Date of acceptance Grade

Instructor

Policy Improvement in Cribbage through Reinforcement Learning

Sean R. Lang

Helsinki April 9, 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 through Reinforcement Learning

Oppiaine — Läroämne — Subject

Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	April 9, 2018	44 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

Contents

1 Introduction

I just want to see if unicode characters will appear. Jag måste se om unicode-characterer funger. minä haluan katsoa kirjain öööö.

Generic introductory stuff giving a sentence or probably a paragraph about each of the sections covered.

Test stuff: don't commit. This is some learnèd agent.

For suits: ♣, ♢, ♡, ♠

5♣, 6♡7♢ 8♠

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 as score of 121 points or more is declared the winner. Players will usually keep track of their points by *pegging* them on a characteristic board. Because of its atypical win condition, different strategies hold differing levels of importance throughout the game.

1.1.1 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, following the scoring rules found in Appendix A could play a complete game, albeit likely not well.

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 toss into the crib.

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

3. A random card is cut from the remaining cards of the deck 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, keeping track of the total value 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. In the latter case, the player last to play a card will be awarded points before the count is reset and play continues. If any of the combination of cards mentioned in Appendix A is seen in the immediately preceding cards, the amount of points earned is immediately pegged on the board for the appropriate player. In cribbage terms, this is called *the play*.

5. After all cards have been played, the pone then counts his or her hand using the randomly cut card as a 5^{th} 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 5^{th} 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.

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. 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. 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 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. The best way to learn, in humans and computers, is by doing. Barring savants such as Bobby Fischer, humans will learn how to play chess better by playing games against varying opponents. 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. Classical teaching methodologies would not be highly applicable in this situation because there is no single correct strategy in chess that can be taught.

1.2.1 Rewards and the Environment

The three most important, constantly interacting components in a reinforcement learning scenario are the environment, the agent, and 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.

1.2.1.1 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 state 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. In golf, the action is again continuous and may be which club to use in which direction and with how much power.

1.2.1.2 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:

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

1.2.1.3 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.

1.2.2 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.

1.2.2.1 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'

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

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

1.2.2.3 Policy Improvement and Iteration

1.2.2.3.1 Generalized Policy Iteration

1.2.2.4 Value Improvement

1.2.2.5 Monte Carlo Methods

Algorithm 1 Policy Evaluation

Require: π
Require: 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 $V \approx v_\pi$

2 Literature Review

Overview of the current literature surrounding this topic.

- Research done in cribbage
- Research done in related imperfect information games (e.g. poker)
- Overview on expert witness machine learning
- Any other topic that ends up getting used (Bayesian logic, statistics?)

2.1 Prior Cribbage Research

3 Data and Methods

Walk-through of how I went about researching the topic. (Maybe I should keep a diary or log or something so this isn't half made up at the end.) When all is said and done, include the final "output" graph in some easily viewable format (121-by-121 table of miniature bar graphs, RGB combination in each cell?).

- Creation of framework
- Creation of "expert advisor"
- Training method for listener/combiner
- Initializations for training

3.1 Methods

3.1.1 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. These included:

- `hand_max_min`: The hand(s) which has the maximum 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. Similar to `hand_max_avg`, this strategy is useful when trying to maximize one's hand's expected score.
- `hand_max_poss`: The hand(s) with the maximum possible score will be the most highly desired. This strategy can be thought of as Hail Mary choice for the player trying to score as many points as possible, not taking into account its unlikelihood.
- `crib_min_avg`: The hand(s) with the minimum average amount of points scored in the crib 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. This strategy is similar to `pegging_max_avg_gained` in its use, but uses a slightly different internal measure for ranking.

- `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 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 and "tossed" to the crib. For each of these 15 possible there are a further $(52 - 6) = 46$ 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 $(46 - 2) = 44$ 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 because whether or not the card is in the crib or not can affect the score.[2] 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,

[2]For example, as per the right-jack 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.

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 each hand.

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. Thankfully, most of the framework for this rewrite had actually been developed previously as part of a mental exercise, so the loss in time for the rewrite was minimal and well worth the time saved in execution. Furthermore, access to a high-performance server allowed for futher paralellization 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 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 an integer.[3] 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

[3]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.

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 lower-valued card that reached that score. 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.

3.1.2 Weighting

For the purposes of this project, how well certain combinations of cards are played with different strategies is not 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 $PlayerScore \in [0, 120]$, $OpponentScore \in [0, 120]$, and $Dealer? \in \{0, 1\}$. 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.

0	6
1	3
2	9

0	a	6
1		3
2	c	9

0	a	6
1	b	3
2	c	9

Figure 1: A visual depiction of the weighting operation.

3.1.3 Training

After a complete game had 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 by those whose values in S were the highest in its column. This subset of strategies, carefully chosen to only include the outliers or top percentage of contributors to avoid resulting uniform 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:

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

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 $0 < d < 1$.

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 yielding control to other strategies.

The reinforcement training framework operated very simply. Given an initialization weights file for each agent, a new agent would be created with those stored weights. 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 weight-state to a checkpoint file. This allowed not only for tracking of weight adjustments over time, but also allowed for the ability to more easily recover potential issues and for more promising states to be explored further by using a checkpoint as an initialization state on a subsequent training run. This checkpointing system allowed agents to be mixed and matched together to allow different combinations to learn from one another.

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 chosen by using the produced p vector as a probability distribution. At any given point in a training game, however, there was a chance of randomly selecting which cards are played, with the percent chance e of random choice related to the variance of the weights according to the formula:

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

where k is a constant around $\frac{1}{3}$ 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 be exploratory in nature. Furthermore, this acted as yet another barrier to the possibility of falling into the pitfall of stationary uniform weights.

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 101, 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. After the match had completed, the winner would advance to the next round, training against another winner from the previous round as the process is repeated until one agent is declared the ultimate winner.

4 Findings

What are the final results of the “experiment.”

- Where did each initialization family end up going?
- Were they different or did the agent learn a “single” strategy in general?
- All this and more will be answered . . . after the break!

4.1 Round 1

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

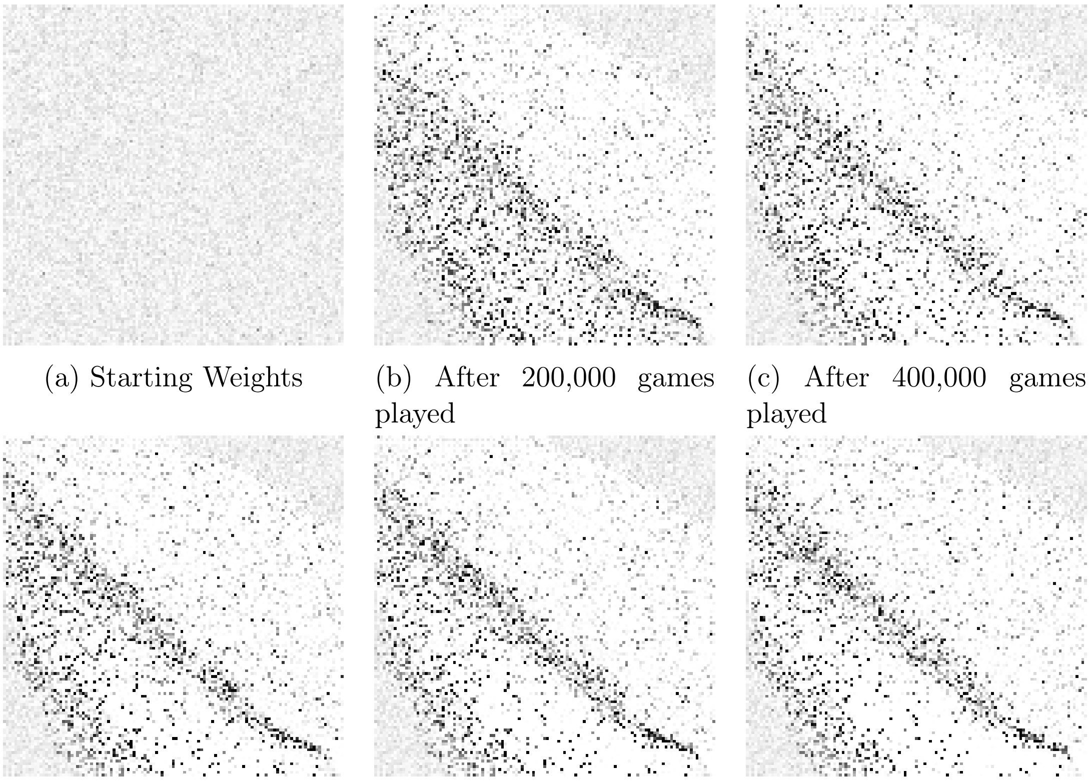
4.1.1 Learning Process

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 is the stark contrast in colors in the majority of the image. The other is 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.



