Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2025-08 Dozent: Karsten Flügge

Themen des Tages

Tag X

Übersicht bisheriger Algorithmen

PPO
ist momentan so der beste algo, wenn man irgendein Problem lösen will, einfach:

```
from stable_baselines3 import PPO
```

auch immer gut zum Vergleichen

Hierarchic RL

Flow?

Übersicht

Unsere Algorithmen und Formeln und Loss

Methode	Typ	Update-Regel / Loss	Bemerkung
SARSA	On-Policy **TD**	$\mathbf{Q(s,a)} \leftarrow Q(s,a) + \mathbf{\alpha}\,[r + \gamma\, Q(s',a') - Q(s,a)]$	Focus auf **Loop**
Temporal Difference TD(0)	On-Policy **TD**	$V(s) \leftarrow V(s) + \alpha\,[r + \mathbf{\gamma\, V(s') - V(s)}]$	ein **Step**, V / Q
Monte Carlo (**MC**)	episodisch	$V(s) \leftarrow V(s) + \alpha\,[\mathbf{G_t} - V(s)] \quad G_t = \sum_k \gamma^k\, r_{t+k}$	**G**anze **Episode**, V / Q
DQN / **D**QL	Off-Policy TD + NN	Loss $L = \alpha\,[r + \mathbf{\gamma\, \max_{a'} Q_{t-}(s',a') - Q_t(s,a)}]^2$	**PG** Loss via Adam
REINFORCE	On-Policy **PG**	Loss $L = \alpha \sum \log \pi(a \mid s) \cdot \mathbf{G_t}$	G_t kann durch A / δ ersetzt werden:
Actor-Critic (**AC**)	Hybrid PG + T**D**	Actor Loss $L = \log \pi(a \mid s) \cdot \mathbf{\delta}$; $\delta = r + \gamma\, V(s') - V(s)$ Critic L= δ	auch mit MC!
PPO	On-Policy PG	$L = E[\min(r_t \hat{A}_t, \mathbf{clip}(r_t, 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$ $r_t = \pi(a\|s)/\pi_l(a\|s)$ ratio	$\mathbf{r_t = \pi/\pi_l \approx 1}$ **nahe policy!**
Dreamer	AC+MC+PPO	Zusammen AC+MC+PPO Aspekte kombinierbar!	

Übersicht

ALLE Algorithmen lassen sich auf alle Probleme anwenden mit zwei Tricks:

Kontinuierlich => Diskret
über 'buckets' (macht man nicht mehr sobald man NN hat)

Diskret => Kontinuierlich
automatisch über neuronale Netze

Kontinuierlich als Regression

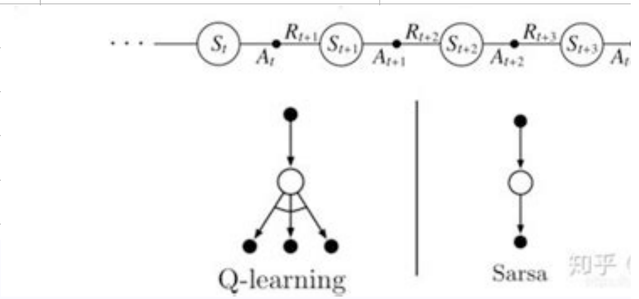
```
um die Aktion macht man noch eine Normalverteilung herum, damit es in algos passt siehe mountainCar.py
Normal(mu, std)  mu = Regressionswert direkt aus dem Netz (gegebenenfalls tanh Wertebereich eingeschränkt) std frei
```

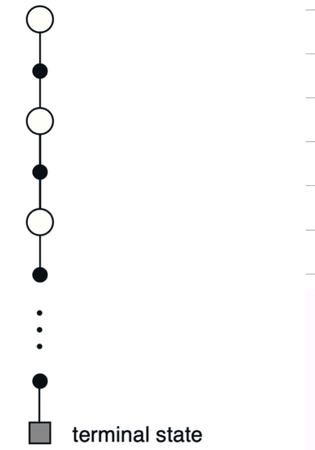
Out of the box

```
from stable_baselines3 import A2C
from stable_baselines3 import PPO
funktionieren für beides!
```

