

# Blackjack

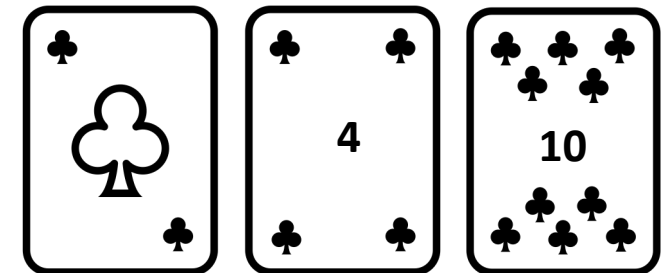
Zwischenpräsentation Semesterarbeit Reinforcement Learning (ReInf)

Lukas Nydegger & Damaris Zosso, Zwischenpräsentation Semesterarbeit ReInf, 17.10.23

# Inhalt



- Blackjack allgemeine Informationen
- Random Agent
- Q-Agent
- Neat (Genetischer Algorithmus)
- Ausblick

# Blackjack



# Blackjack



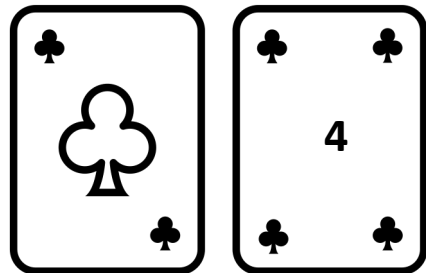
- Teil von Toy Text Umgebungen von Gymnasium
  - Import mit: `gymnasium.make(„Blackjack-v1“)`
- 2 Handlungen:
  - hit
  - stick
- Observation Space:
  - Summe der Handkarten
  - aufgedeckte Karte Dealer
  - brauchbares Ass (  /  )

# Blackjack



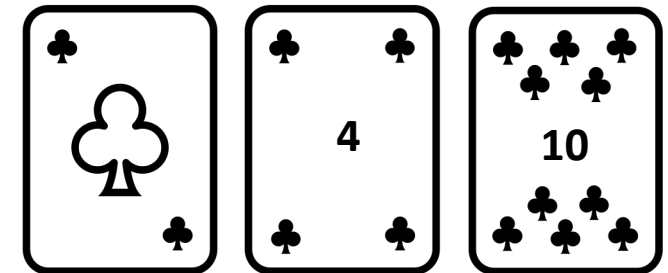
**brauchbares Ass:**

- Observation Space:
  - ( 15, 10, 1 )



**unbrauchbares Ass:**

- Observation Space:
  - ( 15, 10, 0 )



# Blackjack



- Rewards
  - Gewinnen = 1
  - Unentschieden = 0
  - Verlieren = -1
- unendliches Kartenset (zurücklegen)

```
# 1 = Ace, 2-10 = Number cards, Jack/Queen/King = 10  
deck = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
```

- maximale Gewinnchance ca. 42,5%

# Random Agent

- Wählt zufällig Aktionen aus.
- Keine Policy
- Vergleichswert von  $\sim 28\%$  bei 100'000 Iterationen

# Random Agent

```
class RandomAgent:

    def __init__(self, env):
        self.env = env

    def get_action(self, state) -> int:

        return self.env.action_space.sample()

agent = RandomAgent(env)
```



# Q-Agent

- Q-Table mit Einträgen für State-Action Paare

```
self.q_values = defaultdict(lambda: np.zeros(env.action_space.n))
```

- Aktion wählen (3 Exploration Policies ausprobiert)
- Aktion ausführen
  - Reward und neuen State beobachten
  - Q-Table updaten

