ใบงานที่ 6 การสร้างปัญญาประดิษฐ์เล่นเกม Tic-Tac-Toe

วัตถุประสงค์การเรียนรู้

1. เขียนคำสั่งภาษา Python เพื่อจำลองสภาพแวดล้อม (Environment) ของเกม Tic-Tac-Toe ได้
2. เขียนคำสั่งสร้างตัวแทน (Agent) และประยุกต์ใช้อัลกอริทึม Q-Learning ได้
3. เขียนคำสั่งเพื่อคำนวณและอัปเดตค่าความรู้ (Q-Table) ตามสมการ Bellman Equation ได้
4. อธิบายการทำงานของ Epsilon-Greedy Strategy ในการตัดสินใจของ AI ได้
5. พิจารณาเปรียบเทียบประสิทธิภาพระหว่างการฝึกฝนแบบ Single Agent และ Dual Agent ได้

เครื่องมือและอุปกรณ์

1. เครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรม PyCharm IDE
2. เอกสารใบความรู้หน่วยที่ 6
3. แบบฟอร์มบันทึกผลการปฏิบัติงาน (ท้ายใบงาน)

ลำดับขั้นตอนการปฏิบัติงาน

คำชี้แจง: ให้ผู้เรียนสร้างไฟล์ main.py และเขียนโค้ดตามขั้นตอนต่อไปนี้

ขั้นตอนที่ 1: เตรียมสภาพแวดล้อมและเกม (Environment Setup)

1. Import Library: นำเข้า numpy, random, และ defaultdict จาก collections เพื่อใช้จัดการข้อมูลและตาราง
2. สร้าง Class TicTacToe:
   - สร้างเมธอด __init__: กำหนดกระดานขนาด 3x3 ด้วย np.zeros และกำหนดผู้เล่นเริ่มก่อนเป็น 1
   - สร้างเมธอด reset: สำหรับล้างกระดานเมื่อจบเกม
   - สร้างเมธอด available_actions: คืนค่า List ของช่องที่ว่างอยู่ (ที่เป็นเลข 0)
   - สร้างเมธอด make_move: รับค่า action (0-8) แปลงเป็นพิกัดแถว/คอลัมน์ด้วย divmod และลงหมากในกระดาน

- สร้างเมธอด check_winner: ตรวจสอบผู้ชนะ 8 ทิศทาง (แนวนอน, แนวตั้ง, แนวทแยง)
- สร้างเมธอด print_board แสดงผลกระดานเกม

ขั้นตอนที่ 2: สร้างสมองของ AI (The Q-Learning Agent)

1. สร้าง Class QLearningAI
2. กำหนดค่า Hyperparameters (__init__):
   - alpha (Learning Rate) = 0.1
   - gamma (Discount Factor) = 0.9
   - epsilon (Exploration Rate) = 0.2
   - สร้าง q_table ด้วย defaultdict
3. สร้างฟังก์ชัน choose_action (Epsilon-Greedy)
   - สุ่มตัวเลข: หากน้อยกว่า epsilon ให้เลือกเดินแบบสุ่ม (Exploration)
   - หากมากกว่า: ให้เลือกเดินช่องที่มีค่า Q สูงที่สุดจาก q_table (Exploitation)
4. สร้างฟังก์ชัน update (Bellman Equation):
   - เขียนสมการ: OldQ + Alpha * (Reward + Gamma * MaxNextQ - OldQ) เพื่อปรับปรุงค่าในตาราง

ขั้นตอนที่ 3: การฝึกฝนโมเดล (Training Loop)

1. สร้างฟังก์ชัน train: กำหนดค่าเริ่มต้น episodes=1000
2. เขียน Loop การฝึก:
   - วนลูปตามจำนวน episode
   - ในแต่ละรอบ ให้ AI เล่นแข่งกับตัวเอง (Self-Play) จนจบเกม
   - การให้รางวัล (Reward): ชนะได้ +1, แพ้ได้ -1, เสมอหรือเดินทั่วไปได้ 0
   - เรียกใช้ ai.update() เพื่อสอน AI ทุกครั้งที่มีการเดิน หรือจบเกม
3. Return AI: ส่งค่า AI ที่ฝึกเสร็จแล้วกลับออกมา

