Measuring the Solution Strength of Learning Agents in Adversarial Perfect Information Games

Zaheen Farraz Ahmad, Nathan Sturtevant, Michael Bowling

Department of Computing Science
University of Alberta, Amii
Edmonton, AB
{zfahmad, nathanst, mbowling}@ualberta.ca

Abstract

Self-play reinforcement learning has given rise to capable game-playing agents in a number of complex domains such as Go and Chess. These players were evaluated against other state-of-the-art agents and professional human players and have demonstrated competence surpassing these opponents. But does strong competition performance also mean the agents can (weakly or strongly) solve the game? Or even approximately solve the game? No existing work has considered this question. We propose aligning our evaluation of self-play agents with metrics of strong/weakly solving strategies to provide a measure of an agent's strength. Using small games, we establish methodology on measuring the strength of a self-play agent and its gap between a strongly-solving agent, one which plays optimally regardless of an opponent's decisions. We provide metrics that use ground-truth data from small, solved games to quantify the strength of an agent and its ability to generalize to a domain. We then perform an analysis of a self-play agent using scaled-down versions of Chinese checkers.

Introduction

Adversarial games have become widely used environments for developing and testing the performance of learning agents. They possess many of the same properties of real-world decision-making problems in which we would want deploy said agents. However, unlike highly complex real-world environments, games can be more readily modelled and have very clear notions of success and failure. As such, games make an excellent training ground for designing and evaluating AI intended to be scaled to more realistic decision-making scenarios.

Recent advances in AI research have birthed a number of agents that imitate intelligent behavior in games with large state spaces and complex branching factors. Through a combination of different techniques and approaches, highly performant agents were developed to play games such as Checkers (Schaeffer et al. 1992), Chess (Campbell, Hoane Jr, and Hsu 2002; Silver et al. 2018), Poker (Bowling et al. 2017; Moravčík et al. 2017), Go (Silver et al. 2016, 2017, 2018) and Starcraft (Vinyals et al. 2019). These agents were all evaluated against humans and demonstrated strategic capabilities that surpassed even top-level professional human players.

A commonly used method to evaluate the performance of agents in two-player games is to have the agents play against other agents with theoretical guarantees of performance, or humans in a number of one-on-one games and then measuring the proportions of wins and losses. The agent which wins the most games against the others is said to be the more capable agent. However, this metric only provides a loose ranking among the agents — it does not provide a quantitative measure of the actual strength of the agents or how well they generalize to their respective domains.

Alternatively, in the realm of two-player, perfect-information games, one can evaluate the strength of an agent with regard to certain solution concepts. These solution concepts are defined with respect to the game theoretical value obtainable by different strategies (Allis et al. 1994). For instance, a "strong" agent would be able to obtain the value of a position of a game from any legal position while a "weaker" agent can do so only in a smaller subset of states. These measures of an agent's skill are more principled and informative than solely their ranking with regards to other agents. However, finding such measures are computationally expensive and usually require exhaustively searching over all sequences of play. While they have been used in a number of games to different degrees of success (Allis et al. 1994; Schaeffer et al. 2007), the computational requirements prohibit them from being employed in larger games such as Chess or Go.

We focus our investigation solely on two-player, zero-sum, perfect-information games. In this paper, we propose aligning our understanding of the strength of a player using metrics of strongly/weakly solved and error rates. We build an AlphaZero agent that learns to play Chinese checkers and use it to learn to play small board sizes. We then use an existing Chinese checkers solver to evaluate the strength of the AlphaZero player using ground truth data.

Background

Adversarial Games

An *adversarial game* is a sequential decision-making setting in which two players alternate taking actions at differ-

ent game states until a terminating state is reached at which point each of the players observe some utility. We model a zero-sum, deterministic, perfect information game as a set of **states**, $\mathcal{S}$, where at each state a player $i \in \{0, 1\}$, selects an **action** from the set of all actions, $\mathcal{A}$. A **policy** is a mapping $\pi : \mathcal{S} \to \Delta\mathcal{A}$ from a state to a distribution over actions and we say $\pi(a|s)$ is the probability of taking an action $a \in \mathcal{A}$ at $s \in \mathcal{S}$. There is a deterministic **transition function** $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ that returns a new state $s' \in \mathcal{S}$ when action a is taken at s.