Übersicht bisherige Algorithmen

Methode	Lernmethode	Update-Formel	Rückfluss / Zielwert (target)	Zielfunktion J(θ)	Notiz
TD (allgemein)	Temporal-Difference, hier **off**-policy	$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\, \mathbf{r_{t+1} + \gamma\, Q(s',a') - Q(s,a)} \,]$	$r_{t+1} + \gamma\, Q(s',a')$	$E_{(\pi\theta)}[\, r_{t+1} + \gamma\, Q(s',a') \,]$	**ein Schritt**, Bootstrapping, Updates vor Episodenende, geringer Datenbedarf
SARSA	TD, **on**-policy	$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\, \mathbf{r_{t+1} + \gamma\, Q(s',a') - Q(s,a)} \,]$, a′ aus aktueller Politik	$r_{t+1} + \gamma\, Q(s',a')$	$E_{(\pi\theta)}[\, r_{t+1} + \gamma\, Q(s',a') \,]$	Lernt tatsächlich gespielte Politik, stabil in stochastischen Umgebungen
Monte-Carlo	Monte-Carlo-Schätzung (on-/off-policy)	$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\, G_t - Q(s,a) \,]$	$G_t = \Sigma_{k=0}\, \gamma^k\, r_{t+k+1}$	$E_{(\pi\theta)}[\, G_t \,]$	**über Episode**
Q-Learning	TD, off-policy	$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\, \mathbf{r_{t+1} + \gamma\, max_{a'}\, Q(s',a') - Q(s,a)} \,]$	$r_{t+1} + \gamma\, max_{a'}\, Q(s',a')$	$E[\, r_{t+1} + \gamma\, max_{a'}\, Q(s',a') \,]$	**max,** Findet optimale Politik unabhängig vom Verhalten
DQL	Q-Learning mit Neuronalem Netz	$\theta \leftarrow \theta + \mathbf{\alpha}\, \nabla\theta\, (\mathbf{r_{t+1} + \gamma\, max_{a'}\, Q^{-}(s',a') - Q(s,a)}\,)^2$	y wie oben, Target-Netz θ^{-}	$E[\, (y - Q\theta(s,a))^2 \,]$	Skaliert auf **kontinuierliche** Zustandsräume
Policy Gradient (PG)	Direkte Politikoptimierung	$\theta \leftarrow \theta + \alpha\, \nabla\theta \log \pi\theta(a \mid s) \cdot G_t$	G_t wie MC	$E_{(\pi\theta)}[\, G_t \,]$	
α	Lernrate	Man kann diskret und kontinuierlich mit 'tricks' austauschen:	Empfehlung: **MC!**		
γ	Diskontfaktor	kontinuierlich -> diskret buckets für Zustandsraum	Empfehlung: **PG** baselines		
π	Politik (Policy)	kontinuierlich -> diskret Normal sampling für Aktionsraum			
θ	Netzwerk- oder Politikparameter				
Q(s,a)	Quality / Aktionswertfunktion				
G_t	Gewinn/ Goal / Gain ab Zeit t				
Qθ	θ theta steht für Gewichte im NN				

