Date of acceptance Grade

Instructor

Policy Improvement in Cribbage

Sean R. Lang

Helsinki May 16, 2018

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Sean R. Lang

Työn nimi — Arbetets titel — Title

Policy Improvement in Cribbage

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	May 16, 2018	53 pages + 0 appendices

Tiivistelmä — Referat — Abstract

Cribbage is a card game which involves multiple methods of scoring and in which each different method of scoring receives varying emphasis over the course of a typical game. Reinforcement learning is a machine learning strategy in which an agent learns to accomplish a task via direct experience by collecting rewards when successfully completing a subtask. In this thesis, reinforcement learning is applied to the game of cribbage to improve an agent's policy in order to learn which particular strategy is most applicable at a certain position on the cribbage board. From a cribbage player's perspective, a reasonable policy is learned by the agent over the course of a million games, but the resulting agent does not compete well on the level of individual games.

ACM Computing Classification System (CCS):
I.2 [ARTIFICIAL INTELLIGENCE],
I.2.1 [Applications and Expert Systems],
I.2.6 [Learning]

Avainsanat — Nyckelord — Keywords
machine learning, reinforcement learning, cribbage

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Acknowledgments

I would first like to thank my parents, Kevin and Debra Lang, for supporting me throughout the years and especially in the decision to pursue my studies in the distant land of Finland. I would also like to thank Drs. Teemu Roos and Jukka Kohonen for their mentorship throughout the process of developing and writing this thesis. Thanks also to Chelsea Dunham for her help in editing and proofreading this paper, without which, quality would surely have suffered. Additionally, a courteous thank you to Matt Jennings at `dailycribbagehand.org` for providing a data dump which could be used to compare my model against decisions humans made in a variety of situations. Finally, my gratitude is extended to the University of Helsinki for the opportunity to pursue my studies here.

Contents

1 Introduction

This thesis will focus on the process of attempting to learn how to play a game of cribbage following a set of predefined strategies via reinforcement learning. The remainder of this section will introduce the main features of the game and the topic of reinforcement learning to readers unfamiliar to either. Section 2 will provide an overview of related work in reinforcement learning with relevance to playing games as well as previous research done on the game of cribbage. The process used to determine how the agents would learn to play the game and what the predefined strategies were will be described in Section 3. The results of the experiments and reasons for any deviations from expected as well as the reasons for their occurrence are explained in Section 4. Finally, a discussion of the potential applications for this thesis's findings and a summary of improvable areas are presented in Section 5.

1.1 Cribbage

Cribbage is a multi-phase card game, typically played between two opposing players. While variants exist for three or more players, this paper will focus on the two-player variant. The game presents an interesting research area because of its unique scoring methodology: each hand is counted in two slightly different ways within each round and the first player to reach a score of 121 points or more is declared the winner. Players will usually keep track of their points by moving pegs along on a characteristic board in a process called *pegging*. Because of its atypical win condition, different strategies hold differing levels of importance throughout the game.

Rules of the Game

In order to be able to understand the crucial nature of the temporally dependent strategies, the rules and flow of a game of cribbage must be fully understood. While a complete set of tournament rules can be found at [ACCa] or [ACCb], what follows is an overview complete enough such that a novice player, with the assistance of the scoring rules found in Table 1, could play a complete game.

The zeroth step, taken once per game, is to determine which player will be the dealer for the first round and who will be the pone. In order to determine these roles, each player cuts the deck once to get a single card: the player with the lower-valued card[1] is the dealer; the other player is called the pone. In the case of a tie, this step is repeated until two unique cards are cut from the deck. From there, the usual round structure begins and proceeds in the following steps:

1. Each player is dealt 6 cards.
2. Each player selects 4 cards to keep for their own hand and 2 cards to discard, or toss, into what is called the crib.

[1] Ace $<$ 2 $<$ 3 $<$ 4 $<$ 5 $<$ 6 $<$ 7 $<$ 8 $<$ 10 $<$ Jack $<$ Queen $<$ King

Cards or Combinations	**during play**	**in hand or crib**
Jack turned by dealer as cut card	2	—
Jack in hand or crib of same suit as cut card	—	1
two of a kind (pair)	2	2
three of a kind (3 pairs)	6	6
four of a kind (6 pairs)	12	12
straights of three or more cards (per card)	1	1
15-count (sum of any combination of cards)	—	2
four-card flush (only in the hand)	—	4
five-card flush	—	5
reaching a 15-count exactly	2	—
reaching a 31-count exactly	2	—
final card played (without reaching 31-count)	1	—

Table 1: Scoring Rules for a game of cribbage [ACCb]

3. The deck is cut at a random location by the pone and the top card from this cut is selected by the dealer and placed face-up on top of the deck. If this cut card is a Jack of any suit, the dealer is awarded 2 points and pegs the points accordingly.

4. Starting with the pone, each player alternates playing a single card by placing them face-up on the table, keeping track of the total value[2]of all card played so far, until all cards have been played or neither player can play a card without exceeding a collective value of 31. If any of the situations or combination of cards mentioned in Table 1 is seen in the immediately preceding cards, the amount of points earned is immediately pegged on the board for the player who played the last card. In cribbage terms, this is called *the play* or perhaps confusingly, in certain circles due to the rapid nature of the action, *pegging*. These terms will be used interchangeably throughout this paper, with a preference for *pegging*.

5. After all cards have been played, the pone then counts his or her hand using the randomly cut card as a fifth card in hand before pegging these points on the board.

6. The dealer then proceeds to count his or her hand and peg the points in the same fashion, also considering the randomly cut card to be the fifth card in the hand.

7. The dealer then does the same for the crib.

8. The dealer and the pone swap roles and repeat from step 1.

[2]The value of all numbered cards is that number, aces have a value of 1, and all face cards have a value of 10.

If, at any point, a player achieves a score greater than or equal to 121, that player is immediately declared the winner and the game is over.

The win condition for this game can occur at any moment of the game, even beyond either player's control (note Step 3). Because of this, it is crucial to play according to different strategies during different times of the game, where time can be defined by what score the player has, combined with what score their opponent has and which player is the dealer for that round. Typically, during early and middle-game play, the pone will attempt to maximize their own hand, while avoiding giving too much opportunity for the dealer to score points from the crib. However, in later play, this may no longer be a concern. For example, should the player be the pone and their score is 116 and the dealer has 117 points, due to the counting precedence, the player needs not concern themselves with what points the dealer will obtain through the crib, if their own hand has at least 5 points guaranteed, since the pone will count first and win. This only works, however, if the pone does not allow the dealer to score 4 points from *the play*. As can be seen, the player must balance multiple competing factors with varying emphasis over the course of the game.

1.2 Reinforcement Learning

Reinforcement learning is the machine learning equivalent of learning from one's failures rather than being coached to the correct answer.[3] In classical machine learning methodologies, the agent discovers an optimum model for a problem by approximation methods centered around minimizing a set loss function over a given set of data while assuming a known model for the solution. In reinforcement learning, however, the agent finds the optimal solution to the problem by repeatedly taking an action in an environment and gaining a reward or punishment for each action taken. It is the same principle used in teaching a pet or animal to do a trick: offer a full or partial reward for successful completion of the trick or for progress in the correct direction. As a comparison to teaching a human how to add, the strategy used in classical machine learning would be what is used in classrooms today: teach the method of adding digits and handling carry-over, giving some guidance and sample problems to ensure the technique is solidly replicable and generalizable to previously unseen problems. Meanwhile, teaching a human how to add by reinforcement learning would mean merely quizzing the subject by asking him to answer an addition problem while giving a vague hint as to how right or wrong they were. After enough rounds of this, the student will eventually figure out his own method for adding two numbers with the same accuracy as an established method.

While this may seem like a silly example where classical methods would clearly be the superior method, where reinforcement learning comes into its own is in situations in which no known answer exists for a problem. Take for instance the problem of

[3] Unless otherwise specified, all information presented in this section is referenced from Sutton and Barto [SB17].

learning chess. In humans, basic strategies for how to handle certain situations can be taught, but these are all from one's own or others' prior experience, and while they may bolster knowledge on heretofore unknown situations, they cannot possibly cover all possible chessboard layouts. Arguably, the best way to learn, in humans and computers, is by doing. After a game has been played, the player can see what worked and failed during the game to cause the win or loss and extrapolate what to do if a similar situation occurs in order to improve play. Classical teaching methodologies would not be highly applicable in this situation because there is no single optimal strategy or even set of strategies in chess that can be taught.

Rewards and the Environment

The three most important, constantly interacting components in a reinforcement learning scenario are the environment, the agent, and the rewards. The agent must learn to navigate the environment in order to maximize its rewards, much like a mouse navigating a maze to retrieve the cheese at the end.

The Environment In reinforcement learning scenarios, the agent interacts with and navigates what is known as the environment. The environment is a set of states in which an agent can find itself. What exactly constitutes the environment is problem-specific and the line between agent and environment is not often clearly defined. An individual state can be any situation in which the agent finds itself and can be in either discrete or continuous space. For instance, in chess a discrete state would be a specific board arrangement. In golf, an example of a state in continuous space would be the location of the ball along the course of play and the current wind velocity. An action is an interaction the agent can make with the environment to alter its current state. In the example of chess, an action is discrete and would be to move a piece X to position Y, e.g. moving the bishop to G4 (*Bg4*). In golf, the action is, again, continuous and may be which club to use in which direction and with how much power.

Goals and Rewards Merely being able to move around in an environment does not satisfy the requirement for learning unless a given task is being completed. This desired task can be called the goal of the agent. For games scenarios, this is simply the notion of winning.

Rewards can be thought of as a way of enticing the agent to accomplish the goal. A reward is a positive feedback event that encourages and affirms progress towards the goal. As with a dog learning to jump through a hoop, a treat is given after the dog has successfully jumped through the hoop, or perhaps a partial treat for first walking through a stationary hoop or other similar subtask.

Expressed mathematically, a reward R_t is the reward at a given time t. A goal G_t

is the expected return over time:

$$G_t = \sum_{k=t+1}^{T} R_k$$

where T is the final time step. This goal formula can also incorporate a discounting factor γ to encourage actions conducive to reaching the terminal state in a speedy fashion:

$$\begin{aligned} G_t &= \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Policies A policy is a mapping of actions to states. A policy π describes a set probabilities $P(a|s)$ where $a \in \mathcal{A}$ is an action in the set of actions and $s \in \mathcal{S}$ is a state somewhere in the environment. An optimal policy π_*, of which there may be several, is any policy which achieves a maximum expected reward over the course of taking its actions.

Learning an Optimal Policy

Although applicable to both discrete and continuous state representations, it is useful for the sake of illustration to limit the scope of discussion to discrete representations. Heretofore, unless otherwise stated, all discussion will assume a discrete representation.

Metrics A state can have a *value* associated with it given a policy π. The value of a state $s \in \mathcal{S}$ under policy π is denoted $v_\pi(s)$ which can signal the "worth" of the given state and is defined as the expected total reward by following policy π from state s:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \end{aligned}$$

The optimum value of a state $v_*(s)$ is defined as:

$$v_*(s) = \max_\pi v_\pi(s) \ \forall s \in \mathcal{S}$$

Similarly, an action can have its own "worth" or *quality* assigned to it under a specific

policy. The quality of an action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ under policy π is defined as:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \end{aligned}$$

and its optimum $q_*(s,a)$ is accordingly defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \ \forall a \in \mathcal{A} \ \forall s \in \mathcal{S}$$

As previously discussed, an optimal policy π_* is any policy which maximizes the expected reward received. Furthermore, policies can be compared. A policy π is greater than another policy π' if and only if the value of all states under policy π are greater than or equal to the value of all states under π'

$$\pi \geq \pi' \text{ iff } v_\pi(s) \geq v_{\pi'}(s) \ \forall s \in \mathcal{S}$$

Policy Evaluation Policy evaluation is the iterative process of approximating the state-value function v_π for some policy π. By repeatedly ... TODO ...

Algorithm 1 Policy Evaluation

Require: π (the policy), θ (some small number)
  Let $V[1...n]$ be an array of values for all states ($n = |\mathcal{S}|$)
  repeat
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
      $v \leftarrow V[s]$
      $V[s] \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$
      $\Delta \leftarrow \max(\Delta, |v - V[s]|)$
    end for
  until $\Delta < \theta$
  return $V \approx v_\pi$

Policy Improvement and Iteration

Generalized Policy Iteration

Value Iteration

Monte Carlo Methods

Algorithm 2 Policy Iteration

1. Initialization
   $V(s) \in \mathcal{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all s

2. Policy Evaluation (see Algorithm 1).

3. Policy Improvement
   policy-true $\leftarrow$ *true*
   for all $s \in \mathcal{S}$ **do**
   old-policy $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
   if *old-action* $\neq \pi(s)$ **then**
   policy-stable $\leftarrow$ *false*
   end if
   end for
   if *old-action* $\neq \pi(s)$ **then**
   return $V \approx v_*$ and $\pi \approx \pi_*$
   else
   Go to Step 2.
   end if

Algorithm 3 Value Iteration

Require: V initialized arbitrarily (e.g. $V(s) = 0 \; \forall s \in \mathcal{S}$),
θ (some small number)
repeat
$\Delta \leftarrow 0$
for all *something* **do**
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
end for
until $\Delta < \theta$
return Deterministic policy $\pi \approx \pi_*$ such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$

Algorithm 4 Monte Carlo with Exploring Starts

Initialize for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

- $Q(s, a) \leftarrow$ arbitrary
- $\pi(s) \leftarrow$ arbitrary
- $Returns(s, a) \leftarrow$ empty list

Repeat forever:

- Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0
- Generate an episode starting from S_0, A_0, following π
- For each pair s, a appearing in the episode:
  1. $G \leftarrow$ return following the first occurrence of s, a
  2. Append G to $Returns(s, a)$
  3. $Q(s, a) \leftarrow \text{average}(Returns(s, a))$
- For each s in the episode:
  - $\pi(s) \leftarrow \text{argmax}_a\, Q(s, a)$

2 Literature Review

In this section, an overview of major advancements in reinforcement learning as it has been applied to the domain of games is given. Following that, previous research into the topic of cribbage is presented.

2.1 Reinforcement Learning

History

From Samuel's seminal work in checkers [Sam59] to more recent accomplishments against Grandmasters of Go [SHM+16], the domain of games has been an area of great interest in machine learning research. Games provide an isolated environment with a set of distinct rules and clear objectives which make them well suited to being expressed as Markov decision processes [Sam59]. This in turn allows for modeling as a reinforcement learning problem and subsequent exploration.

No paper on reinforcement learning of sufficient size would be complete without mention Arthur L. Samuel's foundational paper on machine learning applied to checkers [Sam59]. In this paper, despite the technological limitations of his time, Samuel develops and describes two main methodologies he used to make an agent learn how to play checkers.

The first method Samuel devises is what he called "Rote learning." This strategy applied an arbitrarily decided polynomial value function to evaluate board positions. Future positions were then found by searching a minimax tree of limited ply depth in which each agent was assumed to play optimally given its own position and using an early form of alpha-beta pruning. Although only a certain number of moves were allowed to be searched forward, a record of previously evaluated positions was stored. These kept records would then be used to virtually extend the depth of

the search tree: if the ply depth of the search is three and a previously evaluated position was located as a leaf on the search tree of the currently held position, then the effective depth of the tree along this route is now six since the stored position has already been evaluated to a depth of three. Samuel even recognized the idea of rewards being necessary for learning a task with what he called a "pushing mechanism" which would direct the agent towards winning the game and not just picking better positions.

Although the learning aspect of his program was limited to merely evaluating deeper and deeper plies and saving results, Samuel's program was trained through self-play, recorded games, and occasionally through human interaction. The recorded, or "book games" as Samuel terms them, provide a mechanism for supervised learning in which the agent could track its own choices against those made by the humans in the actual game, providing another method to update the value of a board position. After training, and without imposing human knowledge such as bank of openings, the playing program was evaluated to play on the same level as a better-than-average novice.

The other learning method Samuel explains is what he called "generalized learning" and was a preliminary simulated annealing or hill-climbing algorithm which was applied to improving the board position evaluation polynomial. In the paper, two agents would play against each other: Alpha, in which the polynomial parameters could be tweaked, the other (Beta) using the same starting parameters but never updating them. If Alpha won enough games in a tournament, Beta would inherit Alpha's parameters and the adjustments would begin again. This had the effect of incrementally improving the learned parameters. If Alpha was incapable of consistently beating Beta, then a local optimum was recognized and Alpha's polynomial would be randomly mutated to another starting position, smoothing out changes as needed to avoid large leaps in the search space. Although the reader is cautioned that the learning method is not guaranteed to find globally optimal parameters, Samuel's proposed hill-climbing technique creates an agent which is capable of playing a better than average game of checkers.

Perfect Information Games

Perfect information games are those in which the entire game state is observable at a given time [Kho10]. Furthermore, an action has a definite and entirely predictable outcome: e.g. moving a chess piece on the board in a specific arrangement will always lead to the same resulting arrangement. Checkers, chess, and Go are all examples of perfect information games.