We define $\mathcal{Z} \subset \mathcal{S}$ to be the set of all the **terminal states**. At a terminal state, no players can take an action and the game ends. A **utility function** $u : \mathcal{S} \to \mathbb{R}$ associates a real-valued utility (or reward) to each state so that the utility observed by each player i at a state s is $u_i(s), \forall_i$. The utility for all non-terminal states is 0 and the utility of a terminal state can be 1, 0 or -1. These are known as the *game-theoretic values* of the terminal states. A utility of 1 signifies a win, -1 a loss and 0 represents a draw (if there are any.) In a zero-sum game the utilities of both players sum to 0 ($u_1(s) = -u_2(s)$) and so if one player wins the other will lose. Typically in most combinatorial games, the player who takes an action that transitions to a terminal state is the winner.

Each state is also associated with a **state-value**, $V(s)$, which is the *expected utility* observed by a player reaching state s and then assuming both players follow their respective policies until a terminal state is reached where $V(z) = u(z), \forall_{z \in \mathcal{Z}}$. An **action-value** or **Q-value** is the value associated with a *state-action pair*. We denote the Q-value of a state-action pair as $Q(s, a)$ and it is defined as the expected utility of taking action a at s and then having both players follow their policies until termination. As we are only examining games with deterministic transitions, we can define $Q(s, a) = V(T(s, a))$ and $V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot Q(s, a)$.

The goal of an agent in a game is to maximize the utility it achieves when playing the game (that is, it strives to win the game.) An agent does so by finding policies that selects actions with higher Q-values at any state of the game ensuring that it has a higher chance of reaching terminal states that provide positive utilities.

Solved Games

A player that behaves optimally in a game is said to exhibit *perfect play* — the player will always behave in a way such that it guarantees an optimal outcome for them, regardless of the opponent's actions. The optimal outcome can be a win, draw or loss. Knowledge of perfect play is conditioned on a game being **solved**. That is, the outcome of the game can be determined when both players are acting perfectly. Allis et al. (Allis et al. 1994) present three different definitions for "solved" in the setting of zero-sum perfect information games,

- ultra-weakly solved
- weakly solved
- strongly solved.

An **ultra-weakly solved** game is one where the game-theoretic value of the initial position is known. Given perfect play, the outcome of a game is known to be win, loss or draw but there need not be a strategy produced that guarantees this outcome. For example, for the game of Hex, it has been proved that the first player will win on all square boards given both players play optimally. However, there is no constructive strategy that is guaranteed to reach this outcome[1].

A **weakly solved** game is one where there is a strategy that guarantees the game-theoretic value of the game when both players play optimally. Checkers is a weakly-solved game (Schaeffer et al. 2007) and is shown to be a draw for both players under perfect play — there is an explicit strategy that is shown to never lose and will always at least draw with any opponent. However, a strategy that weakly solves the game does not necessitate that the player following said strategy will capitalize on situations where the opponent plays sub-optimally. For instance, if the opponent makes a mistake and moves to a losing position, a strategy that guarantees a draw will not necessarily win from there.

In a **strongly solved** game, the game theoretic value of every legal position is known and there is a strategy that guarantees that a player will observe that outcome from that position. Examples of solved games are Tic-tac-toe, Connect-Four (Allis 1988) and smaller variants of Chinese checkers (Sturtevant 2019). A player must capitalize on any mistake an opponent may make and play optimally from every legal position.

Strongly solving a game requires exhaustively enumerating all sequences of play from the initial position of the game and, through backwards induction, eliminating from the strategy paths that lead to sub-optimal outcomes. Predictably, the intense computational requirements currently renders it intractable to solve games such as Chess or Go due to the sheer size of their state space (approximately 10^{50} states for Chess and 10^{170} for Go.) However, knowledge that a strategy is weakly or strongly solving is an irrefutable measure of the competency of an agent's behavior and so it is desirable to solve games.

