Mancala

Viktor Kosin und Johanna Beier

04.09.2020

Introduction

Mancala

- ancient two player game
- **as vector:** $[6, 6, 6, 6, 6, 6, |6, 6, 6, 6, 6, 6, |0, 0]$
- **Goal:** catch more then half of the beans (37)

Mancala Rules

- collect all beans of a hole and drop one in each clockwise following hole
- catch all beans of the last hole, if it contains 6, 4 or 2 beans
- going backwards: collect beans from all following holes with 6, 4 or 2 beans, if there are no other holes in between
- game ends if either one player has no more beans or one player catches at least 37 beans
- total sum of beans: catched beans + beans on own side

MDP

- Mancala can be represented as a Marcov Decision Process (MDP)
- set of states S, set of actions per state A, action a $\in$ A
- How does the Mancala agent learn to choose the best action?

Reinforcement Learning

- **Idea:** reward or punish some action
- **Goal of agent:** maximize total reward
- **here:** Small reward for catching beans, bigger reward for winning the game
- use Q-Learning

Q-learning

- small state space: Q-table
- replace Q-table by Q-function

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

- agents often need to learn actions that do not lead immediately to a reward
- allow a small amount of random actions (exploration rate)

Netz

bild netz, dass wir verwenden

Netz

- **activation function:** Sigmoidfunction
- learningrate for the update-weights-function

Backpropagtion

1. **Step** Generate training data: for a given input set an expected output (e.g. with Q-function)
2. **Step** Calculate for the input $a^{x,1}$:
   - activation $a^{x,l}$ of layer $l = 2, 3, ..., L$ by
   $$a^{x,l} = \sigma(w^l a^{x,l-1} + b^l)$$
   - Output error $\delta^{x,L}$
   - Backpropagate error to each layer: $\delta^{x,l}$
3. Step Use error of each layer to update weights and biases

play

Training

- let two agents play against each other and save pairs of actions and rewards
- update for each boardstate and action the underlying Q-function
- save each board state and dedicated Q-values as training data
- feedforward a boardstate to the net
- loss $=$ output $-$ Q-values

Results