STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 2, 2021
DOI: 10.24193/subbi.2021.2.04

# DEEP REINFORCEMENT LEARNING FROM SELF-PLAY IN NO-LIMIT TEXAS HOLD'EM POKER

TIDOR-VLAD PRICOPE

Abstract. Imperfect information games describe many practical applications found in the real world as the information space is rarely fully available. This particular set of problems is challenging due to the random factor that makes even adaptive methods fail to correctly model the problem and find the best solution. Neural Fictitious Self Play (NFSP) is a powerful algorithm for learning approximate Nash equilibrium of imperfect-information games from self-play. However, it uses only crude data as input and its most successful experiment was on the in-limit version of Texas Hold'em Poker. In this paper, we develop a new variant of NFSP that combines the established fictitious self-play with neural gradient play in an attempt to improve the performance on large-scale zero-sum imperfect-information games and to solve the more complex no-limit version of Texas Hold'em Poker using powerful handcrafted metrics and heuristics alongside crude, raw data. When applied to no-limit Hold'em Poker, the agents trained through self-play outperformed the ones that used fictitious play with a normal-form single-step approach to the game. Moreover, we showed that our algorithm converges close to a Nash equilibrium within the limited training process of our agents with very limited hardware. Finally, our best self-play-based agent learnt a strategy that rivals expert human level.

## 1. Introduction

Learning by interacting with a certain environment (or emulator) has its roots in the way human brain evolved, or how natural intelligence advances [1]. We can consider a game as a simulation of our real world with its own set of rules and features. Some games resemble real-world problems on a smaller scale which means that solutions can provide an intuition for tackling real applications such as financial trading, traffic control, airport and network

Received by the editors: 1 June 2021.
2010 *Mathematics Subject Classification.* 68T05 .
1998 *CR Categories and Descriptors.* I.2.1 [**Artificial Intelligence**]: Learning – *Applications and Expert Systems - Games*.
*Key words and phrases.* Artificial Intelligence, Computer Poker, Adaptive Learning, Fictitious Play, Self-Play, Deep Reinforcement Learning, Neural Networks.

security, routing ([2], [3], [4]). Most of these real-world games involve decision making with imperfect information and high-dimensional information state spaces.

We have experienced the quick advancement of super-human Awe in perfect-information games like Chess and Go (AlphaGo Zero, [5]; LeelaChessZero [6]), but researchers have yet to reach the same progress in imperfect-information games (AlphaStar, [7]). An optimal theoretical solution to these games would be a Nash equilibrium i.e. a strategy no one can gain extra profit by deviating from it.

Fictitious play [8] is a popular method for achieving Nash Equilibria in normal-form (single-step) games. Fictitious Self-Play (FSP) [9] extends this method to extensive-form (multi-step) games. Neural fictitious Self-Play (NFSP, [10]) combines FSP with neural network function approximation. It is an effective algorithm and the first end-to-end reinforcement learning system that learns approximate Nash Equilibrium in imperfect information games without prior knowledge. It uses anticipatory dynamics; the agents choose their strategies from a mixture of average (supervised learning network) and greedy responses (Q-learning network).

With all of that said, it was proven that NFSP provides poor performance in games with large-scale search space and search depth [11], because it uses only crude data as input and its core aspect is represented by a Deep Q-Network which is offline; it doesn't make any real-time computations during the game. Solutions to these problems were proposed (MC-NFSP, [11]) that use Monte Carlo Tree Search instead. This, indeed, provides better and more stable performance but we are interested in a pure neural approach not using any brute force search methods. As we are going to apply this algorithm mainly to Poker, a game where intuition is key in winning, exhaustive search might not always be necessary. In this paper, we address this issue by adding real-time heuristics as features to the agents' field of view and by combining anticipatory dynamics with neural gradient play which yields, in theory, incremental better response search for our strategies. We test that in practice as well using as benchmark the performance against a certain common opponent.

Many AI bots have proven themselves to be above any human in no-limit Hold'em (Libratus [12], Pluribus [13]) but this does not mean that the game is completely solved. For that, we need a mathematical way of showing that the agent will definitely win money, given a certain interval of time or games, which was actually done with Cepheus [14] for the in-limit version. No-limit variant of Texas Hold'em is still considered unsolved in different formats to this day. In this paper, for the main agent we develop, we do provide a mathematical underpinning for the algorithm behind it, in the context of a 2-player zero-sum

game; this is later empirically validated through the experiments in which we successfully approach Nash Equilibrium.

Furthermore, this paper also highlights a direct comparison to some of our previously developed agents. For this, we refer to our previous published paper on this matter: A View on Deep Reinforcement Learning in Imperfect Information Games [15].

We empirically evaluate the agents in heads up computer poker games and explain how an agent trained this way can work even in a multiple-player scheme with some performance loss. As input, we use raw data, as an image of cards from the current visible board combined with two hand-crafted scalar inputs: hard coded rankings of card combinations and Monte-Carlo heuristics for assessing an approximate strength of the opponent hand. The best agent built (with our modest hardware) learnt a strategy close to human expert play.

## 2. Background

There are two main theoretical parts this research project is based upon - fictitious self-play in extensive-form games and reinforcement learning [1] . In this chapter, we aim to provide some mathematical underlying that is going to be referenced in the main chapters.

