Baré Alexandre (r0912072)
Ngo Triet (r0869104)

Multi-Agent Learning in Games

Machine Learning: Project

Katholiek Universiteit Leuven
Faculty of Engineering Science
Academic Year 2021-2022

1 Introduction

The problem statement consists in making agents learn to play games optimally in coordination, dispersion or dilemma settings. Throughout this report, we will experiment with canonical games, Kuhn poker and FCPA poker to progressively build up an understanding of multi-agent reinforcement learning in games.

We divided the work as follows: Alexandre Baré initiated the work on task 1 and 2 while Triet Ngo initiated the work on task 3 and 4. We discussed our initial findings, proofread and improved the work of each other. For task 3 and 4, we divided the experimentation on the different learning algorithms in two. The report was written collaboratively. We spent each around 60 hours on the project (attending the lectures, learning to use open spiel, reading scientific papers, experimenting, solving the 4 tasks, writing the report).

2 Task 1: Literature

We briefly explain in this section what we have learnt from the literature in reinforcement learning and game theory. We focus mainly on the algorithms that we experimented on (see the later sections).

2.1 Lenient Boltzmann Q-Learning

We implemented Lenient Boltzmann Q-learning dynamics following the article of Kianercy et al. (2012, [4]) and the article of Bloembergen et al. (2010, [1]). More precisely, we implemented the utility vector $\mathbf{u}$ mentioned in both papers (respectively in equation (11) and (7)):

$$u_i = \sum_j \frac{a_{ij} y_j \left[\left(\sum_{k:a_{ik} \leq a_{ij}} y_k \right)^\kappa - \left(\sum_{k:a_{ik} < a_{ij}} y_k \right)^\kappa \right]}{\sum_{k:a_{ik} = a_{ij}} y_k} \tag{1}$$

which will replace the term $\mathbf{Ay}$ in the classic equation of the Boltzmann Q-Learning dynamics:

$$\dot{x}_i = \frac{\alpha x_i}{\tau} \underbrace{\left[(\mathbf{Ay})_i - \mathbf{x}^\top \mathbf{Ay} \right]}_{\text{exploitation}} - \alpha x_i \underbrace{\left[\log x_i - \sum_k x_k \log x_k \right]}_{\text{exploration}} \tag{2}$$

α adjusts the balance between exploration and exploitation in the state-action space (in Open Spiel, α is set to 1, we will assume this value from now on); τ is the temperature. It also influences the rate of exploration and exploitation; κ is the number of collected rewards for each action; $\mathbf{A}$ is the payoff matrix; $\mathbf{x}$ is the population state; $(\mathbf{Ay})_i$ is the payoff for strategy i; $(\mathbf{x}^\top \mathbf{Ay})$ is the average payoff over the population.

A theoretical discussion on the formulation and the mathematical properties of Lenient Boltzmann Q-learning algorithm can be found in the paper of Panait et al. (2008, [8]).

2.2 Deep Q-Network

Whereas in Q-Learning a table maps each (state s, action a-pair to its corresponding Q-value, it becomes impractical when the number of (state s, action a)-pairs grow exponentially. The idea behind Deep Q-Network introduced by Mnih et al. (2013, [7]) is to approximate the Q-value by a neural network that we can parameterised ourselves.

The neural network weights θ_i are updated at each step i by stochastic gradient descent on the loss

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)} \left[\left(y_i - Q(s, a; \theta_i) \right)^2 \right] \tag{3}$$

where the temporal difference y_i follows

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \tag{4}$$

$\rho(s, a)$ is a probability distribution over state s and actions a.

In Equation 4, the parameters θ_{i-1} from the last iteration are fixed during the update. In practice, instead of using the parameters from the previous iteration, the stability of the learning algorithm is improved if we copy the network parameters from a few iterations ago. This copy is the target network and is updated every k steps. k is thus a parameter to tune.

DQN relies also on an ϵ-greedy policy. Every step, the optimal action is retrieved from the network with a probability $1 - \epsilon$. With a probability ϵ, a random action is chosen instead. ϵ ensures a balance between exploitation and exploration. It is thus a parameter to tune and that can be set to decay over time for better convergence.

Finally, the paper of Mnih et al. introduced a technique called experience replay. As the game is played, transitions (s_t, a_t, r_t, s_{t+1})[1] are added to the replay buffer whose size can be tuned. While training, a batch of transitions is sampled at each step from the replay buffer instead of just using the last transition. And we compute the loss and its gradient on this batch. This technique reuses transitions from past iterations in many updates for improved data efficiency. It also increases learning stability since sampled transitions within a batch tend to be uncorrelated as opposed to consecutive state transitions.

2.3 Deep Counterfactual Regret Minimization

As detailed in [5] (2019, Lockhart et al.), Counterfactual Regret Minimization is, as Q-Learning, a tabular learning algorithm

Let us define $\eta_{-i,\boldsymbol{\pi}}(h)$ to be only the opponents' contributions to the probability of reaching history h under the joint policy $\boldsymbol{\pi}$.

The counterfactual value if we had played action a at state s is

$$q_{i,\boldsymbol{\pi}}^{(t)\,c}(s,a) = \sum_{h \in s} \eta_{-i,\boldsymbol{\pi}}(h) q_{i,\boldsymbol{\pi}}^{(t)}(h,a) \tag{5}$$

The average value over the set of available actions at state s is

$$v_{i,\pi}^{(t)\,c}(s) = \sum_{a \in \mathcal{A}_i} \pi_i(s,a) q_{i,\pi}^{(t)\,c}(s,a) \tag{6}$$

The counterfactual regret is the difference between Equation 5 and 6.

$$r^{(t)}(s,a) = q_{i,\pi^k}^{(t)\;c}(s,a) - v_{i,\pi^k}^{(t)\;c}(s) \tag{7}$$

The cumulative regret over all iterations from 1 to T is simply obtained as

$$R^{(T)}(s,a) = \sum_{t=1}^{T} r^{(t)}(s,a) \tag{8}$$

A new policy from the cumulative regrets of (s,a) can then be computed using for instance regret-matching (Hart and Mas-Colell, 2000, [3]).

In the work of Brown et al. (2018, [2]), the author extends the classic formulation of CFR by approximating the cumulative regret $R^{(t)}(s,a)$ with a deep neural network $V(s,a|\theta^{(t)})$ as universal function approximate.