Related Work

Planning methods have been instrumental in finding approximately good strategies in sequential domains where exhaustive search is infeasible. A sample-based planner repeatedly samples actions to evaluate at a decision point to approximate the expected value of their outcomes. One such planning method is Monte-Carlo tree search (MCTS), a simulation-based scheme which incrementally builds a *search tree* of sequential states and actions and records statistics of estimated outcomes. At iteration, MCTS selects an action to be sampled using a *selection policy* and follows a separate *rollout policy* (e.g. a random policy) at each subsequent state until a terminal state is reached. The outcome is observed and the estimates of the outcome of the action at the state are updated. A commonly used variant of MCTS called UCT (Kocsis and Szepesvári 2006) attempts to balance the sampling budget between *exploiting* promising ac-

[1]The proof for the game-theoretic of Hex was derived using a *strategy stealing* argument and knowledge that there are no draws in Hex.

tions to improve the estimates of their outcomes and *exploring* new actions. For every action at a state, UCT keeps track of its estimated Q-value and its selection count, and samples the action with the highest one-sided confidence value according to

$$\arg\max_a \left(\bar{Q}(s,a) + C \cdot \sqrt{\frac{\log N}{n_a}} \right), \tag{1}$$

where $\bar{Q}(s,a)$ is the current sampled expected action-value, N is the total number of samples taken and n_a is the number of times action a was sampled. C is an exploration coefficient which influences the rate at which new actions are sampled. While UCT has been shown to work well, it requires all actions to be sampled at least once, limiting its effectiveness in large domains.

Variants of MCTS use learned policies to reduce the search complexity (Silver et al. 2017, 2018). These agents use a *function approximator* that, when given a game state, produces a policy and a value estimate. The value estimate is an approximation of $V(s)$ and the policy is used to guide search using PUCT (Rosin 2011). The action a_t to be sampled at s is selected according to:

$$a_t = \arg\max_a (Q(s,a)) + U(s,a)).$$

The Q-values used for PUCT are calculated as $Q(s,a) = \frac{1}{n_a} \sum_{s'|s,a \to s'} V(s')$ where $V(s')$ is acquired from the function approximator, n_a is the visit count for action a and $s'|s,a \to s'$ represents that s' is reached by playing a at s. $U(s,a) = C(s) \cdot P(a) \cdot \sqrt{N}/(1+n_a)$, where $C(s)$ is the exploration rate that grows with time and $P(a)$ is the prior placed on action a according to the policy. The function approximator is trained and refined using *self-play* and *reinforcement learning*. These agents continuously play games against themselves and their approximators update their parameters so as to move their policy and value estimates closer to the sampling distributions of PUCT and the outcomes respectively. In recent work (Silver et al. 2016, 2017, 2018), these agents were evaluate against other MCTS agents and humans. Unfortunately, these evaluations only rank the skills of the different agents but do not provide a measure of their strength.

Measures of the Strength of An Agent

While competitive comparisons between game-playing agents provide a ranking over the skill of the agents, they do not provide measures on the strength of their strategies. Does an agent that wins against all other agents have good strategies? Or are the strategies of the other agents just worse?

We propose that being able to quantify the strength of the agent in terms of weakly/strongly solving games would present us metrics on the quality of the strategies these agents. Specifically, we may be able to measure of the quality of the strategies learned by agents through self-play and gain new insight into their behavior. For instance, which states do these agents learn to play well? Which do they have difficulty on? Does self-play result in exploitability to which other agents are not susceptible? Moreover, these metrics would be independent of any other players and wholly rely on the agent's capacity of reasoning in the game. To that end, we now propose our *measures of strength* of an agent that reflect the three definitions for solving a game:

Measure of Ultra-Weak This is the proportion of times an agent achieves the game-theoretic value of a game when playing against itself. With a deterministic agent, this reduces down to whether the line of play recovers the game-theoretic value.

Measure of Weak When playing against all possible responses, what percentage of the game outcomes observed following the agent's strategy matches the game theoretic value of the game.

