

Laporan Praktikum Kontrol Cerdas

Minggu ke-4

Nama : Kresensia Meita Indar Mayaningsih

NIM : 224308087

Kelas : TKA 7D

Akun Github (Tautan) : <https://github.com/kzmeita>

Student Lab Assistant :

1. Judul Percobaan :

Reinforcement Learning for Autonomous Control

2. Tujuan Percobaan

Tujuan dari praktikum minggu ke-4 :

- a) Mahasiswa dapat memahami konsep dasar Reinforcement Learning (RL) dalam sistem kendali.
- b) Mahasiswa dapat mengimplementasikan agen RL menggunakan algoritma Deep Q-Network (DQN).
- c) Mahasiswa dapat menggunakan OpenAI Gym sebagai simulasi lingkungan untuk pelatihan RL.
- d) Mahasiswa dapat melatih dan menguji agen RL untuk mengontrol lingkungan secara otonom.
- e) Mahasiswa dapat menggunakan GitHub untuk version control dan dokumentasi praktikum.

3. Landasan Teori

1. Reinforcement Learning (RL)

Reinforcement Learning (RL) merupakan salah satu paradigma dalam machine learning yang berfokus pada bagaimana suatu agen (agent) dapat belajar mengambil keputusan secara otonom melalui interaksi dengan lingkungannya (environment). Dalam RL, proses pembelajaran terjadi berdasarkan prinsip trial and error, di mana agen mencoba melakukan aksi (action) tertentu pada suatu kondisi (state), kemudian menerima umpan balik berupa reward atau hukuman (penalty). Tujuan utama RL adalah menemukan strategi pengambilan keputusan (policy) yang dapat

memaksimalkan total reward jangka panjang (cumulative reward) (Sutton & Barto, 2018).

Secara umum, RL dapat dimodelkan sebagai Markov Decision Process (MDP) yang terdiri dari:

- a. State (S): kondisi lingkungan yang diamati oleh agen.
- b. Action (A): keputusan atau langkah yang dapat diambil agen.
- c. Reward (R): nilai yang diberikan oleh lingkungan sebagai umpan balik.
- d. Policy (π): strategi agen dalam memilih aksi berdasarkan state.
- e. Value Function (V): ekspektasi reward jangka panjang dari suatu state.

2. Q-Learning

Salah satu algoritma RL klasik adalah Q-Learning, yang menggunakan tabel Q (Q-table) untuk menyimpan estimasi nilai (value) dari setiap pasangan state-action. Nilai ini dikenal sebagai Q-value dan dihitung menggunakan persamaan Bellman:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Keterangan:

s : state saat ini

a : action yang diambil

r : reward yang diterima

γ : discount factor ($0 < \gamma \leq 1$), menunjukkan seberapa penting reward jangka Panjang

α : learning rate, menentukan seberapa besar pembaruan nilai Q

Q-Learning efektif untuk permasalahan dengan jumlah state terbatas. Namun, ketika jumlah state sangat besar atau kontinu (misalnya input berupa gambar atau sensor kompleks), Q-table menjadi tidak efisien dan sulit diterapkan.

3. Deep Q-Network (DQN)

Untuk mengatasi keterbatasan Q-Learning, Mnih et al. (2015) memperkenalkan Deep Q-Network (DQN), yang menggabungkan Q-Learning dengan Deep Neural Network (DNN). Alih-alih menyimpan nilai Q pada tabel, DQN menggunakan jaringan saraf untuk memperkirakan fungsi Q. Dengan cara ini, DQN mampu memproses input berukuran besar, termasuk data visual.

Beberapa teknik penting pada DQN:

- a. Experience Replay: menyimpan pengalaman agen (state, action, reward, next state) ke dalam memori replay buffer. Data tersebut diambil secara acak untuk

melatih model, sehingga mengurangi korelasi data dan meningkatkan stabilitas pembelajaran.

