Neural Fictitious Self-Play with Perfect and Imperfect Information

Chris Ji, Elbert Lai
cc2ji@uwaterloo.ca, e6lai@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Many real-world applications of AI can be simplified as games with imperfect information. Much prior work has been done developing algorithms to train AI on both perfect and imperfect information. No work has been done to compare the performance of such models. In this paper, we use Pokemon Go Battle League (GBL) as an environment to compare the performance of a Neural Fictitious Self-Play (NFSP) model trained with perfect information, and a NFSP model trained with imperfect information. The model trained on perfect information outperformed the model trained on imperfect information. Hence, one should always look to train AI with as much information as possible.

Introduction

Games have long been an interest in the field of artificial intelligence (AI). With definitive rules and boundaries, games provide researchers the perfect combination of simple problems and limitless solutions. Many real world applications of AI, such as self-driving cars, travel planning, and facial recognition apps, can be described similarly: simple problems with a myriad of different solutions. The development of AI algorithms for recreational games allows for those same algorithms to be scaled to the complexity of the real world.

One such complexity is the problem of imperfect information. Many AI in the past have been developed that perform near optimally in perfect information games, even with substantial search spaces and complex like Chess and Go (Campbell, Hoane, and hsiung Hsu 2002; Silver et al. 2016). With imperfect information games, such as StarCraft II and Clash Royale, similarly skilled AI have been developed using different methods, but all such AI learned from the perfect information in some way (Vinyals et al. 2019; Boney et al. 2020). In the real world, there are many instances where perfect information is not available to us, such as financial trading, media recommendations, and AI in healthcare. As such, being able to effectively and efficiently train an AI that can navigate such spaces is crucial.

In this paper, we evaluate the performance of Neural Fictitious Self-Play (NFSP, Heinrich and Silver 2016) with imperfect information versus NFSP with perfect information. Of interest is quantifying if NFSP trained with perfect information performs superiorly to NFSP trained with imperfect information.

NFSP is a combination of two neural networks: Fictitious Self-Play (FSP, Heinrich and Silver 2015) and neural network function approximation. FSP is an extension of Fictitious play (Brown, 1951) to multi-step games. Neural network function approximation allows the agent to average over its own historical strategies. We utilize NFSP to train an agent that can battle in a simplified version of Pokemon Go Battle League (GBL).

GBL is a player versus player game consisting of matches that last up to five minutes. Before the match, each player must select three unique Pokemon to be on their team, each having, among other characteristics, one fast attack, two charged attacks, and stamina. A match is finished when the stamina of all three of one player's Pokemon reaches 0 ("faints"), or five minutes have passed. Each fast attack generates energy, charged attacks cost energy, and each player starts the game with two shields, which can be used to negate an opponent's charged attack's damage.

Solving GBL completely requires also solving the metagame of choosing the right Pokemon, which we will not do. Instead, we make teams randomly generated from a set of top rated Pokemon on PvPoke.com. To play GBL, an AI faces several challenges:

- **Exploration.** It is not optimal to always try to do the most damage all the time. For example, if an opponent's Pokemon is at very low stamina, skilled players will use fast attacks until it faints, instead of wasting energy on a charged attack. An AI will have to learn when it is best to use charged attacks to maximize their damage per energy used.
- **Partially-observed state.** The opponent's Pokemon, and their attacks, are only revealed once they are used. Also the charged attack an opponent is using is hidden, leading to a popular strategy where most Pokemon have one lower cost, lower damage charge attack (a "bait" attack), and one higher cost, higher damage charge attack (a "nuke" attack), and players will use fast attacks until they have enough energy to use the nuke, but use the bait instead, to lure the opponent to use a shield.
- **Non-transitive game.** Each Pokemon also has 1-2 types,

and all attacks have a type. Certain types do more/less damage to some types, leading to a rock-paper-scissors between types. There is no obvious deterministic best strategy when choosing which Pokemon to be in battle against an opponent's Pokemon.