ขั้นตอนที่ 4: ทดสอบเล่นกับมนุษย์ (Deployment)

1. สร้างฟังก์ชัน play_against_ai:
    - รับค่า AI เข้ามา
    - วนลูปรับค่า Input จากผู้เล่น (ตำแหน่ง 1-9)
    - ให้ AI (X) ตัดสินใจเดินโดยกำหนด training=False (เอาจริง ไม่เดินมั่ว)
    - แสดงผลกระดานและประกาศผู้ชนะเมื่อจบเกม
2. เขียนส่วน main: เรียกใช้ train() และตามด้วย play_against_ai()

แบบบันทึกผลการปฏิบัติงาน

ส่วนที่ 1: ความเข้าใจในตัวแปรและสมการ (Code Understanding)

ตรงกับวัตถุประสงค์การเรียนรู้ข้อที่ 1 ถึง 4: (

ข้อ 1-2 ตรงกับวัตถุประสงค์ข้อที่ 3,

ข้อ 3 ตรงกับวัตถุประสงค์ข้อที่ 4,

ข้อ 4 และ 5 ตรงกับ วัตถุประสงค์ข้อที่ 1,

ข้อ 6 ตรงกับ วัตถุประสงค์ข้อที่ 2)

คำชี้แจง: จงอธิบายความหมายของตัวแปรในสมการ update ที่ท่านได้เขียนลงไป

1. alpha (Learning Rate) ในโค้ดนี้กำหนดค่าไว้ที่ 0.1 หมายความว่าอย่างไร ?

แนวคำตอบ: หมายถึง "อัตราการเรียนรู้" หรือความเร็วในการปรับตัวรับข้อมูลใหม่ของ AI โดยค่า 0.1 หมายความว่าในการอัปเดตตาราง Q-Table แต่ละครั้ง AI จะยอมรับประสบการณ์ใหม่ (ค่า Q ใหม่) เข้ามาแทนที่เพียง 10% และยังคงรักษาน้ำหนักของความรู้เดิมไว้ถึง 90% ซึ่งช่วยให้ AI ค่อยๆ เรียนรู้อย่างมั่นคง ไม่เปลี่ยนใจไปมาอย่างรวดเร็วเกินไปเมื่อเจอสถานการณ์ที่ต่างไปจากเดิม

2. gamma (Discount Factor) ถ้าเราปรับค่านี้เป็น 0 จะส่งผลต่อพฤติกรรมของ AI ในการวางแผนล่วงหน้าอย่างไร ?

แนวคำตอบ: ค่า Gamma เป็นตัวกำหนดว่า AI จะให้ความสำคัญกับ "รางวัลในอนาคต" มากแค่ไหน หากปรับเป็น 0 จะทำให้ AI กลายเป็นคน "สายตาสั้น (Short-sighted)" คือสนใจแค่รางวัลที่จะได้ในการเดินหมากตาถัดไป (Immediate Reward) เท่านั้น โดยจะไม่รู้จักการวางแผนระยะยาวหรือสร้างกับดักเพื่อให้ได้ชัยชนะในท้ายที่สุด

3. epsilon (Exploration Rate) ในฟังก์ชัน choose_action มีไว้เพื่อวัตถุประสงค์อะไร ?