Measure of Strong When at any possible state of a game, how accurate is the agent at predicting the state value and selecting the optimal action from that state. There is a possibility that a trained agent never learns to correctly predict the state values, but may always learn to take the best action.

In this work, we intend to measure the strength of strategies learned through self-play using these metrics. We do not expect a self-play agent to learn strong strategies for a game as these agents learn from only a fraction of all possible states in large games. But a measure of what proportion of all possible states it can correctly evaluate would provide some notion of how well the agent learns to generalize. However, the nature of self-play training indicates the agent may learn to at least weakly solve the game on which it is trained. Using our defined metric, we propose that we can verify if an agent does indeed learn to weakly solve a game and, if not, how close it can come to doing so.

Experimental Evaluation

Our evaluation is done in the game of Chinese checkers. We train an agent on a smaller sized version of Chinese checkers to evaluate using our metrics of strength. We now give a description of Chinese checkers and the training regime.

Chinese Checkers

Chinese checkers is played on a six-cornered star-shaped board (illustrated in Figure 1) with 10 checkers or pieces per player. While the game can be played between 2, 3, 4 or 6 players, we only focus on the two-player version of the game. In two-player Chinese checkers, each player's pieces start in opposite corners (typically called the top and bottom corners) of the board. The players then alternate moving pieces one at a time across the board. To win, a player must move all their pieces into the opponent's corner.

Usually a player cannot move their pieces into any of the other corners and so play is limited to the central diamond region which we represent as an $(n \times n)$-grid. The game can then be represented with a (9×9)-grid, the region within the dashed lines in Figure 1. There exist smaller versions of the game played with fewer pieces on smaller grids. Figure 2a shows the starting "grid configuration" of a (5×5) game with 6 pieces. For the rest of the paper, the player starting

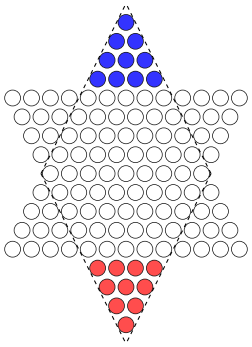
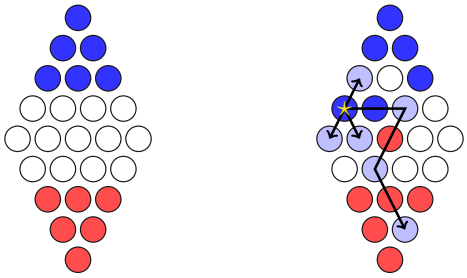


Figure 1: Starting configuration of a full $9/10$ game of Chinese checkers in the star-shaped board.

at the top will be the first player and, for shorthand, we will write n/k to refer to a game on a $(n \times n)$ board with k pieces.



(a) The starting configuration of pieces in a $5/6$ game.

(b) All possible moves for the checker demarcated with a star.

Figure 2: Grid of a 5/6 game.

At each turn, a player may move only one piece either to an empty adjacent spot or *hop* over an adjacent checker to an empty spot. When hopping, a player may continue hopping over adjacent checkers as long there is an empty spot within which to stop. Figure 2b shows a legal configuration of checkers in a $5/6$ game. The arrows show all possible movement paths for the blue checker marked with a star and the lightly shaded spots are spaces to where it can legally move. As can be seen there are a number spaces that can be reached by chains hops over both the player's own and the opponent's pieces. It should be noted that unlike many other games, checkers are not removed from the board after hops.

To win, a player must move all of their pieces into the opponent's corner. However, there are states that are reachable where the player cannot place all of his pieces in the opponent's corner e.g. the opponent neglects to move a piece out of its starting position blocking the player from winning. To handle these situations, we redefine the win condition to include *all states where a player's goal corner is filled with pieces and at least one of those pieces belong to the player* (Sturtevant 2019).