2.1. **Reinforcement learning.** Reinforcement learning (RL) [1] is widely considered as the third paradigm of learning where an environment is fundamentally defined and there are agent(s) that interact with it having a certain goal in mind. Hence, reinforcement learning can be viewed as a tool of solving optimization problems; these are usually modelled as a Markov Decisiion Process (MDP) [1]. Usually, in RL, optimization algortihms makes use of sequential experience. This is a form of history of states and actions that each agent possesses. Appropriately, it is modelled as transition tuples: $(r)$: $(s_t, a_t, r_{t+1}, s_{t+1})$. The goal is to maximize the rewards. To represent that, an *action-value function* $Q$ is used - defined as the expected gain of taking action $a$ in state $s$ and following the policy $\pi$: $Q(s, a) = \mathrm{E}^{\pi}[G_t|S_t = s, A_t = a]$. Here, $G_t = \sum_{i=t}^{T} R_{i+1}$is a random variable of the agent's cumulative future rewards starting from time $t$ [1]. Ideally, we would want to follow the action that gives the highest estimated value $Q$,that's why *Q-learning* [21] was introduced as a way to learn this greedy policy and replaying past experience. In order to approximate the *action-value function* (or any function for that matter), a wide and deep enough neural network can be employed which seems to be the preferred way nowadays of using Q-learning for solving more complex games: *deep Q network (DQN)* [16].

2.2. **Neural Fictitious Self-Play.** Neural Fictitious Self-Play [10] is a model of learning approximate Nash Equilibrium in imperfect-information games using deep learning.
At each iteration, the agents choose their best response (greedy strategy) with a DQN and update their average strategy by supervised learning through a policy network. That is done by storing datasets of each agent's experience in self-play as transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ in a memory $M_{RL}$ (designed for RL) and by storing agent's own behavior $(s_t, a_t)$ in a memory $M_{SL}$ (designed for supervised learning). If we set the self-play sampling in a way that an agent's reinforcement learning memory approximates data of an MDP defined by the other players' average strategy profile, then we can be sure that we find an approximate best response from an approximate solution of the MDP by reinforcement learning.
As we can see, the respective data necessary to train the neural networks through backpropagation is collected within the simulated games during the training process which is offline so it naturally has problems in on-policy games where we need to sample opponents' changing strategy while we play. To see how we can improve on this and take more into consideration the opponents' ever-changing strategies, we need to look deeper at how NFSP uses anticipatory dynamics [17] to stabilize the convergence around Nash Equilibrium points.
Define $\Delta(n)$ as a standard simplex in $R^n$, $v_i \in \Delta(n)$ being the $i$-th vertex and let $\mathrm{H} : \mathrm{Int}(\Delta(n)) \to R$ the entropy function $\mathrm{H}(p) = -p^T \log(p)$. In a two-player game, each player chooses its strategy $p_i \in \Delta(m_i)$, $m_i \in N^*$ and accumulates its reward according to the value-function: $V_i(p_i, p_{-i}) = {p_i}^T M_i p_{-i} + \tau \cdot \mathrm{H}(p_i)$, where$-i$, $i \in \{1, 2, ..., n\}$ refers to the complementary set$\{1, 2, ..., i-1, i+1, ...n\}$ [17] and $M_i$ is the game-dependent reward matrix. Consequently, we can define player $i$'s best response as a function $\beta_i : \Delta(m_{-i}) \to \Delta(m_i)$,$\beta_i(p_{-i}) = \arg\max\ V(p_i, p_{-i})$ and player $i$'s average response until step $k$ in the game as empirical frequencies $\pi_i(k) : N \to \Delta(m_i)$ of player $P_i$, [17].

In our previous work, we defined the differnt time abstractization of Fictitious Play (FP). Recall that in continuous time FP, we need to consider the derivative of the policy change over time:

$$\frac{d}{dt}\pi_i = \beta_i(\pi_{-i}(t)) - \pi_i(t), i = \overline{1,2} \quad (2)$$

Poker falls in this type of abstraction, in which each player has access to the derivative of his empirical frequency $\frac{d}{dt}\pi_i$. The strategy at moment $t$ can be defined as:

$$p_i(t) = \beta_i\left(\pi_{-i}(t) + \eta \tfrac{d}{dt}\pi_{-i}(t)\right),\ \eta \text{ positive parameter } (3)$$

We interpret this formula as a player choosing his best response based on current opponent's average strategy profile combined with a possible change of it that may appear in the future [15].
The authors of the study that we have used to borrow these mathematical notations (*anticipatory dynamics of continuous-time dynamic fictitious play* [17]) prove that for a good choice of $\eta$, the stability in Nash equilibrium points can be improved. Of course, this choice of $\eta$ is game-dependent. The challenge that comes with it though is the fact that the derivative cannot be directly measured and needs to be approximated or reconstructed by empirical frequencies measurements [15].
Recall the equation (3), subtracting $\pi_i$ from both sides and using (1) yields:

$$\frac{d}{dt}\pi_i = \beta_i\left(\pi_{-i}\left(t\right) + \eta\frac{d}{dt}\pi_{-i}\left(t\right)\right) - \pi_i\left(t\right)\,(4)$$

In *NFSP* [10], the authors chose a discrete time approximation of the derivative: ${\beta^{t+1}}_i - {\pi_i}^t \approx \frac{d}{dt}{\pi_i}^t$ which, if substituted in (4) yields:

$$p_i\left(t\right) \approx \beta_i\left(\pi_{-i}\left(t\right) + \eta\left(\beta_i\left(\pi_{-i}\left(t+1\right)\right) - \pi_{-i}\left(t\right)\right)\right) \Leftrightarrow$$

$$p_i\left(t\right) \approx \beta_i\left(\left(1-\eta\right)\pi_{-i}\left(t\right) + \eta\beta_i\left(\pi_{-i}\left(t+1\right)\right)\right)$$

That's how the authors reach the combined policy method: $\sigma \equiv \left(1-\eta\right)\hat{\pi} + \eta\hat{\beta}$ which was empirically porved to be successful for games like in-limit Texas Hold'em Poker.
However, a discrete time approximation does have its limitations, that is why we suggest using an approach that borrows elements from *dynamic gradient play* [17] in order to approximate the derivative taking into consideration the opponents' average strategies as well.

