Reinforcement Learning

Abe Vos

Mei 2022

Vorige keer

Markov Decision Process

- Statenruimte $\mathcal{S}$
- Acties $\mathcal{A}_s, \forall s \in \mathcal{S}$
- Overgangsfunctie $p(S_{t+1} = s' | S_t = s, A_t = a)$
- Beloningsfunctie $R(s, a, s')$

Policy/Value Iteration

- Bereken de waardefunctie
  - Verwachte totale beloning voor iedere staat
- Update beleid met waardefunctie
  - Kies de actie die naar de meest “waardevolle” staat leidt
- Maak gebruik van de overgangsfunctie

Volgende stappen

- Overgangsfunctie/beloningsfunctie niet beschikbaar
- Leer van ervaringen uit de omgeving

Monte Carlo Methodes

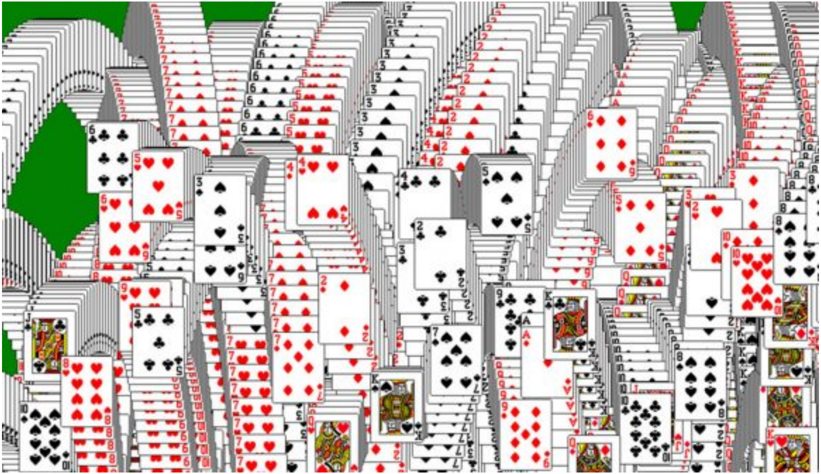


Figure 1: Patience

Schatten door simulatie

- Schudt de kaarten
- Probeer het spel uit te spelen
- Herhaal een paar duizend keer

Figure 2: Monte Carlo, Monaco

Omgekeerde probleem van kansrekening

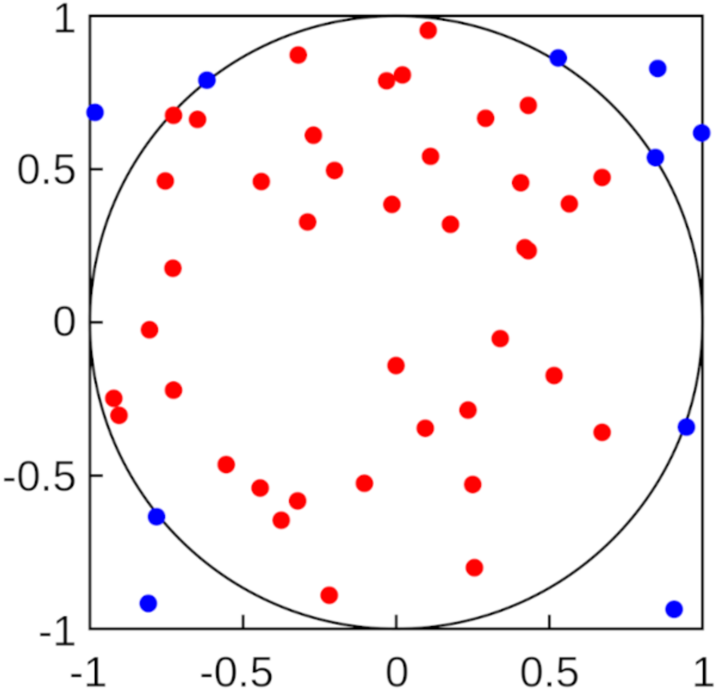
- Kansrekening: Gebruik deterministische methodes voor stochastische problemen
- Monte Carlo: Gebruik stochastische methodes voor deterministisch problemen