(a) Starting Weights

(b) After 200,000 games played

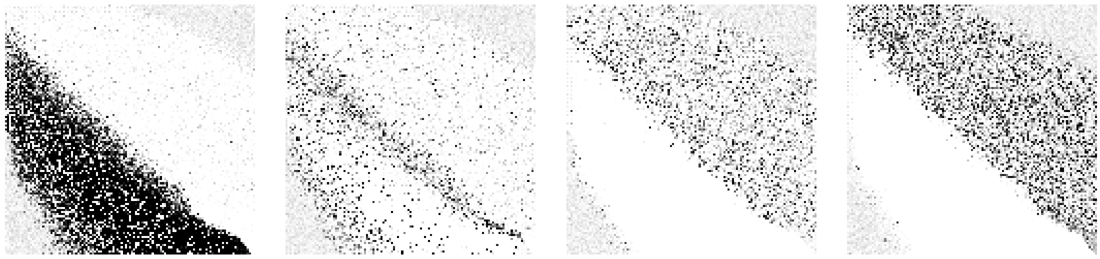
(c) After 400,000 games played

(d) After 600,000 games played

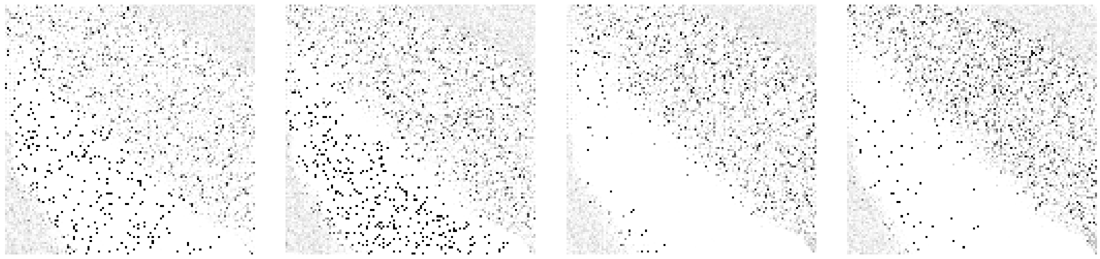
(e) After 800,000 games played

(f) Final Weights

Figure 2: Training weights representation for Agent 0's `hand_max_avg` strategy when the 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.

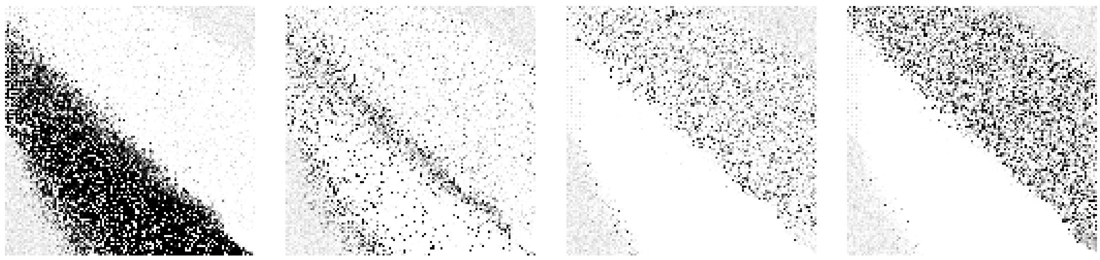


(a) hand_max_min (b) hand_max_avg (c) hand_max_med (d) hand_max_poss

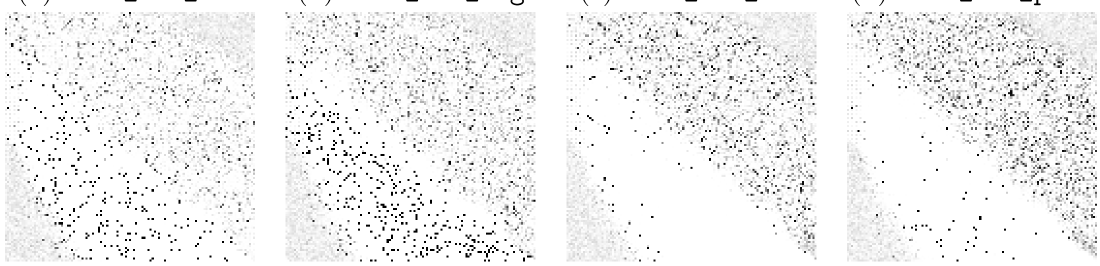


(e) crib_min_avg (f) pegging_max_avg_gained (g) pegging_max_med_gained (h) pegging_min_avg_given

Figure 3: All final strategy strengths for Agent 0 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 Agent 0 when playing as the pone after training for one million games during Round 1.

4.1.2 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 4, 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 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 alignment of the results between `crib_min_avg` and more reasonable `hand_max_min` or `hand_max_avg`.

Of further interest is how little the agent knows 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 to those of the dealer's, 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.

Game	Random	Agent 1
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
Agent 1	9	—	4

Table 1: Results of a nine-game tournament played between a randomly-weighted agent and Agent 1 after learning for one million games.

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.

4.1.3 Performance

While it is aggravating that the agent learns to over-trust a single strategy, it is simultaneously reassuring that general trends in play are detected. However, perhaps the only metric which matters from a learning perspective is the 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 1, 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

Epochs	**Score**	**Spread**	**Wins**
0	126	+523	59
1MM	86	—	41

Epochs	**Score**	**Spread**	**Wins**
100K	122	+200	55
1MM	104	—	45

Epochs	**Score**	**Spread**	**Wins**
200K	108	-27	47
1MM	118	—	53

Epochs	**Score**	**Spread**	**Wins**
300K	105	-180	48
1MM	116	—	52

Epochs	**Score**	**Spread**	**Wins**
400K	87	-510	41
1MM	133	—	59

Epochs	**Score**	**Spread**	**Wins**
500K	97	-386	43
1MM	129	—	57

Epochs	**Score**	**Spread**	**Wins**
600K	117	370	52
1MM	98	—	48

Epochs	**Score**	**Spread**	**Wins**
700K	107	-174	48
1MM	115	—	52

Epochs	**Score**	**Spread**	**Wins**
800K	124	239	56
1MM	96	—	44

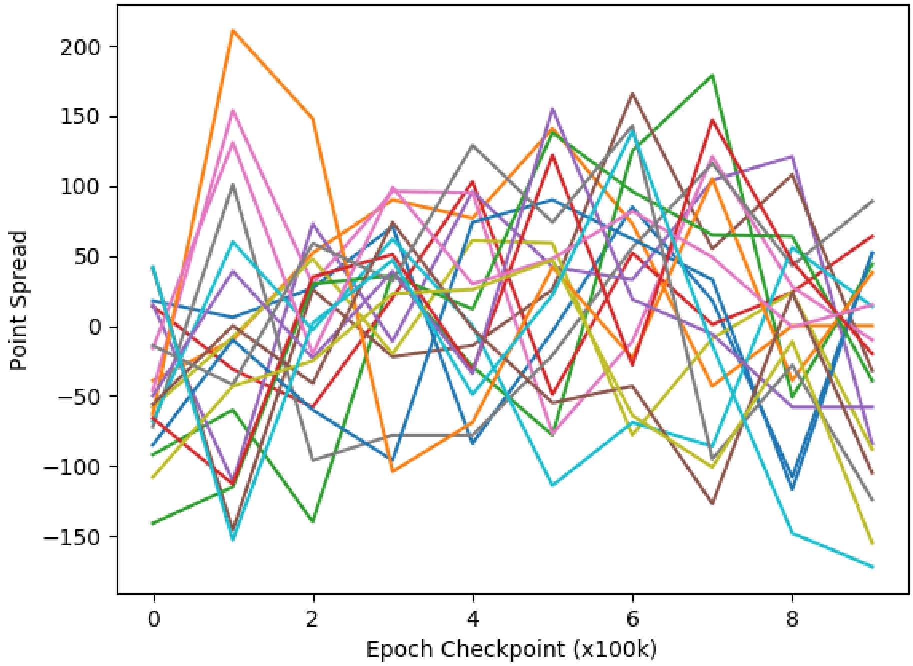
Epochs	**Score**	**Spread**	**Wins**
900K	111	-82	51
1MM	111	—	49

Table 2: Results of multiple 100-game tournaments played between agents using various epoch checkpoints from the training and the final agent from the round. The weights used here are from Agent 1.