To evaluate the performance of NFSP with perfect information compared to imperfect information, we pit both models against an agent that selects actions randomly, and compare the winrates of the models. To train the models, we recreate a simplified version of GBL under an object oriented programming framework. More detail can be found in the methodology.

Main Results

In GBL, the win rate of an agent trained by NFSP with perfect information, against an agent that selects completely random actions, is significantly greater than an agent trained with imperfect information under a large enough sample size.

Contributions

- We compared NFSP trained using perfect and imperfect information states
- We implemented NFSP for GBL
- We recreated GBL in an object oriented programming framework

Related Work

Fictitious play (FP) (Brown, 1951) is an iterative way to find the game value, through each player choosing the best response to their opponent's actions. FP has been proven to converge to the theoretical game value. Fictitious self play (FSP) (Heinrich and Silver, 2015) is a framework for AI that involves choosing between the best response and average response, and replaying their own behaviour to learn their own average strategy.

OpenAI developed AIs to play DotA2 and StarCraft (Berner et al., 2019; Vinyals et al., 2019), both highly complex video game with imperfect information, high dimensional action spaces, and long time horizons. Both OpenAI Five (the AI for DotA2) and AlphaStar (the AI for StarCraft) were trained largely using reinforcement learning and self-play. OpenAI Five and AlphaStar were both hugely successful, with OpenAI Five sporting a 99.4% winrate out of 7,215 games against random humans, and AlphaStar ranking above 99.8% of officially ranked players. However, during training, AlphaStar used a value function evaluated on complete information. In some experiments, AlphaStar's winrate against a control was 22% when trained with imperfect information; with perfect information: 82%. However, both OpenAI Five and AlphaStar were trained using over 100,000 hours of self-play time, a computational feat not available to most.

Boney et al. (2020) developed an AI for Clash Royale, another game with imperfect information. However, Boney et al. tried a different approach. Both OpenAI Five and AlphaStar used model-free reinforcement learning algorithms, whereas Boney et al. used a model-based planning approach. It consists of an oracle planner, an agent that performs self-play tree search on the complete information to compute actions for the follower agent, which learns the oracle planner's actions through supervised learning on its partial observations.

NFSP can learn approximate Nash equilibrium in games, but it relies on a Deep Q-Network, which is difficult to converge in the presence of large search scale and deep search depth. Zhang et al. (2019) develop Monte Carlo Neural Fictitious Self Play (MC-NFSP) and Asynchronous Neural Fictitious Self Play (ANFSP) for this problem. MC-NFSP combines Monte Carlo tree search with NFSP, and Zhang et al. prove it can converge to approximate Nash equilibrium with deep search depth. ANFSP accelerates and stabilizes training. Han, Zhou, and Duan (2021) combine NFSP and KR-UCT (an algorithm utilizing Monte Carlo trees and kernel regression) to play a digital curling game, which features a continuous action space. Pricope (2021) combines NFSP with neural gradient play to improve performance, and show their algorithm still converges to a Nash equilibrium, even with limited training and hardware.

On the topic of comparing learning with perfect information and imperfect information, related literature was not found. There are not many situations where one could have access to complete data, and choose to train on incomplete data. However, in the context of reinforcement, there are a number of ways to deal with incomplete data. In batch Q-learning, Lizotte et al. shows that using Bayesian multiple imputation on the observed data results in less bias than complete case analysis. Awan et al. (2022) outline a reinforcement learning approach to impute missing data. Mai et al. (2019) use inverse reinforcement learning, studying an agent's value using its behaviour, to compute log-likelihood with incomplete data. The training is done using a system of linear equations that does not depend on the amount of missing data.

Methodology

Setup

A battle consists of two teams of 3 Pokemon each. Each Pokemon has type, attack, defense, and stamina characteristics. Each Pokemon also has 1 fast attack and 2 charge attacks. The fast attack has damage, energy generated, turns taken per attack, and type characteristics. The charge attacks has damage, energy cost and type characteristics. We translate this setup into states and actions for reinforcement learning.

