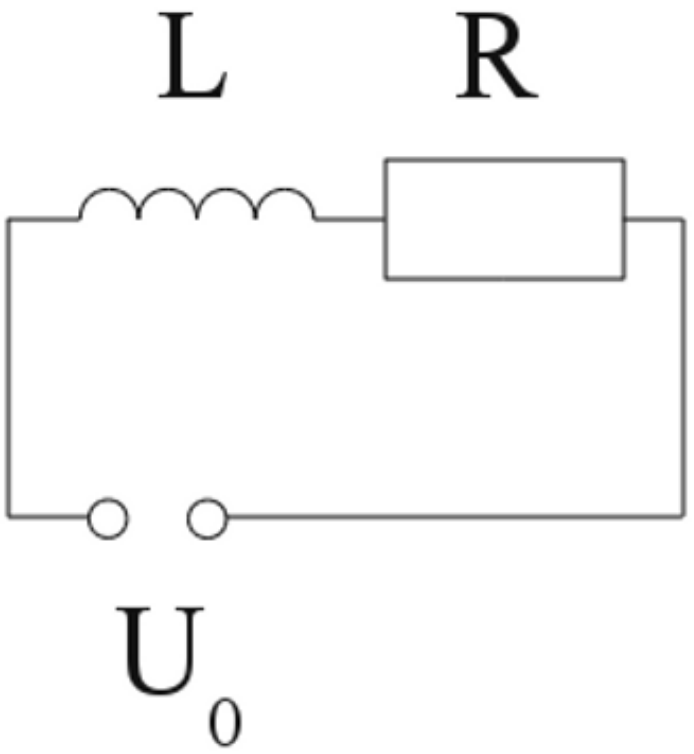
Einsatz von Reinforcement Learning am Beispiel einer Stromregelung durch eine RL-Last

Oktober 2024

Stromkreis mit resistiver und induktiver Last

Spule:

- Strom erzeugt Magnetfeld (→ Selbstinduktion)
- Phasenverschiebung bei Wechselstrom: Strom gegenüber Spannung verzögert



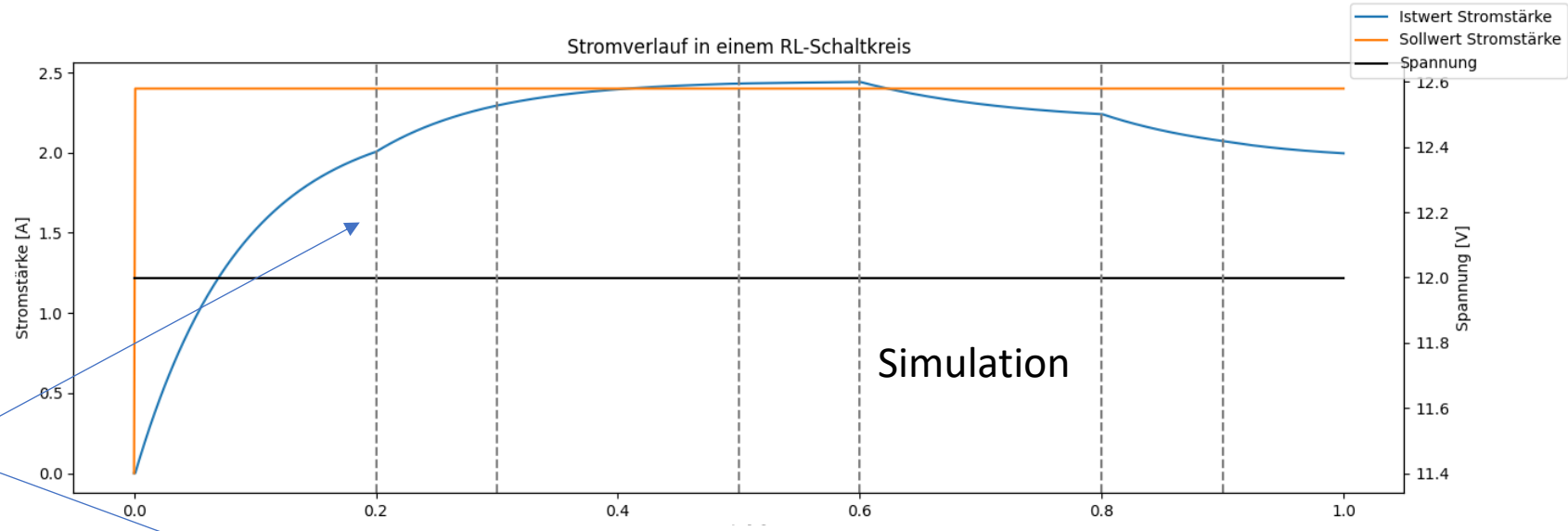
Widerstand:

- Umwandlung von Strom in Wärme (z.B. elektrische Heizung)
- keine Phasenverschiebung bei Wechselstrom

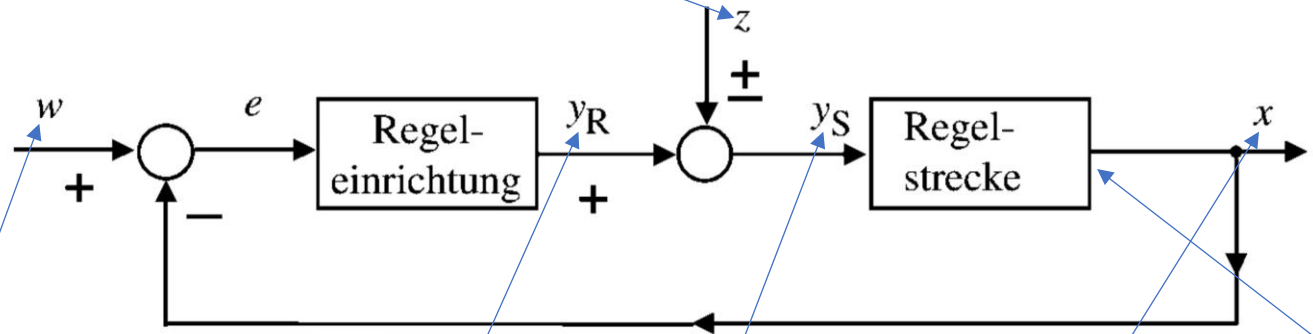
Ziel: Stromregelung durch Anpassung der Spannung

- konstanter Gleichstrom (z.B. für Erzeugung von stabilem Magnetfeld)
- stabiler Wechselstrom (z.B. für Induktion in Elektromotor)