- b. Target Network: salinan dari jaringan utama (primary network) yang diperbarui secara periodik. Hal ini mencegah perubahan bobot yang terlalu cepat dan mengurangi osilasi dalam training.
 - c. Epsilon-Greedy Policy: strategi eksplorasi di mana agen terkadang memilih aksi acak (eksplorasi) dengan probabilitas ϵ , dan sisanya memilih aksi terbaik (eksploitasi). Nilai ϵ biasanya menurun seiring waktu (ϵ -decay).
 - d. Dengan kombinasi ini, DQN terbukti mampu mencapai performa setara manusia dalam beberapa permainan Atari klasik.
4. Implementasi RL dalam Simulasi

Dalam konteks pembelajaran, OpenAI Gym banyak digunakan sebagai platform simulasi untuk eksperimen RL. Dua environment yang sering digunakan adalah:

a. CartPole

CartPole adalah simulasi klasik di mana agen harus menyeimbangkan sebuah tiang (pole) yang dipasang di atas kereta (cart). Agen menerima reward jika berhasil menjaga tiang tetap tegak dalam jangka waktu tertentu. Masalah ini menjadi benchmark sederhana untuk menguji algoritma RL karena memiliki state kontinu (posisi cart, kecepatan, sudut tiang, kecepatan sudut). Dengan DQN, agen dapat belajar strategi optimal agar tiang tidak jatuh meskipun pada awalnya sering gagal.

b. MountainCar

MountainCar adalah environment di mana sebuah mobil kecil berada di lembah antara dua bukit. Tujuannya adalah membuat mobil mencapai puncak bukit di sebelah kanan. Namun, mesin mobil terlalu lemah untuk langsung mendaki bukit. Oleh karena itu, agen harus belajar menghasilkan momentum dengan cara bolak-balik hingga mampu mencapai puncak. Reward diberikan saat mobil berhasil mencapai tujuan, sedangkan setiap langkah yang gagal akan menambah penalti. Kasus ini menantang karena reward yang jarang (sparse reward), sehingga agen membutuhkan eksplorasi yang lebih banyak untuk menemukan solusi.

Melalui kedua simulasi ini, mahasiswa dapat memahami perbedaan karakteristik environment: CartPole relatif lebih cepat memberikan feedback

reward sehingga agen bisa belajar dengan cepat, sedangkan MountainCar membutuhkan strategi eksplorasi dan kesabaran lebih karena reward hanya muncul jika tujuan tercapai.

5. Relevansi DQN dalam Praktikum

Dalam praktikum ini, penerapan DQN memberikan pengalaman langsung tentang bagaimana sebuah agen belajar melalui interaksi dengan lingkungan yang berbeda. Eksperimen dilakukan dengan memvariasikan parameter penting seperti learning rate, discount factor, dan epsilon decay untuk melihat pengaruhnya terhadap kecepatan konvergensi, stabilitas training, dan nilai reward yang diperoleh.

Dari hasil uji coba, terlihat bahwa penerapan DQN dapat mengatasi keterbatasan Q-Learning tradisional, meskipun tetap diperlukan teknik tambahan seperti target network dan tuning hyperparameter agar performa agen optimal di berbagai environment.

4. Analisis dan Diskusi

A. Analisis

a) Bagaimana performa agen dalam mengontrol environment CartPole?

Pada environment CartPole, agen dengan algoritma DQN menunjukkan performa yang cukup baik setelah melalui sejumlah episode pelatihan. Agen mampu belajar untuk menjaga keseimbangan tiang (pole) dalam waktu relatif singkat, karena reward diberikan hampir setiap langkah ketika tiang tetap tegak. Hal ini memudahkan agen untuk mengidentifikasi pola aksi yang mengarah pada reward lebih tinggi. Dari hasil pengamatan, skor agen meningkat secara bertahap seiring bertambahnya episode, hingga mampu mempertahankan tiang selama batas maksimum episode (1000 langkah). Hal ini membuktikan bahwa DQN efektif digunakan pada permasalahan dengan feedback reward yang kontinu.

b) Bagaimana perubahan parameter (misal: gamma, epsilon, learning rate) mempengaruhi kinerja agen?

- Gamma (γ): Semakin tinggi nilai gamma, agen cenderung mempertimbangkan reward jangka panjang, sehingga strategi yang dipelajari lebih stabil. Namun, jika terlalu tinggi mendekati 1, proses konvergensi menjadi lebih lambat.
- Epsilon (ϵ): Nilai epsilon mengatur keseimbangan antara eksplorasi dan eksploitasi. Pada awal pelatihan, epsilon tinggi memaksa agen lebih banyak mencoba aksi acak. Seiring waktu, epsilon menurun (ϵ -decay) sehingga agen

lebih fokus pada aksi terbaik yang telah dipelajari. Jika epsilon terlalu cepat turun, agen bisa terjebak pada solusi suboptimal.