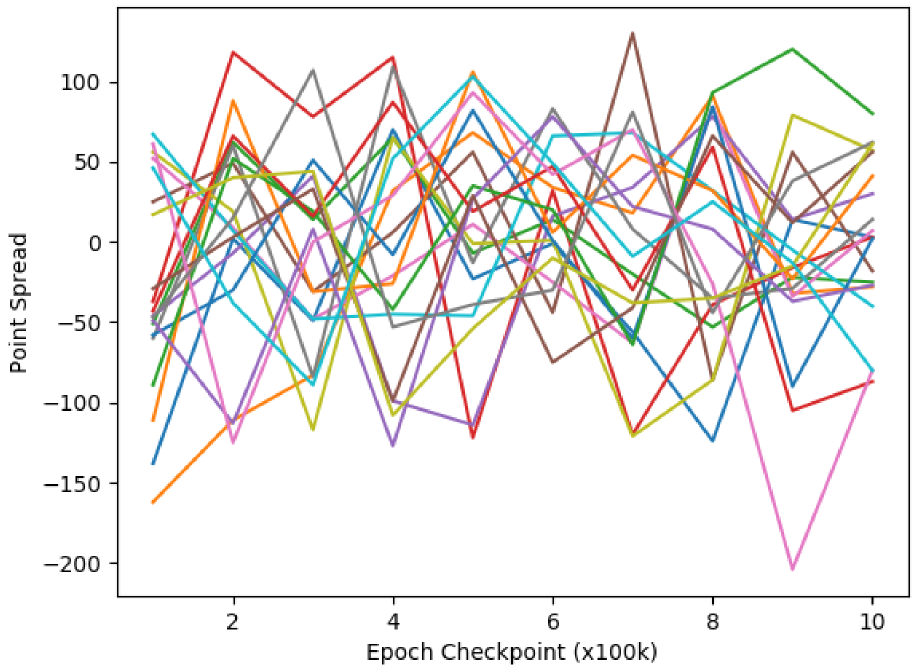
scores can be found in Table 2. The point spreads from twenty separate 100-game tournaments are depicted in Figure 5a. While point spreads are from total points pegged to the board, not gained from wins and therefore not a direct indicator of performance in tournament play, it is useful for determining 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 depicted in Table **??**.

4.1.4 Applications for Round 2

As a result of the 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 tweak 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. Because of the stipulation on the variance previously explained in the Methods section, the most strongly biased weight locations would only allow for exploration even more rarely



(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.

than before. Although only producing a decrease from about 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 diagnosed with 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.

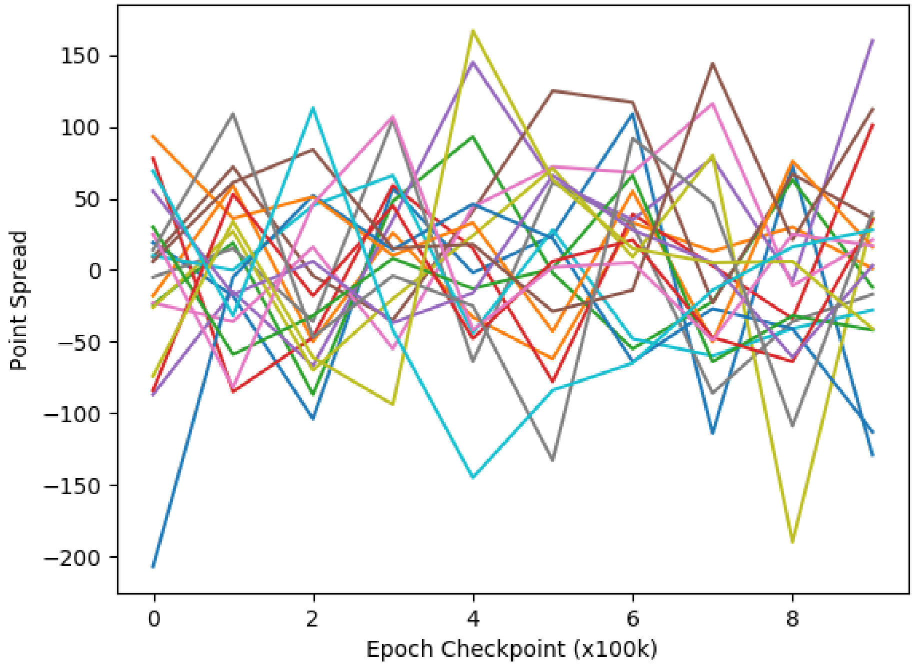
4.2 Round 2

4.2.1 Performance

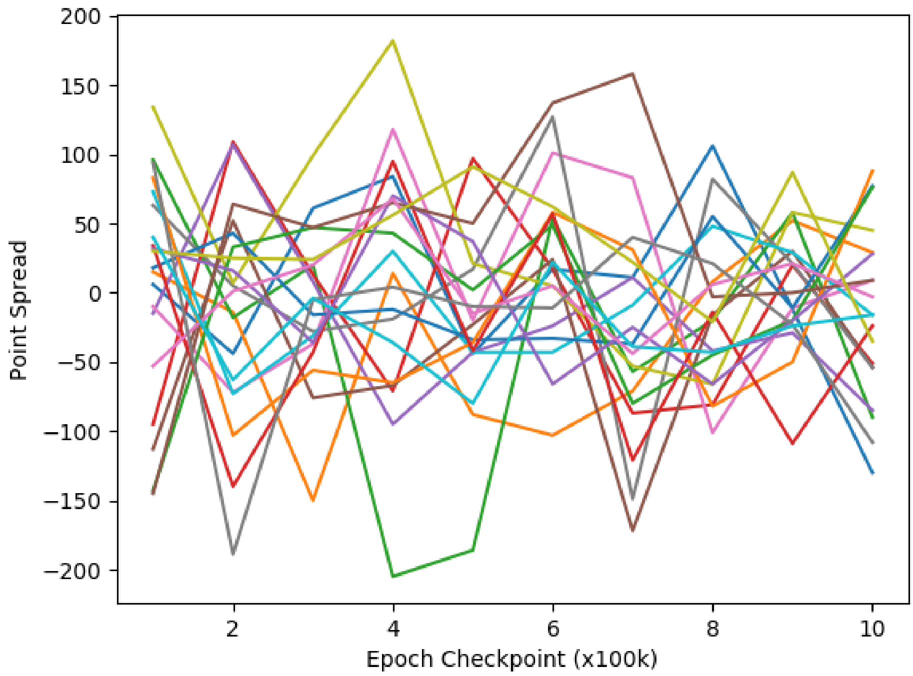
4.2.2 Learning Process and Results

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 smaller levels self-certainty, the patterns of which policy to mainly follow at which times are quick to form. This is useful because it allows for the possibility of running future experiments with fewer training epochs and thus 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

