2048: A comparison between Q-Learning and Expectimax over an Extended Game Space

Ruben Mayer, Magdy Saleh, Jayden Navarro

Figure 1: Max Tile Histogram 4x4

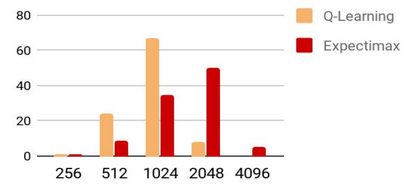
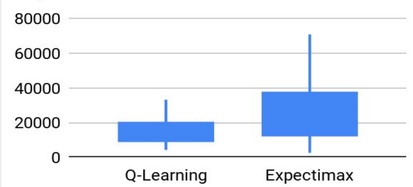


Figure 2: Scores 4x4



Background

2048 is a game that operates on a 4x4 board consisting of number tiles. Every move, same-number tiles are combined, and a 2 or a 4 tile is spawned on an open position. The goal is to reach 2048, but you can continue to even larger powers of two. Since the game has a state space of size $O(10^{15})$, implementing a good player for it can be challenging.

Problem Statement

Our goal for this project was to compare a brute-force search method (Expectimax) with a reinforcement learning method (Q-Learning), and to explore how these methods compared as the state space increased, specifically on board sizes 5x5 and 6x6.

Results

To ensure a fair comparison, and allow time for us to collect enough data, we set a limit of one minute per game for Expectimax (by tuning the hyperparameters to achieve this), and limited the training time for Q-Learning to a comparable time per move (about 2 hours).

Expectimax yielded larger scores and max tiles compared to Q-Learning, This difference was more notable on the 5x5 board, but Q-Learning made up some of the difference on the 6x6 board, which can be explained as follows:

1. Expectimax performs better as its hand-tuned heuristics generalize more effectively on the board positions than the Q-Learning features.
2. Q-Learning's relative performance decreased on the 5x5 board because the feature space grew too large ($O(10^9)$) to enable effective generalization. Meanwhile, Expectimax performance worsened on the 6x6 board as the larger state required a prohibitively low depth of 2.

Future Work

The biggest areas for improvement in our implementations are:

- Increasing the speed of our Expectimax implementation
- Optimizing the storage size of the weight vector for Q-Learning

These will enable us to explore deeper and train longer, respectively.

After making the aforementioned improvements, we will explore how the relative performance between the approaches changes as we increase the running times through longer training and larger hyperparameters.

Figure 3: Max Tile Histogram 5x5

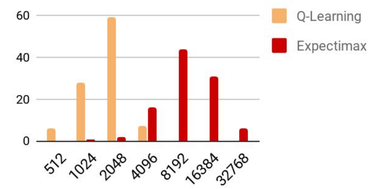
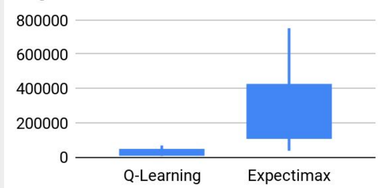


Figure 4: Scores 5x5



Q-Learning with Linear Approximation

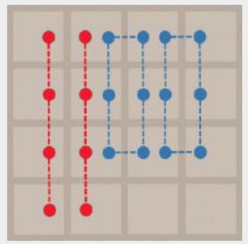
Overview

We implemented a modified version of Q-Learning to generalize the state space. We used a combination of N-Tuple-Network features as well as an adjusted update function.

Learning Features

We implemented two types of N-Tuple Network features:

1. The first feature tracks the values of vertical and horizontal straight-line arrangement of the tiles on the board (a tuple network of size 4, see red lines in figure to the right).
2. The second feature tracks the values of tiles forming 3 x 2 rectangles on the board (a tuple network of size 6, see blue lines in figure to the right).



These features are effective because they encapsulate the most important aspects of the game: the arrangement and position of all adjacent tiles.

Adjustment of Update Function

In order to reduce the impact of update values that were too large, and to progressively reduce the impact of each update on a per-feature basis, we made a modification to the traditional Q-Learning update function, from:

$$weights[feature] = weights[feature] \times \eta UpdateValue$$

to:

$$weights[feature] = weights[feature] \times \frac{\sqrt{UpdateValue}}{\sqrt{1 + i_{feature}}}$$

Figure 5: Max Tile Histogram 6x6

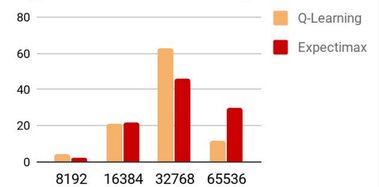
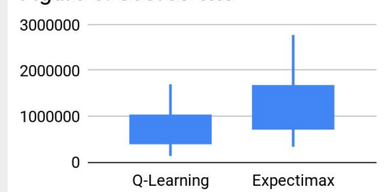


Figure 6: Scores 6x6



Expectimax

Overview

Expectimax uses a brute-force search to find the optimal move in a game where one player is trying to maximize their result, and their "opponent" performs an action according to a probability distribution.

Implementation

To find the best move, Expectimax simulates all possible board states up to a given depth. It alternates between the following:

- *User move:*
  - Simulate all moves: up, down, left, right
  - Return move that yields maximum score
- *Tile Spawn:*
  - Simulate 2 and 4 tile spawns on every empty position
  - Return expected score (sum of all simulation scores weighted by their likelihood of occurring)

After a specified simulation depth or probability cutoff is reached, a heuristic is used to determine the score of the given board.

Heuristic

We consider a heuristic for approximating the strength of a board arrangement that is based on:

- A reward for having the max tile in a corner
- A reward per unoccupied tile on the board
- A penalty for not having the rows and columns monotonically increasing
- A reward for each pair of adjacent equal valued tiles

Optimizations

In order to use sufficiently large simulation depths, we optimized our implementation as follows:

- Converted Python implementation to C++
- Represented board as flat $\log_2$ byte array
- Pruned low-probability simulation states by implementing a probability cutoff