Jumping ahead by nearly 40 years, the basic ideas of Samuel's paper can be seen applied to the slightly different domain of chess with IBM's DeepBlue and its victory against Garry Kasparov. The DeepBlue team greatly expanded upon the idea of a minimax search to create a massively parallel search system for chess evaluation [**?**]. The final system was able to evaluate hundreds of millions of chess positions per

second, millions of times faster than Samuel's capabilities at the time. However, the speed in evaluation was a result of hard-wiring an evaluation function at the chip level using chips custom-built for the purpose. As with Rote learning, very little in terms of actual learning how to play or adjustments to internal mechanisms occurred beyond an expanded search space made possible by more capable hardware. Despite this philosophical gripe, the IBM team was able to outplay a Grandmaster of chess, winning the match four games to two.

Of notable attention recently is Google DeepMind's success in the game of Go with AlphaGo [SHM+16]. More satisfying than DeepBlue's search approach, AlphaGo used multiple neural networks to evaluate value and policy functions. A first neural network was trained using a database of human-played games to predict human play for a given board position. Using this network as a starting point, a policy network was trained along with a value network which worked together to determine the optimal next move for the agent. Both of these previous networks were trained through self-play. Eventually, the network resulting from this self-play was able to beat Grandmaster Lee Sedol in match play by four games to one.

DeepMind then took a step further, removing all human-imposed knowledge and training a single neural network to play *tabula rasa*, i.e. from scratch. Using only this neural network and a tree search algorithm, AlphaGo Zero, the new program, was able to defeat the previous AlphaGo 89 games to 11 [SSS+17].

Stochastic Games

In contrast to the predictability of perfect information games, there exist games in which the entire game state is not fully observable or in which outcomes involve random chance. These latter games, in which state transitions proceed according to a set of transition probabilities, are called *stochastic* [Sha53].

A pivotal example of machine learning applied to stochastic games, and a heavy influence in AlphaGo's creation, was TD-Gammon [Tes95]. In his paper, Tesauro designed and trained a simple feed-forward neural network to play backgammon. Backgammon is an ancient board game in which moving abilities are determined through dice rolls. The inclusion of dice rolls provides a stochastic environment since a desired outcome may not be the actual result of making a move. As a result, the searching methodology of DeepBlue or Samuel's checkers is not applicable as there is no guarantee of reaching a given state. Instead, the neural network is trained to predict the likely outcome of a game with a given position through observing several recorded games. After a game had finished, the graph's weights were readjusted by using a final reward signal representing the actual outcome of the game.

Since predictions made later in the course of the game were expected to be more accurate than those made earlier since earlier states could lead to a wider range of resulting states, a system of diminished rewards was implemented called TD(λ) based on the idea of temporal difference. TD(λ) updates the weight differences

according to the formula:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

where α is a learning rate parameters, $\nabla_w Y_k$ is the gradient of the output with respect to the weights, and λ controls how far back credit is assigned to an outcome or more accurately how much that credit decays. For instance, $\lambda = 0$ means only the last evaluation is credited, while $\lambda = 1$ means all evaluations are credited equally and values in between provide smoothly decaying credit assignments.

Later experiments extended training through the use of self-play. TD-Gammon would play both sides of a single game and adjust its evaluation network accordingly. This training method was proven to be highly successful when TD-Gammon was entered into the 1992 World Cup of Backgammon tournament and was shown to be a contender, losing by only 7 points total over the course of 38 games with highly respected players and even former world champions. In qualitative evaluations by Kit Woolsey, one of backgammon's most respected analysts, TD-Gammon was declared to play better than humans in situations which were considered more complex as its evaluation function was able to precisely calculate the chances of winning without bias or emotion. TD-Gammon has so impressed the backgammon community that strategies previously considered inferior due to human bias have been revisited and reanalyzed due to their favorability in TD-Gammon.

Tesauro credits this success to both the self-play method of training as well as phenomena directly inherited from the nature of the game of backgammon. The motion of the game is reliant on the outcomes of random dice rolls, the nature of the outcomes of these rolls naturally contributes to the exploration of a wide search space without the need for explicit exploration steps as would be needed in a perfect information game such as chess. Furthermore, since checkers can only be moved forward with the exception of being knocked back by an advancing opponent, the game cannot enter a non-terminating loop and must eventually terminate.

In addition to the backgammon community, TD-gammon made leaps and bounds in the area of games AI research, especially with respect to neural networks. After the first few thousand training games, the system was capable of playing elementary backgammon, recognizing a few basic strategies for decent play. This was determined to be the result of simple combinations of features which could be quickly evaluated by a linear function of the raw board position. More complicated, context-sensitive strategies would develop later in training and are credited as being the result of the ability of neural networks to adapt to nonlinear features.

While Tesauro credits the neural network architecture and the game itself for the success of the TD-Gammon system, Drs. Pollack and Blair at Brandeis University undertook to demonstrate that TD-Gammon's successes were not a result of temporal difference learning or even reinforcement learning necessarily, but due to the nature of backgammon itself and how self-play contributes to a constantly evolving environment [PB97]. In their efforts, Pollack and Blair applied basic hill-climbing

techniques to a simple neural network setup similar to that described in Tesauro's paper. To this network, the authors applied the same strategy as Samuel's generalized learning: a mutant would be created by adding Gaussian noise to the "champion" weights and the normal weights would play the mutant for a number of games. These games were done in pairs with each agent getting the chance to play as white and the same random seed to avoid unfair bias in the dice rolls. With this setup, if two agents are identical, then the result of two games would be one win each. If the mutant then won enough games, the champion's weights would then be shifted in the direction of the mutant. At no point was reinforcement or temporal difference credit used to adjust the neural network's weights; not even backpropagation was used.

With this setup, the authors were able to perform similarly to an earlier version of TD-Gammon based on comparative performance against other backgammon-playing systems. This leads to the conclusion that the success of TD-Gammon was very much so the result of the dynamics of backgammon and how they contribute to providing an excellent environment for self-play learning and less a result of the technique used in optimizing the neural network. In normal student-teacher learning, a teacher will attempt to point out edge cases of the student's knowledge and the student in turn learns by asking tougher and tougher questions. In self-play learning, the teacher and student are the same, so it is possible that a student and teacher can reach a draw situation in which both agents are placated and neither is challenged to further best the other. Characteristics mentioned by the authors include the lack of ability for a game to end in a draw, which, in turn, means that the either the teacher or student is always required to improve its performance, leading to continuous improvement.

2.2 Prior Cribbage Research

On the topic of cribbage, very little research has been done, perhaps as a result of its relative lack of popularity to such games as poker or its stochastic environment being too large to tackle. There are three main papers in which cribbage receives the main focus: a mathematical analysis of dealer advantage [Mar00], a genetic learner applied to the discard phase [KS02], and a multilayer perceptron attempt to apply a simpler version of TD(λ) to cribbage [O'C00].

In his senior thesis, Philip Martin set out to find if the player which begins as the dealer had any statistical advantage in winning [Mar00]. The method used to accomplish this task was to enumerate and evaluate all possible combinations of cards which can be dealt to a player. From there, a matrix of potential average crib scores was created as a table indexed on each axis by one of $\binom{52}{2}$ combinations, introducing the minor accuracy of forgetting which cards have been seen by each player from their respective hands. Using the bounds found, certain basic strategies were proven to be suboptimal, e.g. avoiding throwing a pair or fifteen to the crib as the pone, since at least one of their expected outcomes was found to be below the lower bound for optimal strategies. Despite the slight inaccuracy and assumption of

a uniform distribution, the paper did conclude that being the first player to act as dealer gave the player an expected advantage of approximately 5 points on average as a result of starting with a guaranteed count of a crib. However, since different strategies needed for other portions of the game were not mentioned, by the author's own admission, this study barely scratched the surface of research into the game.

The next, and perhaps most useful, study performed on the game of cribbage is Robert O'Connor's undergraduate project [O'C00] on adapting TD(λ) [Tes95] to the domain of cribbage. O'Connor trained a feedforward multilayer perceptron with only a single hidden layer to play a single hand of cribbage. He used TD(λ) to adjust the weights as training proceeded with the network playing full games against itself: no score context data was included in the input vector for training. After training for millions of games, the agent was able to choose hands reliably better than random, but just shy of an algorithm which would choose by maximum expected outcome. As there was no previous work done to produce an AI to play cribbage, no comparison could be made. Furthermore, as the results of games vary wildly due to luck of card games, an infeasible amount of games would need to be played against a human to determine if any advantage was present, and thus a comparison was not made to human play either. Regardless, the system was demonstrated to be learning as it played significantly better than random over the course of matches of one thousand games each.

It should be noted that the paper lacks significant details as to how the network was used, precisely. The output of the network was a single linear output representing the state utility function at any state. These states were represented by 209 binary values representing various cards in hand or in play. There is no mention of how this state space is explored or how this value function is applied to the agent making a decision; it can only be assumed that the agent made the greedy choice among potentially reachable states for each possible legal move.

The final paper worth addressing which attempts to learn cribbage is Kendall's and Shaw's adaptive cribbage player [KS02]. The authors use an evolutionary strategy, similar to a genetic algorithm, to make an agent learn to play the discard phase of the game. In doing so, the limitation was imposed that the suit of a card would not be considered, sacrificing a small degree of accuracy in calculations to vastly simplify the search space. For hand dealt, the evolutionary algorithm attempted to learn a set of weights for choosing which cards to keep and which to throw into the crib, in order to optimize its own hand. The choices made were contrasted against optimal possible choices for that hand when including the cut card in order to determine fitness and weights were updated if the resulting hand scored below a threshold percentage of optimal. While initial results were positive and learning occurred alongside an increase in points obtained by the agents, performance eventually returned to levels on par with the initial randomized weights. This was because values associated with each card converged until suboptimal behavior was reintroduced.

Additionally, the authors attempted also to use a co-evolutionary strategy in which two agents would learn to adjust their weights for each hand when playing each

other by using the opponent's score as comparison for weights adjustment. This proved to be detrimental to agent performance when different hands were dealt to each competing agent as a result of the fact that one agent must always update its weights as the loser and because losing may be unfair since the values of hands can vary widely. However, when the agents were trained by using the same set of cards, results were far more positive since the losing agent was always challenged to improve on its present choice in a fair manner.

The paper continued by prompting the evolutionary agent to learn a different set of weights for the hand when playing as either the dealer or as the pone. As a demonstration, a single set of dealt cards was set to the task and different sets of weights were learned for each position.

As a final experiment, the trained agents were played against a commercially available cribbage-playing program on various difficulty setting levels. To supplement the developed card-choosing agent, a simple heuristic was included to play the pegging portion of the game. This conglomerate agent was able to outperform the opponent program easily on its 'easy' difficulty. The matches were more evenly matched between the agent and the program on its 'medium' difficulty, narrowly winning three of five games. The 'hard' and 'harder' difficulties proved too much for the simple pegging heuristic and the agent lost all five games.

Thus, the authors have demonstrated that an agent can be made to learn how to choose a combination of cards well using evolutionary methods. Furthermore, they have demonstrated that the agent could be trained to recognize differences in play present when playing as the dealer or as the pone.

3 Methods

In this section, an in-depth presentation of how the experimental learner was set up is provided. Multiple basic strategies which a human player would use to play cribbage are presented and a mechanism for their combination is given. Finally, the intended training mechanism for the agent is covered as well as a variety of experimental changes made to determine the precise nature of learning.

3.1 Strategies

A few basic behavioral strategies were coded for the agent to learn over the course of multiple simulated games. These represented most of the basic factors which a player will consider when making their decision as to which cards to keep as well as some that are not quite as useful. These included:

- `hand_max_min`: The hand(s) which has the highest minimum possible score for the kept cards will be more highly desired. This is equivalent to choosing the hand with the largest guaranteed points.

- `hand_max_avg`: The hand(s) with the maximum average points over all possible cut cards will be the most highly desired. This strategy is useful for trying to maximize the expected score of one's own hand.
- `hand_max_med`: The hand(s) with the maximum median points possible to score will be the most highly desired.
- `hand_max_poss`: The hand(s) with the maximum possible score will be the most highly desired. This strategy can be thought of as a Hail Mary choice for the player trying to score as many points as possible, not taking into account its low likelihood to become reality.
- `crib_min_avg`: The hand(s) whose tossed cards led to cribs with the lowest average amount of points for the dealer is the most highly desired. This strategy would be a very defensive strategy typically used by the pone to avoid giving points to the dealer.
- `pegging_max_avg_gained`: The hand(s) with the maximum average points gained through pegging is the most highly desired. This strategy would be useful for end-game play in a tight game. For instance, if both players are close to winning, the dealer may choose to forego placing points into their hand and instead try to "peg out" since it is unlikely that they will get a chance to count their hand at all.
- `pegging_max_med_gained`: The hand(s) with the maximum median points scored during play will be the most highly desired.
- `pegging_min_avg_given`: The hand(s) with the minimum average points scored by the opposing player will be the most highly desired. This is a very defensive strategy also useful at the end of the game to prevent the opposing player from pegging out.

The above definitions refer to a hand's desirability. This can be thought of as an internal ranking of how likely a given strategy would be willing to choose a particular combination of cards. This desirability score is then scaled to lie in the range $[0, 1]$ with 1 representing the best possibilities. This scaling allows for possibilities which are "almost as good" to not be ignored in later weighting stages.

Because of the massive amount of possible combinations—many of which were unique due to the cards' position affecting the scoring outcome—the evaluation of some of these possibilities in which the crib was involved in real-time took a handful of seconds to evaluate during development. This was deemed much too slow as over the course of hundreds or thousands of simulated games, this delay would very quickly accumulate to performance-affecting delays. As a result, an alternative strategy of pre-computing the required knowledge and storing the values of interest into a database was implemented instead.

There are $\binom{52}{6}$ possible combinations of card which can be dealt to either player. Of these possibly dealt hands, there are then $\binom{6}{4} = 15$ possible combinations of

cards that can be kept, with the remainder "tossed" to the crib. For each of these 15 possible combinations, there are a further 46 remaining possible cards that must be considered as possible cut cards for the kept hand. Additionally, for the thrown set of cards, there are $\binom{46}{2} = 1035$ possible combinations of cards which can be thrown by the opposing player into the crib as well as the 44 remaining possible cut cards. These cut cards must be considered separately in such a manner of evaluating $\binom{46}{2} \times 44 = 45540$ possibilities rather than simply looking at each of $\binom{46}{3} = 15180$ possibilities since position of the card in the crib or as the cut card can affect the score.[4] Just as crucially, there are small differences in scoring the crib as opposed to scoring the player's own hand that mean that previous evaluations' results are not reusable. For instance, the rule for gaining points from a flush are more lenient for one's own hand than for the crib: the crib's flush must be a five-card flush containing the cut card whereas the player's own hand needs only be a four-card flush of their own hand with bonus points gained from the crib also matching. This means that, altogether, there are

$$\binom{52}{6}\binom{6}{4}\left((46) + \left(\binom{46}{2} \times 44\right)\right) \approx 1.3921 \times 10^{13}$$

possible combinations of cards that need to be evaluated in total to fully understand the statistics of a cribbage game for every set of cards that can possibly be dealt.

Although the majority of this project was coded in Python for its ease of use and speed of development, due to the performance-crucial, basic mathematical operations involved, the overheads of using a higher-level language was deemed too critical and this particular tool was developed in C instead. To put the performance gains into perspective, at its fastest, most parallelly processed state, the Python database populator was estimated to take approximately 4 months to simply list and evaluate all possible scores on a development machine, disregarding the file I/O operations required to write the results to the database. The same functionality, with the addition of file I/O, in the C program would take a relatively mere 14 days on that same machine. Furthermore, access to a high-performance server allowed for further parallelization which cut the final run time down to approximately 5 and a half days.

Careful consideration and preparation needed to be taken for the retrieval of this information, however. The trillions of combinations could not be quickly searched by card value as doing so would search the entirety of the database on each lookup, decreasing the performance so much as to be worse than simply enumerating the combinations on-demand. A rather simple solution to this problem was to search by index instead of by card comparison. These indices could not simply be stored in the memory of the running program because the sheer size required to store all of the indices at all times would rival that of the database, and its population would take considerable enough time. Thus, a quick and reliable method for creating and

[4]For example, as per the right-jack (his nobs) rule, a hand of 5♣ 5♡ 5♠ J♢ with a cut card of 5♢ is a perfect hand with a score of 29, but moving those cards around so that the jack is no longer in the hand and is instead the cut (i.e. hand of 5♣ 5♡ 5♠ 5♢ with a cut of J♢) yields a score of 28.

recreating these keys needed to be used. As the cards were represented internally as an integer between 0 and 51 (inclusive) and there were only 6 cards for indexing, the concatenation of the cards' digits in (keep,toss) order would create a number with at most twelve digits, with an absolute maximum value of 484950514647, well within the range of numbers addressable by a 64-bit integer.[5] This index could be created by a very simple process of 6 multiplications and 5 additions while still being guaranteeably unique.

While the combinations of chosen and tossed scores could be evaluated before any games had actually been played, the hands' usefulness during the pegging portion of the round needed to be evaluated in semi real-time. A single pegging agent was programmed with a simple one-ahead greedy heuristic: the card that gained the most points when played next was selected with ties broken by choosing the higher-valued card that reached that score to also potentially help outmaneuver the opponent from reaching a count of 31. This agent was then played against a copy of itself with randomly allocated cards and the results of that round were recorded for each agent to provide an initial knowledge base. These results would then be queried by the agent during the choose phase of the game and contributed to at the end of the pegging phase as training progressed.