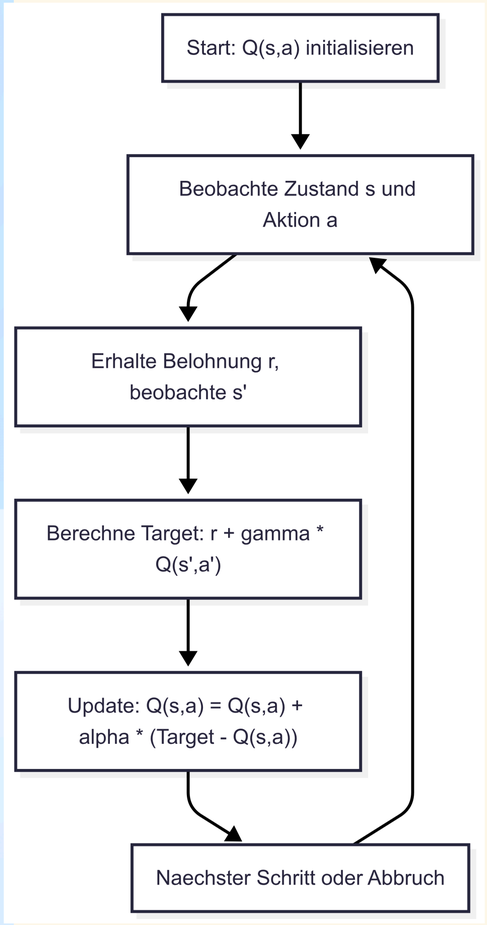




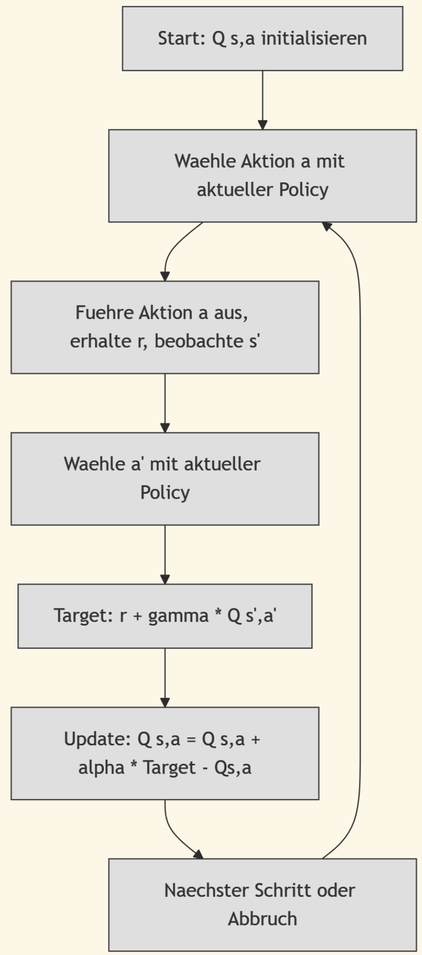
backup diagram for Monte Carlo

Übersicht bisherige Algorithmen

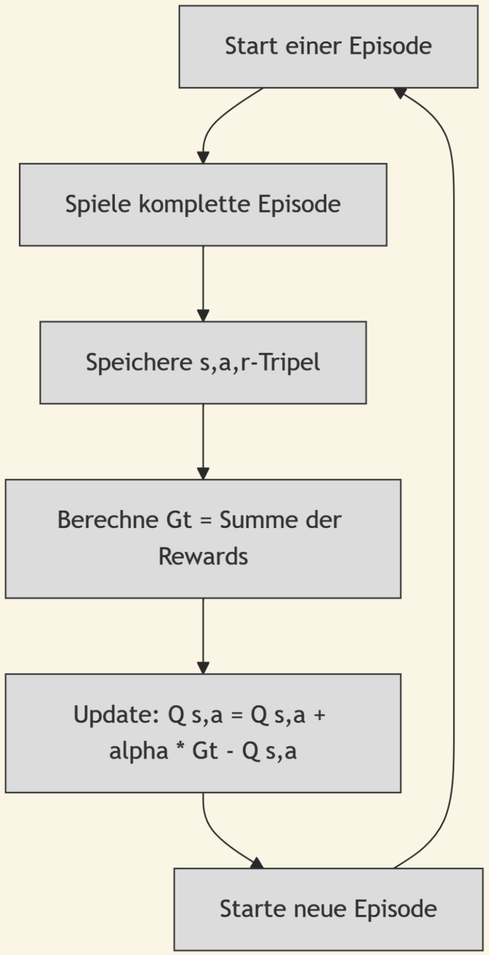
TD



