**Supervised Learning Classfication**

**Studi kasus: Analisis dan Klarifikasi Permainan Snake**

**Pada q-learning**

Meta Dewi Anggraeni

Teknologi Informasi/Informatika

Jl.angkrek Situ No.19,Sumedang 45323,Indonesia

Email:220660121017@student.unsap.ac.id

|  |
| --- |
| ABSTRACT  The Snake game is one of the classic games in which the player controls a snake to eat food that appears randomly on the screen, with each food eaten extending the length of the snake. The main challenge is to avoid collisions with walls or with the snake's own body. Using Q-learning, a reinforcement learning algorithm, we explore the optimal way to control snake movement in the game Snake. Although the basic principles of the Snake game are simple, complexity increases as the snake grows, making navigation strategies more complicated. This provides an interesting opportunity to apply machine learning algorithms, in particular Q-learning, to develop agents that can learn and optimize their behavior in playing Snake. |
| ***Keywords* -** Infoman’s journal, writing guide, article template |

|  |
| --- |
| **ABSTRAK**  Permainan Snake adalah salah satu permainan klasik di mana pemain mengendalikan ular untuk memakan makanan yang muncul secara acak di layar, dengan setiap makanan yang dimakan memperpanjang panjang ular. Tantangan utama adalah menghindari tabrakan dengan dinding atau dengan tubuh ular sendiri. Menggunakan Q-learning, algoritma pembelajaran penguatan, kami mengeksplorasi cara optimal untuk mengontrol pergerakan ular dalam permainan Snake. Meskipun prinsip dasar permainan Snake sederhana, kompleksitas meningkat seiring pertumbuhan ular, membuat strategi navigasi menjadi lebih rumit. Hal ini memberikan peluang menarik untuk menerapkan algoritma pembelajaran mesin, khususnya Q-learning, untuk mengembangkan agen yang dapat belajar dan mengoptimalkan perilakunya dalam bermain Snake. |
| ***Kata Kunci* -** Infoman’s journal, writing guide, article template |

1. **Introduction**

Q-learning adalah algoritma pembelajaran penguatan (reinforcement learning) yang digunakan untuk menemukan kebijakan optimal bagi agen yang beroperasi di lingkungan yang tidak diketahui. Algoritma ini bekerja dengan memperbarui tabel nilai Q, yang memetakan pasangan negara-tindakan ke nilai (nilai Q), yang menggambarkan seberapa menguntungkan mengambil tindakan tertentu dalam keadaan tertentu. Pembaruan dilakukan berdasarkan pengalaman agen dalam bentuk reward yang diterima setelah mengambil tindakan dan mengamati hasilnya.

1. **Research Method**

Metode penelitian permainan "Snake" ini sering digunakan sebagai contoh dalam penelitian pembelajaran mesin, termasuk Q-learning, untuk mengembangkan agen yang dapat belajar bermain melalui interaksi dengan lingkungannya,berikut beberapa Q-learning dalam permainan Snake:

1.Definisi dan Tujuan Q-learning:

Q-learning adalah algoritma pembelajaran penguatan (Reinforcement Learning) yang digunakan untuk menemukan kebijakan optimal untuk agen dalam suatu lingkungan.

Agen berinteraksi dengan lingkungan, memilih tindakan berdasarkan kebijakan, dan menerima reward (penghargaan) berdasarkan tindakan tersebut. Tujuan Q-learning adalah memaksimalkan reward kumulatif.

1. Komponen Q-learning dalam Permainan Snake:

- State (S): Representasi lingkungan. Pada permainan Snake, state bisa berupa posisi ular dan makanan dalam grid, serta arah gerakan ular.

- Action (A): Tindakan yang dapat diambil agen. Dalam Snake, tindakan meliputi bergerak ke atas, bawah, kiri, atau kanan.

- Reward (R): Nilai yang diberikan setelah melakukan tindakan. Pada Snake, reward biasanya positif saat ular memakan makanan dan negatif jika ular menabrak dinding atau tubuhnya sendiri.

- Q-table: Tabel yang menyimpan nilai Q untuk setiap pasangan state-action. Nilai Q diperbarui berdasarkan pengalaman agen.

3. Proses Q-learning:

- Initialization:Q-table diinisialisasi dengan nilai acak atau nol.

- Episode: Agen memainkan beberapa episode permainan. Pada setiap langkah:

- Agen memilih tindakan (biasanya dengan epsilon-greedy policy untuk eksplorasi dan eksploitasi).

