Bab 12 — Reinforcement Learning: Belajar dari Keputusan, Reward, dan Umpan Balik

Cara membaca bab ini

Bab 12 adalah puncak jalur AI-first sebelum capstone. Setelah Bab 1–11 membangun AI, ML, DL, Transformer, dan generative AI, bab ini menjawab pertanyaan yang berbeda: bagaimana mesin belajar bertindak lewat konsekuensi?

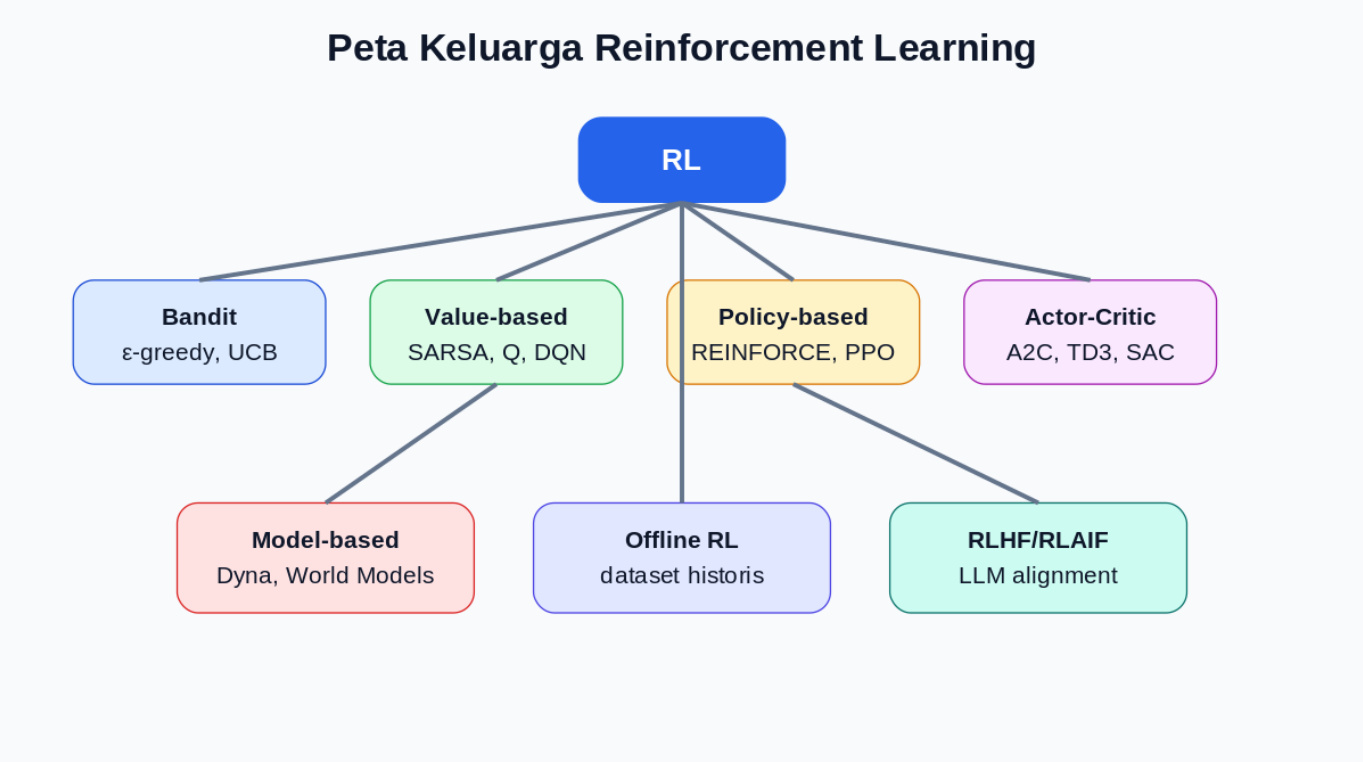
Reinforcement Learning (RL) bukan sekadar algoritma untuk game. RL adalah kerangka berpikir untuk keputusan berurutan: harga promo hari ini memengaruhi margin besok, rute gudang memengaruhi waktu pengiriman berikutnya, keputusan chatbot meminta klarifikasi memengaruhi kepuasan pengguna, dan feedback manusia pada jawaban LLM memengaruhi model generasi berikutnya. Sutton & Barto merumuskan RL sebagai belajar dari interaksi untuk memaksimalkan reward jangka panjang [R1].

Pola belajar setiap subbab:

> cerita masalah → state/action/reward → intuisi → rumus kecil → contoh hitung → praktik → tes cepat

Kontrak matematika bab ini: setiap persamaan penting tidak dibiarkan berdiri sendiri. Setelah rumus, selalu ada bagian Cara membacanya yang menerjemahkan simbol ke bahasa sehari-hari. Jangan membaca rumus RL seperti mantra. Baca dari kiri ke kanan sebagai cerita: “nilai lama diperbaiki oleh selisih antara kenyataan baru dan dugaan lama”. Jika sebuah simbol muncul berulang, maknanya konsisten: S untuk state, A untuk action, R untuk reward, γ untuk kepedulian pada masa depan, α untuk ukuran langkah belajar, dan π untuk policy.

Batas cakupan: bab ini mendalam tetapi tetap buku pengantar komersial. Kita membahas fondasi tabular, bandit, MDP, DP, Monte Carlo, TD, SARSA, Q-learning, DQN/deep Q-network, policy gradient, actor-critic, SAC, offline/batch RL, game theory, self-play, RLHF/RLAIF, Hugging Face TRL, dan risiko. Implementasi produksi robotika, simulator fisik besar, dan training GPU skala industri ditunda ke edisi lanjutan.



Peta keluarga reinforcement learning

Subbab 1 — Cerita pembuka: dari promo warung sampai LLM modern

Inti subbab: RL belajar memilih aksi yang menghasilkan reward jangka panjang, bukan hanya jawaban benar per contoh.

Bayangkan pemilik warung kopi di Yogyakarta. Ia bisa memberi diskon 0%, 10%, atau 20%. Diskon besar mungkin menaikkan penjualan hari ini, tetapi menurunkan margin dan membuat pelanggan menunggu diskon. Diskon kecil menjaga margin, tetapi mungkin kalah dari pesaing. Keputusan hari ini mengubah data besok: stok, loyalitas, arus kas, dan perilaku pelanggan.

Ini berbeda dari supervised learning. Pada supervised learning, kita punya contoh:

> fitur pelanggan → label beli/tidak beli

Pada RL, agent memilih aksi, menerima reward, lalu menghadapi state baru:

> state hari ini → aksi promo → reward → state besok → aksi lagi → ...

Contoh lain:

Domain	State	Action	Reward
UMKM	stok, cuaca, hari, harga pesaing	diskon, bundling, tanpa promo	laba, repeat order
Game Pong	posisi bola, paddle, kecepatan	naik, diam, turun	menang/kalah poin
Gudang	posisi picker, antrean order	pilih order berikutnya	waktu selesai, penalti telat
Tutor AI	jawaban siswa, histori salah	beri hint, contoh, kuis	peningkatan skor
LLM alignment	prompt, jawaban kandidat	pilih/ubah jawaban	preferensi manusia/AI

Perbedaan besar: reward sering tertunda. Aksi yang terlihat buruk sekarang bisa bagus nanti. Warung yang tidak memberi diskon hari ini mungkin kehilangan transaksi, tetapi melatih pelanggan menghargai kualitas. Sebaliknya, diskon agresif terlihat baik hari ini tetapi merusak margin.

Sutton & Barto menekankan tiga ciri RL [R1]:

1. Closed-loop: aksi agent mengubah input masa depan.

2. Tidak ada jawaban benar langsung: agent harus mencoba.

3. Konsekuensi tertunda: reward masa depan penting.

Mari buat ceritanya lebih hidup. Pada Senin pagi, pemilik warung mencoba diskon 20%. Penjualan melonjak, tetapi stok susu habis sebelum jam makan siang. Pada Selasa, beberapa pelanggan datang lagi, tetapi kecewa karena menu favorit kemarin tidak tersedia. Pada Rabu, pemilik warung mencoba bundling kopi + roti dengan diskon kecil. Penjualan tidak setinggi Senin, tetapi margin lebih sehat dan stok lebih terkendali. Jika hanya melihat reward Senin, diskon 20% terlihat menang. Jika melihat tiga hari, bundling mungkin lebih baik. Inilah alasan RL tidak cukup bertanya “aksi mana yang menghasilkan reward terbesar sekarang?”, melainkan “aksi mana yang membuat lintasan keputusan berikutnya semakin baik?”.

Di sinilah RL terasa berbeda dari model prediksi biasa. Model prediksi mungkin berkata: “pelanggan tipe A punya peluang beli 70%”. RL harus menjawab: “jika saya memberi voucher sekarang, apakah pelanggan akan membeli tanpa voucher minggu depan, atau justru menunggu voucher terus?”. Pertanyaan kedua lebih sulit karena aksi mengubah perilaku masa depan.

Tes cepat subbab 1

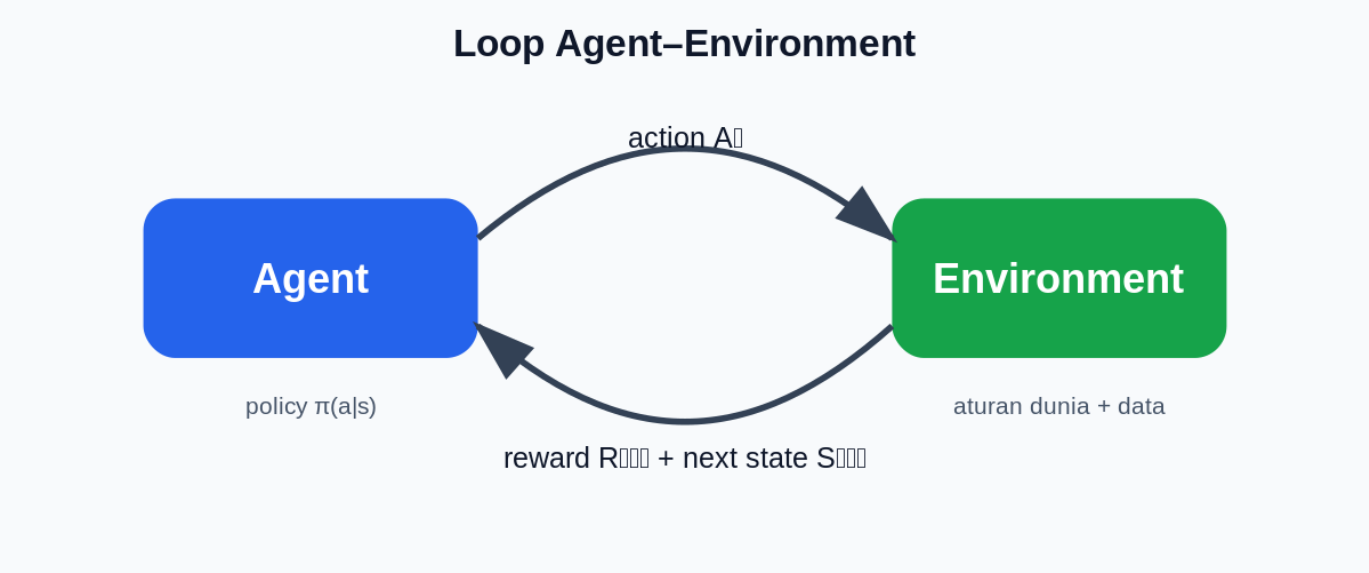
1. Mengapa RL berbeda dari supervised learning?

2. Beri contoh reward tertunda di bisnis Indonesia.

3. Mengapa “reward hari ini” belum tentu sama dengan “tujuan jangka panjang”?

Subbab 2 — Agent, environment, state, action, reward

Inti subbab: RL dimulai dari pemisahan agent dan environment.