Model Architecture

We use a neural network as our function approximator for a game playing agent. The neural network architecture used

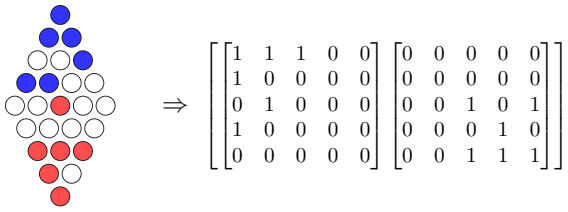


Figure 3: Input to the neural network of a $5/6$ game state.

for the learning agent takes as input states of the game and outputs a prediction for the value of the state and a policy over actions available at the state. Each input state is represented by a stack of two binary matrices, each with dimensions matching the $(n \times n)$ dimensions of the board. Figure 3 depicts a game state and its input representation to the neural network. The first matrix corresponds the locations of the current player's pieces (top,) where a 1 denotes that a piece exists in a space and a 0 is an empty space. The second matrix corresponds to the location of the opponent (bottom.) For consistency, the current player is always represented as starting from the top-left corner of the matrix moving toward the bottom-right corner.

The network used is a residual network containing 3 residual blocks. Each residual block consists of a 3 convolutional layers: a layer of 64 (1×1) filters, a layer with 64 (3×3) filters and a layer of 256 (1×1) filters. The inputs are passed through 1 convolutional layer of 256 (3×3) filters before being passed to the residual blocks. The outputs of the residual blocks are passed to a value head and a policy head. The value head is feedforward layer of 64 hidden units and an output layer of 1 unit using tanh activation. The policy head consists of one convolutional layer with 256 (3×3) filters and and one with 16 (3×3) filters. All hidden units used Rectified Linear Unit (ReLU) activation and all convolutional layers preserved input dimensions.

The value output by the network is a scalar prediction, $v(s) \in [-1, 1]$, of the expected outcome for the player acting from that state. -1 is a loss for the player, 1 is a win and 0 is draw. The policy output is a stack of n number of $(n \times n)$ matrices representing move probabilities. Each matrix in the stack represents a specific location of the board and the values of that matrix are the probabilities of moving a piece from that location to all other locations. If a location does not contain a piece belonging to the acting player, all of the values of the matrix for that location are 0. The probabilities of all illegal moves are zeroed out.

Self-Play Training

We trained an AlphaZero agent on 4/3-sized games of Chinese checkers through self-play reinforcement learning (Silver et al. 2017). At each turn of a game, the agent used PUCT with a budget of 256 samples to search for the best action. The neural network was used to provide prior probabilities over which actions are sampled during search and the value predictions are used to evaluate the outcomes of the leaf nodes when expanding the search tree. Dirichlet noise was added to the priors over actions at the root scaled inversely

Outcome	# of States
P1 Win	154,726
P1 Loss	154,726
Draw	0
Illegal	10,868

Table 1: Outcomes of states in 4/3 Chinese checkers.

proportional to the number of actions at the state. Each game was given a turn limit of 50 turns; a game that was played beyond 50 turns was prematurely ended and the outcome was declared a draw.

At each training iteration, the agent played 800 games before using the game data to update the network parameters, θ. The network parameters were updated to reduce the mean square error (MSE) between the predicted outcomes ($\hat{v}$) and the observed outcomes (v) of the game states, and to reduce the cross-entropy error between the output policy ($\boldsymbol{\rho}$) and search sampling distribution ($\boldsymbol{\pi}$). The objective is defined as

$$\arg\min_{\theta} l(\theta) = (\hat{v} - v)^2 - \boldsymbol{\pi} \log \boldsymbol{\rho} + \lambda ||\theta||^2$$

where λ is the coefficient for L2-regularization. We used a value of $\lambda = 10^{-5}$ set after performing a parameter sweep over λ values. We updated the parameters by gradient descent using Adam optimization with a learning rate $\alpha = 10^{-4}$ that dropped to $\alpha = 10^{-5}$ after 20 training iterations. We trained our agent for a total of 50 iterations.

Solved Chinese Checkers Data

A solver was used to generate ground truth values of the outcomes of the game and all states in 4/3 Chinese checkers. The data was produced by exhaustively enumerating all sequences of play to identify the outcomes of optimal play by each players. It was found that under optimal play, the game theoretic value of the 4/3 Chinese checkers is 1 for the player that moves first i.e. under perfect play, the first player will always win. There are a total of 320,320 different states in the game, 10,868 of which are illegal (unreachable through legal play.) Table 1 summarizes the outcomes of all states in the game.