- Agen melakukan tindakan, mengamati reward, dan state baru.

- Nilai Q diperbarui menggunakan rumus:

\[

Q(s, a) = Q(s, a) + \alpha \left[ r + \gamma \max\_{a'} Q(s', a') - Q(s, a) \right]

\]

di mana \( \alpha \) adalah laju pembelajaran, \( \gamma \) adalah faktor diskon, \( r \) adalah reward, dan \( s' \) adalah state berikutnya.

4. Tantangan dalam Implementasi:

- Dimensi State yang Besar:Permainan Snake memiliki ruang state yang besar karena banyaknya kemungkinan posisi dan panjang ular.

- Sparse Rewards:Ular hanya menerima reward signifikan saat memakan makanan, menyebabkan pembaruan Q menjadi lambat.

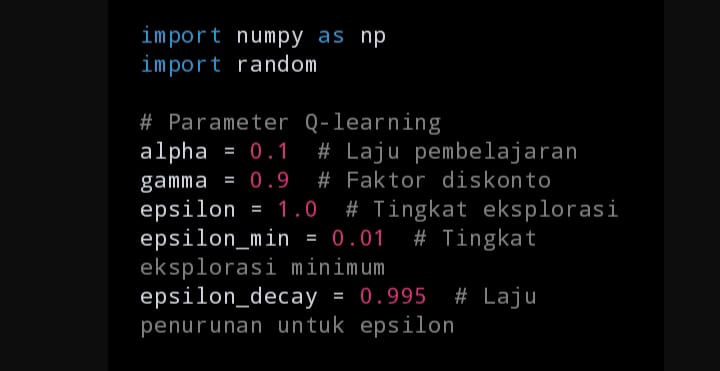
- Exploration vs. Exploitation:Menyeimbangkan eksplorasi (mencoba tindakan baru) dan eksploitasi (memanfaatkan pengetahuan yang ada) sangat penting.

5. Optimasi dan Pendekatan Lanjutan:

-Fitur Ekstraksi:Mengurangi kompleksitas state dengan mengekstrak fitur penting, seperti jarak ke makanan atau dinding terdekat.

- Deep Q-Learning (DQN): Menggunakan jaringan saraf untuk memetakan state ke nilai Q, memungkinkan generalisasi lebih baik di ruang state yang besar.

**3.Hasil dan analisis**



**Gambar 1.**

import numpy as np

import random

Parameter Q-learning

alpha = 0.1 Laju pembelajaran

gamma = 0.9 Faktor diskonto

epsilon = 1.0 Tingkat eksplorasi

epsilon\_min = 0.01 Tingkat eksplorasi minimum

epsilon\_decay = 0.995 Laju penurunan untuk epsilon

**Gambar2.**

Inisialisasi Tabel Q

Q\_table = {}

def get\_state(snake, food):

Fungsi untuk mendapatkan status saat ini (misalnya: posisi relatif makanan, posisi ular)

state = ...

return state

def choose\_action(state):

Fungsi untuk memilih aksi menggunakan kebijakan epsilon-greedy

if random.uniform(0, 1) < epsilon:

action = random.choice(['up', 'down', 'left', 'right'])

else:

action = max(Q\_table.get(state, {}), key=Q\_table.get(state, {}).get, default='up')