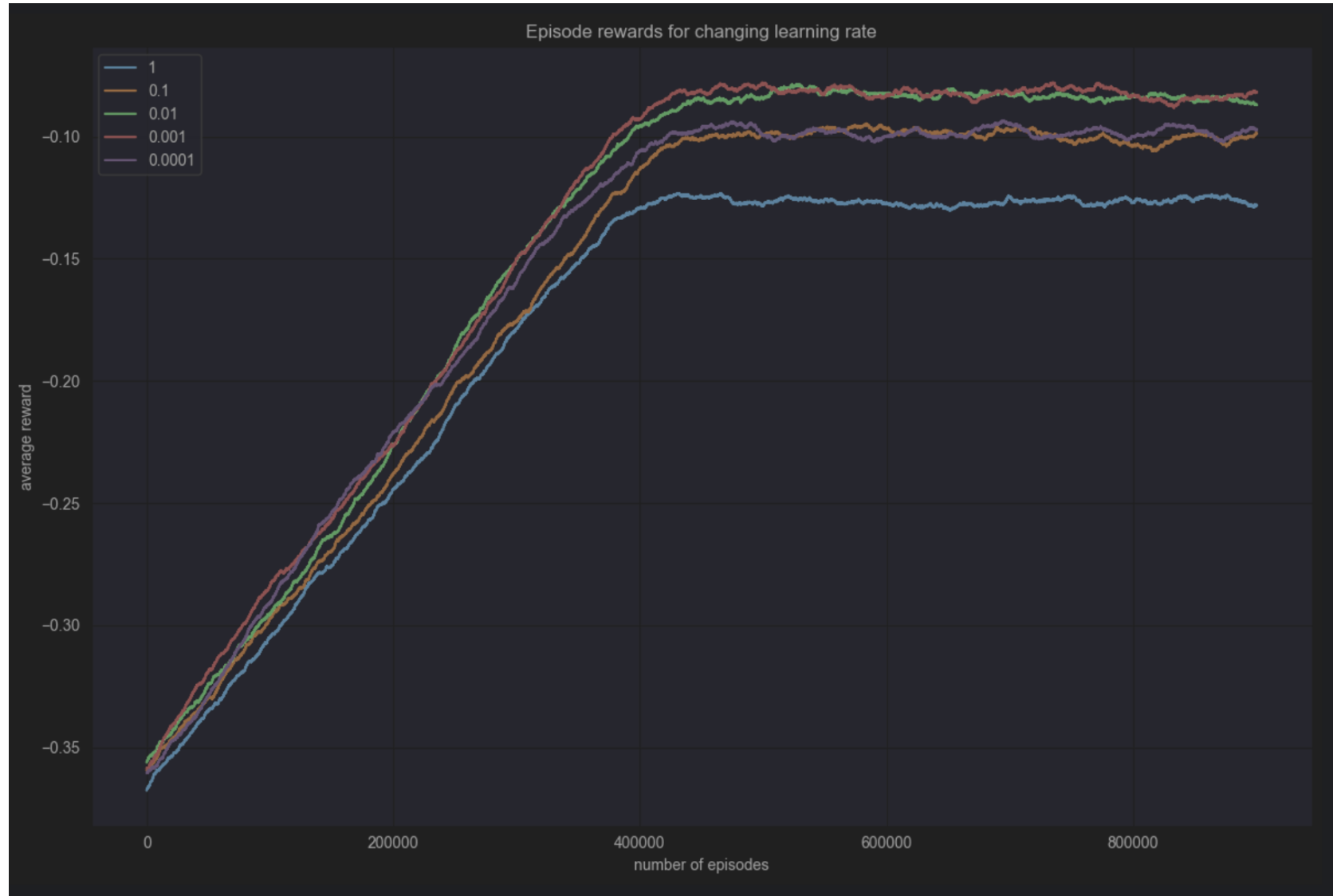
# Q-Agent

$$\blacksquare Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

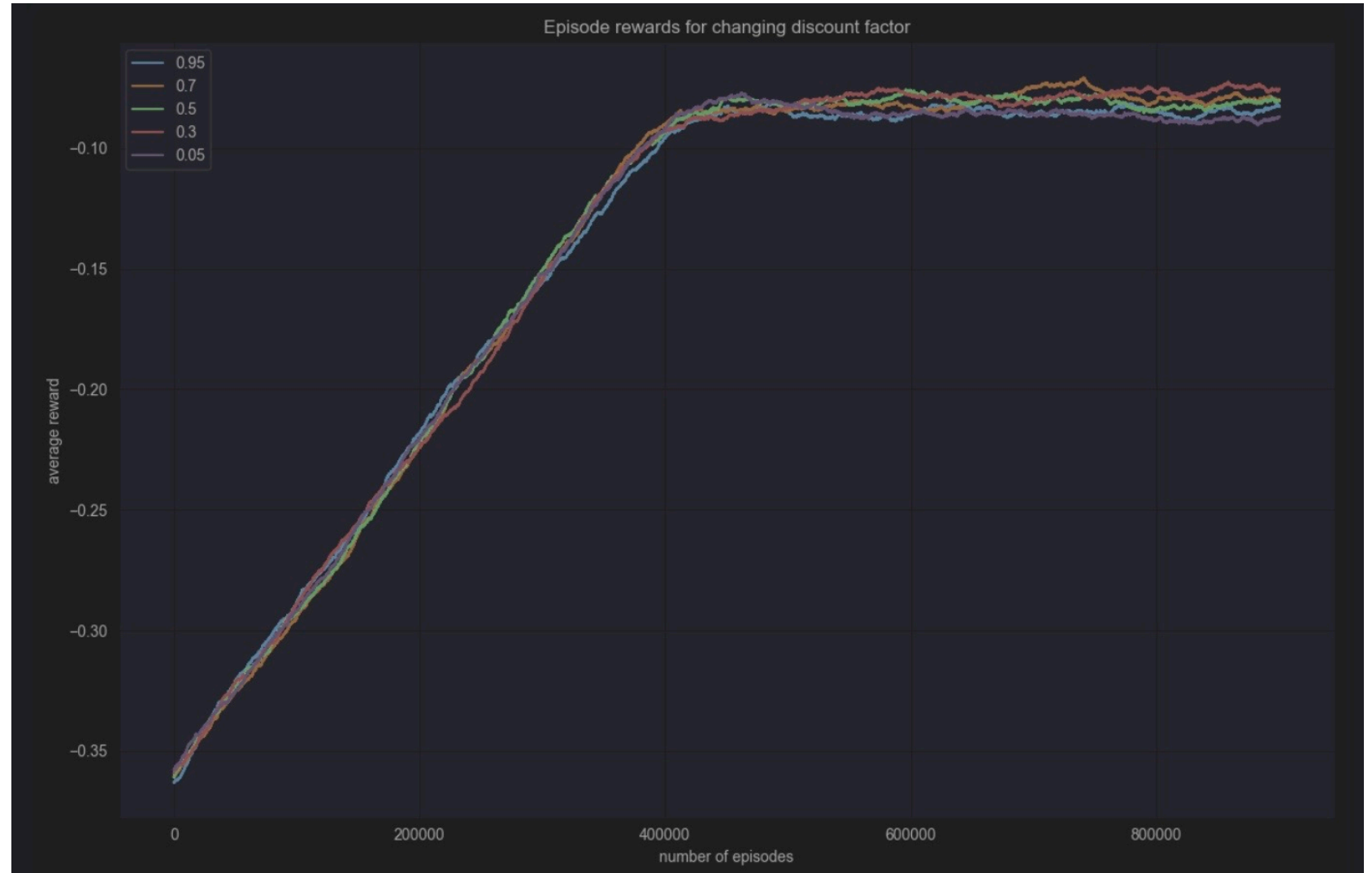
```
# Update Q-value of an action
old_values = self.q_values[state][action]
next_max_values = (not terminated) * np.max(self.q_values[next_state])
temporal_difference = (
    reward + self.gamma * next_max_values - old_values
)
new_values = old_values + self.alpha * temporal_difference

# update the q_values
self.q_values[state][action] = new_values
```