- Learning Rate (α): Learning rate yang besar mempercepat pembaruan bobot jaringan, namun bisa menyebabkan instabilitas dan osilasi nilai reward. Sebaliknya, learning rate yang terlalu kecil memperlambat proses pembelajaran.
- Dari eksperimen, kombinasi parameter yang seimbang memberikan hasil terbaik, sedangkan parameter ekstrem (misalnya epsilon langsung sangat kecil atau learning rate terlalu besar) membuat agen sulit mencapai performa optimal.

c) Apa tantangan yang muncul selama pelatihan agen RL?

Tantangan utama yang muncul selama pelatihan agen RL adalah:

- Stabilitas Training: Kadang reward berfluktuasi tajam akibat pembaruan bobot jaringan yang tidak stabil.
- Sparse Reward: Pada environment seperti MountainCar, reward hanya diperoleh jika mobil berhasil mencapai puncak. Hal ini membuat proses pembelajaran lebih sulit karena agen harus menemukan strategi eksplorasi yang tepat.
- Tuning Hyperparameter: Pemilihan parameter seperti gamma, epsilon decay, dan learning rate sangat berpengaruh terhadap hasil akhir. Perlu dilakukan eksperimen berulang untuk menemukan kombinasi terbaik.
- Overfitting terhadap Environment: Jika agen terlalu lama dilatih dengan parameter tertentu, agen bisa terjebak hanya pada strategi spesifik tanpa kemampuan generalisasi.

B. Diskusi

1. Apa perbedaan utama antara Reinforcement Learning dan metode supervised learning dalam sistem kendali?

Perbedaan utama terletak pada cara agen menerima umpan balik. Pada supervised learning, model belajar dari dataset yang berisi pasangan input-output yang benar, sehingga umpan balik bersifat langsung dan eksplisit. Sementara pada reinforcement learning, agen tidak diberikan label benar-salah, melainkan harus mengeksplorasi sendiri lingkungan dan menerima reward sebagai umpan balik. Dengan kata lain, supervised learning bersifat passive learning, sedangkan RL bersifat interactive learning di mana agen harus aktif mencoba berbagai aksi.

2. Bagaimana strategi eksplorasi (exploration) dan eksploitasi (exploitation) dapat dioptimalkan pada agen RL?

Strategi eksplorasi-eksploitasi merupakan tantangan penting dalam RL. Optimasi dapat dilakukan dengan:

- Menggunakan ϵ -greedy policy dengan ϵ -decay, di mana agen pada awalnya banyak bereksperimen (eksplorasi), lalu secara bertahap lebih sering mengeksploitasi strategi terbaik.
 - Menerapkan metode eksplorasi adaptif, misalnya Boltzmann exploration atau Upper Confidence Bound (UCB) untuk menyeimbangkan trade-off eksplorasi dan eksploitasi.
 - Menambahkan intrinsic reward seperti curiosity-driven exploration, sehingga agen tetap terdorong mencoba hal baru meskipun reward eksternal jarang.
3. Potensi aplikasi lain dari RL dalam sistem kendali nyata apa saja yang dapat diimplementasikan?
- Reinforcement Learning memiliki potensi luas dalam berbagai bidang sistem kendali nyata, di antaranya:
 - Robotika: pengendalian robot otonom untuk navigasi, manipulasi objek, atau kerja kolaboratif di pabrik.
 - Kendaraan Otonom: pengendalian mobil self-driving dalam menghadapi kondisi lalu lintas kompleks.
 - Smart Grid & Energi: pengaturan distribusi energi listrik agar lebih efisien.
 - Industri Manufaktur: optimasi jalur produksi dan pemeliharaan prediktif.
 - Kendali Jaringan Komunikasi: optimasi routing dan manajemen sumber daya pada jaringan 5G/IoT.

Hal ini menunjukkan bahwa RL bukan hanya teori dalam simulasi, tetapi juga berpotensi besar untuk diimplementasikan dalam sistem kendali nyata yang kompleks.

