Bayesian Graphical and Deep Q-Learning Agents in Colonel Blotto Game

Olivia H. Weiner and Thomas L. Mayer

March 20, 2023

Abstract

Colonel Blotto games are an enduring puzzle in game theory with many useful applications. Solving for a viable strategy can be a difficult task, and solving for the optimal strategy depends heavily on context and knowledge of one's opponents. To attempt to compare various methods, we designed two kinds of agents (Deep Q-Learning agents and Bayesian Graphical Agents) utilizing different automated decision-making strategies and had them compete over 1000 iterations of blotto with our baseline agents. We found that over our initial 1000 iterations, the Bayesian Graphical Agents outperformed all other agents, whereas Deep Q-Learning Agents were only able to outperform deterministic baselines. By running a second experiment in which a Deep Q-Learning Agent was trained for 4000 iterations, we showed that the failure of the Deep Q-Learning Agent was likely a fluke of the limited data. We thus showed that under settings with fewer iterations, BGAs outperform Deep Q-Learning Agents.

1 Introduction

1.1 Problem Description

The problem we explore is solving for a viable strategy for the game Colonel Blotto. Blotto is a game that involves any number of players as well as 10 towers which the players compete for. In every round each player is assigned 100 soldiers to allocate among the 10 towers. The scoring happens in pairs, with each player assigned to every other player. Within each pair, the player with the most points at the nth tower wins n points, and the player with the most points at the end of a round wins that round. Variants of the game can be created by changing either the number of towers, the number of soldiers, or both. The winner can also be calculated by seeing which player won the greatest number out of their one-on-one battles, although we will primarily consider the former defintion of winner.

One important detail to note about the game is that there is no single best strategy, and that the best strategy in a tournament is dependent on the other players participating in the game. The best optimal against a skilled player is different from the optimal strategy against an unskilled player. The fact that you could be playing against any number of skilled/unskilled players further increases the complexity. Our project focuses on the interaction uncertainty that arises from uncertainty in the actions of other players in the game. Our project will involve exploring AI agents that can learn from other players to optimize its strategy against them.

Although AI agents have been applied to the Colonel Blotto game in the past, we could not find any examples of Deep Q-Learning or Bayesian Graphical models being applied. We hypothesized that these models could perform well in the game. Most of the research we found on Blotto explored Nash equilibria, such as Li and Zheng's work which explores Nash equilibria in 2 person games, or Boix-Adserà, Edelman, and Jayanti's work which explores Nash Equilibria in other variations of the game, for various numbers of soldiers and towers. However, since the optimal strategy in the game depends on the player you are facing, we were more interested in online methods that could adapt to the unique circumstances of each game, and perform well against a variety of players. The most similar published approach to ours we could find was "Reinforcement Learning Agents in Colonel Blotto" by Joseph Noel. This paper uses Vanilla Q-learning to play against one random agent or against

several random agents. The paper found, surprisingly, that the strategy for playing against multiple players is "almost the complete opposite" of the optimal strategy for playing against a single opponent.

The Colonel Blotto game also has several applications beyond game-playing. Theory developed to optimize agents in the game can also be used to model electoral competition, with each player allocating a certain amount of resources to counties in order to sway the votes in their favour (Roberson). In addition to political strategy, the theory can also be extended to various military strategy applications, R&D patent applications, as well as network strategy (Guan et al.).

1.2 Approach

We approach the problem by simulating a Blotto game using several agents, which will include some baseline agents as well as two AI-informed agents which compete for the winning strategy. As a baseline, we considered agents that use a single strategy every round, and agents that choose randomly from a predetermined set of strategies. For our AI agents, we will include an agent trained on Bayesian graphical models and a Deep Q-Learning agent. These agents will utilize online AI methods, using the results from each round to improve their strategy for the next round and compete for the most wins.