Each state consists of

- the type, current stamina, and energy levels of your 3 Pokemon
- types, attacks and current stamina of the opponent's Pokemon, which is unknown until they have switched all three in
- the number of and which Pokemon
- the 2 Pokemon currently battling (yours and your opponent's)

The set of actions are:

- no action
- fast attack
- either of the charged attacks (if you have enough energy)
- switch to either of the other two Pokemon (if they are alive)

A class diagram illustrating how we set the game up is shown in Figure 1.

Data

The base stamina, attack, and defense of Pokemon in Pokemon Go are easily accessible online. Along with each Pokemon having base stamina, attack, and defense, they are also affected by each Pokemon's individual values (IVs), which can range from 0-15 for each stat. We used PvPIVs.com to obtain the individual values (IVs) that resulted in the highest stat product of Pokemon, and used those IVs for our Pokemon.

We used PvPoke.com to get the 20 highest overall ranked Pokemon for PvP (great league). PvPoke provides an open-source tool for simulating, ranking, and building teams for GBL.

Algorithm

We will be implementing two versions of the Neural Fictitious Self-Play (NFSP) algorithm (Algorithm 1), introduced by Heinrich and Silver (2016), where the models are given complete and incomplete state information respectively.

Henirich and Silver introduced a series of models to approach extensive form games, a model of sequential interaction involving multiple players, and fictitious play, a game-theoretic model of learning from self-play. They first introduce Full-Width Extensive-Form Fictitious Play (XFP), which enables fictitious players to update strategies in behavioural, extensive form, resulting in linear time and space complexity. Then, they approximate XFP with Fictitious Self-Play (FSP) a sample- and machine learning-based class of algorithms. FSP uses reinforcement to compute best responses and supervised learning to compute the average strategy updates. FSP agents generates datasets of their experience. The agent stores two memories. $\mathcal{M}_{RL}$ stores transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ for reinforcement learning and $\mathcal{M}_{SL}$ stores its own behaviour, (s_t, a_t), for supervised learning. Sampling is set up for self-play so that $\mathcal{M}_{RL}$ approximates the data of a Markov Decision Process (MDP) defined by other players' average strategy profile, so that an approximate solution to the MDP is an approximate best response. $\mathcal{M}_{SL}$ approximates the agents average strategy, and is learned using supervised classification.

NFSP combines neural network function approximation with FSP, and trains two neural networks. $Q(s, a|\theta^Q)$ predicts action values from $\mathcal{M}_{RL}$ using off-policy reinforcement learning, while $\Pi(s, a|\theta^\Pi)$ learns to imitate past behaviour.

