Machine Learning Project: Report 2

Ignace Bleukx Quinten Bruynseraede

May 6, 2020

1 Introduction

1.1 Evaluation metrics

Evaluation of agents is traditionally done using **NashConv** and **exploitability**. We introduce these concepts here, using terminology consistent with Lanctot et al. [3].

Given a two-player policyπ, we say that π_i^b is the best response for player i. The best response is defined as the policy that maximizes payoff for player i, given the policies of other players (pi_{-i}).

We then define the incentive to change policies $d_i(\pi)$ as $d_i(\pi) = u_i(\pi_i^b, \pi_{-i}) - u_i(\pi)$, i.e. the possible gain in value when switching to a best-response strategy. It is clear that this can be used as a metric to evaluate a policy: a Nash equilibrium is found when $d_i(\pi) = 0$ (the policy cannot be improved given other player's policies). Any value > 0 captures room for improvement for a given theory.

From this initial notion of policy evaluation, the **NashConv** metric is derived as the sum of $d_i(\pi)$ for both players.

Another metric that is often used, is **exploitability**. For two-player zero-sum games (such as the games we will be learning in this assignment), exploitability equals $\frac{NashConv}{2}$. We therefore only use Exploitability to evaluate our policies (as NashConv can be derived from Exploitability in this case).

Looking for Nash equilibria is interesting in zero-sum games because they guarantee a maximal payoff against any policy the other players might have. We will therefore focus on finding approximations of Nash equilibria (i.e. minimizing exploitability). It should also be noted that convergence toward Nash equilibria is a property of individual algorithms, and is not guaranteed.

1.2 Algorithm 1: Fictitious Self-Play

Fictious play is a classic algorithm introduced by [**?**]. It's goal is to find the optimal strategy or Nash equilibrium for imperfect games in normal form [**?**]. The algorithm is based on the fictitous play of the game, which corresponds to a player simulating a game in it's head and choosing the best response for the current situation. In this simulation, the player assumes the opponent always chooses the best response.

We concretise this idea by explainin the Extensive Fictious Play algorithm, as stated in [2]. The algorithm consisists of two steps: finding the best response for the current state, and updating the current average strategy of the agent. To find the best reponse for the curren state, the game state tree is traversed and the set of best responses is calculated based on the final outcome. Out of this set, a best reponse is selected. The seconds step is to update the average policy by using the calculated best response. The update rule for the policy is the following: $\pi_{i+1}(u) = \pi_i(u) + \frac{\alpha_{i+1} x_{\beta_{i+1}}(\sigma_u)(\beta_{i+1}(u) - \pi_i(u))}{(1-\alpha_{i+1}) x_{\pi_i}(\sigma_u) + \alpha_{i+1} x_{\beta_{i+1}}(\sigma_u)}$. Where $\pi_n(u)$ indicates the action probabilities of the policy at iteration n, β_n the best

reponse calcuated in step 1 and σu a sequence of actions to reach state u. $x\beta(\sigma_u)$ is the realization plan for this sequence, given best response β.
The main issue with this form of fictitous play is the computational effort needed to calculate the best response. Reinforcement learning can address this problem by using machine learning techniques to approximate the optimal response. The reinforcement techniques plug in in a framework called FSP or Fictious Self Play. Here, both calculation of the best response and the update of the policy can be substitued by reinforcement learning methods.

1.2.1 Extension: Neural Fictitious Self-Play

In the NFSP algorithm, the best response and update rule are approximated by training two neural networks.

Best response To calculate the best response, we train a Deep Q-network or DQN. This agent learns an ϵ-greedy policy [1]. This can be done as the games considered have perfect recall and the history of the game is thus available to the agents. By using this history the agent gains experience and approximates the best response for a given state. The implementation of NFSP in Openspiel uses a multilayer perceptron network (MLP).

Average Strategy To compute the action probabilities for a given state in the game, NFSP trains another ANN which maps these states to the corresponding probabilities. The network is trained by using information of past actions performed. As the average policy is approximated, the network does not need to be consulted every iteration of the learning process. This means the update can be much more effecient when using a good ANN. In the Openspiel implementation of NFSP this is an MLP.

1.3 Algorithm 2: Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR in short) is an algorithm that is designed to find Nash equilibria in large games. It introduces the notion of counterfactual regret, and minimizes this to compute a Nash Equilibrium. Zinkevich et al. [6] show that CFR can solve games with up to 10^{12} states, such as Texas Hold'em. We therefore expect it to perform well on reduced variants of traditional poker, such as Kuhn and Leduc poker.

An important concept in reinforcement learning is the notion of regret, which can be paraphrased as the difference between maximum and actual payoff an agent receives when executing a sequence of steps. The goal of many reinforcement learning algorithms is to minimize regret. What makes regret minimization less applicable in large games, is that regret is to be minimized over all game states. This quickly becomes infeasible when dealing with a large number of states. The general idea of counterfactual regret minimization is to decompose regret into a set of regret terms, whose som approaches the actual regret. Zinkevich et al. then show that individually minimizing these regret terms leads to a Nash equilibrium. We will now introduce counterfactual regret minimization more formally, based on [4]

- A history h is a sequence of actions performed by the agents, starting from the root of the game tree (the initial state).
- An information state I consists of a player and the information that is visible to that player.