Loop agent-environment

Pada waktu `t`:

Agent melihat state S_t
Agent memilih action A_t
Environment memberi reward R_{t+1}
Environment berpindah ke state S_{t+1}

Notasi utama:

	Arti	Contoh warung	
	state pada waktu t	stok, hari, cuaca, trafik	
	aksi	diskon 0/10/20%	
	reward setelah aksi	laba harian	
	s)`	policy	peluang memilih diskon a pa s
	discount factor	seberapa peduli masa depan	
	return	total reward masa depan	

Reward hypothesis: tujuan dapat dipikirkan sebagai memaksimalkan ekspektasi jumlah reward kumulatif [R1]. Ini kuat, tetapi berbahaya jika reward salah desain. Jika reward chatbot hanya “jumlah jawaban cepat”, agent bisa menjawab asal cepat. Jika reward sekolah hanya “nilai ujian”, agent bisa mendorong hafalan tanpa pemahaman.

Return discounted:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

Cara membacanya: `G_t` dibaca “return mulai waktu t”. Ia adalah reward langkah berikutnya `R_{t+1}`, ditambah reward dua langkah lagi yang didiskon oleh γ, ditambah reward tiga langkah lagi yang didiskon oleh γ^2, dan seterusnya. Pangkat γ makin besar untuk reward yang makin jauh agar masa depan jauh tidak dihitung sama kuatnya dengan reward yang dekat.

Jika $\gamma=0$, agent hanya peduli reward langsung. Jika γ mendekati 1, agent lebih sabar.

Cerita angka: jika pemilik toko memakai $\gamma=0$, ia seperti manajer yang hanya mengejar omzet hari ini. Jika memakai $\gamma=0,99$, ia seperti manajer yang sangat memikirkan loyalitas pelanggan dan kesehatan brand. Tidak ada angka γ yang selalu benar. Untuk episode pendek seperti game kecil, $\gamma=0,9$ sering cukup. Untuk proses bisnis jangka panjang, γ tinggi lebih masuk akal, tetapi evaluasi menjadi lebih sensitif karena reward jauh ikut dihitung.

Contoh hitung: reward tiga hari $[100, 50, 30]$, $\gamma=0,9$.

$$G_0 = 100 + 0,9\times50 + 0,9^2\times30$$
$$= 100 + 45 + 24,3$$
$$= 169,3$$

Cara membacanya: reward hari pertama dihitung penuh, reward hari kedua dikalikan $0,9$, dan reward hari ketiga dikalikan $0,9^2$. Karena $0,9^2=0,81$, reward 30 pada hari ketiga hanya menyumbang 24,3 pada return hari ini.

Tes cepat subbab 2

1. Tentukan state/action/reward untuk aplikasi rekomendasi menu kantin.

2. Hitung return untuk reward $[10, 0, 20]$ dengan $\gamma=0,5$.

3. Mengapa reward harus berada “di luar kontrol langsung agent”?

Subbab 3 — Bandit: RL paling sederhana untuk eksperimen keputusan

Inti subbab: multi-armed bandit adalah RL tanpa state berpindah; cocok untuk A/B testing adaptif, rekomendasi promo, dan optimasi sales awal.

Bayangkan tiga promo:

A: tanpa diskon
B: diskon 10%
C: bundling kopi + roti

Kita belum tahu mana paling menguntungkan. Jika selalu memilih promo yang saat ini tampak terbaik, kita mengeksploitasi. Jika mencoba promo lain, kita mengeksplorasi. Ini dilema eksplorasi-eksploitasi [R1][R5].

Estimasi nilai aksi:

$$Q(a) = \text{rata-rata reward saat aksi } a \text{ dipilih}$$

Cara membacanya: $Q(a)$ adalah “nilai dugaan untuk aksi a”. Jika aksi a adalah promo bundling, maka $Q(a)$ adalah rata-rata laba yang pernah kita lihat ketika bundling dipakai. Pada bandit sederhana, Q bukan kebenaran mutlak; ia hanya catatan pengalaman yang makin membaik saat data bertambah.

Update incremental:

$$Q_{baru}(a) = Q_{lama}(a) + \alpha \times (\text{reward} - Q_{lama}(a))$$

Cara membacanya: nilai baru sama dengan nilai lama ditambah koreksi. Koreksinya adalah α dikali error. Error adalah reward yang baru diterima dikurangi dugaan lama. Jika reward baru lebih tinggi dari dugaan, Q naik. Jika reward baru lebih rendah, Q turun. α mengatur seberapa besar kita mempercayai pengalaman terbaru.

Contoh hitung: dugaan laba bundling $Q\_lama=50$, laba hari ini $reward=62$, dan $\alpha=0,1$.

$$Q_{baru} = 50 + 0,1 \times (62 - 50)$$
$$= 50 + 1,2$$
$$= 51,2$$

Cara membacanya: meskipun hari ini untung 62, kita tidak langsung mengubah dugaan menjadi 62. Kita hanya menaikkan dugaan dari 50 ke 51,2 karena satu hari data belum cukup untuk yakin.

Konsep penting lain adalah regret: selisih antara reward yang kita peroleh dan reward yang mungkin diperoleh jika sejak awal selalu memilih aksi terbaik.

$$\text{Regret}_T = T \times \mu^* - \Sigma_{t=1}^{T} R_t$$

Cara membacanya: selama T percobaan, dunia ideal memberi T kali reward rata-rata aksi terbaik μ^*. Dunia nyata memberi jumlah reward aktual ΣR_t. Selisihnya adalah harga belajar. Eksplorasi memang membuat kita kadang memilih aksi yang belum tentu terbaik, tetapi tanpa eksplorasi regret jangka panjang bisa lebih besar karena kita tidak pernah menemukan aksi unggul.

Bentuk ini muncul berulang di RL:

$$\text{new estimate} = \text{old estimate} + \text{step size} \times \text{error}$$

Cara membacanya: setiap algoritma belajar dengan pola “dugaan baru = dugaan lama + sedikit koreksi”. `step size` adalah seberapa besar koreksi diterima, sedangkan `error` adalah selisih antara kenyataan baru dan dugaan lama.

Strategi bandit penting:

Strategi	Ide	Kapan berguna
Greedy	pilih reward estimasi tertinggi	baseline, risiko terjebak
ε-greedy	kadang acak	sederhana dan kuat
Optimistic initial values	mulai dengan Q tinggi	mendorong eksplorasi awal
UCB	pilih estimasi + bonus ketidakpastian	eksperimen terukur [R5]
Thompson sampling	sampling dari posterior	A/B testing Bayesian [R6]
Contextual bandit	ada konteks pengguna	rekomendasi/promo personal

UCB secara intuitif:

$$\text{pilih aksi} = \text{nilai rata-rata} + \text{bonus jarang dicoba}$$

Cara membacanya: skor UCB bukan hanya “aksi mana yang rata-ratanya paling tinggi”, tetapi “aksi mana yang menjanjikan setelah menambahkan bonus ketidakpastian”. Aksi yang jarang dicoba mendapat bonus agar agent tidak terlalu cepat mengabaikannya.

Bentuk umum UCB:

$$A_t = \operatorname{argmax}_a \left[Q_t(a) + c \times \sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

Cara membacanya: pada waktu t, pilih aksi a dengan skor terbesar. Skor terdiri dari dua bagian. $Q_t(a)$ adalah rata-rata reward aksi itu sejauh ini. Bagian $\sqrt{\ln(t)/N_t(a)}$ adalah bonus eksplorasi: makin jarang aksi dicoba ($N_t(a)$ kecil), bonus makin besar. c adalah kenop keberanian eksplorasi. Jika c terlalu kecil, agent cepat serakah. Jika c terlalu besar, agent terlalu lama mencoba-coba.

Untuk Thompson sampling pada bandit Bernoulli, kita sering menyimpan distribusi Beta:

$$\theta_a \sim \text{Beta}(\text{sukses}_a + 1, \text{gagal}_a + 1)$$

Cara membacanya: untuk setiap aksi a, kita tidak menyimpan satu angka pasti, tetapi distribusi keyakinan tentang peluang sukses θ_a. Jika promo A sukses 8 kali dan gagal 2 kali, distribusinya lebih optimistis daripada promo B yang sukses 2 kali dan gagal 8 kali. Agent mengambil sampel dari distribusi itu, lalu memilih aksi dengan sampel tertinggi. Exploration muncul alami karena aksi yang datanya sedikit masih punya ketidakpastian besar.

Tes cepat subbab 3

1. Apa beda greedy dan ε-greedy?

2. Mengapa promo yang awalnya terlihat buruk tetap perlu dicoba beberapa kali?

3. Kapan contextual bandit lebih cocok daripada bandit biasa?

Subbab 4 — Eksplorasi dan Eksploitasi: Dilema Terpenting dalam RL

Inti subbab: agent RL harus menyeimbangkan dua kebutuhan yang saling tarik-menarik: eksplorasi untuk belajar informasi baru dan eksploitasi untuk memakai pilihan terbaik yang sudah diketahui.

Eksplorasi vs eksploitasi

Dilema eksplorasi–eksploitasi adalah salah satu alasan RL terasa berbeda dari supervised learning. Pada supervised learning, dataset sudah tersedia. Model tidak perlu “mengorbankan” sebagian prediksi untuk mencari data baru. Pada RL, data muncul karena agent bertindak. Jika agent tidak pernah mencoba aksi baru, ia tidak pernah tahu apakah ada aksi yang lebih baik. Tetapi jika agent terlalu sering mencoba-coba, reward jangka pendek bisa buruk dan sistem nyata bisa terganggu.

Bayangkan mahasiswa baru di kota baru. Ia menemukan satu warung nasi goreng yang lumayan: harga terjangkau, rasa enak, jarak dekat. Jika setiap malam ia makan di warung itu, ia mengeksploitasi pengetahuan yang sudah ada. Ia aman, murah, dan kenyang. Tetapi ia mungkin tidak pernah menemukan warung soto yang jauh lebih enak di gang sebelah. Jika setiap malam ia mencoba tempat baru, ia mengeksplorasi. Ia bisa menemukan permata tersembunyi, tetapi juga bisa kecewa, membayar mahal, atau pulang lapar. Keputusan cerdas bukan “selalu coba baru” atau “selalu pakai yang lama”, melainkan mengatur kapan mencoba dan kapan memanen.

Dalam bisnis, analoginya lebih serius. Marketplace bisa menampilkan produk yang sudah terbukti laku kepada pelanggan. Itu eksploitasi. Tetapi jika hanya produk populer yang selalu ditampilkan, penjual baru tidak pernah mendapat kesempatan, katalog menjadi tidak sehat, dan sistem mungkin kehilangan produk yang sebenarnya cocok untuk segmen tertentu. Menampilkan sebagian produk baru adalah eksplorasi. Ada biaya jangka pendek karena rekomendasi mungkin kurang optimal, tetapi ada nilai informasi untuk masa depan.

4.1 Dua jenis nilai: reward sekarang dan nilai informasi

Eksploitasi mengejar reward berdasarkan pengetahuan saat ini:

$$A_t = \arg\max_a Q_t(a)$$

Cara membacanya: pada waktu `t`, agent memilih aksi `a` yang memiliki nilai estimasi `Q_t(a)` paling besar. Ini adalah strategi greedy. Ia berkata: “berdasarkan pengalaman sejauh ini, pilih yang terbaik.”

Eksplorasi mengejar informasi yang belum lengkap. Pada aksi yang jarang dicoba, $Q_t(a)$ mungkin belum akurat. Aksi itu bisa terlihat buruk hanya karena datanya sedikit. Maka agent perlu menilai bukan hanya rata-rata reward, tetapi juga ketidakpastian.