2.4 Policy Gradient

While Q-Learning and its variants aim at evaluating the values of any action a taken in any state s, it only implicitly improves the agent's policy. In policy gradient (Sutton et al., 1999, [9]), we are directly following gradients with respect to the policy $\pi(a|s,\theta)$ itself parametrised with θ. The update step for the policy loss can be formulated as

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi(s \mid a, \theta)(R_t - b(s_t)) \tag{9}$$

where $b(s_t)$ is a baseline function often chosen to be the value function $V(s)$.

In Advantage Actor-Critic setting (see the work of Mnih et al. (2018, [6])), $b(s_t) \approx V(s_t)$ and $R_t \approx Q(a_t, s_t)$ such that $R_t - b(s_t)$ becomes the advantage function. It gives an estimate of how much better it is to undertake a specific action compared to the average on all possible actions at the current state. Formally, it is defined as

$$A(a,s) = Q(a,s) - V(s) \tag{10}$$

$V(s)$ is the expected value of all possible future rewards, on all possible actions while $Q(a,s)$ represents the expected value of all possible future rewards that the agent is expected to accumulate when it starts from sate s and executes action a. $Q(a,s)$ can be expressed in terms of $V(s)$ and the latter is typically approximated by a neural network $\hat{V}(s|\,\beta)$.

We thus have to optimise two losses, the pi loss for the policy $\pi(a|s,\theta)$ and the critic loss for $\hat{V}(s|\,\beta)$. The latter is a simple L2-loss.

Different variants exists to Advantage Actor-Critic (A2C or A3C): Regret Policy Gradient, Q-based Policy Gradient, Regret Matching Policy Gradient, ... We experimented mainly with A2C and Regret Policy Gradient.

[1] r_t is the reward associated to transitioning from state s_t to s_{t+1} with action a_t

In Regret Policy Gradient setting, $b(s) = \sum_a \pi(a|s,\theta) * Q(a,s)$. The loss is the regret $\sum_a ReLU(Q(a,s) - b(s))$. The update step is then rewritten as

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \sum_a ReLU(Q(a,s) - b(s)) \tag{11}$$

$Q(a,s)$ is approximated by a neural network and is trained with the critic loss.

2.5 Exploitability & Nash Convergence

As mentioned in [5] (Lockhart et al., 2019), a best response policy for player i is defined as

$$b_i\left(\boldsymbol{\pi}_{-i}\right) \in \mathrm{BR}\left(\boldsymbol{\pi}_{-i}\right) = \left\{\boldsymbol{\pi}_i \mid v_{i,(\boldsymbol{\pi}_i,\boldsymbol{\pi}_{-i})} = \max_{\boldsymbol{\pi}'_i} v_{i,\left(\boldsymbol{\pi}'_i,\boldsymbol{\pi}_{-i}\right)}\right\} \tag{12}$$

where $v_{i,\boldsymbol{\pi}}$ is the value of player i that it tries to maximise to obtain an optimal joint policy $\boldsymbol{\pi}$

The exploitability of a policy $\boldsymbol{\pi}_i$ measures how much additional value the other player could gain if it were to switch to a best response policy.

$$\delta_i(\boldsymbol{\pi}) = \max_{\boldsymbol{\pi}'_i} v_{i,\left(\boldsymbol{\pi}'_i,\boldsymbol{\pi}_{-i}\right)} - v_{i,\boldsymbol{\pi}} \tag{13}$$

In a two-player zero-sum game, an ϵ-minmax policy ensures that none of the player has an exploitability above ϵ. A Nash equilibrium is achieved when $\epsilon = 0$.

NashConv is the sum of the exploitabilities. It measures the distance to a Nash equilibrium.

$$NashConv(\boldsymbol{\pi}) = \sum_i \delta_i(\boldsymbol{\pi}) \tag{14}$$

3 Task 2: 'Learning & Dynamics' in OpenSpiel

In this section, we get familiar with basic game theory, learning and dynamics concept in different canonical games.

3.1 Independent learning in benchmark matrix games

For each matrix game, we trained the 2 agents via Q-learning.

(a) Biased rock-paper-scissors

In the zero-sum game of biased rock-paper-scissors, the 2 agents converge towards systematically playing paper. The payoff matrix describe a situation where there is always a dilemma between positive and negative rewards for each player's move. It makes sense for them to play paper as regardless of what the opponent plays, it is the option where the rewards are the closest to 0 from both player's perspective. This option corresponds to a Nash equilibrium as the strategy profile (P0:[R:0.0, P:1.0, S:0.0], P1:[R:0.0, P:1.0, S:0.0]) is such that none of the players have an incentive to unilaterally deviate from it given that the other players keep their strategy fixed. In a social dilemma, the non-cooperative payoff for a player exceeds the cooperative payoff. However, if all players fail to cooperate, everyone suffers. This is why the strategy profile is furthermore Pareto optimal. No individual can improve their own payoff without the other worsening.

(b) Dispersion game

In the dispersion game, one agent ends up only playing A and the other only B. It is indeed one of the 2 possible Nash equilibrium points: (P0:[A:1.0, B:0.0], P1:[A:0.0, B:1.0]) and (P0:[A:0.0, B:1.0], P1:[A:1.0, B:0.0]) as the only way from each player's perspective to have a positive reward is to hope that if one player chooses one move, the other player will choose the other move. Unilaterally changing their strategy would not benefit either player and collaboratively changing their strategy will make them converge to the other optimal point which is neither better or worse. Both strategy profiles have equal rewards and correspond to Pareto optima.

(c) Battle of the sexes

The battle of the sexes is a coordination game. It describes the situation where a player will earn a higher payoff when it selects the same course of action as another player. In our case, both agents end up only playing O. Q-Learning thus converges towards one of the 2 Nash Equilibrium points: (P0:[O:0.0, M:1.0], P1:[O:0.0, M:1.0]) or (P0:[O:1.0, M:0.0], P1:[O:1.0, M:0.0]). Even if each agent would earn his highest reward on only one of these 2 strategy profiles, once they reach one of these points, no agent can be made better off without the other being worse off. These are therefore Pareto optimal.

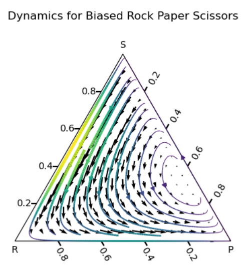
(d) Subsidy game

The subsidy game is also a coordination game. Both agents ends up playing only $S2$ after training. It corresponds to the strategy profile (P0:[S1:0.0, S2:1.0], P1:[S1:0.0, S2:1.0]) which can be characterized by a Nash equilibrium as none of the agents would win a better reward by changing their strategy while the other agent keeps his strategy fixed. It is also Pareto optimal as it is a strategy profile that combines each agent's highest individual reward.

