Actuarial Computation and Simulation

Week 04 : Policy Gradient & Actor-Critic

Aprida Siska Lestia

September 7, 2025

Benang Merah Materi

- **Week 1: Multi-Armed Bandit**
  Fokus: memilih aksi terbaik (ϵ-*greedy*, *UCB*) tanpa mempertimbangkan state.
- **Week 2: Dynamic Programming (DP)**
  Fokus: Markov Decision Process (MDP) dengan *Value Iteration* & *Policy Iteration*.
  Asumsi: model transisi dan reward diketahui lengkap.
- **Week 3: Monte Carlo & TD Learning**
  Fokus: belajar *tanpa model* dengan pengalaman langsung.
  Perbandingan: **SARSA (on-policy)** vs **Q-learning (off-policy)**.
- **Week 4: Policy Gradient & Actor–Critic**
  Fokus: langsung mengoptimalkan *policy* parametrik (probabilistik) tanpa perlu *value function* eksplisit.
  - **REINFORCE**: update berdasarkan return penuh (mirip Monte Carlo).
  - **Actor–Critic**: gabungan policy gradient (actor) dengan value function (critic) berbasis TD error $\rightarrow$ lebih stabil.

Benang Merah Materi

Alur Konsep (lanjutan):

Eksplorasi Aksi (Bandit) $\rightarrow$ MDP dengan Model Lengkap (DP) $\rightarrow$
Belajar dari Pengalaman Nyata (MC/TD) $\rightarrow$
Optimisasi Policy Secara Langsung (Policy Gradient & Actor–Critic)

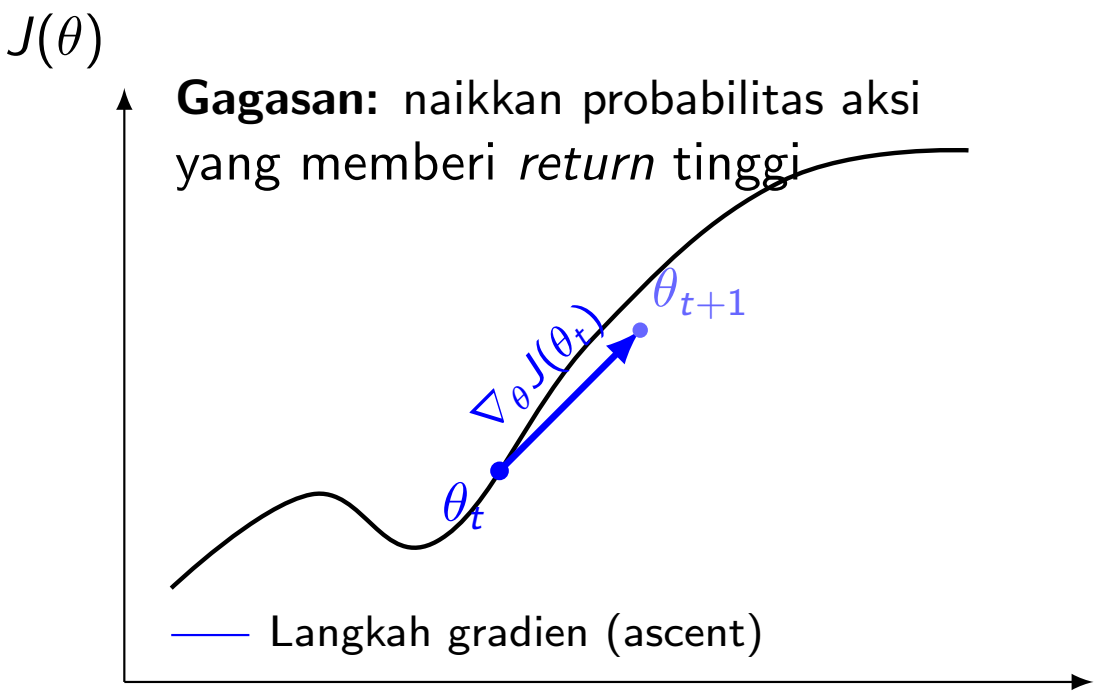
Overview

- Motivasi Policy Gradient
- Algoritma REINFORCE (episodic return)
- Algoritma Actor–Critic (TD error)
- Perbandingan hasil eksperimen
- Lab: implementasi pada CartPole
- Assignment dan diskusi reflektif

Motivasi Policy Gradient

- Konteks :
  - bandit $\rightarrow$ tidak ada state
  - DP $\rightarrow$ butuh model lengkap
  - MC/TD $\rightarrow$ bisa tanpa model, tapi tetap berbasis value
  - PG $\rightarrow$ langsung optimasi policy, cocok untuk kontrol kontinu.
- Policy Gradient langsung memodelkan kebijakan $\pi_\theta(a|s)$.
- Intuisi :
  - Bayangkan menggeser distribusi aksi ke arah yang memberi reward lebih tinggi.
  - Update parameter policy mengikuti arah gradien reward.