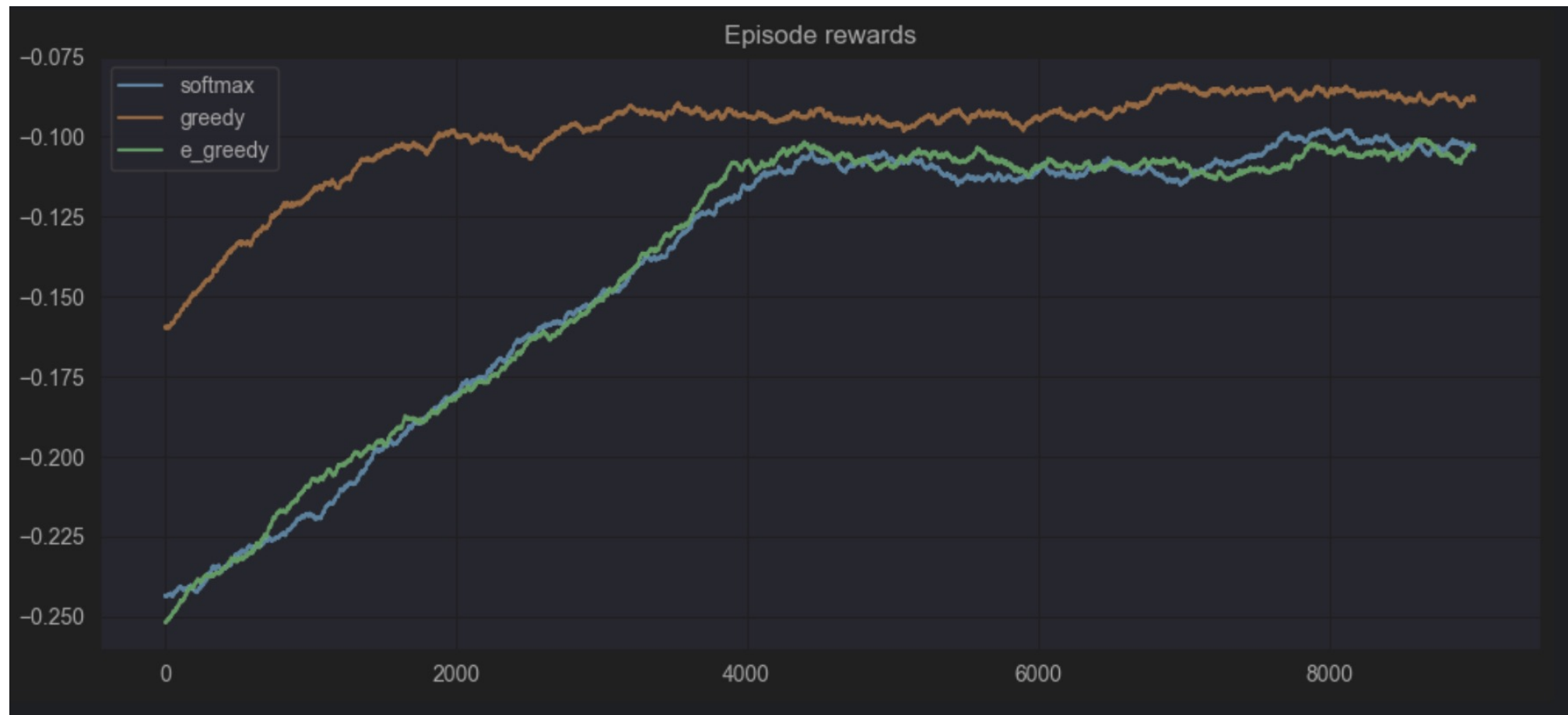
# Q-Agent



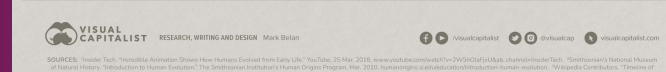
# Q-Agent



# Q-Agent

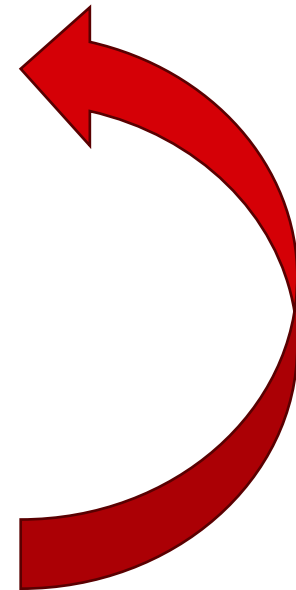


- Genetischer Algorithmus
- Kenneth Stanley and Risto Miikkulainen
- Biomimikry
