Temporal Difference Learning

- Das Temporal Difference (TD) Lernen ist eine bedeutende Entwicklung im Reinforcement Lernen.
- Im TD Lernen werden Ideen der Monte Carlo (MC) und dynamische Programmierung (DP) Methoden kombiniert.
- Im TD Lernen wird wie beim MC Lernen aus Erfahrung ohne Kenntniss eines Modells gelernt, d.h. dieses wird aus Daten/Beispielen gelernt.
- Wie beim DP werden Schätzungen für Funktionswerte durchgeführt ($V^\pi(s)$ oder $Q^\pi(s,a)$), die wiederum auf Schätzungen basieren (nämlich die Schätzungen $V^\pi(s')$ nachfolgender Zustände).
- Wir beginnen mit der Evaluation von Policies π, d.h. mit der Berechnung der Wertefunktionen V^π bzw. Q^π.

TD Evaluation

- TD und MC Methoden nutzen Erfahrung aus Beispiele um V^π bzw. Q^π für eine Policy π zu lernen.
- Ist s_t der Zustand zur Zeit t in einer Episode, dann basiert die Schätzung von $V(s_t)$ auf den beobachteten Return R_t nach Besuch des Zustand s_t
- In MC Methoden wird nun der Return R_t bis zum Ende der Episode bestimmt und dieser Schätzwert für $V(s_t)$ angesetzt.
- Eine einfache Lernregel nach der Every Visit MC Methode hat dann die folgende Gestalt:

$$V(s_t) := V(s_t) + \alpha\,[R_t - V(s_t)] \qquad \text{mit} \quad \alpha > 0$$

- In den einfachen 1-Schritt TD Methoden nur der nächste Zustandsübergang $s \to s'$ abgewartet und der unmittelbar erzielte Reward zusammen mit $V(s')$ benutzt.

- Ein 1-Schritt TD Algorithmus, der sog. TD(0) Algorithmushat die Lernregel

$$V(s_t) := V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \quad \alpha > 0, \quad \gamma \in (0,1]$$

- Zur Erinnerung– es gilt

$$\begin{aligned}
V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\
&= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+2+k} \mid s_t = s\right\} \\
&= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\}
\end{aligned}$$

- Sollwert beim MC Lernen : R_t
- Sollwert beim TD Lernen : $r_{t+1} + \gamma V^\pi(s_{t+1})$

TD(0) – Schätzung von V^π

1. Initalize $V(s)$ arbitrarily, π policy to be evaluated

2. Repeat (for each episode)

   Initialize s

   Repeat (for each step of episode):

   $a := \pi(s)$

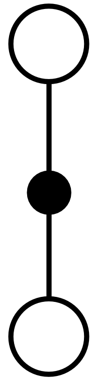
   take a, observe reward r, and next state s'

   $V(s) := V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$

   $s := s'$

   Until s is terminal

TD-Backup Diagramm



$s, s' \in \mathcal{S}$ sind die offenen Kreise

$a \in \mathcal{A}$ die Aktion $\pi(s)$ gefüllter Kreis

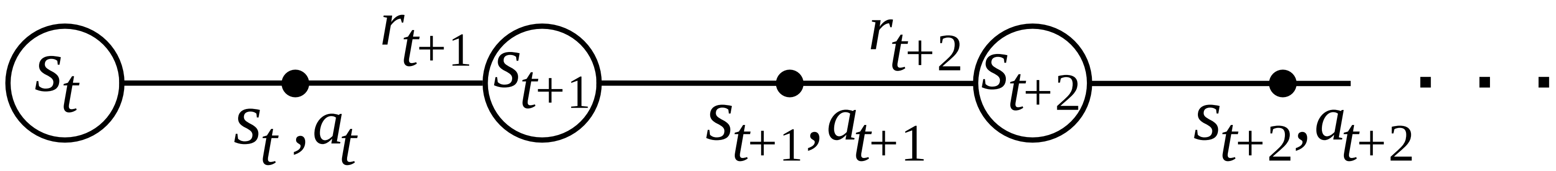
Sarsa

- Ziel ist das Erlernen der Q-Funktion statt der V-Funktion durch On Policy Methode, d.h. Schätzung der Werte $Q^{\pi}(s,a)$ für die verwendete Policy pi.

- Es kann dasselbe Verfahren wie zur Schätzung der V-Funktion verwendet werden mit der Lernregel

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\,[r + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$$

- Hierzu betrachten wir Zustandsübergänge:



Sarsa: Algorithmus

1. Initalize $Q(s,a)$ arbitrarily,
2. Repeat (for each episode)

   Initialize s

   Choose a from s using policy derived from Q (e.g. ϵ-greedy)

   Repeat (for each step of episode):

   Take a, observe reward r, and next state s'

   Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)

   $Q(s,a) := Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$

   $s := s'$; $a := a'$

   Until s is terminal

Q-Learning

Q-Lernen ist das wichtigste Verfahren im Bereich des Reinforcement Lernens, es wurde von Watkins 1989 entwickelt.