Intuisi Policy Gradient



Policy-based (Policy Gradient)

- Optimasi langsung parameter kebijakan $\pi_\theta(a \mid s)$.
- Aturan arah: $\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a \mid s)\, Q^\pi(s,a)]$.
- Intuisi: *perbesar* log-prob aksi yang bernilai (Q) tinggi.

Value-based (pembanding)

- Estimasi dulu $Q(s,a)$ atau $V(s)$.
- $\pi(s) = \arg\max_a Q(s,a)$ (kebijakan *turunan* dari nilai).
- Kurang alami untuk aksi kontinu (butuh argmax/greedy di ruang kontinu).

Policy Gradient Theorem

Pesan kunci

Policy Gradient memindahkan parameter θ ke arah yang meningkatkan $J(\theta)$ secara *langsung*; ActorCritic menurunkan varians dengan *kritikus* (mis. V atau A).

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right]$$

- $J(\theta)$: expected return.
- $\pi_\theta(a|s)$: policy parameterized by θ.
- Intuisi: update mengikuti arah yang meningkatkan reward jangka panjang.

Algoritma REINFORCE

Update Rule

$$\theta \leftarrow \theta + \alpha \, G_t \, \nabla_\theta \log \pi_\theta(a_t|s_t)$$

dengan $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$.

- Implementasi di notebook: `train_reinforce`.
- Menggunakan return kumulatif (episodic).
- Normalisasi return dipakai untuk mengurangi varians.

Contoh : Episode Mini (2 Langkah) dengan Policy Gradient

Lingkungan sederhana:

- Hanya **1 state** (agar fokus ke policy).
- Dua aksi: Left dan Right.
- **Aturan reward (episode 2-langkah)**:

$$r_0 = 0, \qquad r_1 = \begin{cases} +1, & \text{jika di langkah-1 memilih Left} \\ 0, & \text{jika memilih Right} \end{cases}$$

- Diskonto: $\gamma = 0.9$, laju belajar: $\alpha = 0.1$.
- Parameter awal: $\theta_L = 0,\ \theta_R = 0 \Rightarrow \pi(L) = \pi(R) = 0.5$.

Policy & Gradien Log-Softmax (2 Aksi)

Policy (softmax 2-aksi):

$$\pi_\theta(L) = \frac{e^{\theta_L}}{e^{\theta_L} + e^{\theta_R}}, \qquad \pi_\theta(R) = \frac{e^{\theta_R}}{e^{\theta_L} + e^{\theta_R}}.$$

Turunan log-probabilitas:

$$\nabla_{\theta_L} \log \pi_\theta(L) = 1 - \pi_\theta(L), \qquad \nabla_{\theta_R} \log \pi_\theta(L) = -\pi_\theta(R),$$
$$\nabla_{\theta_R} \log \pi_\theta(R) = 1 - \pi_\theta(R), \qquad \nabla_{\theta_L} \log \pi_\theta(R) = -\pi_\theta(L).$$

Aturan REINFORCE (episodik):

$$\theta \leftarrow \theta + \alpha \sum_t \Big(G_t \, \nabla_\theta \log \pi_\theta(a_t|s_t) \Big), \quad G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

Jalannya Episode

1. $\mathbf{t = 0}$: kebetulan $a_0 = \text{Right} \Rightarrow r_0 = 0$.
2. $\mathbf{t = 1}$: ambil $a_1 = \text{Left} \Rightarrow r_1 = +1$ (episode selesai).