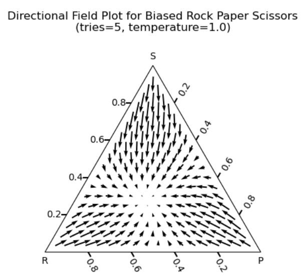
3.2 Dynamics of learning in benchmark matrix games

(a) Biased rock-paper-scissors

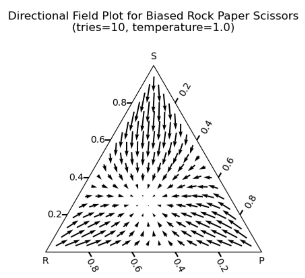
In biased rock-paper-scissors, the replicator dynamics plot indicates that the gradient strongly evolves from scissors to rock to paper.



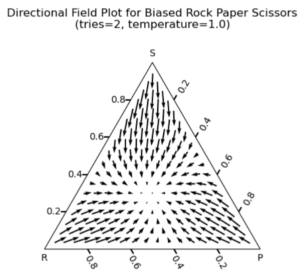
((a)) Replicator Dynamics



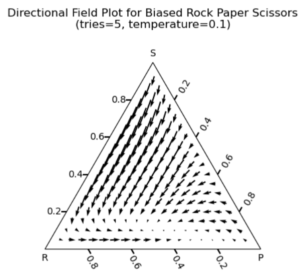
((b)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 1.0$)



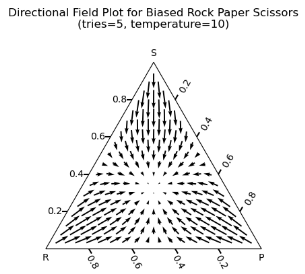
((c)) Lenient Boltzmann Q Dynamics ($\kappa = 10$, $\tau = 1.0$)



((d)) Lenient Boltzmann Q Dynamics ($\kappa = 2$, $\tau = 1.0$)



((e)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 0.1$)



((f)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 10$)

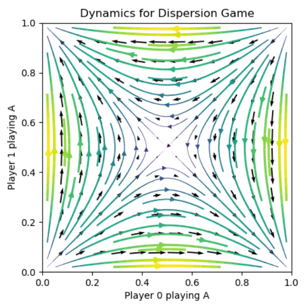
Figure 1: Directional Fields and Training Trajectories in Biased Rock-Paper-Scissors

If we now look at the agent's training trajectory, he follows the expected gradient and converges around $[0, 0.65, 0.35]$ which is close but not exactly the optimal point that we discussed previously. The gradient of the replicator dynamics is indeed close to zero along the axis between scissors and paper, allowing little further improvement around these points.

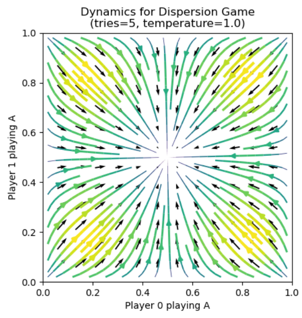
Interestingly, when we rely on lenient Boltzmann Q-Learning dynamics, the point of convergence shifts a bit towards rock and as we further increase the parameter κ. However, we were unable to have a convergence at training time with and without leniency (hence, the absence of training trajectory lines). If we decrease the temperature (i.e. increase exploitation) τ to 0.1, we see that the point of convergence is much less clear as the gradient is less important near the axis between rock and paper. If we increase the temperature (i.e. increase exploration), the agents tend to choose evenly between rock, paper and scissors.

(b) Dispersion game

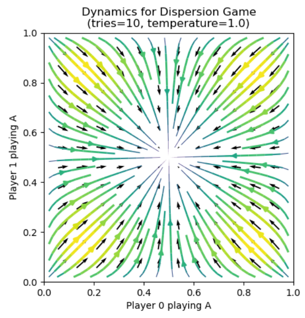
In the dispersion game, the replicator dynamics plot and the corresponding training trajectories describe the convergence towards the 2 possible points that we mentioned before $([1.0, 0.0], [0.0, 1.0])$ and $([0.0, 1.0], [1.0, 0.0])$. We notice that the velocity describes by the gradient is more important when the probability of playing A (or B) is low while the probability of playing B (resp. A) is increasing. It emphasizes once again how the agents are pushed to play as often as possible opposite moves.



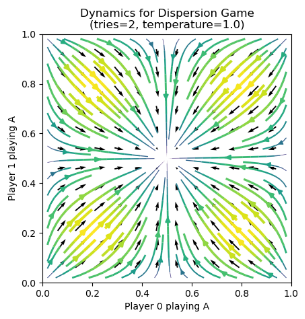
((a)) Replicator Dynamics



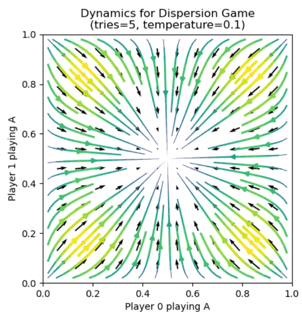
((b)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 1.0$)



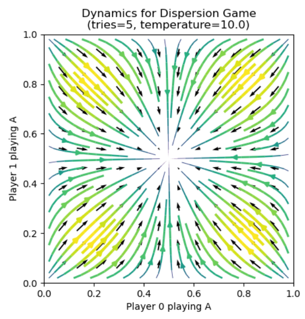
((c)) Lenient Boltzmann Q Dynamics ($\kappa = 10$, $\tau = 1.0$)



((d)) Lenient Boltzmann Q Dynamics ($\kappa = 2$, $\tau = 1.0$)



((e)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 0.1$)



((f)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 10$)

Figure 2: Directional Fields and Training Trajectories in the Dispersion Game

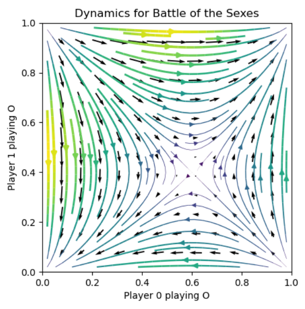
When we rely on lenient Boltzmann Q-Learning dynamics, we manage to converge to a point that we would have guessed as rather unstable: $([0.5, 0.5], [0.5, 0.5])$ where 50% of the time, both players will play the same move and receive with their worse rewards and the other 50%, they will play different moves and be granted with their best rewards. Note that κ and τ do not seem to influence much the results. We argue that to be the consequence of a perfect symmetry in the payoff matrices.