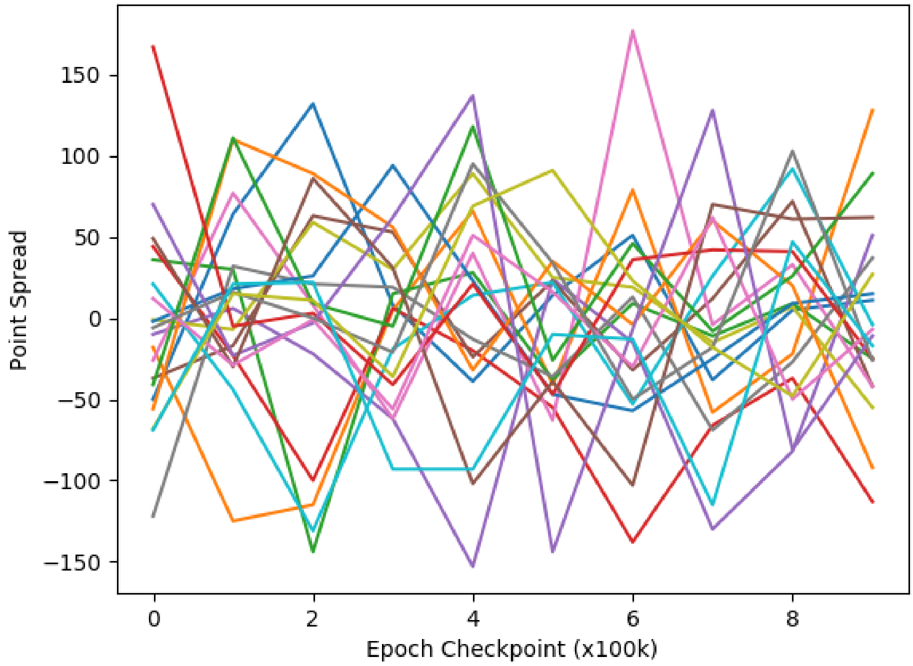


(a) An agent plays against previous iterations of itself.

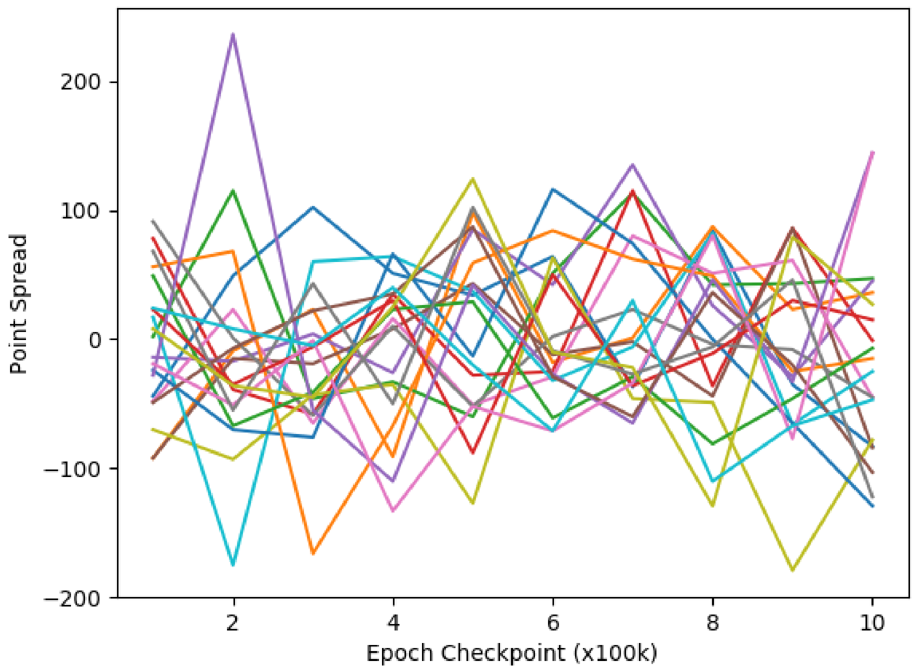


(b) A random agent plays against later, more learned agents.

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 in the Loser's Bracket of Round 2.

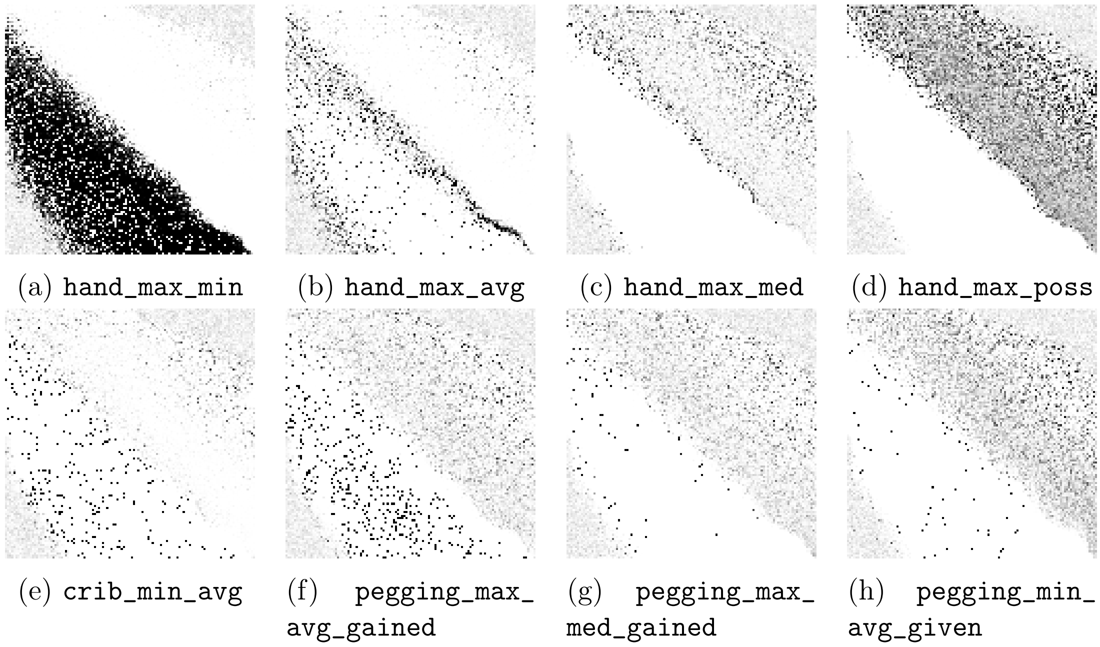
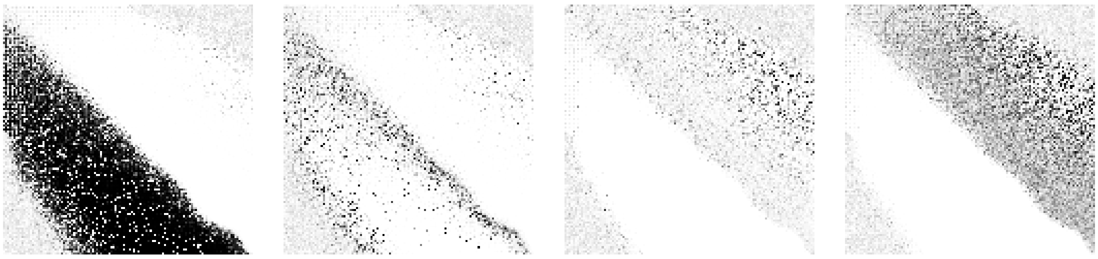


Figure 8: All final strategy strengths for an agent in the “winner’s” bracket when playing as the dealer after training for one million games during Round 2.

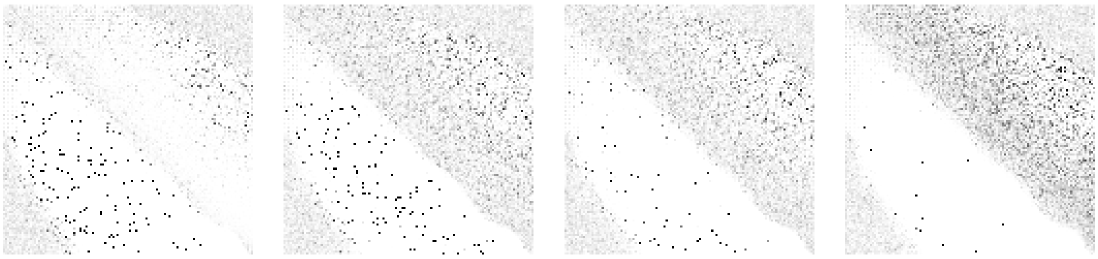
extending further back along the diagonal to earlier game positions, albeit with smaller amplitude. While not being of much use to the agent directly, this is a useful observation from a cribbage player’s perspective. Since it is possible to infer the state-value function, the implications of this wave 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 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 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. While two million games is speculated to be enough to counteract this adjustment, there is no evidence to either support or disprove this conclusion.

