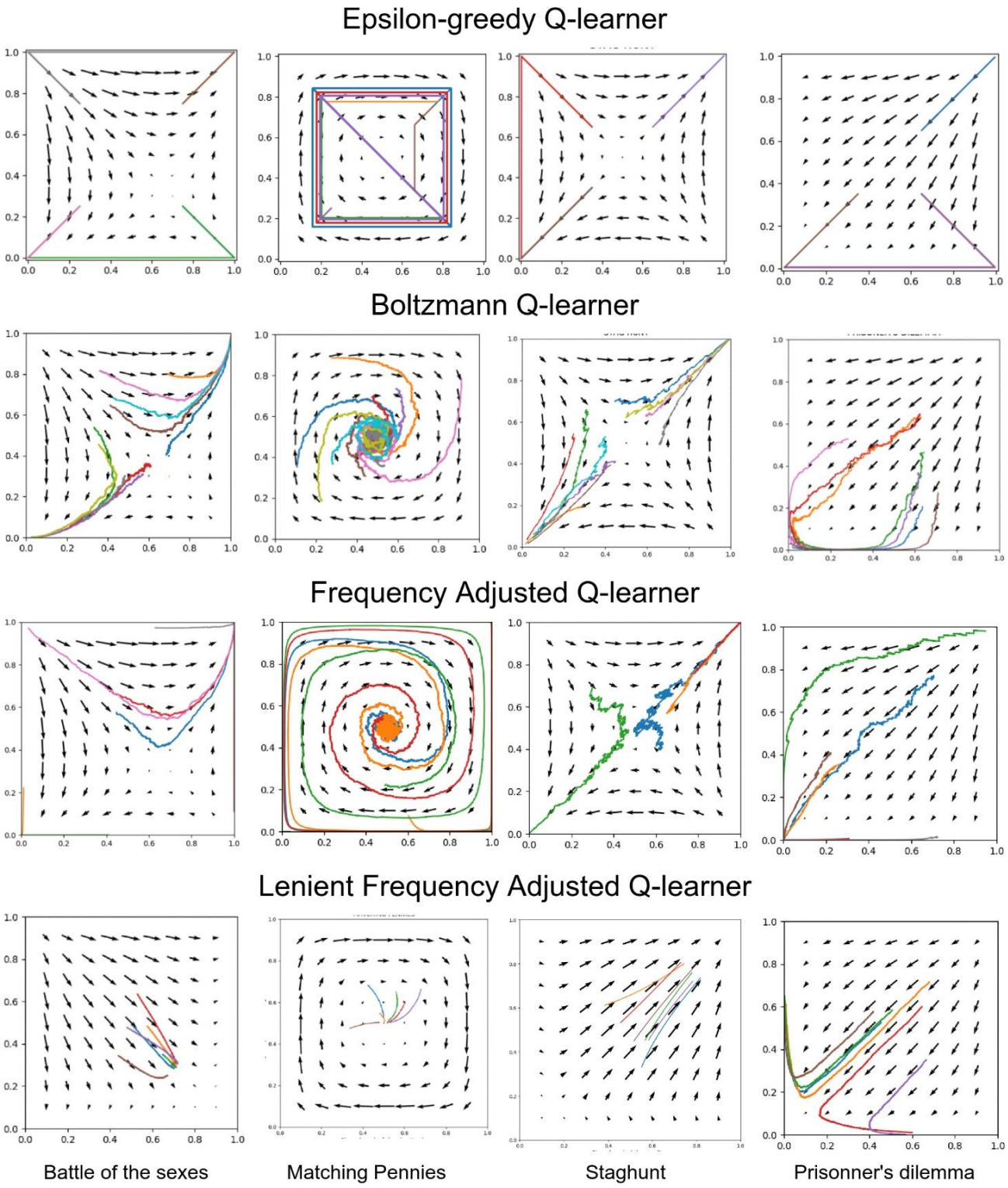

Figure 4: *Trajectory plot for all 2x2 matrix games discussed in the report with on the background their corresponding population dynamics.*

APPENDIX D: Code implementation of the independent learning algorithms

The OpenSpiel library includes an implementation of epsilon-greedy Q-learning. We implemented a child class from this implementation to add exploration annealing and make a change of exploration function easier. This tabular_QLearner class forms the base class for all other algorithms we implemented.

The ***EpsilonGreedy_QLearner*** agents are a copy of the OpenSpiel QLearner with the added option to do exploration annealing.