Voorbeeld: Bereken π

- Eenheidscirkel
  - Straal $r = 1$
- Oppervlakte $A = \pi = 4 \int_0^1 \sqrt(1 - x^2) dx$

Monte Carlo Integration

- Bereken een integraal $I = \int f(x)dx$ binnen een volume V
- Neem steekproeven $x_1, x_2, \cdots, x_N$
  - Uniform verdeelt in V
- $I \approx V \frac{1}{N} \sum_{n=1}^{N} f(x_n)$


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

Schat π

- $f(x) = 1$ als x binnen cirkel ligt, anders $f(x) = 0$
- Eenheidscirkel past in vierkant met oppervlakte 4
- 40 van de 50 punten binnen cirkel
- $4\frac{1}{50}40 = 3.2 \approx \pi$
- Meer simulaties geeft betere schattingen

Verwachte waarde

- X is som van twee dobbelstenen
- Verwachte waarde $\mathbb{E}[X] = \sum_{x \in \mathcal{X}} xp(X = X)$
  - Voor continue stochasten $\mathbb{E}[X] = \int_{-\infty}^{\infty} xp(X = x)dx$
- $p(X = x)$ is onbekend, gebruik Monte Carlo
  - Neem N steekproeven
  - $\mathbb{E}[X] \approx \frac{1}{N} \sum_{n=1}^{N} x_n$

Voorbeeld: Verwachte waarde van twee dobbelstenen

```
>>> import random
>>> N = 10000
>>> total = 0.0
>>> for n in range(N):
..      die_1 = random.randint(1, 6)
..      die_2 = random.randint(1, 6)
..      total += die_1 + die_2
>>> result = total / N
>>> print(result)
7.0029
```

Monte Carlo voor RL

- Policy Evaluation update
  - $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
  - Verwachte toekomstige beloning op staat s
- Overgangsfunctie is niet beschikbaar
- Gebruik simulaties van beleid π in omgeving voor Monte Carlo

Monte Carlo Prediction

```
Dict N, met N[s] = 0 voor elke staat s
Dict V, met V[s] = 0 voor elke staat s

Voor elke iteratie:
    Simuleer episode voor T stappen
    Sla alle staten en beloningen (S, R) op in een lijst

    G = 0
    Voor elke tijdsstap t in (T-1...0):
        S, R = episode[t]
        G = discount_factor * G + R

        Als S niet eerder is gezien deze episode:
            N[S] += 1
            V[S] += (G - V[S]) / N[S]
```

Toenemend gemiddelde

- Voor een reeks waardes $x_1, x_2, \cdots, x_T$, bereken gemiddelde na k waardes te hebben gezien
  - $\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j$
  - Gemiddelde wordt opnieuw berekend voor elke k
- Update gemiddelde μ_{k-1} met x_k:
  - $\mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
- MC Prediction update de schatting van de verwachte/gemiddelde totale beloning

Monte Carlo Methodes

- Levert optimale schatting van waardefunctie op
  - Net als Policy Evaluation
- Heeft geen dynamieken van de MDP nodig
  - *Model-free*

Tekortkomingen

- MC heeft hoge variantie
  - “Gemiddelde afwijking van het gemiddelde”
  - Meer simulaties nodig voor precieze schattingen
- MC Methodes zijn offline
  - Waardefunctie/beleid krijgt update aan eind van episode
  - Lange episodes $\rightarrow$ traag leren

Temporal Difference Learning

Temporal-Difference Learning

- Lijkt op MC
  - Leert van ervaring
  - Model-free
- Maar heeft extra voordelen:
  - Online learning
  - Leert van onvolledige episodes