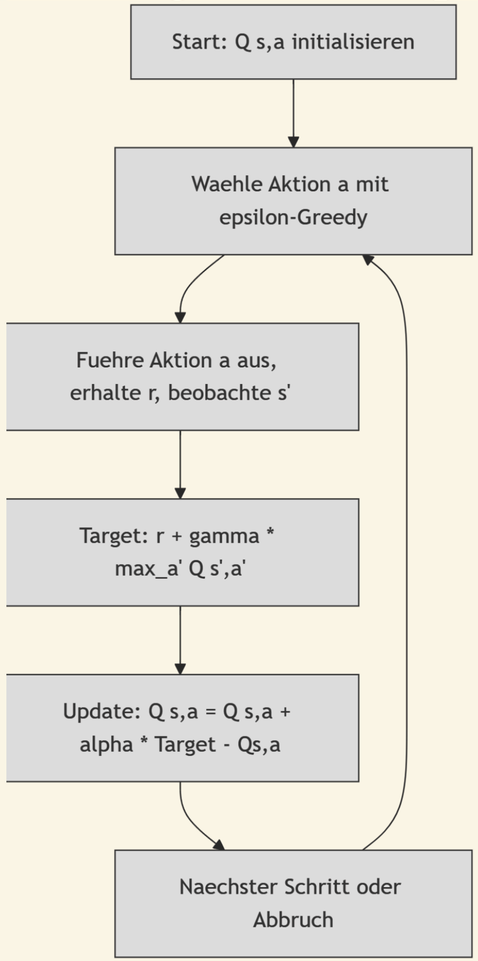
SARSA



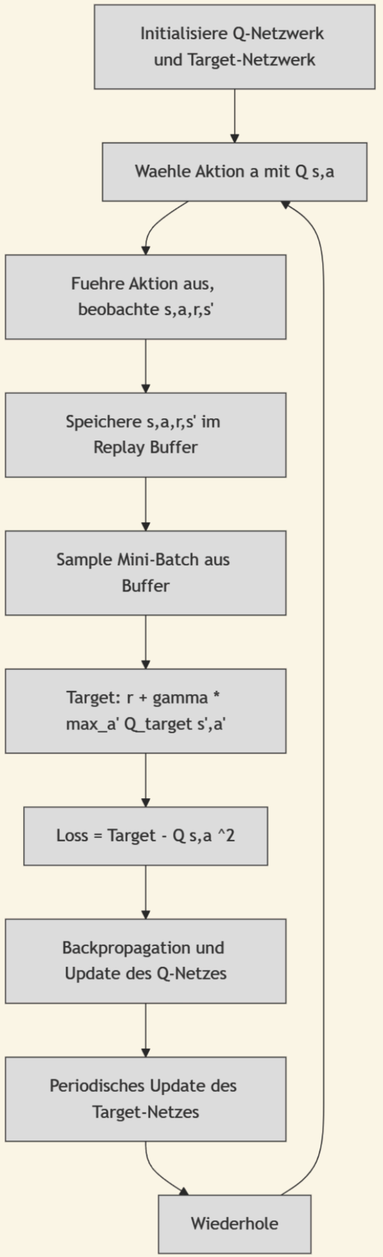
Monte Carlo



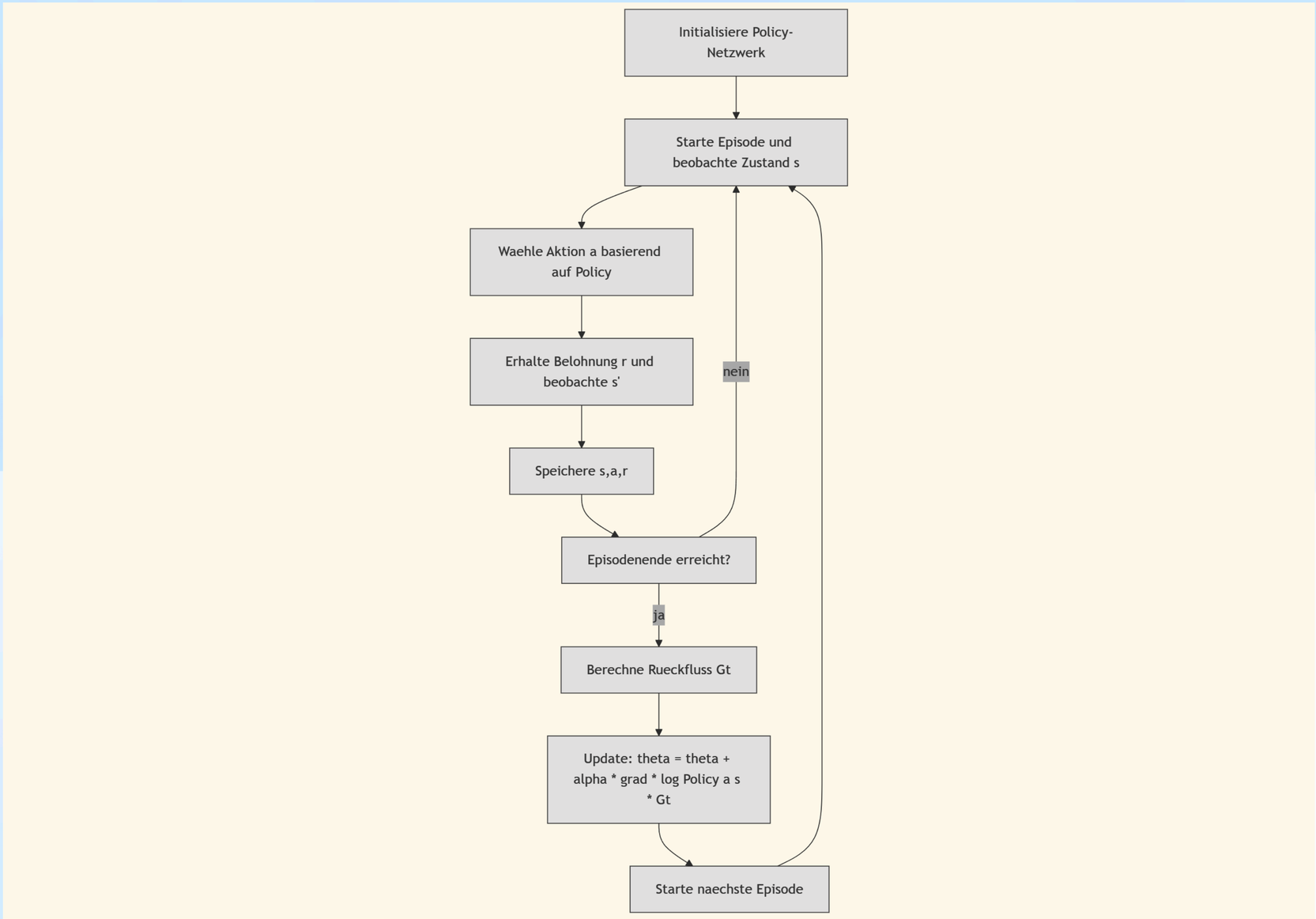
Q-Learning



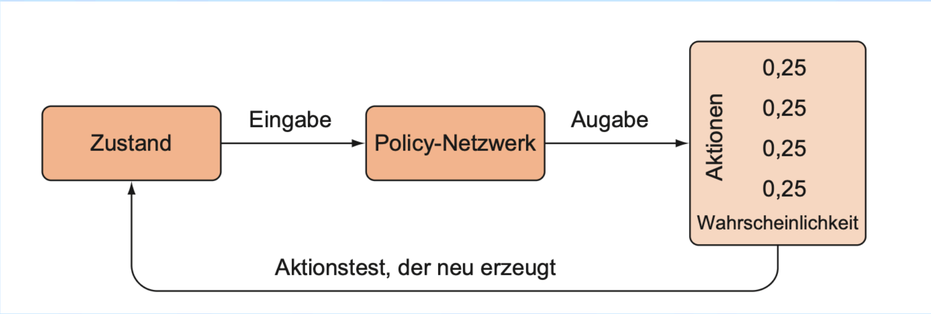
Deep:

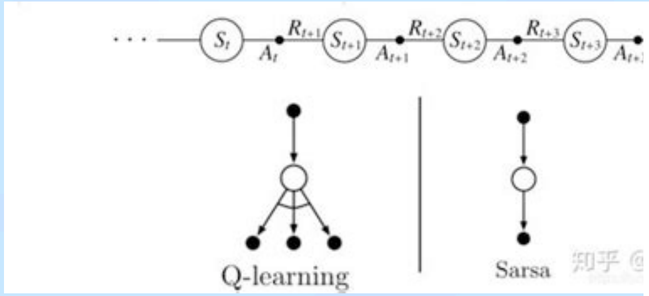


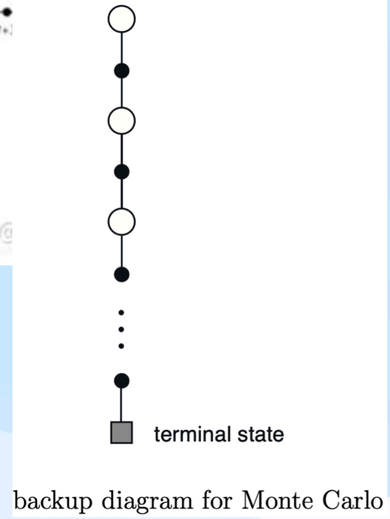
Übersicht bisherige Algorithmen



Übersicht bisherige Algorithmen







backup diagram for Monte Carlo

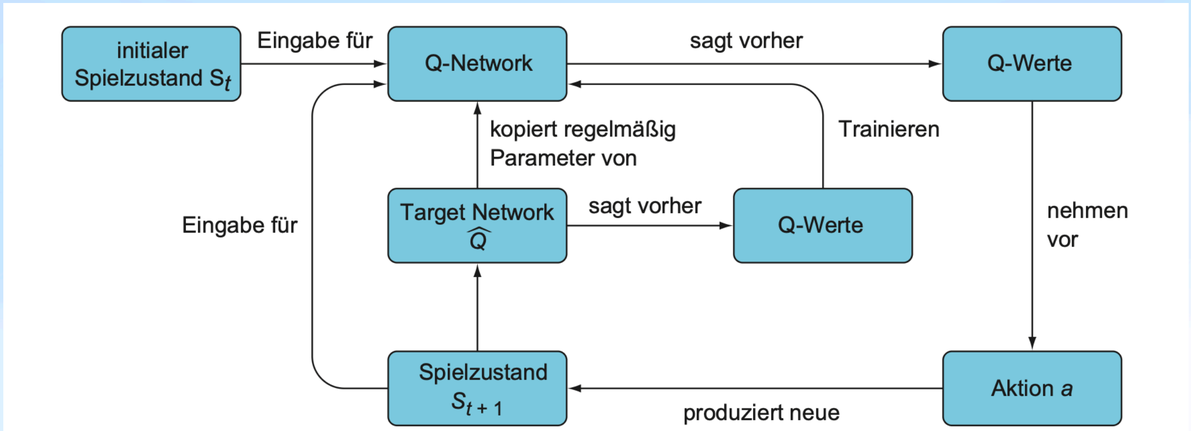
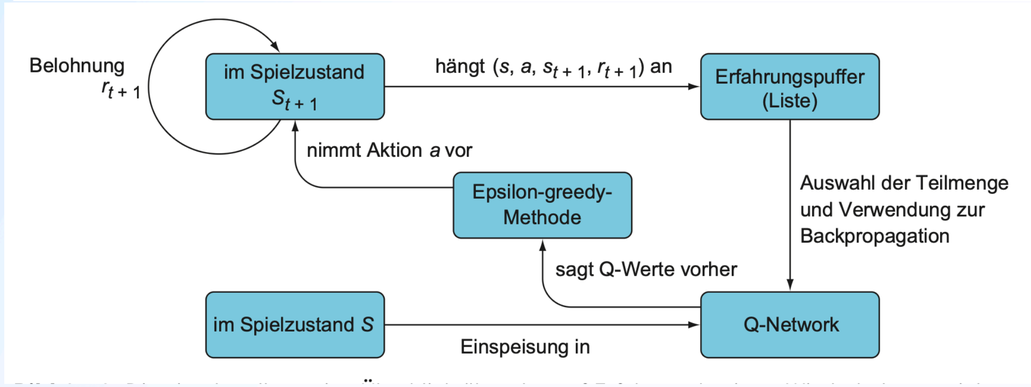


