Instituto Tecnológico y de Estudios Superiores de Monterrey

ESCUELA DE INGENIERÍA Y CIENCIAS

Inteligencia Artificial Avanzada para la Ciencia de Datos II

A4. Labyrinths with Policy Gradients

Presenta:

Miguel Ángel Pérez Ávila - A01369908

Profesor:

Dr. Gerardo Jesús Camacho González

Sante Fe, Ciudad de México a 26 de Octubre del 2025

Resultados

Para la ejecución del programa se utilizaron los siguientes hiper parámetros:

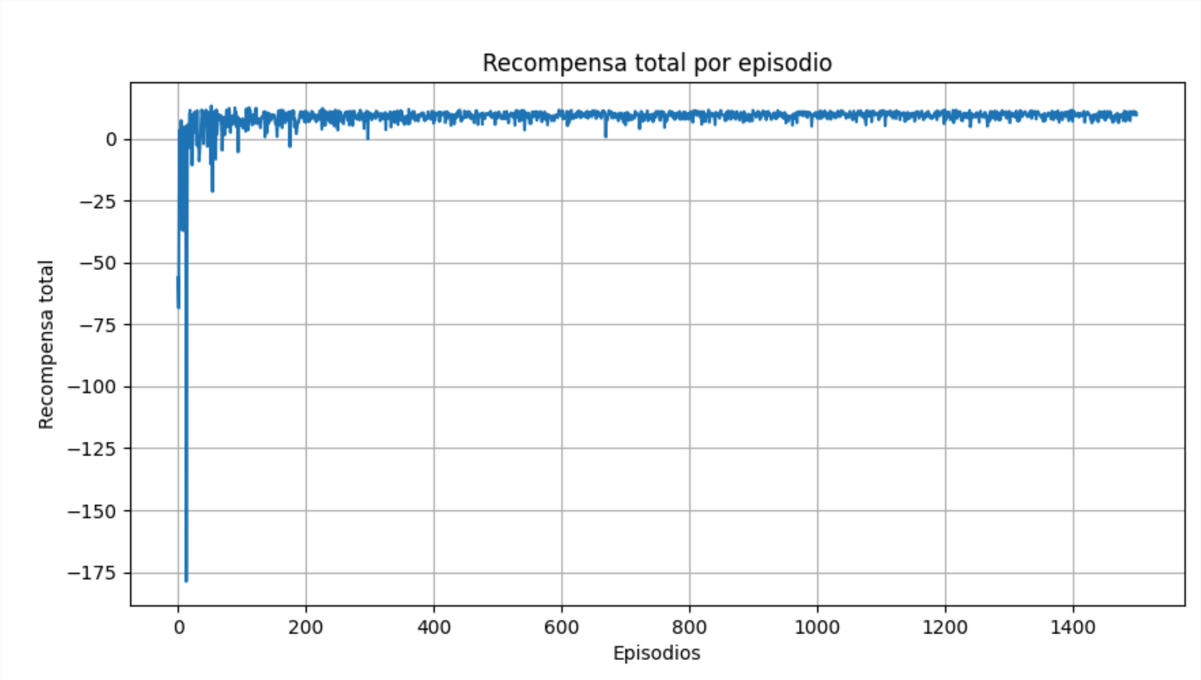
```
    # Tamaño del entorno
    SIZE = 10
    # Número de acciones posibles
    ACTIONS = 3
    # Número de episodios para el entrenamiento
    EPISODES = 1500
    # Número de pasos por episodio
    STEPS = 400
    # Learning rate
    LR = 0.05
    # Factor para el descuento
    DISCOUNT_FACTOR = 0.98
```

Y se obtuvieron los siguientes resultados del proceso de entrenamiento:

```
Entrenamiento del agente...

Episode 100/1500 -- Reward=5.07
Episode 200/1500 -- Reward=11.18
Episode 300/1500 -- Reward=8.58
Episode 400/1500 -- Reward=8.18
Episode 500/1500 -- Reward=9.69
Episode 600/1500 -- Reward=9.88
Episode 700/1500 -- Reward=10.18
Episode 800/1500 -- Reward=7.78
Episode 900/1500 -- Reward=11.48
Episode 1000/1500 -- Reward=8.78
Episode 1100/1500 -- Reward=8.98
Episode 1200/1500 -- Reward=11.67
Episode 1300/1500 -- Reward=10.99
Episode 1400/1500 -- Reward=6.97
Episode 1500/1500 -- Reward=9.59

Success rate: 1497/1500
```