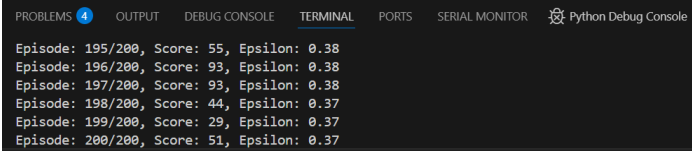
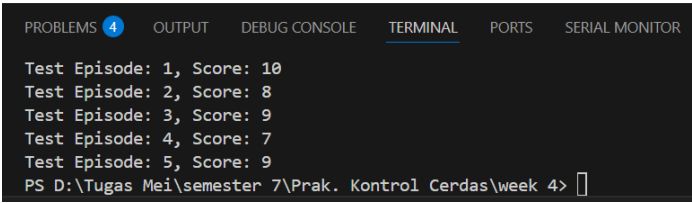
5. Assignment

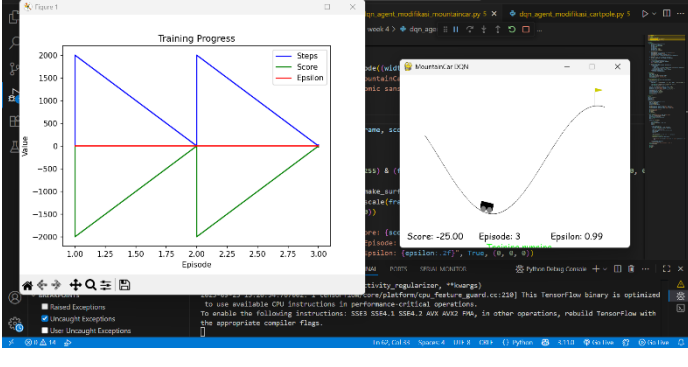
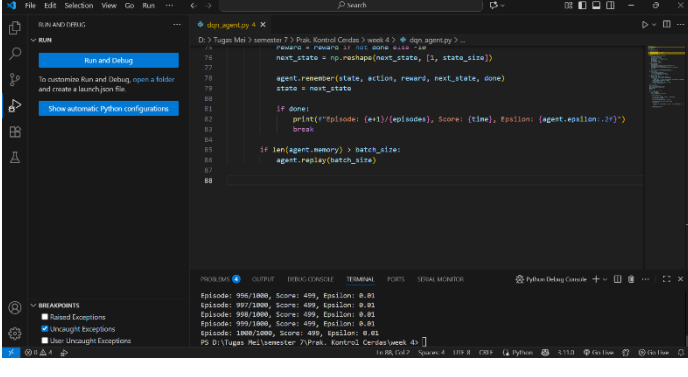
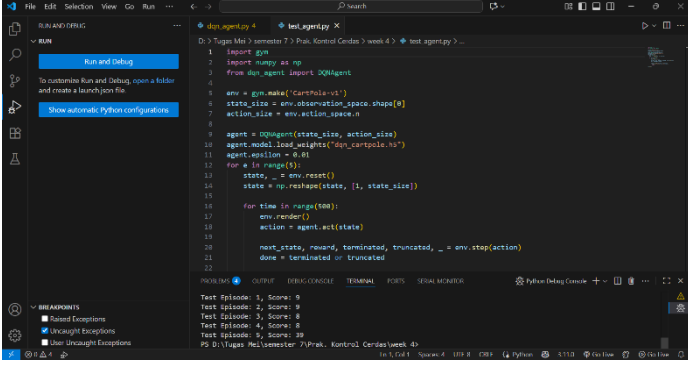
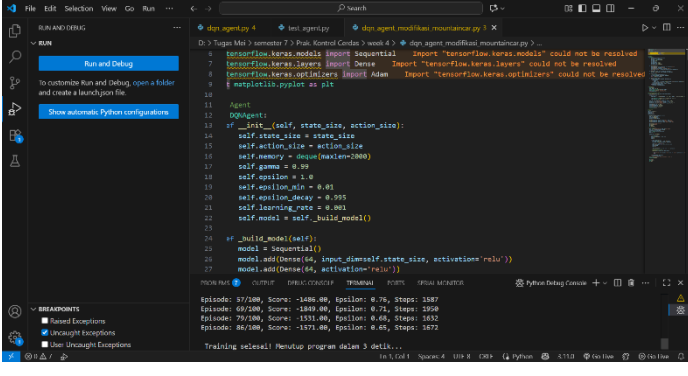
Pada praktikum kali ini kami mempelajari penerapan Deep Q-Network (DQN) sebagai salah satu metode pada Reinforcement Learning (RL). DQN merupakan pengembangan dari algoritma Q-Learning yang menggabungkannya dengan Deep Neural Network agar mampu menangani permasalahan state yang besar maupun kontinu. Pada Q-Learning klasik, nilai dari setiap aksi di setiap state disimpan dalam sebuah tabel yang disebut Q-table. Namun, pendekatan ini memiliki keterbatasan karena ketika jumlah state sangat besar (misalnya data sensor yang kompleks atau gambar dari kamera), ukuran Q-table akan menjadi sangat besar dan tidak efisien. Untuk mengatasi hal ini, DQN menggunakan jaringan saraf dalam memperkirakan fungsi Q, sehingga pembelajaran dapat berjalan lebih efektif. Dalam praktikum ini, kami melakukan beberapa modifikasi terhadap