Ist ein Off Policy TD Lernverfahren definiert durch die Lernregel

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[r + \gamma \max_a Q(s_{t+1}, a - Q(s_t, a_t) \right]$$

Q konvergiert direkt gegen Q^* (vereinfacht die Analyse des Verfahrens).

Policy π legt die Aktion fest, und somit wird durch π die Folge von (s_t, a_t) festgelegt, die in der Episode vorkommen (und damit auch die Stellen an den die Q-Funktion gelernt wird).

Q-Learning: Algorithmus

1. Initalize $Q(s, a)$ arbitrarily,

2. Repeat (for each episode)

   Initialize s

   Repeat (for each step of episode):

   Choose a from s using policy derived from Q
   (e.g. ϵ-greedy)

   Take a, observe reward r, and s'

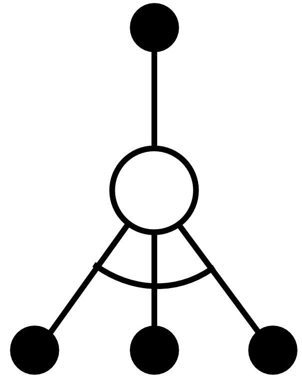
   $a^* :=$ arg max$_a Q(s', a)$

   $Q(s, a) := Q(s, a) + \alpha \left[r + \gamma Q(s', a^*) - Q(s, a) \right]$

   $s := s'$;

   Until s is terminal

Q-Learning Backup



$s, s' \in \mathcal{S}$ sind die offenen Kreise

$a, \in \mathcal{A}$ die Aktion $\pi(s)$ gefüllte Kreise

$\max$ durch Kreisboden

TD n-step Methoden

- Die bisher vorgestellten TD Lernverfahren verwenden den unmittelbar folgenden Reward ($k = 1$-Schritt) r_{t+1}.
- Idee bei den Mehrschritt Methoden ist es, auch die nächsten $k = 2, 3, \dots n$ erzielten Rewards r_{t+k} einzubeziehen.
- Dazu betrachten wir die Zustands-Reward-Folge

$$s_t, r_{t+1}, s_{t+1}, r_{t+2}, \dots, r_T, s_T$$

s_T der Endzustand.

- MC Methoden verwenden zum Backup von $V^\pi(s_t)$ den Return

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \gamma^{T-t-1} r_T$$

R_t ist das Lehrersignal (Sollwert) für die MC Lernverfahren.

- Für 1-Schritt TD Methoden ist das Lehrersignal

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$

hier dient $\gamma V_t(s_{t+1})$ als Näherung für

$$\gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \gamma^{T-t-1} r_T$$

- Bei einem 2-Schritt-TD Verfahren ist der Sollwert

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

wobei jetzt $\gamma^2 V_t(s_{t+2})$ die Näherung ist für

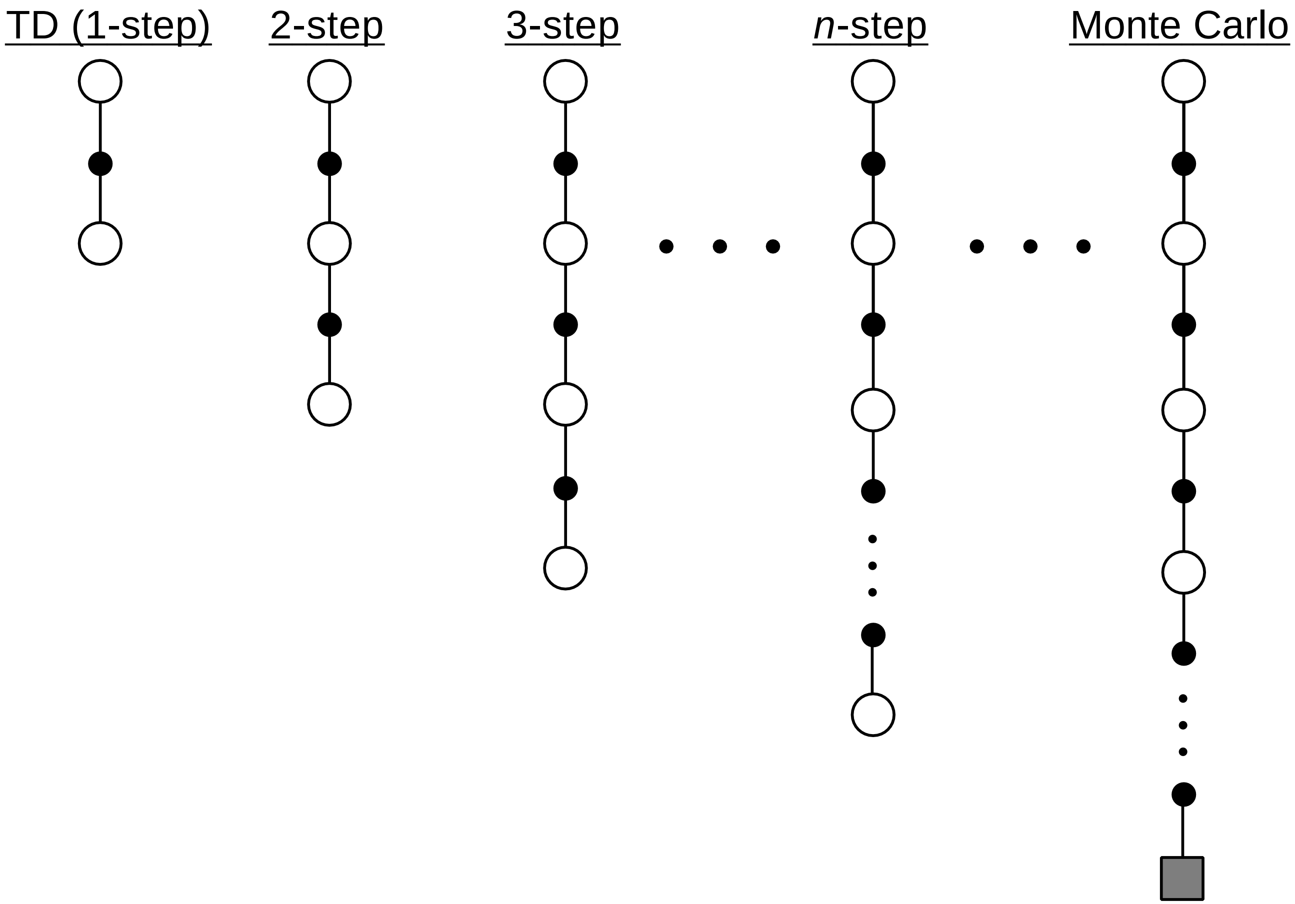
$$\gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^{T-t-1} r_T$$