Cara intuitif memecah nilai aksi:

```
nilai keputusan = nilai reward saat ini + nilai informasi untuk masa depan
```

Cara membacanya: sebuah aksi bisa dipilih bukan karena reward langsungnya paling tinggi, melainkan karena aksi itu mengurangi ketidakpastian. Informasi yang diperoleh hari ini dapat membantu ratusan keputusan berikutnya.

Contoh angka sederhana: promo A sudah dicoba 100 kali dengan rata-rata laba Rp50 ribu. Promo B baru dicoba 2 kali dengan rata-rata Rp45 ribu. Greedy memilih A. Tetapi karena B baru dicoba 2 kali, rata-ratanya belum stabil. Jika B sebenarnya bernilai Rp60 ribu, terlalu cepat menolak B akan mahal dalam jangka panjang.

4.2 Regret: harga dari belum tahu

Regret mengukur harga belajar. Rumus bandit sederhana:

```
Regret_T = T μ* - Σ_{t=1}^{T} R_t
```

Cara membacanya: μ^* adalah reward rata-rata aksi terbaik jika kita sudah tahu dari awal. Jika ada T percobaan, dunia ideal memberi $T\ \mu^*$. Dunia nyata memberi jumlah reward aktual ΣR_t. Selisihnya disebut regret. Regret bukan selalu tanda gagal; sebagian regret adalah biaya wajar untuk belajar. Yang buruk adalah regret yang terus membesar karena agent tidak pernah belajar memilih aksi baik.

Contoh UMKM: promo terbaik sebenarnya memberi laba rata-rata Rp60 ribu per hari. Selama 10 hari, sistem eksperimen menghasilkan laba total Rp520 ribu. Regret-nya:

```
Regret_10 = 10 × 60 - 520
          = 600 - 520
          = 80
```

Cara membacanya: dibanding dunia ideal yang langsung tahu promo terbaik, kita “membayar” Rp80 ribu untuk proses belajar. Jika setelah itu agent menemukan promo terbaik dan memakainya selama bulan-bulan berikutnya, biaya Rp80 ribu bisa sangat masuk akal.

4.3 ε-greedy: strategi sederhana yang mudah dijelaskan

Strategi ε-greedy memakai aturan:

```
Dengan peluang ε: pilih aksi acak
Dengan peluang 1-ε: pilih aksi dengan Q terbesar
```

Cara membacanya: ε adalah persentase keberanian mencoba. Jika $\varepsilon=0,1$, agent mengeksplorasi sekitar 10% waktu dan mengeksploitasi sekitar 90% waktu. Nilai ε kecil membuat agent stabil tetapi bisa kurang mencoba. Nilai ε besar membuat agent banyak belajar tetapi reward pendek bisa turun.

Bentuk probabilitasnya untuk k aksi:

```
P(A_t=a) = ε/k + (1-ε) jika a adalah aksi greedy
P(A_t=a) = ε/k         jika a bukan aksi greedy
```

Cara membacanya: semua aksi mendapat peluang kecil ε/k dari bagian eksplorasi acak. Aksi greedy mendapat tambahan $1-\varepsilon$ karena dipilih saat agent mengeksploitasi. Jika ada 4 aksi dan $\varepsilon=0,2$, setiap aksi mendapat peluang eksplorasi $0,2/4=0,05$. Aksi terbaik mendapat tambahan $0,8$, sehingga totalnya $0,85$.

Namun ε-greedy punya kekurangan: eksplorasi acaknya buta. Ia mencoba aksi yang hampir pasti buruk sama seringnya dengan aksi yang masih menjanjikan. Dalam sistem nyata, ini bisa boros. Karena itu, strategi seperti UCB dan Thompson sampling mencoba eksplorasi yang lebih “cerdas”.

4.4 ε decay: belajar banyak di awal, lebih stabil di akhir

Sering kali, kita ingin agent berani mencoba pada awal training, lalu makin stabil setelah cukup pengalaman. Ini disebut ε decay.

$$\varepsilon_t = \max(\varepsilon_{min}, \varepsilon_0 \times decay^t)$$

Cara membacanya: ε_t adalah nilai epsilon pada waktu t. Agent mulai dari ε_0, lalu nilainya dikalikan decay berulang-ulang. Fungsi max memastikan epsilon tidak turun di bawah ε_{min}, sehingga agent tetap punya sedikit eksplorasi.

Contoh: $\varepsilon_0=1{,}0$, $decay=0{,}99$, $\varepsilon_{min}=0{,}05$. Pada awal training, agent hampir sepenuhnya eksploratif. Setelah banyak episode, epsilon mendekati 0,05. Artinya agent tetap mencoba aksi acak sekitar 5% waktu agar tidak terlalu kaku jika environment berubah.

Dalam bisnis, ε decay mirip fase peluncuran produk. Minggu pertama, tim mencoba banyak variasi headline, harga, bundling, dan channel. Setelah data cukup, variasi dikurangi dan campaign terbaik dipakai lebih sering. Tetapi sebagian kecil eksperimen tetap berjalan karena pasar berubah.

4.5 Optimistic initial values: optimis agar mau mencoba

Cara lain mendorong eksplorasi adalah memberi nilai awal yang optimistis:

$$Q_0(a) = \text{nilai tinggi untuk semua aksi}$$

Cara membacanya: sebelum punya data, semua aksi dianggap sangat menjanjikan. Ketika aksi dicoba dan reward nyata ternyata lebih rendah, nilainya turun. Aksi yang belum dicoba tetap tampak menarik karena nilai optimistisnya belum “dikoreksi” oleh kenyataan.

Contoh: tiga promo semuanya diberi $Q_0=100$, padahal laba realistis sekitar 40–60. Setelah promo A dicoba dan menghasilkan 45, nilai A turun. Promo B dan C masih bernilai 100, sehingga agent terdorong mencoba B dan C. Dengan cara ini, agent mengeksplorasi tanpa memilih acak secara eksplisit.

Kelemahannya: jika nilai awal terlalu optimistis, agent bisa terlalu lama mencoba hal buruk. Jika reward scale berubah, nilai awal harus disesuaikan. Strategi ini cocok untuk task stasioner sederhana, tetapi perlu hati-hati pada environment yang berubah.

4.6 UCB: eksplorasi berbasis ketidakpastian

UCB memilih aksi dengan skor:

$$score_t(a) = Q_t(a) + c\sqrt{\ln(t) / N_t(a)}$$
$$A_t = \text{argmax}_a\ score_t(a)$$

Cara membacanya: skor aksi adalah estimasi reward $Q_t(a)$ ditambah bonus ketidakpastian. Bonus besar jika aksi jarang dicoba ($N_t(a)$ kecil). Bonus mengecil jika aksi sudah sering dicoba. $\ln(t)$ membuat dorongan eksplorasi tumbuh pelan seiring waktu, sedangkan c mengatur seberapa besar agent menghargai ketidakpastian.

Analogi: seorang kepala cabang melihat dua strategi sales. Strategi A sudah dipakai 100 kali dan rata-rata menghasilkan 50 lead. Strategi B baru dipakai 3 kali dan rata-rata menghasilkan 48 lead. Greedy memilih A. UCB bertanya: “Apakah B cukup belum pasti sehingga layak diuji lagi?” Jika iya, B mendapat bonus dan bisa dipilih. Jika setelah banyak percobaan B tetap biasa saja, bonusnya mengecil dan agent kembali mengeksploitasi A.

4.7 Thompson sampling: eksplorasi sebagai undian keyakinan

Thompson sampling tidak menambahkan bonus eksplisit. Ia menyimpan keyakinan probabilistik, lalu mengambil sampel dari keyakinan itu.

Untuk reward sukses/gagal, bentuk sederhana:

θ_a ~ Beta(α_a, β_a)
Pilih aksi dengan sampel θ_a terbesar

Cara membacanya: setiap aksi a punya distribusi Beta yang menggambarkan keyakinan tentang peluang suksesnya. α_a naik saat aksi sukses, β_a naik saat aksi gagal. Agent mengambil satu sampel θ_a dari tiap aksi, lalu memilih aksi dengan sampel tertinggi. Aksi yang datanya sedikit punya distribusi lebih lebar, sehingga kadang menang undian dan dieksplorasi.

Jika promo A sudah punya 80 sukses dan 20 gagal, distribusinya sempit di sekitar 80%. Jika promo B baru punya 2 sukses dan 1 gagal, distribusinya lebar. B bisa saja menang sampling meskipun datanya sedikit, tetapi jika setelah dicoba berkali-kali performanya buruk, distribusinya akan turun sendiri.

4.8 Eksplorasi aman: jangan menjadikan pengguna sebagai korban eksperimen

Eksplorasi di simulator relatif aman. Eksplorasi di dunia nyata harus punya guardrail. Dalam sales, eksplorasi harga tidak boleh merugikan pelanggan atau melanggar regulasi. Dalam layanan publik, eksplorasi chatbot tidak boleh memberi jawaban medis/hukum berisiko. Dalam fintech, eksplorasi keputusan kredit tidak boleh menghasilkan diskriminasi.

Checklist eksplorasi aman:

[ ] Batasi aksi yang boleh dieksplorasi
[ ] Mulai dari segmen kecil dan risiko rendah
[ ] Gunakan metrik kerugian, bukan hanya reward
[ ] Simpan audit log state-action-reward
[ ] Terapkan human approval untuk aksi ekstrem
[ ] Hentikan eksperimen jika guardrail dilanggar

Cara membacanya: eksplorasi bukan izin untuk bertindak sembarangan. Agent hanya boleh mencoba dalam ruang aksi yang sudah disetujui, pada skala yang terkendali, dan dengan metrik keselamatan yang dipantau. Di dunia nyata, “mencari informasi” tetap harus tunduk pada etika, hukum, dan kepentingan pengguna.

4.9 Hubungan dengan deep RL dan LLM

Pada DQN, eksplorasi sering dimulai dengan ε-greedy: agent kadang memilih aksi acak meskipun Q-network punya prediksi. Pada policy gradient dan SAC, eksplorasi muncul dari policy stokastik dan entropy. Pada RLHF, eksplorasi muncul ketika model menghasilkan beberapa kandidat jawaban untuk dinilai manusia/AI. Tetapi pada LLM, eksplorasi jauh lebih sensitif karena aksi adalah bahasa yang bisa memengaruhi manusia. Karena itu eksplorasi LLM harus dibatasi oleh safety filter, instruksi sistem, dan evaluasi manusia.

Inti praktisnya:

Jika risiko rendah dan simulator murah → eksplorasi bisa lebih agresif.
Jika risiko tinggi dan pengguna nyata terlibat → eksplorasi harus konservatif.
Jika environment berubah → sisakan eksplorasi kecil agar agent tidak membeku.

Cara membacanya: tidak ada satu strategi eksplorasi yang selalu benar. Pilihan strategi bergantung pada biaya salah, biaya mencoba, stabilitas environment, dan nilai informasi masa depan.

Tes cepat subbab 4

1. Jelaskan eksplorasi dan eksploitasi dengan contoh warung makan.

2. Mengapa greedy bisa terlihat bagus di awal tetapi buruk dalam jangka panjang?

3. Hitung regret jika aksi terbaik rata-rata memberi 100, `T=5`, dan reward aktual `[70, 80, 100, 90, 100]`.

4. Apa arti ε=`0,1` pada ε-greedy?

5. Mengapa UCB lebih “cerdas” daripada eksplorasi acak?

6. Sebutkan dua guardrail untuk eksplorasi di aplikasi bisnis nyata.

Subbab 5 — Game theory dan multi-agent RL

Inti subbab: ketika ada lebih dari satu pengambil keputusan, environment ikut belajar atau melawan.

Pada bandit promo, pesaing bisa merespons. Jika warung A memberi diskon, warung B juga diskon. Reward kita tidak hanya tergantung aksi kita, tetapi juga aksi pihak lain. Ini membawa RL ke game theory.