Return:

$$G_1 = r_1 = 1, \qquad G_0 = r_0 + \gamma r_1 = 0 + 0.9 \cdot 1 = 0.9.$$

Kontribusi Update

pada $t = 0$ (aksi Right)

Dengan $\pi(L) = \pi(R) = 0.5$ (karena θ awal nol):

$$\nabla_{\theta_R} \log \pi_\theta(R) = 1 - 0.5 = 0.5, \qquad \nabla_{\theta_L} \log \pi_\theta(R) = -0.5.$$

Kontribusi ke parameter (kalikan $G_0 = 0.9$ dan $\alpha = 0.1$):

$$\Delta\theta_R^{(0)} = 0.1 \times 0.9 \times 0.5 = 0.045, \qquad \Delta\theta_L^{(0)} = 0.1 \times 0.9 \times (-0.5) = -0.045.$$

pada $t = 1$ (aksi Left)

$$\nabla_{\theta_L} \log \pi_\theta(L) = 1 - 0.5 = 0.5, \qquad \nabla_{\theta_R} \log \pi_\theta(L) = -0.5.$$

Kontribusi ke parameter (kalikan $G_1 = 1$ dan $\alpha = 0.1$):

$$\Delta\theta_L^{(1)} = 0.1 \times 1 \times 0.5 = 0.05, \qquad \Delta\theta_R^{(1)} = 0.1 \times 1 \times (-0.5) = -0.05.$$

Rekapitulasi Update Parameter

$$\theta_L^{\text{baru}} = 0+(-0.045)+0.05 = \boxed{0.005}, \qquad \theta_R^{\text{baru}} = 0+0.045-0.05 = \boxed{-0.005}.$$

Preferensi selisih: $\theta_L - \theta_R = 0.01$.

$$\pi(L) = \frac{1}{1+e^{-(\theta_L-\theta_R)}} = \frac{1}{1+e^{-0.01}} \approx \boxed{0.5025}.$$

Probabilitas Left naik tipis: $0.5000 \to 0.5025$.

Intuisi yang Didapat

- Aksi **Left** pada $t = 1$ mendapat reward $+1 \Rightarrow$ mendorong $\theta_L \uparrow$, $\theta_R \downarrow$.
- Aksi **Right** pada $t = 0$ tetap *mendapat kredit* karena G_0 memasukkan reward masa depan (γr_1).
- Efeknya saling mengimbangi, namun **netto** tetap mengarah ke Left.

Pelajaran: REINFORCE mengatribusikan reward masa depan ke semua aksi

Catatan: Mengurangi Varians dengan Baseline

Umum dipakai **baseline** b (mis. $V^{\pi}(s_t)$ atau rata-rata return) untuk menurunkan varians:

$$\theta \leftarrow \theta + \alpha \sum_t \Big((G_t - b) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big).$$

Intuisi:

- Jika G_t hanya sedikit di atas baseline, update kecil ($\Rightarrow$ lebih stabil).
- Mengurangi *over-credit* pada aksi di awal episode.

Algoritma Actor–Critic

- **Actor**: meng-update policy parameter θ berdasarkan gradien, mirip REINFORCE.
- **Critic**: aproksimasi value function $V(s;\theta)$ dengan bobot w dan memberikan "kritik" pada actor dengan TD error.

Update Rule

- TD error:

$$\delta_t = r_t + \gamma V(s_{t+1}; w) - V(s_t; w)$$

- Actor:

$$\theta \leftarrow \theta + \alpha\, \delta_t\, \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Critic:

$$w \leftarrow w + \beta\, \delta_t\, \nabla_w V(s_t; w)$$

- Implementasi di notebook: `train_actor_critic`.

Contoh : episode dengan actor critic

Misalkan terdapat episode pendek dengan :

- Diskon, $\gamma = 0.9$
- actor parameter : $\theta = 0.2$
- critic parameter : $w = 0.5$
- learning rate : $\alpha_c = \alpha_a = 0.1$
- Value function : $V(s; w) = w.x(s)$, model linier sederhana dengan fitur $x(s)$.
- Episode: $s_0 \xrightarrow{a_0, r=1} s_1 \xrightarrow{a_1, r=2} s_2$ (terminal).
- Policy : $(\pi_\theta)(a|s)$ dengan parameter θ sederhana (misalkan parameter aksi)
- Ambil inisialisasi parameter awal:
  $V(s_0) = 0.5, V(s_1) = 0.3, V(s_2) = 0$

Langkah 1 : Transisi $s_0 \rightarrow s_1$

- Hitung TD Error di $t = 0$:

$$\delta_0 = r_1 + \gamma V(s_1; w) - V(s_0; w)$$

Misal, estimasi nilai: $V(s_0) = 0.5, V(s_1) = 0.3$.

$$\delta_0 = 1 + 0.9(0.3) - 0.5 = 0.77$$

- update critic

$$w \leftarrow 0.5 + 0.1 \times 0.77 \times x(s_0)$$

(update positif $\rightarrow$ nilai $V(s_0)$ dinaikkan).