Finalmente, en el entorno de prueba, se obtuvieron los siguiente resultados:

```
Test del agente entrenado...

- Test Episode 1 -- Success=True, Total Reward=9.90, Steps=18
- Test Episode 2 -- Success=True, Total Reward=6.80, Steps=29
- Test Episode 3 -- Success=True, Total Reward=10.00, Steps=13
- Test Episode 4 -- Success=True, Total Reward=10.50, Steps=15
- Test Episode 5 -- Success=True, Total Reward=10.70, Steps=17
- Test Episode 6 -- Success=True, Total Reward=10.40, Steps=17
- Test Episode 7 -- Success=True, Total Reward=6.40, Steps=31
- Test Episode 8 -- Success=True, Total Reward=11.20, Steps=13
- Test Episode 9 -- Success=True, Total Reward=9.00, Steps=18
- Test Episode 10 -- Success=True, Total Reward=9.70, Steps=19

Success rate during test: 10/10
```

Código

Implementación de Policy Gradient:

```
# Importación de librerías
from minigrid.wrappers import RGBImgObsWrapper
from minigrid_simple_env import SimpleEnv
import numpy as np
from scipy.special import softmax

# Inicializa los parámetros de la política para cada estado que tiene un
vector de pesos que define la probabilidad de cada acción
def init_params(size, n_actions):
    params = {}
    for i in range(size):
        for j in range(size):
            for d in range(4):  # Direcciones posibles
                # Inicializar los parámetros
                params[(i, j, d)] = np.zeros(n_actions)
    return params

# Selecciona una acción según los parametros actuales
def select_action(params, state):
    probs = softmax(params[state])
    action = np.random.choice(len(probs), p=probs)
    return action, probs

# Calcula las recompensas (reward-to-go) como :
# G_t = r_t + γ * r_{t+1} + γ^2 * r_{t+2} + ...
def compute_rtgo(rewards, gamma):
    returns = np.zeros_like(rewards, dtype=np.float32)
    G = 0
    for t in reversed(range(len(rewards))):
        G = rewards[t] + gamma * G
        returns[t] = G
    return returns

# Entrenamiento del agente mediante REINFORCE (Policy Gradient)
def train(env, policy, EPISODES, STEPS, LR, DISCOUNT_FACTOR):
    print("\nEntrenamiento del agente...\n")

    success_count = 0
    rewards_per_episode = []
```

```python
# Entrenamiento
for episode in range(1, EPISODES + 1):
    # Listas para almacenar trayectoria del episodio
    states, actions, rewards = [], [], []

    obs, _ = env.reset()
    terminated = False
    total_reward = 0

    for step in range(STEPS):
        # Obtener el estado actual
        pos = tuple(env.unwrapped.agent_pos)
        dir = env.unwrapped.agent_dir
        current_state = (pos[0], pos[1], dir)

        # Seleccionar acción según política
        action, probs = select_action(policy, current_state)

        # Ejecutar acción en el entorno
        obs, reward, terminated, truncated, info = env.step(action)
        reward -= 0.001  # Penalización leve por paso
        total_reward += reward

        # Guardar trayectoria
        states.append(current_state)
        actions.append(action)
        rewards.append(reward)

        if terminated or truncated:
            if terminated:
                success_count += 1
            break

    # Calcular recompensas (reward-to-go)
    rtgo = compute_rtgo(rewards, DISCOUNT_FACTOR)

    # Actualizar parámetros de la política
    for state, action, Gt in zip(states, actions, rtgo):
        probs = softmax(policy[state])
        grad_log = -probs
        grad_log[action] += 1.0
        policy[state] += LR * Gt * grad_log
```

```
        rewards_per_episode.append(total_reward)

        # Log
        if episode % 100 == 0:
            print(f"Episode {episode}/{EPISODES} --
Reward={total_reward:.2f}")

    # Mostrar tasa de éxito final
    print(f"\nSuccess rate: {success_count}/{EPISODES}")

    return policy, rewards_per_episode


# Evaluación del agente entrenado sin exploración
def test(env, policy, STEPS, SIZE, EPISODES):
    print("\nTest del agente entrenado...\n")

    # Crear entorno con renderizado visual
    env = SimpleEnv(size=SIZE, render_mode="human")
    env = RGBImgObsWrapper(env)

    success_count = 0

    # Ejecutar episodios de prueba
    for episode in range(1, EPISODES + 1):
        obs, _ = env.reset()
        terminated = False
        total_reward = 0
        steps = 0

        # Ejecutar pasos hasta que el episodio termine
        while not terminated and steps < STEPS:
            # Obtener el estado actual
            pos = tuple(env.unwrapped.agent_pos)
            dir = env.unwrapped.agent_dir
            current_state = (pos[0], pos[1], dir)

            # Elegir la acción más probable según la política
            probs = softmax(policy[current_state])
            action = np.argmax(probs)

            obs, reward, terminated, truncated, info = env.step(action)
```

```
            total_reward += reward
            steps += 1

            if terminated or truncated:
                # Si llega al objetivo o se termina el episodio cortar el
ciclo
                break

        if terminated:
            success_count += 1

        print(f"- Test Episode {episode} -- Success={terminated}, Total
Reward={total_reward:.2f}, Steps={steps}")

    # Mostrar resultados finales
    print(f"\nSuccess rate during test: {success_count}/{EPISODES}")
```

