Principles of AI, Fall 2021

Investigation of Reinforcement Learning Techniques in Minesweeper

Josh Cooper[1]

[1]Rutgers University

December 21, 2021

Abstract

Q Learning was introduced by Watkins[1] as a Reinforcement Learning algorithm in 1989. Since then, the field of RL has blossomed, creating fantastic successes in recommender systems [2], seeing titans fall in the world of Go with the success of AlphaGo [3], and also learning to master many distinct video games on the Atari game console using a technique known as Deep Q Learning [4]. For my part, I sought to understand the reasons behind the algorithmic choices of this field, and to investigate how a Deep Q Learning agent might be implemented in order to succeed in the game of Minesweeper.

I chose Minesweeper specifically because I already had an agent able to play it, but using a distinct set of algorithms (Constraint Satisfaction, Equation Inference, and Random Selection), and comparing the performance of the two agents would be informative. In addition, I created a Mine Prediction network using Supervised Learning as a third comparison, to compare and contrast two distinct types of Neural Net based agents. After training, the Supervised learning agent achieved scores comparable to the CSP solver, but the Q-Learning agent never really did better than a bot playing purely by random selection (the assumed baseline of all the models). I will be discussing methods, choices, reasoning, the philosophical implications of the various models, and eventual next steps to increase model accuracy.

Key Terms: Reinforcement Learning, Q Learning, Supervised Learning, Constraint Satisfaction, Minesweeper

1 Introduction

The concept of creating an agent which is capable of learning the inherent patterns of an input or image, and then acting on this information to maximize a reward has long been a goal of Artificial Intelligence. Through the field of Reinforcement Learning, and specifically the crystallization of this concept into the "Reward Hypothesis" which posits that all goals can be stated as the maximization of expected cumulative reward. The concept of Q learning, its extension to neural networks via Deep Q Learning, as well as some of the modifications made to these algorithms, are all attempts at the realization of this goal.

However, there are other methods which can prove effective in designing an agent to play a game of Minesweeper. One can train a neural net based on labeled examples of game boards, and then use this trained network to predict the likeliest locations of a mine. One can use non-ML methods like constraint satisfaction and equation inference in order to do brute force calculations over all possible states to identify the likely mine locations.

These three distinct methods were chosen for investigation here because they all represent distinct ways of how a human programmer, and a computer, examine information and create new understanding. Specifically, the RL method would be using a computer's internal organizational ability to create its own internal ontology, while the supervised learning and CSP/EI methods depend on external understanding of mine locations and a description of a predefined semantic network.

One thing that is constant over all the approaches is that the sheer size of the possible state space for the problem can create difficulties when attempting to use probabilistic models. I address these concerns in the end of the paper.

In terms of expectations, both on the results side, and the personal information side, this project has yielded unexpected gifts. Before beginning the project, I hadn't anticipated the difficulties which would arise based on the state space size, and the problems of sampling that state space to produce results with Q Learning. I also was expecting the supervised learning agent to perform at about the same level as the Q Learning agent, even if I didn't know where that level would end up being.

The considerations about the ontological problems between the RL, SL, and CSP versions also greatly increased my understanding about the advantages and limitations of the various algorithms, and will prove fruitful in my exploration of future projects.

2 Implementation Details & Method Choices

I am using a modified version of Minesweeper, which allows for the agent to continue playing until every square on the board has been clicked. I judge agent competency based on the average value of the turns when a mine is clicked. If the agent plays the best possible game, then all the mines will be clicked last, and the game score will be at a maximum, based on board size and mine density. IE, for a 3x3 board, there are 9 total moves to make. If the board has 3 mines on it, then the best possible game would involve the mines being clicked on turn 7,8 and 9, giving an average turn value of 8. To compare scores over different board sizes, I take all scores and normalize them to a percentage value of max possible score. All graphs presented have this "Optimal Score Percent" as the common connective value.

For the CSP agent, there are three distinct types shown in the results section. The basic agent is a single location inference agent, only capable of making choices based on clues in a single location. The advanced agent combines a full search lookup CSP agent with multi-location clue inferences. Advanced with Sel augments the advanced agent with a probabilistic random spot choice when it runs out of basic/advanced move choices. Further info about these three agents can be found in the secondary submitted report (440_Project_2.pdf), as they're too in depth to spend time on in this paper.

For my network, I was aiming to keep the inputs as simple as possible, so I settled on using the current visible state of the board as the input, and then either outputting a board of expected return values (for the Q Learning agent) or outputting the sigmoid activated percent chance a location could be a mine (for the Supervised Learning agent). The CSP agent does not function in this manner, and the only time it is used is as a comparison to the two other neural net agents in this paper.