- Allgemein ist der n-Schritt-Return $R_t^{(n)}$ zur Zeit t gegeben durch

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$$

- Lernregel für die V-Funktion mit n Schritt Backups ist also

$$\Delta V_t(s_t) = \alpha \left[R_t^{(n)} - V_t(s_t) \right]$$



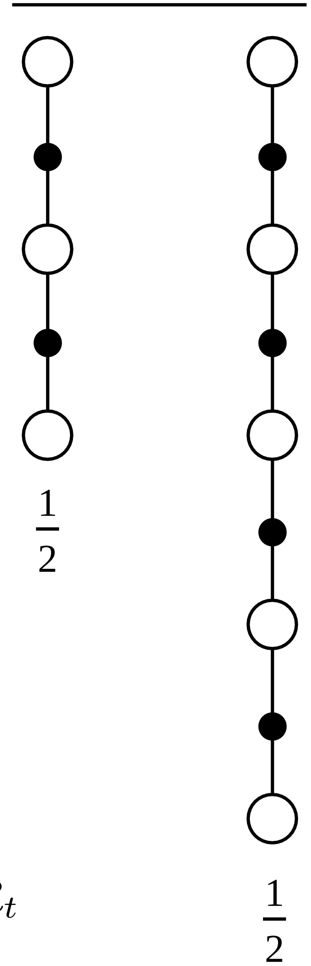
TD(λ)-Verfahren

- Backups können nicht nur auf der Basis von n-Schritt Returns $R_t^{(n)}$, sondern durch Mittelung verschiedener n-Schritt Returns erfolgen, z.B. Mittelwert eines $2-$ und $4-$ Schritt Returns

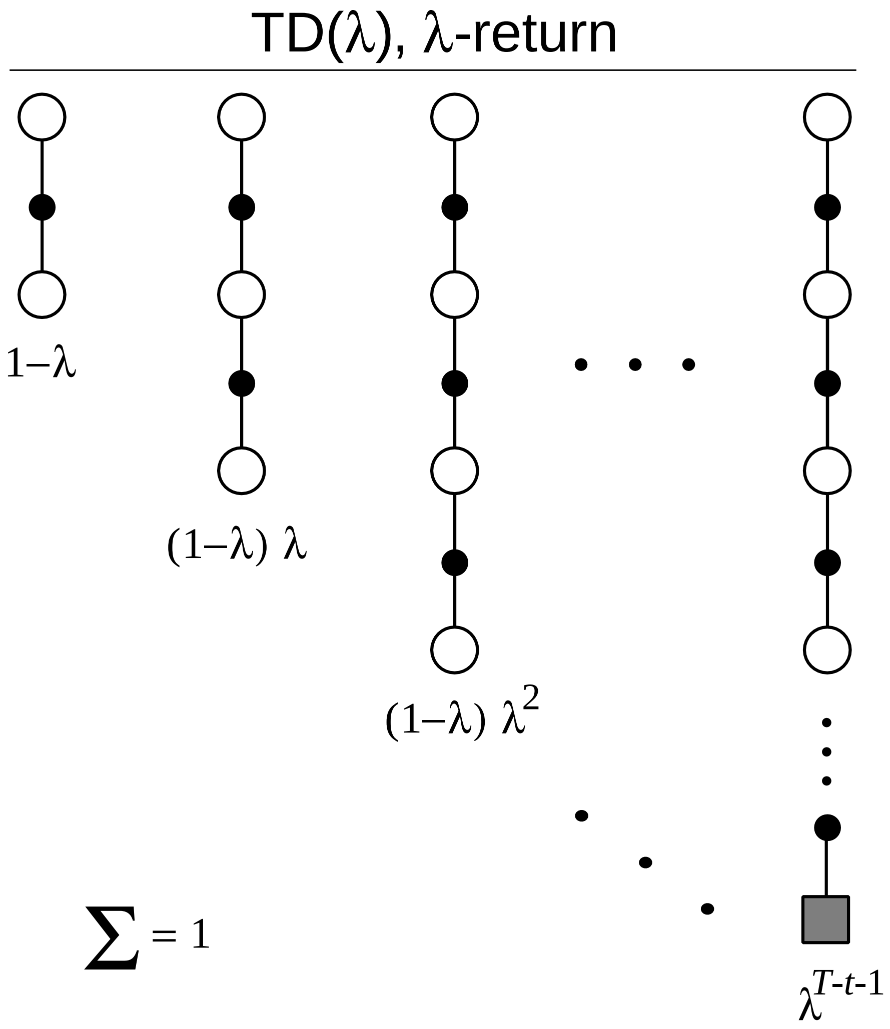
$$R_t^{ave} = \frac{1}{2} R_t^{(2)} + \frac{1}{2} R^{(4)}$$

