Reinforcement learning in 2048 game

Adrián Goga
Supervisor: prof. Ing. Igor Farkaš, Dr.

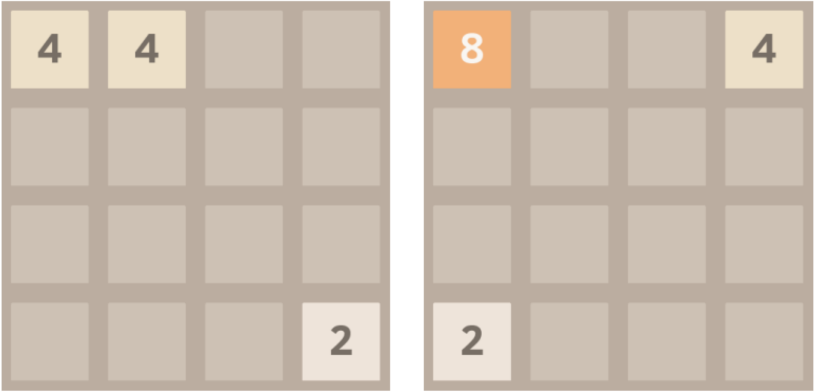
Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

June 27, 2018

AI in games playing

- Board games are an interesting setting for Artificial Intelligence
  - Often require planning
  - Sometimes involve randomness
- The game of 2048 is both stochastic and requires planning

Intro to the game



After selecting the $\leftarrow$ action, the two 4 tiles merge into an 8 and a new 4 tile spawns in the corner.

2048 in AI research

Tree search strategies (e.g. expectimax):

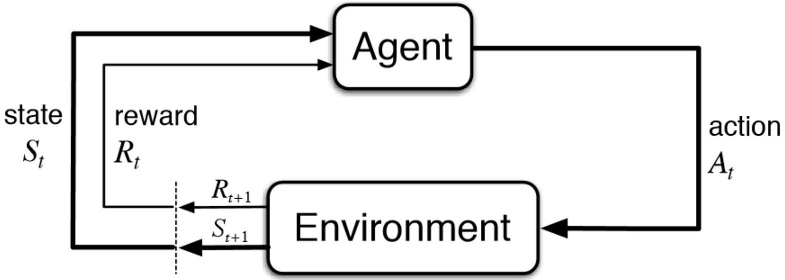
- Use heuristics based on human made analysis
- Achieve very good results
- 'Just' an optimized brute-force, not really AI

Reinforcement learning:

- Similar to learning in nature
- No prior knowledge of the game required

Reinforcement learning

- Agent interacts with the environment
- Trial and error learning (as opposed to supervised)
- Each interaction is a state transition (S_t, A_t, R_t, S_{t+1})
- The goal is to find the optimal policy (maximize the expected rewards)



Q-learning

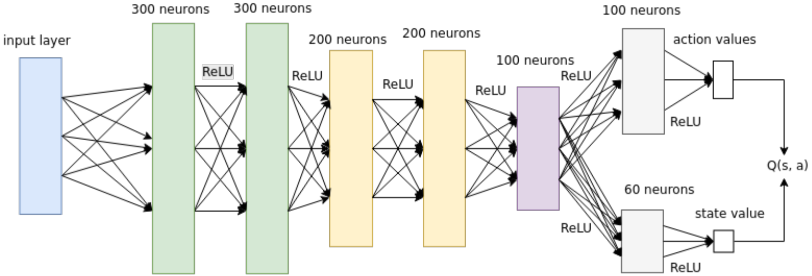
- A popular RL algorithm (Atari from pixels)
- $Q(s, a)$ is the expected cumulative reward obtained by taking action a in state s and playing optimally
- Algorithm works by updating the Q-function
- It is proven that the Q-learning converges to optimal policy

Deep Q Network

- Problem: too many states
- Solution: approximate the Q-function using a neural network
- Convergence is not guaranteed anymore

The base model

- Implemented in Python using Keras library with TensorFlow backend
- Dueling Double DQN + PER
- ϵ-greedy exploration



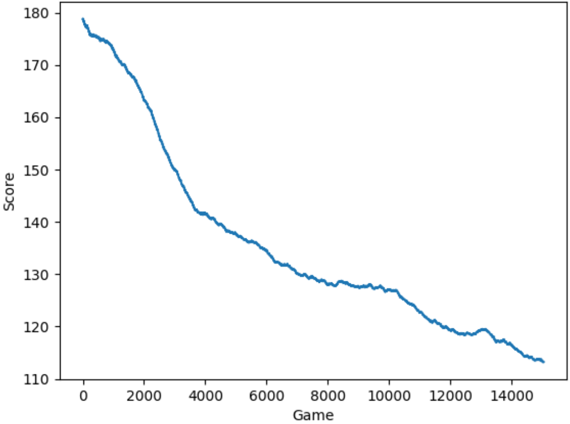
Experimental settings

- Score: the sum of the tiles at the end of the game
- We allow moves that do not move any tiles in hope of better stability of learning