- Evolution durch Selektion und Mutation



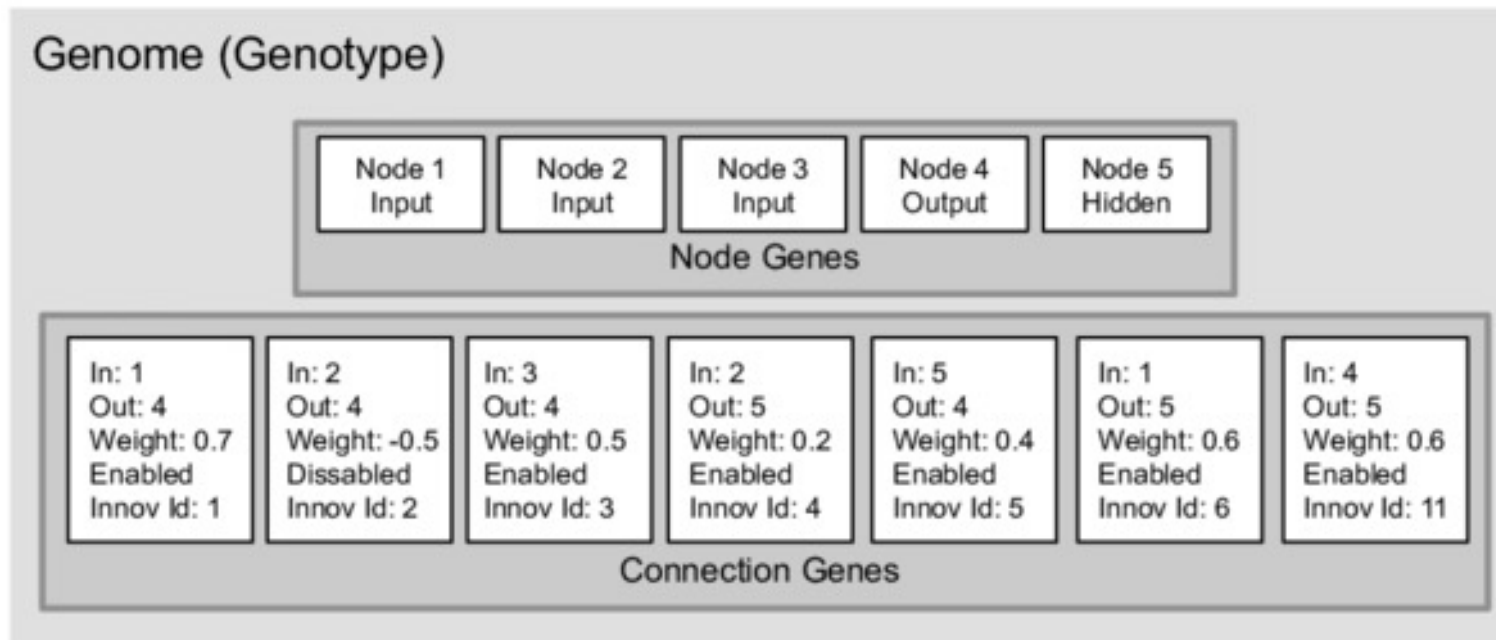
# Neat-Idee

1. Definieren von Population
2. Zufällige minimale Netzwerke
3. Simulation und Fitness bestimmen
4. Besten kommen weiter
5. Repopulation
6. Mutation: Hinzufügen, Wegnehmen oder Anpassen