TD(0)

```
Kies learning_rate lr
Dict V, met V[s] = 0 voor elke staat s

For elke episode:
    Initialiseer S

    Voor elke stap in de episode:
        Kies actie A met beleid gegeven S
        Voer A uit, observeer R en S'
        V(S) += lr * (R + discount_factor * V[S'] - V[S])
        S = S'
```

Target

- MC update: $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$
- TD(0) update:
  $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
- TD(0) gebruikt *schatting* van G_t als target
  - Bootstrapping

Bias/Variance

- $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is een *unbiased* schatting van $v_\pi(S_t)$
  - Hoe meer episodes $\rightarrow$ hoe preciezer de schatting
- De 'echte' TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is een *unbiased* schatting van $v_\pi(S_t)$
  - We willen $v_\pi(S_t)$ schatten, dus we hebben de echte waardefunctie nog niet
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is een *biased* schatting van $v_\pi(S_t)$
  - Ongeacht het aantal episodes, we gaan altijd afwijken van de ware waardefunctie

Bias/Variance (2)

- G_t is afhankelijk van *alle* acties, overgangen en beloningen in de episode
  - Kan veel verschillende waarden aannemen
  - Hoge variantie
- TD target is afhankelijk van 1 actie, overgang en beloning
  - Kan minder mogelijke waarden aannemen gegeven de huidige staat en actie
  - Lagere variantie

Monte Carlo vs Temporal Difference

- MC heeft hoge variantie, geen bias
  - Erg stabiel (convergeert ook goed met functie-benadering)
  - Niet afhankelijk van Markov eigenschap
- TD heeft lage variantie, wel bias
  - Vaak efficienter dan Monte Carlo
  - TD(0) convergeert naar $v_\pi(s)$ (maar niet altijd met functie-benadering)
  - Afhankelijk van Markov eigenschap

n-Step TD

- Monte Carlo gebruikt alle stappen in een episode
  - $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$
- TD(0) gebruikt een enkele stap
  - $G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$
- We kunnen ook n stappen gebruiken:
  - $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$
  - $V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$

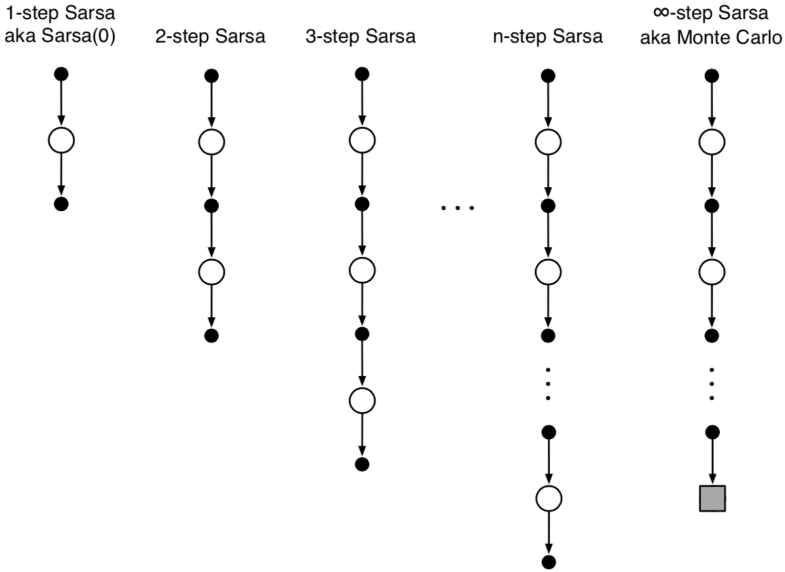


Figure 4: n-step TD

TD(λ)

- Combineer $G_t^{(n)}$ voor alle n
  - $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$
- Invloed van toekomstige stappen op heden:
  - Gewicht $(1-\lambda)\lambda^{n-1}$ wordt kleiner als $n \to \infty$
  - $0 \leq \lambda \leq 1$
  - $1-\lambda$ zorgt dat alle gewichten optellen naar 1
- Forward-view TD(λ)
  - $V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t))$