3.2 Weighting

For the purposes of this project, how well certain combinations of cards lend themselves to being played with different strategies is not directly explored. Instead, only the player's position in score-space affects the decision as to which strategies to play by. Put another way, the agent does not care what cards it is dealt as much as where it is located on the board. Each possible score-space location can be thought of as a discrete coordinate defined by the parameters $\mathit{PlayerScore} \in [0, 120]$, $\mathit{OpponentScore} \in [0, 120]$, and $\mathit{Dealer?} \in \{0, 1\}$. At each score-space location is a vector $w_{p,o,d} = [w_1, w_2, \ldots, w_m]$ where m is the number of all possible strategies to be considered. At the beginning of each round, each of m strategies is evaluated for all $n = \binom{6}{4}$ possible combinations of cards kept to produce an $m \times n$ matrix S where $S_{i,j}$ is the desirability of the i^{th} keep/toss combination according to the j^{th} strategy further constrained by $0 \leq S_{i,j} \leq 1 \ \forall i, j$. A value vector p of length n representing the total perceived value of a possible keep combination can then be computed by $p = wS$ wherein $\operatorname{argmax}_x p_x$ can be thought of to be the most desired combination of cards and $\operatorname{argmin}_x p_x$ as the least. These collective desirability metrics can be later used to determine which combination of cards to choose and which to toss. A visualization of this weighting mechanism is shown in Figure 1.

[5] Thanks to implementation, the order of cards was guaranteed to be sorted within each tuple, so each combination of cards was tracked and not permutation.

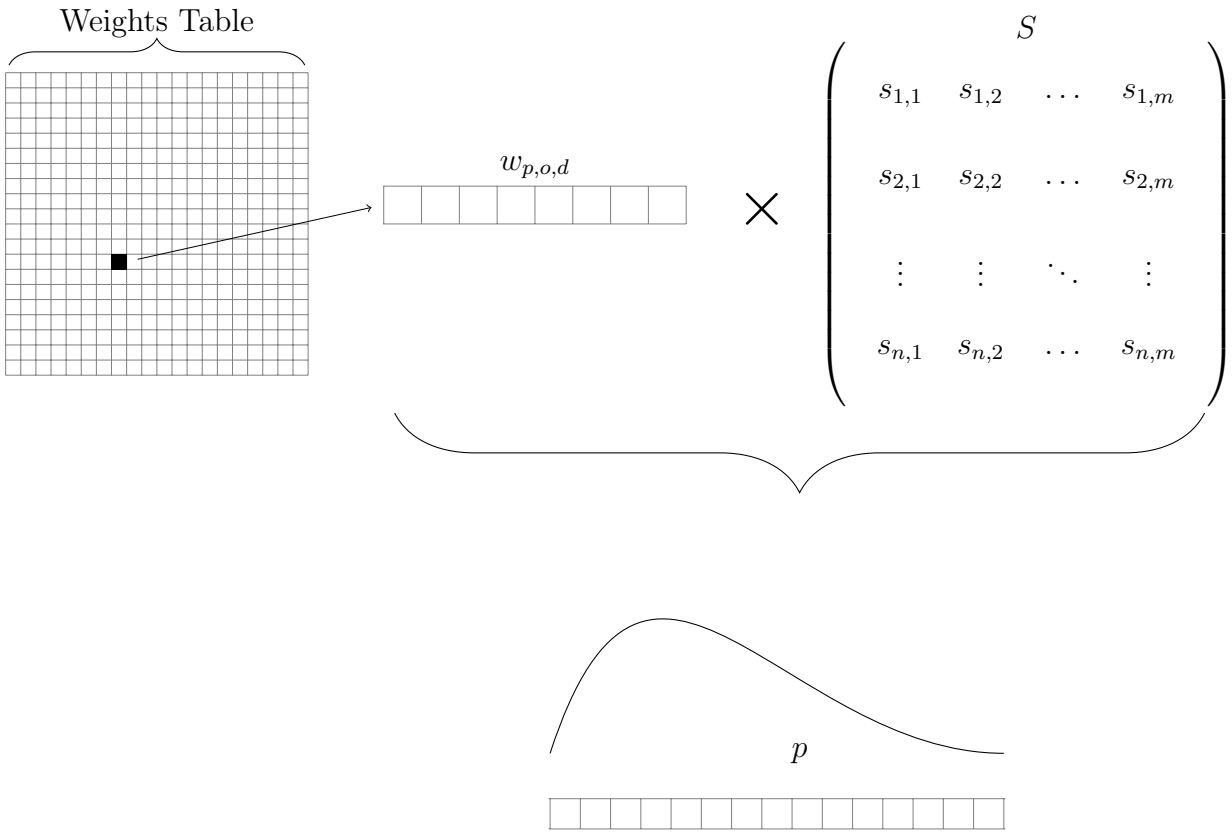


Figure 1: A visual depiction of the weighting operation.

3.3 Training

After a complete game has been played, the winning and losing agents need to modify their weights in order to decide a "correct" strategy at that time coordinate. In contrast with textbook forms of reinforcement learning, the agent is not directly learning which cards to take and throw. To do so would require a search space of $\binom{6}{4}\binom{52}{6}$ at each state. Instead, the agent is learning which subset of strategies make the best decision in combination at a given point. Therefore, there is no deterministic single action taken at any given time. Instead, the strategies which advocated most for the chosen hand would be held responsible for the success or failure of the given hand and thought of as the action at the given step. The strategies which most advocated for the hand were determined as those whose values in S were the highest in its column. This subset of strategies, carefully chosen to only include the top percentage of contributors to avoid stagnant resulting weights, would then be adjusted by a percentage of itself according to the formula:

$$w_{i,new} = cw_{i,old}$$

for all indices i which are in the most-highly-advocated group and where c is a shared adjustment constant. All other weights would be left alone before re-normalizing the weights vector. Since locations earlier in the game have a higher number of potential

resulting states and are less likely to affect the final outcome of the game, they are modified less than their later counterparts. This is accomplished by decaying the adjustment amount for each step taken backward along the visited path of states in manner similar to TD(λ):

$$c_j = C \cdot (1 - d)^{T-j}$$

where j is the index along the path starting at 1, C is a starting value of the adjustment constant, T is the final step index, and d is the rate of decay expressed as a ratio such that $0 < d < 1$ [Tes95]. In order to allow for closer losses to be less severely punished than worse losses and vice versa for wins, the adjustment constant C was defined as proportional to the difference between the players' scores according to the formula:

$$C = s \cdot (MyScore - OppScore)$$

where s is a scaling factor analogous to a learning rate parameter. This scaling factor remains constant throughout training and does not decrease as it would in something like gradient descent. This is because all games are equally important, so a loss using strategy i during game x is just as important as it would be during game y.

The resulting effect is to slightly reward or punish the weights corresponding to only the strategies which were most in favor of choosing the chosen cards. Note that this is not necessarily the same set of weights which were the highest contributors to the final choice. It may be the case that weights x and y are the highest at a given point, but that strategies u and v with slightly lower weight values had higher advocacy for a given hand which summed up to a larger value. Were this not the case and instead the highest weights were to be directly punished or rewarded instead, without looking at their position on the current hand, it is more likely that a single group of weights would simply cycle back and forth in value, never allowing other strategies to be explored.

The reinforcement training framework operated very simply. A simple text file of weights can be loaded to initialize an agent. These agents would then be placed into a game and played against each other. After the game had been completed, the weights for each agent would be adjusted accordingly along the path which the agent took. After a set number of epochs, the agent would save its weights configuration to a checkpoint file. This allowed for the convenient benefit of being able to track how weights adjusted over the course of time.

To facilitate an adequate rate of exploration of the search space, randomized initializations were used for each of the training games. A random score was chosen for each of the agents, keeping the spread of scores to within 60 points in order to limit the search space to only imaginably likely reachable situations. This value was chosen since, with a maximum point total of 121, it is highly unlikely to get into a situation where one player is half the board behind the other. While there have been anecdotes of losses by more than that amount, it is a very rare occurrence and can be thought of as a failure of luck rather than of skill. As this is a matter of training skill and proficiency in the game of cribbage, situations of exceptional bad

luck can be treated as an outlier situation.

A further measure was taken to ensure adequate exploration. Under normal conditions, cards were selected by choosing the combination which had the highest value in the produced p vector. At any given point in a training game, however, there was a chance of selecting which cards are played by random by using p as a probability distribution. This chance e of random choice was related to the variance of the weights according to the formula:

$$e = k - \mathrm{Var}(w_{m,o,d})$$

where k is set at a constant 0.3 empirically chosen during the development process. This was implemented in order to ensure that situations in which there were more uniform weights had a higher chance of being explored whereas those with a higher variance were deemed to be varied enough to have been previously trained.

The training was intended to take the form of a tournament of learning in which better and better agents would be paired off against each other until an ultimate agent was reached, allowing for more experience to be gained for agents from a variety of sources. After a set number of training games, in the case of this project one million, the two agents were played against each other in a tournament match fashion. The winner was determined as the agent with the most points at the end of a series of games, in this case 100, wherein two points would be awarded to the winner of a game and three if that win was by a margin of at least 31 points and ties broken by total point spread, according to ACC rules [ACCa]. After the match had completed, the winner would advance to the next round, training against another winner from the previous round. The process would be repeated until one agent is declared the ultimate winner.

After the tournament structure was found to not provide any further benefits to the learning process, a set of additional experiments were performed to determine the limits of learning possible. While these will be covered in more detail in 4, these modifications include simple items such as learning rate or decay rate adjustments as well as more complicated adjustments including using neighboring weights or punishing a loss less severely than a corresponding win is rewarded in addition to sanity checks by starting from pure strategies.

4 Findings

In this section, the results of the training rounds will be presented. After the tournament was found to be reaching a plateau in performance, a series of experiments was performed in an attempt to improve learning. These experiments include parameter tweaks as well as adjustments to the learning process itself.

4.1 Tournament

Round 1

Round 1 consisted of 32 agents with randomly allocated starting weights paired off against each other. Each pair of agents played one million games against each other, each game starting at a random score location, learning and reinforcing their weight vectors after each game.

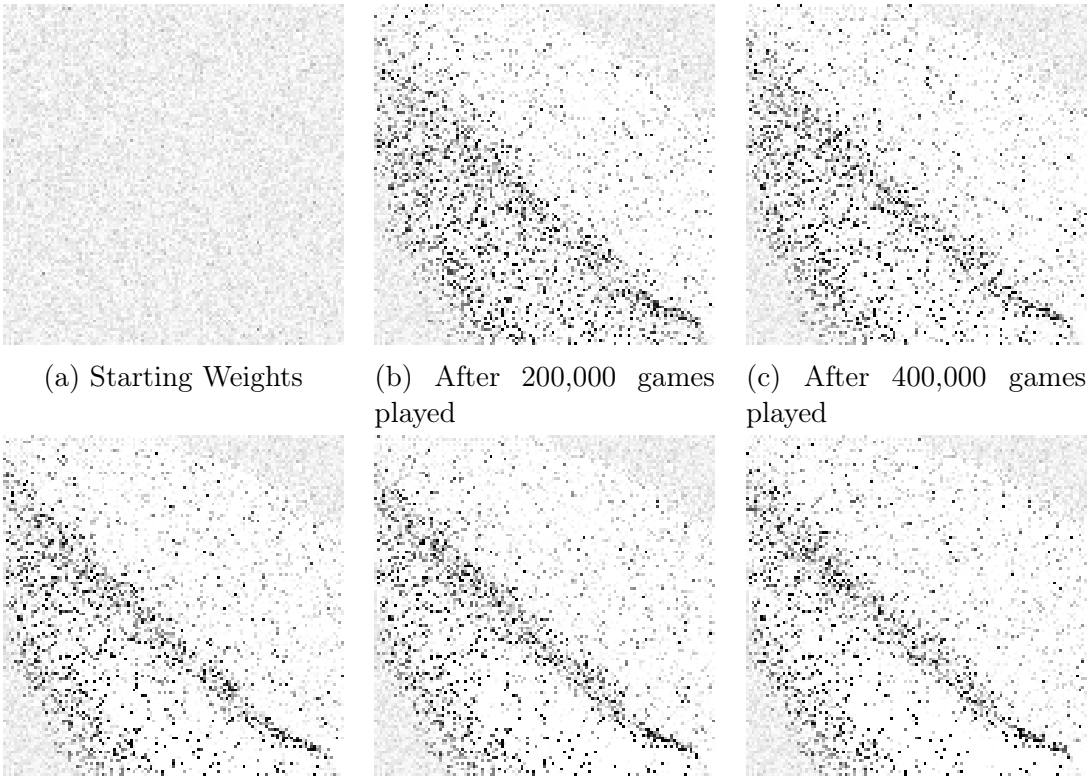
Learning Process The results of the first round's training on a sample agent can be seen in Figure 2. Each individual square within the image represents the strength of a single strategy, in this case `hand_max_avg`, where white means completely absent and black means completely dominant. Each image was taken at an intermediate stage to capture and show transitions.

There are two things to note from these results. The first, the stark contrast in colors in the majority of the image. The other, the area in which these stark contrasts are present.

In the starting phase, all weights are randomly assigned and relatively uniform with only slight variances, hence the blurry dull gray appearance. As time progresses, the image becomes crisper and filled with more contrast. This indicates not only stronger preference for the strategy at the given point, but an almost all-or-nothing attitude towards adhering to a single strategy. This means there is little to no nuance to which cards are chosen and there is little to no chance for other strategies to collectively overrule or veto the major strategy.

Also of note is where the previously mentioned stark contrast is present and where it is absent. Since only those states which have been visited can have their weights influenced, the remainder will continue to stay untouched. As can be seen in the top-right and bottom-left corners, representing extremely unlikely scores to reach in which one player has achieved a rather large lead while only allowing a few points, contain only the dull gray of the initial weights. This is because even with a potential spread of 60 points when initialized, these outlandish scores are outside the realm of potential visitation. Therefore, they have not been a part of any game, so they cannot have their weights adjusted.

Learning Results Despite the all-or-nothing nature of how a single strategy is potentially learned, it is still worth noting that the agent did in fact learn to play different strategies at different times. As can be seen in Figure 3, the strengths of each strategy's weight do vary across score-locations. For instance, when in the lead by roughly two to twenty five points, the agent will prefer to choose the hand with the most guaranteed points in its own hand by following `hand_max_min`. However, when the game is either extremely close or when the agent is well in the lead, the agent will take a slight gamble and play for expected points. Occasionally, the agent will also attempt to pay attention to the points gained through the play phase of



(a) Starting Weights

(b) After 200,000 games played

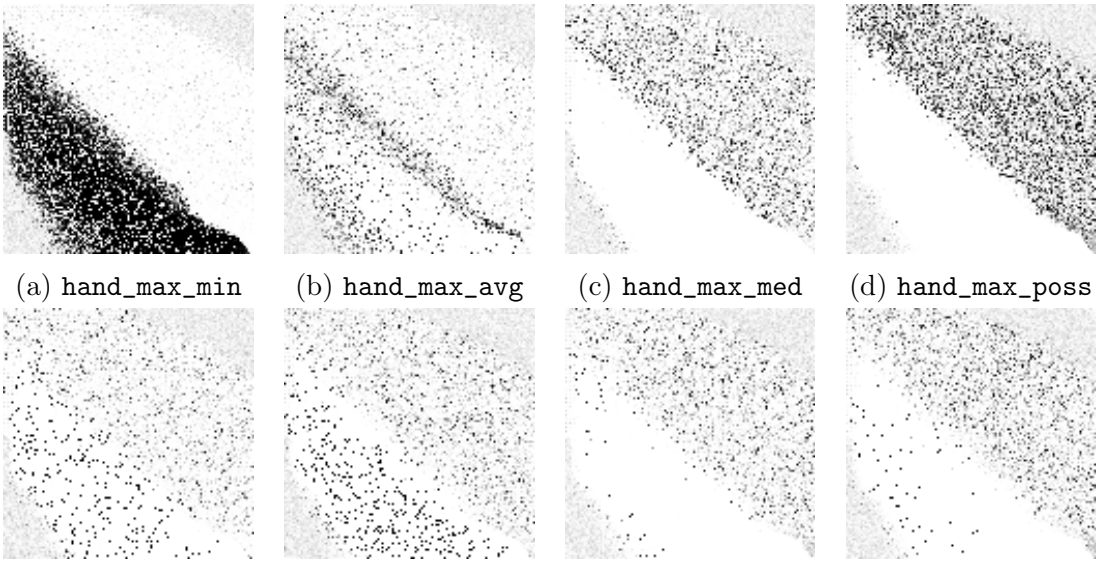
(c) After 400,000 games played

(d) After 600,000 games played

(e) After 800,000 games played

(f) Final Weights

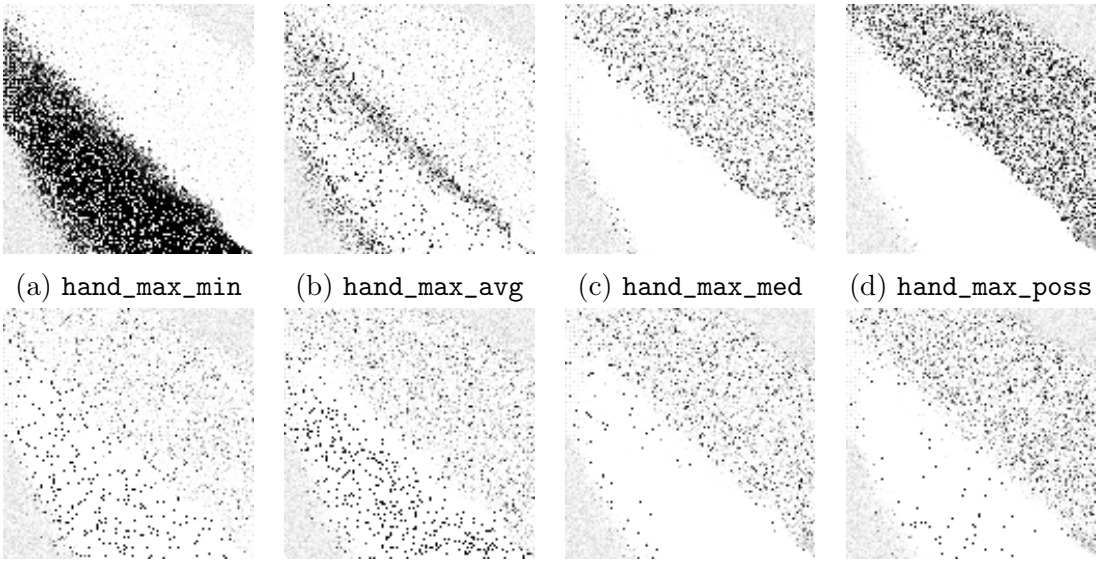
Figure 2: Training weights representation for an agent's `hand_max_avg` strategy when that agent is the dealer over the course of the one million games of Round 1. In these images, the y-axis represents the player's own score, the x-axis the opponent's score, with the origin starting at the top-left of the image.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 3: All final strategy strengths for an agent when playing as the dealer after training for one million games during Round 1.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 4: All final strategy strengths for an agent when playing as the pone after training for one million games during Round 1.

a round by playing a combination that pegs well. Ironically, the agent may play against its own best wishes by minimizing the average return of the crib. This is speculated to be a result of coincidental alignment of the results between `crib_min_avg` and more the reasonable `hand_max_min` or `hand_max_avg`.