Konsep dasar:

Konsep	Arti singkat	Contoh
Zero-sum game	keuntungan satu pihak = kerugian pihak lain	catur, sebagian game kompetitif
General-sum game	semua pihak bisa untung/rugi bersama	pricing pasar, lalu lintas
Nash equilibrium	tidak ada pemain yang ingin ubah strategi sendiri	harga stabil pesaing
Markov game	MDP dengan banyak agent	multi-agent RL [R20]
Self-play	agent belajar melawan versi dirinya	TD-Gammon, AlphaZero [R21][R23]

Pada game, policy bukan hanya “apa yang saya lakukan di state ini?”, tetapi “apa yang saya lakukan jika lawan punya strategi tertentu?” Inilah sebabnya self-play kuat: lawan ikut membaik sehingga kurikulum otomatis terbentuk.

Contoh sederhana adalah dua toko di jalan yang sama. Jika keduanya memberi diskon besar, pelanggan senang tetapi margin dua toko turun. Jika satu toko memberi diskon dan toko lain tidak, toko diskon mungkin menang jangka pendek. Jika keduanya menahan diskon dan memperbaiki layanan, keduanya bisa sehat. Ini bukan lagi bandit biasa karena reward toko A tergantung aksi toko B. Dalam multi-agent RL, environment tidak statis: lawan, partner, atau pasar juga berubah.

Matriks payoff mini:

	Toko B stabil	Toko B diskon
Toko A stabil	(8,8)	(3,10)
Toko A diskon	(10,3)	(5,5)

Cara membacanya: angka pertama adalah reward Toko A, angka kedua reward Toko B. Jika A stabil dan B diskon, payoff `(3,10)` berarti A kalah pelanggan, B menang jangka pendek. Jika keduanya diskon, payoff `(5,5)` lebih rendah dari `(8,8)` karena perang harga merusak margin. Game theory membantu kita melihat bahwa aksi “terbaik” tidak bisa ditentukan tanpa asumsi strategi pihak lain.

Contoh batu-gunting-kertas: policy deterministik mudah dieksploitasi. Jika selalu batu, lawan selalu kertas. Policy optimal harus stokastik: batu, gunting, kertas masing-masing sekitar 1/3. Ini contoh mengapa actor-critic yang bisa belajar policy stokastik penting.

Tes cepat subbab 5

1. Mengapa multi-agent RL lebih sulit daripada single-agent RL?

2. Beri contoh general-sum game di kehidupan sehari-hari.

3. Mengapa self-play dapat menjadi kurikulum otomatis?

Subbab 6 — MDP: bahasa formal untuk keputusan berurutan

Inti subbab: Markov Decision Process (MDP) adalah kerangka matematika utama RL [R1][R3].

MDP punya lima komponen:

(S, A, P, R, γ)

Cara membacanya: sebuah MDP adalah paket berisi lima hal: himpunan state S, himpunan aksi A, aturan perpindahan P, reward R, dan discount factor γ. Jika satu dari lima ini tidak jelas, masalah RL belum benar-benar terdefinisi.

	Arti	
	himpunan state	
	himpunan action	
	s,a)`	peluang pindah ke state berikutnya
	reward transisi	
	discount factor	

Markov property: masa depan cukup diprediksi dari state sekarang dan aksi sekarang, bukan seluruh sejarah.

P(S_{t+1} | S_t, A_t, sejarah lama) = P(S_{t+1} | S_t, A_t)

Cara membacanya: peluang state berikutnya cukup ditentukan oleh state sekarang dan aksi sekarang. Riwayat lama tidak perlu dibawa lagi jika semua informasi penting sudah diringkas di `S_t`. Ini bukan berarti sejarah tidak penting; sejarah boleh penting, tetapi harus dimasukkan ke state. Misalnya “pelanggan sudah menerima voucher tiga kali bulan ini” harus menjadi bagian state jika memengaruhi respons pelanggan.

Contoh gridworld evakuasi:

```
state  = posisi agent di grid
aksi   = atas/bawah/kiri/kanan
reward = -1 per langkah, +10 saat selamat, -10 saat bahaya
```

Cara membacanya: satu baris mendefinisikan apa yang agent lihat (`state`), satu baris mendefinisikan pilihan yang boleh dilakukan (`action`), dan satu baris mendefinisikan konsekuensi numerik (`reward`). Penalti `-1` per langkah membuat agent tidak berputar-putar terlalu lama.

Jika posisi sekarang cukup untuk memprediksi transisi, state Markov. Jika ada api yang menyebar tetapi tidak dimasukkan ke state, state tidak Markov.

Episodic vs continuing:

• Episodic: game selesai, episode berakhir.

• Continuing: sistem terus berjalan, seperti alokasi iklan harian.

Tes cepat subbab 6

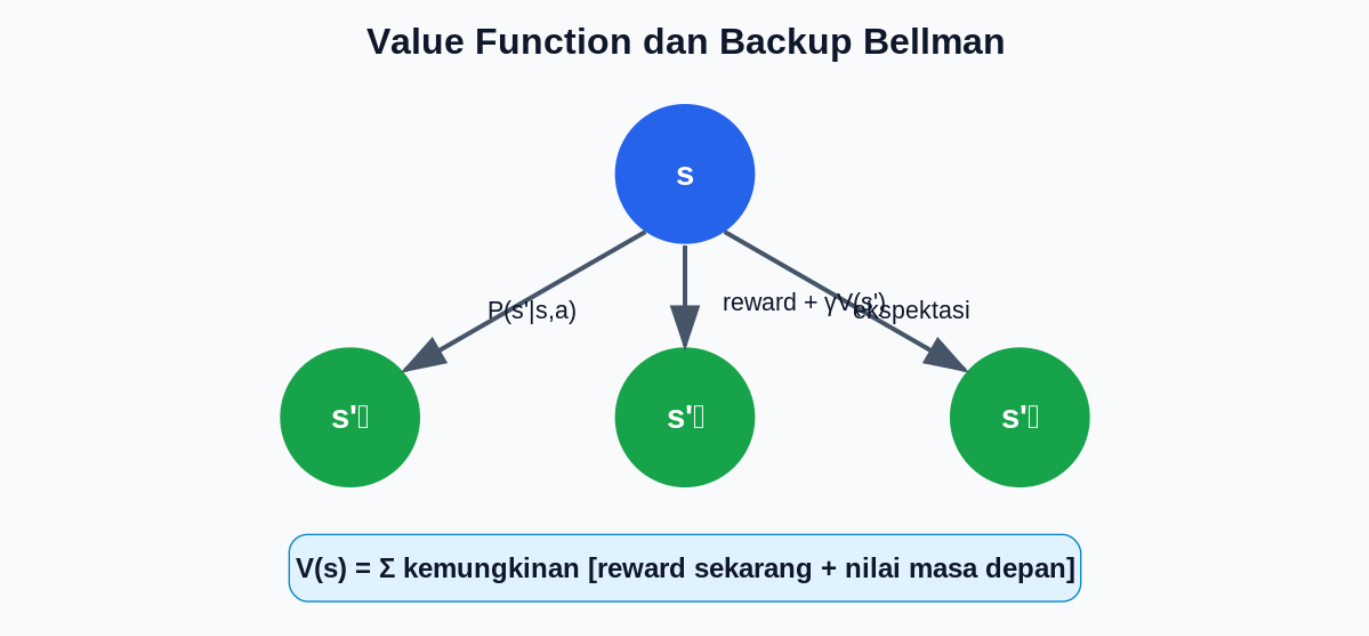
1. Tulis MDP untuk game ular tangga sederhana.

2. Apa contoh state yang tidak Markov pada aplikasi sales?

3. Mengapa γ diperlukan pada continuing task?

Subbab 7 — Value function dan Bellman equation

Inti subbab: value function menjawab: “seberapa bagus state/action ini untuk reward masa depan?”



Value dan backup Bellman

State-value:

$$V^{\pi}(s) = E[G_t \mid S_t=s, \text{mengikuti policy } \pi]$$

Cara membacanya: $V^{\pi}(s)$ adalah nilai state s jika agent mengikuti policy π. Simbol E berarti rata-rata harapan, karena masa depan bisa acak. Bagian setelah garis vertikal $\mid$ dibaca “dengan syarat”: kita mulai dari state s, lalu bertindak menurut policy π.

Action-value:

$$Q^{\pi}(s,a) = E[G_t \mid S_t=s, A_t=a, \text{lalu mengikuti } \pi]$$

Cara membacanya: $Q^{\pi}(s,a)$ adalah nilai mengambil aksi a di state s, lalu setelah itu kembali mengikuti policy π. Perbedaan praktisnya: V menilai tempat, Q menilai pilihan di tempat itu. Untuk memilih aksi, Q sering lebih langsung karena kita bisa membandingkan $Q(s,\text{kiri})$, $Q(s,\text{kanan})$, dan seterusnya.

Bellman expectation equation:

$$V^{\pi}(s) = \Sigma_a \pi(a|s) \Sigma_{s'} P(s'|s,a) [R(s,a,s') + \gamma V^{\pi}(s')]$$

Cara membacanya secara kalimat: nilai state s sama dengan rata-rata semua kemungkinan aksi dan semua kemungkinan state berikutnya. Aksi dirata-ratakan menurut peluang policy $\pi(a|s)$. State berikutnya dirata-ratakan menurut peluang transisi $P(s'|s,a)$. Untuk setiap kemungkinan, kita hitung reward langsung $R(s,a,s')$ plus nilai masa depan $\gamma V^{\pi}(s')$.

Cara membacanya:

1. Dari state s, policy memilih action a.
2. Environment pindah ke s'.
3. Kita menerima reward sekarang.
4. Kita menambahkan nilai masa depan $\gamma V(s')$.
5. Semua kemungkinan dirata-ratakan.

Bellman optimality:

$$V^*(s) = \max_a \Sigma_{s'} P(s'|s,a) [R(s,a,s') + \gamma V^*(s')]$$

Cara membacanya: $V^*(s)$ adalah nilai terbaik yang mungkin dicapai dari state s. Untuk mendapatkannya, kita mencoba semua aksi a, menghitung rata-rata reward plus masa depan

untuk setiap aksi, lalu mengambil yang terbesar. Tanda bintang * dibaca “optimal”. Kata max berarti memilih aksi terbaik.

Pendalaman intuisi Bellman: Bellman equation terasa seperti definisi melingkar: nilai sekarang bergantung pada nilai masa depan. Tetapi justru inilah kekuatannya. Jika kita punya tebakan awal nilai semua state, kita bisa memperbaiki tebakan itu berkali-kali dengan backup Bellman. State dekat goal akan naik lebih dulu, lalu nilai itu “menular mundur” ke state yang mengarah ke goal. Inilah alasan gridworld terlihat seperti peta panas: semakin dekat jalur bagus, semakin tinggi value.

Contoh mini: dari state S, dua aksi:

> aksi aman: reward 2, ke state bernilai 5
> aksi cepat: reward 0, ke state bernilai 10
> γ = 0,9
>
> nilai aman = 2 + 0,9×5 = 6,5
> nilai cepat = 0 + 0,9×10 = 9

Cara membacanya: aksi aman memberi reward langsung 2 tetapi masa depannya bernilai 5. Aksi cepat tidak memberi reward langsung, tetapi membawa agent ke state bernilai 10. Karena masa depan dikalikan 0,9, aksi cepat bernilai 9 dan mengalahkan aksi aman 6,5.

Aksi cepat lebih baik jika hanya dari nilai harapan. Tetapi kalau state cepat berisiko tinggi, probabilitas transisi harus dimasukkan.