(c) Battle of the sexes

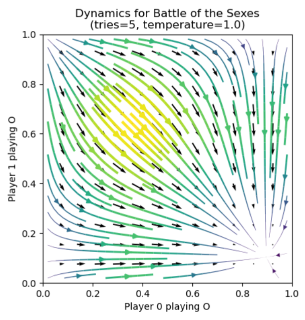
In the case of the replicator dynamics in the battle of the sexes, the directional field pushes the players to play the same action. We can indeed spot the 2 Nash equilibrium points that we discussed before. The gradient is more important from $([0, 1], [1, 0])$ to either $([0, 1], [0, 1])$ or $([1, 0], [1, 0])$. It shows that which player is starting the game matters. Here, at the point $([0, 1], [1, 0])$, the first player is in the worst situation from his perspective as he plays M, a move that will not make him earn his highest reward regardless of the second player's move. At the same time, the second player plays O, a move that could make him earn his highest reward if the first player changes his strategy. Therefore, there is a stronger incentive to leave the initial point $([0, 1], [1, 0])$ than $([1, 0], [0, 1])$ if the first player starts.

When we rely on lenient Boltzmann Q-Learning dynamics that takes into account κ rewards for each action before updating the utility vector and the gradient, we see each time only one clear point of convergence appearing. When

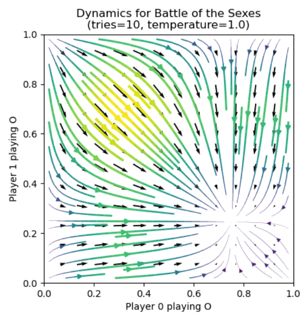
$\kappa = 5$ and $\tau = 1.0$, it is around $([0.9, 0.1], [0.1, 0.9])$ where player 0 tends to play O and player 1 tends to play M. The lower κ is, the closer to $([1.0, 0.0], [0.0, 1.0])$ the point of convergence goes. If we lower the temperature (i.e. increase exploitation), the convergence point is reversed and the agents move towards the strategy profile $([0.0, 1.0], [1.0, 0.0])$. In both cases, the convergence point make for a very bad strategy profile where the rewards for both players will be very low. If we instead increase the temperature, i.e. increase the exploration rate, the point of convergence unsurprisingly shifts towards $([0.5, 0.5], [0.5, 0.5])$.



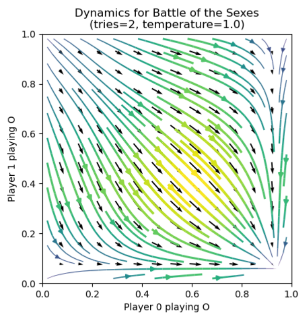
((a)) Replicator Dynamics



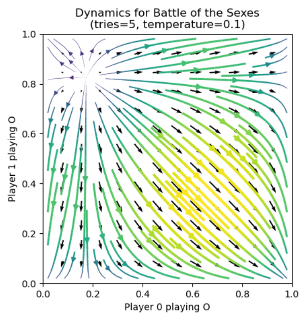
((b)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 1.0$)



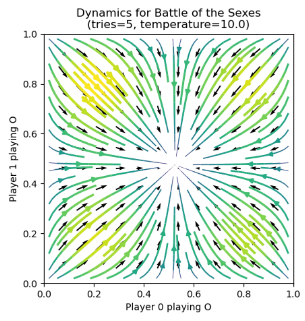
((c)) Lenient Boltzmann Q Dynamics ($\kappa = 10$, $\tau = 1.0$)



((d)) Lenient Boltzmann Q Dynamics ($\kappa = 2$, $\tau = 1.0$)



((e)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 0.1$)



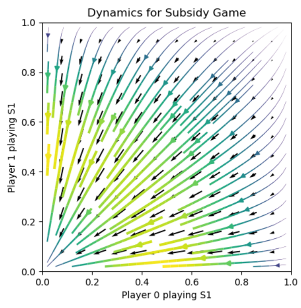
((f)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 10$)

Figure 3: Directional Fields and Training Trajectories in the Battle of the Sexes

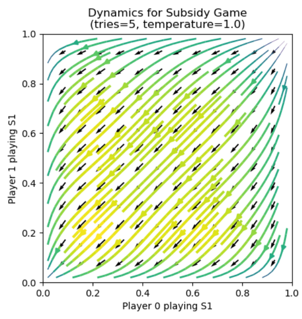
(d) Subsidy game

The replicator dynamics in Figure 4 for the subsidy game shows a convergence to the Pareto optimal strategy profile $([0.0, 1.0], [0.0, 1.0])$ that we discussed before. We observe a less and less important gradient when at least one player (and an even less important gradient when both players) tend to play more often $S1$ instead of $S2$. It means that depending on the initial strategy profile, if the agents start at one of the close-to-zero-gradient region, they may take too long for their strategy profile to converge and miss the Pareto optimal point. Initialization plays thus an important role in this game.

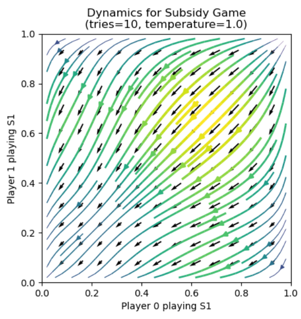
In Figure 4 for the Lenient Boltzmann Q-Learning Dynamics, we see that the strategy profile of convergence is unchanged but the gradient that leads to it can be more uniform when a good balance between κ and τ ($\kappa = 5$; $\tau = 1.0$). But the gradient can also become more important further from (resp. closer to) $([0.0, 1.0], [0.0, 1.0])$ when κ increases such as for $\kappa = 10$ (resp. when κ decreases such as for $\kappa = 2$). Lowering the temperature (i.e. increasing exploitation) speeds up the convergence when we are close to the point $([0.0, 1.0], [0.0, 1.0])$. Increasing it (i.e. increasing exploration) will however tend to push towards a perfectly random policy.