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 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 gradiated. 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



(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 loser's bracket when playing as the dealer after training for one million games during Round 2.

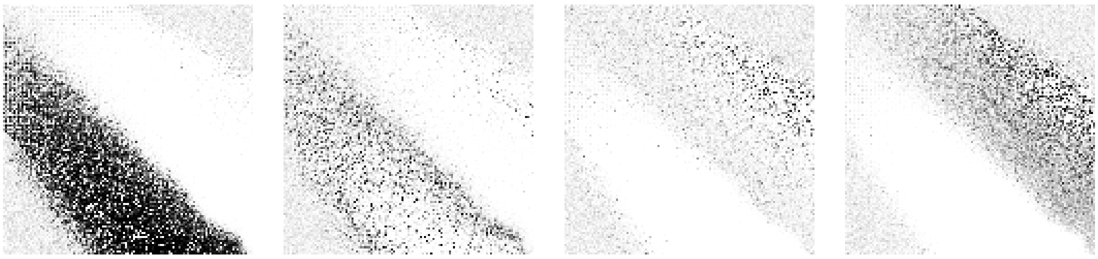
spaces in which strategies such as `crib_min_avg`, `pegging_max_avg_gained`, and `pegging_min_avg_given` have been strengthened within the `hand_max_min` *block*. This is a good indicator that, although present, there is less of a bias towards those strategies which are initially winners.

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 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 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.

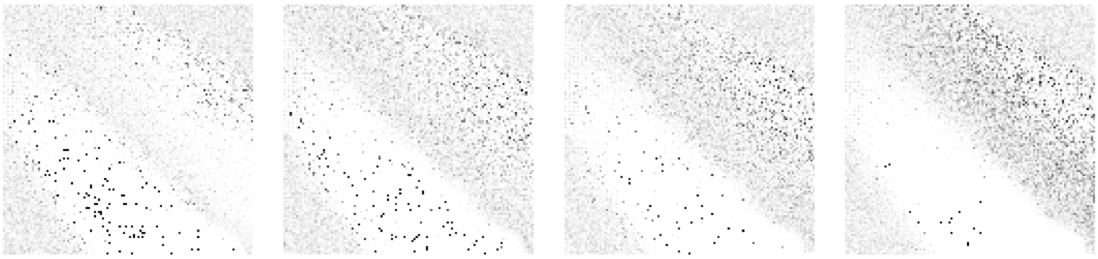
4.2.3 Something TODO

At this point, 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 as to why such poor results are occurring during the review tournaments.

4.2.3.1 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



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



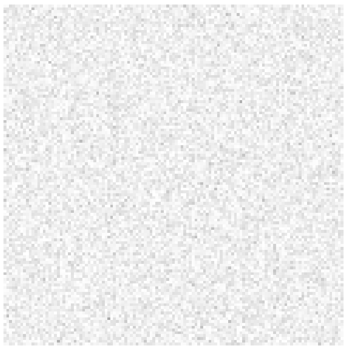
(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 10: All final strategy strengths for a learning agent after playing a completely random agent when playing as the dealer after training for 350,000 games during Round 2.

that the network 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 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 Figure **??**, this is not the case as performance is not apparently linked to training in any way. Therefore, it can be safely concluded that overfitting is not the reason for the poor resulting performance.

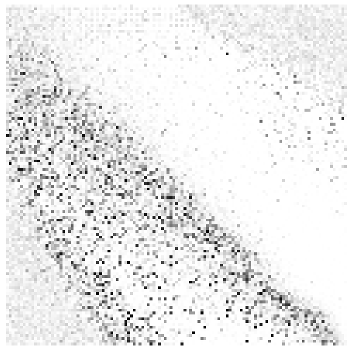
4.2.3.2 Strictness of Policy Following If overfitting is not occurring, then the next question to ask is if there are implementation differences which have led to these discrepancies. The training and testing phases use slightly different implementations of card choosing algorithms. In training, an exploration rate is used to give the agent a chance of choosing cards at uniform random whereas the testing phase does not. This random percentage was added back in for testing and the results can be seen in Figure **??**. As can be seen, the point spreads still display the same randomness and lack of pattern, indicating that the random exploration is not crucial to performance.

In addition to the chance of purely random exploration, the agent also uses the produced *desirability* vector p as a probability distribution to select a card combination with a probability. The use of this probability distribution is the same between both training and testing phases. However, it can be speculated that since the introduc-

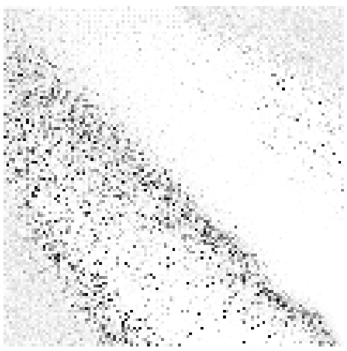


(a) Starting Weights

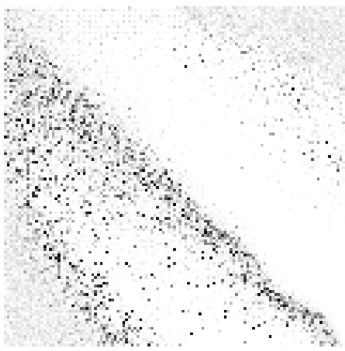
(b) After 200,000 games played



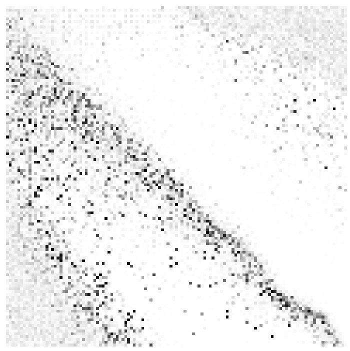
(c) After 400,000 games played



(d) After 600,000 games played

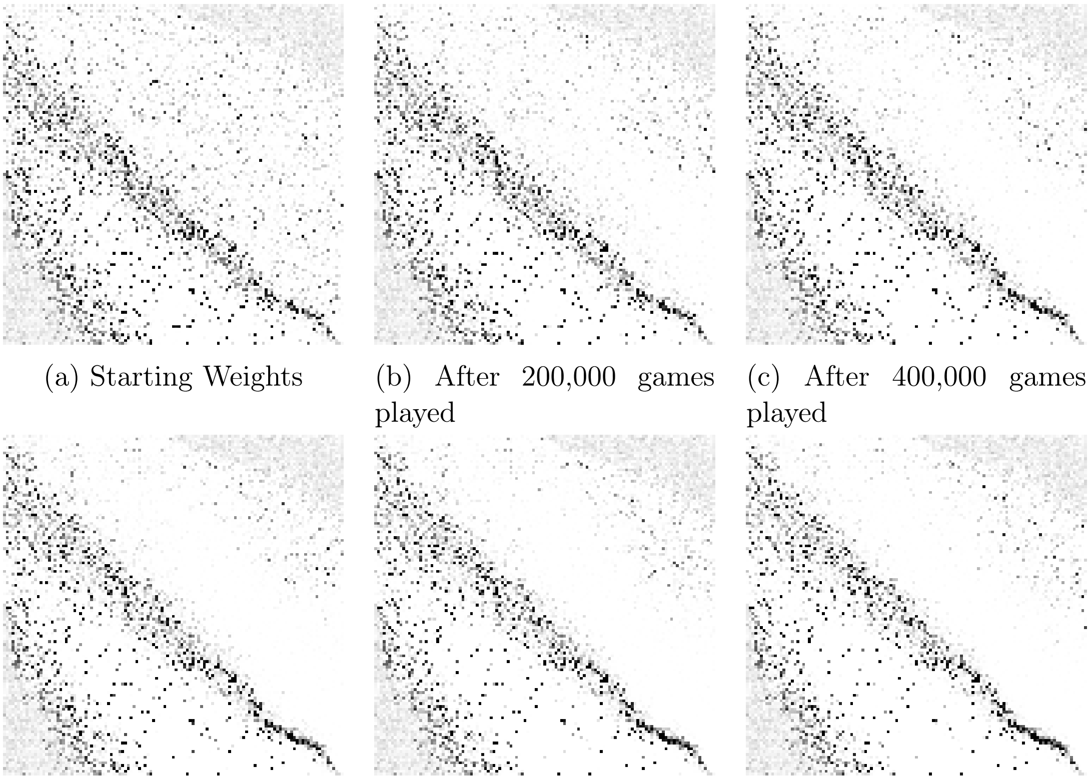


(e) After 800,000 games played



(f) Final Weights

Figure 11: 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.