We chose to use online AI agents because a key detail of the game is that the optimal strategy depends on what players the agent is competing with, so we wanted our agents to be able to develop an optimal strategy in any game, even when they do not know information about the other players in advance. Choosing an online method also let our models compete against each other in real time and adjust to the other player adjusting their strategy. We also chose to include several randomized and hard-coded agents as well as a Nash equilibrium agent to approximate the types of strategies that a general population would use, and so that our agents would learn how to compete against multiple players. These agents also act as a baseline against which we can measure our AI agents' success.

2 Algorithms and Methodology

2.1 Game Simulation

For the simulation of the blotto games, we created an interface for a generic agent with unspecified internal parameters and two functions: one to output its opening strategy, and the other to take in the results of the most recent round (the strategies used by all agents and the resulting scores), potentially update the internal parameters accordingly, and output the strategy for the next round.

Although we based our results and objectives based on maximizing the number of points scored in each round, we also tracked the number of wins each agent achieved in each round, with a win defined as having more points than an opponent in a one-on-one competition. The simulation outputs the historical logs of both metrics at the end, as well as the average result of both for each agent.

2.2 Deep Q-Learning Agent

Deep Q-Learning is a reinforcement learning algorithm - it takes actions through a process of trial and error to explore and learn its environment before it effectively exploits it. The main reason we chose to use Deep Q-Learning instead of Vanilla Q-Learning was because of our vast state space. In fact, the state space grows with $O(S^{PT})$ where P is the number of players, S the number of soldiers each is assigned, and T is the number of towers. Deep Q-Learning replaces the Q-table used to keep track of states in Vanilla Q-Learning with a neural network, which is better able to handle the vast number of states, mapping (action, Q-value) pairs rather than (state-action, Q-value) pairs. Next, we will discuss some design choices we made when implementing our Deep Q-Learning network.

To address the balance of exploration vs. exploitation we choose to implement the Epsilon Greedy Exploration strategy. We experimented with several update rules for the epsilon values to find the optimal balance between exploration and exploitation. Unsurprisingly, the more games we trained on, the more our algorithm benefited from a long exploration phase. In our algorithm we also made the choice to use two neural networks: a model network and a target network. The structure of the two

networks are the same, however, the model network updates every 4 steps while the target network updates every 25 steps by copying over the weights from the model one. This is a common structure for Deep Q-Learning as it increases the stability in the learning process by avoiding having our target network influenced too strongly by the latest batch of updates (Mnih et al.). We chose to include 3 densely connected layers in each of our models, with Relu activation, He uniform initialization, and a Huber loss function. Huber loss was chosen because it is good at balancing learning vs. ignoring outliers in the data (Wu and Benkeser). In designing and writing our code for the Deep Learning model, we were guided by the suggestions of Mike Wang in his article "A Practical Guide to Deep Q-Networks" and his accompanying coded example.

Even when using deep learning, our game has a relatively large state and action space. For the action space I make each action a separate neuron in the neural network output. The amount of actions in the action space is $\binom{S+T-1}{S}$ where S is the amount of soldiers and T is the amount of towers. The action space, however, would be $\binom{S+T-1}{S}^P$ where P is the number of players. Because this is really large, we choose to simplify the state space, by making each state the addition of all the players' strategies for each tower. Here is the link to our Deep Learning agent.

2.3 Bayesian Graphical Agent

The Bayesian Graphical Agent (BGA) is based upon the idea of trying to predict the actions of other agents according to the Markov assumption. The Markov assumption posits (incorrectly but often helpfully) that the next state only depends on the action chosen and current state, irrespective of the total history of the domain. In this case, the current state is totally defined by the most recent allocations entered by every agent, meaning that this model assumes that all other agents base their next allocation only on the most recent round of allocations. Since all of our agents either act independently of history altogether or use the whole history of the simulation, this is a flawed assumption but could still potentially lead to effective strategies.

Basing its work upon the Markov assumption, the BGA uses all the previous allocation data at every step to construct a locally optimal Bayesian graph modelling the causal relationship between the allocations in successive rounds of blotto, under the constraint that a causal relationship can only occur with an allocation in the previous round affecting the expected allocation in the successor round.