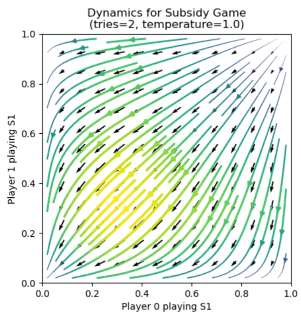
((a)) Replicator Dynamics



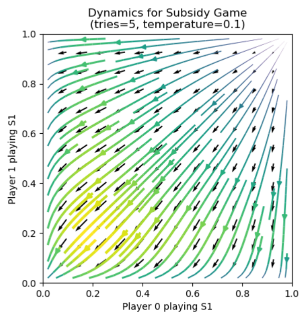
((b)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 1.0$)



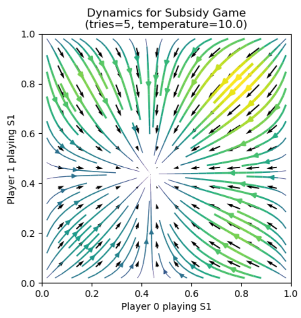
((c)) Lenient Boltzmann Q Dynamics ($\kappa = 10$, $\tau = 1.0$)



((d)) Lenient Boltzmann Q Dynamics ($\kappa = 2$, $\tau = 1.0$)



((e)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 0.1$)



((f)) Lenient Boltzmann Q Dynamics ($\kappa = 5$, $\tau = 10$)

Figure 4: Directional Fields and Training Trajectories in the Subsidy Game

3.3 Conclusion

In independent learning, Q-Learning has always led to a Pareto optimal strategy profile in our benchmark matrix games. The convergence to an optimal policy has indeed been proven provided that each state-action pair is visited infinitely often and that the sum of the learning rates must diverge, but the sum of their squares must converge.

In evolutionary game theory, learning with replicator dynamics is well principled and often leads to a relevant Nash Equilibrium but can suffer from bad initialization, small gradients, unstable convergence, convergence to sub-optimal points, ... Boltzmann Q-Learning dynamics remedied partly to the problem by allowing one to specify a balance between exploitation (the classic replicator dynamics equation) and exploration. Lenient Boltzmann Q-Learning dynamics allows an agent to forgive possible mistakes of other agents that resulted in lower rewards by looking for the highest reward among κ rewards collected for each action. It proved to be helpful in some of our matrix games in order to converge to a Nash equilibrium.

4 Task 3: Kuhn poker

In this section, our goal is to find out, on a simpler problem (Kuhn Poker), which algorithm is worth experimenting further with, in the larger game of FCPA poker. We experimented, each time in a self-learning setting, with different learning algorithms namely policy gradient, deep Q-network and deep counterfactual regret minimization.

4.1 Deep Q-Network agents

Our DQN agents are based on simple Multi-Layer Perceptrons with 2 hidden layers of size 64 and 32 units. They rely, amongst others, on a replay buffer of 10^4 transitions, a learning rate of 0.01, a batch size of 128, no reward discount, an ϵ that starts at 1 but progressively decays until 0.1 during the 10^6 training iterations. The target network weights are updated every 10^3 iterations. We however experience difficulties in tuning stable and high-performance DQN networks.

4.2 Policy Gradient agents

Our Policy Gradient agents are made of simple Multi-Layer Perceptrons with 2 hidden layers of size 64 and 32 units. The critic networks have a learning rate of 0.01 and the policy networks have a learning rate of 0.001. The batch size is of 16. We make use of gradient norm (= 3) clipping to prevent exploding gradients. Every 8th critic learning step, we update the policy networks. We rely on the regret policy gradient loss.

4.3 Deep CFR agents

Our Deep CFR agent achieves the best results. Both the policy and advantage networks are MLPs with 1 hidden layer of 16 units. The learning rate is set to 0.001 and batch size of the advantage network to 128 and of the policy network to 1024. The advantage network and policy network are trained in respectively 20 steps and 400 steps per iteration. We perform 4 iterations at a time and build a tabular policy out of the network's action probability distribution. We also report the loss and NashConv each time.

4.4 Train losses

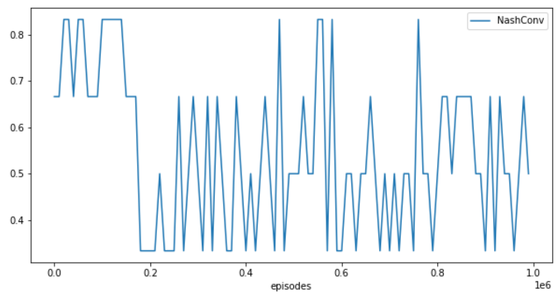
The training losses can be found in Appendix A in Figure 7.

Despite the ϵ-greedy policy with a decaying ϵ that should bring some form of regularization, DQN is hard to train and we observe high variation in the loss for both players. At the final training step, there remains a noticeable difference between both agents' loss.

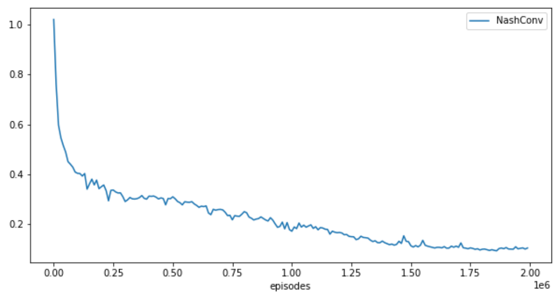
For Policy Gradient, we noticed that using gradient clipping really helped to obtain smoother losses. The critic losses fluctuate widely but we see that the pi losses are decreasing more smoothly and converging which proves that the policy improves for both agents.

In the case of Deep CFR, the losses of both the advantage and the policy network moderately oscillate and significantly decrease at the beginning of the training.

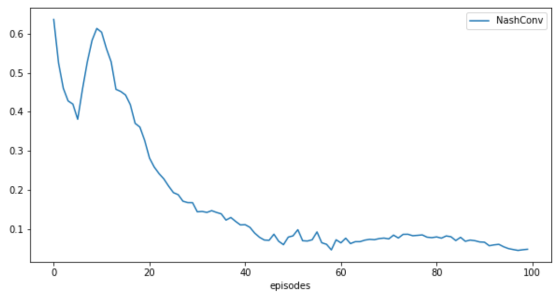
4.5 NashConv



((a)) DQN



((b)) Policy Gradient



((c)) Deep CFR

Figure 5: NashConv under different reinforcement learning algorithms

Given that we are in a two-player zero-sum game, the exploitabilities of the two agents are equal and the NashConv amounts to the sum of them both. We only report below a comparison of the NashConv with respect to the number of iterations. The exploitabilities of each agent can be deduced simply by diving the NashConv values by 2.