- update actor

$$\theta \leftarrow 0.2 + 0.1 \times 0.77 \times \nabla_\theta \log \pi_\theta(a_0|s_0)$$

(update positif $\rightarrow$ aksi a_0 lebih diprioritaskan di s_0)

Langkah 2 : Transisi $s_1 \rightarrow s_2$ (terminal)

- Hitung TD Error di $t = 1$:

$$\delta_1 = r_2 + \gamma V(s_2; w) - V(s_1; w) = 1 + 0.9 \cdot 0 - 0.3 = 1.7$$

- update critic

$$w \leftarrow w + 0.1 \times 1.7 \times x(s_1)$$

(update positif $\rightarrow$ nilai $V(s_1)$ dinaikkan mendekati 2).

- update actor

$$\theta \leftarrow \theta + 0.1 \times 1.7 \times \nabla_\theta \log \pi_\theta(a_1|s_1)$$

(update positif $\rightarrow$ aksi a_1 diperkuat di s_1)

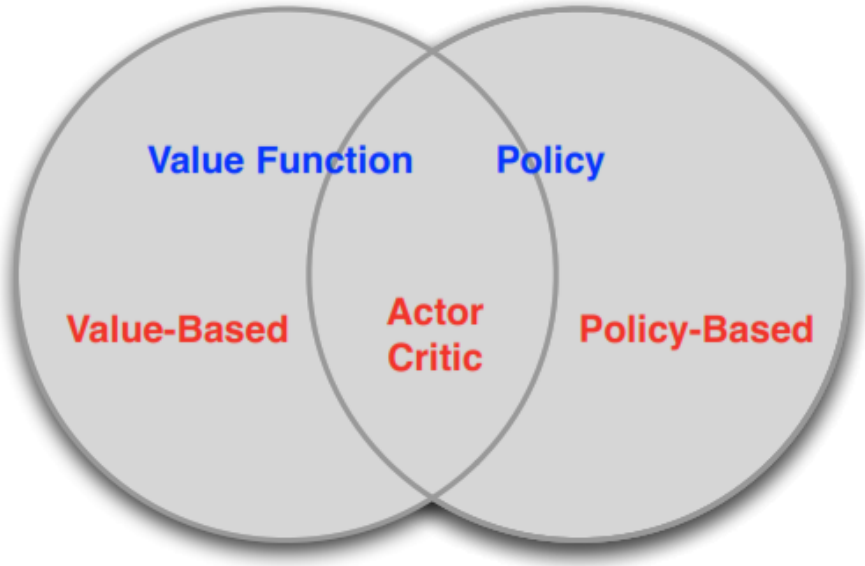
Kesimpulan Episode

- Pada transisi pertama $(s_0 \rightarrow s_1)$: reward lebih besar dari dugaan $\rightarrow$ actor memperkuat aksi a_0.
- Pada transisi kedua $(s_1 \rightarrow s_2)$: reward akhir sangat besar dibanding dugaan $V(s_1) = 0.3 \rightarrow$ actor memperkuat aksi a_1.
- Critic: memperbaiki estimasi $V(s_0)$ dan $V(s_1)$ mendekati reward sebenarnya.
- Actor: memperbaiki kebijakan agar lebih sering memilih aksi yang menghasilkan reward lebih tinggi.
- Kombinasi ini membuat pembelajaran lebih stabil daripada REINFORCE murni.

Policy Gradient vs ActorCritic

Aspek	**REINFORCE (Policy Gradient)**	**ActorCritic**
Sinyal pembelajaran	Return penuh $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$	TD error $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
Waktu update	Setelah episode selesai (Monte Carlo)	Setiap langkah (online / semi-gradient)
Varians	Tinggi (karena seluruh reward episode dipakai)	Lebih rendah (ada baseline berupa $V(s)$)
Komponen	Hanya actor (policy)	Actor (policy) + Critic (value function)
Contoh update episode	$\theta \leftarrow \theta + \alpha G_t \nabla \log \pi$	$\theta \leftarrow \theta + \alpha \delta_t \nabla \log \pi$; $w \leftarrow w + \alpha \delta_t \nabla V$

- **Value Based**
  - Learn values
  - Implicit policy (e.g. ϵ-greedy)
- **Policy Based**
  - No values
  - Learn policy
- **Actor-Critic**
  - Learn values
  - Learn policy



Neural Network dalam PG dan AC

Motivasi: Mengapa Perlu NN?