- Allgemeine Mittelungen sind möglich. Nur die Gewichte sollten nicht-negativ sein und sich zu 1 summieren.
- Dies führt auf die $TD(\lambda)$ Verfahren, hier werden alle n-Schritt Returns gewichtet.
- Mit einem Nomalisierungsfaktor $1-\lambda$ (stellt sicher das die Summe der Gewichte $=1$ ist) definieren wir den λ-Return durch

$$R_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$$



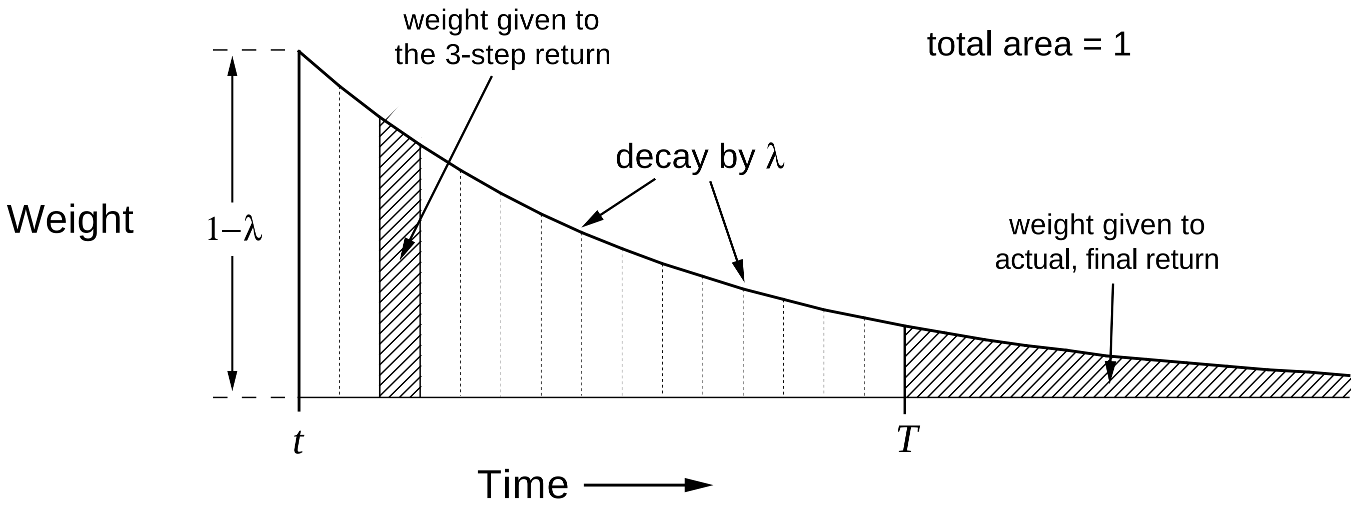
TD(λ)-Backup-Diagramm



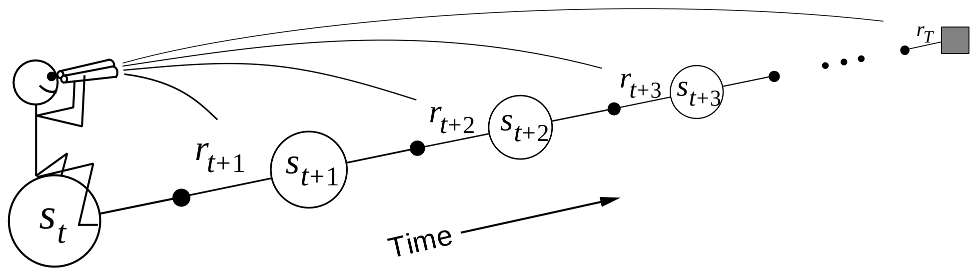
Gewichtung von λ

Update (hier der V-Funktion) bei einem λ-Return Algorithmus

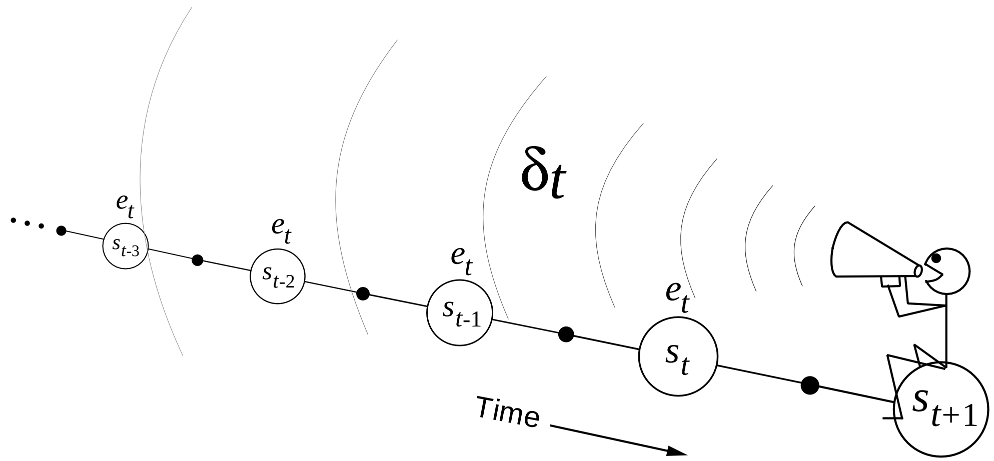
$$\Delta V_t(s_t) = \alpha \left[R_t^\lambda - V_t(s_t) \right]$$



Forward View/Backward View



Forward View: Ist nicht kausal und kann deshalb auch nicht so direkt implementiert werden.



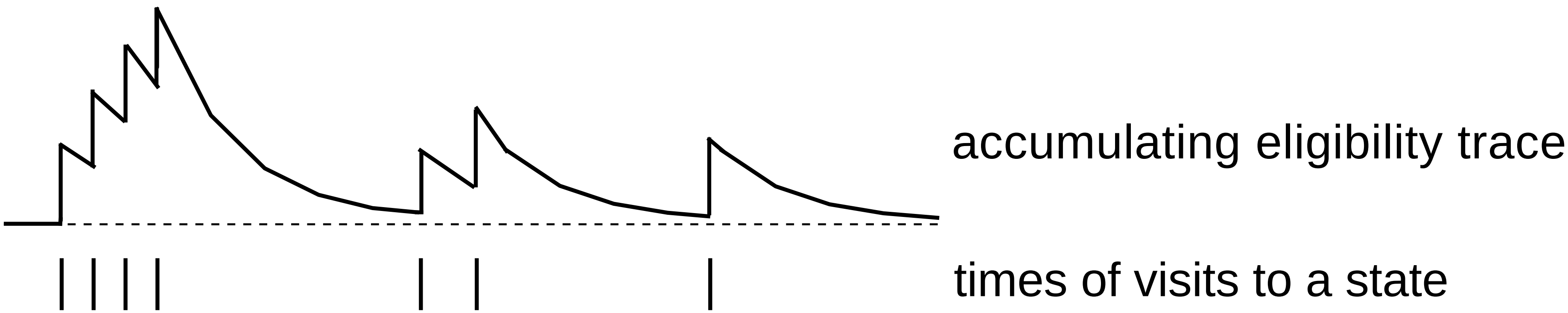
Kummulative Trace-Variable

Backward View benötigt für jeden Zustand eine Trace-Variable $e_t(s)$ die definiert ist als

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & s = s_t \end{cases}$$

Dabei zeigt $e_t(s) > 0$ an, dass der Zustand s kürzlich besucht wurde. Kürzlich ist hierbei durch die Größe $\gamma\lambda$ definiert.

