Using Reinforcement Learning to Teach an Agent to Play 2048

Alexandra Walker
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
Awalke17@uccs.edu

Abstract

In this paper I will propose a test for Reinforcement Learning implementing the Markov method and experimenting with Q-learning and Monte-Carlo Search tree algorithms on 2048. Reinforcement Learning and with the Markov process requires a reward policy, 2048 has several different strategies and scoring systems inherently so testing each method with different reward systems will be interesting. This process plans on reporting results based on learning and search algorithms combined with different reward policy's and recording the best win/loose state for each.

Introduction

The game 2048 is a deceptively simple game, by using your keyboard or mouse or finger you move numbers on a grid to slide in the direction chosen. Human players will discover some strategies that make it easier to win. By using 2048 the agent will be given the chance to develop complex strategies. The goal of the agent will be to win 2048, the goal of this paper is to discover what rewards and algorithms optimize the agent's learning.
The reward set will experiment between rewarding behaviors like keeping high numbers in a corner versus rewarding the combining of numbers. The reward set may also reward a combination between behaviors and actions.

This paper will explore 2 learning algorithms along with the reward system proposed. Monte Carlo Search Tree and Q-learning. Papers (Nathaniel Hoeanwan and Wu 2020), (Kaundinya et al. 2018), and (Wang, Emmerich, and Plaat 2018) and (Chentsov and Belyaev 2020) all explore this idea with very results based on their different approaches. By combining results and making adjustments like adjust the reward system to see if improved results can be achieved.

Reinforcement learning is as described in Reinforcement learning: an Introduction (Sutton and Barto 2020) as ”learning what to do - how to map situations to actions - so as to maximize a numerical reward signal.” This reward signal is a reward sent by the environment to the agent. This reward is then summed in and evaluated in different methods. Over the course of a single episode of learning an agent can be rewarded several times or a single time. How the user chooses to define and distribute the reward can directly effect how the agent preforms and the optimal policy created.

The Markov decision process will also be implemented and will be used to develop the agents policy based on it's environment and rewards. Due to the nature of 2048 every state contain a summary of all moves before it, for example if you have a 16 tile on the board then at some point in the game the player combined 2 eights, 4 fours and somewhere between 2 and 8 twos. (Due to the random nature of the game it is not guaranteed, and in fact is unlikely every single 4 on the board was created by the player.) Therefore 2048 will have the Markov property.

I plan on implementing the agent and the environment in python using a game class for the environment and an agent class. The agent will have the set of actions Up, Down, Left, Right to choose from. As humans have access to the entire environment at all times so will the agent.

Background

Vocabulary

In this paper there are several similar concepts so defining each concept is useful.

- A **spawned number** is a value placed on the board randomly this number can be a 2 or a 4.
- A **combined number** is the value created when two numbers are combined.
- The **highest number** is the number on the board that is greater or equal to all other values placed.

2048

2048 is a 1 player game that released in 2014 and created by Gabriele Cirulli. The game played by using your arrow keys to chose a direction, then the game slides all the numbers on the board in that direction if two numbers are the same then the game will combine them increasing the score. For example if 2 fours are combined then the game increases the score by 8. 2048 will only have numbers on the board consisting of powers of two aka 2,4,8 . . . 1024, 2048 by combining the two 1024 the player will win and then be prompted if they want to continue. If the player does choose to continue the

game then you enter an endless mode where there is no win-condition and the challenge changes to simply trying to get the highest score possible. Due to the size of the board the highest possible tile in 2048 is 131,072 as there is no more space to reach a higher value and is considered the ultimate win state.
2048 can only be lost when the player leaves no more room for a new number to spawn and the player cannot make any more moves.

Strategies

2048 has a few easy strategies that without make the game more difficult. The first strategy can be called three directions. The player picks 3 directions they will use mainly, and the fourth will only be used when in a stuck state, when all 3 moves do not allow for the game to be progressed in any way. The second is to simply keep all the numbers on one wall with the highest value in one corner and each subsequent highest value moving along that wall in a descending order.