Bild 3.15 Dies ist der allgemeine Überblick für Q-Learning mit einem Target Network. Es ist eine



Update policy network

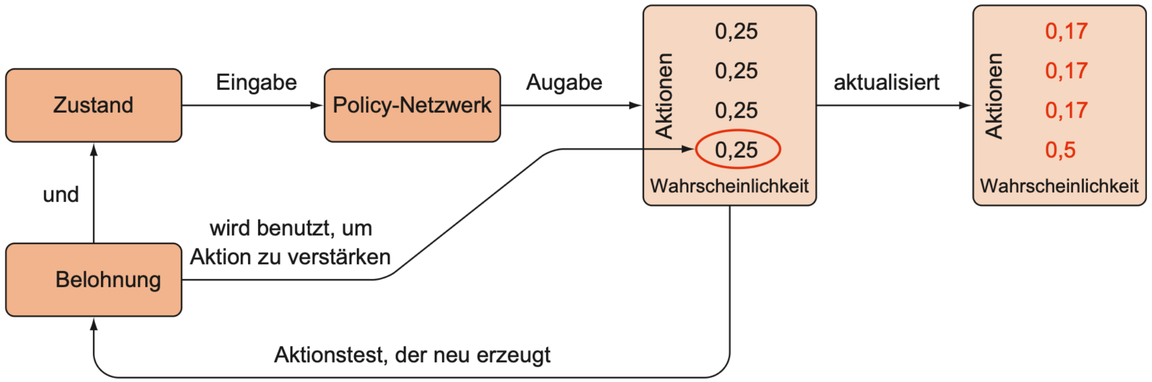
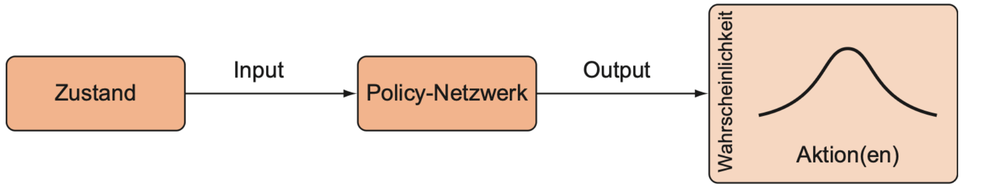


Bild 4.6 Sobald eine Aktion aus der Wahrscheinlichkeitsverteilung des Policy-Netzwerks ausgewählt wird, erzeugt sie einen neuen Zustand und eine neue Belohnung. Das Belohnungssignal wird verwendet, um die ergriffene Aktion zu verstärken, d. h. es erhöht die Wahrscheinlichkeit dieser Aktion angesichts des Zustands, wenn die Belohnung positiv ist, oder es verringert die Wahrscheinlichkeit, wenn die Belohnung negativ ist. Beachten Sie, dass wir nur Informationen über Aktion 3 (Element 4) erhalten haben, aber da die Wahrscheinlichkeiten sich auf 1 summieren müssen, müssen wir die Wahrscheinlichkeiten der anderen Aktionen verringern.



Update policy network

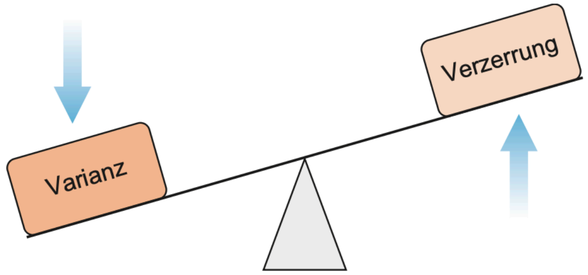


Bild 5.5 Das Verzerrung-Varianz-Dilemma. Zunehmende Modellkomplexität kann Verzerrungen reduzieren, aber sie wird die Varianz erhöhen. Eine Verringerung der Varianz wird die Verzerrung erhöhen.

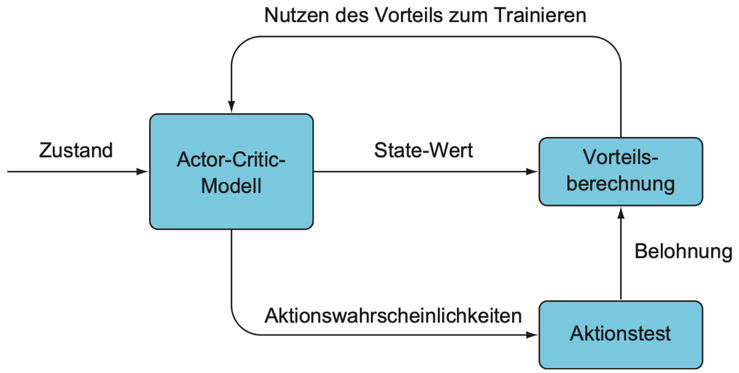


Bild 5.10 Ein Actor-Critic-Modell erzeugt einen Zustandswert und Aktionswahrscheinlichkeiten, die zur Berechnung eines Vorteilswerts verwendet werden, und dies ist die Größe, die zum Trainieren des Modells verwendet wird, und nicht reine Belohnungen wie beim reinen Q-Lernen.