## 3. Developing the agents

We are going to address the technical details and the main process of building the self-play agents mentioned in the introduction. It is important to recall our last published research article on this subject, *A View on Deep Reinforcement Learning in Imperfect Information Game* [15] because we will use some of the agents developed there for direct comparison with the new ones. Only a short introduction of each one will be provided as for more details we recommend reading the original paper.

### 3.1. **Agent 1 (previously developed)** [15].

This first agent is a reinforcement learning free one, we built it as our own mini remake version of Loki [18] featuring betting decisions with card heuristics and opponent-modelling.

We constructed this agent mainly as an expert system at its core with heuristics for betting decisions and opponent-modelling for exploitations [15]. For opponent modelling, this agent uses 2 classifiers: a naïve Bayes classifier (to replicate the Bayesian analysis presented in the Loki paper) and a deep neural network with a CNN architecture, the input being represented as an image of the current board state alongside some scalar associated features.

3.2. **Agent 2 (previously developed)** [15]**.** This deep reinforcement learning agent learnt to play Poker by training with **Agent 1** from scratch. Its strategy of play combines the greedy strategy $\beta$ offered by the action-value function with the average strategy $\pi$ obtained though supervised classification. The second agent managed to learn Poker training with the first agent trying to consistently beat him, treating the opponent as part of the environment.
Therefore, it uses 3 neural networks. First, a *DDQN* system [19] with a *value network* $Q\left(s, a \,\middle|\theta^{Q}\right)$ for predicting the $Q$ values for each action based on data from $M_{RL}$. It trains through backpropagation using the *Bellman equation* with future $Q$ values obtained through a *target network* $Q'\left(s, a|\theta^{Q'}\right)$. Secondly, we use a *policy network* $\Pi\left(s, a \,\middle|\theta^{\Pi}\right)$ to define our agent's average response based on data from $M_{SL}$. We choose our main policy $\sigma$ from a mixture of strategies:$\beta = \varepsilon - greedy\left(Q\right)$ and $\pi = \Pi$: $\sigma \equiv \left(1-\eta\right)\hat{\pi} + \eta\hat{\beta}$, $\eta \in (0, 1]$. This actually represents the same approximation of *anticipatory dynamics* in *discrete time fictitious play* used in NFSP [10], but here we are using it to define our agent in a one-player game, we are not trying to approximate a Nash Equilibrium in this context. The other differences come from the model architectures, inputs and from how often we use each strategy of play to sample games. Moreover, unlike NFSP, we mainly considered a Poker game iteration to be just a hand of play here and reset the main policy accordingly.

3.3. **Agent 3 (our proposed approach in this paper).** Compared to the other two, the third agent, the main focus of this paper, shall decipher poker playing against itself using a new variant of fictitious self-play that employs deep learning.

To clarify, this agent will be based on self-play only, using deep neural nets, without any external help from other players for training and without brute force, real-time exhaustive search. This agent will learn by playing with itself, from scratch, both constantly trying to achieve better rewards. Below (figure 1), we can see the architecture of this self-play system and how the strategies are generated.

Like the Agent 2, we are devising the greedy and average strategies, this time through self-play, though, but we also have a reference to the opponent's

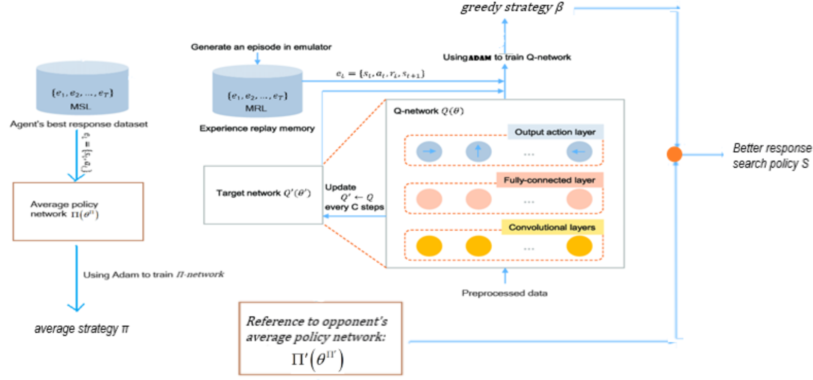


FIGURE 1. Agent 3, self-play system architecture

average strategy to construct a better response search. To understand how this is mathematically done, take the gradient of the value function:

$$\nabla V_i (p_i, p_{-i}) = M_i p_{-i}$$

We are interested in the differential equations system that defines the dynamic gradient play:

$$\tfrac{d}{dt}\pi_i = \mathrm{P}_\Delta [\pi_i (t) + M_i \pi_{-i} (t)] - \pi_i (t) \text{ with } i = \overline{1, 2},$$

where $\mathrm{P}_\Delta : R^n \to \Delta(n)$ is the projection on the simplex $\Delta(n)$: $\mathrm{P}_\nabla [x] = \arg\min_{s\in\Delta(n)} |x - s|$.

