Reinforcement Learning using the example of Mancala

Viktor Kosin und Johanna Beier

04.09.2020

Structure

- Mancala rules
- Q-learning
- Backpropagation
- Network
- Training
- Troubleshooting
- Results / Improvement

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

- choose non empty hole
- 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
- finite but very large number of states
- How does the Mancala agent learns to choose the best action?

Reinforcement Learning

- **Idea:** reward or punish some action
- **Goal of agent:** maximize total reward
- **possible rewards for Mancala:** Small reward for catching beans, bigger reward for winning the game, punish illegal actions and loosing
- use Q-Learning

Q-learning

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

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

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

Netz

- **activation function:** Sigmoidfunction
- **loss function:** Meanvalue

play

- choose action by feeding the current board to the net and take the argmax of the output
- get board and reward after action
- append board and reward to trainingslists
- change player (turn board)

Rewards and Discount

- choosing the right rewards is key for the Q-function to work properly
  - too high rewards lead to large q-value estimations (maybe higher than the activation function would allow)
    $\Rightarrow$ the network is going to increase q-values with every iteration
  - too low rewards lead to q-values close to zero
  - get 0.05 as reward for every caught bean and 10.0 for a winning action
- selecting the right discount and learning rate is difficult and depends on the reward strategy

Training

- let the agent play against itself and save pairs of actions and rewards
- update for each board state and action the underlying Q-function
- save each board state and dedicated Q-values as training data
- feedforward a boardstate to the net

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

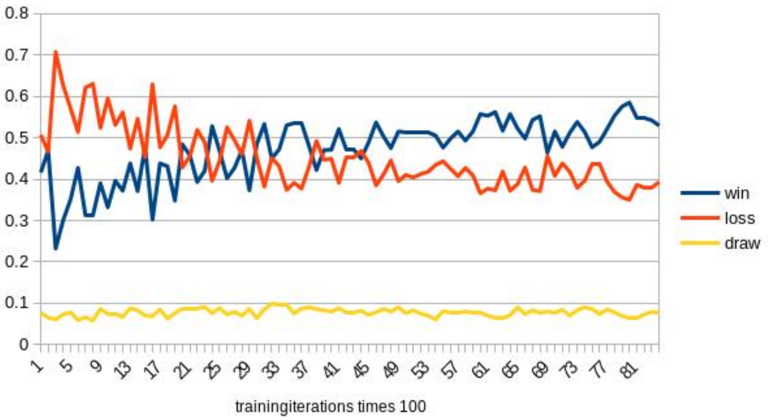
     $$a^{x,l} = \sigma(z^{x,l}), \quad z^{x,l} = w^l a^{x,l-1} + b^l$$

   - Output error $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$
   - Backpropagate error to each layer:

     $$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,L})$$

3. **Step** Use error of each layer to update weights and biases

Results for a straightforward adaptation



- simple q-learning tends to stagnate very early, gets worse after some time
- a 'large' learning rate doesn't always improve the rate

Origins of problems

- couldn't identify any coherence
- to see how the network learns Mancala, we need to simplify the game

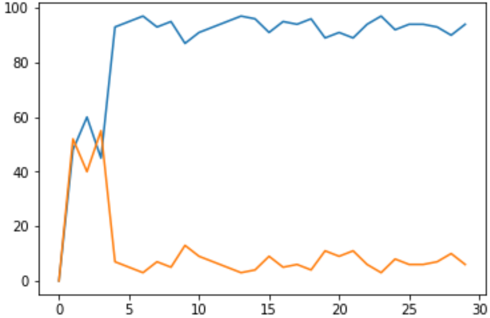
simple game

- $[1, 1, 5, 1, 1, 1|1, 1, 5, 1, 1, 1|0, 0]$
- play against random
- network ist startplayer
- obvious way of playing at least tied
- after some trainingiterations the second player does not win at all
- $\rightarrow$ network is ok

simple board

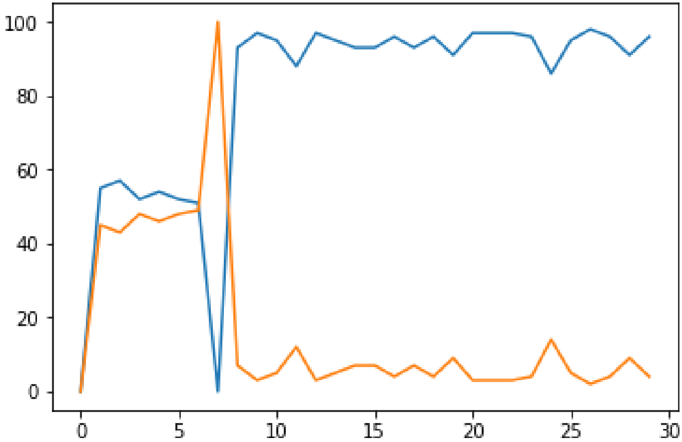
- reduce the board size: $[2,2|2,2|0,0]$
- obvious way of loosing the game: choose first hole
- compare guessed Q-Values with choice of hole
- Q-function decides for wrong side $\rightarrow$ solve this error
- net performs good, if it starts in the right direction
- use flexible exploration rate