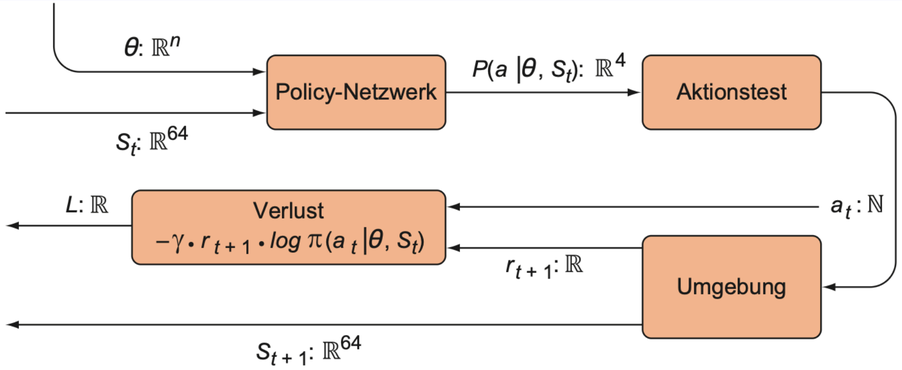
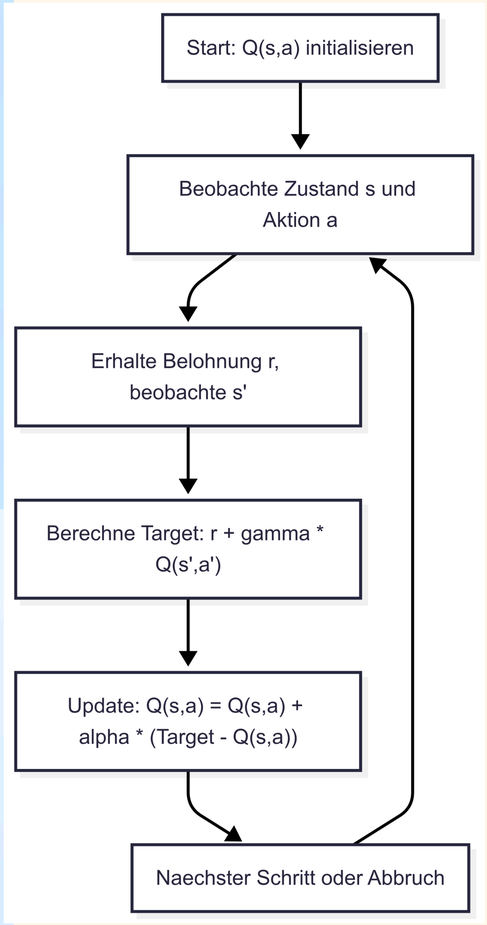


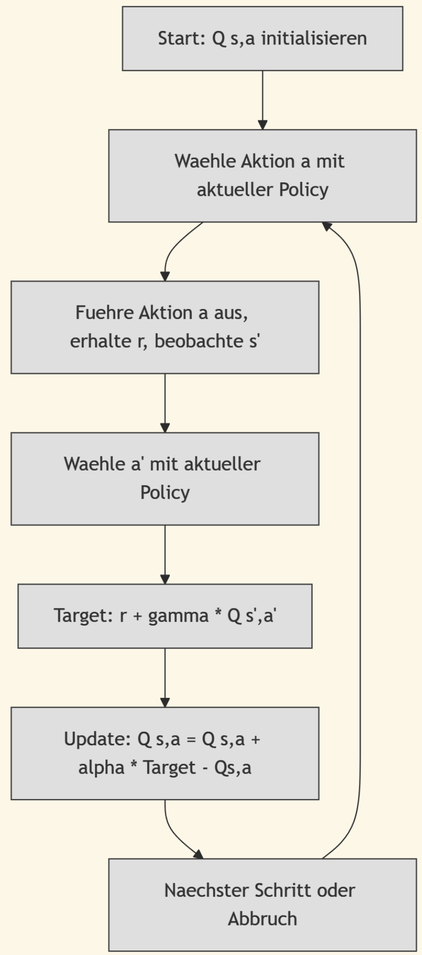
Bild 4.7 Ein String-Diagramm für das Training eines Policy-Netzwerks für Gridworld. Das Policy-