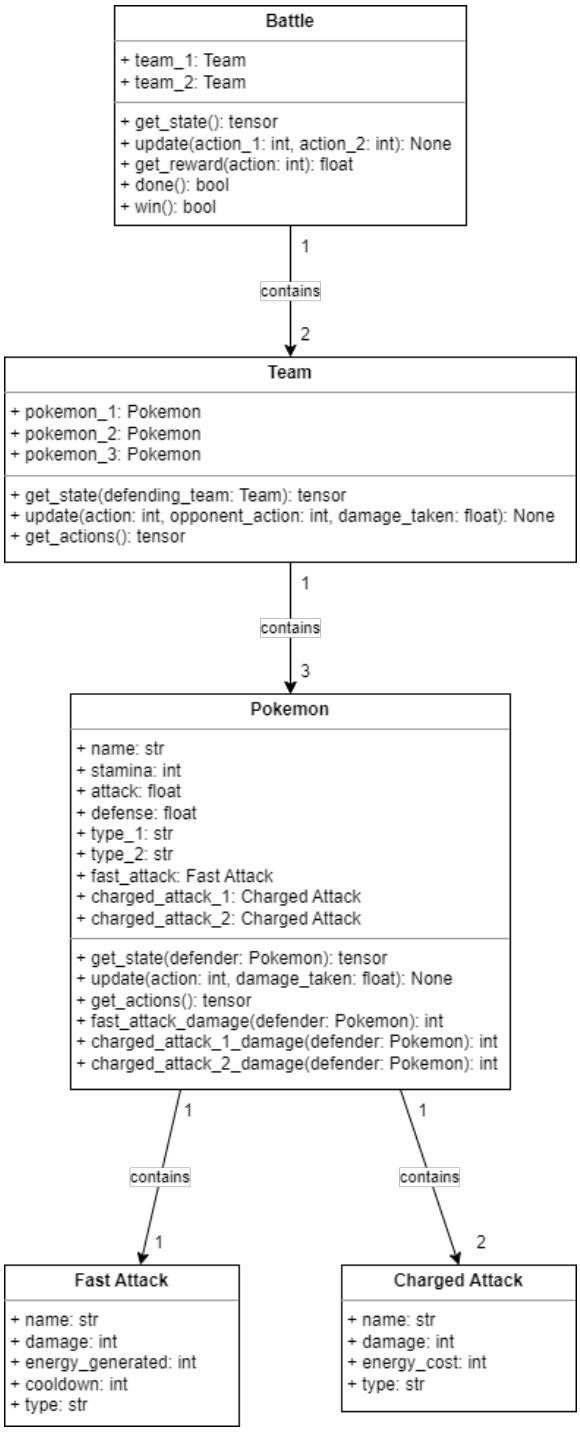

Figure 1: Class diagram of GBL

Algorithm 1 Neural Fictitious Self-Play (NFSP) with fitted Q-learning

Initialize game Γ and execute an agent via RUNAGENT for each player in the game
function RUNAGENT(Γ)
  Initialize replay memories
  Initialize average-policy network $\Pi(s, a|\theta^{\Pi})$ with random parameters θ^{Π}
  Initialize action-value network $Q(s, a|\theta^{Q})$ with random parameters θ^{Q}
  Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
  Initialize anticipatory parameter η
  for each episode **do**
    Set policy

$$\sigma \leftarrow \begin{cases} \epsilon\text{-greedy}(Q), & \text{with probability } \eta \\ \Pi, & \text{with probability } 1-\eta \end{cases}$$

    Observe initial information state s_1 and reward r_1
    for $t = 1, T$ **do**
      Sample action a_t from policy σ
      Execute action a_t in game 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 memory $\mathcal{M}_{RL}$
      if agent follows best response policy $\sigma = \epsilon$-greedy(Q) **then** Store behaviour tuple (s_t, a_t) in supervised learning memory $\mathcal{M}_{SL}$
      Update θ^{Π} with stochastic gradient descent on loss

$$L(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log \Pi(s, a|\theta^{\Pi})\right]$$

      Update θ^{Q} with stochastic gradient descent on loss

$$L(\theta^{Q}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s', a'|\theta^{Q'}) - Q(s, a|\theta^{Q})\right)^2\right]$$

      Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$

Results

Design

Initially we planned on following the typical reward structure of 1 for winning and 0 for losing. However, we found that since exploration takes too long to finish a match, many epochs would end with no reward, resulting in our model not learning. We incentivised the model to use the various attack moves by adding normalized rewards proportional to the damage each move made, and found that this sped up training much more.

States were structured slightly differently for the agents with perfect and imperfect information. The agent with complete information could see the damage each of its Pokemons' attacks would do on all opposing Pokemon, while the agent with incomplete information could only see the potential damage on the opponent's active Pokemon.

There were four parameters to tune: batch size, learning rate, weight decay, and epoch size. We first used grid search on the first three with a lower epoch size (100), and chose the batch size, learning rate, and weight decay which resulted in the least loss. The grid we chose was Batch Size $\in \{2^4, 2^5, 2^6, 2^7, 2^8\}$, Learning Rate $\in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$, and Weight Decay $\in \{10^{-5}, 10^{-6}, 10^{-7}\}$. A plot showing the results of grid search is shown in Figure 2. The final hyper-parameters we chose were batch size of 2^5, learning rate of 10^{-5}, and weight decay of 10^{-7}.