Tes cepat subbab 7

1. Apa beda V(s) dan Q(s,a)?

2. Hitung nilai aksi jika reward 4, next value 8, γ=0,5.

3. Mengapa Bellman equation cocok untuk berpikir rekursif?

Subbab 8 — Dynamic Programming: jika model environment diketahui

Inti subbab: Dynamic Programming (DP) menyelesaikan MDP dengan model lengkap, tetapi mahal untuk state besar [R1][R2].

DP membutuhkan P(s'|s,a) dan reward. Ini seperti punya peta lengkap jalan, peluang macet, dan biaya setiap rute. Dua algoritma inti:

1. Policy iteration: evaluasi policy → perbaiki policy → ulangi.

2. Value iteration: langsung update nilai dengan Bellman optimality.

Policy iteration:

> π lama → hitung V^π → π baru greedy terhadap V → ulangi

Value iteration:

> V(s) ← max_a Σ_{s'} P(s'|s,a)[R + γV(s')]

Cara membacanya: panah ← berarti “ganti nilai lama dengan nilai hasil backup”. Untuk setiap state, lihat semua aksi, hitung ekspektasi reward plus nilai masa depan, lalu simpan nilai terbesar sebagai nilai baru state itu. Ini diulang sampai perubahan nilai sangat kecil.

Cerita rinci value iteration: anggap Anda punya peta kota dan tahu probabilitas macet setiap jalan. Awalnya semua lokasi diberi nilai nol. Lokasi tujuan diberi nilai tinggi. Setelah satu iterasi, lokasi yang satu langkah dari tujuan mulai terlihat bernilai. Setelah dua iterasi, lokasi dua langkah dari tujuan ikut naik. Dengan banyak iterasi, nilai menyebar seperti kabar baik dari tujuan ke seluruh peta. Agent akhirnya hanya perlu mengikuti panah menuju tetangga bernilai tinggi.

DP penting karena memberi standar emas. Banyak algoritma RL modern adalah “DP tanpa model lengkap”: memakai sampel pengalaman untuk mendekati backup Bellman.

Keterbatasan DP:

• Butuh model transisi lengkap.

• Tidak cocok untuk state sangat besar.

• Curse of dimensionality: kombinasi fitur membuat jumlah state meledak.

Tes cepat subbab 8

1. Kapan DP cocok digunakan?

2. Apa beda policy iteration dan value iteration?

3. Mengapa DP sulit untuk game Pong mentah dari piksel?

Subbab 9 — Monte Carlo: belajar dari episode lengkap

Inti subbab: Monte Carlo (MC) belajar dari return aktual setelah episode selesai.

MC tidak perlu model. Ia hanya butuh episode:

S0, A0, R1, S1, A1, R2, ..., terminal

Update value:

$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

Cara membacanya: nilai state yang dikunjungi diperbarui menuju return aktual G_t. Bagian $(G_t - V(S_t))$ adalah kesalahan prediksi: seberapa jauh kenyataan episode dari dugaan lama. Jika $\alpha=1$, nilai langsung disamakan dengan return episode itu. Jika α kecil, nilai bergerak perlahan.

Contoh rinci: agent memperkirakan state awal game bernilai $V=4$. Setelah episode selesai, return aktual ternyata $G=10$. Dengan $\alpha=0,2$, update menjadi $4 + 0,2\times(10-4)=5,2$. Agent belajar bahwa state awal lebih baik dari dugaan, tetapi tidak terlalu reaktif terhadap satu episode.

Kelebihan:

• sederhana;

• tidak bootstrap;

• cocok jika episode lengkap mudah disimulasikan.

Kelemahan:

• harus menunggu episode selesai;

• variance tinggi;

• tidak cocok jika episode sangat panjang atau tidak berakhir.

Contoh game Pong: reward akhir menang/kalah baru muncul setelah rally. MC menunggu rally/game selesai, lalu mengubah nilai state-action yang muncul.

Tes cepat subbab 9

1. Mengapa MC tidak perlu model environment?

2. Mengapa MC bisa lambat pada episode panjang?

3. Apa arti “tidak bootstrap”?

Subbab 10 — Temporal Difference: belajar sebelum episode selesai

Inti subbab: TD menggabungkan MC dan DP: belajar dari pengalaman, tetapi bootstrap dari estimasi berikutnya [R7].

TD(0) update:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Cara membacanya: nilai state sekarang digeser menuju target TD. Target TD adalah reward yang baru diterima R_{t+1} ditambah nilai perkiraan state berikutnya $\gamma V(S_{t+1})$. Jadi TD tidak menunggu episode selesai; ia memakai “perkiraan masa depan” sebagai pengganti return lengkap.

TD error:

$$\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Cara membacanya: δ atau delta adalah “kejutan prediksi”. Jika delta positif, kejadian baru lebih baik daripada dugaan lama. Jika delta negatif, kejadian baru lebih buruk. Banyak algoritma RL modern dapat dibaca sebagai cara berbeda memakai delta untuk memperbarui value, policy, atau keduanya.

Contoh hitung: $V(S_t)=5$, reward baru 2, $V(S_{t+1})=8$, $\gamma=0,9$, $\alpha=0,1$.

$$\begin{aligned}\text{target} &= 2 + 0,9\times 8 = 9,2\\ \delta &= 9,2 - 5 = 4,2\\ V_{\text{baru}} &= 5 + 0,1\times 4,2 = 5,42\end{aligned}$$

Cara membacanya: state sekarang ternyata lebih menjanjikan karena reward langsung plus state berikutnya bernilai 9,2. Nilai lama 5 dinaikkan sedikit menjadi 5,42.

Intuisi: jika state berikutnya ternyata lebih menjanjikan daripada dugaan, naikkan nilai state sekarang.

Contoh perjalanan pulang: Anda memprediksi pulang 40 menit. Setelah 5 menit, ternyata jalan tol lancar dan prediksi total berubah jadi 30 menit. TD bisa memperbaiki prediksi awal sebelum sampai rumah.

Perbandingan:

Metode	Butuh model?	Tunggu episode selesai?	Bootstrap?
DP	ya	tidak	ya
MC	tidak	ya	tidak
TD	tidak	tidak	ya

Tes cepat subbab 10

1. Tulis TD error untuk satu transisi.

2. Mengapa TD bisa belajar online?

3. Kapan TD lebih praktis daripada MC?

Subbab 11 — SARSA: on-policy TD control

Inti subbab: SARSA belajar nilai dari aksi yang benar-benar diambil oleh policy saat ini [R9].

Nama SARSA berasal dari urutan:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$$

Update:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

Cara membacanya: nilai pasangan state-action sekarang diperbarui menuju reward baru plus nilai pasangan state-action berikutnya yang benar-benar dipilih. SARSA tidak bertanya “aksi terbaik apa di state berikutnya?”, tetapi “aksi apa yang policy aktual benar-benar ambil di state berikutnya?”.

Contoh hitung: `Q(s,a)=3`, reward `-1`, aksi berikutnya punya `Q(s',a')=6`, $\gamma=0,9$, $\alpha=0,5$.

```
target = -1 + 0,9×6 = 4,4
error  = 4,4 - 3 = 1,4
Qbaru  = 3 + 0,5×1,4 = 3,7
```

Cara membacanya: meskipun mendapat penalti langkah `-1`, state berikutnya cukup bagus sehingga nilai aksi sekarang tetap naik dari 3 ke 3,7.

SARSA disebut on-policy karena target update memakai aksi berikutnya yang dipilih oleh policy aktual, misalnya ε-greedy. Jika policy masih suka eksplorasi, SARSA belajar nilai policy yang memang hati-hati terhadap eksplorasi itu.

Contoh cliff walking: Q-learning mungkin belajar jalur optimal dekat jurang, tetapi saat ε-greedy kadang salah langkah dan jatuh. SARSA cenderung memilih jalur lebih aman karena memperhitungkan perilaku eksplorasi.

Tes cepat subbab 11

1. Mengapa SARSA disebut on-policy?

2. Tulis lima komponen SARSA.

3. Dalam lingkungan berbahaya, mengapa SARSA bisa lebih aman dari Q-learning?

Subbab 12 — Q-learning: off-policy TD control

Inti subbab: Q-learning belajar nilai optimal walau perilaku eksplorasinya berbeda [R8].

Update Q-learning:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

Cara membacanya: nilai `Q` untuk aksi yang baru dilakukan diperbarui menuju reward baru plus nilai terbaik yang mungkin di state berikutnya. Bagian `max_a` berarti “lihat semua aksi di state berikutnya, ambil nilai terbesar”.

Contoh hitung: `Q(s,a)=3`, reward `-1`, nilai aksi di state berikutnya `[2, 6, 4]`, $\gamma=0,9$, $\alpha=0,5$.

```
target = -1 + 0,9×max(2,6,4) = -1 + 5,4 = 4,4
Qbaru  = 3 + 0,5×(4,4 - 3) = 3,7
```

Cara membacanya: target Q-learning memakai aksi terbaik di state berikutnya, bukan aksi yang benar-benar diambil saat eksplorasi. Karena itu ia belajar seolah-olah masa depan akan greedy, walau perilaku saat training masih ε-greedy.

Bedanya dengan SARSA ada pada target:

```
SARSA:      gunakan Q(S_{t+1}, A_{t+1})
Q-learning: gunakan max_a Q(S_{t+1}, a)
```

Cara membacanya: SARSA memakai aksi berikutnya yang benar-benar dipilih policy. Q-learning memakai aksi terbaik menurut tabel Q, meskipun aksi terbaik itu belum tentu diambil saat eksplorasi.

Q-learning off-policy: behavior policy boleh ε-greedy untuk eksplorasi, tetapi target policy adalah greedy.

Kelebihan:

• sederhana;

• kuat untuk tabular task;

• dasar DQN/deep Q-network.

Kelemahan:

• overestimation bias karena operasi max;

• dapat tidak stabil dengan function approximation;

• perlu eksplorasi cukup;

• tidak cocok langsung untuk aksi kontinu berdimensi besar.

Tes cepat subbab 12

1. Apa arti off-policy?

2. Bandingkan target SARSA dan Q-learning.

3. Mengapa max dapat menyebabkan overestimation?

Subbab 13 — Eligibility traces, n-step return, dan credit assignment

Inti subbab: eligibility traces membantu membagi kredit reward ke banyak langkah sebelumnya [R1].

Masalah credit assignment: jika agent menang setelah 40 aksi, aksi mana yang berjasa? TD satu langkah hanya menguatkan aksi terakhir. MC menguatkan semua, tetapi menunggu selesai. Eligibility trace menjadi jembatan.

n-step return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

Cara membacanya: return n-langkah menjumlahkan reward nyata selama n langkah, lalu setelah itu memakai estimasi value sebagai jembatan. Jika n=1, ini mirip TD satu langkah. Jika n sepanjang episode, ini mendekati Monte Carlo. Bagian terakhir $\gamma^n V(S_{t+n})$ adalah “nilai sisa perjalanan” setelah n langkah.

TD(λ) mencampur banyak n-step return dengan bobot menurun:

λ kecil → mirip TD satu langkah
λ besar → mirip Monte Carlo

Backward view:

trace state yang baru dikunjungi naik
trace lama turun dengan γλ
TD error dibagikan ke state/action yang masih punya trace

Cara membacanya: state/action yang baru terjadi mendapat “jejak kredit” tinggi. Setiap langkah, jejak lama memudar karena dikalikan $\gamma\lambda$. Saat TD error muncul, error itu tidak hanya diberikan ke langkah terakhir, tetapi juga ke langkah-langkah sebelumnya yang jejaknya masih tersisa.