Übersicht bisherige Algorithmen

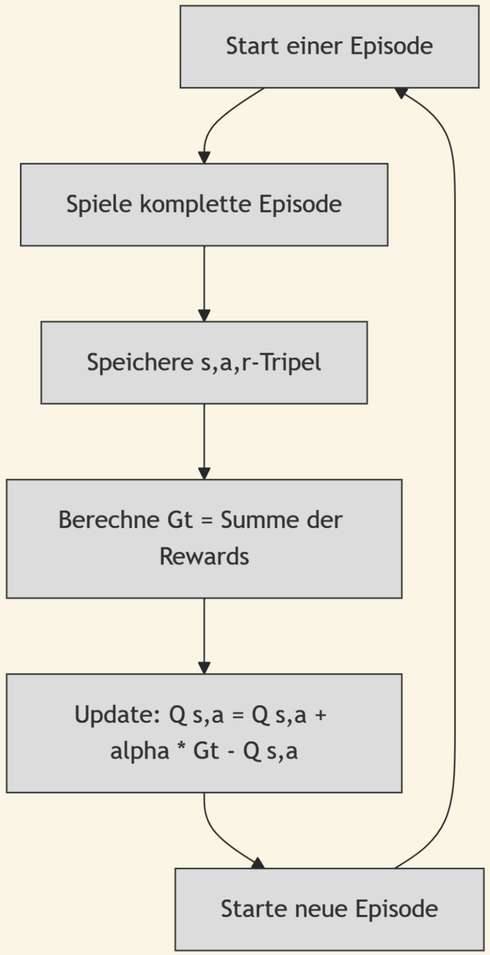
TD



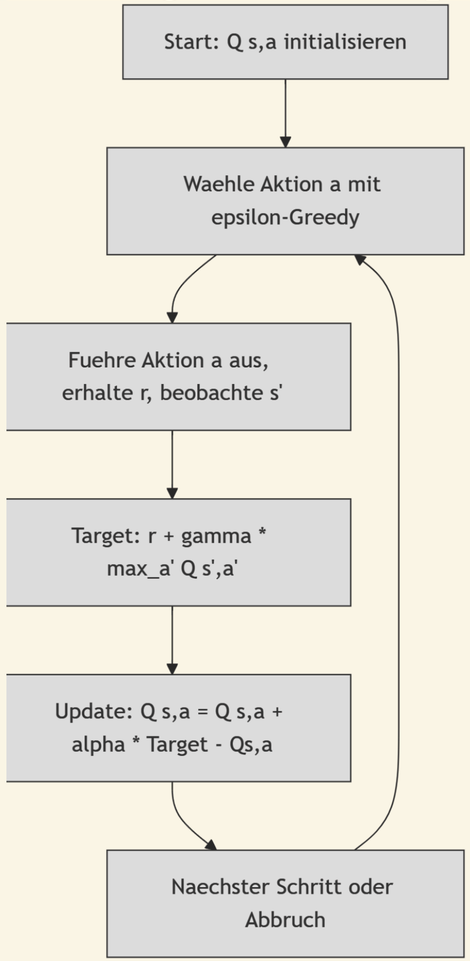
SARSA



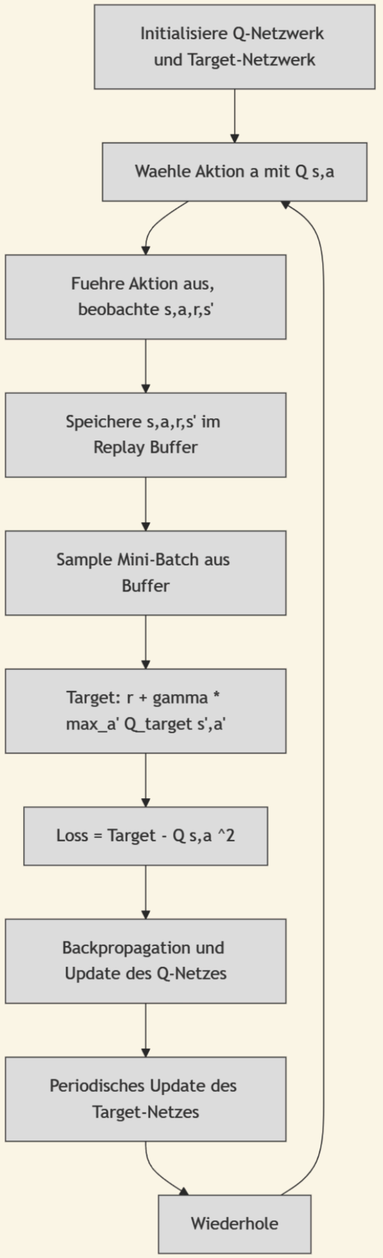
Monte Carlo



Q-Learning



Deep:



Deep Reinforcement Learning

Es lassen sich alle Hilfskonzepte und Hilfsfunktionen welche wir bis jetzt kennen gelernt haben mit **neuronalen Netz trainieren**:

- **Value** Funktionen,
- **Quality** Funktionen oder
- **Policy** direkt!