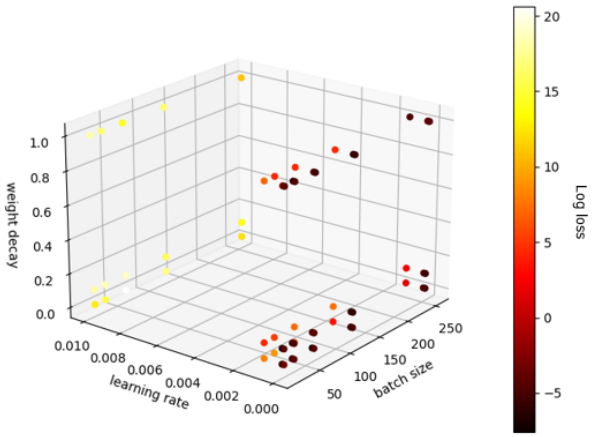

Figure 2: Results of the grid search on batch size, learning rate, and weight decay. The values of (batch size, learning rate, weight decay) which resulted in the minimum loss was $(2^5, 10^{-5}, 10^{-7})$.

We then used the resulting batch size, learning rate, and weight decay to tune epoch count, and chose the epoch size which resulted in the lowest loss, and also resulted in the best performing agent (Epoch Count $= 400$). A plot of a moving average of the loss (where every point in the plot is the mean of the surrounding 20 points) vs epoch count is shown in Figure 3. Note that the initial run in of the approximately first 60 epochs are due to the replay buffer filling up. Our final log loss was approximately 10^{-3}.

Performance of the models

We trained two agents, `agent1` was trained on perfect information, and `agent2` was trained on imperfect information. Then, using each agent, we simulated 100, 1,000, and 10,000 battles against an agent that selects completely random actions. Both teams used completely random teams of 3 Pokemon, chosen from a list of 20. Each simulation was repeated 20 times to get an estimate of variance of win rate.

In the 100 battle simulations, `agent1` won an average of 80.5 of the 100 battles, with a standard deviation of 4.31. `agent2` won an average of 79.35 of the 100 battles, with

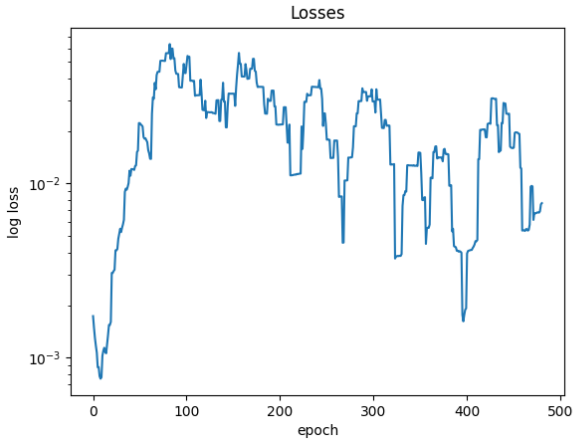

Figure 3: Moving average of log loss vs epoch count. Final epoch count was chosen to be around 400, which both minimized log loss and resulted in the corresponding agent performing well.

a standard deviation of 4.16. Using a two-sample test of proportions, testing whether `agent1` has a greater win rate than `agent2`, we get a test statistic value of $\chi^2 = 0.754$, with a corresponding p-value of 0.1926. Hence, in 100 battles, `agent1` is not significantly better than `agent2`. A table of these results is shown in table 1.