Of further interest is how little the agent knows about how to handle a losing position. As can be seen by looking in the upper-right half of each strategy's individual graph, of the explored losing states, there is little consensus or pattern as to which strategy should dominate. It is possible that agents which end up in these positions lose more often than they win. If this is the case, the resulting punishment will decrease the top two or three strategies that were most responsible for the hand choice at that state, effectively increasing all others. This, in turn, would likely later lead to a cycle in which different strategies are cyclically placed in a role of strongest weight, generating the fuzz seen now.

By comparing the pone's strategy graphs (Figure 4) to those of the dealer's (Figure 3), a few patterns emerge. At first glance, the graphs look highly similar and as if splitting up the search space into two merely increased the complexity of finding a solution without payoff. For instance, the majority of the winning positions favor the `hand_max_min` strategy with a "border" of `hand_max_avg` being used in positions where riskier decisions are needed or can be afforded. Additionally, the losing positions offer no definitive answers and seem to be a jumbled mess of suggestions, the only major suggestion being to play more riskily to regain the lead.

However, upon closer inspection, with a side-by-side—or perhaps even overlaying—comparison, there are subtle differences that can be found. The most major of these differences is that all patterns are shifted lower and more to the left when the agent is the dealer, correlating with scores which are more advantageous to the player. This means that the agent is more "comfortable" when it has a slightly larger lead as the dealer than it necessarily needs as the pone. It also means that the unexplored state area as the pone is larger in the area where it is further ahead.

Perhaps the most interesting difference is the traceable shape of the weights during the final scores of a close game. Although difficult to discern in all but the `hand_max_min` and `hand_max_avg` strategies, the final weights trace a sinusoidal curve shape along the diagonal for approximately the last twenty points of the game. Additionally, these waves are opposing in nature. This presents the fascinating conclusion that the agent is learning that different states or checkpoints are preferred as the player when the dealer and when the pone. Not only is this fascinating since the agent is learning a state-value function indirectly, but in the game of cribbage, there exist different *checkpoints* in which the player aims to get himself by the end of a round to be in favorable position for the next. Here we see evidence that the agent is learning these checkpoints throughout its play.

Performance It is clear from the strategy graphs that, while perhaps strategies are being over-trusted, basic game trends are clearly being learned by the agent. However, perhaps the only metric which matters from a learning perspective is the

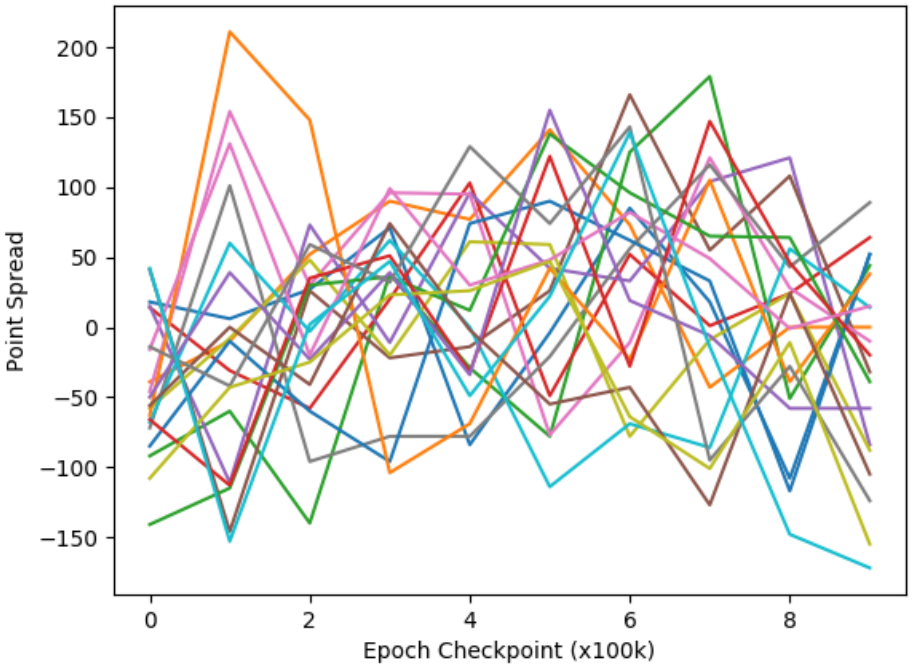
Game	Random	Trained
1	104	121
2	121	114
3	75	121
4	121	118
5	111	121
6	121	99
7	121	87
8	110	121
9	121	64

Agent	Score	Point Spread	Wins
Random	12	+39	5
Trained	9	—	4

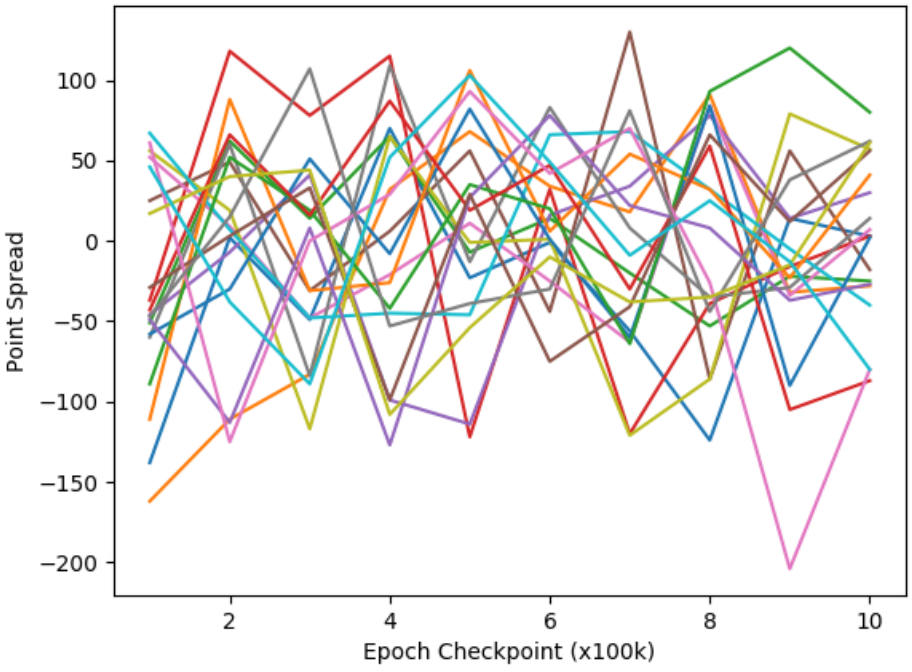
Table 2: Results of a nine-game tournament played between a randomly-weighted agent and a trained agent after learning for one million games.

agent's performance. In this area, the learned agents failed miserably. The winning agent between the two learners was pitted against an agent using randomly allocated starting weights where both agents would only strictly follow the policy generated without exploration. As can be seen in Table 2, the learned agent lost easily to the randomly-weighted agent: with the exception of a spectacular loss in Game 3, all other games were close losses if not wins. This is speculated to be a result of the previously mentioned over-aggressive learning pattern and its all-or-nothing end result. Since the agent aligns so intensely to a single strategy, it is not able to overcome a local optimum and has essentially been overfit to the scenario. Another potential reason for the losses could be the lack of knowledge of how to handle losing situations. As previously discussed, in the event of a loss, most strategies will effectively be increased except those most responsible, contributing to a system of cycling weights. Furthermore, it is also likely that an agent which finds itself in a losing position does not end up recovering and winning the game, meaning that the agent effectively learns that it is mostly by chance that it will recover.

In order to more accurately diagnose if an overfitting situation was occurring, a winning agent was played against its own previous checkpoints. A sample set of scores can be found in Table 3. The point spreads from twenty separate 100-game tournaments are depicted in Figure 5a. While point spreads are from total points pegged to the board, not gained from wins and therefore not a direct indicator of performance in tournament play, it is useful for determining general performance. Furthermore, point spreads are the parameters used for direct feedback as the rewards during the training phase, so it is a direct indicator of learning the task for which it was trained. As can be seen, there is no discernible correlation or pattern between epoch checkpoint and performance. This indicates that, despite a policy being learned, quite a lot is still left to chance of the cards dealt. This has the potential to be exacerbated even further when policies are nearly deterministic and non-adaptive in their weight assignments. Additionally, a random agent was played against its future iterations in the same fashion, with the similar results, not shown.



(a) An agent plays against previous iterations of itself.



(b) A random agent plays against later, more learned agents.

Figure 5: Point spreads across twenty 100-game tournaments pitting a winning agent against its checkpoints. Here, a positive point spread indicates that the fully-trained agent has accumulated more points than its opponent, an agent created from a checkpoint generated after the number of training game epochs indicated on the x-axis.

Epochs	Score	Spread	Wins
0	126	+523	59
1MM	86	—	41
Epochs	**Score**	**Spread**	**Wins**
200K	108	-27	47
1MM	118	—	53
Epochs	**Score**	**Spread**	**Wins**
400K	87	-510	41
1MM	133	—	59
Epochs	**Score**	**Spread**	**Wins**
600K	117	370	52
1MM	98	—	48
Epochs	**Score**	**Spread**	**Wins**
800K	124	239	56
1MM	96	—	44

Epochs	Score	Spread	Wins
100K	122	+200	55
1MM	104	—	45
Epochs	**Score**	**Spread**	**Wins**
300K	105	-180	48
1MM	116	—	52
Epochs	**Score**	**Spread**	**Wins**
500K	97	-386	43
1MM	129	—	57
Epochs	**Score**	**Spread**	**Wins**
700K	107	-174	48
1MM	115	—	52
Epochs	**Score**	**Spread**	**Wins**
900K	111	-82	51
1MM	111	—	49

Table 3: Results of multiple 100-game tournaments played between agents using various epoch checkpoints from the training and the fully trained agent from the round.

Applications for Round 2 As a result of the presumed over-aggressive learning of single strategies, some learning parameters were altered for Round 2. Since it was estimated to be the primary reason for the learning behavior, the first parameter adjustment made was to decrease the learning rate drastically. The learning rate was decreased to one fifth of what was used in Round 1. The other major parameter alteration was to make the exploration rate a constant and not dependent upon the variance of the weights. During Round 1, the exploration rate was defined as $e =: 0.3 - \mathrm{Var} w_{p,o,d}$ in order to allow more trained states to be less affected by randomized exploration. As a result, the most strongly biased weight locations would only allow for exploration even more rarely than before. Although only producing a decrease from a 30% chance to 18%, this would still result in a smaller likelihood of exploring in the current state, potentially further cementing of the dominant strategy in its position. Additionally, it was a complication which, upon further thought, would not result in much difference over the course of time and thus deemed non-beneficial and ultimately unnecessary.

In order to see if the heavily biased weights could be tempered down to more reasonable mixes which could outplay a random agent, the structure of the tournament was updated. In addition to having the winners of the previous pair of agents square off against one another, there was a "loser's bracket" created in which the losers would start over from ground zero. These two ways of playing were intended as a two-pronged approach in order to see if multiple agents could be trained at the same time which could outperform random. The "winners bracket" would determine if a highly-biased set of agents could learn nuance while the "losers bracket" would test if it were possible for two constantly competing agents could ever actually increase

performance when both are each updating their parameters to combat each other.

Since the agents were assumed to be learning how to outplay each other but not the game, the next phase would attempt to determine the extent of this tailored learning. In addition to the previously mentioned alterations to the tournament structure, a new batch of agents would be trained against a static, entirely random agent. Rather than both agents in the game learning and altering their weights after each game, only one agent would train its weights. The prevailing logic behind this decision was that if each agent was indirectly affecting the other, the environment could be said to be slightly altered. This would, in turn, mean that the agents are no longer learning the original problem.

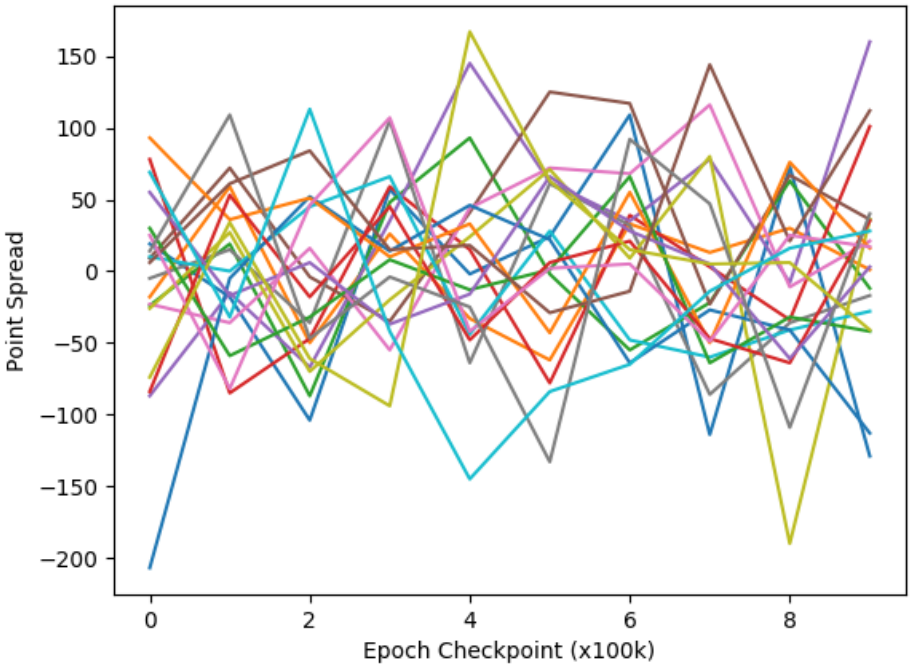
Round 2

Performance Judging by the lack of pattern found in Figures 6, 7, and 8, there is no significant performance increase to be found from Round 2. If learning had occurred, Figures 6a, 7a, and 8a would take on the form of a decreasing curve, gradually approaching zero, while Figures 6b, 7b, and 8b would be their mirror across the x-axis. Figure 6 alone would indicate that an agent trained for one million games plays on par with one trained for two million games. On its own, this would indicate that performance had reached a peak and saturated. However, the similarity to Figure 7 and even Figure 8 show that neither bracket learned to perform better than previous iterations.

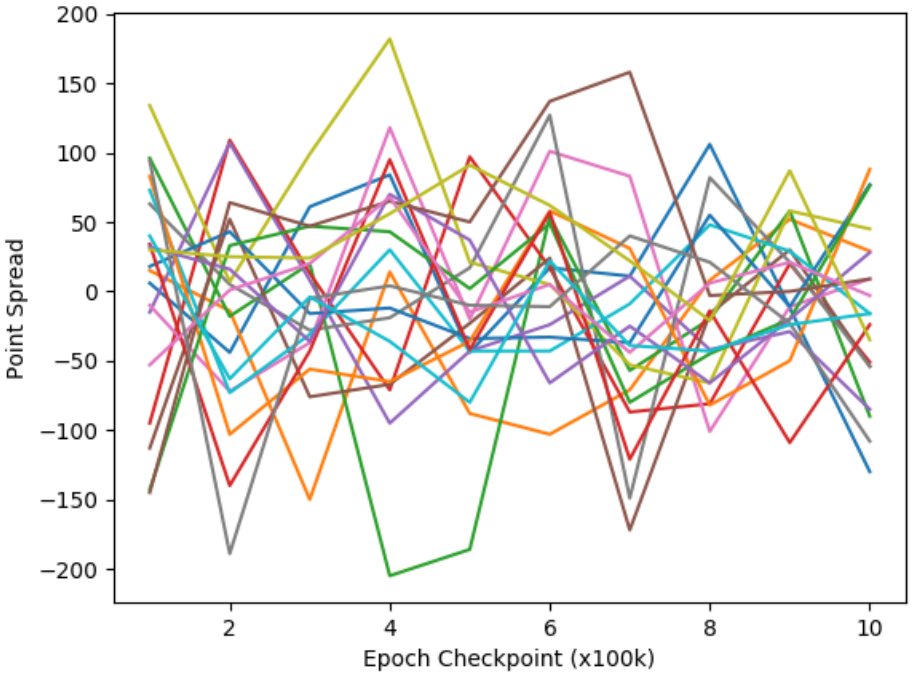
Learning Process and Results Despite the lack of performance increase after another million games played by both the winning and losing brackets, there are still interesting trends to be spotted between the different brackets of play. The most notable is how quickly policies converge to a similar state. While the strategy graphs for the less trained agents are less "crisp" in their appearance thanks to their slower learning rates, the patterns of which policy to mainly follow at which times are still quick to form. This is useful because it allows for the possibility of running future experiments with fewer training epochs in less time. This, in turn, allows for a larger variety of methods to be tested to improve the cribbage-playing agent.

