Reinforcement Learning for playing Connect Four

Simon Hölck, Florian Cimander, Tim Löhr

Mathematical Data Science

August 24, 2020

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

NATURWISSENSCHAFTLICHE
FAKULTÄT

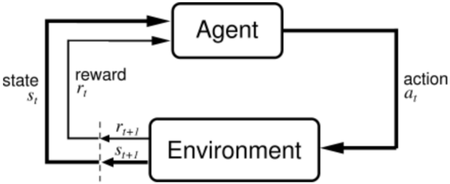
Table of Content

Reinforcement Learning

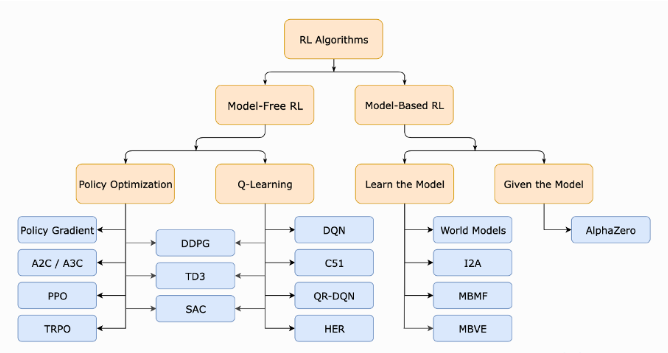
Reinforcement Learning - Terminologies

Markov Decision Process (MDP)

- Agent
- Environment
- Action
- State
- Reward

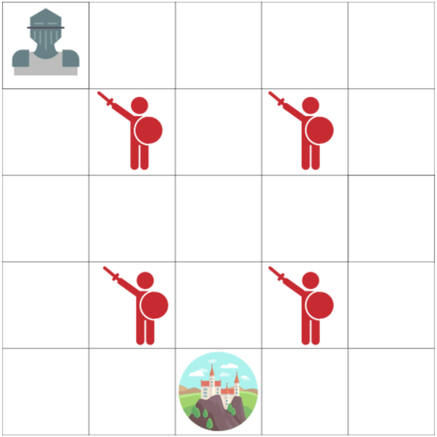


Overview of Reinforcement Learning Algorithms

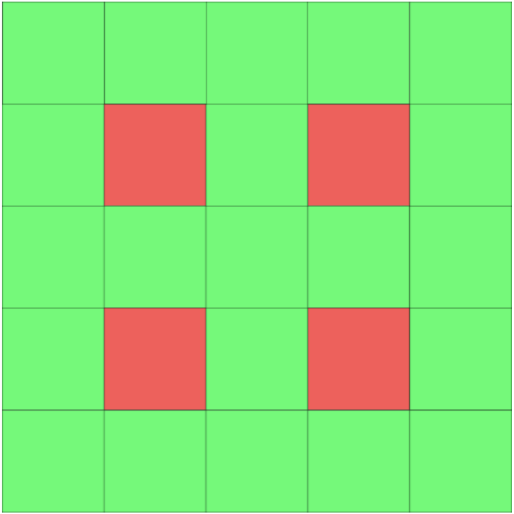


Q Learning

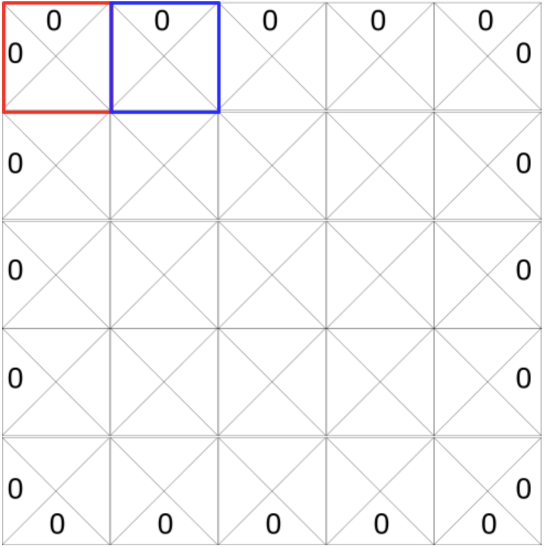
Q Learning



Q Learning



Q Learning



Q Learning

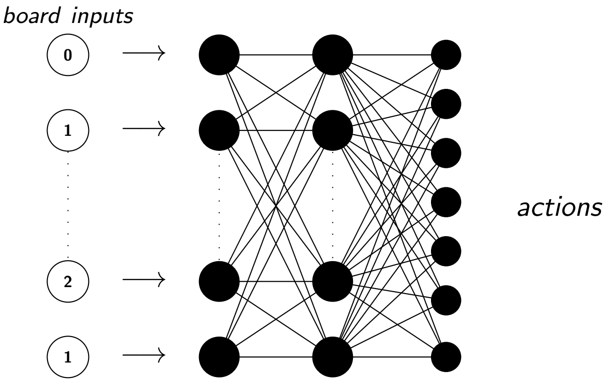
Q Learning is a value-based RL algorithm