Therefore, we can obtain a parametrized approximation of $\frac{d}{dt}\pi_i$ using two forms of behavioral evolution of strategy of play in FP (DT – discrete time FP, GP – gradient play). Using the definition, we get:

$$\frac{d}{dt}\pi_{-i} = \frac{\pi_{-i}(t+\eta) - \pi_{-i}(t)}{\eta} \approx \pi_{-i}(t+1) - \pi_{-i}(t) \overset{(4)}{\Rightarrow}$$

$$\frac{d}{dt}\pi_i + \pi_i (t) = \beta_i (\pi_{-i} (t+1)) \overset{(*)}{\approx} {\beta_i}^{t+1}; i = \overline{1, 2}\,(5)$$

Let $S(t) \in \Delta(n)$ such that $S(t) = \mathrm{P}_\Delta [\pi_i (t) + M_i \pi_{-i} (t)]$ i.e.

$$|\pi_i (t) + M_i \pi_{-i} (t) - S(t)| < \varepsilon \text{ with } \varepsilon \text{ as small as possible.}$$

Then it follows that:

$$\frac{d}{dt}\pi_i + \pi_i (t) = S(t).$$

Combining this with (5) yields that for every $\rho \in [0, 1]$ we have:

$$\frac{d}{dt}\pi_i \approx \rho \left({\beta_i}^{t+1} - \pi_i (t)\right) + (1 - \rho)(S(t) - \pi_i (t)), i = \overline{1, 2}.$$

Substituting now $\frac{d}{dt}\pi_i$ in (3), we get the final formula:

$$p_i(t) = \beta_i \left( (1-\eta)\, \pi_{-i}(t) + \eta \left( \rho \cdot {\beta_i}^{t+1} + (1-\rho) \cdot S(t) \right) \right)$$

which means our agent can choose their actions from a mixture of strategies:

$$\sigma \equiv (1-\eta)\, \hat{\pi} + \eta \left( \rho \hat{\beta} + (1-\rho)\, \hat{s} \right).$$

The motivation behind this choice is that the evolution of the GP strategy follows a better response search, adjusting the strategy of play in the direction of the gradient from the empirical frequencies of the opponent. Thus, using this form, especially in a game with imperfect information, where the best answer is harder to find, it is important that we don't stagnate and we always try to find a better solution than the current one (and if we have already found the best solution then the gradient should suggest so).
We want to favor finding the best response though, that is why are going to set the$\rho$ parameter to be:

$$\rho \approx 1 - \eta + \varepsilon \text{ with } 0 < \varepsilon < 2/100.$$

Below, we present *Algorithm 1*, the main algorithm that agent 3 uses to get learn Poker from self-play.

---

**Algorithm 1 — Agent 3, reinforcement learning (self-play) agent with fitted Q-learning**

---

**for** 1:*no_games* **do**
  Initialize new game $G$ and execute agent via RUN_AGENT for each player in the game
**end for**
**function** RUN_AGENT(G)
  Initialize replay memories $M_{RL}$ (circular buffer) and $M_{SL}$ (own behaviour reservoir)
  Initialize average-policy network $\Pi(s, a|\theta^{\Pi})$ with random weights $\theta^{\Pi}$
  Iniitalize opponennt average-policy network $\Pi'(s, a|\theta^{\Pi'})$ with random weights $\theta^{\Pi'}$
  Initialize action-value network $Q(s, a|\theta^{Q})$ with random weights $\theta^{Q}$
  Initialize target network with weights $\theta^{Q'} \leftarrow \theta^{Q}$
  Initialize parameters $\eta, \rho$.
  **for each** episode **do**

$$S_{i,t} = \begin{cases} \begin{cases} \epsilon - greedy(Q) & \text{with probability}\rho \\ S = P_{\Delta}Pi + Q_{extended(m_i,m_i)} \cdot \Pi' & \text{with probability}1-\rho \end{cases} & \text{with probability}\eta \\ \\ \Pi & \text{with probability}1-\eta \end{cases}$$

Observe initial information state $s_1$ and reward $r_1$
**for** t=1,min_replay_memory_size **do**
Sample action $a_t$ from policy $\sigma$
Execute action $a_t$ in emulator and observe reward $r_{t+1}$ and next information state $s_{t+1}$
Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $M_{RL}$
**if** agent follows best response policy $\sigma = \beta(= \epsilon - greedy(Q))$ **then:**
Store behaviour tuple $(s_t, a_t)$ in supervised learning memory $M_{SL}$
Update $\theta^{\Pi}$ with gradient descent on loss
$L(\theta^{\Pi}) = E_{(s,a)\sim M_{SL}}[KLDivergence\Pi(s, a|\theta^{\{Pi\}})]$
Update $\theta^Q$ with gradient descent on loss
$L(\theta^Q) = E_{(s,a,r,s')\sim M_{RL}}[(r + max_{a'}Q(a', a'|\theta^{Q'}) - Q(s, a|\theta^Q))^2]$
Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^Q$
**end for**
**end function**

---

We are using 3 deep neural networks: a DDQN [19] system to approximate the action-value function and a policy network to approximate the player's own average behaviour. The architecture for these neural nets for the two strategies (greedy and average) are the same. The input is represented by a *17x17x9* 3D array containing the images of the last two board states and the scalar features that we mentioned the *Developing The Agents* section – note that this is the same input as the one Agent 2 uses. As we said in [15], we add the last board state to the input because of the inspiration from *AlphaGo Zero* [5] interpreting it as an *attention mechanism.* The actual architecture of the networks is represented as a CNN with 4 layers of convolution. 2 MaxPooling and 1 fully connected as hidden layers. For the reinforcement learning part, we use *MSE* as loss (together with the Bellman equation to calculate the value

of a state to get the predicted part). For the policy network we use *Kullback–Leibler Divergence* between two probability distributions as it is usually a good loss measurement, also used by the creators of AlphaGo Zero.