Results

We now use our three measures of strength to quantify the strength of the agent trained to play 4/3 Chinese checkers. These measures were computed using ground truth data generated by the solver.

Measuring Ultra-Weakly Solving

Figures 4 through 6 plot statistics of self-play during training. Figure 4 plots change in the win rates of the two players during training time. The blue solid line represents the win rate of the first player, the red dotted line represents that of the second player and the green dashed line represents the draw rate. Even though under perfect play there are no

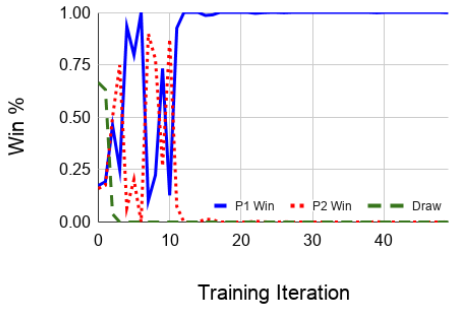


Figure 4: Win rates of the players during training.

draws in the 4/3 game, untrained agents play games that exceed the turn limit resulting a higher number of draws initially. By the 12th training iteration, the first player win rate reaches 100%. This indicates that in 4/3 Chinese checkers, the self-play agent can determine the outcome of the game under rational play from the initial state and thus can be said to ultra-weakly solve the game.

Figure 5 plots the accuracy of the agent's predictions of state values during training. By the 18th training iteration, the agent learns to perfectly predict the values of states it encounters playing under its learned strategy. The initial fluctuations in its predictive ability could be because it learns a strategy that wins as first player, then it learns to beat that strategy as the second player, and so on.

Figure 6 plots the average length of the games played at each training iteration. As the agent learns to play more efficiently, the average length of the games shorten since the agent learns actions leading to quicker victory. However, after 20 training iterations, the average length of the games drop to approximately 5 turns (The minimum number of moves to a win is 3 in the 4/3 game.) We believe that when the agent learns the first player is guaranteed to win, as the second player, it begins to behave indifferently to its actions. Since there is no action that will lead to a victory for the second player, all actions seem the same to the agent and it is now more likely to choose actions that drastically shorten the game. This is a consequence of the rule introduced to prevent players from blocking their starting area in order to keep the other player from winning. A player wins when their goal area is filled and they have at least one piece in the goal. On the 4/3 game, this can happen quickly whereas in larger games it does not. This indicates a potential issue with self-play in that these agents are not trying to explicitly play as strong of a game as possible.

Measuring Weakly and Strongly Solving

In 4/3 Chinese checkers, to weakly solve the game an agent's actions must always lead to a win as the first player against any actions played by the second. To verify whether the agent learned to weakly solve the game, we enumerated all possible trajectories of the game under the agent's strategy. For every action the agent took at any state, we enumerated all of the possible responses for the opponent. To select an action at a state, the agent used PUCT using its value pre-

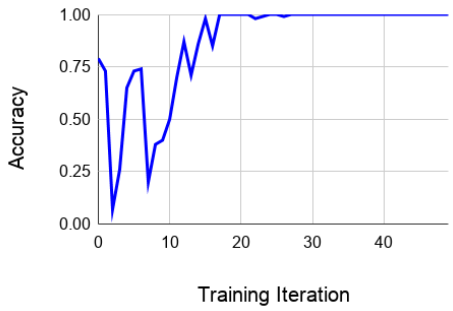


Figure 5: State value prediction accuracy during training.