แนวคำตอบ: เป็นตัวกำหนด "อัตราการลองผิดลองถูก (Exploration)" มีไว้เพื่อให้ AI ได้ลองสุ่มเดินในช่องที่ไม่เคยเดินมาก่อน (ในที่นี้กำหนดไว้ 0.2 หรือ 20% ของการเดิน) ซึ่งเป็นกลไกสำคัญที่ช่วยให้ AI หลุดพ้นจากการติดอยู่ในรูปแบบการเล่นซ้ำๆ เดิมๆ (Local Optimum) และช่วยให้ AI ค้นพบกลยุทธ์หรือวิธีเอาชนะใหม่ๆ ที่ดีกว่าเดิมได้

4. ในฟังก์ชัน make_move เราใช้คำสั่ง r, c = divmod(action, 3) หาก AI ตัดสินใจเลือก action = 7 คำสั่งนี้จะแปลงเป็นตำแหน่งในกระดาน (แถวและคอลัมน์) ที่เท่าไหร่?

แนวคำตอบ: แถวที่ 2, คอลัมน์ที่ 1

5. ในฟังก์ชัน check_winner เรานำทฤษฎีใดของไลบรารี NumPy มาช่วยรวบรวมเส้นทางชนะทั้ง 8 ทิศทางแทนการเขียน if-else ทีละช่อง?

แนวคำตอบ: การใช้ Array Transpose (.T) แนวตั้ง-แนวนอน และฟังก์ชัน .diagonal() แนวทแยง

6. ในคลาส QLearningAI ทำไมฟังก์ชัน state_key จึงต้องแปลงข้อมูลกระดาน (Array) ให้กลายเป็นข้อความ (String) ก่อนนำไปใช้บันทึกใน Q-Table?

แนวคำตอบ: เพราะ Dictionary ของ Python ไม่รองรับการนำ Array มาทำเป็น Key หรือชื่อหน้าสมุด จึงต้องแปลงเป็นข้อความก่อน

ส่วนที่ 2: การเปรียบเทียบวิธีการฝึกฝน (Algorithm Comparison)

ตรงกับวัตถุประสงค์การเรียนรู้ข้อที่ 5: (พิจารณาเปรียบเทียบประสิทธิภาพระหว่างการฝึกฝนแบบ Single Agent และ Dual Agent)

คำชี้แจง: ให้ผู้เรียนทดลองรัน Code ด้วยฟังก์ชัน train (แบบเก่า) และ train_new (แบบใหม่) โดยกำหนดจำนวนรอบฝึกที่ 10,000 รอบเท่ากัน แล้วบันทึกผลการแข่งขันกับ AI จำนวน 5 ตา

```
# ให้ผู้เรียนเพิ่มฟังก์ชันนี้เข้าไปใน code
def train_new(episodes=10000):
    ai_x = QLearningAI()
    ai_o = QLearningAI()
    game = TicTacToe()

    for ep in range(episodes):
        game.reset()
        st_x, act_x = None, None
        st_o, act_o = None, None

        while True:
            if game.current_player == 1:  # ตา X
                state = ai_x.state_key(game.board)
                action = ai_x.choose_action(game.board, training=True)
                if st_x: ai_x.update(st_x, act_x, 0, state)
```

```
            game.make_move(action)
            st_x, act_x = state, action

            winner = game.check_winner()
            if winner is not None:
                r_x = 1 if winner == 1 else -1 if winner == -1 else 0
                r_o = 1 if winner == -1 else -1 if winner == 1 else 0
                ai_x.update(st_x, act_x, r_x, None)
                if st_o: ai_o.update(st_o, act_o, r_o, None)
                break
        else:  # ตา O
            state = ai_o.state_key(game.board)
            action = ai_o.choose_action(game.board, training=True)
            if st_o: ai_o.update(st_o, act_o, 0, state)

            game.make_move(action)
            st_o, act_o = state, action

            winner = game.check_winner()
            if winner is not None:
                r_x = 1 if winner == 1 else -1 if winner == -1 else 0
                r_o = 1 if winner == -1 else -1 if winner == 1 else 0
                if st_x: ai_x.update(st_x, act_x, r_x, None)
                ai_o.update(st_o, act_o, r_o, None)
                break

    return ai_x
```