For network architecture, I started as small as I could, and tested a variety of layers, layer sizes, dropout percent, regularization, and other tweaks which don't have much to do with the philosophical portion of the agent differences. You can find the comparisons between different network types and layer choices attached as images at the end of the paper.

After investigation, I settled on a single dense layer with a size that was 80*(board_size*board_size), and surrounded it with a pair of dropout layers. This led to a dense layer of (board_size*board_size) which functioned as a single cell location for the whole board. Finally, there is a reshaping layer to make the output more human readable. Network shape was consistent for both Q Learning and Supervised learning, with the main difference between them being that Q Learning had a dropout coefficient of 0.375, and Supervised Learning had a dropout coefficient of 0.44.

To update the Q Network, I used the standard Q Learning update equation from Sutton and Barto [5], specifically predicting that a certain state, action pair s,a would be updated by the following relationship.

$$\underbrace{Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{Learning Rate}}{\alpha} \; [\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma} \; \overbrace{\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)]$$

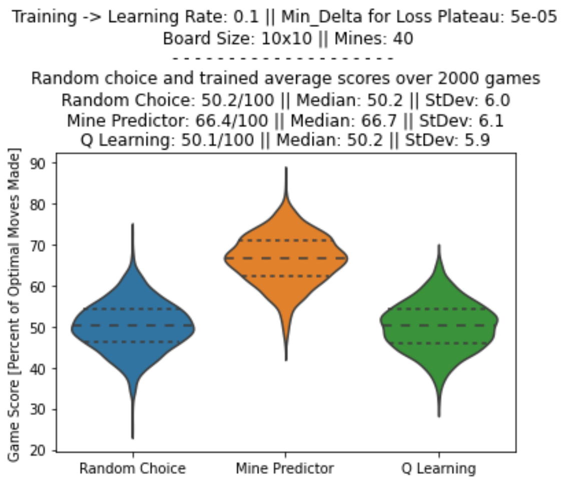

Figure 1: Neural Net Agent Comparisons

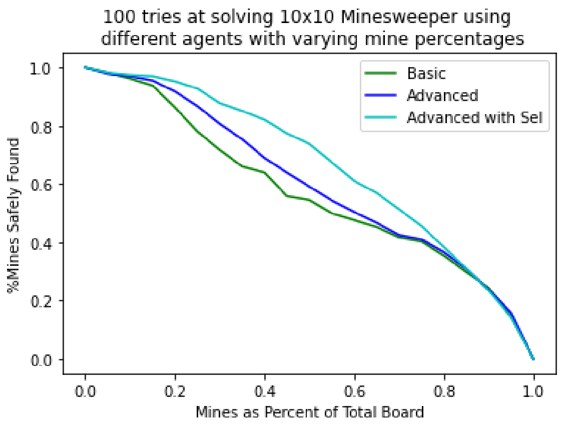

Figure 2: CSP Agents of Varying Complexity

3 Results

Overall, the project did not perform as expected on almost every level. The Q learning agent never outperformed an agent which made random choices, the Supervised learning agent always constantly outperformed the Q learning agent, and the CSP agent always did at least as well, or better, than the Supervised Agent. Board size had no appreciable effect on the neural net agent's performance, but mine density caused massive variation in the supervised learning agent and the CSP agents success rates. Due to time constraints, I wasn't able to collect data about the Q Learning agent over a variety of mine densities, but for reasons discussed in the next section, it wouldn't have mattered.

The Mine Prediction network was able to play games at around 67% optimal moves, the Q Learning agent never got much past 50% optimal, which was almost exactly as good as an agent which chose moves purely at random, and the CSP agent chose the best moves between 65% and 85% of the time, depending on its own internal algorithms and choices.

When comparing these results with other projects which followed along the same lines, my Q Learning agent was much less successful, but my CSP and SL agents were usually more successful than other examples I found. There are quite a few reasons for the Q Learning agents lack of success based on how I set up my data collection, my internal reward structure, and also due to time constraints in calculations.