- All player have their individual strategies, and we call the combination of these strategies at time t: σ^t. $\sigma^t_{I\to a}$ denotes a strategy identical to σ, but action a will always be taken in information state I.
- Counterfactual ($-i$) refers to excluding the probabilities of actions that player i took, as if he guided (with action probabilities 1) the game towards a certain history or information. We can use this definition to calculate $P^\sigma_{-i}(h)$ and $P^\sigma_{-i}(I)$.
- The counterfactual value of a history h under a joint strategy σ is defined as:

$$v_i(\sigma, h) = \sum_{z\in Z, h\sqsubset z} P^\sigma_{-i}(h)P^\sigma(h,z)u_i(z) \tag{1}$$

  where Z denotes the set of possible terminal game histories, and h is any non-terminal subhistory of $z \in Z$. This equation captures that the value of a certain history has to be weighed to account for the counterfactual probabilities. More specifically, the actions of other players (where chance influences their play) influence the probability of a history, which is taken into account.
- The counterfactual regret is defined as:

$$r(h,a) = v_i(\sigma_{I\to a}, h) - v_i(\sigma, h) \tag{2}$$

  This regret captures how much value was lost by *not* picking action a, as opposed to a strategy that does execute a in history h.
- The counterfactual regret of not taking an action a in information state I is simply the sum of regret of all histories in I:

$$r(I,a) = \sum_{h\in I} r(h,a) \tag{3}$$

- Finally, the cumulative counterfactual regret of not taking action a in information state I if called $R^T(I,a)$ and can be expressed as the sum of regrets at all time steps:

$$R^T(I,a) = \sum_{t=1}^{T} r(h,a) \tag{4}$$

As expected, the goal is to locally minimize this regret for each information state. This is done iteratively. In each step, a strategy can be extracted from the counterfactual regrets. One way of doing this is using regret matching, where action probabilities are computed proportional to positive regrets. For example, a high positive regret indicates that small losses have been experienced when not picking a certain an action. It is therefore interesting to perform this action more often, and it will be given a higher weight.

Openspiel uses Hart and Mas-Colell's regret-matching algorithm, which obtains a new strategy σ^{T+1} in the following way:

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & \text{if denominator } > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases} \tag{5}$$

In this equation, $A(I)$ denotes the set of actions that can be taken in information state I, and R^+ is obtained by bounding regret values in the interval $[0,\infty[$. The value of σ^{T+1} is calculated for each information state and each action to obtain a complete strategy.

It should be noted that the average strategy converges to a Nash equilibrium, and not the final strategy. When discussing results of policies obtained using CFR, it is implicitly assumed we have extracted the average policy after training.

1.3.1 Extension: Regression Counterfactual Regret Minimization

Regression CFR (introduced by Waugh et al. [5]) is an extension of CFR where counterfactual regret values are not stored in a tabular format, but estimated using a function $\phi : I \times A \mapsto \mathbb{R}.$, that maps each combination of a (visible) information state and an action to an estimation of the counterfactual regret value.

One interesting property of using regression CFR (RCFR) is the fact that a game can be abstracted down using a appropriate regressor. Consider a game like Texas Hold'em Poker, with roughly 10^{160} possible sequences. Storing all counterfactual regret values quickly becomes impossible. A solution is to assign regret values to clusters and estimate a representative for each cluster. This might reduce the number of values to a manageable size. In RCFR, this clustering is done implicitly when choosing a regressor: a simple model such as bounded regression trees will learn rough abstractions, whereas a neural netwerk may not add any abstraction at all. Furthermore, RCFR is able to adapt the regressor ϕ on the fly, increasing or decreasing abstraction as needed.

As such, we note that using RCFR instead of regular CFR should be used as a technique to make learning more tractable, and not to decrease exploitability. Therefore, the goal should be to approximate the exploitability reached by CFR, but with a reduced number of states.

OpenSpiel supports using RCFR with a user-defined regressor. We chose to use a feedforward neural network of one hidden dense layer containing 13 nodes.

1.3.2 Extension: Counterfactual Regret Minimization against best responder

1.3.3 Extension: Deep Counterfactual Regret Minimization

2 Kuhn Poker

- Which algorithm is most suitable to develop an agent to play Kuhn Poker, minimizing exploitability?
- Can we exploit properties of Kuhn Poker to optimize parameters?

3 Leduc Poker

- Which algorithm is most suitable to develop an agent to play Leduc Poker, minimizing exploitability?
- Can we exploit properties of Leduc Poker to optimize parameters?
- Can we combine agents into an ensemble that minimizes exploitability further than its parts?

References

[1] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. 2016.

[2] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games, 2016.

[3] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. Openspiel: A framework for reinforcement learning in games, 2019.

[4] Todd W. Neller and Marc Lanctot. An introduction to counterfactual regret minimization. 2013.

[5] Kevin Waugh, Dustin Morrill, J. Andrew Bagnell, and Michael Bowling. Solving games with functional regret estimation. *CoRR*, abs/1411.7974, 2014. URL `http://arxiv.org/abs/1411.7974`.

[6] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1729–1736. Curran Associates, Inc., 2008. URL `http://papers.nips.cc/paper/3306-regret-minimization-in-games-with-incomplete-information.`

Appendix

3.1 Time spent