1. การทดลองที่ 1: ฟังก์ชัน train แบบเก่า (เล่นตัวเดียว)
    - สมมติฐาน: AI ใช้สมองก้อนเดียวเล่นเป็นทั้ง X และ O
    - กำหนด episodes = 10,000
    - ผลการแข่งขัน 5 ตา (เรา vs AI):

ชนะ ..... ครั้ง / แพ้ ..... ครั้ง / เสมอ ..... ครั้ง
(หมายเหตุ: ส่วนของจำนวนครั้งที่ ชนะ/แพ้/เสมอ จะขึ้นอยู่กับผลการเล่นจริงของผู้เรียนแต่ละคน ให้ดูที่แนวทางการบันทึกข้อสังเกตพฤติกรรม)

- สังเกตพฤติกรรม AI:
  AI อาจจะเดินหมากแบบสับสนในบางครั้ง ป้องกันการโดนรุกไม่เก่ง และบางครั้งพลาดโอกาสชนะอย่างง่ายๆ เพราะความรู้ในการบุกและการรับผสมปนเปกัน

2. การทดลองที่ 2: ฟังก์ชัน train_new แบบใหม่ (แยกสมอง 2 ตัว)
   - สมมติฐาน: แยก AI เป็น 2 ตัว (X และ O) แข่งกันเองเพื่อเอาชนะ
   - กำหนด episodes = 10,000
   - ผลการแข่งขัน 5 ตา (เรา vs AI):
     ชนะ ..... ครั้ง / แพ้ ..... ครั้ง / เสมอ ..... ครั้ง
     (หมายเหตุ: ส่วนของจำนวนครั้งที่ ชนะ/แพ้/เสมอ จะขึ้นอยู่กับผลการเล่นจริงของผู้เรียนแต่ละคน ให้ดูที่แนวทางการบันทึกข้อสังเกตพฤติกรรม)
   - สังเกตพฤติกรรม AI:
     AI เล่นได้เฉียบคมและฉลาดมากขึ้น มีการบล็อกผู้เล่น (มนุษย์) ได้อย่างรัดกุมเมื่อเรากำลังจะชนะ และรู้จักวางกับดักเพื่อสร้างโอกาสชนะแบบ 2 ทางได้ดีขึ้น (เสมอ หรือ เราแพ้)

ส่วนที่ 3: วิเคราะห์ผล (Critical Thinking)

ตรงกับวัตถุประสงค์การเรียนรู้ข้อที่ 5: (พิจารณาเปรียบเทียบประสิทธิภาพระหว่างการฝึกฝนแบบ Single Agent และ Dual Agent)

1. ทำไมเมื่อแยก AI เป็น 2 ตัว (X และ O) แข่งกันเองเพื่อเอาชนะถึงเล่นเก่งกว่า AI ใช้สมองก้อนเดียวเล่นเป็นทั้ง X และ O ทั้งที่ฝึกจำนวนรอบเท่ากัน ?

แนวคำตอบ: เพราะเกม Tic-Tac-Toe เป็นเกมที่มีเป้าหมายตรงข้ามกันอย่างสิ้นเชิง (Zero-Sum Game คือ X ชนะ = O แพ้)

การใช้ "สมองก้อนเดียว (Q-Table ตารางเดียว)" ในการเรียนรู้ทั้งการบุกของ X และการขัดขวางของ O จะทำให้เกิด "ความขัดแย้งของข้อมูล (Conflicting Rewards)" ค่าน้ำหนักในตารางจะถูกหักล้างกันเอง ทำให้ AI สับสน

แต่เมื่อแยกเป็น 2 สมอง (Dual Agents) AI แต่ละตัวจะมีเป้าหมายแยกกันชัดเจน ตัวหนึ่งเรียนรู้วิธีเอาชนะในฐานะ X อีกตัวเรียนรู้วิธีป้องกันในฐานะ O ทำให้เกิดการแข่งขันและผลักดันกันเอง (Adversarial Training) AI ทั้งสองฝั่งจึงเก่งขึ้นอย่างก้าวกระโดดและมีประสิทธิภาพสูงกว่ามากนั่นเอง