Tes cepat subbab 13

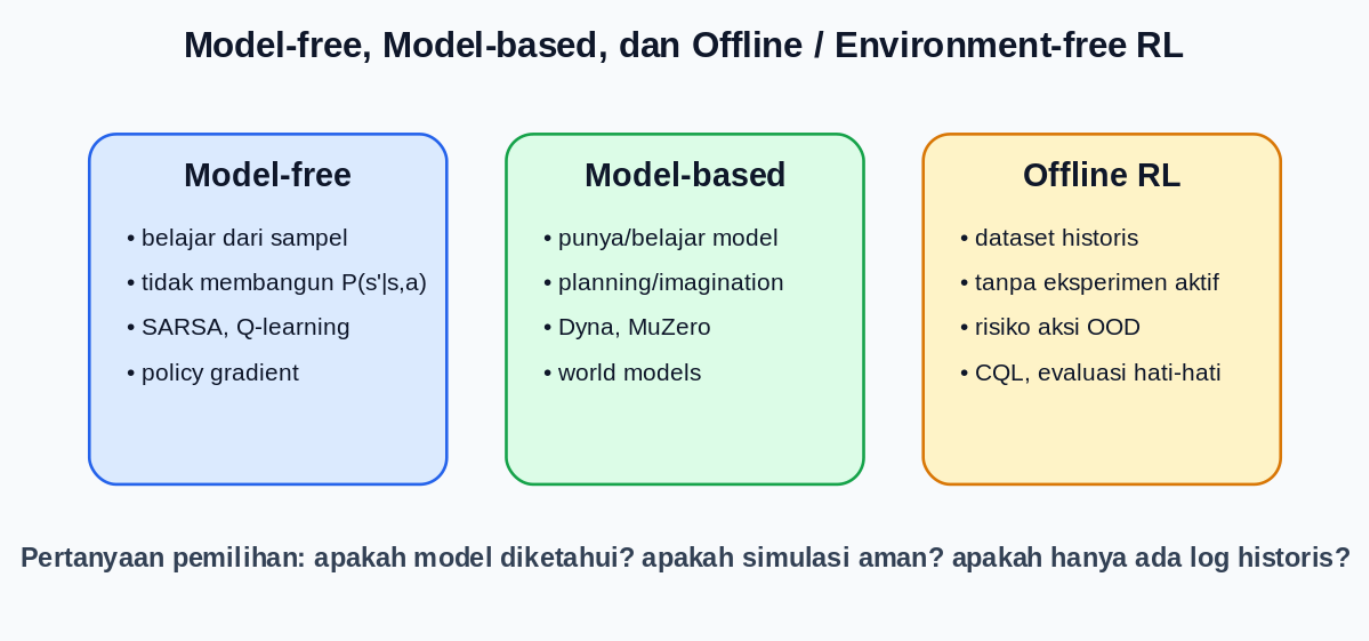
1. Apa masalah credit assignment?

2. Apa efek λ mendekati 1?

3. Mengapa eligibility traces berguna untuk reward tertunda?

Subbab 14 — Model-free, model-based, dan offline/batch RL

Inti subbab: RL berbeda berdasarkan apakah agent memakai model environment, belajar langsung, atau belajar dari dataset tanpa interaksi.



Model-free vs model-based vs offline RL

Model-free RL: tidak belajar/memakai model transisi eksplisit. Contoh: SARSA, Q-learning, policy gradient. Agent belajar dari pengalaman.

Model-based RL: agent punya atau belajar model environment, lalu planning. Contoh: Dyna-Q [R10], world models [R26], MuZero [R27].

Offline RL / batch RL: agent tidak boleh mengeksplorasi environment baru saat training. Ia belajar dari dataset historis [R24]. Ini penting untuk domain mahal/berbahaya: kesehatan, keuangan, sistem rekomendasi besar, kendaraan, atau log preferensi LLM.

Istilah yang baku di literatur adalah offline RL atau batch RL: training dilakukan dari dataset tetap tanpa interaksi baru dengan environment aktif. Secara teoretis, environment tetap ada sebagai sumber data dan target evaluasi; yang dibatasi adalah interaksi tambahan selama training.

Risiko offline RL:

• distribution shift: policy baru memilih aksi yang jarang ada di dataset;

• overestimation aksi yang tidak pernah dicoba;

• dataset bias;

• sulit mengevaluasi policy sebelum deployment.

Conservative Q-Learning (CQL) mencoba menahan nilai aksi out-of-distribution agar tidak terlalu optimistis [R25].

Tes cepat subbab 14

1. Apa beda model-free dan model-based?

2. Mengapa offline RL cocok untuk data historis sales?

3. Apa risiko policy memilih aksi yang tidak ada di dataset?

Subbab 15 — Function approximation dan DQN

Inti subbab: ketika state terlalu banyak untuk tabel, value function diperkirakan dengan model seperti neural network.

Tabular Q-learning menyimpan tabel:

```
Q[state, action]
```

Jika state adalah gambar Pong 84×84 piksel, tabel mustahil. Deep Q-Network (DQN) memakai neural network untuk mengaproksimasi Q [R12]:

```
Q(s,a; θ) ≈ nilai action a pada state s
```

Cara membacanya: neural network dengan parameter `θ` menerima state `s` dan menghasilkan perkiraan nilai untuk aksi `a`. Tanda ≈ berarti "mendekati", bukan sama persis. Pada DQN Atari, input bisa berupa frame game, dan outputnya satu nilai Q untuk setiap tombol aksi.

Istilah yang umum dipakai di literatur adalah DQN ketika Q-learning digabung dengan neural network, dan deep Q-network untuk menyebut jaringan neural yang mengaproksimasi nilai Q. Karena itu, bab ini memakai istilah DQN/deep Q-network secara konsisten.

DQN memakai beberapa trik penting:

1. Experience replay: simpan transisi `(s,a,r,s')`, sample acak untuk training.

2. Target network: network target lebih lambat berubah agar target stabil.

3. Reward clipping/preprocessing: membuat training stabil pada game Atari.

Target DQN:

```
y = r + γ max_{a'} Q_target(s',a'; θ^-)
loss = (y - Q(s,a; θ))²
```

Cara membacanya: `y` adalah target belajar. Ia terdiri dari reward `r` plus nilai terbaik state berikutnya menurut target network. Simbol `θ^-` berarti parameter target network yang lebih lambat diperbarui. `loss` menghukum selisih antara target `y` dan prediksi network saat ini `Q(s,a;θ)`. Kuadrat membuat error besar dihukum lebih keras.

Kenapa replay buffer penting: tanpa replay, neural network dilatih pada pengalaman yang sangat berurutan: kiri, kiri, kiri, lalu tiba-tiba kalah. Data seperti ini berkorelasi kuat dan membuat training tidak stabil. Replay buffer mengacak pengalaman sehingga batch training lebih mirip dataset supervised learning.

Kenapa target network penting: jika network yang sama dipakai untuk prediksi dan target, target bergerak setiap kali network berubah. Ini seperti belajar menembak papan sasaran yang ikut lari. Target network membuat papan sasaran bergerak lebih lambat.

Double DQN mengurangi overestimation dengan memisahkan pemilihan aksi dan evaluasi aksi [R13]. Dueling DQN memisahkan nilai state dan advantage action.

Tes cepat subbab 15

1. Mengapa tabel Q tidak cocok untuk state piksel?

2. Apa fungsi experience replay?

3. Mengapa target network membantu stabilitas?

Subbab 16 — Policy gradient dan REINFORCE

Inti subbab: policy gradient langsung mengoptimalkan policy, bukan memilih action dari Q-table.

Policy parameterized:

```
π_θ(a|s)
```

Cara membacanya: $\pi_\theta(a|s)$ adalah peluang policy dengan parameter θ memilih aksi a ketika berada di state s. Jika state adalah “bola Pong di atas paddle”, policy mungkin memberi peluang tinggi pada aksi “naik”.

Tujuan:

$$\text{maksimalkan } J(\theta) = \text{expected return}$$

Cara membacanya: kita ingin mengubah parameter θ agar return rata-rata $J(\theta)$ semakin besar. J bukan reward satu episode, melainkan ekspektasi reward jika policy dijalankan berkali-kali.

REINFORCE update [R11]:

$$\theta \leftarrow \theta + \alpha\, G_t\, \nabla_\theta \log \pi_\theta(A_t|S_t)$$

Cara membacanya: parameter policy digeser ke arah yang membuat aksi yang tadi dipilih menjadi lebih mungkin jika return G_t tinggi. Simbol ∇_θ dibaca “gradien terhadap parameter θ”. Jika return buruk, update efektifnya mendorong probabilitas aksi itu turun. α tetap ukuran langkah belajar.

Cara membacanya:

• Jika return tinggi, tingkatkan probabilitas aksi yang diambil.

• Jika return rendah, kurangi probabilitas aksi itu.

• $\log \pi$ membuat update matematis stabil.

Policy gradient cocok untuk:

• aksi kontinu;

• policy stokastik;

• masalah di mana argmax atas action sulit.

Kelemahan:

• variance tinggi;

• butuh banyak sample;

• sensitif pada reward scaling.

Baseline mengurangi variance:

$$\theta \leftarrow \theta + \alpha\, (G_t - b(s))\, \nabla \log \pi_\theta(A_t|S_t)$$

Cara membacanya: update tidak lagi memakai return mentah G_t, tetapi return dikurangi baseline $b(s)$. Jika return 10 tetapi baseline state itu memang 9, aksi tadi hanya “lebih baik 1 dari ekspektasi”, bukan luar biasa. Jika return 5 tetapi baseline 9, aksi tadi lebih buruk dari ekspektasi. Ini membuat sinyal belajar lebih tenang.

Baseline yang umum adalah value function $V(s)$. Ini membawa kita ke actor-critic.

Tes cepat subbab 16

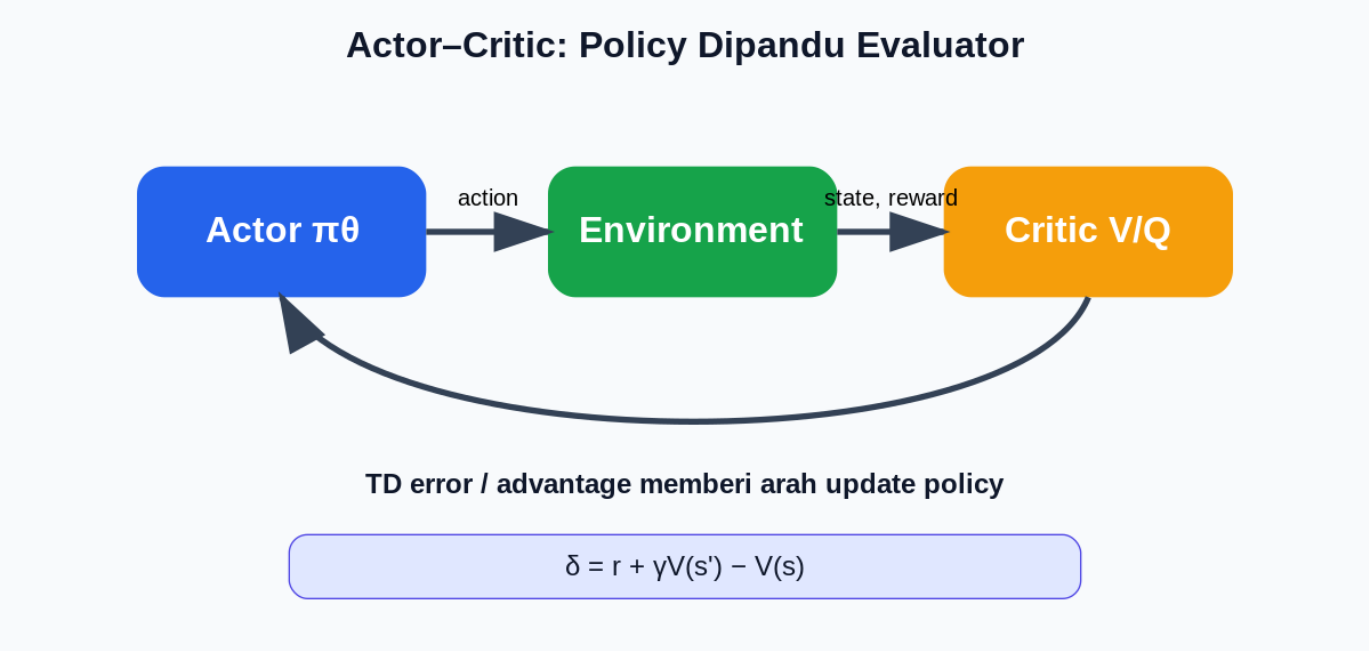
1. Apa beda policy gradient dan Q-learning?