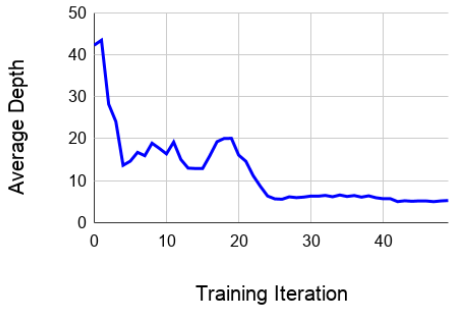


Figure 6: Average length of games during training.

dictions and policy. PUCT was given a budget of 1024 iterations for search after which the most sampled action was selected. We built the solution tree using the agent after 10, 20, 30 and 40 iterations and with an untrained agent for a baseline. After building the solution trees, we looked at the proportion of all outcomes that were a win for the first player. We report these results in Table 2 along with the number of unique states visited in the solution tree. It should be emphasized that examining all trajectories of the game is more informative than just examining all states in the solution tree. For instance, it may be possible for the agent to predict optimal actions for all states except at the root. In this case, a state-wise evaluation would result in a high performance measure but looking at all trajectories would give the agent a 0% win rate.

While the agent does not weakly solve the 4/3 game, the results indicate that the strategy learned by the agent come close. It is important to note that while the untrained agent's actions lead to a win for the first player 88.6% of the time, self-play resulted in a significant improvement to its strategy. The proportion of wins for the first player, flattens out at approximately 98%.

To measure the agent's ability to strongly solve 4/3 Chinese checkers, we use the agent to predict the outcomes and optimal actions over *all* states of the game. Note, that due to symmetry reduction, the agent need only be evaluated on half of all possible states. Figures 7 through 9 plot the predictive capacity of the agent on all states over training, illustrated as solid lines on the plot. We also plot the predictive

Training Iterations	Win %	No. of States
-	0.886	147,813
10	0.955	45,390
20	**0.980**	22,633
30	0.975	35,272
40	0.976	25,236

Table 2: Outcomes reached by enumerating the game tree using the strategy of the agent learned after different training iterations.

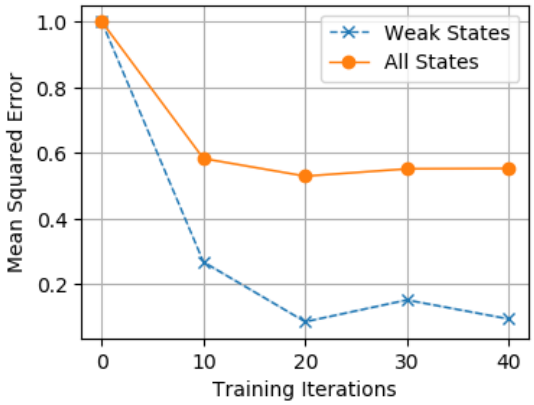


Figure 7: Mean squared error of value of predictions.

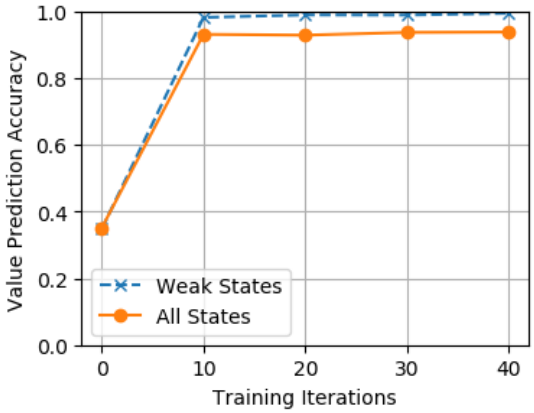


Figure 8: Accuracy of state value predictions.

performance of the agent on the subset of unique states it observes when enumerating the weak solution trees (we call these *weak states*.) These are all the states that an agent may encounter under its strategy and may also include states not seen during training. These are illustrated as dashed lines on the plots.