Even more fascinating observations can be found from the strategy graphs from the winner's bracket (see Figure **??**). By focusing on the `hand_max_min` and `hand_max_avg` strategies in particular, the sinusoidal wave along the diagonal can be observed extending further back along the diagonal to earlier game positions, albeit with smaller amplitude. While not being of much use to the agent directly, this is a useful observation from a cribbage player's perspective. Since it is possible to infer the state-value function, the implications of this wave are that earlier states are not crucial predictors for future success. Not only that, but the agent has learned that changes in lead are likely and not necessarily detrimental to the ability to win the game.

Care needs to be taken with this previous observation on the wave's amplitude and

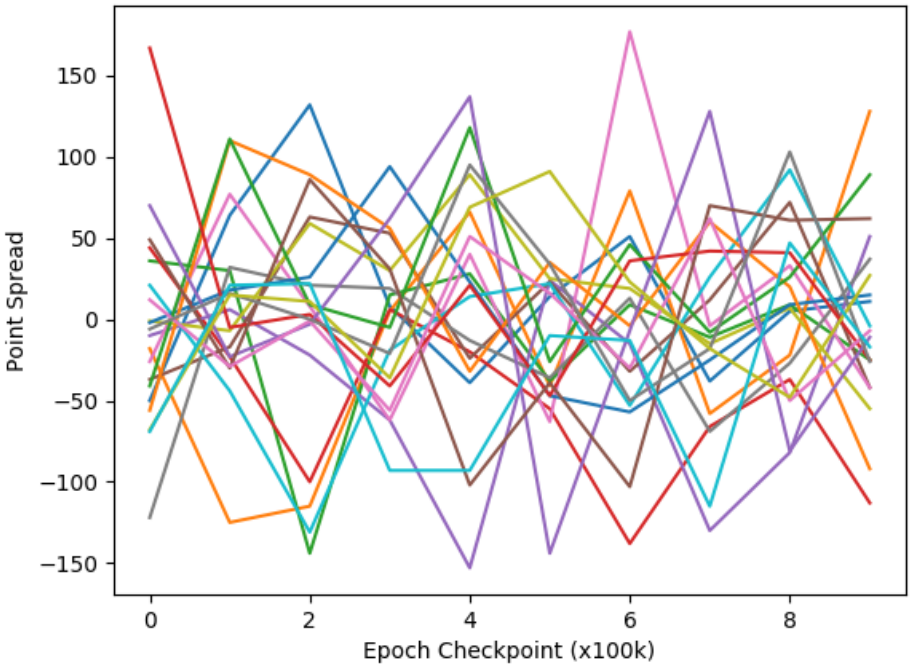


(a) An agent plays against previous iterations of itself.

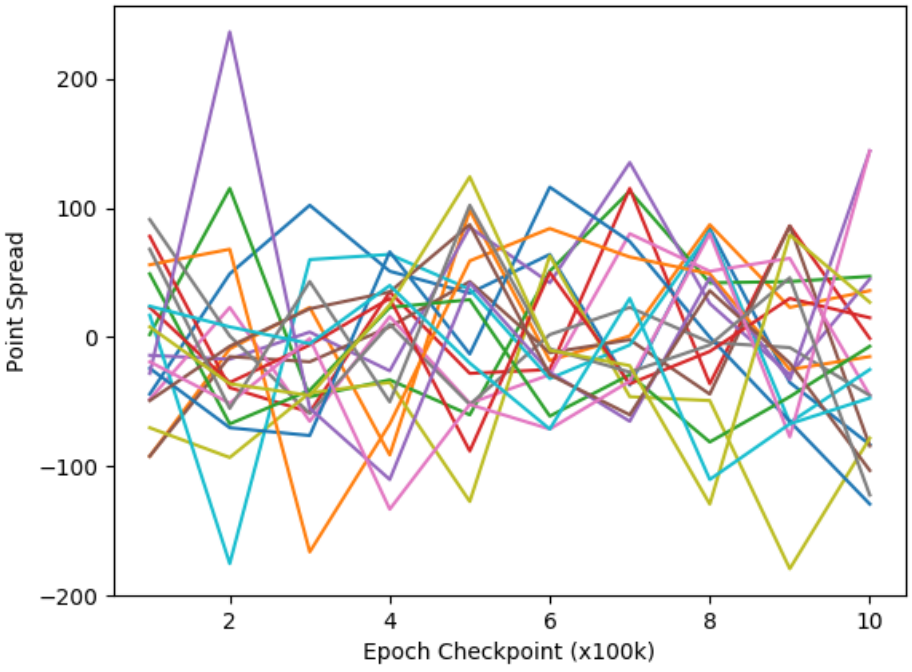


(b) A random agent plays against later, more learned agents.

Figure 6: Point spreads across multiple 9-game tournaments for an agent in the Winner's Bracket of Round 2. Note that since the winner's bracket uses an agent with prior training, the total epochs elapsed is one million more than displayed.

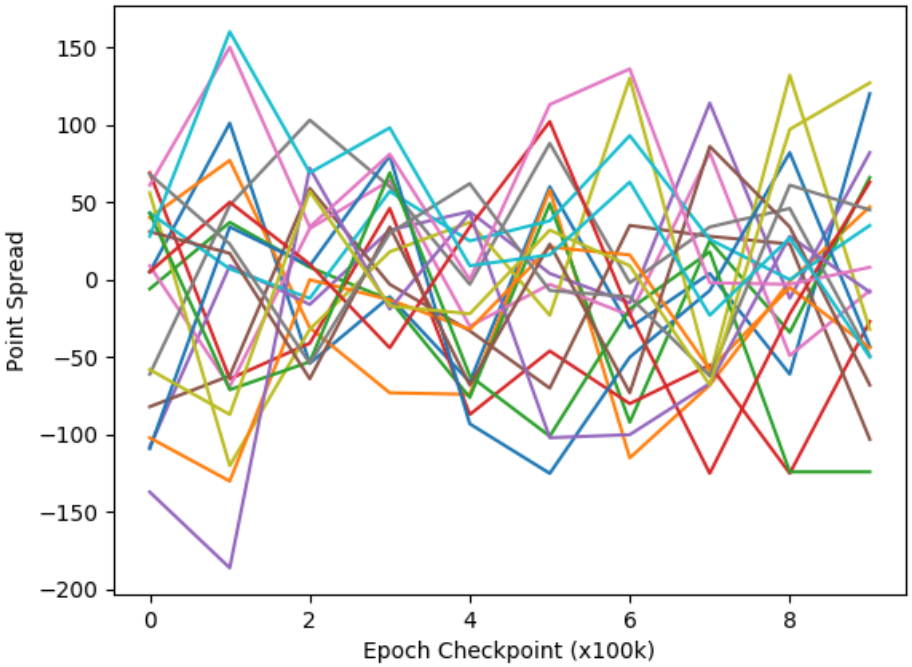


(a) An agent plays against previous iterations of itself.

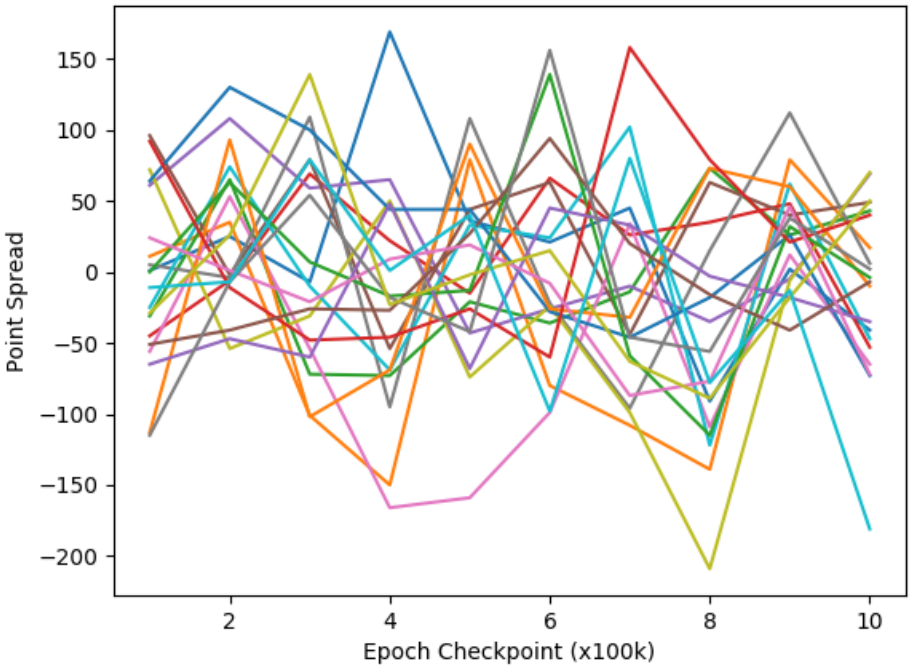


(b) A random agent plays against later, more learned agents.

Figure 7: Point spreads across mutliple 9-game tournamens of an agent in the Loser's Bracket of Round 2.

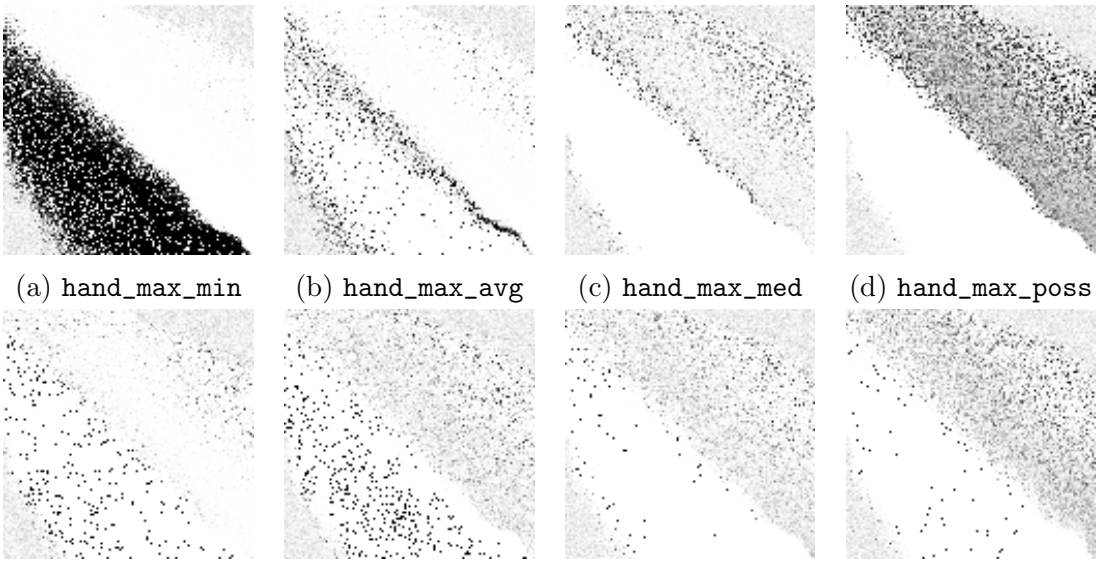


(a) An agent plays against previous iterations of itself.



(b) A random agent plays against later, more learned agents.

Figure 8: Point spreads across mutliple 9-game tournamens of an agent trained against a static, unlearning agent with random weights.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

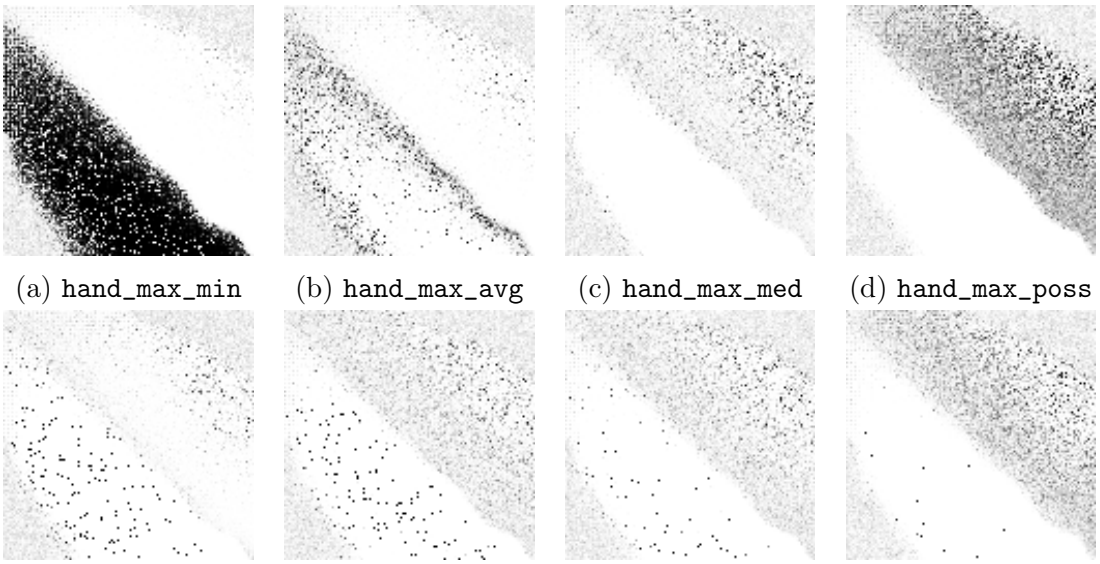
(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 9: All final strategy strengths for an agent in the "winner's" bracket when playing as the dealer after training for one million games during Round 2.

it must be pointed out to be speculation on the part of the author. It may be the case that with enough training the sine wave will be at full amplitude in earlier positions. A reminder must be made that a notable reason for the lack of clarity at the moment is the weight adjustment mechanism for training. The training system pre-supposes that the earlier positions are of less importance in a game and adjusts them with a lower priority than later positions in the game. It may indeed be the case that two million games is still not enough to counteract the decay of temporal difference learning with the learning rate used.

By comparing the winner's bracket strategy graphs to those of the loser's bracket, a vast degree of similarity can be found. In both brackets, the same behavioral trends are learned. However, there does exist a small amount of difference between the two in how quickly and surely each trend is learned. The winner's bracket mostly reinforces its current weight choices with the losing triangle becoming slightly gradiented. Meanwhile, in the loser's bracket, the gradient applies more evenly to the entire range of strategies across the winning-losing boundary. Of additional note, there are fewer spaces in which strategies such as `crib_min_avg`, `pegging_max_avg_gained`, and `pegging_min_avg_given` have been strengthened within the `hand_max_min` *block*. This is a good indicator that, although present, there is less of a bias towards those strategies which are initially winners.