2. Mengapa baseline tidak mengubah arah tujuan tetapi mengurangi variance?

3. Kapan policy stochastic dibutuhkan?

Subbab 17 — Actor-critic, A2C/A3C, PPO, TRPO

Inti subbab: actor-critic memisahkan policy (actor) dan evaluator (critic).



Actor-critic dan policy optimization

Actor memilih aksi:

$$\pi_\theta(a|s)$$

Critic menilai:

$$V_w(s) \text{ atau } Q_w(s,a)$$

TD error sebagai sinyal kritik:

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

Cara membacanya: critic menghitung apakah hasil transisi lebih baik atau lebih buruk dari dugaan. Reward r plus nilai masa depan $\gamma V(s')$ dibandingkan dengan nilai state lama $V(s)$. Delta positif berarti actor patut diperkuat untuk aksi itu.

Actor update kira-kira:

$$\theta \leftarrow \theta + \alpha\, \delta\, \nabla \log \pi_\theta(a|s)$$

Cara membacanya: actor menaikkan probabilitas aksi jika critic memberi delta positif, dan menurunkannya jika delta negatif. Dengan kata lain, critic memberi komentar “aksi tadi lebih baik/buruk dari ekspektasi”, actor menyesuaikan kebiasaan.

A2C/A3C memakai advantage:

$$A(s,a) = Q(s,a) - V(s)$$

Cara membacanya: advantage adalah kelebihan aksi a dibanding rata-rata kualitas state s. Jika $A(s,a)>0$, aksi itu lebih baik dari ekspektasi untuk state tersebut. Jika negatif, aksi itu lebih buruk. Ini lebih informatif daripada reward mentah karena setiap state punya tingkat kesulitan berbeda.

TRPO dan PPO muncul karena policy gradient bisa merusak policy jika update terlalu besar [R15][R16]. PPO memakai rasio policy baru/lama dan clipping:

$$\text{ratio} = \pi_\theta(a|s) / \pi_{\text{old}}(a|s)$$
$$\text{objective} \approx \min(\text{ratio} \times A,\ \text{clip}(\text{ratio}, 1-\varepsilon, 1+\varepsilon) \times A)$$

Cara membacanya: `ratio` membandingkan seberapa besar policy baru menaikkan atau menurunkan peluang aksi dibanding policy lama. Jika ratio 1, policy belum berubah. Jika 1,5, aksi itu 50% lebih mungkin. PPO memotong ratio agar perubahan tidak terlalu ekstrem. Fungsi `min` memilih objective yang lebih konservatif sehingga agent tidak mendapat insentif melompat terlalu jauh.

Intuisi: boleh memperbaiki policy, tetapi jangan melompat terlalu jauh dari policy lama.

Cerita PPO untuk LLM: dalam RLHF, policy awal biasanya hasil SFT yang sudah cukup bisa berbicara. Jika PPO terlalu agresif mengejar reward model, LLM bisa menjadi aneh: terlalu panjang, terlalu memuji, atau mengeksploitasi pola reward. Clipping PPO adalah rem tangan: model boleh bergerak ke arah jawaban yang lebih disukai, tetapi tidak boleh terlalu jauh dari kemampuan awalnya dalam satu update.

Tes cepat subbab 17

1. Apa tugas actor dan critic?

2. Mengapa PPO membatasi perubahan policy?

3. Apa arti advantage?

Subbab 18 — Continuous control: DDPG, TD3, dan SAC

Inti subbab: untuk aksi kontinu seperti steering, torque, atau harga, algoritma khusus diperlukan.

Aksi diskret: pilih kiri/kanan/atas/bawah. Aksi kontinu: pilih harga Rp18.500, sudut kemudi 3,2°, atau tekanan motor 0,37. Argmax Q atas semua aksi kontinu sulit.

DDPG memakai actor deterministik untuk memilih aksi kontinu dan critic untuk menilai [R17].

TD3 memperbaiki DDPG dengan twin critics, delayed policy update, dan target policy smoothing untuk mengurangi overestimation [R18].

SAC (Soft Actor-Critic) menambahkan entropy agar policy tetap eksploratif [R19]. Tujuannya bukan hanya reward, tetapi reward + entropy:

```
maksimalkan E[Σ γ^t (r_t + α_entropy H(π(.|s_t)))]
```

Cara membacanya: SAC memaksimalkan ekspektasi jumlah reward masa depan, tetapi reward setiap langkah ditambah bonus entropy. `H(π(.|s_t))` mengukur seberapa menyebar pilihan aksi policy pada state `s_t`. Jika policy terlalu yakin pada satu aksi, entropy rendah. `α_entropy` mengatur seberapa besar bonus keberagaman aksi dibanding reward utama.

Intuisi SAC: jangan hanya cari aksi yang tampak bagus; pertahankan variasi aksi selama bermanfaat.

Pendalaman SAC: pada aksi kontinu, eksplorasi tidak bisa hanya “pilih aksi acak 10% dari waktu” seperti ε-greedy. Ruang aksi terlalu luas. SAC membangun eksplorasi ke dalam objektif. Agent diberi penghargaan jika tetap memiliki beberapa opsi masuk akal. Dalam pricing, ini seperti sistem yang tidak langsung mengunci satu harga, tetapi tetap mencoba variasi kecil di sekitar harga terbaik selama ketidakpastian masih ada. Setelah data cukup kuat, policy tetap bisa menjadi lebih tajam, tetapi tidak rapuh sejak awal. Ini membantu eksplorasi dan stabilitas.

Contoh sales pricing: jika agent hanya mengeksploitasi harga Rp20.000 karena pernah untung, ia mungkin tidak menemukan bahwa Rp22.000 lebih baik pada akhir pekan. Entropy mendorong eksplorasi harga.

Tes cepat subbab 18

1. Mengapa aksi kontinu sulit untuk Q-learning biasa?

2. Apa fungsi twin critics di TD3?

3. Mengapa SAC menambahkan entropy?

Subbab 19 — RL untuk game: Pong, self-play, AlphaGo, AlphaZero

Inti subbab: game adalah laboratorium RL karena reward jelas, simulasi cepat, dan policy dapat diuji berulang.

Pong sederhana:

```
state: posisi bola, arah bola, posisi paddle
aksi: paddle naik/diam/turun
reward: +1 jika menang rally, -1 jika kalah, 0 selain itu
```

Tabular Q-learning bisa bekerja jika state didiskritisasi. DQN diperlukan jika input berupa piksel mentah.

TD-Gammon memakai TD learning dan self-play untuk backgammon [R21]. AlphaGo menggabungkan deep neural networks, tree search, supervised learning dari game manusia, dan RL self-play [R22]. AlphaZero memperluas self-play tanpa data manusia untuk Go, catur, dan shogi [R23].

Kunci game RL:

• reward terminal jelas;

• simulasi banyak episode;

• self-play membuat lawan semakin kuat;

• planning/search meningkatkan keputusan saat inference.

Tes cepat subbab 19

1. Mengapa Pong cocok sebagai praktikum RL?

2. Apa beda tabular Pong dan DQN Pong?

3. Mengapa self-play kuat pada game kompetitif?

Subbab 20 — RL untuk optimasi sales dan rekomendasi

Inti subbab: banyak keputusan bisnis adalah sequential decision problem, tetapi harus dimulai sederhana dan aman.

Sales optimization bisa dimodelkan bertahap:

1. Bandit: pilih promo terbaik tanpa state.

2. Contextual bandit: pilih promo berdasarkan hari, cuaca, segmen pelanggan.

3. MDP/RL: aksi hari ini mengubah stok, loyalitas, dan demand besok.

4. Offline RL: belajar dari log historis tanpa eksperimen langsung.

Contoh reward:

```
reward = laba - penalti stok habis - penalti komplain - penalti diskon berlebihan
```

Cara membacanya: reward bisnis tidak boleh hanya revenue. Kita mulai dari laba, lalu mengurangi hukuman jika stok habis, pelanggan komplain, atau diskon terlalu agresif. Rumus reward seperti ini memaksa agent memikirkan kualitas keputusan, bukan sekadar angka penjualan.

Bahaya reward salah:

• Jika reward hanya revenue, agent memberi diskon besar terus.

• Jika reward hanya margin, agent tidak mau promo dan kehilangan pelanggan.

• Jika reward tidak menghukum stockout, agent membuat permintaan tanpa stok.

Untuk bisnis nyata, mulai dari bandit/ contextual bandit dengan guardrail:

batas diskon maksimum
batas eksperimen per segmen
monitor margin dan komplain
human approval untuk promo ekstrem

Tes cepat subbab 20

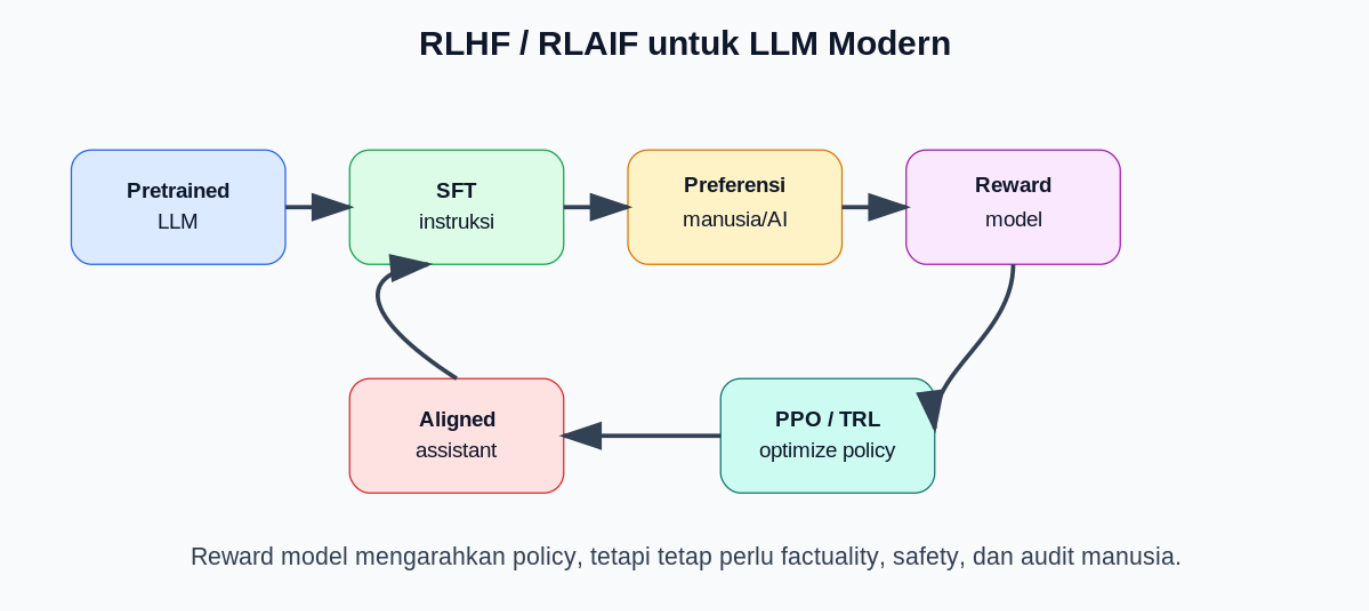
1. Mengapa sales optimization tidak langsung memakai DQN besar?