- State bisa sangat kompleks (gambar, data pasar, sensor).
- Tabel atau fungsi linear tidak cukup.
- Neural Network berperan sebagai **function approximator**.
- Jumlah neuron :
  - input : jumlah variabel state
  - hidden : fleksible
  - output : jumlah kemungkinan aksi (di lingkungan diskrit) atau nilai real/interval (di lingkungan kontinu).

Policy Gradient dengan NN

- di contoh manual : policy $\pi_\theta(a|s)$ hanya berupa probabilitas sederhana dengan θ, 1-2 saja.
- Kalau state s sangat kompleks, tidak mungkin lagi menyimpan tabel probabilitas untuk setiap state. Sehingga, policy $\pi_\theta(a|s)$ dimodelkan dengan NN.:

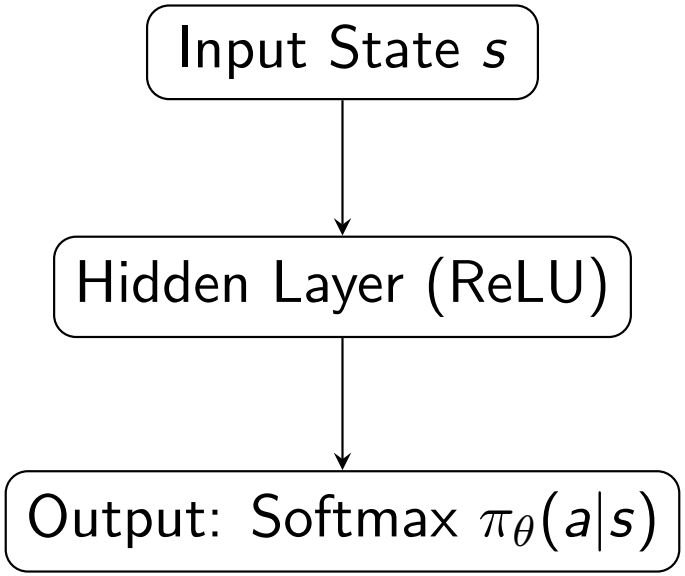
$$\pi_\theta(a|s) = \text{Softmax}(f_\theta(s))$$

- Update parameter (bobot NN):

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Contoh: CartPole (input 4 variabel $\rightarrow$ output probabilitas 2 aksi).

Arsitektur Policy Gradient (dengan NN)



- NN digunakan sebagai **policy approximator**.$ReLU \rightarrow f(x) = max(0, x)$.
- **Input layer**: representasi state (misalnya posisi, kecepatan, data fitur, gambar).
- **hidden layer** : mempelajari representasi yang lebih abstrak.
- **Output layer**: menghasilkan probabilitas (distribusi) aksi berdasarkan state.