Literature review

Playing 2048 with Reinforcement Learning (Li and Peng 2021)
Li and Peng described their rewarding system as directly using the score system from the game. Their paper chose to focus more on the searching algorithm for each AI design. They had a baseline, random action, where the AI reached a highest number of 128 only 48% of the time. With their best performing algorithm the Bream Search having a highest number of 2048, the win state, at 28.5% of the time and even reaching 4096 0.4% of the time.

What's in a Game: Solving 2048 with Reinforcement Learning (Nathaniel Hoeanwan and Wu 2020)
Their paper focused on testing the evaluation function used to determine the best agent plays. They Implemented a Monte Carlo tree search to find the maximum valued play to make. Each test prioritized a different scoring method: taking the largest sum on the board, taking the largest tile or number on the board, or taking the largest of the total score. They also chose the change the number of roll-outs for the Monte Carlo Tree search and experimenting with how well the agent played with 50, 100, 250, and 500 roll-outs.
Their results showed that taking the largest sum of the board with 500 roll-outs had the best results. Managing to make 2048 43% of the time with an average sum of the final board being 2444.58 and the average final score being 19107.28. The runner up method being using the largest total score as the evaluation method. This method had similar results.

Monte Carlo Tree Search Modification for Computer Games (Chentsov and Belyaev 2020)
Chentsov and Blyaev re-imagined the Monte Carlo search tree algorithm to optimise for computer games. They chose to include a parameter in the evaluation function. This parameter would change the probability of the MCT choosing one solution over another. This parameter would be determined by a few properties:

- Location of agents and obstacles
- Previous actions

They concluded that this change would work better for games that don't depend on player location.

Game Playing agent for 2048 using Deep Reinforcement Learning(Kaundinya et al. 2018)
In this paper the authors attempted to create an AI to play 2048. Focusing on Q-learning and SARSA algorithms to solve the game. Their reward function was an attempt at finding what was most beneficial for the agent. They attempted 3 different functions:

- Rewarding based on strategy.
- Rewarding based off of their experience replay for a games over a threshold.
- Using the hole game for their experience replay and training the neural network. Also using the score as a basic reward.

All three of their attempts were deemed unsuccessful as their AI was never able to reach 2048. However its useful to understand where they failed. Their recommendation was to keep rewards between -1 and 1 based on the score. They also recommended using Q-architecture instead of Q-learning.

Monte Carlo Q-learning for General Game Playing (Wang, Emmerich, and Plaat 2018)
The Authors explore the field of General Game Playing (GGP). Focusing on Q-learning and Q-learning with Monte Carlo Lookahead. Finding that Q-learning does indeed converge for real games like Hex, Tic-Tac-Toe, and Connect four, however the convergence is slow and optimal. When exploring Q-learning with Monte Carlo Lookahead the convergence was significantly faster especially in the early game.

Evaluation

The agent will be evaluated on the percentage it successfully creates a 2048 tile. Other evaluation methods like average score per game and number of moves made will also factor into the evaluation of the agent. With Python being a notoriously slow language, as its interpreted, it doesn't make since to evaluate the agent on the time played so I will be omitting that point. Although that data may be logged and reported.

References

Chentsov, D. A.; and Belyaev, S. A. 2020. Monte Carlo Tree Search Modification for Computer Games. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 252–255.

Kaundinya, V.; Jain, S.; Saligram, S.; Vanamala, C. K.; and B, A. 2018. Game Playing Agent for 2048 using Deep Reinforcement Learning. *NCICCNDA*.

Li, S.; and Peng, V. 2021. Playing 2048 With Reinforcement Learning. *CoRR*, abs/2110.10374.

Nathaniel Hoeanwan, S. T.; and Wu, K. 2020. What's in a Game: Solving 2048 with Reinforcement Learning.

Sutton, R. S.; and Barto, A. G. 2020. *Reinforcement Learning an Introduction: second edition*. Westchest Publishing Services.

Wang, H.; Emmerich, M.; and Plaat, A. 2018. Monte Carlo Q-learning for General Game Playing. *CoRR*, abs/1802.05944.