## 4. Experiments

*The computer code is available at: link (backup directory for the whole project). Everyone can play against the agents at request at: poker.ptidor.com.*

We are mainly testing the algorithm on heads-up no-limit variant of the game of Poker. The choice of heads-up is also determined by the limited resources of this research project. For evaluation, we are going to measure the performance of each agent against previously developed ones and some generic players that we previously defined in [15]. We also paired the final agent against a human player to get an intuition of its level of play in real world.

4.1. **General specifications.** The format we are using for the games is heads-up, no-limit with **100** chips as starting stack and **5** chips small blind. To evaluate the agents, we use two metrics: ***average stack*** over a fixed number of games and ***mbb/h*** (milli big blinds per hand) [15]. A mili big blind per hand is 1/1000 of a big blind, if a player wins a big blind it gets 1000 points, if a player wins a small blind it gets 500 points (and it loses the same amounts for the negative case). So, a player that always folds is expected to lose at a rate of 750 mbb/h – we obtain this figure by taking the mean over the big and small blinds. Therefore, the intuition is that the values for a mbb/h metric will usually stay in the interval [-750, 750]. This metric is a standard for Poker research nowadays and many other studies ([10], [13], [14], [11]) make use of it. It is regarded that a human professional player would aim for winnings of **50 mbb/h**, at a minimum.
For comparison reasons, we use a couple of generic Poker players:

1. A player that only calls *(Call_player)*
2. A player that chooses its actions randomly: 3 times out of 5 calls and in the remaining it can equally raise with a random amount or fold *(Random_player)*
3. A player that chooses its actions based only on Monte-Carlo simulations and not look-up tables (*HeruristicMC_player*)

4.2. **No-limit Texas Hold'em Poker.** We want our self-play agent to be unbeatable in the long run, so now an episode will be represented by a game (which can have several hands) and not an only hand of play as we considered

in Agent 2. Also, *Agent 3* will receive an immediate reward of **0** for each move and only at the end of a hand / end of a game, he will receive a non-zero reward depending on how many chips it won. Thus, Agent 3 will not be penalized immediately for a raise of 100 (all-in), for example, but if he loses that hand, then he will receive a reward of negative 100 at the end of it, which is very high. In this way, we tell the AI that it doesn't matter what moves he chooses as long as the reward at the end of the game is maximized.
We let the algorithm train for roughly 3 days straight (80 hours to be exact). For compute, we used an *NVIDIA Tesla T4 Workstation* with *32*GB of RAM and a *NVIDIA GTX 1050ti* with *16*GB of RAM. However, at inference, the artificial players can be run on day-to-day hardware.
The algorithm descendance to Nash-Equilibrium can be observed in figure 2. Parameters $\eta$ and $\varepsilon$ were set to 0.1, 0.9, respectively, $\rho$ was set to 0.92, max length for $M_{RL}$ to 200k and for $M_{SL}$ at 1m. We make one stochastic gradient update of mini-batch size of 256 per network for every 64 steps and the target network parameters were reset every 1000 updates.
The choice of the hyper-parameters (apart from $\rho$) was inspired by the NSFP paper [10]. Little effort was put into experimenting with hyper-parameter search because of time constrains and the fact that similar hyper-parameters already existed within the NSFP context. However, note that even in this paper (NSFP), the choice of hyper-parameters wasn't clearly reasoned. The architecture of the neural networks was not explored in this paper but it was inspired by standard image classification neural networks.
In order for the copies of the same agent to be in Nash-Equilibrium, we have to observe a convergence towards 0 of the difference in modulus in winnings (mbb/h - aggregated over a batch of recent games) of the two players. This is actually what we plot in figure 2 and as we can see, that measurement value narrows down and starts to approach 0 after the 250's batch. Note that we calculate the mean of the absolute difference in winnings over the most recent 500 games for the y-axis in the figure. That's why we have 300 iterations for 150k games.

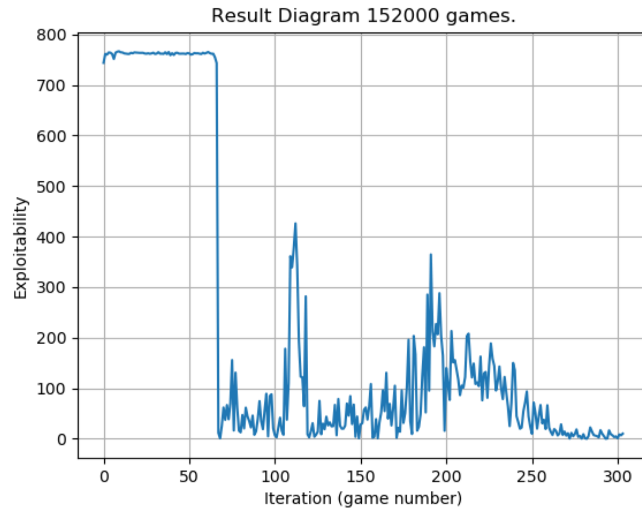


FIGURE 2. Training evolution (mbb/h) of Agent 3 ($\rho$= 0.92), with hand-crafted metrics as input, in 3 days straight.