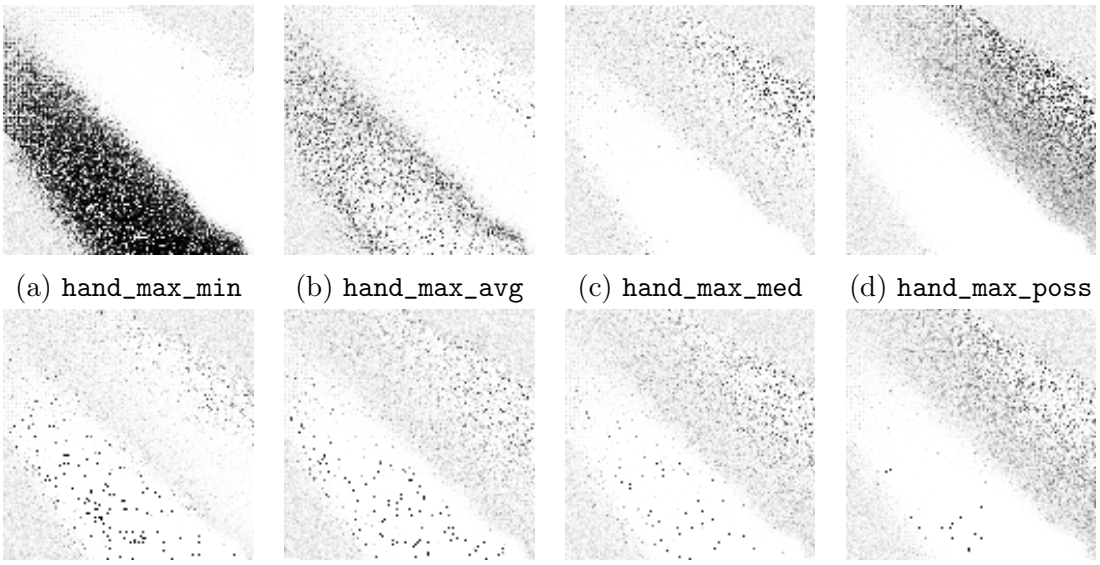
In analyzing the evolution of the `hand_max_avg` strategy in both the winner's and loser's brackets, it can clearly be seen that the general trend of behavior forms relatively quickly, i.e. after the first half million games played. Beyond that amount of games, the agent learns to refine the strategy boundaries, but it can be said that



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

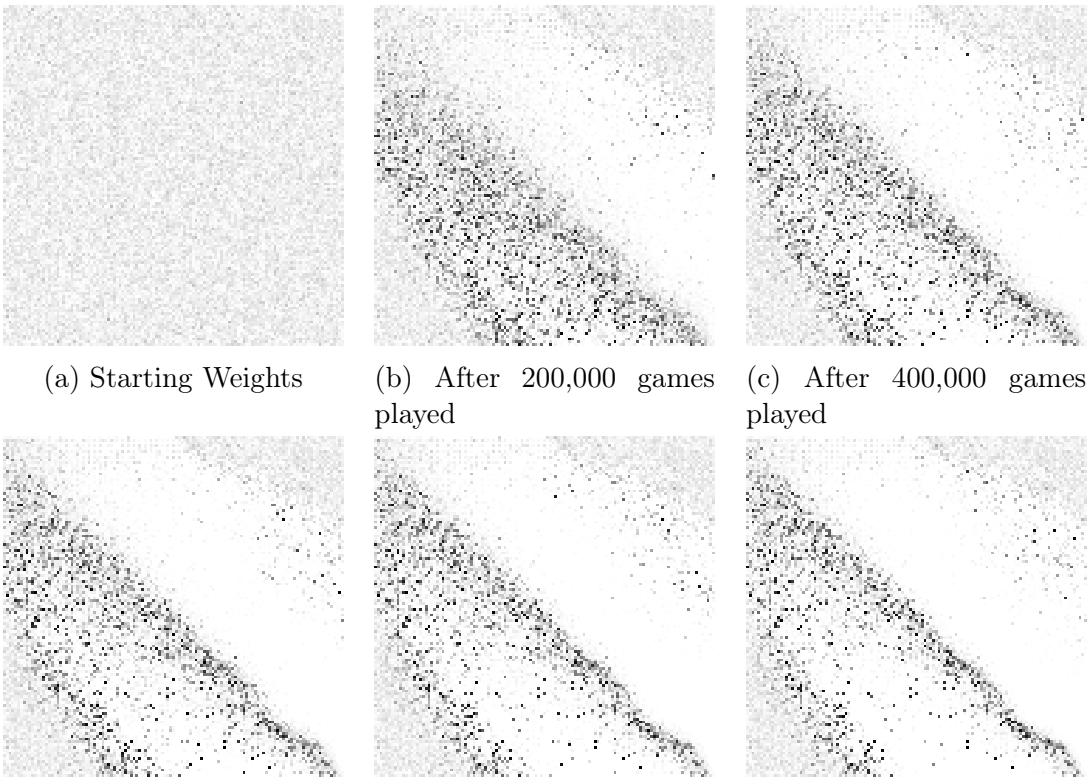
Figure 10: All final strategy strengths for an agent in the loser's bracket when playing as the dealer after training for one million games during Round 2.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 11: All final strategy strengths for a learning agent after playing a completely random agent when playing as the dealer after training for 350,000 games during Round 2.



(a) Starting Weights

(b) After 200,000 games played

(c) After 400,000 games played

(d) After 600,000 games played

(e) After 800,000 games played

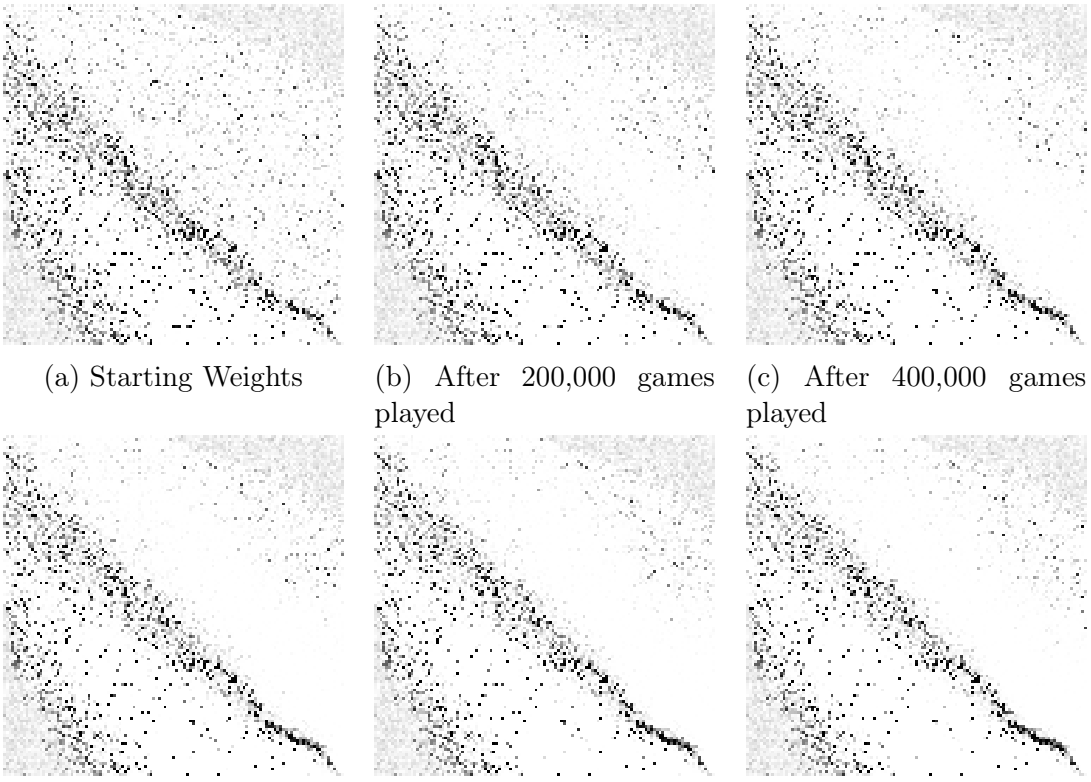
(f) Final Weights

Figure 12: Training weights representation for a loser bracket agent's `hand_max_avg` strategy when the agent is the dealer over the course of the one million games of Round 2.

there is little discovery being made. This is useful because it allows for quicker training rounds to be played, and thus more possible variations in training can be tried in the time given with the goal of improving beyond random play.

Potential Issues The comparable performance to random weights must be addressed as a potential error in implementation rather than in training. Since all methods of training so far have resulted in very similar policies being learned, it is fair to say that policy learned during training has been superior to random in some manner. There are then a few items to consider with regard to why such poor results are occurring during the review tournaments.

Overfitting The first item to consider as the source of the discrepancy between the observation that a policy is learned, but its performance is abysmal is that the weights table is overfitting to the problem. This would have been especially true of the first round where learning rates were much higher. However, if the agents were to be overfitting, a number of differences would be found in their results. Firstly, a different set of policy graphs would likely result from the training phase which is not



(a) Starting Weights

(b) After 200,000 games played

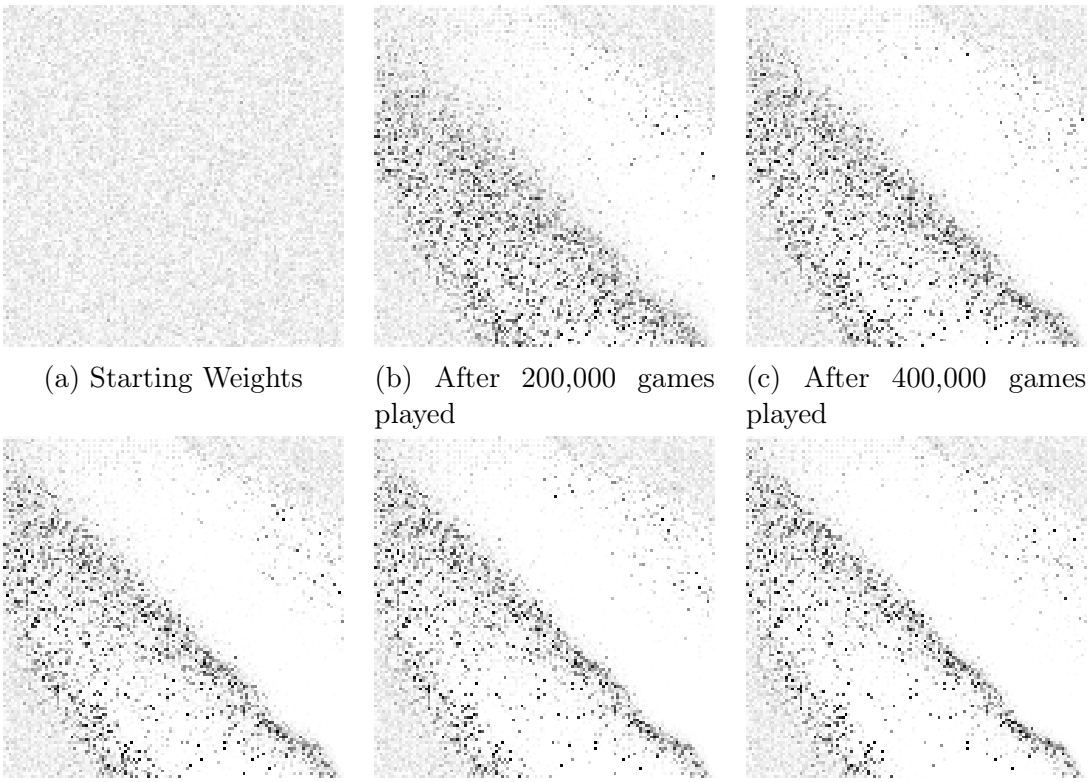
(c) After 400,000 games played

(d) After 600,000 games played

(e) After 800,000 games played

(f) Final Weights

Figure 13: Training weights representation for a winner bracket agent's `hand_max_avg` strategy when the agent is the dealer over the course of the one million games of Round 2. Note that the starting weights are carried over from Round 1, so the total training epochs to reach each position is actually one million higher than expressed.



(a) Starting Weights

(b) After 200,000 games played

(c) After 400,000 games played

(d) After 600,000 games played

(e) After 800,000 games played

(f) Final Weights

Figure 14: Training weights representation for an agent's `hand_max_avg` strategy over the course of one million games in which the opponent always played using randomly allocatd weights.

the case. More importantly, a definitive curve would be present in the tournament graphs as performance would increase before dropping. As can be seen in Figures 6 and 7, this is not the case as performance is not apparently linked to training in any way. Therefore, it would seem safe to conclude that overfitting is not the reason for the poor resulting performance.

Scale of Play The prevailing theory as to the reason for which the agent learns a policy, but following that policy does not yield positive results in performance, is the issue of scale. The agent is trained on a million games, but tested on only a handful. Since the cards being dealt are not taken into account by the policy, the decisions made will likely be inaccurate on the scale of a single game, but not for thousands.

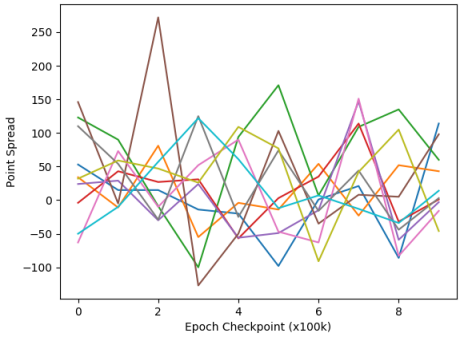
A regulation cribbage match between two human players consists of nine games. As can be seen in Figure 15b, in a series of 100-game matches between a fully trained agent and its previous checkpoints—using a Round 2-trained agent in the loser's bracket for demonstrative purposes—no pattern is discernible in total point spreads between the two playing agents. This demonstrates that, on this scale, the winner of the game is no more predictable than a truly random coin toss.

When the scale is increased to one thousand games, a slight pattern begins to emerge when visualized in Figure 15c. Whereas the majority of the graph remains highly varying and unpatterned, the first few games follow a common pattern. The match against the random agent is still unpredictable, but the matches against the 100,000 and 200,000 game trained checkpoints are consistently beaten by the final agent.

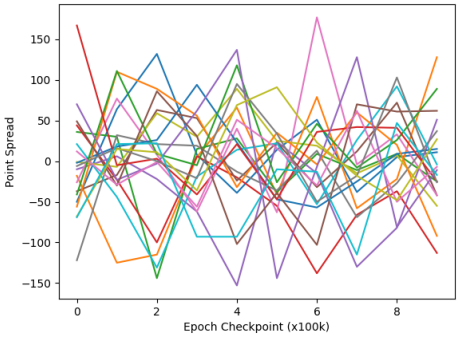
Although fewer matches are played to compensate for the increased time needed to play the increased number of games, the same pattern present in the thousand-game scale is visible in the ten thousand-game matches. With the exception of the purely untrained agent, the least learned agents perform the poorest against the final learned agent. Play is approximately evenly matched between the fully trained agent and its checkpoints after training for 300,000 to 700,000 games. Following these matches, the agent begins to, again, win more consistently when more training is done.

If the agent is being trained correctly and no overfitting is occurring, then the point spread should be a positive number, gradually decreasing and approaching zero as more training is applied to its opponent, forming a similar curve to a loss metric used in classical machine learning. In the event of overfitting, the curve would dip below the zero before reapproaching zero. Neither of these shapes were seen in the resulting graph (see Figure 15d). Instead, the shape of the curve implies that, since the random agent performs consistently better than the trained agent, the learning process is not learning the game so much as how to outplay its opponent in some corner case that does not reappear during testing. This conclusion is supported by the observation that the agents with limited training are steadfastly outplayed.

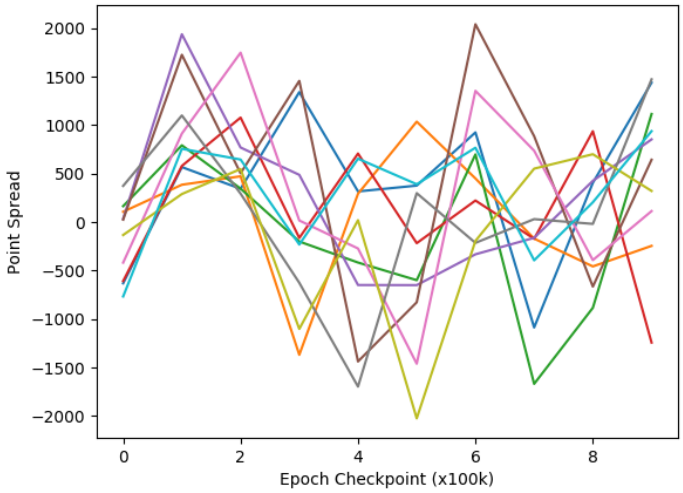
However, since there is a sharp dip after 300,000 training epochs before increasing again, the results are difficult to interpret. The final learned agent is more evenly



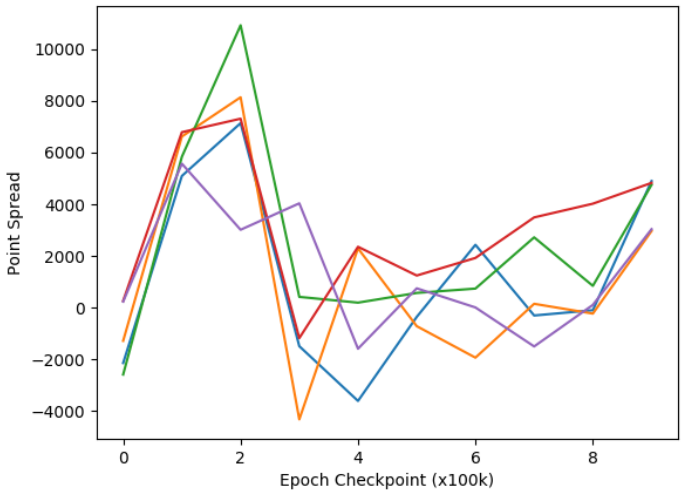
(a) Point spreads for 9-game matches



(b) Point spreads for 100-game matches.



(c) Point spreads for 1,000-game matches.



(d) Point spreads for 10,000-game matches.

Figure 15: Point spreads across matches of varying lengths. In each graph, the final learned agent is played against its previous checkpoint iterations.

matched in performance with an agent trained for 400,000 games than it is against a more recently trained agent. The reintroduction of an increasing curve is an indicator of overfitting in the middle stages. Whereas the final agent is able to play well against these agents, they themselves have decreased in performance capability. This can be speculated to be a result of learning how to outplay its opposing agent rather than an understanding of the game, but similar performance when the opposing agent has fixed random weights—as can be seen in comparisons between Figures 6 and 8—counters this postulate.