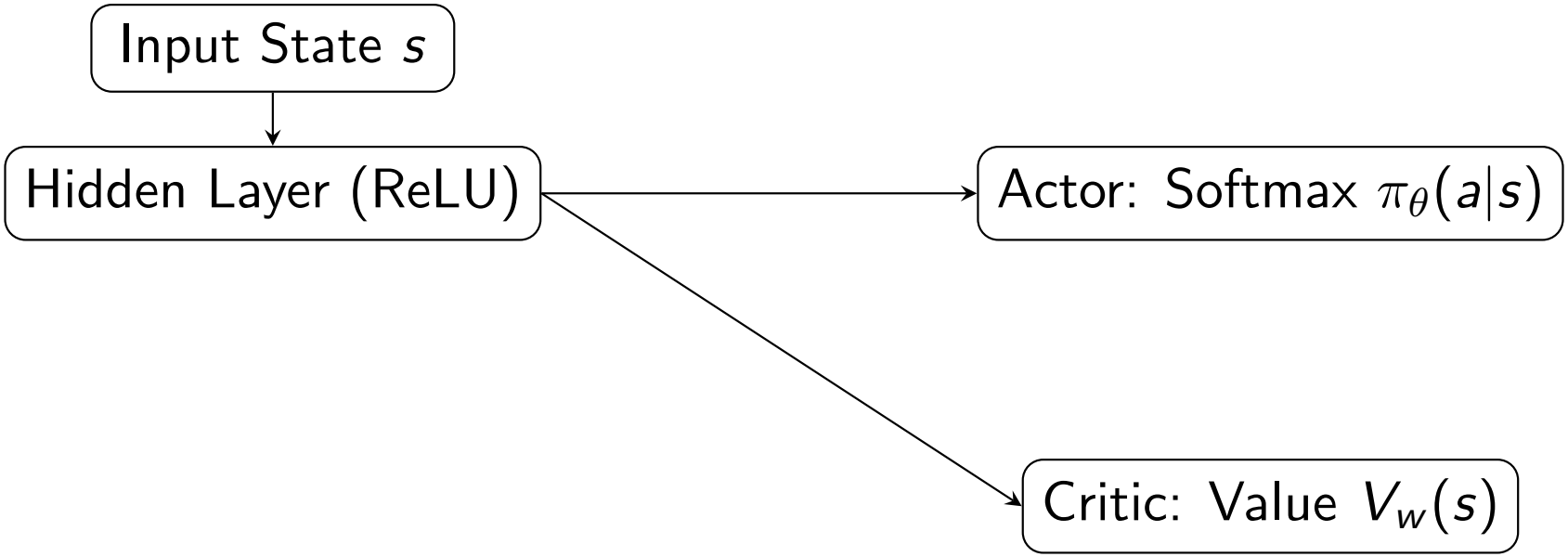
ActorCritic dengan NN

- Actor: policy NN, menghasilkan distribusi aksi.
- Critic: value NN, memperkirakan $V(s)$.
- Update menggunakan TD error:

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

- Actor dan Critic bisa pakai NN terpisah atau berbagi hidden layer.

Arsitektur ActorCritic (dengan NN)



- **Input layer**: representasi state (misalnya posisi, kecepatan, data fitur, gambar).
- **hidden layer** : mempelajari representasi yang lebih abstrak.
- **Output 1 (actor)**: 2 neuron $\rightarrow$ softmax $\rightarrow$ probabilitas.
- **Output 2 (critic)**: 1 neuron $\rightarrow$ linear $\rightarrow$ nilai $V(s)$.

Kadang Actor dan Critic punya network terpisah, kadang mereka berbagi hidden layer lalu bercabang di output.

Lab/Practice

CartPole Environment

- **State (s)**: vektor dengan 4 komponen
  - Posisi kereta (x)
  - Kecepatan kereta ($\dot{x}$)
  - Sudut tiang (θ)
  - Kecepatan sudut tiang ($\dot{\theta}$)
- **Action (a)**:
  - Dorong kereta ke kiri ($a = 0$)
  - Dorong kereta ke kanan ($a = 1$)
- **Reward (r)**:
  - $+1$ untuk setiap langkah jika tiang masih tegak
  - Episode selesai jika tiang jatuh (sudut terlalu besar) atau kereta keluar batas
- **Objective**: memaksimalkan *cumulative reward*, yaitu menjaga tiang tetap tegak selama mungkin.

Hyperparameter Penting di CartPole

Hyperparameter	**Pengaruh Utama**
Hidden layer (ukuran NN)	Semakin besar $\rightarrow$ kapasitas representasi naik, tapi bisa overfitting / lambat.
Learning rate (α)	Besar $\rightarrow$ cepat konvergen tapi tidak stabil; Kecil $\rightarrow$ stabil tapi lambat.
Gamma (γ)	Dekat 1 $\rightarrow$ fokus reward jangka panjang; Kecil $\rightarrow$ lebih myopic (jangka pendek).
Jumlah episode	Lebih banyak $\rightarrow$ peluang belajar pola stabil; Tapi waktu training lebih lama.

Pesan kunci: Eksperimen ini sensitif terhadap setting hyperparameter, jadi cobalah beberapa variasi untuk melihat dampaknya.