Agent	Win rate (sd)	χ^2 test
`agent1`	0.81 (0.043)	0.754 ($p = 0.1926$)
`agent2`	0.79 (0.042)	

Table 1: Win rates and standard deviation of the agents of interest, against agents that choose random actions, over 20 samples of 100 games. A two-sample proportion test is conducted to test if the agent trained on perfect information has a significantly higher win rate than the agent trained on imperfect information.

In the 1,000 battle simulations, `agent1` won an average of 806.3 of the 1,000 battles, with a standard deviation of 12.95. `agent2` won an average of 786 of the 1,000 battles, with a standard deviation of 12.15. Using a two-sample test of proportions, testing whether `agent1` has a greater win rate than `agent2`, we get a test statistic value of $\chi^2 = 25.144$, with a corresponding p-value of 2.66×10^{-7}. Hence, in 1,000 battles, `agent1` is significantly better than `agent2`. A table of these results is shown in table 2.

In the 10,000 battle simulations, `agent1` won an average of 8050.8 of the 10,000 battles, with a standard deviation of 45.1. `agent2` won an average of 7816 of the 10,000 battles, with a standard deviation of 26.7. Using a two-sample test of proportions, testing whether `agent1` has a greater win rate than `agent2`, we get a test statistic value of $\chi^2 = 83.923$, with a corresponding p-value of 2.2×10^{-16}. Hence, in 10,000 battles, `agent1` is significantly better than `agent2`. A table of these results is shown in table 3.

Agent	Win rate (sd)	χ^2 test
`agent1`	0.806 (0.013)	25.14 ($p = 2.7 \times 10^{-7}$)
`agent2`	0.786 (0.012)	

Table 2: Win rates and standard deviation of the agents of interest, against agents that choose random actions, over 20 samples of 10,00 games. A two-sample proportion test is conducted to test if the agent trained on perfect information has a significantly higher win rate than the agent trained on imperfect information.

Agent	Win rate (sd)	χ^2 test
`agent1`	0.81 (0.005)	83.9 ($p = 2.2 \times 10^{-16}$)
`agent2`	0.78 (0.003)	

Table 3: Win rates and standard deviation of the agents of interest, against agents that choose random actions, over 20 samples of 1,000 games. A two-sample proportion test is conducted to test if the agent trained on perfect information has a significantly higher win rate than the agent trained on imperfect information.

Lessons

The biggest lesson we learned is the importance of domain expertise in AI. In the implementation, we tried many different set-ups for the game, as well as rewards, and hyper-parameters. The biggest increases in performance were not due to changing the hyper-parameters, but due to changing the rewards in a way that more accurately reflects how a person would learn the game-play.

Discussion

In this work we used NFSP to developed two different agents that can effectively play GBL. One was trained on perfect information; complete data of the state space, and the other was trained on imperfect information; incomplete state information. Both agents were trained the same amount of time, using the same hyper-parameters. We pit these agents against an agent that only chooses random actions, and both agents could play reasonably well: both beating the random action agent around 80% of the time under all sample sizes.

However, naturally, the agent trained on perfect information had a significantly higher win rate than the agent trained on imperfect information on the larger sample sizes. This result is intuitive because, when trained for the exact same amount under the same hyper-parameters, the agent that learned on perfect information had more information in total to learn off of. This leads to the agent learning a better strategy.

As stated in the related work, no research was found comparing algorithms learning on perfect and imperfect information. However, it is widely acknowledged that training

on perfect information is always better. Many AIs initially start training on perfect information, before learning to play against themselves with imperfect information. The work done here supports the decision to train on perfect information first, as the strategies learned in that environment will be better.

Limitations

Due to the way we set up the game, we were unable to pit `agent1` and `agent2` against each other in the same game. This would have given us a more interpretable way to quantify the difference in their skill.

Due to computational limitations, we did not test a lot of hyper-parameters. Grid search is not the most effective method to obtain optimal hyper-parameters. There is a chance we did not include the best parameters in our search. Furthermore, different loss functions and different optimizers could have been used.

It should also be noted that statistical tests on simulated data is not always legitimate. As illustrated in our results, any small difference can become statistically significant over a large enough sample size.