Experiment 1

- Input encoding: Gray code
- Reward function: sum of the values of merged tiles

Experiment 1



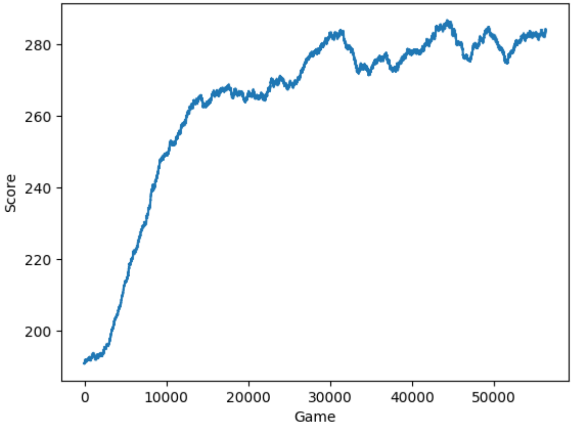
No learning progress achieved

Experiment 2

- Input encoding: Gray code
- Reward function:

$$R(s, s') = \begin{cases} -1, & \text{if } s' \text{ is a terminal state} \\ \min(\#\mathit{tilesMerged}(s, s')/4, 1), & \text{otherwise} \end{cases}$$

Experiment 2



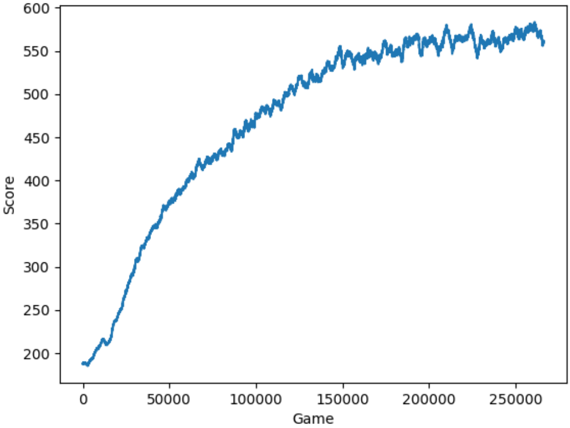
Visible improvement, although not very stable learning progress

Experiment 3

- Input encoding: Normalized board encoding
- Reward function:

$$R(s, s') = \begin{cases} -1, & \text{if } s' \text{ is a terminal state} \\ 1, & \text{if } R^{'}(s, s') \geq 8 \\ R^{'}(s, s')/60, & \text{otherwise} \end{cases}$$

Experiment 3



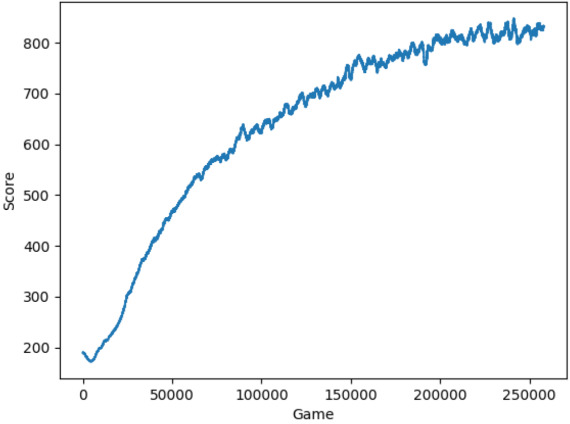
The training is stable and saturated

Experiment 4

- Input encoding: Normalized board encoding
- Reward function:

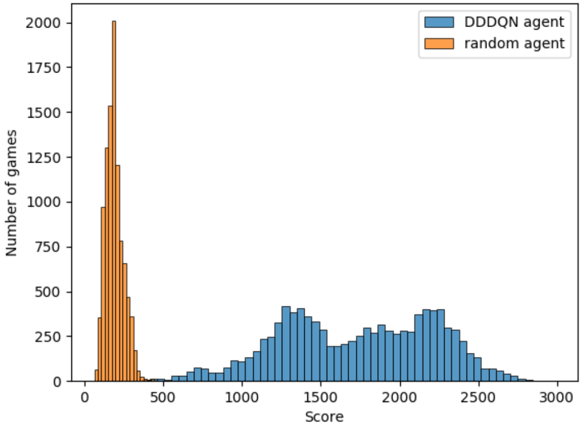
$$R(s, s') = \begin{cases} 1, & \text{if obtained 2048 tile} \\ (\#\mathit{tilesMerged}(s, s') - 1)/8, & \text{otherwise} \end{cases}$$

Experiment 4



Best result so far

Best agent vs. random agent



Distributions of scores of 10000 testing games

Summary

- We designed and implemented RL agent that plays 2048 game
- We had to modify the rewards to achieve better performance
- The best agent achieved the 2048 tile in about 7% of testing games
- Better results can be obtained by using distributional or asynchronous learning methods

End of presentation

Thank you for your attention
Please, feel free to ask questions