$e_t(s)$ zeigt, für welche Zustände $s \in \mathcal{S}$ die Funktion V bzw. Q anzupassen ist.



Die Fehlersignale sind (hier für V-Funktion):

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

Alle kürzlich besuchten Zustände s werden damit adaptiert (wieder für V)

$$\Delta V_t(s_t) = \alpha \delta_t e_t(s) \quad \text{für alle } s \in \mathcal{S}$$

Hierbei ist wieder $\gamma \in (0, 1]$ der Diskontierungsfaktor und $\alpha > 0$ eine konstante Lernrate.

TD(λ)

1. Initalize $V(s)$ arbitrarily and $e(s) = 0$; π policy to be evaluated
2. Repeat (for each episode)

   Initialize s

   Repeat (for each step of episode):

   $a := \pi(s)$

   take a, observe reward r, and next state s'

   $\delta := r + \gamma V(s') - V(s)$

   $e(s) := e(s) + 1$;

   For all s:

   $V(s) := V(s) + \alpha\delta e(s)$

   $e(s) := \gamma\lambda e(s)$

   $s := s'$

   Until s is terminal

Äquivalenz der beiden Methoden

Wir zeigen nun, das die Updates von V der Vorwärts- und Rückwärtssicht für das Off-line-Lernen äquivalent sind.

- Es sei$\Delta V_t^\lambda(s_t)$ die Änderung von $V(s_t)$ zur Zeit t nach der λ-Return Methode (Vorwärtssicht).