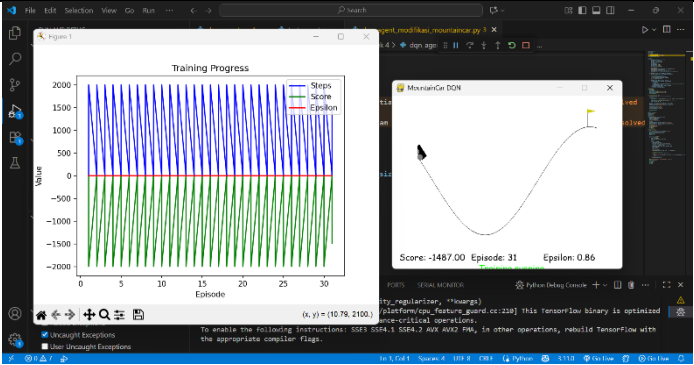
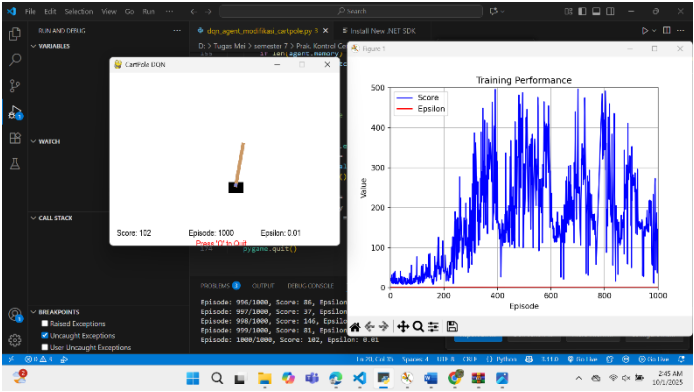
algoritma DQN, salah satunya dengan menambahkan target network agar proses pelatihan lebih stabil. Target network bekerja dengan cara menyalin parameter dari model utama secara periodik, sehingga update bobot tidak terlalu cepat berubah dan mengurangi osilasi saat proses training. Selain itu, kami juga menguji agen pada beberapa environment berbeda seperti LunarLander-v2 dan MountainCar-v0, lalu membandingkan kinerja agen pada masing-masing environment tersebut. Untuk memperdalam pemahaman, dilakukan pula eksperimen dengan mengubah beberapa parameter penting, seperti learning rate, discount factor (γ), dan epsilon decay, sehingga dapat diamati pengaruhnya terhadap kinerja agen baik dari segi kecepatan konvergensi maupun nilai reward yang diperoleh. Semua hasil eksperimen kemudian divisualisasikan dalam bentuk grafik reward, jumlah langkah, serta perubahan nilai epsilon pada setiap episode. Terakhir, seluruh kode program, hasil eksperimen, serta laporan singkat dipublikasikan ke dalam repository GitHub sebagai dokumentasi. Dengan praktikum ini, kami memahami bahwa penerapan DQN memberikan solusi terhadap keterbatasan Q-Learning tradisional, meskipun tetap memerlukan teknik tambahan seperti target network dan tuning parameter agar agen dapat belajar secara optimal pada environment yang berbeda.