$$\underbrace{\boxed{Q^{\pi}(s_t, a_t)}}_{\text{Q-Values for the state given a particular state}} = \underbrace{\boxed{\underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots}}_{\text{Expected discounted cumulative reward}} \underbrace{\boxed{s_t, a_t}}_{\text{Given the state and action}}]$$

Deep Q Learning

Deep Q Learning

- How can we use the basics of Q Learning without having to store a q-table?
  → universal approximation theorem
  → use neural network as approximation to q function



Implementation of a Deep Q Learning Agent

1. **Neural Network**
   - used eight fully connected layers to have enough depth for good approximation
   - added three dropout layers to prevent overfitting
   - used relu as activation function in forward pass
2. **Exploration vs. Exploitation**
   - Exploration: Agent makes out of character decisions that are not given by the network (random)
   - Exploitation: Agent takes actions given by the network

$\rightarrow$ We use *epsilon decay* to find the right balance between the two

Implementation of a Deep Q Learning Agent

3. **Learning Process**
   - Two problems
     a) correlated inputs/outputs
     b) non-stationarity

Implementation of a Deep Q Learning Agent

3. **Learning Process**
   - for every action during a game the following quintuple is stored to the agents *experience replay memory*

     (state, action, reward, new state, done)
   - rewards are determined when game is finished
   - the agent learns the saved transitions in batches that are chosen randomly from its memory

   $\rightarrow$ solves the problem of correlated inputs/outputs

Implementation of a Deep Q Learning Agent

3. **Learning Process**
   - for each transition in the batch we perform the following steps

calculate forward pass of initial state + **calculate target value using the q formula**

$$q_{target} = reward + \gamma \cdot max_{a'} Q(s', a')$$

↓

calculate mean squared error

↓

perform backpropagation and apply calculated gradients

→ problem of non-stationarity

Implementation of a Deep Q Learning Agent

3. **Learning Process**
   - we deal with this problem by using a *target network*
   - the target network is an old version of the underlying DQN network
   - it gets updated very infrequently

calculate forward pass of initial state + **calculate target value using the q formula**

$$q_{target} = reward + \gamma \cdot max_{a'} Q_{target}(s', a')$$

$\downarrow$

calculate mean squared error

$\downarrow$

perform backpropagation and apply calculated gradients

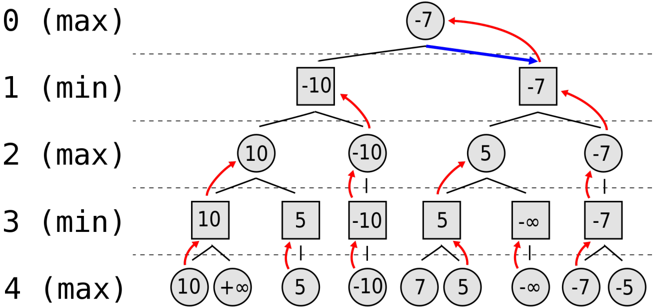
$\rightarrow$ targets stay constant and problem becomes more stationary

Training

Simon Hölck, Florian Cimander, Tim Löhr

Opponent: Minimax

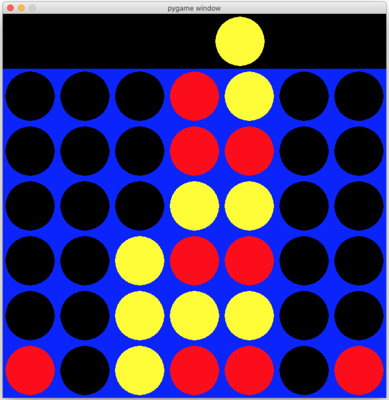
Self-play RL like Alpha Zero is too time consuming.
Solution: Let the Neural Network practice against Minimax