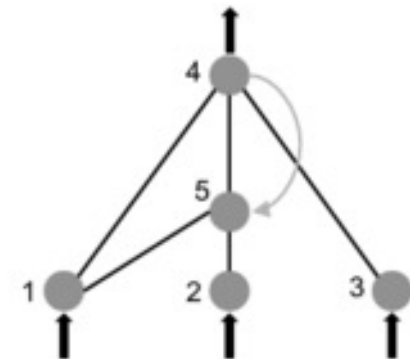


# Neat-Theorie

## ■ Codierung von Netzwerken

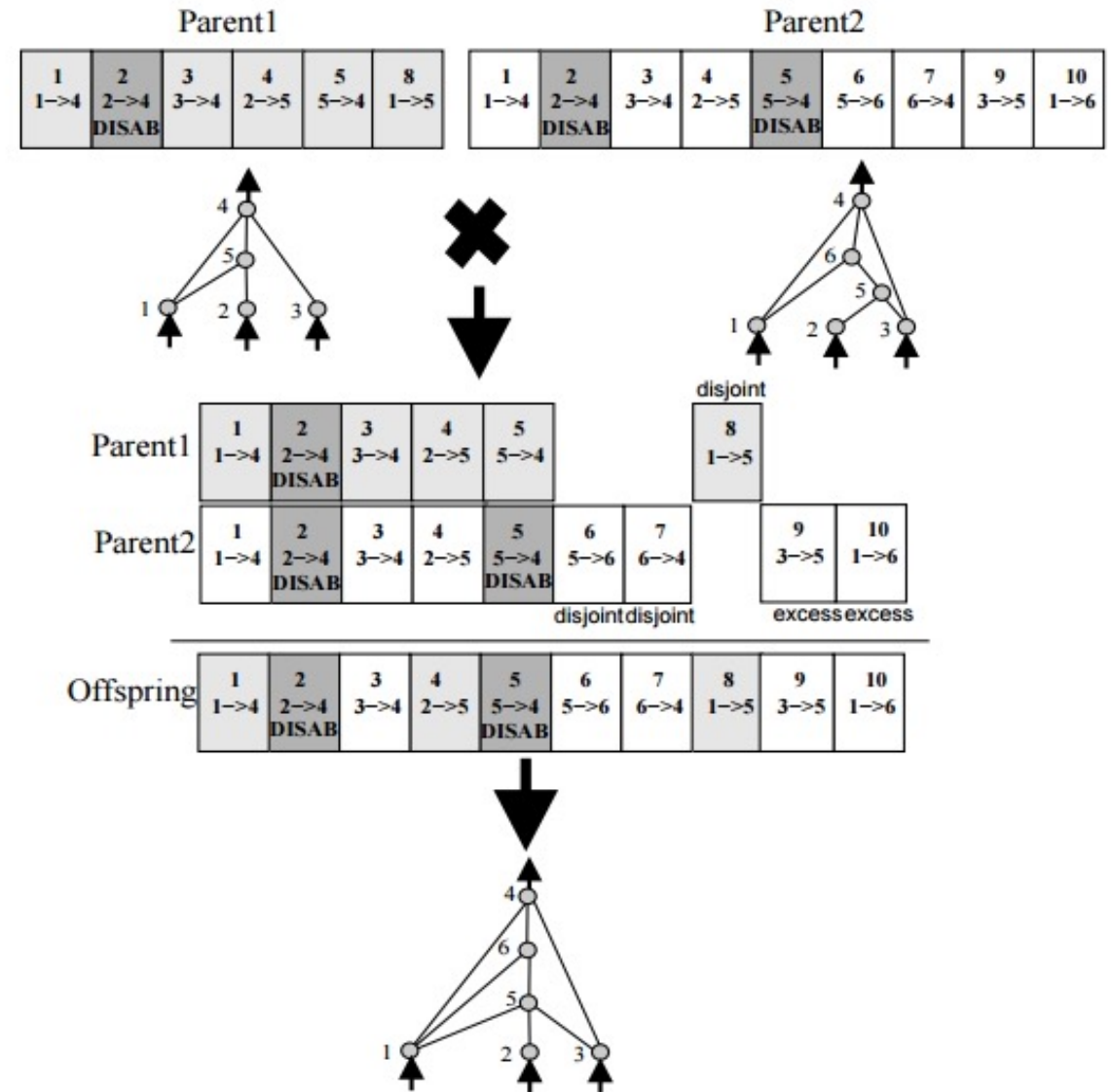
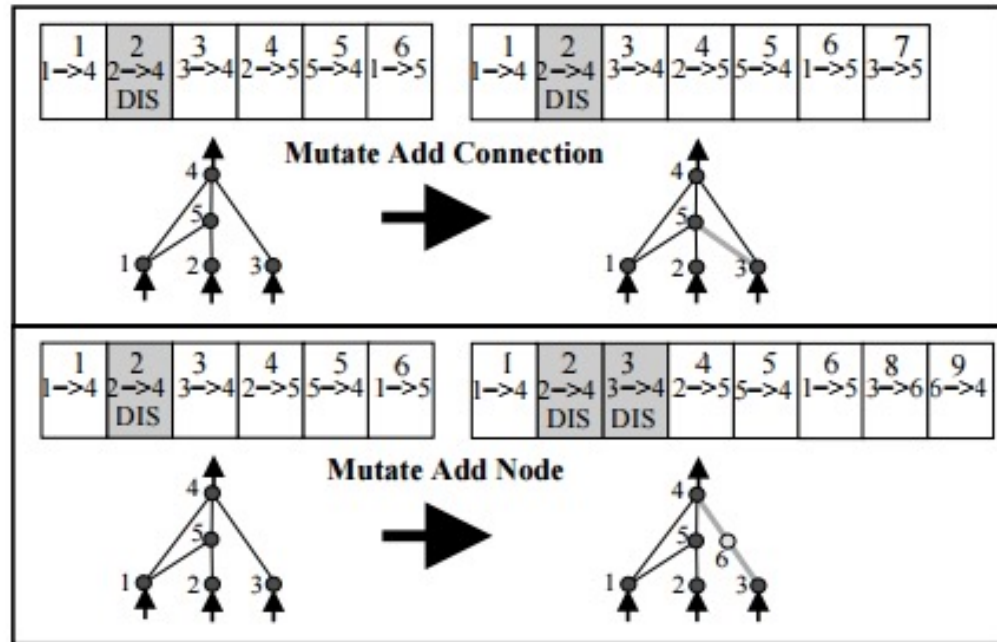


Network (Phenotype)





# Neat-Theorie



# Neat-Implementation

- NEAT-Python Library
    - Konfiguration durch Hyperparameter
    - Übernimmt Mutation, Selektion, Species bestimmung...
  - Evaluation der Fitness
    - 100 Spiele
    - Summe des Rewards = Fitness
    - Hit wird zusätzlich belohnt
  - Definieren der Hyperparameter
- Live Demo der Konfiguration und Simulation

# Ausblick

- Verbesserung Hyperparameter
- Reward anpassen
- 42% erreichen

# Literatur

- T. Vos und D. M. Sabatelli, «Reinforcement Learning and Evolutionary algorithms in the Stochastic Environment of Blackjack».
- K. O. Stanley und R. Miikkulainen, «Evolving Neural Networks through Augmenting Topologies», *Evol. Comput.*, Bd. 10, Nr. 2, S. 99–127, Juni 2002, doi: 10.1162/106365602320169811.