The ***Boltzmann_QLearner*** changes the exploration function as seen in Figure 3. One interesting problem we came across in the implementation was that for extremely low temperature values (such as t<0.0028), the resulting action probabilities overflow (and thus crash the program). This sets a lower bound temperature value, which was empirically found to be around 0.003.

```
def _exploration_function(self, info_state, legal_actions, temperature):
    #set the probability of non-legal actions at 0
    probs = np.zeros(self._num_actions)
    #softmax the probability of legal actions
    for action in legal_actions:
        probs[action] =  np.exp(self._q_values[info_state][action]/temperature) \
                         /  sum( np.exp(np.array(list(self._q_values[info_state].values()))/temperature) )
    #Choose an action based on the calculated probabilities
    action = np.random.choice(range(self._num_actions), p=probs)
    return action, probs
```

Figure 5: *The python implementation of the Boltzmann exploration function. The Boltzmann_QLearner inherits from the tabular_QLearner.*

The ***Boltzmann_FAQLearner*** agents save the probability of the previously selected action in order to balance the learning rate between actions (actions that are selected less often have a higher learning rate). Instead of only saving the probability of the previous action, we save the output probabilities of the Boltzmann exploration function, since we will need these probabilities in the Boltzmann_LFAQLearner. A β-value is also saved to adapt the training update (so that the learning rate is never greater than 1). The value for β was chosen equal to the learning rate α, if α was small enough. Otherwise, a smaller value of β was chosen to have a more desirable path of convergence. An implementation of the FAQ training update step can be found in Figure 6.

```
def _train_update(self):
    # FAQ: Q_i(t+1) = Q_i(t) + min(B/x_i, 1) * alpha * [ r(t+1) + gamma*max_j Q_j(t) - Q_i(t) ]
    self._q_values[self._prev_info_state][self._prev_action] +=
        (min(self._beta / self._prev_probs[self._prev_action], 1) * self._step_size * self._last_loss_value)
```

Figure 6: *The python implementation of the FAQ update rule. The Boltzmann_FAQLearner class inherits from the Boltzmann_QLearner.*

Finally, the ***Boltzmann_LFAQLearner*** agents have a κ-value and save a list of κ actions, probabilities and rewards. After κ-episodes of evaluation (no q-value updates), the best reward and corresponding action and probability is used to perform one FAQ-learning update on the q-values. The list of κ actions, probabilities and rewards are then emptied, and the process reiterates. As such, more training iterations are required to achieve convergence. A higher learning rate was also chosen, since there is a high chance that the agent choses the optimal action at most of the training iterations. A κ-value around 10 was used and is constant whilst training. As the benchmarked games do not have that many different state-action combinations, this value is likely to contain an iteration with an optimal action choice for both agents. An implementation of the LFAQ training update step can be found in Figure 7.

```
# Learn step: don't learn during evaluation or at first agent steps.
if self._prev_info_state and not is_evaluation:
    self._k_rewards.append(time_step.rewards[self._player_id])
    self._k_actions.append(self._prev_action)
    self._k_probs.append(self._prev_probs)
    if len(self._k_actions) == self._k:
        self._exploration = max(self._exploration * self._exploration_annealing, self._exploration_min)
        best_index = np.where(self._k_rewards == np.amax(self._k_rewards))[0][0]
        self._prev_action = self._k_actions[best_index]
        self._prev_probs = self._k_probs[best_index]
        target = self._k_rewards[best_index]
          ...
        prev_q_value = self._q_values[self._prev_info_state][self._prev_action]
        self._last_loss_value = target - prev_q_value
        self._train_update()

        #reset all k_arrays
        self._k_actions = []
        self._k_rewards = []
        self._k_probs = []
```

Figure 7: *The python implementation of the LFAQ update rule. The Boltzmann_LFAQLearner class inherits from the Boltzmann_FAQLearner.*

APPENDIX E: Code implementation of the population dynamics

All implementations of the dynamics can be found under ml_project/src/dynamics/dynamics.py. The ***replicator*** and ***Boltzmann Q-learning dynamics*** are already implemented in the OpenSpiel library. For the ***FAQ dynamics*** we simply create a new function that can be called by the Single and MultiPopulationDynamics as seen in Figure 8.

```
def boltzmann_faqlearning(state, fitness, temperature=0.001):
    exploitation = (1. / temperature) * replicator(state, fitness)
    explor_2 = np.flip(state) * np.log(np.flip(state)/state)        #only valid for 2x2 games
    return exploitation - state * explor_2
```