- Es sei $\Delta V_t^{TD}(s)$ die Änderung von $V(s)$ zur Zeit t von Zustand s nach dem TD(0) Algorithmus (Rückwärtssicht).

Ziel ist es also zu zeigen

$$\sum_{t=0}^{T-1} \Delta V_t^\lambda(s_t)\mathbf{1}_{[s=s_t]} = \sum_{t=0}^{T-1} \Delta V_t^{TD}(s) \quad \text{für alle } s \in \mathcal{S}$$

es ist $\mathbf{1}_{[s=s_t]}$ gleich 1 genau dann wenn $s = s_t$ ist. Wir untersuchen einen einzelnen Update $\Delta V_t^\lambda(s_t) = \alpha\left[R_t^\lambda - V_t(s_t)\right]$.

$$
\begin{aligned}
\frac{1}{\alpha}\Delta V_t^\lambda(s_t) \quad = \quad & -V_t(s_t) + \\
& (1-\lambda)\lambda^0\left[r_{t+1} + \gamma V_t(s_{t+1})\right] + \\
& (1-\lambda)\lambda^1\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})\right] + \\
& (1-\lambda)\lambda^2\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V_t(s_{t+3})\right] + \\
& (1-\lambda)\lambda^2\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 V_t(s_{t+4})\right] + \\
& \ldots \quad \ldots \quad \ldots
\end{aligned}
$$

Summation spaltenweise nach den Rewards r_{t+k} durchführen , dh. zuerst die r_{t+1} mit den Gewichten $(1-\lambda)\lambda^k$ über $k = 0, 1, \ldots$ summieren ergibt den Wert 1 (geometrische Reihe), dann r_{t+2} mit den Gewichten $(1-\lambda)\gamma\lambda^k$ über $k = 1, 2, 3, \ldots$ ergibt den Wert $\gamma\lambda$, usw. mit r_{t+k} für $k \geq 3, 4, \ldots$.

$$
\begin{aligned}
\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) \quad &= \quad -V_t(s_t) + (\gamma\lambda)^0 \left[r_{t+1} + (1-\lambda)\,\gamma V_t(s_{t+1})\right] + \\
&\qquad (\gamma\lambda)^1 \left[r_{t+2} + (1-\lambda)\,\gamma V_t(s_{t+2})\right] + \\
&\qquad (\gamma\lambda)^2 \left[r_{t+3} + (1-\lambda)\,\gamma V_t(s_{t+3})\right] + \\
&\qquad (\gamma\lambda)^3 \left[r_{t+4} + (1-\lambda)\,\gamma V_t(s_{t+4})\right] + \\
&\qquad \dots \quad \dots \quad \dots \\
&= \quad (\gamma\lambda)^0 \left[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right] + \\
&\quad\;\; (\gamma\lambda)^1 \left[r_{t+2} + \gamma V_t(s_{t+2}) - V_t(s_{t+1})\right] + \\
&\quad\;\; (\gamma\lambda)^2 \left[r_{t+3} + \gamma V_t(s_{t+3}) - V_t(s_{t+2})\right] + \\
&\quad\;\; (\gamma\lambda)^3 \left[r_{t+4} + \gamma V_t(s_{t+4}) - V_t(s_{t+3})\right] + \\
&\qquad \dots \quad \dots \quad \dots \\
&= \quad \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}\,\delta_k = \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t}\,\delta_k
\end{aligned}
$$

Wir können somit für die Summe der Updates durch λ-Return schreiben:

$$
\begin{aligned}
\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) \mathbf{1}_{[s=s_t]} &= \alpha \sum_{t=0}^{T-1} \left(\sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \, \delta_k \right) \mathbf{1}_{[s=s_t]} \\
&= \alpha \sum_{t=0}^{T-1} \mathbf{1}_{[s=s_t]} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \, \delta_k .
\end{aligned}
$$

Nun die Updates des TD(0) Verfahrens: Zunächst gilt

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbf{1}_{[s=s_k]}$$

Einsetzen liefert nun

$$\begin{aligned}
\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) &= \sum_{t=0}^{T-1} \alpha\delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbf{1}_{[s=s_k]} \\
&= \alpha \sum_{k=0}^{T-1} \sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathbf{1}_{[s=s_t]} \delta_k \\
&= \alpha \sum_{t=0}^{T-1} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \mathbf{1}_{[s=s_t]} \delta_k \\
&= \alpha \sum_{t=0}^{T-1} \mathbf{1}_{[s=s_t]} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k
\end{aligned}$$

Sarsa(λ)

- Idee von Sarsa(λ) ist, den Sarsa-Algorithmus zum Erlernen der Q-Funktion mit der TD(λ) Methoden zu kombinieren.

- Statt der Variablen $e_t(s)$ für alle $s \in \mathcal{S}$ brauchen wir Variablen $e_t(s,a)$ für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$.