Our best NashConv with DQN is reported to be 0.5, a result that indicates that our agents are quite far from a Nash Equilibrium. We observe high fluctuations in the NashConv curves that can be explained by the ϵ-policy. Indeed ϵ starts at 1 in the first epochs and gradually decreases until 0.1 at the last epochs. There is thus important room for exploration which gives inconsistent NashConv values during the training process. It therefore makes it hard to track the improvements of DQN as we tune it. We will thus not select this approach for task 4.

With policy gradient, we experience a way better convergence of both agents with a NashConv of 0.096 which gets closer to a Nash Equilibrium.

However, it is only via a tabular policy built after repetitively training with Deep CFR that we are finally able to reach a very low NashConv. With a value of 0.048, we can assume that a Nash Equilibrium is reached. We will thus select this approach for task 4.

5 Task 4: FCPA poker

After proving the advantage of Deep CFR on Kuhn Poker, our aim in this section is to extend the complexity of the initial task with the game of FCPA poker. We will show that a carefully tuned Deep CFR can be a valuable choice to learn to play FCPA poker. The directory on the departmental computers where our code and agent are stored is '/cw/lvs/NoCsBack/vakken/ac2122/H0T25A/ml-project/r0869104'.

Note that some extra figures of our experiments are available in Appendix A.

5.1 Objectives & configuration

We can not compute the exploitability and NashConv anymore due to the enormous size of the (state, action)-space. We, therefore, evaluated our agents by having them compete against fixed-strategy bots to obtain the win-rates and average rewards. We implemented bots with the following playing strategy: playing randomly, only check-call, only pot-size bet, only fold, only all-in, 50% call / 50% fold.

Win rates show how much the agent learns to dominate as many games as possible. However, in the game of poker, it seems that this is not a good proxy to evaluate the agents. The goal is not to win every single game which is very challenging due to imperfect information and the intrinsic randomness of poker. The agents are given a number of coins to start the games and the goal is rather to end up with as many coins as possible. We thus focus our analysis on the average coins, i.e. average rewards, they receive over a number of consecutive games.

While training, we saved each time an agent at the current iteration if it increased its average rewards over 1000 games against all bots compared to the last agent saved in the same position. This ensures that we end up with an optimal pair of players that play well in first and second position. They will be able to face an unpredictable/irrational, very aggressive or very cautious opponent.

We want to emphasize that extensive experimentation in this task was particularly difficult to handle with our limited computing resources. [2]

Our final configuration is built with Deep CFR made of two MLPs, each with 2 hidden layers of size 256 and 128. The learning rate is 0.001, the batch size is 1024 and the memory capacity is of 10000. We run the algorithm for 50 iterations and it transverses the game 200 times and trains the networks in 500 steps per iteration.

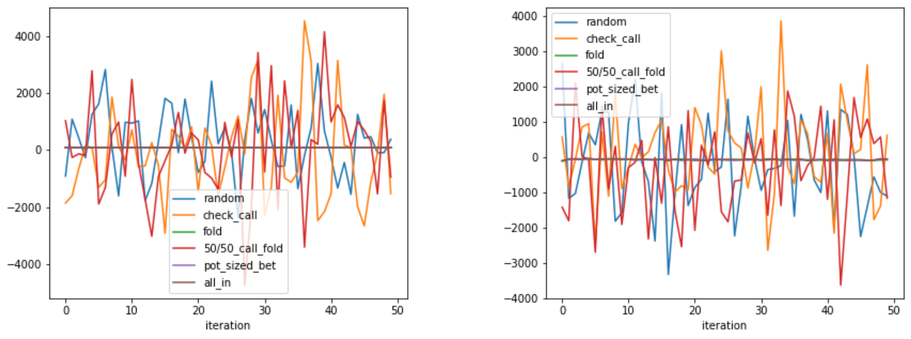
The size and depth of the networks are larger than the previous task to compensate for the increased game size. With the same logic, we also increased the number of traversals to get more samples from the game. Taking into account our limited resources, we progressively adjusted the memory capacity and batch size by binary search so that it could store data without running out of memory.

[2]Even with Colab Pro (25GB RAM and better GPU support), we constantly ran out of memory which resulted in many long experiments (2 to 3 hours) that crashed.

5.2 Training & evaluations

For DQN and Policy Gradient, the losses in Figure 9 and 10) show that none of both agents got clearly the upper hand during the training, which lead to a fairer training. However, for Figure 8, there is a small but systematic difference in the advantage of agent 0 which can be both spotted in the losses and the average rewards. We expect our final submission to play better in first position (agent 0) than second (agent 1).

For the 3 learning algorithms in Figure 13, 14 and 15, we see that agent 0 systematically achieves a 100% win rate against bots that play only all-in, pot-size bet or fold. In other words, when our agent plays first, he can win against those bots with minimal training.



((a)) Agent 0 ((b)) Agent 1

Figure 6: Average rewards against different bots using Deep CFR

Let us now focus on our final submission using Deep CFR. We could not seem to beat the random and check-call bots consistently in terms of win-rate, the best we could achieve is respectively 0.76 and 0.57 (see Figure 13). However, with a lower win-rate, we managed to get good expected rewards using Deep CFR (see Figure 6). Note that as we expected, agent 1 struggles more to win against bots than agent 0.

We submitted in total 2 versions of our bots for the tournament: one using TensorFlow 1 and Policy Gradient and the other using Deep CFR and TensorFlow 2. From the tournament results, the performance of the bot that relied on Deep CFR trumped the one that used Policy Gradient. In our last few tournaments, we also tested out different configurations of our agents with others in the course.

References

[1] Daan Bloembergen, Michael Kaisers, and Karl Tuyls. “Lenient frequency adjusted Q-learning”. In: Oct. 2010, pp. 19–26.

[2] Noam Brown et al. “Deep Counterfactual Regret Minimization”. In: *CoRR* abs/1811.00164 (2018). arXiv: `1811.00164`. URL: `http://arxiv.org/abs/1811.00164`.

[3] Sergiu Hart and Andreu Mas-Colell. “A Simple Adaptive Procedure Leading to Correlated Equilibrium”. In: *Econometrica* 68.5 (Sept. 2000), pp. 1127–1150. URL: `https://ideas.repec.org/a/ecm/emetrp/v68y2000i5p1127-1150.html`.

[4] Ardeshir Kianercy and Aram Galstyan. “Dynamics of BoltzmannQlearning in two-player two-action games”. In: *Physical Review E* 85.4 (Apr. 2012). ISSN: 1550-2376. DOI: `10.1103/physreve.85.041145`. URL: `http://dx.doi.org/10.1103/PhysRevE.85.041145`.