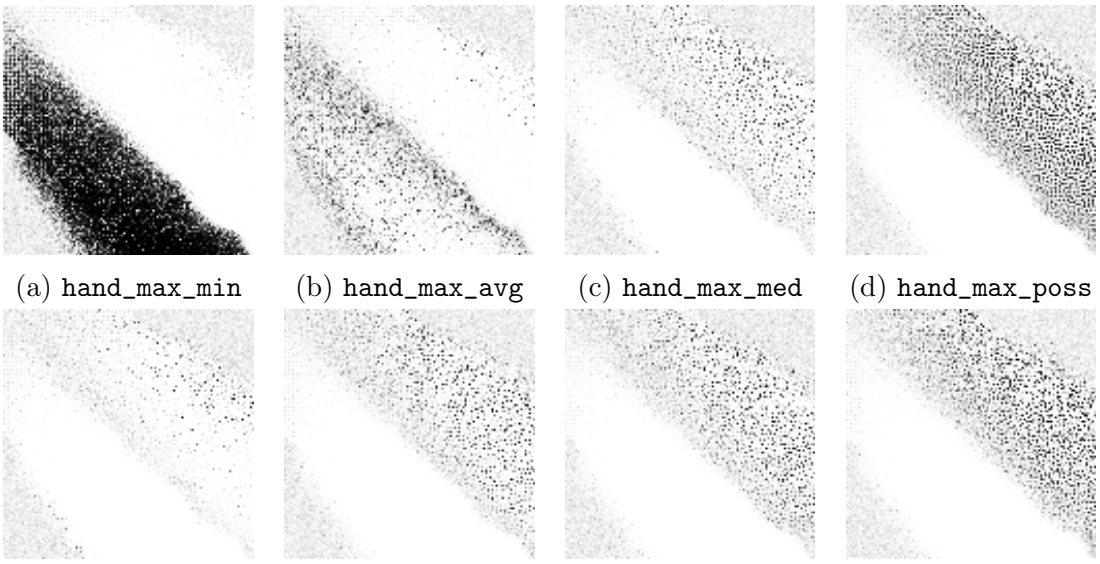
Also of further note, is the scale on the aforementioned graphs. In Figure 15d, the maximum point spread achieved is just a shade above 10,000 points over the course of 10,000 games. This means that the average point spread advantage is approximately one point per game at peak performance—and most often 0.2 points per game—in long-term play. The average spread increases to 25 points per game when fewer games are played, but since performance is unpredictable on this scale, this can be explained as a result of the randomness of the cards given and cannot be considered a reliable measure of performance. In fact, in sampled matches, it was not uncommon to observe losses and wins of 30 points or more as well as much closer games with margins of only a few points. In addition to the randomness of cards dealt, this massive point sway could also be the result of the agents' uncertainty on how to recover from a losing position. While the reasons for this inability cannot be said to be more than speculation, the learning of a policy directly without taking into account cards dealt is the likely culprit, as explained previously.

4.2 Further Experiments

As the results of Round 2 emulated those of Round 1, additional modifications were applied to the learning process. These methods focused on directly affecting the weights applied to each decision made, both in learning and at choosing time. The intended goal of these modifications were to find a policy capable of learning a policy which plays the game consistently better than random. All modifications for these experiments started with the following common parameters:

- Scaling Factor: $s = 2.0$
- Decay: $d = 0.1$
- Random Exploration Chance: $e = 0.3$

Additionally, the pegging records database was standardized across all experiments as one selected from those resulting from the first round of training. As a final note, the mechanism for choosing a combination of cards using the $\hat{p}$ vector during the tournament was altered slightly. As previously mentioned, when making an exploration step, instead of using $\hat{p}$ as a probability distribution, the choice was made at uniform random, in a manner true to ϵ-greedy exploration [SB17]. This was done for the sake of simplicity. Furthermore, the difference in choosing mechanisms



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 16: Final strategy graphs for an agent when playing as the dealer after being trained for 500,000 games by following a policy which uses a weighted sum of neighboring weights.

demonstrably made no noticeable difference in the learning behavior, perhaps as a result of $\hat{p}$ being near-uniformly random for untrained states and in essence a choice between three or four most desirable choices when states had been visited.

Neighboring Weights

In order to smooth out the strategy graphs and prevent isolated states of separate weights, a blending of neighboring weights was developed. Rather than simply take the set of weights allocated to a single score location, the agent instead takes a weighted average of all surrounding weight vectors with its own location. In other words,

$$w'_{m,o,d} = \left| X w_{m,o,d} + Y \sum_{i \in \{-1,0,1\}} \sum_{j \in \{-1,0,1\}} w_{m+i,o+j,d} \right|_1$$

where X, Y are ratios of each vector's effect, $X + Y = 1$, and $m + i$, $o + j \in [0, 120]$. The desired effect was to allow a score location to learn from its neighbors so that a neighborhood effect was present in the decision.

Results The agent was not capable of weighting the different strategies in a gradient-like manner by using the neighboring score locations' weights vectors. In fact, the `hand_max_avg` strategy, which one would expect most to form a gradient with and around the `hand_max_min` strategy, was perhaps the most negatively af-

fected. As seen in Figure 16, the `hand_max_avg` strategy graph has become more sparse in the winning triangle (lower left) with very little gray area surrounding the black.

Interestingly, while the neighboring weights did not solidify any presence in the graphs, the new training method did make progress in negative space. This is to say that the weighted neighbors learning method eliminated the usually present islands of a single strategy within a swathe of another dominant strategy. This white-space was present in both the winning and losing triangles. This finding leads to the conclusion that weighted learning techniques do indeed allow for a sharing of knowledge between like states, but only insofar as to know what not to do.

Furthermore, there is still a vast degree of uncertainty present in the losing triangles of the strategy graphs. The theory that the certainty of the winning triangle might potentially *bleed* over into the losing side was not demonstrated over the course of the first half million training games. However, since the elimination of islands shows an improvement in preventing lucky happenstance dictate a space's future, the neighboring weights training method has shown usefulness in training a cribbage agent in which states neighboring states are often similar in nature.

Regularization

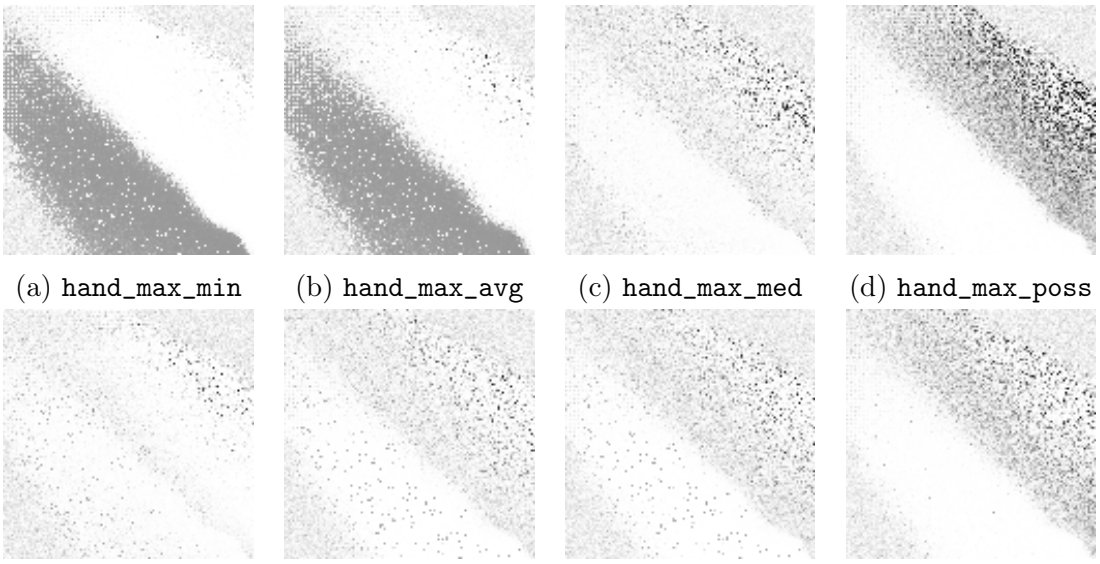
As an attempt to prevent a single strategy's weight from being so strongly preferred that a second strategy could not hope to possibly gain ground, a hard limit was placed on the pre-normalized update value. Expressed mathematically,

$$w'_{m,o,d}[i] = \max\{K, cw_{m,o,d}[i]\}$$

where K is some constant value throughout the training. While the value of $w_{m,o,d}[i]$ could exceed K after re-normalization for a particularly strongly weighted strategy, the value could be seen converging to K within a handful of iterations. The desired intention of this regularization was to allow other strategies the opportunity to overcome the bias of earlier strengthening of the strongest strategy.

Results As expected, the limitation of maximum attainable value did indeed force the agent to learn multiple applicable strategies for each score location. As can be seen in Figure 17, the `hand_max_min` and `hand_max_avg` strategies are now more or less equally represented across the winning triangle area. This is demonstrated by the monotone gray seen in these locations. Rather than each strategy specializing to its own dominant location, the territory is shared between the two most applicable strategies. Therefore, the regularization does indeed prevent the dominance of a single strategy unintentionally discarding all other potential strategies.

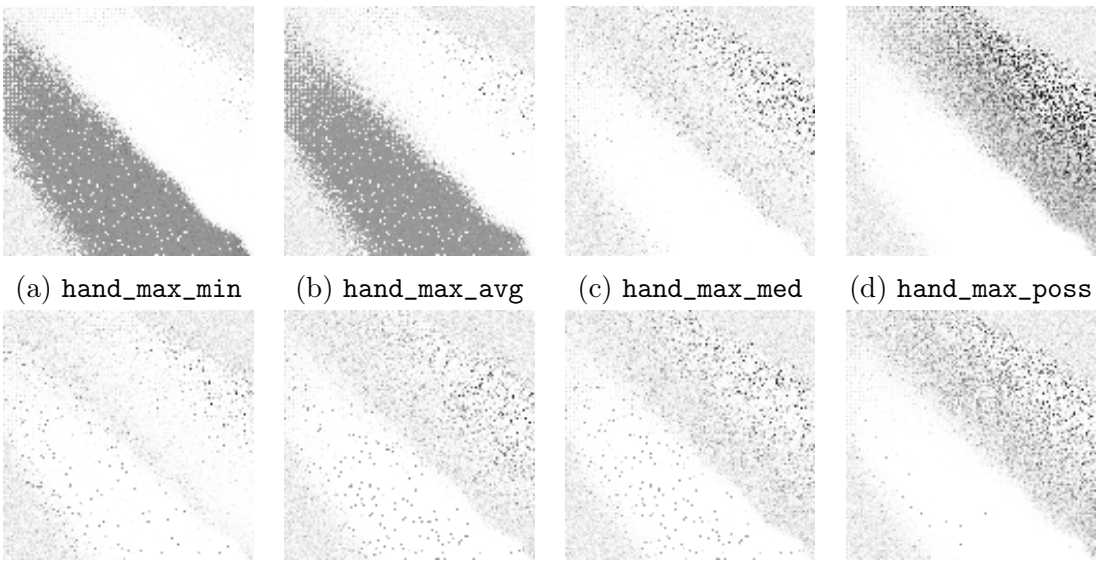
Additionally understandably, the nature of this gray mass alters significantly as the regularization rate is altered. With a lower allowed maximum value, the strategy graphs take on the previously mentioned slightly amorphous gray blob shape. As the



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

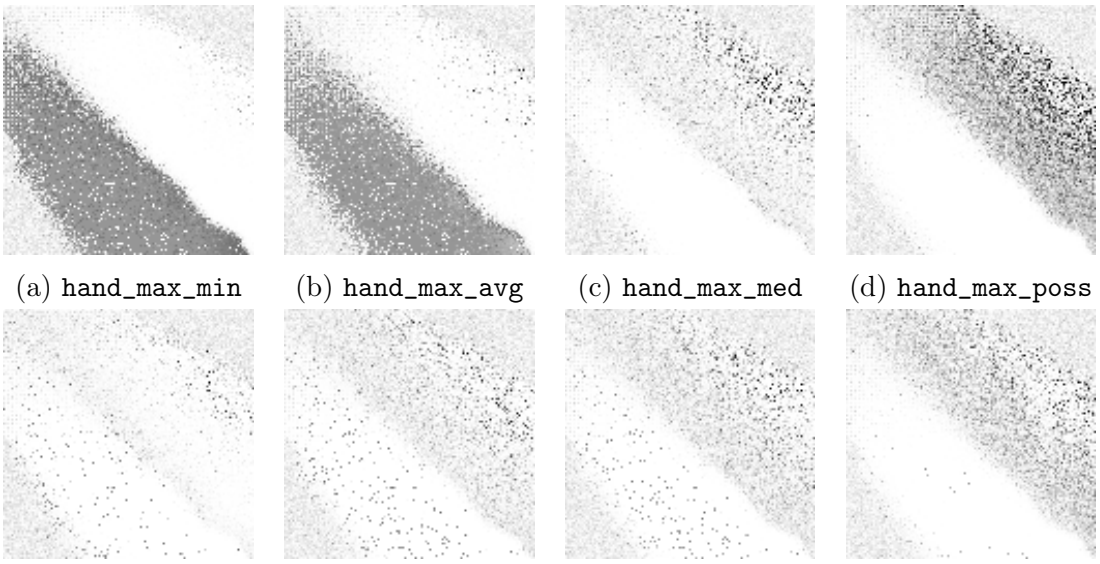
Figure 17: Final strategies for an agent using regularized learning when playing as the dealer and the maximum value weight allowed is 0.50 after training for 500,000 games.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

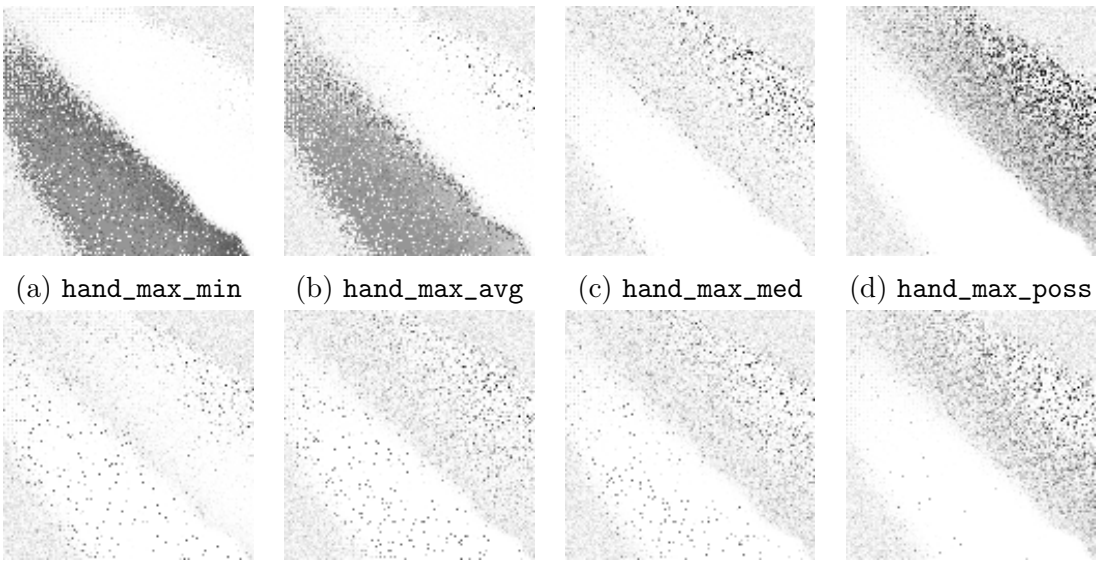
Figure 18: Final strategies for an agent using regularized learning when playing as the dealer and the maximum value weight allowed is 0.60 after training for 500,000 games.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

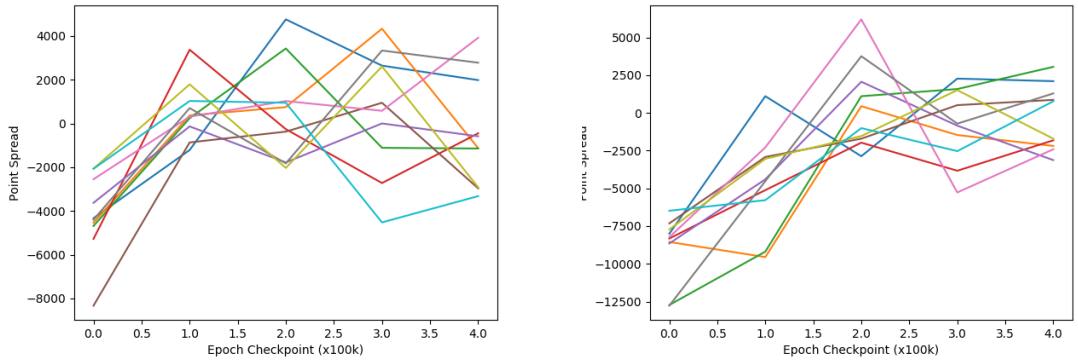
Figure 19: Final strategies for an agent using regularized learning when playing as the dealer and the maximum value weight allowed is 0.70 after training for 500,000 games.



(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

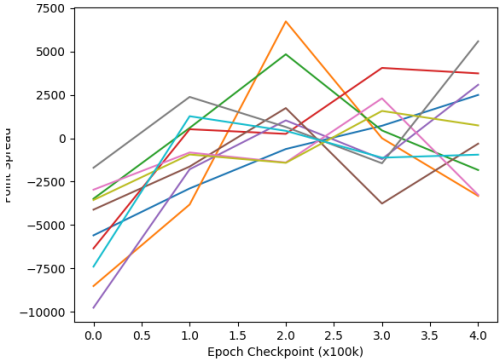
(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 20: Final strategies for an agent using regularized learning when playing as the dealer and the maximum value weight allowed is 0.80 after training for 500,000 games.