Figure 8: *Implementation of the FAQ dynamics.*

For the ***LFAQ dynamics*** we had to alter these populations since the fitness value now changes. The new fitness equation

$$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k:A_{ik} \leq A_{ij}} y_k \right)^{\kappa} - \left(\sum_{k:A_{ik} < A_{ij}} y_k \right)^{\kappa} \right]}{\sum_{k:A_{ik}=A_{ij}} y_k}$$

was implemented as seen in Figure 6. The dynamics update themselves are the same as with the FAQ dynamics.

```
def fitness_calc(self, n, p, ks, states):
    payoff = np.moveaxis(self.payoff_tensor[p], p, 0)
    fitness = np.zeros(shape=ks[p])
    # for each action
    for i in range(ks[p]):
        # for all other players
        p_ = abs(1 - p)  # FOR 2 PLAYERS ONLY
        u_i = 0
        # for each action each other player can take:
        for j in range(ks[p_]):
            strictworse_actions = 0     # k:Aik<Aij
            worse_actions = 0           # k:Aik≤Aij
            equal_actions = 0           # k:Aik==Aij
            #iterate over all actions the other player could have taken
            for k in range(ks[p_]):
                if payoff[i][k] < payoff[i][j]:
                    strictworse_actions += (states[p_][k])
                    worse_actions += (states[p_][k])
                elif payoff[i][k] == payoff[i][j]:
                    worse_actions += (states[p_][k])
                    equal_actions += (states[p_][k])

            u_i += payoff[i][j] * states[p_][j] * ( worse_actions**self.k - strictworse_actions**self.k) / equal_actions
        fitness[i] = u_i
    return fitness
```

Figure 9: *Implementation of the fitness value based on the class k-value.*

APPENDIX F: Quiver plots for each game with the LFAQ dynamics and different κ-values

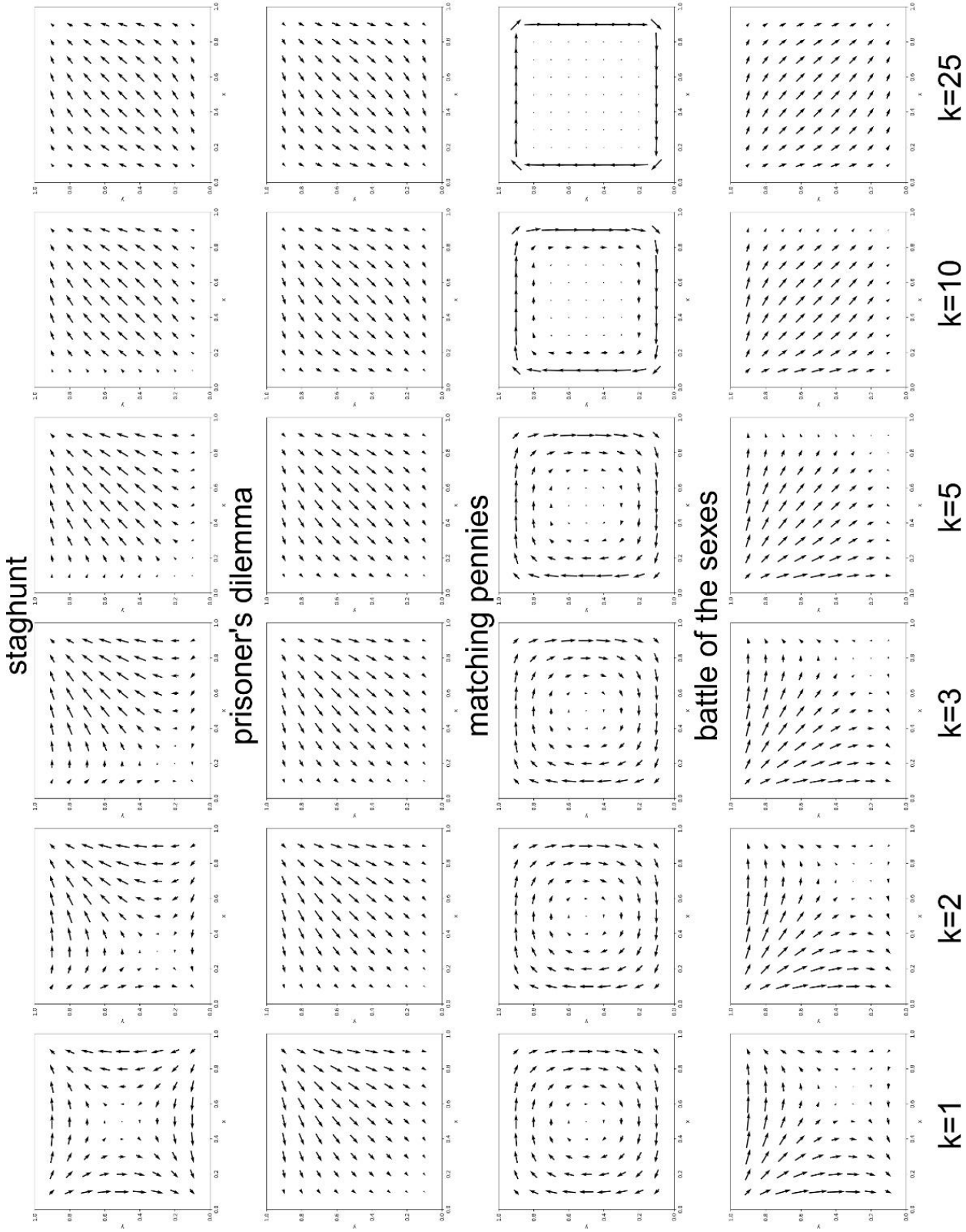

Figure 10: *LFAQ plots for the 2x2 games with different k-values. With higher q-values, payoff dominant equilibria are preferred.*