Minimax depth = 1 vs Neural Network

- **Yellow**: Minimax
- **Red**: Neural Network

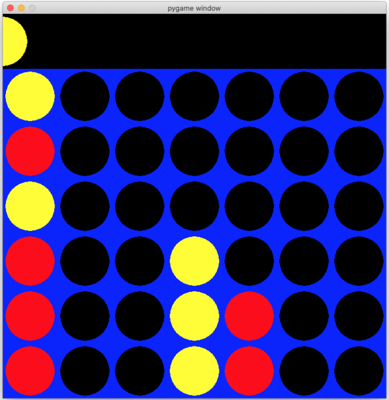
7000 Episodes and Neural Network win-ratio of 11%



Minimax depth = 1 vs Neural Network

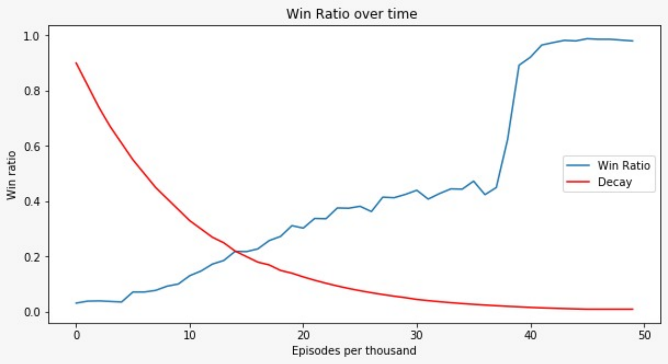
- **Yellow**: Neural Network
- **Red**: Minimax

50000 Episodes and Neural Network win-ratio of 99%



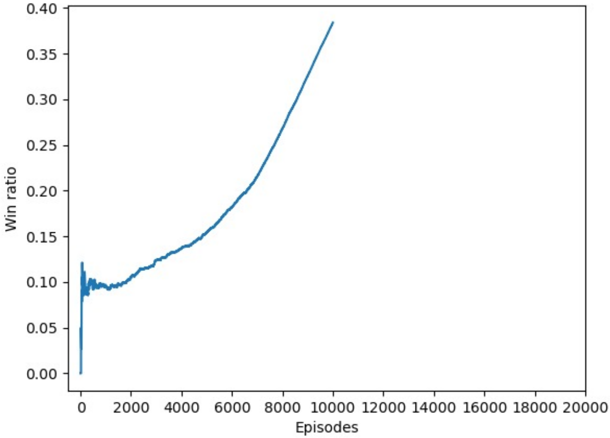
Learning over time

We started by letting the Neural Network train **50000** Episodes against Minimax with depth $= 1$



Save the trained model and start all-over again

Now this 99% ratio Neural Network plays and practices against the Minimax with depth $= 2$



Best Practices

Best Practices

Trial and Error, Error ... and more Errors

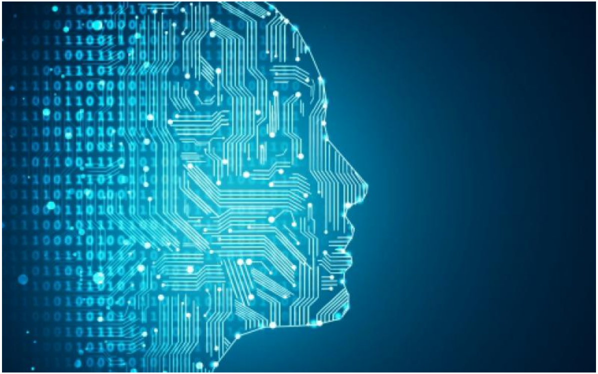
Best set of Hyperparameters so far:

- Randomness 100% with exponential fall down to fixed 1%
- Batch Size = 128
- Memory size = 50000
- Learning Rate for the Neural Network = 0.01
- Episodes 50000 on depth = 1: *15 hours*
- Episodes 10000 on depth = 2: *10 hours*

Demo

Demo

It is time to show a Demo in the GUI



Discussion

Discussion

- Does a 100% win ratio against Minimax of depth 5 or 6 plays good against human opponents?
- Can the Neural Network generalize better if we would have used the Monte Carlo Tree Search (MCTS) as opponent instead of the Minimax?
- Could a ResNet train a better AI against human players with everything else kept the same than our very basic Neural Network architecture?

Thanks for listening to us!

Sources

- Figure 1: https://www.pngwave.com/png-clip-art-jdcjo
- Figure 2: Reinforcement Learning: An Introduction (Sutton, Barto)
- Figure 3: https://www.afcea.org/content/artificial-intelligence-will-change-human-values
- Figures Knight & Princess Game: https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/