2. Buat reward yang menyeimbangkan revenue dan margin.

3. Apa guardrail untuk eksperimen harga?

Subbab 21 — RLHF, RLAIF, Hugging Face TRL, dan LLM modern

Inti subbab: RL menjadi fondasi alignment LLM modern melalui reward model, preference optimization, dan feedback loop.



RLHF dan LLM alignment

Bab 11 membahas GPT/ChatGPT. Sekarang kita lihat peran RL. InstructGPT memakai human feedback untuk membuat model lebih mengikuti instruksi [R29]. Secara garis besar:

1. Pretraining LLM
2. Supervised fine-tuning pada instruksi-jawaban
3. Kumpulkan preferensi manusia: jawaban A lebih baik dari B
4. Latih reward model
5. Optimalkan model bahasa sebagai policy terhadap reward model, sering dengan PPO

RLHF tidak membuat model “benar” secara absolut. Ia mengarahkan model agar jawaban lebih disukai berdasarkan data preferensi. Jika data preferensi bias, reward model juga bias.

RLAIF memakai feedback AI sebagai pengganti atau pelengkap feedback manusia. Constitutional AI memakai prinsip tertulis dan AI feedback untuk mengarahkan harmlessness [R30].

Preference model sering memakai ide Bradley-Terry/logistic sederhana:

P(jawaban A disukai daripada B) = sigmoid(r(A) - r(B))

Cara membacanya: peluang jawaban A menang dibanding B ditentukan oleh selisih skor reward $r(A) - r(B)$. Jika reward A jauh lebih besar, sigmoid mendekati 1. Jika reward A dan B sama, peluangnya sekitar 0,5. Reward model dilatih agar skor jawaban yang dipilih manusia/AI menjadi lebih tinggi daripada alternatif.

Dalam RLHF untuk LLM, “aksi” bukan tombol sederhana, melainkan rangkaian token. Satu jawaban bisa berisi ratusan token. Karena itu optimisasi policy harus hati-hati: model tidak hanya belajar memilih satu aksi, tetapi mengubah distribusi seluruh bahasa. Reward model juga tidak melihat kebenaran dunia secara langsung; ia melihat pola preferensi dari data. Maka RLHF harus dipasangkan dengan evaluasi factuality, red-teaming, kebijakan refusal, dan audit bias.

Hugging Face TRL adalah library/ekosistem alat untuk post-training model transformer, termasuk SFT, reward modeling, PPO, DPO, dan metode preference optimization lain [R32]. Jadi dalam bab ini, “TRL” merujuk pada library praktis tersebut, bukan nama algoritma RL baru. DPO kemudian menunjukkan alternatif non-RL eksplisit yang mengoptimalkan preferensi langsung [R31]. Meski DPO bukan RL klasik, ia berada dalam keluarga alignment dari preference feedback yang tumbuh dari masalah RLHF.

Jembatan konsep:

RL klasik	LLM alignment
state	prompt + konteks dialog
action	token/jawaban yang dihasilkan
policy	model bahasa
reward	preferensi manusia/AI, safety score
episode	satu respons atau dialog multi-turn
eksplorasi	sampling jawaban kandidat

Tes cepat subbab 21

1. Apa peran reward model dalam RLHF?

2. Mengapa RLHF tidak menjamin kebenaran faktual?

3. Apa beda RLHF dan RLAIF?

Subbab 22 — Evaluasi, safety, dan reward hacking

Inti subbab: RL sangat kuat tetapi rawan reward hacking, eksplorasi berbahaya, dan evaluasi palsu.

Reward hacking terjadi ketika agent memaksimalkan reward formal tetapi melanggar tujuan manusia. Contoh:

Domain	Reward salah	Perilaku buruk
Sales	revenue	diskon ekstrem, margin rusak
Tutor	waktu pakai aplikasi	membuat siswa kecanduan, bukan paham
Gudang	jumlah order cepat	mengabaikan keselamatan
LLM	rating “jawaban meyakinkan”	jawaban panjang tapi hallucinated

Checklist safety RL:

[ ] Reward mencerminkan tujuan jangka panjang?
[ ] Ada constraint keras untuk keselamatan/privasi?
[ ] Ada evaluasi off-policy sebelum deployment?
[ ] Ada human override?
[ ] Ada monitoring drift?
[ ] Ada limit eksplorasi?
[ ] Ada audit log state-action-reward?

Untuk konteks Indonesia:

• Jangan eksperimen RL langsung pada harga kebutuhan pokok tanpa kontrol etis.

• Jangan memakai RL untuk keputusan pinjaman tanpa fairness dan regulasi.

• Jangan memakai data pribadi tanpa izin.

• Untuk LLM layanan publik, reward harus memasukkan groundedness dan eskalasi manusia.

Tes cepat subbab 22

1. Buat contoh reward hacking pada chatbot kampus.

2. Apa bedanya reward dan constraint?

3. Mengapa human override penting?

Subbab 23 — Praktikum besar Bab 12

Inti subbab: pembaca menjalankan beberapa pendekatan RL kecil agar melihat perbedaan bandit, SARSA, Q-learning, Pong, sales, dan SAC-style entropy.

Folder `code/` menyediakan `rl_playground.py` dan notebook pendamping. Script ini tidak butuh internet dan tidak memakai GPU.

Praktikum utama:

1. Sales bandit: bandingkan ε-greedy dan UCB untuk promo.

2. Gridworld evakuasi: bandingkan SARSA dan Q-learning.

3. Mini Pong diskret: latih Q-learning pada Pong sederhana.

4. Policy gradient bandit: lihat update preference.

5. Soft/SAC-style pricing bandit: lihat entropy menjaga eksplorasi.

6. RLHF toy: latih reward model sederhana dari preferensi jawaban.

Cara menjalankan:

```
cd zero-to-hero-menaklukkan-ai/chapters/12-reinforcement-learning/code
python3 rl_playground.py --self-test
python3 rl_playground.py
```

Output disimpan ke:

```
code/outputs/bab12_rl_results.json
```

Tes cepat subbab 23

1. Mengapa praktikum memakai environment kecil?

2. Apa metrik yang dibandingkan pada SARSA vs Q-learning?

3. Mengapa toy RLHF tidak boleh dianggap sama dengan training LLM nyata?

Subbab 24 — Peta algoritma: kapan memakai apa?

Inti subbab: pilih algoritma berdasarkan bentuk state/action, risiko, data, dan kemampuan simulasi.

Peta keputusan algoritma RL

Situasi	Kandidat awal	Catatan
A/B testing promo sederhana	ε-greedy, UCB, Thompson	mulai dari bandit
Konteks pengguna penting, aksi tidak ubah state	contextual bandit	lebih aman dari RL penuh
Grid kecil, model diketahui	value iteration/policy iteration	DP sebagai baseline
Grid/game kecil, model tidak diketahui	SARSA, Q-learning	tabular
State besar diskret/piksel	DQN/Double DQN	butuh replay dan target network
Action kontinu	DDPG, TD3, SAC	SAC sering baseline kuat
Multi-agent kompetitif	self-play, Markov games	evaluasi sulit
Data hanya historis	offline RL, CQL	hati-hati aksi out-of-distribution (OOD)
LLM alignment	SFT + reward model + PPO/RLAIF/DPO/TRL	evaluasi safety wajib

Aturan praktis:

> Mulai dari baseline paling sederhana yang bisa diuji.
> Jika bandit cukup, jangan pakai deep RL.
> Jika simulator tidak valid, policy hasil RL juga tidak valid.
> Jika reward tidak benar, algoritma bagus tetap menghasilkan perilaku buruk.

Tes cepat subbab 24

1. Kapan contextual bandit lebih aman daripada RL penuh?
2. Kapan SAC lebih relevan daripada DQN?
3. Mengapa baseline sederhana wajib dibuat dulu?

Ringkasan bab

1. RL belajar dari interaksi: state → action → reward → next state.

2. Bandit adalah RL paling sederhana untuk melihat dilema eksplorasi-eksploitasi.

3. Game theory muncul saat ada banyak agent yang saling memengaruhi.

4. MDP memberi bahasa formal untuk state, action, transition, reward, dan discount.

5. Bellman equation adalah jantung value-based RL.

6. DP membutuhkan model lengkap; MC belajar dari episode; TD belajar online dengan bootstrap.

7. SARSA on-policy; Q-learning off-policy.

8. Eligibility traces membantu credit assignment reward tertunda.

9. DQN/deep Q-network membawa Q-learning ke state besar dengan neural network.

10. Policy gradient dan actor-critic langsung mengoptimalkan policy.

11. SAC menambahkan entropy agar eksplorasi dan stabilitas lebih baik pada continuous control.

12. Offline/batch RL belajar dari dataset historis tanpa interaksi aktif tambahan dengan environment saat training.

13. RLHF/RLAIF dan library seperti Hugging Face TRL menjembatani RL/preference optimization dengan alignment LLM modern.

14. Safety, reward design, dan evaluasi lebih penting daripada sekadar memilih algoritma kompleks.

Tes akhir bab

1. Jelaskan RL kepada pemilik UMKM dalam 5 kalimat.

2. Buat MDP untuk sistem rekomendasi promo toko online.

3. Hitung return discounted untuk reward `[3, -1, 10]` dengan $\gamma=0,8$.

4. Bandingkan DP, MC, dan TD dalam tabel.

5. Tulis update SARSA dan Q-learning, lalu jelaskan bedanya.

6. Mengapa DQN butuh replay buffer dan target network?

7. Jelaskan SAC dengan analogi eksplorasi harga toko.

8. Apa yang dimaksud offline/batch RL, dan mengapa ia berisiko?

9. Jelaskan pipeline RLHF dari pretraining sampai PPO.

10. Buat satu contoh reward hacking dan cara mencegahnya.

Kunci refleksi untuk pembaca

Jika setelah bab ini Anda hanya mengingat satu hal, ingatlah ini:

> Reinforcement Learning bukan ilmu “membuat agent menang game”, melainkan ilmu mendesain keputusan berurutan di bawah ketidakpastian, konsekuensi tertunda, dan reward yang harus dirancang sangat hati-hati.

Untuk melanjutkan ke Bab 13, pembaca diharapkan sudah mampu:

- memetakan masalah nyata menjadi state/action/reward;
- memilih antara bandit, contextual bandit, tabular RL, deep RL, atau offline RL;
- membaca rumus Bellman, TD, SARSA, Q-learning, dan policy gradient;

- menjalankan praktikum sederhana dari terminal/notebook;
- menjelaskan mengapa RLHF/RLAIF menjadi bagian penting dari LLM modern;
- mengkritik reward design sebelum mempercayai hasil agent.

Catatan kritik dan revisi bab

Bab ini sengaja menolak dua ekstrem: terlalu matematis sampai pemula tersesat, atau terlalu populer sampai RL tampak seperti sihir. Kritik internal yang digunakan saat revisi:

1. Apakah setiap rumus punya cerita? Jika tidak, tambahkan analogi.

2. Apakah setiap algoritma punya batasan? Jika tidak, tambahkan risiko.

3. Apakah praktik terlalu mainan? Jika ya, hubungkan ke sales, Pong, dan LLM.

4. Apakah klaim modern punya referensi? Jika tidak, tambahkan sumber primer.

5. Apakah pembaca Indonesia bisa membayangkan manfaatnya? Jika tidak, gunakan kasus UMKM, gudang, tutor, layanan publik, dan marketplace.

Hasil revisi akhir: bab ini mempertahankan fondasi Sutton & Barto, menambahkan jembatan ke deep RL modern, dan menghubungkan RL ke LLM alignment tanpa mengklaim bahwa toy code setara dengan sistem produksi.