The Bayesian graph allows the agent to use the last round of allocations to compute the probabilities of each possible allocation in the next round. From there, it computes the strategy with the locally maximal expected score given the probability distribution.

We believed that this would be a useful agent to consider as the key to blotto, at least anecdotally in the human setting, is predicting what your opponents will do, and usually an effective strategy will follow from a good prediction. We believed that Bayesian networks would be an powerful way to model these predictions, with the actual strategy entirely derived from their distributions.

2.4 Experimental Procedure

We ran a blotto simulation for 1000 rounds between 6 agents. In order to allow for time-effective computation, we used a drastically simplified model, but one still complex enough not to be trivially solvable: 4 towers and 6 soldiers to be allocated among them. We chose 2 baseline agents (one with the deterministic allocation [1, 1, 2, 2] and the other with stochastic allocations uniformly chosen from [1, 1, 3, 1], [1, 1, 1, 3], and [0, 0, 3, 3]), 2 Deep Q-Learning Agents, and 2 BGAs. We wanted to see which agent(s) could achieve the best performance given 1000 iterations to learn about their opponents and the reward system.

However, given the length of time that the Bayesian Graphical model takes per iteration, and the fact that the Deep Learning model learns at a slow rate (only updating the target model every 25 games) we also chose to simulate a game with only the deep learning model and baseline models that we could run for more iterations without taking too much time. We included 4 baselines: one that

plays [1,3,1,1] constantly, one that plays [3,0,3,0] constantly, one that plays [0,0,2,4] and [4,1,1,0] with 25% and 75% weights respectively, and one that plays [1,1,3,1], [1,1,1,3], and [0,0,3,3] with 30%, 30% and 40% likelihood respectively. We ran this simulation for 4000 rounds.

3 Results & Analysis

After 1000 iterations, we calculated the average score of each agent across the iterations in the game which included both BGAs and the DQN agents. We found that the two BGAs easily outperformed all the other agents, with the stochastic baseline agent performing the third best. After that, the two Deep Q-Learning agents performed slightly better than the deterministic baseline, with the gaps between their performance larger than the gaps between the worse-performing Deep Q-Learning agent and the deterministic baseline.

Although we optimized based on score alone, the rankings based on wins was exactly the same, with similar gaps in performance between agents as well. This makes sense, as score and wins are highly interrelated in their calculations, and it would be more surprising to see an agent's performance between the two metrics diverge significantly.

Agent	Scores	Wins
Deterministic baseline	21.479	1.139
Stochastic baseline	26.187	2.673
BGA 1	28.892	3.844
BGA 2	28.887	3.839
Deep Q-Learning 1	22.732	1.908
Deep Q-Learning 2	21.825	1.598

Table 1: Average scores and wins over 1000 iterations

We also kept track of the scores and wins over time, to capture how learning took place and our agents were able to adapt and potentially improve their strategies given more data. Perhaps surprisingly, none of the agents showed consistent improvement or deterioration across the iterations, meaning that there were no models that were able to improve their performance with more data, or if there were, then 1000 rounds did not provide enough data for such improvement to occur.