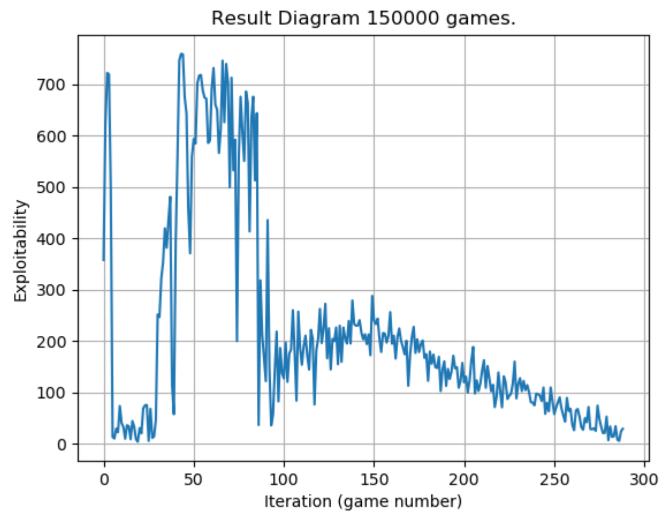


FIGURE 3. Training evolution (mbb/h) of Agent 3 ($\rho$= 0.92), without hand-crafted metrics as input, in 3 days straight.

There are obvious spikes that disturb the balance as we can see around the 100s and 200s iteration, this is because both copies are continuously learning

by playing one another and it is possible that one learnt a clever strategy faster and it is able to exploit that for a brief moment. Of course, one can argue that huge spikes like these can appear again if we let it train for more iterations. This is possible; however, it is unluckily - note that for the last approximatively 50 iterations (or 25,000 games) the mean absolute difference in mbb/h stayed steady in the range 0 to 20, which wasn't the case until then. Moreover, this range is good enough to call this an approximation of the Nash-Equilibrium because if we recall the critical value 50 mbb/h that a professional player usually aims to achieve in a match, everything below that would still be considered indecisive.
We also trained the algorithm with raw data, without hand-crafted input metrics, just like in NFSP [10], to see if the algorithm still converges without any prior knowledge of the domain (figure 3). And if so, how does it compare to the version above in which we are actually using solid prior knowledge of the game?
After the same amount of training time, it seems the algorithm still converges to approximate Nash-Equilibrium, but slower than our main proposed version. We base this claim on the range of the y-axis values for the last 50 iterations. It also concludes a little smaller number of games in 72 hours. This experiment does seem to suggest that hand-crafted metrics do really help a self-play algorithm train better.

4.2.1. *Experimenting with an expert Poker player.* For this experiment, I've invited a semi-professional human Poker player, *Serban.* He is very experienced with the game, playing constantly on real high money stakes but lacks the tournament play.

He played **56** hands against our agent, from *figure 2*, (during a 10-game match) and the results were crushing. our agent recorded winnings of **241.07 mbb/h** with the final score **7-3**.

*The human player said he was very impressed with the style of play of our agent but he recognized some mistakes during the match regarding the preflop stage of the game, which can be very costly during a professional match. Mainly, the agent does not recognize very weak cards in the preflop, such as 7-3, at which point he should not call for a raise.*

A temporary solution could be a Monte-Carlo search, which immediately draws attention to very weak combinations of cards at any stage of the game. Indeed, this version is still not perfect, or close to perfect, but training on more iterations should strengthen our AI bot considerably. It is an important victory, though, all things considered.

4.2.2. *Comparison with NFSP and other artificial Poker players.*

| Results using greedy + average strategy against Agent 2 | | | | |
|---|---|---|---|---|
| Player | No. hours trained | No. Games Played | Final Average Stack | Winnings (mbb/h) |
| Agent3_GP | 6 | 250 | 117.83 | 263.37 |
| Agent3_GP | 11 | 250 | 117.48 | 318.51 |
| Agent3_GP | 17 | 250 | 117.1 | 338.82 |
| Agent3_GP | 45 | 250 | 110 | 340.18 |
| Agent3_DP | 11 | 250 | 120.71 | 318.93 |
| Agent3_DP | 17 | 250 | 115.43 | 309.82 |
| Agent3_DP | 45 | 250 | 99.2 | 192.55 |
| Agent3_GP | 6 | 1000 | 118.45 | 248.35 |
| Agent3_GP | 17 | 1000 | 116.54 | 356.17 |
| Agent3_GP | 45 | 1000 | 111.9 | 361.08 |
| Agent3_DP | 11 | 1000 | 116.42 | 299.03 |
| Agent3_DP | 45 | 1000 | 102.3 | 234.22 |

TABLE 1. Results of different versions of Agent 3 vs Agent 2.