Eligibility Traces

- Aan welke bezochte staten moeten we waarde toeschrijven?
- Frequentie heuristiek: schrijf waarde toe aan veelbezochte staten
- Recentheid heuristiek: schrijf waarde toe aan recente staten
- Eligibility Traces combineren beide heuristieken:
  - $E_0(s) = 0$
  - $E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{I}(S_t = s)$

Backward View TD(λ)

- Gedurende elke tijdstap t, houd een Eligibility Trace bij voor elke staat $s \in \mathcal{S}$
- Update $V(s)$ voor elke staat s
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - $V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$
- $E_t(s) = 0$ zolang we s niet hebben gezien

Model-Free Control

Control in MDP

- Policy Iteration
  - Evalueer $V(s)$ a.d.h.v. $\pi(s)$
  - Verbeter $\pi(s)$ met $V(s)$
- Policy improvement heeft MDP model nodig
  - $\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
- Gebruik een kwaliteitsfunctie $Q(s, a)$
  - Schat waarde voor elk staat/actie paar
  - Policy Improvement is model-free: $\pi'(s) = \arg\max_a Q(s, a)$

Epsilon-Greedy Policy

- We willen blijven ontdekken
  - Exploration vs. exploitation
- Neem uniforme steekproef tussen 0 en 1: x
- Als $x \leq \varepsilon$ kies een willekeurige actie
- Anders kies actie $a^* = \arg\max_a Q(s, a)$

Sarsa

```
Dict Q, met Q[(s,a)] = 0 voor alle (s, a)

Voor elke episode:
    Initialiseer S
    Kies actie A gegeven S met beleid van Q

    Voor elke stap in de episode:
        Voer A uit, observeer R, S'
        Kies A' gegeven S' met beleid van Q
        Q[(S,A)] += lr * (R + discount * Q[(S',A')] - Q[(S,
        S = S'
        A = A'
```

Sarsa

- S, A, R, S', A'
- Temporal Differencing control
- Kan net als TD(λ) naar Sarsa(λ) generaliseren
  - Gebruik eligibility trace voor elk staat/actie paar

Sarsa convergentie

- Convergeert naar optimale kwaliteitsfunctie
- In een eindeloos lang trainingsproces:
  - Kies alle staten/acties oneindig vaak
  - Robbins-Monro Sequence:
    - $\sum_{t=1}^{\infty} \alpha_t = \infty$
    - $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$
    - $\alpha_t = \frac{\alpha}{t}$

On-policy vs. off-policy

- Sarsa is on-policy: gebruik het geleerde beleid om beslissingen te maken en te ontdekken
  - Leren door te doen
  - Exploratie gaat ten koste van vermogen om optimaal beleid te leren
- Off-policy: ontdek met een ε-greedy beleid; train met een greedy beleid
  - Leren door te kijken
  - ε-greedy kan blijven exploreren, terwijl optimaal beleid wordt geleerd
  - Kan oude ervaringen hergebruiken: data-efficient

Q-learning

```
Dict Q, met Q[(s,a)] = 0 voor alle (s, a)

Voor elke episode:
    Initialiseer S

    Voor elke stap in de episode:
        Kies A gegeven S met beleid van Q
        Voer A uit, observeer R, S'
        Q[(S,A)] += lr * (R + discount * max Q[(S',a)] - Q
        S = S'
```

Q-learning

- Het gebruikte beleid komt van $Q(S, A)$
  - Met ε-greedy
- Het geleerde beleid komt van $\max_a Q(S', a)$

Conclusie

Samenvatting

	Signaal	Evaluatie	Controle
DP	$p(s', r \mid s, a)$	Policy Evaluation	Policy/Value Iteration
MC	G	MC Prediction	MC Control
TD	R_t of G_t^λ	TD(0), TD(λ)	Sarsa, Q-learning

Problemen

- $Q(s, a)$ wordt snel groot bij veel staten en acties
- Methodes werken alleen voor discrete staat- en actieruimtes
- Volgende week: functiebenadering en deep RL