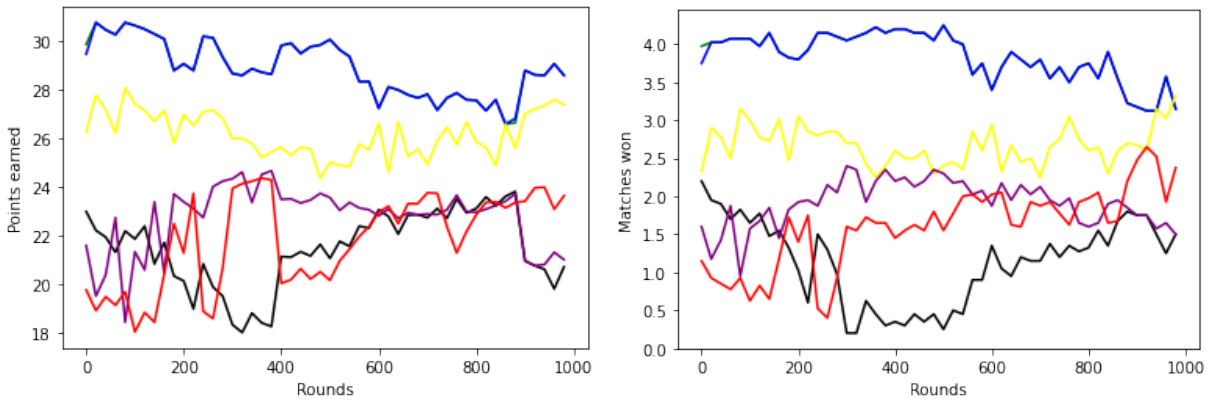

Figure 1: Scores (left) and wins (right) of the deterministic baseline agent (black), the stochastic baseline agent (yellow), the BGAs (green and blue - note that their performance was similar enough that the blue obscures the green), and the Deep Q-Learning Agents (purple and red) across 1000 iterations, averaged across sets of 10 iterations to eliminate noise and enhance readability.

To begin, the difference in performances of the two baselines was quite significant, with one managing to outperform the other consistently throughout the simulation, with a gap that thinned during some iterations but widened at others. This is somewhat to be expected, as a stochastic agent will naturally be harder for our learning agents to consistently beat, resulting in a better performance.

The outperformance of the BGAs as compared to the baseline agents was to be expected, since, given the highly predictable nature of the baseline agents, our BGAs were well-equipped to learn their strategies, with the Markov assumption providing a more complex model than necessary. Indeed, if the Bayesian graphs were truly optimal, then they should have captured the independence of the allocations of the baseline agents from the previous data.

The relative underperformance of the Deep Q-Learning agents was perhaps the most striking, given the power of Deep Q-Learning models. We hypothesized that perhaps due to the large size of the action space (84 possible strategies), the Deep Q-Learning agents were not able to learn effectively within so few iterations.

To test our hypothesis, as noted in our experimental procedure, we ran a second simulation with one Deep Q-Learning agent, and several baseline agents, so as to allow the agent sufficient exploration time to make significant improvements without the computation cost associated with updating BGAs over so many examples.

After 4000 iterations, we calculated the average score of each agent across iterations of the game. We found that the DQN agent outperformed 3 of the other agents.

Agent	Scores	Wins
Deep Q-Learning	24.3	3.0
Stochastic baseline 1	25	3.4
Stochastic baseline 2	14.6	1.4
Deterministic baseline 1	16.4	0.6
Deterministic baseline 2	20.0	1.8

Table 2: Average scores and wins over 4000 iterations

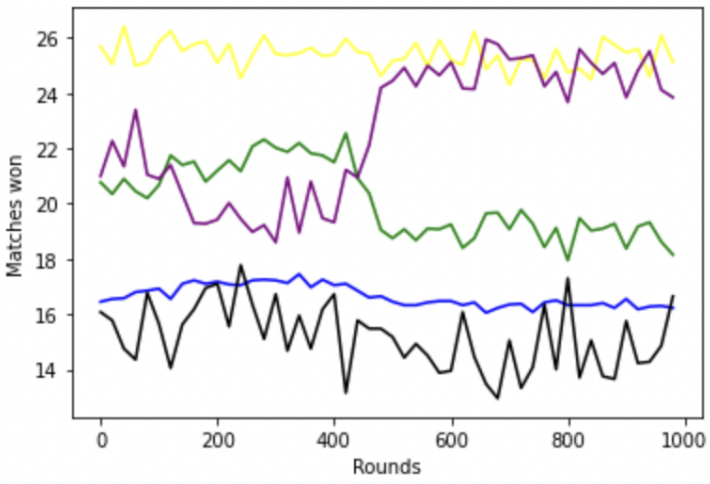

Figure 2: Scores of the deterministic baseline agents (black and blue), the stochastic baseline agents (yellow and green), and the Deep Q-Learning Agent (purple) across 4000 iterations, averaged across sets of 10 iterations to eliminate noise and enhance readability.