- Dann ersetzen wir $V(s)$ durch $Q(s,a)$ und $e_t(s)$ durch $e_t(s,a)$. Also

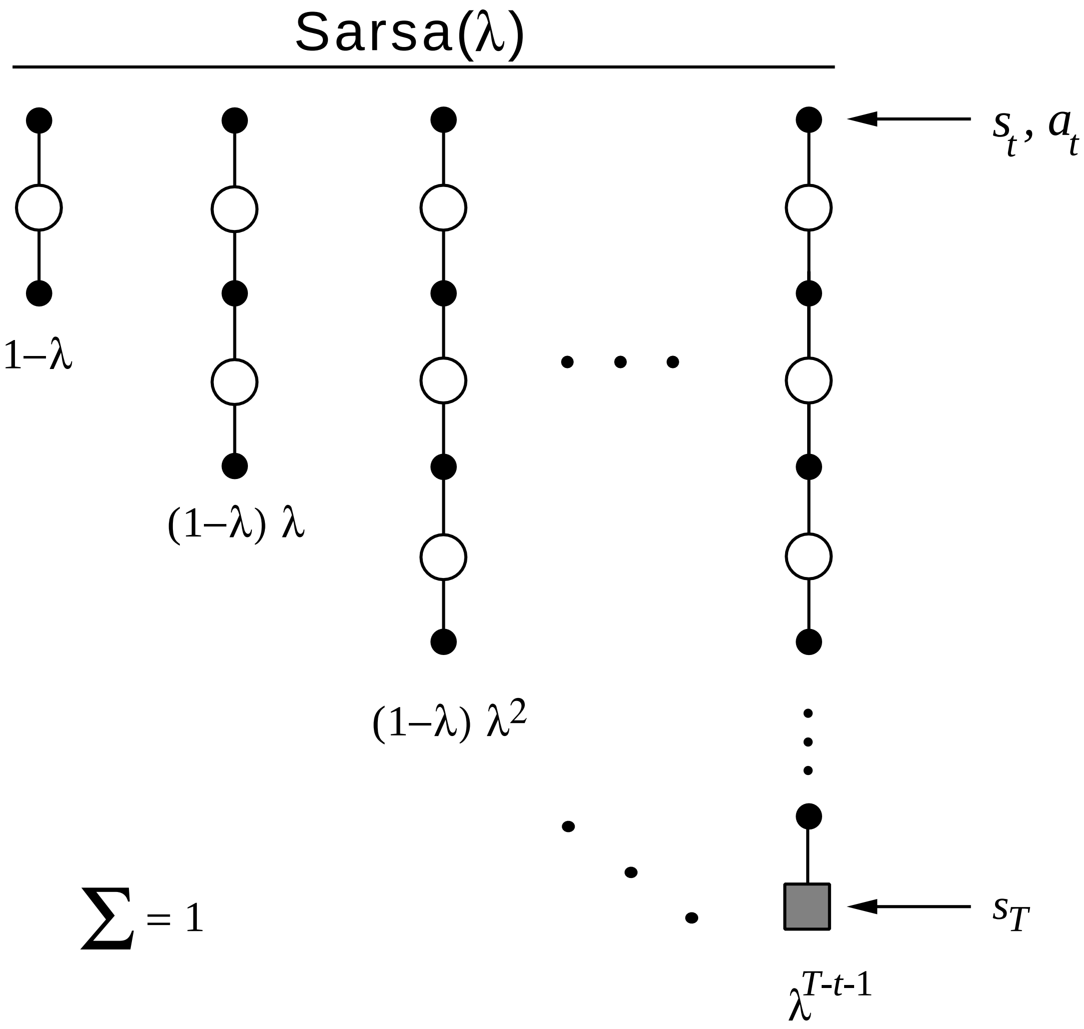
$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a) \quad \text{für alle } s \in \mathcal{S},\ a \in \mathcal{A}$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

und

$$e_t(s,a) = \begin{cases} \gamma \lambda e_{t-1}(s) + 1 & \text{falls } s_t = s \text{ und } a_t = a \\ \gamma \lambda e_{t-1}(s) & \text{sonst} \end{cases}$$

Sarsa Backup Diagramm



Sarsa Algorithmus (Q als Tabelle)

1. Initalize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ all s, a
2. Repeat (for each episode)

    Initialize s, a

    Repeat (for each step of episode):

    Take a, observe reward r, and next state s'

    Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)

    $\delta := r + \gamma Q(s', a') - Q(s, a)$

    $e(s, a) := e(s, a) + 1$

    For all s, a:

    $Q(s, a) := Q(s, a) + \alpha \delta e(s, a)$

    $e(s, a) := \lambda \gamma e(s, a)$

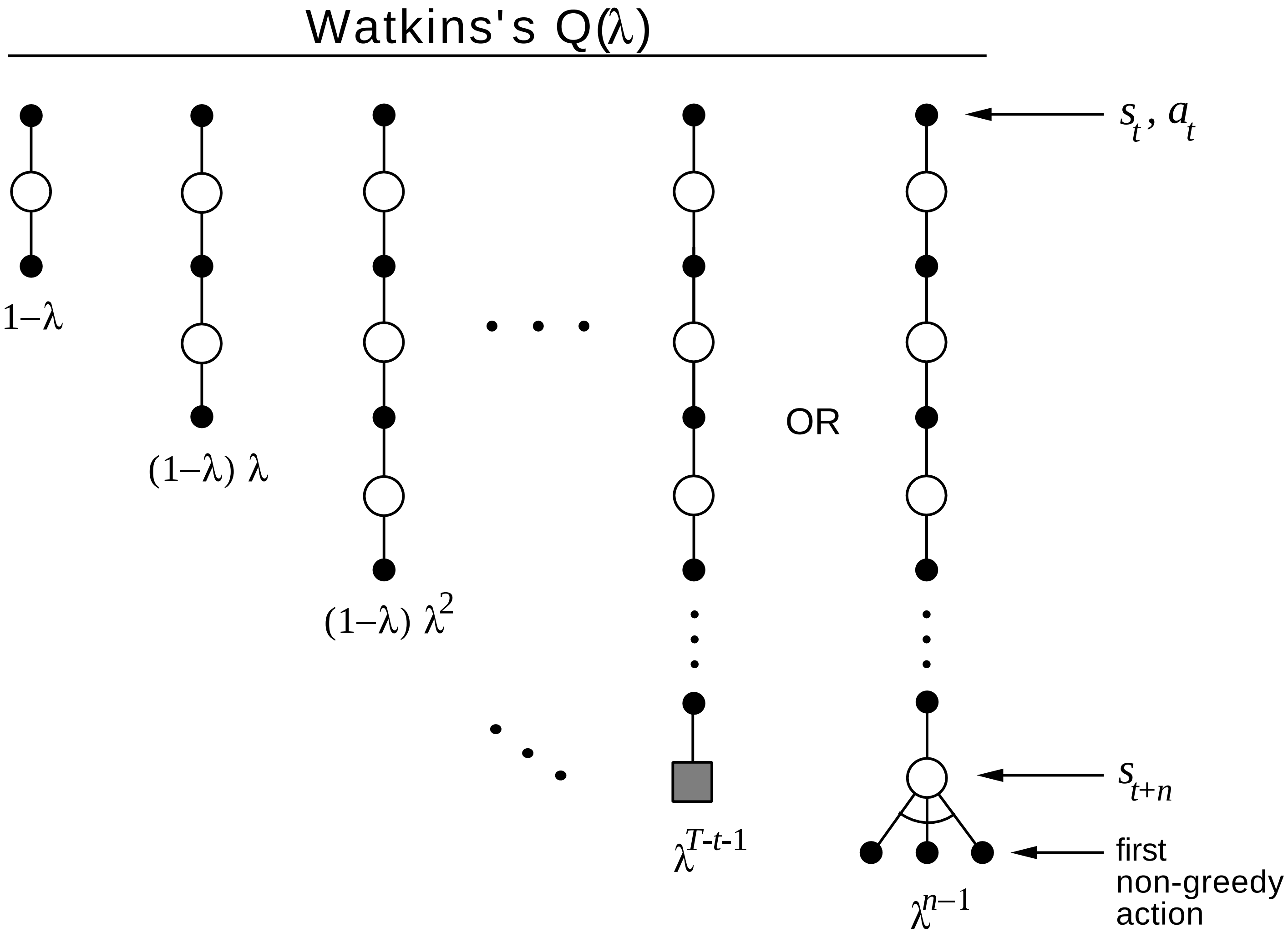
    $s := s'$; $a := a'$

    Until s is terminal

Q(λ)-Lernverfahren

- Es gibt 2 Varianten: Watkin's Q(λ) und Peng's Q(λ) Verfahren (Letzterer ist schwerer implementierbar, deshalb hier nur Watkin's Q-Lernverfahren).
- Q-Lernen ist ein Off-Policy Verfahren.
- Beim Q-Lernen folgt der Agent einer explorativen Policy (z.B. ϵ-Greedy Verfahren bzgl. der Q-Funktion) und adaptiert die Q-Funktion nach der Greedy-Policy (bzgl. der Q-Funktion).
- Hier muss in Betracht gezogen werden, dass der Agent explorative Aktionen durchführt, die keine Greedy Aktionen sind.
- Zum Erlernen der zur Greedy Policy gehörenden Q-Funktionen dürfen diese explorativen Aktionen nicht berücksichtigt werden.
- Deshalb werden die n-step Returns beim $Q(\lambda)$ Verfahren auch nur bis zum Auftreten der nächsten explorativen Aktion berücksichtigt, und nicht stets bis zum Ende einer Episode.

Q(λ) Backup-Diagramm (Watkins)



Q(λ)-Algorithmus (Q als Tabelle)

1. Initalize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ all s, a
2. Repeat (for each episode)

    Initialize s, a

    Repeat (for each step of episode):

    Take a, observe reward r, and next state s'

    Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)

    $a^* :=$ arg max$_b Q(s', b)$ (if a' ties for the max, then $a^* := a'$).

    $\delta := r + \gamma Q(s', a^*) - Q(s, a)$

    $e(s, a) := e(s, a) + 1$

    For all s, a:

    $Q(s, a) := Q(s, a) + \alpha \delta e(s, a)$

    if $a' = a^*$ then $e(s, a) := \lambda \gamma e(s, a)$ else $e(s, a) := 0$

    $s := s'$; $a := a'$

    Until s is terminal