Since we want to test the effect of that better response search through gradient play proposed in the theoretical part, we will analyze the behavior / performance of an Agent 3 trained against a copy of itself taking into account the policy $\hat{s}$, $(\rho < 1)$ and the behavior performance of an Agent 3 trained against a copy of itself without regard to the policy $\hat{s}$, $(\rho = 1)$, as of *Algorithm 1.* We will therefore call these two agents: **Agent3_GP**, **Agent3_DP**, from gradient play, discrete play respectively (which refers to the method used to approximate the CDP derivative). Note that **Agent3_DP** is a theoretically a replica of NFSP.
We tested (table 1) multiple versions of these agents against Agent 2 (the one that beat an amateur human player). In this match-up, it is easier to see the difference between the two versions of Agent 3. In 250 matches played against Agent 2, both variants won, but the one that uses better response search exceeds the threshold of 320 mbb / h, and the situation improves when we increase the number of iterations. For some reason, the performance of Agent3_DP decreases at 45 hours compared to less trained versions. This trend remains consistent for the experiment with 1000 games as well, in which Agent3_GP reaches over 360 mbb/h in winnings but Agent3_DP can't cross 300mbb/h.
We need to mention that Agent3_GP took 6h to train for 50k games, whilst Agent3_DP took 11h, that's why we have no measurement for Agent3_DP for less than 11 hours. Note that Agent3_GP consistently beats Agent 2, in both

experiments (250 and 1000 games, respectively), this makes Agent 3 take the status of the best agent developed so far, after only **6** hours of **self-play**!
We have repeated the experiments many times to assure the consistency of the results, there is a statistical error of around +/- 4.5 in terms of average stack and around +/- 40 for mbb/h for the 250 games case. These figures get roughly halved for the more stable experiments with 1000 games played.

It is important to clarify that this does not necessarily mean that Agent3_DP is definitely worse; however, we have established in the introduction section that such a benchmark will be used to draw interpretations. Agent 2 is the previous best agent we have developed that can rival amateur human play [15], so it is a decent artificial opponent for these 2 agents. It is good practice to evaluate poker bots against each other as we can make use of a bigger amount of sample games (compared to matches against humans) and we can also compute statistical significance.
Out of curiosity, we paired up Agent3_GP after 30 hours of training against Agent 1. The results are not surprising at all, getting a win rate of **88.46%** and an average stack of **175** after **130** games against the expert system with neural opponent modeling.

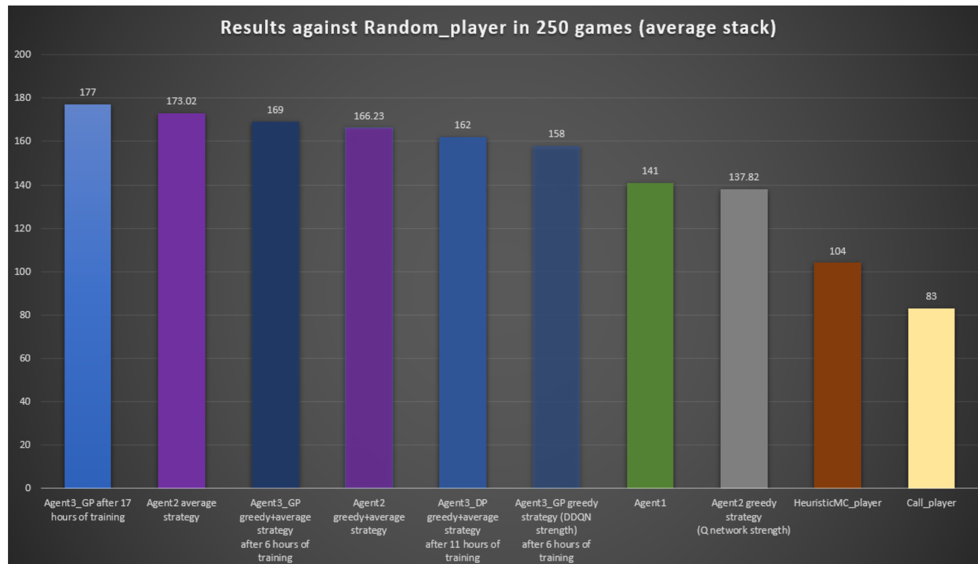


FIGURE 4. Results of some previous players against Random_player compared to Agent 3; statistical error +/-4.5

Next up, we compared the performance of Agent3_GP and Agent3_DP against the Random_player. Both versions are the ones trained on 50k games - this is an important threshold as both agents seem the overpower all the other ones after crossing this limit.
In figure 4, we can observe the performance of most of the agents we have tested during our study (against the Random_player). It is clear that both Agent3_GP and Agent3_DP crush in the benchmarks, however, Agent3_GP reaches almost 180 average stack in 250 games, which hasn't been done by any of our agents until now. This is another bonus point for the better response search technique that Agent3_GP uses.
What is very impressive here (figure 4) is the fact that we used the version of Agent 2 that trained with Random_player, having as sole objective to defeat it. Although Agent 3 had n**o interaction** with Random_player, learning the game of poker only through self-play, he achieves a performance almost identical to that of Agent 2, even surpassing the performance of all the other deep reinforcement learning agents, after just **17 hours** of training!