6. Data dan Output Hasil Pengamatan

Data yang diperoleh pada praktikum minggu ke-1 disajikan dalam tabel dibawah ini:

No	Variabel	Hasil Pengamatan
1.	Hasil training sebelum dimodifikasi dengan 200 episode. Didapatkan score 51 dan epsilon 0.37	 <pre> PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR Python Debug Console Episode: 195/200, Score: 55, Epsilon: 0.38 Episode: 196/200, Score: 93, Epsilon: 0.38 Episode: 197/200, Score: 93, Epsilon: 0.38 Episode: 198/200, Score: 44, Epsilon: 0.37 Episode: 199/200, Score: 29, Epsilon: 0.37 Episode: 200/200, Score: 51, Epsilon: 0.37 </pre>
2.	Hasil test sebelum dimodifikasi dengan 200 episode (test_agent.py) didapatkan test episode 5 dan score 9.	 <pre> PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR Test Episode: 1, Score: 10 Test Episode: 2, Score: 8 Test Episode: 3, Score: 9 Test Episode: 4, Score: 7 Test Episode: 5, Score: 9 PS D:\Tugas Mei\semester 7\Prak. Kontrol Cerdas\week 4> </pre>

3.	Hasil program dqn_agent setelah dimodifikasi dengan 200 episode.	 <p>The screenshot shows a Jupyter Notebook with two plots. The left plot, titled 'Training Progress', is a line graph with 'Episode' on the x-axis (ranging from 1.00 to 3.00) and 'Value' on the y-axis (ranging from -2000 to 2000). It contains three lines: 'Steps' (blue), 'Score' (green), and 'Epsilon' (red). The 'Steps' line starts at 2000 and decreases to 0 by episode 2.00. The 'Score' line starts at -2000 and increases to 0 by episode 2.00. The 'Epsilon' line is a constant red line at 0. The right plot shows a MountainCar environment with a car at the bottom of a valley, with a score of -25.00, episode 3, and epsilon of 0.99.</p>
4.	Hasil training dqn_agent sebelum modifikasi dengan 1000 episode. Didapatkan score 499 dan episode 0.01.	 <p>The screenshot shows a Jupyter Notebook with a code cell and a terminal output. The code cell contains the following code: <pre> 78 next_state = np.reshape(next_state, [1, state_size]) 79 agent.reset(next_state, action, reward, next_state, done) 80 state = next_state 81 if done: 82 print("Episode: (%+1)/(episodes), Score: (time), Epsilon: (agent.epsilon:.2f)") 83 break 84 if len(agent.memory) > batch_size: 85 agent.replay(batch_size) 86 87 </pre> <p>The terminal output shows the following results:</p> <pre> Episode: 999/1000, Score: 499, Epsilon: 0.01 Episode: 999/1000, Score: 499, Epsilon: 0.01 Episode: 999/1000, Score: 499, Epsilon: 0.01 Episode: 1000/1000, Score: 499, Epsilon: 0.01 </pre> </p>
5	Hasil test sebelum modifikasi, dengan 1000 episode didapatkan test episode 5 dan score 39	 <p>The screenshot shows a Jupyter Notebook with a code cell and a terminal output. The code cell contains the following code: <pre> 1 import gym 2 import numpy as np 3 from dqn_agent import DQNAgent 4 5 env = gym.make('CartPole-v1') 6 state_size = env.observation_space.shape[0] 7 action_size = env.action_space.n 8 9 agent = DQNAgent(state_size, action_size) 10 agent.load_weights('dqn_cartpole.h5') 11 agent.epsilon = 0.01 12 for e in range(5): 13 state, _ = env.reset() 14 state = np.reshape(state, [1, state_size]) 15 for time in range(100): 16 env.render() 17 action = agent.act(state) 18 next_state, reward, terminated, truncated, _ = env.step(action) 19 done = terminated or truncated 20 </pre> <p>The terminal output shows the following results:</p> <pre> Test Episode: 1, Score: 9 Test Episode: 2, Score: 9 Test Episode: 3, Score: 8 Test Episode: 4, Score: 8 Test Episode: 5, Score: 39 </pre> </p>
6	Hasil Uji agen pada environment MountainCar-v0. Didapatkan hasil episode mencapai 86, score -1571.00, epsilon 0.65, dan 1672 steps	 <p>The screenshot shows a Jupyter Notebook with a code cell and a terminal output. The code cell contains the following code: <pre> 11 Agent 12 DQNAgent: 13 def __init__(self, state_size, action_size): 14 self.state_size = state_size 15 self.action_size = action_size 16 self.memory = deque(maxlen=2000) 17 self.gamma = 0.99 18 self.epsilon = 1.0 19 self.epsilon_min = 0.01 20 self.epsilon_decay = 0.995 21 self.learning_rate = 0.001 22 self.model = self._build_model() 23 24 def _build_model(self): 25 model = Sequential() 26 model.add(Dense(64, input_dim=self.state_size, activation='relu')) 27 model.add(Dense(64, activation='relu')) </pre> <p>The terminal output shows the following results:</p> <pre> Episode: 57/100, Score: -1480.00, Epsilon: 0.70, Steps: 1587 Episode: 60/100, Score: -1580.00, Epsilon: 0.70, Steps: 2500 Episode: 70/100, Score: -1531.00, Epsilon: 0.60, Steps: 1632 Episode: 86/100, Score: -1571.00, Epsilon: 0.65, Steps: 1672 </pre> </p>

