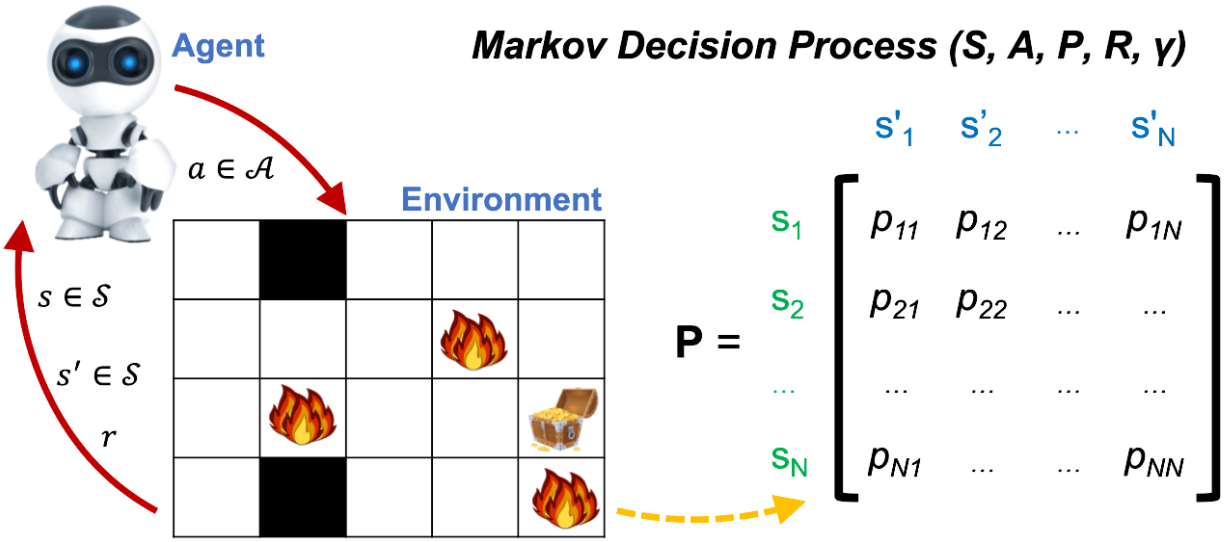
5.3.1 Monte Carlo Prediction 1

Tags

Materi sebelumnya: MDP



Algoritme Policy Iteration		
1:	Inisialisasi V dan π	
2:	**Ulangi**	
3:	Evaluasi V menggunakan π	(Eq. 1)
4:	Improve π menggunakan V	(Eq. 2)
5:	**Sampai** konvergen	

$$V_{t+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} P(s',r|s,a)[r + \gamma V_t(s')] \quad \text{.....(Eq. 1)}$$

$$Q_{\pi}(s,a) = \sum_{s',r} P(s',r|s,a)[r + \gamma V_{\pi}(s')] \quad \text{.....(Eq. 2)}$$

Algoritme Value Iteration

1: Inisialisasi V_0
2: **Ulangi**
3: Improve V_{t+1} menggunakan estimasi dari V_t (Eq. 3) (Persamaan 3)
4: **Sampai** konvergen

$$V_{t+1}(s) = \max_a \sum_{s',r} P(s', r|s, a)[r + \gamma V_t(s')] \quad \text{.....(Eq. 3)}$$

Frozen Lake Problem

Agent berjalan menuju tujuan yang jalurnya berupa es. Namun ada beberapa lubang. Es-nya licin, sehingga agent tidak bisa selalu bergerak sesuai arah yang diinginkannya.

Frozen Lake Problem

- Satu episode berhenti ketika agent mencapai tujuan (goal).
- Agent mendapat 1 reward ketika mencapai tujuan (goal) dan 0 untuk kondisi yang lain.
- Jika masuk ke lubang, agent hanya bisa bergerak searah dengan arah sebelum masuk lubang (peluang 1/3), dan kedua arah tegak lurus (masing-masing peluangnya 1/3).

S = starting point **F = frozen surface** **H = hole** **G = goal**

Aplikasi DP pada permasalahan Rill

Untuk mengaplikasikan algoritme DP, dibutuhkan pengetahuan tentang model environment.