(a) Starting Weights

(b) After 200,000 games played

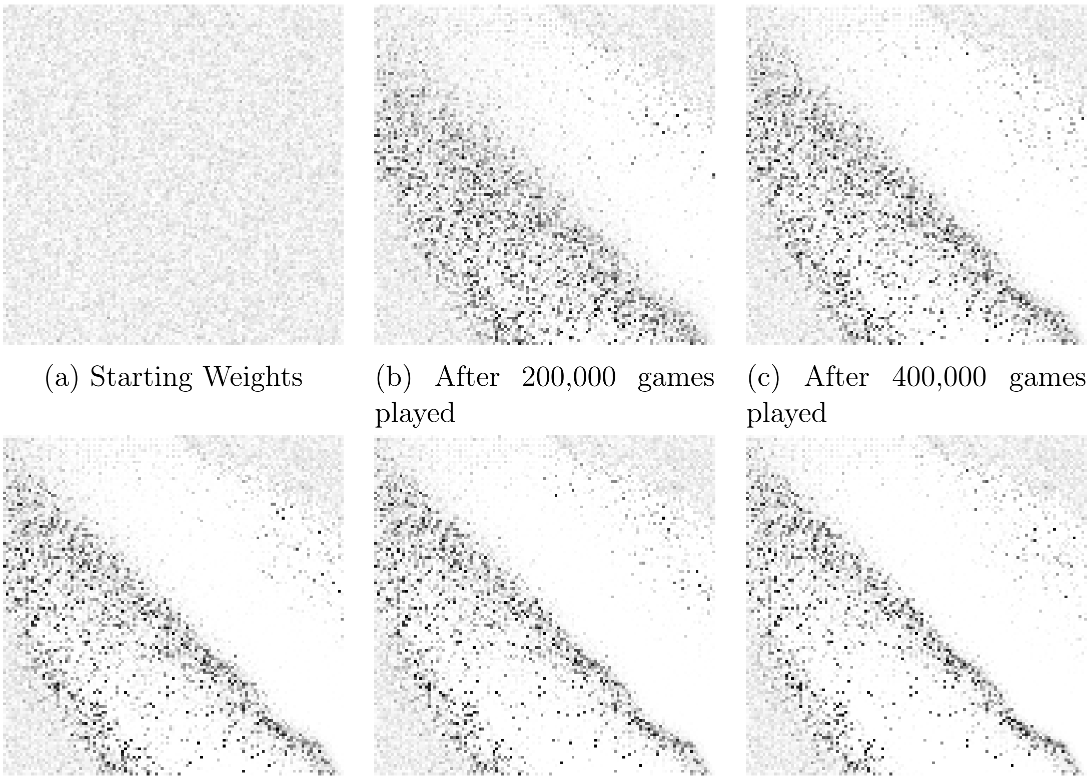
(c) After 400,000 games played

(d) After 600,000 games played

(e) After 800,000 games played

(f) Final Weights

Figure 12: Training weights representation for a 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 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. Training weights representation for an agents `hand_max_avg` strategy over the course of one million games in which the opponent was always played using randomly allocatd weights.

tion of randomness did not worsen the results of testing, it may be the case that the policy needs to be followed more strictly. Since a weights vector is multiplied by a desirability matrix, the resulting probability distribution vector may not always be as varying as the weights vector itself. Therefore, while a clear winner may be discernible as the maximum in the probability distribution, the actual probability of selecting that value could be very low in comparison to selecting any of the remainder. As a result, a strict policy following agent was instead tested in which the maximum was selected from the probability distribution without exploration. The results, found in Figure **??**, indicate that this too was not an issue since as much variance in play is still present as when using a more exploratory implementation. Therefore, even though implementation differences exist between the training and testing phases, they are not the reasons for the poor testing performance.

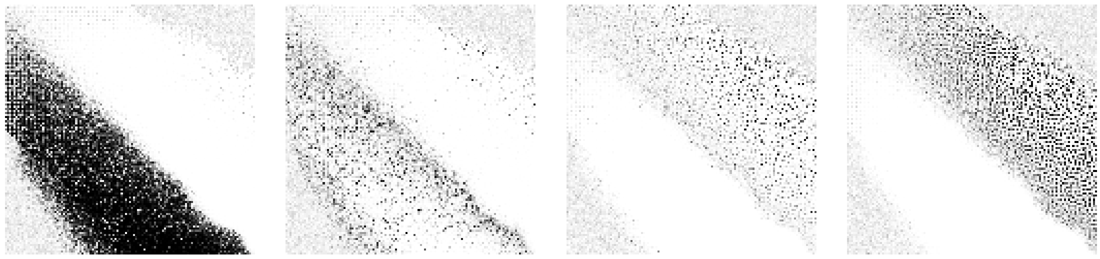
4.2.3.3 Scale of Play The final conjecture as to the reason for which the agent learns a policy but following that policy does not yield positive results in performance is the issue of scale. The agent is trained on a million games, but tested on only a handful. Since the cards being dealt are not taken into account by the policy, the decisions made will likely be inaccurate on the scale of a single game, but not for thousands. As can be seen in Figure **??**, the jump to one thousand games begin to show signs of making a curve. Although, there is still too much variance to make this observation without squinting on the part of the viewer. Figure **??** shows a single agent's tournament against its own previous iterations for a set of ten thousand games. On this scale, the policy begins to be of use and the agent consistently wins against its previous iterations. There is a tighter points spread between the agent and its later checkpoints indicating that the agent is indeed learning and providing further evidence that overfitting is not occurring.

4.3 Experiments

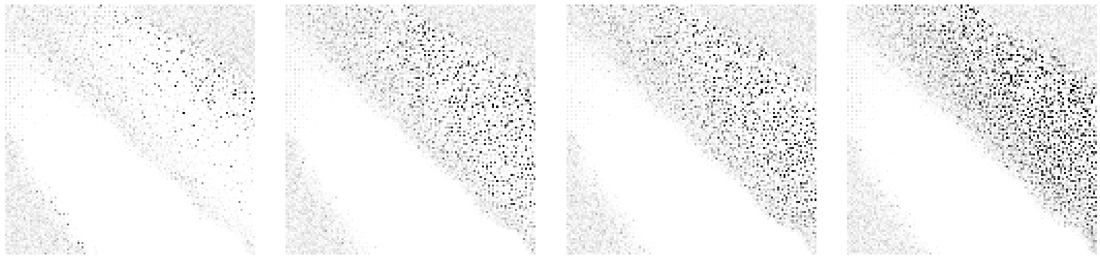
As the results of Round 2 emulated those of Round 1, additional modifications were applied to the learning process. These methods focused on directly affecting the weights applied to each decision made, both in learning and at choosing time. The intended goal of these modifications were to provide a more nuanced policy capable of adapting to an agent beside that which it was trained against.

4.3.1 Round-Robin Play

Since the agents were deemed to be learning to outplay each other, but not necessarily how to optimally play the game, training against a single opponent for a million games was determined to be an issue. To determine how tailoring play to a single opponent affected the final outcome weights, a round-robin structure was implemented in which multiple agents would swap out opponents occasionally. Given a pool of potential agents, every N games, a random pair of agents are selected from



(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 14: 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.

the pool and played against one another. By varying opponents every so often, the intention of the round-robin structure should eliminate an agent's dependence on another's strategy for performance.

4.3.2 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.

4.3.2.1 Results Perhaps unfortunately, 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 affected. As seen in Figure ??, the `hand_max_avg` strategy graph has become more sparse in the winning triangle 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 hope 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.