[5] Edward Lockhart et al. “Computing Approximate Equilibria in Sequential Adversarial Games by Exploitability Descent”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 464–470. DOI: `10.24963/ijcai.2019/66`. URL: `https://doi.org/10.24963/ijcai.2019/66`.

[6] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). arXiv: `1602.01783`. URL: `http://arxiv.org/abs/1602.01783`.

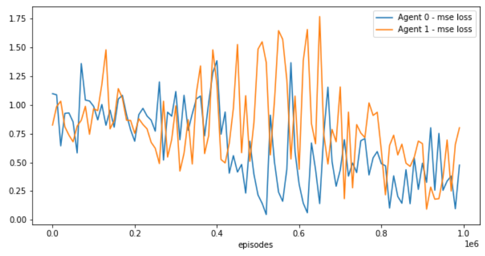
[7] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: `1312.5602`. URL: `http://arxiv.org/abs/1312.5602`.

[8] Liviu Panait, Karl Tuyls, and Sean Luke. “Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective”. In: *Journal of Machine Learning Research* 9.13 (2008), pp. 423–457. URL: `http://jmlr.org/papers/v9/panait08a.html`.

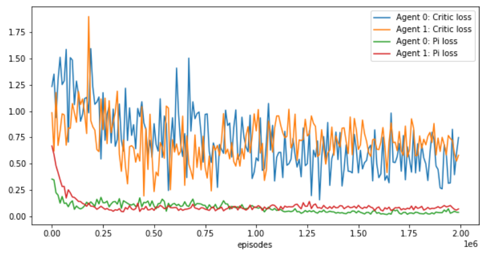
[9] Richard S Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: `https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`.