Model tersebut adalah yang terkait matriks probabilitas transisi dan sistem reward-nya. **DP = Model-based algorithm.**

Meskipun matriks probabilitas transisi dapat ditentukan dari pengalaman, butuh sumber daya (waktu, tenaga, uang, dll) yang besar untuk mendapatkan pengalaman yang cukup.

Pada tahun 1950, ahli matematika bernama Claude Shannon menulis artikel tentang “How to Programme a Computer for Playing Chess”. Dalam artikel tersebut, dituliskan bahwa jumlah pergerakan yang mungkin untuk rata-rata permainan catur diestimasi sebesar 1 x 10120 langkah.

Bisakah environment catur tersebut dimodelkan? **Bisa**.

Efektifkah untuk dilakukan? **Untuk banyak kasus, tidak efektif**.

Apakah ada solusi mengimplementasikan RL tanpa memodelkan environment-nya secara lengkap? **Ada. Model free algorithm** → Monte Carlo.

Monte Carlo Prediction

Konsep Monte Carlo (MC)

- MC tidak mengambil pengetahuan lengkap dari environment.
- MC belajar dari experience, episode per episode, baik itu experience aktual, maupun simulasi.
- MC belajar dari episode-episode secara utuh dan independent, tidak bootstrapping.
- MC didefinisikan untuk jenis episodic environment.
- Ide utama dari MC: Value didapatkan dari rata-rata returns.
- Dengan semakin banyak returns, nilai rata-ratanya diharapkan konvergen pada expected value.
- Seluruh episode dipertimbangkan dalam MC
- Hanya satu pilihan setiap perpindahan state di MC, sedangkan DP mempertimbangkan semua probabilitas transisi pada setiap perpindahan state
- Estimasi-estimasi untuk semua state adalah independent di MC, tidak bootstrap
- Waktu yang dibutuhkan untuk mengestimasi suatu state tidak bergantung pada jumlah total state

Diskusi

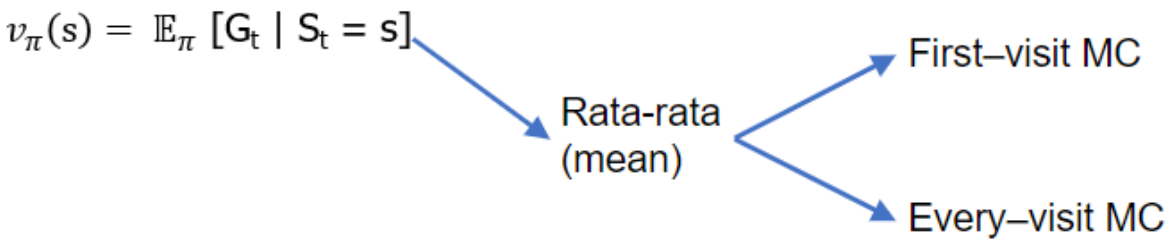
- Sebutkan perbedaan karakteristik DP dan MC!
- Mengapa penghitungan nilai rata-ratanya dari returns bisa menyebabkan iterasi MC konvergen pada expected value?

Elemen Algoritme MC

- **Goal ::** Belajar v_π melalui episode-episode dari experience yang patuh pada policy π:

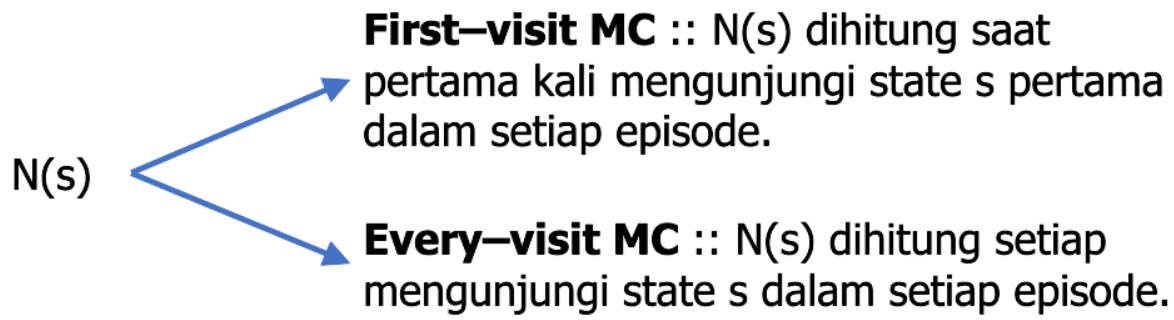
  $S_1, A_1, R_2, ..., S_t \sim \pi$
- **Return ::** Adalah ttal discounted reward:

  $G_t = R_{t+1} + \gamma Rt + 1 + ... + \gamma^{T-1} R_T$
- **Value Function ::** Adalah expected return:



Perhitungan Value Function di MC:

- Total counter $N(s) \leftarrow N(s) + 1$
- Total return $S(s) \leftarrow S(s) + G_t$
- Value $V(s) = S(s)/N(s)$
- $V(s)$ konvergen ke v_π sejauh $N(s)$ mendekati $\propto$



Algoritme MC (First-Visit)

Algoritme First-Visit MC

1. Input: sebuah policy untuk dievaluasi
2. nisialisasi:
3. , untuk semua
4. ← list kosong, untuk semua
5. **Sampai** konvergen

6. Generae episode berdasarkan
7. G ← 0
8. Loop untuk setiap langkah dari episode,
9. $G \leftarrow G + R_{t+1}$
10. Jika sampai pada first visit ke:
11. Append G ke
12. ← rata rata dari

Algoritme MC (Every-Visit)

Algoritme Every-Visit MC

1. **Input**: sebua policy untuk dievaluasi
2. **Inisialisasi:**
3. , untuk semua
4. ← list kosong, untuk semua
5. **Sampai** konvergen
6. Generate episode berdasarkan
7. $G \leftarrow 0$
8. Loop untuk setiap langkah dari episode,
9. $G \leftarrow G + R_{t+1}$
10. Append G ke
11. ← rata rata dari

Contoh

- Misalkan kita punya sebuah environment dengan dua state, yaitu A dan B. Kita telah melakukan observasi selama 2 episode dan menghasilkan urutan seperti ini :

A + 3 → A + 2 → B - 4 → A + 4 → B - 3 → terminate

B - 2 → A + 3 → B - 3 → terminate

- **A+3 →A+2** artinya adalah transition dari state A →A dengan reward = 3 untuk transisi tersebut.
- Kita akan coba mencari V(A) dan V(B) menggunakan Algoritma MC first visit dan juga every visit.

Penyelesaian First Visit: Calculating (VA)

A + 3 → A + 2 → B - 4 → A + 4 → B - 3 → terminate

B - 2 → A + 3 → B - 3 → terminate

Seperti yang bisa kita lihat kalau kita mempunyai 2 iterasi (episode) yang berbeda, kita akan menjumlahkan semua reward tiap episode dengan catatan reward akan dihitung setelah ***agent*** mengunjungi ***state A.***

Sum reward episode 1 = 3 + 2 + (-4) + 4 + (-3)

Sum reward episode 2 = 3 + (-3)

= 0

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yang di atas, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

V(A) = (Sum reward episode 1 + Sum reward episode 2) / Jumlah episode yang ada ***state A***

V(A) = (2+0) / 2

V(A) = 1

Calculating (VB)

Seperti yang bisa kita lihat kalau kita mempunyai 2 iterasi (episode) yang berbeda, kita menjumlahkan semua reward tiap episode dengan catatan reward akan dihitung setelah ***agent*** mengunjungi ***state B.***

Sum reward episode 1 = -4 + 4 + (-3)

= -3

Sum reward episode 2 = -2 + (3) + (-3)

= -2

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yang di atas, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

V(B) = (Sum reward episode 1 + Sum reward episode 2) / Jumlah episode yang ada ***state B***

V(B) = (-3 + (-2)) / 2

V(B) = -5/2 = -2.5

Penyelesaian Every Visit: Calculating

A + 3 → A + 2 → B - 4 → A + 4 → B - 3 → terminate

B - 2 → A + 3 → B - 3 → terminate

Berbeda dengan first visit, pada every visit kita akan menjumlahkan semua reward tiap episode dengan catatan kumulatif reward akan dihitung setiap ***agent*** mengunjungi ***state A***.