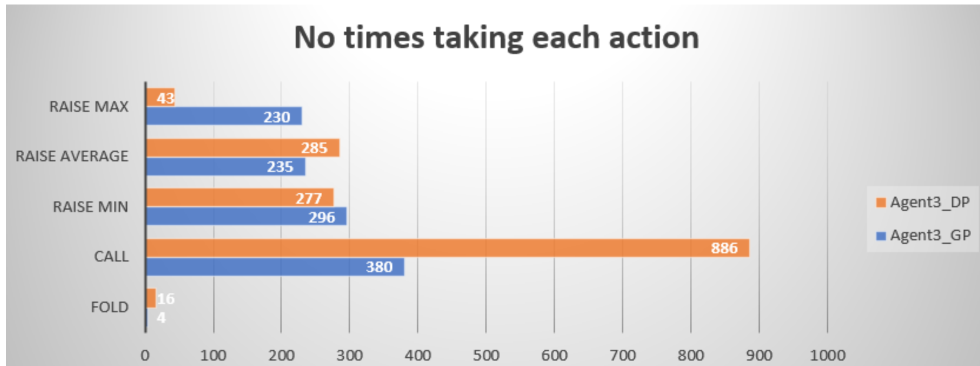


FIGURE 5. Agent 3 play style in 250 games vs Random_player

This match-up was also an opportunity to study the differences between Agent3_GP's style of play and Agent3_DP's. Agent3_DP plays much safer and is much more reserved about a raise, mainly choosing to wait through calls, very rarely choosing to go all-in (figure 5). Instead, Agent3_GP is much more aggressive, bouncing back between calls (predominant action) and raises.
The proposed approach can be adapted to play a multi-player Poker game. Although it may lose performance compared to the heads-up variant, we can make a small change in the inputs that are fed to the predict function to get the next action. The only input components that we use, relevant to a multi-player game, are the average estimated opponent strength, which can be recalculated with respect to the number of players through Monte-Carlo

simulations and the opponent's stack which can be replaced with the average stack of all the opponents.

## 5. Discussion and further research

Although the results looked pretty successful, it is very hard to correctly assess the level of play of the best agents. Until we test them against a professional player or top computer programs like *Hyperborean*, We can't know for sure that they are indeed at top human level. Furthermore, due to time and hardware constrains, we couldn't experiment on more iterations, we can maybe descend even more closer to a Nash-Equilibrium in optimal conditions. Improvements can also be made regarding the format of the game. All the agents were trained in heads-up, no-limit, 100-100 starting stack with 5 small blind formats, but for more general play, it is recommended to consider the small blind as percentage of the starting stack.

## 6. Conclusions

We have showed the power and utility of deep reinforcement learning in imperfect information games and we have developed an alternate new approach to learning approximate Nash equilibria from self-play that does not use any brute force search and only relies on the *intuition* provided by deep neural networks. When applied to no-limit Hold'em Poker, training through self-play drastically increased the performance compared to fictitious play training with a normal-form singe-step approach to the game. The experiments have shown the self-play agent to converge reliably to approximate Nash equilibria with crude data and limited hand-crafted metrics as input and the final artificial player can rival expert human play.

## References

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[2] Lambert Iii, Theodore J., Marina A. Epelman, and Robert L. Smith. "A fictitious play approach to large-scale optimization." Operations Research 53.3 (2005): 477-489.

[3] Nevmyvaka, Yuriy, Yi Feng, and Michael Kearns. "Reinforcement learning for optimized trade execution." Proceedings of the 23rd international conference on Machine learning. 2006

[4] . Urieli, D. and Stone, P. (2014), "Tactex'13: a champion adaptive power trading agent." In Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems, pages 1447–1448.

[5] Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

[6] Gary Linscott, "Leela Chess Zero", 2018.

[7] Arulkumaran, Kai, Antoine Cully, and Julian Togelius. "Alphastar: An evolutionary computation perspective." Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2019.

[8] Brown, George W. "Iterative solution of games by fictitious play." Activity analysis of production and allocation 13.1 (1951): 374-376.

[9] Heinrich, Johannes, Marc Lanctot, and David Silver. "Fictitious self-play in extensive-form games." International Conference on Machine Learning. 2015.

[10] Heinrich, Johannes, and David Silver. "Deep reinforcement learning from self-play in imperfect-information games." arXiv preprint arXiv:1603.01121 (2016).

[11] Zhang, Li, et al. "Monte Carlo Neural Fictitious Self-Play: Approach to Approximate Nash equilibrium of Imper-fect-Information Games." arXiv preprint arXiv:1903.09569 (2019).

[12] Noam Brown, Tuomas Sandholm, "Safe and Nested Subgame Solving for Imperfect-Information Games", 2017. 31st Con-ference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[13] Noam Brown1, Tuomas Sandholm, "Superhuman AI for multiplayer poker", 2019. Brown et al., Science 365, 885–890

[14] Bowling, Michael, et al. "Heads-up limit hold'em poker is solved." Science 347.6218 (2015): 145-149.

[15] Pricope, T.V.. A View on Deep Reinforcement Learning in Imperfect Information Games. Studia Universitatis Babeș-Bolyai Informatica, [S.l.], v. 65, n. 2, p. 31-49, dec. 2020. ISSN 2065-9601.

[16] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[17] Jeff S. Shamma and Gurdal Arslan, "Dynamic Fictitious Play, Dynamic Gradient Play, and Distributed Convergence to Nash Equilibria.", 2005.

[18] Denis Richard Papp, "Dealing with Imperfect Information in Poker.", 1998.

[19] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[20] Yakovenko, Nikolai, et al. "Poker-CNN: A pattern learning strategy for making draws and bets in poker games using con-volutional networks." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 30. No. 1. 2016.

[21] Watkins, C. J. and Dayan, P. "Q-learning", 1992. Machine learning, 8(3-4):279–292.

The University of Edinburgh, School of Informatics, 10 Crichton St, Newington, Edinburgh EH8 9AB, United Kingdom

*Email address*: `T.V.Pricope@sms.ed.ac.uk`