Regelkreis



Störgrößen (hier: zeitliche Änderungen von R und L)



return: -320

$$\sum_t -\left|I(t) - I_{ref}(t)\right|$$

Führungsgröße I_{ref}
- konstanter Gleichstrom
- stabiler Wechselstrom

$\Delta U(t)$

Stellgröße

Regelgröße

RL-Last: PT_1-Glied

$U_0 + \Delta U(t)$

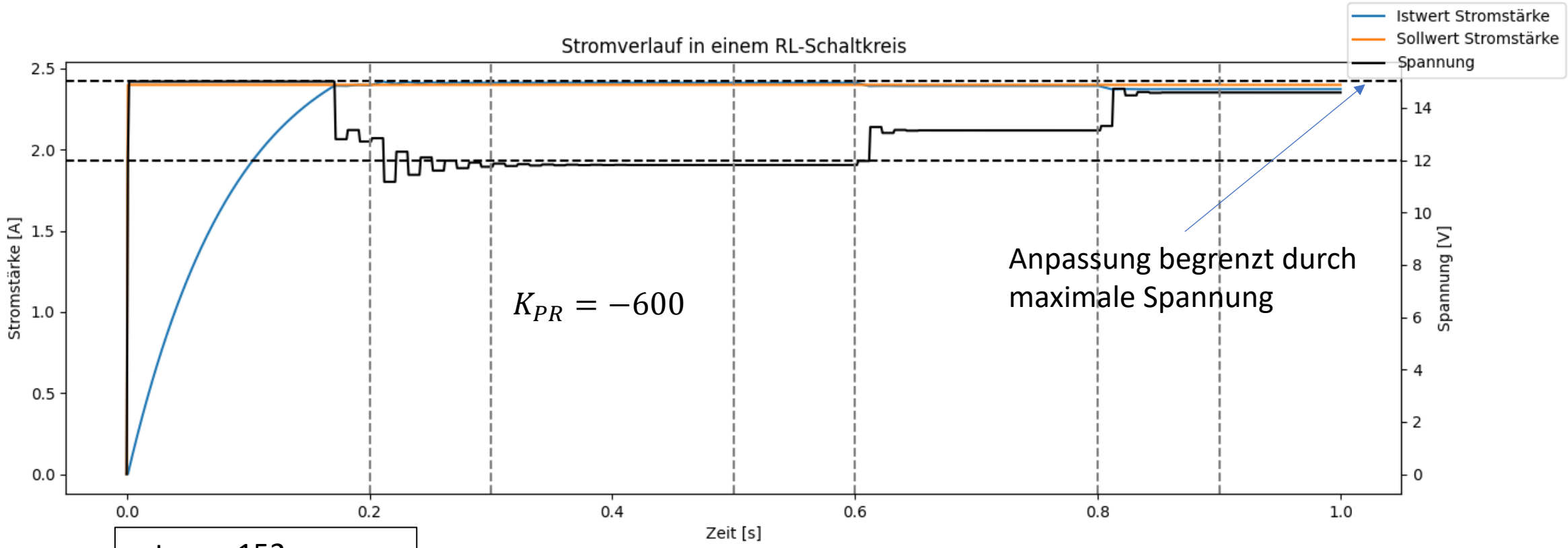
Differenzialgleichung:

$$U(t) = R \cdot I(t) + L \cdot \frac{dI(t)}{dt}$$

Benchmarks

PID-Regler mit Begrenzung

PID: $y_R(t) = K_{PR} \cdot e(t) + K_I \cdot \int e(t)\,dt + K_D \cdot \frac{de(t)}{dt}$



return: -152
Messung alle 0,001 s
Regelung alle 0,01 s

Model Predictive Control (MPC)

Optimal-Control Methode

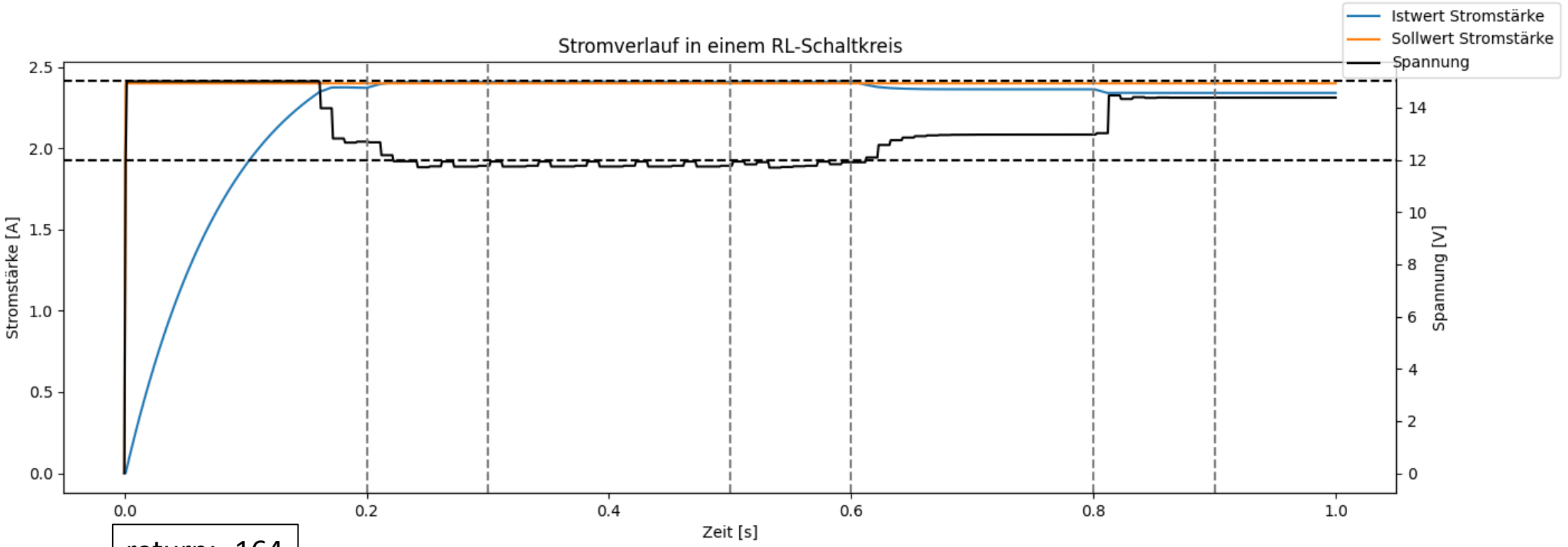
1. Benutzung von Modell des Systems, um künftige Entwicklung zu prognostizieren (potentiell mittels Machine Learning)

   hier: $I(t+1) = I(t) \cdot \frac{dI(t)}{dt} \cdot \Delta t = I(t) \cdot \frac{U(t) - R \cdot I(t)}{L} \cdot \Delta t$