Sum reward episode 1 = (3 + 2 + (-4) + 4 + (-3)) + (2 + (-4) + 4 + (-3)) + (4 + (-3))

Sum reward episode 2 = 3 + (-3)

= 0

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yang di atas, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

V(A) = (Sum reward episode 1 + Sum reward episode 2) / Jumlah episode yang ada ***state A*** disemua episode

V(A) = (2 + 0) / 4

V(A) = 0.5

Calculating V(B)

A + 3 → A + 2 → B - 4 → A + 4 → B - 3 → terminate

B - 2 → A + 3 → B - 3 → terminate

Berbeda dengan first visit, pada every visit kita akan menjumlahkan semua reward tiap episode dengan catatan kumulatif reward akan dihitung setiap ***agent*** mengunjungi ***state B***.

Sum reward episode 1 = ((-4) + 4 + (-3)) + (-3)

= (-3) + (-3)

= -6

Sum reward episode 2 = (-2 + 3 - 3) + (-3)

= (-2) + (-3)

= -5

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yang di atas, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

V(B) = (Sum reward episode 1 + Sum reward episode 2) / Jumlah episode yang ada ***state B*** disemua episode

V(B) = ((-6) + (-5)) / 4/

V(B) = -2.75

Diskusi

- Bagimana perbedaan karakteristik eksplorasi dan eksploitasi dari first-visit MC dengan every-visit MC!

Monte Carlo Estimation & Control

MC Estimation

- Karena model tidak tersedia, perlu untuk mengestimasi action juga, selain hanya mengestimasi state-nya.
- Tidak seperti DP, pada Model Free Algorithm, MC, state saja tidak cukup untuk menentukan policy.

- Pada MC, action diperlukan dalam menentukan policy. Policy Evaluation Problem.
- Policy evaluation problem mengestimasi $q_\pi(s, a)$, yaitu expected return ketika mulai dari state s, melakukan action a, mengikuti policy π
- First-visit MC mengestimasi value dari pasangan state-action (s, a) dengan merata-rata returns, saat pertama kali menemui (visit) state s dan mengambil action a dalam suatu episode.
- Every-visit MC mengestimasi value dari pasangan state-action (s, a) dengan merata-rata retuns, setiap menemui (visit) state s mengambil action a dalam suatu episode
- Implikasinya, tidak semua state-action (s, a) akan ditemui (visited). Apakah masih bisa mendekati expected return, $q_\pi(s, a)$?
- Solusi: ***maintaining exploration***

Maintaining Exploration dilakukan dengan:

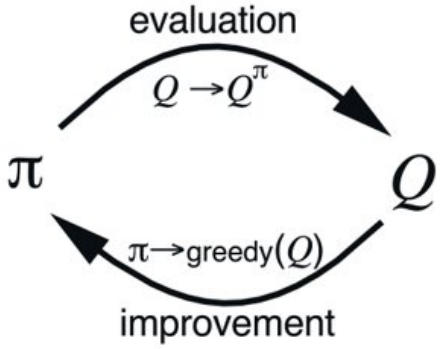
- Episode-episode memulai dengan sebuah pasangan state-action (s, a). Mekanisme pertama dari exploring starts.
- Setiap pasangan memiliki probabilitas yang tidak sama dengan nol untuk dipilih sebagai permulaan episode. Mekanisme kedua dari exploring starts.
- Semua pasangan state-action (s, a) memiliki probabilitas tidak sama dengan nol untuk memilih semua action pada setiap state.

http://rl-lab.com/gridworld-mc

MC Control

Policy Evaluation: Episode-episode dikerjakan dengan mekanisme exploring starts. Lalu, MC akan menghitung $q_{\pi k}$ untuk sembarang π_k

Policy Improvement: Untuk setiap action-value function q, greedy policy-nya, untuk setiap $s \in S$, memilih sebuah action yang merupakan action-value maksimal, yaitu: $\pi(s) = arg\ \max\ _a q(s,a)$. Policy improvement dilakukan dengan membentuk π_{k+1} sebagai greedy policy berdasarkan pada $q_{\pi k}$