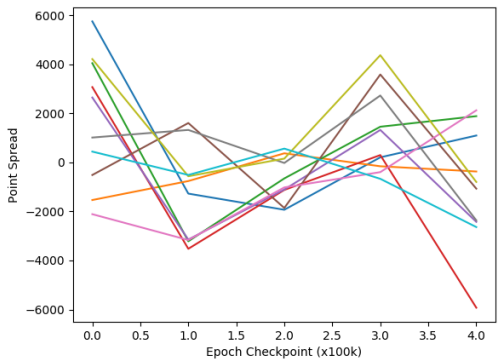


(a) $r = 0.50$

(b) $r = 0.60$



(c) $r = 0.70$



(d) $r = 0.80$

Figure 21: Point spreads of self-tournaments for 10,000 games carried out between agents trained using the regulation method mentioned in Section 4.1 after being trained for 500,000 games. Each tournament plot in n this figure is referenced by the regularization rate r in use during training.

maximum value approaches one, the gray shape begins to specialize more and begins to show resemblance to the final strategy graphs obtained without regularization.

Furthermore, as can be seen in Figure 21, only a regularization rate of $r = 0.80$ (Figure 21d can be said to be similar to a loss curve. With rates of $r = 0.50$ (Figure 21a, $r = 0.60$ (Figure 21b), and $r = 0.70$ (Figure 21c), the tournament point spread curves show that a trained agent plays on par with its recent checkpoints, but consistently worse than random weights. The endpoints of the point spread curves using $r = 0.80$ show a decrease from better-than-random play for the trained agent to more-or-less on-par performance with its later checkpoints, indicative of an increase in performance as training progresses. However, the increases and decreases of performance along the time axis the wide range of spreads present and the sinusoidal nature of intermediate checkpoints' spreads leads to the conjecture that, perhaps, performance is not quite as predictable as that.

Starting Points

Since a desired outcome of the learning process was to be able to use the generated strategy graphs to tell how a hand *should* be played in a certain score position, a comparison was made between the produced agent and pure strategies on a database of choices made by humans. There exists a website in which users are prompted with a set of dealt cards and a given score and must decide which set of cards they would keep in that specific situation [dai].

With access to recorded answers, the agent's choices could be compared to how humans ranked the choice. The retrieved database recorded which responses were given to each query and could be used to determine how well the agent's choice matched with those made by humans in the same situation. For each of the more than 3600 usable records, the choice the agent made was compared against those made by the users of the website. The results of this comparison can be seen in Table 4.

Table 4 shows that the trained agent chooses the same set of cards as the human users only marginally more often than an agent with randomly allocated weights. In approximately half of the cases, the trained agent chooses the same answer as most humans; in almost 78 percent of the cases, the answer given by the agent is within the top three most common human answers. Additionally, most pure strategies, created by setting their weight to 1 while all others are 0, performed worse than the trained mixture. Notable exceptions to this trend are the `hand_max_poss` and `hand_max_avg` strategies, suggesting that in more situations than the agent, the typical human player will play more according to what points can be expected to be gained from the cut card. Interestingly, the `hand_max_poss` strategy's presence as the second most common pure strategy used indicates a significant degree of risk-taking present in the users' responses.

As a result of this finding, each of these strategies were used as initial weights to the learning process in order to determine if the agent could learn to fine-tune a policy starting from a reasonable assertion of good game-play as well as learn to discount demonstrably poor strategies. Since the update mechanism for weights relies upon renormalization of a vector which as been rewarded or punished, no modifications would occur in the case of punishment of a pure strategy since no other weights would have the chance to increase. Therefore, the pure strategies used before were slightly modified so that each other element of the w vector would have a small initial value which would be increased when the pure strategy was punished.

Forgiving Punishments

Since it was deemed likely that being in a losing position early in the game tended to never recover and thus result in a loss, it followed that punishing losing states for something beyond the agent's control was slightly unfair. Furthermore, it was postulated that since the punishment mechanism in effect cycled strategies and that there was a possibility that an occasionally winning strategy was often outweighed

Strategy	Top 1	Top 2	Top 3	Percentage in Top 3 Human Choices
pegging_min_avg_given	160	303	458	12.64
pegging_max_med_gained	268	519	796	21.97
pegging_max_avg_gained	347	650	963	26.58
crib_min_avg	380	177	1081	29.84
hand_max_min	1576	2288	2666	73.59
Random	**1581**	**2318**	**2759**	**76.15**
hand_max_med	1649	2353	2768	76.40
Trained	**1706**	**2426**	**2821**	**77.86**
hand_max_poss	1677	2433	2847	78.58
hand_max_avg	2066	2828	3168	87.44

Table 4: Number of times the agent using a given strategy chose the same cards as the most common choice by human users according to 3623 total parsable records obtained from [dai]. The columns labeled "Top X" display the number of times the given strategy's choice was within the top X choices of the user base. In this table, **Random** is from the best result of five agents which each used independently randomly allocated weights and **Trained** uses an agent trained from Round 2 for one million games.

by the tendency to lose, a less strict method of punishment could be used to ensure that the occasional win from a losing position remains visible. As such, the update step for modifying the weights was adjusted slightly so that the constant adjustment factor was significantly smaller for losing games than it was for winning games. Instead of using the adjustment constant of $C = s \cdot (\text{MyScore} - \text{OppScore})$, for both winning and losing agents, the losing agent's adjustment factor was defined as $C = \frac{1}{4}s \cdot (\text{MyScore} - \text{OppScore})$.

Results

Learning Rate Adjustment

In order to determine if the learning rate was too high in Round 2, even though it had been significantly reduced from Round 1, a varying amount of learning rates were tried. These runs were intended to see if an optimal policy was being overstepped by making too large of an adjustment.

Results

Rate of Decay

In addition to the learning rate, the decay parameter d was also adjusted to see what sort of effect it would have on learning. Instead of the default decay rate of 10%, rates ranging from 0% to 50% were tested to demonstrate the effect of temporal learning.

Results

5 Discussion

This section will cover the reasons for successes and failures of the learning process as well as potential future applications or improvements to the system which can help make a successful cribbage-playing agent in the future.

5.1 Future Possibilities

Due to assumptions and simplifications made throughout the design and implementation process, several suboptimalities exist across this thesis which leave room for improvement for future endeavors to solve the task of learning to play cribbage well.

Pegging

One area of the project that can be improved was the pegging system. Because the focus of the thesis was on whether or not a collection of strategies could be learned in combination, an ideally playing agent was not the outcome. A key reason for this was the lack of time dedicated to and nonchalant nature towards learning how to actually play the crucial pegging phase.

Playing Strategy The first shortcoming of the pegging portion of the agent was the simplicity of its playing strategy. In order to focus on the overall playing strategy and to have a consistent knowledge of how a set of cards have performed in the past, the agent was only capable of following a very simple immediately greedy heuristic: of the cards left which are legal to play, play the one with the highest immediate return. From the cribbage player's perspective, this has the potential issue of opening oneself up to the possibility of allowing the opponent an opportunity to score more than the agent itself just gained, resulting in a net loss. As has been demonstrated by the rest of this thesis, the idea that one strategy can be applied at all times in the game of cribbage is laughable.

In the future, a partial agent could be trained in just the pegging phase alone using reinforcement learning. In a similar way to this thesis, multiple basic strategies such

as offensive or defensive play could be trained by rewarding each behavior and their combinations could in turn be learned through gameplay simulations. Alternatively, a more ground-up approach can be taken to train an unbiased agent by simply having each player learn from trial and error with certain card combinations. This has the disadvantage of taking more time to train to effectiveness. In either case, a more optimized pegging system can be made that would play better.

Performance Evaluation Even though a better pegging system can be made, key to this thesis project was the idea that performance can be predicted or at least anticipated with some amount of certainty. In this project, the performance of the pegging agent was tracked by card combination, recording the points scored and yielded during each occurrence. These records were used to anticipate how each combination of cards would play against an opponent.

Before the first round of training, however, there was relatively little pre-population of these records done: roughly one hundred thousand randomly dealt hands were played against each other. As a result of the small sample size, performance estimates were likely to be inaccurate. Furthermore, the small sample size did not demonstrably cover all of the possible hand combinations. Because of this, data would be missing for multiple combinations of cards for the decision process throughout the first round. This would mean that the card combinations' true desirability would be misrepresented during the calculation, allowing for the possibility of a philosophically unfair punishment or reward for what would amount to a guess by the pegging-based strategies. It also means that the information received by the agent during the weighting operation would not be static as other strategies were and thus could not be as reliably trusted for accuracy. While this was counteracted by using records from a first round training session in all subsequent training sessions, the questions remain whether this was enough and how much the variability in the data provided by the pegging strategies affected learning.

Implementation Decisions

Due to the desire to get a system up and running as quickly as possible, runtime speed took a back seat to development speed. As a result, Python became the language of choice. While this is a great multipurpose language, the overhead of using a higher-level, interpreted language proved to be a mistake not only directly because of its slower speed, but also because of resulting decisions that needed to be made to accommodate this reduced speed.

Database Dependency Because Python was the language of choice, the performance of making a decision for a single hand was sluggish: on a development machine, the decision for a single hand would take approximately ten seconds. As previously mentioned in Section 3, the decision made from that point was to create a database such that all calculations were done ahead of time and the statistics de-

sired would be available upon request. This succeeded in its desired goal, bringing the time required for a single decision to approximately one twentieth of a second. At this point the project was able to play two games in about a second's time on the same development machine.

Despite the speed improvements in a development environment, the entire training process had become I/O-bound rather than CPU bound. In and of itself, this was not a problem until an attempt was made to train multiple agents simultaneously. Because of concurrency issues which would have, if even possible, taken too long to fix, no more than one process could successfully access the SQLite database with any consistency. This would mean that only two agents could be trained on each database at any given time. Furthermore, since this database file was around twenty gigabytes in size and there was only limited local storage space available on each of the compute nodes in the Computer Science Department's Ukko2 high performance compute cluster, jobs had to be constructed carefully and spread across nodes to avoid accidentally disrupting other users' calculations. This was mitigated slightly by using the Physics Department's Kale cluster which allowed for significantly more manageable locally mounted drive space, albeit in a slightly slower hard disk with a RAID0 setup.

Were this project to be repeated, it would be recommended to use the Python code from this attempt as an outline for a rewrite in a slightly lower-level language. Using a language such as C++, which could be optimized more for run-time efficiency and memory management, would likely increase speed of computation enough to remove the necessity of a statistics database entirely. While the data gathered from previously played pegging rounds would still need to be stored and retrieved between training rounds, the size of this data would likely not exceed a few gigabytes and would fit easily in the program's RAM space. This is especially true since the strategies utilizing the median were not learned to be a useful metric for choosing cards, meaning that a list of scores gained and given are not needed, only the cumulative amount and times seen. Also, like weight checkpoints are in the current system, these stored results could be exported in a text or otherwise simple format for transfer between rounds.

With these changes made, the training program would be CPU-bound, which thankfully can be compensated for by adding or improving hardware. In contrast, the I/O-bound training program used in this thesis used only half the total processing power of a single CPU core as the limitation was the database file on a solid state drive.

5.2 Policy Iteration vs. Value Iteration

This thesis focused on an attempt to use policy iteration in order to develop a well-playing cribbage agent capable of weighing different strategies with the intention of using these learned weights to and to recover the strategy weights used to in turn improve the play of the author. In the primary objective of evolving a cribbage

agent to a good strategy, despite the setbacks of poor performance on a per-game level, the agent did remarkably well by removing irrelevant strategies such as `hand_max_med` and `pegging_max_med_gained` and emphasizing more useful strategies like `hand_max_avg` and `hand_max_min`.

One of the reasons for its poor per-game play is the lack of knowledge as to how a set of cards lends itself to being played with a certain strategy. The extent to which the cards themselves affect the action choice in the policy was greatly underestimated. However, to include the knowledge of which cards are known into the policy decision would have increased the dimensionality of the search space from the relatively small three-dimensional problem tackled to a problem of nine dimensions since all dealt cards would need to be considered. At this dimensionality, the search space would be far too massive and sparse to simulate an amount of games which would lead to any worthwhile learning.

An alternative to including the cards in the policy would be to not learn a policy at all. Instead, the state-value function could be learned, as done with TD-Gammon [Tes95], AlphaGo [SHM+16, SSS+17], and O'Connor's senior project [O'C00]. By learning state-value function instead of a policy, an agent could determine which set of cards would likely place the agent in the most optimal resulting position, developing its own policy. This value iteration should allow for a more adaptive agent and thus more optimal gameplay, leading to an agent that would win more consistently.

5.3 Shortcomings of the Model

In addition to the shortcomings of the implementation decisions made or necessitated by other decisions, the nature of the model itself should be considered and evaluated. Several assumptions were made to limit the scope and dimensionality of the problem, but it is likely the case that these limitations also affected the potential learning ability of the agent.

Linearity

One of the model's shortcomings that must be addressed is the linearity of the decision making apparatus. As described in Section 3.2, the mechanism for deciding which combination of cards to choose is—at its core—a simple linear operation. The weights vector $w_{m,o,d}$ is multiplied by a desirability matrix S to produce a probability vector p, from which the maximum value is chosen when the policy is strictly followed. Although a human player would indeed evaluate which combinations are best to play with a set of strategies with varying importance over the course of the game, the nature of this relationship in the player's mind is unlikely to be a simple linear function. Instead, a human player would also consider how two or more strategies would interact with each other at different points in the game.

Strategies

Another shortcoming of the setup to the model to the problem of this thesis is the limitation of the strategies chosen. Since each strategy's contribution to the S matrix can be thought of as a feature, this means that the final model is no more than a simple linear model of eight features used to predict the outcome of a card-game. Furthermore, the features selected, although sensibly selected by a fairly experienced human player, may themselves not be ideally suited to the task at hand.

As a result of these previous shortcomings, a better solution for the future would be to use an architecture that allows for the learning of nonlinearities and automatic feature discovery and learning, such as a neural network as per [SHM+16] and [Tes95].

5.4 Usefulness of Results

Despite the final learned agent's shortcomings in ability to consistently win a single game, the deliverables of this project can still be applied to expand the current knowledge of cribbage as well as serve as a guidance story. As they were intended from the outset, the generated strategy graphs can serve as a set of guidelines as to how a player "should" be playing a certain hand. Although a human player may not be able to calculate the statistics which the agent used as accurately as a computer, a fair amount of experience and intuition can be gained through repeated play which should approximate the expected and guaranteed returns well.

The agent's successes in the macro scale can also be of use to those applications which also operate on the scale of thousands to millions of games. Mainly, the developed agent could be used for the elementary first stages of training of a value-function based agent. Rather than learning from the self-play with no existing knowledge, a policy-following agent can be played against instead. Playing an agent of greater difficulty would allow the value function optimizing agent to more quickly potentially converge to an optimum.

6 Conclusion

This thesis presented an attempt to learn how to play cribbage according to a set number of predefined strategies through reinforcement learning. Section 2 provided an overview of related work in reinforcement learning of games as well as attempts to apply machine learning specifically to the game of cribbage. The methods for how the agents would be trained was presented in Section 3. Section 4 presented the results of the training and experiments to determine ways in which better results may be obtained. Finally, a discussion of potential applications of this thesis were presented in Section 5 as well as a summation of ways in which someone looking to

continue this effort to apply reinforcement learning to the domain of cribbage can proceed.

References

ACCa URL `http://www.cribbage.org/rules`.

ACCb URL `http://cribbage.org/rules/ACCRuleBook2016.pdf`.

dai Daily cribbage hand. URL `dailycribbagehand.org`.

Kho10 Khomskii, Y., Infinite games. Technical Report, Technical report, University of Sofia Bulgaria, Summer Course (July 2010), 2010.

KS02 Kendall, G. and Shaw, S., Investigation of an adaptive cribbage player. *International Conference on Computers and Games*. Springer, 2002, pages 29–41.

Mar00 Martin, P. L., *Optimal Expected Values for Cribbage Hands*. Ph.D. thesis, Harvey Mudd College, 2000.

O'C00 O'Connor, R., Temporal difference reinforcement learning applied to cribbage. Technical Report, University of California, Berkeley, 2000. URL `http://r6.ca/cs486/`.

PB97 Pollack, J. B. and Blair, A. D., Why did td-gammon work? *Advances in Neural Information Processing Systems*, 1997, pages 10–16.

Sam59 Samuel, A. L., Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3,3(1959), pages 210–229.

SB17 Sutton, R. S. and Barto, A. G., Reinforcement learning: An introduction. Incomplete draft used available from author's site at `http://incompleteideas.net/book/the-book-2nd.html`, 2017.

Sha53 Shapley, L. S., Stochastic games. *Proceedings of the National Academy of Sciences*, 39,10(1953), pages 1095–1100. URL `http://www.pnas.org/content/39/10/1095`.

SHM+16 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., Mastering the game of go with deep neural networks and tree search. *Nature*, 529,7587(2016), pages 484–489.

SSS+17 Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al., Mastering the game of go without human knowledge. *Nature*, 550,7676(2017), page 354.

Tes95 Tesauro, G., Temporal difference learning and td-gammon. *Communications of the ACM*, 38,3(1995), pages 58–68.