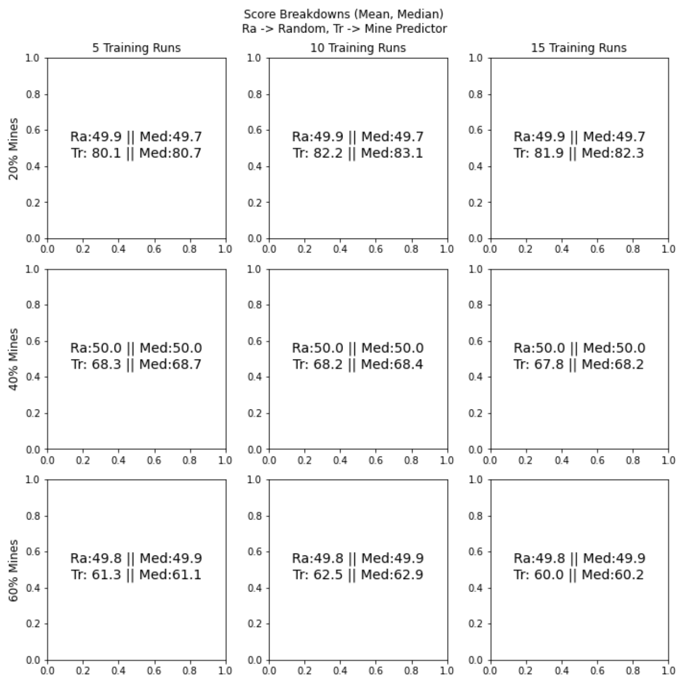

Figure 3: Mine Density and Training Comparisons for Mine Predictor (Supervised Learning)

4 Discussion

These results bring up a pair of distinct problems to be dealt with in future work. First, the state space of a board with dimensions NxN would have a naive upper bound of 10^{N*N} which means that even with the best sampling possible, any board over 3x3 would be having an insane number of states to possibly iterate through. Q Learning is an algorithm which is guaranteed to converge over repeated visits to the same state/action pairs, but even in my largest data collections (constantly sampling from and recreating a buffer of approximately one million states on a 10x10 board) I don't think I ever could have visited the same state more than once. This lack of repeated visit is likely the primary culprit for the Q Learning agent essentially acting no better than random, since it literally never had the chance to extract information from multiple state visits.

Secondly, the reward structure I used in training my Q Learning agent was not ideal upon comparison with other work. I only allowed my agent to gain insight from when it clicked on a mine (as a -1 reward) which means that every other move, including game winning, and clicking on a safe spot, was considered the same. A much more in depth reward structure, which consisted of positive rewards for clicking on a non mine, winning the game, while also including negative rewards for random guesses and clicking on a mine, could have helped alleviate some of the growing pains of the massive search space.[6][7]

One thing that I did notice in all the other papers I surveyed, is that their results were based on much much smaller boards than my own. However, even with that decrease in state space, they still went through training over data sets which dwarfed my attempts by at least two to three factors of 10. Some came up with Q Learning agents drastically outperforming mine [6], some had almost comparable results [8], and others somehow created agents which drastically under performed [7]. I think the takeaway from all of this is that the dataset size and sample amounts are going to be the key indicators for good results using Q Learning.

Based on the concept of AI principles however, this brings up a very interesting dichotomy between the ease of using externally defined information compared to forcing a program to come up with its own internal information organization. The CSP and SL agents are essentially bootstrapping their own internal agents with human provided knowledge, and are thus being given a massive advantage over the Q learning agent, who is forced to come up with its own internal representations of Minesweeper based on a limited number of games. This shows both the power of human in the loop AI training, and the limitations of current dataheavy computational approaches to learning.

5 Future Work

There are a variety of ways which I could extend this project, from the aforementioned dataset extension, to redesigning the reward structure. However, I think the most interesting extension would be involving the use of the Policy Gradient family of algorithms, which are fundamentally different in their approach compared to the Q Learning algorithms. Policy Gradient doesn't depend as heavily on the repeated visits as Q Learning, and is possibly much more capable of finding an optimal policy from a smaller subset of data. I will be attempting to code this after the semester is over for my own understanding.

Conclusions

I instigated this project with the hope of understanding the basic conceptual building blocks of reinforcement learning, and comparing it to a variety of other methods for creating an agent capable of interacting with a limited information game like Minesweeper. On that front, I have succeeded wildly. When it comes to the actual performance of the RL agent, while it may not have performed admirably based on this current run of data collection, I have a variety of ways which I can improve the agent, both in terms of tweaking the structure of the rewards, the type of network, and even the choice of algorithm. As a learning experience, and as an exposure to some of the main ideas which will cause future concerns in the area of AI learning development, this project has been a success. Still wish the Q Learning agent could have broken out of the "literally not doing anything" pen I trapped it in, but hey, who get's it right first try.

References

[1] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[2] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey, 2021.

[3] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre,

George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[6] Evolution strategies and reinforcement learning for a minesweeper agent. https://github.com/jakejhansen/minesweeper_solver/blob/master/article.pdf. Accessed: 2020-12-20.

[7] Reinforcement learning for constraint satisfaction game agents. https://arxiv.org/ftp/arxiv/papers/2102/2102.06019.pdf. Accessed: 2020-12-20.

[8] Minesweeper reinforcement learning. https://upgado.com/2020/04/16/minesweeper.html. Accessed: 2020-12-20.