4.3.3 Regularization

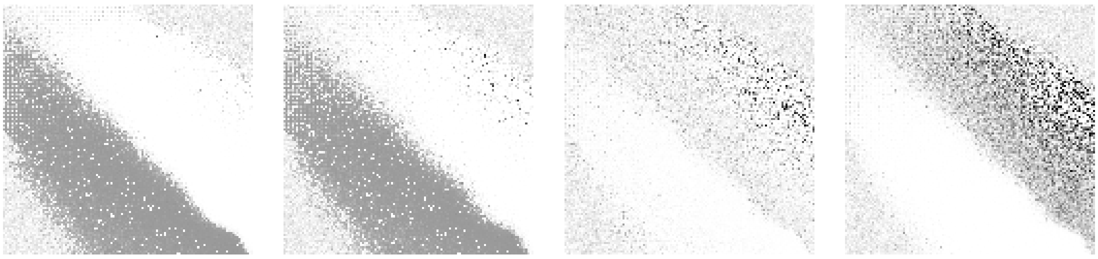
As an attempt to prevent a single strategy's weight from being so strongly preferred that a second strategy could not hope to possibly gain ground, a hard limit was placed on the pre-normalized update value. Expressed mathematically,

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

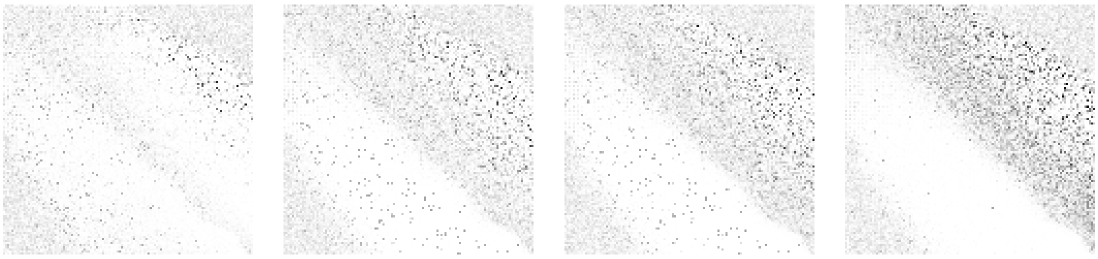
where K is some constant value throughout the training. While the value of $w_{m,o,d}[i]$ could exceed K after re-normalization for a particularly strongly weighted strategy, the value could be seen converging to K within a handful of iterations. The desired intention of this regularization was to allow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

4.3.3.1 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 ??, the `hand_max_min` and `hand_max_avg` strategies are now 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 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

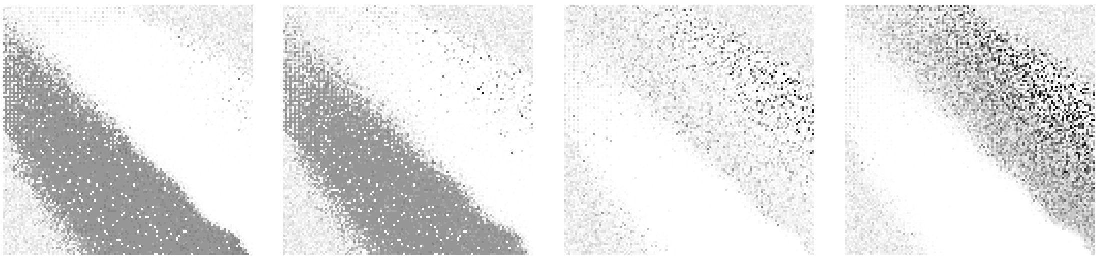


(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

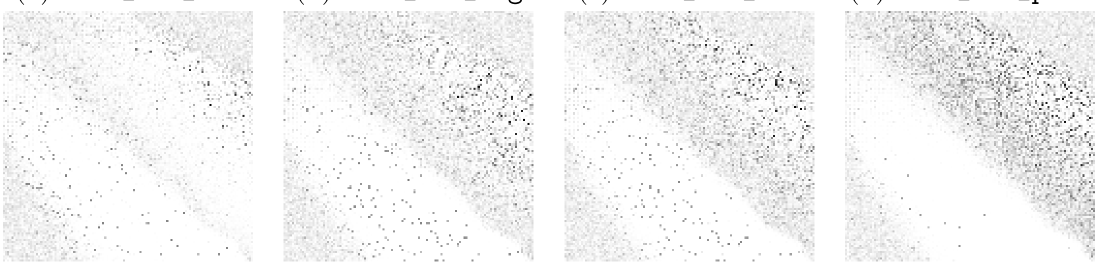


(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 15: 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.

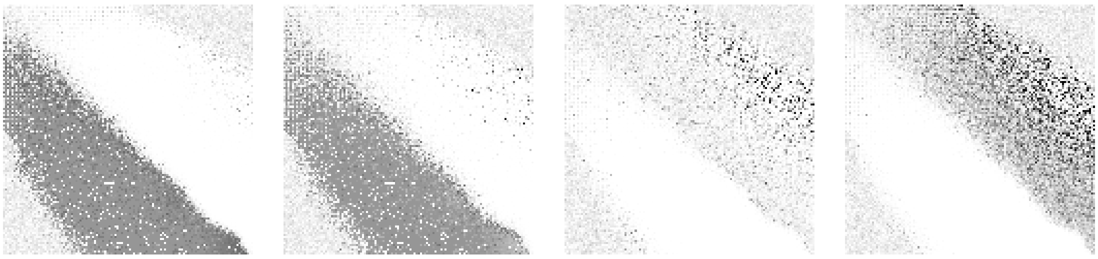


(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

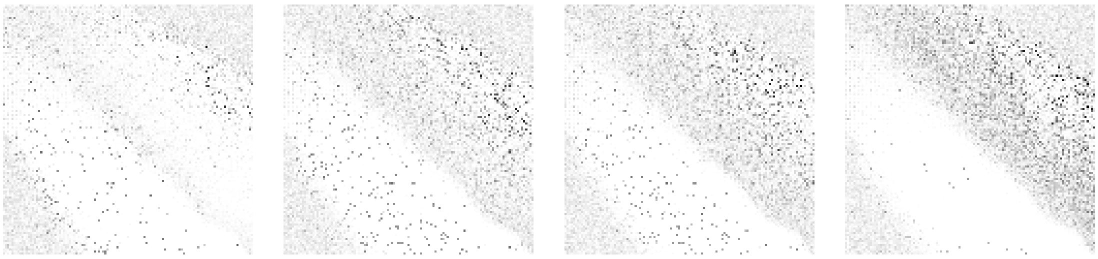


(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 16: 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.

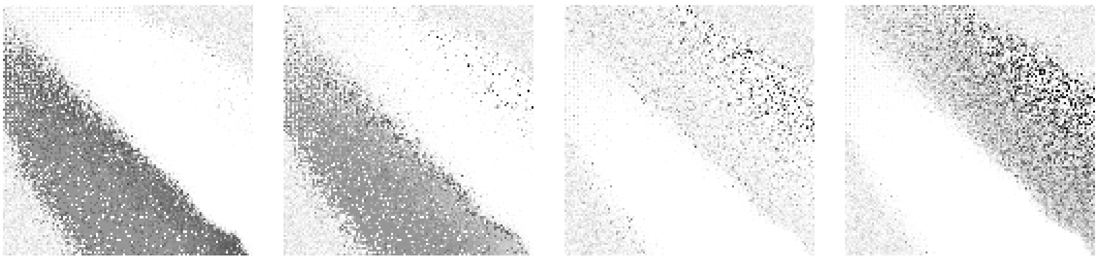


(a) `hand_max_min` (b) `hand_max_avg` (c) `hand_max_med` (d) `hand_max_poss`

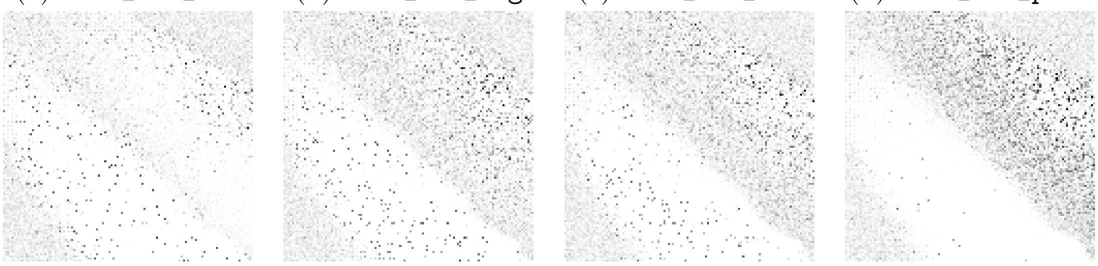


(e) `crib_min_avg` (f) `pegging_max_avg_gained` (g) `pegging_max_med_gained` (h) `pegging_min_avg_given`

Figure 17: 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 18: 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.