Figure 7 plots the mean-squared error (MSE) of the state value predictions against the true value of the state using the solved game data. The values are plotted with their 95% confidence intervals (not visible at this scale.) The MSE of the value predictions decreases with training. The MSE is sig-

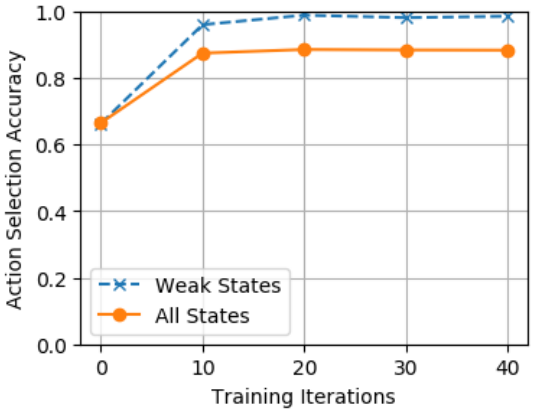


Figure 9: Accuracy of selecting actions.

nificantly higher when the agent is used to predict the values of all states than when it is predicting the subset of weak states. Figure 8 plots the accuracy of state value predictions. To retrieve a value prediction, we take the value output by the agent's network and classify it as a value in $\{1, 0, -1\}$ using thresholds — $\hat{v} > 0.25$ is classified as 1, $\hat{v} < -0.25$ is -1 and 0 otherwise. For the weak states, the agent comes close to perfectly predicting all state values. The agent is weaker at predicting state values of states it will not come across during play but still performs with reasonable accuracy. The MSE indicates that the agent is more confident in its predictions of weak state values.

In Figure 9, we plot the accuracy of selecting an optimal action over all states and all weak states. That is, we verify if the action selected by the agent at each state using PUCT is optimal. An action is optimal if it leads to a state with a true value higher than states reached by the other possible actions. The true value is found using the solved game data. Note that there may be more than one optimal action. As before, the agent's performance increases with training time. Also similarly, it is more capable of finding the best action in the weak states (again, near perfectly) than over all states.

Discussion and Future Work

Our results show that our proposed metrics provide measures of strength that closely reflect what we expect from a learning agent in the small game. However, we cannot yet say these measures scale well to larger games. Our measures show that the self-play agent ultra-weakly solves the 4/3 game but does it converge to the game-theoretic values in other strongly solved board sizes of 4/6 or 5/3? In addition, our measure shows that the agent comes very close to weakly solving the game but ultimately does not converge to the game-theoretic value. Is this a correct measurement? Does this value move towards the game-theoretic value in larger games? Is the agent only losing in trajectories where its opponent plays sub-optimally? The self-play agent is trained to expect rational play and so perhaps limiting weak evaluation to optimal trajectories only is a more correct assessment. We can also ask how our metrics evaluate other agents e.g., a pure UCT-player..

Moreover, we intend on looking at additional metrics using solved data including mistake rates during play (how often, which states, how early in play, etc.) and the strength of the strategies with regards to the difficulty of a state. We also intend on performing ablation tests to measure the strength of the value and policy heads of the agent and the improvements gained by using search. Furthermore, while our work is limited to games where ground truth is available, we intend to examine whether these measures can be approximated in games currently too large to solve.

Conclusions

In this work, we propose measuring the *strength* of a learning agent using metrics of strongly and weakly solved. We introduce methods to measure the strength of an agent using solved game data and demonstrate evaluation using a scaled-down version of Chinese checkers. Our preliminary results show that in this smaller game, a learning agent approaches strongly and weakly solving the game. Additionally our evaluations indicate that these measures may provide us with unique insights into the learned behavior of these agents and directions in which they can be improved.

References

Allis, L. V. 1988. *A Knowledge-Based Approach of Connect-Four*. Master's thesis, Vrije Universiteit.

Allis, L. V.; et al. 1994. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2017. Heads-up limit hold'em poker is solved. *Communications of the ACM* 60(11): 81–88.

Campbell, M.; Hoane Jr, A. J.; and Hsu, F.-h. 2002. Deep blue. *Artificial intelligence* 134(1-2): 57–83.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356(6337): 508–513.

Rosin, C. D. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence* 61(3): 203–230.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *Science* 317(5844): 1518–1522.

Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artificial Intelligence* 53(2-3): 273–289.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676): 354–359.

Sturtevant, N. R. 2019. On Strongly Solving Chinese Checkers. In *Advances in Computer Games (ACG)*. URL http://www.cs.ualberta.ca/~nathanst/papers/sturtevant2019chinesecheckers.pdf.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782): 350–354.