The 2048 Challenge

Rui Qiu[1], Yanpei Tian[2], and Yulian Zhou[3]
[1,3]Department of Computer Science, [2]Department of Electrical Engineering

Motivation

2048 is a single player puzzle game, and its popularity surged immediately after its first release in March 2014. As the game can be modeled into game states and actions spaces, we can leverage multiple tools from CS221 course, and compare different AI strategies in solving the game.

Problem Definition

2048 is played on a 4×4 grid with each numbered tile being a power of 2. The objective of the game is to produce a tile with value 2048 by moving the board 4-directionally and merging two consecutive cells with same value to form a cell with a larger value.

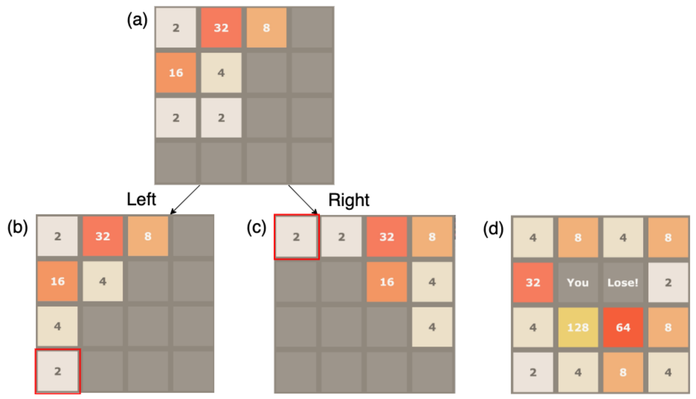

Figure: Different states in one 2048 game.

Challenges

- To define strategies precisely, and test them rigorously.
- To come up with good heuristic functions to evaluate game state for the corresponding strategies.

Approaches

- **Expectimax**
  - Encode each row movement into a lookup table.
  - Eval$(s) = \sum_{i,j} \text{weight}(i,j) \times \text{score}(i,j)$.
  - Minimax with alpha-beta pruning.

$$V_{\text{exptmax}}(s,d) = \begin{cases} \text{Utility}(s) & \text{isEnd}(s) \\ \text{Eval}(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a),d) & \text{Player}(s) = \text{“human”} \\ \sum_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a),d-1)/|\text{Actions}(s)| & \text{Player}(s) = \text{“computer”} \end{cases}$$

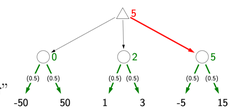

Figure: Illustration of Expectimax strategy.

- **Monte Carlo tree search**
  - State s: Current board layout
  - Action a: left, right, up, or down
  - $Q_\pi(s,a)$: Expected utility starting at s, first taking action a, and then following random policy π
  - Discount factor γ: 1
  - Utility: Total scores at the end of the game

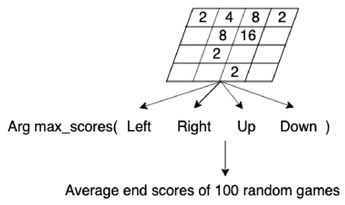

Figure: Monte Carlo tree search

- **Deep reinforcement learning**
  - Model: Inception analogus model
  - Backbone: Conv-ReLU-Conv-ReLU-FC
  - Input: One-hot encoded board
  - Output: V_{opt} of the the board
  - Loss: Mean squared loss
  - Action a: left, right, up, or down
  - Reward(s,a,s'): Scores earned from s to s'
  - RL algorithm: Value iteration $(T(s,a,s') = 1)$

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{isEnd}(s) \\ \max_{a \in \text{Actions}(s)} [\text{Reward}(s,a,s') + V_{\text{opt}}(s')] & \text{otherwise.} \end{cases}$$

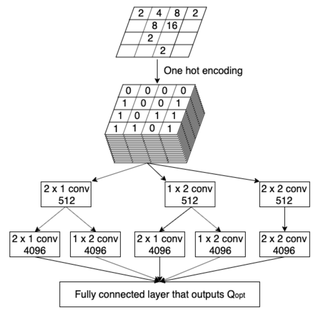

Figure: Deep RL network

Results

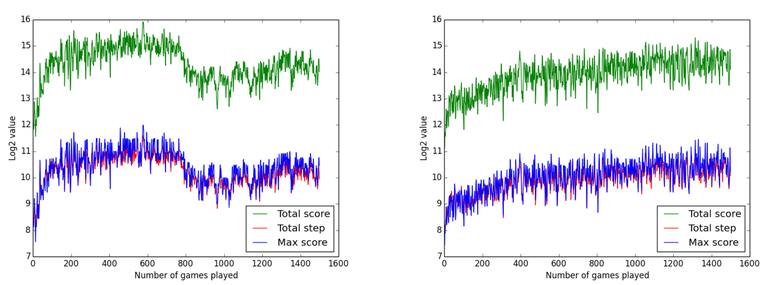

Figure: Training curves of deep reinforcement learning models (different versions of neural networks)

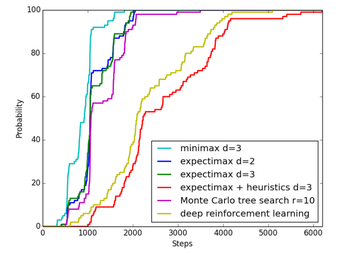

Figure: Cumulative distribution function - Steps

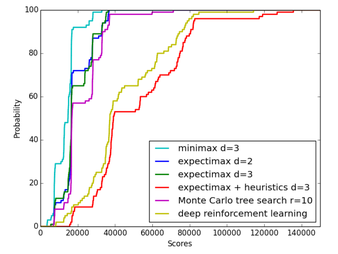

Figure: Cumulative distribution function - Scores

Table: Success number out of 100 games

Max tile value	16	32	64	128	256	512	1024	2048	4096	8192
Random	100	99	93	62	11	0	0	0	0	0
Greedy	100	100	99	91	50	5	0	0	0	0
Expectimax ($d=2$)	100	100	100	100	100	100	89	28	0	0
Expectimax ($d=3$)	100	100	100	100	100	100	87	35	0	0
Expectimax + h ($d=3$)	100	100	100	100	100	100	100	91	47	4
Minimax ($d=3$)	100	100	100	100	100	97	71	8	0	0
Monte Carlo ($r=10$)	100	100	100	100	100	100	92	43	2	0
Deep RL	100	100	100	100	100	100	98	86	34	1

Analysis

- Expectimax
  - We created a lookup table for each movement (rather than performing the actual move), which increased the search speed.
  - As the state space grow exponentially with search depth d, expectimax with chance sampling may help to increase d in future experiments.
- Monte Carlo tree search
  - It is slow but effective, even with 10 random runs per action.
  - We would experiment with other utility heuristics and compare their performance.
  - We would also try different early stop mechanisms to improve the search speed.
- Deep reinforcement learning
  - We have tried epsilon-greedy ($\varepsilon = 0.2$) and Gibbs sampling, yet we found that using the optimal strategy is beneficial for exploring more state space with higher scores.
  - Due to the randomness of the computer move, we should calculate the $V_{opt}(s')$ with transitional probability, but we assume each transition is deterministic to reduce the computational cost. Future improvement could be done by taking the randomness into account.

Acknowledgement

The authors would like to thank all CS221 course staff for instructions and guidance on this project, and we especially thank Di Bai for giving insightful inputs.