Módulo del entorno para minigrid:

```
from __future__ import annotations
from minigrid.core.grid import Grid
from minigrid.core.mission import MissionSpace
from minigrid.core.world_object import  Goal
from minigrid.minigrid_env import MiniGridEnv
import random

class SimpleEnv(MiniGridEnv):
    def __init__(
        self,
        size=19,
        max_steps: int | None = None,
        **kwargs,
    ):
        self.size = size
        self.key_positions = []
        self.lava_positions = []

        self.start_agent_pos=(1,1)

        mission_space = MissionSpace(mission_func=self._gen_mission)

        if max_steps is None:
```

```
            max_steps = 4 * size**2

        super().__init__(
            mission_space=mission_space,
            grid_size=size,
            see_through_walls=True,
            max_steps=max_steps,
            **kwargs,
        )

    @staticmethod
    def _gen_mission():
        return "Reach the goal"

    def _gen_grid(self, width, height):
        self.grid = Grid(width, height)
        self.grid.wall_rect(0, 0, width, height)

        # Place walls in straight lines
        # Vertical walls
        ##for y in range(1, height-1):
        #    self.put_obj(Wall(), width // 2, y)

        # Horizontal walls
        #for x in range(1, width-1):
        #    self.put_obj(Wall(), x, height//2)

        # Create openings in the walls
        #openings =
[(width//2,5),(width//2,15),(5,height//2),(15,height//2),]

        #for x, y in openings:
        #    self.grid.set(x, y, None)


        # Place a goal square in the bottom-right corner
        self.goal_pos = (width - 2, height - 2)
        self.put_obj(Goal(), *self.goal_pos)

        self._place_agent()

        self.mission = "Reach the goal"
```

```python
    def _place_agent(self):
        # Evitar colocar al agente cerca del objetivo
        min_distance = self.size // 2  # distancia mínima al goal

        while True:
            x = random.randint(1, self.size - 2)
            y = random.randint(1, self.size - 2)
            pos = (x, y)

            # Calcular distancia Manhattan al goal
            goal_x, goal_y = self.goal_pos
            distance = abs(goal_x - x) + abs(goal_y - y)

            # Asegurarse de que el lugar esté vacío y lejos del objetivo
            if (
                self.grid.get(*pos) is None and
                pos != self.goal_pos and
                distance >= min_distance
            ):
                self.agent_pos = pos
                self.agent_dir = random.randint(0, 3)
                break


    def reset(self, **kwargs):
        #print("resetting")
        self.stepped_floors = set()
        obs = super().reset(**kwargs)
        # self._place_agent()  # Place the agent in a new random position
        return obs

    def step(self, action):
        prev_pos=self.agent_pos
        prev_dir=self.agent_dir
        obs, reward, terminated, truncated, info = super().step(action)

        SIZE = self.size-2

        reward = -0.2  # base penalty

        if self.agent_pos[0] > SIZE//2 and self.agent_pos[1] > SIZE//2:
            reward += 0.3  # incentivo por acercarse al goal
```

```python
        if prev_dir == self.agent_dir and prev_pos == self.agent_pos:
            reward -= 0.3  # castigo por chocar

        if isinstance(self.grid.get(*self.agent_pos), Goal):
            reward = 10
            terminated = True

        return obs, reward, terminated, truncated, info
```