Reinforcement Learning

- Jorge Caballeros Pérez - 20009
- Mario de León - 19019
- Pablo Escobar - 20936

Task 1

¿Qué es Programación Dinámica y cómo se relaciona con RL?

Una técnica para resolver problemas complejos dividiéndolos en subproblemas más simples y guardando sus soluciones para evitar recalcularlas. Se usa para encontrar políticas óptimas en problemas de decisión secuencial, aprovechando el conocimiento del modelo del entorno (transiciones y recompensas)

Explique en sus propias palabras el algoritmo de Iteración de Póliza.

El algoritmo alterna entre dos fases:

- Evaluación de la póliza: calcula el valor de cada estado bajo una póliza dada
- Mejora de la póliza: ajusta la póliza para elegir acciones que maximicen el valor calculado

Se repite hasta que la póliza se vuelve óptima

Explique en sus propias palabras el algoritmo de Iteración de Valor

El algoritmo actualiza directamente los valores de los estados:

- Actualización de valor: calcula el valor máximo esperado para cada estado tomando en cuenta todas las acciones posibles
- Extracción de la póliza: calcula la póliza óptima eligiendo la mejor acción basada en los valores actualizados

Se repite hasta que los valores convergen

En el laboratorio pasado, vimos que el valor de los premios obtenidos se mantienen constantes, ¿por qué?

Los premios se mantienen constantes para simplificar el análisis y la implementación de los algoritmos. Así hay una convergencia predecible y eficiente al buscar una poliza óptima

Task 2

El objetivo principal de este ejercicio es que simule un MDP que represente un robot que navega por un laberinto de cuadrículas de 3x3 y evalúe una política determinada. Por ello considere, a un robot navega por un laberinto de cuadrícula de 3x3. El robot puede moverse en cuatro direcciones: arriba, abajo, izquierda y derecha. El objetivo es navegar desde la posición inicial

hasta la posición de meta evitando obstáculos. El robot recibe una recompensa cuando alcanza la meta y una penalización si choca con un obstáculo. El laberinto es el siguiente:

- [S][.][X]
- [.][X][.]
- [.][.][G]

Donde:

- S = punto de inicio
- G = punto de meta
- X = son obstáculos

```
import numpy as np
import random

# Definición del robot
class Robot:
    def __init__(self, position: int, score: int = 0):
        self.position = position
        self.score = score

    def move(self, direction: str):
        transitions = {
            'arriba': -3,
            'abajo': 3,
            'izquierda': -1,
            'derecha': 1
        }
        if direction in transitions:
            self.position += transitions[direction]
            self.position = max(0, min(8, self.position))

# Definición del tablero
class Board:
    def __init__(self, layout: int = 3):
        self.layout = layout
        self.states = [[' ' for _ in range(self.layout)] for _ in
range(self.layout)]

    def setObstacles(self, xPos, yPos):
        self.states[xPos][yPos] = 'X'

    def setPrize(self, xPos, yPos):
        self.states[xPos][yPos] = 'G'

    def setPlayer(self, xPos, yPos):
        self.states[xPos][yPos] = 'S'

    def printBoard(self):
        for row in self.states:
```

```
            for element in row:
                print(f'[ {element} ]', end=" ")
            print()

# Configurar tablero
board = Board(3)
board.setPlayer(0, 0)    # S en (0, 0)
board.setObstacles(0, 2) # X en (0, 2)
board.setObstacles(1, 1) # X en (1, 1)
board.setPrize(2, 2)     # G en (2, 2)

# Imprimir el tablero
board.printBoard()
```

```
[ S ] [   ] [ X ]
[   ] [ X ] [   ]
[   ] [   ] [ G ]
```

```
# Definición del MDP
states = list(range(9))
actions = ['arriba', 'abajo', 'izquierda', 'derecha']

# Transiciones
def getTransitions(state):
    transitions = {
        'arriba': state - 3 if state >= 3 else state,
        'abajo': state + 3 if state < 6 else state,
        'izquierda': state - 1 if state % 3 != 0 else state,
        'derecha': state + 1 if state % 3 != 2 else state
    }
    return transitions

transitions = {state: getTransitions(state) for state in states}

# Recompensas
rewards = {state: {a: -1 for a in actions} for state in states}
obstacles = [2, 4]
goalState = 8

for state in obstacles:
    rewards[state] = {a: -100 for a in actions}

rewards[goalState] = {a: 10 for a in actions}

for state in obstacles:
    for action in actions:
        transitions[state][action] = state

# Inicialización de la función de valor
v = {state: 0 for state in states}
```

```
# Parámetros del algoritmo
gamma = 0.9
threshold = 0.001

# Algoritmo de iteración de valor
def valueIteration(states, actions, transitions, rewards, v, gamma,
threshold):
    while True:
        delta = 0
        for s in states:
            v_old = v[s]
            v[s] = max([rewards[s][a] + gamma * v[transitions[s][a]]
for a in actions])
            delta = max(delta, abs(v_old - v[s]))
        if delta < threshold:
            break
    return v

# Extraer la política óptima
def extractPolicy(states, actions, transitions, rewards, v, gamma):
    policy = {}
    for s in states:
        actionValues = {}
        for a in actions:
            actionValues[a] = rewards[s][a] + gamma * v[transitions[s]
[a]]
        policy[s] = max(actionValues, key=actionValues.get)
    return policy

# Algoritmo de iteración de políticas
def policyIteration(states, actions, transitions, rewards, gamma,
threshold):
    # Inicializar una política aleatoria
    policy = {s: random.choice(actions) for s in states}

    def policyEvaluation(policy, states, transitions, rewards, v,
gamma, threshold):
        while True:
            delta = 0
            for s in states:
                v_old = v[s]
                a = policy[s]
                v[s] = rewards[s][a] + gamma * v[transitions[s][a]]
                delta = max(delta, abs(v_old - v[s]))
            if delta < threshold:
                break
        return v

    while True:
        v = {state: 0 for state in states}
```

```
        v = policyEvaluation(policy, states, transitions, rewards, v,
gamma, threshold)
        policyStable = True
        for s in states:
            oldAction = policy[s]
            actionValues = {}
            for a in actions:
                actionValues[a] = rewards[s][a] + gamma *
v[transitions[s][a]]
            bestAction = max(actionValues, key=actionValues.get)
            policy[s] = bestAction
            if oldAction != bestAction:
                policyStable = False
        if policyStable:
            break
    return policy, v

# Ejecutar iteración de valor
v = valueIteration(states, actions, transitions, rewards, v, gamma,
threshold)

# Extraer política óptima de la iteración de valor
policyValueIteration = extractPolicy(states, actions, transitions,
rewards, v, gamma)

# Ejecutar iteración de políticas
policyPolicyIteration, vPolicyIteration = policyIteration(states,
actions, transitions, rewards, gamma, threshold)

print("Función de Valor (Iteración de Valor):")
for state in v:
    print(f"Estado {state}: {v[state]}")
```

```
Función de Valor (Iteración de Valor):
Estado 0: 62.170166475158226
Estado 1: 54.953149827642406
Estado 2: -999.9916647515823
Estado 3: 70.1891664751582
Estado 4: -999.9916647515823
Estado 5: 88.99916647515822
Estado 6: 79.09916647515821
Estado 7: 88.99916647515822
Estado 8: 99.99916647515822
```

```
print("\nPolítica Óptima (Iteración de Valor):")
for state in policyValueIteration:
    print(f"Estado {state}: {policyValueIteration[state]}")
```

```
Política Óptima (Iteración de Valor):
Estado 0: abajo
```

```
Estado 1: izquierda
Estado 2: arriba
Estado 3: abajo
Estado 4: arriba
Estado 5: abajo
Estado 6: derecha
Estado 7: derecha
Estado 8: abajo
```

```
print("\nFunción de Valor (Iteración de Políticas):")
for state in vPolicyIteration:
    print(f"Estado {state}: {vPolicyIteration[state]}")
```

```
Función de Valor (Iteración de Políticas):
Estado 0: 62.170166475158226
Estado 1: 54.953149827642406
Estado 2: -999.9916647515823
Estado 3: 70.1891664751582
Estado 4: -999.9916647515823
Estado 5: 88.99916647515822
Estado 6: 79.09916647515821
Estado 7: 88.99916647515822
Estado 8: 99.99916647515822
```

```
print("\nPolítica Óptima (Iteración de Políticas):")
for state in policyPolicyIteration:
    print(f"Estado {state}: {policyPolicyIteration[state]}")
```

```
Política Óptima (Iteración de Políticas):
Estado 0: abajo
Estado 1: izquierda
Estado 2: arriba
Estado 3: abajo
Estado 4: arriba
Estado 5: abajo
Estado 6: derecha
Estado 7: derecha
Estado 8: abajo
```