$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$

On/Off Policy Monte Carlo

On-Policy MC

Konsep

- On-policy MC adalah algoritme Monte Carlo yang melakukan evaluasi atau improvisasi dari policy yang digunakan untuk membuat keputusan-keputusan.
- Pembahasan MC sebelumnya adalah on-policy MC.
- Lalu disini akan dibahas on-policy MC dengan policy dengan nama $\xi - greedy$.
- Dalam $\xi - greedy$ policy, hampir semua action yang dipilih, mempunyai action value estimasi yang maksimal, tetapi dengan probabilitas ξ dalam memilih action, tidak dengan random/acak.
- Untuk semua nongreedy action, maka: $\pi(a|S_t) \leftarrow \frac{\xi}{|A(s)|}$
  Yaitu saat $a \neq A*$
- Untuk semua greedy action, maka: $\pi(a|S_t) \leftarrow 1 - \xi + \frac{\xi}{|A(s)|}$

> Dalam hal ini, jika $\pi(a|S_t) \geq \frac{\xi}{|A(s)|}$ maka policy yang dikerjakan dinamakan $\xi - soft$ policy

Algoritme

Algoritme On-Policy First-Visit MC

1. **Parameter algoritme**: small
2. **Inisialisasi**:
3. ← policy

   , untuk semua
4. ← list kosong, untuk semua
5. **sampai** konvergen (untuk setiap episode)
6. Generate episode berdasarkan
7. $G \leftarrow 0$
8. Loop untuk setiap langkah dari episode,

9. $G \leftarrow G + R_{t+1}$
10. Jika sampai pada first visit ke ():
11. Append G ke
12. ← rata-rata dari
13. * ←
14. Untuk semua:
15. jika *
16. jika *

Off-Policy MC

Konsep

- Ada dilemma pada hampir semua metode kontrol. Ingin optimal tetapi juga harus bergerak tidak optimal, yaitu harus eksplorasi.
- Off–policy MC adalah algoritme Monte Carlo yang melakukan evaluasi atau improvisasi dari policy yang berbeda dalam meng-generate data (behavior policy), sedangkan target policy-nya sama.
- Targe policy adalah policy yang dipelajari oleh agent. Target policy ini adalah greedy policy yang patuh pada Q.
- Behavior policy (μ) adalah policy yang menggenerate behavior
- Probabilitas dari behavior policy untuk memilih setiap action, yang mungkin dipilih oleh target policy, harus **bukan nol**. Untuk memastikan hal tersebut, kita membutuhkan behavior policy yang **soft**.

Algoritme

Algoritme Off-Policy First-Visit MC

---

1. **Input:** sebuah policy untuk dievaluasi
2. **Inisialisasi**:

3. , untuk semua
4. , untuk semua, adalah greedy yang patuh pada Q
5. **Sampai** konvergen (untuk setiap episode)
6. b ← sembarang policy yang masih dalam lingkup
7. Generate episode berdasarkan
8. $G \leftarrow 0$
9. $W \leftarrow 1$
10. Loop untuk setiap langkah dari episode, selama W 0:
11. $G \leftarrow G + R_{t+1}$
12. Jika, maka keluar dari loop (lanjutkan episode berikutnya)
13. $W \leftarrow W$

Pemrograman Monte Carlo

Frozen Lake Problem

Bagaimana jika Frozen Lake Problem diselesaikan dengan Monte Carlo?

Monte Carlo Prediction

- Belajar meng-koding Monte Carlo dengan Python
- Studi kasus "Frozen Lake"
- Pemrograman akan menggunakan Google Colab

Summary

1. Monte Carlo bekerja dengan pengalaman dari sample, dan menggunakannya untuk belajar secara langsung tanpa model.
2. Pada first–visit MC, value function dihitung saat pertama kali mengunjungi state s pertama dalam setiap episode.
3. Pada every–visit MC, value function dihitung setiap mengunjungi state s dalam setiap episode.
4. Pada on-policy MC, agent berkomitmen untuk selalu eksplorasi dan mencoba mencari policy terbaik.
5. Pada off-policy MC, dilakukan evaluasi atau improvisasi dari policy yang berbeda dalam meng-generate data (behavior policy), sedangkan target policy-nya sama.

References

- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement Learning with Python. 2018