APPENDIX G: All equations together

- **Q-learning**

$$Q_i(t+1) \leftarrow Q_i(t) + \alpha\left(r_{i(t)} + \gamma \operatorname{argmax}_j Q_j(t) - Q_i(t)\right)$$

- **Boltzmann exploration scheme**

$$x_i(Q,\tau) = \frac{e^{\tau^{-1}Q_i}}{\sum_{j\in LegalActions} e^{e^{\tau^{-1}Q_j}}}$$

- **FAQ-learning**

$$Q_i(t+1) \leftarrow Q_i(t) + \min\left(\frac{\beta}{x_i}, 1\right) * \alpha\left(r_i(t) + \gamma \operatorname{argmax}_j Q_j(t) - Q_i(t)\right)$$

- **LFAQ-learning**

$$Q_i(t+1) \leftarrow Q_i(t) + \min\left(\frac{\beta}{x_i}, 1\right) * \alpha\left(r_i(t) + \gamma \operatorname{argmax}_j Q_j(t) - Q_i(t)\right)$$

$$with\ Q_i(t) the\ q\text{-}value\ of\ the\ best\ action\ (=i)\ over\ \kappa\ iterations$$

- **Replicator Dynamics (**=Cross-learning dynamics)

$$\frac{dx_i}{dt} = \dot{x}_i = x_i[f_i - \bar{f}]$$

$$\dot{x}_i = x_i[(Ay)_i - x^T Ay]$$

$$\dot{y}_i = y_i[(x^T B)_i - x^T By]$$

- **Q-learning Dynamics (**with Boltzmann exploration**)**

$$\dot{x}_i = \frac{\alpha x_i}{\tau}\underbrace{\left[(\mathbf{Ay})_i - \mathbf{x}^\top \mathbf{Ay}\right]}_{\text{exploitation}} - \alpha x_i \underbrace{\left[\log x_i - \textstyle\sum_k x_k \log x_k\right]}_{\text{exploration}}$$

- **FAQ-learning Dynamics**

$$\dot{x}_i = \frac{\alpha x_i}{\tau}\underbrace{\left[(\mathbf{Ay})_i - \mathbf{x}^T \mathbf{Ay}\right]}_{\text{exploitation}} - \alpha x_i \underbrace{\left[\sum_k x_k \ln(\frac{x_k}{x_i})\right]}_{\text{exploration}}$$

- **LFAQ-learning Dynamics**

$$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k:A_{ik}\le A_{ij}} y_k\right)^\kappa - \left(\sum_{k:A_{ik}<A_{ij}} y_k\right)^\kappa\right]}{\sum_{k:A_{ik}=A_{ij}} y_k}$$

$$\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau}(u_i - x^T u) + x_i \alpha \sum_j x_j ln(\frac{x_j}{x_i})$$

References

[1] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, “Evolutionary Dynamics of Multi-Agent Learning: A Survey,” *J. Artif. Intell. Res.*, vol. 53, pp. 659–697, Aug. 2015, doi: 10.1613/jair.4818.

[2] M. Kaisers and K. Tuyls, “Frequency Adjusted Multi-agent Q-learning,” p. 8.

[3] D. Bloembergen, M. Kaisers, and K. Tuyls, “Lenient Frequency Adjusted Q-learning,” p. 8.

[4] W. Spaniel, *Game Theory 101*. 2019.

[5] O. Salazar, *19 2 The Replicator Equation 1329*. .

[6] A. Kianercy and A. Galstyan, “Dynamics of Boltzmann Q-Learning in Two-Player Two-Action Games,” *Phys. Rev. E*, vol. 85, no. 4, p. 041145, Apr. 2012, doi: 10.1103/PhysRevE.85.041145.

[7] D. Bloembergen, “analyzing Reinforcement Learning algorithms using Evolutionary Game Theory,” Maastricht University, Maastricht, The Netherlands, 2010.

[8] J. Heinrich, “Reinforcement Learning from Self-Play in Imperfect-Information Games,” University College London, London, 2017.

[9] N. Burch, M. Lanctot, D. Szafron, and R. G. Gibson, “Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions,” p. 9.

[10] M. Zinkevich, M. Bowling, M. Johanson, and C. Piccione, “Regret Minimization in Games with Incomplete Information,” p. 14.