simple board results



- 1 hidden layer with 10 neurons
- 1 Unit = 100 Trainingiterations

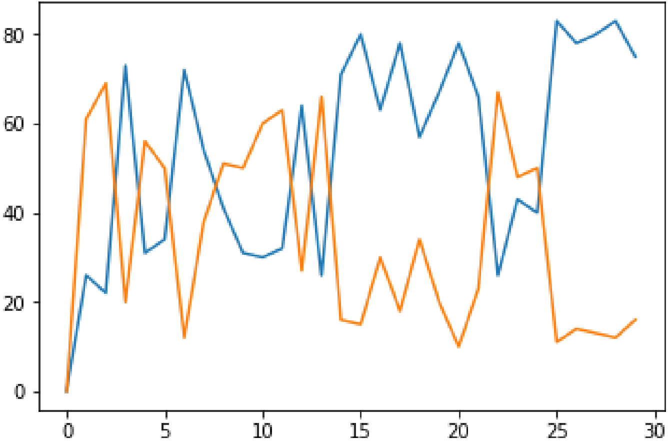
simple board results: jump



Mancala with 4 beans per hole

- $[4, 4, 4, 4, 4, 4|4, 4, 4, 4, 4, 4|0, 0]$
- transfer findings to this game (flexible exploration rate, ...)
- learns something but oscillates
- learns better if starts in the right direction

Mancala with 4 beans per hole



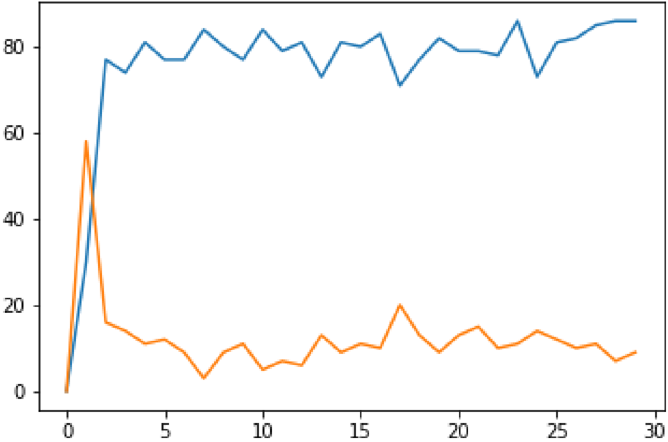
improved Mancala with 4 beans per hole

- Idea: reduce learining rate if net performs good otherwise increase learning rate
- reward only winning not catching beans

flexible parameters

```
if Spieler1gewonnen > 80:
    l=0.01
    ma.a = ma.a/10
    ma.exploration_rate = 0
elif Spieler1gewonnen >70:
    l=0.01
    ma.a = ma.a/10
    ma.exploration_rate = 0.1
else:
    l=1
    ma.a = ma.a+0.1
    ma.exploration_rate = 0.3
```

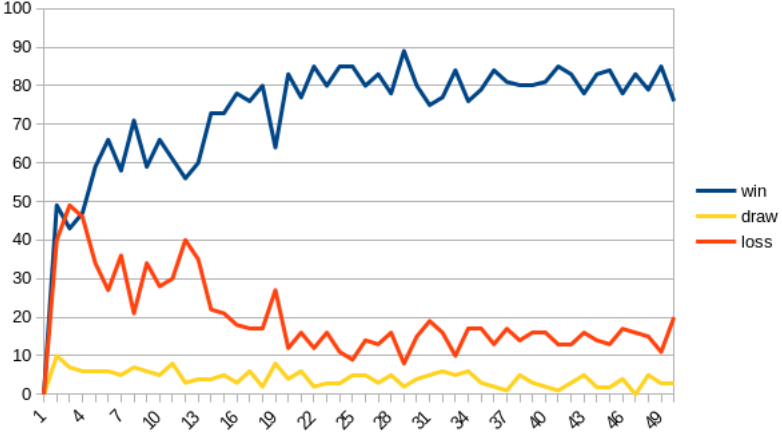
improved Mancala with 4 beans per hole



- keep in mind: Mancala ist still more complex and will possible take more time to converge

Adapting learning of Mancala

- generating a random network of size $[14, 10, 6]$



Possible improvements

- use multiple network to guess optimal parameters of q-function and network, e.g:
  - exploration rate, rewards
  - discount, learning rate
- use tree search algorithm to guess rewards over multiple future actions

Approved agents of other Mancala like games

- MinMax
- Monte Carlo Tree Search
- Asynchronous Advantage Actor-Critic
  - runs parallel agents
  - 3 networks work in concert: input, action and critic

Quellen

- neuralnetworksanddeeplearning.com/chap2.html
- towardsdatascience.com/the-ancient-game-and-the-ai-d7704bea280d
- https://1tr7g949k6uv1s2cx31wn2yt-wpengine.netdna-ssl.com/wp-content/uploads/2017/02/handmad