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

5 Discussion

This is the difficult part. What does this part mean? And how does it differ from what I've already covered above in the Findings section.

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.

5.1.1 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.

5.1.1.1 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.

5.1.1.2 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 cards 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.

5.1.2 Implementation Decisions

Because of the author's previous experience in only small-scale uses for cribbage software, experience with optimizing software for large-scale computation tasks was lacking. This inexperience, in combination with the desire to develop a prototype quickly in a familiar environment, led to the decision to write the majority of this thesis in Python. This, to put it bluntly, was a mistake because of the unforeseen complications and side effects that that decision led to.

5.1.2.1 Database Dependency Because Python was the language of choice, the performance of making a decision for a single hand was abysmal: on a development machine, the decision for a single hand would take approximately ten seconds, As previously mentioned in the Methods section, the decision made from that point was to create a database such that all calculations were done ahead of time and the statistics desired 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, now 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 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 high performance compute cluster, jobs had to be constructed carefully and spread across nodes to avoid accidentally disrupting other users' calculations.

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 countered 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 Value Improvement vs. Policy Improvement

This thesis focused on an attempt to use policy improvement to create an optimal cribbage-playing agent 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. As has been discussed, the agent learns a very good set of guidelines as to how a position should be played in the majority of situations.

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 a 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. 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 improvement would 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 features 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 applied to the potential learning ability of the agent.

5.3.1 Linearity

One of the model's shortcomings that must be addressed is the linearity of the decision making apparatus. As described in Section **??**, 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 distribution 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.

5.3.2 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 [**?**] and [**?**].

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 a first round or two 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 converge to an optimum. Since the results of this thesis were achieved without imparting any particular pre-existing domain knowledge to the initial policy, this faster convergence to optimal would not be unfairly biased towards any form of human play, thus lowering training time without introducing suboptimalities.

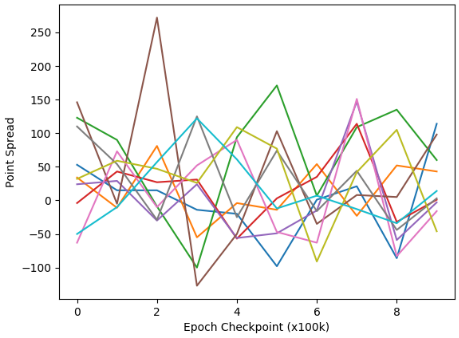
A regulation cribbage match between two human players consists of nine games. As can be seen in Figure **??**, in a series of 100-game matches between a fully trained agent and its previous checkpoints, no pattern is discernible in total point spreads between the two playing agents. This demonstrates 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 **??** 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.

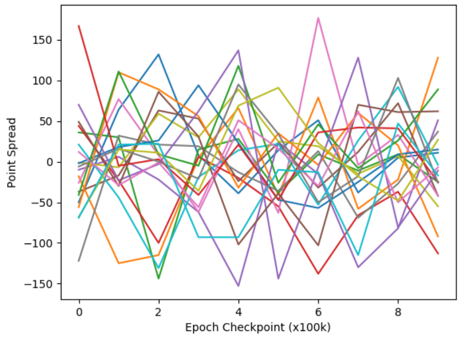
Although fewer matches are played to accommodate the increased time needed to play the increased number of games, the same pattern present in the thousand-game scale is visible in the ten thousand-game matches. With the exception of the purely untrained agent, the least learnèd agents perform the poorest against the final learnèd agent. Play are approximately evenly matched when between agents when the opposing agent has trained 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 19d). 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. This conclusion is supported by the observation that the agents with limited training are steadfastly outplayed.

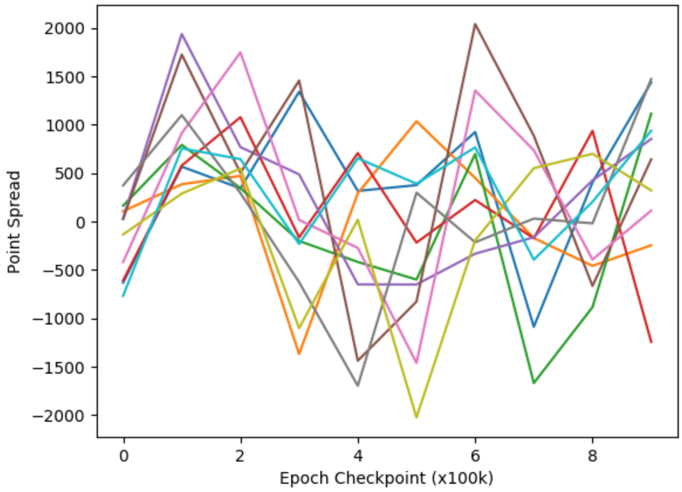
However, since there is a sharp dip after 300,000 training epochs before increasing again, the results are difficult to interpret. The final learnèd agent is more evenly matched in performance with an agent trained for 400,000 games than it is against a more recently trained agent. The reintroduction of an increasing curve is an



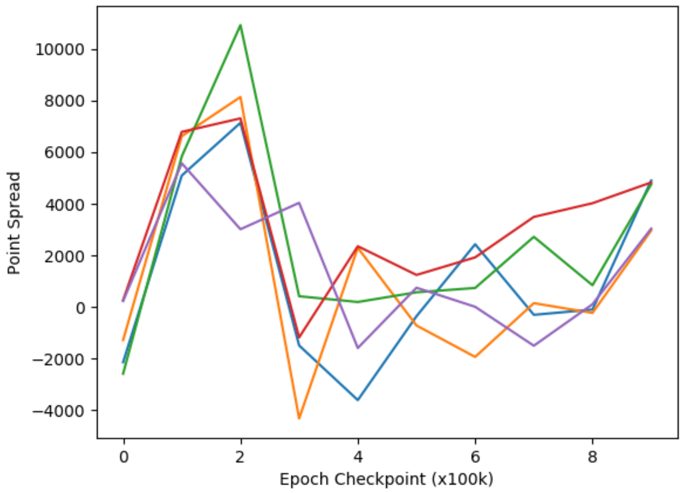
(a) Point spreads for 9-game matches



(b) Point spreads for 100-game matches.



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



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

Figure 19: Point spreads across matches of varying lengths. In each graph, the final learnèd agent is played against its previous checkpoint iterations.

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 is speculated to be a result of learning how to outplay its opposing agent rather than an understanding of the game. This is contingent upon the results of the round-robin play though, so no conclusions can be drawn at this time.

Also of further note is the scale on the aforementioned graphs. In Figure 19d, 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.

6 Conclusion

Generic closing remarks and rephrasing of the original introduction again in parting. Because this is a thesis and likely fairly long, “In section X, we covered Y” is probably allowed and not tacky.

References

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

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

BDSS02 Billings, D., Davidson, A., Schaeffer, J. and Szafron, D., The challenge of poker. *Artificial Intelligence*, 134,1-2(2002), pages 201–240.

BFGL17 Brown, A., Fisher, G., Gilman, S. and Lang, S., Overhead delivery system for transporting products, April 13 2017. URL https://www.google.com/patents/US20170101182. US Patent App. 14/881,217.

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/.

PMASA06 Ponsen, M., Munoz-Avila, H., Spronck, P. and Aha, D. W., Automatically generating game tactics through evolutionary learning. *AI Magazine*, 27,3(2006), page 75.

Pon04 Ponsen, M., *Improving adaptive game AI with evolutionary learning*. Ph.D. thesis, TUDelft, 2004.

RW11 Rubin, J. and Watson, I., Computer poker: A review. *Artificial Intelligence*, 175,5-6(2011), pages 958–987.

SPSKP06 Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I. and Postma, E., Adaptive game ai with dynamic scripting. *Machine Learning*, 63,3(2006), pages 217–248.

A Cribbage Scoring Rules

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 3: Scoring Rules for a game of cribbage [ACCb]

B Something

C Else