Tujuan Eksperimen CartPole

- Membandingkan dua algoritma:
  - **REINFORCE (Policy Gradient)**: update berbasis return episode penuh
  - **ActorCritic**: update berbasis TD target dengan bantuan value function
- Menjalankan eksperimen 400 episode dan memantau reward
- Mengamati perbedaan *stabilitas*, *kecepatan belajar*, dan *fluktuasi*

Lab / Practice

CartPole Experiments

- Gunakan kode dari notebook Colab (ACS_Week04)
- Fungsi yang digunakan : `train_reinforce` dan `train_actor_critic`
- Jalankan eksperimen 500 episode, catat log tiap 50 episode
- Visualisasi learning curve (total reward per episode dan moving average)
- Bandingkan kedua algoritma dari sisi:
  - Kecepatan konvergensi.
  - Stabilitas learning curve.
  - Variasi performa antar-episode.

Perbandingan Hasil Eksperimen

1. Kecepatan konvergensi

- **REINFORCE**: reward cepat naik hingga menembus 100–200 pada episode awal (≈150–200). Artinya ada *learning progress* yang cukup cepat.
- **Actor–Critic**: reward stagnan di 8–10 sepanjang 400 episode → hampir tidak ada konvergensi.

Jawaban: REINFORCE lebih cepat konvergen, sedangkan Actor–Critic gagal belajar.

2. Stabilitas learning curve

- **REINFORCE**: sangat fluktuatif, reward naik-turun tajam; moving average sempat tinggi tapi turun lagi.
- **Actor–Critic**: stabil, tapi stabil di level reward rendah (tidak naik).

Jawaban: Actor–Critic lebih stabil, tapi stabil di titik yang salah. REINFORCE tidak stabil, tetapi mampu mencapai reward tinggi.

Perbandingan Hasil Eksperimen

3. Variasi performa antar-episode

- **REINFORCE**: variasi besar antar episode (ada episode yang tembus >300 reward, ada juga drop ke <50).
- **Actor–Critic**: variasi kecil, hampir semua episode rewardnya mirip (sekitar 8–10).

Jawaban: REINFORCE punya variasi antar episode yang besar; Actor–Critic variasinya kecil tapi "mentok rendah".

Catatan Diskusi & Refleksi

- Amati perbedaan stabilitas REINFORCE vs ActorCritic
- Mengapa normalisasi return di REINFORCE dapat membantu?
- Apa trade-off biasvarians akibat penggunaan TD target di ActorCritic?
- Dalam eksperimen ini: ActorCritic stuck di reward rendah, REINFORCE fluktuatif tapi sempat mencapai reward tinggi. Mengapa?
- Opsional: tambahkan baseline pada REINFORCE untuk mengurangi varians

Refleksi & Jawaban

- **Stabilitas:** REINFORCE fluktuatif tapi bisa tinggi; ActorCritic stabil tapi stuck rendah.
- **Normalisasi return:** mengurangi varians update REINFORCE, gradien lebih stabil.
- **Biasvarians trade-off:** ActorCritic lebih stabil (varians rendah) tapi bias tinggi jika value function tidak akurat.
- **Hasil eksperimen:** ActorCritic gagal belajar, REINFORCE meski fluktuatif kadang tembus reward tinggi.
- **Baseline:** mengurangi varians tanpa mengubah ekspektasi gradien.

Arahan Perbaikan

1. **Eksperimen Hyperparameter**
   - Coba variasi learning rate: 3×10^{-3}, 1×10^{-3}, 5×10^{-4}
   - Ubah nilai γ: 0.95 atau 0.97
2. **Arsitektur Neural Network**
   - Tambahkan hidden layer atau neuron (misalnya 2 layer dengan 128-256 unit)
   - Gunakan aktivasi yang stabil seperti ReLU atau LeakyReLU
   - Coba dropout kecil (misalnya 0.1-0.2) untuk mengurangi overfitting
3. **Stabilisasi ActorCritic**
   - Periksa kembali perhitungan *advantage* $(r + \gamma V(s') - V(s))$
   - Gunakan normalisasi reward atau advantage
4. **Baseline pada REINFORCE**
   - Gunakan value function sebagai baseline untuk mengurangi varians
5. **Optimizers dan Regularisasi**
   - Uji optimizer lain seperti `RMSprop`
   - Tambahkan *entropy bonus* untuk menjaga eksplorasi

Perbaikan Arsitektur Neural Network (2)

1. Tambah hidden layer

```
self.net = nn.Sequential(
    nn.Linear(obs_dim, hidden), nn.ReLU(),
    nn.Linear(hidden, hidden), nn.ReLU(),
    nn.Linear(hidden, act_dim)
)
```