A Appendix

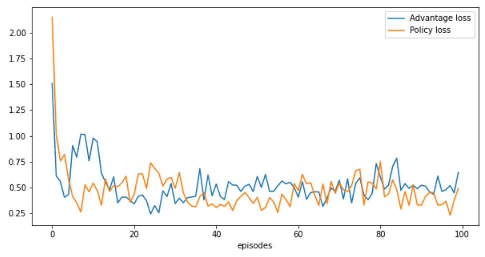
1.1 Task 3 - Training Losses



((a)) DQN Squared Error Loss



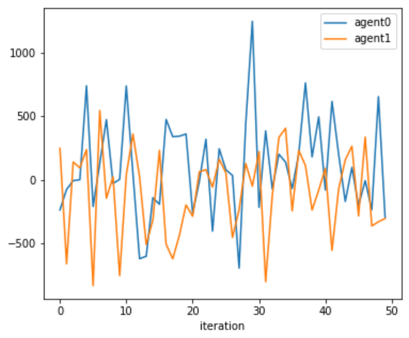
((b)) Policy Gradient Losses



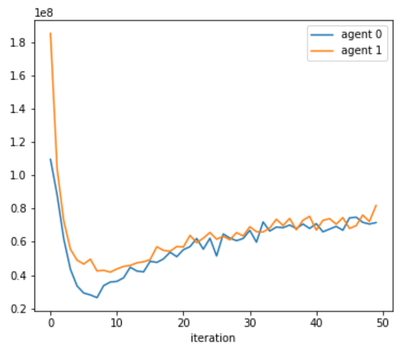
((c)) Deep CFR Losses

Figure 7: Agents' Losses under different reinforcement learning algorithms

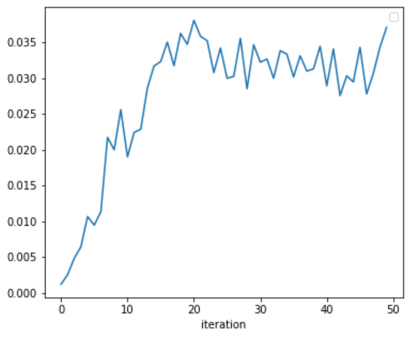
1.2 Task 4 - Training Losses



((a)) Average rewards

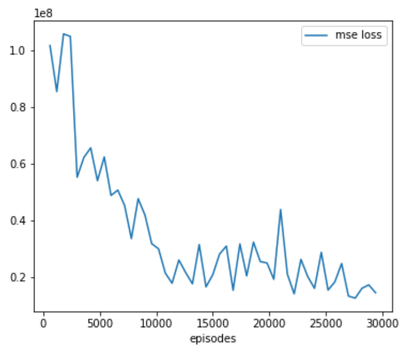


((b)) Advantage losses

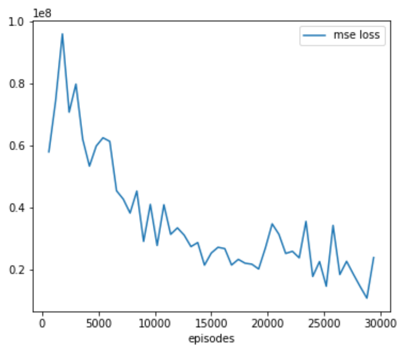


((c)) Policy loss

Figure 8: Average rewards and training losses for Deep CFR

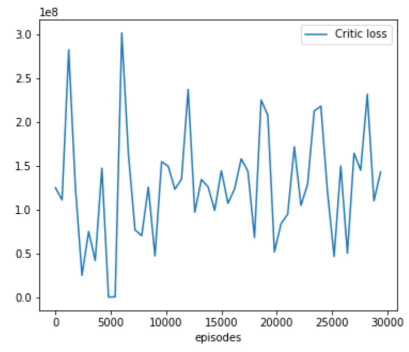


((a)) Agent 0

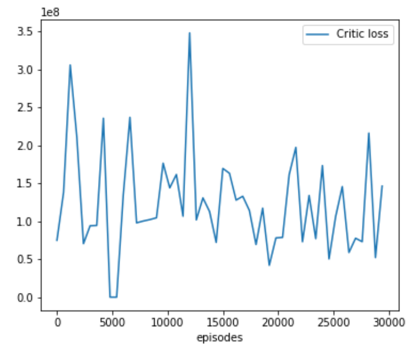


((b)) Agent 1

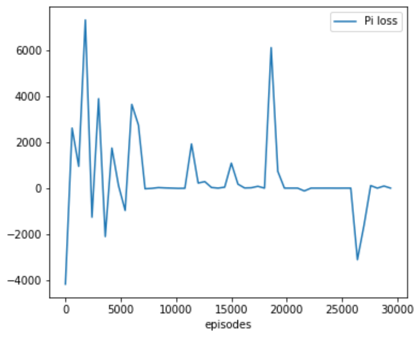
Figure 9: Training losses for DQN



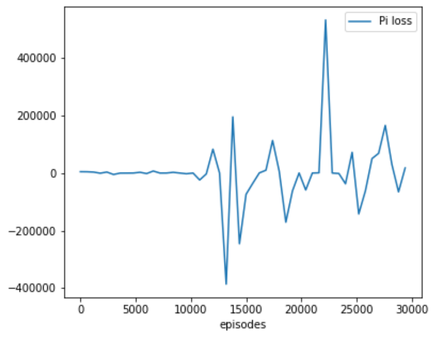
((a)) Agent 0 critic loss



((b)) Agent 1 critic loss



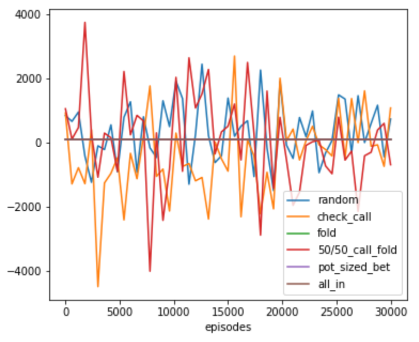
((c)) Agent 0 pi loss



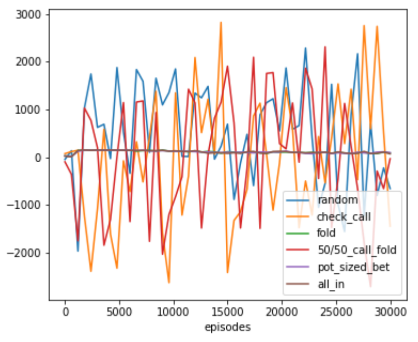
((d)) Agent 1 pi loss

Figure 10: Training losses for Policy Gradient

1.3 Task 4 - Evaluation against different bots

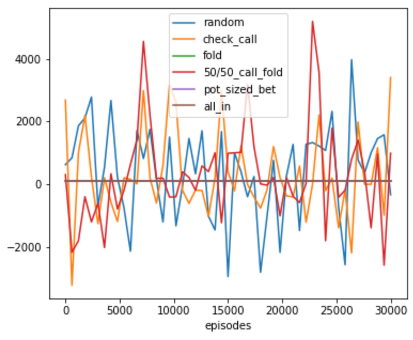


((a)) Agent 0

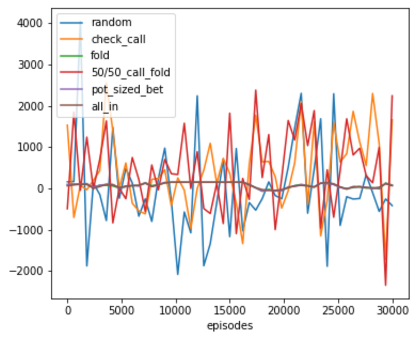


((b)) Agent 1

Figure 11: Average rewards against different bots using DQN

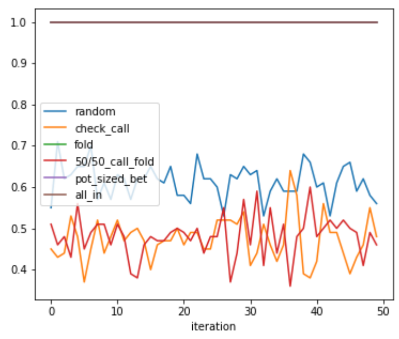


((a)) Agent 0

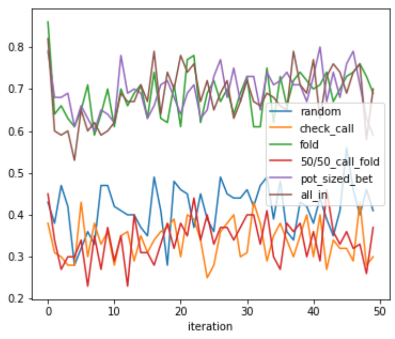


((b)) Agent 1

Figure 12: Average rewards against different bots using Policy Gradient

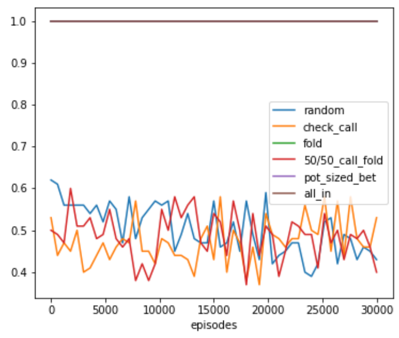


((a)) Agent 0

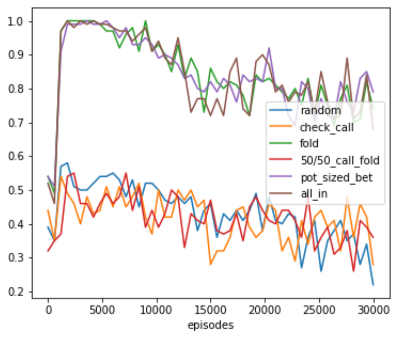


((b)) Agent 1

Figure 13: Win-rates against different bots using Deep CFR

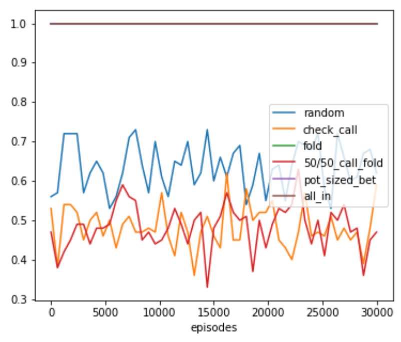


((a)) Agent 0

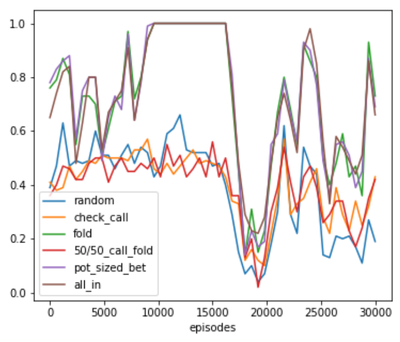


((b)) Agent 1

Figure 14: Win-rates against different bots using DQN



((a)) Agent 0



((b)) Agent 1

Figure 15: Win-rates against different bots using Policy Gradient