2. Optimierung: Anpassung der Stellgröße ($U(t)$) zur Minimierung der Abweichung der Regelgröße (z.B. $\left(I(t) - I_{ref}(t)\right)^2$) über mehrere Schritte
3. Ausführung nur des ersten Schrittes, dann Wiederholung des Prozesses

im Gegensatz zu Reinforcement Learning Kenntnis des Modells nötig

MPC Ergebnisse

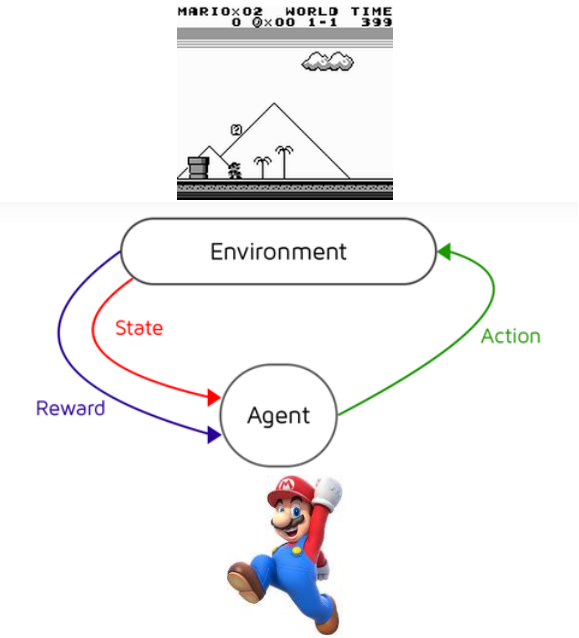


return: -164

Modell berücksichtigt Störungen nicht.

Reinforcement Learning

Reinforcement Learning



State: Größen zur Beschreibung des aktuellen Systemzustands

hier: $I(t), U(t)$

Action: Entscheidung des Agenten in einem bestimmten Zustand

hier: $\Delta U(t)$

Reward: numerische Antwort der Umgebung auf eine Action

hier: $-\left|I(t) - I_{ref}(t)\right|$

Action Policy: Strategie bestehend aus Aktionen für jeden Zustand (Action-Sequenz)
Ziel: Policy zur Maximierung der Summe der künftigen (**verzögerten**) Rewards

Lernen durch **zielgerichtete Interaktion** mit der Umgebung (**trial and error**)
→ weder Input-Output Beispiele (supervised learning) noch Modell der Umgebung nötig

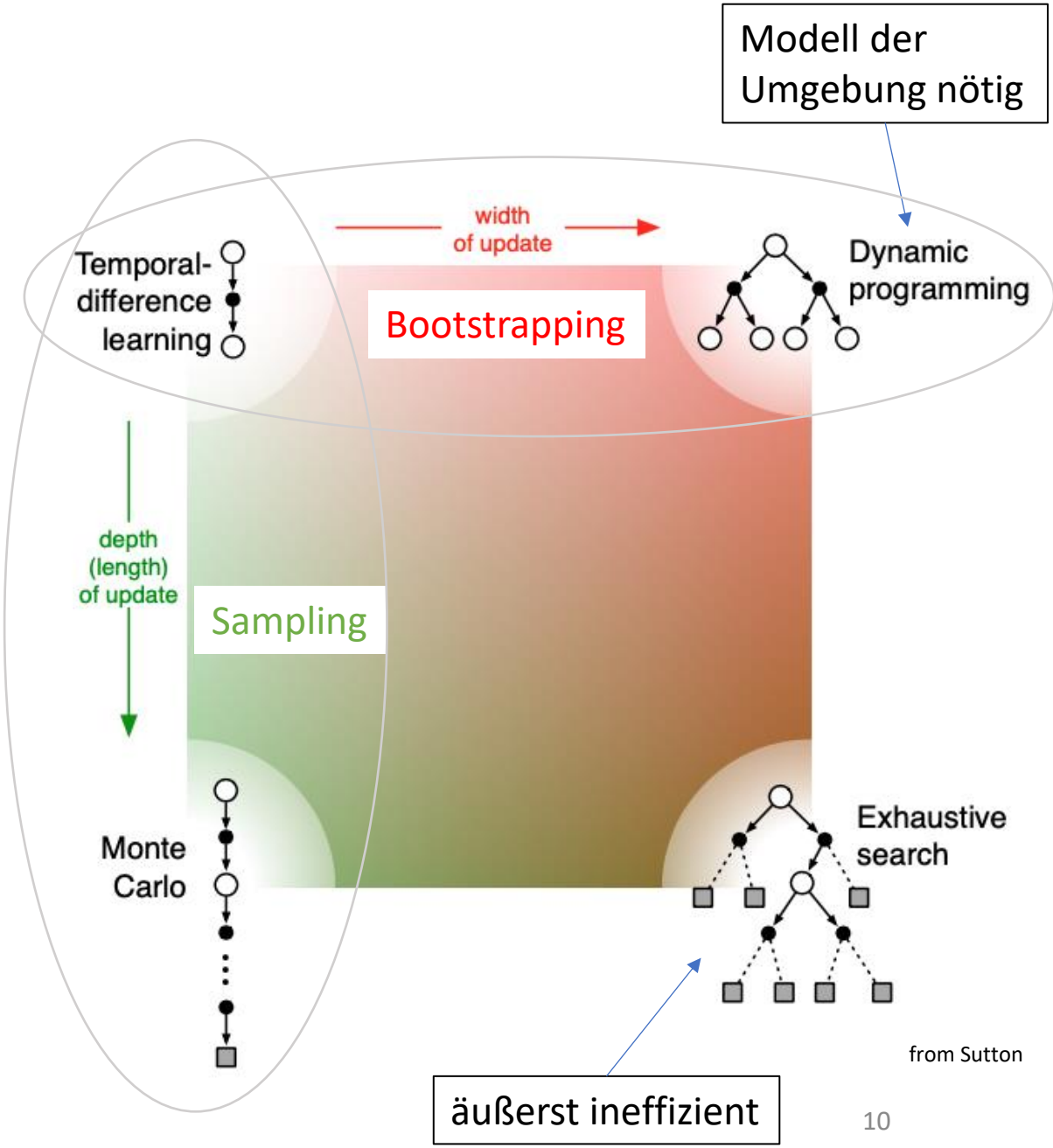
Action-Policy Suche

Methoden in Reinforcement Learning:

Sampling: (zufällige) Stichproben von Aktions-Sequenzen

- Monte Carlo: vollständige Episode
- TD learning: nur nächster Schritt

Bootstrapping: Abschätzung des Wertes des aktuellen Zustands (oder der Aktion) mittels der geschätzten Werte der möglichen Folgezustände → Rekursion



State & Action Values

Idee:

Summe der erwarteten künftigen Rewards, ausgehend vom betreffenden State (bzw. der betreffenden Action)

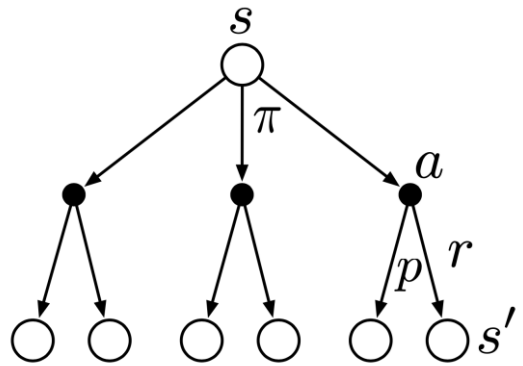
→ Ausdruck der langfristigen Attraktivität von States/Actions

Motivation:

Verbesserung der Effizienz der Suche nach einer guten Action Policy

→ Wähle Actions mit höchsten Action Values

State-Value Funktion



Backup diagram for v_π

from Sutton

Schritte (in Episode)

(discounted) Return

$$v_\pi(s_t) = E_\pi\left[\sum_k \gamma^k r_{t+k+1} \,|s_t\right]$$

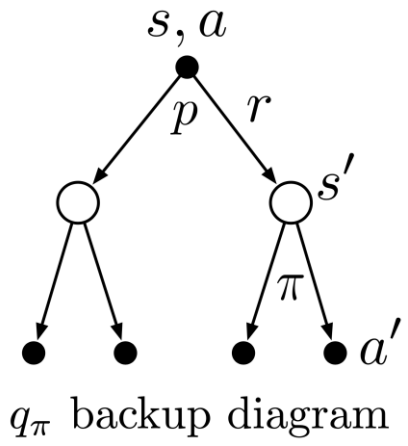
$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s'_{t+1}, r_{t+1}} p(s'_{t+1}, r_{t+1}|s_t, a_t)[r_{t+1} + \gamma v_\pi(s'_{t+1})]$$

Discount-Faktor: zwischen 0 und 1
(essentiell für kontinuierliche Probleme)

Action-Policy
(potentiell stochastisch)

Übergangswahrscheinlichkeiten
(Modell) der Umgebung
(potentiell stochastisch)

Q-Learning



q_π backup diagram from Sutton

Action-Value Funktion:

$$q_\pi(s_t, a_t) = \sum_{s'_{t+1}, r_{t+1}} p(s'_{t+1}, r_{t+1} | s_t, a_t) \left[r_{t+1} + \gamma \sum_{a'_{t+1}} \pi(a'_{t+1} | s'_{t+1})\, q_\pi(s'_{t+1}, a'_{t+1}) \right]$$

entfällt für deterministische Umgebung

Q-learning:
Approximation mit
$\max_a q(s_{t+1}, a_{t+1})$

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1})$$

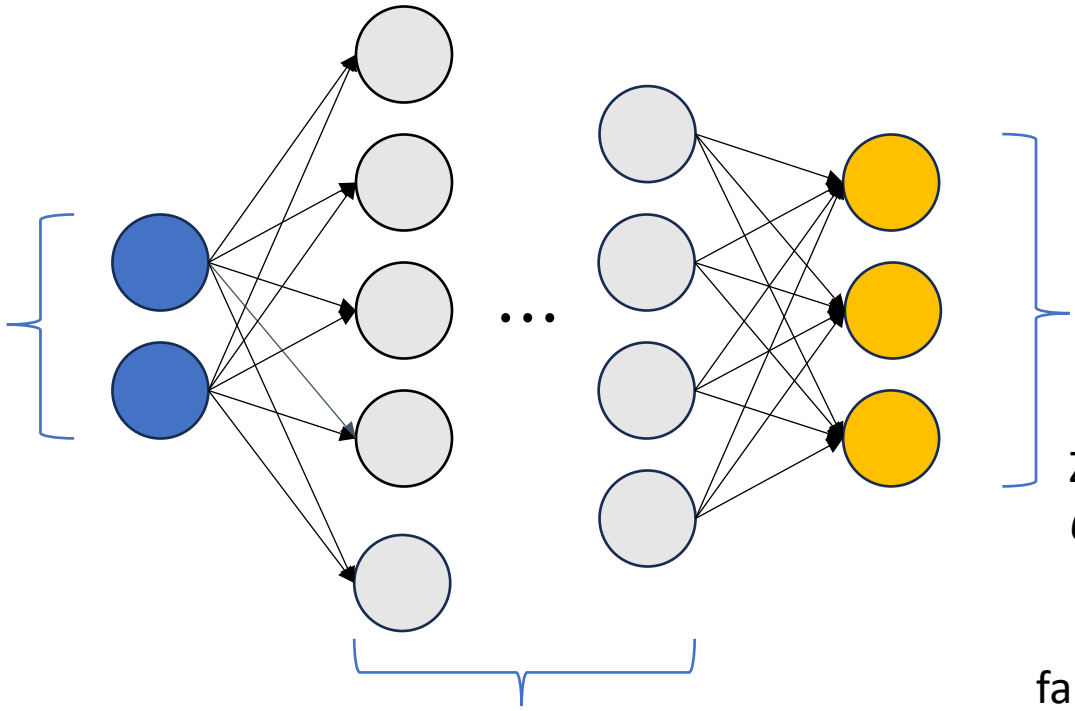
DQN

Deep Q-Network (DQN)