Similar to our other game, we also kept track of the scores and wins over time, to capture how

learning took place and our agents were able to adapt and potentially improve their strategies given more data. Although the DQN agent performed worse than one of the stochastic agents in average scores and wins, we see in the graph that towards the end of the game the DQN agent is performing at a similar level to the best stochastic agent. The DQN agent clearly starts with a lower strategy, we assume while it is in its exploration phase, and later improves to surpass the other strategies. It does so after about half of the rounds, when it has played 2000 games. Similar to our last match, the other agents did not show consistent improvement or deterioration across the iterations, indicating that they did not improve their performance with more data, as we expected. The clear jump in the strategy performance of the DQN model leads us to wonder if it would have performed even better had it been trained for more rounds. However the our time limitation implied that this is out of the scope for our project.

4 Conclusion

Our results demonstrated that BGAs perform better under settings with limited data, whereas Deep Q-Learning agents need many, many iterations of data to learn effective strategies. This result is potentially very important, due to the wide range of applications for Colonel Blotto games, and varying real-world constraints on computation time and sample size.

Unfortunately, our data remains inconclusive on the relative performance of BGAs and Deep Q-Learning agents across larger numbers of iterations, although it is clear that BGAs are not as computationally effective, as the runtime is $O(n^2)$ over n iterations, making it unsuitable for problems with significant time constraints between rounds or other computation restraints.

Additionally, our results are obviously limited by the number of agents surveyed, with only 4 fundamentally different agents explored. A better test of the effectiveness of our agents might involve either 100 or so diversely-coded agents, or perhaps background agents that are somehow trained to mimic human behavior in blotto, as other humans tend to be the opponents in blotto games, rather than fully automated agents (although of course humans may utilize automated agents in deploying their strategies).

Ultimately, even though the results are limited in their potential reach, this experiment still represents an important step in automating blotto strategies. While humans can frequently make irrational decisions, for a variety of reasons, automated agents do not as easily fall victim to such influences. And thus, if effective models can be implemented to outperform human agents, they should be.

5 Contributions

The overall simulation and BGAs were coded by Thomas, while the Deep Q-Learning Agents were coded by Olivia. Experiments were primarily run by Thomas. The write-up was primarily completed by Olivia.

References

Enric Boix-Adserà, Benjamin L. Edelman, and Siddhartha Jayanti. 2020. The Multiplayer Colonel Blotto Game. In Proceedings of the 21st ACM Conference on Economics and Computation (EC '20). Association for Computing Machinery, New York, NY, USA, 47–48. https://doi.org/10.1145/3391403.3399555

Hortala-Vallve, R., Llorente-Saguer, A. Pure strategy Nash equilibria in non-zero sum colonel Blotto games. Int J Game Theory 41, 331–343 (2012). https://doi.org/10.1007/s00182-011-0288-4

Noel, Joseph. (2022). Reinforcement Learning Agents in Colonel Blotto.

Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

Roberson, B. The Colonel Blotto game. Economic Theory 29, 1–24 (2006). https://doi.org/10.1007/s00199-005-0071-5

Kovenock, D., Roberson, B. Generalizations of the General Lotto and Colonel Blotto games. Econ Theory 71, 997–1032 (2021). https://doi.org/10.1007/s00199-020-01272-2

S. Guan, J. Wang, H. Yao, C. Jiang, Z. Han and Y. Ren, "Colonel Blotto Games in Network Systems: Models, Strategies, and Applications," in IEEE Transactions on Network Science and Engineering, vol. 7, no. 2, pp. 637-649, 1 April-June 2020, doi: 10.1109/TNSE.2019.2904530.

Wang, Michael. 2020. A Practical Guide to Deep Q-Networks. Towards Data Science. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

Wu, Ziyue and David C. Benkeser. "A Huber loss-based super learner with applications to healthcare expenditures." ArXiv abs/2205.06870 (2022): n. pag.

Xinmi Li, Jie Zheng, Pure strategy Nash Equilibrium in 2-contestant generalized lottery Colonel Blotto games, Journal of Mathematical Economics, Volume 103, 2022