Conclusion

Often times in the real world, we do not have access to perfect information. Self-driving cars, natural language processing, robotics, and fraud detection are all areas of AI that have to learn to make good decisions with the available information. In this paper, we were interested in comparing the performance of AI that trained on perfect information versus imperfect information.

To compare, we created an environment of GBL that is suited for applying NFSP. NFSP is a derivative of FSP, which uses neural networks to learn best and average responses, instead of reinforcement and supervised learning. These best and average responses are then used to choose future responses.

As one would expect, the performance of an NFSP model trained on perfect information was shown be be significantly better than an NFSP model trained on imperfect information. This shows that, when training an AI, one should always vie for as much information to train it as possible.

For future research, it would also be useful to see how fast each model (perfect vs imperfect information) converges. We have shown in this paper that the final performance of a model trained on perfect information is better than a model trained on imperfect information, but does it reach this performance faster or slower? Is the difference in speed statistically or practically significant?

It would also be interesting to do a similar comparison of NFSP in different environments. The difference between perfect and imperfect information in GBL is relatively small, but in games such as poker, blackjack, and bridge, having perfect information would greatly change your strategy. We would suspect the difference in performance

Finally, NFSP has thus far only been used in games, and can only be used in scenarios where a state takes inputs from various players. To extend the results of this paper to the field of AI in general, not only should NFSP be tested in other environments, but other models should be tested as well. It would be interesting to quantify how other algorithms' performances are affected by learning with perfect versus imperfect information.

References

Awan, S. E.; Bennamoun, M.; Sohel, F.; Sanfilippo, F.; and Dwivedi, G. 2022. A reinforcement learning-based approach for imputing missing data. *Neural Computing and Applications* 34(12):9701–9716.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning. *CoRR* abs/1912.06680.

Boney, R.; Ilin, A.; Kannala, J.; and Seppänen, J. 2020. Learning to play imperfect-information games by imitating an oracle planner. *CoRR* abs/2012.12186.

Brown, G. W. 1951. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*.

Campbell, M.; Hoane, A.; and hsiung Hsu, F. 2002. Deep blue. *Artificial Intelligence* 134(1):57–83.

Han, Y.; Zhou, Q.; and Duan, F. 2021. A game strategy model in the digital curling system based on nfsp. *Complex & Intelligent Systems*.

Heinrich, J., and Silver, M. L. D. 2015. Fictitious self-play in extensive-form games. *Proceedings of the 32nd International Conference on Machine Learning*.

Heinrich, J., and Silver, D. 2016. Deep reinforcement learning from self-play in imperfect-information games. *CoRR* abs/1603.01121.

Lizotte, D. J.; Gunter, L.; Laber, E.; and Murphy, S. A. Missing data and uncertainty in batch reinforcement learning.

Mai, T.; Nguyen, Q. P.; Low, K. H.; and Jaillet, P. 2019. Inverse reinforcement learning with missing data. *CoRR* abs/1911.06930.

Pricope, T.-V. 2021. Deep reinforcement learning from self-play in no-limit texas hold'em poker. *Studia Universitatis Babes-Bolyai Informatica* 66(2):51–68.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; and Hassabis, T. G. . D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J. P.; Jaderberg, M.; Vezhnevets, A. S.; Leblond, R.; Pohlen, T.; Dalibard, V.; Budden, D.; Sulsky, Y.; Molloy, J.; Paine, T. L.; Gulcehre, C.; Wang, Z.; Pfaff, T.; Wu, Y.; Ring, R.; Yogatama, D.; Wünsch, D.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T.; Kavukcuoglu, K.; Hassabis, D.; and Silver, C. A. . D. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575.

Zhang, L.; Wang, W.; Li, S.; and Pan, G. 2019. Monte carlo neural fictitious self-play: Achieve approximate nash equilibrium of imperfect-information games. *CoRR* abs/1903.09569.