Idee: Approximation der Q-Funktion durch ein neuronales Netzwerk

→ Generalisierung mittels supervised learning

Zustandsgrößen des betrachteten States (z.B. I, U)



hidden layers

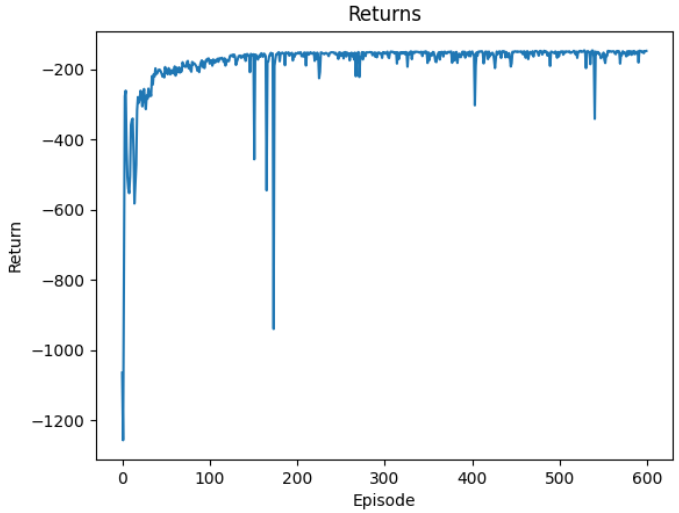
Q-Werte für die verschiedenen Actions (z.B. diskrete ΔU Werte)

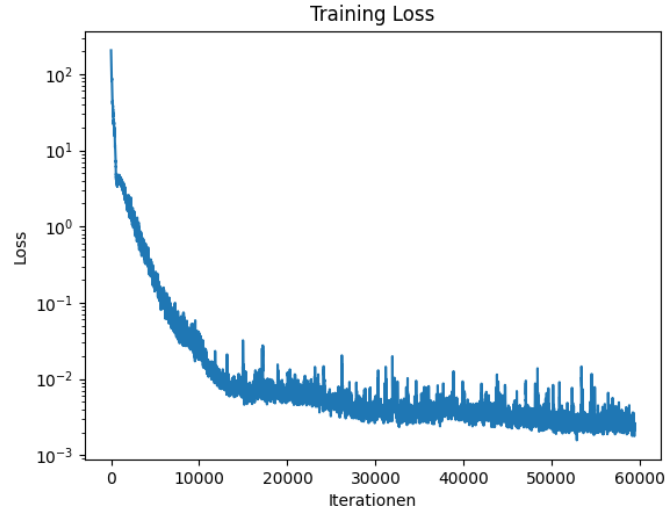
Zielgröße:

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1})$$

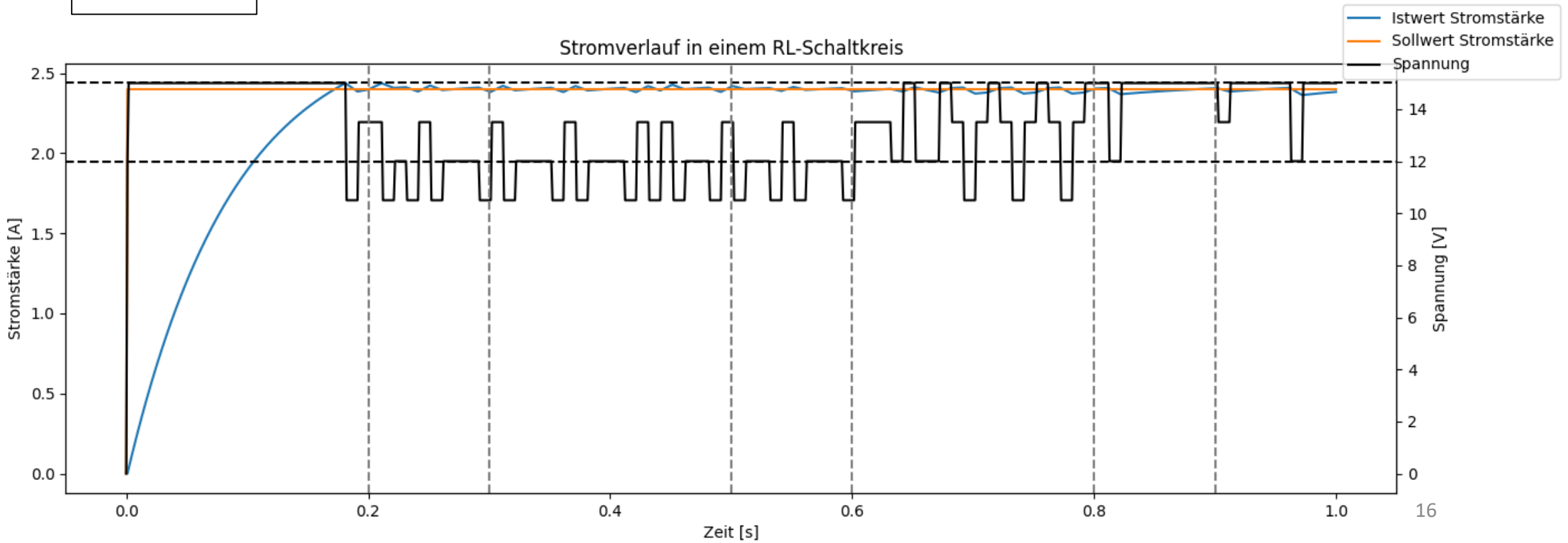
falls 0, nur unmittelbarer Reward

DQN Ergebnisse

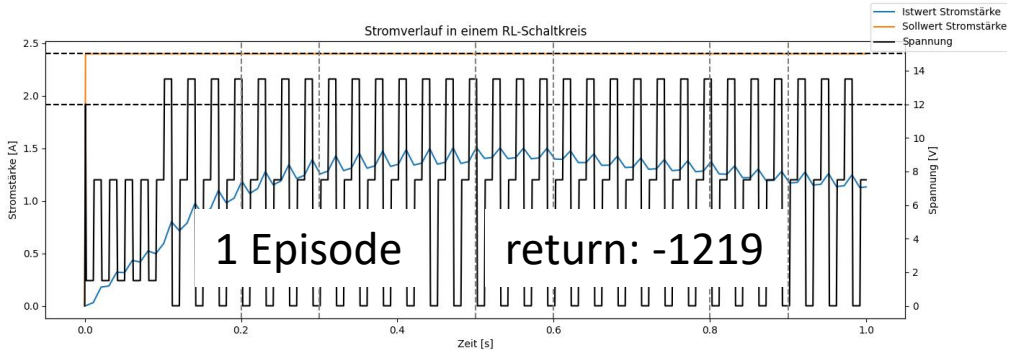


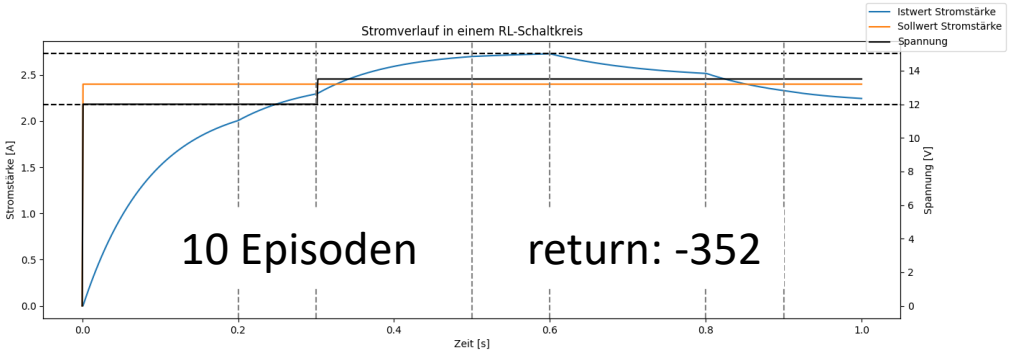


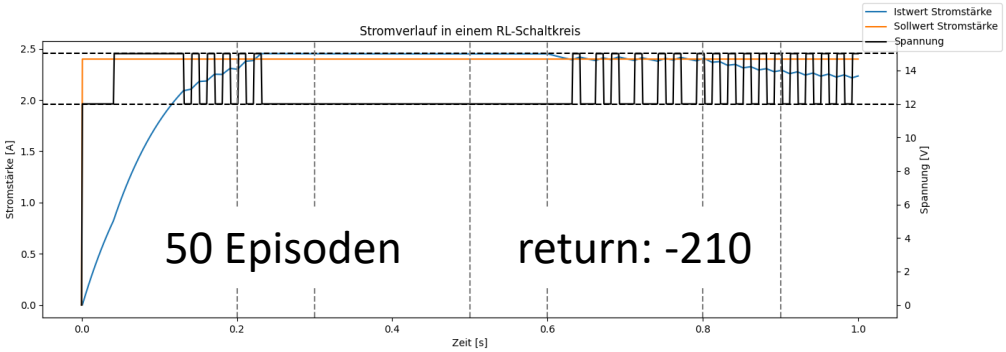
return: -149

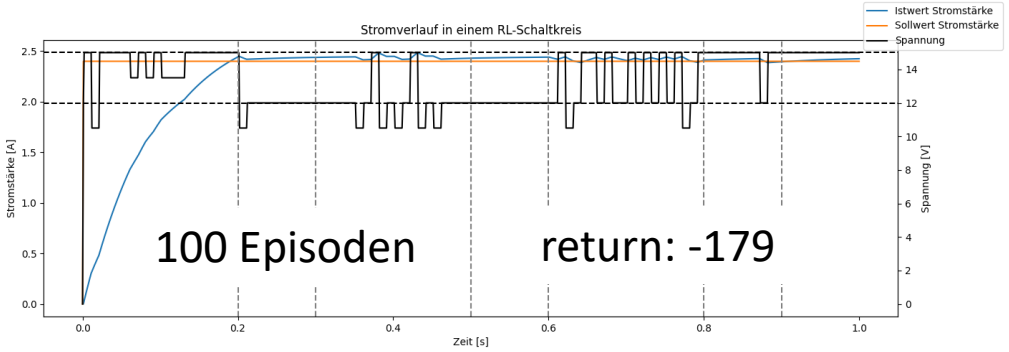


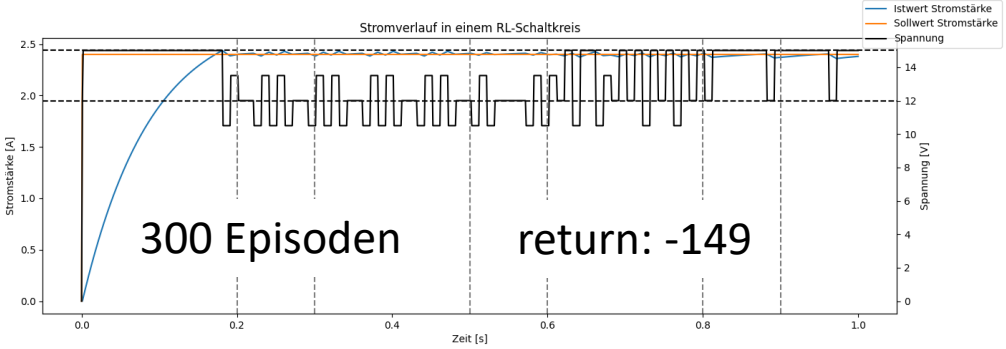
Verbesserung während des Trainings

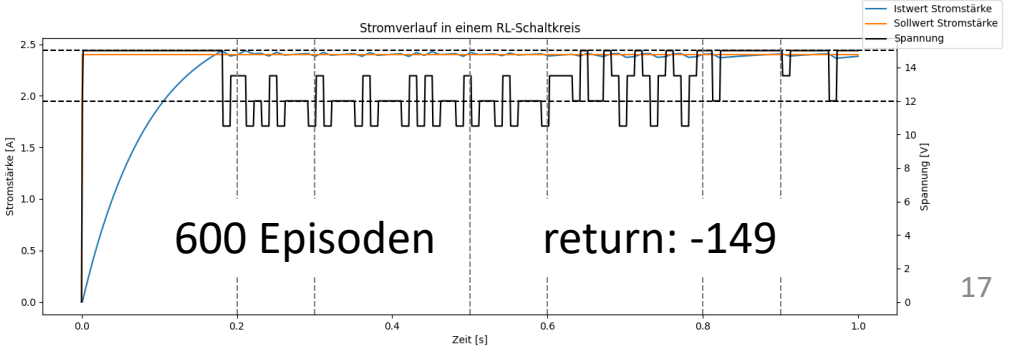












Besonderheiten im DQN-Training

Tricks zur Stabilisierung des Trainings:

experience replay

Anpassung der Netzwerk-Gewichte anhand von zufällig ausgewählten (zuvor gespeicherten) Beobachtungen (Korrelation aufeinanderfolgender Schritte)