return action

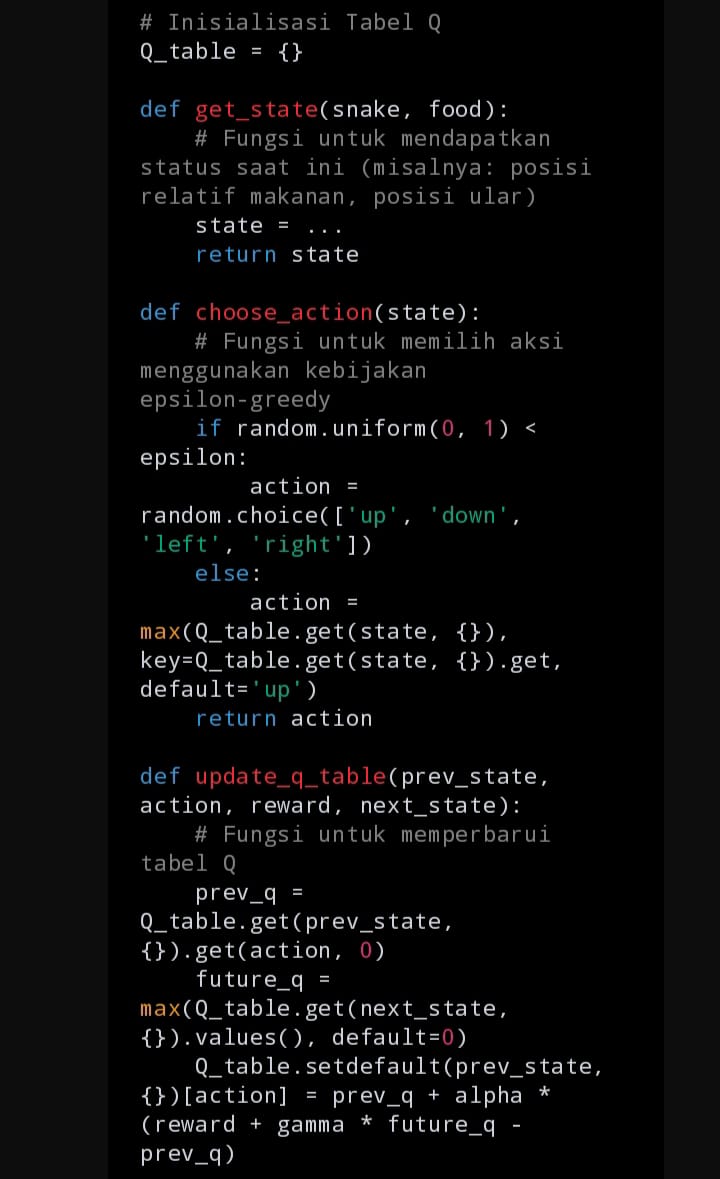
def update\_q\_table(prev\_state, action, reward, next\_state):

Fungsi untuk memperbarui tabel Q

prev\_q = Q\_table.get(prev\_state, {}).get(action, 0)

future\_q = max(Q\_table.get(next\_state, {}).values(), default=0)

Q\_table.setdefault(prev\_state, {})[action] = prev\_q + alpha \* (reward + gamma \* future\_q - prev\_q)



**Gambar 3.**

Main loop permainan

for episode in range(1000):

Inisialisasi ulang permainan

state = get\_state(snake, food)

done = False

while not done:

action = choose\_action(state)

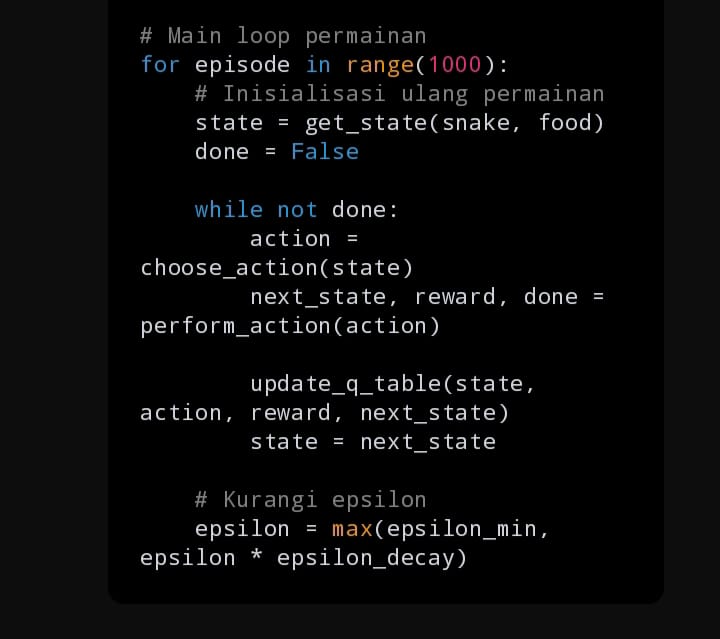
next\_state, reward, done = perform\_action(action)

update\_q\_table(state, action, reward, next\_state)

state = next\_state

Kurangi epsilon

epsilon = max(epsilon\_min, epsilon epsilon\_decay)



Penjelasan kode setiap bagian Q-learning untuk permainan snake:

Inisialisasi Parameter:

alpha = 0.1: Laju pembelajaran, menentukan seberapa cepat agen memperbarui pengetahuannya berdasarkan pengalaman baru.

gamma = 0.9: Faktor diskonto, menentukan seberapa jauh agen memperhitungkan reward masa depan.

epsilon = 1.0: Tingkat eksplorasi awal, menentukan seberapa sering agen mencoba aksi acak.

epsilon\_min = 0.01: Tingkat eksplorasi minimum, memastikan agen tidak berhenti bereksplorasi sepenuhnya.

epsilon\_decay = 0.995: Laju penurunan epsilon, mengurangi tingkat eksplorasi seiring waktu untuk beralih ke eksploitasi.