		
7	Hasil Uji agen pada environment cartpole DQN-v1. Didapatkan hasil episode mencapai 1000, score 102, dan epsilon 0.01	

7. Kesimpulan

Berdasarkan hasil praktikum Reinforcement Learning menggunakan algoritma Deep Q-Network (DQN), dapat disimpulkan bahwa:

1. Penerapan DQN berhasil mengatasi keterbatasan Q-Learning tradisional dengan memanfaatkan jaringan saraf tiruan untuk memperkirakan fungsi Q, sehingga agen mampu belajar pada environment dengan state yang kompleks.
2. Pada environment CartPole, agen menunjukkan performa sangat baik dengan cepat mencapai nilai reward maksimum setelah sejumlah episode, membuktikan bahwa DQN efektif pada kasus dengan feedback reward yang padat (dense reward).
3. Pada environment MountainCar, pembelajaran berlangsung lebih sulit karena sifat reward yang jarang (sparse reward), sehingga agen memerlukan eksplorasi lebih lama untuk menemukan strategi yang tepat. Hal ini menunjukkan bahwa karakteristik environment sangat memengaruhi tingkat kesulitan pembelajaran.
4. Variasi parameter seperti gamma (γ), epsilon decay, dan learning rate memiliki pengaruh signifikan terhadap performa agen. Kombinasi parameter yang tepat mempercepat konvergensi, sedangkan pengaturan ekstrem dapat menyebabkan pembelajaran tidak stabil atau gagal.
5. Penggunaan target network terbukti meningkatkan stabilitas pelatihan dengan mengurangi osilasi nilai Q dan memperbaiki konsistensi reward yang diperoleh agen.

8. Saran

1. Untuk penelitian lanjutan, dapat dilakukan eksplorasi dengan algoritma RL lain seperti Double DQN, Dueling DQN, atau Prioritized Experience Replay agar performa agen semakin optimal.
2. Perlu dilakukan tuning hyperparameter yang lebih sistematis, misalnya dengan grid search atau Bayesian optimization, sehingga didapatkan kombinasi parameter yang lebih optimal pada berbagai environment.
3. Dapat dilakukan eksperimen pada environment yang lebih kompleks, misalnya LunarLander-v2 atau Atari games, agar pemahaman tentang keterbatasan dan keunggulan DQN lebih mendalam.
4. Implementasi dapat diperluas ke sistem kendali nyata (misalnya robot otonom sederhana) untuk menguji performa RL di luar simulasi.
5. Dokumentasi dan visualisasi hasil pelatihan (reward, loss, epsilon decay) sebaiknya dibuat lebih lengkap agar memudahkan analisis dan perbandingan antar eksperimen.

9. Daftar Pustaka

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.
- François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4), 219–354.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.
- Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Silver, D., & Sutton, R. S. (2018). Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).
- Arulkumar, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6), 26–38.