(Beobachtungen potentiell mit anderer Policy erzeugt → off-policy Methode)

separates **target network**

Kopie des Q-Netzwerks, aber nur langsam aktualisiert (dynamische Zielgröße)

Exploration

Action Policy nach dem Training:

- wähle in jedem Schritt Action mit höchstem Q-Wert (Exploitation)

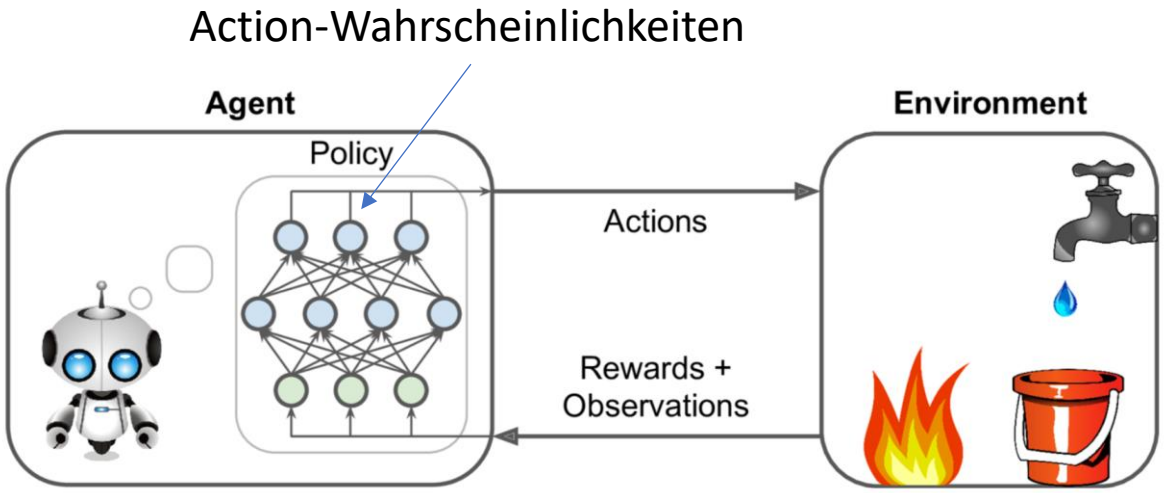
Action Policy während des Trainings:

- wähle manchmal zufällige Action (Exploration)
- z.B. mittels Epsilon-Greedy Algorithmus
- andernfalls Gefahr bessere Strategien zu übersehen

Policy Gradients

Direkte Policy Suche: Policy Gradient Methode

Neuronales Netzwerk (z.B.) mit Actions als Ausgabe/Zielgröße: Policy Netzwerk



beobachtete discounted Returns
(Monte Carlo Sampling)

zu minimierender Loss: $-\log P(a_t|s_t;\boldsymbol{w}) \cdot (r_{t+1} + \gamma r_{t+2} + \cdots)$

gradient ascent: wähle Actions
Richtung größerer Rewards

für stochastisch ausgewählte Action
→ implizite Exploration, on-policy Methode

Actor-Critic Methoden

Interpretation: $-\log P(a_t|s_t;\boldsymbol{w}) \cdot (r_{t+1} + \gamma r_{t+2} + \cdots)$

Actor

Critic

keine Veränderung für Gradient bei Subtraktion von Action-unabhängiger Baseline:

zu minimierender Loss: $-\log P(a_t|s_t;\boldsymbol{w}) \cdot [(r_{t+1} + \gamma r_{t+2} + \cdots) - B]$

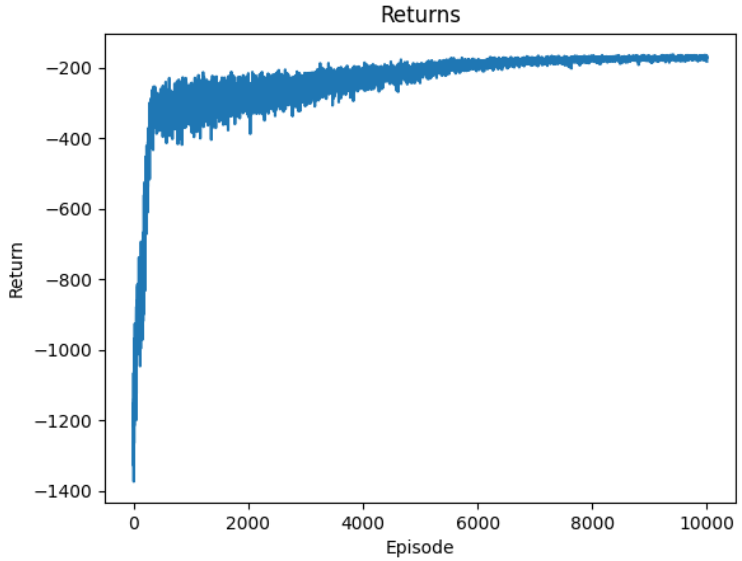
Advantage

mögliche Baselines:

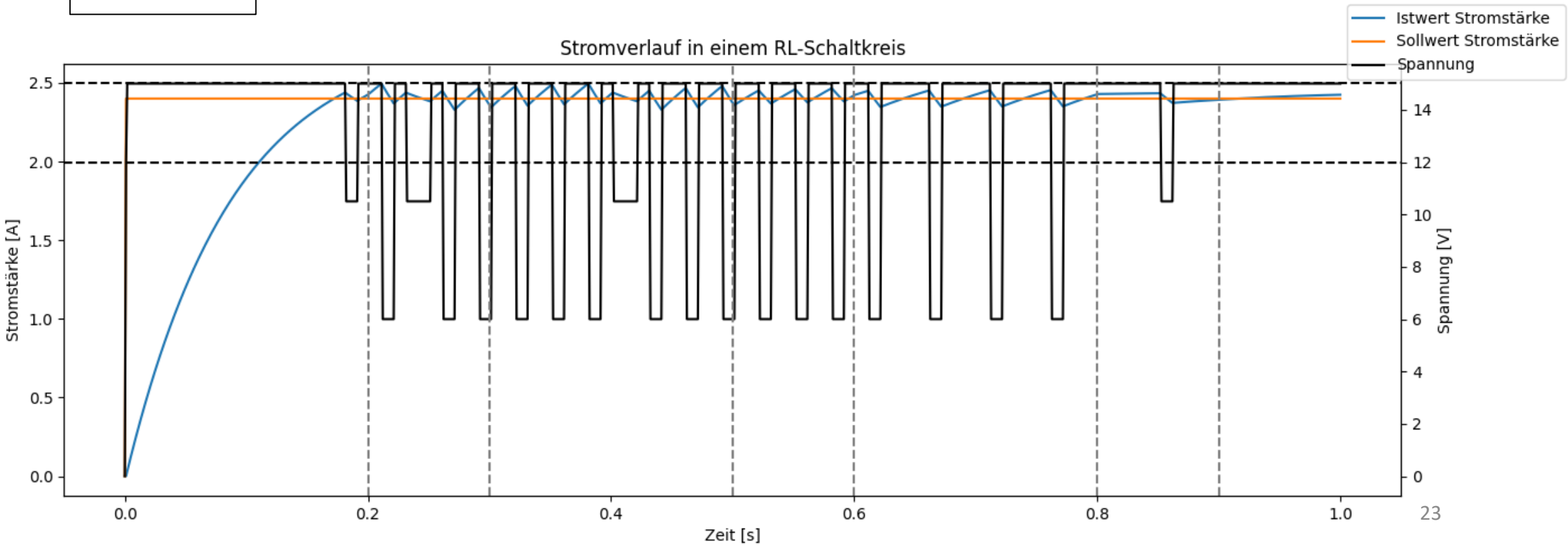
- mittlerer discounted Return
- Value Funktion (Monte Carlo → TD)

→ Reduktion der Varianz der Gradienten

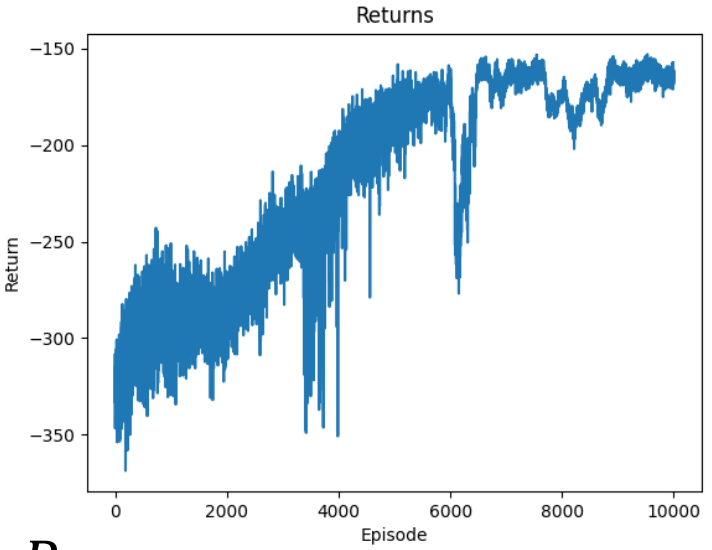
PG - Diskrete Actions



return: -162

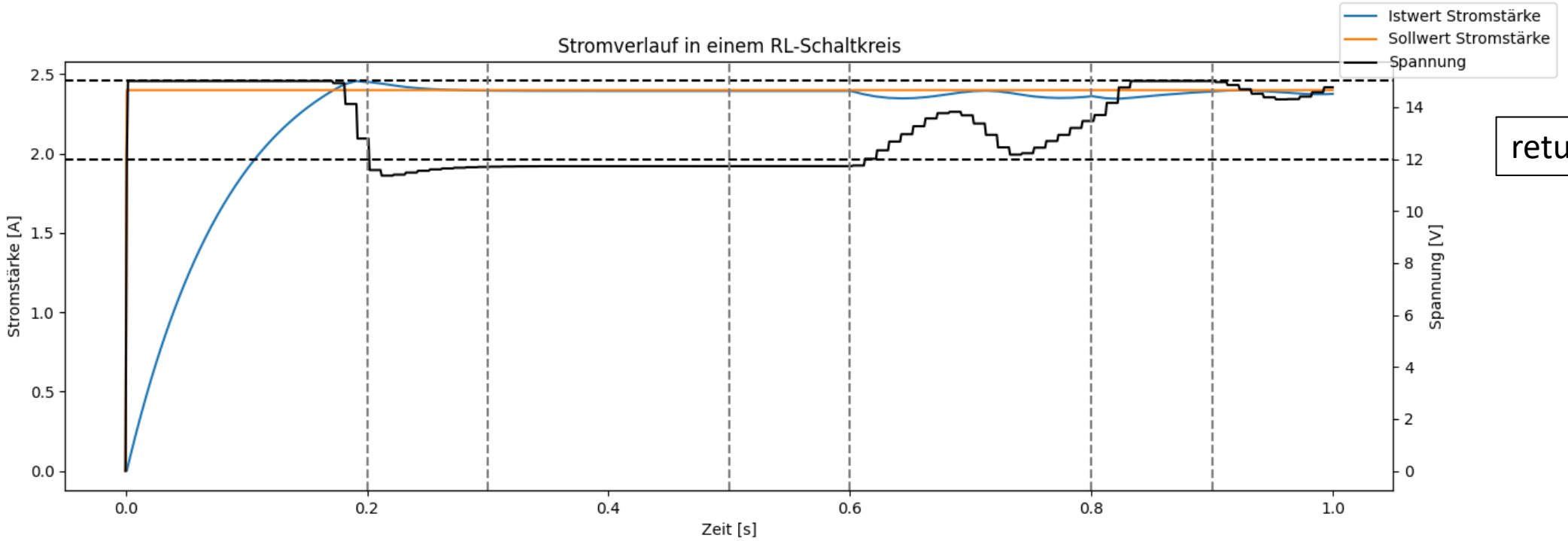


PG - Kontinuierliche Actions



Prognose von Wahrscheinlichkeitsverteilung für Action

- z.B. für Gaußverteilung: zwei Ausgabeknoten für Mittelwert und Standardabweichung
- Action wird zufällig gemäß dieser Verteilung gezogen und $\log P$ dann für die gezogene Action bestimmt



return: -156

Zusammenfassung

Regler bei steigender Komplexität der Regelung:

- PID: reaktiv
- MPC: optimale Planung mittels (approximativem) Modell

  (So machen das Menschen. Ok, nicht unbedingt optimal ☺.)

- Reinforcement Learning: erlernen einer Strategie durch Interaktion mit der Umgebung (Versuch und Irrtum, Brute Force Methode)