Fungsi get\_state:

Fungsi ini mengembalikan representasi status saat ini dari permainan, biasanya berupa informasi tentang posisi ular dan makanan. (state = ... menunjukkan bahwa Anda perlu melengkapi logika untuk mendapatkan status.)

Fungsi choose\_action:

Menggunakan kebijakan epsilon-greedy untuk memilih aksi. Agen memilih aksi acak dengan probabilitas epsilon (eksplorasi), atau memilih aksi dengan nilai Q tertinggi berdasarkan tabel Q saat ini (eksploitasi).

Fungsi update\_q\_table:

Memperbarui nilai Q berdasarkan formula pembaruan Q-learning. prev\_q adalah nilai Q dari status dan aksi sebelumnya, sedangkan future\_q adalah nilai Q maksimum dari status berikutnya.

- Main Loop Permainan:

- for episode in range(1000): Menjalankan simulasi untuk 1000 episode.

- state = get\_state(snake, food): Menginisialisasi ulang status permainan.

- while not done: Loop ini berjalan sampai permainan berakhir.

- action = choose\_action(state): Memilih aksi berdasarkan status saat ini.

- next\_state, reward, done = perform\_action(action): Melakukan aksi dan menerima status berikutnya, reward, dan apakah permainan selesai.

- update\_q\_table(state, action, reward, next\_state): Memperbarui tabel Q berdasarkan pengalaman.

- state = next\_state: Memperbarui status menjadi status berikutnya.

- epsilon = max(epsilon\_min, epsilon epsilon\_decay): Mengurangi nilai epsilon untuk mengurangi eksplorasi seiring waktu.

Terjemahanan

- Laju pembelajaran (alpha): Seberapa cepat agen memperbarui tabel Q dengan pengalaman baru.

- Faktor diskonto (gamma): Seberapa banyak agen menghargai reward di masa depan dibandingkan dengan reward saat ini.

- Tingkat eksplorasi (epsilon): Probabilitas bahwa agen akan memilih aksi secara acak (eksplorasi) daripada berdasarkan tabel Q (eksploitasi).

- Tingkat eksplorasi minimum (epsilon\_min): Nilai terendah untuk epsilon, mencegah agen berhenti bereksplorasi sepenuhnya.

- Laju penurunan epsilon (epsilon\_decay): Laju pengurangan epsilon, membantu transisi dari eksplorasi ke eksploitasi.

**Kesimpulan:**

Q-learning adalah algoritma pembelajaran penguatan yang membentuk agen untuk mengambil tindakan optimal melalui pengalamannya di lingkungan tertentu.Serta menggabungkan Q-learning dengan neural networks (seperti pada DQN) dapat membantu mengatasi masalah generalisasi dan ukuran state space. Metode lain seperti SARSA atau Actor-Critic bisa memberikan peningkatan dalam stabilitas pembelajaran.

Q-learning memberikan kerangka kerja yang kuat untuk mengembangkan agen yang dapat belajar bermain permainan Snake secara otonom. Dengan optimisasi yang tepat dan teknik pembelajaran lanjutan, Q-learning dapat disesuaikan untuk menangani berbagai kompleksitas permainan, dari permainan sederhana hingga lingkungan yang lebih kompleks dan dinamis. Penggunaan Q-learning dalam permainan Snake mengilustrasikan potensi pembelajaran penguatan untuk pengembangan agen AI yang adaptif dan cerdas.

**Referensi**

[1]R. Jaksa et al. “Backpropagation in Supervised and Reinforcement

Learning for Mobile Robot Control,” Proceedings of Computational

Intelligence for Modelling, Control and Automation, 1999.

[2]

B. Huang et al. “Reinforcement Learning Neural Network to the

Problem of Autonomous Mobile Robot Obstacle Avoidance,”

Proceedings of the Fourth International Conference on Machine

Learning and Cybernetics, hal. 85-89, 2005.

[3] S. Dini and M. Serrano, “Combining Q-Learning with Artificial Neural

Networks in an Adaptive Light Seeking Robot,” Swarthmore College,

CS81 Adaptive Robotics Final Projects Swarthmore College, 2012.

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction,

[5]

London, England: MIT press, 1998.

R. Rojas, Neural Networks A Systematic Introduction, Berlin, Germany:

Springer-Verlag, 1996.