2. Ganti fungsi aktivasi

```
self.net = nn.Sequential(
    nn.Linear(obs_dim, hidden), nn.LeakyReLU(0.01),
    nn.Linear(hidden, hidden), nn.LeakyReLU(0.01),
    nn.Linear(hidden, act_dim)
)
```

Perbaikan Arsitektur Neural Network (2)

3. Tambahkan dropout

```
self.net = nn.Sequential(
    nn.Linear(obs_dim, hidden), nn.ReLU(),
    nn.Dropout(p=0.1),
    nn.Linear(hidden, hidden), nn.ReLU(),
    nn.Dropout(p=0.1),
    nn.Linear(hidden, act_dim)
)
```

4. Ubah ukuran hidden units

```
policy = PolicyNet(obs_dim, act_dim, hidden=256)
```

Normalisasi pada Actor-Critic (3)

Pada bagian TD error :

```
# Normalisasi delta (advantage)
adv = (delta - delta.mean()) / (delta.std() + 1e-8)

loss_actor = -logp * adv.detach()
loss_critic = delta.pow(2)
```

Baseline pada REINFORCE (4)

- Update REINFORCE standar menggunakan *return* penuh G_t:

$$\nabla J(\theta) \approx \sum_t G_t \nabla \log \pi_\theta(a_t|s_t)$$

- Varians dari G_t sangat tinggi $\Rightarrow$ gradien tidak stabil.
- Hasil: kurva reward naik-turun tajam, sulit mencapai stabilitas.

Ide Baseline

- Untuk mengurangi varians, tambahkan baseline $b(s_t)$:

$$\nabla J(\theta) \approx \sum_t \big(G_t - b(s_t)\big) \nabla \log \pi_\theta(a_t|s_t)$$

- Syarat: baseline tidak boleh bergantung pada action.
- Pilihan umum: gunakan **value function** $V^\pi(s_t)$.
- Maka: $A_t = G_t - V(s_t)$ disebut **advantage**.
- Keuntungan Menggunakan Baseline
  - Mengurangi varians gradien $\Rightarrow$ training lebih stabil.
  - Tidak mengubah ekspektasi gradien (estimasi tetap tidak bias).
  - Membantu policy fokus pada *advantage* (aksi lebih baik/lebih buruk dari rata-rata).

Revisi Python: REINFORCE dengan Baseline

```
# Hitung return
returns = compute_returns(rewards, gamma)  # shape [T]
# Stack states -> prediksi V(s_t)
states_t = torch.stack(states)                    # [T, obs_dim]
values   = vnet(states_t)                         # [T]
# Advantage dengan value-baseline
advantages = returns - values.detach()
# (opsional) normalisasi advantage (lebih stabil)
advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)
# Gabung log_probs
log_probs_t = torch.stack(log_probs)              # [T]
# Loss policy (pakai advantage)
loss_policy = -(log_probs_t * advantages).sum()
# Loss value (regresi V ke return)
loss_value  = (values - returns).pow(2).mean()
# (opsional) entropy bonus supaya eksplorasi terjaga
# entropy = torch.distributions.Categorical(logits=...).entropy().mean()
# loss_entropy = - ent_coef * entropy
loss = loss_policy + loss_value  # + loss_entropy (jika dipakai)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Optimizers dan Regularisasi (5)

1. Optimizer alternatif : Awalnya pakai Adam, kemudian bisa dicoba RMSprop atau SGD+momentum untuk melihat stabilitas training.
2. Entropy bonus :
   - Distribusi policy yang terlalu tajam (peaked) membuat eksplorasi cepat hilang.
   - Tambahkan term entropy ke dalam loss untuk meratakan distribusi aksi, sehingga agent tetap eksploratif.

Optimizers dan Regularisasi (5)

Optimizer Alternatif

```
#awalnya :
optimizer = optim.Adam(net.parameters(), lr=lr)
#jika pakai RMSprop :
optimizer = optim.RMSprop(net.parameters(), lr=lr)
#atau SGD + momentum
optimizer = optim.SGD(net.parameters(), lr=lr, momentum=0.9)
```

Entropy Bonus

```
#awalnya :
loss = loss_policy + loss_value
#setelah ditambah entropy bonus :
entropy = dist.entropy().mean()   # dist = distribusi aksi dari policy
ent_coef = 0.01                   # koefisien entropy bonus
loss = loss_policy + loss_value - ent_coef * entropy
```